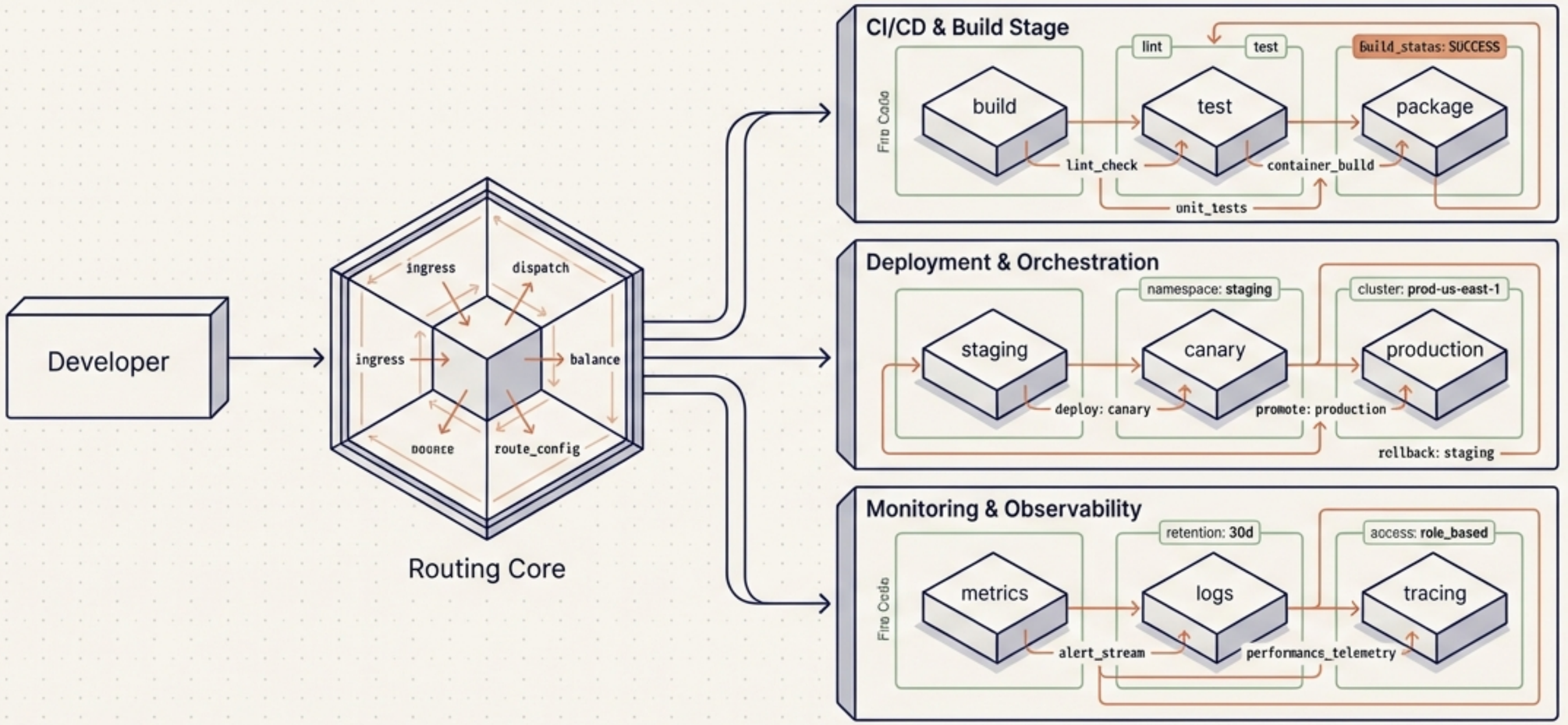
Claude Code as Engineering Infrastructure

