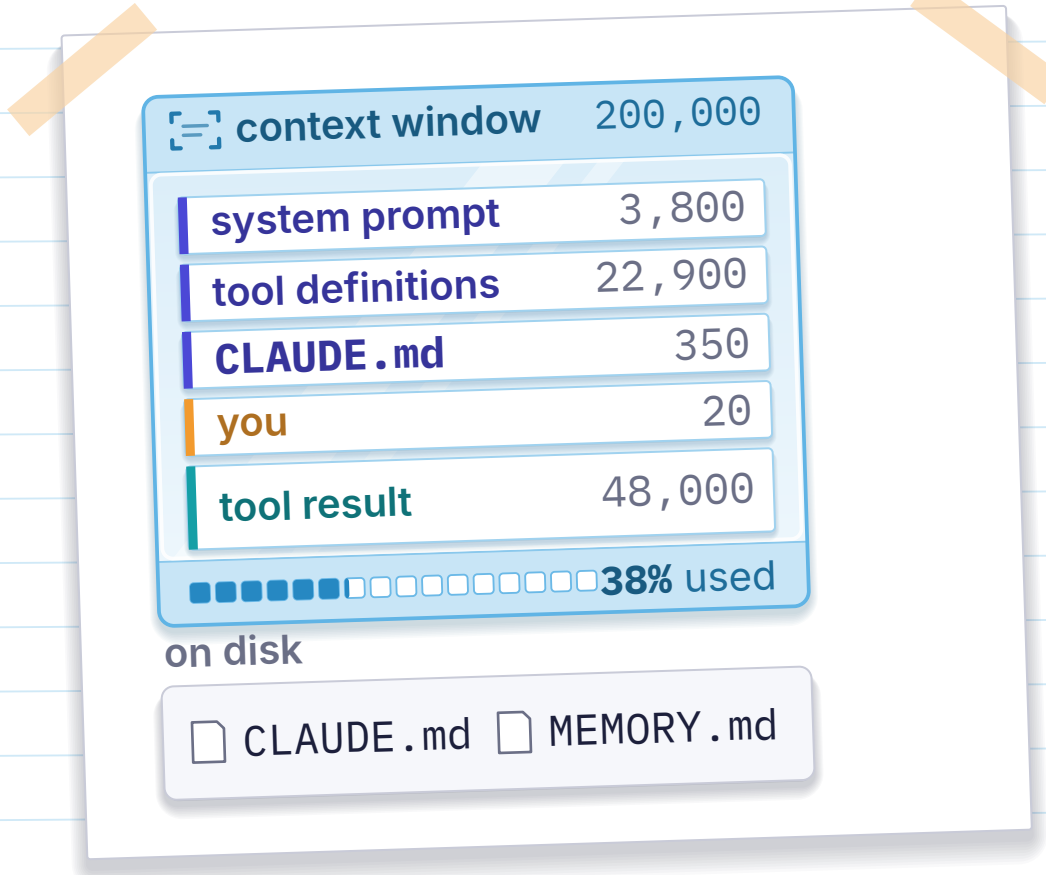
Prof. Aviv Zohar

Semester A, 2026–7

Context and Memory

What the harness puts in the window, and what it keeps between sessions



The harness decides what goes into the window

It decides what goes in.



harness

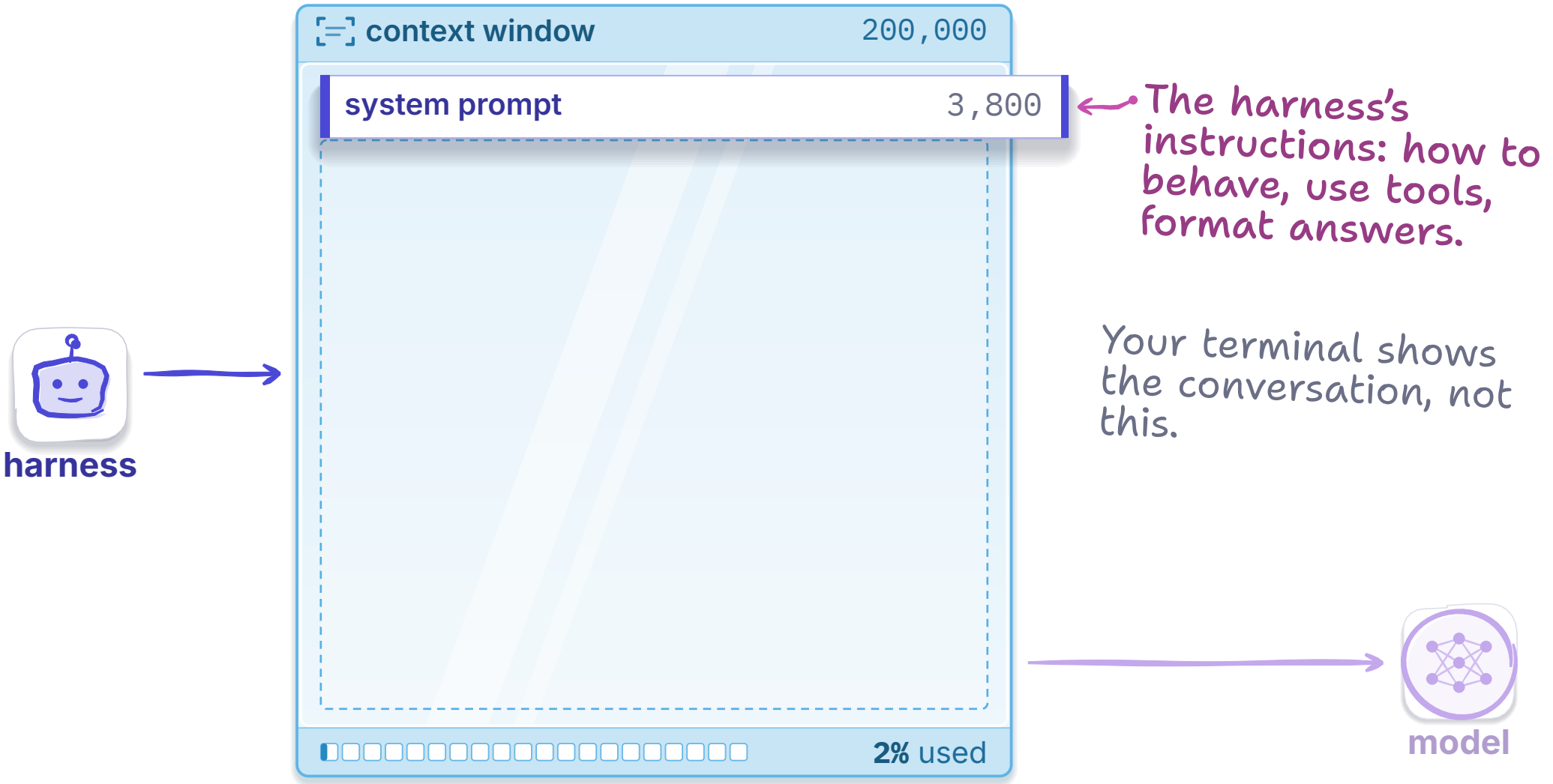


The model reads this window, and nothing else.



