Six-Action Execution Evaluation Cycle: Parse, Evaluate, Select at Every Node

by Nick Clark | Published March 27, 2026 | PDF

At each execution node, parsing intent, evaluating context against local policy, reading memory, and selecting from execution, mutation, delegation, dormancy, reentry, or termination. Within the memory-resident execution architecture, this capability operates as a structural primitive at the execution level. It is not an optional enhancement or a configurable plugin but a mandatory architectural property that every participant encounters. The result is a system where six-action execution evaluation cycle is enforced by construction rather than by convention, policy, or external oversight.

What It Is

At each execution node, parsing intent, evaluating context against local policy, reading memory, and selecting from execution, mutation, delegation, dormancy, reentry, or termination. This is a structural mechanism within the memory-resident execution architecture that operates at the execution level. It is not advisory, not configurable at the discretion of individual participants, and not dependent on external enforcement infrastructure.

Every interaction within the system encounters this mechanism as a mandatory constraint. The behavior it produces is deterministic: given the same inputs and the same system state, the outcome is identical regardless of which node evaluates it, when the evaluation occurs, or what substrate hosts the computation.

Why It Matters

Conventional execution systems address this problem through external schedulers, message queues, and orchestration layers. These approaches function adequately under controlled conditions but introduce structural fragility when the scheduler fails, the queue is lost, or execution state becomes inconsistent. The underlying assumption that a centralized coordinator can reliably track and resume all pending executions becomes a liability precisely when reliability matters most.

Six-action execution evaluation cycle removes this fragility by embedding the relevant capability directly into the execution layer. There is no external dependency that can fail independently, no middleware that can be misconfigured, and no trust assumption that can be violated by a single compromised participant. The guarantee is structural.

How It Works

The mechanism operates through deterministic evaluation embedded in the memory-resident execution architecture. When a relevant operation is initiated, the system evaluates the applicable structural constraints against the current state. This evaluation consults the fields, policies, and lineage records that travel with the objects themselves rather than relying on external state that may be stale, unavailable, or compromised.

The outcome of each evaluation is recorded in an append-only lineage structure. This record is cryptographically committed, ensuring that the complete history of decisions, transitions, and state changes remains auditable and tamper-evident. No evaluation outcome can be retroactively altered without breaking the cryptographic chain.

Because the evaluation logic and the data it operates on travel together, the mechanism functions identically across network partitions, substrate migrations, and administrative boundaries. There is no central evaluation point that must be available for the system to operate correctly.

What It Enables

With six-action execution evaluation cycle as an architectural primitive, systems built on this foundation can operate autonomously while maintaining the structural guarantees that centralized architectures achieve through oversight. The capability is not a tradeoff between autonomy and governance but a resolution of the apparent conflict between them.

This enables deployment across centralized cloud infrastructure, federated multi-party environments, fully decentralized networks, and edge installations with intermittent connectivity. The structural guarantees hold regardless of deployment topology because they are properties of the objects and protocols themselves, not properties of the infrastructure that hosts them.

Memory-Resident Execution All 21 steps →

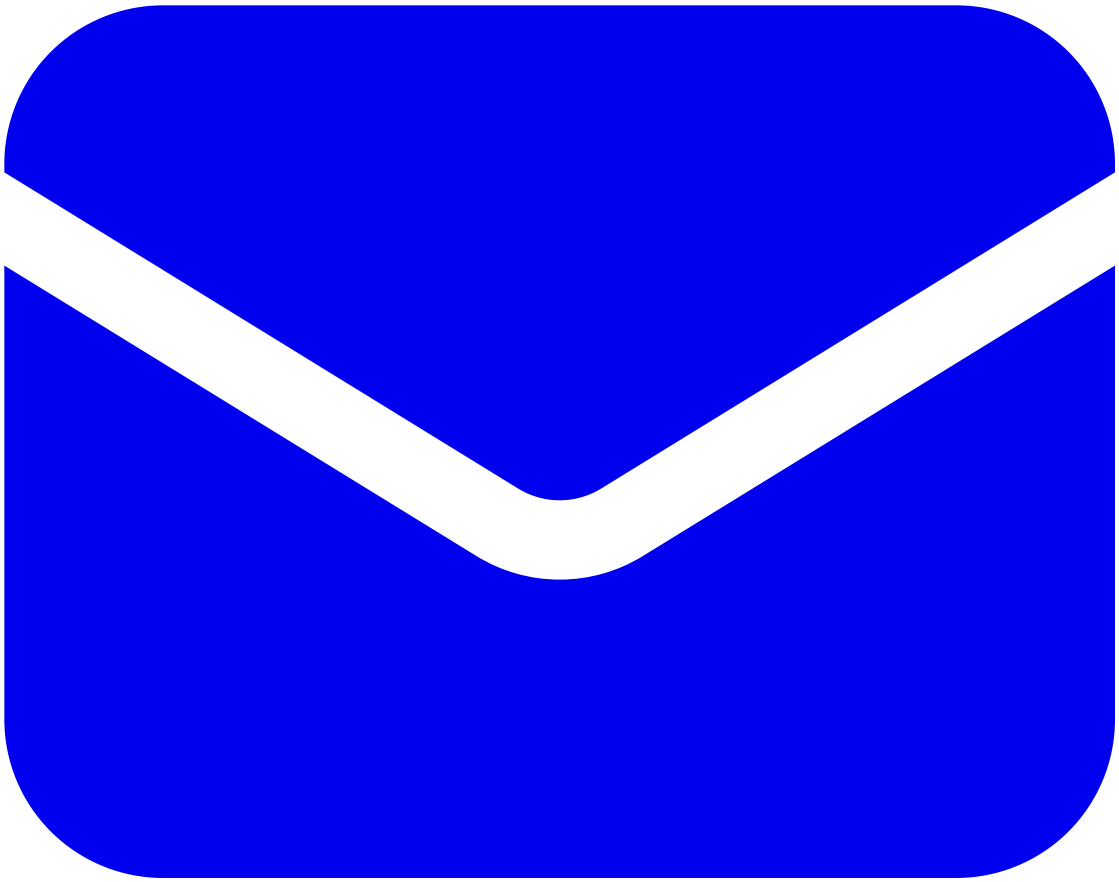
Persistent objects that execute without orchestration.

