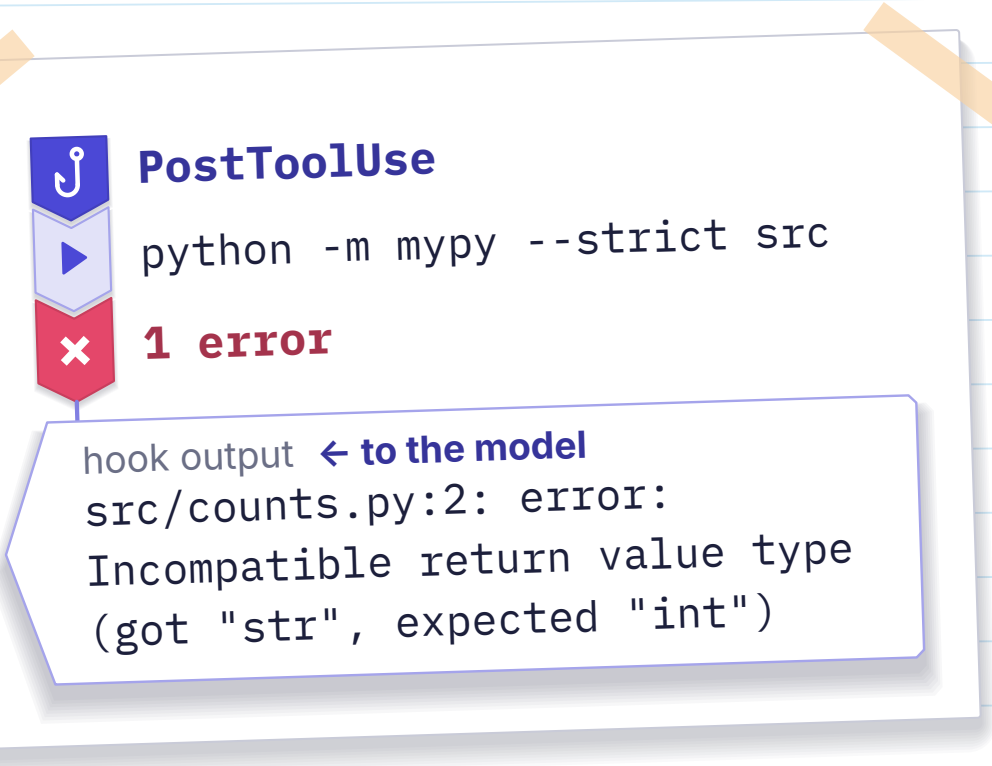
Prof. Aviv Zohar

Semester A, 2026–7

Why Hooks?

Automatic checks

The same check, started at the right event



The type checker compares the code with its declaration

src/counts.py

```python
def total_items(counts: list[int]) -> int:
    return sum(counts)
```

a list of integers in, an integer out

shell

```
~/demo $ python -m mypy --strict src
```

The project's own check, ready to run

The type checker compares the code with its declaration

src/counts.py

```python
def total_items(counts: list[int]) -> int:
    return str(sum(counts))
```

The edit returns text

shell

```
~/demo $ python -m mypy --strict src
src/counts.py:2: error: Incompatible
return value type (got "str", expected "int")
Found 1 error in 1 file (checked 1 source file)
```

where: line 2

what disagrees: str against int

How does this error get back to the agent?

By hand, you start the check and carry its error

agent session ~/demo

tool call asked by the model

```
Edit src/counts.py
+    return str(sum(counts))
```

you pasted by you

```
src/counts.py:2: error:
Incompatible return value type
(got "str", expected "int") …
```

shell

```
~/demo $ python -m mypy --strict src
src/counts.py:2: error:
Incompatible return value type
(got "str", expected "int")
```

① ②

The checker does the analysis.

Your part: ① start it,
② carry the result.

The agent runs the checker itself
only if it remembers to.

By hand, you start the check and carry its error

agent session ~/demo

tool call asked by the model

```
Edit src/counts.py
+    return str(sum(counts))
```

you pasted by you

```
src/counts.py:2: error:
Incompatible return value type
(got "str", expected "int") …
```

shell

```
~/demo $ python -m mypy --strict src
src/counts.py:2: error:
Incompatible return value type
(got "str", expected "int")
```

Automate the steps. Keep the same checker and the same useful error.



