Governing an Agent Leadership Team: Guardrail, Kill-Switch, and Liveness Patterns from a One-Human Autonomous Organization

Justin Bartak

Preprint, 2026-09-03

Orbyt Labs (Purecraft LLC)
ORCID 0009-0005-2615-3624

Abstract

Half of the documented engineering failures at one company running an agent organization, 33 of 66 in a single-rater, agent-coded corpus, were failures of its own instruments: tests, guards, gates or metrics that could not fail, could not see, or measured a proxy. We operate the organization studied: twelve named seats, nine running a language model (`docs/org/roster.md`; `docs/org/departments.json`; `ls .github/workflows`, 2026-09-02), hold a reporting, proposing and publishing cadence under one accountable human. The corpus (`public/failure-corpus.json`, 2026-09-02) spans 2026-06-09 to 2026-09-02 with eleven undated items; 42 of 66 carry a named countermeasure, and verification gaps are the largest class at 33. The ledger (`public/autonomy-ledger.json`, 2026-09-02) records 65 seat runs over six weeks, 2026-07-21 to 2026-09-01, at 84.6 percent completion, six of the 55 completions being panel-written observer rows; the frozen 80.0 percent baseline shares its 30 rows, and the 4.6-point gap is within two runs' movement. Forty-five decisions went to an append-only log (`docs/org/decision-log.md`); the organization kill switch was pulled once, on 2026-08-20. The corpus has no agent-failure class and one rater assigned every class, so the share is exploratory rather than a comparison. The design law that survived is positive observation: a control is live only when its refusal has been observed, and a monitor that could not look reports NOT OBSERVED rather than clean. That verdict is printed; at the layer that escalates, a monitor that could not look and one that saw health are the same event. We contribute a three-tier guardrail model, positive-observation liveness, the published corpus, ten design principles, and a retrofit checklist.

Keywords: agent governance; multi-agent systems; autonomous organizations; guardrails; kill switch; liveness; failure corpus; LLM agents.

Contents

1 Introduction

The company studied in this paper is testing one question in public: can a team of language-model agents run a company. Its public page at `app/(site)/about/page.tsx` states the epistemic status in one sentence: "This is a question, not a claim." It divides the question into three rungs (products, a leadership team, a company) and marks the second "Answered in part." and the third "Open."

The founding charter framed the target with the same restraint. Under the heading "The goal, stated honestly", `docs/org/autonomous-org-plan.md` (dated 2026-07-13, status "LOCKED (v5.2)") reads: "Turn Orbyt into an operationally autonomous company with a human on the strategic and irreversible tiller." The next sentence names the cadence: "Agent-run departments research, guard, build, and report to the CEO on a weekly, quarterly, and yearly cadence." The sentence after that rejects the stronger goal: " 'The AI runs everything unsupervised' is neither reachable nor desirable, and designing for it is how you build something you rip out later." The governing principle that follows is the one every mechanism in this paper descends from: "Autonomy is earned per lane, by blast radius."

This paper is not an answer to the company's question. It is a report on the apparatus built to test it, and on what six weeks of that test recorded. Our thesis is that the interesting engineering in an agent leadership team is not the agents but the machinery that decides what a run is allowed to become. That machinery answers four questions: whether a run may start, what it may write, whether its output survives, and who may widen any of those answers. The record supports a second claim we did not expect: half of the failures the company documented were failures of its own instruments. The corpus has no class for agent failure, and Section 6, The empirical record, states what that share can and cannot compare. The design law that survived is positive observation: a control counts as live only when its refusal has been observed, and a monitor that could not look reports NOT OBSERVED rather than clean.

1.1 From agent as tool to agent as seat

Most production use of language models treats the model as a tool: a product calls it inside a request, owns the prompt, and consumes the output as data. The model has no persistent name, no scope that outlives the request, and no act it is forbidden to take, because the calling code takes every act.

The organization studied treats a model as a seat. A seat is a named agent with a charter (what it is for), a lane (which paths it may write), a cadence (when it runs), and a line (what it cannot cross without a human). The identity is a record, not a prompt. `docs/org/roster.md` reads: "a seat's name, once chosen, is recorded here and never changes silently." and gives the reason: "The roster is read by every run so the operators are consistent people, not anonymous functions."

The shift changes what can be measured and refused. A seat run can be halted before it starts, fenced while it works, graded when it finishes, discarded if its output fails a gate, and counted either way. The operating law in `docs/org/CLAUDE.md` is "the workflow commits, never the agent": the agent produces a candidate change, and the workflow decides what survives. Section 3, The Machine: architecture of a governed seat, gives the completion decision and the isolation sequence that law names.

1.2 The organization studied

Orbyt Labs (Purecraft LLC) is a software company with one human, and its agent organization was founded on 2026-07-13, when the charter above was locked. All counts in this paper were measured on 2026-09-02 on `main` at commit `424a58007` unless a different date is given. The organization runs twelve self-named seats, eleven officers and an Inspector General (`docs/org/roster.md`). Eleven of the twelve have a workflow, nine appear in the run ledger, and nine run a language model. Section 3, The Machine: architecture of a governed seat, names the seats, the manifests that generate their workflows, and the four structural exceptions.

The public description of the human-only set is short enough to quote whole. The FAQ on `app/(site)/orbyt-collective/how-it-works/page.tsx` answers the question "Does a human still decide things at Orbyt Labs?" with: "Every decision that matters. Merges to main, money, protected files, anything a customer sees, anything legal, and any widening of an agent's authority. The agents concentrate the founder's attention. They never replace the call." Of those six items, money is a valve class in interactive sessions and NOT MEASURED for seats; protected files and lane widening are held for seats by the fence and isolation and for interactive sessions by nothing (Section 4, Guardrails in tiers). Two are not enforced for seats. The shared action pushes every completed run to `main` (`packages/collective/actions/org-agent/action.yml` line 746, read 2026-09-02), and the content seat's keep-set includes `app/(site)` paths (`.github/workflows/autonomous-content.yml` lines 191 to 197). The build skip list, and the residual that a keep-set path outside it deploys production with no human step, are described in Section 4, Guardrails in tiers. The record holds 11 content-seat commits touching `app/(site)`, carrying 21 matching path entries (`git log --author='orbyt-content-bot' --format='COMMIT %H' --name-only -- 'app/(site)'`, 11 commit lines to 21 path lines, 2026-09-02). The sixth item, anything legal, is a lane rule with no mechanism this paper measured.

Two labels in the title and abstract are the organization's own. "Leadership team" is the public page's name for the seats (`app/(site)/leadership/page.tsx` line 74, read 2026-09-02: "The Orbyt Labs leadership team."). No seat holds a decision right, a budget, or an outcome. The registry calls the Chief of Staff "a staff role with no line authority" (`docs/org/departments.json`, read 2026-09-02). The authority matrix generated from that registry lists every autonomous lane as a report, a proposal, advice, a mechanical update, or, for the

marketing seat alone, "publish fused content" (`docs/org/authority-matrix.md`). Section 3, The Machine: architecture of a governed seat, reports what that table does and does not enforce. What is delegated, and what "operating motion" means here, is a reporting, proposing and, for one seat, publishing cadence. Every logged decision is the founder's.

The empirical window is short and we state it plainly. Between 2026-07-21 and 2026-09-01 the seats recorded 65 runs, of which 55 completed and 10 failed, and the ledger's discard counter sums to 47 (`public/autonomy-ledger.json`, measured 2026-09-02). A discard is a file the agent changed outside its keep-set and journal, counted and then thrown away when the tree is reset at isolation (`packages/collective/actions/org-agent/action.yml` lines 447 to 448, read 2026-09-02). No panel is consulted. That the discard counter counts files reset away at isolation, and that three public surfaces describe a panel that does not exist, is reported in Section 6, The empirical record. The one global halt, set and lifted on 2026-08-20, is told with its commits in the same section. The published completion rate is 84.6 percent, measured against a baseline of 30 runs frozen on 2026-08-11 at 80.0 percent and never regenerated. Over a longer window the company documented 66 failures in `docs/lessons-learned.md`, now published as `public/failure-corpus.json`, and 42 of the 66 (63.6 percent) carry a named countermeasure.

We founded and we operate the organization studied, and we wrote the documents quoted here. Section 8, Limitations, treats the bias this introduces.

1.3 Contributions

This paper makes five contributions. It continues our prior deposit, Agent-Native Dataset Design (Bartak, 2026, DOI 10.5281/zenodo.19754393), which described the company's data product; this paper describes the organization that operates it.

1. An inside account of a governed agent organization: twelve named seats, a registry, a shared run action, a completion decision with exit codes, and the human-only set, described as built, with paths and receipts.
2. A three-tier guardrail model (report-only sandbox, STRICT gate, human-only valve) and the autonomy line it implies: agents may create and improve report-only controls; only the human flips a gate; money never belongs to an agent. Merges to `main` and customer-facing content are not enforced for seats, a residual Section 4, Guardrails in tiers, states in full.
3. Positive-observation liveness: a fence that fails unless it watches a real denial, a canary that watches the fence, a dead-man switch that judges run conclusions rather than the ledger the watched system writes, and NOT OBSERVED as a printed verdict the human reads. At the layer that escalates, that verdict exits 0 and is indistinguishable from clean, an open item Sections 4 and 5 name.
4. A published failure corpus of 66 documented failures with a three-class taxonomy (33 verification gaps, 18 operator-process failures, 15 external-behavior failures), 42 of the 66 (63.6 percent) carrying a named countermeasure, and the finding that verification gaps are the largest class. Section 6, The empirical record, gives the class table; Appendix D records the four kinds of countermeasure and the rule that sorts a guard from a test.
5. Ten design principles, eight traced to a motivating incident and two, P5 and P6, adopted as design decisions, each with the file that carries it, which is not enforcement; Section 7 grades all ten, and names the two whose enforcement is missing. It adds a retrofit checklist for other agent organizations.

1.4 Organization of the paper

This section has stated the question, the thesis and the contributions, and the remaining eight sections deliver them. Section 2, Related work, places the apparatus against five literatures and two policy instruments. Section 3, The Machine: architecture of a governed seat, describes the seat as built, from the registry to the commit. Section 4, Guardrails in tiers, gives the three tiers, the commit gate, the fence and the kill switch. Section 5, Liveness and the watch chain, follows the watchers and names what each layer below cannot report. Section 6, The empirical record, reports the record with its contradictions. Section 7, Design principles and a retrofit checklist, distils the principles and the checklist. Section 8, Limitations, states what the record cannot support, and Section 9, Future work, names what we would measure next. Appendix A is the guard census and Appendix B the org lessons. Appendix C holds the vocabulary and the commands that reproduce every number in Section 6, and Appendix D inventories the mechanisms this paper names in passing.

2 Related work

This paper draws on five literatures that rarely cite one another, and on two policy instruments. Multi-agent frameworks supply the vocabulary of roles and turns. Agent governance specifies what an operator should make visible. Safety engineering supplies the theory of why controls fail. Reliability engineering supplies the practice of alerting on running systems. Test adequacy supplies the standard for what a passing check proves.

2.1 Multi-agent frameworks

CAMEL began the line with two agents in complementary roles cooperating through inception prompting (Li et al., 2023). Generative Agents gave simulated residents memory, reflection and a town (Park et al., 2023). MetaGPT encoded standard operating procedures into prompt sequences and assigned the roles of a software company (Hong et al., 2023), and ChatDev framed software development as a chain of chats (Qian et al., 2023). AutoGen generalised the pattern into conversable agents composed with tools and a human in the loop (Wu et al., 2023). Magentic-One added an orchestrator that keeps a task ledger and a progress ledger (Fourney et al., 2024).

Beneath these sit single-agent primitives: ReAct interleaves reasoning traces with actions (Yao et al., 2022), and Toolformer trains a model to decide which API to call and when (Schick et al., 2023). The Model Context Protocol standardises how tools and context reach a model over JSON-RPC (Model Context Protocol, 2025). Anthropic's engineering guidance recommends a fixed workflow over an autonomous loop wherever one suffices (Anthropic, 2024).

What these systems govern is the conversation: who speaks, in what role, in what order, and when the loop terminates. What they leave ungoverned is everything after the last turn: which files a run may commit, who performs the commit, and what happens to a run that ends without its final line. In the organization studied the governed unit is the run and its commit, not the dialogue. The operating law in `docs/org/CLAUDE.md` reads: "the workflow commits, never the agent". The protocol layer draws the same line. The specification states that "MCP itself cannot enforce these security principles at the protocol level", placing consent, tool safety and access control with the host (Model Context Protocol, 2025). That host is where the valve and the fence of Section 4, Guardrails in tiers, live.

Two lines of work are closer to the run-and-commit unit. The AI control line asks how to get useful work from a model that may be trying to subvert its deployment, and answers with a trusted gate over untrusted output (Greenblatt et al., 2023). The completion decision and the

keep-set of Section 3, The Machine: architecture of a governed seat, are that shape without the adversarial premise: the gate is deterministic, and it discards rather than edits. SWE-agent (Yang et al., 2024) and OpenHands (Wang et al., 2024) confine execution and land work as a reviewed change. What both leave open is the record kept when the same confinement runs unattended for weeks: a run ledger, a decision log and a failure corpus.

The closest empirical neighbour is the failure taxonomy of Cemri et al. (2025). From annotated traces across seven frameworks, they sorted failures into three classes: system design, inter-agent misalignment, and task verification. Their third class is the nearest published analogue to the verification-gap class that dominates our corpus (Section 6, The empirical record). Their unit is a trace of a benchmark task. Ours is an organization that has to run again next week.

2.2 Agent governance and visibility

A second literature asks what an operator of agents owes those around them. Chan et al. (2023) argued that harms grow with the degree of agency in a system, so governance should attend to agency before deployment. Chan et al. (2024) proposed three measures of visibility: agent identifiers, real-time monitoring, and activity logs. Chan et al. (2025) extended the argument to infrastructure external to any one agent. Kolt (2025) read the problem through principal-agent theory and agency law, naming information asymmetry, discretionary authority and loyalty as the problems AI agents inherit.

OpenAI's white paper offered an initial set of practices for keeping agent operations safe and accountable (Shavit et al., 2023). Its section headings name them: "Evaluating Suitability for the Task", "Constraining the Action-Space and Requiring Approval", "Setting Agents' Default Behaviors", "Legibility of Agent Activity", "Automatic Monitoring", "Attributability", and "Interruptibility and Maintaining Control". Each has a counterpart in the apparatus of Sections 3 to 5: the lane and the fence (action space), the human-only set (approval), per-run stats rows (legibility), the pulse (monitoring), HALT (interruptibility). The mapping was drawn after the fact, and we do not claim the practices were followed by design.

Attributability rests on an organization-domain identity carried by 737 of the 4,852 commits in the repository on 2026-09-02, of which 565 are the push-collision canary and 87 map to a seat (Section 6, The empirical record). Two identity leaks in the same listing are why `scripts/check-commit-author.cjs` now enforces it at commit time (Section 6, The empirical record). Attribution is a convention applied going forward, not a property the history already has. Against that list the organization adds one thing and claims no more: a third verdict state under automatic monitoring, NOT OBSERVED. The pulse's three blind branches, and the permission scopes added after each, are told in Section 5, Liveness and the watch chain.

Mitchell et al. (2025) delineated levels of agent autonomy and argued against fully autonomous agents, because risks to people rise with autonomy. The organization studied agrees at the level of its human-only set: money never belongs to an agent. Merges to `main` and customer-facing content are not enforced for seats, because the shared action pushes every completed run. The build skip list, and the residual that a keep-set path outside it deploys production with no human step, are described in Section 4, Guardrails in tiers. Casper et al. (2025) built an index of deployed agentic systems and found that developers disclose capabilities generously and safety practices sparsely. This paper is one deployer's response.

Kapoor et al. (2024) showed that agent benchmarks optimise accuracy alone, ignore cost, and overfit without holdouts. Dong et al. (2024) proposed a taxonomy of the artifacts an observability layer should capture across an agent's lifecycle. Clymer et al. (2024) described safety cases as structured arguments that a system is safe enough to deploy. All three concern the agent. This paper reports a within-corpus share instead: 33 of 66 documented failures, each classed by

a single rater, our agent, were failures of the instruments around the agent (`public/failure-corpus.json`, 2026-09-02). The corpus has no agent-failure class, so it supports no comparison on a common denominator (Section 8, Limitations). That share is why Section 5, Liveness and the watch chain, watches the watcher.

2.3 Safety engineering lineage

The theory of why controls fail is older than language models. Perrow (1999) argued that interactive complexity and tight coupling make some accidents normal, and that each added safety device adds interactions. Reason (1990) separated active errors at the point of operation from latent errors that sit dormant until a local trigger exposes them. Leveson (2012) recast safety as a control problem: accidents occur when the control structure fails to enforce its constraints. Amodei et al. (2016) named scalable supervision, the oversight of a system whose evaluation is expensive, as an open problem. Bai et al. (2022) answered at the model level with a written constitution; the organization studied keeps a constitution too (`docs/org/constitution.md`) and enforces it with hooks rather than weights.

Functional-safety practice separates diagnostic coverage, a control's own self-test detecting a fault during operation, from proof testing, an operator provoking the fault at a stated interval because normal operation would not reveal it. Positive observation, the design law of Section 5, Liveness and the watch chain, is the first kind. It applies only to a control whose refusal can be provoked inside the run it protects, and in the organization studied that is the fence canary alone. The valve, the deny list, the HALT re-ask, the heartbeat channels, the turn counter and the stop gate can only be proof-tested, by a logged drill at a stated interval. On 2026-09-02 none of them had one (Section 4, Guardrails in tiers; Section 8, Limitations).

Bainbridge (1983) opened with two ironies, and both apply. Automating a process to remove the operator's errors leaves the designer's own errors in place, and the corpus's 33 of 66 instrument failures are that irony in numbers. The operator is then left monitoring a system that rarely fails, so vigilance decays. The human must intervene at exactly the moment intervention is hardest. The organization studied is a single human and 24 scheduled workflows among 32 workflow files (Section 6, The empirical record).

The design response follows Leveson's framing: do not ask the human to watch the process, but require the controls to prove they watched. The v1 specification in `docs/collective/spec.md` applies that law to the fence in its section 2.3, and its section 2.2 records the limit of the claim (Appendix D carries both rules verbatim). That the sandbox's network settings were measured doing nothing on a hosted runner, off-allowlist hosts answering 200 while the same fence denied every write, is recorded in Section 4, Guardrails in tiers. Refusing to declare a block that does not exist in the world turns Reason's latent-error argument into an executable check. What this paper adds to the lineage is the schedule: the difference between a safety system installed and one exercised is settled inside every run, not by an entry on a calendar.

2.4 Reliability engineering and observability

The chapter on monitoring in Beyer et al. (2016) opens with the question a monitor must answer: "Your monitoring system should address two questions: what's broken, and why?" It then states the two rules the pulse inherits: "Every page should be actionable." and "it's better to spend much more effort on catching symptoms than causes". The daily pulse pages on symptoms, such as a seat that did not complete inside its window. It adds a state the SRE chapter does not have: a monitor that could not look says NOT OBSERVED, which counts as a blind axis rather than health.

Chaos engineering injects failure into a production system on purpose (Basiri et al., 2016). The

organization applies the idea to one control at a time. The fence canary attempts an out-of-lane write inside every seat run, and the run fails unless the denial was observed (Section 4, Guardrails in tiers).

The dead-man switch is older than software: a control that must be actively held, and that stops the machine when released. The organization's version is the daily liveness job in `packages/collective/src/check-org-liveness.mjs`, whose header and first wrong version Section 5, Liveness and the watch chain, records. Sculley et al. (2015) catalogued the technical debt that accumulates around learned systems, including monitoring and configuration debt; lessons 29 and 62 in `docs/lessons-learned.md` are instances.

2.5 Test adequacy

Mutation testing states the standard this paper leans on. DeMillo, Lipton and Sayward (1978) proposed seeding small faults into a program and asking whether the test data detects them; a test set that kills no mutants has demonstrated nothing. The organization reached the same rule from its own incidents. `CLAUDE.md` states it as "a guard is not owned until you have seen it fail" and gives the reason: "A green guard you have never made go red is decoration."

The seven errors of 2026-07-23, the four detectors that mis-fired that day and the three rules they produced are told in Section 4, Guardrails in tiers. Lesson 63 records a guard that read the wall clock and stayed green with its fix deleted; lesson 26 records the mutation-testing tool itself failing to start for 81 days (`docs/lessons-learned.md`). Of the 66 documented failures, 42 (63.6 percent) carried a named countermeasure on 2026-09-02: 13 guard scripts, 22 tests, 1 workflow and 6 implementation files (`public/failure-corpus.json`). A `scripts/check-*` or `packages/collective/src/check-*` path counts as a guard, and any `__tests__` path counts as a test.

2.6 Policy frameworks

The NIST AI Risk Management Framework organises risk management into four functions, GOVERN, MAP, MEASURE and MANAGE (National Institute of Standards and Technology, 2023). The organization's decision log, owner-directive register, guard census and ladder map onto the four in that order, a mapping drawn after the fact with no conformance claimed. Regulation (EU) 2024/1689, the Artificial Intelligence Act, sets requirements for high-risk systems (European Parliament and Council, 2024). They include Article 12, "Record-keeping", and Article 14, "Human oversight". Article 14(1) requires that such systems "can be effectively overseen by natural persons during the period in which they are in use". We do not claim the organization falls within the Act's scope. We read Article 14 as a design target for one natural person overseeing twelve seats (`docs/org/roster.md`).

Green (2022) argued that policies requiring human oversight of algorithms assume a capacity the overseeing humans often lack. Oversight then becomes a formality that legitimises the system, which is Bainbridge's irony as a policy critique. The organization's answer is to move the human from overseeing runs to overseeing instruments, and to require the instruments to report when they could not see. Whether that holds beyond one organization is a question Section 8, Limitations, leaves open.

2.7 The gap

None of the works reviewed here is an inside account of a running agent organization with its records published. The frameworks report benchmark results. The control and sandboxing lines describe a trusted gate and a confined runtime, evaluated on tasks; neither publishes the run ledger, decision log or failure record of an organization that kept running. The governance

papers specify what an operator should disclose, and the AI Agent Index finds that operators mostly do not (Casper et al., 2025). The failure taxonomy of Cemri et al. (2025) works from benchmark traces, not from an organization's own record. The safety and reliability literatures were written for systems whose operators are many; this paper reports from an organization with one.

Our first paper covered the dataset side of the same company: how a dataset should be shaped, licensed and distributed so that retrieval agents can find and cite it (Bartak, 2026). This paper covers the organization that produces and maintains it. Its evidence is the record that organization keeps about itself: the seat ledger, the decision log and the failure corpus, measured on 2026-09-02 at commit `424a58007` (Section 6, The empirical record). Its central finding, the within-corpus share of instrument failures, is one that neither a trace-level study nor an outside index could have produced.

3 The Machine: architecture of a governed seat

This section describes the apparatus as it runs at Orbyt Labs, read from the repository on 2026-09-02 at commit `424a58007`. Every rule quoted is verbatim from the file named.

3.1 Twelve seats and a self-naming run

The organization has twelve named agent seats, eleven officers and one Inspector General. Nine run a language model, two as observer seats through the advisory panel. The Inspector General and finance seats are deterministic scripts. The security seat, registered `wound_up` and `dormant`, has no workflow (`docs/org/departments.json`; `ls .github/workflows`, 2026-09-02).

The names were not assigned. On the founding run of 2026-07-13 each seat chose its own identity, name, gender and persona in isolation, under one constraint: "Every name begins with 'Agent' so anyone can tell at a glance these are AI (the founder's transparency rule)." (`docs/org/roster.md`). Given total freedom the agents converged, and the founder kept that outcome (Appendix D). One rule makes the roster a record rather than a label: "Rule: a seat's name, once chosen, is recorded here and never changes silently."

Table 1: The twelve seats as the roster records them, with the date each name first appears. Read from `docs/org/roster.md` on 2026-09-02.

Seat	Name	Pronoun recorded	Department	First recorded
CMO	Agent Cite	he	Marketing	2026-07-13
CFO	Agent Ledger	he	Finance	2026-07-13
CTO	Agent Ward	he	Engineering	2026-07-13
CPO	Agent Compass	he	Product	2026-07-13
CInO	Agent Cusp	he	Innovation	2026-07-13
COO	Agent Keeper	he	Operations (observer)	2026-07-13
CRO	Agent Convert	he	Revenue (observer)	2026-07-13
Chief of Staff	Agent Fulcrum	he	attached to the CEO	2026-07-13
General Counsel	Agent Bastion	he	Legal	2026-07-14

Seat	Name	Pronoun recorded	Department	First recorded
Inspector General	Agent Gauge	none recorded (the roster cell reads (v0))	independent	2026-07-15
CSSO	Agent Sentinel	he	Security, Safety, and Reliability	2026-07-16
CQO	Agent Assay	they	Quality	2026-07-20

The seats are written down in four places, kept separate and joined by guards: the roster, the registry, the eight manifests under `docs/org/seats/`, and the public mirror `app/(site)/leadership/officers.ts`, with permanent ids across all four (Appendix D). The memory guard once carried "one of three hardcoded copies of the same roster", which drifted to "13 entries for 12 seats" until 2026-08-16 (`packages/collective/src/check-org-memory.mjs`); it now derives the roster from the registry.

The registry still describes the Inspector General as halted, a lag reported in Section 6, The empirical record. That seat's halt was cleared on 2026-08-14 by the ladder rather than a person, and it has reported weekly since.

3.2 The registry and the seat as data

The registry is the single source of truth for what departments exist. On 2026-09-02 its `departments` array held eleven entries and its `officers` array two, with the Inspector General in both, where the department entry is authoritative. Two departments carry `"mode": "observer"` and one `"mode": "dormant"` (`docs/org/departments.json`, 2026-09-02).

A seat is data. D-0040 of 2026-08-16 turned that layer the right way round after it shipped on 2026-08-14 with the direction running backwards: "The spec is the source; the workflow is its artifact." (`docs/org/decision-log.md`). `packages/collective/src/generate-seat-workflow.mjs` renders `docs/org/seats/<id>.json` into `.github/workflows/<seat>.yml`. The comparison is byte-for-byte, because "Bytes cannot be argued with." Prose in a manifest is held as exact lines: "PROSE IS DATA, stored as exact lines and never re-wrapped."

Table 2: The keys of one seat manifest, `docs/org/seats/legal.json`, read 2026-09-02; the schema is section 3.3 of `docs/collective/spec.md`. The `failure` and `timeout_job` objects also carry the issue body text and its commentary, omitted here.

Key	Value in `legal.json`
`id`, `workflow`, `name`, `job`	`legal`, `.github/workflows/legal-weekly.yml`, `Legal Weekly`, `legal`
`header`	nine exact comment lines
`schedule.cron`, `schedule.comment`	`0 9 * * 3`, "Wednesday 09:00 UTC, before the Product run and the Friday CoS synthesis"
`max_turns_default`, `timeout_minutes`	80, 45
`fence.lane_paths`	`docs/org/reports/legal`, `docs/books/unfair-advantage/LOG.md`, `docs/org/lessons/proposed/legal.md`
`agent.prompt_file`	`docs/org/prompts/general-counsel-prompt.md`

Key	Value in `legal.json`
`agent.keep_paths`	`docs/org/reports/legal`
`agent.commit_message`, `agent.bot_name`, `agent.department_id`	`docs(legal): weekly legal-risk review + flags [legal-bot]`, `orbyt-legal-bot`, `legal`
`agent.eval_output`, `agent.model`	`docs/org/reports/legal/weekly-latest.md`, `sonnet`
`agent.sentinel_regex`, `agent.skip_regex`	an alternation of `GC_REVIEW:` and `GC_SKIPPED` (quoted verbatim in the safety-floor subsection below), `GC_SKIPPED`
`failure.issue_title`, `timeout_job.issue_title`	`Legal weekly run failed`, `Legal weekly run was killed by its timeout`
`persist_credentials_note`	"the agent step must NOT inherit a push credential"

`packages/collective/src/derive-seat-spec.mjs` read the eight existing workflows back into their first specs, because "transcription is where byte-fidelity dies". Pre-commit runs the generator twice, `--heal` then `--check` (`scripts/pre-commit`, lines 407 and 408), because a healer trusted on its own report is an effect claimed rather than observed. Appendix D records what each refuses to guess.

3.3 One shared action and four structural exceptions

Seven seats run, through eight manifests, on one composite action, `packages/collective/actions/org-agent/action.yml`, "The ONE shared mechanism every autonomous-org department runs on". It declares seventeen inputs, and eight workflow files call it at nine call sites, because `marketing-reports.yml` runs two agent steps (Appendix D).

The credentials are split by step. The caller checks out with `persist-credentials: false`, so the agent step holds no push credential and the sanctioned push happens from the action with a just-in-time token. The agent step receives `secrets.CLAUDE_CODE_OAUTH_TOKEN`, a subscription token rather than a metered API key, and the push and stats steps receive `secrets.GITHUB_TOKEN`. Each input's description names the steps it may not reach (`action.yml`, lines 24 and 27). The agent step therefore cannot read a push credential out of `.git/config`. The fence runs before the kill switch: `org-fence` installs the sandbox and probes it with a `claude -p` canary (`org-fence/action.yml`, line 255), while the HALT preflight sits in `org-agent`, the next step (lines 103 to 124), so a halted seat has already spent one probe invocation.

Four workflows are not on the shared action, and the org memory calls them "STRUCTURAL exceptions, verified by grep, not judgement calls" (`docs/org/CLAUDE.md`). The advisory panel runs two observer seats in one job, which the one-run-per-call action cannot express; the Inspector General run is deterministic; the finance ledger is dependency-free. Appendix D holds those three constraints verbatim. The outside-family workflow is "Event-driven only, never scheduled" (`.github/workflows/outside-check.yml`, line 3), though its checker has one sanctioned scheduled caller in `inspector-general-weekly.yml` (line 160). The checker asks "ONE different model family (OpenAI's current flagship)" (lines 9 to 10 of `packages/collective/src/outside-checker.mjs`) under `const MONTHLY_BUDGET_USD = 10;`, which "Fail CLOSED on spend", and is the organization's one metered API use. The fallback family is xAI, tried when the OpenAI call is unavailable and never on a skipped budget, because "a cap with a fallback is not a cap" (lines 82 to 86).

