ANTHROP\C

Agent Orchestration

University of Pennsylvania Claude Builder Club

March 24th, 2026

Today’s Presenter

Albert Opher
CBC Founder | M&T ‘25

LLMs are single-task specialists and multi-task nightmares

CONFIDENTIAL

LLMs to Agents

Companies are spending more time on **tailoring LLMs to specialize on specific tasks**

Model providers

Model providers are focussed on two buckets:

1. Broad model capability improvements:
    a. Reasoning
    b. Thinking
    c. Size & Compute Per Token
2. Creating repeatable, accurate LLM workflows that progress toward AGI
    a. Single Task Agents
    b. Multi-Task Agents (Fleets)
    c. Autonomous Task Selection and Tool Calling

What Is an AI Agent?

Anatomy of an AI Agent

An agent is an LLM in a loop — it perceives, thinks, acts, and observes until a task is done.

Perceive

Reads user input, tool outputs, context

Think

LLM reasons about what to do next

Act

Calls tools, writes code, makes API calls

Observe

Evaluates results, decides if done

Key distinction: A chatbot responds once. An agent loops autonomously until the task is complete.

Where Single Agents Break Down

Single agents struggle with complexity — this is why we need orchestration.

Context Pollution

Long multi-step tasks bloat the context window. The agent loses focus and starts hallucinating or forgetting earlier steps.

Expertise Mismatch

One system prompt can't make an agent simultaneously great at coding, data analysis, writing, and customer support.

Fragile at Scale

A single point of failure means one bad tool call or hallucination derails the entire workflow. No graceful degradation.

Latency Bottleneck

Tasks that could run in parallel (e.g. analyzing a doc from 3 perspectives) are forced to run sequentially.

Core Concepts

Orchestration Patterns

The Five Orchestration Patterns

Building blocks for multi-agent systems, from simple to complex.

01	**Prompt Chaining**	Sequential pipeline — output of one step feeds the next with quality gates in between
02	**Routing**	Classifier dispatches input to the right specialized agent based on intent
03	**Parallelization**	Fan-out work to multiple agents simultaneously, then aggregate results
04	**Orchestrator-Workers**	Central planner dynamically delegates subtasks to worker agents
05	**Evaluator-Optimizer**	Generator + critic loop that iterates until quality threshold is met

Pattern 1: Prompt Chaining

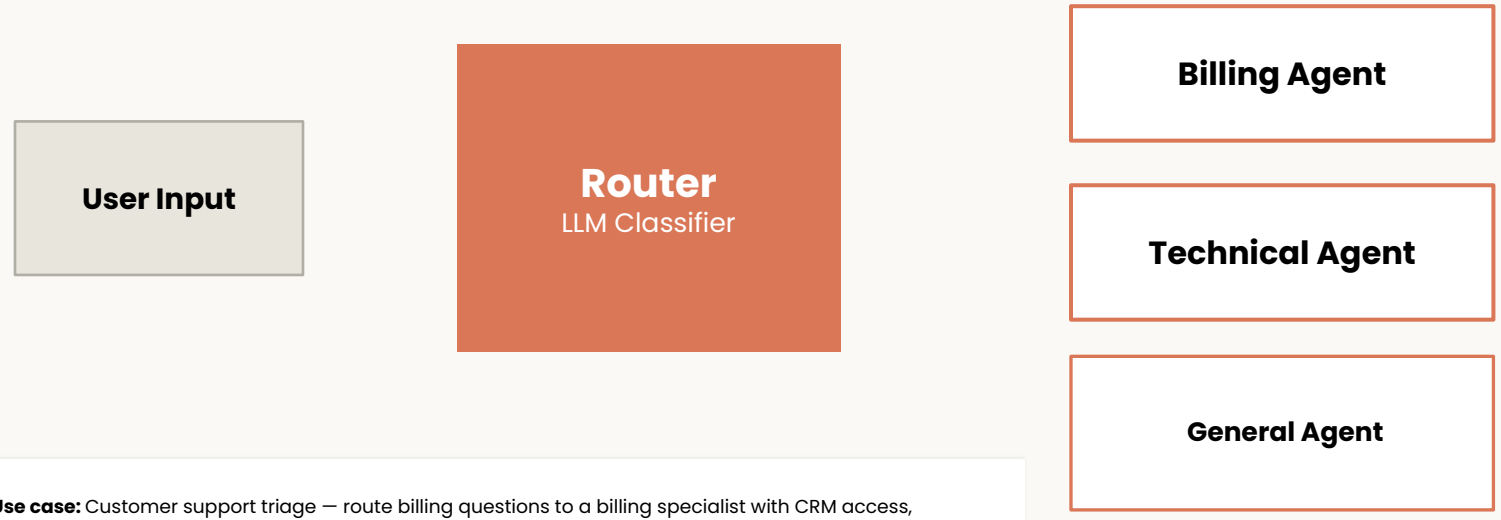
Decompose a task into a fixed sequence of steps, with optional quality gates between them.