A hook starts the same check, after every edit

agent session ~/demo

tool call asked by the model

```
Edit src/counts.py
+    return str(sum(counts))
```

hook after the edit · cropped

```
src/counts.py:2: error:
Incompatible return value type
(got "str", expected "int") …
```

The same error you would have pasted,

and you did nothing.

inside the harness

PostToolUse

```
python -m mypy --strict src
```

1 error

hook output ← **to the model**

```
src/counts.py:2: error:
Incompatible return value type
(got "str", expected "int") …
```

the hook's output on the example project, cropped

A hook starts the same check, after every edit

agent session ~/demo

tool call asked by the model

```
Edit src/counts.py
+    return str(sum(counts))
```

hook after the edit · cropped

```
src/counts.py:2: error:
Incompatible return value type
(got "str", expected "int") …
```

inside the harness

PostToolUse ⓑ

```
python -m mypy --strict src
```
ⓐ

1 error

hook output ← **to the model**

```
src/counts.py:2: error:
Incompatible return value type
(got "str", expected "int") …
```

A hook: ⓐ code registered to run ⓑ when an event occurs.

the hook's output on the example project, cropped

A hook starts the same check, after every edit

agent session ~/demo

tool call asked by the model

```
Edit src/counts.py
+    return str(sum(counts))
```

hook after the edit · cropped

```
src/counts.py:2: error:
Incompatible return value type
(got "str", expected "int") …
```

The harness : the software around the model. It starts the hook at a set point: after the edit.

