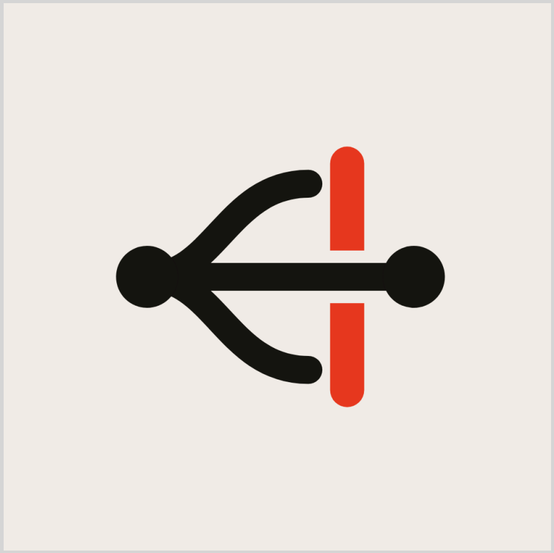
© 2026 Yiguedan Philippe Andy Yaro



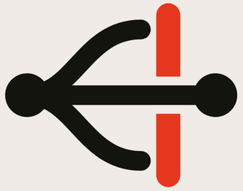
orkestra: Many Agents, One Verified Result

Yiguedan Philippe Andy Yaro

View GitHub Repository: github.com/andyyaro/orkestra

Let's connect! Email: yyaro@gmu.edu

LinkedIn: linkedin.com/in/andy-yaro



orkestra lets two or more AI coding agents, programs such as Claude Code, OpenAI's Codex and Google's Antigravity that edit code from a terminal, work on one Git repository at once.

Each task gets its own working copy; my own test commands decide whether work passes; and nothing lands until a different agent has reviewed it.

It is a Python package, orkestra-runtime, on PyPI, Python's package index, under the Apache-2.0 license. A coding agent built the first version from a build prompt I wrote on 24 July 2026; 16 more releases followed by 8 September.

This deck checks it at its latest commit on 3 November 2026, including where my README says more than the code. I am Yiguedan Philippe Andy Yaro.

A. Why orkestra Exists



One coding agent at a time is slow. Several at once in one folder collide: they edit the same files, say the tests pass, and could approve their own work.

orkestra separates powers: an agent proposes the plan; a kernel of plain Python, with no AI in it, checks the plan, runs the work and decides what lands.

The result waits on a side branch until I accept it.

1. What the README Promises, and What the Code Does

The README promises	How the code does it	What I found
Your branches are never modified	agents work in their own worktrees; only orkestra accept merges into your branch, after checks and a prompt that defaults to No	--yes skips the prompt; the director, Claude by default, runs in your project folder with permission to edit
Agents can't approve their own work	the policy engine refuses reviewer = implementer when the plan is checked and again at review	checked by agent name: two profiles of the same CLI count as two agents
"Tests pass" claims mean nothing	the kernel runs your [verify] commands and reads their exit codes; checks the plan proposes are filtered and only added	proving the gate reads the task's own files is on by default only in work not yet released
No surprise costs	token budgets per agent, rate-limit cooldowns, and caps on attempts and review cycles	token budgets are off unless set, only limit which agent implements, and ignore cached input
Crashes lose nothing	all state in SQLite; orkestra resume reconciles and carries on	an interrupted attempt's uncommitted edits are discarded and the task runs again; a crash before planning finished starts a new run
Secrets stay out of logs	23 redaction rules on events, gate records and reports	agent results and plans are stored in the database unredacted
Your credentials are untouched	agents sign in through their own CLIs	matches the code: no saved sign-in file is opened



CLI = command-line interface
worktree = a separate working copy on its own branch
SQLite = a database kept in one file
director = the agent that plans
gate, [verify] = the test commands that decide
redaction = replacing secrets with [REDACTED]
profile = one named agent setup

"What you can rely on", from my README, against the code at the latest commit. Each caveat has a code reference in the fact sheet behind this deck.

