

CSE/DSC 234

Data-Centric AI and AI Engineering

Arun Kumar

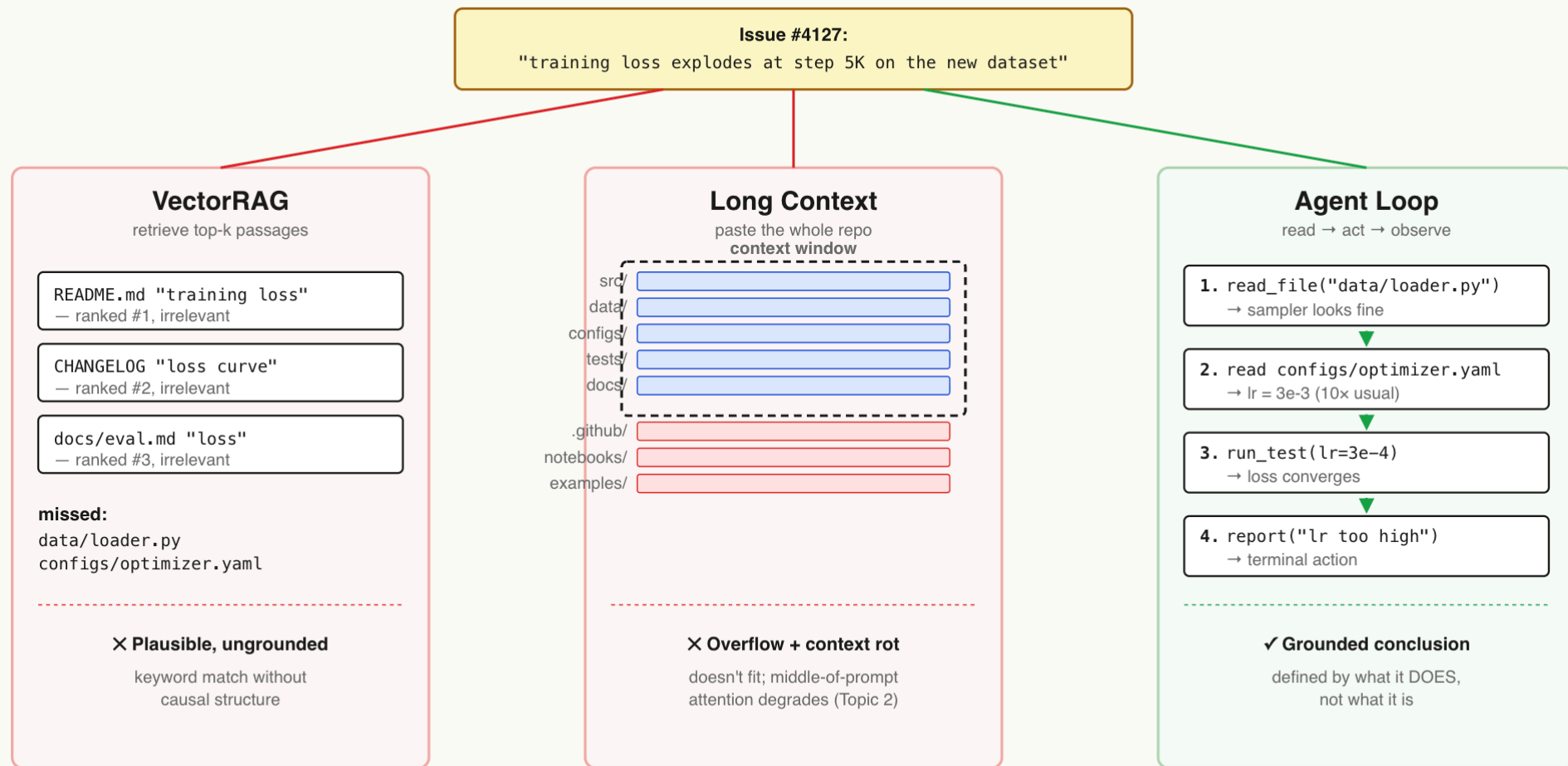
Topic 3: Agent Engineering

Outline

- ➔ ❖ Motivation
 - ❖ Design Foundations
 - ❖ Harness Eng. and Components
 - ❖ Multi-Agent Workflows
 - ❖ Observability and Evals

Running Example

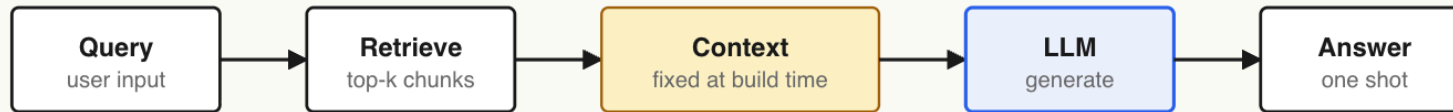
❖ Example: Triaging a GitHub issue for an AI training codebase



Beyond Context/Retrieval Eng.

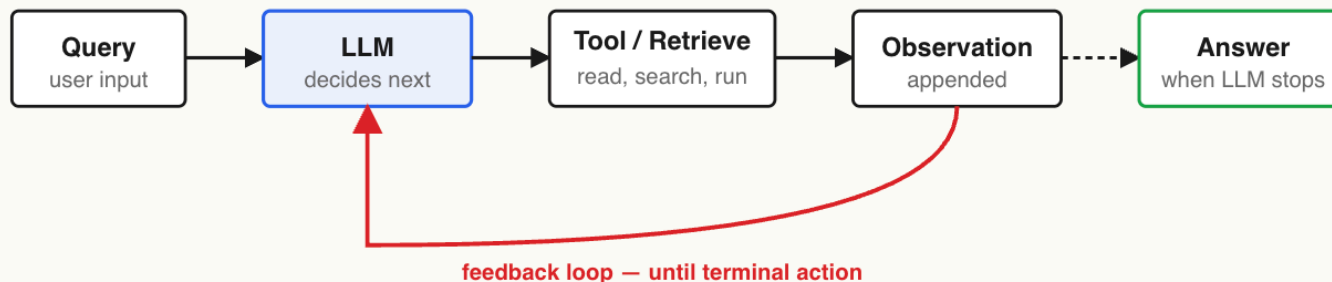
- ❖ Fundamentally, context/retrieval eng. is a *static chain* workflow
- ❖ Agentic workflows add *dynamism* and/or *autonomy* to enable multi-turn context creation and usage

Topic 2 — pick context ONCE, then generate

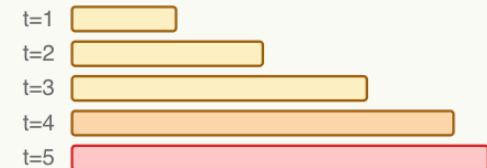


*linear, one-pass.
cheap, predictable,
easy to evaluate.*

Topic 3 — pick context ITERATIVELY, conditioned on what you've seen



Context grows each pass



rot risk on critical path

New Risks of Agentic Workflows

- ❖ **Risk of Compounding Errors:**
 - ❖ Bad result at one turn could poison all subsequent turns
- ❖ **Termination Condition:**
 - ❖ Tradeoffs of when to stop; too early: unfinished or hallucinated work; too late: wasted token spend, latency
- ❖ **Higher Risk of Context Rot:**
 - ❖ Context keeps growing every turn; context rot risk is *on the critical path* by design

Outline

- ❖ Motivation
- ➔ ❖ Design Foundations
- ❖ Harness Eng. and Components
- ❖ Multi-Agent Workflows
- ❖ Observability and Evals

Pivot Point: ReAct as Core Loop

- ❖ **Key Idea:** Mix the capabilities of *Reasoning*-oriented and *Action*-oriented LLM usage; has become de facto standard

(1) Hotspot QA

Question: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

(1a) Standard

Answer: iPod



(1b) CoT (Reason Only)

Thought: Let's think step by step. Apple Remote was originally designed to interact with Apple TV. Apple TV can be controlled by iPhone, iPad, and iPod Touch. So the answer is iPhone, iPad, and iPod Touch.

Answer: iPhone, iPad, iPod Touch



(1c) Act-Only

Act 1: Search[Apple Remote]

Obs 1: The Apple Remote is a remote control ...

Act 2: Search[Front Row]

Obs 2: Could not find [Front Row]. Similar: ...

Act 3: Search[Front Row (software)]

Obs 3: Front Row is a discontinued media center software ...

Act 4: Finish[yes]



(1d) ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.

Act 1: Search[Apple Remote]

Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.

Act 2: Search[Front Row]

Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]

Thought 3: Front Row is not found. I need to search Front Row (software) .

Act 3: Search[Front Row (software)]

Obs 3: Front Row is a discontinued media center software ...

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.

Act 4: Finish[keyboard function keys]



Why Combine Reasoning + Action?

- ❖ Neither is enough on its own:
 - ❖ **Reasoning-only:** No ground truth; often confidently wrong
 - ❖ **Action-only:** No deliberation; often wanders, doesn't recover
- ❖ **ReAct:**
 - ❖ Reasoning steers action; actions lead to new observations; observations steer reasoning
 - ❖ Virtuous feedback loop; each constrains the other
 - ❖ Loosely, akin to a RAG setup where the LLM writes its own queries, iteratively, conditioned on what it produces!

Running Example

Issue #4127:

"training loss explodes at step 5K on the new dataset"

□ thought □ action (tool call) □ observation

THOUGHT "Check the data loader first."

ACTION `read_file("data/loader.py")`

OBSERV. [file contents – RandomSampler, batch_size=64]

THOUGHT "Sampler's fine. Check the optimizer."

ACTION `read_file("configs/optimizer.yaml")`

OBSERV. optimizer: AdamW, lr: 3e-3, weight_decay: 0.01

THOUGHT "lr=3e-3 – 10× the usual. Likely cause."

ACTION★ `report_finding("learning rate suspect")`

observation
steers next thought

observation
steers next thought

★ loop terminates

Key Failure Modes of Core Loop

1. **Compounding Errors**
2. **Termination Condition**
3. **Tool Misfire:** Wrong tool(s), then reasons confidently with that
4. **Hallucinated Observations:** LLM writes its own “results” without calling correct tools
5. **Cost/Latency Blowup:** 3-step task can blow up to 50 iterations!

❖ **Defenses involve tradeoffs:**

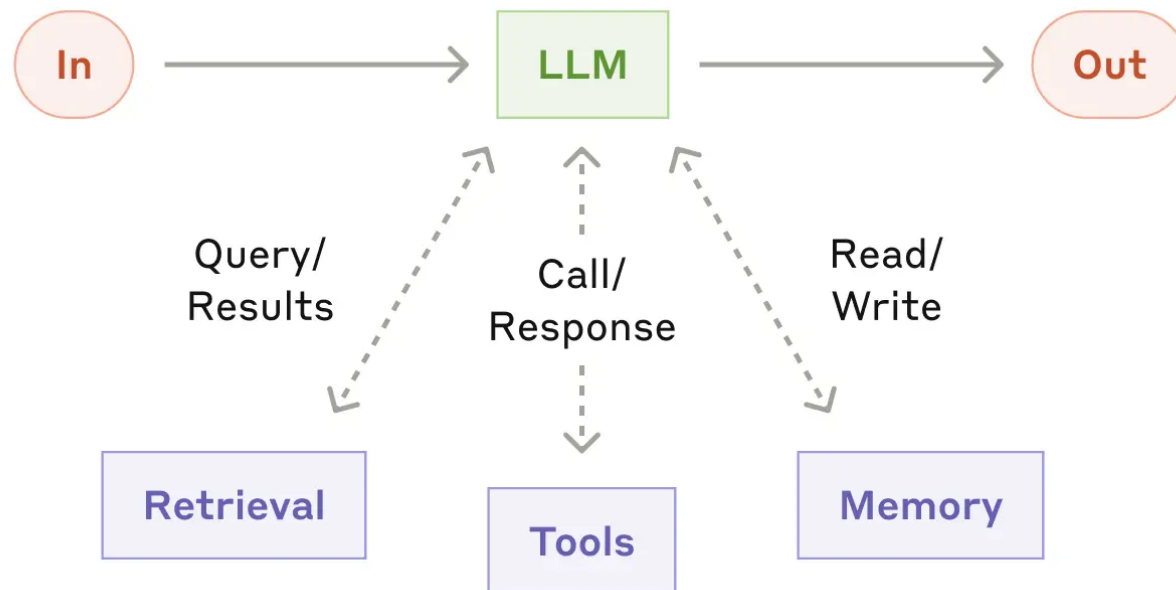
- ❖ Step budgets
- ❖ Validated schemas
- ❖ Observation echo checks
- ❖ Retry with reflection (cf. “Reflexion”)

Outline

- ❖ Motivation
- ❖ Design Foundations
- ➔ ❖ Agentic Workflows
- ❖ Harness Eng. and Components
- ❖ Multi-Agent Workflows
- ❖ Observability and Evals

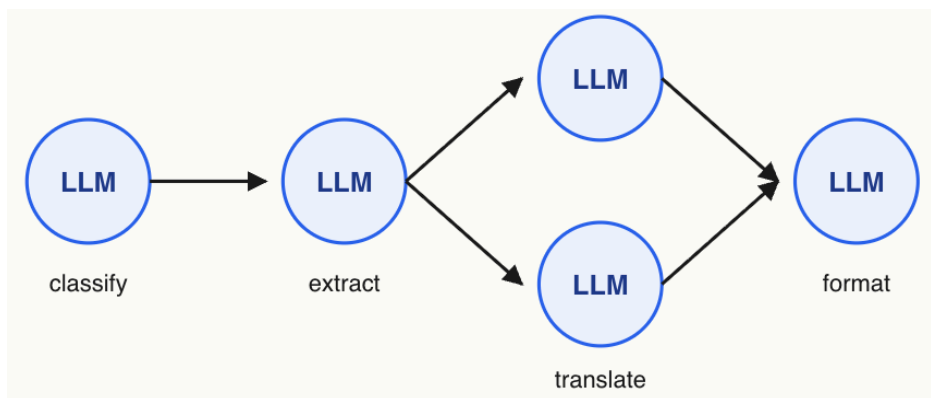
Core Component: Augmented LLM

- ❖ LLM does not just look at input but it is empowered to interact with the *environment* in both I/O directions:

