6

PROMPT CHAIN · ROUTER · PARALLEL · SUPERVISOR · EVALUATOR · AUTONOMOUS

Multi-Agent Orchestration.

Anthropic's canon, framework trade-offs, and the four anti-patterns that kill multi-agent projects.

THE BRIEF, IN THREE BEATS

01 Most "multi-agent" systems should be a workflow, not agents.

02 The framework is a rounding error. The pattern is the design.

03 Production = HITL gates + observability + budgets. Everything else is demoware.

SLIDE 02 · HOOK

"We built a five-agent system.

Then we rewrote it as three if-statements."

The most common multi-agent story of 2026. Anthropic named it: workflows beat agents when the problem is deterministic. The trick is knowing which one you're actually holding.

Field pattern from Anthropic State of AI Agents 2026 · Medium "Multi-Agent in Production 2026 · What Actually Survived" · Reddit r/LangChain deployment threads.

ARCHITECTURE CLAIM

Multi-agent is not more agents. It's the right amount of agent — most of your system is a workflow, and the honest engineer says so out loud.

THREE COROLLARIES

01 Start with a *single agent*. Add coordination only when it demonstrably fails.

02 Pattern > framework. The same pattern is portable across LangGraph/CrewAI/AutoGen.

03 Every agent must have a *budget* — turns, tokens, tools, wall-clock. Or it's not an agent, it's a bug.

THE HONEST TRADE

Workflows are cheaper. Agents are more capable. Pick deliberately.

Workflow wins: predictable, low-cost, debuggable, testable, cheap to change.

Agent wins: open-ended tasks, dynamic tool use, tasks with an unknown number of steps.

> Anthropic's rule of thumb: *"Find the simplest solution possible, and only add complexity when there's a demonstrated need."*

Source: Anthropic "Building Effective Agents" (Dec 2024, updated 2026) · anthropic.com/research/building-effective-agents.

ANTHROPIC CANON

Who decides the next step — your code or the model?

WORKFLOW

Developer controls the flow.

CONTROL · Predefined code path. LLM fills slots.

TOKENS · Predictable. Every LLM call is a discrete node.

DEBUGGING · Trivial — you know which node ran.

USE FOR · Classify + route · summarise + email · extract fields · pipeline of known steps.

PATTERNS · Prompt chain · router · parallelisation.

AGENT

Model controls the flow.

CONTROL · LLM picks the next tool. You supply the loop.

TOKENS · Unpredictable. Budget or it eats itself.

DEBUGGING · Traces & replays — the reason observability matters.

USE FOR · Coding assistants · research agents · open-ended tool use · dynamic planning.

PATTERNS · Supervisor · evaluator-optimiser · autonomous loop.

Anthropic (2026) now calls it a "harness" — the runtime scaffolding around the model. That's the L4 view: you build harnesses, not agents.

Sources: anthropic.com/research/building-effective-agents · Towards AI "Stop Calling It an Agent · Anthropic Calls It a Harness" (2026).

If none of these apply — use a single agent, or a workflow.

REASON 1 · CONTEXT ISOLATION

One context per specialisation.

Each agent gets a fresh 200K window — not a shared, cluttered one. Legal review + financial analysis + summariser don't need to see each other's junk.

Signal: your single-agent prompt is over 8K tokens and still growing.

> This is the reason Claude Code sub-agents exist — memory hygiene, not parallelism.

REASON 2 · REAL PARALLELISM

Wall-clock, not throughput.

10 competitor briefs, serial: 40 min. Parallel: 5 min. Same tokens, radically different wall-clock.

Signal: you have N independent subtasks that don't need to see each other's output.

> Beware the fake parallel — if agent 2 needs agent 1's output, you have a chain, not parallelism.

REASON 3 · ADVERSARIAL SEPARATION

Producer vs. critic.

One agent writes, another critiques. Different system prompts, different roles — better than one agent doing both.

Signal: "grade your own work" is a known weak spot for LLMs. Split the role.

> This is the evaluator-optimiser pattern (slide 10). Also underpins constitutional AI, red-teaming rigs.

Sources: Reddit r/ClaudeAI sub-agent thread · LangChain "Benchmarking Multi-Agent Architectures" · Anthropic "Building Effective Agents".

WORKFLOW · DETERMINISTIC

#01 A → B → C. Fixed order. Optional gates between.

