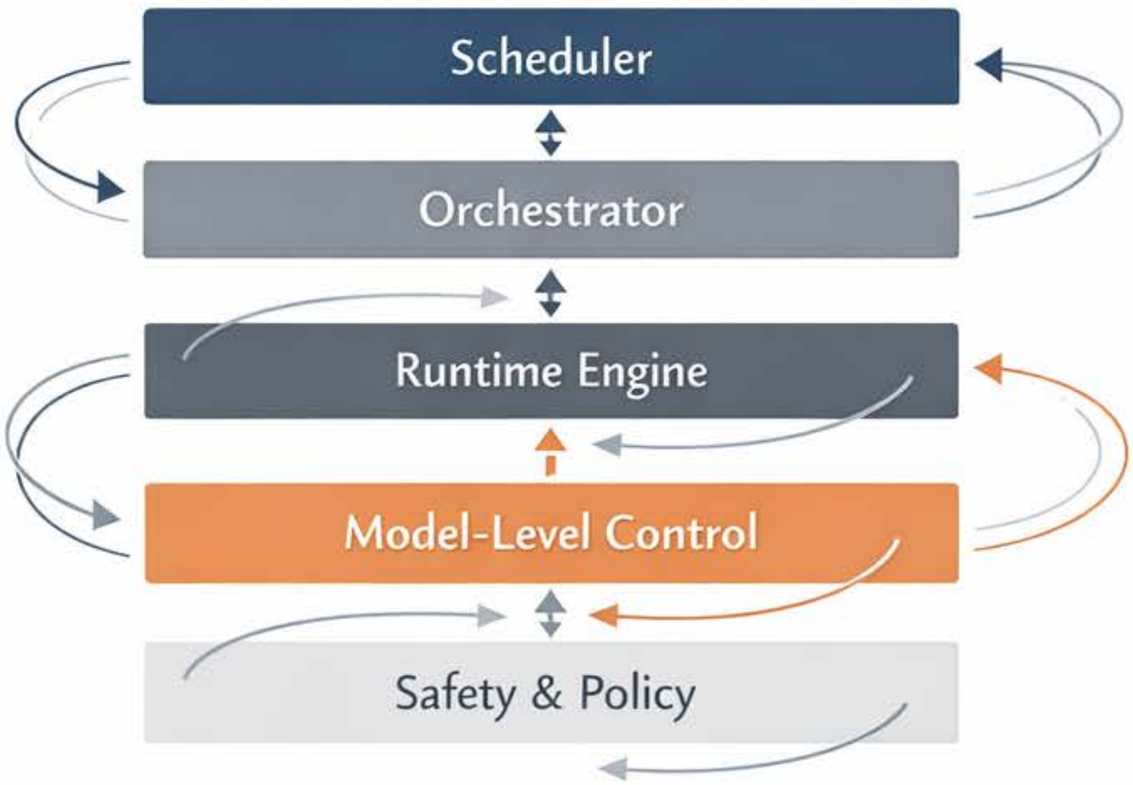
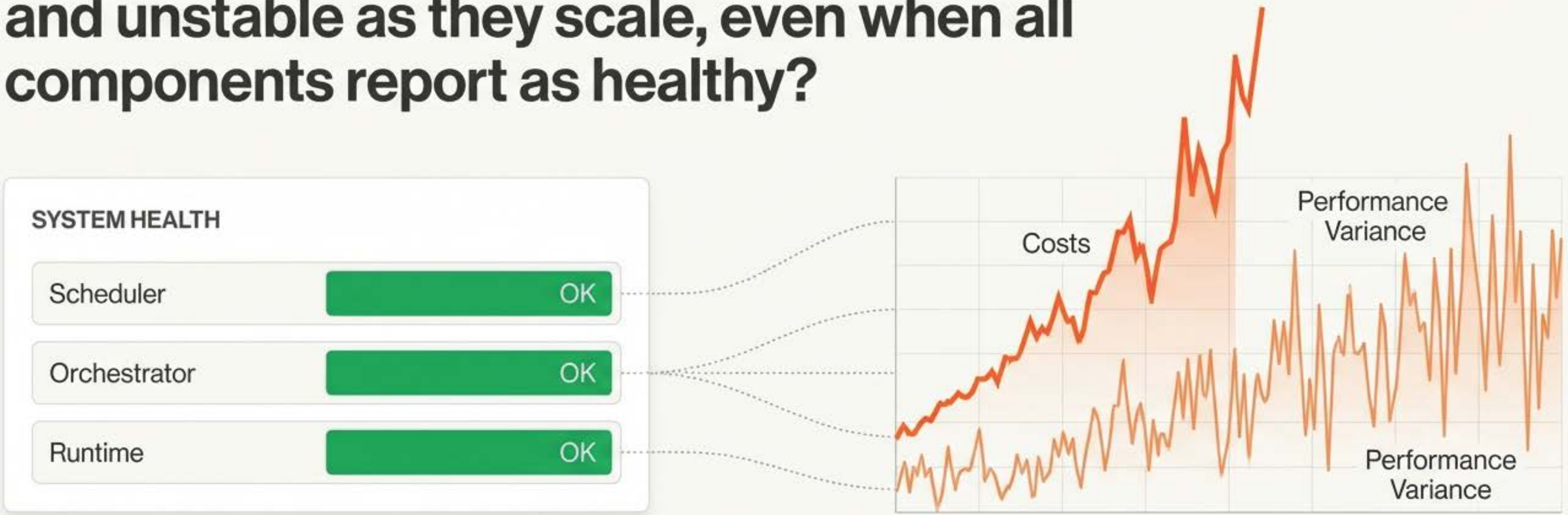
Runtime Control Coherence in Large-Scale AI Systems

