Nick Clark · Published April 25, 2026 · PDF

How to Make Policy and Lineage Travel Across Every Layer of an AI System

If you run agents across execution, indexing, transport, identity, schema, memory, and governance tiers, your policy and audit trail usually get re-stitched at every boundary, and the seams are where identity and attribution break. This guide describes an architectural approach in which those properties are carried by the agent object itself so they stay consistent end to end. It describes an architecture disclosed in United States Patent Application 19/647,395, not a shipping library, and it centers on the Cross-Patent Architecture inventive step.

What You Are Building

You are building an AI system whose governance policy, identity, and audit trail do not stop at a layer boundary. In most deployments an agent's work touches many distinct tiers: an execution substrate that runs it, an index it searches, a transport layer that moves it between hosts, an identity system that authenticates it, a schema that defines its object shape, a memory store that persists its state, and a governance layer that constrains what it may do. The goal here is a single governed architecture in which the fields that define who an agent is, what policy binds it, and what it has done travel across all of those tiers together, so that identity, policy, and attribution are consistent from one end of the system to the other instead of being reassembled at each hop.

The problem is familiar to anyone who has tried to answer "why did this agent do that, under whose authority, and can I prove it" across a distributed system. If the answer lives in seven different logs owned by seven different subsystems, you cannot answer it. This guide teaches an architectural approach to making that answer intrinsic to the agent itself.

The underlying method is disclosed in United States Patent Application 19/647,395, and its home concept is the Cross-Patent Architecture inventive step: the same governed fields converge across tiers that are usually designed and secured independently.

Why the Obvious Approaches Fall Short

The conventional way to govern a multi-tier AI system is to put a control at each boundary. A safety wrapper filters outputs after inference. An identity system asserts a credential at a checkpoint. A memory store persists state in an external database. An orchestrator schedules the agent as a session-bound process. Each of these is a real, reasonable mechanism, and each works within its own tier.

The structural gap appears between the tiers. When governance is a post-inference filter, it sits outside the agent's own state and cannot bind decisions made earlier in the pipeline. When memory lives in an external vector store or platform session, the agent object that migrates or resumes is not the thing that carries the history, so continuity depends on the store being reachable and correctly re-associated. When identity is a static credential presented at a checkpoint, the system cannot distinguish an authorized holder from anyone who possesses the credential, and it asserts identity only at discrete points rather than maintaining it across the interaction. As the spec notes, conventional agent frameworks treat agents as session-bound processes coordinated by external schedulers, with memory maintained outside the agent object and governance applied as a post-inference filter or a system-prompt instruction.

The result is that policy and provenance are stitched together per layer. Every boundary is a place where the audit trail can diverge, where a policy in one tier may not match the policy in force in another, and where identity is re-asserted rather than continuously maintained. You can add more checkpoints, but each one is another seam.

The Architecture

The disclosed approach removes the seams by making the governed properties fields of the agent object itself, and by having those fields participate in every tier through one shared mechanism. The spec describes a semantic agent schema of canonical fields: intent, context, memory, a policy reference field, a mutation descriptor, and a lineage field, extended with cognitive-domain fields such as affective state, integrity, confidence, and capability. Three of those elements do the cross-layer work.

A single lineage chain. The lineage field records the agent's complete behavioral history as one chain. The disclosure requires that cognitive-domain field changes are written into the same lineage chain as all other agent state transitions, so there is one audit record, not one per subsystem. The spec states that the complete behavioral trajectory of the agent is deterministically reconstructible from the lineage field alone. That single-chain property is what lets attribution survive a hop from one tier to another.

Embedded, policy-bounded state. The policy reference field travels with the agent, and every mutation, including mutations to the cognitive-domain fields, is subject to policy validation through the governance mechanisms rather than being applied and filtered afterward. Governance is described as a governance input that participates in a composite admissibility evaluation, not a wrapper applied after generation. Because the policy binding is carried by the object, the same constraints apply wherever the object runs.

Fields that are carried, not reconstructed. The recurring design rule across tiers is that the substrate validates the agent but does not own its state. On the execution substrate, the spec is explicit that the substrate validates proposed transitions but does not retain authority over the agent's cognitive state, because those fields are carried by the agent itself. This is what makes the properties portable.

