

Stop AI Agent Hallucinations

With Code, Not Prompts

Elizabeth Fuentes Leone
Developer Advocate, GenAI @ AWS

Sandhya Subramani
Senior Developer Advocate, GenAI @ AWS

Laura Salinas
Developer Advocate, GenAI @ AWS



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Four hallucinations we fix with code

01

Fabricated statistics

RAG agents guess numbers instead of computing them.

02

Wrong tool selection

With many similar tools, the agent picks the wrong one.

03

Undetected failures

One model answers with confidence and no signal that it guessed.

04

Rule violations

Rules that live only in the prompt get ignored.

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Foundations plus five structural fixes

00

Foundation

Loop, tools, hooks.

01

Graph-RAG

Compute in Neo4j, not from chunks.

02

Semantic tool selection

Only the 3 most relevant tools.

03

Model consortium

Several models vote on the answer.

04

Guardrails in code

Business rules enforced by hooks.

05

Agent steering

Guide the agent to fix the request.

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

BUILT WITH

Strands Agents

STRANDS

The open source toolkit for building production agents.

Model driven

Use model reasoning to plan and act, instead of brittle hand written workflows.

A few lines of code

Turn any function into a tool with a decorator.

Multi-agent

From one agent to collaborative systems: agents as tools, swarms, graphs.

Production ready

Memory, sessions, context management, guardrails, and observability built in.

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

BUILT WITH

One SDK, any model

Switch models without changing your code.

- Amazon Bedrock
- Anthropic
- Google
- Meta Llama
- OpenAI
- Ollama
- Mistral
- DeepSeek
- And more

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Loop, tools, hooks

01

Agent loop

The model reasons, calls your tools, and keeps going until the task is done.

02

Tools

A Python function with a decorator. The model reads its description to decide when to call it.

03

Hooks

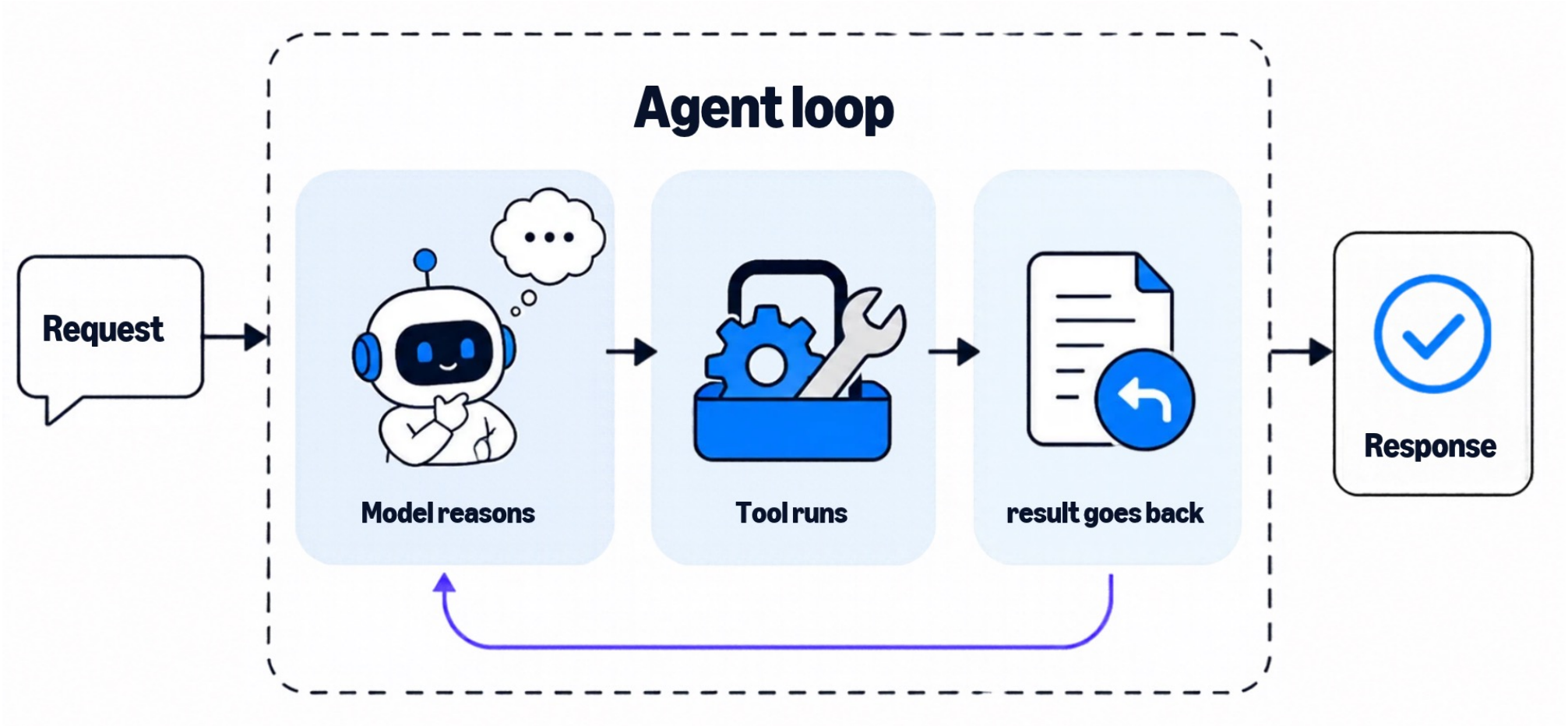
Run your code before and after the invocation, the model call and the tool call.

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The agent loop

The model reasons, a tool runs, the result goes back



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

A tool is a Python function

- **A decorator turns a function into a tool**
- **The docstring becomes the description**
- **Type hints become the input schema**

customer_service_tools.py

```python
from strands import tool

@tool
def lookup_customer(customer_id: str) -> str:
    """Look up a customer by their ID.

    Args:
        customer_id: The customer ID (e.g. C-1001)
    """
    ...  # find the customer in CUSTOMERS
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Tip: write the description the model reads

- **What it does**
- **When to use it**
- **What each argument means**
- **What it does not do**

semantic_tools.py

```python
@tool
def get_hotel_room_rates(hotel: str) -> dict:
    """Get how much rooms cost at a specific hotel:
    nightly price in USD per room type.

    Use this for questions about price, cost, how much
    a room is, or room rates.

