TEAM400

ACCESS DETAILS

Login	IT Training Event
Password	itTraining2026

AFTERNOON SESSION

Where it gets interesting.

Advanced Claude AI, AI Agents and Coding · Team 400

bit.ly/Claude400Slides

https://claudeday.team400.ai/
Password: team400

Build password: build400

Afternoon session · 1:30 – 5:30

Afternoon Session

1:30 – 2:15	AI Agents, Coworking, OpenClaw
2:15 – 2:30	Demos
2:30 – 3:30	Exercises
3:30 – 4:00	Break
4:00 – 5:00	Exercises continue
5:00 – 5:30	Endless Q&A — every question answered

Demo: Autonomous Agents with Co-Work

The Demonstration

- Event Manager AI Agent

Demo: Claude Code - Workspaces - Marketing Workspace

The Demonstration

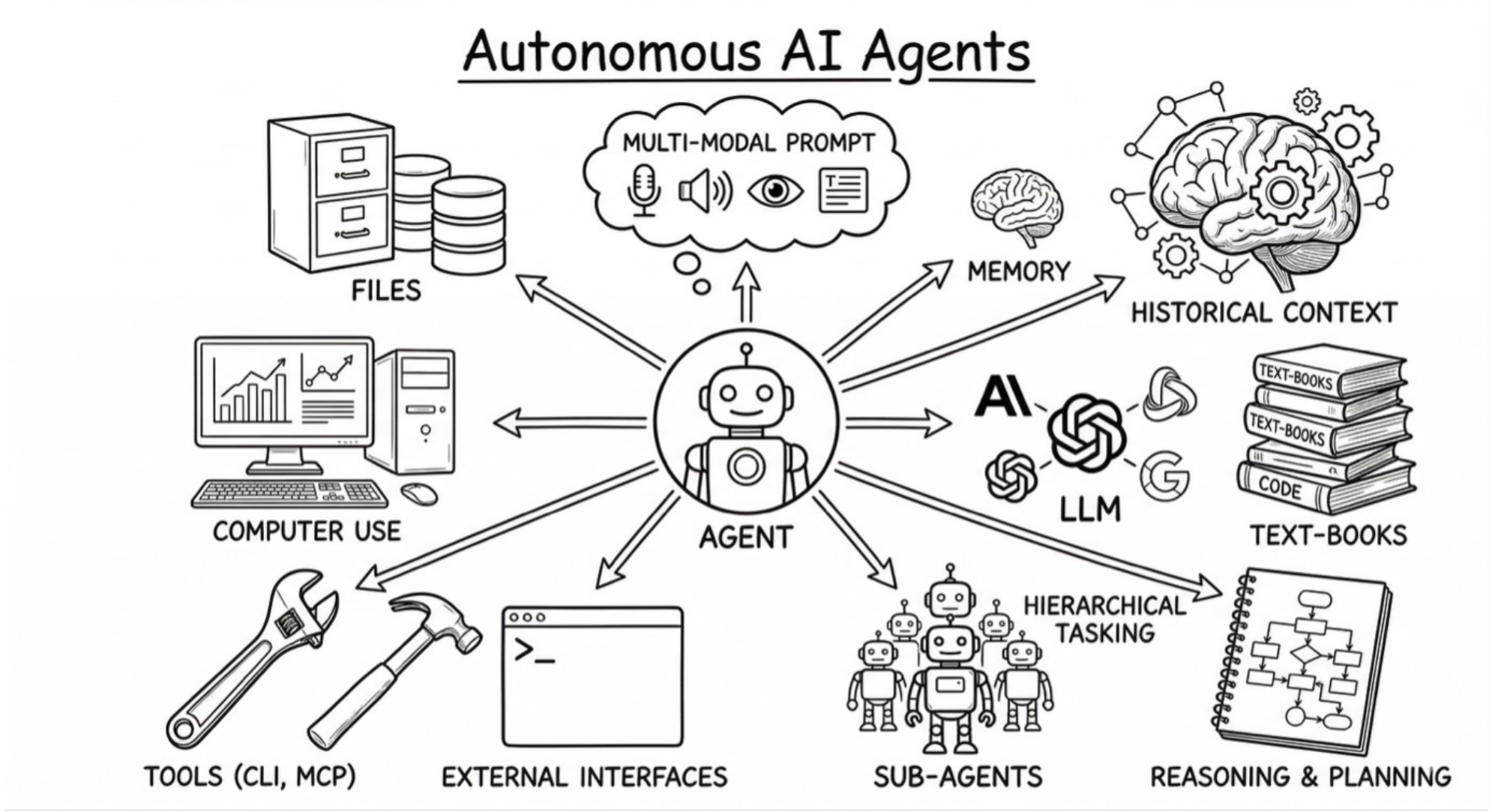
- Marketing Benchtop Workspace

Demo: Claude Code - Workspaces - Fast SEO Website

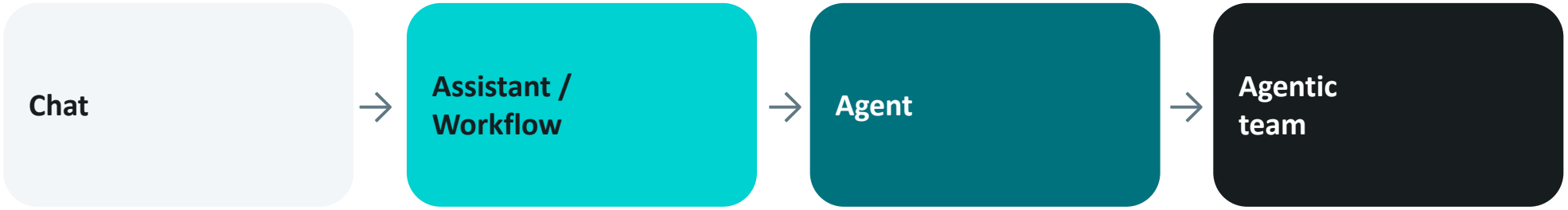
The Demonstration

- Fast SEO Website

Autonomous AI Agents


Autonomous AI Agents
MULTI-MODAL PROMPT
FILES
MEMORY
HISTORICAL CONTEXT
COMPUTER USE
AGENT
LLM
TEXT-BOOKS
TEXT-BOOKS
TEXT-BOOKS
CODE
HIERARCHICAL TASKING
TOOLS (CLI, MCP)
EXTERNAL INTERFACES
SUB-AGENTS
REASONING & PLANNING

It’s a dial, not a switch



left = predictable • right = powerful

More power, more variance — even the “predictable” step is the model improvising.

THE HONEST ROI LINE

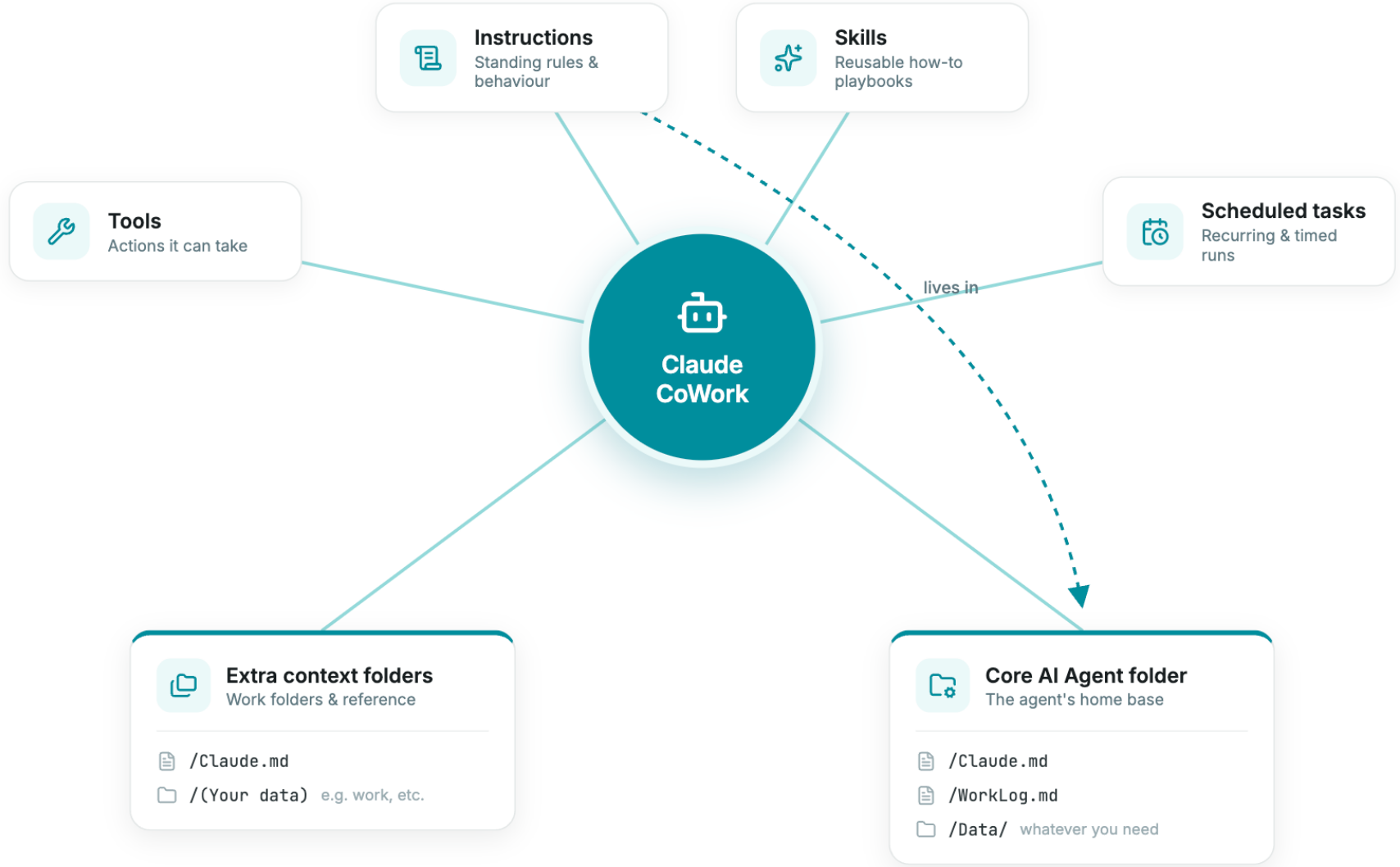
Agents take time to get working well.

