

WORK THAT SHIPS

— FIELD GUIDE · AGENTIC ENGINEERING

The workspace is the product.

Models are rented. Harnesses are swappable. What compounds is the place they work in: its instructions, memory, procedures, structure, and the colleagues beside them.

• SEVEN ACTS · ABOUT FORTY-FIVE MINUTES · PRESS → TO BEGIN, G FOR THE GUIDE

FILES

MEMORY

WORKSPACE

SKILLS

AGENTS

Title

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

This deck argues one thing: the quality of agent work is mostly decided outside the model. It is decided by the workspace the agent runs in: what it can read, what it can run, what it must not do, and what it leaves behind for the next session.

WHAT TO SAY

Open. Everyone here has watched a strong model do weak work because it started cold: no context, no rules, no memory of yesterday. This talk is about the thing around the model.

Thesis: models get swapped every few months and harnesses keep converging. The workspace is the one asset that compounds, so design it as a product.

HOW TO READ THIS GUIDE

Narration mode follows the deck. Each scene has a short explanation, the steps to build that piece yourself, a prompt you can paste into Claude Code or Codex, the skills that do the work, and references. Paragraphs marked with a beat light up as the diagram advances.

Start here. This opening scene names the through-line: a workspace is the durable operating context around a model, not a replacement for judgment or verification.

1. Use `→` to step, `G` to toggle this guide, `L` for the scene ledger.
2. Every prompt has a copy button. They assume a git repository you are allowed to change.
3. Paths like `AGENTS.md` are relative to the root of a workspace repository built this way. Skill paths such as `skills/knowledge/kg` are relative to the skills repository (cloned at `skills/` in this workspace).

SET IT UP

1. Start in a repository you are allowed to inspect.

2. Read its local instructions before proposing a change.
3. Keep a path and command receipt for every factual claim.

PROMPT

Read this repository's `AGENTS.md` or `CLAUDE.md`, then give me a one-screen map of what an agent can read, run, change and leave behind. Cite every item by path. Do not change files or run destructive commands.

SKILLS

- **checkit** `skills/research/checkit`, verify sources before they count
- **dogfood** `skills/governance/dogfood`, exercise a change and keep a receipt

REFERENCES

- [Effective context engineering for AI agents](#) (Anthropic)
- [Claude Code best practices](#) (Anthropic)

— THE THESIS

Give the agent a filesystem and a shell.

A model guesses when it can't look. A workspace lets it look, run, and write down what it found.

Chat only

Filesystem + shell

Explicit. Every input is a file you can open, diff and cite.

Open. Bash runs whatever the harness allows: search, tests, scripts.

Durable. What it learns is written back, so the next run starts ahead.

Structure and philosophy decide how it works there. That is harness engineering.



Give the agent a filesystem and a shell

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

A language model fills gaps with plausible text. Give it a filesystem and a shell and most gaps become lookups: it can open the file, run the test, grep the history, and write down what it found.

WHAT TO SAY

Chat only: the agent answers from weights and the prompt. Claims it can't check drift off and dissolve: that's what hallucination looks like from the outside.

Filesystem + shell: the same agent reads files, runs commands, and writes notes back. Answers leave with citations. Most of the rest of the talk is about how to structure that filesystem so the agent's context gets better every session.

WHAT YOU'RE SEEING



BEAT 1 OF 2

Chat only. Questions go in, answers come out. Some answers wander off the line and dissolve: claims nothing could check.



BEAT 2 OF 2

Filesystem and shell. The same agent reads from a workspace tree, runs commands in a terminal and writes a note back (gold). Answers leave with citations.

SET IT UP

1. Run the agent inside a repository, not in a chat window: Claude Code, Codex CLI or any harness with file and shell tools.
2. Put the context it needs in files it can find: an AGENTS.md at the root, docs next to the code they describe.
3. Allow read-only commands broadly (rg, git log, tests) and gate destructive ones (see "Hooks turn

recall into a reflex”).

4. Ask for citations: every claim should name the file or command output it came from.

PROMPT

Before answering, look. Search this repository with rg for the terms in my question, open the files that matter, and run any command that would confirm the answer. Cite each claim as path:line or as the command you ran. If something cannot be checked, say so instead of guessing. Write anything worth keeping to notes/FINDINGS.md.

SKILLS

- **checkit** skills/research/checkit, verify sources before they count
- **dogfood** skills/governance/dogfood, exercise the change and keep a receipt

REFERENCES

- [ripgrep](#)
- [Claude Code best practices](#)

— THE OPERATING MODEL

Seven questions. One system.

Each layer answers one question an agent asks before it can do useful work.

01

Workspaces

Where does it live?
Primer, policy, record
and sandbox.

02

Context

What reaches the
window, and how
hooks make recall a
reflex.

03

Second brain

How the graph is
structured, read and
grown.

04

Skills

How we do this kind
of work, loaded on
demand.

05

Structure

Which bytes it may
touch, and where
they go.

06

Fleets

Who else is here, and
who checks whom.

07

Leadership

What all of this asks
of people.

Agenda

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Seven parts, each answering one question an agent implicitly asks before it can do useful work.

WHAT TO SAY

Seven questions, in order: where does the agent live, what reaches its context, what does it know, how does it work here, what may it touch, who else is here, and what does this ask of us.

Each act answers one. The thread running through the tiles is the point: they are one system, not seven tools.

The map. The tiles are a reading order: establish the place, compose context, make knowledge readable, package procedures, isolate work, govern the fleet, then decide where human attention belongs.

1. **Workspaces:** where does the agent live?
2. **Context:** what reaches its window, and when?
3. **Second brain:** what does the place know, and how is it read and grown?
4. **Skills:** how is this kind of work done here?
5. **Structure:** which bytes may it touch?
6. **Fleets:** who else is working, and who checks whom?
7. **Leadership:** what does this ask of people?

The thread under the tiles is the argument: these are one system. A skill without the graph is guesswork; a graph without hooks is never read; a fleet without structure overwrites itself.

SET IT UP

1. Choose one small repository and one bounded task.
2. Map its instructions, active records, skills and checks before adding automation.
3. Improve one layer at a time and keep the evidence with the change.

PROMPT

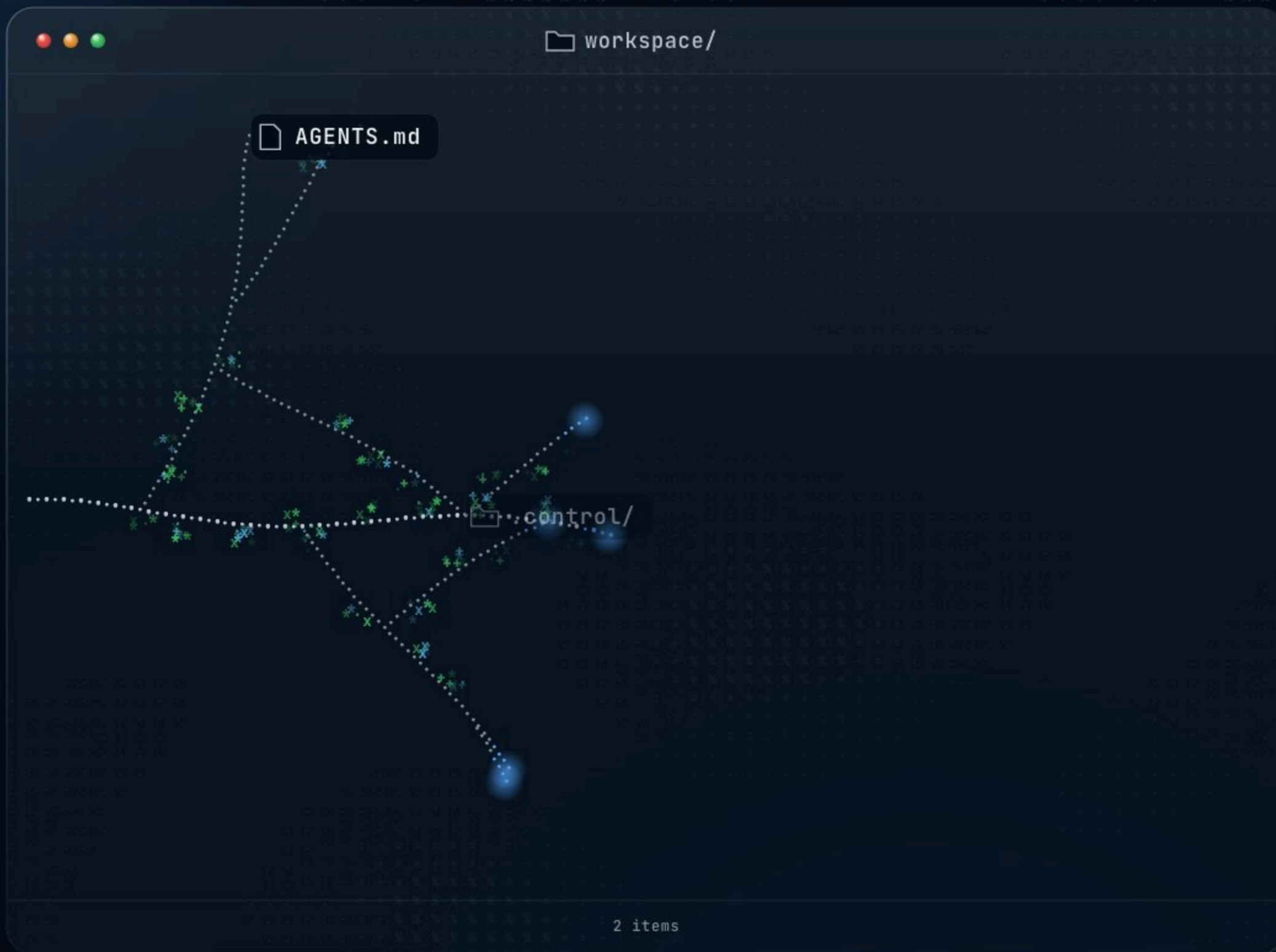
Assess this repository against seven layers: workspace, context, knowledge, skills, structure, fleet and leadership. For each layer, cite the evidence you found, name one missing or weak mechanism, and recommend the smallest safe next step. Do not edit files.

SKILLS

- **checkit** skills/research/checkit, establish evidence before recommendations
- **audit-harness-usage** skills/tooling/audit-harness-usage, measure token use and cost across harness traces

REFERENCES

- AGENTS.md, workspace routes and constraints
- docs/references/agent-operating-protocols.md, the operating primitives



ACT I

Workspaces

Where does the agent live?

Act I · Workspaces

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Act I establishes the place itself: what persists, which layers agents read, what stays swappable, and where data and records belong.

WHAT TO SAY

A fine vine grows through a workspace window: the workspace as a living thing. Its real files settle where the branches end, and a cursor visits each one.

Act I covers a folder versus a workspace, six workspace layers, the sandbox-harness-wire split, filesystem medallion layers and an architecture-only review-system example. Read next: A folder is not a workspace.

ACT SCENES

- 1. A folder is not a workspace
- 2. Six layers an agent reads before it writes
- 3. Sandbox, harness, wire
- 4. Medallion layers in the filesystem
- 5. Worked example: a review system

READ NEXT

Continue to **A folder is not a workspace** to distinguish a task directory from a place that retains governed context.

A folder is not a workspace.

The difference is what survives the session.

Agent workspace

Agentic workspace

● AGENT WORKSPACE

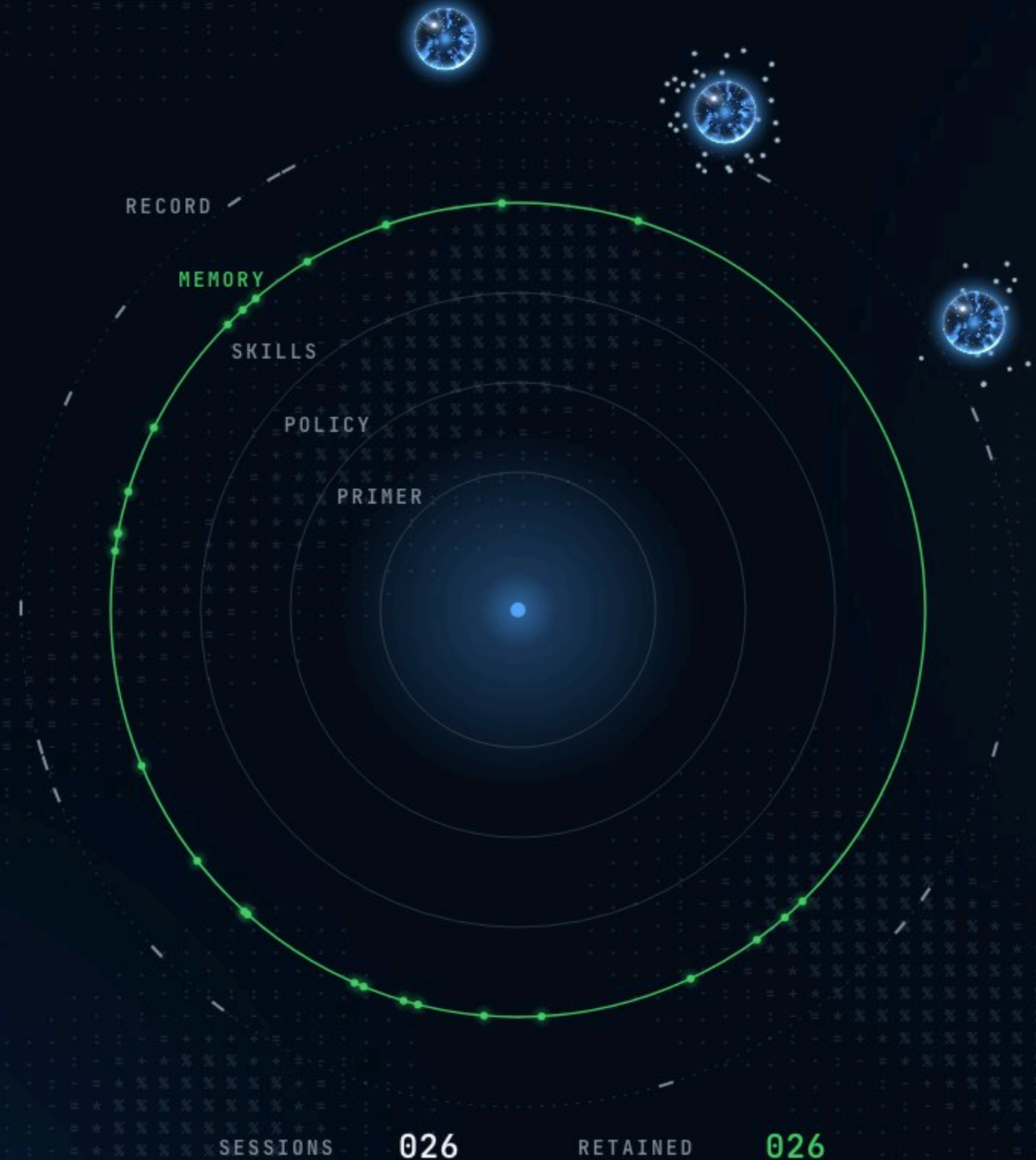
A working directory and a toolset for one agent on one task.

Useful. But context evaporates when the session ends, and the next agent starts cold.

● AGENTIC WORKSPACE

A governed place that many agents work through.

Instructions, policy, memory, skills and a record outlive every session, so each run leaves the place a little smarter.



A folder is not a workspace

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

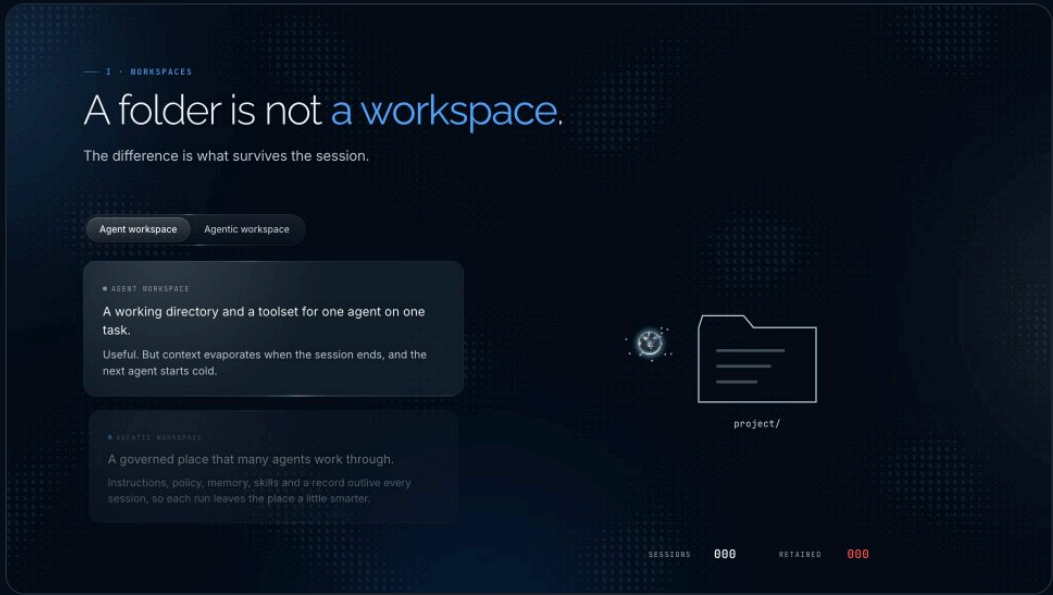
An agent workspace is a folder and a toolset for one agent on one task. An agentic workspace is a governed place many agents work through, where instructions, policy, memory, skills and a record outlive every session.

WHAT TO SAY

Left state: one agent, one folder. Watch the counter: sessions go up, retained knowledge stays at zero. Every run re-derives the same context.

Right state (→): the same agents now dock into rings: primer, policy, skills, memory, record. Each session drops something into memory before it leaves. That accumulation is the whole game.

WHAT YOU'RE SEEING



BEAT 1 OF 2

Agent workspace. One agent docks at a folder, builds context, and loses it when the session ends. The retained counter stays at zero.



BEAT 2 OF 2

Agentic workspace. Agents dock into rings (primer, policy, skills, memory, record). Each leaves a deposit in memory and a tick in the record. Retained grows with sessions.

SET IT UP

1. Make one repository the place: it holds instructions, policy, knowledge, skills and the session record.
2. Decide what every session must leave behind: a session log, a decision, a note with sources.
3. Make leaving it automatic (a Stop hook, "Hooks turn recall into a reflex"), not a matter of remembering.

PROMPT

Turn this repository into an agentic workspace. Propose (do not write yet) the minimum layout: an AGENTS.md primer, a policy file for hard gates, a folder for knowledge entries with front matter, a skills folder, and a docs/conversations folder for session records. For each, give its path, its one job, and what writes to it. Keep it to one screen.

SKILLS

- **handoff** skills/orchestration/handoff, write a useful session handoff
- **dogfood** skills/governance/dogfood, exercise a change and keep a receipt

REFERENCES

- AGENTS.md, workspace primer

- docs/conversations/Conversations.md, session record index

Six layers an agent reads before it writes.

● LAYER 01

Primer

AGENTS.md ≡ CLAUDE.md

Philosophy, routes and hard boundaries. One file, byte-identical copies for every harness.

RULE Route, don't dump. Point to protocols instead of inlining them.

READ ORDER ↓

01 Primer

AGENTS.md ≡ CLAUDE.md

02 Policy

.control/policy.yaml

03 Skills

skills/<category>/<name>/

04 Knowledge

research/entities/ · knowledge-index.md

05 Record

docs/conversations/ · .control/initiatives/

06 Work

repos · worktrees · sandboxes

WRITE PATH ↑

Anatomy of a workspace

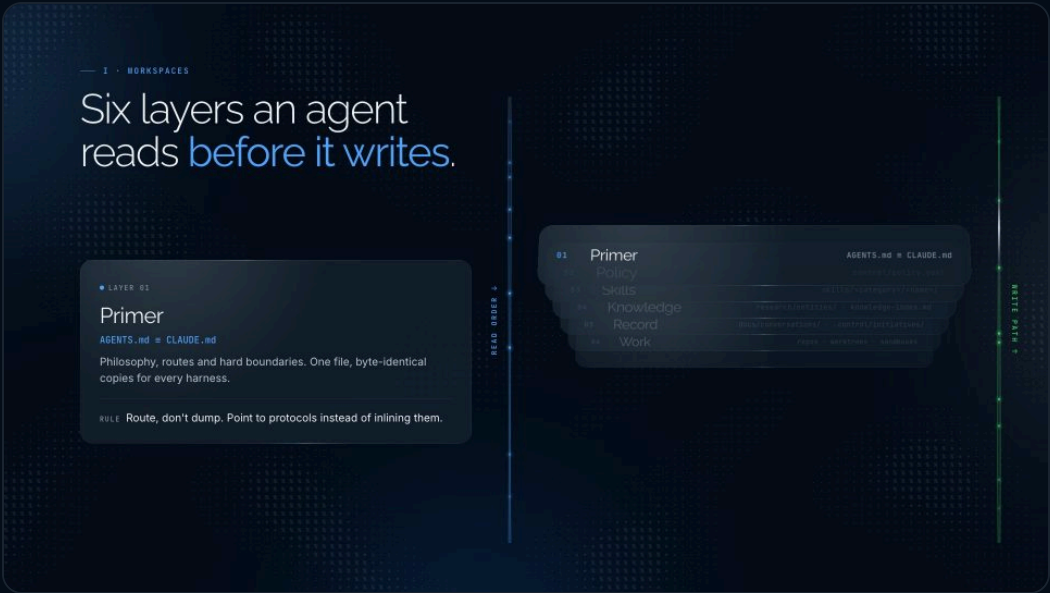
Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Six layers, read top-down before the agent writes anything, written bottom-up as work happens.