Use case
Content generation pipeline: research a topic, draft an article, edit for tone, format for publication.

When to use
Tasks that are naturally sequential with clear handoff points. Each step's output is the next step's input. Gates catch errors early.

Pattern 2: Routing

A classifier examines the input and dispatches to the right specialized agent.



Use case: Customer support triage — route billing questions to a billing specialist with CRM access, technical issues to a debugging agent with log access, and general inquiries to a knowledge-base agent.

Pattern 3: Parallelization

Run multiple agents simultaneously and aggregate results. Two flavors:

Sectioning

Split a task into independent subtasks, run them in parallel, combine outputs.

Example: Analyze a contract from legal, financial, and compliance perspectives simultaneously.

```
Task → Agent A + Agent B + Agent C → Merge
```

Voting

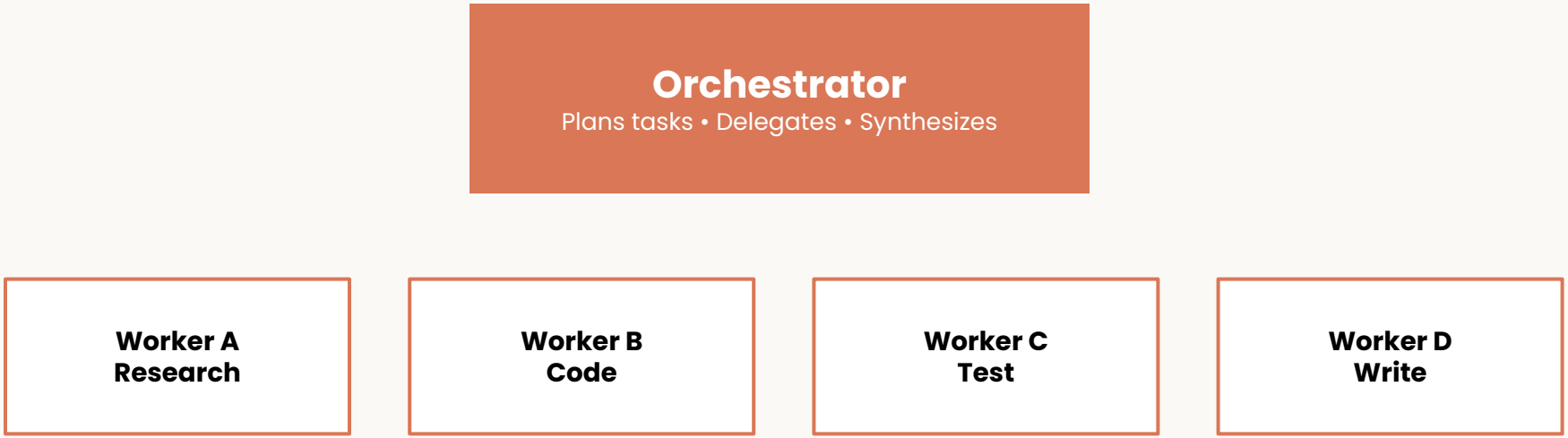
Same task given to multiple agents, results aggregated (majority vote, best-of-N, consensus).

