A DRUPAL DEVELOPMENT TALK

Strategic Programming with AI for Drupal

André Angelantoni · Founder, Performant Labs

performant labs

AGENDA

Table of Contents

QUICK POLL

Poll 1

Answer in Zoom — results will show live.

SECTION 01

Strategic vs. Tactical

The distinction the rest of the talk rests on.

FOUNDATIONS

Strategic vs. Tactical Programming

A distinction drawn by John Ousterhout in A Philosophy of Software Design and his related teaching materials.

S Strategic Programming

An investment mindset

- Long-term design focus
- Deliberately minimizes complexity
- Treats good design as an ongoing investment, not overhead

T Tactical Programming

The fastest path to done

- Get the next feature or bug fix working quickly
- Optimizes for immediate progress over design quality
- Often accumulates technical debt as a side effect

Source: John Ousterhout, A Philosophy of Software Design (and related teaching materials)

The New Strategy vs Tactical

S Strategic Programming

The Army General

- Guides development without knowing all the details.
- Steps in when goals are not being reached.
- Continually improves the process.
- Is directing many people.

T Tactical Programming

An Infantryman

- Knows all the details.
- Since details matter, takes a good amount of time to get things done.
- As a single person, can do only so much.

FOUNDATIONS

It's a spectrum, not a switch

Every line of code you write sits somewhere on this line — most work leans one way or the other, day to day.

SECTION 02

The Landscape

What exists in 2026 — frameworks, tools, providers, and the models I actually use.

LANDSCAPE

Four ways in

Start custom. You learn where the gates actually need to be, and you can always adopt a framework later — once you know what you'd be adopting it for.

FRAMEWORK

Spec Kit

Spec-driven development

FOCUS

A fixed sequence of versioned Markdown artifacts steers any coding agent — the documents, not the agents, are the structure.

THE SEQUENCE

constitution → specify → plan → tasks → implement

FRAMEWORK

Gas Town

Multi-agent orchestration

FOCUS

Coordinates many coding agents at once, handling the state and tracking that fall apart when you run them by hand.

OFTEN DESCRIBED AS

A new kind of IDE — or "Kubernetes for coding agents."

FRAMEWORK

LangGraph

Durable graph execution

FOCUS

Graph-based, stateful, durable orchestration — execution modelled explicitly as nodes and edges that survive failure.

WHY IT GETS PICKED

A common choice for production multi-agent systems.

EASIEST WAY IN

Custom

Roll your own rigor

FOCUS

Nothing to adopt. You define the phases, the gates and how much review each one gets — in prompts and scripts you already understand.

WHAT WE'RE SHOWING TODAY

Both pipelines in this talk are built this way: the custom-rigor coding pipeline, and the Drupal Canvas pipeline.

LANDSCAPE

Six tools in Claude Code's category

Terminal agents, IDE-native agents, and harnesses. Roundups usually compare seven to ten; realistically a few dozen exist and about a dozen matter.

Codex CLI

OpenAI · terminal

The natural fit if the team already pays for ChatGPT — that access usually covers daily coding.

Gemini CLI

Google · terminal

Free, Pro and Ultra users moved to Antigravity CLI in June 2026 — Code Assist keeps it.

OpenCode

Open source · terminal

The strongest terminal-first open-source workflow — dedicated TUI, your own provider keys.

Cline

Open source · VS Code

5M installs, parallel terminal agents in CLI 2.0. Free tool, but you pay it back in API tokens.

Cursor

AI-native IDE (VS Code fork)

Still shipping. The $60B SpaceX deal closed 15 Aug 2026 — now a SpaceXAI subsidiary.

DeepSeek Harness

Agent runtime · MIT

Developer preview since 13 Aug 2026. Plugin-first — an agent factory, not a ready-made agent.

ORIENTATION

Four models, four jobs

PRIMARY

Claude

Fable · Opus · Sonnet

Fable starts design and architecture. Opus orchestrates and decides. Sonnet runs the agent fleet.

SECOND OPINION

ChatGPT

OpenAI

An independent read at the gates — a model that shares none of Claude's blind spots.

PANEL

ChatGPT + Opus

reviewer + decider

The outside model reviews in parallel with the in-session agents. Opus reconciles and decides.

PR REVIEW

DeepSeek

the fourth pass

Every pull request gets one more independent reviewer before it merges.

Every additional model usually finds something the others missed — which is why all four are still necessary in 2026.

SECOND OPINION — ChatGPT

≈ $0.26 per review · **≈ $115** / month

$4 / $20 per M — short-context promo rate

PR REVIEW — DeepSeek

≈ 2.6¢ per review · **≈ $11** / month

$0.44 / $1.32 per M — new peak rate from 16 Aug 2026 (was $0.14 / $0.28)

Estimates assume ~50K input + 3K output per review, 20 reviews a day, 22 days. Priced 21 Aug 2026: GPT-5.6 Sol short-context promo ($4/$20, held to 21 Nov) and DeepSeek V4-Flash peak — off-peak is half price, so ~10× is the conservative read. Version numbers move; the four jobs don't.

SECTION 03

The Pipeline

A test-first backbone, two human gates, and a four-rung dial for how much review each change earns.

> You want to describe the task, you want to describe the guardrails, you want to describe the exit criteria, and then just go let the model cook and come back in a little bit.

