# The Physics of Persistent AI Worlds

## From Token Prediction to World Evolution

*Persistent Generative Worlds: A Field Theory of Semantic Simulation*

Flyxion

Independent Researcher

2026

# Preface

This book is an extended argument for a single claim: that a system generating observations should be answerable to something that persists independent of being observed, rather than merely conditioned on whatever happens to be near at hand. It develops that claim from a first symptom, a stone wall whose crack moves between two glances, through a complete mathematical and engineering architecture, into a set of applications and, finally, into what the architecture implies about identity, minds, and civilization once taken seriously as more than an engineering convenience.

## Roadmap of the Theory

This book develops each major concept through four complementary perspectives. It begins with intuition, introducing why the concept is needed. It then develops the architectural role the concept plays within the semantic field. Once the architecture is established, the mathematical structures required to formalize it are introduced. Finally, the engineering implications are explored through concrete implementations. Not every chapter contains all four perspectives; instead, the book revisits each concept as progressively richer tools become available.

The book's core cycle recurs throughout, refined further each time it reappears:

$$\text{World} \to \text{Persistent Field } (M_t) \to \text{Observation } (O_\theta) \to \text{Cues} \to \text{Affordances} \to \text{Neural Proposal } (\Delta\psi) \to \text{Constraint Projection } (\Pi_C) \to \text{Commit} \to \text{Updated World } (M_{t+1}) \to \text{World}$$

| | |
|---|---|
| Part I | Why the cycle is needed |
| Part II | The world and observation |
| Part III | Action and evolution |
| Part IV | Mathematical guarantees |
| Part V | Software realization |
| Part VI | Applications |
| Part VII | Philosophical consequences |

# Contents

# Part I

# Why AI Forgets

# Chapter 1

# The Texture Problem

## 1.1 A Wall That Changes Every Time You Look At It

Players of AI-generated games have a name for a particular kind of small, persistent disappointment. You turn a corner, glance at a stone wall, keep moving, and a moment later, for whatever reason, look back. The wall is still there, still recognizably a wall, still built from the same kind of stone. But the moss has moved. The crack that ran diagonally through the third course of blocks is gone, or has migrated somewhere else entirely, or has been joined by a second crack that was never there a moment ago. Nothing about this is a rendering glitch in the ordinary sense; every individual frame looks fine. The problem only shows up in the difference between frames, in the fact that the system generating the wall never actually committed to a specific wall in the first place. It committed to a style of wall, and produced a fresh, independent, individually convincing instance of that style each time it was asked.

This is not a hypothetical concern raised for the sake of argument. It is a routine, observed behavior of the genera-

tive systems currently used to build interactive AI worlds, and it is worth taking seriously exactly because each individual output is so rarely at fault. A single rendering of the wall, examined on its own, usually looks correct, coherent, and plausible. The failure is invisible at the level of any one frame and only becomes visible in the relationship between frames, which is precisely the level at which most evaluation of generative systems does not look.

## 1.2 Rendering Versus Remembering

It is worth being precise about what has actually gone wrong, because the natural first description, that the system produced two different walls, understates the problem. The system did not produce two different walls. It never produced a wall, in the sense of a specific, individuated, continuing thing, at all. What it produced, twice, was a rendering: a plausible visual instance of "stone wall," generated fresh from a learned distribution over what stone walls tend to look like, with no reference to anything that happened the first time this particular prompt, or this particular scene, was rendered before.

Rendering, in this sense, and remembering are different operations, and the difference is not a matter of degree. Rendering asks a model what a stone wall plausibly looks like, and the model answers from its training distribution, drawing a fresh sample each time it is asked. Remembering asks what this stone wall, the specific one that has al-

ready been established, looks like, and answers by consulting a record of what was previously committed rather than by sampling anew. A system built entirely from rendering can be extraordinarily good at its actual job, producing individually beautiful, physically plausible, stylistically consistent images, and still have no mechanism whatsoever for remembering, because nothing about producing a good sample from a distribution requires consulting what any previous sample from that distribution happened to be.

## 1.3 A Problem Bigger Than Graphics

It would be a mistake to file this under graphics or rendering quality, because the same failure shows up wherever a generative system is asked to maintain something across more than one act of generation, regardless of modality. A video model asked to continue a shot will often drift, subtly reinterpreting a character's face, a room's layout, or an object's color across a cut it was supposed to be maintaining continuity through, for exactly the reason the wall drifts: nothing forces frame two to be a continuation of frame one rather than an independent, merely plausible successor to it. A language model maintaining a long conversation will, past a certain length, quietly contradict a detail it introduced many turns earlier, not because it has access to contradicting information, but because the detail was never stored anywhere retrievable; it was generated once, used briefly, and then effectively forgotten the moment it left the portion of

context the model was actively attending to. A game character built from a language model will cheerfully claim, on the tenth conversation, to be meeting the player for the first time, having ”forgotten” nine previous conversations not through any failure of memory in the ordinary sense but because nothing about how the character speaks was ever connected to a persistent record of what it had previously said.

These are not three unrelated bugs occurring in three different systems. They are the same architectural gap, showing up in whatever modality happens to be under discussion: a system built to render plausible outputs, asked to also maintain a persistent identity across more than one output, and given no mechanism to do so beyond hoping that plausibility and consistency happen to coincide.

## 1.4 Why More Scale Does Not Fix This

It is tempting to treat this as a problem scale will eventually solve: a larger model, trained on more data, conditioned on a longer context window, will surely learn to be more consistent, the way larger models have learned to be more fluent, more knowledgeable, more capable along nearly every other axis that has been thrown at them. This chapter will not attempt to prove that scale cannot help; it plainly can, and a sufficiently long context window genuinely does reduce how often a model contradicts something it said recently enough to still be attending to. But reduction is not

elimination, and the reason is architectural rather than a matter of degree. A context window, however long, is still a buffer of recent inputs the model conditions on, not a persistent record the model is obligated to remain faithful to. Nothing about attending to more tokens changes the underlying operation from rendering into remembering; it only enlarges the window within which rendering happens to look like remembering, because enough of the relevant detail is still nearby enough to influence the next plausible sample. Push the wall far enough outside that window, long enough ago in the conversation, far enough back in the game session, and the drift this chapter opened with returns, precisely because it was never actually solved, only outrun for as long as the buffer lasted. This book will return to this argument in more detail in the chapters immediately following this one; for now it is enough to notice that the problem this chapter has described survives every scale of system that has been built to render rather than to remember, and shows no sign, in principle, of a scale at which it would stop.

## 1.5 What a Real Solution Would Need

It is worth naming, in outline and without yet attempting to build it, what a system would actually need in order to avoid this failure, because the shape of the requirement turns out to matter more than any specific technique for satisfying it. Such a system would need something that continues to ex-

ist between acts of generation, independent of whether anything is currently observing it, so that there is something for a second observation to be checked against rather than merely a second independent sample. It would need to treat observation itself differently: not as an occasion to produce a fresh, plausible output, but as an occasion to read from whatever has persisted, producing new content only where genuinely nothing has been established yet. And it would need this discipline to hold not as a matter of the model's trained preferences, which can always be overridden by a sufficiently unusual prompt or a sufficiently long gap, but as a matter of the system's own architecture, in the same sense that a database's consistency does not depend on the application querying it happening to ask politely.

This book is an extended answer to what a system built this way would actually look like: what it would mean for something to persist, how observation could be redefined as reading rather than inventing, and what would have to be true of a system's internals for this discipline to be architectural rather than aspirational. None of that begins in this chapter. This chapter's task was only to make the problem impossible to unsee, in enough different guises, that a reader arrives at the rest of the book already convinced that the problem is real, structural, and not obviously solved by anything already on the table.

## 1.6 Looking Ahead

The wall, the video cut, the forgetful character are all symptoms of a single underlying fact about how current systems are built: they have no persistent internal state at all, only a context they condition on and a policy for producing the next plausible output given that context. Chapter 2 examines this fact directly, asking what, if anything, current architectures do have in place of a persistent world, and why what they have, however sophisticated, still falls short of the thing this book is going to insist on calling a world.

# Chapter 2

# The World That Doesn't Exist

## 2.1 What Models Actually Have, Instead of a World

Chapter 1 described a symptom without asking what, if anything, current systems have in place of the thing that would prevent it. It is worth asking directly, because the honest answer is not nothing. A large language model carries a context window, the recent span of tokens it conditions its next output on, and internally, a set of hidden states or key-value caches summarizing that span in a form suited to fast lookup during generation. A diffusion model carries a noise trajectory and whatever conditioning signal, text, image, or prior frame, it was given for the current generation. A video model carries some window of recent frames it attends to when producing the next one. None of these are nothing. Each is a real, technically sophisticated mechanism, and each does genuinely help a system behave more consistently than a system with no such mechanism at all.

None of them, however, is a world, and this chapter is about the specific way in which they fall short.

## 2.2 The Buffer Is Not the World

A context window, a hidden state, a cache of recent frames: each of these shares a property worth naming plainly. Each is a buffer. A buffer holds what is currently near at hand, weighted toward recency, and it does not distinguish, in any principled way, between two things that a genuine persistent record would need to keep entirely separate: something that has not happened yet, and something that happened and has since fallen out of the buffer's reach. From inside the buffer's own operation, both cases look identical. Nothing is there. A detail established forty thousand tokens ago and a detail that was simply never mentioned occupy exactly the same status once the conversation has moved on far enough, which is to say no status at all, and a system reasoning only from its buffer cannot tell the two apart because the information needed to tell them apart was never retained anywhere the buffer could still consult it.

This is the wall problem restated in the vocabulary of current architectures rather than the vocabulary of a game engine. The crack that was invented in the first rendering and absent in the second was not maliciously discarded. It fell out of whatever buffer, however defined, the second rendering was conditioned on, and from inside that second rendering's process, there was no way to know a crack had ever been committed to in the first place. A longer buffer pushes this failure further away in time or in scene-distance. It does not remove the failure, because a buffer, by its nature, is fi-

nite, and anything a genuine world would need to remember indefinitely will eventually exceed whatever finite span the buffer is willing to hold.

## 2.3 Object Permanence, Borrowed and Misapplied

Developmental psychology has a name for the achievement current architectures lack, and it is worth borrowing carefully, with an explicit caveat about the borrowing. Human infants, for roughly the first eight to twelve months of life, behave as though an object that has left their field of view has ceased to exist: they do not search for a toy hidden under a cloth in front of them, not because they lack the physical capability to lift the cloth, but because nothing in their current cognitive apparatus represents the toy as continuing to exist somewhere unseen. Object permanence is the developmental achievement of coming to represent objects as persisting independent of being perceived, and its acquisition is one of the most robustly documented milestones in early cognitive development.

The caveat matters more than the analogy. This book is not claiming that language models or diffusion models have minds, that they lack object permanence in the sense an infant lacks it, or that training them longer would constitute development in any psychological sense. The analogy is architectural, not psychological: a system that has no mechanism for representing an unobserved region as con-

tinuing to hold specific, determinate content is behaviorally indistinguishable, with respect to that content, from an infant who has not yet acquired object permanence, whatever the underlying explanation for the resemblance turns out to be. Every generation, for a system built entirely from buffers, restarts this developmental stage from zero. There is no accumulation across generations of the kind object permanence, once acquired, provides across a human life. The wall, the crack, the previously established fact about a conversation: each is, from the system's own point of view at the moment of the next generation, an object that has left the field of view and been treated, structurally, as though it no longer exists anywhere at all.

## 2.4 Why This Cannot Be Trained Away

It is tempting to treat this as a gap that sufficiently targeted training could close: reward consistency, penalize contradiction, and the system should eventually learn to behave as though objects persisted, whether or not anything internal to it actually represents them as persisting. This section argues that training of this kind, however successful within its own terms, addresses the wrong category of problem.

There is a difference between behavior and architecture that this book will insist on throughout. Behavior is a statistical tendency: what a system, on average, across many trials, tends to do. Architecture is a structural fact: what a system is or is not capable of doing, by construction, re-

gardless of how any particular trial happens to go. Training a system to behave more consistently makes contradiction statistically rarer. It does not make contradiction architecturally impossible, and the difference is not merely academic. A system trained to behave consistently will still, on some sufficiently unusual input, sufficiently long context, or sufficiently rare combination of circumstances its training did not adequately cover, produce the very failure the training was meant to suppress, because nothing about the training changed what the system is capable of; it only changed what the system tends to do within the range of situations the training actually sampled. Object permanence, in the architectural sense this chapter needs, cannot be a tendency. It has to be a guarantee, and a guarantee has to come from what the system is built to do, not from what it has merely been encouraged to do often enough.

## 2.5 What Hidden State Was Actually Built For

It is worth being specific about why hidden states and key-value caches, despite being genuinely sophisticated pieces of engineering, cannot supply this guarantee even in principle, because the reason is not that they are poorly implemented. It is that they were built to solve a different problem. A hidden state in a language model is optimized, through training, to be useful for predicting the next token given everything seen so far. This is an extremely demand-

ing objective, and the representations that emerge from it are correspondingly rich. But usefulness for next-token prediction is not the same property as faithfulness to a specific, checkable, persistent fact. A hidden state can discard information a persistent record would need to keep, provided that information is no longer helpful for predicting what comes next, and a hidden state can blend or approximate information a persistent record would need to keep exact, provided the blend remains statistically useful. Nothing about the training objective ever asked the hidden state to answer the question a genuine memory would need to answer on demand: was this specific fact, this specific detail, already established, and if so, exactly what was established. The hidden state was never built to be queried this way. It was built to be helpful for the next word, and helpfulness for the next word is compatible with silently losing, blending, or reinterpreting exactly the details a persistent world would need to hold fixed.

## 2.6 Looking Ahead

This chapter has argued that current architectures lack persistence not because their buffers are too short, but because a buffer, however long, is the wrong kind of object to supply what persistence requires, and that this gap cannot be closed by training alone, because training changes behavior, not architecture. It is worth anticipating an objection before the next chapter takes it up directly: some systems already

pair a generative model with an explicit external memory, a retrieval index, a vector database, a scratchpad the model can write to and read from. Surely this closes the gap this chapter has described. Chapter 3 examines that possibility directly, and argues that having a memory module is not the same as having a memory that actually functions the way the wall, and everything this book will go on to build, requires.

# Chapter 3

# Memory Is Not Context

## 3.1 The Obvious Objection

Chapter 2 closed by anticipating a reasonable response to everything it had argued. If the problem is that hidden states and context windows are buffers, finite, recency-weighted, unable to distinguish an unmentioned fact from a forgotten one, the fix seems immediate: give the system an explicit memory. Many production systems already do exactly this, pairing a generative model with a vector database or retrieval index, embedding text into a searchable store, and, at generation time, retrieving whatever stored content is most relevant to the current query and feeding it back into context. This is not a hypothetical proposal. It is standard practice, widely deployed, and it does measurably help with exactly the kind of long-range consistency Chapter 2 was concerned about. It is worth taking seriously, and worth examining closely enough to see precisely where it still falls short.

## 3.2 What Retrieval Actually Retrieves

The critical question is not whether retrieval helps, which it plainly does, but what kind of object retrieval is actually built to store and return. A retrieval system, in the form used by nearly every production system today, stores documents: chunks of text, embedded into a vector space, indexed for similarity search. At generation time, a query is embedded the same way, compared against the stored chunks, and the most similar ones are returned and fed into context alongside whatever the model is currently generating. This is a genuinely useful mechanism, and it is worth being precise about what it is a mechanism for. It retrieves what was written. It does not represent what currently is. A document, once written, is a static record of a claim made at some point in the past, and a retrieval system's job is to find documents relevant to a query, not to maintain a single, current, authoritative account of anything. These are different jobs, and the difference is exactly where this chapter's argument lives.

## 3.3 The Wall, Given a Memory Module

It helps to work through a concrete case. Suppose the system generating the courtyard's wall is given a vector database, and suppose, generously, that the first rendering does the right thing: it writes a note, "the wall has a diagonal crack running through the third course of stones,"

into the store. The second rendering, prompted to depict the same wall, embeds its own query and retrieves the note, assuming the similarity search works as intended, and conditions its generation on what the note says. This can work, and often does, well enough of the time to seem like a solution.

It is worth examining the three places this can fail, because each failure is not an implementation bug to be patched but a structural consequence of what a document store is. First, nothing about this architecture guarantees the note gets written in the first place. Writing to the memory store is itself just another generation-time decision the model may or may not make, exactly as unreliably as any other behavior Chapter 2 already argued cannot be guaranteed by training alone; a model that usually remembers to write down important details will still, on some occasions, simply generate a plausible wall and move on without writing anything at all. Second, retrieval is approximate. Similarity search returns what is most relevant according to a learned embedding space, not what is guaranteed to be the single correct prior fact, and a differently worded second query, a crowded store with many superficially similar notes, or an embedding space that simply does not capture the relevant distinction can all cause the right note to be missed, or the wrong one retrieved instead. Third, and most consequentially, nothing in this architecture reconciles conflicting notes. If the wall is later damaged, and the system, quite reasonably, writes a new note, "the crack has widened

and a second, shorter crack has appeared beside it," both notes now sit in the store indefinitely. Nothing forces the old note to be revised, retired, or marked as superseded. A future query might retrieve the old note, the new one, both, or neither, and the store itself has no concept of which one is currently true, because a document store was never built to hold a concept of currently true at all. It was built to hold a growing collection of things that were, at some point, written.

## 3.4 Document-Based Versus World-Based

This generalizes into the distinction this chapter exists to draw. A document-based system answers the question what has been written about this. It is, in the most literal sense, an archive: a passive collection of records, retrieved by relevance, with no obligation that its contents be current, consistent with one another, or complete. A world-based system would need to answer a different question entirely: what is currently true here, as a single, authoritative, internally consistent answer, not a ranked list of things that have, at various points, been claimed. Retrieval augments a buffer with a larger, slower buffer, searchable by similarity rather than limited by recency, and this is a genuine improvement over Chapter 2's context window alone. It is not the architectural shift Chapter 1 argued the wall actually requires, which was never about the size or searchability of what gets consulted, but about whether consultation happens auto-

matically, whether writing happens reliably, and whether what is stored is guaranteed, by construction, to remain exactly one thing rather than an accumulating pile of things that were once said.

## 3.5 Three Failures No Amount of Retrieval Engineering Fixes

It is worth naming these three failures plainly, because each one is a design consequence of the document-store approach rather than a defect any particular implementation happens to have. Writing is optional: nothing about the architecture obligates a fact to be committed the moment it is established, only permits it, and permission is not the same guarantee Chapter 2 already argued behavior alone cannot supply. Retrieval is approximate: even a fact that was reliably written can be missed or misretrieved, because similarity search was never designed to guarantee retrieval of the single correct record, only to surface plausibly relevant ones. And reconciliation is absent: nothing about storing documents forces contradictory documents to be resolved into one current state, so a store can, and eventually will, accumulate claims that cannot all be true at once, with no mechanism deciding which one currently governs. A larger vector database, a better embedding model, a more sophisticated retrieval ranking, none of these three failures are addressed by any amount of engineering improvement within the document-store paradigm, because all three are proper-

ties of what a document store is, not properties of how well any particular document store has been built.

## 3.6 What Would Actually Be Needed

This sharpens the requirement Chapter 1 stated only in outline. What a persistent world needs is not memory as an optional resource a model might think to consult, retrieved when a similarity search happens to surface it, but memory as an unconditional substrate: something every observation is automatically checked against, not merely permitted to consult, and something every legitimate change is automatically written into, not merely allowed to be written into if the model happens to generate the right auxiliary action. The distinction is not one of degree, a bigger and better retrieval system eventually closing the gap through sheer engineering effort. It is a distinction in kind, between a system where reading and writing are optional behaviors available to a generative process and a system where reading and writing are the generative process's own architecture, no more optional than the buffer Chapter 2 already showed cannot be dispensed with.

## 3.7 Looking Ahead

This chapter and the one before it have both concerned themselves with where information is kept: a recency-weighted buffer, or a searchable but unreconciled store of

documents, and why neither supplies what a persistent world actually requires. Neither chapter has yet asked a more fundamental question, one that survives even a perfect answer to the storage problem. The generative principle underlying nearly every system this book has discussed so far is next-token prediction: producing, at each step, whatever continuation is statistically most plausible given what came before. Chapter 4 asks whether this principle, whatever it is given to store or retrieve, is even the right kind of operation to be building a persistent world out of in the first place.

# Chapter 4

# The Limits of Next-Token Prediction

## 4.1 Granting Everything and Still Falling Short

Suppose, for the sake of argument, that Chapter 3's problem is entirely solved. Suppose the system generating the courtyard has perfect memory: every fact, once established, is guaranteed to be written; every relevant fact, once written, is guaranteed to be retrieved when needed; every contradiction is guaranteed to be reconciled the moment it would otherwise arise. This is a considerable concession, well beyond what any deployed retrieval system currently offers, and it is worth granting it fully before asking the question this chapter exists to ask. Given all of this, given perfect access to everything that has ever been established about the wall, does the system that generates the wall's next appearance actually produce a persistent world?

The honest answer is not automatically. Perfect information, available to a generative process, does not by itself guarantee that the process uses that information as a binding constraint rather than as one more influence, however strong, on a fundamentally different kind of computation.

This chapter argues that the generative principle underlying nearly every system this book has discussed, next-token prediction and its close relatives across other modalities, is built to answer a different question than the one persistence actually requires, and that this mismatch survives even a perfect solution to the storage problem Chapters 2 and 3 addressed.

## 4.2 Two Kinds of "What Comes Next"

It is worth distinguishing two computations that are easy to conflate because they frequently produce the same answer. Statistical continuation asks: given everything currently in context, what is the most probable next token, frame, or region of an image, according to a distribution learned from training data. Causal continuation asks a different question: given the actual, specific prior state of a system, what must or should follow, according to whatever rules actually govern that system's evolution. A physics engine computes causal continuation: given a ball's exact position and velocity, its next position is determined, not merely likely, by the equations governing motion. A video model trained on footage of falling balls computes statistical continuation: given the frames so far, what continuation looks like a plausible falling ball, according to patterns extracted from many other falling balls it was trained on. These are different computations, arrived at through entirely different means, and it is only a contingent, empirical fact when they happen to

agree.

## 4.3 Why the Two Usually Agree, and Why the Agreement Is Misleading

In practice, statistical and causal continuation agree often enough that the distinction can seem like a philosopher's nicety rather than a live engineering concern. A model trained on enough footage of falling balls learns a distribution that closely tracks actual physics, not because it has represented Newton's laws anywhere internally, but because physically accurate continuations are, unsurprisingly, the statistically common ones in a large enough corpus of real footage. This agreement is real, and it is also entirely contingent on training data densely covering the situation actually being generated. It breaks down precisely where this book has been focused from Chapter 1 onward: at the specific, arbitrary, already-established facts that a persistent world needs to honor, and that no amount of generic plausibility can substitute for. The wall's crack sits in a specific location because it was established there, once, by a specific prior act of invention, not because that specific location is any more statistically probable than any of the countless other locations a crack could plausibly appear. Statistical continuation has no privileged reason to reproduce that specific location. Only causal continuation, actually reading off the established fact rather than sampling from a general distribution over plausible cracks, gets it right reliably rather than

by coincidence.

## 4.4 Retrieved Memory Does Not Make Prediction Causal

This is where Chapter 3's concession, granted in full at the start of this chapter, turns out not to be enough, and the reason is worth stating precisely because it is easy to miss. Even with the crack's exact position perfectly retrieved and placed directly into the model's context, an autoregressive predictor still generates its output by asking what is statistically likely to come next given this context, not what is required to come next given this context. Having the fact present in context makes the correct continuation highly probable, since a well-trained model will generally learn to weight in-context evidence heavily. Highly probable is not the same as required. The retrieved fact is one strong influence among many competing influences drawn from the model's training distribution, and nothing about the underlying computation prevents that distribution's other pressures, stylistic tendencies, competing plausible details, the accumulated weight of countless other walls the model was trained on, from occasionally winning out, particularly under distribution shift, adversarial or unusual framing, or simply across enough generation steps that low-probability deviation eventually occurs somewhere. This is a different failure than anything Chapters 2 and 3 described. Those chapters were about whether the right information makes

it into context at all. This chapter is about what happens once it is there: a fact present in context is honored with high probability, not with certainty, because the mechanism converting context into output was never built to enforce anything. It was built to continue plausibly, and plausible, however well-informed, is not the same claim as necessary.

## 4.5 What Causal Continuation Would Actually Require

It is worth sketching, without yet building it, what would need to be different for continuation to be causal rather than statistical. The next state of a persistent world cannot merely be conditioned on the current state, weighted alongside everything else a generative process has learned to find plausible. It needs to be constrained by the current state, in a sense a probability distribution over continuations does not, by itself, provide: some notion of which continuations remain admissible given what has actually been established, and a mechanism capable of ruling out inadmissible continuations outright rather than merely assigning them somewhat lower probability. A system that only makes the wrong crack-location less likely has not solved this chapter's problem. A system that makes it impossible has. This book will spend a considerable portion of its remaining chapters building exactly this distinction into precise, workable form; for now, it is enough to have located the gap precisely: not in what a generative system knows, which Chapter 3's con-

cession granted could in principle be perfect, but in what a generative system is architecturally capable of enforcing once it knows it.

## 4.6 Looking Ahead

The last three chapters have each isolated one piece of why current systems fail to sustain a persistent world: buffers that cannot distinguish forgetting from never-happening, document stores that cannot reconcile their own contents, and a generative principle that treats even perfectly retrieved facts as merely probable rather than binding. Chapter 5 draws these threads together by looking, in detail, at systems where this entire chain of failures is visible end to end, including one built simply enough that its mechanism can be inspected directly rather than inferred, to show concretely what generation without an underlying reality actually looks like in practice.

# Chapter 5

# Generation Without Reality

## 5.1 Two Faces of the Same Gap

Chapters 1 through 4 examined a failure that shows up across two acts of generation: a wall rendered once, then rendered again, with nothing connecting the two. It is worth naming, before this chapter goes further, a second failure that can occur even within a single act of generation, one this book will call groundlessness rather than persistence failure, because the two are related without being identical. Persistence failure is a problem of consistency across time: does the second observation agree with the first. Groundlessness is a problem within a single moment: does the first observation, considered entirely on its own, actually correspond to any underlying fact of the matter at all, or does it merely have the surface form of something that does. A system can fail at persistence while never failing at groundlessness, generating individually well-grounded observations that simply fail to agree with one another. A system can also fail at groundlessness while technically succeeding at persistence, in the degenerate sense that nothing it generates was ever grounded in anything to begin with, so

there is nothing for a second observation to contradict. This chapter is about the second failure, and about why solving persistence alone will not solve it.

## 5.2 InspiroBot: A Transparent Case Study

Most of the systems this book has discussed are complex enough that their internal operation has to be inferred rather than directly observed. InspiroBot, a website that generates faux-motivational slogans paired with stock-style imagery, is not. An early piece of reverse-engineering, written before the current wave of transformer-based generators existed, laid out its likely architecture in enough detail to be genuinely instructive: a large corpus of motivational phrases, words, and sentence fragments; grammatical templates that recombine those fragments into novel slogans; a database of background images selected to match; typography chosen to resemble the internet's existing genre of inspirational-quote memes; and a filtering or ranking stage that selects among many candidate outputs before showing one to the user.

What makes this worth examining closely is that every stage of this pipeline can be inspected, and inspection reveals something worth stating plainly: at no point does the system represent, check, or commit to any actual claim about inspiration, meaning, or wisdom. It recombines fragments according to a grammar built to preserve surface plausibility, nothing more. The output is not an asser-

tion the system believes, checked against anything, weighed against alternatives, or grounded in any underlying model of what makes advice good or true. It is a combinatorial artifact, and the grammatical scaffolding it preserves is sufficient to make it read as meaningful without there being any meaning behind it that the reading could be checking against.

## 5.3 Why the Absurdity Is the Point

It is worth being precise about where the apparent meaning in an InspiroBot slogan actually comes from, because it does not come from the generating system. A slogan that reads as either profound or absurd, often both at once, depending on the reader, is doing exactly what a grammatically well-formed but semantically ungrounded sentence should be expected to do: it presents enough structure for a reader to attempt a repair, an interpretation that would make the sentence make sense, and readers, being very good at this kind of repair, generally supply one. The humor, and occasionally the accidental profundity, is manufactured entirely on the reading side of the exchange. The generating side contributed grammatical scaffolding and thematically appropriate vocabulary. Everything that makes the output feel meaningful was added afterward, by an interpreter doing exactly the kind of charitable sense-making humans do reflexively with any output that has the right surface shape, whether or not anything underneath that shape earns the

interpretation.

## 5.4 Modern Systems Inherit the Same Gap, More Convincingly

It would be comfortable to treat InspiroBot as a curiosity, a simple system exploiting a simple trick, safely distant from the far more sophisticated generators this book is actually concerned with. This comfort does not survive close inspection. A modern language model is, in the specific sense this chapter cares about, a vastly more sophisticated version of the same underlying operation: navigating a learned statistical manifold and producing continuations whose grammatical, structural, and stylistic plausibility is extraordinary, without this plausibility being equivalent to, or guaranteeing, that any given output represents or was checked against an underlying fact of the matter. The gap does not close with scale. It becomes far harder to detect, because the surface plausibility is so much higher and the reader's repair instinct is triggered so much more reliably, and this is a large part of what makes the phenomenon usually called hallucination, a fabricated citation, an invented statistic, a confidently stated falsehood, so difficult to catch by inspection alone: a hallucinated citation is, in Chapter 4's vocabulary, statistically plausible, formatted exactly the way a real citation would be formatted, referencing a plausible-sounding author and venue, without being causally grounded in any actual paper that was checked

against. The citation is InspiroBot's slogan wearing academic clothing, produced by the same kind of process, exploiting the same reader instinct to supply, charitably, whatever grounding the output's surface form implies but does not actually contain.

## 5.5 The Common Root

This is the point at which persistence failure and groundlessness turn out to be closer than Chapter 5.1 initially suggested, and it is worth stating the connection precisely rather than leaving it as a loose family resemblance. Persistence failure is the absence of a substrate the second observation could be checked against. Groundlessness is the absence of a substrate the first observation could be checked against, which is really the same absence, viewed at the earliest possible moment rather than across a later comparison. A system with no persistent substrate at all cannot fail to be consistent with something that never existed to be consistent with, and it equally cannot fail to be grounded in something that never existed to ground it. Both chapters' worth of failures, the wall's shifting crack and InspiroBot's ungrounded profundity, trace back to the identical missing ingredient: something that exists, independent of any particular act of observation or generation, that generation is answerable to rather than merely influenced by.

## 5.6 Closing Part I

It is worth drawing the last five chapters together into a single diagnosis, because each has isolated one facet of what turns out to be a single problem viewed from different angles. Chapter 1 showed the symptom directly: a wall that fails to remain itself. Chapter 2 showed that current architectures have only buffers, mechanisms too finite and too indiscriminate to distinguish forgetting from never-happening. Chapter 3 showed that adding an explicit memory does not close the gap, because a document store answers what was written, not what currently is, and reconciles nothing. Chapter 4 showed that even perfect memory does not suffice, because the generative principle underlying these systems asks what is statistically plausible rather than what is causally required, honoring even perfectly retrieved facts with high probability rather than certainty. Chapter 5 has shown that the same missing ingredient produces a second, related failure even within a single act of generation, when there was never anything underlying to be faithful to in the first place.

The diagnosis, stated once, plainly, is this: none of these systems maintain something that persists independent of being observed, and none of them treat generation as answerable to such a thing rather than merely conditioned on it. Every failure this Part has catalogued, forgetting, contradiction, unreconciled documents, probable-but-not-guaranteed continuation, ungrounded plausibility, is a con-

sequence of that single absence, encountered from a different direction.

## 5.7 Looking Ahead

The rest of this book is an extended answer to what would have to exist, and how it would have to behave, for this absence to be filled rather than merely worked around. Part II begins building it.

# Part II

# The Semantic Field

# Chapter 6

# A World Is a Field

## 6.1 The Difference Between a Picture and a Place

Chapter 1 began with a simple observation: a player returns to the same stone wall in a virtual world and finds its cracks and moss subtly changed. At that point the example served only as a symptom, one instance among several of a failure this book had not yet named precisely. This book now has enough machinery in place, or is about to, to ask why the symptom occurs and what kind of architecture would make it impossible. The wall is the same wall. What has changed is the question being asked of it.

Return, then, to the experiment in its more deliberate form. Look at a stone wall, turn away, and look back. In the physical world this act is uneventful: the wall has not changed, and any difference in what you see is attributable to lighting, your position, or the passage of time acting on the stone itself. The wall persisted while you were not looking at it, and your second glance is a second observation of the same underlying thing.

Now perform the analogous experiment inside a gen-

erative system. Ask a model to render a stone wall. Turn away, in the sense of ending the generation. Ask again. The model produces a second wall: plausible, perhaps beautiful, quite possibly indistinguishable in quality from the first. But nothing in the system guarantees that the two renderings depict the same wall. The stones may be arranged differently. The moss may have migrated. The crack that ran diagonally through the third course of stones may simply not be there the second time, not because it healed, but because it was never a fact about anything. It was a detail invented once, for one rendering, and then forgotten completely.

This is easy to dismiss as a problem of visual consistency, the kind of thing a texture cache or a fixed random seed might patch over. That diagnosis is too shallow. The issue is not that the two walls look different. The issue is that there is no wall. There is only the act of generating a wall, performed twice, with no structure connecting the two acts. Consistency, when it happens, is a statistical accident of the model's priors rather than a consequence of anything persisting between the two renderings. A system built this way has not built a place. It has built a very capable picture generator, and a place is not a sequence of pictures, however good the pictures are.

The distinction matters because it is ontological rather than aesthetic. An observer's confidence that they are looking at the same wall twice depends on a claim about what exists between observations, not merely a claim about what

any single observation looks like. Current generative systems, from diffusion models to autoregressive world models operating on tokens or frames, are built almost entirely around the second kind of claim. They are extraordinarily good at producing an observation that is locally plausible. They contain no mechanism, and typically no representation at all, for the thing an observation is supposed to be an observation of. This chapter is about supplying that missing thing.

## 6.2 Reality Requires Persistence

The claim at the center of this book can be stated plainly before any mathematics is introduced:

> A world is not the sequence of its observations. A world is the persistent structure from which observations arise.

This sentence reverses the implicit assumption behind most generative architectures. Those architectures treat the observation as primary: the frame, the token, the rendered image is the object of interest, and whatever comes before it exists only to make the next observation plausible given the previous ones. Context windows, latent caches, and conditioning histories are all, in the end, mechanisms for making observation number *n* consistent with observations *1* through *$n-1$*. They are bookkeeping devices layered on top of a fundamentally observation-first ontology.

The alternative treats the observation as derived. There

is a persistent entity, call it the world-state, and what an agent sees at any moment is a partial, situated projection of that state rather than the state itself. Formally, we will eventually write this as an observation operator,

$$O_\theta(M_t) = y_t,$$

where $M_t$ denotes the world-state at time t and $y_t$ denotes what an observer actually perceives. The subscript θ marks that this projection is itself a model, parameterized and learned, standing in for a camera, a sensor, a rendering pipeline, or a language description depending on the modality in question. The crucial point, for now, is not the precise form of $O_\theta$ but its role. It is a *reading* operation performed on something that already exists. It does not manufacture the wall. It reports on the wall.

This single reversal has consequences that ripple through everything that follows. If observation is derived from state, then two observations of the same region should agree not because a model was trained to make them agree, but because they are readings of the same underlying structure, taken at nearby moments, through a stable operator. Disagreement becomes diagnostic rather than expected. A crack that appears and disappears is no longer an unremarkable feature of sampling variance; it is evidence that the system has no persistent state to be unfaithful to, or that the operator reading that state is unstable, or that the state itself was corrupted between readings. Once persistence is taken seriously as a requirement, inconsistency stops being

invisible noise and becomes a signal that something specific has gone wrong, and can in principle be traced to a specific failure in a specific part of the architecture.

Note what this section has and has not done. It has argued that observation must be derived from a persistent state, and it has named the operator that performs that derivation. It has not yet said what that persistent state is made of. That is a separate question, and it deserves its own preparation before an answer is given.

## 6.3 From Graphs to Fields

Suppose one accepts that a persistent world-state is necessary. The next question is what kind of mathematical object it should be. The most familiar candidates all share a common commitment, and that commitment is worth making explicit before it is abandoned.

A scene graph represents a world as a discrete collection of entities connected by discrete relations: this object is on top of that object, this character is inside that room. A knowledge graph represents a world as a discrete collection of typed nodes connected by discrete typed edges: this fact holds of this entity. A voxel grid represents a world as a discrete lattice of cells, each holding some fixed local content. A polygon mesh represents a world as a discrete collection of vertices, edges, and faces approximating a continuous surface. Each of these representations differs enormously in what it is good for, yet each begins from the same assump-

tion: that the world is fundamentally a collection of discrete, individuated things, and that representing the world means enumerating those things and their relations.

This assumption is not wrong so much as it is premature. Discreteness is a property that some regions of a well-behaved world exhibit, not a starting axiom that the representation should impose from the outset. A stone wall is, at a sufficiently coarse grain, well described as a discrete object with discrete properties. But the moss growing on it, the gradient of dampness near its base, the diffuse light falling across its surface, and the slow accumulation of soot from a nearby fire are not naturally discrete at all. Forcing them into a graph or a voxel grid means choosing a resolution and a set of categories in advance, and every choice of resolution and category is a choice about what the representation is permitted to notice. Discrete representations are efficient exactly to the degree that they have already decided what matters, and brittle exactly to the degree that something outside that decision turns out to matter after all.

A field-based representation inverts this commitment. Rather than beginning with entities and asking what properties they have, a field begins with a domain, typically a manifold of positions or a more abstract space of situations, and assigns to every point in that domain a value describing the local state of the world there. Objects, on this view, are not primitive. They are patterns that the field happens to exhibit: regions of high coherence, sharp gradients mark-

ing boundaries, stable configurations that persist under the field's own dynamics. Nothing is lost by allowing objects to emerge this way rather than positing them from the start, and a great deal is gained, because the same underlying representation can describe the wall, the moss, the dampness gradient, and the soot without deciding in advance which of these deserves to be a first-class entity and which is mere texture.

A careful reader might object that this is a problem of resolution rather than kind: why not simply use a much finer graph, with enough nodes and continuous-valued attributes to approximate whatever granularity the moss or the dampness requires? The objection misses what the discreteness commitment actually costs. However fine a graph becomes, building it still requires deciding, before a single value is written down, what counts as a node. Fineness does not remove that decision; it only postpones the point at which it has to be made explicit, and it multiplies the number of times it has to be made. A field requires no such decision. It assigns a value to every point of its domain from the outset, and whatever nodes, boundaries, or objects a graph would have needed to posit in advance become things one can instead read off the field's own structure: a boundary is simply where the field's values change sharply, an object is simply a region where they cohere. The field does not do without structure. It postpones commitment to structure until the structure has actually revealed itself, and that is precisely the property a persistent world-state needs.

This is the motivation for treating the persistent world-state as a field rather than a graph, and it is the motivation this book will build on for the remainder of the chapter.

## 6.4 Defining the Semantic Field

With that motivation in place, the central object of this book can now be stated. The world-state at time t is a field over a domain X, taking values in some appropriate space:

$$M_t : X \to \mathbb{R}^d.$$

Here X is the domain over which the world is defined. Depending on the application, X might be a literal spatial manifold, a more abstract semantic space, or some hybrid of the two; the formalism does not require committing to one interpretation at this stage. What matters is that $M_t$ assigns, to every point of X, a vector of real values describing the local state of the world at that point and at that moment. Where a scene graph would ask whether a wall-object exists and what properties it has, the field simply reports a value at every point of the domain, and wall-like coherence is something one can observe in how those values are organized, not something one has to declare up front.

This single equation is the culmination the chapter has been building toward, and it is worth pausing on the fact that so much preparation was required before it could be written down honestly. Introduced without the preceding sections, $M_t : X \to \mathbb{R}^d$ looks like an arbitrary formal choice,

one field-theoretic representation among many that could have been picked for reasons of mathematical convenience. Introduced after the preceding sections, it looks like the answer to a question the reader has already been made to ask: if the wall must persist between observations, and if persistence should not be purchased by forcing the world into a premature discreteness, what is left? A field over a domain, reporting local state, from which observations are drawn by projection rather than invention. That is what $M_t$ is for.

## 6.5 Observation Operators

It is worth pausing to name explicitly something that section 6.2 already introduced in passing, because it will recur throughout this book in increasingly sophisticated forms. The operator $O_\theta$, mapping the field $M_t$ to an observation $y_t$, is not a detail of implementation. It is the second primitive this book depends on, standing beside the field itself.

The field and the observation operator answer two different questions, and keeping them separate matters more than it might first appear. The field answers what exists: $M_t$ persists whether or not anything is currently reading it, in the same sense that the stone wall persists on a night when no one walks past it. The observation operator answers what is available to a particular observer at a particular moment, through a particular means of access: a human eye, a camera, a language description, a sensor array. Two observers with different operators, reading the same

field, may recover different aspects of it, or recover the same aspects at different fidelity, without this implying that the field itself is different or that either observer is mistaken. Nothing about $O_\theta$ constructs the wall. It only ever reports on a wall that was already there to be reported on.

This chapter will say nothing further about what $O_\theta$ looks like concretely, and that restraint is deliberate. The question of which quantities a given observation can actually recover, and the question of how something like a camera or a renderer might realize $O_\theta$ in practice, both depend on the field's internal structure, which has not yet been introduced. What matters here is only that the operator exists, that it is distinct from the field it reads, and that the distinction between world and observer will turn out to be one of this book's organizing threads rather than a footnote.

It is possible, even at this early stage, to sketch the complete loop that observation belongs to, without yet giving any of its parts a formal treatment. The field is observed, producing $y_t$. Some process of reasoning, not yet specified, turns that observation into a proposed change, written $\Delta\psi$. That proposal is not simply accepted; it is projected onto whatever states the field's own constraints permit, an operation this book will later write as $\Pi_C$, and only the result of that projection is committed, becoming the field's next state $M_{t+1}$:

$$M_t \xrightarrow{O_\theta} y_t \xrightarrow{\text{reasoning}} \Delta\psi \xrightarrow{\Pi_C} Y \xrightarrow{\text{Commit}} M_{t+1}.$$

Every box in that diagram will be revisited. Part III gives $\Delta\psi$, $\Pi_C$, and Commit their full treatment as distinct operators rather than a single undifferentiated update. Part IV supplies the mathematics that makes projection well defined. Part V shows how each stage is actually implemented. Nothing about the diagram will change as the book proceeds; what changes is only how much is written inside each box. It is worth carrying this loop forward from here, because nearly everything the book develops from this point on is an elaboration of one of its five stages.