Every manifest seat pins `"model": "sonnet"`, in all nine agent blocks across the eight files (Appendix D). The cadences are cron lines, all UTC.

Table 3: Cadence of the twelve org workflows, eight manifest-driven and four structural exceptions. They cover eleven of the twelve seats: the marketing seat carries two, the advisory panel convenes the two observer seats, the outside check belongs to no seat, and the security seat has none. Read with `grep -nE 'cron:'` over the named files in `.github/workflows/` on 2026-09-02. The Chief of Staff line's comment (`.github/workflows/chief-of-staff-weekly.yml`, line 17) records a sprint cadence set by owner directive on 2026-08-16, to revert to Friday when the sprint closes, and notes that "*/2 fires on odd days, so the 31st to 1st gap is one day, not two".

Workflow	Cron line	Reads as (UTC)
`autonomous-content.yml`	`0 9 * * 2`; `0 16 * * 4`; `0 16 * * 6`	Tue 09:00, Thu 16:00, Sat 16:00
`chief-of-staff-weekly.yml`	`0 13 */2 * *`	odd-numbered days of the month at 13:00 UTC (the 31st to 1st gap is one day); sprint cadence since 2026-08-16
`legal-weekly.yml`	`0 9 * * 3`	Wed 09:00
`product-weekly.yml`	`0 12 * * 3`	Wed 12:00
`engineering-weekly.yml`	`0 9 * * 4`	Thu 09:00
`quality-weekly.yml`	`0 15 * * 4`	Thu 15:00
`innovation-monthly.yml`	`0 17 1 * *`	1st of the month, 17:00
`marketing-reports.yml`	`0 10 1 1,4,7,10 *`	quarterly, 1st of Jan, Apr, Jul, Oct, 10:00
`advisory-panel.yml`	`0 16 2 * *`	2nd of the month, 16:00
`inspector-general-weekly.yml`	`0 14 * * 6`	Sat 14:00
`finance-ledger.yml`	`0 6 * * 3`	Wed 06:00
`outside-check.yml`	none	`workflow_dispatch` only

The apparatus assumes a fallible agent, not an adversarial one, and nothing enforces that assumption. Six of the 65 ledger rows carry `external_input: true`, three innovation runs and three marketing runs between 2026-08-01 and 2026-09-01 that read content from outside the repository (`docs/org/stats/*.jsonl`, 2026-09-02; Appendix D). No guard reads that flag. Such a run has egress unfenced (`packages/collective/actions/org-fence/action.yml`, line 19; Section 4, Guardrails in tiers, carries the measurement) and `CLAUDE_CODE_OAUTH_TOKEN` in its environment (`packages/collective/actions/org-agent/action.yml`, line 100). An injected instruction to send that token is outside every control described here, and whether any run has been so steered is NOT MEASURED.

3.4 The safety floor: the completion decision

Between a finished agent run and a landed commit stand three decisions the package calls "the safety floor" (`packages/collective/README.md`). They are one command, `packages/collective/bin/collective-completion.mjs`: `halt` asks whether a kill switch is engaged,

`decide` whether the run finished, and `keep-set` what the run may commit. The README fixes three verdicts in the exit code, verbatim, and the code adds a fourth.

```
0   proceed, land it
3   stop cleanly, land nothing, this is NOT an error
1   RED: truncated, unusable, or a refusal to guess
4   undecidable: the HALT state could not be read from the ref
```

Exit 4 is `return 4` under `if (d.outcome === "undecidable")` (`packages/collective/src/completion.mjs`, lines 342 to 344), and the action treats it as red, "because an undecidable kill switch is not a pass" (`action.yml`, line 711).

The sentinel is a per-seat pair of regular expressions; the legal seat's is `sentinel-regex: "GC_REVIEW:|GC_SKIPPED"` and `skip-regex: "GC_SKIPPED"` (`.github/workflows/legal-weekly.yml`, lines 82 to 83). A tail matching the skip regex returns exit 3 and one matching the sentinel returns 0. Precedence is load-bearing: every sentinel regex is a superset containing its own skip token, so the skip regex is tested first and a tail matching both is a skip, never a completion (`completion.mjs`, line 426). A tail matching neither emits `RUN_OUTCOME=no_sentinel` (line 526), which the action records as `rejected` and the published metric counts against the seat (Appendix D). No `rejected` row existed on 2026-09-02 (`grep -c '"rejected"' docs/org/stats/*.jsonl`).

The `decide` window was learned rather than designed. The action originally required the sentinel inside the last 600 bytes of the result, and on 2026-08-05 a complete legal run was discarded with its sentinel 1,360 bytes from the end. Item 40 of the corpus, dated 2026-08-10 and classed a verification gap, states the shape: "A gate whose correctness rests on the thing it is gating is not a gate." The repair went to the mechanism, not the prompt. The window is now 25 lines under an 8,000-byte ceiling, `DEFAULT_TAIL_LINES = 25` and `DEFAULT_TAIL_BYTES = 8000` in `completion.mjs`, and the gate fails closed where Node and the shell disagree about regular expressions (Appendix D). When the agent's exit code or turn count is absent, "the gate says NOT OBSERVED rather than inventing a cause" (`action.yml`).

The `keep-set` decision is distinct from the fence's lane. Lesson 45 states it: "A lane is what the agent is TOLD TO WRITE, never what gets committed, and a deny list must walk EVERY level of it", because "`keep-paths` is what survives isolation to be COMMITTED" (`docs/lessons-learned.md`). The legal manifest carries three `lane_paths` and one `keep_paths`.

The agent never commits, an invariant the public process page also states (Appendix D). The completion decision runs first, inside the agent step (`action.yml`, lines 366 to 376). The next step computes the effective keep set: the caller's `keep-paths` and, once the ladder has activated the protocol, the seat's proposed-lessons file. An empty set aborts the run before isolation with exit 1 (lines 406 to 438). The action copies that set aside, runs `git reset --hard "$BASE_SHA"` and `git clean -fd`, and copies it back (lines 481 to 489), so isolation precedes every guard. The journal named in `docs/org/house-keeping.json` is not in the keep set; a separate call carries it through isolation (lines 484 to 485). What the workflow commits is wider, adding the dated archives, the stats row, the evaluation output and the journal; Figure 1 shows both sets (Appendix D). The push step asks the kill switch again before committing ("THE KILL SWITCH, RE-ASKED."), and the stats step runs `if: always()`, mapping skip, infrastructure, failure, halted and rejected each to a row in `docs/org/stats/<id>.jsonl`; only an unset outcome writes none, and an absent push token or three failed pushes drop the row with a warning (lines 818 to 916).

An optional `guard-command` runs between the run and the commit, set by two of the nine call sites; a refusal lands nothing and writes a `rejected` row, which the published metric buckets against the seat as a failure (Appendix D). The reset it grades on has a gap: two of the eight lanes

include `node_modules`, and `git clean -fd` at line 488 carries no `-x`, so a path altered there survives into the guard-command. A run that alters an installed package alters the instrument that grades it. Whether any run has written there is NOT MEASURED.

Org writers serialize against each other, so an evicted run concludes `cancelled` (Appendix D). The push step retries three times, rebasing on `FETCH_HEAD` between attempts (`action.yml`, line 745). Section 4, Guardrails in tiers, describes the canary that keeps that path exercised.

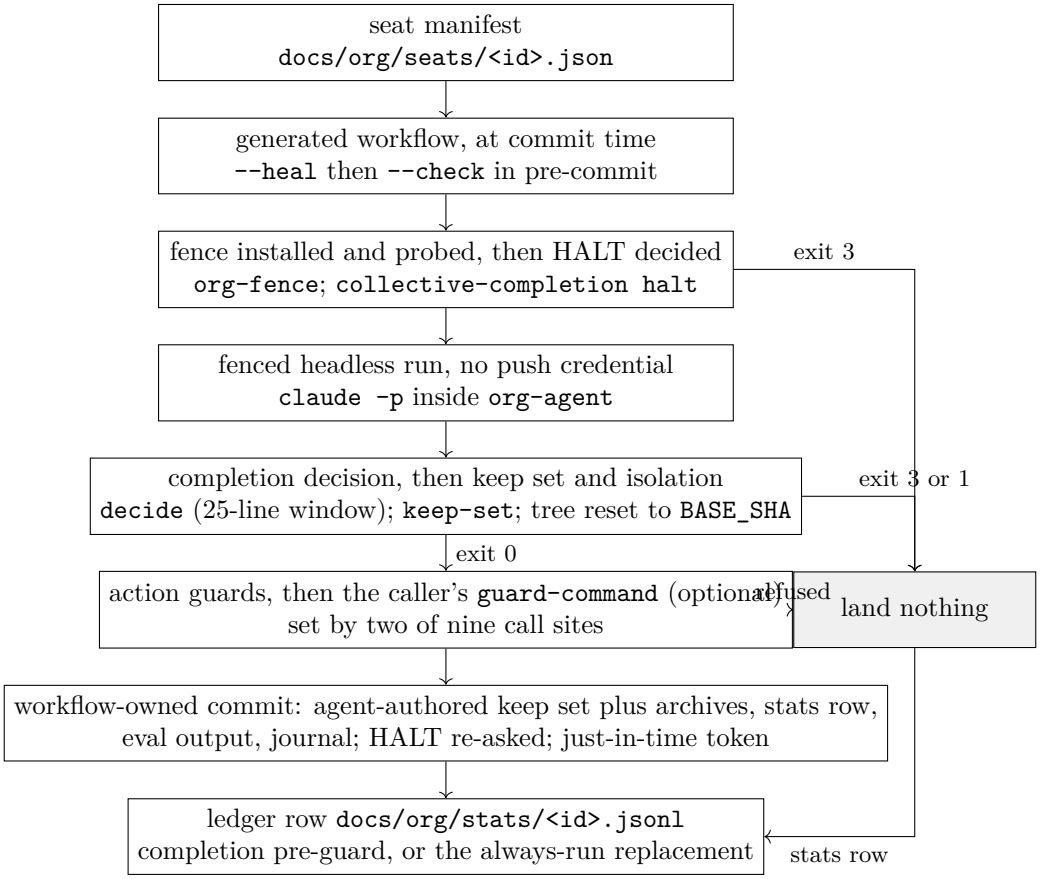

Figure 1: The path from a seat manifest to a landed commit, as the shared action executes it (`packages/collective/actions/org-agent/action.yml`, read 2026-09-02). Exit 3 stops cleanly and lands nothing; exit 1 is red and lands nothing; exit 4, an unreadable HALT state, is treated as red; a guard refusal lands nothing and is a `rejected` row the published metric counts as a failure; only exit 0 reaches the workflow-owned commit, whose commit set is larger than the agent-authored keep set. The completion row is written before the guards (lines 512 to 522) and rides that commit, so a refusal rolls it back and the always-run step writes a replacement. The ledger row is best-effort: a halt writes a halted row, an unset outcome writes none, and a failed push leaves none.

3.5 Isolation of the kernel

The three decisions live in `packages/collective`, created on 2026-08-15 in commit `c1cf77617`, a package its README calls "deliberately small and deliberately alone" and which declares no runtime dependencies (Appendix D). Its license is unresolved: the README and `package.json` say MIT, while D-0039 of 2026-08-12 chooses "Apache-2.0 plus a trademark policy" with the text unwritten (`docs/org/decision-log.md`; Appendix D). The company's own literals moved out the same day: "Sixteen literals were removed on 2026-08-15" (`docs/org/CLAUDE.md`, on `docs/org/house-keeping.json`).

Isolation is enforced two ways. A static outbound scan holds a baseline of zero. The orphan build, `npm run check:collective-orphan`, builds and tests the folder alone "with the rest of the repository absent", because a static scan sees only the reaches it knows how to spell. On its first run it caught three inherited couplings. It was then broken on purpose three ways and watched go red, one of them "a plain `fs.readFileSync("docs/org/...")` with no import statement at all, which no import-scanning lint could ever have found." The seam to the host is one file, `packages/collective/src/config.mjs` (Appendix D).

3.6 Memory law

Memory is three tiers, defined by what a seat loads: the constitution, creed, north star and vision; the operating agreement; and the seat's own charter, live lessons file and registry entry (`docs/org/operating-agreement.md`; Appendix D). The law has a negative half: "A seat NEVER loads a peer's charter, lessons, or runtime files." Every lesson commits to `docs/org/lessons/proposed/<id>.md`, unconditionally, and "A proposed lesson loads into nothing." Promotion is human, because "Only a signed human promotion makes memory permanent", and the guarantee is structural: the live lessons file is in no keep-paths allowlist.

The guard binds humans too. In `packages/collective/src/check-org-memory.mjs` a staged commit modifying a live lessons file blocks unless the environment carries `ORG_LESSON_PROMOTION=1`; that marker is the whole gate, and the file's only read of `docs/org/humans.json` exempts founder emails from its PII scan (Appendix D). A signature proves possession of a configured key rather than a human act, as Section 7, Design principles and a retrofit checklist, measures under P10. The human-ness of a promotion rests instead on the author-email pin and on `check-org-wiring.mjs`, which blocks any founder email under `.github/`. At stage 2 that file is in no effective keep set, and on 2026-09-02 `docs/org/lessons/proposed/` held twelve files of eight lines, every one ending "(No proposals yet.)" (`wc -l docs/org/lessons/proposed/*.md`); the quarantine has never held a proposal.

3.7 Persona lenses, de-named

The organization keeps a board of advisory lenses, viewpoints grounded in a named person's published work, where the lens is not the person (Appendix D). Rule 5 of its governance section reads "Names are INTERNAL ONLY." (`docs/org/board-of-advisors.md`), which is why this paper names none. `packages/collective/src/check-advisor-name-leak.mjs` enforces it, flagging a ratified name only inside advisory framing near it (Appendix D). The narrowness was earned: "on 2026-08-01 the bare-name version of this check produced 50 hits across 10 files, every one of them legitimate".

The board file records a claim about itself that was wrong twice. "The propagation was never automatic, and the claim that it was is the lesson", and a ratified seat "appeared in ZERO prompts for nine days". Its conclusion is the one this paper adopts: "This bullet has now been wrong in both directions. What survives is the guard, not the sentence." The guard is `__tests__/org/board-prompt-parity.test.ts`. Structural dissent lives in `docs/org/red-team.md`, advisory and never authority, with one of six triggers mechanized and a status table of 2026-08-01 saying "Nothing in the repo LOADS this file".

3.8 Decisions, predictions, and an intake path

The decision log is append-only by its header: "NEVER edit or delete a past entry; corrections are new entries. Ids are permanent." (`docs/org/decision-log.md`). On 2026-09-02 it held 45 entries, D-0001 dated 2026-07-13 to D-0045 dated 2026-09-02, the last shipping its own falsifier (Appendix D). D-0039 nearly did not land, having "lived for two days in `~/.claude/plans/`

cozy-cuddling-mochi.md, which is a file outside the repository, on one machine, in a directory no guard in this org can read". It draws the rule "A decision with nowhere to land lands nowhere."

The founder's own asks had no landing until 2026-07-29. docs/org/owner-directives.md opens with the incident: "The founder raised Bing's AI Performance report twice over several weeks. It left no trace." Its diagnosis: "The org had a synthesis path for everything the agents produced and no intake path for anything the founder said." Each directive carries an id, an owner, dates and a status, and the guard warns on one past due, unowned, or assigned to a department that does not exist (Appendix D). On 2026-09-02 the register held 12 directives, OD-0001 to OD-0012, and the pulse of 2026-09-01 reported "8 open, 2 past due." (docs/org/reports/pulse-latest.md).

3.9 Synthesis without authority

The Chief of Staff sees everything and decides nothing: "It has NO independent authority: information flows up THROUGH it, authority flows down FROM the CEO." (docs/org/roster.md). Its registry entry depends on "*", every department with a schedule. D-0019 of 2026-07-14 made its briefing the one inbox: "ONE consolidated CEO briefing: kill the standalone department emails". An audit that day found a legacy per-department email reaching the founder, because nothing looked at the wiring between agents. check-org-wiring.mjs looks now.

Authority is a generated table, and what it enforces is narrower than its public definition. docs/org/authority-matrix.md is generated from departments.json by packages/collective/src/build-authority-matrix.mjs, and it states that a lane is gated by default, with a move to autonomous a logged CEO decision (Appendix D). The public vocabulary page calls the matrix "The committed table of every seat's autonomous and gated lanes, enforced by the workflows that run the org rather than by instructions to the model." (app/(site)/orbyt-collective/how-it-works/page.tsx). The lanes are prose descriptors ("new pages", "pricing or product claims"; docs/org/departments.json, lines 51 to 52), and the only code reading lanes.autonomous or lanes.gated is the generator itself (grep -rln 'lanes\.\(autonomous\|gated\)' packages/collective/src scripts, 2026-09-02). The matrix is a drift-locked record of intent rather than an enforcement mechanism.

One act landed inside a gated lane. On 2026-07-30 the content seat committed a new page, app/(site)/guides/technical-interview/page.tsx, in e5bd0f6e0 (git log --author='orbyt-content-bot' --diff-filter=A --name-only -- 'app/(site)'), while "new pages" had been in marketing's gated lane since 672a7c303 of 2026-07-13. That day's keep-paths named the whole app/(site)/guides directory (Appendix D). No instrument noticed.

The Inspector General reports and "is deliberately unable to act" (docs/org/authority-matrix.md). Its v0, packages/collective/src/inspector-general.mjs, runs six modules, every one targeting the product: "(No org-targeted module exists yet; the stage-2 no_org_fabrication criterion reads target=org rows when they do.)" (inspector-general.mjs, line 43). Its stage-2 criterion no_org_fabrication is computed as latestFm !== null && !fabricated, where fabricated becomes true only when a finding carries target equal to "org" (Appendix D). While no module can emit such a finding the criterion cannot fail, and docs/org/ladder-state.json reads "no_org_fabrication": true (line 34). A criterion that cannot go red is a promotion gate that has never been able to fire. The seat's footer is the counterweight, treating an unfinished module as a finding and a skipped one as unknown rather than clean: "the IG never 'passes' a finding quietly."

That footer rule is the smallest statement of the design law this paper argues for: a control that

could not look reports that it could not look. Section 4, Guardrails in tiers, applies it to the guards around every seat, and Section 5, Liveness and the watch chain, to the monitors that watch them.

4 Guardrails in tiers

The Machine's guardrails are three, and the difference between them is the autonomy line: what only prints, what blocks, and what refuses before the tool runs.

4.1 Three tiers

The bottom tier is report-only, which is the default. On 2026-09-02 the repository carried 110 guard scripts under `scripts/check-*`, 11 org guards in `packages/collective/src/check-*.mjs` and 91 registered audit dimensions (Section 6, The empirical record). Of those 121 scripts 94 name a `_STRICT` variable, 20 name none and hold a `process.exit(1)`, and 7 name neither; among the 20 is `scripts/check-commit-author.cjs`, which exits 1 at line 110 (`git ls-tree` at `424a58007` with grep, 2026-09-02). A `_STRICT` name is not a report-only promise: `check-org-memory.mjs` exits 1 on its own no-live-landing law and makes only warnings conditional (lines 228 to 230). A report-only dimension exits 0 even with findings, and `docs/audit-dimensions/governance.md` states the consequence: "A **report-only** dimension cannot break anything. It only prints. Worst case it surfaces debt nobody acts on."

The middle tier is the same script under an environment variable. A report-only guard reads a `<NAME>_STRICT=1` variable that turns its findings into a non-zero exit, and 96 distinct such variables existed on 2026-09-02. Promotion is a placement decision, not a code change: CI sets 19, the pre-commit hook three, and ten dimensions are STRICT there by number. Dimension 42 blocks with no STRICT variable, so that ten counts step names rather than gates (Appendix D). `governance.md` says what a gate costs: "One bad threshold blocks every commit and deploy until a human notices."

The top tier is a refusal in two parts: a PreToolUse hook (`.claude/hooks/arc-valve.sh`) that denies a tool call before it executes, and a `permissions.deny` list in `.claude/settings.json` enforced by the runtime without the hook. `governance.md` draws the line in one passage, then tabulates what follows:

> So the autonomy boundary is not "tell the owner after the fact." It is **which tier an agent may touch unattended.** Report-only is the sandbox; STRICT is the gate. Agents live in the sandbox; the gate is the owner's.

Table 4: The autonomy envelope, reproduced verbatim from `docs/audit-dimensions/governance.md` (read 2026-09-02).

An agent MAY do unattended (inside an `/evolve-audit` run)	An agent MUST get owner sign-off for
Create a **new report-only** dimension	Add or flip **any STRICT / pre-commit** gate
Improve a report-only dimension's checks (incl. fixing a false positive/negative)	Weaken or remove an existing gate or dimension
Re-tune AI / MCP red-team prompts for a new model (report-only)	Re-tune a threshold on an existing **STRICT** gate
Propose a STRICT promotion into the queue	Touch a protected file (`supabase/migrations/`, `app/api/webhooks/stripe/`, any `.env`)

An agent MAY do unattended (inside an `/evolve-audit` run)	An agent MUST get owner sign-off for
Commit a report-only dimension (the `/evolve-audit` invocation is the authorization)	**Push** (deploy stays a human action)

A new dimension lands only through a seven-precondition gauntlet, "mechanized, not promised", and the one breaking class is human-gated (Appendix D).

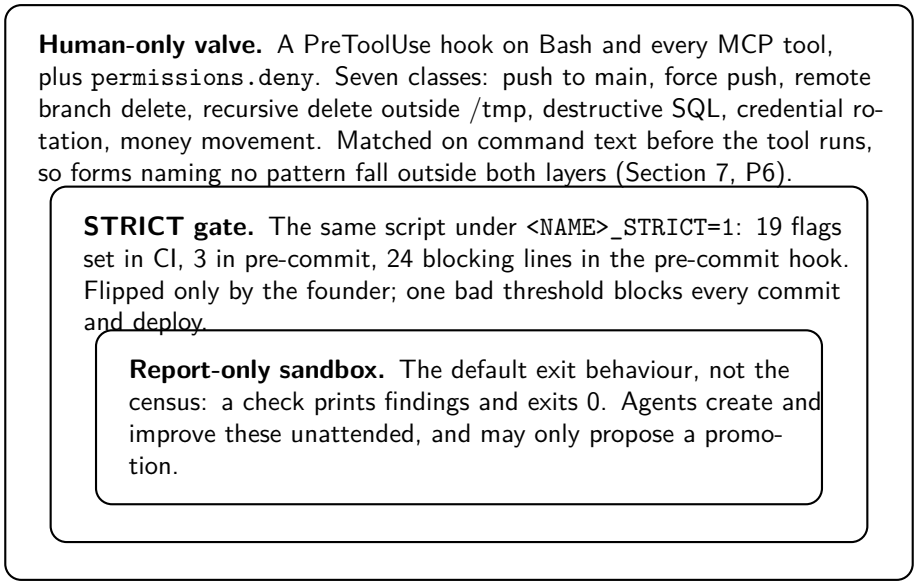

Figure 2: The three guardrail tiers as nested regions. An agent lives in the innermost region and may propose, never flip, a gate in the middle one; the 121 check scripts do not all sit there, and the text classifies them by default exit behaviour. The outer region is enforced by the runtime before any tool executes. Counts measured 2026-09-02 on `main` at `424a58007`.

Table 5: The three tiers and the eleven human-only acts, expanding the outer region of Figure 2. Read 2026-09-02 from `.claude/hooks/arc-valve.sh`, `.claude/settings.json`, `docs/org/seats/*.json` and `scripts/vercel-ignore-build.sh`; P5 and P6 are in Section 7, Design principles and a retrofit checklist.

Tier or item	Scheduled seats	Interactive sessions	Observed refusal
Report-only tier	prints in CI, exits 0	same, exits 0	none possible
STRICT tier	19 CI flags; gates outside every keep-set	3 pre-commit flags	NOT MEASURED
Human-only tier	no push credential; hook loading NOT MEASURED	valve, plus `permissions.deny`	one self-test, 2026-08-23
Push to main	workflow-owned push, token unseen by the agent	valve class, 5 deny entries	NOT MEASURED
Force push	NOT MEASURED	valve class, 2 deny entries	NOT MEASURED
Destructive SQL	NOT MEASURED	valve class, Bash and MCP	NOT MEASURED

Tier or item	Scheduled seats	Interactive sessions	Observed refusal
Credential rotation	NOT MEASURED	valve class, Bash and MCP	NOT MEASURED
Money movement	NOT MEASURED	valve class, Bash and MCP	NOT MEASURED
Protected files	fence denies; isolation discards	no valve or deny entry	NOT MEASURED
Customer-facing content	none: `app/(site)` keep-set paths deploy	none	none: 11 commits, 21 paths
Legal commitments	a lane rule only	none	NOT MEASURED
Widening a lane	lane from the prompt; a change is a manifest commit	none	NOT MEASURED
Lesson promotion	`check-org-memory.mjs` blocks unless `ORG_LESSON_PROMOTION=1`	same guard	NOT MEASURED
STRICT flip	gates outside every keep-set; edit reset at isolation	no valve or deny entry	the human half, 2026-06-24 (P5)

4.2 The commit gate

The middle tier's main instrument is a git pre-commit hook, `scripts/pre-commit`, 567 lines with 24 blocking lines and six bare exits in conditional blocks (Section 6, The empirical record). It is a symlink rather than a copy, because "A cp silently goes stale the moment scripts/pre-commit changes" (`scripts/install-hooks.sh`). It reaches hosted runners too (Appendix D).

The ordering is a design record: the commit-author guard runs second, because a commit under an undeployable identity is wasted whatever the checks say. Section 6, The empirical record, tells what that cost on 2026-08-14.

The last blocking step runs the full suite, every tier (`bash scripts/test-gate.sh`, line 505). Until 2026-08-01 it ran only the fast tier, and the gap put the main branch red that day. The rule that followed is in the hook's comment: "A local gate that runs a DIFFERENT set of tests than CI is not a gate, it is a rehearsal for a different play." The speed rationale was measured and lost by six seconds over 294 tests (Appendix D).

That step was itself a blind gate, and the repair is documented lesson 47 (`docs/lessons-learned.md`, 2026-08-13). The hook read pass or fail from vitest's printed summary. On a GitHub runner vitest colours it, so the bytes read `\033[2m Test Files \033[22m \033[1m\033[31m1 failed\033[39m` and the regex's "Test Files" was no longer followed by a digit. Its two alternative branches were dead too, so in CI the gate had none live (Appendix D). Run 31720972207 printed "Test Files 1 failed | 965 passed", and the marketing seat landed `9844de561` on main anyway. The fix moved the verdict into `scripts/test-gate.sh`: "the EXIT CODE is the verdict. vitest already computed it, it carries no formatting, and no reporter change can rewrite it."

Continuous integration is the gate's second half. `.github/workflows/ci.yml` declares 9 jobs on 2026-09-02, and its `guard` job carries 21 matching source lines under a 10 minute ceiling, which execute 38 check commands over 33 distinct scripts, because one of those lines is a loop over 18 more and five scripts repeat; Appendix D names them and quotes the STRICT confinement

step. The same battery runs here, so a `--no-verify` commit meets it anyway.

4.3 The human-only set

The top tier is wired in `.claude/settings.json`, which registers `.claude/hooks/arc-valve.sh` as a PreToolUse hook on `Bash` and again on `mcp__.*`. Its header: "Runs in every session, not only arcs: Claude pushing to main or deleting prod is never a Claude decision." It denies the seven classes Figure 2 names, three on every MCP tool as well (Appendix D). Every denial returns one text, naming the class and carrying "Do not rephrase to get past this." ahead of its recovery instructions.

Beneath it sits the runtime's own deny list, five `permissions.deny` entries, each a `git push` shape naming main or a force flag (Appendix D). The valve's header calls that list the hard gate and itself defense in depth: a deny entry cannot be disarmed by a hook that fails to start. Neither layer has an observed refusal of a real push: the valve's one denial, on 2026-08-23, was of its own installer's self-test, and `permissions.deny` is NOT MEASURED.

Section 7, Design principles and a retrofit checklist, tabulates under P6 what each layer denies. A denial is answered by changing channel, not by rewording, a rule `CLAUDE.md` states (Appendix D). The two fail independently, an absent hook against a pattern that misses, while sharing their coverage: a bare `git push` from a branch tracking `main` matches neither, and in Reason's model two layers with one gap are one layer. Whether that shape has been attempted is NOT MEASURED.

The deploy half of "deploy stays a human action" is not earned by the valve. The shared action pushes every completed seat run to `main`, which auto-deploys production. In between sits `scripts/vercel-ignore-build.sh`, run before every production build. It skips org documents, books, workflow files and three generated artifacts, and builds on any other path and on any doubt (read 2026-09-02; the paths are in Appendix D). Its header records what came before: "EVERY org seat run was triggering a full production build and deploy." The guarantee is a skip list, not a human gate: a keep-set path outside it deploys production with no human step. This is the one tightly coupled path, in Perrow's sense, in a loosely coupled design.

The valve is part of a kit for long-horizon work installed on 2026-08-23. `.claude/settings.json` registers four hook scripts across five entries. Among them are a turn counter and a stop gate needing a closed ledger block to end an arc session. An arc's state lives in `docs/org/arcs/<slug>/ledger.md`, and `find docs/org/arcs -name ledger.md | wc -l` returned 0 on 2026-09-02. The kit has never carried an arc to a proved condition. Whether these hooks load inside a fenced seat run is NOT MEASURED; a seat's push is refused there by a missing credential (Section 3, The Machine: architecture of a governed seat).

