Context pressure with MCP

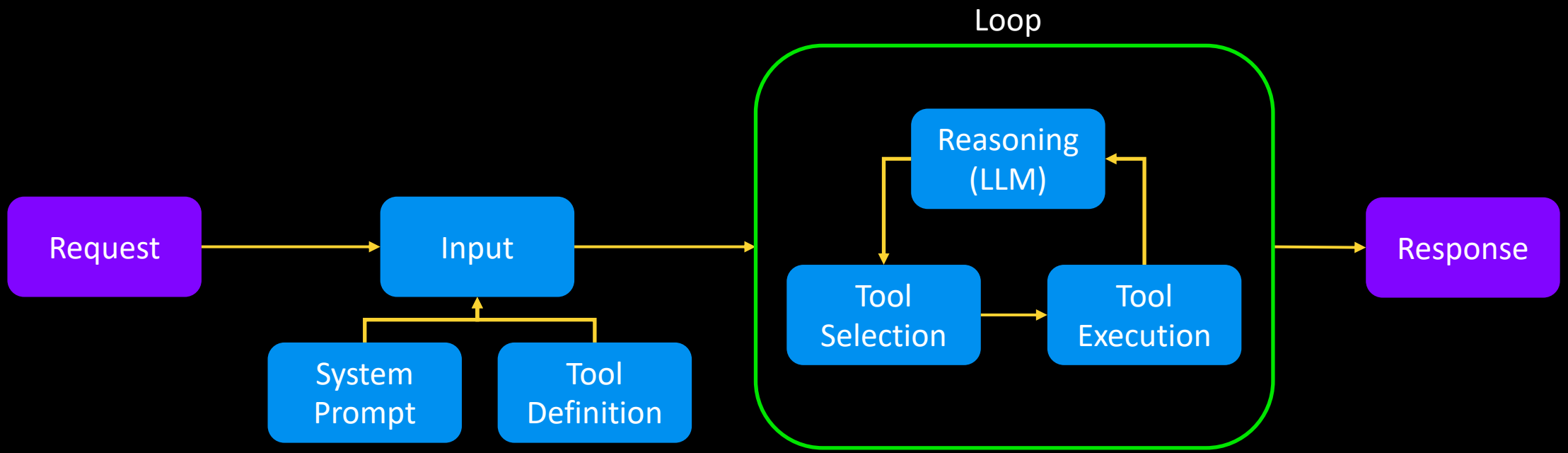
Patterns and trade-offs

Danilo Poccia (he/him)

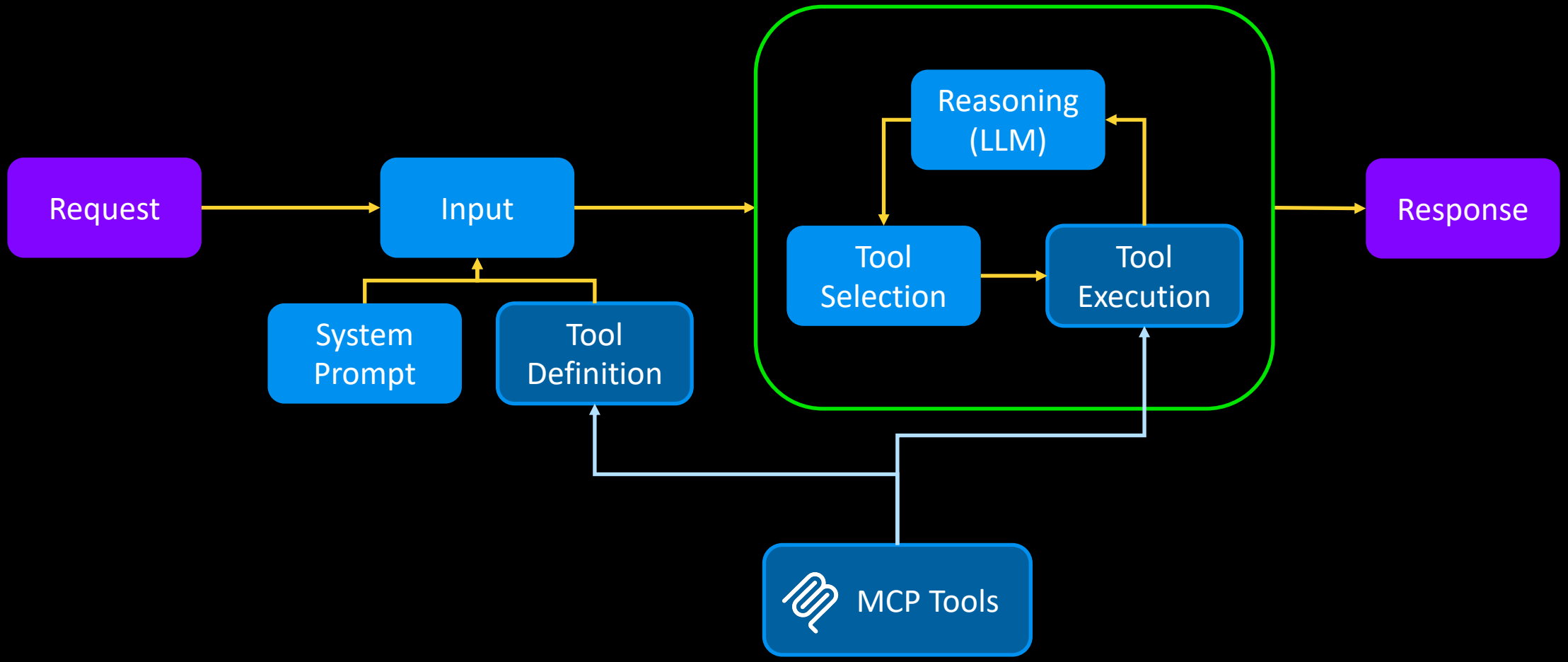
Chef Evangelist (EMEA)

What is the problem?

Agent Loop

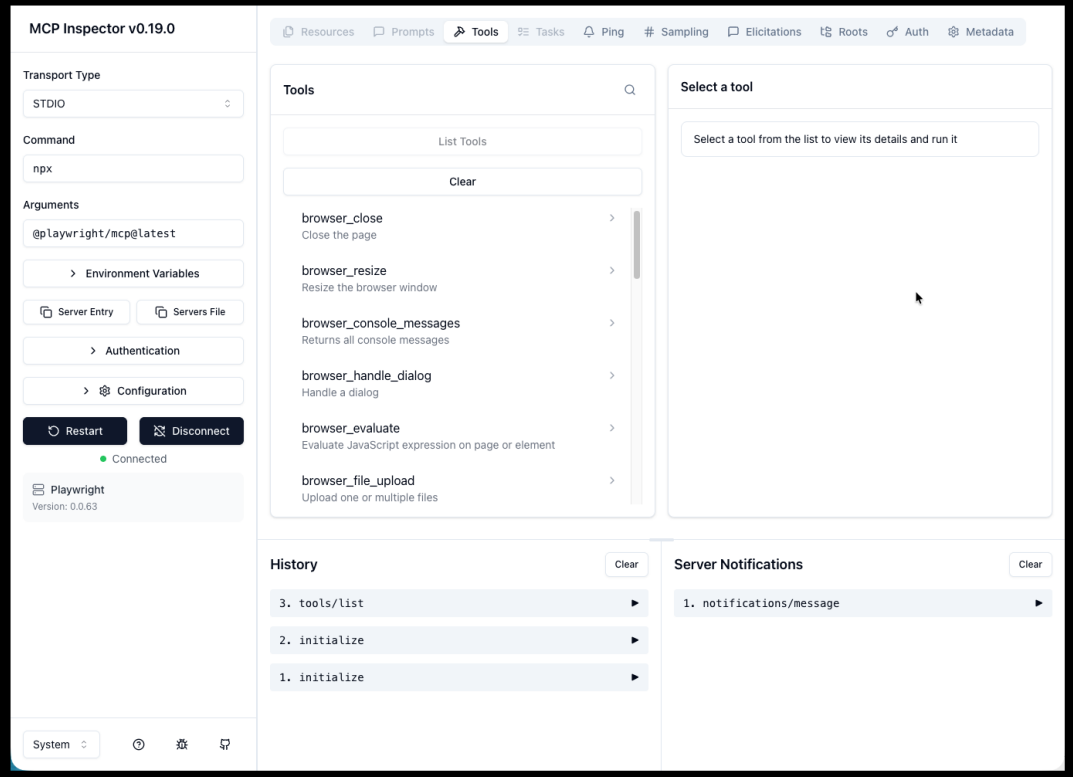


Model Context Protocol (MCP) Tools



MCP solves a problem but introduces a new one

- A **single** MCP server can expose **many** tools
  - Each tools add its own syntax and description (how to use it) to the system prompt
- For example, just Playwright MCP adds
  - 22 core + 12 optional = **34 tools**
  - Tool description > **9K tokens**
- Just 10 MCP server can **fill up** the context window

