09 — Planning

🕒 state of the art 2026-07 · revised 2026-07-26 · 📖 ~12 min read

Learning objectives

By the end of this chapter, you should be able to:

1. **Distinguish** the three planning instruments — plan mode, todo list, and decomposition — and what each one requires;
2. **Explain** why plan mode is implemented as a case of the permission system (enforced, not requested);
3. **Compare** ReAct (interleaving reasoning and action) with plan-then-execute and decide when each one fits;
4. **Evaluate** the tactical × durable stratification (task plan × session goal) and decomposition with dependencies;
5. **Implement** permission-enforced plan mode in harness-zero (step 8).

The problem

Models tend to act rashly: they edit before understanding, they "solve" before mapping the problem. Planning artifacts force a reading-and-design phase before the writing phase — and give the human a cheap approval point (reviewing a plan costs less than reviewing a diff).

Three distinct instruments, frequently confused:

1. **Plan mode** — a *state* of the harness in which writing is forbidden; the agent only researches and proposes.
2. **Todo list** — working memory for the task in progress: what remains, what is done.
3. **Decomposition** — breaking large work into trackable subtasks, possibly with dependencies.

Scientific foundations

The planning literature explains *why* these instruments exist — and warns against trusting the model's plan.

- **Interleaving beats plan-everything-first (when the environment is unpredictable)** — ReAct, arXiv 2210.03629 (ICLR '23) interleaves reasoning traces and tool actions in the same loop: each observation revises the next thought, so the agent recovers from surprises instead of executing a stale plan. Decision: carry reasoning and observations in a single alternating transcript.
- **Planning ahead helps (when the scope is known)** — Plan-and-Solve, arXiv 2305.04091 has the model emit an explicit plan before solving, suppressing missing steps. The two do not contradict each other: they are distinct regimes — the explicit plan for tasks of known scope, interleaving for uncertain environments.

- **Decompose only when needed** — ADaPT, arXiv 2311.05772 decomposes **recursively and only when the executor fails** a subtask, adapting depth to difficulty and to model capability. Decision: try to execute first, decompose on failure — this avoids the over-planning that most harnesses (wisely) do not impose.
- **Isolate context per subtask** — Beyond Entangled Planning, arXiv 2601.07577 (2026) decomposes into a **DAG of sub-goals** and gives each one *scoped* context, so local errors and replanning do not pollute a monolithic history — reporting up to −82% tokens. A direct bridge to subagents (ch. 10).
- **Do not trust the model's plan — externalize** — PlanBench, arXiv 2206.10498 and TravelPlanner, arXiv 2402.01622 show that raw models fail at plan generation and lose track of multiple constraints (GPT-4 ~0.6% on TravelPlanner). Decision: externalize constraint tracking into an artifact (plan/todo) instead of trusting the model to hold everything in context. This is *the* justification for the todo list.
- **The taxonomy as a checklist** — the planning survey, arXiv 2402.02716 organizes the components into five avenues (task decomposition · plan selection · external module · reflection · memory); PlanGenLLMs, arXiv 2502.11221 gives six criteria (completeness, executability, optimality, representation, generalization, efficiency) and PLANET, arXiv 2504.14773 organizes benchmarks by category.

(Full bibliography and pointers: `livro/bibliografia.md`.)

Industry sources

- **Plan mode is a permission layer** — Choose a permission mode (Claude Code): plan mode removes write/execute for the *entire session*; the agent reads and explores, but every mutation is held until you exit (Shift+Tab cycles Normal → Plan → Auto-accept; `/plan`; `--permission-mode plan` for CI). Decision: planning is guaranteed by **revoking the mutation tools**, not by asking the model to "plan first". This is the official confirmation of round 1's discovery.
- **Explore → Plan → Code → Commit** — Best practices (Claude Code): the exploration and planning phases are "the cheapest in tokens and the most valuable in outcome". Decision: separating exploration from execution structurally prevents solving the wrong problem before understanding the code.
- **Todo as a machine-tracked artifact** — Todo tracking (Agent SDK): `TodoWrite` creates checklists with three states (pending/in_progress/completed) updated in real time. Decision: externalizing the plan into a structured artifact gives the agent a working-memory anchor and the user progress visibility — and the evolution into a *tasks* system with dependencies and persistence turns the plan into durable infrastructure, not scrollback.
- **Thinking between actions** — The "think" tool adds a reasoning step *in the middle* of tool use (after the result arrives); extended thinking exposes reasoning blocks with `budget_tokens` and, in the 4-series models, **interleaved thinking** (think → call tool → think about the result → call again). Decision: allocate explicit budget to planning steps and let reasoning interleave with tools — planning is not a one-shot prefix, it is continuous. *(anthropic.com returns 403 through the proxy; confirmed via independent mirrors.)*
- **Spec-driven: the spec is the durable plan** — GitHub Spec Kit formalizes `specify` (what/why) → `plan` (architecture) → `tasks` (actionable list) → `implement`, with approval gates between stages;

Kiro generates `requirements.md` (EARS `WHEN…THE SYSTEM SHALL…`), `design.md`, and `tasks.md`, and **derives a dependency graph** that runs independent tasks in concurrent waves. Decision: the plan becomes a persisted source of truth re-consumed at every phase — it is exactly the method by which **this book is written** (see the project constitution).