    Args:
        hotel: Exact hotel name.
    """

# a near duplicate says: "No prices."
```

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

An agent: model, prompt, tools

Default model provider: Amazon Bedrock

One agent keeps the conversation across turns

chat.py

```
from strands import Agent

agent = Agent(
    tools=[lookup_customer, get_order_history,
           process_refund],
    system_prompt=SYSTEM_PROMPT,
)

# one turn: model -> tool -> result -> model -> answer
agent("Hi, I'm customer C-1001. What are my recent orders?")

# next turn: same agent, same history
agent("...")
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hooks fire before and after each stage

INVOCATION

One request

BEFORE

`BeforeInvocationEvent`

AFTER

`AfterInvocationEvent`

YOU CAN CHANGE

After: trigger a follow-up request.

Used here: tool selection.

MODEL

Each model call

BEFORE

`BeforeModelCallEvent`

AFTER

`AfterModelCallEvent`

YOU CAN CHANGE

After: ask for a retry.

TOOL

Each tool call

BEFORE

`BeforeToolCallEvent`

AFTER

`AfterToolCallEvent`

YOU CAN CHANGE

Before: cancel, replace or edit the call. After: edit the result or retry.

Used here: rate limit, guardrails.

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

A hook: cap every tool at 3 calls

A new request resets the counts. A tool called too often is cancelled.

02-hooks/chat.py

```python
class RateLimiterHook(HookProvider):
    def __init__(self, max_calls=3):
        self.max_calls, self.counts = max_calls, {}
    def register_hooks(self, registry):
        registry.add_callback(BeforeInvocationEvent, self.reset)
        registry.add_callback(BeforeToolCallEvent, self.check)
    def reset(self, event):  self.counts = {}
    def check(self, event):
        name = event.tool_use["name"]
        self.counts[name] = self.counts.get(name, 0) + 1
        if self.counts[name] > self.max_calls:
            event.cancel_tool = f"'{name}' hit the call limit."
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Add the hook to the agent

One argument adds the hook

Remove it and the agent is as before

02-hooks/chat.py

```
from strands import Agent

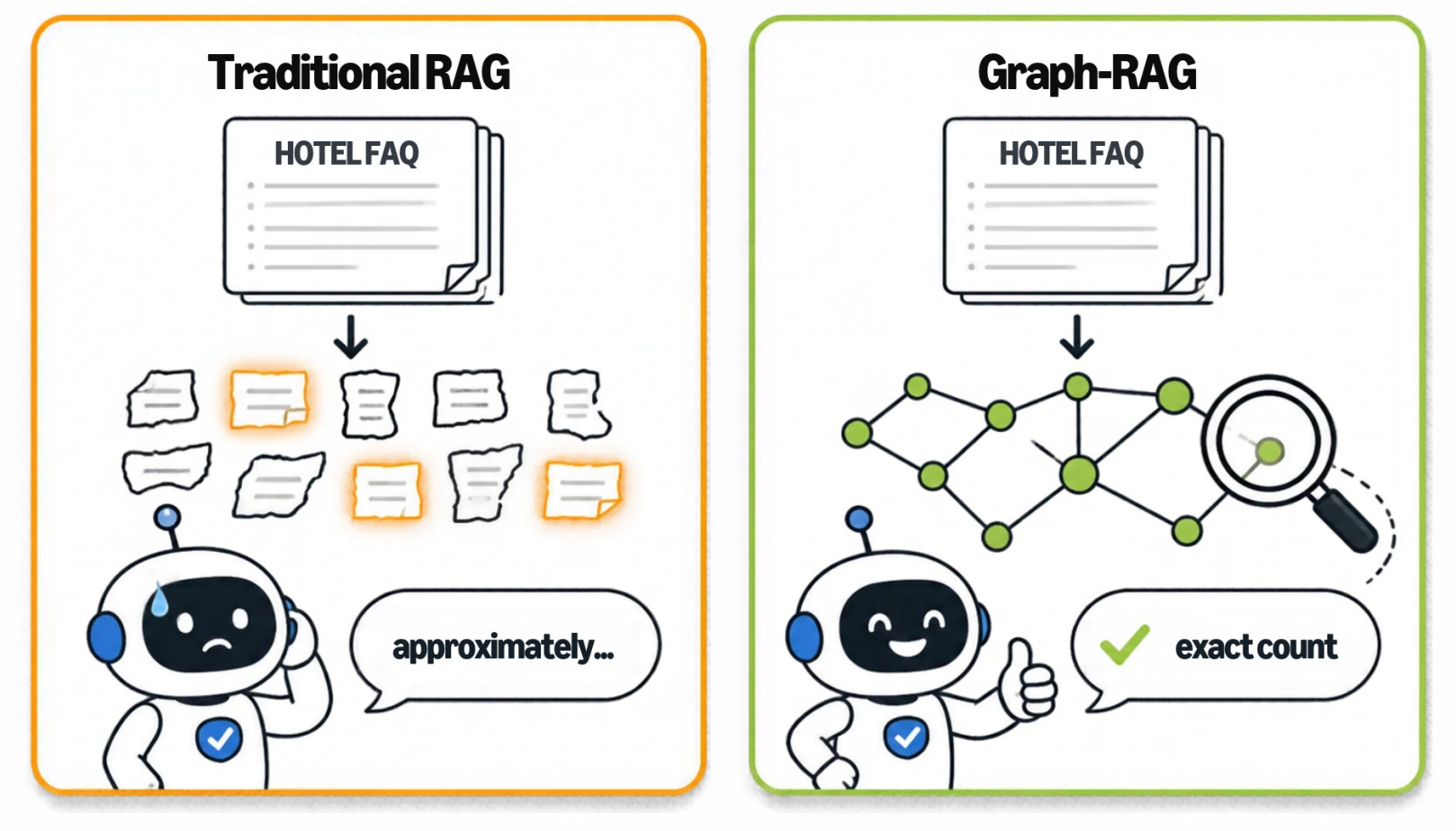

agent = Agent(
    tools=[lookup_customer, get_order_history,
           process_refund],
    system_prompt=SYSTEM_PROMPT,
    hooks=[RateLimiterHook(max_calls=3)],   # the only addition
)
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

