Parallel work: tmux sessions, worktrees & handoff receipts

Key takeaways

1. Parallel progress starts with a dependency graph, not extra terminals.
2. Worktrees isolate files; tmux holds jobs; receipts say how to resume.
3. Switch only while work runs unattended; return on a named event.

Common misunderstandings of parallel development work

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

"Parallel work begins by opening more terminals for the same task."

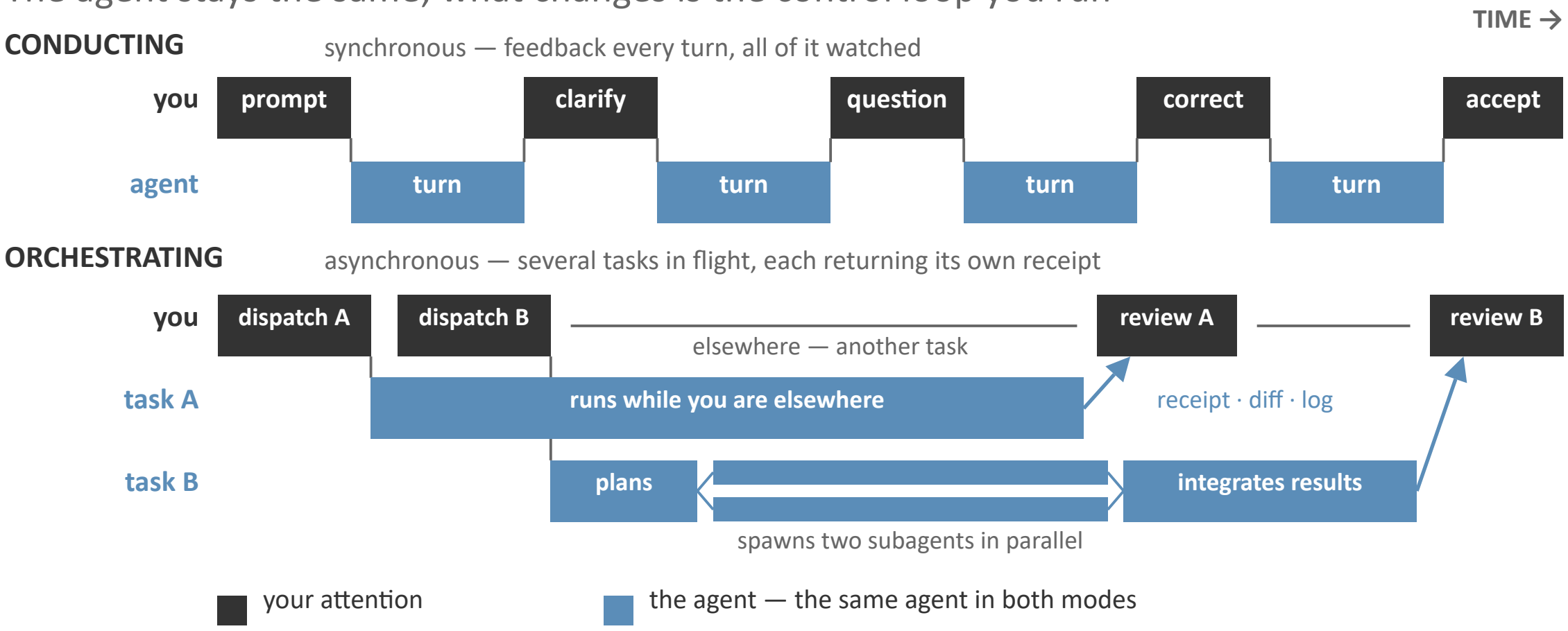
"A git branch gives each concurrent task its own working files."

"tmux preserves the intent needed to resume after an interruption."

