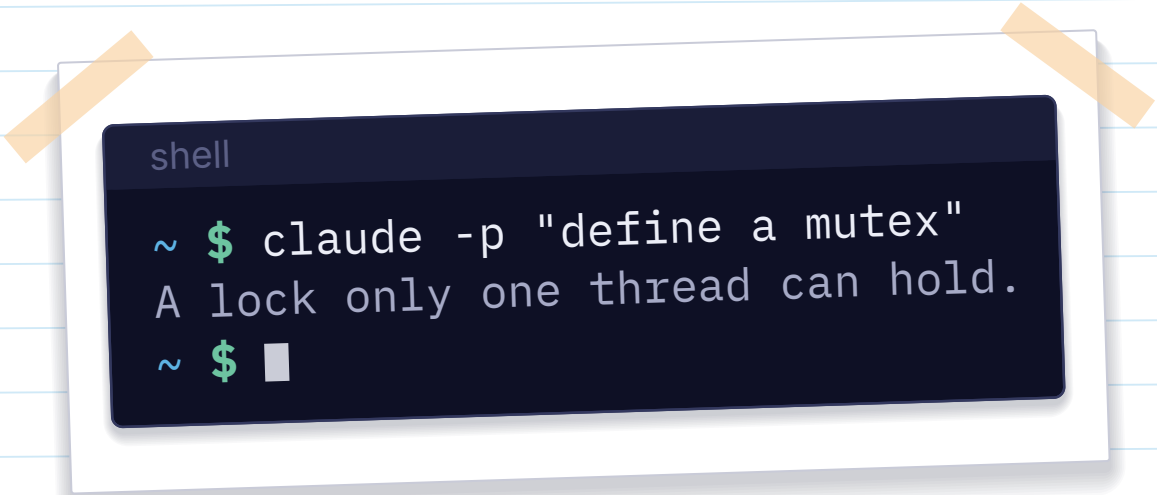
Prof. Aviv Zohar

Semester A, 2026–7

Claude Code: One Prompt from the Shell

Preparing to run and compare prompts in Lab 2



Two ways to run Claude Code

interactive session

shell

```
~ $ claude
> define a mutex
● A lock only one thread can hold.
>
```

Carried forward as context for the next turn.

Waiting for your next turn.

Two ways to run Claude Code

interactive session

shell

```
~ $ claude
> define a mutex
● A lock only one thread can hold.
>
```

one-shot mode

shell

```
~ $ claude -p "define a mutex"
A lock only one thread can hold.
~ $
```

-p: the prompt is on the command line. The answer prints, and you are back at your shell.

Two ways to run Claude Code

interactive session

```shell
~ $ claude
> define a mutex
● A lock only one thread can hold.
> and a semaphore?
● Unlike a mutex, it can let
  several threads in at once.
>
```

The earlier question is still in the context.

one-shot mode

```shell
~ $ claude -p "define a mutex"
A lock only one thread can hold.
~ $ claude -p "and a semaphore?"
A counter that limits how many
threads use a resource at once.
~ $
```

A new conversation: the mutex question is not in it.

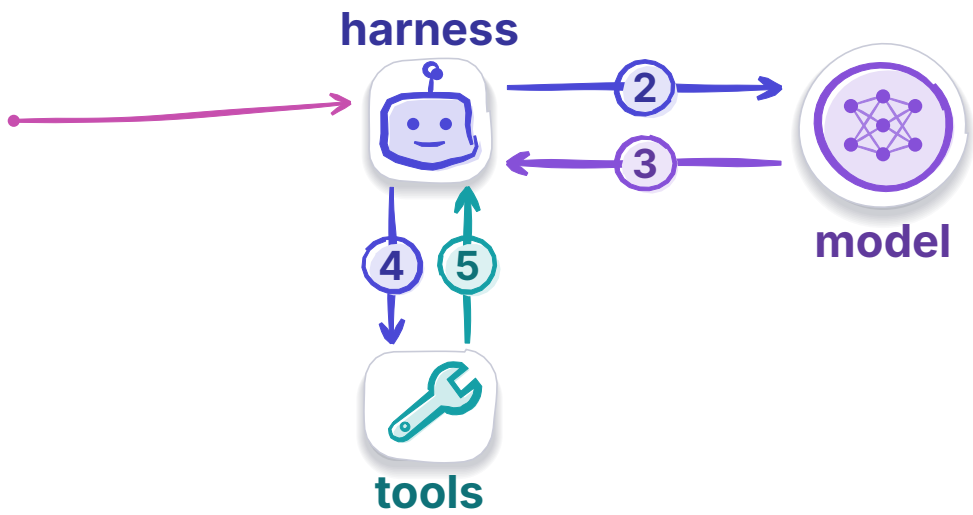
One prompt can mean several model requests

your side

```shell
~ $ claude -p "run the tests"
All 5 tests pass.
~ $
```

One prompt in, one answer out.

