Verified Bookkeeping in the Fablyos Story Engine

Beat evidence, ledger supersession and slip-learned fact selection for long-form interactive fiction

Author: Kevin Lackey · Kevin@fablyos.com

Status: technical white paper, Revision 1 (2026-11-03). **Reading order:** the terms (§2), the conceptual architecture (§3), then how Fablyos implements it (§4), then every technical feature in detail (§5).

Contents

- Abstract
- 1. Introduction
- 2. Terminology
- 3. Conceptual architecture
- 4. Implemented architecture
- 5. Technical features
- 6. Relation to Layered Continuity Architecture and prior work
- 7. Evaluation status and limitations
- 8. Implementation reference
- 9. Future work
- References

Abstract

Fablyos writes an interactive story one short passage at a time with inexpensive models. A second model, the bookkeeper, records what each passage changed: which planned story beats happened, which facts became true or stopped being true. That bookkeeper is cheap, fast and often wrong, and every downstream component trusts what it records. This paper describes two mechanisms that stop its errors from compounding.

The first is **beat bookkeeping with a quote-check verification loop**. A bookkeeper's claim that a plot beat happened is recorded only if a separate judge can copy a sentence of at least six words from the story that shows the beat, and that sentence survives deterministic checks: it must occur verbatim in the source,

name the beat's people, not be a character's hindsight retelling, and, for fight beats, coincide with a combat round the game actually rolled. Event closure is decided once, last, over the beats that survived.

The second is a **layered facts ledger with supersession and slip-learned selection**. The verbatim passages are the authoritative record; the ledger is a regenerable view folded from per-passage updates, so Rewind and Edit roll it back exactly. Each ledger line carries a source pointer to the passage that wrote it. A write-time check replaces a fact that a new line flatly contradicts. A per-story learning record tracks which kinds of ledger line have caught continuity slips, and reweights which lines the continuity checker is shown.

Both mechanisms use no new tables, no embeddings and no network beyond the model calls the story already makes. Everything lives inside each story's encrypted turn data. Section 7 states plainly what has and has not been measured.

---

1. Introduction

In a long interactive story, the expensive failures aren't bad sentences. They are **bookkeeping failures that later passages build on**. If an event is credited before its scene plays, the story skips the scene, pays out its XP and opens the next arc. If a fact is recorded but never retracted, the writer is told two contradictory things are both true. If a continuity checker is shown the wrong 16 of 150 ledger lines, it misses the broken arm that now swings a sword.

Fablyos runs on a deliberate cost constraint: pay-per-use models through OpenRouter, free accounts capped at under $0.25 of AI spend a day, and a bookkeeper call after every passage. A model that cheap gives a confident wrong answer often enough to matter, so trusting a better model is not the fix. The engine treats every model output that changes durable state as **a claim to be checked**, and checks it with the cheapest instrument that can do so. That means string matching first, then a narrow yes/no call, and a full judgement only when nothing cheaper will do.

1.1 Contributions

1. **Evidence-gated beat recording.** A plot beat counts only when there is a verbatim quote from the story showing it, checked against six deterministic and model gates (§5.1.4).
2. **Single-point event closure.** Ordering inside the fold is fixed so that an event closes once, over verified beats only, and never in a passage that just rejected one of its beats (§5.1.5).

3. **A ledger with provenance and write-time supersession.** Each fact knows its source passage. A conservative contradiction detector retires the fact a new line overturns (§5.2.2, §5.2.3).
4. **Slip-learned selection.** The lines shown to the continuity checker are reweighted per story by which kinds of line have actually caught the writer out, with Bayesian smoothing and hard clamps (§5.2.4). Recall of older passages is reweighted in the same way, by whether the writer used them (§5.2.5).

1.2 Claim discipline

Each mechanism below is labelled by its status in the code. **Implemented** means it is on the `dev` branch, and covered by unit tests. **Field-observed** means a playtest story exposed the failure and a fix shipped against it. **Measured** means a before/after number exists. As of this version, nothing in §5.2 is *measured*. The replay harness that would measure it (`continuity_eval.py`) exists, but the comparison has not been run on real stories. Section 7 sets out what would count as evidence.

---

2. Terminology

The terms below have a specific meaning in Fablyos. Each one names an object or mechanism in the code, and the rest of the paper uses them in exactly this sense. The **Usage** column gives the code symbol or the place where a reader will meet the term.

2.1 Record and roles

Term	Definition	Usage in Fablyos
Turn	One entry in a story's record: a player line or an AI passage. The unit that Rewind and Edit act on.	`Story.turns`; each has `tid`, `role` (`player` or `ai`), `text` and, once bookkept, `upd`
Passage	An AI-written turn of 80–150 words. Numbered as the player sees them.	Source pointers and age tags count passages, not turns: "[3 passages ago]"
Writer	The model that drafts each passage from the prompt context.	`_stream_turn`; temperature 0.75
Bookkeeper	The model that reads new passages and proposes what changed, as JSON. It proposes; it never writes state.	`start_bookkeeping` (background thread), `_fold_job`, rules in `engine.py`
Update (`upd`)	The verified record of what one passage changed: beats, facts, presence, position, items, clock. Stored on the last turn it covers.	`turn["upd"]`, written by `_fold_apply`
Fold	Rebuilding a view by applying every `upd` in turn order. Derived state is always folded, never stored.	`folded_updates()`, `memory()`, `E.replay()`, `beat_progress()`
Bookkept boundary	The index of the last turn the bookkeeper has folded. Views stop here; later turns are pending.	`meta["mem_upto"]`
Card	A world or story entity: character, monster, location, event, arc, style. *Live cards* are the base cards with every `card_updates` replayed.	`Story.live()`, `E.render()`
Rewind / Edit / Retry	Player controls. Rewind returns the story to an earlier turn and drops the rest. Edit changes a turn already written. Retry rewrites the latest passage. Because every view is a fold, all three undo state exactly.	`Story.rewind`; `/rewind` and `/retry` routes in `server.py`

2.2 Plot structure

Term	Definition	Usage in Fablyos
Event	A planned story unit on a world card: arc, order, optional place, optional story-clock time, optional level requirement, and an ordered list of beats. *Main* events carry the arc; *side* events are optional.	`open_events()`, `is_side()`
Arc	A group of events. An arc opens only when the previous arc's last main event is done, never by level alone.	`arc_end_line()` writes "[Arc complete]"
Beat	One sentence describing a scene inside an event. Beats play in order.	Keyed `"<event id>#<n>"` in `beats_done`
Colour aside	An optional detail attached to a beat, "(Colour if it fits ...)". It never holds a beat back.	Stripped before the fight test in `beat_needs_fight`
Beat progress	How many beats of an event have been recorded: the highest *n* across all folded `beats_done`. Reset by "Didn't happen" (`events_reopen`).	`Story.beat_progress()`
Next beat	The first unrecorded beat. It is the only future beat the writer is shown.	`beats_text()`: ALREADY HAPPENED / NEXT BEAT / count held back
Focus	The story's pacing setting: Tight, Balanced or Sandbox. It sets how hard the main story pulls and when beats stall.	`FOCUS`, `STALL_PASSAGES`
Director	The mechanism that orders the next beat as a required passage once the hero and the beat's people have been in place long enough.	`director_due`, `director_order`; log line "[Director]"
Beat pull	A light hook toward a beat whose place is elsewhere. It never moves the hero.	`beat_pull`, every 4 passages, at most 4 per beat
Stall / passed beat	A beat that hasn't happened in 20 (Tight) or 30 (Balanced) quiet passages is *passed* so the story moves on. An event closed this way pays no event XP.	`beats_passed`, `events_stalled`; log line "[Beat passed]"