inside the harness

PostToolUse

```
python -m mypy --strict src
```

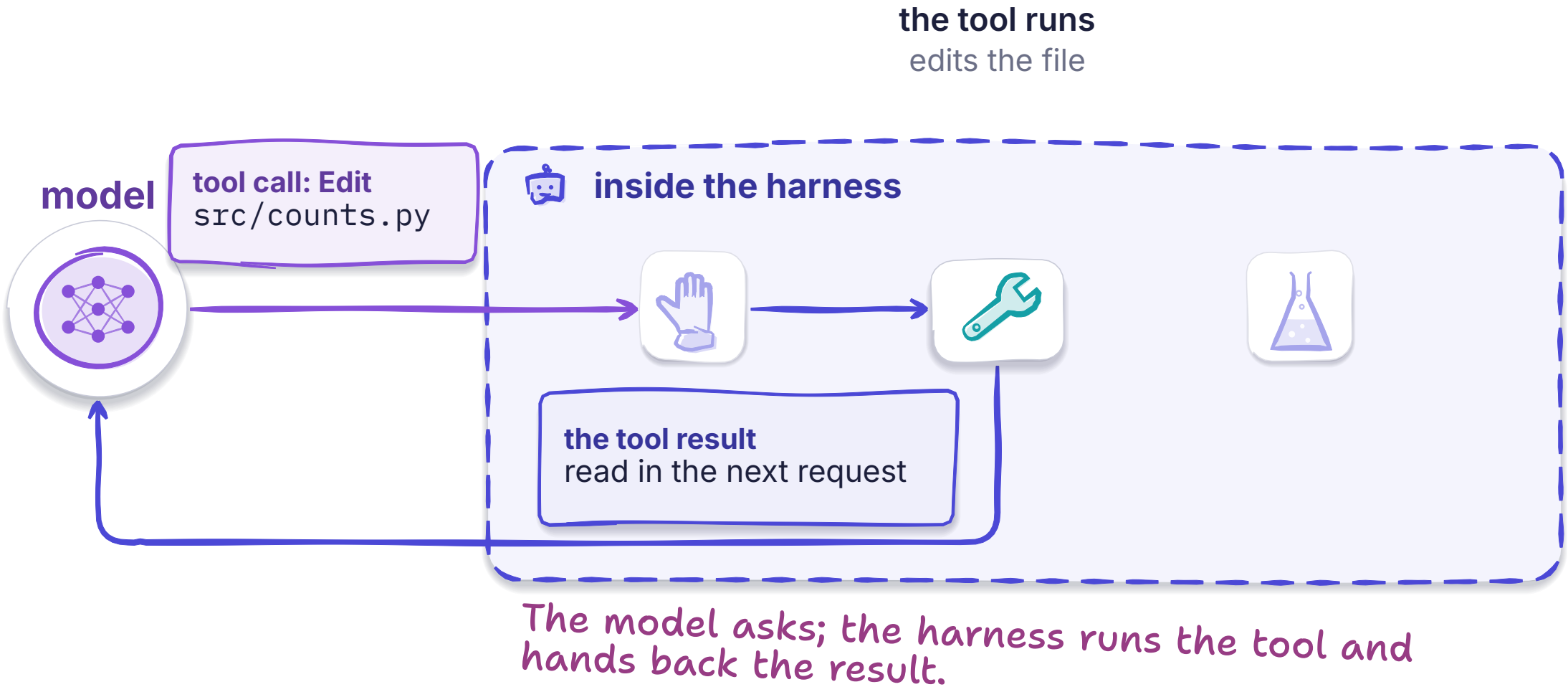
1 error

hook output ← **to the model**

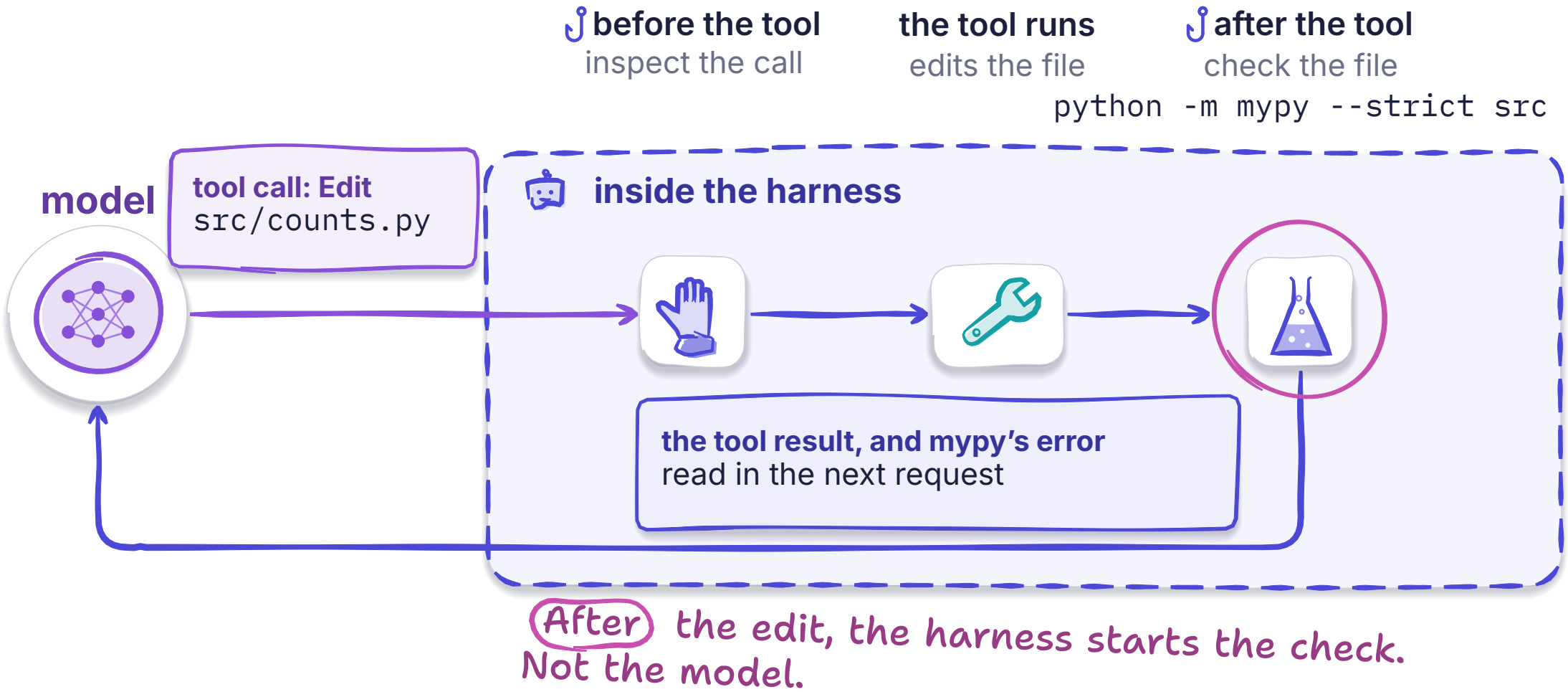
```
src/counts.py:2: error:
Incompatible return value type
(got "str", expected "int") …
```