Boris Cherny

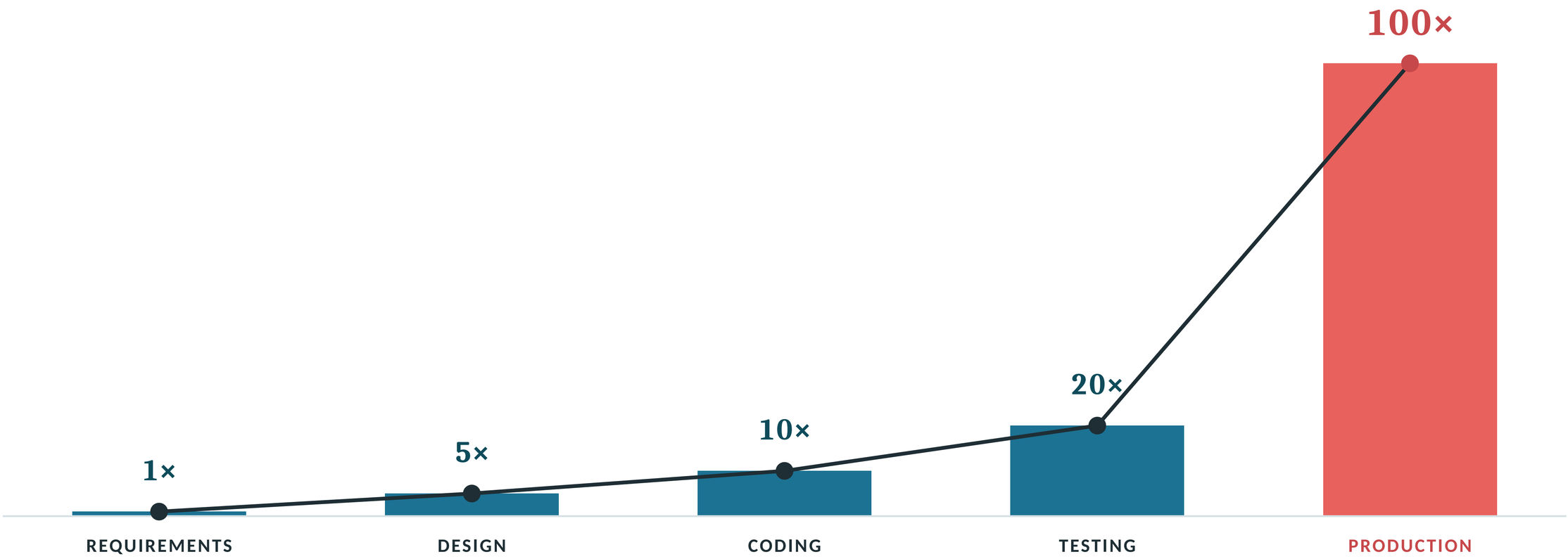
Creator of Claude Code · Y Combinator, July 2026

performant labs

AI PIPELINE

The later you find it, the more it costs

Relative cost to fix the same defect by the stage that catches it. The curve is geometric — production is not one step worse, it is another regime.



The bug is the same on every rung — the only thing that changed is when you found it.

AI PIPELINE

The Playbook and the Handbook

PLAYBOOK

How we build software

- **Conventions, pipeline specs, role templates**
  the standards every agent is held to
- **Read by 20+ downstream repos**
  one definition of the process, not twenty
- **Drift protection in CI**
  evals fail the build when the docs go stale

HANDBOOK

How the infrastructure runs

- **88 docs — hosts, services, local inference**
  what runs where, and why it was built that way
- **Troubleshooting keyed by symptom**
  you search what you're seeing, not what it's called
- **Published as a browsable site**
  the sidebar is the directory tree, so it can't drift

Write it down once and every future session starts from it.

AI PIPELINE

Playbook — the shared engineering standard operating procedures

- **One git-versioned repo**
  conventions, pipeline specs, role templates
- **Consumed by 20+ downstream repos**
  one source of truth for how every project's agents work
- **One test-first pipeline**
  O → D → A → T → F → T → A → U → S
- **Rigor is a per-story dial**
  direct · in-session · second-opinion · panel
- **Epics, not tickets**
  sub-issues for hierarchy; a YAML manifest declares each story's blast radius
- **Drift protection in CI**
  evals sweep for retired terminology and unechoed canonical facts

AI PIPELINE

The whole pipeline, one gate at a time

O	**Orchestrate**	Maps the closest existing feature; default is extend it, not build new
D	**Design**	A human-approved wireframe — only when there's a UI surface
A	**Architecture review**	The plan is checked before any code exists
T	**Tester — RED**	Authors the suite first and proves it fails
F	**Implement**	The Feature implementor writes code against the failing suite
T	**Tester — GREEN**	The same suite, now passing — verified, not asserted
A	**Anti-duplication**	Same reviewer checks the diff against O's map — blocks a parallel path
U	**UI walkthrough**	Drives the live UI in a real browser, every control
S	**Spec audit**	Visual + accessibility authority on the finished surface
✓	**Merge**	PR opened; review and CI driven to done, not handed back

AI PIPELINE

The rigor dial — four rungs

direct

Single agent, no gates

WHAT RUNS

No dedicated phases, roles or gates — and no brief. The agent edits directly and self-checks (build, lint, smoke) instead of running a formal RED/GREEN cycle.