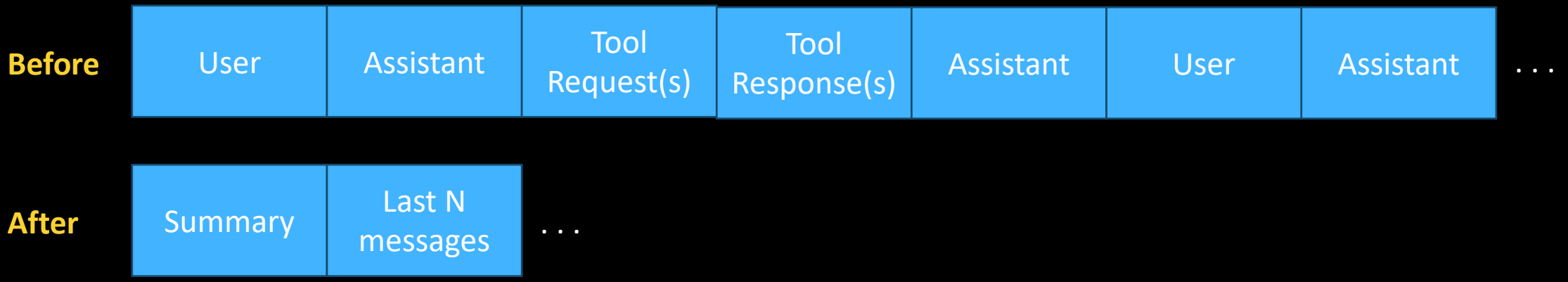


Agent Context

- **Context** is everything the model sees when making a decision: system prompts, tool definitions, conversation history, retrieved data, and tool results. It's the model's **working memory**.
  - **Context engineering** is the discipline of deciding what belongs in that working memory at any given moment, and what doesn't
  - **Context pressure** is what happens when the things we ***want*** the model to see compete for space with things we ***must*** include, including MCP tool definitions
- Context is **scarce** – Your computer has **GBs** of RAM, but a model's working memory is roughly equivalent to a long novel

What happens when the context fills up?

- Usual approach is conversation **compaction** via summarization



- Context looses details each time the conversation is summarized
  - Some tool show the summary, good for understanding and debugging

What about the solutions?

Tool Optimization

Description must be clear

- Use unambiguous parameter names
  - **user_id** not user
- Explicit context about when to use or avoid tools
- Include specialized query formats and niche terminology definitions
  - Domain knowledge
- Add concrete examples for complex parameters
  - If they can’t be simplified
  - Don’t add tools to do so

Token-efficient results

- Return the minimum amount of information
- Return semantically meaningful fields rather than technical details
  - Names, descriptions, relevant IDs instead of low-level IDs or UUIDs
- Avoid overwhelming context with irrelevant data
  - Implement pagination, range selection, and filtering and reranking in tool responses

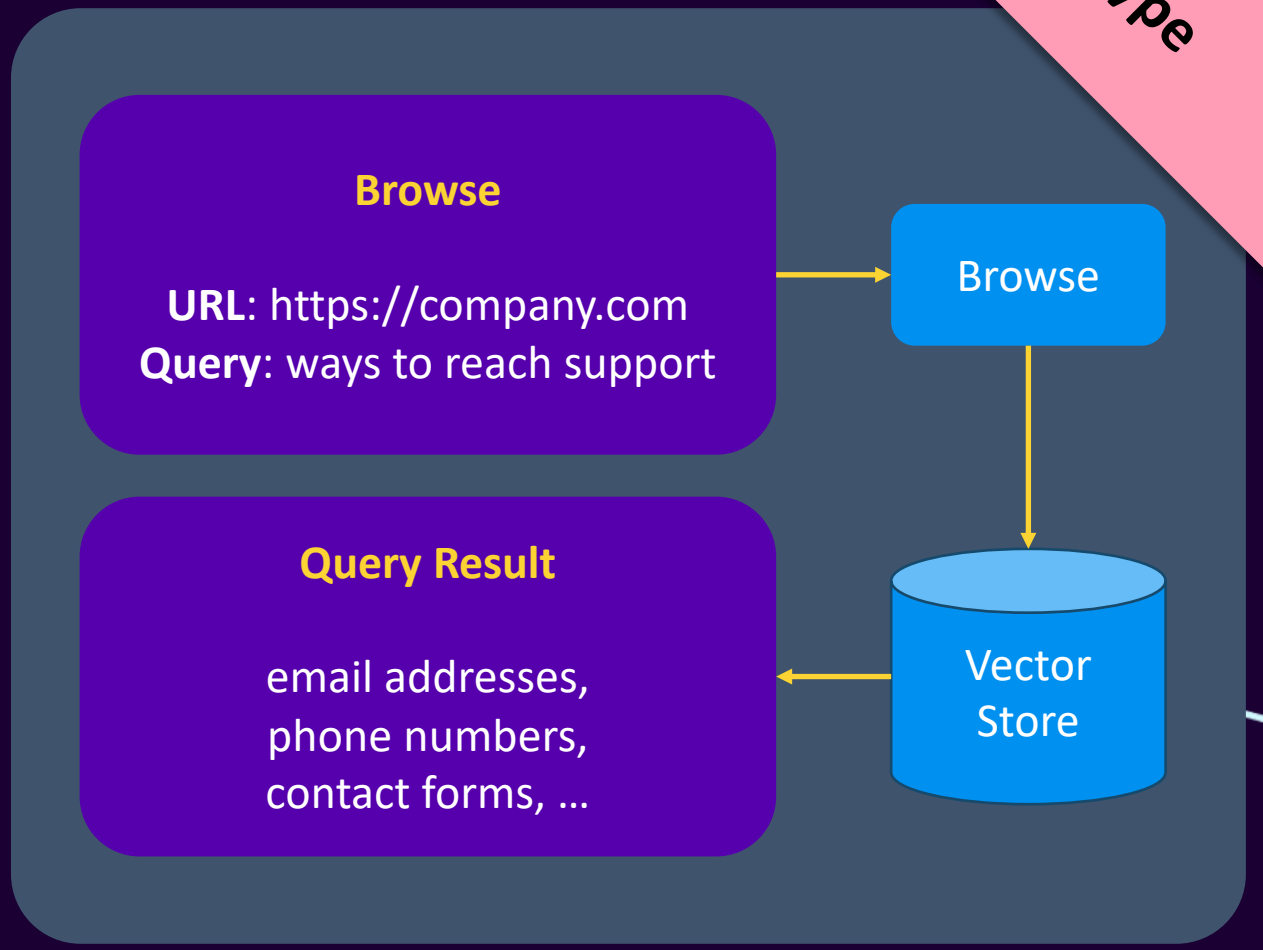
Similar to how a human would interact with a large amount of information

“Light Browser”

A lightweight web browser designed for humans (CLI/TUI) and AI agents (MCP)

Prioritizes content extraction over visual fidelity, making web content accessible in bandwidth-constrained environments

https://github.com/danilop/light-browser



Multi-agent solutions and subagents

- Use **more** than one agent
  - Each agent has their **own** context and access to a **subset** of tools
  - The amount of **information** passed between agents should be less than the overall context of each agent

- Works well when agents are focused on **separate** tasks
  - Research subagents
    - Ge tin input a query and retrieve a lot of content and extract what you need
  - Agents taking a decision, computing a score, or applying classification

Multi-agent architectures

- **Agents-as-tools**
  - Hierarchical systems where specialists serve as intelligent tools
  - Each agent has access only to a subset of the tools and MCP servers
  - Pattern: MCP + A2A using the Agent Card for Tool Definition
- **Graphs**
  - Structured workflows with deterministic execution paths
- **Swarms**
  - Autonomous collaboration with self-organizing teams of agents
- **Meta agents**
  - Dynamic agents that can modify their own orchestration behavior

