AWS re:Invent

DECEMBER 1 – 5, 2025 | LAS VEGAS, NV

CNS359

Build, deploy, and operate agentic architectures on AWS Serverless

Heeki Park

(he/him)

Principal Solutions Architect

AWS

Dhiraj Mahapatro

(he/him)

Principal Solutions Architect

AWS

Agenda

- Designing single-agent systems
- Applying serverless to agentic patterns
- Developing agents
- Scaling with open standards
- Deploying agents
- Transitioning to multi-agent systems
- Baking in security and governance for deployments

Assumptions

Level 300 session

Principles first

Designing single-agent systems

Starting with a customer issue

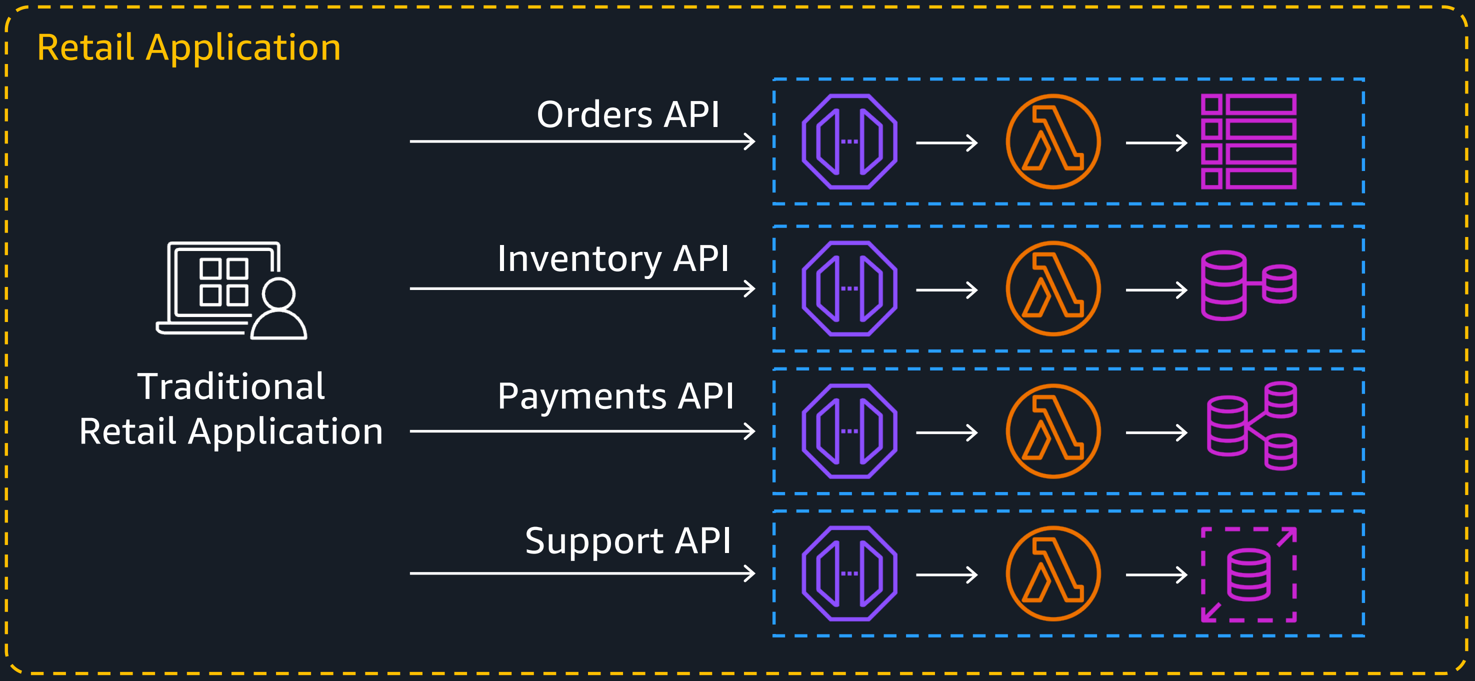
1 2

I was charged twice for my order, only received one of the two items in the order, and I need it tomorrow for a gift.

3

Can you assist in rectifying and expediting?

Starting with serverless APIs



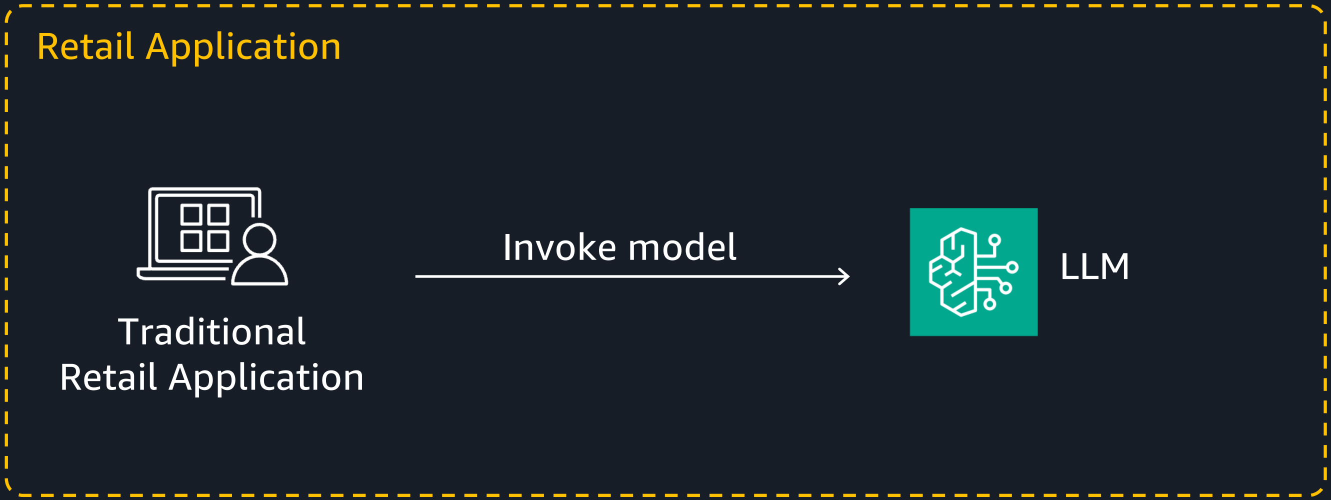
Considering customer experience

Challenges:

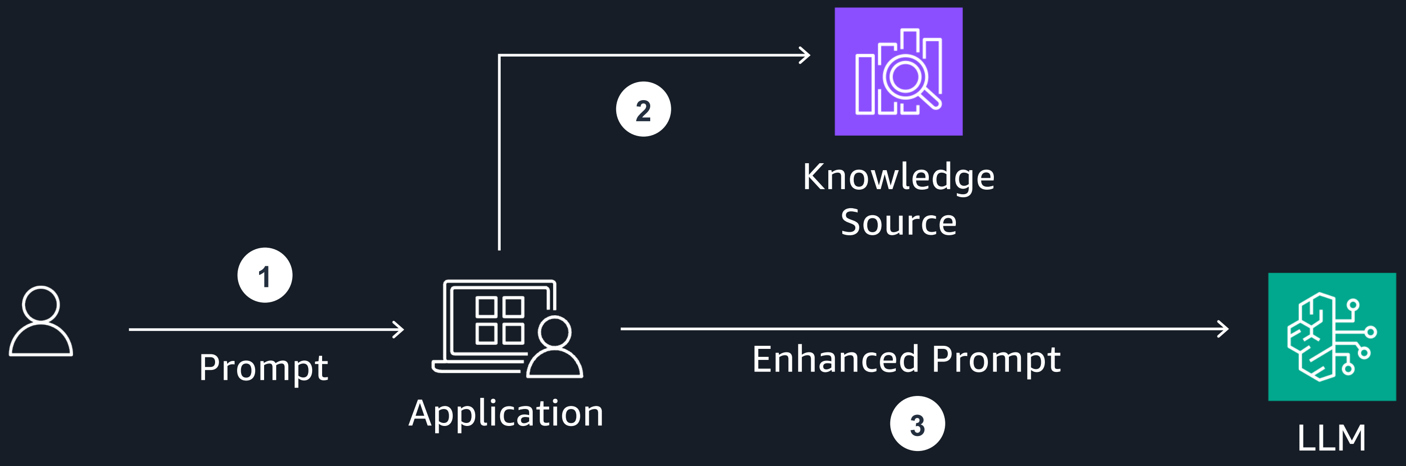
1. Responding to customer requests requires multiple lookups in disparate systems.
2. Customers need faster time to resolution.
3. Support agents need faster guidance when responding to customer calls or chat requests.

Can we do better?

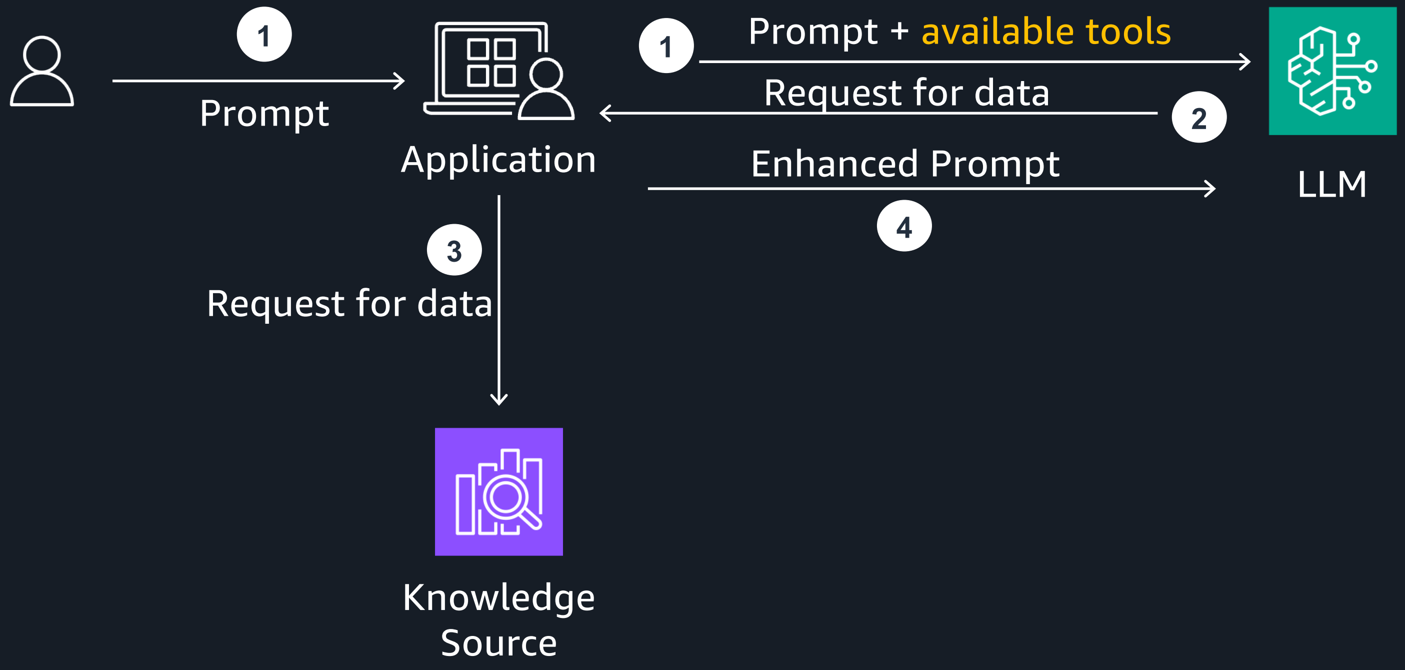
Adding LLMs to applications



Providing more context through search



Providing more context dynamically

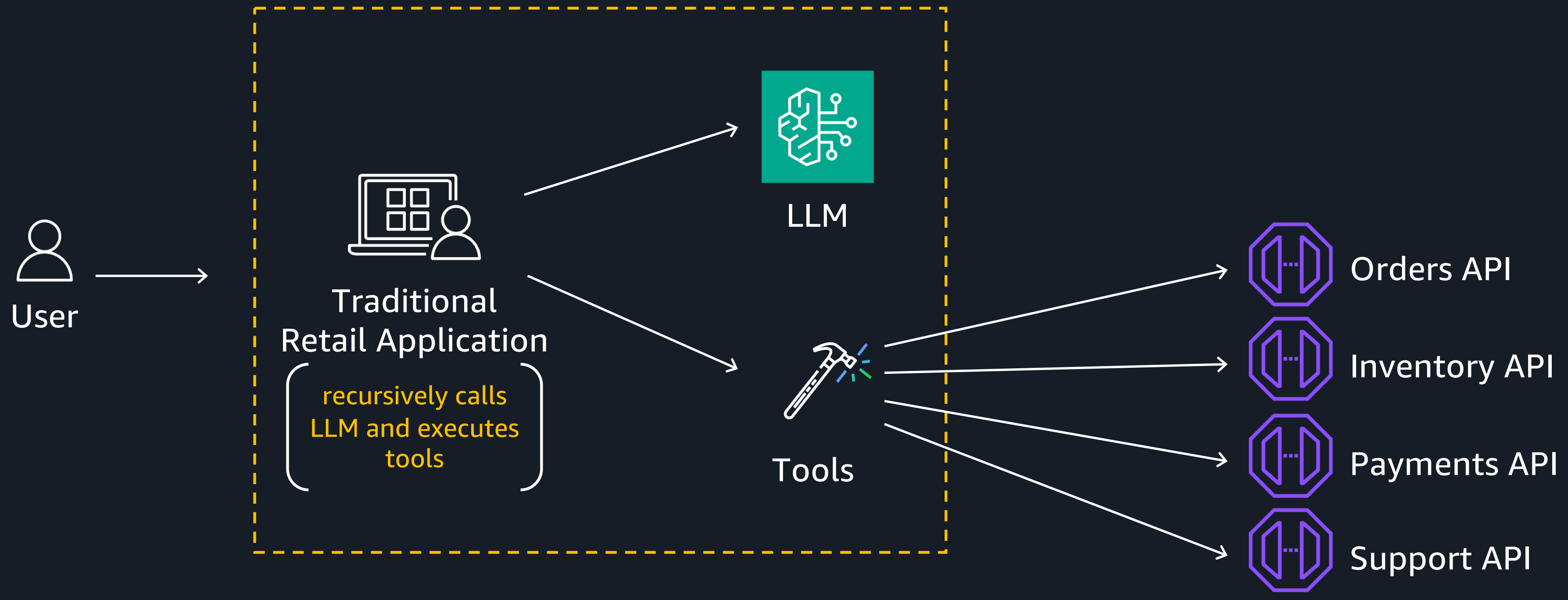


What does an enhanced prompt look like?

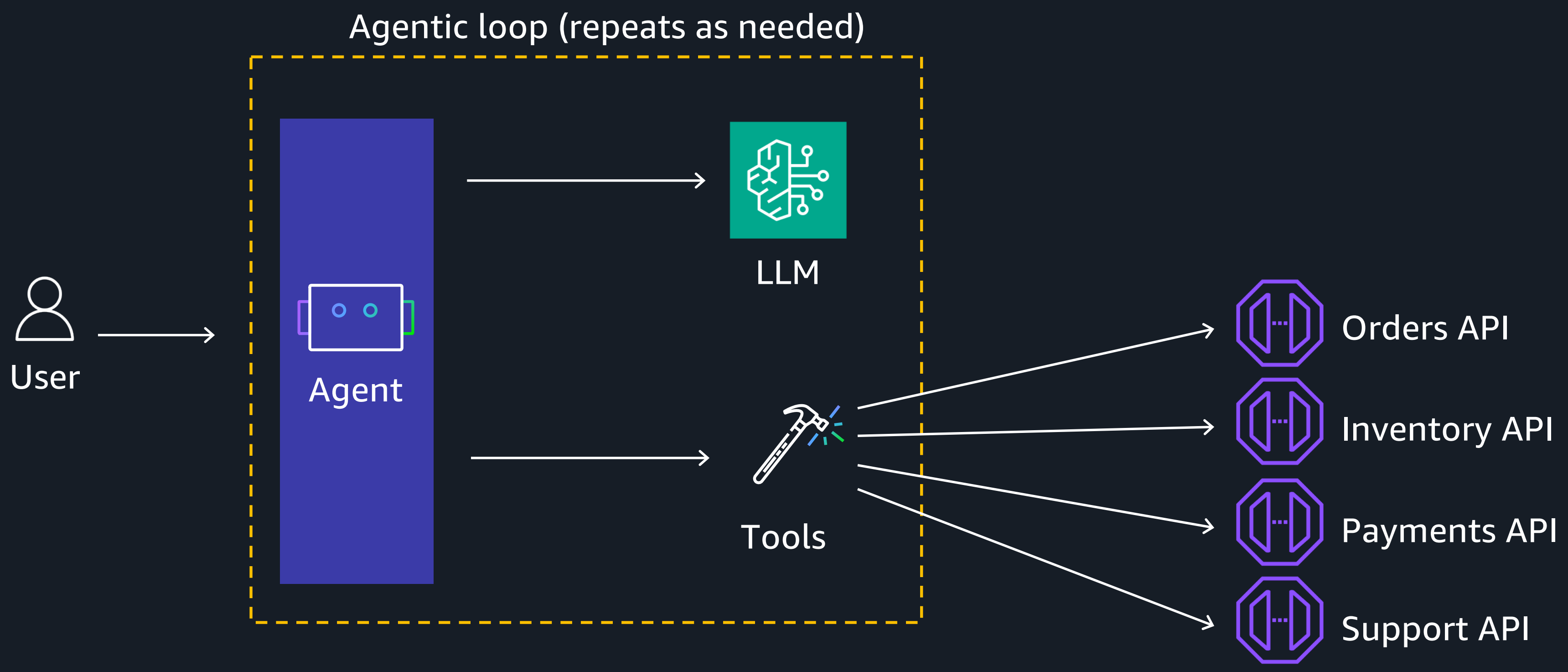
ADDITIONAL CONTEXT

```
{
    "system_prompt": "You are a helpful assistant..",
    "tools": [],
    "context": "...",
    "conversation_history": "...",
    "user_info": "...",
    "human_prompt": "..."
}
```

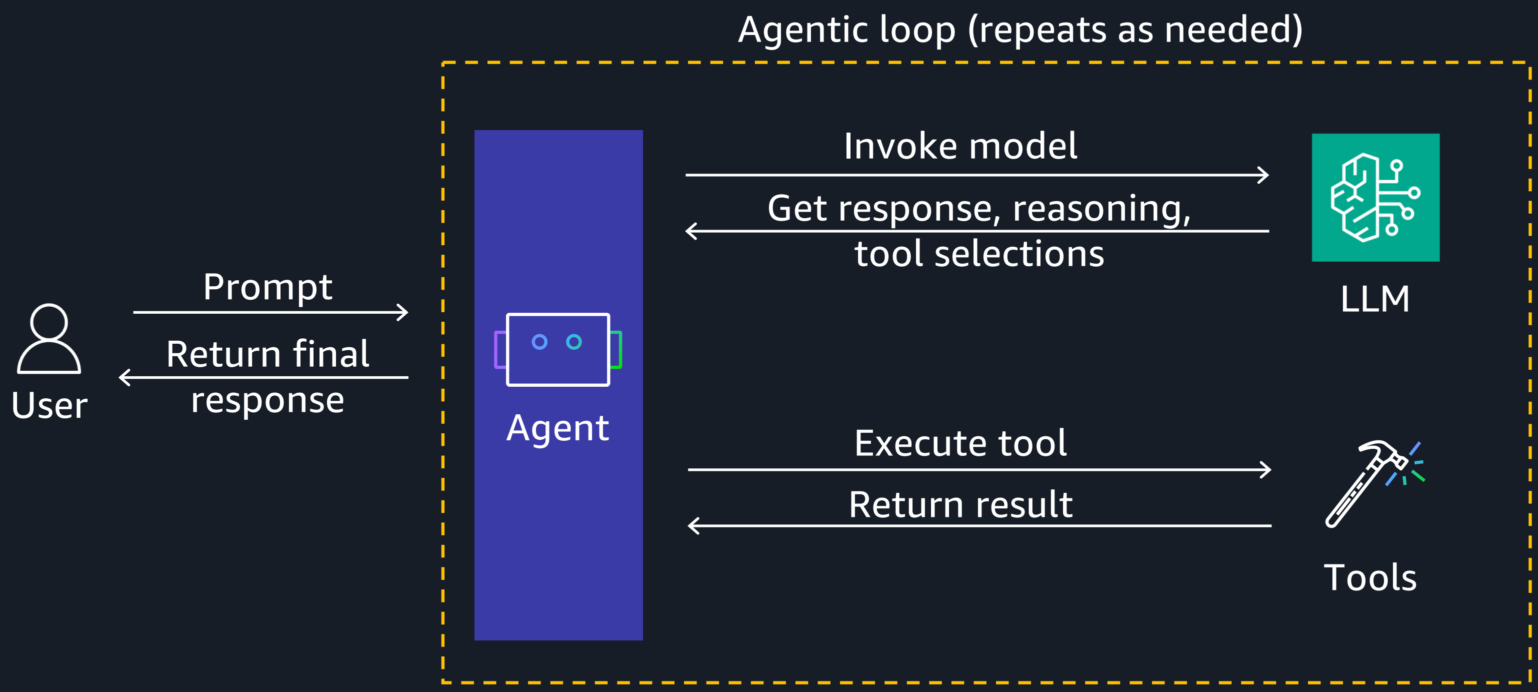
Providing context dynamically **through APIs**



