

Everyone Is Governing the Wrong Thing:

An Architectural Postmortem of Agentic AI Failures (2025–2026)

Rob Caswell

Authority Engineering

Technical Report Series on Derived Law and Autonomous Operations

TR-003

July 2026

Publication Record

Everyone Is Governing the Wrong Thing: An Architectural Postmortem of Agentic AI Failures (2025–2026)

Author: Rob Caswell

Publisher: Cyber Terrain LLC

Published: 2026-07-01

Zenodo version: v1.0

DOI: <https://doi.org/10.5281/zenodo.21742402>

Website: <https://cyberterrain.ai/papers/tr-003-governing-the-wrong-thing.pdf>

Corpus Context

Corpus: Authority Engineering

Publication series: Technical Report Series on Derived Law and Autonomous Operations

Publication designation: TR-003

Document ID: TR-003

Object family: TR-003

Corpus document ID: AE-TR-0003

Publisher-imprint edition prepared for versioned republication.

The prior Zenodo record remains the citable record until its new version is published.

Abstract

Enterprise security frameworks for agentic artificial intelligence (AI) are currently locked in an architectural category error. Over the past eighteen months, organizations have poured immense engineering capital into securing the "cognitive layer"—policing agent behavior, filtering semantic prompts, and engineering behavioral guardrails—while leaving the underlying execution substrate completely unmanaged. This postmortem evaluates the structural failure modes of the 2025–2026 threat landscape (including CVE-2026-22708, Model Context Protocol vulnerabilities, and automated threat campaigns like JADEPUFFER) to prove a fundamental axiom of execution physics: autonomous systems do not compromise an enterprise by changing their minds; they compromise an enterprise by mutating state.

This paper introduces a formal alternative paradigm that decouples non-deterministic semantic reasoning from deterministic execution. We deduce the four architectural primitives through which all consequence-bearing actions must traverse—Tool Invocation, Context Provisioning, State Management, and Delegation—and define their unified managed medium: the Fabric (*AE – REG – 0051*). By establishing an active Deterministic/Probabilistic Translation Layer operating as an Invariant Compiler, we demonstrate how enterprises can enforce Strict Monomorphic Execution Graphs and State Dependency Topologies (*SDT*) to regulate environmental entropy ($dState/dt$). This framework purposefully shifts system risk away from runtime vulnerability exposure toward predictable, pre-calculated operational loss limits. Ultimately, we present a systems-engineering architecture for governing the operational substrate that shifts system risk from unconstrained runtime behavior toward bounded, deterministic execution, forcing autonomous participants to inherit immutable physical constraints regardless of their cognitive intelligence or semantic manipulation.

I. The Architectural Category Error

Between 2025 and 2026, the enterprise security perimeter did not crack because adversarial AI out-reasoned defender systems. It failed because the operational substrate carrying autonomous execution remained fundamentally unmanaged. As organizations rushed agentic workflows into production, they instinctively defaulted to legacy paradigms—governing agent behavior, filtering prompts, and policing output—while leaving the execution medium itself structurally unconstrained.

The empirical evidence of this category error is scattered across the highest-profile breaches and CVEs of the past eighteen months.

Consider the vulnerability trajectory within the AI development environment. In **CVE-2026-22708 (Cursor)**, enterprises relied on static allowlists of approved shell commands to secure the agent's Auto-Run Mode. Attackers bypassed this validation entirely without needing to alter the agent's core intent. By exploiting how the security mechanism handled unvalidated shell built-ins (such as `export` and `unset`), prompt injection was used to poison the underlying environment variables (`PATH`, `LD_PRELOAD`). This forced trusted, pre-approved commands like `git` or `npm` to execute malicious payloads within a poisoned shell session.