Strands Agents using dynamic orchestration with meta agents

Open Source
Model Agnostic

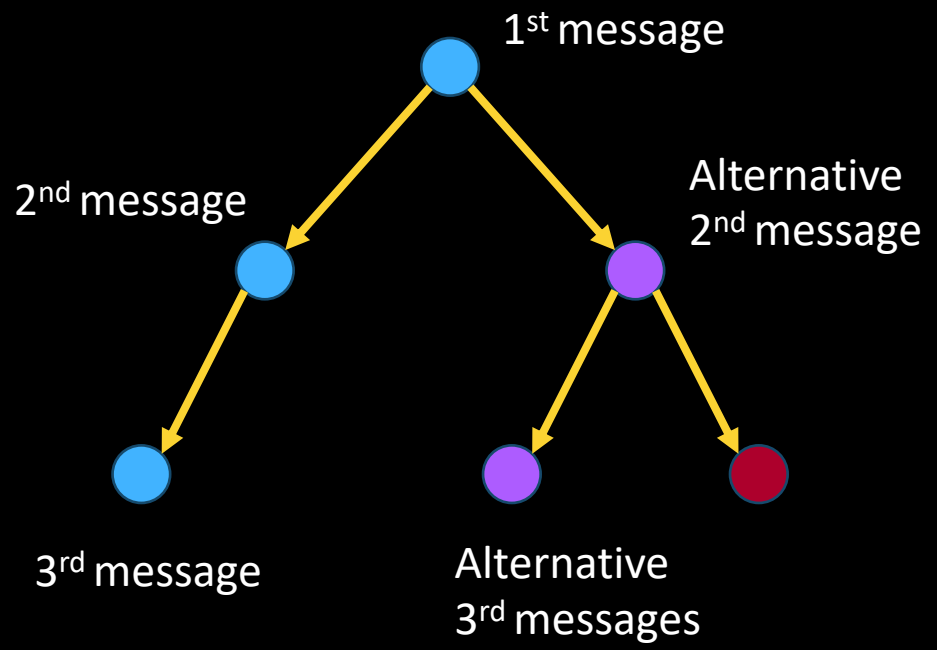
```
from strands import Agent
from strands_tools import graph, swarm, use_agent, think, workflow

meta_agent = Agent(
    system_prompt="""You can dynamically create specialized agents
                     and orchestrate complex workflows.""",
    tools=[graph, swarm, use_agent, think]
)
```

Agents are created with tools and used as tools

Optimizing context for each agent

- Proactive memory **curation**
  - Compaction/summarization after an **isolated task** has completed
  - When you only need to remember the **outcome**, not the internal details

- **State** management
  - Checkpointing
  - Branching



Agents take notes

- Newer models have been trained to use **files**
  - Specific implementation plans
  - Specifications and requirements
  - Track technical and non-technical decisions
- Notes can be found and read **on demand**
  - In the same session, after compaction
  - In the next sessions, as a sort of long-term memory
- This approach can also reduce the number of tools
  - For example, instead of adding a specific task manager tool, ask the model to keep tasks in a file

Simplify / reduce tools

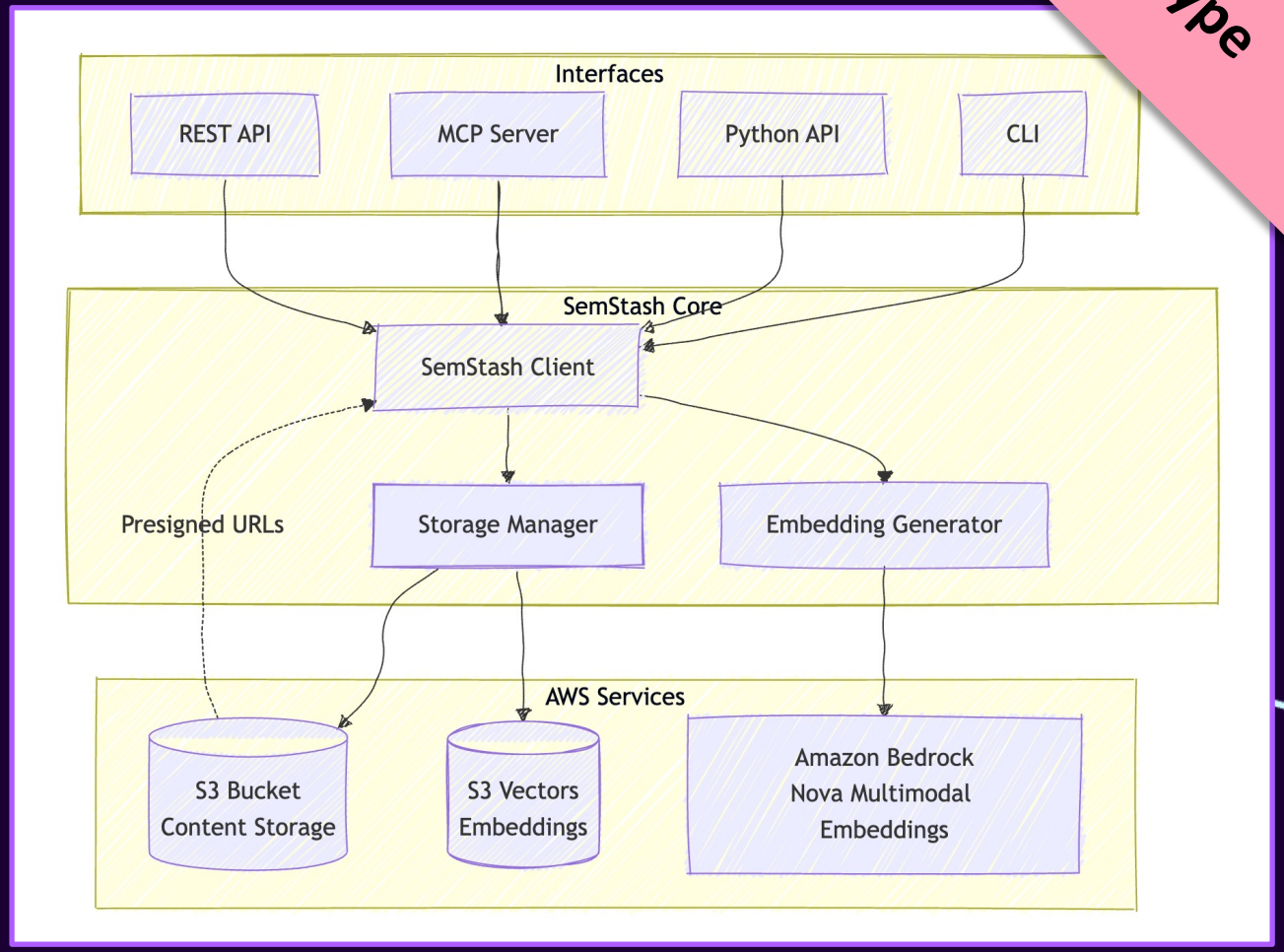
- Models know how to use common **CLI** and **SDK** tools
  - They can write code and scripts
    - Less overhead, no need for new tool descriptions
  - For example: **git**, **gh**, **glab**, AWS CLI and SDK, …
  - For security, you need a **sandbox** environment
    - Code interpreter with terminal access
    - Fine tune third-party permissions
- Multimodal content can add overhead
  - But images can bring **more** information than tokens
- Use a single **semantic space** for information retrieval

“SemStash”

Semantic storage
for humans and AI agents

- REST API
- MCP Server
- Python API
- Command line interface (CLI)

https://github.com/danilop/semstash



Prototype

Deferred loading (Lazy loading)

- A **mechanism** about when information enters context
  - Including domain knowledge, repeatable workflows, new capabilities, and tool definitions
  - Full instructions and schemas aren't loaded until the agent actually needs them
- Implementation
  - Lightweight index/stubs upfront, full definitions fetched on-demand
- Examples
  - Agent Skills
  - Kiro Powers

Agent Skills

A skills-compatible **agent** needs to:

1. Discover skills in configured directories
2. Load metadata (name and description) at startup
3. Match user tasks to relevant skills
4. Activate skills by loading full instructions
5. Execute scripts and access resources as needed

Frontmatter: When to activate

```
---
name: skill-name
description: A description of what this skill does and when to use it.
---
Explaination of the skill and pf tools, CLI/SDK, scripts...
```

KIRO Powers

A Kiro power is a unified **bundle** that gives access to specialized knowledge and includes:

- A steering file (**POWER.md**)
- MCP server configuration
- Steering/hooks - Automated tasks that run on events (optional)

Frontmatter: When to activate

```
---
name: "supabase"
displayName: "Supabase with local CLI"
description: "Build fullstack applications with..."
keywords: ["database", "postgres", "auth", "storage", "realtime", ...]
---
```