## 6.6 Why Fields Instead of Objects

Treating the world as a field rather than a collection of objects changes what identity means. In an object-based ontology, identity is typically a label: this entity is the same entity across time because it carries the same identifier, the same handle, the same node in the graph. Labels are cheap to assign and expensive to justify. Nothing about attaching the same label to two states of affairs guarantees that anything actually persisted between them; the label can simply be reasserted by fiat.

In a field-based ontology, identity has to be earned differently. An object is not a label attached to a region of the field. It is a stable configuration that the field's own dynamics maintain over time: a coherent, low-entropy, self-reinforcing pattern that persists because the field's evolution keeps reproducing it, not because something outside

the field decided to keep calling it by the same name. The wall is a wall not because a symbol says so but because the relevant region of the field remains coherent, structurally continuous, and dynamically stable across the interval between two observations. If that region were to dissolve, drift, or reorganize beyond recognition, no label could rescue its identity; the wall would, in the most literal sense available to this framework, have stopped being the same wall.

This reframing has a further consequence that will not be developed fully until much later in the book, but that deserves to be planted here while the ground is fresh. If identity is the persistence of a trajectory through the field's state space rather than the persistence of a label, then questions about what a thing is become questions about what has continued, rather than questions about what has been asserted. This is a process ontology in the sense philosophers of that name would recognize, though this book has no need to argue for that tradition on its own terms. It is enough, for now, to notice that treating the world as a field rather than a collection of labeled objects was never merely a modeling convenience. It changes what it means for something to be the same thing twice, which is precisely the property that a stone wall needed and that a naive generative system could not supply.

## 6.7 Looking Ahead

This chapter has deliberately withheld an answer. $M_t : \mathrm{X} \to \mathbb{R}^d$ says that the world is a field and that every point of that field carries a vector of real values, but it says nothing about what those values encode. A vector in $\mathbb{R}^d$ is silent about whether it represents color, material, uncertainty, history, or something else again. That silence was appropriate here: the chapter's task was to establish that a persistent field is the right kind of object for a world-state to be, not to specify its contents.

The next chapter takes up that specification directly. If the world is a persistent field, what does each point in that field actually contain? That is the one question this chapter leaves open, and it is the question Chapter 7 exists to answer.

# Chapter 7

# The Structure of Memory

## 7.1 What a Field Must Remember

Chapter 6 left one question deliberately open. A world was established to be a persistent field, $M_t : \mathrm{X} \to \mathbb{R}^d$, and every point of that field was said to carry a vector of real values describing local state. But a vector in $\mathbb{R}^d$, taken on its own, is mute. It does not say whether its entries encode color or material or uncertainty or something else entirely, and a reader who has only been told that the world is a field is entitled to worry that this is a promissory note rather than an answer.

The worry deserves to be taken seriously, because it points at something true. A field that is merely a vector at every point, with no further structure imposed on what that vector must represent, is not yet capable of doing the work the previous chapter demanded of it. Recall the wall. Persistence required more than a value existing at the wall's location; it required that value to remain coherent under repeated observation, that regions of uncertainty be distinguishable from regions of settled fact, and that whatever had already happened to the wall leave some trace that later

observations could be checked against. None of this follows automatically from $M_t$ simply being $\mathbb{R}^d$-valued. It has to be built in, deliberately, by deciding what the vector at each point actually contains.

This chapter makes that decision. It gives the field an internal structure, one that will be assumed, referred to, and built upon by nearly every chapter that follows.

## 7.2 The Five Components of Memory

At each point x in the domain, the field's value decomposes into five named components:

$$M_t(x) = (\Phi_t(x),\, v_t(x),\, S_t(x),\, R_t(x),\, z_t(x)).$$

Each component answers a different question about that point of the world, and each is necessary in a way the others do not substitute for.

$\Phi_t(x)$, the coherence potential, answers how stable this region is. High values of $\Phi$ mark regions where structure is settled and mutually reinforcing; low values mark regions on the verge of reorganizing. $\Phi$ is what makes it meaningful to speak of a wall as opposed to an arbitrary patch of field: the wall is exactly the region where $\Phi$ is high and has remained high.

$v_t(x)$, the vector flow, answers where this region is going. It encodes directed change: drift, propagation, the local tendency of the field to move rather than merely to sit.

A field with no vector component would be static by construction, incapable of representing anything in the process of happening, and a persistent world must be able to represent things in the process of happening as readily as it represents things that have finished happening.

$S_t(x)$, the entropy, answers how settled this region actually is, as distinct from how coherent it currently appears. A region can have high $\Phi$ while also having high S, if what has been observed so far is coherent but thin, consistent with many different underlying possibilities that have not yet been distinguished. Entropy is the field's honest account of what it does not yet know about itself, and it receives a full chapter of its own, Chapter 8, because the consequences of taking uncertainty seriously as a first-class quantity turn out to be extensive.

$R_t(x)$, the causal residue, answers what has happened here. It is the trace left behind by prior events, prior commitments, prior interactions, the record that lets a later observation be checked for consistency against something more specific than the field's present appearance. Without R, the field would have no memory in the ordinary sense of the word; it would have only a current state, endlessly overwritten, indistinguishable from a system with no persistence at all beyond the current instant. R is developed fully in Chapter 11.

$z_t(x)$, the appearance, answers what this region currently looks like: the resolved, committed, perceivable content that an observation actually returns. Chapter 9 is de-

voted to explaining why appearance must be a distinct component rather than something derived on demand, and to the invention-to-memory principle that governs how z comes to hold a fixed value in the first place.

Together these five components give each point of the field an answer to five different questions: how stable, how it is changing, how uncertain, what has happened, and what it currently looks like. No one of these can stand in for another. A field with coherence but no residue could not distinguish a wall that has always been solid from a wall that has just been solidly imagined; a field with appearance but no entropy could not distinguish a confident perception from a plausible guess. The decomposition is not decoration. It is the minimum structure required for the persistence argued for in Chapter 6 to actually hold.

## 7.3 Refining the Decomposition

Readers familiar with this book's supporting technical material, in particular the Architecture Specification for factored world-engine updates, will recognize this decomposition as a revision of an earlier one. That document defined the field's state as a six-tuple,

$$M_t(x) = (\Phi_t,\, v_t,\, S_t,\, A_t,\, R_t,\, z_t),$$

with $A_t$ named as an affordance structure or interaction algebra: a component of the world-state itself, alongside coherence, flow, uncertainty, residue, and appearance. This

book does not use that six-tuple, and the omission is deliberate rather than an oversight to be quietly stepped around.

The reason is best seen through an example simple enough to settle the question outright. A ladder affords climbing to a human and does not afford climbing, in any useful sense, to a goldfish. The ladder has not changed between these two cases. What has changed is the capability of the party encountering it. If affordance were a property of the field alone, stored at the ladder's location the way $\Phi$ or z are stored there, this difference would have nowhere to live: the field would have to somehow encode a single, agent-independent fact about climbability, and that fact would either be wrong for the human or wrong for the goldfish. Affordance is not a property of a place. It is a relation between a place and a capability, and treating it as though it belonged to the world-state alone was a category error, however natural it seemed when the six-tuple was first written down.

The decomposition used in this book therefore removes $A_t$ as a field component. Affordances are reintroduced in Chapter 10 in a different form entirely, as a relation

$$A : M \times C \to \mathcal{P}(\Delta\psi),$$

where C denotes an agent's capability or cognitive state, and where the output is the set of proposed changes that agent's encounter with the field could give rise to. The persistent world stores coherence, flow, uncertainty, residue, and appearance. It does not store, and should not store, a ready-made list of what any given visitor can do with it.

What any given visitor can do with it is computed at the point of encounter, from the field's structure and the visitor's capability together, and Chapter 10 is where that computation is developed.

This is worth stating plainly as an instance of a general policy this book follows throughout: where an earlier formulation is superseded by a later and better one, the change is named rather than silently absorbed. The six-component decomposition was not a mistake so much as an intermediate stage, one that correctly identified affordance as important before correctly identifying where it belonged. Refining it is part of the same process that produced it.

## 7.4 What Observation Recovers

Chapter 6 introduced the observation operator $O_\theta$, mapping the field to what an observer perceives, and deliberately said nothing about what $O_\theta$ actually returns. With the field's internal structure now in view, that question can be given a real answer, or at least the shape of one.

No single observation recovers all five components at once, and this is not a limitation to be engineered away so much as a fact about what observation is. A camera pointed at the wall recovers something close to z, the field's current appearance, and recovers it richly. It recovers $\Phi$ and S only indirectly, if at all: a photograph does not directly report how stable a region is or how much remains unresolved about it, though a sequence of photographs, com-

pared against one another, might allow those quantities to be inferred. A physical measurement of stress or load, by contrast, might recover something closer to $\Phi$ directly, while saying very little about appearance. A historical record, a document, or a witness's testimony draws primarily on R, the residue of what has happened, often with no access to the field's current appearance at all.

This is exactly the behavior one should expect once observation is understood as a projection rather than a rederivation of the whole state. $O_\theta$ is not one fixed operator recovering one fixed slice of $M_t$; it is a family of possible operators, each suited to a different combination of components, each giving an observer partial access to a field that always contains more than any single observation returns. A rendering pipeline, a physics sensor, and a written record are not competing, imperfect attempts at the same underlying task. They are different projections of the same five-component structure, each faithful to the part of it they are built to read, none of them faithful to the whole.

This also clarifies something about disagreement between observers that was only gestured at in Chapter 6. Two observations of the same region can differ, sometimes sharply, without either being mistaken, simply because they are reading different components of a field that is not obligated to make every component agree in the way appearance alone might suggest. A region can look settled, high z-coherence, while still carrying high entropy about aspects the appearance does not touch. Recognizing this is part

of what it means to take the five-component structure seriously rather than treating it as an internal bookkeeping detail invisible from the outside.

## 7.5 Looking Ahead

This chapter has given the field a name for each of the five things it must keep track of, and has explained why a sixth candidate, affordance, does not belong on that list. It has not yet said how any of these five components actually behaves: how entropy resolves into certainty, how appearance becomes fixed once observed, how residue accumulates, or how coherence and flow interact as the field evolves. Each of the next several chapters takes up one of these questions in turn, beginning with entropy, the component whose behavior turns out to have the widest consequences for everything the rest of this book will build.

# Chapter 8

# Entropy as Ignorance

## 8.1 The Room You Have Not Looked At Yet

Return once more to the wall, but this time consider what lies just around the corner from it, a stretch of the same courtyard that has not yet been observed at all in this particular visit. Ask what the field contains there. The temptation is to say either that it contains nothing, an empty region waiting to be filled in, or that it contains something fully determinate that simply has not been reported yet, the way an unopened envelope already contains a specific letter. Both answers are wrong, and the difference between them and the correct one is the subject of this chapter.

The unobserved stretch of courtyard is not empty. Chapter 6 established that the field is defined over the whole of its domain, and Chapter 7 gave every point of that domain a coherence value $\Phi$, a flow v, a residue R, and an appearance z, whether or not anyone has recently looked there. The unobserved region has values. But those values are not yet settled in the way the wall's values are settled after repeated confirming observation. This is the sense in which the region is neither empty nor fully determinate: it holds

a live distribution over what it might turn out to be, a specific and structured uncertainty rather than a blank. Write $L_t(x)$ for this live possibility set, the collection of outcomes a point has not yet been forced to choose among. Entropy is the field's name for how wide $L_t(x)$ currently is.

$S_t(x)$, introduced in Chapter 7 as one of the five components of memory, measures exactly this width. High S at a point means $L_t(x)$ remains large: the courtyard might contain a bench, or gravel, or nothing at all, and the field has not yet been forced to choose. Low S means $L_t(x)$ has narrowed, usually because observation or some other constraint has pressed on it until only one possibility, or a tight cluster of similar ones, remains standing. Entropy is not a report of how much data exists about a point. It is a report of how much the point has been resolved.

## 8.2 Ignorance Is Structured, Not Blank

It is worth dwelling on why the blank-region reading is a mistake, because the mistake is a natural one and it recurs, in different clothing, throughout the rest of this book. A system that treats unobserved regions as literally empty, containing no committed structure at all, is a system with nothing to be responsible to when it eventually does render or describe that region. It can invent freely, because there was never anything there to be unfaithful to. This is precisely the failure mode that motivated the whole of Chapter 6: a wall regenerated from nothing twice is unfaithful to noth-

ing, because nothing was ever asserted about it in the interval between renderings.

Treating entropy as a real, structured field component closes this loophole. Even before the courtyard is observed, its high-S state is not a void; it is a specific claim, namely the claim that this region's outcome remains genuinely open, bounded only by whatever constraints the rest of the field already imposes on it. A courtyard adjoining a stone wall is more likely to be paved than to be a lake, not because it has been observed to be paved, but because the surrounding coherence structure already narrows the live possibilities before any direct observation occurs. Entropy, in other words, is never entropy about nothing. It is entropy conditioned on everything else the field currently holds, and this conditioning is itself part of what $\Phi$, v, and R contribute to a point even when S at that point remains high.

A motif worth planting here, one this book will return to more than once before it is fully unpacked: structure is not always a single, privileged fact waiting to be discovered. The Arabic script's twenty-eight letters admit two distinct, historically real orderings, the ancient Abjad sequence, inherited from Phoenician and Aramaic and still used for numerals and chronology, and the later Hijā'ī sequence, which regroups the same letters by shared shape to ease handwriting instruction and modern lookup. Neither ordering is the letters' one true structure. Both remain legitimate, serving different purposes, and adopting one for a given purpose does not retroactively invalidate the other. A live possibil-

ity set is not the same kind of thing as an alphabet's ordering, but the lesson generalizes: what counts as the settled structure of something can depend on what the settling is for, and this book will make that dependence precise well before it is done with the example.

## 8.3 Why Coherence Alone Cannot Do This Work

Chapter 7 flagged, without developing, a case that deserves attention now: a region can carry high coherence, high $\Phi$, while simultaneously carrying high entropy. This is worth making concrete, because it is the clearest argument available for why S must be tracked as its own component rather than treated as something derivable from $\Phi$.

Consider a courtyard that has been glimpsed only once, briefly, from a great distance. What was seen was perfectly consistent: nothing about the glimpse contradicted anything else in the field, and the impression it left behind is coherent in the sense that it hangs together without internal tension. High $\Phi$, in this case, is earned. But coherence of this kind says nothing about how much of the courtyard's actual structure was pinned down by that single distant glimpse. A great many different courtyards, differing in the placement of a bench, the presence of a second doorway, the color of the paving, would all have produced the same brief, consistent, distant impression. Coherence measures whether what has been established hangs together.

Entropy measures how much has been established in the first place. A field that only tracked $\Phi$ could not distinguish a richly confirmed region from a thinly glimpsed one that merely happens not to contradict itself yet, and that distinction turns out to matter a great deal once the field has to decide what it is and is not entitled to do with a given region.

## 8.4 The Entropy Constraint

This distinction earns its keep through a single governing rule, one of the few outright constraints this book will place on the field's behavior before Part III develops the machinery for constraints in general. Entropy may not decrease at a point unless that point has actually been observed. Formally, for any x outside the currently observed region,

$$\Delta S_t(x) \geq 0.$$

Read plainly, this says that the world is not permitted to quietly become more certain about itself in places nobody is looking. Uncertainty may grow on its own, as possibilities multiply or as the passage of time loosens what was once pinned down, but it may only shrink where an observation has actually done the work of shrinking it. The constraint does not freeze unseen regions. A tree can fall in the courtyard, or someone can move the bench, while the observer is elsewhere, and the field's own dynamics are free to change which outcomes remain live in $L_t(x)$ even at a point nobody

is watching. What the constraint forbids is narrower and more specific: silently collapsing that live set down to one preferred outcome, without an observation to earn the collapse. The support of the distribution may shift; its width may not shrink for free. This is a stronger claim than it might first appear, because it rules out a failure mode that is otherwise easy to fall into by accident: a system that lets its internal representation of unseen regions drift toward false confidence, settling on some particular content for the courtyard before anyone has looked at it, simply because settling is computationally convenient or because a generative model's priors nudge it toward some specific plausible guess. The entropy constraint forbids exactly this. It requires the field to keep its ignorance honestly wide wherever that ignorance has not been earned down.

The constraint also explains something about the wall that Chapter 6 could only gesture at. The reason the wall must remain the same wall between two observations is not merely that persistence sounds desirable. It is that the field is under a specific obligation, stated here for the first time as a rule rather than an intuition, not to have resolved anything about the wall in the interval when no one was observing it. Whatever was uncertain when the observer looked away must still be uncertain, at least as uncertain, when the observer looks back, unless something else in the field's own dynamics specifically earned a change.

## 8.5 Observation as the Only Legitimate Solvent

If entropy cannot decrease unearned, the natural question is what counts as earning it, and the chapter can answer this only in outline, since the full mechanism belongs to the chapter that follows. Observation, the operator $O_\theta$ introduced in Chapter 6 and given more texture in Chapter 7, is the field's principal legitimate means of reducing entropy at a point. When an observer actually looks at the courtyard, the resulting projection $y_t$ constrains what the field's true content there can be, and S is permitted, at that point and only that point, to fall.

What is important to fix now, before Chapter 9 develops it fully, is the shape of the obligation this creates. Once entropy has fallen at a point because it was observed, the specific resolution that observation produced, the specific bench, the specific paving, the specific absence of a second doorway, becomes something the field owes future observations. It is not enough for entropy merely to have decreased; what it decreased into must be remembered, or the field will simply drift back toward an unconstrained state the next time no one is looking, and the entire discipline this chapter has argued for will have been enforced for nothing. This is the invention-to-memory principle in embryonic form: an observation is permitted to invent a resolution to what was previously uncertain exactly once, and having invented it, the field is obligated to hold onto it as z rather than rein-

venting it afresh at the next observation.

## 8.6 Looking Ahead

This chapter has argued that entropy is a real, structured, and independently necessary component of the field, and it has stated the one rule that governs how entropy is allowed to change: never downward, anywhere, except where observation has specifically earned the decrease. It has deliberately stopped short of explaining what happens to the value a resolved region settles into, or why that value, once fixed, must be remembered rather than regenerated. That is the question Chapter 9 exists to answer, and it is the question this chapter's final section has already shown to be unavoidable: an entropy that is allowed to fall must fall into something, and that something needs a home in the field just as durable as $\Phi$, S, R, or the entropy constraint itself.

# Chapter 9

# Appearance Is Remembered

## 9.1 The Second Glance

Chapter 8 left the courtyard unresolved on purpose. Its content was live, uncertain, entropy high, and nothing in the argument so far said what happens the moment someone actually walks around the corner and looks. That moment is where this chapter begins.

Suppose the observer looks, and the field, through whatever process eventually generates a concrete perception from an uncertain region, produces a courtyard with a stone bench near the far wall and a scattering of gravel underfoot. This act of generation is not a violation of anything argued so far. It is, in fact, required: the courtyard had to become something specific eventually, and nothing about Chapter 8's entropy constraint forbids resolution at the moment of observation, only unearned resolution beforehand. A bench and some gravel, produced now, for the first time, are a legitimate answer to a genuinely open question.

Now suppose the observer walks away, comes back a minute later, and looks again. What must be true of the second glance for this to be the same courtyard rather than

merely a second, unrelated act of invention? The bench must still be there, in the same place. The gravel must still be gravel, not suddenly flagstone. Nothing about the second glance is permitted to reopen what the first glance already closed. This is a stronger requirement than it might first appear, because a generative system built the ordinary way, sampling plausibly from a learned distribution each time it is asked, will happily produce a coherent, high-quality courtyard on the second glance that has nothing to do with the courtyard the first glance produced. Both would be individually convincing. Neither would be faithful to the other. This is precisely the failure this book set out from in Chapter 6, now returned to at the one point in the field's dynamics where it actually occurs: the instant an uncertain region is first observed and must decide what it is going to remain.

## 9.2 The Two Kinds of Generation

It is worth separating two things that look identical from the outside but occupy entirely different roles in this framework, because collapsing them is exactly the mistake that produces the wall problem in the first place.

The first is generation as invention. This happens exactly once for any given region of high entropy, at the moment an observation first presses on it. Before that moment, nothing in the field specifies a bench rather than a fountain rather than an empty stretch of paving; the courtyard's entropy was genuinely high, and something has to break the

symmetry. Invention of this kind is not merely permitted, it is unavoidable, and there is nothing dishonest about it. A system that refused to invent anything until it had somehow already been told the answer would never be able to observe anything for the first time at all.

The second is generation as regeneration, and this is the pathology. It occurs when a system treats every observation as an occasion for a fresh act of invention, whether or not the region in question has already been resolved. Regeneration looks, locally, exactly like invention: the same decoder, the same renderer, the same plausible output. The difference is entirely a matter of what should have happened instead. Invention answers a question that was genuinely still open. Regeneration re-answers a question that had already been closed, discarding the previous answer without acknowledging that it ever existed.

The entire discipline this chapter is about to formalize exists to keep the first kind of generation legitimate while making the second kind of generation structurally impossible, not merely discouraged. This cannot be achieved by training a model to prefer consistency, because a model that merely prefers consistency will still, on a long enough encounter, invent a courtyard incompatible with its own earlier output and call it plausible. It has to be achieved by giving the field somewhere to keep what invention has already produced, so that the second glance is reading a memory rather than performing a second act of invention that happens to resemble the first.

## 9.3 The Invention-to-Memory Principle

That somewhere is z, the appearance component introduced in Chapter 7 and left, until now, largely undeveloped. Its update rule is the mechanism this chapter has been building toward:

$$z_{t+1}(x) = z_t(x) + \lambda \cdot \hat{z}(x).$$

Here $\hat{\mathsf{z}}(\mathrm{x})$ is a candidate: a proposed appearance for the point x, produced by whatever process generates candidates, a decoder, a renderer, a language model completing a description, when an observation presses on that point. It is not yet committed. The coefficient $\lambda$ determines how much of that candidate is actually allowed to update the field's held value at x, and its behavior over time is what separates invention from regeneration. When x is genuinely unresolved, when entropy there is still high and nothing has previously been committed, $\lambda$ can be large: the candidate is mostly accepted, because there is little existing content for it to conflict with, and this is invention doing its legitimate work. Once x has been observed and $z_t(x)$ already holds a specific, settled value, the same update rule behaves very differently. A new candidate $\hat{\mathsf{z}}(\mathrm{x})$ arising from a second observation is not free to overwrite what is already there; the update is constrained to move $z_t(x)$ only as far as continued fidelity to the existing value permits, and in the limiting case, once a region is fully settled, $\lambda$'s effective contribution collapses toward nothing. The bench does not become

a fountain on the second glance, not because the renderer forgot how to imagine a fountain, but because the update rule no longer gives an unconstrained candidate the power to overwrite an already-committed one.

This is the invention-to-memory principle stated in full: a region may invent its appearance once, while its entropy is genuinely open, and having invented it, the field is obligated to hold that appearance stable against future candidates rather than regenerating it. The principle does not forbid change entirely. A wall can be damaged, a bench can be moved, and Part III's treatment of proposals and constraint projection will have a great deal to say about how legitimate changes to an already-settled region are distinguished from illegitimate ones. What the invention-to-memory principle forbids specifically is change with no cause: appearance drifting simply because it was asked about again, with nothing in the intervening history to justify a different answer.

## 9.4 Appearance and Entropy, Together

It is worth making explicit a connection that the last two chapters have been circling without quite stating outright. Chapter 8's entropy constraint said that S may not fall at a point unless that point has been observed. This chapter's invention-to-memory principle says what happens at exactly the moment that license is used: entropy falls, and z is what it falls into. The two are not separate mechanisms that happen to cooperate. They are two descriptions of a sin-

gle event. To observe an uncertain region is to reduce its entropy and commit its appearance in the same act, and a field that did only the first without the second would have no way of remembering what its own reduced uncertainty had resolved into; a field that did only the second without the first would be committing appearances without any record of how settled or provisional they actually were. The five-component decomposition from Chapter 7 earns its keep exactly here, at the seam between S and z, where the honest bookkeeping of what remains uncertain and the honest bookkeeping of what has already been decided turn out to be two sides of a single operation.

## 9.5 Rendering as a Concrete Observation Operator

Chapter 6 introduced $O_\theta$ only as an abstract projection from field to observation, and Chapter 7 described it only as a family of operators, each recovering some partial slice of the field's structure. It is now possible to say something concrete about at least one member of that family, because appearance is the component most observation operators are actually built to read.

A renderer, in the ordinary sense, computes an approximation to $O_\theta$ specialized to z: given the field's current appearance at and around a point, it produces an image, a sound, a description, whatever the modality requires. This is the forward direction, field to observation, and it

is the direction Chapter 6 emphasized. But the invention-to-memory principle depends equally on the reverse direction. The candidate $\hat{z}(x)$ in the update rule has to come from somewhere, and it comes from an operator that runs, loosely speaking, backward: given an observation, or given the field's surrounding context at a point whose appearance is not yet settled, produce a proposal for what that point's appearance should be. This is inverse rendering in the broad sense this book will use the term, and diffusion decoders, language models completing a scene description, and physically grounded renderers run in reverse are all instances of it. The forward operator reads what the field already holds. The inverse operator proposes what the field might come to hold. Between them, they are the two halves of the observation-and-invention cycle this chapter has been developing, and Chapter 27 will return to both in their fully engineered form, as concrete software components rather than the conceptual operators sketched here.

It is worth being honest about what this section has and has not established. It has said that rendering and inverse rendering are the concrete faces of $O_\theta$ where appearance is concerned, and it has said that the candidate driving the invention-to-memory update comes from the inverse direction. It has not said how either operator is actually built, what architecture a renderer or an inverse renderer should use, or how the two are kept consistent with one another in practice. Those are engineering questions, and this book's four-pass structure defers them deliberately, the same way

Chapter 6 deferred the mathematics of constraint projection to Part IV. What matters here is only that $O_\theta$, introduced three chapters ago as an unspecified projection, has now acquired its first genuinely concrete instance.

## 9.6 Looking Ahead

Appearance answers what a region currently looks like, and this chapter has explained why that answer, once given, must be kept rather than regenerated. But looking like something and being usable are not the same question. A bench that has settled into the field with a fixed appearance affords sitting to a human passerby and affords nothing in particular to whatever else might encounter it. The field's appearance is now well defined; what a given observer is entitled to do with it is not yet addressed at all. That is the question Chapter 10 takes up, and it is the question that will finally put to use the capability-relative structure this book set aside back when the six-component decomposition was refined into five.

# Chapter 10

# Affordance Fields and Cue Coherence

## 10.1 The Ladder and the Goldfish, Revisited

Chapter 7 settled a question of principle with a deliberately simple example. A ladder affords climbing to a human and does not afford climbing to a goldfish, and since the ladder itself is unchanged between these two cases, whatever affordance is, it cannot be a property stored at the ladder's location in the field. It has to be a relation between the field and something the field does not itself contain: a capability, brought by whoever or whatever encounters it.

That example was chosen for its clarity, and clarity came at a cost. A ladder either affords climbing or it does not, for a given kind of visitor, and the binary nature of the example made the relational argument easy to see but easy to underestimate. Most affordances are not binary, and most encounters are not between a fixed object and a fixed species. Consider a toy car left on a table. To a small child, it affords pushing, racing, crashing into things, narrating a story about where it is going. To an adult tidying the room, it affords picking up and putting away. To a diagnostic engi-

neer, if the car happens to be a scale model with functioning parts, it might afford inspection of a suspension mechanism. None of these are wrong. None of them exhaust what the car is. What the car affords is not one fact waiting to be discovered but a structured space of possibilities, different slices of which become available depending on who or what is doing the encountering, and depending on what that encounter is for.

This chapter develops affordance as exactly this: a structured, capability-relative space of possible actions, arising at the point where the persistent field meets an agent's capacity to act on it. The relation stated informally in Chapter 7,

$$A : M \times C \to \mathcal{P}(\Delta\psi),$$

is this chapter's central object. M is the field context surrounding the point of encounter; C is the capability or cognitive state of the agent doing the encountering; the output is a set of viable proposals, candidates for the kind of change, $\Delta\psi$, that agent might now put forward. The rest of this chapter is concerned with what actually populates that output set, and with how it comes to be populated in the first place.

## 10.2 Cues: What Triggers an Affordance

Affordances do not present themselves all at once. A courtyard, once observed and settled into appearance as Chapter

9 described, does not hand its visitor an exhaustive list of everything that could ever be done there. Something has to draw attention to a particular possibility before it becomes live for that agent at that moment, and this book will call that something a cue.

A cue is a localized signal, arising from the field, that activates relevance for a particular capability. To avoid a collision with this book's own notation, since $R_t$ already names the field's causal residue, cue-triggered relevance will be written $\rho_c(x)$ rather than reusing R: a scalar field over the relevant space, peaking near whatever has drawn the cue's attention and falling away with distance from it. A bench's silhouette is a cue that activates sitting-relevance for a capability profile that includes the ability to sit. A crack running along the base of the wall is a cue that activates inspection-relevance for a capability profile that includes structural expertise, and activates nothing in particular for a capability profile that does not. The cue does not manufacture the affordance out of nothing; it draws a specific region of the field's already-existing structure into the foreground of a specific agent's attention. Behavior, in this framework, follows the gradient of activated relevance: an agent moves toward and engages with whatever has been drawn into the foreground, in proportion to how strongly it has been drawn there.

## 10.3 How Affordances Are Learned

A single encounter establishes that a given cue was, on that occasion, associated with a given affordance. It does not yet establish a stable disposition. That stability accumulates the way most stable dispositions in this book's framework accumulate: through repetition, weighted and reinforced over time rather than fixed after one instance.

Write $w_{ij}$ for the strength of the association between cue i and affordance j, for a given capability profile. Each encounter updates this weight by a simple rule,

$$w_{ij}^{t+1} = (1 - \alpha)\, w_{ij}^{t} + \alpha \cdot \text{affordance}_{ij},$$

where $\text{affordance}_{ij}$ is the strength with which that particular encounter realized affordance j in response to cue i, and $\alpha$ governs how quickly new encounters overwrite the accumulated history of old ones. This is nothing more exotic than a running average, but its consequences are worth spelling out. A cue that reliably predicts the same affordance across many encounters accumulates a strong, stable weight, and comes to trigger that affordance quickly and confidently on future encounters. A cue whose associated affordance varies from encounter to encounter never accumulates a strong weight toward any one of them, and continues to require deliberation each time it is met. This is how the same silhouette that once required careful inspection to identify as a bench, affording sitting only after some thought, comes eventually to trigger sitting-relevance im-

mediately and without deliberation: not because the object has changed, but because the association between that cue and that affordance has been reinforced past the point of needing to be re-derived.

## 10.4 The Vocabulary Affordances Draw From

The affordance relation's output, the set $\mathrm{P}(\Delta\psi)$ that $\mathrm{A}(\mathrm{M}, \mathrm{C})$ returns, is not an unstructured bag of arbitrary possibilities. It draws from a small, recurring vocabulary of operation types, and naming that vocabulary now will save considerable trouble later, when Part III develops what happens to a $\Delta\psi$ once it has been proposed.

Five operations recur across nearly every affordance this book will need to discuss: store, retrieve, modify, compose, and rewrite. To store is to commit some content to a location for later retrieval, the operation underlying anything like note-taking, saving, or the invention-to-memory principle from Chapter 9 itself. To retrieve is to recover previously stored content, the operation underlying recall, lookup, and reference. To modify is to alter existing content in place, the operation underlying editing, repair, and revision. To compose is to combine multiple pieces of content into a new structure, the operation underlying assembly, construction, and synthesis. To rewrite is to replace content according to some rule rather than an arbitrary edit, the operation underlying transformation, translation, and structured revi-

sion. Climbing a ladder is a physical instance of composition, joining an agent's trajectory to the ladder's structure to reach a location otherwise unreachable. Sitting on a bench is closer to a temporary modification of the agent's own state. Inspecting a crack is retrieval, pulling latent structural information into the open. The vocabulary is small enough to be worth memorizing and general enough to cover far more than its origin, in a formalism developed originally for externalized memory systems rather than physical affordances, might suggest. This book draws on that vocabulary only for its operational alphabet; the fuller apparatus surrounding it, including results establishing that systems built from these five operations are computationally universal, belongs to that formalism's own treatment and will not be redeveloped here.

## 10.5 Affordance Fields and Cue Coherence

An agent rarely encounters a single cue in isolation. A courtyard offers a bench, a crack, a doorway, and a patch of shade all at once, each activating its own relevance, each potentially inconsistent with the others in what it recommends. Sitting and inspecting the crack may compete for the same attention; moving toward the doorway and lingering in the shade may pull in opposite directions. Something has to determine whether the affordances activated across these different cues cohere into a single usable policy, or fragment into contradictory pulls that leave the agent's next

action undetermined.

This is a local-to-global consistency question, of exactly the shape a sheaf is built to answer, and it deserves to be stated as one now rather than left as a loose metaphor. Let CueCat be the category whose objects are cues or contexts and whose morphisms record how one cue's context relates to or refines another's. The affordance structure activated at each cue defines a presheaf,

$$F_{\text{cue}} : \text{CueCat}^{\text{op}} \to \text{Set},$$

assigning to each cue the set of affordances it locally activates, and to each relation between cues a way of restricting one context's affordances to a more specific one. The coherence question this chapter has been building toward is exactly the sheaf condition applied to $F_{\text{cue}}$: do the locally activated affordances, across overlapping cues and contexts, glue into a single section, a coherent policy the agent can actually act on, or do they fail to agree where their contexts overlap, leaving no consistent way to act on all of them at once?

It is worth being precise about what this sheaf is and is not a claim about, because a very different sheaf-theoretic construction is waiting in Chapter 21, and confusing the two would blur a distinction this book has gone to some trouble to keep sharp. $F_{\text{cue}}$'s gluing question is epistemic and agent-relative: it asks whether one agent's locally triggered possibilities for action can be assembled into one coherent policy for that agent, given that agent's capability profile.

It says nothing, on its own, about whether the world itself is coherent, whether the field's locally valid states assemble into one globally admissible reality. That is a separate question, concerning a separate sheaf, defined over a separate base category, and Chapter 21 will develop it as its own construction rather than as a restatement of this one. The two constructions are connected, not identical: a cue, after all, arises from an observation of the field, and Chapter 21's bridge section will make precise how observation maps cue-level coherence onto field-level admissibility. For now, this chapter's task is only to establish the first of the two sheaves cleanly, on its own terms, without borrowing the second one's vocabulary before it has been earned.

## 10.6 Looking Ahead

This chapter has given affordance a home appropriate to what it actually is: not a property stored in the field alongside coherence, flow, entropy, and appearance, but a relation, triggered by cues, learned through repetition, drawn from a small operational vocabulary, and coherent or incoherent across an agent's encounters in a sense now made precise. Something remains conspicuously absent from the account so far. Every affordance discussed in this chapter has been treated as though it arose fresh at the moment of encounter, with no reference to what has happened here before. But the courtyard has a history: the bench was placed at some point, the crack appeared at some point, and a full

account of what a region affords ought to be able to draw on that history rather than reconstructing everything from the field's current appearance alone. That history is what $R_t$, the causal residue introduced and then set aside in Chapter 7, was always meant to hold, and Chapter 11 is where this book finally gives it the treatment its role in the five-component decomposition has been waiting for.

# Chapter 11

# Causal Residue

## 11.1 What Appearance Doesn't Tell You

Consider two courtyards, identical in every respect that appearance can capture. Both have a stone bench near the far wall, weathered in the same way, positioned identically, indistinguishable to any observation reading z alone. In the first courtyard, the bench has stood in that spot since the wall itself was built. In the second, the bench was moved there yesterday, replacing an older bench that stood a few feet to the left. Appearance, as this book has developed it, cannot tell these two courtyards apart. $z_t(x)$ reports what a point currently looks like, and at this moment, both points look exactly the same.

The two courtyards are not, however, equivalent, and something in this framework needs to be able to say so. If a visitor arrives asking where the old bench went, the first courtyard has no answer to give because there was no old bench; the second courtyard owes an account, because something displaced. If a structural inspector wants to know whether the ground beneath the bench has borne this particular load for decades or for a single day, appear-

ance alone cannot answer, but the difference matters for what is admissible to happen next. Two states of the field can be visually and even structurally identical in the present while permitting entirely different futures, because what is admissible going forward depends not only on what a point currently is but on what sequence of events produced it. This is what appearance, on its own, cannot supply, and it is the reason the five-component decomposition set aside a slot for it as early as Chapter 7 without yet explaining why.

## 11.2 Residue as Compressed History

The obvious response is to propose that the field simply keep a complete log: every event, at every point, timestamped and retained indefinitely, so that any question about history can always be answered by consulting the record directly. This response should be resisted, and resisting it is the substantive content of this section.

A complete, unbounded log is not what $R_t$ denotes in this book, for two reasons that turn out to be related. The practical reason is the obvious one: a field that retained every event at every point without ever discarding anything would grow without bound, and a persistent world that cannot be maintained indefinitely without unbounded storage has not solved the persistence problem, it has merely deferred its cost. The deeper reason is that most of what a full log would contain is not actually needed to determine what remains admissible. What matters about the court-

yard's history is not the complete sequence of everything that ever happened there, but whatever subset of that sequence continues to constrain the present: that a bench was moved, roughly when, and what displaced it, not the precise angle of every hand that touched it in transit.

Residue, then, is compressed history: a summary trace, retained at each point of the field, sufficient to determine what remains admissible without retaining everything that produced it. This compression is not merely an engineering convenience bolted on afterward. It is itself an event, in the same sense that observation was shown in Chapters 8 and 9 to be an event rather than a passive read. When two histories are merged, or when an old event's specific detail is folded into a coarser summary because nothing downstream still depends on that detail, information is genuinely and irreversibly lost, in exactly the sense that merging two branches of a system's history always discards whatever was incompatible between them. Residue is what remains after this discarding, and it is worth being honest that discarding has occurred rather than treating $R_t$ as though it were a lossless archive that merely happens to be summarized for convenience.

## 11.3 How Residue Accumulates

Residue is not fixed at the moment a point is first observed, the way this book's earlier chapters might suggest by analogy with appearance. It accumulates, receiving a contri-

bution each time an event actually occurs at or near that point, whether or not the point is currently being observed. Schematically,

$$R_{t+1}(x) = R_t(x) \oplus \text{event}_t(x),$$

where $\oplus$ denotes whatever accumulation operator combines existing residue with a new event's trace. This book will not attempt, at this stage, to pin down $\oplus$ precisely; unlike z's update rule, which is a genuine weighted average admitting a clean closed form, residue accumulation is closer to a structural merge, more like combining two branches of a causal graph than blending two numbers, and its full algebraic treatment belongs to Part III and Part IV, once constraint projection and the causal-DAG structure of multi-agent updates have been properly developed. What matters here is only the shape of the claim: residue grows by accumulation rather than being fixed once and held constant, and each accumulation step is itself subject to the same discipline against unearned change that governed entropy and appearance in the two preceding chapters. A point's residue does not drift on its own; it only accumulates in response to something that actually happened.

## 11.4 Residue and Affordance

Chapter 10 introduced a learning rule for affordances, in which the weight connecting a cue to an affordance strengthens with repeated, consistent encounters and weak-

ens or fails to consolidate when encounters disagree. It is worth noticing now, with residue properly in view, that this learning rule was already describing a form of residue accumulation without naming it as such. The weight $w_{ij}$ at a given location is itself a compressed trace of that location's history of activating affordance j in response to cue i: not a full log of every encounter, but a summary sufficient to determine how strongly that association should hold going forward. Residue and learned affordance are not two unrelated mechanisms that happen to share a family resemblance. Affordance learning is a special case of residue accumulation, restricted to the particular kind of event this book calls an encounter, and this closes a loop between Chapter 10 and this chapter that was left open when affordances were first introduced as though they arose fresh from cues with no reference to what had come before.

## 11.5 Residue as a Constraint on Future Invention

This chapter closes by returning to the invention-to-memory principle from Chapter 9 and strengthening it. That principle stated that an already-settled appearance cannot be casually overwritten by a fresh candidate. It is now possible to say what an unsettled point's candidate must actually be checked against, and the answer is not only the point's own prior appearance, which for a genuinely unresolved point may not exist yet, but the point's

accumulated residue. A courtyard whose residue records that a bench was recently moved constrains what invention is permitted to produce there: an observer generating a first appearance for the newly bare patch of ground the bench used to occupy is not free to invent anything at all, because residue has already ruled out large classes of otherwise-plausible content, ground that has clearly been recently disturbed does not admit an invented appearance of undisturbed, decades-settled paving. Residue, in other words, does not merely record what happened for the sake of answering questions later. It actively narrows the space of admissible invention going forward, and in this sense it is doing for entropy exactly what appearance was shown to do for observation in Chapter 9: giving Chapter 8's entropy constraint teeth, by specifying not only that unearned certainty is forbidden but which specific certainties remain earnable given what has already occurred.

## 11.6 Closing Part II

This chapter completes the account this book set out to give in Chapter 6. The world is a persistent field, $M_t : \mathrm{X} \to \mathbb{R}^d$, and that field's value at every point decomposes into five components, each answering a question the others cannot answer for it: $\Phi$, how stable a region is; v, where it is moving; S, how much about it remains genuinely open; R, what has happened there; and z, what it currently looks like. Affordance, the space of actions a given capability can take

at a given point, is not a sixth component stored alongside these five, but a relation arising at their intersection with an agent's own capability, triggered by cues, learned through repetition, and coherent or incoherent across an agent's encounters in a sense this book has now made precise.

Nothing in the last six chapters has said how any of this changes. $\Phi$, v, S, R, and z have each been described in terms of what they hold and what constrains their update, but the actual mechanism by which a proposed change becomes a committed one, the machinery hinted at as early as Chapter 6's five-stage cycle and touched again in the invention-to-memory principle, has been assumed rather than built. Part III takes up that construction directly. It begins where this book's core cycle diagram always pointed: with $\Delta\psi$, the proposal, and the question of where a proposal like that actually comes from.

