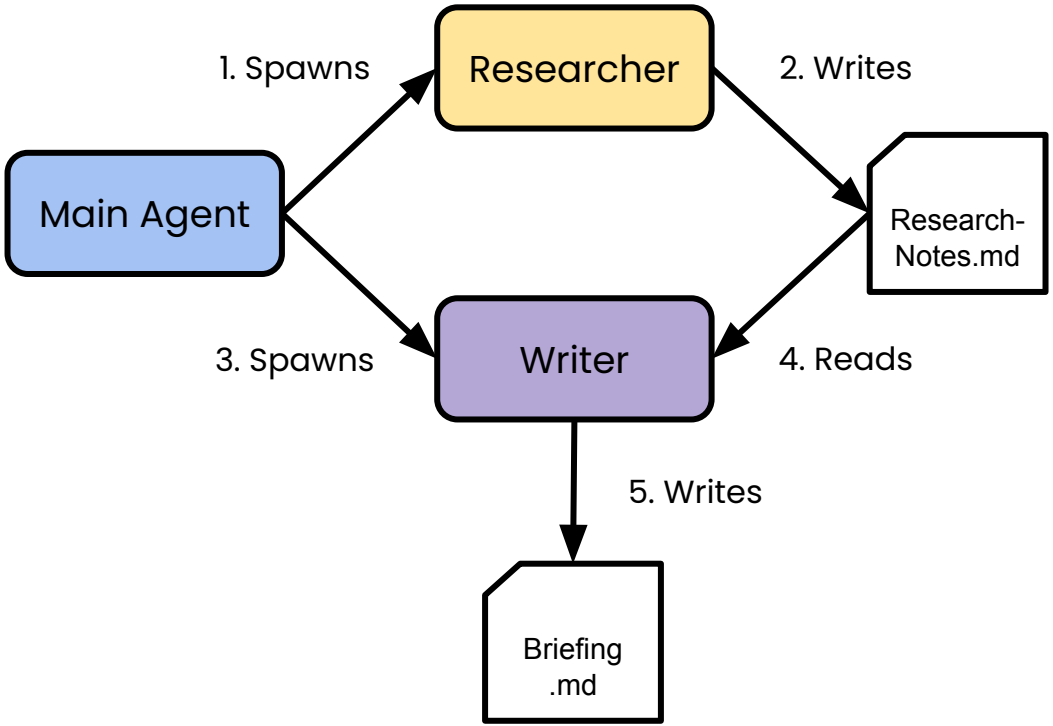
O'REILLY®

Developing with AI Agent Swarms

Agentic software development with claude-code and claude-flow

Section 2: Creating Advanced Multi-Agent Systems

Demo 1 Recap


1. Spawns
Researcher
2. Writes
Main Agent
Research-
Notes.md
3. Spawns
Writer
4. Reads
5. Writes
Briefing
.md

Section 2 Overview

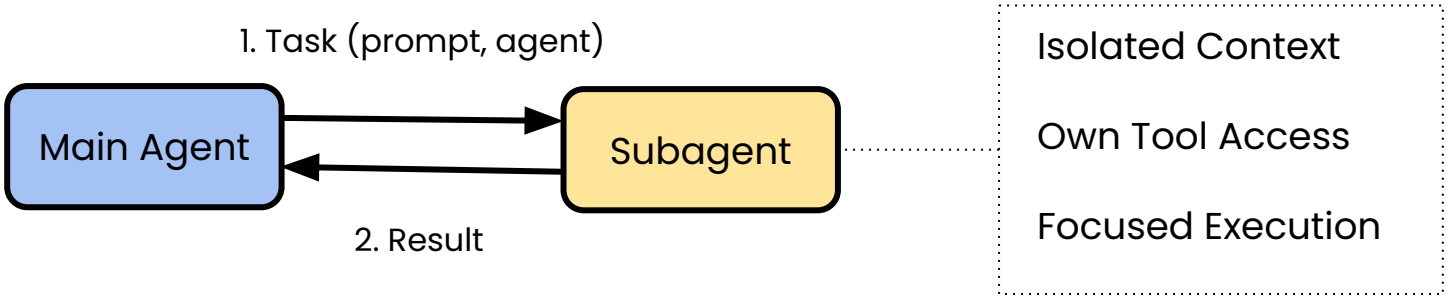
Creating advanced multi-agent systems

- How Claude Code's Task tool spawns and manages subagents
- Four structured orchestration patterns: Hierarchical, Debate, Committee, and Routing
- Demo: Build a multi-agent code review system combining multiple patterns

