Nick Clark · Published April 25, 2026 · PDF

Golem vs object-resident execution state: who carries the task state across nodes?

Golem is a decentralized compute marketplace that lets a requestor rent computation from a distributed network of provider nodes. It solves where a task runs, but the question of where a long-running task's execution state lives across nodes and across time is a separate axis. This article approaches that axis from the perspective of Memory-Resident Execution, the inventive step disclosed in United States Patent Application 19/538,221, which carries execution state inside the task object itself.

What Golem Does

Golem, developed by Golem Factory and generally known as the Golem Network, is a decentralized marketplace for computation. Its architecture connects two roles: requestors, who have work they want computed, and providers, who offer spare compute capacity to the network. A requestor describes a workload, the network matches it to willing providers, the work runs in a sandboxed environment on provider hardware, and settlement occurs through the network's payment layer. In broad terms, Golem turns idle machines around the world into a rentable, permissionless pool of compute.

This is a genuinely hard problem, and Golem addresses it well. Building a permissionless marketplace requires solving provider discovery, workload packaging and isolation, verifiable delivery of results, and trustless payment between parties who have no prior relationship. Golem's model of packaging workloads for execution in isolated environments on untrusted hardware, and settling payment for that execution without a central intermediary, is a substantial engineering achievement. For batch and embarrassingly parallel workloads such as rendering or scientific computation, renting distributed capacity from a marketplace is a natural and effective fit.

The point of this article is not that Golem does its job poorly. It does its job well. The point is that its job is on a different axis from the one the disclosed invention addresses.

The Architectural Axis

The axis here is: where does the execution state of a long-running, multi-step task live as that task moves across nodes and across time?

A compute marketplace answers the question of where a unit of work is executed. It matches a workload to a provider, runs it there, and returns a result. That framing is naturally oriented toward discrete units of work: package a job, dispatch it, receive an output. When a task is long-running, adaptive, or spans many asynchronous intervals, the state that describes how far the task has progressed, what it has already tried, what policy governed each decision, and under what conditions it should resume must be held somewhere. In a marketplace model, that continuity is typically the requestor's responsibility, or is handled by whatever orchestration the requestor layers on top: an external controller that tracks progress, reissues work, and stitches results together across provider invocations.

This is not a defect in Golem. A marketplace is not obligated to be a durable-execution engine, and keeping execution state external to the compute layer is a defensible and common design. It simply means that the continuity of a long-lived task is managed

outside the units of work, by something that coordinates them. The disclosed invention takes a structurally different position on exactly that point.

How the Disclosed Approach Differs

The Memory-Resident Execution disclosed in United States Patent Application 19/538,221 carries execution state inside the task object itself. The unit of work is a persistent executable object comprising an intent field encoding a machine-readable execution descriptor, a context block encoding execution-relevant metadata, and a memory field encoding prior execution state. The memory field stores an append-only execution history: traces, mutation records, delegation references, policy outcomes, and reentry information accumulated as the object executes.

Because the state travels with the object, execution continuity is a property of the object rather than of any external controller. The specification describes that the object propagates among a plurality of execution nodes, and at each node that receives it, the node performs an execution evaluation cycle: it parses the intent field, evaluates the context block against locally applicable execution policy without reliance on centralized coordination, reads the memory field to retrieve prior execution records, and selects an execution action from the group consisting of execution, mutation, delegation, dormancy, reentry, and termination. The outcome is appended to the memory field. As claimed, the object persists across asynchronous execution intervals and resumes execution without re-instantiation based on reentry conditions encoded in the memory field.

Two consequences follow from the specification. First, an execution node need not store execution progress, eligibility, or history for the object outside the object's own memory field; the specification describes an embodiment in which nodes hold no such external state. Continuity does not depend on any particular node remaining available, because the object is serialized for propagation and deserialized prior to each evaluation cycle, preserving continuity independently of execution node identity. Second, the invention

treats dormancy and reentry as first-class execution actions selected locally from object-resident state, so a task can suspend when conditions are unmet and resume when reentry conditions recorded in its memory field are satisfied, without a central scheduler tracking it.

The difference in one sentence: a marketplace routes work to where it runs and leaves cross-node, cross-time state management to whatever orchestrates the work, whereas the disclosed model makes the state travel inside the work.