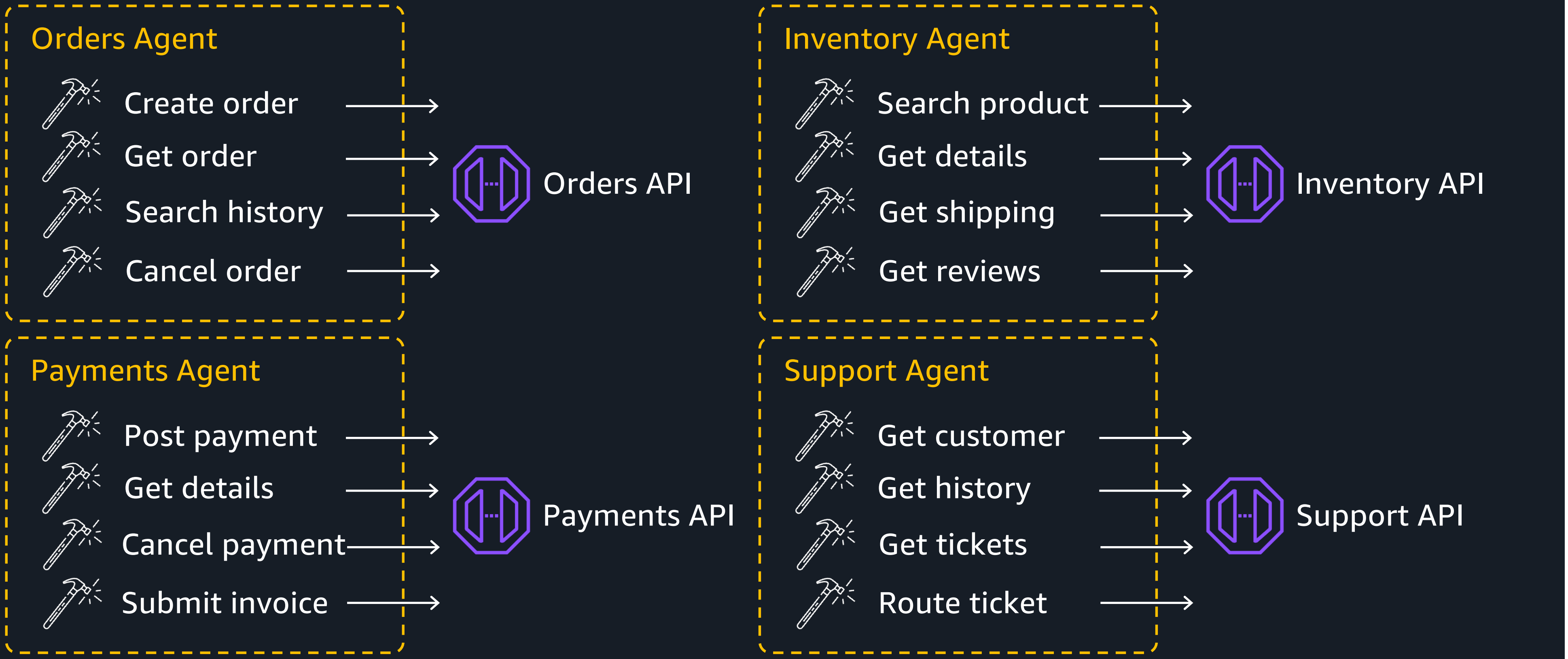
Transitioning to agents



Why agents?


Agentic loop (repeats as needed)
User
Prompt
Return final response
Agent
Invoke model
Get response, reasoning, tool selections
LLM
Execute tool
Return result
Tools

Decomposing the agentic monolith



Orders Agent

- Create order
- Get order
- Search history
- Cancel order

Orders API

Inventory Agent

- Search product
- Get details
- Get shipping
- Get reviews

Inventory API

Payments Agent

- Post payment
- Get details
- Cancel payment
- Submit invoice

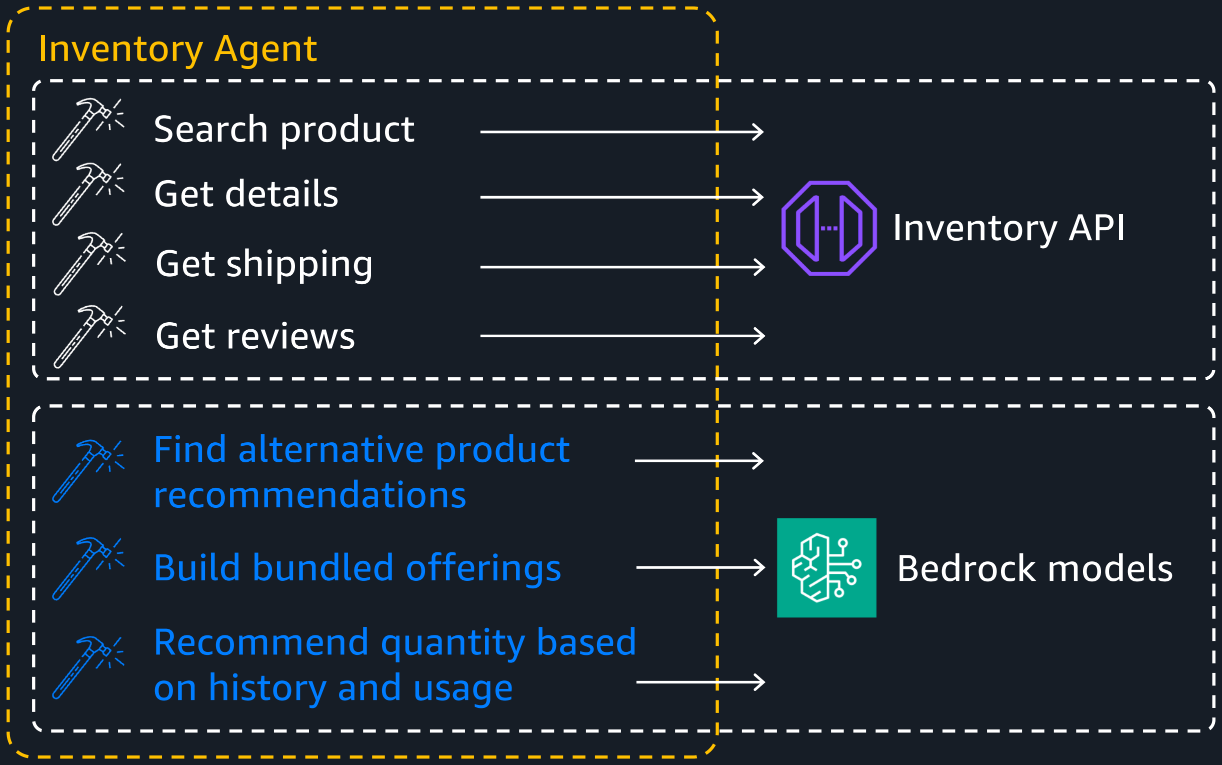
Payments API

Support Agent

- Get customer
- Get history
- Get tickets
- Route ticket

Support API

Refining domain agents



Inventory Agent

- Search product
- Get details
- Get shipping
- Get reviews

→ Inventory API

Initial tool functionality based on existing API capabilities

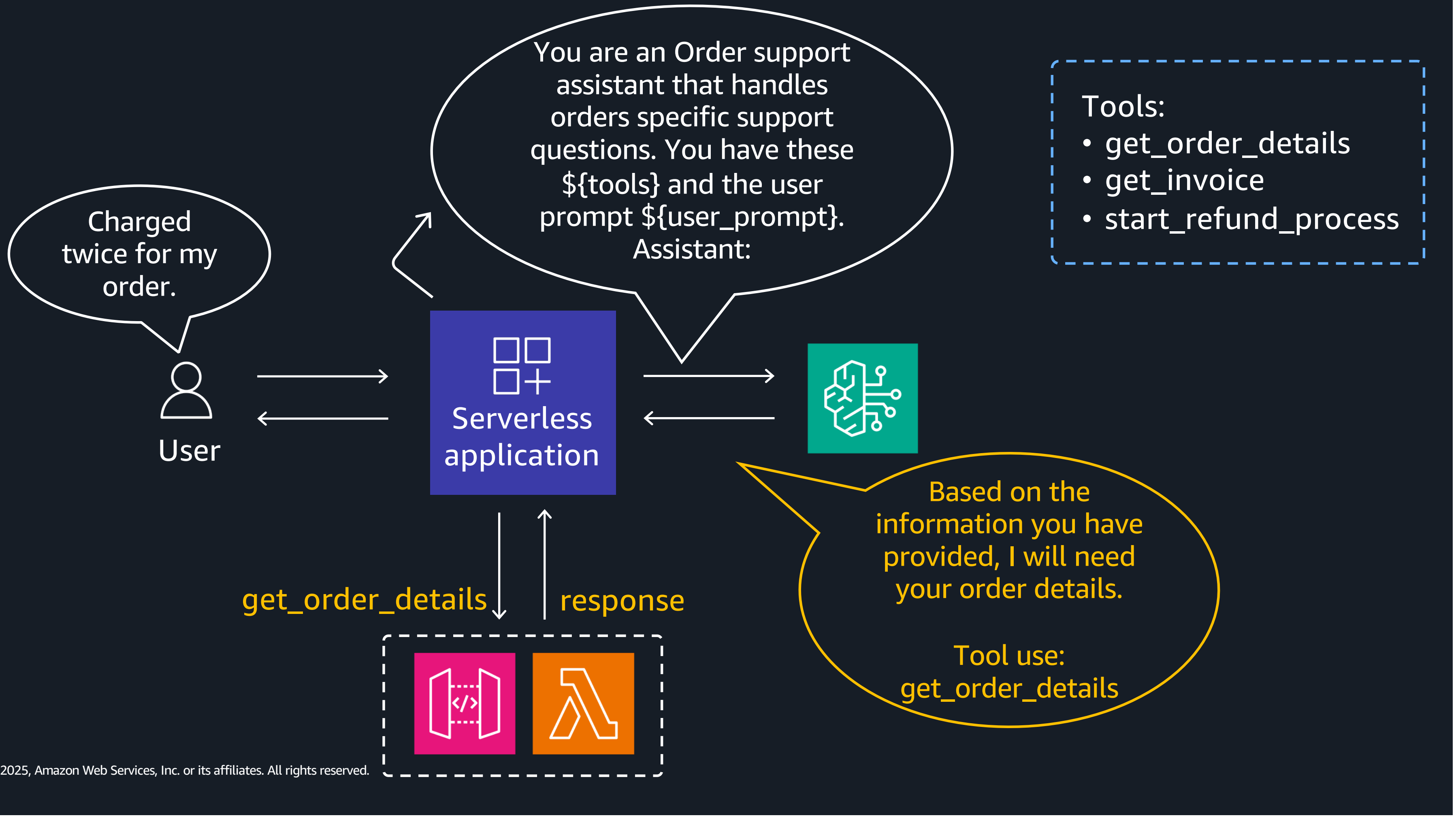
- Find alternative product recommendations
- Build bundled offerings
- Recommend quantity based on history and usage

→ Bedrock models

Extended tool functionality based on model inference

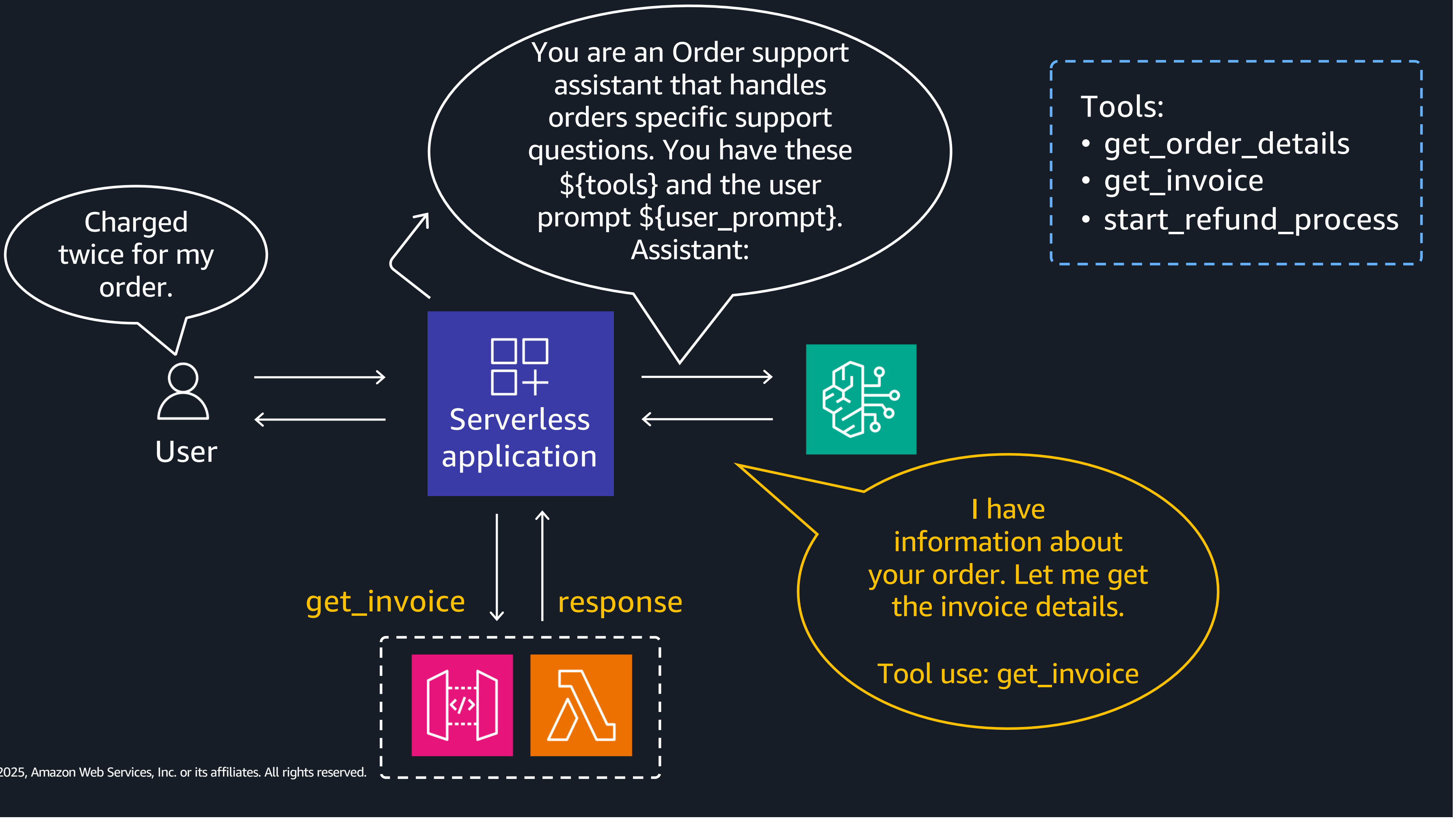
Applying serverless to agentic patterns

Serverless with Tool use



Charged twice for my order.
User
Serverless application
You are an Order support assistant that handles orders specific support questions. You have these ${tools} and the user prompt ${user_prompt}. Assistant:
Tools:
• get_order_details
• get_invoice
• start_refund_process
Based on the information you have provided, I will need your order details.
Tool use: get_order_details
get_order_details
response

Serverless with Tool use



Charged twice for my order.

User

Serverless application

You are an Order support assistant that handles orders specific support questions. You have these ${tools} and the user prompt ${user_prompt}. Assistant:

Tools:
- get_order_details
- get_invoice
- start_refund_process

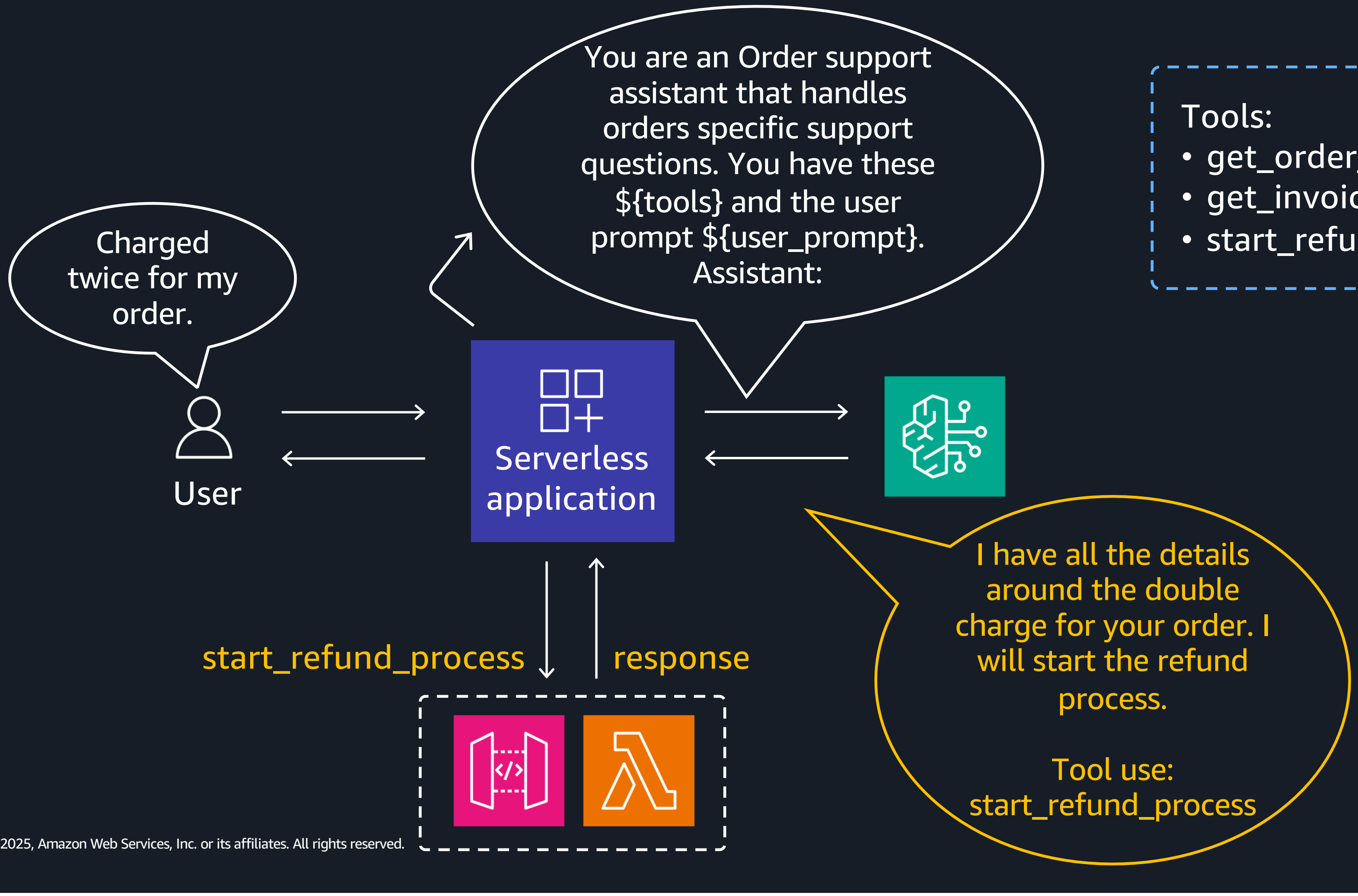
I have information about your order. Let me get the invoice details.

Tool use: get_invoice

get_invoice

response

Serverless with Tool use



Charged twice for my order.

