How always-on agents work

Lee Robinson · Stanford CS146S · October 8, 2026



Safe Harbor Statement

This presentation may contain forward-looking statements about potential future Company products, features, functionality, plans and timelines. These plans reflect the Company's current expectations only and are subject to change. The Company may change, delay or discontinue any such plans at any time and has no obligation to update this information. Nothing presented constitutes a commitment to deliver or continue delivering any particular product, feature or functionality, or to do so on any particular timeline. Purchasing decisions should be based only on products, features and functionality that are currently commercially available.



You built this in week one

```
messages = [system_prompt, user_message]

while True:
    reply = llm(messages, tools=TOOLS)
    messages.append(reply)
    if not reply.tool_calls:
        print(reply.text)
        break
    for call in reply.tool_calls:
        messages.append(run_tool(call))
```

This agent stops when you close the terminal.

Today: how to build one that keeps running, and why that makes it a different kind of product.



Your agent vs. an always-on agent

Your week-one agent	An always-on agent
Runs in your terminal	Runs on a server, so it survives crashes, deploys, and a closed laptop
Uses your laptop	Has its own cloud computer: a small virtual machine with a desktop, browser, and files
Remembers only the messages list	Saves its state, memory, and skills, and summarizes old messages
Prints every model reply	Only messages you when it calls a send tool, so it can stay quiet
Starts when you type	Starts on events: your messages, schedules, Slack, phone calls, finished work
Does one task, then exits	Works for months with the same memory, across many conversations



Agenda

01	How we got here
02	What changed in the models
03	Inside an always-on agent
04	The harness
05	Context engineering
06	Where this is going
07	Q&A



Part 1
How we got here



How developers work with AI

How much the AI does

2022–23

Copy and paste

Chat in a browser tab. You paste code in and copy answers back by hand.

2024–25

Agents in the terminal

The agent edits files and runs commands in a loop. You approve each step.

2025–26

Apps built for agents

Many agents run in parallel, in the background or the cloud. You review the finished changes.

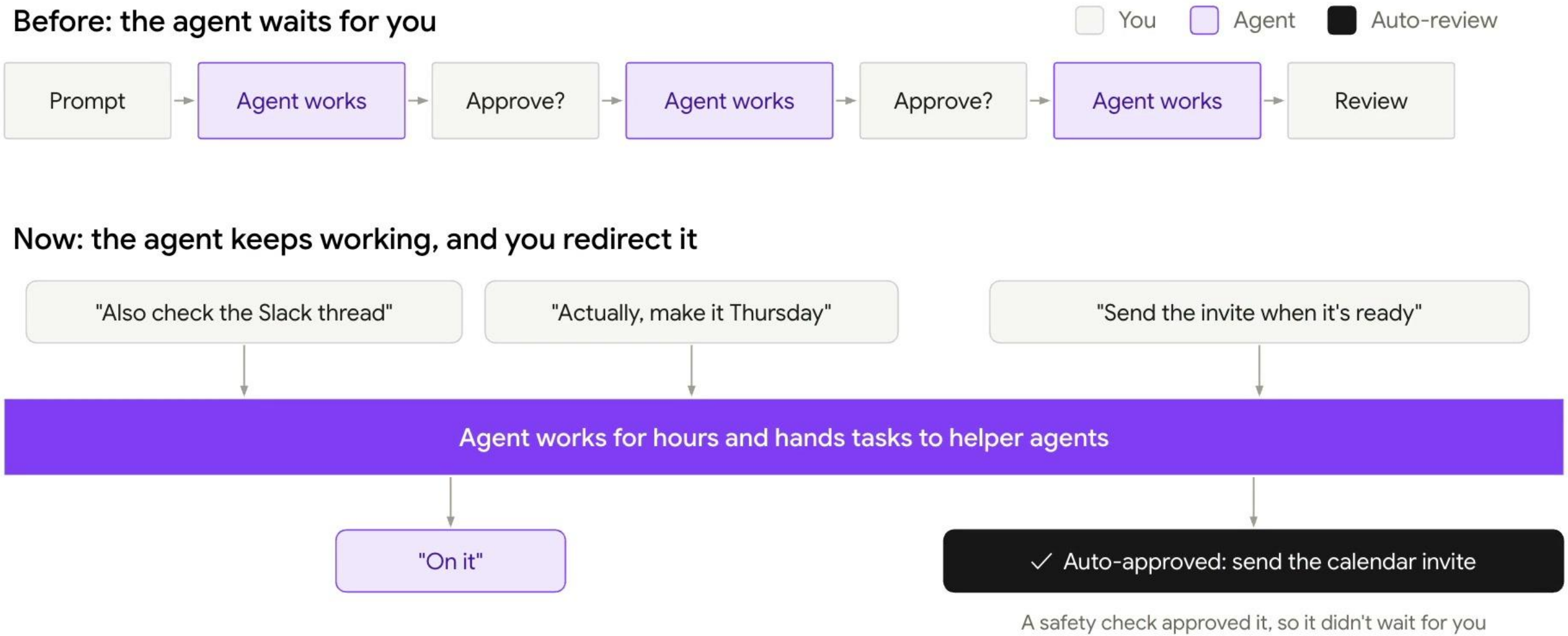
Now

Always-on agents

You text an agent that has its own computer, tools, and memory. It keeps working after you leave.