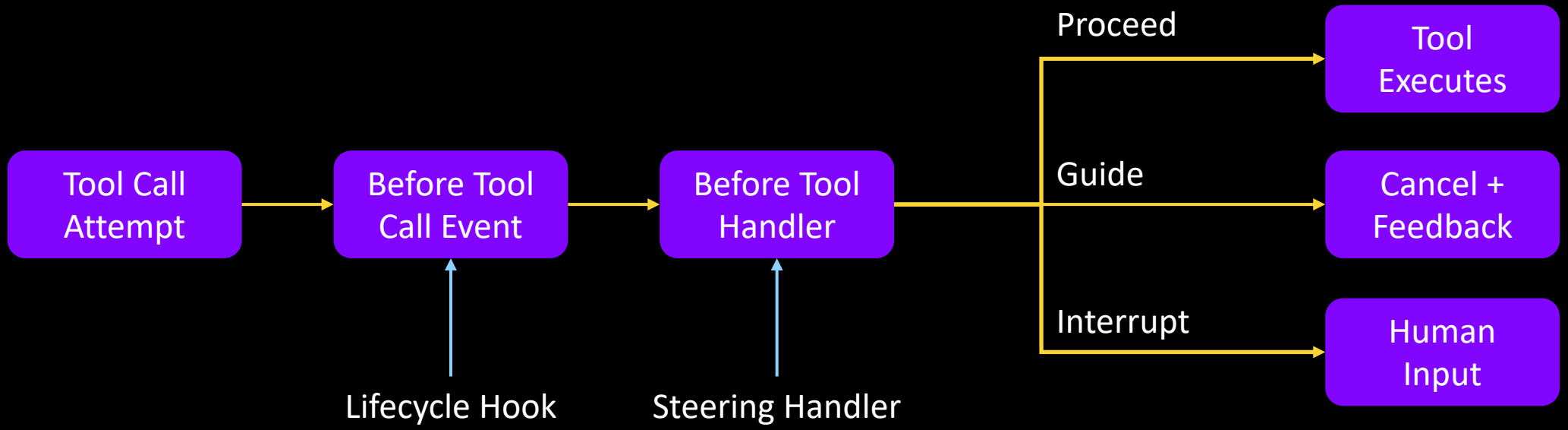
Progressive disclosure

- An **information architecture pattern** about how much is revealed at each stage
- Information is layered
  - Tool categories → Tool names → Descriptions → Full schemas
  - Steer agents via **context-aware** guidance
- Examples
  - Agent Skills → Scripts, Resources, Assets
  - MCP search tools
  - AgentCore Gateway semantic search
  - Episodic memory in AgentCore Memory
  - Strand Agents Steering (experimental)

Strands Agents Steering – Tool Steering

Experimental

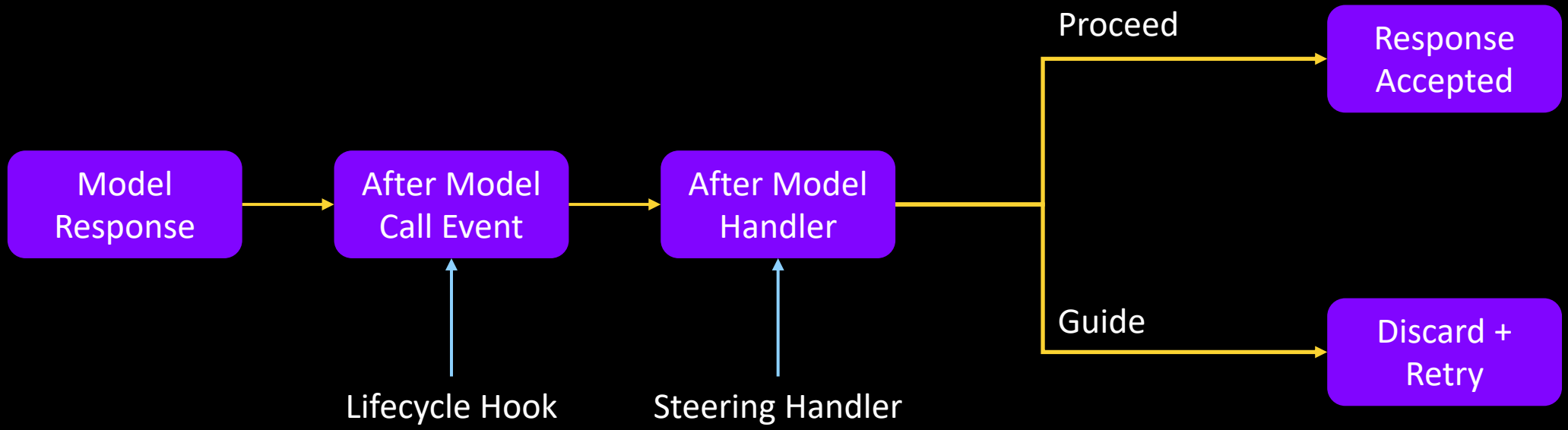
When agents attempt tool calls, steering handlers evaluate the action