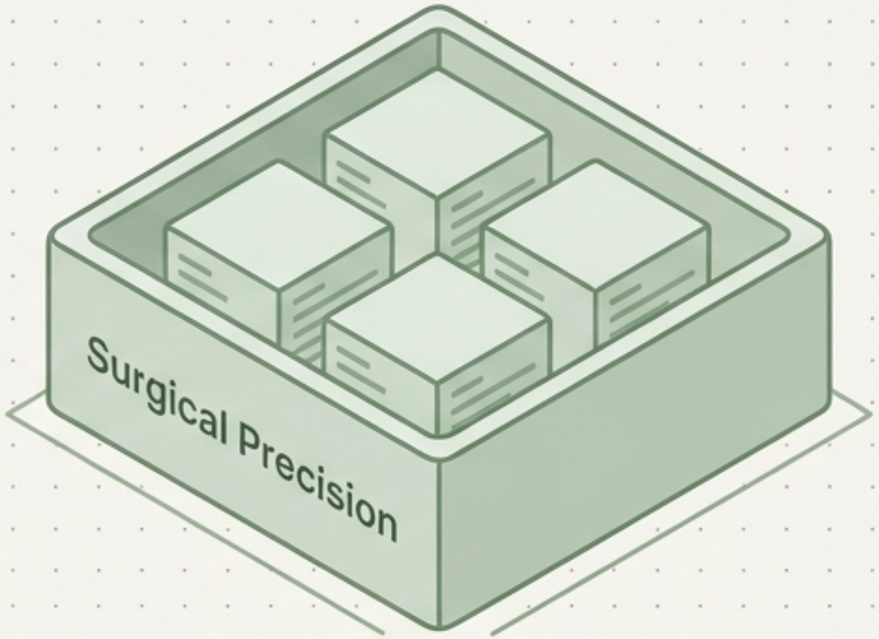
Advanced Configuration, CI/CD, and Team Workflows



A visual reference for the Claude Certified Architect: Foundations exam.

The Core Problem: Scope & Attention Dilution

React linting rules loaded into a Python backend file. **Irrele**vant instructions consume tokens and dilute critical attention.



The exact right rules, applied to the exact right file, in a clean memory space.

Claude Code scales to teams only when you strictly manage the radius of impact.

The CLAUDE.md Configuration Hierarchy

User (`~/.claude/CLAUDE.md`)

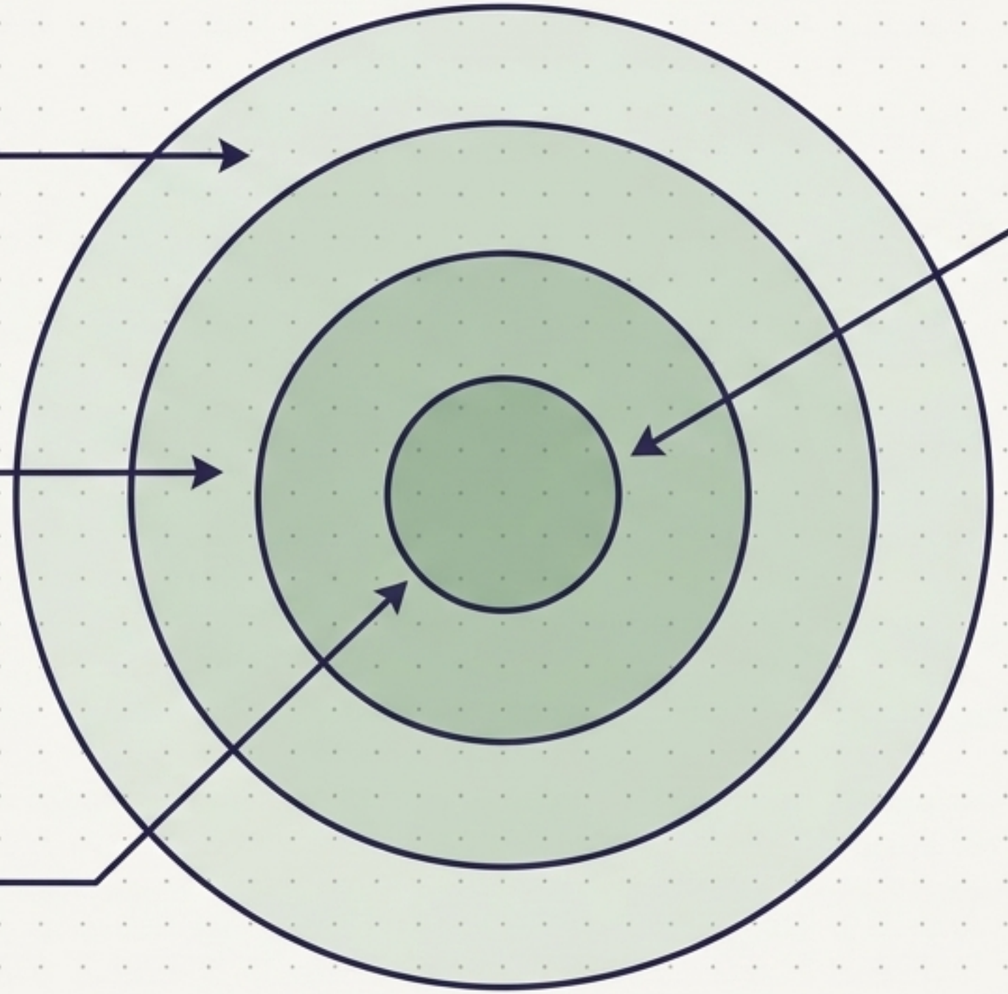
Personal preferences.
NOT shared via Git.
Loads at session start.

