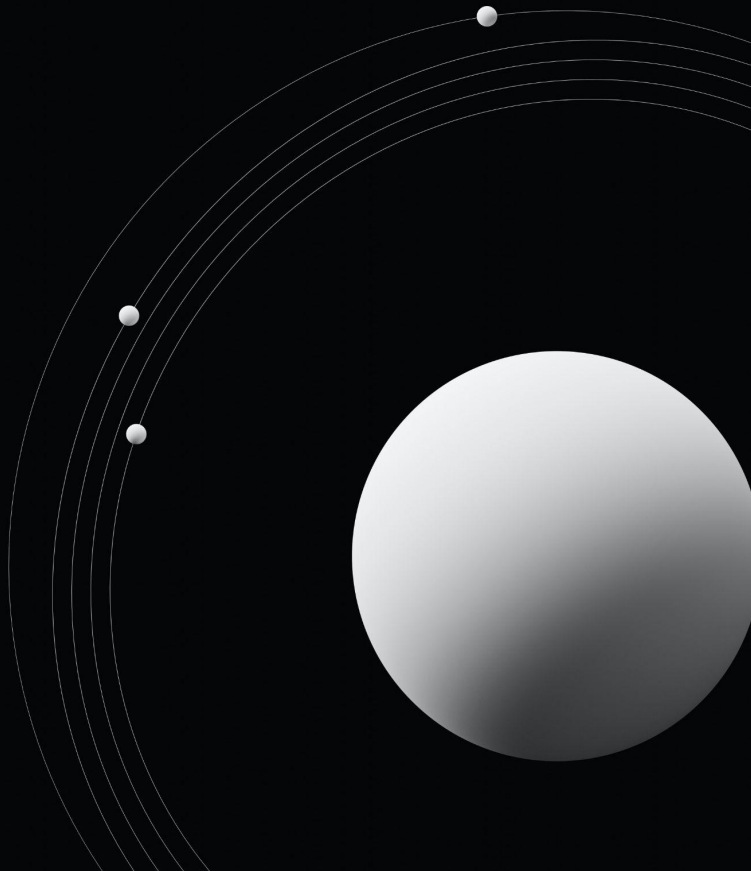
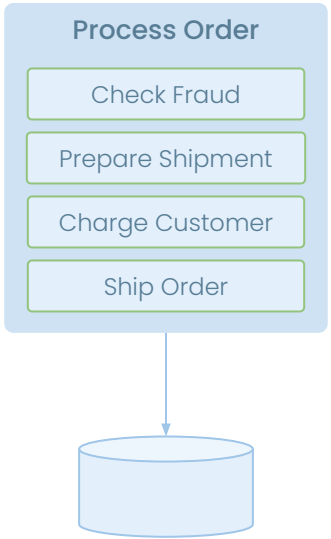
# 1

