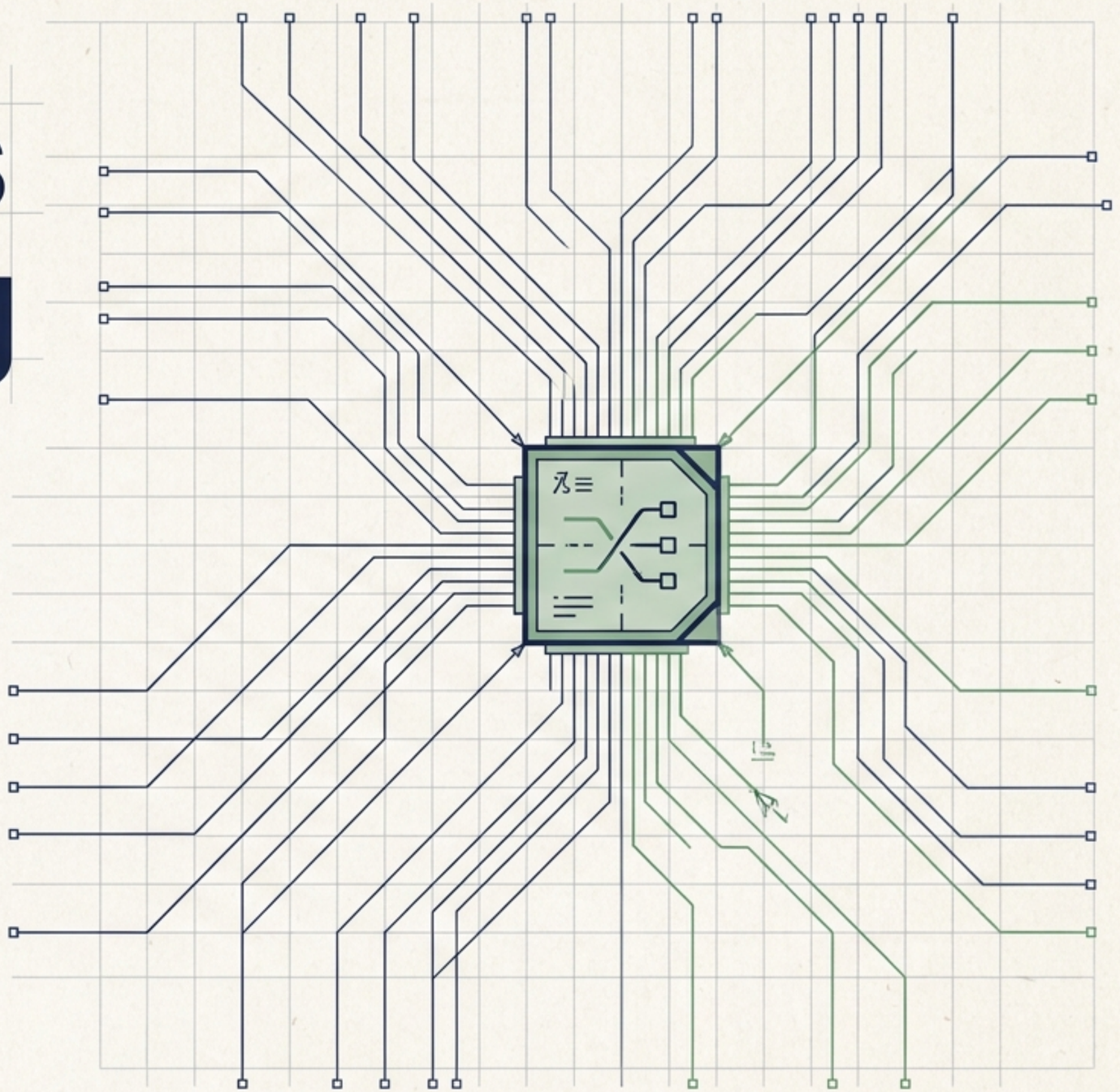
The Physics of AI Coding

> From monolithic prompts to intelligent switchboards: an evolutionary study of AI coding harnesses, context limits, and workflow routing.