The Cross-Patent Architecture inventive step is that these fields converge across tiers that are normally separate:

- **Execution.** The agent carries its governed fields onto whatever substrate hosts it; the substrate evaluates transitions but does not hold the state.
- **Schema.** The foundational fields and the cognitive-domain fields are one object; the cognitive fields write their updates into the foundational lineage field, so schema and audit are unified.
- **Memory.** State is persistent and memory-resident so it survives asynchronous execution intervals, which is what lets an agent suspend and resume without losing its history or its policy binding.
- **Transport.** When an agent migrates between substrates, its governed fields travel with it intact; the spec describes a transit state that freezes field values while the lineage field continues to accumulate transit events such as departure timestamp, transport path, and arrival validation, and the destination validates lineage continuity rather than rebuilding the state.
- **Identity.** Identity is established through trust-slope continuity, accumulated observations over time, rather than a static credential matched at a checkpoint, so identity is a maintained trajectory recorded in lineage rather than a point assertion.
- **Indexing.** The index is used not only as a content-resolution substrate but as an execution substrate for discovery agents that carry their own governance and identity and record each traversal step in their own lineage, so searching the index is itself a governed, attributed act.

- **Governance.** Cryptographically signed policy constraints apply to all agent fields, and the disclosure describes cross-tier interactions, for example evaluating policy freshness on resume, and reducing execution readiness when a hosting substrate's identity continuity fails, so that governance state is coherent across the boundaries rather than per tier.

The through-line is that one set of governed fields, one lineage chain, and one policy binding are present at every tier, which is precisely what conventional per-layer stitching cannot provide.

How to Approach the Build

You are implementing this yourself. The following ordered steps mirror the architecture; the sketches below are illustrative and faithful to the disclosure, not a package you can install.

1. **Define the agent object schema first.** Make identity, the policy reference, memory, and lineage intrinsic typed fields of the object, not external tables. An illustrative shape:

```
Agent {
  intent, context, memory
  policyRef            // governance binding travels with the object
  mutationDescriptor   // how the last transition changed state
  lineage              // append-only chain; single source of history
  // cognitive-domain fields (integrity, confidence, capability, ...)
}
```

   Everything else depends on this being the canonical unit.

2. **Make lineage the single audit chain.** Route every state change through one append-only lineage record, and require cognitive/domain field updates to write into that same chain. Your acceptance test is the spec's own property: you should be able to reconstruct the agent's full behavioral trajectory from lineage alone.
3. **Bind policy at mutation time, not after inference.** Evaluate every proposed mutation against the carried policy reference before it commits, so a rejected transition never happens rather than being filtered from output. Treat governance as an admissibility input to the transition, not a downstream scrubber.
4. **Give substrates a validate-only contract.** Design each execution host to check a proposed transition and confirm lineage continuity, but never to own or mutate the agent's governed state. If a host stores authoritative state, you have reintroduced a seam.
5. **Carry fields across transport, do not rebuild them.** On migration, serialize the whole governed object, keep field values frozen in transit, append transit events to lineage, and have the destination validate lineage continuity on arrival. Never reconstruct identity or policy at the destination from scratch.
6. **Model identity as a continuity trajectory.** Accumulate identity observations over time into a trust-slope chain recorded in lineage, rather than matching a static credential at a checkpoint. This lets identity be verified anywhere the object goes, using the same chain.
7. **Extend the same contract to your index and any new tier.** When you add a subsystem, make the agent traverse or act on it as the same governed object that records its steps in the same lineage, rather than granting the new tier its own separate policy and log.

The recurring tradeoff to weigh: carrying governed state inside the object increases object size and serialization cost, and it forces every tier to honor the same validation contract. You accept that overhead in exchange for end-to-end coherence. If you cut a

corner and let one tier own authoritative state or apply its own after-the-fact filter, that tier becomes the seam the architecture exists to remove.

What This Does Not Give You

This is an architecture, not a drop-in library. There is no package to install and no code here that "just works." The sketches are illustrative of the disclosed design; you implement the schema, the lineage chain, the policy-binding logic, the transport contract, and the identity trajectory yourself, in your own stack.

It is not benchmarked or productized. The approach is disclosed in a patent filing; this guide does not assert performance numbers, latency figures, or production results, because the disclosure does not establish them and neither should you until you measure your own build.

It does not eliminate the need for correct policy authoring, key management, or the underlying cryptographic and transport primitives, which are referenced in the disclosure as mechanisms the architecture builds on. It carries policy and lineage coherently; it does not decide what your policy should be.

