Admissible Degradation

A Mathematical Theory of Priority, Refusal, Recovery, and Continuation Under Constraint

Flyxion
Independent Researcher

Contents

The Mathematical Status of the Theory

The theory developed here originates in recognizable engineering problems: overload, scheduling, restart, rollback, load shedding, and the preservation of critical work under resource contraction. Its definitions are nevertheless not restricted to computers, nor are its mathematical objects merely abbreviated descriptions of the examples from which they were derived. Once a task system, dependency relation, obligation predicate, resource map, and family of degradation operators have been specified, their consequences can be studied independently of any particular implementation.

In this respect the distinction between pure and applied mathematics is better treated as a distinction of orientation than as a partition of subject matter [8]. The same structure may be approached from either direction. One may begin with a concrete system and ask which mathematical object represents it, or begin with an abstract object and ask what follows from its definition before deciding where, if anywhere, it is instantiated. This monograph moves in both directions. Apollo, distributed services, memory architectures, and institutional triage motivate the definitions, but they do not determine the theorems. Conversely, results proved for the abstract objects constrain what may honestly be claimed about those applications.

The simplest of these objects is the order-ideal lattice generated by a finite dependency poset.

Proposition (The survivor lattice)**.** *Let $(P, \sqsubseteq_{\mathrm{dep}})$ be a finite partially ordered set, and let $J(P)$ denote the family of its order ideals. Then*

$$\big(J(P), \cap, \cup\big)$$

is a finite distributive lattice [7].

Proof. The intersection and union of any two order ideals are again order ideals. Under inclusion, intersection is their greatest lower bound and union their least upper bound. Distributivity follows from the distributivity of union and intersection over sets. □

This proposition is elementary, but its role is important. Before resources and obligations are introduced, the possible survivor sets possess a regular algebraic structure. Resource feasibility and obligation preservation do not create that structure; they cut a generally irregular family out of it:

$$\mathfrak{A}(b, \mathcal{O}) = \{A \in J(P) : C(A) \preceq b,\ I_{\mathcal{O}}(A) = 1\}\,.$$

The admissible family need not itself be a sublattice. Two admissible survivor sets can have a union that exceeds the resource budget, while their intersection can destroy the last support of a mandatory obligation. Admissibility therefore appears mathematically as a constraint-induced deformation of an otherwise distributive survivor space.

The exact condition under which it is a sublattice is worth stating precisely rather than left as a vague sufficient condition, since the two set operations behave differently.

Proposition (When the admissible family is a sublattice)**.** *Write* $\mathfrak{I}_{\mathcal{O}} = \{A \in J(P) : I_{\mathcal{O}}(A) = 1\}$ *and* $\mathfrak{F}_b = \{A \in J(P) : C(A) \preceq b\}$*, so that* $\mathfrak{A}(b, \mathcal{O}) = \mathfrak{I}_{\mathcal{O}} \cap \mathfrak{F}_b$*. The meet is automatic:* $C(A \cap B) \preceq C(A) \preceq b$ *for nonnegative costs, so* $\mathfrak{F}_b$ *is always closed under intersection on* $\mathfrak{A}(b, \mathcal{O})$*. The family* $\mathfrak{A}(b, \mathcal{O})$ *is a sublattice of* $J(P)$ *exactly when, for all* $A, B \in \mathfrak{A}(b, \mathcal{O})$*,* $I_{\mathcal{O}}(A \cap B) = 1$ *and* $C(A \cup B) \preceq b$*; the first is not automatic, since a shared task can be the last common support of an obligation neither* $A \setminus B$ *nor* $B \setminus A$ *alone still provides, and the second is not automatic either, since costs only grow under union.*

A convenient sufficient condition follows directly: if $\mathfrak{I}_{\mathcal{O}}$ is itself a sublattice of $J(P)$, so obligation preservation is closed under both intersection and union, and if

$$C\Big(\bigcup_{A \in \mathfrak{A}(b,\mathcal{O})} A\Big) \preceq b,$$

the union of every admissible survivor set together still fits the budget, then every pairwise join $A \cup B$ for $A, B \in \mathfrak{A}(b, \mathcal{O})$ is a subset of that larger union and therefore resource-feasible by monotonicity of C under nonnegative costs, and $\mathfrak{A}(b, \mathcal{O})$ is a sublattice. Sublattice structure by itself says nothing about whether a pruning policy can find any particular element of it; that is a question about accessibility and exchange, addressed on its own terms in Part VIII, not a consequence of algebraic regularity alone.

The resource levels supporting at least one admissible survivor also form an abstract object with a simple but consequential order property.

Proposition (Monotonicity of the capacity region)**.** *Suppose task costs are fixed and nonnegative. If*

$$b \in \mathcal{B}_{\mathcal{O}} \quad \textit{and} \quad b \preceq b',$$

then

$$b' \in \mathcal{B}_{\mathcal{O}}.$$

Thus $\mathcal{B}_{\mathcal{O}}$ *is an upper set in* $\mathbb{R}^m_{\geq 0}$ *under the coordinatewise order.*

Proof. Since $b \in \mathcal{B}_{\mathcal{O}}$, some $A \in \mathfrak{A}(b, \mathcal{O})$ satisfies

$$C(A) \preceq b.$$

Because $b \preceq b'$, transitivity gives

$$C(A) \preceq b',$$

while the dependency and obligation conditions on A are unchanged. Hence $A \in \mathfrak{A}(b', \mathcal{O})$. □

The minimal elements of $\mathcal{B}_{\mathcal{O}}$ form a possibly non-singleton resource frontier. In a multidimensional resource space there need not be one least capacity sufficient for the system. One admissible realization may require more memory and fewer execution cycles, while another requires less memory and more communication bandwidth. These minimal vectors can be mutually incomparable:

$$b^{(1)} \not\preceq b^{(2)} \qquad \text{and} \qquad b^{(2)} \not\preceq b^{(1)}.$$

The boundary of viable operation is therefore generally a Pareto frontier rather than a scalar threshold.

Historical sufficiency supplies a third mathematical construction. For a fixed obligation set $\mathcal{O}$, equality of continuation sets defines an equivalence relation on histories:

$$h_1 \sim_{\mathcal{O}} h_2 \iff \mathrm{Cont}_{\mathcal{O}}(h_1) = \mathrm{Cont}_{\mathcal{O}}(h_2).$$

The resulting quotient

$$\mathcal{H}/\sim_{\mathcal{O}}$$

does not classify histories according to what happened in them. It classifies them according to which obligation-relevant futures they still permit. Histories that are microscopically different may belong to the same class, while histories containing the same retained payload may belong to different classes because their provenance, binding state, or discharged obligations differ.

This quotient-space view clarifies why historical sufficiency is not ordinary compression. Compression attempts to represent a past object economically. Historical sufficiency identifies exactly those differences in the past that no admissible future can observe. It is therefore relative to $\mathcal{O}$: when the obligation set changes, the equivalence classes may split or merge.

Finally, the safe-pruning relation

$$A \rightsquigarrow A'$$

turns the obligation-preserving survivor family into a directed graph. Questions that initially appear to concern scheduler design then become questions about reachability, terminal vertices, accessibility, and path dependence. The existence of an admissible vertex does not imply that every policy reaches it. The existence of a path does not imply that a local rule can discover one. These distinctions belong to the abstract graph before they belong to any software implementation.

The monograph is consequently applied in provenance but not confined to application in content. Its engineering cases determine which questions are worth asking; its mathematical objects determine which answers follow. No claim of purity is needed to recognize that the theory has crossed an important threshold once its definitions support results whose truth no longer depends on the historical example that first suggested them.

Part I. Systems Under Contraction

A system rarely fails because every component becomes unavailable at once. More often, demand exceeds capacity, observations conflict, deadlines contract, or one subsystem begins consuming resources needed elsewhere. The system must then stop doing some things in order to remain capable of doing anything important. This part develops the vocabulary needed to ask that question precisely: what is a system, what does it owe, and what does it mean for a state to still admit a way of meeting those debts.

The central object is not reliability in the ordinary probabilistic sense. Reliability asks whether a system keeps operating. Admissible degradation asks whether it keeps the correct obligations while giving up the correct alternatives. A system may remain active yet become operationally false to its purpose; conversely, it may refuse a large share of its workload while preserving exactly the invariant that defines what it is for.

Chapter 1

Capacity Is Not Capability

A system that has resources to spend is not thereby a system that can do what it needs to do. Capacity is a quantity; capability is a relation between that quantity, a workload, and a purpose. The distinction only becomes visible once all three are given separate names.

Definition 1.1 (System)**.** *A system is a tuple*

$$\mathcal{S} = (X, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{W}),$$

where X is the state space, $\mathcal{T}$ is the set of candidate tasks or transitions available to the system, $\mathcal{R}$ is the space of resources the system can draw on, $\mathcal{O}$ is the system's set of obligations, and $\mathcal{W}$ is its witness structure: the record it is capable of producing about what it did and why.

The witness structure $\mathcal{W}$ is included at the outset rather than added later as an afterthought. A system that degrades correctly but leaves no trace of what it refused, suspended, or preserved is operationally indistinguishable, from the outside, from a system that degraded by accident. Accountability is not a report generated after the fact; it is a component of the system's own definition.

At time t, the active workload is a subset of candidate tasks,

$$P_t \subseteq \mathcal{T},$$

and the available resource vector is

$$b_t = \left(b_t^{(1)}, \ldots, b_t^{(m)}\right) \in \mathbb{R}^m_{\geq 0}.$$

Resources are treated as multidimensional from the start. Execution cycles, memory, bandwidth, attention, and authority to act do not trade against one another at a fixed rate, and collapsing them into a single scalar budget would erase exactly the structure this theory needs.

Each task $p \in P_t$ carries a demand

$$c_t(p) \in \mathbb{R}^m_{\geq 0},$$

and ordinary, unconstrained operation is possible exactly when

$$\sum_{p \in P_t} c_t(p) \preceq b_t,$$

where $\preceq$ denotes the coordinatewise partial order on $\mathbb{R}^m$. Contraction begins the moment this inequality fails: either the workload has grown, or the available resources have shrunk, or both. The word "overload" names this crossing, not any particular magnitude of shortfall.

Remark 1.2. *The inequality failing is not itself a diagnosis. It says only that not every candidate task can run to completion under the current resource state. It says nothing yet about which tasks matter, which can be delayed without cost, and which cannot be interrupted without doing damage that no later resource surplus can undo. Supplying that missing structure is the work of the rest of Part I.*

The mathematical problem this monograph poses is therefore not the familiar one of minimizing workload subject to a budget. It is the problem of selecting a subset of the workload whose execution preserves what the system was for, and of doing so in a way the system itself can later show. Ordinary scheduling theory addresses part of this problem well: given a workload and a resource budget, it has a great deal to say about resource-feasible allocation, and the sequential, resource-constrained version of that problem is the classical subject of dynamic programming and optimal control [5]. What neither generally establishes is that a resource-feasible allocation preserves a constitutive purpose. This monograph is concerned with adding exactly that, together with historical recoverability and witness adequacy, to whatever a resource-feasible allocation already provides.

Chapter 2

Obligations and Constitutive Invariants

Contraction forces a choice among tasks. That choice cannot be made well without first saying what the system is not permitted to give up, and this requires separating two notions that are routinely collapsed: a goal the system is pursuing, and a condition the system must not violate in order to remain the thing it claims to be.

Definition 2.1 (Obligation)**.** *An obligation is a condition that must remain reachable, protected, or satisfied across a bounded future interval. The obligation set at time t is*

$$\mathcal{O}_t = \{o_1, \ldots, o_n\},$$

where each obligation is a tuple

$$o_i = (\kappa_i, \delta_i, \rho_i, D_i).$$

Here κ_i measures the obligation's criticality, δ_i is the latest time at which it can still be satisfied, ρ_i measures how reversible its postponement or violation would be, and D_i is the set of tasks and states on which its satisfaction depends.

An obligation, on this definition, is not simply a preference with a high weight attached. It is a claim about the future: that some condition must remain within reach, whether or not the system is currently spending resources to reach it. A spacecraft's obligation to maintain a viable descent trajectory does not disappear the moment the guidance computer is busy with something else; it remains a live constraint on every other allocation of resources the system makes in the meantime.

Definition 2.2 (Constitutive invariant)**.** *A constitutive invariant I is an obligation, or conjunction of obligations, whose violation means the system no longer performs its declared function, independent of whether it continues to operate. Write $I(x) = 1$ when state x satisfies the invariant and $I(x) = 0$ otherwise.*

The distinction matters because a system can fail in either of two directions that look similar from the outside but are not the same event. It can stop operating, which is failure in the ordinary sense. Or it can keep operating, keep producing outputs, keep consuming resources, while no longer satisfying the invariant that made those outputs meaningful. A guidance computer that continues to compute but no longer maintains a viable descent trajectory has not merely degraded; it has stopped being a guidance computer, whatever cycles it is still spending. Continued execution is not itself the invariant. What the execution is for is the invariant, and the two can come apart.

Remark 2.3. *Not every obligation in $\mathcal{O}_t$ need be constitutive. A system typically carries some obligations it is expected to meet under normal operation but may permissibly abandon under sufficient contraction, and others it may never abandon without ceasing to be what it is. The criticality coordinate κ_i is a first approximation to this distinction; later chapters replace it with a structural one, since a scalar criticality score cannot by itself say which obligations are constitutive and which are merely important.*

Chapter 3

The Feasible Continuation Set

Given a state, a resource level, and an obligation set, the question that matters under contraction is not what the system would ideally like to do but whether any way of continuing exists that both respects the resource limit and keeps every non-negotiable obligation alive. This is a question about a set of trajectories, not about any single one, and it is worth naming as such before asking how a system should choose among its members.

Definition 3.1 (Feasible continuation set)**.** *For state x_t, resource level b_t, and obligation set $\mathcal{O}_t$, let $\Gamma(x_t, b_t, \mathcal{O}_t)$ denote the set of trajectories that satisfy the system's governing dynamics, remain within the resource budget at every future time along the trajectory, and preserve every obligation in $\mathcal{O}_t$ that is constitutive.*

Definition 3.2 (Continuation viability)**.** *A state x_t is continuation-viable under $(b_t, \mathcal{O}_t)$ when*

$$\Gamma(x_t, b_t, \mathcal{O}_t) \neq \varnothing.$$

This definition supports a viability-kernel formulation of the same idea, phrased over states rather than trajectories, in the sense developed by viability theory and set-invariance results in constrained control [1, 3, 4]:

$$K_{\mathcal{O}} = \{\, x \in X : \exists\, \gamma \in \Gamma(x, b, \mathcal{O}) \,\}.$$

$K_{\mathcal{O}}$ is the region of the state space from which some obligation-preserving future remains available at all. Overload becomes geometrically significant precisely when the system's current trajectory threatens to carry it outside $K_{\mathcal{O}}$: not when resources first fall short of the full workload, but when they fall short of every workload that would keep the constitutive invariant satisfiable going forward.