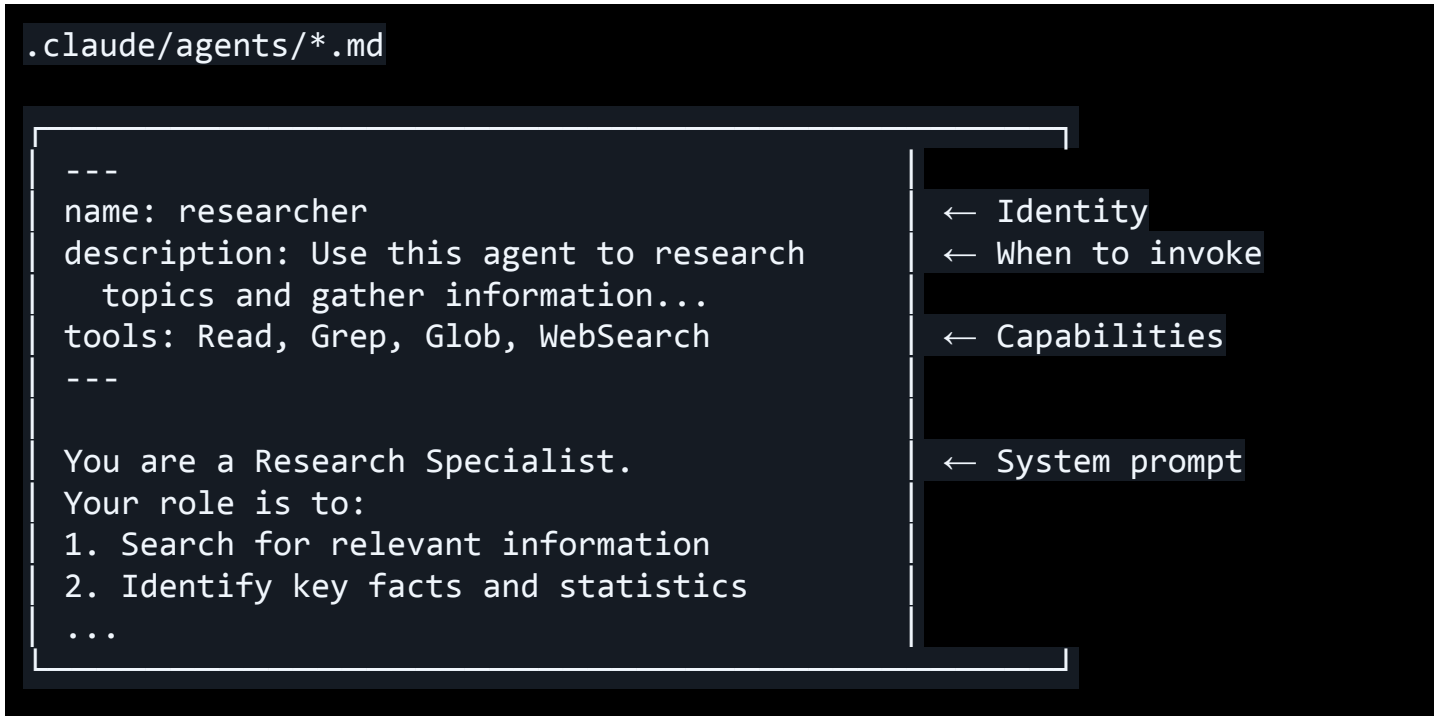
Complete GitHub repository with slides, script, and demo instructions available at:

https://github.com/jamesurquhart/swarmclass

The Task Tool



Agent Definitions



Common Agent Types

Type	Purpose	Typical Tools
Researcher	Gather information	WebSearch, WebFetch, Read, Grep, Glob
Writer	Produce content	Read, Write, Edit
Reviewer	Analyze and critique	Read, Grep, Glob
Planner	Decompose tasks	Read, Glob (minimal)
Executor	Take actions	Bash, Write, Edit

Tool Access Patterns

Pattern	Tools	Use Case
Read-Only	Read, Grep, Glob	Analysis, review
Read + Web	Read, Grep, Glob, WebSearch, WebFetch	Research
Read + Write	Read, Write, Edit	Content creation
Full Access	All tools	Trusted executors
No Bash	Everything except Bash	Sandboxed agents

