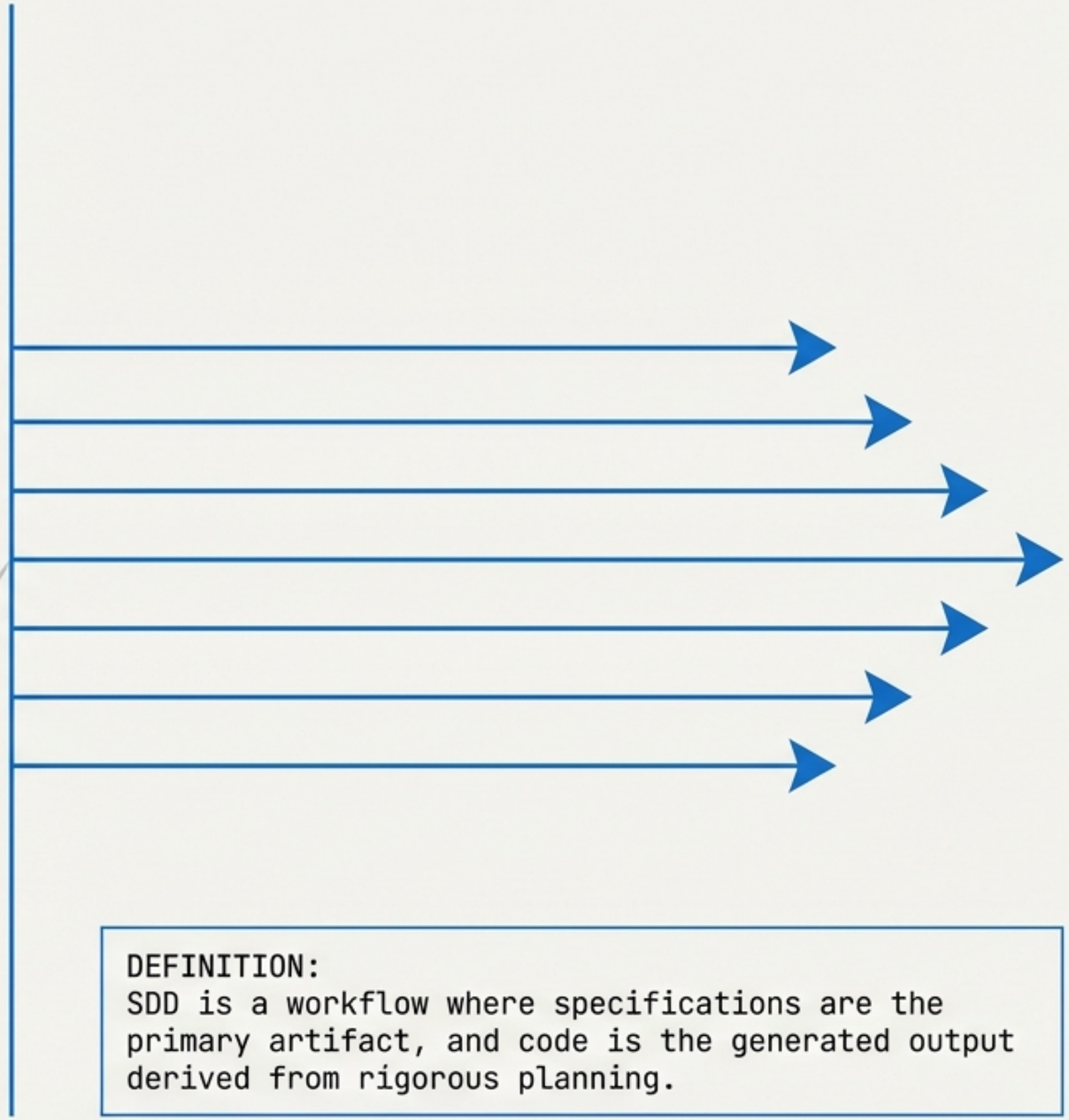
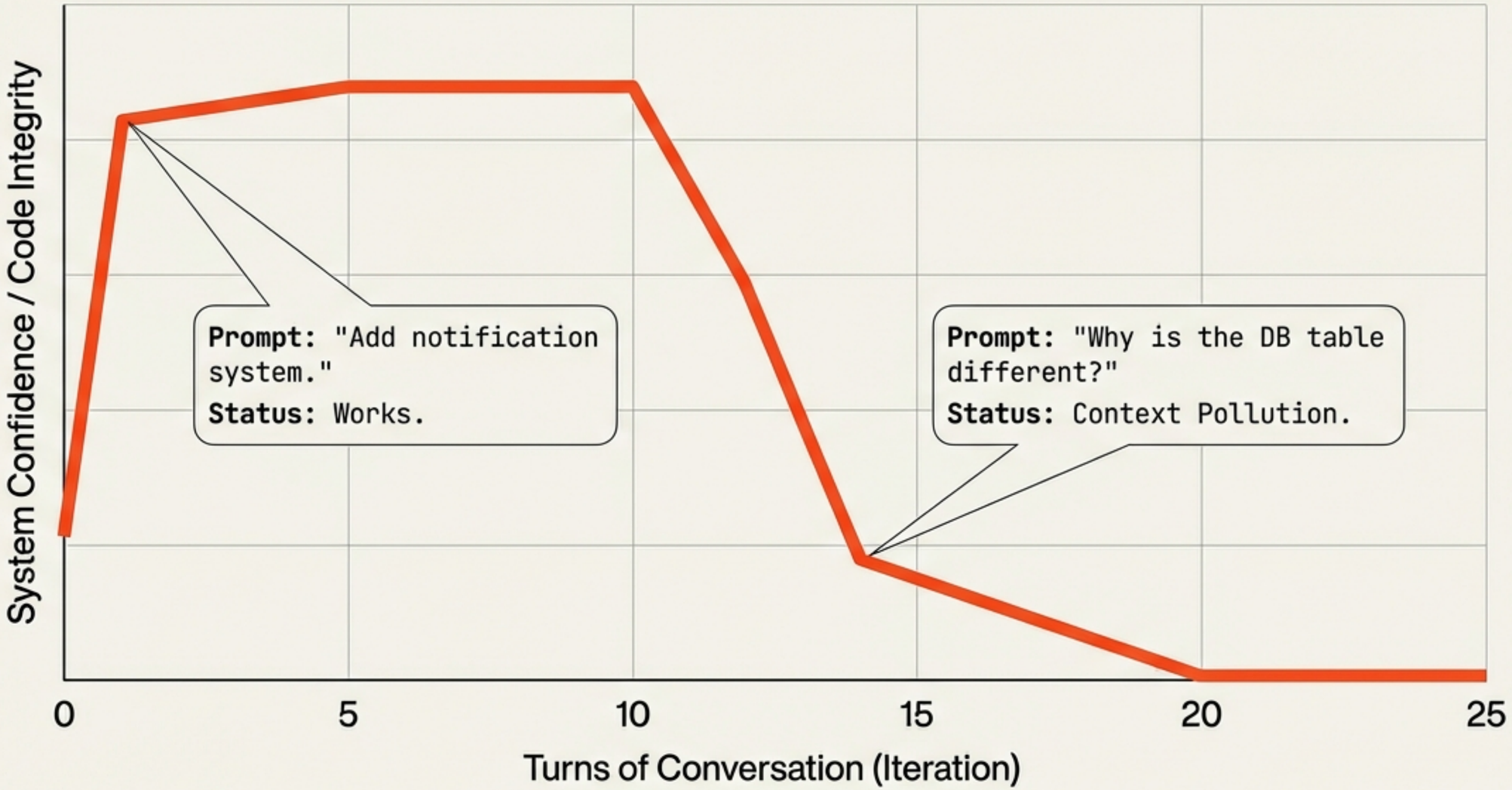
Spec-Driven Development with Claude Code

