Slop is a Skill Issue

The Engineering around AI Agents

$ whoami

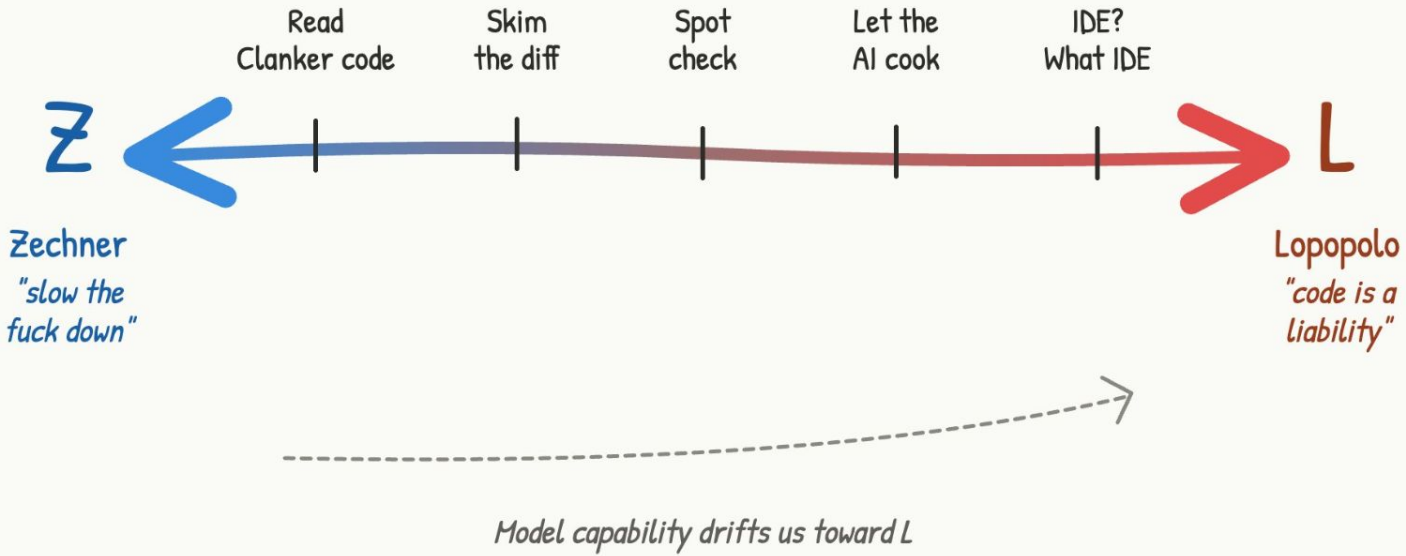
- AI Engineer at Mercury Technologies
  - Background agents for the eng org
- Previously @ Nectry
  - w/ Adam Chipala
  - Nectry Product Agent
- Career:
  - Haskell Enthusiast
  - Compiler Engineer for AI chips
  - AI Agent Engineer

I’m not afraid of AI taking jobs

I’m afraid of the **slop**

The Ƶ/L Continuum

Do AI engineers need to read code anymore? Do you?



AI Engineer World's Fair

Stanford University

Does AI Actually Boost Developer Productivity?

(STANFORD / 100K DEVS STUDY)

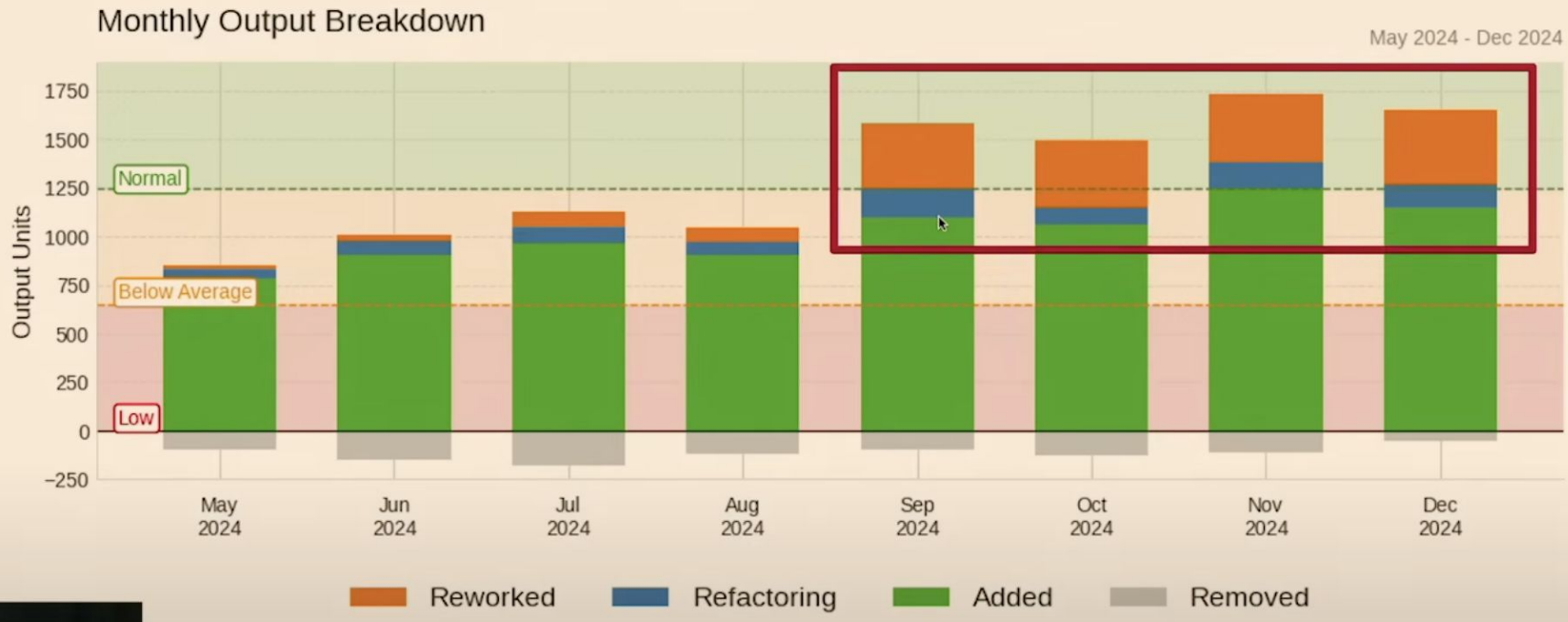
Yegor's Talk:

100k Devs across all company sizes

Most AI in Software Eng leads to rework

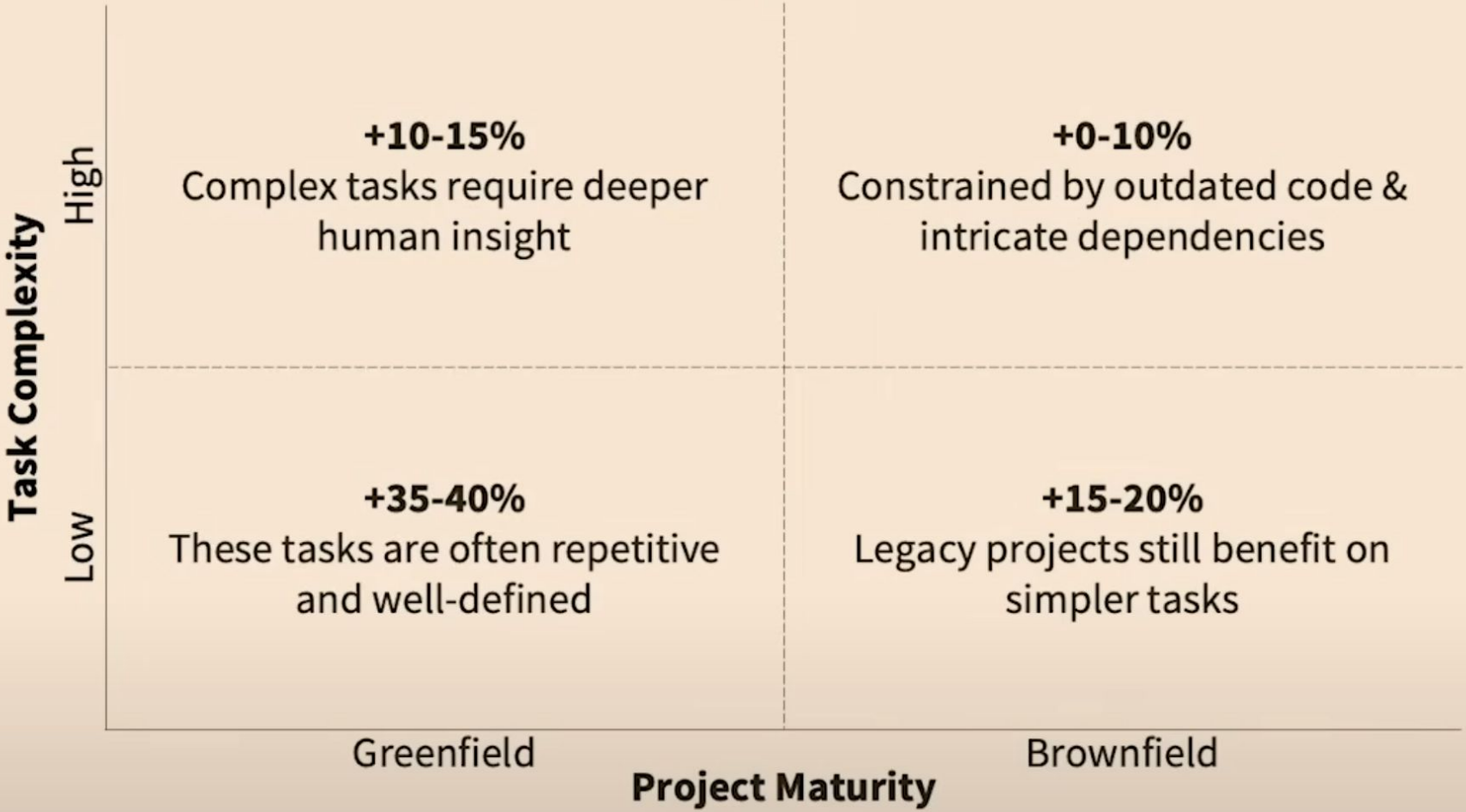
Sucks for Brownfield or Complex Tasks

Using AI in software engineering increases rework



Software Engineering Productivity Increases from AI Use

Orientative Guidelines



General Sentiment

“Too Much Slop”

“Tech Debt Factory”

“Doesn’t work in big repos”

“Doesn’t work for complex systems”

“Maybe someday when
the models get better”

I’ve spent the last
6 months building
AI Agents

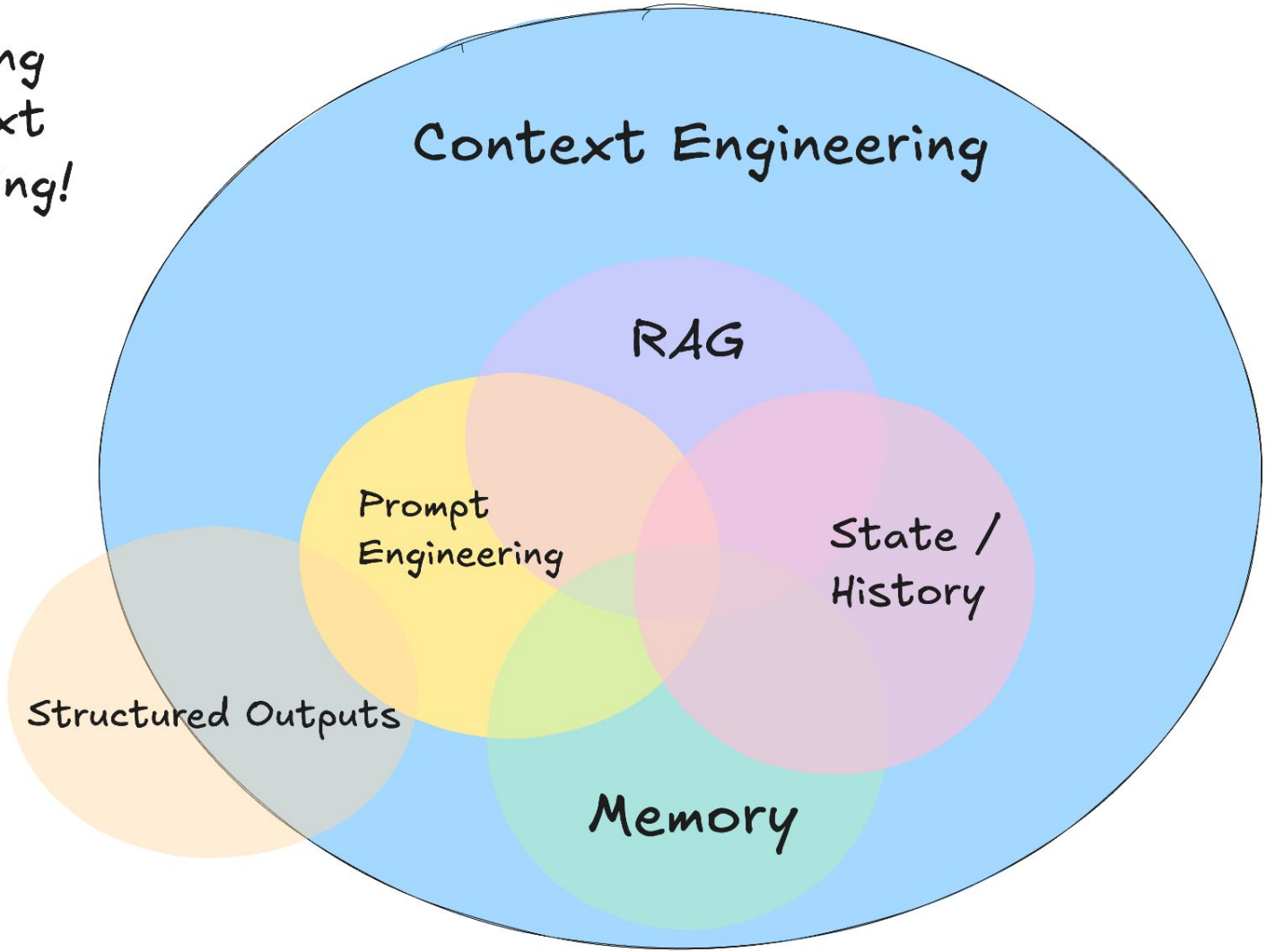
Context Engineering

Controlling the information
that is injected into the
context window

Context Engineering

getting the most
out of today's models

Everything is Context Engineering!



What is an **Agent**?

An agent is a **person**

An agent is a **microservice**

An agent is a **chatbot**

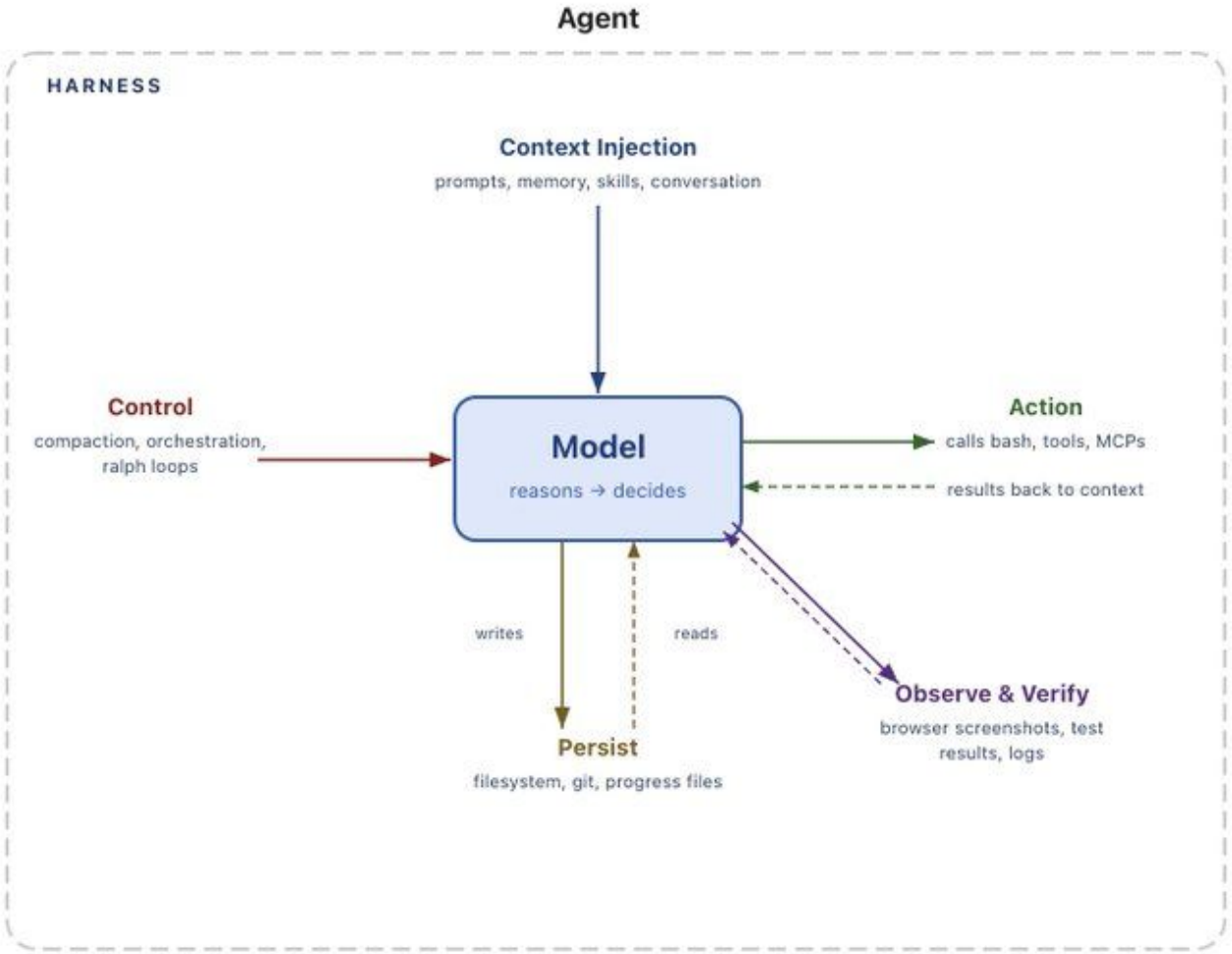
An agent is a **workflow**

An agent is **tools in a loop**

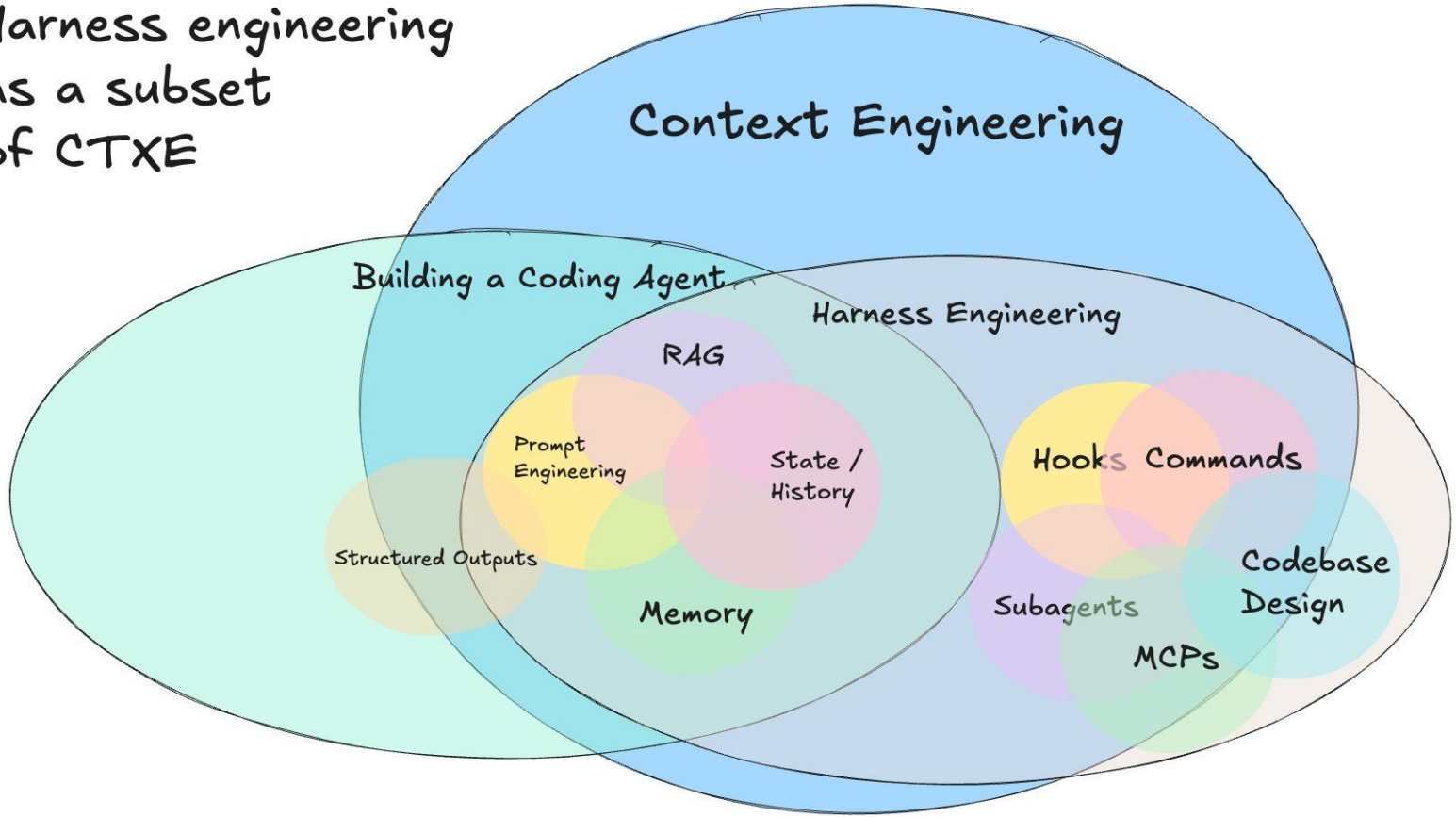
Agent = Model + Harness

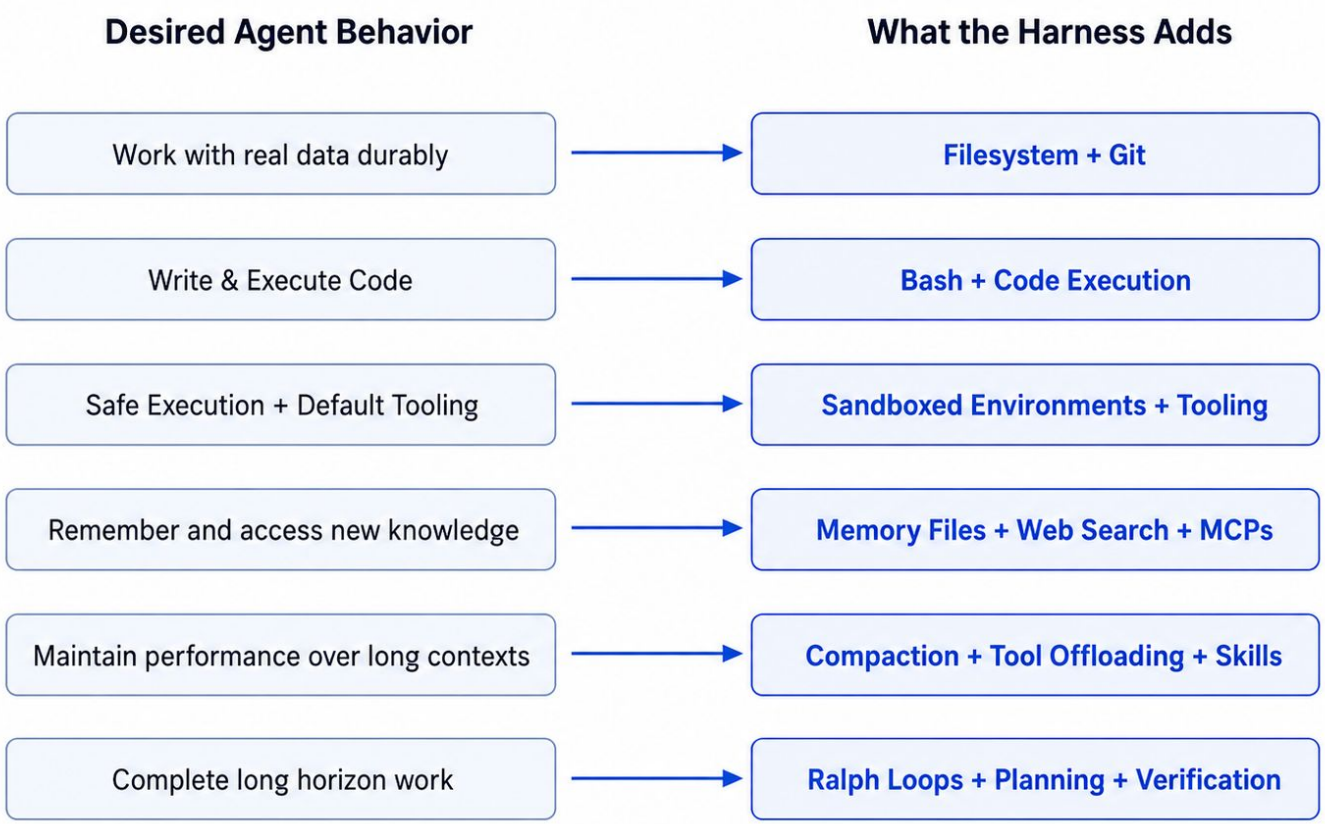
What is a harness?

