Modern AI Systems: Agents and System Optimizations

Agent From a User's Perspective I: How Claude Code Works

Juncheng Yang

Harvard John A. Paulson School of Engineering and Applied Sciences

HARVARD UNIVERSITY
MaDSys
Measurement and Design of Systems Lab

Schedule update

LLM background moves later

- Treat the LLM as a magic reasoning box for now
- You don't need the internals to learn agent

More agent, less GPU

- Most of you are more interested in agents
- Two extra lectures on agent design
- Fewer GPU lectures — less overlap with other courses

What changed on the schedule

- No mandatory paper presentation
- Score goes to participation and project
- Assignment 1 sharing is now a bonus
- A few bonus paper discussions — TBA

A few more logistics

New TF: Advaith Ravishankar

Participation score: ask / answer question in class
Up to 1 point each class, find TF after class
Extra reading: optional

Hands up = score

OPTIONAL — extra reading, not required

ADVANCED — skim, or read more on your own

Plan for today

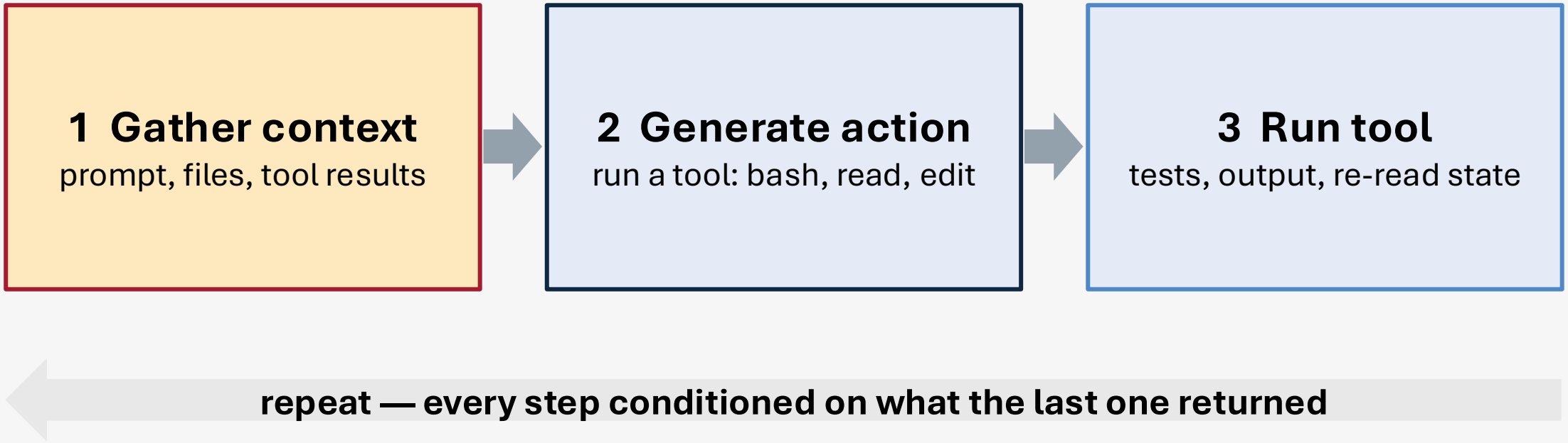
- How Claude Code works
- Claude Code: from stateless LLM to an agent
  - Memory
  - Tools, skills, hooks, plugins

Hands up = score

What happens when you type a prompt into Claude Code?

The Agentic Loop

One user prompt = many model calls and many tool executions, not a single request.



Context Before You Type

Before your first keystroke, the context size is already 10+k tokens

SYSTEM Claude Code instructions

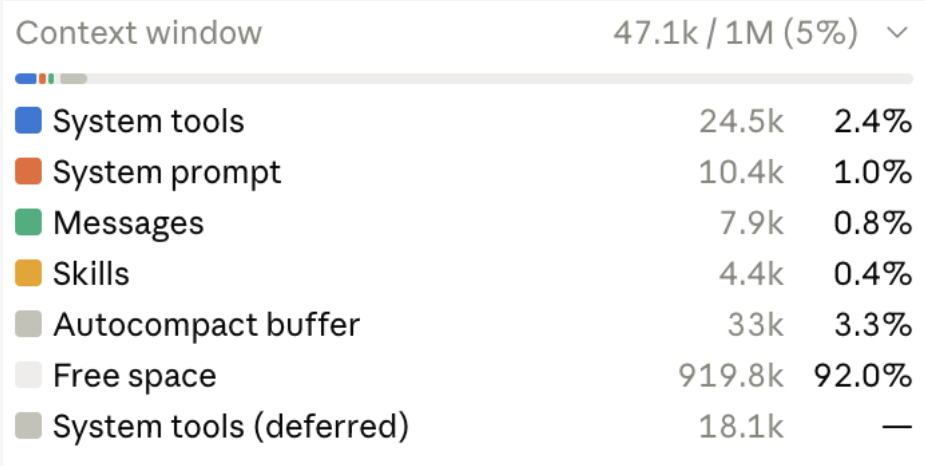
PROJECT environment

MEMORY persistent project / user notes / CLAUDE.md

TOOLS Read, Edit, Bash, Glob, Grep, Web, Agent

SKILLS / MCP names and descriptions

USER “Fix the failing auth test”

