physiome: An Admissibility- and Repair-Driven Architecture for Whole-Body Physiological Simulation

Flyxion
Independent Researcher

August 2026

Abstract

We describe the design and initial implementation of physiome, a whole-body physiological simulator written in Rust with a pure functional core. Rather than encoding homeostasis as a fixed system of coupled differential equations, physiome treats each physiological variable as governed by an admissibility boundary, a range of values the rest of the organism will accept as valid, and models correction as the dispatch of repair operators against detected violations of that boundary. Subsystems are advanced on independent clocks rather than a single global timestep, reflecting the genuine multiplicity of physiological timescales. We present the current architecture and its design rationale, a formal treatment of admissibility and repair dispatch as a rewrite system with an accompanying complexity analysis, a comparison against HumMod's published schema grounded directly in that schema's own specification of its numerical integration and calculation-order model, an eleven-subsystem implementation, a worked infection scenario exercising a cross-subsystem cytokine-to-fever coupling, three substantive bugs surfaced during development of that scenario, a three-level roadmap toward a fuller organ-system specification, and a closing discussion arguing, with appropriate restraint, that the engine's detect-rank-dispatch-reassess loop is a reasonable description of physiological regulation generally rather than an artifact of this codebase. The central claim is architectural rather than empirical: separating a domain-independent repair engine from a declarative library of subsystem-specific operators and boundaries keeps the model's growth additive, at the cost of requiring careful attention to how repair operators are dispatched and how subsystems are scheduled relative to one another, two areas in which our own early implementation contained real errors.

1 Introduction and Motivation

Whole-body physiological simulators in the tradition of HumMod represent the human organism as a large network of coupled differential equations, each variable's rate of change written as an explicit function of the others. This approach has produced simulators of considerable clinical fidelity, but it ties the correctness of the model to the correctness of a very large, tightly coupled system of equations, and it offers no natural place to express the idea that a variable can simply be *out of range* independent of the particular equation that produced that state. physiome starts from a different primitive. Instead of asking what differential equation

governs a variable's evolution, it asks what range of values the rest of the organism is willing to accept from that variable, and treats departure from that range as the trigger for a corrective response, rather than deriving the corrective response from the same equation that produced the departure. Stated as compactly as possible, where most physiological simulators encode Mechanism $\rightarrow$ Behaviour, evaluating every mechanism continuously and letting behavior fall out of the aggregate, physiome encodes Boundary $\rightarrow$ Violation $\rightarrow$ Repair $\rightarrow$ Behaviour: mechanisms still exist, as the transitions inside repair operators, but they are invoked by regulation rather than continuously evaluated, and behavior is whatever trajectory that invocation process happens to produce. This is the single distinction the rest of the paper elaborates, and it is the one most worth holding onto if the details that follow start to blur together.

This follows a broader set of foundational commitments developed across the author's other work: that repair is prior to correctness, in the sense that a system's ability to recognize and correct an inadmissible state does not depend on first deriving a complete predictive account of how it arrived there, and that admissibility is prior to prediction, in the sense that knowing whether a value is acceptable is a weaker and more robust condition than knowing what value should occur next. Applied to physiology, this suggests an architecture in which the engine's only universal knowledge is how to detect a boundary violation and how to dispatch a declared repair operator against it, while everything domain-specific, every constraint range, every corrective mechanism, every subsystem's internal clock, is supplied as data rather than hardcoded into the engine.

A secondary motivation is architectural hygiene under growth. A simulator intended eventually to cover dozens of interacting organ systems needs a mechanism by which adding the twelfth system does not require modifying how the first eleven are dispatched. We return to this claim in Section 4, state it as a structural proposition in Section 6, and test it empirically in Section 7, where four new subsystems were added to an existing two-subsystem sketch without any change to the shared engine.

The remainder of the paper is organized as follows. Section 2 reviews the coupled-equation tradition this work departs from. Section 3 states the departure precisely, including a comparison grounded directly in HumMod's own published schema. Sections 4 and 5 describe the architecture and the reasoning behind its major design choices. Section 6 formalizes admissibility and repair dispatch as a rewrite system, with accompanying termination, convergence, and complexity results. Sections 7 and 8 present the current eleven-subsystem implementation and a worked infection scenario, including three defects the scenario surfaced. Section 9 lays out a three-level roadmap toward a fuller specification. Section 10 discusses the broader interpretation of the architecture, and Section 11 concludes.

2 Related Work

HumMod, developed at the University of Mississippi Medical Center, is the most complete public whole-body physiological model, comprising several thousand interdependent variables spanning cardiovascular, renal, endocrine, and other systems, integrated as a large system of ordinary differential equations. Its strength is clinical detail; its cost is that every new interaction must be expressed as an explicit equation coupling two or more existing variables, so the model's internal complexity grows combinatorially with its coverage. Classical control-theoretic models of individual systems, such as Guyton's circulatory model, share this prop-

erty at smaller scale. physiome does not attempt to match the quantitative fidelity of either; its contribution is architectural, an attempt to separate the part of the model that is genuinely domain-independent (the detection of an inadmissible state and the dispatch of a response) from the part that is irreducibly physiological (what the admissible ranges are and what the repair mechanisms do), so that the two can be developed and reasoned about separately.

3 Why physiome Is Not an ODE Solver

The distinction between physiome's architecture and the differential-equation tradition it departs from is not merely stylistic, and it is worth stating precisely rather than leaving it implicit in the description above. Table 1 summarizes the contrast along five axes that recur throughout the rest of this paper.

Coupled differential equations	physiome
Evolution-first: every variable has a derivative at every instant	Admissibility-first: a variable is simply held until its boundary is violated
The derivative determines the future state	Detected violations determine which repairs, if any, fire
One large, mutually coupled equation system	A library of small, independently specified repair operators
A single global timestep integrates every variable together	Each subsystem advances on its own clock, chosen by the scheduler
Every variable evolves continuously, whether or not anything physiologically interesting is happening to it	Variables evolve only through subsystem continuations or repair operators

Table 1: physiome's admissibility-first architecture contrasted with the evolution-first architecture of a coupled ODE system such as HumMod.

The practical consequence of the right-hand column is the one already argued informally in Section 1: because the engine's obligations (detect a violation, dispatch a matching operator) do not reference any particular subsystem, adding a twelfth subsystem is additive at the engine level in a way that adding a new coupled variable to an ODE system is not, since the latter generically requires touching every existing equation that the new variable interacts with. The cost of this architecture, made concrete in Section 6 and again in the defects recorded in Section 8, is that admissibility-first correctness properties (does every violation eventually get addressed, do two operators responding to the same violation compose sensibly) must be established about the dispatch mechanism itself, rather than inherited for free from the well-studied numerical properties of ODE integration.

3.1 Comparison with the HumMod Schema

The comparison above has so far been with the differential-equation tradition in general; it is worth making concrete against HumMod's own published schema specifically [3], since the schema states its composition units and calculation model explicitly. HumMod's `structure` element is a container for `variables`, `equations`, `functions`, and `definitions`, and the schema is direct that this is the model: a `structure` declares named variables (of which the schema lists eight types), declares differential and implicit algebraic equations over them, and specifies, in `definitions`, the named `block` elements of math that compute derivative and intermediate values, invoked by `call` elements that together encode the calculation sequence. physiome's units of composition are different in kind, not merely in name: an `AdmissibilityBoundary`, a `RepairOp`, and a `Continuation` say nothing about how a variable's value is computed moment to moment, only what range of values are acceptable and what the organism does when they are not.

