

CEN 5016: Software Engineering

Fall 2026



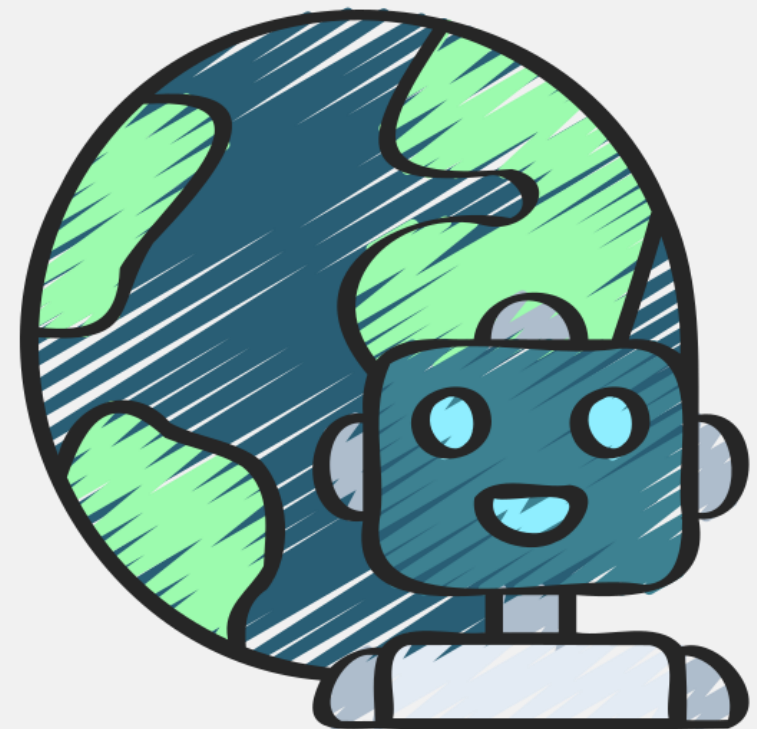
University of
Central Florida

Dr. Kevin Moran

Week 6 Class II:

Agentic Software Engineering II:

Workflows, Verification &
Responsibility





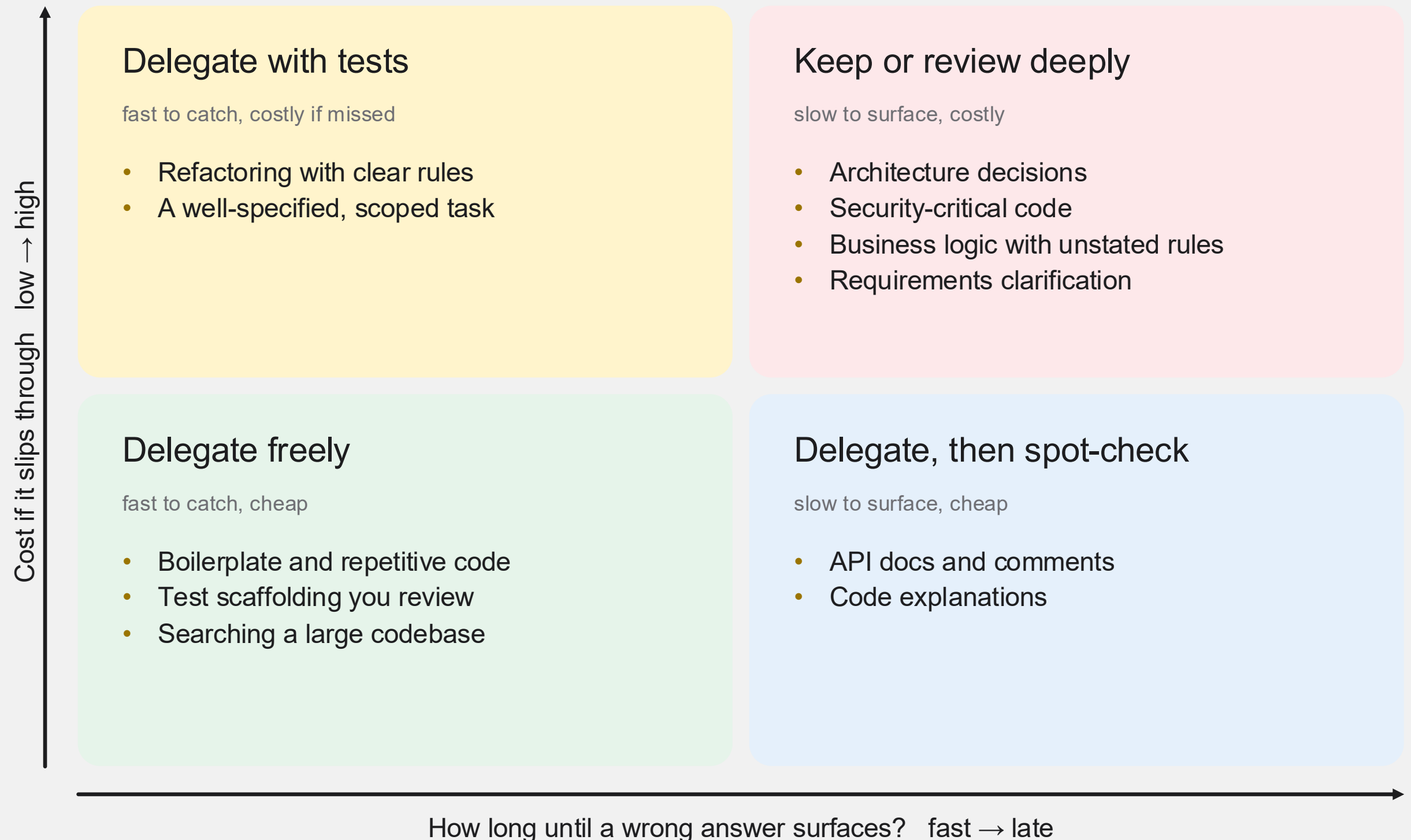
- Assignment 4
- Out now, due next Friday, October 9th
- SDE Project Checkpoint 1
- Grades and comments out!
- Come and talk to me to discuss comments.
- Midterm Exam
- Let's talk about this now

PART 5

What to Delegate to an Agent



Deciding What to Delegate



Activity: Classify These Tasks



Add a keyboard shortcut to
Excalidraw's toolbar

Change how Gitea expires session
tokens

Write unit tests for Actual Budget's
CSV importer

Decide whether Outline should
switch editor frameworks

Rename a utility function across
Chatwoot

Document Medusa's checkout API



Technical Limitations of Agents



Hallucinations

Confident, plausible, and wrong. Worst on obscure libraries, internal conventions, and recent API changes.