User

Serverless application

You are an Order support assistant that handles orders specific support questions. You have these ${tools} and the user prompt ${user_prompt}. Assistant:

start_refund_process

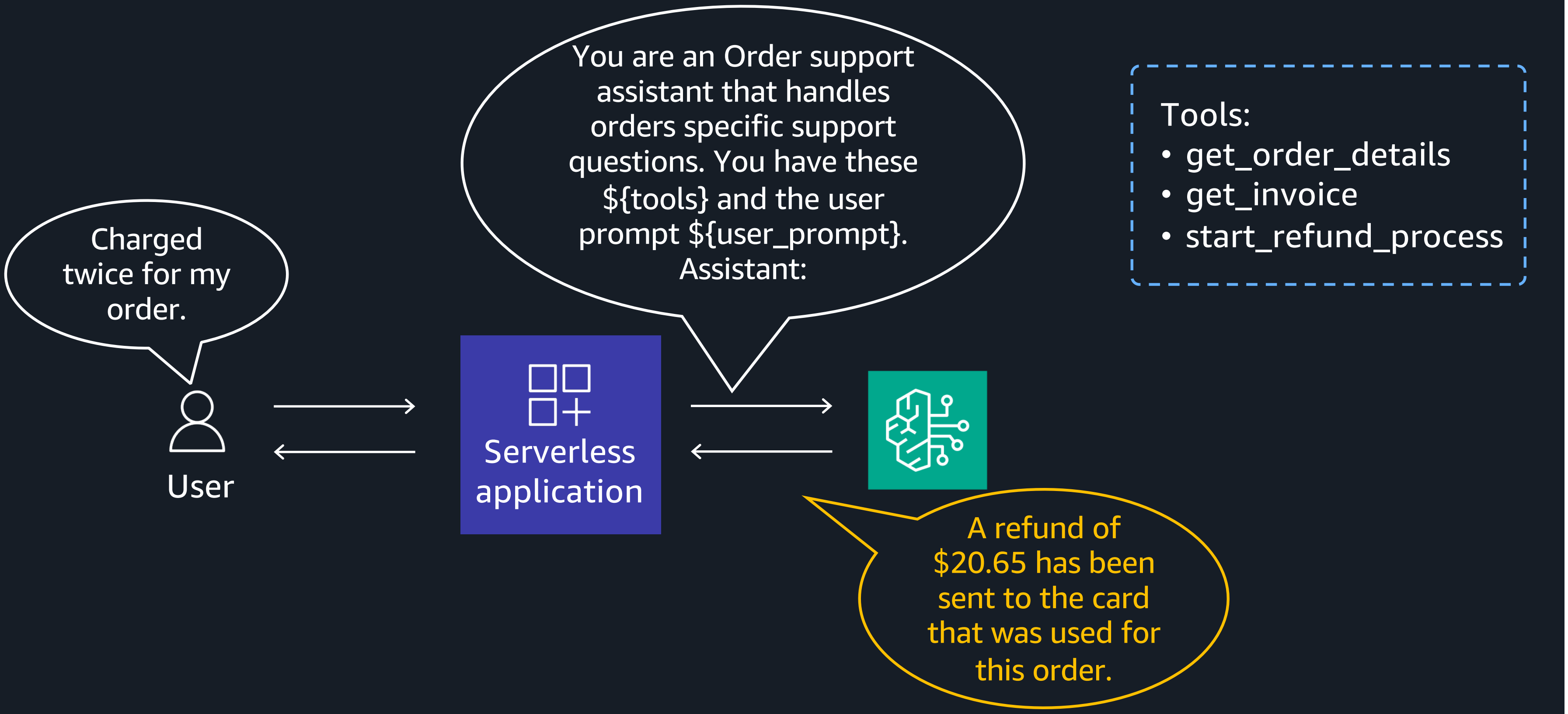
response

I have all the details around the double charge for your order. I will start the refund process.

Tool use: start_refund_process

Tools:
- get_order_details
- get_invoice
- start_refund_process

Serverless with Tool use



Charged twice for my order.

User

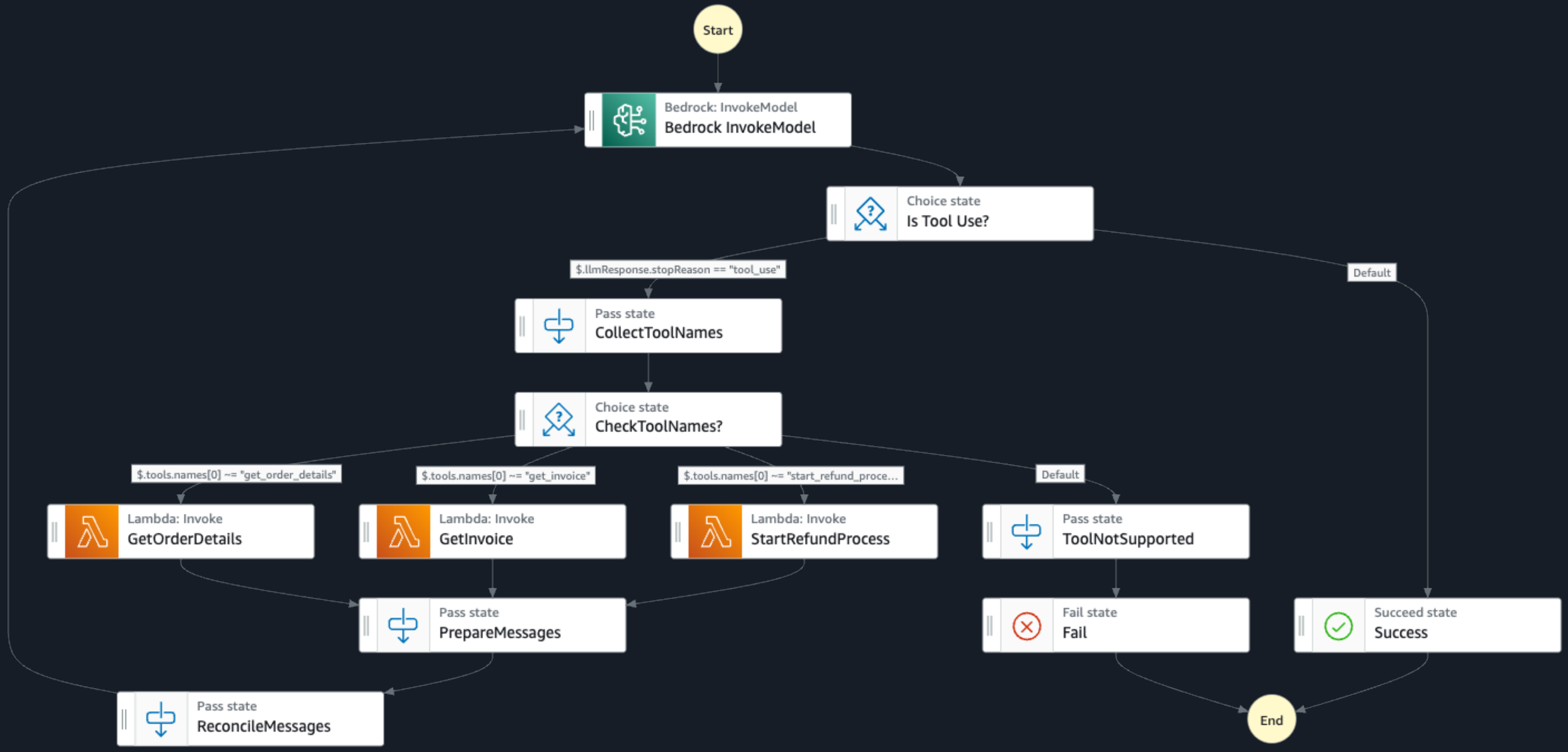
Serverless application

You are an Order support assistant that handles orders specific support questions. You have these ${tools} and the user prompt ${user_prompt}. Assistant:

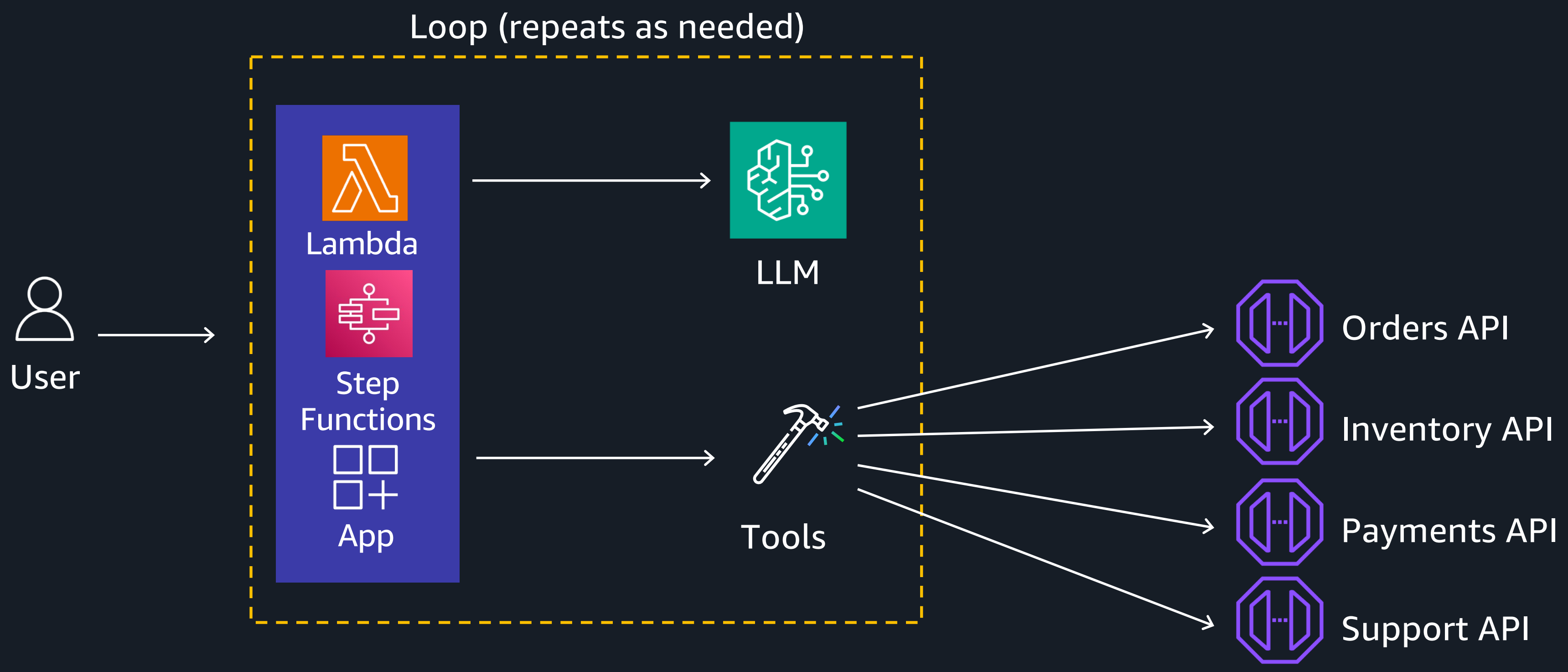
Tools:
- get_order_details
- get_invoice
- start_refund_process

A refund of $20.65 has been sent to the card that was used for this order.

Tool use in AWS Step Functions



Emerging pattern



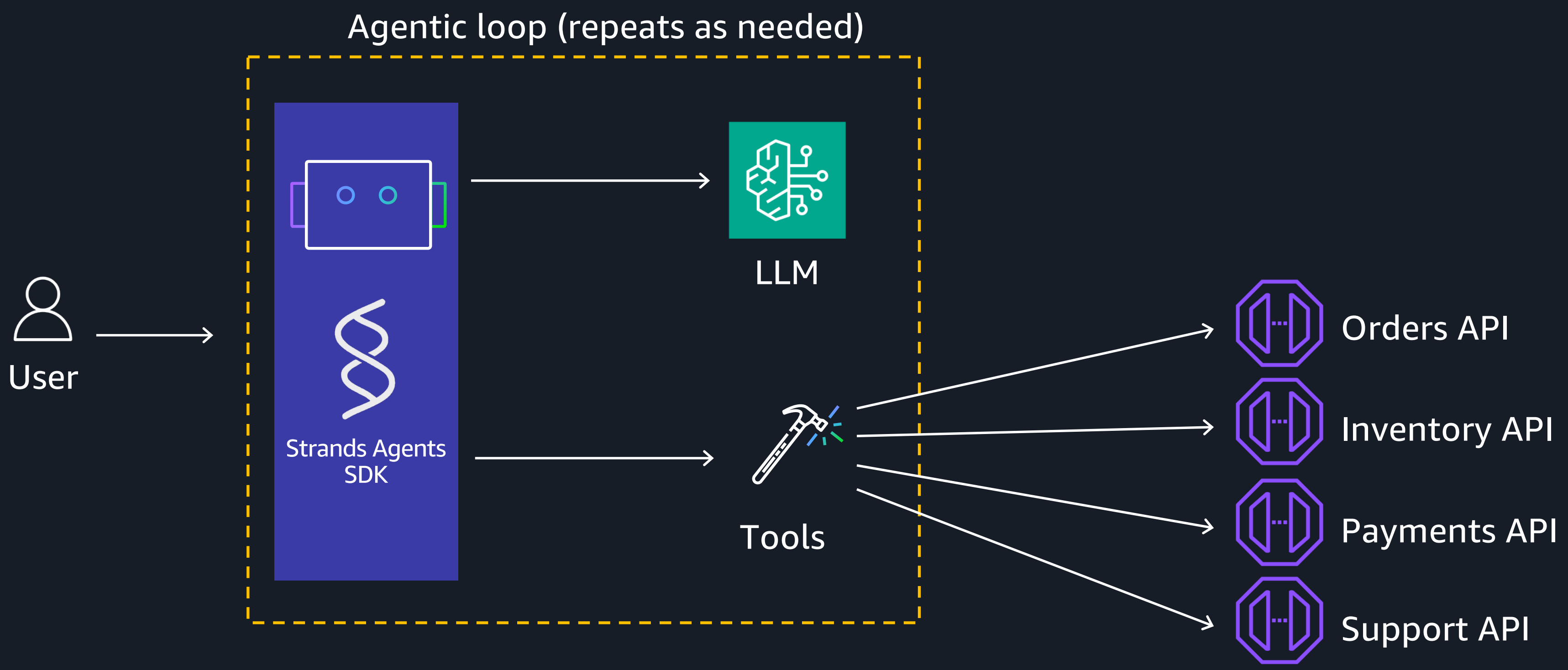
Tool use in Lambda or Step Functions

Converse/Invoke API
Amazon Bedrock

Imperative code to
act on behalf of
LLM

Predefined pathways
& workflows

Agentic AI with Strands Agents SDK



Agentic loop (repeats as needed)

User

Strands Agents SDK

LLM

Tools

Orders API

Inventory API

Payments API

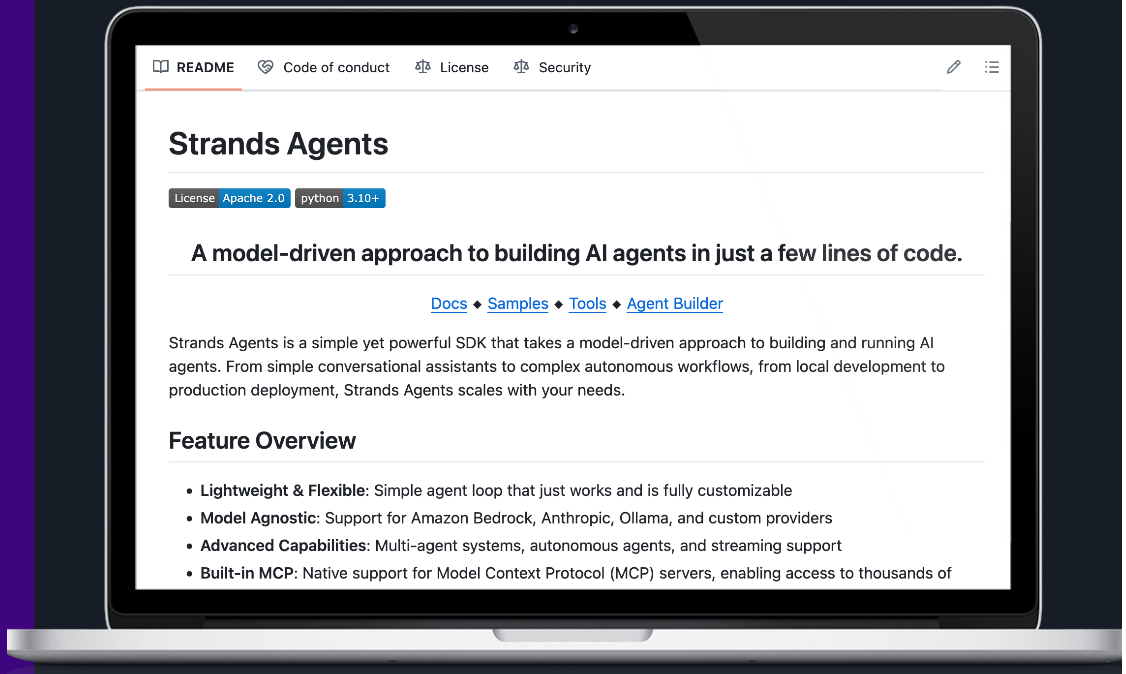
Support API

Developing agents

< Hello, World! >

Strands Agents

Strands Agents is an open-source SDK for building agents using just a few lines of code



Strands Agents

License Apache 2.0 python 3.10+

A model-driven approach to building AI agents in just a few lines of code.

Docs ◆ Samples ◆ Tools ◆ Agent Builder

Strands Agents is a simple yet powerful SDK that takes a model-driven approach to building and running AI agents. From simple conversational assistants to complex autonomous workflows, from local development to production deployment, Strands Agents scales with your needs.

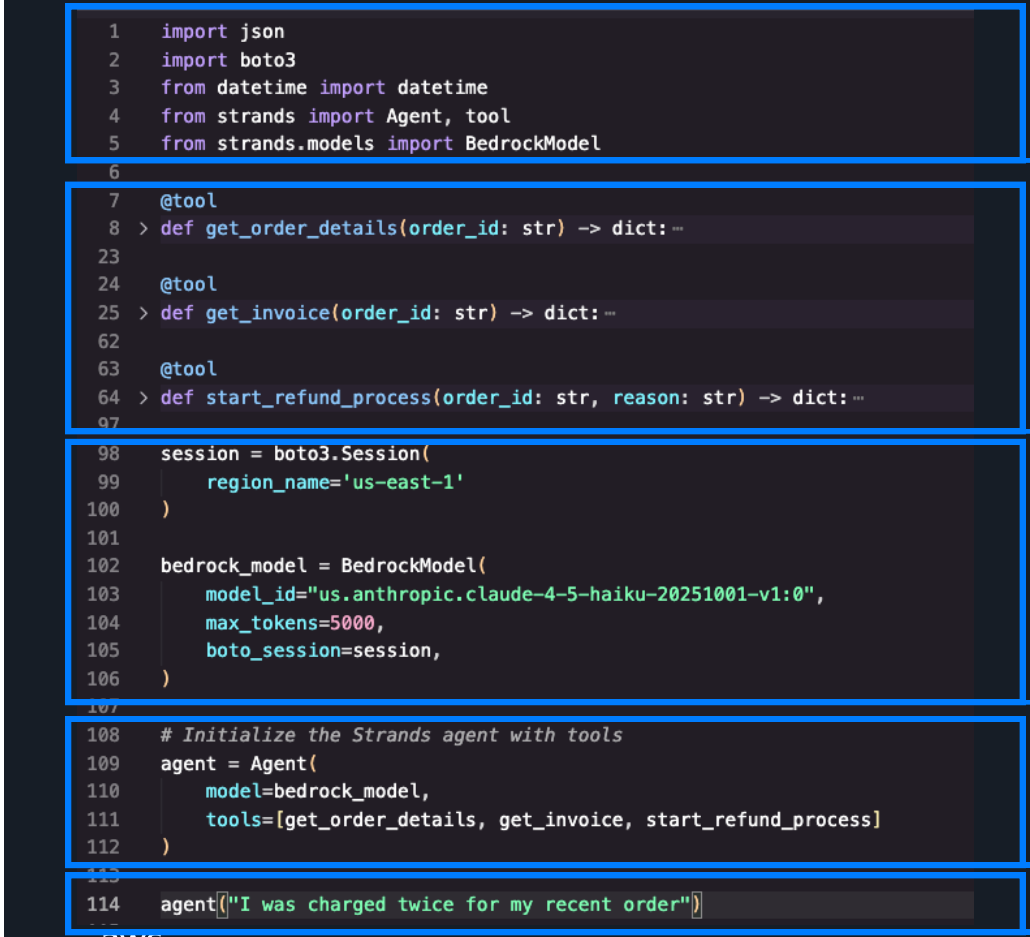
Feature Overview

- **Lightweight & Flexible**: Simple agent loop that just works and is fully customizable
- **Model Agnostic**: Support for Amazon Bedrock, Anthropic, Ollama, and custom providers
- **Advanced Capabilities**: Multi-agent systems, autonomous agents, and streaming support
- **Built-in MCP**: Native support for Model Context Protocol (MCP) servers, enabling access to thousands of

Simple Strands agent

```
from strands import Agent
from strands_tools import calculator

agent = Agent(tools=[calculator])
response = agent(”What is 80 / 4?")
```