- **Planning is an orchestration function — and the tension over parallelizing** — Anthropic's multi-agent system has the *lead* analyze the query, **write the plan to memory**, and only then spawn workers with isolated specs (planning as a dedicated role). Cognition ("Don't Build Multi-Agents") counters: Devin centralizes planning in one continuous context, because planning *is* context management — parallelizing workers becomes a game of "telephone" between conflicting implicit decisions. Decision: decompose-and-parallelize is a cost/benefit gate, not a default (connects to ch. 10). *(cognition.com 403 through the proxy; confirmed via mirrors.)*
- **See also**: the living collection Awesome Harness Engineering — Planning & Task Decomposition gathers more consultable resources for this dimension (patterns, articles, and implementations), curated by problem.

The state of the art

1. Plan mode = permission mode (now the official pattern)

Round 1's discovery — harnesses implement plan mode **as a case of the permission system** (ch. 07), not as its own subsystem — has ceased to be an observation and become a documented standard: the official Claude Code docs describe plan mode exactly that way (removes mutation for the session). Entering plan mode = switching to a ruleset that denies writes; exiting = restoring, with explicit approval. The mature pattern combines three guarantees: read-only that is **enforced** (not requested), the plan as a **persisted artifact** (not just text in the conversation), and **explicit approval** before executing.

2. ReAct became the default; the explicit plan retreated to long-horizon work

The clearest signal in the living book came from n8n: its **Plan-and-Execute Agent was deprecated** (it only exists in legacy V1, alongside ReAct), and V2/V3 converged on the pure Tools Agent — planning implicit in the model. This instantiates the scientific thesis: as models plan better inline, interleaving (ReAct) beats plan-then-execute as the default, and the **explicit plan concentrates where it still pays**: long-horizon work, human in the loop, and decomposition of large tasks. It's not that planning died — it's that cheap planning migrated inside the loop.

3. The todo list is externalized constraint tracking

What PlanBench and TravelPlanner prove (models lose track of multiple constraints) is what the todo list solves: a checklist with states (Codex `update_plan`, `TodoWrite`, the Hermes/Goose `todo`, OpenHarness's `TODO.md`) takes the constraints out of the model's head and puts them in an artifact. The modern evolution is giving that artifact **dependencies and persistence** — gemini-cli's graph tracker, Kiro's dependency graph, the DAG from "Beyond Entangled Planning".

4. Tactical × durable — the personal agents' contribution

Coding harnesses have a *task* plan; what they lack is the *durable* one. **OpenClaw** fills that with four layers: `update_plan` (tactical, one `in_progress` step at a time), **Goals** (one durable goal per session, with token budget and states, injected per turn and visible in the UI), **Task Flow** (durable orchestration with steps and JSON state), and standing orders (persistent policies). This tactical × durable stratification is the frontier the personal-agent category brought to the discipline.

5. Planning is the weakest dimension — and that is a data point, not an accident

Across all rounds, planning was the industry's lowest score (Codex 2, Goose 2, Aider 2, Hermes 2, OpenHands 1, n8n 1, IronClaw 2; only gemini-cli and OpenClaw reach 3). The living book's reading (expiration log, "enforced plan mode", 🔵 open): the prosthesis exists because models act rashly, and it expires when models plan under risk spontaneously — which has not yet happened. The persistent weakness of the dimension *is* the evidence that the prosthesis is still needed.

> EXECUTIVE SUMMARY
>
> What's most modern: plan mode as a permission layer (the official pattern); ReAct/interleaved thinking as the default, with the explicit plan reserved for long-horizon work; the todo/checklist as externalized constraint tracking, evolving into dependency graphs; and the tactical × durable stratification. **What to steal:** enforce read-only through permissions, not through the prompt; externalize the plan into a persisted artifact with states; give thinking budget to planning steps; and decompose-and-parallelize only when the task's breadth pays for the cost.

Hands-on — harness-zero, step 8

Step 8 (`harness-zero/etapas/08-plan/`) adds plan mode to harness-zero by **reusing** the `PermissionPolicy` from step 6: entering plan mode sets a mode the policy translates into "every write tool is denied"; the agent only reads and proposes; exiting asks for approval and restores the mode. It is the concrete demonstration of the chapter's thesis — plan mode is not a subsystem, it is a configuration of the permission domain that already exists. Completeness exercise: `propor_plano` already persists the artifact (`PLAN.md`); you add the requirement that exiting plan mode only happens with an approved `PLAN.md` — the gate between planning and executing.

