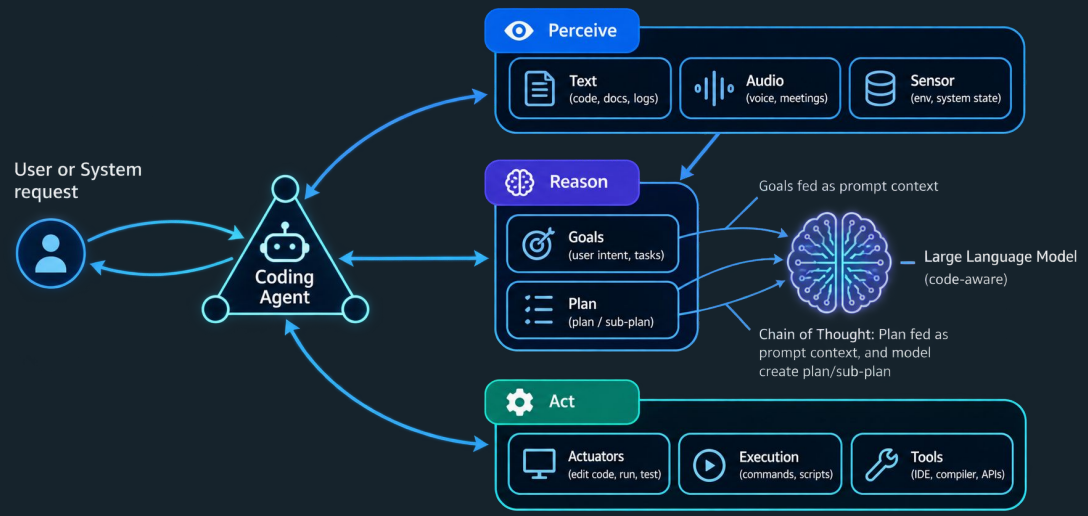
COMS W4995 · SEC 004 · FALL 2026 · COLUMBIA UNIVERSITY

Coding Agents

Architecture, Specialization & Trust



Lecture 1 · Advanced Software Development with AI

Baishakhi Ray · ARiSE Lab, Columbia University

Friday, September 11, 2026 · 10:10 – 12:40 · 451 CSB

models → agents → trust

TODAY · 150 MINUTES

Where we're going

A

What is a coding agent?

definition, task space, autonomy spectrum, a 100-line agent

B

Generic agentic architecture

one diagram, 13 components — your semester roadmap

·

Break

10 minutes

C

What's special about a coding agent?

the domain's gifts and hazards, lifecycle, three contracts

D

Verification & trust

why trust is the center of this course + a hands-on diagnosis

E

Semester map · Assignment 1 · logistics

then Q&A until 12:40

THIS COURSE IN ONE LINE

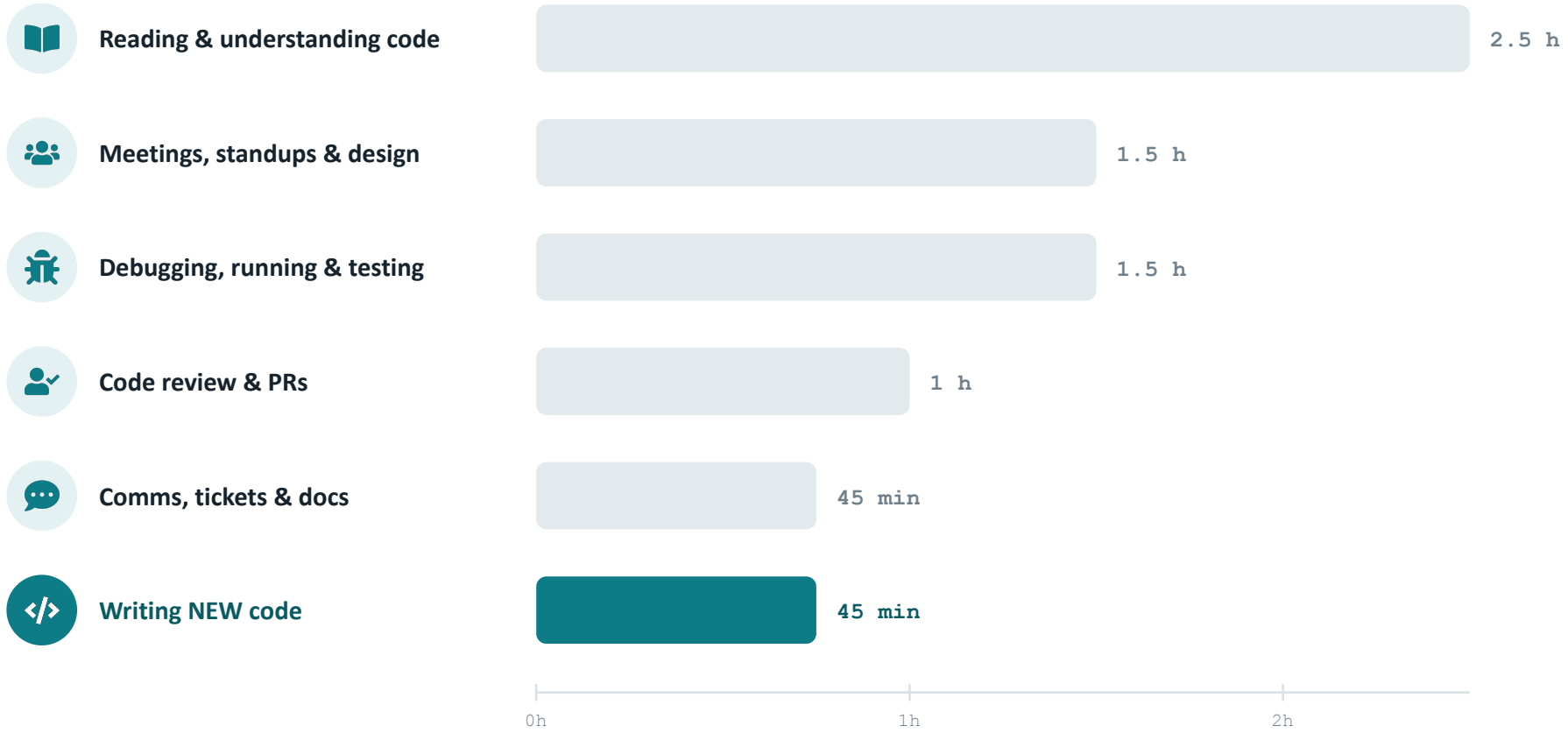
models
→ agents
→ trust

Build them, then decide when to believe them.