inside that one call



Three model requests for one prompt.

your prompt → request 1 → tool runs → request 2 → tool runs → request 3 → the answer

illustrative call

The prompt can arrive on standard input

```shell
~ $ claude -p "define a mutex"

~ $ echo "define a mutex" | claude -p
```

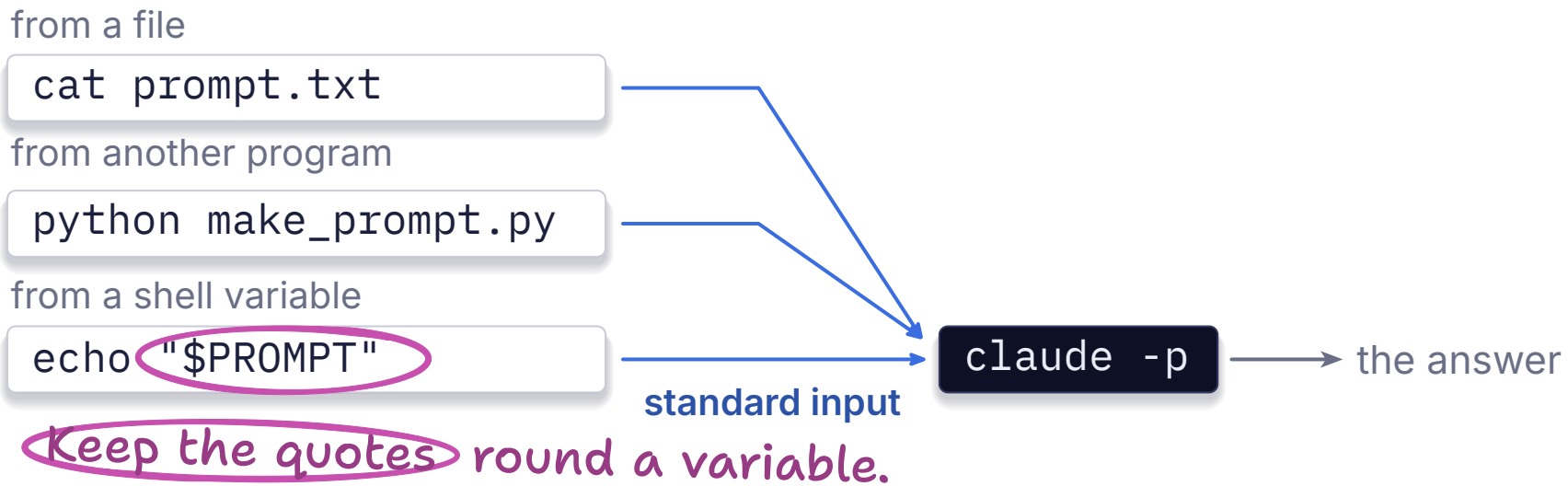
The same prompt, as a quoted argument or through the pipe. Sampling can still vary the answer.



The prompt can arrive on standard input

```shell
~ $ claude -p "define a mutex"

~ $ echo "define a mutex" | claude -p
```



A loop sends the same prompt five times

shell

```
~ $ PROMPT="define a mutex"
~ $ for i in 1 2 3 4 5; do
      echo "$PROMPT" | claude -p
    done
```

In the lab: questions from **your** problem type.