Structural Causes of Cost, Instability, and Non-Determinism Beyond Interconnect Failures



Why do our AI systems get more expensive and unstable as they scale, even when all components report as healthy?



Large-scale AI deployments exhibit cost escalation and non-deterministic behavior that cannot be explained by conventional resource bottlenecks or interconnect limitations.

This divergence between *local correctness* (healthy components) and *global inefficiency* (rising costs, unstable performance) points to a structural risk that conventional observability systematically fails to capture.

This assessment explains the hidden cause of this divergence.

The usual responses fail to address the root cause.

The Myth of Over-Provisioning

"It's a capacity problem." Adding more hardware capacity does not eliminate the underlying conflicts. It often masks symptoms temporarily and can even amplify the problem by expanding the surface for control-plane interactions.

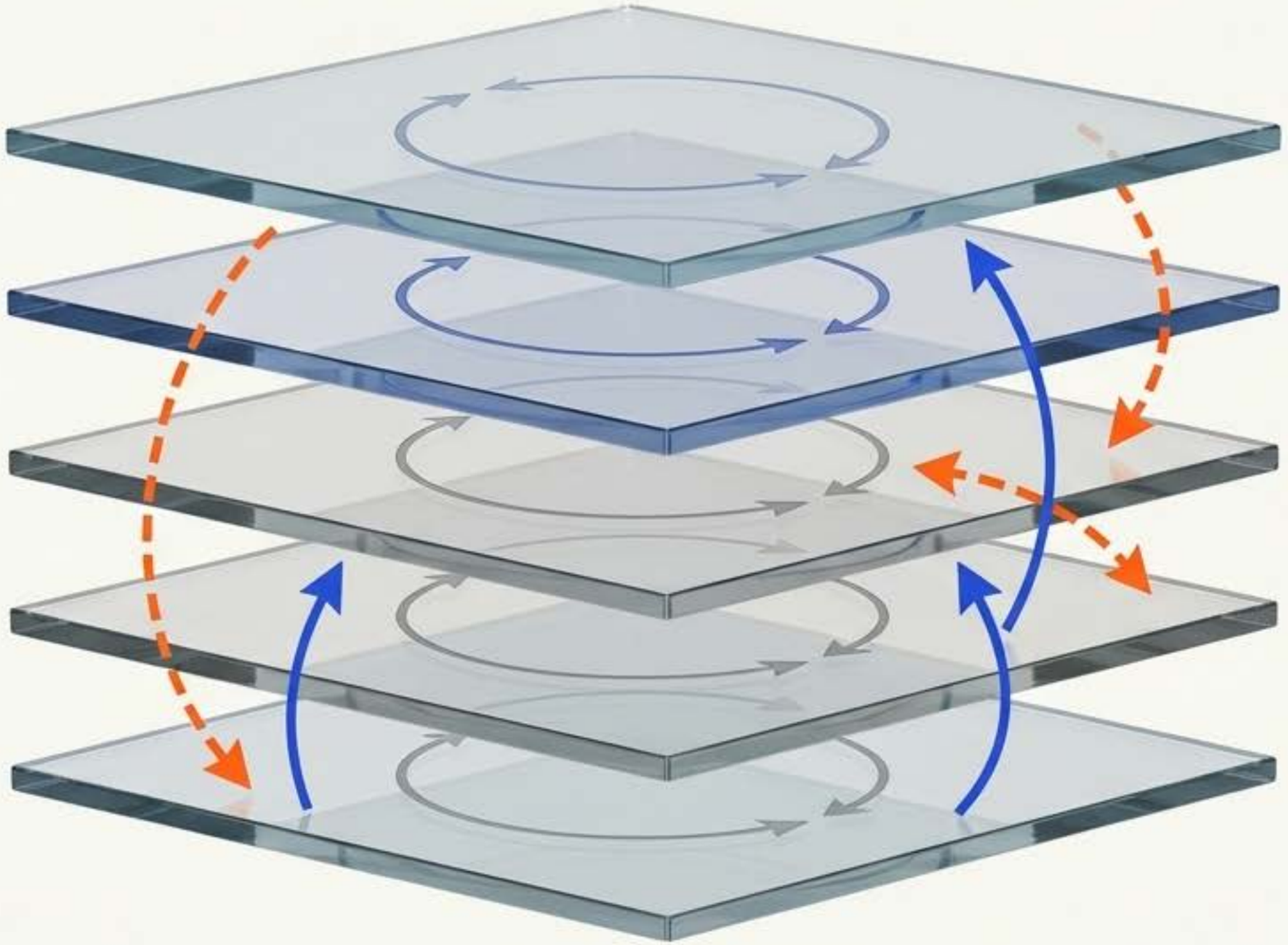
> *"The economic dead end is reached when further capacity additions no longer improve perceived stability because incoherence effects scale with the control surface."*