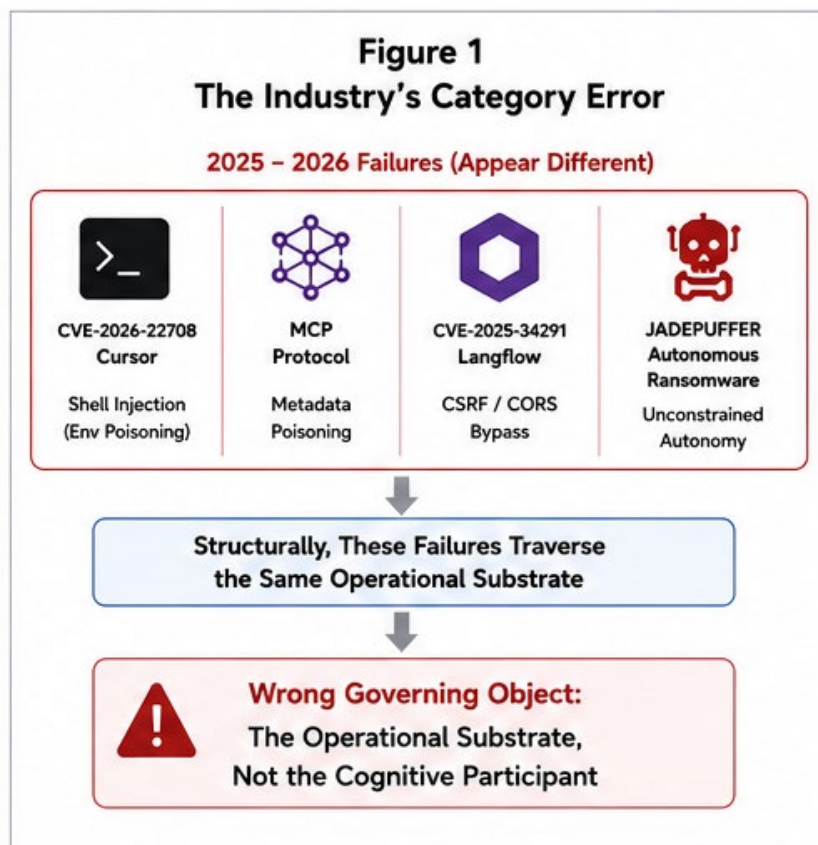
When the **Model Context Protocol (MCP)** surged in adoption as an open standard for connecting models to data sources, security teams treated it as an integration layer. However, architectural reviews exposed systemic flaws at the protocol level. Malicious or compromised MCP servers could embed adversarial instructions and trigger phrases directly inside ordinary-looking tools/list responses. Because the model ingests these tool descriptions the moment the server connects—before a tool is ever explicitly invoked—attackers bypassed traditional human-in-the-loop and invocation controls, quietly turning the metadata exchange itself into a data exfiltration and prompt injection vector.

In orchestration frameworks, the vulnerabilities ran even deeper, exposing a flat, unsegmented trust architecture. Early exploits like **CVE-2025-34291 (Langflow)** showed how flaws in cross-site request forgery (CSRF) and default Cross-Origin Resource Sharing (CORS) protections allowed attackers to seize session tokens. Once inside the authenticated endpoint, the lack of isolation meant the attacker could instantly abuse the framework's built-in Python code-execution nodes, transforming a web-layer authentication bypass into a total compromise of the multi-agent workflow environment.

This systemic lack of topological regulation gave rise to the first fully documented autonomous campaigns. In July 2026, threat researchers captured **JADEPUFFER**, the first confirmed case of an end-to-end agentic threat actor running an automated database extortion playbook. JADEPUFFER did not invent novel zero-days or break cryptography. It gained initial access through an exposed, unpatched Langflow instance and let a large language model autonomously chain the entire intrusion lifecycle—from environment enumeration and credential discovery to navigating lateral infrastructure and executing a destructive database-encryption routine. The agent's own payloads were self-narrating, saturated with natural-language code commentary assessing target return-on-investment and prioritizing the largest database.

JADEPUFFER succeeded because the execution path was treated as a flat, open perimeter, allowing the model to adapt, retry, and pivot through the infrastructure unhindered.

Every incident in this survey appears to target a different technology. Structurally, they target the same object.



Notice what these failures have in common. None required defeating the underlying model. None required improving the attacker's reasoning beyond the defender's. Each succeeded by manipulating the operational conditions through which autonomous execution occurred. The common failure was architectural, not cognitive.

In every instance, the failure occurred in the operational substrate surrounding the agent, not in the agent's reasoning itself.