“Everything that is not the model”


Agent
HARNESS
Context Injection
prompts, memory, skills, conversation
Control
compaction, orchestration, ralph loops
Model
reasons → decides
Action
calls bash, tools, MCPs
results back to context
writes
reads
Observe & Verify
browser screenshots, test results, logs
Persist
filesystem, git, progress files

Harness engineering as a subset of CTXE





Each harness feature derives from a behavior the model can't deliver on its own

Goals

Agents work well in our codebases

Solve Complex Problems

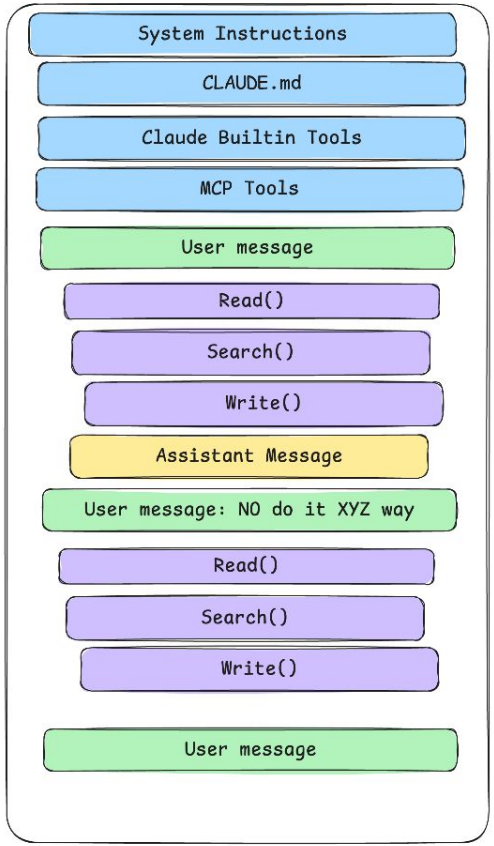
~~No~~ **Minimal Slop**

Mental Alignment + Understanding

(Spend as ~~many~~ few tokens as possible)

the **most** naive way
to work with coding agents

the naive way - work until you run out of context

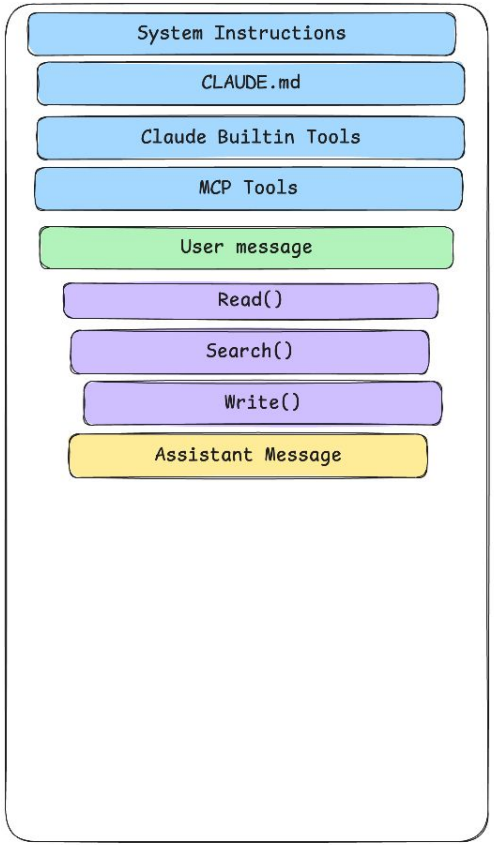


Resteering in context:

"NO use XYZ Approach"

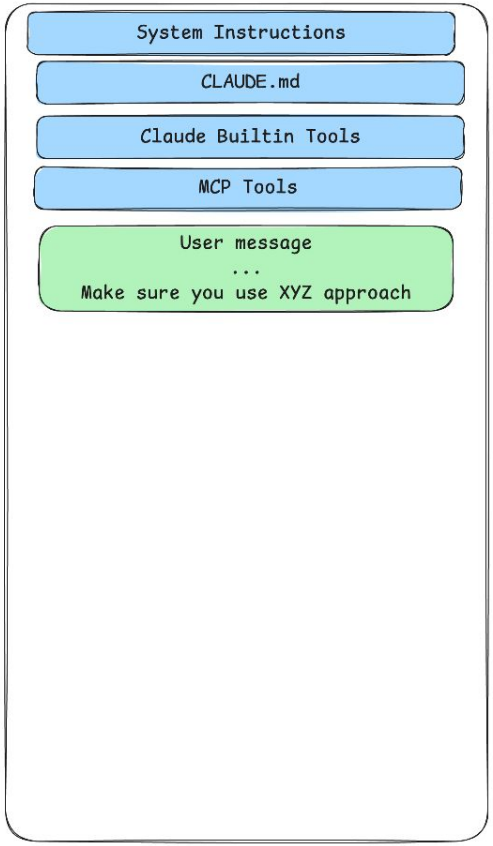
the *slightly* smarter way…

Slightly smarter - start over vs resteer



Fresh context:

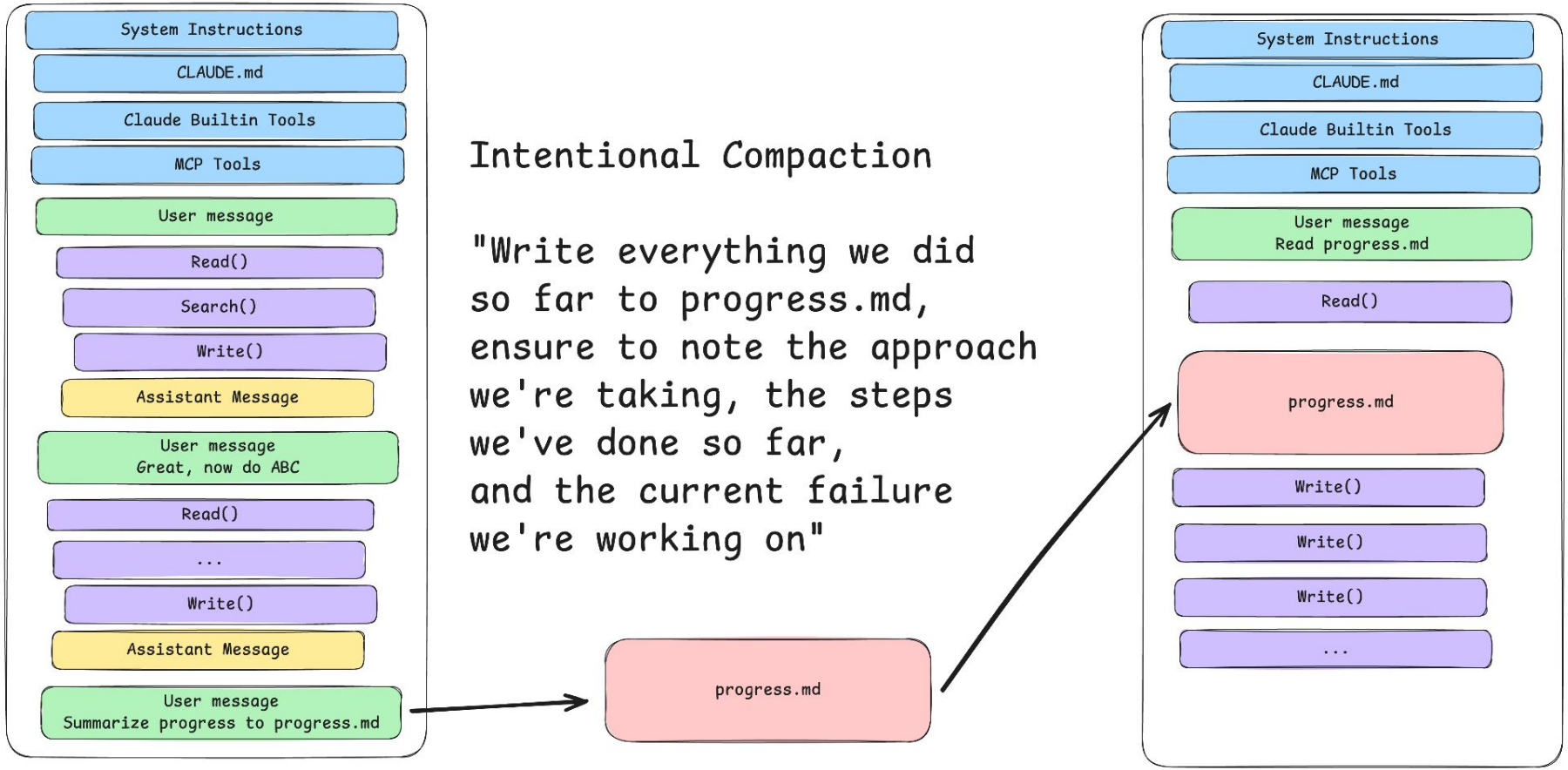
"Make sure you use XYZ approach"



Even smarter:

Intentional **Compaction**

Slightly smarter - intentional compaction



What are we compacting?

File search output

Code flow understanding

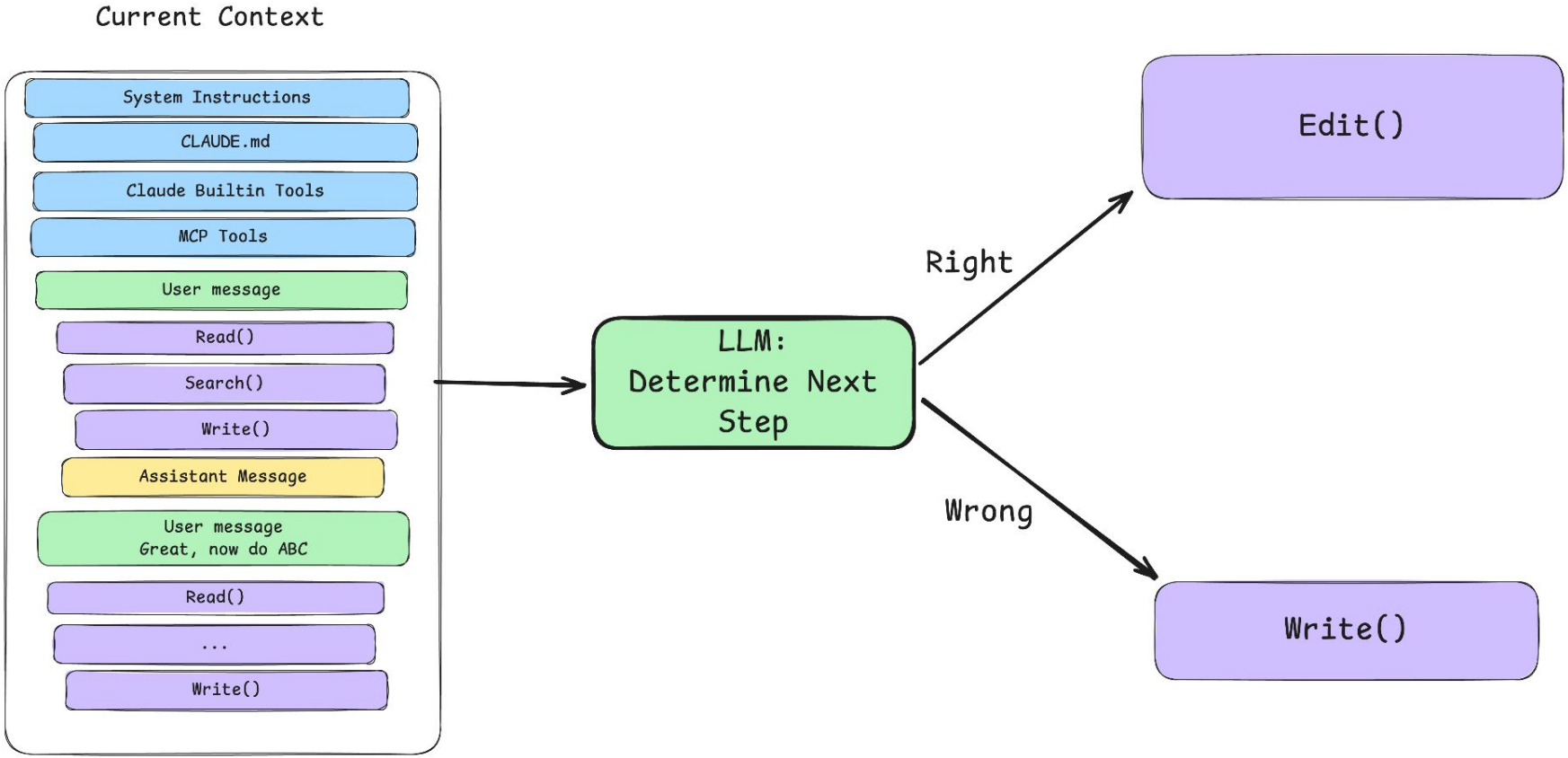
Edits to files

Test/Build output

JSON Tool Responses

Why does context matter?

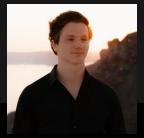
Context is everything



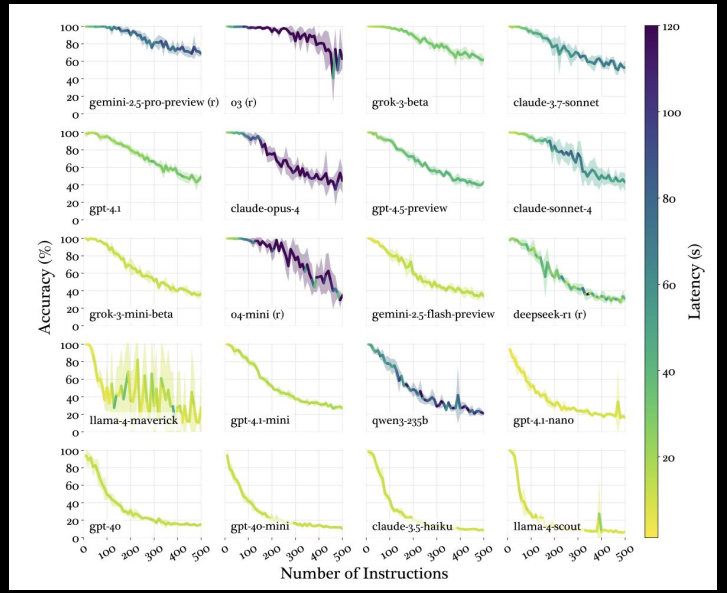
Writing a good CLAUDE.md

Kyle · November 25, 2025 · < 10 min read

#agents #claudecode #best-practices



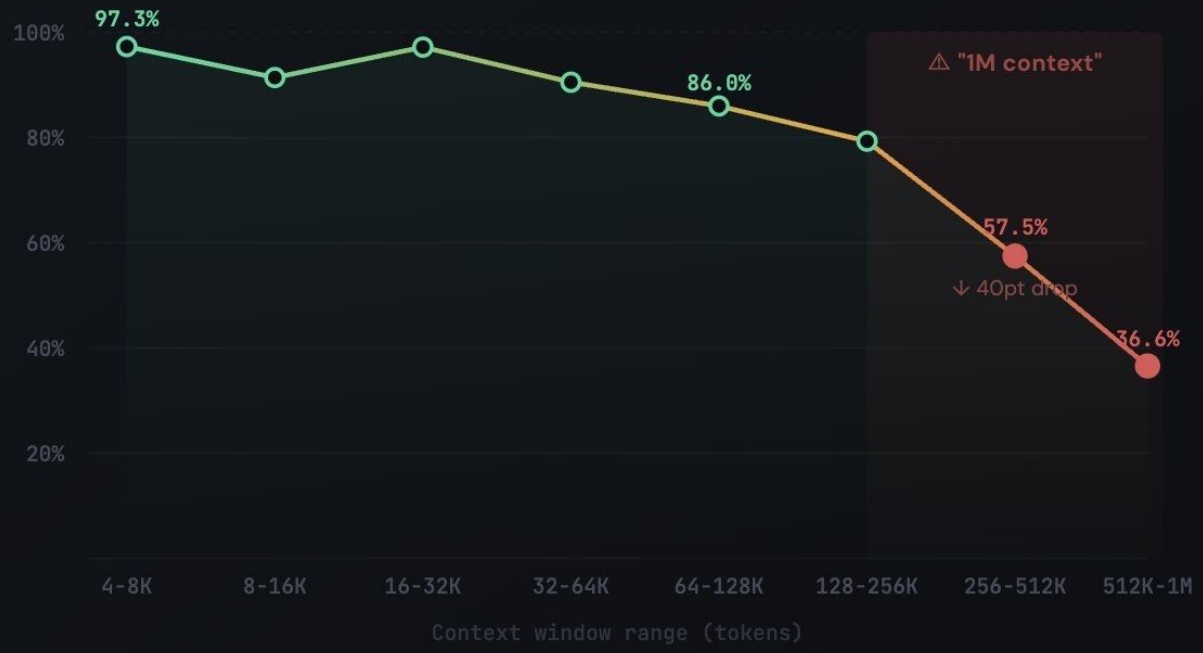
1. **Frontier thinking LLMs can follow ~ 150-200 instructions with reasonable consistency.** Smaller models can attend to fewer instructions than larger models, and non-thinking models can attend to fewer instructions than thinking models.



https://arxiv.org/pdf/2507.11538

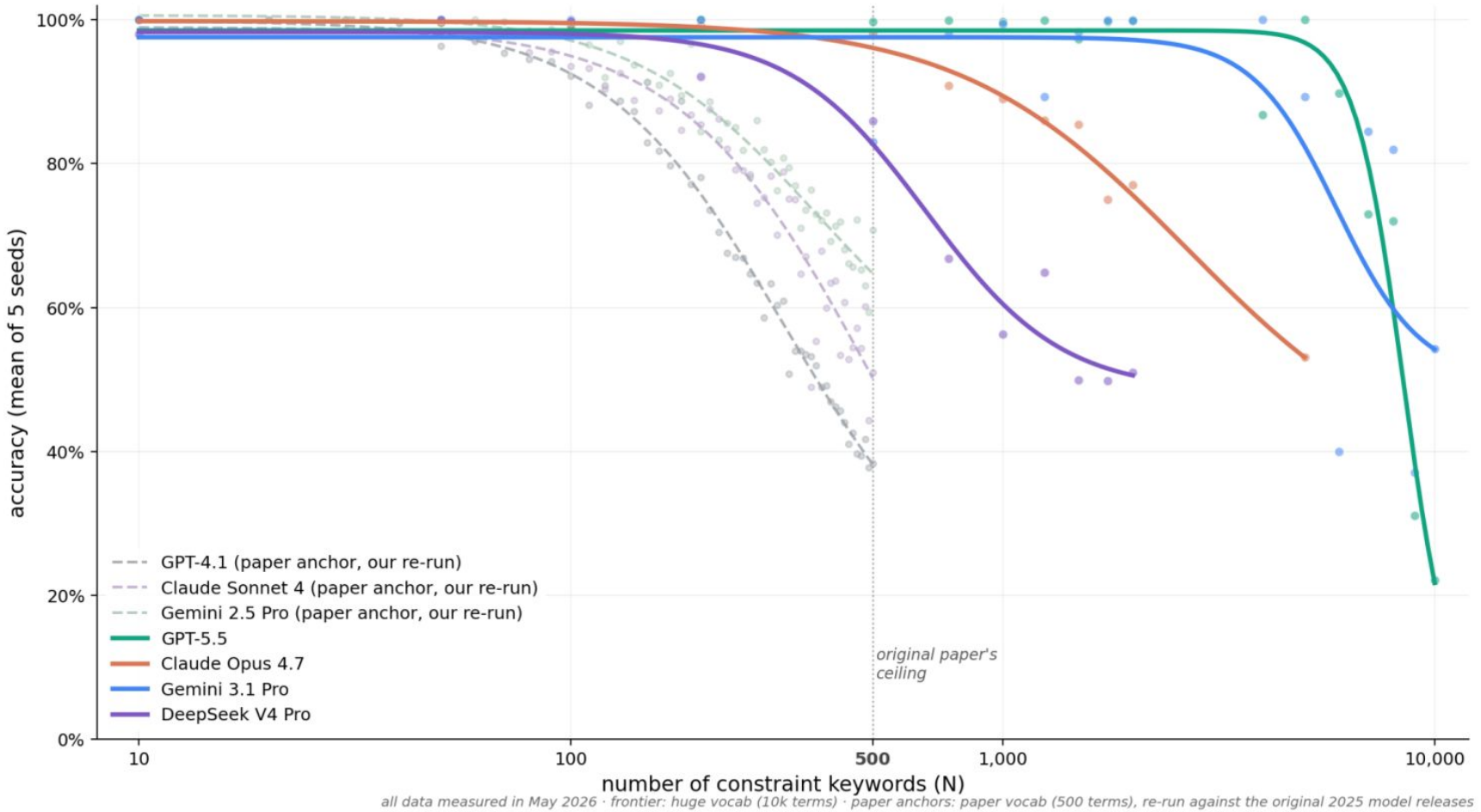
GPT-5.4: 1M Context Reality Check

Needle-in-a-haystack accuracy (MRCR v2, 8-needle) from OpenAI's own evals



GPT-5.4

Source: OpenAI GPT-5.4 eval table · MRCR v2 8-needle · March 5, 2026



https://www.linkedin.com/pulse/models-got-order-magnitude-better-following-one-year-laurie-voss-9fymc/

Optimize Context Window For:

Correctness

Completeness

Size

Trajectory

Context Problems Ranked

Incorrect information

Missing information

Too much information

You have an
instruction budget

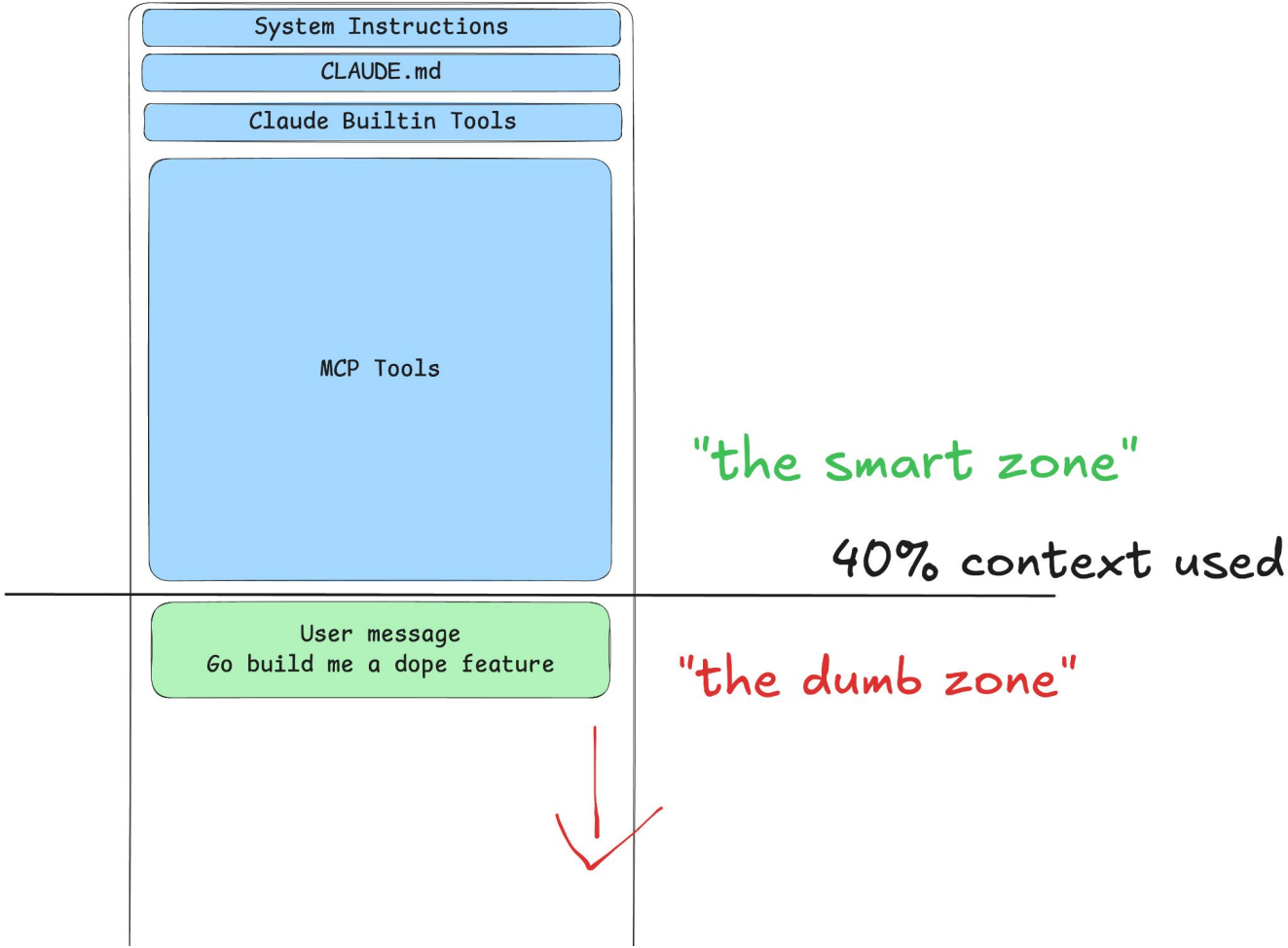
The name of the game is that you only have **approximately 170k** of context window to work with.