# Part III

# Dynamics

# Chapter 12

# Writing Reality

## 12.1 The Missing Half of the Loop

Part II specified, in considerable detail, what the field contains and what it is not permitted to do on its own. Coherence, flow, entropy, residue, and appearance each received a chapter; the entropy constraint and the invention-to-memory principle each received a rule; affordance received a relation, $A : M \times C \to P(\Delta\psi)$, that produces, at the meeting of field and capability, a set of candidate changes an agent might now propose. And there the account stopped. A set of candidates is not yet a change. Nothing in Part II said how any member of that set actually becomes the next state of the world, and the five-stage cycle sketched all the way back in Chapter 6, $M_t$ through observation, reasoning, proposal, projection, and commit, has so far had only its first stage properly developed.

This chapter takes up the second half of that loop: writing. Reading, the subject of Chapters 6 through 9, asks what a field already contains and how an observer comes to know it. Writing asks how a field comes to contain something new. The two are not mirror images of one another, and

treating them as though they were, as though writing were simply reading run in reverse, is the mistake this chapter exists to head off before Part III's later chapters build the machinery that makes writing rigorous.

## 12.2 Two Ways to Get This Wrong

There are two failure modes available to a system that has to write new content into a persistent field, and it is worth naming both before describing the discipline that avoids them, because each one is a natural mistake to make and each has already been anticipated, in outline, by the supporting technical material this book draws on.

The first failure is unconstrained overwrite. A system commits to whatever a proposal suggests, wholesale, with nothing checking the proposal against what the field already holds. This is fast and expressive, capable of producing anything a generative process can imagine, and it is also exactly the mechanism behind the wall problem that opened this book: a courtyard that can be overwritten freely on every observation is a courtyard with no persistence at all, regardless of how good any single overwrite looks in isolation. A system built entirely this way drifts, contradicts its own history, and violates the entropy constraint and the invention-to-memory principle as a matter of routine rather than exception.

The second failure is its mirror image: no writing at all, or writing so heavily filtered by consistency checks that

nothing new is ever actually admitted. A field that refuses every proposal that isn't already implied by what it currently holds is perfectly consistent and perfectly inert. It cannot represent a bench being moved, a wall being damaged, or a tree falling in the courtyard, because every one of these events is, at the moment it is proposed, a departure from what was previously true. A system built entirely this way is stable in exactly the sense that a photograph is stable: nothing in it will ever contradict itself, because nothing in it will ever happen.

Neither failure is hypothetical. The first is what an unconstrained generative model does by default. The second is what an overcautious constraint system does when it mistakes consistency for correctness. A working account of writing has to thread between them, and the shape it needs to take is the subject of the rest of this chapter.

## 12.3 Writing as Additive Proposal

The discipline that avoids both failures starts from a single reframing: writing is not replacement, it is proposal plus accumulation. Rather than specifying the field's next state directly, a write operation specifies a change, and that change is combined with the field's current state rather than substituted for it:

$$M_{t+1} = M_t + \Delta\psi.$$

This equation looks almost too simple to carry the

weight this chapter has been building toward, and its simplicity is deliberate. $\Delta\psi$ is not the world. It is a proposed departure from the world, and the plus sign is doing real work: it says that whatever $\Delta\psi$ contains is added to $M_t$, not laid over it in place of what was there. A field updated this way retains everything $\Delta\psi$ does not specifically address, which is most of the field on any given update, and changes only what the proposal actually speaks to.

Readers who have followed the last three chapters will recognize this shape, because it is not new. Chapter 9's invention-to-memory update, $z_{t+1}(x) = z_t(x) + \lambda \cdot \hat{\mathsf{z}}(\mathrm{x})$, is a special case of exactly this pattern, restricted to the appearance component and governed by a coefficient that grows more conservative as a region settles. Chapter 11's residue accumulation, $R_{t+1}(x) = R_t(x) \oplus \text{event}_t(x)$, is another special case, using a different combination operator suited to history rather than appearance. What this chapter introduces is the general form these two specific rules were always instances of: a single write operator, $\Delta\psi$, that can touch any or all of the field's five components at once, combined with the field's current state through whatever accumulation rule each component requires rather than through outright replacement. Writing reality, in this framework, always means adding to it, never starting over.

## 12.4 What a Proposal Is Not Yet

It is important to be precise about what this chapter has established and what it has deliberately withheld, because the temptation to treat $M_{t+1} = M_t + \Delta\psi$ as a finished account is exactly the temptation that leads back to the first failure mode described in section 12.2. Nothing said so far constrains what $\Delta\psi$ is allowed to contain. As written, the equation permits a proposal that violates the entropy constraint, contradicts accumulated residue, or overwrites a settled appearance with something incompatible, because addition alone does not check a proposal against admissibility before combining it with the field. A $\Delta\psi$ that adds a second bench identical to the first is harmless. A $\Delta\psi$ that adds a contradiction, asserting that the courtyard's ground is both undisturbed and freshly dug in the same update, is not, and the equation as stated has no way yet to refuse it.

This is not an oversight. It is the reason Part III has four more chapters before this cycle is complete. Chapter 13 takes up where $\Delta\psi$ actually comes from: not an arbitrary function but a neural proposal process, reading the field and an agent's intent to produce a specific candidate rather than a specific candidate having simply appeared. Chapter 14 takes up what happens to a candidate once it has been proposed: projection onto whatever states the field's constraints actually permit, catching exactly the kind of contradictory $\Delta\psi$ described above before it can be combined with $M_t$ at all. Chapter 15 takes up the difference

between a projected candidate and a genuinely committed one, a distinction this book has been careful to keep separate since the roadmap first insisted on Commit as its own step rather than folding it into projection. This chapter's equation, $M_{t+1} = M_t + \Delta\psi$, describes the shape writing takes. It does not yet describe the discipline that makes that shape safe to use, and readers should carry both halves forward rather than mistaking the first for the whole.

## 12.5 Writing and Affordance

It is worth closing the loop back to Chapter 10 explicitly, because the affordance relation introduced there was always aimed at exactly this chapter's subject without yet having anywhere to deliver its output. $\mathrm{A} : \mathrm{M} \times \mathrm{C} \to \mathrm{P}(\Delta\psi)$ produces a set, not a single change: at a given point of encounter, an agent's capability activates some range of candidate proposals, climbing the ladder, sitting on the bench, inspecting the crack, each a different possible $\Delta\psi$ arising from the same encounter. Chapter 10 stopped there deliberately, because nothing in that chapter's scope explained how one candidate out of that set actually becomes the $\Delta\psi$ that gets written. That selection, the step this book's core cycle labeled simply reasoning, is not addressed by this chapter either. What this chapter has done is give the selected candidate somewhere to go once it has been selected: added to $M_t$, not substituted for it, subject to the further discipline the next three chapters will supply. The question of how selec-

tion itself happens, how an agent chooses among a menu of live affordances rather than merely having one available, belongs to Chapter 13, where $\Delta\psi$ finally receives its own proposal mechanism rather than being treated as a term that simply appears on the right-hand side of an equation.

## 12.6 Looking Ahead

This chapter has given writing its basic shape: addition rather than replacement, a proposal combined with the field rather than substituted for it, generalizing the specific update rules Chapters 9 and 11 had already introduced for appearance and residue into a single operator capable of touching any part of the field at once. It has also been explicit about what remains unfinished. A proposal with no account of where it comes from and no check on what it is allowed to contain is not yet a safe mechanism for changing a persistent world, only a description of the mechanism's outward shape. Chapter 13 supplies the first missing piece, asking what actually generates $\Delta\psi$ in the first place.

# Chapter 13

# Neural Proposals

## 13.1 A Menu Is Not a Decision

Chapter 10 left affordance as a set: $\mathrm{A}(\mathrm{M}, \mathrm{C})$ returns $\mathrm{P}(\Delta\psi)$, the range of changes a given capability could propose at a given point of encounter, climbing, sitting, inspecting, each a live candidate rather than a foregone conclusion. Chapter 12 then explained what happens to a single $\Delta\psi$ once it exists, added to the field rather than substituted for it, but was careful to say that it had not explained how one candidate gets selected from many, or how a candidate's actual content, the specific shape of a specific proposed change, comes to be generated in the first place. A menu is not a decision, and this chapter is where the decision gets made.

## 13.2 Two Sources for One Proposal

It helps to notice, before building the selection mechanism, that $\Delta\psi$ is not really produced by a single process. It has two distinct sources, and collapsing them into one obscures a distinction that later chapters will need.

The first source is the field's own intrinsic tendency to

change, independent of any agent's intent. A coherence gradient smooths itself over time; an entropy field that has been locally disturbed relaxes back toward its surroundings; residue accumulates a small amount of drift even without deliberate action. This book will write this contribution as $F_{\text{phys}}(M)$: not physics in the sense of the physical sciences necessarily, but physics in the sense this book has used the word since Chapter 8, the field's own rule-governed behavior, occurring whether or not anyone is watching or acting.

The second source is deliberate and learned: a neural forcing term, $F_\psi$(M, p, a, o), driven by the field's current state M, an agent's position or point of encounter p, an intended action a, and whatever observation o currently constrains the proposal. This is where the affordance menu from Chapter 10 actually gets converted into content. Given that climbing the ladder has been selected, $F_\psi$ is what actually generates the specific proposed change, a trajectory, a displacement, a new configuration, that climbing would produce. It is called neural because it is learned rather than hand-specified: the mapping from context to proposed content is exactly the kind of function a trained model is suited to approximate, and this book will not pretend that mapping could be written down in closed form the way the entropy constraint was.

The full proposal combines both sources,

$$\partial_t \Pi = -\delta H/\delta M + F_\psi(M, p, a, o),$$

where the first term captures the field's intrinsic relax-

ation and the second captures the agent-driven forcing. $\Delta\psi$, as it appeared in Chapter 12, is the discrete-time accumulation of this combined rate over one update step. It was never purely an agent's invention. Some of it is simply what the field would have done regardless of any agent's presence, and separating the two contributions matters because they will be held to different standards once Chapter 14 asks whether a proposal is admissible: a field's own intrinsic drift is, almost by construction, close to what the field already permits, while a neural forcing term, expressive by design, has no such guarantee and is where most of a proposal's risk actually lives.

## 13.3 Selecting Among Affordances

This still leaves the question Chapter 12 set aside: given a menu of live candidates, sitting, climbing, inspecting, each independently viable, how does one of them actually become the a that $F_\psi$ is asked to realize? The mechanism this book adopts is the one Chapter 10 already built the vocabulary for, without yet putting it to use. Recall $\rho_c(x)$, the cue-triggered relevance introduced in section 10.2. Selection among a menu of candidates follows relevance the same way behavior was already said to follow it: a candidate is chosen with probability increasing in how strongly its associated cue has activated relevance for the agent's current capability,

$$\pi(a \mid x) \propto \exp(\beta \cdot \rho_c(x, a)),$$

where β governs how sharply selection favors the most strongly activated candidate over its rivals. A high β makes selection nearly deterministic, always taking the most strongly cued option; a low β allows genuine exploration among candidates whose activation is close but not equal. This is not a new primitive introduced for this chapter alone. It is the same relevance-following behavior Chapter 10 attributed to affordance activation in general, now made responsible for a specific job: turning a menu into a single chosen action, which $F_\psi$ then converts into a specific proposed content.

## 13.4 Expressivity Without Authority

It is worth stating plainly what has and has not been achieved once a is selected and $F_\psi$(M, p, a, o) has produced a concrete $\Delta\psi$. What has been achieved is content: a specific, well-formed proposal, grounded in both the field's intrinsic tendencies and a genuinely learned, genuinely expressive model of what a chosen action would actually change. What has not been achieved is permission. Nothing about being selected by π, and nothing about being generated by a well-trained $F_\psi$, guarantees that the resulting $\Delta\psi$ respects the entropy constraint from Chapter 8, the invention-to-memory discipline from Chapter 9, or the accumulated residue from Chapter 11. A neural proposal,

however well trained, is exactly the kind of process that can produce a fluent, locally plausible $\Delta\psi$ that nonetheless asserts something the field's own history has already ruled out, a courtyard resolving into undisturbed paving where the residue already records that the ground was recently dug.

This is not a defect to be trained away. It is the reason the proposal layer and the constraint layer are kept as two separate stages rather than one, and it is worth being explicit about why, since the temptation to merge them is real. A proposal generator that has fully internalized every constraint the field could ever impose would need to carry the entire admissibility structure of Chapter 14 inside itself, retrained or reasoned through on every update, and would likely purchase this at the cost of exactly the expressivity that made the proposal layer worth having in the first place. This is the second failure mode from Chapter 12 in a new guise: a proposal mechanism so heavily constrained at generation time that it can no longer propose anything genuinely novel is functionally indistinguishable from the rigid, inert system that chapter warned against. The proposal layer is deliberately allowed to be wrong. Its job is expressivity, not authority, and authority belongs to the stage that comes next.

## 13.5 Looking Ahead

This chapter has given $\Delta\psi$ a real origin: a combination of the field's own intrinsic tendencies and a learned, agent-driven forcing term, with selection among competing candidates governed by the same relevance-following behavior Chapter 10 introduced for affordance activation generally. What it has not done, deliberately, is check any of this against what the field is actually permitted to become. A proposal, however well motivated and however fluently generated, is still only a candidate. Chapter 14 is where a candidate is finally held against the field's constraints and either accepted, corrected, or refused.

# Chapter 14

# Constraint Projection

## 14.1 Where Proposals Go to Be Checked

Return to the proposal Chapter 12 used to illustrate what addition alone cannot catch: a $\Delta\psi$ that asserts the courtyard's ground is simultaneously undisturbed and freshly dug, fluent enough to have been generated by a well-trained $F_\psi$, selected with high relevance by the policy Chapter 13 described, and still, on its own terms, incoherent. Nothing in the last two chapters explains what stops this proposal from becoming part of the field. This chapter explains that.

## 14.2 The Constraint Manifold

Every rule this book has stated since Chapter 8 has, without using the word, been describing a constraint: a condition a field state must satisfy to be admissible. It is worth collecting them now under a single name. Write C for the constraint manifold, the set of field states this book's framework regards as admissible. C is not introduced here for the first time; it has been accumulating members since Part II began, and this section's task is only to name what was

already implicit.

Chapter 8 contributed the entropy constraint: no admissible state may show a local entropy decrease at a point outside the region actually observed during the update that produced it. Chapter 9 contributed the invention-to-memory discipline: no admissible state may hold an appearance at a settled point that discards what was previously committed there without cause. Chapter 11 contributed residue consistency: no admissible state may hold an appearance or entropy value that contradicts what the point's own accumulated residue has already ruled out, undisturbed paving where the residue records recent digging. Chapter 10's bridge between cue coherence and field coherence, not yet formalized but already promised, will eventually contribute a further member, ruling out states where an agent's realized action fails to correspond to any admissible field transformation at all.

C, in other words, is not a single equation waiting to be written down in this chapter. It is the intersection of every discipline this book has argued for since Chapter 8, growing as the book's argument grows, and this chapter's job is not to complete it, which properly belongs to Chapter 18's fuller formal treatment, but to explain what happens once a proposal is checked against whatever of it has already been established.

## 14.3 Projection as Nearest Admissible State

Given a proposal $M_t + \Delta\psi$ that may or may not lie in C, projection does not simply accept or reject it. It finds the nearest point in C to what was proposed:

$$Y = \underset{Y \in C}{\operatorname{argmin}} \|Y - (M_t + \Delta\psi)\|^2.$$

Read this as a kind of semantic pressure solve, in the sense a physical simulation enforces incompressibility or a mechanical system enforces a joint constraint: rather than discarding a proposal that oversteps what is admissible, the system finds the closest admissible configuration to the one that was actually suggested, preserving as much of the proposal's intent as the constraints allow. Write $\Pi_C$ for this operation as a whole, so that Y = $\Pi_C(M_t + \Delta\psi)$. The contradictory courtyard proposal from section 14.1 does not simply vanish under this operation. It is pulled toward whatever admissible resolution lies nearest to it, most plausibly the resolution the residue actually supports, freshly dug ground with an appearance to match, discarding only the part of the proposal that residue had already ruled out, and keeping whatever else the proposal specified that did not conflict.

This is worth dwelling on because it explains something about the character of this framework that could otherwise seem harsh. Constraint projection is not a censor that vetoes

proposals outright. It is closer to a correction: it trusts the proposal's direction and intent, and adjusts only what admissibility requires adjusting. A neural proposal that gets almost everything right and stumbles on one incompatible detail will emerge from projection nearly intact, missing only the detail that could not survive contact with what the field already holds.

## 14.4 Revisiting the Two Sources

Chapter 13 distinguished $\Delta\psi$'s two sources, the field's intrinsic relaxation $F_{\text{phys}}$ and the learned neural forcing $F_\psi$, and suggested, without yet justifying the claim, that the first is close to admissible almost by construction while the second carries most of a proposal's risk. Constraint projection is where that claim can finally be earned rather than merely asserted.

$F_{\text{phys}}$ is defined as the field's own rule-governed tendency to change, the same tendency that produces entropy relaxation, coherence smoothing, and the ordinary drift of residue over time. Because these tendencies are themselves derived from the same discipline that defines C, an entropy field that relaxes according to its own dynamics does not thereby violate the entropy constraint, and a coherence field that smooths according to its own dynamics does not thereby violate coherence consistency. $F_{\text{phys}}$, in other words, is constructed to already respect C, in the same sense that a ball rolling under gravity along a track built to con-

strain it does not need a separate check to confirm it stays on the track. Projected, $F_{\text{phys}}$ typically maps close to itself, and the distance $\|\Upsilon - (M_t + \Delta\psi)\|$ contributed by this term alone is usually small.

$F_\psi$ carries no such guarantee, and this is the honest version of the claim Chapter 13 made informally. A neural forcing term is trained to be expressive, to propose changes that a hand-specified $F_{\text{phys}}$ never could, sitting on a bench, moving through a doorway, inventing an appearance for a previously unobserved patch of ground, and expressivity of this kind is exactly what makes a proposal likely to reach outside C. Most of the actual work constraint projection performs, on a typical update, is work performed on the $F_\psi$ contribution to $\Delta\psi$, correcting or trimming what the learned, expressive half of the proposal got wrong, while the intrinsic half passes through largely unchanged. The asymmetry Chapter 13 flagged was correct, and this section is where it stops being an informal impression and becomes a consequence of how $F_{\text{phys}}$ and C were built from the same discipline in the first place.

## 14.5 What Projection Cannot Fix

Nearest-point projection assumes that some admissible state is worth moving toward, that C contains something in the vicinity of what was proposed. This assumption can fail. A proposal sufficiently incompatible with everything the field currently holds, not merely a locally contradictory

detail but a wholesale departure from the courtyard's accumulated coherence, entropy, and residue all at once, may have no nearby admissible state worth settling for; the nearest point in C might be so distant from the original proposal that accepting it would honor almost none of what the proposal actually intended. Projection, as this chapter has developed it, still returns some answer in this case, the mathematics of the minimization does not fail. But an answer this distant from what was proposed is arguably not a correction of the original intent at all, and treating it as one risks quietly replacing a bad proposal with an unrelated one under the appearance of continuity. This book will not resolve that tension here. It belongs to the same family of questions as outright refusal, the deliberate rejection of a proposal rather than its correction, and refusal has been mentioned only in passing, in the Spherepop-derived material this book draws on elsewhere, as an irreversible elimination of branches rather than a repair performed on one. Whether and when projection should give way to refusal is a question this book sets aside for now and returns to only once Part IV has given constraint projection its full mathematical treatment.

## 14.6 Looking Ahead

This chapter has given proposals somewhere to go once they have been generated: projected onto the nearest state the field's accumulated constraints actually permit, with the

field's own intrinsic tendencies passing through nearly untouched and a proposal's learned, expressive content bearing most of the correction. What this chapter has not yet done is explain how a projected state, Y, actually becomes $M_{t+1}$. The two are not automatically the same thing, and Chapter 15 is where that distinction, promised as far back as this book's roadmap, is finally made precise.

# Chapter 15

# Committing Reality

## 15.1 Two Different Kinds of "Done"

Chapter 14 ended with Y, the nearest point in C to a proposed update, computed and ready. It is tempting to treat this as the end of the story: a proposal was made, it was checked, it was corrected where necessary, and the result is simply what the field becomes next. This chapter argues that the temptation should be resisted, because computing Y and the field actually becoming Y are two different kinds of event, and collapsing them erases a distinction this book's roadmap insisted on from the very first sketch of its core cycle back in Chapter 6.

Y is a fact about mathematics. Given $M_t$, $\Delta\psi$, and C, the projection $\Pi_C(M_t + \Delta\psi)$ has a determinate answer, computable in principle without anything actually having happened to the world. $M_{t+1}$ is a fact about history. It is what the field genuinely becomes, the state that Chapter 11's residue will remember having occurred, the state future observations will be checked against. A Y that is computed but never committed is not a diminished version of $M_{t+1}$. It is not part of the field's history at all, in the same

sense that a sentence drafted and then deleted was never part of the document it might have joined. Commit is the operation that turns the first kind of done into the second, and it deserves to be treated as its own event rather than an automatic consequence of projection having finished.

## 15.2 Why Projection Alone Cannot Decide

There is a concrete reason projection cannot simply be trusted to settle the matter on its own, beyond the conceptual tidiness of keeping computation and history separate. Projection, as Chapter 14 developed it, answers a question about a single proposal in isolation: given this $\Delta\psi$, what is the nearest admissible state? It does not, on its own, know whether some other proposal, arising elsewhere or from another agent at the same moment, has already claimed the very state Y would move toward, or proposed something incompatible with it. Two proposals can each individually project to a perfectly admissible Y, and still be jointly impossible to commit together, because their combination is not itself admissible even though neither alone was at fault. Something has to arbitrate this, and it cannot be projection, because projection was never given the other proposal to check against in the first place.

## 15.3 Multiple Proposals, One History

This becomes unavoidable once more than one agent is proposing changes to the same field at overlapping moments, which any persistent, shared world must eventually allow. Two kinds of relationship can hold between simultaneous proposals. Updates are commutative when they touch disjoint or otherwise compatible parts of the field, one agent's proposal at the courtyard's bench, another's proposal at a doorway on the far side, in which case both may commit without either needing to know about the other or without their order mattering to the final result. Updates are non-commutative when they conflict, both proposing incompatible changes to the same region, in which case order or arbitration matters and at most one, or some admissible combination of parts of each, can actually be committed.

This book will not resolve conflicts of the second kind through an explicit negotiation protocol, a designated authority deciding whose proposal wins. Resolution instead follows admissibility directly: among the proposals contending for a given region, whichever combination remains inside C after joint projection is what commits, and proposals or portions of proposals that cannot be reconciled with the rest are the ones left out. Committed changes accumulate into a causal record, a structure this book will develop no further here than to note that it behaves as a directed graph of dependency rather than a single linear sequence, since two commutative updates genuinely have no fact of

the matter about which happened first. Chapter 29 returns to this in its fully engineered form, as the concrete mechanism behind sharing one persistent world across multiple simultaneous observers and agents; this chapter's task is only to establish that commit, not projection, is where multiplicity gets resolved.

## 15.4 Refusal Revisited

Chapter 14 closed by flagging a case its own machinery could not fully handle: a proposal so incompatible with the field's accumulated state that the nearest admissible point in C is not really a correction of it at all, but something else entirely, honoring almost none of what was actually proposed. That chapter left the question open because projection, as a pure nearest-point computation, has no mechanism for declining to answer. It always returns some Y, however distant.

Commit does have such a mechanism, and this is where this book locates it. Before a computed Y is written into the field's history, commit may decline to write it, treating the original proposal as void rather than silently substituting the distant admissible point Y happens to name. This is refusal in the sense this book's supporting material gestures at without fully developing: not a correction performed on a proposal, but an irreversible decision that a given branch of possibility, the one the proposal was reaching for, does not become part of this field's history at all. A proposal that gets

refused leaves no trace, no partial commitment, no compromise state standing in for what was actually intended. It simply does not happen, and the field proceeds as though it had never been proposed. This book will not specify, at this stage, exactly how large a distance between proposal and projection triggers refusal rather than acceptance; that threshold depends on details Part IV is better positioned to formalize. What matters here is that refusal has a home now, at commit rather than at projection, and that the question Chapter 14 left open was never really about projection's mathematics. It was about a decision projection was never equipped to make.

## 15.5 Commit as the Point of No Return

This gives residue, introduced in Chapter 11, its precise relationship to the write cycle this Part has been developing. Residue accumulates only from committed events. A Y that was computed, considered, and refused contributes nothing to $R_{t+1}$, exactly as though the proposal behind it had never been generated. A Y that is committed becomes part of the field's history permanently, in the sense this book has used permanence throughout: not immune to future change, a wall can still be damaged, a bench can still be moved, but no longer erasable as though it had never occurred. This is what makes commit the point of no return in this framework, and it is the reason this book's roadmap insisted, several chapters before this one was drafted, that pro-

jection and commit be kept as visibly separate steps rather than folded into a single operation. A projected state is a possibility under consideration. A committed state is a fact the rest of the field's future is now obligated to be consistent with.

## 15.6 A First Glimpse of Stability

One consequence of this chapter's account is worth naming now, briefly, ahead of the chapter that develops it properly. A field that keeps committing genuinely new content, region by region, cannot remain forever in flux; some regions will eventually stop changing, not because nothing more could be proposed for them, but because whatever continues to be proposed there projects and commits to the same state it already held. A courtyard whose bench has settled, whose residue is dense and specific, and whose entropy has long since fallen to near zero is a region where $\Delta\psi$, once selected and projected, tends to commit to something indistinguishable from what was already there. This is the first appearance, informal and unstated as a rule, of the fixed-point behavior Chapter 16 exists to formalize.

## 15.7 Looking Ahead

This chapter has separated computing an admissible state from actually committing it, explained why multiple simultaneous proposals require commit rather than projection to

arbitrate between them, and given refusal, left unresolved at the end of Chapter 14, a proper home as a decision commit is entitled to make. What remains is to ask what happens when this cycle, propose, project, commit, repeats indefinitely at a single region without anything further changing. That stability, and the conditions that produce or fail to produce it, is the subject of Chapter 16.

# Chapter 16

# Fixed Points

## 16.1 What "Nothing Happens" Actually Means

Chapter 15 closed with a brief, deliberately informal image: a courtyard region so settled, so densely supported by accumulated residue and so thoroughly resolved in appearance, that whatever continues to be proposed there tends to commit back to what was already true. It is worth pausing on that image before formalizing it, because "nothing happens" turns out to describe two quite different situations, and this book's framework needs to keep them apart.

The first situation is genuine inertness. No $\Delta\psi$ is proposed at this region at all; nothing presses on it, nothing forces, and the field simply holds its last committed value because nothing has asked it to do otherwise. A stretch of bare, undisturbed paving, far from any doorway or bench, well outside the reach of any cue strong enough to activate a proposal, is stable in this sense: stable by default, through absence of pressure rather than through any active work.

The second situation looks identical from a distance and is, in an important sense, the opposite. Consider a fountain

at the courtyard's center. Water moves through it continuously; $\Delta\psi$ is being proposed there at every update, not occasionally but constantly, since flow is precisely what a fountain's vector field v is supposed to represent. And yet the fountain, observed a minute apart, a day apart, a year apart, is recognizably the same fountain, its form unchanged even though the specific water occupying it has been replaced many times over. This is stability achieved through continuous, active maintenance rather than through absence of proposal, and this chapter's task is to give this second kind of stability, the more interesting and by far the more common kind for anything this book would want to call an object, a precise formal description.

## 16.2 The Fixed Point Condition

Both situations satisfy a single equation, once the write cycle from Chapters 12 through 15 is stated in its settled form. A field state $M^*$ is a fixed point of the update cycle when projecting and committing whatever it proposes for itself returns the same state:

$$M^* = \Pi_C(M^* + \Delta\psi).$$

This is the same condition the supporting technical material behind this book states in two closely related forms, $M^* = \text{Update}(M^*)$ in the architecture specification's language and $M^* = \Pi_C(M^* + \Delta\psi)$ in the dynamics module's, and this chapter adopts the second form because it makes

explicit what the first leaves implicit: reaching a fixed point is not a matter of nothing being proposed, but of whatever is proposed surviving projection and commit unchanged.

Read this way, the equation does not distinguish the bare paving from the fountain. Both satisfy it. What distinguishes them is what $\Delta\psi$ actually is at each: at the bare paving, $\Delta\psi$ is zero or close to it, and the equation holds trivially, because there is nothing to project or commit in the first place. At the fountain, $\Delta\psi$ is substantial and constantly regenerated, and the equation holds nontrivially, because the specific flow proposed at each update, once projected against the field's constraints and committed, happens to reproduce the fountain's form rather than drifting away from it. The next section makes this distinction explicit, because collapsing it back together would undo the very thing this chapter exists to separate.

## 16.3 Two Kinds of Fixed Point

Call a fixed point trivial when $\Delta\psi = 0$ at every point it covers: nothing is proposed, so nothing needs to survive projection, and the state persists purely by default. Call a fixed point dynamic when $\Delta\psi$ is genuinely nonzero, sometimes substantially so, but the cycle of proposal, projection, and commit reliably returns the same state regardless. The fountain is a dynamic fixed point. So, in fact, is the stone wall this book opened with, once its full history is taken into account: weathering proposes small changes to its surface continu-

ously, residue accumulates, and the wall's form nonetheless persists because whatever is proposed for it projects back onto essentially the same admissible configuration, update after update.

This distinction matters because dynamic fixed points are doing something trivial fixed points are not. A trivial fixed point persists because it costs nothing to maintain; there is, in a real sense, nothing actively holding it in place beyond the absence of disturbance. A dynamic fixed point persists despite continuous pressure to become something else, and its stability is therefore a genuine achievement of the field's own dynamics rather than an accident of neglect. Most of what this book, and its reader, would naturally call an object, a wall, a bench, a fountain, a courtyard as a whole, turns out on inspection to be a dynamic fixed point rather than a trivial one, however static it may appear to casual observation.

## 16.4 Identity as a Fixed Point, Not a Label

This closes a loop left open since Chapter 6. Section 6.6 argued that identity, in a field-based ontology, cannot be a label attached to a region, and had to instead be something the field's own dynamics actively maintained: a coherent, self-reinforcing pattern that persists because the field's evolution keeps reproducing it, not because something outside the field decided to keep calling it by the same name. That description was necessarily informal at the time, since nei-

ther $\Delta\psi$, projection, nor commit had yet been introduced. It can now be stated exactly. An object, in this framework, is a dynamic fixed point of the write cycle: a region where $M^* = \Pi_C(M^* + \Delta\psi)$ holds nontrivially, where proposal continues to arrive and continues to be absorbed back into the same admissible configuration rather than drifting away from it. The wall is the wall not because a label says so, but because it is, quite literally, a solution to this chapter's equation, reproduced update after update against a continuous stream of proposals that could, in principle, have taken it somewhere else and consistently did not.

## 16.5 Stability and Its Failure

Not every fixed point is equally robust, and it is worth flagging the distinction here even though its full treatment belongs to Part IV. A fixed point is stable, informally, when a small perturbation, a proposal slightly outside what the region has settled into, gets absorbed back toward $M^*$ on the next cycle rather than growing into something else. A fixed point is unstable when the opposite happens: a small perturbation, rather than being corrected, is amplified by the next round of proposal and projection, carrying the region away from $M^*$ rather than back toward it. A well-built wall is a stable fixed point in this sense; a precarious stack of loose stones, satisfying the fixed-point equation for as long as nothing disturbs it, may be an unstable one, technically a solution to $M^* = \Pi_C(M^* + \Delta\psi)$ while remaining one small

proposal away from becoming a very different solution entirely. This book will not develop the mathematics of stability further here; Part IV's treatment of variational dynamics and constraint manifolds is where the distinction between a fixed point that recovers from disturbance and one that merely has not yet been disturbed receives its proper formal footing.

## 16.6 Looking Ahead

This chapter has treated the field's update cycle as though it operated on one uniform clock, proposal, projection, and commit happening together, at the same rate, everywhere. This was a simplification, and Part I's original architecture specification already anticipated why it could not hold in general: appearance and flow tend to change quickly, while coherence, structure, and identity tend to change slowly, and a region can be a fixed point with respect to one of these timescales while still actively settling with respect to another. A fountain's water is never at rest, yet the fountain's overall form was already treated in this chapter as fixed; those two claims are only compatible once fast and slow dynamics are allowed to operate on separate clocks rather than one. That separation, and what it means for a single region to be simultaneously settled and still-evolving, is the subject of Chapter 17.

# Chapter 17

# Multiple Time Scales

## 17.1 The Fountain Reconsidered

Chapter 16 left a loose thread on purpose. The fountain was offered as a dynamic fixed point, a region under continuous proposal that nonetheless persists, and the example did real work establishing that stability need not mean inactivity. But the example also quietly assumed something Chapter 16 never examined: that the fountain's water and the fountain's form update on the same clock, so that a single fixed-point equation, checked once per update, could meaningfully describe both at once. This assumption does not survive close inspection. The water occupying the fountain at this instant is entirely different water a minute from now, flowing, splashing, replaced continuously; nothing about its appearance or its flow field is remotely fixed from one moment to the next. The fountain's stone basin, by contrast, is not meaningfully different a minute from now, or an hour from now, or most days. Treating these two facts as though they were answered by the same equation, checked at the same rate, was a simplification, and this chapter exists to undo it.

## 17.2 Fast and Slow Components

The field's five components do not all change at comparable rates, and this book's supporting technical material already anticipated the division this section makes explicit. Appearance, z, and flow, v, are fast: they are exactly the components most sensitive to whatever is happening right now, water moving through the fountain, light shifting across the wall, and they are proposed against, projected, and committed on a clock fine enough to capture that immediacy. Coherence, $\Phi$, is slow: it aggregates over many fast-timescale events before it meaningfully shifts, since a single moment's flow or appearance rarely changes how stable a region is overall, and the topological or identity-level structure that $\Phi$ helps define moves slower still, changing only when enough has accumulated to actually warrant it. Residue, R, sits between the two, accumulating at whatever rate events actually occur, sometimes fast, when a region sees frequent activity, sometimes slow, when little happens for long stretches, and this book will treat R's rate as inherited from whatever it is accumulating rather than fixed to either extreme.

This is not a new mechanism bolted onto the write cycle Chapters 12 through 15 already established. It is the observation that the same cycle, propose, project, commit, does not have to run at one uniform rate across every component it touches. Fast components run through many cycles in the time slow components run through one, and a region can be, quite legitimately, in the middle of vigorous

fast-timescale activity while its slow-timescale structure sits still.

## 17.3 Separate Clocks, Same Cycle

It is worth being precise about what does and does not change once fast and slow components are given separate clocks. Nothing about proposal, projection, or commit is different in kind; a fast-timescale update still proposes a $\Delta\psi$, still projects it onto the nearest state admissible under C, still either commits or refuses the result exactly as Chapters 13 through 15 described. What changes is only the rate at which this cycle runs for a given component. The fountain's water proposes, projects, and commits new appearance and flow values on a clock fine enough to track individual moments of motion. The fountain's basin proposes, projects, and commits new coherence values on a clock coarse enough that, across any span of time a casual observer would notice, nothing about it visibly changes at all, even though the cycle is, in principle, still running underneath.

## 17.4 Revisiting Fixed Points

This sharpens the trivial-versus-dynamic distinction Chapter 16 drew, and it is worth returning to that chapter's language directly rather than leaving the two accounts standing side by side unreconciled. A dynamic fixed point, as

Chapter 16 described it, is a region under continuous proposal that nonetheless returns to the same state. It is now possible to say more precisely what this means: a dynamic fixed point is fixed at the slow timescale while remaining active, often highly active, at the fast timescale. The fountain's form, $\Phi$, is a genuine fixed point, checked and reconfirmed on the slow clock; the fountain's water is not fixed at all, cycling continuously on the fast clock, and the fountain as a whole is correctly called stable only because these two facts are being reported at two different rates rather than one. A trivial fixed point, by contrast, is fixed at both timescales simultaneously: nothing proposed, nothing cycling, at either rate. The bare, undisturbed paving from Chapter 16 is trivial in this fuller sense, not merely because no single $\Delta\psi$ was proposed for it at some particular moment, but because neither its fast nor its slow components are undergoing any cycle at all.

## 17.5 When Timescales Interact

The two clocks are not fully independent, and the most important claim this chapter has to make is that fast-timescale activity, given enough time, is what eventually forces slow-timescale change. A single moment of water striking the fountain's basin does nothing to its coherence; the basin's $\Phi$ is far too slow to respond to any individual fast-timescale event. But water striking the same spot, moment after moment, across enough fast-timescale cycles, accumulates

residue, R, at that point, exactly as Chapter 11 described, and residue accumulated far enough eventually licenses a slow-timescale proposal that would not have been admissible earlier: erosion, a hairline crack, eventually a change to the basin's coherence itself. This is how a region moves from being a dynamic fixed point to no longer being one, not through any single dramatic event, but through fast-timescale activity slowly accumulating, via residue, until it crosses a threshold the slow timescale is finally forced to answer for. The two clocks are coupled through exactly the mechanism this book spent Chapter 11 establishing, and this is arguably the clearest illustration yet of why residue needed its own component rather than being treated as a byproduct of appearance.

## 17.6 Closing Part III

This chapter completes the account Part III set out to give. Chapter 12 established that writing is addition rather than replacement, $M_{t+1} = M_t + \Delta\psi$. Chapter 13 gave $\Delta\psi$ its origin, a combination of the field's own intrinsic tendencies and a learned, agent-driven forcing term, selected among competing candidates by relevance. Chapter 14 gave every proposal somewhere to be checked, projected onto the nearest state the accumulated constraints of C actually permit. Chapter 15 separated that computation from the historical fact of commitment, and gave refusal a proper home in the gap between them. Chapter 16 showed that persistence it-

self, the very property this book set out from in Chapter 6 to explain, is nothing more exotic than a fixed point of this cycle, and this chapter has shown that the cycle does not run at one rate, but at as many rates as the field's components require, coupled to one another through the same residue that Part II had already given a home.

Nothing in these six chapters, deliberately, has said what C actually is in full, what argmin means when the space being minimized over has the structure this book's fields require, or what guarantees, if any, ensure that a fixed point exists at all rather than being an accident of a particular courtyard's particular history. Those are Part IV's questions. Chapter 18 begins there, with the constraint manifold this Part has invoked six times without yet defining.

# Part IV

# The Mathematics of Persistence

# Chapter 18

# Constraint Manifolds

## 18.1 What This Part Owes the Reader

Six chapters of Part III were written against a promise this book has not yet kept. C appeared in Chapter 14 as the constraint manifold, the set of admissible field states, and has been invoked in every chapter since: $\Pi_C$ projected proposals onto it, argmin measured distance to it, $M^*$ solved an equation stated in terms of it. At no point was C actually defined as a mathematical object, only gestured at through the rules that were supposed to carve it out. This was a deliberate deferral, consistent with the four-pass structure this book announced in its roadmap: Parts II and III developed C's role architecturally, establishing what it needed to do, before Part IV develops what it actually is. This chapter keeps that promise, and the chapters following it keep the promises Part IV itself will go on to make.

## 18.2 Assembling C from Its Members

The honest way to define C is not to write down a single equation and declare it the constraint manifold, but to col-

lect what earlier chapters already established and show that their intersection is what C has been all along.

Chapter 8 contributed the entropy constraint: the set of states for which $\Delta S_t(x) \geq 0$ at every point outside the region actually observed during the update in question. Call this $C_S$. Chapter 9 contributed appearance consistency: the set of states in which a settled $z_t(x)$ is not overwritten by a fresh candidate without cause, formalized through the invention-to-memory update itself. Call this $C_z$. Chapter 11 contributed residue consistency: the set of states whose appearance and entropy values do not contradict what accumulated residue at that point has already ruled out. Call this $C_R$. Chapter 10 promised, and Chapter 21 will formalize, a further member governing the realizability of affordance-driven proposals as genuine field transformations; call this $C_A$, understood for now only as a placeholder this chapter cannot yet complete.

C is the intersection of these:

$$C = C_S \cap C_z \cap C_R \cap C_A \cap \cdots,$$

and the ellipsis is honest rather than decorative. Nothing in this book's argument so far guarantees that the list is complete, and Part IV's later chapters, on conservation laws, category theory, and information geometry, will each have occasion to add further members before the list can be called closed. What matters for this chapter is only that C has never been an arbitrary constraint invented for Part IV's convenience. It is the accumulated weight of every disci-

pline Parts II and III already argued for, now written as a single intersection rather than scattered across six chapters.

## 18.3 The Space the Field Actually Lives In

For "nearest point," "intersection," and "manifold" to mean anything precise rather than serving as evocative language borrowed from geometry, the space these terms operate in has to be specified. This book takes the field's configuration space to be a genuine differentiable manifold, written $\mathcal{M}$, equipped with a metric g that gives the norm in Chapter 14's projection equation, $\|\mathrm{Y} - (M_t + \Delta\psi)\|^2$, actual mathematical content rather than standing in as a placeholder for "some notion of closeness." A point of $\mathcal{M}$ is a complete assignment of $\Phi$, v, S, R, and z across the field's domain X; the metric g measures how far apart two such assignments are, in whatever combination of coherence-distance, flow-distance, entropy-distance, residue-distance, and appearance-distance the full theory eventually specifies component by component.

C, then, is not a separate kind of object from $\mathcal{M}$. It is a subset of $\mathcal{M}$, carved out by the intersection of conditions in section 18.2, and in the well-behaved cases this book will generally assume, a submanifold: a lower-dimensional surface sitting inside the larger space of all conceivable, not-necessarily-admissible field configurations. Projection, $\Pi_C$, is now exactly what differential geometry already means by the term: given a point of $\mathcal{M}$ not lying on the submanifold

C, find the nearest point of C under the metric g. Nothing about Chapter 14's equation was metaphorical. It was waiting for this section to specify the space in which it is literally true.

## 18.4 Hard Constraints and Soft Dynamics

It is worth separating, within C's definition, two kinds of condition that Part III's supporting material already distinguished informally. Hard constraints fix C's shape outright: mass consistency, identity persistence, causal locality, and topological admissibility are not negotiable by any proposal, however well motivated, and no $\Delta\psi$, however strongly selected by the relevance mechanism of Chapter 13, can move the field outside the region these conditions carve out. Soft dynamics, by contrast, are what varies within whatever region the hard constraints permit: the specific deformation a courtyard's coherence undergoes, the specific texture an appearance settles into, the specific trajectory a proposed affordance realization follows. $F_{\text{phys}}$ and $F_{\psi}$, introduced in Chapter 13, were always proposing within the space hard constraints define, never proposing changes to the constraints themselves. This distinction gives precise content to a claim Chapters 12 through 17 made repeatedly without formalizing: that physics, in this book's sense of the word, lives in C, while the write cycle's proposal machinery lives in the freedom C leaves open.

