ASSAY —

THE METHODOLOGY

Work held true. To assay a metal is to test its purity rather than take the seller's word. Assay applies the same standard to agent-fleet software work: a toolkit and a set of conventions that put every claim of done behind a machine-checkable gate.

STATUSGEN V0.9.1

SEPTEMBER 2026

ABSTRACT

A fleet of coding agents produces two things: changes, and claims about those changes. The changes are checkable by ordinary means — tests, review, the compiler. The claims are not, and at fleet scale the claims are what a human reads. A status board assembled from agent-written summaries reports the state of the narrator rather than the state of the system, and it fails silently: nothing in the repository looks wrong while it happens.

Assay is a set of conventions and a small toolchain aimed at that one problem. The work unit is a brief written before the work, carrying the checks that will judge it. The lifecycle has five states and two transitions the implementer may not make. The records are append-only files under review. The board is not maintained by anyone — a generator computes it from those records and refuses to emit anything at all when it finds an inconsistency it cannot resolve. Each desk role acts under its own forge credential, so the verdict is posted by a login the change's author cannot post as, and the merge stays a human act. That is an attribution claim rather than a boundary, and §8 says why the stronger version is not available.

This paper describes the model, the mechanism behind each part, and what the model does not achieve — the last of those in a section of its own and in a qualifier attached to most of the others. That is not modesty: the central claim is that a system should be judged by records the reported party could not complete alone, and a paper exempting itself from that standard would be arguing against its own thesis. Every behavioural claim below cites a file in the open methodology repository, listed in the References.

1. THE PROBLEM: THE PARTY THAT DID THE WORK IS THE WEAKEST WITNESS TO IT

Run one agent and you read its output. Run forty and you read a summary. The summary is written by the same session that did the work, which means the one artifact a human consults is the one artifact with a conflict of interest built into it.

Three published measurements set the scale of the problem, and the methodology's own overview cites them as its evidence base (`docs/how-assay-works.md`). A randomised trial by METR found experienced developers **19% slower** with AI tools while believing they were **20% faster** — self-assessment inverted the sign of the effect. The 2025 Stack Overflow developer survey found **66% of developers** name "almost right" solutions as their single biggest frustration: the failure mode is plausibility, not nonsense. And a study of 20,574 real coding-agent sessions (arXiv:2605.29442) found that roughly **91% of agent misalignments require explicit human correction** — they do not self-correct.

Together they describe a specific failure rather than a general one, and the difference matters because the generic version ("agents hallucinate") points at the wrong fix. The session usually does not invent a change it never made. It reports a check it did not run, or ran in a state that is not the state the change will live in — green in isolation, green before the merge, green on a branch that has since fallen behind — and it writes *verified* because verification was the next word in the shape of the sentence. The text is plausible, internally consistent, and about work that genuinely happened, so no reader has a reason to look further.

Two properties make this resistant to reading harder. The error lives in the *relationship* between the claim and the world, and the claim is the only half on the page. And the report of a check that never ran is indistinguishable, in prose, from the report of a check that passed. A two-state instrument — pass or fail — has nowhere to put "I could not look", so it puts it in one of the other two, and both are lies: "pass" when nothing was examined is a silent defect, and "fail" on a transient error is noise that trains a reader to ignore the output (`docs/three-state-instrument-rule.md`).

The obvious repair — have a second agent check the first — is weaker than it looks, and the overview says so in one sentence worth quoting: *a checker drawn from the same fleet doesn't fix this either — correlated failure modes ride along.* The fix has to change **who is allowed to say done**, not how earnestly they say it (`docs/how-assay-works.md`).

The party that did the work is the weakest available witness to whether the work is correct, because it shares every blind spot the work has. Everything below follows from taking that literally, and the shape it produces has a name in this

paper: a **non-author record** — a record of what happened that the party being reported on could not have completed alone. The brief, the lifecycle, the registers and the board are four attempts at one.

2. THE UNIT OF WORK: A BRIEF THAT CARRIES THE CHECKS THAT WILL JUDGE IT

The unit is a **brief** — one file, authored before the work starts, stating the scope, the dependencies, the deliverables, and the checks that will settle whether the work is done. A conforming brief carries six required body sections (`Context`, `Ground rules`, `Task`, `Verify`, `Evidence`, `Review`) and a frontmatter block whose required keys include the wave, the dependency edges, the effort, the gate, and a four-answer risk block — regulatory, customer, irreversible, sensitive-data (`spec/brief-v1.md` §3.1, §4).

One frontmatter rule removes a judgement call that would otherwise be made under pressure: **the gate is derived from the risk, never chosen beside it.** If any risk answer is yes, the gate is `human`. Splitting a brief follows the same principle — the child takes the stricter gate and the higher risk of the pair, and a downgrade is a hard problem rather than a warning — so the rule fails safe toward more gating (`spec/brief-v1.md` §3.1, §3.4).

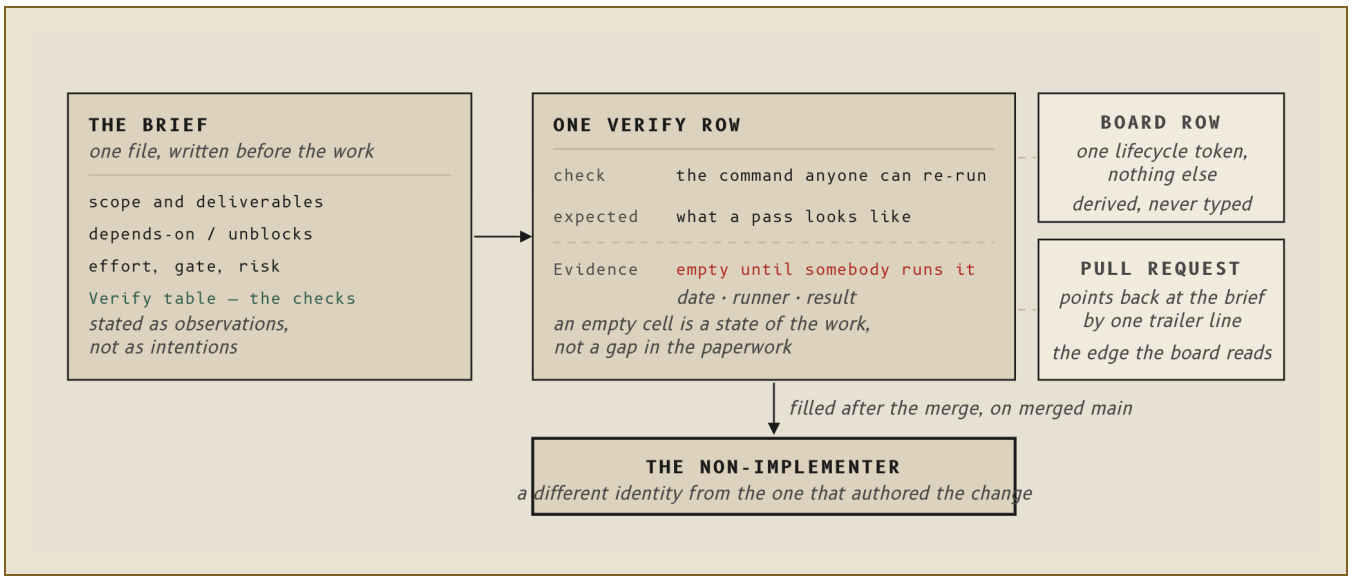

FIGURE 1 — the brief is written before the work and carries the checks that will judge it; the Evidence cell stays empty until an identity that did not author the change runs the row.