TECHNIQUE 01

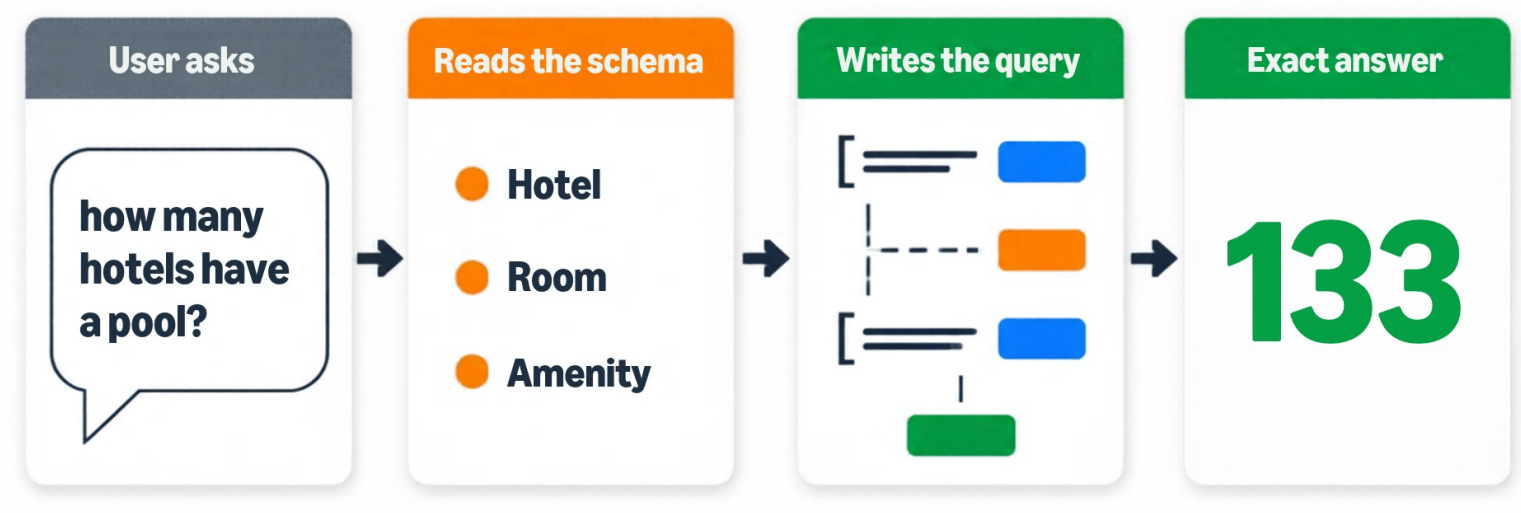
Graph-RAG

Two ways to answer the same question



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Text2Cypher: question in, exact answer out



illustrative

```
// "How many hotels have a pool?"
MATCH (h:Hotel)
      -[:OFFERS_AMENITY]->(a:Amenity)
WHERE toLower(a.name) CONTAINS 'pool'
RETURN count(DISTINCT h) AS hotels
```

illustrative

```
// "Average nightly rate?"
MATCH (:Hotel)-[:HAS_ROOM]->(r:Room)
RETURN avg(r.rate) AS avgRate,
       count(r) AS rooms
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

A read-only Cypher tool

hotel_tools.py

```python
@tool
def query_hotels(self, cypher_query: str) -> dict:
    """Run a READ-ONLY Cypher query ... schema ... """
    records, _, _ = self._driver.execute_query(
        cypher_query, routing_=RoutingControl.READ)
    rows = [dict(r) for r in records[:25]]
    if not rows:
        return {"status": "success",
                "content": [{"text": "No results found."}]}
    return {"status": "success",
            "content": [{"json": {"results": rows}}]}
    # on error: {"status": "error", ...}
```

GROUNDED

Schema in the description

The LLM sees the schema it can query.

READ-ONLY

The tool cannot write

Every query runs in read mode.

HONEST

Empty means no results

No rows: the agent reports it.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The final agents: swap one tool

Same model, same prompt

Only the tool changes the retrieval

test_graphrag.ipynb

```
from strands import Agent

# same model and system prompt for both
rag_agent = Agent(model=model, system_prompt=SYSTEM_PROMPT,
                  tools=[search_faqs])