An Investment

Skills · Connectors · MCP

The three things everyone mixes up

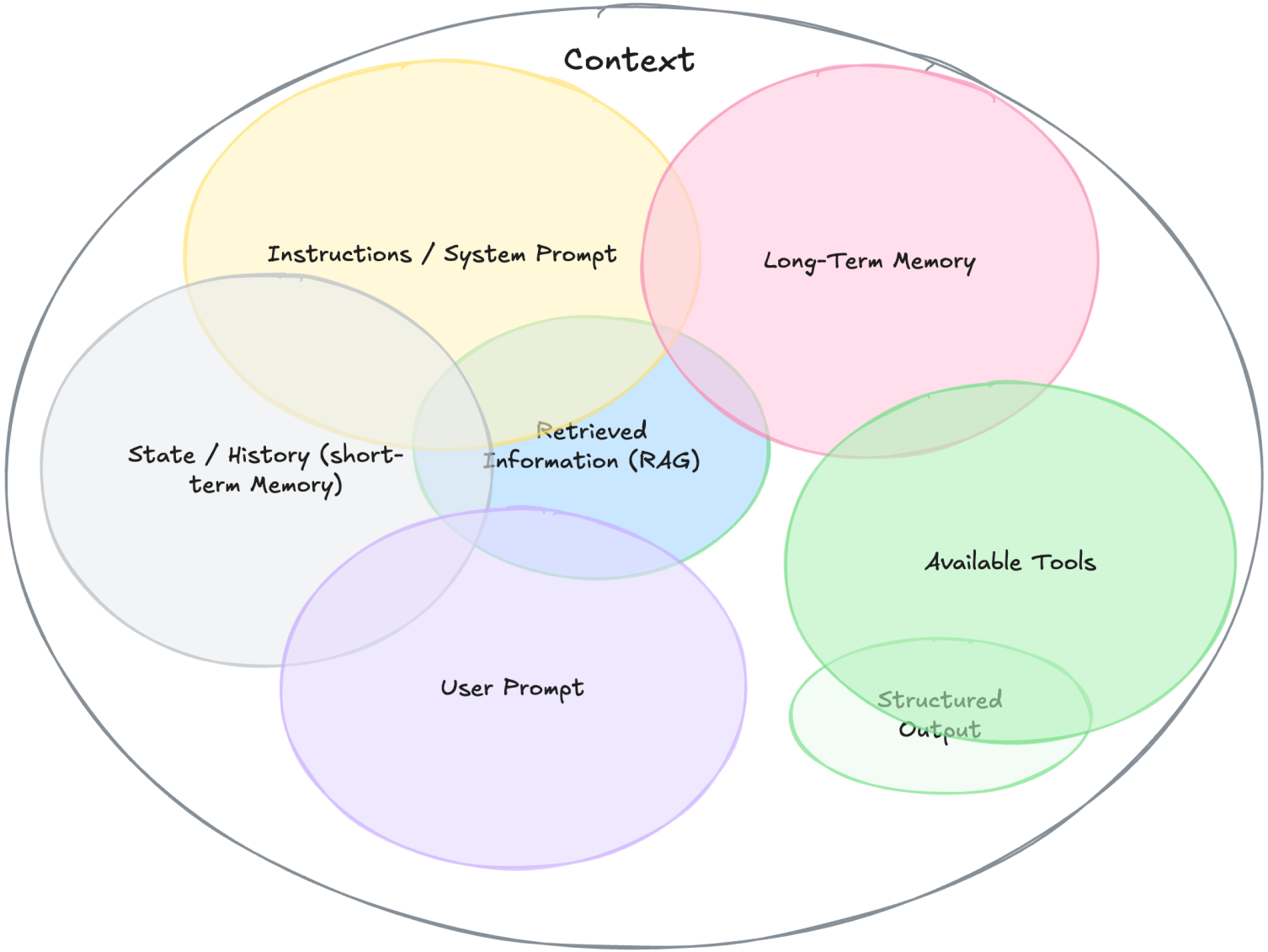
How we can build agents in CoWork



SECTION 2

Context & AI Agents

Context Engineering



SECTION 8

Let’s design one.

Designing a AI Agent

Four questions, every time

1 · Scope — what's the one job? Narrow always beats broad

2 · Tools — what does it touch? Files, email, systems

3 · Knowledge — what must you teach it? Context, SOPs, steps

4 · Guardrails — what must it never do without a human?