This difference is sharpest around numerical integration. HumMod's schema devotes substantial specification to it: differential equations are declared via `diffeq`, `backwardeuler`, `stablediffeq`, `delay`, and `stabledelay` elements, the schema explicitly names backward Euler ("more stable than the standard 1st-order Euler") as the underlying method for several of these, and each carries its own `errorlim`, an integration error limit controlling numerical accuracy. Solving the model additionally requires a `solutionint` (solution interval, the step over which the equation solver advances) and, per the schema, a `displayint` and, historically, a `storageint`, with the schema stating directly that the equation solver's own internal step is less than or equal to the storage interval; time, in other words, belongs to the numerical integrator, and the solution, display, and storage intervals are all specified relative to it. physiome has no analogue of any of this, not because numerical integration was simplified or approximated, but because the engine never performs it: no subsystem in the current implementation solves a differential equation, controls an integration error, or specifies a solver step size, since `Continuation::advance` computes a subsystem's next state directly from an elapsed duration rather than integrating a rate. Time belongs to the scheduler in `step_until`, not to a numerical method; a subsystem requests its own next advance via `interval`, and the scheduler grants it, which is a different ontology of time than one in which every variable's advance is a side effect of a shared solver's step.

A further consequence is what we term correctness by locality. In HumMod's schema, a `block`'s correctness is not fully separable from the rest of the model, since a derivative's math block is, per the schema, called at a particular point in a calculation sequence that must be gotten right for the coupled system to integrate correctly; verifying one block in isolation does not verify its behavior inside the full call graph. A physiome `RepairOp`, by contrast, only has to satisfy a three-part local contract, does its predicate correctly identify the violations it is meant for, does its transition move the state toward admissibility, does it terminate (guaranteed unconditionally by Proposition 1), and this contract can be checked, and was checked in Section 8, one operator at a time, without reasoning about the other twelve. This is the same property already stated as an engine-level design goal in Section 1 and Section 5, restated here as the specific thing HumMod's own schema shows it is not obliged to provide, since a coupled equation system's correctness is a property of the whole system, not of any one equation.

3.2 Placement Among Simulation Paradigms

HumMod is not the only point of comparison worth naming briefly. Agent-based physiological models represent large populations of individually simulated entities (cells, immune agents) whose aggregate behavior produces the phenomena of interest; physiome instead models aggregate physiological variables directly, with repair operators standing in for the mechanism rather than emerging from many simulated individuals, a different level of abstraction rather than a competing one. The scheduler in `step_until` is, in its actual operational structure, closer to a discrete-event simulator than to either of the above: state advances only at discrete, subsystem-chosen event times, and a repair fires because a boundary was crossed at one of those events, not because a fixed timestep elapsed, which is the defining property of discrete-event simulation as usually understood. Finally, the combination of continuous-seeming advance between events (a `Continuation::advance` call) with discrete intervention only when a boundary is crossed (a `RepairOp` firing) places this architecture nearer the hybrid-systems tradition, hybrid automata and switched systems, than the classical ODE tradition it is contrasted with above; the hybrid model sketched as a mathematical roadmap item in Section 9, ordinary continuous dynamics within an admissible region and discrete repair only at its boundary, is in fact the natural completion of this placement rather than a departure from it.

4 Architecture

The engine consists of two small, domain-independent modules. The constraint module defines a `Constraint` as a lower and upper admissible bound, together with a severity function that returns zero when a value is admissible and otherwise a magnitude proportional to how far outside the bound the value sits, normalized by the width of the admissible interval. This normalization is what allows the engine to compare a five-millimeter-of-mercury deviation in blood pressure against a five-milliequivalent-per-liter deviation in plasma sodium on a common scale: both are expressed as a fraction of their own admissible width, so the engine is effectively asking which variable is furthest outside its own contract, not which has the largest raw numerical error.

Every subsystem implements an `ObservableBoundary` trait, exposing its current observable variables by name, its admissible boundary, and its own name as a string. This is the single piece of indirection that lets the engine iterate over an arbitrary, growing collection of subsystems without containing any subsystem-specific logic: a function that wants to check the whole organism for violations calls the same `detect_violations_for` generic function once per subsystem and concatenates the results.

The repair module defines a `RepairOp` as data: a name, a predicate over a detected violation indicating whether this operator is a candidate response, and a pure function from the current state and the violation to a new state. The dispatch function, `step`, takes the current state, a list of available operators, and a list of already-detected violations, and repeatedly selects the highest-severity remaining violation and applies every operator whose predicate matches it, not merely the first such operator, before removing that variable from consideration and continuing until either the list is exhausted or an iteration budget is reached. The decision to apply every matching operator rather than only the first is a correction made during development, discussed in Section 8; it reflects the physiological fact that a single derangement routinely

provokes more than one simultaneous compensatory response.

Because physiological subsystems do not share a clock, a `Continuation` trait is defined separately from the repair machinery. Each subsystem declares an `interval`, the number of seconds until it next wants to be advanced, and an `advance` function describing how its own state changes given an elapsed duration and a set of external inputs (exercise intensity, meal glucose load, ambient temperature, pathogen load, hemorrhage rate). A scheduler, `step_until`, maintains one next-fire time per subsystem, always advances whichever is soonest, and reruns the repair dispatch after every single advance, regardless of which subsystem produced it. This scheduler-owned bookkeeping, rather than a per-call recomputation of next-fire time relative to global simulation time, is itself the product of a bug fix described in Section 8.

The whole-organism state, `PhysiologicalState`, is a struct of structs, one block of fields per subsystem, plus a single scalar simulation time. It is immutable by convention: every transition, whether a repair operator or a continuation's advance function, takes an immutable reference to the current state and returns a new one. Effects external to the pure core, principally the exogenous inputs and any perturbation source, are threaded explicitly as function arguments rather than reached for ambiently, so that a full simulation run is reproducible from its initial state, its input sequence, and nothing else.

Cross-subsystem coupling is deliberately restricted to one channel: a repair operator belonging to one subsystem may read fields from any other subsystem's block of the shared state, and may write to any subsystem's block, but subsystem modules never call each other's functions directly. Two concrete instances of this appear in the current implementation. The hematologic coagulation response reads the hepatic subsystem's `clotting_factors` field, so that impaired synthetic liver function genuinely constrains how much a coagulation repair operator can improve the coagulation index in a single application. The immune subsystem's fever response reads its own `cytokine_level` violation as the trigger, but writes to the thermal subsystem's `core_temp` field as the effect, with the thermal subsystem's own thermoregulation operator in turn reading the immune subsystem's cytokine level to compute an effective, temporarily elevated set point rather than correcting fever back to the unelevated baseline.

5 Design Rationale

The description above states what the architecture does; this section states why, since several of the choices above are easy to describe but easy to under-motivate, and the motivation is what would let a reader judge whether the choices generalize beyond this particular codebase.

5.1 Regulation Instead of Prediction

An ODE model answers, for every variable at every instant, what will this value be next. physiome's engine never answers that question; it only ever answers is this value currently acceptable, and if not, what corrective action is available. This is a deliberate narrowing of the question asked, not an oversight, and it tracks a real distinction in what physiological regulation actually does: a baroreceptor does not compute a target mean arterial pressure five seconds hence, it fires or does not fire based on whether current pressure is within range. The same narrowing separates two things that a coupled differential equation system necessarily fuses into one computation: dynamics, how does the state evolve, and regulation, what is allowed to persist. An ODE integrator answers both at once, every timestep, for every variable,

whether or not anything physiologically interesting is happening. physiome answers them with two different mechanisms operating on two different triggers, a `Continuation` advancing state on its own schedule regardless of admissibility, and a `RepairOp` firing only when a boundary is actually crossed. The mechanism and policy split introduced in Section 4, engine knows detect-rank-dispatch, subsystem content knows ranges and mechanisms, is this same distinction restated at the level of code organization rather than conceptual vocabulary.

5.2 Purity and Replayability

Every `RepairOp::apply` and `Continuation::advance` in the current implementation takes the state by immutable reference and returns a new state by value, with no interior mutability anywhere in the eleven subsystem modules (Section 6 verifies this by inspection rather than by convention alone). The practical payoff is not aesthetic. A run is a pure function of its initial state, its registered operators and continuations, and its input sequence, so any observed trajectory, including a defective one, is exactly reproducible by re-supplying the same arguments; this is what made it possible to isolate each of the three defects in Section 8 to a specific operator or scheduling decision rather than to a vaguely reproducible pattern. A mutable, object-oriented simulator, where a variable's current value is whatever some prior call last set it to, does not offer this for free; reconstructing which of possibly many mutating call sites produced a given state typically requires either careful logging discipline or a debugger session, neither of which is needed here, since the state itself is the complete record.