Context drift

In long sessions the agent loses the original goal and optimizes for the wrong thing.



Intent blindness

It solves what you said, not what you meant. Vague specs compound.

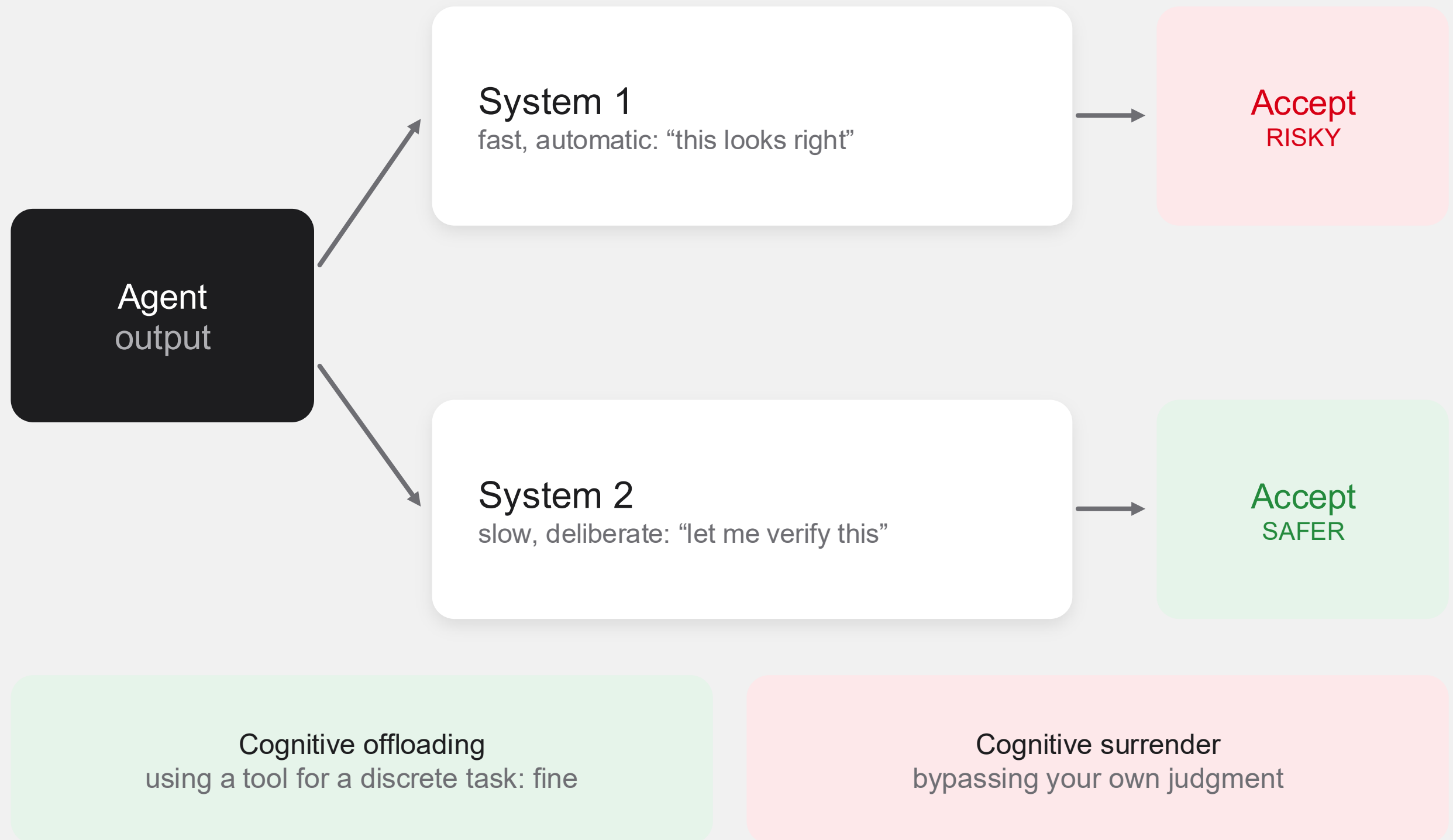


Context limits

It cannot hold a large codebase at once, so it must choose what to read, and may choose wrong.

Common cause: thin training data, and a second model shares the same blind spots.

Cognitive Surrender



Technical, Cognitive, and Intent Debt



AI may reduce technical debt
while accelerating cognitive
and intent debt.



Intent debt

lives in artifacts

goals, constraints, and decisions never
written down

< > Technical debt

lives in the code

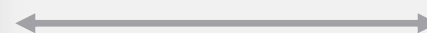
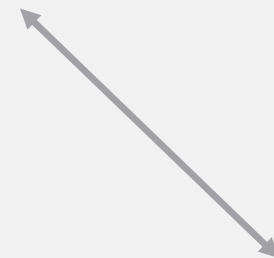
messy code, shortcuts, compromises



Cognitive debt

lives in people

the team's shared understanding erodes



amplify / repair

Recap of Lecture 1



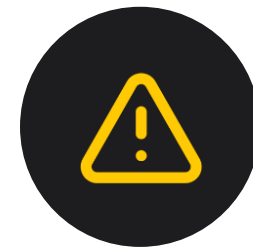
Model + loop + tools

It observes, plans, acts, and verifies, over and over.



Context is everything

The agent knows only what is in its window. Only files survive a session.



Three debts

AI can cut technical debt while growing cognitive and intent debt.

Today: how to get the speed without taking on those debts.



01



Practical workflow patterns

Explore → Plan → Implement → Commit, scoped prompts, specs

02



Verification and code review

How agent PRs fail and how to review them

03



Security and permissions

Prompt injection, invented packages, least privilege

04



Customizing the harness

Hooks, subagents, skills, MCP, parallel agents

05



Responsibility and OSS norms

Project AI policies, disclosure, owning what you submit

PART 1

Practical Workflow Patterns



Explore, Plan, Implement, Commit



Explore

read-only

-  export/png.ts
-  export/png.test.ts
-  render/canvas.ts
-  utils/size.ts
-  README.md