Where They Fit Together

These are complementary layers, not substitutes. Golem answers where computation happens: it supplies a decentralized pool of provider nodes willing to execute sandboxed workloads and settle payment for doing so. The disclosed model answers what carries the long-lived execution state as work moves and time passes: a persistent object whose memory field preserves continuity.

One could, in principle, compose them. The specification frames execution nodes broadly as any computing system, process, or execution environment capable of evaluating a semantic object and performing execution actions based on the information embedded within it, and describes deployment across stateless, federated, edge-oriented, and agent-based environments. A marketplace provider could serve as such an execution node, evaluating an object and appending its outcome, while the marketplace continues to handle discovery, isolation, and settlement. In that arrangement Golem does what it is good at, and the object-resident model supplies the durable, auditable, cross-node continuity that a discrete-job marketplace does not itself set out to provide.

Boundary Conditions

Honesty requires stating the limits on both sides. The disclosed invention is the subject of a patent application, United States Patent Application 19/538,221; a filing describes and claims an approach, and this article describes what the specification discloses rather than an independently benchmarked production system. No performance figures are asserted for the disclosed approach, and none appear in the specification. The specification itself notes that execution may be deterministic or non-deterministic depending on the evaluation mechanisms a given node applies, and that object-resident continuity concerns the preservation and serialization of execution state, not the determinism of the mechanisms that produce state transitions. An object-resident model also implies that meaningful execution state accompanies the object, which is a design tradeoff that suits long-horizon, adaptive, policy-bound tasks more than fire-and-forget batch jobs.

On the Golem side, the characterization here is deliberately kept to widely understood, architecture-level facts: a decentralized marketplace matching requestors to providers, sandboxed execution on provider hardware, and trustless settlement. Golem's specific capabilities, supported workload types, and roadmap evolve over time, and readers should consult Golem's own current documentation for authoritative detail. Nothing here should be read as asserting a defect in Golem or as a claim that a marketplace ought to behave as a durable-execution layer.

Disclosure Scope

The technical description of the disclosed approach in this article is grounded in United States Patent Application 19/538,221 and its specification, which defines the persistent executable object, its intent field, context block, and append-only memory field, and the local execution evaluation cycle performed without centralized coordination. The references to Golem, to compute marketplaces, and to durable-execution and orchestration categories are provided as external market and architectural context to

situate the invention on a specific axis; they are not part of, and do not limit, the claims of the filing. This article does not assert that Golem or any other named product is deficient, infringing, or unsuitable for its intended purpose; Golem is described only at the level of genuine, publicly understood architectural facts and is credited for what it does well. Any comparison is limited to the structural axis the invention addresses, namely where execution state is carried across nodes and across time, and is offered for informational purposes rather than as a legal characterization of any third party.

Memory-Resident Execution (/memory-resident-execution)

All 40 steps → (/inventive-steps)

Persistent objects that execute without orchestration.

U.S. 19/538,221 (/patents/19-538221)

PRIMARY TECHNICAL DISCLOSURE

- Memory-Resident Execution: Persistent Semantic Objects Without Orchestration (/articles/memory-resident-execution-persistent-semantic-objects-without-orchestration)

SECONDARY TECHNICAL

- Six-Action Execution Evaluation Cycle: Parse, Evaluate, Select at Every Node (/articles/memory-resident-execution/execution-cycle)
- Cognition-Authority-Execution Separation: Reasoning Cannot Authorize Action (/articles/memory-resident-execution/cognition-authority-separation)
- Dormancy as First-Class Execution State: Valid Suspension Without Failure (/articles/memory-resident-execution/dormancy-state)
- Semantic Backoff: Retry Pacing From Execution Outcomes Rather Than Fixed Timers (/articles/memory-resident-execution/semantic-backoff)
- Wake Triggers for Dormancy Exit: Explicit Reentry Conditions in Memory (/articles/memory-resident-execution/wake-triggers)
- Persistent Polling Behavior: Autonomous Condition Evaluation Without Schedulers (/articles/memory-resident-execution/persistent-polling)