SHAPE · ONE CALL FEEDS THE NEXT

STEP 1 → STEP 2 → STEP 3

```
extract = llm(EXTRACT_PROMPT, doc)
if not extract.ok: return "cannot process"
translated = llm(TRANSLATE_PROMPT, extract)
summary = llm(SUMMARISE_PROMPT, translated)
return summary
```

> Add a gate ("if step 1 is bad, stop") to catch failures cheap. Anthropic calls this *trading latency for accuracy*.

WHEN TO REACH FOR IT

The task decomposes into *fixed subtasks* in a known order.

Examples: outline → draft → edit · translate → localise → QA · classify → route → respond.

Every step's output is the next step's input. No branching required.

THE TRAP

Prompt chains masquerading as agents.

If your "agent" always does A then B then C in the same order, it's a chain. Stop paying for an autonomous planner. Just write the code.

Sources: anthropic.com/research/building-effective-agents (prompt chaining pattern) · LangGraph docs "sequential graphs".

WORKFLOW · DETERMINISTIC

#02 One classifier picks the branch. The branches are specialists.

SHAPE · CLASSIFY → DISPATCH

CLASSIFY → ONE OF N BRANCHES

```
route = llm(CLASSIFIER, ticket)
match route:
  case "billing": return billing_agent(ticket)
  case "technical": return tech_agent(ticket)
  case "refund": return escalate_to_human(ticket)
  default: return generic_agent(ticket)
```

> Cheap Haiku for the classifier, expensive Opus only on the branch that needs it. This is *tiered inference*.

WHEN TO REACH FOR IT

Inputs fall into *distinct categories* that each want a different prompt / tool set.

Examples: support triage · document type routing · easy/hard question routing · language routing.

The classifier is the whole trick. Get it right; everything downstream stays simple.

THE TRAP

"Other" is the failure category.

If >10% of tickets fall into "other", the taxonomy is wrong. Fix the categories — don't build a fallback branch that's actually doing all the work.

Sources: anthropic.com/research/building-effective-agents (routing pattern) · LangGraph "conditional edges" · CrewAI hierarchical process.

WORKFLOW · DETERMINISTIC

#03 Fan out. Merge. Two flavours: sectioning & voting.

FLAVOUR A · SECTIONING

Different subtasks, run in parallel, merged.

SHAPE · Split the problem into *disjoint* sections. Each runs independently.

EXAMPLE · One agent screens for safety, another for policy, another for facts — all on the same output. Merge = AND of results.

WIN · Wall-clock reduction. Same tokens, lower latency.

TRAP · If sections aren't truly independent, you built a race condition.

FLAVOUR B · VOTING

Same task, N times, majority wins.

SHAPE · Run the same prompt 3–5 times at temp>0. Take majority.

EXAMPLE · "Is this code safe?" × 5. If 4/5 say yes, ship. If 3/2 split, escalate to human.

WIN · Accuracy on the tail. LLMs are more consistent in aggregate than in single-shot.

TRAP · N× the tokens. Only use for high-stakes calls.

Sources: anthropic.com/research/building-effective-agents (parallelisation pattern) · LangGraph "Send" API for fan-out · self-consistency (Wang et al., 2022).

AGENT · LLM-DRIVEN

#04 One supervisor. N specialists. Delegate, wait, integrate.

SHAPE · SUPERVISOR AS TOOL-USER

Supervisor calls specialists like tools. Sub-agents return, control comes back up.

01 · Supervisor plans: "I need research, then code, then a summary."

02 · Calls researcher_agent(query). Waits. Reads output.

03 · Calls coder_agent(spec). Waits. Reads output.

04 · Integrates. Returns final answer.

> This is Claude Code's sub-agent pattern — and LangGraph-Supervisor's core primitive.

WHEN TO REACH FOR IT

Task has *unknown-in-advance* subtasks. Number and order of steps depends on the input.

Examples: deep research assistants · coding agents that call refactor/test/lint · customer service where the flow is data-dependent.

THE TRAP

Supervisor context bloat.

Each sub-agent's full output lands in the supervisor's window. After 5 calls, you're at 40K tokens of scrollback. Have sub-agents return *summaries*, not raw dumps.

Sources: LangChain "Choosing the Right Multi-Agent Architecture" · reference.langchain.com/python/langgraph-supervisor · Anthropic orchestrator-workers.