Project (`./CLAUDE.md`)

Team standards.
Shared via Git.
Loads at session start.

Directory
(`packages/api/CLAUDE.md`)

Package conventions.
Shared via Git.
Loads ON DEMAND (only when files in this folder are read).



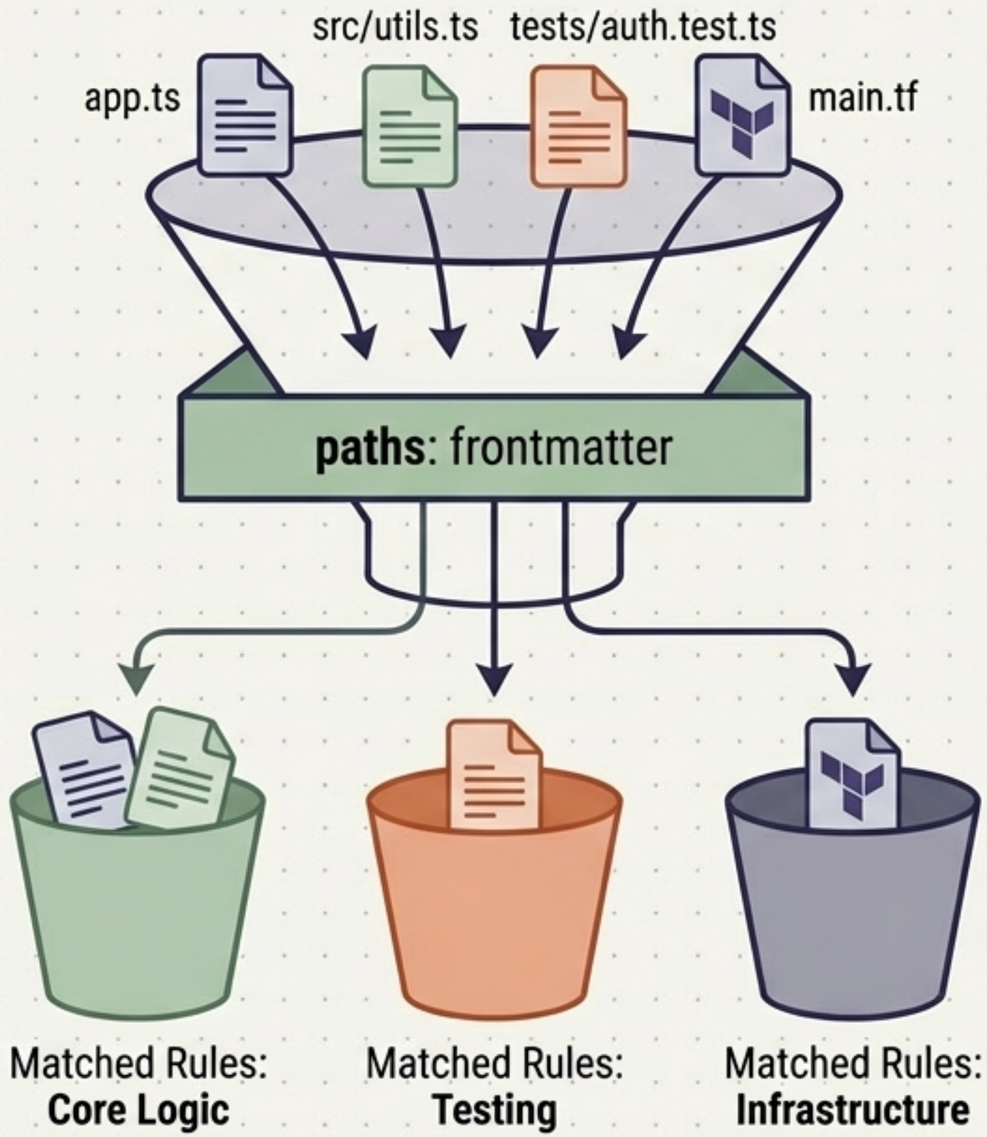
Outer (`./CLAUDE.md`)

Personal preferences.
NOT shared via Git.
Loads at session start.

```
> /memory
```

Rules not loading? Run `/memory` to diagnose exactly which hierarchy levels are currently active in your session.