Moving from Vibe Coding to Production-Ready Orchestration



DEFINITION:
SDD is a workflow where specifications are the primary artifact, and code is the generated output derived from rigorous planning.

The Ceiling of Vibe Coding: The Cycle of Failure

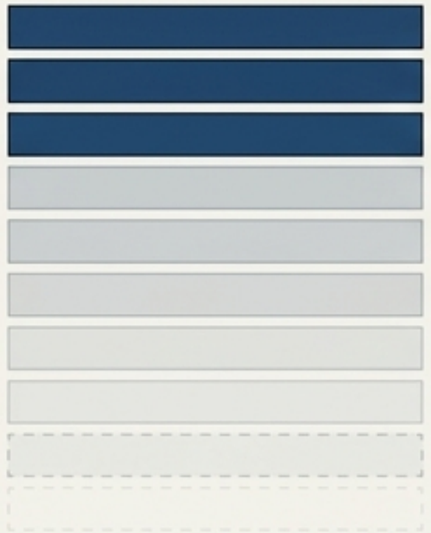


It works for a function. It fails for a system.

Three Structural Failure Modes

Diagnosing the breakdown of conversational workflows.

1. Context Loss



Mechanism: Iterative discovery erases earlier constraints. Newer information overrides older nuance.

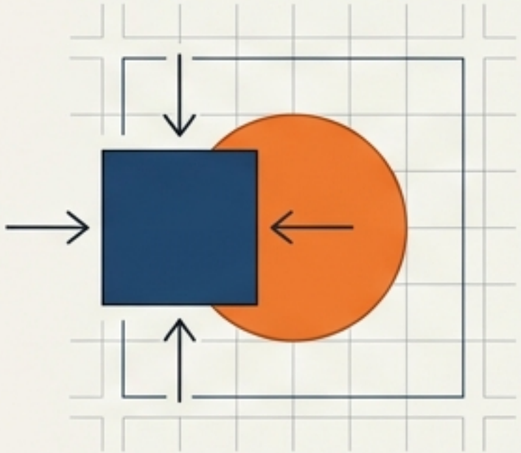
"Features that worked in Turn 5 break in Turn 15."

2. Assumption Drift

Mechanism: AI fills silence with "reasonable defaults" that diverge from specific developer intent.

"Code looks clean but uses a foreign architecture."

