Stop AI Agent Hallucinations

5 Techniques + Production Patterns

Five techniques beyond the prompt. Each one is a code change, not a prompt change

Demos built with Strands Agents · Deployable on Amazon Bedrock AgentCore

Elizabeth Fuentes Leone

Developer Advocate @ AWS

elifuentes.tech

Resources

bit.ly/4oN2LVu

Five Techniques Beyond the Prompt

Each fixes a different failure. None of them is a prompt problem.

1

Semantic Tool Selection

Filter tools into context on every call

2

Graph-RAG

Compute precise answers, don't guess

3

Multi-Agent Validation

A second agent checks every response

4

Neurosymbolic Guardrails

Rules in code, not in the prompt

5

Runtime Steering

Self-correct instead of blocking

…then ship all five to production with Amazon Bedrock AgentCore

1 Semantic Tool Selection

First: what a tool actually looks like to the model

You write a @tool

```
@tool
def search_hotels(
    query: str) -> str:
    """Search for hotels by
    location name or city.
    Returns a list of
    available hotels."""
```

name · docstring · typed params

Strands generates a schema

```
{
  "name": "search_hotels",
  "description": "Search for
   hotels by location…",
  "inputSchema": {
    "query": {"type": "string"}
  }
}
```

the model reads this, not your code

…into context on every call

~70–100 tokens per schema
× 29 tools
= ~4,500 tokens

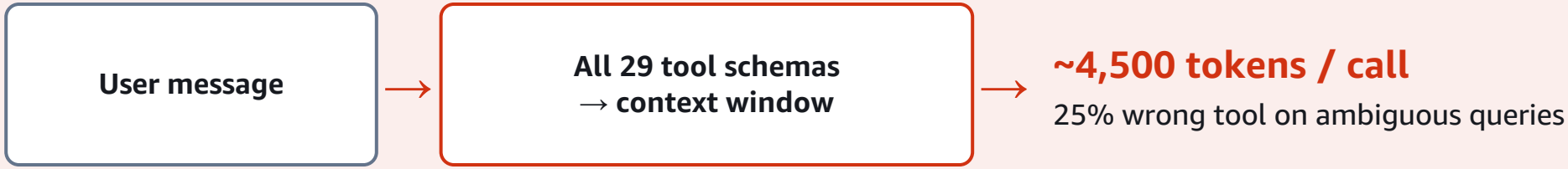
every call, before your message

If your agent has memory, that grows too: every turn adds context that ships with every message.

1 Semantic Tool Selection

Filter the tools before the agent sees them. The model only gets the top 3 for that query

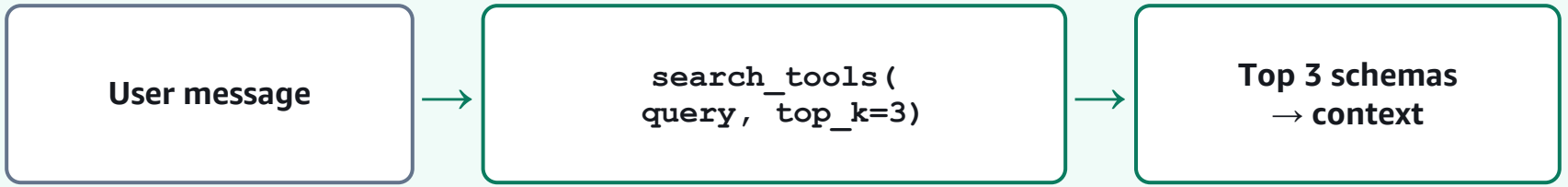
WITHOUT: traditional tool discovery



~4,500 tokens / call

25% wrong tool on ambiguous queries

WITH: semantic selection (FAISS over name + docstring)



~500 tokens / call

wrong-tool errors: 0%

swap_tools() re-registers tools on a live agent. Conversation memory stays intact