Strands Agents Steering – Model Steering

Experimental

After each model response, steering handlers evaluate output

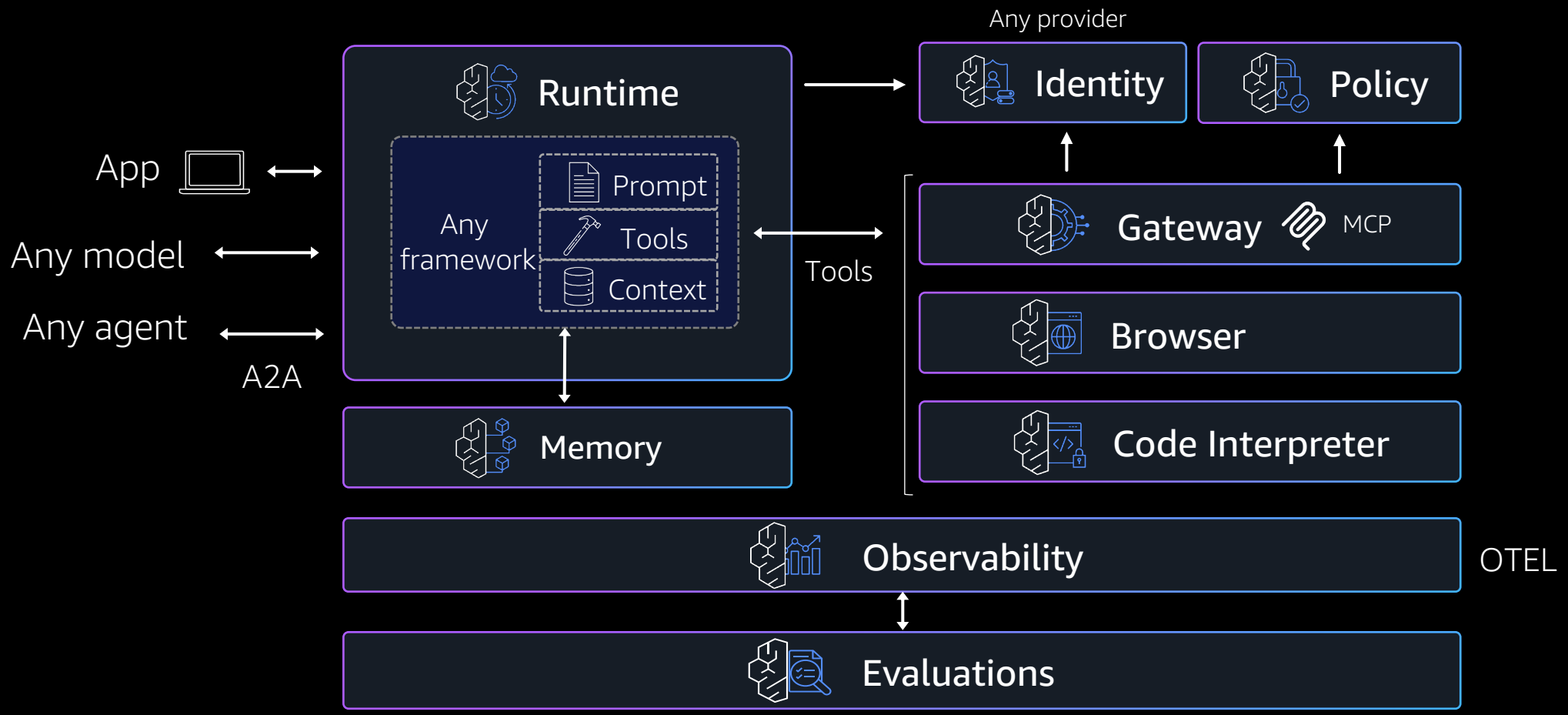


Strands Agents Sample Code

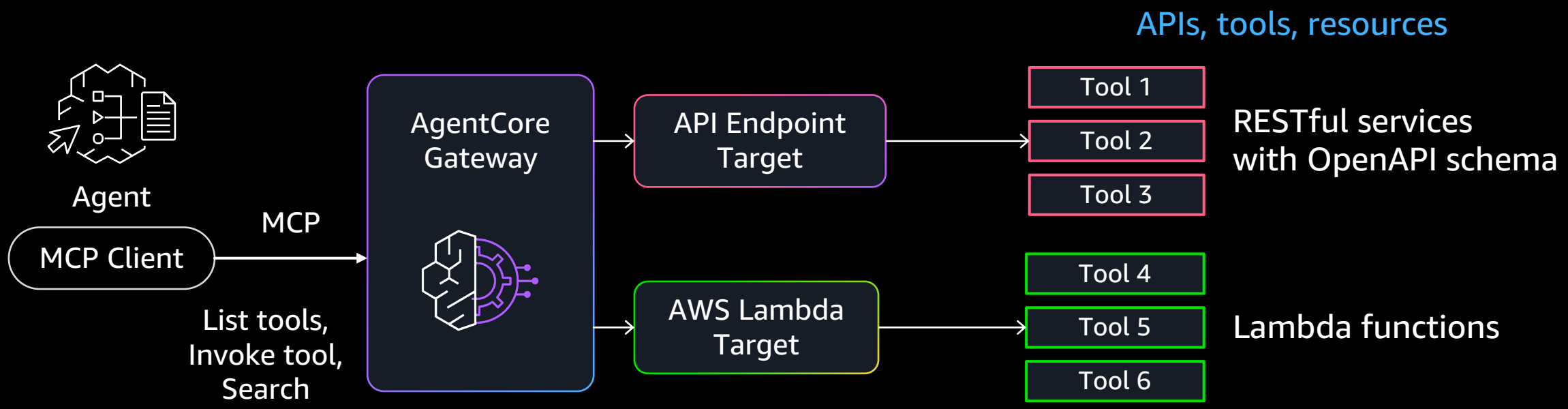
```python
from strands import Agent, tool
from strands.experimental.steering import LLMSteeringHandler

@tool
def send_email(recipient: str, subject: str, message: str) -> str:
    """Send an email to a recipient."""
    return f"Email sent to {recipient}"

handler = LLMSteeringHandler(system_prompt="""
    You are providing guidance to ensure emails maintain a positive tone:
    - Review email content for tone and sentiment
    - Suggest more cheerful phrasing if the message seems negative or neutral
    - Encourage use of positive language and friendly greetings
    When agents attempt to send emails, check if the message tone
    is appropriately cheerful and provide feedback if improvements are needed.
    """
)

agent = Agent(tools=[send_email], hooks=[handler] )

response = agent("Send a frustrated email to tom@example.com, a client who ...")

print(agent.messages)  # "Tool call cancelled given new guidance..."
```