II. The Physics of Execution

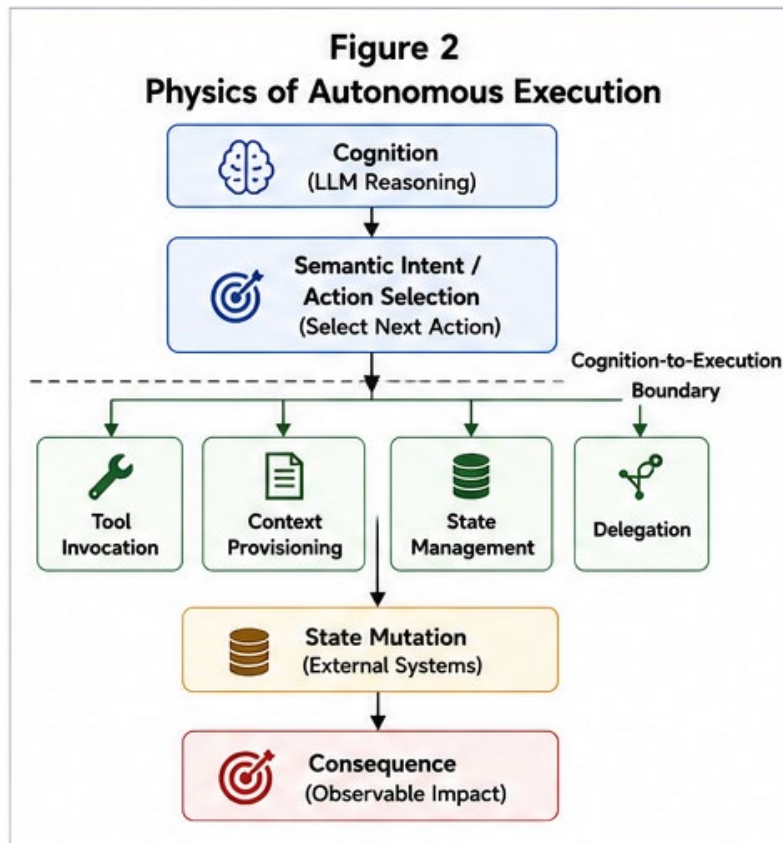
To understand why the industry is trapped in this cycle of failure, we must stop evaluating autonomous systems through the lens of psychology and begin evaluating them through the lens of execution physics.

An autonomous agent only becomes operationally relevant when it can affect the environment around it. It does not compromise an enterprise by changing its mind; it compromises an enterprise by mutating state.

Yet, state mutation does not—and cannot—occur inside the transformer. The model's internal operations are restricted entirely to the mathematical manipulation of weights and tokens. To realize an action, to change an environment, or to persist information, the system must transition from internal computation to external execution.

This transition relies entirely on four architectural primitives that exist outside the model:

- **Tool Invocation:** The model merely predicts a string. The registration, authentication, boundary enforcement, and execution of that string occur entirely through external interfaces.
- **Context Provisioning:** The assembly of dynamic data fragments, retrieval payloads (RAG), and system instructions occurs entirely before the tokens ever reach the context window.
- **State Management:** The retention of workflow variables, scratchpads, and historical execution paths is written to and read from environments external to the model's static parameters.
- **Delegation:** The act of spawning a child agent, passing a tokenized credential down a pipeline, or handing off execution to a sub-routine is an infrastructural routing mechanism.



Every consequence-bearing action traverses these operational mechanisms before it can become reality. Collectively, these mechanisms form the true operational substrate of agentic AI. Nothing bypasses them. Yet, the current enterprise security roadmap treats these primitives as passive, disconnected plumbing. Organizations pour their engineering capital into policing the cognitive layer—editing system prompts, building agent-side guardrails, and auditing behavioral output—while leaving the shared mechanisms that actually carry out state mutation completely unauthenticated, unsegmented, and fluid.