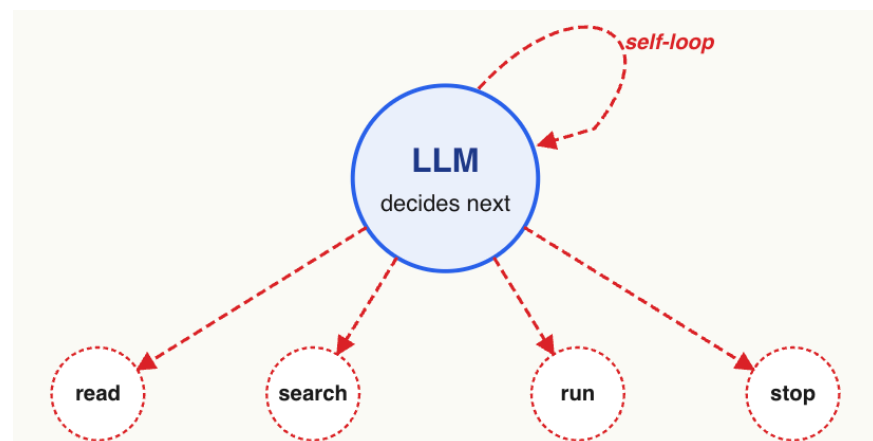


Workflow vs. Agent

- ❖ Boils down to deterministic vs. probabilistic *control flow*
- ❖ Workflow is fixed graph; “agent” offers LLM more autonomy



Vertices and edges fixed at design time;
some edges/paths can be conditional

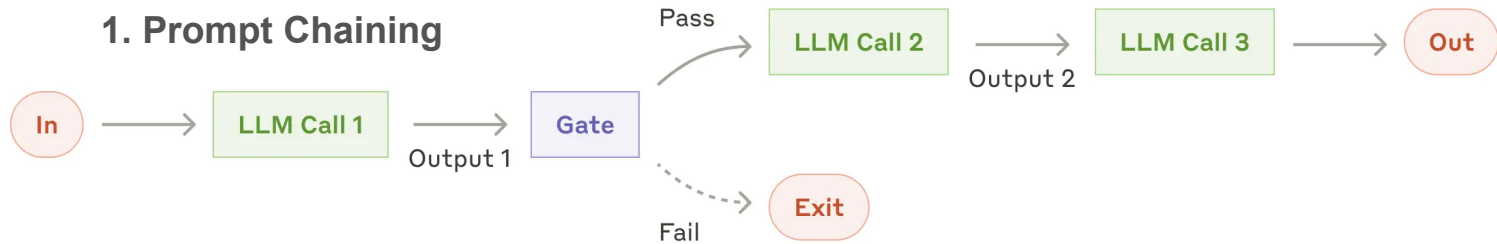


Vertices and edges chosen at run time by LLM
based on content of query / intermediates

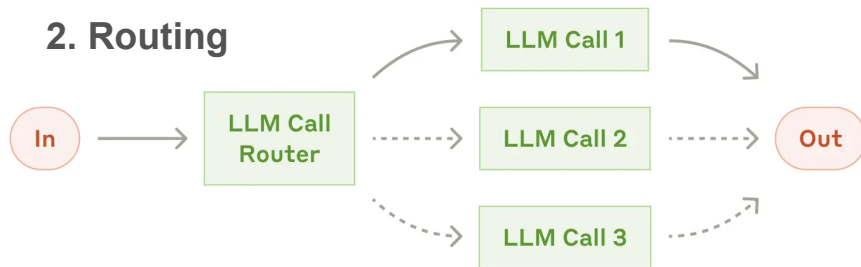
	Workflow	Agent
Structure	Predictable	Flexible
Cost	Cheaper	Costlier
Debugging / Evals	Easier to do	Harder to do

5 Major Workflow Design Patterns

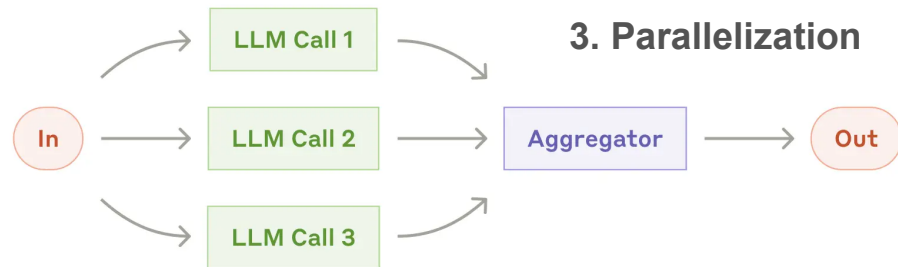
1. Prompt Chaining



2. Routing



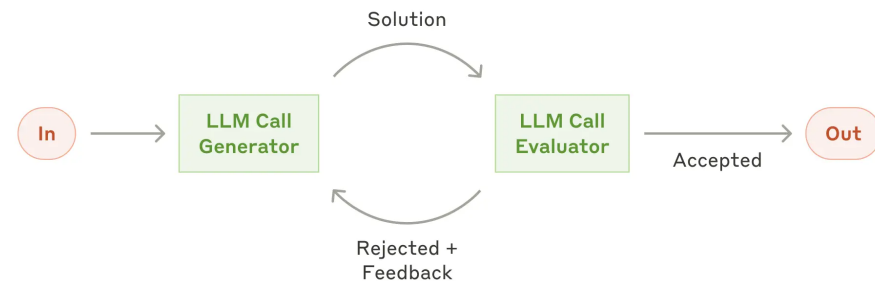
3. Parallelization



4. Orchestrator-Workers



5. Evaluator-Optimizer

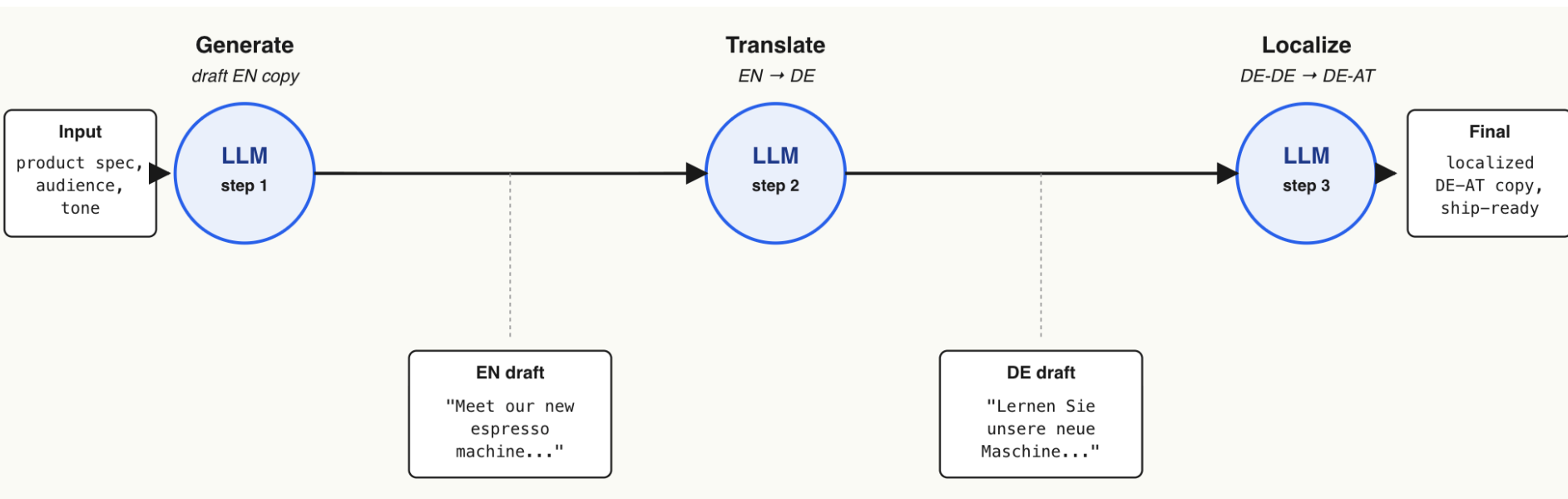


5 Major Workflow Design Patterns

Pattern	Topology	Who picks next step?	Who picks when to stop?
Chaining / Routing / Parallelization	Fixed (authored DAG)	Code	Code
Evaluator-Optimizer	Fixed loop	LLM (revise vs. accept)	LLM
Orchestrator-Workers	LLM picks subtasks; shape fixed	LLM (picks subtasks)	LLM
(Full-Blown) Agent	LLM picks freely; tools predefined	LLM (picks tools & steps)	LLM