2.3 Verification

Term	Definition	Usage in Fablyos
Claim	A bookkeeper assertion that changes plot state, such as a beat or an event done. It is not recorded until it is verified.	`upd["beats_done"]`, `upd["events_done"]` as proposed; `beats_claimed` in debug
Beat judge (story check)	A temperature-0 model call that says which unrecorded beats have happened, with a quote for each.	`resolve_next_beat(e, evidence=True)`
Evidence quote	The exact sentence offered as proof that a beat happened: asked for at six or more words, accepted at four or more, found verbatim in the summary, facts or recent passages.	`EVIDENCE_RULE`, `_ev_norm`, `_beat_quote`
Gate	One accept/reject test on a quote: fight, verbatim, name, hindsight, relevance. The first gate to fail stops the walk.	Reasons recorded as `why` in `beat_trace`
Told, not shown	A beat that a character only narrates as past ("since the night the log-out button vanished"). It doesn't count as played.	`told_not_shown()`, `HINDSIGHT`
Unbacked	An event whose beat the quote check rejected in the current passage. It can't be closed in that passage.	`unbacked` set in `_fold_apply`
Closure	Recording an event as done. Decided once, last, over the beats that survived.	`_close_finished_events()` ; log lines "[Event done]", "[Event complete]"
Sweep	A periodic look for beats played on the page that nobody claimed.	`_sweep_beats`, every 4 passages; log line "[Beat check]"
Catch-up	A one-off check, made before a beat is staged again, that it hasn't already happened.	`catch_up_beats`
beat_trace	The per-claim audit record: beat, outcome, reason, quote, and the judge's raw reply.	Playtest `history.json`
Fail closed	A verifier that errors records nothing: the claim stays open and is checked again later. The turn itself is never blocked.	Judge `None` records no new beat; `_quote_shows` failing returns `False`

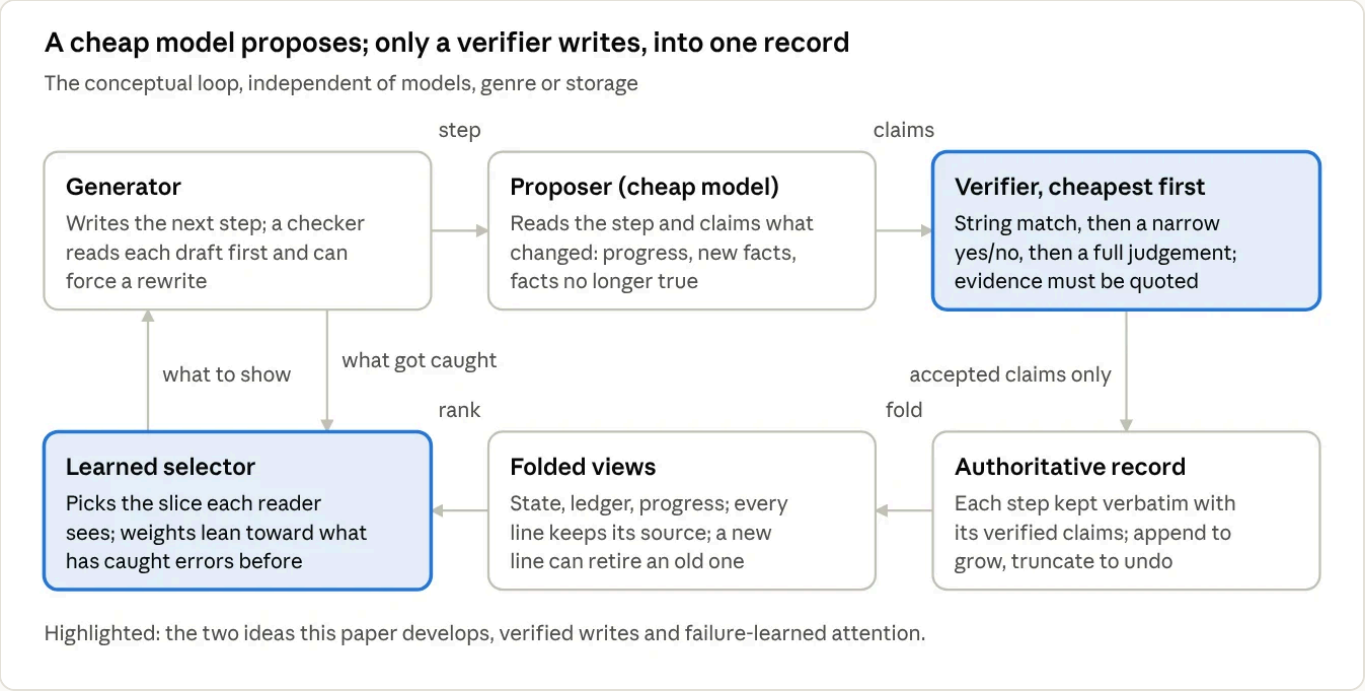
2.4 Ledger and continuity

Term	Definition	Usage in Fablyos
Facts ledger	The folded list of lasting facts (names, promises, debts, injuries, secrets, deaths, where items are), at most 150 lines. Shown to the player under Story notes > Remembered facts.	`memory()["facts"]`, `MAX_FACTS`
Pinned detail	A lasting exact specific the story will need again (a room number, an address). Always given to the writer and never pruned; at most 30.	`memory()["pinned"]`, `MAX_PINNED`
Engine-owned fact	A fact the sheet, the cards or the purse already state. It is dropped from the ledger so there is one source of truth.	`engine_owned_fact()`, `ledger_dropped`
Source pointer	The passage that wrote a live ledger line.	`memory()["fact_src"]`, `_update_refs()`, `fold_sources()`
Age tag	A line's age as the checker sees it: "[12 passages ago]". An old *state* line may have changed without a note.	`aged_facts()`, `FABLYOS_FACT_AGES`
Forgotten line	A line older than a memory-wiping death. It is still true of the world, but the hero no longer knows it.	`forgot_at`, `FORGOTTEN` tag
Supersession	Retiring an existing line because a new line flatly contradicts it, at write time, inside the same passage's update.	`_supersede()`, `contradicts()`; recorded in `upd["superseded"]`
Slip	A contradiction between a new draft and the story so far, reported by the continuity checker as one short line. A slip forces a rewrite with the slip named, at most twice.	`continuity_slip()`, `CONT_REWRITES`
Continuity checker	A small model with no reasoning that reads the draft against the last 3 passages, the cast, the clock (NOW) and the selected ledger lines.	`CONTINUITY_PROMPT`, `FABLYOS_CONTINUITY_MODEL`
Line kind	The class of a ledger line, from ordered patterns: state, knowledge, bond, place, timing, other.	`classify()`, `_PATTERNS`, `_WEIGHT`
Live (name)	Someone in the scene, or named in the draft being checked. Lines that mention them score higher.	`NAME_BONUS` in `select()`
Selection	The ranked choice of about 16 ledger lines handed to the checker.	`checker_facts()`, `select()`

Term	Definition	Usage in Fablyos
Learning record	Per-story counts of lines shown and lines that caught a slip, by kind and by line.	`meta["ledger_learn"]`, `learn()`
Kind multiplier	A kind's smoothed catch rate against the story's average, clamped to 0.6–1.8. It scales the kind's base weight.	`kind_multipliers()`
Line hit	A slip attributed to one exact ledger line. Each one adds a selection bonus, up to two.	`attribute()`, `LINE_HIT_BONUS`
Recall	Retrieval of older passages that have scrolled out of the verbatim window, as "Earlier in the story".	`recall.search()`, `pick()`
Recall use	Whether the writer drew on a recalled piece: two or more of the piece's own rarer words reappear in the new passage.	`meta["recall_use"]`, `was_used()`, `weights()`
Scene presence	Who is in the scene now (IN THE SCENE) and who is elsewhere (NOT HERE).	`memory()["present"]`, `memory()["away"]`
Story clock	The program-kept day and time, advanced by the bookkeeper's report and capped by rules.	`clock.py`, `upd["clock"]`; the NOW line

3. Conceptual architecture

The design reduces to one rule: **a cheap model may propose changes to durable state, but only a verifier writes them, into a single record that every view is rebuilt from.** Nothing in this section depends on Fablyos, on fiction, or on a particular model. It applies to any long-running agent whose own bookkeeping it can't afford to trust.