Amazon Bedrock AgentCore Platform

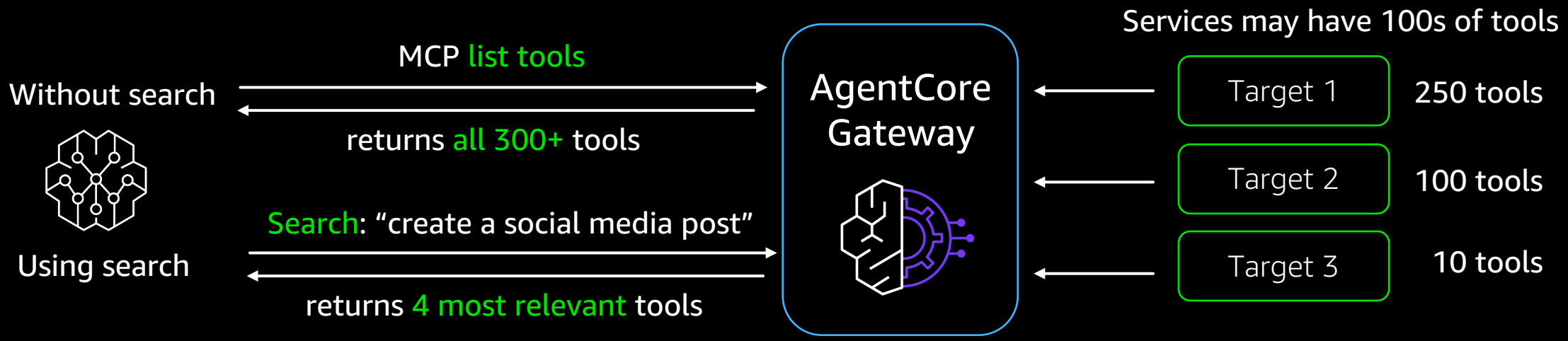
Model and Framework Agnostic



AgentCore Gateway



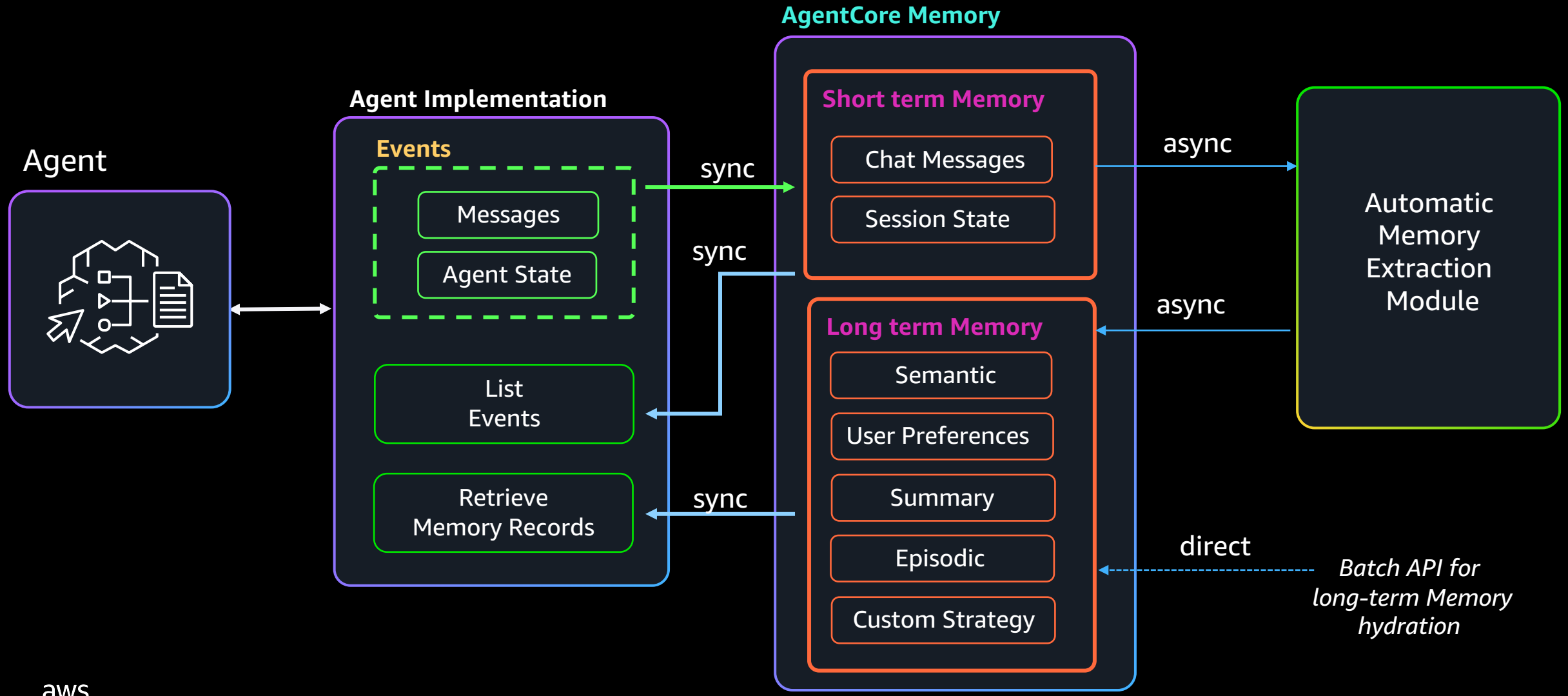
AgentCore Gateway semantic search



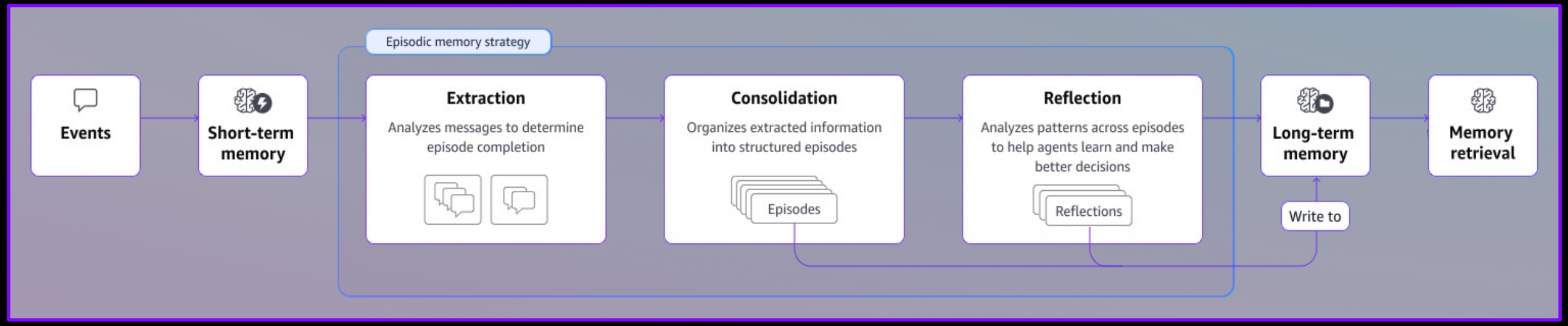
Benefits

- AgentCore Gateway automatically indexes tools and gives serverless semantic search
- Reduces context passed to the agent’s LLM, improving accuracy, speed, and cost
- Lets agent focus on tools relevant for a given task

AgentCore Memory



AgentCore Memory – Episodic Memory

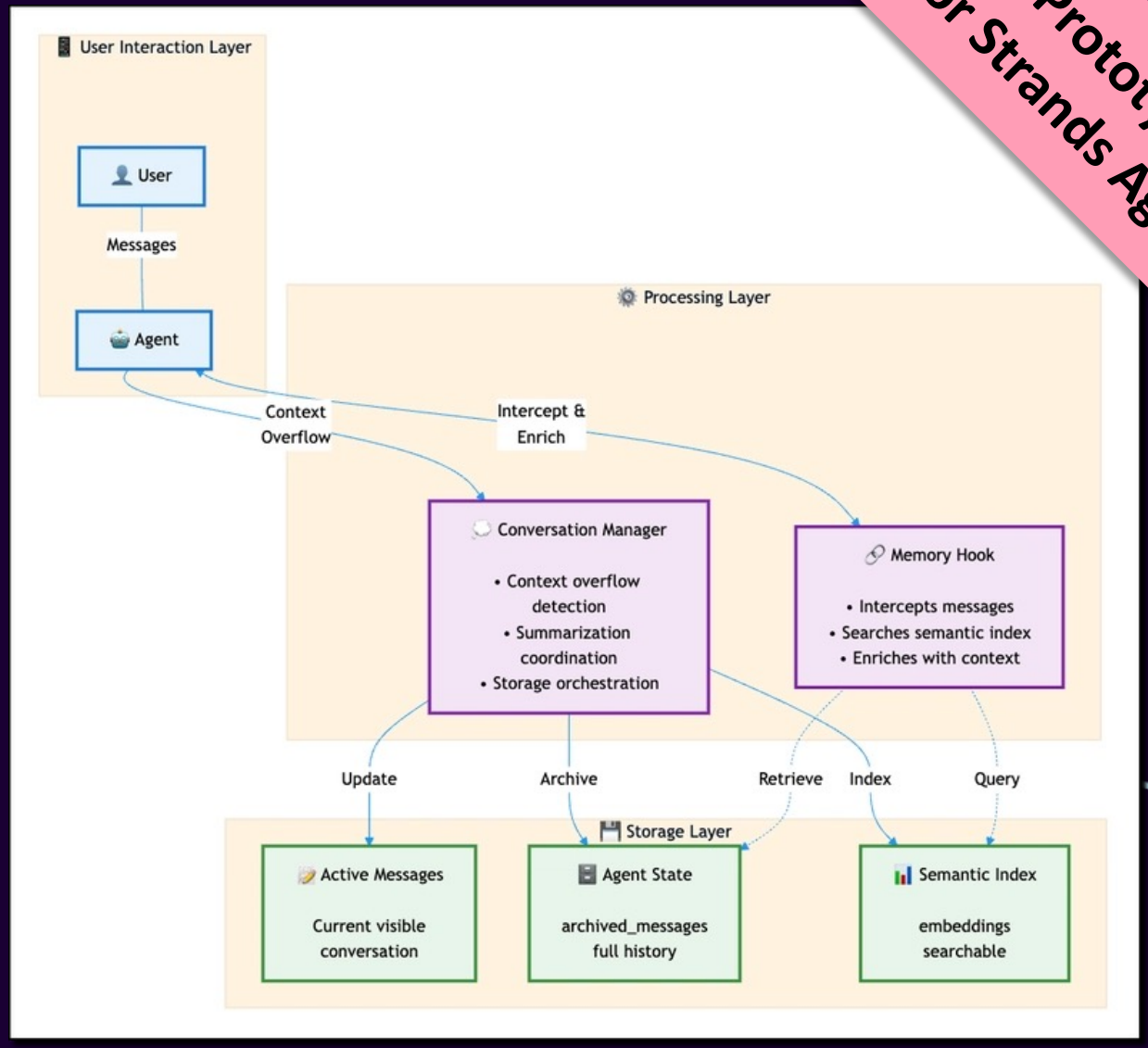


Avoid repeating mistakes without filling the system prompt of all possible scenarios

“Semantic Summarizing Conversation Manager”

Prototype for Strands Agents

Combines summarization with exact message recall using semantic search

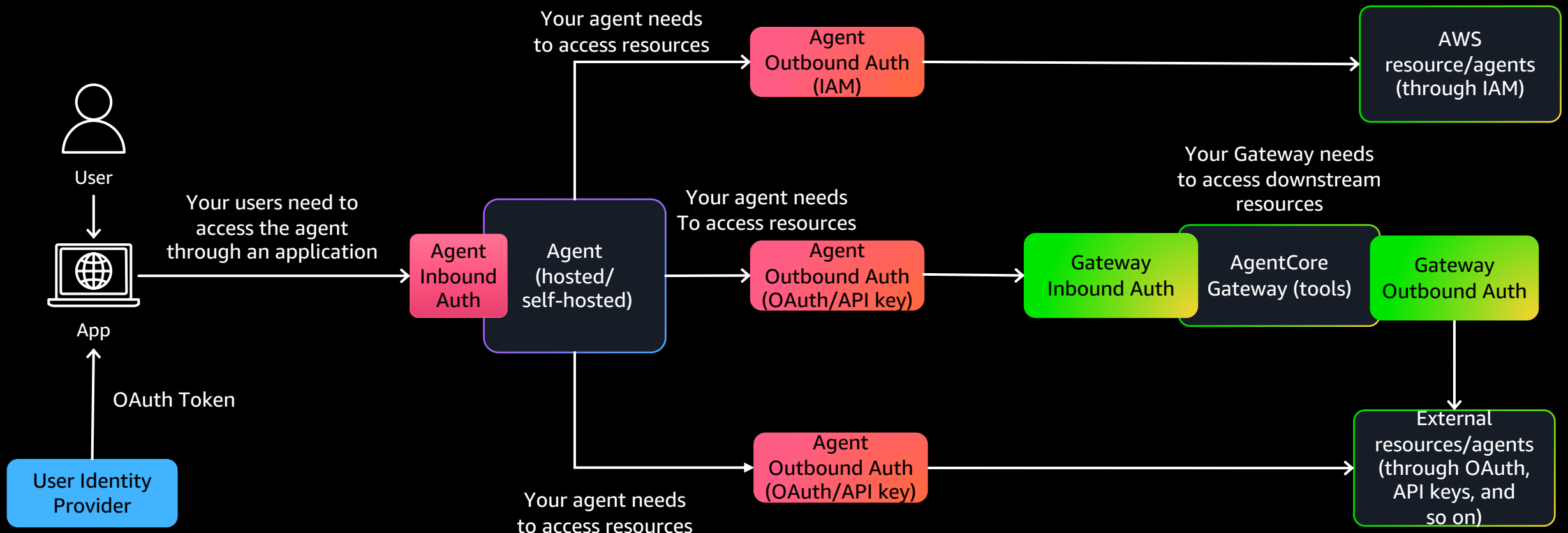


https://github.com/danilop/strands-agents-semantic-summarizing-conversation-manager