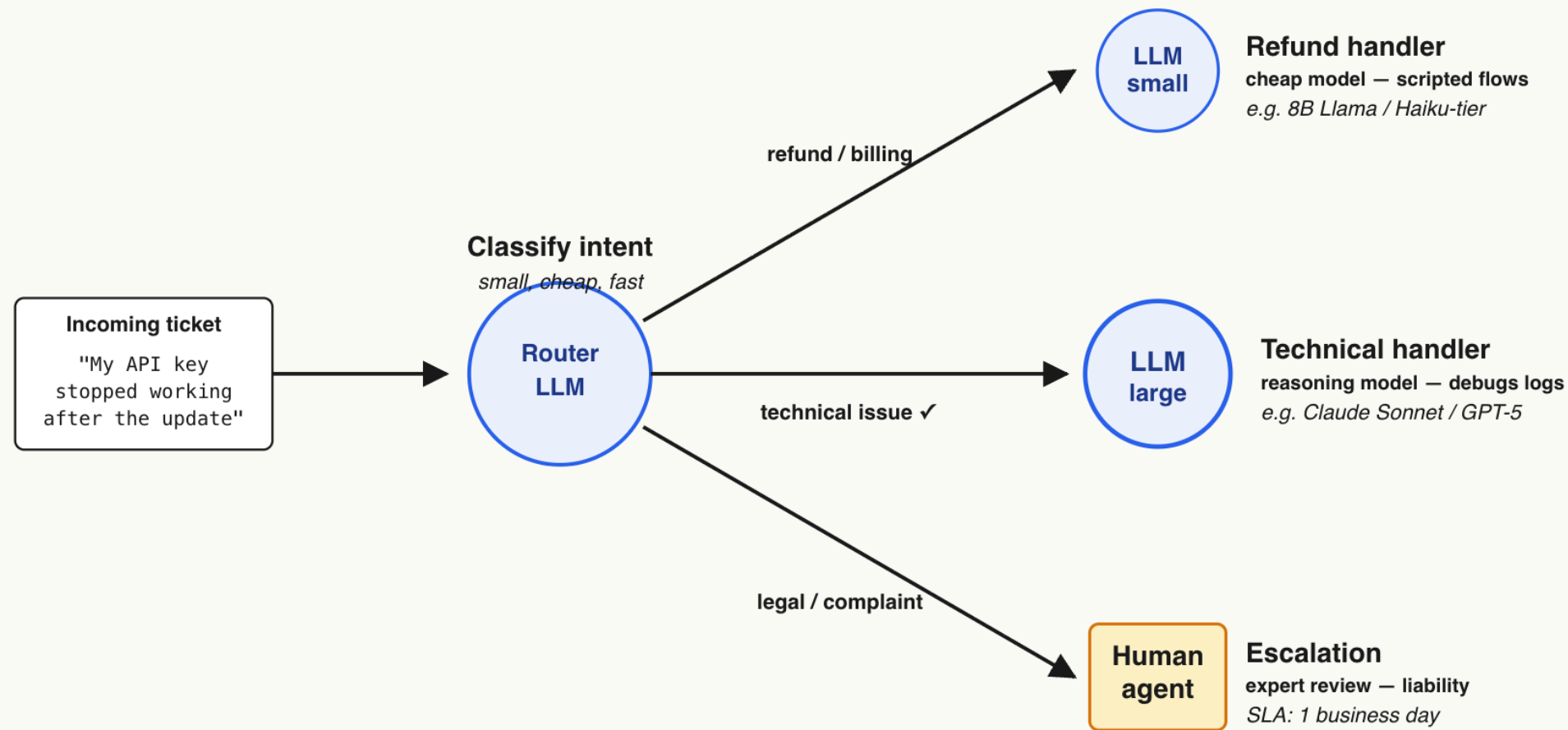
Example: Prompt Chaining

Marketing Copy Pipeline



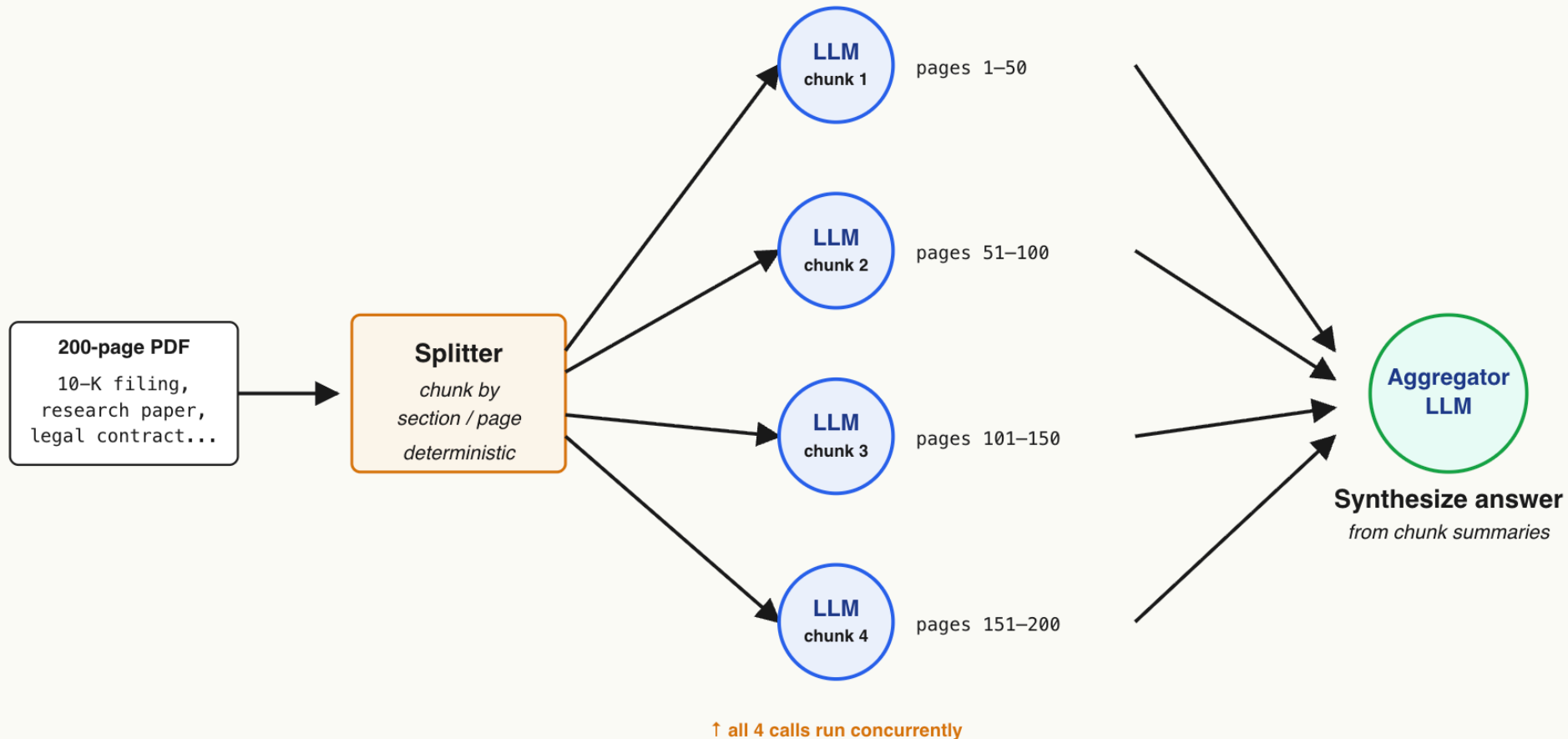
Example: Routing

Customer Support Triage



Example: Parallelization

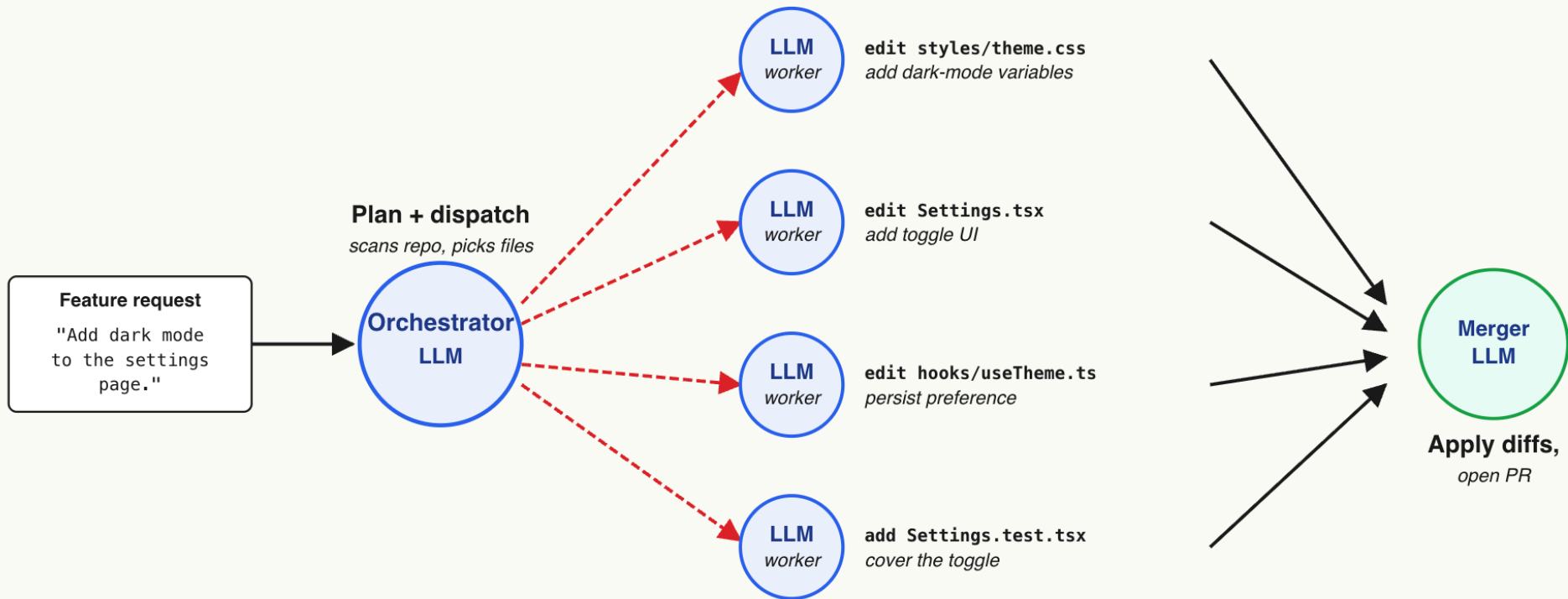
Large Document Q&A



Example: Orchestrator-Workers

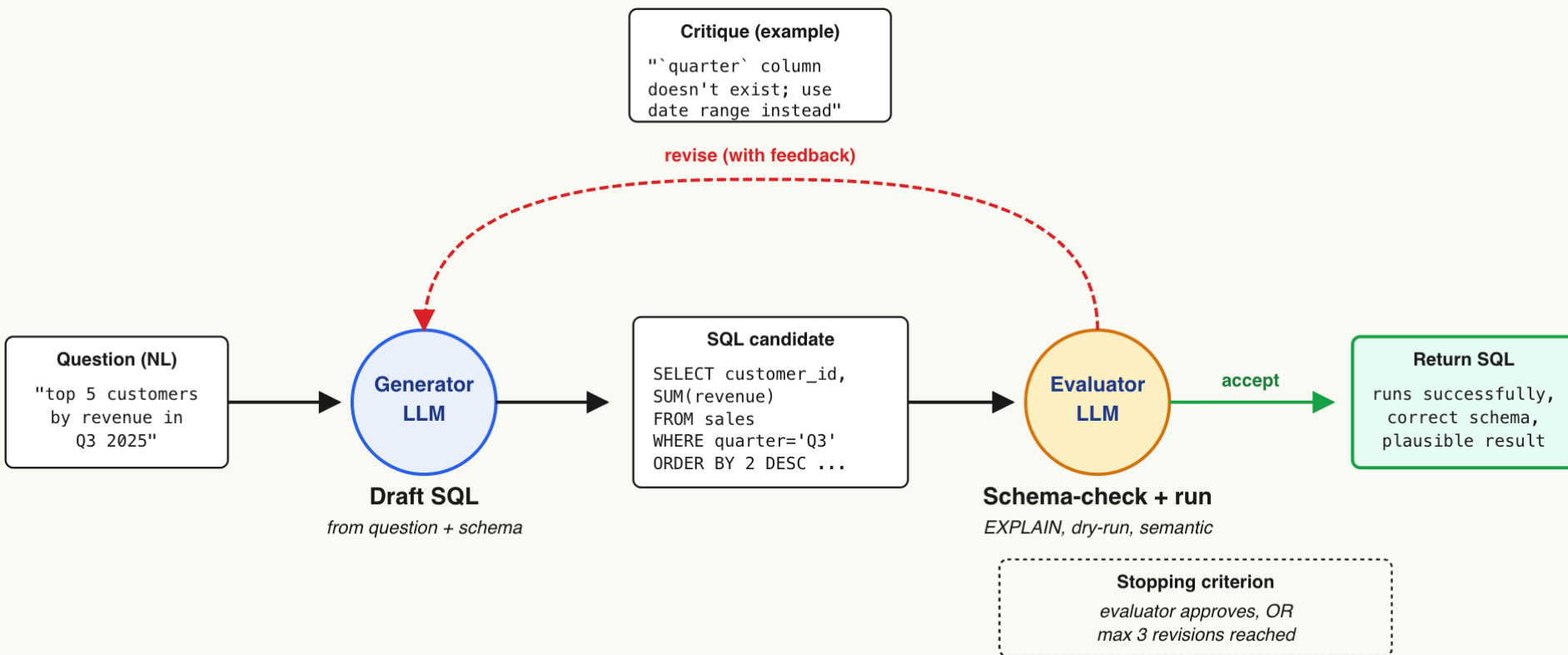
Coding Agent

workers spawn dynamically — which ones, how many, decided at runtime



Example: Evaluator-Optimizer

NL-to-SQL with Schema Checker



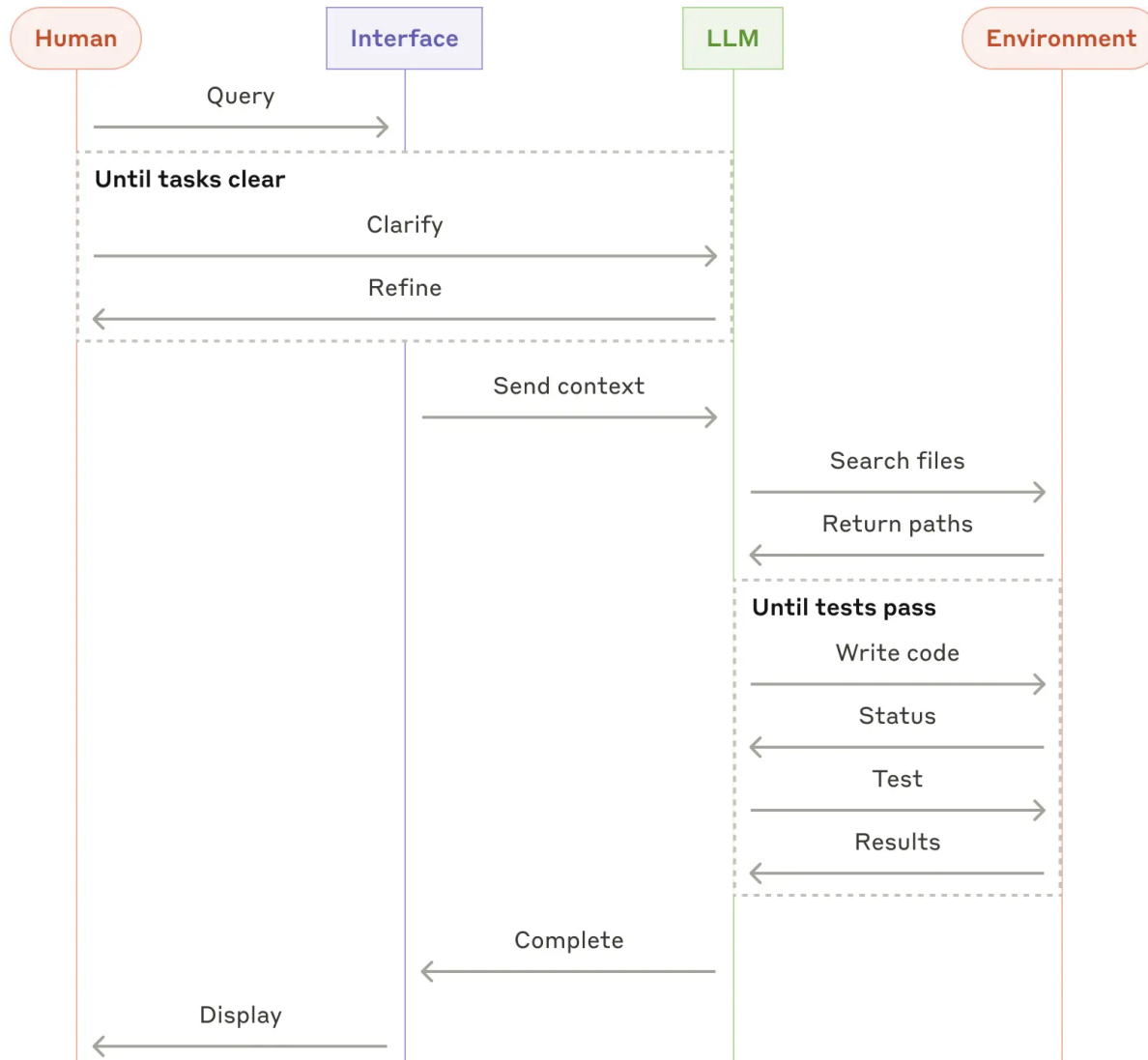
Workflow vs. Agent vs. Both

- ❖ When to reach for workflow vs. agent vs. both?