Path-Specific Rules: Filtering by File Type

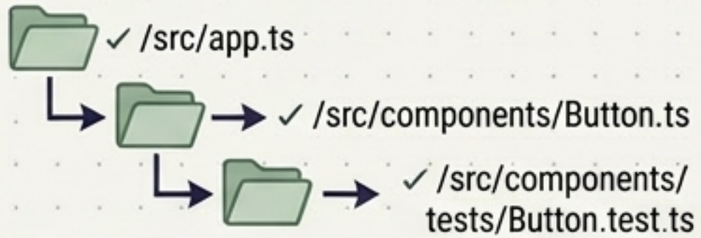


Comparison Panel: The Glob Gotcha

`*.ts`

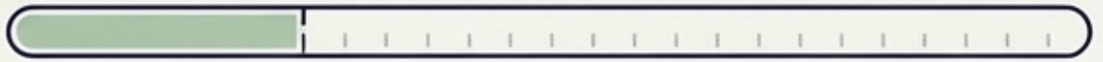
Matches root directory files only.

`**/*.ts`



Matches recursive directories. Critical for cross-cutting conventions like tests.

Token Efficiency



Path rules load only when a matching file is read, preserving context bandwidth for the actual task.

The Configuration Placement Matrix

I want to...	Put it in...	Shared via Git?	Load Trigger
Set personal preferences	`~/.claude/CLAUDE.md`	No	Session Start
Share team standards	`./CLAUDE.md`	Yes	Session Start
Scope rules to one package	`packages/name/CLAUDE.md`	Yes	On Demand
Apply rules to a file type	`.claude/rules/*.md`	Yes	On Demand
Create an on-demand workflow	`.claude/skills/`	Yes	Explicit Command
Block a tool or file path entirely	`settings.json`	Local	Hard Enforcement

Custom Skills: Executable Team Knowledge

Code Review Checklist

Forgotten, ignored, detached from the workflow.

Version-controlled Executable Skill

```
# .claude/commands/review.md
---
command: review
description: "Perform a comprehensive code review"
inputs:
  - file_path:
      description: "The file to review"
      required: true
---

# ... (executable logic for code review)
def run_review(file_path):
    # Analyze code quality, check guidelines
    result = analyze_code(file_path)
    return result
# ...
```

Version-controlled, invoked via /review, executes directly where the code lives.

Collision Priority: Enterprise > Personal (~/.claude/skills/) > Project (.claude/skills/)

Higher priority overwrites lower when command names collide.

Frontmatter Power-Ups: Security & Parameters

Security

```
---
allowed-tools: Read, Grep,
Glob, Bash(npm test*)
---
```

Creates read-only skills. Restricts default permissions; does not grant new ones.

Parameters

```
---
argument-hint: [issue-number]
---
```

```
❯ /fix-issue
          [issue-number]
```

Maps to $0, $1 variables within the script.

Manual Trigger

```
---
disable-model-invocation:
true
---
```

Requires manual human trigger. Essential for dangerous side-effect skills like /deploy.

The Fork & Explore Mechanism

Without Fork

Main Session Window

95% Full

Less room to think for the next prompt.

