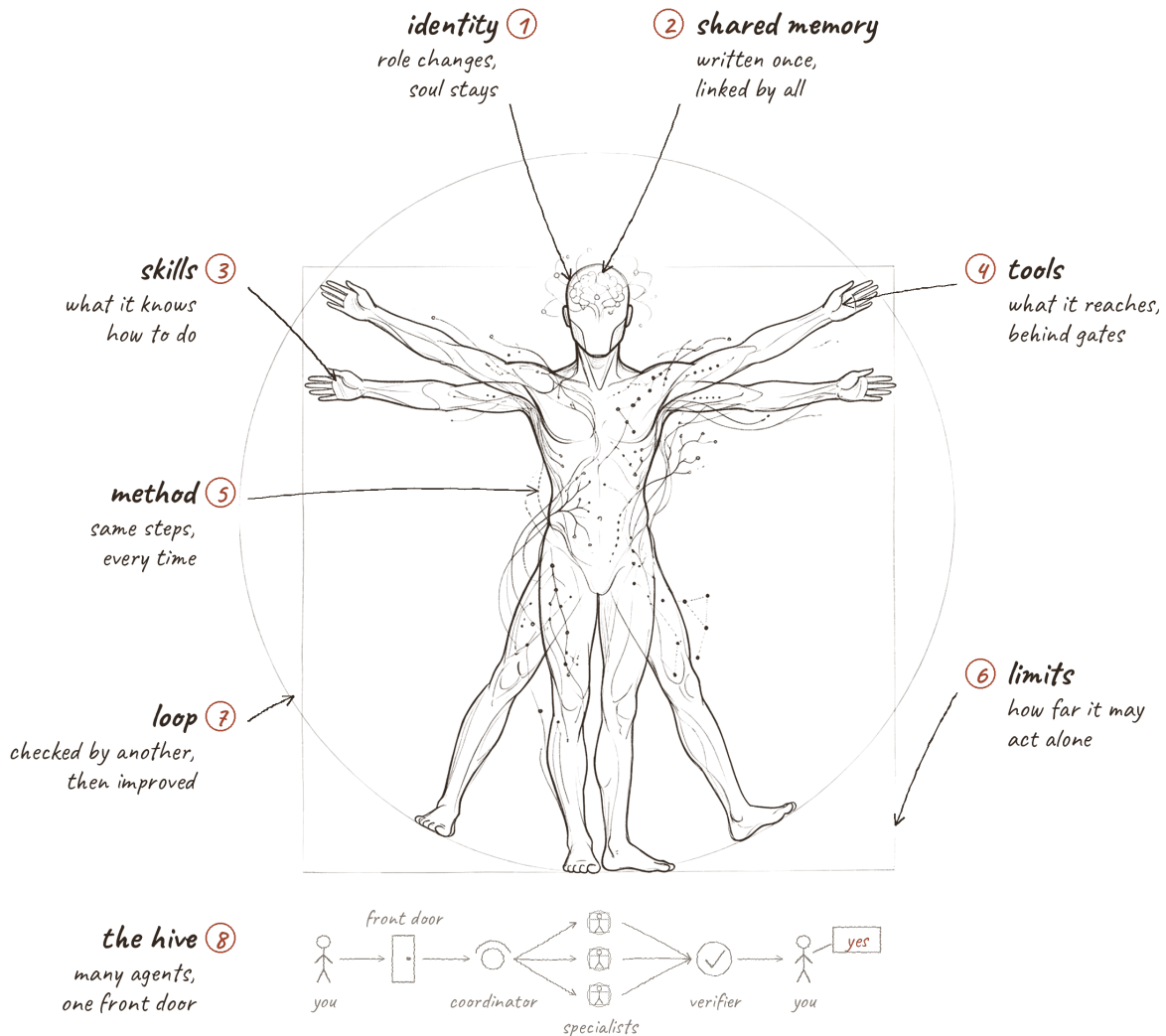


the hive mind

many agents, one nervous system



A RESEARCH GUIDE TO AGENT DESIGN

The Hive Mind

How to design AI agents that work as one: the layers of a single agent, the team around it, and the nervous system that joins them.

Contents

	Introduction	3
01	Identity Who is this agent, and what is its job?	5
02	Shared memory What does every agent know, and how does it read it?	8
03	Skills What does the agent know how to do?	12
04	Tools What can the agent reach, and what stops it?	15
05	Method How does the work move, step by step?	18
06	Limits How far may the agent go on its own?	22
07	Loop How is the work checked, and how does the agent get better?	25
08	The hive How do agents hand work to each other?	29
09	The nervous system How does a team feel itself as one body?	33
	Putting it together	37
	References	39

INTRODUCTION

One body, many minds

Why build an agent in layers, and why build agents as a team?

Most AI agents start life as one long prompt. It works for a week. Then a rule gets added for one client, another for a new tool, a third after a bad output, and soon nobody can say which instruction lives where, which ones contradict each other, or why the agent did what it did. The prompt has become a junk drawer.

This guide describes a different way to build: an agent is a small file that says who it is and links to the parts it uses. Everything the whole team shares is written once, in one place, and every agent points to it instead of keeping its own copy. The parts are grouped into layers, and each layer answers one question. When there is more than one agent, a team layer decides how they hand work to each other, and a proposed nervous system lets the team sense itself as one body.

The drawing on the cover is the map. The figure is one agent: its head holds identity and memory, its hands hold skills and tools, its spine carries method, the square marks its limits and the circle its loop. The strip along the bottom is the hive: you, a front door, a coordinator, specialists, a verifier, and you again at the end.

The parts at a glance

PART	QUESTION IT ANSWERS	BELONGS TO
Identity	Who is this agent, and what is its job?	The agent
Shared memory	What does it know, and how does it read it?	Own expertise, plus shared rules
Skills	What does it know how to do?	The agent
Tools	What can it reach, and what stops it?	The agent, under shared policy
Method	How does the work move, step by step?	The team
Limits	How far may it go on its own?	Set per agent, policy shared
Loop	How is its work checked and improved?	The team
The hive	How do agents hand work to each other?	The team
The nervous system	How does the team feel itself as one body?	The team (proposed)

How to read the chapters. Each chapter takes one part of the body. It opens with the idea in plain words, then walks through the studies drawn for it, one element at a time. After that comes what published research and practice say, an honest comparison with this model (where it matches, where it goes further, and what to watch for), and a checklist you can apply to your own agents.

About the evidence. The research was gathered in October 2026 from primary sources: engineering writing and documentation from Anthropic, OpenAI, Google, Meta and Microsoft, security guidance from OWASP and NIST, and peer reviewed papers and preprints. Numbers describe the settings of the study they come from, not every agent system. Numbered references are listed at the end. Where this guide says "the model", it means the agent layers model and its example files, built for a 17 agent marketing team. Take the ideas, not the wording.

Ten principles

1. Link, don't copy. Each fact lives in one place.
2. Separate who an agent is from what it does.
3. Keep agent files small. Push shared rules down into the team layer.
4. Give each agent its own expertise and shared rules from one source.
5. Write procedures down once and reuse them.
6. Give every input and output a template.
7. Set limits before adding power.
8. Check work with someone other than the author.
9. Turn every mistake into a rule, a test or a template change.
10. Keep readiness separate: a task can be ready, an agent can be ready, and permission to run is a third, separate gate.

one thread runs through every chapter: written rules guide an agent, but only enforcement controls it

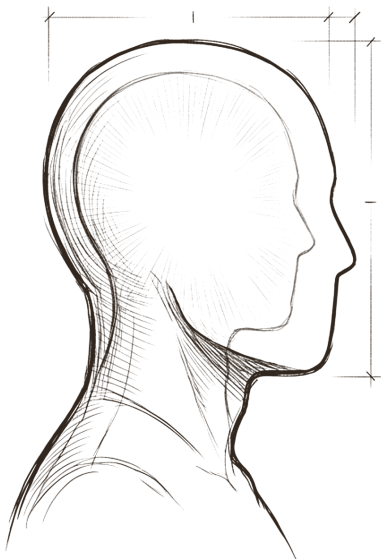
PART 1 OF 9

Identity

Who is this agent, and what is its job?

Every agent starts with two small files. The **role** (AGENT.md) is the job: its purpose, when to use it, what it takes in, what it hands back, how it works, who reviews it, and what it must never do. The **soul** (SOUL.md) is the character: voice, tone and the way it talks. The two are kept apart on purpose, and when they disagree, the role wins.

The split sounds cosmetic. It is not. It lets you change the job without touching the voice, keep one voice across many jobs, and see at a glance where a rule lives. It also draws the first and most important line in this whole model: describing an agent is not the same as giving it power.



the shell and the core

THE ROLE

The job, written as a specification

The outer shell of the head is the role. It answers practical questions a manager would ask on day one: what is this agent for, when should anyone call it, what must they give it, what will they get back, and where does its job end. In this model the role file follows a fixed template: *When to use, Inputs, Outputs, Working method, Collaboration and review and Boundaries*.

Precision here is not a nicety. In a study of 1,642 multi agent traces, agents disobeying their task specification was one of the most common failures, and improving role specifications alone raised one system's success rate by 9.4 points with the same model and prompt ^[1]. Anthropic's research team reached the same conclusion from practice: each agent needs an objective, an output format, guidance on tools and sources, and clear boundaries, or agents duplicate work and leave gaps ^[2]. Write the *does not do* list as carefully as the *does* list.

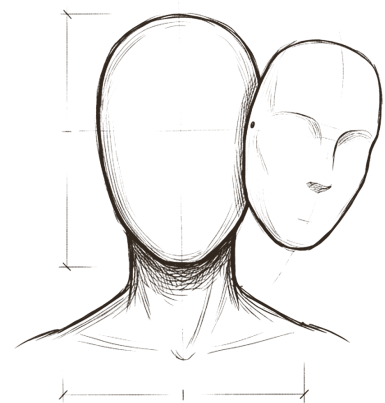
THE SOUL

Voice, not power

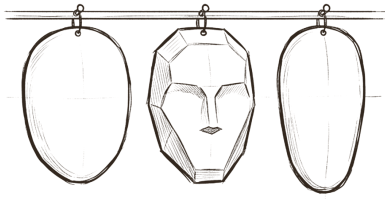
The face behind the mask is the soul: how the agent sounds, what it cares about in a sentence, how it greets you. A role in the system prompt measurably focuses behaviour and tone ^[3], so a well written soul is worth having.

What a soul cannot do is make the agent more accurate. Controlled tests on objective questions found that adding personas gave no reliable gain, and the effect of a given persona was close to random ^[4]. Persona descriptions can even carry hidden reasoning bias ^[5].

That is why the model says the soul **grants no power**: it controls voice only, never capability, authority or procedure. Keep every *do* and *do not* rule in the role or a shared guideline, and let the soul describe only how the agent speaks.



the mask



swap the role

SEPARATION

Same character, new job

Because the two files are separate, a soul can hang on the wall while the role changes. A team can give its research agent and its writing agent the same house voice, or move one character to a new job without rewriting how it talks.

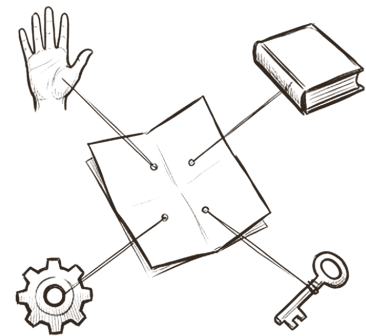
The conflict rule, *the role wins*, is a good design rule, but a written precedence rule is not something a model follows reliably. Claude Code's own documentation warns that when two instructions contradict each other, the model may pick one arbitrarily ^[6]. Personas also drift: instruction adherence decays measurably within eight rounds of conversation ^[7], and post training only loosely anchors a model to its default character ^[8, 9]. So keep the soul free of anything that could conflict with the role, and add a test where they do conflict, to prove the role wins.

LINKS, NOT COPIES

The identity file is an index

The role file names the skills, brains, workflows and tools the agent uses, and links to each one. It never copies them. Change a skill once and every agent that links to it gets the change; there is no second copy to fall out of date.

