Defining the runtime that turns a model into an agent

WHAT IS A HARNESS

WHAT IS A HARNESS • HARNESS 1.0 ARCHITECTURE • A VISUAL PRIMER

§ 01 · DEFINITION

A harness is the *fixed architecture* that turns a model into an agent.

The modern harness was born bottom-up, out of coding agents — Cursor, Claude Code, Windsurf, Codex. These products started with a concrete problem: make an LLM write and edit real code across real repositories. In solving that problem, they independently converged on remarkably similar architectures. A while-loop that calls tools. A context manager that compresses history. A permission layer that keeps things safe. The same patterns, discovered separately, over and over again. The abstractions that define these common architectures are what we call a *harness*.

Two things make a harness fundamentally different from a framework.
First, a harness works out of the box. You don't configure a harness into existence. It ships as a working agent with a fixed architecture — iteration loop, context management, tool registry, permission layer — already wired together and already running. There is no assembly step.

Second — and this is the deeper shift — a harness is not designed for humans to build agents. It is designed for the agent to accomplish almost any task. The model reads instruction files and learns your project. It discovers available tools and composes them. It writes its own skills to extend what it can do. It spawns sub-agents when the task gets too big. The human provides the goal; the harness figures out the rest.

> *"Just as computer architectures matured in the 80s, we expect harness architectures to mature over the next couple years."*
> — ARIZE RESEARCH

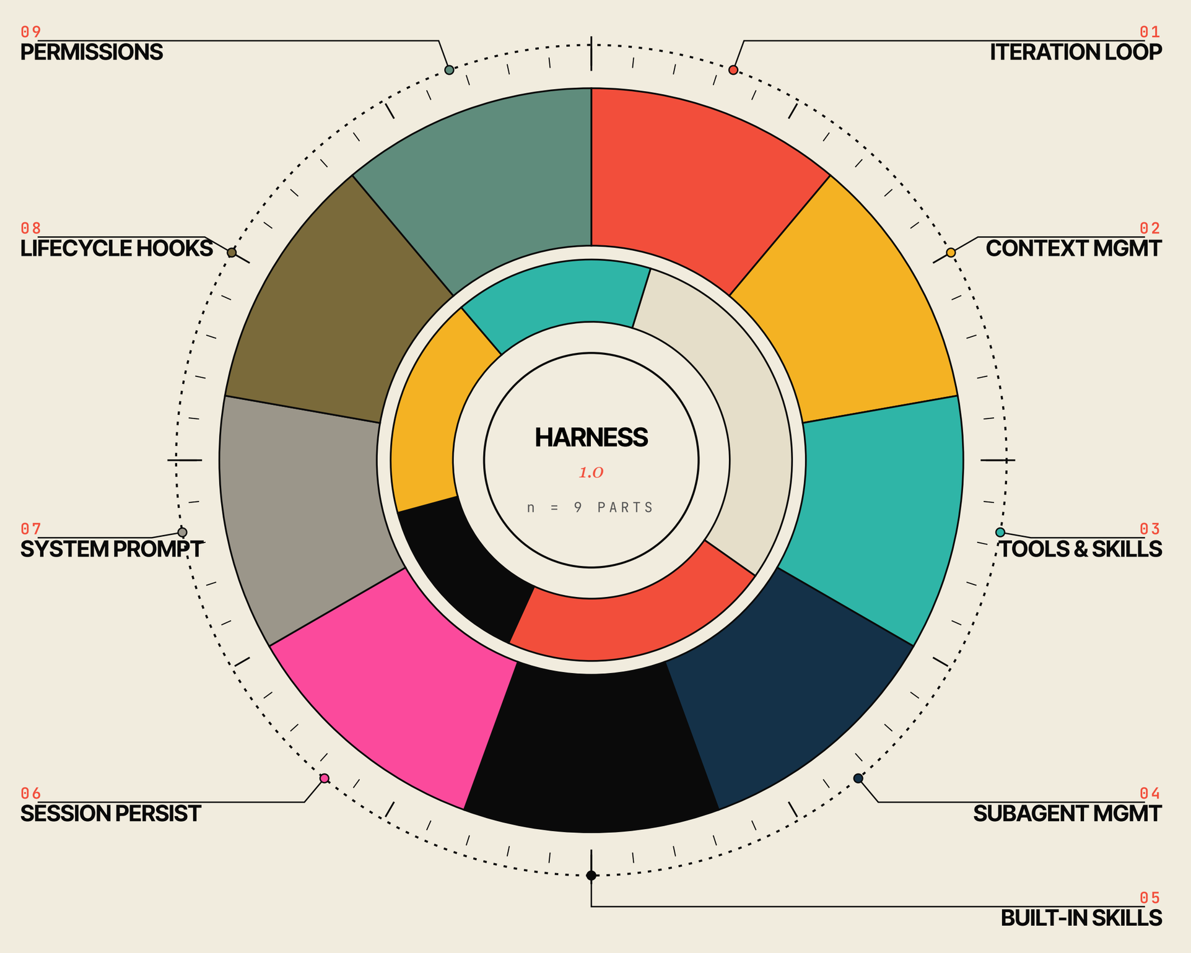
A HARNESS IS	A HARNESS IS NOT
+ A fixed runtime architecture	— LangChain / LangGraph
+ Born from coding agents	— A prompt template
+ A while-loop around the model	— A framework for humans to assemble agents
+ Context, tools, and control-flow	— The model itself
+ Observable end-to-end	— A single API call