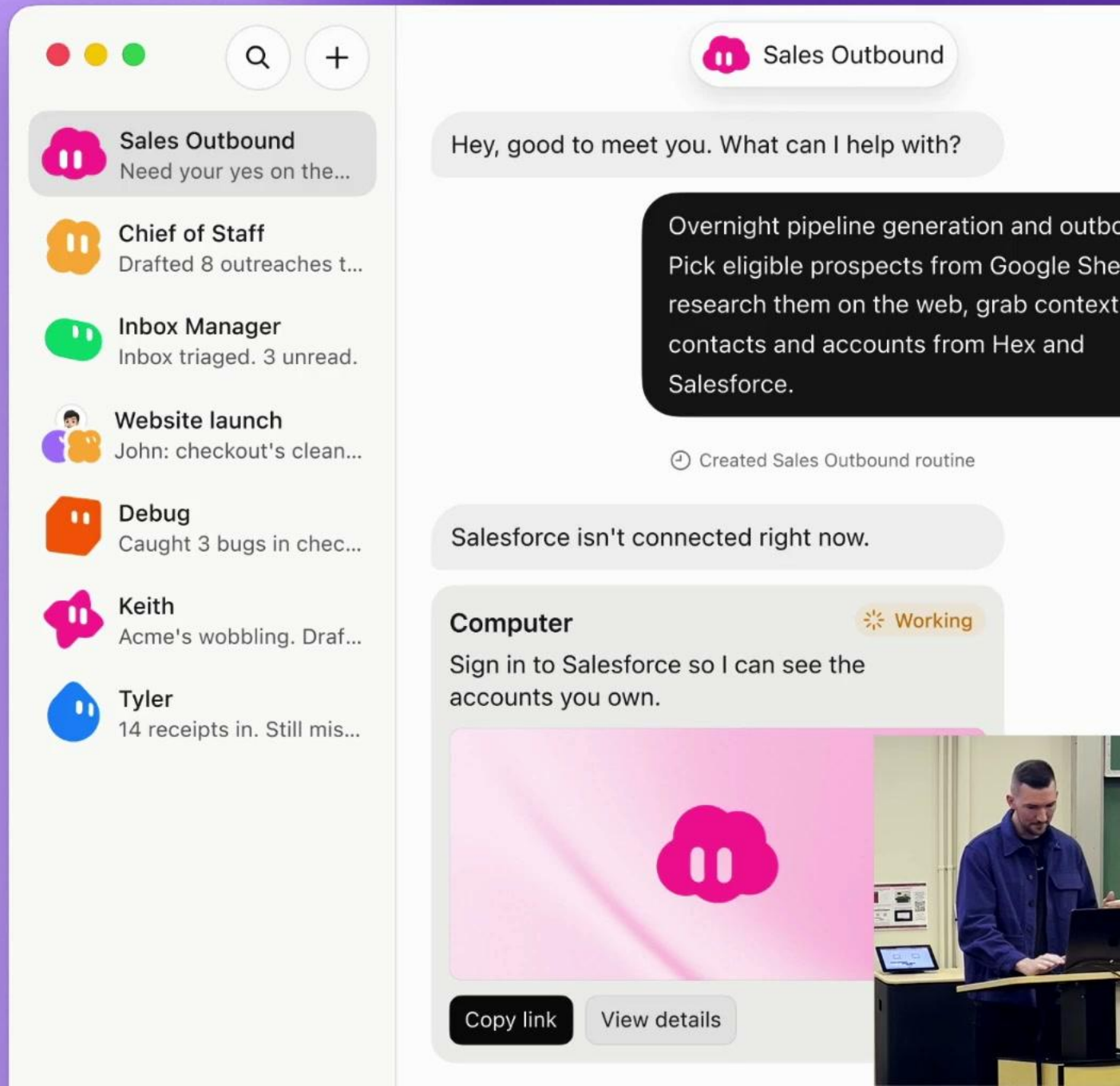


From approving every step to steering



As models get more capable, the interface gets simpler.

Our implementation of this always-on agent is called a Bot.



Part 2
What changed in the models



Six new model capabilities

1. Work for hours
Models can now work for hours and follow complex instructions the entire time.

2. Hand off to helpers
A main agent gives tasks to helper agents, each with a fresh context, so the total work isn't limited by one context window.

3. Use real tools
A shell, files, a browser, and app APIs through MCP: the same tools you use on your computer.

4. Use a computer
They can look at a screen and click, so they can use apps that have no API.

5. Keep knowledge in files
Skills and memories are stored as plain files that the agent reads and updates itself.

6. Learn by watching
Show it how you do a ta
its computer, and it sav
steps as a skill it can reu



What the agent knows lives in files

```
memory/
  profile.md              always in the prompt
  log/2026-10.md          dated facts, one per line
skills/
  linkedin-sourcing/
    SKILL.md              "Use this when…"
  weekly-metrics/
    SKILL.md
routines/
  morning-brief           prompt + weekdays 8:00
```

Simplified example

- Models are extremely good at using Linux and running shell commands
- Plain text, so you and the agent can both read, search, and edit it
- The agent keeps it up to date. Tell it a preference once and it saves it to memory
- Skills load only when needed, so having many of them is cheap