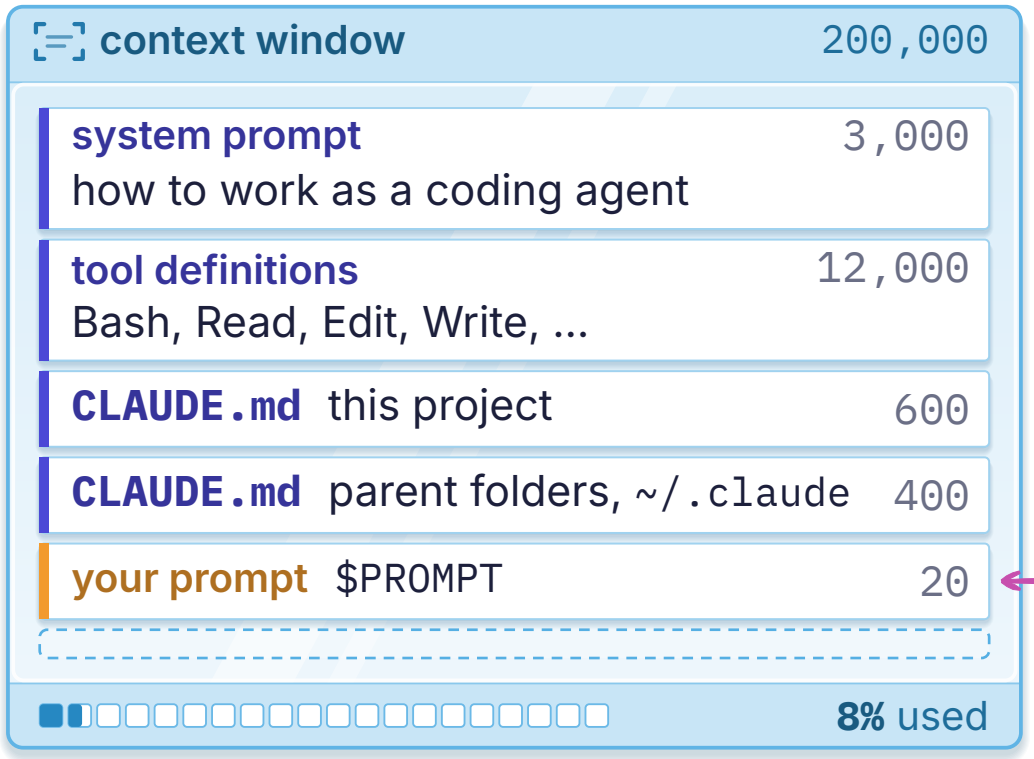
run 1	A lock only one thread can hold.	✓
run 2	A lock for mutual exclusion.	✓
run 3	Only one thread at a time holds it.	✓
run 4	A lock several threads can share.	✗
run 5	One thread holds it; others wait.	✓

4 of 5 correct

Same prompt, fresh conversations, different answers.

illustrative answers

Control what the harness adds



Added by the harness, by default: its instructions, tool definitions and instruction files.

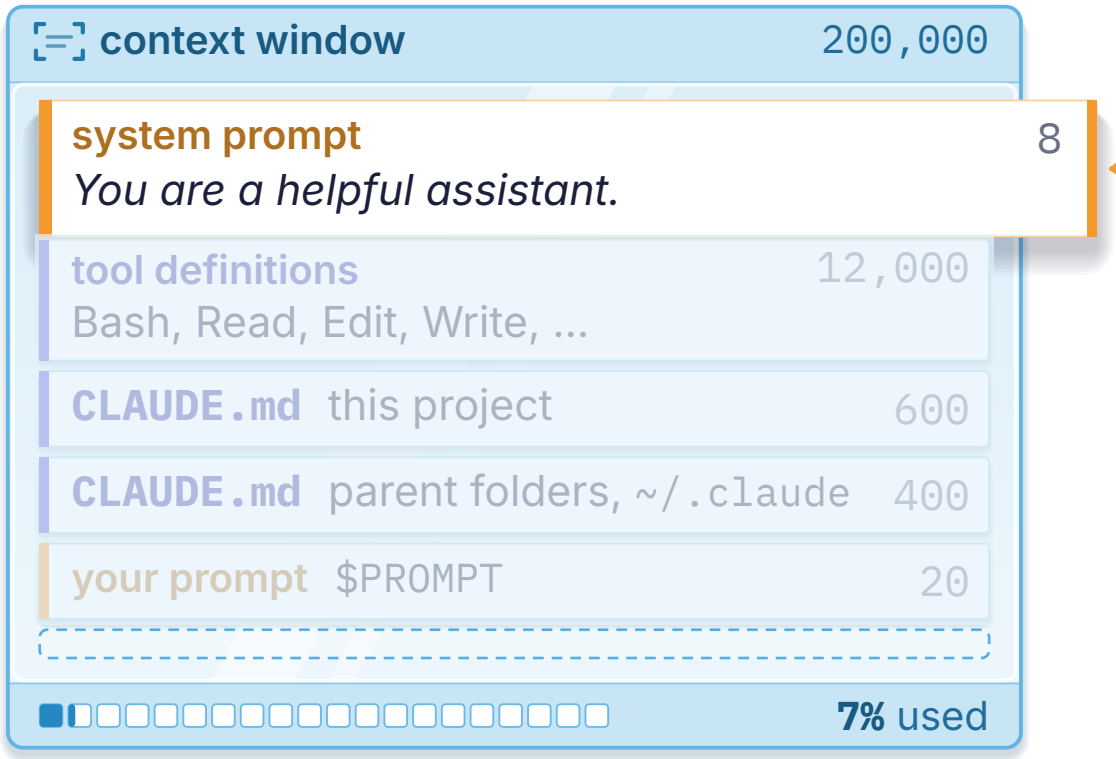
Your prompt is only part of what the model receives.