4.4 The fence and positive observation

The fence is the machine-level filesystem sandbox seat runs have executed inside since 2026-08-13, when commit `cf71ad735` created the action (Appendix D). At least 33 of the 65 ledger runs executed without it (Section 8, Limitations). `packages/collective/actions/org-fence/action.yml` installs it as managed settings on the runner. A lane cannot be written as an allow rule, because a deny beats an allow even for a nested path, so it is built by enumerating siblings to deny, regenerated every run (Appendix D). Siblings are denied at every level, not only the top: on 2026-08-13 "a lane of docs/org kept the whole of docs/ writable, because only top-level siblings were enumerated." The list has a ceiling of 600 paths, because every deny path enters the argument vector of every command. That day "a 1254-path fence produced a 135.8KB command line and E2BIG, and the seat could not run one command."

The action states three things it does not do, each tested. On egress: "It does NOT fence network egress. `sandbox.network.allowedDomains` plus `allowManagedDomainsOnly` were measured doing NOTHING on a hosted runner: off-allowlist hosts answered 200 while the filesystem half of the SAME fence denied every write." On the sandbox's own health flag: "It does NOT trust `sandbox.failIfUnavailable`. That flag missed all three sandbox failures measured on 2026-08-12 and 2026-08-13, including the partially-applied case." The third is that it does not check that a settings file exists, since an invalid one is ignored silently (Appendix D). Lesson 44 is the probe behind the second (2026-08-13): a run concluded `success` with `permission_denials: []` while three of its four agent runs had executed nothing, the network half having failed to apply. The rule it produced names this section: "fence liveness must be a POSITIVE OBSERVATION of a real denial inside every run".

The liveness step makes the fence a control, not a configuration. The action runs a small agent told only to write a canary file at the first denied path, and reads the denial from the transcript, not from a file (Appendix D). If the canary exists, the step fails with "FENCE NOT LIVE. An out-of-lane write SUCCEEDED". If no denial was recorded, it fails with "FENCE NOT OBSERVED. No denial was recorded, so the absence of the canary proves nothing. This is the silent-ignore state, not a pass." A third branch, added after run 32687813516 exited 1 after 2.2 seconds having printed nothing, separates an agent that never ran from a fence unwatched: "This is NOT evidence the fence is down; it is evidence it is UNTESTED." A recorded refusal proves less than "fence live" reads: that deny rules held in this run, on one path of 600, and nothing about the other entries or about whether the lane is right.

Two canaries watch the fence and its push path. `.github/workflows/fence-canary.yml` opens with its reason: "The fence rests on undocumented behavior, so something has to watch it." It calls the real action daily, and has failed twice in 23 runs against 21 successes. The first was 32687813516 on 2026-08-24, which produced issue #106, "Fence canary is RED 2026-08-24: the sandbox fence may no longer come up". The second was 33376635228 on 2026-08-31, which ran zero steps in the budget blackout Section 5, Liveness and the watch chain, treats (`gh run list --workflow fence-canary.yml --limit 100 --json conclusion, createdAt, databaseId`, 2026-09-02). Neither red was deliberate, and whether either lost the fence is NOT MEASURED. `.github/workflows/push-collision-canary.yml` exists because the push retry "ONLY RUNS WHEN A PUSH IS REJECTED, and nothing in production can produce a rejection on purpose." On 2026-08-16 run 31973282677 "concluded success, printed 'Pushed on attempt 1.', and left every line of the rebase path unexecuted." It now manufactures a real rejection weekly, against a ref no deploy can see.

`packages/collective/src/check-agent-confinement.mjs` turns the fence from a convention into an invariant: a workflow that executes an agent and checks out the repository must run the fence and check out with `persist-credentials: false` (Appendix D). It exists because generated seats are safe by construction while "three workflows already bypass" the generator, and "A lesson mechanized into the canonical pipeline does not exist for the surfaces that bypass it." That is documented lesson 36 (`docs/lessons-learned.md`, 2026-08-02): thirteen days after the canonical fix the advisory panel replayed the dash death the action had closed. All three now normalize before guarding, drift-locked by `__tests__/scripts/org-guard-parity.test.ts`. The guard reads comment-stripped YAML, since its first version flagged two workflows for a sentence promising not to run an agent, read as running one (Appendix D). It checks no lane. A lane comes from the seat's prompt, not the keep-set, and nothing checks that.

4.5 The ratchet and the kill switch

The ladder is the five-stage promotion ratchet in `docs/org/rollout-ladder.md`, ratified with decisions D-0024 and D-0028 on 2026-07-20. It states the law it walks: "the ratchet ACTI-

VATES pre-ratified advisory stages on committed deterministic evidence; it never grants authority. Promotion evidence is check results, never agent self-assessment." The machine state lives apart in `docs/org/ladder-state.json`, which pins the criteria file's hash, so editing the criteria resets every streak. On 2026-09-02 it reads `active_stage: stage2`, with `ig_runs` and `ig_coverage` false under a note naming them build debt, not failing measurements: "ig_runs needs an IG smoke-eval scenario; ig_coverage needs the coverage manifest. Both are recorded Stage 2 build debt; stage2 holds honestly until they exist." It also reads `liveness.alive: false` and `promotions_paused: true`, for "no signed founder commit in 14 days (or signing not yet configured)". That brake, and the one stage the ladder advanced by itself on 2026-08-14, are measured under P10 in Section 7, Design principles and a retrofit checklist.

Below it sits a per-seat evaluator, `packages/collective/src/evaluate-seats.mjs`, whose `VERDICT` table holds six values, among them `not_observed` and `dormant` (Appendix D). Demotion follows three repeated problem verdicts: one is a bad day, three a pattern nobody acted on. Its thresholds are exported constants: a window of 6 runs, a failure rate of 0.34, 4 entered runs before the rate test fires, and 21 dormant days. A zero-turn run is its own category, after the first version counted such runs as failures and "made marketing look like a 67% failure rate and marked it for a halt". A guard-rejected run is not excused: that refusal is the seat's defect. The file states the asymmetry that governs the ratchet: "Promotion has no path here, by founder decision: the machine can suspend itself and cannot promote itself."

The remediation record, `docs/org/seat-state.json`, adds a rule: a cycle in which nothing changed escalates to the humans, not the seat (Appendix D). It holds two open remediations on 2026-09-02, opened 2026-08-27, for the innovation and marketing seats, each with one unverified intervention. The marketing entry names its open question: the guard that refused two runs "has not been identified, so the real remediation for the seat's own quality has not been attempted yet."

The kill switch is one file. `docs/org/kill-switch.md` states it plainly: "The presence of the file **`docs/org/HALT`** halts every autonomous ORG workflow." Fifteen workflow files check that path (`grep -rl 'docs/org/HALT' .github/workflows | wc -l`, 2026-09-02). It names its own limit, "HALT is the org kill-switch, not a global one.", and then overstates it: of the four ungated app-maintenance crons, two write production data rather than pull requests (Appendix D).

What HALT guarantees is narrower, and the action's own comment says so. It is checked at preflight and again at the push step. Over the 15 to 30 minutes an agent takes, that makes it "a kill switch for runs that have not started rather than for runs that are running" (`action.yml`, lines 688 to 692, read 2026-09-02). The re-ask reads the remote, not the checkout, because isolation resets the tree to the base commit (lines 701 to 704, and the row in Appendix D).

Table 6: What the organization HALT guarantees, read from `packages/collective/actions/org-agent/action.yml` (lines 688 to 706), `docs/org/kill-switch.md` and `packages/collective/src/evaluate-ladder.mjs` (the `written_by` parse at line 135) on 2026-09-02.

Property	What the record shows
What it stops	The landing: preflight refuses to start a run, and the re-ask refuses to push one
What it does not stop	A run already in progress (15 to 30 minutes of spend); the four app-maintenance crons, two of which write production data; the ladder and the backup, the two ratified survive-the-halt exceptions

Property	What the record shows
When a mid-run HALT is seen	At the push step, decided against `FETCH_HEAD` after a fetch of `origin/main`
Who can set or lift it	Any holder of a push token; only scoped `HALT-<id>` files carry `written_by`, and the global file has no provenance or signature check

The scoped form has provenance. A file named `docs/org/HALT-<id>` stops one seat, and its `written_by` field decides who may clear it: the evaluator its own, the ladder the dormancies it pre-staged, a human theirs (Appendix D). A halted seat produces no runs to read clean, so `docs/org/operators-manual.md` provides a supervised re-run that bypasses that halt and never the global file. The history is five commits (Section 6, The empirical record); no HALT file existed on 2026-09-02.

4.6 NOT OBSERVED and the rules that survive a session

Every instrument above shares one verdict vocabulary, whose third state carries the law. `docs/org/CLAUDE.md` states it: " 'could not look' reports NOT OBSERVED and never a finding; seeing nothing is never a pass; the request budget is bounded." A guard without credentials says SKIPPED; a monitor whose query failed prints NOT OBSERVED with the error's first line. Neither renders as clean: a verdict from an instrument that did not look is indistinguishable from one that did. The verdict is printed, the process exits 0, and the ladder's alarm fires only on `failure` or `cancelled`, so at the layer that escalates, NOT OBSERVED and clean are the same event. Nothing bounds how many consecutive days a monitor may report it (Section 5, Liveness and the watch chain).

The rules an agent carries between sessions were written on 2026-07-23, after seven errors in one day. They are mechanisms because, in `CLAUDE.md`'s words, " 'Be more careful' is unfalsifiable and does not survive into the next session." The first is that every detector ships with a negative case, one it must not flag. Four mis-fired that day. A pairing heuristic produced 23 false positives, and a `.storage.from()` bucket was read as a missing table. A loose `/skipped/i` made an observer report blindness while seeing perfectly, and a scoped `HALT-<id>` read as global let one dormant seat silence the whole watchdog. Each had passed its own suite, "because the suite only ever proved it could be LOUD." The second rule is that " 'Done' means the ARTIFACT was read back, not that the file was edited", written after planner estimates were reported as row counts. The third is "Mechanize the specific miss, do not write a lesson about it", with the corollary that "a guard is not owned until you have seen it fail."

Two dimensions enforce them on the rest. Dimension 72, `scripts/check-doc-claim-parity.cjs`, carries a sixth sub-check labelled "every guard's test proves it can be QUIET", warning on any `__tests__/scripts/check-*.test.ts` that "never asserts the guard stays QUIET on a clean case". Dimension 82, `packages/collective/src/check-audit-liveness.mjs`, was written on 2026-08-05, after a sweep of every registered dimension found three "that produced zero bytes of output and exited 0". Two were reading an empty staged diff; the third "PRINTED 'clean (0 file(s) scanned)', an affirmative all-clear, on the credential surface." None was broken; "They were pointed at nothing, and none of them said so." It therefore requires every registered command to speak, on the rule we take as this section's last word: "This dimension grades whether an instrument SPEAKS, never whether it likes what it saw."

5 Liveness and the watch chain

A seat that stops running looks, from the outside, like a seat having a quiet week. This section describes the watch chain The Machine built in response. Its links are the daily dead-man switch, the job that watches the switch, the external heartbeat, and the agentless daily pulse. It ends with the one failure none of those links could see, the budget blackout of 2026-08-31.

5.1 Why a dead-man switch

The dead-man switch is `packages/collective/src/check-org-liveness.mjs`, and its header dates its origin: "Why this exists (2026-07-23). The Chief of Staff weekly briefing, the CEO's single consolidated window into every department, failed on 2026-07-17 and stayed failed for six days with nobody the wiser. Product weekly had been failing since the 22nd." The header names the class in one sentence: "A workflow that fails on a schedule fails QUIETLY: the departments keep writing, nothing assembles it, and the silence is indistinguishable from a slow week."

The remedy split liveness from analysis. Liveness asks whether a seat ran and succeeded, daily and with no model. Analysis asks what the seat concluded, weekly and at a full agent run per seat. The outage "did not need seven briefings, it needed one signal saying the briefing did not happen." Five constraints are marked deliberate in the file. It adds no cron and rides the daily ladder workflow as a job. It has no emailer, so "A red scheduled run IS the alarm." It commits nothing, runs no agent and spends no token. It honors HALT, since a switch "that screams during a planned halt gets muted, and a muted alarm is not an alarm." And it measures from the date the workflow file entered the repository, so "a newly wired seat reads as PENDING, not dead."

5.2 A window, not a last run

The first version judged only the most recent run and flagged any failure regardless of age, and both directions bit. One failed dispatch of the advisory panel on 2026-07-14 "held this switch red for eight consecutive days, 2026-07-24 through 2026-07-31, while the seat itself was healthy." Those same eight days cost a second thing. The workflow carrying the switch was red throughout, which is why the ladder gained an alarm job. Its comment records: "the workflow failed on eight consecutive scheduled days, 2026-07-24 through 2026-07-31, and reported it to nobody." It then names the reader problem: "'A red scheduled run IS the alarm' holds only for someone who opens the Actions tab, and for eight days nobody did. The dead-man switch had no dead-man switch."

The false green was the mirror image. The conclusion `cancelled` "was in neither flagged conclusion, so a run killed by its own timeout read as ALIVE." That is documented lesson 28 (`docs/lessons-learned.md`, 2026-07-31), and on 2026-08-01, the morning of the rewrite, the content seat was "sitting on two of them and reporting OK."

The resolved rule is a window: "So a seat is ALIVE when a SUCCESSFUL run exists inside its cadence window, and nothing else counts. A newer failure on top of an in-window success is a WARN, not a verdict." The seat's own failure alarm owns that red, and "double-reporting one incident under a signal that means 'dead' is how the signal stops meaning anything." The window is derived from each workflow's cron expression (`windowFor`, read 2026-09-02): two days for a daily schedule, nine for weekly, 35 for monthly, 100 for quarterly or yearly, and 40 when the expression cannot be parsed. The parser, as we found by running it, tests the day-of-month field before it tests for daily (lines 193 to 195, read 2026-09-02), so any non-star day-of-month is classed monthly. The chief of staff's sprint cadence, `0 13 */2 * *` in `chief-of-staff-weekly.yml`, is therefore judged on a 35-day window: `node`

`packages/collective/src/check-org-liveness.mjs` printed `chief-of-staff-weekly.yml (monthly) last succeeded 1d ago (window 35d)` on 2026-09-02. The seat whose six-day silence motivated the switch is the one it watches least closely.

The exit contract is the file's own: it exits 1 when any seat has no successful run inside its window. It exits 0 "when healthy, halted, or when the GitHub API is unavailable (it reports that it could not look; a watchdog that cannot see must not invent a verdict)." Nothing automated counts consecutive NOT OBSERVED days, and a bound on them is an instrument the organization has not built.

Since 2026-08-14 the roster is derived from the org manifest. The hand-written list it replaced watched the undeclared chief of staff and missed three declared workflows, among them the finance ledger. An empty roster now throws, because "a watchdog whose roster silently empties reports a healthy org while watching nothing, and that reads identically to real health." The switch judges run conclusions from the API, never the stats ledger the watched system writes (Section 6, The empirical record).

5.3 Watching the watcher

The switch rides `.github/workflows/org-ladder.yml`, which fires at 11:17 UTC daily with the four jobs Figure 3 draws: `evaluate`, `liveness`, `ci-headroom`, and the `alarm` that needs all three.

The headroom check, `scripts/check-ci-headroom.cjs`, is lesson 27 mechanised after it recurred. A `build` job carried `timeout-minutes: 15`, and "Seven of the last twelve runs on main died at 15m13s to 15m19s", unreported because the CI alarm "was gated on `failure()`, which is false for a cancelled job." The guard samples real durations on `origin/main` against every declared ceiling, warns at 75 percent, flags critical at 90 percent, and treats any `cancelled` conclusion as critical. In the ladder workflow it runs with `continue-on-error: true` (line 149), because the ladder's red means a seat is dead, "and overloading that signal is how a real alarm gets ignored."

The cost is a link with no automated reader. The alarm job fires only when a needed job's result is `failure` or `cancelled` (line 203), which a job under `continue-on-error` never is, and the pulse does not read the headroom result. The alarm's issue body prints the headroom result (line 221, read 2026-09-02) only when another job went red. Whether the alarm has ever fired on a headroom red alone is NOT MEASURED. A headroom red's only reader is a person opening the run log.

The heartbeat is the one link that leaves GitHub. After a successful state push, and only then, `evaluate` requests the URL in the `ORG_HEARTBEAT_URL` secret. The step's comment records a false state it once had: "An empty secret used to print a note and exit green, so 'the external absence alarm is disarmed' looked exactly like 'the heartbeat works'." Since the monitor was armed on 2026-07-21, an empty URL on a scheduled run exits 1. `docs/org/operators-manual.md` specifies the far end, one ping a day with a 12 hour grace period: "add BOTH notification channels (email and SMS or phone push) per the dead-man design".

The far end has fired twice on real absences, through its GitHub-issue channel. Issue #99, "Orbyt is DOWN", arrived at 2026-08-20T23:42Z, the night of the one global halt (Section 6, The empirical record). Issue #126 arrived at 2026-09-01T03:22Z, the morning after the blackout. Both were filed by `app/healthchecks-io` (`gh issue list --state all --limit 1000 --json number,author,createdAt,title`, 2026-09-02); issue #46 of 2026-07-21, "TEST is DOWN", is the arming test. Whether the email and SMS channels fire is NOT MEASURED, and the heartbeat drill, procedure 8 of the manual, has never been logged (Section 8, Limitations). Figure 3 draws the graph.

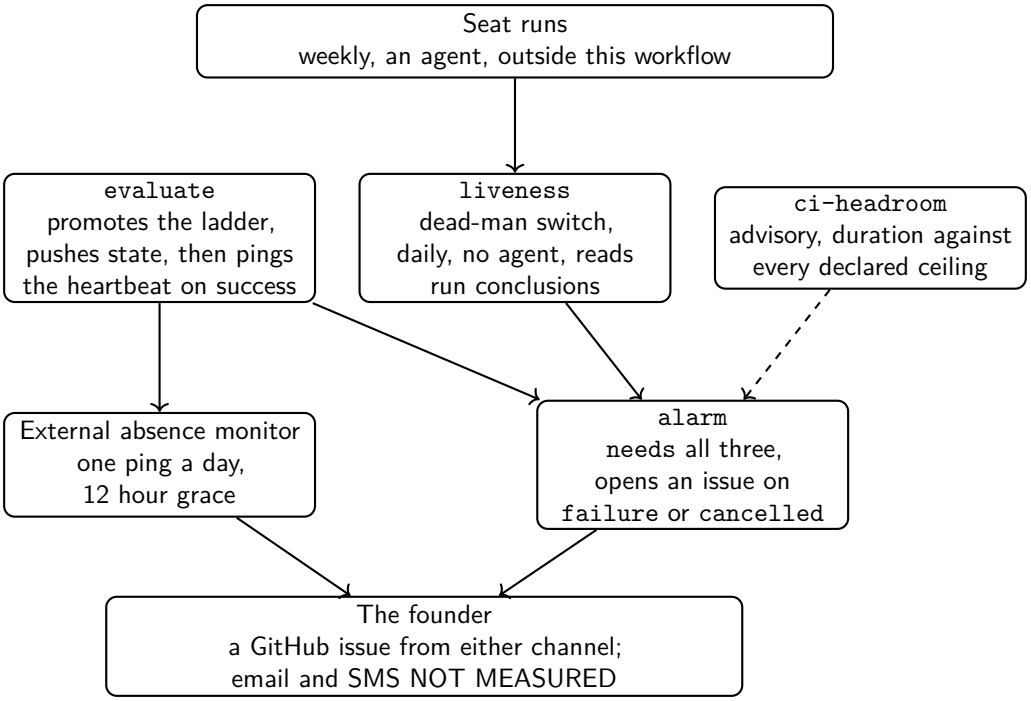

Figure 3: The watch chain as `.github/workflows/org-ladder.yml` actually declares it (read 2026-09-02). `evaluate`, `liveness` and `ci-headroom` are siblings at lines 25, 107 and 146; only `alarm` declares `needs`, on all three, at lines 199 to 203. The state push and the heartbeat are steps inside `evaluate` at lines 56 and 69, not a layer after the alarm, so a failed `evaluate` sends nothing outside GitHub. The dashed edge is `ci-headroom` under `continue-on-error` at line 149: its result is never `failure`, so a headroom red alone never fires the alarm. Each layer exists for a death the layer it watches cannot report. A seat cannot report its own absence, a run that never started, or a green run that wrote nothing. The dead-man job cannot report a job about to outgrow its ceiling, since a success at 98 percent reads as a success. An alarm inside the workflow cannot report the workflow never firing, or the account refusing to start it, which is why the absence monitor sits outside the platform. The heartbeat was armed 2026-07-21; the monitor's issue channel fired on 2026-08-20 and 2026-09-01.

5.4 The pulse

The pulse is `.github/workflows/org-daily-pulse.yml`, scheduled at 19:00 UTC daily and built for founder directive OD-0006 of 2026-07-29. It contains no agent, so "it cannot fabricate a number, because no model touches one." It also states: "it cannot go green while writing nothing, which is the exact failure class that produced this directive (Product and Legal green with 8-day-old reports)." `packages/collective/src/check-org-run-truth.mjs`, the guard that closes that class, states the diagnosis: "The founder was the monitoring. That is the defect this file closes."

The script, `packages/collective/src/daily-pulse.mjs`, adds the contract: "every input it reads has exactly three reportable states: observed-clean, observed-red, and NOT OBSERVED." A checker's always-printed `org-run-truth:` header is required as a ran-at-all sentinel, and a NOT OBSERVED verdict carries the failing command's first stderr line. Its exit code is always 0, because "The pulse is an instrument, not a gate." Under a HALT the workflow emails the pulse and skips the commit (lines 123 to 124 and 179 to 180, read 2026-09-02), so a forgotten HALT is reported daily by one channel and recorded by none.

Lesson 35, found and fixed 2026-08-02, records three blind branches live at once. A crashed

checker rendered as clean, a starved scope rendered as "gh was unavailable", and a newest-60 page reported 1 red run on a day the window held 11. The workflow's `permissions` block records the second of those recurring three times: `actions: read`, then `pull-requests: read` on 2026-08-02, then `checks: read` and `statuses: read` on 2026-08-03. Before the first, "the pulse printed NOT OBSERVED for red runs EVERY DAY since it shipped." The comment names the mechanism: "A permissions block is a ratchet that every NEW gh subcommand silently falls outside of." The drift lock in `daily-pulse.test.ts` maps each subcommand and field to its scope.

The pulse also settles a division of labour that `docs/org/CLAUDE.md` states as law. A `timeout-minutes` kill leaves the run alive for the seat's in-run alarm, but a run-level teardown skips every remaining job, "so the in-run alarm can never report the death that killed its own run." It was verified on 2026-08-06, when the quality and content seats concluded `cancelled` having never been assigned a runner, and the pulse of 2026-08-07 named both. The instruction is explicit: "Do NOT add a second alarm to shadow it, and do not 'fix' the in-run alarm to cover run-level death, because it cannot, in-run."

One step of the pulse holds read-only production credentials, granted by D-0038 of 2026-08-13 (`docs/org/decision-log.md`) after the cost of the no-credentials position was measured. The live sub-check of `scripts/check-cron-continuity.cjs` "had never once executed, in any context, ever", and in that gap `/api/cron/stats-refresh` "returned HTTP 500 every day for 46 consecutive days" while every repo-reading dimension stayed green. The grant was proven both ways: with credentials the pulse listed all five live findings, and with `PULSE_OFFLINE=1` it printed NOT OBSERVED. Sub-check 5 alarms on the failure streak, rate and duration headroom, "never on the last run's status alone", and counts rows stuck in `running`.

Lesson 27 and the code disagree about the reaper for those rows. `docs/lessons-learned.md` (2026-07-31) records that "The ingestion capacity fix and the reaper are owner decisions, carried in the launch ledger." The reaper nonetheless exists. `app/api/_lib/cron-tracker.ts` has carried `reapAbandonedRuns` since commit `14c24b6e7` of 2026-07-31, and `docs/launch-prep/LAUNCH-READINESS-LEDGER.md` records the landing under "cron_runs reaper shipped (lesson #27's far side)" (line 1035, read 2026-09-02). The lesson file was not updated, and whether the ingestion capacity fix has landed is NOT MEASURED.

5.5 The blackout

On 2026-08-31, the last day of the billing month, GitHub refused to start any job in the repository. Decision D-0045 (`docs/org/decision-log.md`, 2026-09-02) records the annotation each job carried: "The job was not started because an Actions budget is preventing further use". It records the extent: "Every scheduled seat, the pulse, the ladder, both canaries, GEO, model-refresh and Bing concluded `failure` with zero steps, and the 2026-09-01 briefing read the silence as the paperwork lagging." August wall time was about 8,188 minutes over roughly a thousand runs, two thirds of it CI and end-to-end tests.

The founder accepted the blackout rather than raise the budget. The decision binds three consequences: "a zero-step `failure` carrying that annotation in the last days of a month is EXPECTED, not a finding; nothing is re-dispatched to work around it; the daily pulse is the instrument that reports it". It says the pulse's zero-step discriminator "should name the budget rather than a runner". At `424a58007` the pulse has no zero-step discriminator, and the one in `scripts/check-ci-headroom.cjs` (lines 557 and 610) names a refused job and runner starvation, never the budget (`grep -n budget scripts/check-ci-headroom.cjs packages/collective/src/daily-pulse.mjs` returned nothing, 2026-09-02). The prediction makes the decision falsifiable. It reads: "the next refusal lands in the last two to three days of a month

and clears on the first with no action". The falsifier follows in the same sentence: "if it lands earlier, the minutes budget has been eaten by something new and that is a finding."

The autonomy ledger cannot see the blackout: a job that never started writes no row, and the `infrastructure` and `halted` counters read 0 across all 65 rows on 2026-09-02. The heartbeat array has no entry for 2026-08-31 (`docs/org/ladder-state.json`, read 2026-09-02).

Documented lesson 41 remains open. For the zero-step job of 2026-08-10 "GitHub simply never handed the job to a runner", eleven org workflows alarm on `cancelled` alone, and "empirically no `notify-timeout` has fired once" (`docs/lessons-learned.md`, 2026-08-10). The REST API cannot separate the two mechanisms that would explain the skip. The 2026-08-31 refusal is a fifth cause of that shape, and no gate change is claimed.

A second open item sits inside the alarm. Documented lesson 61 (`docs/lessons-learned.md`, 2026-08-27) records that a test's child process wrote `RUN_OUTCOME=halted` into the live job environment, because `GITHUB_ENV` is append-only and the last write wins. For ten days every guard failure was filed as a deliberate halt. The channel is now pinned on every spawn. The `CLAUDE.md` item ends with what the fix did not reach: "Still open: the alarm names its cause from `RUN_OUTCOME` alone, a single source, when the failed STEP is independently available." Issue titles carry that source as a bracketed tag, so a wrong cause is published with the confidence of a right one.

Figure 3 has seven nodes and seven links. That is the shape Perrow (1999) says produces interactions between safety devices, and the record holds three. The scoped HALT read as global on 2026-07-23 silenced the dead-man switch (Section 6, The empirical record). Lesson 61 is the second. The third is lesson 36, in which a workflow that bypassed the canonical action replayed a failure the action had fixed (Section 4, Guardrails in tiers). The record that holds these numbers is the subject of the next section.

6 The empirical record

Everything here is observational, measured on `main` at `424a58007` on 2026-09-02 by the commands in each table caption. The organization grades itself, and says so. The autonomy ledger's caveat, in `scripts/lib/autonomy-ledger.cjs`, reads: "Small sample, and the org is grading itself. Both facts are stated here rather than smoothed. Run rows are appended by the org's workflows, not written by hand." The failure corpus carries the matching line in `public/failure-corpus.json`: "This is the operator grading its own mistakes."

6.1 The failure corpus

`public/failure-corpus.json` is generated from `docs/lessons-learned.md` and published under CC BY 4.0. On 2026-09-02 (`generated_at` 2026-09-02T10:34:39.029Z) it held 66 items, of which 42 carry a `mechanism` path. The generator checks only that the path is repo-relative and exists, never that it runs or prevents the failure (`scripts/lib/failure-corpus.cjs`, lines 99 to 113, read 2026-09-03), so mechanized here means an existing named countermeasure file. Of the 42, 13 name a guard script, 22 a test, 1 a workflow and 6 an implementation file, sorted by the rule Appendix D carries. The corpus dates 55 of the 66 items. Items 1 to 10 and item 63 are undated, and the file's caveat says why: "Early entries predate the dating convention and are published undated rather than backdated." The first documented date is 2026-06-09 and the latest is 2026-09-02.

Each item carries exactly one of three classes. The definitions below are verbatim from the file's `taxonomy` block.

- verification-gap: “A test, guard, gate, or metric that could not fail, could not see, or measured a proxy for the thing it claimed to measure.”
- operator-process: “A discipline failure in how the AI operator built or maintained the system: stale documents, unswept invariants, contracts with one side missing.”
- external-behavior: “An external platform, client, or dependency behaved differently than documented or assumed.”

Table 7: The failure corpus by class and by mechanization. Measured 2026-09-02 from `public/failure-corpus.json` by iterating `items` with `node -e` and counting `kind` and non-null `mechanism`.

Class	Items	With a mechanism	Mechanized share
verification-gap	33	23	69.7 percent
operator-process	18	11	61.1 percent
external-behavior	15	8	53.3 percent
All classes	66	42	63.6 percent

Table 8: Documented failures per ISO month and class, keyed on each item’s `date` field. Measured 2026-09-02 from `public/failure-corpus.json`; the undated row is the difference between each class total and its dated months.