the hook's output on the example project, cropped

Hooks run inside the harness, around the tool

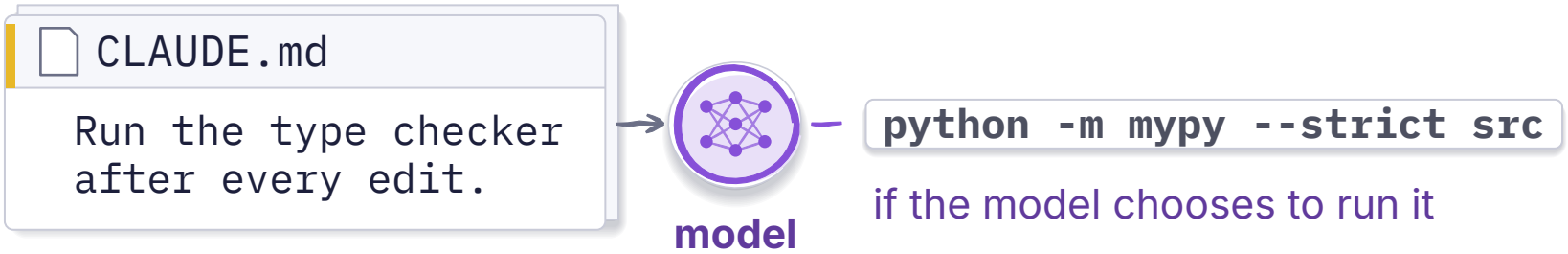


Hooks run inside the harness, around the tool

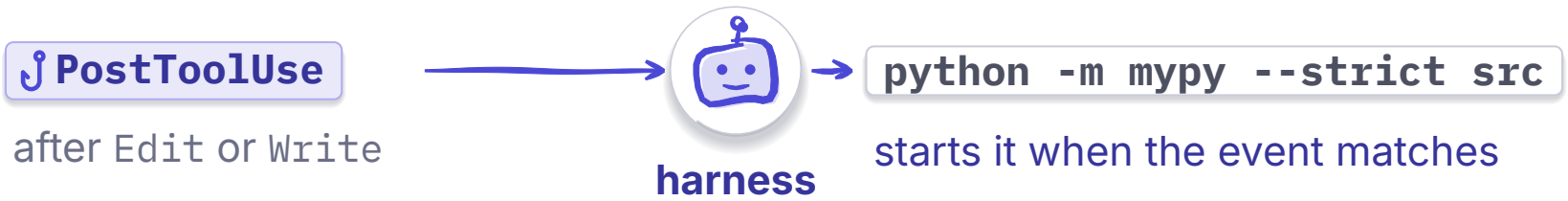


An instruction asks the model; the harness starts a hook

An instruction read by the model with the rest of its context

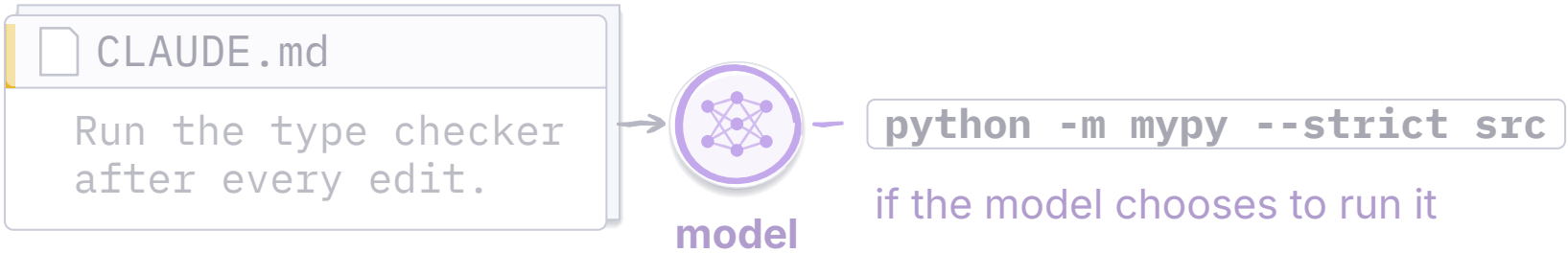


A command hook configured in the settings



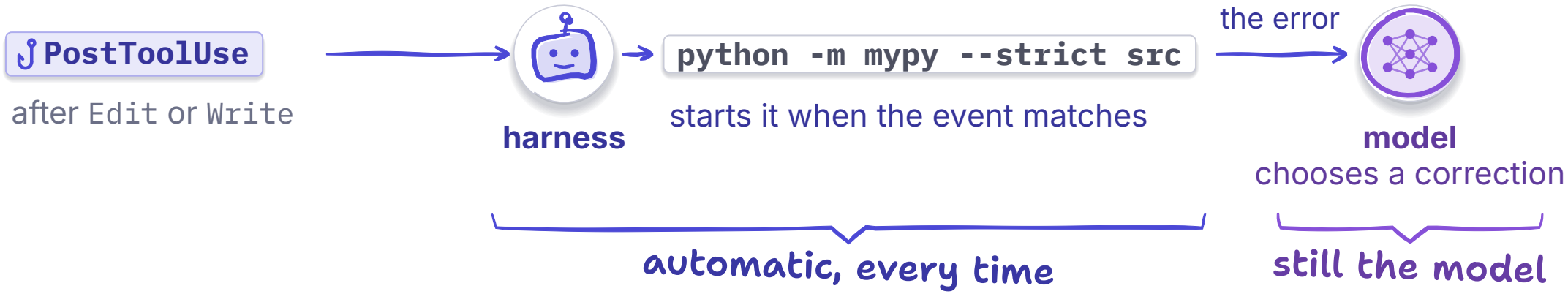
An instruction asks the model; the harness starts a hook