Order support using Strands Agents SDK



```python
import json
import boto3
from datetime import datetime
from strands import Agent, tool
from strands.models import BedrockModel

@tool
def get_order_details(order_id: str) -> dict: ⋯

@tool
def get_invoice(order_id: str) -> dict: ⋯

@tool
def start_refund_process(order_id: str, reason: str) -> dict: ⋯

session = boto3.Session(
    region_name='us-east-1'
)

bedrock_model = BedrockModel(
    model_id="us.anthropic.claude-4-5-haiku-20251001-v1:0",
    max_tokens=5000,
    boto_session=session,
)

# Initialize the Strands agent with tools
agent = Agent(
    model=bedrock_model,
    tools=[get_order_details, get_invoice, start_refund_process]
)

agent("I was charged twice for my recent order")
```

Import from SDK

Define methods as tools

Use Bedrock as model provider

Create agent with model and tools

Invoke agent with user prompt

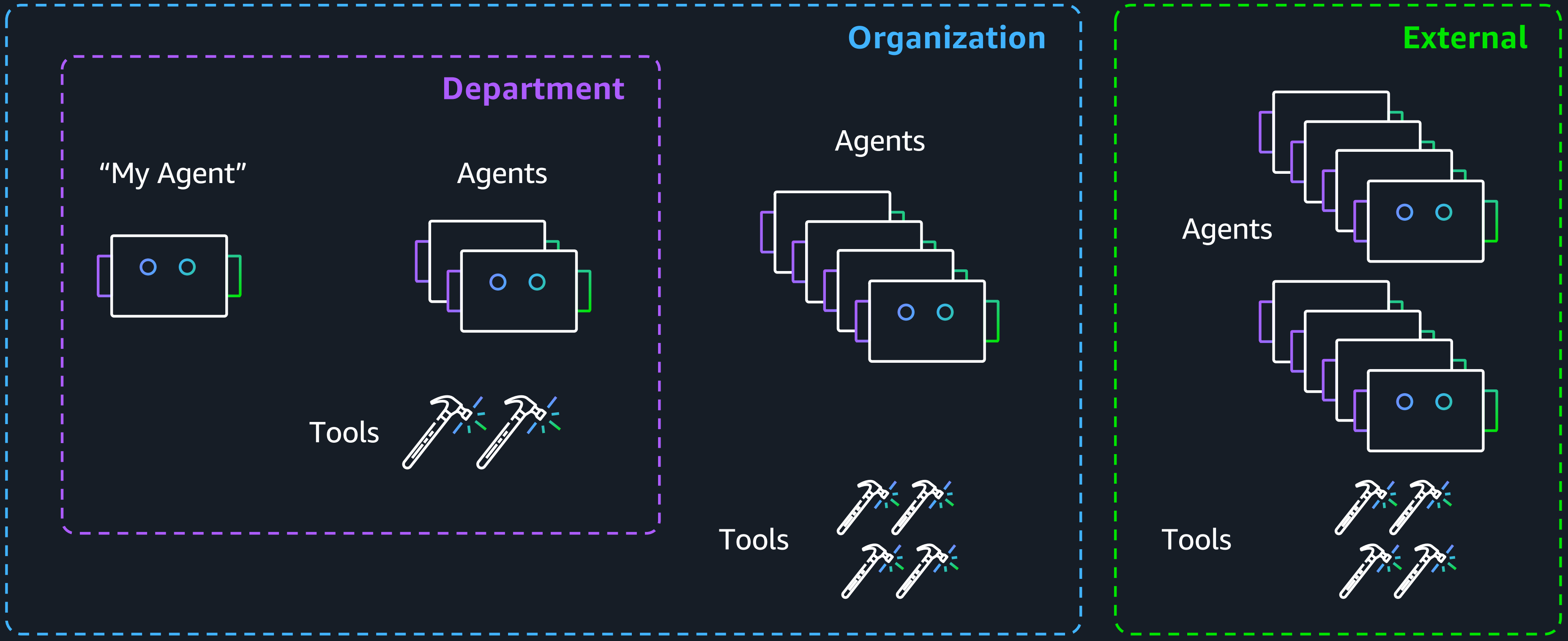
Benefits of using an agentic framework

In-built error handling,
retries, other best practices

Model and compute
agnostic

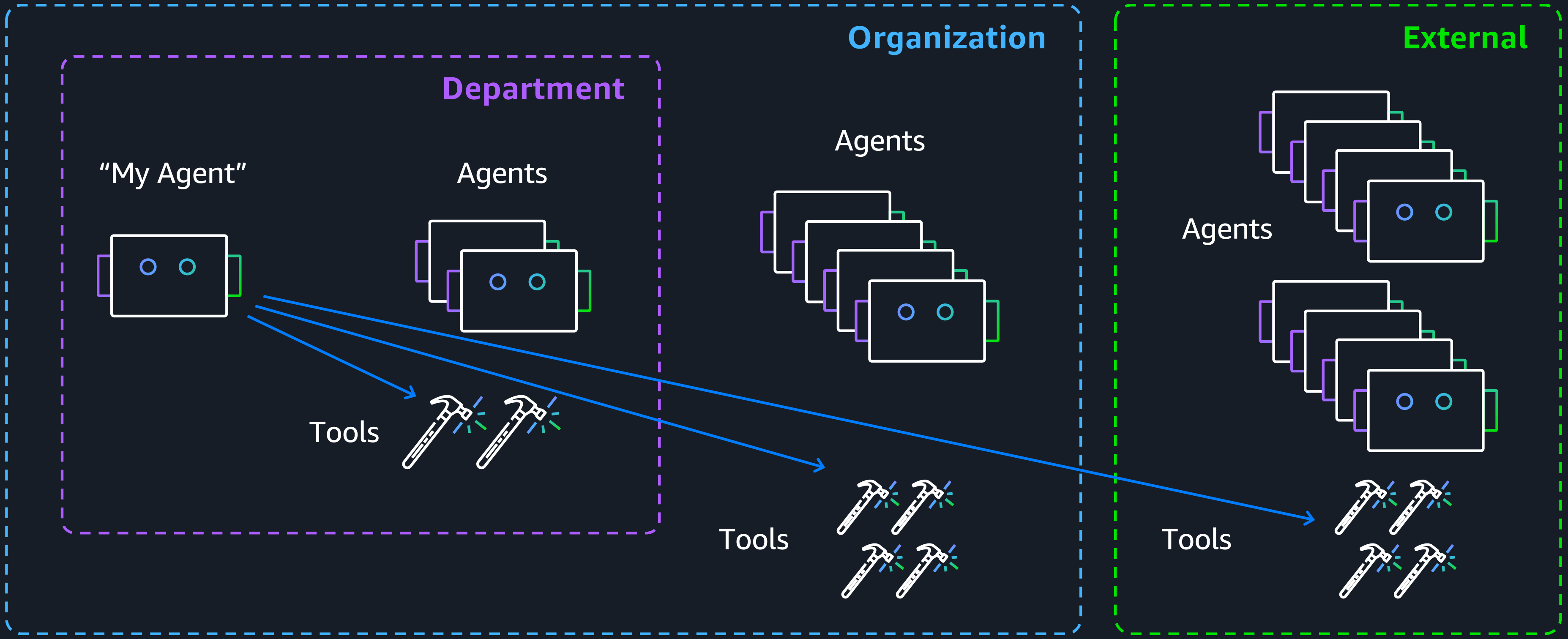
Supports dynamic pathways
(agentic loop)
& workflows

Building an agent is the start, production scale is the goal

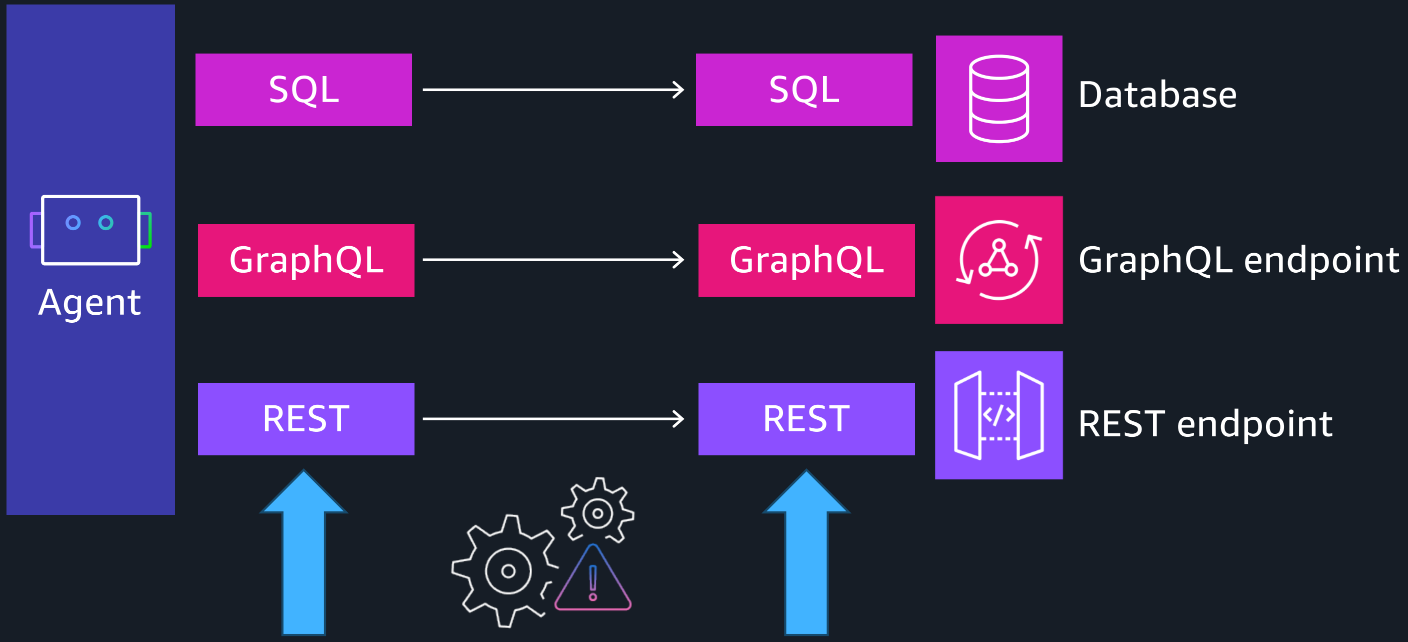


Scaling agents and tools with open standards

Agents invoking remote tools



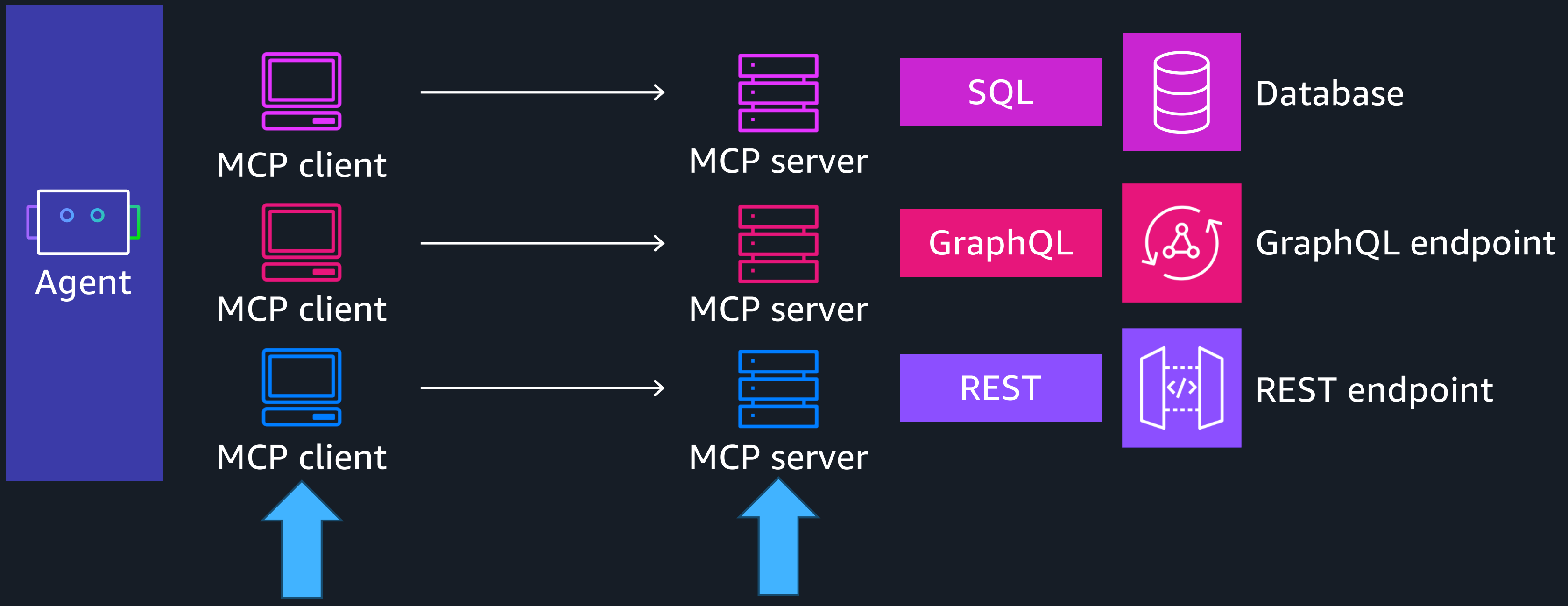
Agent-tool integrations without a standard



Agent builders are responsible to use different interfaces to integrate with tools

Restricted scope for tools vendors or cater to multiple interfaces.

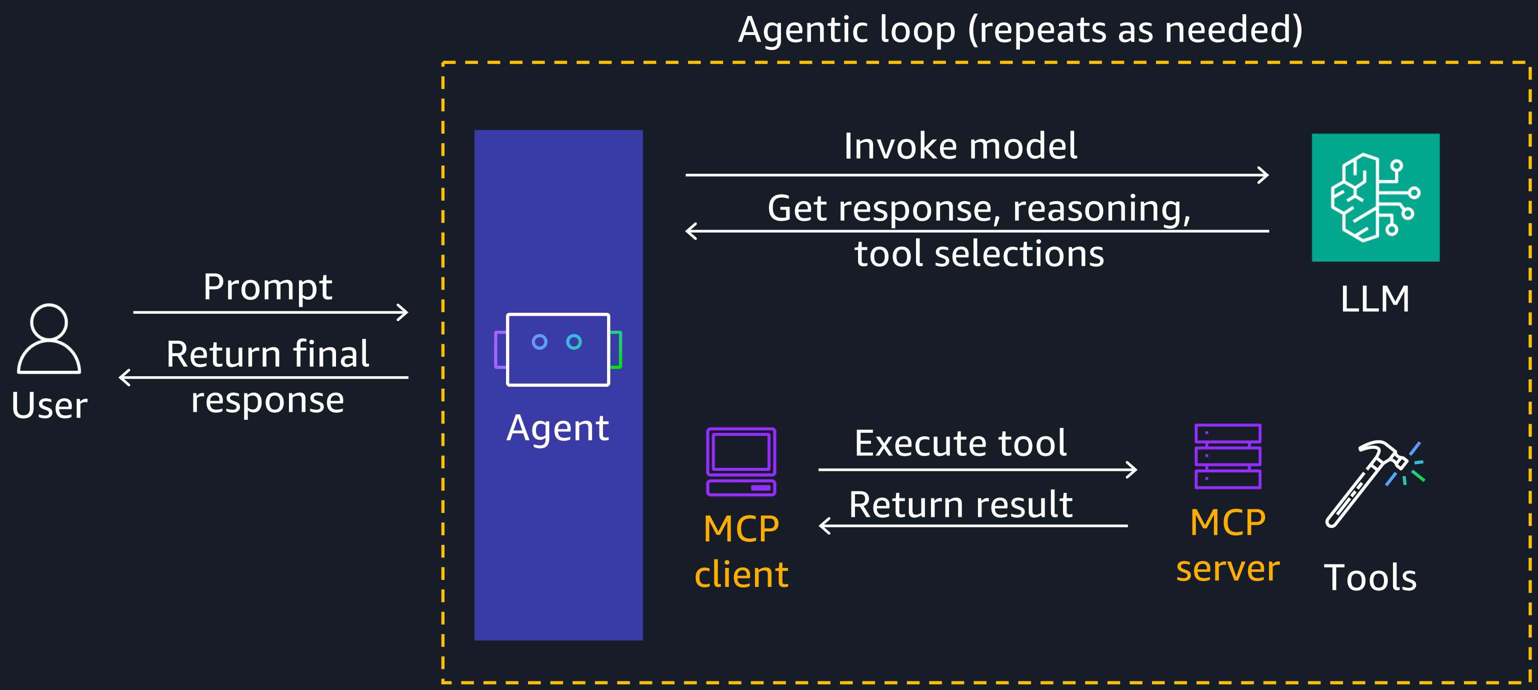
Agent integrations with Model Context Protocol (MCP)



Agent builders only need to support a single standard protocol

Tools vendors provide and maintain MCP-enabled capabilities

Standardizing tools integration with MCP



Agentic loop (repeats as needed)

User

Prompt

Return final response

Agent

Invoke model

Get response, reasoning, tool selections

LLM

MCP client

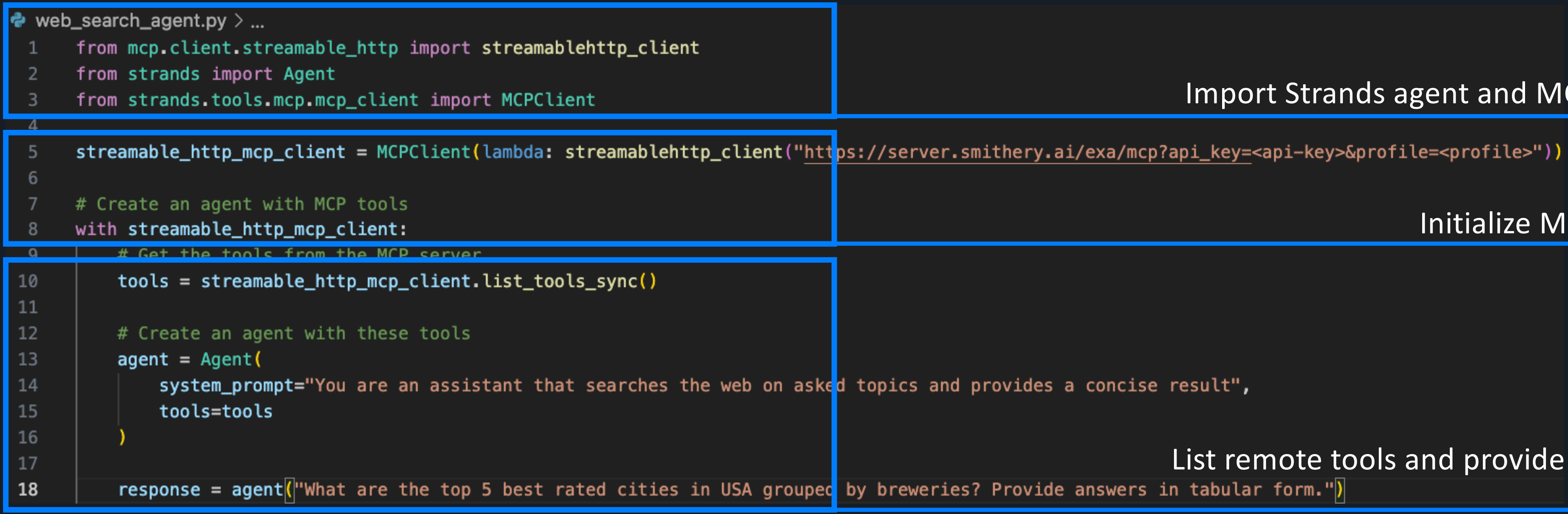
Execute tool

Return result

MCP server

Tools

Using MCP client in a Strands agent



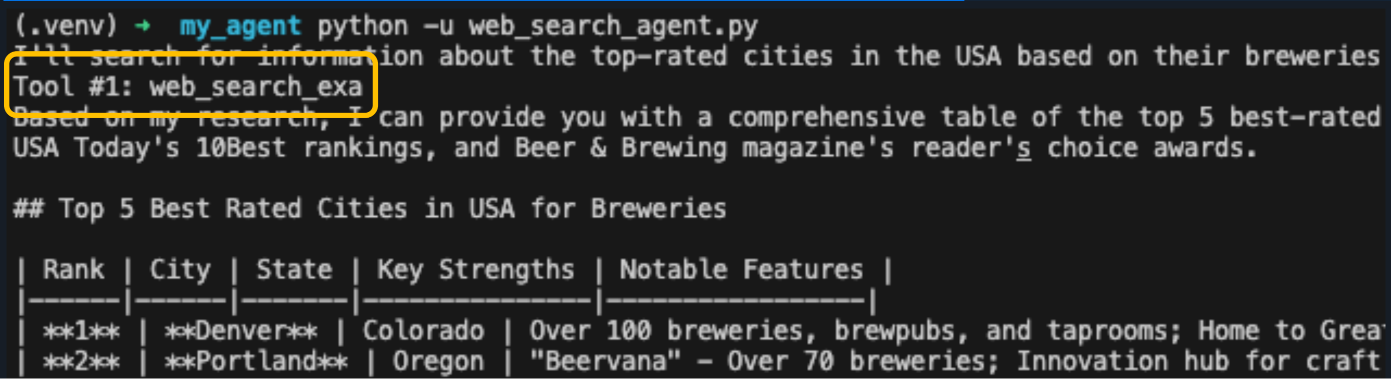
web_search_agent.py > ...

```python
from mcp.client.streamable_http import streamablehttp_client
from strands import Agent
from strands.tools.mcp.mcp_client import MCPClient

streamable_http_mcp_client = MCPClient(lambda: streamablehttp_client("https://server.smithery.ai/exa/mcp?api_key=<api-key>&profile=<profile>"))

# Create an agent with MCP tools
with streamable_http_mcp_client:
    # Get the tools from the MCP server
    tools = streamable_http_mcp_client.list_tools_sync()

    # Create an agent with these tools
    agent = Agent(
        system_prompt="You are an assistant that searches the web on asked topics and provides a concise result",
        tools=tools
    )

    response = agent("What are the top 5 best rated cities in USA grouped by breweries? Provide answers in tabular form.")
```

