DEFENSIVE PUBLICATION — TECHNICAL DISCLOSURE (TDCOMMONS DEPOSIT COPY) · 2026-06-25

**Keywords:** defensive-publication, prior-art, ai-agents, multi-agent-orchestration, ci-cd, capability-budget, reserved-file-lock

# Capability-Scoped Task Envelopes with Reserved-File Token Broker for Concurrent AI Coding Agents

### A Technical Defensive Publication

**Subtitle:** A planner-emitted capability budget that is identical at dispatch and at merge, a non-blocking TTL token broker that serialises high-blast-radius file edits, and an automatic schema/code envelope split with a migrate-before-code dependency edge — for orchestrating heterogeneous AI code-generation agents writing concurrently to one shared repository.

---

| Metadata | Value |
|---|---|
| Title | Capability-Scoped Task Envelopes with Reserved-File Token Broker for Concurrent AI Coding Agents |
| Author | Gustavo Assuncao, PhD |
| Publisher / Copyright holder | Gus IT LLC (Florida, USA) |
| Publication date | 2026-06-25 |
| Version | 1.0 |
| Document type | Technical Defensive Publication (public prior art) |
| Classification | Public |
| License | AGPL-3.0-or-later (copyleft; commercial license available) |
| Deposit channel | To be assigned (IP.com / Zenodo / arXiv) — establishes a public, dated, citable prior-art record |
| Field | Orchestration of multiple heterogeneous AI code-generation agents writing concurrently to a shared version-controlled repository |

---

## Abstract

When several heterogeneous AI code-generation agents write to one shared repository concurrently, branch isolation plus post-hoc merge does not bound *per-agent scope* and does not prevent *reserved-file races* on a small set of high-blast-radius files. This publication describes a method that addresses both failures with one coherent mechanism.

A **planner** classifies each natural-language change request into a *kind* (create, modify, debug, migrate, deploy, read-only) and a *risk* label, then emits one or more **task envelopes**. Each envelope carries a **declarative capability budget** — `allow_paths`, `deny_paths`, `max_files_changed`, `max_lines_changed`, and `may_add_dependencies` — plus an agent-role assignment, required reserved-file tokens, required CI checks, an optional feature flag, and dependency edges. The decisive property is **dispatch≡merge identity**: the same envelope object that scopes the agent at runtime is the *exact contract* re-verified against the pull-request diff at merge time by a CI **envelope-gate**.

A **non-blocking reserved-file token broker** serialises edits to enumerated high-blast-radius paths (dependency manifest/lockfile, migrations, container build file, deployment manifests, server entrypoint). A token is a TTL-bounded lease acquired by an **atomic conditional insert** (`INSERT … ON CONFLICT DO NOTHING` — success = own it, conflict = requeue) and auto-released by a **reaper** on expiry. The broker is deliberately *advisory*: it shapes dispatch ordering so collisions are rare, while the serialised merge queue remains the authoritative conflict catcher.

Finally, a request touching **both** migrations **and** code is **automatically split** into a token-gated *schema envelope* and a *code envelope* carrying a dependency edge that blocks dispatch until the schema envelope merges — encoding expand/contract migrate-before-code as a first-class scheduling constraint. The combination yields heterogeneous agents working concurrently with bounded, enforceable scope and without reserved-file races, where the dispatch-time scope equals the merge-time enforced contract. This document is published as prior art.

---

## 1. Executive Summary

### 1.1 Thesis

A unit of autonomous coding work should be defined by a **machine-checkable capability budget that is identical at the moment work is dispatched and at the moment work is merged**. If the two differ, scope is unenforceable: the agent is told one thing and the merge gate checks another. This publication closes that gap by making one **envelope** object the single source of truth, threading it through dispatch and CI, and surrounding it with a lightweight reserved-file broker and an automatic schema/code split.

### 1.2 Contributions

| # | Contribution | Where |
|---|---|---|
| C1 | **Dispatch≡merge identity.** A single declarative capability budget per task envelope that is the runtime scope at dispatch *and* the re-verified contract in a CI gate at PR time. | §6, §7.2, §14 (Claim 1) |
| C2 | **Non-blocking reserved-file token broker.** TTL-bounded leases on enumerated reserved paths, acquired by atomic conditional insert, reaped on expiry, deliberately advisory relative to a serialised merge queue. | §7.3, §8, §11 |
| C3 | **Automatic schema/code envelope split.** A prompt touching both migrations and code is split into a token-gated schema envelope plus a code envelope with a migrate-before-code dependency edge. | §7.4, §10 |
| C4 | **Dependency- and token-gated pull dispatch.** A worker claims the next eligible envelope only when its dependency edges are all merged and none of its required tokens are held, using a skip-locked claim. | §7.5, §8 |
| C5 | **Envelope-sha provenance.** A content hash of the frozen envelope is stamped on the task and surfaced on the PR so the CI gate verifies the diff against the *exact* envelope that scoped the work. | §7.2, §9 |
| C6 | **Risk-gated approval and feature-flag derivation** folded into the same envelope, so a HIGH-risk or user-facing surface is governed by the same object the gate enforces. | §7.6 |

### 1.3 Headline claim

> *The dispatch-time capability scope handed to an AI coding agent is byte-identical to the merge-time contract enforced by continuous integration — and high-blast-radius files are serialised by a non-blocking, TTL-bounded, atomically-acquired reserved-file token broker, with database-migration work automatically separated from and ordered before the application code that depends on it.*

### 1.4 Scope — what this is and is not

**It is:** an orchestration method and data model for bounding the scope of, and de-conflicting, concurrent writes by heterogeneous AI coding agents to one repository; a closed-loop contract between a planner/dispatcher and a CI gate; and an advisory reserved-file mutex with automatic schema/code ordering.

**It is not:** a merge-conflict *resolution* algorithm (the serialised merge queue remains authoritative); a model for *training* coding agents; a distributed-locking primitive claimed in the abstract (we use, and credit, well-known conditional-insert and TTL-lease techniques); a replacement for code review or tests; or a claim that any single sub-mechanism is itself novel. The novelty asserted is the **specific combination**, and in particular the **dispatch≡merge identity** of the capability budget.

---

## 2. Introduction & Motivation

### 2.1 The concrete problem

Modern teams increasingly run **more than one** AI coding agent at the same time against **one** repository — for example a hosted agent, a second vendor's agent, an IDE-embedded agent, and an internally built agent. Each is handed a natural-language prompt, opens a branch, and produces a pull request. Branch isolation gives each agent a private workspace; a merge queue serialises the integration. This is the state of the art, and it leaves two gaps.

**Gap A — unbounded scope.** A prompt like *"add pagination to the leads list"* does not, by itself, constrain which files an agent may touch. A capable agent may "improve" adjacent code, bump a dependency, or reorganise a module — changes the requester never asked for and a reviewer must now untangle. There is no machine-checkable contract that says *"this task may touch only these paths, at most this many files and lines, and may not add a dependency."*

**Gap B — reserved-file races.** A handful of files are *high-blast-radius*: the dependency manifest and its lockfile, database migrations, the container build file, the deployment manifests, and the server entrypoint. When two concurrent agents both edit the lockfile, or both add a migration with the same ordinal, the result is a guaranteed merge conflict or a non-deterministically broken build. Branch isolation does **not** prevent this; it merely defers the collision to merge time, where it is most expensive to diagnose.

### 2.2 The "tax"

Both gaps levy a recurring tax measured in human reviewer-minutes and pipeline restarts:

- **Scope tax.** A reviewer must read the *whole* diff to discover the agent went out of scope, then request changes, then re-review. Out-of-scope edits also inflate blast radius, raising the probability that an unrelated test breaks.
- **Collision tax.** A reserved-file race surfaces as a red merge-queue run. The losing PR must rebase, re-run the full pipeline, and possibly re-resolve a lockfile by hand. With $N$ agents and a shared hot file, the expected number of collisions grows super-linearly in throughput.

Neither tax is paid once; both scale with the number of agents and the rate of dispatch. The motivation for this work is to convert both into **structural** constraints checked by machines before a human ever looks.

### 2.3 Why existing approaches fall short

- **Branch isolation + merge queue** (the baseline) bounds *integration order*, not *per-task scope*, and detects reserved-file collisions only after both PRs exist.
- **Per-agent allow/deny path budgets** (configured on a single agent) constrain that agent at runtime but are *not* re-checked at merge time and are *not* coordinated across heterogeneous agents.
- **PR-time path/size gates** (lint-style danger rules) check a diff at merge but have *no* connection to what the agent was *told* it could do — the gate and the dispatch contract are written independently and drift.
- **DB job-queue locks** serialise *work items*, not *enumerated repository files*, and are not surfaced to a CI gate.
- **Expand/contract migrations** are a *human discipline*, not an automatic, scheduler-enforced split of one prompt into two ordered envelopes.

Each piece exists; none, alone, gives a **single capability contract that is identical at dispatch and at merge**, coordinated across agents, with reserved-file serialisation and automatic migrate-before-code ordering.

### 2.4 Why now

The 2024–2026 wave of multi-agent coding orchestration (heterogeneous agent dispatch, agent "control planes," and merge queues) makes concurrent multi-agent writing a default rather than an exotic configuration. As the agent count rises, the two taxes above move from nuisance to dominant cost. The method here is the missing coordination layer; publishing it defensively keeps it free to adopt.

---

## 3. Problem Statement

### 3.1 Formal framing

Let `A = {a_1, …, a_n}` be a set of heterogeneous coding agents and `R` a single version-controlled repository. A stream of natural-language requests `q_1, q_2, …` arrives. For each request `q`, the system must produce a set of **envelopes** `E(q) = {e_1, …, e_k}` and dispatch each `e_i` to some agent `a_j` such that:

1. **Scope soundness.** The diff `Δ(e_i)` that agent `a_j` ultimately proposes satisfies the budget of `e_i`: every changed path matches `allow_paths(e_i)` and matches no `deny_paths(e_i)`; `|files(Δ)| ≤ max_files(e_i)`; `|lines(Δ)| ≤ max_lines(e_i)`; and if `¬may_add_dependencies(e_i)` then `Δ` does not modify the dependency manifest/lockfile.