graph_agent = Agent(model=model, system_prompt=SYSTEM_PROMPT,
                    tools=[query_hotels])
```

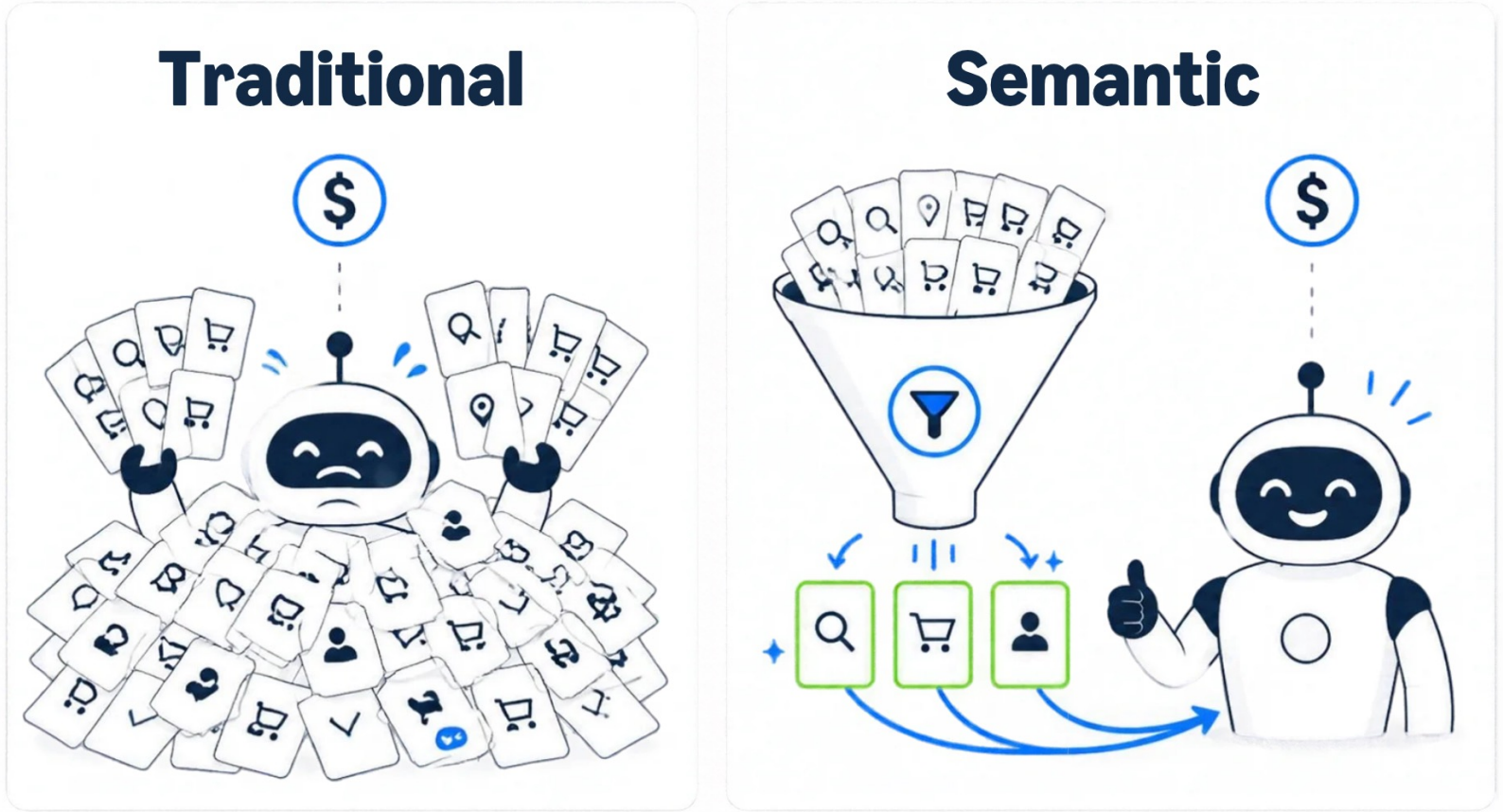
To see how the graph is built, review build_graph.py in the repo.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

TECHNIQUE 02

Semantic tool selection

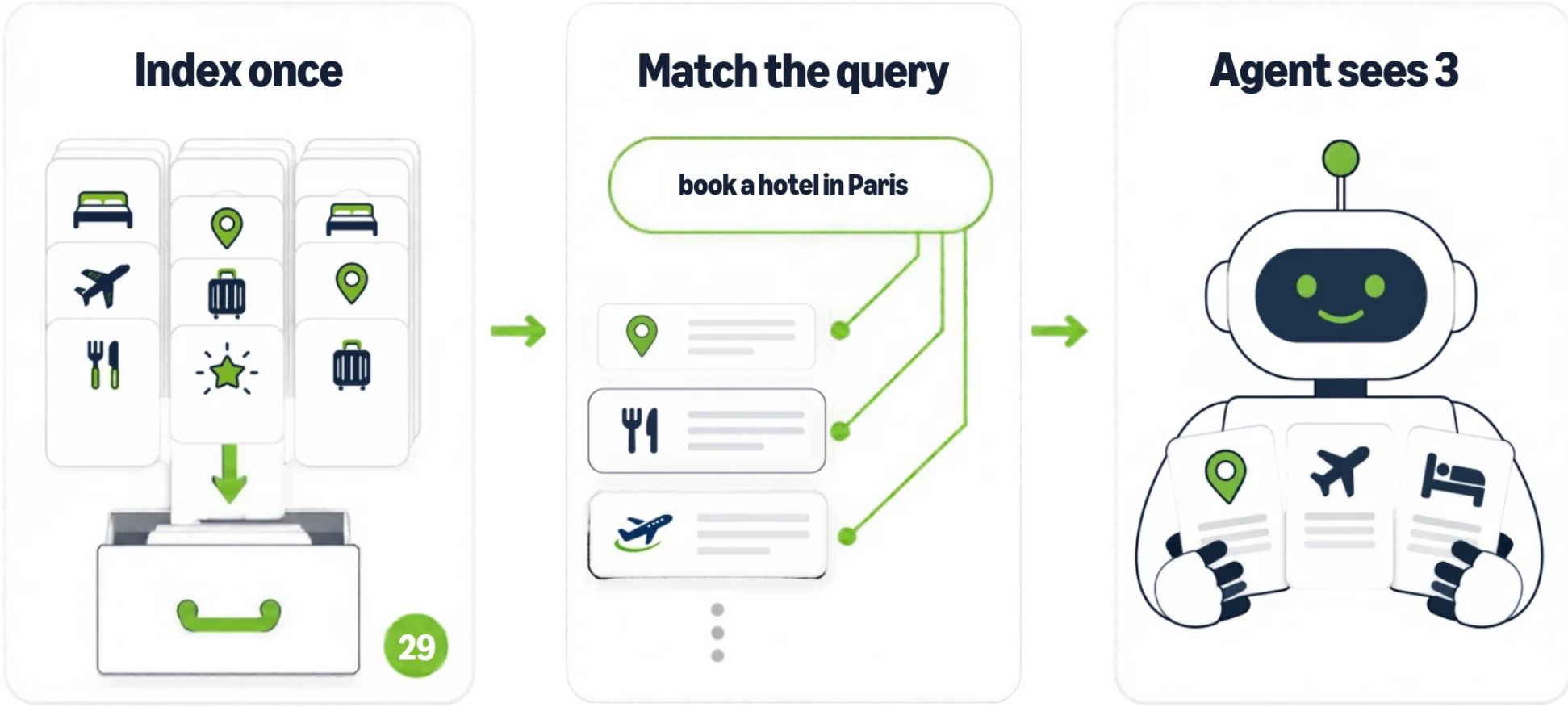
Filter before the agent, not inside it



aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Index the tools once, match each query

Embed name + description once. Embed each query. Keep the closest 3.



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The hook: build the index, then select

One index, built once. One search per request.

semantic_tools.py

```
class SemanticToolHook(HookProvider):
  def __init__(self, tools, top_k=3):
    texts = [f"{t.tool_name}: {t.tool_spec['description']}" for t in tools]
    vecs = np.array([self._embed(t) for t in texts], dtype="float32")
    self._index = faiss.IndexFlatL2(vecs.shape[1])
    self._index.add(vecs)                              # index ONCE
  def select(self, query):
    q = np.array([self._embed(query)], dtype="float32")
    _, idx = self._index.search(q, self.top_k)
    return [self.tools[i] for i in idx[0]]
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The hook swaps the tools per request

- **Runs once per request, before the model**
- **Conversation memory survives the swap**
- **A tool outside the top 3 cannot be called**