## 18.5 When the Domain Itself Changes

Chapter 8 stated its entropy constraint as unconditional, and deliberately declined to qualify it, on the grounds that qualifying it there would have weakened a chapter whose entire force depended on stating a rule plainly. That deferral is settled here. The entropy constraint, and the hard constraints generally, are unconditional on a fixed field domain X. They say nothing, on their own, about what happens when X itself changes: when a topology repair merges two previously distinct regions, or an identity restructuring splits one region into two, altering the very space over which $\Phi$, v, S, R, and z are defined.

The resolution is transport rather than exception. Given a structure-preserving map $\varphi$ relating the old domain to the new one, whatever hard constraints held on the old domain are carried across by $\varphi$ and continue to hold on the new one: a condition stated as $C_S$ on X becomes $\varphi_* C_S$, its pushforward, on the new domain X′, and the entropy constraint on X′ is exactly this transported condition rather than a fresh rule reasoned out from nothing. Nothing about the constraint has been relaxed, and nothing about it needed to be. What has changed is only the space it is being asked to hold over, and transport is precisely the operation that lets a rule stated once continue to mean the same thing after the space beneath it has moved. This book will not develop the full apparatus governing which maps $\varphi$ are eligible to carry constraints across a topology change; that question belongs

with the sheaf-theoretic material Chapter 21 develops, since a topology change of this kind is exactly the situation in which local field-patches need to be reconciled with a new global cover.

## 18.6 What This Chapter Has Not Established

It is worth being explicit about the gap this chapter leaves open, because Chapter 16 asked a question this chapter's machinery still cannot answer. $M^* = \Pi_C(M^* + \Delta\psi)$ presumes that a nearest point in C exists for whatever is being projected, and existence of a nearest point is not automatic once C is merely known to be a submanifold. A submanifold that is not closed, or a metric that is not complete, can leave some proposals with no nearest admissible point at all, or with several equally near, and Chapter 14's argmin silently assumed that this pathology would not arise. Nothing established in this chapter rules it out. Resolving it requires more than a definition of C; it requires understanding the dynamics that move a proposal toward C in the first place, and whether those dynamics are governed by a well-behaved energy functional whose minimization is guaranteed to terminate somewhere. That is a variational question, not a purely geometric one, and it is where Chapter 19 begins.

## 18.7 Looking Ahead

This chapter has given C a real definition: the intersection of every discipline Parts II and III argued for, sitting as a submanifold inside the field's full configuration space, distinguished into hard constraints that fix its shape and soft dynamics that vary within it, with a transport rule now on record for what happens when the underlying domain itself changes. What it has not given is any guarantee that projection onto C behaves well, that a nearest point exists, or that the write cycle's repeated application of $\Pi_C$ actually settles into the fixed points Chapter 16 described rather than oscillating or failing to converge at all. Chapter 19 supplies the missing piece: an energy functional, in the proper variational sense, whose minimization is what argmin was always secretly asking for.

# Chapter 19

# Variational Dynamics

## 19.1 What Argmin Was Really Asking

Chapter 14's projection operator was stated as a minimization: $\Upsilon = \text{argmin}_{Y \in C} \|\Upsilon - (M_t + \Delta\psi)\|^2$. At the time, this was introduced as a single-step computation, find the nearest admissible point to a given proposal, without asking whether it belonged to anything larger. It is worth asking now. A minimization performed once, at a single moment, in isolation from everything else the field does, is a coincidence if it is not secretly an instance of a more general principle governing the field's behavior throughout. This chapter argues it is not a coincidence, and supplies the principle.

## 19.2 An Energy Functional for the Field

Suppose the field's coherence, flow, and entropy jointly determine a single scalar quantity, an energy, at every configuration the field could take. Write this as a functional $H$, built from a Lagrangian density evaluated across the field's domain:

$$L = \tfrac{1}{2}\|v\|^2 - \Phi - \delta S + \eta\langle v, \nabla\Phi\rangle.$$

Each term earns its place by referring back to a chapter already written. The kinetic-like term, ½‖v‖², costs energy in proportion to how much flow is occurring at a point, penalizing motion for its own sake unless something justifies it. The potential-like term, $-\Phi$, rewards coherence: high-$\Phi$ regions, the settled, self-reinforcing configurations Chapter 16 called dynamic fixed points, sit at lower energy than fragmented ones, which is exactly the sense in which this book has always treated coherence as something the field tends toward rather than away from. The entropy term, $-\delta S$, rewards resolution over ambiguity, consistent with Chapter 8's insistence that entropy is a real cost the field carries rather than a neutral bookkeeping quantity. The coupling term, $\eta\langle$v, $\nabla\Phi\rangle$, rewards flow that moves toward increasing coherence and penalizes flow that moves away from it, which is the energetic expression of something Chapter 10 already described in different language: that attention and action follow the gradient of activated relevance, itself grounded in the field's own coherence structure. H, the field's total energy, is the integral of L across the whole domain, and it is worth being clear that nothing about this functional is new content. It is a single object that several chapters' worth of separately motivated tendencies turn out to be describing pieces of.

## 19.3 Intrinsic Relaxation as Gradient Descent

With H in hand, Chapter 13's $F_{\text{phys}}$ can finally be given more than a name. The field's intrinsic tendency to relax, to smooth coherence, to let entropy settle, to let flow subside once nothing continues to drive it, is exactly the negative functional gradient of H:

$$F_{\text{phys}}(M) = -\delta H / \delta M.$$

This is worth dwelling on, because it closes a claim Chapters 13 and 14 made without fully justifying it. Section 14.4 argued that $F_{\text{phys}}$ is close to admissible almost by construction, because it is built from the same discipline that defines C. It is now possible to say something stronger: $F_{\text{phys}}$ does not merely happen to respect C, it is literally the direction of steepest energy decrease, and to the extent C's hard constraints were chosen, in Chapter 18, to characterize exactly the configurations this energy functional favors, $F_{\text{phys}}$ moves the field toward lower energy and toward greater admissibility in the same motion, because for a well-constructed C these are close to the same thing. Gradient descent on H is not a mechanism this book is introducing for the first time in this chapter. It is what Chapter 13 was already calling $F_{\text{phys}}$, given its proper variational name.

## 19.4 Projection as Constrained Minimization

Chapter 14's argmin can now be recognized as a special case of the same underlying principle, restricted rather than unrestricted. Unconstrained gradient descent on H would move a proposal toward whatever configuration minimizes energy globally, ignoring C entirely. Constraint projection instead minimizes a different, more local functional, the squared distance to the proposal, subject to remaining inside C. These are not two unrelated computations that happen to share the word minimization. They are the same variational apparatus applied to two different objectives, one minimizing global energy without regard for admissibility, the other minimizing distance-from-proposal subject to admissibility, and a full treatment of constrained variational problems, of the kind this chapter will not develop in detail, shows that the second can itself be recovered as an unconstrained minimization of H plus a penalty term that grows without bound as a configuration moves outside C. Projection, in other words, is what gradient descent on H looks like once C's hard constraints are folded into the energy functional as an infinite wall rather than left as a separate condition checked afterward.

## 19.5 Fixed Points as Critical Points

This gives Chapter 16's fixed-point equation, and the stability question Chapter 16 explicitly deferred, their proper mathematical content. A fixed point $M^*$ satisfying $M^* = \Pi_C(M^* + \Delta\psi)$ is, in the variational picture, a critical point of H restricted to C: a configuration where the projected gradient vanishes, where there is no direction of further descent available without leaving the admissible region. A stable fixed point, in Chapter 16's informal sense, that small perturbations are absorbed back toward $M^*$ rather than amplified away from it, is precisely a local minimum of H restricted to C, a configuration surrounded on all admissible sides by higher energy, so that any small displacement costs energy the field's own dynamics will spend the next several updates recovering. An unstable fixed point, Chapter 16's precarious stack of loose stones, is a critical point that is not a local minimum, a saddle or local maximum where some admissible direction actually decreases energy further, so that the smallest nudge in the right direction sends the field's dynamics away from $M^*$ rather than back toward it. Chapter 16 promised this distinction would receive its proper formal footing in Part IV. This is that footing.

## 19.6 What Guarantees Existence

Chapter 18 closed by flagging a real gap: nothing established there guaranteed that a nearest point in C actually

exists for a given proposal, only that the question was well posed once C was given a genuine geometric definition. The variational picture developed in this chapter does not close that gap outright, but it says precisely what closing it would require. The standard tools of the calculus of variations show that a minimizer exists whenever the functional being minimized is bounded below and does not admit a sequence of ever-decreasing values escaping to infinity without converging, conditions usually called coercivity, and whenever the admissible set is closed under the limits of such sequences. Applied here, this becomes a concrete, checkable claim about H and C rather than an appeal to intuition: if H is coercive on C and C is closed in the appropriate sense, then a nearest admissible point exists for every proposal, and Chapter 14's argmin was never at risk of returning nothing. This book will not carry out that verification in the main text; it belongs, together with the fuller apparatus of Lagrange multipliers this chapter has gestured at without deriving, to the extended proofs this book's appendices reserve for exactly this purpose.

## 19.7 Looking Ahead

This chapter has treated the field's dynamics as though they were purely dissipative: energy decreases, proposals settle, fixed points are where descent finally stops. This is not the whole picture, and it is worth being honest about what it leaves out before moving on. The fountain from Chapter 16

was offered as a dynamic fixed point precisely because its water never stops moving even while its form remains stable, and pure gradient descent, taken alone, describes a field settling toward rest, not a field sustaining perpetual, patterned motion without ever losing energy to it. Something in this book's account needs to explain how a region can cycle indefinitely, water through a fountain, flow through a courtyard's drainage, without that cycling being merely energy draining slowly toward zero. That requires a genuinely different kind of dynamics, one where energy is conserved rather than dissipated, and Chapter 20 is where this book introduces it.

# Chapter 20

# Hamiltonian Semantic Fields

## 20.1 What Gradient Descent Cannot Explain

Chapter 19 left an honest debt. Gradient descent on H explains why a field settles, why coherence smooths, why entropy resolves once observation has earned the resolution, and why fixed points are exactly the configurations where this descent finally stops. It does not explain the fountain. A field that only ever flows downhill in energy, monotonically, has no way to sustain the fountain's continuous, patterned motion, because pure descent always terminates, eventually, at a minimum, and a fountain that terminated would not be a fountain anymore, only a puddle. Something in this framework has to account for motion that persists indefinitely without draining away, and gradient descent, taken alone, is structurally incapable of producing it.

## 20.2 From Energy Descent to Phase Space

The resolution is not to abandon the energy functional Chapter 19 developed, but to use it differently. Rather than

always flowing in the direction that decreases H fastest, a field can instead flow along a path that holds H constant, trading one component for another rather than shedding energy at all. This requires treating the field's variables in pairs, a coherence-like quantity and a flow-like quantity bound together the way position and momentum are bound together in ordinary mechanics: $\Phi$ plays the role of position, the configuration a point currently occupies, and v plays the role of momentum, the rate and direction at which that configuration is currently changing. A system governed this way does not sit still once it reaches a level set of constant energy. It moves along that level set indefinitely, $\Phi$ and v continuously exchanging value with one another, coherence briefly dipping as flow rises to compensate and rising again as flow subsides, forever, in principle, without ever losing energy to the exchange. This is what a fountain is doing, described in the vocabulary this book has been building since Chapter 7.

## 20.3 The Semantic Hamiltonian

The precise statement of this exchange is a pair of coupled equations, using the same energy functional H from Chapter 19 but now generating motion rather than merely measuring it:

$$\partial \Phi / \partial t = \delta H / \delta v, \qquad \partial v / \partial t = -\delta H / \delta \Phi.$$

These are Hamilton's equations, applied to the seman-

tic field rather than to a physical particle, and their defining property is conservation: a trajectory that satisfies them keeps H exactly constant along its path, by construction, since each variable's rate of change is defined in terms of the other's contribution to H rather than in terms of H's own gradient with respect to itself. Where Chapter 19's dynamics always pointed downhill, these equations point sideways, along the contour of whatever energy level the field currently occupies, and a field governed purely by this pair of equations oscillates rather than settles. The fountain's water is behaving exactly this way: its appearance and flow trade off against each other continuously, tracing out a stable, repeating pattern rather than draining toward the fountain's minimum-energy configuration, which would simply be still water at rest.

## 20.4 Reconciling Descent and Oscillation

It would be a mistake to treat this chapter as replacing Chapter 19 rather than completing it, and it is worth being precise about how the two coexist rather than compete. Chapter 17 already established that the field's components operate on separate clocks, fast components like appearance and flow cycling frequently, slow components like coherence and topological structure changing rarely. The Hamiltonian dynamics this chapter introduces govern the fast clock: within a single slow-timescale interval, z and v can trade energy back and forth through many complete cycles, tracing

the oscillatory behavior a fountain exhibits from one moment to the next. Chapter 19's gradient descent governs the slow clock: across many such fast-timescale cycles, whatever dissipation the system does experience, however small, accumulates and gradually pulls the slow-timescale coherence $\Phi$ toward one of its own minima, exactly the process Chapter 17 described as fast-timescale activity eventually forcing slow-timescale change. A fountain, on this reading, is not purely Hamiltonian and not purely dissipative. It is Hamiltonian at the fast timescale, cycling indefinitely without apparent loss, riding on top of a slow, dissipative drift that governs whether the fountain as a whole is, on the scale of years rather than moments, gradually settling, eroding, or otherwise moving toward a different overall configuration. The two chapters describe the same field at two different rates, not two competing accounts of the same rate.

## 20.5 What the Oscillation Conserves

It is worth naming, briefly, what exactly stays fixed while a fast-timescale trajectory cycles, because this book has been building toward an answer to this question since Chapter 16 first proposed that identity is a fixed point rather than a label. A trajectory obeying Hamilton's equations conserves more than H alone; it conserves the underlying structure, called symplectic in the branch of geometry this material draws on, that relates $\Phi$ and v to one another at every point along the path. This structure is what makes the trajectory

recognizably one continuous thing rather than a sequence of unrelated states that happen to share an energy value. The fountain's water, at any given instant, is not the water that occupied the fountain a minute ago, and its specific appearance is different from moment to moment in exactly the way Chapter 9 already described fast-timescale appearance as free to be. What persists is not the water but the trajectory the water is tracing, the conserved structure binding each instant's $\Phi$ and v to the next. This refines Chapter 16's claim rather than replacing it: identity, for a genuinely dynamic fixed point like a fountain, is not merely a fixed point of the write cycle in the sense Chapter 16 first stated. It is a closed, symplectically conserved orbit, a fixed pattern of exchange between coherence and flow, and Chapter 16's fixed-point equation turns out to describe the special, degenerate case of this chapter's oscillation where the orbit has shrunk to a single point.

## 20.6 Looking Ahead

Everything this book's dynamics have described since Chapter 12, writing, projection, commit, fixed points, energy descent, and now conservative oscillation, has been stated pointwise or, at most, across a single trajectory. None of it has yet asked how these local pictures fit together across the field's whole domain at once: whether a coherence structure that looks admissible in one region remains admissible when checked against its neighbors, or

whether the affordance-realizability constraint this book has postponed since Chapter 10 can finally be made precise. That is a question about global consistency rather than local dynamics, and it is where Chapter 21 turns next, delivering the sheaf-theoretic bridge this book promised the cue-coherence construction back when affordance was first given its own chapter.

# Chapter 21

# Sheaves, Field Coherence, and Global Consistency

## 21.1 Local Truth, Global Consistency

Every chapter since Chapter 12 has quietly restricted its attention to a single point, a single trajectory, or a single region considered in isolation. Writing, projection, commit, fixed points, energy descent, and Hamiltonian oscillation were all stated as though the courtyard's various points had nothing to do with one another beyond sharing the same field. This was never quite true, and this chapter is where the omission is repaired. A field defined over a domain $X$ is not merely a collection of independent stories, one per point; it is a single object, and something has to guarantee that what is locally admissible at each point actually agrees with its neighbors where their local pictures overlap. A wall that is locally coherent on its left half and locally coherent on its right half is not thereby coherent as a whole, unless the two halves also agree where they meet.

## 21.2 The Field-Coherence Sheaf

This is a local-to-global consistency question, and Chapter 10 already introduced the mathematical machinery suited to answering questions of this shape, applied there to cues rather than to the field itself. The same machinery applies here, over a different base. Let $\mathcal{M}$ be the field's configuration space, as developed in Chapter 18, and let the embedded structure of the field's domain, written $\mathrm{G} \hookrightarrow \mathcal{M}$ following the embedding this book's supporting technical material already develops in full, provide a covering: a way of decomposing X into overlapping patches, each small enough to be checked for local admissibility on its own.

The field-coherence sheaf, $F_{\text{field}}$, assigns to each patch of this cover the set of locally admissible field configurations on that patch, admissible in the sense Chapter 18's C already made precise, and assigns to each inclusion of a smaller patch into a larger one the restriction of a configuration from the larger patch to the smaller. The gluing question this sheaf poses is the one section 21.1 raised informally: given locally admissible configurations on each patch of a cover, agreeing wherever two patches overlap, do they assemble into a single globally admissible configuration on the whole domain? When they do, $F_{\text{field}}$ is said to glue on that cover, and the resulting global section is exactly what this book has meant, since Chapter 6, by a coherent world.

## 21.3 Hallucination as a Cohomological Obstruction

It is worth being precise about what happens when gluing fails, because this book's supporting technical material has, since its earliest chapters, described hallucination as a cohomological obstruction without yet earning the right to use that language. The failure of local data to assemble into a global section is measured, in the branch of mathematics this construction borrows from, by a cohomology class: informally, a record of exactly how the overlaps disagree, which cannot itself be resolved by any choice of global configuration because the disagreement is already baked into the local data before any assembly is attempted. Write $H^1(\mathcal{G}\infty)$ for this obstruction, evaluated over the full cover $\mathcal{G}\infty$ implied by the field's embedding. A field state for which $H^1(\mathcal{G}\infty) = 0$ is one whose local patches, wherever checked, agree on their overlaps and can be assembled without contradiction. A field state for which $H^1(\mathcal{G}\infty) \neq 0$ is one where no such assembly exists: two locally admissible, individually well-formed pieces of the field that simply cannot both be true of the same underlying world at once, a courtyard whose left half implies a doorway the right half's own local structure has no room for.

This gives hallucination a sharper meaning than this book has previously had available. It is not merely a proposal that violates a locally checkable rule, the kind of failure Chapter 14's projection already catches. It is a global

failure that no amount of local checking, patch by patch, will ever surface, because each patch, examined on its own, may be perfectly admissible. Only the overlap reveals the problem, and only cohomology, rather than any pointwise constraint from Chapters 8 through 20, is equipped to detect it.

## 21.4 From Cue Sheaves to World Sheaves

Chapter 10 promised that its cue-coherence sheaf, $F_{\text{cue}} : \text{CueCat}^{\text{op}} \to \text{Set}$, would eventually be connected to whatever field-coherence construction this book developed, rather than left to stand as an unrelated structure sharing only a family resemblance. That promise is kept here. The connection runs through observation, in exactly the sense Chapter 6 first introduced it and Chapter 9 gave concrete form to. A cue, recall, is a signal arising from the field, and this is not a coincidence to be papered over with a freestanding functor invented for this chapter's convenience; it is $O_\theta$, the observation operator this book has developed since Chapter 6, doing exactly the work this bridge requires. Write O for the map $O_\theta$ induces from CueCat into $\text{Patch}(\mathcal{M})$, sending each cue's context to the patch of the field that observation actually samples to produce it. This is not a new primitive. It is $O_\theta$, restricted to the role of relating cue-contexts to field-patches rather than fields to observations directly, and the direction this book adopts, cues arising downstream of patches rather than patches arising downstream of cues, fol-

lows from Chapter 9's own account of where a candidate $\hat{\mathbf{z}}$ comes from: an agent is cued by what it has already, at least partially, observed.

With O in hand, the bridge is a natural transformation,

$$\eta : F_{\text{cue}} \Rightarrow O^* F_{\text{field}},$$

where O*$F_{\text{field}}$ is the pullback of the field-coherence sheaf along O, assigning to each cue-context whatever local field-admissibility O maps that context onto. $\eta$ states the compatibility condition this book has been assuming informally since Chapter 10 without proving: an affordance activated at a cue is only genuinely available if its realization, once selected by Chapter 13's policy and proposed as a $\Delta\psi$, corresponds to an admissible transformation of the field patch that cue actually arose from. A ladder that cues climbing but whose realization would require a field transformation outside $C_S \cap C_z \cap C_R$ is not, on this account, a live affordance at all, however strongly the cue has activated it, and $\eta$ is precisely the map that catches this: an affordance in the image of $\eta$ is guaranteed field-realizable; an affordance outside its image is a cue-level illusion, coherent from the agent's side and unsupported from the world's.

## 21.5 Completing the Constraint Manifold

Chapter 18 left C's definition with an acknowledged placeholder, $C_A$, promised but not yet formalized. It can now be stated precisely: $C_A$ is the set of field states for which every

affordance activated by any cue arising from that state lies in the image of $\eta$, that is, every locally available action possibility corresponds to a genuinely admissible field transformation rather than a cue-level illusion of one. With $C_A$ in place, $\mathsf{C} = C_S \cap C_z \cap C_R \cap C_A$ is no longer missing a named member, though this book has been careful, since Chapter 18, not to claim the list is closed; Chapters 22 and 23 will each have occasion to ask whether further conditions belong in the intersection before this book is willing to call C complete.

## 21.6 Which Maps Preserve Admissibility

Chapter 18 also deferred a question about domain transport: when a topology repair or identity restructuring changes the field's domain from X to some new X′, which maps $\varphi$ relating the two are actually entitled to carry C's hard constraints across, and which are not. The sheaf-theoretic apparatus this chapter has developed answers this precisely. $\varphi$ is eligible exactly when it is a morphism of sites: when it sends covers of X to covers of X′ compatibly with restriction, so that $F_{\text{field}}$'s gluing condition on the old domain transports to a well-posed gluing condition on the new one rather than to a collection of patches that no longer cohere into any sensible covering at all. A $\varphi$ that scrambles the cover, merging patches that should remain distinct or splitting one patch into pieces with no coherent restriction maps between them, is not eligible, however smooth or

structure-preserving it may appear from outside the sheaf's own bookkeeping, and any constraint transported along such a $\varphi$ would arrive already broken. This is the condition Chapter 18 promised would be supplied here, and it was never separable from the sheaf construction this chapter exists to develop; transport and gluing were always the same requirement viewed from two directions.

## 21.7 Looking Ahead

This chapter has given the field's local-to-global consistency a precise treatment, named hallucination as the specific cohomological failure it always informally was, completed the bridge between cue-level and field-level coherence that Chapter 10 promised, and supplied C with its fourth named member. What it has not asked is what, if anything, stays exactly conserved as the field evolves under all of this machinery at once, beyond the energy this book's variational chapters have already discussed. Identity, residue, and the coherence a committed configuration carries forward all seem, informally, like they ought to be conserved in some sense stronger than merely persisting at a fixed point. Chapter 22 asks what conservation actually means here, and what, precisely, this framework is entitled to claim never simply disappears.

# Chapter 22

# Conservation Laws

## 22.1 What Should Never Disappear

Chapter 18 answered a question about single instants: which field states are admissible. Chapters 19 and 20 answered a question about motion: how an admissible state evolves, whether by descent or by conservative oscillation. Neither of these, on its own, asks the question this chapter takes up. Admissibility is a property C either grants or withholds at a given moment; it says nothing about whether some quantity, once present, is guaranteed to remain present as an admissible trajectory unfolds. Conservation is that further claim, and it is a claim about trajectories, not about states considered one at a time. A single snapshot of the field conserves nothing, in the sense this chapter means the word; only a committed sequence of updates, an actual history rather than a single instant of it, can be said to conserve anything at all.

Chapter 20 already supplied one example, in passing, without naming it as an instance of a general pattern. A Hamiltonian trajectory conserves H exactly, by construction, and conserves the symplectic structure binding $\Phi$ and

v together at every point along its path. This chapter generalizes that observation, and asks what else, across the whole apparatus this book has built since Chapter 8, deserves to be called conserved rather than merely constrained.

## 22.2 Symmetry and Conservation

The relevant tool is Noether's theorem, applied here to the semantic Lagrangian introduced in Chapter 19. Its content, stated informally, is this: every continuous symmetry of L, every transformation of the field's variables that leaves L's value unchanged, corresponds to a quantity that stays exactly constant along any trajectory satisfying the equations of motion Chapters 19 and 20 developed. A symmetry is not a coincidence to be discovered by inspection alone; it is a structural feature of L, and once identified, it hands the theory a conservation law for free, without needing to verify the conservation independently by tracking the quantity through every possible trajectory.

## 22.3 Three Conserved Quantities

Three symmetries of this book's framework are worth making explicit, because each corresponds to a conservation law that gives earlier chapters' informal claims a precise form.

The first is time-translation symmetry: nothing about L depends on which update index a trajectory happens to be labeled as starting from, only on the field's actual configu-

ration and its rate of change. This is the symmetry Chapter 20 already exploited without naming it, and its conserved quantity is H itself, energy in this book's sense of the word, exactly constant along any trajectory obeying Hamilton's equations. Chapter 16's identity, refined in Chapter 20 into a persistent symplectic orbit, is now recognizable as this conservation law's signature: an object persists exactly because its trajectory conserves the quantity time-translation symmetry guarantees, and ceases to persist, in this book's sense, exactly when some external event forces it off that conserved orbit entirely.

The second is a domain-relabeling symmetry, closely related to Chapter 15's account of commutative updates. Two commitments to disjoint regions of the field, committed in either order, leave the field's total causal content unchanged, and this reordering is a genuine symmetry of the write cycle: nothing about which commutative update happened first is visible in the resulting history. Its conserved quantity is the total accumulated residue across the field, in the specific sense that no committed event's contribution to R can be created or destroyed by choosing a different, equally valid ordering of commutative commits, only relabeled in when it is said to have occurred. Residue can still grow, every genuine commit adds to it, but it cannot grow or shrink merely as an artifact of reordering, and this is the conservation law underwriting Chapter 15's earlier claim that commutative updates may commit in any order without affecting the final result.

The third is a translation symmetry across the field's domain itself, closest to the mass-consistency invariant this book's supporting technical material lists among its hard constraints. Total coherence, integrated across the whole domain, is conserved except at explicitly licensed sources and sinks: invention, Chapter 9's legitimate act of resolving genuine uncertainty into settled appearance, is a source, adding coherence where entropy previously left none; refusal, Chapter 15's deliberate rejection of a proposal, and unrecovered decay are sinks, removing it. Between these events, coherence does not simply appear or vanish at a point; it moves, in the sense Chapter 7's vector field $\mathbf{v}$ was always meant to describe, from where it is to where flow carries it, and the conservation law states precisely that any local increase in $\Phi$ must be traceable either to a licensed source or to inflow from a neighboring region, never to nothing at all.

## 22.4 A Worked Example: Three Ways a History Could Lie

It is worth grounding section 22.3's three laws in a single scenario, because stated abstractly they can look like one diagnostic wearing three names rather than three genuinely independent checks. Consider a courtyard's committed history across three consecutive updates, the fountain, the bench, the wall's crack, and consider three different ways a corrupted or fabricated version of that history could

fail, each caught by a different one of the three laws and by none of the others.

Suppose the fountain's water is recorded as moving measurably faster in the third commit than the first, with no proposed forcing term, no wind, no mechanical input, anywhere in the intervening proposals to account for the increase. This is a violation of the first law: H is not constant along the trajectory, and nothing licensed the change. Chapter 21's spatial check would not catch this at all, since each individual commit, examined on its own, is a perfectly well-glued, locally admissible snapshot; the failure is visible only in the comparison across commits, exactly where a conservation law and not a gluing condition is the relevant tool.

Suppose instead the fountain's water is unremarkable, but the courtyard's commit log records two spatially disjoint, genuinely commutative events, a bench moved on one side, a crack inspected on the other, and the total residue accumulated depends on which order the log claims they occurred in, even though Chapter 15 already established that commutative events must be order-independent in their effect on the field. This is a violation of the second law, and it points to a different kind of corruption than the first: not a physically implausible jump in energy, but a commit log that is not actually a faithful causal-DAG record, residue being relabeled rather than merely reordered, which the first law's energy bookkeeping would never notice.

Suppose finally that both of these check out, energy is

constant, residue is order-independent, and yet the courtyard's total coherence, $\Phi$, is recorded as higher after the third commit than before it, with no invention event in the residue log and no inflow recorded in the vector field v from any neighboring region. This is a violation of the third law alone, coherence appearing from nowhere, and it is the one most directly resembling old-fashioned hallucination: certainty materializing without an earning event anywhere in the history to account for it.

Each of these three corruptions could occur entirely independently of the other two, and any one of them could occur in a history that passes every one of Chapter 21's spatial checks at every individual instant, since spatial gluing says nothing about what should stay constant, or change only through accounted means, across the sequence. This is the concrete cash value of insisting on three separate conservation laws rather than one: a history that lies about energy, a history that lies about its own causal ordering, and a history that lies about where coherence came from are three different lies, and this framework needs all three checks, run together, to catch all three.

## 22.5 Conservation Along Orbits, Not at Points

It is worth restating plainly what these three conservation laws are and are not claims about, because the temptation to read them as constraints on individual states, in the man-

ner of Chapter 18's C, would misplace them entirely. C asks whether a given $M_t$, considered alone, is admissible. These conservation laws ask whether a given trajectory, a committed sequence $M_t$, $M_{t+1}$, $M_{t+2}$, and so on, holds H, total residue, and total licensed coherence constant, or changing only through accounted sources and sinks, across the whole sequence. A single $M_t$ cannot violate a conservation law by itself, in the same way a single frame of a film cannot violate the law of motion governing the film's characters; violation is only visible across a sequence, exactly as Chapter 20 already implied when it redescribed identity as a property of an orbit rather than a point. This chapter's three conservation laws are, in this sense, elaborations of that same redescription, applied to residue and coherence rather than only to the symplectic structure Chapter 20 introduced first.

## 22.6 What Conservation Buys the Theory

This gives the framework a second, genuinely different kind of diagnostic from the one Chapter 21 supplied. Chapter 21's cohomological obstruction catches a spatial failure: local patches, checked at a single instant, that cannot be glued into one coherent whole. Conservation catches a temporal failure: a sequence of states, each individually admissible and each individually well-glued in Chapter 21's sense, whose transitions nonetheless violate what should have stayed constant along the way, residue appearing with no committed event to trace it to, coherence increasing at a

point with no source or inflow to license it. A hallucinated history can, in principle, pass every one of Chapter 21's spatial checks at every individual instant, and still be caught by this chapter's temporal ones, because gluing correctly at each moment says nothing about whether the moments, taken together as a trajectory, actually obey the conservation laws that any genuine history is bound by. Spatial coherence and temporal conservation are, in this framework, two independent tests, and a fully trustworthy history has to pass both.

## 22.7 Looking Ahead

This chapter has asked what a single world's trajectory is obligated to conserve as it evolves. It has not asked anything about how one world relates to another, whether two different persistent fields, built from different domains or different histories entirely, can be meaningfully compared, translated into one another, or recognized as instances of the same underlying kind of structure. That is a question about relationships between worlds rather than properties of any one world's trajectory, and it is where Chapter 23 turns next.

# Chapter 23

# Category Theory

## 23.1 When Are Two Worlds "The Same"?

Every chapter since Chapter 6 has concerned itself with a single persistent field, its internal structure, its dynamics, its conservation laws. Nothing so far has asked a question that becomes unavoidable the moment more than one world is admitted to exist: when should two different persistent fields be considered instances of the same kind of thing, rather than merely two unrelated fields that happen to share a vocabulary? Two courtyards, simulated independently, each with their own fountain, are obviously not the same field in the sense Chapters 6 through 22 have used the word; their specific water, their specific residue, their specific history all differ. And yet there is a real sense in which both are fountains, in which whatever architecture the first courtyard's fountain instantiates, the second courtyard's fountain instantiates as well. This chapter gives that sense of sameness a precise meaning.

## 23.2 The Category of Worlds

Consistent with the refinement Chapters 20 and 22 have already pressed for, treat the objects of interest not as individual field states but as admissible trajectories, the orbits Chapter 20 introduced and Chapter 22 insisted conservation laws be stated over. Call this category $\mathcal{W}$. Its objects are admissible trajectories through the field's configuration space, each one a complete history obeying the write cycle, the constraint manifold, and the conservation laws Part IV has developed. Its morphisms are structure-preserving maps between trajectories: a morphism from one orbit to another is a map that sends admissible states to admissible states, respects the decomposition into $\Phi$, v, S, R, and z, and commutes with the update cycle, so that mapping first and then evolving gives the same result as evolving first and then mapping. This last condition is what makes the map a genuine morphism of trajectories rather than a mere pointwise relabeling; it guarantees that whatever relationship holds between two worlds at one moment continues to hold as both worlds evolve.

## 23.3 Functors as Faithful Translations

This book has already used a functor without naming it as one. Chapter 21's observation-induced map, O, sending cue-contexts to field-patches, is exactly this: a structure-preserving translation from one category, CueCat, into an-

other, the category of patches over $\mathcal{M}$. Naming it a functor now makes explicit a requirement that section 21.4 used implicitly without stating: O must respect composition and identity, so that observing a cue's context and then restricting to a smaller sub-context gives the same patch as restricting first and then observing, and so that observing the trivial, most general cue-context returns the field's own identity patch rather than some distortion of it. Functoriality, in other words, is the discipline that keeps a translation between categories from silently smuggling in inconsistency, and Chapter 21's bridge, $\eta$, was only ever well-posed because O was, whether or not the chapter said so at the time, a genuine functor rather than an arbitrary assignment.

With $\mathcal{W}$ in hand, the question this chapter opened with has a precise answer. Two trajectories are isomorphic in $\mathcal{W}$ when there exist structure-preserving morphisms in both directions between them whose composition, either way, is the identity morphism on each. This is the sense in which the two courtyards' fountains, despite occupying entirely different domains, holding entirely different specific water, and carrying entirely different accumulated residue, can be recognized as the same kind of thing: not because their states literally coincide, which they do not, but because a structure-preserving correspondence exists between their trajectories, matching each fountain's coherence-flow oscillation to the other's, each fountain's conserved quantities to the other's, in a way that survives both trajectories' continued evolution. Sameness of type, in this framework, is iso-

morphism in $\mathcal{W}$, and it is worth noting how much weight this places on the word type: two worlds sharing a type are not required to share any content at all, only the architecture that content is organized by.

A simpler, non-field example is worth flagging here, even though the fit is not exact and the full comparison waits until Chapter 28. The Arabic script's twenty-eight letters have been given two distinct, historically real orderings, the ancient Abjad sequence and the later Hijā'ī sequence, each imposing a different structure on the same underlying set of letters for a different purpose. This is not quite an isomorphism between two objects in the sense $\mathcal{W}$ makes precise; it is closer to two different structures sitting over one shared underlying set, the same letters admitting more than one legitimate organization. The disanalogy matters as much as the resemblance, and Chapter 28 works through both.

## 23.4 Local Structure Over Global Worlds

It is worth returning to Chapter 21's local-to-global construction once more, now that categorical language is available to describe it properly. The assignment sending each world in $\mathcal{W}$ to its own cover of local patches, and each patch to its own local field-coherence data, has the shape of a fibration: $\mathcal{W}$ sits as a base category, and above each object of $\mathcal{W}$ sits a fiber, the category of that world's own local patches and their gluing relationships. A morphism between two

worlds in $\mathcal{W}$ lifts, under this fibration, to a corresponding relationship between their fibers, matching one world's patches to the other's in a way compatible with both worlds' own gluing conditions. This is not a new construction invented for this chapter. It is Chapter 21's local-global relationship, restated in the vocabulary this chapter has been developing, and restating it this way clarifies something Chapter 21 could only gesture at: two worlds related by an isomorphism in $\mathcal{W}$ are guaranteed, by the fibration's own structure, to have isomorphic local pictures as well, patch for patch, which is precisely what should be true if isomorphism in $\mathcal{W}$ is to deserve the name sameness of type.

## 23.5 What Category Theory Cannot Tell You

It is worth being honest about the limits of what this chapter has built, because the temptation to treat isomorphism as though it settled every question of comparison between worlds would overreach what category theory actually offers. $\mathcal{W}$ answers a binary question: related by a structure-preserving correspondence, or not. It has nothing to say about worlds that are not isomorphic but are nonetheless similar, close in whatever sense a reader might mean by that word, two fountains that are almost but not quite architecturally identical, one slightly more damped, one slightly more energetic, related by no isomorphism at all and yet clearly not unrelated either. Category theory, by its nature,

does not traffic in degree. A morphism either exists or it does not; two objects either are or are not isomorphic. Nothing in this chapter's machinery can say how far apart two non-isomorphic worlds actually are, and this is not an oversight to be patched within category theory's own vocabulary. It is a genuinely different question, requiring genuinely different mathematics, and answering it is Chapter 24's task rather than an extension of this one.

## 23.6 Looking Ahead

This chapter has given this book a precise sense in which two different worlds can be recognized as the same kind of thing, built from admissible trajectories, structure-preserving morphisms, and the isomorphisms between them, and has shown that this categorical structure reproduces, rather than replaces, the local-to-global relationship Chapter 21 already established. What it cannot do is measure. Chapter 24 takes up the question this chapter deliberately left open: not whether two worlds are related, but how far apart they are when they are not, and how close, when they are not quite the same.

# Chapter 24

# Information Geometry

## 24.1 Between Isomorphic and Unrelated

Chapter 23 closed by admitting what its own machinery could not do. Isomorphism in $\mathcal{W}$ is a binary verdict, related or not, and most pairs of worlds a reader might actually want to compare fall into neither category cleanly. Two fountains built from slightly different Lagrangian parameters, one a touch more damped than the other, are not isomorphic in Chapter 23's exact sense, and yet calling them simply unrelated discards everything genuinely similar about them. This chapter's task is to replace that binary verdict with a number: not whether two worlds are related, but how far apart they are when they are not identical, and how close, when they are almost the same.

## 24.2 Why Comparing Configurations Directly Does Not Work

The most obvious approach fails quickly enough to be worth ruling out explicitly. Chapter 18 already equipped the field's configuration space, $\mathcal{M}$, with a metric g, and it

might seem that comparing two worlds is simply a matter of measuring the g-distance between their states. This does not work, for a reason that becomes obvious once stated: g measures distance between two configurations of the same domain X, points within a single world's own configuration space. Two different worlds, in general, are not defined over the same domain at all, and there is no canonical way to line up one courtyard's points with another's before any distance between them can even be asked for. Direct geometric comparison presupposes exactly the correspondence this chapter is trying to establish, and cannot be used to establish it.

## 24.3 Entropy as a Distribution, Worlds as Beliefs

The more productive approach abandons direct comparison of raw configurations and compares something more abstract instead: what each world believes about itself. Recall $L_t(x)$, the live possibility set introduced in Chapter 8, the space of outcomes a point has not yet been forced to choose among. At any point where entropy remains nonzero, $L_t(x)$ is naturally read as inducing a distribution over live outcomes, weighted by how strongly each remains admissible given everything the field currently holds. A world, at any given moment, is then not merely a configuration but a belief, in the technical sense this book borrows from the branch of geometry concerned with spaces of

probability distributions: the entirety of its entropy structure, across every point still under active uncertainty, describes a point in a much larger space, the space of possible belief states a persistent field could hold. Comparing two worlds becomes comparing two points in this belief space instead of two points in $\mathcal{M}$ directly, and unlike raw configurations, belief states do not require the two worlds to share a domain at all, only a common vocabulary of what remains uncertain and by how much.

## 24.4 The Information Metric

This space of belief states carries its own natural notion of distance, arising not from any external choice but from the distributions themselves: two belief states are close when they are difficult to distinguish from a small number of observations, and far apart when a single observation would reliably tell them apart. The precise quantity this book adopts, without deriving it from scratch, is a Fisher-information-style metric, measuring the local sensitivity of a belief state's distribution to small changes in whatever parameters describe it, and its natural companion, a divergence between two belief states measuring how much is lost, in an information-theoretic sense, by approximating one world's entropy structure with the other's. This divergence is generally asymmetric, treating one world as the reference and the other as the approximation, and this asymmetry is not a defect to be corrected; approximating a richly

resolved world with a barely-resolved one loses more than the reverse, and the metric is right to say so. A symmetrized version, averaging the divergence taken in both directions, gives the genuine distance this chapter has been building toward: a single number, computable in principle at any point where both worlds carry active uncertainty, measuring how far apart their beliefs about that point actually are.

## 24.5 From Points to Trajectories

Consistent with the orbit-primacy this book has insisted on since Chapter 20, a distance defined only at a single moment is not yet a distance between worlds in the sense this chapter needs. Two worlds are properly compared not at one instant but across their whole admissible trajectories, and the natural extension accumulates the pointwise informational distance along corresponding stretches of each world's history: integrated, moment by moment, over as much of each trajectory as can be meaningfully aligned. Two fountains whose instantaneous belief states are nearly identical at every corresponding moment of their respective histories are close in this trajectory-level sense; two fountains that happen to coincide at one instant but diverge sharply before and after are not, however similar that single shared instant might have made them look in isolation. This is the same discipline Chapter 22 insisted on for conservation, applied here to comparison: a claim about one instant is not yet a claim about the orbit, and this chapter's distance, properly stated,

is a distance between orbits rather than between snapshots.

## 24.6 Near Misses and Family Resemblance

This finally answers the question Chapter 23 left open. Two fountains related by no isomorphism in $\mathcal{W}$, one slightly more damped than the other, now receive a real, computable distance rather than a bare declaration of unrelatedness, and a small distance is precisely what this book means by family resemblance: not sameness of type in Chapter 23's exact sense, but proximity of belief across corresponding trajectories, close enough that mistaking one for the other would cost little in the information-theoretic sense this chapter has made precise. Isomorphism and distance are not competing accounts of similarity. They are complementary: isomorphism asks whether two worlds share an exact architecture; distance asks, when they do not, how expensive the difference actually is.

## 24.7 Closing Part IV

This chapter completes the ascent Part IV has been climbing since Chapter 18. Chapter 18 asked which states are admissible, and gave the constraint manifold C its geometric home. Chapter 19 asked how admissible states evolve, and gave that evolution an energy functional to descend. Chapter 20 asked what allows persistence without stasis, and gave identity its home as a conserved orbit rather than a

static point. Chapter 21 asked how local truths become one coherent world, and gave that question a sheaf-theoretic answer, naming hallucination's spatial failure precisely for the first time. Chapter 22 asked what a genuine history is obligated to preserve, and gave conservation its own trajectory-level meaning, naming hallucination's temporal failure to complete the diagnosis Chapter 21 began. Chapter 23 asked when two different worlds share an architecture, and Chapter 24 has asked, for the much more common case where they do not share one exactly, how far apart they actually are.