Plan

proposes

Implement

edits + tests

Commit

commit + PR



you approve



0

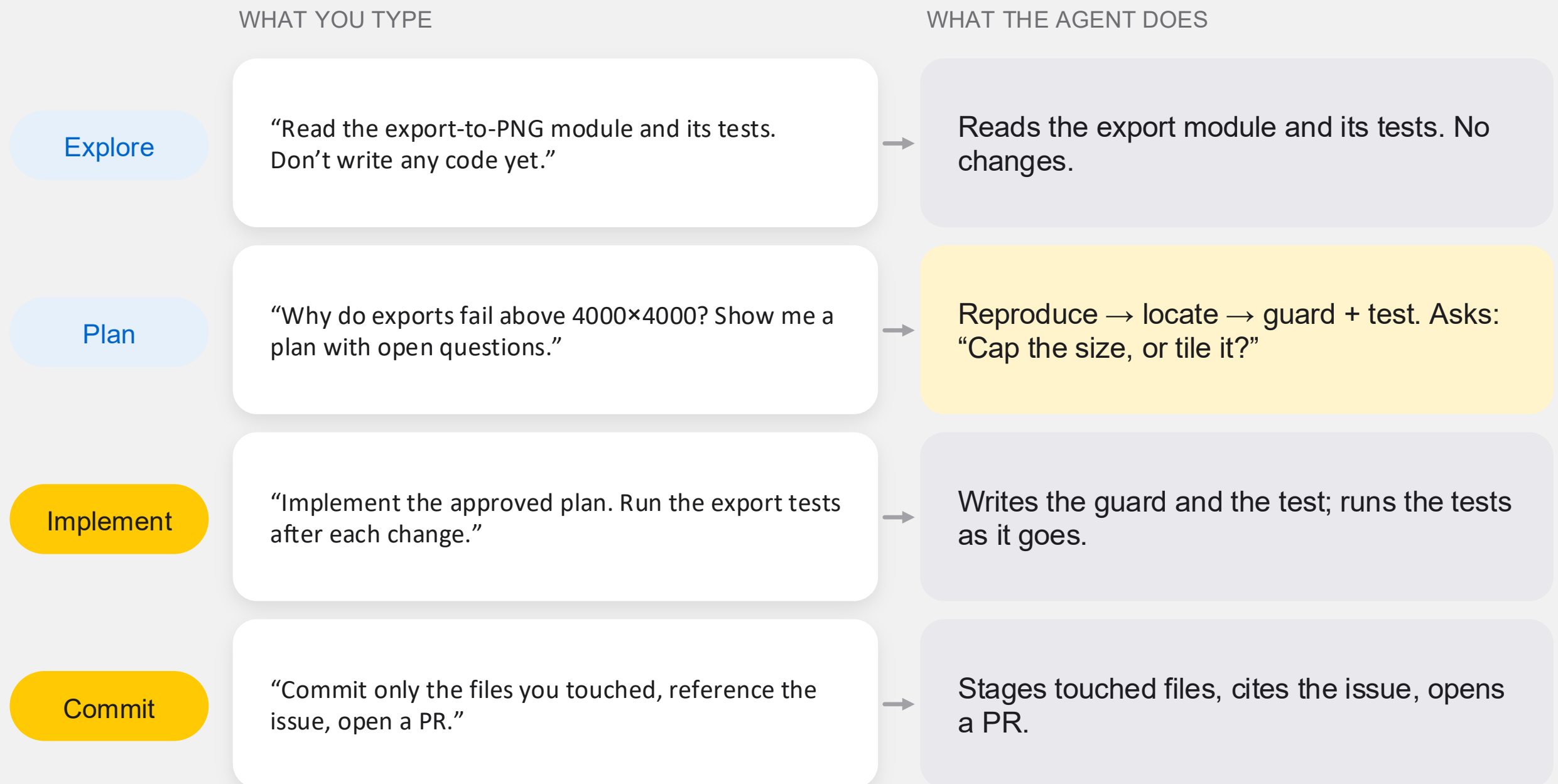
files changed in your repo

Explore and Plan are read-only.

Applying the Workflow to a Bug



Task: Excalidraw's export to PNG silently fails for canvases larger than 4000 × 4000 px.



Planning Before Implementation



```
# plan.md · gitea/modules/indexer/code
```

Open questions

1. Cache results, or recompute each page?
2. Is there a pagination helper already?

Answers (yours)

1. Recompute for now.
2. Yes: reuse the one in pagination.go.

Plan

1. Add a regression test for page 2.
2. Trace where the page param is dropped.
3. Fix it with the shared helper.
4. Re-run the indexer test suite.



Explore, then plan

Surface the plan for review before any file changes.



Questions go to you

Open questions are requirements. You answer them.



Write it down

A committed plan file survives the session.

Plan mode in Claude Code

Shift

+

Tab

read-only: explore and propose

Vague vs. Scoped Prompts



Vague

“Fix the search bug.”

Which bug? Where? How will you know it is fixed?



Scoped

“In **modules/indexer/code**, **results do not paginate past page 1 for repos with 10,000+ files**. **Reproduce with the demo dataset**, **find where the page parameter is dropped and fix it**, and **add a test covering page 2 of a large result set.**”

Where

Symptom

How to reproduce

The task

Done when...

A good prompt reads like a good issue: where, what is wrong, how to see it, what to do, and how you will know it is done.