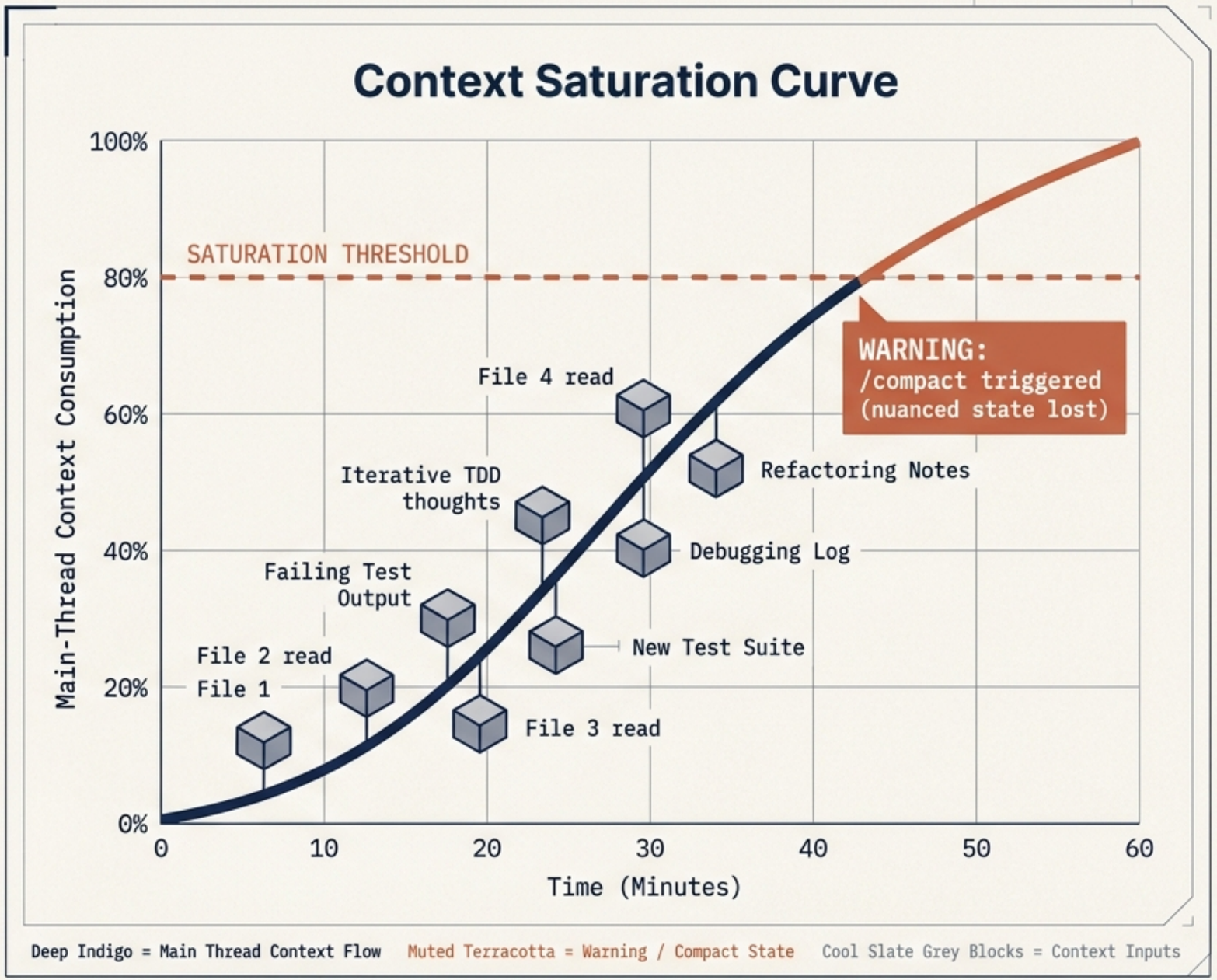
Based on the engineering logs of xingfanxia (April 2026)



The Session That Broke Vanilla Claude

Vanilla Claude Code is exceptional for single-file, single-function tier-1 work. But without scaffolding, it forces a single main thread to hoard every file read, test output, and thought.

Key Insight:
On a 10-file feature with iterative TDD, the main thread becomes a context landfill, hitting 80% saturation in an hour.



