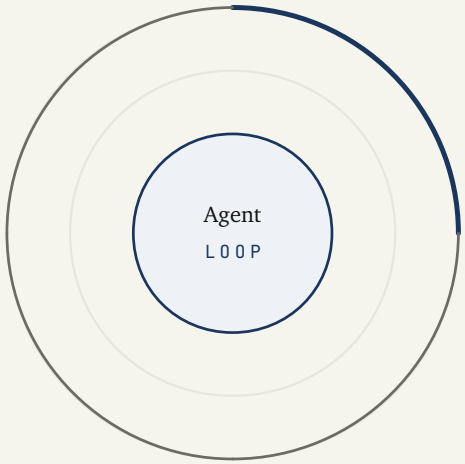
KEYNOTE · 2026

Things you don't know about Agents

Loops, harness, context, memory: what actually moves the needle in production.



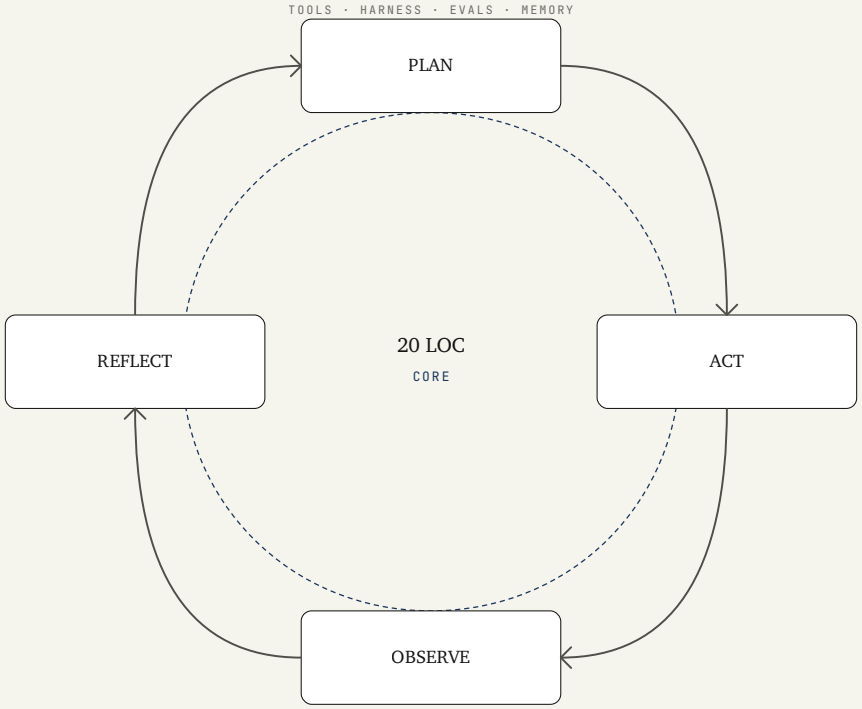
kami slides demo · A4 landscape · 7 slides

Simple core, complex surroundings

The loop is small. The infrastructure around it is what keeps it stable as features grow.

1. A working Agent loop fits in about 20 lines of code.
2. Control flow lives in the tools, not in branchy internal state.
3. Workflow vs Agent: predefined paths in code vs model picks paths at runtime.

> If your loop keeps growing every sprint, you are fixing the wrong layer. The tax is paid outside the loop.



Harness wins over hardware

More expensive models bring gains far smaller than expected. Verification, boundaries, feedback, and fallbacks matter more than model capability.

1. Upgrade the harness first. If accuracy does not move, then try the model.
2. Evaluation is the only honest signal. Test harness before test model.
3. Smaller model + strong harness routinely beats larger model + weak harness in production.

20

lines of code in a working Agent core loop

4

harness layers that matter: verify, bound, feedback, fallback

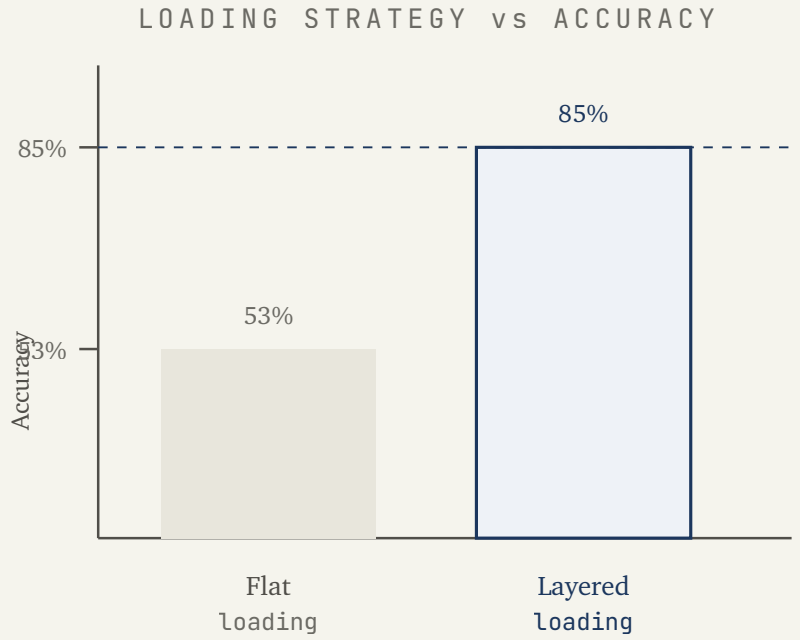
10×

velocity gains trace to execution discipline, not model swaps

Density beats length

Long context windows do not fix weak context design. Context Rot sets in around 300–400K tokens regardless of the model.

1. Layer the load: constant, on-demand, runtime, memory, system.
2. Index first, full content on demand. Beats dumping everything up front.
3. Stable prompt prefixes let caching actually pay off.
4. Every token that is not load-bearing is diluting signal.



Put state outside the context

Tools should match Agent goals, not underlying API shapes. Memory lives on disk, not in the window.

1. ACI principle: Agent-Computer Interface - design for what the Agent wants to do, not for the HTTP verb.
2. A bad tool description looks like a model failure until you re-read the description.
3. File-based state survives restarts. In-context state does not.

Four kinds of memory

1. Working: the context window - fast, expensive, temporary.
2. Procedural: SKILL.md files - how to behave, loaded lazily.
3. Episodic: JSONL logs - what happened, appendable.
4. Semantic: MEMORY.md - what to remember across sessions, consolidated at thresholds.

> Cross-session consistency needs explicit consolidation, not hope.

Pseudocode over syntax

Comments should outnumber code lines. The reader sees logic, not a language tutorial.

1. Write the intent first, then the implementation detail below it.
2. Variable names describe what a thing is, not what type it has.
3. One concept per block. Split ruthlessly.

```
# resolve tool call or decide to stop
function agent_step(context, tools):
  # model picks the next action
  action = model.think(context)

  # terminal condition: model says done
  if action.type == "stop":
    return action.result

  # delegate to the right tool
  result = tools[action.name](action.args)
  return agent_step(context + result, tools)
```

Protocol first. Then parallelism.

Fix your evals before you tweak the Agent. Most of what looks like model failure is infrastructure noise in disguise.