Repository memory

Key takeaways

1. A session’s knowledge dies with the session unless a file keeps it.
2. Loaded memory taxes every future session; retrieved memory costs nothing.
3. A note that outlives its cause misdirects work instead of guiding it.

Common misunderstandings of project memory

Which claim would you defend right now? Choose before the evidence.

"An agent that remembers a lesson on its own makes the project safer."

"Anything worth knowing belongs in AGENTS.md, since that file always loads."

"A note that was correct when written cannot hurt anything by staying."

Keep your choice. Look for the mechanism or counterexample that would make it fail.

The session is the unit of forgetting

The next session starts from zero and rediscovers the same fact — usually the same wrong way

The same repository, two days apart

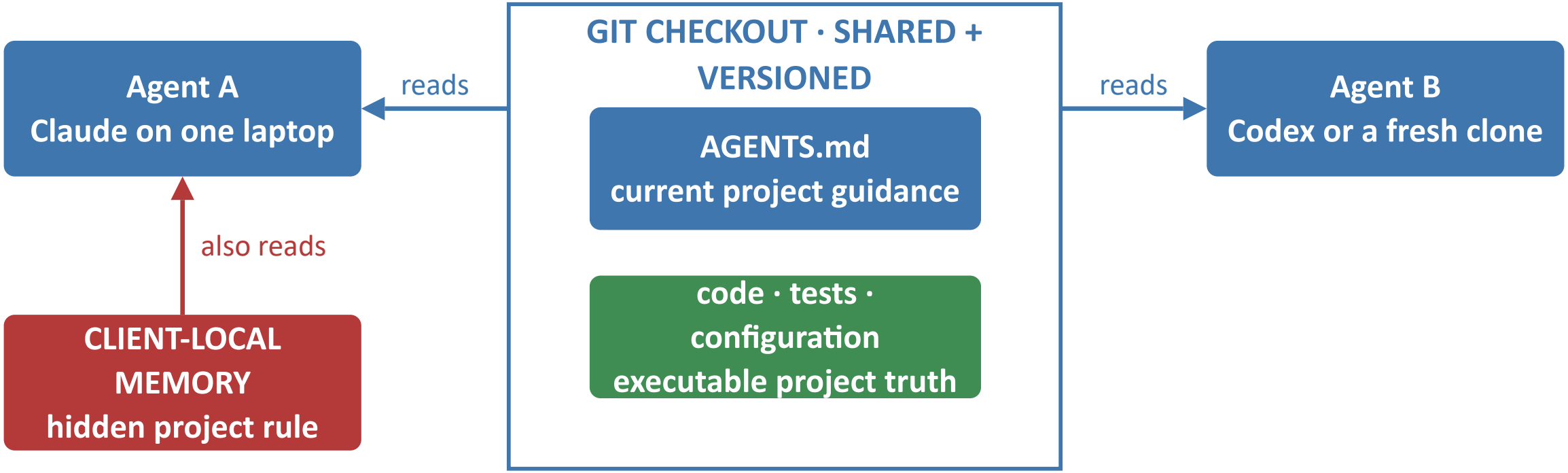
```
# Monday — session 1
you:    the Windows students cannot run `make kit`
agent:  COURSE_ROOT := $(shell git rev-parse --show-toplevel)
you:    that IS the bug — MSYS2 git prints /c/..., which Windows
        Python resolves to the nonexistent C:\c\...
        ... 40 minutes, one wrong repair, one revert

# Wednesday — session 2, same repository, empty context
you:    add a kit target to the new assignment
agent:  COURSE_ROOT := $(shell git rev-parse --show-toplevel)
```

Nothing you explained in the chat became a property of the project.

The agent offers to remember it for you

Client-local memory outlives your session and nobody else's



Personal preferences may be private. Project facts must travel with the checkout.