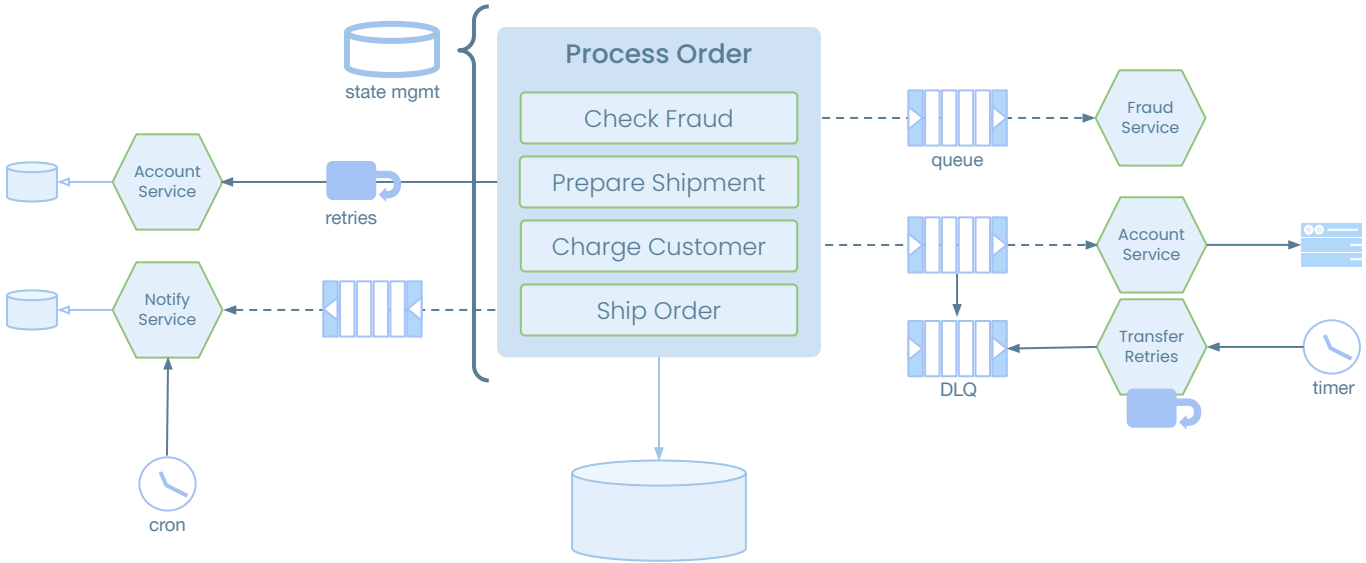
# Temporal intro



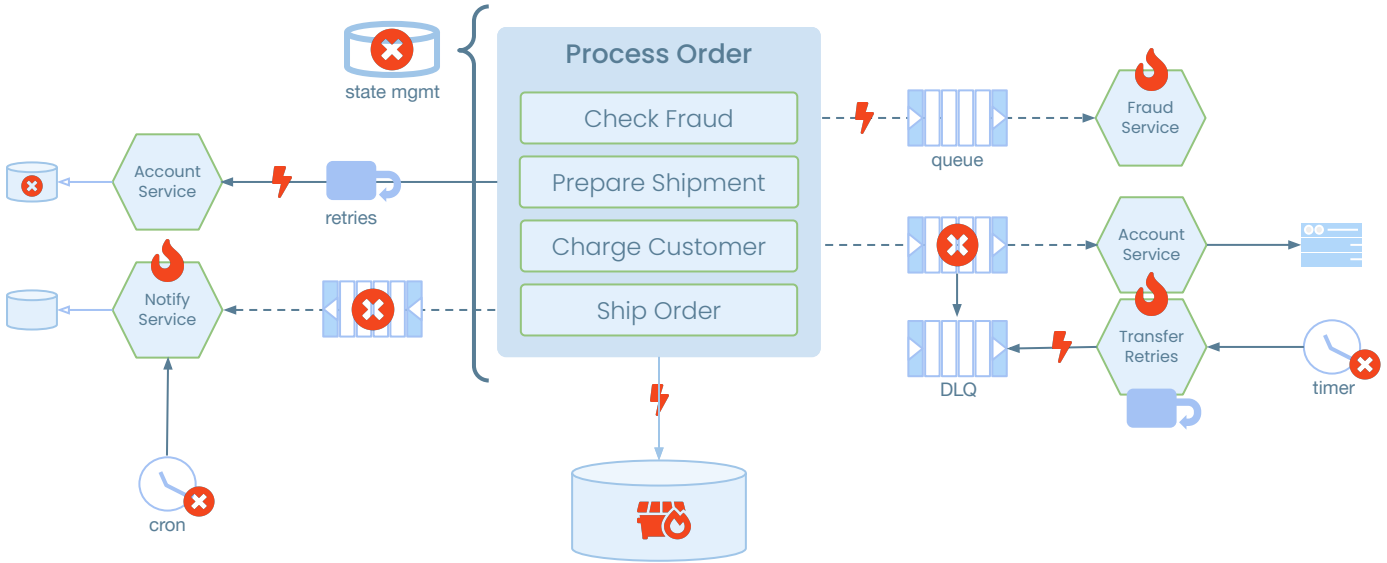
# Simple Implementation



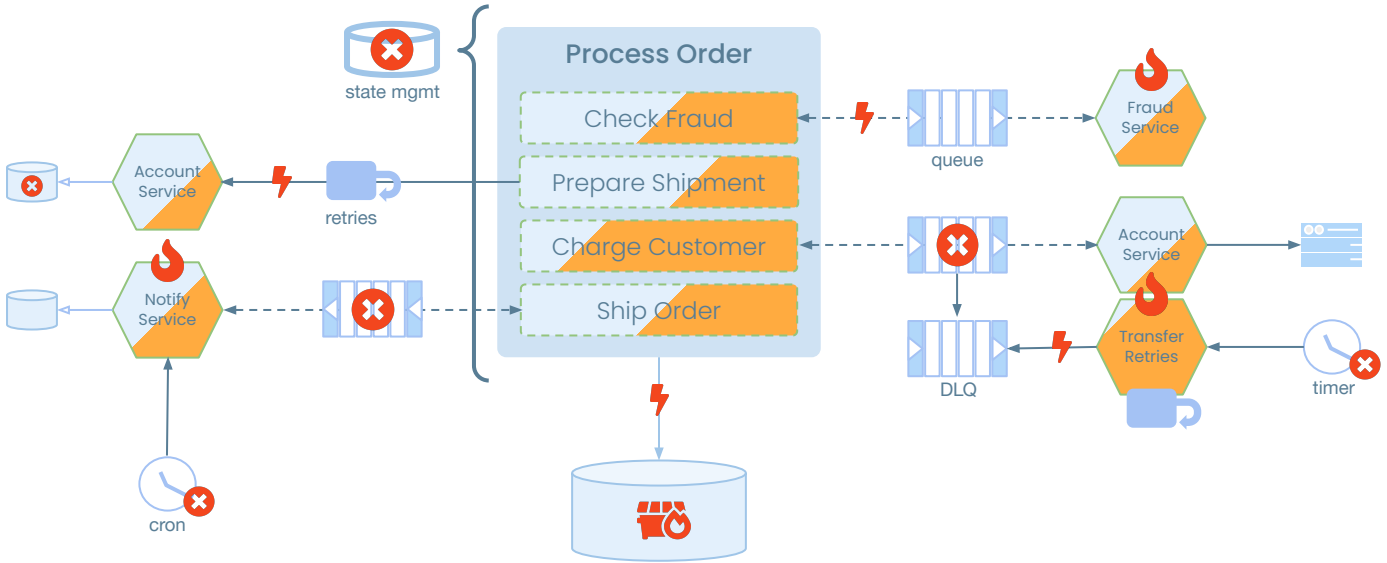
# Simple Becomes Complex



# Failures are Inevitable



# Business Logic vs. Plumbing Code

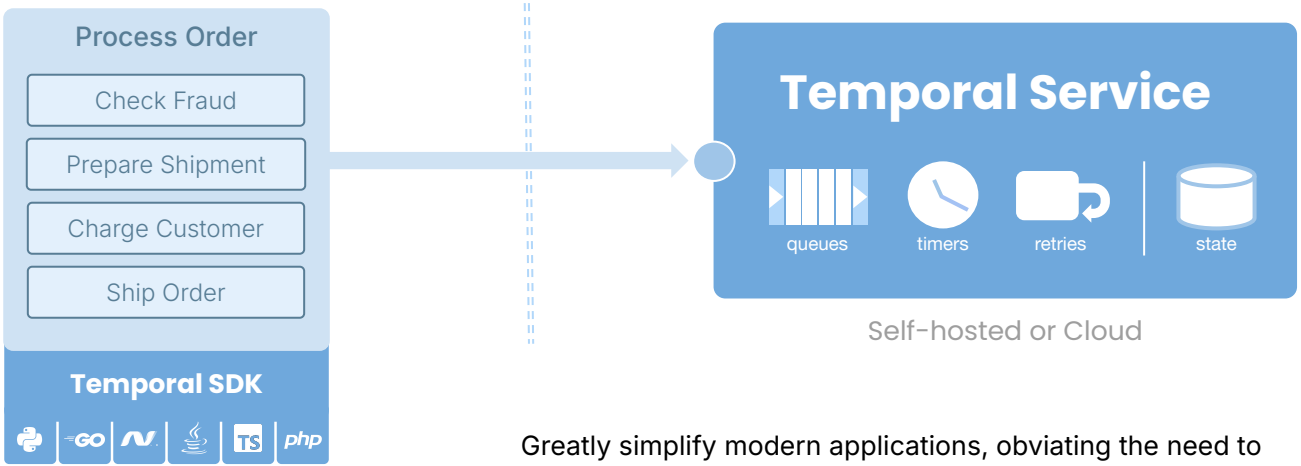



Business Logic

Plumbing Code

# ENTER DURABLE EXECUTION

**Define a workflow and then**
**maintain state and coordinate execution**




Greatly simplify modern applications, obviating the need to create queues, manage timers, publish and consume events, implement retries and rollbacks, checkpoint state, and more.