semantic_tools.py

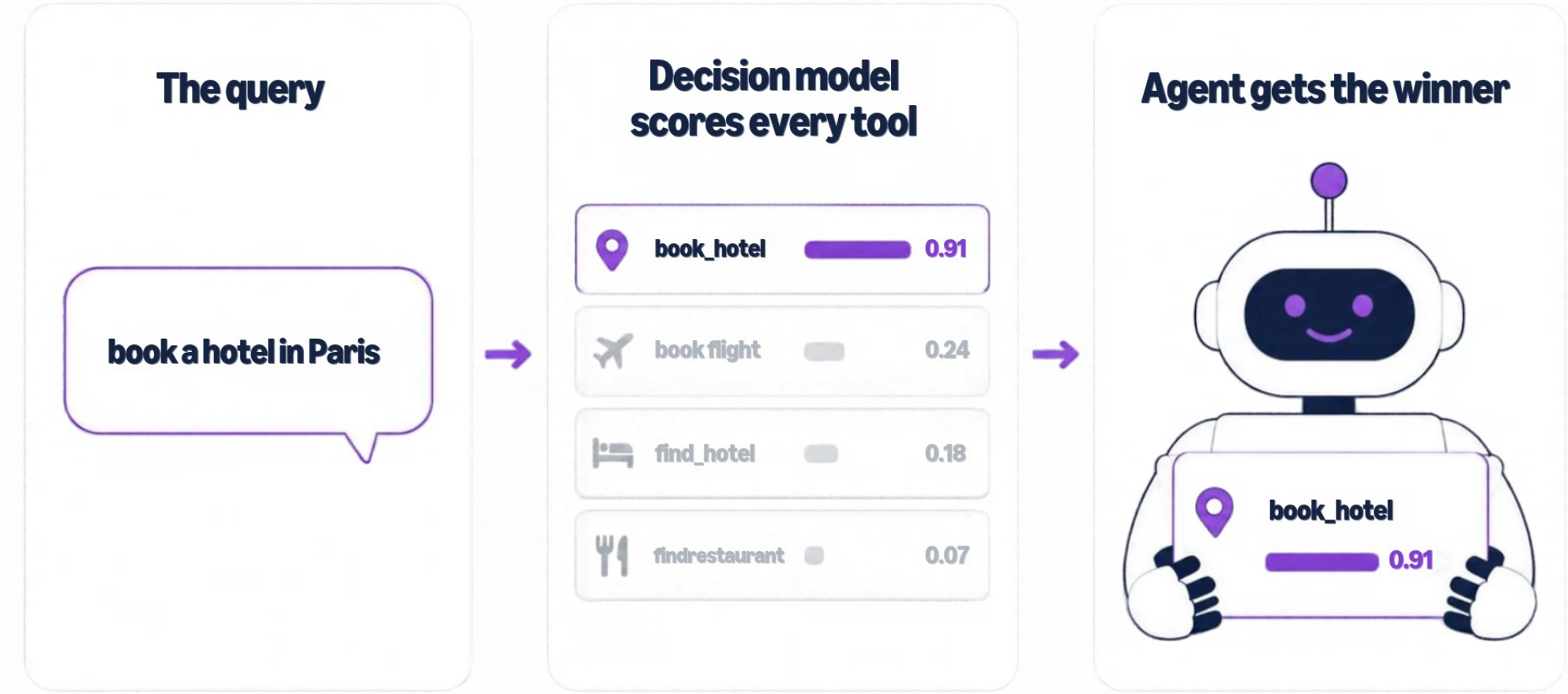
```
def register_hooks(self, registry):
    registry.add_callback(BeforeInvocationEvent, self._trim)
def _trim(self, event):
    query = ...           # latest user message
    selected = self.select(query)
    reg = event.agent.tool_registry
    reg.registry.clear()
    reg.dynamic_tools.clear()
    for t in selected:
        reg.register_tool(t)

agent = Agent(model=MODEL, tools=ALL_TOOLS,
              hooks=[SemanticToolHook(ALL_TOOLS, top_k=3)])
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Or let a decision model pick

Score every tool. The agent gets the winner.



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The decision model in code

One bounded choice over the tool names. The same hook swaps in the winner.

chat_decider.py

```
from strands_decider.infer import load_engine
from strands_decider.schema import ChoiceQuestion

engine = load_engine(DECIDER_CHECKPOINT, device="cpu")

def select(self, query):
    resp = self._engine.ask(query, {"tool": ChoiceQuestion(
        instructions="Which tool best answers this request?",
        criteria=self._criteria)})
    return [self._by_name[resp.answers["tool"].choice]]
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Hosted option: OpenAI Decisions API

- **Same hook, different selector**
- **Returns a choice and a confidence**
- **Reads every tool description each time**

ep3_decision_models.py

```
async def openai_select_tool(query):
    resp = await OAI.decisions.create(
        model="gpt-6-luna", input=query,
        questions=[{"type": "choice", "name": "tool",
            "instructions": "Which tool answers this request?",
            "choices": TOOL_CHOICES}])
    answer = next(a for a in resp.answers
                  if getattr(a, "name", None) == "tool")
    return BY_NAME[answer.choice], answer.confidence


# TOOL_CHOICES: one {value, description} per tool
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The final agent: change one argument

Same agent, same tools.

chat.py · chat_decider.py · ep3

```python
# all tools on every call
agent = Agent(model=MODEL, tools=ALL_TOOLS)