Remark 3.3. *Continuation viability is a property of the pair $(x_t, b_t, \mathcal{O}_t)$, not of x_t alone. The same state can be viable under a generous resource level and non-viable under a stringent one; the same resource level can leave one obligation set satisfiable and another not. Nothing about this definition presumes that viability, once lost, was ever recoverable by any choice available to the system. Some contractions genuinely leave $\Gamma(x_t, b_t, \mathcal{O}_t)$ empty. Part IV returns to this possibility directly, since a theory of admissible degradation that cannot say when degradation is not admissible would not be saying very much.*

Refusal, suspension, and reprioritization, on this view, are not merely efficient responses to scarcity. They are control operations whose purpose is to return the system's trajectory to $K_{\mathcal{O}}$,

or to keep it there, when the unconstrained workload would carry it out. The next part of the monograph asks how a system decides which tasks to give up in order to do this, and argues that this decision cannot be made correctly by any single numerical priority score, however carefully constructed. It must instead be built out of three separate relations: which tasks require which other tasks, which obligations the system may never trade away, and, only once those two have narrowed the field, which of the remaining options it prefers.

Part II. The Algebra of Selective Continuation

Part I established when a state admits some obligation-preserving future at all. It did not say how a system should choose among tasks when the unconstrained workload will not fit. The temptation at this point is to assign every task a single number and remove the lowest scores first. This part explains why that temptation should be refused, and what should replace it.

The refusal is not aesthetic. A scalar priority forces a comparison between tasks that protect different obligations, along dimensions that were never commensurable to begin with: how many cycles a task costs against how much information is lost if it is interrupted, against how close its deadline is, against how essential its obligation is to the system's declared purpose. Any single number claims to have already resolved all of those trade-offs. The claim is usually false, and the falsity is usually invisible, because the number still comes out and a decision still gets made. Classical real-time scheduling theory, from rate-monotonic and earliest-deadline-first analysis onward, has spent decades refining exactly this kind of scalar or near-scalar priority assignment [10, 11, 12]; the point of this part is not to set that literature aside but to be precise about what it does and does not license once obligation preservation, rather than deadline satisfaction alone, is the property being asked for. What this part builds instead is a decision procedure with three separate stages, each doing only the work it is actually equipped to do.

Chapter 4

Precedence, Not Priority

The first stage says nothing about value. It says only which tasks cannot be kept without also keeping certain others.

Definition 4.1 (Dependency preorder)**.** *A dependency preorder is a reflexive, transitive relation* $\sqsubseteq_{\text{dep}}$ *on* P_t*, where*

$$p_i \sqsubseteq_{\text{dep}} p_j$$

means that p_i *is required by* p_j*: no admissible way of retaining* p_j *exists in which* p_i *has been removed.*

The preorder is typically only partial. Two tasks can be incomparable because they serve different obligations and neither presupposes the other. This is a feature of the relation, not a gap to be closed. Forcing every pair of tasks into a comparison is exactly the move that produces artificial trade-offs where none need exist.

Definition 4.2 (Order ideal / survivor set)**.** *Given the dependency preorder, a set* $A \subseteq P_t$ *is an order ideal when*

$$p_j \in A, \quad p_i \sqsubseteq_{\text{dep}} p_j \quad \Longrightarrow \quad p_i \in A.$$

For $p \in P_t$*, its dependent cone is*

$$\uparrow p = \{\, q \in P_t : p \sqsubseteq_{\text{dep}} q \,\},$$

the set of tasks that require p*, directly or transitively.*

A survivor set that is not an order ideal is incoherent: it would retain some task while discarding a task that task requires, which is not a resource-saving move but a contradiction, since the retained task cannot actually run. Restricting attention to order ideals, in the standard sense familiar from order theory [9], is therefore not a simplifying assumption. It is the minimum condition for a survivor set to describe something a system could coherently do.

Definition 4.3 (Admissible family)**.** *Let* $c_t : P_t \to \mathbb{R}^m_{\geq 0}$ *assign resource demand, let* b_t *be the available resource vector, and let* $I_{\mathcal{O}}(A) \in \{0, 1\}$ *indicate whether survivor set* A *preserves every constitutive obligation in* $\mathcal{O}_t$*. The admissible family at time* t *is*

$$\mathfrak{A}_t = \left\{ A \subseteq P_t \;:\; A \text{ is an order ideal}, \; \sum_{p \in A} c_t(p) \preceq b_t, \; I_{\mathcal{O}}(A) = 1 \right\}.$$

Three separate conditions have to hold simultaneously for A to belong to $\mathfrak{A}_t$: coherence under dependency, feasibility under the resource budget, and preservation of the invariant. None of the three is optional and none is reducible to the others. A set can be coherent and resource-feasible while still failing the invariant; it can preserve the invariant and respect the budget while failing to be an order ideal, in which case it is not actually realizable. The first question the theory asks is therefore not which survivor set is best but whether $\mathfrak{A}_t$ is nonempty at all. Only once that question is answered does preference enter.

Definition 4.4 (Selection)**.** *Let $\succeq_t^*$ be a (possibly incomplete) preference relation over $\mathfrak{A}_t$. A selected survivor set A_t is any element of*

$$\mathrm{Max}(\mathfrak{A}_t, \succeq_t^*),$$

the set of $\succeq_t^*$-maximal elements of $\mathfrak{A}_t$.

Nothing requires $\succeq_t^*$ to be complete. When several admissible survivor sets are pairwise incomparable, that incomparability is allowed to remain in the model rather than being resolved by fiat. If a single answer is operationally required, a declared collapse map

$$\nu_t : \mathfrak{A}_t \to \mathbb{R}$$

may be introduced, and the selected set taken to be

$$A_t \in \arg\max_{A \in \mathfrak{A}_t} \nu_t(A).$$

The scalar ν_t is deliberately introduced last, and its role is disclosed rather than hidden: it chooses among alternatives that have already passed every structural gate, and it has no power to admit a set that failed one of those gates, nor to override the dependency preorder that determined which sets were even coherent. Scalarization, when it happens, is a terminal and optional operation, not the mechanism that does the actual work of the theory.

Remark 4.5. *The overall sequence is*

precedence constraints $\longrightarrow$ admissible order ideals $\longrightarrow$ preference-maximal ideals $\longrightarrow$ optional scalar collapse.

This is the same architecture the theory uses elsewhere: binding constraints are established before valuation is permitted to act, and valuation is permitted to act only over alternatives that constraint has already certified as coherent. The set $\mathrm{Max}(\mathfrak{A}_t, \succeq_t^*)$ is preference-maximal with respect to whatever relation $\succeq_t^*$ happens to be; it is Pareto-maximal specifically only when $\succeq_t^*$ is taken to be the coordinatewise relation $\succeq_\sigma$ induced by the task-profile coordinates defined below, or an analogous set-level refinement of it. Pareto maximality is an important special case of preference-maximality, not a synonym for it, and the general definition above should not be read as already assuming that special case.

Each task's relevant properties are collected in a profile rather than a score.

Definition 4.6 (Task profile)**.** *For $p \in P_t$, the profile*

$$\sigma_t(p) = \big(\kappa_t(p),\, \rho_t(p),\, \eta_t(p),\, d_t(p),\, c_t(p)\big)$$

records, respectively, the criticality of the obligation p supports, the reversibility of interrupting p, the information lost if p is interrupted, p's deadline proximity, and p's resource demand.

The profile induces a family of coordinatewise comparisons, $p_i \succeq_\kappa p_j$, $p_i \succeq_\rho p_j$, and so on, one for each component, together with their common refinement

$$p_i \succeq_\sigma p_j \iff p_i \succeq_k p_j \text{ for every coordinate } k.$$

When one task dominates another on every coordinate, the comparison is available without inventing an exchange rate between them. When each is superior along a different coordinate, the two remain incomparable until either a context-specific rule resolves them or an authorized valuation ν_t is invoked to do so explicitly. What the profile is not permitted to become is a single number arrived at silently, since that would reintroduce, under another name, exactly the premature scalarization this chapter exists to block.

Chapter 5

Sacrifice and Reachability

A nonempty admissible family answers the existence question but not the constructive one. A system under contraction does not begin at some arbitrary order ideal; it begins at the full workload P_t, which is typically *not* admissible, and must remove tasks one dependent cone at a time until it either reaches $\mathfrak{A}_t$ or exhausts its options. This is a precedence-constrained selection problem in the sense long studied in scheduling theory [13], though the objective here, obligation preservation under a partial precedence relation rather than minimization of a scalar cost, is not the one that literature usually poses. This chapter asks what can go wrong along the way, and under what condition nothing does.

Definition 5.1 (Safe pruning)**.** *A task $p \in A$ is safely prunable from A when*

$$A' = A \setminus \uparrow p$$

still preserves every constitutive obligation. Write $A \rightsquigarrow A'$ for this transition, and $A \rightsquigarrow^* A'$ for a finite sequence of such transitions.

Two structural facts hold for any such sequence, independent of how the pruned tasks are chosen.

Lemma 5.2 (Closure under cone removal)**.** *If A is an order ideal, then $A \setminus \uparrow p$ is an order ideal.*

Proof. Let $q \in A \setminus \uparrow p$ and $r \sqsubseteq_{\mathrm{dep}} q$. Since A is an order ideal, $r \in A$. If $r \in \uparrow p$, then $p \sqsubseteq_{\mathrm{dep}} r \sqsubseteq_{\mathrm{dep}} q$, so $q \in \uparrow p$, contradicting $q \in A \setminus \uparrow p$. Hence $r \in A \setminus \uparrow p$. □

Lemma 5.3 (Strict resource descent)**.** *Suppose $c(p) \neq 0$ for every $p \in P_t$. If $A \rightsquigarrow A'$, then $C(A') \prec C(A)$ in at least one coordinate, where $C(A) = \sum_{p \in A} c(p)$.*

Because P_t is finite, every pruning sequence terminates after at most $|P_t|$ transitions. Termination, however, only guarantees that the process stops; it does not guarantee that it stops inside $\mathfrak{A}_t$.

Example 5.4 (Branch abandonment)**.** *Let $\mathcal{O}_t$ contain two mandatory obligations. The first, o_1, admits two independent, alternative single-task supports: either a alone or b alone suffices. The second, o_2, requires d. Task c supports no obligation at all. Suppose*

$$c(a) = 6, \qquad c(b) = 3, \qquad c(c) = 3, \qquad c(d) = 4,$$

with budget $b = 7$. The full workload is $P_t = \{a, b, c, d\}$, with $C(P_t) = 16$. The set $A^* = \{b, d\}$, with $C(A^*) = 7$, is admissible: b covers o_1, d covers o_2.

Consider a scheduler that removes the cheapest currently dispensable task at each step, where a task is dispensable in A when $A\setminus\uparrow p$ still preserves every obligation. At P_t, both a and b are dispensable, since each obligation's alternative support is still present, and c is dispensable outright. The cheapest dispensable task is c (cost 3, tied with b); removing the genuinely inert task first looks unimpeachable, and it is: this step never leaves $\mathfrak{A}_t$ unreachable. It leaves $\{a, b, d\}$ at cost 13.

At the next step, a and b remain mutually dispensable, and b is the cheaper of the two, so the same greedy rule removes it, leaving $\{a, d\}$ at cost $10 > 7$. Neither remaining task can now be safely pruned: a has become the sole support for o_1, and d is the sole support for o_2. The process has terminated outside $\mathfrak{A}_t$, even though $A^* = \{b, d\}$ was reachable from P_t by the alternative order that removes c, then a.

The example generalizes: whenever an obligation has more than one support set, deleting part of one support can make the remaining, more expensive support indispensable. Local dispensability is therefore not a property of a task in isolation; it depends on the path already taken.

$$p \text{ dispensable in } A \quad \not\Longrightarrow \quad p \text{ dispensable in every } A' \subseteq A.$$

A locally sensible removal rule can consequently walk the system into a dead end from which the admissible family is no longer reachable, even though it was reachable from the starting point.

Proposition 5.5 (Local-priority insufficiency)**.** *There exists a finite obligation-support system with $\mathfrak{A}_t \neq \varnothing$ for which a scheduler that removes, at each step, the cheapest currently dispensable task terminates outside $\mathfrak{A}_t$.*

Proof. Example 5.4. □

This is the reason the theory refuses premature scalarization one level deeper than the objection raised in the previous chapter. It is not only that a scalar score inadequately expresses value trade-offs. A local scalar ranking, applied one removal at a time, can destroy access to a survivor set that was globally available, simply by resolving an early tie in the wrong direction.

The unconditional existence statement has to be phrased in terms of reachability, not in terms of the admissible family's nonemptiness alone.

Theorem 5.6 (Resource-feasible pruning reachability)**.** *A resource-feasible, invariant-preserving survivor set is reachable by safe cone pruning from P_t if and only if there exists $A^* \in \mathfrak{A}_t$ and a sequence $P_t = A_0 \rightsquigarrow A_1 \rightsquigarrow \cdots \rightsquigarrow A_k = A^*$.*

The statement looks close to definitional, and in one sense it is. Its content is to isolate a distinction the rest of the theory depends on: $\mathfrak{A}_t \neq \varnothing$ says that some good configuration exists; it says nothing about whether a safe transition sequence exists from where the system currently stands to that configuration. The set

$$\mathrm{Reach}_{\rightsquigarrow}(P_t) = \{\, A : P_t \rightsquigarrow^* A \,\}$$

can, in principle, fail to intersect $\mathfrak{A}_t$ even when $\mathfrak{A}_t$ itself is nonempty, if every safe pruning path from P_t is blocked before reaching an admissible set. Example 5.4 does not exhibit this: the intersection $\mathrm{Reach}_{\rightsquigarrow}(P_t) \cap \mathfrak{A}_t$ is nonempty there, since the path removing c and then a reaches $A^* = \{b, d\}$ directly. What the example exhibits is a different and equally important failure: a specific greedy policy, cheapest-dispensable-task-first, chooses another path through the same pruning graph and terminates outside $\mathfrak{A}_t$, despite the admissible set being reachable by a path the policy simply did

not take. Reachability of an admissible set and guaranteed discovery of one by a fixed policy are distinct properties, and the example above is about the second, not the first.

A positive, constructive result is available, but only under an explicit structural hypothesis that says the survivor family does not set the kind of trap Example 5.4 illustrates.

Definition 5.7 (Contraction accessibility)**.** *The survivor family is contraction-accessible from P_t when, for every A^* belonging to it and every A with $A^* \subsetneq A \subseteq P_t$ also belonging to it, there exists $p \in A \setminus A^*$ such that*

$$A^* \subseteq A \setminus \uparrow p$$

and $A \setminus \uparrow p$ still preserves every constitutive obligation.

In words: whenever the system's current survivor set properly contains some target that is already known to be good, at least one removable cone lies entirely outside that target and can be pruned without damaging what the target needs.

Theorem 5.8 (Accessible pruning)**.** *Let P_t be finite with nonnegative resource demands, and suppose the survivor family is contraction-accessible from P_t. If $\mathfrak{A}_t \neq \varnothing$, then a finite safe-pruning sequence exists from P_t to some member of $\mathfrak{A}_t$.*

Proof. Fix $A^* \in \mathfrak{A}_t$. If $P_t = A^*$, the sequence is empty. Otherwise contraction accessibility supplies $p_0 \in P_t \setminus A^*$ with $A^* \subseteq A_1 = P_t \setminus \uparrow p_0$, and A_1 still preserves every constitutive obligation. If $A_1 \neq A^*$, apply accessibility again to obtain A_2, and so on. At each step $A^* \subseteq A_{k+1} \subseteq A_k$, so the finite quantity $\mu_k = |A_k \setminus A^*|$ strictly decreases, since accessibility removes a cone entirely outside A^* at every step. The process therefore reaches A^* after finitely many transitions, and since $C(A^*) \preceq b_t$, the terminal set is resource-feasible. □

The proof leans on a target A^* that is already known to be admissible; it establishes that a path exists, not that an uninformed scheduler operating without foresight will find it. That gap between existence and discovery is real and is not closed by finiteness alone. Three distinct problems sit where the earlier prospectus had assumed one: whether $\mathrm{Reach}_{\rightsquigarrow}(P_t) \cap \mathfrak{A}_t$ is nonempty at all; whether some search procedure over the pruning graph can find a member of it; and whether a purely local, greedy rule can be trusted to do so without search. Contraction accessibility answers the first question affirmatively under a stated hypothesis. It does not, by itself, answer the third, and Example 5.4 shows that the third question has a negative answer in general. A sufficient condition under which greedy pruning *is* safe, tied to obligation-support systems in which an obligation's alternative support branches are pairwise disjoint, so that removing part of one branch can never be mistaken for removing all of it, is postponed to a later chapter, since it is a genuine open design question rather than a routine consequence of what has been established here.