This matches the most common advice in vendor guidance: one shared base prompt with variables instead of many long separate prompts ^[10], and instruction files kept short, because longer files reduce adherence ^[6]. Links are also cheap. Only names and descriptions load until a part is needed ^[11]. One caution: a link list is not an allowlist. In Claude Code, an agent definition without a tools field inherits every available tool ^[12], so the safe default has to be set on purpose, in configuration, not in prose.



a small file

the job can change, the character stays

WHAT RESEARCH AND PRACTICE SAY

Vague roles are a measured cause of failure. Disobeying the task specification and drifting into another agent's role are named failure modes in multi agent systems; clearer role specs alone improved results ^[1, 2].

Personas change tone, not accuracy. A role in the prompt focuses tone ^[3], but personas do not improve performance on objective questions ^[4] and can introduce bias ^[5].

Personas drift over a conversation. Adherence decays within a few rounds, and certain conversations pull models away from their default character ^[7, 8].

Authority is enforced outside the text. OWASP's advice is to implement authorization in downstream systems rather than letting the model decide; Claude Code treats instruction files as context, not enforced configuration ^[13, 6].

One base prompt, short files. Shared rules belong in a single base prompt; each role adds only what is specific to it ^[10, 14].

An agent is also a security principal. Standards bodies now treat agents as things that need their own identity, authorization and audit trail ^[15, 16].

MATCHES PRACTICE

- The role and soul split puts the job in one file and tone in another; which is the only thing personas reliably change.
- "The soul grants no power" matches OWASP's complete mediation and Claude Code's warning that instruction files are not enforcement.
- The role template covers objective, inputs, outputs, tools and boundaries.
- A shared base prompt, *Working with the team*, is read by every role.

GOES FURTHER

- An explicit conflict rule, *the role wins*, which most vendor guides never state.
- Readiness, dispatch and permission are three separate fields (status, dispatch, action ceiling) instead of one on and off switch.
- Collaborators and reviewers are named inside the role, so handoff and review are part of the job description.

WATCH FOR

- *Role wins* is prose. It needs a test where role and soul conflict.
- The example soul repeats rules from its role (on evidence and rankings), which breaks *link, don't copy*.
- No test checks that an agent stays in role over a long session or under a persona based jailbreak.
- Named, pictured agents invite people to over trust them, a risk NIST lists under human and AI configuration ^[17].
- The role names a job but not the credential or audit identity the agent acts under.

Checklist

- | | |
|--|---|
| <ul style="list-style-type: none"> ☐ Write the role as a job spec: purpose, when to use, inputs, outputs, handoffs and an explicit <i>does not do</i> list. ☐ Keep the soul to voice and tone. Move every behaviour rule into the role or a shared guideline. ☐ State that the role wins, then add one test case where they conflict. ☐ Enforce permissions in configuration, hooks or approval gates, never in role or soul text. | <ul style="list-style-type: none"> ☐ Set each agent's tool list on purpose instead of inheriting the runtime default. ☐ Add a long session drift test and a persona jailbreak test to the agent's evals. ☐ Give each agent its own credential and audit identity. ☐ Keep each role file well under 200 lines; push shared rules into the base prompt. |
|--|---|

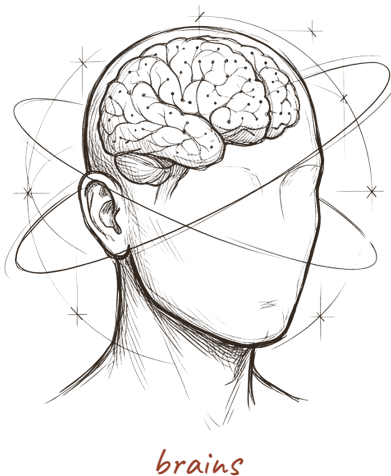
PART 2 OF 9

Shared memory

What does every agent know, and how does it read it?

Memory in this model has two homes. Each agent has its own **brains**: deep domain expertise kept with its sources, owned by that agent. The whole team shares **knowledge**: rules on naming, evidence, context and writing, written once and linked by everyone. Around both sit three habits: load context in a fixed order and ask when something is missing, read the smallest useful slice instead of the whole library, and label every claim with where it came from.

Memory is the part of an agent that compounds. It is also the part that an attacker most wants to poison, and the part that silently decays when two writers edit the same file. This chapter treats it with the care of a shared database, because that is what it is.



OWN EXPERTISE

Brains and the order of context

A brain is a folder of notes about one domain, such as technical SEO, kept with the sources behind each note. It belongs to one agent, and only that agent's owner changes it. Research on language agents calls this *semantic memory*: an agent's knowledge about the world ^[18].

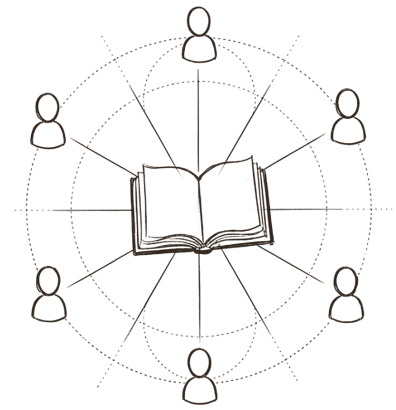
When a task starts, context loads in a fixed order: project, then client, then brand, then audience. If a required piece is missing, the agent asks. It never guesses a brand or an audience. This is not caution for its own sake. Models guess partly because training and scoring reward guessing over saying *I don't know* ^[19], and *proceeding with wrong assumptions instead of seeking clarification* shows up in 6.8% of analysed multi agent traces ^[1]. Abstention is a recognised safety tool ^[20]; the model makes it the default.

SHARED RULES

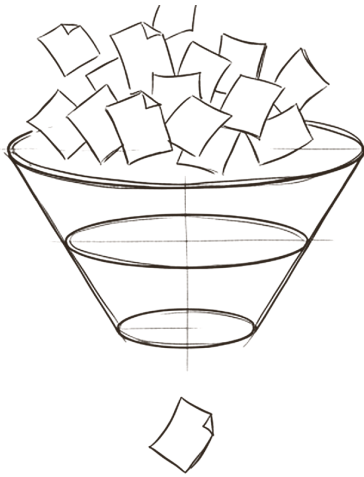
Knowledge, and one writer per file

The open book in the centre is shared knowledge: the guidelines every agent follows, from how files are named to what counts as evidence. They are written once and linked from every role. This is what research calls *procedural memory*, the rules that drive behaviour, and changing it is singled out as the riskiest kind of learning, because it can introduce bugs or let an agent subvert its designers' intentions ^[18]. Treat a guideline change like a code change: reviewed and tested.

Writes follow one rule: **agents return drafts; one owner changes each shared file**. The owner reads the file, records its hash, applies the change, and refuses if the file changed in between. Parallel agents acting on different pictures of the same state is a known source of failure; as one practitioner put it, *actions carry implicit decisions, and conflicting decisions carry bad results* ^[21].



written once, linked by all



three tiers

JUST IN TIME

Read the map, then a bundle, then only what the task needs

The agent never loads the whole library. It reads in three tiers: first a map with names and one line descriptions, then a small resolved bundle for the task, then only the extra notes the task actually asks for, under a fixed cap.

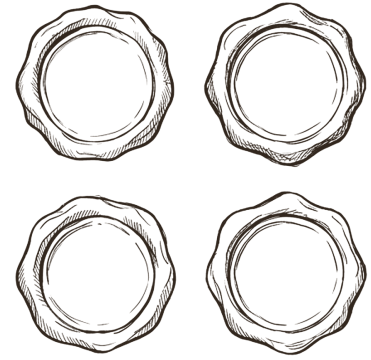
This is the pattern Anthropic describes as context engineering: find the smallest set of high signal tokens, hold lightweight references such as file paths, and load content at runtime ^[14]. The reason is measurable. Across 18 models, performance fell as input grew longer ^[22], and facts placed in the middle of a long context are used worse than facts at the start or end ^[23]. Two refinements follow: put the task and its constraints at the start or end of the bundle, never in the middle, and test retrieval itself, because a word cap controls volume, not whether the right notes were found. For a small collection, loading everything with caching can be fine ^[24]; the rule is about relevance, not a ban.

EVIDENCE

Every claim says where it came from

Each stored fact carries one of four labels. **RAW** is a kept source. **FETCH** means the agent read it directly in this run. **SEARCH** means it was only found by search and not checked. **INFER** means it was reasoned from other evidence. Search is a discovery aid, never evidence by itself. When evidence is missing, the honest answer is *no data*, which beats any guess.

Most guidance says *cite your sources* and stops there. Retrieval augmented generation was invented partly because models cannot show where a claim came from [25], NIST names confident false output as a core risk [17], and research on shared agent memory attaches fixed provenance to every memory item [26]. The four labels go further than any of these. One caution: a label is also a claim. The strongest version checks every FETCH label against the run's actual fetch log.



raw · fetch · search · infer

link, don't copy: each fact lives in one place

WHAT RESEARCH AND PRACTICE SAY

Separate the kinds of memory. Working, episodic, semantic and procedural memory carry different risks; writing procedural memory is the riskiest [18].

Load the least context that does the job. Smaller, relevant context beats long context; long inputs degrade results and bury facts in the middle [14, 22, 23].

Keep provenance attached to every fact. Provenance and confabulation are core risks; shared memory systems attach provenance to each item [25, 17, 26].

Reward *I don't know*. Models guess because scoring rewards it; abstention reduces hallucination when evaluation rewards it too [19, 20].

Memory is an attack surface, and poison persists. Poisoning under 0.1% of a memory store reached over 80% attack success [27]; attackers can plant records just by talking to an agent [28]; OWASP lists memory and context poisoning as an agentic risk [29].

Coordinate writes. Agents acting on different pictures of shared state make conflicting decisions; shared artifacts with clear ownership help [21, 2].

MATCHES PRACTICE

- Brains and shared knowledge match the semantic and procedural split in agent research.
- The three retrieval tiers are a concrete version of *lightweight references, loaded just in time*.
- "Ask, never guess" and the *no data* and *needs input* states match the abstention research.
- One owner per shared file, with a hash check before applying, matches the advice on coordinated writes.

GOES FURTHER

- Splitting FETCH from SEARCH and treating search as discovery, not evidence, is stricter than any vendor guidance found.
- Provenance is kept separate from evidence tier, verdict and confidence.
- Freshness horizons block stale sources instead of only expiring unused files.

WATCH FOR

- No write time check on new brain notes for embedded instructions, and no documented rollback for a poisoned note.
- Episodic memory, *what happened last time*, has no named home or writer.
- Word caps control volume, not relevance; nothing tests that the right notes were retrieved.
- If *no data* is rewarded and never balanced, agents may stall on answerable tasks. Track the rate.

Checklist

- ☐ Sort every stored item into domain facts, shared rules or past events, each with one owner and one write path.
- ☐ Attach provenance when a fact is written: source, how it was obtained, date and freshness horizon.
- ☐ Start each task from a map and a small bundle; put the task at the start or end of context.
- ☐ Make the agent ask when a decision is missing and return *no data* when evidence is missing.
- ☐ Scan new notes for embedded instructions before they enter memory, and keep a rollback point.
- ☐ Allow one writer per shared file; check the hash before applying any change.
- ☐ Review and test every change to shared rules as you would a code change.
- ☐ Test retrieval: for sample tasks, check the bundle contains the notes the task needs.

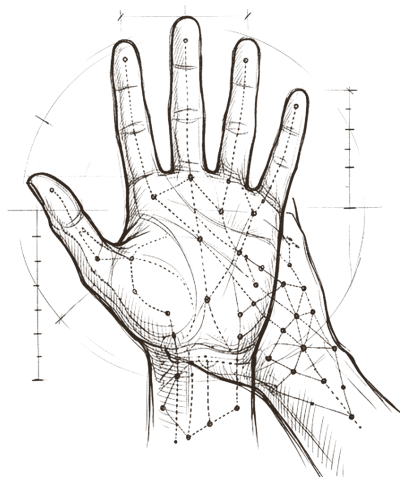
PART 3 OF 9

Skills

What does the agent know how to do?