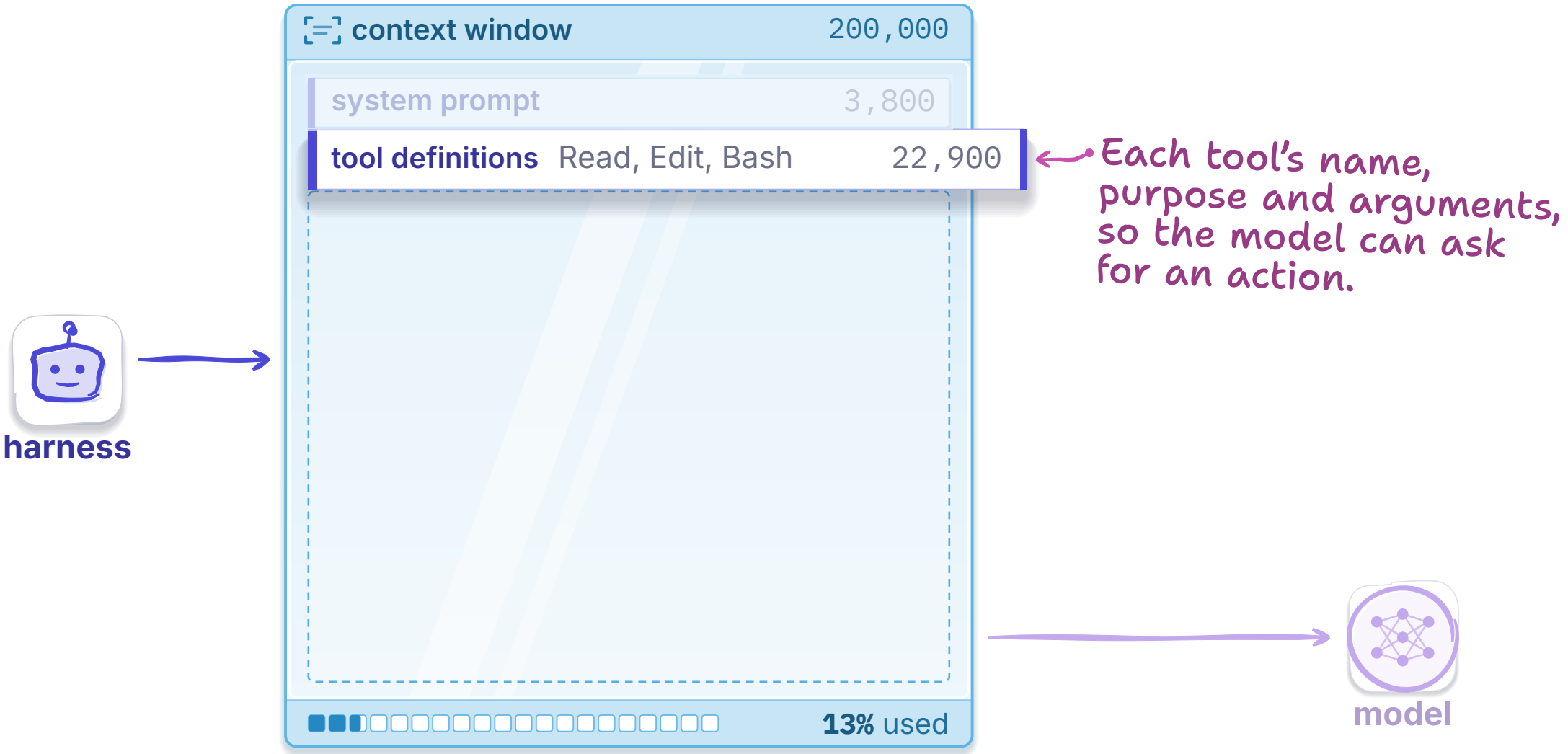
model

The harness decides what goes into the window



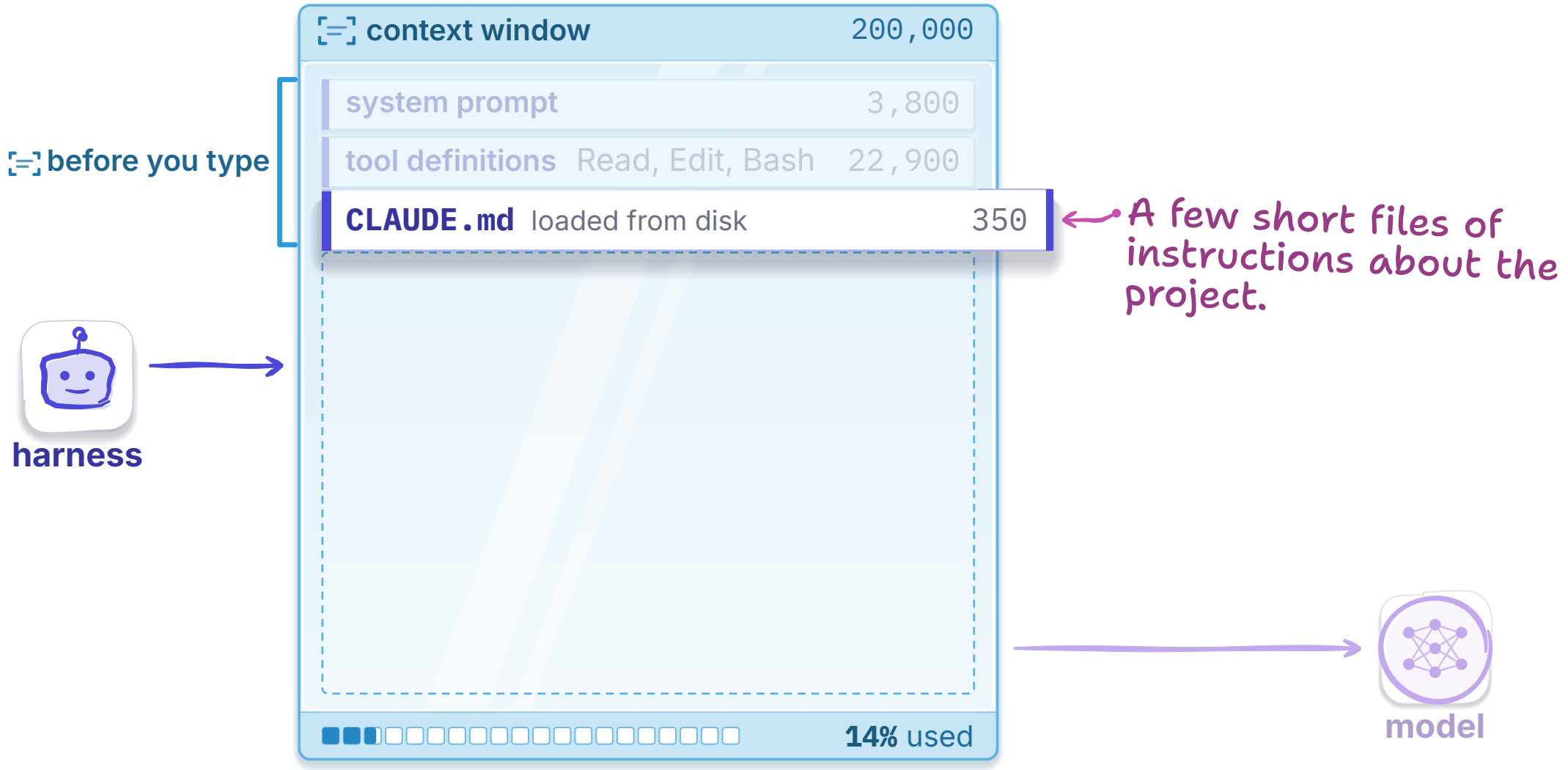
illustrative values

The harness decides what goes into the window



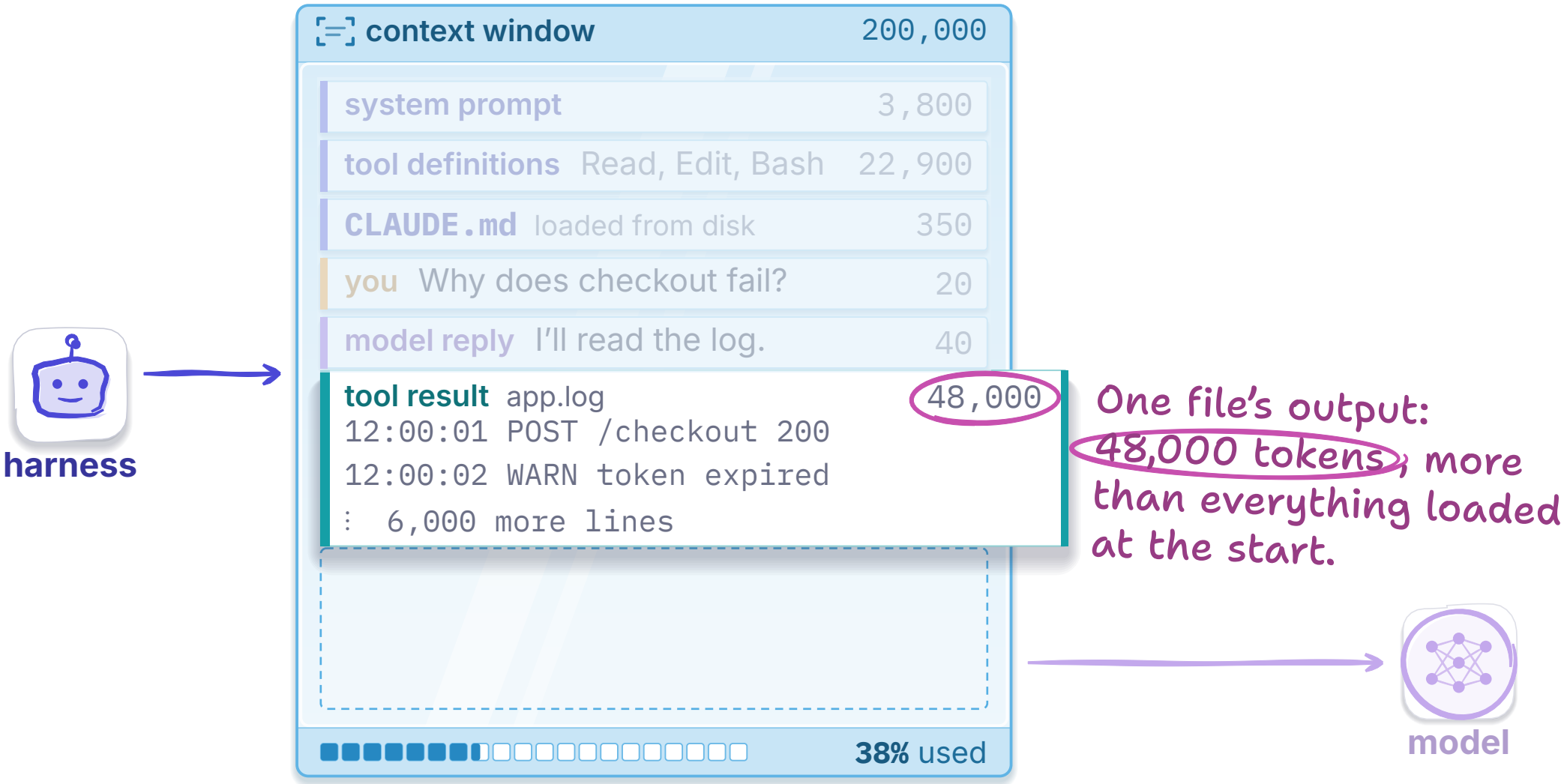
illustrative values

The harness decides what goes into the window



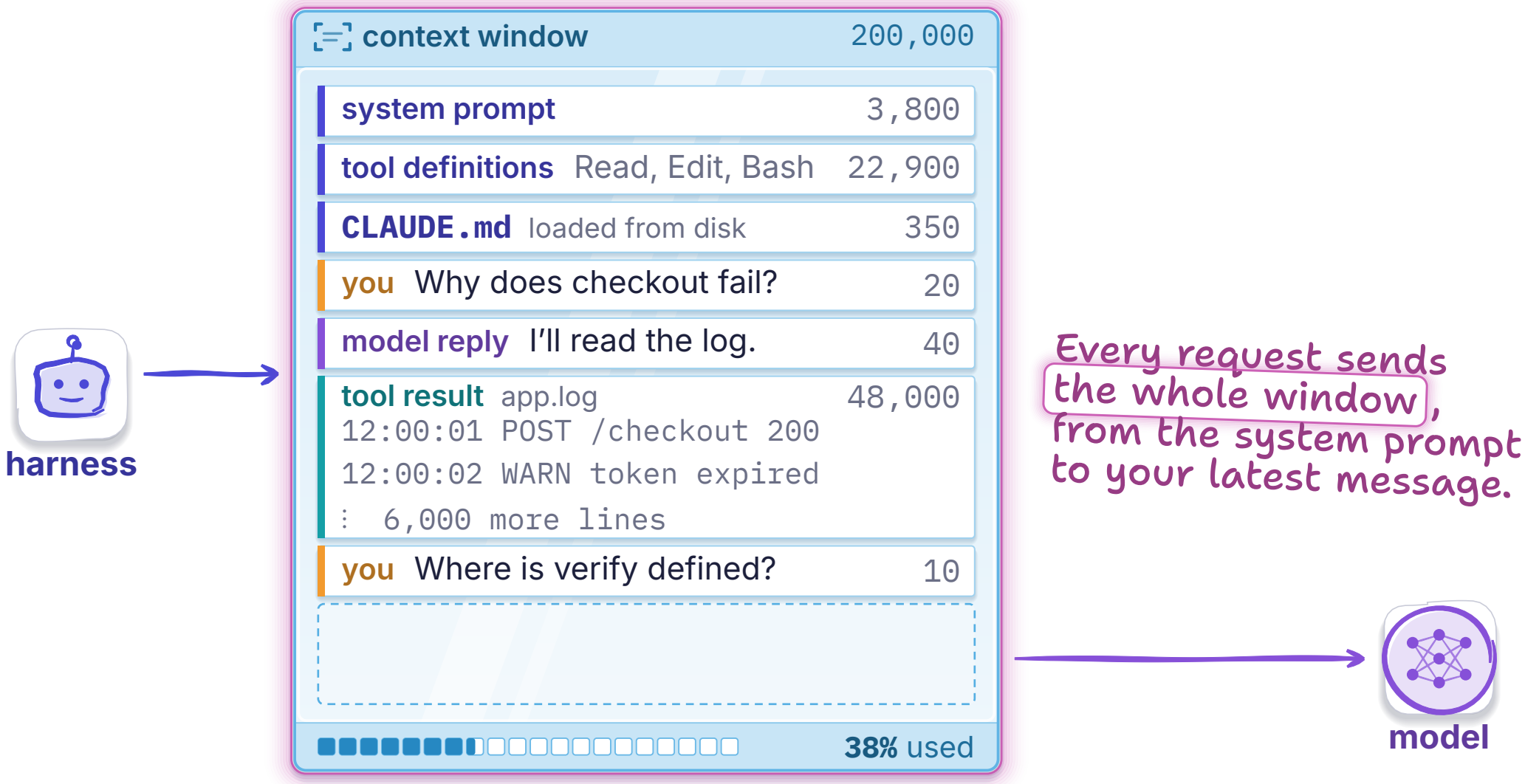
illustrative values

The harness decides what goes into the window



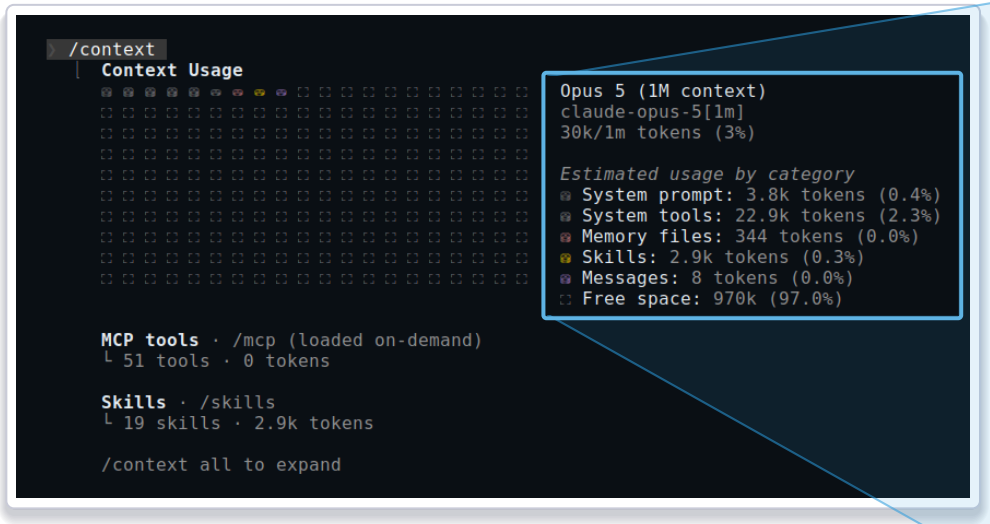
illustrative values

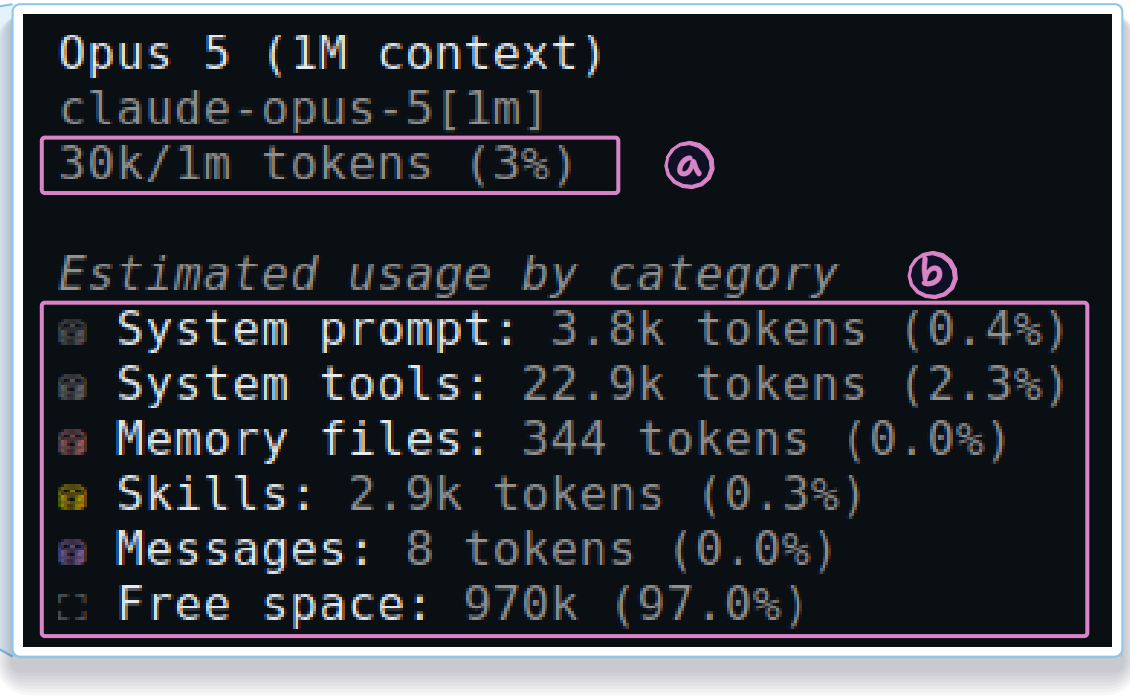
The harness decides what goes into the window