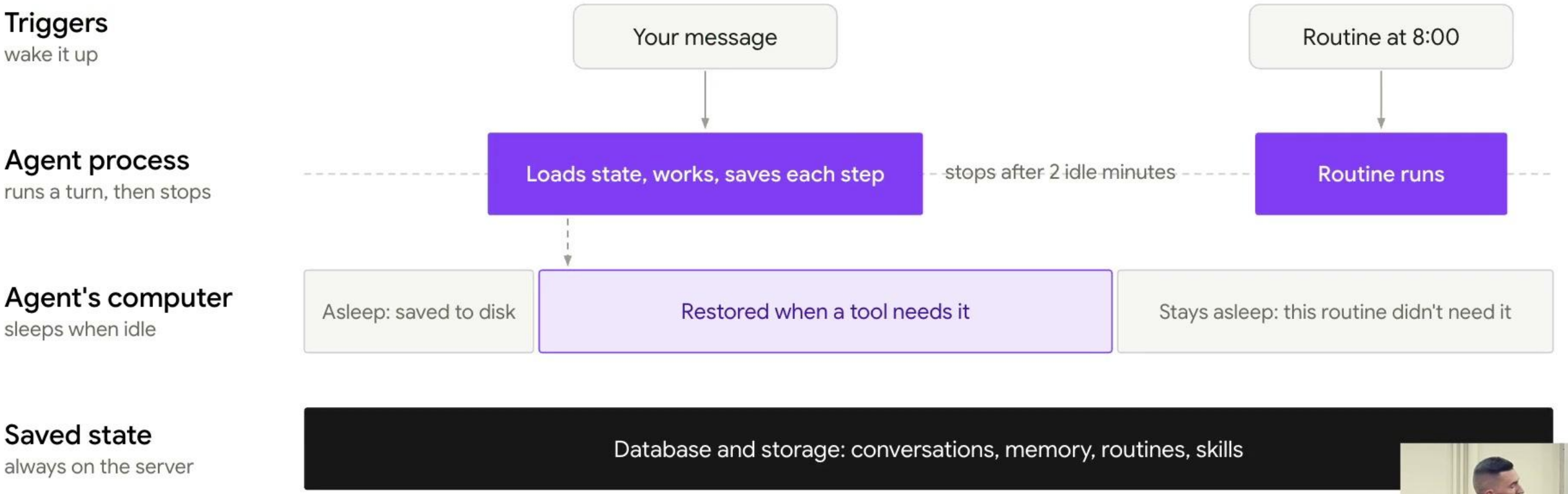
“Everything is computer”



Part 3
Inside an always-on agent



How does it stay always on?

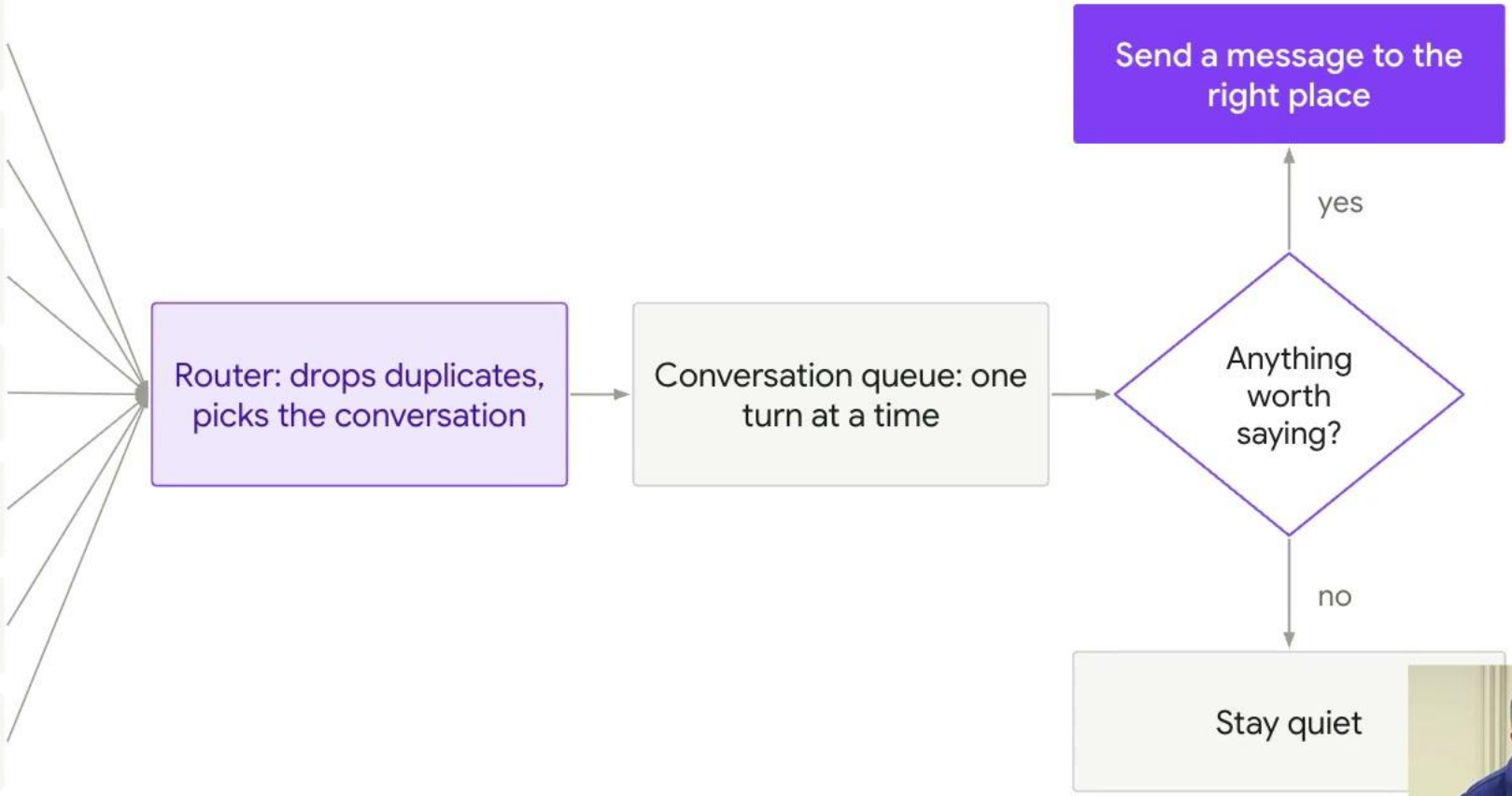


When nothing is happening, nothing runs. Everything that matters is saved, so any trigger can bring it back.