Conceptual architecture · model-agnostic

The loop has six parts. A **generator** produces each step, and a checker reads the draft before it is kept. A **proposer** reads the kept step and claims what changed. A **verifier** accepts or rejects each claim, using the cheapest check that can decide it. The **record** keeps every step with its accepted claims. **Views** are folded from the record on demand. A **selector** decides which slice of those views each reader sees, and learns from what the checker catches.

3.1 Principles

1. **The record is the authority; everything else is a fold.** Derived state is never stored on its own. Undo is truncation, so no compensating delete exists to get wrong.
2. **Every durable write is a claim.** The proposer never writes state directly. A claim that isn't verified is not recorded, and it stays open for a later step.
3. **Evidence, not assent.** A claim is accepted only with a verbatim excerpt from the record that shows it. A verifier that agrees without quoting has not verified anything.
4. **Cheapest sufficient check.** Deterministic string tests run before model calls. A narrow yes/no call runs before a full judgement. Expensive checks fire only on the paths that need them.
5. **Decide once, last.** Decisions derived from claims, such as "this event is finished", are made after every check, over the claims that survived. A rejection in the current step blocks the derived decision in that step.

6. **Views carry provenance and correct themselves.** Each derived line knows the step that wrote it. A new line that contradicts an old one retires the old one at write time, and the retirement is recorded in the record itself.
7. **Attention is learned from failure.** When a reader can see only a slice of the views, choose the slice by what has caught errors before. Weights start neutral, are smoothed and bounded, and are learned per record.
8. **Fail closed, trace everything.** A failed verifier records nothing, and the claim waits for the next check; it never blocks the user. Every accept and reject leaves a reason that can be inspected later.

3.2 From concept to code

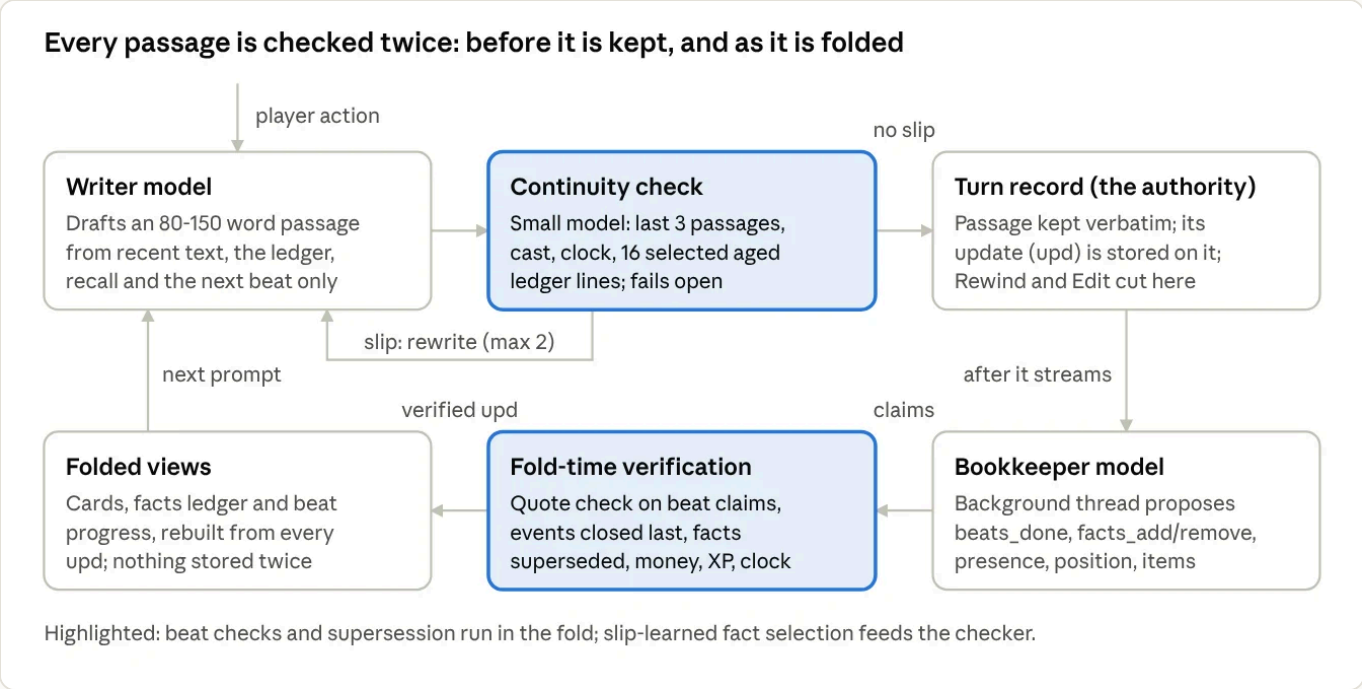
Concept	Fablyos implementation	§
Generator and draft checker	Writer model; continuity check with up to 2 rewrites	4.1
Proposer	Bookkeeper model, run on a background thread	4.1
Verifier, cheapest first	Fold-time checks: the beat evidence judge and the gate chain; supersession	5.1, 5.2.3
Authoritative record	Turn list with a per-passage `upd`, encrypted per story	4.2
Folded views	Cards, facts ledger, beat progress, presence, clock	4.3
Provenance	`fact_src` source pointers and age tags	5.2.2
Learned selector	Kind-scored ledger selection with slip-learned weights; use-weighted recall	5.2.4, 5.2.5
Trace	`beat_trace`, `superseded`, `ledger_learn`, `recall_use`	5.1.4, 7.1

4. Implemented architecture

In Fablyos, the concepts in §3 become four model roles, one authoritative turn record, and a stack of views folded from it.

4.1 Runtime pipeline

A Fablyos story is a list of **turns**: player lines and AI passages, encrypted per story. Four model roles act on it. The **writer** drafts each passage. The **continuity checker** reads the draft before it is kept. The **bookkeeper** runs on a background thread after the passage streams and returns a JSON update. The **beat judge** is called from inside the fold to verify the bookkeeper's plot claims.



Turn pipeline · story.py _stream_turn, start_bookkeeping, _fold_apply

The loop is closed in two places. The checker can reject a draft and force a rewrite with the slip named, at most `CONT_REWRITES = 2` times. The fold can reject the bookkeeper's claims before they are recorded.