Chapter 6

Refusal as a First-Class Transition

A task that is pruned from the survivor set has not simply vanished from consideration. What happens to it, and whether the system can later say what happened to it, is itself part of what this theory is trying to formalize.

Definition 6.1 (Refusal transition)**.** *For a task p excluded from the survivor set at state x_t, the refusal transition is*

$$\mathrm{Refuse}(p, x_t) = (x_{t+1}, w_p),$$

where w_p records which obligation justified the exclusion, whether p remains eligible for later reconsideration, and what would need to change for it to be readmitted.

Four dispositions need to be kept distinct, because collapsing them erases exactly the information a later recovery step would need.

- Refuse(p): p is denied admission to the current survivor set. It was never started, or its progress is treated as void; no restart obligation is created.
- Suspend(p): p is halted with a preserved continuation token, at a state from which it can later resume without repeating discharged work.
- Abort(p): p has been admitted and partially executed, and is now terminated in a way that does not preserve a resumable continuation; any recovery must restart from an earlier boundary.
- Forget(p): the system loses the ability to distinguish p from work that was never proposed, work that was rejected, and work that was completed. This is not a task disposition at all; it is the failure of the witness structure to record one.

The first three are legitimate outcomes of a contraction decision, each with different implications for what recovery, should resources later permit it, will cost. The fourth is not an outcome the theory endorses; it names the failure mode in which $\mathcal{W}$ has not done its job, and it is included here precisely so that later chapters can say when a system has fallen into it.

Remark 6.2. *This chapter connects directly to a distinct existing formalization of exactly these four transitions as primitive operations over an append-only event structure. That correspondence is developed on its own terms in a later part of the monograph and is not assumed here; what matters for the present argument is only that refusal, suspension, and abortion are mathematically distinguishable transitions with distinguishable recovery costs, a distinction the next part makes precise.*

Part III. Recovery Geometry

Part II described how a system chooses what to give up. It said nothing about what giving something up actually costs, or what has to be true for a suspended task to come back. Those two questions are different from each other. A task's priority is a fact about what the system currently owes; a task's recovery geometry is a fact about what would have to be reconstructed, and at what price, if the system later wanted that task back. Ignoring this distinction produces theories in which everything looks equally disposable the instant it is not running, which is false of essentially every real system with any internal state at all.

Chapter 7

Restartability

A suspended task is not a point; it is a trajectory that has been interrupted somewhere along its length, and where along that length it was interrupted matters.

Definition 7.1 (Task trajectory and restart set)**.** *Represent task p by an execution trajectory $\xi_p : [0, T_p] \to X_p$. A restart boundary is a point $r \in X_p$ from which some admissible continuation to completion of p remains available. The restart set is*

$$R_p = \{\, r \in X_p : \Gamma_p(r) \neq \varnothing \,\},$$

where $\Gamma_p(r)$ is the feasible continuation set for p alone, starting from r.

If interruption occurs at a state $x \notin R_p$, resuming p directly is not an option; the system must instead roll back to some earlier point that does belong to R_p.

Definition 7.2 (Rollback cost)**.** *The rollback cost of an interruption at x is*

$$\mathcal{C}_{\mathrm{rb}}(x) = \inf_{r \in R_p} \big[\, d(x, r) + \mathcal{L}(r, x) \,\big],$$

where $d(x, r)$ measures the cost of reconstructing the trajectory back to r, and $\mathcal{L}(r, x)$ measures whatever was produced between r and x that cannot be recovered by that reconstruction.

This single definition covers a family of mechanisms that are usually described separately and treated as unrelated engineering choices: a database checkpoint, a replay log, a protected memory region, and an idempotent operation are all, on this account, different ways of enlarging R_p or of shrinking the cost $d(x, r)$ once interruption occurs at some particular x, and each has its own substantial literature developed independently of the others [14, 15, 16, 17, 18]. A system that never checkpoints has $R_p = \{0\}$; every interruption forces a rollback all the way to the start, and $\mathcal{C}_{\mathrm{rb}}$ grows with however much of the trajectory had already been traversed. A system with frequent, cheap checkpoints keeps R_p dense along ξ_p, and $\mathcal{C}_{\mathrm{rb}}(x)$ stays small no matter where interruption lands.

Remark 7.3. *Restartability is a property of the task and the system's own instrumentation of it, not a property of how important the task is. A low-priority task with no checkpointing can be more expensive to interrupt than a high-priority task that has been carefully checkpointed. Chapter 5's pruning apparatus operates on dependency cones, which say what must be removed together; nothing in that apparatus says what removal costs once it happens. Restart geometry supplies the missing cost term, and the next chapter shows why priority still cannot simply be replaced by this cost term either.*

Chapter 8

Reversibility and Commitment Depth

Some transitions can be undone almost for free. Others cannot be undone at all, regardless of how many resources are later made available. A theory of sacrifice that ignores this difference will systematically undervalue tasks that look cheap and unimportant at the moment of contraction but have already committed the system to something it cannot walk back from.

Definition 8.1 (Transition reversibility)**.** *Each transition a carries a reversibility value $\rho(a) \in [0, 1]$. A value near 1 means the prior state can be reconstructed cheaply and faithfully after a is undone; a value near 0 means a is an effectively irreversible commitment.*

Definition 8.2 (Commitment depth)**.** *For a path $\gamma = (a_1, \ldots, a_k)$, the cumulative commitment depth is*

$$D(\gamma) = \sum_{i=1}^{k} -\log\big(\rho(a_i) + \varepsilon\big),$$

for a small $\varepsilon > 0$ that keeps the expression finite when some $\rho(a_i) = 0$.

A path built from many highly reversible transitions has low commitment depth even if it is long; a single near-irreversible transition can dominate the sum on its own. This is deliberately not folded into the criticality coordinate κ from Chapter 2. A task can support a low-criticality obligation while its own execution has already produced a high-commitment-depth state elsewhere in the system, for instance by having consumed a resource, published a result, or triggered an external effect that cannot be recalled. Treating such a task as freely disposable because its *obligation* looks minor ignores the fact that its *execution* may already be irreversible.

Remark 8.3. *The task profile $\sigma_t(p)$ from Chapter 4 already carries a reversibility coordinate $\rho_t(p)$; this chapter supplies its intended reading. Preservation pressure on a task should be understood as a function of criticality, deadline, reversibility, commitment depth, and information loss taken together,*

$$\textit{preservation pressure} = f(\kappa, \delta, \rho, D, \eta),$$

and no coordinate of that tuple is permitted to stand in for the others. In particular, low κ does not license ignoring high D.

This distinction generalizes the admission thresholds already used elsewhere in the theory. Whether a task is allowed into memory, whether it is ready for computation, and whether it is permitted to act in a way that produces effects outside the system are three separate gates, and they can be crossed at different points along the same trajectory. A task can sit in memory

indefinitely at negligible commitment depth; the moment it is scheduled for computation, some resources become committed even if the result is discarded; the moment it acts, commitment depth can jump discontinuously, since actions on the external world are frequently the least reversible transitions a system has available to it. A sacrifice ordering that only inspects a task's current priority, without asking which of these thresholds it has already crossed, will misjudge exactly the tasks that matter most to misjudge correctly.

Chapter 9

Historical Sufficiency

Suppose a suspended task is later reconsidered and resources become available to resume it. What, precisely, must the system have retained in order for that resumption to be meaningful? The naive answer, retain everything, is both practically impossible and conceptually too strong. The correct answer is retain whatever the task's future obligations could still depend on, and nothing more is required.

Example 9.1 (Retention without recoverability)**.** *Suppose a task computes*

$$y = f_4 \circ f_3 \circ f_2 \circ f_1(x),$$

and execution is interrupted immediately after f_3. Retaining the intermediate value

$$z_3 = f_3(f_2(f_1(x)))$$

looks sufficient, since $y = f_4(z_3)$ can apparently be computed from z_3 alone. But suppose f_4 also depends on a hidden mode m that may have changed since interruption, on the version v of some external resource it reads, and on a token q recording which upstream obligations have already been discharged. Then

$$H_t = \{z_3\} \quad \not\Rightarrow \quad \Gamma_{f_4}(H_t) \neq \varnothing :$$

the stored value preserves data, but the continuation it needs to support may no longer be feasible from that data alone. A sufficient history is instead something like

$$H_t^+ = (z_3, m, v, q, \ell),$$

where ℓ additionally identifies the restart boundary itself and which obligations were already discharged up to that point.

The example isolates the failure mode this chapter is about: it is not that too little was stored, in the sense of raw volume. z_3 may be a large object and (m, v, q, ℓ) small by comparison. The failure is that the retained object did not include the specific coordinates the downstream continuation actually reads. Volume and sufficiency are different axes, and a system can fail on the second while succeeding lavishly on the first.

Definition 9.2 (Continuation-equivalent histories)**.** *Let $\mathrm{Cont}_{\mathcal{O}}(h)$ denote the set of obligation-relevant continuations compatible with a retained history h. Two histories are continuation-equivalent, written $h_1 \sim_{\mathcal{O}} h_2$, when*

$$\mathrm{Cont}_{\mathcal{O}}(h_1) = \mathrm{Cont}_{\mathcal{O}}(h_2).$$

Definition 9.3 (Historical sufficiency)**.** *A retained representation $\widehat{h}$ is sufficient for an original history h, relative to $\mathcal{O}$, when $\widehat{h} \sim_{\mathcal{O}} h$.*

This is a deliberately weaker requirement than exact replay. It does not ask that $\widehat{h}$ reconstruct h itself, only that the two support exactly the same set of future obligation-relevant continuations. Information that no admissible continuation could ever depend on may be discarded without any loss of sufficiency, however much of it there was. In Example 9.1, this is what licenses forgetting f_1 and f_2's intermediate outputs entirely, retaining only H_t^+: nothing downstream of f_3 reads them, so their absence changes nothing about $\mathrm{Cont}_{\mathcal{O}}$.

Remark 9.4. *Historical sufficiency gives Chapter 7's restart set a second, obligation-relative description. A restart boundary $r \in R_p$ is a state from which the task's own continuation remains feasible; a sufficient history is the smallest retained object from which that same continuation remains feasible without requiring the system to sit physically at r. The two notions coincide when the only thing a continuation depends on is the state r itself, and come apart exactly when hidden dependencies like m, v, and q above are present, which is the ordinary case for any system embedded in an environment it does not fully control.*

Two contrasting instances, developed at length in Part V, are worth naming here in outline. A system whose restart protection is engineered around a declared obligation set, so that what it preserves across interruption is exactly what its downstream continuations will need, can suspend and resume work through significant overload without ever retaining more than H_t^+-sized objects at each boundary. A system that instead accumulates raw payload indefinitely, without preserving the provenance, ordering, or discharge status that its own obligations depend on, can hold an arbitrarily large archive while still failing historical sufficiency outright: byte retention and continuation recoverability are not the same property, and a system can maximize the first while losing the second entirely.

Part IV. Overload as a Phase Transition

The chapters so far have treated resource level b_t largely as a fixed background quantity against which a single contraction episode is analyzed. Real systems experience b_t as something that moves, sometimes gradually and sometimes in a step, and the admissible family moves with it. This part asks what happens to $\mathfrak{A}(b, \mathcal{O})$ as b contracts, and what distinguishes a system that responds to that contraction well from one that does not.

Chapter 10

Capacity Boundaries

Chapter 4 fixed a resource level b_t and asked whether $\mathfrak{A}_t$ was nonempty. It is equally natural to fix the obligation set and ask, as a question about b itself, for which resource levels an admissible survivor set exists at all.

Definition 10.1 (Obligation-preserving capacity region)**.** *For a fixed obligation set $\mathcal{O}$, let*

$$\mathfrak{A}(b,\mathcal{O}) = \left\{ A \subseteq P \;:\; A \text{ an order ideal, } \sum_{p\in A} c(p) \preceq b,\; I_{\mathcal{O}}(A) = 1 \right\},$$

matching Chapter 4's definition with b now treated as a variable. The obligation-preserving capacity region is

$$\mathcal{B}_{\mathcal{O}} = \{\, b : \mathfrak{A}(b,\mathcal{O}) \neq \varnothing \,\}.$$

As b shrinks along some path, the system does not cross this boundary gradually in the sense of degree; it crosses it in the sense of kind. On one side, some order ideal exists that keeps every constitutive obligation satisfiable; on the other, none does, regardless of how the workload is pruned. This is the sense in which overload is better modeled as a phase transition than as a continuously worsening quantity: what changes discontinuously at the boundary is not how well the system is doing but whether it is doing the thing it is for at all, though the word "phase transition" should still be read as an analogy here rather than as an invocation of the technical statistical-mechanical apparatus of order parameters and limiting regimes, which this monograph does not develop.

Within $\mathcal{B}_{\mathcal{O}}$ itself, further structure is visible, and stating it precisely, rather than gesturing at it, requires naming what a given resource level actually buys beyond bare admissibility: how much redundancy of support each obligation retains.

Definition 10.2 (Alternative branch)**.** *For a mandatory obligation o, an alternative branch is a minimal set of tasks $S \subseteq P$ such that S alone satisfies o. The branch family of o is $\mathcal{S}_o = \{S_o^1, \ldots, S_o^{k_o}\}$; its cost is $C(S) = \sum_{p\in S} c(p)$.*

Definition 10.3 (Failure-domain relation)**.** *A failure-domain relation is a map $\varphi : P \to F$ assigning each task to the failure domain, machine, memory bank, authority, or network link, whose loss would take it down. Two alternative branches $S, S' \in \mathcal{S}_o$ are failure-independent under φ when $\varphi(S) \cap \varphi(S') = \varnothing$. Taking φ to be injective recovers ordinary task-disjointness as the special case where sharing a task is the only way to share a failure domain.*

Definition 10.4 (Redundancy)**.** *For a constitutive obligation o and survivor set A, let* $\mathrm{pack}_o(A)$ *be the maximum number of pairwise failure-independent branches of $\mathcal{S}_o$ contained in A. The redundancy of resource level b is*

$$r_{\mathcal{O}}(b) = \max_{A \in \mathfrak{A}(b,\mathcal{O})} \min_{o \in \mathcal{O}_{\mathrm{const}}} \mathrm{pack}_o(A),$$

defined whenever $\mathfrak{A}(b, \mathcal{O}) \neq \varnothing$.

Since every $A \in \mathfrak{A}(b, \mathcal{O})$ satisfies $I_{\mathcal{O}}(A) = 1$, every constitutive obligation has at least one branch present in A, so $r_{\mathcal{O}}(b) \geq 1$ throughout $\mathcal{B}_{\mathcal{O}}$; redundancy beyond 1 is what stratification tracks.