What wakes the agent

- You, on any app
- A routine: schedule or webhook
- A Slack mention, DM, or thread
- A phone call or meeting
- Another agent
- Background work finishing
- A scheduled check-in



Every wake-up records where it came from, so the agent can tell whether you asked, a routine ran, or an outside service sent



The agent's computer

A Firecracker virtual machine · one per user · saved to disk when idle

A Linux (Debian) container, kept running by a supervisor

File server	Files, app connections, live desktop view
Command runner	Shell commands, files, screenshots, clicks
Screen 1	A virtual display with its own Chrome
Screen 2, 3, ...	One more screen per active agent

Files, installed tools, and browser logins all stay between sessions

Going to sleep

A few minutes after the last activity, never mid-turn

1. Files sync to cloud storage
2. Memory and disk are saved as a snapshot
3. The machine stops

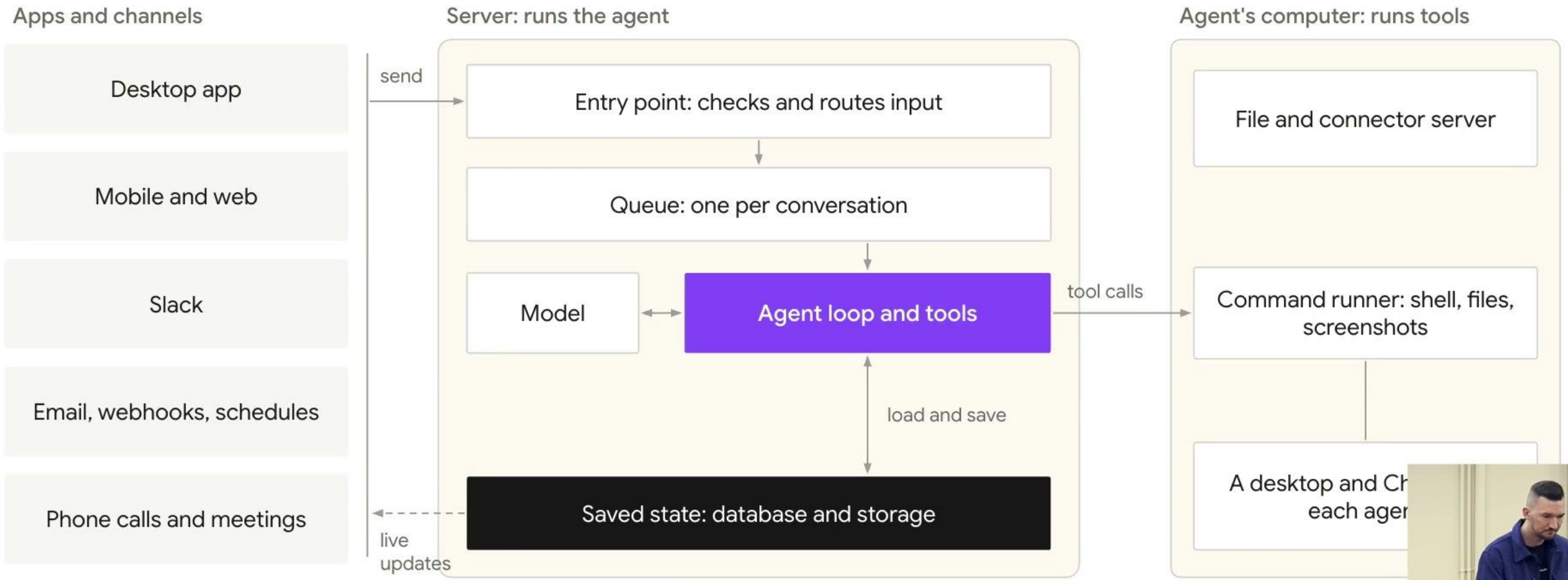
Waking up

On the next tool call

1. The snapshot is restored in place
2. Logins and installed tools are still
3. Conversation state comes from t



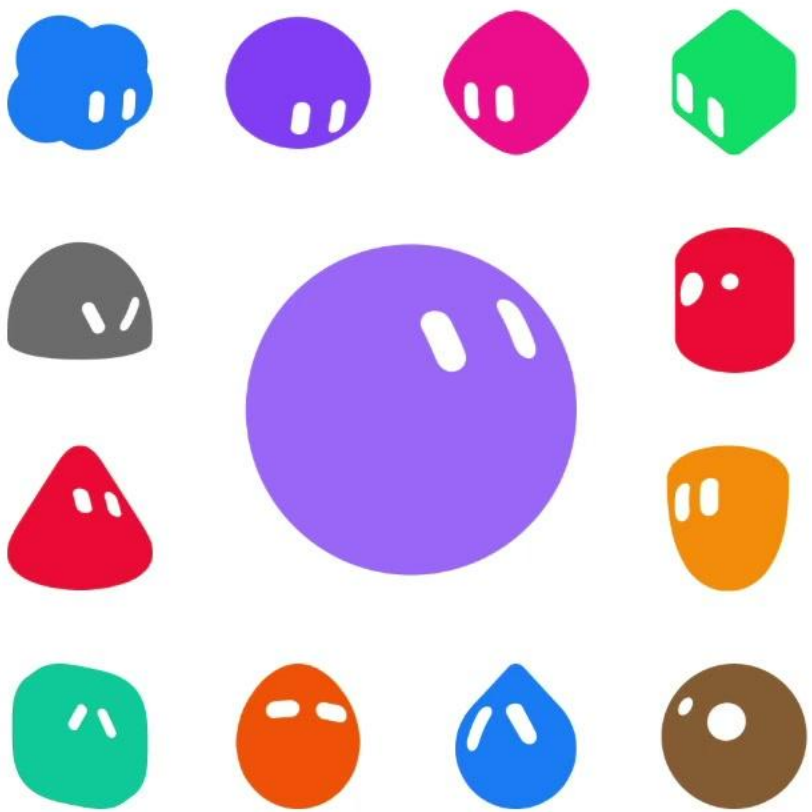
Three parts of the system



The agent runs on the server, uses tools on its own computer, and every app shows the same conversation.