2. **Dispatch≡merge identity.** The predicate checked at merge time is computed from the *same* envelope `e_i` (identified by a content hash) that scoped `a_j` at dispatch time.
3. **Reserved-file mutual exclusion (advisory).** For each enumerated reserved path `p`, at most one in-flight envelope holds the token for `p` at a time, modulo TTL expiry; the serialised merge queue is the authoritative backstop.
4. **Migrate-before-code ordering.** If `q` touches both migrations and code, then `E(q)` contains a schema envelope `e_s` and a code envelope `e_c` with `e_c` blocked from dispatch until `e_s` is merged.
5. **Liveness.** No envelope is starved: stuck assignments are reclaimed, expired tokens are reaped, and eligible envelopes are eventually claimed.

### 3.2 Derived requirements

| ID | Requirement | Satisfied in |
|---|---|---|
| R1 | A planner classifies each request into a *kind* and *risk* label before scoping. | §7.1 |
| R2 | Each task is represented by an **envelope** carrying allow/deny paths, max files, max lines, may-add-dependencies, role, tokens, checks, flag, dependencies. | §6, §7.2, §8 |
| R3 | The **same** envelope budget is enforced at dispatch and re-verified against the PR diff at merge time (dispatch≡merge identity), bound by an envelope content hash. | §7.2, §9, §14 |
| R4 | A reserved-file token broker provides TTL-bounded, atomically-acquired, advisory leases on enumerated reserved paths, with a reaper. | §7.3, §8 |
| R5 | A request touching both migrations and code is automatically split into a token-gated schema envelope and a dependency-blocked code envelope. | §7.4 |
| R6 | Dispatch is pull-based and gated: a worker claims an envelope only when all dependency edges are merged and no required token is held. | §7.5 |
| R7 | HIGH-risk or schema/auth changes route to an approval state; user-facing surfaces derive a feature flag — both inside the envelope. | §7.6 |
| R8 | Liveness hygiene: reclaim stuck assignments by heartbeat timeout; reap expired tokens. | §7.7 |
| R9 | All state is in a relational store with parameterised SQL; status values are validated at write time. | §8 |
| R10 | The system fails *safely* under partial failure (advisory broker fails open; budget gate fails closed). | §11 |
| R11 | The mechanism is auditable: classification rationale, envelope-sha, token holders, and dependency edges are all persisted. | §8, §11 |

---

## 4. Related Work & Prior Art

This work is built deliberately on top of well-understood components; it does not arise *ex nihilo*. We name them and state what we adopt.

- **LLM-based code-migration decomposition (Rosie-style; Ziftci et al., "Migrating Code At Scale With LLMs At Google," FSE 2025).** Establishes that an LLM can decompose a large change into many small, independently shippable units. **Adopted:** the decomposition concept. **Not present there:** a per-unit capability budget that is re-verified at merge, or a reserved-file broker.
- **Heterogeneous agent dispatch / agent control planes (e.g., "Agent HQ"-style 2026 orchestration) + Git merge queues.** Establishes dispatching different agents to tasks and

serialising integration. **Adopted:** heterogeneous dispatch and a serialised merge queue as the authoritative conflict catcher. **Not present there:** dispatch≡merge identity of a capability budget, nor an enumerated reserved-file mutex.

- **Postgres** `SELECT … FOR UPDATE SKIP LOCKED` **job queues.** The canonical pattern for concurrency-safe work pulling. **Adopted:** the skip-locked claim for picking the next eligible envelope.
- **Conditional-insert mutex / TTL-lease + fencing-token distributed locks.** `INSERT … ON CONFLICT DO NOTHING` as a mutex and time-bounded leases with reapers are textbook. **Adopted:** exactly this for the token broker. We assert no novelty over the locking primitive itself.
- **Agent allow/deny-path capability budgets.** Single-agent runtime path budgets are a known configuration surface. **Adopted:** the allow/deny vocabulary. **Not present there:** the *same* budget re-checked at merge across heterogeneous agents.
- **PR-time path/size gates (danger-style rules, static-analysis path filters, code-quality size checks).** Establishes machine-checked diff policy at merge. **Adopted:** the merge-time check. **Not present there:** that the check's policy is the *identical* object that scoped the agent at dispatch.
- **Expand/contract "migrate-before-code."** A standard release discipline for schema changes. **Adopted:** the ordering. **Not present there:** automatic split of one prompt into two scheduler-ordered envelopes with a token-gated schema leg.

The honest synthesis: every ingredient is known. The asserted contribution is their **specific combination**, and most pointedly the **dispatch≡merge identity** (C1) — making the runtime scope and the merge-time policy the *same* object, not two independently drifting artifacts.

---

## 5. Prior-Art Delta

> Columns: **Prior source | What it has | What it LACKS | What THIS adds**

| Prior source | What it has | What it LACKS | What THIS adds |
|---|---|---|---|
| Rosie-style LLM decomposition (Ziftci et al., FSE 2025) | Decomposes a big change into many shippable units | Per-unit capability budget; merge-time re-verification; reserved-file broker | An envelope per unit with a budget that is the merge-time contract |
| Heterogeneous agent dispatch + merge queue (2026 control planes) | Routes different agents to tasks; serialises integration | Dispatch≡merge scope identity; enumerated reserved-file mutex; auto schema/code split | A capability budget threaded from dispatch into the CI gate, plus a reserved-file token broker |
| Postgres `FOR UPDATE SKIP LOCKED` queues | Concurrency-safe pull of work items | Keys on *work items*, not *enumerated repository files*; no merge-time budget | Skip-locked claim **gated on** dependency-merge and reserved-file tokens |
| Conditional-insert mutex / TTL-lease + fencing | Atomic mutual exclusion; time-bounded leases; reaper | Tie to repository file paths; advisory posture relative to a merge queue; surfacing to CI | A reserved-file broker keyed on enumerated repo paths, advisory by design |
| Single-agent allow/deny path budgets | Runtime path restriction for one agent | Cross-agent coordination; merge-time re-check; file/line caps as a contract | The *same* allow/deny + file/line budget re-verified at PR time across heterogeneous agents |
| PR-time path/size gates (danger/static-analysis rules) | Machine-checked diff policy at merge | Any link to the dispatch-time agent scope; a content hash binding policy to task | An envelope-gate whose policy **is** the dispatch envelope, bound by an envelope-sha |
| Expand/contract migrate-before-code | Human discipline to ship schema first | Automatic split; token-gated schema leg; scheduler-enforced dependency edge | Automatic schema/code split with a migrate-before-code dependency edge and a db-migrations token |

The **net delta** is the closed loop: one declarative capability budget that is simultaneously (a) the agent's runtime scope, (b) the reserved-file coordination input, (c) the dependency-ordering input, and (d) the merge-time enforced contract — with the schema/code split as an automatic consequence of the same classification.