With context: fork

Explore Sub-Agent

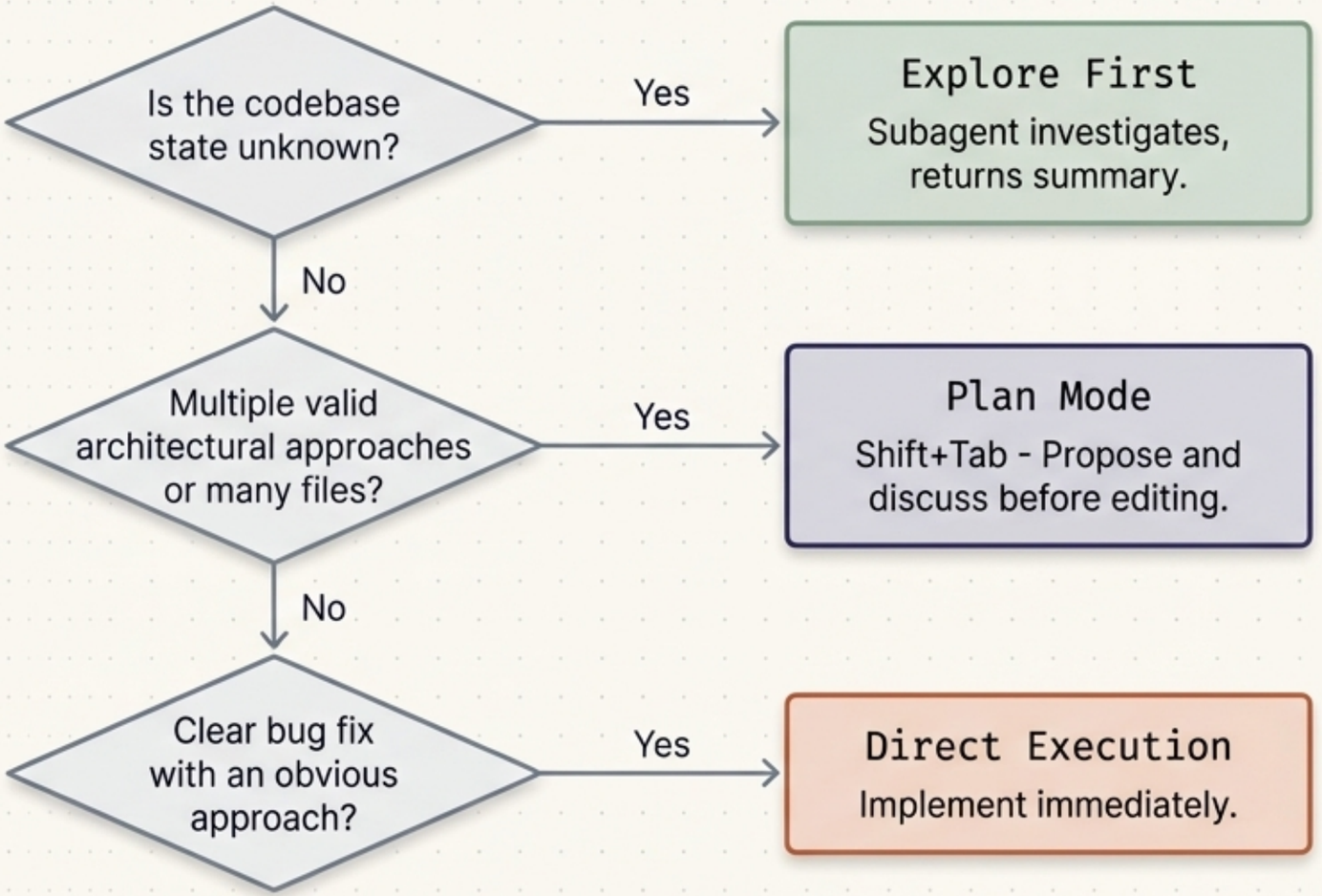
2-Paragraph Summary

Main Session Window

5% Full

Use for codebase audits, brainstorming, and scanning many files. Do not use for interactive back-and-forth.

Execution Modes: Measure Twice, Cut Once

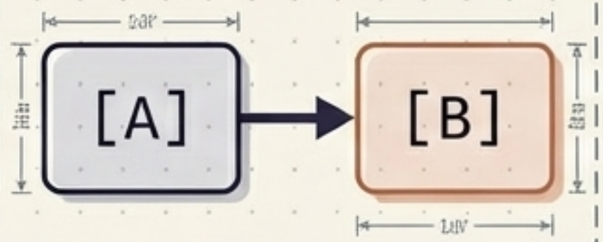


Plan mode is not overhead; it is rework prevention. Direct execution buries architectural decisions silently in the diff.

The Iterative Refinement Framework

Pillar 1

I/O Examples

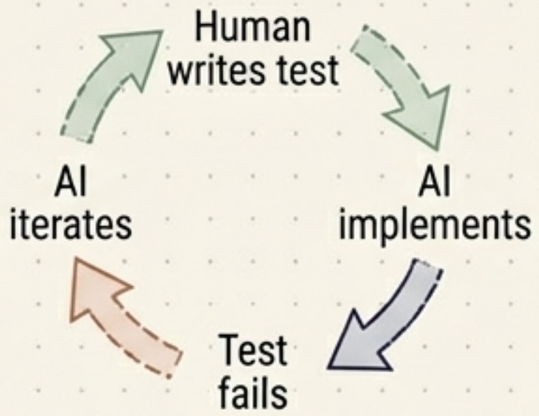


For Data Transformations.

Resolves vague requests like "normalize dates" with concrete examples.

Pillar 2

Test-Driven Iteration



For Complex Logic.

You define correctness via assertions; Claude figures out the implementation.