WHAT TO SAY

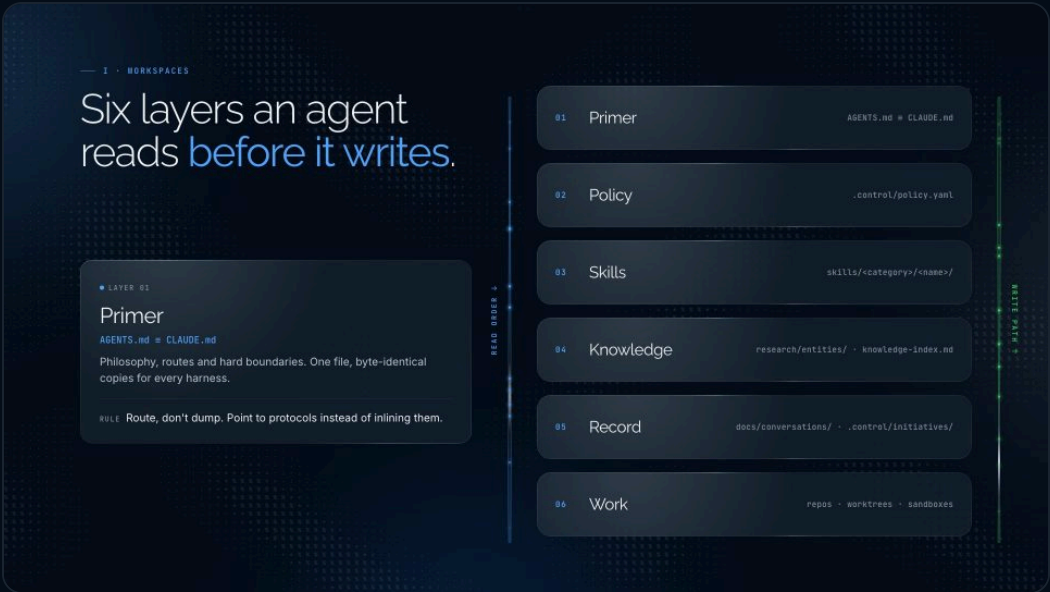
Read top-down, write bottom-up. The primer routes; it does not dump everything into context. Policy is data that hooks and CI enforce. Skills and knowledge load on demand. The record says what is in flight right now. Only the bottom layer is where bytes actually change.

Hover a plate. The rule under each is the one-line discipline that keeps that layer honest.



BEAT 1 OF 2

Read path. The first three layers constrain and route the work before tools run: primer, policy and on-demand procedures.



BEAT 2 OF 2

Write path. Knowledge, the work record and the product surface collect evidence after the agent has acted.

THE LAYERS

1. **Primer** AGENTS.md $\equiv$ CLAUDE.md: philosophy, hard boundaries and read routes. Route to protocols instead of inlining them, so it stays short enough to be read every time.
2. **Policy** .control/policy.yaml: gates as data that hooks and CI enforce.
3. **Skills** skills/<category>/<name>/SKILL.md in the skills repository: procedures loaded on demand.
4. **Knowledge** research/entities/ plus a compiled docs/knowledge-index.md.
5. **Record** docs/conversations/ and .control/initiatives/: what happened and what is in flight.
6. **Work**: product repositories, worktrees and sandboxes, where bytes change.

SET IT UP

1. Keep AGENTS.md canonical and copy it to CLAUDE.md (cp AGENTS.md CLAUDE.md), so every harness reads the

same primer.

2. Write the primer as a routing table: "when X, read Y". Measure it: if it is longer than a screen or two, move detail into the routed files.

PROMPT

Draft AGENTS.md for this repository as a routing primer, under 300 lines. Sections: philosophy (4 rules), non-negotiable boundaries, a "when / read" table pointing at real files in this repo, and completion checks (the exact commands to run before calling work done). Link to protocols; do not inline them. Then show me the table of files it routes to and whether each exists.

SKILLS

- **role-x** skills/orchestration/role-x, route work to the right local context
- **kg** skills/knowledge/kg, load graph context by topic

REFERENCES

- [AGENTS.md convention](#)
- [Claude Code memory \(CLAUDE.md\)](#)

— I · WORKSPACES

Sandbox, harness, wire.

Three layers, three different answers. Buy the box, swap the brain, and don't bet on the wire yet.

Claude Code

Codex

Gemini CLI

Your own

UNSOLVED

Wire

phone · desktop · chat



ACP: stdio-first · remote transport in draft

SWAP

Harness

the agent loop



your own loop

BUY

Sandbox

microVM · persistent volume

microVM

/workspace (volume)

exec · files · tools

MCP SERVER + CLI

the integration point never moves when the harness does

Sandbox, harness, wire

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

An agent runtime splits into three layers with three different build-or-buy answers.

WHAT TO SAY

Sandbox: isolated microVMs with a persistent volume are a commodity now. Buy them.

Harness: the agent loop. Click through the harnesses: the integration rail underneath (an MCP server plus a CLI) never moves. If you integrate through one harness's internals, you've married it.

Wire: how a human on a phone or a desktop talks to a remote agent. The agent-client protocol (ACP) is stdio-first; its remote transport is still a draft. Keep that layer thin and replaceable.

- **Sandbox:** an isolated VM with a persistent volume. Buy it.
- **Harness:** the agent loop (Claude Code, Codex, Gemini CLI, your own). Keep it swappable.

- **Wire:** how a person on a phone or desktop reaches a remote agent. The agent-client protocol (ACP) is stdio-first; its remote transport is still a draft. Keep it thin.

WHAT YOU'RE SEEING



BEAT 1 OF 4

Claude Code. One harness around the same filesystem, tools and integration rail.



BEAT 2 OF 4

Codex. A second harness proves why the integration should not depend on one product's internals.



BEAT 3 OF 4

Gemini CLI. A third harness uses the same stable surfaces: files, a CLI and MCP.



BEAT 4 OF 4

Your own. The loop can change without replacing the sandbox or the wire.

SET IT UP

1. Expose your system through an MCP server (for remote, authenticated access) and a CLI (for local, scriptable access). Every harness can call both.
2. Do not build on one harness's internal APIs. A mod or plugin that only one harness runs ties you to it.
3. Pick a sandbox with snapshots and a persistent volume, and treat its disk as the agent's working memory.

PROMPT

List every place this project integrates with a specific agent harness (config files, plugins, hooks, SDK calls). For each, say whether the same capability could live behind an MCP server or a CLI that any harness can call, and what moving it would take. Do not change anything.

SKILLS

- **harness-engineering-playbook**
skills/governance/harness-engineering-playbook, evaluate harness boundaries

REFERENCES

- [Agent Client Protocol: transports](#)
- [Model Context Protocol](#)
- [Vercel Sandbox](#)
- [E2B](#)
- Entity: research/entities/concept/agent-harness-three-layer-split.md

Medallion layers live in the filesystem.

Each workspace keeps its own bronze, silver and gold as plain folders. A shared ledger records what was decided.

One workspace

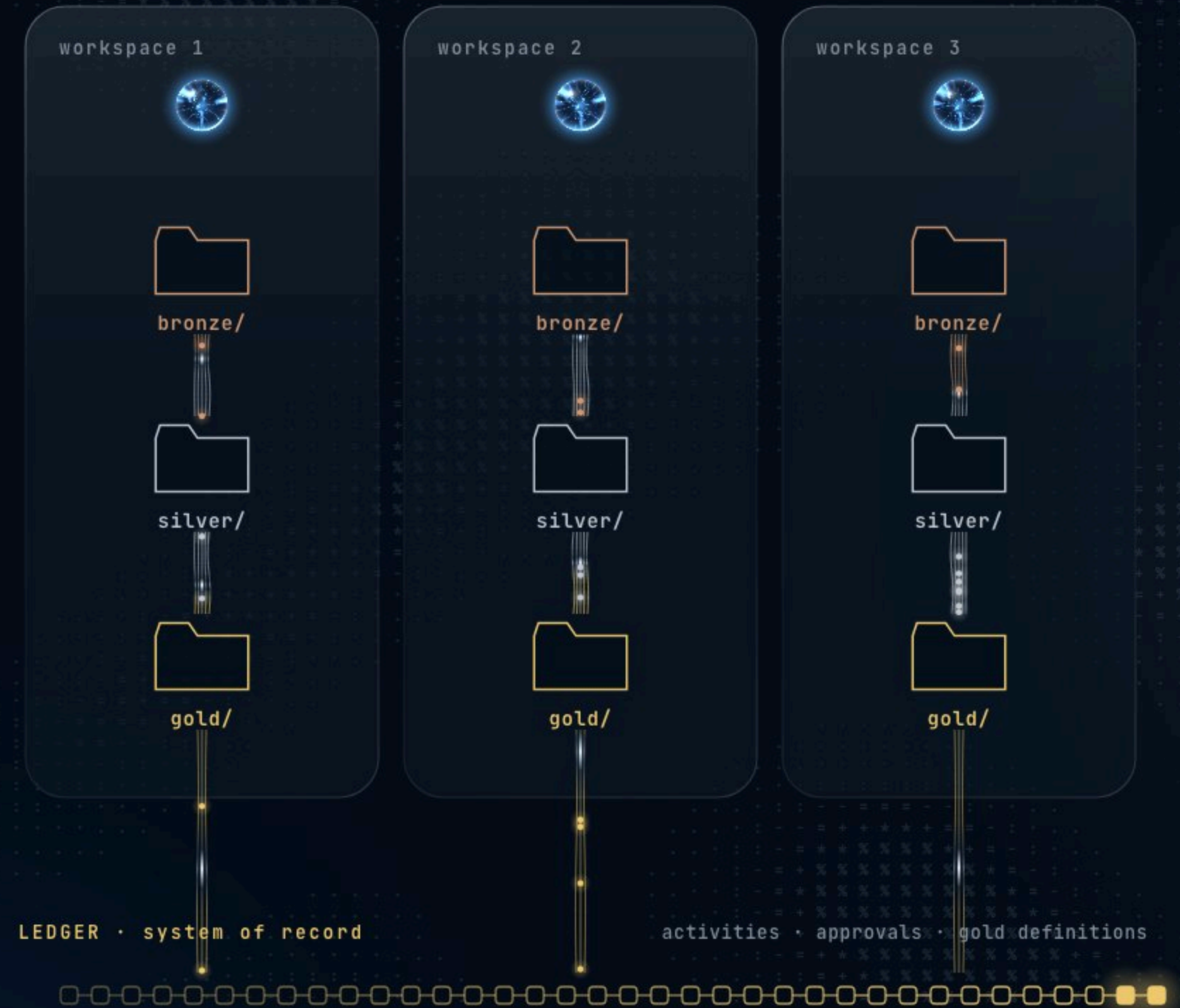
Many workspaces

bronze/ Inputs as received. Append-only, hashed, never edited.

silver/ Typed and validated. Every value cites its bronze source.

gold/ Decisions and definitions the organization trusts.

Files are the agent's working memory. The ledger is the organization's.



Medallion layers in the filesystem

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

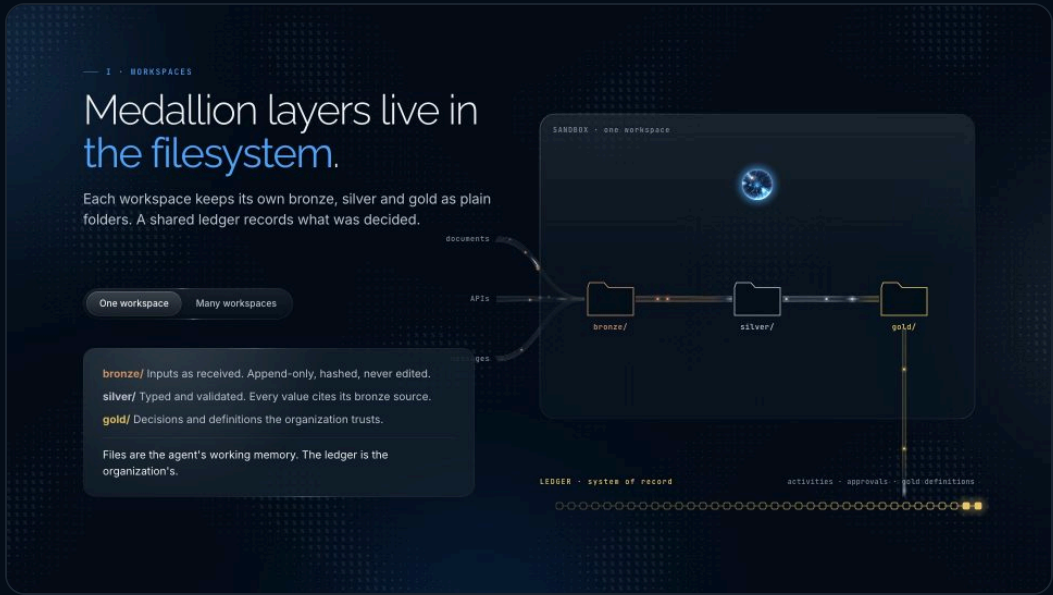
Agents are best at files: grep, diff, read, write. So the data layers live as folders inside each workspace's filesystem, and only the record leaves.

WHAT TO SAY

The agent works in files because files are what agents are best at: grep, diff, read, write. So the medallion layers live inside the workspace's own filesystem, in its sandbox, next to its own ontology.

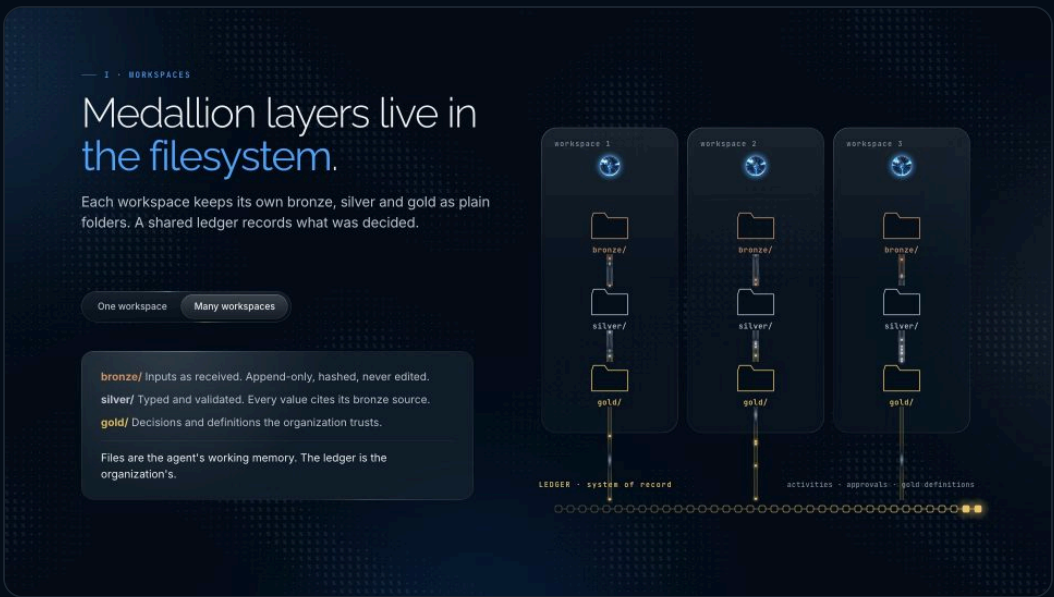
What leaves the sandbox is the record: activities, approvals and gold definitions go to one shared ledger. Toggle to many workspaces: each is sovereign over its files, and all of them write to the same system of record.

WHAT YOU'RE SEEING



BEAT 1 OF 2

One workspace. Inputs land in `bronze/` untouched. The agent produces typed, cited records in `silver/`. Decisions and definitions land in `gold/`, and each one appends a block to the ledger.



BEAT 2 OF 2

Many workspaces. Each sandbox keeps its own medallion folders and ontology. All of them write to one shared ledger, the system of record.

SET IT UP

1. Make `bronze/` append-only and hash every input on arrival.
2. Require every `silver/` value to cite its bronze source (file, page, span).
3. Write to the ledger from code, not from the model: activities, approvals, gold definitions.

PROMPT

Design a medallion layout for this workspace's data as folders: bronze/ (inputs as received, append-only, sha256 recorded), silver/ (typed records, each with a citation back to bronze), gold/ (decisions and definitions). Specify the ledger table that records activities, approvals and gold definitions, and which component writes each row. Output a tree and a table schema.

SKILLS

- **checkit** skills/research/checkit, preserve source evidence during intake
- **cross-review** skills/governance/cross-review, review the boundary design independently

REFERENCES

- docs/references/workspace-context.md, privacy and storage boundaries

- research/entities/concept/agent-harness-three-layer-split.md, the runtime boundary

A review system, built on the pattern.

Sandbox

Readers

Judges

Ledger

• BEAT 04 · LEDGER

The ledger is the system of record

Activities, approvals and gold definitions go to one shared Postgres ledger outside every sandbox. Files are the working memory; the ledger is what the organization trusts.



Worked example: a review system on the pattern

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

An architecture pattern from a real review system, with the domain removed: one sandbox per case, readers that only extract, code that decides, a ledger that remembers.

WHAT TO SAY

Architecture only, no domain specifics. One sandbox per case with the medallion layers as folders. The harness agent runs the review skill: the skill is the workflow. One reader subagent per document turns untrusted text into typed claims with quotes. Code verifies every quote and judges; a claim that can't be verified stays claimed and can only push the outcome toward manual review.

The shared Postgres ledger, outside the sandbox, records activities, approvals and gold definitions for every workspace. Step through the four beats.

WHAT YOU'RE SEEING



BEAT 1 OF 4

Sandbox. One microVM per case with a persistent volume. The medallion folders live inside it. The agent may write only scratch, notes and proposals.



BEAT 2 OF 4

Readers. The harness agent runs the review skill (the skill is the workflow) and launches one reader subagent per document. Each returns typed claims with verbatim quotes and locations.



BEAT 3 OF 4

Judges. Code checks every quote against its source. An unverifiable claim stays a claim and can only push the outcome toward manual review. No decider sees raw document text.



BEAT 4 OF 4

Ledger. Activities, approvals and gold definitions go to one shared Postgres ledger outside every sandbox.

WHY IT HOLDS

Untrusted text never reaches the component that decides. The model's job is reading; the code's job is judging; the ledger's job is remembering. Each boundary can be tested on its own.

SET IT UP

1. Give each case an isolated working directory and read-only input mount.
2. Make extraction output typed claims with source locations.
3. Verify claims with deterministic code before rules or approval use them.
4. Append activities and approvals outside the case workspace.

PROMPT

Write a review skill for documents in bronze/. For each document, launch one reader subagent with read-only tools. It returns claims as JSON: {field, value, quote, page, bbox}. Then run a deterministic verifier that checks each quote occurs verbatim in the source; unverified claims keep status "claimed". Only verified claims reach the rules. Record every step in the ledger. Show the skill file and the verifier's tests.

SKILLS

- **skillify** skills/tooling/skillify, turn a proven procedure into a tested skill
- **cross-review** skills/governance/cross-review, a different model reviews the verifier

REFERENCES

- research/entities/concept/agent-harness-three-layer-split.md, the architecture boundary
- docs/references/workspace-context.md, privacy and workspace limits

ACT II

Context

What reaches the window, and when?

Act II · Context

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Act II explains what reaches the model window, how to make recall automatic, and when to use files, graph links, vectors or live tools.

WHAT TO SAY

A black hole drawn in text: everything the model knows is what falls into the window this turn.

Act II covers context composition, hooks that make recall automatic and a retrieval ladder from files to live systems. Read next: How an agent's context is built.

ACT SCENES

- 1. How an agent's context is built
- 2. Hooks turn recall into a reflex
- 3. Files first. Vectors when words diverge.

READ NEXT

Continue to **How an agent's context is built** to identify the always-on layers before adding more context.

How an agent's context is built.

Everything the model knows this turn is what reached the window. The workspace decides what reaches it.

Bare session

Workspace session

Hover any layer of the window. Each one has a source on disk, a moment it loads, and a budget.

Less guessing is the product of more grounded layers, not of a bigger model.

Harness system prompt
built into the harness

Primer
CLAUDE.md ≡ AGENTS.md

Memory index
MEMORY.md → one file per fact

SessionStart hooks
board facts · freshness · metrics

Skill descriptions
one line per installed skill

Your message
what you typed

Intake hook
lens · graph constraints · entities

Tool results
files read · bash output

Conversation
prior turns

CONTEXT WINDOW

guessing

every turn

session start

session start

session start

session start

each turn

each prompt

while working

grows each turn

more grounded layers, less guessing

proportions illustrative, not measured

How an agent's context is built

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

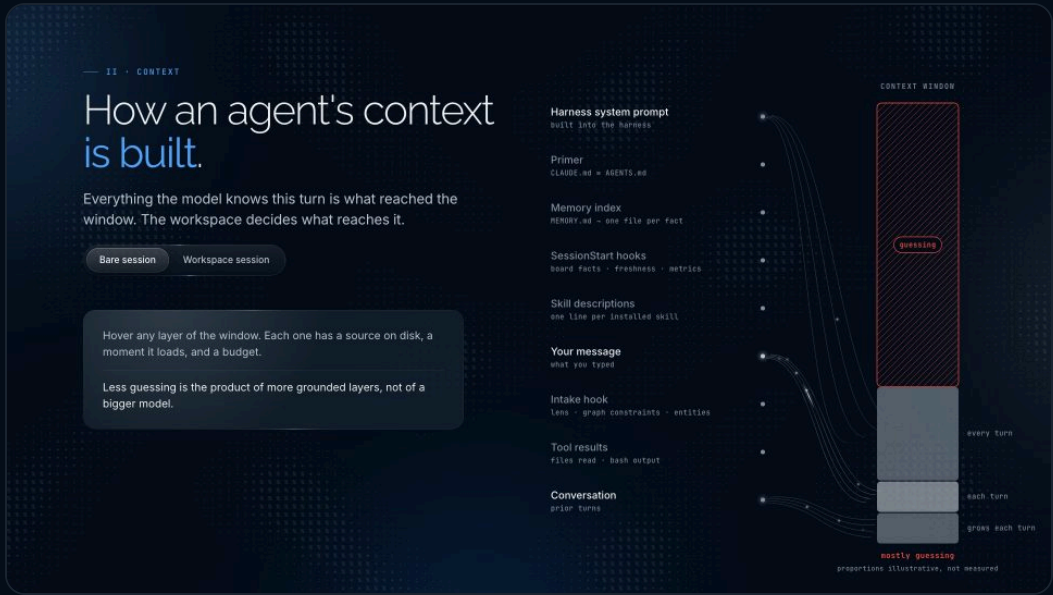
The model knows only what reached its context window this turn. A workspace decides what reaches it, from where, and when.

WHAT TO SAY

Bare: the harness system prompt and your message. Everything else the model needs, it fills in. That hatched block is guessing.