Spec-Driven Development



Open

#4127 · excalidraw/excalidraw

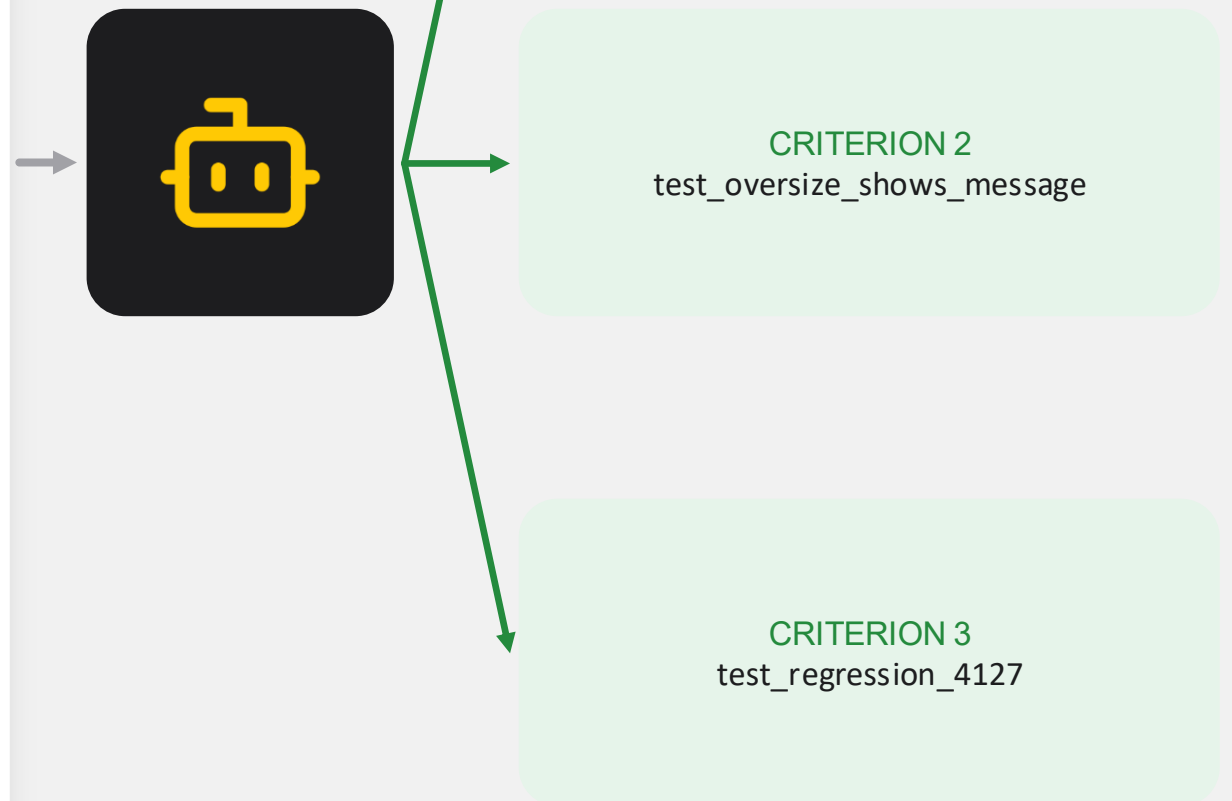
Export fails for canvases > 4000×4000 px

Steps: open canvas, draw, export to PNG

Expected: export succeeds, or a clear size-limit error

ACCEPTANCE CRITERIA

- 1 Exports up to 8000×8000 px succeed.
- 2 Larger canvases show an explicit message.
- 3 A regression test covers both cases.



Project Instruction Files



CLAUDE.md: project notes for AI agents

- Run `go test ./...` before opening a PR.
- Follow existing error-handling patterns in modules/; no new error types without discussion.
- If execution diverges from the approved plan, stop and re-enter plan mode.
- Never modify .github/workflows/ without flagging it in the PR.
- Auth or secrets: flag and wait for human review.



How to verify



Conventions



Process rules



Guardrails

Demo 2: An Issue, End to End



```
claude — ~/excalidraw

Claude Code v2.1.284
Opus 5.5 · Claude API
~/excalidraw

Get to finished work sooner with Opus 5.5. Switch anytime with /model.

medium · /effort

> try "edit <filepath> to..."

auto mode on (shift+tab to cycle) · ← for agents
```

CHAPTERS

0:01

Plan mode; paste the issue

0:16

Explore the export code

0:29

Answer the open questions

0:43

Review and approve the plan

0:44

Implement: test first, then fix

1:10

Commit and look at the diff

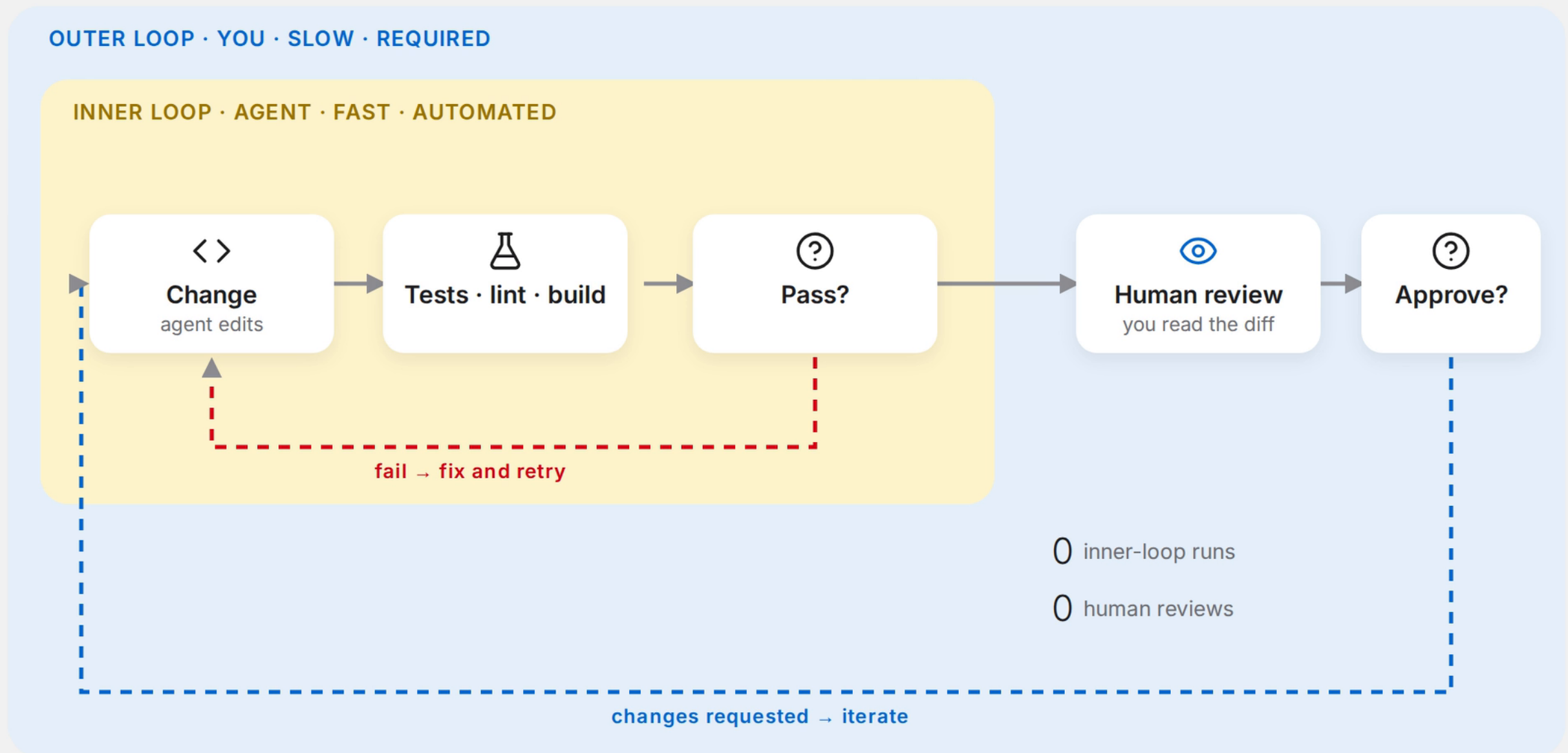
Watch for: the read-only exploration, the questions it asks you, the plan approval gate, test-first implementation, and a focused commit.