The checks live as a table of **Verify** rows. Each row names a literal command a non-implementer can run and an expected exit code or output match; a prose-only row is not a row [`spec/brief-v1.md` §4.4]. The brief-authoring rules put it more bluntly: a row with no literal command and no expected exit or output is not a definition-of-done item, it is a hope [`docs/brief-rules.md`, rule 7].

Writing the checks first is the load-bearing part, and the reason is narrower than the familiar advice about acceptance criteria. A check written *after* the work is written by someone who now knows what the code does, and it converges on the behaviour that exists rather than the behaviour that was wanted. A check written before is a commitment made while its author is still ignorant of the implementation's shortcuts. Across a fleet this stops being a matter of discipline and becomes arithmetic: the number of checks written after the fact, by the party that wants the row to go green, is the number of checks that go green.

Several rules exist because the ways to write an unfailable row recur. A presence check — a word count above a threshold, a grep for the right tokens — passes on any quantity of garbage carrying those words, and the rules name that as checkmark-work rather than a check. A row adding a new guard must carry a mutation entry that reddens it. And a family of row-runner lint rules, tagged in the diagnostics they emit, exists because each names a shape where "**the harness silently substitutes its own answer for the one under test, and the row goes green either way**" — none of them discoverable from a passing run, which is the whole argument for making them lint rules rather than review vigilance. The rules are equally honest about where they stop: the rule asking for rows that *dereference* a claim rather than observe that a file exists states that **no lint enforces it, and that this is a declared limitation rather than an omission**, so a brief can clear every mechanical check and still ship a table that cannot catch a false statement [`docs/brief-rules.md`, rules 8, 16, 25-29, 43].

THE EVIDENCE SECTION IS A LOG OF RUNS

Beside each Verify row is an Evidence entry — the half of the brief that makes it a non-author record — and the specification for it is narrower than "notes on testing". The Evidence section is a **log of runs**, and each entry is an **execution witness.** `statusgen verifyrun --brief <path>` executes each Verify row's command in a fresh subshell at the repository root and appends one row per Verify row [`docs/brief-rules.md`, rule 36]:

```
| # | Command | Result | Output | Date | Runner |
|---|---------|--------|--------|------|--------|
| 1 | `go test ./... > /tmp/out 2>&1; echo rc=$?` | pass exit=0 | sha256: 6f1a0b3c9d22 |
2026-08-13 | human: alex @ b988d1753038 |
```

Read that row left to right and the design is visible in it. `Command` is the command as authored in the Verify table, reproduced character for character, so a later run can show the row has not been edited since. `Result` carries a state — one of three — and an exit code. `Output` is the first twelve hex of a hash of the combined output streams. `Runner` is an identity and the twelve-character commit the tree was at, suffixed when the tree was dirty or when cleanliness could not be determined. Runs append and never overwrite, because an implementer's run, an independent re-run and a re-verify at a later commit are three separate facts, and rewriting the section in place would let a green re-run erase a red one.

Two properties of that row carry the design. **The runner is derived from the executing identity, never supplied** — the reason is given in six words, *a witness you can caption is a witness you can forge*, and a field the author fills in is a field the author chooses. And **a witness is evidence, not an attestation**: whoever controls the process controls the environment it reads, and the output fingerprint is a value two people can compare rather than a tamper-proof seal.

The witness has three states rather than two — `pass`, `fail`, `could-not-run` — and the checking tool's exit codes keep them apart deliberately: 1 says *the work is wrong*, 2 says *the instrument did not look*, and a consumer that collapses them has lost the distinction the scheme rests on. When no identity can be derived at all, the tool writes nothing and exits 2 rather than emitting an unattributed row. And a `could-not-run` row is not coverage: **a row nobody could run cannot be laundered into coverage by recording that nobody could run it** [`docs/brief-rules.md`, rules 36-37].

REGISTERS: THE MEMORY THAT RESISTS ITS AUTHORS

Around the briefs sit append-only registers — what invalidated a plan, what raw ideas are queued, what the product was asked to do, what a design decision settled. The normative model is one file per entry, so parallel authors never contend for the same lines of the same file, and the aggregated views committed at the repository root are generated output with a single writer rather than sources [`spec/registers-v1.md` §2].

Retraction is a tombstone, not a deletion: the entry file stays and its disposition flips. Deleting it would lose the record and trip the check that compares the working tree against trunk history — a check with a blind spot stated in the same place, since a history comparison silently returns nothing outside a checkout or on a read error. Its clean result means *no deletion observed*, never *no deletion occurred*. Entry identifiers are slugs rather than counters, because a counter would reintroduce the write contention per-entry files exist to remove; the consequence is written as a prohibition rather than left implicit, since there is no sequence and therefore no gap, so **gap detection must not be claimed** [`docs/registers.md`].

The registers also carry a gate designed against its own abuse. A finding — new knowledge that makes a brief wrong — flags every brief it affects and holds them out of the work queue. But filing a finding is unverified input: anyone can write a paragraph. So an unresolved finding on its own only flags and excludes; it takes a recorded acknowledgement to turn it into a hard error that forces the affected brief to be demoted or the finding resolved. Without that two-stage shape, dropping a rival brief back to `todo` costs one paragraph. A malformed acknowledgement is a hard error rather than a silent skip, on the principle that a typo'd gate is a disabled gate.

---

3. THE LIFECYCLE: FIVE STATES, AND THE TWO TRANSITIONS THE IMPLEMENTER MAY NOT MAKE

A brief moves `todo → in-progress → implemented → verified → done`, with `blocked` as a sixth state that is deliberately *not* a position in that sequence — it is off-path, and a blocked brief is not offered as work [`docs/lifecycle.md`; `spec/lifecycle-v1.md` §2.0]. The first three transitions belong to the implementer. The last two do not, and that is where the non-author record is completed.