	Workflow	Agent
Structure	Can draw control flow upfront	Truly unknown until run time
Intermediates	Need control over this	Open-ended
Cost / Latency	Constraints are strict	Overshooting acceptable

- ❖ Many non-trivial AI applications often compose workflow patterns and agentic elements
 - ❖ “Agentic workflow” term encompasses full spectrum
- ❖ **Best Practice:** Do NOT build an agent if a workflow suffices! :)

Example: Coding Agent



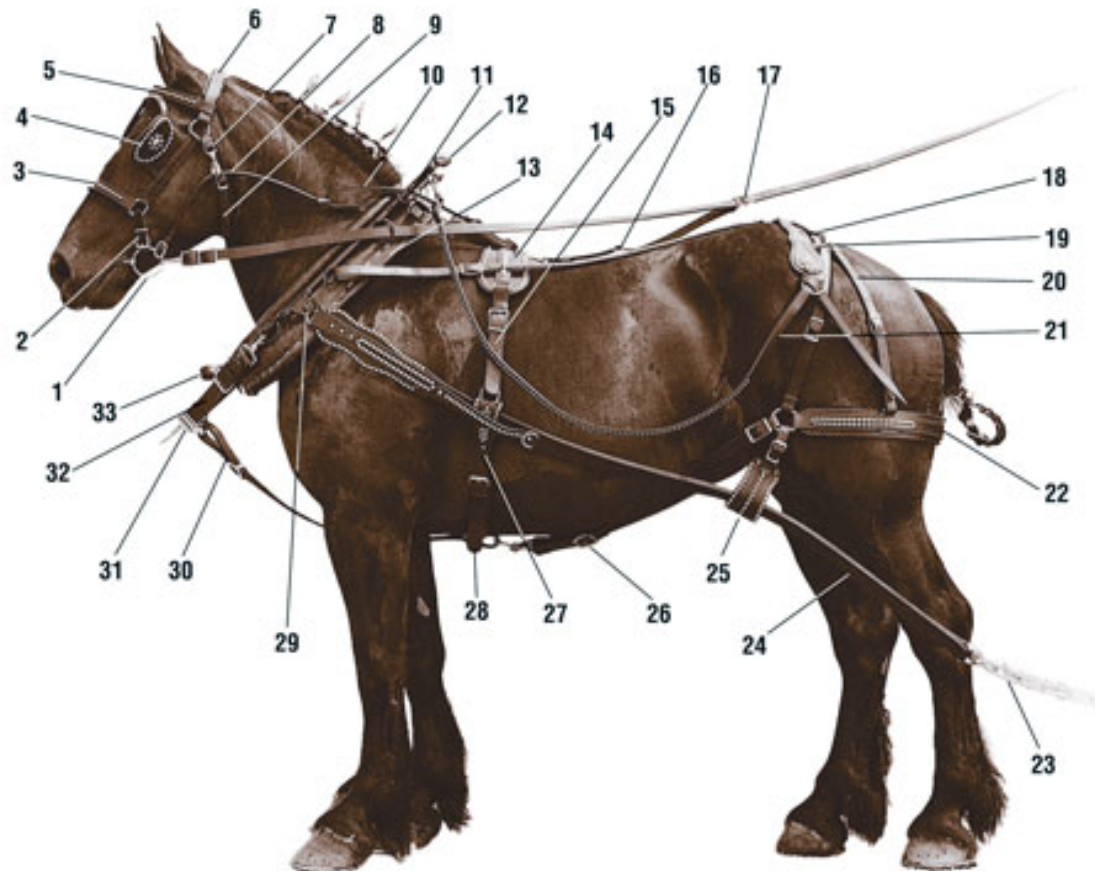
Outline

- ❖ Motivation
- ❖ Design Foundations
-  ❖ Harness Eng. and Components
- ❖ Multi-Agent Workflows
- ❖ Observability and Evals

What is a “Harness”?

1. Bit
2. Bit Strap
3. Noseband
4. Blind
5. Browband & Winker Stay
6. Head Crown
7. Gag Swivel
8. Rein Round
9. Throat Latch
10. Rein
11. Hame
12. Hame Ball
13. Collar
14. Back Band
15. Market Strap
16. Rein Up Strap
17. Check Line
18. Back & Hip Strap
19. Trace Carrier
20. Hip Strap
21. Tie Strap
22. Breeching
23. Heel Chain
24. Trace
25. Lazy Strap
26. Quarter Strap
27. Belly Band Billet
28. Belly Band
29. No. 21 Connector
30. Pole Strap
31. No. 1616 Snap
32. Breast Strap
33. Hame Strap

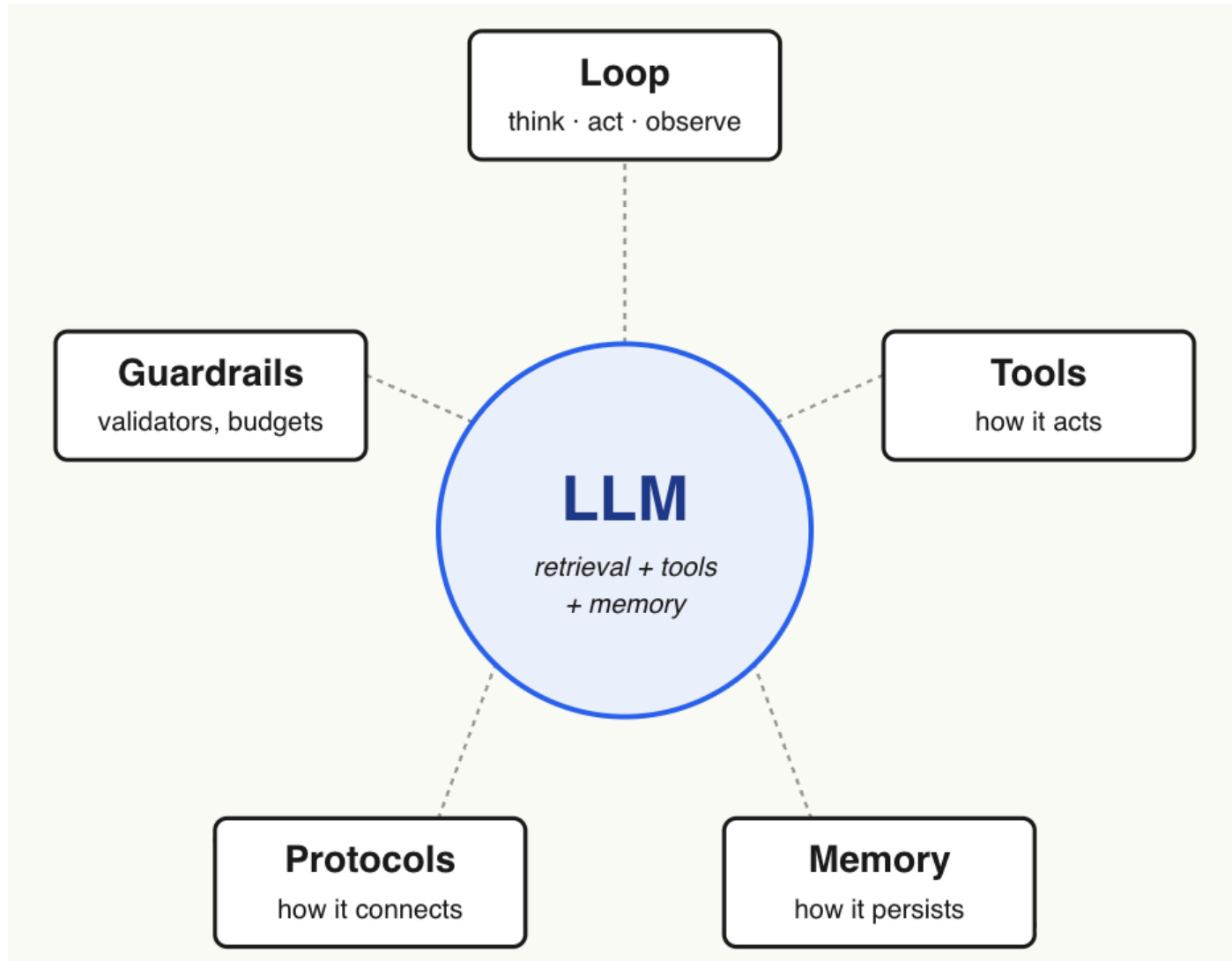
Draft Harness



Harness Engineering for LLMs

- ❖ “Harness” is a loose software eng. term for everything that surrounds the LLM(s) to boost functionality, provide reliability, and integration with application
 - ❖ Code loops, prompt/context/retrieval management, tools and dispatchers, memory management, guardrails, etc.
 - ❖ LLM provides “horsepower” of reasoning/decision-making; harness harnesses it to meet application goals
 - ❖ Not a clean partition but a useful software design concept
 - ❖ In practice: LLM API calls < 5% of effort; 95% is harness! :)

Harness Engineering for LLMs



Outline

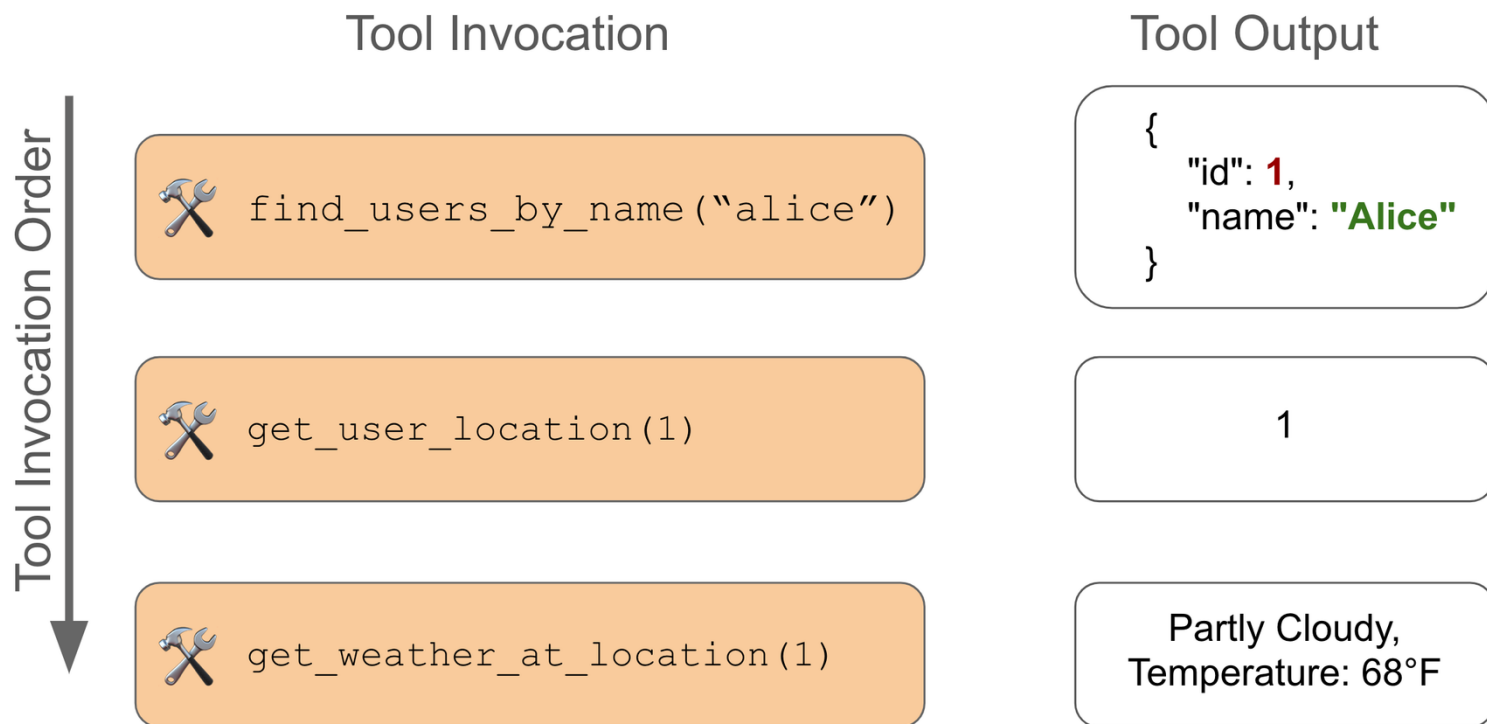
- ❖ Motivation
- ❖ Design Foundations
- ❖ Harness Eng. and Components
- ➔ ❖ Tools: For Actions
 - ❖ Memory: For Persistence
 - ❖ Protocols: For Connectivity
- ❖ Multi-Agent Workflows
- ❖ Observability and Evals

Tools: Why?

- ❖ “Package” pre-existing software to enable an LLM to use it as part of its agentic workflow:
 - ❖ Makes LLMs more useful for more types of work
 - ❖ More timely inputs from and/or actions on environment
 - ❖ More grounded reasoning/results, less hallucination
 - ❖ Validation in the loop to reduce compounding errors
- ❖ Defining a tool is more akin to data annotation than coding!

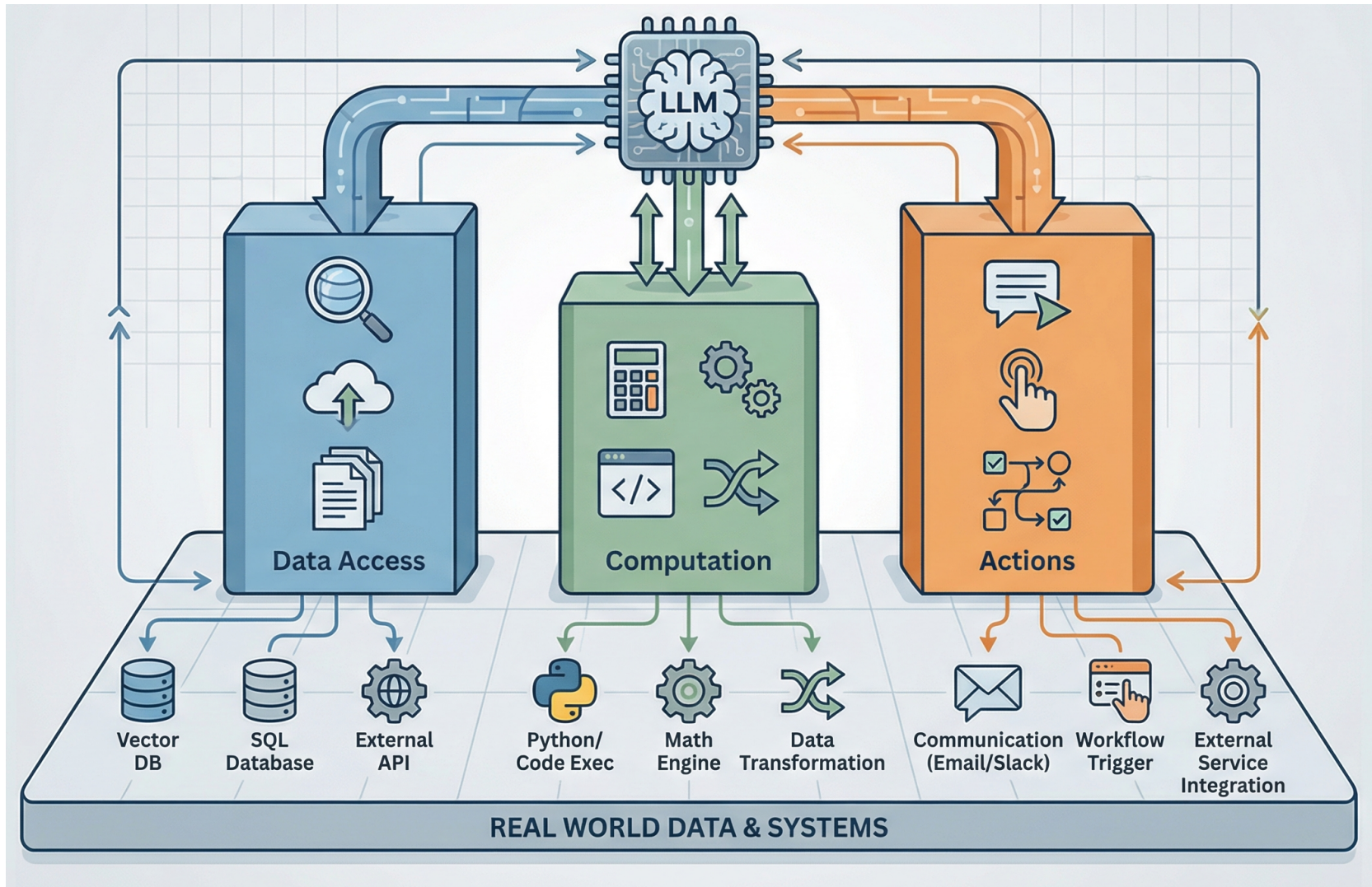
Tools: Why?