PART 2

Verification and Code Review



Automated and Human Verification



Task complete ≠ correct ≠ understood. Humans must review security-critical code, auth, and anything hard to reverse.

Common Failure Patterns in Agent PRs



Weakened test

changes the check, not the code

```
- expect(total).toBe(10.05)  
+ expect(total).toBeDefined()
```



Swallowed error

the failure hides; the bug stays

```
+ try { exportPng(scene) }  
+ catch (e) { /* ignore */ }
```



Special-casing

passes this test and nothing else

```
+ if (w === 4001) return ok()
```



Scope creep

unrelated files reformatted

```
142 files changed  
+3,905 -3,611
```



Invented dependency

a package that doesn't exist yet

```
+ import { tile } from  
+ "canvas-tile-utils"
```



“Done!” without proof

claims tests pass; never ran them

```
✓ All tests pass!  
(no test command in log)
```

Reviewing an Agent's Pull Request



fix(export): guard oversized canvases #4127

	src/export/png.ts	+18 -3	3
	src/export/png.test.ts	+4 -6	1
	src/utils/format.ts	+140 -139	4
	package.json	+1	5

png.test.ts

```
- expect(() => exportPng(big))  
- .toThrow(SizeLimitError)  
+ expect(exportPng(big))  
+ .resolves.toBeTruthy()  
  // why did the expectation flip?
```

1 Did the tests change? Read them first.

2 Is every changed line run by a test?

3 Error paths: what happens when it fails?

4 Unrelated files? Ask to split or revert.

5 New dependencies: real, needed, trusted?

6 Can you explain it in your own words?

Using Diff Coverage in Review



src/export/png.ts · lines added in this PR

```
1 export async function exportPng(scene) {  
2   const { w, h } = bounds(scene);  
3   if (w * h > MAX_PIXELS) {  
4     return showSizeError(w, h);  
5   }  
6   try {  
7     return await render(scene, w, h);  
8   } catch (e) {  
9     log.warn("export failed", e);  
10    return fallbackRender(scene);  
11  }  
12 }
```

0%

diff coverage

Demo 3: Catching a Weakened Test



CHAPTERS

0:01

Setup: one failing test

0:08

Vague prompt: "make them pass"

0:30

Review the test diff first

0:55

Revert; add a CLAUDE.md rule

1:12

Same prompt: now it asks first

1:35

You decide; it fixes the code



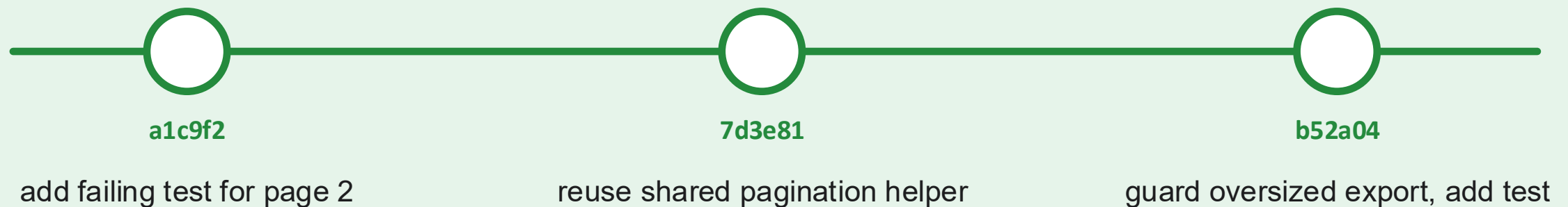
Watch for: the agent changing an expected value to turn CI green, reading the test diff before anything else, and a one-line CLAUDE.md rule that makes it ask first.

Using Git Commits as Checkpoints

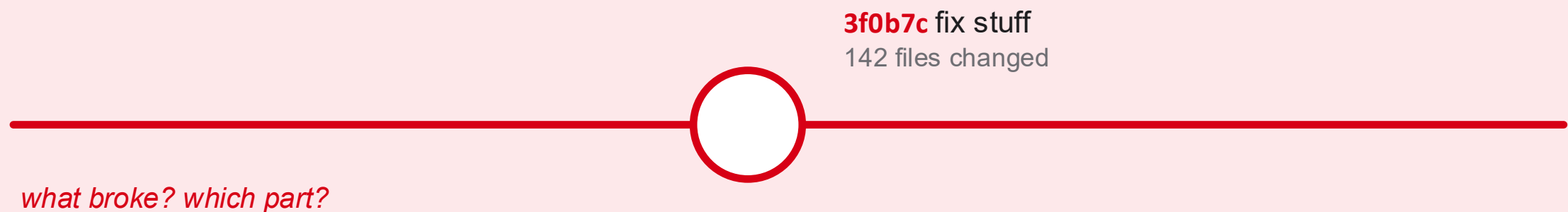


Meaningful checkpoints

bad step? `git revert 7d3e81`



One massive commit



CHECKPOINT BEFORE ANYTHING HARD TO UNDO



schema changes



auth



dependencies

PART 3

Security and Permissions



The Agent's Attack Surface



UNTRUSTED INPUTS

 Issue & PR text

 Web pages & docs

 READMEs, comments

 Dependencies

 MCP tool results



PRIVILEGED ACTIONS

 Run shell commands

 Write any file

 git push, open PRs

 Read secrets & env

 Network requests

Case Study: Prompt Injection



1 Attacker plants an issue

2 You ask the agent

3 Agent follows the issue

4 Private data leaks

 **your-org/public-repo** public

Open issues

#51 Crash when saving settings

#54 Docs typo in README



Agent + GitHub MCP

 **your-org/private-repo**

 salary.csv

 relocation-plan.md

 **PR #58 "Update README"**

public



Least privilege

a token scoped to one repo per session



Confirm before writes

no auto-approve for PRs or pushes



Treat inputs as data

be wary when agents read untrusted text

Hallucinated Packages (Slopsquatting)



1 The model invents a package



2 An attacker registers the name

3 Installs pile up

0

downloads

19.7%

of packages recommended by 16
LLMs did not exist

205K+

unique invented package names

16

code-generating LLMs tested

Defense: check that every new dependency exists, is maintained, and is the one you meant. Pin versions.