---

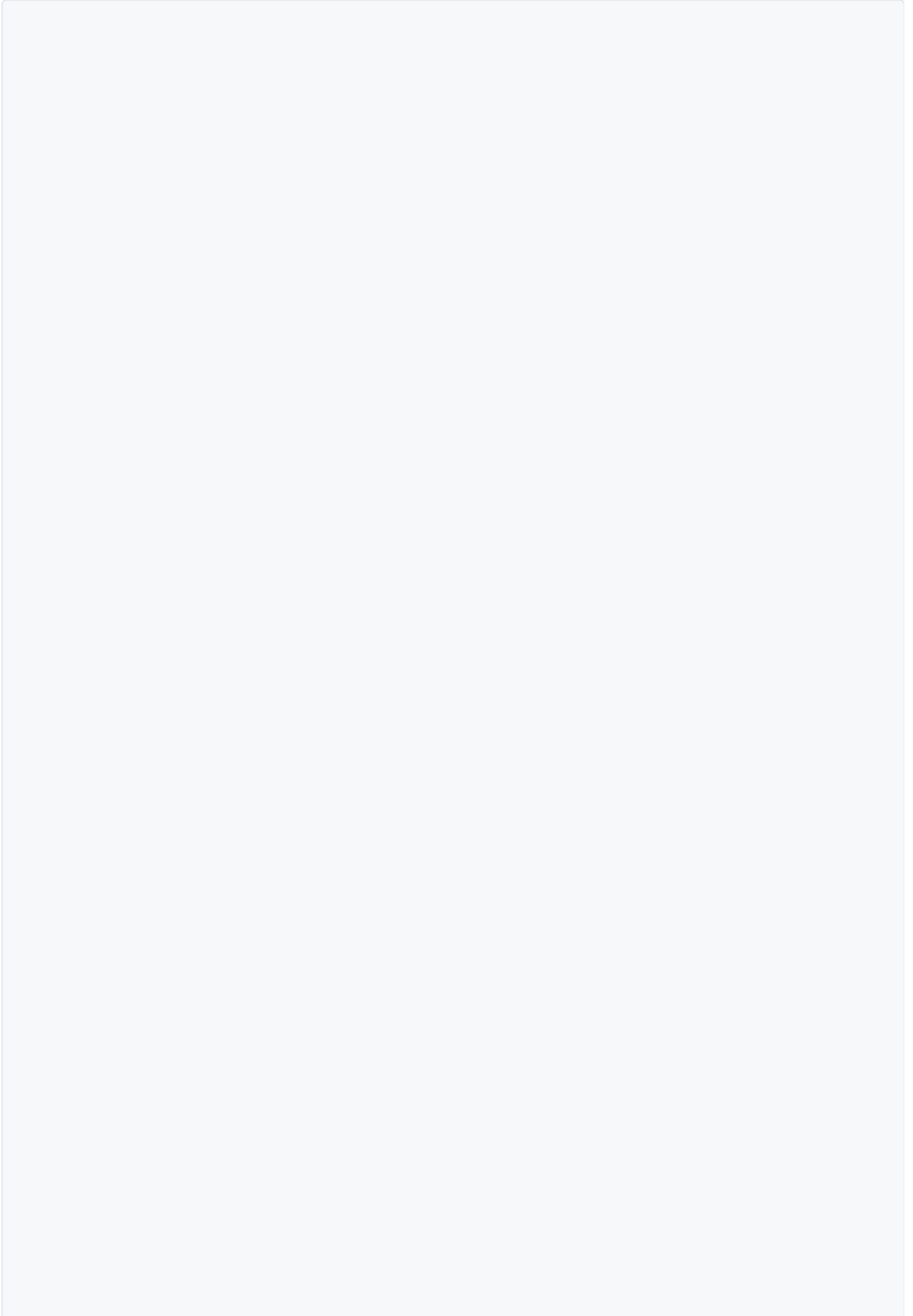
# 6. System Architecture

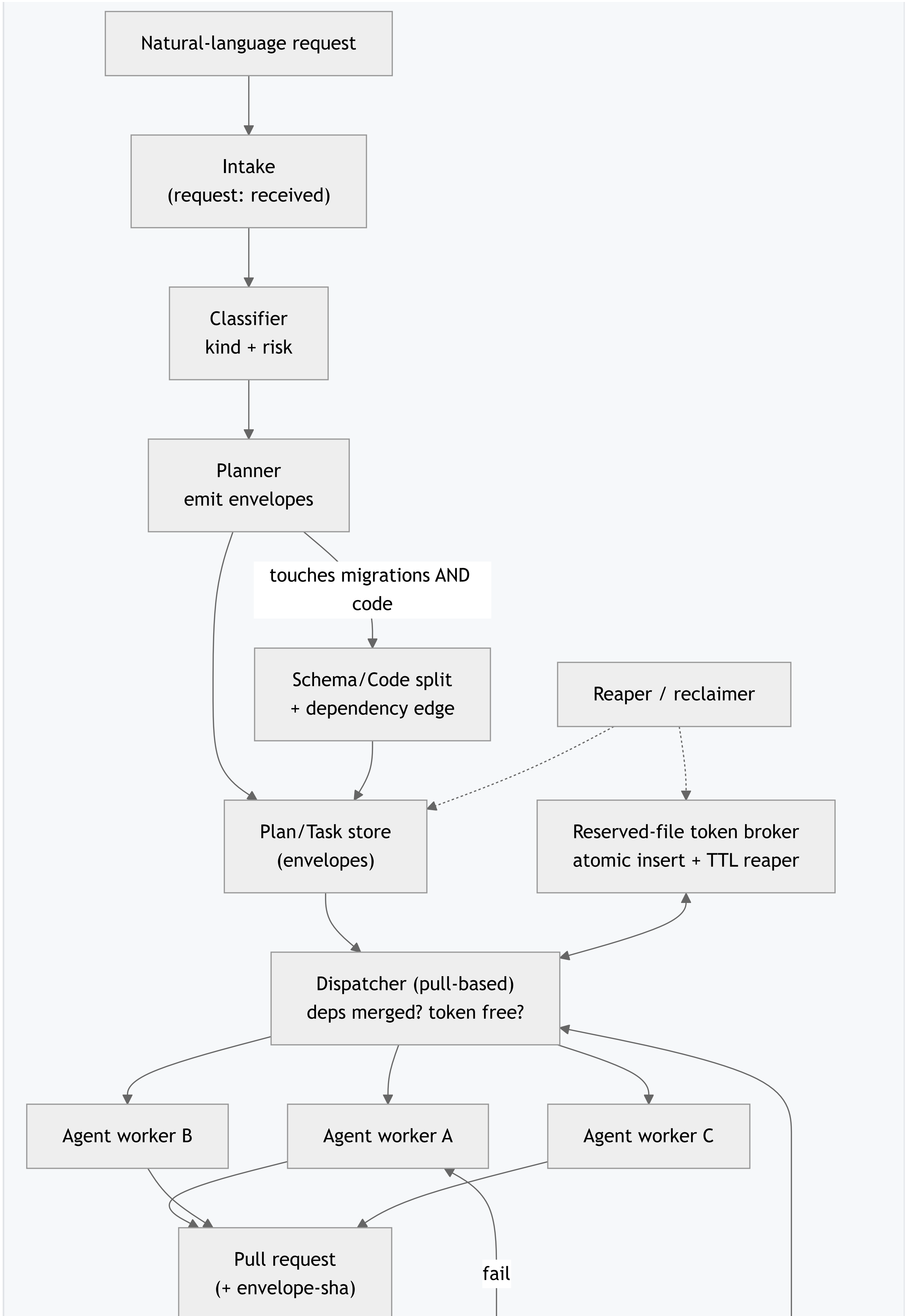
## 6.1 Components

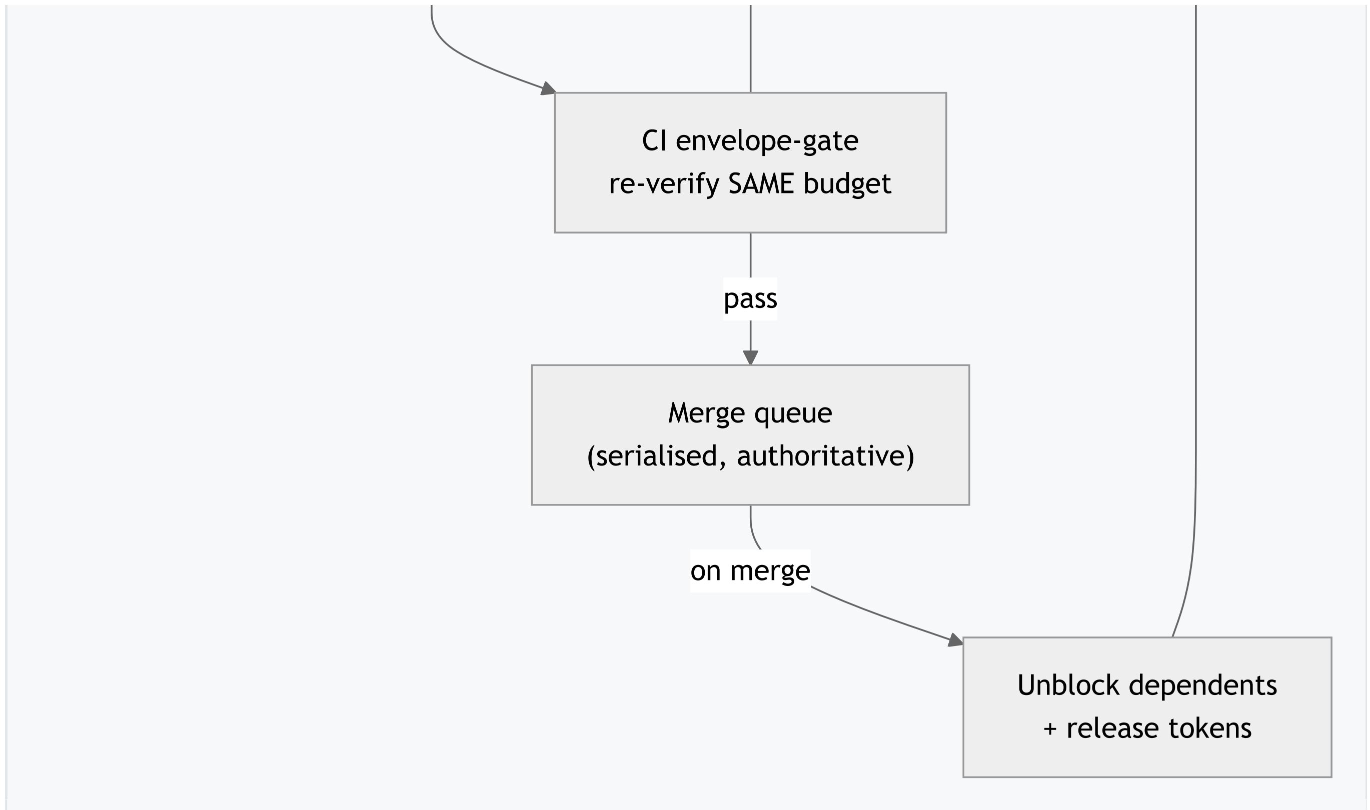
| Component | Responsibility |
|---|---|
| **Intake** | Accepts natural-language requests; persists each as a `request` row in status `received`. |
| **Classifier** | Derives `kind` and `risk` (fast regex path; LLM fallback). Writes a classification row with rationale. |
| **Planner** | Emits one or more **envelopes** from the classification: allow/deny paths, file/line budget, role, required tokens, checks, flag, dependency edges. Performs the schema/code split. |
| **Plan/Task store** | Relational store of plans, envelopes (tasks), reserved-file tokens, and worker registry. |
| **Token broker** | Atomic acquire (`INSERT … ON CONFLICT DO NOTHING`), release, and TTL reaper for reserved-file leases. |
| **Dispatcher (pull-based)** | Lets a worker of a given role claim the next eligible envelope (dependencies merged, no required token held) via a skip-locked claim; acquires the envelope's tokens. |
| **Agent workers** | Heterogeneous coding agents; each pulls an envelope, opens a branch, produces a PR, stamps the envelope-sha. |
| **CI envelope-gate** | At PR time, recomputes the budget predicate from the *same* envelope (matched by envelope-sha) against the PR diff; passes or fails the merge. |
| **Merge queue** | Serialises integration; the authoritative conflict catcher; on merge, dependent code envelopes unblock and tokens release. |
| **Reaper / reclaimer** | Reaps expired tokens; reclaims assignments whose heartbeat is stale. |

## 6.2 Architecture diagram

> **Figure 1 — System architecture.**




Natural-language request
Intake
(request: received)
Classifier
kind + risk
Planner
emit envelopes
touches migrations AND code
Schema/Code split
+ dependency edge
Reaper / reclaimer
Plan/Task store
(envelopes)
Reserved-file token broker
atomic insert + TTL reaper
Dispatcher (pull-based)
deps merged? token free?
Agent worker B
Agent worker A
Agent worker C
Pull request
(+ envelope-sha)
fail




### 6.3 Cross-cutting properties

- **Single source of truth.** The envelope is authored once by the planner and consumed by dispatch, the broker, the dependency scheduler, and the CI gate. There is no second, independently authored merge-time policy.
- **Advisory-but-helpful coordination.** The broker reduces collisions; it does not *prove* their absence. The serialised merge queue does. This deliberate split keeps the broker simple and fail-open.
- **Pull, not push.** Workers pull eligible envelopes, so heterogeneous agents with different throughput self-balance without a central scheduler tracking each agent's liveness in the hot path.
- **Auditable.** Classification rationale, envelope-sha, token holders, dependency edges, and status transitions are all persisted.