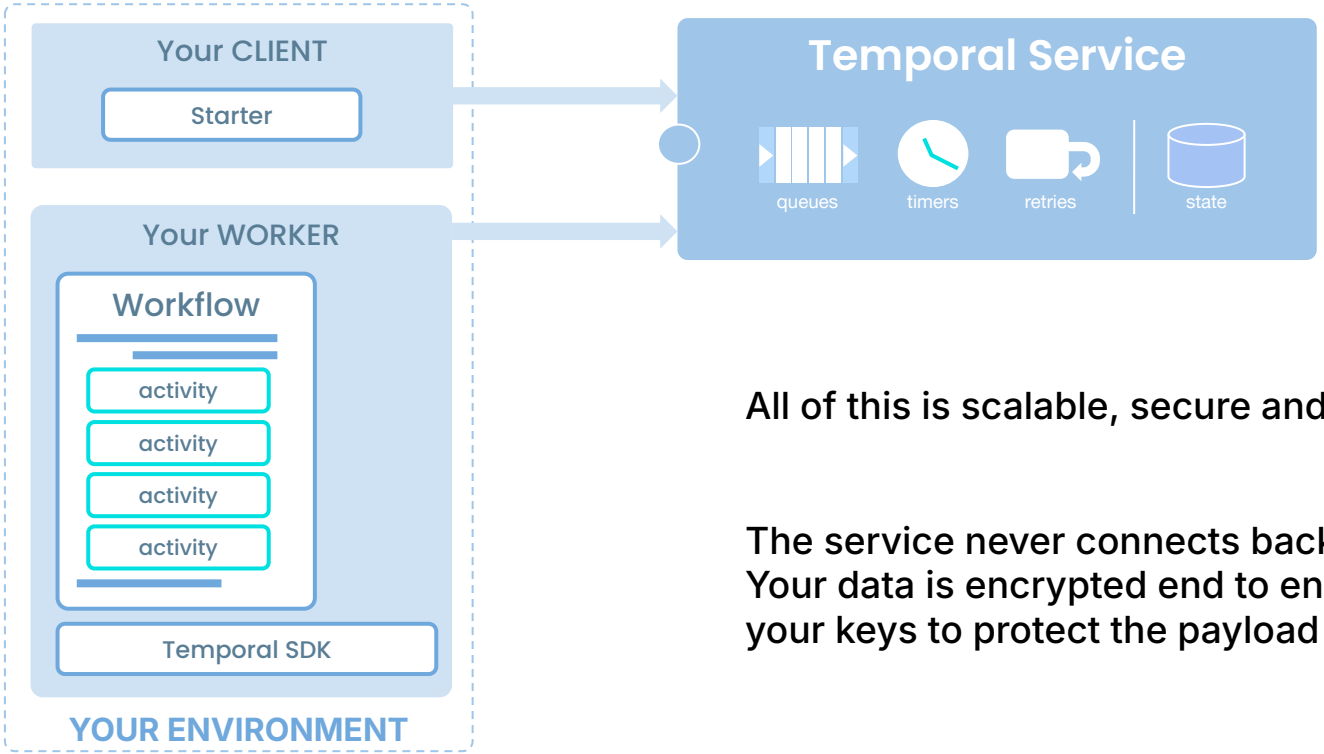
# MODERN ENGINEERING CHALLENGES

**TYPICAL MODERN APPLICATION:** 20% BUSINESS LOGIC, 80% PLUMBING


**WITH TEMPORAL:** 80% BUSINESS LOGIC, 20% PLUMBING


LESS CODE OVERALL

# HOW DOES IT WORK?

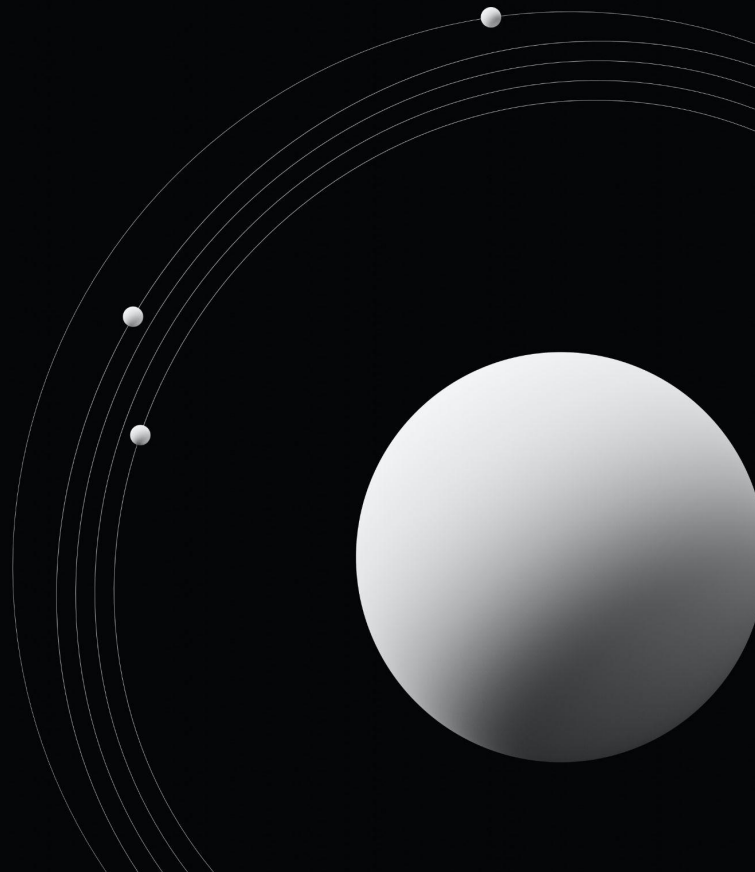



All of this is scalable, secure and reliable

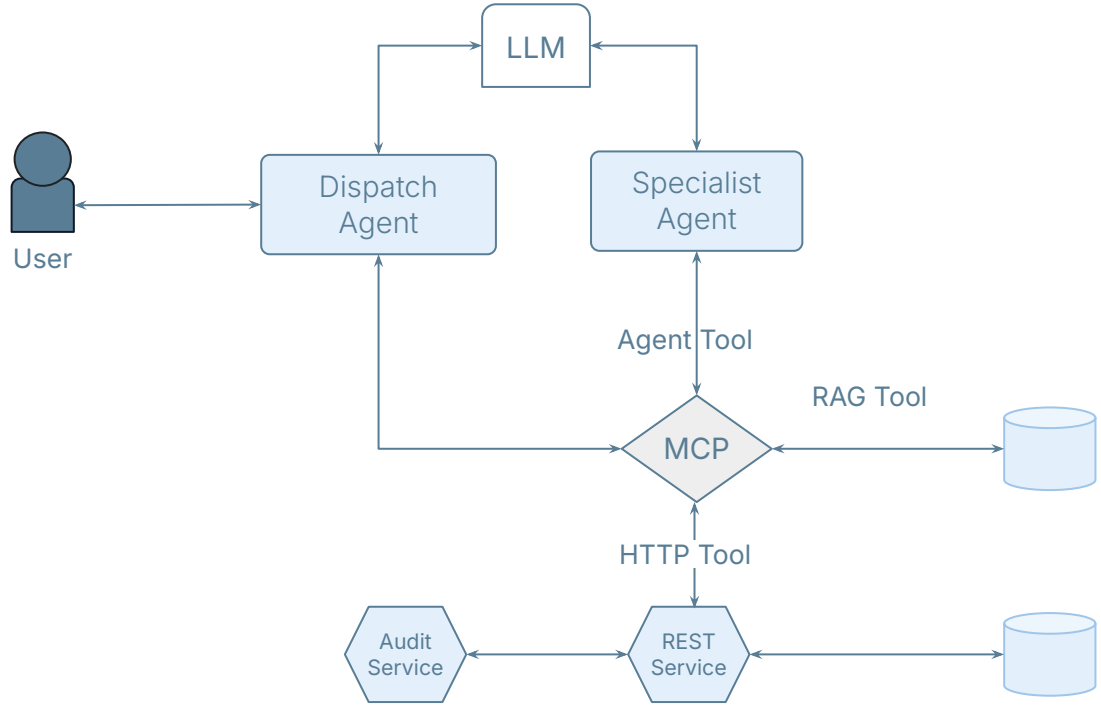
The service never connects back to your app.
Your data is encrypted end to end and you can use your keys to protect the payload data