4.2 The turn record is the authority

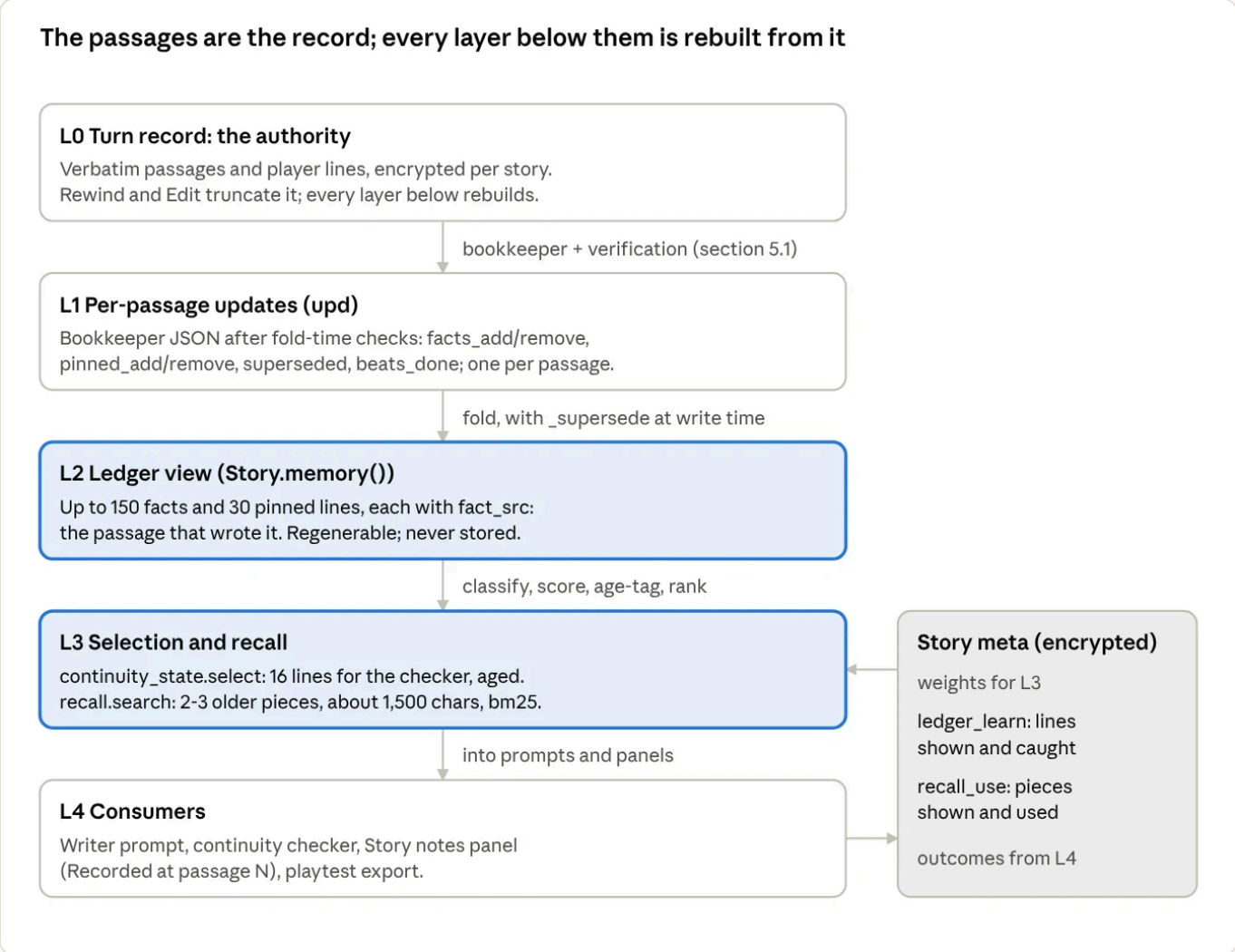
No derived state is stored on its own. The bookkeeper's output, after verification, is saved as `upd` on the last turn it covers. Every view (live cards, the facts ledger, beat progress, scene presence, the clock) is recomputed by folding the `upd` records of turns `0..mem_upto` in order (`Story.folded_updates()`, `Story.memory()`, `E.replay()`, `Story.beat_progress()`). This one design choice carries most of the weight in what follows:

- **Rewind and Edit are exact.** Truncating the turn list removes the updates, so every fact, beat, supersession and source pointer they introduced disappears with them. There is no compensating delete to get wrong.

- **Verification is cheap to bolt on.** A check that edits `upd` before it is saved changes every later view consistently.
- **Nothing needs a migration.** A new derived field is either present in newer `upd` records or recomputed from older ones on the next fold.

4.3 Data layers: the record and its views

The continuity data sits in five layers. Only L0 is stored as such; L1 rides on the turns; L2 to L4 are recomputed.



Continuity layers · story.py memory, _update_refs; continuity_state.py; recall.py

The only state outside the turns is the per-story learning record in encrypted `meta`. It holds counts, never story facts, so a lost record costs some ranking quality and nothing else. Learning restarts from neutral weights.

5. Technical features

Seventeen features carry the architecture. Nine verify plot progress (§5.1) and eight maintain and use the ledger (§5.2).

#	Feature	What it does	§
1	One beat at a time	Writer sees the beats done, the next beat, and only a count of the rest	5.1.2
2	Director	Orders the next beat as a required passage once the hero and the beat's people have been in place for a streak	5.1.2
3	Beat pull	Plants a hook toward a beat's place elsewhere every 4 passages, at most 4 times	5.1.2
4	Stall pass	Passes a beat after 20 or 30 quiet passages, with no event XP	5.1.2
5	Evidence judge	Re-judges each claimed beat against the summary, facts and recent passages, shown only the unrecorded beats	5.1.3
6	Gate chain	Fight, verbatim, name, hindsight and relevance gates on every quote	5.1.4
7	beat_trace	Reason, quote and raw judge reply per claim, in playtest exports	5.1.4
8	Single-point closure	Closes an event once, last, never in a passage that rejected one of its beats	5.1.5
9	Sweep, catch-up, side-event check	Records played beats nobody claimed, so a scene is never staged twice	5.1.6
10	Ledger fold with caps	150 facts and 30 pinned lines; facts the engine already owns are dropped	5.2.1
11	Source pointers and age tags	Each line knows its passage; the checker sees how old it is	5.2.2
12	Memory-wipe tags	Lines older than a memory-wiping death are marked as forgotten by the hero	5.2.2
13	Supersession	Retires a contradicted fact at write time; recorded and rewindable	5.2.3
14	Kind-scored selection	Picks the 16 most relevant lines for the checker, with a fallback to numbered lines	5.2.4
15	Slip-learned weights	Kind multipliers and per-line bonuses from slips the checker caught	5.2.4
16	Continuity rewrite loop	A slip forces up to 2 rewrites with the slip named	4.1, 5.2.4
17	Learned recall	bm25 recall reweighted by whether the writer used each piece	5.2.5

5.1 Beat bookkeeping and the quote-check verification loop

A planned story beat is recorded only when the story itself can be quoted showing it. The bookkeeper's claim starts the check but is never enough on its own. Plot progress drives XP, arc transitions and what the writer is told to stage next, so a false positive costs more than a false negative. A missed beat is caught later by the sweep or passed by the stall timer. A wrongly credited beat skips a scene the player never sees.

5.1.1 Events and beats

Worlds carry **event cards**: an id, an arc, an order, an optional place, an optional story-clock time (`at`), an optional level requirement, and an ordered list of **beats**. A beat is one sentence describing a scene, optionally followed by a `(Colour if it fits …)` aside that never blocks it. `open_events()` exposes, per arc, the next main event and any side events whose preceding main event is done. Arcs open only when the previous arc has ended, never by level alone.

Progress is the key `"<event id>#<n>"`. `Story.beat_progress()` folds every `beats_done` list in turn order, keeping the maximum `n` per event and resetting on `events_reopen`. Because it is a fold, a rewind un-credits beats automatically.

5.1.2 The writer sees one beat at a time

`Story.beats_text()` gives the writer an event as *ALREADY HAPPENED: … / NEXT BEAT (k of N): … / M more beat(s) held back*. The writer can't jump to an event's ending, and can't leak what later beats reveal, because it is never shown them. Three pacing devices push the next beat onto the page, and each defers to the player's own choice of action:

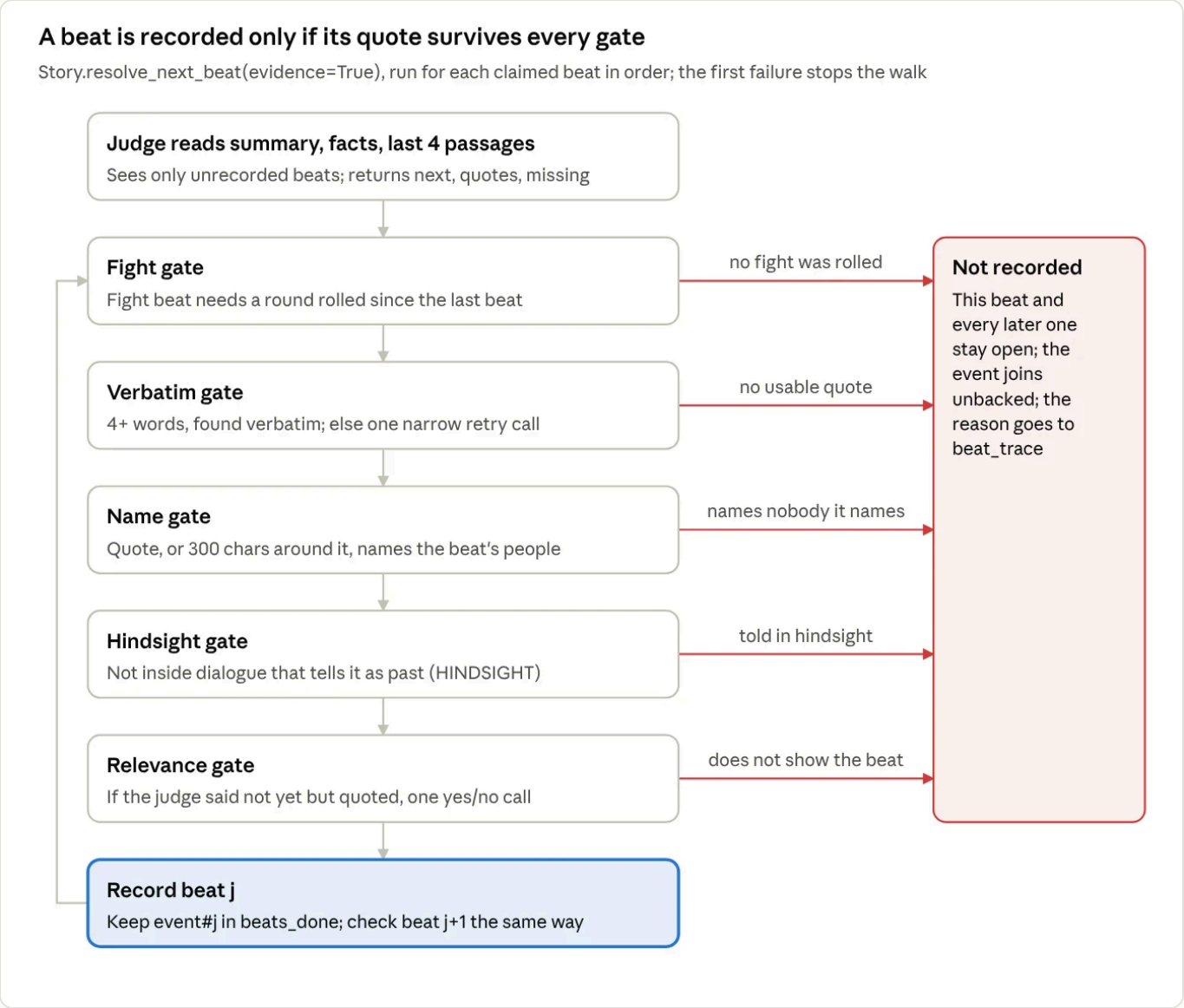
- **Director** (`director_due`): once the hero has been at the beat's place with its people for a focus-dependent streak of passages, the beat is ordered as a required passage. A beat that doesn't land is retried at most `DIRECTOR_TRIES = 2` times, `DIRECTOR_BACKOFF = 3` passages apart.
- **Beat pull** (`beat_pull`): when the beat's place is elsewhere, the writer is told to plant a hook toward it every `BEAT_PULL_EVERY = 4` passages, at most `BEAT_PULLS = 4` times. It never moves the hero.
- **Stall** (`STALL_PASSAGES`: 20 passages on Tight focus, 30 on Balanced, never on Sandbox): a beat that hasn't happened in that long is *passed*. It is recorded in `beats_passed`, and an event closed this way gets no event XP (`events_stalled`).

5.1.3 Claim, then judge

The bookkeeper prompt (`engine.py`) asks for `beats_done` only for beats that "actually played out on the page". It excludes beats that were mentioned, planned or told as backstory, and fights that started but weren't decided. It accepts a beat the player settled another way when the stakes were settled for good. Prompts are not enforcement, so every claimed beat is re-judged at fold time by `Story.resolve_next_beat(e, summary, evidence=True)`:

- The judge (temperature 0, JSON mode, 420 tokens) gets the summary just written, the last 40 facts, the current places, the hero's name, and the last `CHECK_PASSAGES = 4` passages, **including the one being folded** (`recent_passages(pending=True)`).
- It sees **only the beats not yet recorded**, numbered as they really are, with an explicit note that beats 1..r are done. Earlier, showing it the recorded beats led it to answer "beat 1 hasn't happened" once their scenes had scrolled out of the window. That pushed its answer below the recorded count and wrongly dropped real claims (story 42).
- `EVIDENCE_RULE` requires, for every beat counted as happened, an exact sentence of at least six words naming at least one person the beat names. It also tells the judge that narration of "you" or "I" *is* the hero.
- It returns `{"next": k, "evidence": {"<beat>": "<quote>"}, "missing": "…"}`. When `missing` is empty and the judge supplied quotes past `next`, the walk extends over those beats too. Every one of them still has to pass the gates.

5.1.4 The gate chain



Beat evidence gate chain · story.py resolve_next_beat, told_not_shown, _beat_quote, _quote_shows

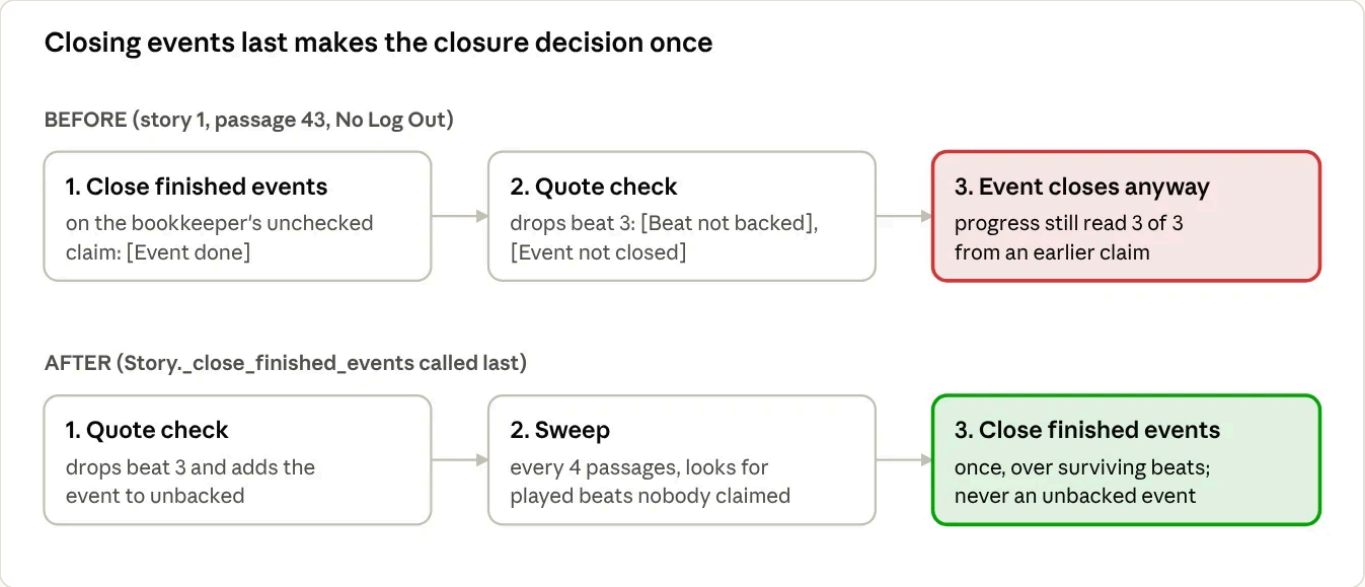
Each gate is a specific, field-observed failure turned into a check. The deterministic gates cost nothing. The two extra model calls are narrow and fire only on the paths that need them.

Gate	Check (code)	Failure it closes	Seen in
Fight	`beat_needs_fight()`: fight words in the beat's core, in a world that rolls combat; `fought_since_beat()`: a round rolled on a player turn since the last passage that recorded a beat	Fight beat credited when the hero merely arrived at the monsters' field; 200 XP and a level-up paid out	Story 44
Verbatim	Quote normalised (`_ev_norm`), at least 4 words, a substring of summary + facts + recent passages. Failing that, `_beat_quote()` asks once for one sentence copied exactly	Judge confirmed beats but gave no usable quote, so five real beats were thrown away	Story 43
Names	The beat's people (cards already in play) appear in the quote or within 300 characters of it	A word that only sounds like the beat ("the vanguard drums" for "the royal class deployed as the vanguard") with none of its people there	Docstring example
Hindsight	`told_not_shown()`: quote lies inside quoted speech that matches `HINDSIGHT` ("ago", "since the night", "used to", ...)	Beat credited from a stranger's remark ("since the night the log-out button vanished") hours before the scene could happen	Story 29
Relevance	Only when the judge said beat *j* has not happened yet but quoted it anyway: `_quote_shows()` asks a 40-token yes/no question	"finds Glasscannon in the slime fields" accepted as "the log-out button is gone"	Story 44

The walk stops at the first failure (`n = j; break`): a later beat can't be credited past an unproven earlier one. Every decision is appended to `beat_trace` with the gate's reason, the quote and the judge's raw reply (500 characters), and lands in the playtest `history.json`. That trace is how the last four rows of the table were diagnosed.