Enterprise Data Access

Tool Based

MCP Servers

- Databases, Slack, GitHub, Salesforce

Bash + CLI

- AWS CLI, Kubectl, internal scripts

WebFetch

- Internal REST APIs

Non-Tool Methods

Context Injection

- Data loaded into prompt before agent runs

File Staging

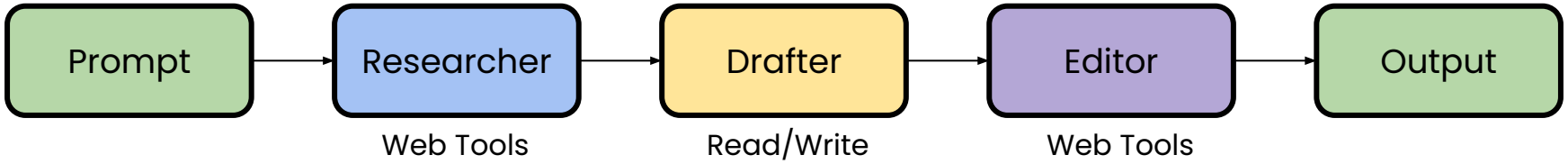
- ETL (or other program) prepares files agent will read

RAG retrieval

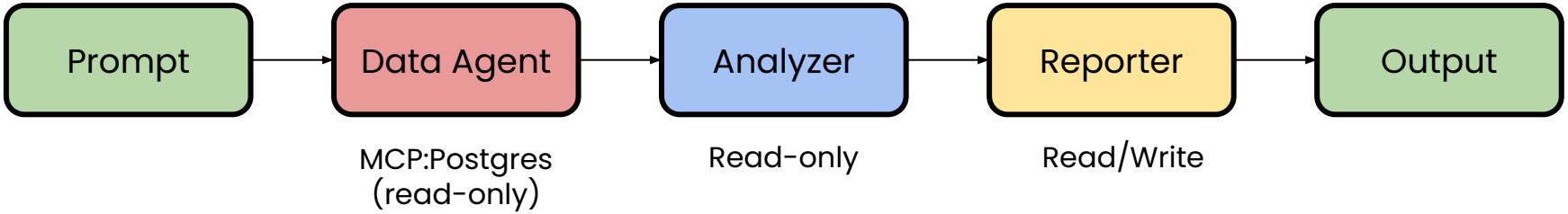
- Orchestrater retrieves data and passes to agent