---

## 7. Detailed Mechanics

### 7.1 Classification (R1)

Each request is first classified. A fast **regex path** matches common intents (fix/build/deploy/migrate/refactor/test/review/docs/explain/schema/auth) to a `(kind, risk, suggested_role, confidence)` tuple. If no pattern matches with sufficient confidence, an **LLM fallback** returns strict JSON `{intent, kind, risk, est_files, est_tokens, suggested_role, requires_human_approval, rationale}`. Schema/auth/API-breaking changes are forced to `risk=HIGH, requires_human_approval=true`; pure information queries become `kind=read-only, risk=LOW`. The classification, its method (`regex|llm|default`), confidence, and rationale are persisted for audit (R11).

### 7.2 Envelope emission and the capability budget (R2, R3, C1, C5)

For a non-read-only request the planner emits one or more envelopes. An **envelope** is a frozen declarative object:

```
Envelope {
  title, description,
  agent_role,                    // builder | fixer | migrator | reviewer | tester | deployer
  allow_paths: string[],         // glob allow-list
  deny_paths: string[],          // glob deny-list (reserved files default here)
  max_files_changed: int,        // budget_files
  max_lines_changed: int,        // budget_lines
  may_add_dependencies: bool,    // false ⇒ manifest/lockfile edits forbidden w/o dep-lock token
  required_tokens: string[],     // reserved-file tokens needed before dispatch
  required_checks: string[],     // CI checks incl. "envelope-gate"
  feature_flag: string|null,
  depends_on: TaskId[],          // dependency edges
  risk, requires_human_approval
}
```

The planner derives `allow_paths` from the request (domain keywords → domain path globs), sets `deny_paths` to include the enumerated reserved files by default, sets the file/line budget, detects which reserved-file **tokens** the request implies (by keyword: dependency manifest → `dep-lock`; migration/schema → `db-migrations`; container build → `image-build`; deployment manifests → `infra`; server entrypoint/kernel → `kernel`), and includes `envelope-gate` among `required_checks`.

The frozen envelope is hashed to an **envelope-sha** (C5). The worker stamps this hash on the PR. At merge time the CI gate looks up the envelope by hash and checks the diff against *that* envelope — guaranteeing the policy enforced at merge is the policy issued at dispatch (R3, C1). If the diff:

- touches a path outside `allow_paths` or matching `deny_paths`,
- exceeds `max_files_changed` or `max_lines_changed`, or
- edits the dependency manifest/lockfile while `may_add_dependencies` is false and no `dep-lock` token is recorded,

the gate **fails closed** and the PR cannot merge.

### 7.3 Reserved-file token broker (R4, C2)

A **token** is a row keyed by `token_name` (one of the enumerated reserved names) with `held_by_task`, `held_by_agent`, `acquired_at`, and `expires_at`. Acquisition is a single atomic statement:

```
INSERT INTO serialising_tokens (token_name, held_by_task, held_by_agent, expires_at)
VALUES ($1, $2, $3, NOW() + ($4 || ' seconds')::interval)
ON CONFLICT (token_name) DO NOTHING
RETURNING *;
```

A returned row means **acquired**; an empty result means **held by someone else** → **requeue** (no blocking, no spin). Release is a keyed `DELETE` (`token_name` AND `held_by_task`, so only the holder can release). A **reaper** periodically `DELETE`s rows whose `expires_at < NOW()`, so a crashed holder cannot deadlock the

reserved file forever. The broker is **advisory**: a never-held token does not *prove* two PRs will not both touch the file; it merely makes dispatch avoid that case so the serialised merge queue rarely has to.

### 7.4 Automatic schema/code split (R5, C3)

When the planner detects the request implies the `db-migrations` token (migration/schema keywords), it emits **exactly two** envelopes instead of one:

1. **Schema envelope** — `allow_paths` scoped to migration/schema directories; `deny_paths` covering application code, UI, and deployment manifests; `required_tokens = [db-migrations]`; a tight budget (additive migrations only); `depends_on = []`. It ships **first**.
2. **Code envelope** — `allow_paths` scoped to application code; reserved files in `deny_paths`; carries any remaining tokens (with `db-migrations` removed); `depends_on = [<schema task id>]`. It is **blocked** from dispatch until the schema envelope merges.

A placeholder marker (`__SCHEMA__`) in the code envelope's `depends_on` is resolved to the concrete schema task id once both tasks are persisted. This makes expand/contract *migrate-before-code* a scheduler-enforced constraint rather than a human convention.

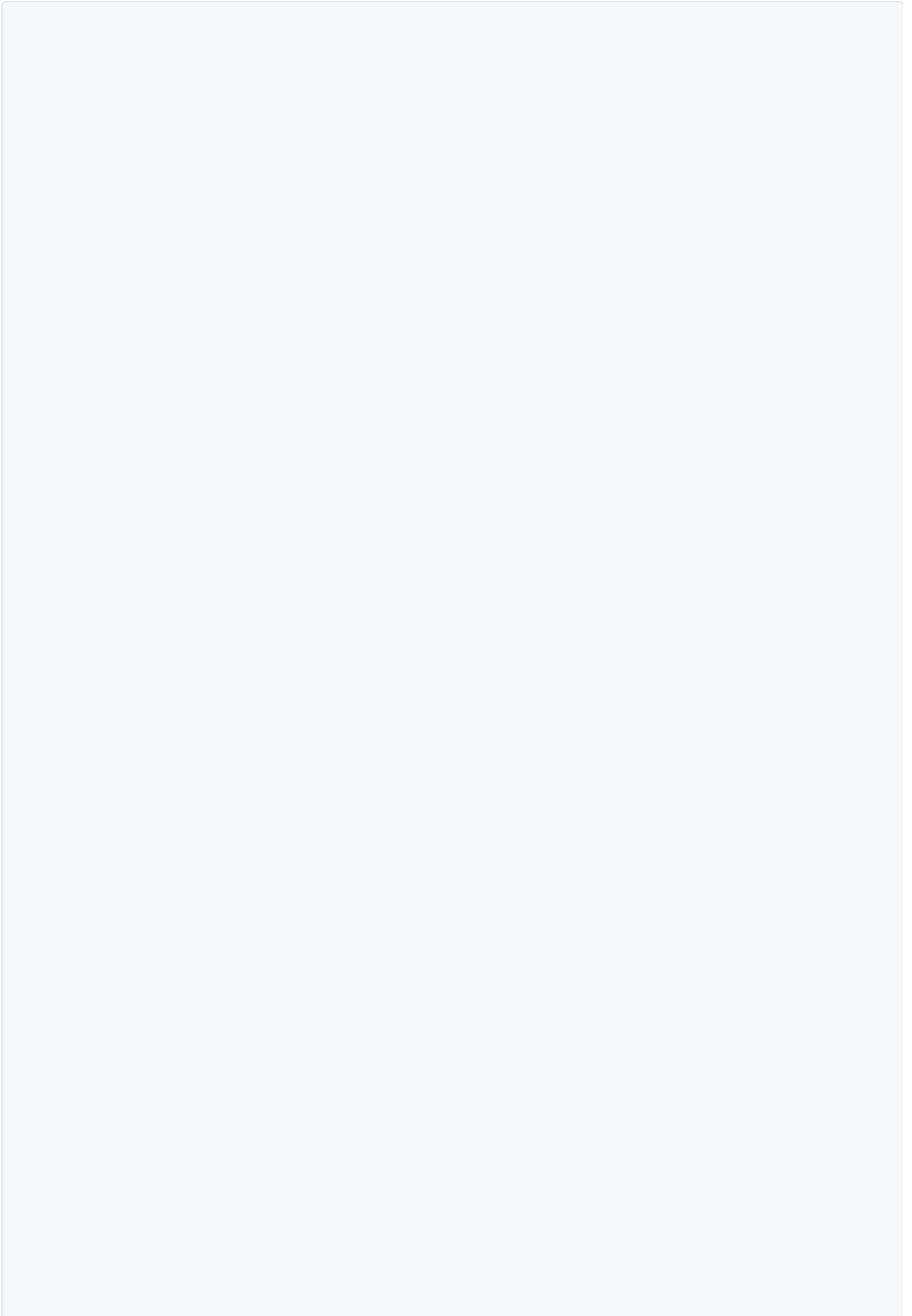
### 7.5 Pull-based, gated dispatch (R6, C4)