Definition 10.5 (Redundancy strata)**.**

$$B_i = \{\, b \in \mathcal{B}_{\mathcal{O}} : r_{\mathcal{O}}(b) \geq i \,\}, \qquad i \geq 1.$$

Proposition 10.6 (Nesting of the strata)**.** *$B_1 = \mathcal{B}_{\mathcal{O}}$, and $B_{i+1} \subseteq B_i$ for every $i \geq 1$.*

Proof. $B_1 = \mathcal{B}_{\mathcal{O}}$ follows from $r_{\mathcal{O}}(b) \geq 1$ holding throughout $\mathcal{B}_{\mathcal{O}}$, as noted above. $B_{i+1} \subseteq B_i$ is immediate, since $r_{\mathcal{O}}(b) \geq i + 1$ implies $r_{\mathcal{O}}(b) \geq i$. □

This gives the stratification

$$\mathcal{B}_{\mathcal{O}} = B_1 \supseteq B_2 \supseteq \cdots,$$

in which a crossing from B_i into a resource level outside B_i but still inside B_{i-1} means precisely that the system's weakest constitutive obligation has lost one layer of independently realizable support, not that any obligation has yet lost its last one. Crossing out of $\mathcal{B}_{\mathcal{O}} = B_1$ altogether is the qualitatively different, terminal event: no complete obligation-preserving survivor set remains at any redundancy level, including the minimal one.

Remark 10.7. *A system need not, and generally should not, wait until b_t has left $\mathcal{B}_{\mathcal{O}}$ before changing how it operates. Recognizing that b_t has crossed from stratum B_{i-1} into B_i is itself useful information, precisely because $r_{\mathcal{O}}(b_t)$ has decreased: it says some obligation's independently realizable supports have thinned, and a further, smaller contraction could remove the last of them. A system that only responds once $\mathfrak{A}_t = \varnothing$, that is, once b_t has left B_1 entirely, has already waited past the point where the response could still have been chosen deliberately, from among several remaining options, rather than forced.*

Chapter 11

Graceful and Ungraceful Degradation

Chapter 4 established which survivor sets are admissible. Chapter 6 established four distinct dispositions a removed task can be given. This chapter joins the two: what makes a transition from P to some smaller A a good one is not only that A is admissible, but that every task outside A has been given one of the first three dispositions from Chapter 6, and not the fourth.

Definition 11.1 (Graceful degradation)**.** *A degradation from workload P to survivor set $A \subset P$ is graceful relative to $\mathcal{O}$ when*

$$I_{\mathcal{O}}(A) = I_{\mathcal{O}}(P)$$

and the excluded tasks admit a full disposition partition,

$$P \setminus A = R_{\text{refused}} \sqcup R_{\text{suspended}} \sqcup R_{\text{aborted}},$$

where each task in $P \setminus A$ has received exactly one of the three dispositions defined in Chapter 6, with a witness recorded for each.

Definition 11.2 (Ungraceful degradation)**.** *A degradation is ungraceful when either the invariant is not preserved, or some excluded task's fate is not recorded under any of the three dispositions, so that it has effectively been* Forget*-ed in the sense of Chapter 6: not distinguishable, after the fact, from work that was never proposed at all.*

The distinction is not cosmetic. Two systems can arrive at the identical survivor set A, one having deliberately refused, suspended, or aborted every excluded task with a recorded reason, the other having simply failed to schedule them under incidental competition, undefined arrival-order tie-breaking, or corrupted state. Both preserve $I_{\mathcal{O}}$ at the current instant. Only the first has any basis for later reconstructing what happened, resuming what was suspended at its recorded restart boundary, or explaining, to an operator or to itself, why the particular set A was the one that survived. Graceful degradation is therefore not a stronger requirement on which tasks survive; it is a requirement on what can be said about the tasks that did not.

Remark 11.3. *This chapter deliberately stops short of stating a general theorem guaranteeing that graceful degradation is always achievable whenever $\mathfrak{A}(b, \mathcal{O})$ is nonempty. Chapter 5 already showed that reachability, not mere nonemptiness, governs whether some admissible survivor set can be found at all, and that an uninformed local rule can fail to find one even when it exists. A corresponding claim about* graceful *reachability, that some sequence of Chapter 6 dispositions can accompany a successful pruning path, inherits the same dependence on contraction accessibility and is addressed, together with its precise hypotheses and its limits, in Part VI.*

Part V. Worked Example: Apollo 11

The apparatus built so far is abstract by design, and abstraction earns its keep only by being tested against a case where the stakes, the constraints, and the outcome are all independently known. The Apollo 11 lunar descent supplies such a case. The commitment being made here, that a general theory should be built alongside, and repeatedly checked against, a single concrete worked example rather than developed in isolation and illustrated only afterward, has a well-known precedent in a different field: Needham's development of complex analysis from visual and diagrammatic reasoning follows much the same discipline, letting a concrete picture carry real argumentative weight rather than serving as decoration after the algebra is done [6]. What follows is not a reconstruction of the Apollo Guidance Computer's assembly-language internals, which this monograph does not attempt to reproduce, but a reading of the publicly documented history of one incident through the vocabulary built in Parts I through IV. The reading is offered to see what the vocabulary clarifies and, in the closing chapter, to say plainly what it does not establish.

Chapter 12

The Guidance Computer as a Constrained System

The Apollo Guidance Computer ran a real-time executive, designed under Margaret Hamilton's direction at the MIT Instrumentation Laboratory, that scheduled a handful of concurrent jobs, each carrying a fixed priority number assigned at design time [25, 26]. A companion routine, the Waitlist, handled short timed tasks separately. Bookkeeping for each running job consumed two distinct, scarce resources: a slot in the Core Set, and, for jobs needing working storage, a VAC area. In the vocabulary of Chapter 1, this already exhibits the multidimensionality the resource vector b_t was built to represent: a job can be blocked because no execution cycles remain, or because execution cycles remain but no bookkeeping slot does, and the second failure mode has nothing to do with raw computational throughput.

Instantiating the system tuple:

$$\mathcal{S}_{\mathrm{AGC}} = (X_{\mathrm{AGC}}, \mathcal{T}_{\mathrm{AGC}}, \mathcal{R}_{\mathrm{AGC}}, \mathcal{O}_{\mathrm{AGC}}, \mathcal{W}_{\mathrm{AGC}}),$$

X_{AGC} is the spacecraft's guidance state together with the executive's internal bookkeeping; $\mathcal{T}_{\mathrm{AGC}}$ is the set of jobs available to run at a given moment, including guidance computation, radar data processing, display servicing, and engine control; $\mathcal{R}_{\mathrm{AGC}}$ is the pair of execution cycles and bookkeeping slots just described; $\mathcal{O}_{\mathrm{AGC}}$ contains, as its constitutive invariant, the requirement that a viable powered-descent trajectory remain reachable; and $\mathcal{W}_{\mathrm{AGC}}$ is the alarm and display system, which the next three chapters treat as doing real work rather than as an incidental readout.

The executive's priority numbers deserve a precise place in Chapter 4's architecture. They were not discovered at runtime by any search over an order-ideal graph; they were fixed in advance by engineers who had already reasoned about which obligations mattered most. In the terms of Chapter 4, the priority table is best read as a declared collapse map ν_t, authorized before flight and applied only after the underlying dependency and admissibility structure had already been engineered to make that collapse safe. This is a meaningfully different claim than "the AGC used a scalar priority and it worked," and the distinction matters for what Chapter 16 will and will not conclude from this case.

Chapter 13

The 1201/1202 Event

During the powered descent on July 20, 1969, a checklist error left the rendezvous radar switch in a position that caused the radar hardware, which was not needed for descent, to continue supplying data to the computer; a separate hardware timing mismatch between the radar's own electronics and the computer's clock caused this unwanted input to generate spurious interrupts at a low but steady rate [31, 32]. In the language of Chapter 1, this is an exogenous perturbation to the workload,

$$P'_t = P_t \cup q_t,$$

where q_t is the phantom radar-servicing demand, arriving without warning and without having been budgeted for in the executive's design. The perturbation did not exhaust execution cycles outright. It exhausted the bookkeeping resource: the executive's accounting for the phantom job, repeated every cycle, eventually left no Core Set entry available for a new job (triggering the 1201 alarm, "Executive Overflow, No Core Sets") and, moments later in the same descent, no VAC area available (triggering the 1202 alarm, "Executive Overflow, No VAC Areas").

Both alarms report the same underlying event in Chapter 10's terms: the resource level b_t, considered along its bookkeeping coordinate rather than its cycle-time coordinate, crossed out of $\mathcal{B}_{\mathcal{O}}$ for the survivor family that included the phantom job. The crossing was not a single, one-time event. Because the hardware mismatch causing q_t was never removed during the descent, the computer crossed the same boundary repeatedly, several times within a few minutes, each crossing forcing another instance of the recovery response described in the next chapter.

Remark 13.1. *It is worth being precise about what overloaded and what did not. The guidance and navigation computations themselves, the tasks bound directly to the constitutive invariant, were never short of the cycles they needed. What ran out was the executive's own capacity to keep track of how many jobs it was tracking. A theory of admissible degradation that only modeled compute-cycle scarcity would have had nothing to say about this incident at all; it is a genuinely bookkeeping-resource overload, which is exactly why Chapter 1 insisted on treating b_t as multidimensional from the outset rather than collapsing it into a single scalar budget.*

Chapter 14

Priority Display, Restart Protection, and Recovery

The executive's response to each overflow was not a fine-grained, task-by-task application of Chapter 5's safe-pruning apparatus. It was coarser than that, and the coarseness is itself informative. On detecting that no bookkeeping resource remained, the system performed a restart: it discarded the transient state of every currently scheduled job uniformly, low-priority and high-priority alike, and rebuilt the job set from a fixed priority table, admitting jobs back in priority order as bookkeeping resources again became available. This is closer to pruning an entire order ideal down to $\varnothing$ and reconstructing than to Chapter 5's model of removing one dependent cone at a time.

What made this uniform, wholesale restart survivable, rather than catastrophic, was Chapter 7's and Chapter 9's machinery, not Chapter 5's. The spacecraft's guidance state, position, velocity, and the parameters of the intended descent trajectory, was held in a region of memory that the restart protocol did not touch. In the language of Chapter 9, this protected region functioned as a sufficient history $\widehat{h}_{\mathrm{AGC}}$ for the guidance obligation: it was not the complete state of every job that had been running, in the way z_3 in Example 9.1 was not the complete history of the interrupted composition, but it supported exactly the continuations that the constitutive invariant depended on. Because $\widehat{h}_{\mathrm{AGC}} \sim_{\mathcal{O}_{\mathrm{landing}}} h_{\mathrm{AGC}}$, restarting the executive and letting it repopulate the Core Set from the priority table in order lost the phantom job's accumulated bookkeeping, which was exactly what needed to be lost, while losing nothing the descent itself depended on.

Each restart is thus better described using Chapter 6's vocabulary precisely rather than loosely: the phantom radar-servicing demand was effectively Abort-ed each cycle, since its partial bookkeeping state was destroyed without a preserved continuation; the guidance-critical jobs were Suspend-ed and immediately resumed, since their continuation was recoverable from $\widehat{h}_{\mathrm{AGC}}$ at negligible cost; and no job was silently Forget-ed, since the alarm codes recorded, each time, that a boundary crossing and a restart had occurred. The recurrence of the same alarm code across the descent is not evidence that the recovery mechanism was failing. It is evidence that the same exogenous perturbation kept recreating the same crossing of $\mathcal{B}_{\mathcal{O}}$, and that the system's response to that crossing was working identically and correctly each time it happened.

Chapter 15

The Alarm as a Witness Object

The alarm did not stay internal to the computer. It appeared as a numeric code on the astronauts' display, was read aloud by Buzz Aldrin, and was relayed by Neil Armstrong to Mission Control, where guidance officer Steve Bales had a matter of seconds to decide whether the descent should continue [30, 33]. Bales's decision was informed by a rule established by Jack Garman from an earlier simulation in which the same alarm had appeared under conditions later judged safe: if the alarm was not recurring in a way that indicated guidance data itself was being lost, the descent could proceed. Hamilton's own account, written shortly afterward, describes the underlying software's recovery behavior directly rather than only its outward symptoms, and is worth reading alongside the mission's own archival record of the exchange [27, 30].

This structure is exactly the layered witness object described in Chapter 1's inclusion of $\mathcal{W}$ as a first-class component of the system, now cashed out concretely. The witness recorded, at minimum, that a capacity boundary had been crossed, that the executive's internal recovery had already been invoked and completed, and that the constitutive invariant remained satisfied as far as the computer itself could determine, while leaving open a further question the computer was not positioned to answer alone: whether the mission-level continuation, encompassing consequences well beyond what the executive's own obligation set formalized, remained the right one to pursue. This produces a layered decision structure,

$$\text{automatic triage} \subset \text{human authorization} \subset \text{mission commitment},$$

in which the innermost layer resolved itself, correctly, in well under a second, while the outer layers took the fifteen or so seconds available to a human decision-maker working from a rule prepared in advance. The alarm neither landed the spacecraft nor merely reported a fault after the fact. It connected an internal, already-resolved transition to an external authorization structure that needed, and was given, the chance to disagree.

Chapter 16

What the Apollo Case Does and Does Not Prove

The case demonstrates, concretely, that engineered priority, restart protection built around a deliberately protected sufficient history, and a witness structure that surfaces its own recovery decisions rather than concealing them, can together preserve a constitutive invariant through repeated overload of a resource the original designers had not anticipated being exhausted in quite this way. It demonstrates this for one system, under one incident, documented after the fact through public historical accounts rather than through the original engineering notebooks or assembly listings, and the chapters above should be read with that qualification attached throughout.

Several things the case does not show are worth stating as plainly as the thing it does show. It does not show that scalar priority is generally adequate for sacrifice ordering, the claim Chapters 4 and 5 spent real effort refuting. What it shows is the narrower and more interesting claim that a scalar priority table can be safe once it is deployed only as the terminal, declared collapse map ν_t over a survivor family whose coherence has already been secured by other means, here by restart protection and a protected sufficient history rather than by any real-time reasoning about dependency cones. It does not show that the Accessible Pruning Theorem's hypothesis, contraction accessibility, was ever explicitly verified by the original engineers in those terms; the reading offered here is a retrospective formal description, not a claim about what MIT's Instrumentation Laboratory proved or intended to prove in 1969. And it does not show that every critical system reduces to a fixed priority table plus a protected memory region: the AGC's obligation structure was comparatively simple and largely static across a single descent, which is precisely the condition under which a design-time collapse map can be trusted; systems whose obligation sets shift meaningfully within a single episode of contraction cannot claim the same warrant merely by citing this precedent.

One further piece of historical framing deserves care rather than a loose gesture at biography. Hamilton later formalized a general preventative methodology she named Development Before the Fact, aimed at making entire classes of interface error structurally inexpressible at design time, developed and named across the decades following Apollo rather than during it [28, 29]. It would overstate the connection to claim she gave that name to the AGC's runtime restart architecture itself. What can be said carefully is that the same underlying instinct, that reliability under conditions not individually anticipated depends on what has already been made safe to happen by the time those conditions arrive, is visible in both the earlier engineering and the later, explicitly named methodology. The Apollo case, read this way, is not a proof that intelligent real-time improvisation saved the landing. It is an argument, textured by one well-documented historical

episode, that the improvisation available to any system under contraction is bounded above by how much of the survivor family's geometry was already made safe before the contraction arrived.