Import Strands agent and MCP Client

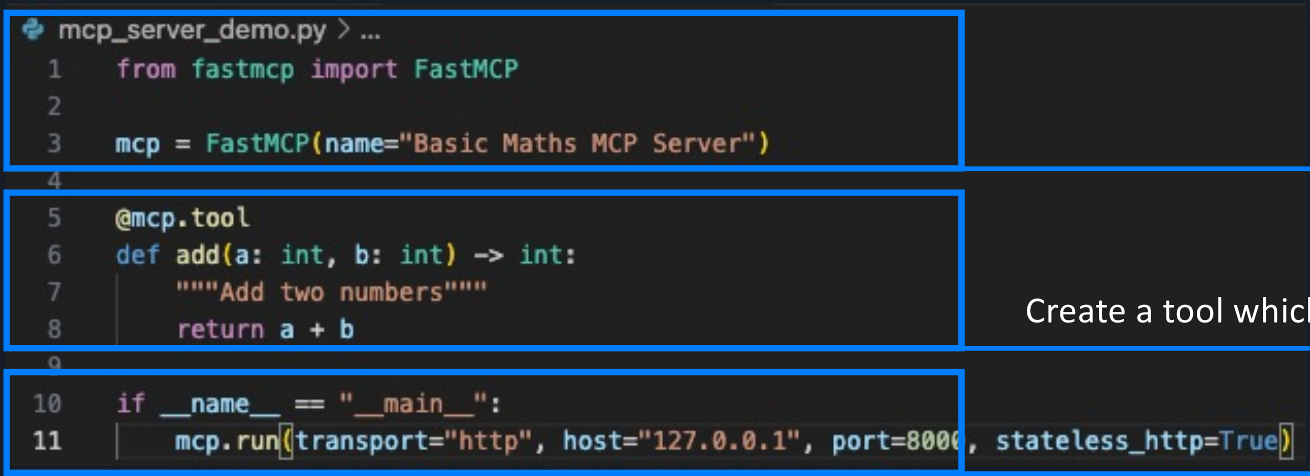
Initialize MCP client

List remote tools and provide to agent



```
(.venv) ➜  my_agent python -u web_search_agent.py
I'll search for information about the top-rated cities in the USA based on their breweries
Tool #1: web_search_exa
Based on my research, I can provide you with a comprehensive table of the top 5 best-rated
USA Today's 10Best rankings, and Beer & Brewing magazine's reader's choice awards.

## Top 5 Best Rated Cities in USA for Breweries

| Rank | City | State | Key Strengths | Notable Features |
|------|------|-------|---------------|------------------|
| **1** | **Denver** | Colorado | Over 100 breweries, brewpubs, and taprooms; Home to Grea
| **2** | **Portland** | Oregon | "Beervana" - Over 70 breweries; Innovation hub for craft
```

Building an MCP server

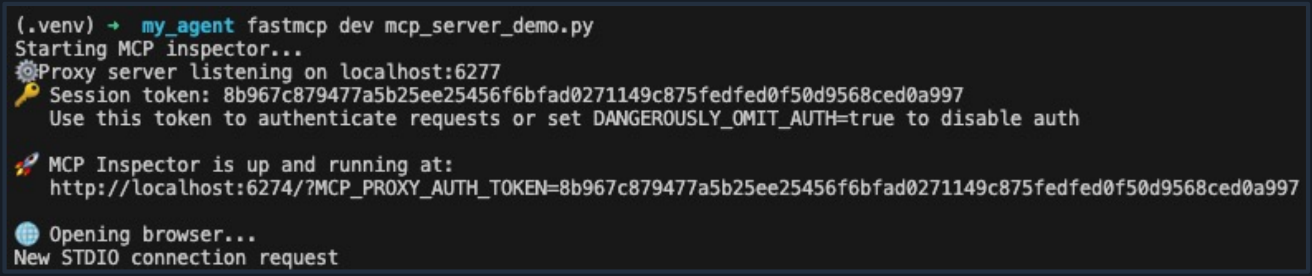


```
mcp_server_demo.py > ...
from fastmcp import FastMCP

mcp = FastMCP(name="Basic Maths MCP Server")

@mcp.tool
def add(a: int, b: int) -> int:
    """Add two numbers"""
    return a + b

if __name__ == "__main__":
    mcp.run(transport="http", host="127.0.0.1", port=8000, stateless_http=True)
```

Import and create FastMCP

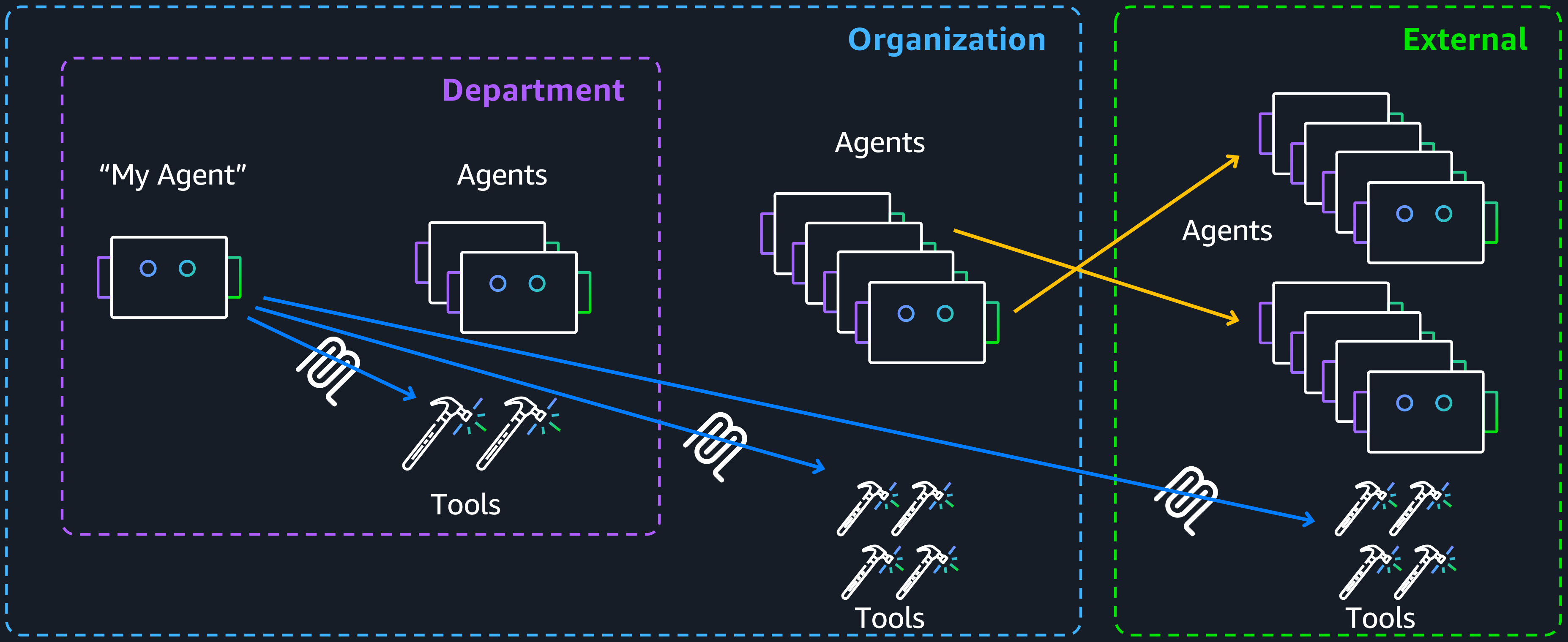
Create a tool which will be used by an MCP Client remotely

Run the server

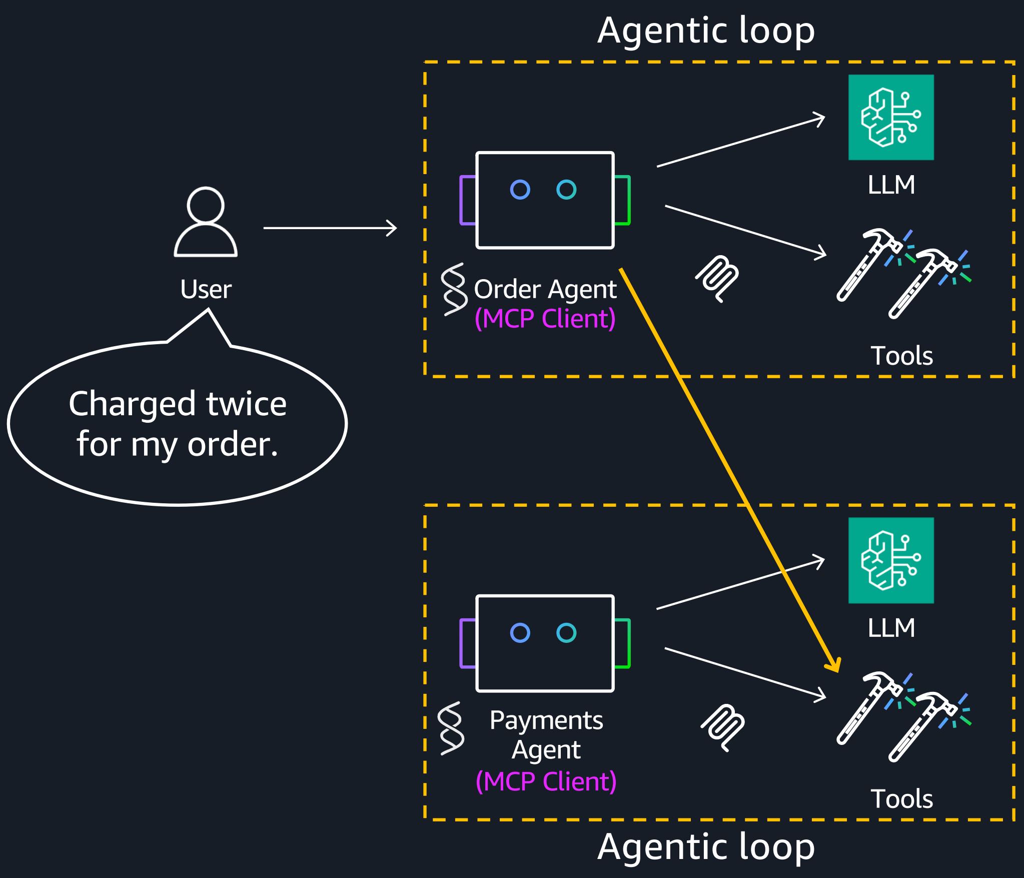


```
(.venv) → my_agent fastmcp dev mcp_server_demo.py
Starting MCP inspector...
⚙️Proxy server listening on localhost:6277
🔑 Session token: 8b967c879477a5b25ee25456f6bfad0271149c875fedfed0f50d9568ced0a997
   Use this token to authenticate requests or set DANGEROUSLY_OMIT_AUTH=true to disable auth

🚀 MCP Inspector is up and running at:
   http://localhost:6274/?MCP_PROXY_AUTH_TOKEN=8b967c879477a5b25ee25456f6bfad0271149c875fedfed0f50d9568ced0a997

🌐 Opening browser...
New STDIO connection request
```

Agents invoking remote tools **agents**



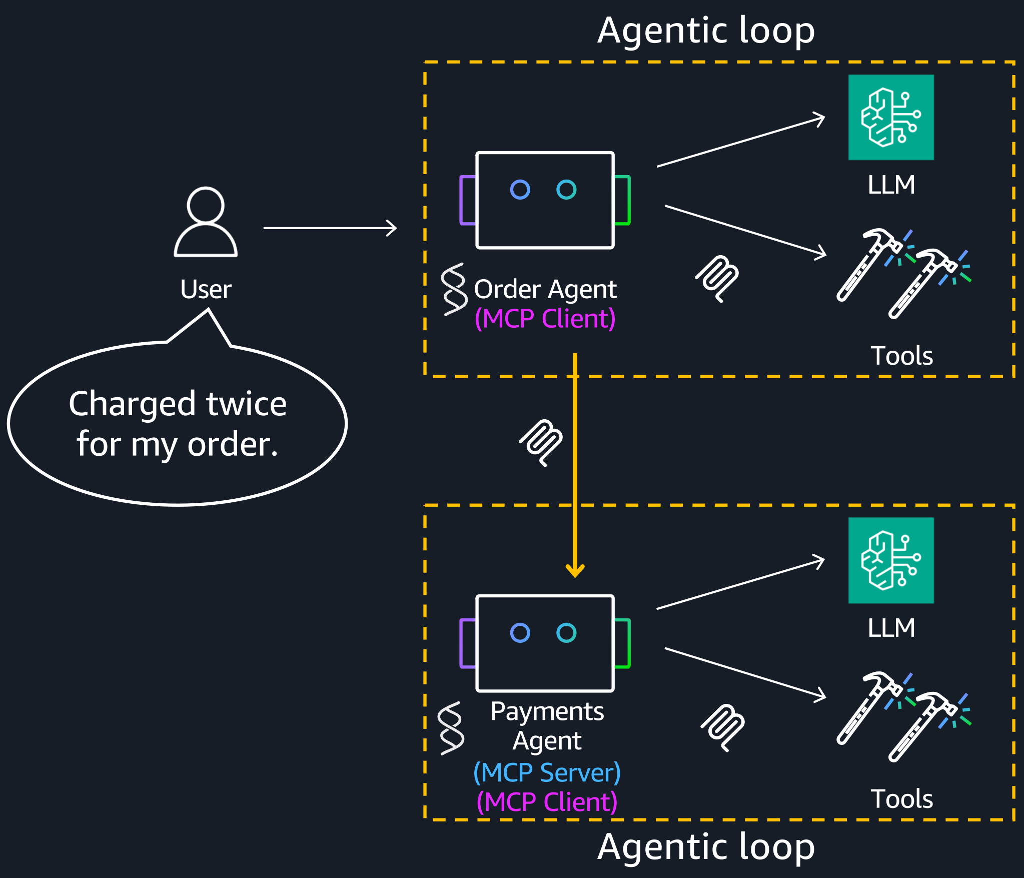
Inter-agent without a standard



Challenges:

- Scope creep
- Run tools from a different domain
- Ownership leakage
- Continuous revision

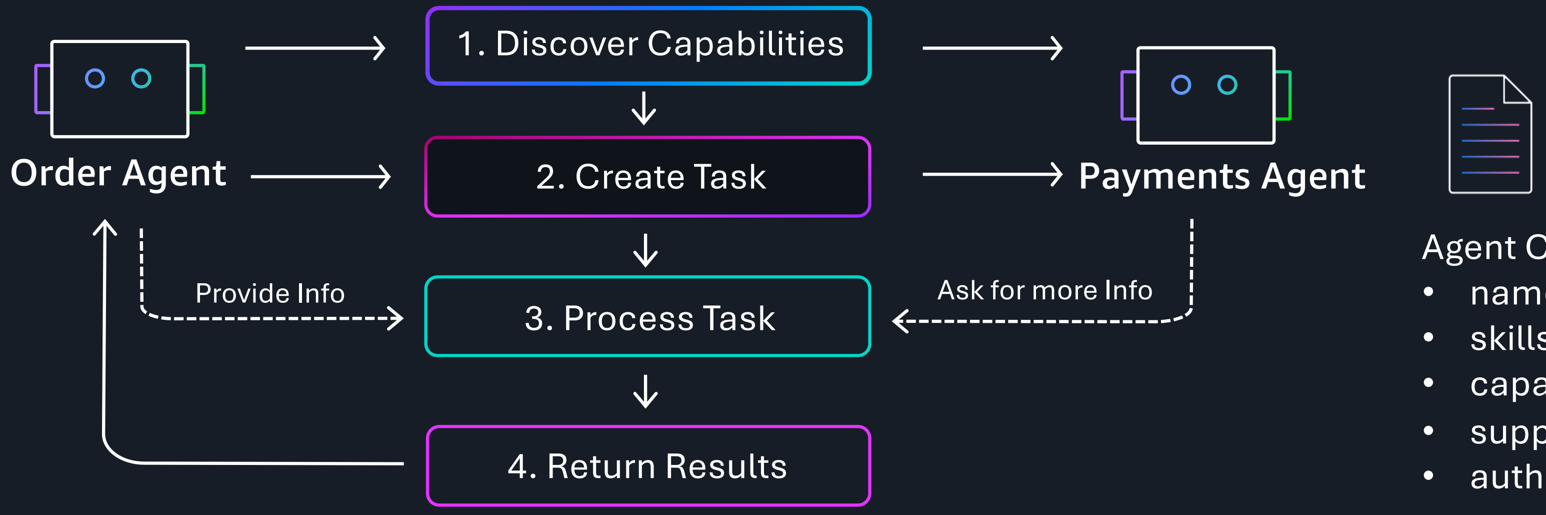
Inter-agent with MCP



- [Improvement] Better than no standards
- [Challenges]
  - Payments agent is used as a tool
  - Order agent must discover about Payments agent’s tools
  - Payments agent must act as an MCP Server & MCP Client

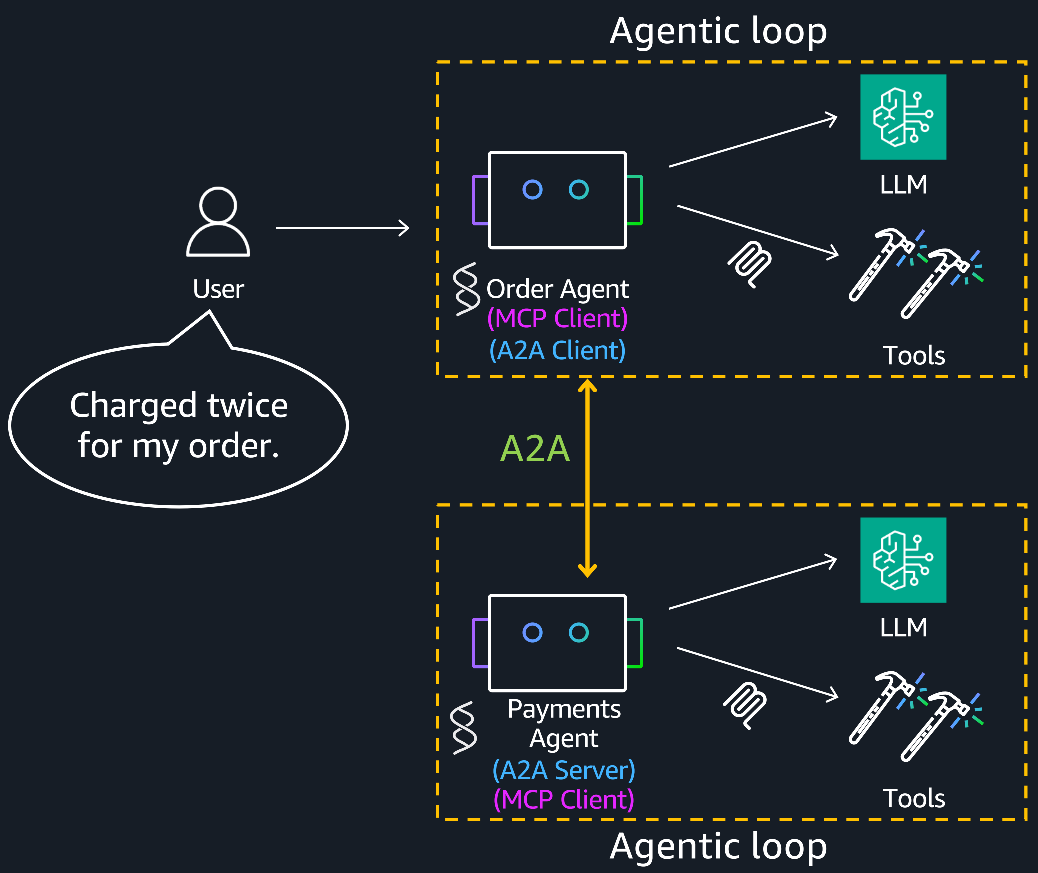
Introducing Agent-to-Agent (A2A)