shell

```
~ $ echo "$PROMPT" | claude -p
```

illustrative sizes

Control what the harness adds



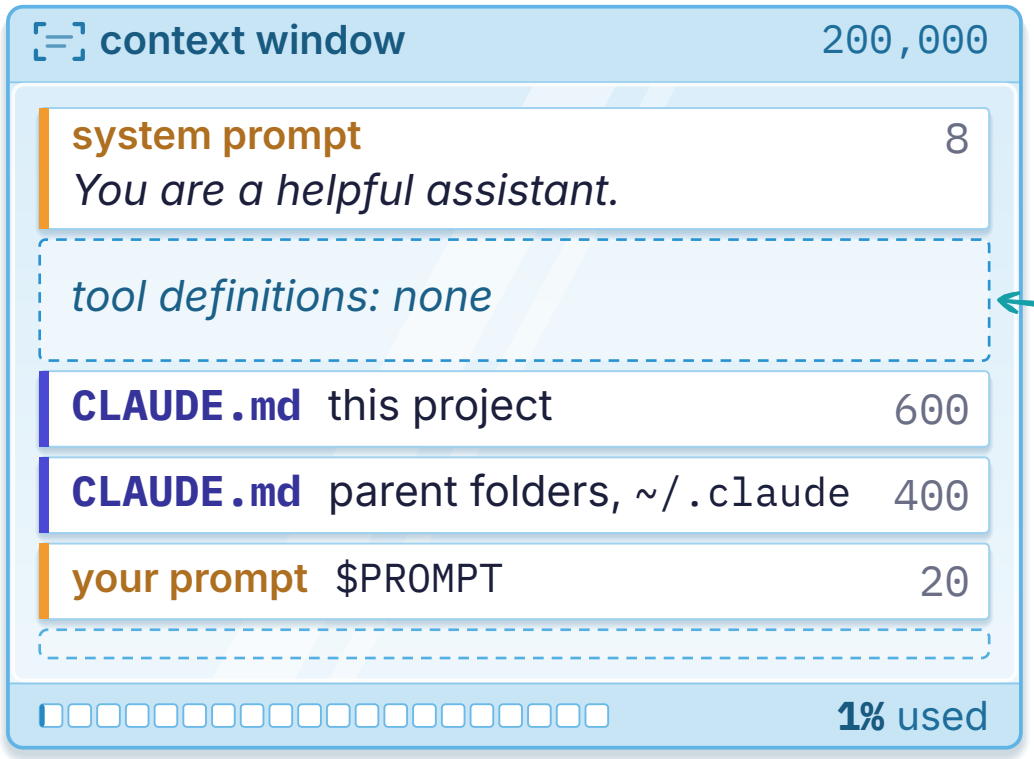
`--system-prompt` replaces the harness's instructions with yours. The trained model stays the same.

shell

```
~ $ echo "$PROMPT" | claude -p \
    --system-prompt "You are a helpful assistant."
```

illustrative sizes

Control what the harness adds



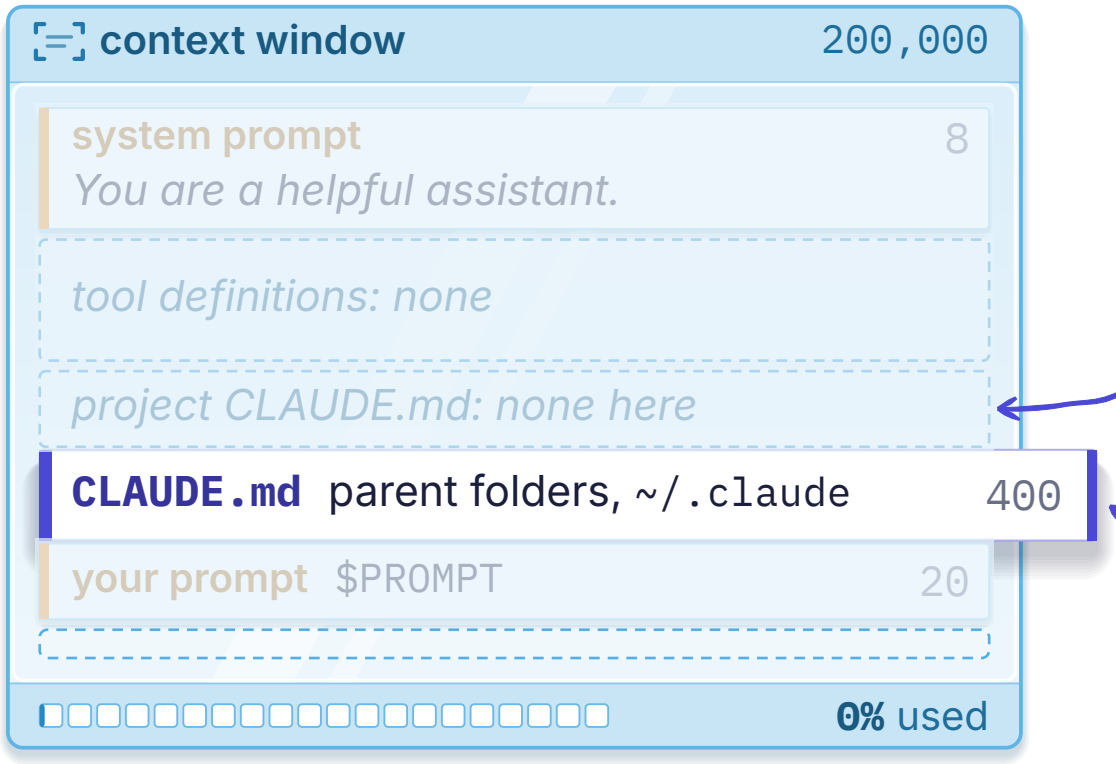
`--tools ""` disables the built-in tools; disable MCP tools too. The model answers from the context and its training.