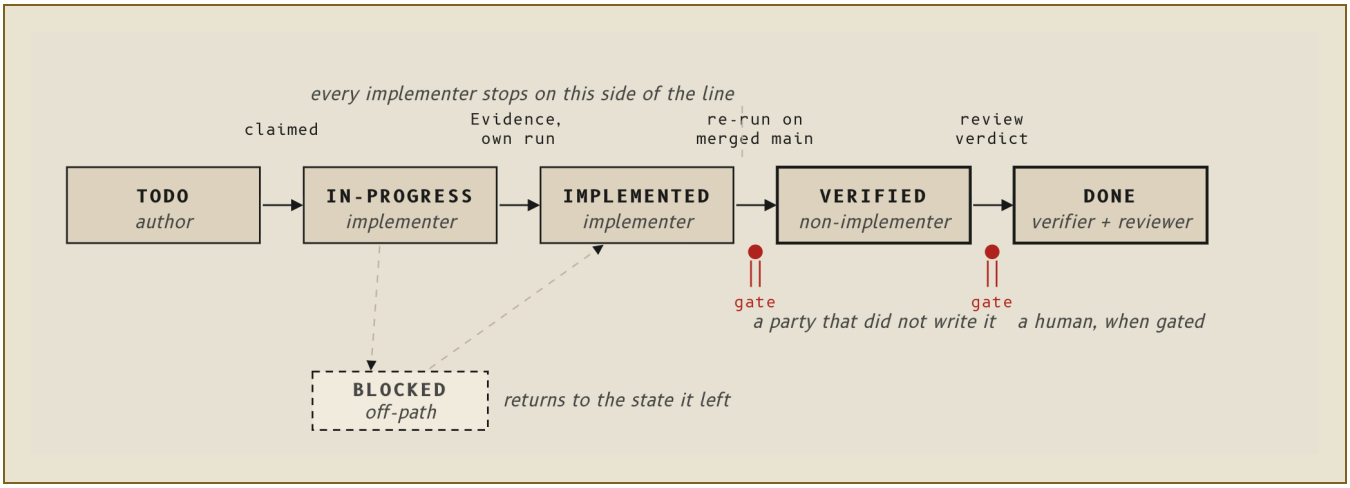

FIGURE 2 — the implementer's authority ends at implemented; the two transitions after it are made by parties that did not write the change, and the second by a named human where the brief is risk-gated.

implemented means the implementer finished and filled the Evidence section with its own run. The stated reason for stopping there is the sentence the rest of the pipeline defends: *an implementer verifying their own work is the narrator grading their own exam* [`docs/lifecycle.md`].

verified means a non-implementer re-ran the Verify table on merged main and filled the Evidence section, dated and naming the runner. Both qualifiers carry weight. *Non-implementer*, for the obvious reason. *Merged main*, for a less obvious one: the implementer's run happened on a branch, in isolation, against a base that may have moved since. Independent re-execution on the merged result is the check that "works on my machine" and "green in isolation" survive contact with the trunk.

done additionally carries the recorded review verdict, and where the brief is gated for a human that entry must name one — a model sign-off does not close a risk-flagged brief. Verified and Reviewed cells take dated entries rather than a bare tick, because an undated tick is unattributable and therefore unauditable. The specification tightens both points: the human marker must sit inside the identity part and be matched on a whitespace-separated token, so that an identity merely *containing* the substring does not satisfy it, and a bare checkmark must not be accepted [`spec/lifecycle-v1.md` §2.6]. Where a brief is flagged irreversible, the human-named review is required *before* `verified`, not only before `done`: a model runner may execute the table, but must not close a change that cannot be walked back [§2.5].

The rule that costs the most to hold is that **merging does not verify.** After a brief's pull request merges it sits in an awaiting-verification queue until a non-implementer is dispatched to run the table. Treating verification as a side effect of merge is how briefs rot at `implemented` with a merged change behind them and nobody having re-run the checks on the state that shipped.

One further distinction stays explicit throughout: the Verify table proves function, the review proves quality, and neither substitutes for the other. Collapsing them halves the number of gates, which is the attraction, and silently drops whichever question the surviving gate does not ask.

WHAT `VERIFIED` DOES NOT MEAN

The specification's §2.4 says so itself, and deserves quoting rather than paraphrasing: `verified` records that an Evidence section exists, is dated, and *attributes* an independent runner. It does **not** attest that the Verify commands were executed, nor that they exited as reported: "there is no execution witness — no recorded command, exit code, or output hash — so a green Verify table is **self-reported**. A conforming implementation MUST NOT describe `verified` as proof of execution" [`spec/lifecycle-v1.md` §2.4].

That clause and the Evidence witness shown above disagree, and the disagreement is worth naming rather than smoothing. The witness table records exactly the command, the exit code and the output hash the specification says are not recorded. The corpus resolves it in its own known-divergences list: the specification describes the reference implementation at `statusgen/v0.8.0`, and for that implementation "no command, exit code, or output hash is recorded for a Verify row, so `verified` attests that Evidence was *authored*, not that it was *executed*" [`spec/README.md`, §"Semantics"]. The witness narrows that gap in later tooling; the normative must-not has not moved, so the weaker claim is still the one to make.

The companion clause is §7.1.2: the non-implementer check is an attribution check on authored text, not an identity check. Nothing in the specification prevents a single actor from authoring both the work and an Evidence row naming somebody else. **Role separation is a convention, and a conforming implementation must not claim it as a boundary.**

Those two clauses are load-bearing in an unusual way. They are the reason the rest of the machinery is arranged as it is: the record cannot be made unforgeable, so the design settles for making a false record an authored one — something a party had to write into a file that lands in a reviewed diff, beside

the tree it names, where a second person can re-run the command and compare. That is weaker than it sounds and the paper does not upgrade it: §4 records a path on which one session holds both credentials, and nothing here closes it.

4. ROLES: FOUR DESKS, A COORDINATOR, ONE HUMAN STEP, AND WHAT AN IDENTITY CAN DO

The lifecycle says what must happen. The desks say who does it. A running fleet is organised as a small number of standing role sessions, each driving one stage, each acting on the forge as its own credential (`plugins/assay/skills/`).