A Distinct and Complementary Risk

"It's an interconnect problem." While interconnect stability is critical, this assessment addresses a separate class of failures. These are not physical, but logical and regulatory coupling effects arising from the interaction of autonomous control mechanisms.

> *"Control coherence risks concern how conflicting or misaligned control decisions destabilize system behavior even when the underlying interconnect operates within specification."*

The problem lives in the interactions between autonomous control layers.



1. **Cluster & Batch Schedulers**
   Govern resource allocation and job placement at the infrastructure level.

2. **Orchestration Layers**
   Manage container lifecycle, service deployment, and scaling decisions.

3. **Runtime Execution Engines**
   Manage framework-level execution, retry logic, and backpressure.

4. **Model-Level Control**
   Handle adaptive inference, dynamic batching, and checkpointing.

5. **Safety & Policy Layers**
   Enforce timeouts, rate limits, circuit breakers, and compliance gates.

Each layer operates with its own objective, timescale, and partial view of

Each layer operates with its own objective, timescale, and partial view of the system. This autonomy is the foundation of the risk.

The structural risk has a name.

Runtime Control Incoherence

The degree to which distributed control decisions operating on a shared execution structure remain mutually consistent over time.

Incoherence is the absence of this consistency, where control actions negate, counteract, or unintentionally amplify one another. It is a structural property of the system as a whole, not a fault within any individual component.

Incoherence arises from autonomous loops with conflicting goals.

Each autonomous control loop is defined by:

- **Objective:** What it tries to achieve (e.g., maximize throughput vs. minimize latency).
- **Optimization Target:** The specific metric it watches (e.g., queue depth vs. response time).
- **Reaction Timescale:** How fast it acts (e.g., milliseconds vs. minutes).
- **Trigger Conditions:** What causes it to act (e.g., metric thresholds vs. timeouts).
- **Actuation Mechanism:** How it enforces its decision (e.g., job preemption vs. traffic throttling).

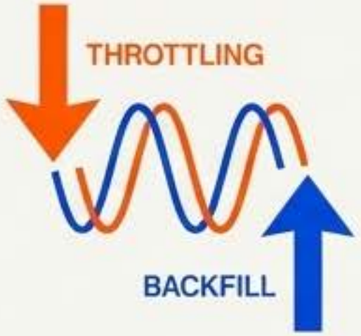
When loops with different objectives, targets, and timescales operate on shared resources without coordination, they create structural conflict.

Common incoherence patterns create predictable instability.