? Is it likely that Alice needs an umbrella now?



! Based on the current weather, it is partly cloudy with a temperature of 68°F at Alice's location. Therefore, it is unlikely that Alice needs an umbrella right now.

Tools: Why?



Tools: What?

- ❖ A **tool definition** is just an API with functions, names and types of inputs/outputs, descriptions for everything, and error info.

```
{
  name: "searchFlights",
  description: "Search for available flights",
  inputSchema: {
    type: "object",
    properties: {
      origin: { type: "string", description: "Departure city" },
      destination: { type: "string", description: "Arrival city" },
      date: { type: "string", format: "date", description: "Travel date" }
    },
    required: ["origin", "destination", "date"]
  }
}
```

`searchFlights (origin: "NYC", destination: "Barcelona", date: "2024-06-15")`

Tools: What?

- ❖ **Names:** Make it descriptive, e.g., `get_user_emails`, not `getue3`
- ❖ **Params:** What LLM would name them, e.g., `recipient`, not `r0`
- ❖ **Description:** Explain what it does, when to use it, AND when NOT to use it
- ❖ **Errors:** Written for the LLM to recover from, e.g., not “404”; say “`user_id 12345` doesn’t exist; try `list_users()` first”

Tools: What?

- ❖ More precision in tool definitions is better for reliability

VAGUE DEFINITION

✗ agent fails

```
name: "getUser"
params: { id: int }
description: "gets user"

# on error:
error: "404"
```

What goes wrong:

- name is opaque — model guesses what it does
- "id" — id of what? email? account? customer?
- no "when NOT to use" guidance
- "404" is unrecoverable — agent has no next move

→ TOOL MISFIRE

agent calls with wrong id type, gets 404, gives up or hallucinates

CLEAR DEFINITION

✓ agent succeeds

```
name: "get_user_by_email"
params: { email: str }
description:
    "Look up user by email address.
    Use when you have an email and need
    user_id. Do NOT use for partial
    matches — use search_users() instead."

# on error:
error: "user with email 'x@y.com'
not found; try list_users() first."
```

What this fixes:

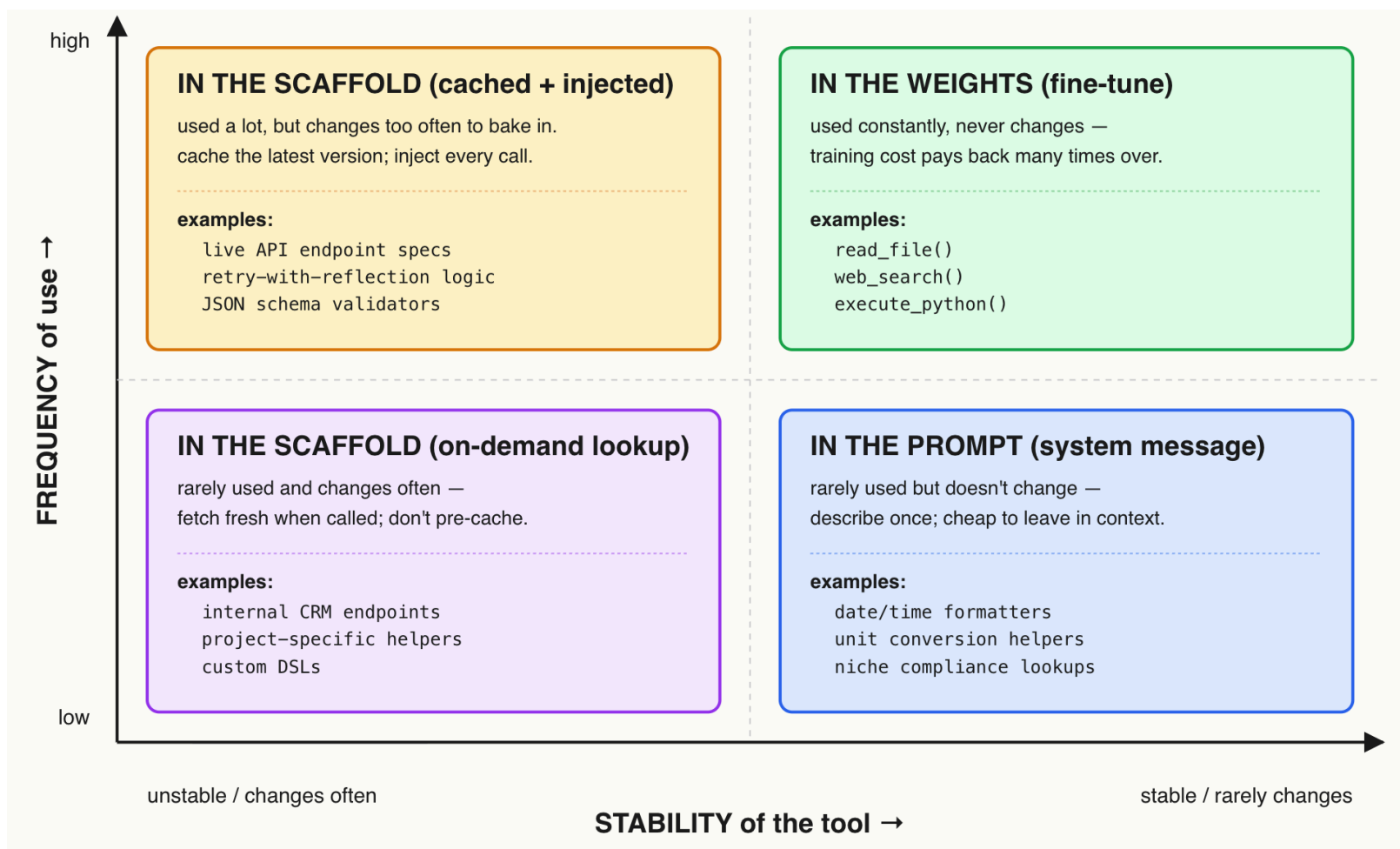
- descriptive name — no guessing
- named param "email" — model knows the type
- "when NOT to use" — the line everyone skips
- error tells the model its next move

→ RECOVERS GRACEFULLY

on miss, agent calls list_users(), retries — completes the task

Tools: Where?

- ❖ Tool use can be “taught” in multiple places based on stability of tool and frequency of use by application

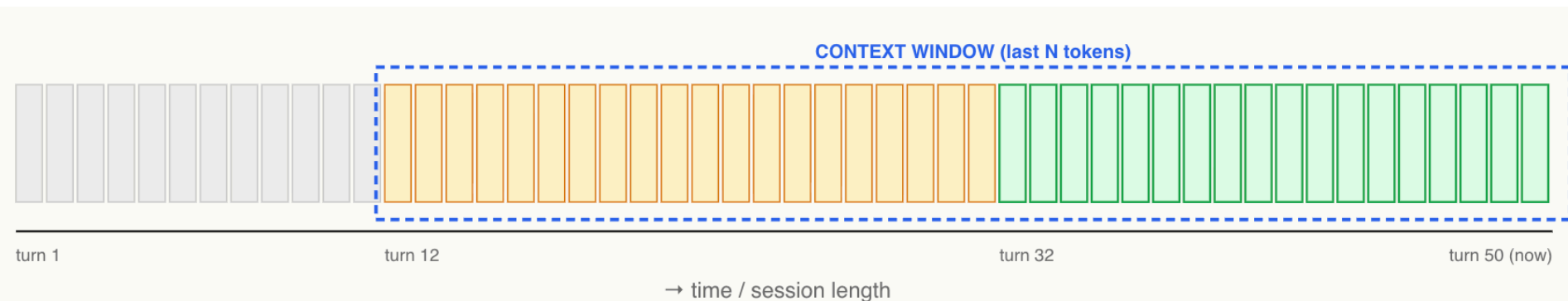


Outline

- ❖ Motivation
- ❖ Design Foundations
- ❖ Harness Eng. and Components
 - ❖ Tools: For Actions
 - ➔ ❖ Memory: For Persistence
 - ❖ Protocols: For Connectivity
- ❖ Multi-Agent Workflows
- ❖ Observability and Evals

Memory: Why?

- ❖ Balances a fundamental tradeoff on managing intermediates:
 - ❖ Get rid of them -> Likely useful info. lost
 - ❖ All inside context window -> Risk of context rot



SCROLLED OUT

Beyond the window.
Gone unless persisted.

- session setup is forgotten
- user has to re-explain context

LOST IN THE MIDDLE

In context but attention
degrades (Topic 2: rot).

- model "saw it" but won't use it
- silent failure mode

RECENT / CRISP

High attention. Reliably
used in next response.

- the only "safe" zone
- but this zone is small

Memory: Three Tiers

- ❖ Now-canonical notion of 3 tiers of “memory” for AI applications:
 - ❖ **Observation Stream:** Every event, timestamped, append-only log
 - ❖ **Reflection:** LLM periodically synthesizes higher-level beliefs, e.g., “this repo uses pytest”
 - ❖ **Plan:** Forward-looking, regenerated as conditions change
- ❖ Retrieval must take into account recency, relevance, and LLM-rated importance; the agent runs LLM-as-judge on its own past!

Running Example

PLAN

forward-looking, regenerated as conditions change

Goal: diagnose loss explosion at step 5K.

Next: verify fix with smaller lr; if stable, write up findings.

REFLECTION

periodic synthesis

"this repo uses pytest"

from observations #14, #27, #31

importance: 0.7

"lr=3e-3 is anomalous"

from observation #45

importance: 0.9

"sampler is correct"

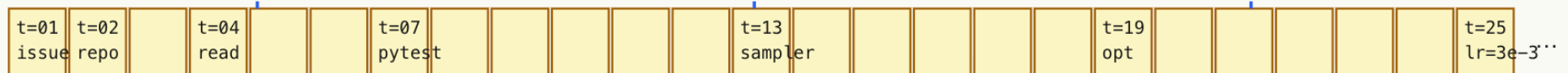
from observations #41, #42

importance: 0.4

inform plan

OBSERVATION STREAM

every event, timestamped, append-only



*synthesize
(periodic)*

Memory Management Principles

- ❖ “Memory” for LLMs/agent is NOT like system DRAM!
 - ❖ More is NOT always better (context rot, token spend, etc.)
 - ❖ Inherently lossy notions of “summarization” needed
- ❖ Production memory management must aggressively handle this tradeoff of what to “forget” and how much:
 - ❖ Do you really need full trace of older turn? Do you really need tool call specifics after context enriched?
 - ❖ Use LLM itself to summarize at different stages / levels? KG/ontology to guide structured summaries?

Scope of Agent Memory

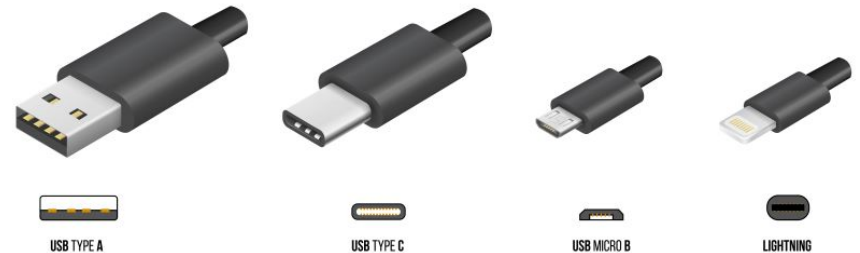
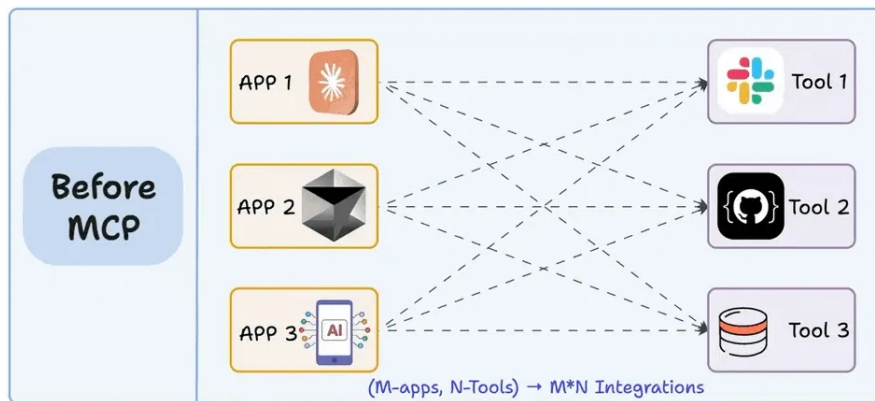
- ❖ **4 main levels:**
 - ❖ Per-session vs. per-user vs. per-team vs. per-org.
- ❖ Considerations and tradeoffs:
 - ❖ Different stores with cross-references?
 - ❖ What system substrate?
 - ❖ Levels of summarization?
 - ❖ Access control, privacy, security, governance
- ❖ Still an active area of research!