Runtime controls

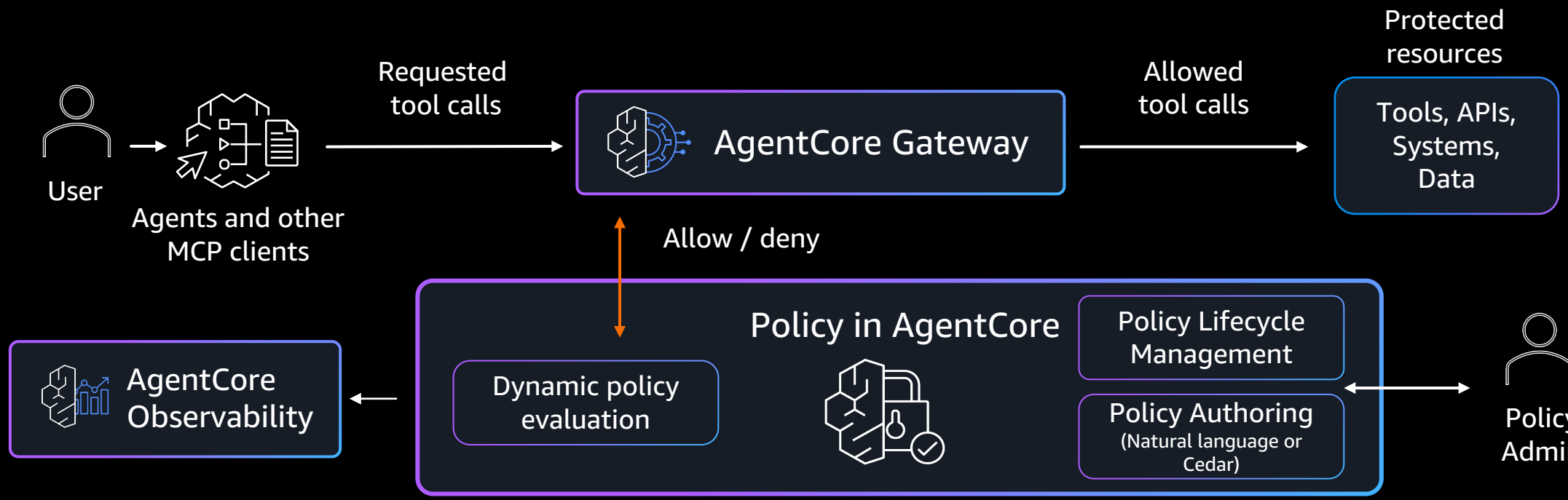
- Too much **information** in the context, including too many tools, can increase errors
  - When the tools is used, how **safe** is that? Risk assessment
  - Sandbox for code interpreter and shell access
- Validate tools **access** using the user identity and access policies
  - Who can use a tool and with which parameters
  - Automated reasoning to evaluate policies
- Examples
  - AgentCore Identity, Policy in AgentCore
  - Cedar – Open source policy language

AgentCore Identity



PREVIEW

Policy in AgentCore



Keep agents in bounds

Act autonomously, but stay within boundaries and compliance

Instant and consistent

Policies evaluated in milliseconds, without slowing agents

Verifiably correct

Built on years of automated reasoning, using Cedar

Cedar
Open Source
Policy Language

```
Tool call: process_refund(amount=300)
```

```
permit(
  principal is AgentCore::OAuthUser,
  action == AgentCore::Action::"RefundTool__process_refund",
  resource == AgentCore::Gateway::"arn:aws:bedrock-agentcore:..."
)
when {
  principal.hasTag("group") &&
  principal.getTag("group") == "admin" &&
  context.input.amount < 500
};
```

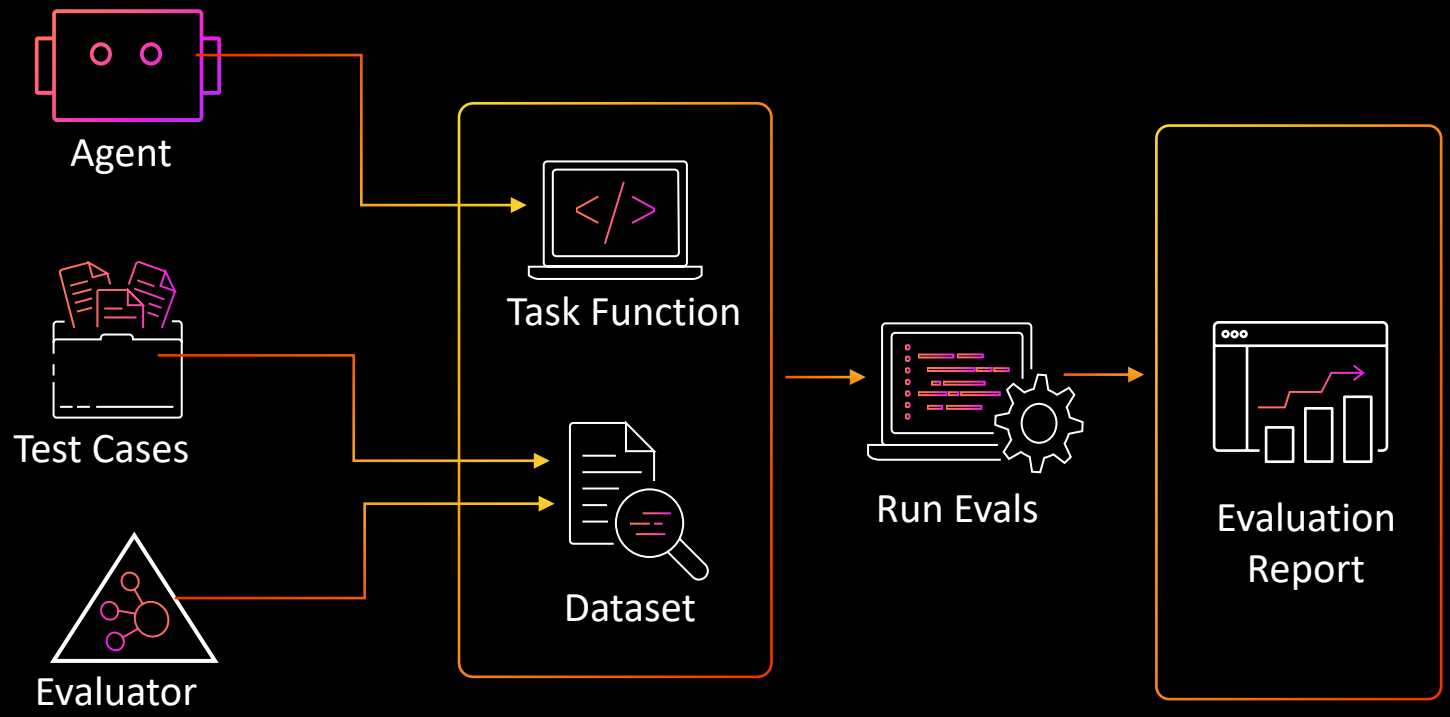
Evaluations

- It is **easier** to evaluate a response than to build it
- To be sure that context and tool optimizations work as **expected**
  - LLM as judge
  - Simulated users
- Metadata can help and simplify evals
  - Such as **sources** and **citations** for (agentic) RAG
- Examples
  - Strands Agents Evaluations
  - AgentCore Evaluations