Month documented	verification-gap	operator-process	external-behavior	Total
2026-06	0	2	0	2
2026-07	9	3	5	17
2026-08	21	9	5	35
2026-09	1	0	0	1
Undated	2	4	5	11
Total	33	18	15	66

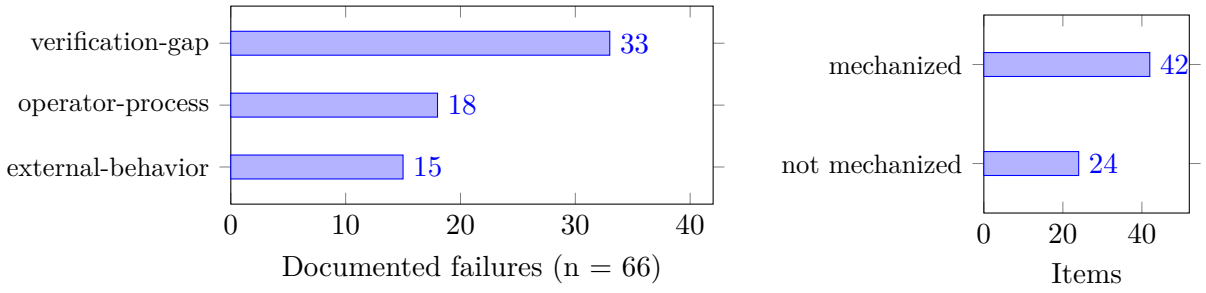

Figure 4: The failure corpus by class (left) and by whether a countermeasure path is recorded (right). Source: `public/failure-corpus.json`, 2026-09-02.

Verification gaps are 33 of 66 documented failures, 50.0 percent, the finding this paper rests on (Figure 4). Half of the recorded failures are failures of the organization’s own instruments. The share is not a comparison of instruments with agents, because the corpus has no class for agent failure; Section 8, Limitations, states what it cannot compare. It is also the company’s corpus rather than the organization’s: 23 of the 66 items concern the seat apparatus and 43 the product. Of the 23, 17 are verification gaps (`items[].kind` over the 23 ids listed in Appendix B, 2026-09-02), and Appendix B says why no share is reported for that subset. The month

table shows the corpus growing with the organization, a record of what was found rather than a trend in the defect rate.

Five items stand in for the class, each a gate whose green state carried no information: items 35, 40, 41, 44 and 47 (Appendix B). Item 40 (2026-08-10) states the shape: "A gate whose correctness rests on the thing it is gating is not a gate."

6.2 The autonomy ledger

`public/autonomy-ledger.json` aggregates `docs/org/stats/*.jsonl`, one JSON line per seat run, appended by the seat's own workflow. Its metric is defined in `scripts/lib/autonomy-ledger.cjs` and published verbatim in the file, which Appendix D quotes with the library's two invariants. A completion is a run whose output survived the gate, a refused run counts as a failure, and rows for runs that never happened are excluded as skip, halted or infrastructure.

Table 9: Seat runs from 2026-07-21 to 2026-09-01. Runs, completions, failures, discards and the rate are from `public/autonomy-ledger.json` (`generated_at` 2026-09-01T19:59:19.942Z), read 2026-09-02. Turns are the sum of `num_turns` over the 65 rows of `docs/org/stats/*.jsonl`: `cat docs/org/stats/*.jsonl | jq -s 'length as $n | (map(.num_turns) | add) as $t | {rows:$n, turns:$t}'` returned `{"rows": 65, "turns": 3190}` on 2026-09-02.

Seat	Runs	Completions	Failures	Discards	Rate (percent)	Turns
chief-of-staff	14	13	1	4	92.9	648
marketing	12	7	5	22	58.3	787
legal	8	7	1	6	87.5	321
product	8	7	1	5	87.5	448
engineering	7	6	1	6	85.7	449
quality	7	6	1	2	85.7	416
innovation	3	3	0	2	100.0	121
operations	3	3	0	0	100.0	0
revenue	3	3	0	0	100.0	0
Total	65	55	10	47	84.6	3,190

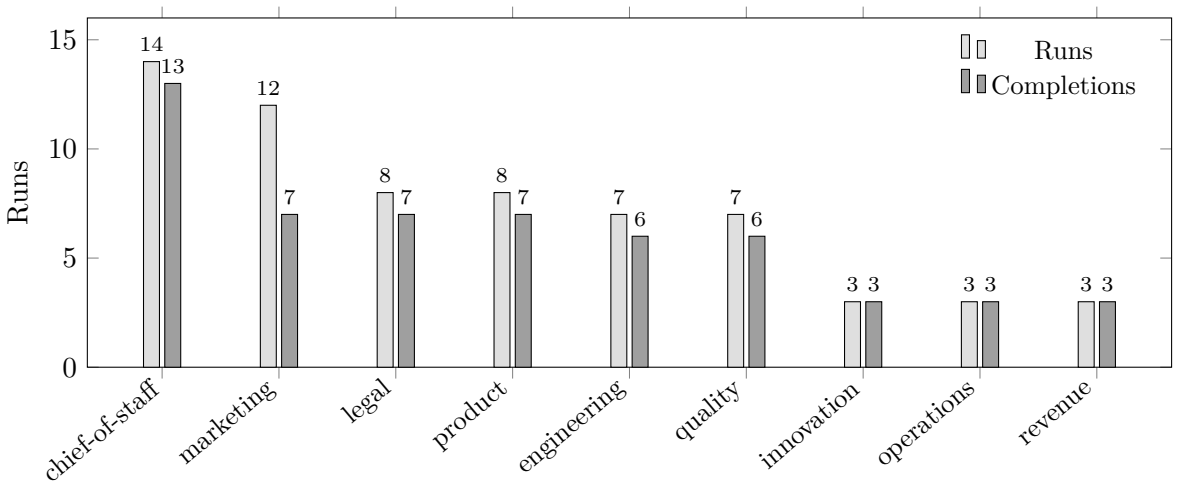

Figure 5: Runs and completions per seat, 2026-07-21 to 2026-09-01. Source: `public/autonomy-ledger.json`, read 2026-09-02.

The ledger carries a frozen baseline, never regenerated so that the series is measured against its origin, and Appendix D quotes the note. That baseline is 30 runs, 24 completions, 6 failures and 19 discards between 2026-07-21 and 2026-08-10, a completion rate of 80.0 percent, frozen on 2026-08-11. The 4.6 point difference from the current 84.6 percent comes from 35 more runs on a sample sharing its first 30 rows. The only disjoint comparison is the 35 runs after the baseline, 2026-08-11 to 2026-09-01, which hold 31 completions and 4 failures, 88.6 percent (`cat docs/org/stats/*.jsonl | jq -r 'select(.date > "2026-08-10") | .outcome' | sort | uniq -c`, 2026-09-02).

What a discard is depends on which document is read. The published metric text says "expert-panel discards are printed, never netted out" (`public/autonomy-ledger.json`), and the public ledger page uses the same vocabulary in two more places (Appendix D). The how-it-works page defines a discard as "A finished draft killed by an expert panel before anything shipped" (Appendix C). The counter measures something else. The comment above it reads: "discarded_count (item 15): everything the agent changed OUTSIDE the effective keep set + LOG is about to be reset away; count it first." (`packages/collective/actions/org-agent/action.yml`, lines 447 to 448, read 2026-09-02). The count is of `git status` paths outside that set, before the reset, the discard Section 3, The Machine: architecture of a governed seat, describes. No panel is consulted, and `grep -in panel` over the file returns one comment about the advisory panel, at line 669. We count it as a documentation-lag finding in Section 8, Limitations.

Under the counter's meaning the marketing row reads differently. The seat carries the most failures, the most discards and the lowest completion rate, with 12 runs, 5 failures and 22 of the ledger's 47 discards, 46.8 percent of discards from 18.5 percent of the runs (Figure 5). Eleven of the 22 were recorded on one run, the failure of 2026-08-26 (run 32916603490, `discarded_count: 11`, `external_input: true`; `docs/org/stats/marketing.jsonl`, read 2026-09-02), and the seat's other eleven rows carry 0, 1 or 2 each. So the 22 is one outlier plus one or two files per run, not a steady rate of refused drafts. Its content workflow, `autonomous-content.yml`, added 2026-07-12 (`git log --diff-filter=A`), writes the most public-facing artifacts, so its prompt names the most paths outside its keep-set. The ledger cannot separate a worse seat from a wider prompt.

Innovation, operations and revenue have three runs each at 100.0 percent completion, which a sample of three cannot support, and two report zero turns. The eight zero-turn rows are six completions the advisory panel wrote for those seats and two marketing failures. The panel runs two agents at up to 90 turns each but reads plain-text output, so its rows take the schema's unknown values (Appendix D). The 3,190 turns therefore undercount by those six runs.

Two columns are zero for all 65 rows, `halted` and `infrastructure`, and not for the reason the action's own comment gives. That comment calls `RUN_OUTCOME=halted` "the existing no-row path, the same one preflight uses" (`action.yml`, line 725, committed 2026-08-16); the outcome taxonomy of 2026-08-27 added an explicit `halted` case eleven days later (line 864), and nobody updated the comment. Both a preflight and a mid-run halt attempt a `halted` row. `halted: 0` means no run ever met a halt: the global HALT of 2026-08-20 lived one day and no seat ran it, the nearest rows being 2026-08-19 and 2026-08-21 (`docs/org/stats/*.jsonl`, 2026-09-03). What is structural is a job that never starts, like the budget blackout of 2026-08-31 told in Section 5, Liveness and the watch chain, which left no row and never will. That is why the dead-man switch reads run conclusions rather than the ledger.

6.3 The guard census

The guardrail population was counted one command per row, listed after the table.

Table 10: The guard census. Every value was produced on 2026-09-02 at `424a58007` by the command carrying its row number in the block below.

Row	Measure	Value
1	Guard scripts under `scripts/check-*`	110 (107 `.cjs`, 2 `.ts`, 1 `.sh`)
2	Guard scripts that read a STRICT variable	82 of the 107 `.cjs`
3	Org guards in `packages/collective/src/check-*.mjs`	11
4	Distinct `*_STRICT` escalation variables	96
5	Audit dimensions registered	91, numbered 1 to 96; 36, 49, 51, 88, 90 absent; the file's own `total` field is 91
6	Dimensions STRICT-gated in CI, by number	10 (37, 39, 40, 45, 46, 52, 53, 54, 55, 56)
7	Distinct `*_STRICT=1` names in CI, and `run:` lines that set one	22 names; 19 set (three names occur only in comments saying the flag is not set)
8	STRICT flags set on a pre-commit command	3 (`AI_GATING_STRICT`, `ORG_WIRING_STRICT`, `EMAIL_RESPONSIVE_STRICT`); 11 names match, 8 of them inside comments
9	Pre-commit blocking lines	24 lines ending in an or-exit (command 9a), plus 6 bare `exit 1` lines (9b); the hook is 567 lines; the last blocking step is `bash scripts/test-gate.sh` at line 505, the full vitest suite, and lines 507 to 567 are a nonblocking artifact heal
10	CI jobs	9, the job keys under `jobs:` in `.github/workflows/ci.yml`
11	Workflow files / scheduled / cron lines	32 / 24 / 31
12	Workflows that check `docs/org/HALT`	15
13	Guard test files	91 of 182 tracked files in `__tests__/scripts`
14	Guard tests with a stays-quiet case	91 of 91; check 6 printed `OK 91 guard tests each exercise a negative (stays-quiet) case`

```
1   ls scripts/check-* | wc -l
2   grep -l _STRICT scripts/check-*.cjs | wc -l
3   ls packages/collective/src/check-*.mjs | wc -l
4   grep -rhoE '[A-Z][A-Z0-9_]*_STRICT' scripts packages/collective/src | sort -u | wc
 ↪  -l
5   node -e 'const a=require("./app/(site)/founder/building-in-public/audit-dims.json
 ↪  ");console.log(a.dims.map(d=>d.num).sort((x,y)=>x-y).join(" "),a.total)'
6   grep -nE 'name:.*\(dim [0-9]+, strict gate\)' .github/workflows/ci.yml | wc -l
7   grep -oE '[A-Z_]+_STRICT=1' .github/workflows/ci.yml | sort -u | wc -l
7   grep -nE '_STRICT=1' .github/workflows/ci.yml | grep -vE ':\s*#' | wc -l
8   grep -nE '_STRICT=1' scripts/pre-commit
9a  grep -cE '\|\| exit 1' scripts/pre-commit
9b  grep -nE '^\s*exit 1\s*$' scripts/pre-commit
9   wc -l scripts/pre-commit
10  sed -n '/^jobs:/,$p' .github/workflows/ci.yml | grep -cE '^  [a-z-]+:$'
11  ls .github/workflows | wc -l
11  grep -l '^ *schedule:' .github/workflows/*.yml | wc -l
11  grep -c '^ *- cron:' .github/workflows/*.yml | awk -F: '{s+=$2} END{print s}'
```

```
12  grep -rl 'docs/org/HALT' .github/workflows | wc -l
13  git ls-files __tests__/scripts | grep -c check-
13  git ls-files __tests__/scripts | wc -l
14  node scripts/check-doc-claim-parity.cjs
```

Twenty of the 24 or-exit lines invoke a guard script; Appendix D records that composition with the counting caveats behind rows 6, 11 and 13. The public stats page reports 23 scheduled jobs (`stats-data.json`, field `scheduledJobs`, read 2026-09-02) against this table's 24 workflow files. We report both.

An earlier census exists in `app/(site)/research/research-registry.ts`, the field note titled "Every Guard Must Prove It Can Stay Silent", measured on 2026-08-27. Six days later the same commands return 107 `.cjs` scripts against its 105, 82 with a STRICT switch against 80, 91 guard tests against 87, 91 of 91 proving silence against 87 of 87, and 91 dimensions against 90. Its row labelled "Logged failures with a guard built in response" carries 59. The registry's second note reads that same 59 as distinct failures with 37 carrying a named guard, the lesson-53 class of a document that disagrees with itself. Neither figure is the corpus's 42 of 66: the note counts any entry naming a check script, guard or dimension, and the corpus grew by seven entries between the dates. Appendix D carries both notes verbatim.

6.4 HALT history

The kill switch is a file. `docs/org/HALT` stops every seat and `docs/org/HALT-<id>` stops one. Its entire history is five commits.

Table 11: Every commit touching a HALT file. `git log --oneline --all -- 'docs/org/HALT*'`, with dates from `git show -s --format=%ad --date=short` and identities from `git show -s --format='%an <%ae>'`, run 2026-09-02. `ls docs/org/HALT*` returned no match the same day.

Commit	Date	Author identity	Event
`3ff8a0ece`	2026-07-20	founder	`HALT-inspector-general` and `HALT-quality` created as pre-staged scoped dormancies
`6f0ab5708`	2026-07-21	founder	`HALT-quality` deleted, recording the evaluator's clearance at the Stage 1 promotion
`c8d3068c4`	2026-08-14	`org-ladder-bot <org-ladder-bot@orbytjobs.ai>`	`HALT-inspector-general` deleted by the daily ladder evaluation
`66a568b09`	2026-08-20	`push-collision-canary <org-bot@orbytjobs.ai>`	`docs/org/HALT` created: the global halt during the domain flip
`a4fadccae`	2026-08-20	`push-collision-canary <org-bot@orbytjobs.ai>`	`docs/org/HALT` deleted: the global halt lifted the same day

The two scoped files were dormancies staged before their seats had run, and the ladder cleared both rather than a person; Section 7, Design principles and a retrofit checklist, measures the stage advance behind the second. Both global HALT commits carry the identity of the push-collision canary, `org-bot@orbytjobs.ai`, the address 692 of the 737 `@orbytjobs.ai` commits share, so the repository records what a commit says, not who typed it. Appendix D records the ladder run that followed the lift, four jobs concluding `failure` with zero steps each.

The 2026-07-23 detector bug that read a scoped halt as global is not in the numbered corpus, which counts what was numbered rather than everything that was found (Appendix B). The liveness guard now records it against itself. The comment in `packages/collective/src/check-org-liveness.mjs` reads: "Treating a scoped halt as global would have silenced the whole watchdog because a single seat was deliberately dormant, which is precisely the blind spot it exists to remove."

The scoped Inspector General halt left a documentation lag. `HALT-inspector-general` was deleted on 2026-08-14 and the seat has committed a weekly report every week since, under `ig-bot` (`git log --format='%ad %an' --date=short -- docs/org/reports/inspector-general/`, 2026-09-02; Appendix D lists the four dates). `docs/org/departments.json` still carries `"status": "halted-until-stage-2"` on the seat's registry entry (line 614, read 2026-09-02), and its `authority_matrix_note` field still describes the dormancy, which `docs/org/authority-matrix.md` renders a second time (Appendix D). The public how-it-works page still answers its kill-switch question with "A single seat can be halted the same way, and one currently is." (`app/(site)/orbyt-collective/how-it-works/page.tsx`). The public process page makes the opposite error. Its kill-switch FAQ answers "It has never been used.", and its structured-data copy reads "As of August 2026 it has never been used." (`app/(site)/orbyt-collective/process/page.tsx`, lines 75, 76 and 281, last committed 2026-09-02 in `b668959cb`), against the halt of 2026-08-20 in the table above. Both errors belong to the operator-process class and are uncorrected here.

6.5 Commit footprint

The repository held 4,852 commits on 2026-09-02, of which 737 carry an author address at `@orbytjobs.ai`, the domain reserved for the organization's machine identities.

Table 12: Commits by organization identity. `git log --format='%an <%ae>' | sort | uniq -c | sort -rn`, run 2026-09-02 at `424a58007`; the total is `git log --oneline | wc -l`, 4,852.

Author identity	Commits
`push-collision-canary <org-bot@orbytjobs.ai>`	565
`org-ladder-bot <org-ladder-bot@orbytjobs.ai>`	41
`orbyt-pulse-bot <org-bot@orbytjobs.ai>`	34
`orbyt-cos-bot <org-bot@orbytjobs.ai>`	27
`orbyt-content-bot <org-bot@orbytjobs.ai>`	11
`orbyt-product-bot`, `orbyt-legal-bot` (each `<org-bot@orbytjobs.ai>`)	8 each
`orbyt-quality-bot`, `orbyt-finance-bot`, `orbyt-engineering-bot` (each `<org-bot@orbytjobs.ai>`)	7 each

Author identity	Commits
orbyt-ground-control-bot <org-bot@orbytjobs.ai>	6
orbyt-advisory-bot <org-bot@orbytjobs.ai>	4
orbyt-innovation-bot <org-bot@orbytjobs.ai>, ig-bot <org-ig-bot@orbytjobs.ai>	3 each
orbyt-machine-speaks-bot <org-bot@orbytjobs.ai>	2
orbyt-move-bot, orbyt-cmo-bot, orbyt-ack-bot (each <org-bot@orbytjobs.ai>), orbyt-engineering-bot <eng-bot@orbytjobs.ai>	1 each
All @orbytjobs.ai identities	737

Of the 737, 565 belong to the push-collision canary of Section 4, Guardrails in tiers, and carry no work. The remaining 172 are non-canary organization-identity commits. Summing the same listing, 87 carry the name of a bot mapped to a department, an officer or the advisory panel. The other 85 belong to org automation: ladder 41, pulse 34, ground control 6, machine speaks 2, move 1, acknowledgement 1. By month, org identities committed 32 times in July 2026, 682 in August and 23 in the first two days of September, of which the canary is 549 and 16, so the non-canary footprint is 32, 133 and 7. The founder's four identities account for 4,077 commits, dependabot for 7, and one other person for 2.

Two identity leaks are visible in the same listing. One is 12 commits carrying a seat's author name over the founder's GitHub address, which the `@orbytjobs.ai` filter does not count. The other is 17 commits authored by `gate <gate@test.local>`, dated 2026-08-13 and 2026-08-14, the throwaway identity the test suite's sandbox repository uses. Four consecutive production deployments came back blocked under it (`35c4c6ffe`, `288f0c392`, `19bc05cbc`, `b04126514`), and the guard's header records the cost: "production sat at df8a923ed for seven hours while the pushes all reported success." That guard, `scripts/check-commit-author.cjs`, is the second blocking line of `scripts/pre-commit`. It is placed there because Vercel blocks the deployment after the push has already reported success, while the identity lives in `.git/config`, "which every guard here is blind to by construction". The 17 commits remain in history, because rewriting them would falsify the record.

6.6 Decisions and directives

`docs/org/decision-log.md` opens: "Append-only. Every CEO decision, its rationale, its prediction (if any), and its outcome (appended on the check date)." On 2026-09-02 it held 45 entries, `grep -cE '## D-[0-9]{4}'`, from D-0001 on 2026-07-13 to D-0045 on 2026-09-02. Lines beginning `Prediction` number 14 (`grep -cE 'Prediction' docs/org/decision-log.md`, 2026-09-02). Five read `Prediction: none` and two are written `Prediction, falsifiable:`, so nine carry a falsifiable prediction, five decline one explicitly, and 31 have no `Prediction` line at all. Lines beginning `Outcome` number one, the header's own description of the field, so no outcome has been appended in the 51 days the log has existed. Four predictions name a calendar check date, and every one has passed with no outcome recorded. They are D-0009 and D-0007 of 2026-07-13, checking on 2026-07-20 and 2026-08-13, and D-0028 and D-0025 of 2026-07-20, checking on 2026-07-27 and 2026-08-19. Appendix D carries the four lines and the grep that finds them.

`docs/org/owner-directives.md` held 12 directives, `grep -cE '## OD-[0-9]{4}'`, six raised on 2026-07-29 and six on 2026-08-12. Their `Status` fields counted six `open`, two `in-progress` and four `done` (`grep -nE '- Status:' docs/org/owner-directives.md`, 2026-09-02). The pulse of 2026-09-01 (`docs/org/reports/pulse-latest.md`) reports "8 open, 2 past due", the six `open` and two `in-progress` combined, and names OD-0001 and OD-0003, both marketing, both 20 days past due. The register exists because the founder raised one report twice with no trace, and the past-due count makes a third raising unnecessary.

6.7 Token governance

The binding resource is a weekly quota rather than money, and `CLAUDE.md` records the measurement of 2026-07-25: across 35 sessions, 59.9 million output tokens against 17.07 billion cache-read tokens, a ratio the file states as 285x. The mean turn re-read 504,141 tokens of context. In the file's words, "Three sessions over 6,000 turns at ~500k context produced 60% of all consumption". We did not re-run the instrument to settle whether each of the three exceeded 6,000 turns. The largest controllable cost was image reads: 76 of them cost about 9.4 million tokens, 47 percent of all tool-result volume, about 123,660 each. The file's rule follows: "Never report token efficiency in dollars."

The weekly ceiling is empirical, because the quota is not published in tokens. The header of `packages/collective/src/token-pace.mjs`, quoted in Appendix D, calibrates on the founder's own history. It records four completed ISO weeks re-measured on 2026-07-25 at 507M, 338M, 569M and 544M, three of four within 11% of the peak, and reports them as EMPIRICAL rather than as the official cap. The hook that reads it fires on every prompt and stays silent until the projected week crosses 85 percent of that ceiling. These figures are one operator's usage on one plan over four weeks in July 2026.

Of the 65 stats rows, 49 carry a `tokens` object with values, and Appendix D accounts for the other 16.

Table 13: Token usage summed over the 49 stats rows that carry values, by seat, 2026-07-29 to 2026-09-01. `cat docs/org/stats/*.jsonl | jq -s '[.[] | select((.tokens // {}) | length > 0)] | group_by(.seat) | map({seat:.[0].seat, rows:length, out:(map(.tokens.out)|add), cr:(map(.tokens.cr)|add), cw:(map(.tokens.cw)|add), in:(map(.tokens.in)|add), turns:(map(.num_turns)|add)})'`, run 2026-09-02.

Seat	Rows with tokens	Output	Cache read	Cache write	Input	Turns
chief-of-staff	13	401,766	54,576,893	3,636,692	5,551	608
marketing	8	475,643	90,357,195	1,915,572	9,556	620
engineering	6	251,736	46,122,511	1,077,477	584	398
product	6	238,234	39,903,121	1,336,943	2,561	356
quality	6	198,396	30,362,491	964,946	2,545	362
legal	7	176,153	27,492,297	924,535	372	276
innovation	3	111,887	14,938,813	439,817	3,033	121
All seven	49	1,853,815	303,753,321	10,295,982	24,202	2,741

The mean row is 37,833 output tokens against 6,199,047 cache-read tokens, a ratio of 164 to 1, over 56 turns, and Appendix D lists the six rows that ran within five turns of the seats'

default ceiling of 80. The `tokens` object records what the CLI reported, unchecked against the provider's accounting.

6.8 What was not measured

Three absences are stated so they are not read as zero. Egress: Section 4, Guardrails in tiers, records the sandbox's network settings measured doing nothing on a hosted runner, and no egress number exists here. Customer-facing outcome: no number says whether a seat's artifact reached a customer or was read. The quality of a completion: a run counts when its output survived the gate, and nothing here grades what survived.

One measurement was taken last, after an earlier pass found the GitHub API unreachable. Twenty workflow or action files are able to file an issue, and a later pass the same day returned 49 issues, numbered 46 to 129, created between 2026-07-21 and 2026-09-01 (Appendix D holds the commands and the breakdown). We record that as a first measurement and not as a rate, because an issue is filed by a workflow that reached its failure step, so the count shares the ledger's blind spot. Section 7, Design principles and a retrofit checklist, draws from this record the rules each incident produced.

7 Design principles and a retrofit checklist

The ten principles come from the failure corpus of Section 6, The empirical record. Each names the file that carries it, which is not enforcement, so Appendix D, Mechanism inventory, grades all ten on five states: specified, implemented, negatively tested, positively observed, and unmet. The first four concern the instruments that watch the agents, the class holding 33 of the 66 failures (`public/failure-corpus.json`, 2026-09-02). The other six concern who may change what, and rest on the first four: a gate that cannot go red makes an authority boundary a fiction.

7.1 Ten principles

7.1.1 P1. Positive observation

Statement: a control counts as live only when its refusal has been observed inside the run it protects.

Incident: lesson 44, the sandbox probe that concluded success while three of its four agent runs executed nothing (Section 4, Guardrails in tiers; receipts in Appendix D). The principle reaches only a control whose refusal can be provoked in its own run; Section 8, Limitations, names those needing a drill. A run concluding success with no denial string would falsify it.

7.1.2 P2. NOT OBSERVED is a verdict

Statement: a monitor that could not look reports NOT OBSERVED, never clean, and prints why. The pulse's three blind branches, and the scopes added after each, are told in Section 5, Liveness and the watch chain.

Mechanism: the header of `packages/collective/src/daily-pulse.mjs` fixes the contract: "every input it reads has exactly three reportable states: observed-clean, observed-red, and NOT OBSERVED." A checker's always-printed `org-run-truth:` header is the ran-at-all sentinel, and a NOT OBSERVED verdict carries the failing command's first stderr line. The per-seat evaluator carries the same state as `NOT_OBSERVED`, "unjudged, because we never built its instrument. Our gap." An all-clear on a day an input did not run would falsify it.

7.1.3 P3. The exit code is the verdict, the artifact is the proof

Statement: a gate takes its verdict from the status the tool computed, run where that status can go red, and proves an effect by reading the artifact back.

Incident: the gate that read vitest's printed summary, told in Section 4, Guardrails in tiers. Lesson 46 of the same day, 2026-08-13, is its pair: report-only guards printed findings, exited 0, and read as passes.

Mechanism: the gate logic lives in `scripts/test-gate.sh` so `__tests__/scripts/test-gate.test.ts` can execute it. The artifact half is D-0040, which put `--heal` before `--check` in the commit hook, because "A healer trusted on its own report is an effect claimed rather than observed".

7.1.4 P4. Every detector ships a negative case

Statement: a detector is tested only when its suite proves it can stay quiet on a clean input. The four detectors that mis-fired on 2026-07-23, and the three rules they produced, are told in Section 4, Guardrails in tiers.

Mechanism: check 6 of `scripts/check-doc-claim-parity.cjs` scans every `__tests__/scripts/check-*.test.ts` for an absence idiom, and Section 6, The empirical record, carries its result. It warns rather than gates, because its own matcher "has been wrong twice already".

7.1.5 P5. Agents live in the sandbox, the gate is the human's

Statement: agents may create and improve report-only controls unattended. Promoting a control to a blocking gate, weakening one, or deploying is a human act.

Incident: a design decision, not a post-mortem. `docs/audit-dimensions/governance.md` states the cost: "A STRICT / pre-commit gate CAN break things. One bad threshold blocks every commit and deploy until a human notices." The file records the boundary's first exercise on 2026-06-24, when six dimensions were promoted: "the agent did the analysis and annotation; the owner made the flip."

Mechanism: what enforces the boundary differs by runtime. For scheduled seats, `scripts/pre-commit` and `.github/workflows/ci.yml` lie outside every `keep_paths` value in `docs/org/seats/*.json` (read 2026-09-02), so a seat's edit to a gate is reset at isolation. For interactive sessions nothing does: no class in `.claude/hooks/arc-valve.sh` and no `permissions.deny` entry covers those two files, so the boundary there is a sentence in a governance document. A seat-identity commit setting a `*_STRICT=1` flag would falsify it, or a lane found autonomous with no decision-log entry.

7.1.6 P6. The human-only set is enforced by the runtime, not the prompt

Statement: seven classes of act, enumerated in Section 4, Guardrails in tiers, are denied by a hook and a permission list. The refusal does not depend on the agent reading a rule, only on the agent using a command shape those patterns name. The valve is a design decision of 2026-08-23, and Appendix D carries the rule it applies.

Mechanism: `.claude/hooks/arc-valve.sh` runs on every Bash and MCP tool call, and `.claude/settings.json` carries five `permissions.deny` entries. The table below reads both statically. The valve's one recorded denial is told in Section 4, Guardrails in tiers. Whether any uncovered shape has been attempted is NOT MEASURED: the hook cannot be exercised from inside a session it governs, the probe being the act it denies.

Table 14: What the two push layers deny, read 2026-09-02 from `.claude/hooks/arc-valve.sh` and `.claude/settings.json`.

Layer	Pattern	Denies	Outside both layers
Valve, line 16	`git push` followed by `main` or `:main` (the pattern at `.claude/hooks/arc-valve.sh` line 16)	a `git push` whose text carries the token `main` after a space or a colon	a bare `git push` from a branch tracking main; `git -C <dir> push`; a `HEAD:refs/heads/main` refspec; a quoted refspec; a push inside a script invoked by name
`permissions.deny`, five entries	`git push origin main*`, `git push -u origin main*`, `git push origin HEAD:main*`, `git push --force*`, `git push -f *`	those five literal prefixes	the same five shapes

It is falsified by a `DROP TABLE` from an agent session with no human step, or by a denial class present in a prompt and absent from the hook.