An Evolutionary Taxonomy of AI Frameworks

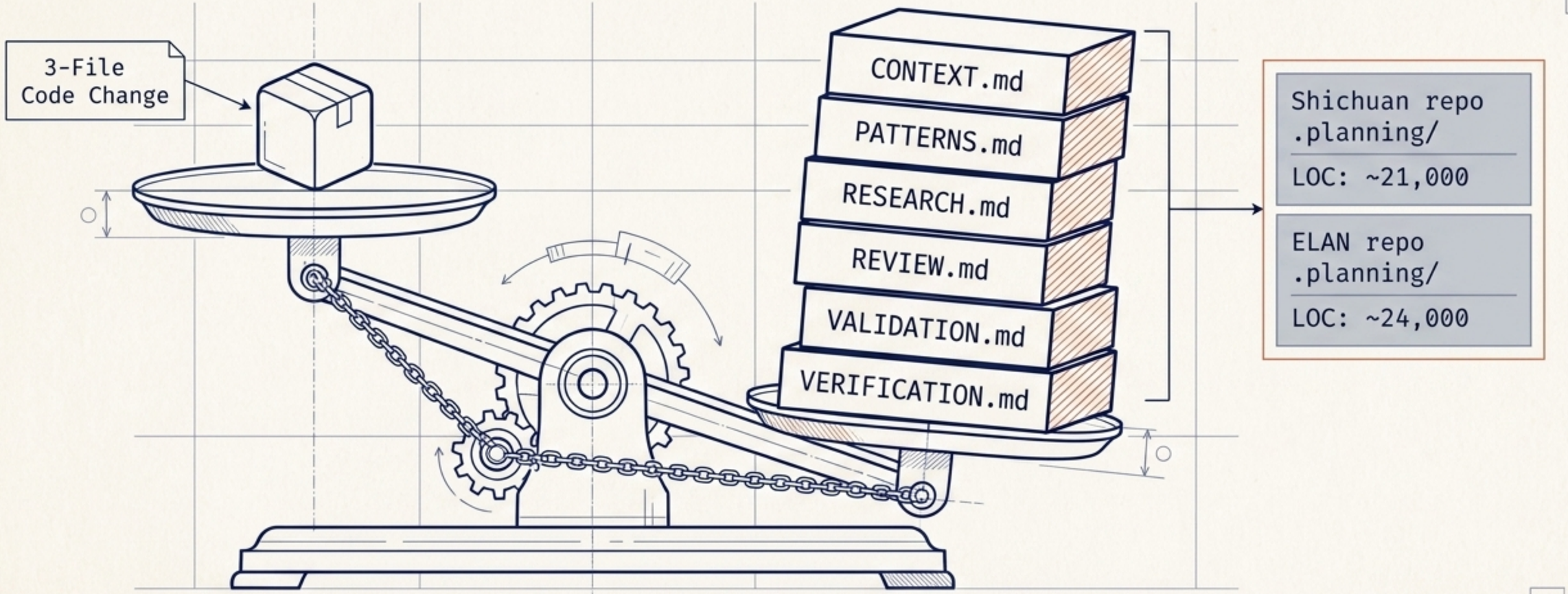
To solve the context ceiling, the harness architecture had to evolve. Over two months, three distinct architectural species were tested, each adapted to specific development environments—and each breaking when taken out of its niche.

	Vanilla	GSD (get-shit-done)	Superpowers
Core Mechanism	Main session reads/edits.	Multi-agent phase chains (discuss -> plan -> execute).	14 general-purpose composable skills.
Ideal For	Typos, hotfixes, Q&A.	Schema migrations, cross-phase dependencies.	Spec drift, standard features.
The Break Point	>3 files, iterative tests.	Solo dev / Tier 3 work.	Long autonomous runs.

Deep Indigo = Architectural Structure / Primary Text; Cool Slate Grey = Secondary Data / Row Alternation; Muted Terracotta = Break Points / Warnings

GSD and the Weight of Ceremony

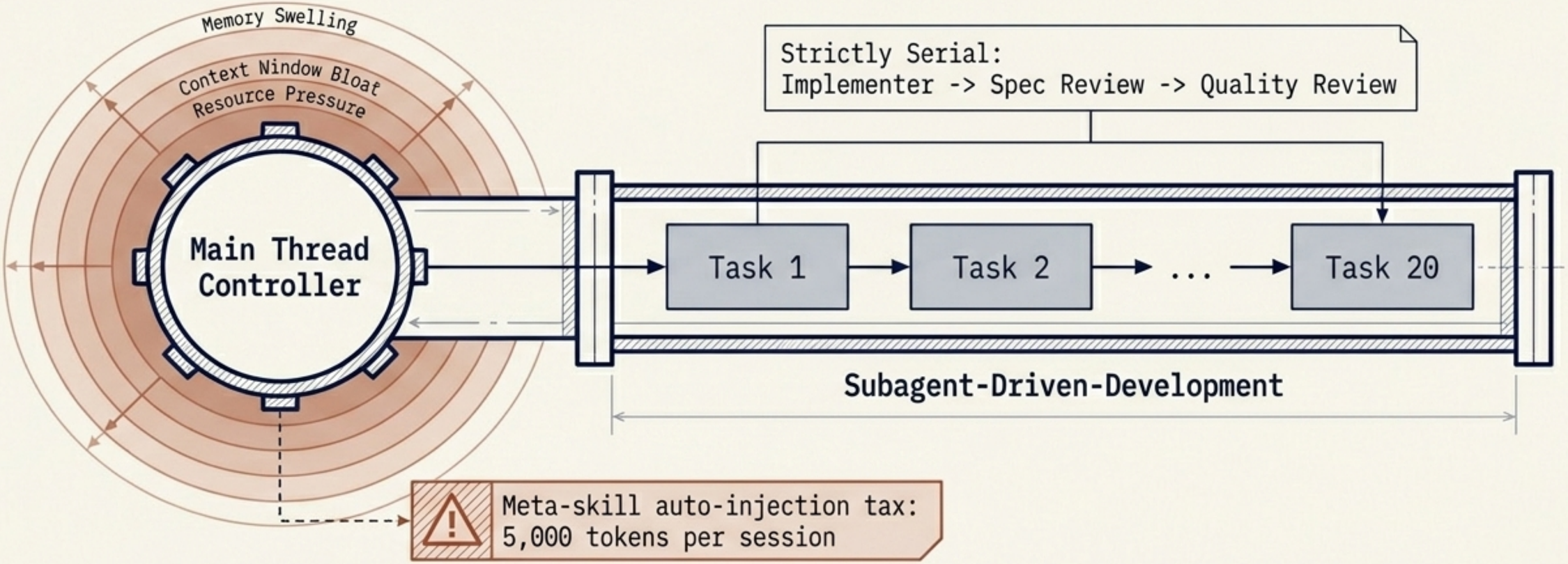
GSD excels at tier-4, highly structured work. But for solo dev and tier-3 tasks, it is over-engineered. Reading through six different planning artifacts takes longer than writing the actual code.