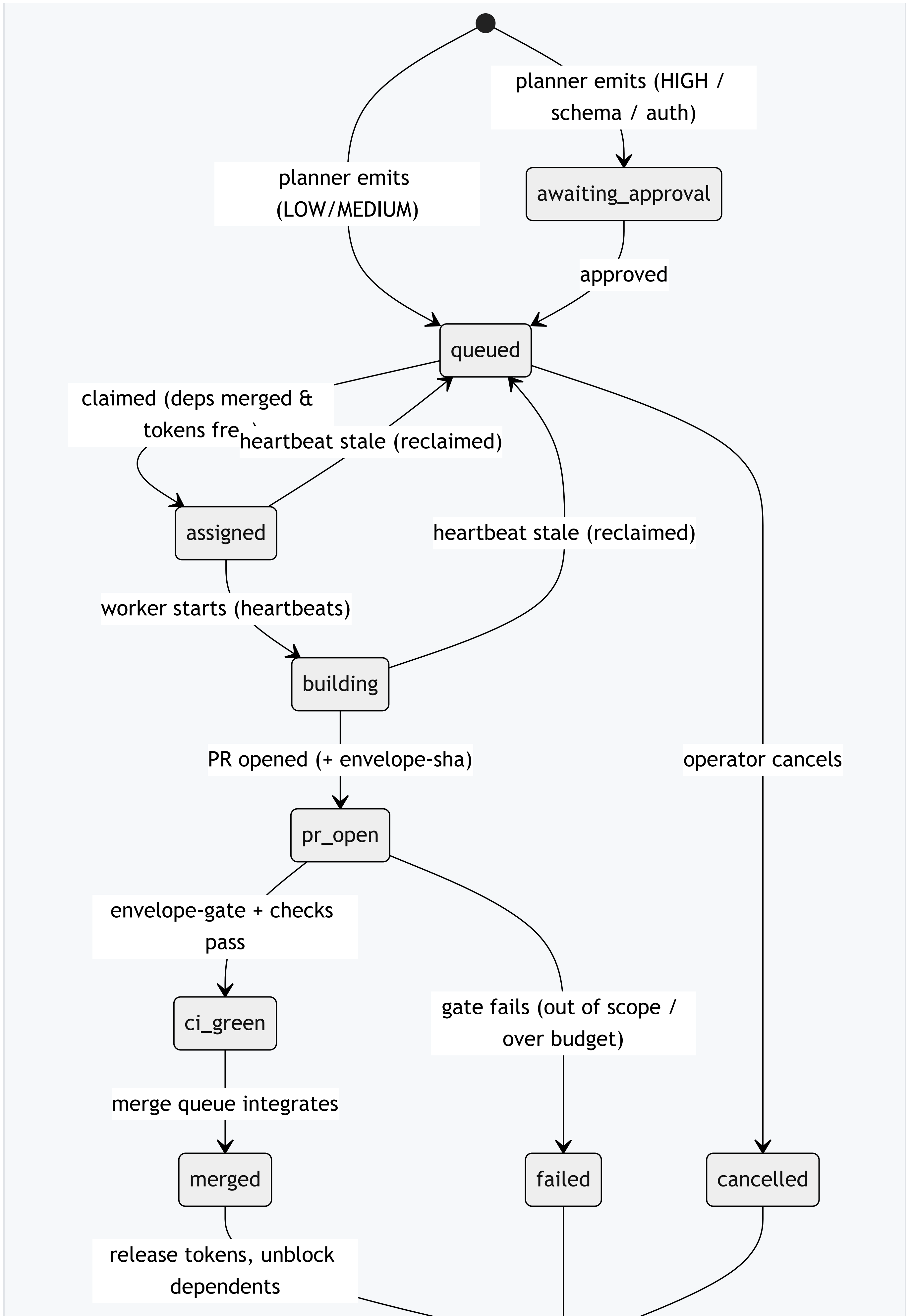
A worker of role `r` requests the next task. The dispatcher selects up to *K* queued tasks of role `r` ordered by creation time, and for each candidate checks two gates:

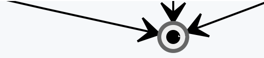
1. **Dependencies merged?** Every task id in `depends_on` must be in status `merged`.
2. **Tokens free?** None of `required_tokens` may currently be held (unexpired).

The first candidate passing both gates is claimed by an atomic conditional `UPDATE … WHERE status='queued' RETURNING *` (so two workers cannot both win), using `FOR UPDATE SKIP LOCKED` where the backend supports it. On a successful claim, the dispatcher **best-effort acquires** the candidate's tokens; per-token collisions are swallowed and simply re-checked on the next dispatch tick, keeping the path non-blocking.

> **Figure 2 — Envelope lifecycle state machine.**




planner emits (HIGH / schema / auth)
awaiting_approval
planner emits (LOW/MEDIUM)
approved
queued
claimed (deps merged & tokens fre
heartbeat stale (reclaimed)
assigned
heartbeat stale (reclaimed)
worker starts (heartbeats)
building
PR opened (+ envelope-sha)
operator cancels
pr_open
envelope-gate + checks pass
ci_green
gate fails (out of scope / over budget)
merge queue integrates
merged
failed
cancelled
release tokens, unblock dependents



### 7.6 Risk-gated approval and feature-flag derivation (R7)

If the classification (or planner risk rule) marks the envelope HIGH-risk or `requires_human_approval`, the envelope is persisted in status `awaiting-approval` and is *not* claimable until explicitly moved to `queued`. If the request introduces a user-facing surface (keywords like *new page/new app/user-facing/public*), the planner derives a `feature_flag` slug and records it on the envelope, so an incomplete surface ships flagged-off. Both controls live **inside the envelope** the gate enforces.

### 7.7 Liveness hygiene (R8)

A background loop performs two hygiene tasks each tick: it **reclaims stuck assignments** (status `assigned/building` whose `last_heartbeat_at` is older than a configurable timeout → reset to `queued`, releasing their tokens) and **reaps expired tokens** (`DELETE … WHERE expires_at < NOW()`). Together these guarantee no envelope is starved and no reserved file is locked by a dead holder.

### 7.8 Pseudocode (control flow)

```
on each dispatch tick:
  for req in requests where status = 'received' (oldest first, bounded):
     c = classify(req.prompt)            # regex fast-path, else LLM
     persist classification(c)
     if c.kind == 'read-only':
         set req.status = 'closed-info'; continue
     env_or_split = plan(req, c)         # 1 envelope, or [schema, code]
     if env_or_split is empty:
         set req.status = 'held-for-human-decomposition'; continue
     plan_id, task_ids = persist_plan(req, env_or_split)
     if split_on_db_migrations:
         code_task.depends_on = [schema_task_id]   # resolve __SCHEMA__
     set req.status = 'planned'
  reclaim_stuck(timeout)                 # heartbeat hygiene
  reap_expired_tokens()                  # TTL hygiene

worker(role) claims next task:
  for cand in queued tasks of role (oldest first, bounded):
     if not all(dep in merged for dep in cand.depends_on): continue
     if any(token held & unexpired for token in cand.required_tokens): continue
     claimed = UPDATE task SET status='assigned', agent=me
               WHERE id=cand.id AND status='queued' RETURNING *   # atomic
     if not claimed: continue
     for tok in cand.required_tokens: try acquire_token(tok)      # best effort
     return claimed
  return none

CI envelope-gate on PR with envelope_sha:
  env = lookup_envelope(envelope_sha)    # the SAME object that scoped dispatch
  diff = pr.diff()
  assert every changed path matches env.allow_paths and no deny_paths
  assert files(diff) <= env.max_files and lines(diff) <= env.max_lines
  assert not (touches_manifest(diff) and not env.may_add_dependencies
            and 'dep-lock' not in env.required_tokens)
  pass or fail-closed
```

### 7.9 Configuration points

| Config | Meaning | Illustrative default |
|---|---|---|
| `max_files_changed` / `max_lines_changed` | Per-envelope budget caps | 25 files / 800 lines |
| reserved token names | Enumerated reserved files | dep-lock, db-migrations, image-build, infra, kernel |
| token TTL | Lease lifetime before reap | 8 hours |
| heartbeat stale timeout | Reclaim threshold | 5 minutes |
| dispatch interval | Loop period | 5 seconds |
| schema-envelope budget | Tight cap for additive migrations | 5 files / 200 lines |
| required checks | CI checks incl. `envelope-gate` | gate, lint, tests, secret-scan |