shell

```
~ $ echo "$PROMPT" | claude -p \
    --system-prompt "You are a helpful assistant." --tools ""
```

illustrative sizes

Control what the harness adds



An empty directory: no project file.

These can still load, with saved memories. Check what yours loads, and keep it the same.

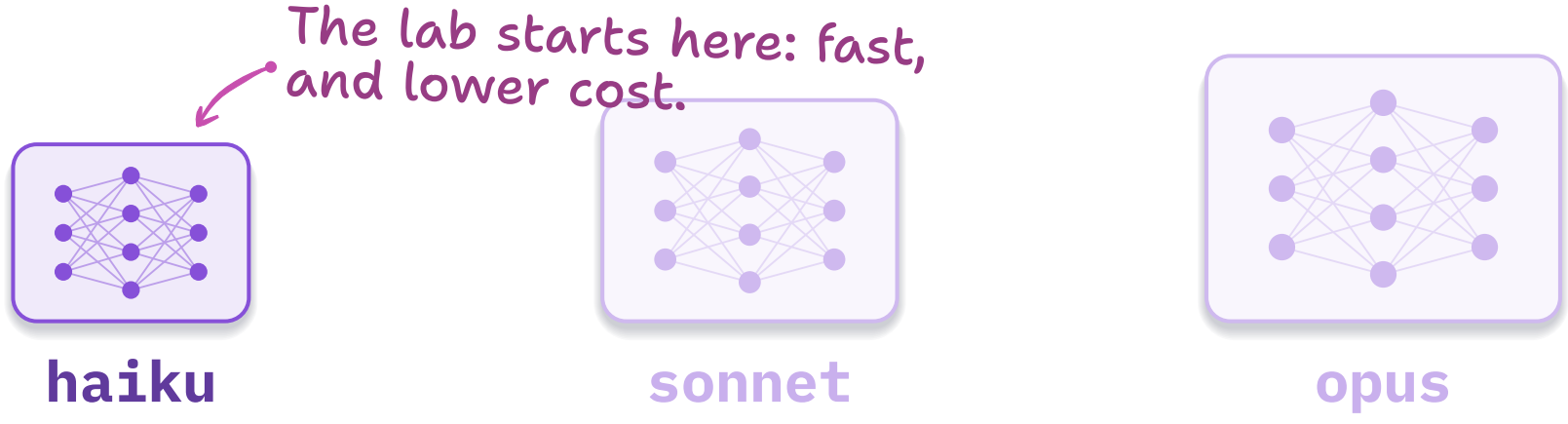
shell

```
~/empty $ echo "$PROMPT" | claude -p \
    --system-prompt "You are a helpful assistant." --tools ""
```