USE WHEN

Genuinely trivial: a typo, a config value, a comment, a one-line copy tweak.

in-session

Every gate, no outside model

WHAT RUNS

Architecture Reviewer, Tester, UI Walkthrough and Spec Auditor all run — but no outside model is ever called.

USE WHEN

Thin read-and-render, templating, mechanical changes.

second-opinion

One outside model at each gate

WHAT RUNS

Adds an independent outside model reviewing both the brief and the diff. Hard findings block the merge.

USE WHEN

The default minimum for non-trivial implementation work.

panel

Cross-vendor pair, reconciled

WHAT RUNS

Adds a cross-vendor pair — the outside model plus a fresh-context Opus — on a byte-identical prompt, then reconciles their findings.

USE WHEN

High-stakes changes, where a second vendor's opinion earns its cost.

AI PIPELINE

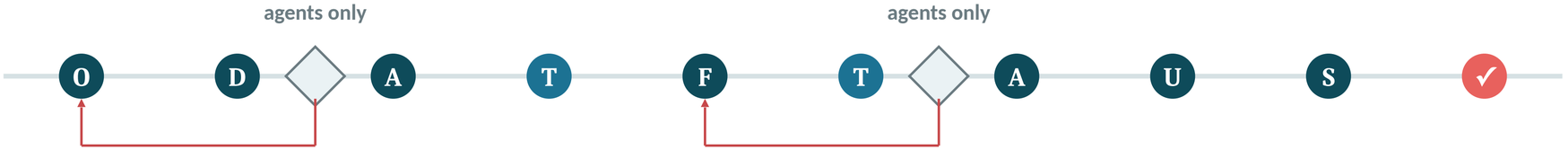
All four rungs at a glance

From in-session upward the same test-first backbone runs every time; each rung adds more review at the two gates. direct skips the backbone entirely.

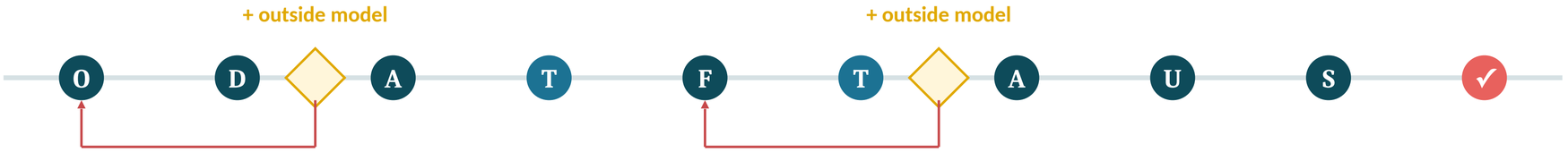
direct

One agent. No brief, no dedicated roles, no gates — build / lint / smoke replaces the formal RED→GREEN cycle.

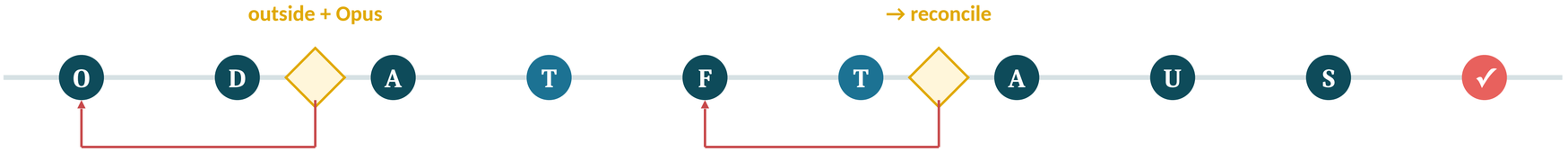
in-session



second-opinion



panel



O Orchestrate · D Designer · A Architecture · T Tester (RED→GREEN) · F Implement · U UI walkthrough · S Spec audit · ✓ Merge phase.

Diamonds (◆) are the two human gates; a blocked gate sends work back (red arrows) to an earlier

AI PIPELINE · DIRECT

direct — no gates, no roles, no brief

The lightest rung on the dial. One agent edits the file and checks its own work. Nothing else runs — and for a typo, nothing else should.

WHAT DOESN'T RUN

- No brief.md — nothing to write, nothing to approve
- No Architecture Review, UI Walkthrough or Spec Audit
- No Tester RED→GREEN cycle — self-check replaces it
- No gates, so no outside model and no human approval step

WHERE THE LINE SITS

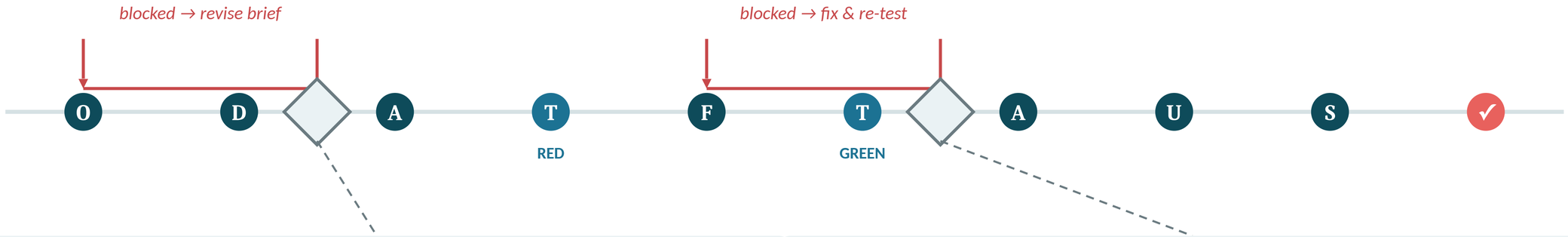
- A typo or a copy tweak
- A single config value
- A comment
- A one-line change with no behaviour to test

The discipline is in choosing it honestly. If the change has behaviour worth testing, it isn't a direct change — move up a rung.