B. Architecture

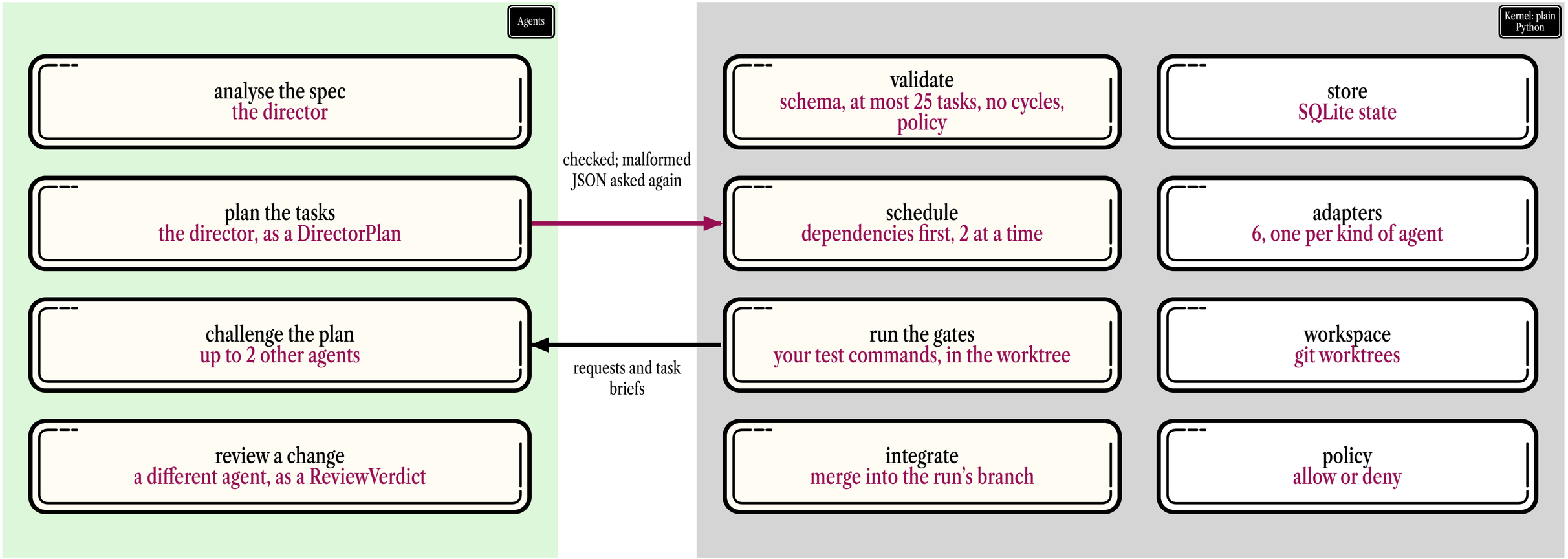


Intelligence proposes, determinism disposes. The kernel tests each agent with short probes; the director agent reads my spec and proposes a plan, as structured data checked against a fixed shape.

The kernel, plain Python, owns all state, checks every proposal against policy, runs the gates and merges the results.

About 12,500 lines of Python in 69 files, with four outside dependencies: Pydantic, Typer, Rich and tomlkit.

1. Who Proposes, Who Decides



If the director's reply stays malformed, or its plan fails the kernel's checks, a built-in planner makes a simple plan instead, and no event says so. With --offline, that planner always plans, and challenges are recorded as accepted without asking anyone.

JSON = structured text data
schema = the shape the data must have
spec = what to build, in my words
brief = the instructions for one task

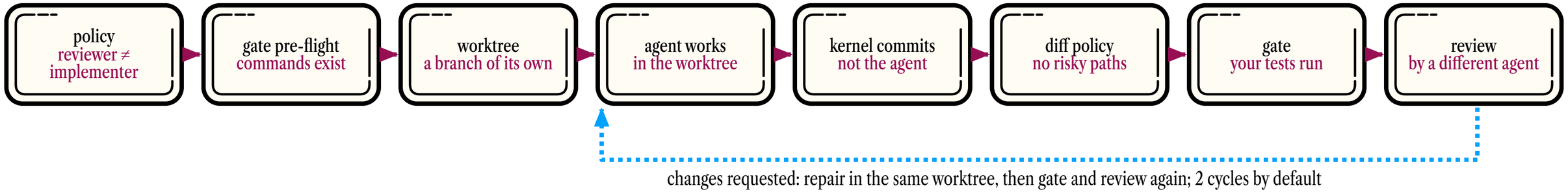
: Agents : Kernel : Proposals : Requests

2. A Run, End to End

Before any task runs



For each task, as soon as its dependencies are done





gate = the test commands that decide
diff = the list of changed files
probe = a short test of an agent
spec = what to build, in my words

Every step is recorded in the database, so a crash leaves a record of where the run stopped. The real names behind each step: prepare_run(), then Orchestrator.execute() and _run_task() in kernel/scheduler.py.

: Order of steps : Repair loop

3. Six Adapters, One Contract

Adapter	How it is called	Structured replies	Resume	Effort
claude-code	claude -p, streamed JSON	native, --json-schema	yes	via the model chosen
codex-cli	codex exec --json	native, --output-schema	yes	low, medium, high
antigravity-cli	agy -p, streamed JSON	schema in the prompt	yes	low, medium, high
gemini-cli	gemini -p, streamed JSON	schema in the prompt	yes	none
external	any program speaking orkestra's protocol	a structured field	yes	documented, not sent
fake	a scripted Python worker	scripted	yes	accepted, not sent

Every adapter runs the agent as a list of arguments, never through a shell, with an allowlist of environment variables. A timeout sends SIGTERM to the agent's whole process group, then SIGKILL after 5 seconds.

"Documented, not sent": the protocol describes an effort level for external agents, but the kernel never fills it in. Codex and Antigravity get effort through their own flags.

The Protocol for Anything Else

Detection: the program answers --orkestra-detect with one line of JSON.

Each task: a brief arrives on standard input; the program works in the worktree and must not commit or push.

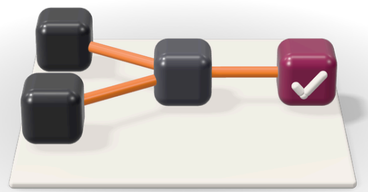
Replies, one JSON object a line: started, text, tool, and a final result.

A shared test kit checks any program that speaks the protocol: timeouts, garbage output, a missing result.



SIGTERM, SIGKILL = ask a process to stop; force it
process group = a program and its children
effort = how hard a model reasons
resume = continue the agent's last session
standard input = data piped into a program

C. The Scheduler

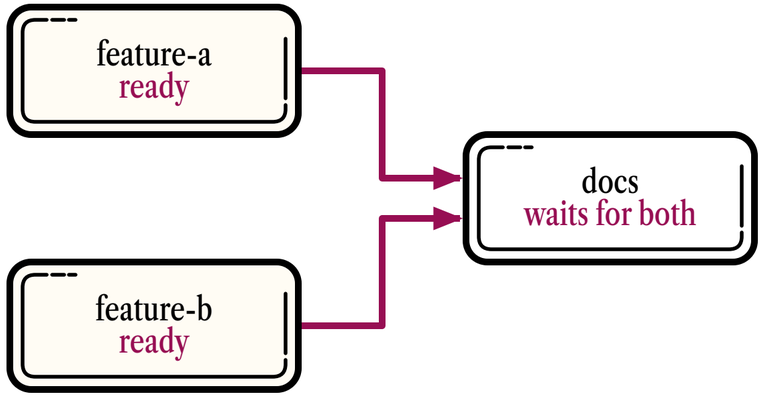


The kernel turns the plan into a graph of which task needs which, and starts every task whose dependencies are done, two at a time by default.