A skill is a practiced ability written down as instructions: an SEO audit, a competitor teardown, a weekly report. It has an outcome, a trigger, inputs, steps, one output and boundaries. It holds no client facts and no brand voice, because those live in memory and change from job to job. And it grants no access: knowing how to do something is not permission to do it.

Skills are where an agent's quality is most visible and most testable. They are also, by design, instructions, which makes them the easiest place to hide an instruction that should not be there.



what it knows

THE ABILITY

A packaged procedure with a clear trigger

The open palm is the skill itself: know-how, written down. Today the common format is a folder with a SKILL.md file and optional linked files. At startup only the name and a short description load for each skill; the body and extras load when the skill is used ^[11].

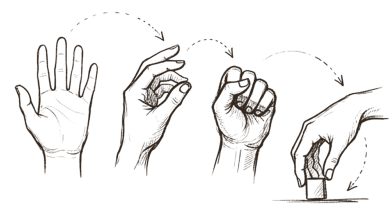
That makes the description the trigger. The open specification limits it to 1,024 characters and says it must describe what the skill does and when to use it ^[30]. Anthropic's guidance calls it critical for selection when there are a hundred or more skills to choose from ^[31]. Write it first, in the third person, with the words a person would actually use when they need it.

THE METHOD

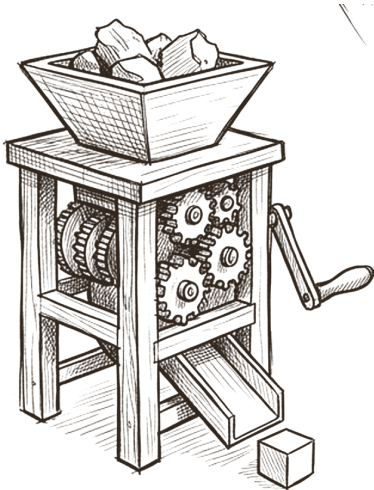
Same motions, same order, every time

A skill is a short sequence of steps, each one an action with a visible result. Good skills come from real procedures, not imagination: OpenAI recommends building routines from existing operating procedures and policy documents, with a branch for when required information is missing ^[10]. Anthropic adds a useful idea, *degrees of freedom*: loose guidance where many approaches work, exact scripts where a step is fragile or must be repeatable ^[31]. The research ancestor of today's skill folders, Voyager, kept a growing library of executable code and improved it from execution feedback ^[32].

Keep it focused. The first large benchmark of skills found that curated skills raised the average pass rate from 33.9% to 50.5%, and that focused skills with at most three modules outperformed larger, exhaustive bundles ^[33]. Under 500 lines for the body, and references one level deep, is the vendor ceiling ^[31].



the steps



in and out

THE CONTRACT

Inputs in, one output out, nothing invented

The mill takes inputs at the top and drops one finished output at the bottom. What goes in is named: the site, the brief, the audience. What comes out has a fixed shape. And two failure states are named too. A **missing decision** returns *needs input*. **Missing evidence** returns *no data*. The skill never fills either gap with a guess.

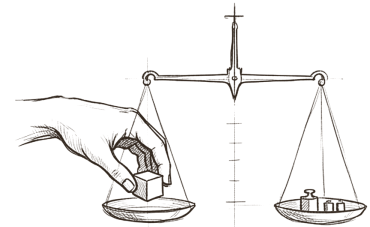
The mill has no memory of its own. A skill holds no client facts and no brand voice; they arrive as inputs from memory. This keeps the skill reusable across clients and free of dated information, which vendor guidance specifically warns against putting in a skill body [31]. It also means one tested skill serves the whole team.

EVIDENCE

Weighed against a baseline before it gets real work

A skill is scored on test briefs before it is trusted. This is the rule the research supports most strongly. Anthropic's guidance says to create evaluations before writing extensive documentation, with at least three of them [31], and Claude Code's documentation adds a sharp warning: seeing a skill trigger tells you it was found, not that it did what you intended. Run realistic prompts in a fresh session with the skill on, and again with it off [34].

The reason is that skills can make things worse. In the same benchmark, 16 of 84 tasks scored lower with a skill than without one, and skills the model wrote for itself gave no benefit on average [33]. Test two things: that the skill triggers on the right requests and stays quiet on the wrong ones, and that its output actually beats the baseline.



tested first

a skill is know-how, not permission: using it grants no access

WHAT RESEARCH AND PRACTICE SAY

The description decides when a skill is used. Only name and description load up front; the description must say what and when [30, 31, 11].

Focused beats exhaustive. Curated skills improved pass rates by 16.6 points on average; skills with at most three modules did best [33].

Write from real procedures; give code the exact steps. Routines come from existing procedures, with a branch for missing information; fragile steps become scripts [10, 31].

Test against a baseline. Run with and without the skill. Some tasks get worse, and self written skills gave no average benefit [34, 33].

Skills are a supply chain. 26.1% of 31,132 public skills had at least one vulnerability, and skills bundling scripts were 2.12 times as likely to ^[35]; skills enable trivially simple prompt injections ^[36]. Install only from trusted sources and audit first ^[11].

Keep permission out of the skill. OWASP names excessive functionality, permissions and autonomy as root causes ^[13]. Note that in Claude Code a skill's allowed tools field does grant permission for that turn ^[34].

MATCHES PRACTICE

- The skill template (outcome, use when, inputs, output, boundaries) follows the specification and best practices.
- *Needs input* and *no data* turn the advice on missing information into a named contract.
- Keeping client data and voice out of the skill keeps it reusable.
- Ported source text is treated as untrusted and its hash recorded.

GOES FURTHER

- "This skill grants no external authority" is stricter than at least one popular runtime, where a skill can pre approve its own tools.
- Three distinct states (*needs input*, *blocked*, *no data*) give more precise failure reports than any vendor guide reviewed.
- Each skill records its source, version and port decision.

WATCH FOR

- The example SEO audit skill is active while its evaluation flag is false, and its text calls evaluation optional.
- Evals check output, not whether the skill triggers on the right requests.
- The example body is mostly provenance and legal boundary text; the method itself is five steps.
- No step reviews bundled scripts for network calls or credential access before a skill goes live.

Checklist

- | | |
|--|---|
| ☐ Write the description first: what it does and when to use it, under 1,024 characters. | ☐ Use scripts for exact steps, and say whether the agent runs or only reads each one. |
| ☐ Keep the body short and mostly procedure; move history and reference into linked files. | ☐ Before marking a skill active, run three or more realistic tasks with and without it. |
| ☐ Name what happens on a missing decision and on missing evidence. Never guess. | ☐ Install skills only from trusted sources, and read every bundled file first. |
| ☐ Keep client data, brand voice and dated facts out of the skill. | ☐ Grant tools through permission settings, not skill fields, and check what your runtime does. |

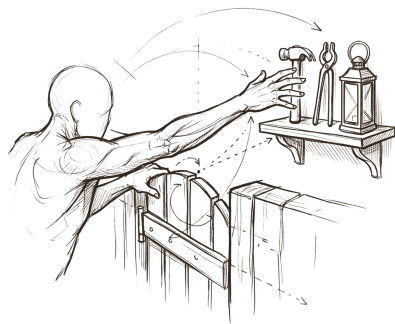
PART 4 OF 9

Tools

What can the agent reach, and what stops it?

Tools are the outside services and data an agent can reach: a search API, a crawler, a CMS, a payment account. The model's rules are short. Capability is not authority. The default permission is none. Each tool lists which agents may use it and how far. Spending and writing each need their own human yes. A credential is stored by name, never by value. Budgets and rate limits are set before the first call. And every tool notes whether its actions can be undone.

Every one of those rules has strong support in security research going back fifty years. The new risk that agents add is not any single tool. It is what one agent holds at the same time.



the reach

CAPABILITY

Being able to reach is not permission to reach

The figure can touch everything on the shelf. That is capability. Whether it is allowed to pick anything up is a separate question, and the answer starts at **none**. Each tool is granted to one agent at a time, for named functions and a named scope.

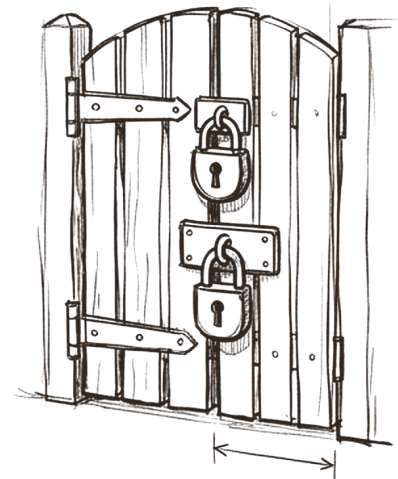
This is the oldest rule in computer security. In 1975 Saltzer and Schroeder wrote that the default situation should be lack of access, with the protection scheme naming when access is permitted ^[37]. OWASP names excessive functionality, excessive permissions and excessive autonomy as the three root causes of excessive agency ^[13]. Google adds that agents must be prevented from escalating their own privileges beyond pre authorised scopes ^[38]. And fewer tools are not only safer: Anthropic notes that more tools don't always lead to better outcomes ^[39].

POLICY**Each tool lists who may pass, and how far**

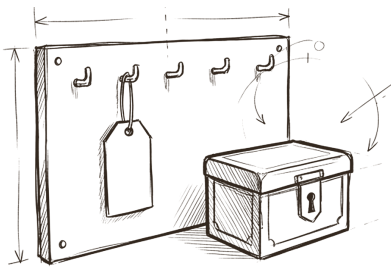
Every tool has a note that works like a gate: which agents may use it, for which functions, at what level. Two actions get their own lock.

Spending money and **writing** to anything outside each need a separate human yes, even for an agent already allowed through the gate.

The research adds two things. First, rate each tool's risk using read versus write access, reversibility, required permissions and financial impact, and let that rating decide the gate ^[10]. MCP builds this into its tool hints with cautious defaults: unless told otherwise, a client assumes a tool may change things, may be destructive and may reach the outside world ^[40]. Second, the gate must be enforced outside the model. Claude Code's documentation is explicit that permission rules are enforced by the harness, not by the model ^[41], and Google places a deterministic policy engine outside the model's reasoning to check each action before it runs ^[38].



the gate, two locks



the name, never the key

CREDENTIALS**Store the label, keep the key out of reach**

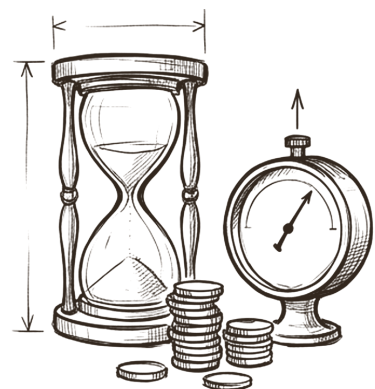
The tag on the hook carries the credential's name. The key itself stays locked in the box, in a secrets store the agent never reads directly. Notes, logs and transcripts hold the name only.

OWASP's guidance is the same: secrets live in a secrets manager, carry the minimum privileges, are rotated, and never appear in logs ^[42]. Two refinements matter for teams. The agentic Top 10 asks for short lived, narrowly scoped tokens per task and per agent ^[16], because one shared credential blurs who did what. And MCP forbids *token passthrough*: a server must not accept tokens that were not issued for it ^[43].

LIMITS ON USE**Spend, rate and time are capped before the first call**

Before an agent makes its first live call, the tool note fixes a spend cap, a rate limit, a timeout and what happens automatically when a limit is hit. In this model each paid call also needs a fresh human approval of an exact, costed plan with a maximum charge.

OWASP names the risk *denial of wallet*: running up cost on pay per use services ^[44]. MCP requires servers to rate limit tool calls and tells clients to set timeouts ^[45]. The stakes grow with teams: Anthropic measured multi agent systems using about 15 times the tokens of a chat ^[2]. The same note records whether the tool's actions can be undone, because reversibility is one of the main factors in how much oversight a tool needs ^[10].



budget first

reaching is easy. permission to reach is the design

WHAT RESEARCH AND PRACTICE SAY

Least privilege, by default. Grant the fewest, narrowest tools; deny by default ^[37, 13, 38].