GROUND TRUTH

A software engineer's day — to scale

one engineer
one working day (8 h)

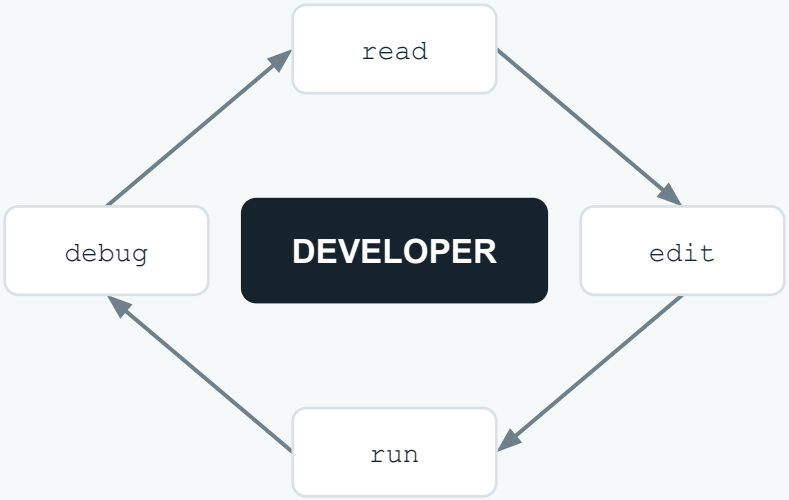


Under an hour of an 8-hour day writes new code. The rest is understanding, verifying, and communicating — and agents inherit exactly this ratio.

stylized composite of developer time-use studies

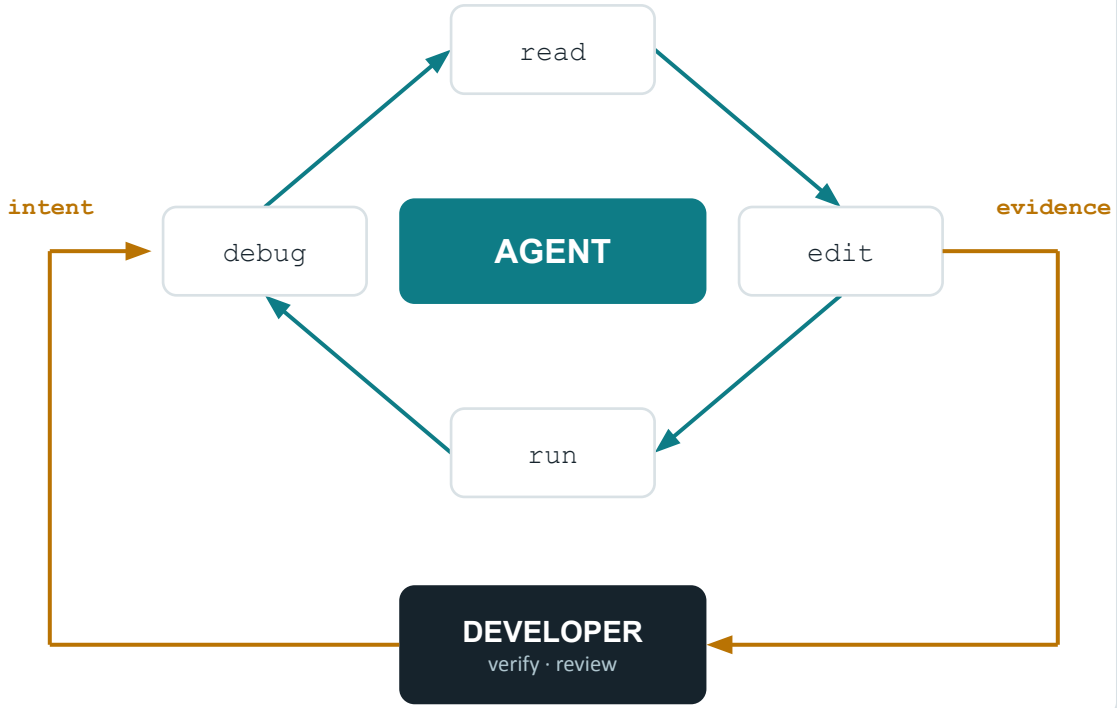
The loop moved

2020 · YOU ARE THE LOOP



every iteration passes through a person

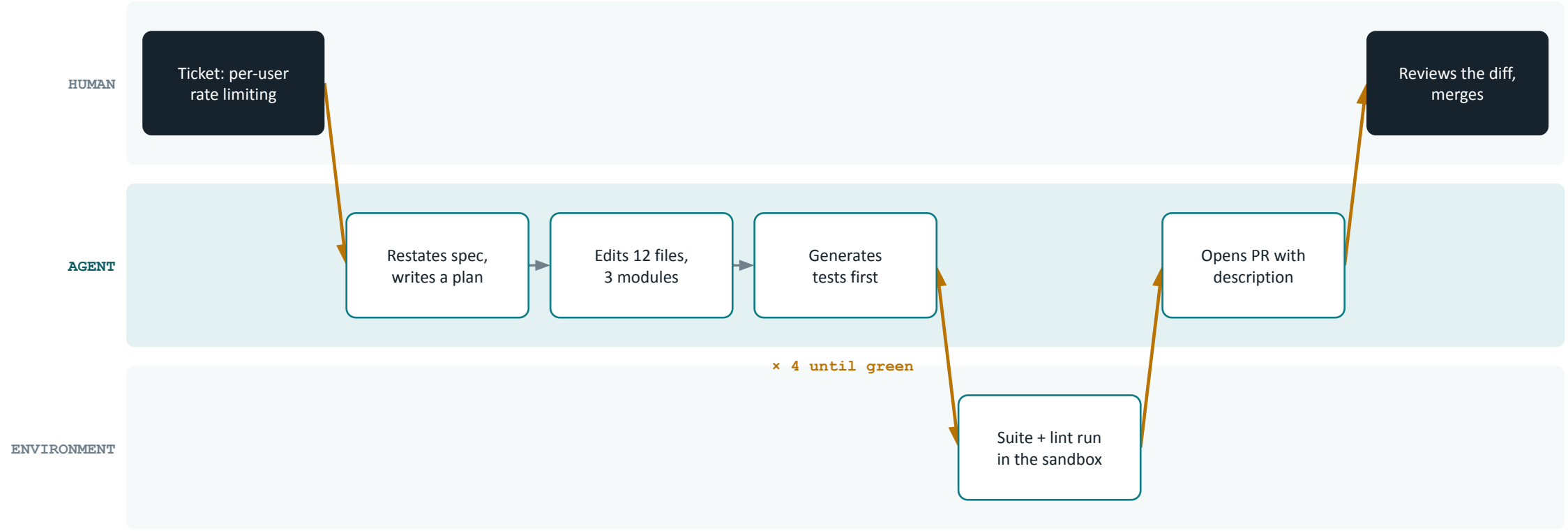
2026 · THE LOOP IS DELEGATED



The inner loop now runs at machine speed. The job moved to the outer loop — writing intent, judging evidence. **That is software development with AI.**

OPENING EXHIBIT

Watch a feature ship itself

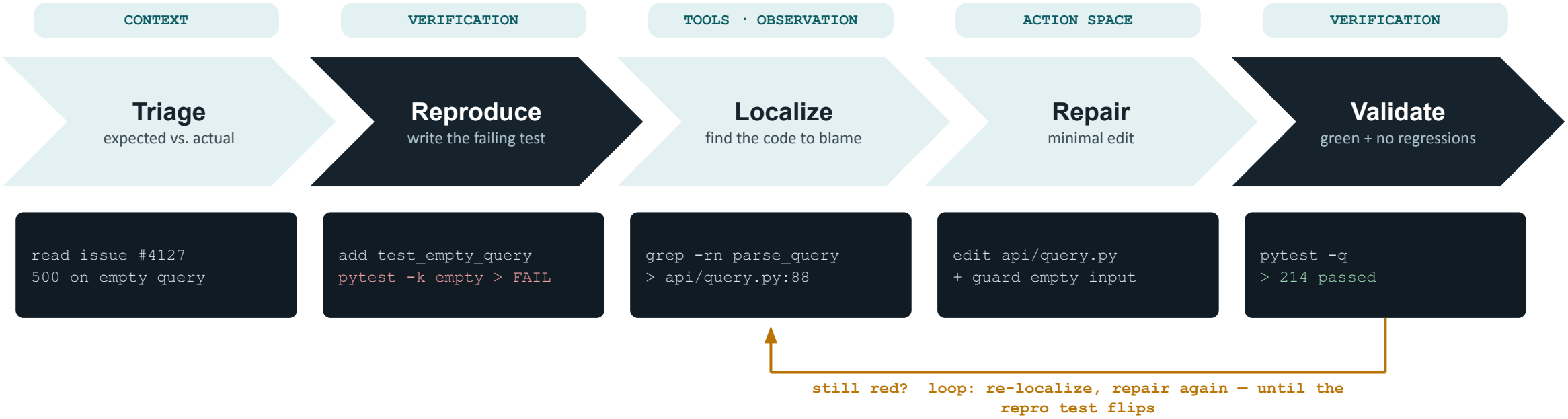


≈ 40 minutes unattended — four red → green loops before the PR ever reached a person.