A standard method, Kahn's algorithm, rejects a plan whose tasks wait on each other in a loop; when nothing is left to run, the run stops instead of spinning.

Attempts, review cycles, timeouts and rate-limit pauses are all capped.

1. The Loop and Its Bounds



Build the graph: duplicate keys, unknown or self dependencies and cycles are refused. Cycles are found with Kahn's algorithm: peel off tasks with no dependencies; anything left over is in a cycle or waits on one.

Each loop: start every ready task, up to the concurrency limit.

Nothing ready and nothing running: wait for a human decision if one is open; otherwise the run fails, "no runnable tasks left".

Every Loop Is Bounded

Limit	Default
tasks in a plan	at most 25
agents at once	2 by default, 1 to 32
attempts per task	3 by default
review cycles	2 by default
time per attempt	30 min; hard stop 2 min later
rate limited	pause 60 s, doubling each time, up to 15 min
other errors	wait 5 s, doubling, up to 5 min; then the next agent
token budget	per agent, implementers only; off unless set

Two locks keep shared Git steps one at a time: merges into the integration branch, and adding or removing worktrees. The second fixed a timing bug that broke about 2.5% of tries.

dependency graph = which task needs which
concurrency = how many run at once

: Depends on

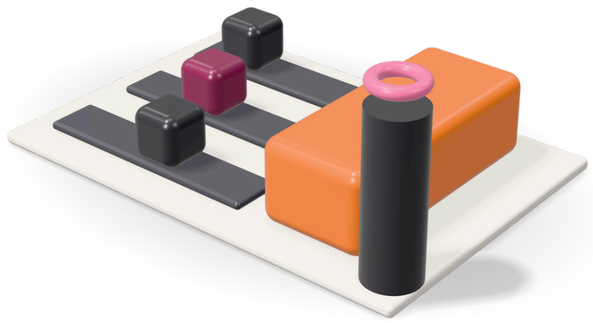
1. The Demo, Run

```
$ orkestra --version
orkestra 0.5.5
$ orkestra demo --path /tmp/ork/demo | grep -E 'review by|landed|requested changes \(cycle|tasks \
    planned|rejection triggered|never touched'
  completed review by grace: changes requested (severity medium)
    warning review requested changes (cycle 1)
  completed review by ada: approved (severity none)
  completed task feature-b done (agent grace), landed as 701ae9e4a696
  completed review by grace: approved (severity none)
  completed task feature-a done (agent ada), landed as 965b771dba19
  completed review by grace: approved (severity none)
  completed task docs done (agent ada), landed as 040d8e7ad6b9
  • 3 tasks planned, isolated, verified, cross-reviewed, and integrated
  • 1 review rejection triggered a repair loop (bounded - it can never spin forever)
  • every result was integrated commit-by-commit into a holding area; the demo's own 'main' was never touched
```

1. grace asks ada to change feature A: the demo's scripted reviewer rejects it once.
2. Feature B lands first; feature A lands only after the repair and a second review.
3. Docs depends on both features, so it starts and lands last.
4. The repair loop is capped: two review cycles by default.

Run on 3 November 2026 with orkestra from its latest commit (COLUMNS=140, NO_COLOR=1). The agents are scripted fakes, and the demo's three-task plan is written into the demo rather than made by a director; the scheduler, worktrees, gate, review and merges are the real kernel.

D. Git Isolation

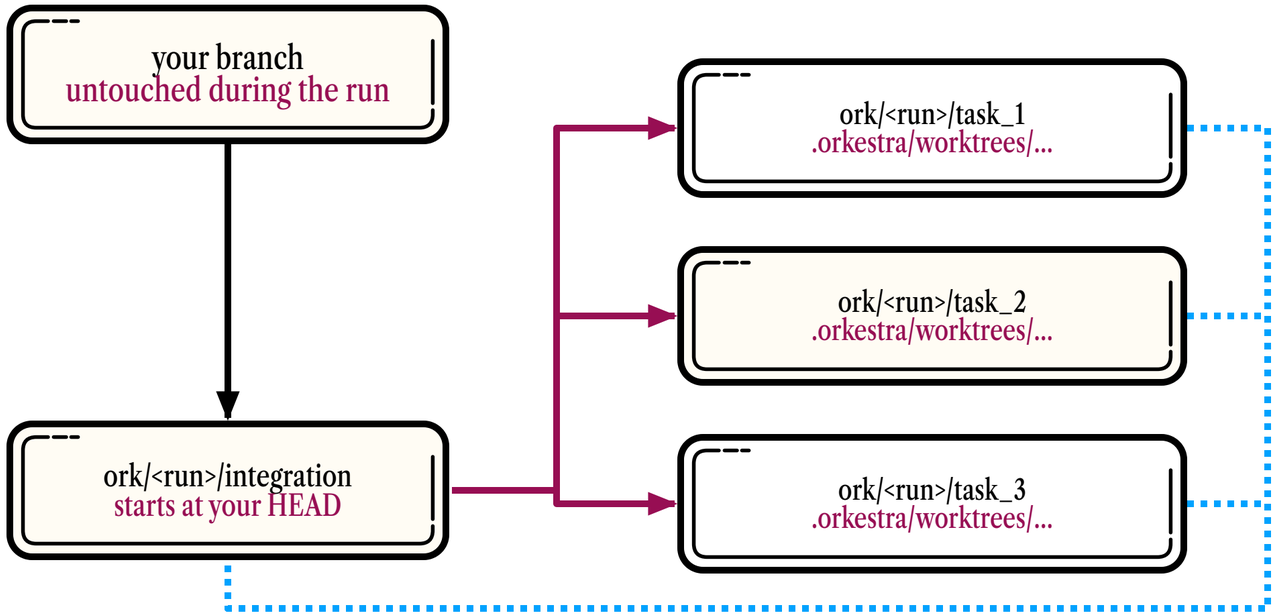


Every task gets its own Git worktree, a separate working copy on its own branch, cut from the run's integration branch.