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

Conducting and orchestrating

The agent stays the same; what changes is the control loop you run



source: verb framing adapted from the conductor/orchestrator roles in Osmani, Saboo & Kartakis, The New SDLC With Vibe Coding (Google, May 2026)

When to conduct, when to orchestrate

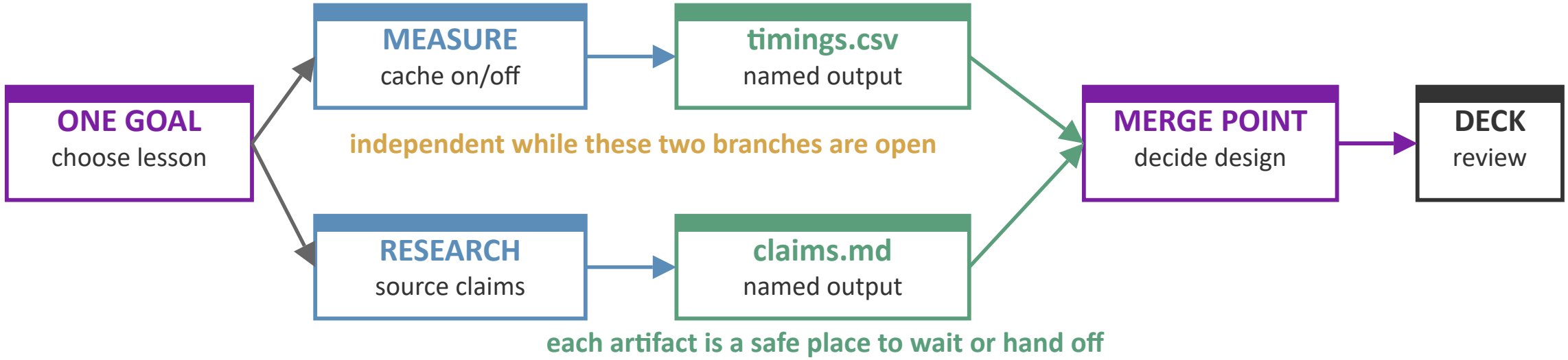
Five contrasts that pick the mode for a task

	Conducting	Orchestrating
Tempo	synchronous — you watch every turn	asynchronous — work runs while you are elsewhere
You control	each step, corrected as it happens	the goal, constraints, and acceptance evidence
Feedback	immediate, in the session	delayed — a diff, log, or receipt reviewed later
Throughput	capped at your watching speed	capped by how well the tasks separate
Reach for it	debugging, unfamiliar code, new APIs	well-specified tasks with checkable outputs

Conduct when each step needs your eyes; orchestrate when a receipt can stand in for them.

Parallel work means resumable state

- Parallel progress is not typing in two terminals at once.
- It is dividing one goal into tasks that can wait without losing their state.
- tmux preserves the conversation with the machine; a worktree isolates the files.



A task is ready to run in parallel when you can name its output and its merge point.

Remote work has three owners

Host, files, and session have different lifetimes

State	What owns it	What survives
Host	CPU, memory, processes, and the tmux server	Only while that particular machine stays up
Files	Worktree, branch, log, and result artifact	NFS or git can carry them to another machine
Session	Shell, tmux windows, panes, and running jobs	A disconnect, but not loss of the owning host

Files may follow you to another host. Running processes do not.

Begin with one goal, then draw the dependency edges

Only tasks without an incoming edge are ready now

Workstream	Concrete output	Can begin when…
Technical content	claims + sources	the lesson question is fixed
Local experiment	measurement table	the metric and baseline are fixed
Slide implementation	buildable deck script	the arc is stable enough to render
Schedule integration	generated HTML	the deck name and date are known
Visual QA	review notes or clean render	the PDF exists

Parallelize research and experiments; serialize decisions that change both.

Split on artifact boundaries

A good workstream can finish and hand back something reviewable

Good split	Weak split	Why
measure cache-on vs. cache-off	work on performance	the first returns a table
draft five source-backed claims	research the topic	the first has a stopping rule
implement one isolated deck	edit the course	the first names its file boundary
review a frozen diff	review while code changes	the first has a stable input

If two tasks continuously edit the same decision, they are one task pretending to be two.