(Initially be very specific, you can loosen over time)

Brief it like a new hire

Same four questions you'd ask any new staff member:

- → Responsibilities
- → Tools & access
- → Training & SOPs
- → Limits & sign-off

What good scoping looks like

A good first agent

- Narrow + repeatable + clear guardrails

A bad one

- Vague + broad + “just do my job” — no edges, nothing to test

The move

- Start embarrassingly small. Make it work. Then widen it.

A scoped example

Job Draft replies to refund requests

Tools Inbox (read), drafts folder

Never Send without my sign-off

Done when Draft in 2 min, in my tone

Practices that work

1 Keep it simple

The fewest moving parts that actually work

2 Discuss Plans & Iterate on the workflows

See the plan before it acts - never let it work in the dark

3 Great instructions = great tools

A clear, well-written skill beats any clever prompt trick

THE THROUGH-LINE

Brief it simply. Make it show its work. Write the instructions like an SOP - then dry-run before it ever acts.

Demo and Exercise - Building an AI Agent

- Go to hands-on-exercises.docx
- Follow instructions for:
  - Exercise 1
  - Exercise 2

SECTION 11

Meet  OPENCLAW

When you want to own the whole thing

It Actually Does Things

Most AI tools answer questions. OpenClaw solves problems.
You don't ask it for information and then go do the work yourself. You describe what you need, and it goes out and does it — configures systems, writes automations, sets up monitoring, sends messages, manages workflows. Then it keeps doing it, automatically, going forward.

You talk to it
"Our deploy notifications aren't going to the right Slack channel and I need a summary of failures every morning at 9am"
No tickets. No dev team. No backlog. Just a conversation.

It builds the solution
OpenClaw figures out the tools it needs, writes the configuration, sets up the scheduled task, tests it, and tells you when it's done.
No developer required. No code to maintain.

It keeps running
Tomorrow morning at 9am, you get your failure summary. Every morning after that too. If the deploy channel changes, tell it and it adapts.
Mini solutions, built on the fly, running forever.

It's your COO and CTO in one - for the things that don't need a human
Think about all the small operational tasks that never get prioritised: fixing notification routing, generating daily reports, triaging alerts, keeping docs updated, checking that backups ran. Each one is too small for a project, too important to ignore. OpenClaw handles them by conversation. That's the real value — not the AI itself, but the hundreds of small automations it builds for you without needing a developer.

What Can It Actually Do?

Real examples of what people ask OpenClaw to do every day

“Monitor our deploys and Slack me if anything fails”
OpenClaw watches your CI/CD pipeline and sends you a summary every morning — failures get flagged immediately.

“Summarise this week’s support tickets”
It reads through your ticket queue, groups issues by theme, and drafts a report you can send to leadership.

“Schedule a meeting with everyone on this project”
It checks calendars, finds a time that works, creates the invite, and adds the agenda you described.

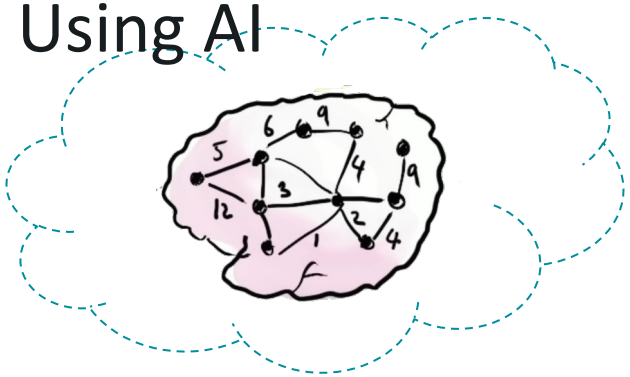
“Send the finance team our updated numbers”
It pulls the latest data from your dashboard, formats it into an email, and sends it to the distribution list.

“Every Friday, email me a summary of what the team shipped”
It sets up a recurring task — checks your repos and project boards, generates the summary, and sends it on schedule.

“Find out why the website is slow and tell me what to fix”
It checks your monitoring tools, identifies the issue, and gives you a plain-English explanation of what’s wrong and what to do.

These aren’t hypothetical. These are the kinds of tasks people use OpenClaw for today. You describe the problem in plain English, it figures out the solution.

Different Modes for Using AI



Modes:

Chat	Cowork	Code	Claw
Conversational Thinking	Automate Work Tasks	Build things with code	Do anything autonomously

Has its own computer/phone
Could be initiated by anything/anyone
Could be given access to any information
Acts autonomously

What's the big deal?

Autonomy

1

Own Computer

Each agent runs in its own isolated environment with dedicated resources

2

Full Permissions

Agents operate with full autonomy — no waiting for human approval at every step

3

Multi-trigger

Kick off from chat, cron jobs, webhooks, CI pipelines, or other agents

Do I need a MacMini?