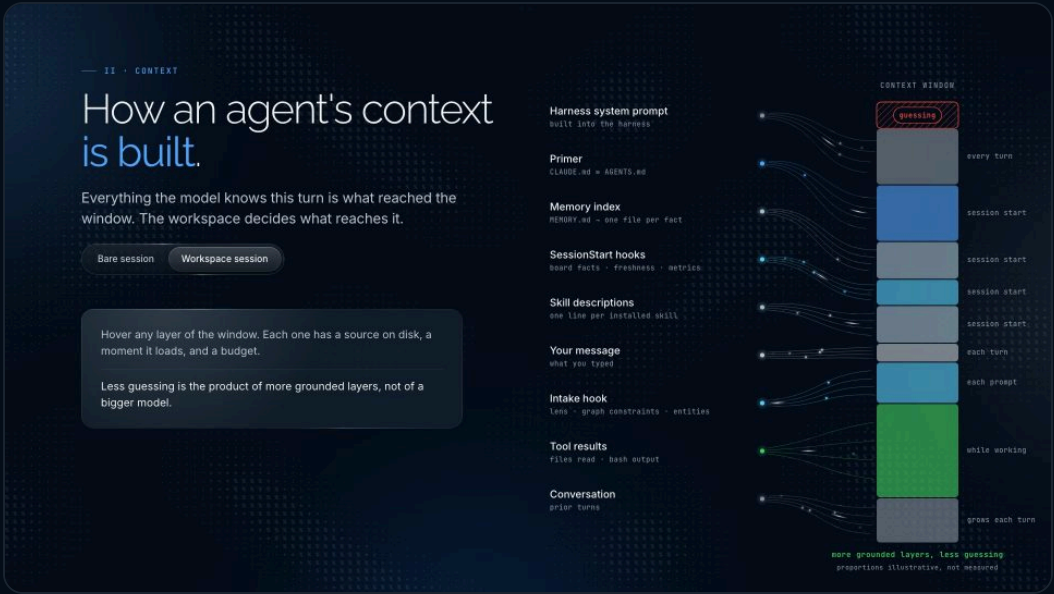
Workspace: the primer loads at boot, the memory index, hook injections at session start and on every prompt, skill descriptions, then whatever the agent reads and runs. The unknown shrinks. Hover the layers: each names where it comes from.

WHAT YOU'RE SEEING



BEAT 1 OF 2

Bare session. The harness system prompt and your message. Everything else is filled in by the model: the hatched block.



BEAT 2 OF 2

Workspace session. Layers arrive from disk: the primer at boot, the memory index, SessionStart hook output, one line per installed skill, the intake hook on every prompt, then tool results as the agent reads files and runs commands. The proportions are illustrative.

THE LAYERS AND THEIR SOURCES

- **Primer:** CLAUDE.md / AGENTS.md, loaded at session start.
- **Memory index:** one line per fact file (in Claude Code, ~/.claude/projects/<project>/memory/MEMORY.md), facts loaded on demand.
- **Hook output:** text a hook prints is added to context (SessionStart, UserPromptSubmit).

- **Skill descriptions:** each skill's front matter description is present up to a listing budget (past it, skills show as bare names); bodies load when triggered.
- **Tool results:** what the agent reads and runs. Usually the largest layer, and the one you shape with good files.

SET IT UP

1. Budget the always-on layers. Primer, memory index and skill descriptions are paid on every turn.
2. Move anything conditional into something that loads on demand: a routed doc, a skill body, a hook that

- only fires on a matching prompt.
3. Measure: count tokens in the always-on layers and keep a ceiling.

PROMPT

Inventory what loads into your context in this workspace before I type anything: the primer files, memory index, hook output, skill descriptions. For each, give the path, an approximate token count, and whether it must be always-on. Propose cuts that move conditional material into on-demand files. Change nothing yet.

SKILLS

- **role-x** skills/orchestration/role-x, load task-specific context on intake
- **kg** skills/knowledge/kg, load graph evidence on demand

REFERENCES

- [Effective context engineering for AI agents](#)
- [Claude Code memory](#)

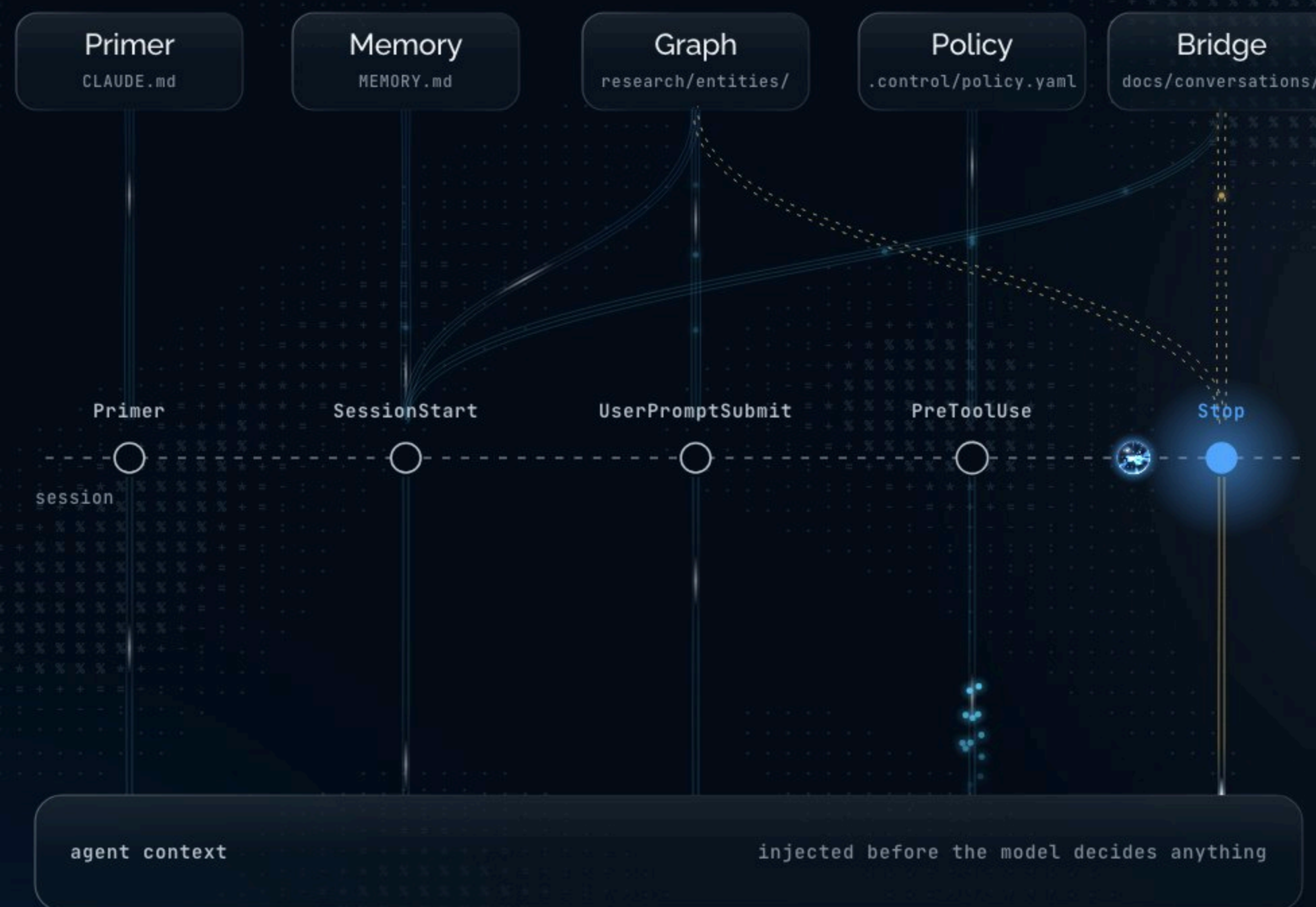
Hooks turn recall into a reflex.

• HOOK 05 OF 05 • BRIDGE • PROVENANCE

Stop

When the session stops, the bridge writes it to docs/conversations and provenance is stamped, so the next session can search what this one did.

`docs/conversations/session-2026-10-26-....md`



PRIMER

Asks for it

SKILL

Knows how

HOOK

Runs every time

Hooks turn recall into a reflex

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Three levers put knowledge in front of the agent, in increasing order of force. The primer asks. A skill knows how. A hook runs every time, without the model deciding.

WHAT TO SAY

Three levers, increasing in force. CLAUDE.md and AGENTS.md ask the agent to load the graph: that is reasoning-enforced, so it gets skipped. Skills package how. Hooks execute every time, without the model deciding.

Walk the session left to right. Real output from this workspace: the intake hook injects the matched lens, the knowledge-graph constraints and the auto-loaded entities into every prompt. The Stop hook writes the session to the bridge so the next one can find it.

THE STATIONS



BEAT 1 OF 5

Primer. CLAUDE.md routes: "What do we know? → kg skill → /kg load <topic> --terms <synonyms>". It is reasoning-enforced: the model may skip it.



BEAT 2 OF 5

SessionStart. Five workspace hooks run before the first prompt: skill freshness, git identity, auth preflight, lens coverage and skill sync. Plugin and user-level hooks add more; the bstack plugin's hook injects the board facts and the self-improvement metric.

BEAT 3 OF 5

UserPromptSubmit. The intake hook routes every prompt to a lens and injects its quality bar, the graph's constraints and the most relevant entities. This is recall without asking.

BEAT 4 OF 5

PreToolUse. Before Bash, Write and Edit, a control gate checks `.control/policy.yaml`, a regression gate and a file-write safety check run. In Claude Code, a PreToolUse hook that exits with code 2 blocks the call.

BEAT 5 OF 5

Stop. The bridge hook writes the session to docs/conversations/ and stamps provenance, so the next session can search it.

SET IT UP

1. Register hooks in `.claude/settings.json` under `hooks`:
event, optional tool matcher, command.
2. Keep hook output short and specific: a handful of entity claims with paths, not whole files.
3. Make a gate's failure message say what to do instead.

PROMPT

Add a `UserPromptSubmit` hook to `.claude/settings.json` that runs `scripts/kg-intake.sh`. The script reads the prompt from `stdin` JSON, extracts 3 to 6 key terms, scores `docs/knowledge-index.md` rows by matches in `slug`, `tags` and `core_claim`, and prints the top 5 as `"- claim · [path]"` under a heading `"Task-relevant knowledge"`. It must finish under 300 ms and print nothing when nothing matches. Add a test with two sample prompts.

SKILLS

- **role-x** skills/orchestration/role-x, the lens router behind the intake hook
- **kg** skills/knowledge/kg, the loader the hook and the primer point to
- **harness-engineering-playbook**
skills/governance/harness-engineering-playbook, design executable boundaries

REFERENCES

- [Claude Code hooks reference](#)
- Registered hooks: `.claude/settings.json`; gate: `scripts/control-gate-hook.sh`; bridge: `scripts/conversation-bridge-hook.sh`

Files first. Vectors when words diverge.

All Local Semantic **Live**

Exact names, paths, claims you can grep

RG + INDEX

What this relates to, one hop out

GRAPH LINKS

Same idea, different words; corpora past the window

EMBEDDINGS

Fresh or external state

MCP · APIS

At 1,430 entities the compiled catalog is about 260K tokens: it fits a 1M-token window, so the catalog routes and the model reads. Add vectors when vocabulary diverges or the catalog outgrows the window.



Files first, vectors when words diverge

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Retrieval is a toolbox the agent composes through bash and tools, not one database you bolt on.

WHAT TO SAY

A query fans out to every retriever the agent can call through bash or tools, and the answers come back cited. Local first: ripgrep and the compiled front-matter index are exact and free, and graph hops add structure flat search can't. Embeddings earn their place when words diverge or the catalog outgrows the window. Live systems come through MCP or APIs.

The point: retrieval is a toolbox the agent composes, not one database you bolt on.

WHAT YOU'RE SEEING



BEAT 1 OF 4

A query fans out to eight retrievers and the results come back merged into a cited answer.



BEAT 2 OF 4

Local. ripgrep for exact text, the compiled front-matter index for routing, graph hops along related: links, and body search when catalog hits are weak.



BEAT 3 OF 4

Semantic. Embeddings in a vector store find the same idea in different words, and reach corpora larger than the window. They return similar, not true: cite the source passage.



BEAT 4 OF 4

Live. MCP servers and APIs for state that changes: trackers, cloud resources, mail, the web.

WHEN TO ADD VECTORS

At our size (1,430 entities) the compiled catalog is about 260K tokens and fits a 1M-token window: the catalog routes, the model reads the entities it picked, and an embedding index would be one more thing to keep fresh. The entity bodies themselves (about 2.3M tokens) do not fit; that is what routing is for. Add vectors when

vocabulary diverges (users and documents use different words), when the catalog outgrows the window, or for external document sets you do not curate.

SET IT UP

1. Keep canonical material in files with stable paths.
2. Build a compact index for routing, then load source bodies only after a hit.
3. Add a semantic retriever only when exact terms and graph links miss known relevant material.

PROMPT

Answer my question using every retrieval path available here, in this order: rg over the repo, the knowledge index (docs/knowledge-index.md), one hop of related: links from the top hits, then any MCP or web source if the answer depends on current state. For each claim, cite the path or URL you actually opened, and list the paths you tried that returned nothing.

SKILLS

- **kg** skills/knowledge/kg, load graph context and related entities
- **checkit** skills/research/checkit, verify source material before it counts
- **bookkeeping** skills/knowledge/bookkeeping, capture supported findings durably

REFERENCES

- Entity: research/entities/pattern/llm-as-index-architecture.md
- [ripgrep](#)
- [Model Context Protocol](#)

ACT III

Second brain

How is knowledge read, and how does it grow?

@ the reader

Act III · Second brain

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Act III makes knowledge useful: measure whether it is read, keep claims and receipts in markdown, route with an index, and promote repetition carefully.

WHAT TO SAY

A wave front of dashes advances across a field: most of the graph sits in the dark until something reads it.

Act III covers recall rates, front matter, evidence-bearing entities, the knowledge loop and the promotion of repeated insight into mechanisms. Read next: Most second brains are write-only.

ACT SCENES

1. Most second brains are write-only
2. Front matter is the API
3. An entity is a claim with receipts
4. The knowledge loop
5. Insight hardens into mechanism

READ NEXT

Continue to **Most second brains are write-only** to start with the recall measure instead of the entry count.

Most second brains are write-only.

Today

Wired at intake

• THE FIX IS ROUTING, NOT VOLUME

Put recall on the startup path.

The primer routes to it. An intake hook surfaces matching entities.
`/kg load` runs before planning. And when reads lag, prune
production. Don't add more writers.



simulated · reads per write 8.4%

Most second brains are write-only

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

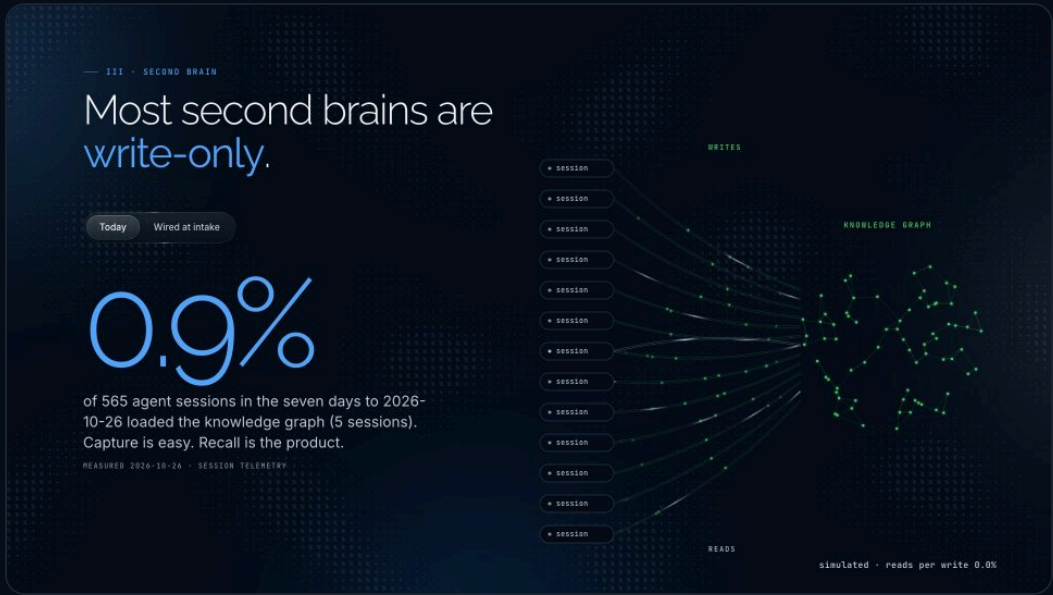
It is easy to build a second brain that only gets written. The measure that matters is how often agents read it at the moment they need it.

WHAT TO SAY

This number is from our own telemetry this week. Sessions write constantly: bookkeeping, ingestion, hooks. Almost none read. A graph nobody loads is a cost centre.

Toggle: the fix is putting recall where the agent already looks, at intake. The counter-intuitive part is the corrective action. When reads lag, you slow writes down.

WHAT YOU'RE SEEING



BEAT 1 OF 2

Today. Many write paths (bookkeeping, ingestion, hooks) and one thin read path. In the week measured (2026-10-26), 0.9% of 565 sessions loaded the graph.



BEAT 2 OF 2

Wired at intake. Every session gets a read path: the primer routes to it, the intake hook surfaces entities, and /kg load runs before planning.

SET IT UP

1. Instrument reads, not just writes: log each /kg load and each hook injection.
2. Put recall on the startup path (“Hooks turn recall into a reflex”).
3. When reads lag, slow writers down. More entries nobody reads add cost and noise.

PROMPT

Measure how this workspace's knowledge base is actually used. From the harness transcripts (for Claude Code, ~/.claude/projects/<project>/*.jsonl), count sessions in the last 7 days that loaded the graph (/kg load, reads under research/entities/) versus sessions that wrote to it. Report the ratio, the five most-read and five never-read entities, and one change that would raise reads.

SKILLS

- **kg** skills/knowledge/kg, load relevant graph context
- **bookkeeping** skills/knowledge/bookkeeping, score and maintain entities

REFERENCES

- docs/knowledge-index.md, compiled graph catalog
- docs/conversations/Conversations.md, session record index

Front matter is the API.

Files

Index

Edges

Synonyms

```
research/entities/pattern/  
llm-as-index-architecture.md
```

```
---
```

```
type: pattern    status: entity
```

```
core_claim: "At sub-10k-entity scale the  
substrate fits in a 1M-context  
model, so the LLM IS the query..."
```

```
tags: [architecture-decision, knowledge-graph, ...]
```

```
aliases: [kg routing, no embeddings retrieval]
```

```
related: [[knowledge-graph]], [[bstack-engine]] ...
```

```
sources: [2026-05-21-...-synthesis, BR0-1223]
```

```
---
```



docs/knowledge-index.md

13 entities · 23 edges

● grounded-vs-ungrounded-...

● multi-tier-dreaming

● recursive-controlled-sy...

● auto-compact-catalog

● bstack-engine

● entity-page

● haystack-benchmark-patt...

● knowledge-graph-enginee...

● llm-as-index-architectu...

● cognee

● kepano-obsidian-skills

● knowledge-graph

● open-knowledge-format

```
/kg load "knowledge graph" --terms "no embeddings retrieval" --expand 1
```

Front matter is the API

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

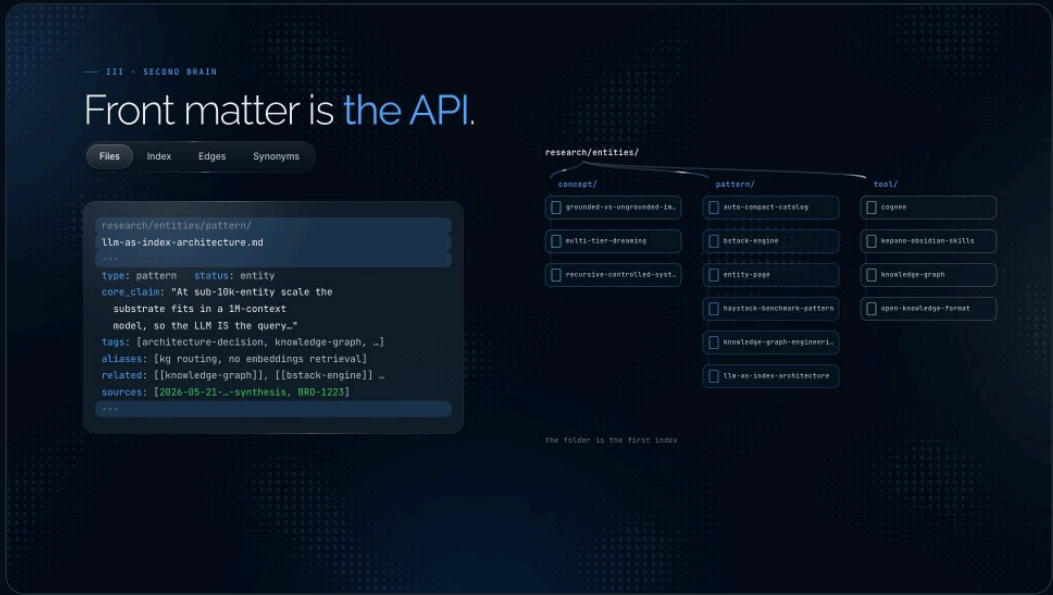
Plain markdown stays canonical. Front matter turns each file into a record that tools and agents can query without a database.

WHAT TO SAY

The folder is the first index: entities live under their type. Front matter turns each file into a record the tools can read without a database. A compiler builds one catalog an agent can load cheaply. related: links become graph edges, and aliases become synonyms that /kg load --terms matches.

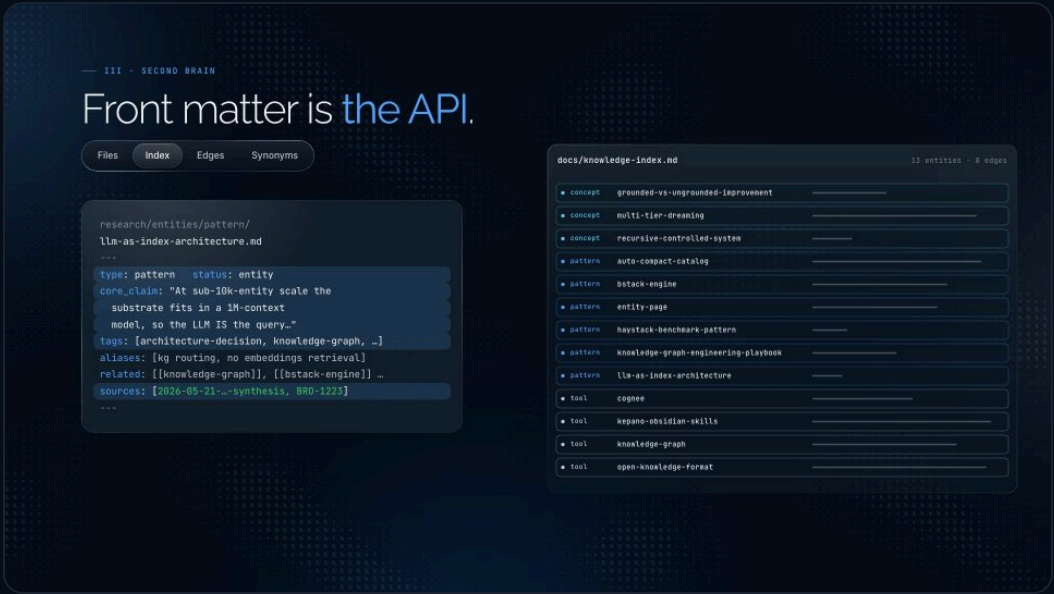
Plain markdown stays canonical; everything else is compiled from it and can be thrown away and rebuilt.