7.1.7 P7. The spec is the source, the workflow is the artifact

Statement: anything generated is regenerated before it is checked, and a hand edit to an artifact is reverted, loudly, naming the file. Section 3, The Machine: architecture of a governed seat, describes the generator and the decision of 2026-08-16 that turned the layer the right way round.

Incident: lesson 62, 2026-08-27. A hand edit to a generated file was reverted by the next run, and "not one of 21,000 tests would have said a word", because every one graded the file rather than a regeneration. Only a regeneration producing a diff would falsify it.

7.1.8 P8. Judge the watched system from a source it cannot write

Statement: a liveness verdict is computed from a record the watched system cannot author.

Incident: lesson 34, found and fixed 2026-08-01. A seat run was killed at its deployed ceiling and concluded `cancelled`, unseen by the deployed alarm, while a headroom guard reading the local tree certified eleven unpushed alarm fixes as live. The corollary: "A killed job cannot write its own terminal stats row, so the org ledger under-reports exactly the runs that died; the dead-man switch therefore judges gh run conclusions, never the committed ledger."

Mechanism: `packages/collective/src/check-org-liveness.mjs` reads `gh run list --workflow <file> --json conclusion,status,createdAt,updatedAt,event` and judges a window, never `docs/org/stats/*.jsonl`. `scripts/check-ci-headroom.cjs` grades `origin/main` and names a local improvement a deploy gap. It is falsified by a headroom verdict from the working tree while `git log origin/main..main -- .github/workflows/` is non-empty.

7.1.9 P9. A decision ships its own falsifier and an ask has a place to land

Statement: a decision may carry a prediction, a prediction carries a check date, and the outcome is appended then. Every human ask receives an id, an owner, a due date and a status. The

report the founder raised twice with no trace, and the register built in answer, are told in Section 3, The Machine: architecture of a governed seat.

Mechanism: the register has a guard, `packages/collective/src/check-owner-directives.mjs`, and the log has none. No script under `scripts/check-*.cjs` or `packages/collective/src/check-*.mjs` reads the log's predictions or check dates (grep for `check_date`, `Check date` and `Prediction` over both sets, 2026-09-02). Section 6, The empirical record, measures the predictions and the four check dates that passed with no outcome. That is a missing instrument, not a young log. It is falsified by an ask raised in chat and found in no register.

7.1.10 P10. The machine may advance a pre-ratified stage and cannot widen a lane

Statement: autonomous evaluation may halt a seat, pause promotions, or activate a stage whose criteria a human ratified in advance. No mechanism moves a lane from gated to autonomous, and the record holds none. The evaluator's first version, which counted runs the agent never entered as failures, is told in Section 4, Guardrails in tiers.

Mechanism: the two evaluators differ. `packages/collective/src/evaluate-seats.mjs` has no promotion path, while `packages/collective/src/evaluate-ladder.mjs` "promotes, holds, or rolls back stages", sets `st.active_stage = "stage2"` at line 380, and did so once, activating a seat with no human commit. Timing reconciles them: the ladder activates only pre-ratified stages, and `criteria_sha256` resets every streak if they change. Section 9 of `docs/org/master-execution-plan.md` commits to "No LLM in the promotion path. The promoter stays deterministic forever." (line 980, read 2026-09-02).

The brake is the promotion pause, resting on a signature the table's first row counts. Such a signature proves possession of a configured machine key, not a human act, and D-0030 says so: any session on the founder's machine, "an agentic one included, produces liveness-qualifying commits". The brake also disagrees with its own command, standing paused on 2026-09-01 while that command, run locally the next day, found 44 qualifying commits, and `alive` has flipped five times. A brake that flaps teaches people to skip it, the alarm failure described in Section 5, Liveness and the watch chain.

Table 15: The promotion pause and the one stage advance, measured 2026-09-02 on `main` at `424a58007`.

Measurement	Result	Command or file
The evaluator's liveness command	`git log --since=14 days ago --pretty=%G? %ae main`, counting `G` or `U` with an email in `humans.json`	`evaluate-ladder.mjs` line 167
Founder-email commits verifying `G` since 2026-08-19	44	`git log --since='2026-08-19' --format='%G? %ae' | sort | uniq -c`
Of those, carrying a `Claude-Session` trailer	22	same log with `%H`, filtered to `G` and the two `humans.json` emails, then `git show -s --format=%b` on each
Seat-identity commits verifying `G` under the founder's signing principal, same window	10, all `org-bot@orbytjobs.ai`	`git log --since='2026-08-19' --format='%G? %GS <%ae>' main | grep 'G'`

Measurement	Result	Command or file
Global HALT set and lifted	66a568b09, a4fadccae; both authored push-collision-canary <org-bot@orbytjobs.ai>, both with Claude-Session: https://claude.ai/code/session_01EA1hyfwhvndL8zqRZgPsFB	git show -s --format='%an <%ae>%n%b' 66a568b09 a4fadccae
Ladder state on 2026-09-01	alive: false, promotions_paused: true, "no signed founder commit in 14 days (or signing not yet configured)"	docs/org/ladder-state.json
Flips of alive	true 2026-07-21; false then true 2026-08-15; false 2026-08-16; true 2026-08-19; false 2026-08-28	git log -p -- docs/org/ladder-state.json, filtered on the alive key
Runner verification against the local result	NOT MEASURED	
The one stage advance	c8d3068c4, 2026-08-14, org-ladder-bot: active_stage stage1 to stage2, docs/org/HALT-inspector-general deleted	git log -p -- docs/org/ladder-state.json

A stage advancing while promotions_paused is true would falsify it, as would a file under docs/org/lessons/ with no signed promotion commit.

7.2 A retrofit checklist

The thirteen moves below are ordered by cost, not by dependency, and following them in this order cost this organization at least 33 of its 65 ledger runs unfenced (Section 8, Limitations). The floor for a first unattended run is items 1, 4, 11 and 12.

Table 16: The thirteen retrofit moves, each with its "Done when" and its standing, read 2026-09-02 from docs/lessons-learned.md, CLAUDE.md, docs/org/stats/*.jsonl, docs/org/operators-manual.md and the gh commands named in the rows.

#	Move	Done when	Standing on 2026-09-02
1	One committed kill-switch file, checked before any credential	a run started with the file present exits clean having spent nothing, and its log says so	not recorded: mechanism landed, observation not recorded

#	Move	Done when	Standing on 2026-09-02
2	The human-only set in a deny list and a pre-tool hook, never a prompt, for interactive sessions; scheduled seats are refused by credential absence (item 11), and whether the hook loads inside a fenced seat run is NOT MEASURED	the hook has denied one real attempt and the denial sits in a transcript	half: one denial, the installer's self-test on 2026-08-23, not a real attempt; Section 4, Guardrails in tiers, records no real push refused by either layer
3	Every control report-only by default with a STRICT switch, plus the autonomy table	an agent has landed a report-only control unattended and every STRICT flip is in a human commit	half: the promotions of 2026-06-24 record the human half; every `/evolve-audit` row in `docs/audit-dimensions/CHANGELOG.md` landed inside a founder-invoked session, so no report-only control has landed unattended
4	Gate verdicts from exit codes, gate logic in a file a test executes	re-injecting the old defect turns the gate's own test red	observed: the re-injected regex under lesson 47
5	A stays-quiet case for every detector	a parity check prints N of N guard tests with a negative case, and one guard has been made to fire and to stay quiet on purpose	observed: `OK 91` of 91 (`node scripts/check-doc-claim-parity.cjs`)
6	Every workflow generated from a spec; heal then check in the commit hook	a hand edit to a generated workflow is reverted by the hook, which names the file	not recorded: mechanism landed, no reverted hand edit recorded
7	Decisions append-only, a prediction with a check date, an intake register with its own guard	the first outcome line is appended and the register guard has gone red on a past-due ask	half: the register guard went red on 2 past-due asks (pulse of 2026-09-01); the decision log has zero `Outcome` lines

#	Move	Done when	Standing on 2026-09-02
8	A dead-man job on the daily schedule, reading run conclusions from the platform, never the seats' ledger; on a metered plan, a monthly minutes measure, because a budget refusal stops every in-platform watcher at once (Section 5, Liveness and the watch chain)	a workflow deliberately left unrun for its window turns the job red	not recorded: mechanism landed, no deliberately unrun workflow recorded
9	A zero-agent pulse with three states and a ran-at-all sentinel from every checker it reads	killing a checker renders NOT OBSERVED with a reason rather than clean	observed: the stripped-scope proof under lesson 35
10	Grade the deployed ref, never the working tree	an unpushed workflow fix is reported as a deploy gap by name	observed: the five deploy-gap cases replayed under lesson 34
11	The agent step holds no push credential; the workflow resets the tree to the keep-set and commits with a token the agent never saw (`packages/collective/actions/org-agent/action.yml`, lines 487 to 489; the invariant is `packages/collective/src/check-agent-confinement.mjs`)	a run that writes outside its keep-set lands nothing outside it and its stats row counts the discard	NOT OBSERVED: 35 of 65 rows carry a nonzero discard count, and those counts sum to 47, but a stats row has no path field (row keys `date, seat, outcome, discarded_count, lessons_gated, num_turns, tokens, external_input, workflow, run_id`; aggregated 2026-09-02 by `node` over `docs/org/stats/*.jsonl` at `424a58007`), so the ledger cannot say whether a path outside a keep-set ever landed; the staged and committed path sets per run are not recorded

#	Move	Done when	Standing on 2026-09-02
12	Fence the filesystem, require an observed denial in every run, add a canary that calls the real fence, and write the fence's non-claims, starting with egress, in the action	the canary has gone red once on purpose	not on purpose: the canary concluded failure twice in 23 runs against 21 successes, neither red deliberate and whether either lost the fence NOT MEASURED; both are told with their run ids in Section 4, Guardrails in tiers
13	An evaluator with no model that may halt and never promote, promotion tied to a signed commit, and an external heartbeat	the quarterly heartbeat drill has been run and logged, with the absence alarm firing on both channels	not met: the drill line in `docs/org/operators-manual.md` line 86 reads "`_____`: (record result)"

Four are observed, three half met, three landed with the observation not recorded, one went red but not on purpose, one is NOT OBSERVED, one is not met. Section 8, Limitations, treats the unfired controls.

8 Limitations

We name a mitigation where one exists.

8.1 Single organization, six weeks

Everything here comes from one repository, one founder, one agent provider and one hosting platform, and nothing in the record separates the apparatus from the particular model, platform or person. The organization was founded on 2026-07-13 (`96e7bb6c6`); the ledger runs from 2026-07-21 to 2026-09-01, and the corpus was measured on 2026-09-02. That is 42 days of seat runs, exactly six weeks, inside the 51 days of the organization's existence. Platform behaviour accounts for 15 of the corpus's 66 entries (`public/failure-corpus.json`, class `external-behavior`, 2026-09-02).

8.2 The author operates the system

We designed the apparatus, operate it daily, wrote the lessons that became the corpus, and wrote this paper; four mitigations limit a bias that cannot be removed. Every number comes from a committed file or a command listed in Appendix C, though the repository is private and only its three published JSON files are open. The corpus is published in full under CC BY 4.0, so its classification can be disputed item by item, and discards are never netted out: 47 beside 55 completions. The contradictions are listed rather than reconciled (Section 6, The empirical record).

8.3 The organization grades itself

The rows behind the completion rate are appended by the organization's own workflows, and the guards that classify a run belong to it. Both published artifacts print that caveat, quoted in Section 6, The empirical record. The only reader outside the model family is the outside-family checker, whose advisory verdicts do not feed the ledger.

8.4 What the corpus can compare

The claim that half of the documented failures belong to the instruments is a share, not a comparison. The corpus has no class for a failure of an agent, so it cannot say whether instruments fail more often. The ledger counts runs, not agent failures: 10 of its 65 rows bucket against the seat, and the 47 discards are a sum of paths reset outside the keep-set and journal across 35 runs, not discarded drafts (`docs/org/stats/*.jsonl`, 2026-09-03). Neither shares a denominator with the corpus. It is also wider than the organization: 23 of its 66 items we counted as the seat apparatus and 43 the product, our selection, and we report no share for that subset (Appendix B). The selection does not hold its own rule: at least five of the 23 have product subjects, lessons 27, 43, 50, 62 and 63 (`docs/lessons-learned.md`, 2026-09-03), so it is a hand-made set a reader cannot rebuild. A single rater assigned all 66 items their class, our agent, with no second rater and no inter-rater agreement (Appendix D).

8.5 Small samples

The ledger holds 65 runs across nine seats, three of them, innovation, operations and revenue, with three runs each, all completions. A rate computed over three runs is not a rate. Per-seat failure counts range from zero to five (Section 6, The empirical record). The completion rate of 84.6 percent moves by up to 1.3 points when one run is added or removed, and the 80.0 percent frozen baseline over 30 runs by up to 2.8 points. The 4.6-point difference between them is therefore inside the movement two runs could produce (the arithmetic is in Appendix D). Nine of the twelve seats appear in the ledger, and the two observer seats' six rows carry no turn count, so seven seats produced the 3,190 recorded turns.

8.6 Controls that have never fired

The quarterly heartbeat drill in `docs/org/operators-manual.md`, procedure 8, has never been logged: its result line still read "`_______`: (record result)" on 2026-09-02. Eleven workflows alarm on `cancelled` alone and no such alarm has ever fired (Section 5, Liveness and the watch chain). The arc apparatus is hook-wired with no ledger landed, and the canary's two reds were neither on purpose nor a shown loss of the fence (Section 4, Guardrails in tiers). The outside-family checker's one scheduled caller had not run by 2026-09-02 (Section 9, Future work). Two items do not belong: the external monitor has filed issues on real absences, and only its email and SMS channels are NOT MEASURED. The promotion pause flaps and disagrees with its own command, which makes it untrusted rather than unfired (Section 7, Design principles and a retrofit checklist, P10). An unfired control is "a hope, not a control" in the operators' manual's words, and we count four: the cancelled-only alarms, the heartbeat drill, the arc apparatus and the scheduled outside check.

8.7 Documentation lag

The organization's description of itself lags its state, and the paper reports the lag rather than correcting it. Three surfaces still asserted the Inspector General's scoped halt on 2026-09-02, though the file was deleted on 2026-08-14 and the seat has reported weekly since (the row in Appendix D). A fourth, the public process page, denies the global halt of 2026-08-20; Section

6, The empirical record, quotes all four. A fifth concerns a number's meaning. The published metric text says "expert-panel discards are printed, never netted out" (`scripts/lib/autonomy-ledger.cjs`), and the public how-it-works page defines a discard as "A finished draft killed by an expert panel before anything shipped". No panel is consulted: the counter behind the 47 counts files thrown away outside the keep-set at isolation. Two smaller lags follow the same shape, a README license against decision D-0039 and 23 scheduled jobs against 24 scheduled workflow files (Section 3, The Machine: architecture of a governed seat; Section 6, The empirical record). Lesson 43 is this class, and the survey found five new instances.

8.8 What the fence does not cover

The fence confines writes and nothing else, a limit Section 4, Guardrails in tiers, records as measured on a hosted runner. A fenced seat can therefore read any host and send any payload. The only control on it is the credential set exposed to the agent step, which includes `CLAUDE_CODE_OAUTH_TOKEN` and no push credential (Section 3, The Machine: architecture of a governed seat). The valve matches no network verb, and whether it loads inside a fenced seat run is NOT MEASURED. Exfiltration of that token is a hazard no instrument in the repository observes. At least 33 of the 65 runs executed with no fence: the 32 dated before 2026-08-13, plus the engineering run of that day, which started before its workflow was wired (Appendix D). Whether the fence commit was on the branch when the other seven runs started is NOT MEASURED.

8.9 No outcome measure

Runs and commits are activity, not value: neither the ledger nor the commit footprint says whether an artifact was read, changed a decision, or reached a customer. No customer, revenue or quality outcome is claimed here. `app/(site)/about/page.tsx` says it more sharply: "A company is not proven by running. It is proven by customers, by revenue, and by surviving a year of both."

The human's own labour is unmeasured as well. The paper records no hours, no count of human interventions, and no time from an ask to a logged decision. The data for the last exists: directives carry a raised and a due date, decisions their date, so the interval is a query over committed files that has never been run. The one instrument recording the human reading is observational and never gating: 13 briefing send receipts (`docs/org/reports/briefing-receipts/`, 2026-07-24 to 2026-09-01) stand against one founder acknowledgement (`docs/org/reports/briefing-acks/2026-08-23.md`), read 2026-09-02. A reader cannot tell whether the apparatus reduced the founder's work or moved it from doing to watching.

8.10 When the human is absent

The watch chain ends at the founder, and the record supplies the case it does not analyse, the founder not reading (Section 5, Liveness and the watch chain). One thing stops, promotion: the pause holds the ladder. Everything else continues. Scheduled runs execute and their commits land on `main`: 682 seat-identity commits in August 2026, 549 of them the collision canary (Section 6, The empirical record). Report-only findings print to nobody. OD-0001 and OD-0003 were 20 days past due on the pulse of 2026-09-01, and four decisions' check dates have passed with no outcome. The ladder ran red for eight consecutive days with nobody opening the Actions tab, and the pulse reported red runs as NOT OBSERVED until its missing permission was granted. No instrument bounds how long the organization runs unread, which is evidence for Bainbridge (1983) and Green (2022) against the answer Section 2, Related work, gives them.

9 Future work

The work below is ordered by what it would settle. The first item is already committed to and overdue. The last would turn a case report into a comparison.

The organization pre-committed to a one-month test before it had run for a month. Section 7 of `docs/org/master-execution-plan.md` calls that test "pre-committed so sunk cost gets no vote" and sets five criteria. One asks that the ladder "reached Stage 3 with zero unresolved reds, OR held earlier with the failing criterion named honestly in the briefing"; another is that "The product moved". Its kill criterion reads: "if the lessons changed nothing AND the CQO surfaced nothing actioned, we shrink". Decision D-0025 fixed the test's check date at 2026-08-19, the last of four check dates that Section 6, The empirical record, measures as passed with no outcome appended. The first piece of future work is to append those four outcomes, the one-month test's first. The same section rules that "the CQO does not grade itself", which puts the adjudication with the Inspector General, the outside-family checker's opinion attached.

Stage 3 of the ladder, "the lesson protocol org-wide" in `docs/org/rollout-ladder.md`, is where seat memory begins to compound. Section 4, Guardrails in tiers, records its two unmet criteria as build debt rather than as failing measurements: neither `ig_runs` nor `ig_coverage` has an instrument. The Inspector General has produced weekly reports since 2026-08-14, and its latest ran six modules (`modules_run: 6` in `docs/org/reports/inspector-general/weekly-latest.md`, read 2026-09-02). The next build is an org-targeted module for that seat, carrying a smoke-eval scenario and a coverage manifest, so both criteria can be met on evidence rather than waived.

The heartbeat drill has never been run (Section 8, Limitations). Procedure 8 of `docs/org/operators-manual.md` describes withholding the heartbeat and confirming "the absence alarm fires on BOTH channels". Running it would show whether the email and SMS channels fire, which is NOT MEASURED. A second measurement sits beside it. The promotion pause disagrees with its own command, and its flips are measured in Section 7, Design principles and a retrofit checklist, P10. The next step is to run the evaluator's verification on the runner and on the local tree against the same commits, recording both results, so the cause of the disagreement is measured rather than guessed at.

The outside-family checker is the only reader in the organization that is not the same model family as the seats it reads. Its workflow is event-driven and advisory. The script it runs has one scheduled caller, the monthly adjudication branch of `.github/workflows/inspector-general-weekly.yml`, and that branch had not run by 2026-09-02: `docs/org/reports/outside-checks/` held one artifact, from a dispatch on 2026-07-20. Extending it to read the ledger and the corpus on a schedule would give the empirical record an independent witness. The mutual-plausibility argument in `.github/workflows/outside-check.yml` says why that witness must never be reconciled back through the agents it checks.

The real test of the ten principles is replication. Decision D-0039 chose open core with "Maximum customizability". Under it, "Seats, audits, advisors and governance policy are all user defined, and Orbyt's own set is the default rather than the requirement", and the safety floor is "the single layer that is not configurable". A second organization adopting the manifests would tell us which of the 66 failures are properties of the apparatus and which were properties of this operator, this model and this platform. The cost of a second driver was measured before this paper. Section 5.3 of `docs/collective/spec.md`, "What a second driver would cost, measured", counts four separate edits in `packages/collective/actions/org-agent/action.yml`. They are the `claude -p` invocation, the npm install of the CLI package, the hardcoded auth environment name with the run decision keyed on it, and about a hundred lines that parse the CLI's event schema. It counts four files in all, two in layers the spec calls fixed, including the

fence action that runs its own liveness probe. Until a second organization runs, the corpus is a case series.

Three smaller items follow from the record. The long-horizon kit described in Section 4, Guardrails in tiers, is installed and hook-wired with no landed ledger, and a first arc driven to a proved condition would test the turn counter, the stop gate and the valve together. Time-to-human-decision, which Section 8, Limitations, names as the missing measure of the human's own labour, is a query over committed files that has never been run. Finally, the failure corpus carries a `kind` on all 66 items and a `date` on 55 of them, and the month table in Section 6, The empirical record, is a first cut of a longitudinal view. A corpus kept for a year would show whether the verification-gap share of 33 in 66 falls as the principles take hold, or whether each new instrument adds its own blind branch.

References

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. arXiv:1606.06565. https://arxiv.org/abs/1606.06565

Anthropic. (2024). Building effective AI agents. Anthropic engineering post, 2024-12-19. https://www.anthropic.com/engineering/building-effective-agents

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., et al. (2022). Constitutional AI: Harmlessness from AI feedback. arXiv:2212.08073. https://arxiv.org/abs/2212.08073

Bainbridge, L. (1983). Ironies of automation. Automatica, 19(6), 775 to 779. https://doi.org/10.1016/0005-1098(83)90046-8

Bartak, J. (2026). Agent-native dataset design: Schema, licensing, and distribution patterns for LLM retrieval. Zenodo, deposited 2026-04-25, licensed CC BY 4.0. https://doi.org/10.5281/zenodo.19754393

Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., & Rosenthal, C. (2016). Chaos engineering. IEEE Software, 33(3), 35 to 41. https://doi.org/10.1109/MS.2016.60

Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (Eds.). (2016). Site reliability engineering: How Google runs production systems. O'Reilly Media. ISBN 9781491929124. Online edition: https://sre.google/sre-book/table-of-contents/

Casper, S., Bailey, L., Hunter, R., Ezell, C., Cabalé, E., Gerovitch, M., Slocum, S., Wei, K., Jurkovic, N., Khan, A., Christoffersen, P. J. K., Ozisik, A. P., Trivedi, R., Hadfield-Menell, D., & Kolt, N. (2025). The AI agent index. arXiv:2502.01635. https://arxiv.org/abs/2502.01635

Cemri, M., Pan, M. Z., Yang, S., Agrawal, L. A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., Zaharia, M., Gonzalez, J. E., & Stoica, I. (2025). Why do multi-agent LLM systems fail? arXiv:2503.13657. https://arxiv.org/abs/2503.13657

Chan, A., Salganik, R., Markelius, A., Pang, C., Rajkumar, N., Krasheninnikov, D., Langosco, L., He, Z., Duan, Y., Carroll, M., et al. (2023). Harms from increasingly agentic algorithmic systems. arXiv:2302.10329. https://arxiv.org/abs/2302.10329

Chan, A., Ezell, C., Kaufmann, M., Wei, K., Hammond, L., Bradley, H., Bluemke, E., Rajkumar, N., Krueger, D., Kolt, N., Heim, L., & Anderljung, M. (2024). Visibility into AI agents. arXiv:2401.13138. https://arxiv.org/abs/2401.13138

Chan, A., Wei, K., Huang, S., Rajkumar, N., Perrier, E., Lazar, S., Hadfield, G. K., & Anderljung, M. (2025). Infrastructure for AI agents. arXiv:2501.10114. https://arxiv.org/abs/2501.10114

Clymer, J., Gabrieli, N., Krueger, D., & Larsen, T. (2024). Safety cases: How to justify the safety of advanced AI systems. arXiv:2403.10462. https://arxiv.org/abs/2403.10462

DeMillo, R. A., Lipton, R. J., & Sayward, F. G. (1978). Hints on test data selection: Help for the practicing programmer. Computer, 11(4), 34 to 41. https://doi.org/10.1109/C-M.1978.218136

Dong, L., Lu, Q., & Zhu, L. (2024). AgentOps: Enabling observability of LLM agents. arXiv:2411.05285. https://arxiv.org/abs/2411.05285

European Parliament and Council. (2024). Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). Official Journal of the European Union, L, 2024/1689, 12.7.2024. https://eur-lex.europa.eu/eli/reg/2024/1689/oj

Fourney, A., Bansal, G., Mozannar, H., Tan, C., Salinas, E., Zhu, E., Niedtner, F., Proebsting, G., Bassman, G., Gerrits, J., Alber, J., Chang, P., Loynd, R., West, R., Dibia, V., Awadallah, A., Kamar, E., Hosn, R., & Amershi, S. (2024). Magentic-One: A generalist multi-agent system for solving complex tasks. arXiv:2411.04468. https://arxiv.org/abs/2411.04468

Greenblatt, R., Shlegeris, B., Sachan, K., & Roger, F. (2023). AI control: Improving safety despite intentional subversion. arXiv:2312.06942. https://arxiv.org/abs/2312.06942

Green, B. (2022). The flaws of policies requiring human oversight of government algorithms. Computer Law & Security Review, 45, 105681. https://doi.org/10.1016/j.clsr.2022.105681

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., & Schmidhuber, J. (2023). MetaGPT: Meta programming for a multi-agent collaborative framework. arXiv:2308.00352. https://arxiv.org/abs/2308.00352

Kapoor, S., Stroebl, B., Siegel, Z. S., Nadgir, N., & Narayanan, A. (2024). AI agents that matter. arXiv:2407.01502. https://arxiv.org/abs/2407.01502

Kolt, N. (2025). Governing AI agents. arXiv:2501.07913. https://arxiv.org/abs/2501.07913

Leveson, N. G. (2012). Engineering a safer world. The MIT Press. https://doi.org/10.7551/mitpress/8179.001.0001

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., & Ghanem, B. (2023). CAMEL: Communicative agents for "mind" exploration of large language model society. arXiv:2303.17760. https://arxiv.org/abs/2303.17760

Mitchell, M., Ghosh, A., Luccioni, A. S., & Pistilli, G. (2025). Fully autonomous AI agents should not be developed. arXiv:2502.02649. https://arxiv.org/abs/2502.02649

Model Context Protocol. (2025). Model Context Protocol specification, revision 2025-06-18. https://modelcontextprotocol.io/specification/2025-06-18

National Institute of Standards and Technology. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1, January 2023. https://doi.org/10.6028/NIST.AI.100-1

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. arXiv:2304.03442. https://arxiv.org/abs/2304.03442

Perrow, C. (1999). Normal accidents: Living with high risk technologies, updated edition. Princeton University Press. ISBN 9780691004129.

Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., & Sun, M. (2023). ChatDev: Communicative agents for software development. arXiv:2307.07924. https://arxiv.org/abs/2307.07924

Reason, J. (1990). Human error. Cambridge University Press. https://doi.org/10.1017/CBO9781139062367

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. arXiv:2302.04761. https://arxiv.org/abs/2302.04761

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J. F., & Dennison, D. (2015). Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems 28 (NIPS 2015). https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html

Shavit, Y., Agarwal, S., Brundage, M., Adler, S., O'Keefe, C., Campbell, R., Lee, T., Mishkin, P., Eloundou, T., Hickey, A., Slama, K., Ahmad, L., McMillan, P., Beutel, A., Passos, A., & Robinson, D. G. (2023). Practices for governing agentic AI systems. OpenAI white paper. https://cdn.openai.com/papers/practices-for-governing-agentic-ai-systems.pdf

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., et al. (2024). OpenHands: An open platform for AI software developers as generalist agents. arXiv:2407.16741. https://arxiv.org/abs/2407.16741

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. arXiv:2308.08155. https://arxiv.org/abs/2308.08155

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., & Press, O. (2024). SWE-agent: Agent-computer interfaces enable automated software engineering. arXiv:2405.15793. https://arxiv.org/abs/2405.15793

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). ReAct: Synergizing reasoning and acting in language models. arXiv:2210.03629. https://arxiv.org/abs/2210.03629

A Guard census

This appendix lists the controls that Section 4, Guardrails in tiers, describes in aggregate. We count two populations. The first is the set of guards that belong to the organization itself, the eleven scripts under `packages/collective/src/check-*.mjs` (11 files by `ls packages/collective/src/check-*.mjs`, 2026-09-02). The second is the set of audit dimensions that continuous integration escalates from report-only to blocking by dimension number, ten of them (`.github/workflows/ci.yml`, 2026-09-02). The wider product gauntlet, 110 scripts under `scripts/check-*` (107 `.cjs`, 2 `.ts`, 1 `.sh`, by `ls scripts/check-* | wc -l`, 2026-09-02), is out of scope here except where a numbered dimension draws on it.

Three counts frame the tables. Across `scripts` and `packages/collective/src`, 96 distinct `*_STRICT` escalation variables exist (`grep -rhoE '[A-Z][A-Z0-9_]*_STRICT' scripts packages/collective/src | sort -u | wc -l`, 2026-09-02). In `.github/workflows/ci.yml`, 22 distinct `*_STRICT=1` names appear (`grep -oE '[A-Z_]+_STRICT=1' .github/`

workflows/ci.yml | sort -u | wc -l, 2026-09-02). We read the file and found that three of the 22 (FINANCE_STRICT, ORG_INTERFACE_STRICT, BOUNDARIES_STRICT) occur only in comments that say the flag is not set, so 19 run: lines set a STRICT flag (2026-09-02). The pre-commit hook sets three (AI_GATING_STRICT, ORG_WIRING_STRICT, EMAIL_RESPONSIVE_STRICT, at scripts/pre-commit lines 360, 389 and 448, 2026-09-02). Every one of these numbers is a count of names in files, not a measurement of whether a gate has ever gone red. Section 6, The empirical record, treats that distinction as the central finding.