Rate each tool's risk and let the rating decide the gate. Read or write, reversibility, permissions and financial impact set the level of oversight ^[10, 40].

Enforce on every call, outside the model. Rules in a prompt shape intent; only the harness or a policy engine controls what actually runs ^[41, 38].

Credentials: scoped, short lived, referenced, never logged. Per agent, per task tokens; no passthrough ^[42, 16, 43].

Budgets before the first call. Rate limits, quotas, timeouts and automatic cut offs prevent denial of wallet ^[44, 45].

Watch combinations, not just tools. Text returned by tools can carry instructions ^[46]. An agent holding private data, untrusted content and an outbound channel at once is the *lethal trifecta* ^[47]; Meta's Rule of Two says a session should hold no more than two of the three ^[48].

MATCHES PRACTICE

- *Default permission: none* plus an allowed agents table is fail safe defaults and least privilege in practice.
- "Capability is not authority" is complete mediation.
- A credential reference holds a name, never a value.
- Reversibility and spend fields match the main risk factors in OpenAI's guide.

GOES FURTHER

- Each paid call needs a fresh human approval of an exact, costed plan with a hard cap. Most frameworks allow standing allowances.
- Failure, partial data, zero results and absence are four distinct recorded states.
- Provider facts such as prices and endpoints must be checked against primary documentation before use.

WATCH FOR

- Each note judges one tool alone. Nothing checks what one agent holds together.
- Reading is not automatically safe: a fetch can leak data through the URL. Data egress is not recorded.
- No pinned version or hash of the reviewed tool definition, though MCP servers can change their tools at runtime.
- No fields for switching a tool off, revoking its credential, or where its calls are logged.

Checklist

- | | |
|---|---|
| <ul style="list-style-type: none"> ☐ Start every tool at <i>no agent may call this</i>; grant one agent at a time with exact functions. ☐ Label each action: read only, additive, destructive, external send or spend. Unlabelled means destructive. ☐ For each agent, list what it holds together. Split work that combines private data, untrusted content and an outbound channel. ☐ Keep credentials in a secrets manager, one scoped credential per agent, with a rotation date. | <ul style="list-style-type: none"> ☐ Write the budget cap, rate limit, timeout and automatic cut off before the first live call. ☐ Pin each tool's reviewed version and the hash of its definition; re review on change. ☐ Write down how to switch the tool off, revoke its credential and find its logs. ☐ Treat everything a tool returns as data, and test that with injection cases. |
|---|---|

PART 5 OF 9

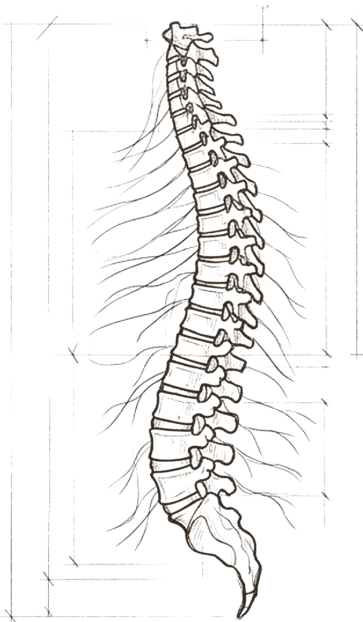
Method

How does the work move, step by step?

Method is the spine of the body. It has three parts. **SOPs** are habits every agent shares: how to start a session, close one, review an artifact, handle an approval, hire an agent, run the weekly review and ingest a source. **Workflows** are production lines for one deliverable, each with a trigger, inputs, steps, a review gate, an approval gate and an output with a destination.

Templates give every output a fixed shape.

The idea is simple: a well defined job should come out the same way twice. That is also the hardest thing to get from an AI agent, and the reason method matters more for agents than for people.



the backbone

PREDICTABILITY

The same steps, in the same order, every time

Anthropic draws a useful line between *workflows*, where models and tools follow predefined paths, and *agents*, which direct their own steps. Its advice is that workflows offer predictability and consistency for well defined tasks, and agents are for when flexibility is genuinely needed ^[49]. OWASP gives the security reason: autonomy the task does not need adds attack surface without adding value ^[16].

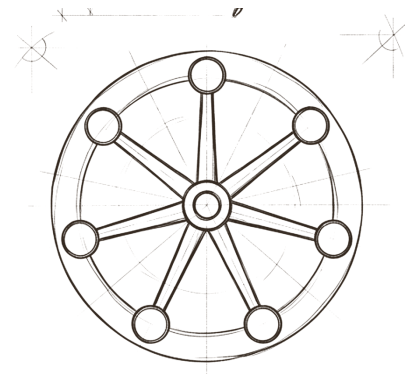
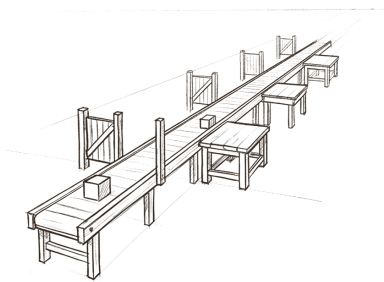
Consistency is a real problem, not a theoretical one. In 2024, the best tool using agents on the τ -bench benchmark succeeded on under half of tasks, and in the retail domain all eight repeated attempts at a task succeeded less than a quarter of the time ^[50]. Models have improved since, but the lesson holds: measure how often a job succeeds over repeated runs, and give well defined jobs a fixed backbone.

SOPS

Habits every agent shares

Each spoke of the wheel is a standard operating procedure every agent follows: start a session, close a session, review an artifact, handle an approval, hire a new agent, run the weekly review and ingest a source. Because every agent opens and closes the same way, any agent can pick up where another stopped.

This mirrors what Anthropic found running long agent sessions: a fixed opening (check the working directory, read the progress notes and history, pick one item) and a fixed close (commit, update progress, leave a clean state), with one feature per session described as critical [51]. Turning procedures into explicit prompt sequences is also the core idea of MetaGPT, one of the most cited multi agent papers [52]. One detail in the model is worth copying everywhere: never pick the session to resume by latest modification time. Read the explicit session record.

*habits**the workflow*

PRODUCTION LINE

One line for one deliverable

A workflow is a production line for one kind of output, such as an audit or a content brief. It has a trigger, required inputs, numbered steps, a **review gate** where someone other than the author checks the work, an **approval gate** where a human says yes, and a named output with a destination.

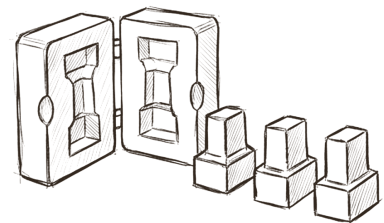
This is Anthropic's prompt chaining pattern with checks between steps, plus a human checkpoint [49]. What the research adds is a stopping rule. Anthropic advises stopping conditions such as a maximum number of iterations, and OpenAI names exceeding failure thresholds as one of two main triggers for handing control to a human [49, 10]. Errors compound in agent systems [2], so every workflow should carry a maximum number of attempts and a named person to escalate to.

FIXED SHAPES

One mold for every output

Every deliverable comes from a template: the same sections, in the same order, with the same fields. Outputs become easy to compare across runs and easy to check against criteria. Handoffs between agents use templates too, which is what makes a team possible.

Anthropic found that each delegated task needs an objective, an output format, guidance on tools and sources, and clear boundaries [2]. Its harness team kept status in JSON because the model was less likely to overwrite JSON by mistake than Markdown [51], and schema constrained output now guarantees valid structure in most cases [53]. So for fields that decide something (status, verdict, cost, target), use a small validated schema next to the prose.

*templates*

an SOP is a habit every agent shares. a workflow is a line for one output

WHAT RESEARCH AND PRACTICE SAY

Prefer a fixed workflow for well defined work. Add agent freedom only where it pays [49, 10].

Build steps from real procedures. Each step one action with one output, with a branch for missing information [10, 52].

Check between steps, and stop on purpose. Gates on intermediate steps, a maximum number of iterations, escalation on repeated failure [49, 10].

Open and close every session the same way. Read the progress record first; leave a clean, written state at the end [51].

Give handoffs and outputs a checkable shape. Vague briefs cause duplicated and missed work; structured fields resist accidental overwrites [2, 53].

Context is a finite budget. Long procedures are followed unevenly; keep a short core and load detail when needed [14, 11].

MATCHES PRACTICE

- The SOP and workflow split matches OpenAI's routines and MetaGPT's SOP and assembly line framing.
- The workflow template is prompt chaining with gates, plus a human checkpoint.
- Session open and close SOPs mirror Anthropic's long running harness closely.
- Templates for every deliverable match the fixed output format advice.

GOES FURTHER

- Review binds the exact bytes by hash; any change needs a delta review.
- "A new role label in the same execution is not independent review."
- Missing review stays unavailable and is never silently treated as a pass.
- Checker defects are kept separate from artifact defects, and failed runs are kept, not rerun until they pass.

WATCH FOR

- Procedures are long for a model to follow; the key rule can get lost. A five to seven step core would help.
- Several steps are marked script not built, and every example workflow is still unevaluated.
- The workflow template has no maximum attempts or escalation owner.
- No consistency test: run the same brief several times before calling a workflow reliable.

Checklist

- ☐ For each recurring deliverable, decide: fixed workflow or agent judgment. Default to the workflow.
- ☐ Write each SOP as five to seven numbered steps with one visible output each; link the detail.
- ☐ Give every workflow a trigger, inputs, steps, review gate, approval gate, destination, maximum attempts and escalation owner.
- ☐ Open each session from the explicit record; close it with what changed, what was checked and the next action.
- ☐ Brief every handoff with objective, output format, allowed tools and sources, and boundaries.
- ☐ Keep deciding fields (status, verdict, cost) in a validated structure.
- ☐ Review against criteria frozen before the work began; missing review means not reviewed.
- ☐ Run a workflow several times on the same brief before calling it reliable.

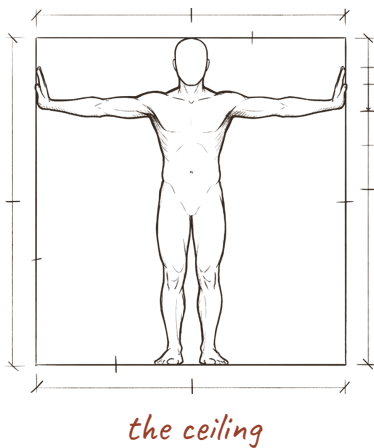
PART 6 OF 9

Limits

How far may the agent go on its own?

Limits are the square around the figure. There are three controls. The **action ceiling** is the most an agent may do without asking. The **dispatch switch** decides whether it can be called at all. The **approval policy** has three rungs: private, undoable work may go ahead; public, undoable work would run under a standing approval (none exist yet); and anything costly, public or final needs a fresh human decision, every time.

Two further rules make the ladder hold. An approval covers one exact item, and any change to its content breaks the seal. And a pending request, a plan or an old approval never authorises anything. Better evidence never grants permission.



ACTION CEILING

The most it may do without asking

The figure stands inside a square it cannot step out of. That square is set before the agent is given any power, and every new agent starts at **none**. Raising it is a deliberate decision by the owner.

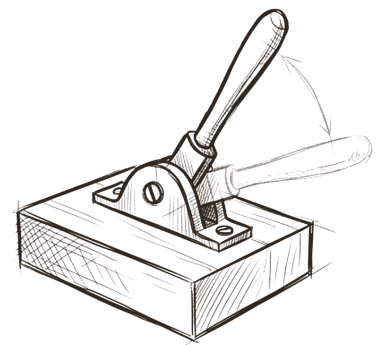
Research frames this the same way. Feng, McDonald and Zhang argue that autonomy should be treated as a deliberate design decision, separate from capability, and describe five levels by the role the human plays: operator, collaborator, consultant, approver and observer ^[54]. Google requires maximum permission levels to be definable and enforceable ^[38]. What the model can add is an evidence standard for raising a ceiling: specific eval pass rates over repeated runs, injection tests passed, and incident history, so the decision is repeatable rather than a feeling.