FIG. 1 — The harness is the layer between a pre-trained model and a useful behavior. Source: Arize Research, *What is a Harness*, 04.2026.

§ 02 · ANATOMY

Nine components of the Harness 1.0 architecture.

Components fall into nine distinct areas. Every production harness we've studied implements all of them — and the quality of each is where vendors differentiate.



THE NINE PARTS

01 · Outer Iteration Loop `while()`
The while-loop at the heart of the architecture: decide → call tool → push result → repeat until done.

02 · Context Management `tokens`
What enters context, how large data is compressed, how tool results are summarized before returning.

03 · Skills & Tools `registry`
A registry teams can extend: build skills, add tools, manage availability, dispatch to the right executor.

04 · SubAgent Management `spawn`
Isolated child agents with their own session, restricted tools, and focused prompt. Spawn · Restrict · Collect.

05 · Built-in Skills `baseline`
The non-negotiable baseline: read/write/edit files, shell, grep, glob, plus higher-level commit/PR/test skills.

06 · Session Persistence `JSONL`
Append-only state to disk; resume exactly where a crash left off; validate, detect drift, probe health.

07 · System Prompt Assembly `dynamic`
CLAUDE.md · git · environment · tools · permissions — stitched per project with character budgets.

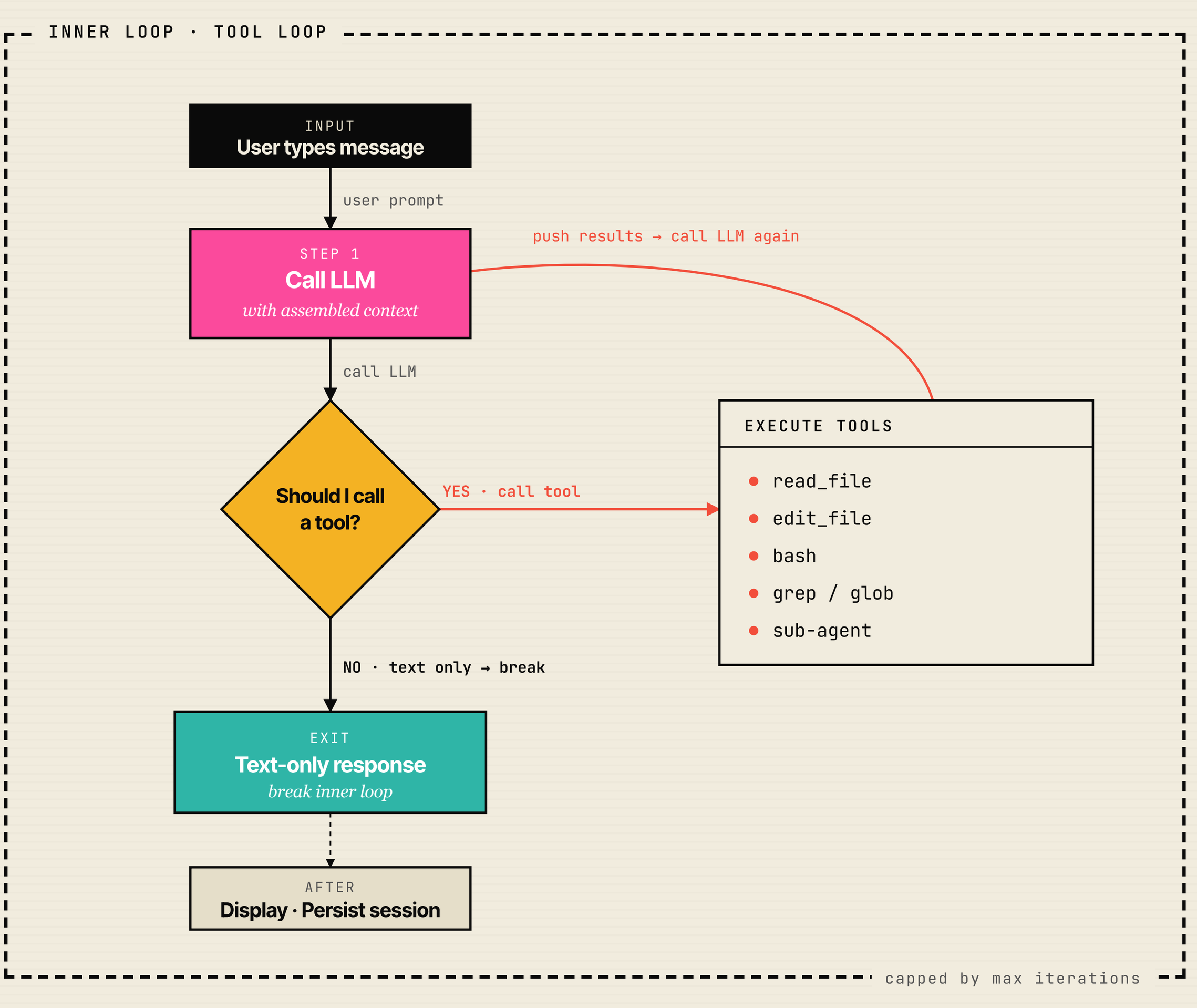
08 · Lifecycle Hooks `pre/post`
The extensibility seam: pre/post-tool hooks allow, deny, or modify — audit, policy, and workflow.

09 · Permission & Safety `enforce`
Read-only · workspace-write · full access. Static modes + interactive approval + declarative rules.

§ 03 · THE CORE LOOP

The inner loop is a while-loop.

FIG. 2 — machine-paced · capped by max iterations



The while-loop is the **core architectural foundation** *of the Harness 1.0 architecture.*

Decide
The model reads the system prompt and current context, then picks a tool — or produces a text-only answer to end the loop.

Dispatch
The harness validates the call against permissions, runs the tool, captures stdout/stderr and structured results.

Feed back
Tool results are pushed back into context as a `tool_result` block. The model is called again.

Terminate
Loop exits on a text-only response, on max iterations, or on unrecoverable error.

