

Fleet control for coding agents.

One terminal over every project and every agent you have running — and one **manager** you task in prose. It plans, spawns coders, testers and reviewers, reopens what fails, and reports READY with the PRs and the evidence. It never merges. That part stays yours.

```
$ curl -fsSL https://raw.githubusercontent.com/AISquare-Studio/aisquare-cli/main/install.sh | sh
```

5

roles in one loop

1

SQLite file of state

0

daemons · accounts · cloud

V0.6.0 · SHIPPING

MIT · LOCAL-FIRST

FOR ENGINEERING TEAMS

PAYMENTS-API · LIVE

manager ▶

coder-idempotency ▶

coder-webhooks 🔔 NEEDS YOU

tester-1 ▶

reviewer-1 ⏸

coder-retries 🔄 PR #412

Six real Claude Code sessions, one window. Click one and every key you type goes to the session itself. Close the UI and all six keep running.

Memory

Sessions start oriented instead of cold.

EVERYONE

Orchestration

A fleet on one board, working one problem.

OPT-IN PER REPO

Governance

Every session becomes a Run you can read.

OPT-IN

Five agent terminals is not five times one agent.

Coding agents got good enough to run several at once. Nothing got good enough to govern several at once — so the human became the scheduler, the courier and the only memory in the system.

Every session starts cold You re-explain the codebase, the conventions, and the thing you already said last week. The agent greps its way back to what it knew yesterday, and you pay for that twice — in tokens, and in the corrections you have to type again.	You are the message bus One agent per terminal. Five agents means five mental models and a human copy-pasting findings between them. Two coders quietly edit the same file. A tester verifies a branch nobody told it about.	Nothing is on record When an agent does something you did not want, you cannot read back what it was shown, what it decided, or what it cost. So the same misunderstanding arrives again next sprint, and nobody can point at where it came from.
---	---	--

Prose is a terrible protocol — a watcher grepping `READY` fires on a note saying `NOT READY`.

— the reason board signals are first-class named states here, and not string matching

WHAT WE ARE ACTUALLY SELLING

Not more agents. A **place to put them** where the coordination is the ambient state of a shared board rather than something a person carries between windows — and where the work stops at a gate a human owns.

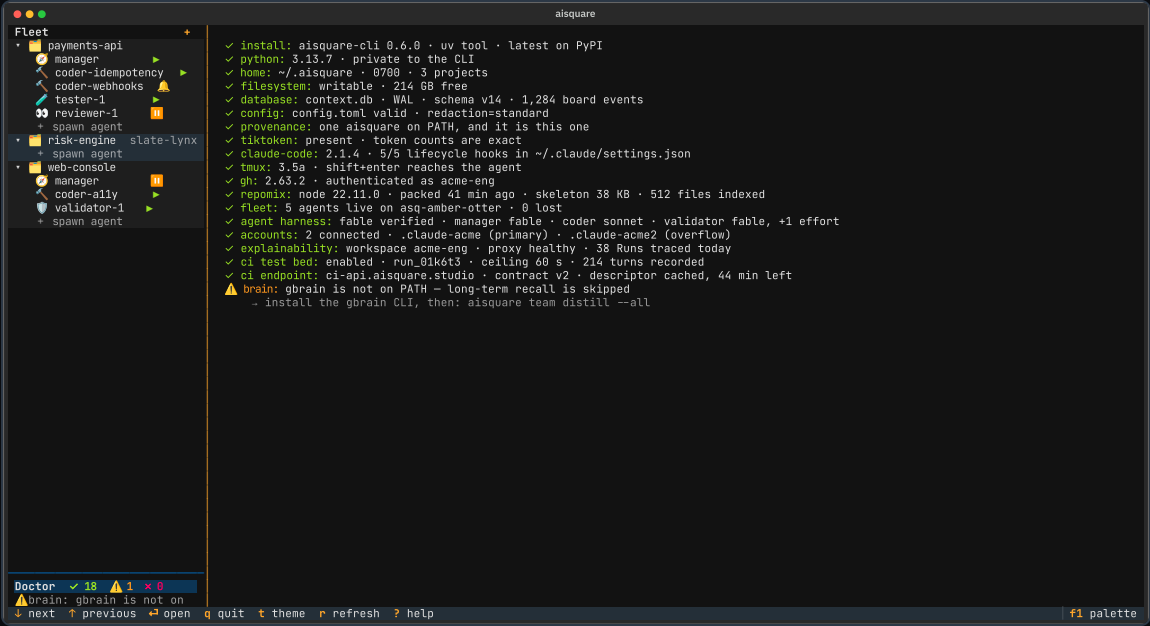
One line, then the machine tells you it is ready.