Deep Indigo = Architectural Structure / Primary Text; Cool Slate Grey = Secondary Data / Component Inputs; Muted Terracotta = Break Points / Warnings

Superpowers and the Serial Bottleneck

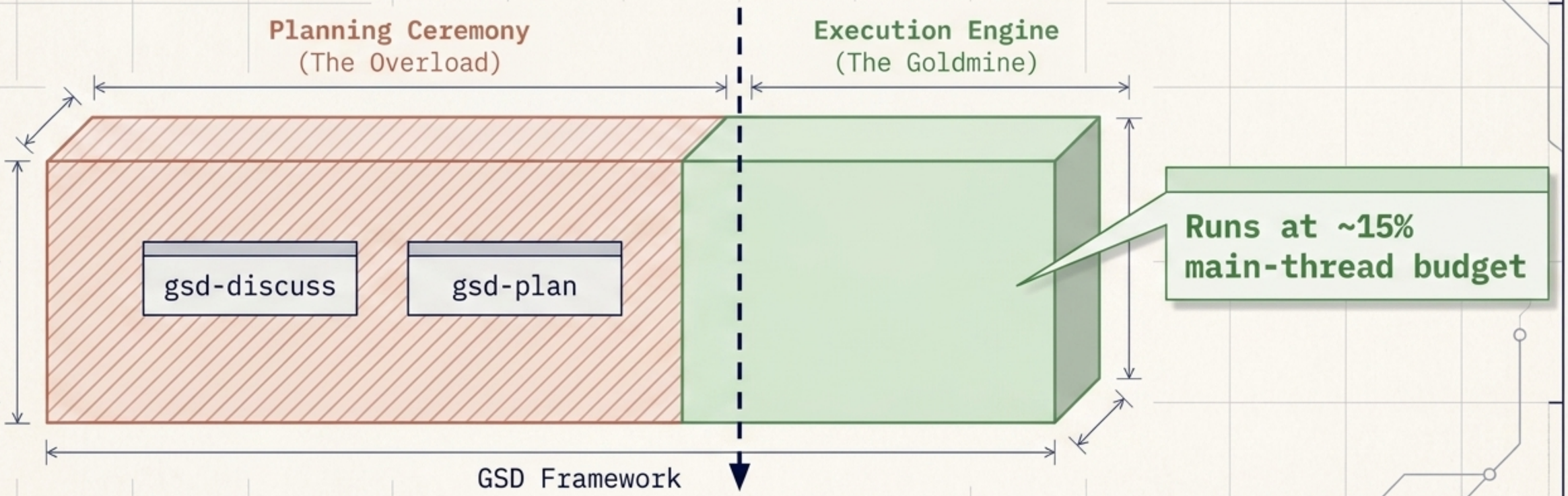
Superpowers offers brilliant composable skills and catches spec drift. However, it explicitly forbids parallel implementation to avoid merge conflicts. On a 20-task plan, the controller must coordinate a strict sequence, swelling its memory and consuming the context window.



Deep Indigo = Architectural Structure / Primary Text; Cool Slate Grey = Secondary Data / Component Inputs; Muted Terracotta / Warnings; Crisp Sage Green = Success / Efficient Paths