illustrative values

/context shows what is in the window



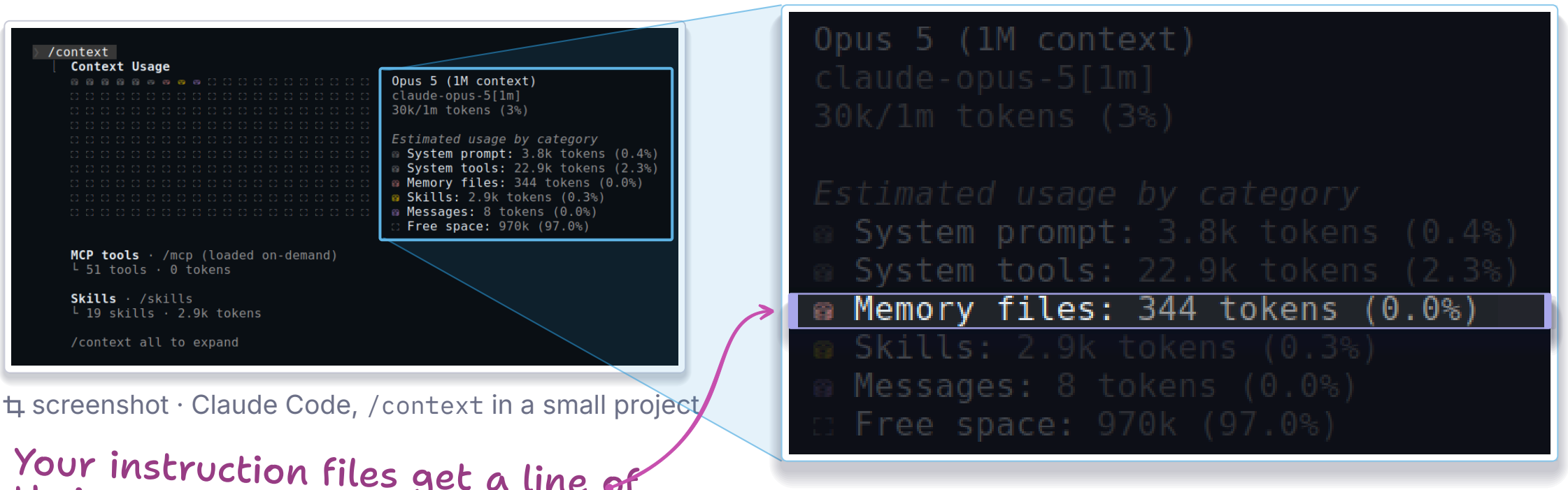


screenshot · Claude Code, `/context` in a small project

ⓐ how much of the window is used

ⓑ what fills it, by category, in tokens

/context shows what is in the window

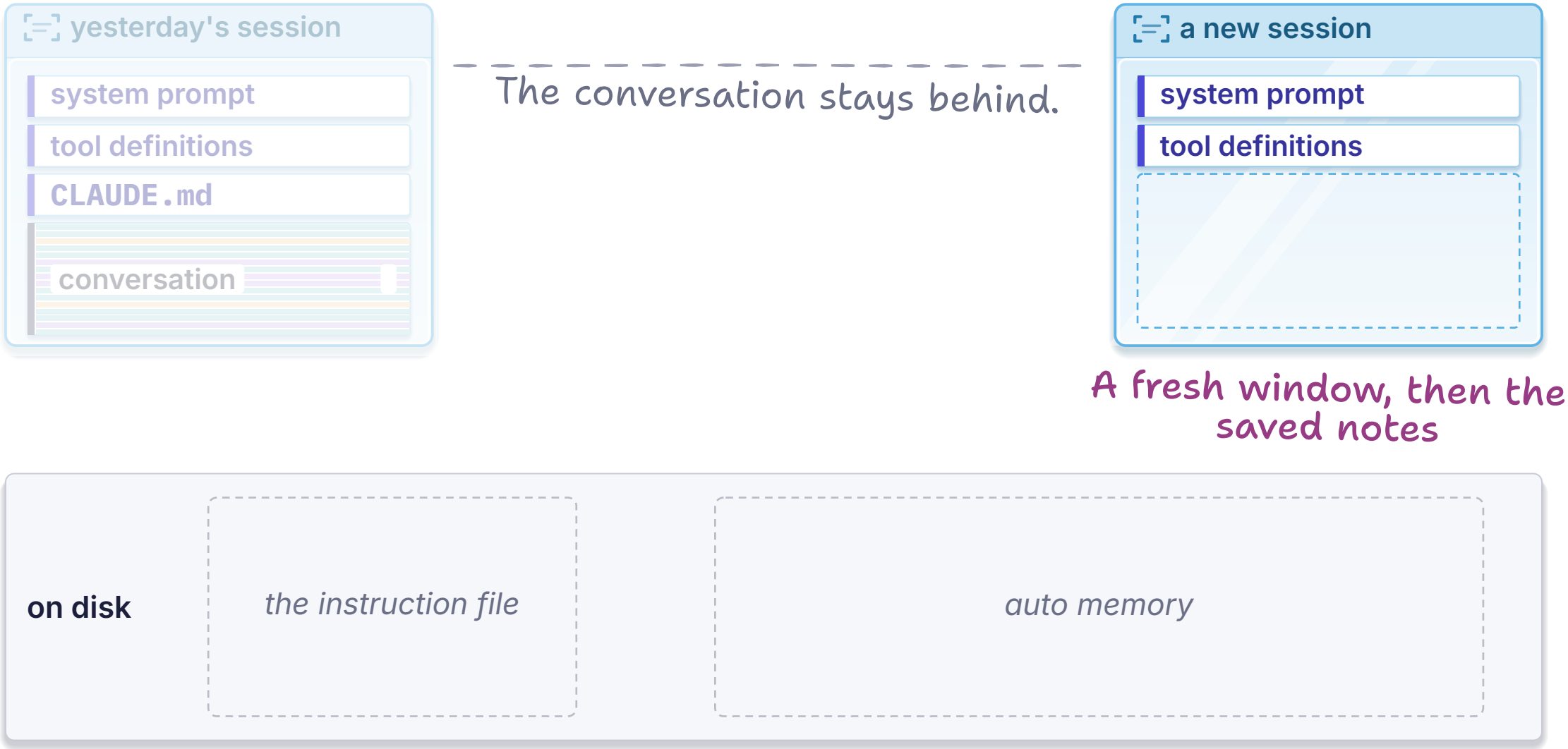


screenshot · Claude Code, `/context` in a small project

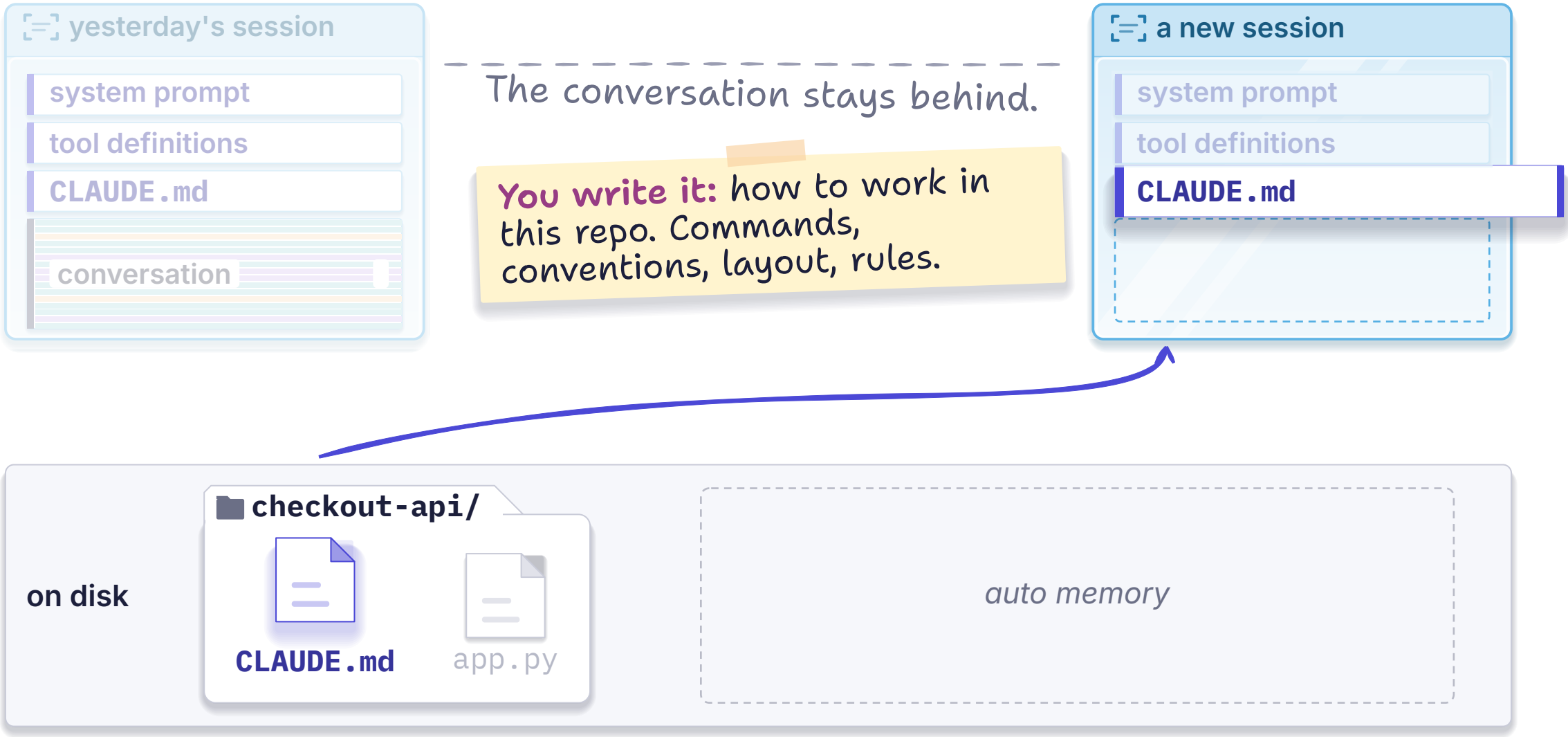
Your instruction files get a line of their own. Is the file you expect counted?

`/memory` finds the file and opens it for editing

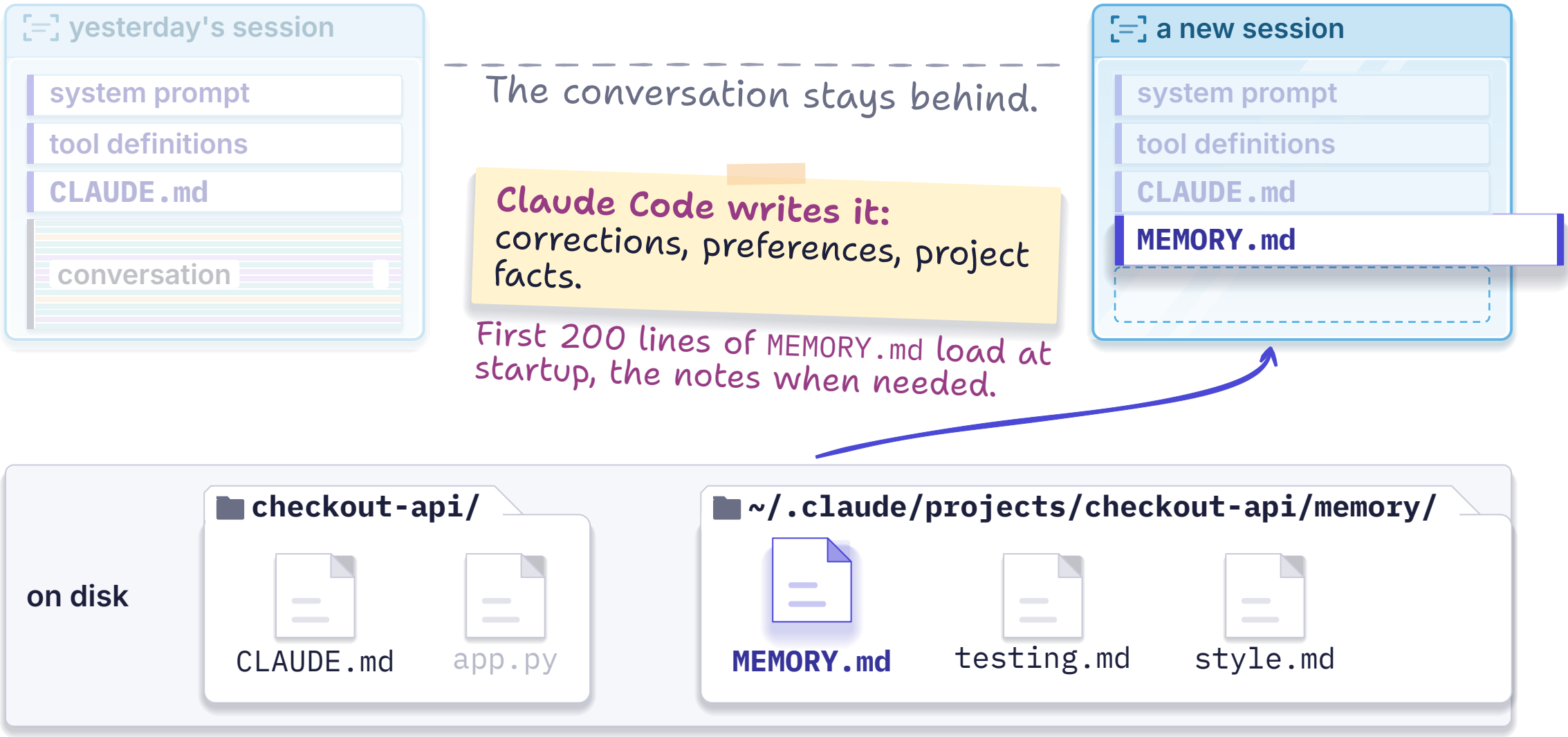
Two kinds of memory carry across sessions



Two kinds of memory carry across sessions



Two kinds of memory carry across sessions



Two kinds of memory carry across sessions

yesterday's session

system prompt

tool definitions

CLAUDE.md

conversation

The conversation stays behind.

The model forgets. What it must know about your project is yours to write down, and to check.

a new session

system prompt

tool definitions

CLAUDE.md

MEMORY.md

on disk

checkout-api/

CLAUDE.md

app.py

~/.claude/projects/checkout-api/memory/

MEMORY.md

testing.md

style.md

`/init` drafts an instruction file for you

agent session `~/checkout-api`

you typed by you

```
/init
```

tool call asked by the model

Read `pyproject.toml`

tool call asked by the model

Bash `ls src tests`

agent written by the model

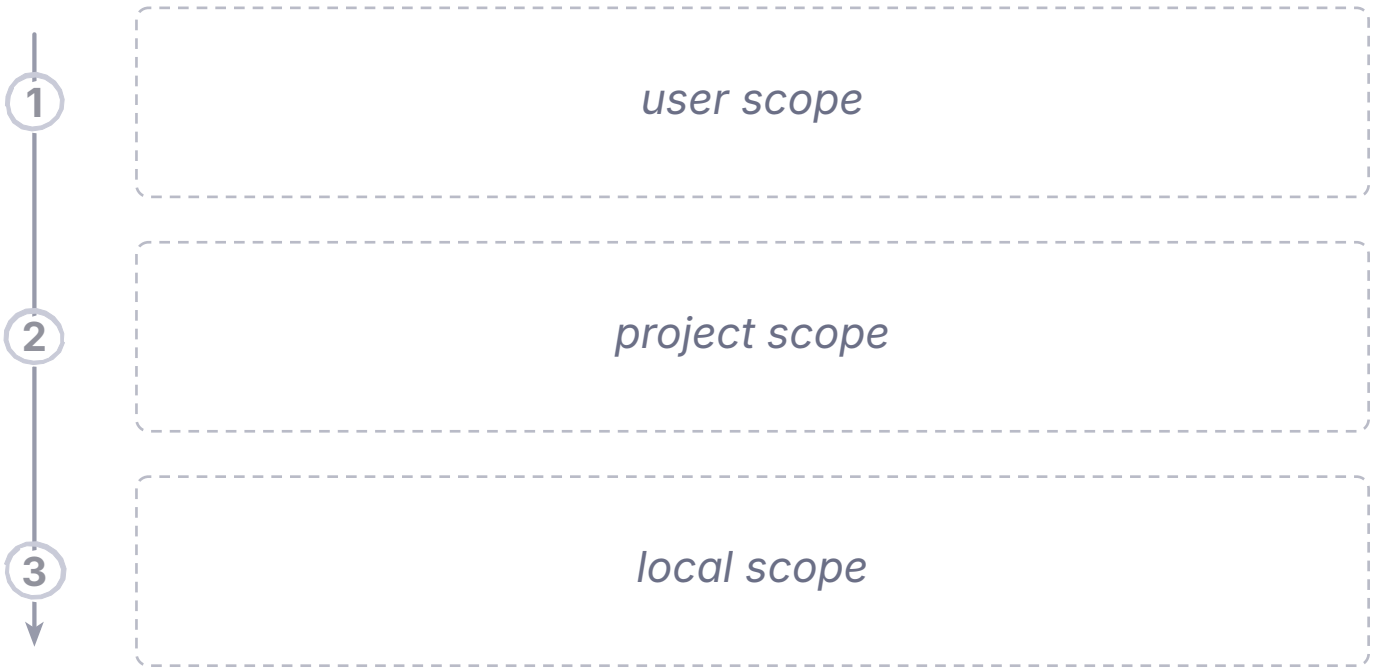
I wrote a draft `CLAUDE.md` from the build commands, conventions and files I found.