Example: 3 agents classify customer sentiment, majority vote determines the label.

```
Task → Agent × 3 → Vote → Answer
```

Pattern 4: Orchestrator-Workers

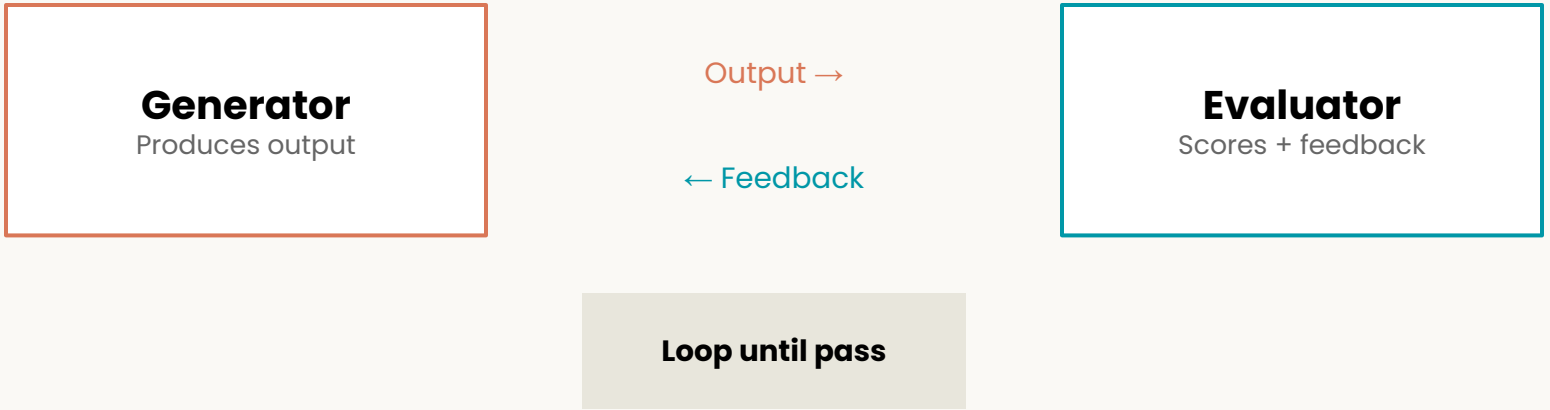
A central LLM dynamically plans, delegates to workers, and synthesizes results.



Key advantage: The orchestrator decides at runtime how many workers to spawn and what each does — most flexible pattern.

Pattern 5: Evaluator-Optimizer

One agent generates output, another evaluates it. Loop until quality threshold is met.



Use cases: Code generation with automated test validation • Writing with editorial review • Translation with quality scoring

When to Use What

Start simple. Only add complexity when a simpler pattern fails.

Pattern	Best For	Complexity	Tradeoff
Prompt Chaining	Fixed sequential workflows	Low	Rigid — can't adapt at runtime
Routing	Multi-domain input classification	Low	Only as good as the classifier
Parallelization	Independent subtasks or voting	Medium	Aggregation logic can be tricky
Orchestrator-Workers	Complex, dynamic task planning	High	Expensive — extra LLM calls
Evaluator-Optimizer	Quality-critical iterative work	High	Latency from multiple loops

Implementation

Agent Frameworks

Framework Landscape

The major frameworks for building multi-agent systems today.

LangGraph	LangChain	Graph-based orchestration. Agents are nodes, edges define control flow. Explicit state management.
CrewAI	CrewAI Inc.	Role-based multi-agent framework. Define agents with roles, goals, and backstories. Sequential or hierarchical.
AutoGen	Microsoft	Conversational multi-agent framework. Agents communicate via message passing. Highly configurable.
OpenAI Swarm	OpenAI	Lightweight handoff pattern. Minimal abstraction — agents transfer control via function calls. Educational.
Claude Native	Anthropic	Tool use + MCP + extended thinking. No framework needed — orchestrate directly via the API with tool calls.

LangGraph

Graph-based agent orchestration with explicit state management and conditional routing.

Key Concepts

Nodes
Each node is a function (often an LLM call). Agents, tools, and logic live here.

Edges
Define control flow. Conditional edges let the graph branch based on state.

State
A shared TypedDict passed through the graph. Persists across nodes. Checkpointing built in.

```python
from langgraph.graph import StateGraph

graph = StateGraph(AgentState)

graph.add_node("research", research_agent)
graph.add_node("write", writing_agent)
graph.add_node("review", review_agent)

graph.add_edge("research", "write")
graph.add_conditional_edges(
    "review",
    should_revise,
    {"revise": "write", "done": END}
)

app = graph.compile()
```

CrewAI

Role-based multi-agent framework. Define agents with personas, then assign them tasks.

Key Concepts

Agents
Defined by role, goal, and backstory. Each agent gets its own persona and tools.

Tasks
Assigned to agents with a description and expected output format.

