

# Orleans Made Virtual Actors Practical. The Actors Still Execute on Request.

by Nick Clark | Published March 27, 2026 | PDF

Microsoft Orleans brought the virtual actor model to .NET, making distributed stateful objects as simple to use as ordinary method calls. Grains activate transparently, persist state automatically, and scale across clusters. The developer experience is remarkable. But Orleans grains activate when called and deactivate when idle. They do not carry their own execution cycle, self-evaluate from governed memory, or act autonomously. The gap is between demand-driven virtual actors and memory-resident self-executing objects.

---

Orleans originated from Microsoft Research and powers production systems at scale. Its virtual actor model eliminates the complexity of actor lifecycle management. The gap described here is not about the programming model. It is about what the model enables versus what autonomous execution requires.

## Virtual activation is demand-driven

Orleans grains exist virtually: they always appear to be available, but physically activate only when a method is called. The runtime handles placement, activation, and deactivation transparently. This is elegant. It eliminates the need to manage actor lifecycle.

But virtual activation is demand-driven. A grain that nobody calls does not execute. A grain with urgent internal state that warrants action cannot act until something calls it. Timers and reminders provide periodic callbacks, but these are the runtime calling the grain, not the grain executing from its own evaluation.

## State persistence without semantic governance

Orleans provides automatic state persistence through configurable storage providers. Grain state is serialized and restored transparently. But the persisted state is a developer-defined data structure. There is no typed schema for governance, memory, or lineage. The runtime persists data. It does not govern it.

## What memory-resident execution provides

Memory-resident execution objects carry their own execution cycle, independent of external calls. They self-evaluate from governed semantic memory on each cycle. They determine whether to act, delegate, wait, or enter dormancy based on their own state. They record every mutation in lineage with governance provenance.

Orleans' virtual actor model, automatic placement, and state persistence could serve as infrastructure for memory-resident objects. But the self-evaluation cycle, governance validation, and autonomous execution must be intrinsic to the object, not dependent on external method calls.

## The remaining gap

Orleans made virtual actors practical. The remaining gap is in execution autonomy: objects that carry their own execution cycle, evaluate from governed semantic memory, and act without waiting for calls. That is the difference between a virtual actor and a memory-resident semantic object.

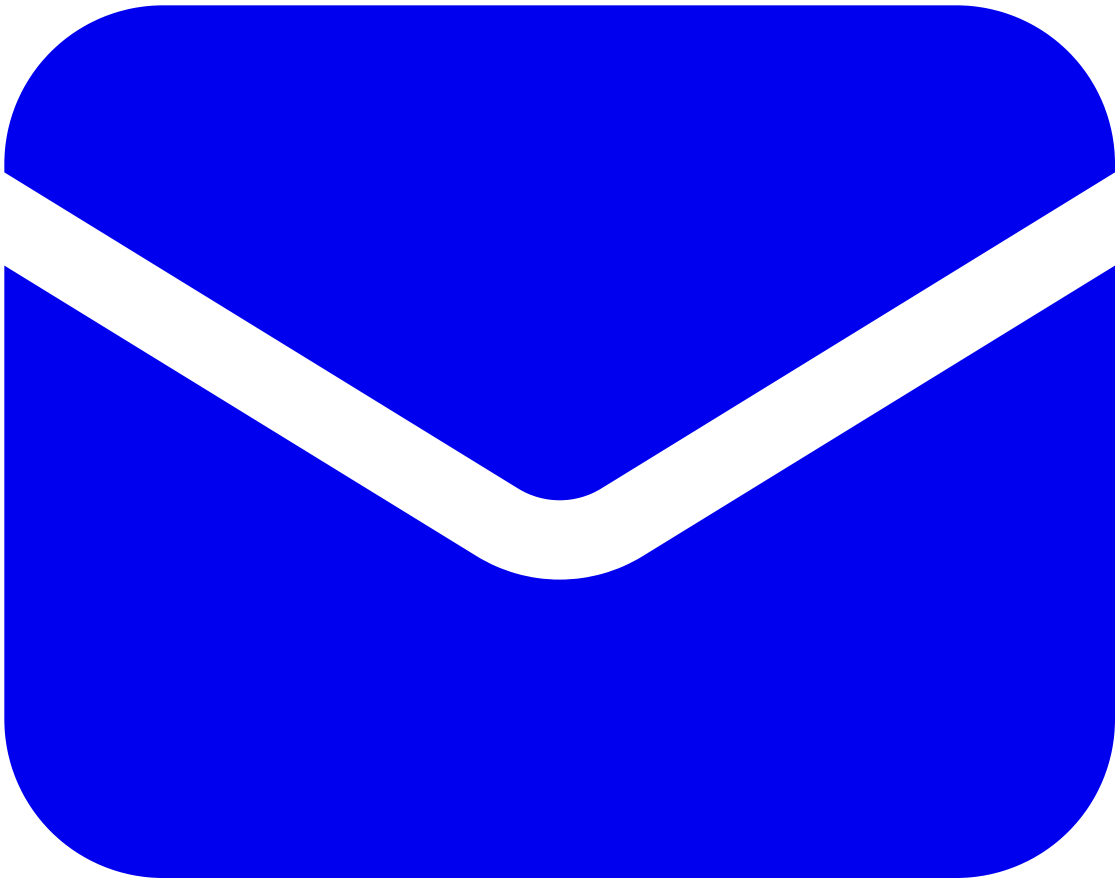
Memory-Resident Execution All 21 steps →

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

Last updated: 2026-03-03




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

Invented by Nick Clark | Founding Investors: Devin Wilkie