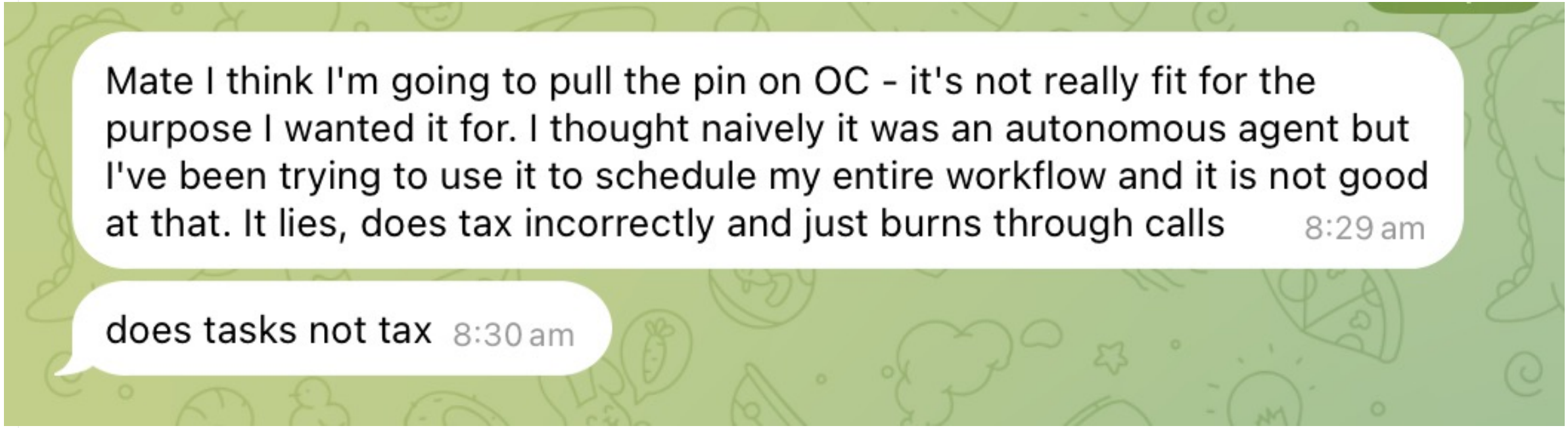
How much will my tokens cost?

CRM "Build a CRM that automatically scans my Gmail and Google Calendar to discover contacts from the past year. Store them in a SQLite database with vector embeddings so I can query in natural language ('who do I know at NVIDIA?' or 'who haven't I talked to in a while?'). Auto-filter noise senders like marketing emails and newsletters. Build profiles for each contact with their company, role, how I know them, and our interaction history. Add relationship health scores that flag stale relationships, follow-up reminders I can create, snooze, or mark done, and duplicate contact detection with merge suggestions. Link relevant documents from Box to contacts so when I look up a person, I also see related docs."

Newsletter and CRM Platform Integration "Connect my AI assistant to Beehiiv (newsletter platform) and HubSpot (sales CRM). For Beehiiv: track subscriber count, growth rate, churn, per-post open rates and click rates, and subscriber segments. For HubSpot: sync deals (stage, value, active deals), contacts, and pipeline status. Cache both locally in SQLite for fast queries. Feed all data into the business advisory council for holistic cross-platform analysis alongside YouTube, social media, and other signals."

Meeting Action Items (Fathom) "Create a pipeline that polls Fathom for meeting transcripts every 5 minutes during business hours. Make it calendar-aware so it knows when meetings end and waits for a buffer before checking. When a transcript is ready, match attendees to my CRM contacts automatically, update each contact's relationship summary with meeting context, and extract action items with ownership (mine vs. theirs). Send me an approval queue in Telegram where I can approve or reject each item. Only create Todoist tasks for approved items. Track other people's items as 'waiting on' but exclude internal team members from the waiting-on list (only track external contacts). Run a completion check 3x daily (8am, 12pm, 4pm) that shows what's overdue, pending, and what I'm waiting on from others. Auto-archive items older than 14 days."

Real World


Mate I think I'm going to pull the pin on OC - it's not really fit for the purpose I wanted it for. I thought naively it was an autonomous agent but I've been trying to use it to schedule my entire workflow and it is not good at that. It lies, does tax incorrectly and just burns through calls 8:29 am
does tasks not tax 8:30 am

Agents are folders. Folders move.

- Both Claude and OpenClaw express an agent as files - skills, instructions, reference data
- So an agent you build as a folder in CoWork is largely portable
- Prototype on your desktop in the nice UI, then lift it onto your always-on server
- That's not marketing - it's just what happens when everything is plain text

SAFE ROLLOUT

Risks with agents

Roll this out without getting burned

Three places it can go wrong

1 · The model

- Confidently wrong, or talked into doing something daft

2 · The environment

- The tools it can touch - it can only do damage with the tools you gave it, so give it the fewest it needs

3 · What it reads

- Emails, web pages and PDFs can carry hidden instructions — “prompt injection.” Treat untrusted text like a new hire who believes everything they read

Read-only → draft → gated writes

STAGE 1

Read-only

Reads and tells you. No actions - let it prove it’s right.

STAGE 2

Draft

It drafts - the email, the proposal - a human presses send.

STAGE 3

Gated writes

It acts on its own, with hard approvals on anything irreversible.

Earn each stage - start read-only. Most people fail by starting at Stage 3 on day one.