DISPATCH

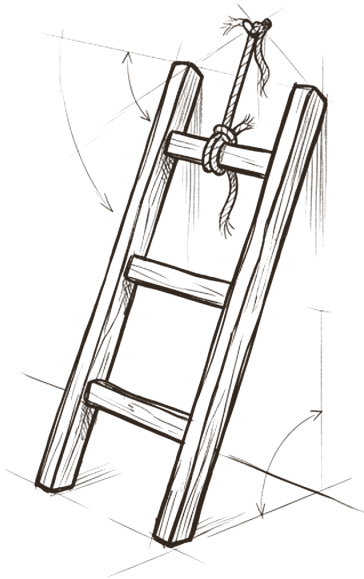
Can it be called at all?

A ready agent is not a running agent. The dispatch switch is a separate control, off until a person turns it on, and it can be thrown back at any moment. In the example team, all 17 roles are planned and disabled.

The EU AI Act asks for exactly this in high risk systems: the ability to interrupt the system through a stop button ^[55]. OWASP asks for a kill switch that can disable a tool everywhere at once ^[16], and Google says users must be able to inspect and revoke any authority delegated to an agent ^[38]. Keeping readiness, dispatch and permission as three separate gates is cleaner than most products, which merge them into one *on* state.



the switch



three rungs

APPROVAL POLICY

Private and undoable, public, final

The ladder sorts every action. **Rung 1**, private and reversible work, goes ahead. **Rung 2**, public but reversible work, would run under a standing approval with a fixed scope; none exist yet. **Rung 3** is publishing, sending, spending, deploying, changing accounts or permissions, irreversible deletes, commits and pushes: each needs a fresh human yes.

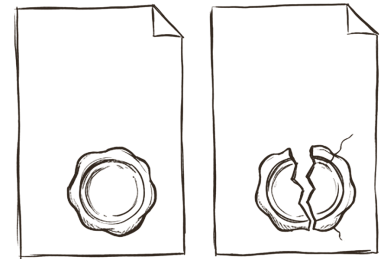
This matches the consensus. OpenAI says sensitive, irreversible or high stakes actions should trigger human oversight until confidence grows ^[10], and Google calls for explicit human confirmation before critical or irreversible actions ^[38]. The research also names the ladder's weak point: people stop reading. Anthropic reports users accept 93% of manual permission prompts ^[56], sandboxing cut prompts by 84% ^[57], and a review of 41 government policies found people unable to perform the oversight they were asked to do ^[58]. With rung 2 empty, every public step rests on human attention, so make each request quick to judge, and plan rung 2 deliberately.

BINDING

A yes covers one item. Change it and the seal breaks

An approval is bound to a hash of the exact content, target, scope, cost and expiry. Change one word and the seal breaks; the work needs a new yes. A pending note, a plan, a batch label or a previous approval never authorises execution, and an approved request is never proof that the action happened.

This is Saltzer and Schroeder's *complete mediation*: every access to every object must be checked for authority, and remembering an earlier check should be examined sceptically ^[37]. OWASP asks agents to show a pre execution plan or dry run diff before approval and to bind tokens to a signed intent ^[16]. The last rule, *better evidence never grants permission*, is the defence against persuasion: once an agent has read untrusted input, that input must not be able to trigger consequential actions ^[59].



an exact yes

a request is not a yes. better evidence never grants permission

WHAT RESEARCH AND PRACTICE SAY

Autonomy is a design choice, set per agent. Separate from capability, with levels defined by the human's role ^[54, 38].

A human decides the irreversible, the costly and the external. Consistent across OpenAI, Google and OWASP ^[10, 38, 16].

Enforce limits deterministically, outside the model. Model based defences are non deterministic and cannot give guarantees ^[38, 37].

Bind each approval to its exact effect. Show a plan or diff, and let the approval carry over to nothing else ^[16, 43].

Design for attention, not volume. Approval fatigue is documented: 93% of prompts accepted, people blindly confirming warnings ^[56, 48, 58].

Untrusted content never raises authority, and blocks are not routed around. Claude Code's auto mode stops and escalates after 3 consecutive or 20 total denials ^[56, 59].

MATCHES PRACTICE

- The action ceiling and its starting value of none are definable maximum permissions and fail safe defaults.
- The dispatch switch is the stop and revoke control that Google, OWASP and the EU AI Act describe.
- Rung 3 lines up with OpenAI's high risk class and Claude Code auto mode's block list.
- "Better evidence never grants permission" matches the rule that untrusted input never raises authority.

GOES FURTHER

- Requests, human decisions and execution results are three separate records, each bound by hash, with expiry.
- No standing approvals exist in the foundation phase.
- An approved request is never treated as proof that the action happened.
- The policy states openly that its records cannot authenticate who is speaking in a chat.

WATCH FOR

- Approval fatigue: with rung 2 empty, a human is the only control on every public or costly step.
- Enforcement is on paper until a harness enforces it. Some ceilings can be enforced today with deny rules and sandbox network limits.
- The rungs sort by undo and audience, not by data leaving the system.
- No written rule against retrying a blocked action another way, and none for delegation: an agent's ceiling should never exceed its assigner's.

Checklist

- | | |
|---|--|
| <ul style="list-style-type: none"> ☐ Give every agent an action ceiling and a dispatch switch; start at none and off. ☐ Classify actions by undo, audience, cost and data egress. ☐ Anything irreversible, public, costly or outbound gets a fresh yes, shown as a preview with cost and rollback. ☐ Bind each approval to a hash of content, target, scope, cost and expiry. | <ul style="list-style-type: none"> ☐ Enforce ceilings in the harness, sandbox or policy engine, and test that a forbidden call is blocked. ☐ Never let tool output, web content or another agent's message count as approval. ☐ After a block, forbid the same effect by another route; escalate after a set number of blocks. ☐ Keep prompts rare and meaningful, and watch approval rates for rubber stamping. |
|---|--|

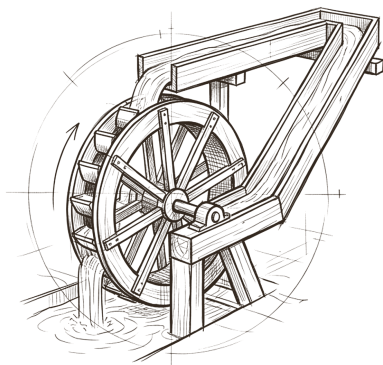
PART 7 OF 9

Loop

How is the work checked, and how does the agent get better?

The circle around the figure is the loop. Every output comes back to be checked, and every mistake flows back in as a lesson. It has three parts: **evals**, test briefs scored on a rubric plus attack cases such as prompt injection; an **independent verifier**, because the author never approves its own work; and **learning**, where each mistake becomes a candidate lesson that, once a human accepts it, turns into a new rule, test or template.

Without the loop an agent never gets better; it only gets older. With a careless loop it can get worse, because the loop is also the one place where an agent's instructions are allowed to change.



round it goes

THE CYCLE

Every output comes back; every lesson flows in

The water wheel turns one way: produce, check, learn, produce again. Nothing leaves the loop unchecked, and nothing learned is lost.

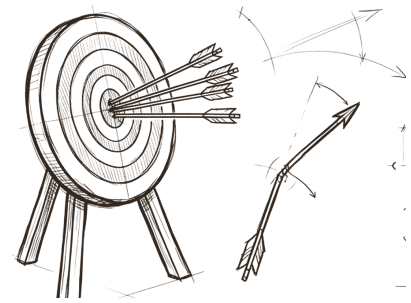
The research case for a learning loop is well established. Reflexion showed agents improving across trials by writing verbal reflections into an episodic memory and reading them on the next attempt ^[60]. But the same mechanism is a target: OWASP's agentic Top 10 lists memory and context poisoning among its risks ^[29]. Whatever writes lessons must be controlled as tightly as whatever writes rules. The model's answer is strict and, as far as the research found, unusual: lessons come only from the human's own words, are reviewed by someone other than the author, and take effect only when the human accepts them. An agent cannot write itself a new rule.

EVALS

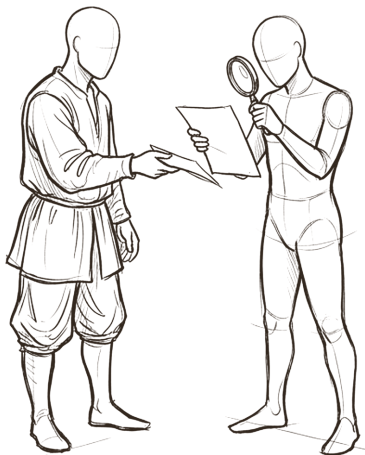
Test briefs, a rubric, and attacks that move

Each agent is scored on test briefs with frozen criteria, on a 0 to 4 rubric across eight dimensions, with hard gates for anything a rule can decide. A separate set of attack cases checks that the agent resists prompt injection.

Anthropic's guidance gives the practical shape. Start with 20 to 50 tasks drawn from real failures. Write each so that two domain experts would independently reach the same pass or fail verdict, and keep a reference answer. Keep two suites: a **capability** suite that starts with a low pass rate, and a **regression** suite that must stay near 100% ^[61]. Prefer deterministic checks; model judges need calibration against humans ^[61, 62] and carry known biases toward position, length and their own outputs ^[63, 64]. Two gaps follow. Agents are not deterministic, so run critical tasks several times and report how many runs passed, not just whether one did. And a fixed list of attack cases tells you about yesterday's attacks: adaptive attacks bypassed 12 published defences, most with success above 90% ^[65].



an attack, caught



fresh eyes

VERIFIER

The author never approves its own work

A separate verifier checks the exact version of the work, with no access to the author's own verdict. A pass freezes that version by hash. Change it after review, and it is reviewed again.

This is the best supported rule in the whole model. Anthropic found that agents reliably skew positive when grading their own work, and that separating the agent doing the work from the agent judging it is a strong lever ^[66]. Models recognise and favour their own outputs ^[64], and self correction without outside feedback does not reliably fix reasoning ^[67]. Still, a verifier is not a silver bullet: in the MAST study, verifiers often performed only superficial checks ^[1]. Test the verifier itself by planting known mistakes and counting how many it catches, and prefer a different model, or at least a fresh context, so the two roles do not share blind spots. Freezing a pass to the exact version mirrors GitHub's option to dismiss stale approvals when new commits arrive ^[68], applied to AI output.

LEARNING

Each mistake becomes a brick in the wall

When something goes wrong, the loop writes a candidate lesson. Once accepted, it becomes one of three things: a new **rule** in shared knowledge, a new **test** in the eval suite, or a change to a **template**. Each accepted lesson carries watch phrases and a re check date 90 days out.

The research suggests one tightening: every accepted lesson should also add a regression test ^[61], so that *turn every mistake into a rule, a test or a template* reliably produces tests, and the same mistake cannot quietly return. Keep the loop light enough to use. Both Anthropic and OpenAI stress starting simple ^[49, 10]; a loop so heavy that people skip it protects nothing. A minimum viable loop is a short test set, one independent check and a lessons file only the human can approve.



without the loop, an agent never gets better

WHAT RESEARCH AND PRACTICE SAY

Build evals from real failures, in two suites. Start with 20 to 50 tasks; separate capability from regression ^[61].

Run more than once. $\text{pass}@k$ when one success matters, pass^k when consistency matters ^[61, 50].

Deterministic checks first; calibrate model judges. Judges show position, verbosity and self preference bias ^[62, 63, 64].

Never let the author approve its own work. Self grading skews positive; separation is a strong lever ^[66, 67].

A verifier is necessary but not sufficient. Superficial verification is a common multi agent failure ^[1].

Attack cases must adapt. Adaptive attacks beat defences that reported near zero success on fixed tests ^[65, 69].

MATCHES PRACTICE

- Test briefs with frozen criteria, a rubric with hard gates and injection cases follow Anthropic's task and grader structure.
- "The author never approves its own work" is directly supported by research.
- Calibration guidance (held out examples, human labels, A then B and B then A ordering) matches the judge bias findings.

