

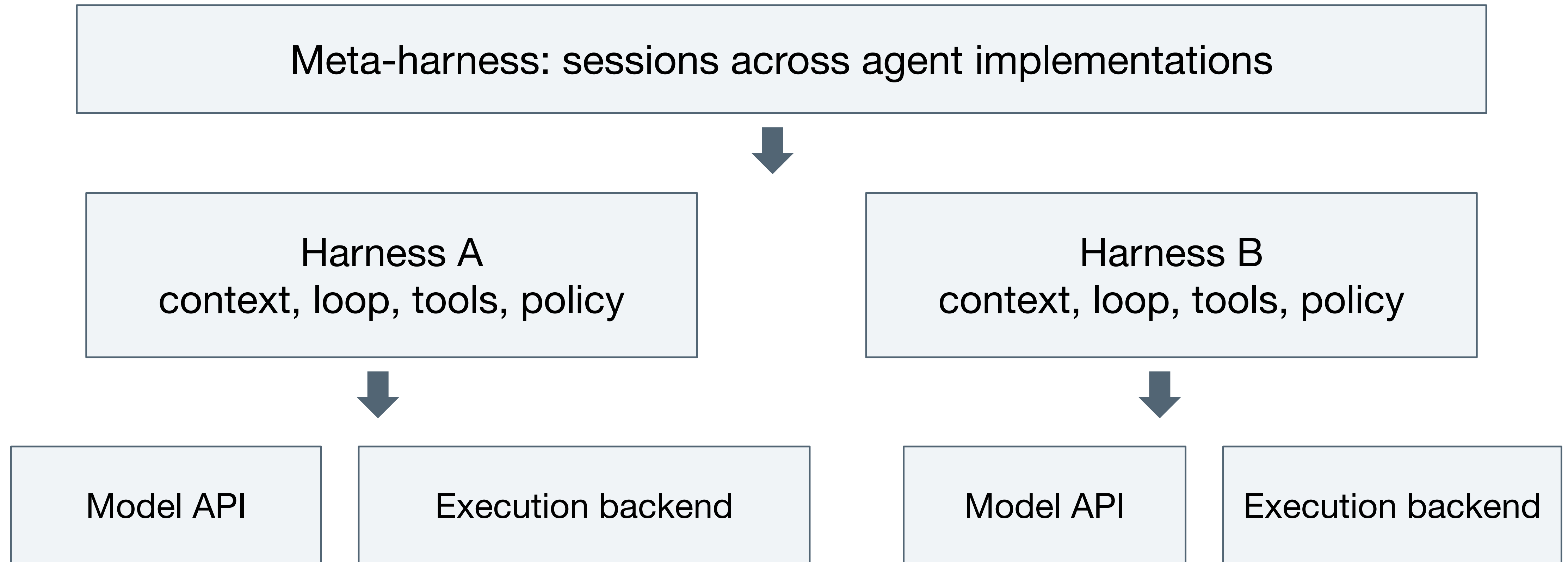
# **Agent Harnesses, Meta-Harnesses, and Runtimes**