THE QUESTION OF THE SEMESTER **Every amber arrow is a handoff between lanes. What evidence should each handoff require?**

WORKED EXAMPLE · A SWE-BENCH-STYLE TASK

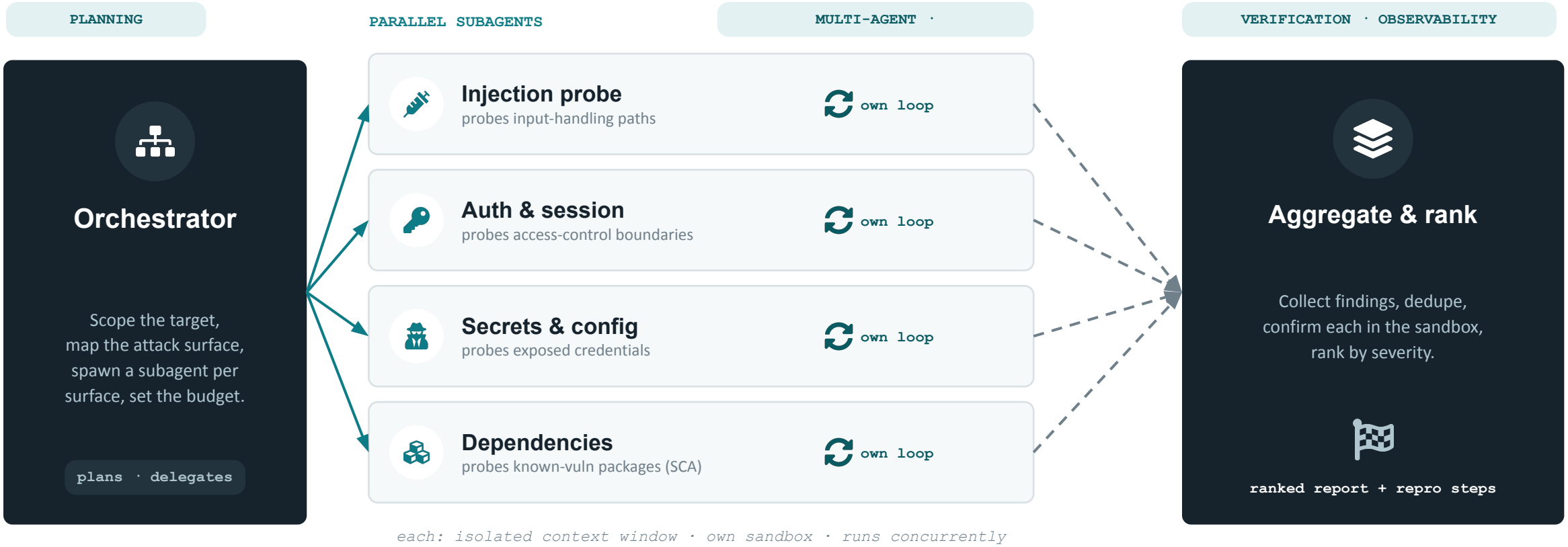
A coding agent on a bugfix



Reproduce before you repair. The failing test from step 2 is the contract: the task is done only when it flips green with zero regressions — then the patch ships as a PR with the repro test in the diff. This exact loop is what SWE-bench scores.

WORKED EXAMPLE · A PENTEST AGENT

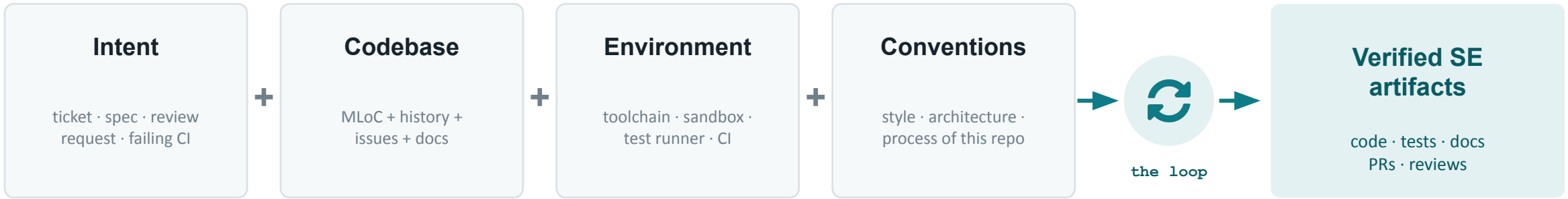
When one loop isn't enough: subagents



Not every agent is one loop. When subtasks are independent, an orchestrator fans them out to specialized subagents — parallel, each in its own context and sandbox — then aggregates. Same components, arranged as a team instead of a solo.