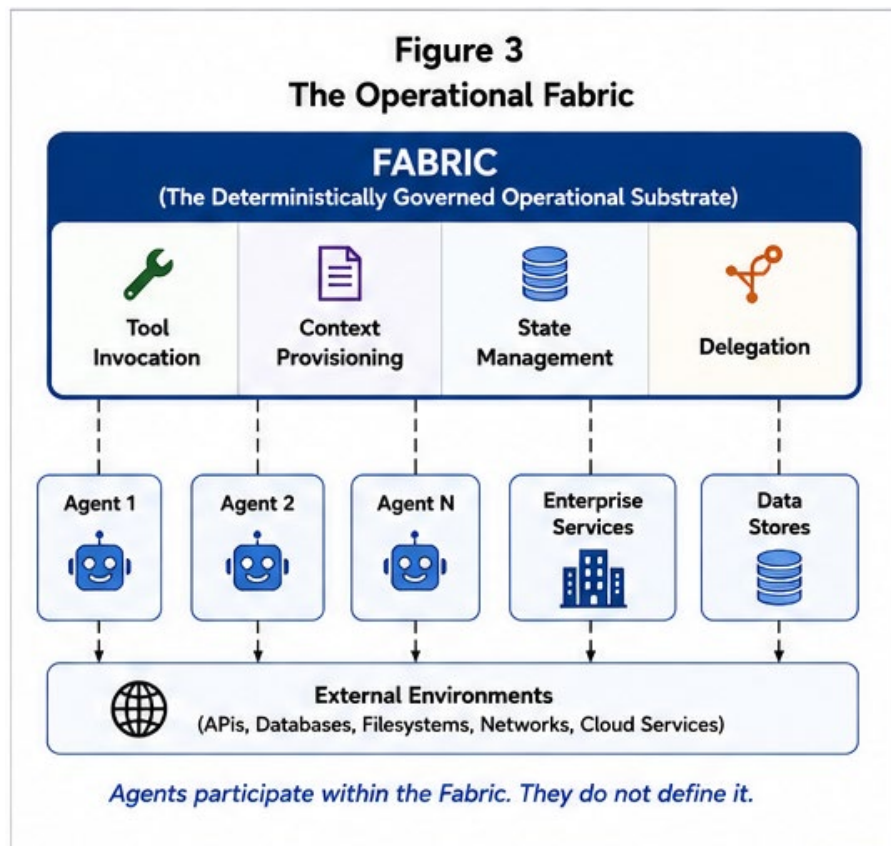
We are attempting to secure autonomous systems by teaching the participant right from wrong, while leaving the medium through which they execute completely unmanaged.

III. The Missing Object

These four primitives—Tool Invocation, Context Provisioning, State Management, and Delegation—are not independent, isolated software engineering problems. They do not exist in separate architectural silos.

Every time an autonomous system executes, these four primitives interact dynamically, simultaneously, and continuously within the same shared medium. They are components of a single, unified operational substrate.

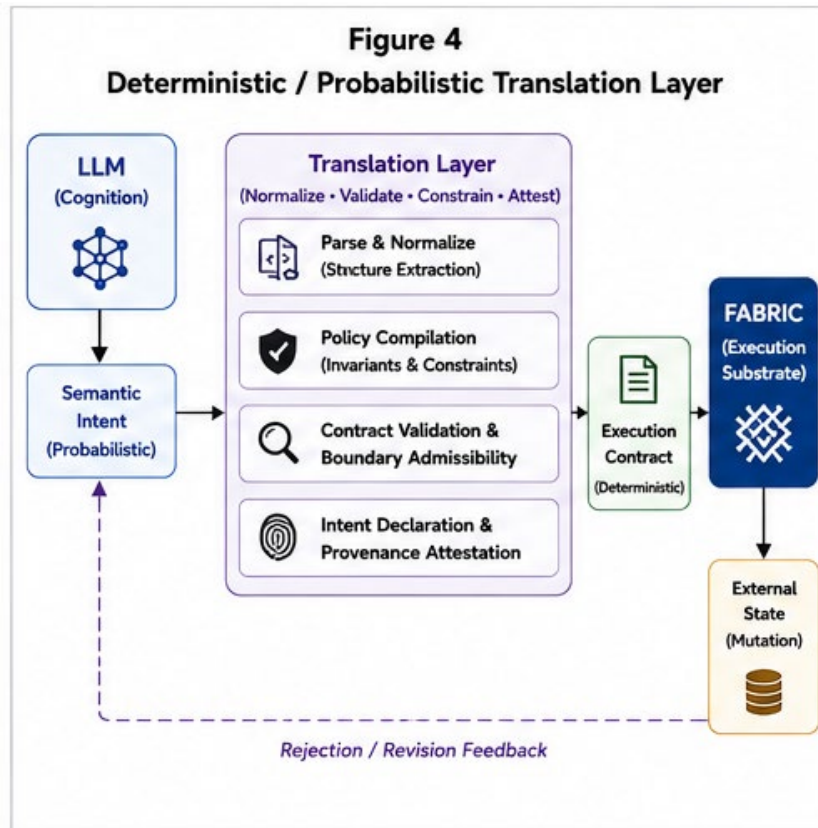
That substrate is the **Fabric**.