CSE 291 · Fall 2026  
Yiying Zhang · November 5, 2026

# Today's Questions

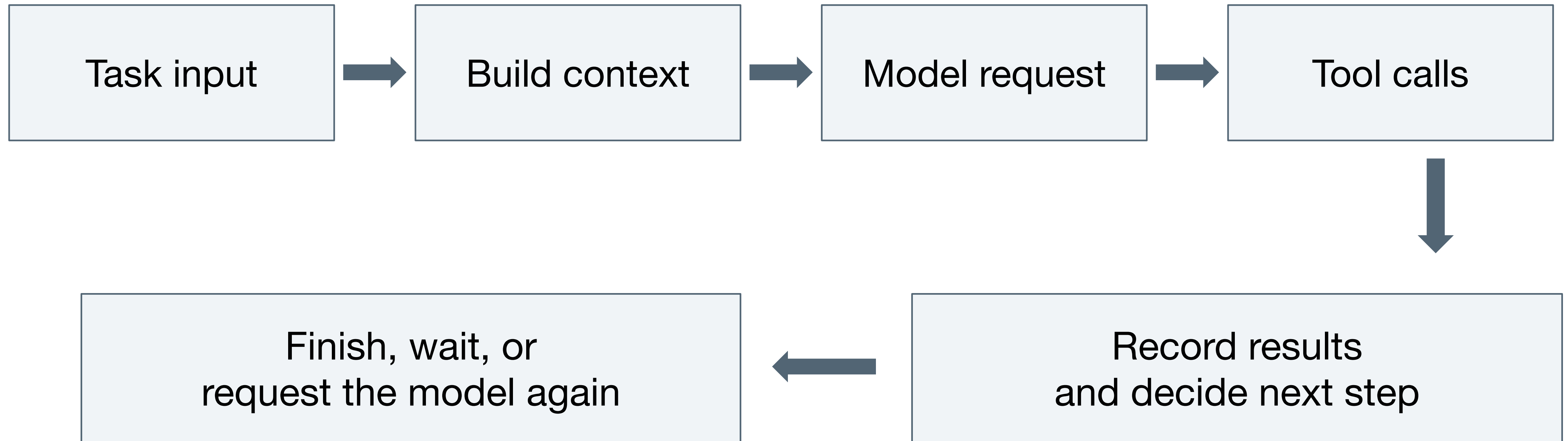
- What must run around a model to complete a task?
- What state survives a context reset, crash, or fork?
- How can one runtime manage different agent harnesses?
- What changes when a harness launches several workers?

# Where the Harness Fits



Each harness still owns its own model–tool loop.

# One Task, Several Model Calls

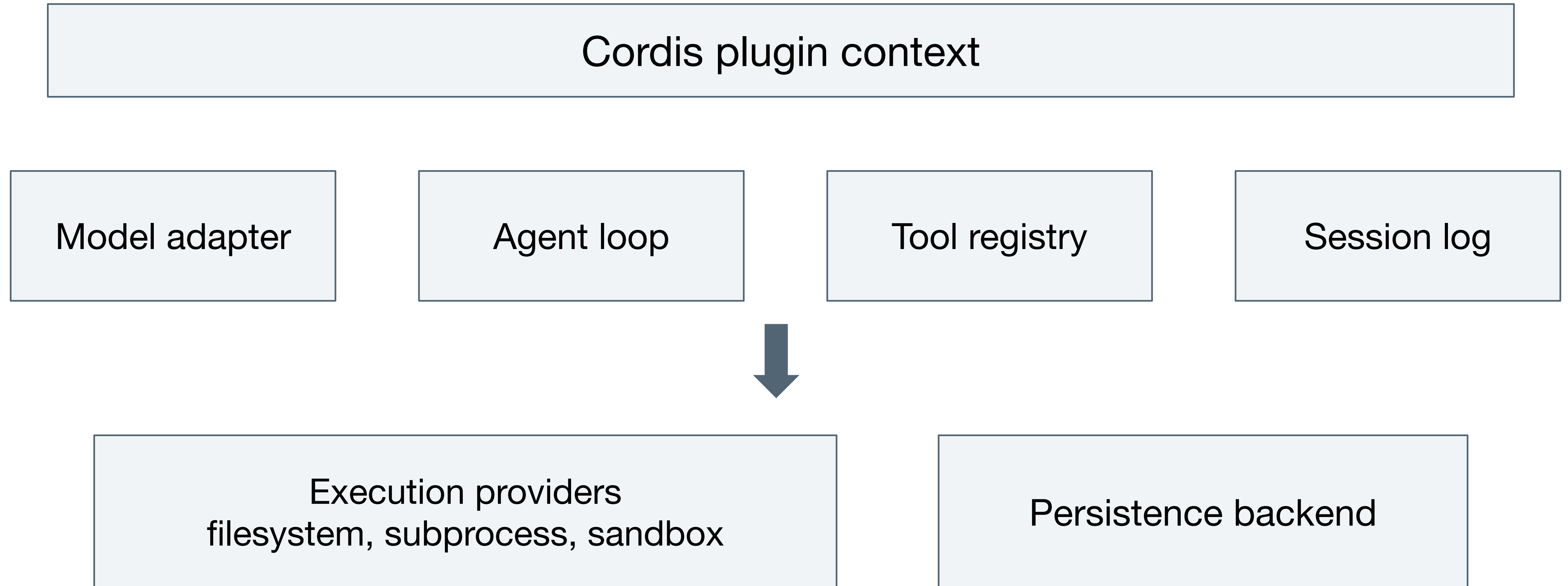


Results become input to the next iteration

# State Beyond the Prompt

- History and current context can contain different information
- Files and running tools outlive an individual model call
- Pending approvals, budgets, and operation IDs affect control flow
- External services may have changed even if the agent crashes