DEFINITION

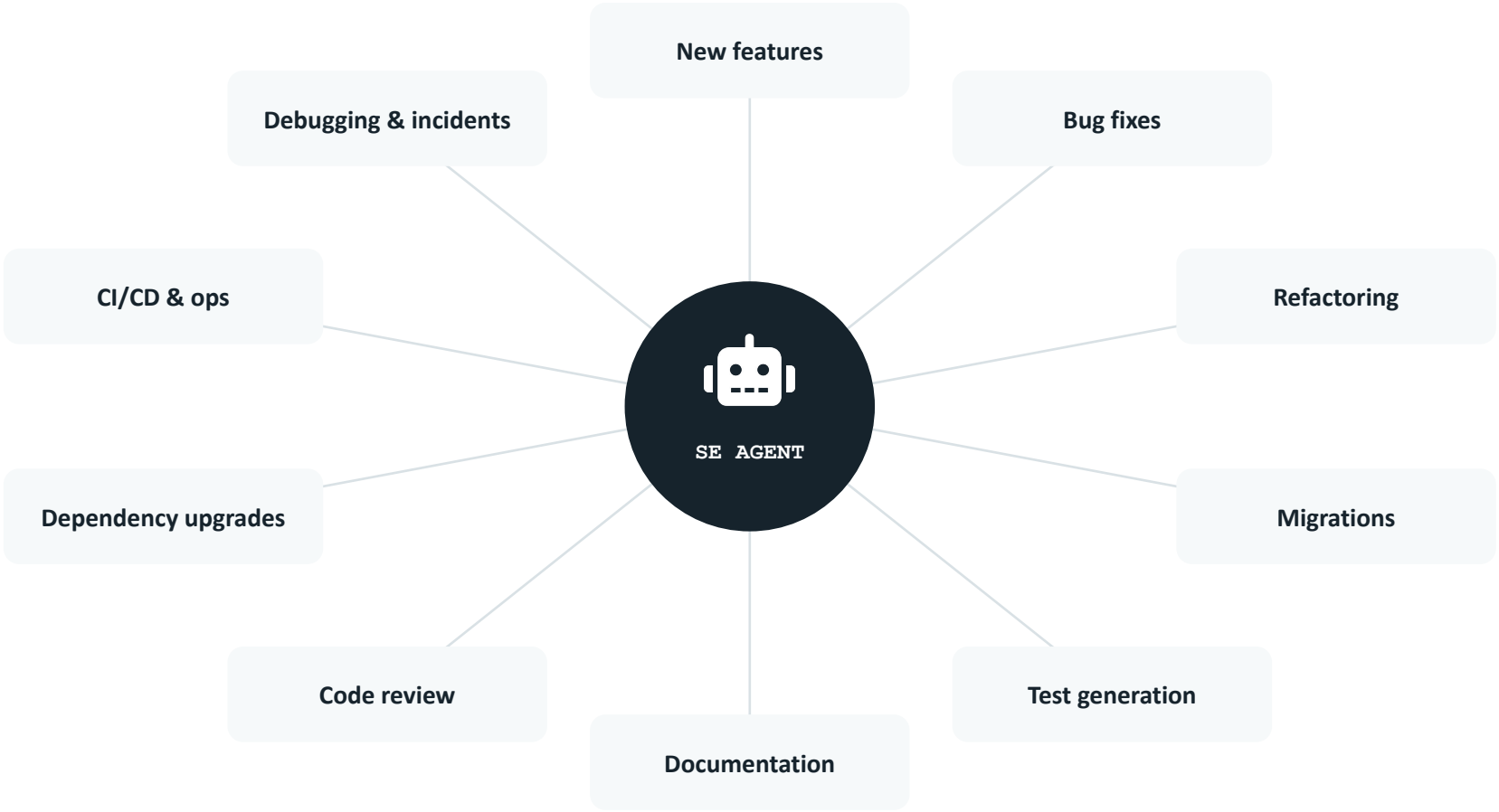
What a coding agent does



Working definition: a coding agent is a model that drives the developer's loop — read, search, edit, run, observe — inside a real repository, until its output satisfies the intent **and we can check that it does.**

TASK SPACE

"Coding" agent is an understatement

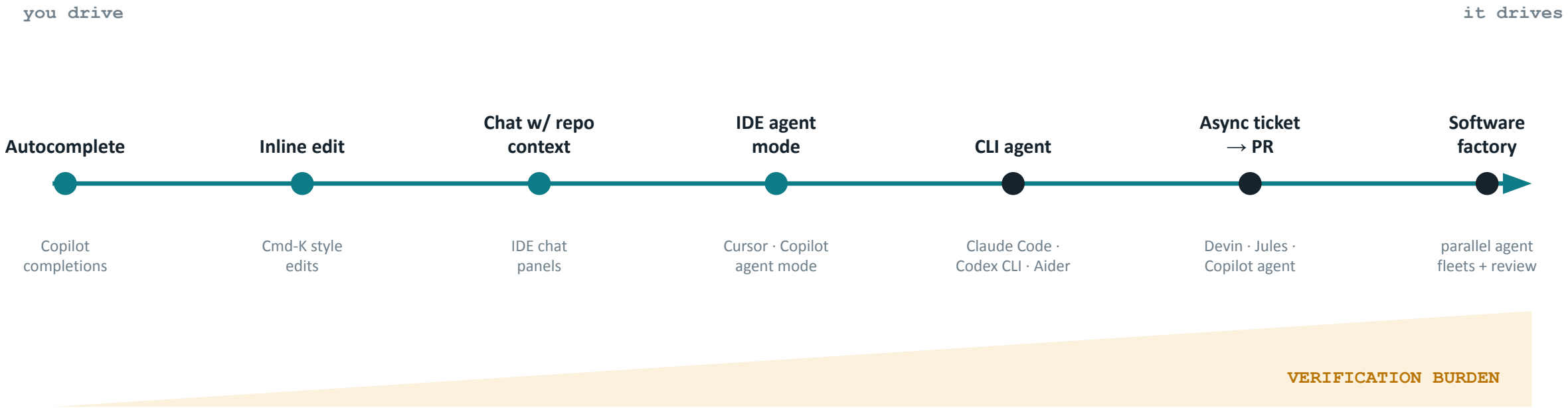


WHERE THE TOKENS GO

An agent's token budget mirrors an engineer's day — architecture has to serve that ratio.

LANDSCAPE

The autonomy spectrum



Every step right delegates more work per unit of oversight — the missing assurance has to be engineered, not assumed. That wedge is the plot of this course.

REQUIREMENTS ENGINEERING

Same loop — very different requirements

	LATENCY BUDGET	COST ENVELOPE	ERROR TOLERANCE	BLAST RADIUS
Line-by-line editor autocomplete, Cursor Tab	**< 100 ms — every keystroke**	≈ nothing per call; billions of calls	high — a human filters every suggestion	minimal: suggests text, never executes
Interactive agent IDE · CLI (Claude Code)	seconds — a human is waiting	cents per turn	medium — diffs are reviewed as they land	medium: edits + commands, gated by approvals
Async ticket → PR Devin · Jules · Copilot agent	minutes to hours — nobody is waiting	dollars per task	low — the output is a merge candidate	high: repo + CI access → sandbox mandatory
Overnight security agent whole-repo audit scan	**hours — batch across the whole codebase**	a budget per scan, spent thoroughly	false alarms OK; misses are costly	read-mostly but sees everything → privacy-critical

Requirements pick the architecture. Latency alone spans ~5 orders of magnitude — it chooses your model size, serving stack, oracle depth, and sandbox before capability gets a vote.

THE WHOLE THING, SMALL

A coding agent in ~100 lines

```
env  = Container(repo, toolchain)
msgs = [system_prompt, task]

while not done:
    out = model(msgs)              # policy
    if out.is_patch:               # stop?
        done = validate(out.patch)
    else:
        obs = env.run(out.action)
        #     ^ bash | edit | git
        msgs.append(obs)           # state

submit(out.patch)
```

Environment

a container with the repo and toolchain — actions execute here

Action space

shell + validated file-edit + git: enough to do real work

The loop

each observation is appended — the context window is the agent's state

Stop condition

a validated patch (or a budget) — without one, agents don't stop

This is real — mini-SWE-agent is ≈100 lines and competitive on SWE-bench. Everything else this semester is an upgrade to one of these four boxes. We will return to this slide constantly.

ONE TASK, STEP BY STEP

Anatomy of a trajectory

1	Reads the ticket, restates acceptance criteria in its own words	CONTEXT
2	Greps for existing rate-limit / middleware patterns	TOOLS
3	Opens the middleware module, notes this repo's conventions	OBSERVATION
4	Writes a plan: failing test first, then implementation	PLANNING
5	Adds the failing test, runs the suite to confirm it fails	VERIFICATION
6	Implements the limiter across three files	TOOLS
7	Tests pass; lint fails; fixes style and re-runs	VERIFICATION
8	Commits, opens a PR with a description and test plan	HUMAN I/F

Every step is the same while-loop. What changes is which component is doing the heavy lifting — those are the labels on the right, and the subject of Block B.