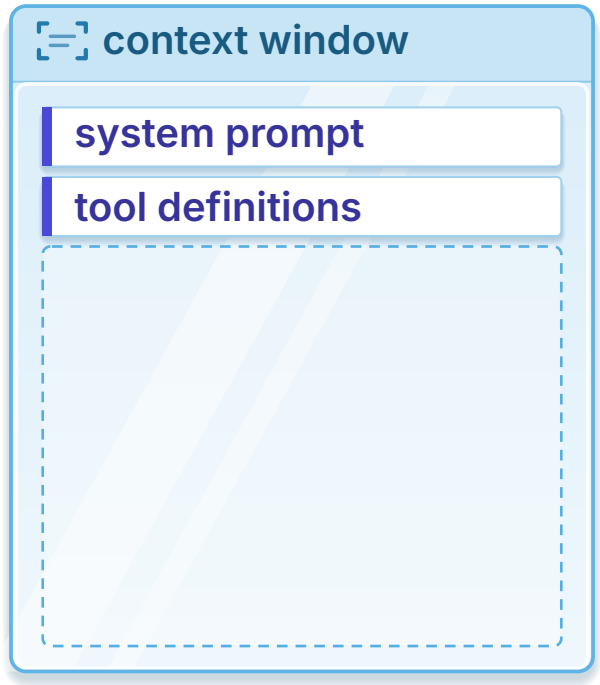
`CLAUDE.md` — draft

```
# checkout-api
## Build
- Test: uv run pytest -q
- Lint: uv run ruff check .
## Layout
- src/checkout/: the API
- tests/: pytest tests
```

Your turn: review the draft, run its commands, fix what it misunderstood.

Instructions load from several places, broad to local





Read broadest first. They must agree: a file read later does not reliably settle a conflict.

Instructions load from several places, broad to local



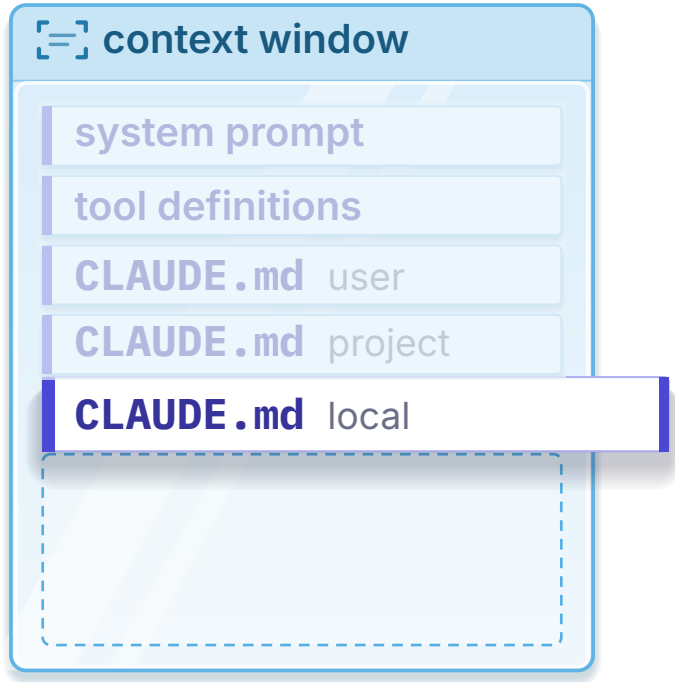
Instructions load from several places, broad to local



git commit shared with your team

Instructions load from several places, broad to local

1. **user** `~/.claude/CLAUDE.md`
   your rules, for all your projects
2. **project** `./CLAUDE.md` or `./.claude/CLAUDE.md`
   committed to Git: the whole team reads it
3. **local** `./CLAUDE.local.md`
   your notes for this project, kept out of Git

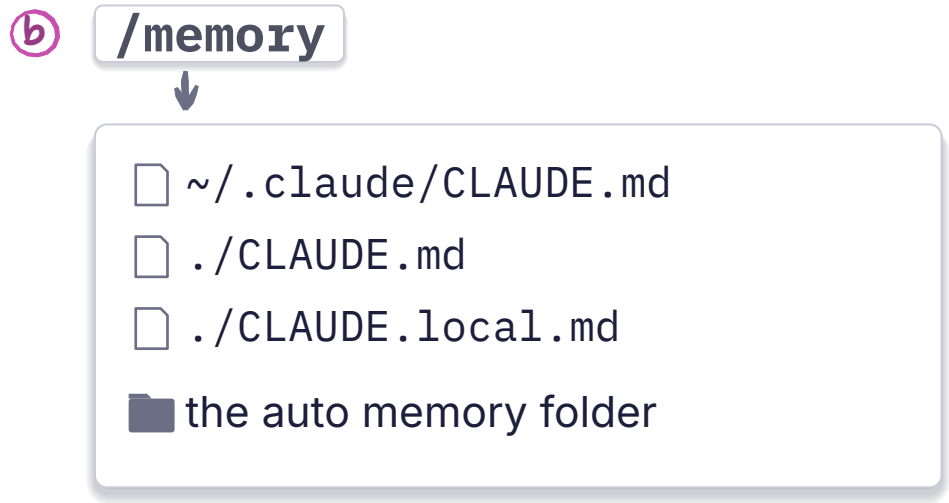


`.gitignore` add it here, so it stays out of commits

Two useful additions



ⓐ `@path` imports a file. One line brings in `AGENTS.md`, the name other agentic CLIs read.



ⓑ opens the instruction files for editing, and the auto memory folder too.

What makes a good instruction file

CLAUDE.md 1 line

```
1 # checkout-api
```

It explains how to work in this repository: the commands, the conventions, the layout and the rules.

What makes a good instruction file

CLAUDE.md — 5 lines

```
# checkout-api

## Build
- Test: uv run pytest -q
- Lint: uv run ruff check .
```

Short. A cheat sheet of commands and standing rules: under a couple hundred lines, as a rule of thumb. Detailed docs stay in the project.

What makes a good instruction file

CLAUDE.md — 8 lines

```
1 # checkout-api
2
3 ## Build
4 - Test: uv run pytest -q
5 - Lint: uv run ruff check .
6
7 ## Style
8 - Type everything; run mypy --strict
```

Short. A cheat sheet of commands and standing rules: under a couple hundred lines, as a rule of thumb. Detailed docs stay in the project.

Structured. Headings and short lists group related rules.

What makes a good instruction file

CLAUDE.md — 11 lines

```
# checkout-api

## Build
- Test: uv run pytest -q
- Lint: uv run ruff check .

## Style
- Type everything; run mypy --strict

## Don't touch
- migrations/ is generated: leave it alone
```

Short. A cheat sheet of commands and standing rules: under a couple hundred lines, as a rule of thumb. Detailed docs stay in the project.

Structured. Headings and short lists group related rules.

Specific. Exact commands and real paths, not vague wishes.

A concrete rule gives the agent a clear action

vague

```
- Write clean code.
```

Leaves many choices open.

concrete

```
- Run ruff check before finishing.
```

Names a check, and its result shows in the session.

Write rules you can check. In the file, a rule is available to the agent. It is not guaranteed to be followed.

agent session ~/checkout-api

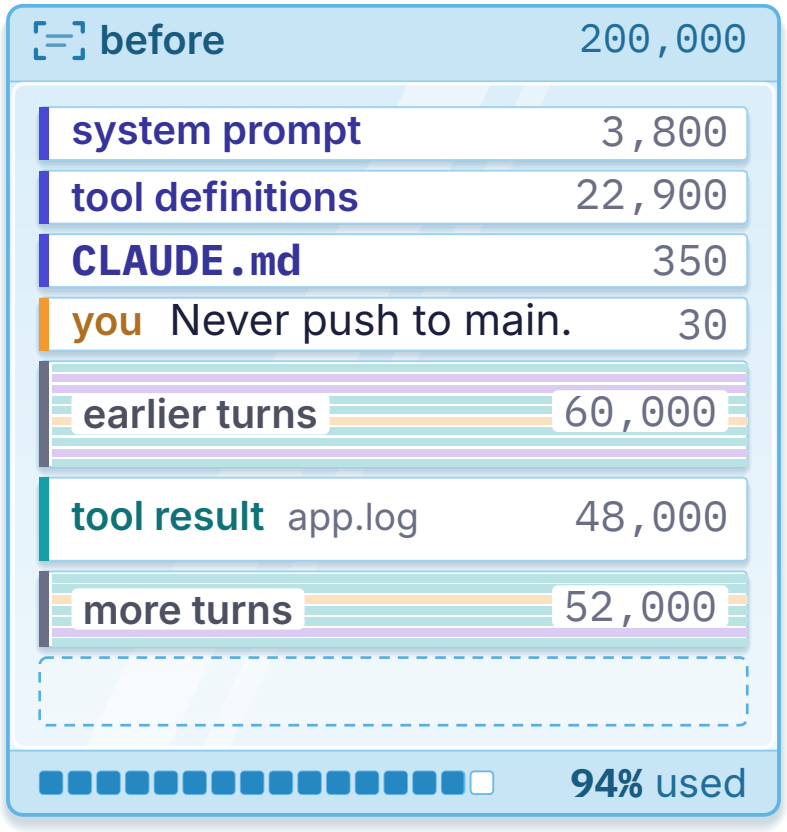
tool call asked by the model

```
Bash uv run ruff check .
```

tool result run by the harness

```
All checks passed!
```

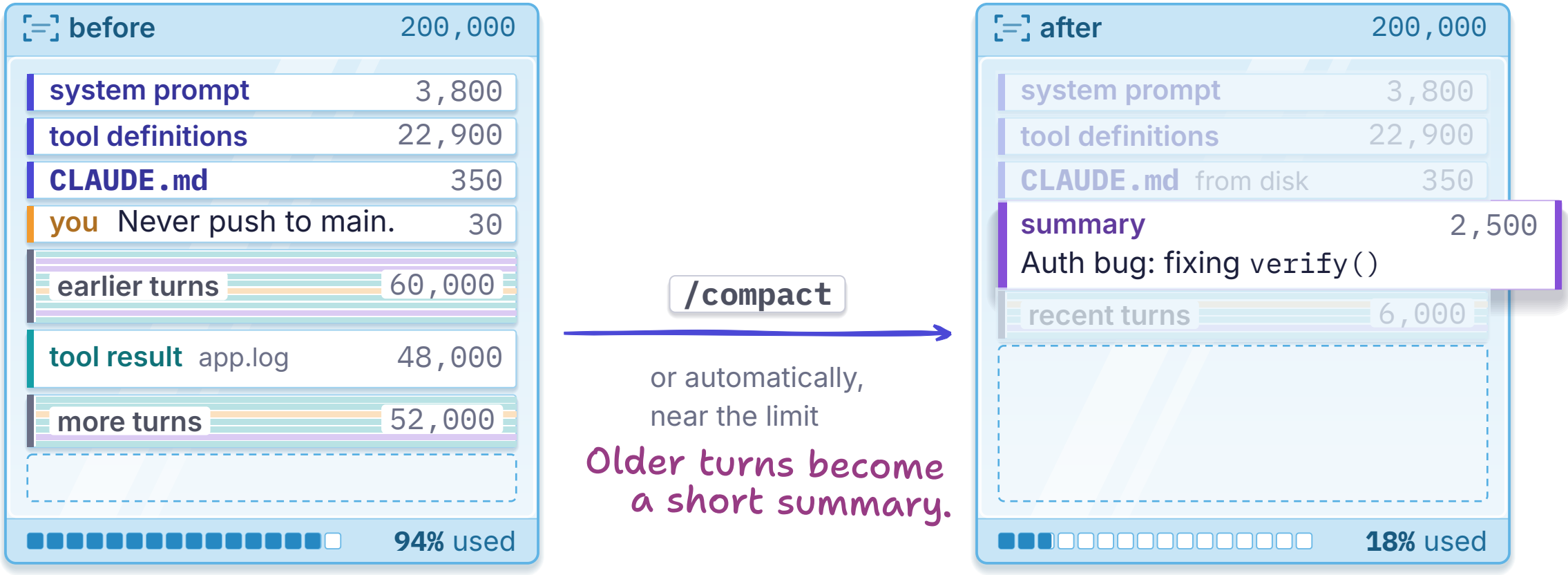
When the window fills, the harness compacts



94% used. A request that no longer fits cannot be sent, so the harness must make room.