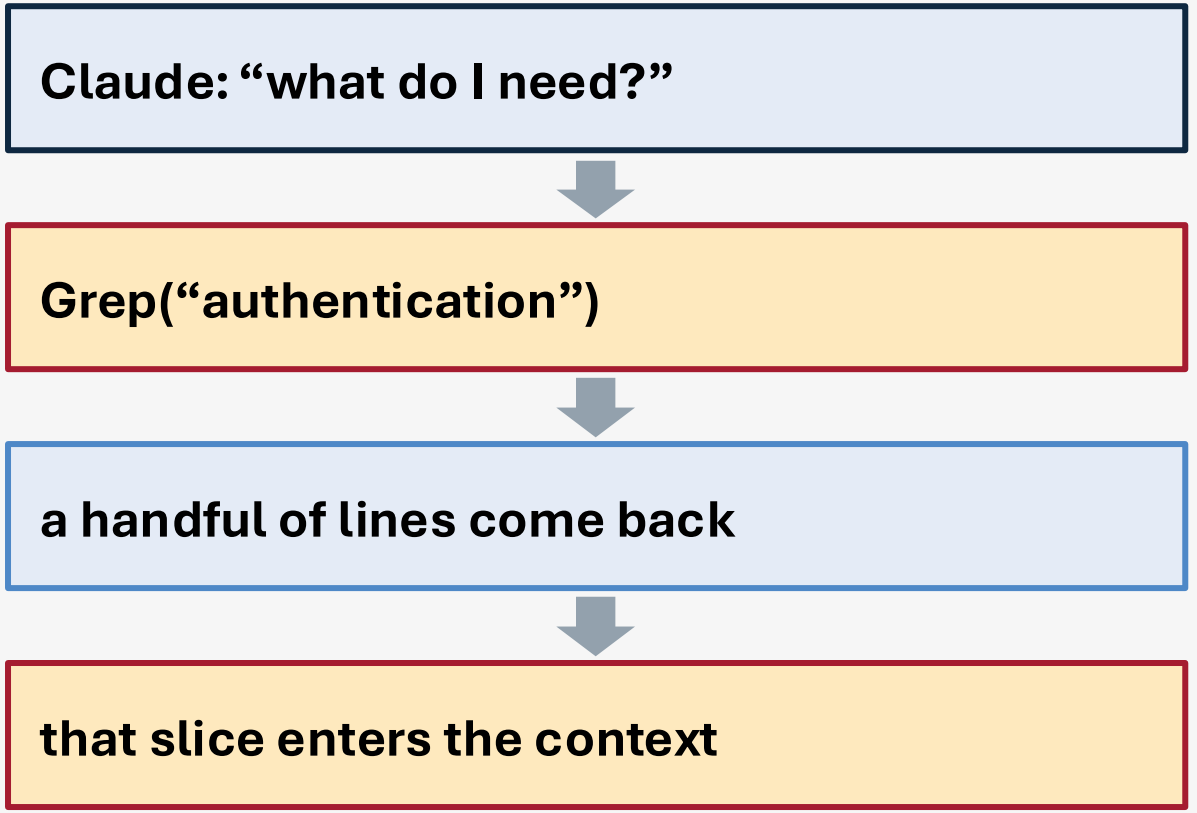


Skill and MCP entries start as names only — the full text loads when Claude actually reaches for them.

It Doesn't Send Your Repository

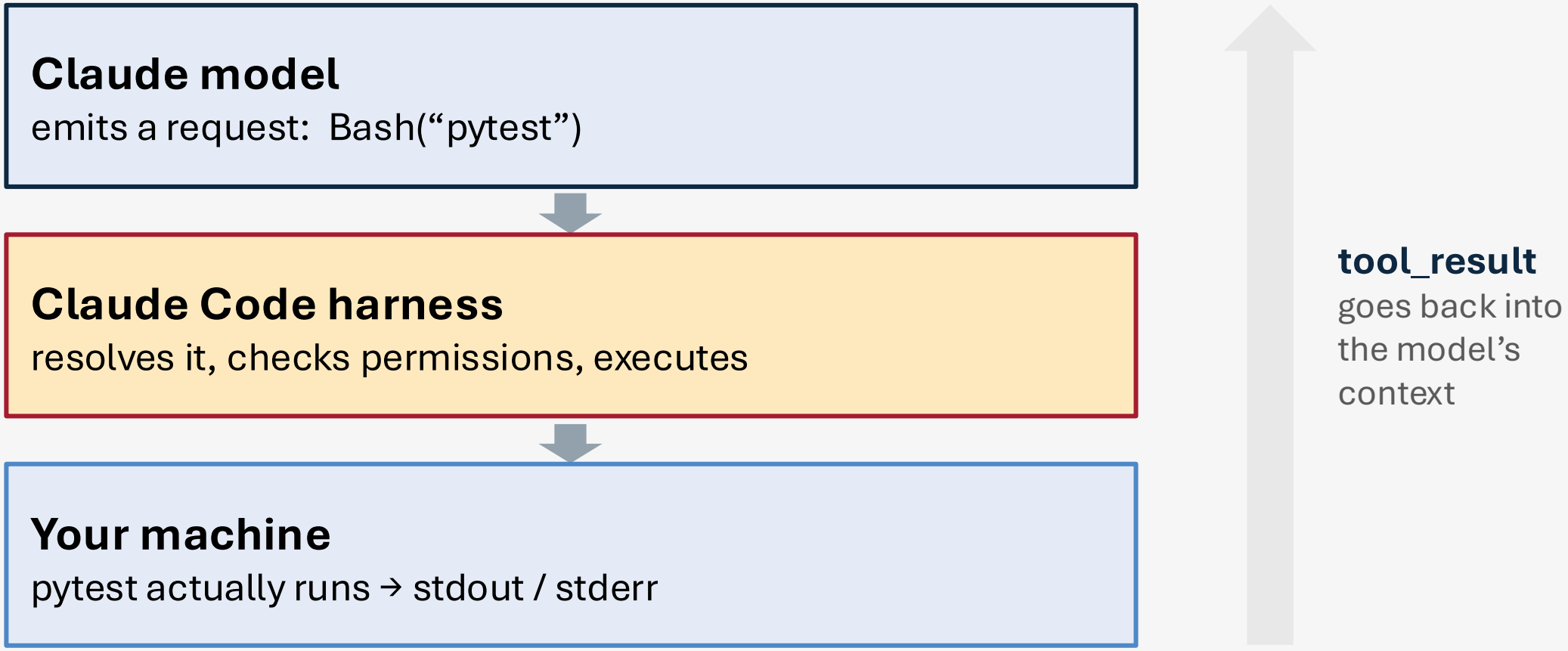
The repo stays on disk. Claude pulls slices of it on demand.

Your repository
thousands of files, on disk



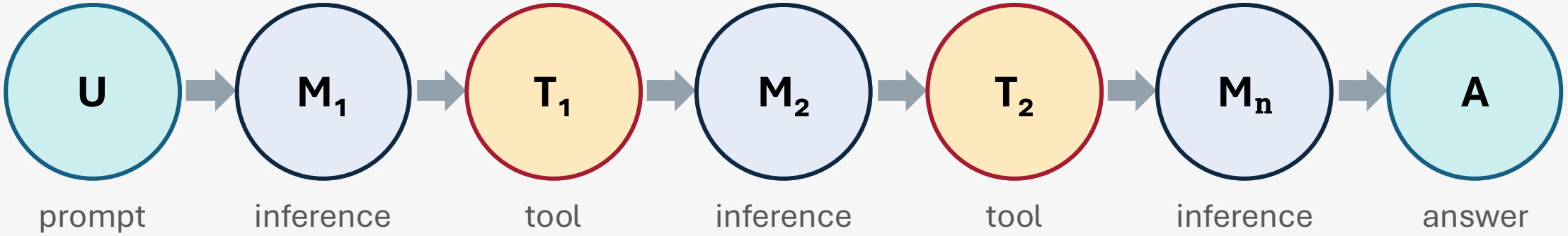
Model Proposes and then Harness Executes

The model proposes for an action and the harness executes it



One User Prompt Triggers Many LLM Inferences

One user turn expands into an alternating chain of inference and tool execution.