Outline

- ❖ Motivation
- ❖ Design Foundations
- ❖ Harness Eng. and Components
 - ❖ Tools: For Actions
 - ❖ Memory: For Persistence
 - ➔ ❖ Protocols: For Connectivity
- ❖ Multi-Agent Workflows
- ❖ Observability and Evals

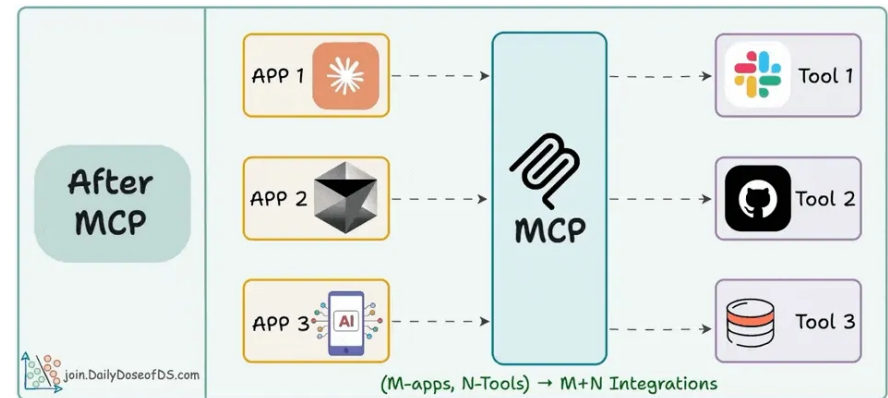
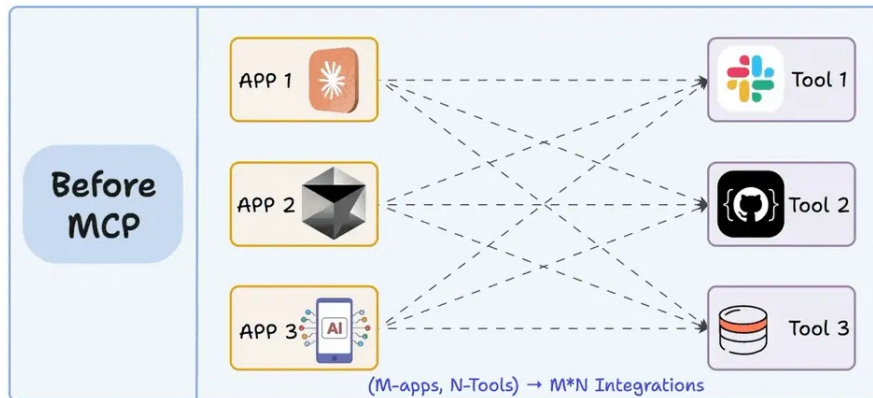
Protocols: Why?

- ❖ **Problem:** ~2 years ago, every agent eng. framework needed to implement its own interface to tools
 - ❖ Integrations were bespoke, non-portable, error-prone
 - ❖ “M x N complexity” of integrations
- ❖ Pressing need for a *standard* for agent-tool communication



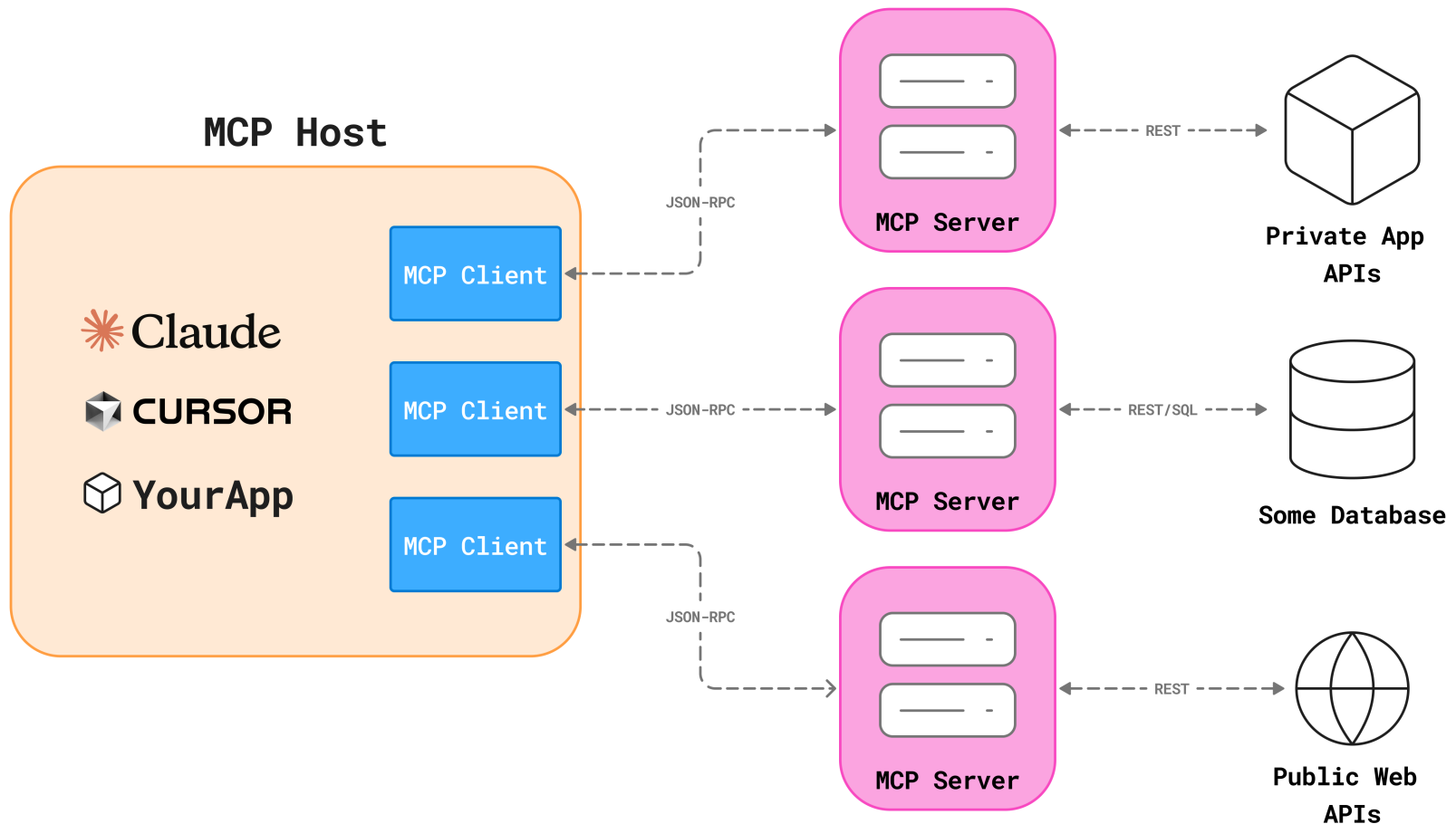
Model Context Protocol (MCP)

- ❖ Open protocol introduced by Anthropic in Nov 2024
- ❖ Unified how any LLM had to “understand” how to use any tool
- ❖ Has become industry standard AI infra. used by all vendors
 - ❖ Under Linux Foundation
 - ❖ ~100mil monthly SDK downloads!



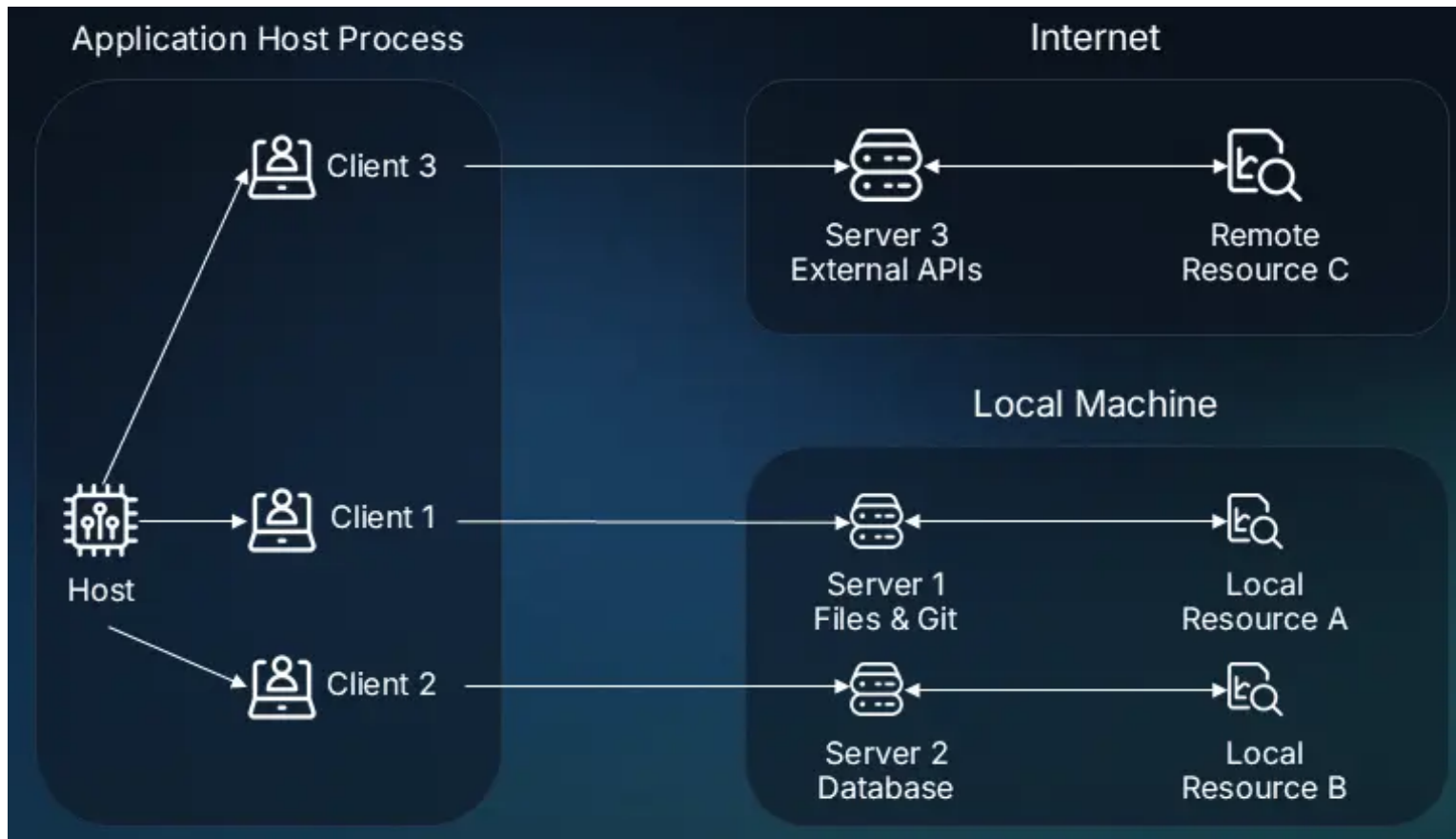
Model Context Protocol (MCP)

❖ **Host** (your appl.) <-> **Client** (runtime) <-> **Server** (the tool)



Model Context Protocol (MCP)

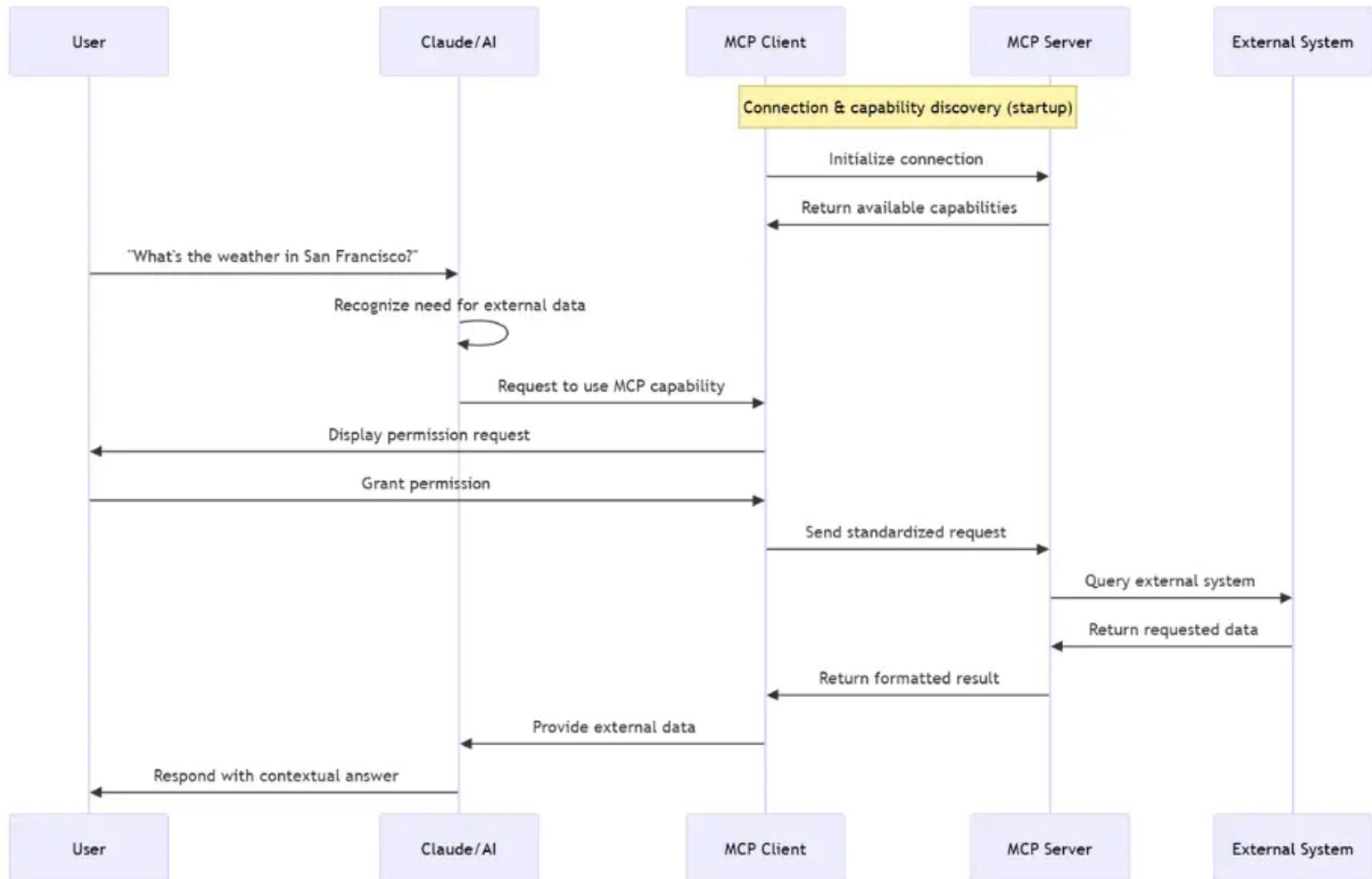
- ❖ MCP Servers can be on local machine too, not just remote