Part VI. General Results

Part V's discipline forces a structural correction that the earlier parts of this monograph could not have anticipated on their own. The Apollo case did not realize admissible degradation through Chapter 5's cone-pruning procedure; it realized it through uniform restart backed by a protected sufficient history, a mechanism belonging to Chapters 7 and 9. Treating Chapter 5's apparatus as though it were the definition of admissible degradation, rather than one way of achieving it, would have made the theory unable to recognize its own best worked example. This part repairs that: it states what admissible degradation requires independently of any particular mechanism, and then places the pruning apparatus and the Apollo case as two different, independently verified ways of satisfying that requirement.

Chapter 17

Contraction Operators and the Realization Family

Definition 17.1 (Contraction operator)**.** *A contraction operator is a (possibly partial) map*

$$D : X \times \mathcal{B} \times 2^{\mathcal{O}} \longrightarrow X,$$

taking a pre-contraction state, a resource level, and an obligation set to a post-contraction state.

Definition 17.2 (Admissibility-preserving operator)**.** *D is admissibility-preserving relative to $(b, \mathcal{O})$ when, for every x with $\Gamma(x, b, \mathcal{O}) \neq \varnothing$,*

$$D(x, b, \mathcal{O}) \in K_{\mathcal{O}}.$$

This definition says nothing about how D is implemented. It says only that whenever some obligation-preserving continuation exists at all, D finds a state from which one still does. Several structurally different mechanisms, already developed piecemeal across Parts II and III, are instances of this one definition rather than competitors to it:

$$\mathcal{D} = \{\, D_{\text{prune}},\ D_{\text{restart}},\ D_{\text{rollback}},\ D_{\text{refuse}},\ D_{\text{shed}} \,\}.$$

D_{prune} is Chapter 5's safe cone-removal process, deleting dependent cones one at a time until the survivor set fits the budget. D_{restart} discards the transient state of the entire workload uniformly and reconstructs from a retained history, succeeding exactly when that history is sufficient in Chapter 9's sense. D_{rollback} applies Chapter 7's restart-set machinery to a single interrupted task rather than to the system as a whole, returning that task to some $r \in R_p$ without disturbing the rest of the workload. D_{refuse} acts at admission rather than after the fact, denying new demand entry before it is ever scheduled, leaving an already-running survivor set untouched. D_{shed} drops load in response to contraction without necessarily producing a disposition or a witness for what was dropped; it is included in the family precisely so that Chapter 11's distinction between graceful and ungraceful degradation has an operator to attach to, since D_{shed} can satisfy the admissibility-preservation definition above while still failing the witness requirement developed in the next chapter.

Remark 17.3. *Nothing requires a system to commit to one operator. A single system can run D_{refuse} at its admission boundary, D_{rollback} for individually interruptible tasks, and D_{restart} as a last-resort response to a resource class it cannot otherwise recover from, exactly as the Apollo executive combined ordinary priority-governed scheduling with a restart mechanism reserved for bookkeeping exhaustion specifically. The theory's commitment is to the preservation condition, not to any single member of $\mathcal{D}$.*

Chapter 18

Obligation-Preserving Contraction and Witnessed Recovery

Definition 18.1 (OPCC)**.** *A system $\mathcal{S}$ running operator D satisfies the Obligation-Preserving Contraction Condition, written* $\mathrm{OPCC}(\mathcal{S}, D)$*, when D is admissibility-preserving relative to every $(b, \mathcal{O})$ the system can encounter.*

OPCC combines two things that Chapter 4 already insisted on keeping separate at the level of a single decision: feasibility and selection. Here, at the level of an entire operator, it combines them again, correctly, because an operator that sometimes finds no continuation when one exists has failed at exactly the job Parts II and III were built to formalize, regardless of which mechanism it used to fail.

Preservation alone is not the whole of what Chapter 1 asked for. The witness structure $\mathcal{W}$ was included in the system's definition from the outset, and a theory that verifies preservation while remaining silent about whether the system can say what it did has only answered half the original question.

Definition 18.2 (Operational and complete witness production)**.** *$\mathcal{S}$ has operational witness production,* $\mathrm{WT}_{\mathrm{operational}}(\mathcal{S})$*, when every contraction event produces an externally visible record of the boundary crossed and the Chapter 6 disposition applied to each excluded or restarted task, without that record itself certifying that the resulting state satisfies $\Gamma \neq \varnothing$ going forward. $\mathcal{S}$ has complete witness production,* $\mathrm{WT}_{\mathrm{complete}}(\mathcal{S})$*, when the record additionally constitutes a proof that the resulting state lies in $K_{\mathcal{O}}$.*

The distance between these two is real and is not a matter of degree. An alarm code, a log entry, or a displayed status can satisfy $\mathrm{WT}_{\mathrm{operational}}$ outright while depending on an external party, a human operator, a separate verification process, or simply an engineering argument made in advance, to close the gap to $\mathrm{WT}_{\mathrm{complete}}$. Systems that report having degraded are common; systems whose report is itself a certificate of continued viability are rare, and conflating the two overstates what most real witness structures actually provide.

Definition 18.3 (Accountable admissible degradation)**.**

$$\mathrm{AAD}(\mathcal{S}, D) = \mathrm{OPCC}(\mathcal{S}, D) \wedge \mathrm{WT}_{\mathrm{complete}}(\mathcal{S}).$$

This is the strongest joint property the theory defines, and it should be recognized as strong: it requires both that the system never fails to find an available continuation and that its own witness

structure proves as much. The next chapter exists because almost no interesting real system, including the one this monograph has studied at length, is in a position to claim it outright.

The gap between operational and complete witness production is not merely a matter of typical engineering practice falling short of an ideal. In a precise sense, it is sometimes unbridgeable in principle, independent of any particular system's effort.

Theorem 18.4 (Witness impossibility under observational equivalence)**.** *Let* $\mathrm{obs} : X \to Y$ *be the information available to a witness producer, and suppose there exist states* $x_{\mathrm{v}}, x_{\mathrm{n}} \in X$ *with*

$$\mathrm{obs}(x_{\mathrm{v}}) = \mathrm{obs}(x_{\mathrm{n}}), \qquad \Gamma(x_{\mathrm{v}}, b, \mathcal{O}) \neq \varnothing, \qquad \Gamma(x_{\mathrm{n}}, b, \mathcal{O}) = \varnothing.$$

Then no witness computed solely from $\mathrm{obs}(x)$ *can be a sound and complete certificate of continuation viability for both states.*

Proof. Any witness function taking $\mathrm{obs}(x)$ as its only input must return the same certificate for x_{v} and x_{n}, since they are observationally identical. If that certificate asserts viability, it is unsound at x_{n}; if it asserts non-viability, it is unsound at x_{v}; if it asserts neither, it is incomplete at whichever of the two actually holds. □

This gives $\mathrm{WT}_{\mathrm{operational}}$ and $\mathrm{WT}_{\mathrm{complete}}$ a separation that does not depend on any particular system's history, Apollo's included. A node in a distributed system that can faithfully report "restart occurred" from its own local state may have no access to the global state needed to certify that every obligation remains jointly satisfiable elsewhere in the system; two global configurations, one still viable and one not, can be indistinguishable from that node's vantage point. The theorem says this is not a limitation of the node's diligence. It is a limitation of what any function of $\mathrm{obs}(x)$ alone can ever certify, however carefully computed, whenever obs genuinely conflates a viable state with a non-viable one.

Chapter 19

Scope and Warrant

A claim that a system satisfies OPCC can be true in several different ways, and the earlier draft of this chapter collapsed two independent questions into one linear ranking: how far the claim reaches, and how it was established. Those are not the same axis, and treating them as one produces comparisons that do not actually hold. A formally verified property of a bounded mathematical model is not straightforwardly weaker or stronger than an empirically observed property of one real run; the two are differently warranted, not differently sized. This chapter replaces the earlier single hierarchy with two axes, and is more useful for it, since it stops the theory from implying a total order where the actual structure is only partial.

Definition 19.1 (Scope)**.** *The scope* Σ *of an OPCC claim about* $\mathcal{S}$ *is one of:*

$$\Sigma \in \{\, \textit{episode},\ \textit{finite tested class},\ \textit{declared disturbance class},\ \textit{universal} \,\},$$

indicating, respectively, a single recorded history and event; a specific finite collection of histories or simulated runs; every disturbance in a declared class E*; or every reachable state and disturbance* $\mathcal{S}$ *could in principle encounter.*

Definition 19.2 (Warrant)**.** *The warrant* W *of an OPCC claim is one of:*

$$\mathsf{W} \in \{\, \textit{observed},\ \textit{tested},\ \textit{analytically proved},\ \textit{mechanically verified} \,\},$$

indicating, respectively, that the claim rests on what was seen to happen; on results across a battery of runs; on a mathematical proof from stated hypotheses; or on a machine-checked derivation of that proof.

Definition 19.3 (Claim signature)**.** *An OPCC claim about* $\mathcal{S}$ *carries a signature* $\Lambda = (\Sigma, \mathsf{W})$*. The scope axis is ordered by inclusion of what is covered; the warrant axis is ordered by strength of justification for a fixed scope. Across the two axes jointly,* Λ *is only partially ordered: a stronger warrant at a narrower scope is not comparable to a weaker warrant at a wider scope.*

The definitions already in use survive this reframing; they simply occupy specific cells rather than rungs of one ladder.

Definition 19.4 (Observed satisfaction, restated)**.** *For a specific recorded history* h *and disturbance event* e *that occurred along it,*

$$\mathrm{ObservedOPCC}(\mathcal{S}, e, h)$$

holds when the continuation actually realized along h *in response to* e *lay in the admissible family, given the resource level and obligation set in force at that point. This is the claim of signature* $\Lambda = (\textit{episode}, \textit{observed})$*.*

Definition 19.5 (Class-relative satisfaction, restated)**.** *For a declared disturbance class E and obligation regime $\mathcal{O}$,*

$$\mathrm{OPCC}_{E,\mathcal{O}}(\mathcal{S})$$

holds when, for every reachable state x and every contraction b induced by some $e \in E$,

$$\mathfrak{A}(x, b, \mathcal{O}) \neq \varnothing \implies D(x, b, \mathcal{O}) \in \mathfrak{A}(x, b, \mathcal{O}).$$

This fixes Σ = declared disturbance class, but leaves W open: the same scoped claim can carry warrant tested, when established by exhaustive simulation across E, or warrant analytically proved, when established by an argument covering every member of E from stated hypotheses. These are different claims about the same scope, not the same claim at different strengths, and a monograph or a system's own documentation should say which one it is making.

A stated model theorem, such as the Accessible Pruning Theorem of Chapter 5, occupies $\Lambda = (\text{universal}, \text{analytically proved})$, universal in the sense that it holds for every system satisfying the theorem's stated hypotheses, not in the sense that the hypotheses themselves are guaranteed to hold of any particular real system. That qualification is not a weakness specific to this theorem; it is what every proved theorem means, and stating it here is only to head off the temptation to read "universal" as "unconditional."

Claim	Scope Σ	Warrant W	Example
Observed $\mathrm{OPCC}(\mathcal{S}, e, h)$	episode	observed	Apollo 11's descent, one recorded event
$\mathrm{OPCC}_{E,\mathcal{O}}(\mathcal{S})$, tested	declared class	tested	SpherePOP's proof-of-concept trace
$\mathrm{OPCC}_{E,\mathcal{O}}(\mathcal{S})$, proved	declared class	analytically proved	SpherePOP's formal transition system, relative to a declared policy
Accessible Pruning Theorem	universal (within hypothesis)	analytically proved	Chapter 5's D_{prune} result
$\mathrm{OPCC}(\mathcal{S}, D)$ unqualified	universal	(any)	the property no system in Part VII actually claims outright

Remark 19.6. *Two cells that look adjacent are not automatically comparable. A proof about a formal transition system that has never been run (declared class, analytically proved) says nothing about whether the deployed implementation matches that transition system; an extensively tested proof-of-concept (declared class, tested) says nothing about disturbances outside the tested class, however analytically well-understood the underlying model is. Neither dominates the other, and a monograph, or a system's documentation, that reports only one cell while implying the other has also been earned is making an unwarranted claim, not a conservative one.*

Chapter 20

The Preservation Theorem

Theorem 20.1 (General preservation)**.** *If D is admissibility-preserving relative to $(b, \mathcal{O})$, then for every state x with $\Gamma(x, b, \mathcal{O}) \neq \varnothing$, applying D yields a state from which some obligation-preserving continuation remains available.*

The theorem is close to definitional, deliberately. Its purpose is to be the fixed point that every operator-specific result in this monograph is a corollary of, rather than an independent claim standing beside them.

Corollary 20.2 (Pruning realization)**.** *D_{prune} is admissibility-preserving relative to $(b, \mathcal{O})$ whenever the survivor family $\mathfrak{S}_{\mathcal{O}}$ is contraction-accessible from the full workload P, by the Accessible Pruning Theorem of Chapter 5.*

Proposition 20.3 (Apollo 11 episode)**.** *Under the abstraction adopted in Part V, D_{restart} is admissibility-preserving for the descent obligation set $\mathcal{O}_{\text{descent}}$ under the modeled executive-overflow disturbance class $E_{1201/1202}$, because the protected guidance state retained across each restart satisfies*

$$\widehat{h}_{\text{restart}} \sim_{\mathcal{O}_{\text{descent}}} h_{\text{pre-overflow}},$$

so that restarting from $\widehat{h}_{\text{restart}}$ and re-admitting jobs in priority order yields a state in $K_{\mathcal{O}_{\text{descent}}}$.

The proof of this proposition rests on Chapter 9's continuation-equivalence relation, not on Chapter 5's cone-removal apparatus; priority enters only afterward, in determining the order in which jobs are re-admitted once the resource that had been exhausted, bookkeeping capacity, is again available. Priority is not the operation that performed the recovery, and stating the proposition this way keeps that distinction visible rather than letting the presence of a priority table suggest that Chapter 5's machinery must be doing the work.

Remark 20.4. *The historically supportable signature of this proposition, in the terms of the previous chapter, is $\Lambda = (\textit{declared class}, \textit{analytically proved})$ for the modeled disturbance class $E_{1201/1202}$, resting on the abstraction adopted in Part V rather than on the original assembly-language implementation. The single most defensible statement about the actual historical episode is narrower still, and carries a different signature entirely, $(\textit{episode}, \textit{observed})$, not a weaker version of the same claim but a different one:*

$$\text{ObservedOPCC}(\mathcal{S}_{\text{AGC}}, e_{1202}, h_{\text{Apollo 11}}).$$

This monograph does not claim $\Lambda = (\textit{universal}, \cdot)$ for the Apollo Guidance Computer at any warrant level; it claims that the documented history is consistent with, and well explained by, the general

theorem above under the declared-class reading, and separately, and on different grounds, that the one recorded episode itself succeeded. On the witness side, the alarms displayed and radioed during descent give $\mathcal{S}_{\mathrm{AGC}}$ *operational witness production: they recorded that a boundary had been crossed and that recovery had been invoked. They do not by themselves constitute complete witness production, since the judgment that the resulting state remained obligation-preserving at the mission level was supplied externally, by Jack Garman's pre-established rule and Steve Bales's authorization, rather than certified by the alarm itself. The system is therefore properly described as an instance of*

$$\mathrm{AAD}_{\mathrm{obs}}(\mathcal{S}_{\mathrm{AGC}}) = \mathrm{ObservedOPCC}(\mathcal{S}_{\mathrm{AGC}}, e_{1202}, h_{\mathrm{Apollo\,11}}) \wedge \mathrm{WT}_{\mathrm{operational}}(\mathcal{S}_{\mathrm{AGC}}),$$

an observed instance of accountable admissible degradation, not a proof of the unrestricted property $\mathrm{AAD}(\mathcal{S}_{\mathrm{AGC}}, D_{\mathrm{restart}})$.