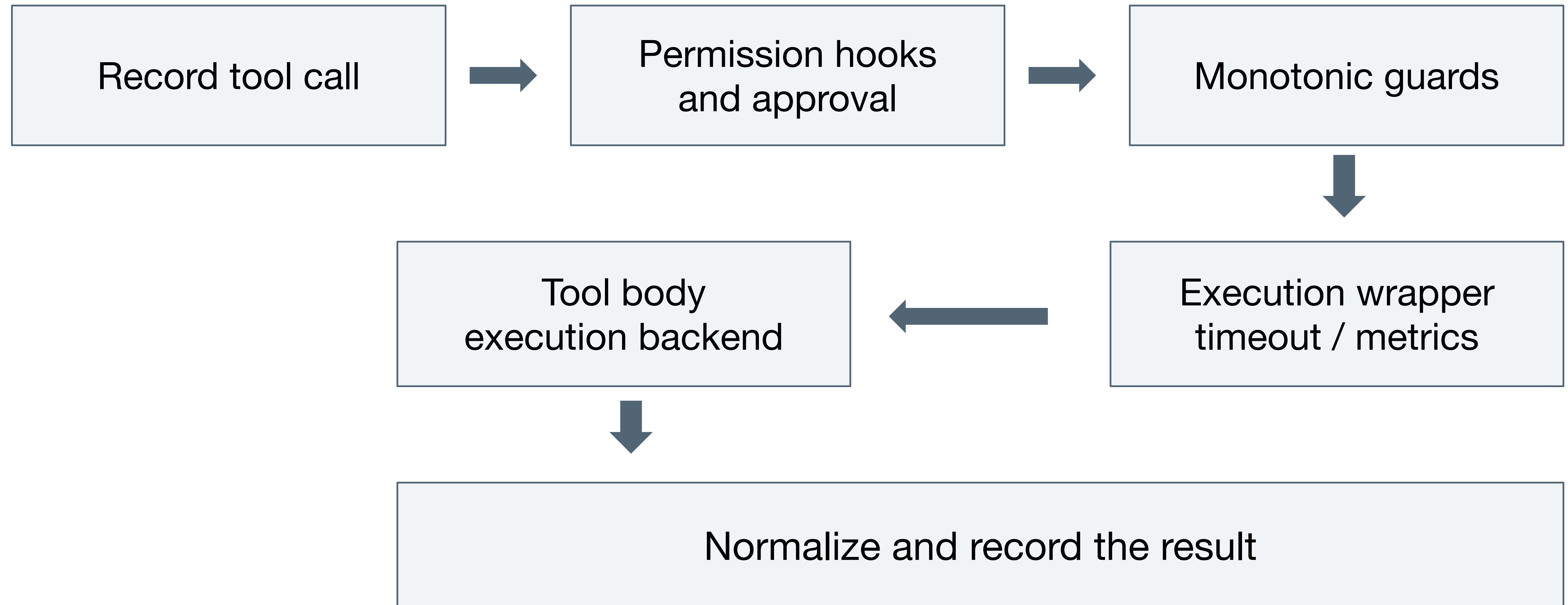
# DeepSeek Harness: Replaceable Components



# DeepSeek: Turns and Steps

- A step contains one model request and its tool executions
- A turn can contain several steps, or no step if input is rejected
- The loop assembles context, calls the model, then processes tools
- Live extension events and recorded session events serve different roles

# The Tool Execution Pipeline



Denied calls also produce a recorded outcome.

