March 14, 2026 | Chris Lema

Most AI Agent Systems Are Built Backwards

Most developers treat the LLM as the agent. That's the mistake. The Jido Framework flips the model: the agent is the actor, the LLM is a tool it picks up when it needs to reason. Here's why that changes everything.

AI · Claude Code · Product Work · Agents · For Engineers

Most people building multi-agent AI systems have made the same mistake.

They've treated the LLM as the agent.

It's not. And building as if it is, that's why these systems break in ways that feel impossible to fix.

Here's what I mean.

The Wall You've Likely Hit Already

If you're building with Claude Code right now, you can define custom subagents. You give each one a system prompt. You restrict which tools it can use. You define how it should behave.

That's real role definition. You can build a deployer agent that only has access to specific commands, governed by rules you wrote.

But those subagents are isolated.

They report to the parent. They can't talk to each other.

So when a subagent hits a wall, say it needs to know if the tests passed before it deploys, it has two options: escalate back to the parent, or fail. It can't ask the test agent directly. Everything goes up the chain. The main agent becomes the bottleneck.

There's another feature called Agent Teams, where teammates can message each other. That helps. But every teammate is spawned as a generic agent. Every one of them gets the full permission set of the

lead agent. You can't give the researcher read-only access while the implementer gets write access.

The "constitution" you wrote is a suggestion, not a wall.

So you end up with two broken options. Constitution without collaboration. Or collaboration without constitution. Not both at once.

What Jido Does Instead

The Jido Framework is built on Elixir and OTP. And it ships both things together.

In Jido, an agent's limits are built in, not written in. A Jido agent is a module with a schema, registered Actions, and signal routes. It literally cannot do things it doesn't have Actions for. It talks to other agents through a signal bus, not through a parent.

Here's the key difference. In Claude Code, the rules are written in plain language. You're asking the LLM to follow instructions. In Jido, the rules are compiled. The type system enforces them.

When a Jido deployer agent needs to know if the tests passed, it reads the test agent's results off the signal bus. No escalation. No bottleneck. The agents work it out sideways, the same way microservices do.

Think of it this way. One team where every decision has to go through the manager. Another team where everyone knows their lane and has

a shared scoreboard. One model creates bottlenecks. The other one scales.

What Happens When Things Break

When a Claude Code subagent fails, the parent LLM gets the error and has to figure out what went wrong. It burns through context trying to decide what to try next. The LLM handles recovery the same way it handles everything else: by reasoning about it in words.

That's a design problem disguised as a limitation.

When a Jido agent crashes, OTP supervision restarts it. The parent doesn't need to know. The supervision tree handles it. This is called "let it crash," and it has 30 years of production history in banking, telecom, and messaging systems.

Claude Code treats failure as a thinking problem. Jido treats failure as an infrastructure problem, and solves it with infrastructure.

You Can't Test Vibes

Here's the part that should bother every technical founder.

When your agent is the LLM, every test is a coin flip. Same prompt, different output. You end up writing fuzzy checks. Does the response contain the right keywords? Is the tone roughly right? You're not testing logic. You're testing vibes. And you're paying for API calls every time you run it.

Jido's core operation is a pure function. Same inputs, same outputs. Every time. No API call. No network. No randomness. Milliseconds per test run.

You can test every layer by itself. Actions are pure input/output with schema validation, unit tests as simple as any regular function. You can test agents directly: given this state and this action, verify the exact output. You can test signal routing without spinning up a single process. You can test every valid state transition, and verify that invalid ones get rejected.

How do you regression test a Claude Code agent after changing a prompt? You run it and hope. That's not a test suite. That's a prayer.

The testing story is a trust story. You can't ship agents to customers if you can't prove they work.

The Most Counterintuitive Thing About Jido

Jido's core framework has zero LLM dependency. The AI layer is a separate, optional package.

That's the thing most people miss coming from Claude Code or LangChain.

A Jido agent with no LLM attached can run multi-step workflows with validated state transitions. It can subscribe to events and route them to the right Actions. It can schedule messages on a delay and chain them with routing for fault-tolerant, process-aware cron jobs. It can spawn and monitor child agents. It can make decisions through

pattern matching. It can sit in chat rooms alongside humans and handle routing and permissions without generating a single AI response.

A Jido agent with no LLM is still a full actor. Supervised, message-driven, with validated state and crash recovery built in.

The LLM shows up when you need reasoning. When you need to interpret natural language. When you need judgment calls that can't be reduced to logic. Most agent work isn't that. Most of it is process-driven. The expensive thinking gets reserved for decisions that actually need it.

What This Means for What You Build

If you're building an AI-powered product, the question isn't whether to use an LLM.

The question is where to put it.

Most developers put it at the center. The LLM is the brain. Everything else is scaffolding around it.

Jido says: the agent is the actor. The LLM is a tool the actor picks up when it needs to think. That shift changes your testability, your crash recovery, your agent coordination, and your ability to ship something customers can trust.

The architecture is proven. Thirty years of OTP production history backs it up.

The question is whether you'll build on a foundation where the rules are a suggestion, or one where they're built in.

ABOUT THE AUTHOR

Chris Lema has spent twenty-five years in tech leadership, product development, and coaching. He builds AI-powered tools that help experts package what they know, build authority, and create programs people pay for. He writes about AI, leadership, and motivation.

AI IS MOVING FAST. YOU DON'T HAVE TO FIGURE IT OUT ALONE.

I help business leaders cut through the hype and put AI to work where it actually matters.

Let's Talk AI

CHRIS LEMA

Making Experts Dangerous

Tools

Your Voice Profile

Your Audience Segments

Your Content Agent

Your Platform Profile

Your Lead Magnet Agent

Content

Connect