Chapter 21

Impossibility Results

Positive results are only as informative as the negative results that bound them. Three impossibility results, each already implicit in earlier chapters, are worth stating together as a single family of limits on what any operator in $\mathcal{D}$ can achieve.

Proposition 21.1 (Capacity impossibility)**.** *No contraction operator can be admissibility-preserving at $(b, \mathcal{O})$ when $\mathfrak{A}(b, \mathcal{O}) = \varnothing$.*

This is immediate from the definitions of Chapter 10, and it is stated formally only to close off a temptation the rest of the monograph has otherwise tried to guard against, that sufficiently clever recovery machinery can substitute for capacity that is not there. When mandatory obligations' joint demand exceeds every available resource level, no member of $\mathcal{D}$, however constructed, produces a continuation, because none exists to produce.

Proposition 21.2 (Recoverability impossibility)**.** *No operator relying on rollback or restart can reconstruct an obligation-preserving continuation across an interruption at state x if no retained representation $\widehat{h}$ satisfies $\widehat{h} \sim_{\mathcal{O}} h$, regardless of how much raw state has been retained.*

This restates Chapter 9's volume-versus-sufficiency distinction as a limit on D_{restart} and D_{rollback} specifically. A system can retain an arbitrarily large archive of pre-interruption state and still fail this proposition's hypothesis, if the retained archive happens not to include the particular coordinates a downstream obligation depends on. Sufficiency, not volume, is what the proposition's hypothesis is about, and no amount of the wrong kind of retention satisfies it.

Proposition 21.3 (Local-priority insufficiency, general form)**.** *No contraction operator that selects survivors by a fixed local scalar score, applied one removal or admission decision at a time without reference to a precomputed target, is admissibility-preserving in general, even when $\mathfrak{A}(b, \mathcal{O}) \neq \varnothing$, unless either the survivor family is contraction-accessible from the current state, or the scalar score has itself been fixed in advance as a declared collapse map ν_t over a survivor family whose coherence was already secured by other means.*

The first disjunct restates Chapter 5's Local-Priority Insufficiency proposition and its example directly. The second disjunct is what Part V's honest reading of the Apollo priority table actually licenses: the AGC's fixed priority numbers are not a counterexample to this proposition, because they were never used to make real-time decisions about an uncertain, evolving survivor family. They were applied only after D_{restart} had already reduced the problem to re-admitting jobs into a bookkeeping structure whose sufficiency for the landing obligation had been secured in advance by the protected guidance state. A scalar priority table used this way, as the second disjunct's escape

clause rather than as the whole mechanism, is exactly the case Chapter 4 always intended to leave open.

Admissible degradation, on the view this part completes, is a property a system either has or lacks with respect to a stated operator, disturbance class, and obligation regime; it is not a single procedure. Chapters 4 and 5 supply one way of realizing it, built from first principles and provably correct under a stated structural hypothesis. Part V supplies a second, historically grounded way, built from restart and protected sufficient history rather than from cone-pruning, and provably correct only relative to a documented disturbance class rather than universally. Keeping these as two realizations of one property, rather than treating either as the definition of the whole, is what allows the theory to recognize both.

Part VII. Applications

The formal spine built across Parts I through VI is now stable enough that the applications that follow can function as comparative tests rather than further theory-building. Each is given the same analytic signature, so that describing a system in this vocabulary is a disciplined act rather than a loose analogy.

Definition 21.4 (Analytic signature)**.** *A system's admissible-degradation profile is the tuple*

$$\mathfrak{C} = (\mathcal{O},\ E,\ \mathcal{D}_{\text{impl}},\ H,\ W,\ \Lambda),$$

where $\mathcal{O}$ is the declared obligation set, E is the contraction class the system is built to survive, $\mathcal{D}_{\text{impl}} \subseteq \mathcal{D}$ is the set of contraction operators the system actually implements, H is what it retains as history across contraction, W is its witness mechanism, and $\Lambda = (\Sigma, \mathsf{W})$ is the scope-and-warrant signature, in Chapter 19's terms, that the system can honestly claim.

Filling in $\mathfrak{C}$ for a real system is itself the analytic work of each chapter that follows. A system that cannot have $\mathcal{O}$ filled in at all, because its obligations were never declared, has already told the theory something important about itself before any question about E, $\mathcal{D}_{\text{impl}}$, H, or W is even asked.

Application	Principal operators	Likely Λ	Central failure
Distributed computing	Refuse, Shed, Restart, Rollback	Tested or verified, for bounded fault models	Silent loss disguised as successful service
MEM|8	Refuse, Rollback, Restart via replay	Observed, over tested traces; model-level where formally specified	Retention mistaken for sufficiency
SpherePOP	Refuse before commitment; rollback before collapse	Verified model, plus observed traces	Confusing recorded, verified, and admissible events
AI context management	Shed; approximate pruning; retrieval as reconstruction	Usually below observed, since $\mathcal{O}$ is undeclared	Invisible truncation and obligation drift
Institutions	Triage, refusal, shedding, deferral	Episode-relative at best	Contested obligations, hidden sacrifice ordering

Chapter 22

Distributed Systems

Distributed computing is the most technically conventional of the five applications, and for that reason it serves best as the calibration case: a domain where the vocabulary built in Parts II and III can be checked against mechanisms that are already well understood on their own terms.

A circuit breaker instantiates Refuse: it declines new work before that work is admitted, leaving whatever is already committed untouched. Load shedding instantiates D_{shed}: it can discard work already in flight, including work a client believes has been accepted. Process supervision, restarting a crashed worker from a checkpoint or from scratch, instantiates D_{restart}. Transactional compensation and checkpoint restoration instantiate D_{rollback}, returning the system to an earlier admissible boundary rather than discarding forward. Each of these mechanisms has its own substantial independent literature [19, 20, 23, 24], which this chapter is not attempting to summarize so much as to re-read through Chapter 6's dispositions.

Remark 22.1. *The four are not interchangeable, and the habit of calling all of them "fault tolerance" obscures exactly the distinction Chapter 6 was built to preserve. A circuit breaker's refusal and a load shedder's discard look similar from a distance, both reduce the accepted workload, but they carry different historical obligations to the request that was affected. A refused request was never admitted and owes nothing further. A shed request may have been admitted, may even have been partially executed, and its disposition needs to be recorded as* Abort *rather than silently folded into the same category as a request that never arrived.*

Definition 22.2 (Request history)**.** *For a request r, its history $h(r)$ takes values in*

$$\{\,\mathsf{unseen},\ \mathsf{admitted},\ \mathsf{committed},\ \mathsf{refused},\ \mathsf{aborted},\ \mathsf{completed}\,\}.$$

A distributed system fails WT precisely when a client cannot distinguish two histories that call for different responses, most importantly refused from committed-but-acknowledgment-lost. These two histories look identical from outside if the only signal available is silence, yet they demand opposite next actions: a refused request can be safely retried, while a committed request retried carelessly risks duplication. This chapter's central point follows directly: exactly-once execution, the property distributed systems research has spent decades chasing, is frequently less important in practice than exactly-once disposition witnessing. A system need not eliminate the uncertainty produced by a lost acknowledgment. What Chapter 18's WT actually requires is that the system preserve, and make available, which uncertainty remains, so the client's own recovery logic can act on an honest disjunction rather than a false certainty in either direction.

Chapter 23

MEM|8

MEM|8 is best read as the historical-sufficiency application, since its governing question is not how much information a contraction event allows the system to retain but whether what it retains preserves the same obligation-relative continuation set as what it discarded:

$$\widehat{h} \sim_{\mathcal{O}} h.$$

Chapter 9 already showed that payload retention alone does not answer this question. A system can retain content in full while losing the provenance, binding context, event ordering, verification status, or refusal status that a downstream obligation actually depends on, and Chapter 9's distinction between volume and sufficiency applies here without modification.

It is useful to separate three distinct things a memory system can be contracting:

$$D_{\text{payload}}, \qquad D_{\text{provenance}}, \qquad D_{\text{continuation}}.$$

D_{payload} retains content: the bytes themselves. $D_{\text{provenance}}$ retains content's historical relations: where it came from, what produced it, what it superseded. $D_{\text{continuation}}$ retains specifically what the system needs to know to resume its outstanding obligations, which may be a small fraction of either of the first two. Only $D_{\text{continuation}}$ directly targets historical sufficiency as Chapter 9 defined it; a system optimized to maximize D_{payload} or even $D_{\text{provenance}}$ can still fail $\widehat{h} \sim_{\mathcal{O}} h$ if neither was built with $\mathcal{O}$ in view.

Remark 23.1. *A hash concatenated from multiple retained fragments can appear, on inspection, to certify that the underlying material has been faithfully preserved, while actually permitting more than one distinct reconstruction consistent with that hash. This is a sharp illustration of the general failure mode this chapter is about: apparent retention without unambiguous reconstruction is not a rare edge case but close to the default outcome whenever a system is built to maximize D_{payload} without an explicit argument connecting that payload back to $\mathcal{O}$.*

MEM|8's own claim to admissible degradation should be split rather than stated as a single point on one scale, in exactly the sense Chapter 19 makes precise. Its abstract replay conditions, stated as properties of the formal architecture independent of any particular run, can support a claim of signature (declared class, analytically proved) when the stated hypotheses are checked directly against the specification. Its write-ahead log, guardian-replay mechanism, and related experimental infrastructure support a differently signatured claim, (finite tested class, tested), over the specific histories that have actually been exercised,

$$\mathrm{OPCC}_{E_{\text{test}},\mathcal{O}}(\mathcal{S}_{\text{MEM8}}), \quad \text{warrant tested, not proved.}$$

Treating the second as though it inherited the strength of the first would repeat, in a different domain, exactly the error Part V took care to avoid for Apollo.

Chapter 24

SpherePOP

SpherePOP is the cleanest application of refusal before irreversible commitment available in this monograph, and its principal theoretical contribution is not a superior recovery operator but a pipeline structured so that several kinds of inadmissible event never reach the recovery problem in the first place:

$$\text{Pop} \longrightarrow \text{Refuse} \longrightarrow \text{Bind} \longrightarrow \text{Transform} \longrightarrow \text{Verify} \longrightarrow \text{Collapse}.$$

Each arrow changes the status of the event passing through it, and the chapter's central discipline is to keep four predicates that are easy to conflate strictly distinct: an event's mere existence in the input stream, its having been successfully transformed, its having been successfully verified, and its having been granted permission to collapse into the operational ledger. An event can satisfy any prefix of this list without satisfying the next member; treating them as one property, "the event happened," is precisely the conflation Chapter 6's four dispositions exist to prevent at the level of a single pipeline.

A representative observed trace: an event is initially refused with an explicit rejection code, later re-submitted and admitted, transformed along one of the pipeline's declared transformation paths, verified against the relevant policy, and finally committed. This trace supports an observed property of the pipeline, an instance of ObservedOPCC over that specific run. It does not by itself establish the pipeline's general correctness. A stronger claim, that every event satisfying the declared admission policy is eventually either refused with a recorded reason or collapsed with a verified transformation, is a claim about the formal transition system underlying the pipeline, and it should be proved, when it is proved at all, relative to a declared policy and a bounded class of transformations rather than asserted from the observed trace alone.

SpherePOP's characteristic operator is therefore D_{refuse}, deployed at the earliest point in the pipeline where inadmissibility can be detected. D_{rollback} remains relevant up through Verify, where an event that fails verification can still be returned to an earlier stage without cost. Past Collapse, this monograph's ordinary recovery vocabulary stops applying cleanly, because the commitment recorded there may be irreversible in Chapter 8's sense. What is needed beyond that point is not rollback but what might be called compensating continuation: a forward-only correction that acknowledges the collapsed commitment as a fixed fact and constructs a new admissible continuation around it, rather than one that attempts, impossibly, to make the collapse not have happened, the same insight distributed-transactions research reached independently under the name of a saga [21, 22].

Chapter 25

Context Saturation in AI Systems

This chapter is the negative application, in the sense that it names a domain where the vocabulary developed so far applies cleanly to the mechanisms in use while revealing that the accountability those mechanisms owe is, in current practice, rarely supplied. Contemporary AI systems perform contraction routinely: conversations exceed context windows, documents exceed what can be attended to at once, retrieved material competes for a fixed budget of tokens. What is usually missing is a declared $\mathcal{O}$ against which any of this contraction could be evaluated, and a witness sufficient to show what was given up.

Context-window truncation is an instance of D_{shed}: material is dropped once a budget is exceeded, typically by position or recency rather than by any assessment of which material an outstanding obligation depends on [35, 36]. Retrieval is an attempted reconstruction operator, in Chapter 9's sense, standing in for a sufficient history the system did not itself retain [34]. Summarization is an obligation-relative compression only if the obligations it is meant to preserve were declared before the compression was performed; a summary produced without a declared $\mathcal{O}$ is compression relative to an unstated and possibly inconsistent target. High output quality, and even the appearance that attention was well allocated across the retained material, is not evidence that the discarded material was safe to discard, since quality is assessed against the same output the contraction already shaped.

This yields a conditional result stronger than a mere caution:

$$\text{no declared } \mathcal{O} \implies \text{no evaluable OPCC claim.}$$

The reasoning is not merely that declaring $\mathcal{O}$ would be nice practice. Without a declared obligation set fixed in advance, any apparent confirmation that the correct material was preserved is reconstructed after the fact from whatever the system actually produced, which makes the claim self-sealing: the obligations are inferred from the output, and the output is then judged against those same inferred obligations. A claim that cannot fail this way has no signature Chapter 19's classification can assign it, since even the narrowest, most modest cell available, episode scope with observed warrant, presupposes an obligation set fixed independently of the single history being checked against it.

The strongest witness typical current systems supply is operational at best: a system may report that truncation or summarization occurred, without supplying anything that would count as evidence that every obligation-supporting element of the discarded material remains recoverable elsewhere. AI context management, on the evidence available, therefore usually satisfies neither OPCC nor complete witness production, regardless of how strong the surface quality of its output appears.

Remark 25.1. *This domain also exhibits a complication Apollo's case did not, and the two should not be confused. Within a single conversation, new instructions can revise the obligation set itself, not merely the resource level:*

$$\mathcal{O}_t \neq \mathcal{O}_{t+1}.$$

This is a moving-target problem in the fullest sense: the survivor family $\mathfrak{A}(b, \mathcal{O}_t)$ that was correct to aim for at time t may no longer be the one relevant at $t+1$, independent of any change in b. Apollo's descent obligations were comparatively stable across the incident this monograph examined, which is one of the structural reasons its priority table could function as a safe, design-time-fixed collapse map. Nothing about that case licenses a conclusion about systems whose obligation set itself is a live variable.