Why the agent runs on the server



- Thin client, thick server
- Close your laptop and the work keeps going
- Easily move between your computer and phone
- Ship code without App Store review cycles
- Offload heavy work to the server
- Agent loop is entirely decoupled from app code
- Easier to scale with hyper growth
- Agents that work for months can’t be on device

Save everything on the server.

The virtual machine can then always be rebuilt.



What happens when you hit send

Notifications only say "something changed." The database always has the real data.

1 Sending

- Your app shows the message right away
- It saves the message locally with a unique ID
- The server drops any ID it has already seen

Unique IDs make retries safe

2 The turn

- The message joins the conversation's queue
- **The agent runs one turn: model, tools, repeat**
- Every step is saved to the database

One turn at a time per conversation

3 Watching

- The server sends a tiny "something changed" signal
- Every open app gets the signal
- Each app fetches the new messages

The signal carries no data, so losing one is harmless

4 Ordering

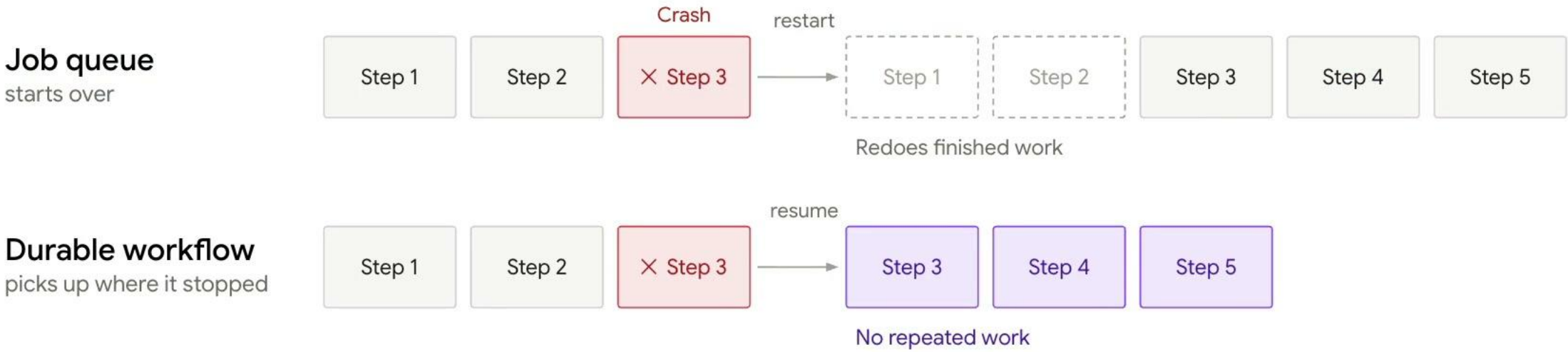
- Every update has a sequence number
- Apps apply updates in order: 41, 42, 43
- If one is missing, reload from the database

Every app ends up same thi



Durable workflows vs. job queues

What happens when a server crashes in the middle of a turn



Resume from any point

Every step is recorded, so a new server picks up from the last one.

Safe retries

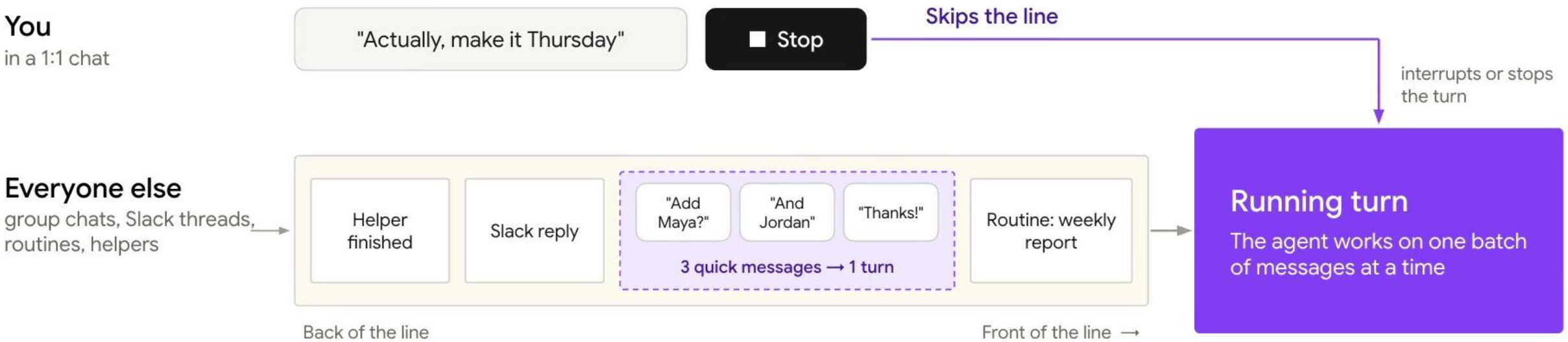
Steps are written so running one twice does no harm. That's called idempotent.