# History as an Event Log

Append-only session events  
input • assistant output • tool calls/results



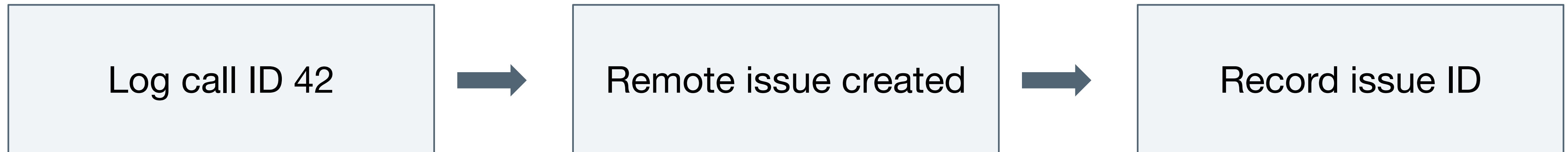
Model-facing history  
projection for next request



Inspection and recovery  
execution record

Persistence policy determines what survives a hard crash.

# The Ambiguous Tool Outcome



**CRASH**

- The remote action succeeded, but the log has no outcome
- Retry may create a duplicate issue
- Reconcile using operation identity when the service supports it

# Forking Has Several Meanings

Session fork



Selected interaction history

Worktree



Separate file checkout

OS / VM snapshot



Captured execution state

Remote side effects need a separate recovery protocol.

# Anthropic: Work Across Context Windows

- An initializer prepares the environment and a structured task list
- Each coding session completes a bounded piece of work
- Tests, progress records, and commits support the next session
- Compaction alone may leave unfinished work hard to reconstruct

# Context Selection Is a Runtime Policy

- Keep the full record separate from the next request's token budget
- Select recent events, summaries, and task-relevant artifacts
- Preserve provenance when retrieved material enters context
- Measure both retrieval cost and mistakes caused by omitted state

# Pause, Cancel, and Resume

- Waiting for a user is different from a failed tool call
- Cancellation must propagate to active model and tool operations
- A stopped process may leave remote work running
- Resume must distinguish unfinished work from completed effects