illustrative sizes

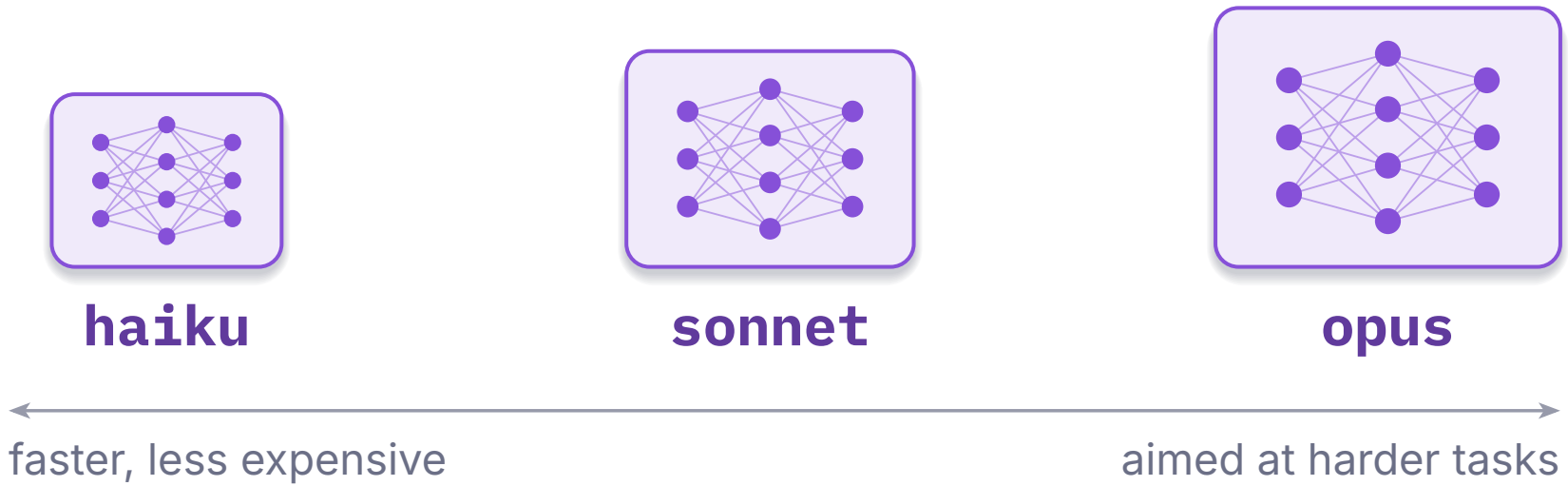
Choose the model with `--model`

```shell
~ $ echo "$PROMPT" | claude -p --model haiku
```



Choose the model with `--model`

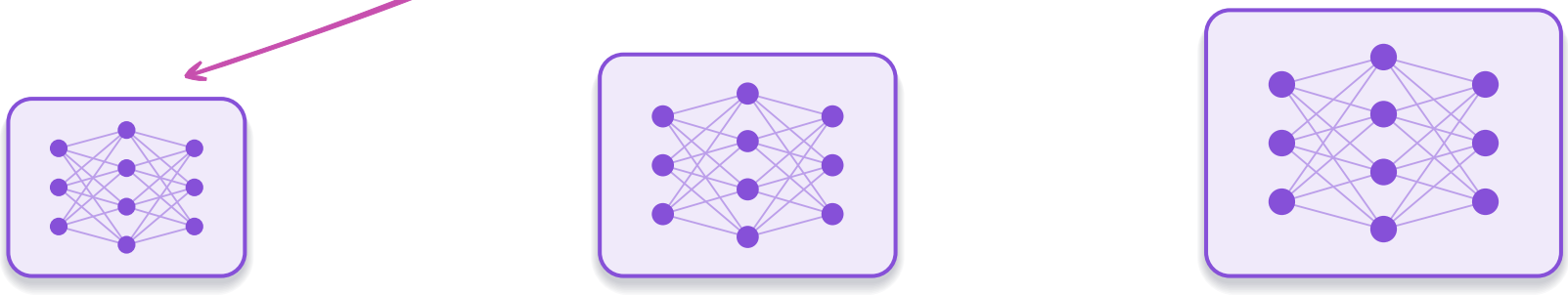
```shell
~ $ echo "$PROMPT" | claude -p --model haiku
```



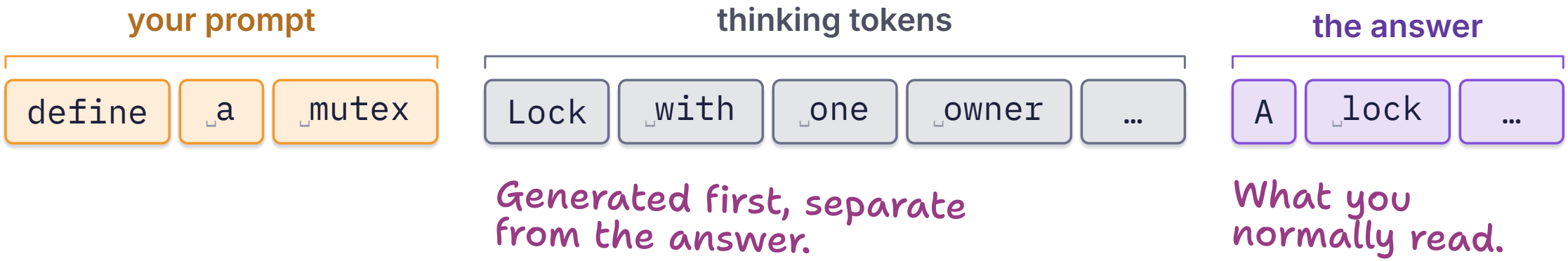
Choose the model with `--model`

```shell
~ $ echo "$PROMPT" | claude -p --model haiku
```