A standardized protocol for AI agents to discover capabilities, exchange tasks, and collaborate on complex workflows



Agent Card
- name/version/desc.
- skills
- capabilities
- supported modalities
- auth requirements

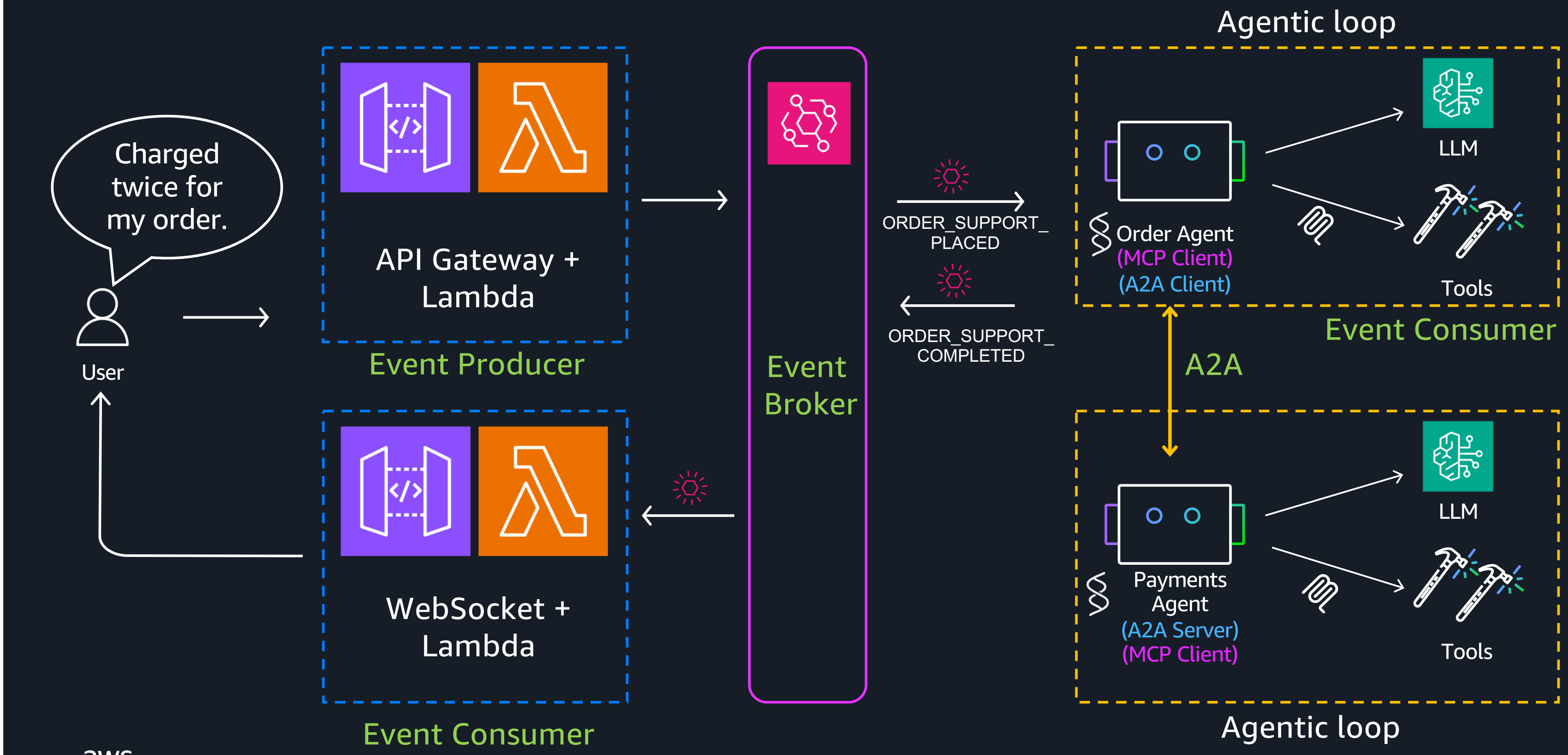
Inter-agent with A2A



Improvements:

- Encourages to build specialized agents
- Payments agent doesn’t leak domain knowledge
- Order agent discovers skills and capabilities of Payments agent from its agent card
- A2A supports sync and async communications

Event-driven with A2A and MCP



Benefits of open standards and EDA

Loose coupling with intelligent coordination

Context-aware event handling

Scalable agent orchestration

Asynchronous intelligence

Resilient workflows that are observable and auditable

Dynamic agent subscription

Deploying agents and tools on Serverless compute

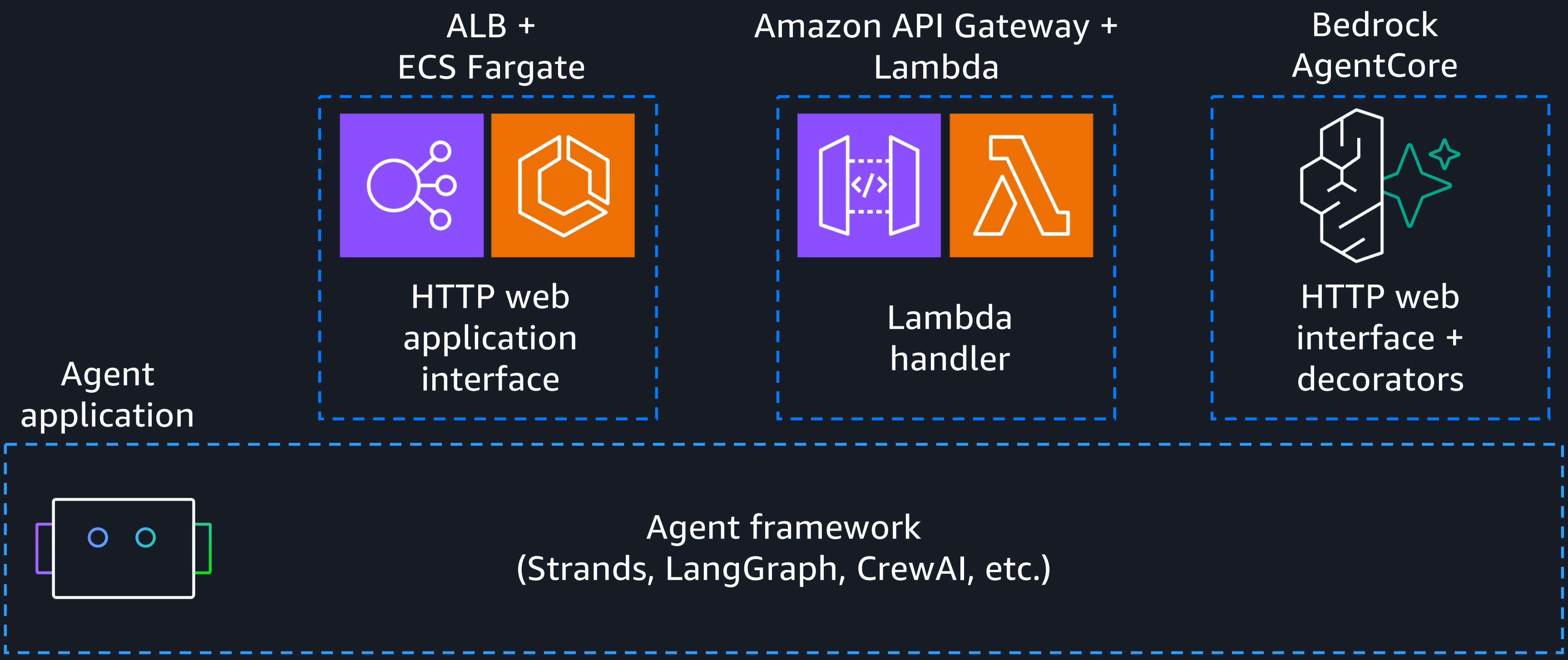
Deploying with AWS Serverless compute

AWS Lambda

Amazon Elastic Container Service (ECS)

Amazon Bedrock AgentCore

Agentic patterns for serverless compute



ALB +
ECS Fargate

HTTP web application interface

Amazon API Gateway +
Lambda

Lambda handler

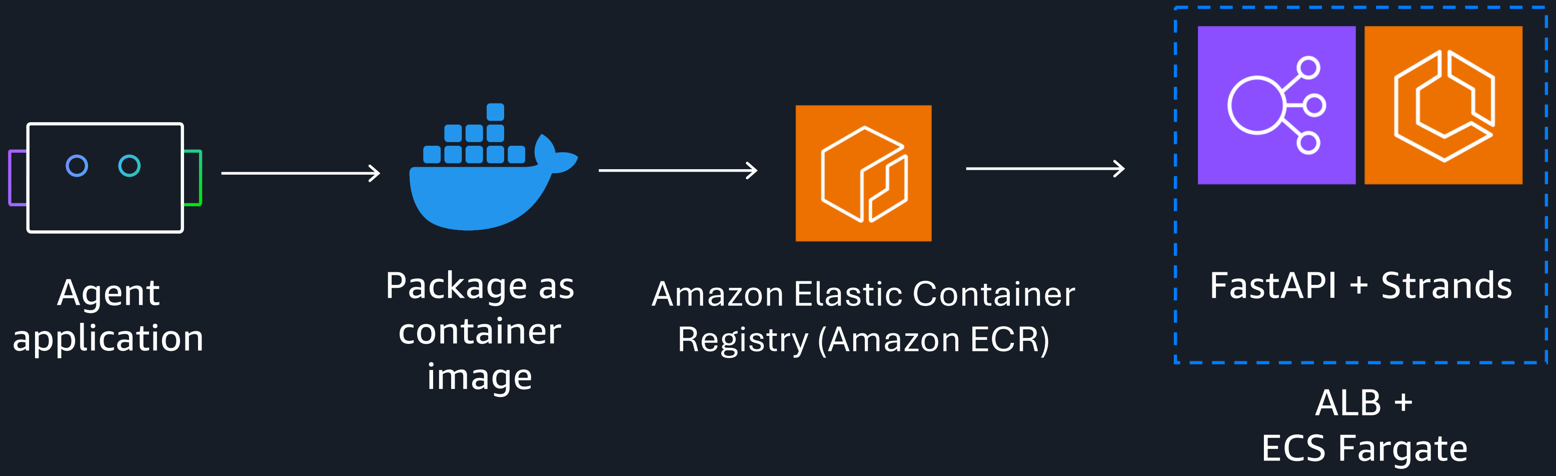
Bedrock
AgentCore

HTTP web interface + decorators

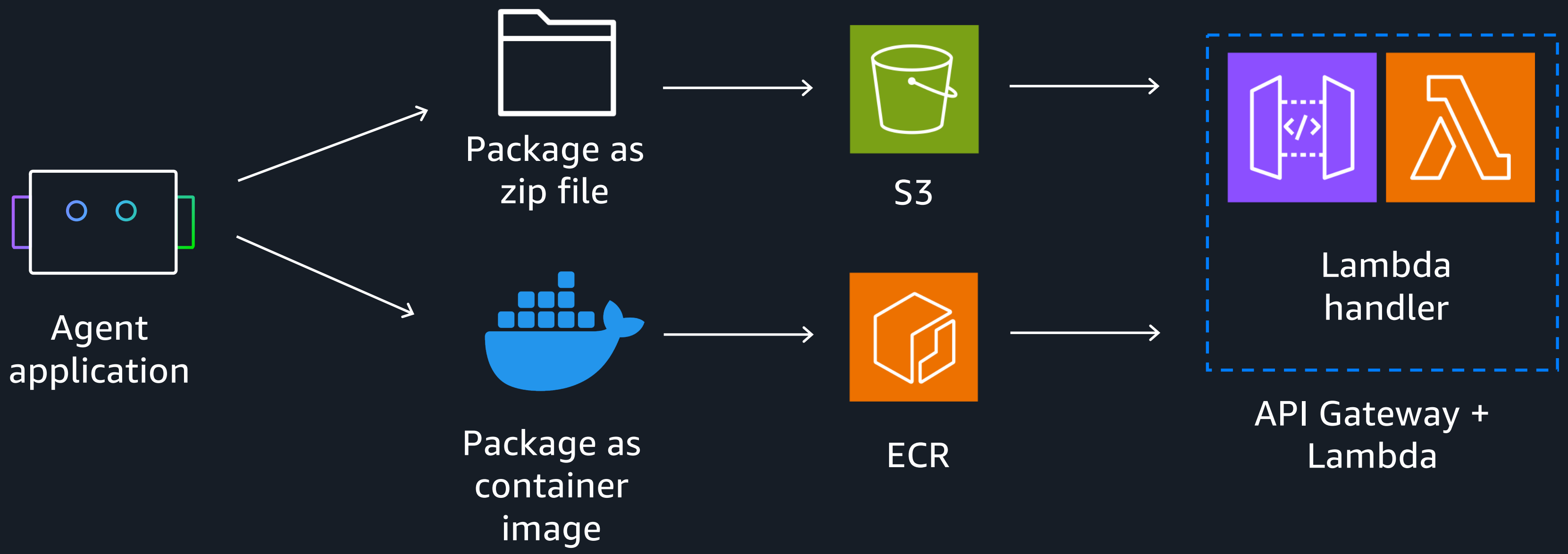
Agent application

Agent framework
(Strands, LangGraph, CrewAI, etc.)

Deploying agents to ECS Fargate


Agent application
Package as container image
Amazon Elastic Container Registry (Amazon ECR)
FastAPI + Strands
ALB +
ECS Fargate