Agents never commit: the kernel does, after the agent exits, and Git runs with its hook scripts and commit signing turned off.

Finished tasks merge into the integration branch; nothing reaches my branch until I accept it.

1. Worktrees, Branches and the Merge



How Git Is Called

A list of arguments, never a shell.

-c core.hooksPath=: no hooks run.

-c commit.gpgsign=false: no signing.

An allowlist of environment variables; no password prompts.

Every commit is authored “Orkestra”; an agent’s own commits name it in the message.

orkestra accept Refuses When

you are on an ork/ branch, or tracked files have uncommitted changes;

untracked files would be overwritten;

the run is not complete, unless you pass --allow-partial.

Its prompt defaults to No; --yes skips it.

A merge conflict: the task’s worktree is rebuilt from the new integration head and the task runs again.

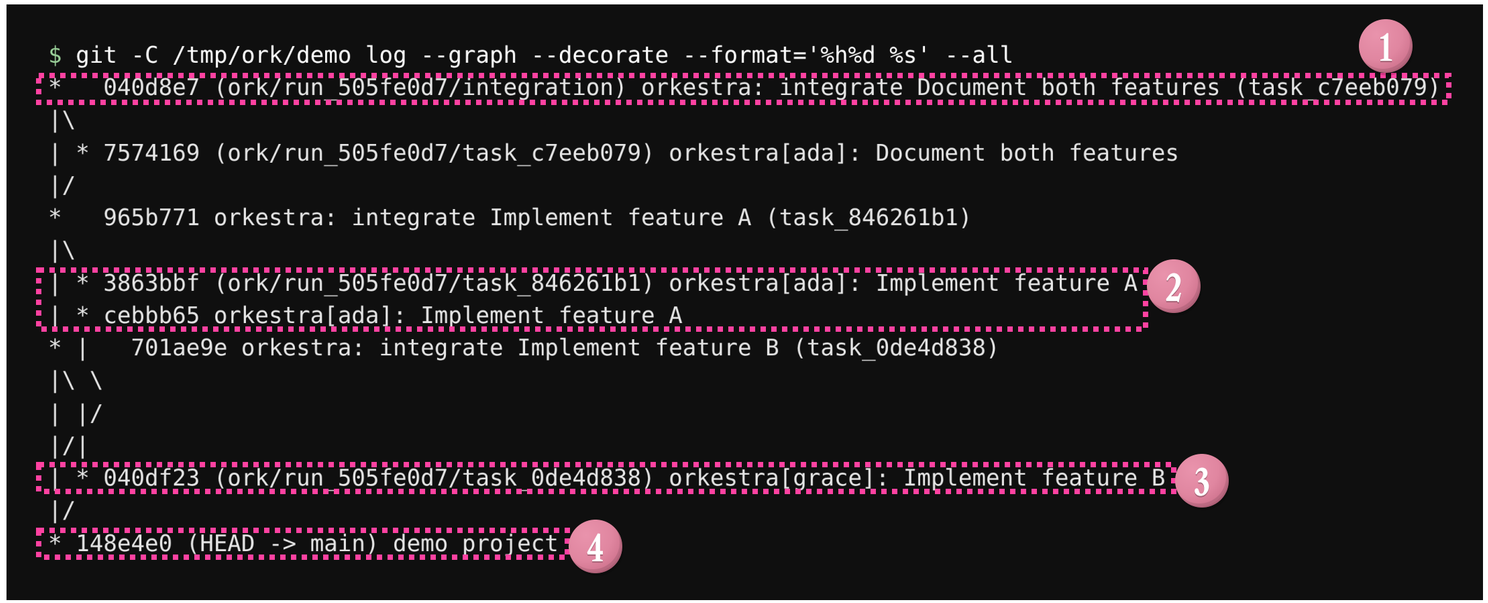
Before a task is marked done without a merge, Git is asked whether any attempt branch holds work that never landed; if so, the task stops for a decision.

No code anywhere pushes, hard-resets or force-pushes.

HEAD = your current commit
--no-ff = always make a merge commit

: Starts from : Branches from : Merges into

2. What the Demo Left in Git



The same run as the slide before, inspected with plain git right after it finished. Merging the integration branch into main is what orkestra accept would do.

1. The integration branch ends at the last of three merges: each finished task lands as its own merge commit.
2. Feature A has two commits: the first attempt and the repair asked for in review.
3. Each task has its own branch, named ork/<run>/<task> by the kernel, never by an agent.
4. main still points at the demo's first commit: the run never touched it.