AI PIPELINE · IN-SESSION

in-session — the pipeline's own agents

The lightest rung that still runs the full backbone. A human approves at each gate, but the pipeline's own agents are the only reviewers — no outside model is called. Used for thin read-and-render, templating or mechanical changes.



BRIEF GATE

In-session only: the Architecture reviewer checks the plan.

DIFF GATE

In-session only: anti-duplication check and spec audit on the diff.

A blocking finding at a gate sends the work back (red arrows) to an earlier phase to fix — then it re-enters the flow.

O Orchestrate · D Designer* · A Architecture · T Tester (RED→GREEN) · F Implement · U UI walkthrough* · S Spec audit · ✓ Merge * UI surfaces only

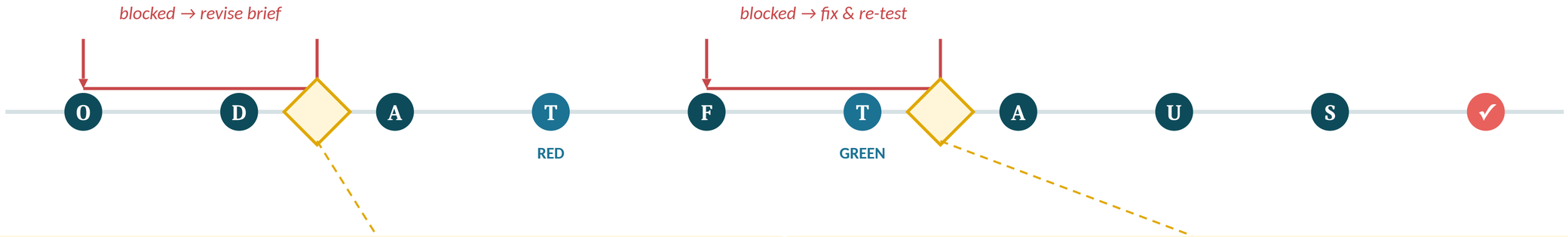
QUICK POLL

Poll 2

Answer in Zoom — results will show live.

second-opinion — the Critic at each gate

The default for non-trivial work. Adds the Critic — one independent outside model — at both gates, on top of the in-session agents. A hard finding blocks the merge.



BRIEF GATE

The Critic reviews the brief before you approve it.

DIFF GATE

The Critic reviews the diff before you approve the merge.

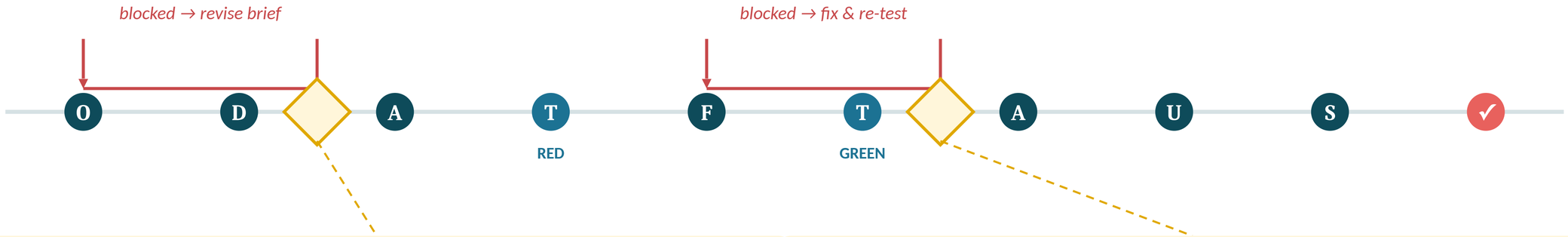
A blocking finding at a gate sends the work back (red arrows) to an earlier phase to fix — then it re-enters the flow.

O Orchestrate · D Designer* · A Architecture · T Tester (RED→GREEN) · F Implement · U UI walkthrough* · S Spec audit · ✓ Merge * UI surfaces only

AI PIPELINE · PANEL

panel — two-model pair, reconciled

The highest rung for high-stakes changes. At each gate the same prompt goes to two independent reviewers in parallel — the Critic and a fresh-context Author — and their findings are reconciled.



BRIEF GATE

Critic + fresh-context Author review the brief, then reconcile.

DIFF GATE

Critic + fresh-context Author review the diff, then reconcile.

A blocking finding at a gate sends the work back (red arrows) to an earlier phase to fix — then it re-enters the flow.

O Orchestrate · D Designer* · A Architecture · T Tester (RED→GREEN) · F Implement · U UI walkthrough* · S Spec audit · ✓ Merge * UI surfaces only

SECTION 04

A Session in Practice

What one real evening of this work actually looks like.