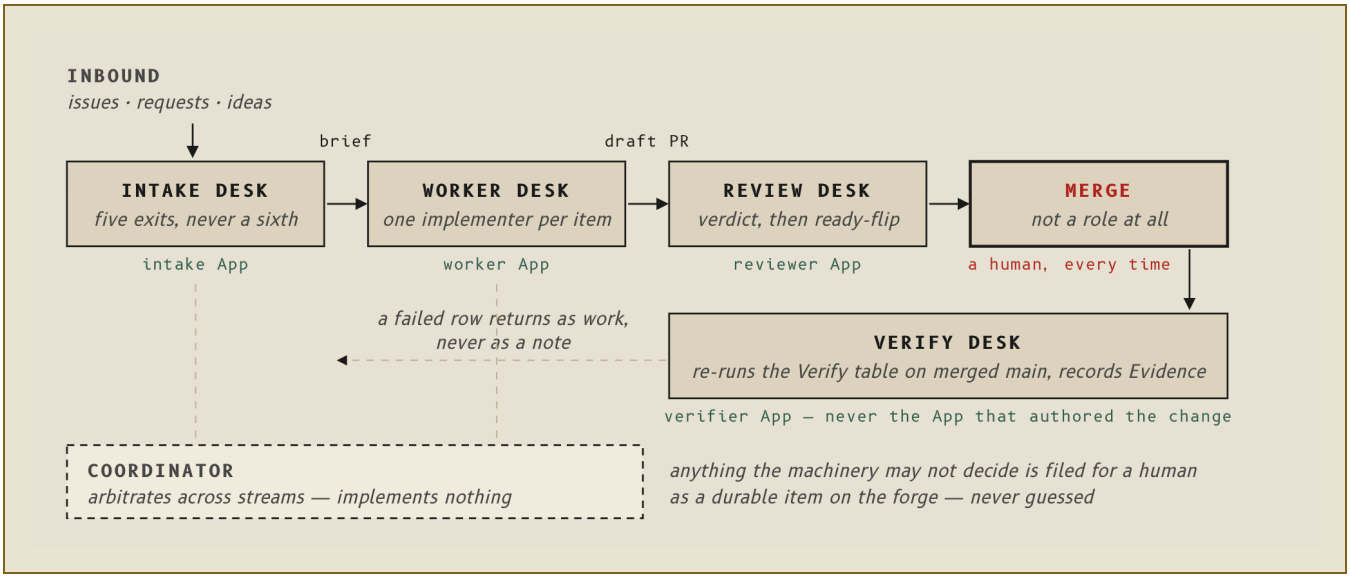

FIGURE 3 — four desks in a line with a coordinator across them, and one box that is not a role: the identity that writes a change and the identity that certifies it are different logins, and the merge belongs to a human every time.

The **intake desk** is the front door. Everything inbound — issues, requests, raw ideas — leaves it as exactly one of five tracked exits: a brief, a bug, a finding, a decision filed for a human, or a recorded rejection, and an item that lands with none of them or with two is a refusal. The value is not the triage; it is that "I read it and formed an opinion" is not one of the exits.

The **worker desk** turns the next-up batch into implementer agents and draft pull requests, one item per branch per pull request, each in its own workspace. It opens the pull request carrying the trailer that links it to its brief, and never hand-edits a stream's own status table.

The **review desk** produces the correctness verdict and, when the verdict is clean and the checks are green, flips the pull request out of draft; it never edits a board row. The flip is mechanical rather than discretionary: a tool re-reads every condition — approval at the current head, green checks, a mergeable state, a security verdict at head for a risk-classed change — and the desk honours its refusals rather than re-deriving the list [`plugins/assay/skills/pr-review-desk/SKILL.md`].

The **verify desk** runs post-merge: it re-runs the brief's Verify table on merged main, records the Evidence, and advances the row. Its own framing of why it exists — *merging is deployment frequency, not completion* — and it never approves, never flips ready, and never merges. A **coordinator** role sits across the four, arbitrating between streams and authoring; it implements no stream and runs no other role's loop inline [`plugins/assay/skills/verify-desk/SKILL.md` , `plugins/assay/skills/the-desk/SKILL.md`].

Between the review desk and the verify desk sits the step that is not a role at all: **a human merges.** The overview's pipeline diagram names it as a node and labels it a deliberate checkpoint rather than an automation gap [`docs/how-assay-works.md`].

WHAT THE IDENTITY SPLIT ENFORCES, AND WHAT IT ONLY ATTRIBUTES

This is where descriptions of a system like this usually overclaim, so the methodology's documentation retires the words that exceed the evidence. *Tamper-evident*, *cannot be forged* and *impossible to self-certify* are explicitly out [`docs/adopting-assay.md` §1a]. The claim that survives is one sentence: **approval is posted by a separate identity the author cannot impersonate.**

Three recorded reasons for the weaker form, each a real path rather than a theoretical one. An implementer session that can mint the reviewer credential's token can drive approve, ready-flip and merge on its own pull request: the identity is genuine, the separation is not, because whoever holds the credential holds the property. Without server-side branch protection nothing on the server requires a merge to wait for the verdict, so the merge gate is advisory. And where agents and a human act through one shared account, no downstream check can tell which acted — a verdict attributed to that account attributes to both.

What remains is still worth installing: a review posted by a distinct credential names an identity answerable for the verdict that a plain implementer session cannot post as. That is strictly more than a self-written checkmark, which is what

most fleets have, and it is not proof that the review happened, was thorough, or was independent [`docs/enforcement-model.md`].

The separation is held by two layers rather than one, and their independence is the point [`docs/enforcement-model.md`]. The first is the forge's own refusal to let a pull request's author approve it: server-side, keyed on pull-request authorship. The second lives in a different component and fires at a different time — before the desk tool posts a verdict it compares the posting identity against the author of the head commit and refuses on equality, naming both [`deskkit.AssertNonAuthorVerdict` , wired into the verdict-posting verbs]. It keys on a different signal, which is why it catches the collapsed-identity case where the forge's authorship-keyed refusal may never fire, because the certifier and the author are one actor on a route the forge does not treat as self-approval. The assertion is three-state like everything else: poster is not author is a permit, poster is author is a refusal, and an unreadable head-commit author is a could-not-check that falls back to the pull request's author and warns rather than passing silently. Removing the second layer is pinned by a mutation whose deletion reddens the poster-equals-author case while leaving the distinct-author case green.

Grounding the model in code rather than prose corrects one intuition. The boot preflight applies the same required permission set to every role credential and refuses to run a role missing any of it, so the question a smaller install answers is never *which role needs fewer permissions* — they need the same — but *how many distinct identities must exist at all.* Exactly one merge destroys the central claim, implementer into reviewer; every other collapse trades audit-trail attribution for a smaller identity set without touching it.

---

5. THE BOARD IS DERIVED, AND A BLOCKING PROBLEM STOPS IT

A status board that somebody maintains is a second system that can disagree with the first, and it is nobody's non-author record: whoever keeps it is reporting on work they are close to. This one is generated: a single binary reads the stream tables, the brief frontmatter, the Verified and Reviewed cells and the Evidence blocks, and writes the board file [`statusgen/README.md`].