# 2

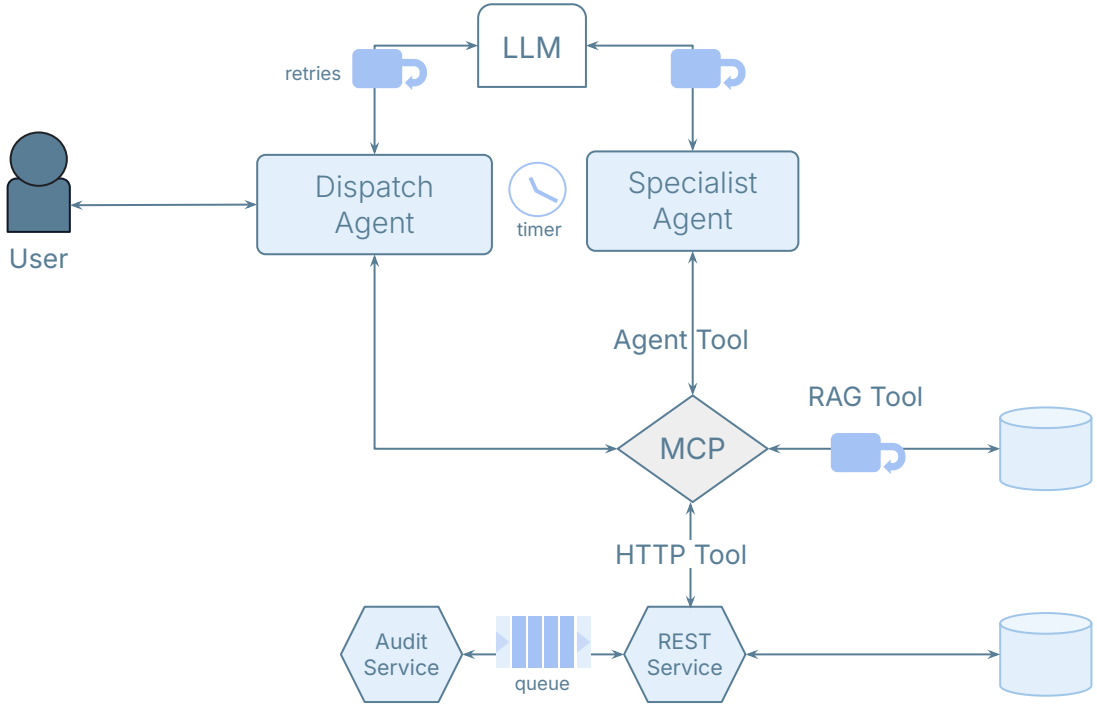
# AI Agents with Temporal



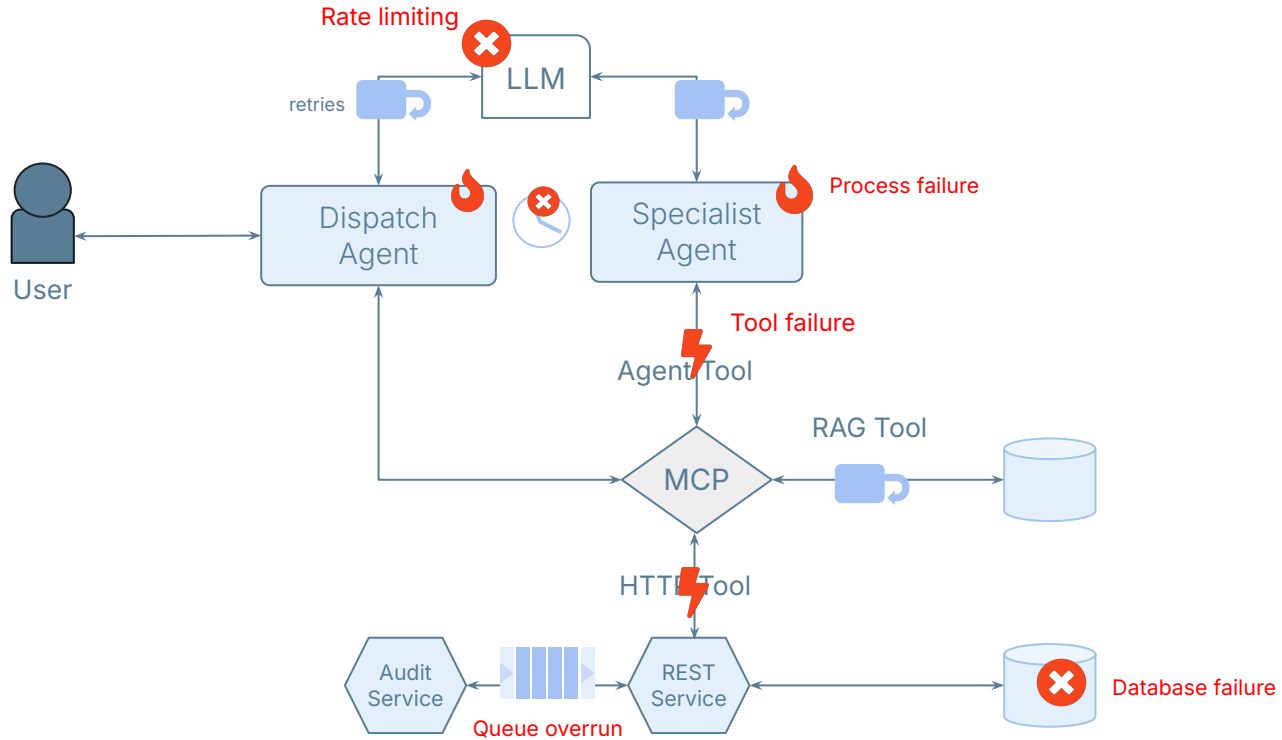
# AI agents are distributed systems in disguise