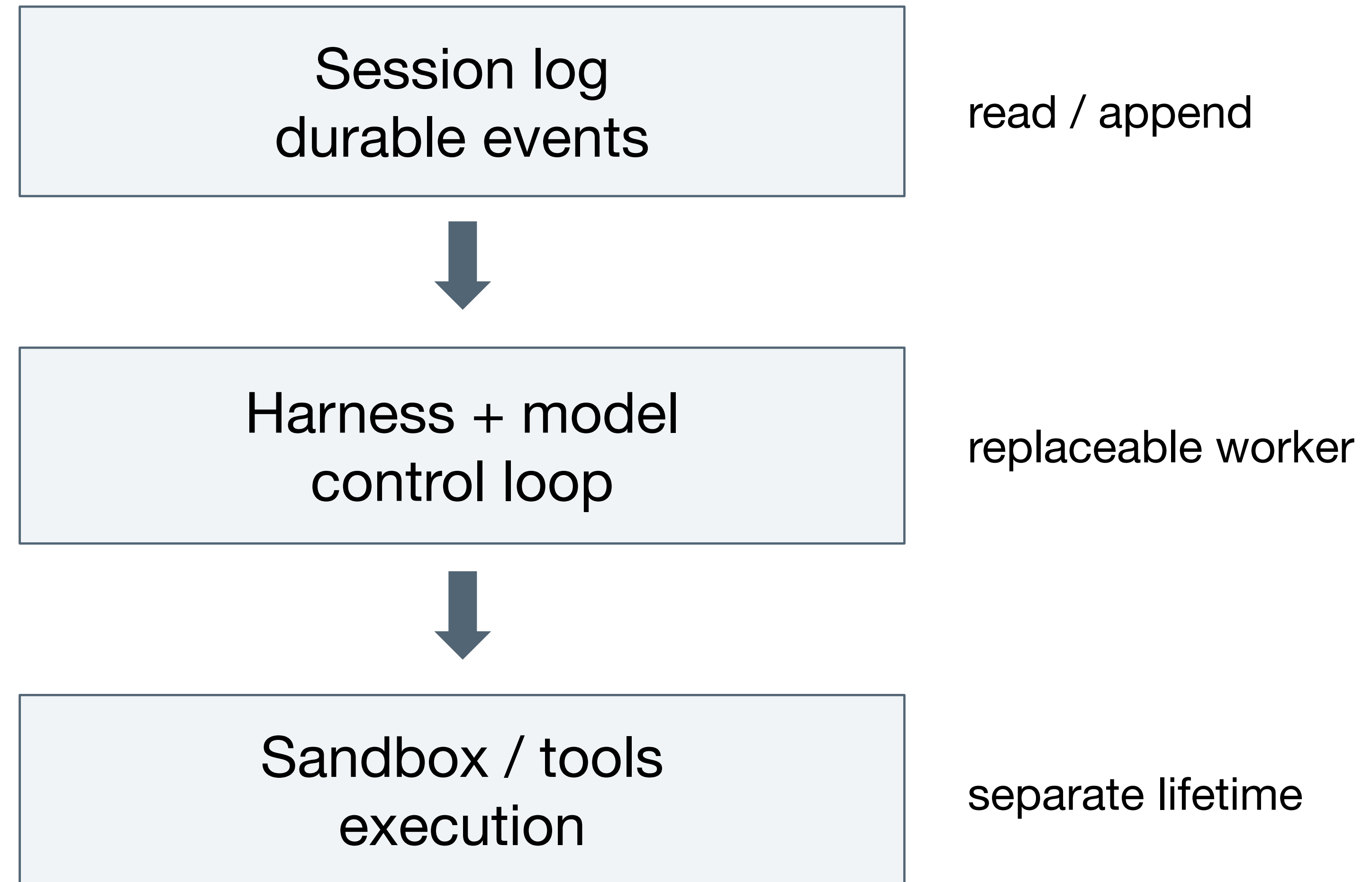
# Runtime Design Exercise

- An agent edits a file, starts a test job, then loses its connection
- Which events and identifiers should already be durable?
- How does a replacement harness find the test result?
- What can be retried safely, and what requires reconciliation?

# **BREAK TIME**

5 minutes

# Anthropic: Separate Failure Domains



# Recovery After Separation

- A replacement harness reads the external session log
- A sandbox failure becomes a tool error the harness can handle
- A replacement sandbox may need environment reconstruction
- Discussion: which local artifacts require their own persistence?