5.3 Independent Clocks as a Physiological Fact, Not an Implementation Convenience

A fixed global timestep, the choice a coupled ODE system essentially must make so that every equation can be integrated together, forces cardiovascular reflexes (seconds), hormonal regulation (minutes to hours), and hematopoiesis (days) onto one clock, typically the fastest one, since a global timestep must be no coarser than the fastest dynamics or the integration becomes unstable. An adaptive global timestep improves this but still shares one clock across the whole system at any given instant. Asynchronous per-subsystem clocks, the `Continuation` trait's actual choice, avoid this by construction: a subsystem's own `interval` is a direct assertion about the physiological timescale it represents, independently adjustable, and Table 2 shows intervals ranging from one second (cardiovascular) to fifteen minutes (renal) coexisting in the same simulation without either being distorted by the other's cadence. Section 8's first defect is a cautionary note on this design rather than a rebuttal of it: getting independent clocks right required scheduler-owned bookkeeping that a naive implementation gets subtly wrong, but the underlying motivation, that forcing physiological subsystems onto one clock is a numerical convenience rather than a fact about the organism, still holds.

5.4 Declarative Repair Operators and Explicit Coupling

The alternative to a `RepairOp` as data would be an `if mean_arterial_pressure < 70 then ...` branch written directly inside the dispatch engine, one such branch per variable per condition. This is the plugin-architecture comparison worth making explicit: just as a plugin host knows only how to discover and invoke a plugin's declared interface, never the plugin's

internals, `step` knows only how to detect a violation and invoke a matching operator's declared predicate and transition, never anything about baroreflexes or urea cycles specifically. This is also why cross-subsystem coupling is restricted to reads and writes through the shared `PhysiologicalState` rather than direct calls between subsystem modules: a call from, say, the hematologic module directly into the hepatic module would be an undeclared dependency invisible to anyone auditing the operator registry, whereas a read of `state.hepatic.clotting_factors` inside a hematologic operator's `apply` function is visible in exactly the same place the operator's other logic is, and is the only kind of cross-subsystem dependency the architecture permits.

5.5 Why Rust

Several of the properties above are stated as design intentions elsewhere in this paper but are, in the current implementation, also structurally enforced by the host language rather than merely followed by convention. The absence of interior mutability invoked in Section 6's purity invariant is checkable by grep for `Cell`, `RefCell`, or `unsafe`, precisely because Rust makes uncontrolled mutation, not merely undisciplined mutation, a compile error; a Python or JavaScript implementation of the same architecture could describe the same purity discipline but could not have it enforced by the compiler. Ownership and borrowing similarly make the shared-state-only coupling channel of Section 4 closer to a language-level guarantee than a code-review convention: a repair operator's `apply` function receives `&PhysiologicalState`, an immutable borrow, so it is not merely discouraged from mutating another subsystem's fields in place, it is unable to. None of this is a claim that Rust is required for this architecture, only that several properties this paper otherwise has to state as intentions and then verify by inspection are, in this implementation, also reinforced by the type system, which is one reason the invariants in Section 6 could be checked directly against the code rather than merely asserted.

What follows is a formal restatement of the architecture just described. Section 6 is intended to make the informal claims above checkable rather than to introduce new mechanism; every object it defines already exists in the Rust implementation described above, and every proposition it states is checked directly against that implementation's actual code, including one place where the informal description above (the fever response being "simultaneously dispatched" alongside immune resolution) needs a more careful statement once written down precisely.

6 A Formal Framework for Admissibility and Repair

6.1 Admissibility as a Relation

Let s range over the global state space $\mathcal{S}$ of a `PhysiologicalState`, and let K be the finite set of named observable variables exposed across all subsystems (for example `cardiovascular.mean_arterial_pres` or `immune.cytokine_level`). Each $k \in K$ has an observation function $o_k : \mathcal{S} \to R$, implemented as a lookup into the `HashMap` returned by that variable's subsystem's `observables` method, and an admissible set $\mathcal{B}_k \subseteq R$, currently always a closed interval $[\mathrm{lo}_k, \mathrm{hi}_k]$ returned by that subsystem's `boundary` function. Admissibility of k at state s is the relation $o_k(s) \in \mathcal{B}_k$.

The severity function

$$v_k(s) = \begin{cases} 0, & o_k(s) \in \mathcal{B}_k \\ \dfrac{d\big(o_k(s), \mathcal{B}_k\big)}{|\mathcal{B}_k|}, & \text{otherwise} \end{cases}$$

is exactly `Constraint::severity`, where $d(\cdot, \mathcal{B}_k)$ is distance to the interval and $|\mathcal{B}_k| = \mathrm{hi}_k - \mathrm{lo}_k$ is its width. Dividing by $|\mathcal{B}_k|$ is what makes $v_k(s)$ comparable across variables with unrelated units and unrelated natural scales, and is the property the repair engine actually relies on when it ranks violations against one another. A state s is globally admissible when $v_k(s) = 0$ for every $k \in K$; the violation set is $V(s) = \{\, k \in K : v_k(s) > 0 \,\}$. It is worth treating $\mathcal{B}_k$ itself as the primary object here, on equal footing with an equation of motion rather than as auxiliary bookkeeping around one: a differential equation $\frac{dx}{dt} = f(x)$ specifies how a variable evolves, while $\mathcal{B}_k$ specifies what is allowed to persist, and the architecture in this paper is organized around treating the latter as the thing the engine actually operates on, with evolution relegated to `Continuation` and never consulted by the dispatch mechanism itself.

6.2 Repair Operators as Parameterized State Maps

A repair operator is a pair (p_m, r_m): a predicate p_m over a violation (a pair of a variable name and its current severity) deciding whether the operator is a candidate response, and a transition $r_m : \mathcal{S} \times K \to \mathcal{S}$ describing how the operator rewrites the state when it fires, parameterized by which violation triggered it. Because r_m takes the triggering violation as an argument, it is not, strictly, an endomorphism on $\mathcal{S}$; for a fixed violation k, however, $R_m^{(k)} : \mathcal{S} \to \mathcal{S}$ defined by $R_m^{(k)}(s) = r_m(s, k)$ is an ordinary endomorphism, and it is this fixed-k restriction that composes in the dispatch step below. Writing $R_m(s; k)$ interchangeably with $R_m^{(k)}(s)$, the implementation's dispatch, for a single selected violation k^*, applies every operator whose predicate matches, in the fixed registration order of the operator list, as the sequential composition

$$s' = R_{m_j}\big(\, \cdots R_{m_2}\big(R_{m_1}(s; k^*); k^*\big) \cdots ;\, k^*\big), \qquad \{m_1, \ldots, m_j\} = \{\, m : p_m(k^*) \,\}.$$

This composition is sequential, not simultaneous in the sense of a parallel or commuting update: each operator in the matching set sees the output of the previous one, in registration order. The informal description of the fever scenario in Section 8 as immune resolution and fever response being "simultaneously dispatched" should be read in this precise sense, sequential composition against a shared trigger, not literal simultaneity, and in general

$$R_i\big(R_j(s; k); k\big) \neq R_j\big(R_i(s; k); k\big)$$

whenever the two operators write to overlapping fields; the current implementation happens to avoid this for the fever pair, since `immune_resolution` writes only within the immune block and `fever_response` writes only to `thermal.core_temp`, so the two commute in this specific instance, but nothing in the architecture guarantees commutativity in general, and a future operator pair that both write to the same field would need either an explicit ordering contract or a redesign as a single combined operator.