§ LOOKING AHEAD · HARNESS 2.0

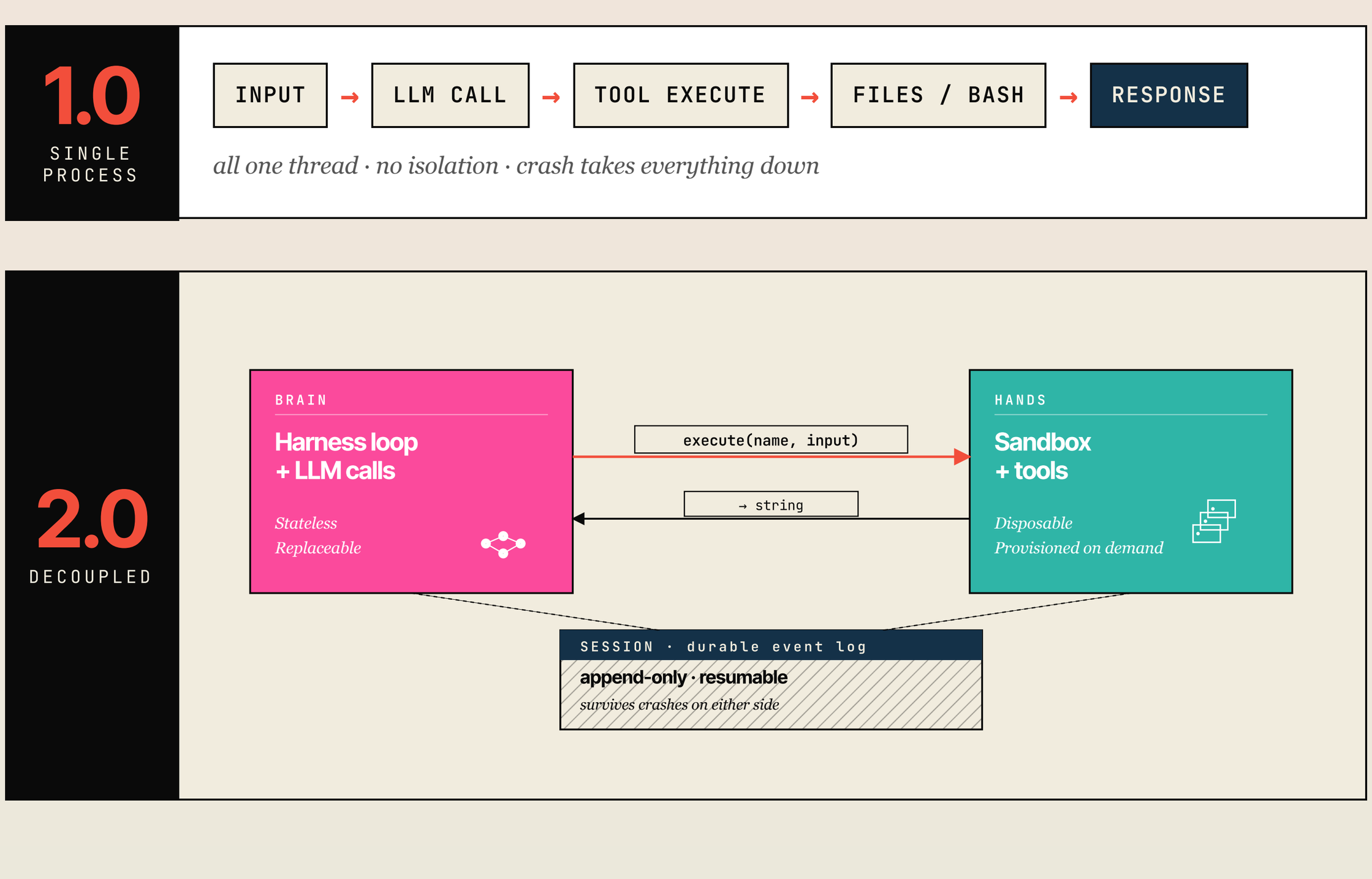
Anthropic defines Harness 2.0 as a decoupling of brains and hands.

Harness 1.0 is a single process. The loop calls the LLM, executes tools, reads files, runs bash — all on one thread. It's how every coding agent works on your laptop today.