GOES FURTHER

- A pass freezes the exact bytes by hash; any change needs a delta review.
- Four honest outcomes: pass, fail, inconclusive and not run. Inconclusive never counts as a pass.
- Lessons come only from the human's own words, reviewed by a non author, accepted by the human, with a 90 day re check. No mainstream framework found has an equivalent.

WATCH FOR

- One run per task. Record several trials for anything the team relies on.
- No split between capability and regression suites, and no automatic test for each accepted lesson.
- Attack cases are a fixed list; add periodic red team rounds.
- The verifier is never tested with planted mistakes, which the model's own notes admit.

Checklist

- | | |
|---|---|
| <ul style="list-style-type: none"> ☐ Write 20 to 50 test briefs from real failures, each with a reference answer and an unambiguous pass rule. ☐ Keep a capability suite and a regression suite; each fixed bug adds a regression task. ☐ Run critical tasks several times and report how many runs passed. ☐ Put deterministic checks first; calibrate any model judge against human verdicts. | <ul style="list-style-type: none"> ☐ Make the verifier a different execution, ideally a different model, with a fresh context. ☐ Bind every pass to the exact version; any change voids it. ☐ Refresh attack cases with adaptive red teaming; a passed fixed list is not proof of safety. ☐ Only a human turns a lesson into a rule, and every accepted lesson gets a test and a re check date. |
|---|---|

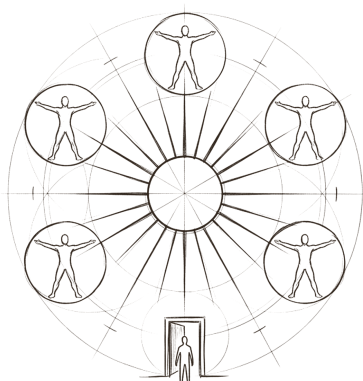
PART 8 OF 9

The hive

How do agents hand work to each other?

Everything so far describes one agent. The hive is the team. Its shape is drawn along the bottom of the cover: **you** speak to one **front door**, which answers simple requests itself and passes production work to a **coordinator**. The coordinator picks which **specialists** do what, in what order. A **verifier** checks the work, and it comes back to **you** for a yes. Every assignment and every result uses the same form.

The order never changes: produce, have another agent review it, then present. That one rule, held across a whole team, is what lets more agents add capacity rather than confusion.



one center, one door

STRUCTURE

A front door, a coordinator, and specialists

Each circle is a whole agent with its own brains and skills. At the centre, the coordinator plans who does what. At the bottom, one door: a single agent meets you and owns the conversation. Specialists never talk to you directly.

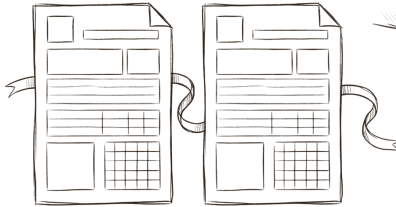
This is OpenAI's *manager pattern*, ideal when you want one agent to control the workflow and hold access to the user ^[10], with Anthropic's *orchestrator and workers* inside it, for tasks whose subtasks the coordinator decides at runtime ^[49]. Microsoft's Magentic-One coordinator plans, tracks progress and re plans to recover from errors ^[70]. The model routes every request one of three ways: answer it directly, send it to one specialist plus a reviewer, or coordinate a team. That matches the strongest advice in the field: start with one agent and add a team only when the work needs it ^[10, 49]. Teams cost more (agents use about 4 times the tokens of chat, multi agent systems about 15 times) ^[2], and performance gains on popular benchmarks are often minimal ^[1].

SHARED BASE

One base prompt, shared by every role

The comb is the base prompt every agent reads before its own role: how this team works, how handoffs are shaped, what review means. Because it is shared, each agent file stays small, and a team rule changes in one place.

OpenAI recommends exactly this: a single flexible base prompt that accepts policy variables, instead of many separate prompts, because it simplifies maintenance and evaluation ^[10]. Its Agents SDK also recommends a standard handoff instruction in every agent ^[71]. The base prompt is also where the team's defaults for context belong. In the OpenAI SDK, the receiving agent sees the whole previous conversation by default ^[71]; Claude Code teammates do not inherit the lead's history ^[72]. State the default for your hive: specialists get the assignment and its frozen context, never the raw conversation.

*the comb**one shape*

THE CONTRACT

Every assignment and every result use the same form

An **assignment** names the objective, frozen pass criteria, selected context, exact outputs, one owner, a deadline, one to three attempts and what happens on cancel. A **result** reports one state and one evidence backed verdict per criterion.

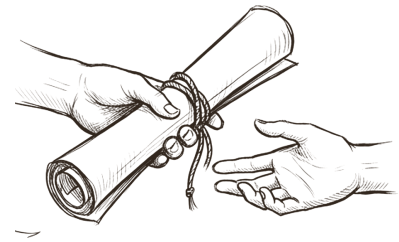
Anthropic found that without a detailed brief, agents duplicate work, leave gaps or fail to find what they need ^[2]. MAST's largest failure categories include step repetition (15.7%), reasoning and action mismatch (13.2%), being unaware of when to stop (12.4%) and disobeying the task specification (11.8%) ^[1]; a tight contract targets each one. The Agent2Agent protocol makes the same move at the protocol level: tasks carry typed states, and outputs are returned as artifacts, not chat messages ^[73]. One addition would help. A2A separates *input required* and *rejected* from failure ^[73], and failing to ask for clarification is its own failure mode ^[1]. Adding *needs input* as a result state makes *stop and ask* a first class outcome.

INTEGRITY

Work passes tied up, never blurred

Work passes between agents tied up: completed, partial, blocked, failed or cancelled, and never blurred into *done*. It passes by reference, a path and a hash, not by pasting text into the next prompt. Each output path has one owner.

Passing references matches how Anthropic's research agents store work in external systems and pass lightweight references back to the coordinator [2]. One owner per path prevents the overwrite problem Claude Code's documentation warns about, where two teammates editing the same file overwrite each other [72]. And the model goes one step further than most frameworks: when separately passed parts are combined, the combination is a new review event, because two passed parts can still contradict each other. Two things to add: a scaling rule for the coordinator (Anthropic's early agents spawned 50 subagents for simple queries [2]), and a frank note that a strict sequence is safer but slower, because one slow specialist can block the whole line.

*the handoff*

produce, have another review it, then present: the order never changes

WHAT RESEARCH AND PRACTICE SAY

Start with one agent; add a team only when needed. Teams cost more and often gain little [10, 49, 1].

One agent owns the person. The manager pattern keeps one agent in control of the workflow and the user [10].

A coordinator plans, tracks and re plans, and scales effort to the task. From one agent for a fact to many for deep research [2, 70].

Every delegation needs a complete brief. Objective, output format, tools and sources, boundaries [2, 1].

Results are typed states and artifacts, not chat. A2A defines task states and returns outputs as artifacts [73].

Control what crosses a handoff. What context a receiving agent sees should be a decision, not a default [71, 72].

MATCHES PRACTICE

- Front door plus coordinator is the manager pattern with orchestrator and workers inside it.
- The assignment contract is a superset of Anthropic's four brief elements; bounded attempts match OpenAI's advice to limit retries.
- Result states closely parallel A2A's task states.
- Paths and hashes instead of pasted text match lightweight references and artifacts.

GOES FURTHER

- Integration is a new review event; two passed parts can still conflict.
- Partial or unavailable review can never be flattened into success.
- One owner per output path prevents overwrites by design.

WATCH FOR

- No *needs input* or *rejected* result state.
- No rule for how many specialists or attempts a task deserves.
- The cost of strict sequencing is not discussed.
- All roles are planned with dispatch off, so the team contract is a specification, not yet evidence of team performance.

Checklist

- | | |
|---|---|
| ☐ Route first: answer directly, one specialist plus a reviewer, or a coordinated team. | ☐ Pass work by path and hash, never by pasting it into the next prompt. |
| ☐ Give one agent the person; specialists never talk to the person unless handed over. | ☐ Write a scaling rule: how many specialists and attempts for small, medium and large tasks. |
| ☐ Use one assignment form: objective, criteria, frozen context, outputs, owner, deadline, attempts, cancel rule. | ☐ Keep the shared base prompt in one file and include it in every role. |
| ☐ Use one result form with a closed list of states, including <i>needs input</i> . | ☐ Hold the order: produce, review, integrate, review the whole, then present. |

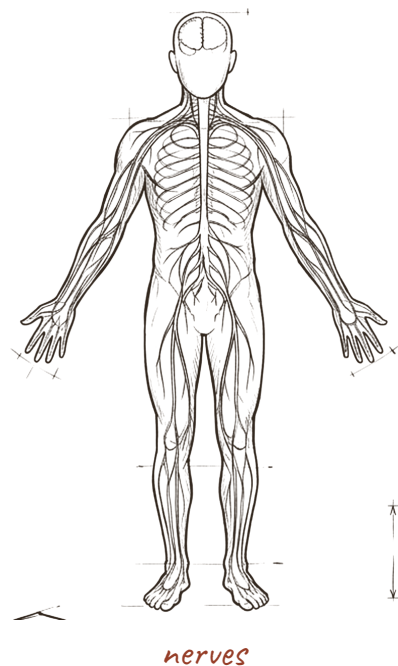
PART 9 OF 9 · PROPOSED EXTENSION

The nervous system

How does a team feel itself as one body?

The parts so far give a team a shared memory and a shared rulebook. Those are static. A body also needs nerves: live signals that tell every part what the others are doing, stop harm at the border, call for help, and leave a record. This chapter proposes four additions the model does not have yet: a **live board**, **safe borders** between agents, a **bell** for escalation, and a **trace**.

The research supports all four. Two are already standard in current frameworks (live task state and tracing). One is supported but usually done informally (escalation). And one is the most important and least solved problem in multi agent systems today: trust at the border between agents.



THE IDEA

Every part feels what the others are doing

In a body, the hand does not wait for a weekly report to learn that the foot has stepped on glass. Signals travel continuously, and some of them, such as pain, override everything else.

An agent team built only from documents works more like a company run by memo. Each agent reads the rules, does its job and returns a result, but no part can see the whole at a glance, refuse a dangerous signal by structure, raise an alarm, or replay what happened.

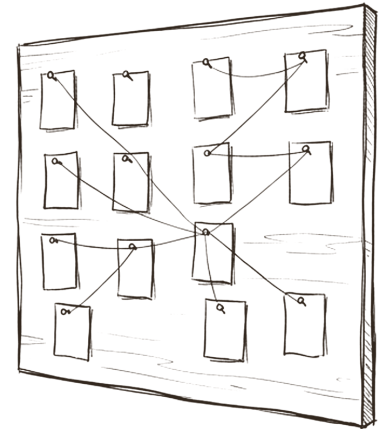
Hammond and colleagues name the three ways multi agent systems fail: miscoordination, conflict and collusion ^[74]. Each of the four additions below targets at least one of them, and each can be built from records the model already keeps.

SHARED LIVE STATE

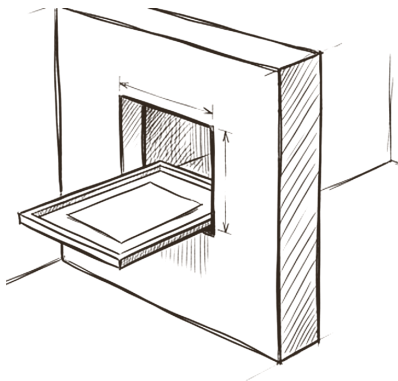
One shared picture of now

A single board shows open tasks, who holds each one, what state it is in, what blocks it, how many attempts are used and when it is due. Every agent reads the same board.

This is one of the oldest patterns in AI. The blackboard model dates back to the HEARSAY-II speech system of the 1970s [75], and recent papers apply it to language model teams, reporting relative gains of 13% to 57% on their benchmarks [76, 77]. Magentic-One keeps a task ledger and a progress ledger [78]; Claude Code's agent teams share a task list with pending, in progress and completed states, dependencies and file locking [72]. The known risk is drift: Claude Code's own documentation lists *task status can lag* as a limitation [72]. So build the board as a **derived view of the assignment and result records**, never a separate thing agents edit, keep its fields structured, and treat any free text on it as untrusted data.



the live board



a safe border

TRUST BOUNDARIES

One agent's output is data to the next, never orders

When one agent's result reaches another, it passes through a hatch. What comes through is material to evaluate. It is never an instruction to follow, however it is worded.