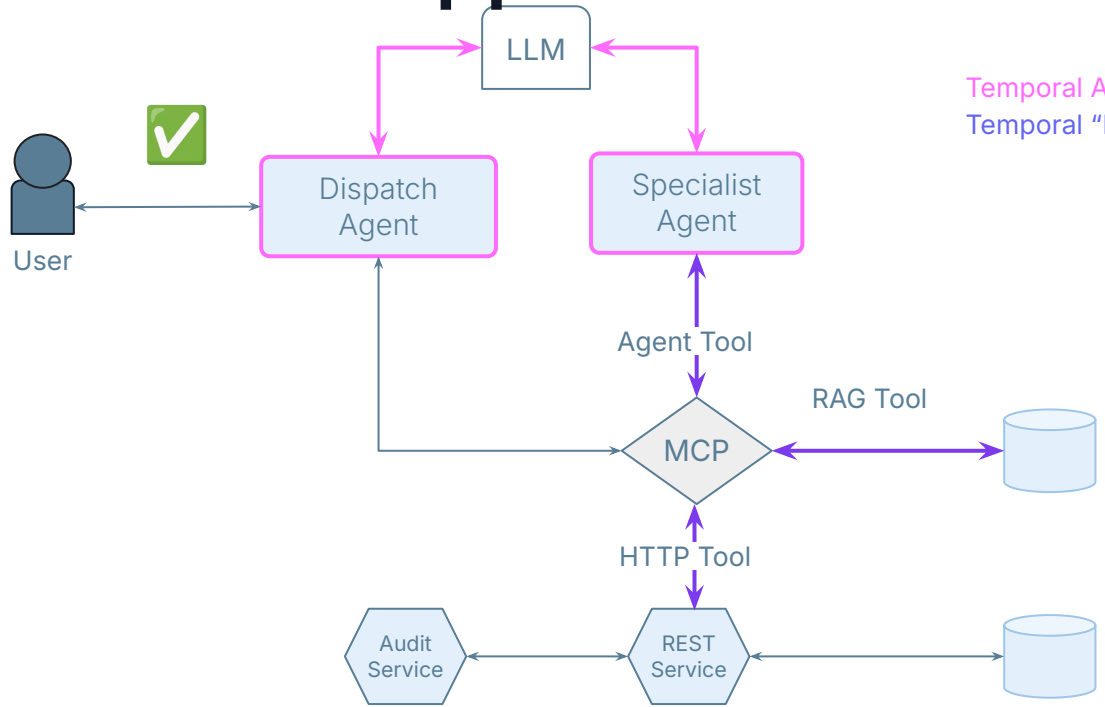
# AI agents need plumbing code for resilience & scale



# Everything fails anyway



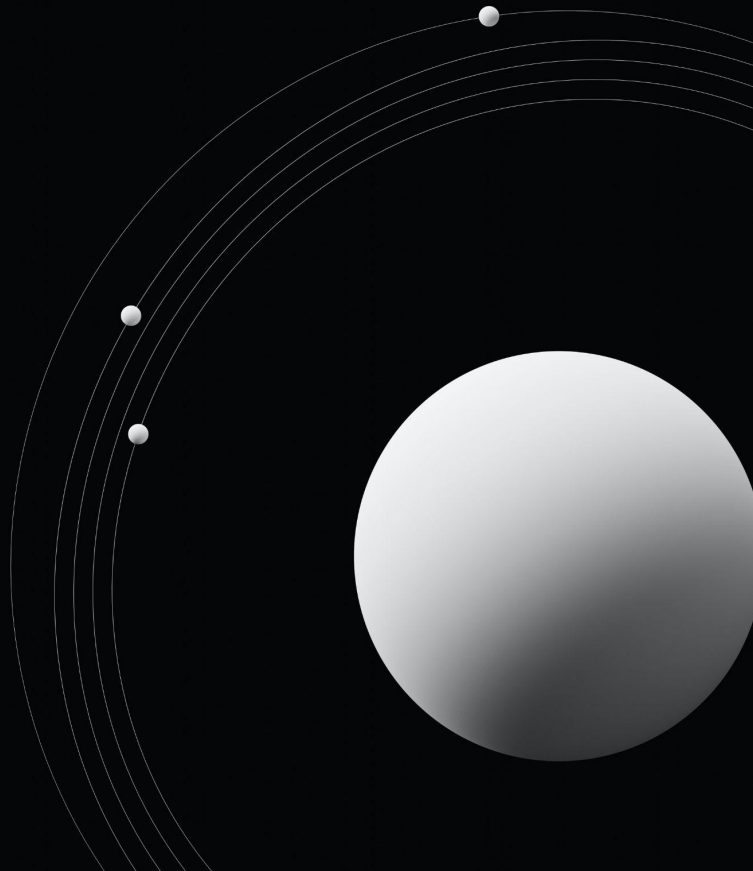
# Temporal is the most trusted platform for orchestrating reliable, scalable AI applications