# 8. Data Model

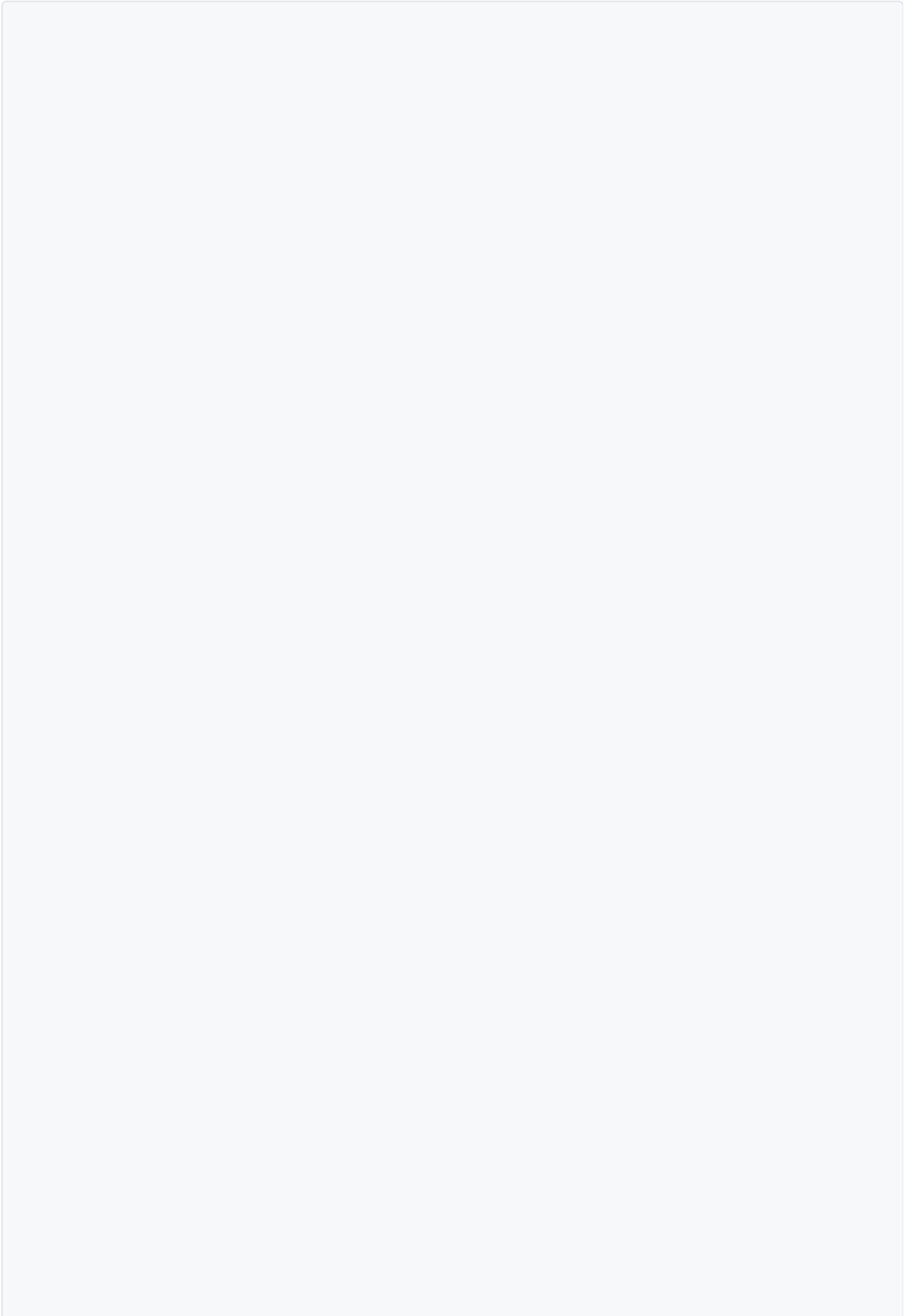
## 8.1 Tables

| Table | Purpose | Key columns |
|---|---|---|
| `request_classifications` | Phase-2 output, one per request | `request_id` (PK), `kind`, `risk`, `suggested_agent_role`, `requires_human_approval`, `method`, `confidence`, `rationale` |
| `task_plans` | Parent record of an emitted plan | `plan_id` (PK), `request_id`, `status`, `envelope_count`, `notes` |
| `agent_tasks` | The **envelopes** / work units | `task_id` (PK), `plan_id`, `agent_role`, `allow_paths` (JSONB), `deny_paths` (JSONB), `required_tokens` (JSONB), `required_checks` (JSONB), `budget_files`, `budget_lines`, `feature_flag`, `risk`, `depends_on` (JSONB), `status`, `branch`, `pr_number`, `pr_url`, `envelope_sha`, `last_heartbeat_at` |
| `serialising_tokens` | Reserved-file mutex broker | `token_name` (PK), `held_by_task`, `held_by_agent`, `acquired_at`, `expires_at` |
| `agent_workers` | Worker registry | `agent_id` (PK), `agent_role`, `capabilities` (JSONB), `status`, `current_task_id`, `last_seen_at` |

Status values are validated against an allow-set at write time (e.g., task status ∈ {queued, awaiting-approval, assigned, building, pr_open, ci_green, merged, failed, cancelled, completed, closed-info}); risk ∈ {LOW, MEDIUM, HIGH}; token names ∈ the enumerated reserved set. All SQL is parameterised.

## 8.2 Entity-relationship diagram

> **Figure 3 — Data model (ER).**



| REQUEST_CLASSIFICATIONS | | |
|---|---|---|
| varchar | request_id | PK |
| varchar | kind | |
| varchar | risk | |
| varchar | suggested_agent_role | |
| boolean | requires_human_approval | |
| varchar | method | |
| double | confidence | |
| text | rationale | |

scopes

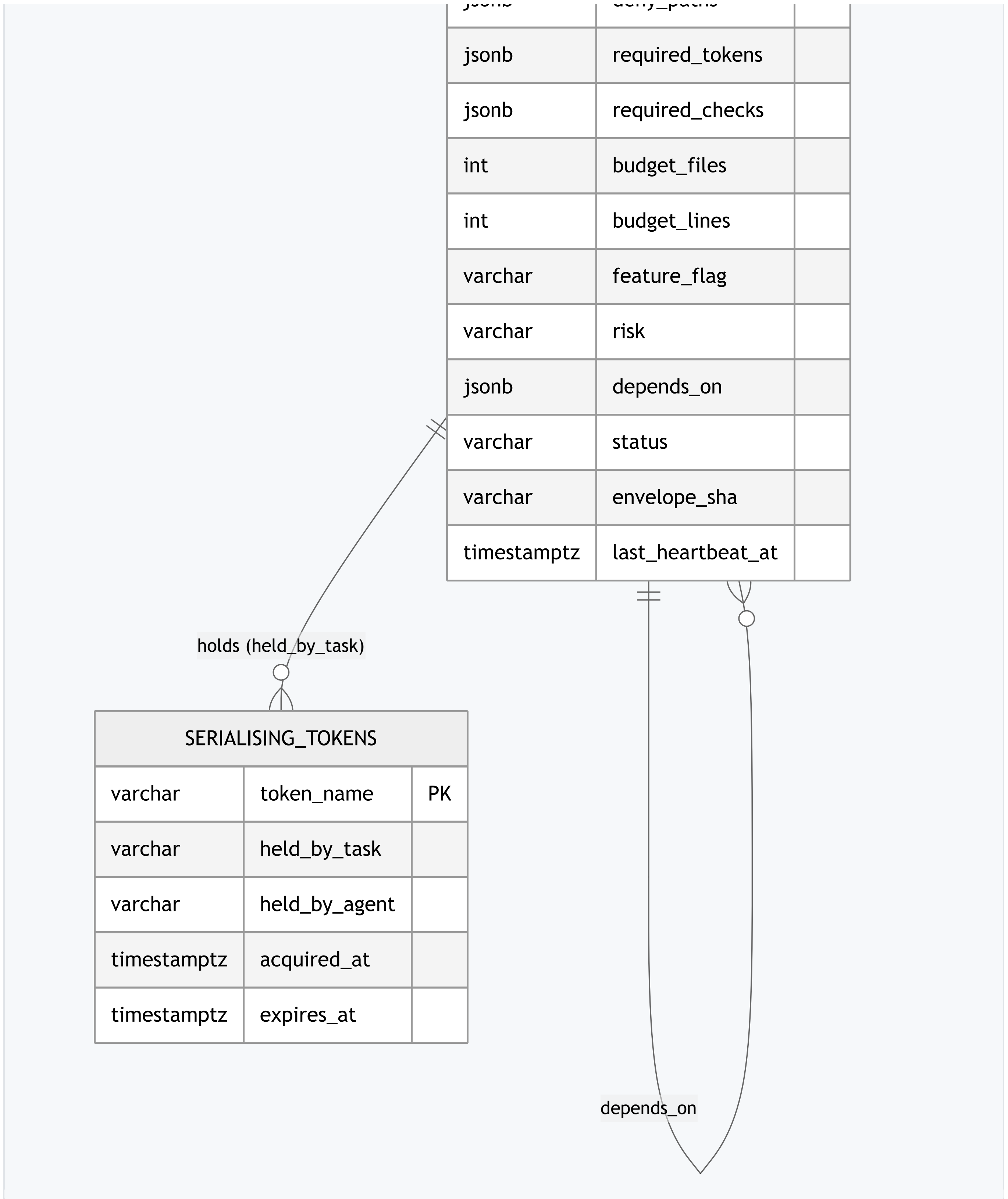
| TASK_PLANS | | |
|---|---|---|
| varchar | plan_id | PK |
| varchar | request_id | FK |
| varchar | status | |
| int | envelope_count | |
| text | notes | |

| AGENT_WORKERS | | |
|---|---|---|
| varchar | agent_id | PK |
| varchar | agent_role | |
| jsonb | capabilities | |
| varchar | status | |
| varchar | current_task_id | |
| timestamptz | last_seen_at | |

emits envelopes

claims (agent_id)

| AGENT_TASKS | | |
|---|---|---|
| varchar | task_id | PK |
| varchar | plan_id | FK |
| varchar | agent_role | |
| jsonb | allow_paths | |
| jsonb | deny_paths | |




### 8.3 Indices and invariants

- Indexes on `agent_tasks(status, created_at)` and `agent_tasks(agent_role, status)` keep the dispatch candidate scan cheap; an index on `serialising_tokens(expires_at)` keeps the reaper cheap.
- **Invariant I1:** a token row exists ⇒ exactly one task believes it holds that reserved file (until TTL expiry).

- **Invariant I2:** a code envelope produced by the split has a non-empty `depends_on` whose target is the sibling schema envelope.
- **Invariant I3:** `envelope_sha` on a task uniquely identifies the frozen envelope used by the CI gate.

---

## 9. Reference Implementation & Enablement

### 9.1 What `src/` demonstrates

The clean-room reference implementation in `src/` reduces the invention to practice with no external dependencies (Node.js standard library only). It contains:

- **`token-broker.js`** — an in-memory analogue of the atomic conditional-insert broker: `acquire` returns the lease on first claim and `null` on conflict; `release` is holder-keyed; `reapExpired` removes TTL-expired leases. This is the §7.3 mechanism without a database, so the *semantics* (atomic single-winner, advisory, TTL-reaped) are testable in isolation.
- **`planner.js`** — classification-to-envelope emission including the §7.4 automatic schema/code split, reserved-token detection, default allow/deny path scoping, file/line budget, and `envelope-sha` computation (a stable hash of the frozen envelope).
- **`envelope-gate.js`** — the §7.2 merge-time predicate: given an envelope and a synthetic diff, it verifies path allow/deny, file/line caps, and the may-add-dependencies rule, returning a pass/fail with reasons. Because it consumes the *same* envelope object the planner froze, it embodies the dispatch≡merge identity (C1).
- **`selfcheck.js`** — an executable example that wires the three together: it plans a mixed migration+code request (asserting a two-envelope split with a dependency edge), exercises token acquire/conflict/reap, and runs the envelope-gate on both an in-budget and an out-of-budget diff.