IN PRACTICE

A Typical Work Session

SUBSCRIPTION

Claude Max

Pro $20	Max 5x $100	Max 20x $200
baseline paid tier	5× Pro's usage	20× Pro's usage

PARALLEL SESSIONS

Several sessions running at once

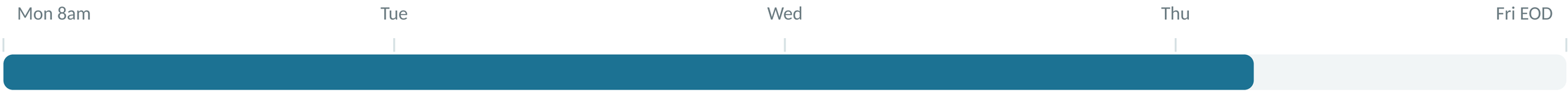
- **Claude Code** — terminal
- **Claude Desktop** — app

MODEL ASSIGNMENT

All Claude 5 family

Orchestrator	Opus
Most agents the bulk of the work	Sonnet
Design / architecture starts here	Fable
Deciding roles issue-writing · arch review · spec audit	Opus

A HEAVY WEEK



~80% *of the weekly allowance consumed by end of day Friday — and only by being careful to run Sonnet for most tasks.*

IN PRACTICE

What's Actually Running

A “session” isn't one thing talking to you. It's an orchestrator plus spawned background subagents — each with its own context window, running in parallel.

ORCHESTRATOR

A 200-line conversation, still going

Stays responsive while the work happens elsewhere.

BACKGROUND SUBAGENT

Fixing a production outage

live system, mid-session

own context window

BACKGROUND SUBAGENT

Moving a Piper TTS voice

to a different GPU node

own context window

BACKGROUND SUBAGENT

Investigating routing architecture

read-only exploration

own context window

All three ran at once. None of them blocked the others — or the conversation.

So “multiple sessions” and “delegating within one session” aren't in tension — you can do both at once.

IN PRACTICE

Epics, Issues, and Parallelization

EPICS DECOMPOSE

Into tracked issues

A big initiative — ship a new language's full content and audio — breaks into discrete, sequenced issues. Each is a standalone unit of work with its own acceptance criteria.

NO AD-HOC EDITS

Everything traces to an issue

Even a one-line fix starts with `gh issue create`. Issues are the audit trail, not an afterthought.

CAPPED AT SIX

Parallelism has a ceiling

A standing rule holds concurrent agents, builds and in-flight PRs at six or fewer. The rest queue and start as slots free.

ISOLATED WORKTREES

One per workstream

Each parallel stream gets its own git worktree, so simultaneous agents never collide editing the same files in the same place.

ONE RECENT EVENING — SIX PRs IN FLIGHT

Each was independently reviewed, fixed and merged — and not one of them blocked another.

IN PRACTICE

The Review Loop, Not Just the Write Loop

AI-assisted coding isn't "write once." The write step is one node in a cycle that has to close before anything ships.

ONE SCRIPT, ONE EVENING

TWO REAL FINDINGS

- No timeout on a fetch call
- Non-atomic file writes

Both fixed, then re-reviewed.

75 before → **88** after *then merged*

Across one evening, three separate rounds surfaced real, substantive findings before anything shipped. That is the loop working, not the loop failing.

IN PRACTICE

Guardrails That Make This Safe to Trust

Autonomy is only comfortable because the fences are real — and the agent cannot take them down.

Not skippable

Enforced automatically

- Pre-commit hooks — secret scanning, linting
- Pre-push blocked on main
- CI required before any merge

None of it optional. None of it bypassable by the agent.

Issue first

A standing rule

- File a tracked issue before fixing anything
- Even the trivial one-line changes
- No ad-hoc edits, ever

The work is always traceable to a reason.

Read the findings

Scores are not a rubber stamp

- Review output is read and acted on every time
- Findings get fixed, not noted
- A high score alone is not “done”

The gate only works if someone walks through it.

IN PRACTICE

When It Goes Wrong: A Real Debugging Story

Production TTS synthesis went down mid-session. What followed is the part that doesn't fit the "generates functions" story.

1

Traced, not guessed

Diagnosis came from the actual server logs via journalctl — not from inference about what might be wrong.

2

Root cause found off-piste

A wedged internal proxy process — several layers removed from any code being written that night.

3

Verified for real

Fixed, then confirmed with real synthesis requests. A green health check would not have proved it.

4

Written up

Documented as an incident writeup, so the next occurrence starts from knowledge instead of scratch.

The tool doing real operational work — diagnosing a live system, not generating functions in a sandbox.

IN PRACTICE

What a Session Still Costs You

Automation doesn't remove the judgment calls. It surfaces them faster.

A REAL DECISION FROM THAT SESSION

How should the app choose between two possible voices for a language?

It didn't decide. It stopped and asked — because that's a product call, not a technical one.

Set the expectation accordingly:

Less typing, not less deciding.

performant labs

IN PRACTICE

Tips and Tricks

None of these are clever. They're just the five places where a sentence of yours saves an hour of the agent's.

1

BEFORE IT STARTS

“Do you have any questions before you begin?”

Surfaces the ambiguity while it's still cheap — before a wrong assumption is baked into a diff.

2

WHEN IT REACHES FOR A CUSTOM SOLUTION

“Investigate how others are handling this, including libraries and plugins. Cast a wide net.”

Agents default to writing code. This makes them look for the thing that already exists first.

3

FOR AN EPIC YOU'LL RUN AFK

“Drive this to completion with maximum parallelization. Stop only for hard blockers. Ask questions now.”