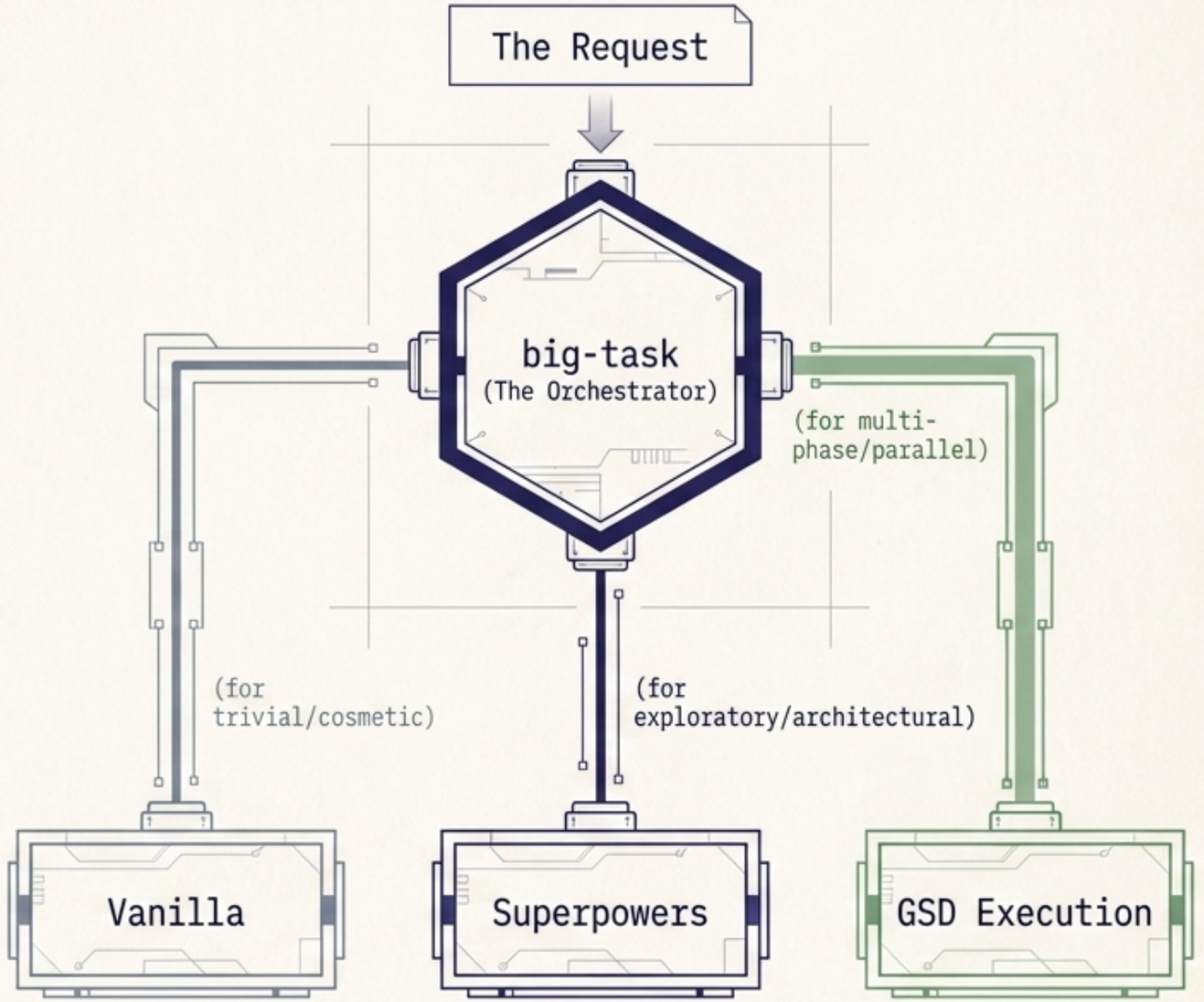
The Conflation Error

The desire to throw out GSD entirely was based on a conflation error. The planning overhead is crushing, but the execute-phase engine itself is remarkably lean. The solution wasn't replacing frameworks—it was extracting the best mechanics and stitching them together.