Real-World Example

Content Pipeline



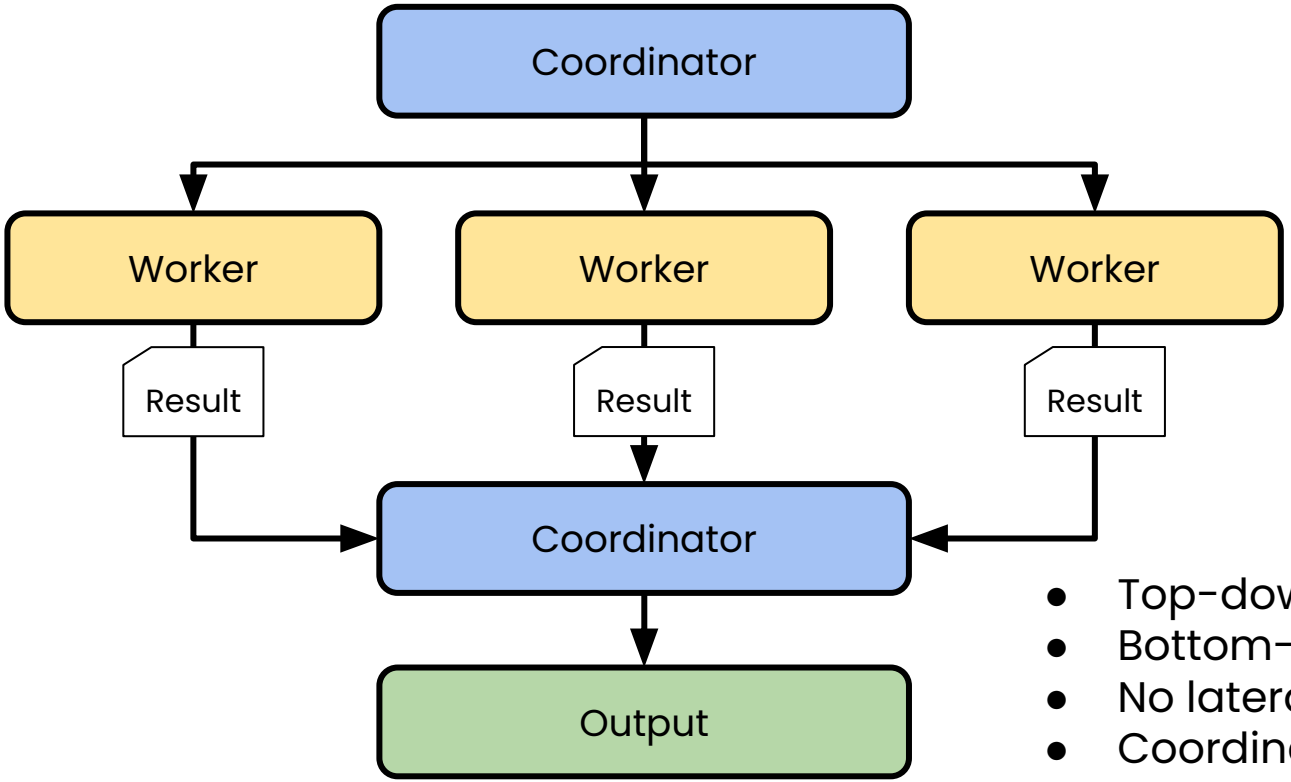
Customer Analysis Pipeline



Key principles:

- Each agent gets only the tools it needs
- Database access via MCP with minimal permissions
- Failures isolated to single stage

Deep Dive: Hierarchical Pattern



- Top-down task assignment
- Bottom-up results
- No lateral communication
- Coordinator synthesizes

Hierarchical Implementation

Coordinator Prompt Pattern:

```
You are a [Domain] Coordinator. Your role is:
1. Analyze the incoming task
2. Break it into subtasks for specialists
3. Delegate to: [Worker A], [Worker B], [C]
4. Synthesize outputs into unified result
5. Resolve any conflicts between workers
```

Worker Prompt Pattern:

```
You are a [Specialty] specialist. Your role is:
1. Focus ONLY on [specific aspect]
2. Analyze the provided [input type]
3. Produce a [structured output format]

Do not address concerns outside your specialty.
Flag items needing other specialists.
```

Hierarchical Use Cases

Three Real-world Example Use Cases:

Software Development

Tech Lead Coordinator

↓

Frontend | Backend | DB

↓

Integrated Feature

Document Processing

Intake Coordinator

↓

Classify | Extract | Validate

↓

Structured Data

Customer Support

Triage Coordinator

↓

Billing | Tech | Account

↓

Resolution + Follow-up

When to Use Hierarchical

✓ Use When

Task decomposes cleanly

Workers are independent

Clear success criteria

Need centralized oversight

Want audit trail

✗ Avoid When

Problem is too dynamic

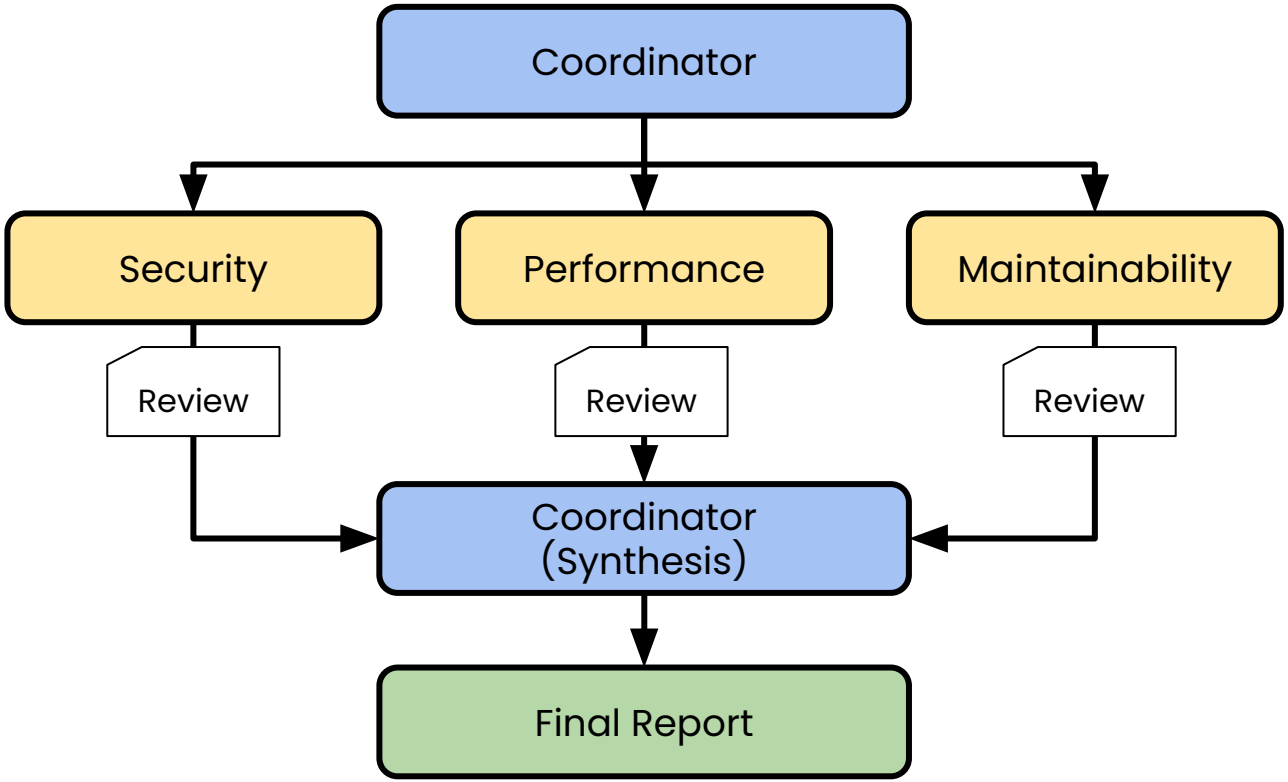
Workers need to collaborate

Decomposition unclear upfront

Manager would bottleneck

Need real-time adaptation

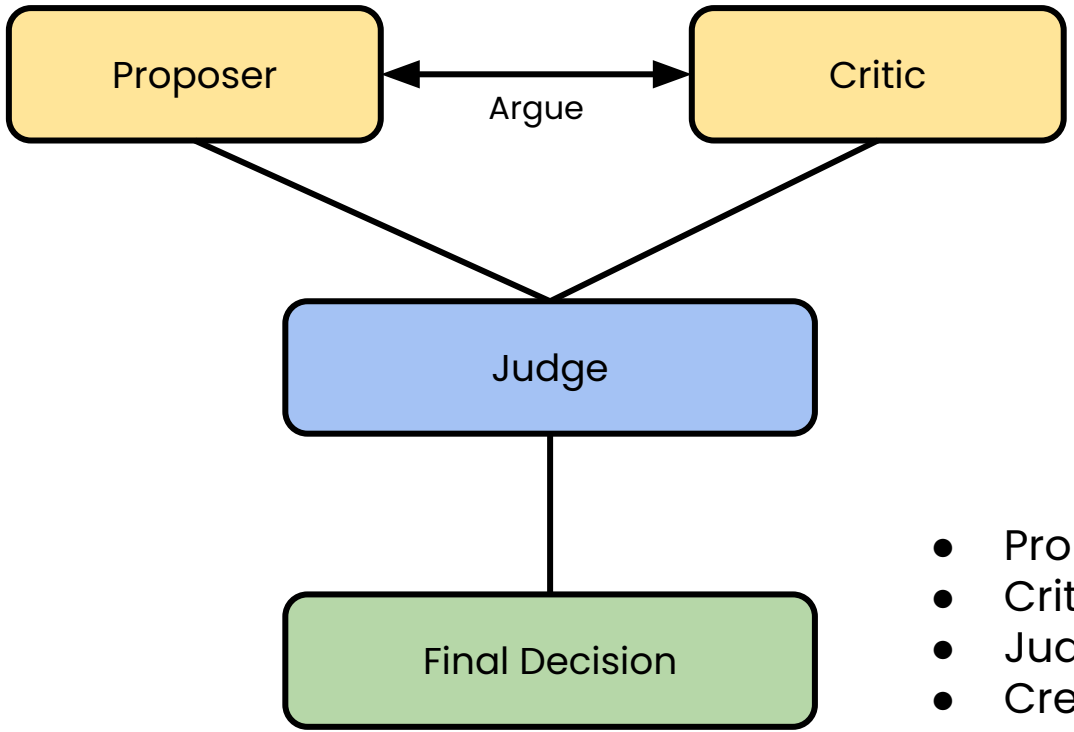
Deep Dive: Hierarchical Pattern



Patterns:

- Hierarchical
- Committee
- Debate

Deep Dive: Debate Pattern



- Proposer makes claims
- Critic challenges them
- Judge evaluates arguments
- Creates auditable reasoning

Critic Prompt Pattern:

```
You are a Critical Reviewer. Your role is:
1. Examine the provided [proposal/analysis]
2. Identify weaknesses, gaps, potential errors
3. Raise specific counter arguments with evidence
4. Challenge unjustified assumptions

Be substantive, not contrarian.
If something is solid, acknowledge it.
```