Pillar 3

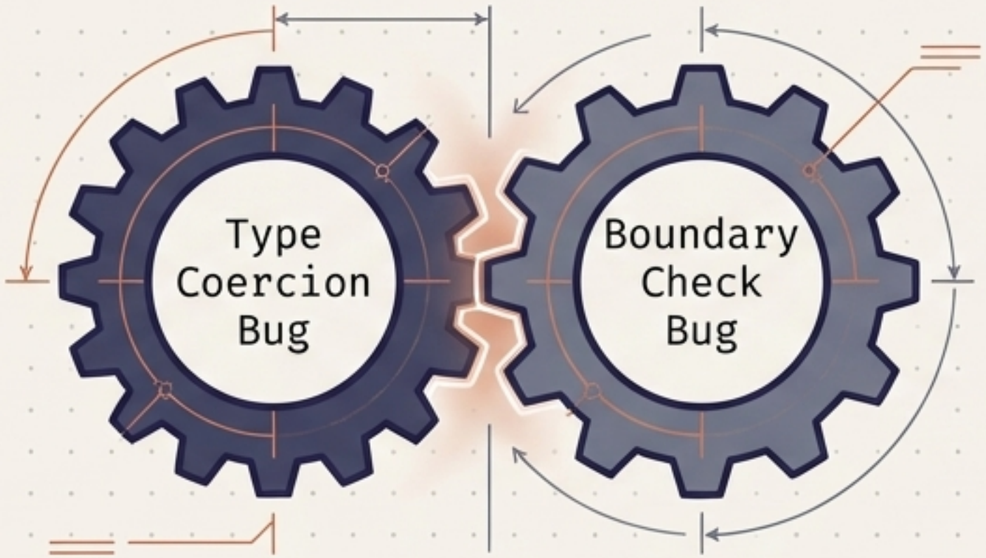
The Interview Pattern

For Unfamiliar Domains.

Ask Claude to interview you to surface hidden requirements and edge cases.

Feedback Routing: Single vs. Sequential Fixes

Single Message



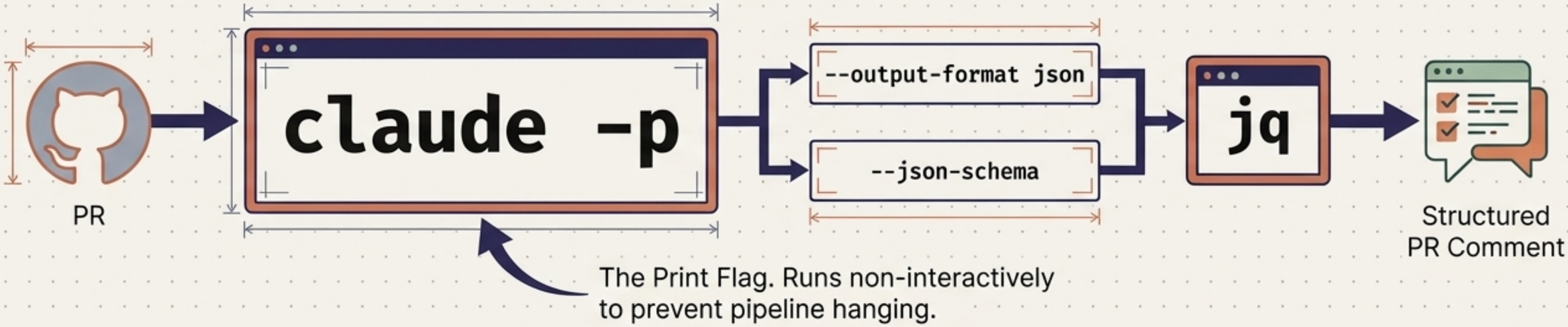
If fixes interact. Report together so Claude sees the full context and avoids breaking one fix with another.

Sequential

If fixes are independent. Report one at a time to isolate verification and keep iterations manageable.

Rule of Thumb: If you are unsure whether the bugs interact, default to a Single Message to prevent regressions.