Temporal Agent Workflows
Temporal “Durable Tool” Workflows

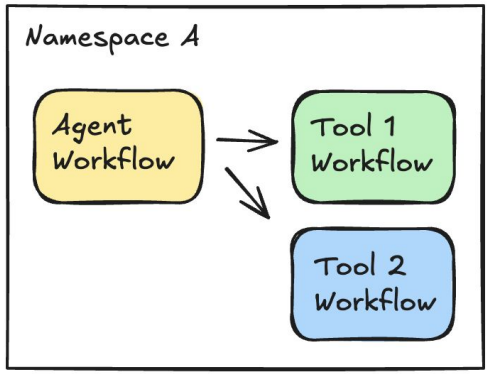
# 3

## Governance for AI Agents with Temporal Nexus



# Initial AI Agent Deployment - Day 1

**Single namespace**




**Decoupling**: Agent owns Tools

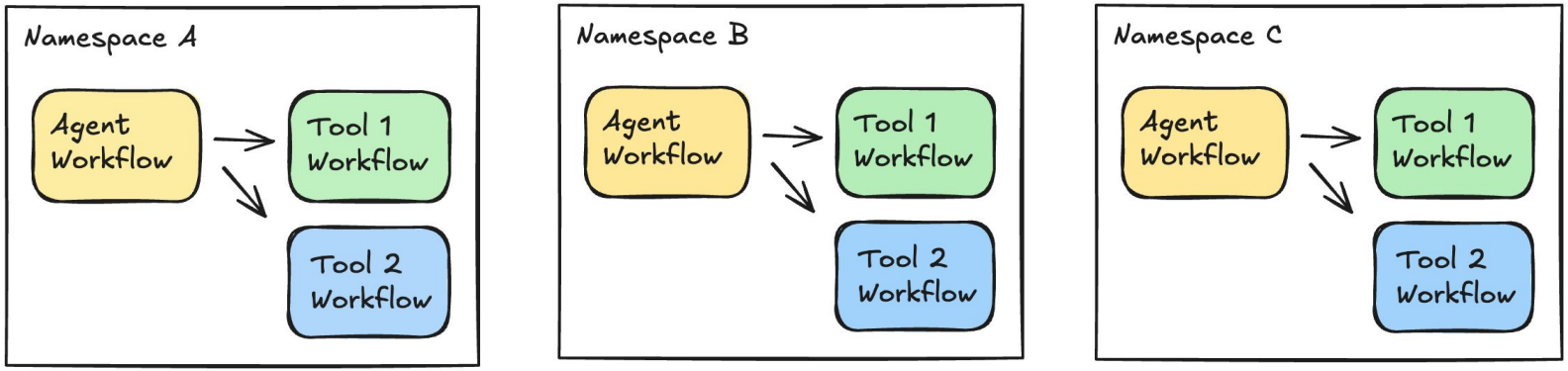
**Standardization**: Custom Tool workflows

**Language**: One language

Good starting point

# Broader Team Adoption - Day 2

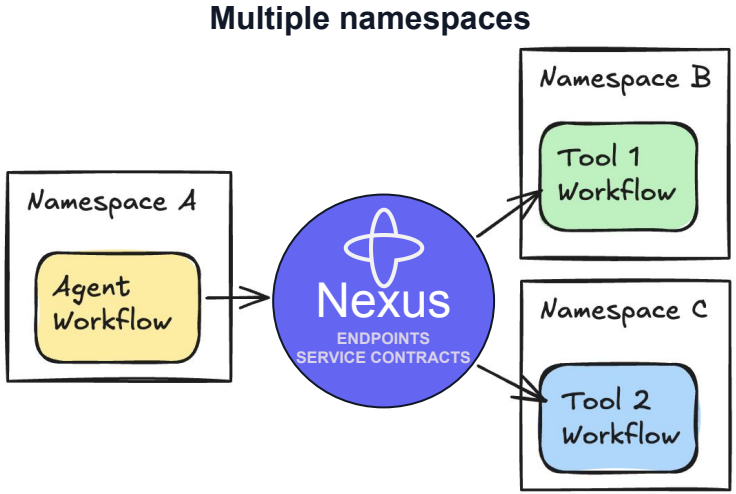
**Multiple namespaces**




**Challenges**:

- Code duplication
- Not sharing resources
- Lack of centralized governance

# Service Oriented Agent Architecture



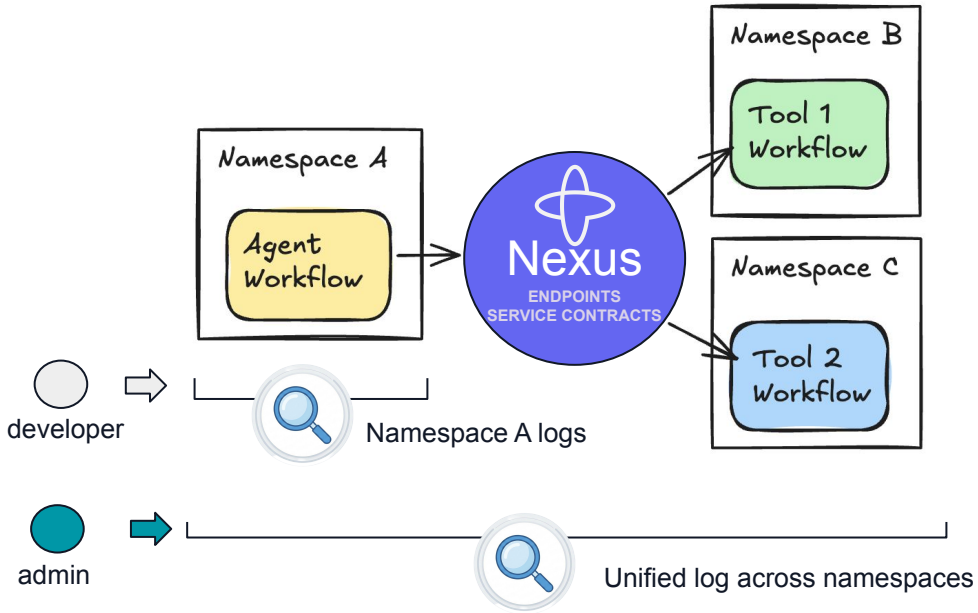

**Decoupling**: Agent requests Tools

**Standardization**: Reusable APIs

**Language**: Language agnostic

Better governance

# **Nexus**: Governance, Safety, Scale




**Zero trust**:

- Nexus endpoints can specify namespace allowlists
- Nexus allows observability across namespaces if user has sufficient privileges
- Nexus connects to namespaces through service mesh mTLS

**Blast radius:**

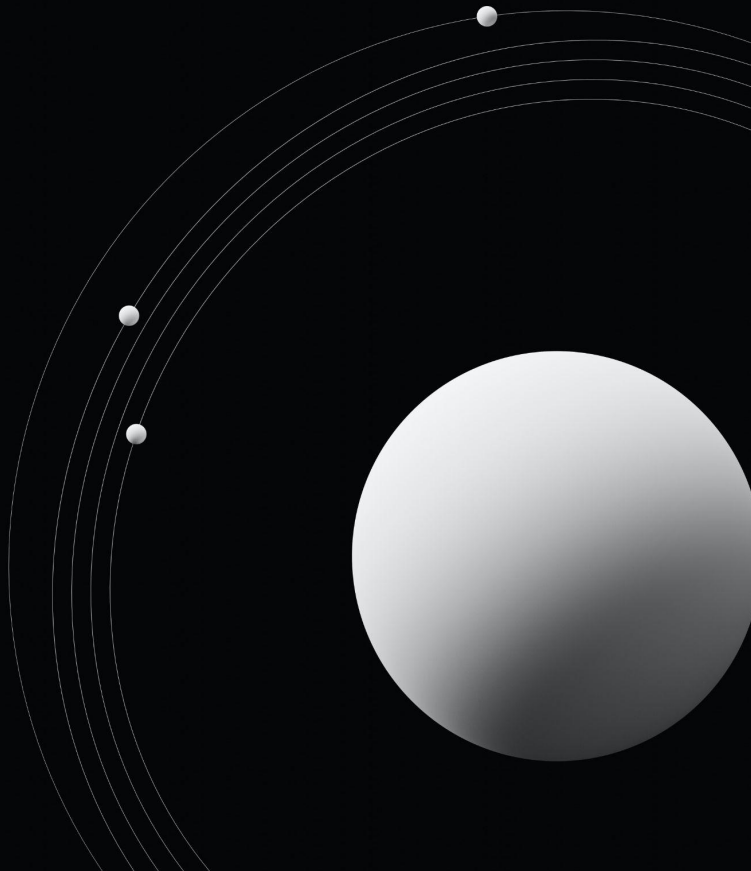
- Workflows are isolated and only Nexus Service Contracts exposed