This non-commutativity is one entry in what is, informally, an algebra of repairs, worth naming even though the current thirteen operators do not yet exercise all of it. An operator is an *identity repair* at (s, k) when $R_m^{(k)}(s) = s$, matched and dispatched but not actually

changing the state; this is not merely hypothetical here, `nervous::autonomic_rebalance`'s predicate matches any violation in the nervous subsystem, but its transition only actually modifies state when the violated variable is `sympathetic_tone` and its value exceeds the upper bound, so a violation of `parasympathetic_tone`, or of `sympathetic_tone` below its lower bound, is dispatched, logged as a `RepairEvent`, and left completely unchanged, an identity repair the log does not visibly distinguish from an effective one. An operator is *idempotent* when $R_m^{(k)}(R_m^{(k)}(s)) = R_m^{(k)}(s)$, firing again produces no further change; none of the current operators are idempotent by design, since each applies a fixed increment or decrement regardless of the resulting value, though an operator that clamped a variable to a boundary rather than nudging it would be. An operator is *absorbing* for k if repeated firing drives v_k to a fixed nonzero floor rather than to zero, which would indicate a modeled physiological limit rather than a defect; `fever_response`'s hard clamp at 41.5°C is close to this in spirit, though it bounds `core_temp` directly rather than bounding $v_{\text{core\_temp}}$ at a nonzero value. We do not attempt an *inverse* repair notion here, an operator that exactly undoes another's effect, since nothing in the current architecture needs one and defining it precisely would require committing to an algebraic structure (a group action on $\mathcal{S}$, at minimum) well beyond what thirteen small, mostly-independent operators currently motivate; this is flagged as a direction rather than developed, consistent with the general principle in this paper of not formalizing further than the implementation warrants.

6.3 Termination and Convergence

Proposition 1 (Single-call boundedness)**.** *For any invocation of `step` with an initial violation list of size v_0 and iteration budget `max_iterations`, the dispatch loop performs at most* $\min(\texttt{max\_iterations}, v_0)$ *rounds and always terminates.*

Proof sketch. Each round either finds no operator matching the current worst violation, in which case the loop breaks immediately, or applies the matching operators and then unconditionally removes that violation's variable from `remaining` via `retain`, regardless of whether the applied operators actually reduced its severity to zero. Since `remaining` strictly shrinks by exactly one element per completed round and the loop is additionally capped at `max_iterations` rounds, termination follows immediately from the loop's structure and requires no assumption about what the operators compute. □

This proposition is intentionally weak: it says nothing about whether the resulting state is more admissible than the one it started from, only that the procedure halts. It is worth stating explicitly because the removal of a variable from `remaining` is unconditional, not contingent on resolution, a design choice that trades a guarantee of eventual admissibility within one call for a guarantee of bounded work per call; a variable that is still in violation after this round's repair attempt simply waits to be re-detected on the next scheduler tick, described next.

Proposition 2 (Convergence under bounded progress)**.** *For a repair action R matching a violation k, define the progress functional $\Delta_k(s) = v_k(s) - v_k(R(s))$, the reduction in severity a single firing achieves. Suppose a variable k is re-evaluated at every scheduler tick (which `all_violations` does, from scratch, every tick), and suppose $\Delta_k(s) \geq \varepsilon_k > 0$ whenever the matched operator fires and $v_k(s) > 0$, and suppose no other process driving k (a `Continuation::advance` or another operator's write) increases v_k by more than ε_k per tick on average. Then k reaches admissibility within a finite number of ticks.*

Proof sketch. Under the stated hypotheses, v_k evaluated at successive ticks where the violation persists forms a sequence bounded below by zero with $\Delta_k \geq \varepsilon_k$ net of any competing drive at each tick the corresponding operator fires; a strictly decreasing sequence of nonnegative reals with a uniform lower bound on its decrements reaches zero, or crosses into the admissible interval, within finitely many steps. Stating the hypothesis as a lower bound on Δ_k rather than as a specific arithmetic form keeps the proposition independent of how a given operator computes its correction: a nonlinear or state-dependent operator satisfies the hypothesis exactly when its progress functional clears ε_k, with no need to restate the proposition if the operator's internals change. This distinguishes it from an operator whose correction is instead proportional to current severity (a fixed fractional reduction), for which $\Delta_k(s) \to 0$ as $v_k(s) \to 0$ and no fixed $\varepsilon_k > 0$ bounds it uniformly, so such an operator converges to admissibility only in the limit and never reaches it exactly. The current implementation's operators mostly satisfy the stronger, finite-time hypothesis as a consequence of a simpler design choice, subtracting a fixed constant per firing rather than a fraction of the current excess: `immune_resolution` does this for each of its three target variables, and so do `baroreflex`, `raas`, and the majority of the other twelve operators. □

Proposition 2 is a sufficient condition, not a property the engine enforces; an operator implemented with a proportional decrement, or an environment input that drives a variable's severity upward faster than the matched operator can reduce it (as happened, transiently, in the second defect recorded in Section 8, before the operator's direction itself was corrected), falls outside its hypotheses, and the engine provides no independent mechanism for detecting that failure other than the persisting violation itself showing up, tick after tick, in the log.

One consequence of the formalization above is worth stating plainly: dispatch is not a mechanism sitting beneath the physiology, it is the computation that produces the physiological trajectory. There is no separate process computing what the organism's state "should" be that `step_until` then merely records; the sequence of states a run produces is exactly the sequence of successive $R_{m_j} \circ \cdots \circ R_{m_1}$ compositions and `Continuation::advance` calls the scheduler happens to choose, so the scheduler's choices are part of the semantics of a run, not an implementation detail sitting outside it. This is why the scheduling defect in Section 8 produced a wrong trajectory rather than merely a slow one: the scheduler was not an optimization concern layered on top of otherwise-correct physiology, it was choosing, incorrectly, which physiology happened at all.

6.4 Complexity

Let n be the number of registered subsystems, m the number of registered repair operators, and v the number of variables in violation at a given tick. Detection (`all_violations`) calls `detect_violations_for` once per subsystem, each doing work proportional to that subsystem's own small, fixed number of observable variables, so detection is $O(n)$. Within a single `step` call, each of up to $\min(\texttt{max\_iterations}, v)$ rounds re-sorts the shrinking `remaining` list from scratch, $O(v \log v)$ in the worst case for the first round, and filters the operator list against the current worst violation, $O(m)$; summed across rounds this is $O(v^2 \log v + vm)$ in the worst case for a single `step` call. This repeated full re-sort is the dominant term for any tick with more than a handful of simultaneous violations, and it is avoidable: since only the identity of the current maximum is needed each round, not a total order over all of `remaining`, replacing the sort with a priority queue (a binary max-heap keyed on severity, with lazy deletion for

the unconditional removal `retain` performs) would reduce this term to $O(v \log v)$ total rather than per round, an optimization the current implementation does not perform. Scheduling in `step_until` currently performs a linear scan over the n next-fire times every tick to find the minimum, $O(n)$ per tick, rather than the $O(\log n)$ achievable with a binary min-heap keyed on next-fire time; this is a genuine, presently unrealized optimization opportunity rather than a claim about the current code, which we flag explicitly rather than allowing the more familiar complexity to be assumed. Because `step_until` re-runs detection and dispatch after every single tick regardless of which subsystem produced it, the total cost of a simulation run over horizon H is dominated by the number of ticks fired by the fastest-clocked subsystem, currently the one-second cardiovascular clock, which is a real scaling concern: a future subsystem modeled at sub-second resolution would force detection and dispatch to run at that resolution across the entire organism state, not merely within the fast subsystem itself, and this cost should be weighed explicitly against the modeling benefit before any such subsystem is added.

6.5 Engine Locality

The additive-growth claim made informally in Section 1 and exercised empirically in Section 7 has a structural counterpart worth stating as its own proposition, since unlike Proposition 2 it requires no assumption about what any operator computes.

