Nick Clark · Published April 25, 2026 · PDF

Ray (Anyscale) alternative: where does governance and identity live when agents move between nodes?

Ray, the distributed compute framework maintained by Anyscale, scales Python and AI workloads across a cluster using stateless tasks, stateful actors, and a shared object store coordinated by a head node. It is a proven substrate for the compute side of agent systems. The question this article addresses is a different one: when an agent's orchestration state, policy scope, and lineage need to survive movement across trust boundaries, where does that state live? That axis is addressed by the Execution Platform, disclosed in United States Patent Application 19/230,933.

What Ray (Anyscale) Does

Ray is an open source unified framework for scaling Python and AI applications, originating from UC Berkeley's RISELab and maintained commercially by Anyscale. Its core exposes a small set of primitives: tasks, which are stateless functions executed somewhere in the cluster; actors, which are stateful worker processes that encapsulate state and communicate through asynchronous message passing; and objects, which are immutable values placed in a distributed object store and made accessible across nodes. A cluster typically runs a single head node that coordinates scheduling and object management, with worker nodes executing tasks and holding distributed memory.

This design is genuinely good at what it targets. The distributed scheduler spreads work so that no single machine is overloaded, and the abstractions map cleanly onto the parallelism patterns that machine learning and reinforcement learning workloads need. Higher-level libraries built on Ray Core cover data processing, training, tuning, and serving, and the actor model gives a natural home for long-lived stateful components such as parameter servers or model replicas. For teams that need to move a Python workload from one machine to a cluster without rewriting it around a bespoke distributed runtime, Ray is a strong and widely adopted choice.

Ray is, in short, a compute and orchestration substrate. It answers the question of how to run many pieces of work across many machines efficiently and reliably.

The Architectural Axis

The axis worth examining is not throughput or scheduling. It is the location of an agent's governing state.

In the Ray model, an actor holds application state inside the worker process, and the cluster's control plane holds the orchestration state: what is scheduled where, which actor owns which object, how work is routed. These are separate concerns living in separate places. The actor is the thing that runs; the scheduler and object manager are the things that decide where it runs and how its outputs move. Policy about what an actor is allowed to do, which data it may touch, and how its actions should be audited is not part of the actor primitive itself. That policy typically lives in surrounding systems: the code the developer writes, an external authorization service, a data governance layer, or operational controls around the cluster.

This is a reasonable and conventional separation. It is also the axis on which the disclosed Execution Platform differs. The platform's premise is that for memory-bearing semantic agents, the orchestration state, the policy scope, and the lineage

should not be external concerns held by a control plane. They should travel inside the agent object itself.

Framed as a difference rather than a defect: Ray keeps the runtime and the control plane distinct, which is exactly right for general distributed compute. The Execution Platform collapses part of that distinction on purpose, so that an agent carries its own governing state wherever it goes.

How the Disclosed Approach Differs

The Execution Platform, disclosed in United States Patent Application 19/230,933, defines the unit of execution as a memory-bearing semantic agent object with a fixed schema of structured fields: an intent field, a context block, a memory field, a policy reference field, a mutation descriptor field, and a lineage field. These fields collectively encode the agent's objective, its semantic environment, its execution history, its permissible behaviors, its transformation eligibility, and its ancestry. Rather than storing orchestration and governing state beside the running process, the agent carries all information necessary to determine how it should behave, under what conditions it may mutate or delegate, and where it may be routed or rehydrated.

Several mechanisms follow from this that sit on the axis above.

First, policy is embedded and enforced at runtime rather than applied externally. Each agent's policy reference field contains cryptographically signed links to policy contracts. Before any mutation, delegation, or propagation, a policy enforcement engine evaluates that field against the active trust zone governance, and the action is deterministically permitted or denied without reliance on centralized authorization or post-execution filtering. The specification describes the substrate enforcing a deterministic denial and quarantine at the point of execution when a self-modifying mutation lacks the required preconditions, rather than allowing the action and resolving a violation afterward.

Second, identity is entropy-resolved rather than key-based. The specification describes a Dynamic Agent Hash derived from the agent's memory, mutation history, and lineage, entangled with a Dynamic Device Hash reflecting the host substrate's runtime entropy. Authentication proceeds through trust slope validation, confirming that an agent's identity has evolved along an acceptable trajectory across execution cycles, without reliance on persistent static credentials. This lets an agent be recognized across substrates by its identity slope rather than by a long-lived key managed by an external authority.

Third, lineage and orchestration state travel with the agent. The memory field records mutation outcomes, policy validation decisions, delegation events, and trust zone transitions as a tamper-evident, cryptographically linked record. When an agent moves between environments, described in the specification across centralized, federated, decentralized mesh, and edge substrates, its context, policy scope, and semantic lineage move with it as self-describing fields, and propagation eligibility at each boundary is determined by evaluating those fields and validating slope continuity rather than by consulting a central routing table.