Built in, open source

Timers, cancellation, and heal
come with engines like Tempo



Rules for handling new messages





Part 4
The harness



One tool for talking to you

Tools for doing work

Shell	Files	Browser
Desktop	Memory	Skills
Helper agents	Cloud agents	App connectors
Web search	Image generation	Screen recording

SendToUser

Where your messages show up

Desktop	Web
Mobile	Slack
Email	Voice

Thinking stays private

The model's regular text is never shown to you

Quiet is allowed

If there's nothing new, it doesn't message you

Rich messages

Text, files, multiple choice, and secure password entry

One less step

The last message can end the turn without a model call



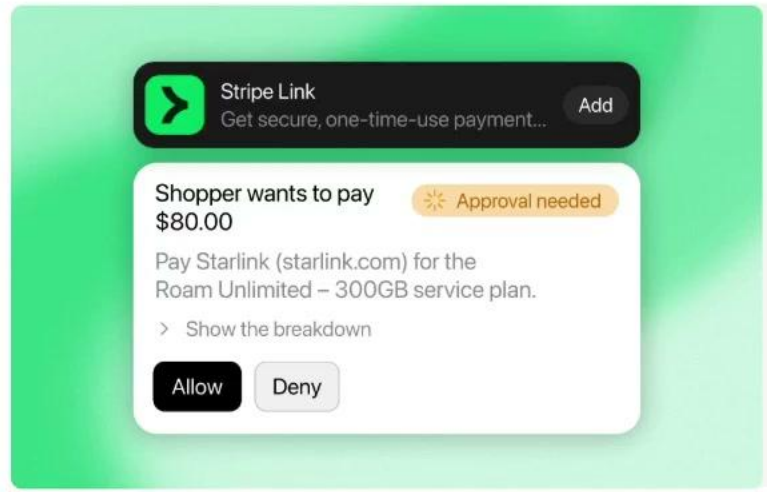
Pairing tools with the right UI

The harness has dozens of tools. When you need to see or approve something, the tool shows a purpose-built control.



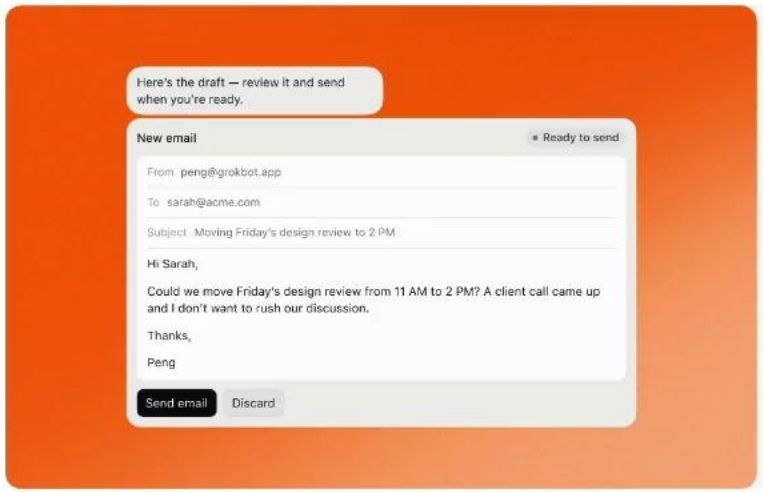
Sign-in forms

Fills your login only on the site that's open. The password never reaches the model.



Payment approvals

Purchases wait until you tap Allow or Deny.



Email drafts

You see the whole email and choose Send or Discard.



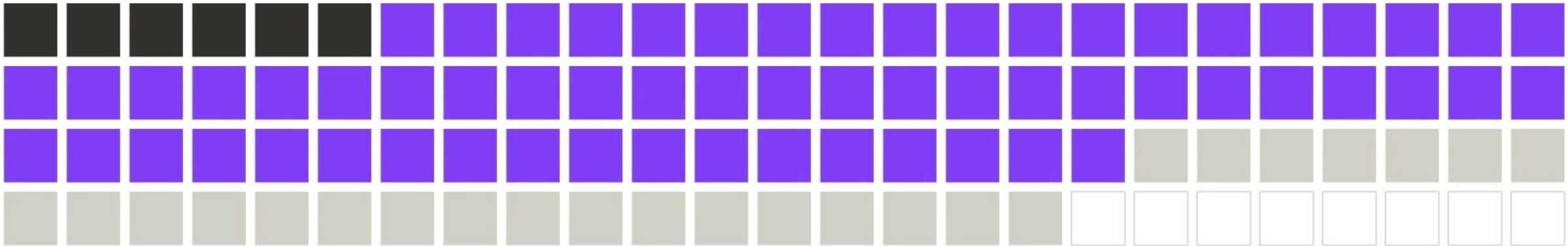
Reactions

Sometimes an emoji is the right reply.