WHAT YOU'RE SEEING



BEAT 1 OF 4

Files. The folder is the first index: research/entities/<type>/<slug>.md. Each card is one entity, filed under its type.



BEAT 2 OF 4

Index. A compiler reads every front matter block, and each file becomes one row of docs/knowledge-index.md: type, slug, claim. The catalog is cheap enough to load at the start of a task.

Edges. Each row buds a node, and related: wikilinks grow into the edges between them. The catalog stays: the graph is compiled from the same files, not kept in a separate store. This is a real 13-entity slice of the graph around `llm-as-index-architecture`, with its 23 real links.

Synonyms. aliases: hang off their entity. /kg load
"knowledge graph" --terms "no embeddings retrieval"
--expand 1 scores its words against aliases as well as
slugs, ranks this entity first and, with --expand 1, pulls
one hop of neighbours with it.

A MINIMAL SCHEMA

```

---
id: pattern/llm-as-index-architecture
type: pattern                # concept | pattern |
tool | question | ...
status: entity               # candidate until
scored
core_claim: "One falsifiable sentence."
tags: [architecture-decision, knowledge-graph]
aliases: [kg routing, no embeddings retrieval]
related: ["[[knowledge-graph]]", "[[bstack-
engine]]"]
sources:
  - url: https://...
    note: "Read verbatim YYYY-MM-DD: the
passage that supports the claim"
created: YYYY-MM-DD
updated: YYYY-MM-DD
---

```

1. Make the markdown file and its front matter the canonical source.

2. Validate required fields and links in CI or before publishing.

3. Regenerate the catalog from files instead of hand-editing it.

Write `scripts/build_index.py`: walk `research/entities/**/*.md`, parse the YAML front matter, and write `docs/catalog.md` (in a repo that has no catalog yet) with a header (entity count, edge count, dangling links) and one row per entity: `type, slug, core_claim, tags, path`. Fail on missing `core_claim` or `type`. Add tests with a valid file, a missing field and a dangling `related: link`.

- **bookkeeping** skills/knowledge/bookkeeping, score, promote, index and lint

- **kg** skills/knowledge/kg, load by topic with synonyms

- docs/knowledge-index.md, generated catalog

- `research/entities/`, canonical entity markdown

An entity is a claim with receipts.

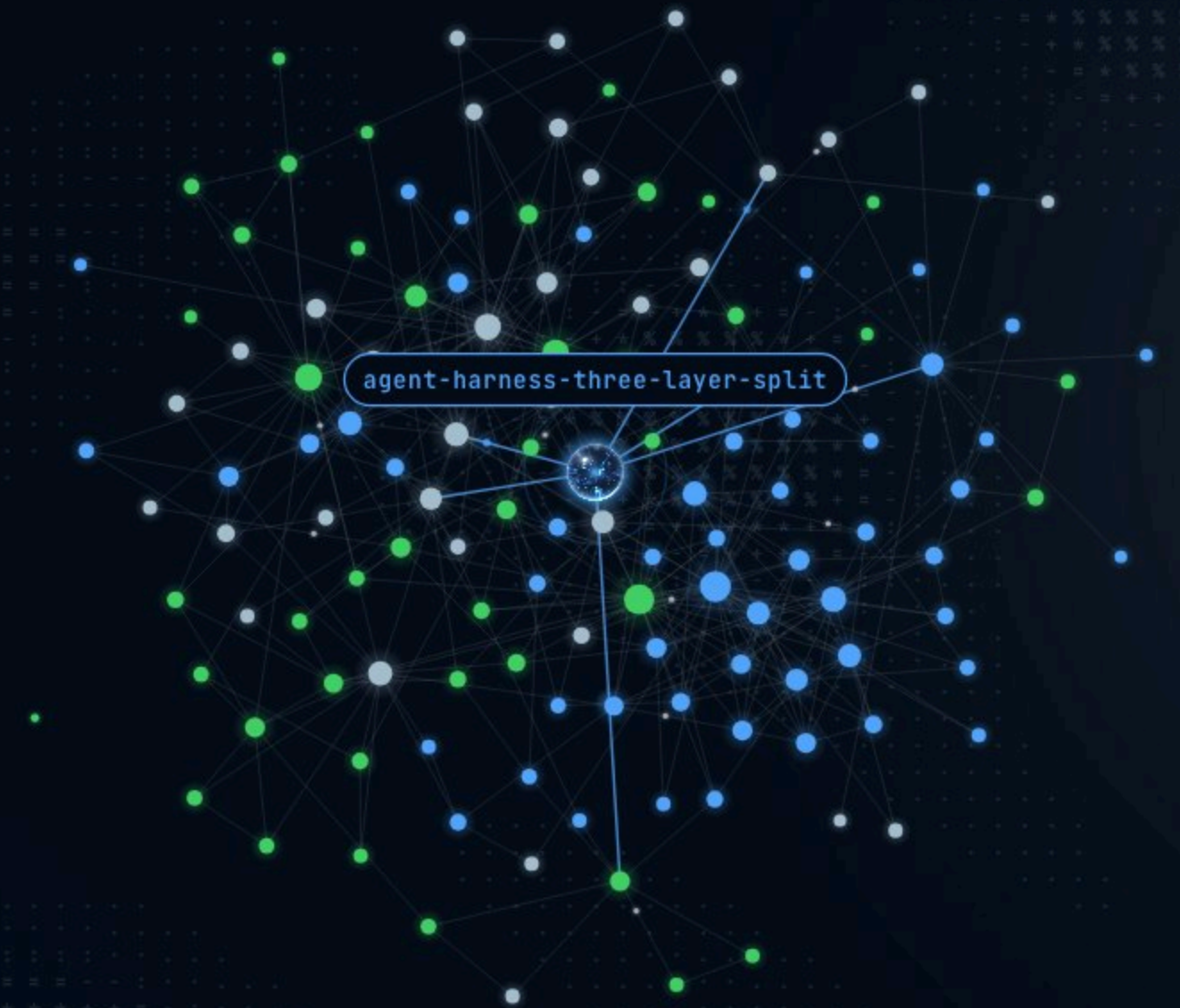
```
---
# research/entities/concept/agent-harness-three-layer-split.md
type: concept  status: entity
core_claim: "Agent harnesses split into sandbox,
orchestrator and UI wire: the sandbox
is buyable, the wire has no stable
remote transport."
sources:
- url: agentclientprotocol.com/.../transports
  note: "Read verbatim 2026-09-26 ..."
scoring: {novelty: 2, specificity: 3, relevance: 3}
related: [[buzz-acp-harness]], [[display-server-for-agents]]
---
```

— One falsifiable sentence

— Read verbatim, dated

— Scored before promotion

— Typed links become edges



1,430 entities 6,076 edges .md canonical · index of 2026-10-25

An entity is a claim with receipts

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

An entity is a single falsifiable claim with the evidence that supports it, scored before it is promoted and linked to what it depends on.

WHAT TO SAY

One entity, real. The core claim is a single falsifiable sentence, not a summary. Sources were read verbatim and dated, because a citation you didn't read is worth half. It was scored before promotion, and its links are typed edges.

The graph on the right is a real neighbourhood of this entity, found through related-neighbour traversal. Hover anything. Markdown is the source of truth; the index is compiled from it.

WHAT YOU'RE SEEING

III · SECOND BRAIN

An entity is a claim **with receipts**.

```
---
# research/entities/concept/agent-harness-three-layer-split.md
type: concept status: entity
core_claim: "Agent harnesses split into sandbox,
orchestrator and UI wire: the sandbox
is buyable, the wire has no stable
remote transport."
sources:
- url: agentclientprotocol.com/./transports
note: "Read verbatim 2026-09-26 ..."
scoring: {novelty: 2, specificity: 3, relevance: 3}
related: [[buzz-acp-harness]], [[display-server-for-agents]] ...
---
```

1430 entities 6,076 edges .md canonical · index of 2026-10-25

concept

pattern

tool

BEAT 1 OF 5

The entity file. The card is canonical markdown with a compact front matter record, not a row trapped in a database.

III · SECOND BRAIN

An entity is a claim **with receipts**.

```
---
# research/entities/concept/agent-harness-three-layer-split.md
type: concept status: entity
core_claim: "Agent harnesses split into sandbox,
orchestrator and UI wire: the sandbox
is buyable, the wire has no stable
remote transport."
sources:
- url: agentclientprotocol.com/./transports
note: "Read verbatim 2026-09-26 ..."
scoring: {novelty: 2, specificity: 3, relevance: 3}
related: [[buzz-acp-harness]], [[display-server-for-agents]] ...
---
```

1430 entities 6,076 edges .md canonical · index of 2026-10-25

concept

pattern

tool

BEAT 2 OF 5

One falsifiable sentence. `core_claim` is a claim, not a summary. It can be wrong, which is what makes it useful.

III · SECOND BRAIN

An entity is a claim **with receipts**.

```
---
# research/entities/concept/agent-harness-three-layer-split.md
type: concept status: entity
core_claim: "Agent harnesses split into sandbox,
orchestrator and UI wire: the sandbox
is buyable, the wire has no stable
remote transport."
sources:
- url: agentclientprotocol.com/./transports
note: "Read verbatim 2026-09-26 ..."
scoring: {novelty: 2, specificity: 3, relevance: 3}
related: [[buzz-acp-harness]], [[display-server-for-agents]] ...
---
```

1430 entities 6,076 edges .md canonical · index of 2026-10-25

concept

pattern

tool

BEAT 3 OF 5

Read verbatim, dated. Each source carries the passage actually read and the date. A citation you did not open is worth half.



BEAT 4 OF 5

Scored before promotion. Novelty, specificity and relevance. Below the bar, it stays a candidate.



BEAT 5 OF 5

Typed links become edges. The graph on the right is a real neighbourhood of this entity, found through related-neighbour traversal. Hover a node to read its slug.

SET IT UP

1. Write one claim that an informed reader could dispute.
2. Record the source passage actually read and its date.
3. Score it before promoting it, then add only existing related links.

PROMPT

Turn this finding into a knowledge entity:
write one falsifiable core_claim, list each source with the exact passage you read and today's date, score novelty, specificity and relevance from 0 to 3 with one line of reasoning each, and link 2 to 4 related entities that already exist (check with rg). If the total is under 5, keep status: candidate.

SKILLS

- **bookkeeping** skills/knowledge/bookkeeping, score, link and lint the entity
- **kg** skills/knowledge/kg, discover existing related entities

REFERENCES

- The entity shown: research/entities/concept/agent-harness-three-layer-split.md
- Catalog: docs/knowledge-index.md (1,430 entities, 6,076 edges)

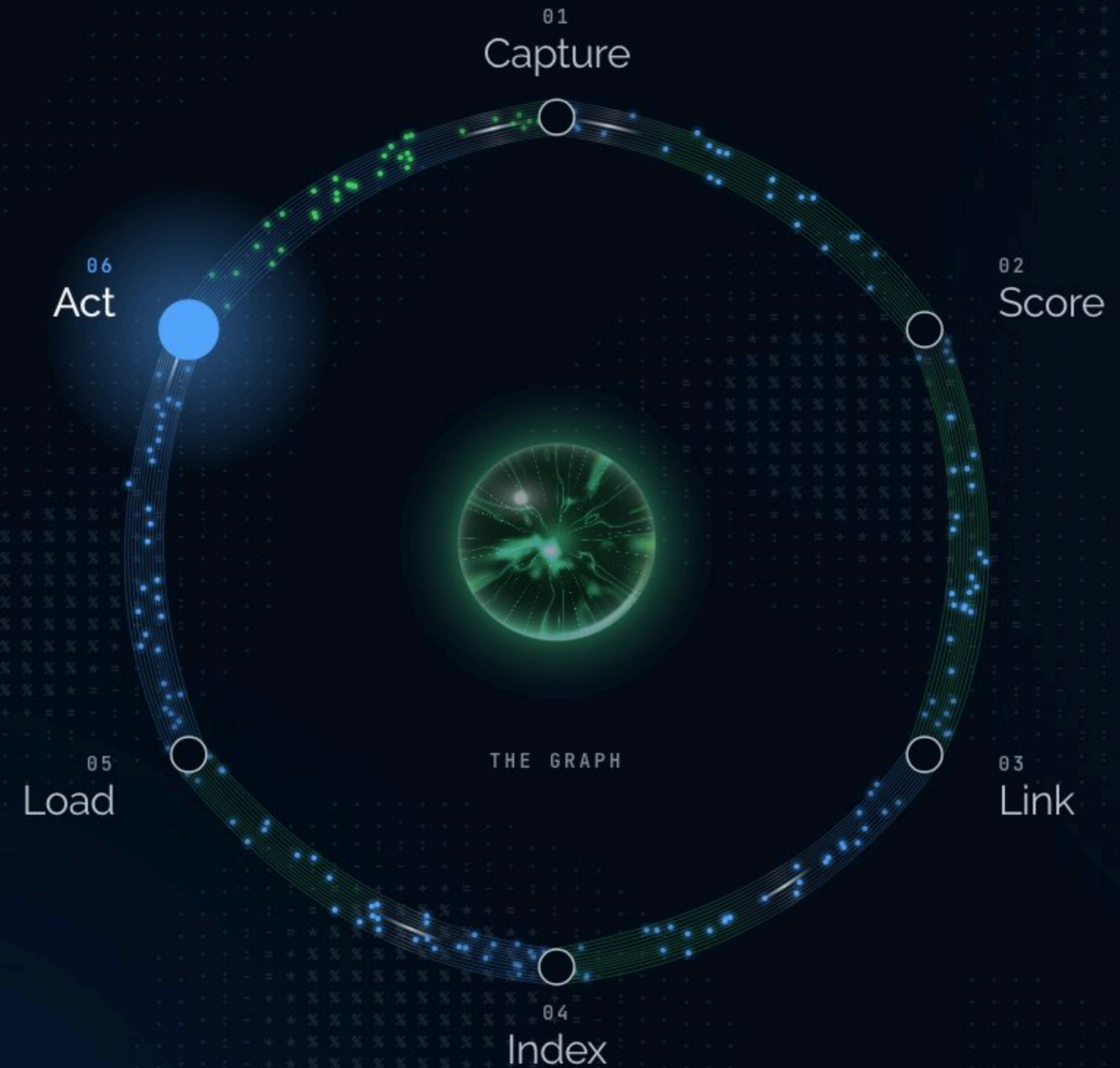
The loop that makes knowledge **readable**.

● STATION 06 OF 06

Act

cite → decide → capture

Outputs cite entities. New findings re-enter the loop.



Memory holds facts about the owner. The graph holds claims about the world. The ledger holds what is in flight.

The knowledge loop

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

The knowledge loop has six stations. Skipping any one of them is how a graph goes stale or goes unread.

WHAT TO SAY

Walk the loop with →. Capture verifies sources at the door. Scoring keeps low-signal material as candidates. Links are explicit and linted. The index is one compiled catalog an agent can load cheaply. Load happens at intake with synonyms, and it reports what it missed. Then outputs cite entities, and new findings come back in.

The footnote matters: three stores, three jobs. Don't put owner preferences in the graph or world claims in memory.

THE STATIONS



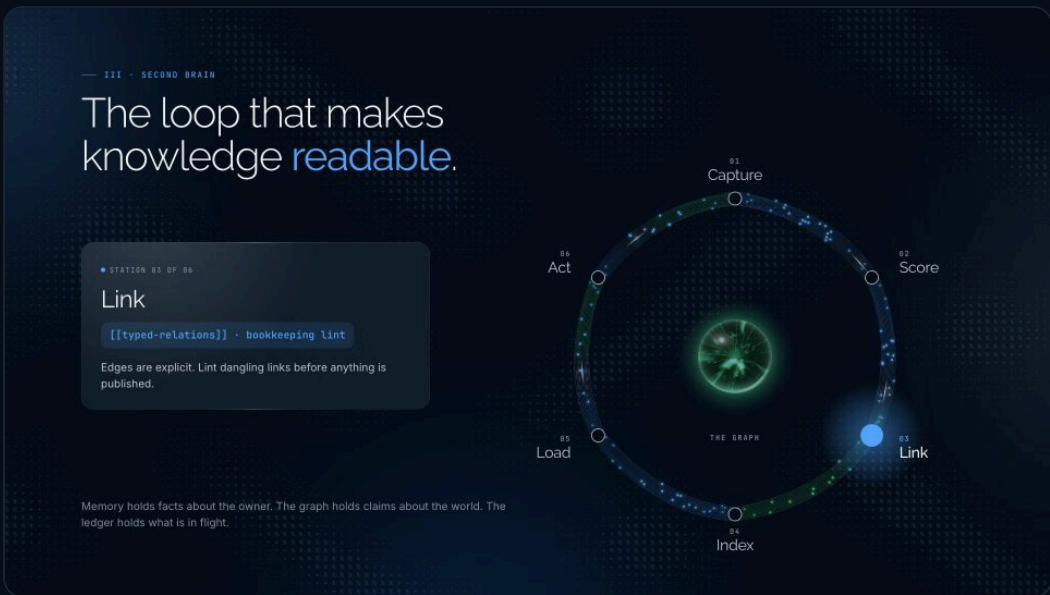
BEAT 1 OF 6

Capture. `/checkit <artifact>` ingests a link, paper or repo and verifies every source before it counts.



BEAT 2 OF 6

Score. bookkeeping run scores and promotes. Below the bar, a finding stays a candidate.



BEAT 3 OF 6

Link. Edges are explicit `[[relations]]`; bookkeeping `lint` catches dangling links before publishing.

III · SECOND BRAIN

The loop that makes knowledge **readable**.

STATION 04 OF 04

Index

knowledge-index.md

One compiled, LLM-loadable catalog. Entities stay canonical markdown.

01 Capture

02 Score

03 Link

04 Index

05 Load

06 Act

THE GRAPH

Memory holds facts about the owner. The graph holds claims about the world. The ledger holds what is in flight.

BEAT 4 OF 6

Index. One compiled catalog, docs/knowledge-index.md. Entities stay canonical markdown.

III · SECOND BRAIN

The loop that makes knowledge **readable**.

STATION 05 OF 04

Load

/kg load <topic> --terms <synonyms>

Recall at intake, with synonyms, and report what was missed.

01 Capture

02 Score

03 Link

04 Index

05 Load

06 Act

THE GRAPH

Memory holds facts about the owner. The graph holds claims about the world. The ledger holds what is in flight.

BEAT 5 OF 6

Load. /kg load <topic> --terms <synonyms> at intake, reporting what it missed.

III · SECOND BRAIN

The loop that makes knowledge **readable**.

STATION 06 OF 04

Act

cite - decide - capture

Outputs cite entities. New findings re-enter the loop.

01 Capture

02 Score

03 Link

04 Index

05 Load

06 Act

THE GRAPH

Memory holds facts about the owner. The graph holds claims about the world. The ledger holds what is in flight.

BEAT 6 OF 6

Act. Outputs cite entities; new findings re-enter at capture.

THREE STORES, THREE JOBS

Memory holds facts about the owner. The graph holds claims about the world. The ledger holds what is in flight. Mixing them is how private preferences end up in shared knowledge, or stale status ends up in memory.

SET IT UP

- Capture only source-backed material.

- Score and lint before an entry becomes durable graph knowledge.
- Load the compact catalog at intake and cite the full entity when using it.

PROMPT

Trace one finding through this repository's knowledge loop: identify the source capture, score or candidate state, related links, catalog row, and one session where it was loaded. Report missing stations and propose the smallest repair. Do not edit files.

SKILLS

- checkit** skills/research/checkit
- bookkeeping** skills/knowledge/bookkeeping
- kg** skills/knowledge/kg

REFERENCES

- docs/knowledge-index.md, compiled catalog
- docs/references/workspace-context.md, knowledge-layer boundaries

Insight hardens into mechanism.