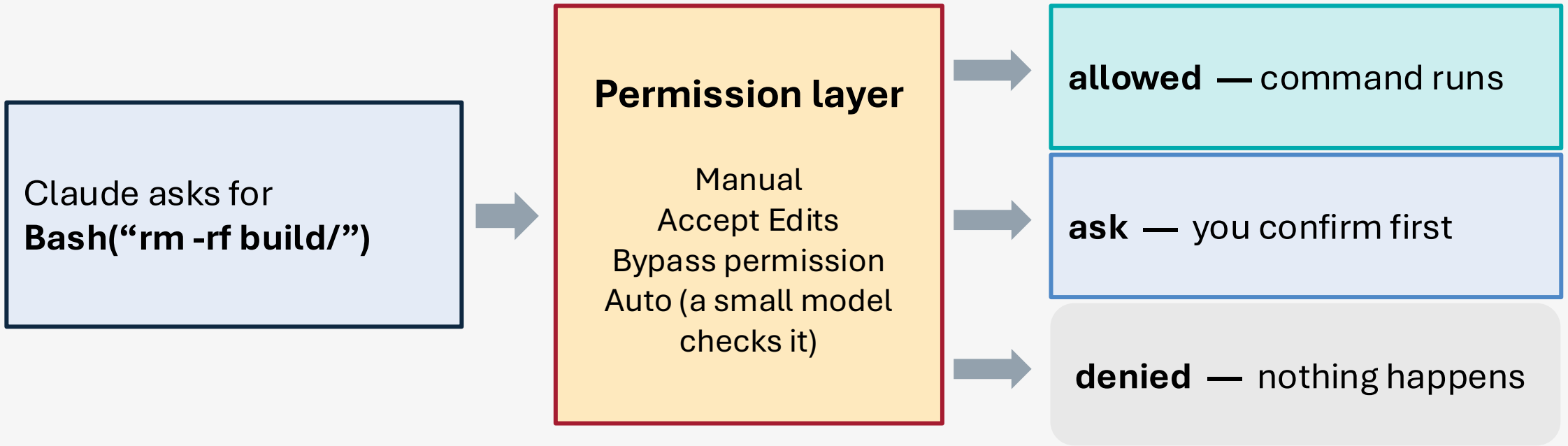
What that looked like in practice

pytest	read session.py	grep create_session	edit session.py	pytest	**all green**

Every request is conditioned on the previous inference and the previous tool result.

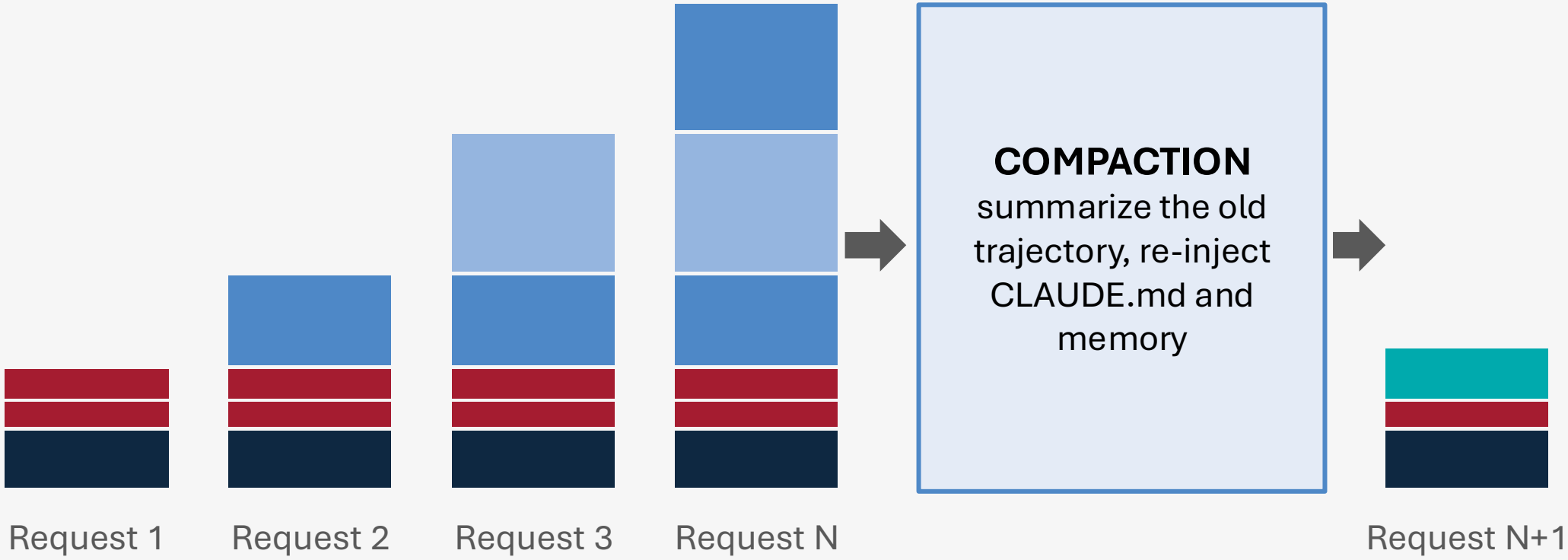
Permissions Sit Between Intent and Execution

A tool request is a proposal that needs your or another model's approval.