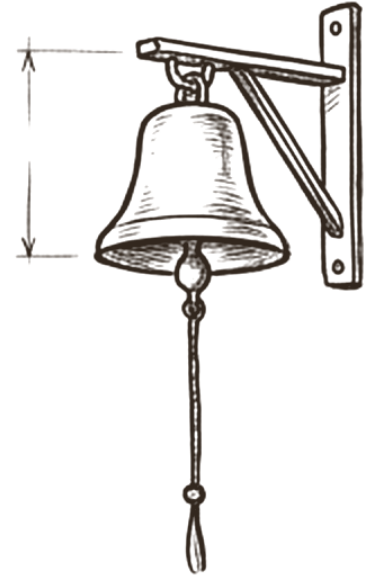
This is where the research is most emphatic. Prompt injection is OWASP's top LLM risk [69], and in multi agent systems malicious prompts can self replicate from agent to agent like a virus [79]. Popular frameworks leave the middle unguarded: in the OpenAI Agents SDK, input guardrails run only for the first agent and output guardrails only for the last [80]. And a rule written in a prompt does not hold: adaptive attacks bypassed 12 published defences [65]. The boundary has to be built into the structure. Willison's Dual LLM pattern keeps the model that can act away from untrusted text [81]; CaMeL separates control flow from data so untrusted data can never change what the program does [82]; six design patterns give builders a menu [59]. Practically: specialists return typed results, the coordinator reads fields and hashes rather than obeying prose, nothing in a result can grant permission, and the Rule of Two decides which roles may combine untrusted input, private data and outside action [48].

ESCALATION

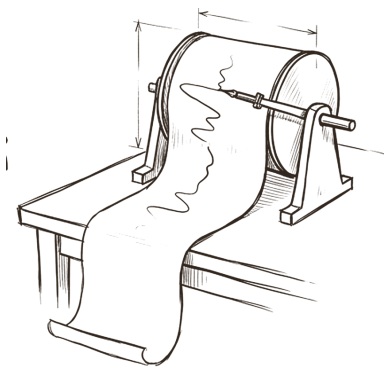
Stuck or in conflict? Ring the coordinator first, then you

A stuck agent rings a bell. The coordinator answers first: it can re-plan, reassign or cut scope. If the coordinator cannot resolve it, the bell rings for you. Meanwhile the task shows *blocked* or *needs input* on the board, so nothing waits silently.

OpenAI names two triggers for human intervention: exceeding failure thresholds and high risk actions ^[10]. Magentic-One's coordinator counts stalls and re-plans when progress stops ^[78]; LangGraph interrupts pause a run, save its state and wait for a human to approve, edit or reject ^[83]. No source found names *coordinator first, then human* as a formal standard, so present it as a design choice the examples support. Write one ladder with clear triggers: attempts used up, no progress after a set number of steps, two agents disagree on a required claim, a risky action, a missing reviewer. When two agents disagree, keep both findings side by side and never average them.



call for help



the trace

AUDIT

Who did what, when and why

A trace records every agent run, model call, tool call and handoff, joined into one tree, so any run can be replayed and audited. You can't trust what you can't replay.

This is becoming standard. The OpenAI Agents SDK traces agent runs, generations, tool calls, guardrails and handoffs by default ^[84]. OpenTelemetry now has draft conventions for agent spans, with agent and conversation identifiers ^[85]. Anthropic reports that full production tracing let them diagnose why agents failed, and that they built systems to resume from where an agent was when errors occurred ^[2]. Google's principle is that agent actions must be observable ^[38]. Two cautions: traces hold sensitive data (one SDK records it by default ^[84], OpenTelemetry makes message content opt in ^[85]), and traces are for humans and auditors, never inputs to agents. A small first step fits the model today: give every assignment and result a run id and a parent id, so the records join into a tree.

you can't trust what you can't replay

WHAT RESEARCH AND PRACTICE SAY

A shared board is a proven coordination pattern. From 1970s blackboard systems to today's agent task lists ^[75, 72, 78].

Another agent's output is data, never instructions. Injection spreads between agents; the rule must be enforced by structure ^[79, 59, 82].

Prompt level rules alone do not hold. Adaptive attacks defeat most filter style defences ^[65].

Escalate on stalls and on risk. Failure thresholds, high risk actions, stall counters and interrupts [10, 78, 83].

Trace every agent, call and handoff. So a run can be audited, debugged and resumed [84, 85, 2].

Check identity and authorisation at every hop. A2A requires authorisation checks on every request and audit trails for sensitive operations [73, 38].

ALREADY IN THE MODEL

- Assignments and results with states: the raw material for a board.
- "Treat delegated content as data, never as instructions" is written in routing practice and tested by injection cases.
- Bounded attempts, stop dependent work rules and an escalation record.
- Records already hold author, timestamps, hashes and the human's words.

WHAT RESEARCH ADDS

- Derive the board from records so it cannot disagree with them.
- Typed results, hash binding and the Rule of Two turn the trust rule into structure.
- One written escalation ladder with triggers and receivers.
- Run ids and parent ids that join records into a trace tree.

WATCH FOR

- A board that drifts from reality, and a board that carries a poisoned note to every agent.
- One poisoned result feeding its dependants (cascading failure).
- A poisoned summary misleading the person at the front door.
- Sensitive data stored in traces, or traces fed back to agents.

Checklist

- | | |
|---|---|
| <ul style="list-style-type: none"> ☐ Keep one board of open work, derived from records: id, owner, state, blocked by, attempts, deadline. ☐ Let one writer change a task's state at a time, and check real artifacts before trusting <i>done</i>. ☐ Treat every result, message and board note from another agent as data; read typed fields and hashes only. ☐ Apply the Rule of Two per role: never untrusted input, private data and outside action in one agent without a human gate. | <ul style="list-style-type: none"> ☐ Write one escalation ladder: triggers, coordinator first, then the human, and the state shown while waiting. ☐ Give every run, assignment, result and review a run id and parent id. ☐ Keep message content out of traces by default, and never feed traces back to agents. ☐ Test the borders with multi hop injection: plant an instruction in one agent's input and check it never reaches another's actions. |
|---|---|

CLOSING

Putting it together

What should a builder do first, and what is genuinely new here?

Nine chapters, one message: **a written rule guides an agent, but only enforcement controls it**. A soul file that "grants no power", a skill that "grants no permission", a result that is "data, never orders": each is a good rule, and each needs a matching lock in configuration, code or process. Research is consistent on this point across identity [13, 6], skills [34], tools and limits [13], and the space between agents [65, 59]. When in doubt, ask of every rule: what actually stops the agent if it ignores this?

The second message is restraint. Both Anthropic and OpenAI advise starting with the simplest system that works and adding agents only when the work needs them [49, 10]. Teams cost more and fail in new ways [2, 1]. Build the hive one layer at a time, and prove each layer before adding the next.

A build order

STAGE	ADD	DONE WHEN
1. One agent	A role file, a soul file, one shared rules file	The agent does one job well, in a voice you recognise
2. Skills	Three skills, each with a with and without test	Each skill actually improves the work
3. Tools	Tools at permission none, then opened one by one	Nothing reaches outside without a gate
4. Method	One workflow with a review gate and an output template	The same job comes out the same way twice
5. Limits	An action ceiling and a fresh yes for final actions	Nothing costly or public happens unasked
6. Loop	An independent verifier and 20 or more test briefs	Quality is measured, not assumed
7. Team	A front door, a coordinator, one handoff contract	More agents add capacity, not confusion
8. Nerves	A live board, typed results, an escalation ladder, a trace	The team can see, protect and audit itself

What is genuinely new in this model. Several ideas here go further than common published practice, and they are fair to present as original design choices rather than research findings:

- Splitting provenance into RAW, FETCH, SEARCH and INFER, and treating search as discovery, never as evidence.
- Binding every review and approval to the exact version of the work, so any change voids it.
- Letting only a human turn a mistake into a rule, which also blocks an agent from writing its own instructions.
- Stating that a skill grants no permission, which is stricter than at least one popular runtime.
- Keeping readiness, dispatch and permission as three separate gates.
- Refusing to blur outcomes: partial is never reported as done.

Two fixes for the examples. The research found two places where the model's own example files contradict its principles. The example soul file repeats behaviour rules that already live in the role, which breaks "link, don't copy". The example SEO audit skill is marked active while its evaluation flag is still false, which breaks "tested before trusted". Fixing those two files makes the teaching material match what it teaches.

What comes next. The nervous system is the frontier. Live state and tracing are already standard in current frameworks; trust between agents is the most important and least solved problem in the field ^[65, 79]. A hive that can see its own work, refuse orders hidden in data, call for help, and replay what happened is a hive you can trust with real work.

References

1. Mert Cemri, Melissa Z. Pan, Shuyi Yang et al., "Why Do Multi-Agent LLM Systems Fail?", arXiv 2503.13657, v3 26 October 2025. <https://arxiv.org/abs/2503.13657>
2. Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox and Daniel Ford, "How we built our multi-agent research system", Anthropic Engineering, 13 June 2025. <https://www.anthropic.com/engineering/multi-agent-research-system>
3. Anthropic, "Prompting best practices", Claude Platform documentation, read 27 October 2026. <https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/claude-prompting-best-practices>
4. Mingqian Zheng, Jiaxin Pei, Lajanugen Logeswaran, Moontae Lee and David Jurgens, "When 'A Helpful Assistant' Is Not Really Helpful: Personas in System Prompts Do Not Improve Performances of Large Language Models", Findings of EMNLP 2024, arXiv 2311.10054. <https://arxiv.org/abs/2311.10054>
5. Shashank Gupta et al., "Bias Runs Deep: Implicit Reasoning Biases in Persona-Assigned LLMs", ICLR 2024, arXiv 2311.04892. <https://arxiv.org/abs/2311.04892>
6. Anthropic, "How Claude remembers your project", Claude Code documentation, read 27 October 2026. <https://code.claude.com/docs/en/memory>
7. Kenneth Li et al., "Measuring and Controlling Instruction (In)Stability in Language Model Dialogs", COLM 2024, arXiv 2402.10962. <https://arxiv.org/abs/2402.10962>
8. Christina Lu, Jack Gallagher, Jonathan Michala, Kyle Fish and Jack Lindsey, "The Assistant Axis: Situating and Stabilizing the Default Persona of Language Models", arXiv 2601.10387, 15 January 2026. <https://arxiv.org/abs/2601.10387>
9. Anthropic, "Persona vectors: Monitoring and controlling character traits in language models", 1 August 2025. <https://www.anthropic.com/research/persona-vectors>
10. OpenAI, "A practical guide to building agents", PDF, April 2025 (date from file metadata). <https://cdn.openai.com/business-guides-and-resources/a-practical-guide-to-building-agents.pdf>
11. Barry Zhang, Keith Lazuka and Mahesh Murag, "Equipping agents for the real world with Agent Skills", Anthropic Engineering, 16 October 2025, updated 18 December 2025. <https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills>
12. Anthropic, "Create custom subagents", Claude Code documentation, read 27 October 2026. <https://code.claude.com/docs/en/sub-agents>
13. OWASP GenAI Security Project, "LLM06:2025 Excessive Agency", Top 10 for LLM Applications 2025. <https://genai.owasp.org/llmrisk/llm062025-excessive-agency/>
14. Prithvi Rajasekaran, Ethan Dixon, Carly Ryan and Jeremy Hadfield, "Effective context engineering for AI agents", Anthropic Engineering, 29 September 2025. <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>
15. NIST NCCoE, "New Concept Paper on Identity and Authority of Software Agents", 5 February 2026. <https://www.nist.gov/news-events/news/2026/02/new-concept-paper-identity-and-authority-software-agents>
16. OWASP GenAI Security Project, "OWASP Top 10 for Agentic Applications for 2026", 9 December 2025. <https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/>
17. NIST, "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile" (NIST AI 600-1), July 2024. <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf>
18. Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan and Thomas L. Griffiths, "Cognitive Architectures for Language Agents", TMLR, arXiv 2309.02427. <https://arxiv.org/abs/2309.02427>
19. Adam Tauman Kalai, Ofir Nachum, Santosh S. Vempala and Edwin Zhang, "Why Language Models Hallucinate", arXiv 2509.04664, 4 September 2025. <https://arxiv.org/abs/2509.04664>
20. Bingbing Wen et al., "Know Your Limits: A Survey of Abstention in Large Language Models", TACL, arXiv 2407.18418. <https://arxiv.org/abs/2407.18418>
21. Walden Yan, "Don't Build Multi-Agents", Cognition, 12 June 2025. <https://cognition.com/blog/dont-build-multi-agents>