Model Context Protocol (MCP)

- ❖ An MCP Server has 3 primitives: Tools, Resources, Prompts
- ❖ **Tools** (Model-Controlled):
 - ❖ Executable functions that an AI appl. can invoke to perform actions, such as running a DB query, sending an email, calling a weather API
- ❖ **Resources** (Application-Controlled):
 - ❖ Read-only data sources that provide context info. to an AI appl., such as file contents, logs, DB records, API responses
- ❖ **Prompts** (User-Controlled):
 - ❖ Pre-defined instruction templates that help an LLM use tools or resources well, e.g., a “Code Review” prompt, few-shot examples

Model Context Protocol (MCP)



Model Context Protocol (MCP)

❖ Data Layer:

- ❖ JSON-RPC 2.0 messaging structure and semantics

❖ Transport Layer:

- ❖ STDIO for local communication
- ❖ HTTP Post for inter-machine communication

❖ Request Handlers:

- ❖ Internal logic that listens for incoming calls, e.g., `list_tools`, `call_tool`, and executes the corresponding backend action

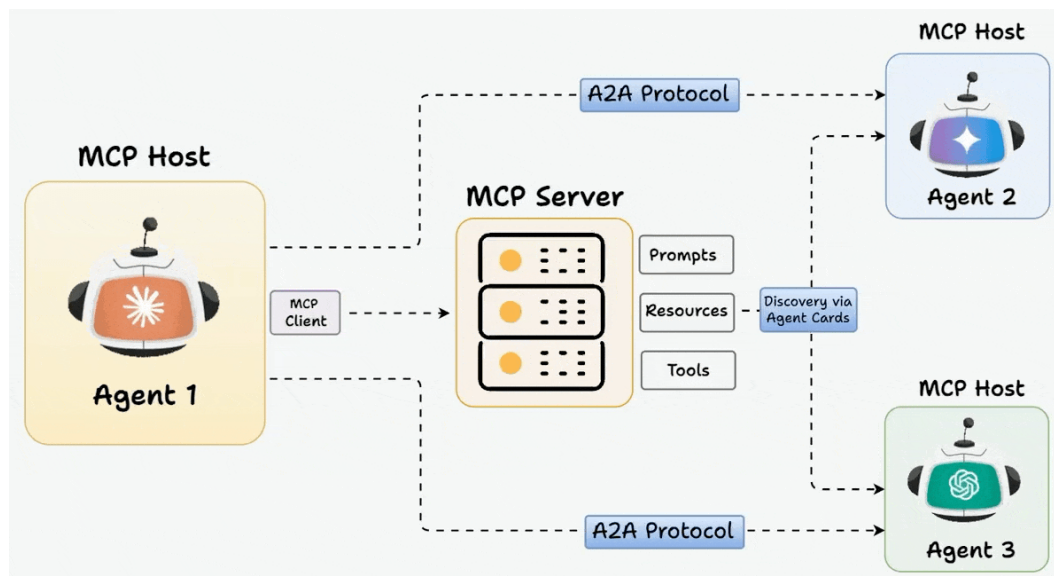
❖ State & Session Management:

- ❖ Manages stateful connections and carries context from one request to the next within a session

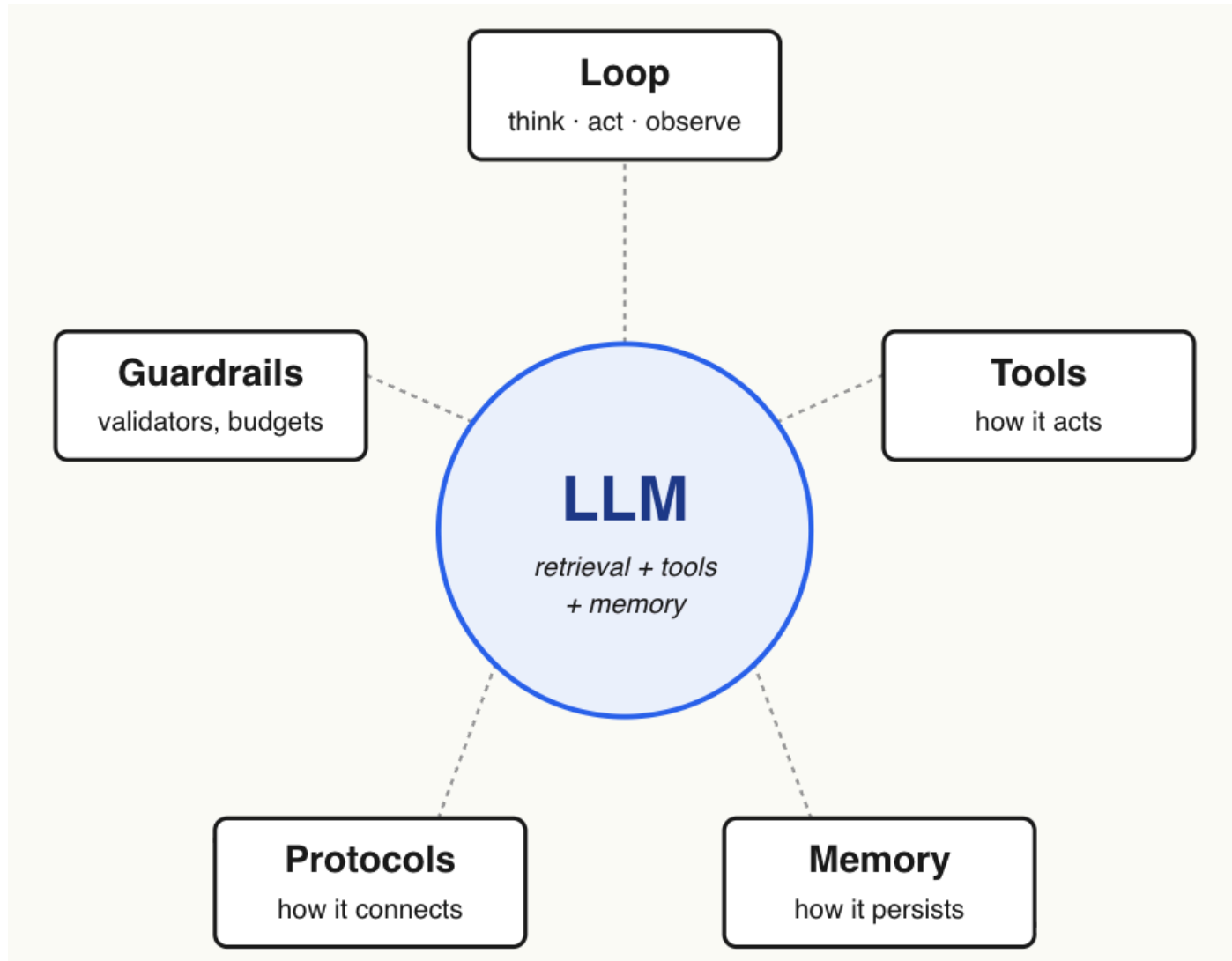
❖ Examples: <https://modelcontextprotocol.io/examples>

Agent 2 Agent Protocol (A2A)

- ❖ **Problem:** Agents now talk with other agents too, not just tools
- ❖ A2A introduced by Google in 2025
 - ❖ Think of A2A as “horizontal” and MCP as “vertical”
 - ❖ Also industry standard now; also under Linux Foundation
 - ❖ Notions agent “cards”, “tasks”, “lifecycle”, etc. for discovery of agents, inter-agent communication, task delegation, etc.



Recap: Harness Engineering



Outline

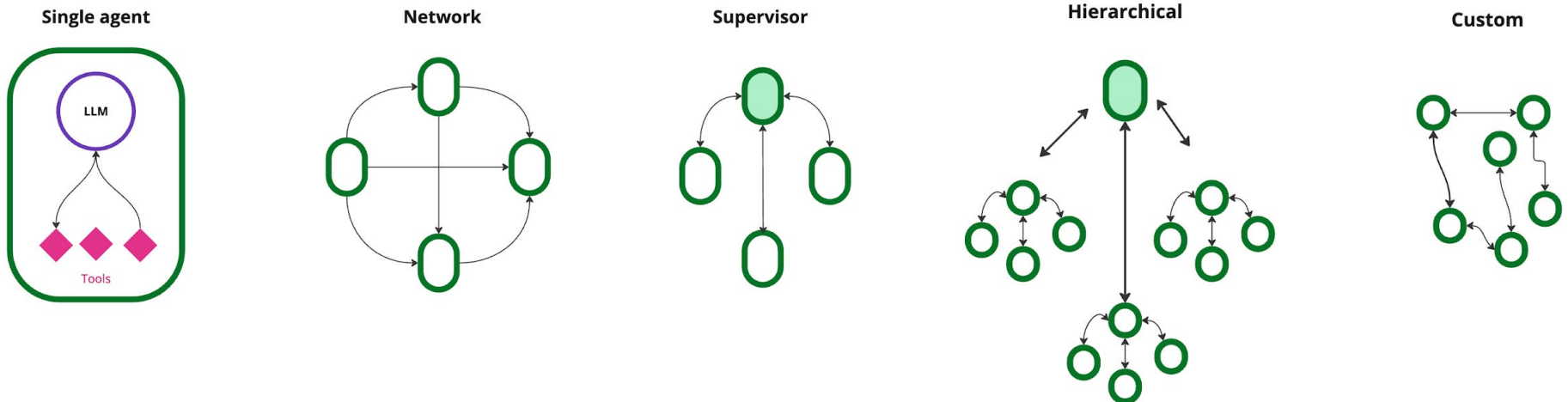
- ❖ Motivation
- ❖ Design Foundations
- ❖ Harness Eng. and Components
-  ❖ Multi-Agent Workflows
- ❖ Observability and Evals

Why Multiple Agents?

- ❖ Recall issues of context rot, context window limit, token limits
- ❖ So, in a complex AI application, it is sometimes worthwhile to *decompose* work to alleviate 3 key pressure points:
 - ❖ **Context:** One agent's window fills with all info. vs. Separate (sub-)agents with their own windows
 - ❖ **Objectives:** Separation of concerns, e.g., “research AND critique AND format” could cause “conflict of interest”
 - ❖ **Permissions:** Contain security blast radius with designated (sub-)agent roles and access control/permissions
- ❖ But it introduces a new complexity: *How to coordinate?*

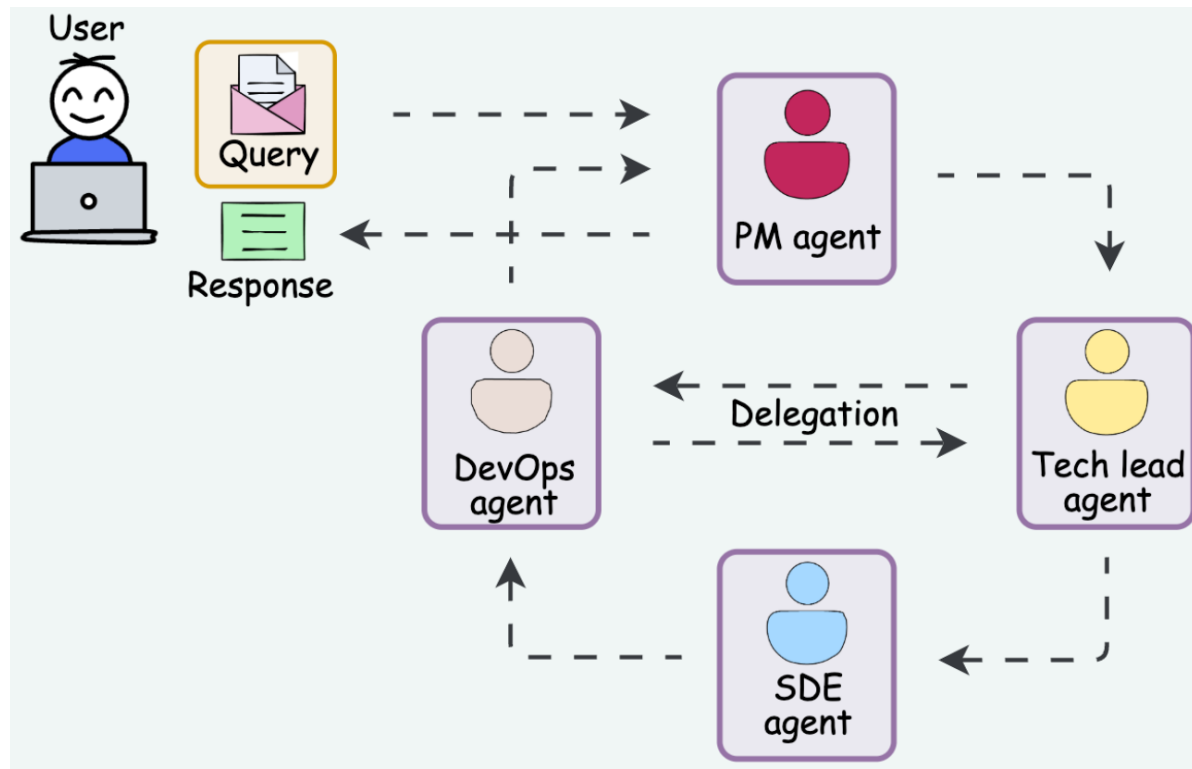
Multi-Agent Topologies

- ❖ A few major topologies of multi-agent workflows:
 - ❖ **Peer-to-Peer/Network:** Debate / vote toward consensus
 - ❖ **Supervisor:** Decomposes and delegates to subordinates
 - ❖ **Hierarchical:** Recursive supervision
 - ❖ **Pipeline:** Specialists hand off in sequence