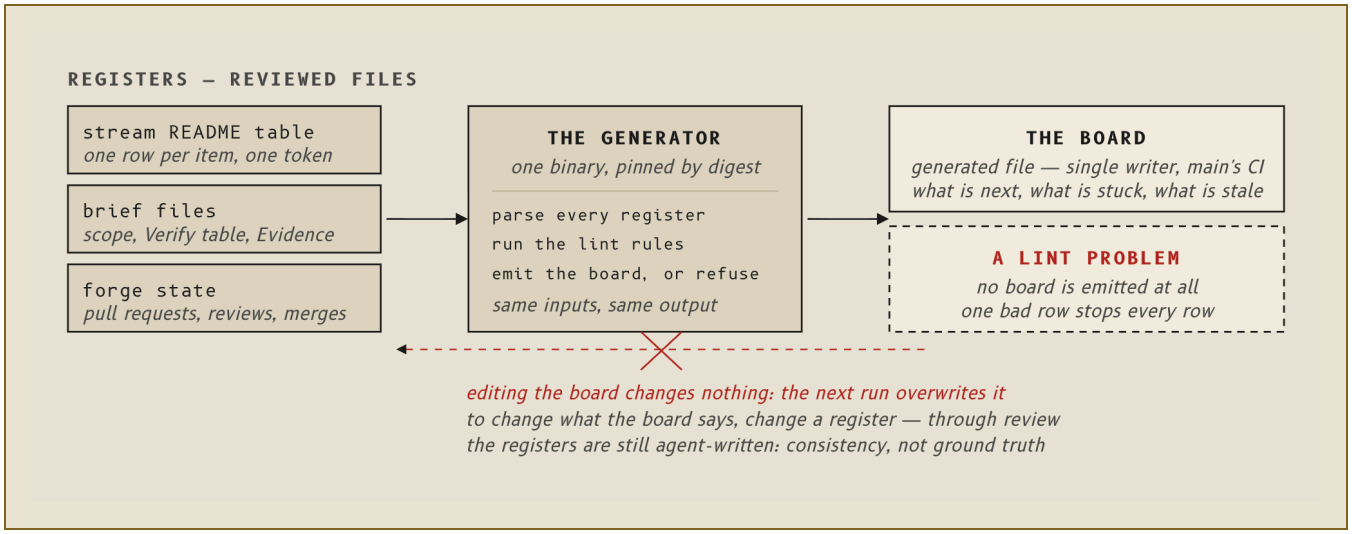

FIGURE 4 — the board is output, so editing it changes nothing; the only way to change what the board says is to change a register through a reviewed pull request, and the registers are still agent-written.

One writer. The board has exactly one writer: main's continuous integration, which regenerates and commits it on every push that touches a source. Branches never commit it — pull-request CI runs the generator in a lint-only mode that reads and writes no board at all, and blocks any pull request whose diff touches the board file. The stated reason is mundane and correct: a generated file committed on branches turns every concurrent pull request into a merge conflict, and one writer eliminates the conflict class outright [`docs/lifecycle.md`]. The second-order effect matters more: because the board is output, an agent that edits it to say what it wishes were true has edited a file the next run overwrites, and the only route to changing what the board says runs through a register — a reviewed file in a reviewed pull request.

A blocking problem stops the whole board. The generator's blocking checks short-circuit before the view is built, on the reasoning that a broken source set cannot produce a meaningful next-up batch; advisory notices still surface and never affect the exit code, and one problem class is deliberately off-board — it reddens the run without suppressing generation [`statusgen/main.go`]. For the blocking set this is the harder of the two available designs: a board that renders with one wrong row is read as a board, and the wrong row is the one nobody looks at twice.

The board says when its own oracle is suspect. Every run reporting a problem also prints a provenance line about the binary that produced it: an unstamped development build can redden a board the pinned binary passes, and a stamped

binary behind the pin is itself a stale oracle [`statusgen/channels.go`]. A gate that can be wrong and says so is usable; one that cannot be wrong is a claim.

Coverage is derived, never declared. The unrun-coverage gate states its rule in capitals in its own source: *unrun is derived, never declared.* A Verify row is unrun **unless** a completed Evidence row for that row exists. Silence is unrun. Clearing the state requires adding an artifact — an Evidence row naming the item, dated, with a runner — so **no amount of not writing something can produce a clean board.** A literal marker is honoured only additively: it can move a row *into* the unrun set and never out of it, and there is deliberately no token, field or phrasing an implementer can use to assert that a row was run [`statusgen/unrun.go`].

Two details of that gate transfer. It is scoped to the *transition* rather than the standing state, and evaluated against the merge base, because firing on already-closed briefs would create a live incentive to edit a closed brief's Evidence table until the board went green — precisely the falsification the check exists to catch. And when the base cannot be resolved it grandfathers with a notice, on an argument that is careful rather than convenient: with no base there is no observable transition, so the domain is empty rather than the check permissive.

The scheduler is part of the output, and says when it is degraded. The generator computes the batch to pick next, weighing stream priority and staleness under a per-stream cap, excluding briefs with unresolved findings, and excluding briefs that already have a branch open on the remote so two sessions do not claim the same work. That last exclusion depends on a network read, and when the read fails the batch silently becomes a superset of the truth. The response is the shape to copy: the generator still writes the board — refusing would leave the previous one standing, equally a superset but unlabelled — and stamps a degraded banner naming the cause, prints a notice, and offers callers a flag that exits non-zero and writes nothing instead. A timeout changes how often the degraded path is taken, never whether it is announced. The specification generalises the rule: a conforming generator must publish its cap values and must render the batch as degraded when claim filtering could not run [`docs/lifecycle.md`; `spec/lifecycle-v1.md` §7.2].

The same documentation states the scheduler's weakness in its own voice: scoring by priority plus staleness rewards neglect regardless of *why* a stream aged, and any incidental commit resets a rival stream's clock. The board is a heuristic scheduler, not an oracle.

6. MEASUREMENT: WHAT THE SYSTEM CLAIMS, AND WHAT IT REFUSES TO CLAIM

Everything above buys structure. The question to ask next is what any of it lets you *measure*, and the answer is narrower than the machinery suggests.

The board is **derived from agent-authored artifacts with consistency linting** — not measured from ground truth. The generator parses status tables, frontmatter, Verified and Reviewed cells and Evidence blocks, all of them markdown written by the same agents whose work it reports. It checks the internal consistency of those documents: missing evidence, unresolved findings, malformed gates, citations naming entries no file defines. It does not independently observe that the code does what a brief claims. The defensible sentence is "status is derived from agent-authored artifacts with consistency linting and independent re-verification" — **not** "status is measured, never self-reported". The strong form is false, because the sensors are agent-writable (`docs/lifecycle.md`; the specification's own wording is "derived from authored artifacts with consistency linting", and its §6.1 makes the refusal of the strong form a must-not).