Competing Shutdown vs. Retry Logic

A safety layer terminates a task (e.g., timeout), while a runtime layer immediately retries it, believing it was a transient failure.

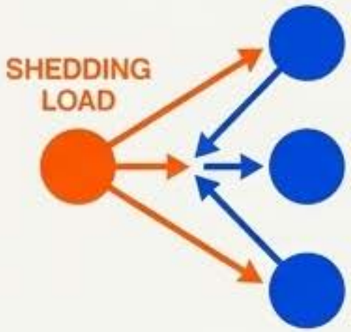
Result:
Resources are consumed in a loop of execution and termination with no progress.



Safety Throttling vs. Scheduler Backfill

A policy layer throttles execution to enforce rate limits, creating idle capacity. The scheduler sees this idle capacity and tries to fill it.

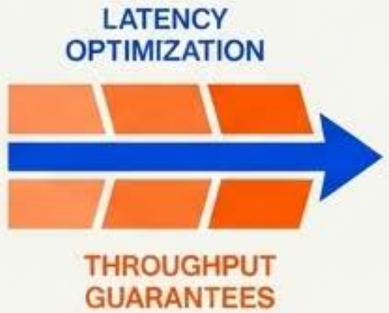
Result:
Oscillatory behavior, wasted scheduling overhead, and resource churning.



Local Load Shedding vs. Global Synchronization

An individual node sheds load to protect itself, but a distributed workload requires all nodes to be present for a global synchronization step.

Result: The entire job stalls, fragmenting global execution despite available aggregate capacity.



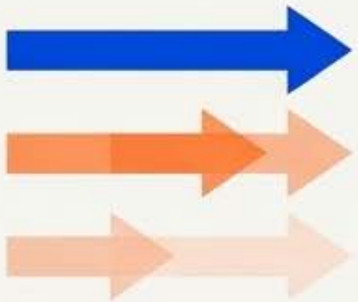
Latency Optimization vs. Throughput Guarantees

One control loop prioritizes and accelerates latency-sensitive requests, disrupting the batching and scheduling needed by another loop to guarantee overall throughput.

Result: Both objectives are partially compromised; tail latency might improve while overall system throughput drops.

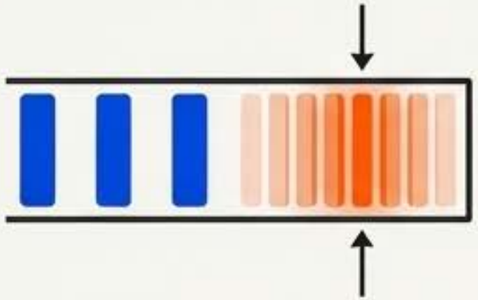
The outcome is not failure, but “Soft Degradation.”

The system operates, health checks pass, and no alarms are triggered, yet efficiency and predictability erode.



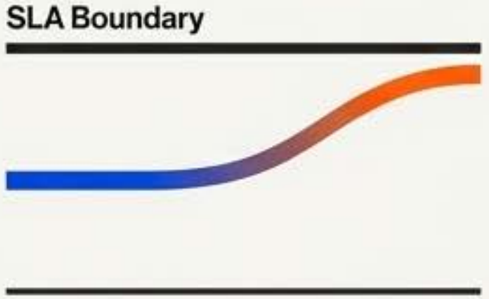
Retry Amplification without Error States

Retries are initiated by control actions, not true faults. The system performs redundant work, consuming capacity, but since retries eventually succeed, no errors are logged.



Queue Inflation without Saturation

Work items accumulate in queues due to misaligned pacing between layers. Latency increases, but no single queue hits its technical limit, so monitoring appears normal.



SLA Drift without Threshold Breach

Performance gradually degrades towards, but doesn't cross, SLA boundaries. The margin for error is silently eroded, increasing fragility.

Resource Consumption without Output Progression

Compute, memory, and energy are consumed, but the system makes no proportional progress toward completed work. Utilization is high, but value delivery is low.

Soft degradation creates “Ghost Costs.”

Economic expenditures that are not attributable to any identifiable fault and are silently absorbed into operational overhead.

Key Ghost Cost Mechanisms

- **Accelerator Time without Progress:** GPUs/TPUs are allocated and consuming power but are stalled by coordination waits or executing redundant computations.
- **Token Burn without Inference Delivery:** In inference systems, computational tokens are spent on aborted, retried, or otherwise wasted attempts that never deliver a result to the user.
- **Engineering Time on Non-Root Causes:** Teams chase symptoms and triage phantom incidents that are emergent effects of incoherence, diverting valuable engineering capacity.