5.1.5 Ordering: closure is decided once, last

Inside `Story._fold_apply()` the beat logic runs in a fixed order. Claims are parsed, keeping only beats of events that were open. Director keys are confirmed. Stalled beats are passed. The quote check drops unbacked beats and records their events in `unbacked`. The sweep looks for unclaimed beats. Last, `_close_finished_events()` closes every open event whose recorded progress reaches its beat count, unless the event is in `unbacked`.



Fold-time ordering, before and after the 2026-10-28 fix · story.py _fold_apply

The ordering matters because three passes once appended to and removed from one shared list. In story 1, passage 43, the turn log carried `[Event done]`, `[Beat not backed]` and `[Event not closed]` for the same event, and the event closed anyway. An earlier passage had recorded the same beat number on its own unverified claim, so `beat_progress()` still read 3 of 3. The `unbacked` set closes that hole: an event whose beat was rejected in *this* passage is never closed in it, however far its recorded progress has got. A guard test in `test_beat_closure.py` checks the call order so it can't regress.

5.1.6 Safety nets and failure behaviour

- **Sweep** (`_sweep_beats` , every `SWEEP_EVERY = 4` passages): reads the recent passages for the first open main event and records beats nobody claimed, through the same evidence path. It catches scenes played in the opening or spread over several passages.
- **Catch-up before re-staging** (`catch_up_beats`): before the director orders a beat again, one cheap check confirms it hasn't already happened. The same scene is never staged twice.
- **Side event on leaving** (`_side_on_leave`): side events never stall, so when the hero leaves a side event's place, the evidence judge is asked once how many of its beats really happened.
- **A failed judge fails closed:** if `resolve_next_beat` returns `None` , no new beat is recorded in that passage and the event is not closed; the claim stays open for the sweep or a later passage. A failed relevance call counts as "does not show". An outage delays bookkeeping and never credits it, and it never costs the player a turn.

5.1.7 Cost

On the normal path, one passage with a beat claim costs one judge call. `_beat_quote` (160 tokens) is added only when the judge confirmed a beat without a usable quote. `_quote_shows` (40 tokens) is added only on the contradictory path. The sweep adds at most one judge call per four passages, for one event. Passages that claim no beat cost nothing. Every call is logged under the `beat check` cost type.

5.2 The layered ledger: supersession and slip-learned selection

The facts ledger is a **view**, not a store. Each line carries a pointer to the passage that wrote it. A new line that flatly contradicts an old one retires it at write time. The lines the continuity checker sees are chosen by a scorer that learns, per story, which kinds of line actually catch the writer out.

5.2.1 The ledger as a fold

The bookkeeper writes `facts_add` (at most 4 lines of up to 200 characters a passage), `facts_remove`, `pinned_add` (lasting exact specifics such as room numbers and addresses; at most 4) and `pinned_remove`. `clean_memory()` clamps them and drops facts the engine already owns (`engine_owned_fact`: the sheet, the cards and the purse say it). `Story.memory()` folds them in turn order into at most `MAX_FACTS = 150` facts and `MAX_PINNED = 30` pinned lines. Facts are never summarised away. Past 80 facts the bookkeeper is asked to retire 10–20 lines that no longer matter (`LEDGER CLEANUP DUE`). The player can delete a wrong fact from the Story notes panel, and that deletion is itself recorded on the latest passage, so Rewind brings the fact back.

5.2.2 Source pointers

`Story._update_refs()` returns one `{tid, passage}` per folded update, in the same order. The fold uses it to build `fact_src`, which maps each live line to the passage that first added it while the line lives. A line removed and re-added takes the new passage. `continuity_state.fold_sources()` reproduces this for offline replay. Three things use the pointers:

1. **Aged lines for the checker** (`Story.aged_facts`, `FABLYOS_FACT_AGES`): "Holt's arm is broken [12 passages ago]". The checker prompt says an old *state* fact may have changed without a note, while names, deaths, debts and fixed details stay true however old. This targets false positives from stale wound, door and mood lines. The facts budget in the prompt went from 1,200 to 1,500 characters to pay for the tags.

2. **Memory wipes** (`Story._respawn_rules` , `forgot_before`): in a world whose death rule erases the hero's memories, lines older than the wipe are tagged "[the hero no longer remembers this]". They stay true of the world, but the hero not knowing them is never a slip.
3. **Provenance for people**: the notes panel shows "Recorded at passage N" on each fact, and the playtest export carries `facts_from` .

5.2.3 Supersession: the missing write-time check

The fold had only add and remove. When the bookkeeper wrote "The cellar door stands open" and forgot to remove "The cellar door is locked", both lines were true for every later passage. `Story._supersede()` runs on each passage's update right after `clean_memory()` . Each new line is compared against the live ledger *before* this passage, using `continuity_state.conflicts()` . A contradicted fact is added to `facts_remove` as if the bookkeeper had written it there, and pinned lines are treated the same way. A new *fact* that contradicts a *pinned* line is recorded but does not remove it, because pinned lines are specifics the bookkeeper changes on purpose. Every replacement is kept in `upd["superseded"]` as `{old, new, pinned?, kept?}` (at most 12), so it shows in the playtest history and rolls back with Rewind.

`contradicts(new, old)` compares a signature of each line: the leading subject word, its content words, a polarity per state group, any free negation, its numbers, and whether it states a standing value. It is deliberately conservative, and fires only when the two lines have **the same leading subject** and the smaller line's content words lie at least 66% inside the larger's, and one of these holds:

- an **opposite state** in a shared group: lock (locked/open), life (alive/dead), hurt (broken/healed), know (knows/unaware), trust (trusts/hostile), have (has/lost). The *last* polar word in a line wins, so "Mara's arm, broken in the collapse, was healed" reads as healed;
- a **negation flip** of otherwise identical words ("is passable" / "is not passable");
- a **different number for a standing value**: both lines carry a state verb such as owes, costs, lives or remains.

The behaviour below was checked by running `continuity_state.contradicts` from the `dev` branch directly:

New line	Old line	Supersedes?	Why
The cellar door stands open	The cellar door is locked	Yes	lock group flips
Holt owes Mara 5 silver	Holt owes Mara 20 silver	Yes	standing value, new number
Mara's arm, broken in the collapse, was healed by the temple healer	Mara's left arm is broken	Yes	last hurt word is "healed"
The bridge is not passable	The bridge is passable	Yes	negation flip
Holt is dead	Holt is alive	Yes	life group flips
Bought 5 arrows at the stall	Bought 3 arrows at the stall	No	events, not a standing value
Mira trusts Holt	Holt trusts Mira	No	different leading subject