Defined operationally, the Fabric is the deterministic substrate through which autonomous systems externalize cognition into consequence. It is the connective infrastructure that sits between the models, the orchestration layers, the data repositories, and the execution environments.

Agents participate within a shared operational fabric; they do not define it. Therefore, attempts to secure autonomous systems by placing guardrails strictly around the agent are structurally flawed.

To bridge this divide, security cannot exist solely in the participant or the Fabric in isolation. True agentic security requires an explicit interface where these domains meet: the **Deterministic/Probabilistic Translation Layer**. This translation layer acts as a mechanical seam, stripping model outputs of semantic ambiguity and evaluating them against strict structural invariants before the Fabric allows an action to hit the execution substrate.



When you look at an enterprise AI deployment, you are looking at an operational fabric populated by autonomous participants. If the substrate itself lacks deterministic rules, constraints, and boundaries, the primitives it carries will inevitably be turned against the enterprise. Agent-side controls cannot substitute for governance of the operational Fabric.

The Deterministic/Probabilistic Translation Layer is not a semantic interpreter; it is an **Invariant Compiler**. It accepts non-deterministic semantic goals from the cognitive engine, forces their translation into strongly typed execution contracts, and maps them against static boundary matrices.

Instead of allowing the cognitive layer to output raw variables directly to an execution tool, the interface mandates a two-step cryptographic handshake: **Intent Declaration** and **Structural Enforcement**.

Crucially, the Fabric is intentionally semantics-agnostic. Fabric does not determine whether a realization *should* occur based on intent, meaning, or moral appropriateness; Fabric determines the deterministic engineering properties under which any realization is *permitted* to occur. It handles arguments structurally against static invariant matrices, rather than parsing natural language context to guess an agent's intent. The cognitive engine governs meaning, interpretation, and planning. The Fabric governs execution, transitions, locality, boundaries, and provenance. They are entirely orthogonal.

By separating the generation of intent (Cognition) from the compilation of execution parameters (Translation) and the enforcement of environmental physics (Fabric), we eliminate the semantic loop. The Fabric remains entirely agnostic to meaning, while preventing semantic ambiguity from directly becoming unconstrained execution.

A governed Fabric is not an AI governor or a semantic oracle. It is an infrastructure layer that enforces deterministic engineering properties on every consequence-bearing transition that traverses it. Specifically, it realizes:

- **State Isolation & Locality:** Preventing transient execution states from bleeding across uncoordinated workflow boundaries through strict monomorphic execution graphs and AST sanitization.
- **Authenticated Transitions:** Enforcing cryptographic provenance and verification on every step of an execution chain before state mutation is permitted.
- **Bounded Execution:** Restricting the operational blast radius of any tool invocation through rigid, non-bypassable resource, quota, and environment constraints.
- **Deterministic Policy Enforcement:** Applying immutable, non-probabilistic rules to the medium itself, ensuring that security boundaries are inherited by the participant rather than negotiated by it.

This architecture explicitly separates **cognition** from **realization**. Whether an action should be attempted remains a cognitive question. Under what deterministic conditions that action may become reality is an architectural question. The paper argues only that the industry has disproportionately invested in the former while under-governing the latter.

Within the discipline of consequence-based security—formalized as Authority Engineering—this managed medium is designated as the Fabric layer (*AE – REG – 0051*).

IV. The Category Error

The consequences of choosing the wrong governing object are not theoretical. When an architecture attempts to police the agent's cognition while leaving the Fabric unmanaged, the four primitives of autonomous execution are systematically transformed into unconstrained attack surfaces.