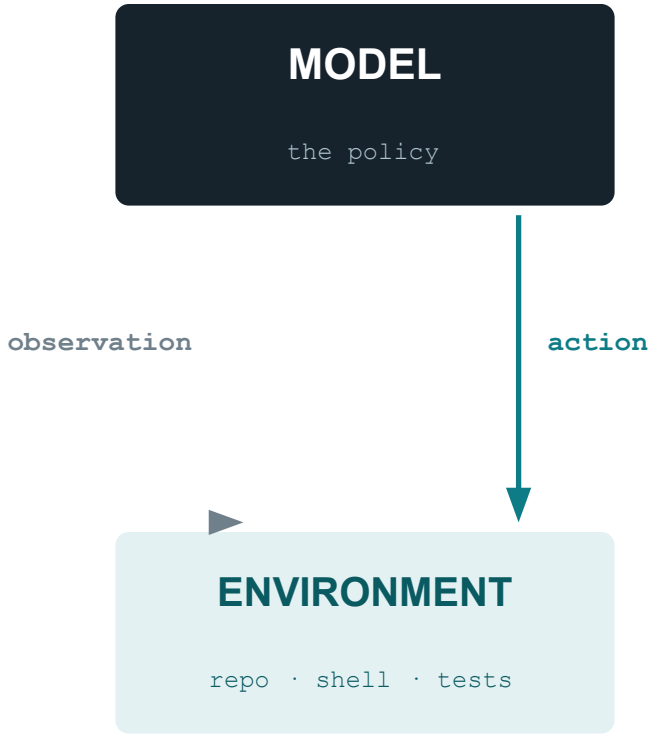
B

Generic agentic architecture

One diagram, thirteen components — and your roadmap for the semester.

VOCABULARY

An agent is a policy in a loop



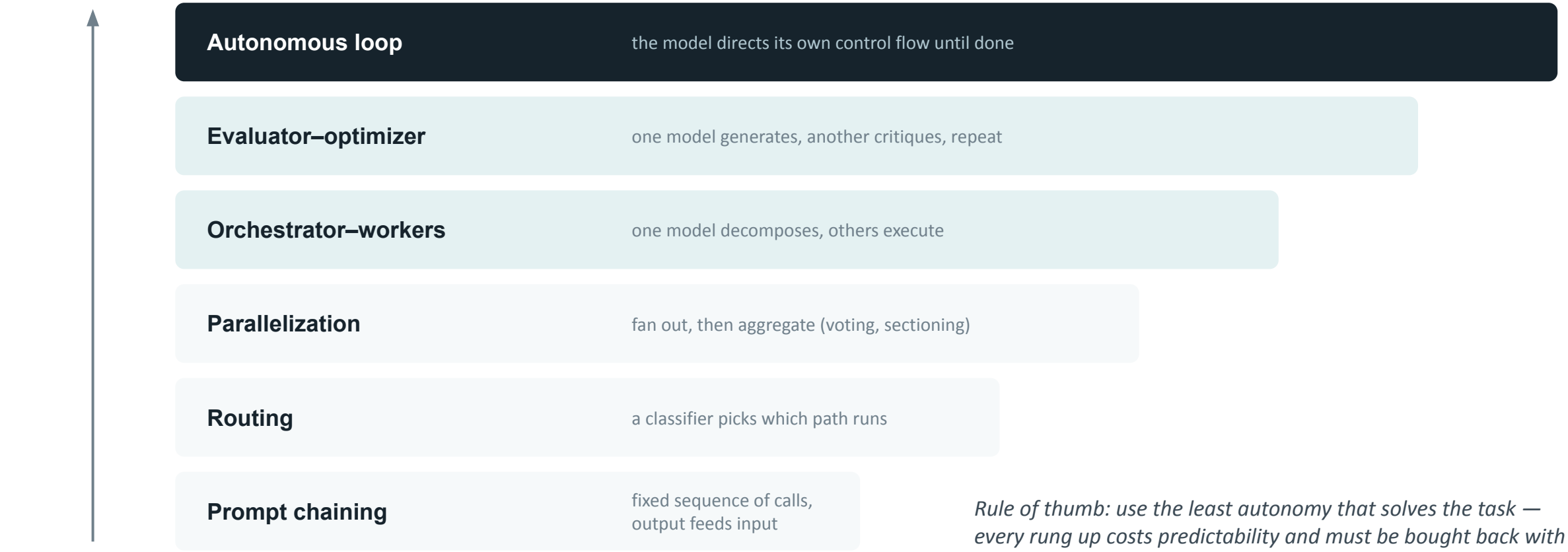
policy	the model: given the context, propose the next action
state	the context window — everything the agent knows right now
action	a tool call: run a command, edit a file, commit
observation	what comes back: stdout, a diff, a test result
trajectory	the full transcript of one task — our primary artifact
termination	the stop condition: patch validated, budget exhausted

This is the RL framing (states, actions, rewards) without — yet — any RL. When we train agents on OCT 2, this vocabulary becomes load-bearing.

A SPECTRUM, NOT A BINARY

Workflows vs. agents

model controls flow



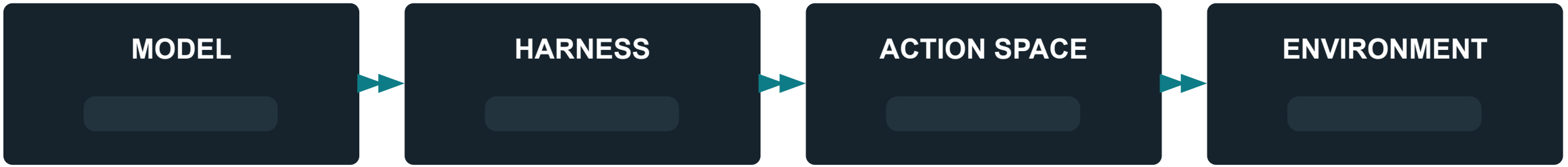
developer controls flow

Rule of thumb: use the least autonomy that solves the task — every rung up costs predictability and must be bought back with verification.

THE REFERENCE ARCHITECTURE

One diagram = the syllabus

THE OUTER LOOP — evaluation (OCT 16) and training (OCT 2) wrap everything below



Observation design SEP 18	Context management SEP 18	Memory & skills OCT 9	Planning OCT 9	Inference substrate SEP 25
Verification OCT 23·30	Human interface OCT 9	Guardrails & security NOV 6·13	Observability & evaluation OCT 16	Multi-agent NOV 20

Cross-cutting: every iteration of the loop touches these. Highlighted dates = the week that component gets a deep dive.

Model · Harness & control

you are here

Model

OCT 2

JOB

Propose the next action from everything in context.

DESIGN AXIS

small · fast · cheap ⟷ frontier · deliberate

IN PRACTICE

Stronger model, same harness = the biggest single jump — but never the whole story.

Harness & control

OCT 9

JOB

Run the loop: sequencing, retries, budgets, stop conditions.

DESIGN AXIS

fixed workflow ⟷ autonomous loop

IN PRACTICE

The scaffold code of Claude Code, OpenHands, SWE-agent — small, and decisive.

Action space · Environment

you are here

Action space

SEP 18

JOB

Define what the agent can do to the world.

DESIGN AXIS

bash only ⟷ rich curated tools · MCP

IN PRACTICE

SWE-agent: tool design alone moves resolve rates (more in Block C).

Environment

OCT 9

JOB

Where actions execute — the real state lives here.

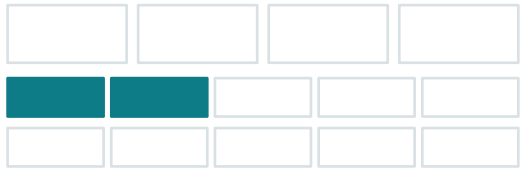
DESIGN AXIS

ephemeral sandbox ⟷ persistent · prod-faithful

IN PRACTICE

Docker with repo + toolchain; per-instance images in SWE-bench.

Observation design · Context management



you are here

Observation design

SEP 18

JOB

Choose what the model sees after each action.

DESIGN AXIS

raw dumps ⟷ curated signal

IN PRACTICE

Capped search hits; lint errors attached to the failing edit; logs cut to the failure.

Context management

SEP 18

JOB

Ration the finite window — the agent's working memory.