Finally, it does not fit every system. If your agents are stateless single-shot calls that never migrate, never resume, and never cross a trust boundary, the per-layer stitching that this architecture removes may not be a cost you are paying, and a lighter approach may serve you better. The value here is proportional to how many tiers your agents actually cross.

Disclosure Scope

The architectural approach described in this guide is disclosed in United States Patent Application 19/647,395, "Systems and Methods for Autonomous Agents with Persistent Cognitive State, Self-Regulated Execution, and Cross-Domain Behavioral Coherence."

This guide is educational. It explains an approach a developer can build and does not constitute a warranty, a guarantee of results, or an offer of software, a package, or a service. Every mechanism described above is drawn from that filing; where the filing does not state a parameter, guarantee, or result, this guide does not claim one.

Cross-Patent Architecture (/cross-patent-architecture)

All 40 steps → (/inventive-steps)

Cross-cutting architectural principles that compose every primitive into a coherent platform.

Chapter 1 (/patents/19-647395/chapters/foundation)

PRIMARY TECHNICAL DISCLOSURE

- Cross-Patent Architecture, Articles (/articles/cross-patent-architecture)

SECONDARY TECHNICAL

- Transit Cognitive State (/articles/cross-patent-architecture/transit-cognitive-state)
- Substrate Identity Revocation During Active Cognition (/articles/cross-patent-architecture/substrate-identity-revocation)
- Policy Freshness Across Asynchronous Execution (/articles/cross-patent-architecture/policy-freshness-asynchronous-execution)
- Governance Authority Evaluation via Integrity Trajectory (/articles/cross-patent-architecture/governance-authority-integrity-trajectory)
- Discovery Agent as Schema-Conformant Index Traverser (/articles/cross-patent-architecture/discovery-agent-schema-index-traverser)
- Unified Substrate for Governed Information Acquisition (/articles/cross-patent-architecture/cross-tier-navigation-world-as-model)

APPLICATIONS · GENERAL

- One Governed Platform, Not Four Integrated Systems: A Unified Architecture Spine for Agent Execution, Cognition, Content, and Spatial Tiers (/articles/cross-patent-architecture/unified-governed-platform)

- World-as-Model Systems: Navigating the Physical World, Cognition, and Discovery as One Governed Model (/articles/cross-patent-architecture/world-as-model-systems)
- End-to-End Lineage and Audit: Reconstructing Any Agent Action Across Every Tier of the Stack (/articles/cross-patent-architecture/end-to-end-lineage-and-audit)
- Moving Governed AI Agents Across Clouds and Vendors Without Losing Identity: Substrate Portability via the Cross-Patent Architecture (/articles/cross-patent-architecture/portability-across-substrates)
- Cross-Patent Architecture: Why a Coherent AI Platform Needs a Shared Governance Authority at the Foundation, Not as a Feature (/articles/cross-patent-architecture/ai-platform-foundation)
- Regulated Cross-Domain Deployment: One Governance Authority and Policy-Freshness Model Across Every Tier of an End-to-End System (/articles/cross-patent-architecture/regulated-cross-domain-deployment)

APPLICATIONS · SPECIFIC

- Palantir Foundry and AIP (the ontology-based data/operations platform plus its AI orchestration layer) vs a cross-tier governed architecture: where does end-to-end action attribution live? (/articles/cross-patent-architecture/palantir-foundry-aip)
- Microsoft's integrated AI stack (Azure AI Foundry, Microsoft Fabric, Entra, and Copilot) vs a single cross-domain governance architecture: how do coherence and one governance chain differ from an integrated product suite? (/articles/cross-patent-architecture/microsoft-ai-stack)
- Amazon Web Services' integrated AI/data stack (Bedrock, SageMaker, and surrounding data/identity services) vs a unified cross-tier governed agent architecture (/articles/cross-patent-architecture/aws-ai-stack)
- NVIDIA's full-stack AI platform (NVIDIA AI Enterprise, NIM microservices, and the CUDA/hardware-to-software stack) vs a substrate-independent governance architecture (/articles/cross-patent-architecture/nvidia-ai-enterprise)
- Databricks Data Intelligence Platform (lakehouse plus Mosaic AI, Unity Catalog governance, and agent tooling) vs an agent-resident cross-patent architecture: where governance lives (/articles/cross-patent-architecture/databricks-data-intelligence)

Cross-Patent Architecture overview → (/cross-patent-architecture)