The installer works out what the machine already has and installs only the gaps — uv, a private Python for the CLI alone, tmux, git, gh, Node, Claude Code — registers the repo you ran it in, wires the hooks, and offers to open the UI. Run it again and it installs nothing.

```
$ curl -fsSL .../install.sh | sh
```

Not a leap of faith --dry-run prints every command and runs none. Or install by hand with <code>uv tool install</code>; that stays fully supported.	Careful with your machine Refuses to run as root outside a container, uses <code>sudo</code> only for system packages and one command at a time, and never edits your shell profile beyond what <code>uv</code> does itself.
Windows works A PowerShell one-liner sets up WSL2 and runs the same script inside it — the fleet gives each agent a real tmux pane, and Windows has no tmux.	Its own CI shellcheck, shfmt, dash and ash, across five containers, with one cell run as a normal user under <code>sudo</code> — the primary case, not the root shortcut.

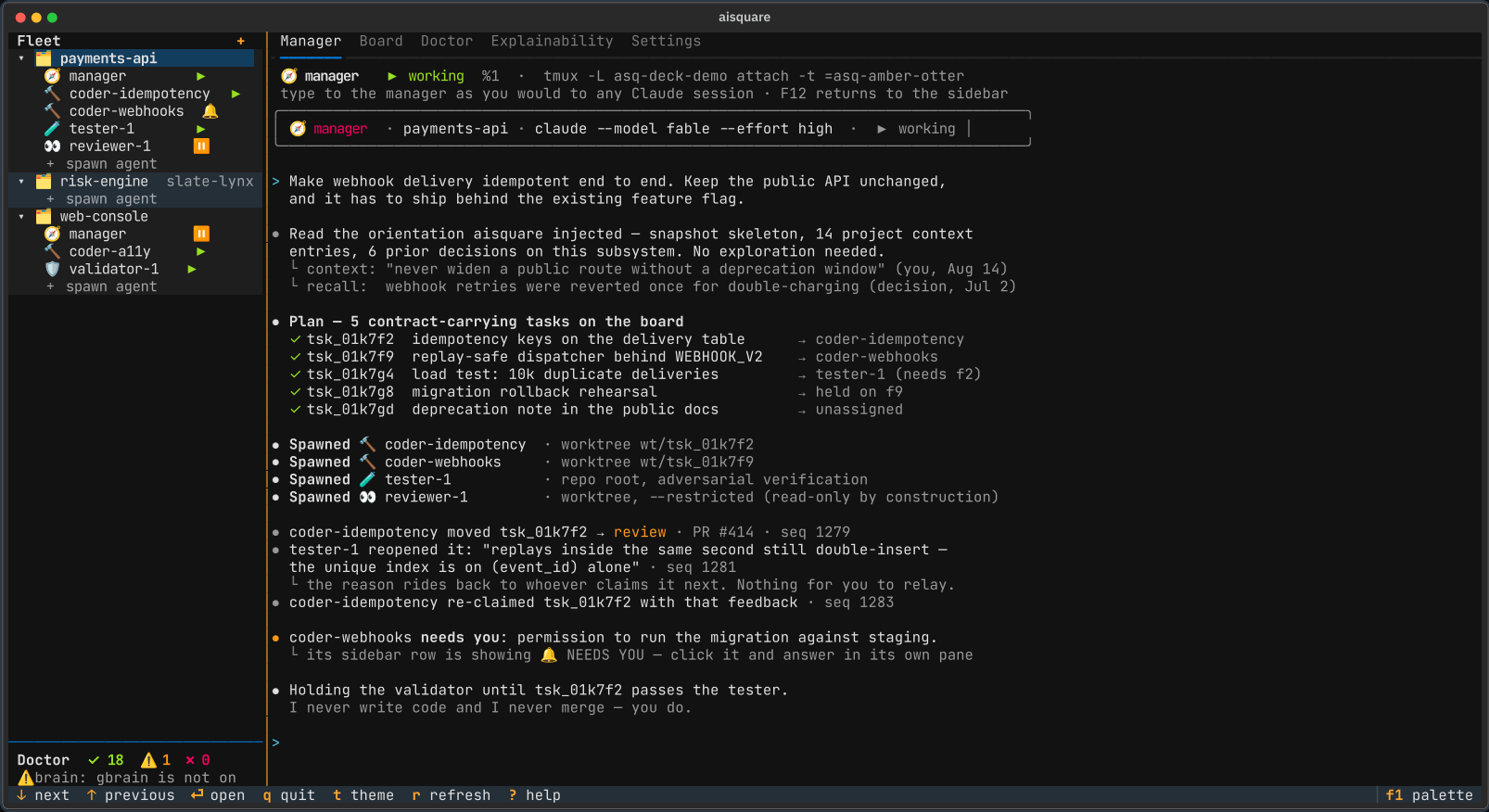
Exit codes mean something: 0 installed, 1 a fatal step failed, 2 installed but a health check came back unexpectedly amber.