Proposition 3 (Engine locality)**.** *Let a subsystem be any type S implementing `ObservableBoundary` and `Continuation`, together with a finite set of `RepairOp` values. Registering S, that is, adding it to the lists of subsystems whose violations are detected and whose continuations and operators are dispatched, requires no modification to the body of `step`, `step_until`, or `detect_violations_for` itself.*

Proof sketch. `detect_violations_for` is generic over any `S: ObservableBoundary` and dispatches through the trait rather than through a match on subsystem identity, so its own body is identical for every S and never needs editing to support a new one; `step` operates only on `ops: &[RepairOp]` and a violation list, never naming a specific operator; `step_until` operates only on `continuations: &[&dyn Continuation]`, dispatching through the trait object rather than a concrete type. None of the three contains a branch, match arm, or field access that names any particular subsystem, so no subsystem-specific code path exists in them for a new registration to fall outside of. □

Proposition 3 is deliberately narrower than the informal claim it formalizes, and the gap between the two is worth stating rather than eliding. Registering a new subsystem does require touching code outside the subsystem's own module in exactly two places: adding a field to the `PhysiologicalState` struct and a line to `all_violations` in `state.rs`, and adding a line each to `all_repair_ops` and `all_continuations` in `lib.rs`. Proposition 3 is a claim about the dispatch algorithm specifically, `step`, `step_until`, and `detect_violations_for`, not about every line of code in the crate; the registration glue it excludes is bounded, one field and a handful of one-line insertions per subsystem, and does not grow with the number of subsystems already registered, which is the property that actually matters for the additive-growth claim and is what Section 7's four-subsystem addition, and Section 9's planned further ones, exercise in practice.

6.6 The Repair Graph

The registered repair operators induce a directed graph $G = (K, E)$ on the named observable variables, with edge set

$$E = \{\, (k_i, k_j) : \exists\, R_m \text{ dispatched by a violation of } k_i \text{ that writes to } k_j \,\},$$

including the self-loop (k, k) when an operator corrects the same variable that triggered it, the common case. The current thirteen operators instantiate one genuine cross-subsystem edge of this kind, `immune.cytokine_level` → `thermal.core_temp` via `fever_response`, alongside a read-only dependency (not a graph edge, since it does not write) of `hematologic.coagulation_index`'s repair on `hepatic.clotting_factors`. A canonical example worth naming even though it is not yet implemented is the physiological renin-angiotensin-aldosterone loop, MAP → aldosterone → sodium → plasma volume → MAP, a directed cycle rather than a simple chain; the current `renal::raas` operator is named after this pathway but does not yet instantiate it as a graph edge, since its implementation writes only to fields within the renal subsystem's own block and never reads or writes `endocrine.aldosterone` or `cardiovascular.mean_arterial_pressure`. Building out the true cross-subsystem RAAS cycle, and auditing the rest of the operator library for similarly aspirationally named but not-yet-coupled operators, is listed as roadmap work in Section 9. Once a sufficient fraction of the intended physiological couplings are actually instantiated as graph edges, G becomes a natural object to analyze directly, for cycles (feedback loops, which may or may not be desirable depending on whether they are stabilizing), for variables with unusually high in-degree (bottleneck targets many operators write to), and for weakly connected components (subsystems that are, as modeled, causally isolated from the rest of the organism, which may be a modeling gap rather than a deliberate simplification). Classical physiology already draws this graph informally, as a pathway diagram, arrows from one regulated quantity to the next; G is not a documentation aid alongside that tradition so much as its computational counterpart, the same diagram made executable, since an edge in G is not asserted but derived directly from which operators are registered and what they write.

6.7 Subsystem-Level Coupling

Section 4 describes cross-subsystem coupling operationally: an operator may read another subsystem's fields and write to any subsystem's block, but subsystems never call each other directly. This can be stated as a coarsening of the repair graph G above rather than as a separate object. Let $\pi : K \to I$ map each named variable to the subsystem that owns it, where I is the finite set of registered subsystems, and, for subsystem $i \in I$, let F_i be the set of *all* struct fields belonging to i, not only its observable set $\pi^{-1}(i) \subseteq K$; the distinction matters because at least one instantiated coupling in the current implementation, hematologic's coagulation response reading `hepatic.clotting_factors`, targets a field that is not part of any `ObservableBoundary`, so a formalization restricted to observable sets would miss it. Let $W_i \subseteq F_i \cup \bigcup_{j \neq i} F_j$ be the fields any of i's repair operators write, restricted by Section 4's discipline to lie in some subsystem's own field set, and R_i the fields any of i's operators read. The subsystem-level dependency graph has $i \to j$ whenever $W_i \cap F_j \neq$, subsystem i's repairs write into subsystem j's state, which is exactly π applied to the write-edges of G; a read-only dependency, $R_i \cap F_j \neq$ with no corresponding write, is a separate relation, drawn as a dashed edge if the graph is rendered, since Section 6's treatment of G already declined to conflate reads with

edges for the same reason. Under this definition the current thirteen operators instantiate one solid edge, immune $\rightarrow$ thermal, and one dashed edge, hepatichematologic; both are the same two dependencies already named in Section 4 and in the discussion of G above, restated here with enough structure that a future audit can compute I's dependency graph mechanically from the operator registry rather than by re-reading each module's source, which is precisely the fifth component of the specification template in Section 9, that operators declare what they read and write, made checkable rather than merely requested.

6.8 Observability versus Repairability

Every $k \in K$ has an observation function o_k and an admissible set $\mathcal{B}_k$ by construction, since both come from the same `ObservableBoundary` implementation; not every k has a matching repair operator, and the current implementation makes no attempt to guarantee one does. Formally, call k *repairable* at s if some registered operator's predicate matches a violation of k, and merely *observable* otherwise. Two concrete gaps of this kind exist in the current thirteen-operator library: `gi.gastric_ph` carries an admissible range via `gi::boundary` and is checked by `detect_violations_for` like any other variable, but no registered operator's predicate ever matches a `gastric_ph` violation, since `gi::motility_correction` matches only `motility_index`; `cardiovascular.heart_rate` is in the same position, constrained but unaddressed, since `baroreflex` matches only `mean_arterial_pressure`. A violation of either variable is detected, ranked, and, per Proposition 1's proof, simply dropped from `remaining` once `step` finds no matching operator, with no distinct signal in the log to distinguish "detected and left unaddressed" from "never violated at all" beyond the violation's absence from any `RepairEvent`. This is a genuine gap in the current implementation, not a claimed feature, but it does suggest a real distinction worth carrying forward into the specification template of Section 9: a subsystem's five-component specification should record, for each observable variable, whether it is presently repairable or only observable, so that a gap like the two above is a declared, auditable fact about the model's current coverage rather than something a reader has to rediscover by cross-referencing boundaries against operator predicates by hand, which is how both of the gaps above were found.

6.9 State-Dependent Admissibility

The formalization above defines $\mathcal{B}_k$ as fixed, independent of s, matching every `boundary` function in the current implementation. A more general and more physiologically accurate formalization replaces this with $\mathcal{B}_k(s) \subseteq R$, an admissible set that is itself a function of the whole organism state, so that, for example, the admissible range for core temperature could genuinely widen during an active inflammatory response, or the admissible range for heart rate could genuinely widen during active exercise. The current implementation approximates this idea in exactly one place without actually generalizing $\mathcal{B}_k$: `thermal::thermoregulation_apply` computes an internal `fever_shift` proportional to `immune.cytokine_level` and uses it to decide, inside the operator's own branching logic, whether to warm or cool, while the admissibility check that determines whether `thermal.core_temp` is in violation at all still consults the fixed, unmodified $[36.5, 37.5]$ interval returned by `thermal::boundary`. Put formally, the operator's own repair policy has effectively adopted a shifted target, while the admissibility policy consulted by detection has not: the two disagree, $\mathcal{B}_k \neq \mathcal{B}_k(s)$ in everything but name,

since nothing in the current implementation actually computes the right-hand side. One consequence of this architectural separation is that the four-hour infection scenario in Section 8 ends with one residual violation reported even though the fever it describes is, physiologically, a successfully bounded and arguably intended response rather than a true derangement: the detection layer has no way to know that the elevated temperature is currently tolerated, only the repair operator's internal logic does. Making $\mathcal{B}_k$ properly state-dependent, so that `detect_violations_for` itself consults the rest of the organism rather than a fixed lookup table, would resolve this specific artifact, would bring detection and repair back into agreement, and is listed as a near-term engine extension in Section 9.