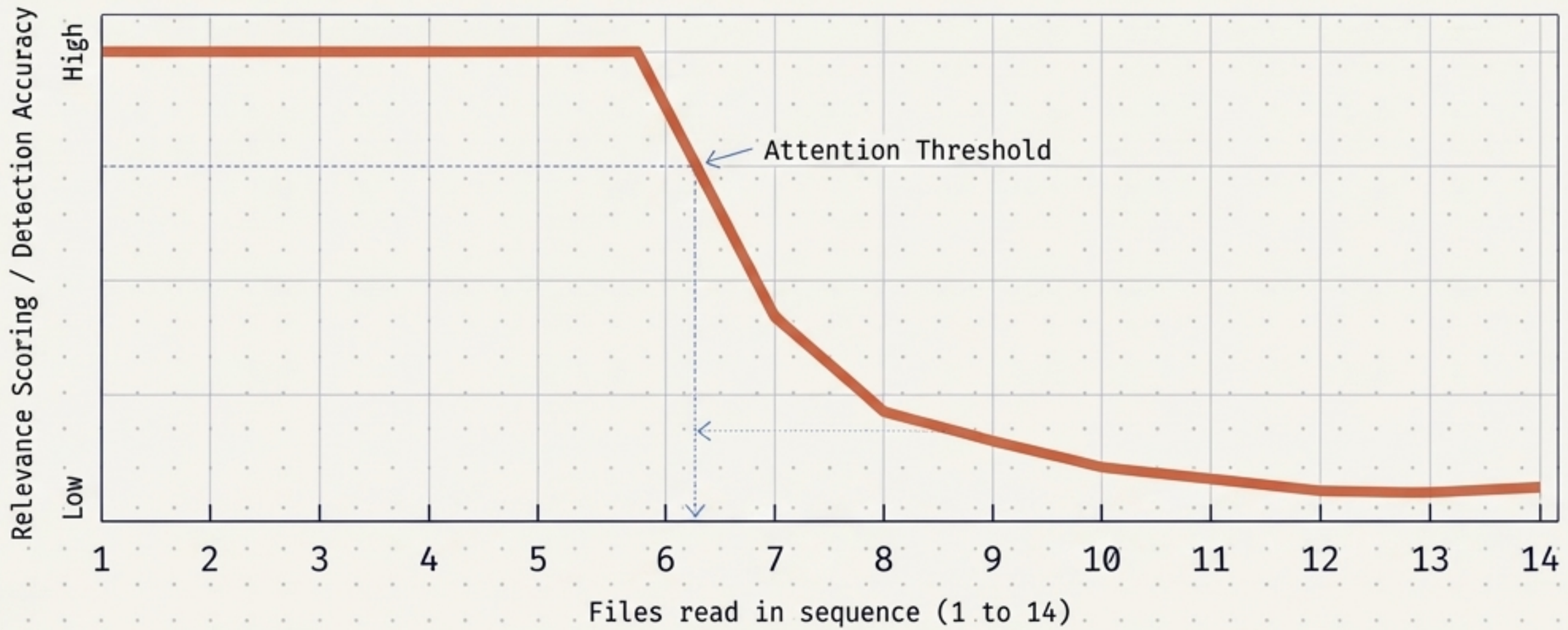
Claude in the CI/CD Pipeline



API Choice Matrix

Real-Time API	Message Batches API
Pre-merge checks & PR review comments. Use when a human is waiting.	Nightly technical debt reports. 50% cost savings. Use when there is no urgency.