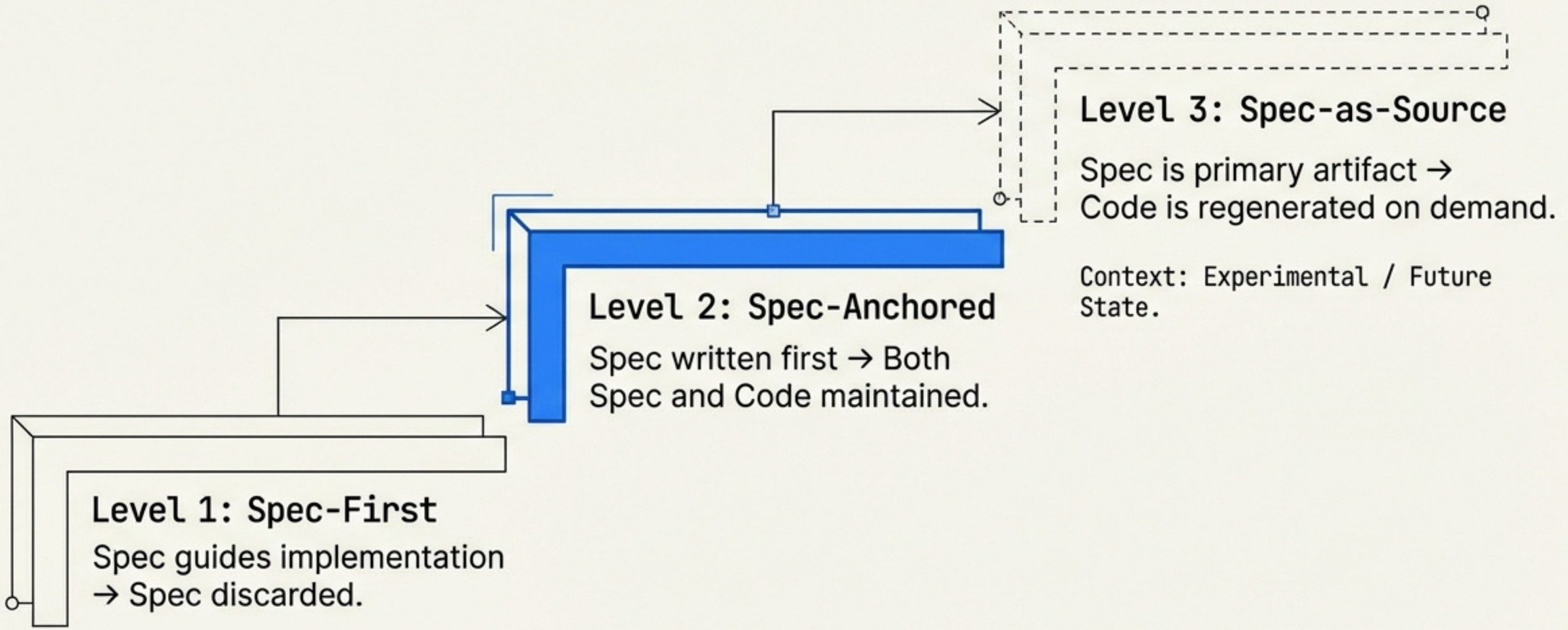
3. Pattern Violations



Mechanism: Generic training data clashes with your specific project patterns.

"Introducing a new DB table when a unified event bus already exists."

Inverting Control: From Chatting to Orchestrating



Insight: Front-loading context prevents the AI from guessing. In SDD, the Specification is the source of truth.

The 4-Phase Workflow