6.10 Architectural Invariants

Independent of any particular subsystem's physiology, the implementation is intended to preserve five invariants, stated here so that future extensions can be checked against them explicitly. Time monotonicity: `state.t` is reassigned to `next_fire[idx]` at every tick, which is always `state.t` plus a continuation's `interval`, so simulation time strictly increases provided every `interval` implementation returns a strictly positive value; this is currently an unenforced assumption rather than a checked one, since `interval` returns a bare `f64` with no debug assertion guarding against a future implementation returning zero or a negative number. Deterministic replay: because every transition is a pure function of its explicit arguments, with no interior mutability and no ambient I/O anywhere in the eleven subsystem modules, a full run is a pure function of its initial state, its continuation and operator registries, and its input sequence; this currently holds vacuously with respect to genuine stochasticity, since `Perturbation`'s seed is not yet consulted by any `advance` implementation, and preserving the invariant once it is wired in will require a seeded deterministic generator rather than a system entropy source. Pure state transitions: every `RepairOp::apply` and every `Continuation::advance` takes the current state by immutable reference and returns a new state by value, a property enforced structurally by the absence of any interior-mutability type (no `Cell`, `RefCell`, or unsafe code) anywhere in the current subsystem implementations, and therefore checkable by inspection rather than merely assumed. Scheduler fairness: since `next_fire[idx]` strictly increases by that continuation's own `interval` every time it fires, and every currently implemented `interval` is a fixed positive constant independent of state, every continuation's next-fire time eventually becomes the minimum and it fires again within a bounded number of ticks, so every continuation fires infinitely often over an unbounded horizon; this guarantee is specific to fixed, state-independent intervals and would need to be reestablished, rather than assumed to carry over, once state-dependent cadences are introduced. Bounded repair iteration per tick: guaranteed unconditionally by the single-call boundedness proposition above.

7 Implementation

The current implementation covers eleven subsystems, each contributing an admissibility boundary over two to four observable variables, one or more repair operators, and a clock. Table 2 summarizes their observable variables, admissible ranges, and characteristic timescale.

Subsystem	Observable variables (admissible range)	Principal repair operator(s)	Clock interval
Cardiovascular	mean arterial pressure (70–105 mmHg), heart rate (50–100 bpm)	baroreflex	1 s
Renal	GFR (90–120 mL/min), plasma sodium (135–145 mEq/L)	RAAS-style response	15 min
Hepatic	bilirubin (0.1–1.2 mg/dL), albumin (3.5–5.0 g/dL), ammonia (11–35 μmol/L)	urea-cycle upregulation, synthetic upregulation	10 min
Gastrointestinal	gastric pH (1.5–3.5), motility index (0.6–1.4)	motility correction	5 min
Nervous (autonomic)	sympathetic tone (0.1–0.7), parasympathetic tone (0.1–0.7)	autonomic rebalance	2 s
Immune	WBC count (4.5–11.0 $\times 10^3/\mu$L), CRP (0–10 mg/L), cytokine level (0–0.3, relative)	immune resolution, fever response	5 min
Hematologic	hemoglobin (12–17 g/dL), platelet count (150–400 $\times 10^3/\mu$L), coagulation index (0.8–1.2, relative)	coagulation response, erythropoiesis	5 min
Endocrine	insulin (2.6–24.9 μU/mL), cortisol (5–23 μg/dL)	insulin response	2 min
Metabolic	blood glucose (70–140 mg/dL), lactate (0.5–2.2 mmol/L)	glucagon response	1 min
Respiratory	blood pH (7.35–7.45), $PaCO_2$ (35–45 mmHg), PaO_2 (75–100 mmHg)	ventilation response	4 s
Thermal	core temperature (36.5–37.5 °C)	thermoregulation	1 min

Table 2: Current subsystem coverage. Clock intervals are the seconds of simulated time between successive advances of that subsystem, not wall-clock time.

A whole-organism baseline state, constructed with all eleven subsystems at normal resting values, is fully admissible under every constraint listed above; this is checked directly by an automated test, described in Section 8, and serves as the starting point for scenario construction.

8 Experimental Results

To exercise the architecture beyond unit-level construction, we constructed a four-hour infection scenario: starting from the admissible baseline, a sustained exogenous pathogen load input drives the immune subsystem's white cell count, C-reactive protein, and cytokine level upward at each of its clock advances, while the immune resolution operator, triggered whenever any of those three variables is found in violation, pulls them back toward baseline. When cytokine level nonetheless exceeds its admissible upper bound, the fever response operator fires as a second, simultaneously dispatched operator against the same violation, raising core temperature, with the increment capped at a hard physiological ceiling of 41.5 °C regardless of the triggering violation's severity.

At a pathogen load sufficient to exceed the immune resolution operator's per-tick correction capacity, the four-hour run settles into a mild, bounded fever: core temperature rises from a 37.0 °C baseline to 37.6 °C and remains there, with one low-severity thermal violation still outstanding at the horizon rather than a fully corrected state, while white cell count, C-reactive protein, and cytokine level all stabilize just inside or just outside their respective boundaries rather than diverging. At a lower pathogen load, immune resolution alone is sufficient to keep cytokine level under its threshold for the full four hours, and no fever response fires at all; this is not a failure of the model but a legitimate outcome, a case in which the modeled immune response fully compensates for the modeled challenge.

Three automated tests accompany this scenario: that the baseline state is fully admissible across all eleven subsystems with no violations detected; that, with no pathogen load, the organism remains within half a degree of baseline core temperature over the same four-hour window; and that a sustained pathogen challenge both raises core temperature above baseline, confirming the cytokine-to-fever coupling actually fires, and keeps it below 40 °C, confirming that the coupling is bounded rather than runaway.

Constructing this scenario surfaced three substantive defects in the initial implementation, each worth recording because each reflects a general hazard in this style of architecture rather than a one-off coding mistake.

The first was a scheduling defect. The original `Continuation` trait computed each subsystem's next-fire time as the current global simulation time plus a fixed interval, recomputed fresh on every iteration of the scheduler. Because the fastest subsystem's next-fire time is, by this computation, always exactly one interval ahead of the current global time, it is always the minimum among all subsystems' next-fire times, and the scheduler therefore advanced only that one subsystem forever, never allowing any slower subsystem to fire at all. A four-hour run under this defect produced zero repair events, since the immune subsystem, on a five-minute clock, was never once advanced. The fix moves next-fire bookkeeping out of the trait and into the scheduler itself, one `f64` per subsystem, incremented by that subsystem's own interval each time it actually fires, rather than recomputed relative to a global clock every iteration.

The second was a directional defect in a repair operator. An operator named leukocytosis was registered as the corrective response to an elevated white cell count violation, but its implementation only ever increased white cell count further, since it was actually modeling the pathogen-driven mobilization process rather than any corrective mechanism. Because the violation it was nominally repairing was never actually resolved, the operator fired on every tick indefinitely, compounding without bound; a run under this defect produced a final core temperature in excess of ten million degrees, a strong single-scenario existence proof of the

general point that a repair operator whose predicate matches a violation it does not actually resolve will not merely fail to help but will actively write to the log as though the fix were working. The corrected implementation separates the two mechanisms: the pathogen-driven rise is modeled directly inside the immune subsystem's continuation, as the disease process it is, and a distinct operator, immune resolution, is the actual corrective response, decreasing white cell count, cytokine level, and C-reactive protein when any of them is found in violation.

