Long-Running Autonomous Workflows Without External Orchestration

by Nick Clark | Published March 27, 2026 | PDF

Enterprise workflows that span weeks or months require orchestration engines to track state across steps, handle failures, manage retries, and coordinate participants. These engines become the most complex and failure-prone component in the system. Memory-resident execution offers a structural alternative: workflow objects that carry their own state, manage their own execution lifecycle, and navigate multi-step processes autonomously through dormancy, wake triggers, and self-evaluation.

The orchestration burden in enterprise workflows

Consider a regulatory compliance review that takes six weeks, involves twelve approvers across four departments, requires document collection from three external parties, and must maintain an audit trail of every state change. Orchestration engines like Temporal, Camunda, and Apache Airflow manage this by maintaining a centralized workflow state machine that tracks where each instance stands and what needs to happen next.

The orchestration engine must be highly available because every active workflow depends on it. It must be durable because workflow state loss means restarting processes that took weeks to reach their current step. It must handle versioning because workflow definitions change while instances are in flight. The orchestrator meant to simplify workflow management becomes the most operationally demanding component in the stack.

At scale, the problem compounds. An insurance company processing fifty thousand claims concurrently, each following a multi-week workflow with conditional branches and parallel approval paths, puts enormous load on the orchestration tier. Every state transition, every timer event, every retry is coordinated through the central engine.

Why durable execution improves but does not resolve the problem

Durable execution frameworks persist workflow state through event sourcing: every state transition is recorded as an event, and workflow state can be reconstructed by replaying the event log. This provides durability and fault tolerance. But the execution model is still external to the workflow object. An external framework manages the event log, triggers replays, and coordinates execution. The workflow itself is a passive definition that the framework interprets.

Event replay introduces its own operational challenges. As workflows run longer and accumulate more events, replay times grow. Non-deterministic side effects must be carefully managed. Version migrations of workflow definitions require careful event log compatibility analysis. The operational complexity of durable execution grows with workflow duration and complexity.

How memory-resident execution addresses this

Memory-resident execution treats each workflow instance as a self-governing object that carries its own state, execution logic, and lifecycle management. The workflow object does not depend on an external engine to track its progress. It tracks itself.

When a compliance review reaches the document collection step, the workflow object enters dormancy with a wake trigger set for document arrival. The object consumes no compute resources while dormant but retains its full state: which steps are complete, which approvals are pending, what documents are expected, and what the audit trail contains. When a document arrives, the wake trigger fires, the object resumes, evaluates the document against its policy constraints, and advances to the next step.

Failure handling is self-managed. If a step fails, the object evaluates the failure against its own retry policy and semantic backoff parameters. It does not need an external retry engine. Conditional branching is evaluated against the object's own state. Parallel approval paths are managed through recursive delegation to sub-objects that carry their own execution cycles.

The audit trail is inherent. Every state mutation is recorded in the object's append-only memory. No external logging system is needed because the object's history is its memory, carried as an intrinsic property.

What implementation looks like

An enterprise deploying memory-resident workflows replaces the orchestration engine with a substrate that provides persistence and wake-trigger delivery. Each workflow instance is a persistent object that manages itself. The substrate does not interpret workflow logic. It provides the environment in which self-governing objects operate.

For healthcare organizations managing patient treatment workflows that span months, each treatment plan is a memory-resident object carrying the full treatment history, pending steps, approval states, and compliance constraints. The object advances through treatment steps autonomously, entering dormancy between appointments and waking when results arrive.

For legal firms managing case workflows with dozens of parallel document reviews, depositions, and court filings, each case is a self-managing object that tracks its own deadlines, manages its own document dependencies, and escalates its own overdue items without an orchestration engine coordinating the effort.

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

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

Invented by Nick Clark | Founding Investors: Devin Wilkie