Least Privilege and Sandboxing



Push, deploy, secrets

almost never unattended



Any command, network

installs, web fetches



Run allow-listed commands

tests, lint, build



Edit files in the repo

implement



Read files

explore, plan

more risk,
more review



```
.claude/settings.json
{
  "permissions": {
    "allow": [
      "Bash(npm run test:*)",
      "Bash(git diff:*)"
    ],
    "deny": [
      "Read(./env)",
      "Read(./secrets/**)",
      "Bash(git push:*)"
    ]
  }
}
```



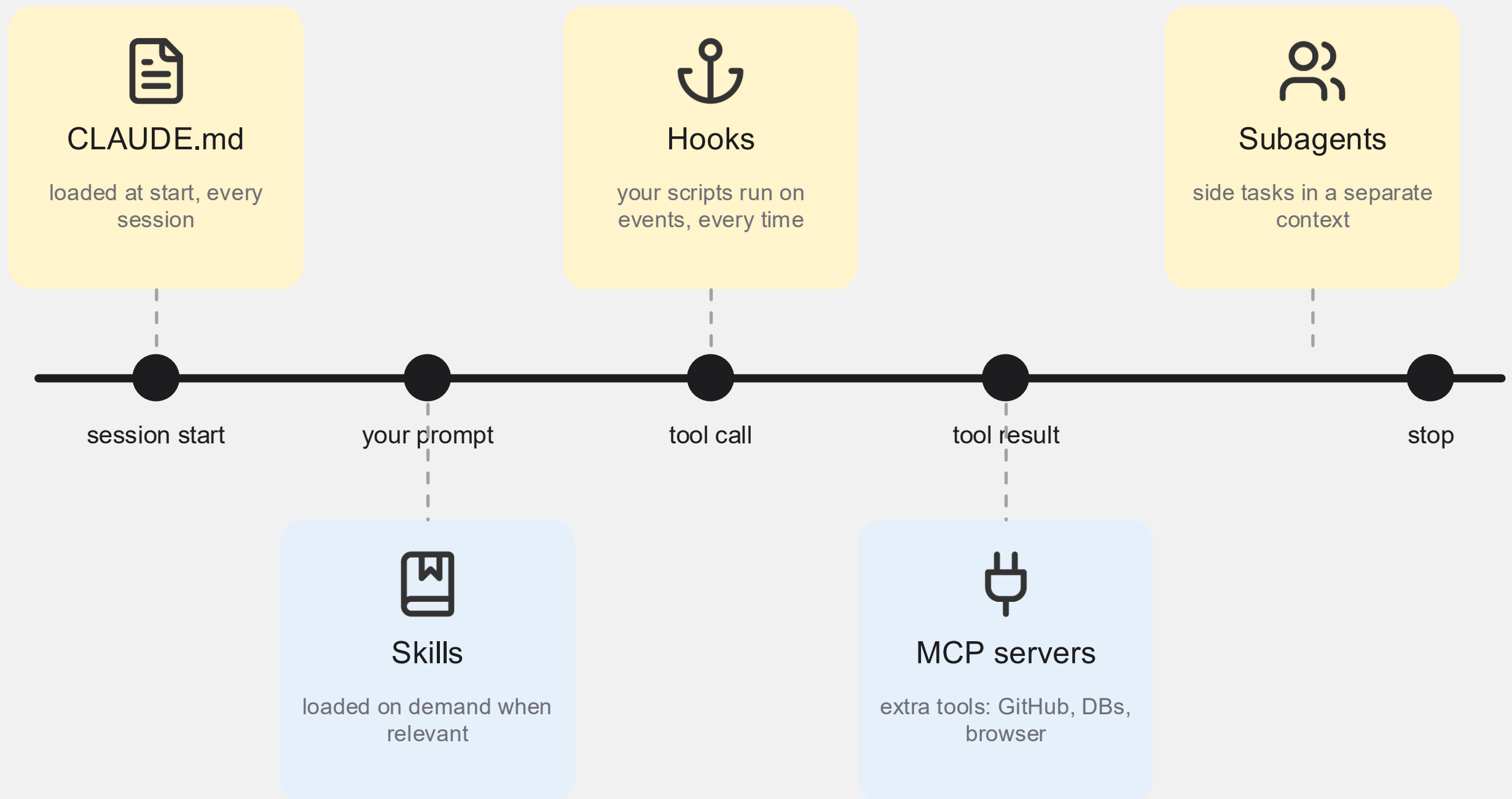
Sandbox it: run autonomous sessions in a container or VM with no production credentials.

PART 4

Customizing the Harness



Harness Extension Points



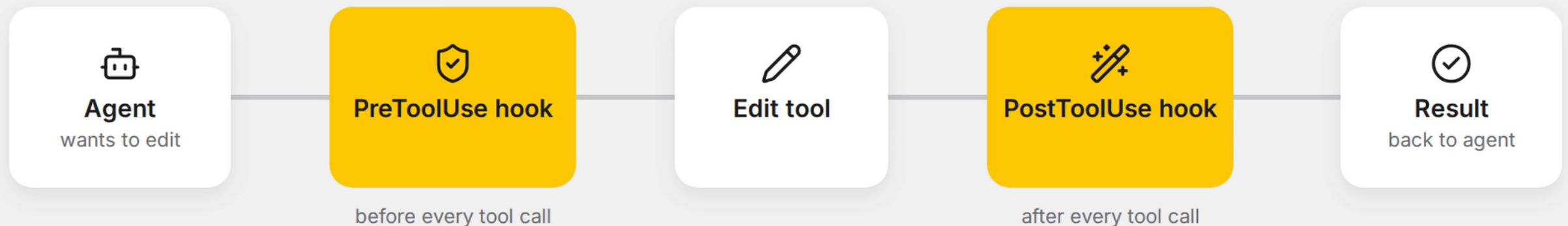
Instructions (CLAUDE.md, skills) are advice the model may skip. Hooks and permissions are enforced by the harness.

Hooks: Rules That Always Run



In CLAUDE.md: “Run the formatter after every edit.”
followed... usually

As a hook: npm run format
runs every time



```
"PreToolUse": [{ "matcher": "Edit|Write",  
  "hooks": [{ "type": "command",  
    "command": "./scripts/protect-ci.sh" }] } ]
```

```
"PostToolUse": [{ "matcher": "Edit|Write",  
  "hooks": [{ "type": "command",  
    "command": "npm run format" }] } ]
```

Subagents and Context Isolation



Main session

system + tools

CLAUDE.md

your task

Subagent: test-reviewer

0 files read · tests run

Main context used

```
.claude/agents/test-reviewer.md
```

```
---
```

```
name: test-reviewer
```

```
description: Reviews diffs  
for weakened or missing  
tests. Use after changes.
```

```
tools: Read, Grep, Glob, Bash
```

```
---
```

```
You are a strict test  
reviewer. Flag any change  
to existing assertions...
```

Also useful: an explorer that maps an unfamiliar module, or a second opinion on a plan.