source: dotfiles template: AGENTS.md Configuration ownership

Four destinations, four different costs

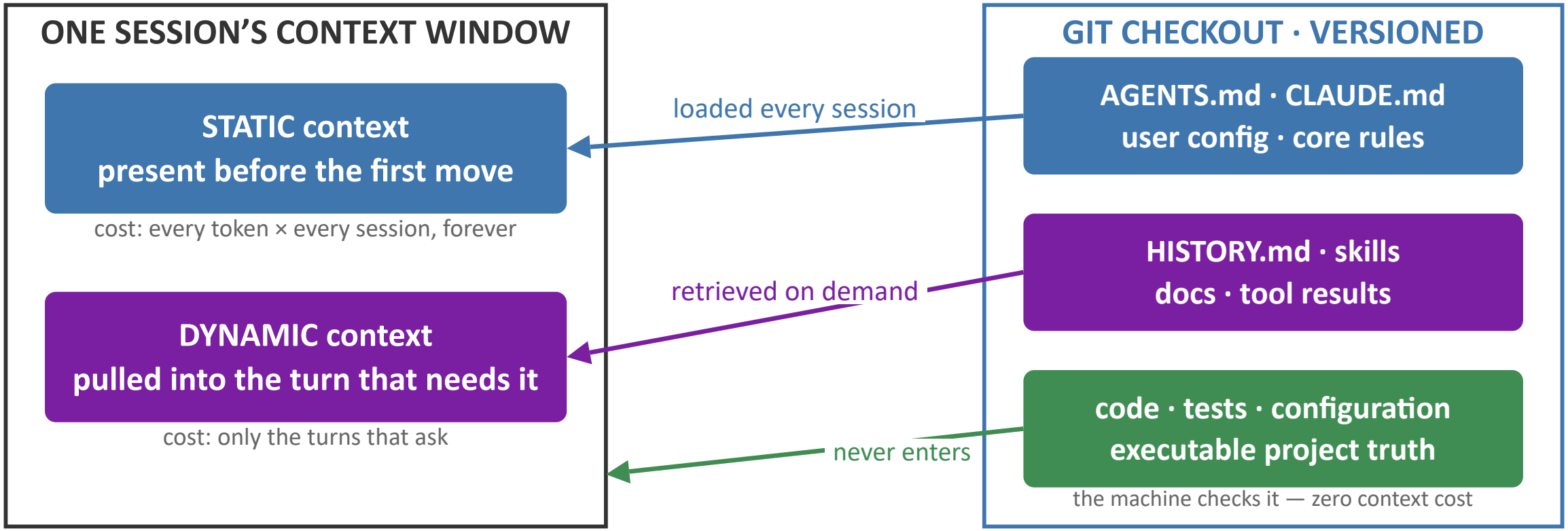
Persisting a fact is not free: the destination decides who pays, and how often

Destination	What it costs	Right when
code · test · config	written once; the machine checks it	the repository can detect the violation
AGENTS.md — loaded	every session, every task, forever	a session must know it before its first move
HISTORY.md — retrieved	nothing until someone greps it	the question is why, not what
issue · plan · handoff	expires with the task	status and next step, not project law

Loaded memory is a tax on every future session; that is why AGENTS.md stays short and HISTORY.md may grow forever.

Static vs. dynamic context

Google's paper names two of the four destinations; it omits the one the machine enforces



What always loads vs. what is retrieved is architecture: review the boundary like code.

source: adapted from Osmani, Saboo & Kartakis, The New SDLC With Vibe Coding (Google, May 2026), Fig. 4; the enforced column is this course's addition

Enforced: let the repository reject it

The cheapest memory is the one nobody has to read

Move a checkable invariant into a test

```
# Prose-only memory
# "Do not regenerate schedule.html by hand."

def test_schedule_is_regenerated() -> None:
    expected = generate_schedule(LECTURES)
    assert SCHEDULE_HTML.read_text() == expected

# The repository now detects stale generated output itself.
```

Use AGENTS.md to explain the workflow; use code and tests to reject violations.

source: utac web/AGENTS.md and web/test_gen_schedule.py

Loaded: AGENTS.md is the operating manual

Every line here is re-read by every session, so every line has to earn it

Put here	Leave out
where changes belong and which file owns policy	a tour of every module
non-obvious invariants and active landmines	resolved bugs and old workarounds
required commands, ordering, and environment assumptions	full build logs or troubleshooting transcripts
scoped rules near the code they govern	client-specific private copies of project facts

A landmine is an active trap with a consequence, not a historical anecdote.