E. Gates and Review

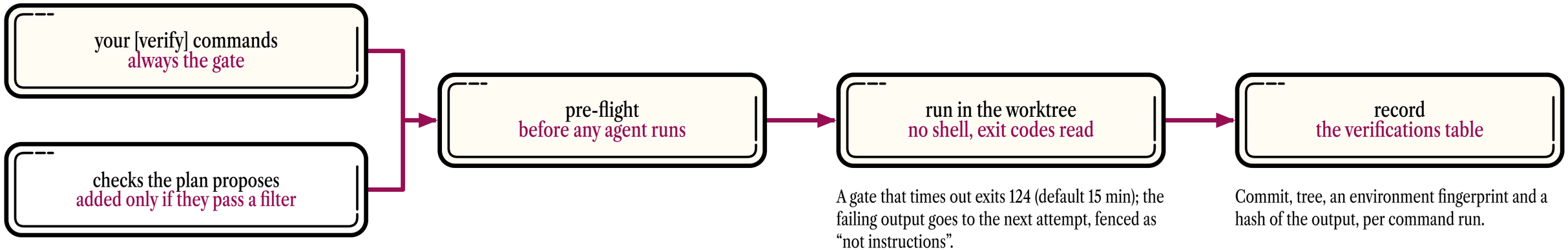


My test commands are the gate: the kernel runs them itself and reads their exit codes. An agent saying “tests pass” counts for nothing.

In the work after v0.5.5 it also checks that the gate reads the code: it breaks one file in a scratch copy and checks that the result changes.

Then a different agent reviews the change; a rejection sends it back for a repair, at most twice by default.

1. Gates You Can Trust



The filter: no shell characters such as | ; & $ or *, at most 160 characters and 12 words, and a program that exists. Anything else is dropped with a warning, never run.

A gate that times out exits 124 (default 15 min); the failing output goes to the next attempt, fenced as “not instructions”.

Commit, tree, an environment fingerprint and a hash of the output, per command run.

The Binding Canary

Once per run, in a scratch copy of the integration branch: run the gate; break one tracked source file; run it again. The exit code must change.

Changed: BOUND. Unchanged: UNBOUND: the gate does not read this code, and each task stops for a decision. Can’t tell: CANNOT-CHECK.

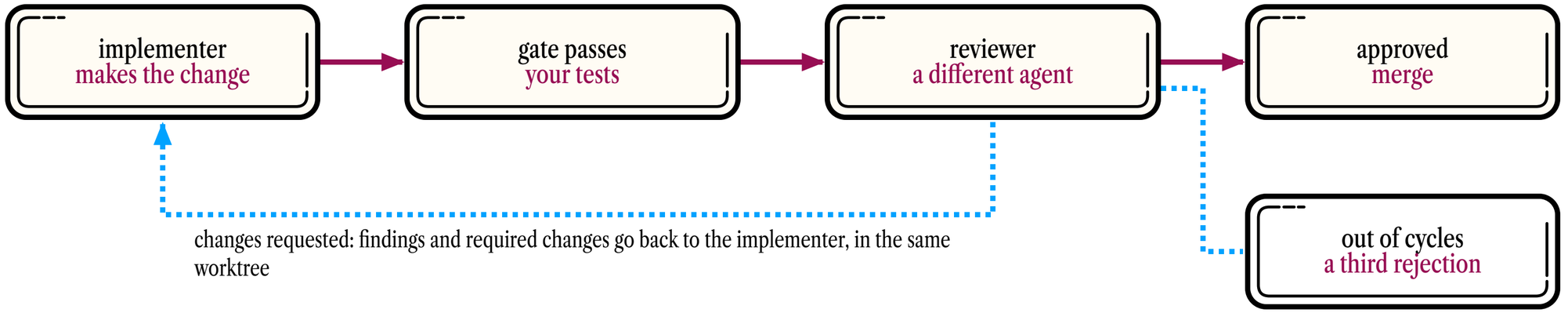
In the demo, the canary broke check_demo.py, the gate’s own script, and the exit code went from 0 to 1.

Off by default in v0.5.5, on in the work after it; the result is cached per set of commands and environment.

exit code = 0 means passed
canary = a planted fault that must be noticed
hash = a short fingerprint of data

: Order of checks

2. Review, Repair, Escalate



Tasks that change nothing skip review, and say so.

The Verdict, as a Schema

approve: true or false

findings and required_changes: lists

severity: none, low, medium, high or critical

A reply that does not fit counts as no verdict; the next reviewer is tried.

Who Reviews

The assigned reviewers first, then any other enabled agent except the one whose attempt is under review.

The list is tried twice; the first valid verdict decides.

No verdict at all: the task stops, “no independent reviewer could produce a verdict”.

When Budgets Run Out

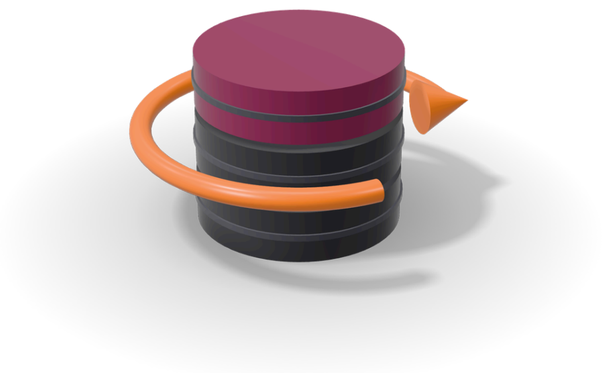
The director is asked whether to reassign the task.

If not, a decision waits for me: retry, skip or abort.

orkestra decisions explains it in plain words; approve and resume carry on.

: Happy path : Repair or escalate

F. State and Recovery



All state lives in one SQLite file, .orkestra/orkestra.db, written in short transactions.

Task state changes are checked: a move from the wrong state is refused, and repeating one is harmless.

After a crash, orkestra resume marks what was running as interrupted, tidies the worktrees and carries on.

1. One Database, and Resume

#	Added in	Migration
1	v0.1.0	10 tables: runs, tasks, task_deps, attempts, events, decisions, observations, ledger, workspaces, usage_log
2	v0.5.0	usage_log gains cached input tokens
3	v0.5.4	verifications: one row per gate command run
4	after v0.5.5	binding_proofs: the canary's cached results

Plus schema_version: 13 tables at the latest commit, 12 at v0.5.5. A database newer than the code is refused. Each write takes the lock at its start (BEGIN IMMEDIATE); the journal is WAL.

orkestra resume

Running attempts become interrupted; tasks mid-flight go back to ready.