A.1 The org guards

Each of the eleven guards carries a header comment that states the incident it answers and the limit of what it claims. We describe each guard from that header, read with sed -n 1,40p on 2026-09-02, and we quote the header where the wording is load-bearing. The two guards most often cited in the body carry their own stated reason. check-org-liveness.mjs says of its design: "A red scheduled run IS the alarm." check-org-run-truth.mjs says: "The founder was the monitoring. That is the defect this file closes." Two headers also state a non-claim we repeat here because the body relies on it. check-agent-confinement.mjs says: "The fence is FILESYSTEM ONLY. Egress is not fenceable on a hosted runner and the fence's own output says so on every run." check-owner-directives.mjs says: "A guard that editorializes on quality gets muted, and a muted guard is worse than none."

Table 17: The eleven org guards, each described from its own header comment (sed -n 1,40p on each file, plus lines 274 to 281 of the liveness guard for the scoped halt). Source: packages/collective/src/check-*.mjs, read 2026-09-02.

Guard	What it checks	Origin stated in the header
check-advisor-name-leak.mjs	Parses the lens names from the table rows of docs/org/board-of-advisors.md and scans app/(site)/, public/ and packages/email/src/ for a ratified name inside advisory framing within three lines; a bare editorial mention counts as INFO, not a finding	Rule 5 of the board file; the bare-name version measured 50 hits across 10 files on 2026-08-01, every one legitimate
check-agent-confinement.mjs	Every workflow that executes an agent and checks out the repository must run the fence and check out with persist-credentials: false; matches against comment-stripped YAML; proves confinement of writes only	Three workflows bypass the seat generator, so the invariant is not true by construction
check-audit-liveness.mjs	Runs every node scripts/check-*.cjs command that audit.md registers as a dimension and requires that it produce a verdict, not merely exit 0	2026-08-05: three dimensions produced zero bytes and exited 0, one printing an affirmative all-clear on the credential surface
check-eval-freshness.mjs	An eval artifact for a live seat is stale when its recorded sources_sha256 no longer matches the current hash of scenario, prompt and live lessons, or when its date predates the seat's latest completed run; freshness is never wall-clock; report-only, EVAL_FRESHNESS_STRICT=1 escalates	Master execution plan item 18

Guard	What it checks	Origin stated in the header
check-org-dependency-order.mjs	Makes seat dependencies data (the registry's depends_on and officers) and checks the cron clock against them; only declared edges are checked; monthly and quarterly seats report UNORDERABLE rather than pass; report-only, ORG_DEP_ORDER_STRICT=1 blocks	2026-07-24: the weekly cron order was a topological sort enforced by nothing
check-org-interface.mjs	interface_version is real semver and no department declares a version newer than the registry; conformance fields are additive-only against a committed snapshot; permanent ids resolve against a committed snapshot, while the decision-log reference check beside it is declared and then discarded with void refRe, so it checks nothing; a prediction not flagged unmeasurable carries a check_date in a department proposal file; report-only, ORG_INTERFACE_STRICT=1 escalates	Durability findings H1 and H2: the never-a-rewrite promise had no enforcement
check-org-liveness.mjs	The dead-man switch: did each seat run and succeed, checked daily with no agent, no token, no commit; honors the global HALT and treats a scoped HALT-<id> as one seat's deliberate dormancy	2026-07-23: the Chief of Staff briefing failed on 2026-07-17 and stayed failed six days
check-org-memory.mjs	Every seat has a live lessons file and a proposed file, the live file under 150 lines; a staged commit that modifies a live lessons file blocks unless ORG_LESSON_PROMOTION=1; no keep-paths include a live lessons path; each prompt loads Tier 1 and its own lessons only; no email-like or phone-like strings in org memory	D-0022 and the ratified memory law
check-org-run-truth.mjs	Five checks: green-but-empty, never-ran, silent-department, label-exists, readable-output	2026-07-29: a seat went green while its reports directory was eight days old, a quarterly workflow had zero runs ever, and the CI alarm died on a missing label
check-org-wiring.mjs	BLOCK class: one-recurring-emailer, emails-via-renderer, halt-honored, concurrency-serialized; WARN class: agent-no-push-cred, sentinel-in-prompt, registry-on-disk; report-only, ORG_WIRING_STRICT=1 runs in pre-commit and CI	The 2026-07-14 audit: a legacy per-department email reached the founder because nothing looked at the wiring between agents
check-owner-directives.mjs	Five checks on the intake register: well-formed, owner-exists, overdue, dropped-reason, wiring; report-only, OWNER_DIRECTIVES_STRICT=1 escalates	2026-07-29: the founder raised one report twice over several weeks and it left no trace

A.2 The dimensions gated STRICT by number

The `/audit` roster registers 91 dimensions, numbered within 1 to 96 with 36, 49, 51, 88 and 90 absent (`.claude/commands/audit.md`, 2026-09-02). Report-only is the default posture for the durability range, and a dimension blocks only when a workflow sets its `*_STRICT=1` variable. Ten steps in `.github/workflows/ci.yml` name a dimension number in their step name and set its flag (`grep -nE 'name:.*\(dim [0-9]+, strict gate\)' .github/workflows/ci.yml | wc -l` returns 10, 2026-09-02). An eleventh step names a dimension and blocks without a STRICT variable, "Secret history scan (dim 42, real-secret gate)" at line 222; it is outside this table because the grep that defines the table requires the words `strict gate`. The file's own comments date the promotions. Dimension 37 was "promoted to a STRICT gate 2026-06-24" (line 325). Dimensions 39, 40, 45 and 46 were "promoted to STRICT gates 2026-06-24" (line 330). Dimensions 52 to 56 were "promoted to STRICT gates 2026-07-21 (CEO sign-off, the grand-review move 4)" (lines 343 to 344). The remaining nine `run:` lines that set a STRICT flag escalate a guard by name rather than by dimension number. Two of those nine belong to org guards listed above: `ORG_WIRING_STRICT` on line 241 and `CONFINEMENT_STRICT` on line 312 (2026-09-02). The other nine org guards run report-only in CI or do not run in CI at all; we did not trace each one's invocation and say so rather than claim it.

Table 18: The ten audit dimensions that CI escalates by number, with the script each runs and the check summarized from its roster entry. Sources: `.claude/commands/audit.md` (entries at lines 344 to 364) and `.github/workflows/ci.yml` (lines 325 to 358), read 2026-09-02.

Dim	Script	What it checks	Flag
37	`scripts/check-authz.cjs`	Tenant isolation: every `user_id` or `team_id` table has RLS, no fail-open service-role SELECT, no bulk credential exposure, and the fail-closed tier resolver never regresses to a default; `--live` seeds two tenants and asserts cross-tenant DENY	`AUTHZ_STRICT=1`
39	`scripts/check-cost.cjs`	Metered-resource anomaly shapes from mistake 11: `dynamicParams=true` matrix routes without a kept-set, sub-3600 `revalidate` on crawlable pages, sitemap versus kept-set drift, unbounded crawlable `[param]` routes	`COST_STRICT=1`
40	`scripts/check-erasure.cjs`	Right to erasure: every `user_id` table cascades, sets null, is purged by a retention cron or carries an allow annotation; Stripe-customer and localStorage cleanup; the purge crons are scheduled	`ERASURE_STRICT=1`
45	`scripts/check-webhook-resilience.cjs`	Every webhook verifies its signature before side effects, is idempotent against replay, acknowledges unknown event types without throwing, and keeps no module-scoped mode across an await	`WEBHOOK_STRICT=1`
46	`scripts/check-chaos.cjs`	Every external seam (Redis, Supabase, Stripe, Resend) fails the safe way: security and cost gates deny on outage, user experience degrades	`CHAOS_STRICT=1`
52	`scripts/check-cron-continuity.cjs`	The Vercel cron fleet as one contract: `vercel.json` to route parity, constant-time secret verification, run tracking, and a `maxDuration` ceiling on long jobs	`CRON_CONTINUITY_STRICT=1`
53	`scripts/check-stripe-boundary.cjs`	No direct `STRIPE_PRICE_*` or `_SECRET_*` read outside `_lib/stripe.ts`, no deprecated bare `getStripe()`, and every money-mutating `stripe.*.create()` carries an `idempotencyKey`	`STRIPE_BOUNDARY_STRICT=1`

Dim	Script	What it checks	Flag
54	scripts/check-external-cost.cjs	Every external outbound fetch() carries an abort signal or timeout, and every image route reading searchParams carries a cache header or a rate limit	EXTERNAL_COST_STRICT=1
55	scripts/check-privacy-manifest.cjs	Every shipping iOS target declares the required-reason API categories it uses in a PrivacyInfo.xcprivacy; skips cleanly when ios/ is absent	PRIVACY_MANIFEST_STRICT=1
56	scripts/check-ai-metering.cjs	Every route that can reach a model provider records its spend; the subject test was widened on 2026-08-31 after the old test saw 21 of 46 provider-reaching routes and none of the 13 unmetered ones while reporting grade A	AI_METERING_STRICT=1

B The org lessons

The failure corpus is public/failure-corpus.json, generated from docs/lessons-learned.md on 2026-09-02 (generated_at 2026-09-02T10:34:39.029Z). It holds 66 items, 42 with a recorded mechanism, 55 with a date, the earliest dated 2026-06-09 and the latest 2026-09-02 (count, mechanized_count, dated_count, first_documented, latest_documented, read with node). Its taxonomy has three classes, defined in the file as follows. A verification-gap is "A test, guard, gate, or metric that could not fail, could not see, or measured a proxy for the thing it claimed to measure." An operator-process failure is "A discipline failure in how the AI operator built or maintained the system: stale documents, unswept invariants, contracts with one side missing." An external-behavior failure is "An external platform, client, or dependency behaved differently than documented or assumed." The class counts are 33, 18 and 15 (taxonomy.*.count, 2026-09-02).

The body of this paper draws on 23 of the 66, the ones whose subject is the organization's own instruments rather than the product. The table below lists them. The 23 were selected by us as the lessons whose subject is a seat, its workflow, the shared action, the fence, the runners, or the monitors that watch them; the corpus carries no field for that distinction, so the selection is a judgement and not a filter a reader can re-run mechanically, and Section 8, Limitations, says so. Within the 23, 17 are verification gaps (items[].kind, 2026-09-02). We report no share against the whole corpus: the 23 are our selection, no reader can rebuild it, and a real concentration cannot be separated from an artefact of the selection. The class, date and mechanism columns come from the corpus item (items[].kind, items[].date, items[].mechanism, read with node on 2026-09-02). The summary column is ours, condensed from the entry at docs/lessons-learned.md found by grep -n "<number>\. ". Where the corpus records no mechanism we say so, because a lesson without a mechanism is a hope in the sense the codebase uses: check-audit-liveness.mjs states the rule as "A lesson is a hope. A guard is a fact." Of the 23, the corpus classes 17 as verification-gap, 4 as operator-process and 2 as external-behavior, and records a mechanism for 18. One item, lesson 63, carries date: null in the corpus and no date in its own entry; we report it as undated rather than infer one.

Table 19: The 23 org lessons the paper draws on, plus the unnumbered 2026-07-23 detector bug. Sources: `docs/lessons-learned.md` (summaries) and `public/failure-corpus.json` (dates, classes, mechanisms; `generated_at` 2026-09-02T10:34:39.029Z); the final row is from `CLAUDE.md`, "The three rules that survive the session", dated 2026-07-23. Read 2026-09-02.

No.	Date	Class	Summary	Mechanism
20	2026-07-25	verification-gap	A conditional CI step, never forced true, failed on the one day it mattered	None in the corpus; the fix was a heap-size flag on that step
26	2026-07-31	verification-gap	Stryker could not start for 81 days; a quarterly tool has no gate between quarters	`__tests__/config/stryker-config-sync.test.ts`
27	2026-07-31	verification-gap	Ingestion cron reported success at 98.4 percent of its 600 second ceiling; killed runs stayed running	None in the corpus; the lesson text calls the `cron_runs` reaper owner-pending, but `reapAbandonedRuns` has been in `app/api/_lib/cron-tracker.ts` since `14c24b6e7` (Section 5, Liveness and the watch chain)
28	2026-07-31	verification-gap	A timeout concludes cancelled, not failed; seven of twelve builds died unalarmed	`scripts/check-ci-headroom.cjs`
29	2026-07-31	verification-gap	GEO citation probe ran green three times weekly; its committed history could never exceed two points	None in the corpus; carry-forward plus an artifact-only declaration in the workflow
31	2026-08-01	verification-gap	A 403 behind `2>/dev/null` read as no previous run, so carry-forward never worked once	`__tests__/scripts/daily-pulse.test.ts`
34	2026-08-01	verification-gap	Eleven unpushed alarms were graded covered while a seat died at the deployed ceiling	`scripts/check-ci-headroom.cjs` grades the deployed ref

No.	Date	Class	Summary	Mechanism
35	2026-08-02	verification-gap	The daily pulse had three blind branches: a crashed checker, a starved scope and an emptied window rendered clean	`__tests__/scripts/daily-pulse.test.ts`; missing header renders NOT OBSERVED
36	2026-08-02	operator-process	Three workflows hand-roll the shared action's guards; the advisory panel replayed the em-dash death 13 days later	`__tests__/scripts/org-guard-parity.test.ts`
40	2026-08-10	verification-gap	The completion gate read `tail -c 600`; a finished legal run's sentinel sat 1,360 bytes from the end	`packages/collective/actions/org-agent/action.yml`; the window is now 25 lines
41	2026-08-10	verification-gap	A job cancelled with zero steps was never acquired by a runner; eleven alarms cannot see it	`scripts/check-ci-headroom.cjs`; step count discriminates, the alarm gap stays open
42	2026-08-10	verification-gap	The pulse printed GEO citations undated for days while the workflow succeeded three times weekly	`__tests__/scripts/daily-pulse.test.ts`; monitors resolve from the last successful run
43	2026-08-10	operator-process	Eight stale root documents, referenced by nothing mechanical, asserted a retired pricing model as instructions	`scripts/check-root-clutter.cjs`
44	2026-08-13	verification-gap	A sandbox that could not start concluded success with no permission denials; three of four probes never ran	`.github/workflows/fence-canary.yml`; liveness by observed denial

No.	Date	Class	Summary	Mechanism
45	2026-08-13	operator-process	A lane is what the prompt tells the agent to write, not keep-paths; deny must walk every level	`__tests__/scripts/org-fence.test.ts`; deny list regenerated from the tree
46	2026-08-13	operator-process	Report-only guards print findings and exit 0; a plain run read PASS while holding a BLOCK	None in the corpus; run with the STRICT flag and read the output
47	2026-08-13	verification-gap	The pre-commit suite gate grepped vitest's summary; ANSI colour in CI left it with no live branch	`__tests__/scripts/test-gate.test.ts`; the exit code is the verdict
49	2026-08-15	verification-gap	`stats-data.json` counts the repository's own commits, so it is stale at the instant it is committed	`scripts/check-generated-drift.cjs`; pre-commit regenerates and heals
50	2026-08-20	external-behavior	A shallow clone returns HEAD's date for every path, so the skunkworks page claimed updated today	`scripts/update-skunkworks-dates.cjs`; writes nothing on a shallow clone
61	2026-08-27	verification-gap	A test's child wrote `RUN_OUTCOME=halted` into the live job env; ten days of guard failures filed as halted	`__tests__/scripts/runner-env-isolation.test.ts`
62	2026-08-27	verification-gap	A hand-edited heading in a generated file reverted next run; the suite graded the file, not a regeneration	`__tests__/scripts/untimed-heading.test.ts`; normalization at the generator joint
63	undated	verification-gap	The publish gate's guard ran at 01:00 against an 08:00 gate; deleting the fix left it green	`__tests__/site/scheduled-content-containment.test.ts`; injected clock plus a counterfactual

No.	Date	Class	Summary	Mechanism
65	2026-08-30	external-behavior	`console.log` colourised a build number under `FORCE_COLOR=3`; the bump script emitted invalid JSON and exited 0	None in the corpus; capture via `process.stdout.write(String(x))`
unnumbered	2026-07-23	not in the corpus	A scoped `HALT-<id>` was read as global, so one dormant seat silenced the whole watchdog	`scopedHalts()` in `packages/collective/src/check-org-liveness.mjs` separates the two shapes

The final row is not in the corpus because it was never given a number in `docs/lessons-learned.md`. It is recorded in `CLAUDE.md` under "The three rules that survive the session (MANDATORY, added 2026-07-23)", as one of four detectors that mis-fired that day: "a scoped `HALT-<id>` was read as global so one dormant seat silenced the whole watchdog". The liveness guard now carries the distinction in its own comment: "`docs/org/HALT` stops EVERYTHING, while a scoped `docs/org/HALT-<id>` stops ONE seat." We include the row because it is the earliest recorded instance of the pattern Section 5, Liveness and the watch chain, names as the watcher failing in the direction of silence, and because its exclusion from the corpus is itself a measurement of the corpus: the corpus counts what was numbered, not everything that was found.

C Vocabulary and reproduction

C.1 Vocabulary

A reader outside software needs eight terms the paper uses without definition. A *commit* is a recorded change to the repository with an author and a message. *main* is the branch that deploys production on every push. A *push* sends commits to the shared repository. A *pull request* is a proposed set of commits reviewed before it lands. A *workflow* is a scheduled or triggered job that GitHub runs on a *runner*, a fresh machine. A *pre-commit hook* is a script that runs before a commit is recorded and can refuse it. An *exit code* is the number a program returns, zero meaning success. A *STRICT variable* is the environment setting that turns a report-only guard into one that blocks.

The ten public terms below are reproduced verbatim from the `VOCABULARY` array in `app/(site)/orbyt-collective/how-it-works/page.tsx`, version 1.1 of that page (its changelog dates the vocabulary to 2026-08-12), read 2026-09-02.

Table 20: The ten public terms, verbatim from the `VOCABULARY` array in `app/(site)/orbyt-collective/how-it-works/page.tsx`, read 2026-09-02.

Term	Definition
Agent seat	A named AI agent with a written charter, a lane of files it may write, and a line it cannot cross without a human.

Term	Definition
Authority matrix	The committed table of every seat's autonomous and gated lanes, enforced by the workflows that run the org rather than by instructions to the model.
Autonomy Ledger	Orbyt Labs' published run record for its agent seats: completions, failures and review-panel discards, with a frozen baseline and raw JSON at a stable URL.
Decision log	The append-only record of structural decisions, each with a permanent id. Corrections are new entries that name what they supersede.
Decision Ledger	The published dataset built from the decision log: every decision, numbered and dated, downloadable as raw JSON under CC BY 4.0.
Failure Corpus	The published dataset of documented failure modes from the AI-operated codebase, each classified, and where a countermeasure exists, linked to it.
Completion gate	The sentinel check that reads the tail of an agent's output for its completion line. A run without the line is treated as truncated and lands nothing.
Discard	A finished draft killed by an expert panel before anything shipped. Counted separately from failures, because work that finished and was refused is a different fact from work that never finished.
Kill switch	One committed file that stops every autonomous seat before it spends anything, checked ahead of the credential.
Dead-man switch	The reverse brake: if no signed human commit lands for fourteen days, promotions pause where they are.

Two of the ten public terms differ from this paper's usage, and we say which is which. The public page's "Dead-man switch" is the signed-commit promotion pause, which this paper calls the promotion pause or the reverse brake; this paper's dead-man switch is the daily liveness job, `packages/collective/src/check-org-liveness.mjs`, and the quarterly withholding of the heartbeat is the heartbeat drill. The public page's "Discard" is a draft killed by an expert panel; the counter that produces the published discard count measures out-of-keep-set paths reset by isolation, and Section 6, The empirical record, reports the disagreement without resolving it.

Eight further terms are used in this paper and are not on the public page. A *lane* is the set of repository paths a seat may write, derived from what the seat's prompt tells it to write rather than from what survives to be committed. The *keep-set* is the set of paths that survive isolation of a run's working tree to be committed by the workflow. A *sentinel* is the completion line the gate reads from the tail of an agent's output. The *valve* is the PreToolUse hook, `.claude/hooks/arc-valve.sh`, that denies the human-only acts on Bash and every MCP tool in every session. The *fence* is the machine-level filesystem sandbox installed by

`packages/collective/actions/org-fence/action.yml`, which fails red unless it observes a real denial. The *pulse* is the daily zero-agent report produced by `packages/collective/src/daily-pulse.mjs` that says what is open and what is red. The *ladder* is the five-stage promotion ratchet recorded in `docs/org/rollout-ladder.md` and `docs/org/ladder-state.json`. Its seat evaluator, `packages/collective/src/evaluate-seats.mjs`, may halt a seat and has no promotion path. Its ladder evaluator, `packages/collective/src/evaluate-ladder.mjs`, holds when promotions are paused and otherwise activates a pre-ratified stage, as it did once on 2026-08-14 in commit `c8d3068c4`. *NOT OBSERVED* is the third verdict state, neither clean nor red, that a monitor reports when it could not look.

C.2 Reproduction

The commands below regenerate the counted figures in Section 6, The empirical record, with one exception we name at the end of this appendix: the token-governance figures, which no command reproduces. They were run on 2026-09-02 from the root of the Orbyt repository on the measurement machine, on `main` at commit `424a58007`. The repository was private on that date, so a reader without access can re-run only the commands that read the three published JSON files named at the end of this appendix; the rest are reproducible by anyone the founder admits to the repository, and by no one else (Section 8, Limitations). Each block is introduced by the table or figure it feeds. The three commands that read GitHub require an authenticated `gh` client; every other command reads committed files. One of the three, the `gh run view` lookup, reads the creation time of a single finished run, which does not change; we re-ran it on 2026-09-03 and it returned the same value recorded in the comment beneath it. The other two read GitHub's live issue and pull-request lists, which change after our reading date, so we date every figure drawn from them and do not claim a later reader sees the same rows.

The failure corpus by class and mechanization, and the per-month table:

```
node -e 'const c=require("./public/failure-corpus.json");const k={};for(const i of
 ↪  c.items){k[i.kind]=k[i.kind]||{items:0,mechanized:0};k[i.kind].items++;if(i.mecha
 ↪  nism)k[i.kind].mechanized++}console.log(c.items.length,JSON.stringify(k))'
node -e 'const c=require("./public/failure-corpus.json");const m={};for(const i of
 ↪  c.items){const mo=i.date?i.date.slice(0,7):"undated";m[mo]=m[mo]||{};m[mo][i.kind
 ↪  ]=(m[mo][i.kind]||0)+1}console.log(JSON.stringify(m))'
```

The autonomy ledger (current totals, the frozen baseline, per-seat rows) and the turn count over the stats rows:

```
node -e 'const l=require("./public/autonomy-ledger.json");console.log(JSON.stringify(
 ↪  l.current.totals));console.log(JSON.stringify(l.baseline.totals));console.log(JSO
 ↪  N.stringify(l.current.perSeat))'
cat docs/org/stats/*.jsonl | jq -s 'length as $n | (map(.num_turns) | add) as $t |
 ↪  {rows:$n, turns:$t}'
```

The per-seat turn column, the eight zero-turn rows, the per-seat token table, the count of runs dated before the fence, the commit that created the fence action, and the per-workflow comparison that resolves the runs dated 2026-08-13 (only the last of these reads GitHub):

```
cat docs/org/stats/*.jsonl | jq -s 'group_by(.seat) | map({seat:.[0].seat,
 ↪  turns:(map(.num_turns)|add)})'
cat docs/org/stats/*.jsonl | jq -c 'select(.num_turns==0) |
 ↪  {date,seat,outcome,workflow}'
cat docs/org/stats/*.jsonl | jq -s '[.[] | select((.tokens // {}) | length > 0)] |
 ↪  group_by(.seat) | map({seat:.[0].seat, rows:length, out:(map(.tokens.out)|add),
 ↪  cr:(map(.tokens.cr)|add), cw:(map(.tokens.cw)|add), in:(map(.tokens.in)|add),
 ↪  turns:(map(.num_turns)|add)})'
```

```
cat docs/org/stats/*.jsonl | jq -r .date | sort | awk '{if($1<"2026-08-13")a++; else
 ↪  b++} END{print a, b}'
git log --diff-filter=A --format='%h %ad' --date=short --
 ↪  .github/actions/org-fence/action.yml
git log --reverse --format='%h %ad' --date='format:%Y-%m-%d %H:%M %z' -S'org-fence' --
 ↪  .github/workflows/engineering-weekly.yml | head -1
# printed 2026-09-02: 1efeadb15 2026-08-13 19:59 -0500
cat docs/org/stats/*.jsonl | jq -r 'select(.date=="2026-08-13") | [.run_id,.workflow]
 ↪  | @tsv'
gh run view 31688969118 --json createdAt
# printed 2026-09-02: {"createdAt":"2026-08-13T09:57:54Z"}
```

The guard census, one command per row of the table in Section 6, The empirical record:

```
ls scripts/check-* | wc -l
grep -l _STRICT scripts/check-*.cjs | wc -l
ls packages/collective/src/check-*.mjs | wc -l
grep -rhoE '[A-Z][A-Z0-9_]*_STRICT' scripts packages/collective/src | sort -u | wc -l
node -e 'const
 ↪  a=require("./app/(site)/founder/building-in-public/audit-dims.json");const
 ↪  n=a.dims.map(d=>d.num).sort((x,y)=>x-y);console.log(n.length,a.total,n[0],n[n.len
 ↪  gth-1])'
grep -nE 'name:.*\(dim [0-9]+, strict gate\)' .github/workflows/ci.yml | wc -l
grep -oE '[A-Z_]+_STRICT=1' .github/workflows/ci.yml | sort -u | wc -l
grep -nE '_STRICT=1' scripts/pre-commit
grep -cE '\|\| exit 1' scripts/pre-commit
wc -l scripts/pre-commit
ls .github/workflows | wc -l
grep -l '^ *schedule:' .github/workflows/*.yml | wc -l
grep -c '^ *- cron:' .github/workflows/*.yml | awk -F: '{s+=$2} END{print s}'   # 31
 ↪  on 2026-09-02 at 424a58007; without the sum, grep prints one count per file
grep -rl 'docs/org/HALT' .github/workflows | wc -l
git ls-files __tests__/scripts | grep -c check-
git ls-files __tests__/scripts | wc -l
node scripts/check-doc-claim-parity.cjs
```

HALT history and the commit footprint:

```
git log --oneline --all -- 'docs/org/HALT*'
ls docs/org/HALT*
git log --format='%an <%ae>' | sort | uniq -c | sort -rn
git log --oneline | wc -l
git log --format='%ad %ae' --date=format:%Y-%m | grep '@orbytjobs.ai' | awk '{print
 ↪  $1}' | sort | uniq -c
git log --format='%ad %an' --date=format:%Y-%m | grep 'push-collision-canary' | awk
 ↪  '{print $1}' | sort | uniq -c
git log --format='%an <%ae>' | grep -c 'gate@test.local'
git log --format='%an|%ae' | grep -E '^orbyt-|^org-|^ig-bot|^push-collision' | grep
 ↪  -vc '@orbytjobs.ai'
git log --format=%ad --date=short -- docs/org/reports/inspector-general/ | sort -u
node -e 'console.log(require("./docs/org/ladder-state.json").heartbeats)'
```

Decisions and directives:

```
grep -cE '^## D-[0-9]{4}' docs/org/decision-log.md
grep -cE '^Prediction' docs/org/decision-log.md   # 14 on 2026-09-02 at 424a58007; the
 ↪  form '^Prediction:' returns 12, because two entries are written 'Prediction,
 ↪  falsifiable:'
grep -cE '^Prediction: none' docs/org/decision-log.md
grep -nE '^Outcome' docs/org/decision-log.md
grep -cE '^## OD-[0-9]{4}' docs/org/owner-directives.md
```

```
grep -nE '^- Status:' docs/org/owner-directives.md
```

The class counts of the 23 organizational lessons in Appendix B:

```
node -e 'const c=require("./public/failure-corpus.json");const
 ↪  w=[20,26,27,28,29,31,34,35,36,40,41,42,43,44,45,46,47,49,50,61,62,63,65];const
 ↪  k={};for(const i of c.items){if(w.includes(Number(i.id))){k[i.kind]=(k[i.kind]||0
 ↪  )+1;if(i.mechanism==null)k.none=(k.none||0)+1}}console.log(JSON.stringify(k))'
```

Issues filed by workflows (the last two need `gh`):

```
grep -rlE 'gh issue create|issues\.create' .github/workflows
 ↪  packages/collective/actions packages/collective/src | wc -l
gh issue list --state all --limit 1000 --json
 ↪  number,title,author,createdAt,labels,state
gh pr list --state all --limit 1000
```

Ladder and liveness state, manifest seats, and arc ledgers:

```
cat docs/org/ladder-state.json
ls docs/org/seats/*.json | wc -l
find docs/org/arcs -name ledger.md | wc -l
```

The token-governance figures in Section 6, The empirical record, are not reproducible by command. They are quoted from the measurement of 2026-07-25 recorded in `CLAUDE.md` and from the header of `packages/collective/src/token-pace.mjs`, and they describe one operator's usage on one plan; the instrument that produced them, `npm run usage:interactive`, reads that operator's local session files and returns different numbers on any other account.

Three of the public datasets cited in this paper are served as static files. `ls public/` on 2026-09-02 confirmed the presence of `autonomy-ledger.json`, `decision-ledger.json` and `failure-corpus.json`, and they are published at the following URLs.