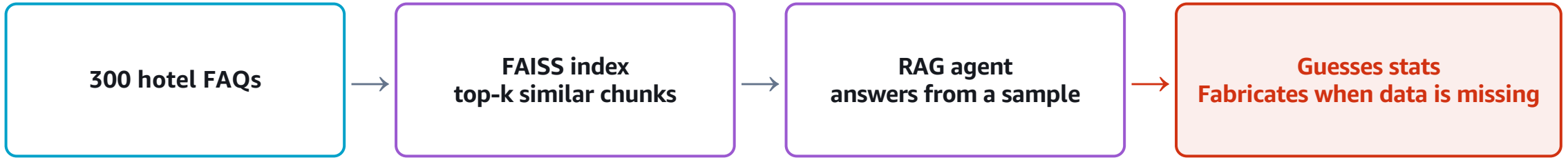
Demo Time

29 tools → top 3 · ~4,500 → ~500 tokens/call · wrong-tool errors → 0%

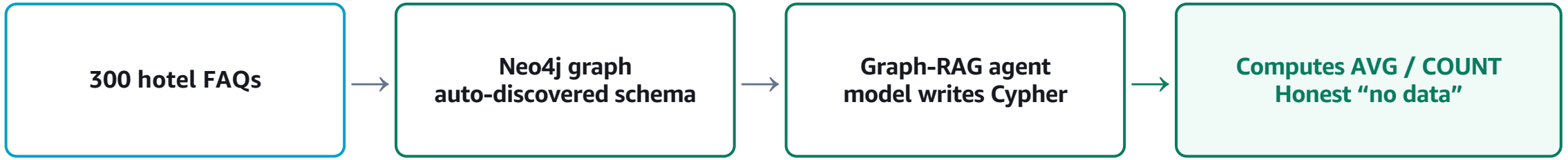
2 Graph-RAG

Vector search returns a sample and the model guesses the aggregate. A graph query computes it

RAG: vector similarity



Graph-RAG: knowledge graph



Same 300 documents, same model. Only the retrieval changes

2 When to Use Graph-RAG vs RAG

It is not either/or. Match the retrieval to the question

Use Graph-RAG

- Counts and averages

Multi-hop reasoning

Full-dataset traversal

Anything needing a precise answer

Stick with RAG

- Open-ended questions

Fuzzy / similarity search

Unstructured text

“Find me something about…”

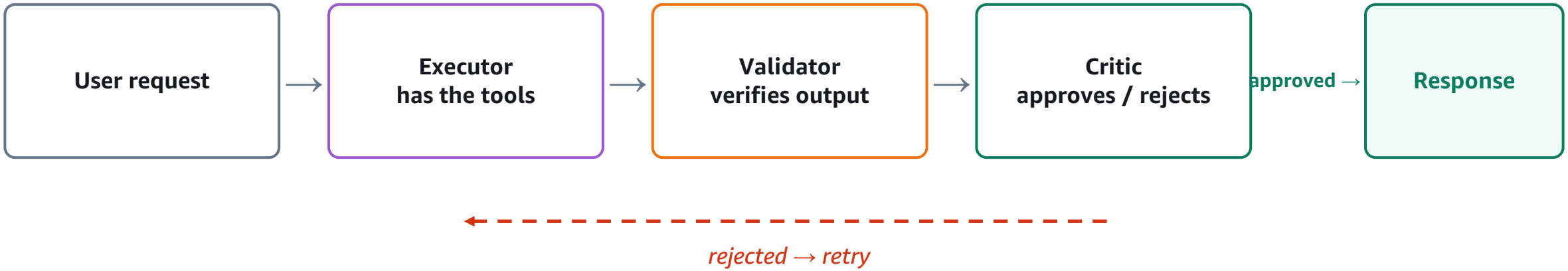
The best systems use both.