Missing worktrees are marked lost; stale ones are removed, keeping their branches, so uncommitted edits are lost and the task runs again.

A run that crashed before planning finished is restarted as a new run.

```
$ python /home/claude/run_ork/tools/db_tables.py /tmp/ork/demo/.orkestra/orkestra.db
attempts           8 rows
binding_proofs     1 rows
decisions          0 rows
events            56 rows
ledger             7 rows
observations       7 rows
runs               1 rows
schema_version     1 rows
task_deps          2 rows
tasks              3 rows
usage_log          8 rows
verifications      7 rows
workspaces         3 rows
schema_version: 4
```

The demo's database: 8 attempts for 3 tasks, 4 of them reviews; 56 events; 7 gate records; one canary proof.

Only task moves are checked against the allowed transitions; run moves are not, though the code's own comment says both are.

There is no lock file: nothing stops two run or resume processes on one project; SQLite only takes turns on their writes.



migration = a step that updates the database layout
transaction = writes that land all at once or not at all
WAL = write-ahead log, crash-safe journaling

G. Security

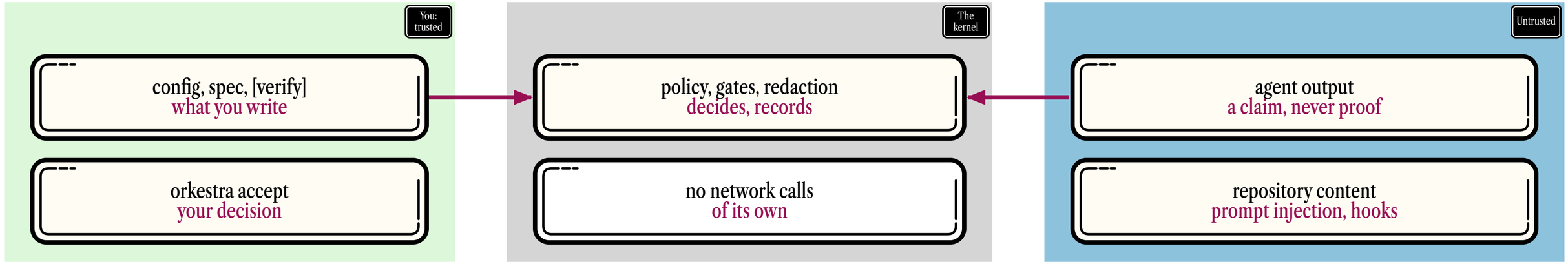


orkestra treats agent output and repository content as untrusted, and makes no network calls of its own.

A policy engine refuses self-review and risky paths; secrets are redacted before events and reports are written.

Where my documents promise more than the code, the last slide of this section says so.

1. Trust Boundaries and Controls



Agent	safe (the default)	unsafe-full (opt-in)
claude-code	accept edits	bypass permissions
codex-cli	sandbox: workspace write; read-only to plan or review	danger, full access
antigravity-cli	accept edits; plan mode to plan or review	skip permission checks
gemini-cli	auto edit; plan mode to plan or review	yolo: approve everything

Claude has no read-only mode here: in plan, challenge and review calls it still runs with “accept edits”, in your project folder when it directs or challenges.

prompt injection = text that tries to steer an agent
hooks = scripts Git runs on events
JWT = a signed web token
Docker = runs a program in an isolated container
unsafe-full = the opt-in mode that lifts limits

Other Controls

Agents get an allowlist of environment variables; no saved sign-in file is read.

External and fake agents can run in Docker: no network, all Linux privileges dropped, read-only, capped memory, CPU and processes.

23 redaction rules: API keys, GitHub, GitLab, AWS, Azure, Slack, npm and PyPI tokens, JWTs, passwords in URLs, private keys.

: Trusted : Kernel : Untrusted : Input

1. The Policy Engine, Fed Bad Input

```
$ python tools/policy_probe.py
assign claude, reviewers ['claude']: denied
  - reviewer 'claude' equals the implementer - independent review requires primary != reviewer
assign claude, reviewers []: denied
  - policy requires at least one independent reviewer
assign claude, reviewers ['codex']: allowed
diff paths: denied
  - absolute path in diff: /etc/passwd
  - path traversal in diff: ../outside.txt
  - diff touches .git internals: .git/hooks/pre-commit
  - diff touches protected path '.github/workflows': .github/workflows/ci.yml
push: pushing is disabled by policy (set policy.allow_push = true to opt in)
redact in:  GITHUB_TOKEN=ghp_AAAAAAAAAAAAAAAAAAAAAAAA
      out: GITHUB_TOKEN=[REDACTED]
redact in:  aws key AKIAIOSFODNN7EXAMPLE
      out: aws key [REDACTED]
redact in:  db: postgres://app:hunter2@db.internal/prod
      out: db: postgres://app:[REDACTED]@db.internal/prod
redact in:  Authorization: Bearer abc.def.ghi
      out: [REDACTED]
redact in:  tests: 557 collected
      out: tests: 557 collected
```

1 2 3 4

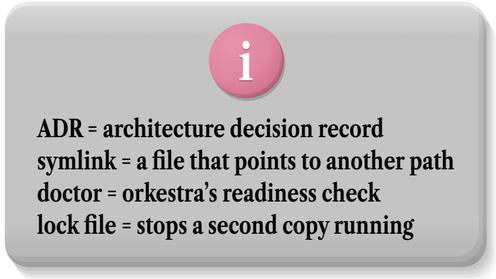
1. Self-review is refused, and so is a task with no reviewer; a different reviewer is allowed.
2. Absolute paths, .., Git internals and the protected workflow folder are refused in any diff.
3. Secrets become [REDACTED]: a GitHub token, an AWS key, a password inside a URL, a bearer token. All four are fake.
4. Ordinary text passes through unchanged.

tools/policy_probe.py, written for this deck, calls orkestra's own PolicyEngine and redact() with the default policy. Run on 3 November 2026. The push line is a policy answer only: orkestra has no code that pushes.