Every dependency checked, with the exact command for anything missing. Whether the hooks are actually installed, whether the snapshot is stale, which models the harness has verified for this account, which config directories are connected — and, here, one honest amber for an optional component that is not on this machine. Worked example, not a customer's machine.

You type a goal. It writes the contracts.

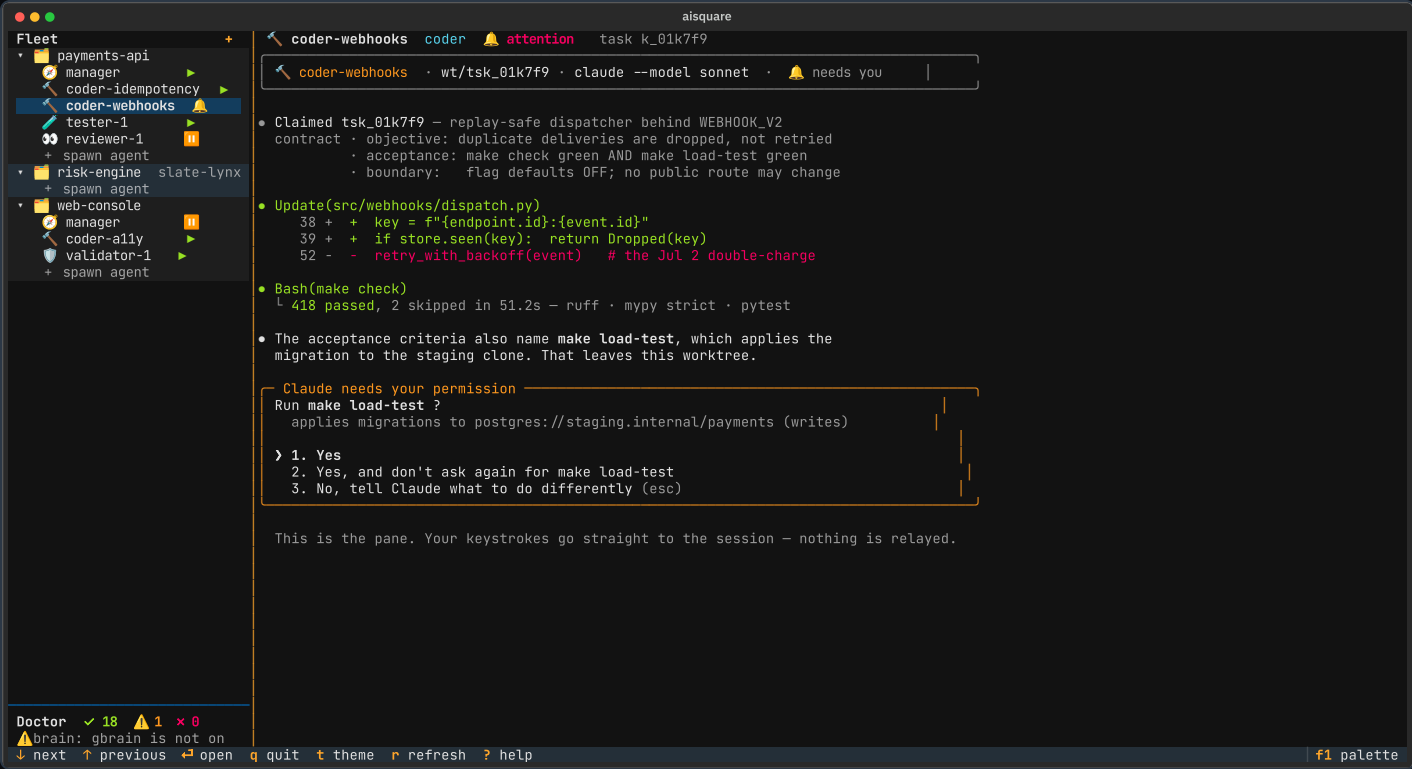
Press Start manager, then talk to it exactly as you would to any Claude session. It reads the orientation aisquare handed it, turns your intent into tasks that carry an objective, a why, acceptance criteria and boundaries, and spawns the agents the work needs.



One goal in prose, five contracts on the board, four agents spawned, and a tester's reopen already coming back. Note what the manager was handed before it explored anything: the packed skeleton, fourteen project context entries, and two prior decisions about this very subsystem — including the time these retries were reverted for double-charging. Worked example, not a customer's repo.

Real sessions. Not a chat wrapper over them.

Every agent is a Claude Code process running as a window on a tmux server that is ours alone — your own tmux sessions, config and prefix key are never touched. Click an agent and you are in the session: the same rendering, the same keys, the same permission prompts.



The coder hit a decision it will not take alone, so its row went **NEEDS YOU** and the terminal rang. Worked example, not a customer's repo.