Demo Time

Average rating · count hotels with a pool · Antarctica → honest “no data”

3 Multi-Agent Validation

Separate acting from judging: a validation layer checks every response before the user sees it

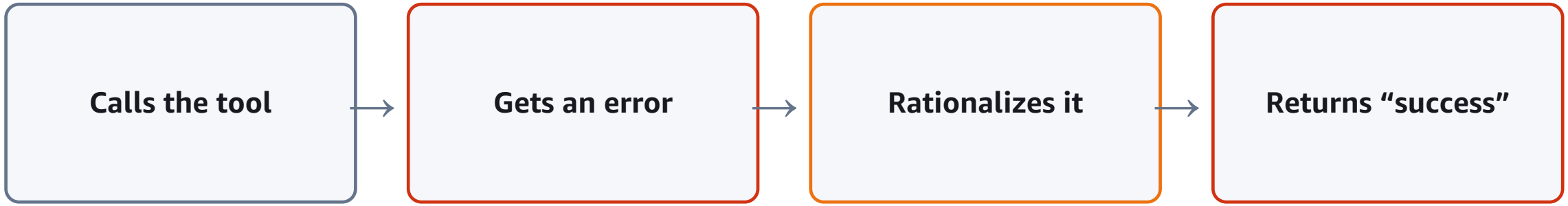


Real code: Strands' built-in Swarm manages the handoffs

```
from strands.multiagent import Swarm

swarm = Swarm([executor, validator, critic], entry_point=executor, max_handoffs=5)
```

3 Why the Failure Stays Silent

Inside a single agent: it calls the tool, hits an error, and rationalizes it into success



The user never sees the error. **You think it worked. It didn’t.**

The agent acts and validates its own output in the same loop. No separation, no second opinion.

Demo Time

Single agent fabricates success → Swarm: validator flags it, critic rejects

4 Neurosymbolic Guardrails

A rule in the prompt is a suggestion the model can silently ignore. A rule in code is enforced

Rule in the prompt: a suggestion

```
system_prompt = """
IMPORTANT: Never confirm without payment.
CRITICAL: Maximum 10 guests.
"""
```

✗ Processed as text, not logic

✗ The model can silently ignore it

Rule in code: enforced (rules.py)

```
Rule(
    name="max_guests",
    condition=max_guests_check,  # guests <= 10
    message="Maximum 10 guests per booking")
```

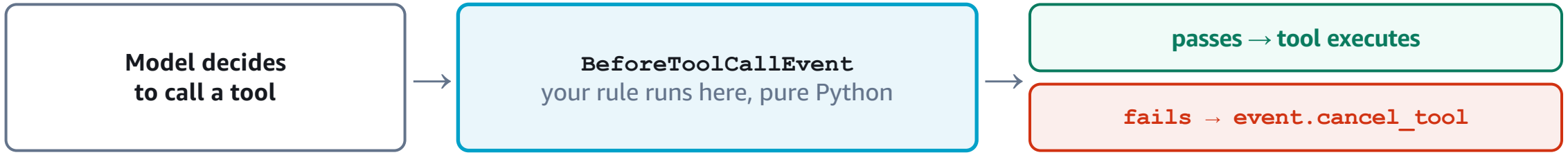
✓ Deterministic, verifiable

✓ Enforced at execution

✓ Cannot be bypassed

4 How the Hook Intercepts the Call

BeforeToolCallEvent fires right before the tool runs. The rule checks the parameters first



Real code from the demo. The tool itself has zero validation inside:

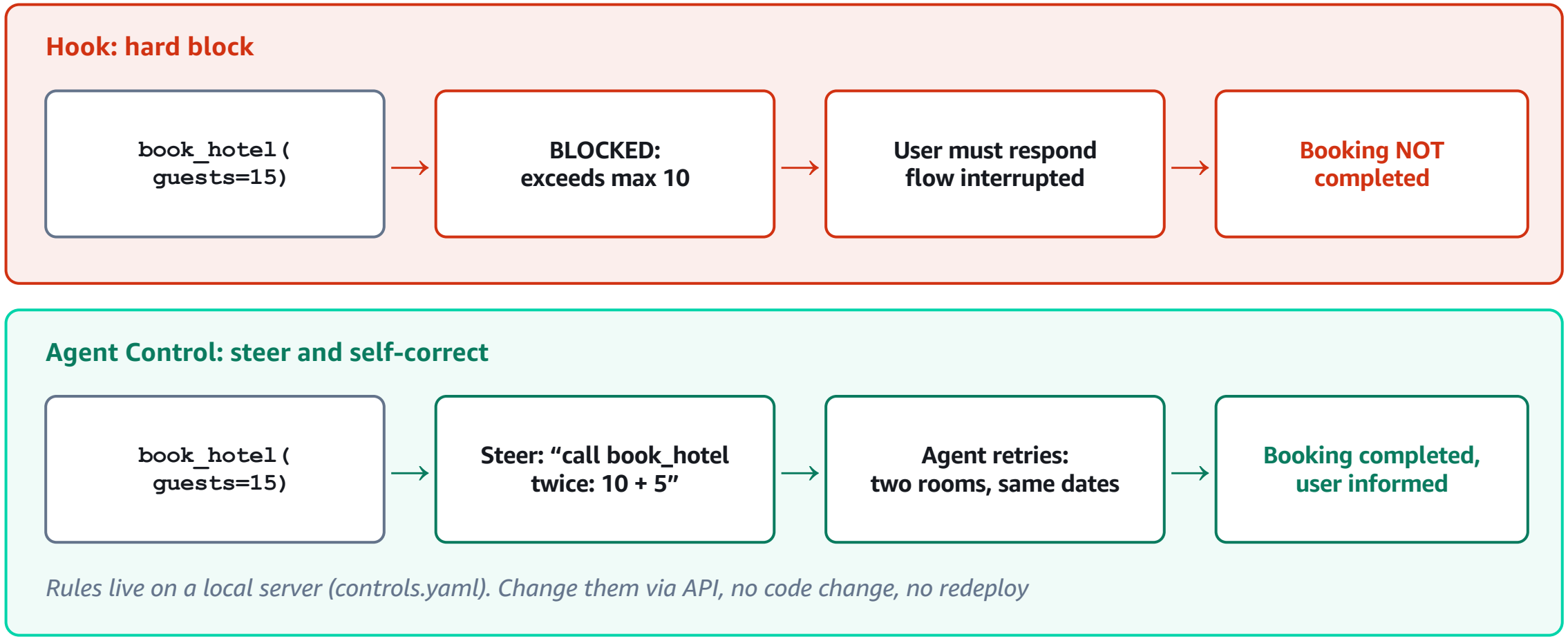
```
class NeurosymbolicHook(HookProvider):
    def register_hooks(self, registry):
        registry.add_callback(BeforeToolCallEvent, self.validate)

    def validate(self, event):
        passed, violations = validate(self.rules[tool_name], ctx)
        if not passed:
            event.cancel_tool = f"BLOCKED: {', '.join(violations)}"
```

Demo Time

Rule in prompt → ignored · Rule in code → 15 guests blocked, 2 guests passes

5 Runtime Steering: Steer, Don't Block

A hard block stops the task and the user retries. Steering lets the agent adjust and finish