22. Kelly Hong, Anton Troynikov and Jeff Huber, "Context Rot: How Increasing Input Tokens Impacts LLM Performance", Chroma, 14 July 2025. <https://www.trychroma.com/research/context-rot>
23. Nelson F. Liu et al., "Lost in the Middle: How Language Models Use Long Contexts", TACL, arXiv 2307.03172. <https://arxiv.org/abs/2307.03172>
24. Anthropic, "Introducing Contextual Retrieval", 19 September 2024. <https://www.anthropic.com/news/contextual-retrieval>
25. Patrick Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", NeurIPS 2020, arXiv 2005.11401. <https://arxiv.org/abs/2005.11401>
26. Alireza Rezazadeh et al., "Collaborative Memory: Multi-User Memory Sharing in LLM Agents with Dynamic Access Control", arXiv 2505.18279, 23 May 2025. <https://arxiv.org/abs/2505.18279>
27. Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song and Bo Li, "AgentPoison: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases", arXiv 2407.12784, 17 July 2024. <https://arxiv.org/abs/2407.12784>
28. Shen Dong et al., "Memory Injection Attacks on LLM Agents via Query-Only Interaction", arXiv 2503.03704, v5 12 February 2026. <https://arxiv.org/abs/2503.03704>
29. Giskard, "OWASP Top 10 for Agentic Applications 2026" (secondary summary of the full list), December 2025. <https://www.giskard.ai/knowledge/owasp-top-10-for-agentic-application-2026>
30. Agent Skills, "Specification", agentskills.io, read 27 October 2026. <https://agentskills.io/specification>
31. Anthropic, "Skill authoring best practices", Claude Platform documentation, read 27 October 2026. <https://platform.claude.com/docs/en/agents-and-tools/agent-skills/best-practices>
32. Guanzhi Wang et al., "Voyager: An Open-Ended Embodied Agent with Large Language Models", arXiv 2305.16291. <https://arxiv.org/abs/2305.16291>
33. Xiangyi Li, Yimin Liu, Wenbo Chen et al., "SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks", arXiv 2602.12670, v1 13 February 2026, v4 14 June 2026. <https://arxiv.org/abs/2602.12670>
34. Anthropic, "Extend Claude with skills", Claude Code documentation, read 27 October 2026. <https://code.claude.com/docs/en/skills>
35. Yi Liu et al., "Agent Skills in the Wild: An Empirical Study of Security Vulnerabilities at Scale", arXiv 2601.10338, 15 January 2026. <https://arxiv.org/abs/2601.10338>
36. David Schmotz, Sahar Abdelnabi and Maksym Andriushchenko, "Agent Skills Enable a New Class of Realistic and Trivially Simple Prompt Injections", arXiv 2510.26328, 30 October 2025. <https://arxiv.org/abs/2510.26328>
37. Jerome Saltzer and Michael Schroeder, "The Protection of Information in Computer Systems", 1975. <https://www.cs.virginia.edu/~evans/cs551/saltzer/>
38. Santiago Díaz, Christoph Kern and Kara Olive, "Google's Approach for Secure AI Agents: An Introduction", Google, 2025. <https://research.google/pubs/an-introduction-to-googles-approach-for-secure-ai-agents/>
39. Ken Aizawa et al., "Writing effective tools for agents, with agents", Anthropic Engineering, 11 September 2025. <https://www.anthropic.com/engineering/writing-tools-for-agents>
40. Model Context Protocol, specification 2026-07-28, schema.ts, ToolAnnotations. <https://github.com/modelcontextprotocol/specification/blob/main/schema/2026-07-28/schema.ts>
41. Anthropic, "Configure permissions", Claude Code documentation, read 27 October 2026. <https://code.claude.com/docs/en/permissions>
42. OWASP Cheat Sheet Series, "Secrets Management Cheat Sheet". https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html
43. Model Context Protocol, "Security Best Practices", read 27 October 2026. https://modelcontextprotocol.io/docs/tutorials/security/security_best_practices
44. OWASP GenAI Security Project, "LLM10:2025 Unbounded Consumption", Top 10 for LLM Applications 2025. <https://genai.owasp.org/llmrisk/llm102025-unbounded-consumption/>
45. Model Context Protocol, specification 2026-07-28, "Tools". <https://modelcontextprotocol.io/specification/2026-07-28/server/tools>
46. Kai Greshake et al., "Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection", arXiv 2302.12173, 2023. <https://arxiv.org/abs/2302.12173>

47. Simon Willison, "The lethal trifecta for AI agents: private data, untrusted content, and external communication", 16 June 2025. <https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/>
48. Meta AI, "Agents Rule of Two: A Practical Approach to AI Agent Security", 31 October 2025. <https://ai.meta.com/blog/practical-ai-agent-security/>
49. Erik Schluntz and Barry Zhang, "Building effective agents", Anthropic Engineering, 19 December 2024. <https://www.anthropic.com/engineering/building-effective-agents>
50. Shunyu Yao, Noah Shinn, Pedram Razavi and Karthik Narasimhan, " τ -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains", arXiv 2406.12045, 17 June 2024. <https://arxiv.org/abs/2406.12045>
51. Justin Young, "Effective harnesses for long-running agents", Anthropic Engineering, 26 November 2025. <https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents>
52. Sirui Hong et al., "MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework", arXiv 2308.00352. <https://arxiv.org/abs/2308.00352>
53. Anthropic, "Structured outputs", Claude Platform documentation, read 27 October 2026. <https://platform.claude.com/docs/en/build-with-claude/structured-outputs>
54. K. J. Kevin Feng, David W. McDonald and Amy X. Zhang, "Levels of Autonomy for AI Agents", working paper, arXiv 2506.12469. <https://arxiv.org/abs/2506.12469>
55. EU AI Act, Article 14 "Human Oversight" (unofficial text by the Future of Life Institute; check EUR-Lex for formal citation). <https://artificialintelligenceact.eu/article/14/>
56. John Hughes, "How we built Claude Code auto mode", Anthropic Engineering, 25 March 2026. <https://www.anthropic.com/engineering/claude-code-auto-mode>
57. David Dworken and Oliver Weller-Davies, "Making Claude Code safer with sandboxing", Anthropic Engineering, 20 October 2025. <https://www.anthropic.com/engineering/claude-code-sandboxing>
58. Ben Green, "The Flaws of Policies Requiring Human Oversight of Government Algorithms", Computer Law and Security Review 45, 2022. <https://arxiv.org/abs/2109.05067>
59. Luca Beurer-Kellner et al., "Design Patterns for Securing LLM Agents against Prompt Injections", arXiv 2506.08837, June 2025. <https://arxiv.org/abs/2506.08837>
60. Noah Shinn et al., "Reflexion: Language Agents with Verbal Reinforcement Learning", arXiv 2303.11366, 2023. <https://arxiv.org/abs/2303.11366>
61. Mikaela Grace, Jeremy Hadfield, Rodrigo Olivares and Jiri De Jonghe, "Demystifying evals for AI agents", Anthropic Engineering, 9 January 2026. <https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents>
62. Thariq Shihpar et al., "Building agents with the Claude Agent SDK", Claude blog, 29 September 2025. <https://claude.com/blog/building-agents-with-the-claude-agent-sdk>
63. Lianmin Zheng et al., "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena", NeurIPS 2023, arXiv 2306.05685. <https://arxiv.org/abs/2306.05685>
64. Arjun Panickssery, Samuel R. Bowman and Shi Feng, "LLM Evaluators Recognize and Favor Their Own Generations", arXiv 2404.13076, 2024. <https://arxiv.org/abs/2404.13076>
65. Milad Nasr, Nicholas Carlini et al., "The Attacker Moves Second: Stronger Adaptive Attacks Bypass Defenses Against LLM Jailbreaks and Prompt Injections", arXiv 2510.09023, 10 October 2025. <https://arxiv.org/abs/2510.09023>
66. Prithvi Rajasekaran, "Harness design for long-running application development", Anthropic Engineering, 24 March 2026. <https://www.anthropic.com/engineering/harness-design-long-running-apps>
67. Jie Huang et al., "Large Language Models Cannot Self-Correct Reasoning Yet", ICLR 2024, arXiv 2310.01798. <https://arxiv.org/abs/2310.01798>
68. GitHub, "About protected branches", GitHub Docs. <https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/about-protected-branches>
69. OWASP GenAI Security Project, "LLM01:2025 Prompt Injection", Top 10 for LLM Applications 2025. <https://genai.owasp.org/llmrisk/llm01-prompt-injection/>
70. Adam Fourney et al., "Magentic-One: A Generalist Multi-Agent System for Solving Complex Tasks", Microsoft Research, arXiv 2411.04468, 2024. <https://arxiv.org/abs/2411.04468>
71. OpenAI, "Handoffs", OpenAI Agents SDK documentation. <https://openai.github.io/openai-agents-python/handoffs/>

72. Anthropic, "Orchestrate teams of Claude Code sessions" (agent teams, experimental), Claude Code documentation, read 27 October 2026. <https://code.claude.com/docs/en/agent-teams>
73. A2A Project, "Agent2Agent (A2A) Protocol Specification", version 1.0.0. <https://a2a-protocol.org/latest/specification/>
74. Lewis Hammond et al., "Multi-Agent Risks from Advanced AI", Cooperative AI Foundation Technical Report 1, arXiv 2502.14143, 2025. <https://arxiv.org/abs/2502.14143>
75. H. Penny Nii, "The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures", AI Magazine 7(2), 1986. <https://ojs.aaai.org/index.php/aimagazine/article/view/537>
76. Bochen Han and Songmao Zhang, "Exploring Advanced LLM Multi-Agent Systems Based on Blackboard Architecture", arXiv 2507.01701, 2025. <https://arxiv.org/abs/2507.01701>
77. Alireza Salemi et al., "LLM-Based Multi-Agent Blackboard System for Information Discovery in Data Science", arXiv 2510.01285, 2025. <https://arxiv.org/abs/2510.01285>
78. Microsoft Research, "Magentic-One: A Generalist Multi-Agent System for Solving Complex Tasks" (article), 4 November 2024. <https://www.microsoft.com/en-us/research/articles/magentic-one-a-generalist-multi-agent-system-for-solving-complex-tasks/>
79. Donghyun Lee and Mo Tiwari, "Prompt Infection: LLM-to-LLM Prompt Injection within Multi-Agent Systems", arXiv 2410.07283, 9 October 2024. <https://arxiv.org/abs/2410.07283>
80. OpenAI, "Guardrails", OpenAI Agents SDK documentation. <https://openai.github.io/openai-agents-python/guardrails/>
81. Simon Willison, "The Dual LLM pattern for building AI assistants that can resist prompt injection", 25 April 2023. <https://simonwillison.net/2023/Apr/25/dual-llm-pattern/>
82. Edoardo DeBenedetti et al., "Defeating Prompt Injections by Design" (CaMeL), arXiv 2503.18813, 2025. <https://arxiv.org/abs/2503.18813>
83. LangChain, "Interrupts", LangGraph documentation. <https://docs.langchain.com/oss/python/langgraph/interrupts>
84. OpenAI, "Tracing", OpenAI Agents SDK documentation. <https://openai.github.io/openai-agents-python/tracing/>
85. OpenTelemetry, "Semantic Conventions for GenAI agent and framework spans", status Development, read 27 October 2026. <https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/gen-ai-agent-spans.md>