Crew
Manages the agents and tasks. Runs sequentially or in a hierarchical process.

```
from crewai import Agent, Task, Crew

researcher = Agent(
    role="Senior Researcher",
    goal="Find key market trends",
    backstory="Expert analyst...",
    tools=[search_tool]
)

task = Task(
    description="Research AI market",
    agent=researcher,
    expected_output="Report with data"
)

crew = Crew(
    agents=[researcher, writer],
    tasks=[task, write_task],
    process=Process.sequential
)
```

AutoGen & OpenAI Swarm

AutoGen (Microsoft)

Conversational multi-agent framework where agents communicate via message passing.

Strengths:
- Highly configurable agent behaviors
- Human-in-the-loop support
- Group chat orchestration
- Code execution sandboxing

Best for: Complex, configurable multi-agent conversations with human oversight.

Swarm (OpenAI)

Minimal agent handoff pattern. Agents transfer control to each other via function returns.

Strengths:
- Dead simple API (~100 lines of core code)
- Transparent handoff mechanism
- Easy to understand and extend
- Great for learning orchestration concepts

Best for: Learning, prototyping, and simple handoff patterns. Explicitly not production-grade.

Framework Comparison

	LangGraph	CrewAI	AutoGen	Swarm
Abstraction	Graph (nodes + edges)	Roles + Tasks	Conversational agents	Function handoffs
State Mgmt	Built-in TypedDict + checkpoints	Shared context object	Message history	Context variables
Learning Curve	Medium-High	Low-Medium	High	Very Low
Flexibility	Very high — arbitrary graphs	Medium — sequential or hierarchical	High — highly configurable	Low — intentionally minimal
Production Ready	Yes	Yes	Yes (v0.4+)	No (educational)
Best Use Case	Complex stateful workflows	Team-of-agents metaphor	Multi-agent conversations	Simple handoff prototypes

Real World

Building Reliable Agents

Failure Modes in Multi-Agent Systems

Know the ways your system will break before it breaks.

Infinite Delegation Loops

Agent A delegates to Agent B, which delegates back to A. No termination condition. Your bill goes vertical.

Context Corruption

State gets mangled across agent handoffs. Agent B receives garbled context from Agent A and confidently hallucinates.

Cascading Failures

One agent fails, downstream agents operate on bad data. Errors compound through the chain. No circuit breaker.

Cost Explosion

Orchestrator spawns too many workers, or evaluator loops too many times. A $0.10 task becomes a $15 task.

Best Practices

Keep agent scopes narrow	Each agent should do one thing well. A research agent researches. A coding agent codes. Don't build Swiss Army knife agents.
Implement circuit breakers	Set max iterations, token budgets, and timeout limits. If an agent loops more than N times, kill it and surface the error.
Use structured outputs between agents	Agents should pass JSON or typed objects, not free-form text. This prevents context corruption at handoff boundaries.
Log everything (observability)	Every agent call, tool invocation, and state transition should be logged. Use tools like LangSmith, Braintrust, or Weights & Biases.
Test with deterministic inputs first	Before deploying, run your agent graph on a fixed set of inputs with known expected outputs. Measure pass rates.
Start with a single agent	Only add orchestration when a single agent demonstrably fails at the task. Premature multi-agent design adds cost and complexity.

What's Next for Agent Orchestration

Agent Fleets

Persistent pools of specialized agents that can be dynamically assembled for any task. Companies like Hebbia and Cognition are leading here.

Long-Term Memory

Agents that remember across sessions. Shared memory stores, vector DBs, and persistent state enable agents that learn over time.

MCP as the Tool Layer

MCP standardizes how agents connect to tools. Orchestrated agents can dynamically discover and use any MCP server's capabilities.

Autonomous Task Selection

Agents that don't just execute tasks but choose which tasks to work on. Moving from reactive to proactive AI workflows.

Resources

Anthropic: Building Effective Agents	https://docs.anthropic.com/en/docs/build-with-claude/agent
LangGraph Documentation	https://langchain-ai.github.io/langgraph/
CrewAI Documentation	https://docs.crewai.com
AutoGen Documentation	https://microsoft.github.io/autogen/
OpenAI Swarm (GitHub)	https://github.com/openai/swarm

Join Penn CBC Slack: https://join.slack.com/t/penncbc/shared_invite/zt-3o5dxtpeq-DJNBfeM_C~EUxH4hVLZFgw

ANTHROPIC