Name the trigger, action, and consequence

Monday’s forty minutes, written down so Wednesday’s session never spends them again

A real operative note from this repository

```
## Repository-wide landmines

- Never let `git rev-parse --show-toplevel` reach a Python
  interpreter. MSYS2 git prints `/c/...`, which Windows Python
  resolves to the nonexistent `C:\c\...`.
  - In Python: walk up to `.git`, testing `.exists()` — in a
    linked worktree `.git` is a file, not a directory.
  - In make: keep the root relative. Pattern: hw/pdf2md/Makefile.

# Trigger:     computing a repository root
# Action:      walk up, and keep it relative in make
# Consequence: `uv run /c/...` fails for every Windows student
```

If the note does not change the next command, it is spending context without managing risk.

source: utac/AGENTS.md: Repository-wide landmines

Retrieved: HISTORY.md keeps the reason

Dated, append-only, never loaded — written so nobody tidies the fix away and restores the bug

The entry behind the previous slide's landmine

```
> Agents: this is the why-archive — append-only, newest at the
> bottom. The operative state is in AGENTS.md, already loaded.
> Grep this for the rationale behind a specific decision.

## 2026-08-12 — Repo-root discovery in make needs a relative path

The obvious repair does not work, which is the part worth
recording. A make-level walk-up to `.git` still produces `/c/...`
under MSYS2 make, so an absolute course root reintroduces the bug
it was meant to remove. What fixes it is keeping the root
relative: a relative path carries no drive prefix to mistranslate.
```

History is a good archive and a bad operating manual.

source: utac/HISTORY.md, 2026-08-12

Expiring: task state dies with the task

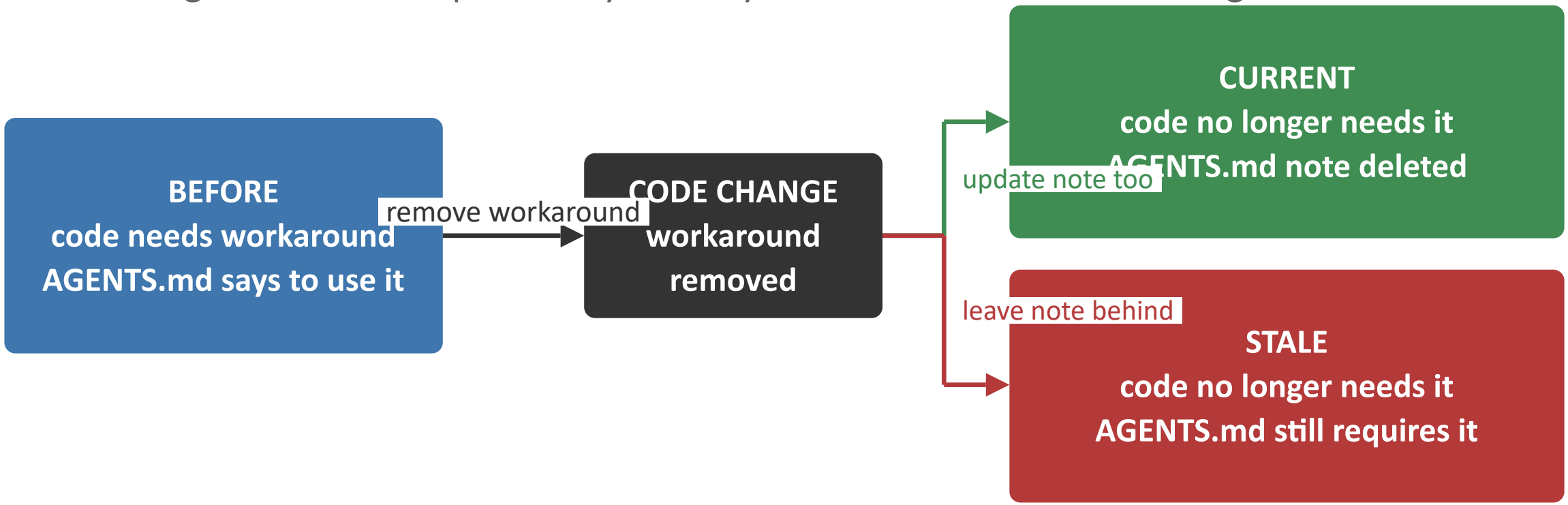
A handoff resumes unfinished work; it should not become permanent project law

Temporary handoff	Durable repository memory
goal and current status	stable architecture or ownership boundary
next action and open question	active invariant or recurring trap
commands already run and their results	test or command that future work must run
delete or close when the task lands	review and maintain with the code

Promote only the lesson that should change future work; let the rest die with the task.