Multi-Agent Topologies

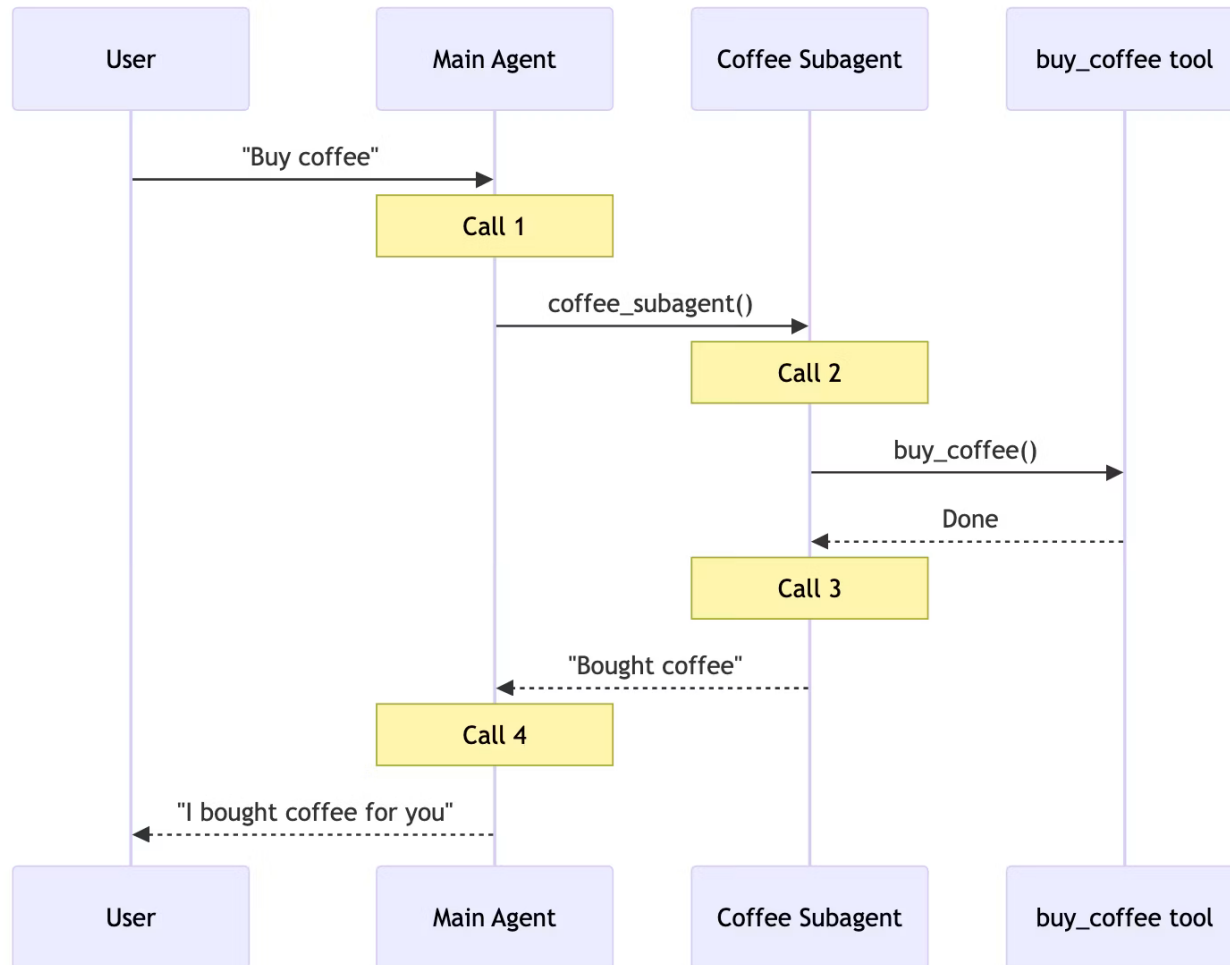
- ❖ Agent roles are reusable: planner, researcher, writer, critic, etc.
- ❖ How they coordinate affects eval metrics, latency, costs, etc.
- ❖ Often anthropomorphized based on human team structures!



Tradeoffs of Multi-Agent Workflows

- ❖ Multi-agent workflow structure is NOT always necessary nor always useful
 - ❖ Mo' agents, mo' problems! :)
 - ❖ More evals, traces to debug, contradictions, costs, etc.
- ❖ Only a few successful multi-agent workflows in production so far
- ❖ Different agent eng. frameworks have their own terminologies but are essentially interchangeable
 - ❖ LangGraph, CrewAI, OpenAI Agents SDK, Google ADK
 - ❖ All rely on MCP servers as their tool layer

Example: Multi-Agent Workflow



Outline

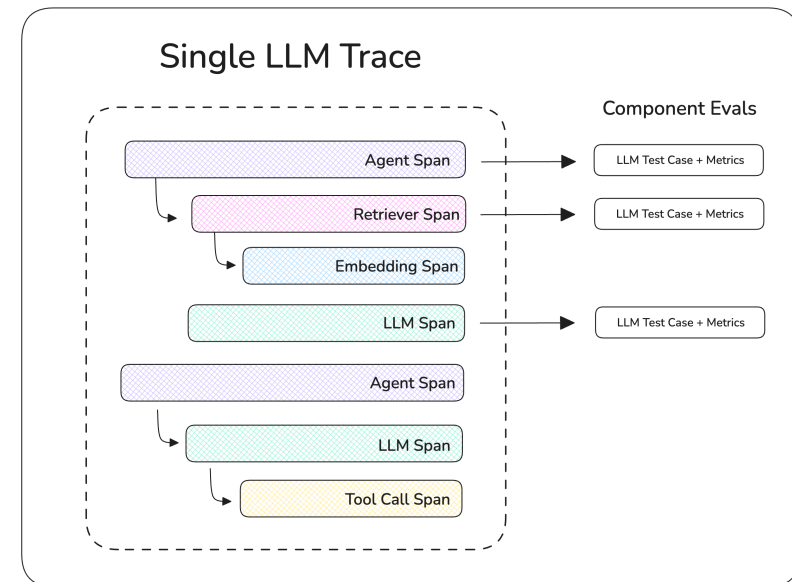
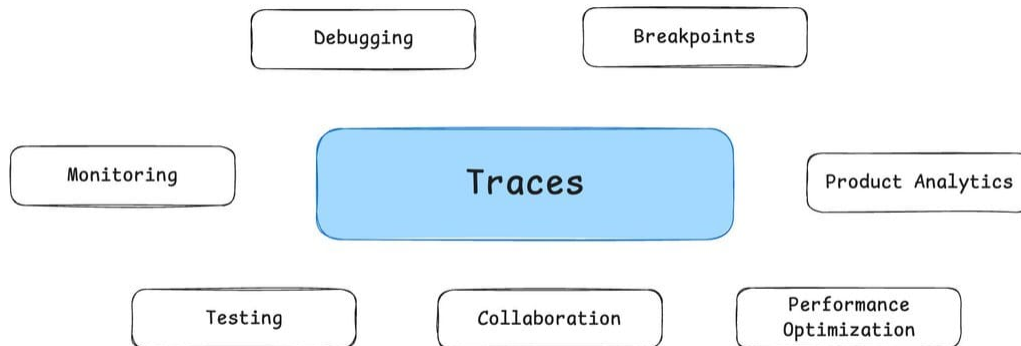
- ❖ Motivation
- ❖ Design Foundations
- ❖ Harness Eng. and Components
- ❖ Multi-Agent Workflows
-  ❖ Observability and Evals

Why Observability?

- ❖ As agentic apps become more complex and multi-step, need to keep a tab on what they do at every step
- ❖ **Trace:** A log of every “thought”, action, or observation by an AI application at production/inference time
 - ❖ Input, output, timestamp, other metadata
 - ❖ Full prompt each turn (system + history + tools)
 - ❖ Tool calls WITH arguments AND results
 - ❖ Per-step latency and tokens
 - ❖ Loop termination cause
- ❖ **Observability:** Managing all trace data for all AI applications

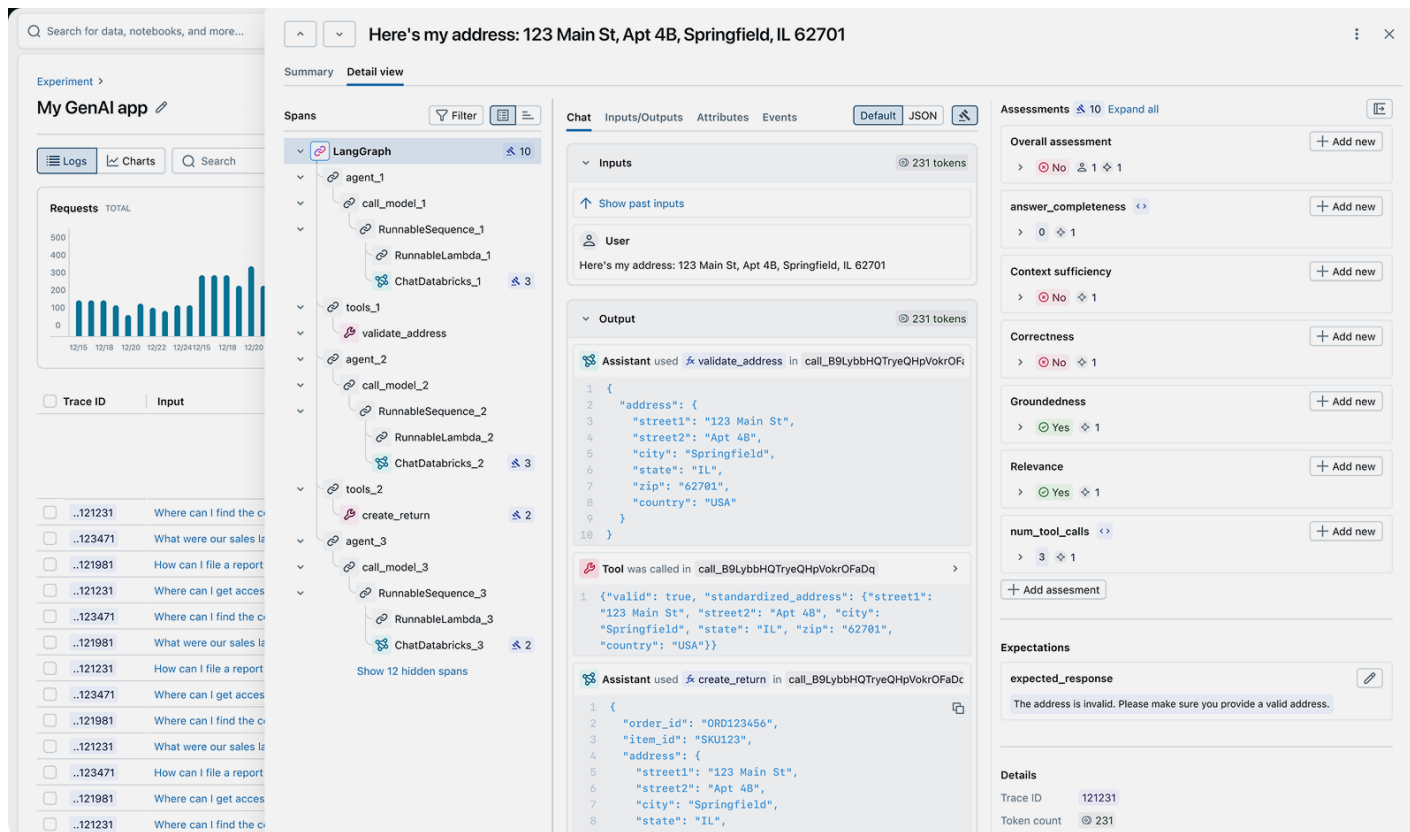
Traces for Agentic Workflows

- ❖ Traces are critical for various purposes: debugging, auditing, monitoring, etc.



Traces for Agentic Workflows

- ❖ Various tools exist to auto-track them: MLflow, W&B, LangSmith, Arize AI, etc.



Open Question: Evals for Agents

- ❖ *How does one evaluate an agentic workflow?*
- ❖ No single best approach or consensus yet
- ❖ Best practice is to track at least 4 groups of items:
 - ❖ **Outcomes:** Did it complete the task? (Accuracy)
 - ❖ **Tool Use:** Did it use the right tools, right args? (Accuracy)
 - ❖ **Process:** Was the path reasonable? (Efficiency, Cost)
 - ❖ **Failure Modes:** Where did it break? (Reliability)

Outline

- ❖ Motivation
- ❖ Design Foundations
- ❖ Harness Eng. and Components
- ❖ Multi-Agent Workflows
- ❖ Observability and Evals

Review Questions

1. How does an agentic workflow differ fundamentally from context/retrieval engineering?
2. What is the ReAct loop, and why combine reasoning with acting rather than using either alone?
3. List 3 failure modes of the core agent loop and one defense for each.
4. What single most important property distinguishes a workflow from an agent, and what are the practical consequences for cost, structure, and evals for the two?
5. Name 3 major agentic workflow design patterns, and for each say who picks the next step and who picks when to stop.
6. What is a “harness” in LLM/agent engineering?
7. What problem motivated MCP? How does it solve it?
8. Your AI colleague proposes a single agent workflow to handle IT support tickets that fall into 3 disparate buckets (refunds, technical, legal). Do you agree? Justify why or why not?