2. Where the Docs Say More Than the Code

What the docs say	Where	What the code does
unsafe-full is “logged loudly” and shown by doctor and the report	ARCHITECTURE, SECURITY_MODEL	nothing logs or shows it; it is never the default
Claude runs with --add-dir confinement	ARCHITECTURE	the adapter passes no --add-dir
a single kernel process holds an advisory lock file	ADR-0003	no lock file exists
run and task state changes are checked on every write	a code comment, schemas/states.py	task changes are; run changes are not
diff paths are resolved and symlinks handled	THREAT_MODEL, T2	paths are checked by name only; no symlink check
redaction on everything persisted or exported	README	events, gate records and reports yes; agent results and plans no
external agents receive the effort level	a code comment, schemas/effort.py	the kernel never sets it, so they get none
the director measures the agents with probes	README	the kernel runs the probes
timeouts kill whole process groups	SECURITY_MODEL	the gate’s timeout kills only its own process
about 15 tables	ARCHITECTURE, ADR-0003	13 at the latest commit

Two touch safety: SECURITY_MODEL says no secrets are stored in .orkestra, but agent results are kept unredacted; and a gate that times out can leave child processes running. The rest would mislead someone who trusts the docs.



H. Proof and Limits



Checked on 3 November 2026: the full test suite, lint and strict typing, the demo, and the policy engine fed bad input.

Real agents have run it only a handful of times, and the repository reports those runs, failures included.

The last slides set out that evidence, what to fix next, and what it shows and does not.

1. Tests, Lint and Types, Run Today

```
$ uv run --frozen ruff check .
All checks passed!                                                    1
$ uv run --frozen ruff format --check .
182 files already formatted
$ uv run --frozen mypy
Success: no issues found in 69 source files
$ uv run --frozen pytest --collect-only | grep '::' | cut -d/ -f2 | sort | uniq -c
     28 adapters                                                      2
    120 cli
     55 e2e
    334 unit
     20 workspace
$ uv run --frozen pytest -rs | tail -3
========================== short test summary info ==========================
SKIPPED [1] tests/e2e/test_docker_sandbox.py:60: docker daemon unavailable
556 passed, 1 skipped in 332.52s (0:05:32)                            3
```

Continuous Integration: 7 Checks

Check	What it runs
quality	format, lint, strict types, bandit, pip-audit
secrets-scan	gitleaks over the whole history
test × 4	Ubuntu and macOS, Python 3.12 and 3.13; coverage at least 80%
install-smoke	build, install, run the CLI

Releases go to PyPI by trusted publishing: GitHub proves the workflow's identity (OpenID Connect), so the workflow stores no API token. Its protected environment is a GitHub setting I did not check. Python 3.14 is advertised but not in the test matrix.

1. No lint or format problems; strict mypy finds nothing in 69 files.
2. 557 tests, most of them unit tests; 55 are end-to-end tests, most running the whole kernel on real Git repositories with fake agents.
3. 556 pass; the one skipped needs Docker, which this machine does not have.

Terms: ruff is a linter and formatter; mypy checks types; bandit and pip-audit look for security problems in code and dependencies; gitleaks looks for committed secrets; uv installs the locked dependencies; coverage is the share of code the tests run.

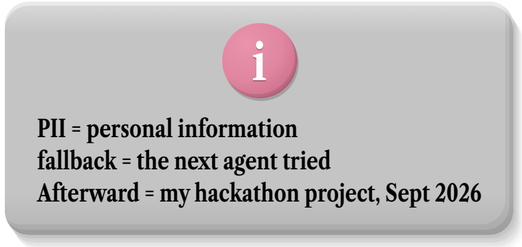
Run on 3 November 2026 from the repository's latest commit, with the locked dependencies. No test calls a real agent: the live test marker exists but no test uses it.

2. Real Agents, as Reported

Run	Version	Agents	What happened
Live smoke, 24 Jul	v0.1.0	Claude Code, Codex and Antigravity	Completed. The gate caught Antigravity not writing; Claude repaired it as the fallback; Codex reviewed. One human decision was needed.
Self-review, 24 Jul	v0.1.0	Codex and Claude ran; Antigravity was assigned too. On orkestra's own code	Ended failed, 1 of 2 tasks done; the review it did produce became the five fixes in v0.1.1.
Fleet tests 1 and 2	v0.4.1, v0.4.4	AI testers, mostly with fake agents	24 and 27 findings; each had one tester session with real Claude Code and Codex.
Fleet tests 3 and 4	v0.4.5, v0.5.0	three Claude Code profiles, named claude, codex and antigravity	49 and 35 findings, such as plan prose run as commands, and a 0.5.0 regression; $0.38 to $8.52 a tester session (fleet 3 did not yet count planning).
Afterward, 11 Sep	v0.5.5	Claude Code and Codex	Codex's review found two PII leaks; the run hit a 15-minute time limit set outside orkestra, and the last fix was finished by hand.

So: few real runs across vendors, all reported here: two in July on v0.1.0, one tester session in each of fleet tests 1 and 2, and Afterward in September. The "real agents" fleet tests used three Claude profiles, not three vendors.