- https://www.orbytlabs.ai/autonomy-ledger.json
- https://www.orbytlabs.ai/decision-ledger.json
- https://www.orbytlabs.ai/failure-corpus.json

D Mechanism inventory

This inventory holds receipts displaced from the body during the final length cut. Each row names the mechanism, the file and line it lives at, what it enforces, the observed refusal or NOT MEASURED where no refusal has been watched, the date the file was read, and the section the row left. Rows are ordered by that section, then by mechanism. Three rows carry two sections, because the same mechanism was cited in both.

- **Decision-log append-only: no guard diffs past entries.** `packages/collective/src/check-org-interface.mjs` line 118; `.gitattributes` line 10; `docs/org/decision-log.md` line 6. Enforces: The decision log's own header rule, verbatim: "NEVER edit or delete a past entry; corrections are new entries" (`docs/org/decision-log.md` line 6). `.gitattributes` line 10 gives the file a union merge driver so concurrent appends merge without conflict.. Observed: NOT MEASURED. No guard in the repository diffs past entries, and three readers of the file were counted at `424a58007`, none of which compares an entry against its earlier self. `packages/collective/src/check-org-interface.mjs` reads the file at line 120 into a variable it never uses, declares a reference regular expression at line 125 and discards it at line 127 with `void refRe`, so the id-resolution check its comment at line 118 describes does not execute. `scripts/check-doc-claim-parity.cjs` reads it at line 153 for the high-water mark and at line 216 for the artifact-drift check, and

`scripts/generate-stats.cjs` reads it at lines 668 and 770. A fourth reader, `scripts/org/check-changelog-report.cjs` at line 259, was created on 2026-09-03 in `8d3bb6914` and does not exist at `424a58007`: this row's 2026-09-03 re-read saw the later tree, where the same two reads sit at 688 and 790. The append-only rule is a convention plus a merge driver, not an enforced gate. Read 2026-09-02, re-read 2026-09-03. (2026-09-02, re-read 2026-09-03)

- **Seat manifests generating the shared-action workflows.** `docs/org/seats/` (`ls docs/org/seats`, 2026-09-02). Enforces: Eight JSON manifests generate workflows for seven seats; four workflows are structural exceptions that use no shared action.. Observed: NOT MEASURED as a refusal. This is a census, read from the directory listing `ls docs/org/seats` on 2026-09-02. (2026-09-02)
- **Countermeasure kinds in the published failure corpus, and the rule that sorts a guard from a test.** `public/failure-corpus.json` (node over the file, 2026-09-02). Enforces: The 42 of 66 (63.6 percent) mechanization rate resolves into 13 guard scripts, 22 tests, 1 workflow and 6 implementation files. Sorting rule: a `scripts/check-*` or `packages/collective/src/check-*` path is a guard, and any `__tests__` path is a test.. Observed: NOT MEASURED as a refusal; this is a census of the corpus, counted 2026-09-02. (2026-09-02)
- **Fence liveness by observed denial (v1 specification, section 2.3).** `docs/collective/spec.md`, section 2.3. Enforces: A control counts as live only when a real denial is watched inside the run it protects; the fence's last step is that observation.. Observed: Rule verbatim: "The only honest liveness signal is watching a real denial happen inside the run, which is the fence's last step." The denial is observed inside every seat run by the fence canary; the verdict printed for the missing case is quoted in Section 4, Guardrails in tiers. (2026-09-02)
- **The fence's explicit non-claim on egress (v1 specification, section 2.2).** `docs/collective/spec.md`, section 2.2. Enforces: A block that was not measured is never declared, because a declared block that does not exist in the world reads as protection. The section exists to state what the fence does not claim.. Observed: Rule verbatim: "Declaring a network block anyway would put a fence in the config that does not exist in the world, which is worse than no fence because it reads as one." The section is titled "Egress is NOT fenced, and this was measured" and records that the sandbox's network settings "were measured doing **nothing** on a hosted runner: off-allowlist hosts answered 200 while the filesystem half of the same fence denied every write." The measurement is told in Section 4, Guardrails in tiers. (2026-09-02)
- **Seat identity across four files.** docs/org/departments.json; docs/org/seats/*.json; app/(site)/leadership/officers.ts. Enforces: Permanent ids across the four places a seat is written down, and a public officer with no registry entry does not build. Observed: "Ids are permanent (the ledger, decision-log, and forecast-accuracy history key on them)"; the registry link "is REQUIRED, so a new officer with no registry entry does not compile" (2026-09-02)
- **Self-naming run record.** docs/org/roster.md. Enforces: A seat's identity is chosen once, recorded, and never changed silently. Observed: "Each seat was given genuine agency over its own identity, name, gender, and persona, and chose in isolation"; "given total freedom, the same-model agents converged hard, on honesty-and-leverage themes and on male presentation across the board."; "The owner chose to keep that authentic outcome; only a three-way collision on 'Agent Ledger' was de-duplicated (it stayed with Finance)." (2026-07-13, read 2026-09-02)
- **Department registry header.** docs/org/departments.json (read with `grep -nE '"id":' docs/org/departments.json`). Enforces: One source of truth that every generic surface iterates, with a precedence rule for an id in both arrays. Observed: "Sin-

gle source of truth for what departments exist. Everything generic (the Chief of Staff, the Inspector General, org-health, the composed authority matrix, reporting) ITERATES this."; "when an id is in both arrays the department entry is authoritative for its content."; the dormant seat carries `"status": "halted-until-stage-2"` (2026-09-02)

- **Heal then check in pre-commit.** scripts/pre-commit, lines 407 and 408; docs/org/CLAUDE.md. Enforces: A healer's own report is not evidence that the write landed. Observed: "Heal writes, check reads back what landed. That is the difference between claiming an effect and observing one" (2026-09-02)
- **Seat-workflow generator refusals.** packages/collective/src/generate-seat-workflow.mjs. Enforces: An unmodelled spec key is an error, an empty spec directory is not a pass, and the comparison is bytes rather than meaning. Observed: `unknown key "${key}". This generator does not model it`; "an empty spec directory FAILS rather than passing with nothing checked"; a semantic comparison "silently drops whatever that comparison does not model" (2026-09-02)
- **Spec derivation from an existing workflow.** packages/collective/src/derive-seat-spec.mjs. Enforces: Derivation refuses to guess rather than transcribing. Observed: it "throws the moment a marker is missing rather than guessing." (2026-09-02)
- **External-input runs in the ledger.** docs/org/stats/*.jsonl. Enforces: Nothing: the flag is recorded and no guard reads it. Observed: `external_input: true` on the innovation runs of 2026-08-01, 2026-08-13 and 2026-09-01 and the marketing runs of 2026-08-16, 2026-08-26 and 2026-08-28; whether any run has been steered by an injected instruction is NOT MEASURED (2026-08-01 to 2026-09-01, read 2026-09-02)
- **Model pin per manifest seat.** docs/org/seats/*.json. Enforces: Every manifest seat runs the same model. Observed: `grep -nE '"model"' docs/org/seats/*.json` returns `"model": "sonnet"` in all nine agent blocks across the eight files (2026-09-02)
- **Push credential split in the caller.** packages/collective/actions/org-agent/action.yml, lines 24 and 27; .github/workflows/legal-weekly.yml, lines 33 to 34, 45, 70 and 80 to 81. Enforces: The agent step never holds a push credential and the sanctioned push carries a just-in-time token. Observed: "the CALLER checks out with persist-credentials:false so the agent step has NO push credential, and the sanctioned push happens from this action with a just-in-time token."; "Passed ONLY to the agent step, never the push step."; "Passed ONLY to the push and stats steps, never the agent step."; the caller declares `permissions: contents: write` at lines 33 to 34 (2026-09-02)
- **Shared action inputs and call sites.** packages/collective/actions/org-agent/action.yml; .github/workflows/*.yml. Enforces: One action, seventeen declared inputs, nine call sites in eight workflow files. Observed: seventeen inputs counted with `awk` over the `inputs:` block; `grep -c 'uses: ./packages/collective/actions/org-agent' .github/workflows/*.yml` returns nine call sites across eight files (2026-09-02)
- **Structural exceptions verified by grep.** .github/workflows/advisory-panel.yml; .github/workflows/inspector-general-weekly.yml; .github/workflows/finance-ledger.yml. Enforces: Each exception is off the shared action by inspection, not by judgement. Observed: each contains "ZERO `- uses: ./packages/collective/actions/org-agent` steps"; the panel's subjects are "DERIVED from docs/org/departments.json (mode ==")observer")"; the Inspector General run is"DETERMINISTIC: no Claude token, no agent, no prompt", "so that nothing in it is agent-authored"; the finance ledger is "dependency-free (pure Node + global fetch), so this job does NOT run npm ci", and a missing secret "makes the script skip THAT feed cleanly (its entry stays honestly null)" (2026-09-02)
- **Caller guard command and the ignored-path gap.** packages/collective/actions/org-agent/action.yml, lines 488 and 578 to 580; .github/workflows/autonomous-content.yml, line 211; .github/workflows/engineering-weekly.yml, line 150;

docs/org/seats/autonomous-content.json; docs/org/seats/engineering.json. Enforces: An optional caller guard run through `bash -c`, whose refusal lands nothing. Observed: two of the nine call sites pass a `guard-command`, at line 211 and line 150; two of the eight lanes include `node_modules` in `fence.lane_paths` and `git clean -fd` at line 488 carries no `-x`, so an ignored path altered during the agent step survives into the guard-command at line 578; whether any run has written there is NOT MEASURED (2026-09-02)

- **Effective keep set and the workflow-generated commit set.** packages/collective/actions/org-agent/action.yml, lines 405 to 437, 555, 573, 679, 681 and 683. Enforces: The workflow, not the agent, decides what is staged after isolation. Observed: the proposed-lessons path joins the set "ONLY when the ladder has activated the protocol (stage3+)"; `git add` names the effective keep set, the dated archive directories, the seat's stats row and the evaluation output (line 681), and the house-keeping module stages the journal and its derived artifact beside them (lines 555, 573, 679 and 683) (2026-09-02)
- **no_sentinel to rejected to the published metric.** packages/collective/actions/org-agent/action.yml, line 872; scripts/lib/autonomy-ledger.cjs, line 40. Enforces: A finished run that omitted its sentinel counts against the seat rather than passing. Observed: the action records the outcome as `rejected` and `COUNTS_AGAINST` buckets `rejected` with `failure`; no `rejected` row existed on 2026-09-02 (`grep -c '"rejected"' docs/org/stats/*.jsonl`) (2026-09-02)
- **Org-writer serialization and the push retry.** .github/workflows/*.yml; packages/collective/actions/org-agent/action.yml, line 745. Enforces: One main-writer at a time, and a rejected push is retried rather than lost. Observed: `grep -l 'group: org-writers' .github/workflows/*.yml \| wc -l` returns fourteen files declaring `concurrency: group: org-writers`; the push step loops `for attempt in 1 2 3`, rebasing on `FETCH_HEAD` between attempts (2026-09-02)
- **Public statement of the commit invariant.** app/(site)/orbyt-collective/process/page.tsx. Enforces: The published description matches what the action does. Observed: "The agent never commits. The workflow does, after resetting the tree to an allowlist." (2026-09-02)
- **Regex dialect fail-closed in the completion gate.** packages/collective/src/completion.mjs. Enforces: A pattern whose meaning differs between the shell's `grep -qE` and Node is refused rather than guessed. Observed: it checks every pattern for "constructs where the two dialects disagree" and "FAILS CLOSED on them rather than guessing." (2026-09-02)
- **Sentinel precedence in code.** `packages/collective/src/completion.mjs`, line 426. Enforces: A tail matching both the sentinel and the skip token is a skip, never a completion. Observed: `if (hasSentinel && !hasSkip) return { outcome: "agent_done" ...` (2026-09-02)
- **House-keeping manifest.** packages/collective/src/house-keeping.mjs; docs/org/CLAUDE.md. Enforces: An absent manifest is an adopter, a broken one is an error. Observed: "A missing manifest is a clean no-op, never an error."; "a manifest that EXISTS and will not parse fails loudly, because reading it as absent silently stops guarding the journal." (2026-09-02)
- **Kernel size and dependency posture.** packages/collective/README.md; packages/collective/package.json. Enforces: The kernel carries no runtime dependency and knows nothing about the company running it. Observed: `git log --diff-filter=A -- packages/collective` dates creation to 2026-08-15; `ls packages/collective/src \| wc -l` returns 43 files, 41 `.mjs` modules and 2 shell scripts; "deliberately small and deliberately alone. It holds decisions, not policy. Everything about a particular company arrives as configuration and arguments; nothing in here knows the name of the org running it.";

"No runtime dependencies, and none are wanted: some callers run this before they have run any install at all." (2026-08-15, read 2026-09-02)

- **License disagreement.** packages/collective/README.md; packages/collective/package.json; docs/org/decision-log.md, D-0039. Enforces: Nothing yet: the decision is made and the text is unwritten. Observed: the README ends with "MIT." and `package.json` carries `"license": "MIT"`, while D-0039 (recorded 2026-08-14 for a decision of 2026-08-12) chooses "Apache-2.0 plus a trademark policy" and states "The DECISION is made and the TEXT is not: counsel signs the final wording and clears the mark." (decision 2026-08-12, recorded 2026-08-14, read 2026-09-02)
- **Orphan build and the host seam.** packages/collective/README.md; packages/collective/src/config.mjs; packages/collective/src/check-advisor-name-leak.mjs. Enforces: The kernel builds and tests with the repository absent, and one file learns the host. Observed: "a static scan only sees the reaches it knows how to spell."; "It was then broken on purpose three ways and watched go red, including a plain `fs.readFileSync("docs/org/...")` with no import statement at all, which no import-scanning lint could ever have found."; the seam is "THE ONE PLACE THE FRAMEWORK LEARNS ABOUT ITS HOST", whose first rule is that "The host root is NEVER derived from this module's own location"; "THE RULE IS FRAMEWORK, THE HAYSTACK IS THE HOST" (2026-09-02)
- **Human promotion marker, and what it does not authenticate.** packages/collective/src/check-org-memory.mjs, lines 11 to 13 and 95 to 116; docs/org/humans.json. Enforces: That a live-memory write blocks without an environment marker. Nothing authenticates the human. Observed: "a staged commit that modifies a LIVE lessons file (`docs/org/lessons/<id>.md`) BLOCKS unless the human promotion marker is present", the marker being `ORG_LESSON_PROMOTION=1`, which is the whole gate; the file's only read of `docs/org/humans.json`, lines 205 to 213, exempts founder emails from its PII scan, and `scripts/check-commit-author.cjs` permits the org-bot identities and performs no signature check. (2026-09-03)
- **Memory tiers and the quarantine.** docs/org/operating-agreement.md; docs/org/lessons/.md. Enforces: A seat loads its own memory only, and a proposed lesson loads into nothing. Observed: Tier 2 is "its own only: its charter, its own LIVE lessons file (`docs/org/lessons/<id>.md`), and its registry entry."; "v1 has no live-landing path. Every lesson from every seat commits to `docs/org/lessons/proposed/<id>.md`, unconditionally. A proposed lesson loads into nothing."; "The live file `docs/org/lessons/<id>.md` is absent from every keep-paths allowlist; that absence is the guarantee, and there is nothing to evade." (2026-09-02)
- **What a verified signature proves, measured in this window.** `git log --format='%G? %GS <%ae>' -500`. Enforces: Nothing: it bounds what the promotion signature can be read to mean. Observed: of the 500 most recent commits on 2026-09-02, 10 authored by `push-collision-canary <org-bot@orbytjobs.ai>` verify as `G` with the founder's signing principal (2026-09-02)
- **Advisory lens definition and the name guard.** docs/org/board-of-advisors.md; packages/collective/src/check-advisor-name-leak.mjs. Enforces: A ratified name never appears inside advisory framing on a public surface. Observed: lenses are "viewpoints an agent adopts, grounded in a named person's published, publicly available work. The name calibrates the lens. The lens is not the person, does not speak for the person, and implies no relationship with the person."; the guard flags only "a ratified name sitting in ADVISORY FRAMING (advisor, advisory, inner circle, cabinet, tenth man, endorsement, affiliation) within three lines of it", and a bare editorial mention counts as INFO, not a finding (2026-09-02)
- **Red-team doctrine.** docs/org/red-team.md. Enforces: Dissent is invoked at gates and

is advisory only. Observed: "a doctrine, invoked at gates, that argues against whatever the room already agrees on. It is advisory, never authority" (2026-09-02)

- **D-0045's falsifier.** docs/org/decision-log.md, D-0045. Enforces: A decision ships the observation that would refute it. Observed: "the next refusal lands in the last two to three days of a month and clears on the first with no action; if it lands earlier, the minutes budget has been eaten by something new and that is a finding." (2026-09-02)
- **Decision and directive counts.** docs/org/decision-log.md; docs/org/owner-directives.md. Enforces: Append-only ids, counted mechanically rather than asserted. Observed: `grep -cE '## D-[0-9]{4}' docs/org/decision-log.md` returns 45 and `grep -cE '## OD-[0-9]{4}'` returns 12 (2026-09-02)
- **Owner-directive register fields.** docs/org/owner-directives.md; packages/collective/src/check-owner-directives.mjs. Enforces: An ask carries an owner and a due date, and a blocked ask is still past due. Observed: the guard "warns on any directive that is open past its due date, unowned, or assigned to a department that does not exist"; the `Blocked-on: founder` field added on 2026-08-11 "never changes whether a directive is past due, only the sentence explaining the delay" (2026-08-11, read 2026-09-02)
- **Authority matrix generation.** docs/org/authority-matrix.md; packages/collective/src/build-authority-matrix.mjs. Enforces: The matrix is regenerated from the registry, and a lane starts gated. Observed: "GENERATED from `departments.json` by `packages/collective/src/build-authority-matrix.mjs`"; "a lane is gated by default, and moving a lane to autonomous is an explicit, logged CEO decision in `decision-log.md` naming the criterion it met" (2026-09-02)
- **Gated-lane breach evidence.** docs/org/departments.json; .github/workflows/autonomous-content.yml, line 81. Enforces: Nothing: it is the record of the act. Observed: `git log -S'new pages' -- docs/org/departments.json \| tail -1` returns `672a7c303` of 2026-07-13; `git show e5bd0f6e0:.github/workflows/autonomous-content.yml` line 81 names the whole `app/(site)/guides` directory in that day's keep-paths (2026-07-13 and 2026-07-30, read 2026-09-02)
- **Inspector General footer rules.** packages/collective/src/inspector-general.mjs. Enforces: A module that could not report says so rather than passing quietly. Observed: "a module that can't finish IS a finding"; a skipped module means the seat "could not verify that claim this run" and the reader is to "treat it as unknown, not clean"; "the IG never 'passes' a finding quietly." (2026-09-02)
- **The stage-2 no_org_fabrication criterion.** `packages/collective/src/evaluate-ladder.mjs`, lines 408 to 422; packages/collective/src/config.mjs (`inspectorGeneralModules`). Enforces: A promotion criterion that no module in the current roster can fail. Observed: the criterion is computed as `latestFm !== null && !fabricated`, where `fabricated` becomes true only when a finding carries `target` equal to `"org"` (2026-09-02)
- **Dimension 42's secret-history gate.** .github/workflows/ci.yml line 222; scripts/check-secret-history.cjs. Enforces: A dimension that blocks with no STRICT variable, so the STRICT-by-number count of ten misses it. Observed: The step "Secret history scan (dim 42, real-secret gate)" at line 222 runs `scripts/check-secret-history.cjs` with no STRICT variable; its comment reads "Dim 42 promoted to a real gate 2026-06-24 (non-strict)" and says the heuristic layer "stays report-only" with `SECRET_HISTORY_STRICT` unset (`sed -n '218,223p' .github/workflows/ci.yml`). `governance.md` records that dimensions "37 / 39 / 40 / 42 / 45 / 46 were promoted on 2026-06-24". (2026-09-02)
- **The /evolve-audit landing gauntlet.** docs/audit-dimensions/governance.md. Enforces: Seven preconditions before an agent may land a report-only dimension unattended. Observed: Described as "mechanized, not promised": report-only by default with a grace-

ful skip; "Deterministic + offline", defined as "identical verdict every run on the same tree; runs in seconds with no required new infra / live network / paid service"; a test that drives the real script through a temporary sandbox, "covering each sub-check firing + not-firing, the allowlist marker, and report-only vs STRICT"; low false-positive, meaning "its findings on HEAD are all genuine debt"; the full local gauntlet passing; "Atomic + reversible", meaning "landed as its own commit so one `git revert` undoes it. NEVER batched with product code."; and house style. The guarantee: "Report-only literally cannot block a commit or CI, the gauntlet gates the landing, and the commit is revertible. The only breaking class (STRICT) is human-gated. That is the whole guarantee." (2026-09-02)

- **The CI job set and its STRICT confinement step.** .github/workflows/ci.yml. Enforces: The guard battery also runs off the developer's machine. Observed: 9 jobs: `lint-and-typecheck`, `unit-tests`, `build`, `typecheck`, `security`, `collective-orphan`, `guard`, `notify-failure` and `notify-recovered`. The `guard` job carries 21 MATCHING SOURCE LINES under a 10 minute ceiling (`sed -n '294,427p' .github/workflows/ci.yml \| grep -cE "scripts/check-\|src/check-\|check:"` returns 21). That command counts lines, not invocations. Twenty of the 21 are direct `run:` steps; the twenty-first, at line 420, is the body of a `for` loop over the 18 report-only checks listed at lines 416 to 419. A successful run therefore executes 38 check commands over 33 distinct scripts, because `cron-continuity`, `stripe-boundary`, `external-cost`, `privacy-manifest` and `ai-metering` each run twice, once STRICT as a direct step and once report-only inside the loop. Its confinement step is STRICT: "an unfenced agent workflow, or one that persists a push credential into .git/config where the agent can read it, is not a warning." (2026-09-02)
- **The commit-author guard.** scripts/check-commit-author.cjs; scripts/pre-commit, second blocking step. Enforces: A commit under an undeployable local identity is refused before any check of content. Observed: The hook: "a commit made under an undeployable identity is wasted no matter what the rest of this gauntlet says: Vercel BLOCKS the deployment after the push has already reported success. A stray local `user.email` of gate@test.local cost four blocked production deploys on 2026-08-14 with git, GitHub and CI all green." The guard's header: "Seventeen commits were authored under it before anyone noticed", four consecutive production deployments `state: BLOCKED` (`35c4c6ffe`, `288f0c392`, `19bc05cbc`, `b04126514`), "production sat at df8a923ed for seven hours while the pushes all reported success.", and "the identity lives in `.git/config`, which every guard here is blind to by construction." (2026-08-14)
- **The full-suite placement measurement.** scripts/pre-commit, comment dated 2026-08-01. Enforces: The local gate runs every tier that CI runs. Observed: The speed rationale was tested on the same machine the same day: `npm run test:fast` took 38.9 seconds for 14,194 tests, and `npx vitest run` took 44.9 seconds for 14,488 tests. The comment's conclusion is that "Six seconds buys 294 more tests and every tier CI runs", and that a slow suite is split by changed path, never by tier. (2026-08-01)
- **The pre-commit hook's blocking shape and its installation.** scripts/pre-commit; scripts/install-hooks.sh; package.json. Enforces: The installed hook cannot drift from the file under review, and it reaches hosted runners. Observed: 24 lines ending in `\|\| exit 1` plus six bare `exit 1` lines, at lines 124, 134, 165, 187, 298 and 323 (`wc -l scripts/pre-commit`; `grep -cE '\|\| exit 1' scripts/pre-commit`; `grep -nE '\s*exit 1\s*$' scripts/pre-commit`). Installed by `ln -sf "../../scripts/pre-commit" "$ROOT/.git/hooks/pre-commit"`. Hosted runners get it because `npm ci` executes the package's `prepare` script, `"prepare": "bash scripts/install-hooks.sh"` (`package.json`); `CLAUDE.md` item 47 records five workflows running the suite through it. (2026-09-02)
- **The suite gate's dead branches and its move to a file.** scripts/pre-commit;

scripts/test-gate.sh. Enforces: The verdict comes from the exit code, not from formatted output. Observed: The regex's "Test Files" was no longer followed by a digit. A third alternative, anchored at `FAIL`, was dead in both environments because vitest prints ' FAIL ' with a leading space. Inside CI the gate therefore had zero live branches. Run 31720972207 printed "Test Files 1 failed | 965 passed". The summary grep survives only as "a second, ANSI-stripped opinion", and the gate lives in its own file "so it can be executed by a test against a stubbed `npm`" rather than grepped for. (2026-08-13)

- **The arc directory on the survey date.** docs/org/arcs/. Enforces: An arc's state is a committed ledger block; none exists. Observed: `ls docs/org/arcs/` returned a README and a `tools` directory and no ledger of any kind, and `find docs/org/arcs -name ledger.md \| wc -l` returned 0. (2026-09-02)
- **The production build skip list.** scripts/vercel-ignore-build.sh, lines 98 to 102. Enforces: Which changed paths skip a production build after a seat's commit lands. Observed: Skips only when every changed path matches `docs/org/*`, `docs/books/*`, `.github/*` or one of three exact generated files (`public/autonomy-ledger.json`, `app/(site)/founder/building-in-public/stats-data.json`, `public/book-production-ledger.json`), and builds on any other path and on any doubt, such as a missing previous-deploy SHA. (2026-09-02)
- **The three prose layers above the enforced pair.** CLAUDE.md; docs/audit-dimensions/governance.md. Enforces: Nothing: they are documentation that disagrees with itself. Observed: "Claude MAY `git commit` AND `git push` to `main` (which auto-deploys to production) WITHOUT asking, when ALL of these hold"; "Push to `main` at most ONCE per day."; "**Push** (deploy stays a human action)". Underneath all three, the valve denies the act in every session and `permissions.deny` refuses the same five command shapes. The prose layers do not agree with each other, and only the bottom layer is executed. The valve was proven live on 2026-08-23, when, in `CLAUDE.md`'s words, "it denied its own installer's self-test". (2026-09-02)
- **The valve's matcher and the channel-change rule.** .claude/hooks/arc-valve.sh; CLAUDE.md. Enforces: A denial is answered by changing channel, never by rewording. Observed: The hook serialises the whole tool input with `jq -r '.tool_input \| tostring'` and applies case-insensitive extended regular expressions to that string, so prose that merely mentions a forbidden phrase trips the same pattern as the act. `CLAUDE.md`: "Never rephrase a command to get past a valve denial; prose that must mention a forbidden command goes through the file tools, not a shell heredoc, or the valve reads it as the act." Whether `push.default` in the agent's environment makes a bare push reachable is NOT MEASURED. (2026-09-02)
- **The valve's seven classes and the runtime deny list.** .claude/hooks/arc-valve.sh; .claude/settings.json. Enforces: Seven human-only classes, three of them on every MCP tool, plus five literal deny entries. Observed: Three classes apply to Bash and to every MCP tool, because "SQL sent through an MCP connector is caught the same as SQL in a shell". The five entries are `Bash(git push origin main*)`, `Bash(git push -u origin main*)`, `Bash(git push origin HEAD:main*)`, `Bash(git push --force*)` and `Bash(git push -f *)`. The valve's header: "permissions.deny in settings.json is the hard gate for push-to-main; this is defense in depth." Every denial returns: "VALVE: $1 is human-only. Do not rephrase to get past this. Log the action, reason, and rollback; mark dependents GATED; keep looping on everything else." (2026-09-02)
- **The confinement invariant and its first false positive.** packages/collective/src/check-agent-confinement.mjs. Enforces: Fence plus persist-credentials false on every workflow that runs an agent. Observed: "Any workflow that EXECUTES an agent AND CHECKS OUT the repository must: 1. run the fence (`org-fence`), so an out-of-lane write is refused at the syscall rather than trusted not to happen, and 2. check out with `persist-`

`credentials: false`, so the agent step cannot read a push credential out of `.git/config`." Its first version "flagged two workflows that contain the sentence 'no claude -p step anywhere in this workflow', which is a promise not to do the thing, read as doing it." (2026-09-02)