The third defect was in the dispatch function itself. `step` selected, for each violation in turn, the first operator in the operator list whose predicate matched, using a short-circuiting search. Once both immune resolution and fever response were written to match the same cytokine-level violation, whichever of the two happened to be registered earlier in the operator list was applied and the other was never tried at all, regardless of how the scenario's inputs were tuned; no amount of raising the pathogen load could make fever response fire, since the search never reached it. This is not a parameter-tuning problem but a structural one, and it was only visible once a scenario existed that depended on two operators legitimately responding to the same trigger. The corrected dispatch applies every matching operator for the selected violation before moving to the next, which is also the physiologically more accurate model, since a single derangement routinely provokes more than one simultaneous compensatory response in a real organism.

We record these three defects not as an appendix of trivia but because they are the clearest evidence available so far for the architecture's central claim in Section 4: that separating detection, dispatch, and domain content into distinct layers keeps each layer's correctness independently checkable, and each of these three defects was in fact caught by asking a very small number of direct questions of the running system (does any event fire at all; does the corrective operator actually move the variable toward the boundary; does every operator that should fire for a given violation actually get the chance to) rather than by inspecting the full state trajectory for physiological plausibility. The cost of this style of architecture is that these questions must be asked explicitly and early; nothing about the design prevents a directional or dispatch defect from persisting silently.

9 Subsystem Hierarchy

The eleven subsystems in Table 2 cover a narrow slice of a complete physiological model, and were selected for architectural coverage (fast versus slow clocks, single-subsystem versus cross-subsystem repair, a nontrivial coupling chain) rather than for clinical completeness. Rather than list the remaining work as a single flat table, we organize it into three levels, chosen because they correspond to a real distinction the `Continuation` trait already makes: Level 1 systems are organ-scale, lumped subsystems comparable in grain to the current eleven, each with its own clock on the order of seconds to days; Level 2 systems refine an existing Level 1 subsystem, or a modulatory layer that cuts across several of them, into anatomically or mechanistically resolved detail, typically on a comparable or slightly finer timescale; and Level 3 systems sit beneath the organ level entirely, at the cellular or molecular scale, where the current organ-level `Continuation` scheduling would need to be extended downward rather than merely added to. This hierarchy is a reorganization of the roadmap, not a change to the engine: every entry at every level still conforms to the five-component specification template stated below.

Level 1 system	Scope	Coupling
Lymphatic	Lymph flow, interstitial fluid volume, edema index, lymphocyte trafficking; repair operators increase lymphatic pumping, enhance drainage, or redistribute interstitial fluid	Reads cardiovascular plasma volume; influences immune cell transport
Musculoskeletal	Muscle force-length-velocity relationships, joint torque, posture-dependent cardiovascular loading, fatigue accumulation and recovery, glycogen depletion	Turns exercise into a genuine mechanical process rather than a scalar input
Integumentary	Skin as an organ: barrier function, perfusion, and the anatomical seat of the thermoregulatory effectors detailed at Level 2	Coupled to thermal and cardiovascular
Reproductive	Ovarian and testicular cycles, pregnancy-associated physiological shifts, placental circulation, lactation	Independent subsystem with its own multi-week to multi-month clock
Skeletal	Calcium buffering, phosphate homeostasis, bone remodeling, marrow function	Couples tightly with endocrine and hematologic
Microcirculation	Capillary exchange, tissue perfusion, oxygen extraction, local autoregulation, beyond cardiac output and arterial pressure alone	Reads cardiovascular and respiratory; feeds tissue-level oxygen delivery to every other organ-level subsystem

Table 3: Level 1: additional organ-scale subsystems, comparable in grain to the current eleven.

Level 2 refinement	Scope	Refines
Pulmonary alveoli	Alveolar ventilation-perfusion matching, dead space, lung compliance, work of breathing	Respiratory

Level 2 refinement	Scope	Refines
Nephron segments	Filtration, tubular reabsorption and secretion segment by segment, concentration and dilution mechanisms	Renal
Endocrine axes	Hypothalamic-pituitary-adrenal (CRH, ACTH, cortisol), hypothalamic-pituitary-thyroid (TRH, TSH, T3/T4), hypothalamic-pituitary-gonadal (GnRH, LH, FSH, sex steroids), renin-angiotensin-aldosterone as a genuine cross-subsystem loop rather than the current renal-only placeholder, and vasopressin (ADH), each as an explicit feedback loop rather than a single hormone concentration	Endocrine, Renal, Cardiovascular
Coagulation cascade	The clotting cascade resolved stepwise rather than as a single coagulation index	Hematologic
Adaptive immune decomposition	Innate immunity, adaptive immunity, inflammation, antibody production, tissue repair, and immunological memory (memory cell populations) as distinct compartments	Immune
Skin thermoregulation effectors	Sweating, vasodilation, vasoconstriction, and evaporative cooling as distinct effectors with different time constants, rather than one lumped correction	Thermal, Integumentary
Digestive expansion	Stomach, small intestine, pancreas, liver, and colon resolved separately, with nutrient absorption, gastric emptying, and bile secretion	Gastrointestinal, Hepatic
Nervous subdivision	Autonomic (continuing to handle rapid reflexes), sensory, and central components, including sleep, circadian regulation, pain, and stress response	Nervous
Skeletal remodeling	Bone remodeling mechanics resolved beneath the Level 1 skeletal subsystem	Skeletal, Endocrine
Microbiome	Gut microbial population dynamics and their metabolic and immunomodulatory contributions	Gastrointestinal, Immune
Pharmacokinetics and pharmacodynamics	Absorption, distribution, metabolism, and elimination of exogenous compounds, coupled to the relevant subsystems' repair operators as agonists or antagonists	Cross-cutting

Level 2 refinement	Scope	Refines
Injury and trauma	Discrete, large-magnitude perturbations (hemorrhage beyond the current single scalar rate, fracture, burn) with subsystem-specific acute responses	Cross-cutting
Growth, development, and aging	Slow, multi-year modulation of baseline values and admissible ranges themselves, rather than a fixed boundary	Cross-cutting; requires $\mathcal{B}_k$ to become a function of simulation time

Table 4: Level 2: mechanistic or anatomical refinements of an existing or planned Level 1 subsystem, and cross-cutting modulatory layers.

Level 3 system	Scope	Notes
Cellular metabolism	ATP production, mitochondrial activity, oxidative stress, reactive oxygen species, apoptosis, autophagy	A bridge toward a multi-scale version of physiome
Gene regulation	Transcriptional response as a slow modulator of a cell's own repair capacity	Longest-horizon addition
Signaling pathways	Intracellular cascades underlying the organ-level repair operators above	Would let a Level 1 or Level 2 operator's magnitude itself become a modeled quantity rather than an asserted constant
Mitochondrial dynamics	Fission, fusion, and mitophagy	Couples most directly to cellular metabolism
Tissue remodeling	Extracellular matrix turnover underlying organ-level repair operators such as `synthetic_upregulation`	Couples most directly to hepatic and skeletal

Table 5: Level 3: cellular and molecular systems beneath the organ level, where organ-level lumping is insufficient.

Extending the current `Continuation` scheduler down through Level 3 is itself an architectural extension rather than a new subsystem in the sense of Tables 3–4: it requires `Continuation`

to compose across levels, so that a Level 3 signaling pathway's clock can be meaningfully nested beneath the Level 1 or Level 2 subsystem it underlies, rather than registered as one more independent entry in a flat list alongside eleven organ systems. This is listed separately from any individual Level 3 row because it affects how `Continuation` itself is defined, not merely which subsystems exist.

Beyond adding subsystems, three extensions to the engine itself are anticipated. First, the admissibility boundary for a variable should in general be permitted to depend on the state of other subsystems, as already done ad hoc for the thermal subsystem's fever-adjusted set point; a principled version of this would make `AdmissibilityBoundary` itself a function of the whole `PhysiologicalState` rather than a fixed lookup table, at some cost to the current implementation's simplicity. Second, the deterministic perturbation source presently carries only a random seed and is not yet threaded into any subsystem's advance function; a full implementation requires deciding, subsystem by subsystem, which observable variables are subject to genuine stochastic variation and wiring the perturbation source into their continuations. Third, validating any of the above against clinical reference ranges and, where available, published physiological datasets is a separate undertaking from the architectural work described here, and should be treated as its own workstream once a given subsystem's specification is judged complete, rather than folded into subsystem development itself.