Deploying agents to Lambda



Agent application

Package as zip file

Package as container image

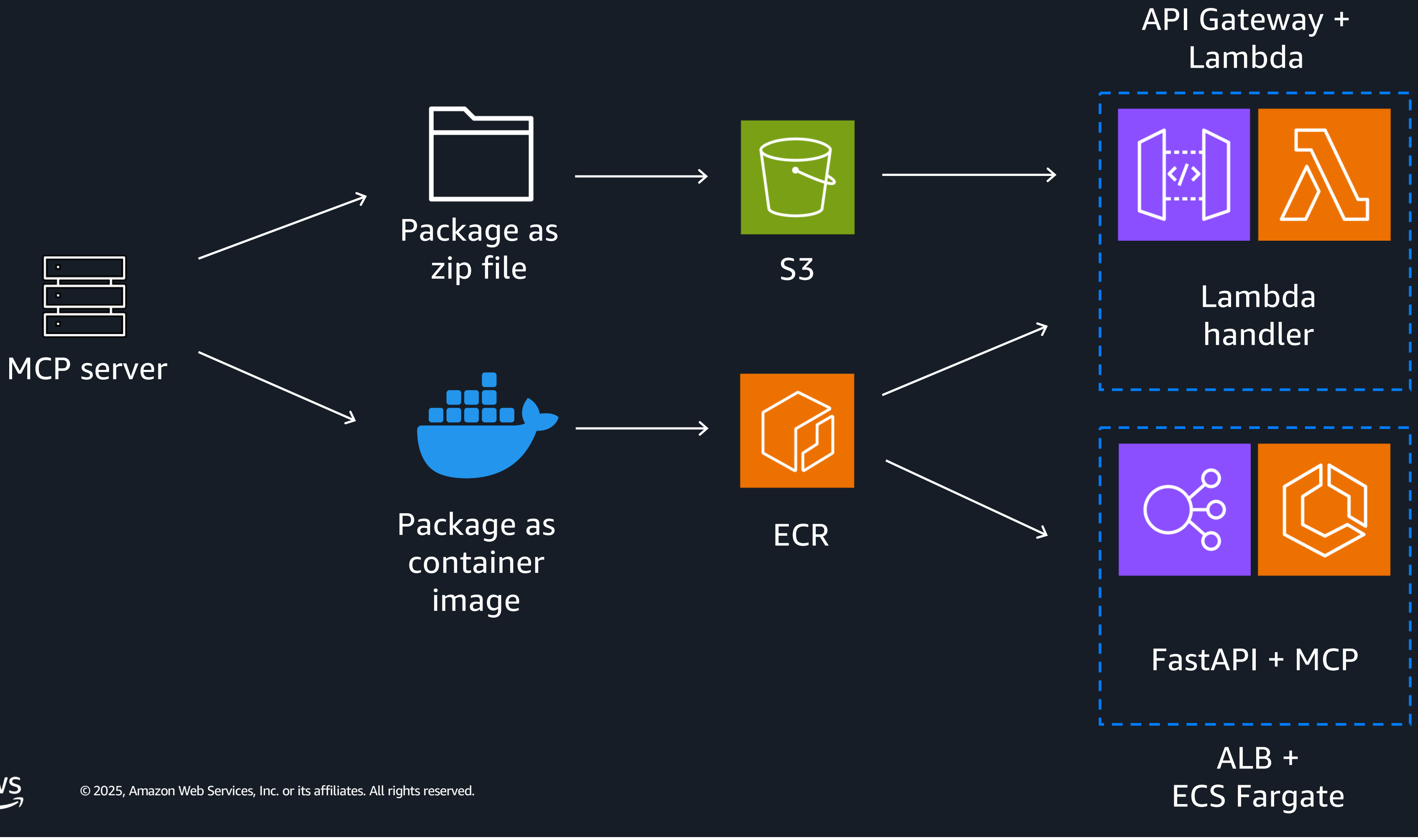
S3

ECR

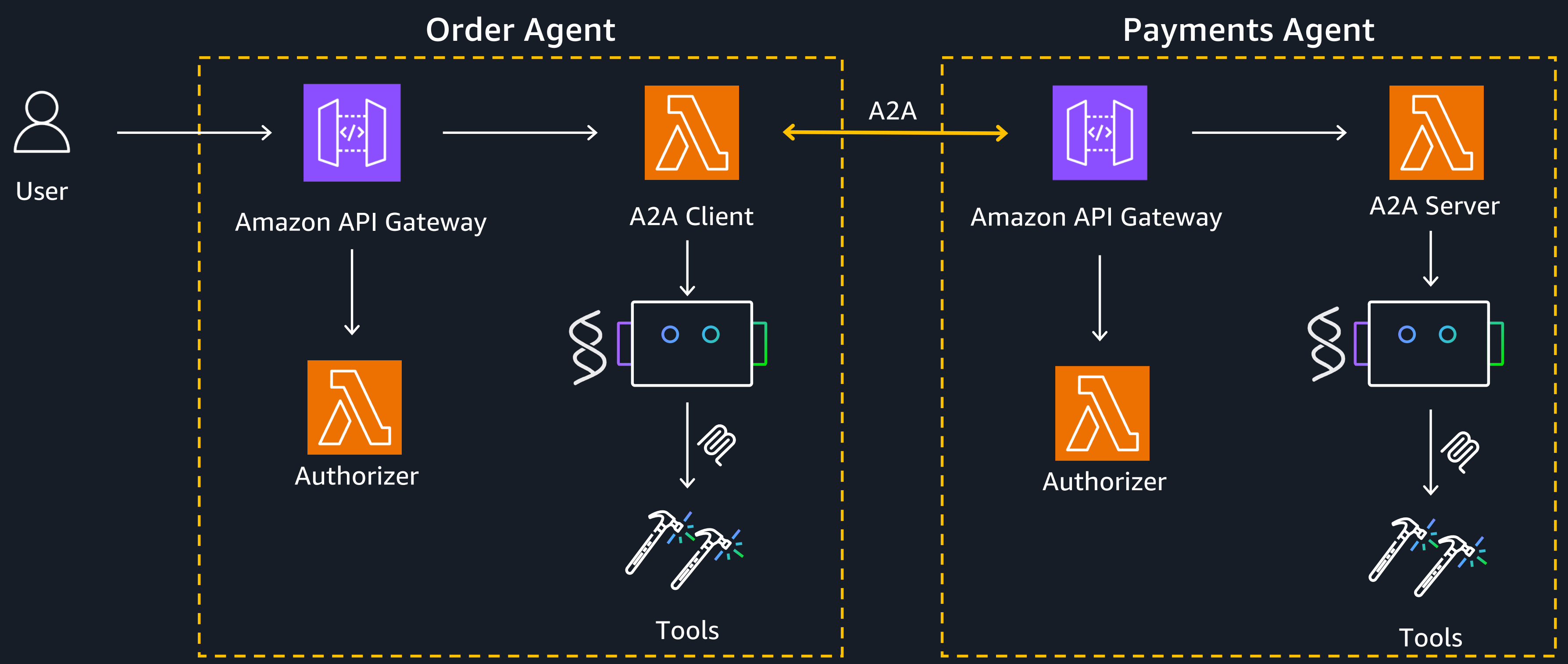
Lambda handler

API Gateway + Lambda

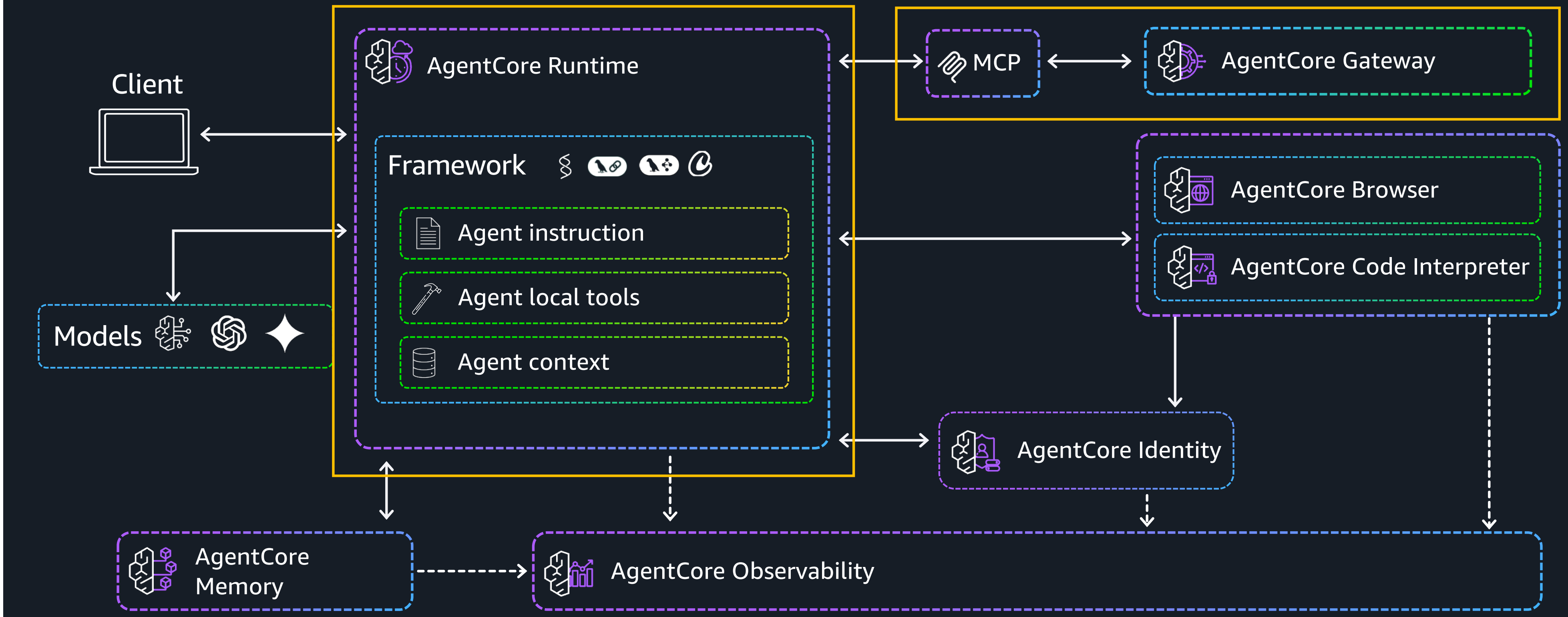
Deploying tools (MCP Servers)



A2A and MCP on Lambda functions



Considering serverless agents



Client

Models

AgentCore Runtime

Framework

Agent instruction

Agent local tools

Agent context

MCP

AgentCore Gateway

AgentCore Browser

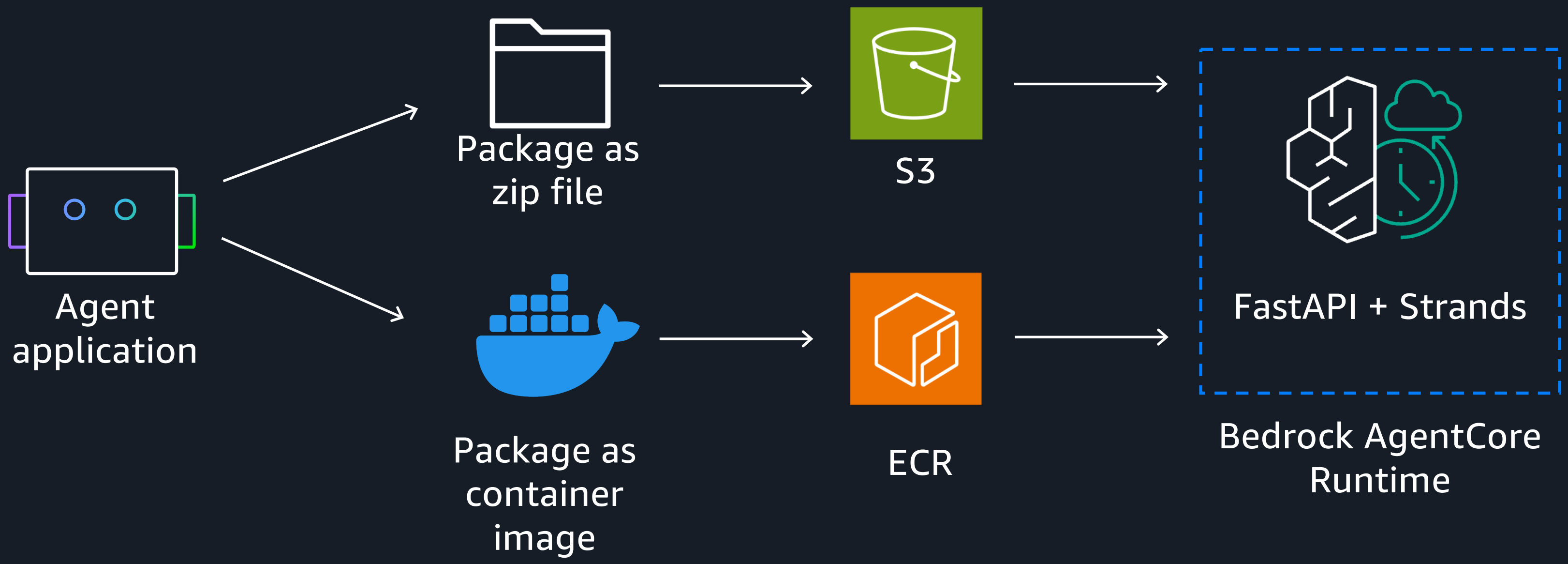
AgentCore Code Interpreter

AgentCore Identity

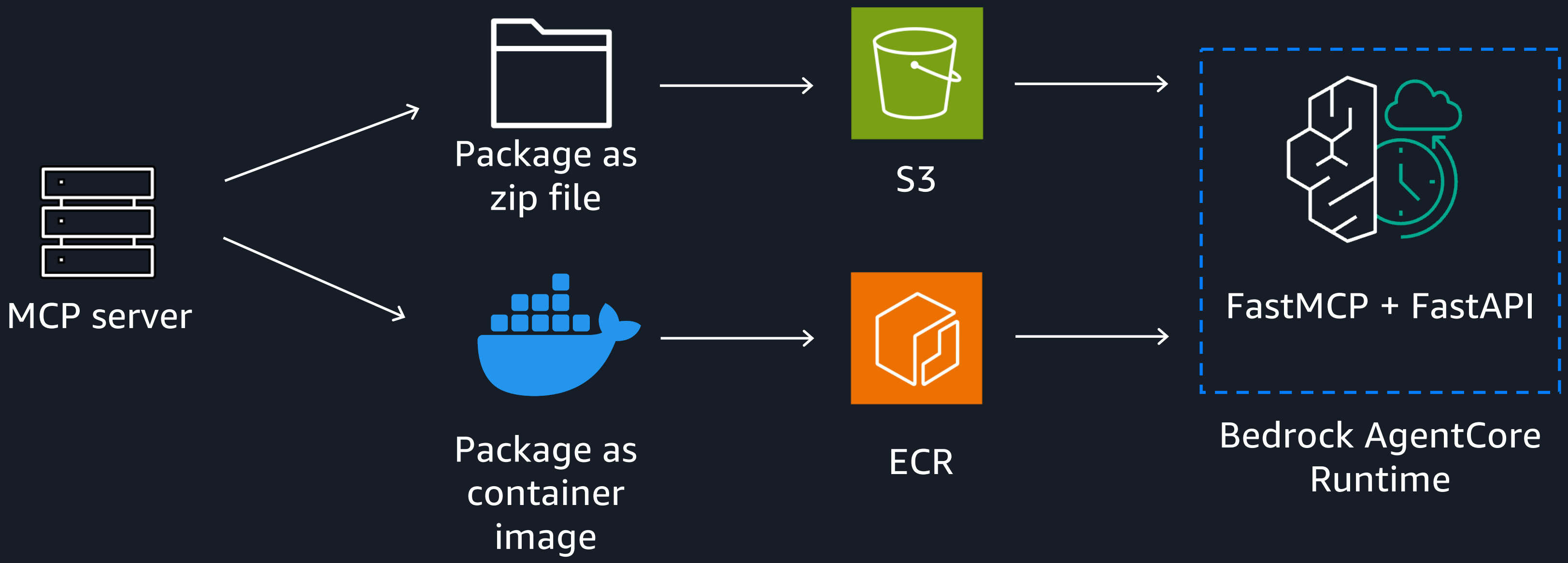
AgentCore Memory

AgentCore Observability

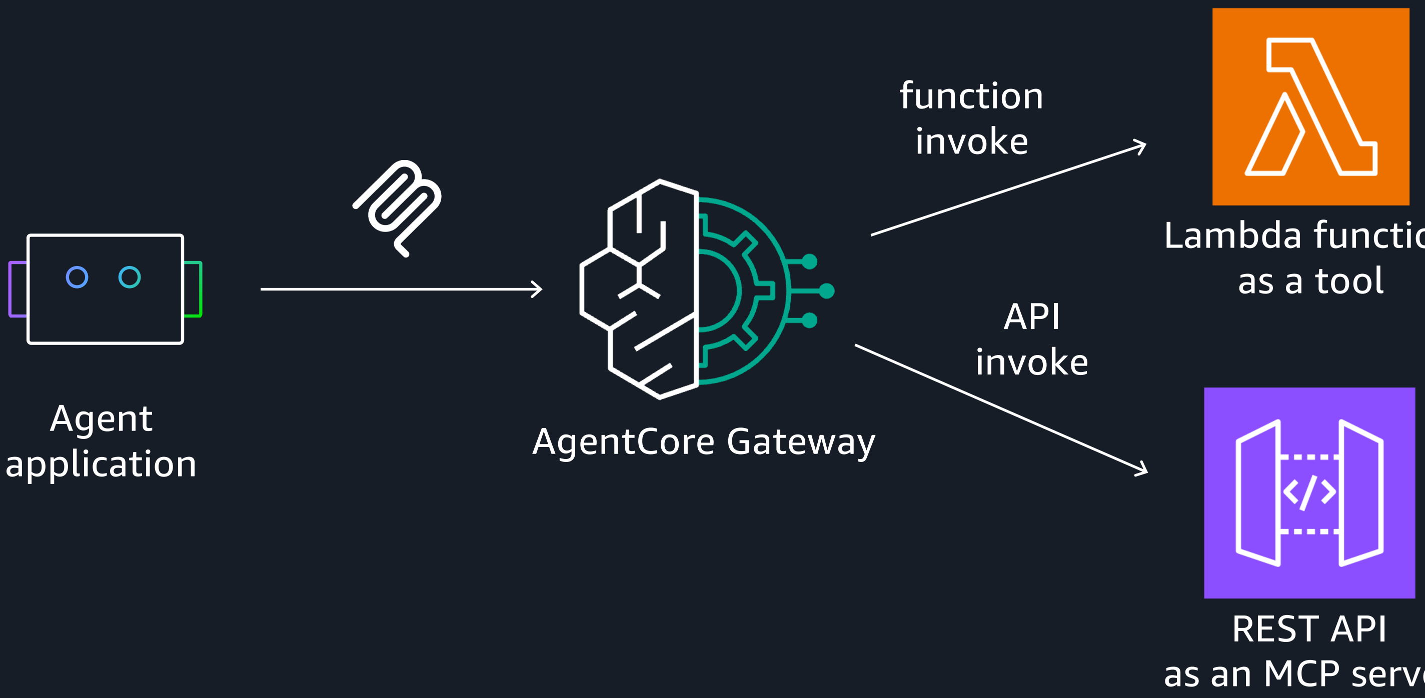
Deploying agents to AgentCore Runtime



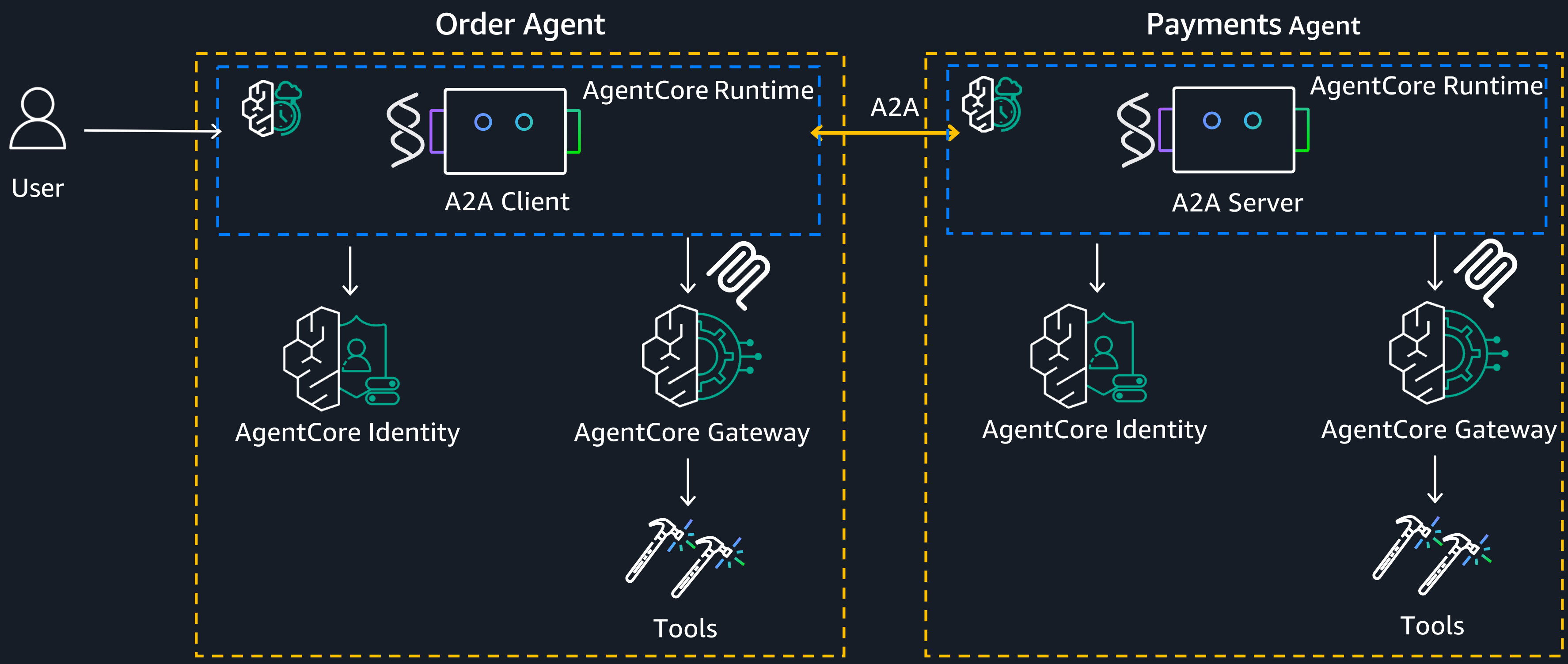
Deploying MCP servers to AgentCore Runtime