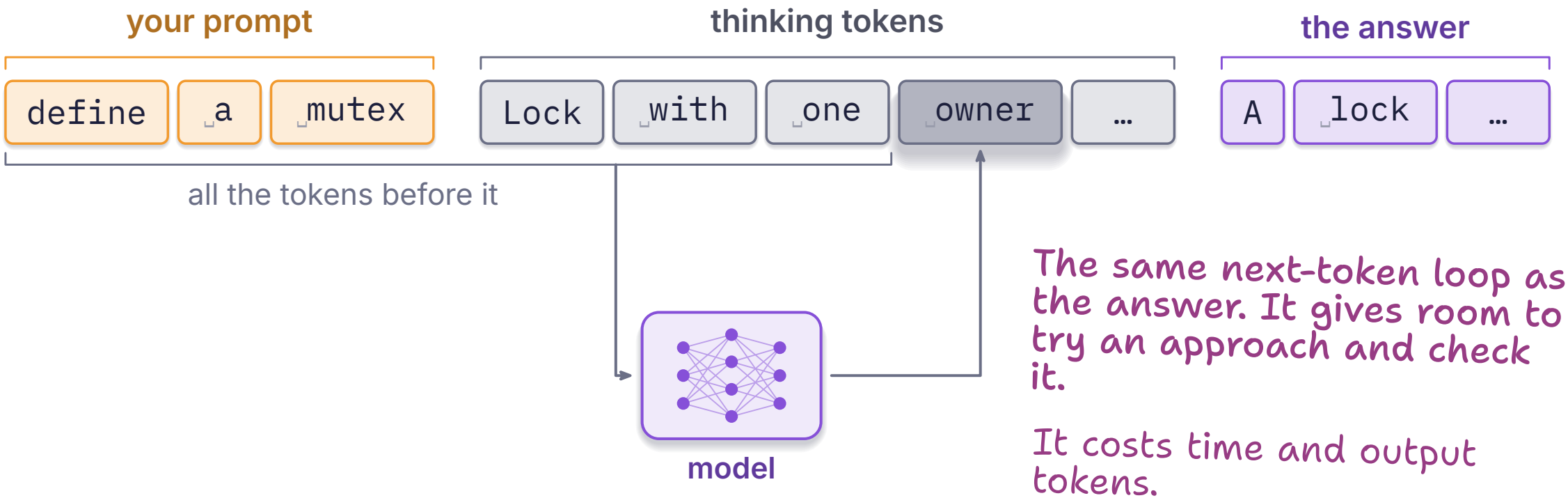
Change `haiku` to `sonnet` or `opus` and send the same prompts. Record the model version for each set of runs.



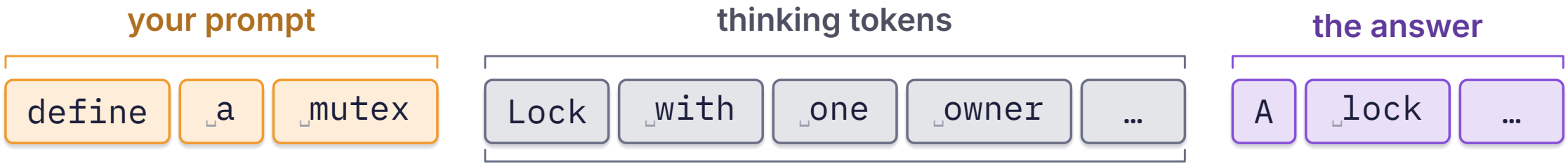
Thinking tokens come before the answer



Thinking tokens come before the answer



Thinking tokens come before the answer



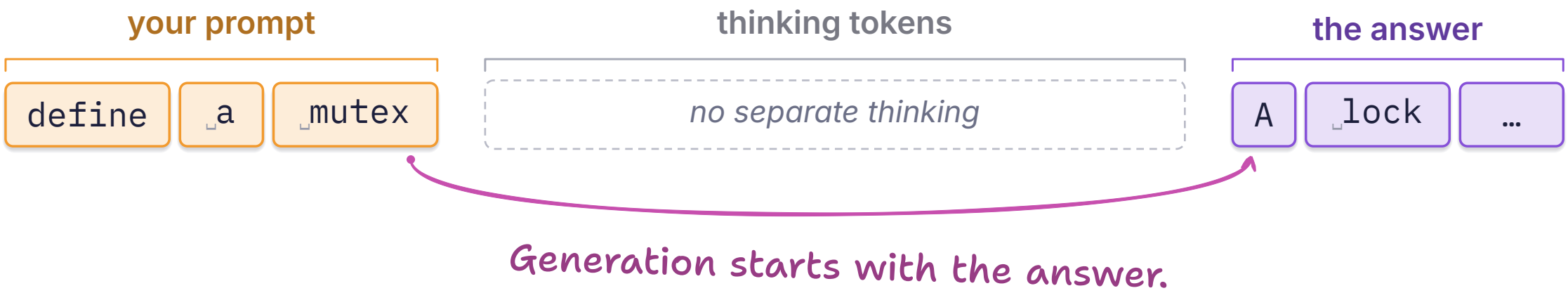
`/effort` `--effort`
how much reasoning to request

Not an exact token count.