Loaded memory must change with the code

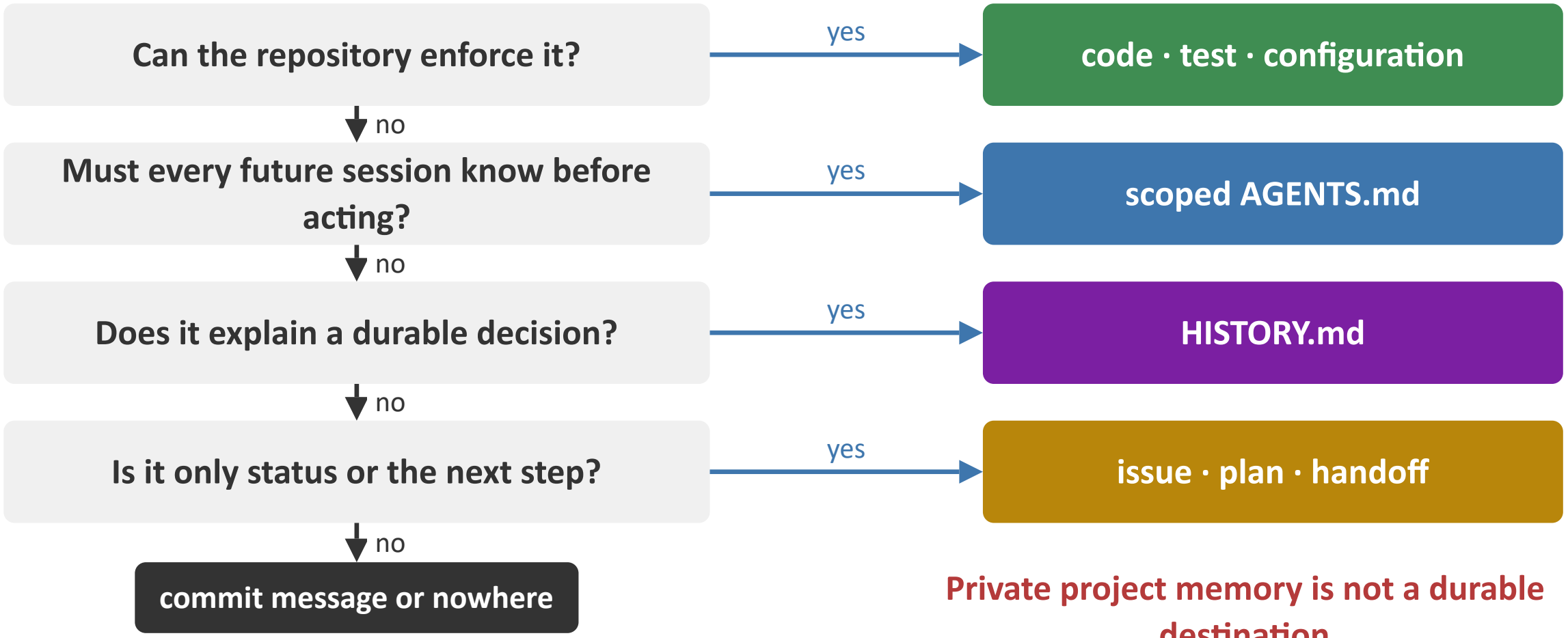
Same-change maintenance prevents yesterday's workaround from becoming tomorrow's rule



A stale loaded note misdirects every future session; delete or update it with the code change.

Route the next fact you learn

Ask in order; the first yes owns the fact, and most facts never reach an answer



Private project memory is not a durable destination.

You vs. the Agent: repository memory

Your Responsibility	Agent's Responsibility
Choose what deserves persistent attention.	Propose the narrowest durable destination.
Review instructions that affect future work.	Verify named files, commands, and assumptions.
Resolve conflicts between policy, evidence, and intent.	Update or remove invalidated guidance with the code.

Shared: Prefer executable truth; keep the remaining prose current and scoped.

Key Takeaways

1. What a session learns is lost unless it lands in a file that ships with the checkout.
2. Route by cost: enforce it in code, load it from AGENTS.md, retrieve it from HISTORY.md, or let it expire.
3. Writing a note incurs a maintenance debt: delete or update it in the same change that invalidates it.

route what your project already knows

10 min

- Route four facts: (1) run `pytest -x tests/test_search.py`; (2) duplicate Message-IDs must stay searchable; (3) FTS5 was rejected because the assignment is the inverted index; (4) next reproduce one failing sender-filter query.
- Choose for each: code/test, AGENTS.md, HISTORY.md, handoff, or nowhere.
- Compare with a partner and resolve one disputed route by naming who needs the fact and when.
- Write the most valuable durable entry in its proper form. Deliverable: the four-row table, the disputed route, and one entry.