An instruction read by the model with the rest of its context

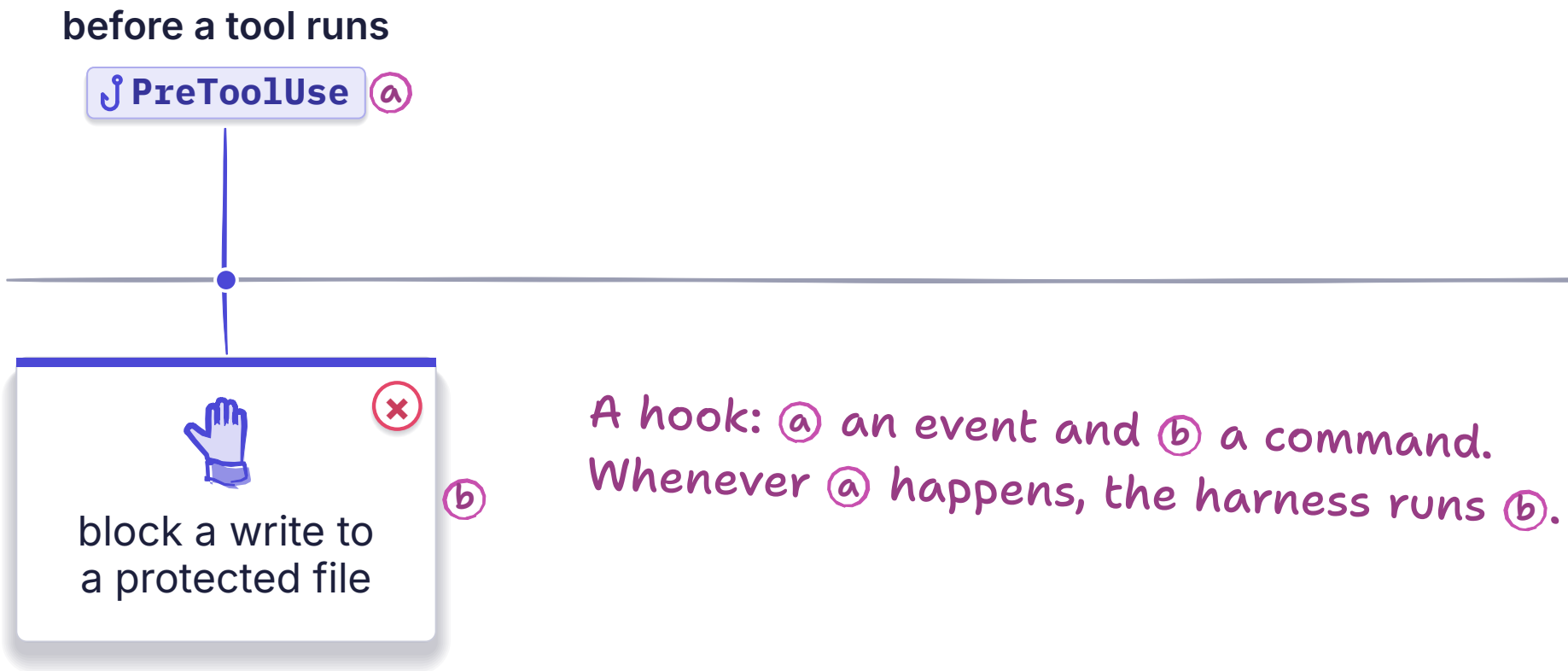


Not only when someone remembers to run it.