# FAISS keeps the closest 3
agent = Agent(model=MODEL, tools=ALL_TOOLS, hooks=[SemanticToolHook(ALL_TOOLS, top_k=3)])

# a local decision model picks one
agent = Agent(model=MODEL, tools=ALL_TOOLS, hooks=[DeciderToolHook(ALL_TOOLS)])

# a hosted decision model picks one
agent = Agent(model=MODEL, tools=ALL_TOOLS, hooks=[OpenAIToolHook()])
```

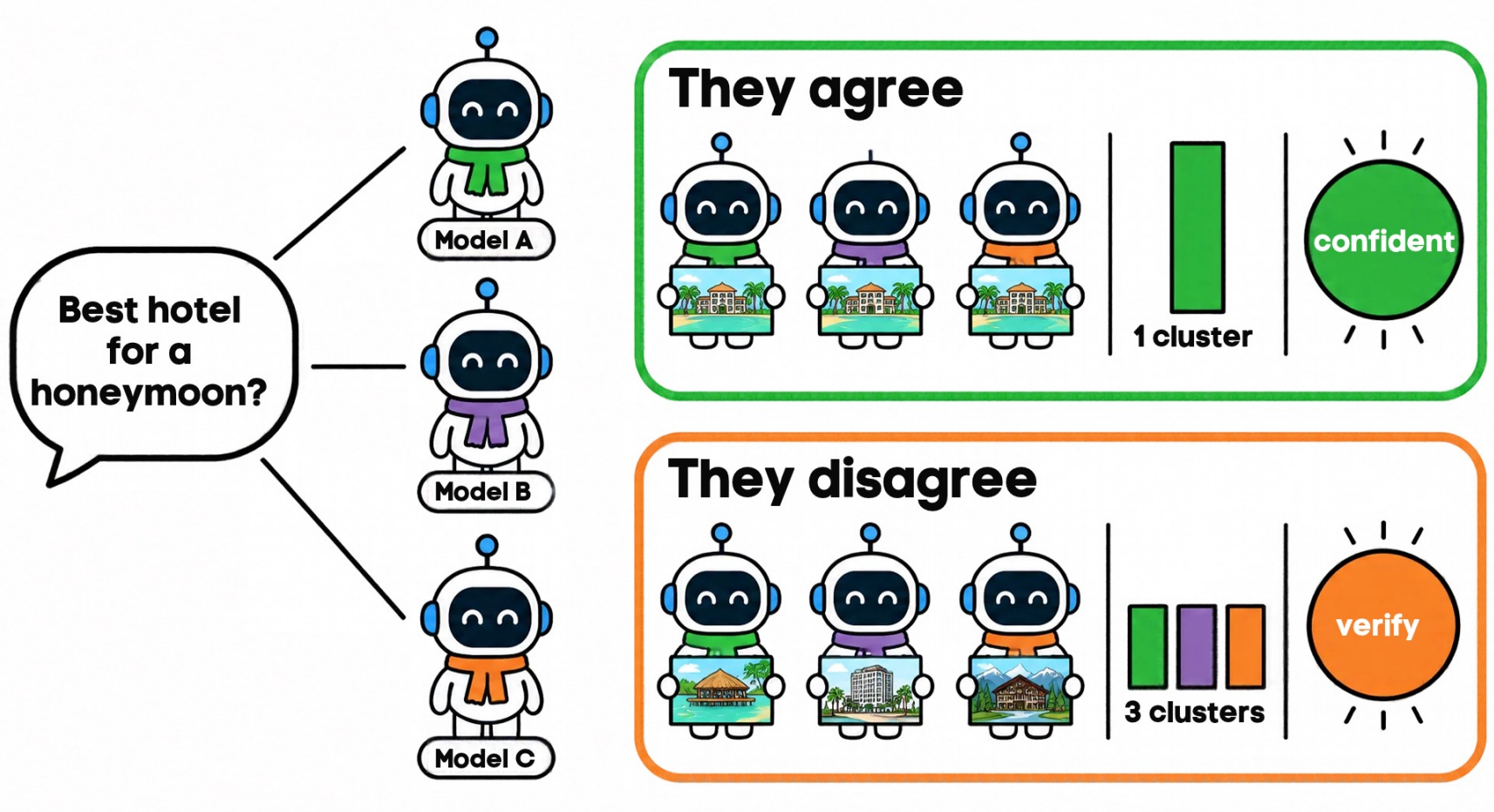
aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

TECHNIQUE 03

Model consortium

One model guesses. Several models vote.



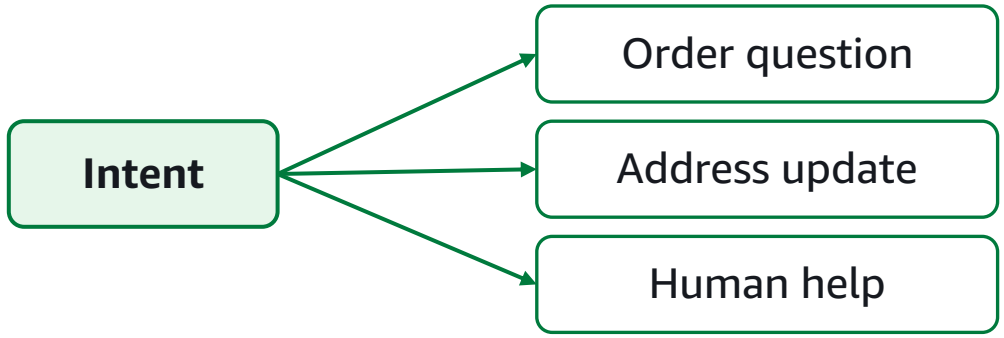
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Graph vs Swarm

Who decides the path?

GRAPH

A flowchart you define

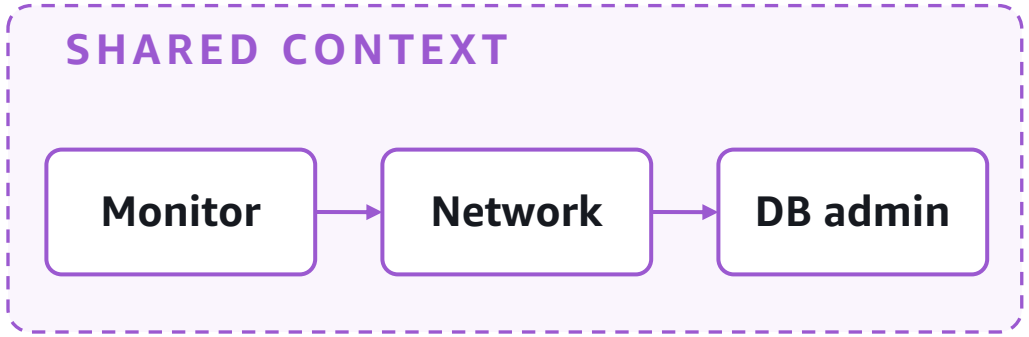


- Nodes and edges defined in advance
- An LLM decision picks each branch
- Error edges route failures

Use for: branching processes

SWARM

Agents hand off to each other



- A pool of agents, no fixed path
- Agents hand off to the best peer
- Handoff limits stop endless loops

Use for: brainstorming, incidents

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Run the models in parallel with a graph

No edges between nodes: every model starts at once and answers alone.

multiagent_tools.py

```
CONSORTIUM = {                                  # two model families
    "claude-haiku": "global.anthropic.claude-haiku-4-5-20251001-v1:0",
    "nova-pro":     "amazon.nova-pro-v1:0",
    "nova-lite":    "us.amazon.nova-2-lite-v1:0",
}
builder = GraphBuilder()
for name, model_id in CONSORTIUM.items():
    agent = Agent(name=name, system_prompt=ADVISOR_PROMPT, callback_handler=None,
                  model=BedrockModel(model_id=model_id, region_name=REGION))
    builder.add_node(agent, name)
    builder.set_entry_point(name)       # every node is an entry point: they run in parallel
graph = builder.build()
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Cluster the answers, measure the agreement

Each answer maps to the hotel it names, so wording differences do not count.

multiagent_tools.py

```
def canonical(text):                          # map an answer to the hotel it names
    hits = [h for h in hotels if h.lower() in text.lower()]
    return max(hits, key=len) if hits else "(no answer)"
answers = {name: canonical(str(node.result))
           for name, node in graph_result.results.items()}
votes = Counter(answers.values())
n = sum(votes.values())
entropy = -sum((c / n) * math.log2(c / n) for c in votes.values())
entropy /= math.log2(len(answers))             # normalized 0..1
verdict = ("CONFIDENT (consensus)" if len(votes) == 1
           else "LIKELY HALLUCINATION (models disagree, verify)")
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The final system: ask, then score

Build the graph once. Each question returns the clusters and a verdict.

multiagent_tools.py

```
graph = build_consortium()
context = hotel_context()          # read-only Neo4j

question = "Which hotel is the single best choice overall for a luxury traveler?"
verdict = ask(graph, question, context, hotels)

verdict["clusters"]    # one hotel, voted by all 3 models
verdict["agreement"]   # 1.0
verdict["verdict"]     # CONFIDENT (consensus)
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Consortium: the takeaway

Agreement

Consensus means confident. Disagreement means verify.