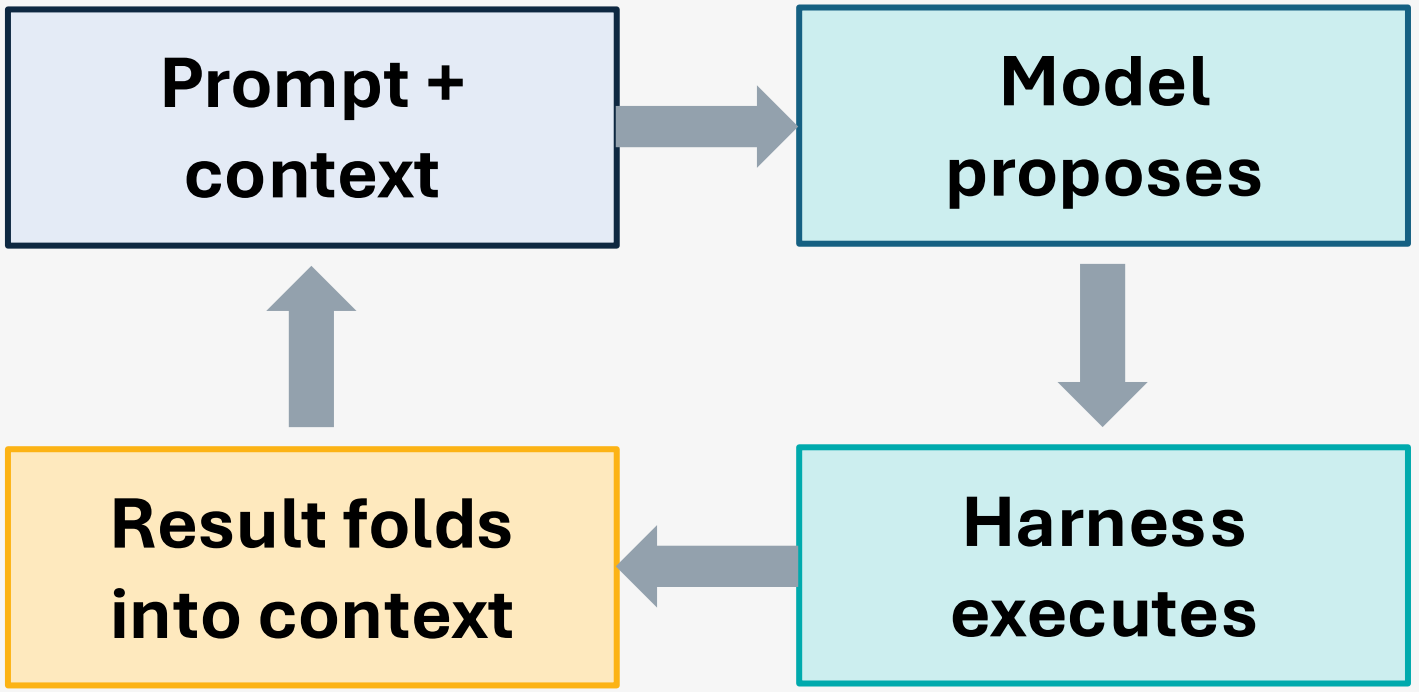
Context Grows — Then Gets Compacted

Typical context size: 1M tokens (Sonnet / Opus / Fable, GPT 5.6); 256K was common earlier this year
Context is not free: prefix-cache reads account for most of the cost



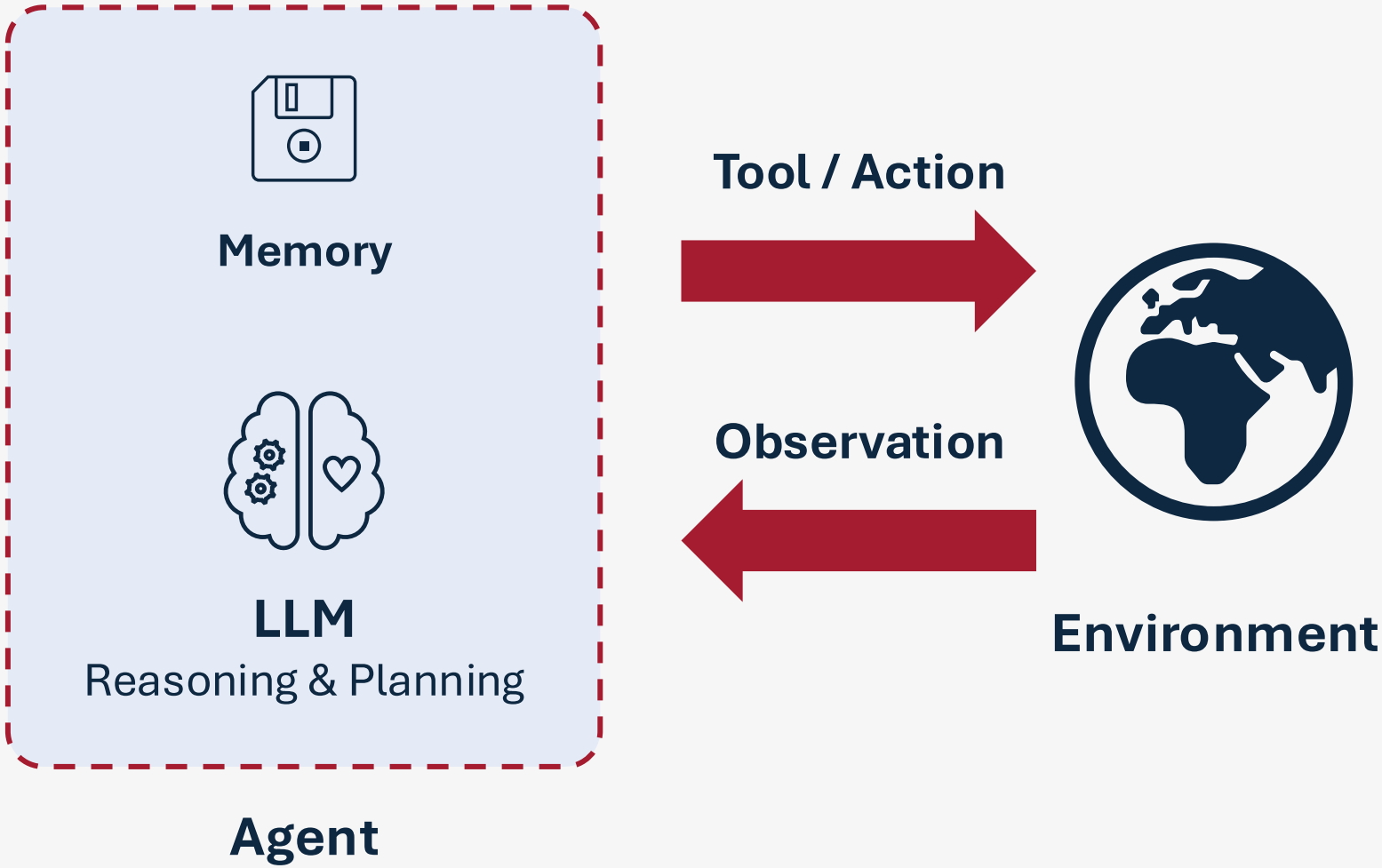
Each step adds tokens: model replies, file contents, command output.
Context management is an important component of harness design (discuss next week).

Summary: The Agentic Loop



The loop repeats until the task is done

Tools and Memory Enable Agentic Capability



Agent Memory

Context Is One Type of Memory

- Agents are stateless between runs — every session starts from zero
- In practice, “memory” is just files loaded at session startup

Every memory creation is a decision about what gets re-read forever.

Context: Working Memory

What the model can see right now — it dies with the session.

Persistent Memory

What survives the session and is reloaded next time.

Why Agents Need Memory

Repeated instructions

The same setup explained every session

Lost decisions / assumptions

The “why” disappears when the window compacts

Repeated discovery

The agent re-reads the same files to relearn the layout

Inconsistent behavior

Different sessions make different choices

Memory is how instruction becomes convention.

Two Systems Carry Knowledge Forward

Both load at the start of every session into context.

	CLAUDE.md files	Auto memory

Two Systems Carry Knowledge Forward

Both load at the start of every session into context.