AGENT · LLM-DRIVEN

#05 Generate. Critique. Iterate. Until the critic says done.

SHAPE · TWO AGENTS, ONE LOOP

GENERATOR ↔ EVALUATOR (BUDGETED)

```
draft = generator(task)
for i in range(MAX_ITER):
  verdict = evaluator(draft, rubric)
  if verdict.pass: break
  draft = generator(task, verdict.feedback)
return draft
```

> The evaluator needs a *rubric*, not vibes. Concrete criteria = better feedback = fewer iterations.

WHEN TO REACH FOR IT

You have a *testable quality bar* the generator alone can't reliably hit.

Examples: code that must compile + pass tests · translation with style constraints · answers with citation requirements · red-team probe generation.

THE TRAP

MAX_ITER is not optional.

Without a hard iteration cap, the loop can run forever — especially if the evaluator's rubric is too strict. Cap at 3–5. Log every iteration.

Sources: anthropic.com/research/building-effective-agents (evaluator-optimiser pattern) · Reflexion (Shinn et al., 2023) · Self-Refine (Madaan et al., 2023).

AGENT · FULLY AUTONOMOUS

#06 Think → act → observe → repeat. No preset flow.

SHAPE · THE REACT LOOP

MODEL DECIDES EVERY STEP

```
while not done and turns < MAX:
  thought, action = agent.think(state)
  if action == "answer": return thought
  obs = tools[action.name](action.args)
  state.append(thought, action, obs)
raise BudgetExceeded
```

> This is what Claude Code, Manus, AutoGPT all are underneath. The engineering is in the *tools list* and the *termination conditions*.

WHEN TO REACH FOR IT

Truly open-ended tasks. Unknown steps. Unknown tools per step. Number of iterations unpredictable.

Examples: coding agent debugging live code · research agent chasing citations · Manus building an app from spec.

THE FOUR BUDGETS YOU MUST SET

- `TURNS` · Max iterations. Default 25.
- `TOKENS` · Total context + total generated.
- `$ COST` · Hard stop above threshold.
- `WALL-CLOCK` · Timeout on the whole run.

Sources: ReAct (Yao et al., 2022, arXiv:2210.03629) · anthropic.com/research/building-effective-agents (agents section).

EXPLICIT STATE GRAPH

LG

If you want to see the graph, this is the default.

THE PITCH, IN ONE PARAGRAPH

Every node is a function. Every edge is explicit. Every state transition is a checkpoint you can replay.

PRIMITIVES · StateGraph · nodes · conditional edges · subgraphs · Command (for handoffs).

SUPERVISOR · Ready-made lib (`langgraph-supervisor`) for the pattern.

HITL · Native `interrupt()` for approval gates — the reason many teams pick it.

OBSERVABILITY · LangSmith integration is first-class; token cost per node is visible.

REACH FOR IT WHEN

You want *explicit control* over state, branching, and human interrupts.

You need *token-cost predictability* (every node = one LLM call).

You already use LangChain / LangSmith.

THE TRAP

Steep learning curve for what could be a workflow.

If your problem is a 3-step chain, LangGraph is over-engineered. Save it for graphs with real branching, loops, or human gates.

Sources: langchain.com/blog/choosing-the-right-multi-agent-architecture · reference.langchain.com/python/langgraph-supervisor · pub.towardsai.net production guide 2026.

CA

Assign roles like a project manager. Watch the crew execute.

THE PITCH, IN ONE PARAGRAPH

Declare Agents (role + goal + backstory) and Tasks (description + expected_output). CrewAI schedules them.

PRIMITIVES · Agent · Task · Crew · Process (sequential / hierarchical / consensual).

HIERARCHICAL · Built-in manager agent that delegates to workers — supervisor pattern out of the box.

MEMORY · Short-term, long-term, entity-based — opt-in per crew.

TIME-TO-DEMO · Under an hour for a working 3-agent crew. Fastest of the three.

REACH FOR IT WHEN

Prototype speed matters more than production control.

Roles map naturally to your problem (researcher / writer / editor).

The task shape is *hierarchical*, not a graph.

THE TRAP

Token opacity in complex crews.

The scheduler hides some LLM calls behind the abstraction. In production, wire OpenTelemetry from day one — or you'll get a Pro plan bill from a demo.