Chapter 26

Institutional Triage

This chapter should be the most restrained of the five, because an institution's obligation set is not merely difficult to observe from the outside; it is, in a meaningful sense, contested rather than given. Hospitals, utilities, emergency-response agencies, and administrative systems all use refusal, deferral, shedding, and priority-based triage under contraction, and their constitutive invariants carry a normative content that this monograph's formalism is not positioned to settle [37, 38, 39].

What the formalism can do is reveal structure that would otherwise stay implicit: who or what was sacrificed, under which contraction, by which authorized relation, and with what witness. What it cannot do is derive, from resource data alone, which obligation set an institution ought to have declared. The theory can make a sacrifice ordering visible and precise once it has occurred; it does not adjudicate whether that ordering was the right one.

Definition 26.1 (Declared and revealed obligations)**.** *An institution's declared obligation set $\mathcal{O}^{\mathrm{decl}}$ is what it states, in charter, policy, or public commitment, that it owes. Its revealed obligation set under a recorded history of contraction, $\mathcal{O}^{\mathrm{rev}}(h)$, is the obligation set that the survivor sets actually realized along h would have had to be constitutive in order to explain the sacrifices that were actually made.*

A divergence

$$\mathcal{O}^{\mathrm{decl}} \neq \mathcal{O}^{\mathrm{rev}}(h)$$

is empirical evidence, not merely rhetorical suspicion, that an institution's actual sacrifice ordering under stress differs from what its public constitution claims. This gives precise, checkable form to a claim this monograph has made informally since its opening pages: a system's declared purposes are tested, and sometimes contradicted, by what it turns out to be willing to give up once resources genuinely run short.

Chapter 27

Comparative Synthesis

Read across the five applications, no single contraction operator is intrinsically graceful, and this is the chapter's central finding rather than a hedge. D_{restart} fails outright without a protected, sufficient history to reconstruct from. D_{rollback} fails once the boundary it would roll back to lies on the far side of an irreversible commitment, as SpherePOP's post-Collapse discussion showed directly. D_{refuse} fails when the very act of refusing violates some other obligation the system also owes, for instance an obligation to respond at all. D_{prune} fails without contraction accessibility, exactly as Chapter 5 proved. And D_{shed} fails on accountability specifically, producing no disposition witness even when the underlying resource arithmetic happens to work out.

The general conclusion is therefore

$$\text{gracefulness} \neq \text{operator type};$$

gracefulness is instead a joint property of the operator together with everything the operator depends on to succeed. That joint property can be stated precisely enough to be a theorem rather than a summary, once it is named as something more specific than the informal "gracefulness" Chapter 11 introduced, since that definition required invariant preservation and witnessed disposition but never universally required historical sufficiency; a permanently refused task need not remain recoverable, and Chapter 11's notion should not be asked to carry more than it claims.

Definition 27.1 (Accountable recoverable degradation)**.** *A degradation episode is accountably recoverable relative to $\mathcal{O}$, written* $\text{ARD}_{\mathcal{O}}$*, when:*

$D_{\mathcal{O}}$*: its degradation operator returns the system to an obligation-preserving continuation;*

$F_{\mathcal{O}}$*: the operative obligation model is fixed independently of the outcome being evaluated;*

$H_{\mathcal{O}}$*: every task designated for future resumption retains a historically sufficient representation; and*

$W_{\mathcal{O}}$*: every excluded or transformed task receives an adequate disposition witness.*

Theorem 27.2 (Four-factor characterization)**.**

$$\text{ARD}_{\mathcal{O}} \iff D_{\mathcal{O}} \wedge F_{\mathcal{O}} \wedge H_{\mathcal{O}} \wedge W_{\mathcal{O}}.$$

This biconditional is close to definitional, in the same sense several results earlier in this monograph have been flagged as close to definitional: $\text{ARD}_{\mathcal{O}}$ is defined as the conjunction, so the theorem mainly fixes the name. The substantive content is not the biconditional itself but that none of the four conjuncts is redundant given the other three.

Proposition 27.3 (Independence of the four factors)**.** *Each of $D_{\mathcal{O}}, F_{\mathcal{O}}, H_{\mathcal{O}}, W_{\mathcal{O}}$ can fail while the other three hold.*

Proof. Four episodes, one per factor, exhibit this.

$D_{\mathcal{O}}$ failing: an episode with a fixed obligation model, a fully sufficient retained history, and a complete witness trail, in which the transition actually taken nonetheless lands outside $I_{\mathcal{O}}$, a defect in the operator itself rather than in its bookkeeping.

$F_{\mathcal{O}}$ failing: an episode in which $\mathcal{O}$ is inferred retrospectively from the surviving output, exactly Chapter 25's self-sealing case. The operator, history, and witness can each look correct relative to that retroactively chosen $\mathcal{O}$, but the check is unfalsifiable rather than a real constraint, since no obligation was fixed independently for the outcome to be measured against.

$H_{\mathcal{O}}$ failing: Chapter 9's Example 9.1 directly. Restart is invoked and witnessed, and the obligation model was fixed in advance, but the retained representation $\{z_3\}$ omits m, v, q, ℓ, so a later resumption that needs them fails despite every other factor having held at the moment of restart.

$W_{\mathcal{O}}$ failing: an episode in which the invariant is genuinely preserved and the obligation model was fixed in advance, using a sufficient retained history, but excluded tasks receive no recorded disposition, Chapter 6's Forget outright. □

Remark 27.4. *The independence result is what upgrades Chapter 11's earlier summary into a minimality claim: not merely that these four things together suffice for accountable recoverable degradation, but that no proper subset of them does, for any of the four. A system missing exactly one factor is not slightly less accountable; it fails in the specific, identifiable way that factor's own counterexample demonstrates, and the remaining three factors, however well satisfied, do not compensate for it.*

This synthesis is also what allows the monograph to return to Apollo one last time without overclaiming on its behalf. The AGC exemplifies one successful configuration of $D_{\mathcal{O}} \wedge F_{\mathcal{O}} \wedge H_{\mathcal{O}} \wedge W_{\mathcal{O}}$: uniform restart, a priority table used only as a design-time-fixed collapse map applied after that history had already secured recovery, a protected and genuinely sufficient history, and a witness structure distributed across machine output, procedural knowledge, and human authorization. MEM|8, SpherePOP, ordinary distributed services, AI context management, and institutional triage occupy other points in the same four-dimensional space, some closer to Apollo's configuration than others, and none of them, Apollo included, should be read as the template the others are measured against. Each is measured, instead, against the same four independent requirements, which is the only sense in which this monograph's opening question, what must a system still be able to do once it can no longer do everything, ever receives a general answer.

Admissible degradation begins as an engineering problem but becomes a mathematical theory when survivor sets, capacity regions, history classes, and contraction paths can be studied without presupposing the machines that first made them visible.

Part VIII. Two Extensions

Two questions were deliberately left open earlier in this monograph rather than answered with an unearned theorem: a precise structural condition under which *some* greedy rule is provably safe (Chapter 5 called this a genuine open design question), and a formal treatment of what happens when the obligation set itself changes mid-episode (Chapter 25 named the problem without building the apparatus to handle it). This part answers the first fully and the second as far as a first pass reasonably can, in each case being explicit about where the new results stop.

Chapter 28

A Sufficient Condition for Greedy Pruning

Chapter 5 showed that a scheduler removing the cheapest currently dispensable task first can terminate outside the admissible family even when an admissible set was reachable, and identified the culprit as overlapping alternative supports: removing part of one obligation's cheap alternative can leave its expensive alternative as the only remaining option. It deferred the natural next question, what structural condition on the obligation-support system licenses a greedy rule after all, as open. This chapter answers it, using the alternative-branch notation $\mathcal{S}_o$ already introduced in Chapter 10, and the answer is narrower than the most obvious guess.

Before constructing a concrete policy, it is worth stating what *any* policy must satisfy to succeed, since this separates the definitional part of the problem from the part that requires real structure.

Definition 28.1 (Viable pruning basin)**.** *For budget b, the viable pruning basin is*

$$V_b = \{\, A \in \mathfrak{S}_{\mathcal{O}} : \exists\, A^* \in \mathfrak{A}(b, \mathcal{O}) \text{ with } A \rightsquigarrow^* A^* \,\},$$

the survivor sets from which some safe-pruning path to an admissible set exists.

Definition 28.2 (Budget-safe policy)**.** *A deterministic pruning policy π, writing $A \rightsquigarrow_\pi A'$ for its selected successor, is budget-safe on V_b when, for every $A \in V_b$ with $C(A) \not\leq b$,*

$$A \rightsquigarrow_\pi A' \implies A' \in V_b.$$

Theorem 28.3 (Greedy completeness criterion)**.** *Let P be finite and suppose every pruning step removes at least one task. A pruning policy π reaches an admissible survivor from every $A \in V_b$ if and only if π is budget-safe on V_b.*

Proof. If π is budget-safe, every over-budget successor remains in V_b; since P is finite and each step strictly shrinks the survivor set, the trajectory terminates, and it cannot terminate at an over-budget vertex of V_b, since such a vertex has a path to $\mathfrak{A}(b, \mathcal{O})$ and therefore a safe successor by budget-safety. Hence it terminates in $\mathfrak{A}(b, \mathcal{O})$. Conversely, if π is not budget-safe, some $A \in V_b$ has a selected successor $A' \notin V_b$, from which no continuation of π can reach an admissible survivor by definition of V_b, so π is not complete. □

Remark 28.4. *This criterion is close to definitional, in the same sense Chapter 20's Preservation Theorem is close to definitional: it restates what success means in terms of the basin V_b rather than deriving success from independent structure. Its value is organizational, not explanatory. It converts*

"does this policy work" into "does this policy stay inside V_b," which is exactly the question the rest of this chapter answers for one concrete, natural policy under an explicit structural hypothesis, rather than leaving "budget-safe" as an unexplained property to be checked by hand for every candidate rule.

Definition 28.5 (Disjoint-branch system)**.** *The obligation-support family $\mathcal{S} = \bigcup_{o \text{ mandatory}} \mathcal{S}_o$ satisfies:*

(D1) Disjointness*: the members of $\mathcal{S}$ are pairwise disjoint.*

(D2) Independence*: each $S \in \mathcal{S}$ is itself an order ideal, and no dependency edge of $\sqsubseteq_{\text{dep}}$ links a task in one member of $\mathcal{S}$ to a task in a different member, or to any task outside $\bigcup \mathcal{S}$.*

A system satisfying (D1) and (D2) is a disjoint-branch system. Tasks belonging to no branch, supporting no obligation, are treated as their own trivial, always-dispensable branches.

Disjointness alone does not settle how a greedy rule should compare candidates, and the most natural-looking fix to Chapter 5's failure, simply reversing the comparison to remove the most expensive currently dispensable *task* first, is not sufficient on its own.

Example 28.6 (Why per-task reversal is not enough)**.** *Let o have two alternative branches, $S_1 = \{p_1, p_2\}$ with $c(p_1) = c(p_2) = 3$, so $C(S_1) = 6$, and $S_2 = \{q\}$ with $c(q) = 5$, so $C(S_2) = 5$. The branch S_2 is cheaper overall and is the one a minimum-cost admissible set would keep. All three tasks are simultaneously dispensable at the start, since either branch alone covers o. A rule comparing raw per-task cost removes q first, since $5 > 3$, exactly backwards: S_2 is destroyed while the more expensive S_1 is still fully intact, so p_1 and p_2 freeze as the sole remaining support. The process terminates at $\{p_1, p_2\}$, cost 6, which fails any budget strictly between 5 and 6, even though $\{q\}$, cost 5, was reachable by removing p_1 and p_2 first instead.*

The comparison has to be made at the level of whole branches, not individual tasks.

Definition 28.7 (Branch-level expensive-first policy)**.** *A branch $S \in \mathcal{S}_o$ is branch-dispensable at survivor set A when some other $S' \in \mathcal{S}_o$, $S' \neq S$, is fully contained in A. The policy π_{bexp} repeatedly selects, among currently branch-dispensable branches (treating an obligation-irrelevant task as its own trivial branch), one of maximum cost $C(S)$, and removes every task in it, in any order; it halts when the resource budget is met or no branch-dispensable branch remains.*

Lemma 28.8 (Atomic removability of a dispensable branch)**.** *If S is branch-dispensable at A via a fully intact S', then every task in S is individually safely prunable, in any order, throughout the removal of all of S, provided S' remains untouched during that removal.*

Proof. At any point during the removal, some prefix of S has already been removed and S' is still fully present, which alone satisfies o by definition of an alternative branch; removing the next task of S therefore preserves $I_{\mathcal{O}}$ regardless of which tasks of S remain. □

Lemma 28.9 (Within-family convergence)**.** *Applied to a single obligation's branch family $\mathcal{S}_o = \{S^1, \ldots, S^k\}$ in isolation, sorted so $C(S^1) \leq \cdots \leq C(S^k)$, repeated application of π_{bexp} removes $S^k, S^{k-1}, \ldots, S^2$ in that order and never removes S^1.*

Proof. As long as two or more branches remain, every remaining branch is branch-dispensable, since any other surviving branch witnesses it. The policy selects the maximum-cost branch-dispensable branch, which, among the surviving branches, is S^k on the first round, S^{k-1} on the second once S^k is gone, and so on. Once only S^1 remains, no other branch witnesses its dispensability, so S^1 is no longer branch-dispensable and π_{bexp} cannot select it. □

Theorem 28.10 (Disjoint-branch sufficiency)**.** *Under a disjoint-branch system, π_{bexp} reaches a member of $\mathfrak{A}_t$ whenever $\mathfrak{A}_t \neq \varnothing$.*

Proof. By (D1) and (D2), the branch families of distinct obligations, and the trivial branches of obligation-irrelevant tasks, share no tasks and no dependency edges, so the elimination order within one obligation's family, established by the previous lemma, is unaffected by whatever interleaving occurs with other families. If π_{bexp} halts early because the budget is met, the halting condition itself, together with the fact that every intermediate set along the way preserves $I_{\mathcal{O}}$ by the atomic-removability lemma, places the halting state in $\mathfrak{A}_t$ directly. If instead π_{bexp} runs to exhaustion, the previous lemma guarantees the terminal state retains, for every choice-obligation, exactly its cheapest branch S_o^1, and retains no obligation-irrelevant task, which is exactly the minimum-cost configuration achievable under (D1) and (D2), since the branch families are independent and the total cost decomposes as a sum of per-obligation minima. This minimum cost is a lower bound on the cost of every member of $\mathfrak{A}_t$, so if $\mathfrak{A}_t \neq \varnothing$, the minimum itself satisfies the budget, and the terminal state belongs to $\mathfrak{A}_t$. □

Remark 28.11. *Example 28.6 is worth keeping in view rather than treating as a curiosity. It shows that disjointness of alternative supports does not, by itself, license any greedy repair; the repair also has to compare candidates at the correct granularity. A rule that looks like the obvious fix to Chapter 5's counterexample, reverse the direction of comparison, silently reintroduces a new failure mode the moment branches differ in size. The sufficient condition this chapter actually establishes is narrower and more specific than "prefer expensive tasks": prefer expensive* branches*, compared as wholes.*

Remark 28.12. *The disjointness hypothesis (D1) is strong. The general laminar case, where alternative supports may be properly nested rather than only disjoint or identical, such as a shared mandatory prerequisite that several obligations draw on simultaneously, is not covered by the theorem above. Worked examples of such nesting behave correctly under π_{bexp} when checked directly, since a shared mandatory element functions as a sunk cost that does not disturb the relative ranking of the discretionary alternatives built on top of it, but a general proof for arbitrary nesting depth is not given here. The open question Chapter 5 deferred is accordingly narrower after this chapter than before it: not general obligation-support systems, but specifically systems with genuine partial overlap between branches for different obligations.*

Corollary 28.13 (π_{bexp} is budget-safe under (D1)–(D2))**.** *Under a disjoint-branch system, π_{bexp} is a budget-safe policy on V_b for every b.*

Proof. Immediate from the Disjoint-branch sufficiency theorem together with the Greedy completeness criterion: π_{bexp} reaches $\mathfrak{A}(b, \mathcal{O})$ from P_t whenever it is reachable at all, and the same argument applies unchanged starting from any $A \in V_b$ rather than from P_t specifically, since the proof used only the branch structure of A, not that $A = P_t$. □

A second, complementary route to budget-safety does not require the disjoint-branch hypothesis at all, at the cost of requiring foreknowledge of a target.