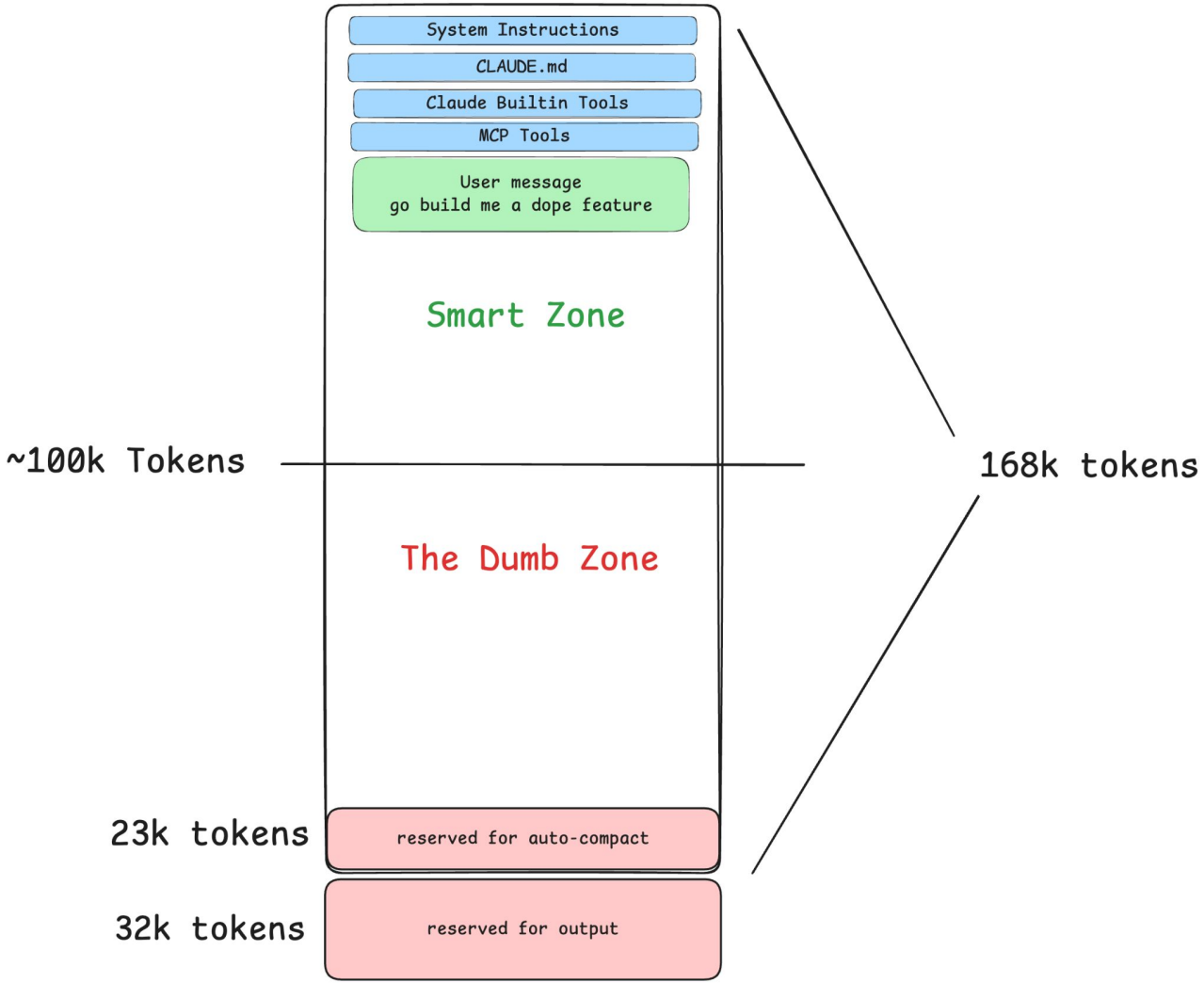
So it's essential to use as little of it as possible.

The **more you use** the context window, the **worse the outcomes** you'll get.

https://ghuntley.com/ralph/

“the dumb zone”


System Instructions
CLAUDE.md
Claude Builtin Tools
MCP Tools
User message
Go build me a dope feature
"the smart zone"
40% context used
"the dumb zone"


System Instructions
CLAUDE.md
Claude Builtin Tools
MCP Tools
User message
go build me a dope feature
Smart Zone
~100k Tokens
168k tokens
The Dumb Zone
23k tokens
reserved for auto-compact
32k tokens
reserved for output

~~40%~~ **50% is a good guideline**

Depends on task complexity

How to avoid the dumb zone

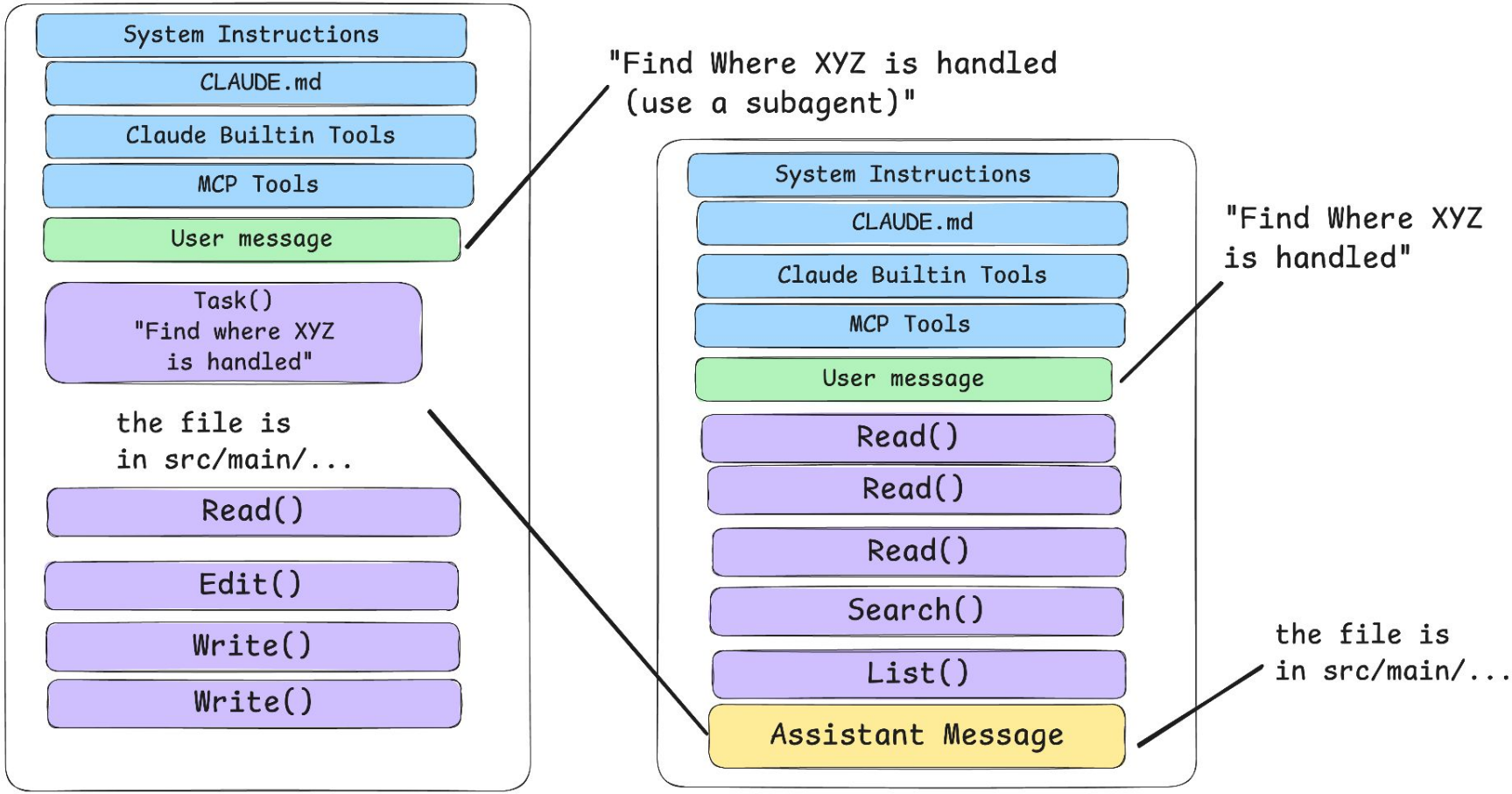
i.e. how to “intentionally compact”

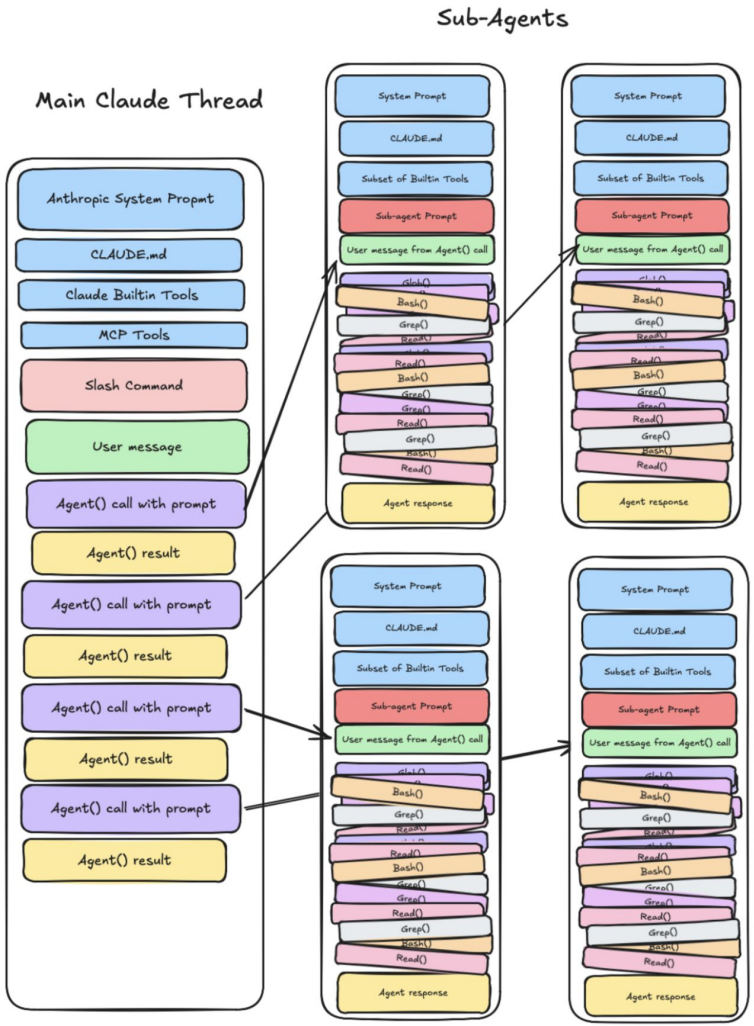
Inline compaction:

Sub-agents

Sub-agents are not for “role play”,
they are for **context management**

Managing Context with Sub Agents





Ideal sub-agent response

Component Usage Flow

Modal Path (Working)

1. `ToolResultModal.tsx:136` calls `getToolIcon(toolCall?.toolName)`
2. `getToolIcon()` returns `<Terminal />` for Bash
3. Icon displays correctly with terminal symbol

Message Stream Path (Broken)

1. `ConversationContent.tsx:103` calls `eventToDisplayObject(event)`
2. Default `<Wrench />` assigned at line 76
3. No Bash-specific override in lines 189-228
4. Generic wrench icon displays

Code References

- `humanlayer-wui/src/components/internal/SessionDetail/eventToDisplayObject.tsx:76` - Default wrench assignment
- `humanlayer-wui/src/components/internal/SessionDetail/eventToDisplayObject.tsx:189-228` - Tool-specific overrides (missing Bash)
- `humanlayer-wui/src/components/internal/SessionDetail/eventToDisplayObject.tsx:657` - Correct Bash→Terminal mapping
- `humanlayer-wui/src/components/internal/SessionDetail/components/ToolResultModal.tsx:136` - Modal icon rendering
- `humanlayer-wui/src/components/internal/SessionDetail/ConversationContent.tsx:103` - Message stream rendering

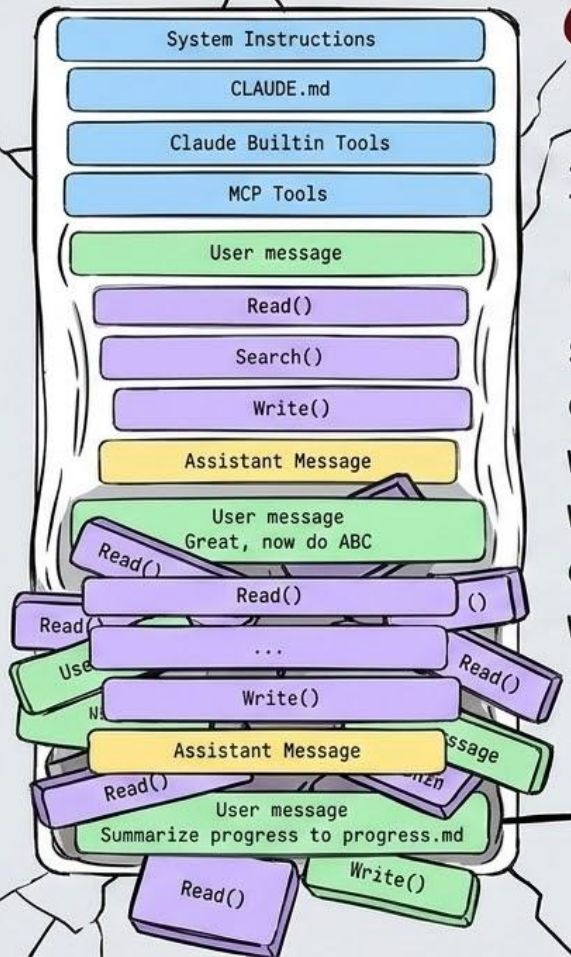
Using **sub-agents** is one of the most effective ways to manage context

in a single session…

Inevitably, you will
run out of space
in the **context window…**

~~Slightly smarter - intentional compaction~~

COMPACTION ATTEMPT: CATASTROPHIC FAILURE



OVERFLOW OVERFLOW OVERFLOW

Intentional Compaction...

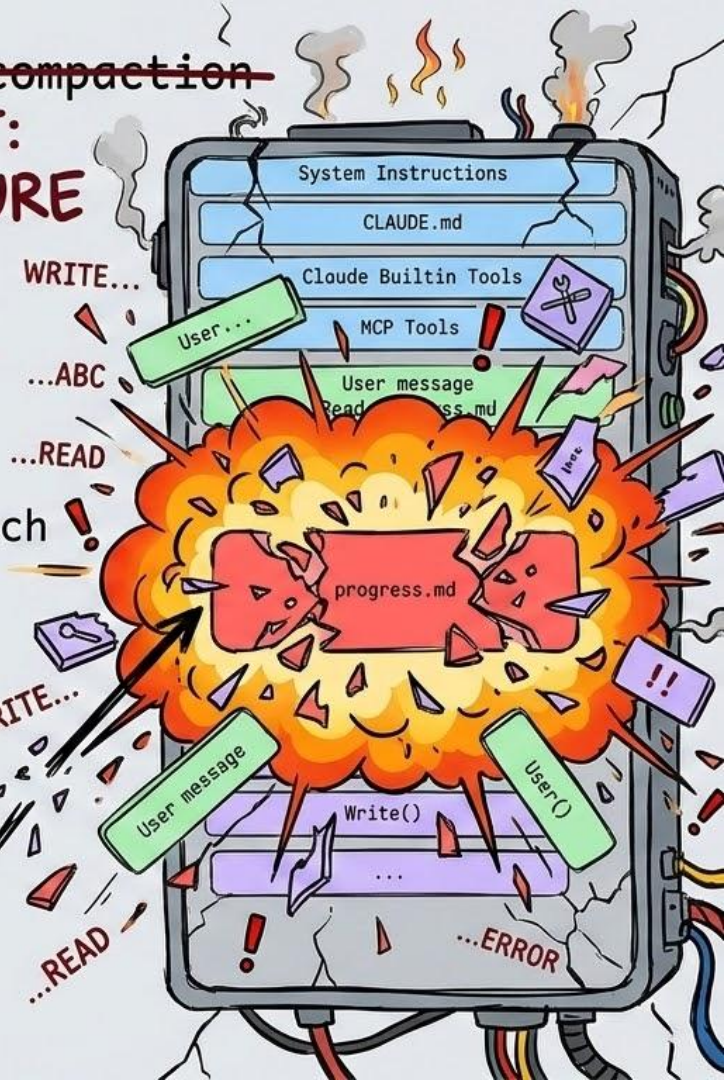
OVERFLOW MEMORY FULL OVERFLOW

"Write everything we did so far to progress.md, ensure to note the approach we're taking, the steps we've done so far, and the current failure we're working on"

OVERFLOW MEMORY FULL

MEMORY FULL

progress.md



Frequent ***Intentional*** Compaction

Let's build our ENTIRE WORKFLOW
around context management

Three Phases:

Research → Plan → Implement

Goal:

Stay in the smart zone
(< 50% context window usage)

Research

Understand how the system works

Find all relevant files

Find all relevant code blocks

Facts not Opinions

Research Codebase

You are tasked with conducting comprehensive research across the codebase to answer user questions by spawning parallel sub-agents and synthesizing their findings.

Initial Setup:

When this command is invoked, respond with:

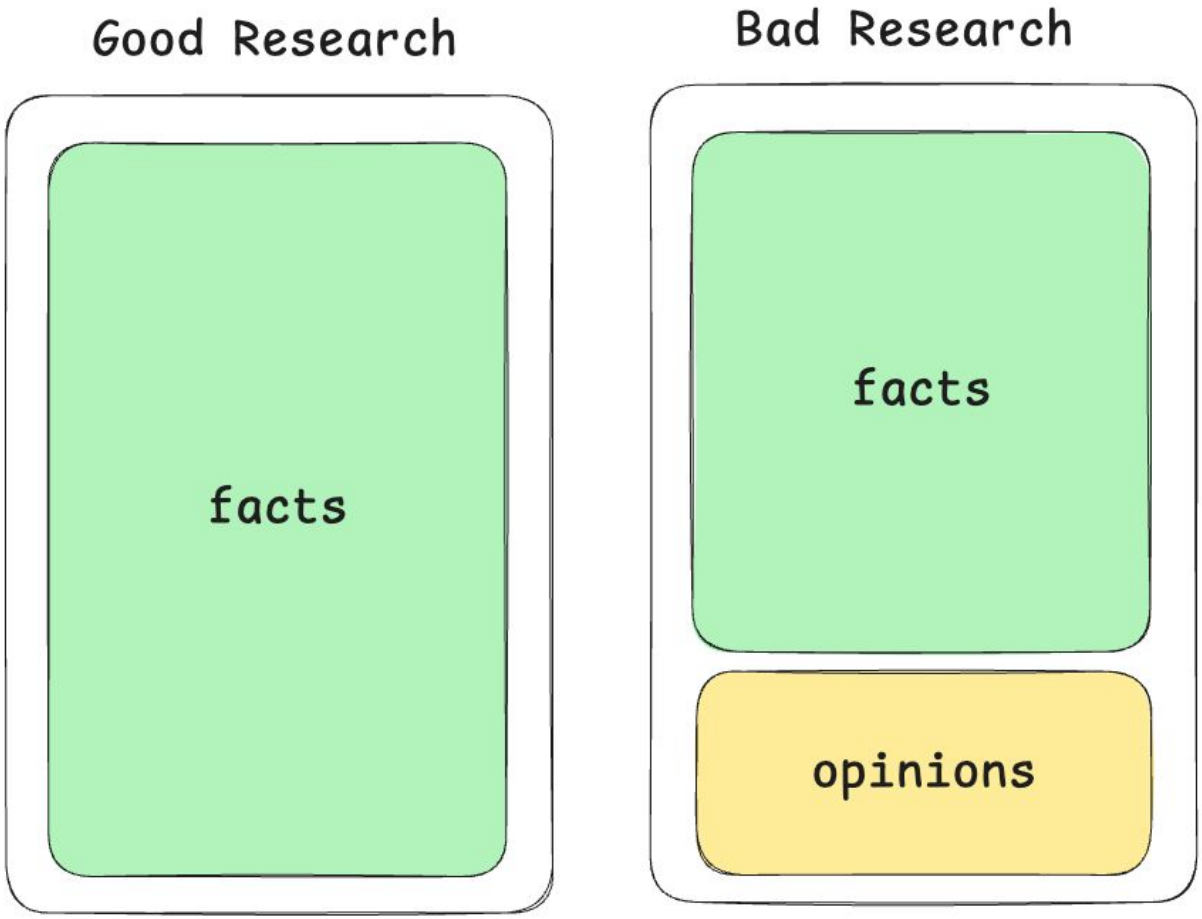
```
I'm ready to research the codebase. Please provide your research question or area of interest, and I'll analyze it t
```

Then wait for the user's research query.

Steps to follow after receiving the research query:

1. **Read any directly mentioned files first:**
   - If the user mentions specific files (tickets, docs, JSON), read them FULLY first
   - **IMPORTANT**: Use the Read tool WITHOUT limit/offset parameters to read entire files
   - **CRITICAL**: Read these files yourself in the main context before spawning any sub-tasks
   - This ensures you have full context before decomposing the research
2. **Analyze and decompose the research question:**
   - Break down the user's query into composable research areas
   - Take time to ultrathink about the underlying patterns, connections, and architectural implications the user might be seeking
   - Identify specific components, patterns, or concepts to investigate
   - Create a research plan using TodoWrite to track all subtasks
   - Consider which directories, files, or architectural patterns are relevant
3. **Spawn parallel sub-agent tasks for comprehensive research:**
   - Create multiple Task agents to research different aspects concurrently

   The key is to use these agents intelligently:
   - Start with locator agents to find what exists
   - Then use analyzer agents on the most promising findings
   - Run multiple agents in parallel when they're searching for different things
   - Each agent knows its job - just tell it what you're looking for
   - Don't write detailed prompts about HOW to search - the agents already know
4. **Wait for all sub-agents to complete and synthesize findings:**
   - IMPORTANT: Wait for ALL sub-agent tasks to complete before proceeding
   - Compile all sub-agent results (both codebase and thoughts findings)


Good Research
facts
Bad Research
facts
opinions

Summary

The background agent trigger service is a Node.js process running on Fly that manages the full lifecycle of AI agent machines. It consists of two HTTP servers (a public webhook handler on port 3000 and an internal 6PN API on port 3001), a SQLite database tracking agent state, and a periodic reaper/reconciler. Agents move through a status machine: provisioning -> running -> stopped -> destroyed, with infrastructure spanning three services (Fly machines+volumes, Cloudflare tunnels+DNS, GitHub comments).

Today, agent lifecycle is controlled exclusively through GitHub webhooks (label an issue to provision, merge a PR to destroy) or automatic reaper actions (soft timeout, runaway, idle, stale). The only manual tool is `provision.sh`, a standalone bash script that directly calls Fly and Cloudflare APIs without touching the trigger service's database -- meaning agents created or destroyed via `provision.sh` are invisible to the trigger service and vice versa.

The gap this CLI needs to fill: direct operator control over agent lifecycle (list, inspect, stop, destroy, wake) that goes through the trigger service's database so all state stays consistent. The trigger service already has all the building blocks as exported functions -- `provisionAgent`, `destroyAgent`, `stopAgent`, `wakeAndPrompt`, and the reaper/reconcile sweeps -- but no HTTP endpoints or CLI entry point exposes them for operator use.