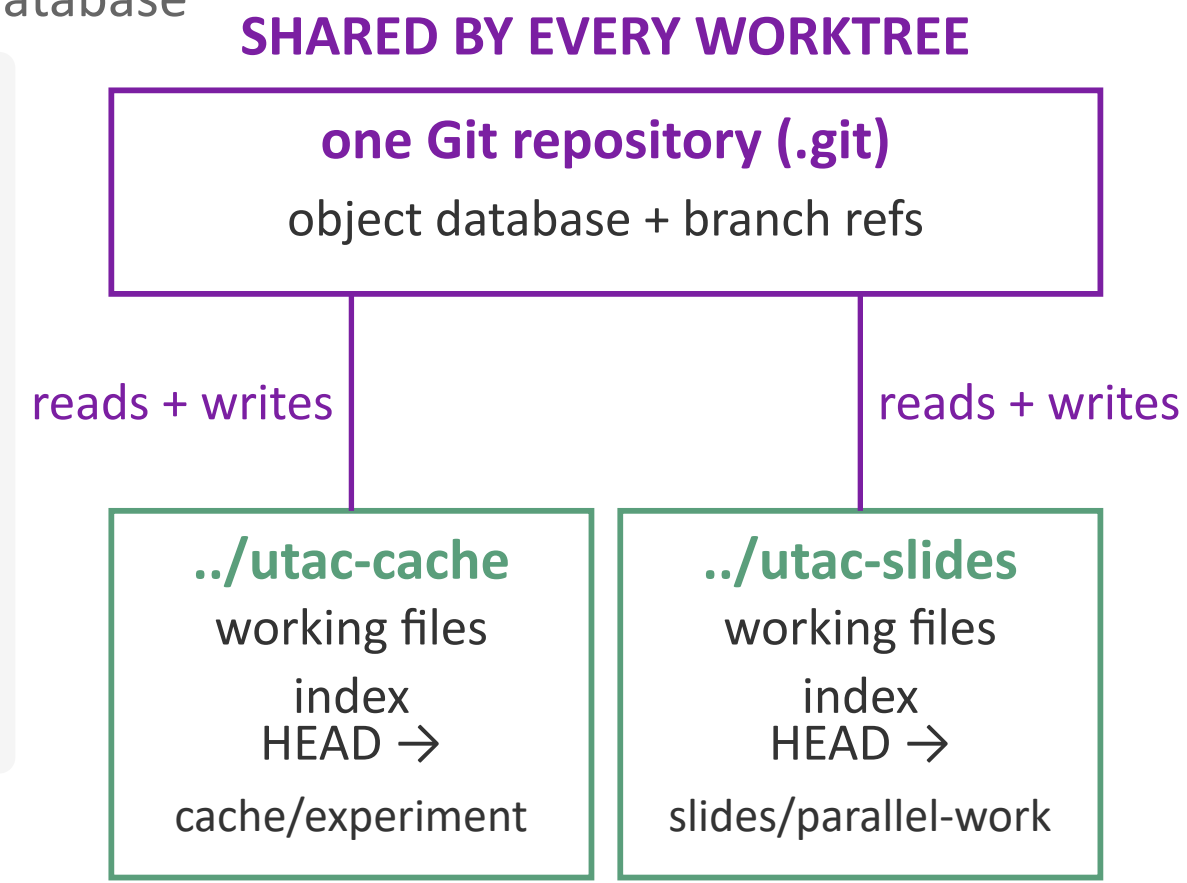
A worktree gives each task its own files

Separate checkout and branch; shared commit database

```
# From the main repository
git fetch origin
git worktree add ../utac-cache -b cache/
experiment origin/main
git worktree add ../utac-slides -b slides/
parallel-work origin/main
# -b creates and checks out the named branch

# See every checkout and its branch
git worktree list

# Remove only after the branch is integrated
git worktree remove ../utac-cache
```



Significant or overlapping agent work needs its own worktree.