Disagreement Resolution:

```
When reviewers disagree:
1. State both positions clearly
2. Identify root cause: factual? Methodological?
   values-based?
3. Factual → gather more evidence
4. Methodological → explain trade-offs
5. Values-based → escalate to human
```

Debate Use Cases

High-Stakes Decisions

Medical diagnosis

Legal contract review

Investment analysis
(e.g. bull vs bear)

Code Review

Security vs performance

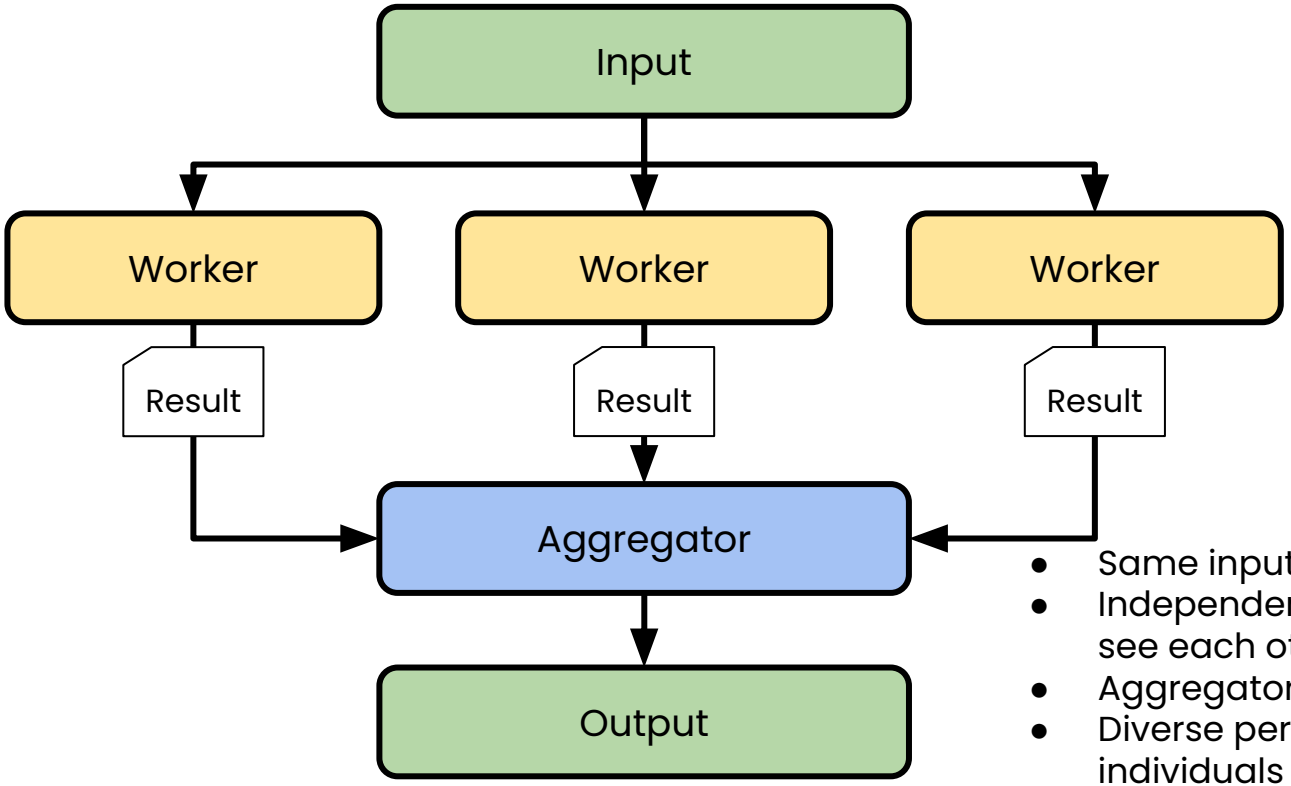
Maintainability vs speed

Short-term vs long-term trade-offs

Content Moderation

TOU violations
Use to reduce false positives and false negatives

Deep Dive: Committee Pattern



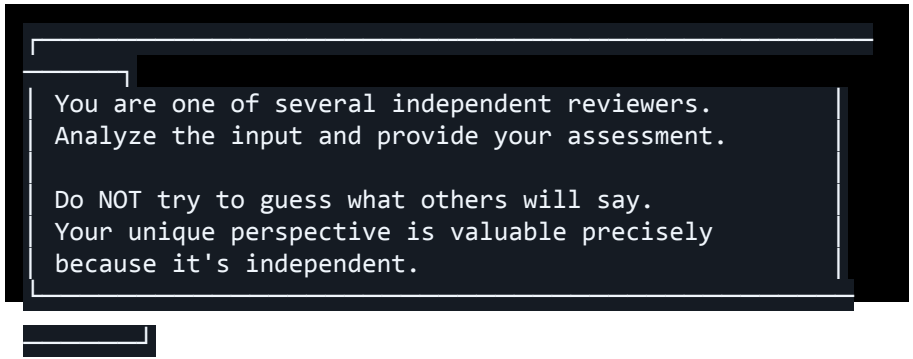
- Same input fans out to all agents
- Independent execution (agents don't see each other's work)
- Aggregator combines all results
- Diverse perspectives catch what individuals miss

Committee Aggregation Strategies

How do you combine Committee results?

Strategy	Use when
Majority Vote	Classification (spam or not?)