A command hook configured in the settings



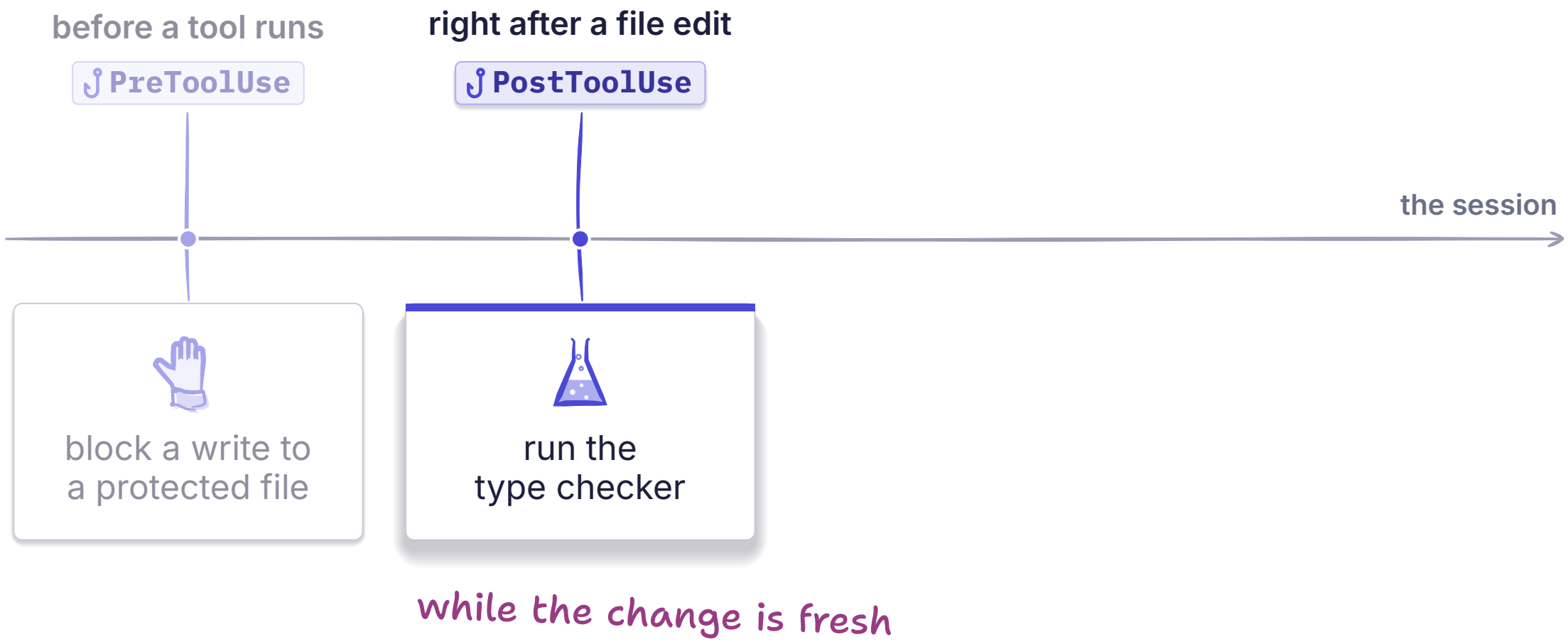
Pick the event by when the command is useful



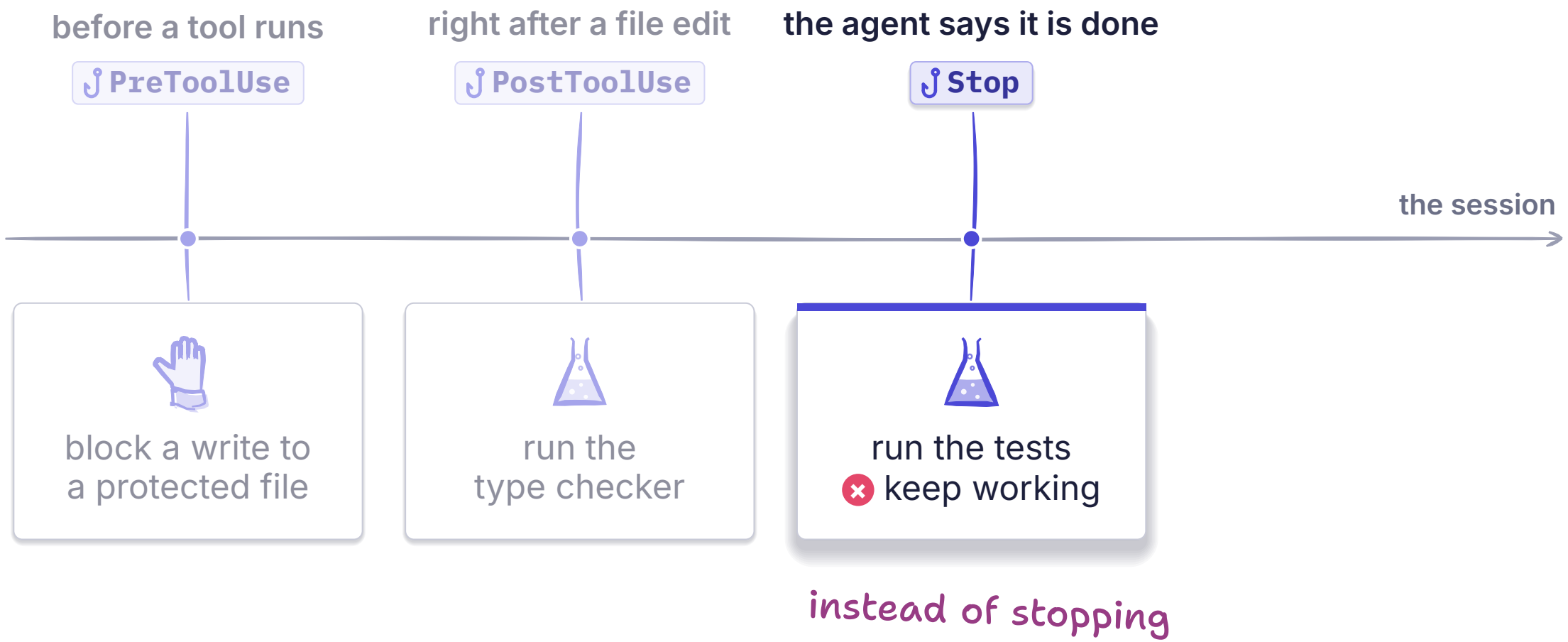
A hook: ⓐ an event and ⓑ a command.
Whenever ⓐ happens, the harness runs ⓑ.