A worktree creates a landing obligation

Use the smallest safe boundary, then close every temporary branch

Change	Use	Finish
Read-only work or a tiny edit to disjoint files	One checkout; Git and the agent can keep the diff scoped	Review and stage only the intended files
Significant, long-running, or possibly overlapping edits	A feature branch in its own worktree	Land immediately after verification

- Fast-forward the finished branch to main
  - Rebase first if main advanced; landing keeps one linear history.
- Clean up immediately after landing
  - Remove the worktree and both branch names; forgotten files can be lost, and an unlanded branch grows stale as main moves.

Landing = fast-forward to main, remove the worktree, delete both branch names.

tmux separates job lifetime from connection lifetime

The server and job live on the remote host; the client may detach

```
# First connection: record the owner, then start the long job
ssh HOST
hostname -f
tmux new-session -s cache-exp -c ../utac-cache
./run-benchmark 2>&1 | tee evidence/cache-exp.log
# C-b d detaches; the tmux server and job keep running

# Later connection: return to the SAME host
ssh HOST
tmux list-sessions
tmux attach-session -t cache-exp
```

A different host may see the files, but it does not own this tmux server.

Name sessions so /resume can find them

In Claude Code, /color changes only the prompt-bar color

utac — cache-exp **/rename cache-exp**

> rerun with the cache off **/color green**

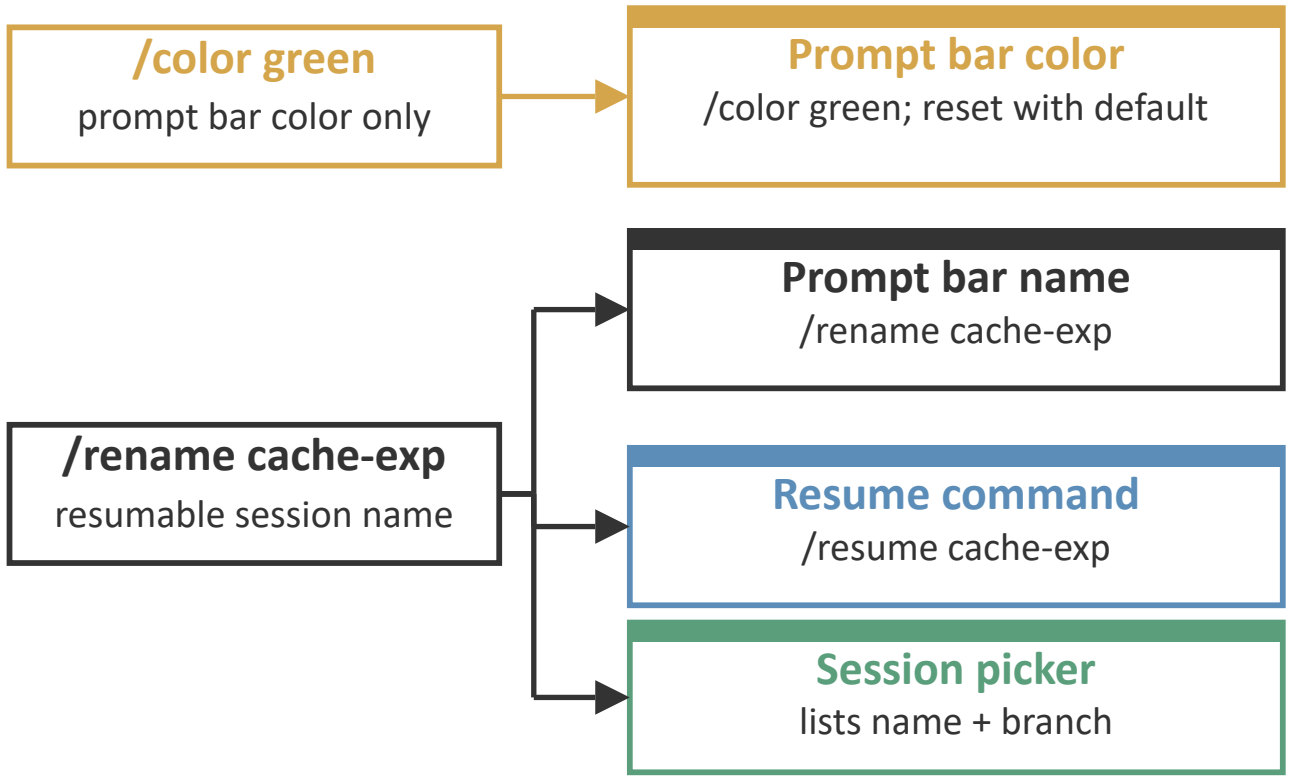
utac — cache-sources

> five claims, sources only

utac — cache-slides

> waiting on timings.csv

- /color default restores the default prompt-bar color.



/color changes only display; /rename sets the resumable session name.

A handoff receipt is ordinary text, not saved agent state

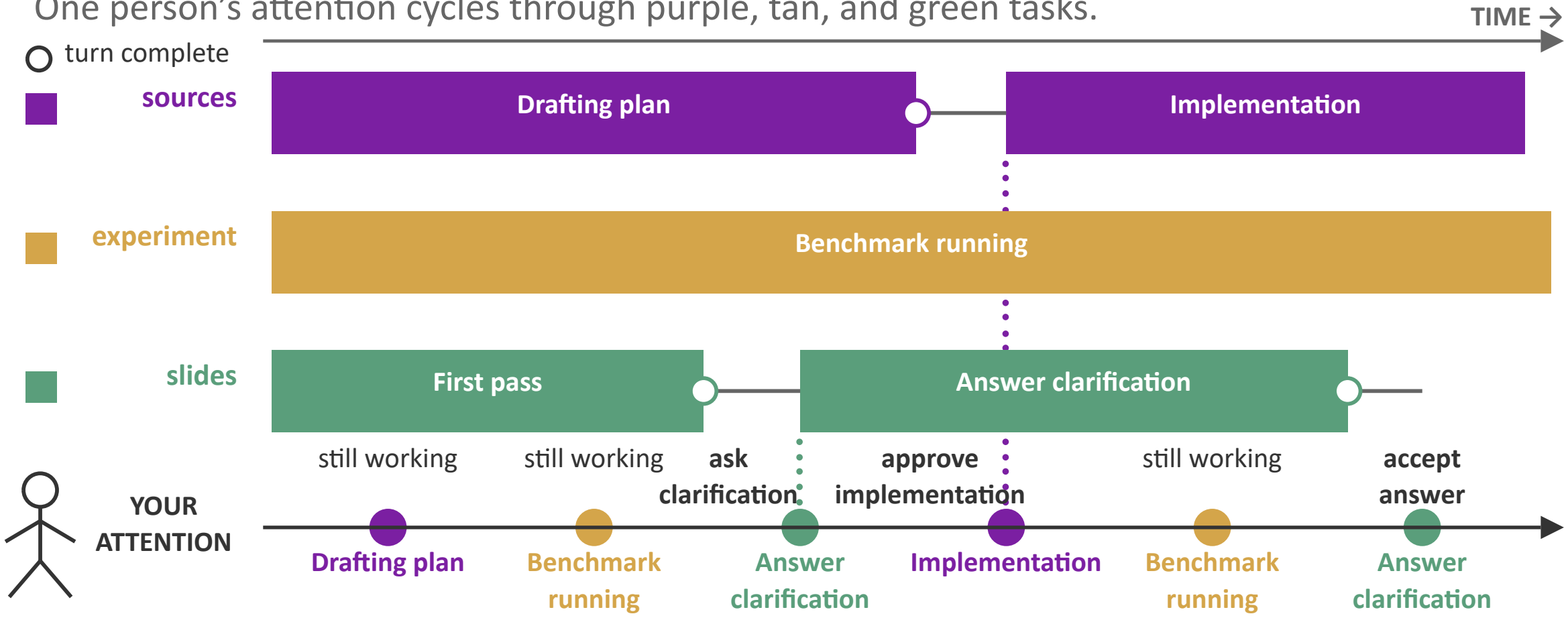
Record enough external state that a fresh session can continue