- Intent Refinement During Execution: Adaptive Objectives Without Re-Instantiation (/articles/memory-resident-execution/intent-refinement)
- Compositional Execution Through Recursive Delegation: Parent-Child Lineage Tracking (/articles/memory-resident-execution/recursive-delegation)
- Negative Capability Signals: Recording What Cannot Be Done as Structured Constraint (/articles/memory-resident-execution/negative-capability)
- Swarm-Based Execution Emergence: Coordinated Behavior Without Centralized Control (/articles/memory-resident-execution/swarm-execution)
- Latency and Failure as Semantic Signals: Structured Inputs From Adverse Conditions (/articles/memory-resident-execution/failure-signals)
- LLM as Advisory Execution Node: Inference Without Authority Over Agent State (/articles/memory-resident-execution/llm-advisory-node)
- Append-Only Memory Field: Preserving Execution Lineage Through Appended Records (/articles/memory-resident-execution/append-only-memory)

APPLICATIONS · GENERAL

- Execution Continuity for DDIL Coalition C2: Memory-Resident Tasking Across Disconnected, Trust-Divergent Tactical Networks (/articles/memory-resident-execution/defense-tactical-edge-ddil)
- Stateful Serverless: Eliminating Cold Starts and State Loss in FaaS (/articles/memory-resident-execution/serverless-persistence)
- Long-Running Business Workflows Without an Orchestration Engine (/articles/memory-resident-execution/long-running-workflows)
- Autonomous Drone Operations Surviving Ground Control Link Loss (/articles/memory-resident-execution/autonomous-drone-operations)
- Deep Space Agent Execution Without Ground Control (/articles/memory-resident-execution/space-exploration-agents)
- Autonomous Underwater Vehicle Mission Autonomy Without Surface Connectivity (/articles/memory-resident-execution/underwater-robotics)
- Offline Clinical Agents for Rural Healthcare With Intermittent Connectivity (/articles/memory-resident-execution/rural-healthcare-agents)
- Disaster Response Software That Works When Infrastructure Is Destroyed (/articles/memory-resident-execution/disaster-zone-operations)
- Offline Payment Agents That Stay Compliant When the Network Drops (/articles/memory-resident-execution/offline-financial-agents)

APPLICATIONS · SPECIFIC

- Cloudflare Durable Objects vs Memory-Resident Execution: Who Holds Authority Over the Object (/articles/memory-resident-execution/durable-objects)
- Azure Durable Actors Alternative: Governed, Cross-Domain Execution Beyond Service Fabric Reliable Actors (/articles/memory-resident-execution/azure-actors)
- Akka Alternative: Governed, Self-Executing Objects Beyond the Reactive Actor Model (/articles/memory-resident-execution/akka)
- Microsoft Orleans Alternative: Governed, Cross-Domain Execution Beyond Silo-Cluster Grains (/articles/memory-resident-execution/orleans)
- Dapr Alternative for Governed State: Where Authority Lives When State Moves (/articles/memory-resident-execution/dapr)
- wasmCloud vs Memory-Resident Execution: Message-Reactive Actors and Self-Executing Objects (/articles/memory-resident-execution/wasmcloud)
- Spin Alternative for Governed Agents: WebAssembly Serverless vs Memory-Resident Execution (/articles/memory-resident-execution/spin)
- Fermyon Spin vs a Persistent Executable Object: Which Hosts Governed Agents That Carry Their Own Policy and Lineage? (/articles/memory-resident-execution/fermyon)
- Fly Machines Alternative: Governed, Self-Carrying Execution Beyond Externally Orchestrated Micro-VMs (/articles/memory-resident-execution/fly-machines)
- Railway Alternative for Long-Running Autonomous Services: Memory-Resident Execution vs Trigger-Driven Deployment (/articles/memory-resident-execution/railway)
- Temporal alternative: object-resident execution state versus a durable-execution service (/articles/memory-resident-execution/temporal)
- Restate vs object-resident execution state: where durable execution keeps the journal (/articles/memory-resident-execution/restate)
- AWS Step Functions alternative: where does execution state live, in the orchestrator or in the object? (/articles/memory-resident-execution/aws-step-functions)
- **Golem vs object-resident execution state: who carries the task state across nodes? (/articles/memory-resident-execution/golem)**

Memory-Resident Execution overview → (/memory-resident-execution)