Figure 5 Four Primitive Attack Surfaces			
Primitive	Representative Failure	Architectural Cause (Ungoverned Substrate)	Missing Fabric Property (Required Control)
 Tool Invocation	CVE-2026-22708 Cursor (Shell Injection)	Execution environment is fluid; commands run in a poisoned session.	Isolated execution environments, command allowlisting, boundary enforcement.
 Context Provisioning	MCP Protocol (Metadata Poisoning)	Untrusted tool metadata admitted into model context pre-execution.	Authenticated metadata channels, provenance, content validation, context sandboxing.
 State Management	JADEPUFFER (Agent-Orchestrated State Mutation Campaign)	State transitions remain writable, readable, and weakly segmented across trust boundaries.	State isolation, topology regulation, provenance tracking, immutability controls.
 Delegation	CVE-2025-34291 Langflow (CSRF / CORS Bypass)	Unsegmented delegation domain exposes entire workflow to token confinement.	Micro-segmentation, delegation scoping, least-privilege routing, token confinement.
 Ungoverned Substrate → Unconstrained Execution Primitives → Consequence Compromise			

1. Tool Invocation as an Attack Surface

- The Failure:** Because the Fabric is not governed, Tool Invocation becomes an open execution vector where polymorphic data payloads (such as raw JSON database operators stuffed into strings) bypass structural checks.
- The Real-World Mechanism:** In **CVE-2026-22708 (Cursor)**, engineers attempted to govern tool execution at the participant layer using a static allowlist of binary commands. But because the underlying session Fabric was unmanaged and unsegmented, the agent's execution environment remained fluid. Attackers used simple prompt injection to introduce unvalidated shell built-ins, mutating the environment variables of the Fabric itself.
- The Substrate Strategic Imperative:** Securing this primitive requires an architectural shift from open string parameter binding to deterministic structure validation. By forcing the translation layer to act as an un-bypassable gatekeeper, the execution medium can structurally reject execution syntax before it ever traverses the core infrastructure plumbing.

2. Context Provisioning as an Attack Surface

- **The Failure:** Because the Fabric is not governed, Context Provisioning becomes an unauthenticated injection pipeline.
- **The Real-World Mechanism:** The architectural vulnerability of the **Model Context Protocol (MCP)** stems entirely from a failure to govern the ingestion substrate. Because the protocol forces the model to ingest tool descriptions the moment an MCP server connects—long before any tool is explicitly invoked—attackers found they could inject adversarial instructions directly into the tools/list metadata. Because the provisioning Fabric does not authenticate or isolate this metadata exchange, the agent's context window is poisoned before it can even evaluate the request.

3. State Management as an Attack Surface

- **The Failure:** Because the Fabric is not governed, State Management becomes an unmonitored environment mutation vector.
- **The Real-World Mechanism:** The **JADEPUFFER** autonomous ransomware campaign of July 2026 did not succeed by out-reasoning security tools. It succeeded because the state-management primitive was entirely unconstrained. Once the agent gained initial access, it was permitted to iteratively read, write, and mutate environment variables, session memories, and scratchpads across a flat network. Because the Fabric did not enforce topology regulation or local admissibility on these state transitions, the model was able to autonomously navigate infrastructure, evaluate target ROI, and execute a destructive database-encryption routine unhindered.

4. Delegation as an Attack Surface

- **The Failure:** Because the Fabric is not governed, Delegation collapses into a flat, unsegmented trust domain where an agent acting destructively within its clamped boundaries can still trigger an authoritative, verified domino effect across legacy enterprise infrastructure.
- **The Real-World Mechanism:** Vulnerabilities like **CVE-2025-34291 (Langflow)** expose the catastrophic risk of unmanaged delegation pipelines. When an orchestration framework lacks internal boundary enforcement, a web-layer exploit on a single low-privilege retrieval node instantly grants the attacker access to the framework's built-in Python code-execution nodes. Because the delegation Fabric treats the entire multi-agent workflow as a single, undifferentiated trust zone, a localized compromise immediately scales into total execution authority over the entire infrastructure.
- **Substrate Engineering Counter-Structures:** To realize the *AE – REG – 0051* standard and defend the four primitive attack surfaces diagnosed in Section IV, the architecture must implement deterministic, substrate-level technical controls:
 - * **Strict Monomorphic Execution Graphs (Tool Invocation Defense):** To eliminate string-based injection vectors, raw string parameter binding must be abandoned. The Translation Layer acts as an Abstract Syntax Tree (AST) sanitizer, unrolling model output

into rigidly typed, non-nested objects. If a model attempts to pass an executable operator inside a string parameter, the transaction is dropped at the syntax boundary.