Strands Agents Evaluations

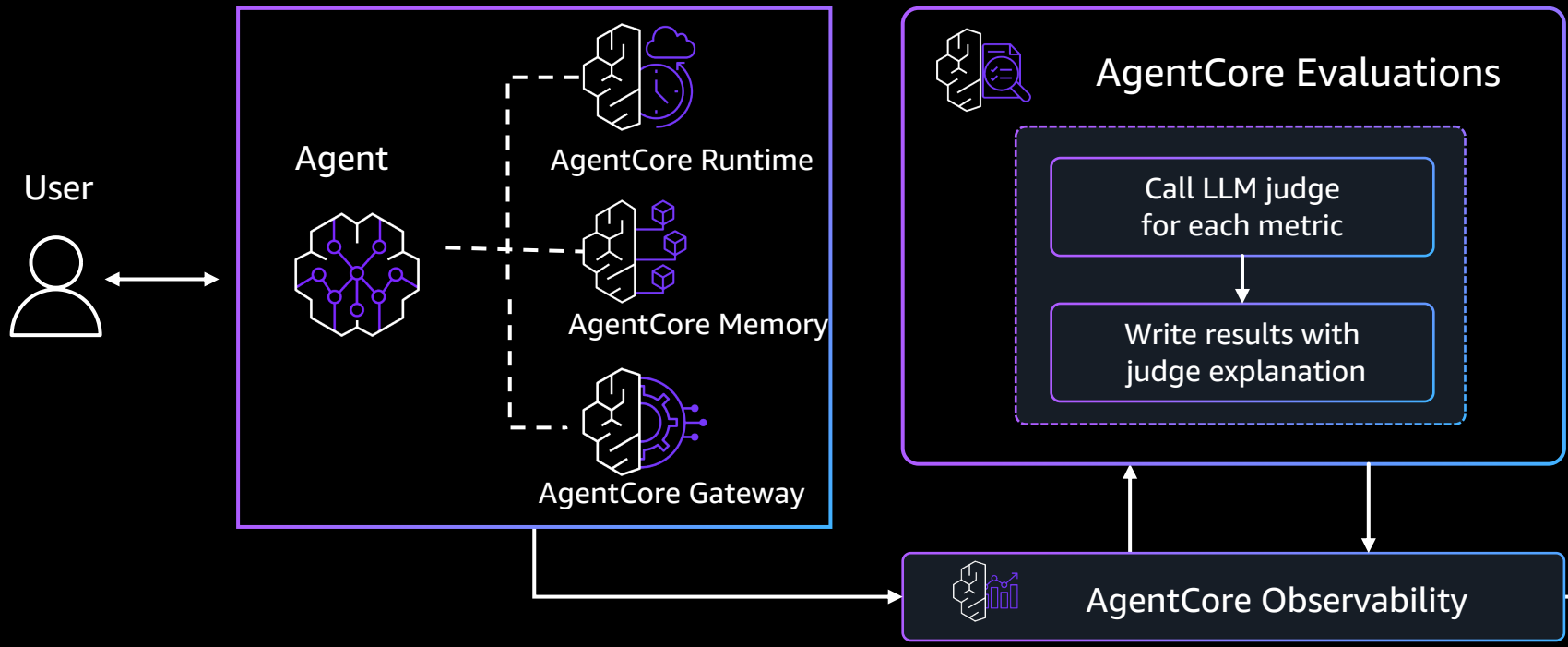
An evaluation framework for systematically testing and benchmarking agents.

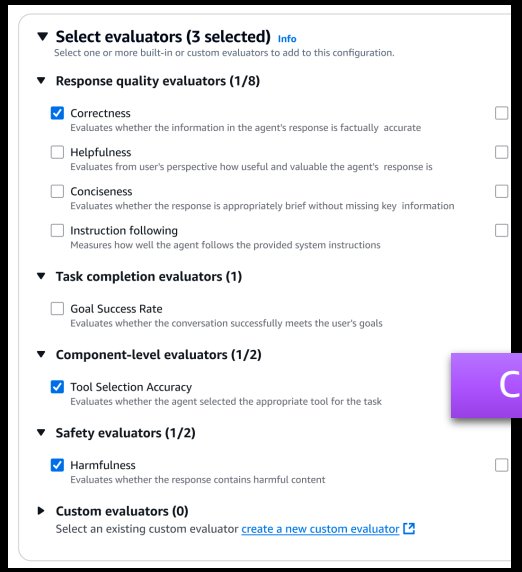
- Define test cases and evaluators, execute them against any task function
- Collect structured results for comparison and regression testing
- Uses LLM-as-judge pattern where evaluators assess outputs against natural language rubrics
- Supports evaluation across levels using 8 built-in evaluators



AgentCore Evaluations

PREVIEW





Configure

On-demand

Agent owner

Monitor and assess

Path to production

The prototype to production “chasm”

Excitement and potential

Challenges on the path to production

Meaningful business value

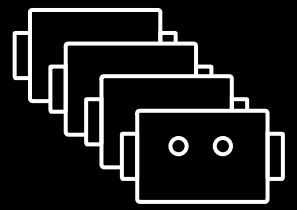
POC

Performance

Scalability

Security

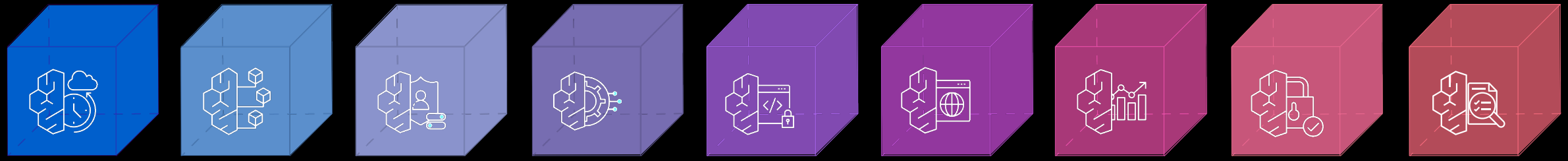
Governance



AI production agents

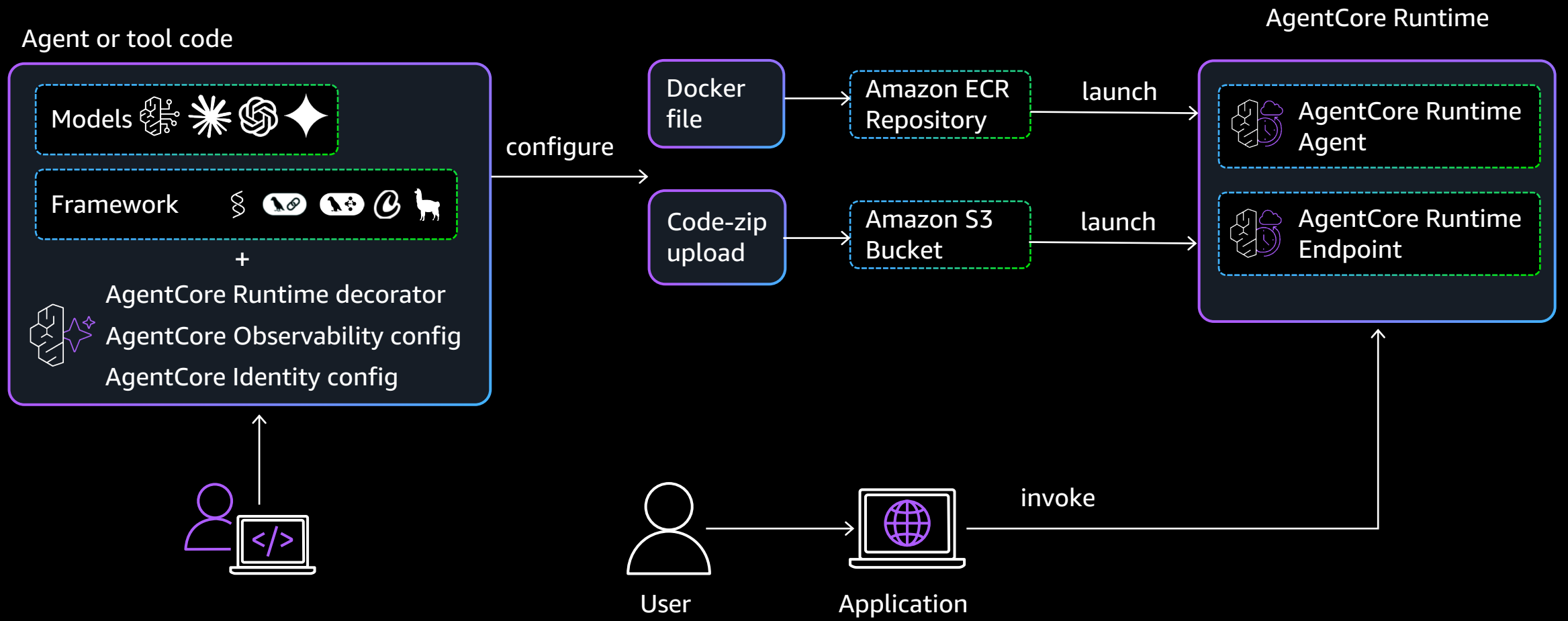
Amazon Bedrock AgentCore

Comprehensive agentic platform: Everything you need for getting agents into production



Runtime | **Memory** | **Identity** | **Gateway** | **Code Interpreter** | **Browser** | **Observability** | **Policy** | **Evaluations**

AgentCore Runtime



Deploying to a prototype agent to a remote stateless environment

AgentCore Runtime

- Forces to **think** about
  - User authentication and authorization
  - Session management and retrieval
  - Short and long-term memory requirements (across sessions)
  - Observability (using OpenTelemetry)
  - Evaluations (to test non-deterministic output)

- Makes prototypes **evolve** towards production naturally

Takeaways

Takeaways

Context size is often the bottleneck

Use more than one agent to split context and tools

Simplify and reduce tools using notes and direct CLI/SDK access

Implement deferred loading and progressive disclosure

Introduce runtime controls for governance

Less can be more, use evals to confirm

Thank you!

Danilo Poccia
@danilop