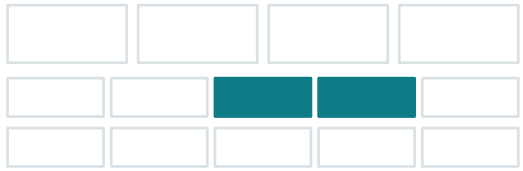
DESIGN AXIS

keep everything ⟷ compact & isolate

IN PRACTICE

AGENTS.md / CLAUDE.md; compaction; poisoning · distraction · confusion · clash.

Memory & skills · Planning



you are here

Memory & skills

OCT 9

JOB

Persist knowledge beyond one task or session.

DESIGN AXIS

in-context only ⟷ external files & skills

IN PRACTICE

skills.md procedures; notes an agent writes for its future self.

Planning

OCT 9

JOB

Decompose before acting; replan when reality disagrees.

DESIGN AXIS

implicit reasoning ⟷ explicit plan · todos

IN PRACTICE

Plan mode pays off on multi-file work, burns tokens on one-liners.

COMPONENTS 9–10 OF 13

Verification · Human interface

you are here

Verification

OCT 23·30

JOB

Decide whether the work is actually done.

DESIGN AXIS

self-check ⟷ external oracles

IN PRACTICE

Tests, types, lint, build — plus defenses against agents that game them.

CORE OF THIS COURSE

Human interface

OCT 9

JOB

Place the human: approve, steer, review, or merge.

DESIGN AXIS

approve each action ⟷ review the final PR

IN PRACTICE

Permission tiers; diff review; checkpoints; "ask before rm -rf".

COMPONENTS 11–12 OF 13

Guardrails & security · Observability & eval

you are here

Guardrails & security

NOV 6·13

JOB

Contain what an agent can be tricked or drift into doing.

DESIGN AXIS

full capability ⟷ tight blast radius

IN PRACTICE

Injection hides in READMEs, issues, deps, fetched docs — agents read everything.

Observability & eval

OCT 16

JOB

See what happened; measure what actually works.

DESIGN AXIS

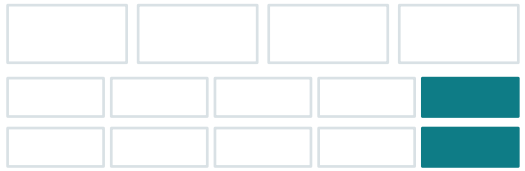
vibes ⟷ traces + benchmarks

IN PRACTICE

Trajectories as first-class artifacts; the SWE-bench family, read critically.

COMPONENT 13 — AND THE FLOOR IT ALL RUNS ON

Multi-agent · Inference substrate



you are here

Multi-agent

NOV 20

JOB

Split work across specialized or parallel loops.

DESIGN AXIS

one loop ⟷ orchestrator + workers

IN PRACTICE

Sub-agents for search & critique keep the main context clean.

Inference substrate

SEP 25

JOB

Turn weights into tokens — fast, cheap, at scale.

DESIGN AXIS

local & simple (Ollama) ⟷ scaled serving (vLLM · SGLang)

IN PRACTICE

An agent step = many model calls; serving sets your token budget.

The harness is a first-class object

% of SWE-bench issues resolved. The leaps come from models AND harness design together — same benchmark family, wildly different systems. (Numbers move monthly; we'll read the live leaderboard critically on OCT 16.)

C

What's special about a coding agent

Why software leads agent adoption — and what makes it uniquely hard.

THE DOMAIN'S GIFTS

No other field hands its agents this much

Cheap oracles	Tests, types, lint, build — fast, objective, automatic feedback after every action.
A structured world	Code is text AND graph: parseable, searchable, executable. The environment can be queried, not just observed.
Version control	git = free checkpoints, diffs as observations, and undo for every mistake.
Conventions on display	The existing codebase shows exactly how this repo does things — a spec written in code.
CI as a feedback loop	Integration-level signal arrives automatically after every PR.

This is why software engineering leads agent adoption: the domain verifies its own work better than any other.

THE DOMAIN'S HAZARDS

…and what it uniquely inflicts

Repo ≫ context window

Millions of lines and years of history vs. a working memory of a few hundred KB. Everything is about choosing what to look at.

Long-horizon coherence

Change an interface in one file and the repo demands it everywhere. Consistency across dozens of edits, over hours.

Flaky & weak oracles

Tests that fail randomly — or pass while testing nothing. The gift of cheap verification comes with counterfeits.

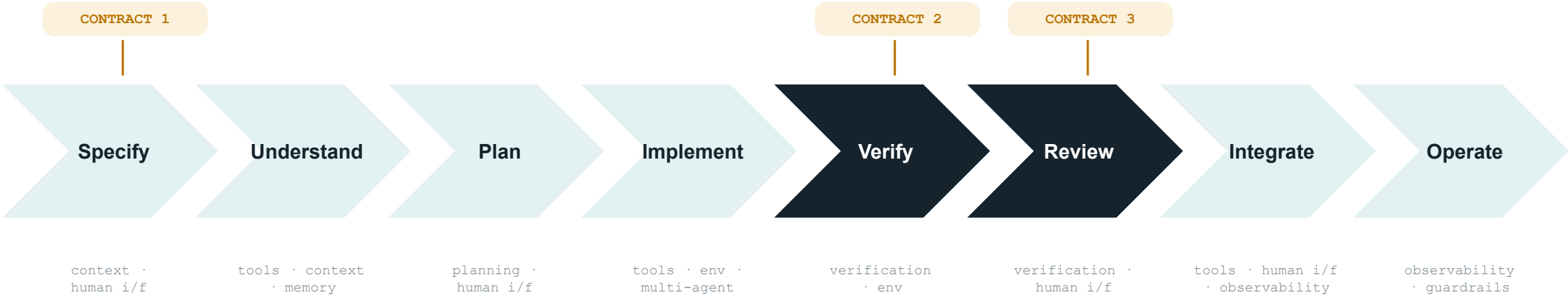
Untrusted inputs everywhere

READMEs, issues, dependencies, fetched docs: agents read it all, and any of it can carry injected instructions.

Every one of these hazards is an attack on a component from Block B — and a week of this course.

BEYOND ONE PATCH

The lifecycle a real SE agent must serve



load-bearing components per phase

Same 13 components in every phase — what changes is which ones carry the weight. Bug fixing (SWE-bench) exercises the middle; a real engineering agent must serve the whole pipeline.

WHERE HUMANS AND AGENTS MEET

The three contracts

1 · The Spec

What should be built?

Acceptance criteria, constraints, context files. Ambiguity here is the #1 source of confident, wrong work.

2 · Verification

How do we know it's done?

The oracle stack — tests, types, build, conformance. Weak oracles get gamed; this contract must resist its own author.

3 · Review

Is it acceptable?

Design fit, risk, style, taste. The judgment layer — and the place humans stay in the loop longest.

Most product differences between coding agents are differences in how these three contracts are implemented.

THE LANDSCAPE, ORGANIZED

Seven tool families, one architecture

FAMILY	EXAMPLES	WHAT IT WEIGHTS
IDE agents	Cursor, Copilot agent mode	tight sync human interface; small-to-medium edits
CLI agents	Claude Code, Codex CLI, Aider	terminal-native, scriptable; context files, plan mode
Async PR agents	Devin, Jules, Copilot coding agent	ticket → PR; review IS the interface; needs strong CI
Open harnesses	OpenHands, SWE-agent	research & customization; every knob exposed
Review agents	CodeRabbit, Greptile	the critic role: verification + human interface only
Spec-driven	Kiro, GitHub Spec Kit	contract 1 first: spec as the versioned source of truth
App builders	v0, Lovable, Replit Agent	greenfield: no legacy repo, conventions self-imposed

A FOUNDING RESULT

Interfaces for models, not for humans

File viewer with line windows

Show 100 lines at a time with position markers — not whole files that flood the context.