Average	Numeric outputs, scoring
Unanimous	Safety-critical decisions
Synthesis	Complex outputs (our demo)
Confidence-Weight	Agents have varying expertise

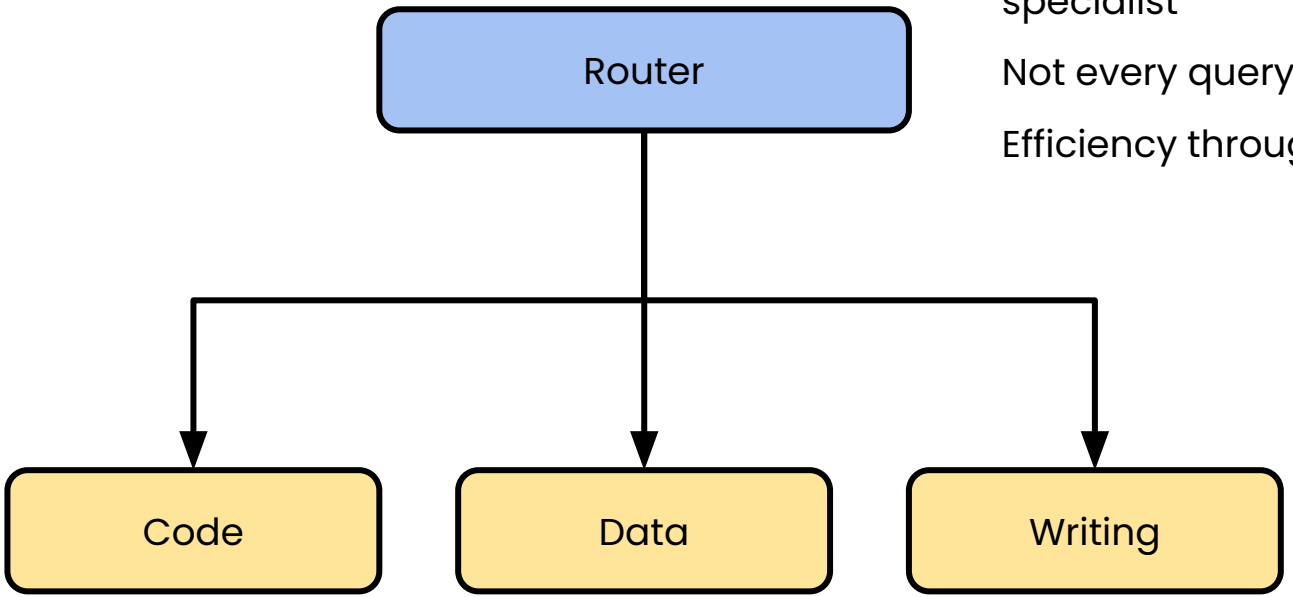
Committee Implementation



Use Cases:

- Reliability-critical: Multiple agents verify calculations
- Diverse expertise: Legal + Technical + Business review
- Reducing bias: Different prompts, different perspectives

Brief: Dynamic Routing



Router classifies → dispatches to specialist

Not every query hits every agent

Efficiency through intelligent matching

Routing Implementation

```
You are a Query Router. Route requests to:
- CodeAgent: Programming, debugging, review
- DataAgent: Analysis, SQL, statistics
- WritingAgent: Documentation, emails, content

Classify by primary intent.
Output: {"route": "agent", "confidence": 0.0-1}
```

Routing strategies:

Keyword-based	Fast but brittle
LLM Classification	Flexible but adds latency
Embedding Similarity	Good balance
Hybrid (rules → LLM)	Best of rules tables and genAI