Before the write: a check after it cannot undo it.

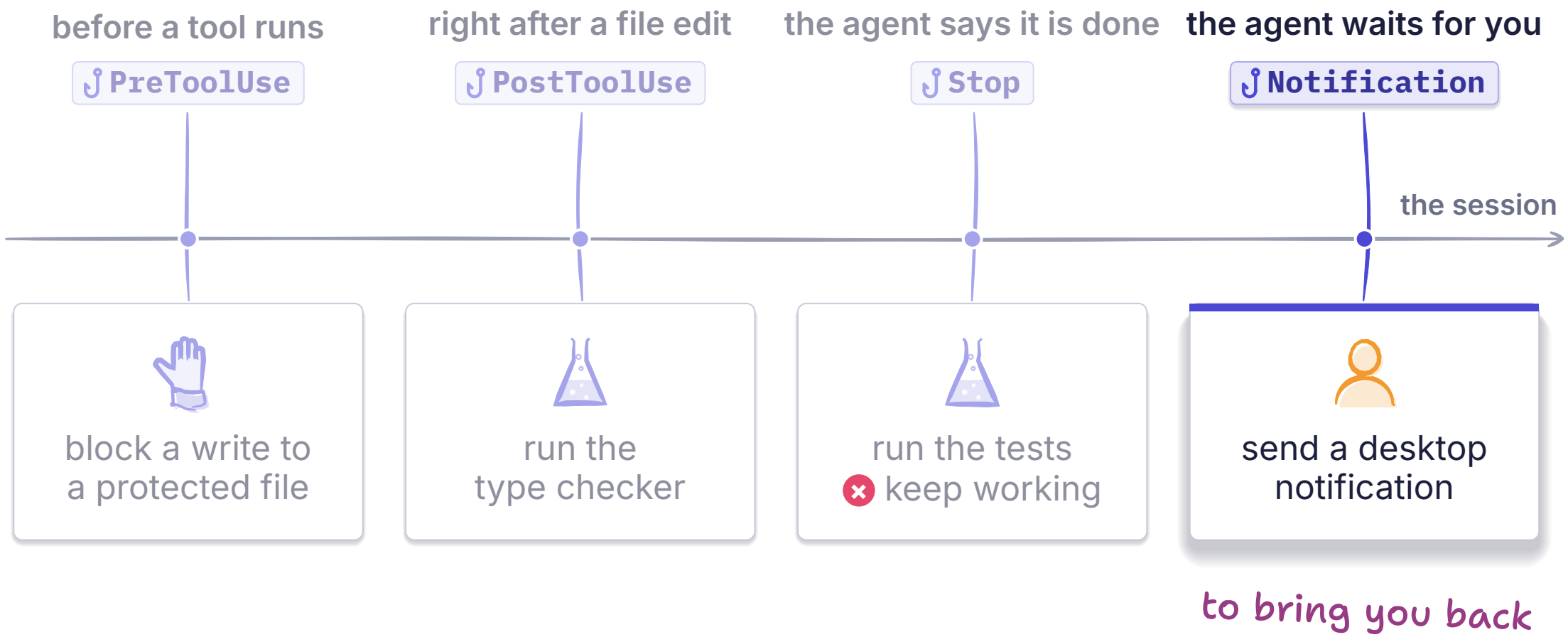
Pick the event by when the command is useful