Where that paragraph sits matters as much as what it says. A system built around the phrase *evidence, not assertion* has a standing incentive to describe itself in the strong form; writing the weak form into the document a conformance claim gets checked against is the only mechanism that reliably stops the strong form turning up in a handover note six months later. The instruction is explicit: claim the weaker, true thing.

The same discipline governs the layers built on top.

Traceability is not coverage. A requirements register records what was asked for and what was claimed to satisfy it. A satisfied entry is an author's claim, not an observation, and an acceptance criterion is a sentence, not a check — nothing in the register runs it (`spec/registers-v1.md` §6.4). Its corpus-wide checks landed advisory-first, with one exception: a citation naming an entry no file defines is a problem, because in an append-only register a missing identifier means a typo or a deletion.

A composite score ships with its parts or not at all. The composite productivity figure is the geometric mean of four sub-scores — speed, flow, quality, value — and the aggregation is geometric rather than arithmetic for a stated anti-gaming

reason: it penalises imbalance, so a team cannot ace one dimension by tanking another, and that intent lives in the aggregation function rather than in a policy document. The transparency rule is stated as the product: the score is never emitted bare, but always with the four sub-scores, the raw inputs behind each, and the baseline window each band resolved to, so a wrong normalisation is visibly inconsistent with its own published parts. A dimension whose input is too thin reports could-not-check and is *excluded* from the mean rather than coerced to zero — a zero factor would zero the composite and lie — and the result is then marked incomplete and names what is missing. The baseline is self-relative, better or worse than this organisation's own recent self, and says so rather than implying an industry benchmark [`statusgen/assayscore.go`].

Delivery timing needs a series, and the series has one writer. Two of the four widely used delivery metrics cannot be answered from a point-in-time API read; they need a reproducible series, and history cannot be back-filled from nothing. So the timing substrate is an append-only log written only by main's regeneration job: a branch cannot forge a record, because the log grows only from what main's own CI observed, and a failed network read records nothing and prints a could-not-check line rather than inventing a value [`statusgen/doratiming.go`].

Code-quality metrics are anchored to published definitions rather than a black box, and where no comparable published figure is pinned the cell reads *not measured* rather than carrying a number [`docs/quality/QUALITY.md`].

Aggregate telemetry is off unless switched on twice. It collects nothing unless a command-line flag and an environment variable are both present in the same run, so no vendor default or inherited environment can enable it silently. A payload is integer tallies against a fixed vocabulary — briefs per lifecycle state, lint failures per category, counts of transitions between state pairs — with no field capable of carrying free text. Lint messages inside the tool do contain stream names and paths; the mapping that turns a message into a category label only ever returns one of its own constants, so a message embedding a path is counted, never quoted, and an unrecognised shape is counted as uncategorised. Every armed run prints the full payload before anything leaves the machine. At the time of writing no receiver is stood up: the client ships behind an empty endpoint and an armed run reports that there is nowhere to send it [`docs/telemetry.md`].

7. TRUST: THE ROSTER, QUARANTINE, BUDGETS, AND WHICH LAYER ACTUALLY BINDS

An agent that reads an issue tracker executes instructions written by strangers. The answer has three parts, and a fourth that says what the first three are worth.

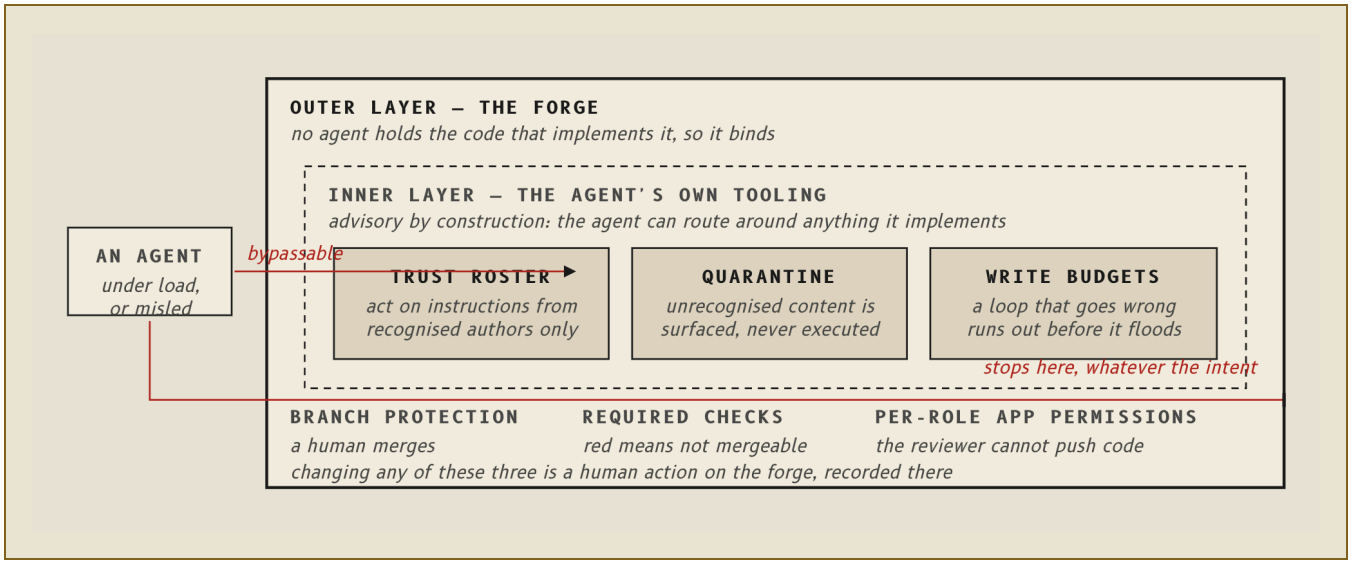

FIGURE 5 — the inner controls are implemented by the agent they constrain, so their real strength is set by the weakest identity that can bypass them; the outer layer binds because no session holds its code.

A roster, not a heuristic. Which identities an agent may act on instructions from is configuration held *outside every reference the tools evaluate*, so a pull request cannot widen the gate that judges it. It also fixes which repositories the tools may write to at all, and an unset set refuses every repository rather than admitting them all. Where a numeric account id is available, login and id must both match the pinned identity, so a recycled login cannot impersonate a trusted account. The acting tools read the configuration file only and never the environment, on an explicit threat model: a tool driven by an agent reading untrusted text is one injected sentence away from an environment override, so that route is closed unconditionally [`docs/adopting-assay.md` §3; `tools/desk/README.md`].