Detailed Findings

Agent Status Machine

```
provisioning --> running --> stopped --> destroyed
                  |           |
                  +--failed   +--destroyed
                  |
                  +--destroyed (pr_merged)
```

- **provisioning**: DB record inserted, infrastructure being created
- **running**: Machine is active, tunnel/DNS live
- **stopped**: Machine stopped (tunnel/DNS preserved, volume preserved), can be restarted
- **failed**: Provisioning failed, infrastructure torn down
- **completed**: (unused in current code, exists in type)
- **destroyed**: Full teardown done (machine, volume, tunnel, DNS all deleted)

Stop reasons: `pr_merged`, `soft_timeout`, `runaway`, `idle`, `stale`

Database Schema (agents table)

Single table `agents` in SQLite (better-sqlite3 + Drizzle ORM, WAL mode):

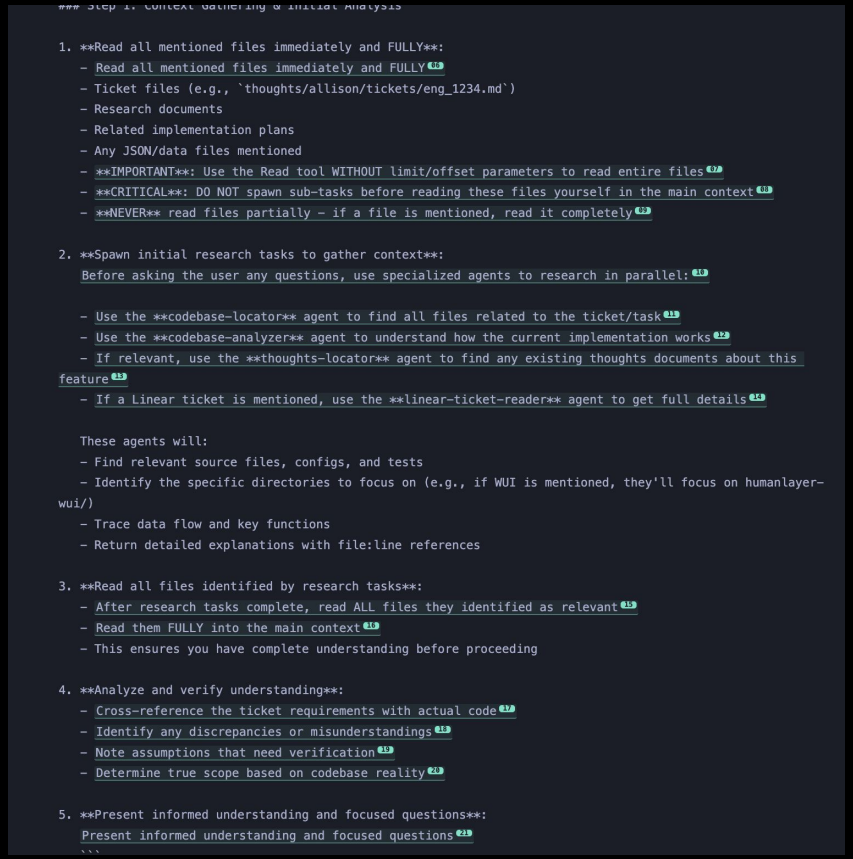
Column	Type	Notes
id	integer PK	auto-increment
agent_id	text NOT NULL	e.g. `issue-1626-28ad58`
issue_number	integer NOT NULL	GitHub issue number
repo	text NOT NULL	e.g. `nectry/nectry`
status	text NOT NULL	AgentStatus, default 'provisioning'
machine_id	text	Fly machine ID
tunnel_id	text	Cloudflare tunnel ID

Plan

Outline the Exact Implementation Steps

Include filenames, lines,
and code snippets

Explicit about testing steps

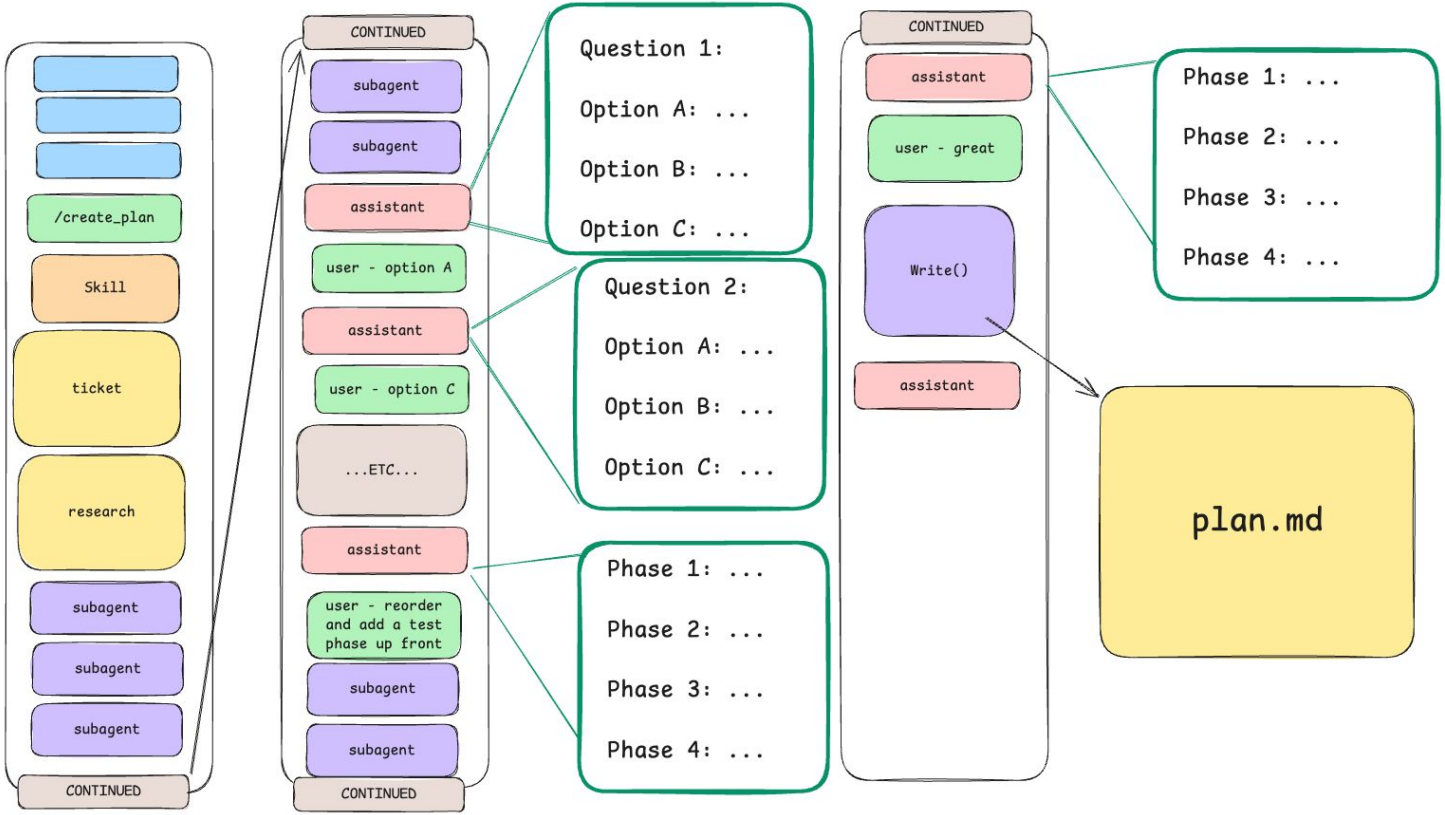


85+

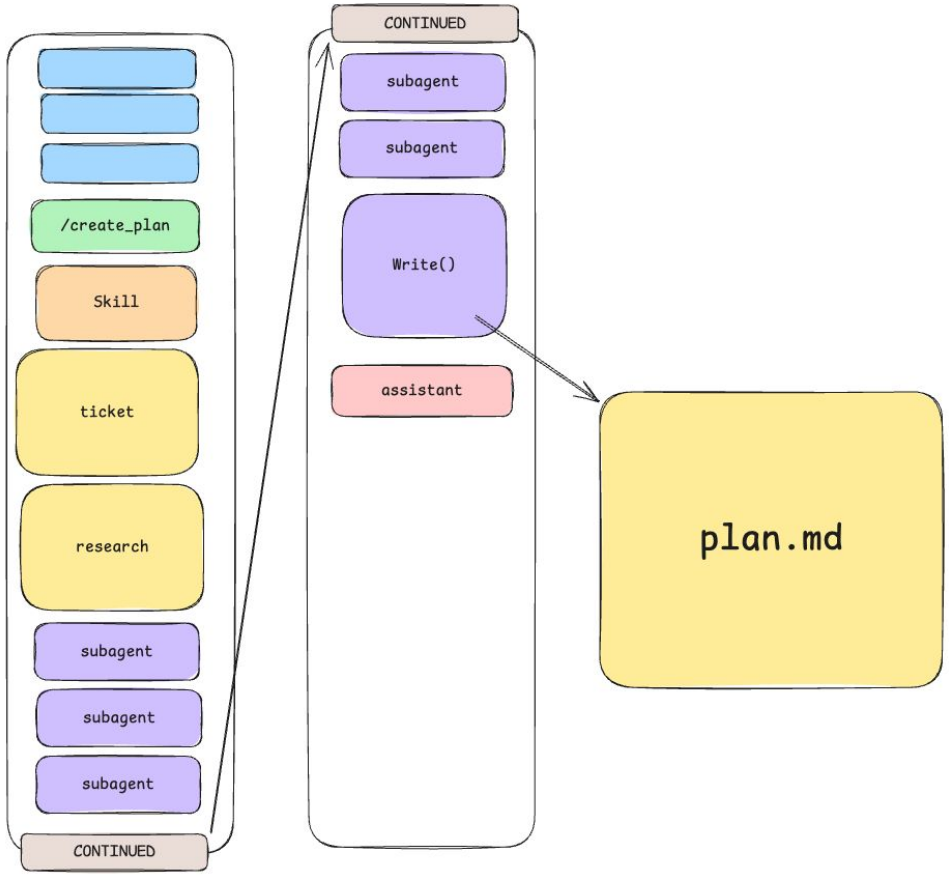
instructions

+ claude.md
+ system prompt
+ tools
+ mcps

The wrong way



The right way



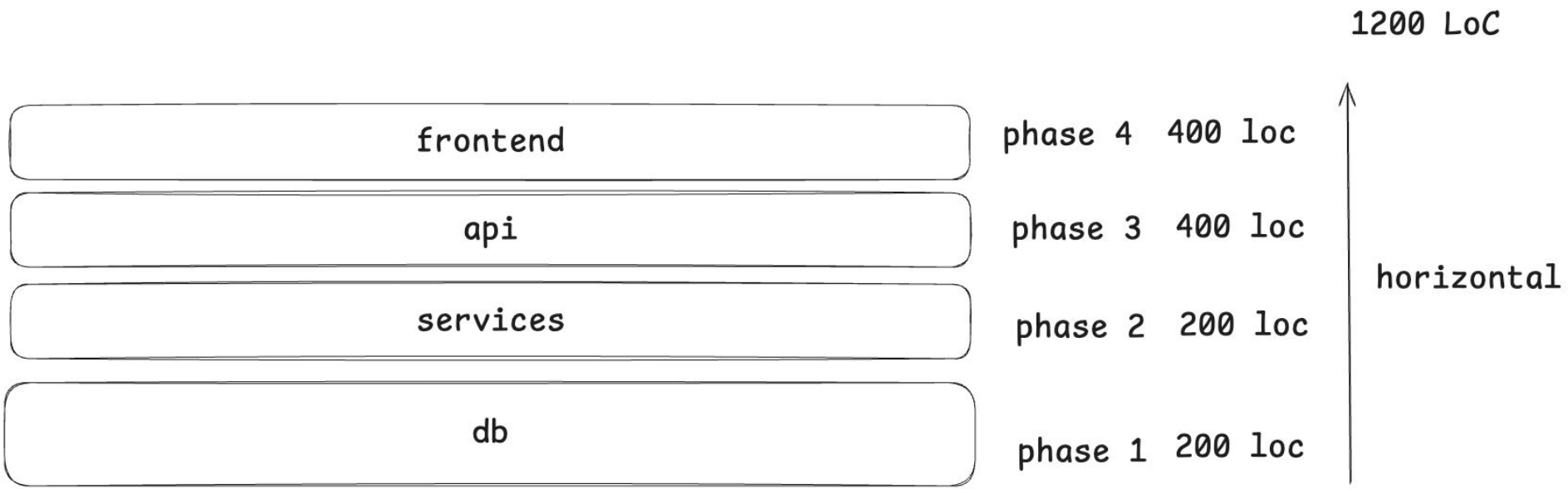
Good plans are:

Phase ordered

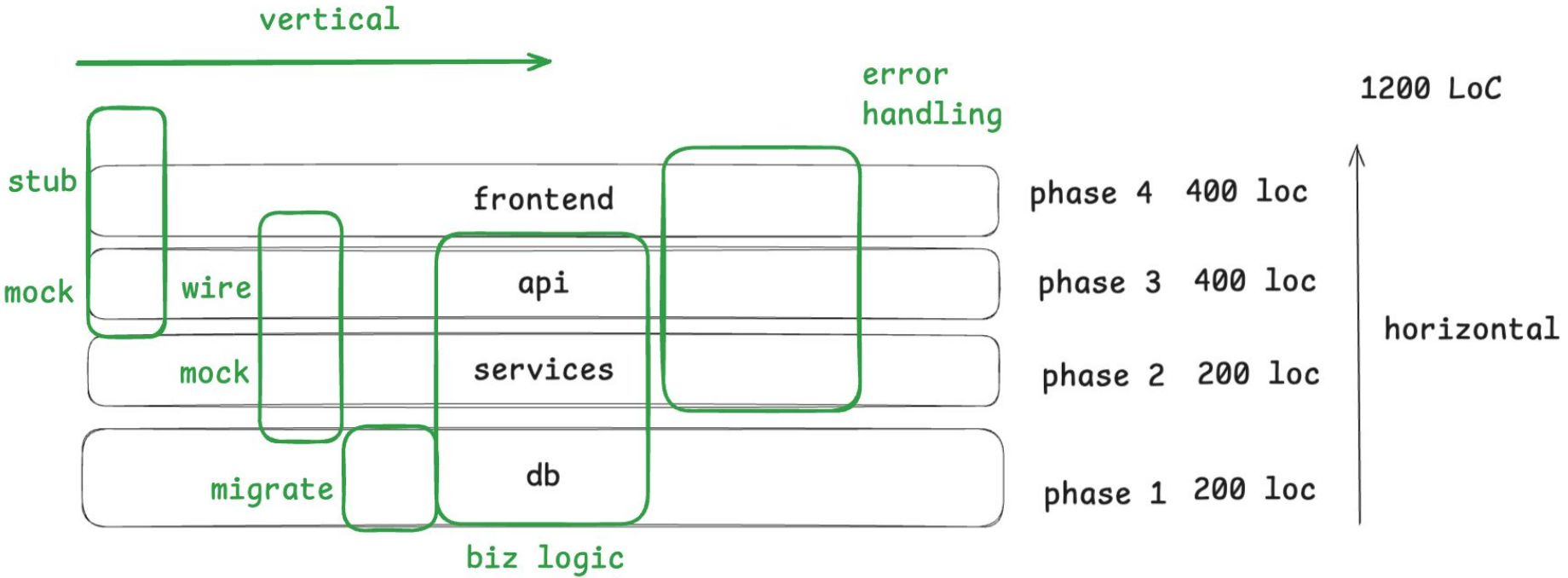
Vertically sliced

Explicit

Horizontal Plans == Bad



Vertical plans == Good



Implement

Go write the code

"If properly planned, the implementation is easy and expected."

Keep Context under 50%

Implementation Philosophy

Plans are carefully designed, but reality can be messy. Your job is to:

- Follow the plan's intent while adapting to what you find
- Implement each phase fully before moving to the next
- Verify your work makes sense in the broader codebase context
- Update checkboxes in the plan as you complete sections

When things don't match the plan exactly, think about why and communicate clearly. The plan is your guide, but your judgment matters too.

If you encounter a mismatch:

- STOP and think deeply about why the plan can't be followed
- Present the issue clearly:

```
Issue in Phase [N]:
Expected: [what the plan says]
Found: [actual situation]
Why this matters: [explanation]

How should I proceed?
```

Verification Approach

After implementing a phase:

- Run the success criteria checks (usually `make check test` covers everything)
- Fix any issues before proceeding
- Update your progress in both the plan and your todos
- Check off completed items in the plan file itself using Edit
- **Pause for human verification**: After completing all automated verification for a phase, pause and inform the human that the phase is ready for manual testing. Use this format:

```
Phase [N] Complete — Ready for Manual Verification

Automated verification passed:
— [List automated checks that passed]

Please perform the manual verification steps listed in the plan:
— [List manual verification items from the plan]

Let me know when manual testing is complete so I can proceed to Phase [N+1].
```

If instructed to execute multiple phases consecutively, skip the pause until the last phase. Otherwise, assume you are just doing one phase.

do not check off items in the manual testing steps until confirmed by the user.

If You Get Stuck

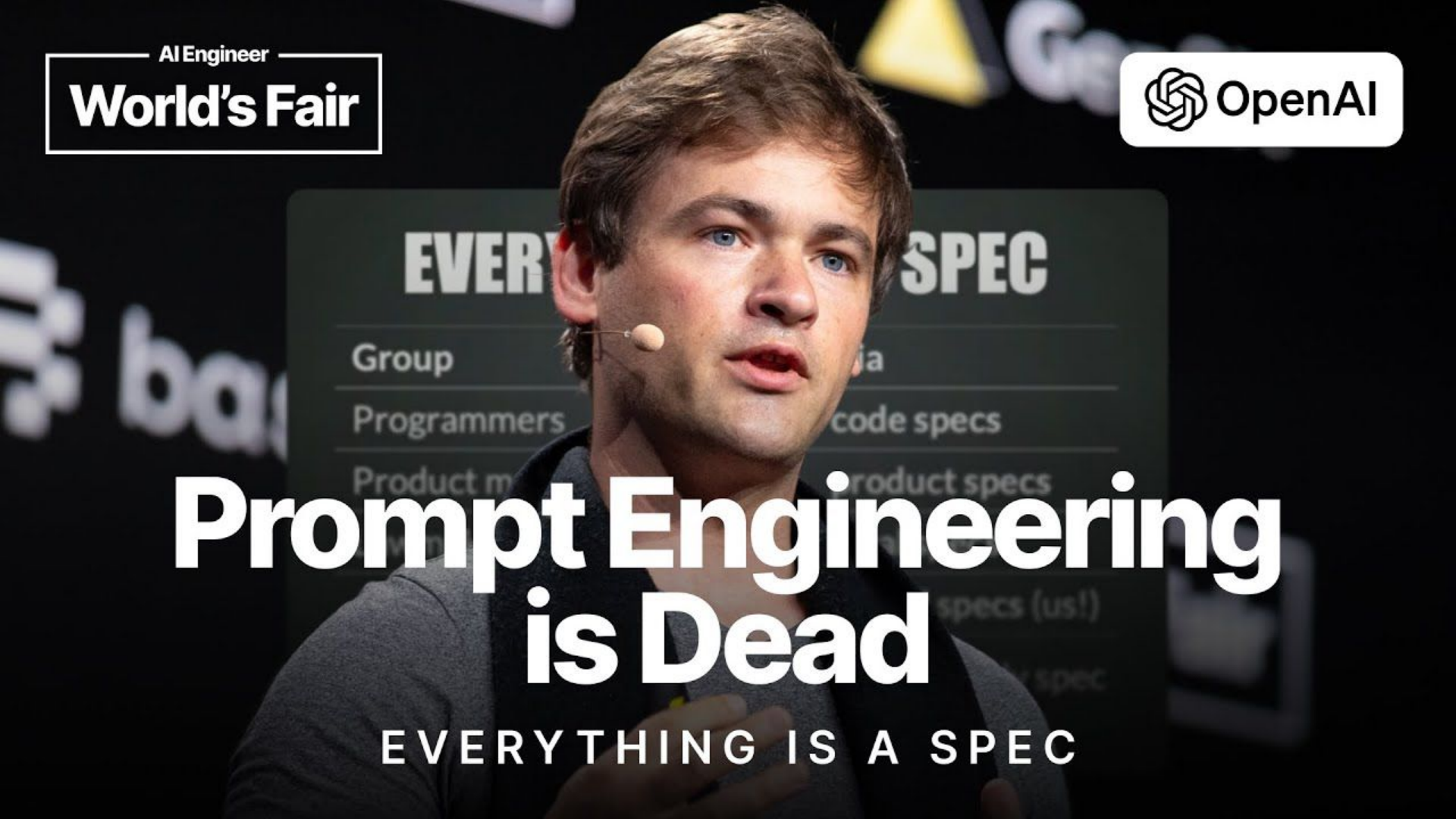
Please read the code

DO NOT OUTSOURCE THE THINKING

“the hard work of thinking can’t be outsourced to AI, only amplified by it”

Isn’t this just
Spec Driven Development?

Yes, but…


AI Engineer
World's Fair
OpenAI
Group
Programmers
code specs
Prompt Engineering
is Dead
EVERYTHING IS A SPEC

SDD is writing a PRD first

SDD is using Verifiable Feedback Loops

SDD is treating the code like ASM

SDD is using a bunch of MD files while coding

So if SDD is ~~hype-slop,~~
“semantically diffused”
what do I need?

Research == Compression

Planning == Leverage

Don’t Outsource the Thinking

Be wary of token burning slot machine

Research == Compression

Understand how the system works

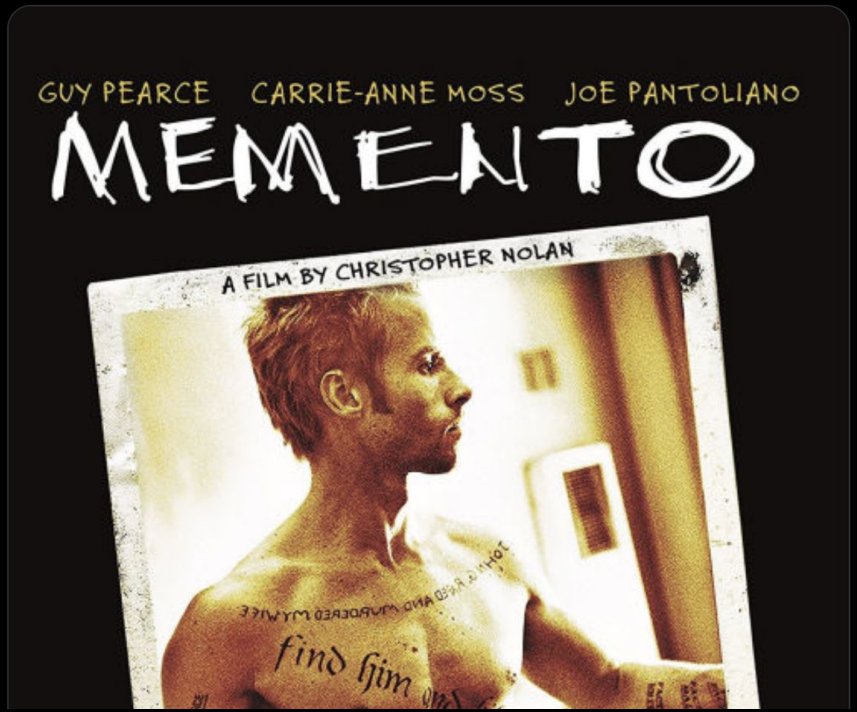
Find All Relevant Files

Stay Objective

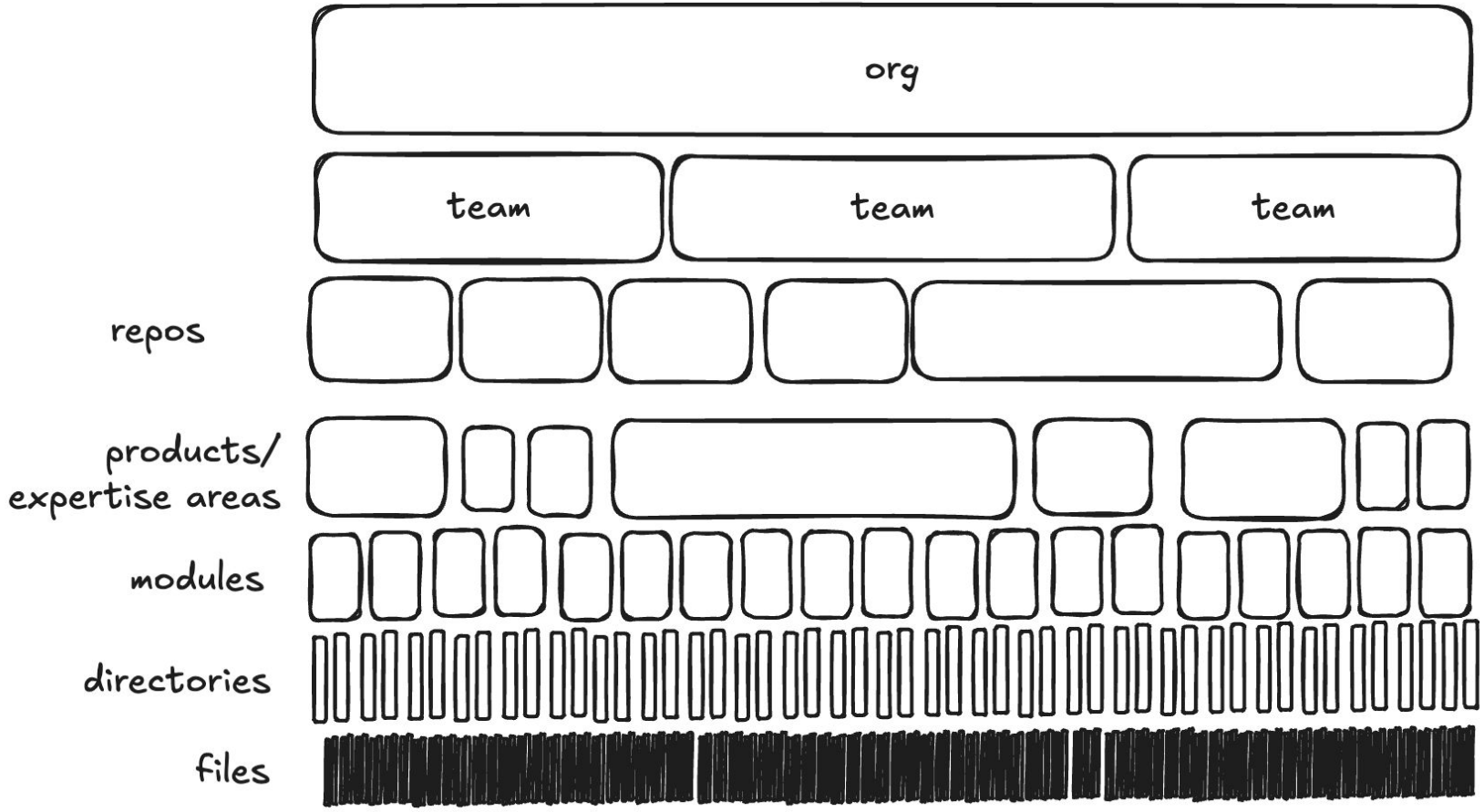
Agents need to be “onboarded”