Sets autonomy, throughput and the interrupt threshold in one line — and front-loads the questions.

4

AT THE U GATE

“Walk the new feature looking for obvious errors. Correct them, then backfill with a Playwright test.”

Turns a walkthrough into permanent coverage: every bug found this way leaves a test behind.

5

FOR ANYTHING FROM TODAY

“Ask Grok — it has far more access to X.com than the other models.”

Training cutoffs and thin web access leave other models stale on breaking news; Grok's live X feed doesn't have that problem.

SECTION 05

Front-End Work: Drupal Canvas

A second pipeline — for building sites, not fixing code.

DRUPAL CANVAS

Drupal Canvas pipeline — build, not audit

This is the Website Front-End pipeline — not the coding pipeline, and not the Website Audit pipeline.

O Orchestrate · D Designer · A Architecture · T Tester · F Implement · U UI walkthrough · S Spec audit

STACK & RULES

- Drupal Canvas + SDC on a DripYard theme — adapter plus a per-site profile
- F implements in the right CSS layer. Never patch Canvas-rendered HTML (Layer 4)
- Reuse DripYard components first — the schema is each .component.yml

CANVAS GOTCHAS

- Omit component_version — let preSave() set it
- noUi stays in regions
- drush cex does not export Canvas pages

T is headless — markup, axe, state. U clicks. S looks.

DRUPAL CANVAS

The CSS Locator

The key mechanism in this pipeline. Given a CSS change, it finds where that change belongs in the layer hierarchy — across many files — before a line is written.

T Without it

Style the symptom in the nearest file

- Add !important, or out-specificity whatever is upstream
- The rule lands in whichever file was already open
- It works today — and the cascade gets harder to reason about

S With the Locator

Place the cause at the right layer

- Reads the project's declared layer hierarchy
- Resolves one target: the correct layer, and the file inside it
- No !important, no specificity fight — the change lands where it belongs

This is Ousterhout's argument, in CSS. The tactical rule always works today; the strategic one is still cheap to change in a year.

Playwright + which model

Playwright shows up in three places — different jobs, different models.

T — Tester

scripts, not an agent loop

- Chromium via Playwright
- axe-check · state-invariants.spec.js
- cascade / perturb
- perf + a11y gates

Sonnet

T only reads reports — Playwright is the runner.

U — UI walkthrough

live browser

- Default backend: Playwright MCP — headless Chromium, same contract as coding-pipeline U
- Clicks every control on the real path
- Viewport × theme variant · console + network
- SPA/swap path, not only a hard reload

Sonnet

Throughput; drives MCP tools. Preview MCP is a local-only backup.

S — Spec audit

pixels + WCAG

- Screenshots at 375 → 768 → 1280, then pixel-diff
- Chrome MCP cannot lock those viewports — Playwright is required
- Skip S only with a computed-value proof

Opus

Vision required. Prose about a screenshot is not a pass.

Install once: `playwright + @axe-core/playwright + @playwright/test` then `npx playwright install chromium`

T = Sonnet + Playwright tests · U = Sonnet + Playwright MCP · S = Opus + Playwright screenshots

SECTION 06

Getting Started

What to subscribe to, what to buy, and in what order.

GETTING STARTED

Starting Out in 2026 — Single Developer

RULE ZERO

Don't try anything local until you have the cloud-based setup working.

$20/mo

Start here

Enough for initial testing — proving out the workflow, not running it.

Expect to outgrow it.

$100/mo

You'll bump here quickly

5× the usage. For most solo developers this becomes the working tier within weeks.

The realistic steady state.

$200/mo

Upgrade on the trigger

20× the usage. Go here mid-month when tokens are running short, or ahead of a known big task.

Prorated — no need to wait for renewal.

The ladder is the point: start cheap, move up on evidence, and don't buy hardware to solve a subscription problem.

QUICK POLL

Poll 3

Answer in Zoom — results will show live.

GETTING STARTED

Starting Out in 2026 — Team of Devs

Same rule first: per-seat cloud subscriptions before any hardware. Self-host only once the workflow is proven.

THE SEQUENCE

- One Max 5x seat per active dev; Max 20x for whoever runs agent fleets
- Self-host for privacy, compliance or predictable volume — not to save money
- Hybrid is the norm: steady work local, spiky work bursts to the API
- Most teams over-build local and underestimate how cheaply the API absorbs peaks

IF YOU GO LOCAL: SIZE FOR BATCHING

- Past ~4 concurrent users, vLLM or SGLang — not Ollama
- Throughput comes from continuous batching + PagedAttention
- VRAM = weights + KV cache. At 32K context, batch 4, the cache alone adds 20–40 GB
- Buy ≥20% headroom over your estimate — batching is what dies first when VRAM is tight

96 GB-class card

$14k–16k

RTX PRO 6000 Blackwell, street price

Practical serving node

~$35k–40k

Two cards + server, all-in

Not included

power · cooling · ops

Capex is the easy half

The entry point moved: NVIDIA doubled this card's MSRP to $16k on 13 Aug 2026, so a two-card node is now ~$35k, not ~$20k. That strengthens the rule above — self-host for privacy, not to save money. 24 GB consumer cards still can't hold the KV cache batching needs. Prices as of 21 Aug 2026.

RUNNING IT YOURSELF

Cloud vs. local: not a swap

Not a migration. The subscription buys capability the cluster cannot reach yet; the cluster buys guarantees the subscription cannot offer.