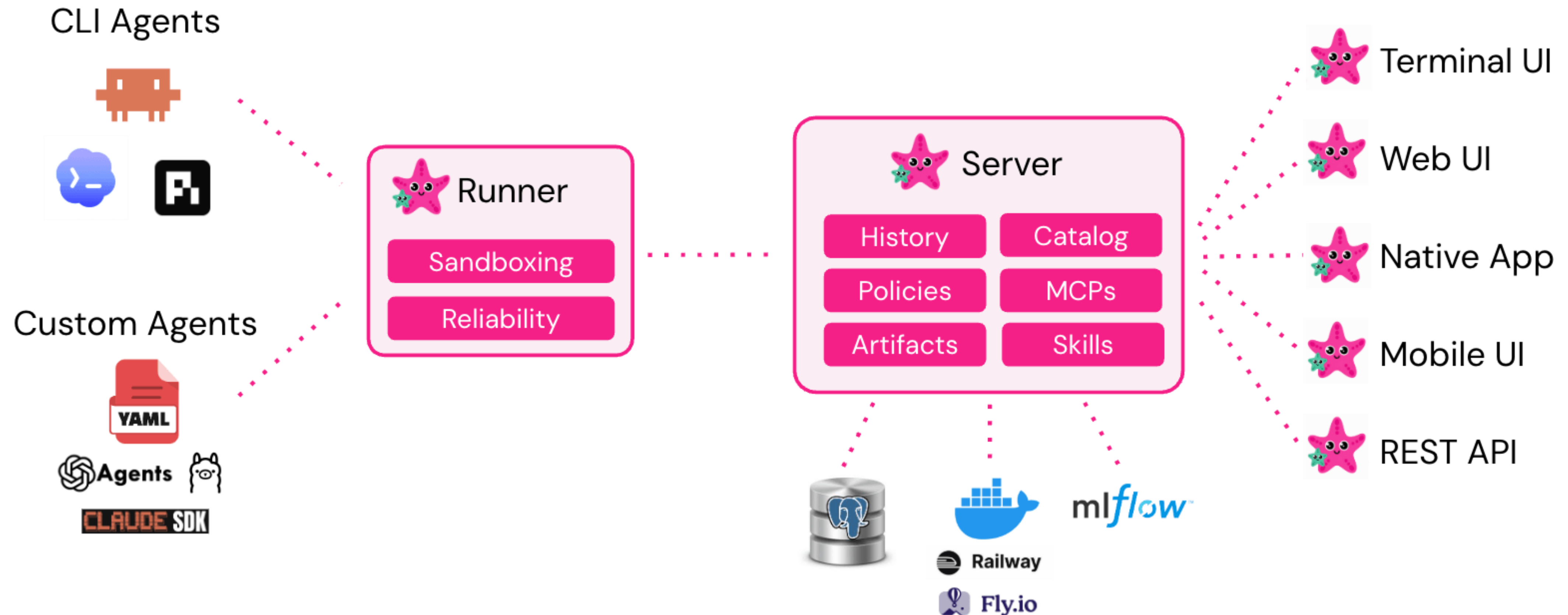
# Credentials and Untrusted Execution

- Managed Agents keeps service credentials outside the sandbox
- A proxy can attach credentials when accessing an external service
- The runtime still needs policy for allowed operations and data
- Question: can an authorized tool become a data-exfiltration path?

# Why a Meta-Harness?

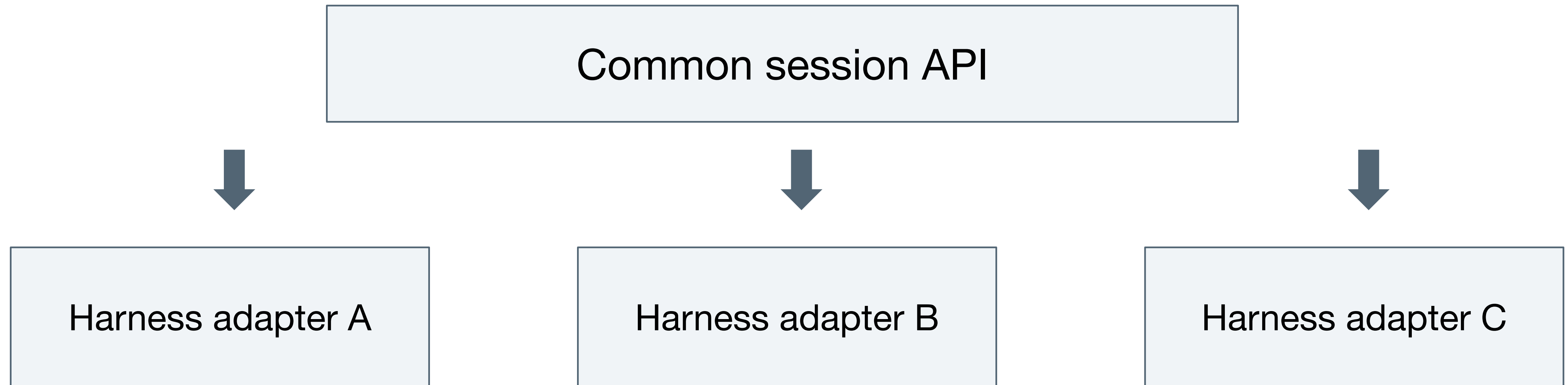
- Existing harnesses already own model loops and tool behavior
- A common layer can manage sessions across those harnesses
- Shared history, policies, and workspace access simplify operations
- Adapters must expose differences in resume, fork, and cancellation

# Omnigent Meta-Harness



Source: <https://www.databricks.com/blog/introducing-omnigent-meta-harness-combine-control-and-share-your-agents>