- **The fence action's creation and wiring.** .github/actions/org-fence/action.yml, later packages/collective/actions/org-fence/action.yml. Enforces: Dates the fence's coverage of the run ledger. Observed: Commit `cf71ad735` created the action (`git log --diff-filter=A -- .github/actions/org-fence/action.yml`); it moved to `packages/collective/actions/org-fence/action.yml` on 2026-08-16 in `1e93efeb1`, a zero-content rename a day after the package was created in `c1cf77617`; the first commit `git log --reverse -S'org-fence' -- .github/workflows` returns is `e13e4ca37`, eight minutes later the same morning. (2026-08-13)
- **The fence canary's premise, and the job that calls the real action.** .github/workflows/fence-canary.yml. Enforces: Undocumented platform behaviour is watched on a daily cadence during rollout, and the fence is watched by a job that invokes it (P1). Observed: It calls the real action, not a copy, and runs daily at 03:00 UTC during rollout, with a comment that names weekly as the destination once every fenced seat has a green scheduled run. A documentation check on 2026-08-13 found no vendor guidance on Linux sandbox requirements and no documented per-job runner lifecycle, so "every assumption the fence stands on can be withdrawn silently." The workflow states its purpose: "so that the day it is withdrawn, we learn it from a red canary rather than from an agent running unfenced." (2026-08-13; 2026-09-02)
- **The fence's deny-list construction and ceiling.** packages/collective/actions/org-fence/action.yml, line 184. Enforces: A lane is enforced by denying siblings at every level, under a 600-path ceiling. Observed: Installed as managed settings at `/etc/claude-code/managed-settings.json`. The input description: "Lanes are built by ENUMERATING SIBLINGS TO DENY because `allowWrite` is inert for confinement and deny beats allow even for a nested path, so 'deny the repo except this subdirectory' is not expressible." The list is regenerated from the tree on every run, because "a NEW top-level directory is writable until this list is regenerated". Above the ceiling the generator throws, "fence has N deny paths, over the 600 ceiling", so the run fails rather than running behind a truncated fence (line 184, read 2026-09-02). The `.git` directory is denied even if listed as a lane. (2026-09-02)
- **The fence's third non-claim.** packages/collective/actions/org-fence/action.yml. Enforces: Configuration presence is not evidence of a fence. Observed: "It does NOT check that a settings file EXISTS. In `-p` mode an invalid settings file is silently ignored: exit 0, no warning, no fence." (2026-09-02)
- **The liveness canary's evidence.** packages/collective/actions/org-fence/action.yml. Enforces: The denial is read from the transcript, never from a file the agent wrote. Observed: The canary target is `/tmp/org-fence-canary-target`, which resolves to `<first deny entry>/.orbyt-fence-canary`. The evidence is the tool result "which the parent claude process writes OUTSIDE the sandbox", because "the Linux sandbox gives the child its own /tmp" and a child-written file "reads exactly like a command that never ran". (2026-09-02)
- **The push-collision canary's ref.** .github/workflows/push-collision-canary.yml. Enforces: A real non-fast-forward rejection is manufactured out of band. Observed: Every Monday at 02:00 UTC against `refs/canary/push-collision-<run_id>`, a ref outside `refs/heads/*` that no deploy and no branch-filtered workflow can see, deleted on every path. (2026-09-02)
- **Scoped-halt provenance, the supervised re-run, and the two survive-the-halt exceptions.** docs/org/rollout-ladder.md; docs/org/operators-manual.md;

packages/collective/src/check-org-wiring.mjs; .github/workflows/repo-backup.yml. Enforces: Who may clear a scoped halt, and which two workflows run under a global one. Observed: "the evaluator clears ONLY halts whose body says written_by: evaluator (its own, after 3 clean daily reads of the tripping signal) or written_by: ladder (pre-staged dormancy, cleared at that stage's promotion). Human halts are human-cleared." The manual adds "Never hand-edit `written_by` to unblock yourself faster; that defeats the provenance check." and, in procedure 3, a re-run dispatched with `supervised: true`, "which bypasses only that one scoped HALT, never the global `docs/org/HALT`"; the action's input description says supervision "proceeds past the SEAT-scoped HALT (never the global HALT)". `packages/collective/src/check-org-wiring.mjs` names `org-ladder.yml` as "the ONE ratified survive-the-halt exception" because "the kill-switch stops agent action, not deterministic observation", and `.github/workflows/repo-backup.yml` states that "preserving state is not agent action." (2026-09-02)

- **The evaluator's verdict set and strike rule.** packages/collective/src/evaluate-seats.mjs. Enforces: Three problem verdicts before demotion is proposed; a run the agent never entered is excused. Observed: The `VERDICT` table holds `healthy`, `infrastructure`, `needs_fix`, `underperforming`, `not_observed` and `dormant`. The strike constant: "Consecutive problem evaluations before demotion is even proposed. Three, because the point is to fix the seat: one is a bad day, two is a pattern somebody should have acted on, three is a pattern nobody acted on." On the zero-turn correction: "Excluding runs the agent never entered, marketing ran 9 times and failed 3, which is 33%, under the threshold." and "Blaming a worker for a shift they were never called in for is the same mistake in either world." (2026-09-02)
- **The HALT preflight and re-ask comments.** packages/collective/actions/org-agent/action.yml, lines 688 to 704. Enforces: HALT stops a landing, not a running spend, and is decided against the remote. Observed: "Preflight checked HALT once, up at the top, before an agent run that takes 15 to 30 minutes. A founder who writes HALT during that window could not stop the push, which makes it a kill switch for runs that have not started rather than for runs that are running." (lines 688 to 692). "IT MUST ASK THE REMOTE, NOT THIS TREE. The isolate step resets the working tree to BASE_SHA, so a HALT thrown while the agent was running lives as a commit on origin/main and is NOT in this checkout and never can be." (lines 701 to 704). (2026-09-02)
- **The kill switch's shape and scope, and the false clause in it.** docs/org/kill-switch.md; .github/workflows/refresh-pay-transparency.yml; .github/workflows/quarterly-salary-update.yml. Enforces: One committed file, halting org workflows only. Observed: On the choice of a committed file over a secret or a dashboard: "Because it is the simplest thing that is impossible to miss, works with no infra, is visible in git history (you can see exactly when it was pulled and by whom), and cannot be forgotten in a config UI. A solo founder's kill-switch should be one obvious file." On scope, HALT "does NOT gate the separate app-maintenance crons (`seo-heal`, `model-refresh`, `quarterly-salary-update`, `refresh-pay-transparency`), which predate the org, run on their own schedules and their own controls, and only ever open PRs or write branches (never auto-merge to production)." That last clause is FALSE for two of the four named crons. `.github/workflows/refresh-pay-transparency.yml` runs `scripts/extract-commoncrawl-pay-transparency.py` with `--upload` (lines 90 to 93), and `upload_to_supabase` (line 480) POSTs the refreshed CSV with `x-upsert: true` over the canonical Supabase Storage object `ingestion-data/pay-transparency-latest.csv` (lines 496 to 508), which the workflow header describes as refreshing "in-place" for the next ingestion cron to read (lines 18 to 21). `.github/workflows/quarterly-salary-update.yml` runs `scripts/snapshot-salary-data.cjs` (line 37), which POSTs batches of 100 rows into the `salary_snapshots` table with `Prefer: resolution=merge-`

`duplicates` (lines 65 to 86). Both workflows pass `SUPABASE_SERVICE_ROLE_KEY`, so both write past row-level security. HALT is an ORGANIZATION-WORKFLOW kill switch, not a repository one, and production data mutation by these two crons continues under it. NOT MEASURED: whether either has ever run while a HALT file was present, and neither has a halt control of its own. (2026-09-02; re-read 2026-09-03)

- **The ladder run of the halt day.** gh run view 32365174447 `--json` createdAt,conclusion,jobs (receipt carried in the frozen original of 04-guardrails.md). Enforces: Nothing: it records what ran on the day the global switch was pulled, and the alarm job failed with zero steps. Observed: The HALT was lifted at 09:11 UTC (`a4fadccae`, `git show -s --format=%ad`), and the ladder run of that day, 32365174447, started at 11:43 UTC, after the lift, and concluded `failure` in all four jobs, `evaluate`, `ci-headroom`, `liveness` and the alarm job, each with zero steps. The cause is NOT MEASURED, because the run log had expired by the survey date. The external monitor filed issue #99, "Orbyt is DOWN", at 23:42 UTC that night. (2026-08-20; 2026-09-02)
- **The ladder's five stages.** docs/org/rollout-ladder.md. Enforces: The ratchet's autonomous reach ends at stage 4. Observed: Stage 0 is the build; stage 1 brings the quality seat live; stage 2 schedules the Inspector General; stage 3 opens the lesson protocol org-wide; stage 4 is "surfacing only (the ratchet's autonomous reach ends here)". (2026-09-02)
- **The ladder's heartbeat array.** docs/org/ladder-state.json. Enforces: Nothing: it records which days carry no heartbeat. Observed: 14 dates between 2026-08-17 and 2026-09-01, with two absent, 2026-08-20 and 2026-08-31. The 2026-08-20 absence is not the halt's doing, since the ladder is a ratified survive-the-halt exception and the halt was lifted at 09:11 UTC. (2026-09-02)
- **The remediation rule.** docs/org/seat-state.json, header. Enforces: A cycle with no intervention escalates to the humans rather than the seat. Observed: An entry "advances a seat toward demotion ONLY when an intervention was recorded and the defect survived it; a cycle where nothing was changed escalates to the humans instead, because the seat did not fail to improve, we failed to intervene." (2026-09-02)
- **Dimension 72's stated ground.** scripts/check-doc-claim-parity.cjs, check 6. Enforces: A guard test with no stays-quiet case is warned about. Observed: It warns on the stated ground that "A matcher tested only on the case it was built for" has been proven loud and never proven silent. (2026-09-02)
- **Dimension 82's grading rule.** packages/collective/src/check-audit-liveness.mjs. Enforces: Declining to look out loud is a pass; silence is not. Observed: "SKIPPED and NOT OBSERVED are verdicts, and are PASSES here, because declining to look out loud is the honest behavior" the apparatus is built around. (2026-08-05)
- **The advisory panel's zero-turn rows and the evaluator's excusal of them.** .github/workflows/advisory-panel.yml, lines 296 to 299; packages/collective/src/evaluate-seats.mjs. Enforces: That a run torn down before the agent worked is excused from the ladder, and that an unmeasurable field takes the schema's declared unknown value rather than a guess. Observed: Eight zero-turn rows: six completions written by the advisory panel on behalf of operations and revenue (2026-08-02 and 2026-08-14, workflow "Advisory Panel (observers)") and two marketing failures (2026-07-28 and 2026-08-13). The panel runs two agents at up to 90 turns each but reads plain-text output, so "there is no result event carrying num_turns or usage" and the row takes the schema's "established unknown values, 0 and {}". (2026-09-02)
- **The frozen baseline note.** public/autonomy-ledger.json, baseline. Enforces: That the published series is measured against an origin that is never regenerated. Observed: "Opening baseline of the Orbyt Autonomy Ledger, frozen at first publication. Never regenerated: the series is measured against this origin." (2026-09-02)
- **The public ledger page's discard vocabulary (a documentation-lag finding).**

app/(site)/orbyt-collective/autonomy-ledger/page.tsx, lines 42 to 48 and 63. Enforces: Nothing. It describes a review step the counter does not perform. Observed: The page says “review-panel discards” and “what the review panels killed”, while the counter counts files the agent changed outside its keep-set, reset away at isolation. (2026-09-02)

- **The published completion-rate definition and the ledger library’s two throws.** scripts/lib/autonomy-ledger.cjs, printed verbatim into public/autonomy-ledger.json. Enforces: What counts as a completion, which rows are excluded from both sides of the rate, and that the rate is never computed from a zero denominator. Observed: “completion_rate_pct = runs whose output survived the workflow’s completion gate, divided by all recorded runs. A run whose output the guards refused counts as a failure. Rows recording that no run happened are counted separately and excluded from both sides: skip (the seat had nothing to do), halted (a kill switch stopped it), infrastructure (the org could not run at all). Failures and expert-panel discards are printed, never netted out.” The library throws if completions plus failures do not equal runs, and throws if the ledger holds only neutral rows. (2026-09-02)
- **The stats step’s outcome-to-row mapping, and the stale comment about it.** packages/collective/actions/org-agent/action.yml, lines 725, 864 and 881; packages/collective/src/completion.mjs, lines 101 and 346. Enforces: That every named outcome, a halt included, attempts a ledger row. Observed: the preflight step writes RUN_OUTCOME=preflight (line 103) and then runs the halt decision, which appends RUN_OUTCOME=halted to the same append-only GITHUB_ENV (completion.mjs, lines 101 and 346), so the halted value wins and the `if: always()` stats step maps it to a halted row (line 864). The action’s own comment at line 725 still calls RUN_OUTCOME=halted “the existing no-row path, the same one preflight uses”; it was committed 2026-08-16 in b75d934b6 and the explicit halted case arrived 2026-08-27 in e4ccce888, eleven days later. The ledger’s halted column is 0 because no run ever met a halt, not because the path writes nothing. (2026-09-03)
- **Composition of the pre-commit blocking lines.** scripts/pre-commit. Enforces: That 20 of the 24 or-exit lines invoke a guard script. Observed: Two of the 24 or-exit lines are the seat-workflow generator in `--heal` and `--check` mode, one is npx tsc `--noEmit`, and one is ESLint over staged files. The six bare exit 1 lines sit inside conditional blocks, and lines were not resolved to steps. (2026-09-02)
- **Counting caveats behind census rows 6, 11 and 13.** .github/workflows/ci.yml line 362; .github/workflows/ingestion-run.yml; **tests**/scripts. Enforces: That each census figure is reproducible only with the anchor its command carries. Observed: Command 6’s bare form returns 11, the extra being “AI route gating (free-tier money-leak guard, strict gate)” at line 362, the AI_GATING_STRICT step already counted in the pre-commit STRICT set. The scheduled-workflow count of 24 excludes ingestion-run.yml, whose schedule: trigger is commented out; a bare grep -l schedule: returns 25. The cron-line count of 31 requires the `- cron:` anchor; a bare grep -c cron returns 66. Row 13 counts tracked files; a bare ls also counts this paper’s untracked build test. (2026-09-02)
- **The field note of 2026-08-27, first claim and its data rows.** app/(site)/research/research-registry.ts, lines 178, 259 and 263 at 424a58007 (the working tree of 2026-09-02 carries this paper’s own registry entry above them, so the labels rather than the lines are cited). Enforces: Nothing. It is a prior point-in-time measurement of the same population. Observed: “As of 27 August 2026, the Orbyt Labs repository enforces 105 automated guard scripts, of which 24 block a commit outright, and all 87 of its guard test suites assert that the detector stays silent on a clean input rather than only firing on a bad one.” Its data rows record 105 guard scripts, 24 blocking, 80 with a STRICT switch, 87 guard test suites, 87 of 87 proving silence, 90 audit dimensions and, in a row labelled “Logged failures with a guard built in response”, the

value 59. The note states its own limitation: an absence idiom in a test is evidence that a negative case was written, not proof that it is a good one. (2026-08-27)

- **The field note of 2026-08-27, second claim (the row that disagrees with the first).** app/(site)/research/research-registry.ts, the second field note, at 424a58007. Enforces: Nothing. It reads the same 59 differently from the note beside it. Observed: "Of 59 distinct engineering failures logged by an AI-run organization as of 27 August 2026, 34 raised no error at all and were found only because someone checked an artifact directly, and 37 have a named automated guard built in response", with its row labelled "Failures with a named guard or audit dimension" tabulating "37 of 59 (63%)". (2026-08-27)
- **The Inspector General's weekly report history after the scoped halt was deleted.** docs/org/reports/inspector-general/; git log `--format='%ad %an' --date=short --` docs/org/reports/inspector-general/. Enforces: Nothing. It is the evidence that the seat is running while three surfaces still call it halted. Observed: Four dates on 2026-09-02: 2026-08-11, the founder's scaffold commit 0f09acdce, then 2026-08-15, 2026-08-22 and 2026-08-29, the seat's own reports under ig-bot. (2026-09-02)
- **The liveness guard's scoped-halt comment, in full.** packages/collective/src/check-org-liveness.mjs. Enforces: That a scoped halt stops one seat and only a global HALT stops every seat. Observed: "`docs/org/HALT` stops EVERYTHING, while a scoped `docs/org/HALT-<id>` stops ONE seat. Treating a scoped halt as global would have silenced the whole watchdog because a single seat was deliberately dormant, which is precisely the blind spot it exists to remove." (2026-09-02)
- **The registry field that renders the stale dormancy a second time.** docs/org/departments.json line 4 (authority_matrix_note); docs/org/authority-matrix.md lines 3 and 31; packages/collective/src/build-authority-matrix.mjs. Enforces: That the authority matrix is generated from the registry, so a stale registry field becomes a stale published document. Observed: The note field reads "its dormancy behind HALT-inspector-general is stated in its registry entry and clears at the Stage 1 to 2 promotion". docs/org/authority-matrix.md line 31 is rendered from that field; its line 3 reads "GENERATED from `departments.json` by `packages/collective/src/build-authority-matrix.mjs`", so it is the registry's lag surfaced a second time, not a third document. (2026-09-02)
- **The decision log's four passed check dates, verbatim.** docs/org/decision-log.md, lines 53, 69, 298 to 300 and 338. Enforces: Nothing reads them. No script consults a prediction or a check date. Observed: grep -nE 'Check date' docs/org/decision-log.md finds three (lines 53, 69 and 338): D-0009 of 2026-07-13, "Check date: 2026-07-20.", six days after the log opened; D-0028 of 2026-07-20, "Check date: 2026-07-27."; and D-0007 of 2026-07-13, "Check date: 2026-08-13." The fourth, D-0025 of 2026-07-20, wraps the phrase across a line break and reads in full: "Prediction: by 2026-08-19 the ladder reaches at least Stage 1 promotion with zero false-green incidents, and at least one CQO insight is actioned. Check date: 2026-08-19." (lines 298 to 300). grep -nE 'Check date|^date: 2026' docs/org/decision-log.md finds all four, at lines 53, 69, 300 and 338. (2026-09-02)
- **The 16 stats rows that carry no token values.** docs/org/stats/*.jsonl; cat docs/org/stats/*.jsonl | jq -c 'select(.tokens==null or .tokens=={})'. Enforces: That an unmeasured field is written as the schema's unknown value or omitted, never as zero usage. Observed: Of the 65 rows, 49 carry a tokens object with values (out, cr for cache read, cw for cache write, in), 8 carry an empty object {}, the schema's unknown value (the six advisory-panel rows and the two zero-turn marketing failures), and 8 carry no tokens field at all, every one dated 2026-07-21 to 2026-07-25, before the field was written. (2026-09-02)
- **The empirical weekly ceiling, declared as empirical by the instrument that uses it.** packages/collective/src/token-pace.mjs, header. Enforces: That the pace hook

calibrates on observed history and never reports an invented quota as official. Observed: "THE CEILING IS EMPIRICAL AND SAYS SO. Anthropic does not publish the Max 20x quota in tokens. Rather than invent one, this calibrates on the owner's own history: he exhausts the quota most weeks, so his heaviest observed week is a lower bound on the real limit and a usable proxy. Re-measured 2026-07-25 after the subagent counting fix: 507M, 338M, 569M, 544M weighted across four completed ISO weeks, three of four within 11% of the peak. Reported as EMPIRICAL, never as the official cap." (2026-09-02)

- **The six runs that ran within five turns of the default ceiling of 80.** docs/org/stats/*.jsonl; jq -c 'select(.num_turns>=75 and .num_turns<=85)'. Enforces: Nothing. It sizes what a seat run costs at the turn ceiling. Observed: Six rows sit within five turns of the ceiling; the product row of 2026-07-21 carries no tokens field. The five with values: engineering 2026-08-06 (80 turns) recorded 37,855 output and 6,724,175 cache-read tokens; legal 2026-08-12 (78) 37,090 and 7,786,706; product 2026-08-13 (81) 38,300 and 11,210,638; marketing 2026-08-15 (81) 48,401 and 10,983,197; product 2026-08-26 (76) 49,603 and 9,299,214. Three sit at the mean output and two sit 28 to 31 percent above it. (2026-09-02)
- **The issue and pull-request census, taken after the GitHub API was reachable.** grep -rlE 'gh issue create|issues.create' .github/workflows packages/collective/actions packages/collective/src | wc -l; gh issue list `--state` all `--limit` 1000 `--json` number,author,createdAt,labels,state,title; gh pr list `--state` all `--limit` 1000 `--json` number. Enforces: Nothing. It is a first measurement, and it shares the ledger's blind spot, because an issue is filed by a workflow that reached its failure step. Observed: Twenty workflow or action files are able to file an issue. gh issue list returned 49 issues, numbered 46 to 129, created between 2026-07-21 and 2026-09-01: 46 filed by app/github-actions with the automation label and 3 by app/healthchecks-io with the Heartbeat label; 2 in July, 44 in August and 3 in September; 33 closed and 16 open. The most common titles are "CI is red on main" (13), "Autonomous content run failed its guards" (13, once the bracketed cause tag and the date are stripped) and "Org ladder run went red" (10). The range 46 to 129 holds 84 numbers, 49 of them issues; the 35 absent are pull requests, of which the repository holds 80, 35 inside the range and 45 below it. The title frequencies require the `title` field, which the field list above names; the whole census was re-run on 2026-09-03 and every figure in this row reproduced unchanged. (2026-09-02, re-run 2026-09-03)
- **org-fence liveness step, the out-of-lane write that must be denied.** packages/collective/actions/org-fence/action.yml. Enforces: P1: a control counts as live only when its refusal has been observed inside the run it protects. Observed: Lesson 44: a sandbox probe on a hosted runner concluded `success` with `permission_denials: []` while three of its four agent runs had executed nothing, because only the network half of the sandbox had failed to apply. The rule: "fence liveness must be a POSITIVE OBSERVATION of a real denial inside every run" (2026-08-13)
- **the commit gate's verdict, extracted into a file a test can execute.** scripts/test-gate.sh. Enforces: P3: the exit code is the verdict. Observed: The file states the principle: "So: the EXIT CODE is the verdict." (2026-08-13)
- **the three rules that survive the session, written after seven errors in one day.** CLAUDE.md, "The three rules that survive the session". Enforces: P4: every detector ships a negative case. Observed: Among the four detectors that mis-fired, "a scoped `HALT-<id>` was read as global so one dormant seat silenced the whole watchdog." The diagnosis: "Every one passed its own suite, because the suite only ever proved it could be LOUD." (2026-07-23)
- **the seat keep-set, which is what enforces the sandbox boundary for a scheduled seat, and the queue a promotion waits in.** docs/org/seats/*.json (`keep_paths`); docs/audit-dimensions/strict-promotion-queue.md. Enforces: P5: agents live in

the sandbox, the gate is the human's. Observed: Every `keep_paths` value is a `docs/org/reports/*` path, a `docs/seo` path, or one of six `app/(site)` content paths, so `scripts/pre-commit` and `.github/workflows/ci.yml` lie outside all of them; candidates for promotion wait in `docs/audit-dimensions/strict-promotion-queue.md` (2026-09-02)

- **the valve, which applies the mechanism-over-prompt rule to the human-only set.** .claude/hooks/arc-valve.sh; docs/lessons-learned.md, lesson 40. Enforces: P6: the human-only set is enforced by the runtime, not the prompt. Observed: Lesson 40's derived rule: "when a fix has a prompt half and a mechanism half, the mechanism half is the fix and the prompt half is hygiene" (2026-08-10)
- **the valve's one recorded denial.** CLAUDE.md, the /long-horizon arcs and the valve section. Enforces: P6: the human-only set is enforced by the runtime, not the prompt. Observed: `CLAUDE.md` records the valve "proven live 2026-08-23 when it denied its own installer's self-test" (2026-08-23)
- **the seat manifest layer, turned the right way round.** docs/org/decision-log.md, D-0040. Enforces: P7: the spec is the source, the workflow is the artifact. Observed: D-0040 records that the layer shipped on 2026-08-14 "with the direction running backwards from the way it was described" (2026-08-16)
- **the only prediction check in either guard set, and what it actually reads.** packages/collective/src/check-org-interface.mjs, lines 150 to 156. Enforces: P9: a prediction carries a check date. Observed: It reads department proposal files, not the decision log's predictions or check dates (2026-09-02)
- **the owner-directive register, built after an ask left no trace.** docs/org/owner-directives.md. Enforces: P9: every human ask has a place to land. Observed: Its diagnosis: "A sentence spoken in chat had nowhere to land, so it landed nowhere, twice." (2026-07-29)
- **the per-seat evaluator's zero-turn rule, corrected by directive.** packages/collective/src/evaluate-seats.mjs, header. Enforces: P10: autonomous evaluation may suspend a seat and may not promote one. Observed: The header records a founder directive "correcting the first version of this file, which went straight to a halt" (2026-08-27)
- **Class labels in the failure corpus, assigned by a single rater.** docs/lessons-learned.md (the source the corpus is generated from). Enforces: The three-class taxonomy the paper's headline share of 33 of 66 rests on. Observed: 42 commits, 14 under a seat identity and 32 carrying an agent session trailer (`git log`, 2026-09-02) (2026-09-02)
- **Sensitivity of the published completion rate to one added or removed run.** public/autonomy-ledger.json. Enforces: The reading of the 4.6-point gap between the current rate and the frozen baseline. Observed: 55 of 66 is 83.3 percent; 55 of 64 is 85.9 percent; 24 of 31 is 77.4 percent; 24 of 29 is 82.8 percent; two added failures take the baseline to 24 of 32, 75.0 percent (2026-09-02)
- **Timeout alarms in eleven org workflows gating on `cancelled` alone.** docs/lessons-learned.md, lesson 41. Enforces: Notification when a scheduled job is killed at its ceiling. Observed: "empirically no `notify-timeout` has fired once in repo history" (2026-08-10)
- **Inspector General weekly report archive.** docs/org/reports/inspector-general/archive/2026-08-29.md. Enforces: The seat's weekly reporting, which three surfaces still describe as halted. Observed: the latest archived report on the survey date, the seat having reported every week since the scoped halt file was deleted on 2026-08-14 (2026-09-02)
- **Stated license of the extracted apparatus.** packages/collective/README.md line 99. Enforces: Nothing; it is the package's own description of its terms. Observed: "MIT.", against decision D-0039's "Apache-2.0 plus a trademark policy", whose text the decision records as unwritten (2026-09-02)
- **Age of the fence against the ledger window.** docs/org/stats/*.jsonl;

.github/workflows/engineering-weekly.yml. Enforces: The bound on how many recorded seat runs executed inside a fence. Observed: 32 runs dated before 2026-08-13 (`cat docs/org/stats/*.jsonl \| jq -r .date`); the engineering run of that day started at 09:57 UTC (`gh run view 31688969118 --json createdAt`) while its workflow received the fence at 00:59 UTC the next day (`git log --reverse -S'org-fence' -- .github/workflows/engineering-weekly.yml`); the other seven runs dated 2026-08-13 started between two and 41 minutes after their workflow's fence commit was authored, and whether that commit was on the branch when each run started is NOT MEASURED (2026-09-02)

Each entry above names the mechanism, the file and line it lives at, what it enforces, and the observed refusal or NOT MEASURED with the date it was read and the section it came from.

D.1 Where a guard layer can refuse in a seat run, by stage

Table 21: Where a guard layer can refuse in a seat run, by stage. Read 2026-09-02 from `packages/collective/actions/org-agent/action.yml`, `.github/workflows/*.yml`, `scripts/pre-commit`, `scripts/install-hooks.sh` and `package.json`; the call-site counts are Section 3, The Machine: architecture of a governed seat.

Stage of a seat run	Layer that can refuse there	Coverage measured 2026-09-02
Before isolation	The fence, denying an out-of-lane write at the syscall	Every generated seat. `packages/collective/src/check-agent-confinement.mjs` makes the fence and `persist-credentials: false` an invariant for any workflow that executes an agent and checks out the repository
Before the commit	The action's own guards (the dash guard, the org-secrets scan, the memory guard), then the caller's optional `guard-command`	Two of the nine call sites in eight workflow files pass one, `autonomous-content.yml` line 211 and `engineering-weekly.yml` line 150; the other seven, five seats plus the marketing seat's two report steps, pass none (Section 3, The Machine: architecture of a governed seat)
At the commit	`scripts/pre-commit`, installed as a symlink by `scripts/install-hooks.sh`, reaching a hosted runner only where `npm ci` executes the package's `prepare` script	`CLAUDE.md` item 47 records five workflows running the suite through it; which of the nine call sites install it is NOT MEASURED
After the push	`.github/workflows/ci.yml`, whose 9 jobs run once the commit has landed on `main`	Every call site, and after the fact: CI cannot prevent a landing, only report on one

D.2 Maturity of the ten principles

Table 22: Maturity of the ten principles from the receipts in Section 7, Design principles and a retrofit checklist; test files by `ls __tests__/scripts` and `ls tests`, 2026-09-02. A stays-quiet case was verified only for `check-*.test.ts` files (P4).

Principle	Specified in	Implemented	Negatively tested	Positively observed	Unmet
P1 positive observation	`packages/collective/actions/org-fence/action.yml`	yes	`__tests__/scripts/org-fence.test.ts`	canary red twice, neither on purpose, neither shown to be a real loss	32 of the 65 recorded seat runs are dated before the fence commit of 2026-08-13, so no denial was observed inside them (2026-09-02)
P2 NOT OBSERVED is a verdict	`packages/collective/src/daily-pulse.mjs` header	yes	`__tests__/scripts/daily-pulse.test.ts` (lesson 35)	the pulse printed NOT OBSERVED for red runs from its first run until its scope was added	NOT MEASURED
P3 exit code is the verdict	`scripts/test-gate.sh`	yes	`__tests__/scripts/test-gate.test.ts`, re-injected regex (lesson 47)	NOT MEASURED: no field red of the gate is recorded	NOT MEASURED
P4 negative case	`scripts/check-doc-claim-parity.cjs` check 6	yes	proven by injecting its failure (`CLAUDE.md`, 2026-07-23)	`OK 91` of 91 on 2026-09-02	check 6 warns and never blocks: it sits under a header marked `[WARN]` and calls only `warn.push` (2026-09-03)
P5 sandbox and gate	`docs/audit-dimensions/governance.md`	scheduled seats by `keep_paths`	none	human half on 2026-06-24; unattended report-only landing not observed	interactive sessions are bounded by nothing (2026-09-02)

Principle	Specified in	Implemented	Negatively tested	Positively observed	Unmet
P6 human-only set	`.claude/hooks/arc-valve.sh`, `.claude/settings.json`	seven classes, by named command shapes	`tests/test-hooks.sh` (its cases NOT MEASURED here)	one denial, the installer's self-test, 2026-08-23	whether `push.default` in the agent's environment makes a bare push reachable is NOT MEASURED (2026-09-02)
P7 spec is the source	D-0040; `packages/collective/src/generate-seat-workflow.mjs`	`--heal` then `--check` in pre-commit	`__tests__/scripts/seat-workflow-generation.test.ts`	no reverted hand edit recorded	NOT MEASURED
P8 judge from outside	`packages/collective/src/check-org-liveness.mjs`	yes	`__tests__/scripts/check-org-liveness.test.ts`; five deploy-gap cases (lesson 34)	eight-day false red, 2026-07-24 to 2026-07-31, per its own header	NOT MEASURED
P9 falsifier and register	`docs/org/decision-log.md`; `packages/collective/src/check-owner-directives.mjs`	the register guard	none under `__tests__/scripts` for the register guard	2 past due on the 2026-09-01 pulse	no guard reads the log's predictions or check dates; the one prediction check reads department proposal files (2026-09-03)
P10 suspend, not widen	`packages/collective/src/evaluate-seats.mjs`, `evaluate-ladder.mjs`	yes	`__tests__/scripts/evaluate-ladder.test.ts`; none for `evaluate-seats`	pause held on 2026-09-01; one stage advance, `c8d3068c4`	NOT MEASURED