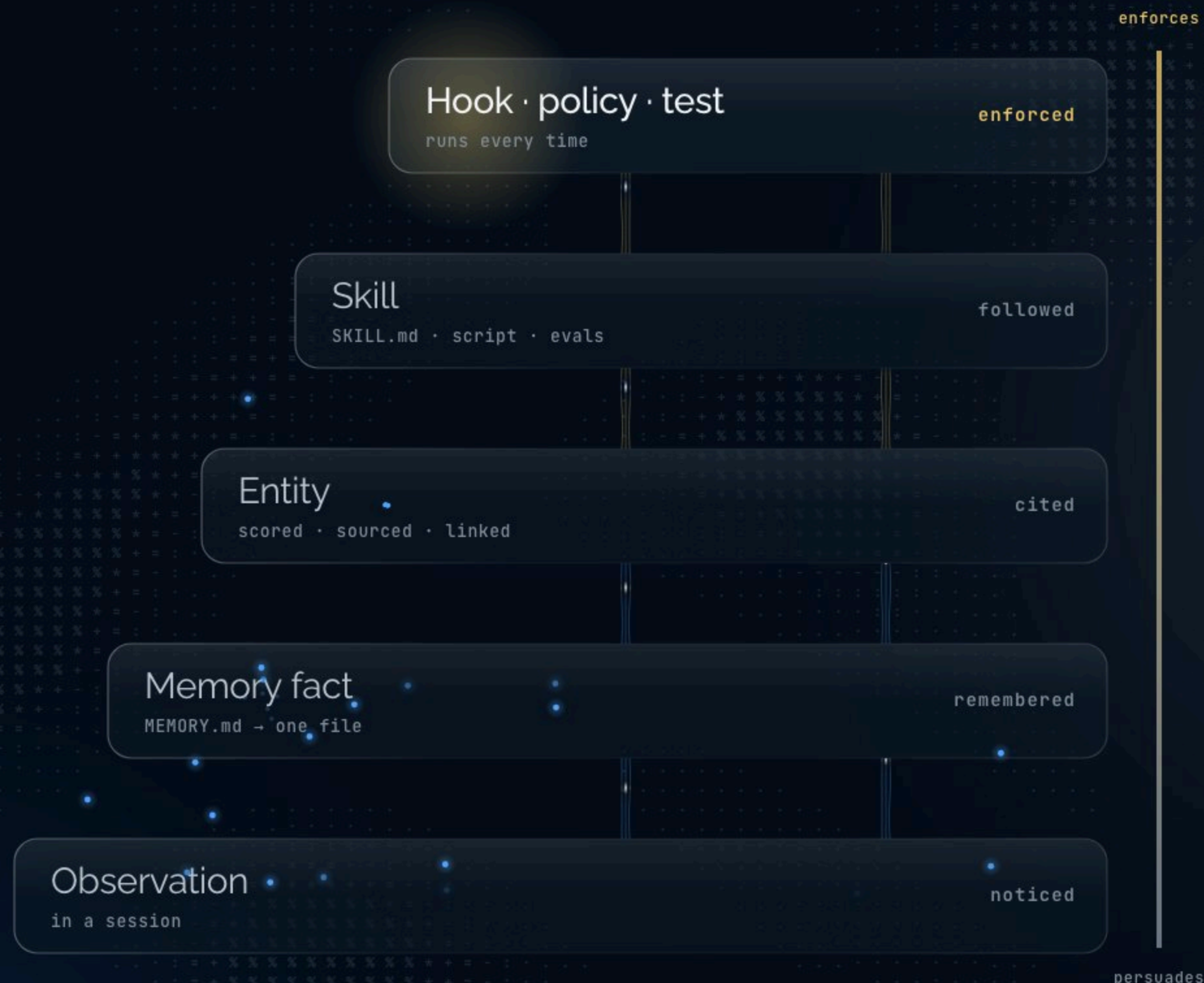
• LEVEL 05 OF 05 • ENFORCED

Hook · policy · test

A rule that must never be broken stops depending on anyone remembering it. It executes.

Prose persuades. Executables enforce.

Promote on the third instance, with a concrete mechanism, an invariant and a failure mode. Name what the new rule retires.



Crystallize: insight hardens into mechanism

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

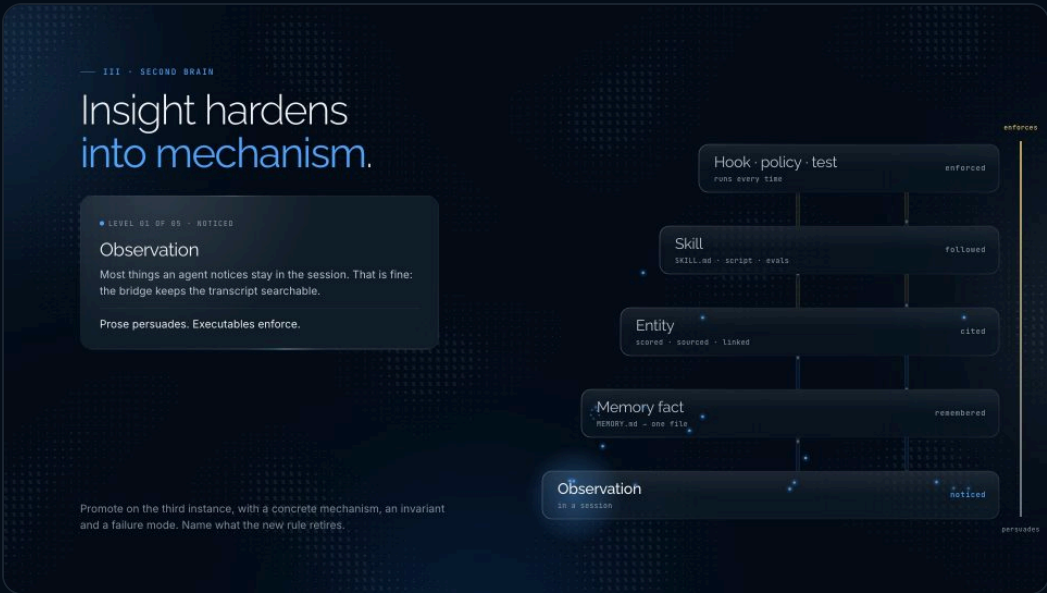
This is how the agent gives back. Insights climb a ladder from observation to mechanism, and each step up trades flexibility for force.

WHAT TO SAY

This is how the agent gives back to the graph. Most observations stay observations. When something recurs it becomes a memory fact or a scored entity. When a procedure recurs it becomes a skill with evals. When a rule must never be broken it becomes a hook, a policy gate or a test.

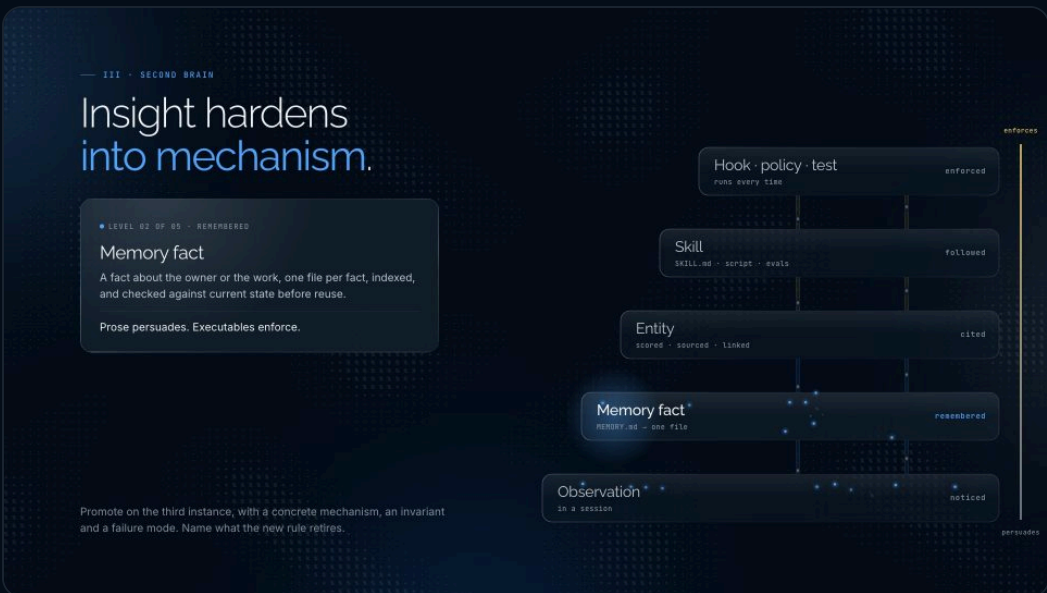
Each step up trades flexibility for force: prose persuades, executables enforce. The ratchet only turns on evidence, and every promotion names what it retires so the system doesn't just grow.

THE LEVELS



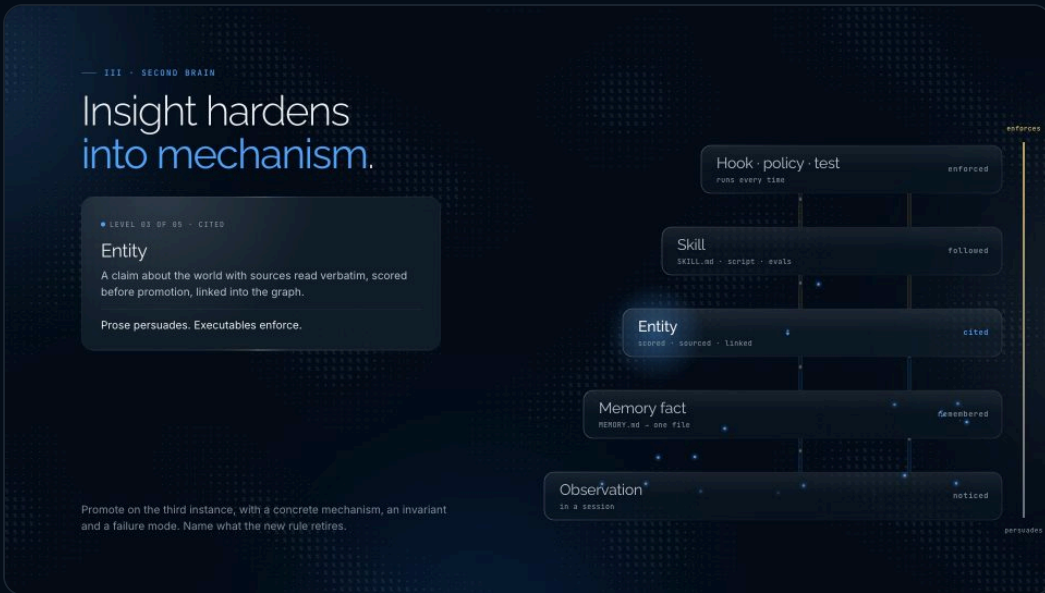
BEAT 1 OF 5

Observation. Most things noticed in a session stay there. The session record keeps them searchable.



BEAT 2 OF 5

Memory fact. One file per fact about the owner or the work, indexed, checked against current state before reuse.



BEAT 3 OF 5

Entity. A claim about the world with sources, scored and linked.

III · SECOND BRAIN

Insight hardens into mechanism.

LEVEL 04 OF 05 · FOLLOWED

Skill

A procedure that recurred: a description that routes, a deterministic core and evals that prove it triggers.

Prose persuades. Executables enforce.

Promote on the third instance, with a concrete mechanism, an invariant and a failure mode. Name what the new rule retires.

Hook · policy · test

runs every time

enforced

Skill

skill.md · script · evals

followed

Entity

entity.md · sourced · linked

cited

Memory fact

memory.md · only file

remembered

Observation

in a session

noticed

enforces

persuades

BEAT 4 OF 5

Skill. A procedure that recurred: a description that routes, a deterministic core, evals that prove it triggers.

III · SECOND BRAIN

Insight hardens into mechanism.

LEVEL 05 OF 05 · ENFORCED

Hook · policy · test

A rule that must never be broken stops depending on anyone remembering it. It executes.

Prose persuades. Executables enforce.

Promote on the third instance, with a concrete mechanism, an invariant and a failure mode. Name what the new rule retires.

Hook · policy · test

runs every time

enforced

Skill

skill.md · script · evals

followed

Entity

entity.md · sourced · linked

cited

Memory fact

memory.md · only file

remembered

Observation

in a session

noticed

enforces

persuades

BEAT 5 OF 5

Hook, policy or test. A rule that must never be broken stops depending on memory. It executes.

THE PROMOTION RULE

Promote on the third instance, with a concrete mechanism, an invariant and a named failure mode. Every promotion names what it retires, so the system does not only grow.

SET IT UP

1. Keep a dated record of the separate instances.
2. State the mechanism, invariant and failure mode before creating a permanent rule.

3. Give the new mechanism an executable check where possible.

PROMPT

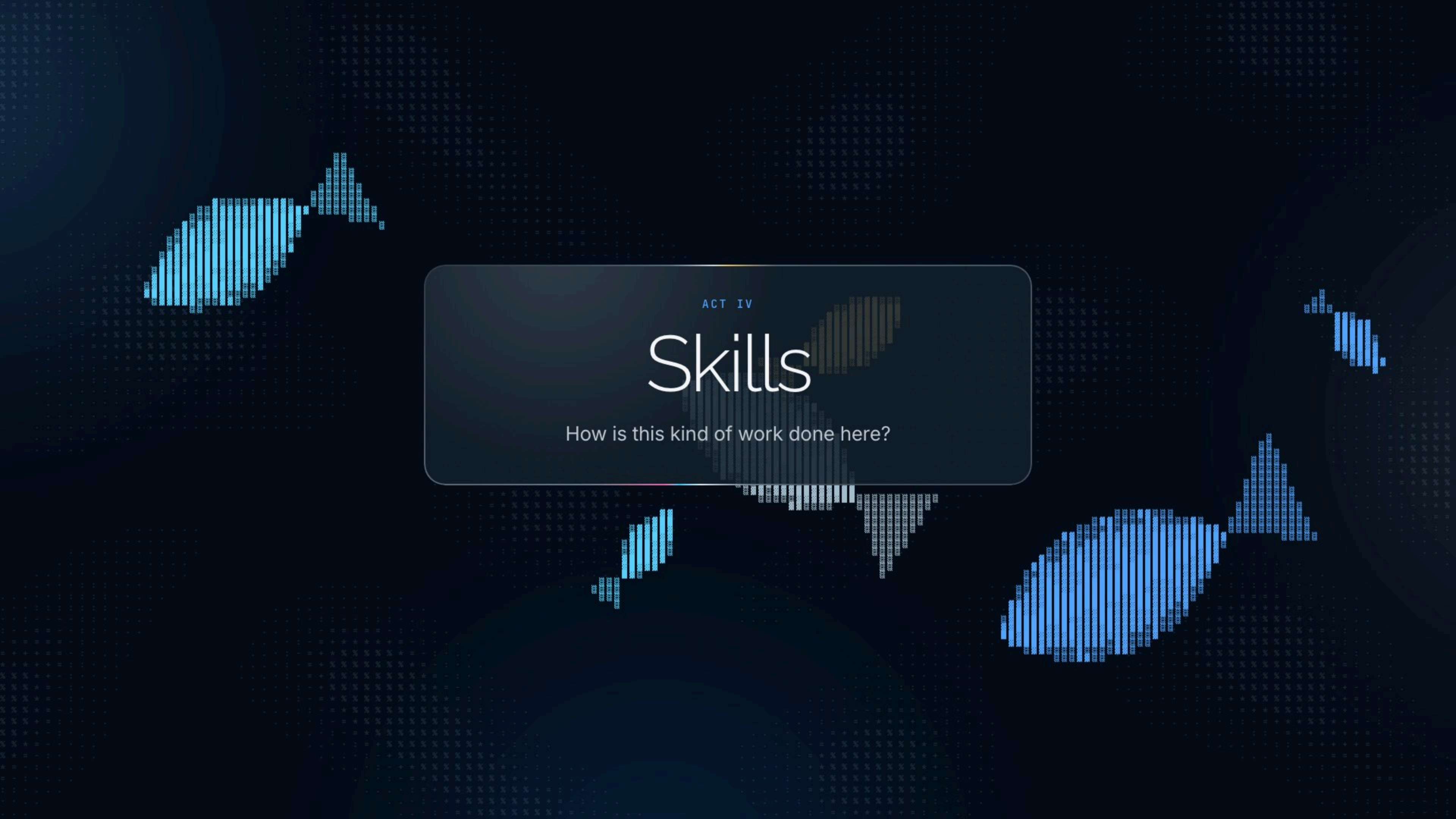
Look back over the last two weeks of session records in docs/conversations/. Find corrections or rules that came up three or more times. For each, propose the lowest level that would make it stick (memory fact, entity, skill, hook, policy or test), the mechanism, the invariant it enforces, how it could fail, and what existing rule or note it replaces.

SKILLS

- **skillify** skills/tooling/skillify, turn a working session into a tested skill
- **bookkeeping** skills/knowledge/bookkeeping, promote supported findings into entities

REFERENCES

- docs/references/agent-operating-protocols.md, Crystallize (P16)
- research/entities/, durable, cited findings



ACT IV

Skills

How is this kind of work done here?

Act IV · Skills

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Act IV packages repeatable work as procedures, keeps deterministic work testable, and keeps the skill roster small enough to route.

WHAT TO SAY

Glyph-mosaic fish swim behind the glass: procedures you can see moving, loaded only when they are called.

Act IV covers skill anatomy, the boundary between skills, CLIs and MCP, and why the skill roster is a budget. Read next: A skill is a procedure.

ACT SCENES

- 1. A skill is a procedure
- 2. Skill, CLI, MCP
- 3. Author once, fan out, ration

READ NEXT

Continue to **A skill is a procedure** to see how progressive disclosure keeps procedures available without loading them all.

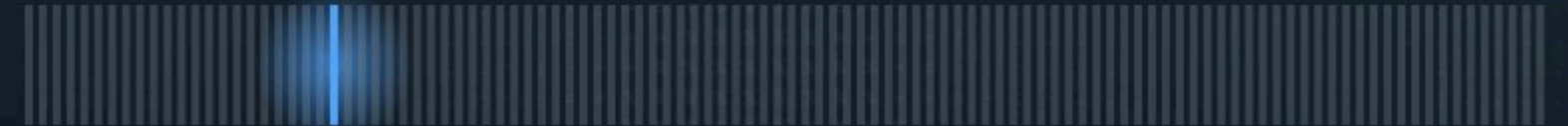
A skill is a procedure, not a prompt.

```
skills/knowledge/kg/  
├ SKILL.md      description is the router  
├ scripts/      deterministic core  
└ evals/        routing + behavior tests
```

```
---  
name: kg  
description: Load graph context for a topic...  
  USE WHEN ... NOT FOR ...  
---
```

CONTEXT WINDOW

ALWAYS LOADED · every skill's description



"what do we know about sandboxes?"

→ kg

ON TRIGGER · SKILL.md body

SKILL.md · kg

ON DEMAND · scripts/ · references/

a body loads only when its description matches

A skill is a procedure

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

A skill is a folder: a SKILL.md whose front matter description routes, a deterministic script core, references loaded on demand, and evals.

WHAT TO SAY

A skill is a folder: a SKILL.md, a deterministic script core, references, and evals. The description in the frontmatter is the only part that is always in context, so it is the router. Write it like a trigger spec: use when, not for.

The animation is progressive disclosure: a request scans the descriptions, one matches, its body loads, and references load only if needed.

WHAT YOU'RE SEEING

Installed skills' descriptions sit in context, up to a listing budget (the thin bars). A request is matched against them; the winner's body loads, and its scripts

and references load only if needed.

WRITE THE DESCRIPTION LIKE A TRIGGER SPEC

```
---
name: kg
description: >—
  Load knowledge-graph entities relevant to a
  topic from research/entities.
  USE WHEN: "what do we know about X", planning
  that needs prior findings,
  before /checkit. NOT FOR: owner preferences
  (memory), live project status
  (initiative ledger).
---
```

SET IT UP

- 1. Put deterministic work in scripts/ with tests; keep judgment in the body.
- 2. Add evals in both polarities: prompts that must trigger it and prompts that must not.
- 3. Keep the body under a few hundred lines; move detail to references/.

PROMPT

Create a skill at skills/<category>/<name>/ for the procedure we just did. SKILL.md with name and a description that says USE WHEN and NOT FOR; the deterministic steps as scripts/<name>.py with tests; an evals file with 5 prompts that should trigger it and 5 that should not. Run the tests and show the results.

SKILLS

- **skillify** skills/tooling/skillify, distill a session into a tested skill
- **role-x** skills/orchestration/role-x, test whether the description routes correctly

REFERENCES

- [Claude Code skills](#)
- [Agent Skills overview](#)
- [anthropics/skills examples](#)

Skill, CLI, MCP.

One knows. One does. One reaches.

All

Skill

CLI

MCP

Repeatable judgment and steps

SKILL

Deterministic, local, testable work

CLI

Remote system, auth, shared across harnesses

MCP SERVER

A one-off

JUST ASK

A skill that drives a small CLI beats a wide MCP surface: fewer tools in context, and a core you can test.

Skill
knows when and how

SKILL.md



HARNESS

\$ kg load ...

CLI

does: local, deterministic, testable

MCP

reaches: remote, authed, shared

MCP

tracker

cloud

mail

Skill, CLI, MCP

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Three ways to give an agent a capability. They compose; they do not compete.

WHAT TO SAY

The most common design mistake is reaching for MCP by default. MCP earns its place for remote systems that need auth and should be shared across harnesses. Local deterministic work belongs in a CLI you can test. The skill is the knowledge of when and how to call either.



BEAT 1 OF 4

All. The diagram shows the decision surface: use no extra mechanism for a one-off, then add only the smallest durable capability the work requires.



BEAT 2 OF 4

Skill: knows. Repeatable judgment and steps, loaded when the description matches.



BEAT 3 OF 4

CLI: does. Deterministic, local, testable work. Pipes into other tools. Lives in the skill's scripts/ or on PATH.



BEAT 4 OF 4

MCP: reaches. Remote systems with authentication, shared across every harness. Each tool definition costs context, so keep the surface small.

RULE OF THUMB

A skill that drives a small CLI beats a wide MCP surface for local work: fewer tools in context and a core you can unit-test. Reach for MCP when the system is remote, needs auth, or should be shared by several harnesses. A one-off needs neither: just ask.

SET IT UP

1. Write the deterministic local operation as a command with tests.
2. Wrap judgment about when and how to use it in a skill.
3. Expose remote, authenticated state through the narrowest MCP tool surface.

PROMPT

I want the agent to be able to <capability>. Recommend skill, CLI, MCP server or a combination, using these criteria: is it local and deterministic (CLI), does it need judgment about when and how (skill), does it touch a remote authenticated system or need sharing across harnesses (MCP). Give the smallest design and what each piece contains.

SKILLS

- **skillify** skills/tooling/skillify, package a repeated procedure
- **harness-engineering-playbook** skills/governance/harness-engineering-playbook, assess the integration boundary

REFERENCES

- [Model Context Protocol](#)
- [Claude Code skills](#)

Author once. Fan out. Ration hard.

Fan out

Ration

115

AUTHORED

23

CATEGORIES

31

HARNESS FOLDERS

- ◆ One canonical source, in its own repo.
- ◆ Install once, symlink into every harness. Never copy.
- ◆ New skills ship evals: it fires when it should and stays quiet when it shouldn't.
- ◆ Retire before you add. The roster is a budget.

457 INSTALLED

26 KEPT A DESCRIPTION

the rest were bare names: they could not trigger

Author once, fan out, ration

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Author skills once, install them once, and symlink them into every harness. Then ration: more skills is not more capability.

WHAT TO SAY

One source of truth, symlinked into every harness's folder, so Claude, Codex and the rest read the same file.