Why conventional observability is blind to control incoherence

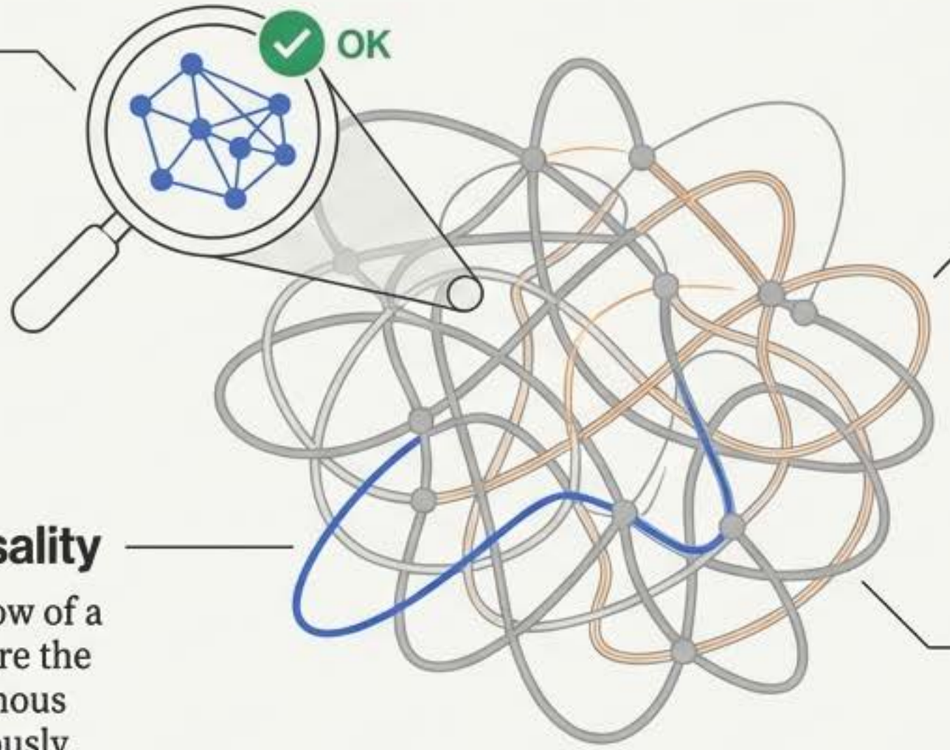
The problem is not a lack of data, but a mismatch between what is measured and the structural properties that govern system behavior.

KPIs are locally correct but globally misleading

Each control loop reports healthy metrics against its own objectives. Aggregating green lights doesn't reveal the systemic conflict.

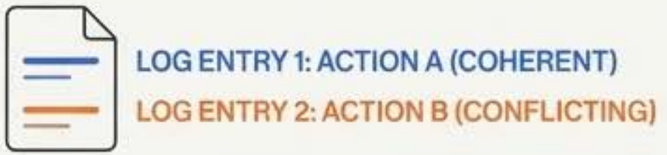
Traces miss cross-loop causality

Distributed tracing follows the data flow of a single transaction, but it cannot capture the causal relationships between autonomous control loops that operate asynchronously.