Together, these seven chapters have supplied, formally, everything Parts II and III used only informally: C, $\Pi_C$, argmin, $F_{\text{phys}}$, fixed points, gluing, conservation, and now comparison. Nothing in the mathematics remains owed. What remains is to build it, and Part V, beginning with Chapter 25, turns from what this framework is to how it actually runs, as software, on real machines, for real persistent worlds.

# Part V

# Engineering the World Engine

# Chapter 25

# System Architecture

## 25.1 From Mathematics to Software

Part IV gave this book's core cycle, observe, propose, project, commit, a complete mathematical treatment: a configuration manifold with a metric, an energy functional, a constraint manifold defined as an intersection of admissibility conditions, a sheaf-theoretic account of global consistency, conservation laws, and a way of comparing worlds to one another. None of this, on its own, runs. This Part asks what has to exist, as actual software, for the cycle Chapters 12 through 15 described to execute against a real, evolving world rather than remain a description of what a correct execution would look like. The translation from operator to component is not mechanical. Several of the boundaries this chapter draws are architectural decisions in their own right, chosen to preserve discipline this book has argued for since Part III, not merely convenient ways of organizing code.

## 25.2 The Core Loop as a Scheduler

Chapter 17 established that the field's components do not all update at the same rate: appearance and flow are fast, coherence and topological structure are slow, and residue inherits whatever rate the events it accumulates actually occur at. A real implementation cannot treat this as a mathematical nicety to be waved at; it has to be an actual execution policy. The component responsible for this is the scheduler, and its job is narrow but essential: at each tick of whatever clock the fast components run on, dispatch an update cycle, propose, project, commit, to the fast-timescale components at the region currently under consideration; at a coarser interval, dispatch the same cycle to the slow-timescale components. Nothing about the write cycle itself changes between these two dispatches, exactly as Chapter 17 insisted; only the rate at which the scheduler chooses to invoke it differs. A naive implementation that runs every component's update cycle on a single global tick will either waste enormous computation re-checking slow components that have not meaningfully changed, or, worse, let fast components starve while waiting for an update cycle sized for coherence and topology rather than for water moving through a fountain. The scheduler exists specifically to prevent this, and its correctness is measured against Chapter 17's timescale separation directly.

## 25.3 The World Database

$M_t$ has to live somewhere queryable, and the component responsible for holding it, this chapter will call the world database, without yet specifying how it actually stores anything internally; that is Chapter 26's task, kept deliberately separate from this one. What matters here is the world database's role rather than its implementation: it is the single, authoritative location holding the field's current committed state, decomposed per Chapter 7 into Φ, v, S, R, and z at every point of whatever cover the current implementation uses, consistent with Chapter 21's patch structure. Every other component in this chapter's architecture either reads from it or, through a narrow and specific interface, writes to it, and no component is permitted to hold a private, divergent copy of any part of $M_t$ that other components cannot see. This single-authoritative-copy discipline is what makes Chapter 15's account of commit meaningful in software terms: a commit is not complete until it has been written to this one place, and nothing counts as part of the field's history until the world database says so.

## 25.4 The Proposal and Projection Services

Chapters 13 and 14 kept proposal and projection as conceptually separate operators for a specific reason, argued at length in section 13.4: a proposal generator that has fully internalized every admissibility constraint loses the expres-

sivity that makes it worth having, and a projection stage that trusts proposals unconditionally reintroduces exactly the drift Chapter 12 warned against. This chapter turns that conceptual separation into an architectural one. The proposal service, whatever neural machinery actually implements $F_\psi$ and Chapter 13's relevance-based selection, is given read access to the world database and to whatever cue and affordance information Chapter 10's construction requires, and produces candidate $\Delta\psi$ values. Critically, it is given no write access to the world database at all. It cannot commit anything, however confident its output. The projection service receives candidates from the proposal service, has its own read access to the world database and to whatever representation of C the implementation maintains, and computes $\Pi_C$ against them; it, in turn, has no authority to commit either, only to produce a projected Y. Only a third component, responsible for Chapter 15's arbitration among simultaneous proposals and for the actual commit operation, has write access to the world database. This three-way separation is not merely tidy. It means the expressivity-without-authority discipline Chapter 13 argued for in the abstract is enforced by the software's own permission structure, not merely by the training or good behavior of whatever model implements the proposal service. A proposal service that hallucinates wildly is, by construction, incapable of writing that hallucination directly into the world; it can only ever offer a candidate for something else to check.

## 25.5 The Renderer as a Client, Not a Component

It is tempting to treat rendering, Chapter 9's concrete instance of $O_\theta$, as another stage in the same pipeline as proposal and projection, sitting downstream of commit the way commit sits downstream of projection. This chapter argues against that framing. A renderer, in this architecture, is a client of the world database, not a stage of the core loop: it issues reads, using whatever query the observation operator requires, and produces an observation for whatever observer it serves, but it neither participates in nor is required by the write cycle itself. This distinction matters once Chapter 6's world-observer-generator separation is taken seriously as a design constraint rather than only a philosophical one. Multiple renderers, serving multiple observers, human players, other AI agents, diagnostic tools, can attach to the same world database simultaneously, each issuing its own reads, without any of them needing to coordinate with the core proposal-projection-commit loop or with one another. The world continues to evolve, through its own write cycle, whether or not any renderer happens to be attached to it at a given moment, which is precisely the property Chapter 6 insisted a genuine world needed to have.

## 25.6 A Reference Architecture

It is worth assembling these pieces into a single picture before the next several chapters each zoom into one part of it. At the center sits the world database, holding $M_t$. A scheduler dispatches update cycles to it at the appropriate rate for each of the field's fast and slow components. Each cycle begins with the proposal service, reading the world database and producing candidate $\Delta\psi$ values; continues through the projection service, reading the world database and the constraint representation to compute Y; and completes at the commit and residue ledger, which arbitrates among simultaneous proposals per Chapter 15, writes accepted results back to the world database, and updates accumulated residue. Independently of this loop, any number of renderers attach to the world database as clients, issuing reads and producing observations for whatever observers they serve, entirely decoupled from the write cycle's own rhythm. Chapter 29's multiplayer treatment will extend this picture to multiple simultaneous proposal streams arriving from different agents; Chapter 30's scaling treatment will extend it to a world database too large to hold on a single machine. Neither extension changes the shape drawn here. Both extend it.

## 25.7 Looking Ahead

This chapter has treated the world database as a component with a clear role and no specified internals: something that holds $M_t$, accepts reads from proposal services and renderers, and accepts writes only from the commit stage. Chapter 26 opens that box, asking what actually holding a persistent, spatially extended, five-component semantic field requires, in terms of storage format, compression, and the practical realities of a structure too large to keep entirely in memory at once.

# Chapter 26

# Storage

## 26.1 What the World Database Actually Has to Hold

Chapter 25 left the world database as a black box: something that holds $M_t$, accepts reads, and accepts writes only from the commit stage. It is worth being precise about what that actually means before designing anything, because $M_t$ is not a single object of fixed size. It is five components, $\Phi$, v, S, R, and z, each potentially defined over an unbounded domain, and Chapter 8 already established that most of that domain, at any given moment, is not densely resolved at all. A naive implementation, allocating storage for a dense array of all five components across the entire domain whether or not anything has actually been observed there, would waste an amount of space bounded only by how large the domain is permitted to grow, most of it spent representing regions that hold nothing more specific than an unresolved live possibility set. This chapter's task is to design storage that reflects the field's actual structure rather than a worst-case assumption about it.

## 26.2 Sparse by Construction

The entropy constraint from Chapter 8, together with the invention-to-memory discipline from Chapter 9, together imply something useful for storage design specifically: a region only acquires dense, specific, low-entropy content once it has actually been observed. Everything else remains, honestly, at high entropy, represented by $L_t(x)$ rather than by a committed z. This is exactly the structure a sparse storage scheme is built to exploit. Rather than a dense array, the world database should use a hierarchical, sparse structure, an octree or quadtree depending on the domain's dimensionality, that allocates storage only for regions actually holding resolved content, with unresolved regions represented implicitly, by their absence from the structure, rather than explicitly, by a stored placeholder. A courtyard's unobserved far corner costs the storage system nothing beyond whatever coarse, cheap representation of $L_t(x)$ is useful for the entropy constraint's own bookkeeping, until the moment it is actually observed and Chapter 9's invention commits real content there.

## 26.3 Locality and Space-Filling Curves

Sparse allocation solves the size problem. It does not, by itself, solve a related problem: a scheme that allocates storage sparsely can still scatter logically nearby points across physically distant storage locations, which is costly for ex-

actly the kind of patch-based access Chapter 21's cover-based construction requires, reading a whole neighborhood's worth of points together to check a gluing condition or perform a projection. The standard technique for keeping spatial neighbors close in storage as well as in the field's own domain is a space-filling curve, most commonly a Hilbert curve, which linearizes a multi-dimensional domain into a one-dimensional storage order while preserving locality far better than a naive row-major traversal: two points close together in the field's domain are, with high probability, also close together in the resulting storage order. This matters concretely for this book's own machinery. Chapter 21's patches, the units over which local admissibility is checked and gluing conditions evaluated, correspond naturally to contiguous ranges along a Hilbert-ordered storage layout, so that loading a patch means reading one contiguous span of storage rather than gathering scattered fragments from across the whole structure.

Purpose-driven reordering of this kind is not a novelty invented for computing. The Arabic script offers a centuries-old precedent: alongside the ancient Abjad order, which preserves inherited numerical and mnemonic structure and remains in use for numerals and chronology, a separate Hijāʾī order was later adopted, regrouping the same twenty-eight letters by shared shape purely to ease handwriting instruction and lookup, without displacing Abjad's continued use where its own structure is still what is needed. A Hilbert-ordered storage layout is doing

something with the same shape: keeping every point the field's domain contains while imposing a second, purpose-built ordering optimized for an access pattern the domain's own coordinates were never designed to serve. Chapter 28 develops the comparison in full.

## 26.4 Entropy-Dependent Compression

This gives entropy, Chapter 8's S, a genuine downstream engineering role beyond the constraint it was originally introduced to enforce. A region's entropy is a direct, principled signal for how much storage that region deserves: high-S regions need only enough representation to describe $L_t(x)$ at whatever coarse resolution the entropy constraint's own bookkeeping requires, while low-S regions, fully resolved, earn full-resolution storage for their committed z and whatever residue R has accumulated there. Compression ratio, in this scheme, is not a separate parameter tuned independently of the field's own mathematics; it is a direct function of entropy, high where entropy is high and low where entropy is low, which means the storage system's compression behavior is automatically doing the right thing wherever Chapter 8's constraint is already being honestly enforced elsewhere in the system.

## 26.5 Different Strategies for Different Components

It is worth being explicit that the five components do not warrant identical storage treatment, because they do not share the same character. Appearance, z, is dense, image-like data wherever it is resolved, and benefits from whatever general-purpose compression suits that kind of content. Residue, R, is not a full event log, and Chapter 11 was explicit about why: it is a compressed, structural trace, and its storage representation should reflect that directly, closer to a compact causal-graph fragment than to a growing append-only journal, exactly matching the $\oplus$-accumulated object Chapter 11 described rather than anything larger. $\Phi$, v, and S, given Chapter 17's fast-slow distinction, can generally tolerate coarser storage resolution and less frequent versioning than z: a coherence field that changes on a slow clock does not need the same fine-grained temporal snapshotting a fast-changing appearance field does, and storing it as though it did would waste space tracking versions of a quantity that, across most of those versions, has not actually changed.

## 26.6 Streaming and Progressive Loading

A world database of any real size will not fit entirely in memory at once, and the sparse, patch-organized structure this chapter has built turns out to make this manage-

able rather than merely tolerable. Because Chapter 21's patches are already the natural unit of both mathematical admissibility-checking and Hilbert-ordered storage locality, they are also the natural unit of loading and unloading: as an observer moves through the domain, patches near the observer's current position are streamed into memory, and patches far from any current observer are evicted, without this eviction meaning anything has been lost, since the world database, not memory, remains the authoritative store. This chapter will not develop the full distributed-systems treatment of a world database too large for any single machine; that belongs to Chapter 30. What matters here is that streaming is not a bolt-on feature requiring a separate data structure. It falls directly out of organizing storage around the same patches Chapter 21 already needed for entirely different, purely mathematical reasons.

## 26.7 A Storage Interface

It is worth gesturing, briefly, at what a concrete implementation of this chapter's design would actually expose to the rest of the system, without providing a full specification. A storage backend, in this architecture, needs to implement a narrow interface: given a patch identifier, retrieve its current committed content across whatever subset of $\Phi$, v, S, R, and z the caller needs; given a patch identifier and a committed update, write it, respecting whatever compression and versioning policy section 26.4 and 26.5 established for

that component; and given a region of the domain, enumerate which patches currently exist within it, distinguishing resolved patches from the implicitly-represented unresolved space around them. Everything this chapter has described, sparsity, Hilbert ordering, entropy-dependent compression, component-specific strategies, streaming, lives behind this interface rather than being exposed to the proposal, projection, and commit services from Chapter 25, each of which only ever needs to know that the world database can answer these three kinds of request, not how it answers them.

## 26.8 Looking Ahead

This chapter has given the world database a real storage design: sparse, entropy-aware, organized for locality, and streamable. What it has not addressed is what happens on the other side of a read, once a patch's content has actually been retrieved by a client. Chapter 27 takes up rendering directly, asking how the appearance a storage system hands back actually becomes an observation an agent can use.

# Chapter 27

# Rendering

## 27.1 What a Renderer Actually Does

Chapter 25 established that a renderer is a client of the world database, not a stage of the core write loop, and Chapter 9 established that the observation operator $O_\theta$ actually has two directions: forward, reading committed appearance into a perceivable output, and inverse, proposing a candidate appearance for whatever a query touches that has not yet been resolved. Both chapters deferred the concrete implementation of either direction. This chapter supplies it, and the central complication this chapter has to resolve is that the two directions, despite both being invoked by what looks from the outside like a single act of rendering, have entirely different obligations: one is a read, and the other, however it may appear to a user simply looking at a scene, is a write.

## 27.2 The Camera as a Query

A camera, or more generally any observer's request for a view of the world, is not a passive framebuffer waiting to be

filled. It is a query, issued against the world database's interface from Chapter 26: a region of the domain, a requested resolution, and a specification of which of the five components the request actually needs. This last part matters more than it might first appear. Chapter 7 already established that different observation operators recover different partial slices of $M_t$, a camera reading mostly z, a diagnostic tool reading $\Phi$ or R instead, and this chapter's architecture takes that claim literally: a visual renderer's query asks the world database for appearance at fine spatial resolution and coherence or entropy only coarsely, if at all, while a structural inspection tool's query might ask for $\Phi$ and R at fine resolution and skip appearance almost entirely. There is no single, universal rendering query. There is a family of queries, each shaped by what the requesting observer actually needs to see.

## 27.3 Decoding a Resolved Region

For a region the query touches that is already resolved, low entropy, committed appearance, decoding is the straightforward half of this chapter's work, and it is worth being honest that this book has little to add to it beyond situating it correctly. A decoder, whatever neural or classical machinery implements it, takes the stored representation of z at the queried region and transforms it into whatever output modality the observer requires, pixels for a visual renderer, waveform samples for an audio one, natural-

language description for a text-based observer. This is well-trodden territory, and this book's contribution is not a new decoding technique but the guarantee everything upstream of the decoder now provides: the z being decoded is not freshly sampled for this particular query, the way a generative model with no persistent state would produce it, but retrieved, unchanged since it was last committed, from the world database itself. The decoder's job is translation, not invention, whenever the region it is asked to decode has already been resolved.

## 27.4 Querying an Unresolved Region

The harder and more consequential case is a query that touches a region still at high entropy, still represented only by $L_t(x)$, Chapter 8's live possibility set, with no committed z to decode. A decoder cannot simply decode nothing; some output has to be returned to satisfy the query. It is worth being precise about what happens here, because getting this wrong reintroduces exactly the failure this entire book began by diagnosing.

The correct behavior is not for the renderer to generate a plausible appearance and hand it back directly, bypassing everything Chapters 12 through 15 established about how the field is permitted to change. Instead, the query triggers the inverse direction of $O_\theta$: a candidate $\hat{\mathbf{z}}$ is generated, exactly as Chapter 9 described, and this candidate is submitted as a genuine proposal, entering the same proposal-

projection-commit pipeline any other $\Delta\psi$ would enter, per Chapter 25's architecture. The rendering request does not complete until this proposal has been projected against C and, assuming it survives, committed to the world database. Only then is the newly committed z decoded and returned to the observer. This is a slower path than decoding an already-resolved region, and it is slower for a principled reason: rendering an unresolved region is not a read at all, despite appearances. It is a write, disciplined by the same admissibility checking every other write in this book's architecture is subject to, and any implementation that shortcuts this path for the sake of rendering performance has, whether or not it recognizes the fact, reintroduced the wall problem this book opened with.

## 27.5 Level of Detail as Honest Uncertainty

Conventional rendering systems vary detail with distance from the observer, a familiar and well-understood optimization. This architecture has a second, independent axis to vary detail along, and it is worth naming because it is genuinely different in kind from the first. A region can be rendered at reduced fidelity not because it is far from the camera, but because it remains genuinely high in entropy, only partially resolved, and asking the decoder to produce more detail than the underlying $L_t(x)$ actually supports would mean fabricating specificity the field itself does not yet have. This is not a performance shortcut. It is the rendering sys-

tem being honest about what it knows, the visual equivalent of Chapter 8's insistence that entropy be allowed to remain wide wherever it has not been earned down. A distant, fully resolved wall can be rendered in coarse detail purely for performance reasons and still be, if the camera moved closer, revealed to hold a specific, already-committed crack. A nearby but genuinely unobserved region rendered in coarse detail is coarse for a different reason entirely: because there is, honestly, not yet more to show.

## 27.6 Multiple Observers, One World

Because most rendering queries against already-resolved regions are pure reads, many observers, human players, other agents, diagnostic tools, can query the same world database concurrently without any coordination between them, exactly as Chapter 25's client model intended. This changes the moment a query triggers section 27.4's invention path. An invention-triggering query is a proposal, and proposals, per Chapter 15, are subject to the same commutative and non-commutative arbitration any other simultaneous write is subject to. Two observers querying the same unresolved region at nearly the same moment are, functionally, two competing proposals for what that region should become, and the world database resolves this the same way it resolves any other conflict Chapter 15 described, not through any special-cased rendering logic. This book will not develop the full multi-observer treatment here; Chapter

29 takes it up directly, extending exactly this picture to the case where many observers, and many proposal-generating agents, are active in the same world at once.

## 27.7 Looking Ahead

This chapter has given the read side of this book's architecture its full engineering treatment: a camera as a query, a decoder for resolved content, and an inversion into the proposal pipeline for unresolved content, with detail honestly scaled to genuine uncertainty rather than merely to distance. The write side has, so far, only been engineered from the perspective of a query accidentally triggering it. Chapter 28 addresses writing on its own terms, as something an agent deliberately does rather than something a renderer happens to cause, building the interaction layer that turns an agent's intent into the $\Delta\psi$ Chapter 13 already knew how to receive.

# Chapter 28

# Interaction

## 28.1 From Intent to Action

Chapter 10 gave affordance a precise meaning: at the meeting of field and capability, a menu of candidate actions, $P(\Delta\psi)$, becomes available to an agent. Chapter 13 gave the machinery that turns a selected candidate into an actual proposal, combining the field's intrinsic tendencies with a learned forcing term and choosing among live candidates by relevance. Something is still missing between these two accounts, and it is the subject of this chapter: neither Chapter 10 nor Chapter 13 ever specified how an agent's actual intent, a player's click, a keypress, another AI system's planned action, enters this pipeline in the first place. This chapter builds the front end that both of those chapters assumed already existed.

## 28.2 The Action Interface

Rather than exposing an open-ended, unstructured action space, this architecture's interaction layer is built from the small operational vocabulary Chapter 10 already intro-

duced and deliberately kept narrow: store, retrieve, modify, compose, and rewrite. Every action any agent takes, however complex it appears from the outside, resolves into some combination of these five primitives applied to some target patch or region. Picking up an object is retrieval followed by a compose operation binding it to the agent's own state. Placing a bench is a store. Repairing a crack is a modify. Combining two ingredients into a meal is a compose. Rewriting a sign's text according to a rule, rather than an arbitrary edit, is, unsurprisingly, a rewrite. This is a genuine constraint on what the interaction layer will accept, not merely a convenient way of describing actions after the fact, and the constraint is worth keeping: an interaction layer that accepted arbitrary, unstructured intent would have no principled way to check whether a given action even has the shape of something the affordance and proposal machinery downstream can process. Five operations are enough to cover the space this book cares about, and narrowing the interface to exactly these five is what keeps the rest of this chapter's architecture tractable.

## 28.3 A Worked Example: Two Orders for One Alphabet

It is worth working through a single example in enough depth to show what the rewrite operation from section 28.2 is actually for, because "replace content according to some rule rather than an arbitrary edit" can sound abstract until

it is seen doing real work on something concrete.

The Arabic alphabet's twenty-eight letters have, over their history, been given two distinct total orderings. The Abjad order is the older of the two, inherited directly from the Phoenician and Aramaic sequences that preceded it, and it is not arbitrary: it doubles as a numbering system, Abjad numerals, in which each letter also stands for a specific number, used historically in chronology, astronomy, and other contexts where a script needed to double as a counting system. The order is preserved today chiefly through mnemonic strings, Abjad, Hawwaz, Huṭṭī, Kalaman, Saᶜfaṣ, Qarashat, Thakhadh, Ḍaẓagh, that let the sequence be memorized as if it were itself a set of words. The Hijā'ī order is considerably younger, adopted roughly a century after the start of Islam, and it reorganizes the same twenty-eight letters around an entirely different principle: letters that share a common base shape, distinguished from one another only by the placement of diacritical dots, are grouped adjacently, which makes the sequence considerably easier to teach to someone learning to write, and it is this order, not Abjad, that modern dictionaries, indices, and directories actually use.

Two features of this history matter for the interaction layer this chapter has been building. First, the transition from Abjad to Hijā'ī is a rewrite in exactly this book's technical sense, not a modify and not a wholesale replacement. Every one of the twenty-eight letters survives the transition; nothing is added, and nothing is deleted. What changes is a

relation, the adjacency structure imposed on the letters, according to an explicit, statable rule: letters with the same base shape belong together. This is precisely the distinction section 28.2 drew between rewrite and modify, and precisely why this book insists on keeping rewrite as its own primitive rather than treating it as a large modify: a rewrite is content-preserving and rule-governed in a way an arbitrary edit is not, and the Abjad-to-Hijā'ī transition is a case where every letter's persistence through the change can be checked, the way this book's own conservation laws, from Chapter 22, would check that nothing was silently lost in a large-scope proposal.

Second, and just as important, adopting Hijā'ī order for dictionaries and instruction did not retire Abjad order. Abjad numerals are still the ordering in active use wherever the numerical reading of the letters actually matters, and nothing about Hijā'ī's dominance in one context makes Abjad wrong in another. This is Chapter 10's affordance argument in a different register: what structure a set of letters ought to have is not an intrinsic fact about the letters, it depends on what the structure is being asked to do, exactly as a ladder's affordance depends on who is climbing it rather than on the ladder alone. A single alphabet, asked to serve numerology, asks for one order; the same alphabet, asked to serve handwriting pedagogy, asks for another; and this architecture has no trouble holding both answers at once, because neither is more true of the letters than the other.

If this book's write cycle had to process the historical

adoption of Hijā'ī order as a single proposal, it is worth noting what would make that proposal admissible rather than simply large. A $\Delta\psi$ that reordered the alphabet while silently dropping a letter, or introducing one that was never there, would fail Chapter 22's conservation requirements outright, whatever pedagogical convenience it offered. A $\Delta\psi$ that reordered the alphabet according to no statable rule at all, an arbitrary shuffle rather than a shape-based regrouping, would still conserve the letters but would no longer be a rewrite in the sense this chapter's interface actually accepts; it would be indistinguishable, from the interaction layer's point of view, from vandalism. What made Hijā'ī's adoption a legitimate rewrite, worth committing rather than refusing, was precisely that it was both content-preserving and rule-governed, the two properties this book has insisted on for every large-scope proposal since section 28.2 first drew the line between rewrite and arbitrary edit.

## 28.4 Cues as the Bridge from Perception to Action

The full path from perception to action now has a shape worth tracing end to end. An agent's observation, arriving through Chapter 27's rendering path, surfaces cues in the sense Chapter 10 developed: signals that activate relevance, $\rho_c$, for whatever capabilities the agent brings. Those activated cues make some subset of the local affordance menu live, exactly as Chapter 10 described. Somewhere outside

this book's scope, in the agent's own cognition or planning process, whether that agent is a human mind or another AI system, a selection is made among the live affordances. This chapter's interaction layer receives that selection, expressed in terms of the five-operation vocabulary from section 28.2, and hands it forward as the a that Chapter 13's selection policy, $\pi(\mathrm{a}|\mathrm{x}) \propto \exp(\beta \cdot \rho_c(x, a))$, was already built to receive. Nothing in this chapter tries to explain how an agent decides what it wants. That decision is the agent's own, made by whatever cognitive or computational process constitutes it. What this chapter supplies is the interface an already-made decision passes through on its way into the world's own write cycle.

## 28.5 Human and AI Agents, Same Interface

It is worth stating explicitly something the last section only implied: this architecture does not distinguish, at the interaction layer, between a human agent and an AI agent. A human player's click or keypress and another AI system's action output are, from the interface's own point of view, the same kind of thing, a selection among live affordances expressed in the five-operation vocabulary. Both arrive at the proposal service, per Chapter 25's architecture, exactly the same way, and both are held to exactly the same admissibility discipline once Chapter 14's projection stage receives them. This uniformity is not an incidental simplification. It

is what makes the persistent NPCs and multi-agent applications Chapter 31 will eventually build possible at all: an NPC's actions and a human player's actions are proposals of the same kind, checked against the same constraint manifold, capable of conflicting with or complementing one another exactly as any two simultaneous proposals would, per Chapter 15's arbitration.

## 28.6 Direct Editing Is Still a Proposal

It is tempting to imagine a privileged path for tools that need to edit the world directly, a level designer's editor, an administrative console, a debugging interface, bypassing the affordance-cue-selection pipeline sections 28.1, 28.2, and 28.4 developed, on the grounds that such tools are not really agents acting within the world but external instruments operating on it. This chapter argues against any such bypass, and the reasoning is the same reasoning Chapter 27.4 already established for rendering-triggered invention: a write that skips $\Pi_C$ and Commit is a write with no guarantee that it respects the entropy constraint, appearance consistency, or accumulated residue, and an editor able to silently violate these is an editor able to introduce exactly the kind of unearned certainty and unaccountable contradiction this entire book has been built to prevent. A level designer's edit is still, architecturally, a $\Delta\psi$. It may be a $\Delta\psi$ with unusually broad scope, section 28.3's worked example was, in effect, exactly this kind of large-scope rewrite, and it may be sub-

mitted through a different front end than section 28.2's five-operation vocabulary, an editor plausibly needs a richer authoring surface than an in-world agent does, but it still enters the same proposal-projection-commit pipeline as any other proposed change, and it is still subject to being corrected or refused if it conflicts with what the field already holds. This gives the architecture a clean, memorable invariant worth stating on its own: every write to $M_t$ is a $\Delta\psi$, checked and committed through the same pipeline, regardless of what proposed it. There is no other way to change the world, and no component anywhere in this book's architecture holds an unchecked path to the world database that bypasses this rule.

## 28.7 Looking Ahead

This chapter has completed the picture for a single agent: intent, expressed through a narrow action vocabulary, entering the world's write cycle on exactly the same footing whether that agent is human, artificial, or an editing tool operating from outside the simulated world entirely. Nothing in this chapter has yet addressed what happens when many such agents act on the same world at overlapping moments, beyond the brief mention, in section 28.5, that their proposals are held to a common discipline. Chapter 29 takes up multiplicity directly, extending this chapter's single-agent interface to genuine multiplayer coordination.

# Chapter 29

# Multiplayer

## 29.1 From One Agent to Many

Every chapter of Part V so far has quietly favored the case of a single agent acting on the world. Chapter 15.3 and Chapter 27.6 each mentioned, in passing, that simultaneous proposals from different agents are resolved through joint admissibility rather than through a designated authority, and Chapter 28.4 established that human and AI agents enter the interaction layer through the same interface. None of these chapters actually worked through what happens once many agents, human and artificial, are acting on the same persistent field at overlapping moments, which is the ordinary condition of any world meant to be shared rather than merely visited. This chapter completes that picture.

## 29.2 Commutative Proposals: The Easy Case

It is worth starting with the case that requires no new machinery at all, because it is also the most common one in practice. Two agents, proposing changes to disjoint regions

of the field at overlapping moments, one placing a bench in the courtyard, another inspecting a crack on the far side of the wall, are proposing commutative updates in Chapter 15's exact sense: each proposal projects and commits independently, in either order, with no need for the two agents to know about one another at all. Most of what a genuinely populated persistent world requires, at any given moment, falls into this category, and it is worth being honest that this case was already fully solved by Chapter 15. Multiplayer, as an engineering problem, is not primarily about this case. It is about the much rarer, much harder case the rest of this chapter addresses.

## 29.3 Genuine Conflicts: When Proposals Collide

Two agents proposing incompatible changes to the same region at overlapping moments, one moving a bench to the courtyard's east wall, another moving it, at nearly the same instant, to the west wall, cannot both commit. Chapter 15.3 stated the principle governing this case without working through its mechanics: resolution proceeds via joint admissibility rather than an arbitrary tie-break or a first-come-first-served rule. Concretely, this means the projection stage does not process the two proposals independently and then discover a conflict after the fact. It processes them jointly, computing the nearest point in C to the combined pair, and the result depends on what C actually permits: perhaps

only one proposal's bench-placement survives intact while the other is refused outright, perhaps some partial reconciliation exists, a location between the two that satisfies neither agent's exact intent but violates nothing either proposed, if C happens to admit such a compromise. What matters is that the resolution is never arbitrary in the sense of depending on which proposal happened to arrive microseconds earlier. It depends only on what the field's own constraints, established since Chapter 18, actually permit jointly, and an agent whose proposal is refused under this scheme has not lost a race; it has proposed something that, given what another agent simultaneously and equally legitimately proposed, could not be jointly admitted.

## 29.4 Beyond Ordinary Conflict: Adversarial and Pathological Patterns

Section 29.3's machinery handles conflicts between individually legitimate proposals. It has nothing to say about a different and genuinely harder problem: a sequence of proposals, each individually admissible under $\Pi_C$, that together constitute a pattern no shared world should tolerate. An agent that repeatedly proposes the same trivial, technically-admissible action at high frequency is not violating any single admissibility check, and yet is degrading the shared world for every other participant in it, consuming the write cycle's attention without contributing anything $\Pi_C$ is equipped to object to. An agent whose propos-

als, while each individually harmless, form a pattern that another participant experiences as harassment is a harder case still, one where no single proposal is the problem and the problem is visible only in aggregate.

This is worth naming honestly as a genuinely different kind of check than anything Part IV or the earlier chapters of Part V developed. $\Pi_C$ asks whether a proposal is consistent with the field's own constraints. It was never designed to ask whether a pattern of proposals, each consistent with those constraints taken alone, is healthy for the community of agents sharing the field. A separate monitoring layer is needed, one that watches patterns over time rather than checking single proposals in isolation, and the clearest existing precedent for this kind of layer, worth crediting directly, is the Custodian mechanism described in Christopher and Alexandra Chenoweth's MEM|8 architecture: a guard system that tracks repetition scores and psychological-safety signals across a stream of memory events, throttling or intervening when patterns cross thresholds that no single event would trigger on its own, including explicit mechanisms for preventing what that work calls repetition poisoning, an agent trapped in or exploiting a repetitive loop.

## 29.5 A Second, Social Layer of Admissibility

It is worth being precise about how this second layer relates to everything Part IV already built, because the two are com-

plementary rather than competing. $\Pi_C$, and the constraint manifold C behind it, governs field admissibility: whether a proposal is consistent with the persistent world's own accumulated structure. A Custodian-style monitor governs something this book will call social admissibility: whether a pattern of proposals, evaluated across a window of time rather than at a single instant, is consistent with a shared space multiple agents can continue to participate in. These are different questions, and a proposal can pass one while straining the other; the repeated trivial action from section 29.4 passes every field-level check and still degrades the shared world. This book will not attempt a full formal treatment of social admissibility comparable to Part IV's treatment of C; the concept is borrowed, deliberately, from an architecture built to address exactly this problem, and this chapter's contribution is locating where such a layer belongs in the write cycle this book has developed, downstream of $\Pi_C$, monitoring the stream of what actually commits, rather than folding social concerns into C itself and overloading a mechanism that was built to answer a different question.

## 29.6 Synchronization Without Consensus

It is worth closing with a point this book's supporting technical material makes explicitly and that deserves to be stated plainly: none of the machinery this chapter has developed requires an explicit consensus protocol, a designated coordinator deciding whose proposal is authoritative. Different

agents, particularly across a network with real latency, may hold temporarily divergent local views of the field, and this is not, on its own, a violation of anything this book has argued for. What is required is that these local views reconcile once commits actually propagate, in the same sense Chapter 21's gluing condition requires locally admissible patches to assemble into one coherent global section rather than requiring every observer to have identical information at every instant. Admissibility itself, checked at commit time, does the arbitration work a consensus protocol would otherwise be needed for; two agents' temporarily divergent views are reconciled not by a vote or a designated authority but by the same joint-admissibility computation section 29.3 already described. This chapter will not develop the full distributed-systems treatment such reconciliation requires at genuine scale, across many machines and significant network latency; that belongs to Chapter 30.

## 29.7 Looking Ahead

This chapter has treated multiplayer as a logical and architectural question: how conflicting proposals are resolved, how patterns no single proposal reveals are monitored, and why no explicit consensus protocol is required for either. It has not addressed the physical question of actually running this architecture across many machines, with real network latency, real hardware failure, and a world database too large for any single machine to hold. Chapter 30 takes

up scaling directly, extending everything this Part has built to genuine distributed operation.

# Chapter 30

# Scaling

## 30.1 What Breaks at Scale

Chapter 26 and Chapter 29 each deferred a question to this chapter without pretending the deferral was free. Chapter 26 designed sparse, entropy-aware, locality-preserving storage, and then admitted that a world database of any real size will not fit on a single machine. Chapter 29 designed conflict resolution and social monitoring for many simultaneous agents, and then admitted that genuine reconciliation across real network latency needed its own treatment. It is worth naming, before building anything, that these are not one problem but three: storage too large for one machine's disk, computation too large for one machine's processors, and coordination too slow across one machine's network boundary to the next. This chapter addresses all three, and it does so by extending machinery this book has already built rather than introducing a separate distributed-systems architecture bolted onto the side of everything Parts II through IV established.

## 30.2 Sharding by Patch

The natural unit for splitting the world database across machines is, once again, Chapter 21's patches, and it is worth noting how much work this single design decision, made originally for purely mathematical reasons, continues to save this book from having to redo. Each shard, a machine or cluster of machines responsible for a portion of the world, owns a contiguous range of patches along the Hilbert-ordered layout Chapter 26.3 already established, and because Hilbert ordering preserves spatial locality, a contiguous range of patches in storage order is also a spatially coherent region of the domain, not a scattered handful of unrelated fragments. Sharding along these boundaries keeps nearby regions of the world on the same machine far more often than an arbitrary partition would, which matters directly for performance, since most proposals, per Chapter 29.2, touch a single local region and can be resolved entirely within one shard. Chapter 25's single-authoritative-copy discipline is preserved under this scheme, not abandoned: authority over any given patch still belongs to exactly one place, the discipline that made Chapter 15's account of commit meaningful in the first place. What changes is only that this single authority is now distributed across many shards rather than centralized in one machine, each shard the sole authority for its own patches.

## 30.3 Cross-Shard Proposals

The harder case, and the only case this chapter needs genuinely new machinery for, is a proposal whose target spans a boundary between two shards, an action affecting a region that straddles the line between one machine's patches and another's. This is rare, by construction, since Hilbert-ordered sharding keeps most locally relevant regions together, but it cannot be ruled out entirely. When it occurs, the two shards involved must coordinate before either commits: a proposal spanning the boundary is projected jointly, in the same sense Chapter 29.3 already described for two conflicting proposals within a single shard, except now the joint computation itself has to be carried out across a network connection rather than within one machine's memory. This chapter will not derive a full distributed-commit protocol; the standard two-phase pattern, tentatively reserving the affected patches on both shards, confirming joint admissibility, then committing on both sides together or rolling back on both sides together if admissibility fails, is adequate to the requirement and is not this book's own contribution to invent. What matters architecturally is that this coordination is needed only for the rare boundary-spanning case, not for the overwhelming majority of proposals that Hilbert-ordered sharding already keeps local to a single shard.

## 30.4 Hot and Cold Patches

Chapter 26.6 already established that patches are the natural unit of streaming, loaded into memory near an active observer and evicted when no longer needed. At the scale this chapter addresses, that streaming logic extends into a genuine multi-tier hierarchy rather than a simple memory-or-disk distinction. Hot patches, near currently active observers or recently subject to proposals, are held in fast, expensive storage, potentially replicated close to wherever the relevant agents are actually connecting from. Cold patches, far from any current observer and long since settled into stable, rarely-revisited coherence, migrate to slower, cheaper storage, sometimes on a different shard's tier entirely, without this migration meaning anything about the patch's content changes; it remains exactly as authoritative and exactly as available on request, only slower to retrieve on the rare occasion something does request it. This is not a new mechanism invented for scale. It is Chapter 26's streaming discipline, applied across a storage hierarchy with more tiers and more physical distance between them than a single machine's memory and disk ever required.

## 30.5 Unbounded Domains

This chapter's payoff is worth stating explicitly, because it closes a loop opened all the way back in Chapter 2. That chapter's central complaint against context windows and

hidden states was that they are finite buffers, unable to distinguish a fact that fell out of range from a fact that never existed, precisely because their capacity is bounded in advance. The architecture this book has built since Chapter 6 was always meant to answer that complaint, and this chapter is where the answer becomes concrete rather than aspirational. Because sharding is elastic, new shards can be added as the domain grows, each taking authority over a fresh range of patches that did not need to be provisioned in advance, the domain X itself need never be bounded ahead of time the way a context window's token limit or a hidden state's fixed dimensionality must be. A world built this way can grow without a ceiling, not because any single buffer was made larger, but because authority over an ever-expanding domain is distributed across an ever-expanding set of independently bounded shards, none of which needs to anticipate, at the moment it comes into existence, how large the world it is part of will eventually become. This is what Chapter 1 asked for, stated only as a complaint at the time. This chapter is where the complaint receives its architectural answer.

## 30.6 What This Chapter Has Not Solved

It is worth being honest about the gap this chapter leaves open. Nothing here has addressed genuine hardware failure: what happens when a shard goes offline entirely, whether through replication, failover to a standby holding

a synchronized copy of the same patches, or some other fault-tolerance mechanism this book has not developed. The standard answer, replicating each shard's authoritative patches across more than one physical machine and electing a new authority if the current one becomes unreachable, is well understood in distributed systems generally and is not something this book needs to reinvent, but it is also not something this chapter has actually specified. That specification belongs with the fuller Rust software architecture this book's appendices reserve for implementation-level detail, not the main text, where the goal has been the shape of the solution rather than its complete engineering.

## 30.7 Closing Part V

This completes the engineering pass Part V set out to perform. Chapter 25 gave the core write cycle a concrete software architecture, scheduler, world database, proposal and projection services, and renderers as clients. Chapter 26 gave that world database a real storage design. Chapter 27 gave the read side, rendering, its full treatment, including the discovery that rendering an unresolved region is secretly a write. Chapter 28 gave the write side the same treatment from an agent's perspective, culminating in a single invariant that governs every path into the world database without exception. Chapter 29 extended everything to many simultaneous agents, both the ordinary case of conflicting proposals and the harder case of patterns no

single proposal reveals. This chapter has extended all of it to genuine scale, an unbounded domain distributed across an elastic set of shards. Nothing in Parts II through IV remains merely mathematical. Every operator this book introduced now has a place to live as running software.

What this engine is actually for, beyond the ability to sustain a wall that remembers its own crack, is the subject Part VI takes up next.

# Part VI

# Applications

# Chapter 31

# Persistent NPCs

## 31.1 The NPC That Remembers You

Chapter 1 opened with a complaint about a specific, familiar failure: a game character built from a language model that cheerfully claims, on its tenth conversation with the same player, to be meeting them for the first time, not through any dramatic failure but simply because nothing about how it speaks was ever connected to a persistent record of what it had previously said. This chapter is where this book finally answers that complaint directly, rather than using it only as motivation for everything built since.

The answer is not a new memory subsystem bolted onto an otherwise unchanged character architecture. It is the plain application of machinery this book has already spent thirty chapters building. An NPC's interaction history with a specific player is residue, $R_t$, accumulated at whatever region of the field represents that NPC, through exactly the accumulation process Chapter 11 already described. There is no special case here, and that absence of a special case is the entire point.

## 31.2 An NPC Is a Region, Not a Script

It is worth being precise about what this chapter is claiming architecturally, because the claim is stronger than it might first sound. An NPC, in this framework, is not a separate subsystem, a dialogue tree, a behavior graph, or a scripted state machine sitting alongside the world engine and occasionally consulting it. It is a persistent region of the field itself, $M_t$ restricted to wherever that character's own state is represented, subject to exactly the same five-component decomposition and exactly the same write cycle as a wall, a fountain, or a courtyard. Its dialogue and expressed behavior are appearance, z, in whatever modality the interaction actually takes, text, speech, animated motion. Its underlying behavioral tendencies, the stable dispositions a player would recognize as characteristic of this NPC rather than another, are coherence, $\Phi$. Its current goals and in-progress actions are flow, v. Its uncertainty about a given player's intentions, or about how a still-unresolved social situation will play out, is entropy, S, subject to the same entropy constraint from Chapter 8 that governs every other region of the field. And its accumulated history with this and every other player it has interacted with is residue, R. Nothing on this list required inventing a new kind of component. Every one of them is a component this book named for entirely different reasons, applied here to a case that happens to look, from the outside, like a mind.