Governance itself is scoped through trust zones, logical enforcement domains that validate mutation requests through zone-local validators or escalate contested requests to a meta-policy layer. The distinction the specification draws is between memory anchoring, handled by nests, and semantic control, handled by zones, so that an agent migrating between nests can retain memory continuity while being subjected to new governance rules.

Where They Fit Together

These are not substitutes for each other on most of their surface area. Ray answers how to run work across a cluster. The Execution Platform describes what governing, identity, and lineage state an agent should carry so that its behavior remains policy-bound and auditable as it moves.

A plausible composition treats Ray as the compute and scheduling substrate and treats the disclosed agent object as the thing being scheduled. The specification frames its architecture as modular and substrate-independent, capable of augmenting legacy systems or being deployed from inception, and describes nests as instantiable in centralized clusters, federated nodes, decentralized mesh, or on edge devices. Nothing in that framing requires displacing a distributed compute runtime; the substrate that executes an agent and the governing state the agent carries are different layers. Where they genuinely diverge is on the narrow axis: if the goal is that policy scope, identity, and lineage remain intrinsic to the agent regardless of which node runs it, that property comes from the agent schema and substrate enforcement described in the filing, not from the compute framework underneath.

Boundary Conditions

Honesty requires stating limits on both sides. Ray is mature, widely deployed, and battle-tested at large scale for exactly the compute and orchestration problems it targets; none of the above suggests otherwise, and for many agent systems a Ray actor holding state and an external policy service is a perfectly sound design.

On the disclosed side, the Execution Platform is described in a patent application and traces to a chain of provisional filings; it is an early-stage disclosure rather than a benchmarked, generally available product. This article makes no performance claims for it, because none are being asserted here. The properties described, embedded policy enforcement, entropy-resolved identity, and lineage-bearing agents, are architectural mechanisms set out in the specification, and their real-world behavior at scale is a matter for implementation and validation rather than assertion. The comparison is scoped strictly to where governing state lives, which is the axis the filing addresses, and does not extend to Ray's core competencies in distributed scheduling and throughput.

Disclosure Scope

The mechanisms attributed to the disclosed approach in this article are those set out in United States Patent Application 19/230,933, and only that filing defines the scope of what is disclosed and claimed. References to Ray and to Anyscale are provided as external context to locate the disclosure within a familiar architectural landscape; they describe a real, independently developed framework and are not characterizations of the filing, nor does anything here assert a defect, limitation, or deficiency in Ray, Anyscale, or their products. The separation Ray draws between runtime and control plane is a sound and deliberate design; the comparison is offered only to clarify the specific axis, the location of an agent's policy, identity, and lineage state, on which the disclosed approach is structurally different, and should not be read as a competitive or disparaging claim about any named product.

Execution Platform (/execution-platform)

All 40 steps → (/inventive-steps)

The complete runtime for governed, persistent agents.

U.S. 19/230,933 (/patents/19-230933)

PRIMARY TECHNICAL DISCLOSURE

- A Cognition-Native Execution Platform for Distributed, Stateful, and Governable Agents (/articles/a-cognition-native-execution-platform-for-distributed-stateful-and-governable-agents)

SECONDARY TECHNICAL

- Six-Field Canonical Agent Schema: Structural Definition of Autonomous Semantic Agents (/articles/execution-platform/canonical-schema)
- Semantic Nest Instantiation: Dynamic Execution Environments From Agent Density and Entropy (/articles/execution-platform/nest-instantiation)
- Trust Zone Overlay Governance: Logical Policy Domains Independent of Network Topology (/articles/execution-platform/trust-zone-overlay)

- Scoped Quorum Mutation Validation: Independent Validators With Meta-Policy Escalation (/articles/execution-platform/quorum-validation)
- Meta-Policy Override Resolution: Higher-Level Governance for Local Quorum Decisions (/articles/execution-platform/meta-policy-override)
- Semantic Router: Schema-Aware Propagation Replacing Address-Based Forwarding (/articles/execution-platform/semantic-router)
- Dynamic Agent Hash Derivation: Deterministic Identity From Memory and Mutation History (/articles/execution-platform/dah-derivation)
- Dynamic Device Hash Derivation: Substrate Identity From Device-Local Entropy (/articles/execution-platform/ddh-derivation)
- Content Anchor Hash Derivation: Perceptual Identity for Non-Executing Digital Content (/articles/execution-platform/cah-derivation)
- DAH-DDH Slope Entanglement: Binding Agent Identity to Host Device Lineage (/articles/execution-platform/dah-ddh-entanglement)
- Trust Slope Validation Across Zone Migration: Continuity Verification With Quarantine (/articles/execution-platform/zone-migration)
- Pseudonymous Propagation: Recognition by Slope Rather Than Global Identifier (/articles/execution-platform/pseudonymous-propagation)
- Alias Slope-Band Indexing: Symbolic Resolution Through Slope-Indexed Anchor Pathfinding (/articles/execution-platform/slope-band-indexing)
- Fallback Rehydration: Recovering Partial Agents Through Contextual Policy Inference (/articles/execution-platform/fallback-rehydration)
- Structural Validator With Fallback Routing: Schema Verification Before Execution (/articles/execution-platform/structural-validator)
- Execution Graph Manager: Structured Lineage of Agent Reasoning and Transformation (/articles/execution-platform/execution-graph)
- Full and Partial Agent Interoperability: Cross-Boundary Semantic Exchange Under Policy (/articles/execution-platform/agent-interoperability)
- Cross-Topology Substrate Deployment: Identical Agent Structure Across All Substrates (/articles/execution-platform/cross-topology)