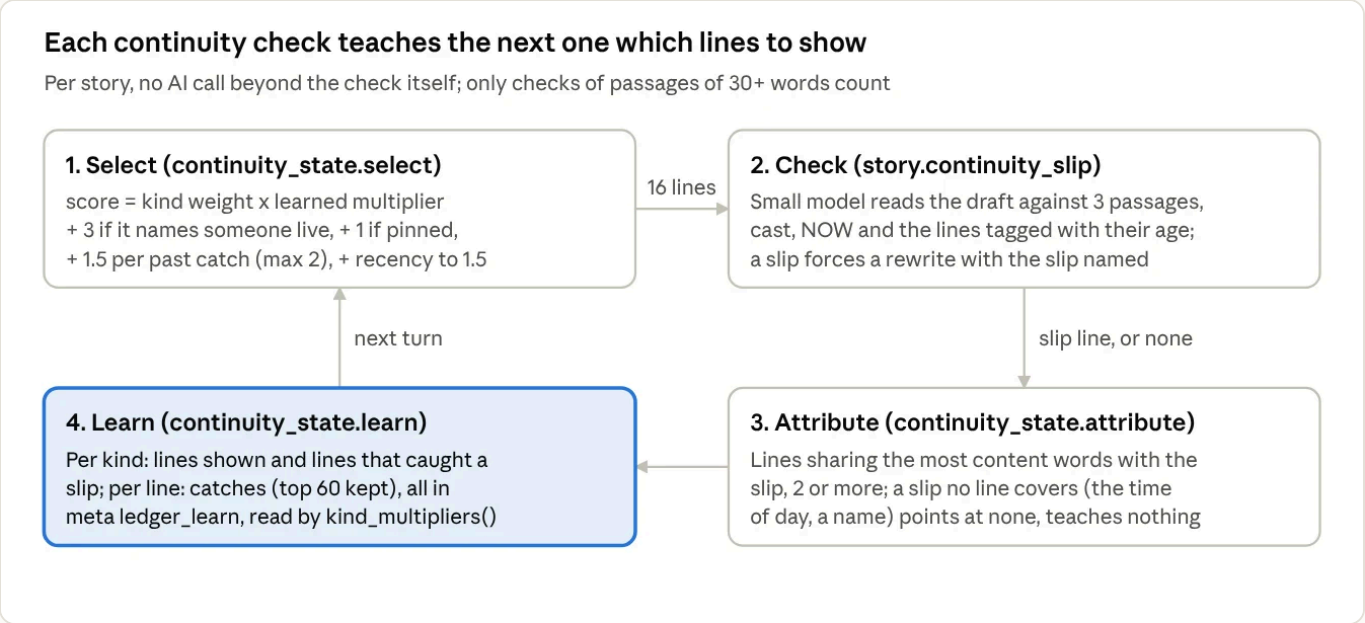
5.2.4 Slip-learned selection

The checker can be given only about 16 lines (`checker_facts(n=16)`), trimmed to 1,500 characters, so **which lines it sees decides which slips it can catch at all**. The earlier filter (`check_facts`) took only lines with a number or a duration in them, which suited one bug (a cooldown read as a cast time) and missed the slips players notice most.

`continuity_state.classify()` sorts each line into a kind by ordered regular expressions: **state** (wounds, locks, deaths, what someone carries), **knowledge** (knows, told, secret), **bond** (trusts, owes, promised), **place**, **timing** (numbers, durations, coins) or **other**. Each line then gets a score:

$$\text{score}(\ell) = w_k\, m_k + 3\,[\ell \text{ names someone live}] + 1\,[\ell \text{ pinned}] + 1.5\ \min\,(c_\ell, 2) + 1.5\,\frac{i_\ell}{n}$$

Here *w* is the base weight of the line's kind (state 3, knowledge 3, timing 2, bond 1.5, place 1.5, other 0.5). *m* is the learned multiplier. "Live" means in the scene, or named in the draft being checked. *c* is how many slips this exact line has caught before, and *i/n* is its position from oldest to newest. Numbered lines that the selection misses are appended after it, so the old filter's cases are kept. If anything raises, `checker_facts` falls back to `check_facts()` alone.



Slip-learned selection loop · story.py checker_facts, learn_check; continuity_state.py

After every real check, `Story.learn_check()` calls `continuity_state.learn()`. For each kind it records how many lines were shown and how many were caught (`[s_k, h_k]`). `attribute()` decides which line a slip was about: the lines sharing the most content words with the checker's slip line, at least two. A slip about the time of day points at no line and teaches nothing. The multiplier is each kind's catch rate against the story's own average, smoothed with 30 pseudo-showings and clamped:

$$\bar{o} = \frac{H+1}{S+20}, \qquad m_k = \mathrm{clip}_{[0.6,\ 1.8]}\left(\frac{(h_k + 30\,\bar{o})\ /\ (s_k + 30)}{\bar{o}}\right), \qquad m_k = 1 \text{ while } H = 0$$

In practice, a story whose writer keeps undoing injuries comes to show the checker more injury lines, and one whose writer keeps reviving the dead shows more of those. In a synthetic run of the dev-branch code, the same four lines were shown 40 times and "Holt's arm is broken" caught a slip every fifth check. The multipliers came out at state 1.73, bond 0.60, place 0.60, with knowledge and timing unchanged at 1.00. The arm line also earned a per-line catch count of 8, worth +3.0 at the cap.

5.2.5 Learned recall

Older passages are cut into 800-character pieces and indexed in SQLite FTS5 over hashed terms (`recall.py`), so nothing leaves the story's encryption. Each turn, up to `RECALL_MAX = 3` pieces within `RECALL_BUDGET = 1,500` characters go to the writer as "Earlier in the story", but only pieces that share at least two non-name words with the moment. `Story.learn_recall()` then checks whether the writer *used* each piece: at least two of the piece's own five-plus-letter words, other than the search words, must

reappear in the new passage. The use record reweights each piece's bm25 score, normalised to the best match:

$$f_p = \mathrm{clip}_{[0.5,\ 1.5]}\left(\frac{2\,(u_p + 2)}{s_p + 4}\right)$$

The prior is two uses in four showings, so a piece with no record has weight 1.0. The weight only reorders matches that already passed `MIN_TERMS` and `MIN_TOPIC`, and never adds one. Ranking lives in `recall.pick()` so it can be tested without a database.

---

6. Relation to Layered Continuity Architecture and prior work

The ledger work in §5.2 is a direct, deliberately narrow adoption of three ideas from MindStone's *Layered Continuity Architecture* preprint (Bodungen, v2.1, 2026-09-21). LCA's core rule is that the episodic record is authoritative and every other memory is a regenerable view of it. Fablyos already had that shape before the comparison was made. Hooking MindStone in as a service was ruled out for two reasons: it is built for coding-agent harnesses, and semantic recall would need plaintext story text outside the per-story encryption.

LCA concept	Fablyos counterpart	Status
Append-only history as the authority	Turn record; every view folded from per-passage `upd`	Present before LCA
Durable memory as a regenerable view	Facts and pinned ledger rebuilt by `Story.memory()`	Present before LCA
Bounded per-turn recall	`recall.py`: FTS5 bm25, at most 3 pieces and 1,500 characters	Present before LCA
Provenance on memory lines	`fact_src` source pointers; age tags for the checker	Adopted 2026-10-28
Ranking that learns from use and correction	`ledger_learn` kind multipliers; `recall_use` weights	Adopted 2026-10-28
Verification stage on memory writes	`_supersede()` contradiction check at fold time	Adopted 2026-10-28
Identity layer, consolidation, multi-agent authority	Not applicable: one story, one writer, no persona continuity	Not adopted

The beat verification loop in §5.1 was developed independently, from playtest failures, before the LCA comparison. It is closest in spirit to the *LLM-Modulo* pattern, where a model proposes and an external verifier decides, which LCA also cites. It is narrower than general LLM-Modulo: the verifier is mostly string-level (a quote must exist verbatim), and model calls are used only where a string check cannot decide.

Two departures from LCA are deliberate. First, **recall is lexical, not semantic**. Terms are hashed with a per-story key before indexing, so the index never holds plaintext and no embedding provider sees story text. The cost is the paraphrase gap that embeddings would close. Second, **learning is per story**: a story starts at neutral weights, so a replay can't peek at its own earlier slips. The cost is a cold start in every new story (§9).

The score in §5.2.4 follows the familiar recency + relevance + importance form of *Generative Agents* (Park et al., 2023). The difference is that "importance" is not rated by a model. It is learned from which lines have actually caught errors in this story.

7. Evaluation status and limitations

The beat loop in §5.1 is **field-observed**: every gate answers a failure seen in a named playtest story, and each fix was reproduced in unit tests against the judge outcomes that caused it. The ledger work in §5.2 is **implemented but not measured**. No before/after slip rate exists yet, and none should be quoted until one does.