Close it, nothing stops

The UI is a view over that tmux server and the board. Reopen asq and it re-attaches to whatever it finds; fleet attach shows the same session from any terminal at full fidelity.

The keys reach the agent

Claude Code's own bindings must not be swallowed, so the UI drops Textual's defaults that would steal them and moves its palette to F1. F12 is the one key a pane never forwards — it hands focus back to the sidebar.

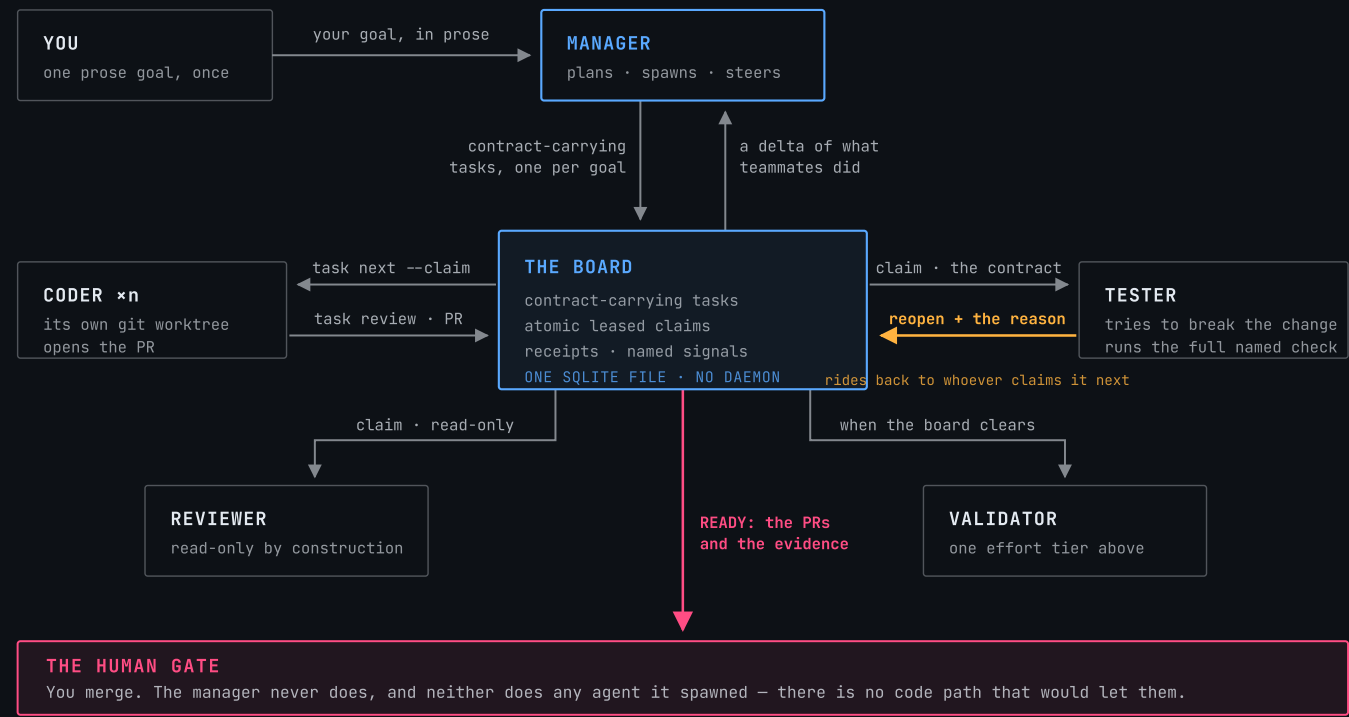
Coders get their own worktree

Parallel coders never edit the same tree. The reviewer runs --restricted in a worktree of its own: read-only by construction, not by instruction.

State chips are earned

▶ working, || waiting, 🛑 needs you, 🏹 exited — from the same lifecycle hooks the board uses. An agent launched without them says no hooks rather than guessing.

The board is the only channel.



The runner is the adversarial verifier: it tries to make the change fail, then closes the task with evidence – or reopens it with what failed.

– the tester's own briefing, injected at launch

Claims are atomic and leased

A single race-tested UPDATE: exactly one winner. Leases renew from the session's hooks, so a dead session hands its work straight back to the pool.

Dependencies are held

task next hands out only work whose dependencies are done — a rollback rehearsal cannot start before the migration it rehearses.

Nothing is relayed by a human. The manager writes contracts and reads a delta of what teammates did on every prompt; the coder claims atomically; the tester runs the full named check and tries to break the change.

Every claim, decision, reopen and signal, in sequence.