Check your understanding

1. Why does it make sense to implement plan mode as a mode of the permission system, instead of a dedicated subsystem? (It reuses an existing mechanism and gets for free the guarantee that read-only is *enforced*, not suggested to the model.)

2. Your agent operates in an unpredictable environment (API responses change the next step). Do you plan everything up front or interleave reasoning and action? Why? (Interleave — ReAct: each observation revises the next thought; a fixed plan goes stale.)
3. A benchmark shows your agent losing track of 8 constraints in a task. Which planning instrument attacks that, and why? (Todo list / checklist — it externalizes constraint tracking out of the model's context.)

Appendix A — How each repository handles planning

> Per-harness evidence, with paths — supplemented online, expanded each round.

opencode (round 1) — plan as an agent

Plan mode is a **built-in `plan` agent** with a read-only ruleset (denies edits, asks confirmation for bash) — planning is switching agents, not just modes. The `plan_exit` tool (`tool/plan.ts`) closes the cycle: it asks for approval, **writes the plan to a file**, and transitions to the `build` agent. Dedicated prompts (`prompt/plan-mode.txt`, `plan-reminder-anthropic.txt` — reminders per model family). Per-session todos via `todowrite` (`session/todo.ts`).

gemini-cli (round 1) — plan with gatekeeping and decomposition

`ApprovalMode.PLAN` (`policy/types.ts`) with `enter-plan-mode`/`exit-plan-mode`: a read-only state whose prompt lists the available tools, and `getApprovedPlanPath()` **gatekeeps execution**. Todos via `WriteTodosTool`. The instrument the others don't have: the optional **tracker** (`trackerTools.ts`) — tasks with dependencies (`tracker_add_dependency`) and a graph (`tracker_visualize`). Plan mode has its own behavioral eval (`evals/plan_mode.eval.ts`).

OpenHarness (round 1) — the minimal, correct version

`EnterPlanModeTool` sets `settings.permission.mode = PLAN` (blocks all writes); `ExitPlanModeTool` restores — the most direct implementation of the plan-mode-is-permissions equivalence. Todos in `TODO.md` via `TodoWriteTool` (persistent, readable). Bundled `plan` skill; heavyweight decomposition in the autopilot subsystem (a queue of `RepoTaskCard`).

OpenClaw (round 2) ⭐ — tactical × durable in four layers

`update_plan` (multi-step plan, one `in_progress` at a time), **Goals** (a durable per-session goal with token budget and states, injected per turn and visible in the UI), **Task Flow** (durable orchestration with steps and JSON state), and **standing orders** (persistent policies). The tactical × durable stratification that coding harnesses lack.

Codex CLI (round 2) — structured checklist

`update_plan` tool (checklist visible in the TUI (Terminal User Interface)) + `ReviewTask`. No two-phase plan mode with plan approval before execution — the economy of "planning first" lives in the checklist, not in a permission gate.

Aider (round 2) — plan-then-edit via coder modes

`/ask` (discusses without editing), `/architect` (reasons about the "how" before delegating), and `/context` (uses the repo-map to converge on the files). Lightweight plan-then-edit, with no persisted plan artifact and no todo list; the `architect→editor` split executes the plan with a second model.

Goose (round 2) — declarative recipes

Recipes (YAML/JSON with instructions, typed parameters, `response.json_schema`, retry) + the `todo` extension + `final_output_tool`. Declarative/reusable planning, without a two-phase plan mode.

Hermes (round 2) — todo + Kanban

`todo` tool + iteration budget + a **Kanban system** for multi-agent coordination with specs. Planning coupled to the loop, without a separate formal planner.

n8n (round 2) — the planning that retreated

The **Plan-and-Execute Agent** exists but is **legacy** (V1 only, alongside ReAct/Conversational); V2/V3 converged on the pure Tools Agent. Planning became implicit in the model (+ optional `ToolThink`). The clearest case of the explicit plan losing to interleaving.

IronClaw (round 2) — temporal planning, not decomposition

No first-class task decomposition; the loop's "planner" is strategy composition. Its strength is *temporal* planning (scheduling, leases/heartbeats — supplementary dim. 14).

OpenHands / ohmo (round 2)

OpenHands: a planner tab in the UI and hooks, without a first-class decomposition subsystem in this repo (score 1; the core migrated to the SDK). ohmo: inherited plan mode/todos that assume a TUI — no plan-approval surface in a chat channel.

Frameworks (frameworks round)

LangGraph: planning as an explicit graph of nodes (the plan *is* the topology); Agents SDK and CrewAI: planner/executor roles and sequential/hierarchical processes; the spec-driven camp (Spec Kit/Kiro) treats the plan as a versioned artifact with gates. Where coding harnesses improvise the plan inside the loop, frameworks materialize it as first-class structure.