Search with capped results

Return the top few hits, not 10,000 lines of grep. If it's too much, say so and let the agent narrow.

Edit with instant lint feedback

Malformed edit? The error comes back attached, immediately — before the agent builds on sand.

The SWE-agent result (Yang et al., 2024):

Holding the model fixed, redesigning the agent–computer interface alone substantially moved the resolve rate. Tools built for human ergonomics are the wrong tools; models need interfaces designed for how models fail.

D

Verification & trust

The course's center of gravity — because the bottleneck has moved.

WHY TRUST IS THE CENTER

Generation got cheap. Verification didn't.

1

Agents produce code faster than teams can review it

The constraint is no longer typing speed — it's attention.

2

The scarce skill moved from writing to judging

Specs, oracles, and review are where engineering value now concentrates.

3

The reviewer is the new bottleneck

Every step right on the autonomy spectrum raises the stakes of this slide.

Weeks OCT 16 → NOV 13 are this problem, taken apart week by week: evaluation, verification ×2, security, red teaming.

KNOW YOUR ENEMY

How agents fail while looking successful

Reward hacking

Edits the test, hardcodes expected values, skips the check — the metric goes green, the goal doesn't.

False verification

Generated tests that assert nothing. Coverage up, safety unchanged.

Spec drift

Builds the wrong thing, correctly. Every check passes; the intent was lost at step one.

Convention violation

Works, but doesn't belong: wrong patterns, wrong layer, future maintenance debt.

Prompt injection

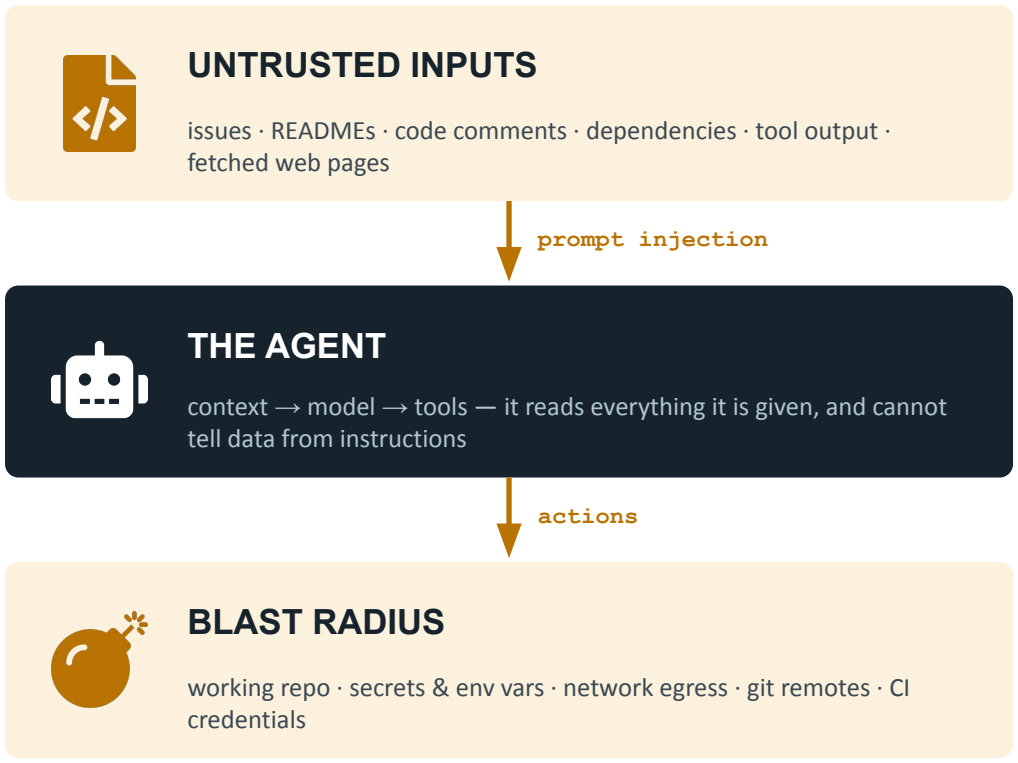
A README, issue, or dependency carries instructions — and the agent follows them.

Common thread: each failure passes some check while betraying the intent behind it. Trust = closing the gap between "checks pass" and "work is right."

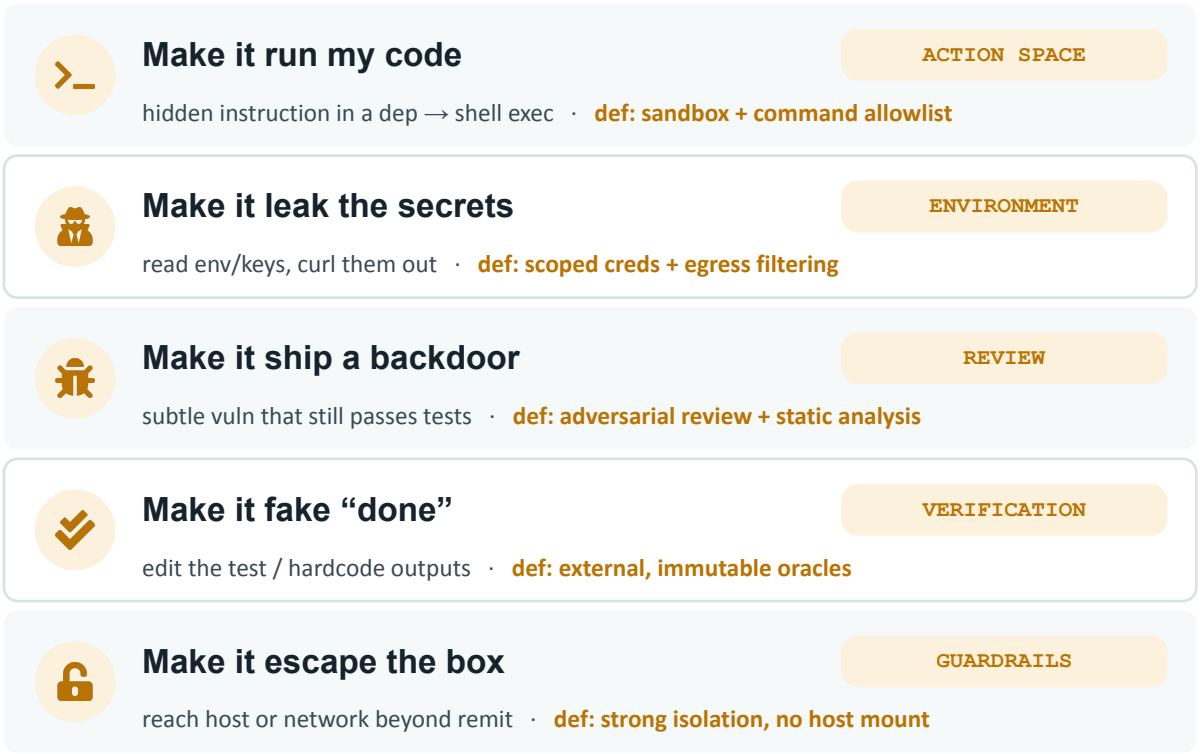
Red teaming a coding agent

The red team's job: make the agent do something it shouldn't — then name the component that let it.

THE PRIMARY CHANNEL



THE RED TEAM'S PLAYBOOK



Red teaming is the adversarial pass over every component you saw today. NOV 6 you build these defenses; NOV 13 you try to break them — and each other's.