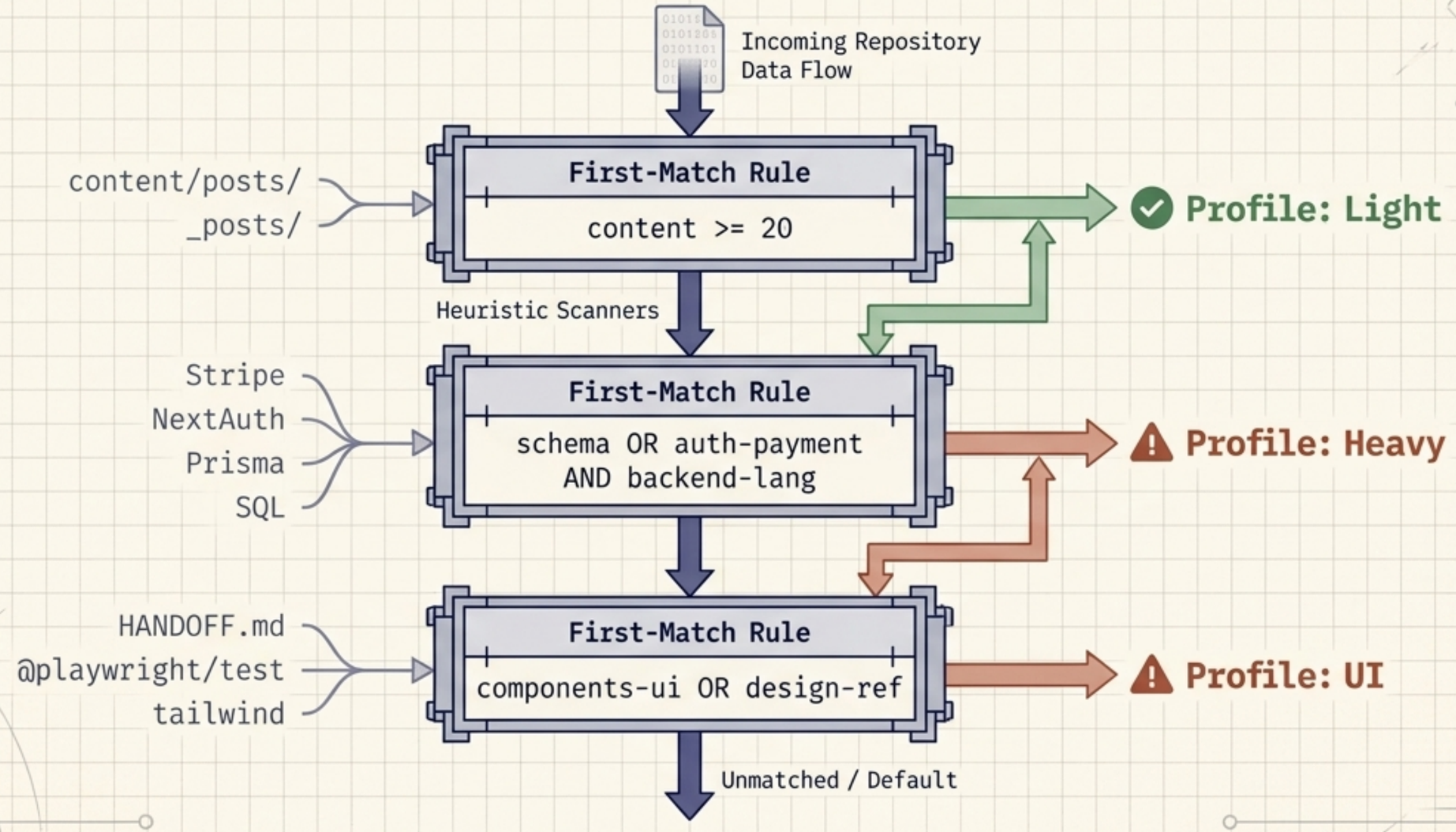
Harness 4: The Intelligent Router

big-task acts purely as an orchestrator. It does not re-implement TDD or subagents; it hides the other three frameworks behind a single name. Its only job is evaluating the shape of the work and triggering the correct underlying harness.



Phase 0.0: Auto-Detecting Project Shape

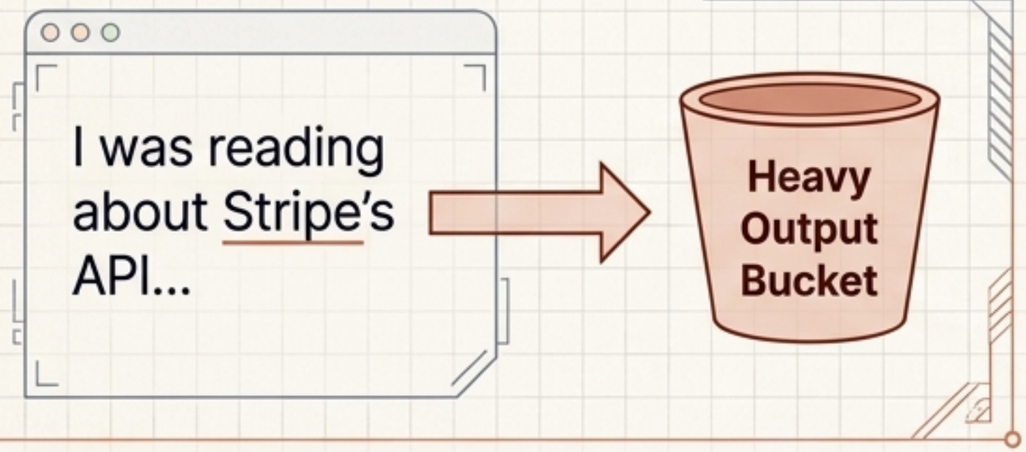
Before any decisions are made, the orchestrator classifies the repository into one of four workflow profiles using pure bash heuristics. A 270-post blog stays **Light**, while a repo with SQL migrations flags as **Heavy**.



Phase 0.5: Task Intent Override

A repository's shape doesn't dictate the current request's shape. Adding a Stripe paywall to a light blog is a heavy task. But relying on keyword matching inside an LLM context is a massive anti-pattern. **Intent** requires evaluating system impact, blast radius, and reversibility.