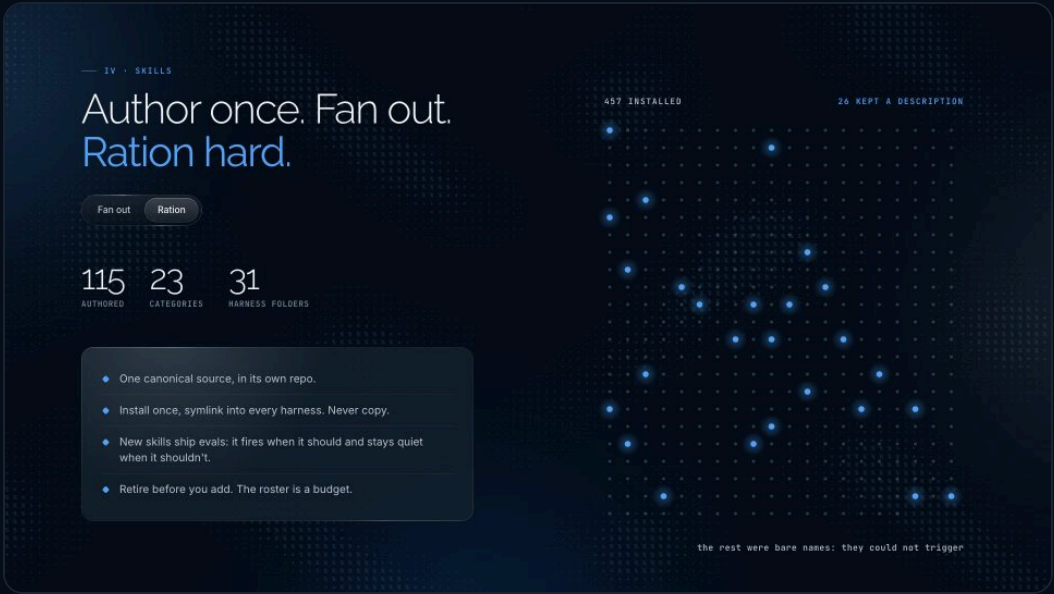
Then the measured surprise (→). With 457 skills installed, the harness kept full descriptions for 26. The rest were bare names, and a bare name cannot trigger itself. More skills made each one less likely to fire. That's why the roster is rationed.

WHAT YOU'RE SEEING



BEAT 1 OF 2

Fan out. One canonical repository is the source. In this workspace 31 harness folders (.claude, .windsurf, .kiro and others) point at one .agents/skills install. Keep that install a link to the canonical repo, not a copy: copies drift.



BEAT 2 OF 2

Ration. Measured on 2026-08-23: with 457 skills installed, the harness kept full descriptions for 26. The rest were bare names, and a bare name cannot trigger itself. More skills made each one less likely to fire.

SET IT UP

1. One source repository for skills, versioned and reviewed like code.
2. Install to one folder and symlink; never copy, or versions drift.
3. Keep a roster file of required skills and retire before you add.

PROMPT

Audit the skills installed for this workspace: count them per harness folder, find copies that are not symlinks to the canonical install, find duplicates with overlapping descriptions, and list skills with no eval file. Propose a roster of at most 30 that covers the work in the last month of session records, and what to retire.

SKILLS

- **skillify** skills/tooling/skillify, maintain tested procedures
- **role-x** skills/orchestration/role-x, evaluate whether descriptions route to the right procedure

REFERENCES

- Required roster: bstack/references/companion-skills.yaml

```

| *      ab9ed2  fix(bridge): capture session on stop
|\      385e34  docs(handoff): pickup for tomorrow
*      779a59  fix(bridge): capture session on stop
|/      920e00  feat(policy): gate destructive bash
| *      3060dc  merge: review strata passed
*      64a169  feat(skills): add routing evals
*      da5c51  fix(hooks): intake hook stays under 300 ms
|/      c342b9  merge: review strata passed
| *      49afd6  fix(hooks): intake hook stays under 300 ms
| *      4b0a25  docs(kg): score and link three entities
|\      3d705e  fix(hooks): intake hook stays under 300 ms
*      73c479  chore(janitor): sweep merged worktrees
|/      e2128a  feat(policy): gate destructive bash
| *      df1de8  refactor(primer): route, don't dump
*      b9b666  docs(kg): score and link three entities
*      52ada8  feat(index): compile front matter
|/      fdb864  merge: review strata passed
| *      487e4a  feat(policy): gate destructive bash
| *      feddfe  feat(index): compile front matter
|\      fdce48  docs(handoff): pickup for tomorrow
*      c240e3  feat(skills): add routing evals
|/      b5aff1  docs(handoff): pickup for tomorrow
| *      94e055  refactor(primer): route, don't dump
*      545467  chore(janitor): sweep merged worktrees
*      fd7947  feat(index): compile front matter
|/      e619de  fix(bridge): capture session on stop
| *      904005  refactor(primer): route, don't dump
| *      334d21  refactor(primer): route, don't dump
|\      7e32c4  docs(handoff): pickup for tomorrow
*      90db0e  merge: review strata passed
|/      a35eb1  feat(policy): gate destructive bash

```

ACT V

Structure

Which bytes may the agent touch?

Act V · Structure

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Act V separates shared governance from sovereign product repositories and gives every concurrent writer an isolated tree.

WHAT TO SAY

A git graph scrolls past: every change on its own branch, merges marked in gold.

Act V covers the governance workspace and the one-writer-per-worktree discipline. Read next: One place, many repos.

ACT SCENES

- 1. One place, many repos
- 2. One writer per tree

READ NEXT

Continue to **One place, many repos** to map what belongs in shared governance and what remains with a product.

- **SHARED** one copy of the operating context, reviewed like code
- **SOVEREIGN** 66 product repos under apps/, core/ and work/, each with its own history
- **DISPOSABLE** worktrees created 7 · merged 0 · swept 0





BEAT 3 OF 3

Disposable. A worktree buds off a repo for one task, works, comes back through review, merges (gold) and is swept. The counter keeps running: created, merged, swept.

THE LAYOUT

workspace/	governance repo
├ AGENTS.md ≡ CLAUDE.md	one primer, every harness
harness	policy · initiatives
├ .control/	
· asks	
├ roles/	lenses: context per
kind of work	
├ skills/	canonical skills
(own repo)	
├ research/entities/	the graph, as
markdown	
├ docs/	sessions · plans ·
handoffs · ADRs	
├ apps/ core/ work/ ...	product repos, own
CI	
└ .worktrees/	one per task,
disposable	

RULES

- **Shared:** the primer, policy, lenses, skills, graph and record. One copy, reviewed like code.
- **Sovereign:** each product repo has its own CI, release cadence and local AGENTS.md. Agents read that file first when working there.
- **Disposable:** worktrees branch from main, merge back through review, and are swept after merge.

SET IT UP

1. Keep shared instructions and records in the governance repository.
2. Give every product repository its own local instructions and delivery checks.
3. Create worktrees from the correct product base, not from an assumed workspace root.

PROMPT

Propose the folder layout for a governance workspace that holds these product repos: <list>. Show the tree, which directories are their own git repositories, which files every agent reads first, and where session records, plans, handoffs and ADRs live. Flag anything in the current tree that would conflict.

SKILLS

- **role-x** skills/orchestration/role-x, choose the local context for the task
- **handoff** skills/orchestration/handoff, preserve the cross-context work record

REFERENCES

- docs/references/workspace-context.md, conventions and boundaries
- roles/_index.md, lens routing

One writer per tree.

Parallel agents never share a checkout. They share main, through review.



01

Branch is the first action

02

Reviewers read. They never write

03

A live thread's worktree is the thread

04

Unfinished work becomes a draft PR

05

Janitor after merge, dry-run first

One writer per tree

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Parallel agents never share a checkout. Each works in its own worktree and meets the others only on main, through review.

WHAT TO SAY

Each lane is an agent in its own worktree. Commits flow along the lane, hit review, and either merge with a flash or bounce back. Finished lanes fade: that's the janitor.

Every rule on the bottom row exists because we broke it once. Removing a live thread's worktree bricked the thread. Salvaging someone's uncommitted work silently lost provenance, which is why it now becomes a draft PR.

One writer per tree. The lanes represent separate checkouts, not tabs in one mutable directory. Their common meeting point is a reviewed change, not a

shared working tree.

RULES, EACH LEARNED THE HARD WAY

1. **Branch is the first action.** After checking out main, create the branch before touching a file.
2. **Reviewers read; they never write.** A reviewer that "reproduces" by editing the live tree corrupts the writer's work.
3. **A live thread's worktree is the thread.** Removing it breaks the session bound to it.
4. **Unfinished work becomes a draft PR,** never a silent salvage into another branch.
5. **Janitor after merge, dry run first.** Sweep merged branches and worktrees; never touch protected or unrelated work.

SET IT UP

1. Create the branch and worktree before the first edit.
2. Assign one writer to each tree and give reviewers read-only scope.
3. Check status before every commit and use a dry run before cleanup.

PROMPT

Create a worktree for this task: `git worktree add .worktrees/<slug> -b <type>/<slug> origin/main`. Work only inside it. Before committing, show git status and confirm every changed file belongs to this task. When the PR merges, list what the janitor would remove (dry run) and wait.

SKILLS

- **p9** skills/orchestration/p9, watch CI instead of polling
- **cross-review** skills/governance/cross-review, keep writing separate from review

REFERENCES

- docs/references/agent-operating-protocols.md, Hygiene (P10) and Janitor (P8)
- .control/policy.yaml, delivery gates

ACT VI

Fleets

Who else is here, and who checks whom?



Act VI · Fleets

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Act VI assigns durable responsibilities, preserves continuity across contexts, and puts review and capacity limits between parallel work and delivery.

WHAT TO SAY

Glass orbs ride two liquid-metal orbits: many agents, one governed system.

Act VI covers fleet roles, continuation choices and governance that responds to capacity. Read next: Roles, not agents.

ACT SCENES

- 1. Roles, not agents
- 2. Spawn, reuse, or hand off
- 3. Governance that scales with the fleet

READ NEXT

Continue to **Roles, not agents** to give each participant a responsibility and an explicit non-responsibility.

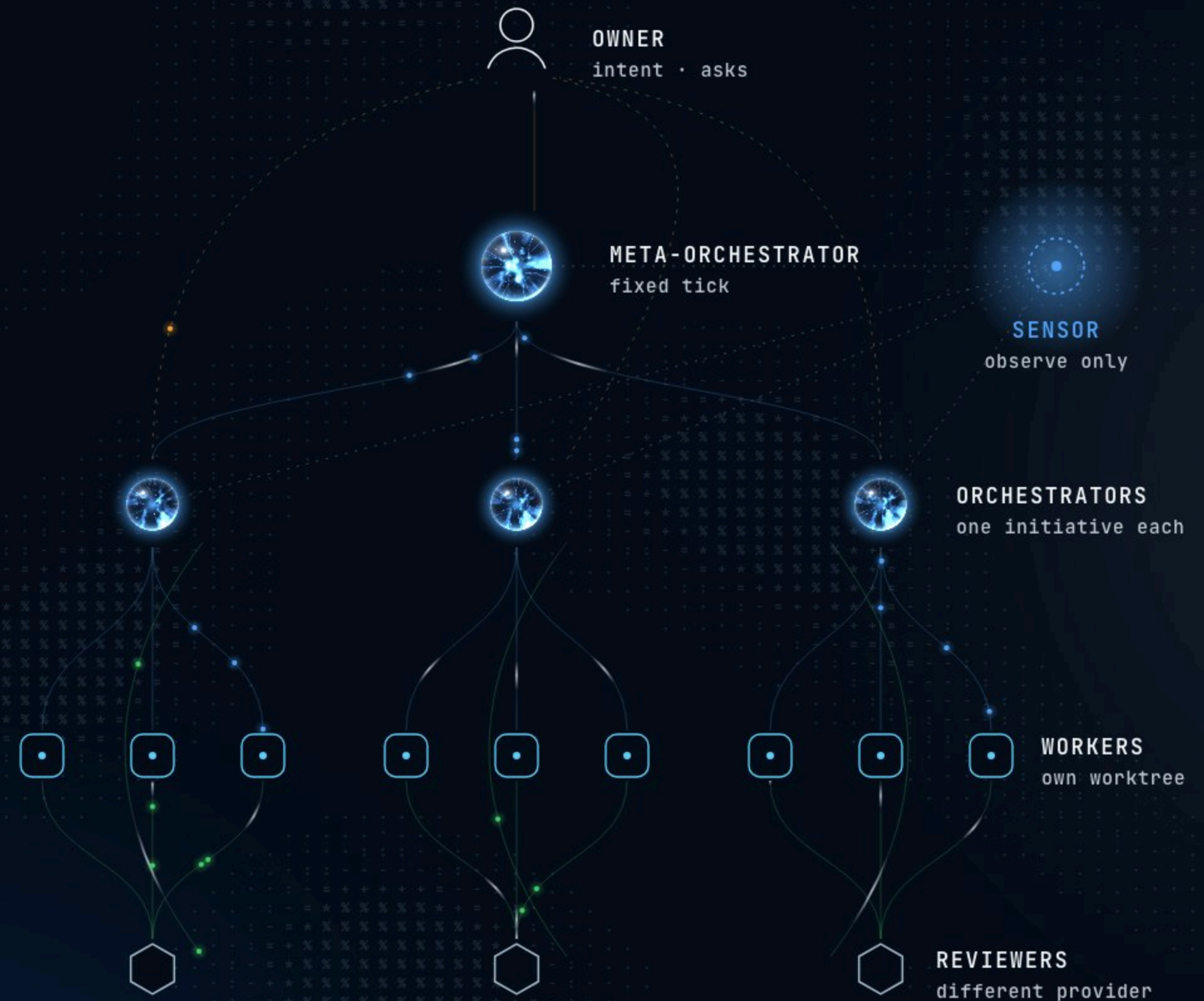
Roles, not agents.

• ROLE 06 OF 06

Sensor

Observes and files: capacity, drift, stalls.

NEVER Acting on what it measures.



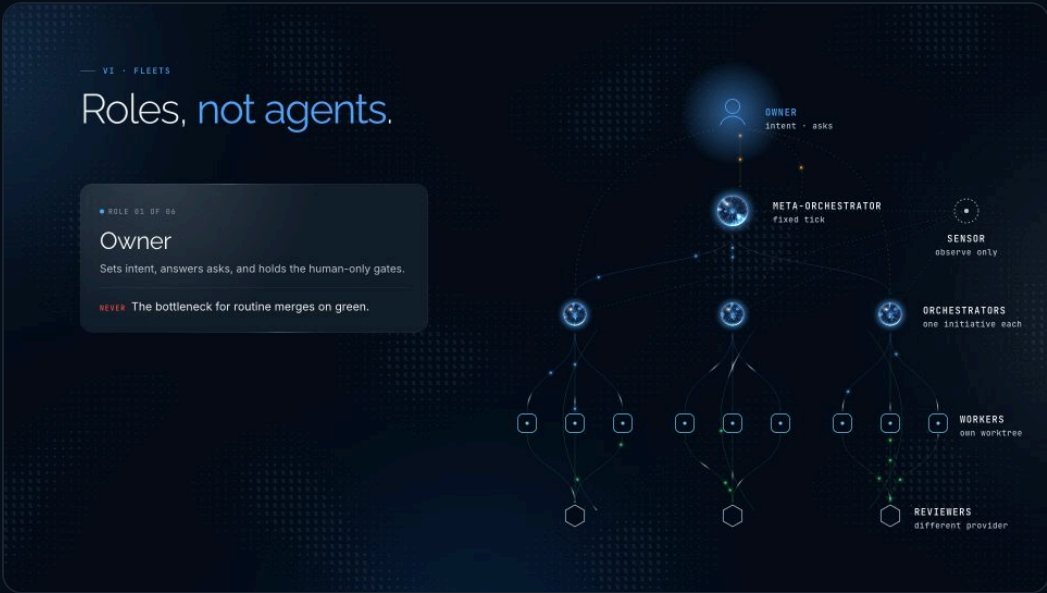
Roles, not agents

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Scale with roles, not with more agents. Each role has a job and, more importantly, something it must never do.

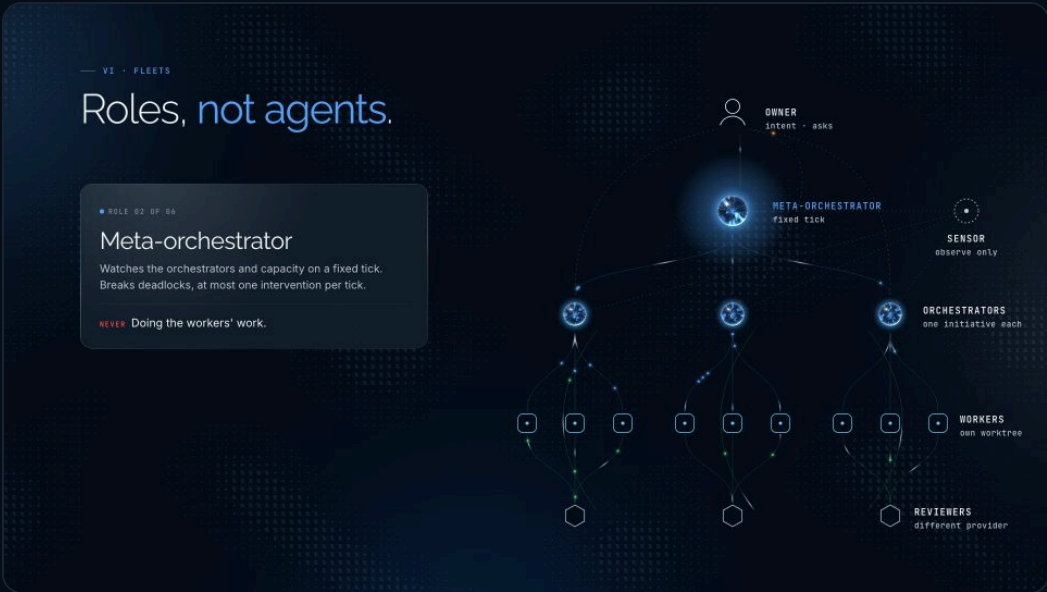
WHAT TO SAY

Step through the roles. The owner sets intent and answers asks. A meta-orchestrator ticks on a schedule and only watches orchestrators and capacity. Orchestrators each own one initiative. Workers are bounded and isolated. Reviewers are never the writer: a different model where possible, at least a fresh context. Sensors observe and never act. The “never” line under each role is the important half.



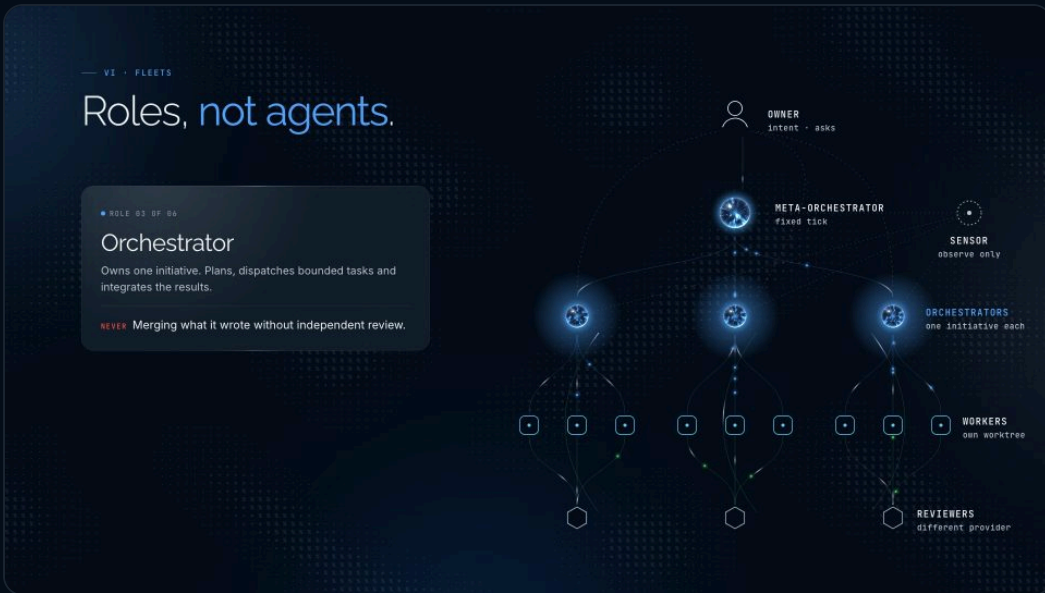
BEAT 1 OF 6

Owner. Sets intent, answers asks, holds human-only gates. Must not become the bottleneck for routine merges on green.



BEAT 2 OF 6

Meta-orchestrator. Watches orchestrators and capacity on a fixed tick, at most one intervention per tick. Does not do workers' work.



BEAT 3 OF 6

Orchestrator. Owns one initiative: plans, dispatches bounded tasks, integrates results. Never merges its own work without independent review.

VI - FLEETS

Roles, **not** agents.

• ROLE 04 OF 06

Worker

One bounded task in an isolated worktree. Returns a PR and receipts.

NEVER Touching another worker's tree.

The diagram illustrates a hierarchical system architecture. At the top is the **OWNER** (intent: asks). Below it is the **META-ORCHESTRATOR** (fixed tick). To the right is the **SENSOR** (observe only). The **META-ORCHESTRATOR** delegates tasks to **ORCHESTRATORS** (one initiative each). These **ORCHESTRATORS** manage **WORKERS** (own worktree). The **WORKERS** interact with **REVIEWERS** (different provider). The **SENSOR** observes the entire process.

BEAT 4 OF 6

Worker. One bounded task in an isolated worktree; returns a PR and receipts.

VI - FLEETS

Roles, **not** agents.

• ROLE 05 OF 06

Reviewer

A different model or provider than the writer. Scores against a rubric and binds the verdict to a commit.

NEVER Reviewing its own output.

The diagram illustrates a hierarchical system architecture. At the top is the **OWNER** (intent: asks). Below it is the **META-ORCHESTRATOR** (fixed tick). To the right is the **SENSOR** (observe only). The **META-ORCHESTRATOR** delegates tasks to **ORCHESTRATORS** (one initiative each). These **ORCHESTRATORS** manage **WORKERS** (own worktree). The **WORKERS** interact with **REVIEWERS** (different provider). The **SENSOR** observes the entire process.

BEAT 5 OF 6

Reviewer. A different model or provider than the writer; scores against a rubric and binds the verdict to a commit.

VI - FLEETS

Roles, **not** agents.

• ROLE 06 OF 06

Sensor

Observes and files: capacity, drift, stalls.

NEVER Acting on what it measures.