In **Harness 2.0**, the harness is just the outer loop. The **brain** is the harness loop and LLM calls — stateless, replaceable. The **hands** are sandboxes where tools run — disposable containers provisioned on demand. Between them sits the **session**: a durable event log that survives crashes on either side. The brain just calls `execute(name, input)` and gets a string back. It doesn't know if the hands are local or across the world.

> *Many brains, many hands, one stable set of interfaces. This is where harness architecture is heading.*
> ANTHROPIC.COM/ENGINEERING/MANAGED-AGENTS ↗

FIG. 5 — Decoupled architecture per Anthropic's *Managed Agents*.



N BRAINS · N HANDS · 1 STABLE SET OF INTERFACES

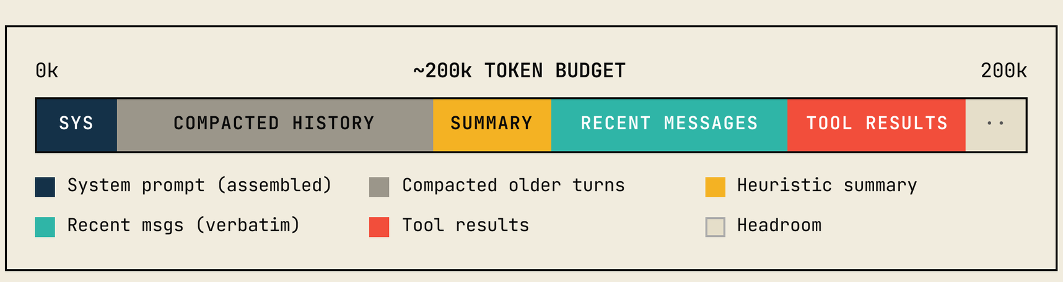
Context, Prompts, & *Sub-Agents*

DIAGRAMS FOLLOW
§ 03.1 — 03.3

§ 03.1 · CONTEXT MANAGEMENT

What actually fits in the context window.

Session trajectory grows every turn. The harness decides what to keep verbatim, what to summarize, and what to throw away — before the next LLM call.



COMPACT TRIGGER: 100k tokens → keep 4 most recent messages verbatim · summarize older turns (heuristic, not an LLM call).

Tool-result budgets
- `read_file` — 10 MiB max, line-windowed
- `grep` — 250 lines returned
- `glob` — 100 files returned
- `bash` — 16 KB of stdout/stderr

What streams into context
- User messages (verbatim)
- Assistant messages + reasoning blocks
- Tool results (variable, often large — hence the caps)
- Search + command output — truncated & selected

What the harness never passes through
- Raw file contents above the line-window
- Full command output above the buffer
- Full message history above the compact trigger

FIG. 3 — illustrative token shares · derived from *context-management* diagram (Arize Research).

§ 03.2 · SYSTEM PROMPT ASSEMBLY

Stitched per project.

The system prompt is not a string — it is a pipeline. A static scaffold below a dynamic, per-project boundary.

Discovery walks ancestor dirs:
- {dir}/CLAUDE.md
- {dir}/AGENTS.md
- {dir}/.config/instructions.md
- {dir}/.config/rules.md

Budgets: 4k chars per file · 12k total

Config merge order (last wins):
1. ~/.config/settings.json (user)
2. {cwd}/.config.json (project)
3. {cwd}/.config/settings.json

STATIC ▲ **Static Scaffold**
Intro: "You are an interactive agent..." · # System (tools, permissions, hooks) · # Doing tasks (engineering discipline) · Fixed across all projects.

=== DYNAMIC BOUNDARY ===

DYN ▼ **# Environment context**
Model family · working directory · current date · OS name + version.

DYN ▼ **# Project context**
`git status` (short + branch) · staged + unstaged diff · last 5 commits · current branch.

DYN ▼ **# Agent instructions**
Per-file subsections with scope labels · content from discovered instruction files · dedup across ancestor dirs.

DYN ▼ **# Runtime config**
Merged JSON: permissions, MCP servers, model family, hooks. Per-project override wins.