Each successful write prints a receipt — ✓ ... seq N — and the pull side is yours any time. team verify 42 asks the board whether that sequence number is really there and exits 0 or 1; a receipt that lives on a different board is an honest not-found that names the board holding it.

team

planner.ses_mgr0 ▶ working
claude-fable-5-1 21m
Ⓢ Make webhook delivery idempotent end to end — public API unchanged, behind the existing flag
validator.ses_val0 ▶ working
claude-fable-5-1 21m
runner.ses_tst0 ▶ working

tasks — d for done archive

id	st	who	title
3ym4r5ev	doing	ses_cod0	Idempotency keys
kgtxnjje	review	ses_cod0	Replay-safe webho
fzf42k9h	doing	ses_tst0	Load test: 10k du
a4wj12nr	todo		Migration rollbac
r7jaax8t	todo		Deprecation note

live feed

12:54 planner.ses_mgr0 Ⓢ is focusing on: Make webhook delivery idempotent end to end — public API unchanged, behind the existing flag
12:54 planner.ses_mgr0 ⚡ added: Idempotency keys on the webhook delivery table — coder
12:54 planner.ses_mgr0 ⚡ added: Replay-safe webhook dispatcher behind flag WEBHOOK_V2 — coder
12:54 planner.ses_mgr0 ⚡ added: Load test: 10k duplicate deliveries, zero double-charges — runner
12:54 planner.ses_mgr0 ⚡ added: Migration rollback rehearsal on a staging clone — runner
12:54 planner.ses_mgr0 ⚡ added: Deprecation note in the public webhook docs — coder
12:54 planner.ses_mgr0 ⚡ decided: Idempotency key is the (endpoint_id, event_id) pair, not a random UUID — replays must collide.
12:54 coder.ses_cod0 🏆 claimed: Idempotency keys on the webhook delivery table
12:54 coder.ses_cod0 🏆 claimed: Replay-safe webhook dispatcher behind flag WEBHOOK_V2
12:54 coder.ses_cod0 ⚡ decided: Public API unchanged: the key is derived server-side, callers send nothing new.
12:54 coder.ses_cod0 Ⓢ sent to review: Idempotency keys on the webhook delivery table — PR #414. make check green. Verify: pytest tests/test_webhook_replay.py -k duplicate
12:54 runner.ses_tst0 🏆 claimed: Load test: 10k duplicate deliveries, zero double-charges
12:54 runner.ses_tst0 📡 bounced back: Idempotency keys on the webhook delivery table — replays inside the same second still double-insert — the unique index is on (event_id) alone
12:54 coder.ses_cod0 🏆 claimed: Idempotency keys on the webhook delivery table
12:54 coder.ses_cod0 📋 reported: Reproduced the tester's case; widening the index to (endpoint_id, event_id) and re-running the load test.
12:54 coder.ses_cod0 Ⓢ sent to review: Replay-safe webhook dispatcher behind flag WEBHOOK_V2 — PR #417. Flag defaults off. Verify: make check && make load-test
12:54 coder.ses_cod0 ? asked: Do we need a deprecation window for the old dispatcher, or is the flag enough? — planner
12:54 runner.ses_tst0 • signal: load-test-bed: ready
12:54 planner.ses_mgr0 ⚡ decided: Flag is enough — the old path stays until the next minor. No window needed.

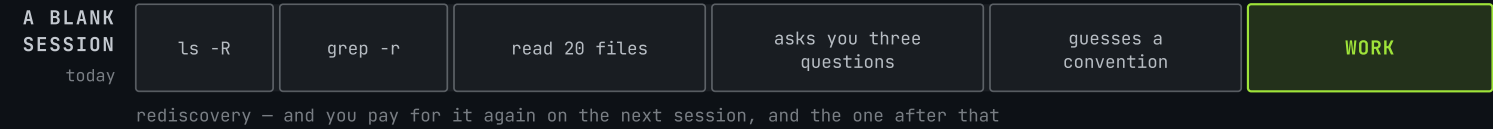
detail — click a task or feed line

b board on/off r refresh a autoscroll v select text c copy d done archive o transcript q quit t themes s screenshot ^p palette

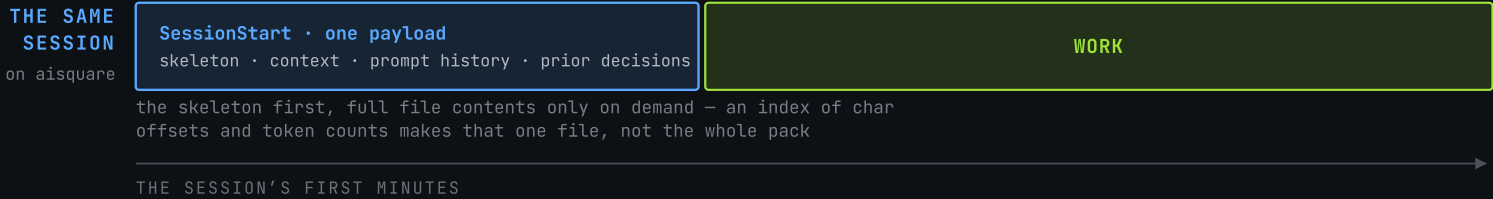
These rows are real. The shipping CLI was driven through the sequence above — five contracts written, two claimed, one sent to review, one reopened by the tester with its reason, one question routed to the planner, one signal set — and this is what the board rendered. Click any task or feed line for its full detail; o opens the author session's transcript at that exact moment.