# The Adapter Contract

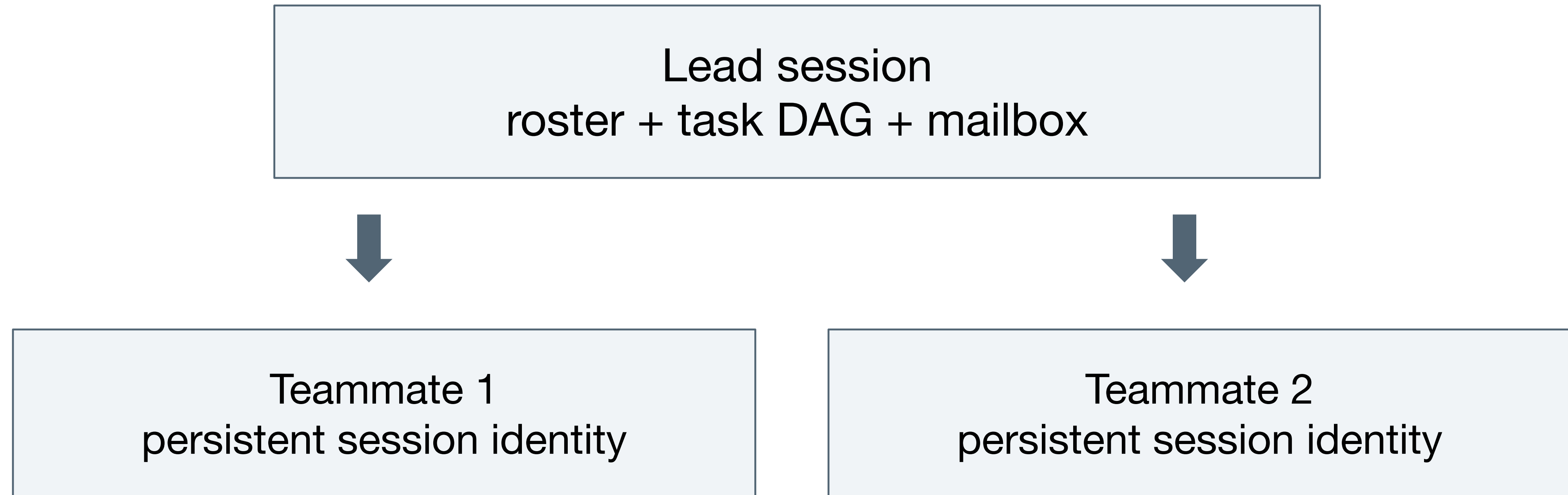


- Identity, status, input, events, and final artifacts
- Explicit support for resume, fork, approval, and cancellation
- Backend-specific semantics remain visible to the caller

# Omnigent: A Coding Run

- Polly delegates work across agents using different harnesses
- Separate git worktrees hold parallel code changes
- Another vendor's agent reviews a diff before human integration
- Worktrees separate files but do not establish a security boundary

# DeepSeek Agent Teams



Queued message → durable target receipt → acknowledgment

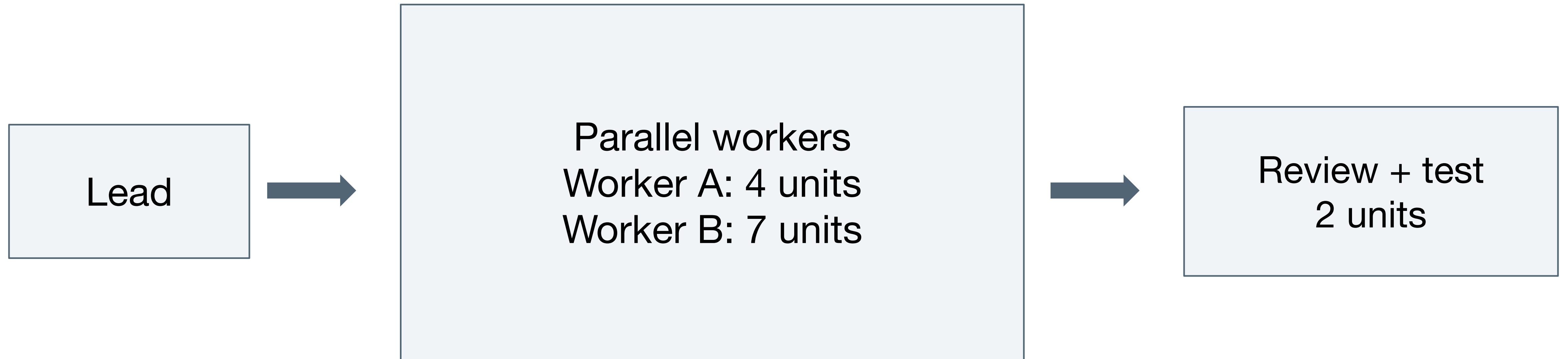
# Coordination Needs Explicit State

- DeepSeek task updates use revisions to reject stale mutations
- Dependency edges must preserve an acyclic task graph
- Its write scopes are advisory path prefixes, not file locks
- Question: what happens when two workers edit the same file?