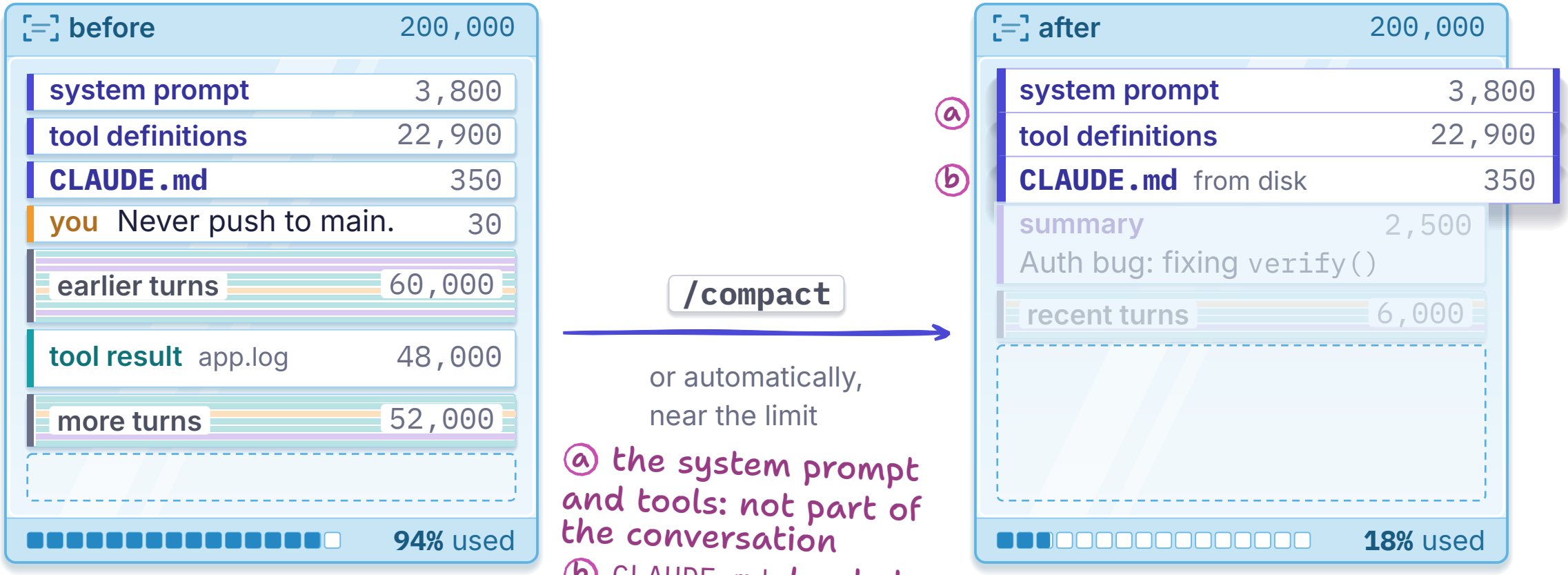
illustrative values

When the window fills, the harness compacts



Say what to keep: `/compact focus on the auth bug`

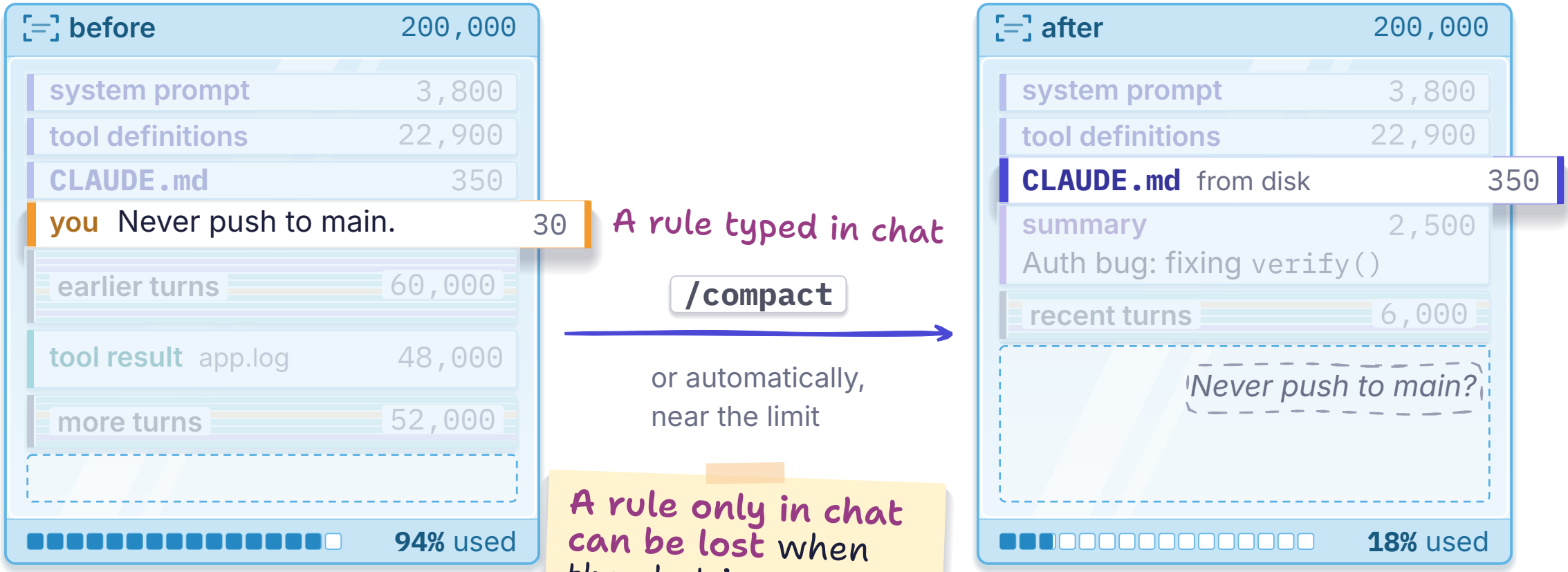
When the window fills, the harness compacts



Details that were only in the chat survive only if the summary kept them.

illustrative values

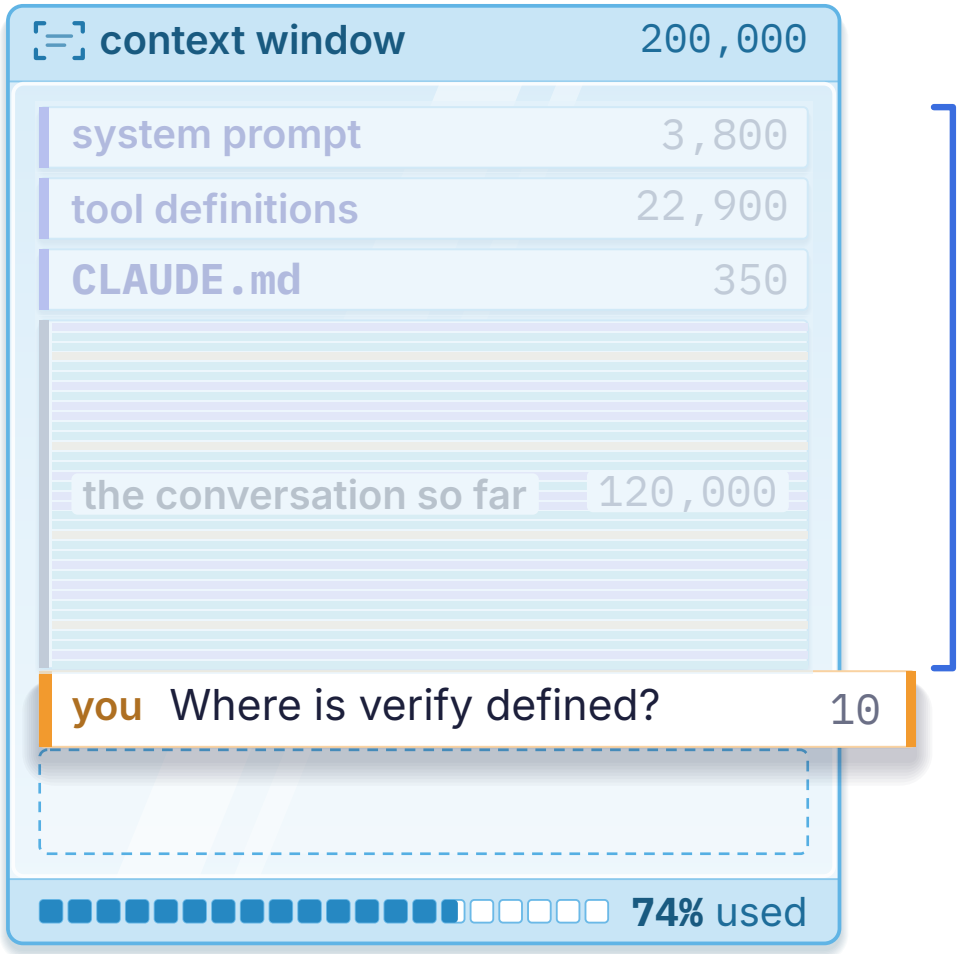
When the window fills, the harness compacts



A rule only in chat can be lost when the chat is summarized. Put standing rules in `CLAUDE.md`.

illustrative values

A long window costs time, tokens and attention



Every request sends the whole window. An unchanged beginning, the prefix, was already processed: the provider reuses that work.

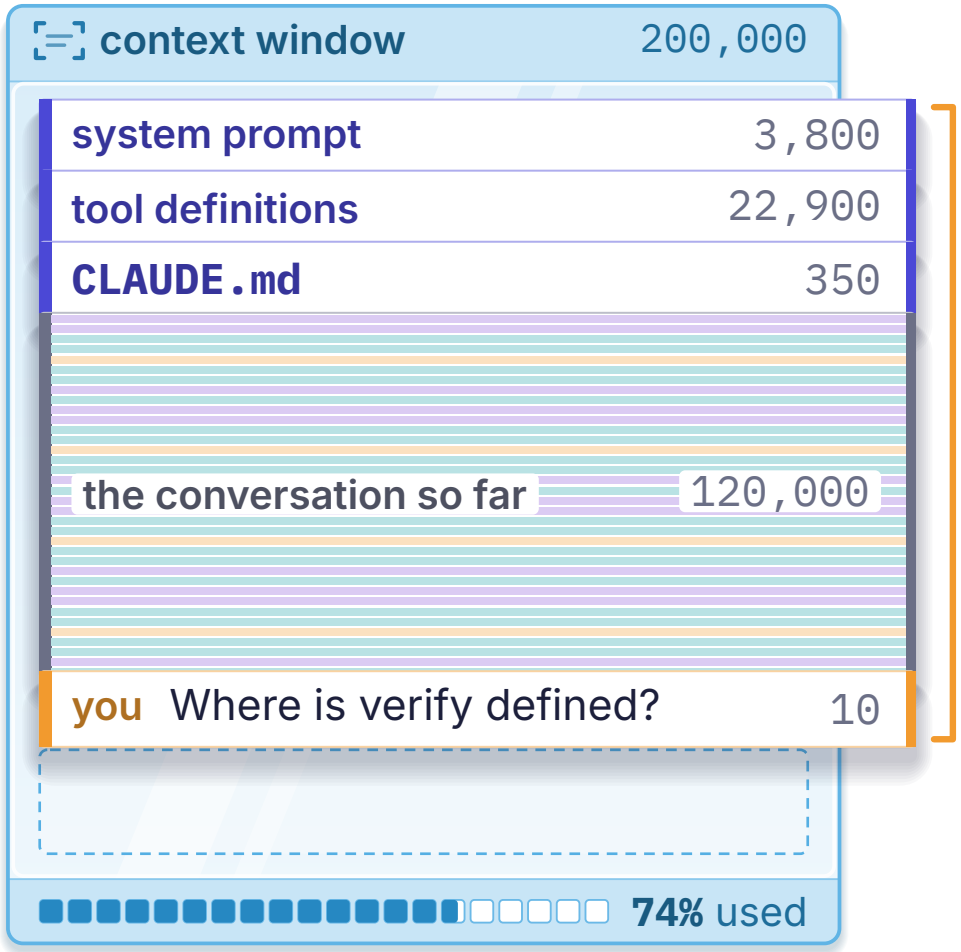
Processed before the first output token

cache hit

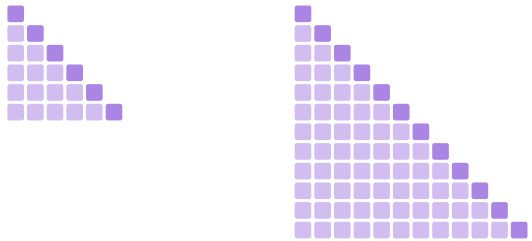
cache miss

illustrative values

A long window costs time, tokens and attention



processed again



6 tokens
21 comparisons

12 tokens
78 comparisons

Each position compares with every position before it.

Processed before the first output token

cache hit

cache miss

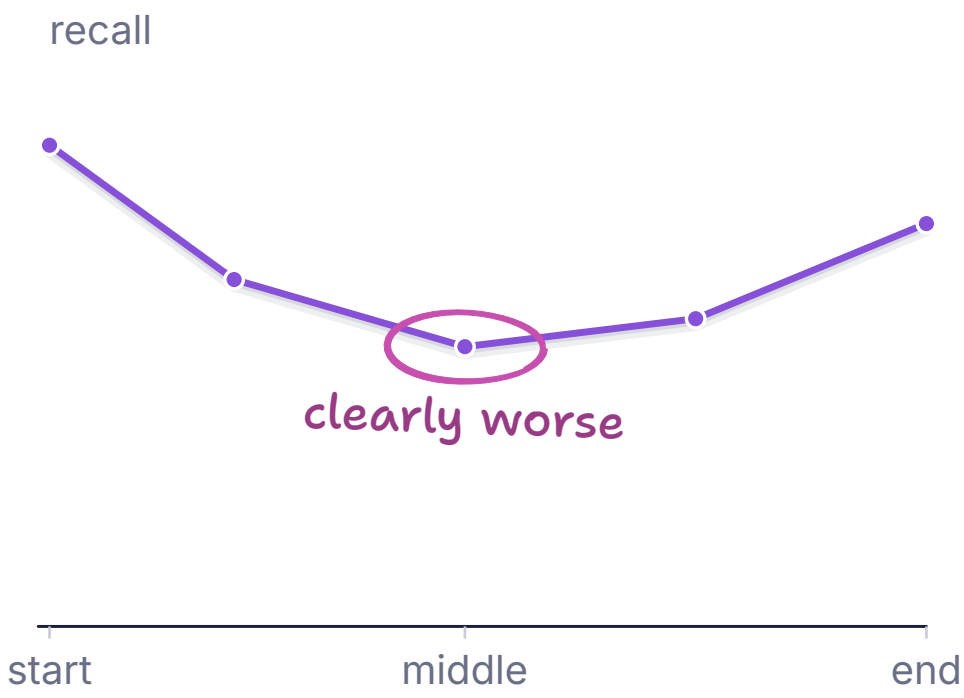
More time to the first token, and more input tokens to pay for.