Remark 28.14 (Design-time collapse as a route to budget-safety)**.** *If a target $A^* \in \mathfrak{A}(b, \mathcal{O})$ is chosen in advance, by a design-time collapse map ν_t of the kind Chapter 4 licensed as a terminal operation, and pruning is thereafter restricted to cones disjoint from A^*, then contraction accessibility from Chapter 5 already guarantees a safe path to A^*, and the restricted policy is trivially budget-safe on the sub-basin of states from which A^* is reachable. This is closer to what Part V's*

Apollo case actually required than either greedy policy in this chapter: the priority table did not discover an admissible target during the descent, it re-admitted jobs into a bookkeeping structure whose target sufficiency had already been secured by the protected guidance state before the overflow occurred.

The general open question beneath both routes, identifying structural conditions under which budget-safety can be verified locally, from a survivor set's own immediate structure, without first solving for a target admissible set or assuming full branch disjointness, is close in spirit to questions already answered for other combinatorial selection problems by the theory of greedoids and related accessible set systems, where a language of exchange and accessibility axioms exists for exactly this kind of question. Whether that theory transfers cleanly to obligation-support systems, where the relevant structure is a preserved predicate over branch families rather than an independence system over a single matroid, is not settled here.

Chapter 29

Obligation-Set Revision

Every result through Part VII holds the obligation set $\mathcal{O}$ fixed within the episode being analyzed. Chapter 25 named the case where this assumption fails, new instructions revising what a system owes in the middle of a single conversation, as a moving-target problem Apollo's stable descent obligations did not exhibit, without building the machinery to analyze it. This chapter builds that machinery.

Definition 29.1 (Obligation trajectory and revision event)**.** *An obligation trajectory is a sequence* $\{\mathcal{O}_t\}_{t\geq 0}$*. A revision event at* t *is a transition* $\mathcal{O}_t \to \mathcal{O}_{t+1}$ *with* $\mathcal{O}_t \neq \mathcal{O}_{t+1}$*.*

A revision can retire a satisfied obligation, admit a new one, or drop an unsatisfied one, and these are not equally innocent. Parallel to Chapter 6's task-level dispositions, an obligation-level disposition vocabulary is needed to keep them distinct.

Definition 29.2 (Obligation-level dispositions)**.** *For* $o \in \mathcal{O}_t \setminus \mathcal{O}_{t+1}$*: o is* Retire-d *when it was already discharged,* $I_o = 1$*, by time* t*. o is* Waive-d *when it was not yet discharged but is explicitly dropped with a recorded witness stating the authorization for dropping it. o is silently dropped when neither holds, an obligation-level analogue of* Forget*: an outside observer cannot then distinguish a legitimate lapse of relevance from an unaccountable abandonment of a standing commitment.*

Definition 29.3 (Admissible revision)**.** $\mathcal{O}_{t+1}$ *is an admissible revision of* $\mathcal{O}_t$*, written* $\mathcal{O}_t \triangleright \mathcal{O}_{t+1}$*, when every* $o \in \mathcal{O}_t \setminus \mathcal{O}_{t+1}$ *is either* Retire-d *or* Waive-d*.*

Obligation growth interacts with the resource-side theory of Part IV in a way worth stating precisely, dual to the monotonicity result already given for resources.

Proposition 29.4 (Dual monotonicity)**.** *If* $\mathcal{O} \subseteq \mathcal{O}'$*, then* $\mathfrak{A}(b, \mathcal{O}') \subseteq \mathfrak{A}(b, \mathcal{O})$*.*

Proof. $I_{\mathcal{O}'}(A) = 1$ requires satisfying every obligation in $\mathcal{O}'$, hence every obligation in $\mathcal{O} \subseteq \mathcal{O}'$, so $I_{\mathcal{O}'}(A) = 1 \Rightarrow I_{\mathcal{O}}(A) = 1$; combined with the shared resource constraint $C(A) \preceq b$, this gives the inclusion. □

More resources can only enlarge the admissible family; more obligations can only shrink it. An obligation revision that admits new commitments can therefore force a new round of contraction with b_t held fixed, a phenomenon Part IV's resource-only framing has no vocabulary for.

The obligation-level dispositions above govern whether a revision is accountable. A separate question, at the level of the current survivor set rather than the obligation set, is whether the system's actual task configuration at the moment of revision remains workable under the new regime, or needs an explicit transition.

Definition 29.5 (Static revision compatibility)**.** *$\mathcal{O}_t$ and $\mathcal{O}_{t+1}$ are statically compatible at A_t, the currently active survivor set, when*

$$\mathfrak{A}(b_t, \mathcal{O}_t) \cap \mathfrak{A}(b_t, \mathcal{O}_{t+1}) \neq \varnothing.$$

Static compatibility means some survivor set satisfies both the old and the new regime at once, so the revision can be absorbed without changing what is currently running. Its absence does not make the revision impossible; it means the system needs a bridge.

Definition 29.6 (Transitional compatibility)**.** *$\mathcal{O}_t \to \mathcal{O}_{t+1}$ is transitionally compatible when there exists a continuation γ and a switching time τ such that*

$$I_{\mathcal{O}_t}(\gamma(s)) = 1 \ (s < \tau), \qquad I_{\mathcal{O}_{t+1}}(\gamma(s)) = 1 \ (s \geq \tau).$$

Transitional compatibility is strictly weaker than static compatibility, since it only requires the old regime to hold up to a declared handoff and the new regime from that point on, not a single survivor set satisfying both simultaneously; every statically compatible revision is transitionally compatible with $\tau = t$, but the converse can fail. This gives the AI-context chapter's criticism a sharper edge than Chapter 25 could state on its own: context systems are usually criticized for losing inputs, but the deeper failure is revising the operative obligation set without producing a witness that a revision, static or transitional, occurred at all. Treating $\mathcal{O}_t \to \mathcal{O}_{t+1}$ as ordinary forgetting, in Chapter 6's sense, misdiagnoses the event; the missing operation is not storage but authorized obligation revision, and the disposition vocabulary above exists to name it correctly.

The viability apparatus of Chapter 3 needs a corresponding generalization, since asking whether a continuation is admissible under the *current* $\mathcal{O}_t$ says nothing about whether it survives the next revision.

Definition 29.7 (Revision-robust viability)**.** *For a declared revision class $\mathcal{R}$, a set of obligation trajectories the system is built to tolerate, the revision-robust viability kernel is*

$$K_{\mathcal{O}(\cdot),\mathcal{R}} = \{\, x_t : \forall\, \{\mathcal{O}_s\}_{s \geq t} \in \mathcal{R},\ \exists\, \gamma \in \Gamma(x_t, b_t, \mathcal{O}_s \text{ at each } s) \,\},$$

a state from which some continuation remains available against every obligation trajectory the class $\mathcal{R}$ admits, not merely against $\mathcal{O}_t$ held fixed.

This is the same move Chapter 19 made for resource disturbance classes, applied to obligation trajectories instead. Historical sufficiency needs the same generalization, and here the direction of time matters in a way it did not before.

Definition 29.8 (Revision-sufficient history)**.** *Retained history $\widehat{h}_t$ is revision-sufficient across $\mathcal{O}_t \to \mathcal{O}_{t+1}$ when $\widehat{h}_t \sim_{\mathcal{O}_t} h_t$ and, in addition, $\mathrm{Cont}_{\mathcal{O}_{t+1}}(\widehat{h}_t) \neq \varnothing$: the retained history does not itself foreclose satisfying whatever is newly admitted.*

The second condition is ordinarily easy to satisfy, since a newly admitted obligation usually only constrains the future. It becomes hard exactly when a new obligation's own dependency set reaches backward into what has already happened, a revised instruction that a fact from earlier in the episode must have been preserved. This produces a failure mode with no counterpart anywhere else in this monograph.

Proposition 29.9 (Retroactive insufficiency)**.** *No amount of per-step OPCC compliance under a moving obligation trajectory guarantees revision-sufficiency for a future obligation whose dependency set reaches into history contracted before that obligation was admitted. An earlier contraction can be fully admissible relative to $\mathcal{O}_t$ and still discard exactly what a later $\mathcal{O}_{t+1}$ requires, since $\mathcal{O}_t$ cannot be required to anticipate $\mathcal{O}_{t+1}$.*

This sharpens Chapter 25's diagnosis rather than merely restating it. That chapter identified one failure mode, an obligation set that is never declared at all. This proposition identifies a second, distinct failure mode available even to a system that declares $\mathcal{O}_t$ faithfully at every step: declaring the obligation set correctly at each instant does not, by itself, protect against a later revision demanding something an earlier, contemporaneously correct contraction had no reason to keep.

Remark 29.10. *A positive result is available, at the cost of a stated assumption about the revision class rather than a promise that holds unconditionally. Suppose $\mathcal{R}$ only ever admits new obligations whose dependency set lies within the last k steps of history. A system that retains a lookback margin of at least k steps of raw, obligation-unfiltered material at every point, beyond whatever $\widehat{h}_t$ Chapter 9's ordinary sufficiency already requires, satisfies revision-sufficiency for every transition $\mathcal{R}$ admits, since the newly admitted obligation's dependency set is guaranteed to fall inside the retained margin. This is the same discipline Part V credited to Apollo's engineers, applied to the obligation dimension instead of the resource dimension: safety under a revision that has not yet happened is bought by what was already protected before it arrived, not by cleverness exercised at the moment it does.*

Bibliography

[1] Jean-Pierre Aubin. *Viability Theory*. Birkhäuser, Boston, 1991.

[2] Jean-Pierre Aubin, Alexandre M. Bayen, and Patrick Saint-Pierre. *Viability Theory: New Directions*. Second edition, Springer, Berlin, 2011. https://doi.org/10.1007/978-3-642-16684-6.

[3] Mitio Nagumo. Über die Lage der Integralkurven gewöhnlicher Differentialgleichungen. *Proceedings of the Physico-Mathematical Society of Japan*, 24:551–559, 1942.

[4] Franco Blanchini. Set invariance in control. *Automatica*, 35(11):1747–1767, 1999. https://doi.org/10.1016/S0005-1098(99)00113-2.

[5] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Fourth edition, Athena Scientific, Belmont, Massachusetts, 2017.

[6] Tristan Needham. *Visual Complex Analysis*. Clarendon Press, Oxford University Press, Oxford, 1997.

[7] Garrett Birkhoff. *Lattice Theory*. Third edition, American Mathematical Society Colloquium Publications, volume 25, American Mathematical Society, Providence, Rhode Island, 1967.

[8] G. H. Hardy. *A Mathematician's Apology*. Cambridge University Press, Cambridge, 1940.

[9] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Second edition, Cambridge University Press, Cambridge, 2002.

[10] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973. https://doi.org/10.1145/321738.321743.

[11] John P. Lehoczky, Lui Sha, and Ye Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 166–171, 1989. https://doi.org/10.1109/REAL.1989.63567.

[12] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Third edition, Springer, New York, 2011. https://doi.org/10.1007/978-1-4614-0676-1.

[13] Eugene L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2:75–90, 1978. https://doi.org/10.1016/S0167-5060(08)70342-8.

[14] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985. https://doi.org/10.1145/214451.214456.

[15] Elmootazbellah N. Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, 2002. https://doi.org/10.1145/568522.568525.

[16] John W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, 1974. https://doi.org/10.1145/361147.361115.

[17] John T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312, 2006. https://doi.org/10.1016/j.future.2004.11.016.

[18] Brian Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, SE-1(2):220–232, 1975. https://doi.org/10.1109/TSE.1975.6312842.

[19] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978. https://doi.org/10.1145/359545.359563.

[20] Jim Gray. Notes on database operating systems. In R. Bayer, R. M. Graham, and G. Seegmüller, editors, *Operating Systems: An Advanced Course*, pages 393–481. Springer, Berlin, 1978. https://doi.org/10.1007/3-540-08755-9_9.

[21] Hector Garcia-Molina and Kenneth Salem. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pages 249–259, 1987. https://doi.org/10.1145/38713.38742.

[22] Pat Helland. Life beyond distributed transactions: An apostate's opinion. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research*, pages 132–141, 2007.

[23] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013. https://doi.org/10.1145/2408776.2408794.

[24] Martin Kleppmann. *Designing Data-Intensive Applications*. O'Reilly Media, Sebastopol, California, 2017.

[25] David G. Hoag. The history of Apollo onboard guidance, navigation, and control. Charles Stark Draper Laboratory, Cambridge, Massachusetts, 1976.

[26] Eldon C. Hall. *Journey to the Moon: The History of the Apollo Guidance Computer*. American Institute of Aeronautics and Astronautics, Reston, Virginia, 1996.

[27] Margaret H. Hamilton. Computer got loaded. Letter to the editor, *Datamation*, March 1, 1971.

[28] Margaret H. Hamilton and Saydean Zeldin. Higher order software—A methodology for defining software. *IEEE Transactions on Software Engineering*, SE-2(1):9–32, 1976. https://doi.org/10.1109/TSE.1976.233798.

[29] Margaret H. Hamilton and William R. Hackler. Universal Systems Language: Lessons learned from Apollo. *Computer*, 41(12):34–43, 2008. https://doi.org/10.1109/MC.2008.541.

[30] National Aeronautics and Space Administration. Apollo 11 Lunar Surface Journal: Program alarms. In *Apollo 11 Lunar Surface Journal*. https://www.nasa.gov/history/alsj/a11/a11.1201-pa.html.

[31] National Aeronautics and Space Administration. *Apollo 11 Mission Report*. Technical Report MSC-00171, Manned Spacecraft Center, Houston, Texas, November 1969.

[32] National Aeronautics and Space Administration. *Apollo Experience Report: Guidance and Control Systems*. NASA Technical Note, Washington, D.C., 1972.

[33] David A. Mindell. *Digital Apollo: Human and Machine in Spaceflight*. MIT Press, Cambridge, Massachusetts, 2008.

[34] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[35] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. https://doi.org/10.1162/tacl_a_00638.

[36] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long-context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 3119–3137, 2024. https://doi.org/10.18653/v1/2024.acl-long.172.

[37] Michael Lipsky. *Street-Level Bureaucracy: Dilemmas of the Individual in Public Services*. Russell Sage Foundation, New York, 1980.

[38] Institute of Medicine. *Guidance for Establishing Crisis Standards of Care for Use in Disaster Situations: A Letter Report*. National Academies Press, Washington, D.C., 2009. https://doi.org/10.17226/12749.

[39] Institute of Medicine. *Crisis Standards of Care: A Systems Framework for Catastrophic Disaster Response*. National Academies Press, Washington, D.C., 2012. https://doi.org/10.17226/13351.