STAY IN THE CLOUD FOR

Frontier work and scale

- Frontier reasoning at the top of the pipeline — the model that plans, not just types
- Dozens of live contexts at once — six orchestrators plus their subagents is ~24, against one 48 GB pool
- No driver, no VRAM budget, no 3am debugging when a build breaks
- Capacity scales the moment the work does

RUN IT LOCALLY FOR

Privacy and repetition

- Code that contractually cannot leave the building
- High-frequency small calls — speech, transcription, embeddings, classification
- No rate limit, no queue, no model deprecated out from under you
- Fixed cost once bought — the marginal call is free

THE REAL MATH

The saving isn't $200 → $0. It's $200 → $100 — the subscription stays for the frontier work. So $2,650 of GPU pays back over ~26 months, and the M7 Pro / Max land late 2027, roughly a year before it gets there.

Buy local for the work the cloud can't do — not to cut the bill.

WRAPPING UP

What to Take Away

01 Strategic beats tactical — but only if you make it a habit

Ousterhout's investment mindset is the whole argument. AI makes tactical work cheaper, which makes the discipline matter more, not less.

02 The review loop is the product, not the writing

Write → review → fix → re-verify → merge. Every extra model usually finds something the others missed.

03 Guardrails are what make autonomy comfortable

Pre-commit hooks, an issue before every change, a cap on parallel work, and findings that get acted on rather than skimmed.

04 Start in the cloud, and climb the ladder on evidence

$20 to test, $100 as the working tier, $200 when a big task demands it. Buy hardware only once the workflow is proven.

05 Less typing, not less deciding

The judgment calls don't go away — they arrive faster, and they're still yours.

OVER TO YOU

Questions?

André Angelantoni · Founder, Performant Labs

Slides and the generator script are available — ask and I'll share the link.

SECTION A

Appendix: Running It Yourself

Not part of the talk — what to buy for local work, the cluster on my desk, and one a thousand times its size.

LANDSCAPE

Where Qwen3.8-27B ranks on Code Arena

Code Arena's WebDev leaderboard, mid-2026 — Arena score across 15 models. This is the model running on my own hardware.

Qwen3.8-27B places 9th of 15 — within 6% of the #1 score, and ahead of Gemini, GLM, and both DeepSeek-V4 variants.

Source: Arena AI leaderboard (arena.ai/leaderboard/code). Arena score, not token volume or market share — a different ranking from the provider chart elsewhere in this section.

RUNNING IT YOURSELF

Buying a New Laptop

TWO SPECS DECIDE LOCAL SPEED

GPU × memory bandwidth

- GPU cores do the arithmetic — prompt processing, image work, anything compute-bound
- Bandwidth sets tokens/sec — generating each token streams every active weight out of memory
- Unified memory lifts the 8–16 GB VRAM ceiling; bandwidth decides how fast that memory feeds the GPU

UNIFIED MEMORY BANDWIDTH, GB/s

HAS WINDOWS / LINUX CAUGHT UP?

Partly — AMD “Strix Halo”

- Ryzen AI Max+ 395: up to 128 GB unified, ~96 GB to the GPU
- Laptops scarce and dear — ZBook Ultra G1a ~$4,100 at 128 GB
- ~215–273 GB/s vs 546 GB/s on an M4 Max — fewer tokens/sec
- ROCm / Vulkan lag MLX; Windows runs 20–30% behind Linux

48 GB IS THE FLOOR

The model is not the only thing running. Weights plus KV cache plus your editor, browser, containers and the OS — 48 GB leaves room for all of it at once.

14" MacBook Pro
M5 Pro · 48 GB · 1 TB
$2,600–3,300

16" MacBook Pro
M5 Pro · 48 GB · 1 TB
$3,100–3,600

16" MacBook Pro
M5 Max · 48 GB · 2 TB
$4,400–5,000

OR WAIT?

M6 Pro / Max / Ultra are cancelled. The AI-focused M7 Pro and Max land late 2027.

Ranges reflect Apple's 2026 price rises during the memory shortage — the same 16" 48 GB config listed at $3,099 in March and $3,599 by July; street deals run $200–550 under list. M1 figure is the base M1 (M1 Pro 200, M1 Max 400). Roadmap per Bloomberg's Mark Gurman — reporting, not an Apple announcement: base M6 only this year, then M7 (H1 2027), M7 Pro / Max (late 2027), M7 Ultra (2028).

THE CLUSTER

What I run myself

Two hosts, two jobs. Jupiter sits on the desk and has the GPUs; Uranus is a Hetzner VM with none.

Jupiter

on the desk · Ryzen 9 9900X · Arc Pro B70

Flotilla	per-PR preview environments
Garage	S3-compatible store behind Flotilla
Zot	model-artifact registry
Ray head + llama-swap	serves the local models
2 GitHub runners	self-hosted CI
Coolify	the PaaS managing the rest

Uranus

Hetzner VM, Helsinki · no GPU

Harbor	container / CI image registry
Forgejo	self-hosted Git forge
Hermes	OpenAI-compatible agent gateway
4 GitHub runners	self-hosted CI
Grafana · Loki · Mimir · Tempo	observability stack
~40 more services	Mattermost, Penpot, LiteLLM, DDEV…

Harbor serves the pl-runner CI image to the runners on both hosts — the one thing that crosses the boundary.