illustrative values

Models do not use a long window evenly

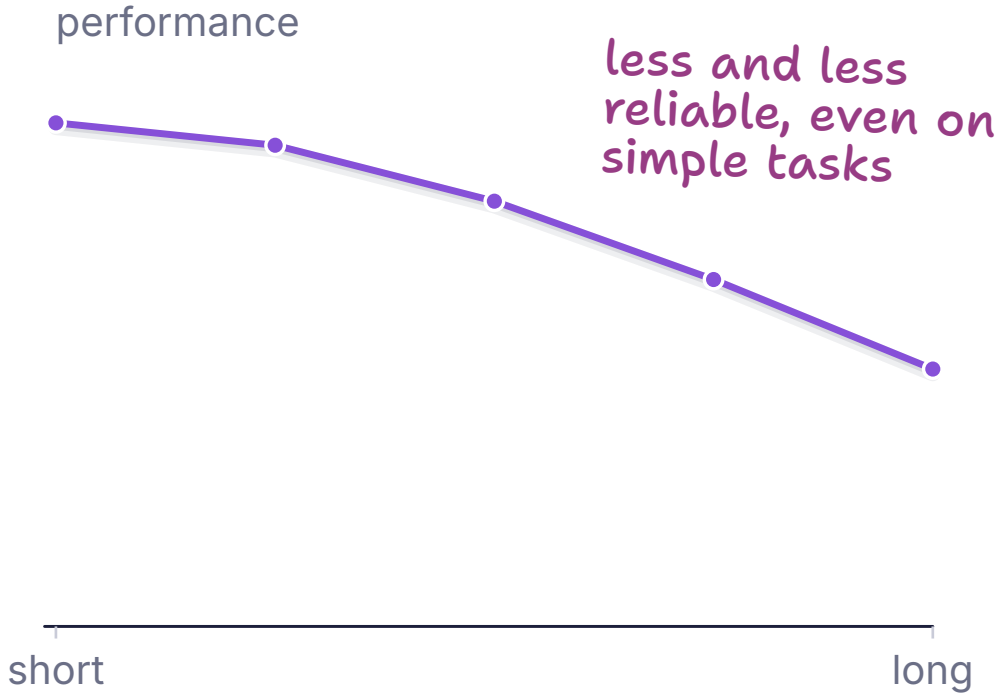
Where the one relevant fact sits

Liu et al., *Lost in the Middle*, TACL 2024



As the input gets longer

Hong et al., *Context Rot*, Chroma 2025 · 18 models



schematic shapes, after the two studies

Fitting in the window is only the first requirement

It fits



74% used

It costs more to process



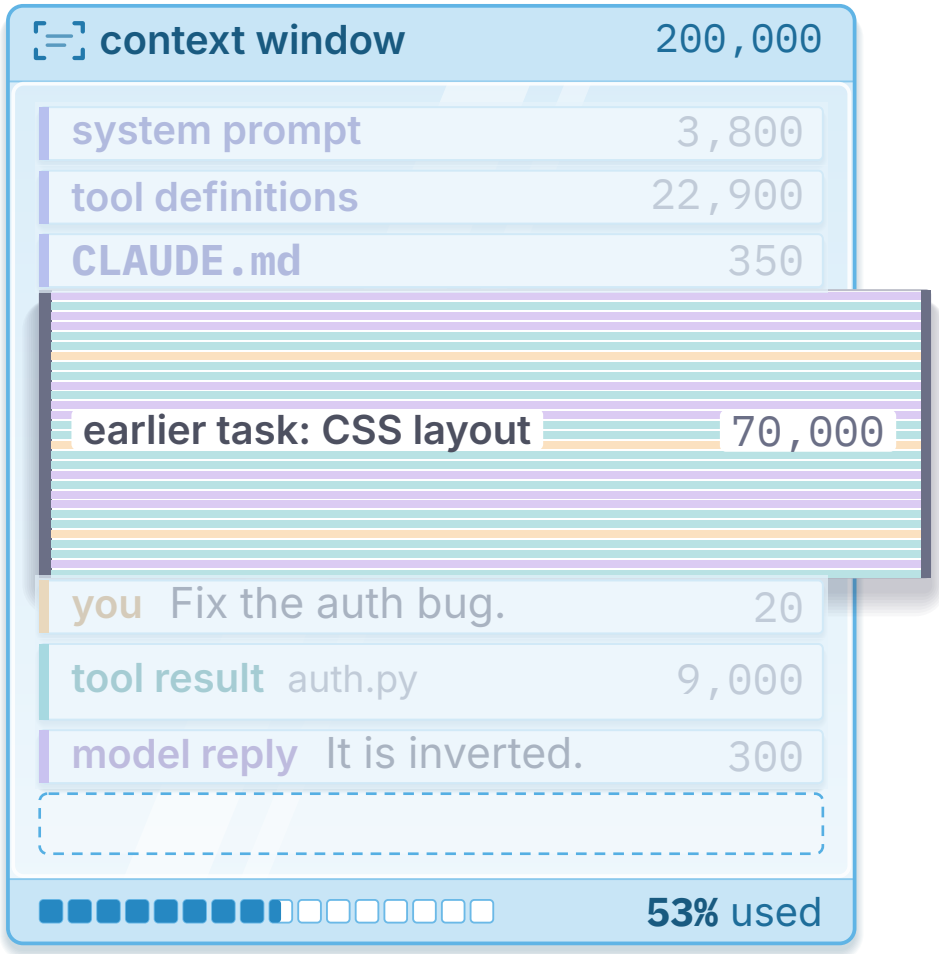
time and input tokens

Details can be missed



the middle is read least well

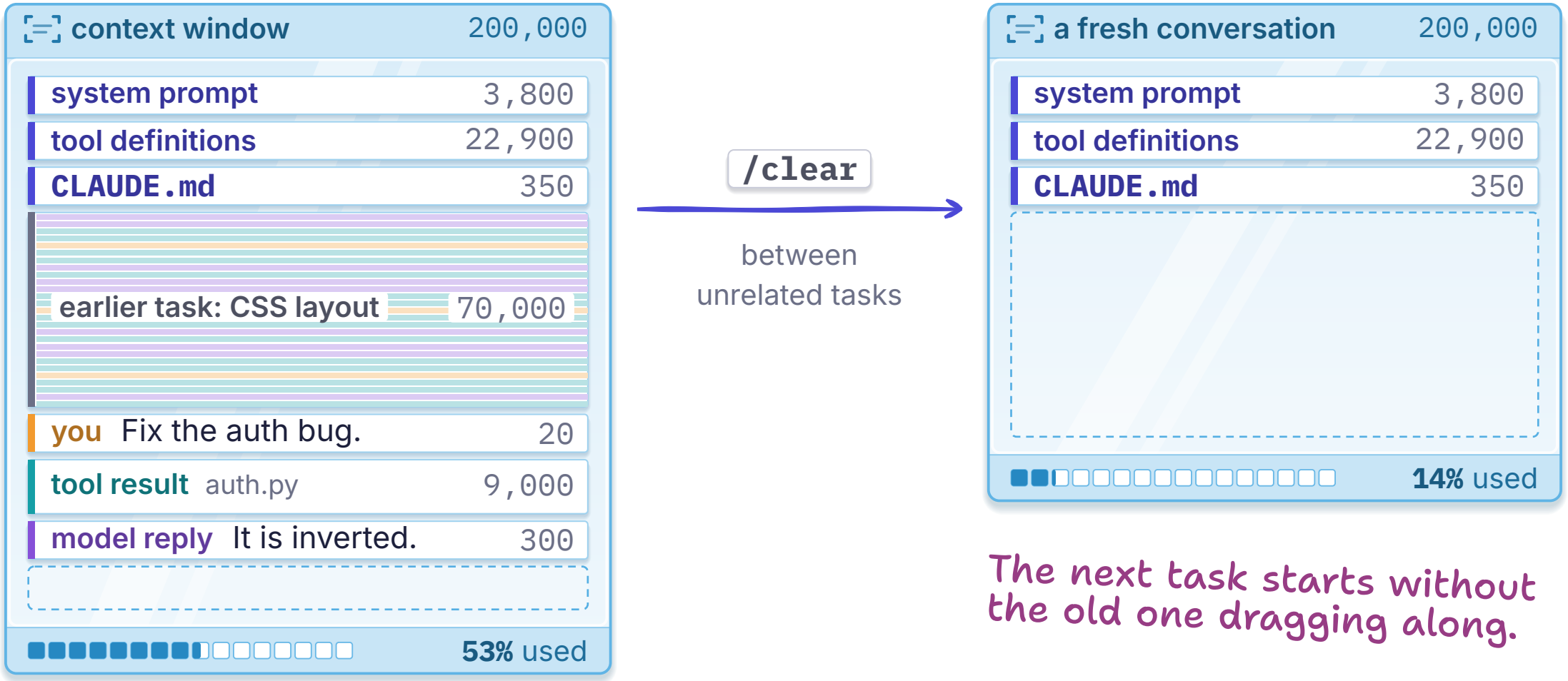
Keep the window on the task



Old, unrelated conversation takes room that useful project information could use.

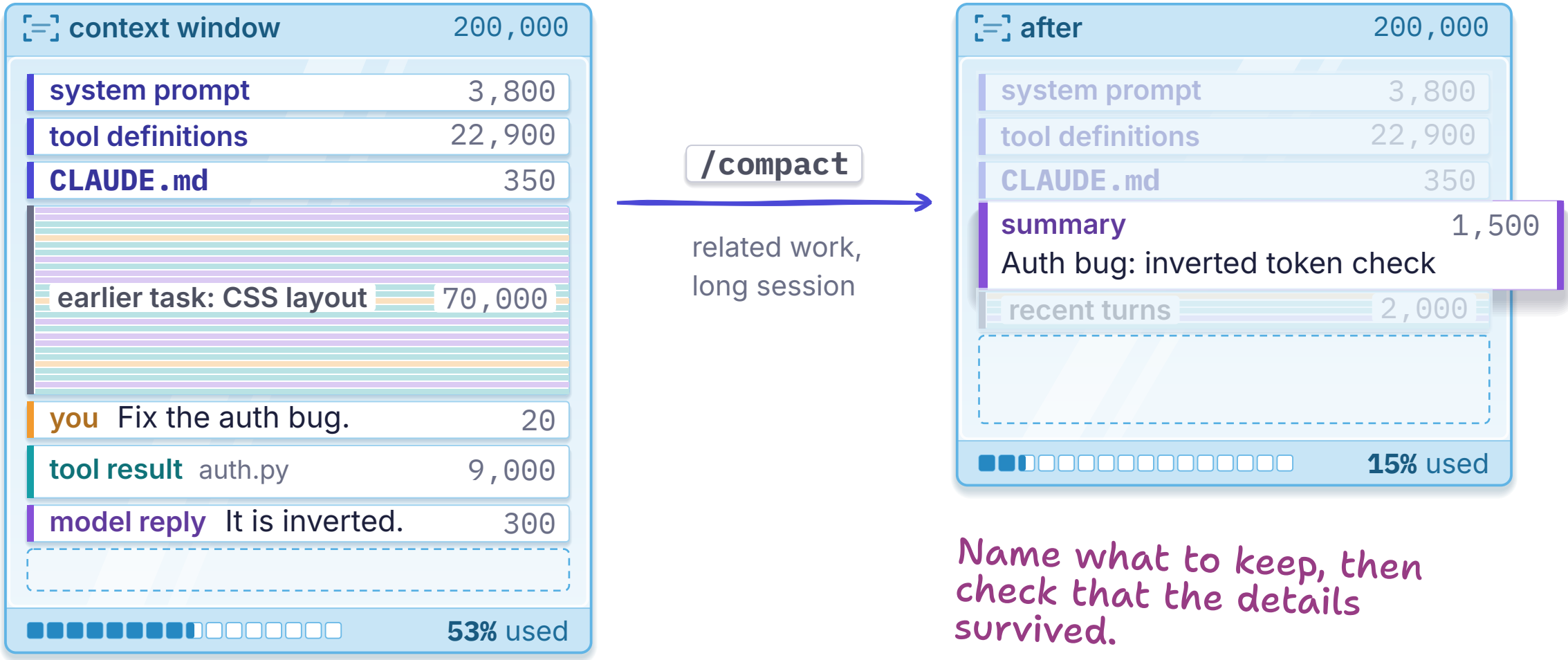
illustrative values

Keep the window on the task



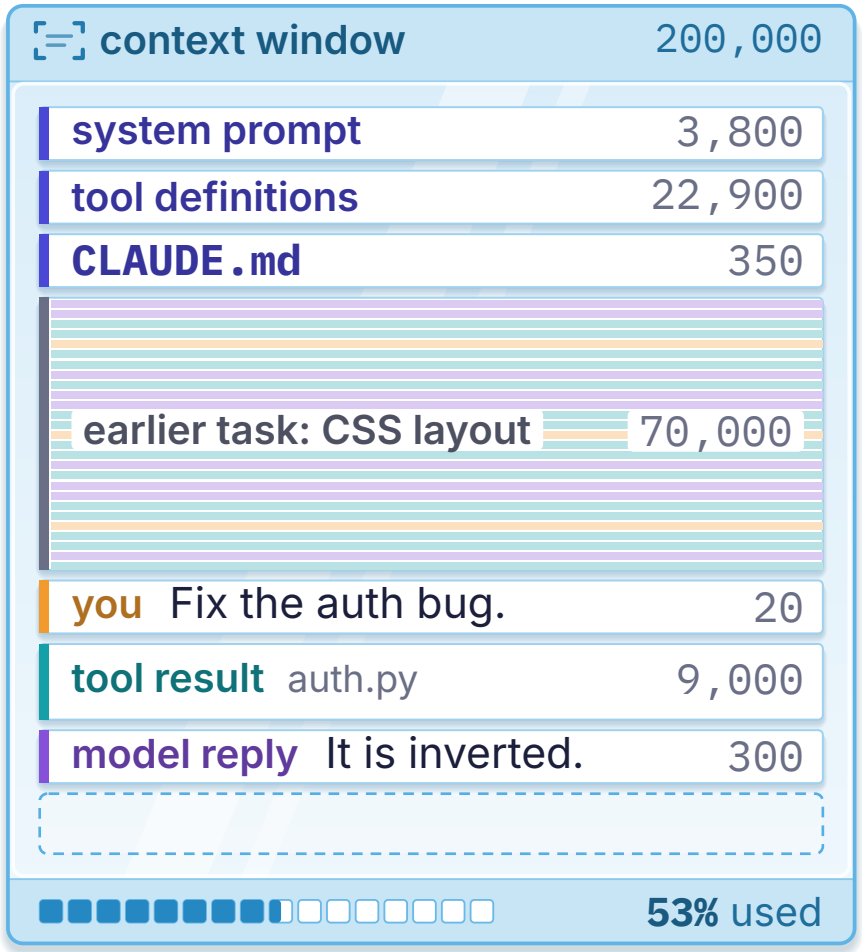
illustrative values

Keep the window on the task



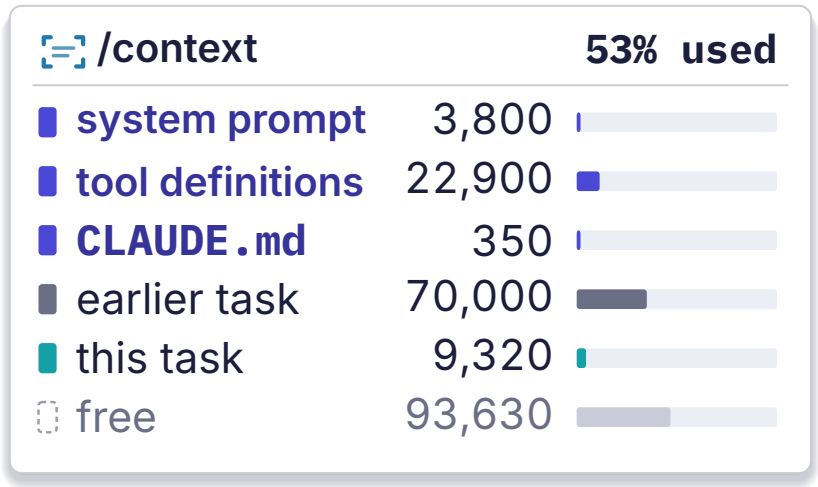
illustrative values

Keep the window on the task



/context

what fills
the window

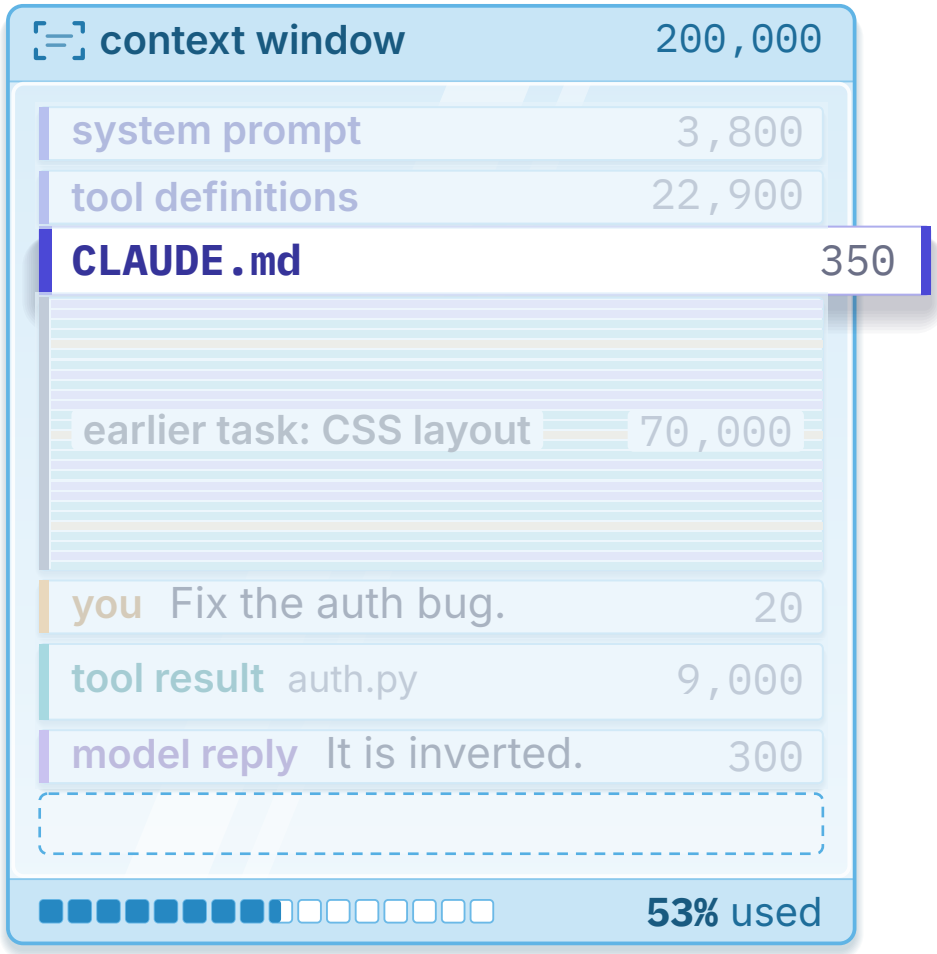


Mostly the task you are on?
/compact and keep going.