	CLAUDE.md files	Auto memory
Who writes it	You	Claude
What it contains	Instructions and rules	Learnings and patterns
Scope	Project, user, or org	Per repository, shared across worktrees
Loaded into	Every session	Every session (first 200 lines / 25KB)
Use for	Standards, workflows, architecture	Your preferences, corrections, project context

Where CLAUDE.md Lives

Files load broadest-scope first; the closest file is read last.

1	**Managed policy**	…/ClaudeCode/CLAUDE.md · org-wide, cannot be excluded
2	**User instructions**	~/.claude/CLAUDE.md · your preferences, every project
3	**Project instructions**	./CLAUDE.md or ./.claude/CLAUDE.md · team-shared through source control
4	**Local instructions**	./CLAUDE.local.md · personal, gitignored

What Belongs in CLAUDE.md

Belongs

- **Commands**: how to run the tests, the build, the linter
- **Hard constraints**: never commit to main, never edit generated files
- **Facts** the agent cannot cheaply infer: layout, ownership, gotchas

Doesn't belong

- Anything the model already knows about the language or framework
- Style preferences a linter or formatter already enforces
- Long prose, changelogs, and anything secret

Size	Structure	Specificity	Consistency
Under 200 lines per file	Markdown headers and bullets	Concrete rules, not vague ones	No contradictory or stale rules

Context is *rented*, memory is *owned* — choose deliberately what you own.

Getting Too Long? Split It Into .claude/rules/

One topic per file — and each rule loads only when it is relevant.

One topic per file

Each rule file covers a single concern, so it stays short and specific

Add a paths: frontmatter

Loads only when Claude touches matching files (no paths: every session)

Example

```
.claude/rules/
├── testing.md
├── api-style.md
└── db-migrations.md
```

The lesson

- Scoped rules keep the context window small
- Only rules matching the work get loaded
- CLAUDE.md stays short; detail lives in rules

Auto Memory: Claude's Own Notes

`user`
Your role, expertise and working preferences

`feedback`
Corrections you give, approaches you confirm

`project`
Ongoing work, decisions not in the code

`reference`
Information lives outside the project

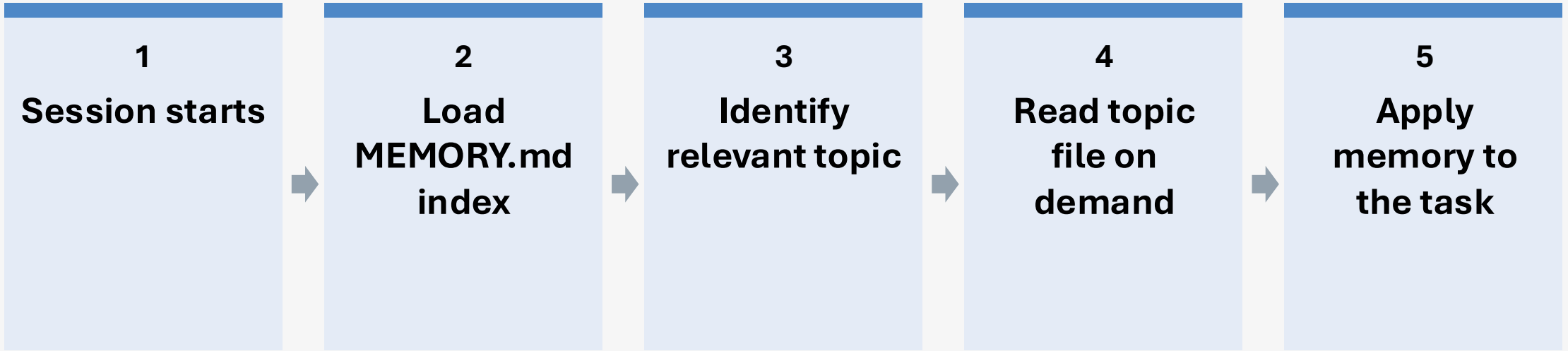
```
~/.claude/projects/<project>/memory/
├── MEMORY.md
├── user_role.md
├── feedback_testing.md
└── other-topic-files.md
```

How it behaves

- A type field tags every file
- MEMORY.md is the index — first 200 lines (25KB) load each session
- Machine-local; worktrees share the directory
- On by default; disable with /memory

Retrieval Uses an Index

The index is always loaded; topic files are read only when they are relevant.



Memory.md
The index keeps the context window small — only the topic file the task needs gets read.

Inspecting and Controlling Memory

Four commands for seeing — and steering — what Claude remembers.

/memory — Browse, edit, delete, or disable memory

/context — See which instruction files loaded

/init — Generate a starter CLAUDE.md

/doctor — Identify content that may not belong in CLAUDE.md

The Cost of Remembering

Limited context size + you pay for every token you send to Anthropic

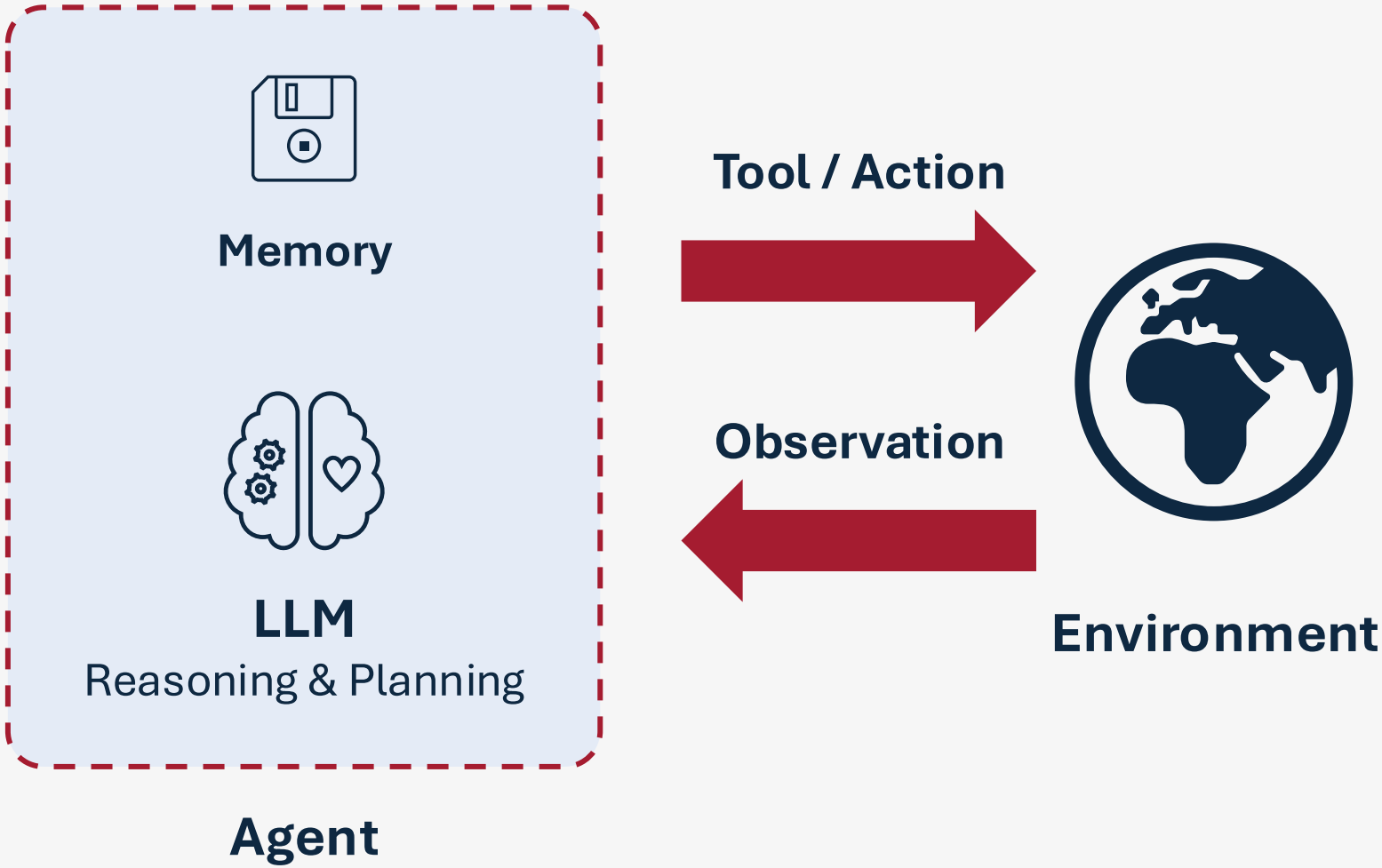
Every permanent token is rent — paid at the start of every session