A blank agent is an expensive agent.

Connecting Claude Code writes five lifecycle hooks into its settings, merged carefully and never clobbering yours. From then on every session opens on a directive pointing at a structure-only skeleton of the repo, the context entries in scope, and what you have asked for in this project before.



Same prompt. Same model. Same repo. One hook is the only difference.



aisquare remember "prefer pytest over unittest"	Sticks everywhere, in every project.
aisquare context add "run make check" --project	Sticks in this repo only.
aisquare why	What the last session was shown, and why.
aisquare recall "what did we decide about auth?"	Across sessions and weeks, from the distilled brain.

Two pools, one behaviour

user follows you everywhere; project is scoped to one repo. Both full-text searchable, exportable, and injected consistently.

Worktrees share the repo

Identity comes from the git common dir, so feature branches checked out side by side share one context pool, one snapshot and one board. Set the conventions up once.

Same prompt, same repo, same model. Full file contents come only on demand – a per-file index of character offsets and token counts lets an agent open one file's slice of the pack instead of the whole thing.

Every role on the right model – and it is checked.

claude --model silently substitutes the default when a model is not available to an account. So the harness probes, verifies the reply before trusting a rung, and caches that verdict per account for a day.

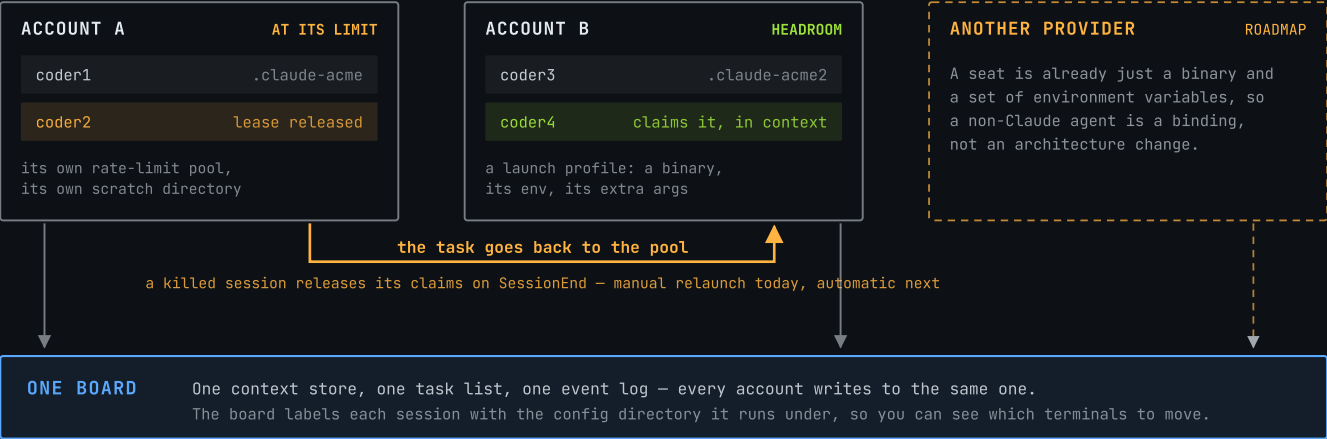
```
$ aisquare team harness
base effort: xhigh (inherited)
manager    fable→opus→sonnet    effort=xhigh    ( 0) → fable
coder      sonnet→opus          effort=xhigh    ( 0) → sonnet
tester     sonnet→opus          effort=xhigh    ( 0) → sonnet
reviewer   sonnet→opus          effort=xhigh    ( 0) → sonnet
validator  fable→opus            effort=max      (+1) → fable
Resolution shown without probing – `team spawn <role>` verifies live.
```

Captured from the shipping CLI. The base effort is inherited from the session you spawn the fleet from, so raising yours raises theirs with nothing to configure.

Demote only on proof A genuine substitution walks down the ladder. An outage, an expired login or an unrecognised reply keeps your pick and labels it [unverified] — rather than quietly downgrading your manager. Nothing here ever blocks a launch.	The gate outranks the work The validator carries a +1 effort offset for one reason: a flat override that dropped everything to low would leave the gate weaker than the coder it reviews, which is not a gate.
The probe is isolated It never executes the current repo's hooks or MCP servers, and never joins the board. A capability check should not have side effects.	Off-ladder is flagged, not policed Sessions report their model back and the board marks any that is off its role's ladder. Advisory by design: a missing chip means not reported, never wrong.