The diagram illustrates a hierarchical system architecture. At the top is the **OWNER** (intent: asks). Below it is the **META-ORCHESTRATOR** (fixed tick). To the right is the **SENSOR** (observe only). The **META-ORCHESTRATOR** delegates tasks to **ORCHESTRATORS** (one initiative each). These **ORCHESTRATORS** manage **WORKERS** (own worktree). The **WORKERS** interact with **REVIEWERS** (different provider). The **SENSOR** observes the entire process.

BEAT 6 OF 6

Sensor. Observes and files: capacity, drift, stalls. Never acts on what it measures.

HOW IT RUNS HERE

Orchestrators and workers run as T3 Code threads and delegated child tasks with their own worktrees. Earlier Paseo-hosted sessions are history.

SET IT UP

1. Give every task one owner and one bounded output.

2. Separate the writer, reviewer and observer roles.
3. Define the stop condition and the evidence a worker must return.

PROMPT

For this initiative, define the owner, orchestrator, worker, reviewer and sensor roles. For each, state one bounded responsibility, what it must never do, the evidence it produces, and the handoff point to the next role. Do not create tasks yet.

SKILLS

- **cross-review** skills/governance/cross-review, make review independent of authorship
- **handoff** skills/orchestration/handoff, preserve state between bounded tasks

REFERENCES

- [bstack/references/delegation.md](#),
[bstack/references/orchestrator-tick.md](#)
- [docs/maestro-orchestration-loop.md](#)

Spawn, reuse, or hand off.

Independent of what I know?

Spawn a child task in its own worktree.

Needs my context?

Continue here, or fork the thread.

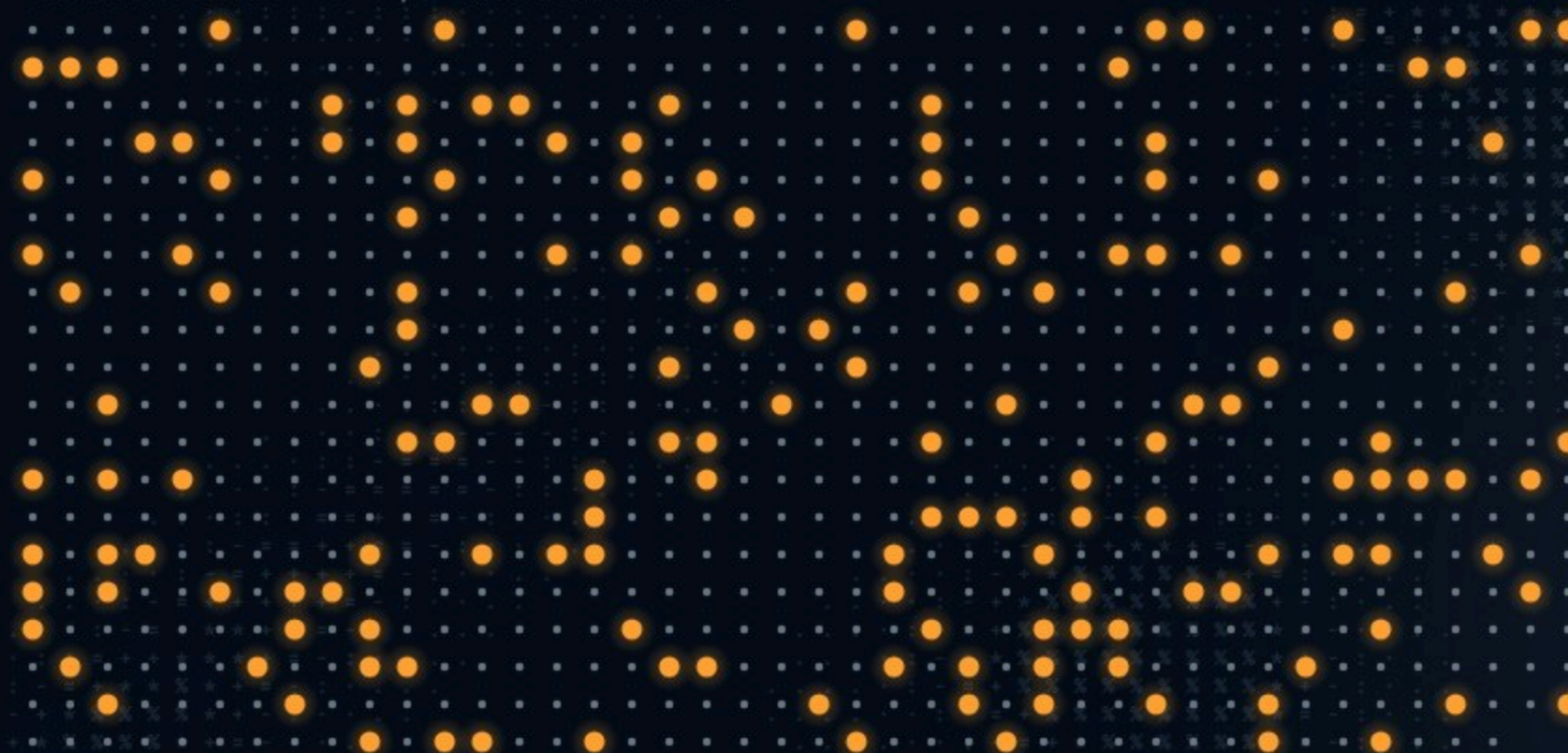
Outlives this context window?

Write a persistent brief with stop conditions.

Only the owner can decide?

File an ask: context, options, a recommendation and a safe default.

838 main sessions · 30 days · measured 2026-10-10



Long sessions carry most of the token cost. Roll over with a handoff.

Handoff

narrative for the next agent

Handback

message for the human

Brief

loop state across contexts

Ask

a decision, with its context

Receipt

evidence it actually works

Spawn, reuse, or hand off

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Four questions choose the mechanism for the next piece of work.

WHAT TO SAY

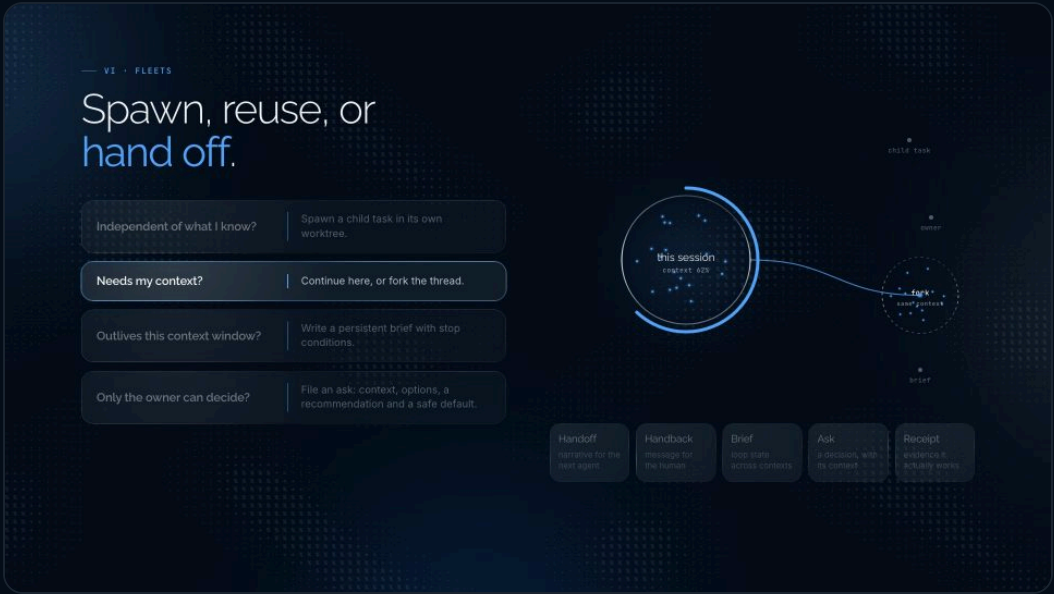
Four questions decide the mechanism. Independent work gets a fresh child in its own worktree, which also avoids anchoring it on your conclusion. Work that needs your context continues here.

The chart is why you shouldn't stretch one session forever: in a month of our transcripts, 158 of 838 sessions ran past 400K tokens and carried 87% of the cost. Long sessions roll over to a successor with a handoff doc.

WHAT YOU'RE SEEING



Independent of what I know? Spawn. A child task takes the work to its own worktree with a fresh context, which also keeps it from anchoring on your conclusion, and comes back with a receipt.



Needs my context? Keep it. The task joins this session and the context arc grows; a fork copies the same context into a new thread when you want two lines of work.



BEAT 3 OF 5

Outlives this context window? Write a brief: progress, checks, stop conditions. This session can end; a successor reads the brief and carries on.



BEAT 4 OF 5

Only the owner can decide? File an ask with its context: summary, options with trade-offs, a recommendation, sources. The answer comes back to

the session that asked.



BEAT 5 OF 5

Why sessions roll over. Each dot is one of 838 main sessions. The 158 that ran past 400K tokens grow and turn amber: they carried 87% of the cost.

WHY SESSIONS ROLL OVER

In a 30-day sample (measured 2026-10-10), 158 of 838 main sessions ran past 400K tokens and carried 87% of the cost. Long sessions hand off to a successor with a handoff document instead of growing forever.

THE FIVE ARTIFACTS

- **Handoff:** narrative for the next agent (docs/handoffs/).
- **Handback:** a message for the human.
- **Brief:** machine state for a persistent loop.

- **Ask:** one decision, with its context.
- **Receipt:** evidence that it works.

SET IT UP

1. Use a fresh isolated task for independent work.
2. Write a handoff before a task needs more context than the current session can safely carry.
3. Escalate only decisions that cannot be resolved from the recorded evidence.

PROMPT

We are near the end of this context. Write a handoff to docs/handoffs/YYYY-MM-DD-<arc>.md: a three-line TL;DR, the current state (branch, PRs with SHAs, what is verified and what is not), the first action for the next session, and open asks with your recommendation.

SKILLS

- **handoff** skills/orchestration/handoff
- **persist** skills/orchestration/persist

REFERENCES

- docs/conversations/Conversations.md, conversation index
- .control/asks/, decisions that need owner authority

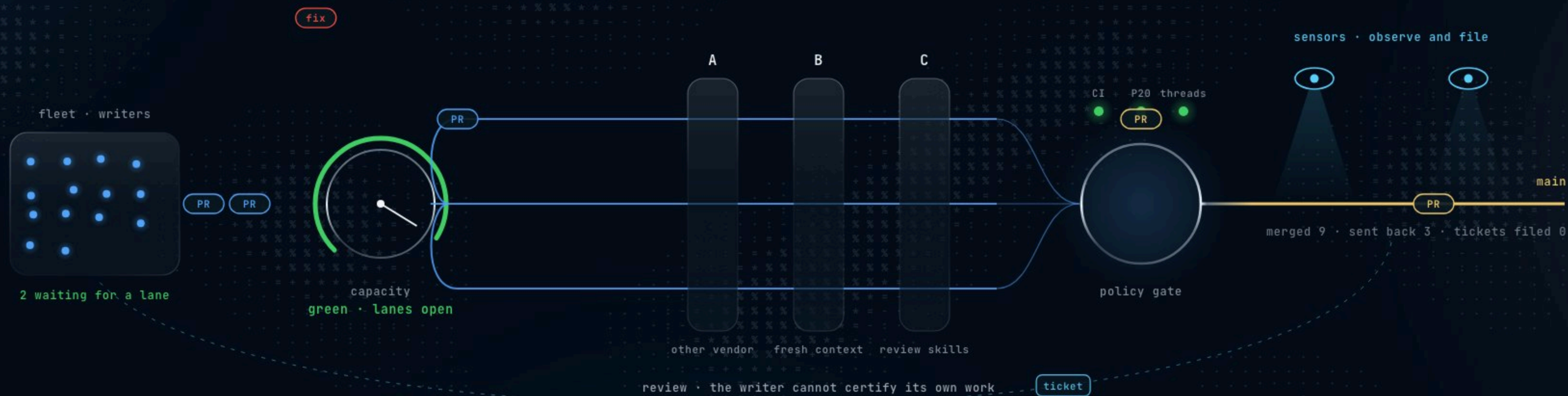
Governance that scales with the fleet.

Pace

Review

Policy

Sensors



PACE

From measured capacity, not hope

REVIEW

The writer cannot certify its own work

POLICY

Executable gates. Prose is only advice

SENSORS

Observe and file. Agents act

Governance that scales

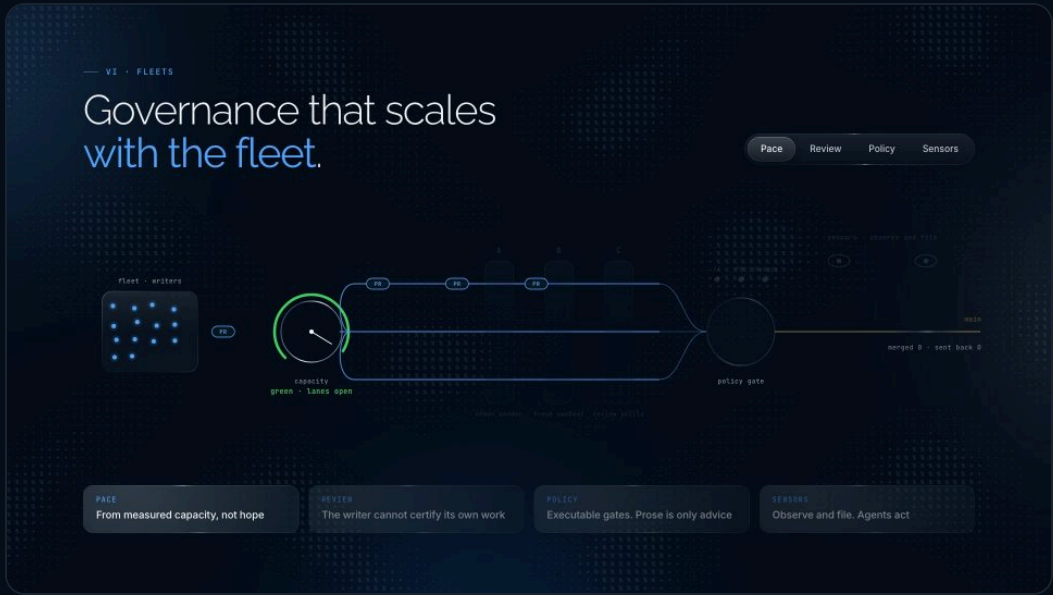
Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Governance has to scale with the number of agents, or it becomes the bottleneck.

WHAT TO SAY

Four principles, one pipeline, left to right. Pace: a governor sets lanes from measured headroom, several on green, one on amber, none on red, so work queues instead of failing mid-run. Review: an independent reviewer for substantive work, three strata for governance; rejections go back to the writer. Policy: an executable gate before main. Sensors: they watch main and file tickets, and agents act on them. Governance scales because each of these is a mechanism, not a meeting.

WHAT YOU'RE SEEING



BEAT 1 OF 4

Pace. A capacity governor reads measured account headroom and sets the lanes: several on green (five in the configured non-Claude green fallback; actual limits depend on the provider and meta setpoints), one on amber, none on red. The dial shows the pattern; the setpoints move. Work queues at the dial instead of failing mid-run.



BEAT 2 OF 4

Review. Substantive PRs get an independent review before merge; governance changes need all three strata: a different vendor's model, a fresh-context devil's advocate, and composed review skills. Rejections arc back to the fleet as fixes; the writer never certifies its own work.



ACT VII

Leadership

What does this ask of people?

Act VII · Leadership

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Act VII makes human judgment the scarce resource: use loops for repeated work, clear contracts for delegation, narrow asks and shared standards.

WHAT TO SAY

One liquid-metal loop with spectral light at its highlights: the unit of work is now a loop.

Act VII turns the system back to people: loops, delegation contracts, attention and visible standards.
Read next: The unit of work is now a loop.

ACT SCENES

1. The unit of work is now a loop
2. Everyone leads a fleet
3. Attention is the new ration
4. Bars rise in the open

READ NEXT

Continue to **The unit of work is now a loop** to distinguish a one-off task from a learning cycle with evidence.

The unit of work is now a loop.

When an attempt costs minutes, making gets abundant and
deciding gets scarce.

Task

Loop

Every question is something you can start today.

Every repeated task is something you can automate or improve.

Every piece of work leaves the system better: a note, a skill, a
check.

The question moves from “can I do this?” to “should this exist, and
how will I know it's right?”



making

deciding

scarce

decisions taken 2

The unit of work is now a loop

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

From *The New Dynamics of Agentic Leadership*: when an attempt costs minutes, the unit of work stops being a task and becomes a loop.

WHAT TO SAY

Last act: what this asks of people. A task waits on someone deciding it's worth doing, then one attempt. A loop states intent, lets agents act in parallel, looks at what came back, sharpens the intent and runs again.

Watch the two gauges: the scarce resource moves from making to deciding. This is the premise of The New Dynamics of Agentic Leadership, the essay this act comes from.



BEAT 1 OF 2

Task. Someone decides it is worth doing, a ticket waits in a queue, one person attempts it once.



BEAT 2 OF 2

Loop. State an intent, let agents act in parallel, look at what came back, sharpen the intent, run again. Making becomes abundant; deciding becomes scarce.

THE NEW BASELINE

- Every question is something you can start today.
- Every repeated task is something you can automate or improve.
- Every piece of work leaves the system better: a note, a skill, a check.
- Time agents free up is reinvested on purpose.

SET IT UP

1. State an observable intent for the loop.
2. Run a bounded attempt, capture the evidence and update the next attempt from it.
3. Stop when the evidence condition is met or a named escalation is required.

PROMPT

Turn this recurring task into a bounded loop. Define its intent, one reversible first iteration, the evidence to collect, the decision that updates the next iteration, a cost ceiling, and the condition that stops or escalates it. Do not start the loop yet.

SKILLS

- **agentic-leadership** skills/strategy/agentic-leadership, the source of this act: five-term contract, attention rationing, ask template (published in broomva/skills; run `git -C skills pull` if your checkout lacks it, then link it into your harness's skills folder)
- **persist** skills/orchestration/persist, carry bounded loop state across contexts

- **decision-log** skills/strategy/decision-log, preserve the decision that changes the next run

REFERENCES

- [The New Dynamics of Agentic Leadership](#) (X article)
- docs/references/agent-operating-protocols.md, Persist (P12) and Orchestrate (P19)

Everyone leads a fleet.

• TERM 05 OF 05

Review

who judges the result before it ships

You do not certify your own work: a separate session or a different model does.



What each person brings to their fleet is taste and judgment: knowing what good looks like, and deciding what to accept.

Everyone leads a fleet

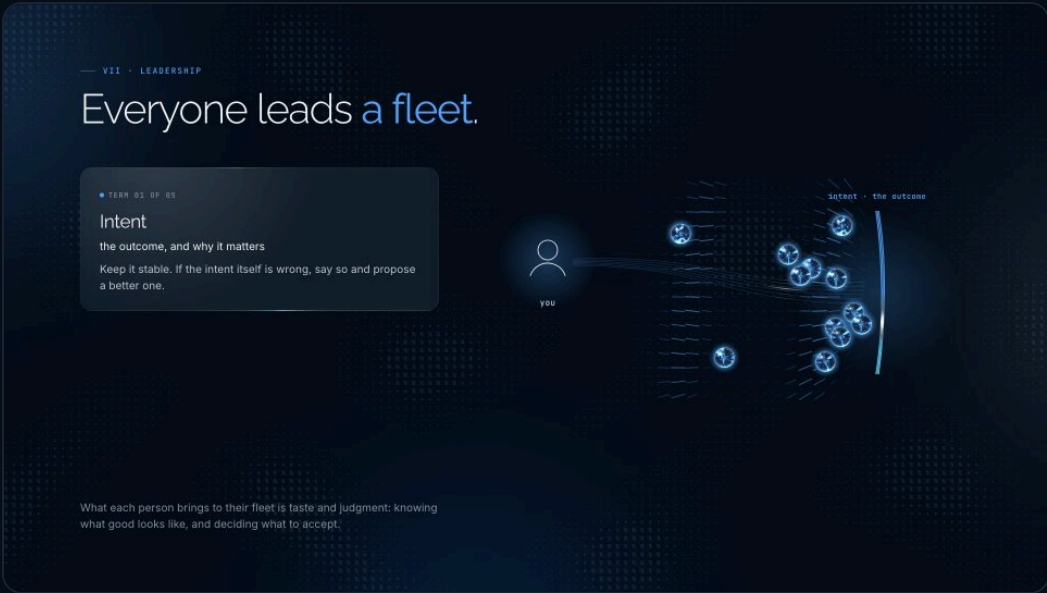
Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Every delegated piece of work, to an agent or a person, runs under a five-term contract.

WHAT TO SAY

Delegation runs under a five-term contract, for agents and for people: intent, bounds, evidence, escalation, review. Step through the terms: each one adds a force to the same fleet and keeps acting after you move on. Intent turns wandering agents into a flow toward the goal. Bounds are a wall they bounce off, and the one heading for the red zone is stopped. Evidence streams receipts into your ledger. An escalation stops one agent until you answer. Review is a judge of a different colour before work lands on the gold line.

None of this is specific to code. Software got there first because tests made verification cheap.



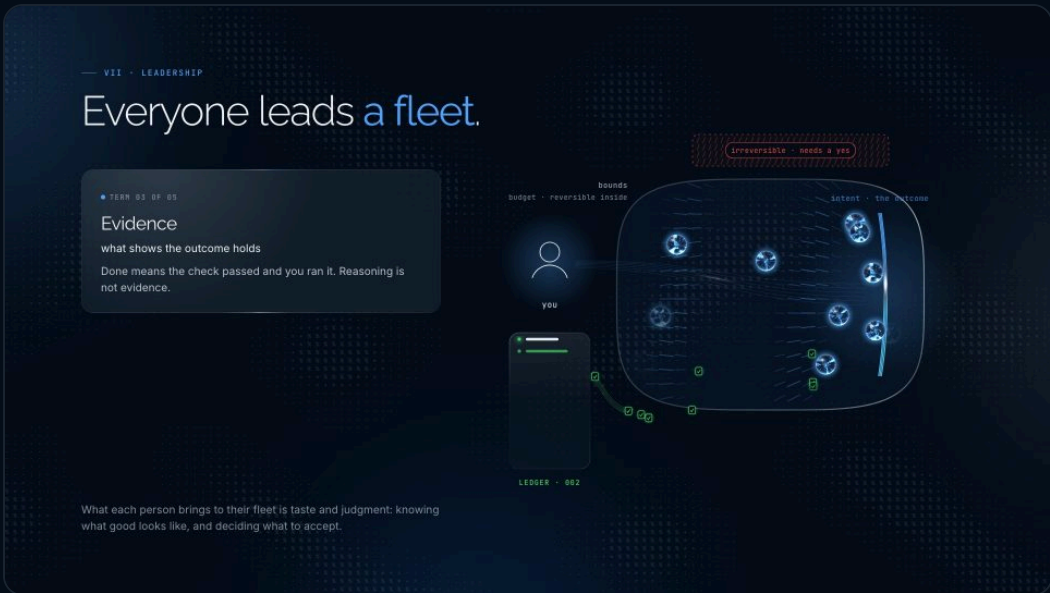
BEAT 1 OF 5

Intent: the outcome and why it matters. Keep it stable; if it is wrong, say so and propose a better one.