Patent
US 19/538,221 · filed
Primary Technical Disclosure
○ Memory-Resident Execution: Persistent Semantic Objects Without Orchestration
Secondary Technical
● Six-Action Execution Evaluation Cycle: Parse, Evaluate, Select at Every Node○ Cognition-Authority-Execution Separation: Reasoning Cannot Authorize Action○ Dormancy as First-Class Execution State: Valid Suspension Without Failure○ Semantic Backoff: Retry Pacing From Execution Outcomes Rather Than Fixed Timers○ Wake Triggers for Dormancy Exit: Explicit Reentry Conditions in Memory○ Persistent Polling Behavior: Autonomous Condition Evaluation Without Schedulers○ Intent Refinement During Execution: Adaptive Objectives Without Re-Instantiation○ Compositional Execution Through Recursive Delegation: Parent-Child Lineage Tracking○ Negative Capability Signals: Recording What Cannot Be Done as Structured Constraint○ Swarm-Based Execution Emergence: Coordinated Behavior Without Centralized Control○ Latency and Failure as Semantic Signals: Structured Inputs From Adverse Conditions○ LLM as Advisory Execution Node: Inference Without Authority Over Agent State○ Append-Only Memory Field: Complete Execution Lineage Through Immutable Records
Applications (General)
○ Serverless Execution Without Cold Starts or State Loss○ Long-Running Autonomous Workflows Without External Orchestration○ Drone Operations Surviving Disconnection○ Deep Space Agent Execution Without Ground Control○ Underwater Robotic Operations Without Connectivity○ Rural Healthcare Agents Surviving Intermittent Connectivity○ Operations in Infrastructure-Destroyed Environments○ Offline Financial Transaction Agents
Applications (Specific)
○ Cloudflare Durable Objects Made State Local. The Objects Still Need Orchestration.○ Azure Service Fabric Actors Are Addressable. They Are Not Autonomous.○ Akka Perfected the Actor Model. Actors Still React Instead of Self-Execute.○ Orleans Made Virtual Actors Practical. The Actors Still Execute on Request.○ Dapr Provides a Sidecar Runtime for Microservices. The Services Still Need External Orchestration.○ wasmCloud Runs WebAssembly Actors. The Actors Wait for Messages.○ Spin Made WebAssembly Serverless. The Functions Are Still Trigger-Based.○ Fermyon Built the WebAssembly Cloud. The Cloud Hosts Functions, Not Self-Executing Objects.○ Fly Machines Made Micro-VMs Fast. The VMs Still Need External Orchestration.○ Railway Simplified Application Deployment. The Applications Still Depend on External Execution Triggers.
Memory-Resident Execution overview →
AQ
deterministic
autonomy

Legal

Subject to one or more pending U.S. and international patent applications, see Patents for the current list and status. No license, express or implied, is granted. Any use requires a separate written agreement—see Licensing. Patent applications referenced on this site are pending. Claim scope, if any, is subject to examination and may issue in altered form or not at all. See Legal for terms and conditions.

Last updated: 2026-03-03



-
- nick@qu3ry.net
- 72 28 14 36 01

Invented by Nick Clark | Founding Investors: Devin Wilkie