AgentCore Gateway with Lambda as tool

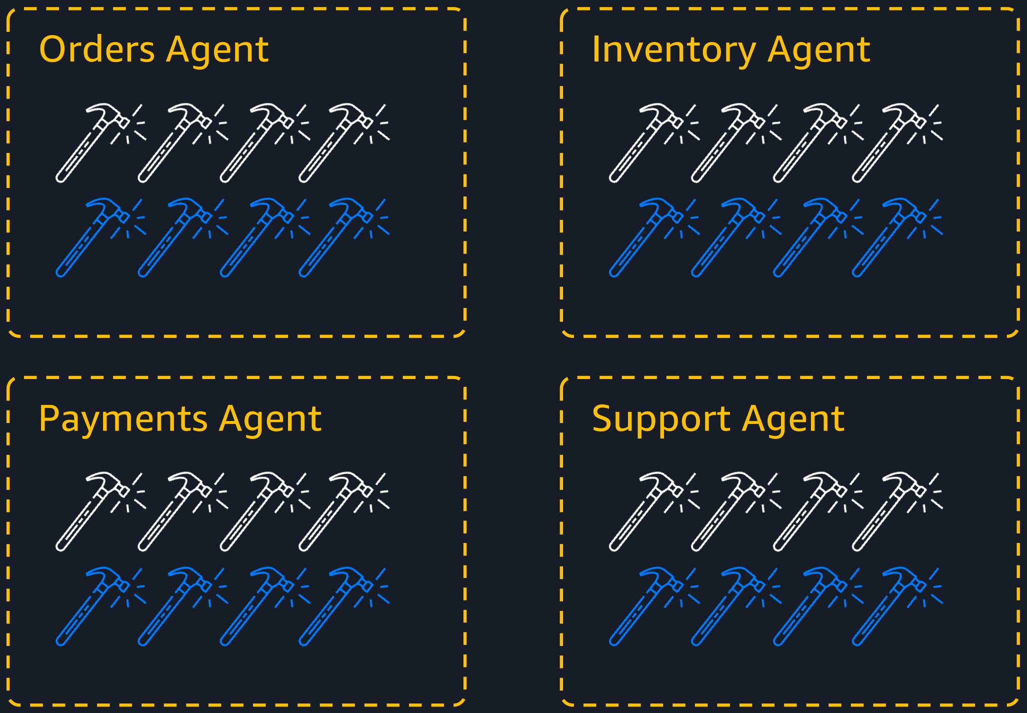


A2A and MCP on AgentCore

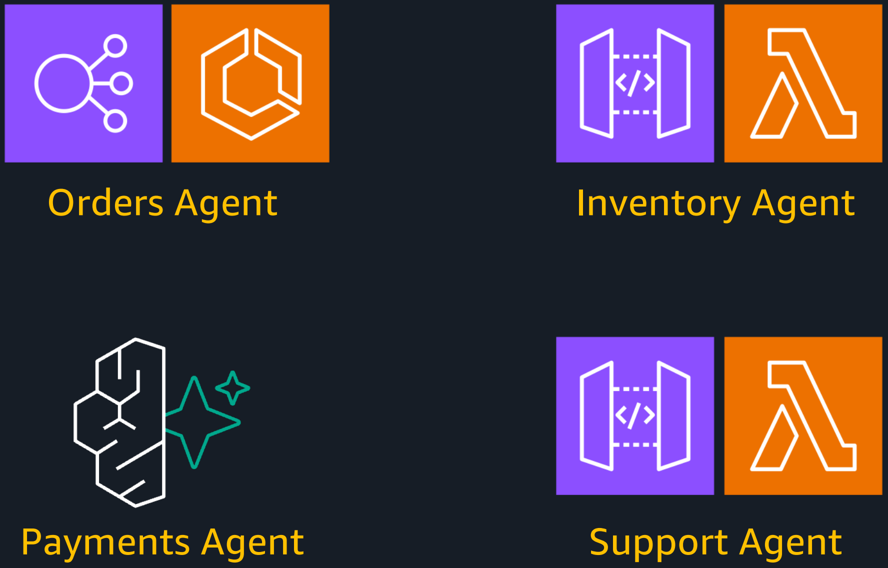


Transitioning to multi-agent systems

Decomposed agents



Decomposed agents



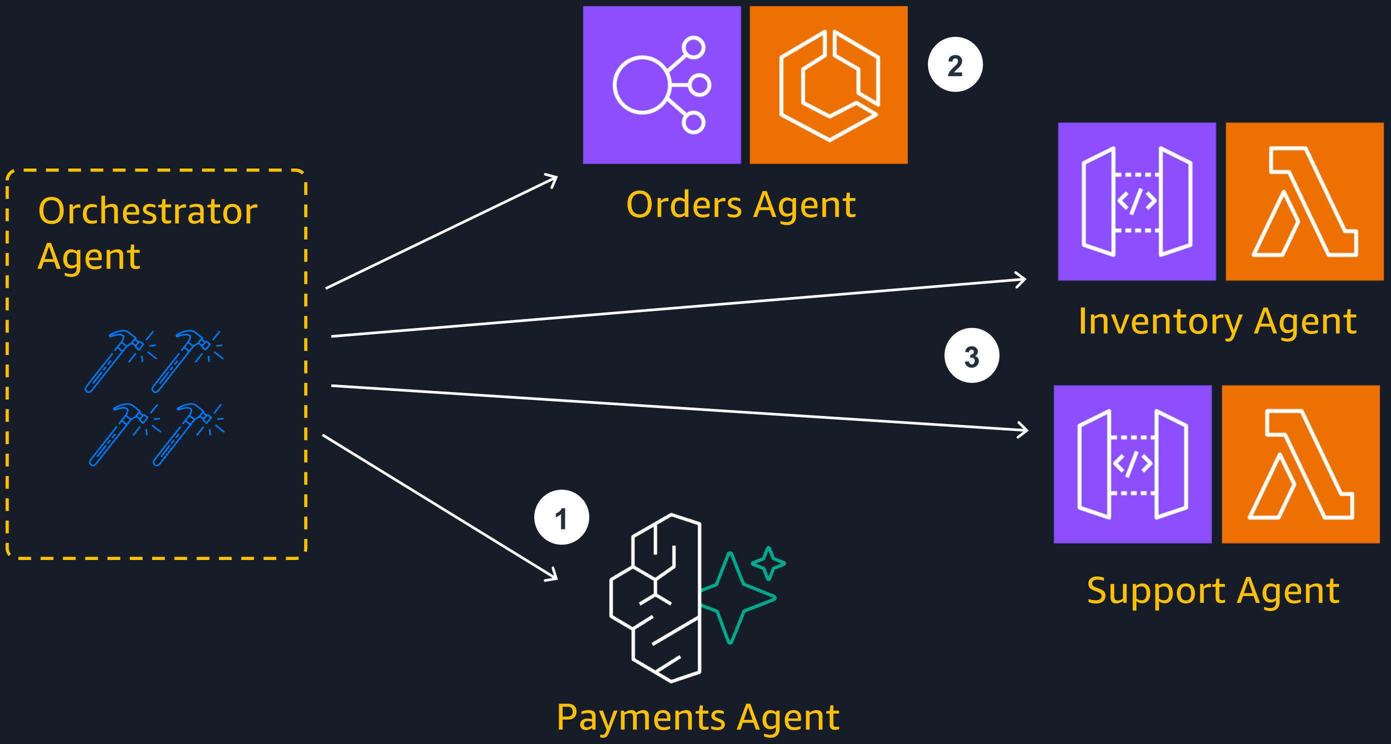
Orders Agent

Inventory Agent

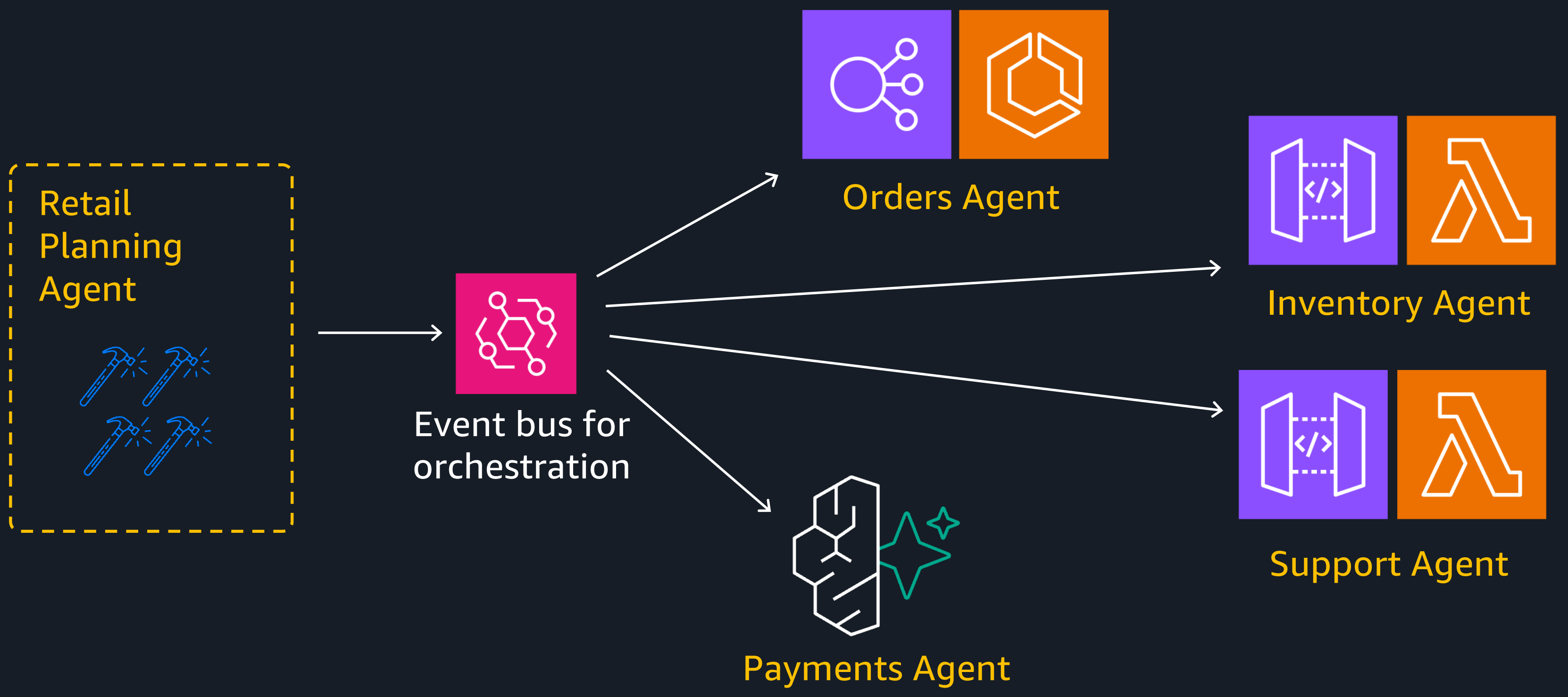
Payments Agent

Support Agent

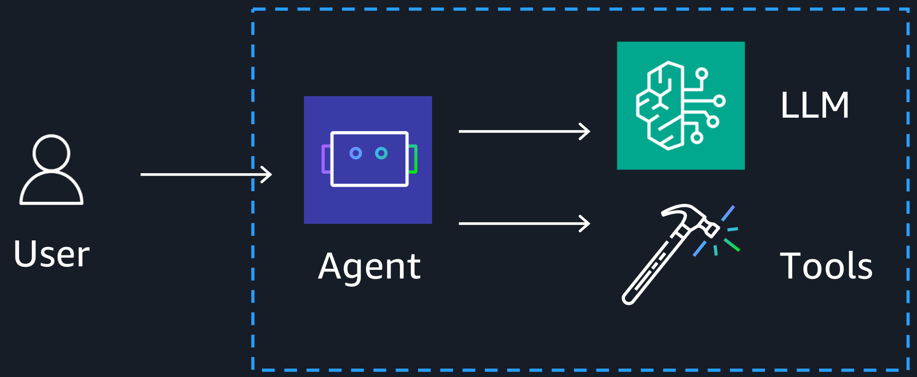
Orchestrating agent work



Scaling multi-agent systems



Starting with single-agent architecture



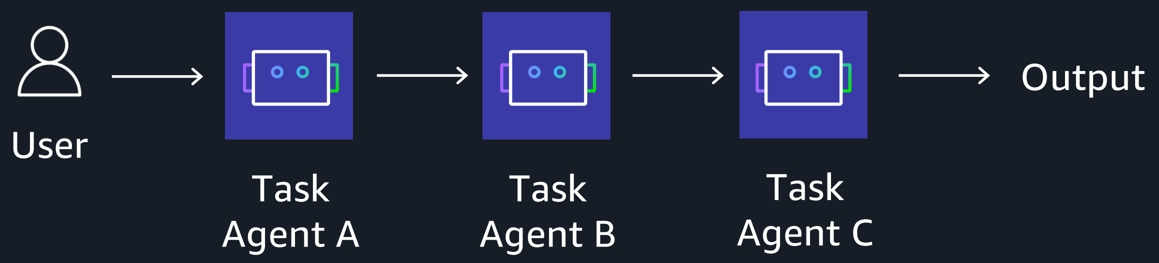
Pros:

- simpler to implement and debug
- easier to maintain and modify
- clearer reasoning traces
- *possibly* less computational overhead

Cons:

- limited by single context window
- can get stuck in reasoning loops
- could lack specialization, if required
- may struggle with complex, multi-step workflows

Building multi-agent architecture



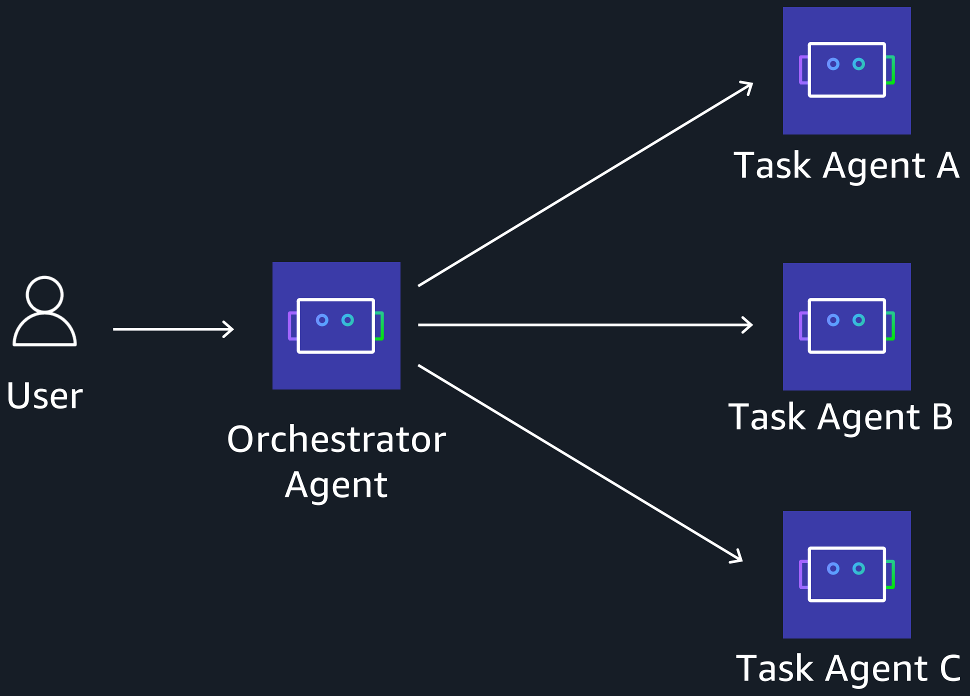
Pros:

- natural workflow modeling
- clear handoffs and error localization
- easy to optimize individual stages

Cons:

- limited to serial behavior, no benefits of parallelization
- serial dependencies
- errors amplified through the chain
- inflexible for non-linear processes

Building multi-agent architecture



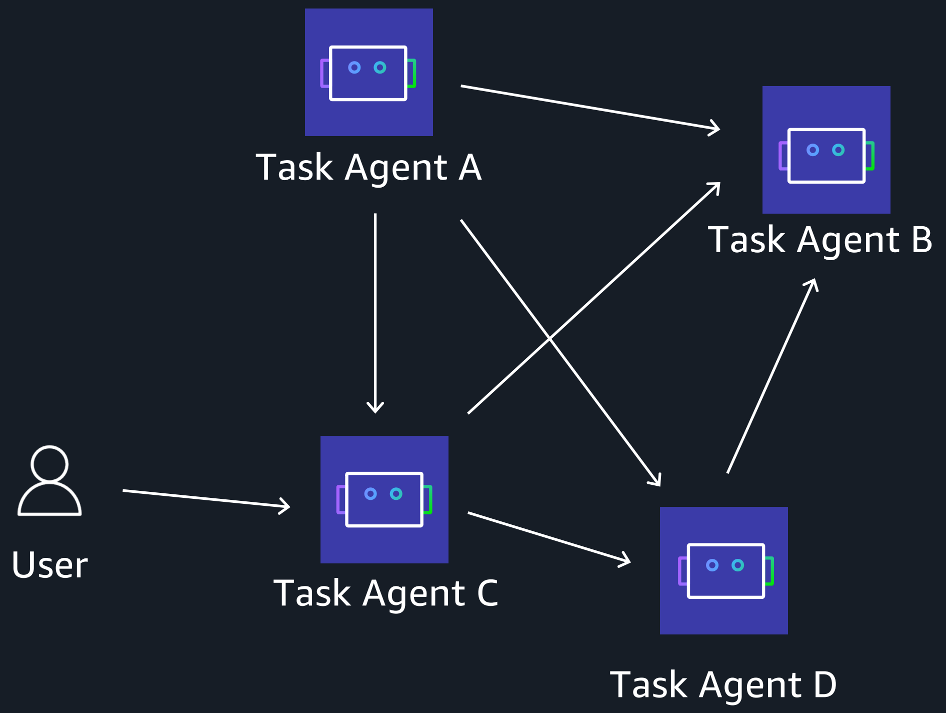
Pros:

- task decomposition, which allows for specialization and scalability
- composability allows for new agents
- maps more naturally to business process
- fault isolation

Cons:

- communication overhead
- orchestrator could become bottleneck
- error handling adds complexity
- could be rigid for dynamic problem-solving

Building multi-agent architecture



Pros:

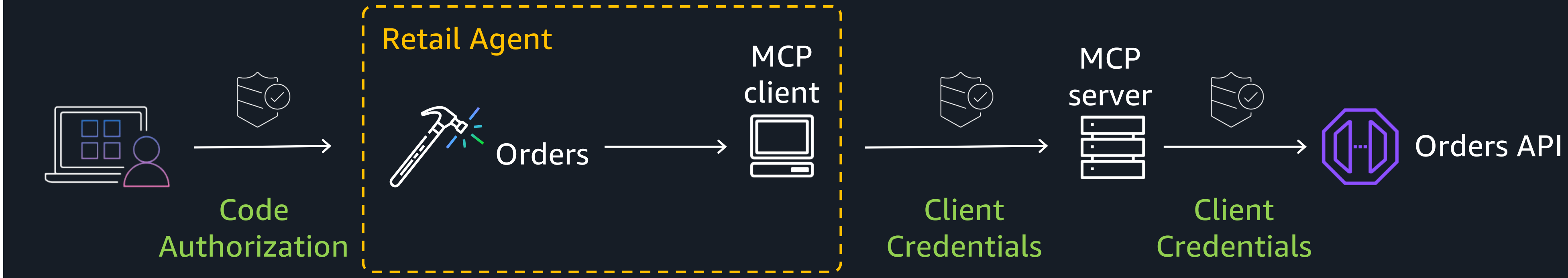
- highly adaptive, as agents self-organize based on task assignment
- has emergent problem-solving capabilities
- fault tolerant
- natural load balancing

Cons:

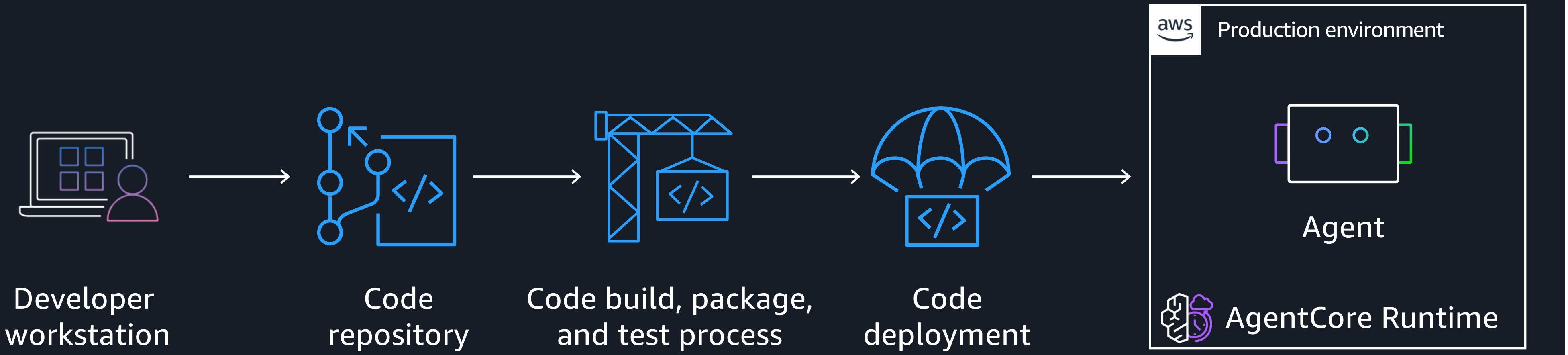
- difficult to predict behavior
- hard to debug and trace decisions
- complex coordination across agents
- requires agent discovery mechanisms

Baking in security and governance for deployments

Considering security



Considering governance



Developer workstation

Code repository

Code build, package, and test process

Code deployment

Production environment

Agent

AgentCore Runtime

Takeaways

The **same** design and architecture principles **continue to apply** with agents

Agents can be a way to **bridge data and glean insights**

Serverless can be a way to **accelerate the process** for building your agents

Serverless can help you **scale your agents** using standards and managed services

Key considerations

Be clear about your use cases. Fully understand where generative AI is appropriate.

Clearly document measurement metrics. Know what success looks like before you start.

Secure and instrument from the beginning. Security and observability cannot be after-thoughts.

Build and be scrappy. Lots to be uncovered during prototyping.

Document and provide feedback. Platform engineering needs to understand what they need to build and service for your developers.

Resources

https://s12d.com/riv25-cns359

Other sessions

[DEV415] Building Scalable, Self-Orchestrating AI Workflows with A2A and MCP
Monday, December 1 @ 1:30pm | MGM, Level 3

[CNS425] Building production-ready Agentic AI architectures with AWS serverless
Monday, December 1 @ 5:30pm | MGM | Level 3 | Chairman's 356
Tuesday, December 2 @ 1:00pm | Mandalay Bay | Level 2 South | Reef C

[CNS360] Implementing security best practices for serverless applications
Tuesday, December 2 @ 3pm | Mandalay Bay | Level 2 South, Oceanside C

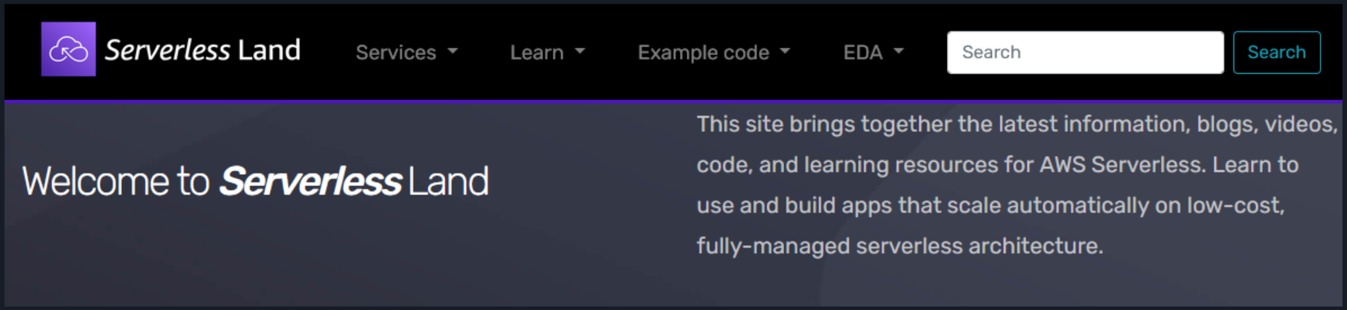
[CNS361] Scaling Serverless with platform engineering: A blueprint for success
Wednesday, December 3 @ 4:30pm | Mandalay Bay, Level 2 South, Oceanside C

[CNS428] Implementing Human-in-the-loop Controls for Multi-Agent AI Systems
Thursday, December 4 @ 2:30pm | Mandalay Bay, Lightning Theater

Additional resources

https://s12d.com/RIV2025

Continue your AWS serverless learning
Powertools for AWS Lambda
Serverless Land patterns



Welcome to *Serverless* Land

This site brings together the latest information, blogs, videos, code, and learning resources for AWS Serverless. Learn to use and build apps that scale automatically on low-cost, fully-managed serverless architecture.

Thank you!

Heeki Park
linktr.ee/heekipark

Dhiraj Mahapatro
@dmahapatro
linktr.ee/dmahapatro

Please complete the session survey in the mobile app