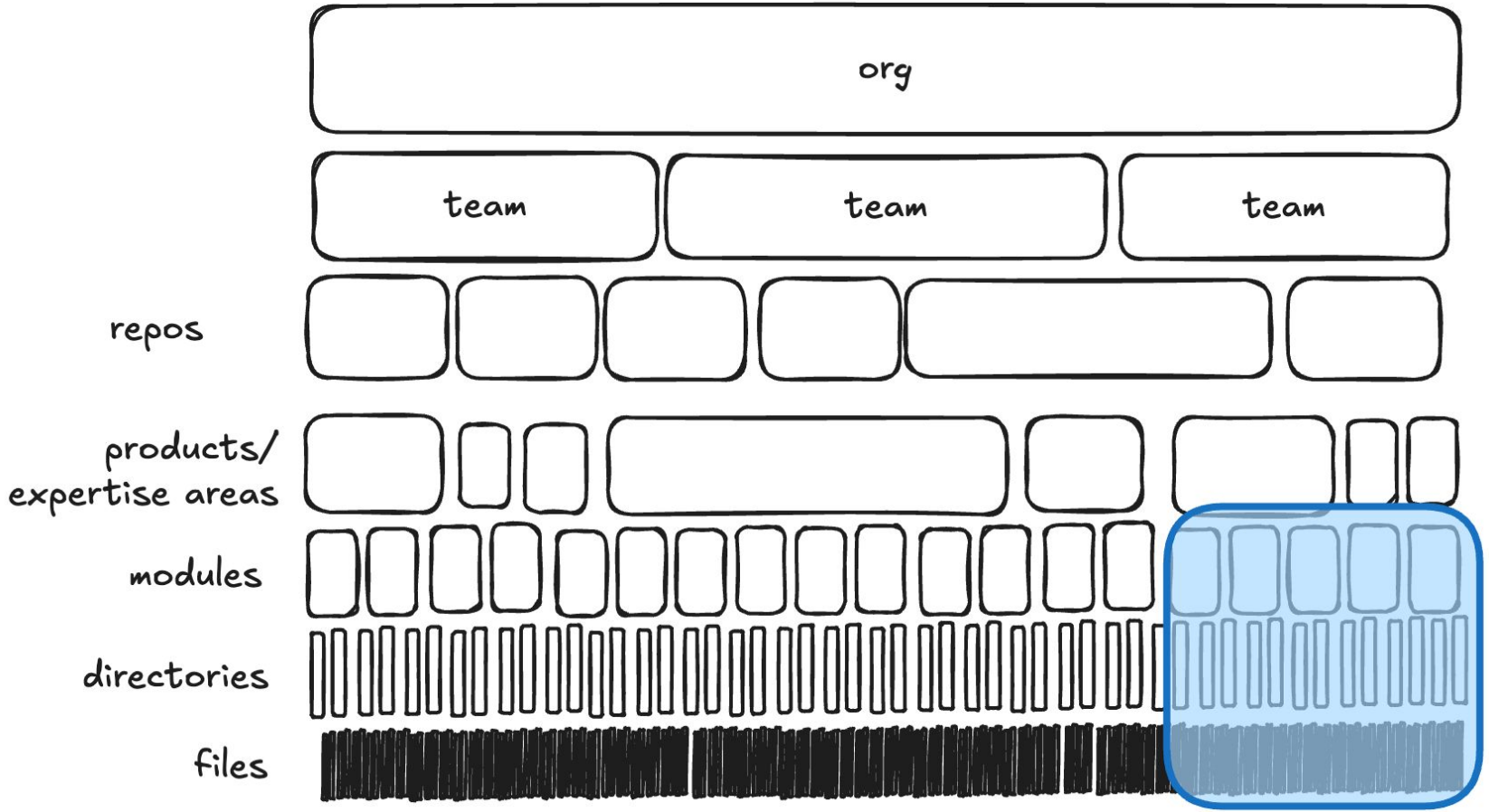
Peter J. Liu
@peterjliu

the best movie on context engineering

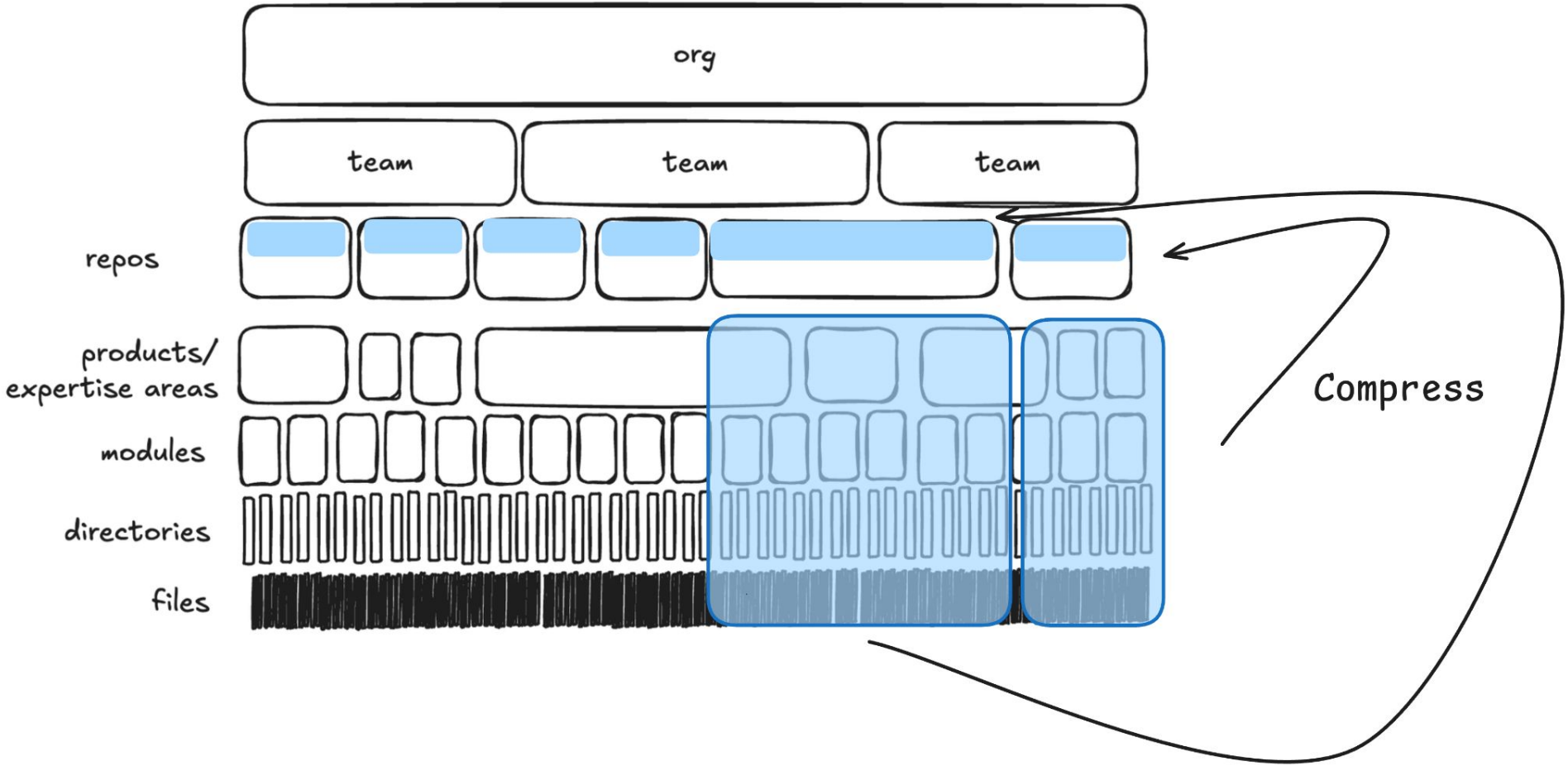


Otherwise they make stuff up!


org
team
team
team
repos
products/
expertise areas
modules
directories
files


org
team
team
team
repos
products/
expertise areas
modules
directories
files

Docs at the repo root

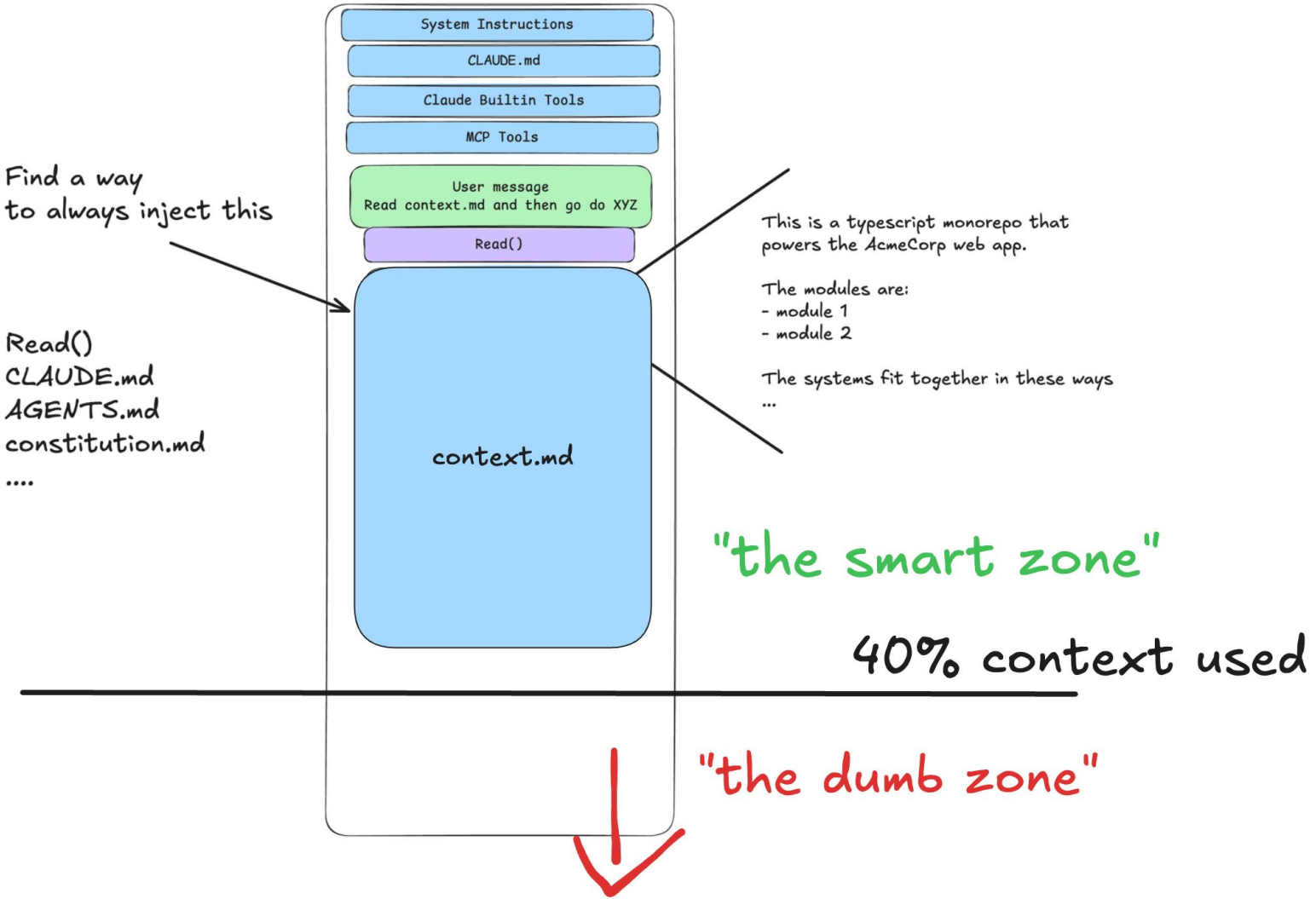


Challenges w/ this approach

Underspecified

Too long

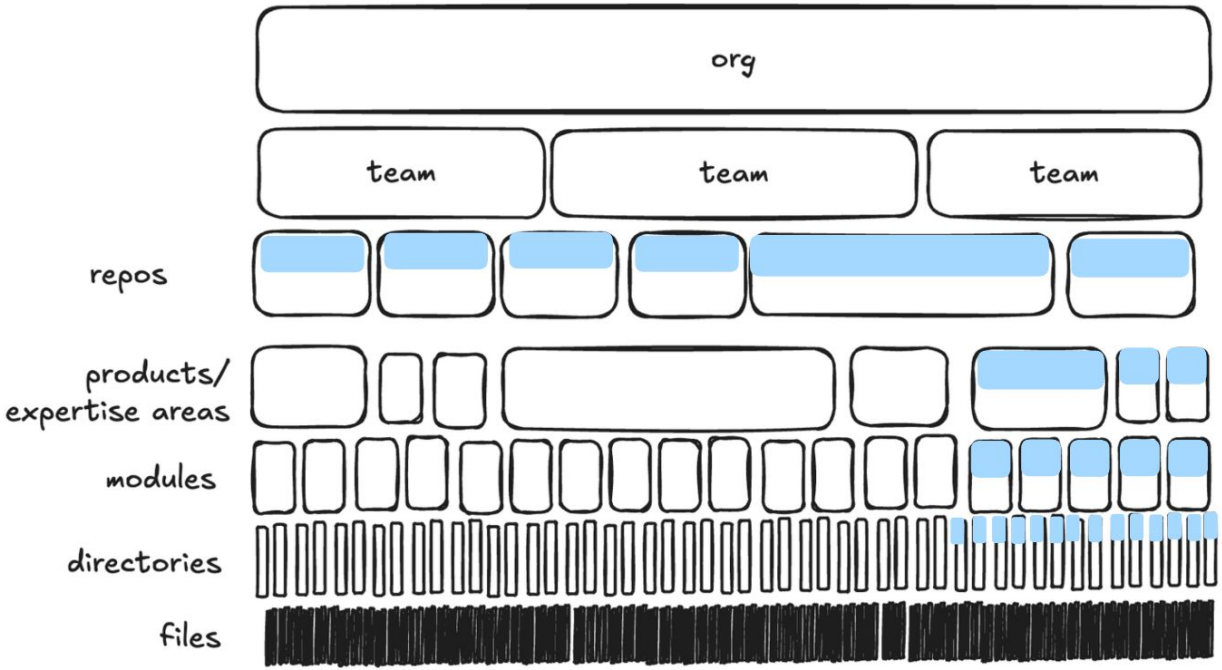
Incorrect


System Instructions
CLAUDE.md
Claude Builtin Tools
MCP Tools
User message
Read context.md and then go do XYZ
Read()
context.md
Find a way
to always inject this
Read()
CLAUDE.md
AGENTS.md
constitution.md
....
This is a typescript monorepo that
powers the AcmeCorp web app.
The modules are:
- module 1
- module 2
The systems fit together in these ways
...
"the smart zone"
40% context used
"the dumb zone"

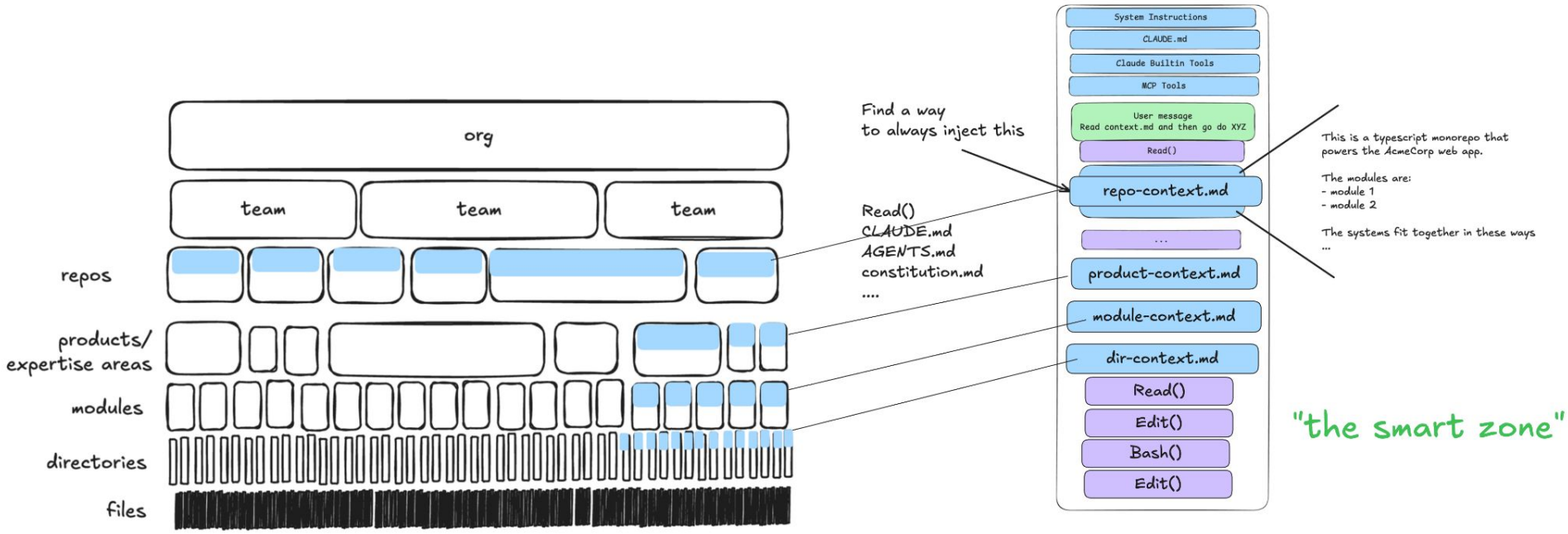
What about
Progressive Disclosure?

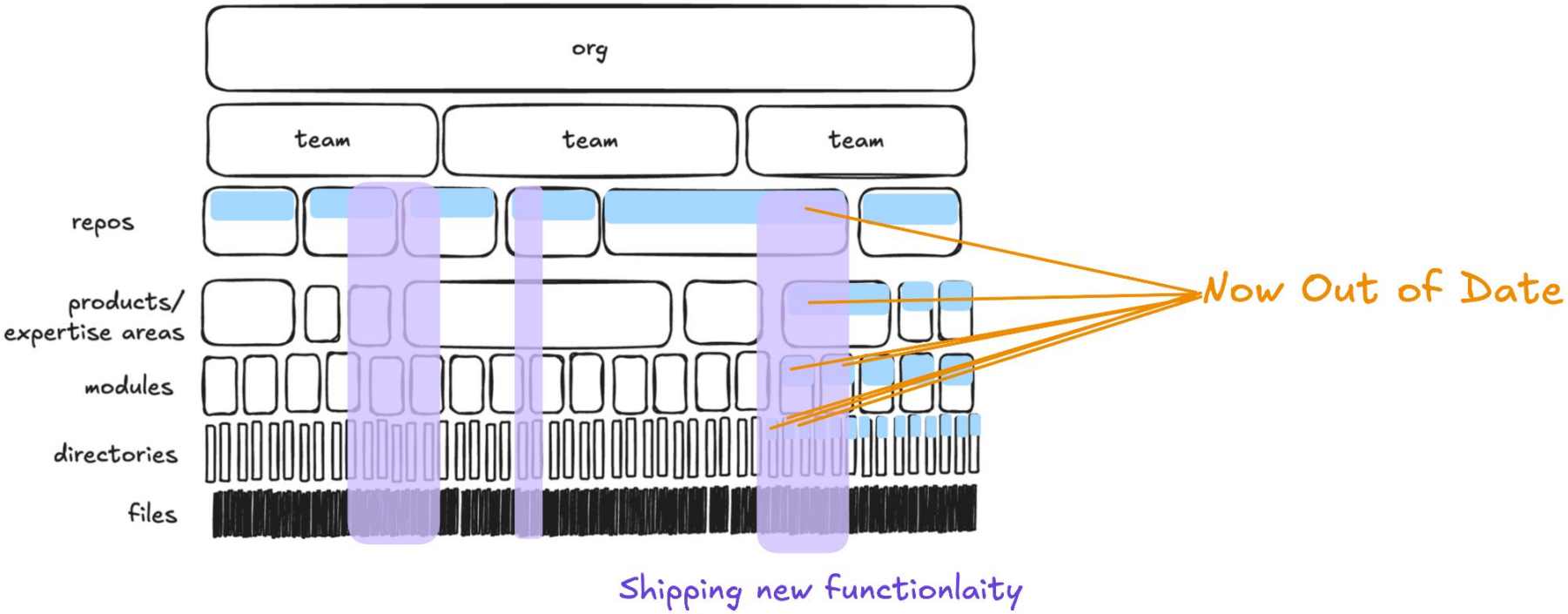
Shard the info down the stack

Docs in each directory

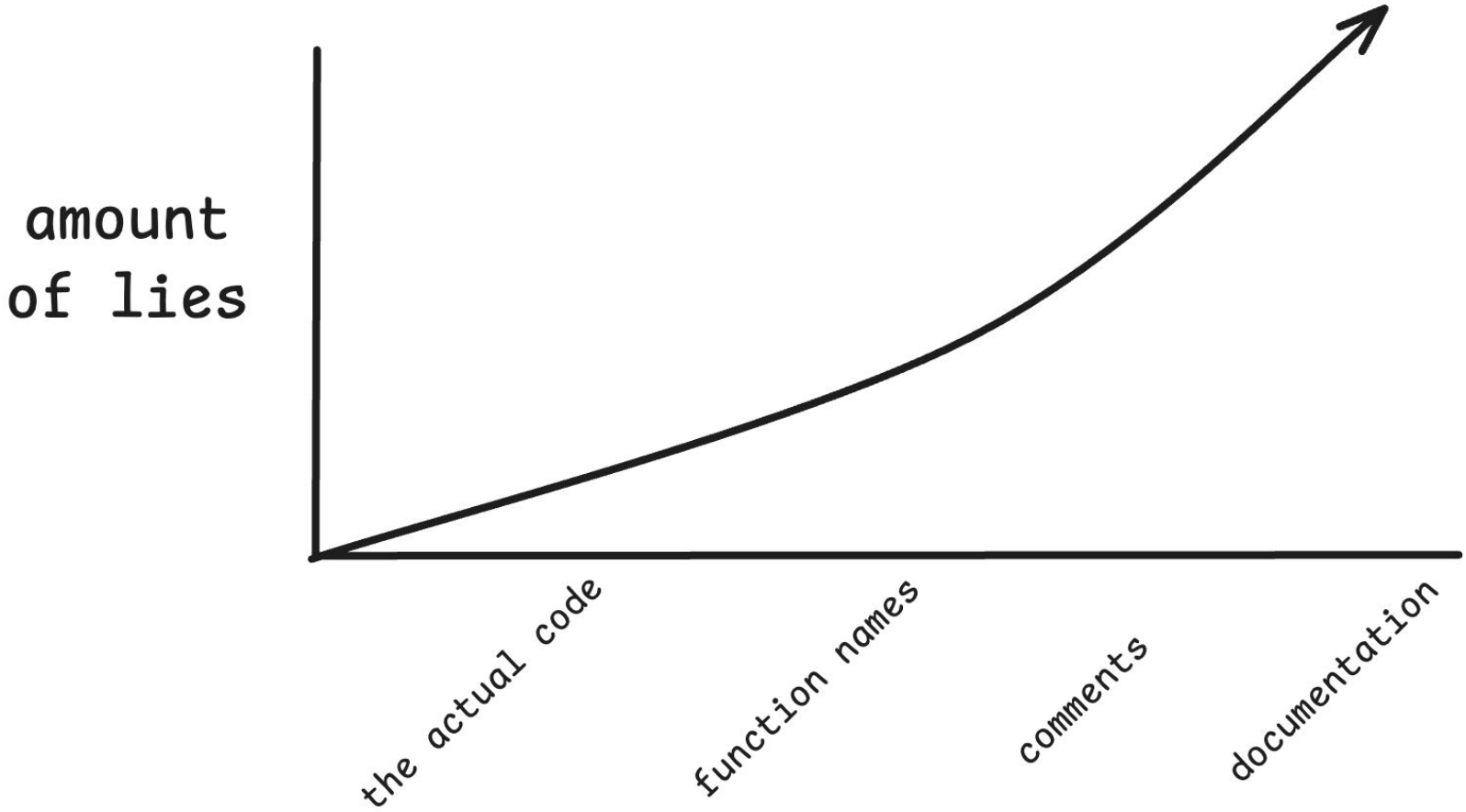


Don't Document files, they ARE the source of truth


org
team
team
team
repos
products/
expertise areas
modules
directories
files
Find a way
to always inject this
Read()
CLAUDE.md
AGENTS.md
constitution.md
....
System Instructions
CLAUDE.md
Claude Builtin Tools
MCP Tools
User message
Read context.md and then go do XYZ
Read()
repo-context.md
...
product-context.md
module-context.md
dir-context.md
Read()
Edit()
Bash()
Edit()
This is a typescript monorepo that powers the AcmeCorp web app.
The modules are:
- module 1
- module 2
The systems fit together in these ways
...
"the smart zone"


org
team
team
team
repos
products/
expertise areas
modules
directories
files
Now Out of Date
Shipping new functionlaity

You *could* make it part of your PR merge process to update this


amount
of lies
the actual code
function names
comments
documentation

The best approach

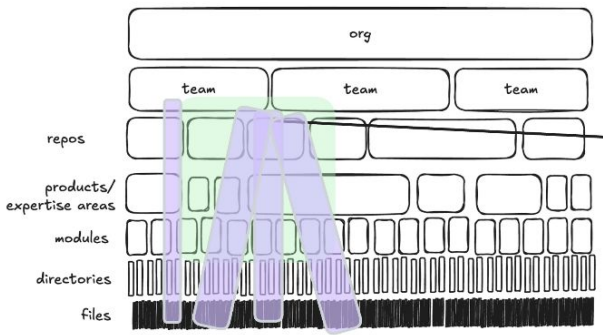
On-demand compressed context

(with steering)

```
❯ /research Let's refactor the ide/src/app/llm direcrtory. We will replace the custom tool
  call loop impl with Vercel's AI SDK. We want to rip out all the LLM provider abstractions
  and replace all the custom tool definition and tool loop implementation with AI SDK.

  There are some quirks here. We have a custom plan mode. Our custom tool loop allows things
  like transitioning between modes and disallowing certain tool calls in different modes. We
  should strive to preserve a that functionality to the best of our ability. The backend
  doesn't have to work exactly the same way it works now but it should be semantically
  similar enough such that the frontend does not noticeably change.

  ~/Code/nectry | (main) | Opus 4.7 (1M context) | 4%
  ▸▸ bypass permissions on (shift+tab to cycle)
                                    control this session from your phone · /remote-control
```



System Instructions

CLAUDE.md

Claude Builtin Tools

MCP Tools

User message
/research_codebase ...