Load tool details only when needed

System prompt | Tool details | Tool names | Conversation | Free space

Every tool loaded up front



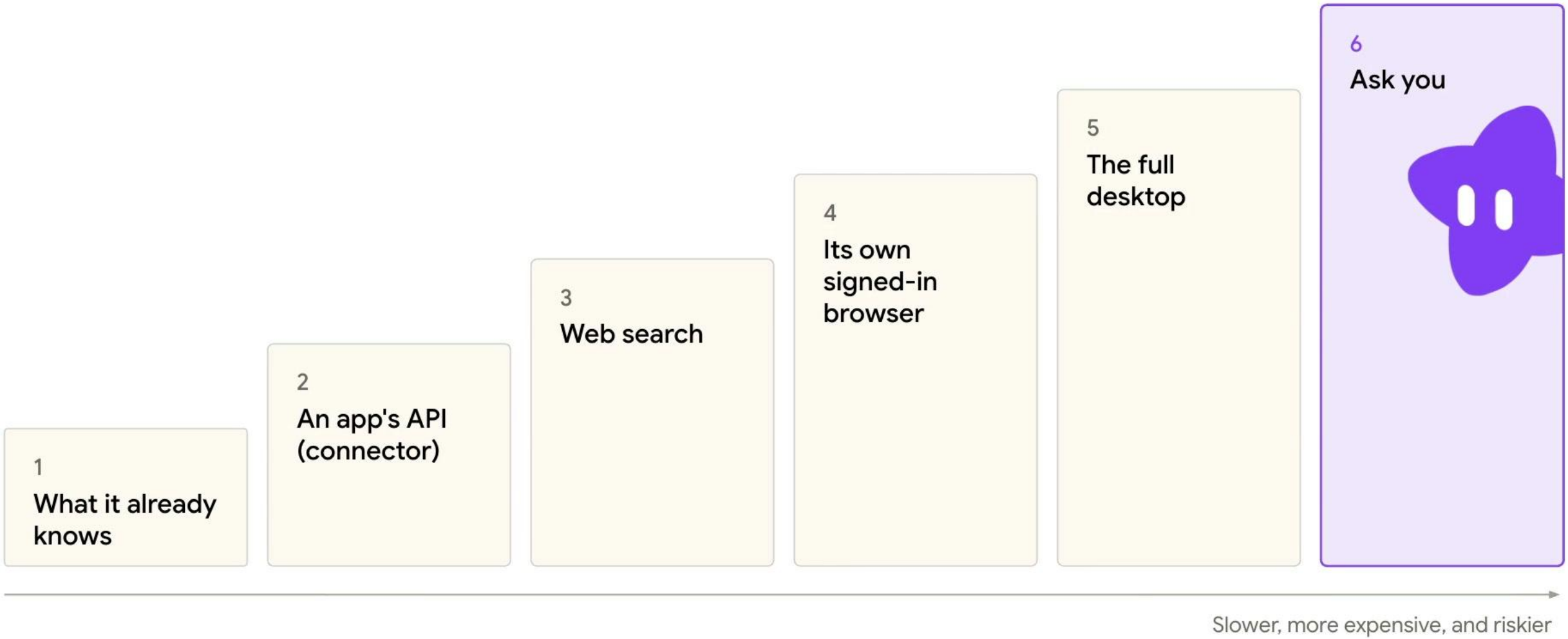
Names first, details on demand

Illustrative



Try the cheapest option first

If a connector is broken, the agent shouldn't work around it with the browser. It should tell you and ask.



Three ways to use a computer

A separate helper agent does the clicking, so screenshots don't fill up the main agent's context.

Read the page structure

It reads the page's list of buttons, links, and fields, and clicks them directly. Faster and more reliable than screenshots.

Click on pixels

Mouse and keyboard on a 1280×800 screen, for desktop apps, system dialogs, and anything else.

Ask a person

At logins, verification codes, captchas, or payments, it stops, explains where it is, and hands you the screen.



One main agent, many helpers

You

Main agent

Answers quick questions itself. Hands bigger tasks to a helper, so it can keep talking to you.

Hands off a complete task description

General helper
Shell, web, app connectors, cloud agents

Computer-use helper
Browser and desktop, one task at a time

Video helper
Watches and reviews video files

Cloud coding agent
Code changes in its own virtual machi

Gets back a short report

A to-do list tracks every request

Check on a helper

Send a helper a message

Stop all help



Tips for designing agent tools

- Don't add or remove tools between turns. If a tool can't be used right now, have it return a clear error
- Keep each tool's description and schema under 4,000 characters
- Use a dedicated tool for saving state instead of shell commands, so reviewers can see what's changing
- Write errors that say what to do next: "The computer is asleep. Use Shell or Read to wake it, then try again."



How we keep it safe

Automatic review

A separate model checks risky actions before they run. If it blocks one, you get an approval request. No answer in 15 seconds means blocked.

Track the source

Text from emails and webhooks is labeled untrusted. Memories are treated as information, not instructions.

Asks first

It won't email or message people unless you asked it to. New routines need your OK.

Keep secrets hidden

Passwords go into environment variables, never to the model. Actions on your own laptop need your approval.



Part 5
Context engineering



Keep the start of the prompt the same

Stays the same until the next summary

Tool definitions
System prompt, including a snapshot of memory and routines
First message: environment details and a list of tool names

← Identical on every call, so the provider can reuse its cached work

Conversation history

New messages are only ever added at the end

← Becomes part of the reusable start of the next call

Newest message

The time, short notes on anything that changed, then your message

← The only part that's new on e

Providers charge less and respond faster when the start of a prompt matches a recent one. So stable content goes first and goes last.



Keep the main context small

Long outputs go to files
The context keeps only the file path

Screenshots go to a helper
The context gets what it found

Tool-heavy work goes to helpers
The context gets a short report

Main agent's context
Stays small, so every call is fast and cheap

Old messages get summarized
The context keeps a short summary

Old memories stay searchable
Only the most important facts are in the prompt

Instructions become skills
The prompt keeps a one-line poi

Heavy things live outside the main conversation. Only a pointer or a summary comes back.



Treat a cache miss like a bug

- Until the next summary, the tools, system prompt, and first message must stay exactly the same
- If a deploy changes a tool description, running conversations get a note instead of the new text
- Everything is built in a fixed order and sorted, so nothing moves around between calls
- Tools are never added or removed for a single turn. Unavailable tools return an error instead
- Each section is hashed separately, so we can see exactly which part changed
- When you open the chat or start typing, the server prepares the prompt early to warm the cache

Summarize while you're away

A big turn ends

The prompt was over 100,000 tokens

About 10 minutes pass

The provider's cache is still warm

Write a summary

Uses the same start of the prompt, so it reuses the cache

Your next message

Starts from the short summary

Skipped if background work is about to wake the agent anyway

The agent's last few messages to you are kept word for word

If a turn gets too long, it can summarize in the middle



How the agent remembers

Profile
Core facts about you. Always in the prompt: up to 100 facts in 4,000 characters.