`MAX_THINKING_TOKENS`
a fixed budget, as a number

On models with a fixed budget. The lab uses `0`.

Thinking tokens come before the answer



`/effort` `--effort`
how much reasoning to request

`MAX_THINKING_TOKENS=0`
separate thinking disabled

The answer can still show **steps**, if your prompt asks for them.

The assignment applies to the command right after it

shell

```
~ $ echo "$PROMPT" | MAX_THINKING_TOKENS=0 claude -p
```

echo

claude gets the setting

Set for `claude`, the command right after it.

The assignment applies to the command right after it

shell

```
~ $ echo "$PROMPT" | MAX_THINKING_TOKENS=0 claude -p
```

echo — **claude** gets the setting

Set for `claude`, the command right after it.

shell

```
~ $ MAX_THINKING_TOKENS=0 echo "$PROMPT" | claude -p
```

echo gets the setting — **claude** its usual setting

`claude` never receives it.

A surprising comparison? Check that the setting reached the process you meant to test.

The starting command for Lab 2

```shell
~/empty $ PROMPT="<your prompt>"
~/empty $ echo "$PROMPT" | MAX_THINKING_TOKENS=0 claude -p \
    --model haiku --tools "" \
    --system-prompt "You are a helpful assistant."
```

Set `PROMPT` to whichever prompt you need.

Run it from the empty directory you prepared.

The starting command for Lab 2

shell

```
~/empty $ PROMPT="<your prompt>"
~/empty $ echo "$PROMPT" | MAX_THINKING_TOKENS=0 claude -p \
    --model haiku --tools "" \
    --system-prompt "You are a helpful assistant."
```

change one at a time — the model | harder problems | asking for step-by-step working

keep the same — tools off | separate thinking off | the system prompt | loaded memory

To test asking for working: keep the model and the problems, and change **only** that instruction in your prompt.

The starting command for Lab 2

shell

```
~/empty $ PROMPT="<your prompt>"
~/empty $ echo "$PROMPT" | MAX_THINKING_TOKENS=0 claude -p \
    --model haiku --tools "" \
    --system-prompt "You are a helpful assistant."
```

change one at a time the model · harder problems · asking for step-by-step working

keep the same tools off · separate thinking off · the system prompt · loaded memory

one setting · five fresh runs per input

input 1	✓ ✓ ✗ ✓ ✓	**4 of 5 correct**
input 2	✓ ✗ ✓ ✓ ✗	**3 of 5 correct**
input 3	✓ ✓ ✓ ✓ ✓	**5 of 5 correct**

Build the test program with your agent. Decide how to check answers **first**, then inspect a few responses to confirm the scoring.

illustrative runs

The same command with Codex

```shell
~/empty $ echo "$PROMPT" | codex exec --ephemeral --skip-git-repo-check \
    -s read-only -m gpt-5.6-sol -c model_reasoning_effort="none" \
    -c model_instructions_file=$HOME/sys.md -c web_search="disabled" \
    --disable shell_tool --disable unified_exec --disable view_image \
    --disable multi_agent --disable apps --disable plugins \
    --disable skill_search --disable tool_suggest --disable sleep_tool \
    --disable browser_use --disable computer_use --disable image_generation
```

`codex exec` one-shot mode

`-m` selects the model

`"none"` separate thinking off

`model_instructions_file` your instructions

`--disable` turns off the tools

No need to memorize the switches: the handout has the command, and **what each one does**.

The same command with Codex

shell

```
~/empty $ echo "$PROMPT" | codex exec --ephemeral --skip-git-repo-check \
    -s read-only -m gpt-5.6-sol -c model_reasoning_effort="none" \
    -c model_instructions_file=$HOME/sys.md -c web_search="disabled" \
    --disable shell_tool --disable unified_exec --disable view_image \
    --disable multi_agent --disable apps --disable plugins \
    --disable skill_search --disable tool_suggest --disable sleep_tool \
    --disable browser_use --disable computer_use --disable image_generation
```

models for `-m`

`gpt-5.6-sol` `gpt-5.6-terra` `gpt-5.6-luna` `gpt-6-astra`

`gpt-6-astra` cannot turn thinking off. Its lowest effort is `low`, so it always thinks a little. If you test it, say so in your report.