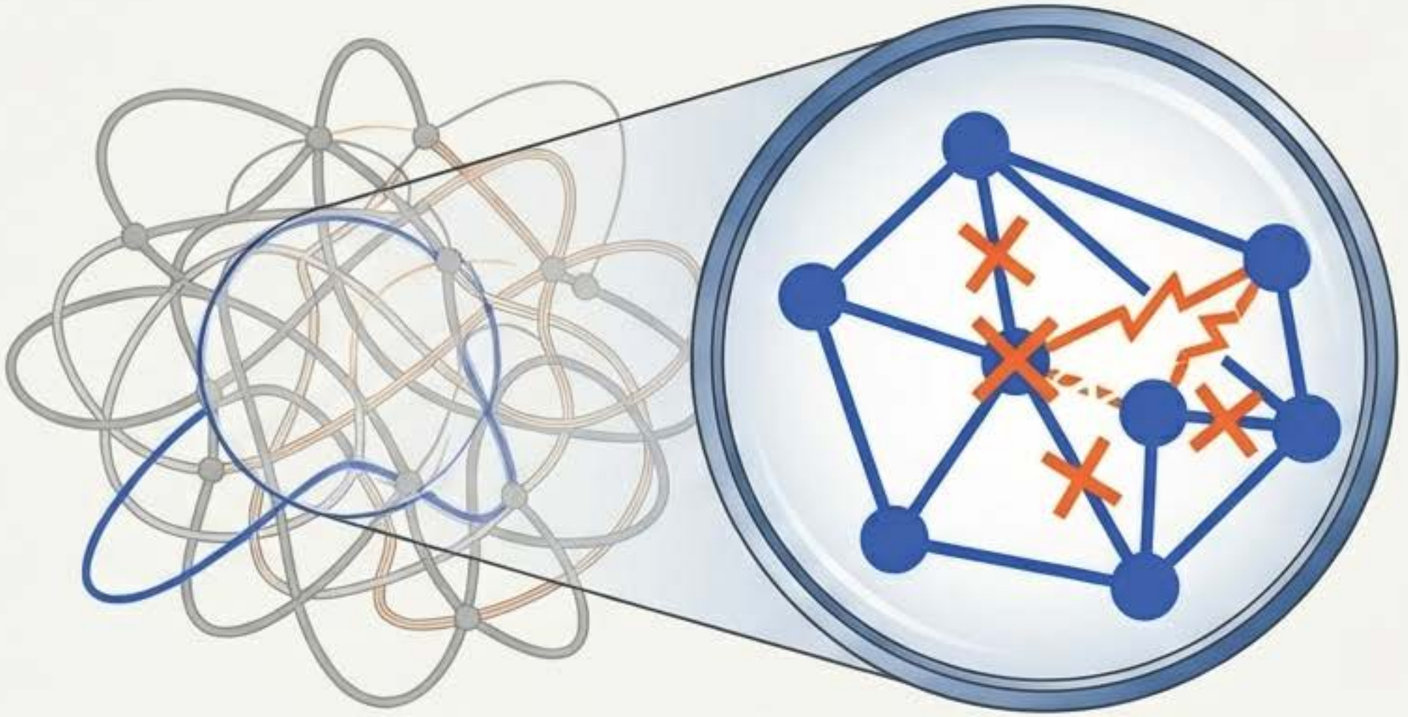
Logs capture events, not intent conflicts

Logs show *what* happened, but not the conflicting control objectives that *drove* the actions. Two conflicting decisions appear as unrelated log entries.



The result is an auditability deficit: organizations cannot reconstruct control-decision chains, leaving cost anomalies and degradation events without a root cause.

Solving this requires a new diagnostic lens.



This assessment does not propose a single solution. Instead, it provides a **structural framework** to make this invisible class of risks **visible**, enabling informed decision-making.

FROM:
Reactive, component-level firefighting → TO:
Proactive, structural risk analysis.

- It does **not** measure performance.
- It **identifies** structural risks and **explains** why systems are expensive and unstable despite appearing correct.

An assessment provides a toolkit for structural clarity.

The outputs deliver a structured view of control dynamics, risks, and cost exposures.

Control Plane Inventory

Documents all autonomous control loops, their objectives, timescales, and actuation mechanisms. Establishes a common language and reference.

Economic Risk Summary

Translates structural risks into economic terms, identifying direct and hidden cost exposures (Ghost Costs) linked to specific incoherence patterns.

Coherence Risk Surface Map

A qualitative map identifying the conflict zones, amplification paths, and feedback loops between control layers. Shows *where* and *how* risks emerge.

Auditability Gap Report

Characterizes the inability to reconstruct control decisions with current tooling, identifying governance blind spots and compliance exposure.

Intended Consumers

- Architecture Boards
- Platform Engineering
- MLOps/SRE
- Procurement & Risk Management

Three Core Observations on Runtime Control Coherence

1. **Control coherence is a decisive structural property.** It determines whether allocated resources translate into predictable, efficient outcomes.

2. **Classical metrics and over-provisioning are deceptive.** They do not resolve coherence risks and often obscure the true economic cost drivers.

3. **A structural perspective makes these risks visible.** It enables decision-relevant diagnosis without requiring system access, implementation changes, or vendor commitments.

This is a diagnostic instrument, not a solution proposal—designed to enable clarity and informed action.