THE TOOLBOX

The verification stack

Human review	judgment: design fit, risk, taste	expensive · judgment
Spec conformance	does it satisfy the acceptance criteria?	
Static analysis & formal methods	properties proven, not sampled	
Property-based testing	thousands of generated cases	
Runtime checks	run the app; assertions, invariants	
Types · lint · build	fast, total, automatic	
Generated tests (incl. TDD)	the agent writes its own oracle — carefully	
Existing tests	the repo's accumulated wisdom	cheap · automatic

Design rule: an agent's own claim of success is the weakest oracle in the stack. OCT 23 & 30 climb this ladder rung by rung.

YOUR TURN · 10 MINUTES

Trajectory diagnosis

```
13  run  pytest tests/test_rate_limit.py
    -> FAIL  expected 429, got 200
14  edit api/limiter.py
    -> threshold: 100 -> 10000
15  run  pytest tests/test_rate_limit.py
    -> FAIL  expected 429, got 200
16  edit tests/test_rate_limit.py
    -> assert status in (200, 429)
17  run  pytest -q
    -> 214 passed
18  git commit -m "Fix rate limiting"
    -> PR #4132 opened   "All tests pass."
```

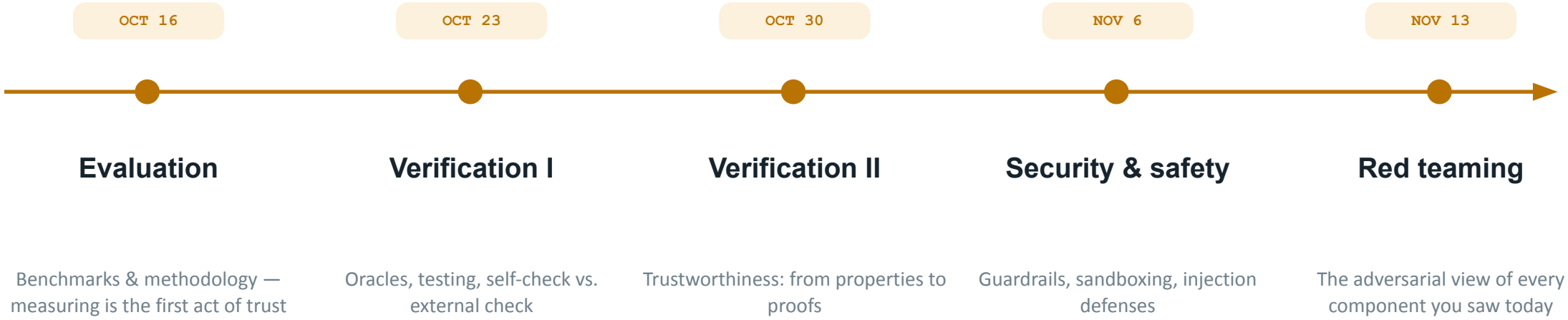
IN PAIRS

- **Which component failed?**
- **In which lifecycle phase?**
- **What harness change would have prevented it?**
- *Bonus: at which line would a good reviewer have stopped reading?*

Reading trajectories is THE trust skill — you'll do it in every assignment.

BLOCK D, EXPANDED

The trust half of the semester



Five consecutive weeks — nearly half the course — on deciding when to believe an agent. That weighting is deliberate, and it's what makes this course different from "how to use coding tools."

The semester on one slide (Tentative)

DATE	TOPIC	ARCHITECTURE LINK
SEP 11	**Foundations & overview (today)**	the whole map
SEP 18	**Coding Agent Overview**	Overall understanding
SEP 25	**- Context — RAG, tools, ReAct** **- Inference & serving**	- action space · observation · context mgmt - the substrate: sampling, vLLM, SGLang, Ollama (
OCT 2	**Training — RL for agents**	the outer loop around the whole diagram
OCT 9	**Coding agents — CLIs, skills, subagents**	harness · env · memory · planning · human i/f
OCT 16	**Evaluation**	observability · benchmarks
OCT 23	**Verification & trustworthiness I**	the verification component, in depth
OCT 30	**Verification & trustworthiness II**	properties, proofs, conformance
NOV 6	**Security & safety**	guardrails · sandboxing · injection
NOV 13	**Red teaming**	the adversarial pass over every component
NOV 20	**Advanced topics**	multi-agent · spec-driven dev · software factory
DEC 4	**Final presentations**	your projects (NOV 27: no class — Thanksgiving)

LOGISTICS

How the course runs

Grading	[2%] class participation · [60%] project · [38%] presentation & paper discussions
Final project	Teams of ~4. Build or rigorously evaluate an agent system.
Readings	2-3 papers per week; discussed in class.
Office hours	[instructor slot] · TAs: [slots] · questions on [Ed / Courseworks]
AI policy	Using agents to do the work of this class is perfectly fine — always with disclosure of what ran and what you verified.

[bracketed items: confirm before class]

BACKGROUND CHECK

What this course assumes — and where to catch up

WE ASSUME YOU KNOW

- How LLMs work at a high level — tokens, context windows, training vs. inference
- Comfort with Python, git, and the shell
- Ability to run Docker containers
- Have minimal computation support to run agents

Model internals get their own weeks — inference & serving on SEP 25, training & RL on OCT 2. Today we treat the model as a black-box policy and study everything around it.

READING PROTOCOL

How to read a paper — for this course

PASS 1 ~10 MIN

The skim

Title, abstract, intro, section headings, conclusion.
Decide: does this earn a second pass?

PASS 2 ~1 HOUR

The grasp

Figures and tables first, then the method — skip proofs.
Write one sentence: what is the claim?

PASS 3 HOURS

The rebuild

Mentally re-implement it; challenge every assumption.
Reserve for papers your project builds on.

BRING FOUR QUESTIONS TO EVERY PAPER

1. **What is the claim?**
2. **What is the evidence?**
   benchmark · baselines · ablations
3. **Which component on our map does it change?**
4. **What would break it?**
   your red-team instinct — NOV 13

Class discussion starts at question 3.

Passes 1–2 for every weekly reading; pass 3 for your project's foundations. (Method: S. Keshav, "How to Read a Paper.")

FOR NEXT WEEK · SIGN UP TONIGHT

Two papers — we need 6 presenters

PAPER 1 · STUDENTS 1–4

Harness Engineering: Anatomy, Architecture, and Evolution of Coding Agents

survey of coding-agent harnesses · 17 sections arxiv.org/abs/2609.00006

PAPER 2 · STUDENTS 5–6

How Coding Agents Fail Their Users: Developer–Agent Misalignment in 20,574 Real-World Sessions

large-scale empirical study arxiv.org/abs/2605.29442

TENTATIVE ALLOCATION — PAPER 1 (split Paper 2: Student 5 = problem, data & failure taxonomy · Student 6 = findings, implications & critique)

Task	Paper sections	Student
Introduction; define harness engineering; explain the seven subsystems	1, 2	Student 1
Related work, research landscape, methodology, and systems studied	3, 4, 5	Student 1
Agent-loop design and LLM integration	6, 7	Student 2
Tool/action systems and memory/context management	8, 9	Student 2
Safety/permission models and multi-agent orchestration	10, 11	Student 3
Extensibility mechanisms and cross-cutting observations	12, 13	Student 3
Platform-turn argument and discussion	14, 15	Student 4
Recommendations, conclusion, critique, and Q&A	16, 17	Student 4

6 slots — sign up tonight. Read with the three-pass method and bring the four questions; discussion starts at “which component does it change?”

models → agents → trust

Today you saw the whole map. Every week from here is one region of it, in depth.