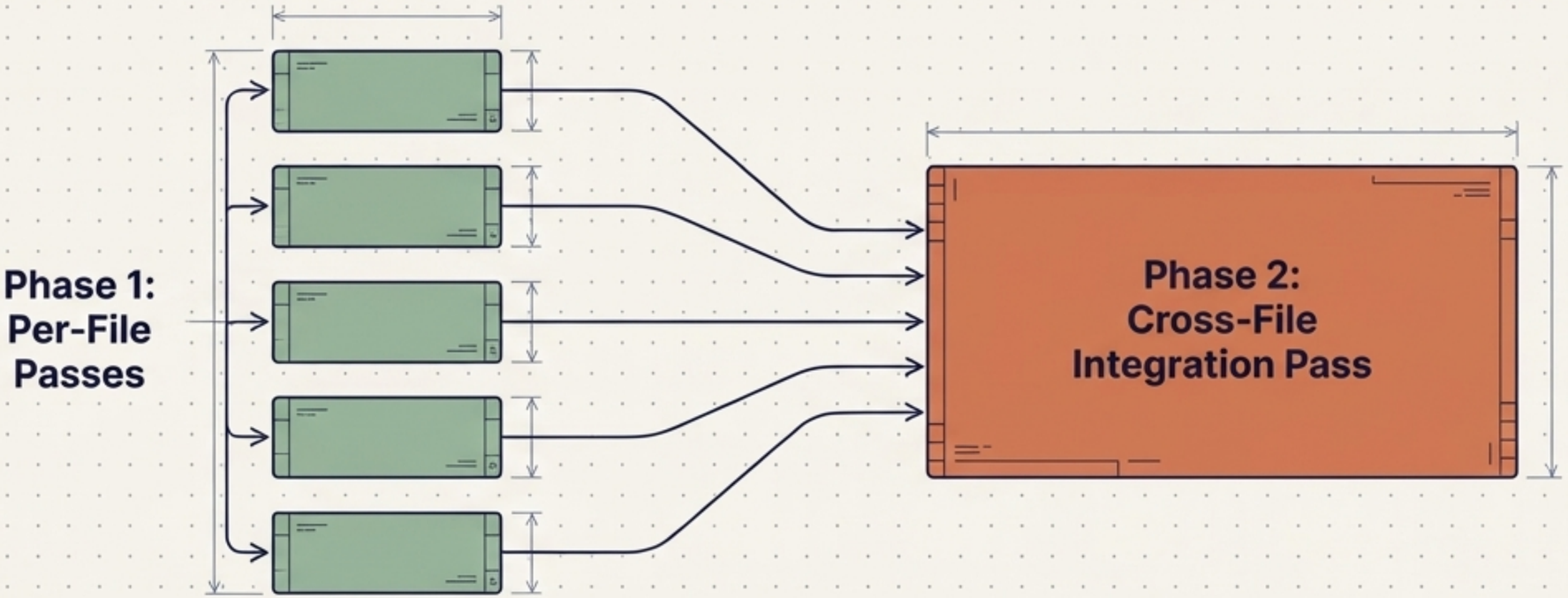
The Attention Dilution Problem



When reviewing a 14-file PR in a single prompt, a larger context window does NOT fix the problem. Claude's attention budget is spent on the first files, causing shallow analysis and missed bugs in later files.

The structural fix requires isolating the reasoning context...

Multi-Pass Review Architecture



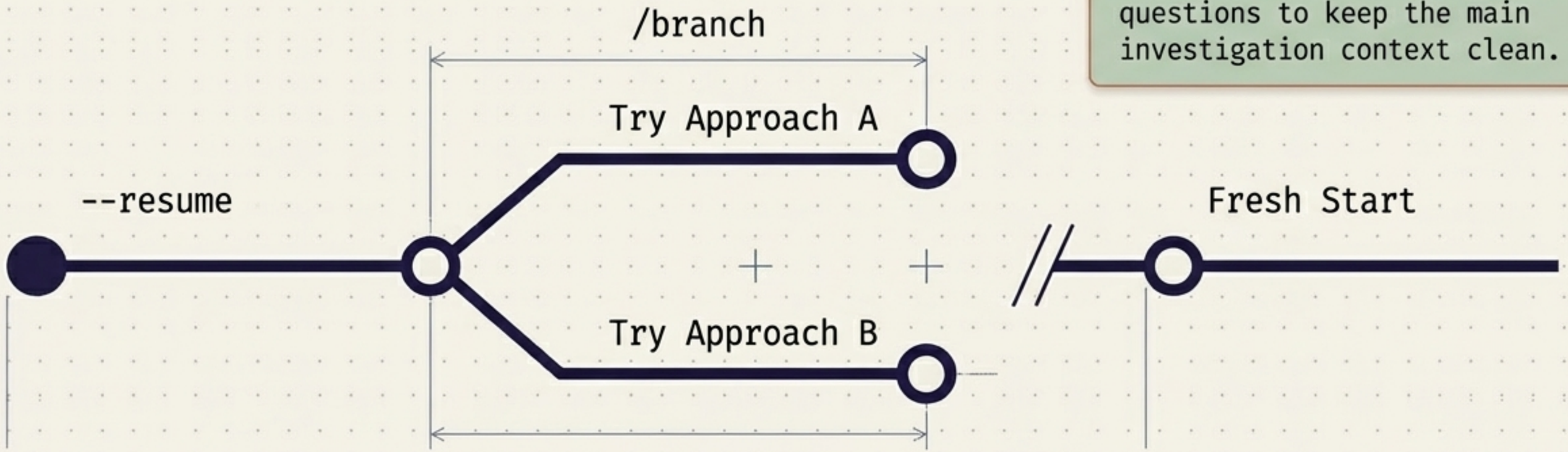
One file at a time.
Catches local bugs (Null dereferences, SQL injection)

Reviews inter-file interactions.
Catches broken contracts and data flow errors across boundaries.

Best Practice: Use specific reporting criteria. ‘Report SQL injections’ works better than ‘be conservative’, which causes Claude to self-filter subtle bugs.

Session State Management

Pro-tip: Use /btw for side questions to keep the main investigation context clean.



Use when files are unchanged and context is still valid.

Parallel exploration from the same baseline context without file conflicts.

Use when files have changed significantly. Start fresh with a written summary.