For each planned subsystem across all three levels, a complete specification, in the sense used throughout this project, consists of five components: the subsystem's observable variables together with their units; the admissible range for each variable, ideally sourced from a stated clinical reference rather than an illustrative placeholder; the subsystem's characteristic clock interval, justified by the physiological timescale it is meant to represent; the repair operators available to it, each stated as a named, deterministic transition together with the violation predicate that triggers it; and the specific fields of other subsystems, if any, that its repair operators read from or write to, since this is precisely the information needed to keep cross-subsystem coupling auditable as the model grows. The current eleven subsystems already conform to this template, as shown in Table 2 and the coupling examples in Section 4; extending coverage is a matter of applying the same template to each new system in Tables 3 through 5, and doing so is deliberately additive at the engine level, requiring one new module and a handful of registration lines, not a modification of `step` or `step_until`.

Alongside the organ-system roadmap, a distinct mathematical roadmap follows from Section 6. The state-dependent admissibility boundaries $\mathcal{B}_k(s)$ motivated there and only approximated once, ad hoc, in the current thermal subsystem, are the most immediately actionable item. Beyond that, the interval-based severity function could be generalized to a continuous metric over a richer state space per variable, rather than a scalar distance to a fixed interval, which would matter most for variables whose admissibility is graded rather than binary. The deterministic repair operators of Section 6 could be generalized to probabilistic operators, sampling from a distribution over corrective transitions rather than applying a fixed one, using the already-present but currently unwired `Perturbation` source; this would also make Proposition 2's finite-time guarantee inapplicable in its current form and require a probabilistic convergence statement in its place. A further, more speculative direction is learning admissible boundaries and repair magnitudes from clinical reference data rather than hand-specifying them, which would change the epistemic status of Table 2's entries from asserted to fitted and would need its own validation methodology, distinct from the architectural correctness questions this paper has focused on. A compositional algebra of subsystem interfaces, in which

the currently ad hoc struct-of-structs `PhysiologicalState` and the read/write coupling discipline of Section 4 are replaced by an explicit, checkable specification of which fields each subsystem is permitted to read and write, is a natural target once enough subsystems exist to make the current informal discipline hard to audit by inspection, as the current eleven still can be. Finally, a hybrid model, in which ordinary continuous dynamics (differential equations, in the tradition contrasted in Section 3) govern a variable's evolution while it remains within its admissible region, and repair operators intervene discretely only at the moment a boundary is crossed, would combine this architecture's admissibility-first dispatch with the quantitative continuity that a pure discrete-repair model presently lacks between violations, and is plausibly the most direct route toward clinical-grade fidelity without abandoning the separation of engine and content that motivates this architecture in the first place.

10 Discussion: Homeostasis as Repair

The formal and architectural material above has been organized around making the engine checkable; this final section steps back to a claim about physiology itself, that the engine's detect-rank-dispatch-reassess loop is not merely a convenient computational form but a reasonable description of what homeostatic regulation actually is. `baroreflex` detects a mean arterial pressure violation, ranks it against whatever else is currently violated, dispatches a chronotropic and inotropic correction, and the next tick reassesses; `insulin_response` does the same for blood glucose, `coagulation_response` for the coagulation index, `thermoregulation` together with `fever_response` for core temperature, `immune_resolution` for the inflammatory markers, and `ventilation_response` for $PaCO_2$. These six are not a curated subset chosen to make the analogy work; they are simply named among the current thirteen operators because each corresponds to a physiological control loop with a standard textbook name, and each already fits the four-stage loop without modification, which is some evidence, short of a proof, that the loop is closer to a genuine description of physiological regulation than to an artifact of how this particular codebase happens to be organized.

Two design principles follow from taking that description seriously. The first, argued at length by example in Section 8, is that failure is itself information rather than something to suppress: a variable that stays in `remaining` after every matching operator has fired, or that has no matching operator at all as in Section 6's observability-versus-repairability discussion, is not a crash or an exception, it is a persisting entry in the violation log, exactly the kind of output a clinician reading an unresolved abnormal value would treat as informative rather than as a system failure. The second is that the engine is not trying to compute an optimal physiological response, only a valid one; `step` does not search over candidate operators for the one that would minimize some cost function, it applies whichever registered operators match, in registration order, and stops once the state is admissible or the iteration budget is exhausted. This is a substantive commitment, not a shortcut: biological homeostasis is generally understood as satisficing rather than optimizing, a system that returns a variable to within range rather than to a computed ideal point within that range, and an engine that only ever asks is this admissible, never what is the best admissible value, is a closer structural match to that understanding than an optimization-based controller would be.

It is also worth being explicit that this architecture was not designed top-down from these principles and then implemented; Section 8 documents the reverse. A single global scheduler

became independent per-subsystem clocks only after the monopoly defect made the problem visible; a single-operator-per-violation dispatch became the multi-operator composition of Section 6 only after the fever scenario needed two operators to respond to one trigger; the formal admissibility relation in Section 6 was written down only once enough of the implementation existed to check it against. The abstractions in this paper emerged from getting a concrete scenario to behave correctly, not from a prior theoretical commitment that was then implemented, which is offered here as a fact about how this particular project developed, not as a general claim about how architecture papers should be written.

Whether any of this generalizes beyond physiology is a genuinely open question that this paper does not attempt to answer, and we flag the temptation explicitly rather than yielding to it: admissibility boundaries, declarative repair operators, and asynchronous boundary-driven scheduling are not obviously specific to physiological simulation, and a robotics controller, a distributed system's supervisory layer, or an ecological model could plausibly be organized the same way. Saying so is speculation, not a result of this paper, which has one implementation, one scenario, and thirteen operators behind it; the honest version of this closing thought is that the architecture described here is worth trying on a second domain before any claim broader than "this worked for a small physiology sketch" is warranted.

11 Conclusion

physiome is presently a small, early-stage sketch: eleven subsystems, thirteen repair operators, one worked scenario, none validated against clinical data, and a formal framework in Section 6 whose propositions are checked against the current implementation rather than against a fuller one. Its contribution at this stage is architectural, an admissibility- and repair-first alternative to the coupled-differential-equation style of whole-body modeling, implemented with an explicit separation between a small domain-independent engine and a growing, declarative library of subsystem-specific content, and now stated formally enough that its dispatch semantics, non-commutativity, and complexity can be reasoned about directly rather than only demonstrated by example. The three defects surfaced in Section 8, a scheduling defect, a directionally backwards repair operator, and a dispatch function that silently discarded all but the first matching operator, were each caught by a small number of pointed questions asked of a single concrete scenario, and each is the kind of defect this architecture is specifically intended to make checkable in isolation; Section 6's convergence proposition and repair-graph formulation are an attempt to make that checking systematic rather than incidental, though neither has yet been exercised against a case with genuinely competing, non-commuting operators, which remains open work. Whether the architecture's checkability continues to hold as the subsystem count grows toward the three-level hierarchy in Tables 3–5, and as state-dependent admissibility and probabilistic repair operators are introduced per Section 9, is the central open question for the next phase of this work.

References

[1] R. L. Hester, A. J. Brown, L. Husband, R. Iliescu, D. Pruett, R. Summers, and T. G. Coleman. *HumMod: A Modeling Environment for the Simulation of Integrative Human Physiology*. Frontiers in Physiology, 2011.

[2] A. C. Guyton, T. G. Coleman, and H. J. Granger. *Circulation: Overall Regulation*. Annual Review of Physiology, 1972.

[3] T. G. Coleman, R. L. Hester, and R. L. Summers. *HumMod 2010 Schema*. University of Mississippi Medical Center, 2010.