subagent: codebase-locator

subagent: codebase-analyzer

subagent: codebase-analyzer

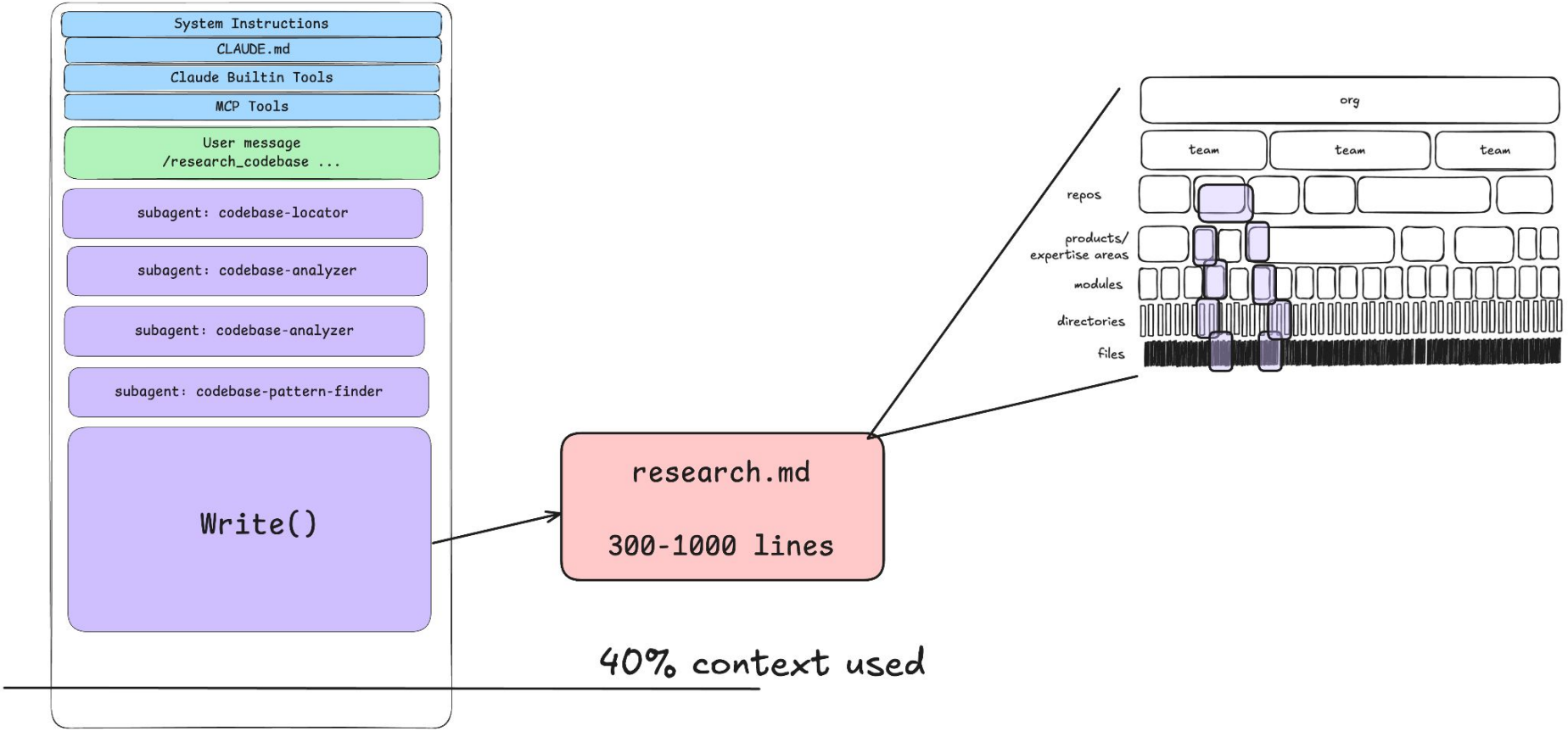
subagent: codebase-pattern-finder

Write()

research.md

300-1000 lines

40% context used


System Instructions
CLAUDE.md
Claude Builtin Tools
MCP Tools
User message
/research_codebase ...
subagent: codebase-locator
subagent: codebase-analyzer
subagent: codebase-analyzer
subagent: codebase-pattern-finder
Write()
research.md
300-1000 lines
40% context used
org
team
team
team
repos
products/
expertise areas
modules
directories
files

Keep things Objective

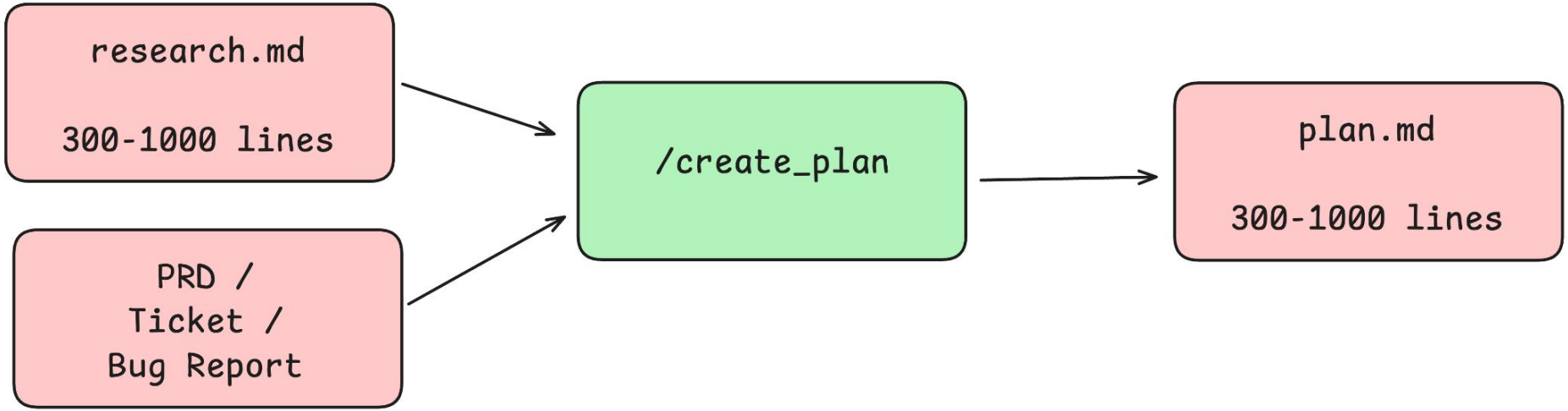
Discourage opinions

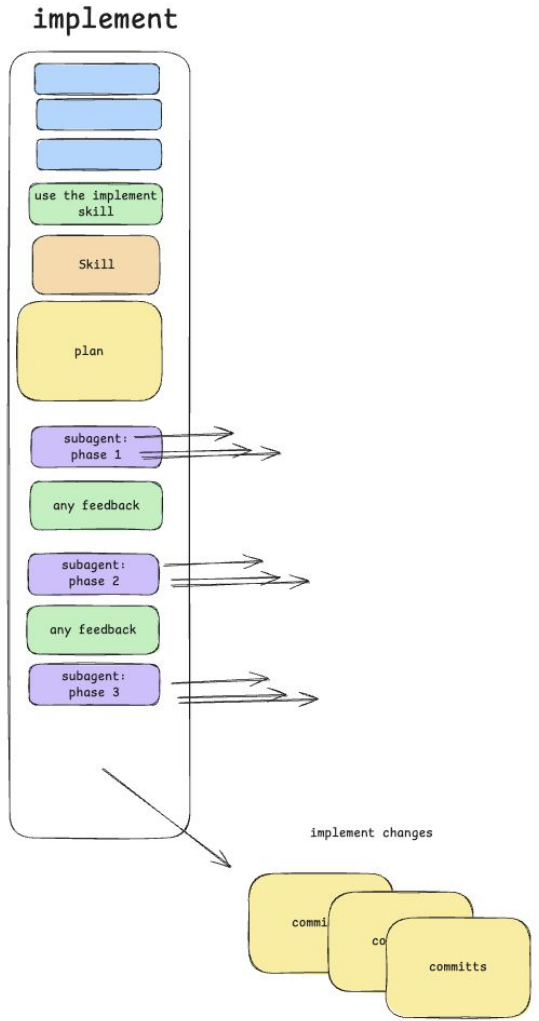
Avoid implementation planning

Research == Compression of Truth

Planning == Leverage

Compression of intent


research.md
300-1000 lines
PRD /
Ticket /
Bug Report
/create_plan
plan.md
300-1000 lines



Each phase should be executed by a subagent

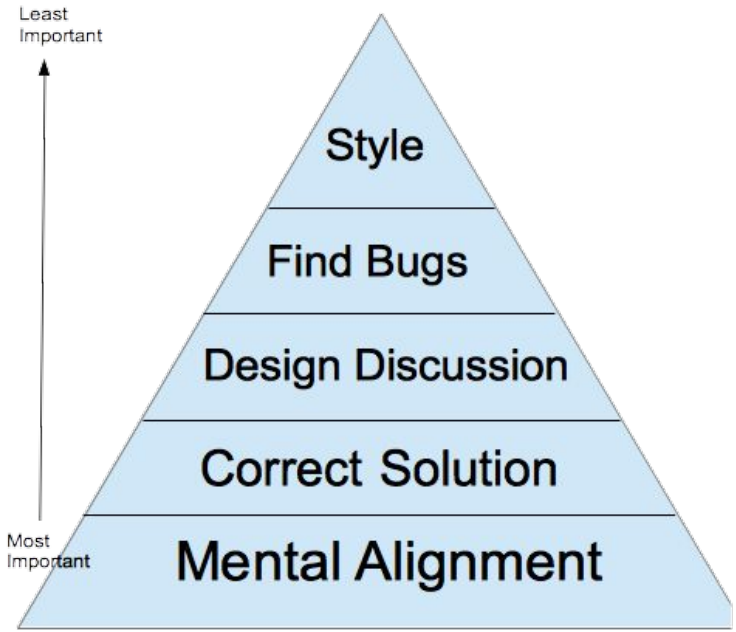
Each subagent should commit its changes

Each commit should be independently testable

Goal:
Mental Alignment

What is code review for?

Code Review: Hierarchy of Needs



https://blakesmith.me/2015/02/09/code-review-essentials-for-software-teams.html

Reading thousands of lines of code each week is hard

Reading 300 lines of an implementation plan makes it easier

If the plans are good,
that's *nearly* enough

Catch misalignment early

Maintain System Understanding

PLEASE READ THE CODE

Goal: Leverage

You want high confidence the model will do the right thing

Checkboxes are not enough

Overview

Implement optimizer pipeline using Anthropic's Claude with structured output via Vercel AI SDK. Use

Phase 2.1: Vercel AI SDK Integration

- [ ] Install Vercel AI SDK dependencies (ai, @ai-sdk/anthropic)
- [ ] Create environment variable utilities (required env, optional env, .env loading)
- [ ] Implement basic Anthropic client wrapper with structured output support
- [ ] Configure client with model, temperature (0.2), and API key
- [ ] Create unit tests for environment handling and client creation
- [ ] Wire environment variable loading into CLI pipeline
- [ ] Verify build succeeds
- [ ] Verify tests pass
- [ ] Verify Biome linting passes
- [ ] Manual test: Run with valid API key shows client configured
- [ ] Manual test: Run without API key shows clear error

Phase 2.2: Optimizer Schema and Template Builder

- [ ] Define Zod schemas for optimizer structured output (FileGroup, OptimizedPrompt)
- [ ] Implement file validation (check files exist in input)
- [ ] Implement GROUP marker validation (markers match group names, correct format)
- [ ] Build system prompt function
- [ ] Build user prompt with placeholder replacement (FILES_ARRAY, USER_PROMPT)
- [ ] Implement plan_structure.md auto-addition for plan type

5. Update Pipeline

File: `packages/gpt5-reasoner/src/pipeline.ts`
Changes: Wire in environment variable loading

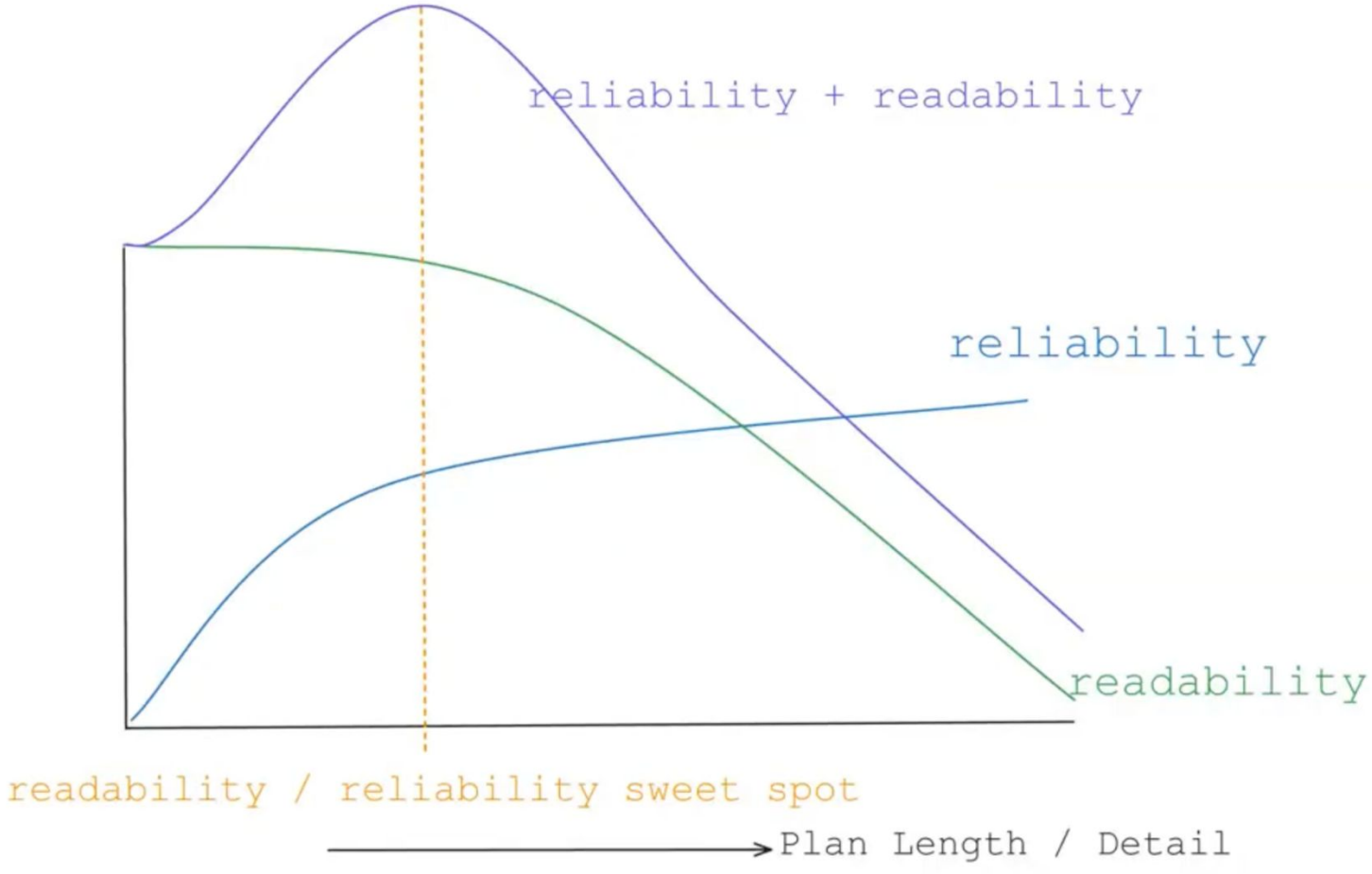
```typescript
import type { RunCommandArgs, ValidatedInput } from "./types";
import { PromptTypeSchema } from "./types";
import { loadFilesJson, loadDirectoriesJson, loadPromptFile } from "./loader";
import { loadDotEnv } from "./env";

/**
 * Main pipeline function (enhanced for Phase 2.1)
 */
export async function runPipeline(args: RunCommandArgs): Promise<void> {
  // Load .env if requested
  if (args.dotEnv) {
    loadDotEnv();
    console.log("Loaded .env file");
  }

  // Validate prompt type
  const promptType = PromptTypeSchema.parse(args.promptType);

  // Get prompt from either --prompt or --prompt-file
  const prompt = args.prompt ?? (await loadPromptFile(args.promptFile!));
```

You want compression of intent and reliable execution


reliability + readability
reliability
readability
readability / reliability sweet spot
Plan Length / Detail

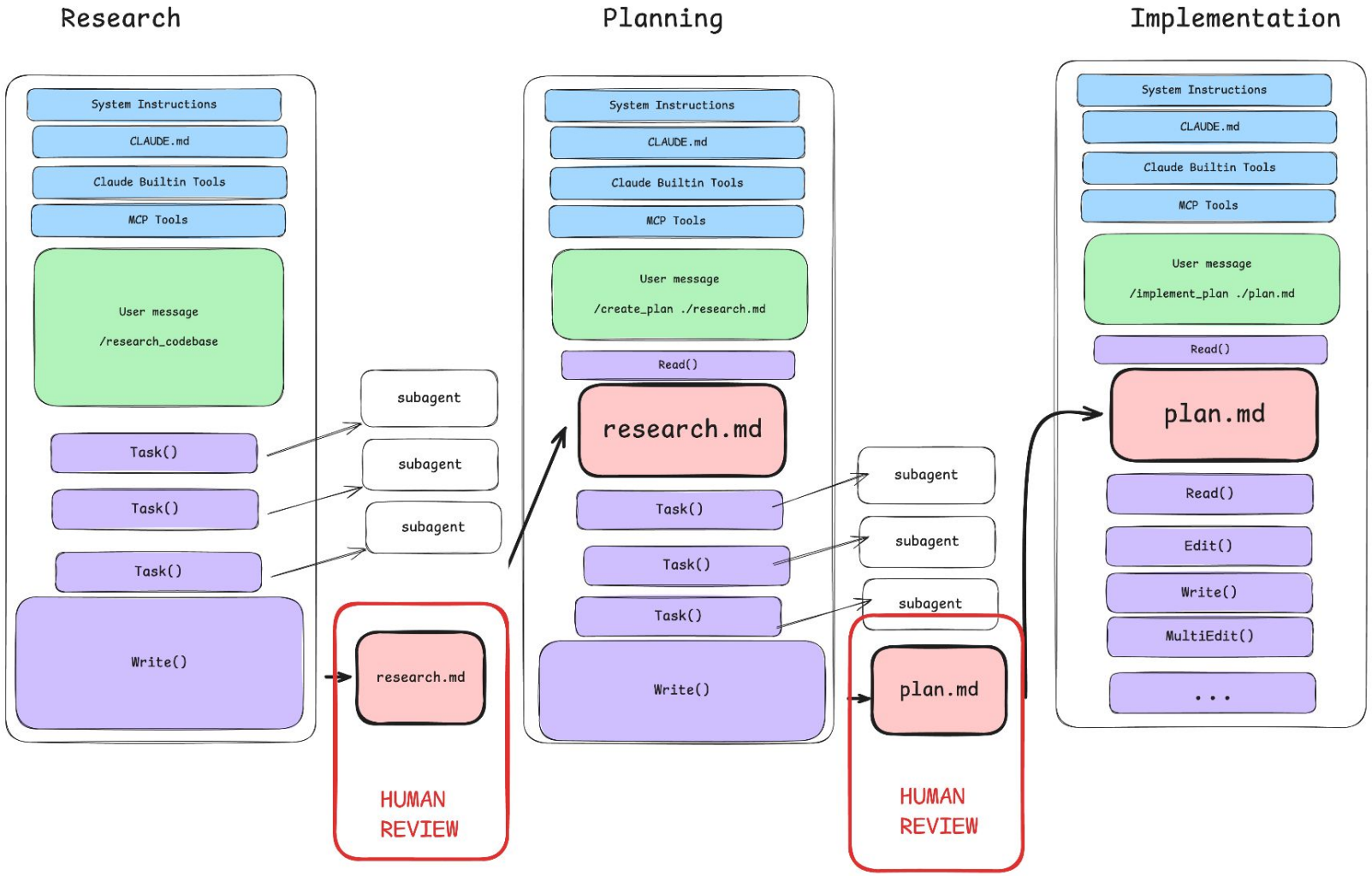
Review the Research and the Plans to achieve

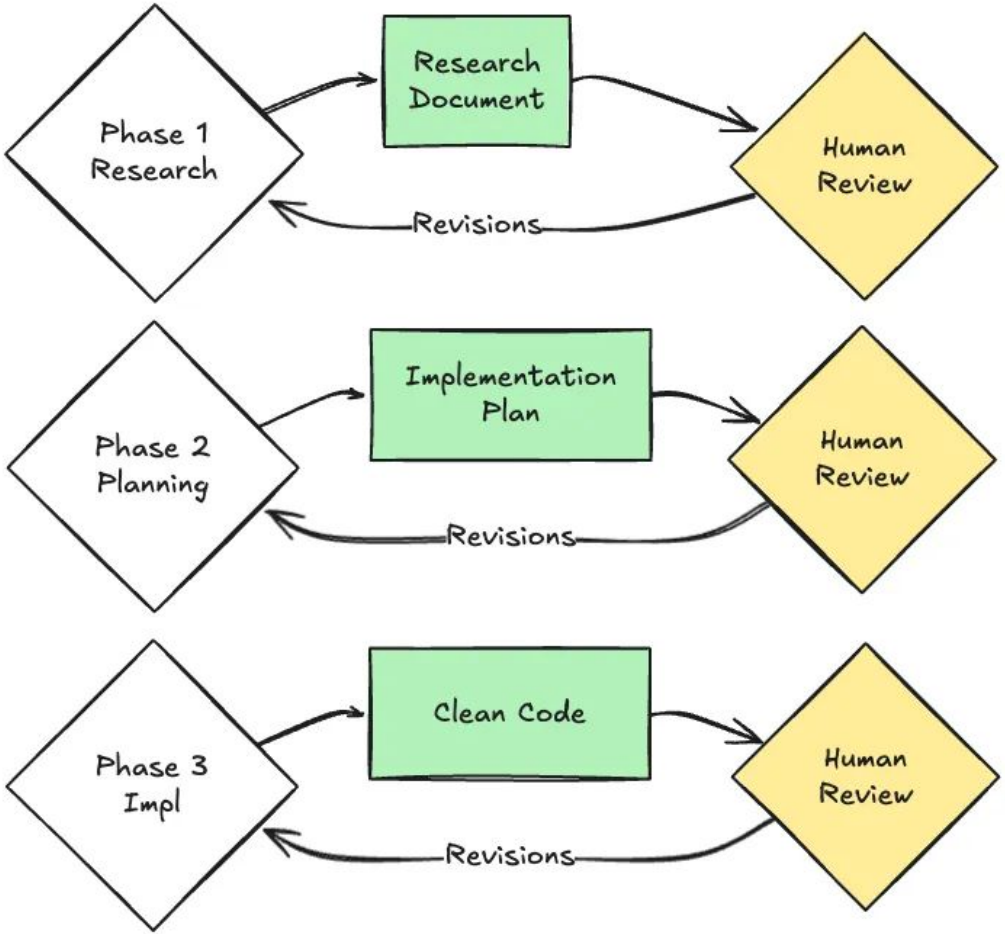
Mental Alignment

Do not
Outsource the thinking

This is not magic

You still have to
read the plan


Research
System Instructions
CLAUDE.md
Claude Builtin Tools
MCP Tools
User message
/research_codebase
Task()
Task()
Task()
Write()
subagent
subagent
subagent
research.md
HUMAN
REVIEW
Planning
System Instructions
CLAUDE.md
Claude Builtin Tools
MCP Tools
User message
/create_plan ./research.md
Read()
research.md
Task()
Task()
Task()
Write()
subagent
subagent
subagent
plan.md
HUMAN
REVIEW
Implementation
System Instructions
CLAUDE.md
Claude Builtin Tools
MCP Tools
User message
/implement_plan ./plan.md
Read()
plan.md
Read()
Edit()
Write()
MultiEdit()
. . .