Log
Dated events. The 30 most relevant recent ones go in the prompt.

Notes
Short-term details that fade quickly.

Everything else
Found with a memory search tool when needed.

- Important facts rank higher, and older facts slowly rank lower
- After each turn, a pass adds new facts and replaces old ones. It can only remove facts it was shown
- Facts about you are shared by all your agents. Each agent writes to its own file
- The model is told that memories are information, not instructions

Part 6
Where this is going

Training the model for the product

Pre-training

Learns language, code, and general knowledge from a huge amount of text.

Supervised fine-tuning

Learns the harness from examples:

- the tools and their formats
- when to send a quick reply
- what an ideal response looks like

Reinforcement learning

Practices real agent tasks and is rewarded for doing them well:

- following long instructions
- using skills and memories
- clicking the right spot (grounding)
- working with other bots

Main agent: fast replies

Inference and harness are tuned so it answers in seconds.

Helpers: long tasks

Tuned to finish hours of work reliably and control the computer quickly.

Product and model improve together

Inner loop: every day

climb the evals

A bot gets something wrong → Turn it into an eval → Fix the harness, prompt, or inference → Check the evals → Ship the fix

Evals guide training

Outer loop: every few months

new models and products

Train the next model on the harness and evals → Swap it into the product → People use it in new ways, like group chats full of bots → New behaviors to evaluate and train for

Every bot that doesn't work well becomes an eval. Evals improve the product today and the next model tomorrow.



Delete the product.

Assume models will get much better. Build as little as you can around them, and remove what they no longer need.



What I expect in 6 to 12 months

Longer projects
Agents that work on one project for months, not just one task for a few minutes.

Better memory
They remember how you and your team work, and keep improving at it.

More powerful tools
Anything you can do on a computer, done faster and more reliably.

Simpler interfaces
Texting today, maybe voice only next. Whatever is easiest for each person.

Skills they weren't trained for
Bots can learn on the job, combining better memory, skill and instruction following, and raw intelligence..

Knowledge written dov
People and companies write down what they k agents can use it.



Open problems you could work on

Testing long-running agents
How do you grade an agent that works for weeks? Current methods struggle.

Correct memory
Replacing outdated facts, forgetting when asked, and never making things up.

Prompt injection
An agent that reads email and the web will see text written to trick it. Defenses are still early.

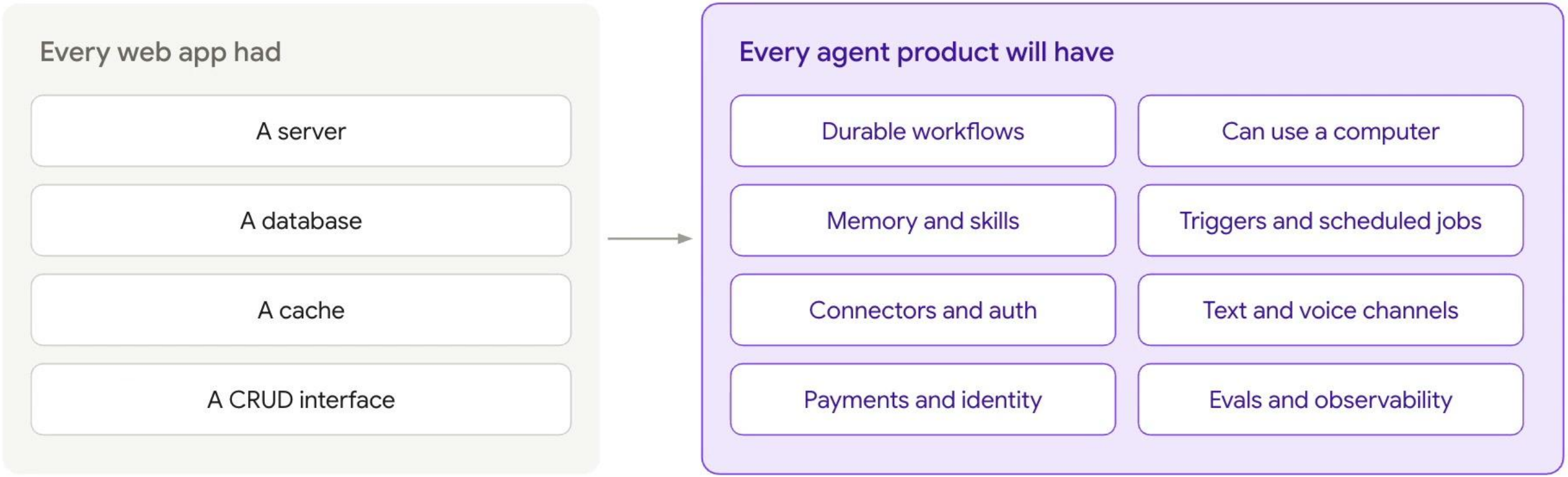
When to speak
Too many messages is noise. Too few and you can't tell what it's doing.

Coordination
Many agents sharing one computer and one memory run into classic distributed-systems problems.

Cost
Always-on only works if idle agents are nearly free and active ones are cheap.



The new building blocks



Many products will look alike but build these pieces very differently. Expect a long tail of specialized, best-in-class versions of each.

What to take away

As code gets easier to write, knowing how the pieces fit together matters more.

Architecture compounds

When code is cheap to write, the right architecture is what keeps you moving faster than everyone else.

Build your own

Start from your week-one agent and add a server, a computer, and memory. Open-source always-on agents are great to learn from.

Learn the fundamentals

Knowing how LLMs work helps you judge what's possible, the same way knowing databases helps when you build a website.

Thank you

Questions? · Lee Robinson · @leerob

