Coherence in Persistent Adaptive Systems:
From Individual Memory to Collective Governance

Mark Lubin
mark@synix.dev

April 2026

Abstract

This note summarizes an ongoing line of work on what persistent systems—individual agents and collectives of agents—require structurally in order to maintain coherence over time. The work currently consists of four artifacts at different stages of maturity: Synix (implemented memory pipeline system), a formal paper on the structural properties (Compress, Divide, Renew—in preparation), LENS (longitudinal benchmark with initial findings), and Aegis (coordination protocol, early specification). This note sketches the ideas, illustrates them at two scales, and identifies the open questions at the collective level.

1 Three Structural Properties

A system with finite resources that has to keep processing new information over time faces three structural problems. We develop formal arguments for each in the companion paper, but the intuitions are straightforward.

Compression is lossy and permanent. Any system with finite memory that receives a continuous stream of new information must eventually forget something—once the incoming information exceeds what the memory can hold, some of it gets discarded. The deeper problem is that this loss is permanent. No amount of clever internal reorganization can recover what was overwritten. Absent re-exposure from the environment, the set of questions the system can no longer answer only grows over time.

But compression is not just an unavoidable cost—it is also necessary for the system to function. Without it, the system drowns in its own history. Attention degrades over large contexts (the "lost in the middle" problem), irrelevant information pollutes retrieval, and with no value function over what to keep, every query becomes a needle-in-a-haystack search. Compression is what creates salience—it is the mechanism by which the system decides what matters. The problem is not that compression happens but that the value function driving it is tuned to the current distribution—and the distribution drifts. What mattered yesterday may not matter tomorrow, and what was discarded as noise may turn out to have been signal. This is what connects compression to renewal: the compression scheme itself has a shelf life.

Division helps. Given that compression is unavoidable, the question is how to organize it. The most obvious axis is temporal: a system tracking both fast-changing and slow-changing phenomena with a single update rate either misses fast changes or wastes resources

re-encoding slow ones that haven't changed. Partitioning state into components matched to different timescales reduces worst-case error.

But timescale is not the only axis. Partitioning can also be spatial, functional, or organizational. The deeper reason to partition is not efficiency but robustness: a perturbation to a partitioned system damages only one component, while the same perturbation to a monolithic system can degrade information everywhere simultaneously. Division is what makes a system antifragile rather than merely fault-tolerant: isolated failure in one partition creates pressure to renew that partition without destabilizing the rest.

Renewal is necessary. Things go wrong over time. Compression schemes drift as the world changes underneath them. Learned patterns turn out to be maladaptive. Hypotheses get disproven by new evidence. Pathologies emerge—runaway feedback loops, self-reinforcing biases, subsystems optimizing for themselves at the expense of the whole. None of this is preventable by better initial design. It is a consequence of operating under uncertainty in a non-stationary environment.

The response is not to prevent degradation—that is not possible—but to make renewal a first-class operation. Every component has a finite useful life, and there is an optimal lifespan computable from the degradation curve: replace when the cost of operating a stale compressor exceeds the cost of rebuilding it. Sacrificing a component at one level is what prolongs coherence at the level above it—a stale partition is renewed so the hierarchy survives, a stale agent is culled so the colony adapts. The system must be able to let parts of itself die so the whole can continue to function.

These three problems do not arise once and then stop. Each component produces compressed output that becomes the input for the next level of the hierarchy. As long as the compressed output still carries genuine information about longer-timescale structure, the same three problems—compression, division, renewal—recur at every level. The hierarchy's depth is set by the source: it extends until no further timescale structure remains or coordination costs exceed the benefit. The result is a multi-level architecture that emerges from information constraints, not from engineering convention.

A system that cannot compress degrades under its own accumulated state. A system that cannot divide has no fault isolation. A system that cannot renew accumulates stale assumptions until its behavior diverges from its environment. That divergence is misalignment, stated in architectural terms.

2 Individual Scale: Memory as Tiered Lifecycle

One way the structural properties might instantiate for an individual agent is as a tiered memory architecture, where each tier operates at a different timescale with its own physics—not different database schemas, but different access patterns, different lifecycle rules, and translation interfaces between them.

We think a plausible partition looks something like this:

Execution (milliseconds to minutes)
Stack, heap, and registers. KV-cache native, attention-routed, volatile. Frame push/pop provides scoped lifecycle.
Session (minutes to hours)
Working set. In-memory, incremental, per-turn updates. LRU plus salience-based eviction. Bridges execution and experience—the implicit conversation history made explicit

and managed.

Experience (hours to weeks)
Consolidation. Episodic. Episodes arrive from session eviction and are consolidated into narrative arcs, restructured when significant events arrive.

Identity (weeks to permanent)
The slowest-changing, most consequential region. Identity is *formed* through compression of lived experience—repeated episodes become character. Conservative: requires strong episodic evidence to update.

The invariants at this scale are the interfaces between tiers. Consolidation flows downward: hot state cools and compresses. Retrieval flows upward as a page-fault cascade. Corrections propagate upward via invalidation—a user correction writes directly to identity and flushes stale entries above it, the same pattern as cache invalidation in a memory hierarchy.

The system is event-driven, not clock-driven. Structural triggers always fire (frame pop, session end). Pressure triggers fire on threshold (working set full). Salience triggers fire on weight (a significant event cascades deeper). Correction triggers fire on contradiction and cascade upward. The trigger taxonomy determines which tier processes what, and when—it is the mechanism by which the tiers maintain their timescale separation rather than collapsing into a single undifferentiated store.