Mostly old, unrelated work?
/clear and start fresh.

illustrative values

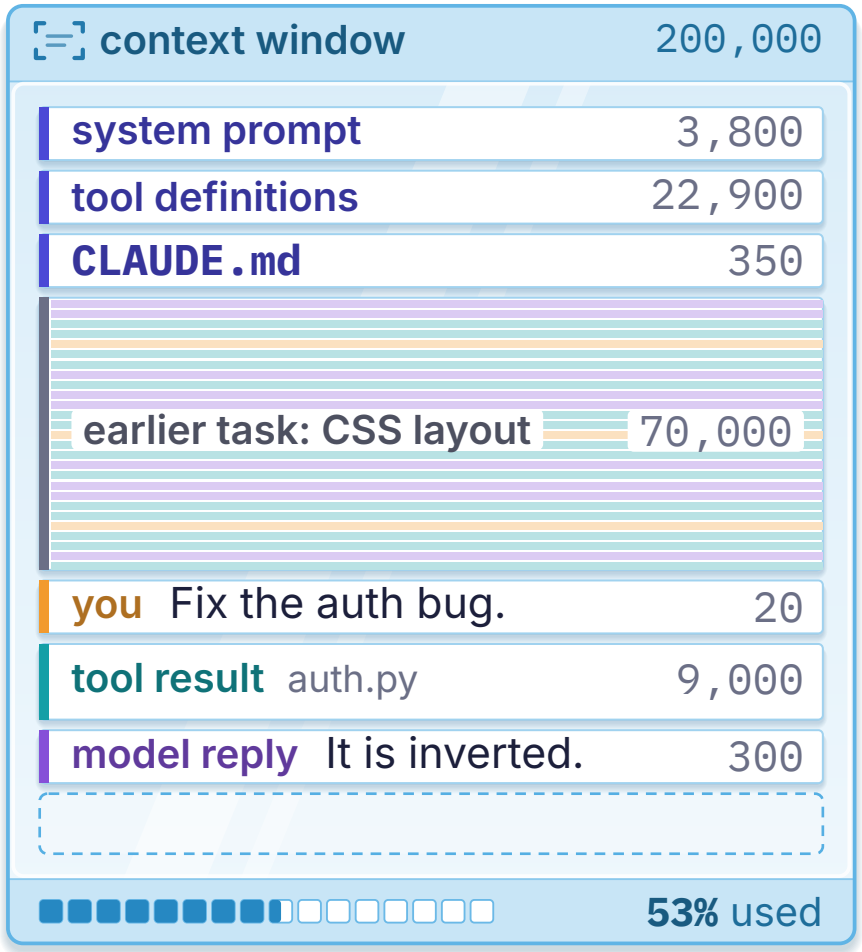
Keep the window on the task



In the window on every request. Keep it lean: more room for the real work.

illustrative values

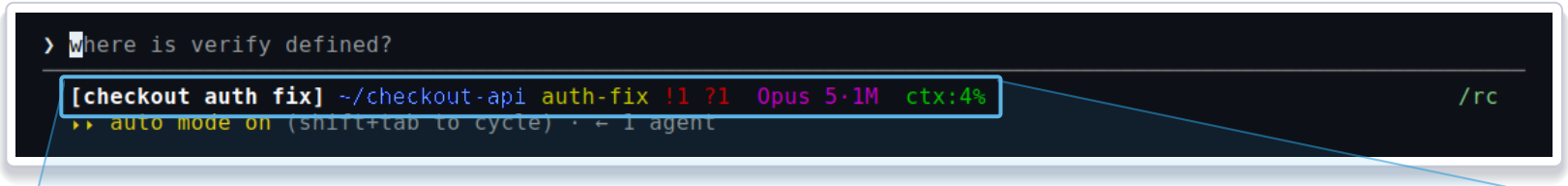
Keep the window on the task



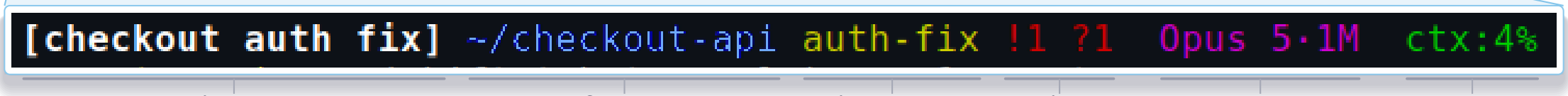
`/clear`	unrelated work
`/compact`	keep the thread of a long task
`CLAUDE.md`	standing rules, kept focused

illustrative values

A status line keeps the window in view



screenshot · the instructor's status line, under the prompt



session name
set with `/rename`

folder

git branch

edited, untracked

model

context used



The status line comes from a script

stdin: from Claude Code

```
{
  "session_name": "checkout auth fix",
  "model": {"display_name": "Opus 5"},
  "workspace": {
    "current_dir": "/home/you/checkout-api",
    "git_worktree": "auth-fix" },
  "context_window": {"used_percentage": 4},
  ⋮
}
```

~/.claude/statusline.sh

```
input=$(cat)  # the JSON
git branch --show-current
⋮
echo "..."
```

In a worktree, its name is in the JSON: put it on the line to tell parallel sessions apart.

It prints the line.

```
[checkout auth fix] ~/checkout-api auth-fix !1 ?1  Opus 5·1M  ctx:4%
```

Or describe the line you want

agent session `~/checkout-api`

you typed by you

`/statusline` show the session name, folder, git branch with edited and untracked counts, model and context used

tool call asked by the model

Write `~/.claude/statusline.sh`

tool call asked by the model

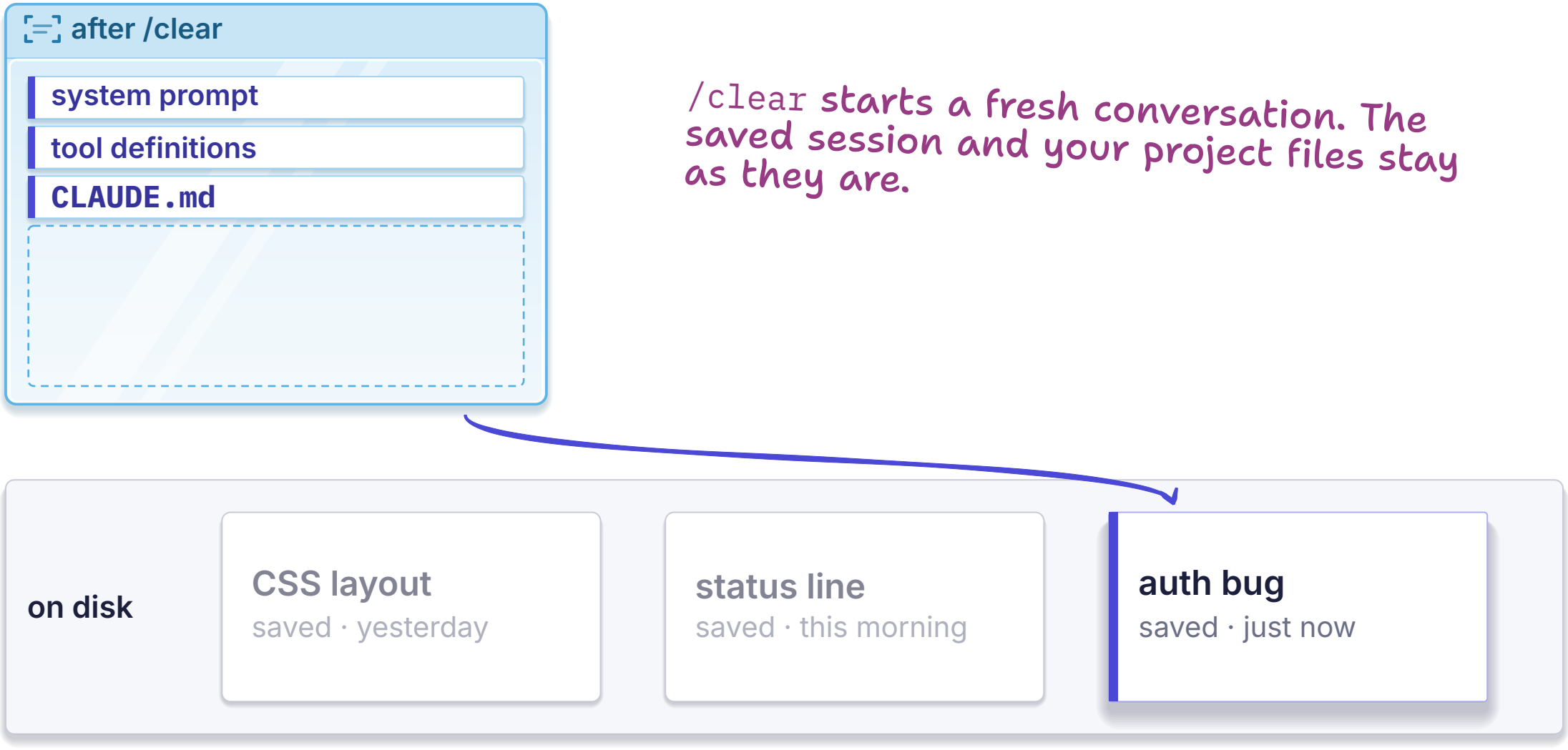
Edit `~/.claude/settings.json`

`~/.claude/settings.json`

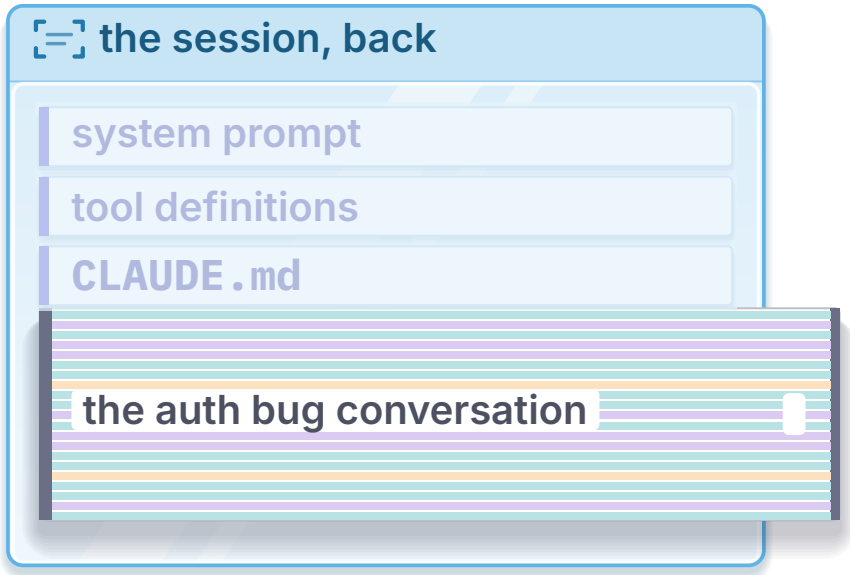
```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh"
  }
}
```

A built-in setup agent writes the script and the setting. You describe the result instead of editing the file by hand.

A cleared conversation is still saved



A cleared conversation is still saved



shell

```
~/checkout-api $ claude --continue
# or: claude -c
```

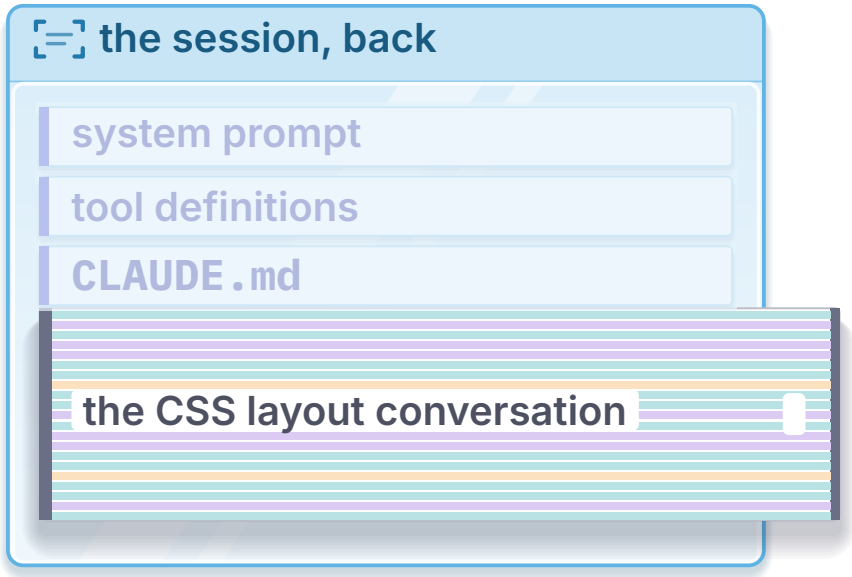
The most recent session in this folder, history and all.