APPLICATIONS · GENERAL

- Governable AI Agents: Auditable Reasoning, Policy-Constrained Orchestration, and Training-Artifact Traceability (/articles/execution-platform/ai-agent-governance)
- Multi-Cloud Agent Orchestration Without a Centralized Scheduler (/articles/execution-platform/multi-cloud-orchestration)

- Autonomous Fleet Coordination Through Self-Governing Agents (/articles/execution-platform/fleet-coordination)
- Enterprise Workflow Automation Without Orchestration Servers (/articles/execution-platform/enterprise-workflow-automation)
- Smart Contract Alternative Without Blockchain Latency: Governed Contract Execution (/articles/execution-platform/smart-contract-alternative)
- Reproducible Scientific Computing With Provenance-Bearing Governed Agents (/articles/execution-platform/scientific-computing)
- Supply Chain Autonomous Agents (/articles/execution-platform/supply-chain-agents)
- Distributed Energy Grid Management With Governed Autonomous Agents (/articles/execution-platform/energy-grid-management)
- Disaster Response Coordination Without Central Command (/articles/execution-platform/disaster-response-coordination)
- Sovereign Agent Runtimes: Running AI Agents Air-Gapped and On-Premises for Defense and Regulated Industries (/articles/execution-platform/sovereign-agent-runtimes)

APPLICATIONS · SPECIFIC

- Kubernetes Orchestrates Containers. It Does Not Know What They Are Doing. (/articles/execution-platform/kubernetes)
- Temporal Alternative for Governed Agent Execution: Durable Workflows Have No Semantic Identity (/articles/execution-platform/temporal-io)
- Apache Airflow vs. Governed Agent Execution: DAG Scheduling or Agent-Level Governance? (/articles/execution-platform/apache-airflow)
- Prefect Alternative for Governed Agent Execution: Beyond Python Task Scheduling (/articles/execution-platform/prefect)
- AWS Step Functions Alternative for Governed Agent Execution (/articles/execution-platform/aws-step-functions)
- Azure Durable Functions vs a Governed Execution Platform: Where Does Step Authority Live? (/articles/execution-platform/azure-durable-functions)
- HashiCorp Nomad vs. a Governance-Bearing Execution Platform: Where Does Workload Authority Live? (/articles/execution-platform/nomad)
- Docker Swarm Alternative for Governed Agent Execution: Beyond Opaque Containers (/articles/execution-platform/docker-swarm)
- Apache Mesos Managed Datacenter Resources. The Resources Had No Semantic Governance. (/articles/execution-platform/mesos)
- Argo Workflows Alternative for Governed Pipelines: Kubernetes-Native DAGs Without a Governance Substrate (/articles/execution-platform/argo-workflows)

- Dagster Alternative for Governed Pipelines: Software-Defined Assets Without a Governance Substrate (/articles/execution-platform/dagster)
- Luigi Alternative for Governed Agent Execution: Beyond Task-Dependency Pipelines (/articles/execution-platform/luigi)
- Camunda vs Governed Agent Execution: BPMN Orchestration Beyond the Process Engine (/articles/execution-platform/camunda)
- Zeebe vs Governed Agent Execution: Does Governance Scale With Throughput? (/articles/execution-platform/zeebe)
- AWS RoboMaker vs Governed Agent Execution at the Fleet Edge (/articles/execution-platform/aws-robomaker)
- NVIDIA Cosmos vs Governed Agent Execution: World Models Need a Runtime (/articles/execution-platform/nvidia-cosmos)
- NVIDIA DRIVE vs Governed Agent Execution: A Cross-Vehicle Substrate Alternative (/articles/execution-platform/nvidia-drive)
- NVIDIA Isaac vs a Governed Agent Execution Substrate (/articles/execution-platform/nvidia-isaac)
- NVIDIA Metropolis vs Governed Agent Execution: A Metropolis Alternative for Edge Cognition (/articles/execution-platform/nvidia-metropolis)
- LangGraph (LangChain) alternative: where does agent state, policy, and lineage actually live? (/articles/execution-platform/langgraph)
- Microsoft AutoGen vs a substrate-embedded execution platform: where does agent orchestration state live? (/articles/execution-platform/microsoft-autogen)
- CrewAI alternative: where does delegation and policy state live at runtime? (/articles/execution-platform/crewai)
- **Ray (Anyscale) alternative: where does governance and identity live when agents move between nodes? (/articles/execution-platform/ray-anyscale)**

Execution Platform overview → (/execution-platform)