3 Collective Scale: Governance and Self-Modification

The recursive structure of the properties means that when you move from a single agent to a collective, the same problems reappear at population scale. A single node can compress, divide, and renew its own memory—but it cannot produce organizational knowledge, allocate work across domains, or scale beyond one agent's context window.

Overlapping nodes are compressed (merge). Different domains get dedicated nodes at appropriate clock speeds (divide). Stale nodes are refreshed or decommissioned (renew). Colonies form through pressure, not top-down design: when an agent's scope drifts into uncovered territory, it spawns a specialist. Collective decisions—spawn, cull, merge—follow quorum-based governance. Individual observations promote to shared organizational knowledge through an evidence pipeline where each stage requires corroboration from multiple independent sources.

What can change?

The harder question at the collective level is self-modification: which parts of the system are allowed to change, and which must remain invariant?

Not everything can be evolvable—some properties are structural, and removing them breaks coherence. But not everything can be fixed either. A system whose rules can never change accumulates the same distributional drift that makes individual compression schemes go stale. The protocol we are developing distinguishes three classes:

Hardwired (immutable)
Structural invariants the protocol's safety depends on. Provenance requirements (nothing enters memory without a traceable source). The eval trace format (the system must not modify how it measures its own performance). The conflict resolution hierarchy. The structural properties themselves—if the formal model holds, they follow from the constraints rather than being design choices.

Evolvable (system-proposed, validated)
Parameters the system may propose changes to, given sufficient evidence from its own operational traces. Decay rates, confidence thresholds, routing rules, renewal intervals. Every modification carries the evidence that motivated it, a counterfactual evaluation, and a computed review date.
Learnable (system-discovered, compiled)
Patterns discovered through experience and compiled into explicit rules. Classification shortcuts, query routing patterns, contradiction detection heuristics. These carry computed expiration dates and are automatically demoted when their performance degrades.

Every compiled rule follows a lifecycle: experience, hypothesis, evidence, compilation, monitoring, renewal or culling. The lifecycle is evidence-gated, and every rule that reaches compilation has a computable optimal lifespan. The rules governing the collective are themselves subject to the same lifecycle—they compile from experience, carry expiration dates, and get demoted when they stop working.

Where this has worked before

This three-layer structure (hardwired, evolvable, learnable) is not novel. It appears in biological systems: DNA is mostly hardwired, epigenetic markers are evolvable on developmental timescales, and gene expression is continuously adaptive. The immune system maintains invariant structural components, an adaptive repertoire that evolves through selection, and short-lived effector responses that are continuously renewed. Similar patterns show up in common law (constitutional structure vs. statute vs. case-level adaptation) and in commons governance.

Elinor Ostrom's design principles are particularly suggestive. Clearly defined boundaries map to bounded state. Nested enterprises—polycentric institutions at multiple scales—map to the recursive necessity of the structural properties at every level of organization. Ostrom's principles are derived empirically from case studies of fisheries, forests, and irrigation systems. The formal model suggests these may not be just patterns people have found—they may be forced by the constraints.

The open question is where exactly the boundary between hardwired and evolvable falls—and whether the biological and sociological models can provide better guidance on that boundary than the engineering intuitions we started with.

Current Work and Findings

The ideas described above emerged from a progression of concrete implementation and evaluation work, each stage motivating the next.

The starting point was **Synix**, a memory pipeline system that implements tiered lifecycle management for individual agents. It handles cold ingestion (documents, conversation history) through a map-reduce-style pipeline with configurable synthesis and indexing stages, and hot ingestion (live conversation) through a stack-and-heap context protocol that manages execution memory within the context window. The stack-and-heap protocol achieves 65% token reduction over unmanaged context on coding tasks. Building Synix is what surfaced the structural properties in the first place—the tier architecture was initially an engineering intuition, and the question of why it needed to be hierarchical is what led to the formal work.

That formal work became a paper, **Compress, Divide, Renew** (in preparation), which

develops the structural properties as information-theoretic results: the tradeoff between monolithic and partitioned compression, the existence of optimal renewal intervals, and the recursive structure. The formal results gave the architectural intuitions from Synix a foundation, but they also raised a question: do existing memory systems actually fail in the ways the theory predicts?

To answer that, we built **LENS**, a longitudinal benchmark that tests eight memory architectures over 120-episode sequences across multiple domains. The core question is not whether information can be retrieved once but whether representations remain coherent under accumulation, compression, and distributional shift over time. The central finding so far is a negative result: no dedicated memory architecture outperforms a simple lossless retrieval baseline, because the dedicated systems' compression schemes degrade under exactly the drift the structural properties predict. That result confirmed the theory's predictions at the individual level, and raised the natural next question: do the same problems recur when multiple agents try to coordinate?

That question led to **Aegis** (early specification), a protocol for multi-agent coordination: colony formation, collective decision-making, organizational memory promotion, and the self-modification lifecycle described in the previous section. A reference implementation is in progress. Aegis is where the structural properties meet governance—the same compression, division, and renewal problems, but now applied to the rules and conventions that hold the collective together.

Future Directions

The main open questions are at the collective level. The hardwired/evolvable boundary in the self-modification protocol is an engineering conjecture, not a formal result—we do not yet have a principled account of which properties must be invariant versus which are safe to evolve. Detecting compression degradation at the collective level—when an organization's shared model has gone stale—is harder than detecting it at the individual level, and the right signals are not yet clear. And the failure modes of self-modifying governance systems—what happens when modification occurs at a layer that should have been invariant—are not well understood. The biological and sociological precedents suggest guardrails, but translating those into protocol-level constraints is ongoing work.