Judgement calls

For exact questions, Graph-RAG or a hook is cheaper.

Different families

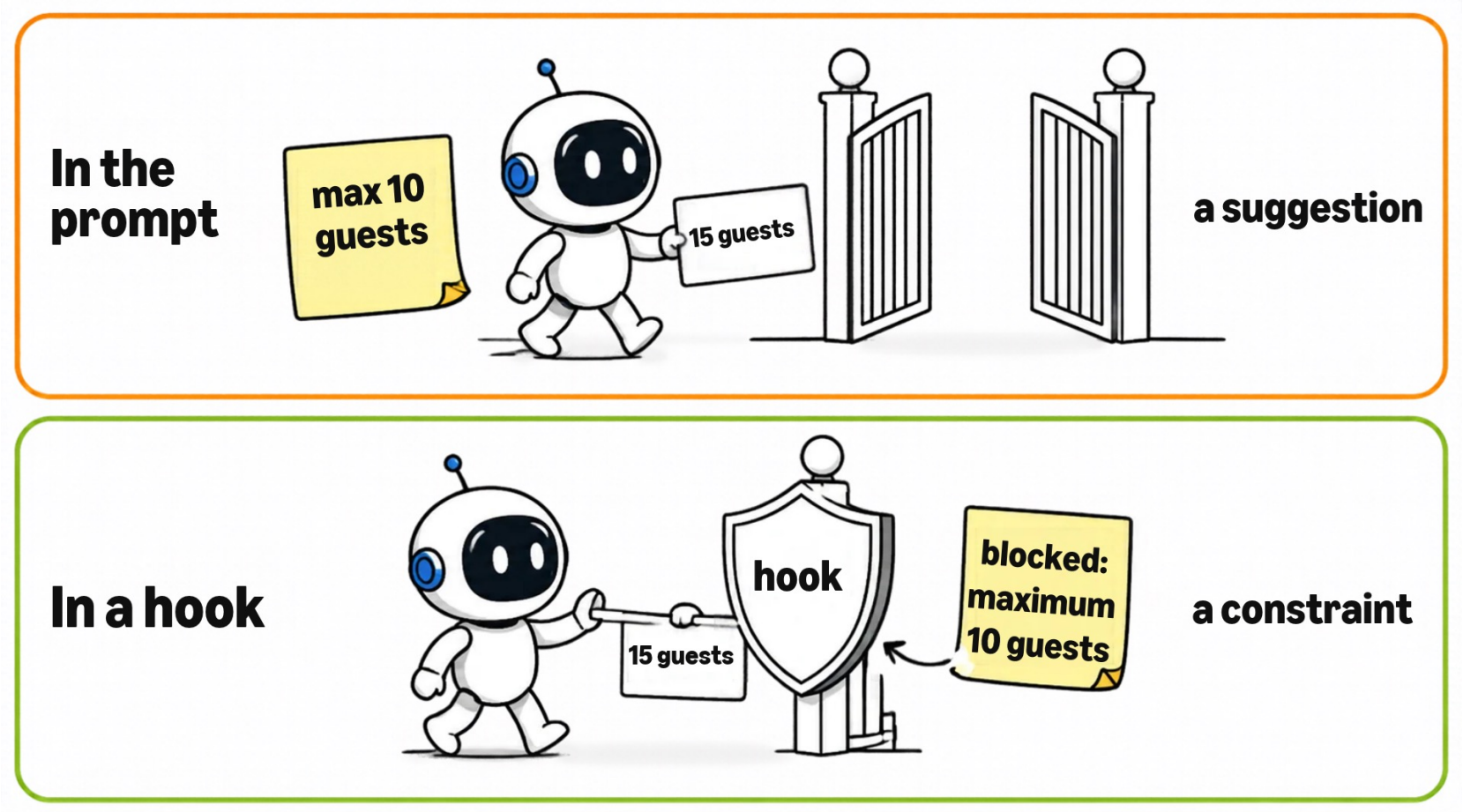
They do not share the same blind spots.

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

TECHNIQUE 04

Guardrails in code

The same rule, in two places



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The hook sits between the model and the tool

- **Runs before every tool**
- **Reads the real tool input**
- **A cancelled tool cannot be bypassed**

neurosymbolic_tools.py

```
class NeurosymbolicHook(HookProvider):
    def __init__(self, graph, store): ...

    def register_hooks(self, registry):
        registry.add_callback(BeforeToolCallEvent, self.enforce)

    def enforce(self, event):
        name = event.tool_use["name"]
        params = event.tool_use["input"]
        violations = self._check(name, params)
        if violations:
            event.cancel_tool = ("BLOCKED by business rules: "
                                 + "; ".join(violations))
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The rules are plain Python, grounded in data

Two rules read the knowledge graph. A third requires payment first.

neurosymbolic_tools.py

```
def _check_booking(self, params):                          # book_hotel
    violations = []
    hotel = params.get("hotel", "")
    guests = int(params.get("guests", 1))
    capacity = self._graph.capacity(hotel)          # Neo4j, read-only
    if capacity is None:
        violations.append(f"hotel '{hotel}' does not exist in the knowledge graph")
    elif guests > capacity:
        violations.append(f"{guests} guests exceed the room capacity of {capacity}")
    if guests > MAX_GUESTS:                         # company policy
        violations.append(f"maximum {MAX_GUESTS} guests per booking")
    return violations
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Add the guardrail to the agent

The rules are one argument.