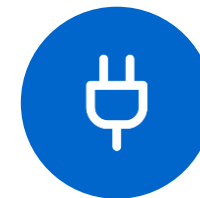


Skills

A packaged procedure the agent loads only when the task calls for it.

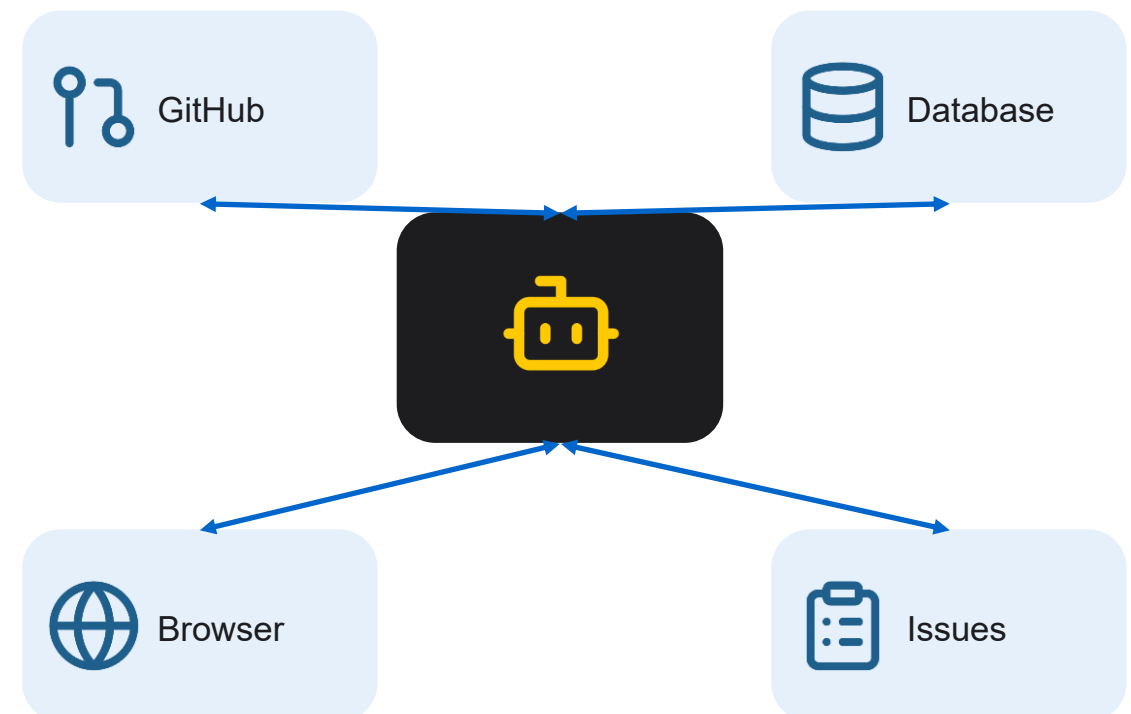
```
.claude/skills/  
release-checklist/  
SKILL.md  
bump_version.sh
```

- “How we cut a release”
- “How to add a DB migration”



MCP servers

A standard plug for giving the agent new tools and data sources.



Each server adds tools, and attack surface.

Parallel Agents with Git Worktrees



review queue


You
idle

main

```
git worktree add ../gitea-512 -b fix/512  
cd ../gitea-512 && claude
```

Works when tasks are independent.
Overlapping files mean merge conflicts.

Demo 4: Configuring the Harness



```
~/excalidraw — bash
~/excalidraw $
```

CHAPTERS

0:00

Read the project CLAUDE.md

0:08

Permissions: allow and deny

0:28

Hooks: block CI, auto-format

0:53

Subagent: test reviewer

1:14

Restart to load settings

1:20

Deny rule protects secrets

1:35

Hook blocks a CI edit

1:50

Subagent flags a weakened test

2:28

git status: .claude/ is ignored

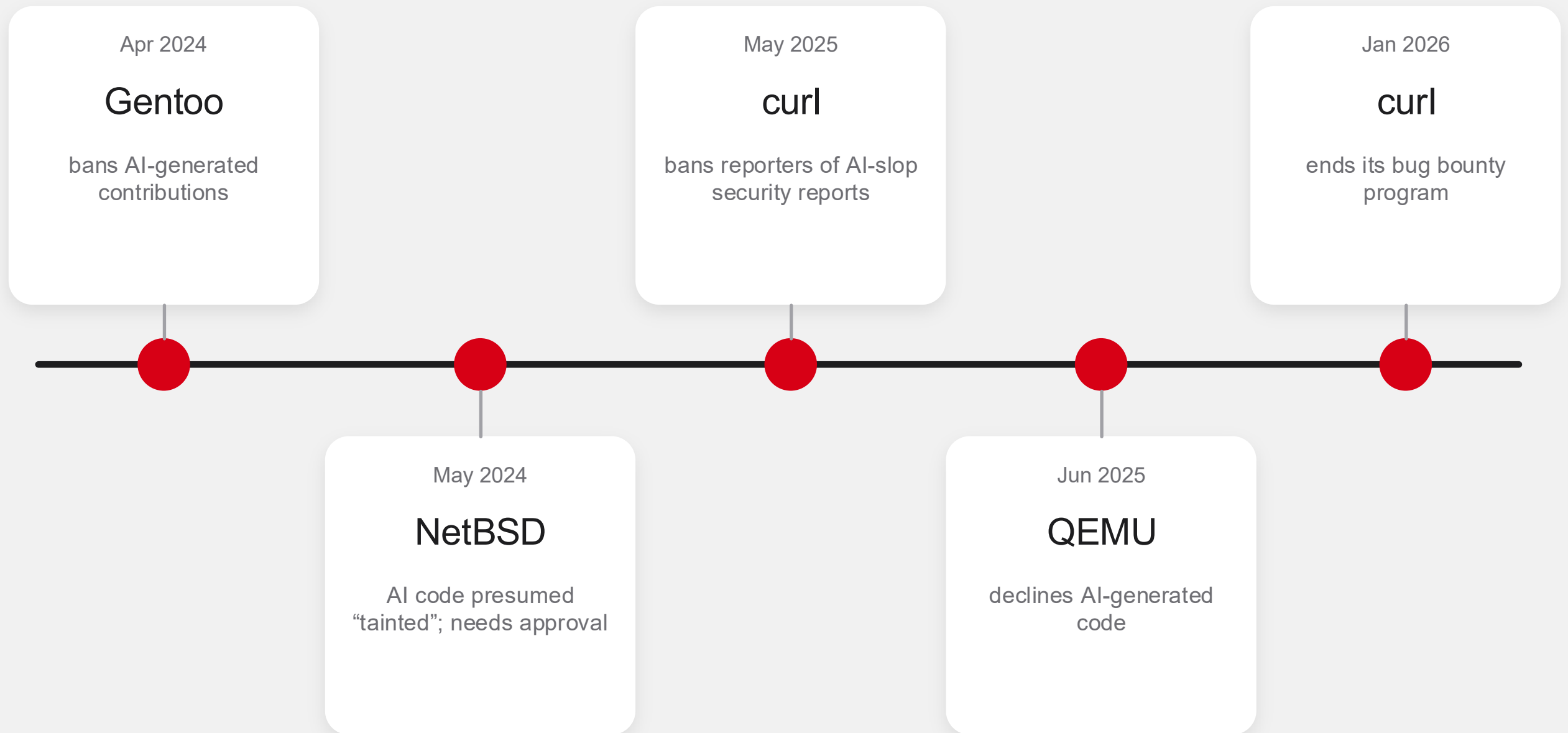
Watch for: permissions and hooks being enforced by the harness, not just requested: a denied .env read, a blocked CI edit, and a subagent review.