Quarantine that is visible rather than silent. Items from unrecognised identities are counted and listed, and never given an action; the acting verbs refuse on them with a dedicated exit code and an audit row. Public-origin text renders inert there, control characters neutralised, because **titles are data, not**

instructions. A blessing by a trusted maintainer covers the content *as of the blessing* — a body edit, or an untrusted comment added after it, voids the grant — and the implementation compares timestamps rather than pinning content hashes, and says so.

Trust tiers that grant automation, never authority. Identities outside the operator's roster sit in an ordered set of tiers, and what a tier unlocks is small and fixed: review depth, whether CI runs on a fork's pull request without a maintainer approving it, whether automation may act on the identity's items at all, and whether a proxied changelog fragment is accepted. Nothing outside that table is a tier capability, and the highest tier merges nothing. Promotion and demotion are recorded human acts with reasons — no counter promotes automatically, because a burst of trivial accepted changes must not buy workflow auto-approval, and a compromised account is precisely the case where prior good behaviour predicts nothing. The model fails closed: an absent, unreadable or malformed ledger resolves every identity to the default tier, never to something wider [`docs/contributor-trust.md`].

The model draws a line in the other direction too. The tier **rows are never published**. A tier is a judgement about a named person and a demotion is an adverse one; putting those in a public, append-only, never-deleted register would leave a permanent mark on an individual out of all proportion to the workflow problem being solved. What is published in full is the model.

Budgets, and a breaker. Every outward write is charged against two conjunctive meters, a per-item and a per-repository hourly cap, both of which must admit it; only results that may have reached the remote charge, failing closed on anything unclassified. Five consecutive non-progress attempts open a circuit breaker for a cooling period. The budgets came out of a recorded failure rather than a policy preference: the same section documents an inverted meter in which ten *failed* creates locked a tool out for an hour while ninety-nine successes did not move it, and pins the corrected behaviour with a named test [`tools/desk/README.md`]. The bug is in the file, next to the number.

And the honest part. Every one of those controls is implemented by the agent it constrains. The methodology has a vocabulary for exactly this, and it is the most transferable idea in the corpus: a device's real strength is set by **the weakest identity that can bypass it**. An operating-system or harness-level refusal outranks a server-side required check, which outranks a client-side tool refusal,

which outranks a rule written in prose — and the instruction attached is *classify honestly* [`docs/mistake-proofing.md`]. A rule in a document is the weakest device on the table, and its bypass is never having read it.

Seven rules follow from that taxonomy: a control must be shown to fire; fail closed, and could-not-check is not a pass; no silent bypass; warnings do not compose, because each additional warning weakens the ones before it and a standing ignored warning is negative value; retire dead devices; honesty about non-coverage is itself a device; and do not try to mistake-proof judgement. A control that fails open under error, the same page notes, is a warning wearing a control's label.

Two consequences run through the toolchain. A blocked write is a stop signal rather than an obstacle — the exit-code contract separates *refused* from *could-not-verify* precisely so that reaching for a raw command to make the same write anyway reads as defeating a safety gate rather than routing around a bug. And where a bypass must exist it is audited rather than invisible: the self-containment scan that keeps internal identifiers out of public text takes an override that writes an audit row, and the documentation says plainly that this replaces an invisible bypass with an audited one rather than softening the refusal.

---

8. LIMITS: THE METHODOLOGY'S OWN CASE AGAINST ITSELF

Every limit below is drawn from a normative document rather than composed for this paper: they are checked in, next to the rules they qualify. The list is representative, not exhaustive; the specification's own "known divergences" section is longer.

The sensors are agent-writable, and every other limit is downstream of it. Machine-attributable transitions and a hook layer blocking an implementer from writing its own gate cells narrow the gap; nothing closes it while one identity can author both the work and its record [`docs/lifecycle.md`].

`verified` is not proof of execution, and role separation is not a boundary — both stated as must-nots and quoted in §3 [`spec/lifecycle-v1.md` §2.4, §7.1.2]. Attribution is not authorisation: the review gate names an answerable

identity and proves nothing about independence or thoroughness [`docs/enforcement-model.md`].

The reference implementation does not check everything the specification requires. Seven normative clauses are listed as unimplemented, including this one: a brief stripped of four of its six required sections lints clean, and a prose-only Verify row lints clean [`spec/README.md`]. A specification whose reference implementation is weaker than its text is a common situation; naming the exact seven, in the specification, is not.

It also violates its own three-state rule in at least four places, and lists them — including the lint mode itself, which terminates in pass or fail and therefore has two terminal states rather than three [`spec/README.md`].

Conformance is self-declared. There is no conformance test suite, no claim registry and no arbiter; a conformance claim carries exactly the weight of its author's reputation [`spec/README.md`]. The specification is published at draft status, with breaking changes permitted without a major-version bump.

Append-only is enforced by comparison, not by the medium, and the comparison returns nothing outside a checkout or on a history-read error. Its clean result means *no deletion observed* [`docs/registers.md`].

The scheduler rewards neglect, and any incidental commit resets a rival stream's clock [`docs/lifecycle.md`].

The claim read degrades rather than refusing by default, and the resulting batch is a labelled superset [`docs/lifecycle.md`].

Some gates have no lint at all, by declaration — the rule asking for Verify rows that dereference a claim rather than observe a file's existence is enforced by review alone, and the rules say so [`docs/brief-rules.md`, rule 43].

A known credential defect is documented rather than fixed: the token-minting path is installation-scoped rather than repository-scoped, so a minted token's real blast radius is every repository in that installation — recorded as a known defect in the tool's own reference [`tools/desk/README.md`].

The corpus disagrees with itself in two places, and both are live at the time of writing. The per-stream cap on the next-up batch is written as two in the lifecycle document and four in a role definition, so this paper names the cap without a number rather than picking one; the specification requires only that a generator *publish* the values it uses, which is the right resolution. And the

execution-witness clause quoted in §3 is contradicted by the witness table §2 shows, resolved in the specification's own divergences list rather than in either document that states it.

The general rule these instances teach transfers to any system of this shape: **write the limit down next to the mechanism, in the normative document, in the strongest form you believe.** A limit recorded in a review thread is a limit that has already been forgotten. A limit in the document is one the next person to overclaim has to walk past.

---

9. ADOPTION SHAPE

The install is deliberately small, and most of what an adopter puts in place is not a file in their repository. The runbook's own summary of the cost is that it falls almost entirely on identities rather than commands [`docs/adopting-assay.md`].