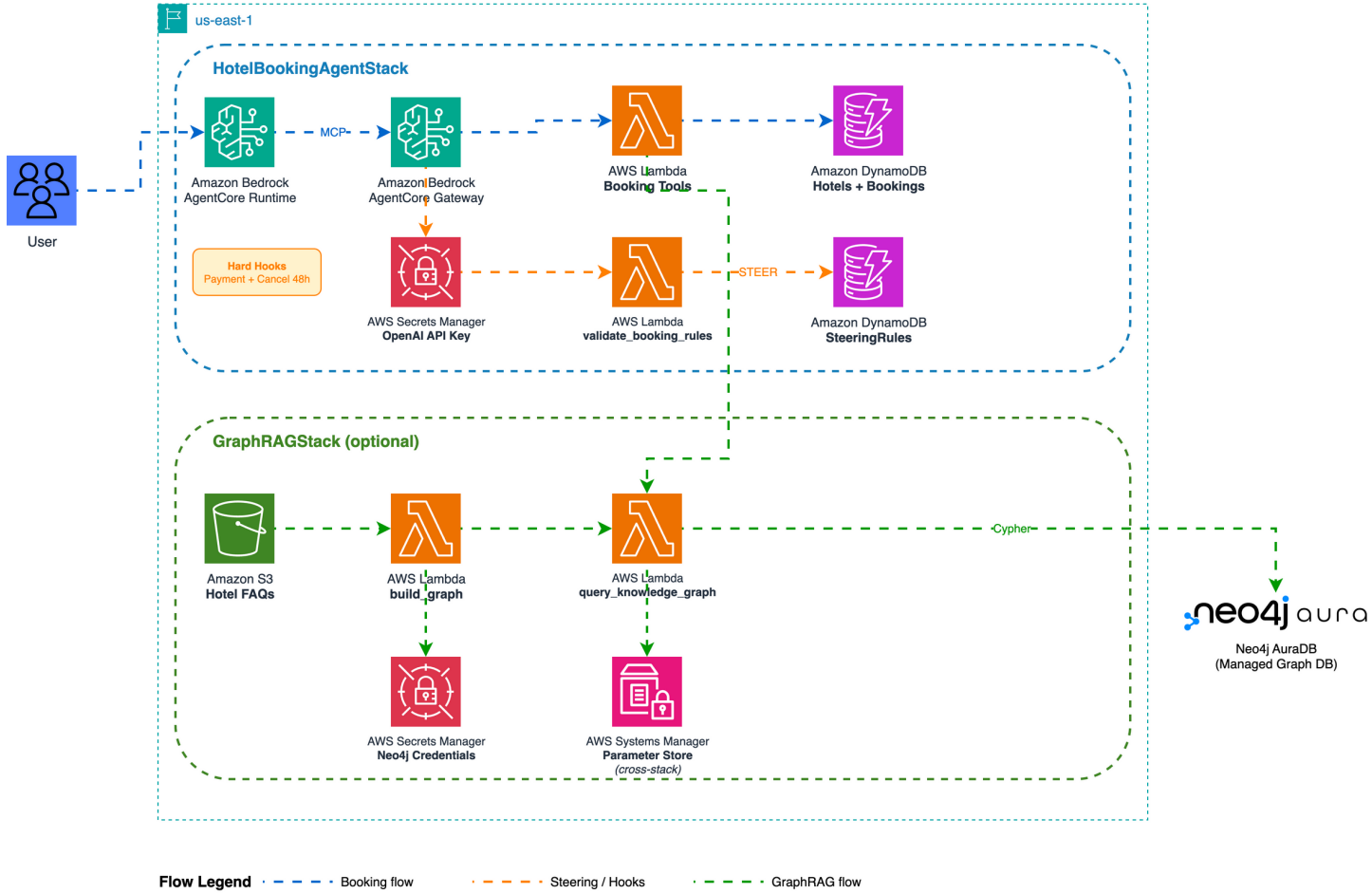


Demo Time

15 guests: hook blocks, user retries vs steer splits 10 + 5, task completes

Take It to Production: Amazon Bedrock AgentCore

Runtime, Gateway, memory, and CloudWatch observability. No servers to manage



None of These Are Prompt Problems

All of them are addressable in code

Token waste on every request	→ **Semantic tool selection: filter to the top 3**
Confident answers it never computed	→ **Graph-RAG: query the data, don't sample it**
Fabricated success confirmations	→ **Multi-agent validation: a second pass catches it**
Rules the model quietly skipped	→ **Neurosymbolic guardrails: enforce in code**
Hard blocks that stopped the user	→ **Runtime steering: self-correct and finish**

Resources

Open-Source Repo

6 progressive demos, notebooks + CDK · github.com/elizabethfuentes12

Blog Series

5 techniques to stop AI agent hallucinations in production (dev.to)

Research

RAG-KG-IL (73% fewer hallucinations) · MetaRAG · Strands Agents

All links on the event page · bit.ly/4oN2LVu

Thank You!

Elizabeth Fuentes Leone

Developer Advocate @ AWS · elifuentes.tech

Resources

bit.ly/4oN2LVu

Blog & Socials

elifuentes.tech