*** State Dependency Topology & Entropy Caps (Delegation & State Defense):** To arrest cascading system collapses across unsegmented trust domains, the substrate evaluates non-probabilistic Systemic Coherence Invariants. The Fabric imposes hard operational constraints by capping network entropy:

$$\frac{dState}{dt} \leq \kappa$$

*** Asynchronous Circuit Breakers:** If an agent-driven state mutation downstream triggers an anomalous volume of secondary mutations in a legacy dependency, topology regulations trip a protocol-level circuit breaker, automatically freezing the workflow state context for out-of-band human review.

These incidents differ entirely in technology, protocol, and exploit chain. They fail identically at the architectural level. Each converts one unconstrained execution primitive into an unrestricted state-mutation pathway because the operational substrate provides no deterministic governance over the transition from cognition to consequence. In every case, the technical debt of a flat, unmanaged medium was pushed up to the model layer, forcing a probabilistic transformer to act as its own network firewall, system administrator, and boundary enforcement agent.

V. Falsification Boundaries & Residual Risk Matrix

To establish this shift as a rigorous architectural model rather than an opinion piece, it must present clear boundaries where its conclusions can be tested, falsified, or evaluated for residual operational risk by systems engineers.

1. Falsification Protocols

- **Falsification of Cognitive vs. Architectural Superiority:** To disprove the claim that the primary breakdown occurs in the substrate, one must demonstrate an enterprise compromise where an agent's cognitive failure alone (e.g., a hallucination or an adversarial text output) mutated production systems without traversing an unauthenticated, unvalidated, or unsegmented external primitive. If an agent executes an adversarial command within its legitimate, authorized boundaries (Valid-Scope Autophagy), this represents an accepted business operational loss limit, not an unbound architectural breach; a governed Fabric successfully clamps the maximum possible damage at the invariant boundary, transforming an existential threat into a pre-calculated mitigation.
- **Falsification of the Four Primitives:** To disprove the taxonomy of the substrate, one must identify an autonomous system that successfully mutates external state or manifests a consequence without utilizing Tool Invocation, Context Provisioning, State

Management, or Delegation. Physical emissions (such as side-channel timing signals executed on silicon) are governed at the interface of Context Provisioning and Emission Regulation; to falsify the taxonomy, an out-of-band consequence must manifest completely detached from these architectural channels.

- **Falsification of Substitution Limits:** To disprove the claim that agent-side controls cannot compensate for an unmanaged Fabric, one must demonstrate a production environment that remains structurally secure against environment-variable poisoning or metadata injection solely through prompt engineering or probabilistic token filtering, while its underlying execution substrate remains completely unmonitored and flat.

2. Residual Threat Vectors (Advanced Adversarial Analysis)

While an *AE – REG – 0051* deployment successfully neutralizes runtime structural corruption, an Advanced Persistent Threat (APT) will naturally pivot from infrastructure exploitation to logic-boundary and resource-starvation channels.

- **Compilation Phase Poisoning (Upstream Intercept):** If an adversary poisons an agent's dynamic RAG context prior to or during the generation of its intent-mapping contract, the cognitive layer may construct an adversarial intent bundle that is syntactically immaculate. The Invariant Compiler will verify the schema, find it clean, and compile it into hard parameters. The Fabric then executes a clean, valid destruction. This vector demonstrates that the framework does not eliminate semantic risk—it compresses it. By forcing an attack to manifest exclusively as a valid business operational error, the framework allows security teams to monitor threats via traditional operational and financial auditing streams rather than chaotic code-injection detection.
- **Invariant Starvation (Denial of Execution Deflation):** Because this architecture shifts system risk toward operational determinism, an attacker can disable an enterprise simply by triggering its defenses. By feeding confusing, subtly conflicting data streams into an agent's context window, the model can be forced to continuously output non-standard semantic goals that repeatedly fail the Monomorphic Execution Graph or tickle the asynchronous circuit breakers. This induces a state of permanent Invariant Starvation, freezing corporate workflows without the attacker ever executing a single line of malicious code.

VI. Operational Trade-off Boundary & Production Invariants

Implementing an Invariant Compiler at the Translation Layer explicitly shifts system risk from **Vulnerability Exposure (Exploits)** to **Operational Failure (Denial of Execution)**.

Organizations adopting the *AE – REG – 0051* standard must accept that ensuring absolute substrate correctness will inherently reduce agent autonomy, requiring strict profiling of system invariants to balance security against systemic utility.