**Durability:**

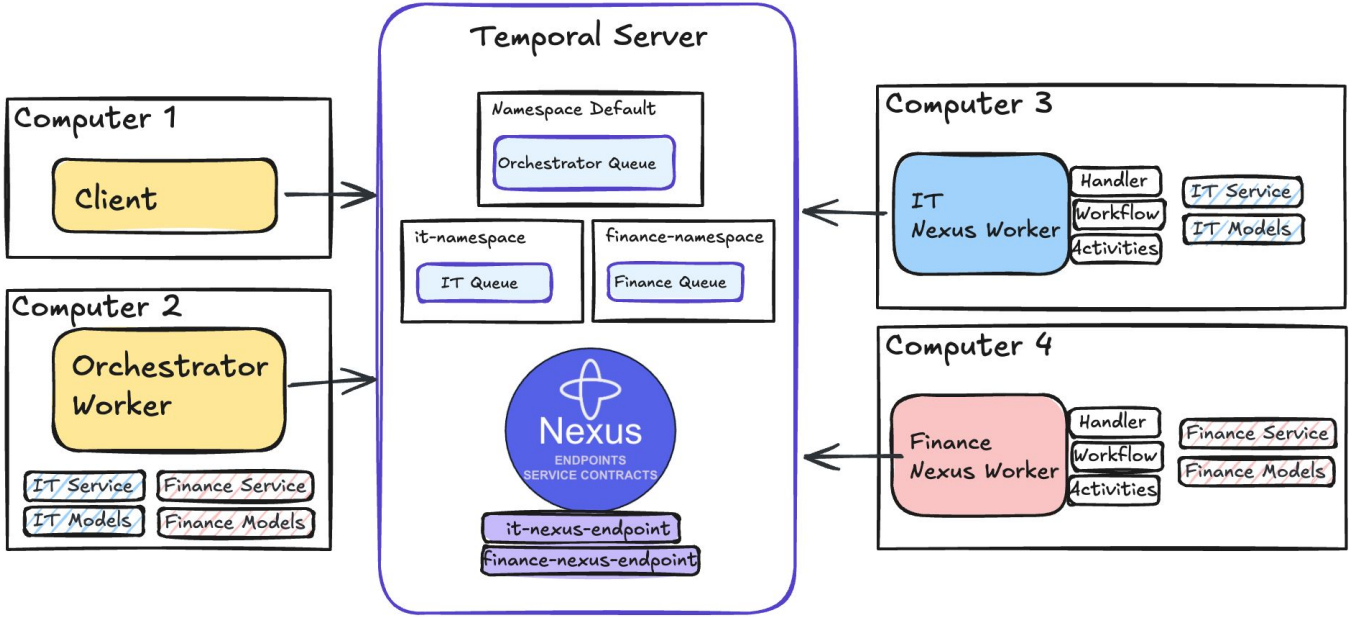
- Full Temporal durability guarantees

# 4

## Demo: litellm_temporal



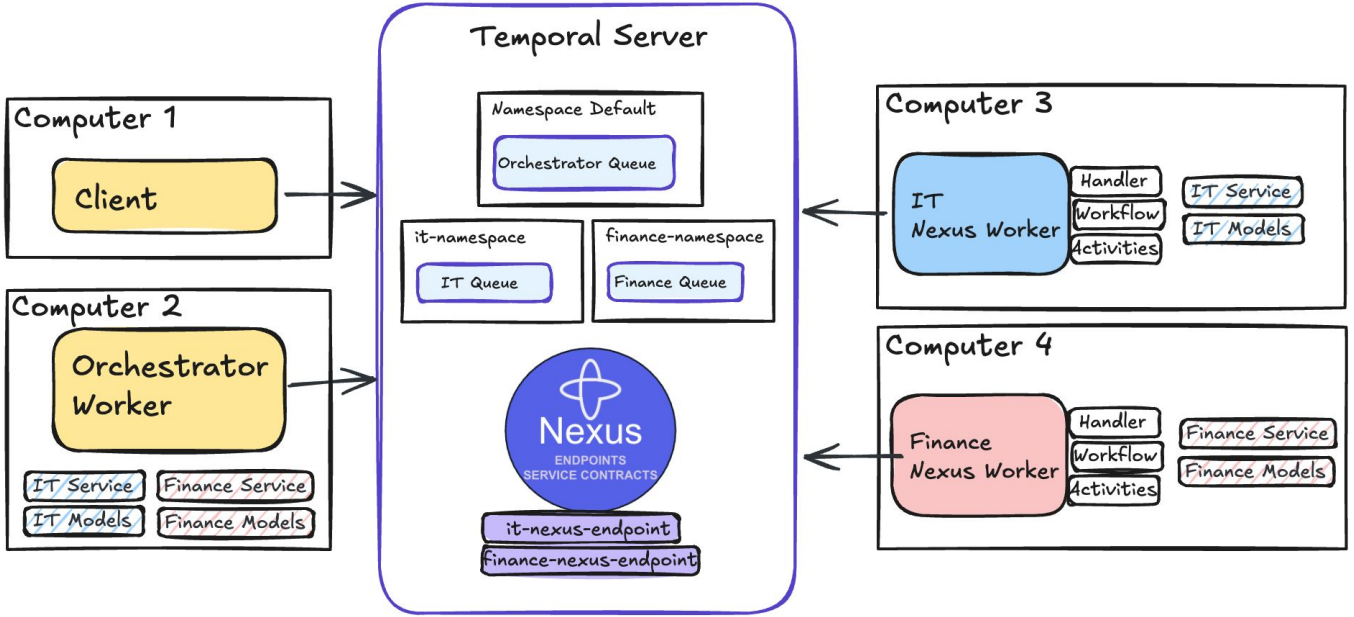
# Demo 1: litellm_temporal




**Demo Overview**
- AI Agent with pure python
- Nexus endpoints
- No MCP servers

**Takeaways**
- Litellm integration
- Direct calling Nexus endpoints
- Nexus service contracts

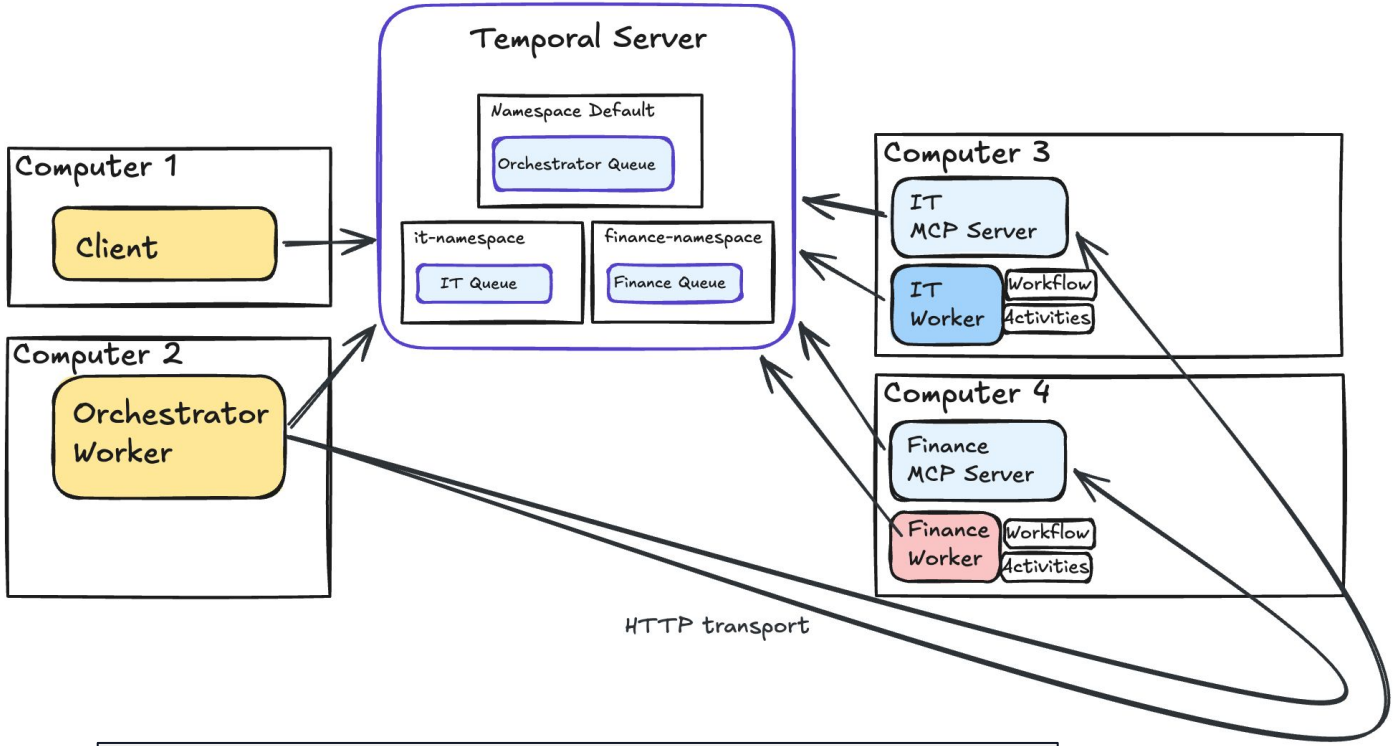
# Demo 2: openai_temporal




**Demo Overview**
- OpenAIAgents SDK Plugin
- Nexus endpoints
- No MCP servers

**Takeaways**
- Litellm integration
- OpenAI Agent SDK Nexus integration
- Nexus Service contracts