7.1 What is in hand

Evidence	Covers	Size
`test_layered_continuity.py`	Supersession, source pointers, ages, learning, recall weights	29 tests
`test_continuity_state.py`	Line classification and selection	27 tests
`test_beat_pull.py`, `test_beat_closure.py`, `test_beat_scope.py`, `test_beat_quote.py`, `test_beat_told.py`, `test_beat_check.py`	Beat pacing, closure order, quote and hindsight gates	44 tests
`beat_trace` in playtest `history.json`	Per claim: gate outcome, reason, quote, the judge's raw reply	Every playtest run
Full suite at the last recorded run	Whole engine	1,020 tests passing (DEVLOG, 2026-10-28)

7.2 How §5.2 should be measured

`continuity_eval.py` replays stories already written through the continuity check. It folds the ledger to each passage, not to the end of the story, and reports **slips per 10,000 words**, a rate that stays comparable as stories get longer. Three switches isolate the mechanisms:

- `--facts numeric` against `--facts state` compares the old number-only filter with the selection in §5.2.4.
- `--no-ages --no-learn` turns off the source-pointer age tags and the learned weights. Learning starts from nothing in each replayed story, so the replay can't see the original run's slips.

- `--model` checks the same passages with another model before the checker is moved to it.

The harness writes nothing and generates no prose. It does spend one small-model call per passage, so it prints an estimate and asks before running. A flag is the checker's opinion, not proof. The only honest headline number is a slip rate *plus* a hand-labelled sample of the flagged lines, so that precision is known. For §5.1, the matching study is to count claims and rejections by gate reason from `beat_trace` across a playtest series, and hand-label a sample of each.

7.3 Limitations

- **English-only patterns.** `HINDSIGHT`, `classify()` and the supersession word groups are English. Checked directly: "A porta da adega está aberta" against "… está trancada" does not supersede. A Portuguese-language story (such as one in the Novela das 8 world) gets the beat gates that are model-based but none of the supersession, and only name matches in selection. Nothing breaks; the mechanisms simply never fire.
- **Supersession misses reworded subjects.** "The door to the cellar is open" does not supersede "The cellar door is locked", because the leading subject words differ. This is the price of zero false supersessions between different people. The bookkeeper's own `facts_remove` still covers these cases.
- **The learning signal is the checker's opinion.** A kind that draws false positives is upweighted, shown more, and can draw more of them. The 0.6–1.8 clamp and the 30-showing prior limit the drift, but nothing corrects it.
- **Attribution is lexical.** A slip is credited to the lines sharing the most content words with it. A slip described in different words teaches nothing, or teaches the wrong line.
- **Quotes need a window.** The judge sees the summary, 40 facts and the last 4 passages. A beat that played long ago and survives only in a loose summary may not be quotable; the stall timer then passes it with no event XP.
- **Outages slow progress.** Because a failed judge records nothing, a long model outage holds events open until the sweep can check them or the stall timer passes the beat, which pays no event XP. The trade is deliberate: an error should never be a way to get a beat credited.
- **One open trace.** In story 1, passage 43, `beats_done` kept the rejected beat even after the ordering fix explains the closure. The path that re-added it was not found in the code. The DEVLOG flags it for the next playtest.

8. Implementation reference

All paths are under `infiniteWorlds/` on `dev` at `67b159d`. No schema change was needed for any of this: new state lives in `upd` records or in the story's encrypted `meta`.

File	Role	Key symbols
`story.py`	Turn loop, fold, beat verification, ledger	`_stream_turn`, `_fold_job`, `_fold_apply`, `resolve_next_beat`, `_beat_quote`, `_quote_shows`, `told_not_shown`, `beat_needs_fight`, `fought_since_beat`, `_sweep_beats`, `_close_finished_events`, `catch_up_beats`, `_side_on_leave`, `beats_text`, `memory`, `_update_refs`, `_supersede`, `checker_facts`, `aged_facts`, `learn_check`, `learn_recall`, `continuity_slip`, `EVIDENCE_RULE`, `CONTINUITY_PROMPT`
`continuity_state.py`	Line kinds, selection, sources, learning, supersession	`classify`, `select`, `fold_facts`, `fold_sources`, `aged`, `attribute`, `learn`, `kind_multipliers`, `contradicts`, `conflicts`
`recall.py`	Encrypted FTS5 recall with learned use weights	`search`, `pick`, `was_used`, `note_use`, `weights`
`engine.py`	Bookkeeper prompt and schema (`beats_done`, `facts_add`, `pinned_add`, …)	bookkeeping rules, `replay`
`continuity_eval.py`	Offline replay: slips per 10,000 words	`--facts`, `--no-ages`, `--no-learn`, `--model`
`playtest.py`	Bot playtests; exports `beat_trace`, `superseded`, `ledger_learn`, `recall_use`	`history.json`

Constant	Value	Meaning
`CONT_REWRITES`	2	Rewrites allowed after a continuity slip
`CHECK_PASSAGES`, `CHECK_CHARS`	4, 1,800	Passages, and characters per passage, the beat judge reads
`Story.SWEEP_EVERY`	4	Passages between sweeps for unclaimed beats
`DIRECTOR_TRIES`, `DIRECTOR_BACKOFF`	2, 3	Director retries per beat and the passage gap between them
`BEAT_PULL_EVERY`, `BEAT_PULLS`	4, 4	Hook cadence and limit per beat
`STALL_PASSAGES`	20 Tight, 30 Balanced	Quiet passages before a beat is passed; never on Sandbox
`MAX_FACTS`, `MAX_PINNED`	150, 30	Ledger caps
`NAME_BONUS`, `PINNED_BONUS`, `RECENCY`, `LINE_HIT_BONUS`	3.0, 1.0, 1.5, 1.5	Selection score terms
`LEARN_PRIOR`, `LEARN_MIN`, `LEARN_MAX`, `LEARN_LINES`	30, 0.6, 1.8, 60	Kind-multiplier smoothing, clamp and line memory
`RECALL_MAX`, `RECALL_BUDGET`, `CHUNK_CHARS`	3, 1,500, 800	Recall pieces, characters per turn, piece size
`USE_MIN`, `USE_MAX`, `USED_WORDS`	0.5, 1.5, 2	Recall weight clamp and the "used" threshold

Switches: `FABLYOS_CONTINUITY=off` disables the checker. `FABLYOS_FACT_AGES=0` removes the age tags. `FABLYOS_CONTINUITY_MODEL` moves the checker to another model.

---

9. Future work

1. **Run the measurement in §7.2** on five or more real stories, with and without `--no-ages --no-learn`, and hand-label the flags. This should come before anything else in this list.
2. **Feed player corrections into learning.** A player deleting a fact (`remove_fact`) is a stronger correction signal than the checker's opinion, but it isn't yet recorded in `ledger_learn`.
3. **A cross-story prior.** Learning starts from 1.0 in every story. A per-world or global prior per kind would warm up new stories faster, at the cost of one cross-story table.

4. **Language packs for the patterns.** Per-language word groups for `HINDSIGHT`, `classify()` and supersession, starting with Portuguese.
5. **Beat-gate metrics as a playtest flag.** Report rejection counts by reason per run from `beat_trace`, so that a drift in judge behaviour shows up without reading traces by hand.

References

- Bodungen, C. *Layered Continuity Architecture: experiential memory, authoritative history, and resonance-weighted recall.* MindStone Agent research preprint, v2.1, 2026-09-21. mindstoneagent.ai
- Park, J. S., et al. *Generative Agents: Interactive Simulacra of Human Behavior.* UIST 2023.
- Sumers, T. R., et al. *Cognitive Architectures for Language Agents (CoALA).* TMLR 2024.
- Kambhampati, S., et al. *LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks.* ICML 2024.
- Fablyos source: `fablyos/fablyos` (private), branch `dev`, commit `67b159d`; design history in `infiniteWorlds/docs/DEVLOG.md` (entries of 2026-10-25 to 2026-11-03).