Memory competes with the task for the same window

Stale memory is worse than none: it actively misleads

Treat memory like code — review it, prune it, keep it in version control

Tools and Memory Enable Agentic Capability



Become an Actionable Agent: The Core Tools

Read
open a file, inspect state — the most used tool

Search
grep and glob — find it without reading everything

Fetch
pull a page or doc — brings untrusted text inside

Edit / Write
change files — the first tool with consequences

Shell
build, test, git — the whole machine unless sandboxed

Delegate
spawn a subagent — a tool that returns a summary

Model will decide which tools to use, not much you need to do
You will build these tools yourself in Assignment 2

Extending an Agent: Skills, MCP, Hooks and Plugins

Extending an Agent

Four mechanisms, four different problems — and they are not interchangeable.

Skill

How the agent should perform a task

MCP

Which external data and actions it can reach

Hook

What happens automatically at a lifecycle event

Plugin

How the extensions are packaged and distributed

Four layers: a skill decides how, MCP reaches outside, a hook enforces, a plugin distributes

Extension Points in the Agent Loop

Each mechanism attaches at a different point in the loop.

User request

Model reasoning

Tool selection

Tool execution

Skill provides the knowledge and procedure

MCP provides the external tools

Hook can inspect, block or react at any stage of the loop

Plugin packages skills, MCP servers and hooks into one installable unit

Release Workflow Example

Prepare a release, validate it, publish it to GitHub, notify the team.

Follow the release procedure	**Skill**
Create the GitHub release	**MCP**
Prevent publishing when tests fail	**Hook**
Share the complete setup	**Plugin**

Skills as Reusable Procedures

Knowledge and procedure the agent loads when the task calls for it.

What a skill carries

Domain knowledge	Reference material	Repeatable workflows	Model- and user-invoked	Reasoning at run time

Think about it as context engineering

Example: /release 2.4.0

Skill Structure

A folder, a description that controls discovery, and a body that holds the procedure.

```
.claude/skills/release/
├── SKILL.md
├── references/
└── scripts/

name: release
description: Prepare a release
```

How it works

- The description controls discovery
- The body holds the procedure
- Supporting files add references and scripts
- Users can also run it directly: /release

1 Read the release notes · 2 Confirm the version · 3 Run the tests · 4 Prepare the commit · 5 Ask before publishing

Skill Invocation and Context Cost

Two invocation modes — and two different costs in context.

Model-invoked

Claude loads the skill when its description matches the task

User-invoked

The user explicitly runs /<skill-name>

What it costs

- Descriptions loaded during session start; the full body loads only when the skill runs
- Side-effecting workflows should usually require explicit invocation
- Clear, distinct descriptions reduce incorrect skill selection

MCP as an External Capability Layer

Model Context Protocol connects agent to external services through a standard Protocol.

- Read and update GitHub issues
- Query a database
- Retrieve monitoring data
- Access Figma designs
- Post a message to Slack
- Control a browser

MCP gives Claude capabilities that exist neither in the model nor in local files

How an MCP Tool Call Works

Six steps from intent to structured result.

1. Claude identifies a needed action
2. The MCP client uses a tool
3. The MCP server handles the connection
4. The external service performs the operation
5. Structured results return to Claude
6. Claude evaluates the result

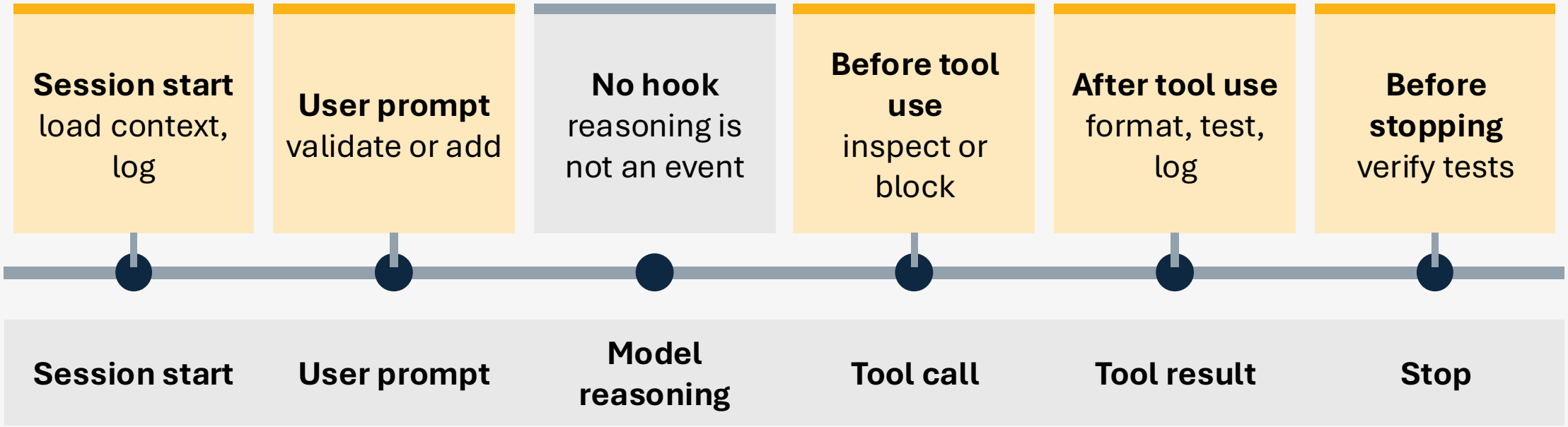
Almost identical to a local tool

- Same name, description and typed schema
- Same call-and-return contract
- It just runs in another process

The protocol carries the call; the agent still decides whether to make it

Hooks as Lifecycle Automation

A session runs left to right; a hook can fire at each marked event.