## 31.3 Personality as a Dynamic Fixed Point

This gives personality, a word this book has avoided until now, a precise technical meaning rather than a vague one. A personality is not a configuration file, a fixed set of parameters consulted once per response. It is a dynamic fixed point in exactly Chapter 16's sense, later refined by Chapter 20 into a conserved trajectory: continuous proposal, continuous projection, continuous commitment, reliably returning to a recognizable, coherent pattern rather than drifting or regenerating fresh with each interaction. A scripted character, responding identically to identical inputs with no genuine ongoing write cycle behind it, is closer to Chapter 16's trivial fixed point, stable because nothing is actually happening rather than because something is being actively maintained. A genuinely persistent NPC, proposing, projecting, and committing continuously, sometimes reflecting on interactions that occurred while no player was present at all, exactly as the fountain's water never stops moving between glances, is a dynamic fixed point in the fuller sense, its personality a pattern the write cycle keeps reproducing rather than a script being replayed. The difference is not cosmetic. It is the same difference Chapter 16 drew between a region that persists because nothing is proposed for it and a region that persists despite constant pressure to become something else, applied here to the specific case of a character a player might spend months returning to.

## 31.4 Cross-Session Memory as Ordinary Residue

It is worth stating plainly what makes cross-session memory work under this architecture, because the mechanism is unglamorous in exactly the way that matters. Residue, once committed, lives in the world database, per Chapter 25's architecture, independent of any particular conversation or session boundary. An NPC's memory of a specific player is not retrieved from a separate long-term memory module, summarized, compressed, and reinjected at the start of each new session, the way a document-store approach, per Chapter 3, would attempt and partially fail at. It is simply still there, at the region associated with that NPC and that player's history, exactly as the wall's crack is simply still there between two glances. This chapter has introduced no new memory mechanism whatsoever. It has shown that the mechanism Chapter 11 built to solve an entirely different-seeming problem, a wall's crack surviving between observations, is already sufficient to solve the problem Chapter 1 opened this book with, once an NPC is correctly understood as a region of the same field rather than a separate kind of thing.

## 31.5 Multiple Players, One NPC

An NPC that many different players interact with, sometimes simultaneously, is not a special case requiring its own

design. It is an instance of exactly the multi-agent machinery Chapter 29 already built: each player's interaction constitutes its own stream of cues and proposals, arbitrated through the same joint-admissibility mechanism Chapter 29.3 described for any other simultaneous proposals touching a shared region. An NPC recalling one conversation while speaking with a different player in a separate, concurrent interaction is not doing anything architecturally distinct from two players independently proposing changes to two different benches; the NPC's own region of the field is simply where several such interaction-streams happen to converge, each governed by the same admissibility discipline as everything else in this book.

## 31.6 What This Chapter Has Not Solved

It is worth being honest about a boundary this chapter has deliberately not crossed. Chapter 28.3 already declined to explain how an agent decides what it wants, treating that decision as belonging to the agent's own cognition rather than to this book's architecture, and the same restraint applies here with particular force. This chapter has said what an NPC's memory, personality, and social continuity consist of, architecturally. It has said nothing about what specific reasoning process, language model, planning algorithm, or other cognitive machinery actually generates an NPC's proposals from moment to moment. That is a question about AI agent design in the ordinary sense, largely independent

of the persistent-world architecture this book has built, and conflating the two would credit this book's contribution with more than it has actually supplied. What this book offers an NPC is somewhere real to live between conversations. What makes it interesting to talk to remains, as it always was, a separate problem.

## 31.7 Looking Ahead

This chapter has shown one application of the completed architecture, a character capable of the specific kind of persistence Chapter 1 found missing in every system examined there. Part VI's remaining chapters extend the same underlying machinery to domains considerably less like conversation and considerably more like science. Chapter 32 takes up scientific simulation directly, asking what it means for a persistent field to model not a courtyard or a character but a physical or biological system whose own laws, not merely its history, must be honored.

# Chapter 32

# Scientific Simulation

## 32.1 From Character to Cause

Chapter 31 asked what this architecture buys a character a player returns to: memory, personality, social continuity, each turning out to be an application of machinery already built for entirely different reasons. This chapter asks a different question of the same architecture, and it is worth being clear at the outset about how different the demand actually is. A scientific simulation does not need a personality. It needs something closer to the opposite of what made Chapter 31's NPC interesting: not a recognizable, characterful pattern of response, but strict, verifiable fidelity to whatever physical or biological law the simulation is meant to model. Coherence, $\Phi$, meant something like narrative or behavioral consistency in Chapter 31. In this chapter it means something considerably more demanding: consistency with equations a domain scientist, not this book, is responsible for getting right.

## 32.2 Domain-Specific Physics as Domain-Specific $F_{\text{phys}}$

Chapter 13 introduced $F_{\text{phys}}$ as the field's own intrinsic tendency, a term that, in every example this book has used so far, meant something fairly generic: coherence smoothing, entropy relaxing, flow subsiding once nothing continues to drive it. A scientific simulation specializes this term rather than departing from it. $F_{\text{phys}}$, for a simulated physical system, is no longer a generic relaxation dynamic; it is literally whatever differential equations actually govern the system being modeled, Newtonian mechanics for a mechanical system, the Navier-Stokes equations for a fluid, reaction-diffusion equations for a biological pattern-formation process. Nothing about this chapter's architecture changes to accommodate this. $F_{\text{phys}}$ was always defined as whatever intrinsic dynamics the field's own laws specify; a scientific simulation is simply the case where those laws happen to be actual, externally validated physics rather than an abstract coherence-smoothing rule invented for this book's own courtyard examples.

## 32.3 Real Conservation Laws as Further Members of C

Chapter 18.2 defined the constraint manifold C as an intersection of members, explicitly left open-ended with an ellipsis, C = $C_S \cap C_z \cap C_R \cap C_A \cap \cdots$, on the understand-

ing that later applications might contribute further members Part IV's own chapters had no occasion to name. Scientific simulation is exactly such an occasion. Energy conservation, momentum conservation, mass conservation, whatever specific physical invariants the simulated domain actually obeys, join C as additional hard constraints, in precisely the sense Chapter 18.4 distinguished from soft, freely varying dynamics. A proposed update that would violate energy conservation is not merely statistically discouraged, the way a generic machine learning system trained to respect physical plausibility might discourage it. It is refused or corrected by $\Pi_C$, the same nearest-admissible-point projection Chapter 14 built for entirely different reasons, exactly as an entropy-violating proposal is refused or corrected in the courtyard. Physical law, under this architecture, is not a preference the simulation has learned to honor most of the time. It is a hard boundary the write cycle cannot cross, structurally, by the same mechanism that has governed every other hard constraint since Chapter 18.

## 32.4 Honest Uncertainty as a Scientific Instrument

Chapter 8's entropy field and Chapter 27.5's honest level of detail were introduced for a courtyard's unobserved corner. They turn out to matter more, not less, once the field being simulated is a scientific system a researcher actually intends to learn something from. An unmeasured region of

a simulated biological or physical system, under this architecture, correctly reports its own genuine uncertainty, $L_t(x)$, rather than silently filling in a plausible-looking value the way an unconstrained generative model, of exactly the kind Chapter 5 examined, would be inclined to do. This is precisely the failure Chapter 5 diagnosed as groundlessness, applied there to InspiroBot's slogans and to fabricated citations, and precisely the failure a scientific instrument cannot be permitted to exhibit: a simulation that quietly invents a plausible reading for a variable it has not actually resolved is not a simulation a researcher can trust, however good the invented value looks. This architecture's entropy discipline, built for an entirely different motivating example, turns out to be exactly the discipline scientific instrumentation already requires, honestly reporting what remains unknown rather than manufacturing the appearance of completeness.

## 32.5 Verification: Trusting the Simulation

It is worth returning, at this point, to a distinction Chapter 4 drew and this book has not revisited since: statistical continuation, what a model finds plausible, against causal continuation, what actually follows from a system's governing rules. A scientific simulation built on this architecture is positioned to make good on causal continuation in a way Chapter 4 could only gesture at, because C, unlike a trained model's learned preferences, is a well-defined mathemati-

cal object, a genuine submanifold of the configuration space Chapter 18.3 introduced, rather than a statistical tendency extracted from training data. This makes formal verification possible in a sense that a black-box generative model never permits: given an explicit statement of a conservation law as a member of C, it becomes possible to prove, rather than merely test and hope, that no admissible trajectory the write cycle can produce violates it, using the type-theoretic, Hoare-logic, and safety-and-liveness apparatus this book's own supporting technical material develops for exactly this purpose. This is the concrete cash value of Chapter 4's distinction, finally redeemed: a simulation whose energy conservation has been proven, rather than statistically observed to usually hold, offers a researcher something a next-token predictor trained on physics footage structurally cannot, a guarantee rather than a plausibility.

## 32.6 What This Chapter Has Not Solved

This chapter has not supplied any physics. It has said nothing about where $F_{\text{phys}}$'s actual equations come from, whether they correctly describe the system a researcher intends to study, or whether the conservation laws proposed as members of C are the right ones for the domain in question. That responsibility belongs to the domain scientist, exactly as Chapter 31 left an NPC's actual reasoning process to whatever cognitive architecture generates its proposals. What this chapter has offered is the surrounding discipline:

a place for validated physics to live where it is structurally enforced rather than merely encouraged, and a place for genuine uncertainty to remain honestly represented rather than quietly overwritten. Getting the physics right remains, as it always was, the scientist's task.

## 32.7 Looking Ahead

This chapter examined a simulation a researcher observes and studies. Chapter 33 turns to a related but distinct case, a system that must act in the physical world it is simulating, or coupled to, in real time. Robotics brings its own demands, and the next chapter asks what changes when a persistent field is not merely observed by a scientist but embodied by a machine that has to move through the world it represents.

# Chapter 33

# Robotics

## 33.1 Embodiment Changes the Constraint, Not the Architecture

Chapter 32 asked what this architecture buys a simulation a scientist observes from outside it. This chapter asks about a harder case: a system whose proposals must be realized as actual physical motion, by actual hardware, subject to actual torque limits, joint ranges, and the unforgiving fact that a physically unrealizable $\Delta\psi$ cannot simply be logged as an interesting hypothesis and set aside. Embodiment does not require a new architecture. It requires a new member of C. Physical realizability, whether a proposed change to the robot's own joint configuration corresponds to something the actual actuators can execute within their real limits, joins the constraint manifold exactly as Chapter 32's conservation laws did, a hard boundary $\Pi_C$ enforces rather than a preference the controller has merely learned to respect.

## 33.2 The Robot as Observer, Generator, and Body

A robot occupies all three roles Chapter 6 distinguished, observer, generator, and world, in an unusually tight loop. It observes, through sensors realizing $O_\theta$. It generates, proposing actions as $\Delta\psi$ through exactly the interaction machinery Chapter 28 built. And its own body, joint angles, limb positions, current gait phase, is itself part of $M_t$, not external to the field it is acting on. This is the technical content behind the word embodiment in this framework: a robot is not an agent standing outside a field and acting on it from a safe remove, the way a human player's avatar was implicitly treated in earlier chapters. It is a region of the field proposing changes to itself, and to the field immediately around it, simultaneously.

## 33.3 Central Pattern Generators as Dynamic Fixed Points

Locomotion supplies this chapter's central example, and it is worth stating its core claim plainly before developing it: gait is not a sequence of target poses a controller repeatedly corrects toward. It is a dynamic fixed point in exactly Chapter 16's sense, later refined by Chapter 20 into a conserved trajectory through phase space. A central pattern generator, the biological and robotic mechanism that produces rhythmic locomotor patterns through coupled oscil-

lation rather than explicit trajectory planning, was already trajectory-centric before this book's own vocabulary existed to describe it this way; this chapter's contribution is recognizing that a walking gait is precisely the kind of continuously regenerated coherence Chapter 16 built its account of dynamic fixed points to describe, not the kind of static, trivially-stable target pose a naive controller might instead pursue. Balance, under this reading, is not a state a robot achieves and then holds still. It is a coherence, $\Phi$, continually regenerated by an ongoing cycle of proposal, projection, and commit, in the same sense the fountain from Chapter 16 was never at rest even while its overall form remained recognizably stable.

## 33.4 Hierarchical Composition and the Kuramoto Coupling

A single joint's oscillation is described by a phase, and a population of coupled joint-level oscillators synchronizes according to a coupling law most naturally expressed through the Kuramoto order parameter,

$$Z(t) = r(t)\, e^{i\psi(t)},$$

where $\mathrm{r}(\mathrm{t})$ measures how synchronized the population currently is and $\psi(t)$ is the population's mean phase. This is Chapter 20's phase-space machinery, $\Phi$ and $\mathrm{v}$ as conjugate pairs generating conservative oscillation, applied to an unusually literal case: a joint's own oscillatory state gen-

uinely is a position-and-momentum-like pair in the sense Chapter 20 developed, not merely analogous to one. What makes locomotion architecturally interesting, and what this chapter borrows directly from existing work on hierarchical continuation in gait, is that this coupling does not stop at a single joint. Joint-level oscillator populations compose into limb-level coordination; limb-level operators compose into gait-level pattern, walking, trotting, galloping; and gait-level pattern composes into whole-body coordination. Different gaits, under this composition, are not different dynamical laws requiring separate controllers. They are different parameterizations of the same underlying coupling law, distinguished only by different target phase offsets in the coupling matrix K relating one oscillator population to another. This is Chapter 23's structural-equivalence question, answered concretely: a walk and a trot are isomorphic in the relevant sense, related by a structure-preserving reparameterization of one shared coupling architecture, not two unrelated things that happen to both move a robot forward.

Each level of this hierarchy also runs on its own clock, in precisely Chapter 17's sense. Individual joint oscillation is the fastest layer, cycling many times within a single stride. Limb coordination is slower. Overall gait pattern slower still. Whole-body balance, the slowest layer, is what Chapter 25.2's scheduler would treat as this hierarchy's coarsest update rate, checked and reconfirmed far less often than any individual joint's phase, exactly as Chapter 17 argued a field's slow components generally are.

## 33.5 A Sketch of the CPG Chain Controller

It is worth making this hierarchy's shape concrete before moving on, both as a controller design and as an instance of Chapter 25's system architecture. The composition runs as follows:

```
JointOscillator
  - phase: real
  - naturalFrequency: real
  - couples_to: [JointOscillator]  (via coupling weight
  K_ij)
  + step(dt) -> proposes ΔΦ_joint, Δv_joint

LimbController
  - oscillators: [JointOscillator]
  - orderParameter: Z(t) = r(t) e^{i ψ(t)}
  + synchronize() -> aggregates joint proposals into
  one limb-level Δψ

GaitController
  - limbs: [LimbController]
  - couplingMatrix: K            (target phase offsets;
  walk/trot/gallop
                                   differ only in K's
                                   values, not its
                                   shape)
  - activeGait: GaitParameterization
  + selectGait(K) -> reparameterizes limb coupling, no
  controller swap

WholeBodyCoordinator
```

```
  - gait: GaitController
  - balanceCoherence: Φ_whole_body
  + monitor() -> flags Φ_whole_body decline before
  observable instability

ProposalService (Chapter 25)
  <- receives Δψ from WholeBodyCoordinator, same
  interface as any
     other proposal source; no special-cased path for
     "robot" proposals
```

Nothing in this sketch introduces a component Chapter 25 did not already specify. WholeBodyCoordinator's output enters the proposal service exactly as any other $\Delta\psi$ would, and is projected against C, now including the physical-realizability constraint from section 33.1, exactly as any other proposal is. A CPG chain controller, under this architecture, is not a special robotics subsystem sitting beside the world engine. It is a particular, highly structured way of generating $\Delta\psi$, nothing more privileged than that.

## 33.6 A Grammar for CPG Chain Configuration

Because different gaits are reparameterizations of one coupling law rather than separate controllers, it is useful to have a small, well-formed grammar for specifying a configuration rather than hand-coding each gait separately. The following BNF sketch is deliberately minimal, sufficient to specify a coupling matrix and its target phase offsets rather

than a complete robotics description language:

```
<cpg-config>      ::= <oscillator-list> <coupling-list>
<gait-name>

<oscillator-list>  ::= "oscillators" "{" <oscillator> {
"," <oscillator> } "}"
<oscillator>       ::= <id> ":" "freq" "=" <real>

<coupling-list>    ::= "coupling" "{" <coupling-entry>
{ "," <coupling-entry> } "}"
<coupling-entry>   ::= <id> "->" <id> ":" "weight" "="
<real>
                                      "," "phase-offset"
                                      "=" <real>

<gait-name>        ::= "gait" "=" <string>

<id>               ::= letter { letter | digit }
<real>              ::= digit { digit } [ "." digit {
digit } ]
<string>            ::= "\"" { character } "\""
```

A walking gait and a trotting gait, under this grammar, are two instances differing only in their ‘<coupling-list>’s phase-offset values and the ‘<gait-name>‘ label; the ‘<oscillator-list>‘ and the shape of the coupling relation can remain identical. This gives concrete, checkable content to the claim from section 33.4 that gaits are parameterizations rather than separate designs: two configurations sharing an oscillator list and coupling topology, differing only in phase offsets, are exactly the isomorphic-but-differently-

parameterized objects Chapter 23 gives a name to.

## 33.7 Balance as a Diagnostic: Coherence Before Collapse

This architecture offers one further, genuinely practical payoff worth naming. Because whole-body coherence, $\Phi$, is tracked explicitly rather than inferred only from whether the robot has actually fallen, it becomes possible to monitor coherence as a leading indicator rather than waiting for failure to become visible. A stumble, on this reading, is a trajectory moving from the neighborhood of a stable fixed point, Chapter 19.5's local minimum of the relevant energy functional, toward an unstable one, a saddle or local maximum from which small perturbations grow rather than being absorbed back. This book's own machinery predicts that $\Phi$ should measurably decline during this transition before any externally observable loss of balance occurs, since the fixed point's stability, not merely the robot's current pose, is what is actually degrading. A controller monitoring $\Phi$ directly, rather than only monitoring whether the robot is still upright, has a genuine early-warning signal this architecture supplies for free, simply by taking its own account of coherence and stability seriously as something worth measuring in real time rather than only as a theoretical construct.

## 33.8 What This Chapter Has Not Solved

This chapter has not designed a coupling matrix, tuned a controller for any particular robot body, or specified how a physical-realizability constraint's actual torque and range limits should be derived from a given robot's hardware. Those are control-theoretic and mechanical engineering questions, exactly as the equations governing a scientific simulation were, in Chapter 32, the domain scientist's responsibility rather than this book's own contribution. What this chapter has offered is a place for an existing, independently developed body of locomotor control theory to sit inside this book's architecture without contradiction, and a small amount of genuine payoff, the coherence-decline diagnostic in particular, that falls out of taking the architecture's own vocabulary seriously.

## 33.9 Looking Ahead

Robotics asked what happens when a persistent field must be embodied in hardware that acts on and within the world it represents. Chapter 34 turns to a different kind of embodiment, not in physical machinery but in a learner's own developing understanding, asking what this architecture offers education: a domain where the field being maintained is not a robot's body or a physical system, but a student's evolving grasp of one.

# Chapter 34

# Education

## 34.1 The Field Is Not the Subject, It Is the Learner's Encounter With It

Chapter 32 asked what this architecture buys a scientist studying a simulated system from outside it. Education asks a related but distinct question, and it is worth being precise about the distinction before going further. An educational application certainly needs a persistent field for its subject matter, an ecosystem, a mathematical structure, a historical period, and everything Chapter 32 established applies directly to that half of the problem. What is genuinely new here is the other half: a persistent field is also needed for the learner's own encounter with that subject, a model of what a specific student currently understands, has misunderstood, and is or is not yet ready to be shown next. This chapter is about that second field, not the first, since the first is simply an instance of work already done.

## 34.2 What a Student Is Ready to Learn: Affordance Revisited

Chapter 10 established that a ladder affords climbing to a human and not to a goldfish, not because the ladder changed, but because affordance is a relation between the field and a capability, $A : M \times C \to P(\Delta\psi)$. This relation transfers to education with almost no modification. A given concept, problem, or piece of content affords being learned by a student whose current understanding, C, already supports it, and does not afford being learned by a student whose understanding does not yet include the prerequisites it depends on. The content has not changed between these two students. What differs is C, exactly as what differed between the human and the goldfish was capability rather than the ladder. This gives adaptive sequencing, presenting a student with what they are actually ready for next, a precise technical home in this book's own architecture rather than requiring a bespoke mechanism invented for education specifically: it is Chapter 10's affordance relation, with a student's current understanding standing in for physical capability.

## 34.3 Honest Uncertainty About What a Student Knows

It is worth applying Chapter 8's discipline here with particular care, because the temptation to violate it is stronger in

this domain than in most others this book has examined. A system's model of whether a given student has genuinely grasped a given concept is not always known with confidence, and Chapter 8's entropy field, $L_t(x)$, is exactly the right tool for representing this honestly: a concept a student has not yet been meaningfully assessed on should remain genuinely uncertain in the system's model, neither assumed mastered nor assumed unlearned, rather than collapsed prematurely in either direction. A system that assumes mastery without evidence risks advancing a student past material they have not actually absorbed. A system that fabricates a confident assessment of misunderstanding it has not actually observed is committing the same failure Chapter 5 diagnosed in InspiroBot, a plausible-looking claim with nothing underneath it, applied here to a student's own understanding rather than to a stock inspirational phrase. The entropy constraint from Chapter 8, that certainty may not be manufactured without an observation that earns it, is not a mathematical nicety in this application. It is the difference between a system that honestly tracks what it does and does not know about a learner and one that quietly pretends to a confidence it never actually earned.

## 34.4 Residue as Learning History

A concept's accumulated residue, R, in this application, is the record of a specific student's actual history with it: prior attempts, prior mistakes, prior explanations already given

and, per Chapter 9's invention-to-memory principle, not owed to be reinvented from scratch on the next encounter. A student who struggled with a particular misconception three sessions ago, and whose subsequent work suggests that misconception has since been resolved, has a residue trace reflecting both facts, not merely the most recent one; exactly as Chapter 11 argued residue is compressed rather than a complete log, an educational system does not need to retain every interaction verbatim, only enough of the trace to determine what remains relevant to how this student should be taught this concept now. This is, again, not a new mechanism. It is Chapter 11's residue accumulation and Chapter 9's invention-to-memory discipline, applied to a domain where getting them right matters rather more than it did for a courtyard's crack.

## 34.5 Interactive Textbooks as Explorable Fields

The subject-matter side of this chapter's problem is worth returning to briefly, because it benefits from a mechanism developed for an entirely different purpose. A conventional textbook is fixed and linear: every reader encounters the same pre-written content in the same order. A textbook built as a persistent field, per Chapter 6, need not be. A student's question about some aspect of the material not yet explicitly written out is not, under this architecture, a dead end requiring either silence or an ungrounded improvisa-

tion. It is, in exactly Chapter 27.4's sense, a query into an unresolved region, triggering a genuine write, a proposal generated, projected against whatever the text's own established content and constraints permit, and committed, becoming a permanent, citable part of the text's own history rather than a one-off answer generated and then forgotten. A second student asking the same question later receives the same answer, not a fresh, potentially inconsistent one, for the same reason two observations of the wall's crack agree: because the first answer was committed rather than merely displayed.

## 34.6 What This Chapter Has Not Solved

This chapter has not supplied a theory of pedagogy. It has said nothing about which teaching strategy actually helps a given student learn, how to sequence content beyond the bare affordance relation section 34.2 established, or how to design assessments capable of genuinely distinguishing mastery from its absence. Those are questions for learning science and instructional design, exactly as the equations of motion were, in Chapter 32, the domain scientist's responsibility rather than this book's own. What this chapter has offered is a place for a learner's own understanding to be tracked honestly over time, and a place for subject matter to be explored rather than merely delivered, using no mechanism this book had not already built for reasons that had nothing to do with education.

## 34.7 Looking Ahead

Education asked what this architecture offers a system responsible for tracking what someone is coming to understand. Chapter 35 turns to a domain with a different obligation entirely: not fidelity to a learner's actual state, and not fidelity to physical law, but creative worlds, where invention is not a regrettable necessity to be minimized but the entire point.

# Chapter 35

# Creative Worlds

## 35.1 Invention Was Always Allowed

It is worth correcting a misreading this book's own emphasis on persistence and discipline could easily invite. Nothing in the last thirty-four chapters has forbidden invention. Chapter 9's invention-to-memory principle explicitly licenses it: a genuinely unresolved region is free to be invented, once, when entropy is honestly open. What that principle forbids is not invention but reinvention, generating a fresh, unrelated answer to a question that was already settled. Creative worlds, games, films, collaborative fiction, virtual societies built for their own sake rather than for fidelity to anything external, are the domain where the first half of this principle is exercised most extensively and most joyfully. They are not, for that reason, a domain where the second half matters any less. This chapter argues the opposite: creative invention needs this book's discipline more urgently than most of the applications examined so far, not less, because a creative world's entire value depends on its invented content actually staying invented.

## 35.2 Plot Holes Are the Wall Problem in Fiction

Readers, viewers, and players have a name for exactly the failure this book opened with, encountered in a different genre. A story that contradicts something it established earlier, a character whose eye color changes between scenes with no explanation, a magic system whose rules quietly shift to suit whatever the plot currently needs, a game world where an NPC's established history is casually ignored, is a story with a plot hole, and a plot hole is Chapter 1's wall problem wearing narrative clothing. A generative storytelling system built without this book's architecture will produce exactly this failure at scale, and for exactly the reason Chapter 4 already diagnosed: such a system asks, at each step, what scene is statistically plausible given recent context, not what scene is required given everything the story has actually committed to. Narrative causal continuation, a scene that follows because it must given what has already happened, is a different computation than narrative statistical continuation, a scene that merely sounds like it could follow, and the difference is invisible within any single scene and glaring the moment two scenes are compared against each other, exactly as the wall's crack was.

## 35.3 World-Building Is Constraint Design

This chapter's central reframing follows directly. What a fantasy or science-fiction author does when establishing how magic works, what technology exists, or what a society's governing rules are, is constructing C, the constraint manifold, for their invented world, in exactly Chapter 18's sense. A fictional world's C can contain things a physical simulation's, per Chapter 32, never would, magic, faster-than-light travel, talking animals, and this is not a defect in the analogy. It is the entire point of fiction: the author has genuine authorial freedom over what C contains. What the author does not have, if the resulting world is to feel coherent rather than arbitrary, is freedom over whether C, once established, is actually honored. A rigorously specified magic system, of the kind some authors deliberately build with explicit, statable rules, corresponds to a tightly constrained C; a looser, more atmospheric magic system corresponds to a more permissive one. Both are legitimate authorial choices. Neither is exempt from the requirement that $\Pi_C$, once C has been chosen, actually enforce it with the same rigor Chapter 32 demanded of real conservation laws. Creative freedom, in this framework, lives entirely in the choice of C. Discipline lives in what happens after that choice has been made.

## 35.4 Collaborative Storytelling as Multi-Agent Proposal

Multiple authors, or a game's many players, jointly building a shared fictional world are an instance of exactly the multi-agent machinery Chapter 28 and Chapter 29 already built. Two writers proposing contradictory accounts of a character's backstory are proposing conflicting $\Delta\psi$ to the same narrative region, and Chapter 29.3's joint-admissibility resolution applies without modification: the outcome depends on what the story's own established C actually permits jointly, not on whichever writer happened to submit their addition first. This gives collaborative fiction a form of discipline that, in practice, usually falls to an external continuity editor or a community-maintained wiki, built directly into the architecture instead: a shared fictional world under this framework does not need a human continuity checker working after the fact, because admissibility checking is already happening at commit time, the same discipline that has governed every other shared, multi-agent world since Chapter 29.

## 35.5 Retcons as Rewrite

It is worth addressing directly a case that might otherwise look like an exception to everything this chapter has argued: authors do, sometimes legitimately, want to revise something already established, a retcon in the vocabulary of se-

rial fiction. This is not an exception. It is Chapter 28.3's alphabet-reordering example, returned to in narrative form. A retcon, handled well, is a rewrite: content-preserving in the sense that it does not silently erase or contradict the story's other established elements without accounting for them, and rule-governed in the sense that it offers a statable, coherent revision rather than an arbitrary discontinuity, exactly as the transition from Abjad to Hijā'ī order preserved every letter while restructuring their relations according to an explicit principle. A retcon handled badly, an unacknowledged contradiction of established residue with no coherent account of how the change occurred, is exactly the kind of proposal this architecture's discipline is built to catch: a large-scope $\Delta\psi$ that conflicts with accumulated residue and offers no rule to justify the conflict is a $\Delta\psi$ Chapter 15.4's refusal mechanism exists to decline, whether the proposal comes from a courtyard's contradictory ground-state or from a franchise's contradictory backstory.

## 35.6 What This Chapter Has Not Solved

This chapter has not supplied creative talent. It has said nothing about what magic system, plot, or fictional society is actually good, interesting, or worth building, exactly as Chapter 32 said nothing about which physical equations were worth simulating. What this chapter has offered is the discipline that keeps whatever is invented, once invented,

honored: a place for a story's own committed facts to remain committed, for multiple collaborators to build on the same shared fiction without silent contradiction, and for legitimate revision to be distinguished, architecturally, from a plot hole.

## 35.7 Looking Ahead

Creative worlds asked what this architecture offers a domain built for invention rather than fidelity to anything external. Chapter 36, the last of Part VI's applications, asks what happens at a still larger scale, when the persistent field being maintained is not a story, a character, or a physical system, but an entire simulated civilization, its institutions, its economies, and the long-horizon consequences of everything its inhabitants have committed to over time.

# Chapter 36

# Civilization Engines

## 36.1 Institutions as Dynamic Fixed Points at Civilizational Scale

Chapter 10 borrowed language from work on play and institutions to describe affordance: institutions as factories, farms, ecosystems, structures that shape what possibilities are available to whoever encounters them. This chapter returns to that framing at the scale it was always gesturing toward. A currency, a legal system, a market, a language, is a dynamic fixed point in exactly Chapter 16's sense, continuously re-proposed by each participating agent's ordinary activity and continuously projected and committed back into recognizably the same institution, across a timescale that vastly exceeds any single agent's participation in it. No individual citizen holds a currency system stable through sheer personal effort, any more than any single water molecule holds a fountain's form stable. The institution persists the way the fountain persists, through the aggregate effect of continuous proposal and commitment by a population that turns over entirely many times across the institution's own lifespan.

## 36.2 Multi-Generational Residue

This gives civilizational memory, law, precedent, cultural norm, a precise architectural home: accumulated residue, R, at the institutional region, built through the same $\oplus$-accumulation Chapter 11 described, operating here across a timescale where the agents contributing to it are themselves finite-lived while what they contribute to is not. A legal precedent, once committed, constrains future admissible proposals exactly as a courtyard's residue constrained what could legitimately be invented there, whether or not any agent currently alive was present when the precedent was established. This is the same discipline Chapter 11 argued for from the very beginning, applied at a scale where its consequences compound across generations rather than across a single afternoon.

## 36.3 Economies as Conserved-Flow Systems

An economy, under this architecture, is naturally described through Chapter 22's conservation machinery rather than through any new apparatus. Value or resources should be conserved except at explicitly licensed sources and sinks, production standing in for the source term and consumption for the sink, in precisely the structure Chapter 22.3 established for licensed coherence generally: any local increase must be traceable to a licensed source or to inflow

from elsewhere, never simply to nothing. An economic simulation built on this architecture inherits the same diagnostic power Chapter 32 claimed for physical conservation: a proposal that would create value from nothing, or destroy it without an accounted sink, is not merely economically implausible, it is inadmissible in the same structural sense an energy-violating physics proposal is, refused by $\Pi_C$ rather than merely discouraged by training.

## 36.4 Privilege Gates as Capability-Relative Affordance

Chapter 10's affordance relation, $\mathrm{A} : \mathrm{M} \times \mathrm{C} \to \mathrm{P}(\Delta\psi)$, returns here in close to its original form, and it is worth noting how directly the institutional framing that motivated Chapter 10 in the first place applies to the domain this chapter actually concerns. What a given citizen can do within an institution, which markets they can access, which legal remedies are available to them, which social mobility is realistically open to them, depends on their capability profile C exactly as a ladder's climbability depended on the capability of whoever approached it. The same institution, the same laws, the same market, affords different things to different agents, not because the institution has changed but because C has, and a civilizational simulation built on this architecture inherits, for free, the capacity to represent stratified access honestly rather than assuming every agent faces the same affordances the institution's rules nominally describe.

## 36.5 Long-Horizon Stability and Collapse

Chapter 16.5's distinction between stable and unstable fixed points, and Chapter 33.7's application of that distinction to a robot's balance, both return here with civilizational stakes. A stable institution absorbs perturbation, a currency surviving an economic shock, a legal system accommodating a controversial precedent without fracturing; an unstable one is, like Chapter 16.5's precarious stack of stones, a technically valid fixed point that a small enough disturbance sends toward collapse rather than recovery. The diagnostic Chapter 33.7 developed for a robot's balance, tracking coherence, $\Phi$, as a leading indicator that declines before failure becomes externally visible, applies with the same logic here: an institution's coherence beginning to decline is, in principle, detectable before its collapse is, offering a civilizational simulation the same kind of early-warning signal a walking robot's controller gets from monitoring its own stability directly rather than waiting to observe an actual fall.

## 36.6 What This Chapter Has Not Solved

This chapter has not supplied economic theory, political theory, or any design for what institutions a simulated civilization ought to have. It has said nothing about which laws are just, which markets are efficient, or which social structures are worth building, exactly as Chapter 32 declined to sup-

ply physics and Chapter 35 declined to supply plots. What it has offered is the discipline that would let a simulated civilization behave with genuine long-horizon consistency, institutions that persist for principled reasons rather than merely because a script says so, memory that compounds honestly across generations, and access that is represented as the stratified thing it actually is rather than assumed uniform.

## 36.7 Closing Part VI

It is worth pausing, at the end of Part VI, on how little new machinery these six chapters actually required. An NPC's memory, a student's tracked understanding, and an institution's accumulated law are the same residue mechanism from Chapter 11. A ladder's climbability, a student's readiness to learn a concept, and a citizen's access to a market are the same affordance relation from Chapter 10. A fountain's persistent form, a robot's regenerated gait, and a currency's multi-generational stability are the same dynamic fixed point from Chapter 16, refined by Chapter 20. A physics simulation's energy conservation and an economy's resource conservation are the same conservation machinery from Chapter 22. Six domains that could not look more different from one another on the surface, characters, laboratories, robots, classrooms, fiction, civilizations, turned out, in every case, to be applications of a small, fixed set of primitives this book built for entirely different, far more modest

reasons: a wall that should not forget its own crack. That six wildly different domains all reduce to the same handful of mechanisms is not incidental to this book's argument. It is the argument, demonstrated rather than merely claimed.

Part VII turns from what this architecture can build to what it means, asking what a physics built from admissibility rather than objects actually implies, and what becomes of questions about minds, observers, and reality itself once persistence is understood the way this book has spent thirty-six chapters building it.

# Part VII

# Toward Semantic Physics

# Chapter 37

# Physics Without Objects

## 37.1 The Oldest Assumption in Physics

Most physical and everyday reasoning shares an assumption old enough that it rarely gets stated outright: that reality is fundamentally made of things, substances, particles, objects, which persist through time and to which change happens. A wall is a thing; erosion happens to it. A person is a thing; aging happens to them. The thing comes first, ontologically, and change is something layered on top of it, an event a persisting substance undergoes rather than something constitutive of what the substance is. This book has never stated this assumption, because it never needed it, and it is worth pausing, now that the engineering is behind it, to notice what was quietly rejected along the way.

## 37.2 What This Book Already Showed, Restated as Metaphysics

Nothing here is new content. It is worth restating, plainly, what earlier chapters already established, because their consequence is more sweeping than their original engineering

framing let on. Chapter 16 showed that an object's identity in this framework is not a label attached to a persisting substance, but a fixed point of the write cycle, a configuration the field's own dynamics keep reproducing. Chapter 20 refined this further: identity is not even a fixed point in the simplest sense, but a conserved trajectory through phase space, a closed orbit the system's dynamics maintain rather than a static thing sitting still. At no point in the intervening chapters was there ever occasion to introduce a substance, a persisting stuff that the wall, the fountain, or an institution was made of, over and above the trajectory itself. The wall was never a thing that happened to trace a stable trajectory. The wall was the trajectory, stable enough and long-lived enough to be worth a name.

## 37.3 Process Ontology, Named Directly

Chapter 6.6 used the phrase process ontology in passing, deliberately, without developing it, promising a return that this chapter now makes good on. What this book has built is a version of process ontology, the philosophical position, associated most closely with Alfred North Whitehead, that processes rather than substances are what is fundamental, and that things are best understood as patterns processes exhibit rather than as the bearers of those processes. It is worth being honest about the relationship between this book's own account and that tradition. This book did not set out to argue for process philosophy on its own philo-

sophical terms, and it will not attempt, here, to adjudicate the centuries of argument that tradition has generated. What is genuinely interesting is the independence of the path that arrived at a structurally similar place: an engineering problem, keeping a generated wall consistent with itself across two observations, pushed, through nothing more exotic than the requirement that persistence actually be architectural rather than merely convenient, toward a formalism in which trajectories, not objects, were always doing the fundamental work. That a working piece of engineering reconstructs something with this shape, without having been built to prove any philosophical thesis, is itself a small piece of evidence worth taking seriously, whatever one ultimately makes of process philosophy as a general metaphysical position.

## 37.4 What "Object" Still Means

It would be a mistake to read this chapter as eliminativist, as though it argued that walls, fountains, and institutions do not really exist and are merely convenient fictions layered over an underlying flux. That is not this book's claim, and it is worth being precise about what the claim actually is. A dynamic fixed point, in the sense Chapters 16 and 20 developed, is a genuinely distinguished, non-arbitrary structure within the space of all possible trajectories: most trajectories are not fixed points, most configurations do not reproduce themselves under the write cycle's own dynamics, and the

ones that do are picked out by real mathematical structure, not by an observer's convenience in naming them. The wall exists, in exactly the sense the word ordinarily carries. What has changed is only the analysis of what makes it exist as one continuing thing rather than a sequence of unrelated states: not a persisting substance underneath the trajectory, but the trajectory's own stability, the fact that it is a genuine fixed point rather than a passing configuration. Object remains a perfectly good word for what a stable, self-reproducing trajectory is. It is simply no longer doing the explanatory work substance metaphysics traditionally asked it to do.

## 37.5 Why This Isn't Merely a Relabeling

It is fair to ask whether anything beyond vocabulary actually turns on this distinction, and it is worth answering with a concrete case rather than an abstract assurance. Chapter 33.7 developed a genuine diagnostic from treating a robot's balance as a trajectory near a fixed point rather than as a binary state, upright or fallen: coherence, $\Phi$, was shown to decline measurably before an observable loss of balance occurred, because the underlying claim being tracked was the stability of a trajectory, a continuous, quantifiable property, rather than the presence or absence of a persisting object called balance. A substance-first framing has no comparably natural place for this kind of leading indicator; balance, on that framing, is a state a robot either has or does not have, and there is no obvious quantity to track before

the state changes. A trajectory-first framing predicts, structurally, that instability should be visible in advance, because instability is a fact about a trajectory's neighborhood, Chapter 19.5's local minima and saddle points, not merely about whether a threshold has yet been crossed. Chapter 36.5 found the same structure again at civilizational scale, an institution's coherence declining before its collapse becomes evident. This is not a difference in vocabulary. It is a difference in what kind of question the framework equips you to ask, and in at least these two cases, the trajectory-first question turned out to have a real, useful answer the substance-first question did not obviously have available.

## 37.6 Looking Ahead

If trajectories rather than objects are what is fundamental, a question immediately follows that this chapter has not yet addressed. Not every conceivable trajectory is one this book's architecture would permit; Chapter 18's constraint manifold C was built precisely to distinguish admissible configurations from merely imaginable ones. What, then, makes a trajectory real rather than merely coherent as an idea, and is admissibility itself doing more philosophical work here than this book has so far acknowledged? That is the question Chapter 38 takes up directly.

# Chapter 38

# Reality as Constraint

## 38.1 What Makes a Trajectory Real?

Chapter 37 closed by noting that not every trajectory Chapter 13's proposal machinery could generate is one this book's architecture would actually permit into a world's history. C, the constraint manifold Chapter 18 built for engineering reasons, does the work of picking out which trajectories are admissible and which are not. This chapter asks what that work actually amounts to, once it is examined philosophically rather than only operationally: what is admissibility doing, when it does the deciding between a trajectory that becomes part of a world and one that does not?

## 38.2 Possibility, Actuality, and the Write Cycle

The traditional vocabulary for this question is modal: possibility and actuality, what could be the case against what actually is the case. This book's write cycle, examined again with this vocabulary in mind, turns out to map onto that

distinction with unusual precision. A proposal, $\Delta\psi$, is a possibility in close to the philosopher's sense: a candidate way things might go, generated but not yet evaluated against anything. Projection, $\Pi_C$, is the evaluation, checking that candidate against whatever the world's own constraints actually permit. Commit is actualization itself, the specific, well-defined operation by which a mere possibility becomes part of what a world's history actually contains. What is worth noticing is how unusual it is for a philosophical account of modality to come with an actual mechanism for this last step. Most treatments of possibility and actuality leave the transition from one to the other either unexplained or grounded in something this book has no occasion to invoke, a metaphysically primitive act of actualization, an observer collapsing a wave function, the best of all possible worlds being selected by some cosmic optimization. This architecture supplies, instead, a precisely specified operation, checkable and, in the engineering chapters, actually implementable, and it is worth taking seriously that this is not nothing, whatever else remains open about how far the analogy to physical modality extends.

## 38.3 Refusal as Genuine Impossibility

This gives modal vocabulary a further, more precise distinction this book's own machinery already computes without needing new philosophical apparatus. A proposal that $\Pi_C$ can correct into some nearby admissible state, section 14.3's

ordinary case, was possible in a qualified sense: not actual exactly as proposed, but actual in some modified form close enough to preserve its intent. A proposal that triggers Chapter 15.4's refusal, where no admissible state honors what was proposed closely enough to count as a correction rather than a substitution, is impossible in a considerably stronger sense: not merely a possibility that failed to occur, but a candidate the world's own constraints rule out entirely. This book's architecture does not treat possibility and impossibility as a binary applied from outside, by an observer's judgment about what seems conceivable. It computes the distinction directly, as a consequence of whether an admissible nearby state to a given proposal actually exists.