Started another one since? Pick the earlier one from the resume list.

on disk

CSS layout	status line	**auth bug**
saved · yesterday	saved · this morning	saved · just now

A cleared conversation is still saved

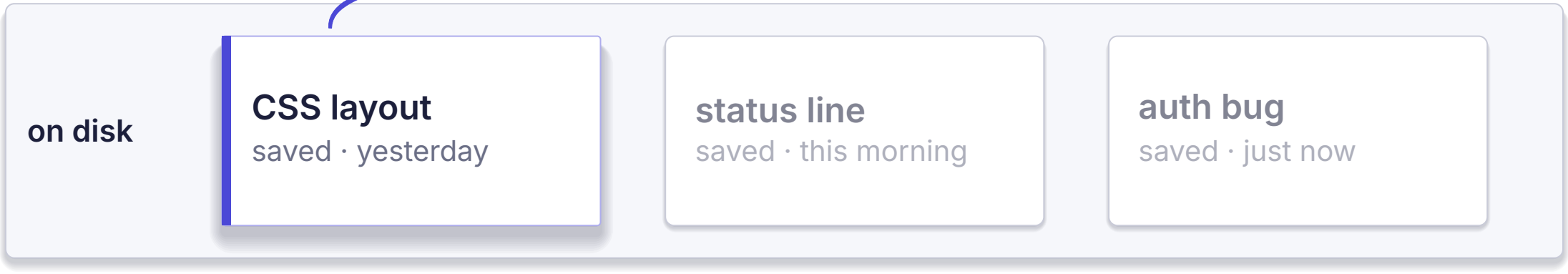


shell

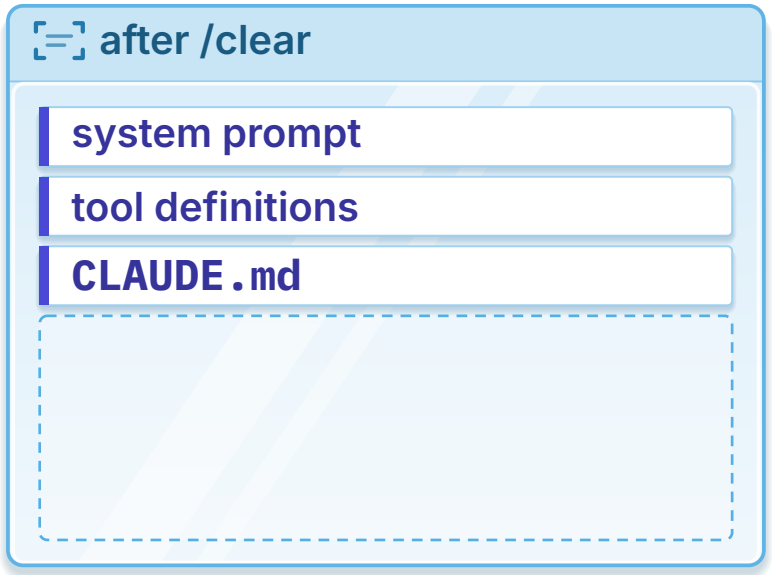
```
~/checkout-api $ claude --continue
# or: claude -c
~/checkout-api $ claude --resume
```

A list of your past sessions to pick from.

`/resume` inside a session: switch without quitting



A cleared conversation is still saved

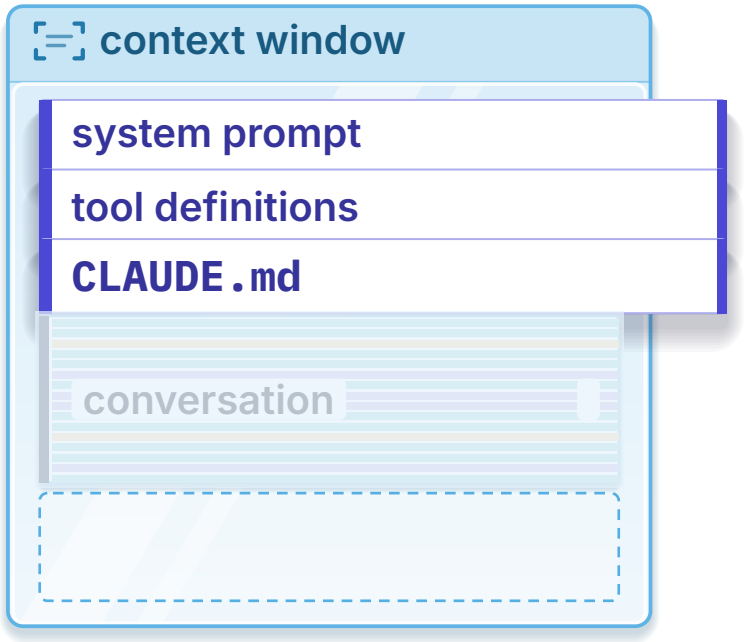


Clearing the window and deleting a saved session are different actions. Use a fresh context for new work; go back to a saved one when you need it.

on disk

- **CSS layout** — saved · yesterday
- **status line** — saved · this morning
- **auth bug** — saved · just now

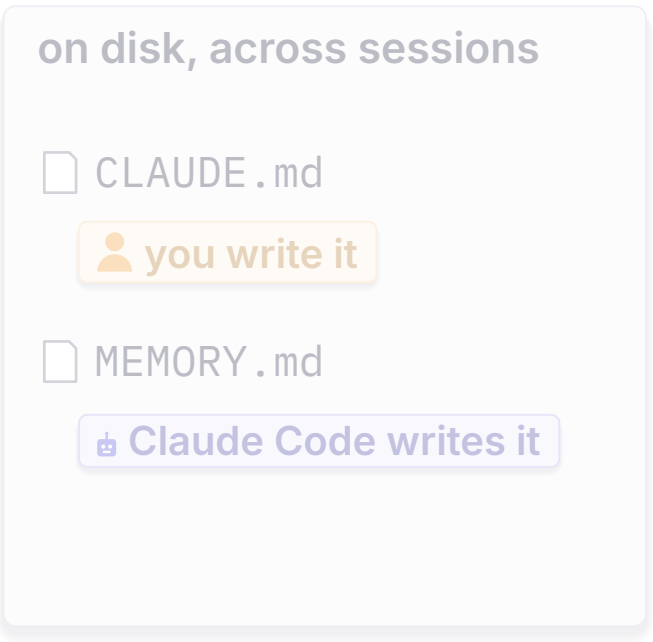
What the window holds, and what lasts



Before you type: system prompt, tools, instruction files.

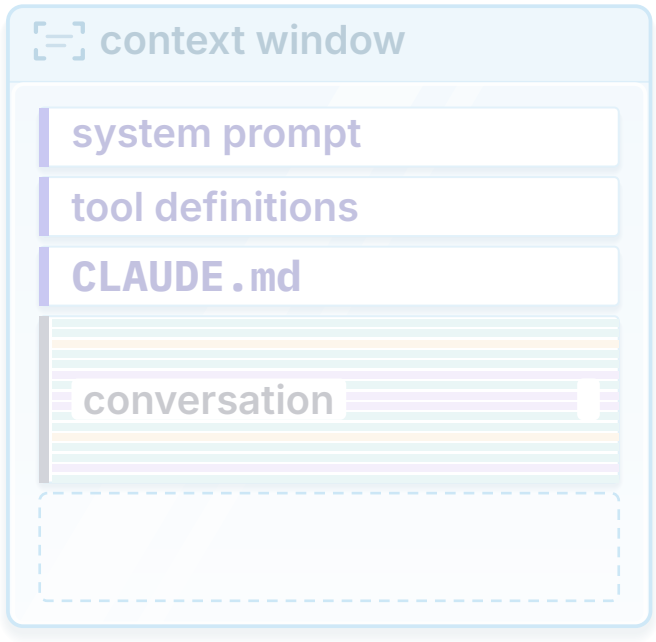


Compaction shortens the conversation. Standing rules come back from disk.

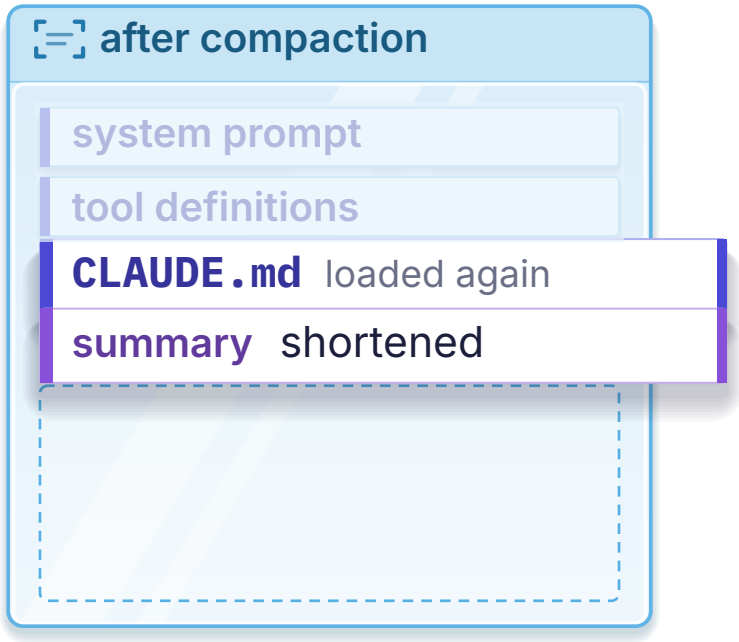


Two memories cross sessions.

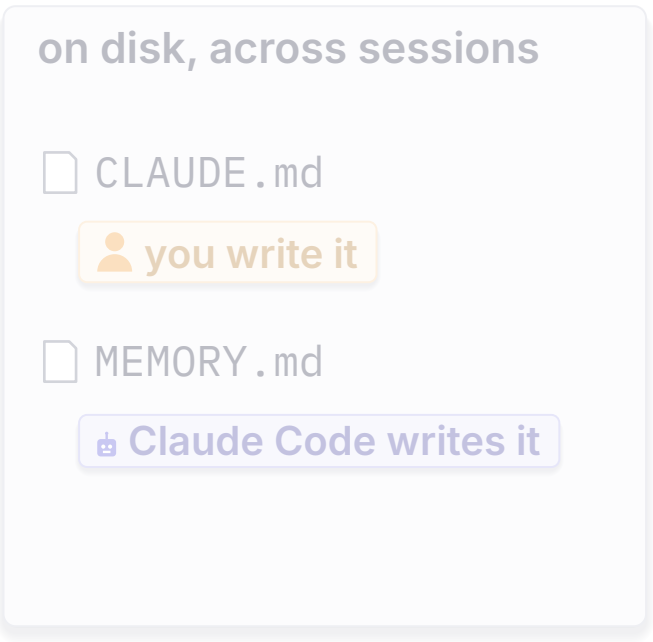
What the window holds, and what lasts



Before you type: system prompt, tools, instruction files.

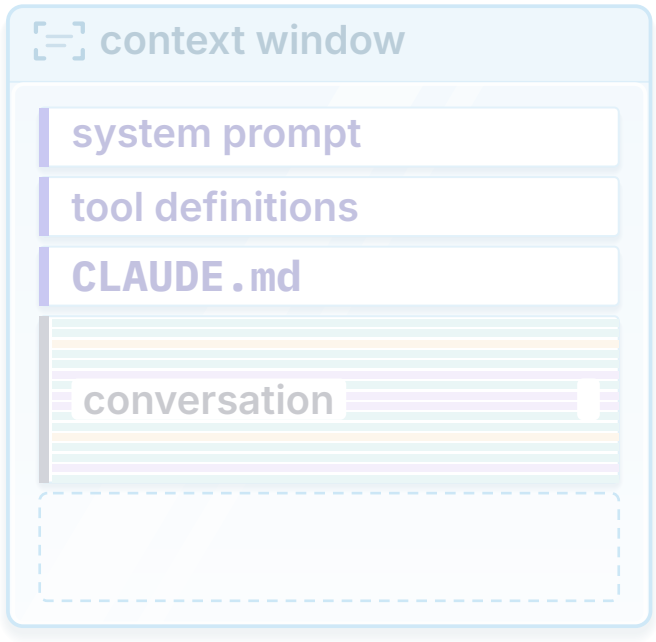


Compaction shortens the conversation. Standing rules come back from disk.



Two memories cross sessions.

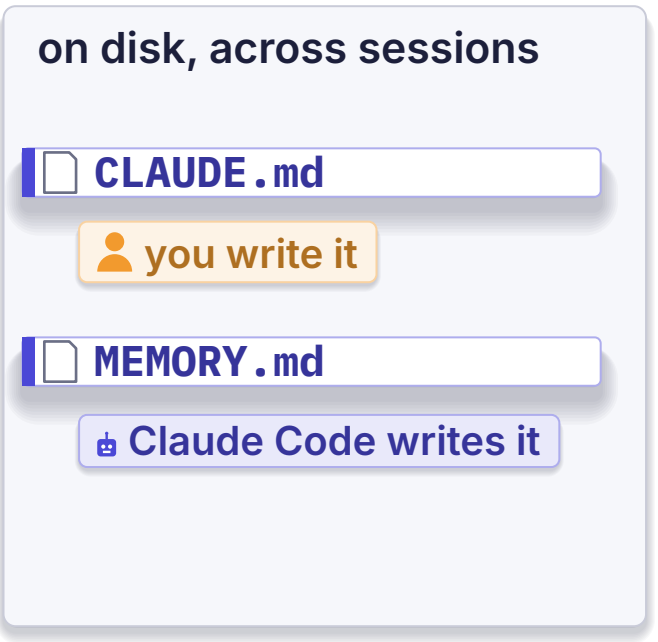
What the window holds, and what lasts



Before you type: system prompt, tools, instruction files.

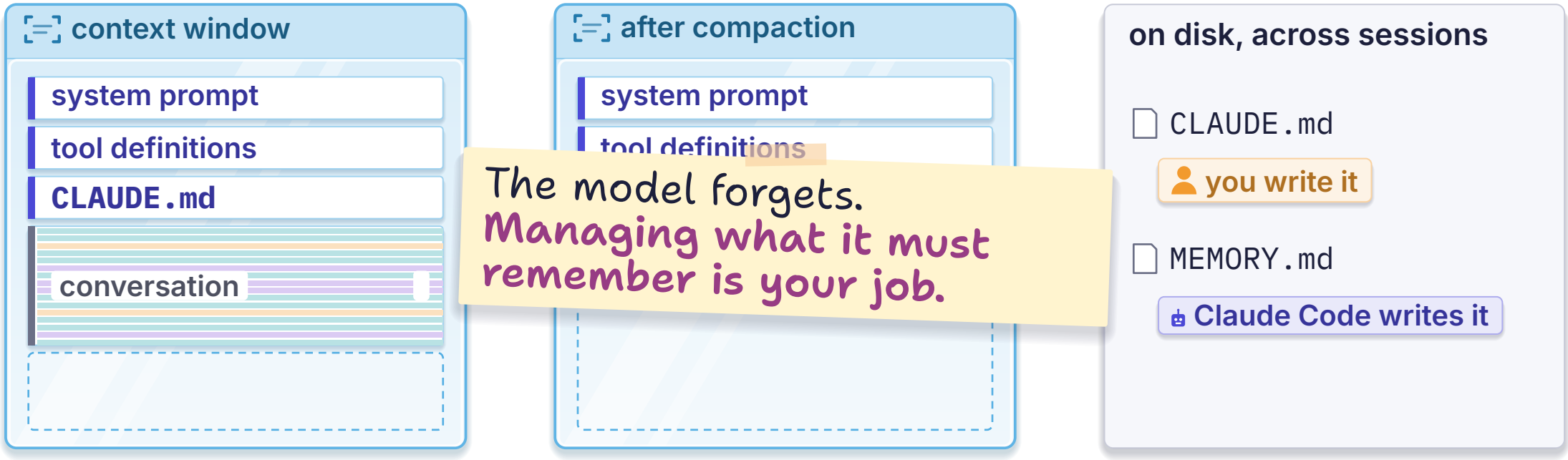


Compaction shortens the conversation. Standing rules come back from disk.



Two memories cross sessions.

What the window holds, and what lasts



Before you type: system prompt, tools, instruction files.

Compaction shortens the conversation. Standing rules come back from disk.

Two memories cross sessions.