↑ SENT AS SYSTEM PROMPT TO LLM ↑

§ 03.3 · SUBAGENT MANAGEMENT

Spawn · Restrict · Collect.

At some point a task gets too big or too parallel for a single conversation. Sub-agents work in isolation and report results back — the parent polls a manifest on disk.

PARENT AGENT · main thread
Full session · All tools · Interactive permission prompts · Invokes the sub-agent tool with a prompt string and a type.
SPAWN WORKER THREAD
RETURNS STATUS: RUNNING

↓ SPAWN · ISOLATE · EXECUTE ↓

Explore READ-ONLY
Navigates the code and the web. Cannot mutate state. Best for planning and recon.
TOOLS: `read_file` `grep` `glob` `web_fetch` `web_search` `skill`

General Purpose FULL
Full capabilities — but cannot spawn further sub-agents. The everyday worker.
TOOLS: `bash` `read` `write` `edit` `grep` `glob` `web` `skill` `todo`

Verification EXECUTE + READ
Runs commands and reads output. Built for testing and post-change confirmation.
TOOLS: `bash` `read_file` `grep` `glob` `web_fetch` `todo`

KEY CONSTRAINT · *all sub-agents run with an isolated session, an appended sub-agent directive in the system prompt, and no interactive permission prompts. The sub-agent tool is* **not** *in the sub-agent's own toolset — no infinite spawn. Results land in* `agent-{id}.json` *(manifest) and* `agent-{id}.md` *(output); the parent polls.*

The Nine Components *in detail*

REFERENCE SHEET
DIAGRAMS & COPY

A closer look at each component.

Full architecture reference · direct from the Harness 1.0 spec.

01 · OUTER ITERATION LOOP
The while-loop at the core
The model uses the system prompt and decides what tools to call. It iterates on tools until it is finished. This is the foundation.

02 · CONTEXT MANAGEMENT
What enters the window, what gets compressed
How data is pulled into context, how it is simplified and compressed, how tool results are returned. Cheap to get wrong, expensive to notice.

03 · SKILLS & TOOLS
A registry teams extend
Built-in tools (read, edit, bash, grep) are primitives. **Skills** are the `SKILL.md` layer on top — organizational knowledge encoded as markdown the agent invokes by name.

04 · SUBAGENT MANAGEMENT
Isolated, restricted, collected
A sub-agent gets its own session, a restricted tool set, and a focused system prompt. Isolation is the architectural rule.

05 · BUILT-IN SKILLS
The non-negotiable baseline
File ops, shell, code nav — if the agent cannot read and edit files, it is not a coding agent. Higher-level built-ins (commit, PR, tests) are where vendors differentiate.

06 · SESSION PERSISTENCE
Append-only JSONL
Each message, each tool result, each compaction — one line at a time. Resume exactly where a crash left off; detect drift; run a health probe after compaction.

07 · SYSTEM PROMPT ASSEMBLY
Static scaffold · dynamic project context
Walks ancestors for CLAUDE.md / AGENTS.md. Injects git status, env, tool list. Stitched with character budgets so the window isn't blown before turn one.

08 · LIFECYCLE HOOKS
Pre- and post-tool extensibility
Structured protocol: JSON on stdin, exit codes for allow/deny. Hooks enforce policy: "never `rm -rf`," "log all writes," "approve bash that touches prod."

09 · PERMISSION & SAFETY
Static modes + interactive approval
Read-only · workspace-write · full access. Each tool declares its minimum. Bash is classified dynamically (`ls` is read-only, `rm` isn't). Human pauses on the dangerous ones.

§ 05 · CLOSING

The harness is the architecture. *The model is the CPU.*

— ARIZE RESEARCH · AI ENGINEERING DESK · 04.2026

FURTHER READING
- Context management diagram — internal
- Harness components (9-part) — internal
- Control-loop diagram — internal
- System-prompt assembly — internal
- SubAgent management — internal

ON ARIZE.COM
- AX — AI engineering platform
- Phoenix OSS tracing
- Evaluating AI Agents (learn)