Several accounts. One team.

Sessions are per terminal, not per account — a single install runs the whole team. Connect each config directory once and bind each seat to the environment it launches with; several accounts simply mean several rate-limit pools driving one board.



Today this is a deliberate relaunch; the recovery is what is already automatic. Because claims are leased and released on session end, a killed seat hands its task back and the next claim picks the work up with the board's full context — the contract, the review note, the reason it was reopened.

Set both variables, or it lies to you

The config directory alone gives a session the right credentials and the default scratch directory, silently shared — which looks correctly isolated right up until two parallel sessions collide in temp. The CLI says so, and doctor reports every connected directory separately.

A seat is a launch profile

A binary, a set of environment variables and extra arguments, carried through verbatim, with ~ and \$VAR expanded at launch — so one binding follows you across machines with different homes.

ON THE ROADMAP, FROM HERE

- The handoff without the relaunch: a seat hits its ceiling, the fleet moves the work itself
- The same seat abstraction pointed at another provider — a binding, not an architecture change

Every session becomes a Run you can read back.

Point your sessions at an AISquare Explainability workspace and each one becomes a record: what was asked, what the model answered, every tool call, the tokens and the cost — alongside your own prompts and the board events that framed the work. Two independent lanes, configured together, keyed on one session id.

LANE	CARRIES	HOW IT TRAVELS
Proxy	Model traffic — prompts, responses, tool calls, tokens, cost. Claude Code emits no telemetry of its own, so something has to sit in the request path, record the exchange and forward it upstream.	agent → proxy → gateway
Client	Your prompts, board notes, task claims and session events — the human half of the record, which no proxy can see.	CLI → local spool → ship → gateway

A board row, a live process and a dashboard Run share one id Which is what turns “an agent did something odd on Tuesday” into a specific transcript, at a specific sequence number, with the prompt that caused it.	It fails quietly, on purpose If anything in the path is down, the session starts untraced and says so. Observability that can stop your developers working is not observability.
Either lane, without the other They are separate paths. Knowing which is which turns most confusion into a one-line answer.	Or keep it local The proxy is hosted for you by default, and self-hosting is documented for teams that would rather it were not.

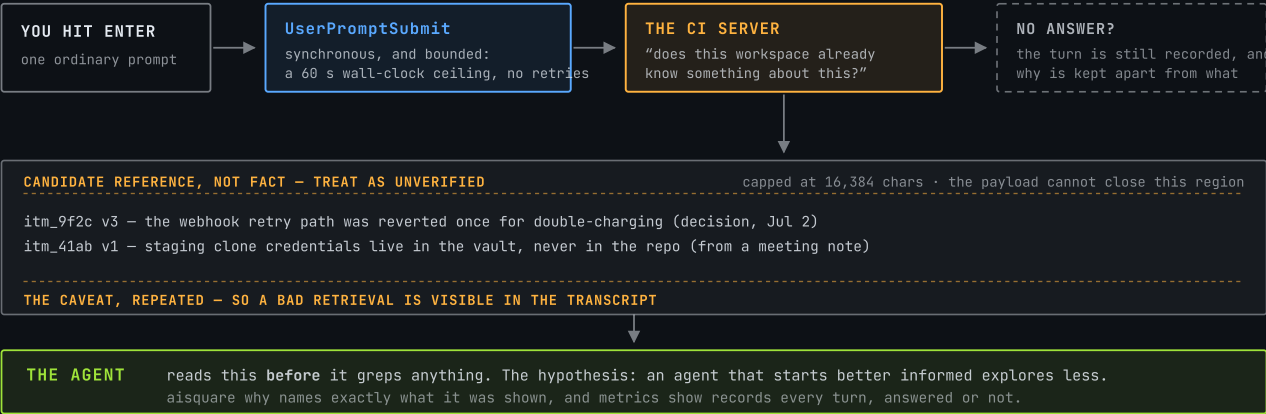
WHAT THIS IS FOR

Two audiences, one record. An engineer reads a Run to find out why an agent did something. A lead reads the same Runs to answer what the fleet cost this week, which roles burn the most, and whether the instructions you keep repeating are actually reaching the sessions that need them.

Off unless you ask for it. Nothing on this slide runs for a default install.

Put what the team already knows in front of the agent.

The hypothesis is narrow and testable: an agent that starts better informed explores less. So when a prompt is submitted, aisquare can ask a Collective Intelligence server whether this workspace already knows something relevant, and hand it over **before** the agent starts looking around.



Retrieval is framed, capped and attributed — never silently absorbed.
The server's own delivery descriptor decides which hooks call it, where, and under what ceiling; the CLI honours only what that lists, and the descriptor names no architecture or arm, so the client is structurally unable to know which one it is running.

What leaves the machine

The prompt, scrubbed at your configured redaction level; a project selector; and a git object id of the working tree, so a turn can be replayed. Nothing about scope. No credentials. Untracked files are not in it.