Guardrails and Boundaries

HARD STOPS

Boundaries

Hard coded boundaries for what the agent can do based on its auth.

FUZZY STOPS

Guardrails

We ask the AI to behave in a certain way.

ENFORCED IN CODE

- Authorisation via login / token scope
- Custom MCPs or APIs
- Custom CLIs
- Customised agents

The operating rule

- Start with the smallest layer that works - often a prompt, not an agent
- Add context deliberately, not by the bucket
- Add tools selectively
- Keep the writes gated until trust is earned
- Verify what it produces
- Go fully autonomous only when the job genuinely needs it

Excited and sensible.

AI Slop Still Applies

Write like a human

- Edit AI output - don't just copy-paste. Add your voice, cut the filler, rewrite flat sentences
- Inject specifics - real names, concrete numbers, actual examples from your experience
- Break the pattern - vary sentence length, use fragments, add personality and opinion
- Remove the buzzwords - delete delve, tapestry, it's important to note, leverage

Prompt better, get better output

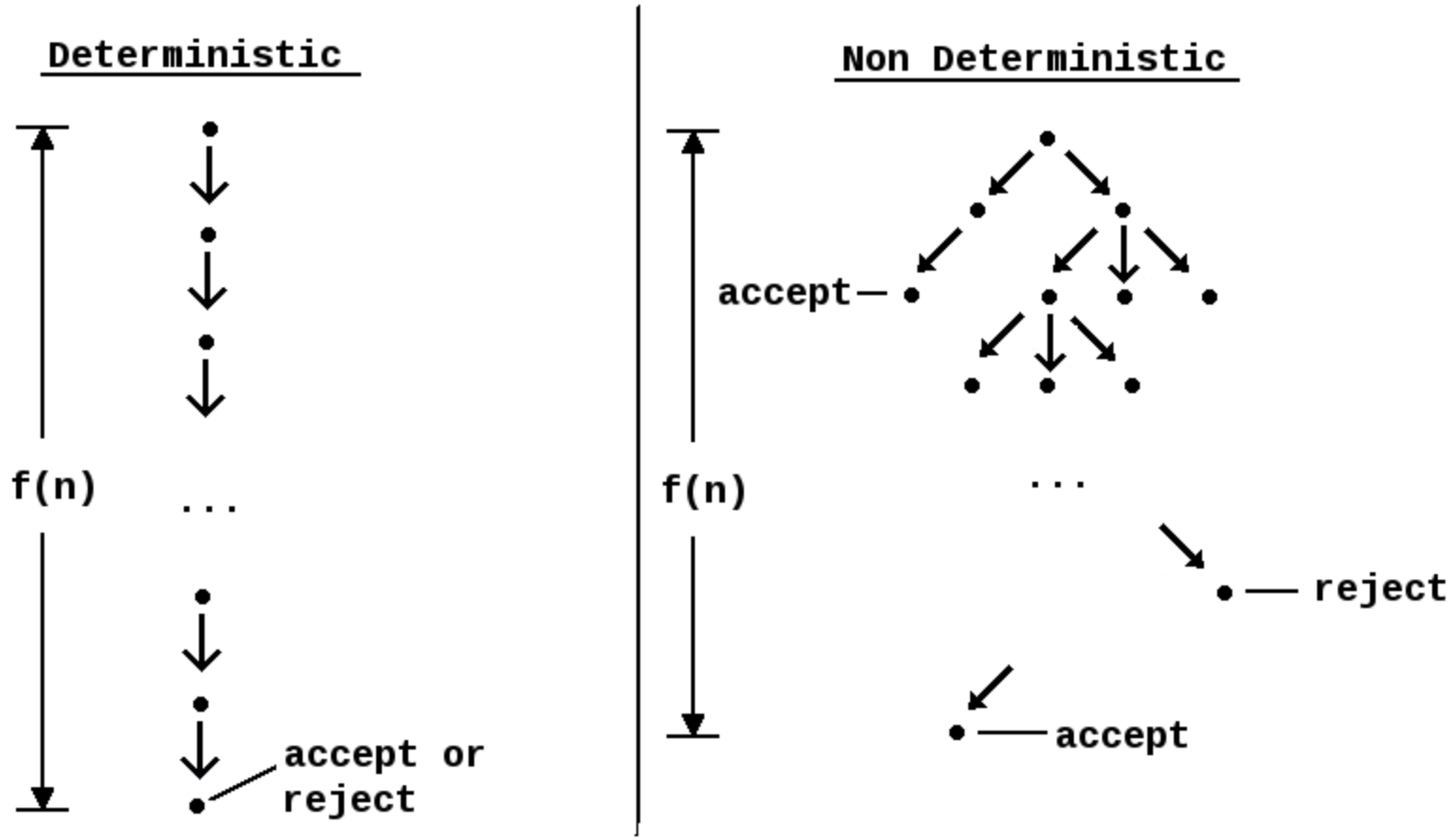
- Give Claude your actual context - audience, purpose, constraints, tone
- Show examples of what "good" looks like before asking it to write
- Ask for a specific structure - "3 paragraphs, no intro, lead with the data"
- Tell it what NOT to do - "No bullet lists, no filler phrases, no generic intros"