Regex-over-Natural-Language ✗



Fails: Triggers heavy workflow based purely on word presence.

Task Intent (Judgment) ✓

What does this do to the system?

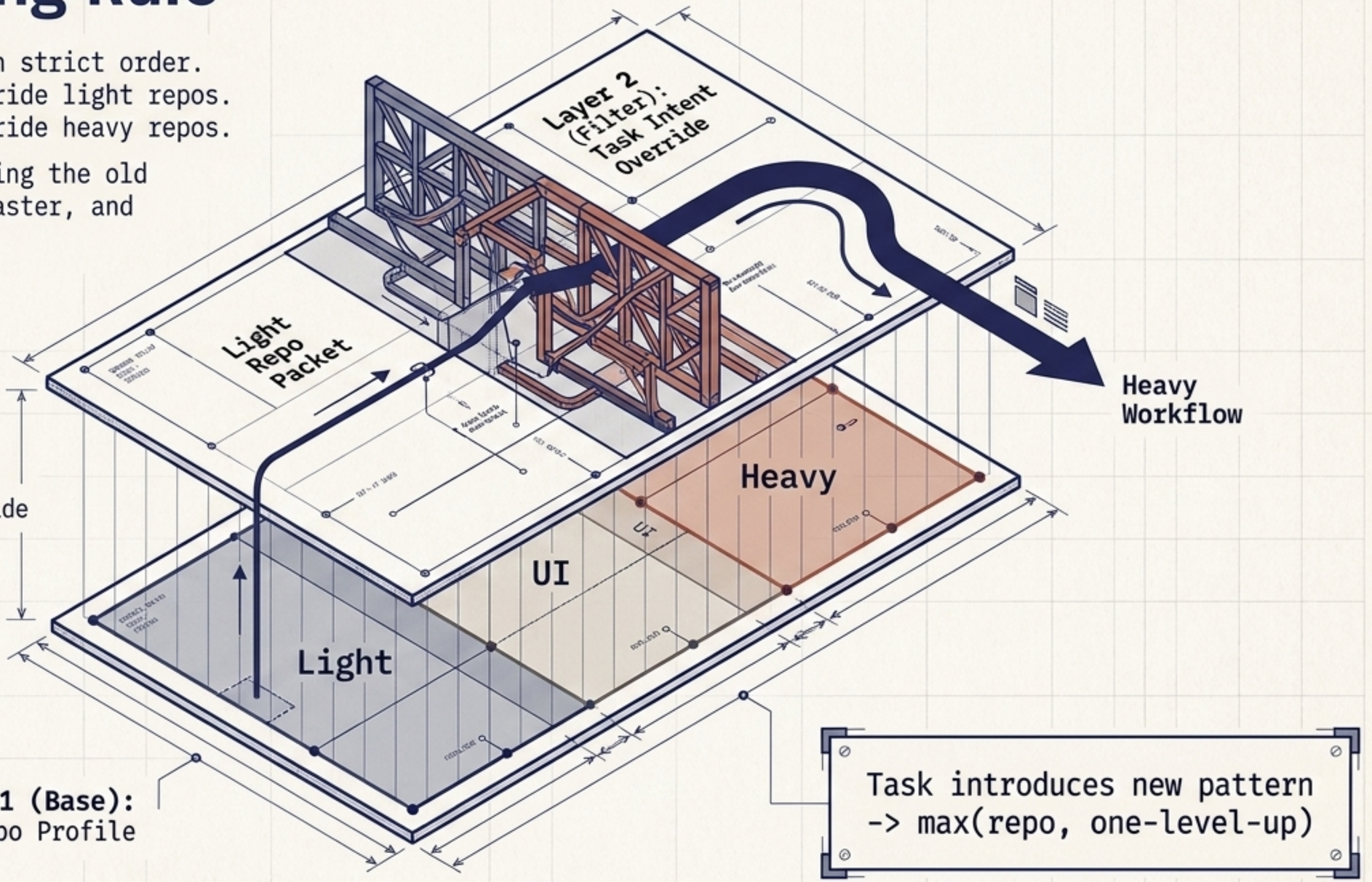
- **Heavy:** Persistence changes, trust boundaries, revenue paths.
- **Light:** Prose, cosmetic tweaks.

Succeeds: Evaluates system impact over vocabulary.

The Combining Rule

The router applies rules in strict order. Heavy-by-nature tasks override light repos. Light-by-nature tasks override heavy repos.

The resulting logic replacing the old keyword list is shorter, faster, and vastly more accurate.