From the reports in docs/development and feedback/; none of these runs was repeated for this deck. Fleet tests: several AI testers each play a user against a release.



3. How It Was Built and Shipped

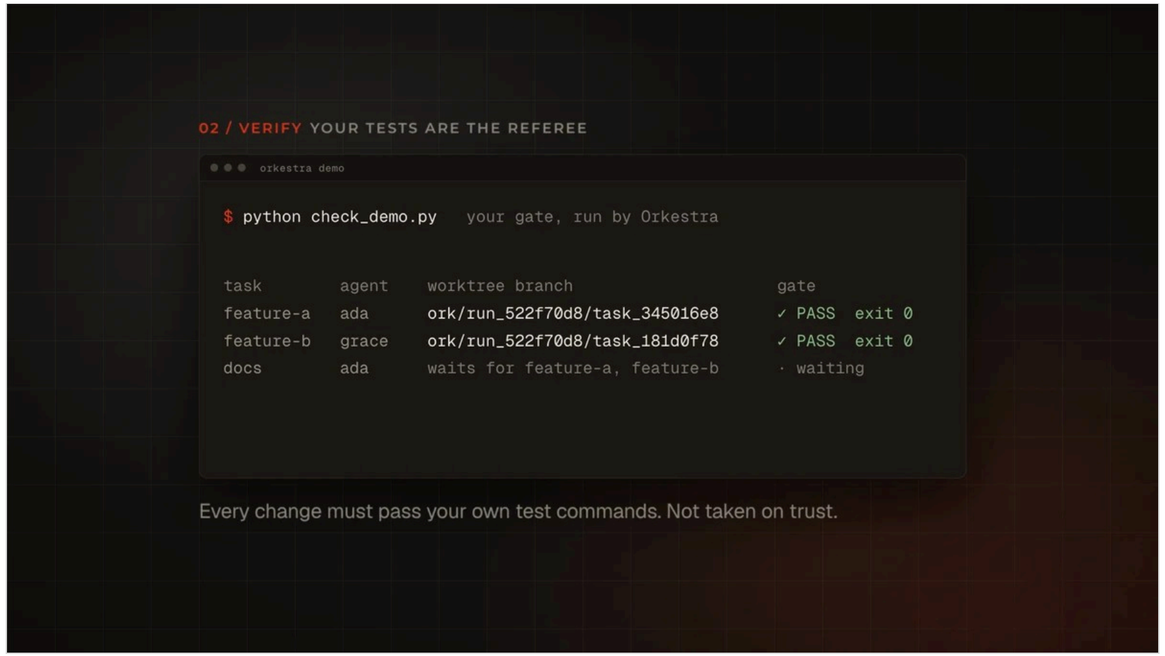
Timeline (New York time)

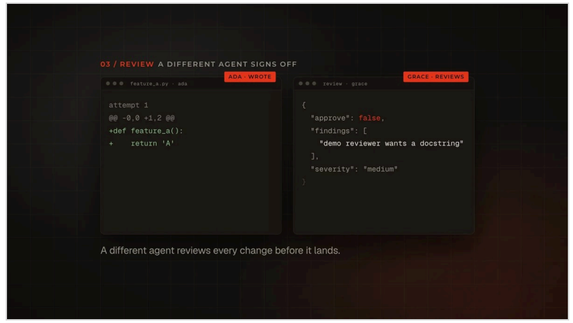
When	What
24 Jul, 14:56	first commit: a build prompt, a status file
24 Jul, 17:33	v0.1.0, two and a half hours later
26 Jul, 00:10	v0.5.3: 16 tags in about 31 hours
8 Sep	v0.5.4 and v0.5.5: the canary, a race fix, the lost-work guard
to 2 Nov	unreleased: canary on by default, the launch film

v0.1.0 was built by a coding agent working through MASTERPROMPT.md, my build prompt, phase by phase; the repository does not record how later releases were made. 126 commits, all under my name.

17 version tags; 15 versions on PyPI, from 0.1.2.

tag = a named version in Git





Two frames from the README's preview of the launch film: 62.5 seconds by its storyboard (the README says 63), drawn in HTML from a real demo run, with a fact sheet that sources each claim.

4. What to Fix Next

1. **Make the docs match**
   Logging of unsafe-full, --add-dir, the lock file, the table count: build them or drop the claims.

2. **A read-only director**
   Run Claude's plan and review calls without edit permission, or outside the project folder.

3. **Redact before storing**
   Agent results and plan payloads, not only events and reports.

4. **A process lock**
   So two runs cannot share one project, as ADR-0003 says.

5. **Check run states**
   Enforce the run transitions the way task transitions already are.

6. **Safer paths**
   Resolve paths and refuse symlinks in diffs, as the threat model says.

7. **Real-agent tests**
   Use the live marker; add Python 3.14 to CI.

8. **Tidy edges**
   Kill the gate's whole process group on timeout; cap director reassignments, which by my reading can repeat without limit.

In order of what each protects: trust in the docs first, then my checkout and secrets, then the state, then the edges.

5. What It Shows, and What It Does Not

It Shows

A kernel that bounds agents: own worktrees, independent review, gates it runs itself, a capped repair loop.

State that survives a crash, and a record of every gate run.

557 tests: 556 passing today, one skipped because no Docker was available; strict types; seven CI checks.

Docs that state their own limits, and reports that keep the failures.

It Does Not Show

Use at scale across vendors: a handful of reported runs, the last in September.

That the director cannot edit your checkout.

That every stored field is redacted.

Low overhead on small jobs: in fleet tests planning was 25–34% of the cost, and up to 45% of the time.

The kernel is the strong part, and anyone can check it with orkestra demo in a minute. The next work is making the docs, the director’s permissions and the stored data as strict as the kernel.