Separating Planning from Execution

Phase 1: Research

Parallel Subagents.
Input: Goal.
Output: Investigation Summaries.

Phase 2: Specification

The Contract.
Input: Findings.
Output: spec.md.

Phase 3: Refinement

The Interview.
Input: Draft Spec.
Output: Decisions.

Phase 4: Implementation

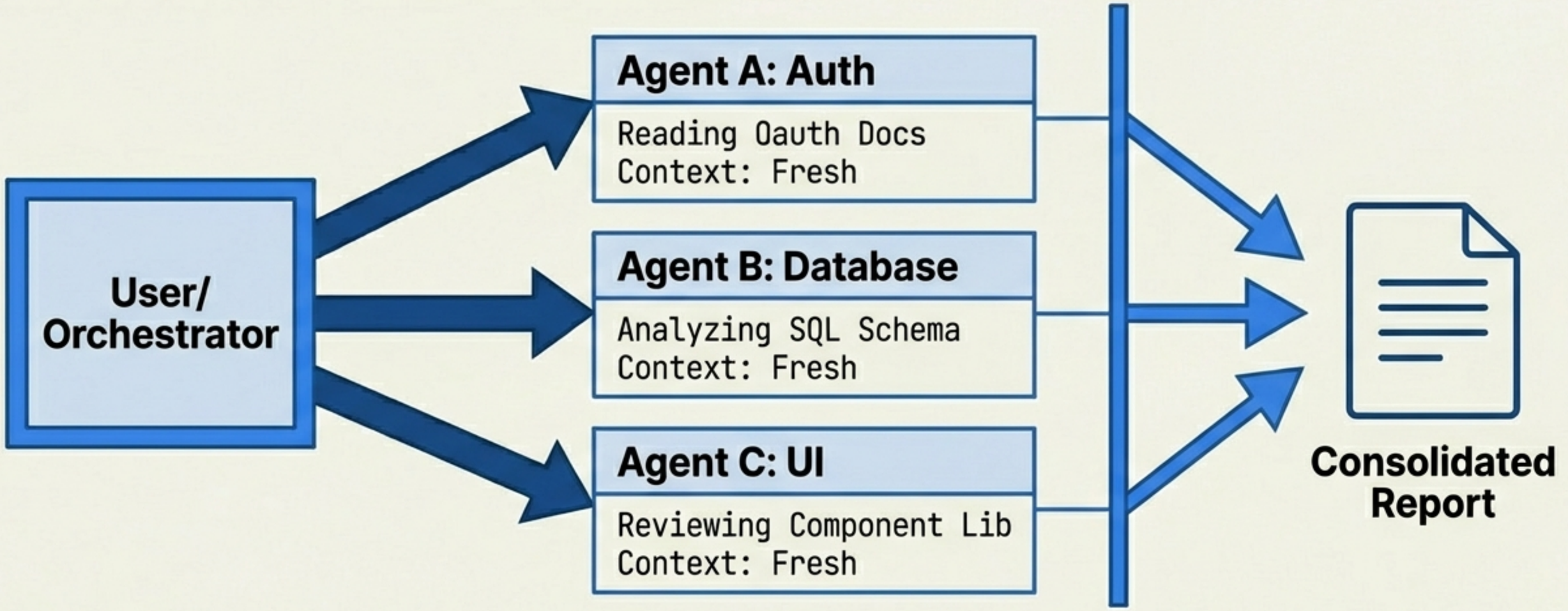
Task Delegation.
Input: Final Spec.
Output: Atomic Commits.