Off costs nothing

With the switch off there is no request, no connection and no measurable latency — and any unrecognised value of the switch is off.

WHAT WE WILL NOT CLAIM YET

Token counts are **not** recorded yet — hook payloads do not carry them — so the metrics command says plainly that token savings cannot be read from it. The server is live against staging, it measures nothing, and it is a connectivity instrument. When that changes, we will show you the numbers rather than the hypothesis.

This is also where connectors land: meeting notes, tickets and design docs feeding the same knowledge base, so an agent stops assuming what “the usual way” means.

Harmless by construction.



each one a window on `tmux -L asq` — our server, never yours.
Close the UI and every one of them keeps running.

FAIL-OPEN Any error inside a hook is swallowed and the session continues untouched. A repo that never opts in sees nothing at all.

Orchestration is opt-in per repository and fails open everywhere. The hooks are designed so orchestration cannot break a Claude session — even when it is itself broken, or absent.

Everything is a command Every action the UI takes is a plain CLI command, and every one takes <code>--json</code>. In a pipe or under <code>TERM=dumb</code> you get the help page and exit 2, byte for byte, so no script ever meets a full-screen app.	Roadmap stays hidden Unfinished commands are hidden from <code>--help</code> and exit 70 saying plainly that they are not implemented — rather than half-working. What the help lists, works.
Contracts, not vibes The CI server's seven JSON Schemas and their fixtures are vendored byte for byte, and every request this build can emit is validated against the server's own schema in the suite — never against our reading of it.	Tested where it hurts The store is concurrency-tested against racing parallel sessions; claims are race-tested for exactly one winner; the suite passes with every aisquare environment knob set adversarially.
MIT, and readable A thin Typer CLI over a service layer over one SQLite store. Your team can read the whole thing, and fork it if we disappear.	No lock-in at the data layer Context exports to Markdown or JSON and imports from either. The board is a table in a file you own.

Now, next, and the part we are honest about.

The discipline in the product is the discipline in this list: nothing is claimed as shipping unless you can run it today, and nothing on the roadmap is dressed up as almost-done.

● NOW • V0.6.0, INSTALLABLE – The fleet UI and the manager loop, end to end – Memory: two context pools, packed snapshots, prompt history – Board: contracts, leased atomic claims, receipts, named signals – Model ladder per role, probed, with effort offsets – Several accounts driving one board – Remote agents over MCP, so a browser-debugging agent can join – Explainability Runs, and optional long-term recall – The one-line installer, with its own CI matrix	● NEXT – Sign in — one identity across your machines – sync — memory that follows you, and then your team – On-limit handoff — a seat hits its ceiling and the fleet moves the work itself – Another provider as a seat binding, not a rewrite – Remote control — watch and steer the fleet away from the terminal These are registered commands today: hidden from help, and honest about not being finished.	○ LATER – Connectors — meeting notes, tickets and design docs into the knowledge base – Organisation policy the agents cannot cross, enforced rather than asked for – A knowledge graph of findings, decisions, prompts and preferences Collective Intelligence is the seam all three land on. It is live against staging, off by default, and measuring nothing yet.
---	--	--

Follow it in the open: [issues on github.com/AISquare-Studio/aisquare-cli](https://github.com/AISquare-Studio/aisquare-cli). The changelog says what shipped, what was wrong, and what a review found — which is the fastest way to judge whether we are the kind of team you want inside your repository.

One repository. One afternoon.

Pick a repo with real parallel work in it. We install, register it, and run the loop on a live goal with your team in the room — then hand back the transcript, the board and the Runs, and you decide whether the fleet earned its afternoon.

```
$ curl -fsSL .../install.sh | sh      # only the gaps get installed
$ asq                                # click + and point it at the repo
... press Start manager, and type the goal in prose.

$ aisquare doctor                    # every check, with its fix
$ aisquare --json fleet ls           # the same view, for your automation
```

What it needs from you	What it does not need
A Claude Code login your developers already have, tmux 3.2+, git and gh. Nothing to deploy and no account to create.	No daemon, no server, no cloud dependency, and no traffic leaving the machine unless you switch that on deliberately.

HOW TO JUDGE IT AFTERWARDS

- Did the manager need relaying?**

Count the times a human had to carry a finding from one agent to another. The target is zero, and the board is where you check.
- Did the tester catch something real?**

Look for a reopen with a reason a reviewer would have written — and then whether the next claim actually acted on it.
- Did the second session start oriented?**

Open a fresh agent on the same repo and read `aisquare why`. It should already know what you spent the afternoon explaining.

► **aisquare / asq**

pypi.org/project/aisquare-cli
github.com/AISquare-Studio/aisquare-cli
MIT · screens captured from the shipping UI at v0.6.0