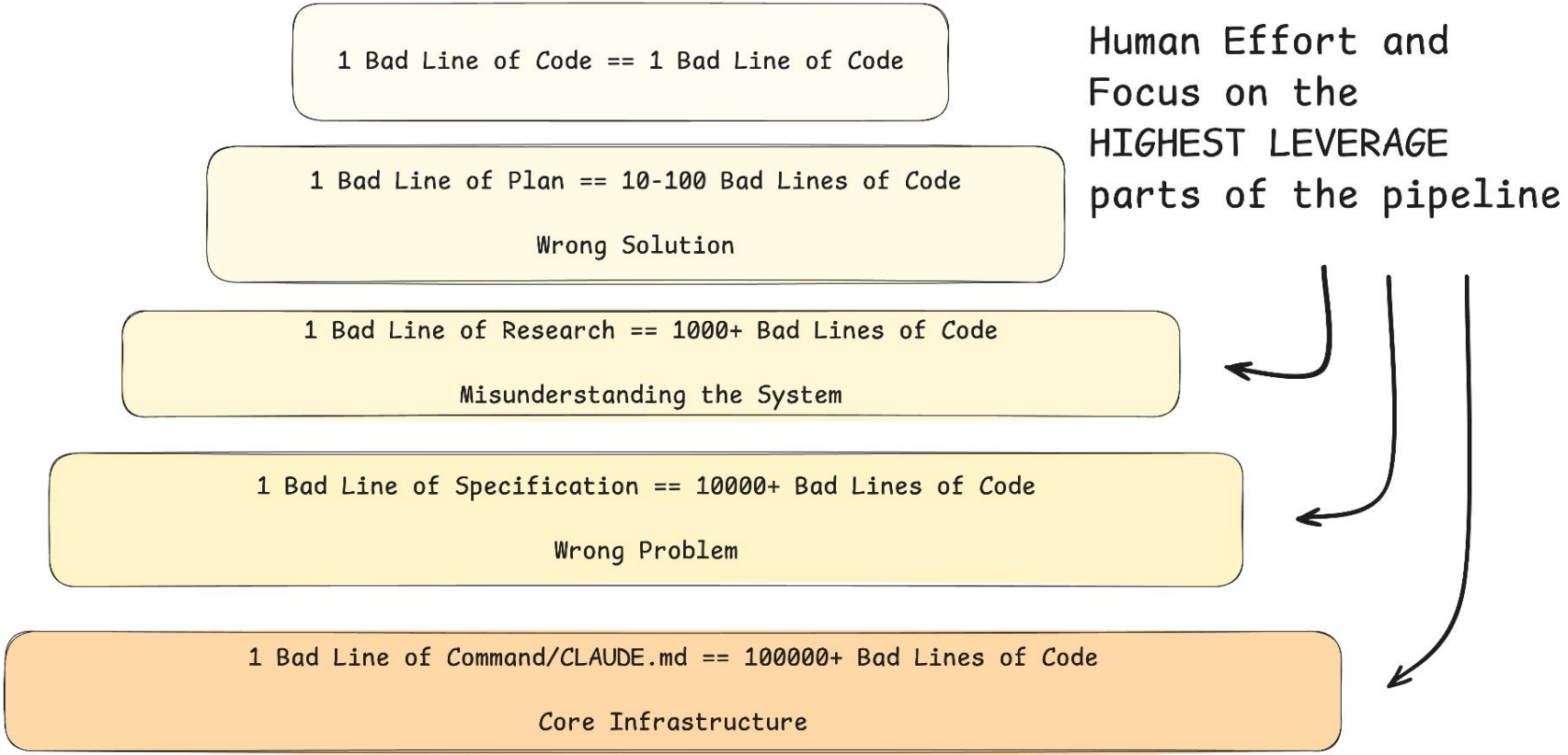
JAKE NATIONS

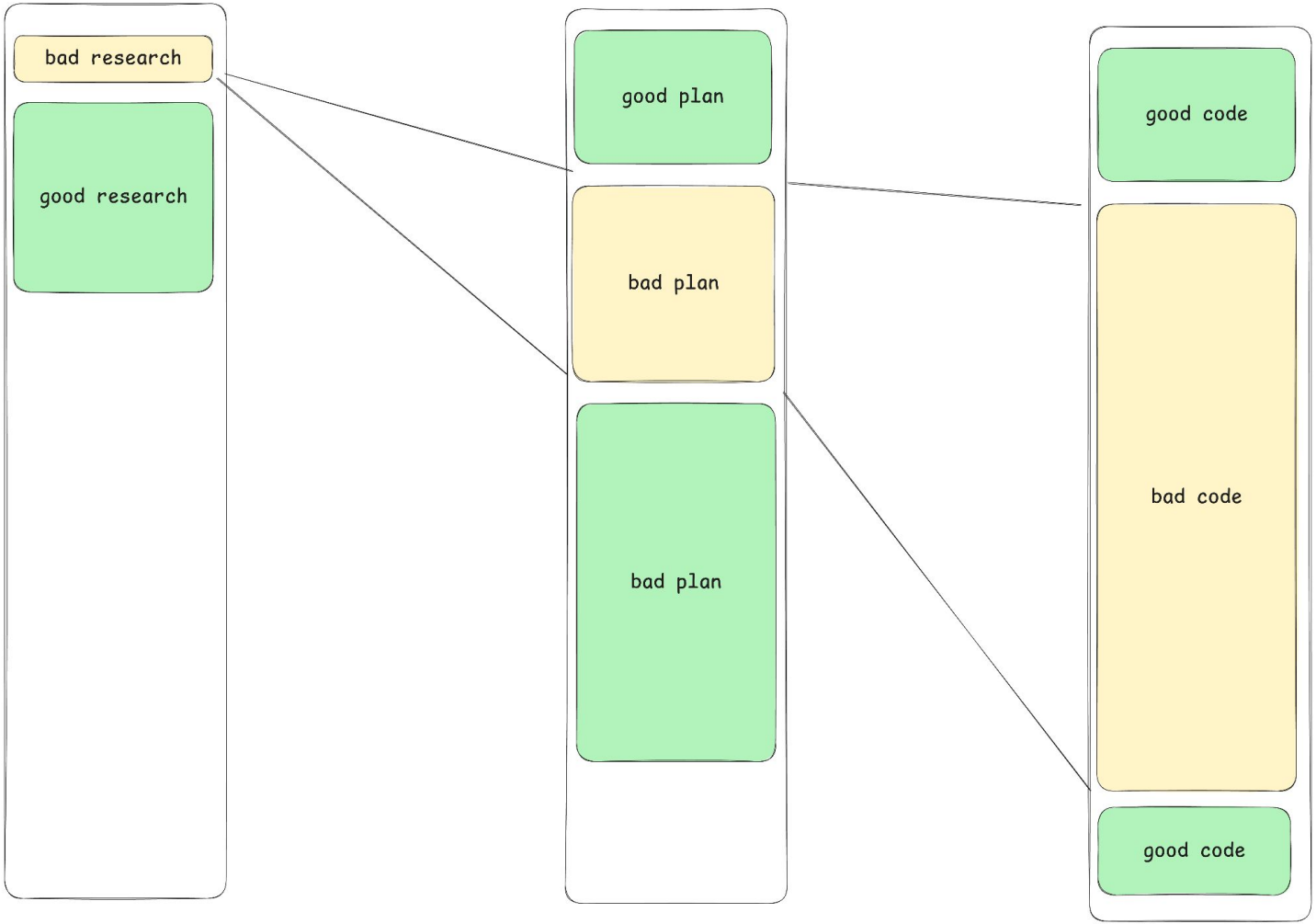
SEP 04, 2025

https://www.arthropod.software/p/vibe-coding-our-way-to-disaster

Goal: Leverage

Hierarchy of Leverage

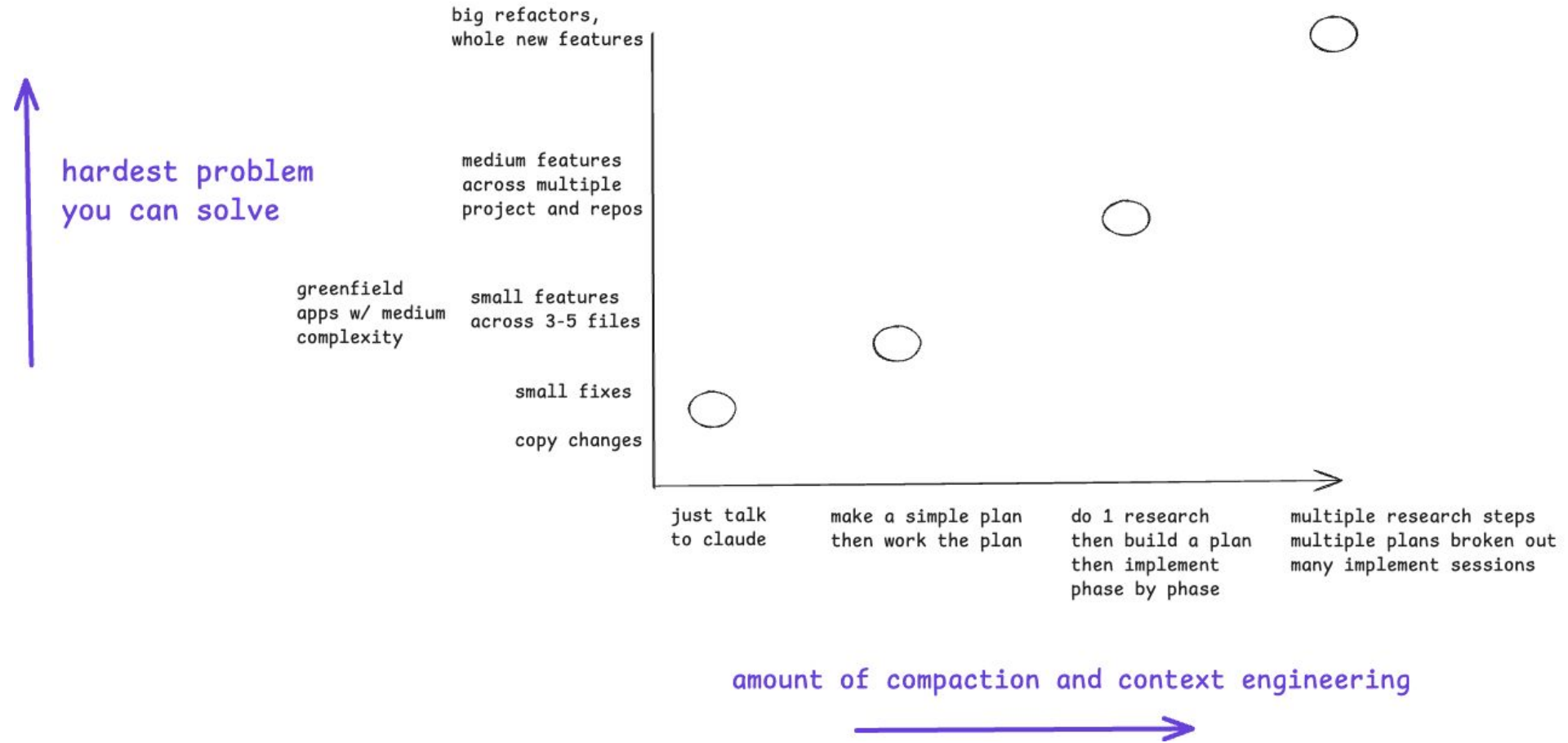



bad research
good research
good plan
bad plan
bad plan
good code
bad code
good code

Sometimes

Research → Plan → Implement

is overkill


hardest problem
you can solve
big refactors,
whole new features
medium features
across multiple
project and repos
greenfield
apps w/ medium
complexity
small features
across 3-5 files
small fixes
copy changes
just talk
to claude
make a simple plan
then work the plan
do 1 research
then build a plan
then implement
phase by phase
multiple research steps
multiple plans broken out
many implement sessions
amount of compaction and context engineering

How much CTXE should you use?

Get your reps in

Pick ONE tool

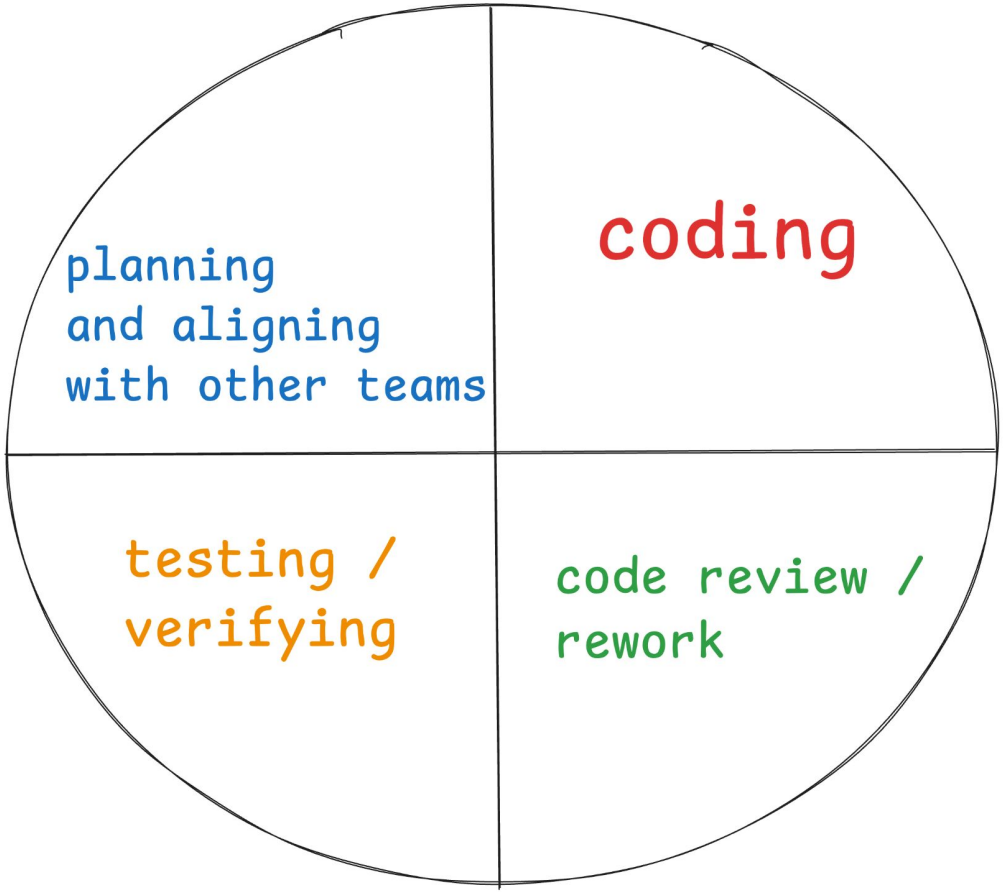
Intuition doesn't come from constantly min-maxing across models and tools

People are calling this

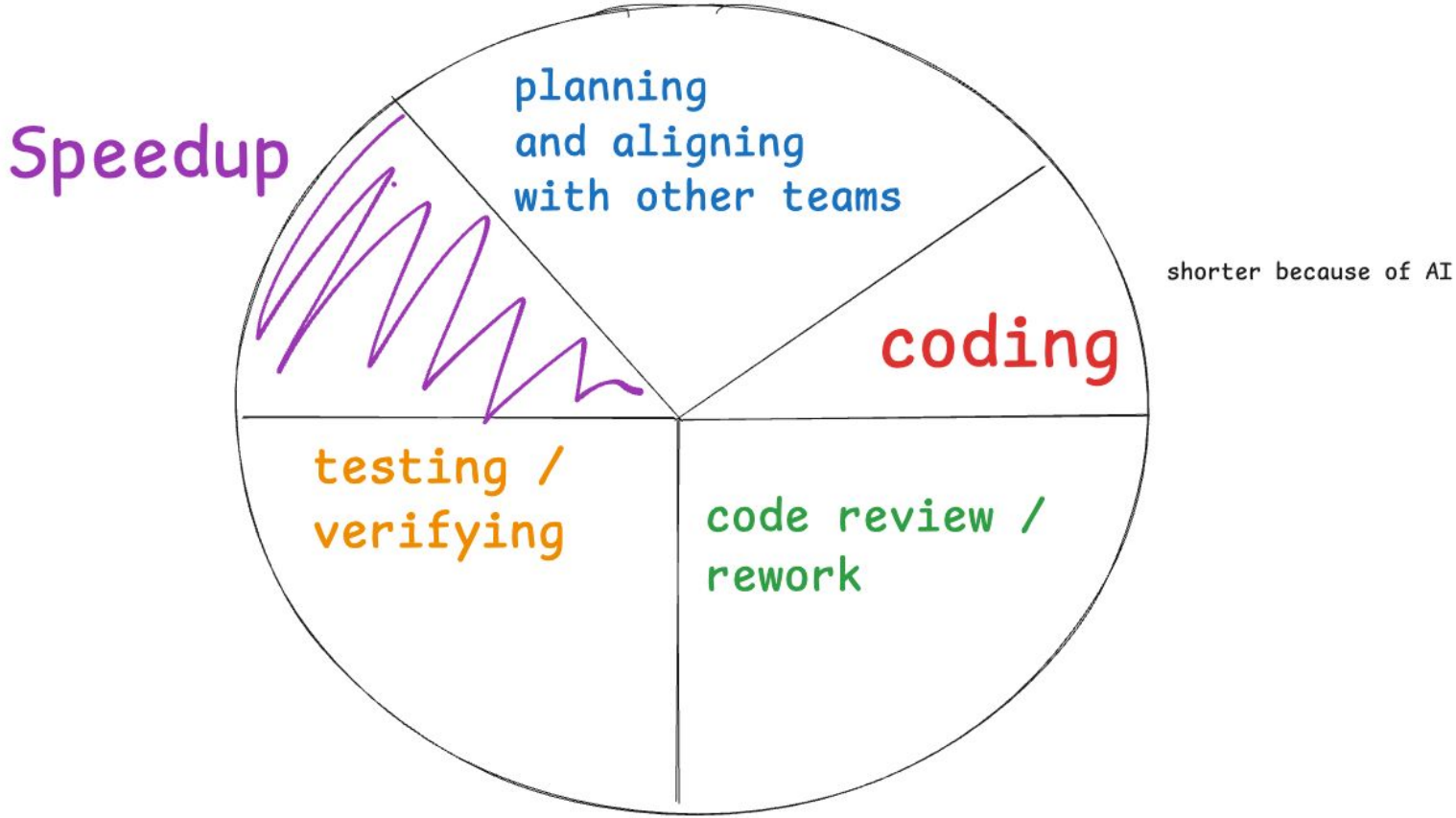
RPI

Goal:
2-3x Productivity Boost

Pre AI:
"It's a two day feature - the coding will take 2-4 hours"

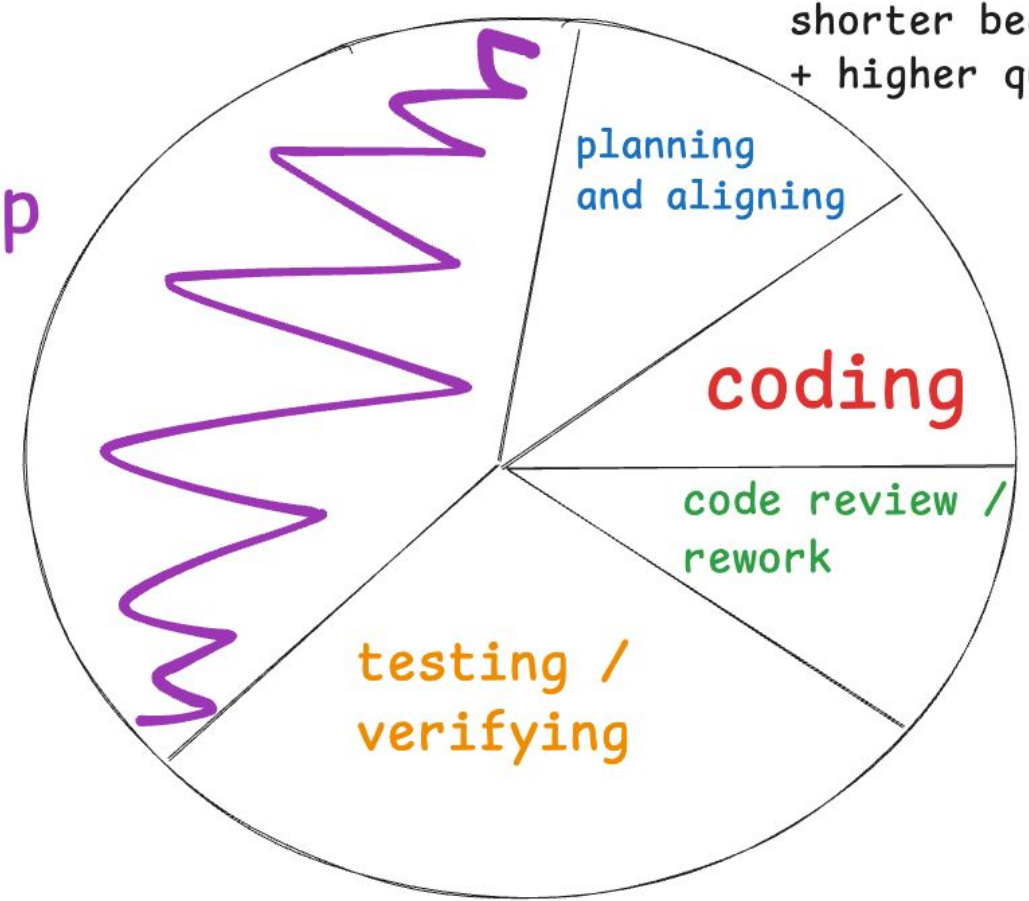


Naive AI:
"It's a two day feature - the coding will take 20 minutes"