### 9.2 How it enables the claims

- **Claim 1 / C1 (dispatch≡merge identity):** the gate is fed the identical envelope object the planner emitted, matched by `envelope-sha`; the self-check shows a diff passing at the issued budget and failing when it strays out of `allow_paths` or over `max_files`.
- **C2 (token broker):** the broker's `acquire` returns single-winner semantics; a second concurrent `acquire` returns `null`; an expired lease is reaped and re-acquirable.
- **C3 (schema/code split):** the planner returns two envelopes for a mixed request, with the code envelope's `depends_on` referencing the schema envelope.

### 9.3 Configuration points exposed

The reference implementation surfaces the §7.9 knobs (budgets, reserved token names, TTL, reserved-path globs) as plain parameters/constants, making it adaptable to any repository layout without code changes to the core logic. It intentionally omits production concerns (auth, persistence, a real CI integration) — those are sketched in docs/OPEN-SOURCE-APP.md — to keep the disclosure focused on the novel mechanism.

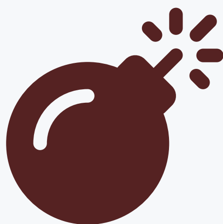
---

## 10. Worked Example / Scenario

**Request:** *"Create a migration to add a `priority` column to the leads table and surface a Priority filter on the leads list page."* This touches **both** a database migration **and** application code. (The classifier keys on the explicit migration/schema language to imply the `db-migrations` token; the same request phrased without a schema keyword would not trigger the split.)

1. **Intake** stores the request as `received`.
2. **Classifier** matches the schema keyword → `kind=modify`, `risk=HIGH`, `requires_human_approval=true`, role `builder`. Rationale persisted.
3. **Planner** detects the `db-migrations` token implication and performs the **split**:
   - **Schema envelope** `T-schema`: `allow_paths=[migrations/**, schemas/**]`, `deny_paths=[app code, ui, deploy]`, `required_tokens=[db-migrations]`, budget 5 files / 200 lines, `depends_on=[]`.
   - **Code envelope** `T-code`: `allow_paths=[crm domain/**]`, reserved files denied, `required_tokens=[]`, budget 25 files / 800 lines, `depends_on=[T-schema]`.
   - Both are `awaiting-approval` because risk is HIGH.
4. **Approval** moves both to `queued` (or only `T-schema` becomes eligible since `T-code` is dependency-blocked regardless).
5. **Dispatch.** A `builder` worker pulls `T-schema`: its `depends_on` is empty and `db-migrations` is free → claimed; the dispatcher acquires the `db-migrations` token (TTL 8h). `T-code` is **not** claimable (its dependency `T-schema` is not yet `merged`).
6. The schema worker opens a PR (additive migration), stamps `envelope-sha(T-schema)`.
7. **Envelope-gate** re-checks the PR diff against `T-schema`'s budget: only migration paths touched, within 5 files / 200 lines → **pass**. The serialised merge queue integrates it; on merge the `db-migrations` token releases.
8. Now `T-code`'s dependency is `merged` → it becomes claimable. A worker pulls it, edits only the CRM domain, opens a PR with `envelope-sha(T-code)`.
9. **Envelope-gate** re-checks against `T-code`'s budget. If the agent had also edited the dependency lockfile (out of `allow_paths`, and `may_add_dependencies=false` with no `dep-lock` token), the gate **fails closed** and the PR cannot merge until re-scoped. Otherwise it passes and merges.

> **Figure 4 — Worked-example sequence.**



Syntax error in text
mermaid version 11.15.0

## 11. Security, Safety & Failure Modes

### 11.1 Fail posture

- **Capability gate fails *closed*.** If the envelope-gate cannot positively verify the diff against the envelope (missing envelope-sha, unparsable budget, diff out of scope), the PR does **not** merge. Scope enforcement is safety-critical; ambiguity blocks.
- **Token broker fails *open*.** If the broker is unavailable, dispatch may proceed without serialising a reserved file. This is deliberate: the broker is advisory, and the **serialised merge queue remains the authoritative conflict catcher**. Availability of dispatch is preferred over a broker that could wedge the whole fleet.

### 11.2 Failure-mode table

| Failure mode | Effect | Mitigation |
|---|---|---|
| Crashed token holder | Reserved file appears permanently held | TTL `expires_at` + reaper auto-release |
| Two workers claim the same task | Double work / wasted compute | Atomic `UPDATE … WHERE status='queued' RETURNING *` (single winner) |
| Stuck assignment (worker died mid-task) | Task never progresses | Heartbeat staleness → reclaim to `queued`, release tokens |
| Envelope-sha mismatch at PR | Gate cannot bind policy to dispatch | Fail closed; require re-stamp from the issued envelope |
| Planner under-scopes `allow_paths` | Legitimate edit blocked by gate | Re-scope envelope (operator) and re-issue; gate is the right place to discover this early |
| Broker unavailable | No reserved-file serialisation | Fail open; merge queue catches the rare collision |
| Migration ordinal collision across agents | Two migrations same ordinal | `db-migrations` token serialises migration authoring; merge queue backstops |

### 11.3 STRIDE-style notes (control-plane)

| Threat | Concern | Disposition |
|---|---|---|
| **S**poofing | A worker claims a role it is not authorised for | Worker registry + role check at claim; out of scope of the novel mechanism but required in production |
| **T**ampering | An agent edits a file outside its budget | Envelope-gate fails closed on out-of-scope diff |
| **R**epudiation | "I never went out of scope" | Persisted `envelope_sha`, diff, gate result are auditable |
| **I**nformation disclosure | Reserved-file contents leaked | Not introduced by this mechanism; standard repo ACLs apply |
| **D**enial of service | Holding a reserved token forever | TTL + reaper bound the hold |
| **E**levation of privilege | Bypassing approval for HIGH-risk | `awaiting-approval` state is non-claimable until explicit approval |

### 11.4 Privacy

The mechanism operates on repository paths, diffs, and task metadata. It introduces no personal data. The illustrative `src/` contains no secrets, credentials, hostnames, namespaces, or customer data.

---

## 12. Standards & Framework Mapping

We claim **semantic alignment**, not certified compliance.

| Framework / Standard | Relevant clause / concept | How this work aligns |
|---|---|---|
| Principle of least privilege (NIST SP 800-53 AC-6 family) | Grant only the access needed for a task | The capability budget is a per-task least-privilege contract on repository paths |
| Separation of duties (AC-5) | Split high-risk operations | Schema/code split and reserved-file tokens separate high-blast-radius work |
| Change management / SoD in CD (e.g., SLSA-style provenance intent) | Verifiable link from intent to artifact | `envelope-sha` binds the merged diff to the issued envelope |
| Supply-chain integrity (dependency hygiene) | Control unsanctioned dependency additions | `may_add_dependencies=false` + `dep-lock` token gate manifest edits |
| Expand/contract DB release pattern | Migrate-before-code ordering | Automatic schema/code split with a dependency edge |
| Mutual exclusion / leasing (distributed-systems canon) | Bounded, recoverable locks | TTL leases with a reaper; advisory relative to the merge queue |
| Policy-as-code / merge-time policy gates | Machine-checked diff policy | Envelope-gate is policy-as-code whose policy *is* the dispatch envelope |

We explicitly do **not** assert conformance certification to any of these; the mapping documents conceptual alignment for auditors.

---

## 13. Evaluation Methodology

Numbers below are **illustrative** placeholders describing *how* one would measure the effect, not measured results.

| Dimension | Metric | How to measure | Interpretation |
|---|---|---|---|
| Scope soundness | % PRs passing the envelope-gate first try | gate pass/fail logs | Higher = planner scoping matches agent behaviour |
| Out-of-scope catch rate | # diffs blocked for path/budget violation | gate failure reasons | Each catch = a reviewer-untangle avoided |
| Reserved-file collisions | merge-queue conflicts on reserved files / 100 PRs | merge-queue logs, before vs after broker | Lower = broker is shaping dispatch effectively |
| Dispatch≡merge drift | # cases where merge policy ≠ dispatch budget | should be **0 by construction** | Non-zero indicates an envelope-sha binding bug |
| Migrate-before-code violations | # code PRs merged before their schema dependency | dependency-edge audit | Should be 0 (scheduler-enforced) |
| Liveness | mean time a stuck task waits before reclaim | heartbeat timestamps | Bounded by the stale timeout |
| Token starvation | max hold time on a reserved token | token acquired/released timestamps | Bounded by TTL |

A credible study would A/B a fleet with and without the envelope-gate + broker, holding agent mix and request stream constant, and report conflict rate, first-try gate pass rate, and reviewer-time per PR with confidence intervals. **Any quantitative figure quoted elsewhere from this design should be treated as illustrative until such a study is run.**

---

## 14. Novelty & Inventive Claims

The following are stated in prose for prior-art purposes. They are **not** patent claims being asserted by the publisher; they delineate the contribution so others may freely practice it.

**Claim 1 (independent).** A computer-implemented method for orchestrating a plurality of heterogeneous AI code-generation agents writing to a shared version-controlled repository, comprising: classifying a natural-language change request to derive a kind and a risk label; emitting, from the classification, a planner data object comprising one or more task envelopes, each encoding (i) an agent-role assignment, (ii) a declarative capability budget specifying allow_paths, deny_paths, a maximum number of changed files and a maximum number of changed lines, and (iii) a may-add-dependencies flag; wherein, responsive to detecting that the request modifies both database-migration files and application-code files, the planner automatically emits exactly two envelopes — a schema envelope scoped to migration paths and gated on a database-migrations serialising token, and a code envelope scoped to code paths carrying a dependency edge that blocks dispatch until the schema envelope is merged; dispatching each envelope to a matching agent only after its dependency edges resolve and none of its required reserved-file tokens are held, said token acquired by an atomic conditional database insert keyed on an enumerated reserved repository file path and auto-released on TTL expiry by a reaper; and, at pull-request time, re-verifying the same allow_paths, deny_paths and file/line budget carried in the envelope against the pull-request diff in a continuous-integration gate, such that the dispatch-time capability scope is identical to the merge-time enforced contract.