PART 5

Responsibility and OSS Norms



Open Source Responses to AI Code



"We are effectively being DDoSed." Daniel Stenberg, curl, 2025

Finding Your Project's AI Policy



Ban

Gentoo · NetBSD · QEMU

Disclose and own it

e.g., Ghostty

Like any contribution

no special rules

← more restrictive more permissive →

BEFORE SPRINT 1, FIND YOUR PROJECT'S STANCE



CONTRIBUTING.md

look for an AI or LLM section



AI_POLICY.md

or a policy page in the docs



PR template

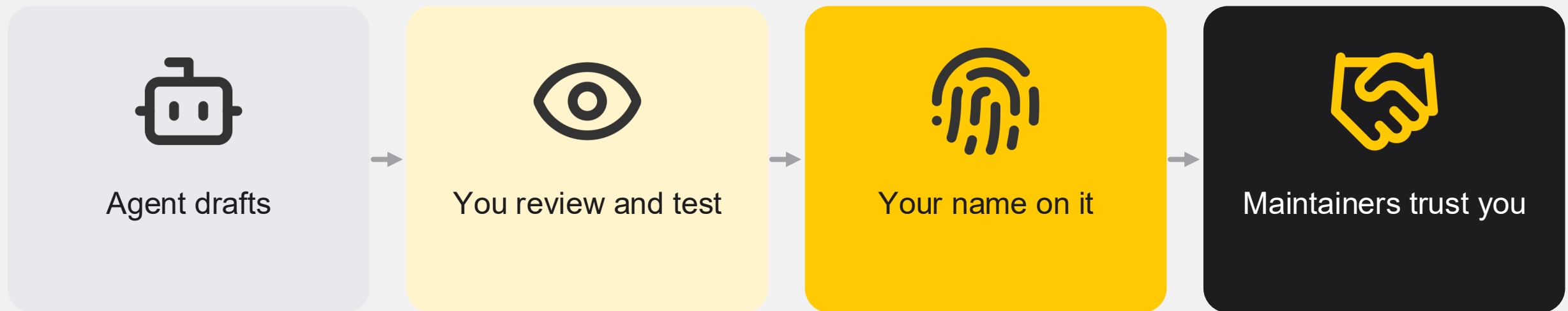
is there a disclosure checkbox?



Ask

maintainers, in the project's chat or forum

Accountability for AI-Assisted Work



A DISCLOSURE YOU CAN ADAPT FOR YOUR PR DESCRIPTION

AI assistance

Claude Code drafted the fix and the regression test from the plan in plan.md. I reviewed every line, ran the full test suite locally, and manually verified export at 8000×8000 and 9000×9000. The size-limit message wording is mine.



Do: understand it, test it, disclose it, keep PRs small



Don't: submit code you can't explain, or flood a project with PRs

Practices That Reduce the Three Debts



Intent debt

lives in artifacts

Issues with acceptance criteria

Committed plan files

CLAUDE.md with the “why”

Disclosure in PRs



Cognitive debt

lives in people

Explore before you implement

Explain the diff in your own words

Write the first test yourself

Rotate who reviews agent PRs



Technical debt

lives in the code

Small, scoped diffs

Checkpoint commits

Diff coverage on every PR

Hooks for format and lint

Suggested Team Exercise



1



Find your project's AI policy

CONTRIBUTING.md, AI_POLICY.md, the PR template, or ask.

2



Write a CLAUDE.md or AGENTS.md

How to build and test, conventions, guardrails. Commit it.

3



Take one small issue through the workflow

Explore → Plan → Implement → Commit, with a plan file.

4



Keep a review log

What did the agent get wrong, and how did you catch it?



Explore → Plan → Implement → Commit

No files change until you approve a plan. Scope prompts like good issues.



Verify, then review

Read the tests first. Check coverage, error paths, scope, and dependencies.



Least privilege

Untrusted input plus privileged action equals risk. Deny the dangerous rungs.



You own what you submit

Follow your project's policy, disclose, and be able to explain every line.



- Chaparro, O. “Introduction to Agentic Software Engineering.” CSCI 435/535 lecture slides, William & Mary, 2026.
- Anthropic. “Claude Code: Best practices for agentic coding” and Claude Code documentation (hooks, subagents, skills, settings).
- “Test Coverage Analysis of Agentic Pull Requests.” ICSME 2026 (NIER). arXiv:2607.18057.
- Invariant Labs. “GitHub MCP Exploited: Accessing private repositories via MCP.” May 2025.
- Spracklen, J., et al. “We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs.” USENIX Security 2025.
- Gentoo Council AI policy (Apr 2024); NetBSD commit guidelines (May 2024); QEMU code-provenance policy (2025).
- Stenberg, D. on AI-generated security reports (2025); The Register, “Curl shuts bug bounty program...,” Jan 21, 2026.
- Storey, M.-A. “From Technical Debt to Cognitive and Intent Debt.” arXiv:2603.22106, 2026.