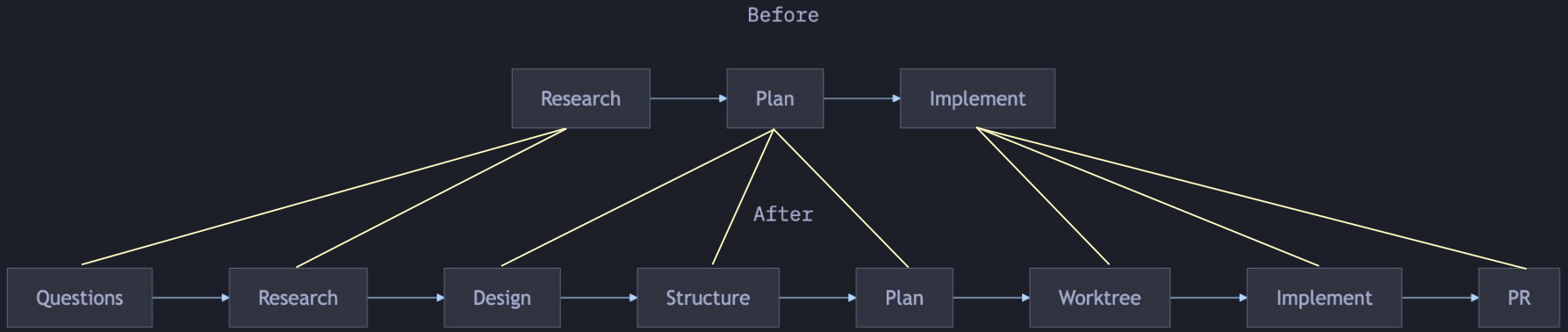
shorter because of AI

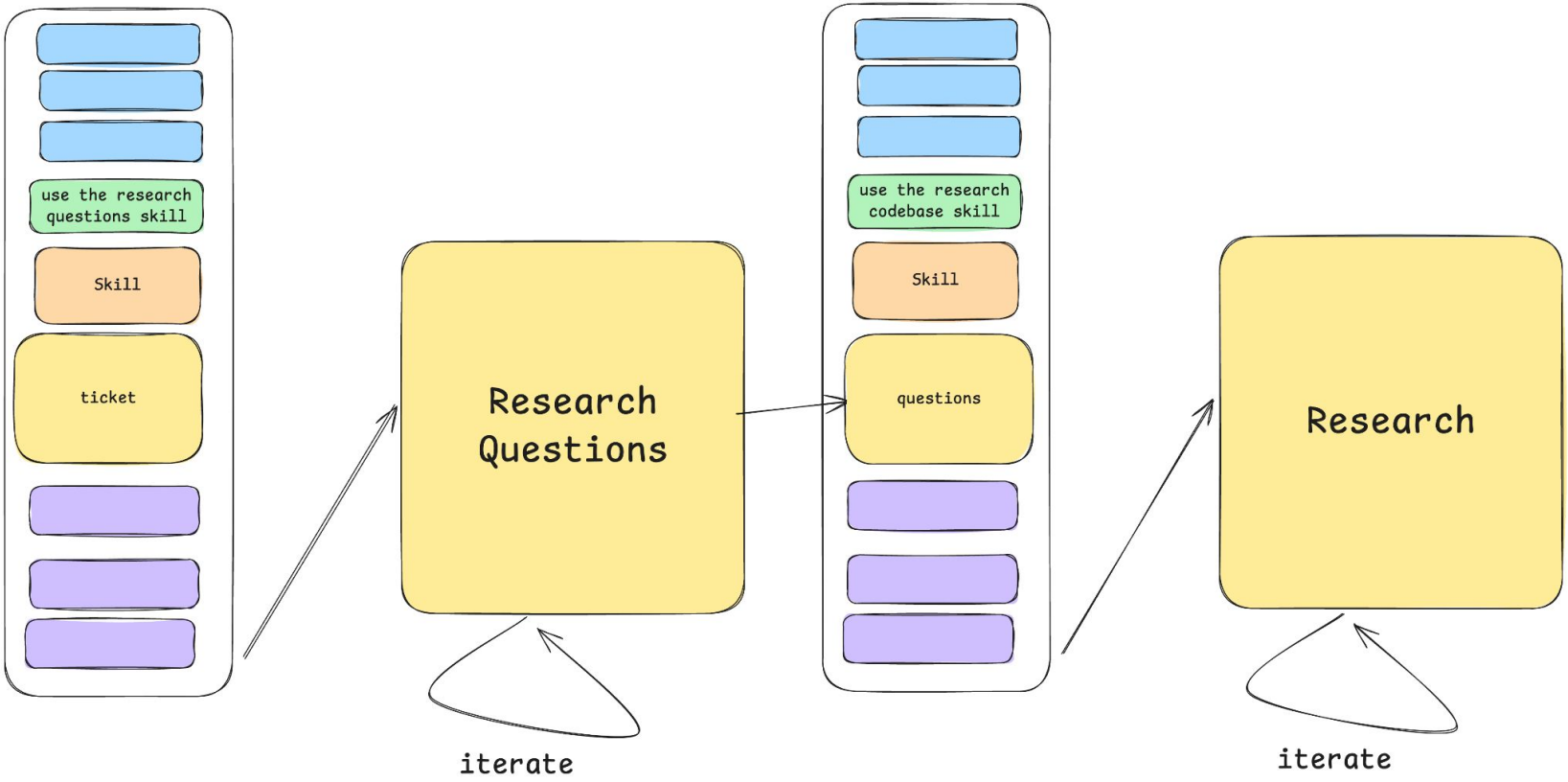
Model-Assisted Planning + Alignment

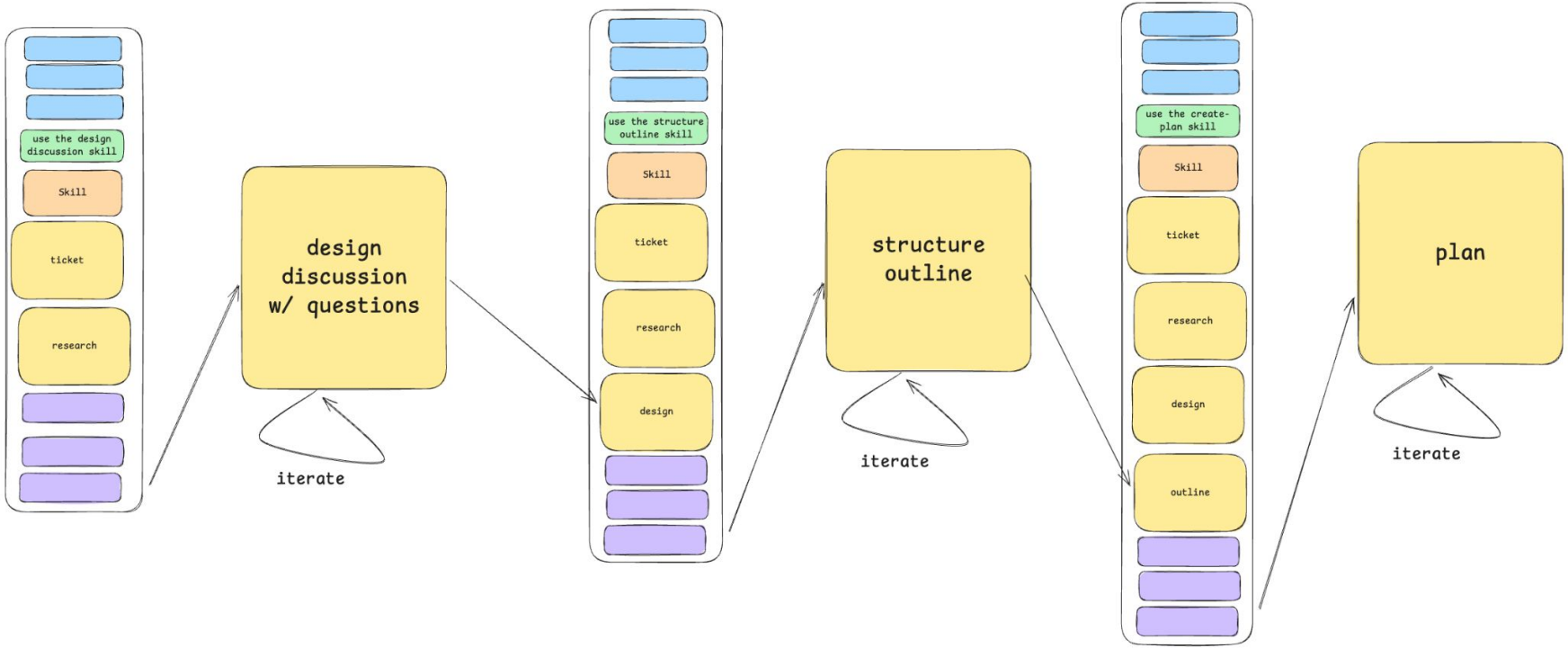


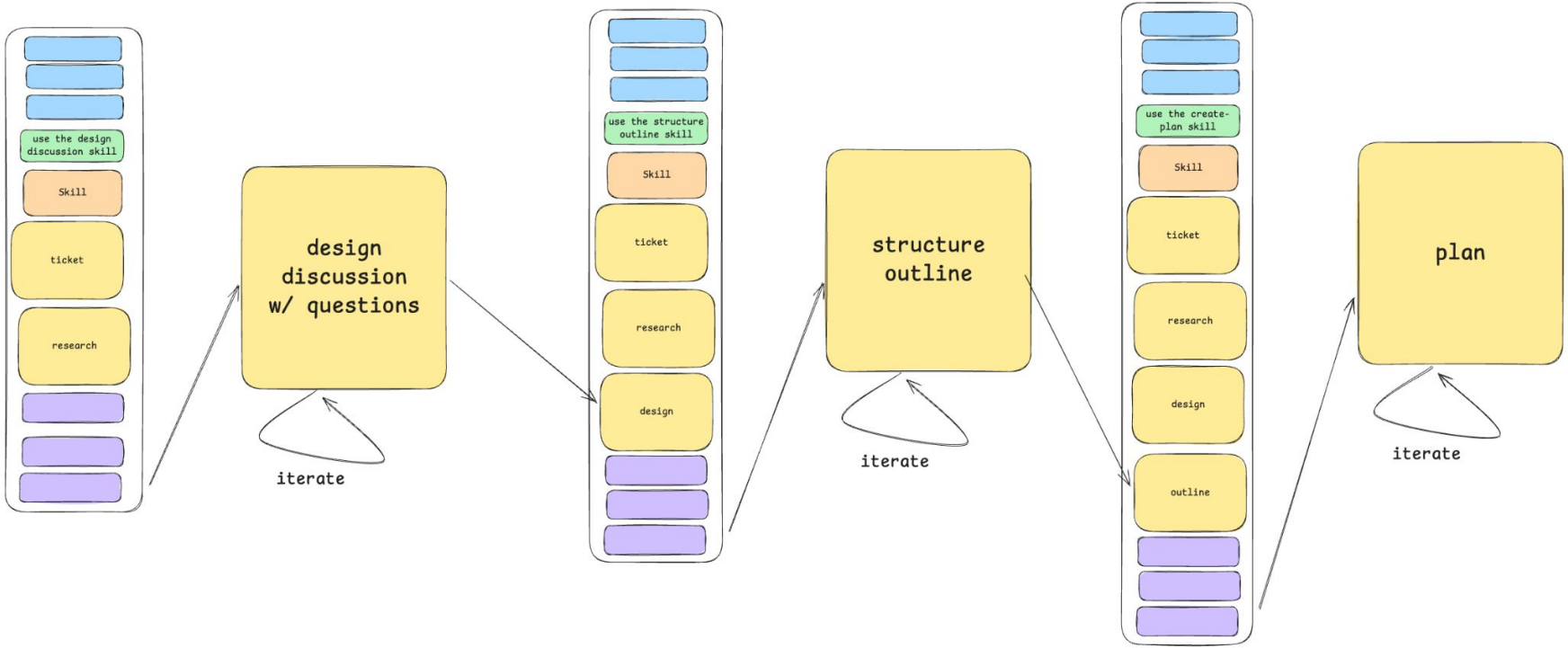
But actually…

Splitting this up

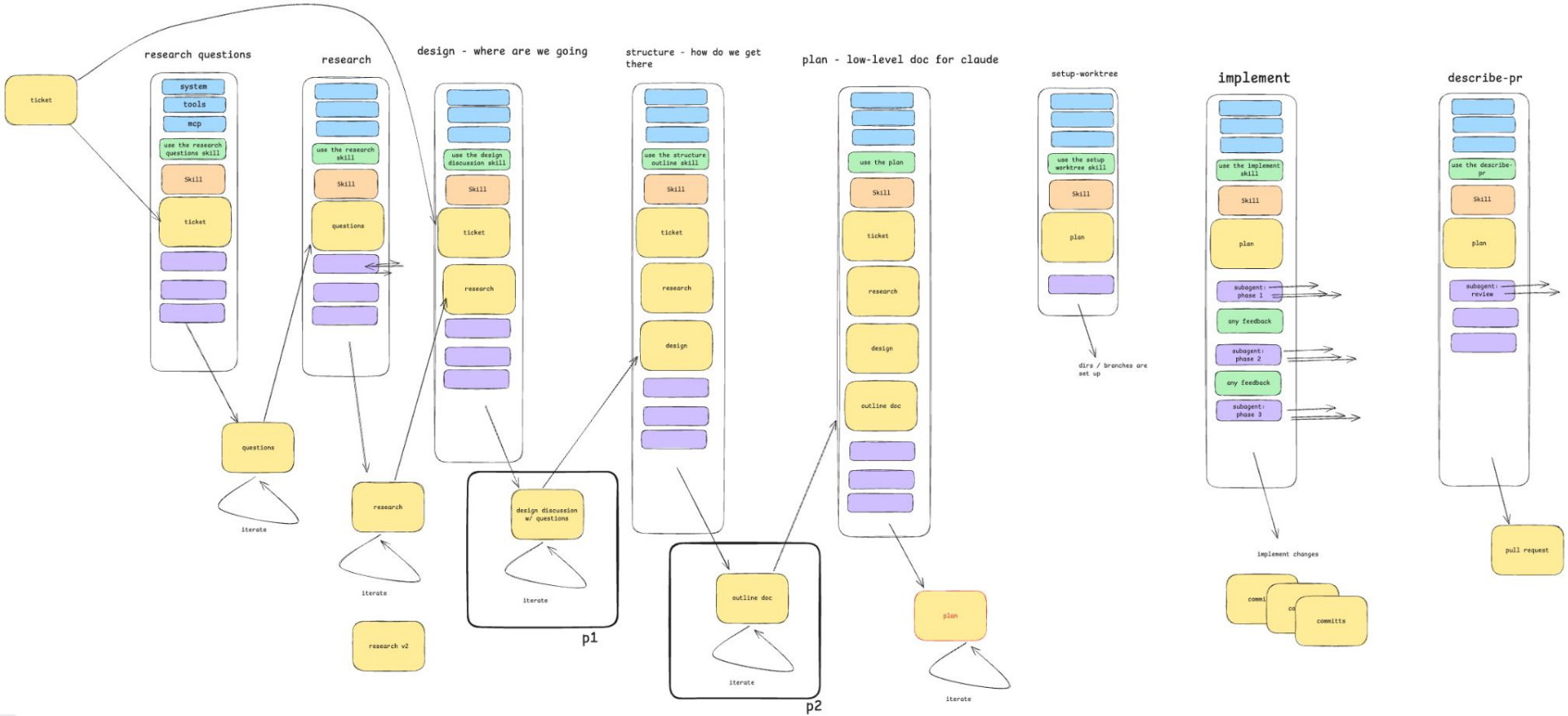



use the research questions skill
Skill
ticket
Research Questions
iterate
use the research codebase skill
Skill
questions
Research
iterate


use the design discussion skill
Skill
ticket
research
design discussion w/ questions
iterate
use the structure outline skill
Skill
ticket
research
design
structure outline
iterate
use the create-plan skill
Skill
ticket
research
design
outline
plan
iterate


use the design discussion skill
Skill
ticket
research
design discussion w/ questions
iterate
use the structure outline skill
Skill
ticket
research
design
structure outline
iterate
use the create-plan skill
Skill
ticket
research
design
outline
plan
iterate

CORE WORKFLOW

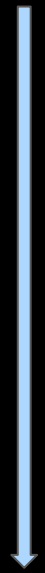


But why… ?

Before: 85+ instructions

After: 38 instructions

The Process

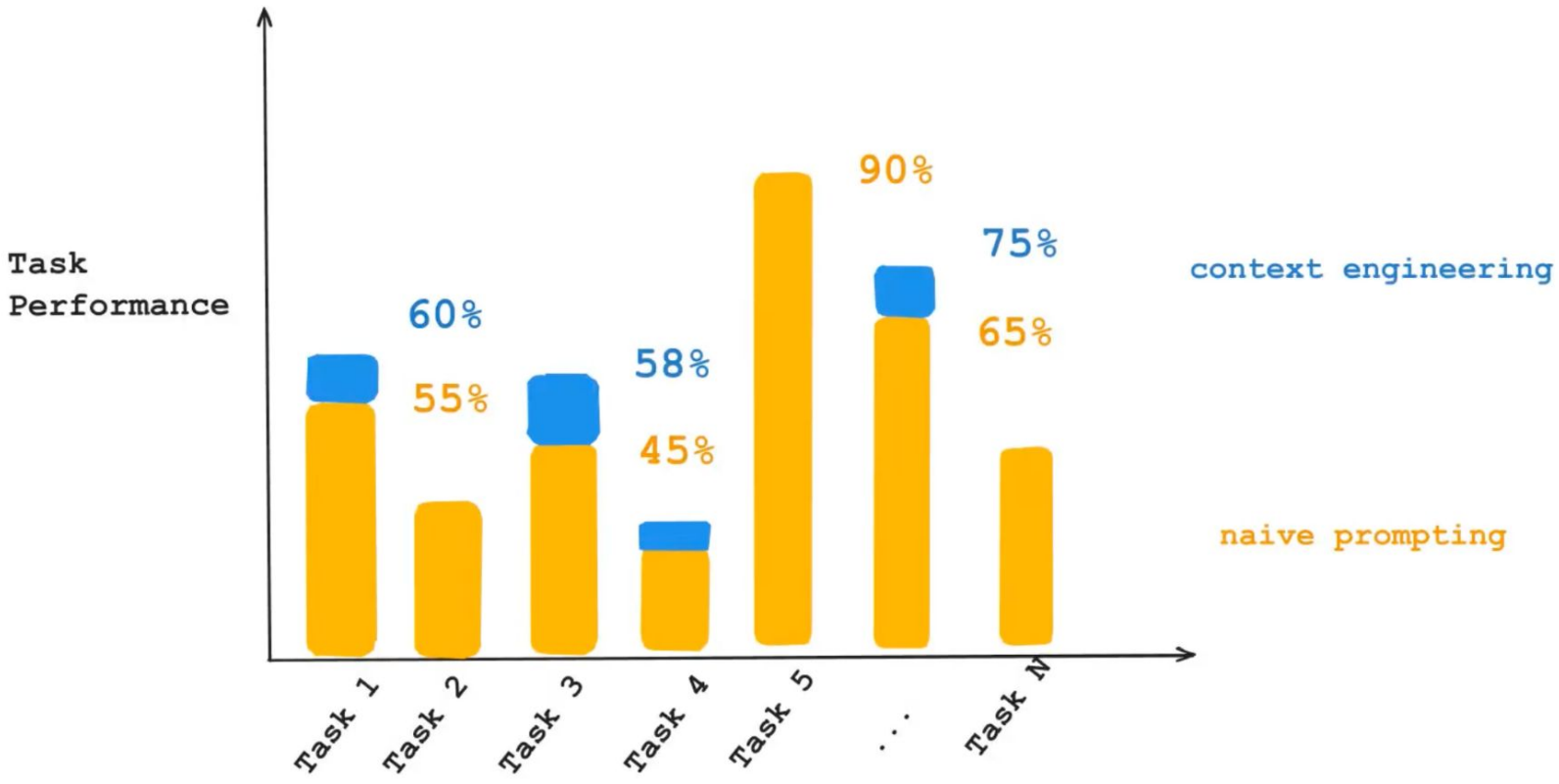


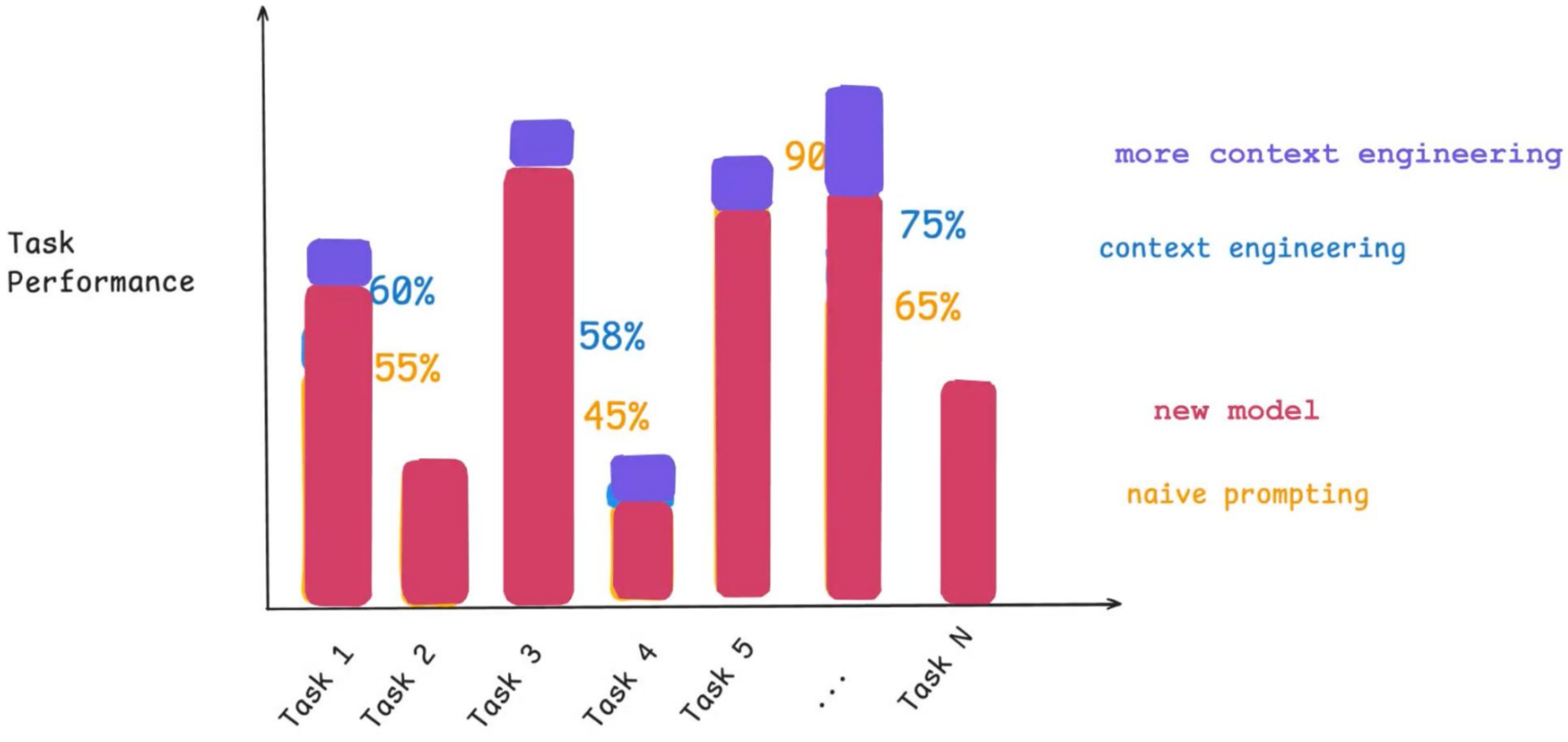
Questions
Research
Design
Structure Outline
Plan
Worktree
Implement
Pull Request

QRSPI

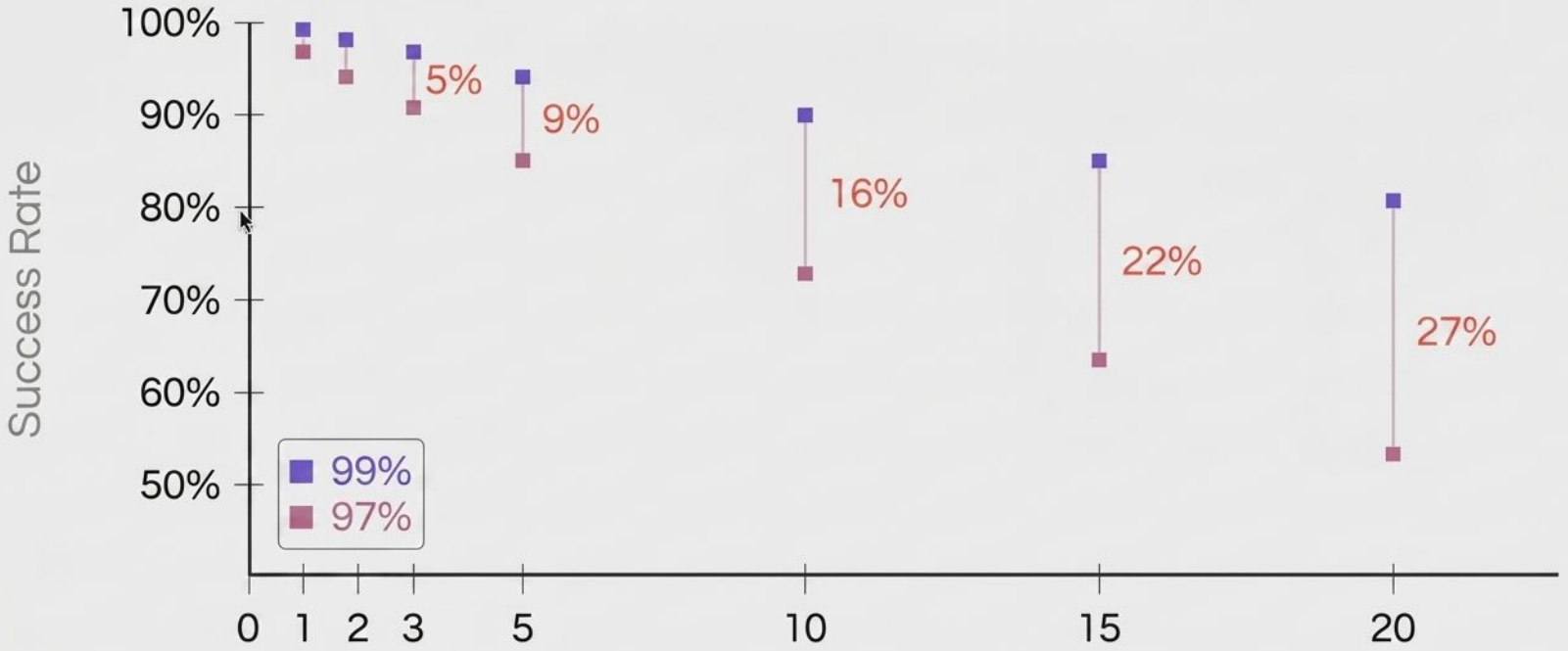
Derivation of QRSPI prompts left as exercise for the reader

won’t this just get
bitter lessoned!?


Task Performance
60%
55%
58%
45%
90%
75%
65%
context engineering
naive prompting
Task 1
Task 2
Task 3
Task 4
Task 5
...
Task N


Task Performance
60%
55%
58%
45%
90
75%
65%
more context engineering
context engineering
new model
naive prompting
Task 1
Task 2
Task 3
Task 4
Task 5
...
Task N

Why every +1% matters

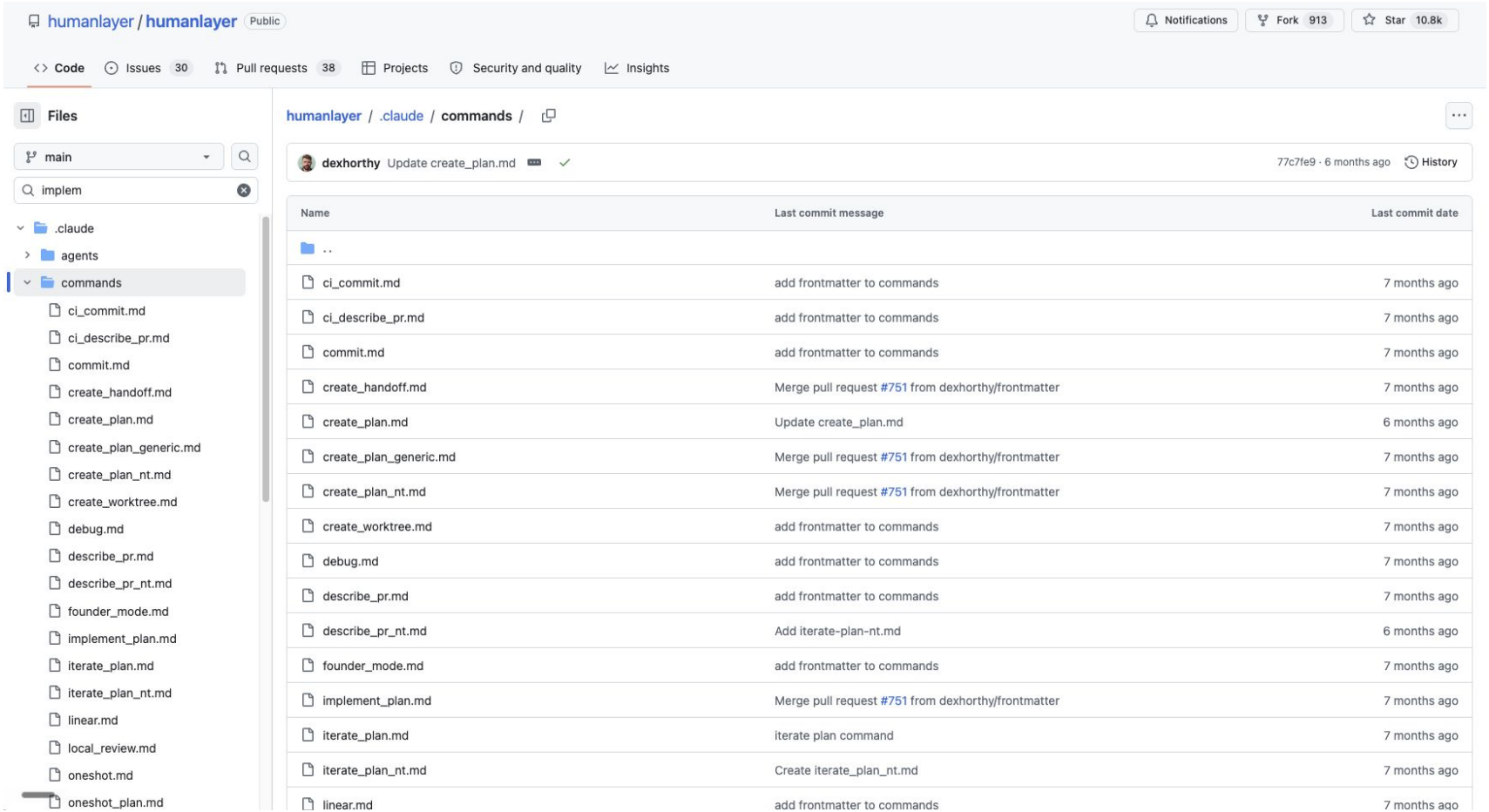


https://boundaryml.com

There is no magic Prompt

Focus on context engineering and instruction budgeting

https://github.com/humanlayer/humanlayer/tree/main/.claude/commands

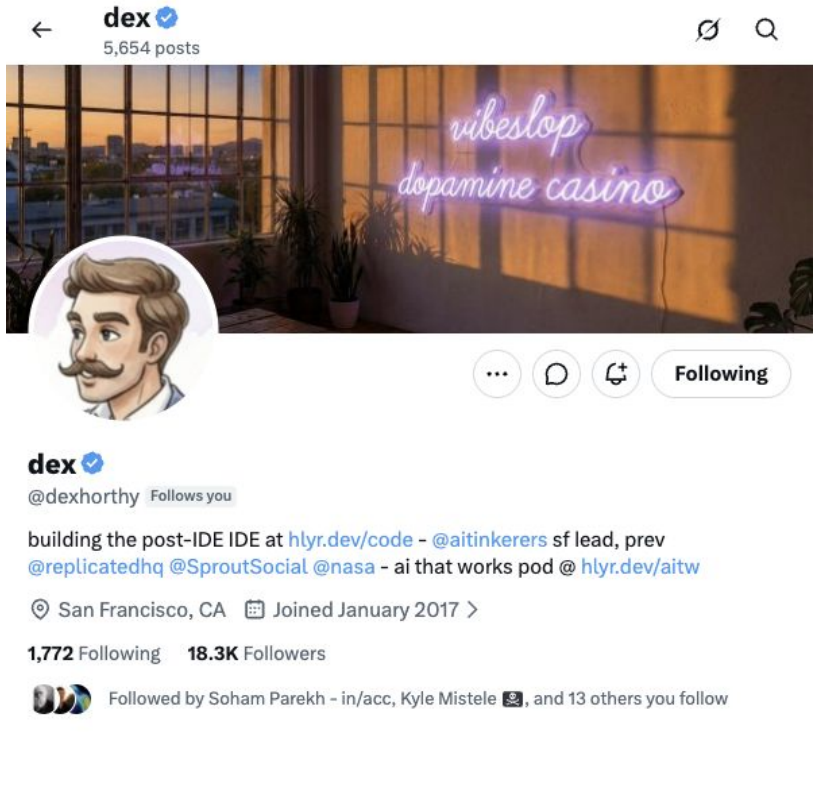


THANK YOU

THANK YOU

(Please read the code)

And **THANK YOU** to HumanLayer



https://www.humanlayer.dev/

Contact info

thomas.w.dietert@gmail.com

https://x.com/tthomasdd

https://www.linkedin.com/in/tdietert

On Agent Swarms…

Bottleneck is your ability to ensure quality

10x speed doesn't matter if its **slop**

Shoot for 2-3x

Good software engineering fundamentals matter now more than ever.

Good software architecture allows engineers and agents alike to easily understand, matain, extend, or modify.

Frontier models are good at coding, but not good at thinking. **You cannot outsource the thinking.**