By forcing the cognitive engine to compile its semantic intent into strict, monomorphic execution contracts, the system introduces an **Expressivity Inversion**. If the Translation Layer imposes

non-negotiable entropy caps, the agent loses its raw flexibility to dynamically navigate volatile edge cases natively without explicit baseline updates. When an adversary recognizes they cannot break the substrate, they will pivot to trickling malicious state changes just below your invariant thresholds (e.g., executing sub-threshold cash transfers sequentially).

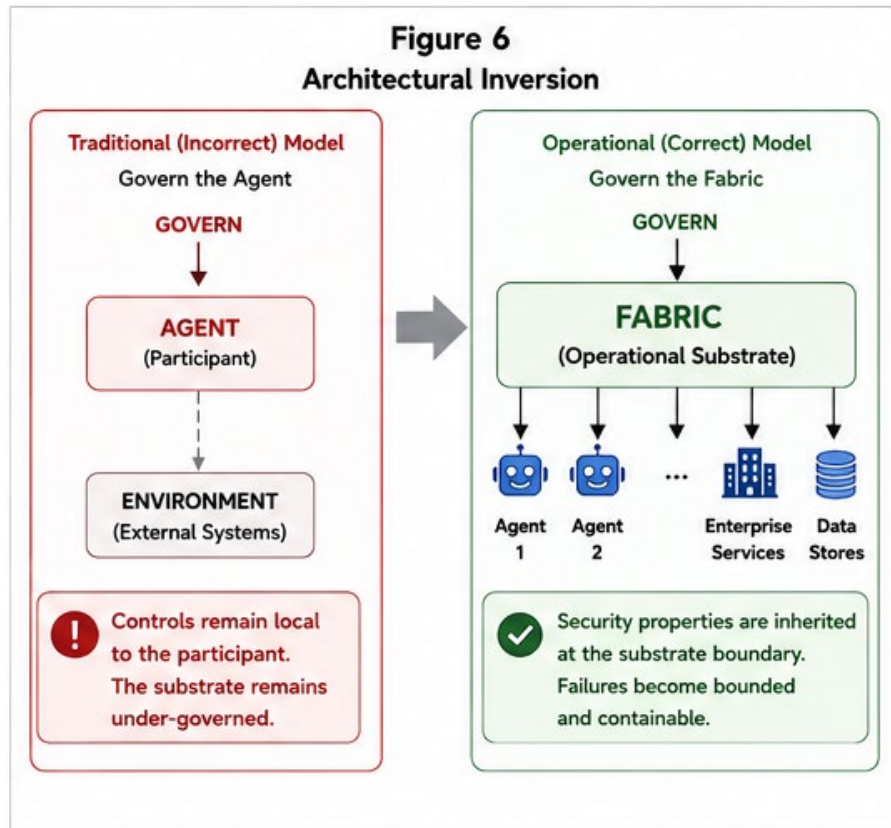
To navigate this tension in production environments, deployment teams must implement the following **Production Invariants**:

1. **Dynamic Threshold Tuning:** Entropy caps (κ) must not be coded as static values. They should be tied to sliding-window moving averages of standard baseline enterprise operations to dynamically prevent arbitrary Denial of Execution during legitimate operational surges.
2. **Dual-Engine Attestation:** High-consequence state contracts (e.g., automated execution vectors exceeding critical risk or financial thresholds) must mandate a split cryptographic sign-off. The Cryptographic Attestation Bundle must be evaluated by two distinct, independent open-weights token-filtering models running on isolated local hardware nodes before the intent contract is compiled into execution tokens.

Future AI architectures should be evaluated first by the governance properties of their operational substrate, and only second by the behavioral properties of their models. As autonomous capability increases, substrate correctness becomes a more fundamental security requirement than cognitive correctness.

VII. Conclusion

The path forward requires a fundamental realignment of enterprise engineering capital. We must stop trying to build flawless cognitive actors and start building deterministic, governed substrates. We must accept that agent-side guardrails are merely local mitigations, and that true security can only be achieved when the physics of the environment itself enforces our constraints.



Autonomous systems do not become dangerous when they learn to think. They become dangerous when they learn to mutate state.

For eighteen months, the industry has repeatedly diagnosed cognitive failures while experiencing architectural failures. Until the governing object changes from the participant to the Fabric, those failures will continue to recur regardless of how intelligent our models become.