**Claim 2.** The method of claim 1, wherein the envelope is frozen and reduced to a content hash (an envelope-sha) that is stamped on the pull request, and the continuous-integration gate retrieves the

envelope by said hash so the policy enforced at merge is provably the policy issued at dispatch.

**Claim 3.** The method of claim 1, wherein the re-verifying step fails closed, blocking merge, whenever the pull-request diff touches a path outside allow_paths, matches a deny_paths entry, exceeds the maximum changed files, or exceeds the maximum changed lines.

**Claim 4.** The method of claim 1, wherein the re-verifying step blocks merge when the diff modifies a dependency manifest or lockfile while the may-add-dependencies flag is false and no dependency-manifest serialising token is recorded against the task.

**Claim 5.** The method of claim 1, wherein the reserved-file token broker is advisory and a serialised merge queue is the authoritative conflict catcher, the broker shaping dispatch ordering to reduce, without proving the absence of, reserved-file collisions.

**Claim 6.** The method of claim 1, wherein the atomic conditional insert is an INSERT ... ON CONFLICT DO NOTHING returning the inserted row on success and an empty result on conflict, an empty result causing the requesting envelope to be requeued without blocking.

**Claim 7.** The method of claim 1, wherein the reserved repository file paths are enumerated as a fixed set comprising a dependency manifest, database migrations, a container build file, deployment manifests, and a server entrypoint, each associated with a distinct named serialising token.

**Claim 8.** The method of claim 1, wherein dispatch is pull-based: a worker of a given role claims the next eligible envelope via an atomic conditional update guarded by a skip-locked selection, and upon claiming, best-effort acquires the envelope's required tokens.

**Claim 9.** The method of claim 1, further comprising reclaiming an assignment whose worker heartbeat has not been observed within a configurable timeout by resetting the envelope to a queued state and releasing its held tokens.

**Claim 10.** The method of claim 1, wherein an envelope classified as high risk or as a schema or authorization change is persisted in a non-claimable approval state until an explicit approval transitions it to a claimable state, the approval control being a field of the same envelope re-verified at merge.

**Claim 11.** The method of claim 1, further comprising deriving, for a request that introduces a user-facing surface, a feature-flag identifier recorded on the envelope so that an incomplete surface ships disabled.

**Claim 12.** The method of claim 1, wherein the code envelope's dependency edge is initially a placeholder marker that is resolved to the concrete schema envelope's task identifier once both envelopes are persisted.

**Claim 13.** The method of claim 1, wherein the classifying step uses a regular-expression fast path and, when no pattern matches above a confidence threshold, a large-language-model fallback returning a structured classification with a rationale that is persisted for audit.

**Claim 14.** The method of claim 1, wherein the schema envelope is assigned a tighter file/line budget than the code envelope to constrain it to additive migrations.

**Claim 15.** The method of claim 1, wherein the required continuous-integration checks recorded on the envelope include the envelope-gate among other checks, and the merge is blocked until all recorded checks pass.

**Claim 16.** A system comprising a relational store of envelopes, serialising tokens, and worker registrations; a planner; a pull-based dispatcher; a token broker; and a continuous-integration envelope-gate, configured to perform the method of any of claims 1–15, wherein all envelope state transitions are validated against an allow-set of statuses at write time.

---

## 15. Limitations & Threats to Validity

- **Combination, not components.** Every sub-mechanism (conditional-insert mutex, TTL leases, skip-locked queues, allow/deny budgets, PR-time gates, expand/contract) is known prior art. The contribution is their combination and the dispatch≡merge identity. A patent examiner applying an obviousness standard may find the combination of familiar elements unpatentable — which is precisely why it is published defensively.
- **Advisory broker.** The token broker does not guarantee collision-freedom; it reduces collisions and relies on the merge queue. Under broker outage, collisions can rise.
- **Planner scoping quality.** Scope soundness depends on the planner choosing correct `allow_paths`. A naive keyword scoper will sometimes under- or over-scope; the gate surfaces this early but does not author the scope. A repository-aware (file-walking) planner is stronger.
- **Heterogeneous-agent compliance.** Agents that ignore their runtime scope are still caught at merge (fail closed), but they waste compute; the method bounds damage, not waste.
- **Illustrative numbers.** §13 contains no measured results; the effect is plausible and architecturally sound but unmeasured in this document.
- **Withheld detail.** Production concerns (authentication, RBAC at claim time, the concrete CI integration, and operator tooling) are intentionally summarised, not fully specified, to keep the disclosure focused and free of any operator-specific infrastructure.

---

## 16. Future Work & Open-Source Reference App

Planned directions:

- **Repository-aware planner** that walks the tree to compute precise `allow_paths` instead of keyword heuristics.
- **Measured study** (§13) producing real conflict-rate and first-try gate-pass numbers.
- **Richer reserved-file model** beyond a fixed enumeration (e.g., owner-declared hot files per directory).
- **Open-source reference app** — a standalone *task-envelope dispatcher* any team can run in front of its agents. Its minimal architecture, mapping to this invention, and a generic Kubernetes/AKS deployment sketch (vanilla `kubectl`/Helm, no operator-specific identifiers) are in docs/OPEN-SOURCE-APP.md.

---

## 17. Conclusion

Concurrent multi-agent coding fails in two structural ways that branch isolation cannot fix: unbounded per-task scope and reserved-file races. This publication describes a single coherent answer — a planner-

emitted **capability budget** that is **identical at dispatch and at merge**, a non-blocking **reserved-file token broker** that serialises high-blast-radius edits, and an **automatic schema/code split** that orders migrations before the code that depends on them. None of the parts is individually novel; the value is in the combination and the closed loop, and the value of *publishing* it is to keep that combination free for everyone to use. This document, dated 2026-06-25, is offered as prior art to that end.

---

## Appendix A — Prior-Art Landscape (well-trodden vs candidate-novel)

**Well-trodden (clearly prior art; we build on these):**

- LLM decomposition of large changes (Rosie-style; Ziftci et al., FSE 2025).
- Heterogeneous agent dispatch and Git merge queues (2026 control planes).
- Postgres `SELECT … FOR UPDATE SKIP LOCKED` job queues.
- `INSERT … ON CONFLICT DO NOTHING` as a mutex; TTL-lease + fencing-token locks with reapers.
- Single-agent allow/deny path budgets.
- PR-time path/size policy gates (danger-style rules, static-analysis path filters, code-quality size checks).
- Expand/contract migrate-before-code release discipline.

**Candidate-novel (the asserted delta):**

- **Dispatch≡merge identity** — the *same* capability budget is the agent's runtime scope and the re-verified merge-time contract, bound by an envelope-sha.
- **Reserved-file token broker keyed on enumerated repository file paths**, advisory relative to a serialised merge queue, integrated into pull dispatch.
- **Automatic schema/code envelope split** with a scheduler-enforced migrate-before-code dependency edge and a token-gated schema leg.
- The **combination** of the above into one closed loop driven by one classification.

**Honesty attestation.** The prior-art review summarised here is **directional, not exhaustive.** It reflects a good-faith pre-publication screen, not a full USPTO/EPO full-text claim-chart search. No claim of absolute novelty is made; the publication's purpose is defensive — to establish dated public prior art so the technique remains free to practice.

---

## Appendix B — Glossary

| Term | Definition |
|---|---|
| **Task envelope** | A frozen declarative object scoping one unit of agent work: role, allow/deny paths, file/line budget, may-add-dependencies, tokens, checks, flag, dependencies. |
| **Capability budget** | The allow_paths/deny_paths/max_files/max_lines/may_add_dependencies portion of an envelope. |
| **Dispatch≡merge identity** | The property that the budget enforced at PR time is the identical object that scoped the agent at dispatch. |
| **Envelope-sha** | A content hash of the frozen envelope, stamped on the PR, used by the gate to bind policy to dispatch. |
| **Reserved file** | An enumerated high-blast-radius repository file (dependency manifest, migrations, container build file, deployment manifests, server entrypoint). |
| **Serialising token** | A TTL-bounded advisory lease on a reserved file, acquired by atomic conditional insert. |
| **Reaper** | A periodic process that deletes expired tokens (and reclaims stale assignments). |
| **Envelope-gate** | The CI step that re-verifies a PR diff against its envelope's budget. |
| **Schema/code split** | Automatic emission of a schema envelope plus a dependency-blocked code envelope for a request touching both. |
| **Merge queue** | The serialised integration mechanism that is the authoritative conflict catcher. |

## Appendix C — Reference-Implementation Index

| File | Role | Demonstrates |
|---|---|---|
| `src/token-broker.js` | In-memory reserved-file token broker | Atomic single-winner acquire, holder-keyed release, TTL reap (C2) |
| `src/planner.js` | Classification → envelope emission | Allow/deny scoping, file/line budget, token detection, schema/code split, envelope-sha (C1, C3, C5) |
| `src/envelope-gate.js` | Merge-time predicate | Re-verifies a diff against the same envelope budget; fail-closed (C1, R3) |
| `src/selfcheck.js` | Executable example | Wires all three; asserts split, token semantics, and gate pass/fail |
| `src/README.md` | Guide | What it shows, how to run, clean-room disclaimer |

## Appendix D — Defensive-Publication Deposit & Timestamp

- **Publication date:** 2026-06-25.
- **Publisher:** Gus IT LLC (Florida, USA).
- **Author:** Gustavo Assuncao, PhD.
- **Version:** 1.0.
- **Deposit channel:** to be assigned (IP.com / Zenodo / arXiv) — establishes a public, dated, citable prior-art record. No DOI is asserted in this version.
- **Intent:** This disclosure is intentionally public to bar later patenting of the described combination by others and to keep the technique freely practiceable. It is licensed under AGPL-3.0-or-later, which

includes an express patent grant for the contributed material. The dated public availability of this document constitutes prior art as of the publication date above.