Sources: datacamp.com/tutorial/crewai-vs-langgraph-vs-autogen · docs.crewai.com · pickaxe.co/post/crewai-vs-langgraph-vs-autogen.

If your cloud is Azure or AWS, these are the natural picks.

AUTOGEN · MICROSOFT

Conversation-first · event-driven.

MODEL · Agents exchange messages. Actor-model semantics. Async by default.

STRENGTH · Enterprise pedigree, .NET/Python parity, deep Azure integration.

USE FOR · Long-running conversations between agents · human-agent-agent flows · agent labs / research.

CAVEAT · Steeper setup than CrewAI; less transparent than LangGraph. Reward: stability at scale.

STRANDS · AWS (2026)

Minimal SDK · three patterns baked in.

MODEL · Loop + tools + system prompt. Everything else opt-in.

3 PATTERNS · Swarm (peer agents) · Graph (DAG) · Workflow (sequential). Plus A2A protocol.

USE FOR · Serverless AWS Lambda / Fargate deploys · Bedrock-first stacks · orgs already on AWS SSO/KMS.

CAVEAT · Newer — smaller community than LangGraph. Reward: no vendor drift on AWS.

Sources: strandsagents.com · docs.aws.amazon.com/prescriptive-guidance/latest/agentic-ai-frameworks/strands-agents.html · microsoft.com AutoGen docs.

FOUR GATE PATTERNS

Match the gate to the reversibility of the action.

#	GATE PATTERN	USE WHEN	EXAMPLE
01	**Runtime approval gate.** Blocks execution until human clicks approve.	Irreversible actions: send email, wire money, delete data, deploy to prod.	Agent drafts email → pause → human reviews in Slack → approve/reject → send.
02	**Confidence-based escalation.** Only pauses when the agent is unsure.	High-volume tasks where 90% are safe to autopilot.	Ticket triage: confidence < 0.8 → human queue. Everything else auto-resolves.
03	**Output validation queue.** Agent completes; human batch-reviews later.	Async workflows. Cost of a bad output is fixable, not catastrophic.	Overnight research briefs → 8am review queue → editor red-pens before send.
04	**Checkpoint pause/resume.** Save state at milestones; human resumes.	Long-running (hours+) tasks where the human wants to inspect progress.	Coding agent: pause after design doc, resume after human edit. LangGraph `interrupt()`.

Sources: galileo.ai/blog/human-in-the-loop-agent-oversight · developers.cloudflare.com/agents/concepts/agentic-patterns/human-in-the-loop · redis.io/blog/ai-human-in-the-loop.

TWO PARADIGMS · ONE TRAP

Shared memory vs. explicit handoffs. Pick one deliberately.

PARADIGM A · SHARED MEMORY

A store all agents read/write.

SHAPE · Redis / Postgres / Mem0 / Zep. Agents write facts, read facts.

WIN · No context bloat in individual agents. Cross-session persistence.

USE FOR · Long-running conversations · shared knowledge graphs · agents that need to remember users across sessions.

> Trap: race conditions, stale reads, no clear ownership. Test *concurrent writes* before shipping.

PARADIGM B · EXPLICIT HANDOFFS

Message passing. What A sends, B sees.

SHAPE · Agent A returns a payload. Agent B receives exactly that payload. No back-channel.

WIN · Deterministic. Debuggable. Clear ownership. No race conditions.

USE FOR · Supervisor patterns · workflows · anything you'll need to trace in production.

> Trap: context bloat — every handoff carries the payload. Force *summaries*, not raw dumps.

Sources: mem0.ai/blog/multi-agent-memory-systems · Reddit r/aiagents "shared memory vs handoffs" · medium "Multi-Agent in Production 2026".

THE L3 GATE

If you can't answer "why did the agent do that?" in 60 seconds, you don't have observability.

01 · TRACE

The whole run, replayable.

Every node, every tool call, every prompt, every response — timestamped, ordered, click-to-inspect.

Stack: LangSmith · Langfuse · Arize Phoenix · OpenTelemetry.

> Non-negotiable at L3. Do this before you ship the second agent.

02 · METRICS

The aggregate view.

Per agent, per hour: tokens · $ · p50/p95 latency · success rate · turn count · tool-call histogram.

Alerts: tokens/task > 2× median · success rate drop > 10% · runaway turn count.

> The dashboard your CFO actually looks at.

03 · EVALS

Regression, not vibes.