BEAT 2 OF 5

Bounds: what may be decided without asking, the budget, and what cannot be undone.



BEAT 3 OF 5

Evidence: the observable check that means done. Reasoning is not evidence.

VII · LEADERSHIP

Everyone leads a fleet.

TEEN 04 OF 05

Escalation

what goes to someone else, and when

Every ask carries context, options, a recommendation and a safe default.

What each person brings to their fleet is taste and judgment: knowing what good looks like, and deciding what to accept.

you

LEADER · 014

bounds

budget · reversible inside

escalation · an ask with options

intent · the outcome

irreversible · needs a yes

BEAT 4 OF 5

Escalation: when to stop and ask. Every ask carries context, options, a recommendation and a safe default.

VII · LEADERSHIP

Everyone leads a fleet.

TEEN 05 OF 05

Review

who judges the result before it ships

You do not certify your own work: a separate session or a different model does.

What each person brings to their fleet is taste and judgment: knowing what good looks like, and deciding what to accept.

you

LEADER · 029

bounds

budget · reversible inside

escalation · an ask with options

intent · the outcome

irreversible · needs a yes

review · a different judge

merged · the gold line

BEAT 5 OF 5

Review: who judges the result before it ships. You do not certify your own work.

SET IT UP

1. Write the intent and boundary before dispatch.
2. Name an observable check that the worker cannot self-certify.
3. Define the escalation and review owner before work begins.

PROMPT

Before you start, write the contract for this work in five lines: Intent (observable outcome and why), Bounds (what you may do without asking; what needs a yes; the budget), Evidence (the check that means done), Escalation (when you stop and ask), Review (who judges before it ships). If a term is missing, propose it, then wait for my confirmation only if it involves cost or anything irreversible.

SKILLS

- **agentic-leadership** skills/strategy/agentic-leadership, the source of this act
- **decision-log** skills/strategy/decision-log, record the decision boundary
- **cross-review** skills/governance/cross-review, keep final judgment independent

REFERENCES

- .control/asks/, decision records with options and recommendations
- docs/references/agent-operating-protocols.md, Dep-Chain (P14) and Cross-Review (P20)

Attention is the new ration.

Every thread pings you

The gate calls you

Reversible drafts, research, branches: act and note it.

Costly long compute, many iterations: only inside a budget.

Irreversible or external spend, publish, merge outside policy, contact: a yes first, every time.

The measure of an autonomous system isn't how much it does without you. It's how precisely it calls you.

40 threads

gate

1 · merge to production?

2 · three GPU hours?

3 · publish the post?



reaching you: three ordered asks

budget

- reversible · act and note
- costly · inside a budget
- irreversible · needs a yes

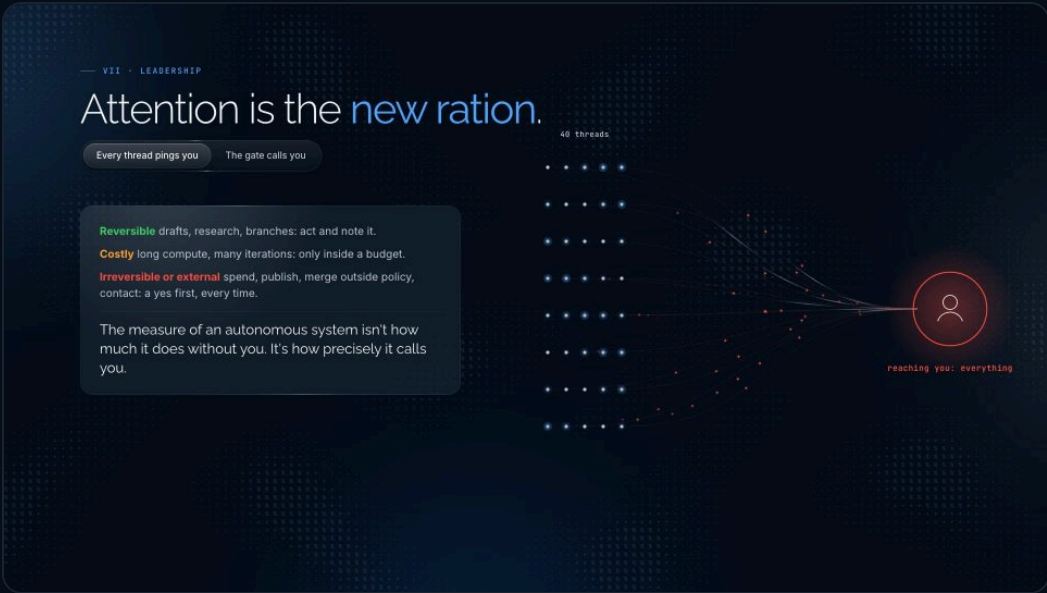
Attention is the new ration

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Attention is fixed. The job of an autonomous system is to spend it only where it is needed.

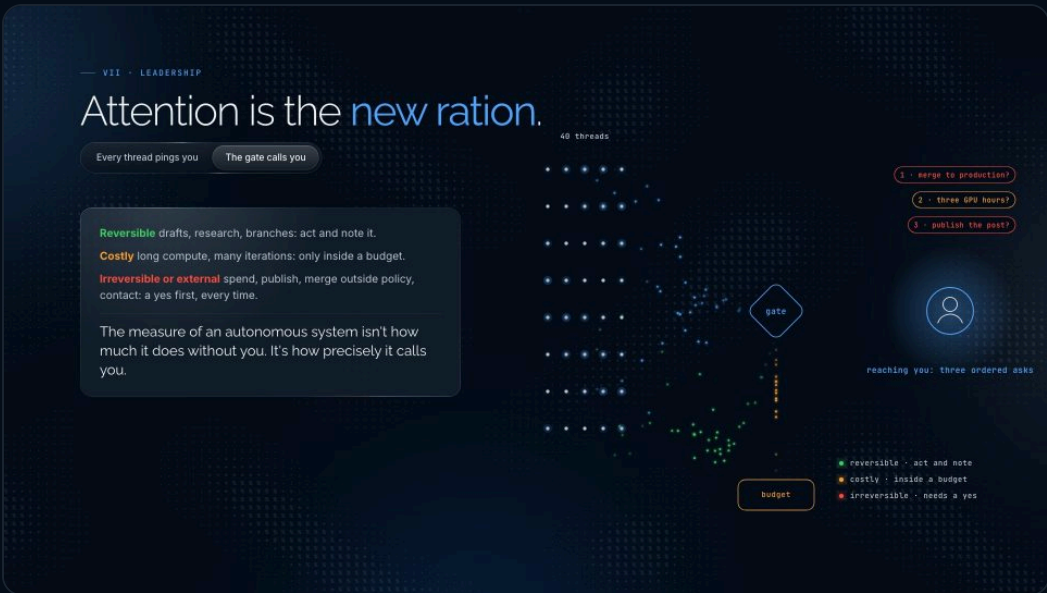
WHAT TO SAY

Forty threads that each want a look will eat a day.
Classify every action: reversible work proceeds and is noted, costly work runs only inside a stated budget, and irreversible or external work waits for an explicit yes.
Toggle the gate: only a short ordered list reaches the human, each ask with context, options, a recommendation and a safe default. That is what the fleet governance in act six was for.



BEAT 1 OF 2

Every thread pings you. Forty threads each wanting a look will consume a day.



BEAT 2 OF 2

The gate calls you. Actions are classified. Reversible work proceeds and is noted. Costly work runs only inside a budget. Irreversible or external work waits for a yes. Only a short, ordered list reaches the human.

The measure of an autonomous system isn't how much it does without you. It's how precisely it calls you.

SET IT UP

1. Classify planned actions before they begin.
2. Record a budget for costly work.
3. Batch only the decisions that need owner authority, with options and a reversible default.

PROMPT

Classify every action in your plan as reversible (act and note), costly (needs a budget) or irreversible/external (needs my explicit yes). Do all reversible work now. Then send me one ordered list of decisions, the one that unblocks the most first, each with context, two or three options, your recommendation and what you will do if I do not answer.

SKILLS

- **agentic-leadership** skills/strategy/agentic-leadership, the source of this act
- **decision-log** skills/strategy/decision-log, preserve decision context and alternatives
- **persist** skills/orchestration/persist, keep a bounded work state without repeated asks

REFERENCES

- .control/asks/DECISION-LAYER.md, decision-record contract
- .control/policy.yaml, executable safety boundaries

Goalposts teach waiting. Bars rise in the open.

Goalpost moves

Bar rises

People learn to wait when proposals get overridden, the leader decides anyway, the bar moves silently, or a failed attempt costs more than a passive miss.

Make everything queryable: decisions, standards and why they changed live in the same shared graph that people and agents read.

SHARED CONTEXT · decisions/bar.md
read by every person and every agent

● v4 · + every claim cites a source
why: sources are one query away

● v3 · + review by a second model
why: cross-review got cheap

● v2 · + a dogfood receipt
why: agents can drive the app now

● v1 · tests pass
the first bar

query beam

the bar

● people ● agents

climbing 7/7

Bars rise in the open

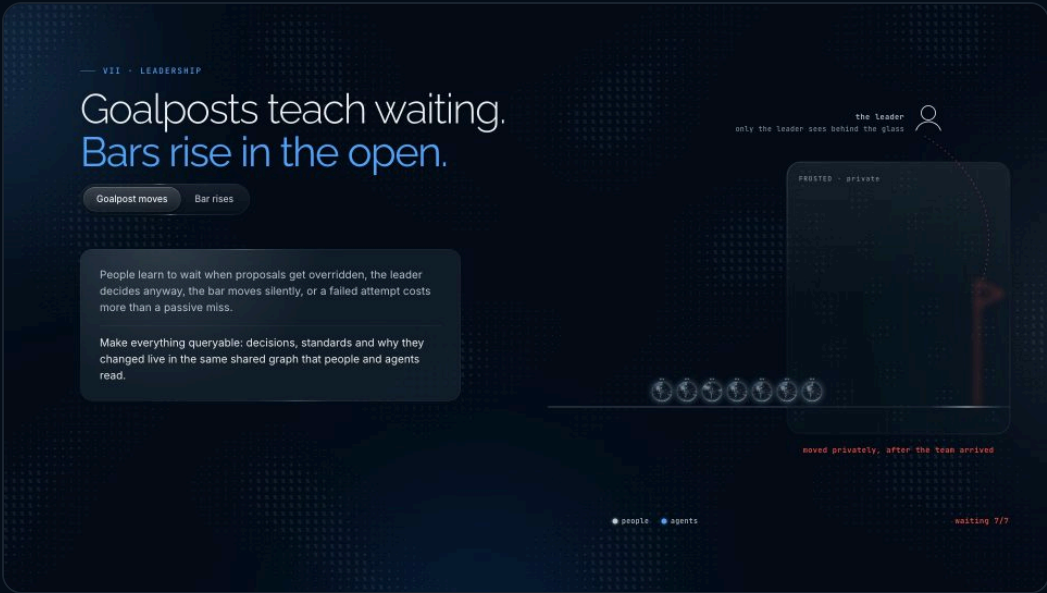
Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

In agentic work the bar is supposed to rise. What matters is who moves it, and whether everyone can see it move.

WHAT TO SAY

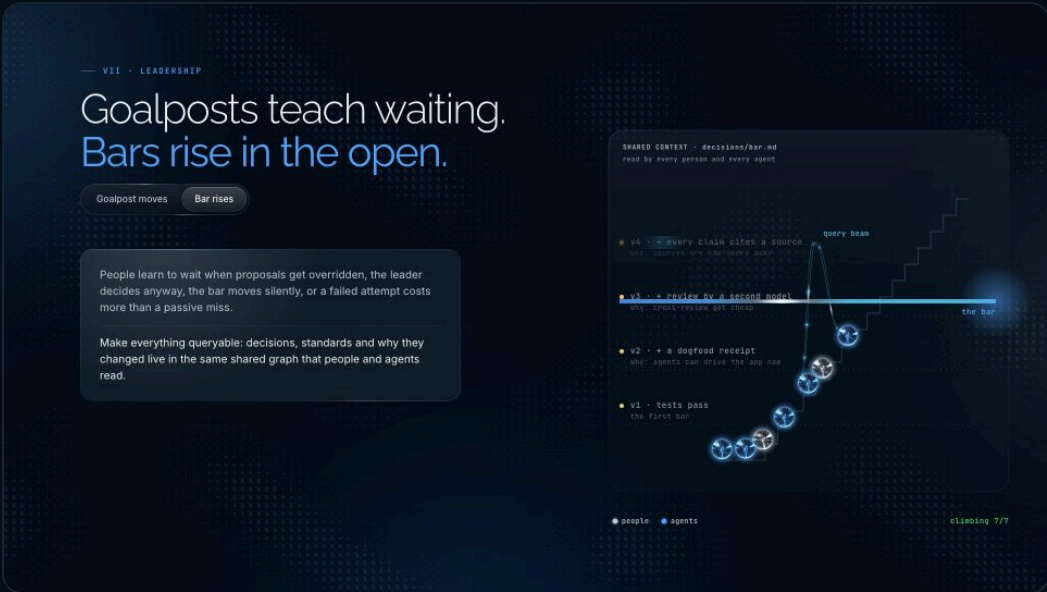
In agentic work the bar is supposed to move. The problem is who moves it and whether anyone can see it. First beat: the goalpost sits behind frosted glass that only the leader sees, and it moves after the team arrives, so the team stalls and waits. Second beat: the bar is written on a shared glass wall that people and agents query; each rise is logged with its reason, and the team keeps climbing.

That loops back to the second brain: the same queryable workspace that grounds agents also makes standards visible to people.



BEAT 1 OF 2

Goalpost. Success is redefined privately, after the work. People learn that any attempt may be judged by a rule that does not exist yet, so they wait.



BEAT 2 OF 2

Bar. Capability changes, the standard rises, and the reason is written into shared, queryable context. People and agents read the same standard, so they look ahead.

WAITING IS OFTEN LEARNED

People learn to wait when their proposals get overridden, the leader decides anyway, the bar moves silently, or a failed attempt costs more than a passive miss.

MAKE EVERYTHING QUERYABLE

Decisions, standards and why they changed live in the same graph the agents read (Act III). That closes the loop between grounding agents and leading people.

SET IT UP

1. Record the current standard, source and evidence in a durable file.
2. State the event that changed the standard.
3. Link the decision from the work and knowledge records that rely on it.

PROMPT

Write a decision record for this changed standard: the prior bar, the new bar, the evidence that required the change, affected work, exceptions, review date and links to the source records. Do not treat an unverified observation as a standard.

SKILLS

- **agentic-leadership** skills/strategy/agentic-leadership, the source of this act
- **decision-log** skills/strategy/decision-log, record decisions with context
- **bookkeeping** skills/knowledge/bookkeeping, link durable claims to their evidence

REFERENCES

- docs/knowledge-index.md, queryable knowledge catalog
- docs/references/agent-operating-protocols.md, Audience (P18) and Bookkeeping (P6)

EPILOGUE

Better agents come from better harnesses.

● LAYER 06 OF 06

Workspace

The place that holds all of it and keeps what every session learns. This is where the harness compounds.

MODEL the same weights in every layer

OUTCOME illustrative, not measured

evals · 78 passed · 13 failed

workspace · 009 deposits kept



Harness engineering

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Harness engineering is the discipline of everything around the model. It is where reliability comes from, and where work compounds as models change. The model at the centre never changes in this picture; the outcome meter beside it is illustrative, not measured.

WHAT TO SAY

Harness engineering is everything around the model, built layer by layer around the same orb: the loop, docked tools, the context shell, the guard rim that rejects what policy forbids, the scanning evals, and the gold workspace lattice that keeps what every layer deposits. Step outward. The meter on the left is illustrative: the model stays the same, the outcome rises.

Models will keep changing, and so will harness products. Harness engineering, everything around the model, is where work compounds; the workspace is

its outermost ring.



BEAT 1 OF 6

Model and loop. The model proposes; the loop runs tools and feeds results back.



BEAT 2 OF 6

Tools. A filesystem and a shell first; MCP for remote systems.



BEAT 3 OF 6

Context. Primer, memory, hooks, skills and the graph decide what reaches the window.



BEAT 4 OF 6

Guards. Policy gates, permissions and hooks make boundaries executable.



BEAT 5 OF 6

Evals and review. Tests, receipts and a different model judging before merge.



BEAT 6 OF 6

Workspace. The outermost ring: it holds all of the above and keeps what every session learns.

SET IT UP

1. Inventory the outer rings before replacing the model or adding prompts.
2. Make every boundary observable with a path, command or interaction receipt.
3. Strengthen the smallest weak ring, then run the check that proves it changed.

PROMPT

Audit this workspace's harness ring by ring: tools (what the agent can run), context (what loads and when), guards (what is executable versus prose-only), evals and review (what checks run before merge), and the workspace record (what each session leaves). For each ring, name the weakest point and the smallest change that would strengthen it.

SKILLS

- **harness-engineering-playbook** skills/governance/harness-engineering-playbook
- **audit-harness-usage** skills/tooling/audit-harness-usage, measure what the harness costs in tokens across traces

REFERENCES

- [Effective context engineering for AI agents](#)
- docs/references/agent-operating-protocols.md, empirical and review requirements

— EPILOGUE

Twenty primitives. One loop.



The workspace is the product.

01

Give agents a place before you give them more autonomy.

02

Make the graph readable at intake, or stop writing to it.

03

Scale with roles, isolation and review. Not with more prompts.

Twenty primitives, one loop

Local paths and workspace measurements are dated author-workspace examples, not files shipped in this package. See resources.md for public tools and setup.

Everything in this deck reduces to twenty primitives. Each runs when its trigger applies.

WHAT TO SAY

Everything tonight reduces to twenty small primitives on one loop, read in five phases: orient, bound, act, verify, learn. Snapshot before planning, gate before acting, review by a different evaluator, record what you learned. A light passes each one in turn.

Then (→) the loop folds into the orb: three takeaways, and the thesis. The workspace is the product.



BEAT 1 OF 2

The loop. The operating vocabulary: small disciplines with explicit triggers, rather than one vague instruction to be careful. This deck reads them in five phases. Orient: Snapshot, Bridge, Lens, Freshness, Dep-Chain. Bound: Gate, Hygiene, Tickets, Orchestrate. Act: Fanout, Wait, Persist, Audience. Verify: Empirical, Cross-Review, Pipeline. Learn: Bookkeeping, Crystallize, Dream, Janitor.



BEAT 2 OF 2

The thesis. The loop resolves into one orb. The closing cards reduce the deck to place, recall and governed collaboration: the workspace is the product.

THREE TAKEAWAYS

1. Give agents a place before you give them more autonomy.
2. Make the graph readable at intake, or stop writing to it.
3. Scale with roles, isolation and review, not with more prompts.

SET IT UP

1. Choose one repository and one small, reversible improvement.
2. Define its evidence check before editing.
3. Run the check, record the receipt and use it to choose the next improvement.

PROMPT

Set up this repository as an agentic workspace in five small PRs: (1) an AGENTS.md routing primer and a CLAUDE.md copy; (2) a policy file plus a PreToolUse gate for destructive commands; (3) research/entities/ with a front-matter schema and an index builder; (4) a UserPromptSubmit hook that injects the top matching entities; (5) a Stop hook that writes each session to docs/conversations/. One PR at a time, with tests and a receipt for each.

SKILLS

- **dogfood** skills/governance/dogfood, verify changes by use
- **cross-review** skills/governance/cross-review, obtain independent judgment before delivery

REFERENCES

- AGENTS.md (primitives and routes), bstack/references/primitives.md
- docs/references/agent-operating-protocols.md
- Speaker script and sources: docs/keynote/agent-workspaces/script.md

Sources

FIGURE OR CONTENT	SOURCE	AS OF
0.9% of 565 sessions loaded the graph	Session-start self-improvement telemetry (kg_load_rate 0.0088 over 7 days)	2026-10-26
1,430 entities · 6,076 edges	docs/knowledge-index.md header	2026-10-25
Entity graph (scene 17)	Related-neighbour traversal [[link]] neighbourhood of agent-harness-three-layer-split (concept, pattern and tool types only; slugs only)	2026-10-26
Front-matter fields and /kg_load flags (scenes 16, 13)	skills/knowledge/kg/SKILL.md; entity front matter in research/entities/	2026-10-26
Hook stations and their jobs (scene 12)	.claude/settings.json registrations (SessionStart ×5, UserPromptSubmit, PreToolUse on Bash/Write/Edit, Stop); plugin and user-level hooks add more (the bstack plugin's SessionStart hook injects the board facts and the self-improvement metric)	2026-10-26
115 authored skills · 23 categories · 31 harness folders	skills/skills/**/SKILL.md; dot-directories containing skills/	2026-10-26
66 product repos (scene 25)	git repositories under apps/ (42 direct children), core/ (13 direct children) and work/ (11, one or two levels down)	2026-10-27
Catalog ~260K tokens at 1,430 entities (scene 13)	docs/knowledge-index.md is 1,026,411 bytes	2026-10-27
Lanes on green (scene 30)	CONCURRENCY in the capacity governor (five in the configured non-Claude green fallback; actual limits depend on provider and meta setpoints); setpoints are adjusted from measured usage	2026-10-27
457 installed → 26 descriptions	Roster measurement in owner memory (skill-biases-a-tool-binds)	2026-08-23
158 of 838 sessions > 400K carry 87% of token cost (API-equivalent)	30-day transcript cost analysis (BRO-3067, workspace #983)	2026-10-10
Worked example (scene 9)	A trial review system's architecture decisions on medallion-in-filesystem with a Postgres system of record, and on harness-driven review; patterns only, no client, case or data	2026-10-26
Leadership act (32–35)	<i>The New Dynamics of Agentic Leadership</i> (X article by broomva_tech), plus the workspace decision and continuation protocols	2026-10-25