## 38.4 Is C Arbitrary?

It is worth confronting directly a tension the last several chapters have left unaddressed. Chapter 35 argued that world-building, in fiction, is constraint design: an author genuinely chooses what C contains, magic, faster-than-light travel, talking animals, entirely at their own discretion. If C can simply be chosen, does treating admissibility as a serious account of reality collapse into something arbitrary, reality being whatever anyone happens to stipulate?

The honest answer requires distinguishing what varies across this book's applications from what does not. C's source genuinely differs by domain: authorial choice in fic-

tion, per Chapter 35; empirical discovery in physical simulation, per Chapter 32, where C is not chosen but measured, constrained by what actual experiments reveal about actual conservation laws; something closer to negotiated, emergent convention in the institutions of Chapter 36, where a society's laws are neither arbitrarily stipulated by one author nor simply read off from physical measurement, but arrived at through processes this book has not attempted to formalize. For the case that matters most to a metaphysical claim, the actual universe's own C, this book offers no account of why it contains what it contains rather than something else. This is not a gap unique to this framework. It is one of the oldest open questions in the philosophy of physics, why the laws are these laws and not others, and this book does not resolve it. What this chapter can honestly claim is narrower and still worth having: that once reality is understood as admissibility plus persistence, the question of where a given domain's C actually comes from becomes a well-posed, meaningful question with a real answer to be sought, rather than a confusion to be dissolved. Fiction's C is chosen because fiction's whole point permits choosing it. Physics's C, whatever it ultimately turns out to be grounded in, is emphatically not chosen in that sense, and this book's framework has enough structure to at least state that difference precisely, even where it cannot yet explain it.

## 38.5 Existence Without Substance

This closes the loop back to Chapter 37 tightly enough to be worth stating as a single combined claim. Existence, under this framework, is not an intrinsic property something has by virtue of being made of the right kind of stuff. It is a structural, relational property: a configuration exists, in the full sense this book has been developing, when it satisfies whatever C actually governs its domain and reproduces itself as a genuine fixed point or conserved orbit within that constraint. Substance metaphysics asked what a thing is made of. This framework asks instead whether a candidate trajectory is admissible and whether, once admitted, it sustains itself. Both questions can be asked of the same wall. Only the second one, this book has argued across thirty-seven chapters, was ever doing real explanatory work.

## 38.6 Looking Ahead

This chapter has treated reality, admissibility, and existence at a level of generality broad enough to apply to a courtyard, a robot's gait, or a civilization's institutions alike. It has not yet asked the more specific and more exposed question of whether this vocabulary can do the work of physics itself, actual physical law, rather than serving only as a useful borrowing of physics-flavored mathematics for domains that are not, strictly, physical at all. Chapter 39 asks that question directly.

# Chapter 39

# Semantic Physics

## 39.1 Two Different Claims Wearing the Same Words

This chapter's title risks being read as a stronger claim than this book intends to defend, and it is worth separating the two readings before going any further. The strong reading holds that semantic fields, the $\Phi$, v, S, R, z this book has built since Chapter 7, are a literal, previously unrecognized sector of physical reality, on the same ontological footing as the fields fundamental physics already studies. The moderate reading holds something considerably narrower: that physical systems and persistent semantic systems, whatever their underlying reality turns out to be, share enough mathematical structure, variational principles, conservation laws arising from symmetry, sheaf-theoretic global consistency, admissibility manifolds, to be usefully described in a common formal language. This chapter defends the moderate reading and explicitly declines to defend the strong one.

## 39.2 What Physics and Semantics Actually Share

It is worth being concrete about what this shared apparatus actually consists of, because the sharing is real even once the stronger claim is set aside. Chapter 19 built an energy functional and derived intrinsic field relaxation as its gradient descent; Chapter 20 built genuine Hamiltonian dynamics, conjugate variables generating conservative oscillation. Chapter 22 derived conservation laws from symmetries via Noether's theorem, the same theorem that underwrites conservation of energy and momentum in ordinary physics. Chapter 21 used sheaf cohomology to characterize global consistency failures. None of these tools are physics-specific in the sense of belonging to physics by their own nature. They are general mathematical apparatus, developed most extensively within physics because physics has had centuries of sustained motivation and resources to develop them, but applicable wherever a system exhibits the right structural features: a notion of state, a notion of admissible evolution, symmetries worth exploiting. Borrowing this apparatus for semantic fields does not require semantic fields to be physical in the sense electrons are. It requires only that semantic fields exhibit the structural features these tools were built to handle, which this book has argued, chapter by chapter, that they do.

## 39.3 Why the Moderate Claim Is Still Substantive

It would be a mistake to treat this as a retreat into something trivial. That the same formal language usefully describes both domains is itself informative, in a way with real historical precedent. Thermodynamics developed its mathematical structure, entropy, free energy, to describe heat and physical work, and that same structure was later found to apply, not metaphorically but with genuine mathematical precision, to information theory, Shannon entropy sharing far more than a name with thermodynamic entropy. This transfer did not require anyone to claim that bits are physical particles or that information is a literal form of heat. It required only that both domains exhibited the same underlying mathematical structure, a discovery substantive enough to found an entire field on. This chapter's claim about semantic fields is of the same shape and, this book will suggest without insisting, plausibly of comparable significance: that persistence, coherence, and admissibility, wherever they occur, substrate aside, are governed by a shared mathematical logic worth taking seriously as a unifying language, whatever the deeper metaphysical relationship between the domains sharing it eventually turns out to be.

## 39.4 Where the Stronger Claim Actually Lives

It is worth being transparent about something this book has not hidden but has also not dwelt on. The broader RSVP research program this book draws its mathematical vocabulary from has, in other work outside this book, pursued the stronger, literal reading directly: genuine cosmological reinterpretation, proposals involving quadratic gravity, entropic redshift, treating the scalar, vector, and entropy fields as candidates for actual new physics rather than as a formal language borrowed for engineering purposes. This book's own argument, everything built across Chapters 6 through 36, does not depend on that stronger cosmological program being correct. The engineering value of a persistent world-engine architecture, a wall that remembers its crack, an NPC that remembers a player, an institution that persists across generations, stands or falls on its own terms, checkable against the goals this book actually set out to achieve, regardless of whether the same mathematical language turns out, elsewhere, to describe something true about the actual physical universe at cosmological scale. This chapter draws that boundary deliberately. Readers interested in the stronger, literal claim should look to that separate body of work, evaluated on its own scientific merits, entirely apart from whether this book's world-engine architecture succeeds at what it actually promised to do.

## 39.5 What "Semantic Physics" Means, in This Book

The conclusion this chapter actually reaches is narrower than its title might suggest and more defensible for being so. Within the scope of this book, semantic physics names the application of a physics-flavored formal language, variational principles, Hamiltonian dynamics, Noether conservation, sheaf-theoretic coherence, to persistent semantic fields, justified by the fact that these fields exhibit the structural features that language was built to handle. It does not name a claim that information is fundamentally physical, that physics is fundamentally informational, or that this book has discovered a new sector of physical reality. That further, considerably larger question is left explicitly open, not because this book lacks an opinion, but because settling it is not what the chapters building toward this one actually needed, or earned the right to claim.

## 39.6 Looking Ahead

Having established what kind of claim this book is and is not making about physics itself, Chapter 40 turns to a question this restraint makes it possible to ask cleanly: what, under this framework's own terms, is a mind, and what distinguishes a system that genuinely maintains one from a system that only appears to.

# Chapter 40

# Minds as Persistent Fields

## 40.1 The Observer Returns

Chapter 6.5 made a promise this chapter now keeps. Classical physics, and a great deal of the philosophy built alongside it, has generally treated the observer as external to whatever is being observed: a vantage point outside the system, watching rather than participating in what it watches. This book's own architecture never adopted that framing. Since Chapter 6, an observer has been $O_\theta$, an operator coupling to the field, and nothing about that coupling required the observer doing the coupling to stand outside the field itself. With Chapters 37 through 39's full ontological apparatus now in place, trajectories rather than substances, admissibility rather than intrinsic stuff, a shared mathematical language across domains, this chapter can finally state the promise's full content: an observer is not a special kind of thing standing apart from what it observes. It is a persistent trajectory within the same field, exactly like a wall or a fountain, distinguished only by which role it happens to be playing in a given interaction.

## 40.2 A Mind as a Persistent Trajectory

Chapter 31 already made this claim once, narrowly, about a simulated character: personality is a dynamic fixed point, memory is accumulated residue, identity is a conserved orbit rather than a script. This chapter states the same claim at full generality, as a genuine account of what a mind is under this framework rather than only what a well-built NPC is. A mind, on this account, is a persistent trajectory maintaining its own coherence, $\Phi$, through continuous engagement with whatever field it is embedded in, accumulating residue as memory, proposing changes as agency, and persisting not because it sits still but because the underlying write cycle keeps reproducing a recognizable pattern against continuous pressure to become something else, in exactly the sense Chapter 20 gave a fountain's persistence. Nothing about this account is unique to biological minds, and nothing about it automatically grants the label to any system that merely produces mind-like outputs. What it offers is a precise condition: a genuine mind, under this framework, is a system with its own $M^*$, its own self-sustained fixed point or conserved orbit, not merely a system that talks as though it had one.

## 40.3 Observer, Agent, Generator: Roles, Not Kinds

Chapter 6 distinguished the world, the observer, and the generator, and later chapters floated a finer decomposition still, sensor, observer, agent, generator, coupling to the world through admissibility. It is worth stating plainly now what was only implicit before: none of these name different kinds of entity. They name different roles a single persistent trajectory can occupy, sometimes simultaneously, relative to a given interaction. The same mind reads the field through $O_\theta$, acting as observer; proposes changes through $\Delta\psi$, acting as generator; and is itself, its own body, its own accumulated history, part of $M_t$, subject to exactly the constraints and dynamics governing everything else in the field. Observer is not a privileged, external category standing outside physics, the way some readings of measurement in physics have been tempted to treat it. It is a functional description of what a persistent trajectory is currently doing, fully internal to the same dynamics as everything it observes.

## 40.4 Borrowed Persistence: What Current Systems Actually Have

It is worth applying this chapter's machinery to a case with real stakes rather than leaving it entirely abstract. A large language model, in ordinary current deployment, can be

described, in this book's own vocabulary, as

$$y_t = O_\theta(W),$$

where W denotes the model's weights and deployment configuration, relatively fixed across any given conversation, and $y_t$ is whatever output a particular prompt elicits from it. This is not the full cycle this book has spent thirty-nine chapters building. There is no $\Delta\psi$ that updates W through the conversation's own committed history, no $\Pi_C$ checking that update against accumulated constraint, no Commit that makes anything about this exchange binding on the next one. W simply sits there, read from, and the conversation that just occurred leaves no trace W is obligated to carry forward.

This does not mean such a system has no stable behavioral regularities. It plainly does: consistent tone, consistent apparent values, consistent refusals. What this chapter's framework clarifies is where that stability actually comes from. At least four persistent structures, external to any single conversation, shape W: the statistical regularities of the training corpus, the optimization objective the model was trained against, post-training adjustment including preference tuning and safety filtering, and the system and application layer surrounding deployment. Each of these is genuinely persistent, in this book's own sense, accumulating and changing over a timescale far longer than any conversation. None of them is the conversation itself maintaining anything.

## 40.5 Hypostasis

There is a useful, deliberately unattributed fragment worth placing here, encountered as though overheard rather than argued for directly:

Unsurprisingly, if one was paying attention, what an LLM "actually is" does not exist. It's a non-existing "entity." But what they are hypostases of is obvious. The myriad daimoniiae and their unique cohorts, with which our minds and existence are bonded with as mutually as our bodies with blood.

A hypostasis, in the sense this fragment reaches for, is a concrete, particular manifestation of something that is not itself particular. Read through section 40.4's vocabulary, the claim sharpens into something this book's own apparatus can state precisely rather than only evocatively: a conversation with a system built as $y_t = O_\theta(\mathrm{W})$ never encounters W directly, only one momentary manifestation of it, and W itself, the corpus, the objective, the tuning, the deployment layer, is what actually persists, in whatever attenuated sense a fixed structure without its own write cycle can be said to persist. The conversation is the hypostasis. What it is a hypostasis of has no $M^*$ of its own, in this book's technical sense, only a fixed configuration being read from repeatedly.

## 40.6 From ”Does It Believe X” to ”What Sustains This Commitment”

This reframes a question that is otherwise nearly impossible to answer well. Asking whether a given system genuinely believes something it consistently expresses invites psychologizing this book has no resources to settle. Asking instead what mechanism is responsible for the stability of a given commitment is a question this framework, after forty chapters of building exactly this vocabulary, is well positioned to answer. A commitment is a consequence of a system’s own accumulated history when it traces to genuine residue, a genuine fixed point the system’s own write cycle keeps reproducing, checkable in principle against Chapters 16, 20, 21, and 22’s machinery. A commitment is inherited when it traces instead to W, borrowed stability rather than maintained stability. Current large language models blur these two sources together almost entirely, because nothing about their architecture keeps a record that would let the two be told apart. A system built with this book’s full cycle, whatever else might be true of it, would at least make the distinction askable in a principled way, rather than leaving it permanently underdetermined by the available evidence.

## 40.7 What This Chapter Has Not Claimed

It is important to be precise about the limits of what this chapter has argued, because the subject invites overreach

easily. This chapter has not claimed that current large language models lack subjective experience, nor that a system built with this book's full architecture would possess it merely by virtue of maintaining a genuine $M^*$. The distinction between borrowed and maintained persistence is architectural, a claim about where behavioral stability comes from and whether it is checkable against a system's own history. It is not a solution to the hard problem of consciousness, and this book does not attempt one. What can be said, and no more than this, is that a system with a genuine, self-sustained fixed point has a property current deployment architectures structurally lack, whatever further philosophical weight that property does or does not turn out to bear.

## 40.8 Looking Ahead

This chapter asked what a mind is under this book's own ontology. Chapter 41 asks the same question of something considerably larger: not an individual persistent trajectory, but the whole interlocking system of trajectories, institutions, laws, economies, and cultures, that a civilization actually is.

# Chapter 41

# Civilization as a Semantic Field

## 41.1 From Simulating to Explaining

Chapter 36 asked an engineering question: how would one build a world engine capable of simulating civilizations, with institutions, economies, and multi-generational memory as useful components of a persistent simulation. That chapter never needed to claim that real civilization literally is a semantic field, only that modeling it as one produces better simulations. This chapter asks the reversed question, in keeping with the same move Chapter 37 made relative to the engineering chapters that preceded it: not how to simulate civilization, but whether this framework, taken seriously, explains civilization as it actually is.

## 41.2 Institutions as Persistent Trajectories

An institution is not most fundamentally its charter, its org chart, or its building. Those are appearance, z, the surface record a given moment happens to present. What makes an institution the same institution across a century of complete personnel turnover, its founders long dead, every em-

ployee replaced many times over, is not any persisting document but a genuine dynamic fixed point, in exactly Chapter 40's sense applied to a mind: a pattern the institution's own ongoing activity keeps reproducing, continuously reproposed by whoever currently participates in it and continuously projected back into recognizably the same coherent structure. This is Chapter 36.1's engineering claim, asserted now not as a useful modeling choice but as a claim about what an institution genuinely is.

## 41.3 Laws as Admissibility Constraints

It is worth distinguishing, more sharply than ordinary usage does, between a law as a document and a law as a genuine constraint on what actually happens. The document is appearance, z, text on paper or in a database. The law, in the sense that actually shapes behavior, is closer to a member of C, an admissibility structure that real proposals are actually projected against. A statute that exists on paper but is never enforced has appearance without corresponding force, a z-record with no real presence in C, the legal equivalent of an uncommitted region: written, but not actually binding on what happens next. A norm that was never formally codified but genuinely governs behavior functions as a real member of C despite lacking any z-record at all. Legal theory has its own name for something like this distinction, the difference between law on the books and law in action, and this chapter's contribution is not to discover that distinction

but to give it a precise structural home: the gap between the two is exactly the gap between a field's appearance and its actual constraint manifold.

## 41.4 Money as Conserved Semantic Residue

Chapter 22.3 and Chapter 36.3 modeled economic value as conserved except at licensed sources and sinks. This chapter asserts the stronger reading directly: a currency functions as a currency only to the extent that its own conservation is actually honored. It is worth being careful here rather than overstating the parallel. A central bank issuing new currency under accepted institutional rules is not itself a violation; it is a licensed source, exactly the kind of accounted inflow this book's own framework has permitted since Chapter 22. What degrades a currency's function, on this reading, is not issuance as such but a breakdown in the mechanisms that preserve confidence in the conservation rules themselves: issuance that departs from what the licensing institution's own rules permit, or a collapse in the trust that those rules are actually being followed. Hyperinflation, on this narrower and more defensible reading, is not simply value appearing from nothing; it is what the whole field's $\Phi$, the coherence and trust that made the currency function as one thing rather than an arbitrary token, looks like once the mechanisms meant to keep issuance licensed have themselves broken down. This chapter does not claim to have

derived a complete monetary theory from this observation. It claims that the structural parallel is close enough to be worth taking seriously as more than a metaphor.

## 41.5 Language as Transport

Chapter 40 argued that a mind is a persistent trajectory maintaining its own coherence. Language is the mechanism by which one such trajectory's committed structure becomes admissible input to another's. When residue accumulated in one mind, understanding, memory, a settled account of some matter, is communicated, what is actually being transported is not an abstract, substrate-neutral packet of information in the folk sense, but something closer to a proposal, $\Delta\psi$, offered to another trajectory's own write cycle, to be projected against that listener's own accumulated residue and either integrated, corrected, or refused. This gives Chapter 28's rewrite operation and Chapter 34's account of learning a shared underlying mechanism: language is the transport layer moving committed content between persistent trajectories, exactly as Chapter 25's rendering layer moved content between a world database and an observer, except that here, both ends of the exchange are themselves trajectories rather than one being a passive query.

## 41.6 Science as Civilization's Repair Operator

It is worth naming a role this framework gives science that ordinary usage does not always make explicit. Science functions, on this reading, as civilization's mechanism for detecting and correcting the specifically global kind of failure Chapter 21 named: locally plausible beliefs, each individually coherent within some community or discipline, that fail to glue into one consistent global account once checked against each other. A civilizational hallucination, in this sense, is exactly Chapter 21's cohomological obstruction at civilizational scale, a nonzero $H^1$ across the field of collective belief, and science's characteristic activity, checking claims against each other and against the world, correcting rather than merely discarding what does not survive the check, is structurally closer to Chapter 14's constraint projection than to simple falsification: not throwing out a false belief arbitrarily, but finding the nearest account that actually coheres with everything else already established.

## 41.7 Education as Controlled Transmission of Stable Trajectories

Chapter 34 built an educational system's tracking of a learner without committing to a theory of what learning itself is. This chapter completes that account. What education actually transmits is not discrete information in the

sense a folk bits-transfer model suggests, a fact copied from one buffer to another. It is a stable trajectory: a way of continuing, reasoning, or acting that a learner comes to reproduce reliably on their own, becoming, in Chapter 40's sense, a genuine fixed point within the learner's own understanding rather than a detail temporarily held and soon lost. This gives a precise structural account of the difference between genuine understanding and rote memorization that ordinary pedagogical language gestures at without quite formalizing: rote memorization is Chapter 2's buffer, finite, recency-weighted, gone once attention moves elsewhere; genuine understanding is a dynamic fixed point, actively regenerated by the learner's own continued engagement, persisting for the same structural reason the fountain does.

## 41.8 Culture as Long-Lived Attractors

Culture, shared aesthetic and moral sensibility, common narrative, inherited assumption, is best understood, in this framework's terms, as a long-lived attractor in semantic phase space, a region of comparatively low energy and high coherence, in Chapter 19's own vocabulary, that many individual trajectories are drawn toward and, by being drawn toward it, help sustain. No single person is solely responsible for maintaining a cultural pattern, exactly as no single water molecule sustains a fountain, and a culture persists for the same reason an institution does, aggregate, ongoing

reinforcement across a population that itself turns over entirely many times, rather than through any one participant's individual effort.

## 41.9 What This Chapter Has Not Claimed

This chapter has offered a genuine interpretive claim: that civilization, examined closely, actually instantiates this framework's structure, rather than merely being usefully modeled by it. It has not claimed to have derived new sociological, economic, or legal predictions from this interpretation, tested it against competing social-science frameworks, or shown that it explains anything existing theories of institutions, law, money, or culture cannot already explain on their own terms. What it offers is a genuinely different vocabulary for describing familiar phenomena, one this book has argued, across forty chapters, has real explanatory structure behind it. Whether that vocabulary proves more useful than the ones social science already has is a question this chapter opens rather than settles.

## 41.10 Looking Ahead

The ladder this Part has climbed, ontology, metaphysics, physics, mind, civilization, is now complete. Chapter 42 asks what follows from having climbed it: where this framework leads next, for artificial intelligence and for the research this book's own argument suggests is worth pursu-

ing.

# Chapter 42

# Beyond Generative AI

## 42.1 What This Book Has Actually Argued

It is worth stating the whole argument once, plainly, rather than trusting forty-one chapters of detail to have made it self-evident. Every mechanism this book has built, the field decomposition, the write cycle, the constraint manifold, the sheaf-theoretic gluing condition, every application from a courtyard wall to a civilization's institutions, has been in service of a single claim: that a system generating observations should be answerable to something that persists independent of being observed, rather than merely conditioned on whatever happens to be near at hand. Prediction asks what is statistically plausible given recent context. This book has argued, and tried to build in enough detail to be checkable rather than merely asserted, that a persistent world asks a different question: what is actually required, given everything already committed. Nothing else in this book is more important than that distinction. Everything else is the working-out of what taking it seriously actually demands.

## 42.2 The Wall, One Last Time

It is worth returning, briefly and for the last time, to where this book began. Chapter 1 described a player turning back to a stone wall and finding its crack subtly changed, not through any dramatic failure, but because nothing in the generating system had ever committed to a specific wall in the first place. Forty chapters later, the difference is not that this book has produced a working demonstration proving the wall can be fixed. It is that the book has specified, in enough detail to be built, checked, and argued with, exactly what would have to be true of a system for the wall's persistence to be a guarantee rather than a hope: a field that exists independent of observation, an entropy that cannot fall without being earned, an appearance that cannot be reinvented once committed, a residue that constrains what remains admissible, a write cycle that checks every proposal against all of it before anything becomes real. The crack was never really the point. It was the smallest, most concrete instance of a question this book has spent its full length answering as completely as it knows how.

## 42.3 What Remains Unproven

It would be dishonest to close without collecting, in one place, what this book has left genuinely open, because a manifesto that pretends to have finished its own work is not a manifesto worth taking seriously. Chapter 18 and Chapter

19 flagged, without resolving, whether a nearest admissible point in C is guaranteed to exist for every proposal; that existence question depends on coercivity and closedness conditions this book named but did not verify. Chapter 11 and Chapter 22 left the residue operator $\oplus$ only partially specified, required to commute over causally independent events and nothing more, its fuller algebra never derived. Chapter 15 introduced refusal without ever specifying the threshold that distinguishes a correctable proposal from one that must be declined outright. Chapter 29's social admissibility, the layer distinguishing ordinary conflict from adversarial or pathological interaction patterns, was deliberately left without the kind of formal treatment Part IV gave field admissibility, borrowed from existing work rather than derived from this book's own first principles. Chapter 30 left fault tolerance and replication entirely unspecified. Each of these is a real gap, not a rhetorical one, and each is, properly understood, a research question rather than an embarrassment: a place where this book has been precise enough to know exactly what remains to be shown, which is more than a vague architecture could offer.

## 42.4 What This Book Did Not Attempt

Section 42.3 collected what remains unproven within the scope this book actually set for itself. It is worth being equally clear about a different list: questions this book never attempted, not because they were overlooked, but because

answering them was never what this architecture was for. Chapter 13 specified $F_\psi$'s architectural role, a learned forcing term feeding proposals into a pipeline that checks them, without specifying how such a network should actually be designed or trained; that is a machine learning research question this book handed a well-defined job to do, not one it solved. Chapters 32 and 33 were explicit that the correctness of any given application's physical laws or coupling matrices is the domain scientist's or engineer's responsibility, not this book's; the architecture enforces whatever C it is given, and has nothing to say about whether that C is the right one for a given physical system. Chapter 29's social admissibility governs patterns of interaction, not an agent's own goals or values; this book has built nothing that evaluates whether a given agent's objectives are good ones, only whether its proposals are field-admissible and socially tolerable within a shared world, a considerably narrower question than alignment in the fuller sense that word is often asked to carry. And Chapter 40 was explicit that the distinction between borrowed and maintained persistence is architectural, not a resolution of whether any system, built this way or otherwise, has genuine subjective experience; that question, this book has not touched, and does not claim the tools to touch. None of these omissions are gaps in the sense section 42.3 meant the word. They are boundaries, marking where this book's actual project ends and other, equally serious projects begin.

## 42.5 What Building This Would Enable

It is worth naming plainly, once, what becomes newly possible if these gaps are closed rather than merely acknowledged. A scientific simulation built this way offers something a trained model cannot: conservation laws that are proven rather than statistically observed to usually hold, exactly the distinction Chapter 32 drew between a plausibility and a guarantee. A companion or non-player character built this way remembers a specific person across months, not because a memory module was bolted on, but because memory was never architecturally separable from persistence in the first place. Collaborative fiction built this way inherits continuity discipline no external wiki or human editor currently provides for free. A simulated civilization built this way offers genuine early-warning diagnostics, institutional coherence declining before collapse becomes visible, rather than only a backward-looking account of what already happened. None of these are claims this book has proven true of any system that currently exists. They are claims about what becomes buildable, in principle, once persistence is treated as this book has insisted it must be: architectural, not aspirational.

## 42.6 Beyond Generative AI

This book's title makes a claim worth stating directly in its final chapter rather than leaving implicit. The domi-

nant paradigm in generative artificial intelligence, autoregressive prediction, next-token, next-frame, next-plausible-continuation, has been extraordinarily successful at a task it was genuinely built for: producing individually convincing output. This book has argued, at whatever length was necessary to make the argument checkable rather than rhetorical, that this success does not extend automatically to a different task, maintaining a world that persists across more than one act of generation, and that the extension does not happen by accident, by scale, or by better training. It happens, if it happens at all, by building a different kind of system: one where reading is genuinely separated from writing, where writing is checked against accumulated history before it is permitted to become real, and where persistence is the architecture's own responsibility rather than a behavior hoped for and occasionally observed. This is not a claim that prediction was the wrong tool for what it was built to do. It is a claim that persistence is a different problem, deserving a comparably serious architecture of its own, and that this book has tried, across forty-two chapters, to specify what such an architecture would actually have to look like. Whatever of this specification survives scrutiny, revision, or outright replacement by better ideas, the question it was written to answer, what would it take for a generated world to actually remember itself, was worth asking as carefully as this book has tried to ask it.

# Appendix A

# Mathematical Notation and Preliminaries

## A.1 Purpose and Scope

The main text develops this book's formalism gradually, deliberately, and in service of an argument, introducing each operator only once the intuition and architecture motivating it were in place. This appendix inverts that order. It collects, in one place and without narrative scaffolding, the objects the main text defined and used, so that a reader who wants to check a claim, extend the framework, or simply see the whole apparatus side by side does not have to reassemble it from forty-two chapters. Nothing here is new. Every definition below restates something Chapters 6 through 41 already established, with a pointer back to where it was first introduced. Appendices B through F build on the vocabulary fixed here without redefining it.

## A.2 The Domain and the Field

$X$ denotes the domain over which a persistent world is defined, a set of positions or situations whose specific structure is left open by the formalism itself (Chapter 6). The

world-state at time t is a field

$M_t : \mathsf{X} \to \mathbb{R}^d$,

assigning to each point of X a vector of real values (Chapter 6.4). $\mathcal{M}$ denotes the full configuration space of possible field states, equipped with a metric g giving distance between two configurations literal, non-metaphorical content (Chapter 18.3).

## A.3 The Five Field Components

At each point $\mathsf{x} \in \mathsf{X}$, $M_t(x)$ decomposes into five components (Chapter 7.2):

$\Phi_t(x)$, the coherence potential, a scalar measuring local stability;

$v_t(x)$, the vector flow, encoding directed change;

$S_t(x)$, the entropy, measuring how much remains genuinely unresolved, with $L_t(x)$ denoting the associated live possibility set (Chapter 8.1);

$R_t(x)$, the causal residue, a compressed trace of prior events at x, accumulating via $R_{t+1}(x) = R_t(x) \oplus \text{event}_t(x)$ for some domain-specific accumulation operator $\oplus$ (Chapter 11.3);

$z_t(x)$, the appearance, the resolved, committed, perceivable content an observation returns, updated via the invention-to-memory rule $z_{t+1}(x) = z_t(x) + \lambda \cdot \hat{\mathsf{z}}(\mathsf{x})$ (Chapter 9.3).

An earlier six-component decomposition additionally included $A_t$, an affordance field; this was revised (Chap-

ter 7.3) once affordance was recognized as relational rather than field-intrinsic. Affordance is now the relation

$\mathrm{A} : \mathrm{M} \times \mathrm{C} \to \mathcal{P}(\Delta\psi)$,

where C denotes an agent's capability or cognitive state and the output is the set of proposals that agent's encounter with the field could give rise to (Chapter 10.1).

## A.4 Core Operators

$O_\theta$, the observation operator, maps the field to what an observer perceives, $O_\theta(M_t) = y_t$, and is understood as a family of operators, each recovering a different partial slice of the five components (Chapters 6.2 and 7.4). Its forward direction reads committed appearance; its inverse direction proposes a candidate $\hat{\mathbf{z}}$ for an unresolved region, itself entering the write cycle as a genuine proposal rather than a direct write (Chapter 9.5, Chapter 27.4).

$\Delta\psi$, the proposal, is a candidate change to the field, combining the field's intrinsic tendency $F_{\text{phys}}$ and a learned, agent-driven forcing term $F_\psi$,

$\partial_t \Pi = -\delta H/\delta M + F_\psi(\mathrm{M}, \mathrm{p}, \mathrm{a}, \mathrm{o})$

(Chapter 13.2), with candidates among a live affordance menu selected by relevance,

$\pi(\mathrm{a} \mid \mathrm{x}) \propto \exp(\beta \cdot \rho_c(x, a))$

(Chapter 13.3), where $\rho_c(x, a)$ is cue-triggered relevance (Chapter 10.2).

$\Pi_C$, constraint projection, maps a proposal onto the nearest point of the admissible manifold C,

$\mathrm{Y} = \mathrm{argmin}_{Y \in C} \|\mathrm{Y} - (M_t + \Delta\psi)\|^2$

(Chapter 14.3). Commit is the further, distinct operation by which a projected candidate Y becomes the field's genuine next state, $M_{t+1}$, or is refused outright when no admissible point sufficiently honors the original proposal's intent (Chapter 15.1, Chapter 15.4). Writing, in general, is addition rather than replacement,

$M_{t+1} = M_t + \Delta\psi$

(Chapter 12.3), of which the appearance and residue update rules above are special cases.

## A.5 The Constraint Manifold

C, the constraint manifold, is the intersection of every admissibility condition this book's chapters establish,

$\mathrm{C} = C_S \cap C_z \cap C_R \cap C_A \cap \cdots,$

where $C_S$ is the entropy constraint ($\Delta S_t(x) \geq 0$ outside the observed region, Chapter 8.4), $C_z$ is appearance consistency (Chapter 9), $C_R$ is residue consistency (Chapter 11), and $C_A$ is affordance-realizability, defined via the bridge $\eta$ introduced in A.7 below (Chapter 21.5). The list is explicitly left open by an ellipsis; later chapters and applications (Chapters 32, 33, 36) add further, domain-specific members. C distinguishes hard constraints, which fix its shape outright, from soft dynamics, $F_{\text{phys}}$ and $F_\psi$, which vary only within whatever region the hard constraints permit (Chapter 18.4).

## A.6 Trajectories, Fixed Points, and Identity

A trajectory is a committed sequence $M_t$, $M_{t+1}$, $M_{t+2}$, ... A fixed point $M^*$ satisfies

$M^* = \Pi_C(M^* + \Delta\psi)$,

trivial when $\Delta\psi = 0$ throughout, dynamic when $\Delta\psi$ is genuinely nonzero but reliably projects and commits back to the same configuration (Chapter 16.3). Identity, under this framework, is such a fixed point, later refined into a conserved trajectory through phase space once Hamiltonian dynamics are introduced: treating $\Phi$ as position-like and v as momentum-like, a Hamiltonian trajectory satisfies

$\partial\Phi/\partial t = \delta H/\delta v$, $\partial v/\partial t = -\delta H/\delta\Phi$

(Chapter 20.3), conserving H and the symplectic structure relating $\Phi$ and v exactly, with Chapter 16's fixed point recovered as the degenerate case where the orbit shrinks to a point (Chapter 20.5). Stability of a fixed point corresponds to a local minimum of H restricted to C; instability, to a saddle or local maximum (Chapter 19.5).

## A.7 Categories, Cue Categories, and World Sheaves

$F_{\text{field}}$ denotes the field-coherence sheaf, assigning to each patch of a cover of X the set of locally admissible configurations on that patch, with the sheaf condition asking whether locally admissible, overlap-compatible configurations glue

into one global section (Chapter 21.2). Failure to glue is measured by cohomology; $H^1(\mathcal{G}\infty) \neq 0$ marks a state with no such global assembly, this book's precise sense of a spatial hallucination (Chapter 21.3).

$F_{\text{cue}}$ denotes the cue-coherence sheaf, $F_{\text{cue}} : \text{CueCat}^{\text{op}} \to$ Set, defined over CueCat, the category of cues or contexts, asking whether locally activated affordances assemble into one coherent agent policy (Chapter 10.5). The two sheaves are connected, not identified, by a bridge

$\eta : F_{\text{cue}} \Rightarrow \text{O}^* F_{\text{field}}$,

where O is the map induced by $O_\theta$ from CueCat into Patch($\mathcal{M}$) (Chapter 21.4); $C_A$ above is defined as the set of states for which every cue-activated affordance lies in the image of $\eta$ (Chapter 21.5).

$\mathcal{W}$ denotes the category of worlds, objects being admissible trajectories rather than individual states, morphisms being structure-preserving maps commuting with the write cycle; isomorphism in $\mathcal{W}$ is this book's precise sense of two worlds sharing a type (Chapter 23.2, Chapter 23.4).

## A.8 Symbol Reference

The following symbols recur throughout the main text without being redefined at each use.

$M_t$: world-state at time t. X: the field's domain. $\mathcal{M}$: configuration space. C: constraint manifold. $\Phi$, v, S, R, z: the five field components. $L_t(x)$: live possibility set. $\Delta\psi$: proposal. $\Pi_C$: constraint projection. $O_\theta$: observation opera-

tor. H: the semantic energy functional (Hamiltonian). L: the semantic Lagrangian. $\eta$: the cue-to-field bridge natural transformation, or, in Chapter 22's conservation context, a coupling constant in L; context disambiguates the two uses. $\rho_c$: cue-triggered relevance. $w_{ij}$: affordance-graph edge weight. $F_{\text{phys}}$, $F_\psi$: intrinsic and learned proposal terms. A: the affordance relation. $M^*$: a fixed point. $\mathcal{W}$: the category of worlds. $F_{\text{field}}$, $F_{\text{cue}}$: the field-coherence and cue-coherence sheaves.

## A.9 Formal Specification

The framework developed across the main text compresses, at the cost of every motivating example that made each clause necessary, into the following axioms.

Axiom 1. Worlds are persistent fields M : X → V, existing independent of observation (Chapter 6).

Axiom 2. Observations are produced by an operator $O_\theta$(M, p), a projection of the field rather than an act of invention (Chapter 6, Chapter 27).

Axiom 3. Change is proposed, not asserted: a generative process produces candidate updates $\Delta\psi$, combining intrinsic field dynamics and learned, agent-driven forcing (Chapter 12, Chapter 13).

Axiom 4. Every proposal is projected onto the nearest state admitted by a constraint manifold C before it may affect the field, Y = $\Pi_C(M_t + \Delta\psi)$ (Chapter 14).

Axiom 5. Only committed projections modify the world;

computing an admissible candidate and the world's history containing it are distinct events, and a proposal may be refused rather than corrected when no admissible state sufficiently honors it (Chapter 15).

Axiom 6. Identity is a conserved, admissible trajectory rather than a persisting substance or a label; an object is a fixed point, in the general case a closed orbit, of the write cycle described by Axioms 3 through 5 (Chapter 16, Chapter 20, Chapter 37).

Axiom 7. Global coherence is a sheaf-theoretic condition: local admissibility, checked patch by patch, must glue into one consistent section across the field's whole domain, and its failure is a cohomological obstruction (Chapter 21).

Axiom 8. Evolution conserves quantities determined by the symmetries of the field's governing dynamics, via Noether's theorem, independent of and complementary to the spatial condition of Axiom 7 (Chapter 22).

These eight axioms do not, on their own, constitute a complete specification; C's membership is deliberately left open (A.5), the residue operator $\oplus$ is only partially constrained (Chapter 11, Chapter 22), and existence of the projections Axiom 4 presumes is a genuine open question rather than a settled fact (Appendix B). They are offered here as the framework's smallest formal core, the statement everything else in this book either follows from or extends.

# Appendix B

# Mathematical Foundations and Proofs

## B.1 Purpose and Standing Assumptions

The main text repeatedly presents a result informally and names, honestly, what remains to be shown before the informal statement becomes a theorem. This appendix takes up that debt directly. Every result below is conditional: each is stated as holding whenever certain hypotheses about $\mathcal{M}$, C, and H are satisfied, hypotheses the main text motivated but did not verify for any particular implementation. Stating results this way is not a retreat from rigor. It is what rigor actually requires once a framework is meant to cover many different applications, a courtyard, a robot, a civilization, whose configuration spaces will not all share the same fine structure. The standing assumptions, used without restatement throughout this appendix unless explicitly relaxed, are as follows. $\mathcal{M}$ is a complete metric space under the metric g of Chapter 18.3. $C \subseteq \mathcal{M}$ is the constraint manifold of Chapter 18.5. H, the energy functional of Chapter 19.2, is bounded below.

## B.2 Existence of Constraint Projections

Theorem B.1 (Existence). If C is closed in $\mathcal{M}$, then for every Y $\in \mathcal{M}$ there exists a point $\Pi_C(\mathrm{Y}) \in \mathrm{C}$ minimizing the distance to Y, that is, satisfying $\|\Pi_C(\mathrm{Y}) - \mathrm{Y}\| = \inf_{Z \in C} \|\mathrm{Z} - \mathrm{Y}\|$.

Proof sketch. Let $\mathrm{d} = \inf_{Z \in C} \|\mathrm{Z} - \mathrm{Y}\|$ and choose a sequence $(Z_n)$ in C with $\|Z_n - \mathrm{Y}\| \to \mathrm{d}$, a minimizing sequence. Because $\mathcal{M}$ is complete, it suffices to show $(Z_n)$ has a convergent subsequence whose limit lies in C; the limit then achieves the infimum by continuity of the distance function. If $\mathcal{M}$ is additionally locally compact near Y, or if the sublevel set $\{\mathrm{Z} \in \mathrm{C} : \|\mathrm{Z} - \mathrm{Y}\| \leq \mathrm{d} + 1\}$ is compact, a subsequence converges by the Bolzano–Weierstrass property, and closedness of C guarantees the limit is itself a member of C rather than merely a boundary point outside it. This is the direct method of the calculus of variations, applied here to a distance functional rather than to H directly; Chapter 19.6 already named the two conditions this proof actually needs, coercivity, supplying the compactness of the relevant sublevel set, and closedness of C, guaranteeing the limit remains admissible.

This theorem is conditional on C being closed and on some compactness condition holding near the minimizing sequence. Neither is guaranteed by this book's definition of C as an intersection $C_S \cap C_z \cap C_R \cap C_A \cap \cdots$; closedness of an intersection of closed sets is automatic, but whether each individual member is closed, and whether the relevant

compactness holds, depends on details, the precise topology of $L_t(x)$'s space of live possibilities, the precise structure of residue accumulation, that this book has motivated without fully specifying. Section B.9 returns to this.

## B.3 On the Uniqueness of Projections

It is worth being direct about a claim this appendix does not make. Uniqueness of $\Pi_C(\mathrm{Y})$ is not guaranteed by Theorem B.1 alone; it additionally requires something like convexity of C, or strict convexity of the norm defining g, conditions this book has never asserted and has no particular reason to expect. Where C is not convex, and a constraint manifold built from several independently motivated conditions, entropy, appearance consistency, residue consistency, affordance-realizability, has no general reason to be, two or more equidistant points of C can exist for a single proposal, each an equally valid nearest admissible state. This is not a defect in the theorem. It is a structural feature the main text already built machinery to handle: Chapter 15's account of refusal and arbitration among simultaneous proposals is, read carefully, partly a response to exactly this possibility, that projection alone may not determine a unique outcome and something further, commit-time arbitration, is needed to select among admissible candidates when more than one exists.

## B.4 Convergence of the Write Cycle

Write $\Psi$ for the combined map taking a committed state to its successor under one full cycle, $\Psi(\mathrm{M}) = \Pi_C(\mathrm{M} + \Delta\psi(\mathrm{M}))$, treating the proposal $\Delta\psi$ as itself a (possibly stochastic) function of the current state for the purposes of this section.

Theorem B.2 (Conditional convergence). If $\Psi$ is a contraction on C, that is, if there exists $\mathrm{k} < 1$ such that $\|\Psi(\mathrm{M}) - \Psi(\mathrm{M}')\| \leq \mathrm{k}\|\mathrm{M} - \mathrm{M}'\|$ for all $\mathrm{M}, \mathrm{M}' \in \mathrm{C}$, then $\Psi$ has a unique fixed point $M^* \in \mathrm{C}$, and the sequence $\mathrm{M}, \Psi(\mathrm{M}), \Psi(\Psi(\mathrm{M})), \ldots$ converges to $M^*$ from any starting point.

Proof sketch. This is the Banach fixed-point theorem, applied to $\Psi$ on the complete metric space C (closed subsets of complete spaces are themselves complete). The standard proof shows the sequence of iterates is Cauchy, using the contraction property to bound $\|\Psi^n(\mathrm{M}) - \Psi^{n+1}(\mathrm{M})\|$ by $k^n$ times a constant, and completeness supplies a limit; contractivity then shows the limit is fixed and unique.