THE CLUSTER

What the cluster is

ONE FRONT DOOR

Ray Serve on port 8000

OpenAI-compatible: chat · speech · health

Jupiter is the head

It must stay up. Io dual-boots, so it cannot own the head.

Io joins as a worker

If Io is gone, CUDA work waits — Jupiter keeps serving.

Ray does not see “two GPUs”

It sees named pins: jupiter_gpu, io_gpu, macbookpro_gpu.

The CPU pools are in Ray too

Jupiter 24 threads + Io 24 = 48. Proxies and CPU engines schedule here, not on VRAM.

GOTCHA

num_gpus=1 silently means Io only — the B70 is invisible to Ray's NVIDIA detector.

THE CLUSTER

The three machines

Jupiter

HEAD

`jupiter_gpu`

CPU

Ryzen 9 9900X · 24 threads
Ray scheduling, proxies, Piper (CPU TTS)

GPU

Intel Arc Pro B70 · 32 GB

JOB

Local chat / coding LLM (Qwen3.8 on vLLM XPU)
Cluster control plane

ALSO

AMD iGPU drives the desktop — not a Ray resource
9 drive bays · 30 TB RAID

Io

WORKER

`io_gpu`

CPU

Ryzen AI 9 HX 370 · 24 threads
Listeners, some CPU fallback

GPU

RTX 5080 · 16 GB

JOB

CUDA media — Kokoro / XTTS / Chatterbox / Whisper
One GPU engine at a time

NOTE

Dual-boots, so it can never hold the head role

MBP M1

OPTIONAL

`macbookpro_gpu`

CPU

Apple Silicon
Dev workstation, not the head

GPU

M1 GPU (unified memory)

JOB

Light / on-laptop work — e.g. Kokoro-class TTS

NOTE

Not the 32 GB coding card, and not the 5080

Zot: the local model shelf

ZOT

An OCI registry on Jupiter

The on-LAN cache. Io, Jupiter and the M1 pull weights from here instead of from Hugging Face.

69

model artifacts on disk today. Four OS installers also live here, not counted as models.

CATEGORY	NAMESPACE	N	PURPOSE
LLM (general)	`general/`	17	Chat / reasoning
LLM (coding)	`code/`	9	Coding specialists
Vision / VLM / OCR	`vision/`	17	Image+text, OCR
TTS	`tts/`	18	Speech out
STT	`asr/`	3	Speech in
Other	`supir · legacy · test`	5	Upscale, older builds, fixtures
Total		**69**	*model artifacts*

THE CLUSTER

I Built My Own GPU Cloud (and you can too)

Two desktops, a laptop, and a switch. One OpenAI-compatible endpoint on the LAN — no colo, no rack, no datacenter card.

WHAT I BOUGHT

- **Jupiter — head node**
  Ryzen 9 9900X · 24 threads
- **Intel Arc Pro B70** — $1,250
  32 GB · the coding card
- **Io — worker node**
  Ryzen AI 9 HX 370 · 24 threads
- **NVIDIA RTX 5080** — ~$1,400
  16 GB · CUDA media work
- **Storage in Jupiter**
  9 drive bays · 30 TB RAID
- **MacBook Pro M1** — $0
  already owned — joins as a worker

WHAT I INSTALLED

- **Ray + Ray Serve**
  the single front door on :8000, OpenAI-compatible
- **vLLM (XPU build)**
  the chat / coding LLM, pinned to the B70
- **CUDA media stack**
  Kokoro · XTTS · Chatterbox · Whisper, pinned to Io
- **Piper**
  CPU text-to-speech — no GPU required
- **Zot**
  OCI registry: ~60 model artifacts cached on the LAN
- **Named resource pins**
  jupiter_gpu · io_gpu · macbookpro_gpu

~$2,650 of GPU across both nodes — 48 GB of VRAM in total, spread over two machines rather than concentrated in one expensive card.

Prices as paid, not MSRP — the B70's $949 launch price was not what it cost in the channel. Excludes the host machines, drives and networking.

THE CLUSTER

The Other End of the Scale

Alec Glover — "I Built My Own GPU Cloud (and you can too)" and "everything i learned building 100 GPU servers." Same problems, four orders of magnitude up: zero to 1,000 GPUs, hand-built in a warehouse.

COMPONENT	WHAT HE USED	WHY IT MATTERED
GPU — consumer	**NVIDIA RTX 5090**	Newest generation gives the best depreciation and rental longevity
GPU — enterprise	**NVIDIA B200 / H200**	SXM and OAM backplanes at the high end, not PCIe
Motherboard	ASRock Rack	Native SlimSAS and MCIO — he moved off cheap PCIe risers
Chassis	10U / 7U; barebones from SuperMicro, Gigabyte, ASUS	Airflow is the whole reason for the form factor
Power	Power Delivery Board + redundant 1600 W PSUs	6 kW+ per server, five PSUs in parallel, 80% load rule
Cabling	SlimSAS / MCIO, PCIe Gen 5, redrivers	Signal integrity — bad risers show up as AER errors and crashes
Storage & network	U.2 NVMe backplanes; 10G → 100G	PXE boot and bare-metal provisioning to scale the fleet

His racks run 20–40 kW with hot-aisle containment. Mine runs off a wall socket. The engineering questions are identical — VRAM per dollar, airflow, power headroom, signal integrity — the budget is not.