Subagent Dispatch Modes

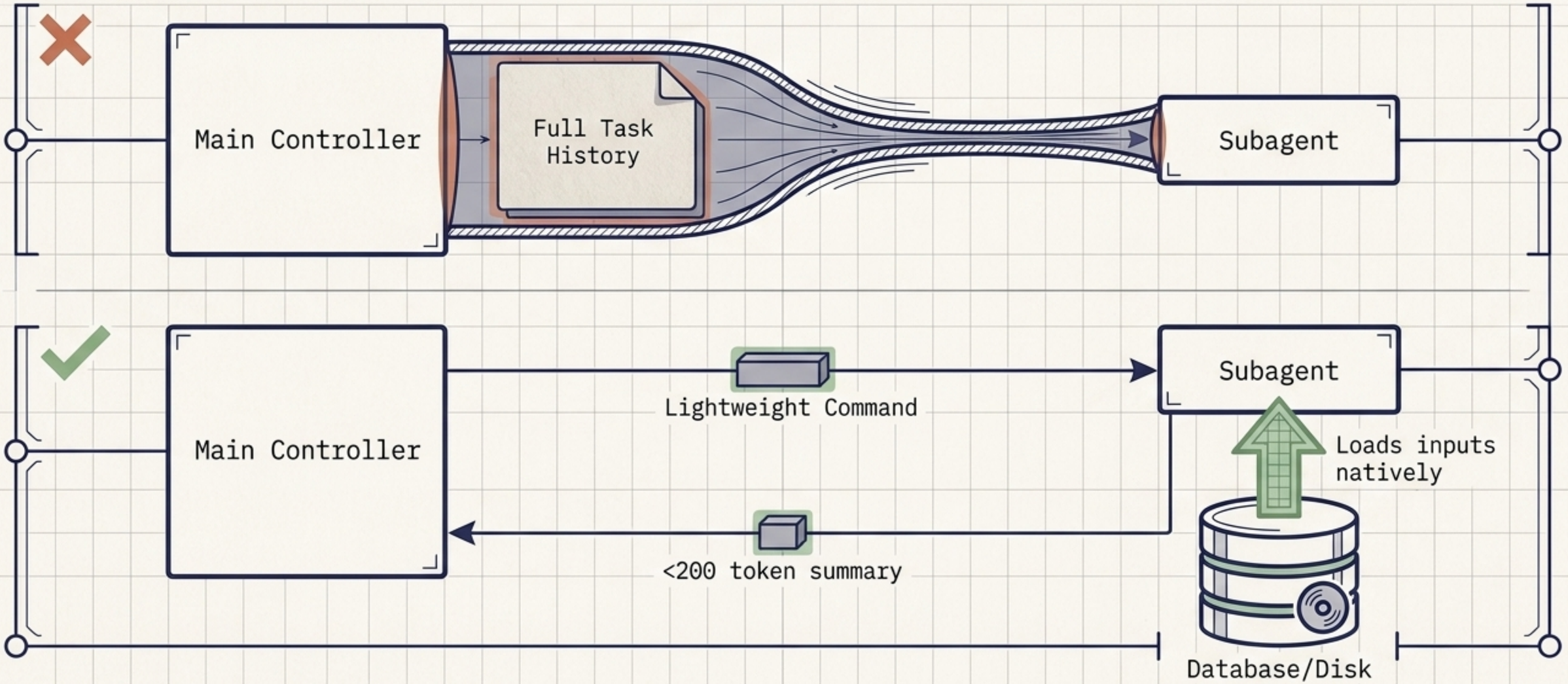
Every phase in the orchestration carries a strict dispatch policy. By explicitly announcing the mode (e.g., Phase 2b · `parallel-readonly`), routing becomes auditable and prevents accidental inline overuse.

Mode	When to trigger	Main-thread consumption
`parallel-worktree`	Implementation on disjoint files. Subagents write in own worktrees.	<20% 
`parallel-readonly`	Investigation, audit, visual verification. Fan out one per target.	<20% 
`serial-subagent`	Implementation on shared files. Fresh per task.	~30% 
`inline`	Single-file change, Tier 2 hotfix.	100% 

RULE: Never inline when tier >= 3 AND independent task count >= 3.

Orchestrator Discipline

The dispatch mode only works if discipline holds. The controller must never inject **full task text** into subagent prompts. State must live on disk, subagents must **self-load** their inputs, and returns must be heavily structured summaries under 200 tokens.



Match the Harness to the Work Shape

The mistake is picking a harness first and forcing work to fit it. GSD wastes tokens on small features; Superpowers serializes long runs; Vanilla blows up context on medium work. The ultimate harness isn't a single engine—it's a system that routes by shape.

Left side (The Work Shape)	Right side (The Target Engine)
Trivial change →	**Vanilla**
Content / Cosmetic →	**Vanilla** OR **Light-profile big-task**
Standard feature, locked design →	**Superpowers** (subagent-driven-development)
Multi-phase, parallelizable across files →	**GSD execute-engine** (lean orchestrator)
Unclear spec, architectural decisions →	**Superpowers** (brainstorming -> spec -> plan chain)