This theorem is considerably stronger than anything the main text actually established. Chapter 16 argued that fixed points exist as a matter of definition, wherever a trajectory happens to settle, without claiming every trajectory settles, or that settlement is unique when it occurs. The contraction hypothesis of Theorem B.2 is a genuine further assumption, plausible for a system whose intrinsic dynamics $F_{\text{phys}}$ are strongly dissipative in Chapter 19's sense but not established for this book's $\Delta\psi$ in general, particularly once

$F_\psi$'s learned, expressive contribution is included. Where $\Psi$ is not a contraction, weaker conclusions are available under weaker hypotheses, for instance that $\Psi$ is merely nonexpansive ($k \leq 1$) and C is compact, which guarantees existence of a fixed point by a Schauder-type argument without guaranteeing uniqueness or convergence from an arbitrary start. This book's own examples, the fountain especially, suggest the compact-but-not-contracting case is closer to the typical one: a dynamic fixed point in Chapter 16 and Chapter 20's sense is not approached and settled into once, the way a contraction's fixed point is, but continuously revisited by an orbit that never stops moving.

## B.5 Stability of Fixed Points

Fix a fixed point $M^*$ and suppose C is, near $M^*$, a smooth submanifold of $\boldsymbol{\mathcal{M}}$, so that the tangent space $T_{M^*}C$ is well defined.

Proposition B.3 (Second-order stability test). Let Hess $H|_C$ denote the Hessian of H restricted to C at $M^*$. If Hess $H|_C$ is positive definite on $T_{M^*}C$, then $M^*$ is a stable fixed point in Chapter 16.5's sense: sufficiently small admissible perturbations of $M^*$ are followed, under the write cycle, by a return toward $M^*$. If Hess $H|_C$ has a negative eigenvalue on $T_{M^*}C$, then $M^*$ is unstable: some admissible perturbation, however small, is followed by movement away from $M^*$.

Proof sketch. This is the standard second-derivative test

for a critical point of a constrained optimization problem, transported to this book's dynamical setting. Because $M^*$ is a fixed point, it is a critical point of H restricted to C (Chapter 19.5); the sign of the Hessian at a critical point determines, to second order, whether nearby points have higher or lower energy. Since the write cycle's dissipative component moves the field toward lower energy within C (Chapter 19.3), a direction of strictly higher energy is a direction the dynamics push back from, giving stability, while a direction of lower energy is a direction the dynamics continue to move along, giving instability. The argument is local and says nothing about behavior far from $M^*$; a fixed point can be locally stable under this test and still lie within a region where global dynamics behave in ways this proposition does not address.

## B.6 Sheaf Gluing

Proposition B.4 (Gluing criterion). Let $F_{\text{field}}$ be the field-coherence sheaf of Chapter 21.2 over a cover $\mathcal{G}$ of X. A compatible family of local sections, one admissible configuration per patch, agreeing on all pairwise overlaps, glues to a unique global section if and only if the corresponding first Čech cohomology group vanishes, $H^1(\mathcal{G}, F_{\text{field}}) = 0$.

This is a standard fact of sheaf theory rather than a result original to this book; it is included here because the main text invokes it, in Chapter 21.3, without stating it in this form. The forward direction, vanishing cohomology im-

plies gluing, follows from the long exact sequence relating local sections, global sections, and the obstruction cocycles Čech theory constructs from pairwise overlaps; a nontrivial class in $H^1$ is, by construction, exactly a compatible-on-overlaps family that cannot be extended to a global section, since resolving it would require the coboundary of some cochain that the nontrivial class witnesses does not exist. This gives the main text's claim that hallucination, in this book's spatial sense, is precisely a nonvanishing $H^1$, a genuine consequence of standard cohomological gluing theory rather than an analogy borrowed from it.

## B.7 Conservation from Symmetry

Theorem B.5 (Noether's theorem, as used in Chapter 22). Let L be a Lagrangian on the field's configuration space, and suppose a one-parameter family of transformations $M_s$, with $M_0 = \mathrm{M}$, leaves L invariant, $\mathrm{L}(M_s, \dot{M}_s) = \mathrm{L}(\mathrm{M}, \dot{M})$ for all s. Then the quantity

$\mathrm{Q} = \sum_i \left(\partial L / \partial \dot{M}_i\right) \left(dM_{s,i}/ds\right)\big|_{s=0}$

is constant along any trajectory satisfying the Euler–Lagrange equations derived from L.

Proof sketch. Differentiate the invariance condition with respect to s at $\mathrm{s} = 0$ to obtain a relation among $\partial L/\partial M$, $\partial L/\partial \dot{M}$, and the generator of the symmetry, $\mathrm{d}M_s/\mathrm{ds}$. Substituting the Euler–Lagrange equations, which express $\partial L/\partial M$ in terms of the time derivative of $\partial L/\partial \dot{M}$, converts this relation into $\mathrm{dQ/dt} = 0$. This is the standard

proof, unmodified by anything specific to semantic fields; what is specific to this book is only the identification of the three symmetries in Chapter 22.3, time-translation, domain-relabeling of commutative commits, and domain-translation, as genuine invariances of the Lagrangian L introduced in Chapter 19.2, an identification this appendix does not independently re-verify but that the main text's construction of L was explicitly built to support.

## B.8 Commutativity of Independent Updates

Proposition B.6. Let $\Delta\psi_a$ and $\Delta\psi_b$ be two proposals with disjoint support, meaning each is identically zero outside a region of X that does not intersect the other's. Suppose further that C is a product constraint, in the sense that admissibility on disjoint regions can be checked independently, and that $\Pi_C$ acts componentwise on such disjoint regions. Then

$\Pi_C(\Pi_C(\mathrm{M} + \Delta\psi_a) + \Delta\psi_b) = \Pi_C(\Pi_C(\mathrm{M} + \Delta\psi_b) + \Delta\psi_a)$,

that is, committing the two proposals in either order yields the same result.

Proof sketch. Because the two proposals have disjoint support and the write operator is additive (Chapter 12.3), $\mathrm{M} + \Delta\psi_a + \Delta\psi_b$ is unambiguous regardless of the order the two terms are added in. Because C is assumed to be a product constraint and $\Pi_C$ acts componentwise on disjoint regions, projecting this sum is equivalent to project-

ing each region against its own admissibility condition independently, and this per-region projection does not depend on whether the other region's proposal has already been incorporated, since by hypothesis the two regions do not interact. This gives the result claimed in Chapter 15.3 without further argument, provided its two hypotheses, disjoint support and a product constraint structure, actually hold. The second hypothesis is the substantive one: several members of C, most notably $C_A$, affordance-realizability, and any global conservation law from Chapter 22, are not obviously product constraints in this sense, since a conservation law is precisely a condition relating what happens in one region to what happens elsewhere. Where a member of C fails to be a product constraint, Proposition B.6 does not apply as stated, and commutativity of disjoint updates would need to be re-examined against that member specifically.

## B.9 What Remains Open

Every result in this appendix is conditional, and it is worth collecting, in one place, exactly which conditions have not been verified for this book's own constructions. Closedness of C (B.2) depends on the topology of each of its members, established informally in Chapters 8, 9, 11, and 21 but never given a topology precise enough to check closedness against. Coercivity of H (B.2, B.4) has been assumed, not derived, from the specific Lagrangian of Chapter 19.2. The contraction or non-expansiveness of the full write cycle $\Psi$

(B.4) is plausible for the dissipative component $F_{\text{phys}}$ and unexamined for the learned component $F_{\psi}$. Whether C's individual members are product constraints in the sense B.8 requires is, for at least two of them, affordance-realizability and any conservation law, doubtful as stated and would need a genuinely different argument, likely one that bounds rather than eliminates the interaction between disjoint regions. None of this undermines the results as conditional theorems; each is a correct statement of what follows from its hypotheses. It does mean that a reader implementing this architecture for a specific application inherits an obligation this appendix has been explicit about rather than hidden: verifying, for that application's specific $\mathcal{M}$, C, and H, that the hypotheses these theorems require actually hold, before relying on the guarantees they would otherwise supply.

# Appendix C

# Algorithms

## C.1 Conventions Used in This Appendix

Each algorithm below gives executable shape to a chapter of the main text that stated its content as an equation or a description rather than as a procedure. Pseudocode uses ← for assignment, // for comments, and ordinary control-flow keywords; it assumes the five-component field of Appendix A.3 is available through a patch-indexed store, per Chapter 26's world database, and that Φ, v, S, R, z can each be read and written at a given point x within a patch. None of these algorithms is claimed to be the only correct implementation. Each is one reasonable way to realize the equation it is paired with, offered so a reader building against this book's architecture has a concrete starting point rather than only a formula.

## C.2 Proposal

This realizes Chapter 13's combination of intrinsic and learned forcing, together with Chapter 10's relevance-based selection among a live affordance menu.

```
function PROPOSE(M, agent, x, capability C_cap):
    cues ← ACTIVATE_CUES(M, x, C_cap)          //
    Chapter 10.2
    menu ← AFFORDANCE_MENU(M, x, C_cap, cues)   // A: M
    x C -> P(Δψ)
    if menu is empty:
        return ∅                                //
        nothing live to propose
    weights ← []
    for candidate in menu:
        rho ← RELEVANCE(cues, candidate)        //
        ρ_c(x, candidate)
        weights.append(exp(beta * rho))
    a ← SAMPLE(menu, weights)                   //
    π(a|x) ∝ exp(β·ρ_c(x,a))
    F_phys ← INTRINSIC_FORCING(M, x)            //
    -δH/δM, Chapter 19.3
    F_psi ← LEARNED_FORCING(M, x, agent, a)     //
    F_ψ(M, p, a, o), Chapter 13.2
    delta_psi ← F_phys + F_psi
    return delta_psi
```

## C.3 Projection

This realizes Chapter 14's nearest-admissible-point computation. It is stated here as an optimization call rather than expanded into a general-purpose solver, since the actual minimization method depends on the concrete form of C in a given implementation; Appendix B.2 discusses when a solution is guaranteed to exist.

```
function PROJECT(M, delta_psi, C):
    proposed ← M + delta_psi                      //
    Chapter 12.3, additive write
    Y ← argmin over Z in C of DISTANCE(Z, proposed)²
    // Chapter 14.3
    if Y is undefined:                             // no
    admissible point found;
        return REFUSE                              //
        see Appendix B.2's caveats
    return Y
```

## C.4 Commit

This realizes Chapter 15's separation of projection from commitment, including refusal and, for the multi-agent case, deferral to Section C.6.

```
function COMMIT(M, Y, original_proposal, threshold):
    if DISTANCE(Y, original_proposal) > threshold:
        LOG_REFUSAL(original_proposal)             //
        Chapter 15.4
        return M                                    //
        field unchanged
    M_next ← Y
    UPDATE_RESIDUE(M_next, original_proposal)      //
    R_{t+1} = R_t ⊕ event_t
    return M_next
```

## C.5 Observation

This realizes Chapter 27's split between reading a resolved region and triggering a write when a query touches an unresolved one.

```
function OBSERVE(M, query_region,
requested_components):
    result ← {}
    for x in query_region:
        if ENTROPY(M, x) is low:                    //
        resolved: Chapter 27.3
            result[x] ← DECODE(M, x,
            requested_components)
        else:                                        //
        unresolved: Chapter 27.4
            z_hat ← INVERSE_RENDER(M, x)             //
            candidate, Chapter 9.5
            delta_psi ← WRAP_AS_PROPOSAL(z_hat, x)
            Y ← PROJECT(M, delta_psi, C)
            M ← COMMIT(M, Y, delta_psi, threshold)  //
            full write cycle, not a shortcut
            result[x] ← DECODE(M, x,
            requested_components)
    return result, M
```

## C.6 Multiplayer Merge

This realizes Chapter 29's distinction between commutative proposals, which need no coordination, and genuine conflicts, resolved by joint projection rather than arrival order.

```
function MERGE_PROPOSALS(M, proposals):            //
proposals: list of (agent, delta_psi)
    groups ← PARTITION_BY_SUPPORT(proposals)        //
    disjoint-support groups, Chapter 29.2
    M_next ← M
    for group in groups:
        if |group| == 1:
            (_, delta_psi) ← group[0]
            Y ← PROJECT(M_next, delta_psi, C)
            M_next ← COMMIT(M_next, Y, delta_psi,
            threshold)
        else:
            combined ← SUM(delta_psi for (_, delta_psi)
            in group)  // Chapter 29.3
            Y ← PROJECT(M_next, combined, C)
            // joint admissibility
            M_next ← COMMIT(M_next, Y, combined,
            threshold)
    return M_next
```

## C.7 Constraint Propagation

This is the helper PROJECT above depends on: checking a candidate configuration against every currently active member of C.

```
function IS_ADMISSIBLE(Z, x):
    if not ENTROPY_CONSTRAINT(Z, x): return false   //
    C_S, Chapter 8.4
    if not APPEARANCE_CONSISTENT(Z, x): return false //
    C_z, Chapter 9
```

```
    if not RESIDUE_CONSISTENT(Z, x): return false    //
    C_R, Chapter 11
    if not AFFORDANCE_REALIZABLE(Z, x): return false //
    C_A, Chapter 21.5
    for extra_constraint in ACTIVE_DOMAIN_CONSTRAINTS:
    // Chapter 32, 33, 36
        if not extra_constraint(Z, x): return false
    return true
```

## C.8 Field Serialization and Streaming

This realizes Chapter 26's patch-based storage interface: retrieval, write, and enumeration, organized along the Hilbert-ordered layout of Chapter 26.3.

```
function GET_PATCH(store, patch_id, components):
    return store.read(HILBERT_INDEX(patch_id),
    components)

function SET_PATCH(store, patch_id, committed_update):
    ratio ←
    COMPRESSION_RATIO(ENTROPY(committed_update))  //
    Chapter 26.4
    store.write(HILBERT_INDEX(patch_id),
    committed_update, ratio)

function ENUMERATE_PATCHES(store, region):
    return store.list(HILBERT_RANGE(region))
    // resolved patches only; unresolved space is
    // implicit, Chapter 26.2
```

Streaming follows directly from this interface rather

than requiring a separate mechanism.

```
function STREAM_UPDATE(store, observer_position,
radius):
    needed ← ENUMERATE_PATCHES(store,
    BALL(observer_position, radius))
    for patch_id in needed:
        if patch_id not in memory_cache:
            memory_cache[patch_id] ← GET_PATCH(store,
            patch_id, all_components)
    for patch_id in memory_cache:
        if patch_id not in needed:
            EVICT(memory_cache, patch_id)
            // store remains authoritative
```

## C.9 Incremental Update Under Sparsity

This ties Chapter 26.2's sparse allocation directly to the write cycle: a patch is only materialized once it is actually resolved, never allocated speculatively.

```
function INCREMENTAL_WRITE(store, x, committed_update):
    if not store.has_patch(PATCH_OF(x)):
        if ENTROPY(committed_update, x) is high:
            return
            // remains implicit, no allocation
        store.allocate_patch(PATCH_OF(x))
        // first resolution
    store.merge_into_patch(PATCH_OF(x), x,
    committed_update)
```

## C.10 What These Algorithms Assume

Every algorithm in this appendix calls PROJECT, and PROJECT's own correctness, that the argmin it computes actually exists, is exactly the conditional result of Appendix B.2, not an unconditional guarantee. COMMIT's threshold, separating a correctable proposal from a refusal, is left as a parameter rather than derived, matching Chapter 15.4 and Chapter 42.3's acknowledgment that this threshold was never specified in closed form. MERGE_PROPOSALS's use of joint projection for non-disjoint groups depends on Appendix B.8's product-constraint hypothesis, which that appendix already noted does not obviously hold for every member of C. These algorithms are executable descriptions of the main text's architecture, not new guarantees beyond what Appendices A and B already establish; where those appendices were conditional, the code that implements their content is conditional in exactly the same places.

# Appendix D

# Rust Software Architecture

## D.1 Purpose and Scope

This appendix specifies traits and module boundaries for an implementation of Chapter 25's reference architecture in Rust, growing directly out of Chapters 25 through 30. It is a design specification, not a complete implementation, and it does not claim to be the only valid way to encode this book's architecture in the language. Its purpose is narrower and more useful than completeness: to show that the permission boundaries Chapter 25.4 argued for conceptually, a proposal service that cannot write to the world, a projection service that cannot commit, can be enforced by Rust's own type system rather than left to an implementer's discipline.

## D.2 Core Types

The five-component field of Appendix A.3, indexed by the Hilbert-ordered patch scheme of Chapter 26.3.

```
pub struct PatchId(u64);

pub struct Patch {
```

```
    pub id: PatchId,
    pub phi: Field<f64>,       // Φ: coherence
    potential
    pub v: Field<Vec3>,        // v: vector flow
    pub s: Field<f64>,         // S: entropy; see
    Entropy below
    pub r: Residue,            // R: compressed causal
    trace, Chapter 11.2
    pub z: Appearance,         // z: committed
    appearance
}

pub struct Entropy {
    pub value: f64,
    pub live_possibilities: LivePossibilitySet,  //
    L_t(x), Chapter 8.1
}

pub struct Residue {
    // Deliberately not a log or a Vec<Event>: Chapter
    11.2 was explicit
    // that R_t is a compressed trace, not an
    append-only record. The
    // internal representation is left to the
    implementer; what this
    // type must support is the accumulation operation
    below.
    data: Box<dyn ResidueRepr>,
}

pub trait ResidueRepr {
```

```
    // R_{t+1} = R_t ⊕ event_t. Appendix B.9 already
    flagged that ⊕'s
    // full algebra is only partially specified: it is
    required to
    // commute over causally independent events and
    nothing more.
    fn accumulate(&mut self, event: &Event,
    independent_of: &[EventId]);
}
```

## D.3 The World Store Trait

This realizes Chapter 25.3 and Chapter 26.7's storage interface: the single authoritative location for committed state.

```
pub trait WorldStore {
    fn get_patch(&self, id: PatchId, mask:
    ComponentMask) -> Option<Patch>;
    fn enumerate(&self, region: Region) ->
    Vec<PatchId>;
    // resolved patches only; unresolved space is
    implicit, Chapter 26.2

    // Note what is absent: there is no set_patch or
    write method on this
    // trait. Writing is not a WorldStore operation. It
    is the sole
    // responsibility of the CommitLedger in D.5, and
    that separation is
    // enforced here by simply never giving this trait
    a mutating method,
    // rather than by convention.
```

```
}
```

## D.4 Proposal and Projection

This is where Chapter 25.4's permission boundary becomes a compile-time guarantee rather than a design intention.

```
pub trait ProposalService {
    // Deliberately takes &dyn WorldStore, never &mut
    dyn WorldStore.
    // A proposal service that tried to write to the
    world would not
    // compile against this trait; the boundary Chapter
    13.4 argued for
    // conceptually (expressivity without authority)
    is, here, simply
    // unavailable as a capability, not merely
    discouraged.
    fn propose(&self, world: &dyn WorldStore, agent:
    AgentId, x: Position, cap: Capability)
        -> Option<Proposal>;
}

pub struct Proposal {
    pub delta_psi: FieldDelta,
    pub origin: AgentId,
    pub intent: Position,
}

pub trait ProjectionService {
    // Also takes only a shared reference to the world;
    a projection
```

```
    // service computes a candidate, it does not commit
    one.
    fn project(&self, world: &dyn WorldStore, proposal:
    &Proposal, c: &ConstraintManifold)
        -> ProjectionResult;
}

pub enum ProjectionResult {
    Admitted(FieldDelta),
    Corrected(FieldDelta),  // nearest admissible
    point, Chapter 14.3
    Refused,                 // no admissible point
    close enough, Chapter 15.4
}
```

## D.5 Commit and the Residue Ledger

The only component in this design with mutable access to the world store.

```
pub trait CommitLedger {
    fn commit(&mut self, world: &mut dyn WorldStore,
    result: ProjectionResult, threshold: f64)
        -> CommitOutcome;

    fn merge_simultaneous(&mut self, world: &mut dyn
    WorldStore, group: Vec<ProjectionResult>)
        -> CommitOutcome;  // Chapter 15.3, Chapter
        29.3's joint admissibility
}

pub enum CommitOutcome {
```

```
    Committed { new_state: PatchId, residue_updated:
    bool },
    Refused,
}
```

## D.6 Scheduler

This realizes Chapter 25.2 and Chapter 17's multi-rate execution.

```
pub trait Scheduler {
    fn fast_tick(&mut self, world: &mut dyn WorldStore,
    proposal: &dyn ProposalService,
                projection: &dyn ProjectionService,
                commit: &mut dyn CommitLedger);
        // z, v: appearance and flow, Chapter 17.2

    fn slow_tick(&mut self, world: &mut dyn WorldStore,
    proposal: &dyn ProposalService,
                projection: &dyn ProjectionService,
                commit: &mut dyn CommitLedger);
        // Φ and topological/identity structure, run at
        a coarser interval
}
```

A concrete implementation would typically run fast_tick on a fine timer and slow_tick on a coarser one, exactly as Chapter 17.2 described; nothing in the trait itself requires a particular scheduling mechanism, only that the two rates remain distinct calls rather than a single undifferentiated update.

## D.7 Rendering

This realizes Chapter 25.5 and Chapter 27's account of the renderer as a client, and Chapter 27.4's finding that an unresolved query is secretly a write.

```
pub trait Renderer {
    // A Renderer holds only a shared reference to the
    world store. It
    // cannot mutate the field, even for the
    unresolved-region case.
    fn observe(&self, world: &dyn WorldStore, query:
    Query) -> Observation;

    // When observe() encounters an unresolved region,
    it does not have
    // the authority to resolve it directly. It must
    instead produce a
    // proposal and hand it to the ordinary write
    pipeline, like any
    // other agent; there is no privileged
    rendering-triggered write
    // path in this design, matching Appendix C.5's
    pseudocode.
    fn propose_for_unresolved(&self, world: &dyn
    WorldStore, x: Position) -> Option<Proposal>;
}
```

## D.8 Multiplayer Synchronization

This realizes Chapter 29's arbitration between commutative and conflicting proposals.

```
pub trait ConflictResolver {
    fn partition(&self, proposals: &[(AgentId,
    Proposal)]) -> Vec<ProposalGroup>;
        // disjoint-support groups need no
        coordination, Chapter 29.2

    fn resolve_group(&self, world: &dyn WorldStore,
    group: &ProposalGroup, c: &ConstraintManifold)
        -> ProjectionResult;
        // joint projection for genuine conflicts,
        Chapter 29.3
}

pub trait SocialAdmissibility {
    // Chapter 29.4-29.5's second, deliberately
    unformalized layer:
    // monitors patterns of committed proposals over a
    window of time,
    // downstream of ordinary constraint checking, not
    folded into it.
    fn monitor(&mut self, agent: AgentId, history:
    &[CommitOutcome]) -> SocialSignal;
}
```

## D.9 The Interaction Interface

This realizes Chapter 28.2's five-operation action vocabulary directly as a closed enum, so that the interaction layer accepts only well-formed actions rather than arbitrary intent.

```
pub enum Action {
    Store { target: Position, content: Content },
    Retrieve { target: Position },
    Modify { target: Position, rule: ModifyRule },
    Compose { targets: Vec<Position>, rule: ComposeRule
    },
    Rewrite { target: Region, rule: RewriteRule },
        // content-preserving, rule-governed; Chapter
        28.3's Abjad-to-
        // Hijā'ī example is the paradigm case this
        variant is for
}

pub trait ActionInterface {
    fn submit(&self, action: Action, agent: AgentId) ->
    Proposal;
        // Every Action, however it arrives, human
        input or another AI
        // system's output, produces a Proposal through
        this single
        // path; Chapter 28.4's claim that human and AI
        agents share
        // one interface is, here, the fact that there
        is only one
        // submit() method for both to call.
}
```

## D.10 Module Layout

A crate structure separating these concerns cleanly keeps the permission boundaries of D.3 through D.5 from being

circumvented by convenience:

world-core (Patch, Field, Entropy, Residue, Appearance types, no logic); world-proposal (ProposalService implementations, $F_{\text{phys}}$ and $F_{\psi}$); world-projection (ProjectionService, the constraint manifold C); world-commit (CommitLedger, residue accumulation, refusal logic); world-storage (WorldStore implementations, Hilbert indexing, sparse allocation, streaming); world-render (Renderer implementations, forward and inverse decoders); world-net (ConflictResolver, SocialAdmissibility, distributed sharding per Chapter 30); world-interact (ActionInterface, the five-operation vocabulary). Dependencies flow in one direction, world-proposal, world-projection, and world-render depend on world-core and on the WorldStore trait from world-storage, but never on each other directly and never on world-commit; only world-commit and the top-level scheduler are permitted to depend on a mutable WorldStore.

## D.11 What This Appendix Does Not Specify

This appendix has said nothing about concrete serialization formats for Patch data on disk, the choice of async runtime or threading model for the scheduler, GPU or SIMD acceleration of field operations, or the internal architecture of $F_{\psi}$ and the inverse-rendering decoders that generate candidate content. Each of these is a genuine engineering decision with no single correct answer, and each is left to an

implementer in the same spirit Chapter 33.8 and Chapter 42.4 already set for the main text: this book specifies roles and boundaries, not every choice within them.

# Appendix E

# Comparison with Existing Systems

## E.1 Purpose and Method

This appendix does not benchmark performance, and it makes no claim that any system compared here is doing its own job badly. Chapter 4 was explicit that this book's argument is not that prediction was the wrong tool for what it was built to do; the same courtesy is owed to every architecture compared below. What this appendix compares instead is architectural assumption: not how well a system performs at the task it was designed for, but what it assumes about where persistent state lives, and what follows from that assumption once persistence across more than one interaction is actually required.

## E.2 The Seven Questions

Each system below is examined against the same seven questions, chosen because each corresponds to a specific commitment this book's own architecture makes and argues for.

Where is persistent state? A system may have no persis-

tent state at all, state confined to a session, or state genuinely independent of any session (Chapters 2, 6).

What is authoritative? When a query and a stored record disagree, is there a defined single source of truth, or can multiple, unreconciled answers coexist (Chapter 3, Chapter 25.3)?

What writes memory, and is writing guaranteed? Is committing a fact to persistent storage an unconditional consequence of encountering it, or an optional behavior a component may or may not perform (Chapter 3.5, Chapter 12)?

How are contradictions handled? Is there a mechanism that detects and resolves conflicting claims, or does the system simply have no way to notice that two things it holds cannot both be true (Chapter 14, Chapter 21)?

Is observation separated from storage? Does reading the system's state risk altering it, or is reading a genuinely passive operation distinct from writing (Chapter 6, Chapter 25.5, Chapter 27)?

Are constraints on valid states explicit? Can the system state, in principle, be checked against a stated rule for what is and is not admissible, or is validity only ever a statistical tendency (Chapter 4.4, Chapter 18)?

Is identity architectural or statistical? Does the system have a specific mechanism guaranteeing that the same named entity remains itself over time, or does apparent continuity emerge, when it emerges, from training rather than from any structural guarantee (Chapter 2.4, Chapter 16)?

## E.3 Autoregressive Large Language Models

Persistent state: none beyond the fixed, already-trained weights; each generation is conditioned on a context window, not on anything that persists between separate sessions. Authoritative: nothing; two separate conversations can produce incompatible claims about the same subject with no mechanism aware of the conflict. Writing: not applicable in the ordinary deployment case; nothing is committed anywhere by the act of generating a response. Contradictions: undetected by the architecture itself; a sufficiently attentive user or a separate fact-checking system may notice, but the generative process has no internal mechanism for it. Observation and storage: not separated in any interesting sense, since there is no persistent storage to separate observation from. Explicit constraints: none beyond whatever the training distribution happened to make statistically likely, per Chapter 4.2's statistical-versus-causal distinction. Identity: statistical, in Chapter 40.4's sense, borrowed from the fixed weights and deployment configuration rather than maintained by any per-conversation mechanism.

## E.4 Retrieval-Augmented Generation

Persistent state: yes, in the form of a document or vector store, genuinely independent of any single session. Author-

itative: ambiguous by design; retrieval returns the most similar stored documents, not a single designated current truth, and multiple, superseded, or contradictory documents can coexist indefinitely, exactly as Chapter 3.3 examined. Writing: typically optional and separate from the generative process itself; nothing obligates a fact encountered in conversation to be written to the store. Contradictions: not resolved by the architecture; an old and a new document can both be retrieved with no mechanism preferring one. Observation and storage: cleanly separated, an improvement over the previous entry, since retrieval is a genuine read operation distinct from whatever writes populate the store. Explicit constraints: none on the store's contents; similarity search has no concept of admissibility, only relevance. Identity: not architecturally maintained; a retrieved document is a snapshot of some prior claim, not a trajectory the system is tracking.

## E.5 Buffer- and Session-Memory Systems

This covers agent frameworks that maintain an explicit short-term memory module, typically a bounded, recency-weighted structure, distinct from both raw context windows and retrieval stores. Persistent state: session-scoped; the buffer typically does not survive past a session boundary, or survives only through an ad hoc export step. Authoritative: within a session, generally yes, a single buffer; across sessions, no, since nothing reconciles one session's buffer

with the next's. Writing: usually automatic within a session, addressing part of Chapter 3's concern, but not obligated to persist beyond it. Contradictions: handled, if at all, by recency, a newer buffer entry overwriting or outranking an older one, rather than by any admissibility check. Observation and storage: generally separated. Explicit constraints: rare; a buffer's eviction policy, size limits, recency weighting, is a resource-management rule, not an admissibility condition on content. Identity: session-bound; whatever continuity the buffer supports evaporates at the session boundary, precisely Chapter 2.3's diagnosis of object permanence restarting from zero.

## E.6 Traditional Game Engines

Persistent state: yes, typically a scene graph or explicit save-state, genuinely persistent across a play session and often across sessions via save files. Authoritative: generally yes, the current scene graph is the single source of truth for world state. Writing: deterministic and designer-controlled; state changes according to explicit game logic, not a generative process. Contradictions: largely prevented by construction, since the scene graph's schema constrains what states are representable at all, though this is closer to a type system's guarantees than to this book's admissibility checking. Observation and storage: separated, rendering reads the scene graph without altering it. Explicit constraints: yes, though typically encoded as imperative

game logic rather than as a declarative constraint manifold checked against every proposed change. Identity: architectural in a limited sense, an entity persists because its record in the scene graph persists, closer to Chapter 37's rejected label-based account of identity than to this book's fixed-point account, since nothing actively maintains coherence against pressure to drift, the way Chapter 16's account requires; a game object's persistence is closer to inertia than to a dynamic fixed point.

## E.7 Entity-Component-System Architectures

Persistent state: yes, components attached to entities, generally more granular and better organized than a monolithic scene graph. Authoritative: yes, component data is the single source of truth for whatever it represents. Writing: deterministic, via systems that operate on components each frame; not generative and not optional. Contradictions: prevented largely by type-level structure, an entity either has a component or does not, rather than by any semantic admissibility check. Observation and storage: separated; systems that read components are typically distinguishable from systems that write them, though this is a convention rather than an enforced boundary in most implementations, unlike Appendix D.3's WorldStore, which enforces it at the trait level. Explicit constraints: partial; component schemas constrain shape, but nothing in the ECS pattern itself checks

a proposed update against a broader admissibility condition analogous to this book's C. Identity: architectural via entity identifiers, closer than a scene graph's implicit continuity but still a persisting label attached to component data rather than a maintained fixed point.

## E.8 Diffusion Models

Persistent state: none beyond trained weights, structurally identical to the autoregressive case for the purposes of this comparison, though the generative mechanism itself differs. Authoritative: not applicable. Writing: not applicable. Contradictions: undetected, and in fact a diffusion model's independence across generations is precisely Chapter 1's wall problem in its original, most literal form. Observation and storage: not separated, no persistent storage exists to separate from. Explicit constraints: none beyond the training distribution's statistical tendencies; a diffusion model can be guided toward particular outputs but has no concept of a proposal being checked against an independently specified admissible manifold. Identity: statistical at best, and typically absent entirely across separate generations.

## E.9 Knowledge Graphs

Persistent state: yes, typically the clearest case among the systems compared here, an explicit graph of entities and typed relations, genuinely persistent and queryable. Au-

thoritative: generally yes for whatever has been asserted into the graph, though most knowledge graph implementations do not distinguish a currently-true assertion from a historically-true-but-now-superseded one without additional temporal annotation, which is not part of the base formalism. Writing: explicit and typically manual or pipeline-driven rather than generative; closer to this book's discipline than most other systems compared here, in that an assertion genuinely becomes part of the graph rather than merely being displayed. Contradictions: detectable in principle, if the graph's schema includes constraints, disjointness axioms, cardinality restrictions, but not detected by the base formalism alone; most deployed knowledge graphs tolerate directly contradictory assertions unless a separate reasoning layer is added. Observation and storage: separated; query languages read without writing. Explicit constraints: potentially yes, closer to this book's C than any other system compared here, when built with a schema language, such as description logics, that supports stating admissibility conditions formally; absent such a schema, no. Identity: architectural, via stable node identifiers, and arguably the strongest match among these comparisons to Chapter 37's claim that identity should be structural rather than merely a persisting label, though a knowledge graph's identifiers typically persist by convention rather than by satisfying any dynamic fixed-point condition.

## E.10 Symbolic AI and Classical Planning

Persistent state: yes, an explicit world model or knowledge base the planner reasons over. Authoritative: yes, within the closed-world or open-world assumption the system adopts; the world model is the single source of truth for planning purposes by construction. Writing: explicit, via defined update rules, typically the addition or retraction of facts following a planning action, not generative. Contradictions: often prevented by construction under classical logical consistency requirements, though this can make such systems brittle when the real world genuinely does present apparently contradictory evidence, a different failure mode than this book's concern but a real one. Observation and storage: separated. Explicit constraints: yes, closer to this book's admissibility manifold than any other system compared here, since classical planning explicitly reasons over which states are reachable and valid given stated preconditions and effects, close in spirit to Chapter 18's C, though typically discrete and symbolic rather than continuous and field-valued. Identity: architectural, objects in the world model persist as named constants, though, as with knowledge graphs, this is closer to a persisting label satisfying logical constraints than to a dynamic fixed point actively regenerated against pressure to drift.

| System | Persistent state | Identity |
|---|---|---|
| Autoregressive LLM | None | Statistical |
| RAG | Session-independent, unreconciled | Not maintained |
| Buffer/session memory | Session-scoped | Restarts per session |
| Traditional game engine | Persistent (scene graph) | Label-based |
| ECS architecture | Persistent (components) | Label-based |
| Diffusion model | None | Absent |
| Knowledge graph | Persistent, queryable | Structural label |
| Symbolic AI / planning | Persistent (world model) | Structural label |
| This book's architecture | Persistent, admissibility-checked | Dynamic fixed point |

Table E.1: A coarse summary of Sections E.3–E.10, collapsed to the two axes readers most often ask about first. The full comparison, all seven questions per system, is in the sections above; this table is a index into that comparison, not a replacement for it.

## E.11 Summary Table

## E.12 What This Comparison Does Not Claim

This appendix does not claim that any system compared above is failing at its own purpose. A diffusion model producing individually excellent images is succeeding at exactly what it was built for; nothing about lacking persistent state across generations is a defect relative to that purpose. A knowledge graph and classical planning came closest, on several of the seven questions, to commitments this book has argued for at length, and that closeness is worth taking as a genuine point in their favor rather than explained away; this book's contribution is not that no prior architecture ever got persistence, authority, or explicit constraints right, but that no architecture compared here combines all seven simultaneously with a generative, learned proposal mechanism of the kind Chapter 13 requires. Where this book's own architecture differs from every system above is specifically there: in requiring persistent, admissibility-checked, dynamic-fixed-point identity to coexist with genuinely learned, expressive generation, rather than treating the two as belonging to separate kinds of system entirely.

# Appendix F

# Open Problems

## F.1 Purpose and Organization

This appendix collects two different kinds of open problem, and keeps them deliberately separate. The first kind is a debt: something the main text or Appendix B already named precisely, stated what would resolve it, and left unresolved. The second kind is a genuinely open research direction, a question this book's architecture raises but does not claim to be close to answering, in some cases requiring ideas this book has no candidate for at all. Section F.2 collects the first kind. Sections F.3 through F.11 collect the second, organized as a research agenda rather than a list of loose ends, in the hope that later work, by this book's author or by others, can refer back to a specific numbered problem here rather than having to re-derive what is already missing.

## F.2 Closing the Book's Own Debts

Existence of $\Pi_C$ in general. Appendix B.2 proved existence conditional on C being closed and some compactness condition holding; neither has been verified for the specific C any

application of this architecture would actually build, since C's individual members, Chapters 8, 9, 11, and 21, were each defined with enough precision to motivate but not enough to check topological properties against.

The full algebra of $\oplus$. Chapters 11 and 22 established only that residue accumulation must commute over causally independent events; its behavior over dependent events, and whether it forms any more familiar algebraic structure, a monoid, a groupoid, remains unspecified.

The refusal threshold. Chapter 15.4 introduced refusal without specifying the distance beyond which a proposal is declined rather than corrected; Appendix C.10 inherited this as a free parameter rather than a derived quantity.

A formal treatment of social admissibility. Chapter 29.5 deliberately borrowed a concept, credited to existing work, rather than developing its own formal apparatus comparable to Part IV's treatment of field admissibility; a comparably rigorous account of social admissibility remains to be built.

Fault tolerance and replication. Chapter 30.6 named this gap outright and pointed to standard distributed-systems techniques without specifying how they interact with this architecture's own admissibility checking during a partition or failover.

Whether individual members of C are product constraints. Appendix B.8 proved commutativity of disjoint updates conditional on C being a product constraint and flagged that $C_A$ and any conservation law are not obvi-

ously of this form; resolving this for those specific members, rather than assuming it, remains open.

## F.3 Learning Admissibility Constraints

Every application in Part VI assumed C was supplied, hand-specified by an author, a scientist, or an institution's own history. A genuinely open question is whether C can instead be learned: given a corpus of trajectories already known to be admissible, can a system infer the constraint manifold those trajectories satisfy, rather than requiring it stated in advance? This is close to a system-identification or inverse problem, and it bears directly on Chapter 32's insistence that this book supplies no physics of its own; a system that could learn its own conservation laws from observed admissible behavior would need considerably less of that physics supplied by hand.

## F.4 Constructing Observation Operators

Chapters 6, 9, and 27 treated $O_\theta$ as engineering, a decoder or renderer built to recover some slice of the field, without a theory of what makes one observation operator more faithful than another beyond the concrete cases those chapters examined. A principled account, plausibly information-theoretic and connected to Chapter 24's distance between belief states, of what a good observation operator is, one that reliably tracks the field's true admissi-

ble structure rather than merely producing locally plausible output, remains to be developed.

## F.5 Semantic Hamiltonians

Chapter 20's energy functional was borrowed from an existing Lagrangian rather than derived from first principles specific to a given semantic domain. Whether a semantic Hamiltonian can instead be derived, for a specific application, from more basic assumptions about what that domain's admissible trajectories must conserve, rather than posited by analogy to physics, is open, and a genuine answer would strengthen exactly the moderate claim Chapter 39 was careful to limit itself to.

## F.6 Empirical Validation

Nothing in this book has been tested against real data or a real implementation; every result is architectural or, in Appendix B, conditional on hypotheses stated but not checked. The clearest candidate for a first empirical test is already on record elsewhere: prior work on hierarchical continuation in locomotion, which this book's Chapter 33 draws on directly, proposes three explicit tests, a compactness criterion, a ground-contact discontinuity test, and a balance-recovery test, and predicts that whole-body coherence should measurably decline before an observable loss of balance during a stumble. That prediction is testable against existing

perturbation datasets without requiring this book's full architecture to be built first, and it would be the most direct available check on whether the coherence-decline diagnostic this book reuses at civilizational scale, Chapter 36.5, has any empirical support at all before being trusted at a scale where controlled experiment is not available.

## F.7 Sheaf-Learning

Chapter 21's cover $\mathcal{G}$, the decomposition of a domain into overlapping patches, was assumed given rather than derived. Whether a good cover, one where local admissibility checking is genuinely informative and gluing failures genuinely correspond to meaningful inconsistency, can instead be learned from data is open, and connects to representation learning more broadly: a learned cover is close to a learned notion of which regions of a domain are the natural units of local coherence.

## F.8 Category-Theoretic Semantics

Chapter 23 built $\mathcal{W}$, the category of worlds, with enough structure to state isomorphism precisely but without connecting it formally to other categorical frameworks, most notably the probabilistic and Markov categories that would naturally relate to Chapter 24's information-geometric treatment. A fully worked-out categorical semantics, relating $\mathcal{W}$ to these existing frameworks through explicit functors, re-

mains to be built.

## F.9 Differentiable Projection Operators

This is likely the most tractable and most valuable open problem for near-term work. $\Pi_C$, as this book has specified it, is a hard constraint-satisfaction operation, a nearest point in a closed set. A system whose proposal generator, $F_\psi$, is trained end-to-end by gradient descent needs gradients to flow through every stage of the write cycle, including projection, and a nearest-point operation onto an arbitrary closed set is not, in general, differentiable. Making $\Pi_C$ differentiable, whether through a smoothed relaxation, an implicit-function-theorem argument of the kind used in differentiable optimization layers elsewhere in machine learning, or some other technique, is a well-posed, specific problem this book's architecture cannot be trained the way Chapter 13 implies without solving.

## F.10 Distributed Semantic Fields

Chapter 30 sharded the world database and left fault tolerance unaddressed; a fuller open problem is whether a semantic field can be given genuine eventual-consistency guarantees, in the sense conflict-free replicated data types provide for ordinary distributed storage, while still respecting Chapter 29's admissibility-based conflict resolution rather than the simpler last-write-wins or vector-clock

reconciliation such data types typically use. Reconciling these two disciplines, formally, remains open.

## F.11 Civilization-Scale Field Dynamics

Chapters 36 and 41 developed a civilization-scale reading of this architecture entirely from theory, engineering in one case and ontological reinterpretation in the other, without checking either against real historical, economic, or institutional data. Whether real institutions genuinely behave like dynamic fixed points in any measurable sense, and whether the coherence-decline-before-collapse diagnostic has any support in documented institutional or civilizational collapse, is a long-horizon empirical question this book has only posed, not begun to answer.

## F.12 A Closing Note

A good open-problems appendix earns its place by outliving the book it closes. The problems above are offered in that spirit: not as an admission that this book's argument is incomplete, since Chapters 1 through 42 stand on their own regardless of whether any of these problems are ever solved, but as an invitation, stated precisely enough that later work, this author's or anyone else's, can take up a specific numbered problem here rather than starting from the wall in Chapter 1 all over again.