# Demo 3: openai_mcp_temporal

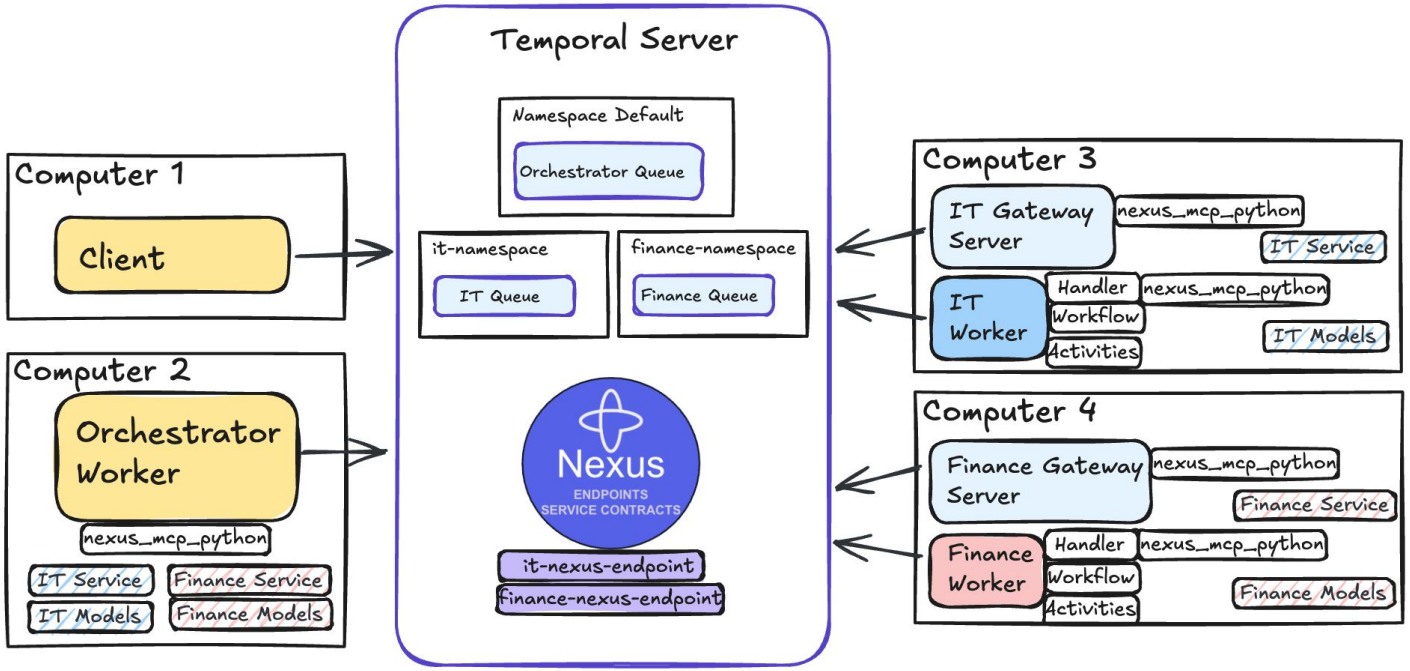



**Demo Overview**
- Dynamic tool discovery implemented by OpenAIAgentSDK
- MCP Servers both listen to http and behave like Temporal clients

**Takeaways**
- Simple
- No durability for http calls
- No Nexus auth federation/allowlist
- No bidirectional logs

*Orchestrator detects new tools dynamically because of OpenAI Agents SDK feature*

# Demo 4: nexus_mcp_temporal




**Demo Overview**
- nexus_mcp_python project
- MCP support with Nexus durability and security

**Takeaways**
- Advanced use case for when you need durability guarantees over MCP tool calls