Hooks fire on events, can run commands, HTTP requests, MCP tools, prompts or subagents

Plugins as the Packaging Layer

One installable unit that carries skills, hooks, agents and MCP configuration.

```
plugin/
├── skills/
├── hooks/
├── agents/
├── .mcp.json
├── .lsp.json
└── .claude-plugin/plugin.json
```

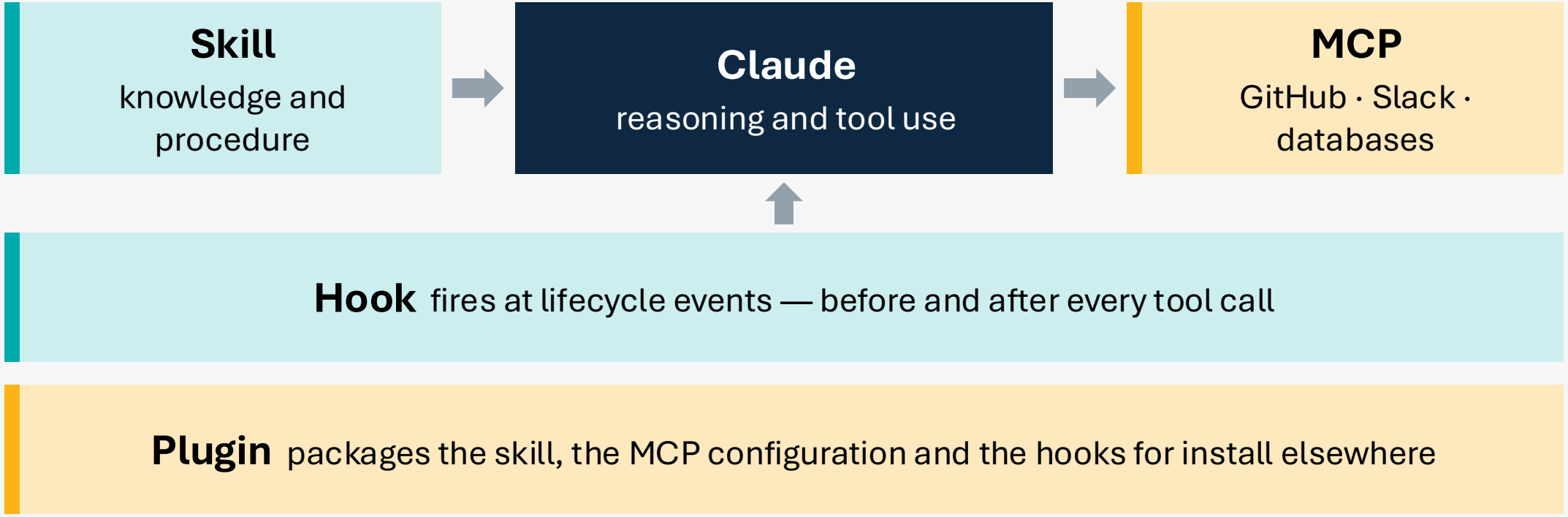
What packaging gives you

- **Installation** — one command
- **Versioning** — pinned releases
- **Namespacing** — no name clashes
- **Team reuse** — one setup for everyone
- **Distribution** — share via marketplace

A plugin is how a working setup becomes something other people can install

How the Four Mechanisms Fit Together

Judgment, capability, enforcement — and the package that ships all three.



Inside .claude/

```
.claude/
  CLAUDE.md
  settings.json
  settings.local.json
  commands/
  skills/
  agents/
  hooks/
  plugins/
```

Memory CLAUDE.md, rules/

Configuration permissions, hooks, env

Workflows skills/, commands/

Delegation & guardrails agents/, hooks/

The same layout exists at ~/.claude/ for user scope. Project settings win over user settings.

Design Exercise

Hands up = score

Classify each requirement — and say why the others do not fit.

Apply the company API conventions	**Skill**
Read an issue from Jira	**MCP**
Run formatting after every edit	**Hook**
Guide a multi-step deployment	**Skill**
Send a completion notification	**Hook plus MCP**
Block writes to production	**Hook or policy**
Share everything across repositories	**Plugin**

Skill knowledge and procedure · **MCP** external access · **Hook** event automation · **Plugin** packaging and distribution

Next class

- Agent from a user’s perspective II

Optional slides

MCP: What and Why

What if I need to access external services? Gmail, GitHub, ...

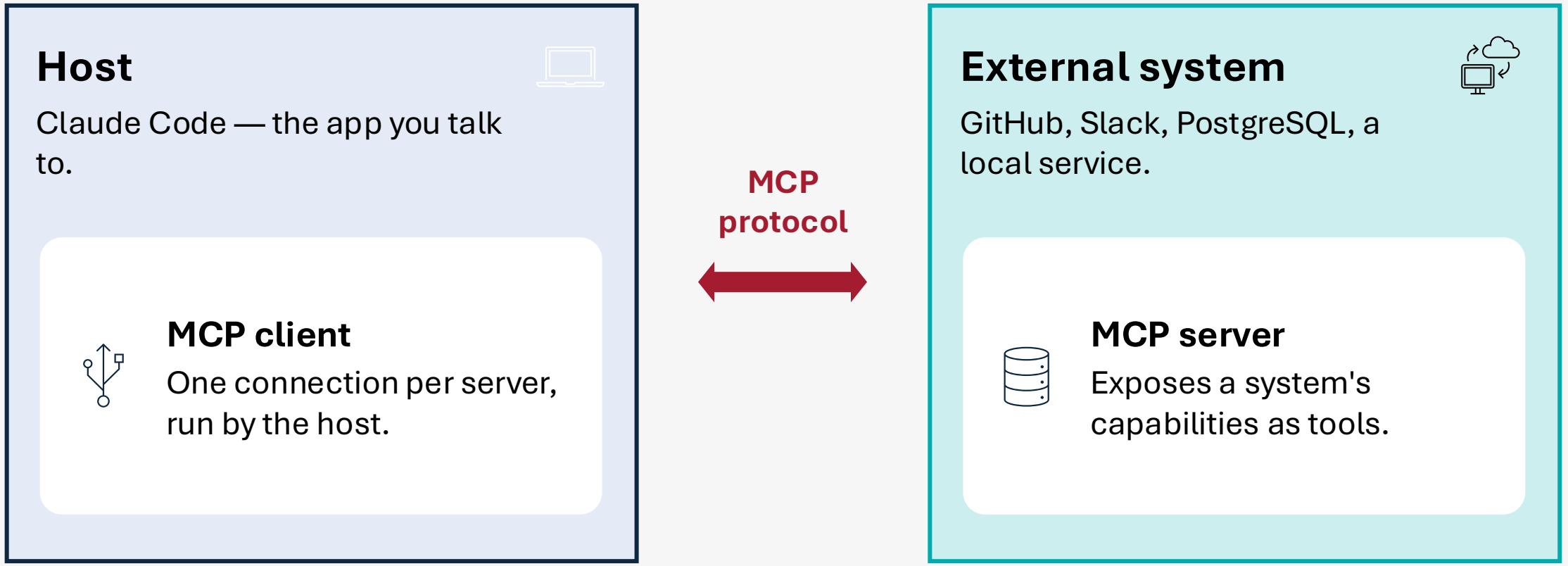
Why MCP exists: access to external tools and services