Before switching	Record explicitly	Fresh-session action
Name the owner	hostname + tmux target	return to that host and session
Locate the code	worktree/branch + git status	inspect the exact diff
Locate the evidence	command + log/artifact path	inspect; do not rerun
Name the wake-up event	process exits or dependency arrives	return only after that event

The receipt preserves facts needed to resume; it snapshots neither files nor agent context.

Switch when work can continue without you

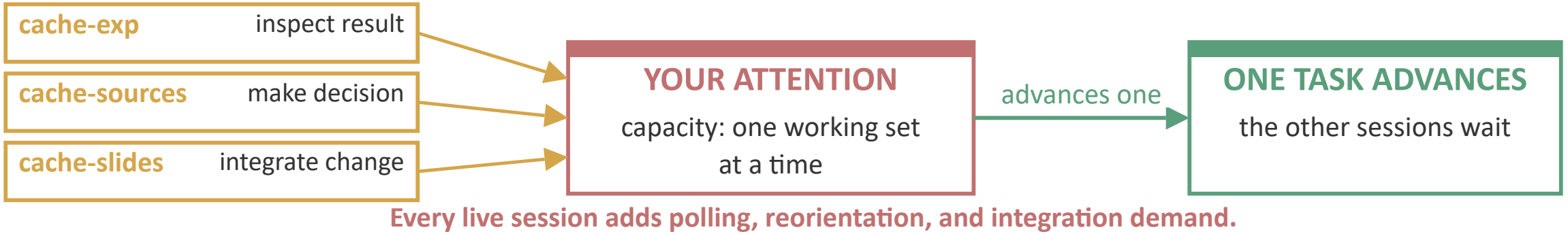
One person's attention cycles through purple, tan, and green tasks.



At each visit: act on a completed turn, or move on if work is still running.

More sessions can mean less progress

Parallelism pays only when waiting time exceeds coordination cost



Every live session adds polling, reorientation, and integration demand.

Smell	What it reveals	Repair
many unnamed shells	no task boundary	rename or close them
same files in two sessions	false independence	serialize the work
constant conflict resolution	split crossed an interface	move the boundary
forgotten finished jobs	no polling cadence	keep a short active-session list
long reorientation	weak handoff note	leave the next question explicitly

Human vs. agent ownership

The coordinator owns the task graph and integration decisions

Your Responsibility
Choose task and worktree boundaries
Name dependencies and merge points
Limit work in progress
Accept the integrated result

Agent’s Responsibility
Execute one bounded task
Leave commands and evidence
Report blockers instead of widening scope
Summarize the branch or artifact

One practical setup

The names encode enough state to recover after an interruption

- Overall goal: ship the cache lesson
  - One controller session tracks the task graph and integration order.
- cache-exp
  - Worktree: cache/experiment · output: timing table · next: explain variance.
- cache-sources
  - Read-only session · output: five claims with primary-source links.
- cache-slides
  - Worktree: slides/cache · waits for the experiment and claim list.
- Switch protocol
  - Record the host/session, branch, command, output, and next check. Work another ready task; return when the job ends or a named prerequisite arrives.

leave a remote run receipt

10 min

- You must edit `parser.py` and run a 30-minute benchmark from the exact reviewed commit; the run must produce `runs/summary.csv`.
- In pairs, decide: sequential or parallel, same or separate worktrees, and the output and merge point of each task.
- Draft six receipt lines: host/session, worktree/branch, command, check, artifact, and failure action.
- Swap receipts. The reader circles every fact they would otherwise have to guess; revise until the work can be recovered without chat history.