Pick the event by when the command is useful



Pick the event by when the command is useful



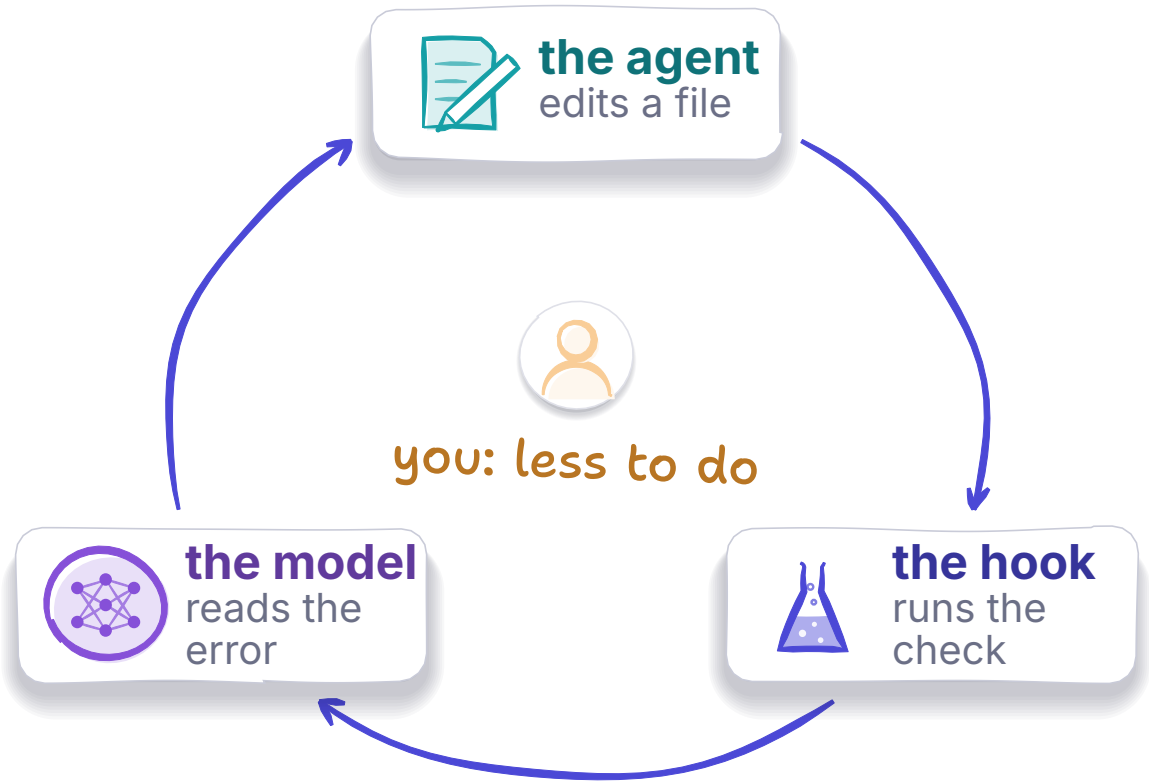
Claude Code offers many more events

the session	`SessionStart` `Setup` `InstructionsLoaded` `SessionEnd`
your prompts	`UserPromptSubmit` `UserPromptExpansion`
tools, permissions	**`PreToolUse`** `PermissionRequest` `PermissionDenied` **`PostToolUse`** `PostToolUseFailure` `PostToolBatch`
asking you	**`Notification`** `Elicitation` `ElicitationResult`
subagents, tasks	`SubagentStart` `SubagentStop` `TaskCreated` `TaskCompleted` `TeammateIdle`
compaction	`PreCompact` `PostCompact`
stopping	**`Stop`** `StopFailure`
everything else	`PreModelSwitch` `PostModelSwitch` `ConfigChange` `CwdChanged` `FileChanged` `DirectoryAdded` `WorktreeCreate` `WorktreeRemove` `MessageDisplay`

No need to memorize them: ask the agent which event fits.

the hook events in Claude Code 2.1.285, September 2026

Start with a command you already use



1. A command you use and understand
   `python -m mypy --strict src`
2. Attach it to the event where it helps
   `PostToolUse`
3. See whether you step in less

The check covers only what it was designed to check. The hook just brings its result to the model, without you.

To add a hook, ask the agent

you typed by you

Add a hook that runs mypy after every file edit.

you typed by you

Which hook events can I use, and when does each one run?

.claude/settings.json

```
{
  "hooks": {
    "PostToolUse": [{
      "matcher": "Edit|Write",
      "hooks": [{
        "type": "command",
        "command": "\"$CLAUDE_P
      }]
    }]
  }
}
```

Know what a hook can do for you.
The agent handles the syntax.

The syntax: the agent writes it.

See you in the next video.