Vibe Coding: Planning & Execution interleaved. Review implies approval fatigue.

SDD Workflow: Planning precedes Execution. Review happens at phase gates.

Phase 1: Parallel Research

Solving Context Pollution via Isolation



Benefit: Hidden conflicts become visible. Agent A's confusion does not pollute Agent B's findings.

`Prompt: "Spin up multiple subagents for your research task."`

Phase 2: The Specification Template

The Source of Truth (spec.md)

spec.md

1. Reference & Current Architecture

What ‘good’ looks like vs. where we are starting.

2. Implementation Plan

The phased approach strategy.

3. Implementation Checklist

Atomic tasks ready for delegation.

4. Constraints & Success Criteria

Explicit boundaries: ‘What NOT to build’.
Measurable Metrics: ‘P95 latency < 100ms’.

Principle: Combine PRD thinking (Why) with SRS precision (How).

Phase 3: Refinement via Interview

The 10x Rule: Catching ambiguities before they become bugs.

1 Ambiguity in Spec = 10 Bugs in Code

Tool: ask_user_question.
Question: Should we migrate existing data or start fresh?

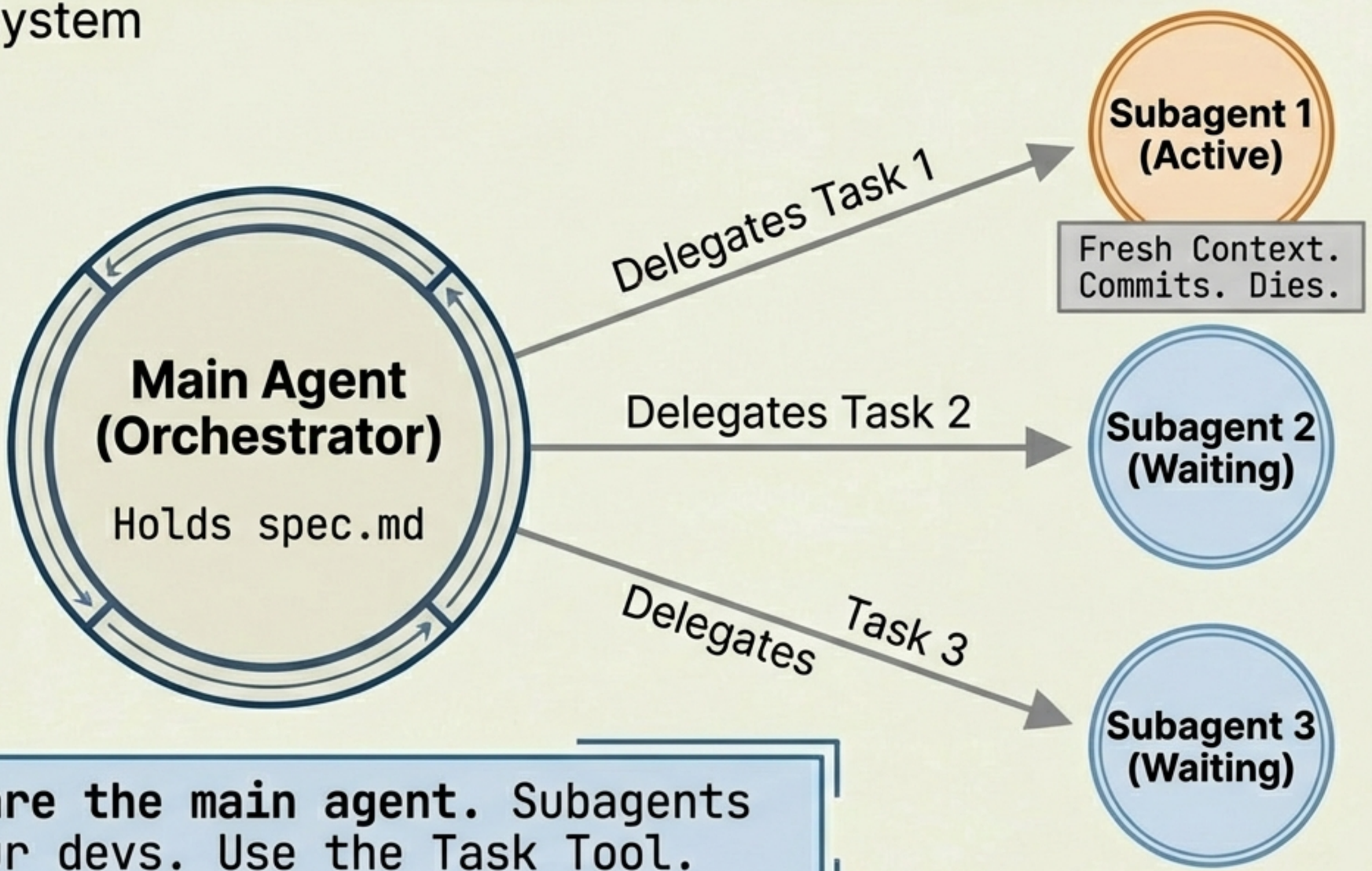
Migrate. Keep last 30 days.

Question: Conflict resolution? Last write wins or user prompt?

Ambiguity Categories
1. Data Decisions
2. Conflict Resolution
3. Failure Recovery