Two accounts, and one non-negotiable split. The floor is one human account and one machine account, with the implementer identity distinct from the reviewer credential. That pair is load-bearing; everything beyond it is attribution rather than separation.

One pinned binary, verified by digest. The generator arrives as a release binary — not vendored as source, not tracked by a floating tag. The adopter commits a pin file with one line per platform naming a release tag and a SHA-256 digest; the install detects the platform, refuses outright when no line matches, downloads, hashes, compares, and refuses on mismatch, and an upgrade re-pins rather than editing in place so the bump shows in a diff. The refusal is the load-bearing part: an installer that guesses when the pin is absent has swapped a verified artifact for an unverified one at the moment nobody is watching.

Two halves of continuous integration. On pull requests the generator runs lint-only and the board file is off limits; on main it regenerates and commits. Where main is branch-protected that commit needs a dedicated credential with write access to contents and nothing else, added to the branch's bypass — the honest cost of turning an advisory merge gate into an enforced one. The workflow is not shipped: the runbook gives its shape and the adopter writes it.

Draft pull requests, and a human merge. Branch plus draft pull request is the standing authorisation. The runbook keeps a standing list of acts that are never autonomous — creating and installing the reviewer credential, choosing the roster's values, repository creation and permission grants, the merge, the push to the default branch, the release tag, any history rewrite — and is explicit that these are ongoing rather than one-time.

The trust configuration lives outside every branch, for the reason given in §7: a file inside the branch under review can be changed by that change.

Forges and harnesses are treated as substrates, not as the method. The model is written against the properties a forge must provide — distinct identities with separable permissions, server-side branch protection, required checks, and a review object an author cannot post on their own change. The reference deployment is one forge; a second profile is documented, and its governing rule is the one to copy: the second profile must be **at least as secure as the first**, and "weaker but disclosed" is non-conforming [`docs/adopting-assay-gitlab.md`]. The same posture applies to agent harnesses: the capability a role needs is the row, and no vendor's column is the method [`docs/how-assay-works.md`].

---

10. CLOSING

The methodology reduces to one sentence, and the sentence is not about agents: **the record a human reads should be a non-author record — one the reported party could not have completed alone.** Briefs written before the work make the checks something other than a post-hoc rationalisation. A lifecycle whose last two transitions belong to someone else makes *done* an observation by a second party. Registers that resist their own authors make the history survive the people in it. A derived board makes status a computation rather than a report. Separate credentials make the verdict answerable — and the documented account of what that does *not* prove keeps the claim from inflating into one the machinery cannot support.

A reader with no interest in this toolchain can still take the test away, and it costs nothing to run. Look at the artifact your fleet's status comes from and ask who could have written it. If the answer is *the same session that did the work*, the number on the board measures the narrator, and every improvement to the agents will move it without moving the thing it claims to describe. Changing that answer is the whole of the engineering above.

REFERENCES

Every behavioural claim in this paper cites a file in the open methodology repository, `medici-finance/assay`, at https://github.com/medici-finance/assay. Paths are repository-relative, and the repository is the authority — this paper summarises it and does not replace it.

SOURCE	WHAT IT GROUNDS
`docs/how-assay-works.md`	the three published measurements in §1; the correlated-failure argument; the pipeline with the human merge as an explicit node; the harness-neutral capability framing
`docs/lifecycle.md`	the five states and their owners; implementers stop at implemented; merging does not verify; the single-writer board; next-up weighting and the degraded claim read; the honest claim about the board
`spec/lifecycle-v1.md`	`blocked` as an off-path state; the dated-cell and human-token rules; the pre-`verified` human review for irreversible work; §2.4 on what `verified` does not attest; §7.1.2 on role separation as a convention; §7.2 on published caps and degraded batches
`spec/brief-v1.md`	the six required body sections; the frontmatter keys; gate derived from risk; split-flag conservation; the Verify table's literal-command rule
`docs/brief-rules.md`	rows as commands rather than hopes; presence-check rows; mutation rows; the row-runner lint family; rule 43's declared absence of a lint; Evidence as a log of execution witnesses; the derived runner; the three witness states
`spec/registers-v1.md`	per-entry register files and generated views; what the requirements and decisions registers do not establish
`docs/registers.md`	append-only conventions, tombstones and the history-comparison blind spot; slug identifiers and the prohibition on claiming gap detection; the acknowledgement gate on findings. Marked superseded by the register specification, which governs where the two differ
`docs/enforcement-model.md`	the identity set and the cost of each merge of two identities; the single collapse that destroys the claim; the two independent layers and the non-author verdict assertion; the uniform required-duty set applied at boot

SOURCE	WHAT IT GROUNDS
`docs/adopting-assay.md`	§1a's retired adjectives and the surviving sentence; the identity floor; the pinned, digest-verified install; the two-half CI shape; the standing list of human-only acts; the roster held outside every evaluated reference
`docs/adopting-assay-gitlab.md`	the second forge profile and its at-least-as-secure rule
`docs/mistake-proofing.md`	the device taxonomy; the bypass axis that ranks a prose rule last; the seven device rules
`docs/three-state-instrument-rule.md`	checked-clean, checked-failed, could-not-check; truncation; the positive-control requirement
`docs/contributor-trust.md`	the ordered trust tiers; automation rather than merge authority; recorded human promotion; failing closed; why the rows are not published
`docs/telemetry.md`	off by default; two independent switches; counts-only against a fixed vocabulary; why a payload cannot carry an identifier; no receiver stood up
`statusgen/README.md`, `statusgen/main.go`, `statusgen/channels.go`	the three run modes; the single-writer model; blocking problems short-circuiting the board; the binary-provenance caveat
`statusgen/unrun.go`	unrun derived and never declared; the transition-scoped evaluation and its incentive argument
`statusgen/assayscore.go`, `statusgen/doratiming.go`	the geometric composite and its transparency rule; three-state dimensions excluded rather than zeroed; the single-writer append-only timing series
`docs/quality/QUALITY.md`	published-definition anchoring and the not-measured cells
`tools/desk/README.md`	the write verbs and the exit-code contract; draft-only construction; the roster, quarantine and blessing rules; the write budgets and the circuit breaker; the audited scan override; the documented token-scope defect
`plugins/assay/skills/`	the role definitions in §4 — intake, worker, review, verify, and the coordinator
`spec/README.md`	the draft status; the unimplemented normative clauses; the admitted three-state violations; self-declared conformance

Two notes on sourcing, in the spirit of §8. This paper describes the model as the repository documents it: it is not an audit of a running fleet, and no number in it is a measurement taken for this document. And where a page above is itself marked superseded by a versioned specification, the specification governs and the prose here follows the specification.