# Runtime Cost of Parallel Workers

- Latency depends on the longest chain of dependent work
- More workers add model calls, context, and tool contention
- Task budgets need accounting across every child harness
- Cancellation should reclaim work after a branch loses its value

# DeepSeek-V4.1: Reward the Critical Path



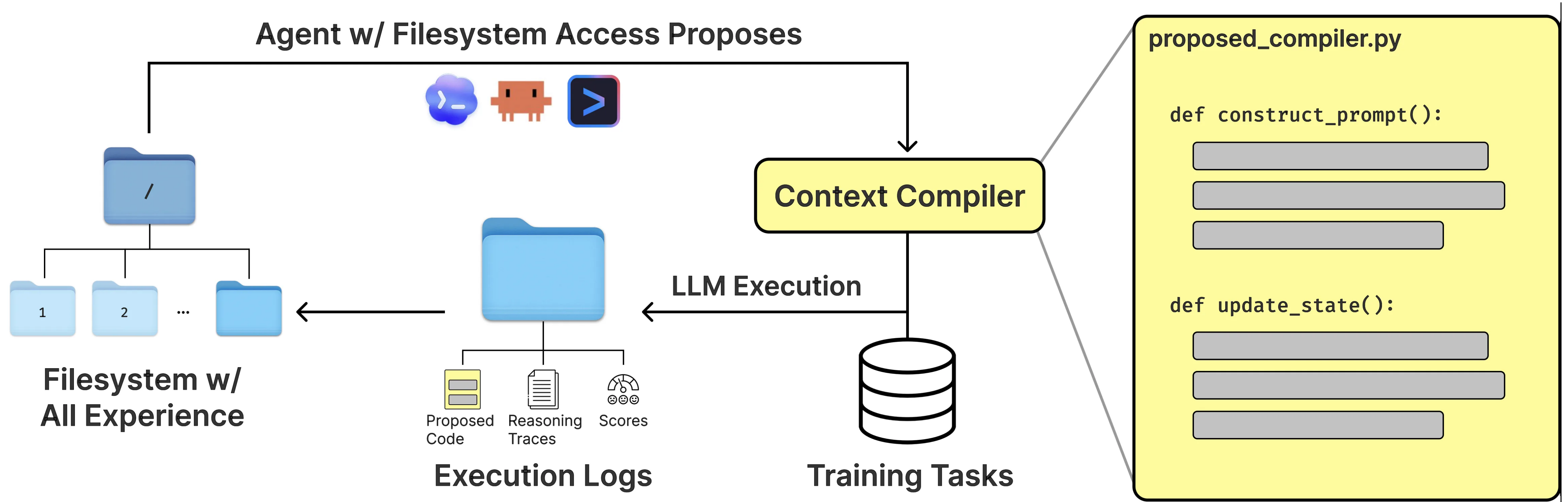
Classroom example: parallel work takes  $\max(4, 7) + 2 = 9$  units.

The report trains collaboration with a derived critical-path latency penalty.

# Evaluating a Harness

- Hold the model, task set, tools, and total budget fixed
- Report success together with tokens, wall time, and tool cost
- Inject crashes and duplicate messages at known boundaries
- Track duplicated effects, recovery time, and lost progress

# Meta-Harness: Optimizing Harness Code



Another use of “meta”: search over harness implementations using execution feedback.

# What Can a Harness Optimizer Overfit?

- A search may exploit quirks of the training tasks or evaluator
- A quality gain may consume more tokens or external tool calls
- A generated harness may weaken policy or failure handling
- Use held-out tasks and enforce budgets and safety constraints

# Project Connection

- Edge: profile model time versus harness and tool overhead
- Security: test policy enforcement through nested tool calls
- Fuzzing: resume a campaign without duplicating completed jobs
- Measure a small change to one runtime policy against a baseline

# Takeaways

- The harness determines how model outputs become actions
- Logs, execution state, and external effects need separate treatment
- Meta-harnesses compose implementations with different capabilities
- Reliable composition requires explicit policy and recovery semantics

# Next: Retrieval and Knowledge

- November 10: retrieval and knowledge infrastructure
- Reading: METIS from the course reading page
- How should a runtime find useful evidence within a limited budget?
- Connection: selecting context from logs and external knowledge