Phase 4: Implementation & Delegation

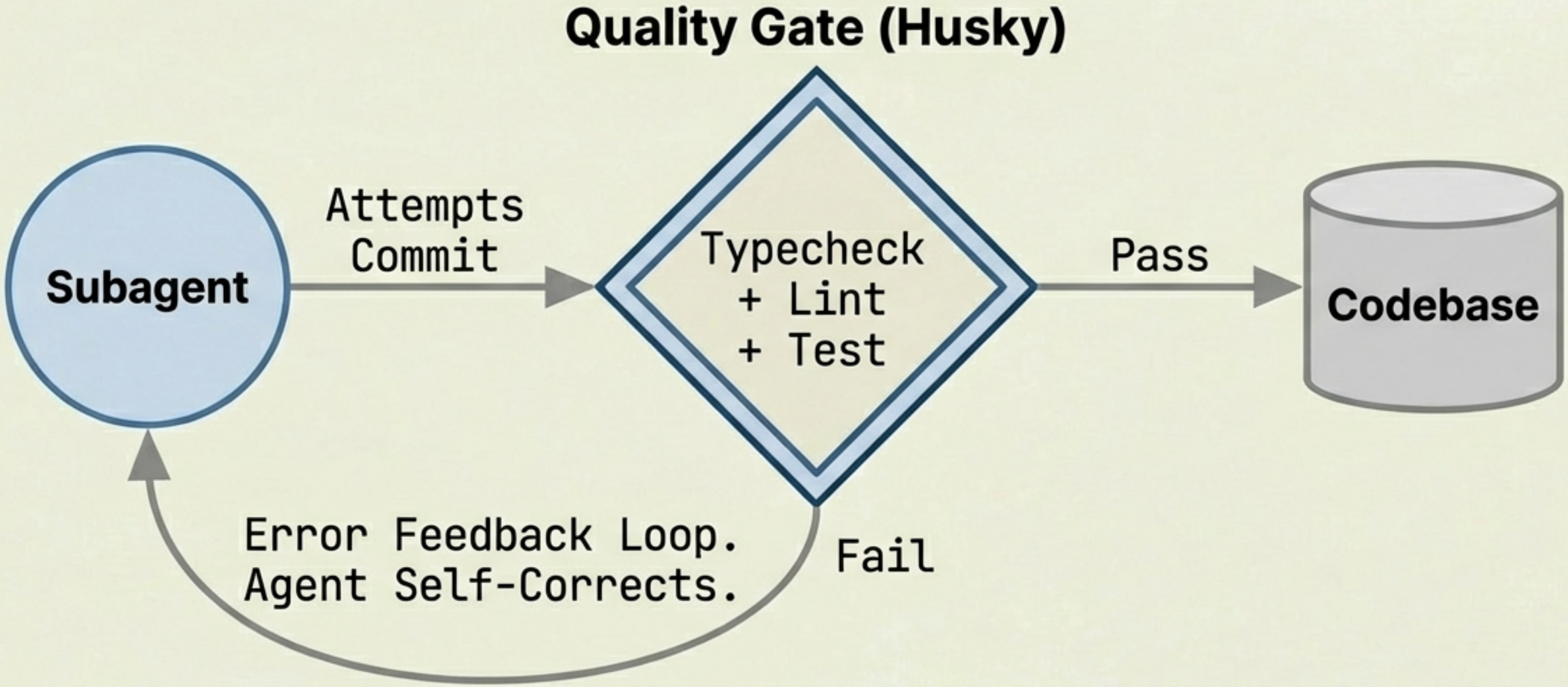
The Task System



> **You are the main agent.** Subagents are your devs. Use the Task Tool.

Backpressure & Quality Gates

Trust but Verify via Pre-Commit Hooks



Result: The human is not the bottleneck for syntax or basic logic checks.

Proof of Concept: Alexop.dev Migration

Migration from SQLite to IndexedDB (15+ Files)

45

Minutes Total

vs 4+ hours manual

14

Atomic Tasks

0 Rollbacks

71%

Context Remaining

vs 0% (Context Failure) in Vibe Coding

5 Research Agents → **1** Refined Spec → **14** Implementation Tasks

Decision Framework

Heuristic: Does the complexity exceed working memory?

USE SDD WHEN...

- Large Refactors (15+ files)
- Unclear Requirements (Research needed)
- Legacy Modernization
- Learning new libraries

SKIP SDD WHEN...

- Single-file bug fixes
- Exploratory prototyping
- Production fires (Incident response)

The Lightweight Spec Pattern

The Middle Ground for Borderline Tasks

CONSTRAINTS

Boundaries to prevent scope creep.

SUCCESS CRITERIA

The definition of Done.

Strategy: Start lightweight.

If it reveals hidden complexity → Expand to full 4-Phase Workflow.

If not → Ship it.

Prompt Patterns Cheat Sheet

RESEARCH

```
> Spin up multiple subagents for your research task. Your goal is to write a report.
```

SPEC-FIRST

```
> Your goal is to write a document. Do not write code yet.
```

REFINE

```
> Use the ask_user_question tool to identify ambiguities before we implement.
```

IMPLEMENT

```
> Use the task tool. Each task by a subagent. Commit after each task. You are the main agent; subagents are your devs.
```

The Future of Engineering

Typopatrist in Helvetica Now Display
General Agents Build Custom Agents

SDD isn’t about slowing down.
It’s about moving fast without breaking things.

Start with one spec on your next complex feature.

Spec-Driven Development / Claude Code