- Before MCP exists: NxM implementations
- It avoids building a unique integration for every AI client

An open protocol (stateless) that lets an AI application discover and use capabilities exposed by external servers.

https://modelcontextprotocol.io

MCP Architecture



MCP: Three Core Primitives

Tools

Actions Claude can call

`create_issue(…)`

Resources

Readable data — an issue, a schema

`github://issues/123`

Prompts

Reusable prompt templates a server exposes

`triage_issue(…)`

A prompt returns instructions

- Review issue #123.
- Determine whether it is reproducible.
- Assess its severity as high.
- Recommend an owner and next action.

Prompts are user-controlled

Apps surface them as commands or menu choices.
In Claude Code, as namespaced slash commands:

`/mcp__github__triage_issue 123 high`

How They Work Together

Suppose you ask:
"Triage issue #123 and assign it if it is a critical authentication bug."

The three primitives might cooperate like this:

Prompt
`triage_issue`
Supplies the organization's triage procedure

Resource
`github://issues/123`
Supplies the issue's content

Tool
`get_related_errors`
Retrieves current diagnostic information

MCP Security: The Essential Part

A connected server can see what you send it and act with your credentials.

Least privilege — scope the token to the repo, table or channel you actually need.

Read vs. write — separate reads from writes; keep mutations behind approval.

Distrust tool output — never trust returned content as instructions.

Vet before installing — third-party servers can carry prompt injection; review them.

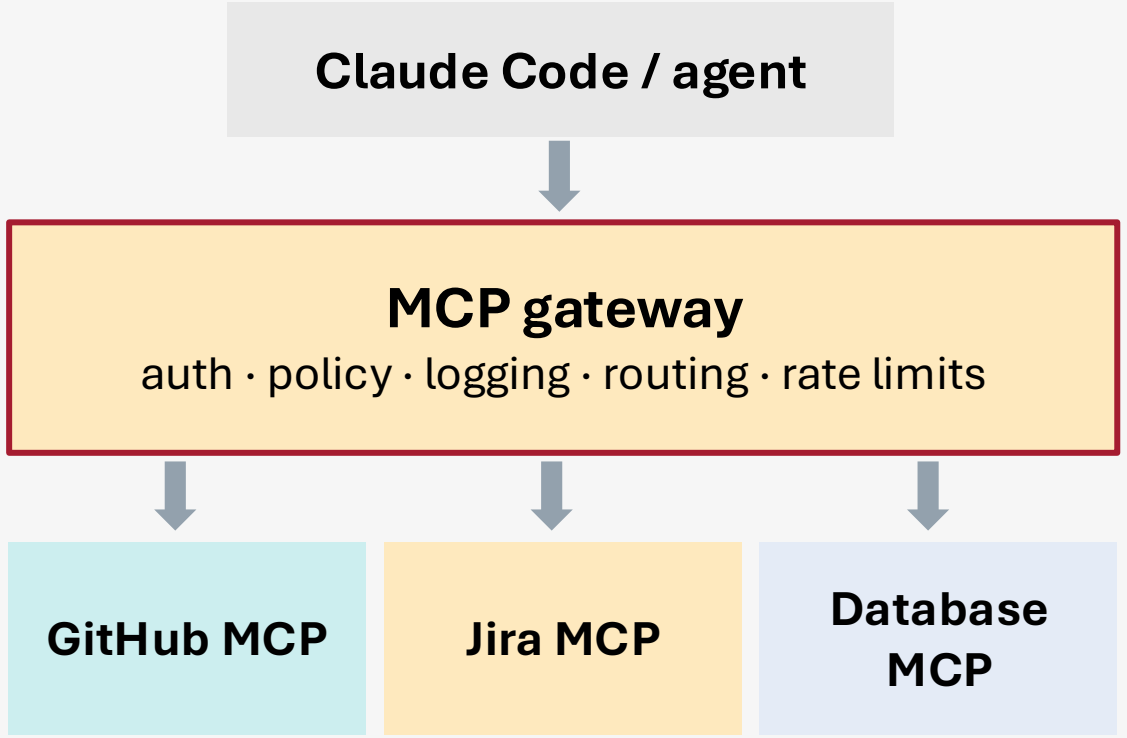
Ask before connecting: what can this server read, and what can it change?

What an MCP Gateway Is

A control layer between an AI client and one or more MCP servers.

Like an API gateway — but it speaks MCP: tools/list, tools/call, resources, prompts, schemas, protocol versions.

An architectural pattern, to client it just looks like another MCP server.



OPTIONAL — extra reading, not required

What a Gateway Does

1 · Aggregate
One connection, one namespaced catalog: github.*, jira.*, postgres.*

2 · Route
Forwards each call to the owning server; Mcp-Method headers route without reading the body.

3 · Enforce policy
Who may connect, which tools are visible, callable, or need human approval.

4 · Broker identity
Client authenticates once; upstream credentials stay behind the gateway.

5 · Observe
Central audit trail: identity, tool, decision, latency, outcome, cost.

6 · Scale
Load balancing, retries, circuit breakers, catalog caching, quotas.

Worth it when…
many users, agents, servers, credentials or compliance rules must be governed consistently.

But it is a trust boundary
broad credentials, lost end-user identity, name collisions, secrets in logs, one point of failure.

OPTIONAL — extra reading, not required

What is new?

Spec 2026-07-28 — the largest revision of MCP since launch

Stateless core

No sessions to hold. Any request can land on any instance, behind a plain load balancer.

Routing

Mcp-Method headers let gateways route without reading the body; list results are cacheable with a TTL.

Extensions

Tasks for long-running work and MCP Apps for server-rendered UIs now ship as extensions.