30–100 golden Q&A pairs. Run on every prompt change / model swap.

Stack: promptfoo · LangSmith Evaluations · Ragas · Bedrock/Azure/Vertex native eval.

> "It worked yesterday" is not a shipping criterion. The eval says yes or no.

04 · USER FEEDBACK

The ground truth.

Thumbs · comments · corrections. Piped back into the golden set.

The loop: user 👎 → sample into next eval batch → fix prompt → re-run eval → deploy.

> The flywheel that separates L2 from L4. Traces + metrics are dead without it.

Sources: langsmith.com · langfuse.com · arize.com/blog/agents-observability · aws.amazon.com/blogs/machine-learning strands agents observability.

PRE-FLIGHT CHECK

The four ways multi-agent quietly fails.

#	THE TRAP	WHAT ACTUALLY HAPPENS	THE EXIT
01	**Multi-agent when workflow would do.** The "let's have 5 agents!" reflex.	3× the tokens, 5× the latency, unpredictable outputs, hostile debugging — for a problem that was 3 if-statements.	Start with a workflow. Add an agent only when the workflow demonstrably fails.
02	**Framework-first thinking.** "We picked LangGraph, now what's the problem?"	Framework decisions before pattern decisions. Six months in, you're fighting the framework.	Draw the pattern on a napkin first. THEN pick the framework that expresses it cleanly.
03	**No budgets on the loop.** "It'll converge eventually."	Agent goes into 40 turns burning $12 debugging the same off-by-one, then times out with no answer. Bill lands Monday.	Every agent MUST have MAX_TURNS + MAX_TOKENS + MAX_COST + timeout. Log breach.
04	**Ship without HITL gates on irreversible acts.** "We tested it in staging."	Agent sends 200 emails, wires wrong amount, deletes prod table — and there's no undo button. The one story that ends the project.	Wire the runtime approval gate BEFORE the agent gets tool access. Not after v1 ships.

Sources: medium "Multi-Agent in Production 2026 · What Actually Survived" · generativeai.pub "6 Patterns That Work · 2 That Don't" · Anthropic 2026 State of AI Agents.

Find your rung. Pick the next one.

STAGE 0 · ZERO

Single-agent + tools

Behaviour: One agent, ReAct loop, 5-10 tools. No orchestration framework yet.

Stack: plain SDK (Anthropic / OpenAI) + a while-loop.

Signal: "One agent handles 90% of what I need."

> Exit: your single agent's prompt is over 8K tokens and you're still adding rules.

STAGE 1 · REACTIVE

Workflow patterns

Behaviour: Chain / router / parallel patterns. LLM fills slots; you drive.

Stack: LangGraph nodes + edges. First framework decision.

Signal: Token cost drops 30-50% vs. single agent.

> Exit: your workflow needs branches that depend on runtime data the LLM must decide.

STAGE 2 · PROACTIVE

Supervisor + eval + HITL

Behaviour: Supervisor coordinates 2-4 specialists. Golden-set eval in CI. HITL gates on irreversible acts.

Stack: LangGraph-supervisor + LangSmith + promptfoo.

Signal: You debug a wrong answer from traces alone.

> Exit: your supervisor is bottlenecked; other teams want to plug in their own agents.

STAGE 3 · HERO

Platform & harness

Behaviour: An internal harness other teams build agents against. A2A / MCP protocols. Multi-tenant memory. Budget enforcement centralised.

Stack: LangGraph + MCP gateway + Redis/Postgres shared memory + OTel + team SDK.

Signal: Junior engineers ship agents in a day without asking you.

> You're the one Anthropic quotes when they update the essay.

Composite ladder — synthesized from Anthropic State of AI Agents 2026 · LangChain benchmarking · AWS Strands course progression.

THIS SPRINT

Which agent in your stack should have been a workflow?

(a) · The chain in disguise

"Agent" that always does A → B → C.

Rewrite as a prompt chain. Cut tokens 2-3×. Debug becomes trivial.

(b) · The runaway loop

Agent with no budget.

Wire MAX_TURNS + MAX_TOKENS + MAX_COST + timeout this sprint. Alert on breach.

(c) · The missing gate

Autonomous on an irreversible action.

Add a runtime approval gate. Before v1 ships. Before Monday.

▲

My bet is (b). Pick yours — but the honest simplification is worth more than the next agent you'll add.