09: Multiple agents

Building and reviewing with more than one agent

Why this matters

- More agents means more tokens, not a smarter answer
- Reading parallelizes; writing usually doesn't
- Much "multi-agent" gain comes from sampling and voting
- Without a verification step, errors compound
- The best-supported recipe is independent reviewers, then a verifier

Four shapes of "multi-agent"

Shape	How it works	Example
Subagent	A main agent spawns a helper; it returns a summary	Claude Code's Explore agent; Codex subagents
Agent team	Peers share a task list and message each other	Claude Code agent teams (experimental, as of November 2026)
Separate sessions	You run several and carry information between them	One writes, another reviews
Orchestrated pipeline	Code, not a model, decides who runs when	A CI job that runs three review prompts

Four benefits, four costs

Why use more than one	Why not
Parallelism: wall-clock time, not intelligence	Cost: about 15× the tokens of a chat
Context isolation: read 200 files, return a paragraph	Coordination fails: 14 failure modes in 1,600+ traces
Different perspectives: prompts, roles, models	Context lost at every hand-off
Independent verification: a reviewer not swayed by the author	Errors compound: 17.2× vs 4.4× with a central check

Reading parallelizes. Writing usually doesn't.

	Read-heavy work	Write-heavy work
Examples	Research, review, verification, competing hypotheses	Building features in one codebase
Each agent produces	Claims: "line 42 can divide by zero"	Edits that embed decisions
Combining them	Deduplicate, check, discard the wrong ones	Conflicts in git and in design
Two agents disagree	A minute of checking	A broken build

The evidence lines up

- Anthropic built its multi-agent system for research, not coding
- Claude Code's docs recommend teams for "research and review," not same-file edits
- In Kim et al., every multi-agent setup did slightly *worse* on SWE-bench Verified
- At Cursor, twenty agents slowed "to the effective throughput of two or three"

When parallel writing works

On Anthropic's C compiler, many failing tests meant "parallelization is trivial."

The Linux kernel is one giant test. "Every agent would hit the same bug, fix that bug, and then overwrite each other's changes."

Compiling most files with GCC makes each failure point at a few files.

Writing parallelizes when the oracle splits the work into pieces that don't share decisions.

Part 2: In practice

Patterns, three recipes, and budgets

Patterns and when each helps

Pattern	Helps when	Watch out for
Orchestrator–workers	Subtasks can't be predicted in advance	Vague briefs: duplicated or missing work
Fan-out, then merge	The work splits into independent slices	Conflicts surface at the merge
Voting / best-of-N	Answers can be compared or tested	Costs N×; shared blind spots don't vote away
Generator–critic	Clear criteria exist	Critics always find *something*; cap rounds
Debate	Arguments can be checked	Agents cave to persuasive wrong answers
Judge panel	One judge's bias would skew results	Shared biases; calibrate against human labels
Pipeline	Steps are stable and well understood	Early errors flow downstream

Two design choices

Choice	When generating	When reviewing
One model or mixed models?	The best single model usually wins	Mixed families: different blind spots
Shared or isolated context?	Share full context so agents don't diverge	Don't share the author's reasoning

Share the spec and the code. Withhold the author's conversation.

Recipe 1: a review swarm

freeze the target → reviewers by concern → deduplicate → verify each finding → rank → a human decides

Step	What happens
Freeze	One commit or diff, the spec, read-only repo access, for everyone
Fan out	One reviewer per concern: correctness, security, tests, API, simplicity
Deduplicate	Merge findings by location and root cause
Verify	A fresh agent tries to *disprove* each finding
Rank	Confirmed findings, plus plausible ones only if high severity

Verification separates a swarm from noise

```
A reviewer claims: "<finding>".
Your job is to check whether it is real. Trace the code path.
If possible, write a test that demonstrates the failure and run it.
Answer CONFIRMED (with evidence),
PLAUSIBLE (can't prove either way, explain),
or REFUTED (explain why the scenario can't happen).
```

AI critics catch many bugs and invent some. Human-plus-critic teams invent fewer.

Recipe 2: cross-model review

Models recognize their own writing, and tend to prefer it.

- At minimum, review in a fresh context, never the writing session
- Better, have a different lab's model review
- Keep the reviewer strong, or you get confident noise
- Verify across models; cross-family agreement is stronger evidence

Recipe 3: parallel builders, when you must

1. Write the interfaces first, in one human-reviewed change
2. One agent, one worktree, one branch per module
3. Give each agent its own failing tests
4. Merge one at a time: rebase, run the full suite, merge, repeat
5. Or run best-of-N: same task, N worktrees, keep the best

Budgets and stop conditions

- Cap agents, turns, spend, and wall-clock time
- Stop when a full pass finds no new confirmed finding
- Don't loop review-and-fix forever
- Fire noisy reviewers
- Measure tokens and confirmed findings per review

Anti-patterns

Anti-pattern	What goes wrong
Org-chart theater	CEO, CTO, and QA agents: same information, no tests
Debate without a referee	Agreement is not correctness
The telephone game	Detail evaporates at each summary hop
Self-review only	One model, one context, grading itself
Unbounded fan-out	"Use as many agents as you need," no cap

Part 3: What the research says

More sobering than the hype

Debate or vote?

- Sampling several answers and taking the majority is a strong, early baseline
- "Majority Voting alone accounts for most of the performance gains" (Choi et al., 2025)
- Debate can *lower* accuracy as agents abandon right answers (Wynn et al., 2025)
- Self-correction without outside feedback can degrade (Huang et al., 2023)

Why multi-agent systems fail

MAST: 14 failure modes in three groups (Cemri et al., 2025)

Category	Failure modes
System design	Disobeying task or role spec; step repetition; lost history; unaware of termination
Inter-agent misalignment	Resets; no clarifying questions; derailment; withholding; ignoring input; reasoning–action mismatch
Task verification	Premature termination; no or incomplete verification; incorrect verification

Look hardest at the third row.

What the evidence adds up to

1. Sampling plus a vote is the baseline to beat
2. Gains shrink as single models improve
3. Verification is the part that matters
4. Diversity helps judging more than generating
5. Match the architecture to the task's shape

On multi-agent review

> **Bryan's take:** Multi-agent *review* is where I get the most value: several agents, and ideally several models, reviewing each other's work. Building in parallel works for me when the tasks are truly independent. Reviewing in parallel works almost every time.

On which models

> **Bryan's take:** For coding, Claude and Codex are the best right now, in my opinion, so they're the pair I cross-review with: one writes, the other reviews. That's a dated opinion; check module 03 for the current landscape.

Try it

On your own, a three-reviewer swarm by hand:

1. Take a recent PR of 50–300 lines
2. Three fresh sessions: correctness, security, tests
3. Merge and deduplicate the findings
4. Verify each one in a fresh session, ideally another lab's model
5. Tally confirmed findings, false positives, and token cost

As a class or team, compare one agent, three voting, three debating.

Discussion

1. Why do review and research parallelize better than writing code?
2. 90% better at about 15× the tokens: better design, or more compute?
3. If models prefer their own output, what about a one-vendor team?
4. Share full context, or withhold the author's reasoning? Reconcile them.

Read more

- Module 09: Multiple agents building and reviewing
- Anthropic, How we built our multi-agent research system (June 13, 2025)
- Walden Yan, Don't Build Multi-Agents (Cognition, June 12, 2025)
- Cemri et al., Why Do Multi-Agent LLM Systems Fail? (2025)