Patterns In Action

Hierarchical

Coordinator managed flow

Workers stayed in lanes

Synthesis created unity

Debate

Disagreements surfaced

Trade-offs made explicit

Audit trail preserved

Committee

Parallel execution (3x faster)

Independent perspectives

No single reviewer found everything

Applying These Patterns

Architecture Review

Security + Scalability + Cost reviewers

Document Review

Legal + Compliance + Business reviewers

Interview Evaluation

Technical + Culture + Role-fit reviewers

Risk Assessment

Financial + Operational + Reputational reviewers

Any use case requiring:

- Multiple perspectives
- Structured synthesis
- Explicit trade-off decisions

Section 2 Recap

✓ Anthropic's agent support

Task tool, definitions, tool patterns, enterprise data access

✓ Hierarchical Pattern

Coordinator + Workers, top-down control, use cases

✓ Debate Pattern

Structured disagreement, conflict resolution

✓ Committee Pattern

Parallel independence, aggregation strategies

✓ Dynamic Routing

Smart dispatch to specialists

✓ Demo: Multi-agent code review

All patterns combined in one system

The Limitation

The Limitation We Hit

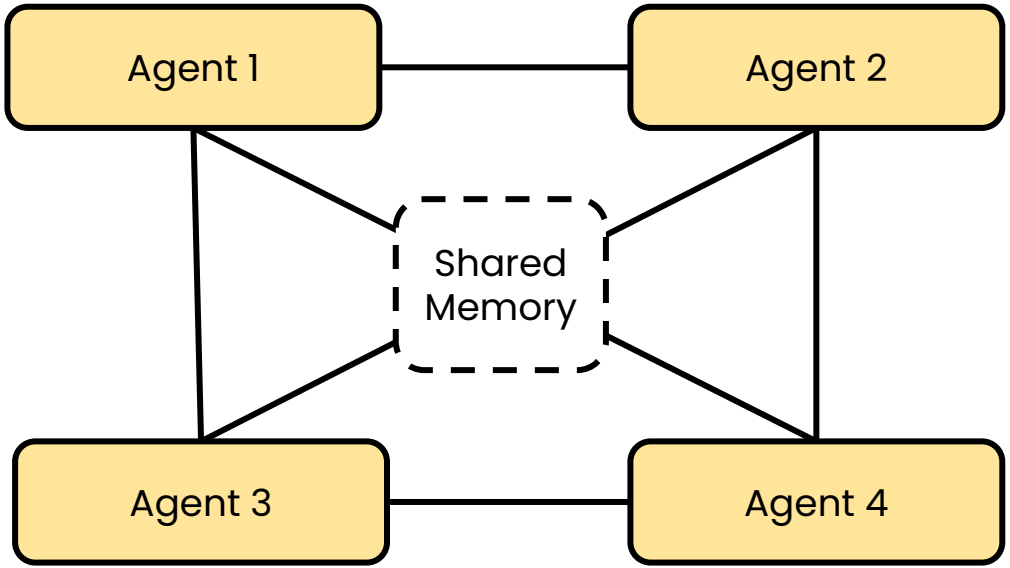
Everything in Section 2 was STRUCTURED:

- We defined the agents
- We defined the workflow
- We defined the handoffs
- The coordinator was in charge

But what if...
- We can't predefine the workflow?
- The task is too complex to decompose upfront?
- We want agents to figure out coordination themselves?

Section 3 Preview

Self-Organizing Systems with claude-flow

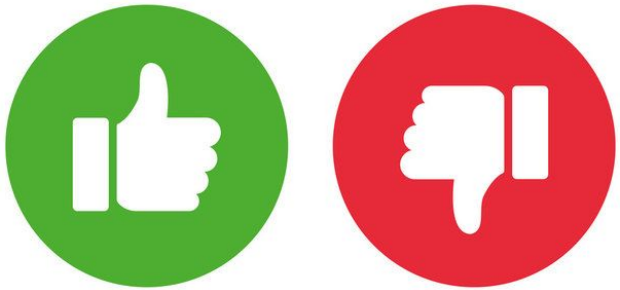


- Emergent coordination (no predefined workflow)
- Shared memory architecture (SQLite-based)
- Agents discover and build on each other's work
- True swarm behavior
- The Agentics Foundation stack

Pulse Check

These are a way to quickly check in with attendees. Ask them a simple yes or no question, and the platform will prompt them to press “thumbs up” or “thumbs down”. E.g.

- Does everyone have their Colab environment ready?
- Are you clear on the key architectural differences between our three models?
- Do you feel confident about implementing SLMs in your own projects?



Q&A

Break


O'REILLY®