Verify everything

- Check every fact, stat, and quote - AI confidently fabricates all three
- Google any source it cites - if you can't find it, it probably doesn't exist
- Run the "would I stake my reputation on this?" test before publishing
- Cross-reference with primary sources, not just what the AI tells you

Use AI as a thinking partner

- Start with your own outline or draft - don't ask AI to generate from scratch
- Use AI to challenge, improve, and refine YOUR ideas, not replace them
- Ask it to poke holes in your argument rather than write one for you
- Treat output as a first draft that needs your expertise, not a finished product


Deterministic
f(n)
...
accept or
reject
Non Deterministic
accept
f(n)
...
reject
accept

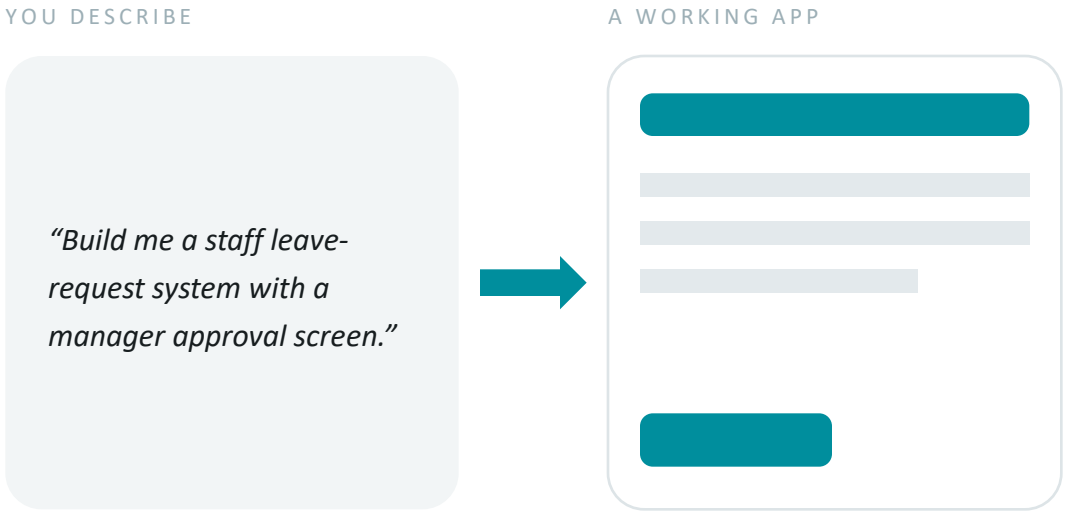
SECTION 12

Vibe Coding

Presented by Libin

What is vibe coding?

Vibe coding means creating software by explaining what you want to AI. The AI writes the code while you direct, test and refine the result.



You do not need to know how to write all the code. You need to clearly understand the problem and desired outcome.

What can you build?

- Internal business tools
- Customer portals
- Dashboards and calculators
- Booking and workflow systems
- AI assistants
- Websites and prototypes

Example - turn a spreadsheet used to track customers into a searchable internal application with forms, reporting and reminders.

Start with a small, specific problem. Do not begin by trying to build an entire enterprise platform.

Version control with Git

Git creates a recoverable history of the project. People need to understand:

- Repositories, commits and branches
- How to return to a working version
- Never make large changes without committing first
- Review what the AI changed
- Keep secrets and generated files out of Git
- Use GitHub as the controlled source of truth

CORE LESSON

Commit before every significant AI-generated change.

Development, staging and production

Keep three separate environments:

Development
Where you build and break things

Staging
Where you test the complete system

Production
Where real users and real data exist

Never allow an AI coding agent to experiment directly against production.

CORE LESSON

Build locally, test in staging, deliberately release to production.

CI/CD and controlled deployment

Deployment is a repeatable process — not copying files or telling the AI to “put it live”.

- GitHub-connected deployments
- Automated builds and tests
- Preview deployments
- Deployment logs
- Rollbacks
- Production approval steps
- Database migrations

> “
>
> *Friends don’t let friends right-click deploy.*

CORE LESSON

Code reaches production through a controlled pipeline.

Authentication and authorisation

Authentication determines who the user is

Authorisation determines what that user can see and do

A login screen does not make an application secure. The server must enforce permissions on every protected operation.

- User roles
- Ownership of records
- Server-side permission checks
- Admin access
- Session handling
- Testing as different users
- Preventing one user from accessing another user’s records

CORE LESSON

Hiding a button is not authorisation.

Databases and data design

An application is more than its interface.

- Tables, records, relationships and IDs
- Data validation
- Who owns each record
- Backups and restoration
- Development data versus production data
- Database migrations
- Avoiding destructive schema changes
- Minimising personal or sensitive information

CORE LESSON

Design the data and access rules before building screens around it.

Secrets and configuration

API keys, database passwords and service credentials must never appear in:

- ✗ Source code
- ✗ Git repositories
- ✗ Browser-side JavaScript
- ✗ Prompts
- ✗ Screenshots
- ✗ Error messages

Instead: use environment variables and platform secret stores.

> CORE LESSON
>
> Anything sent to the browser or committed to Git must be treated as public.

Client-side versus server-side code

Client-side

Runs on the user’s device.

Can be inspected, copied or modified by anyone using the application.

Server-side

Runs in a controlled environment.

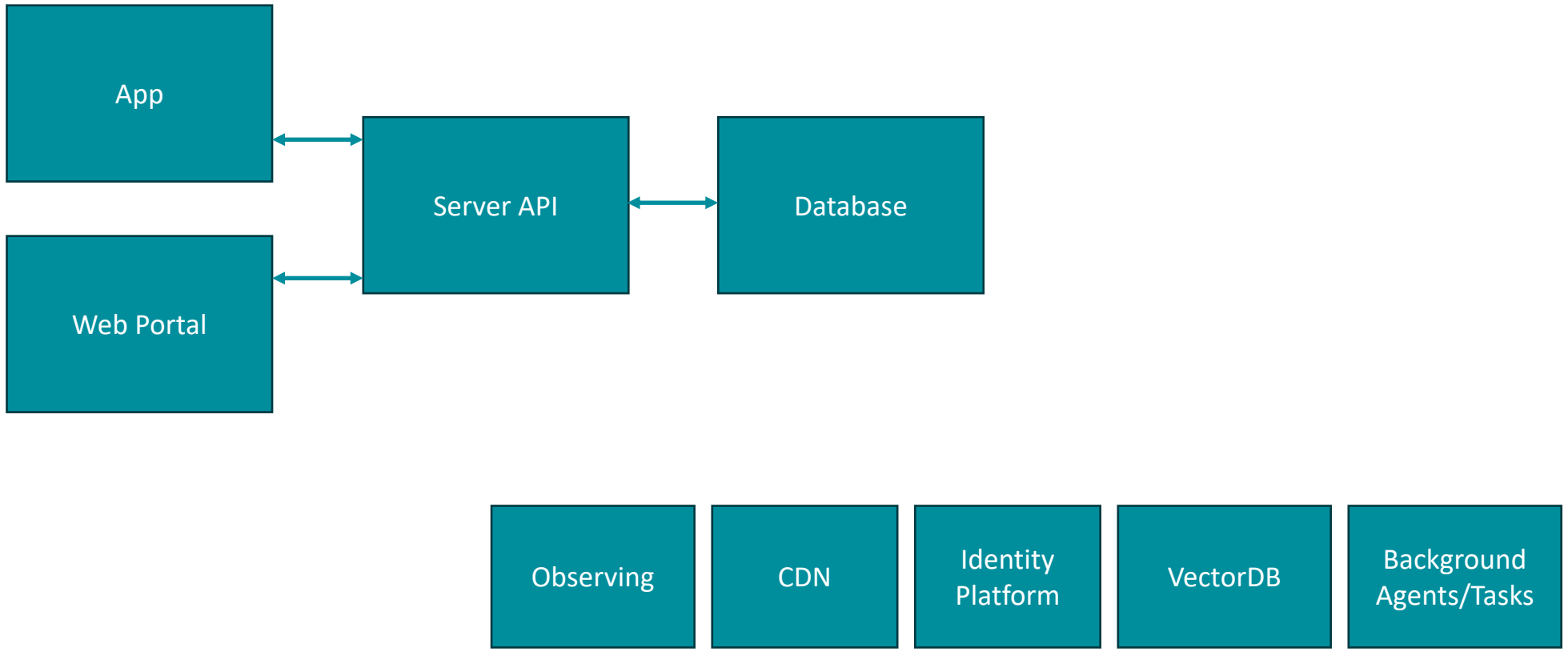
Where sensitive rules, credentials and permission checks belong.

If it matters for security or correctness, it must be enforced on the server — not in the interface.

CORE LESSON

The browser cannot be trusted to enforce security.

Simple with Architecture



Scope and risk boundaries

Not every application is suitable for casual vibe coding.

Good early projects

- Internal calculators
- Content tools
- Simple workflow systems
- Prototypes
- Dashboards using non-sensitive data
- Low-risk administrative tools

Needs professional review

- Payments
- Health or financial information
- Sensitive customer data
- Public user accounts
- Complex permissions
- Critical business operations
- Systems connected to production databases
- Anything where failure could create legal, financial or safety consequences

Demo and Exercise - Vibe Coding

- Follow instructions for:
  - Claude_Code_Setup_Guide_Recreated.pdf
  - Go to hands-on-exercises.docx
  - Exercise 3 - Vibe-code a website with a blog (and deploy it free), using the designs system we already have.

After the break, you build

https://bit.ly/Claude400Arvo

https://bit.ly/Claude400Slides

Notes:

- Presenters will be walking around to help as much as possible.
- If you get stuck with something, it can be good to ask AI to help you solve that problem. You can ask Claude what you're trying to achieve and say, "The instructions aren't working for me. Can you help me find another way to do this?"
- Meet and discuss with the person next to you.