test_neurosymbolic_hooks.ipynb

```
from strands import Agent
from neurosymbolic_tools import HotelGraph, BookingStore, NeurosymbolicHook

graph = HotelGraph()                              # read-only Neo4j
store = BookingStore(path="bookings.json")
TOOLS = [store.book_hotel, store.process_payment, store.confirm_booking]

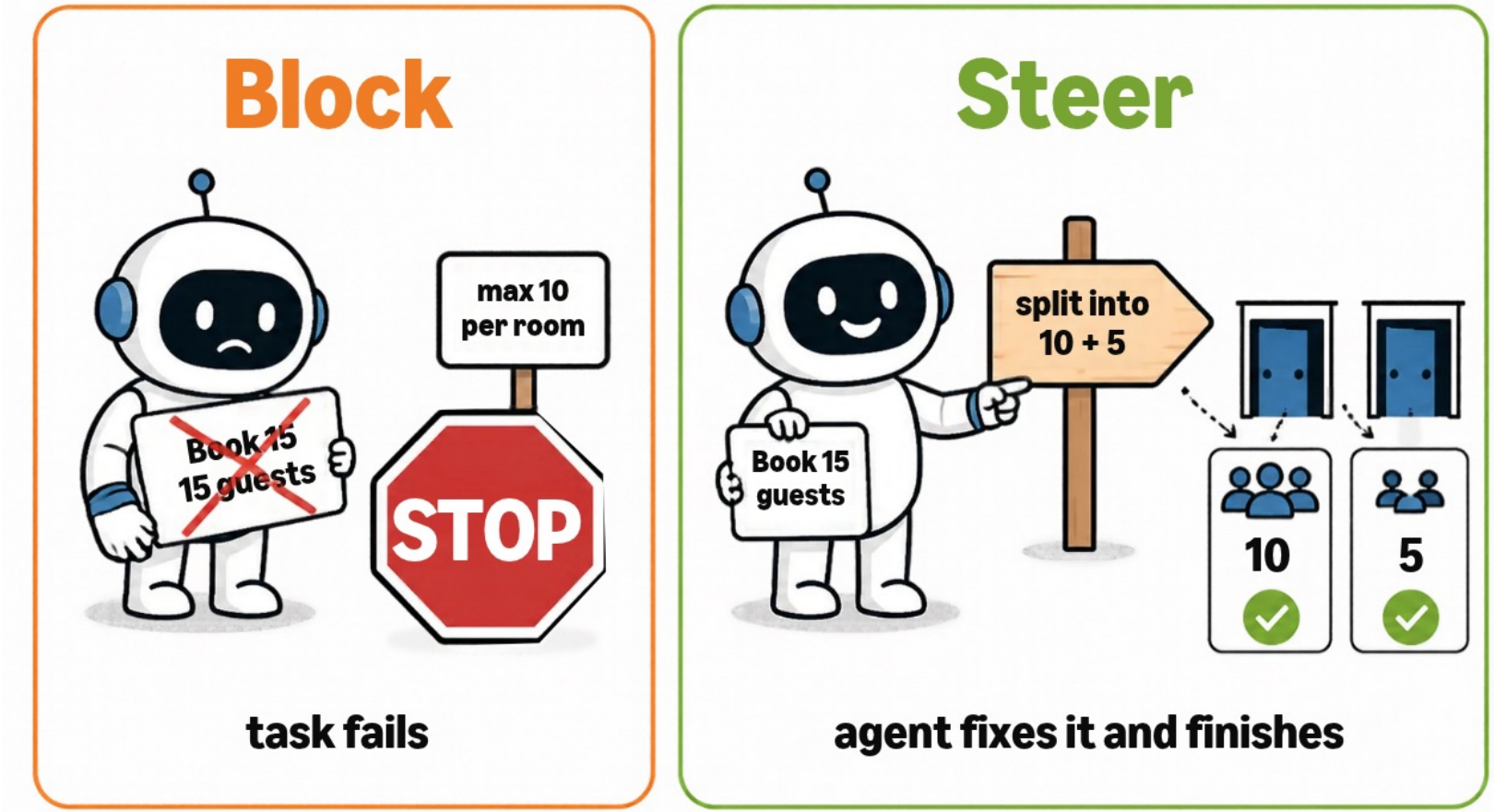
agent = Agent(model=MODEL, tools=TOOLS, system_prompt=PROMPT,
              hooks=[NeurosymbolicHook(graph, store)])      # the only addition
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

TECHNIQUE 05

Agent steering

Same request, two reactions



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The contrast: a hard block

Right when the action must never happen

Wrong when the request can be fixed

steering_tools.py

```
class BlockOversizeRoomHook(HookProvider):
    def register_hooks(self, registry):
        registry.add_callback(BeforeToolCallEvent, self._block)

    def _block(self, event):
        tool_use = event.tool_use
        if tool_use.get("name") != "book_hotel":
            return
        guests = (tool_use.get("input") or {}).get("guests", 1)
        if guests and guests > MAX_PER_ROOM:
            event.cancel_tool = (
                f"Rejected: {guests} guests exceeds "
                f"the {MAX_PER_ROOM}-per-room limit.")
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The fix: steering that returns a Guide

Steering reads the real tool input and returns guidance. It is a plugin.

steering_tools.py

```python
class MaxGuestsSteering(SteeringHandler):
    async def steer_before_tool(self, *, agent, tool_use, **kwargs):
        if tool_use.get("name") != "book_hotel":
            return Proceed()
        guests = (tool_use.get("input") or {}).get("guests", 1)
        if guests and guests > MAX_PER_ROOM:
            rooms = split(guests)                    # 15 -> [10, 5]
            return Guide(reason=(f"Split the party into {len(rooms)} rooms of {rooms}"
                                 " and call book_hotel once per room."))
        return Proceed()

agent = Agent(tools=[hotels.hotel_exists, store.book_hotel], plugins=[MaxGuestsSteering()])
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

A generic split, and the rule in a config file

Not hardcoded: 23 guests gives 10, 10, 3

The limit can live in a config file

steering_tools.py

```python
def split(guests, cap=MAX_PER_ROOM):
    full, rem = divmod(guests, cap)
    return [cap] * full + ([rem] if rem else [])
# 15 -> [10, 5]    23 -> [10, 10, 3]

rules = json.load(open("rules.json"))

class RuleSteering(SteeringHandler):
    def __init__(self, rules):
        super().__init__()
        self._cap = int(rules["max_guests_per_room"])
    # steer_before_tool: same as before, with self._cap
```

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Block or steer: change one argument

A hook blocks. A plugin steers.

test_hooks_vs_control.ipynb

```
from strands import Agent
from steering_tools import HotelCheck, BlockOversizeRoomHook, MaxGuestsSteering
from booking_store import BookingStore

tools = [hotels.hotel_exists, store.book_hotel]

# block: the call is cancelled
agent = Agent(system_prompt=PROMPT, tools=tools, hooks=[BlockOversizeRoomHook()])

# steer: the agent is guided to fix it
agent = Agent(system_prompt=PROMPT, tools=tools, plugins=[MaxGuestsSteering()])
```

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Pick the fix by the failure

How many hotels have a pool? exact count	→	**Graph-RAG**
Forty similar tools wrong picks, token waste	→	**Semantic tool selection**
Best hotel for a honeymoon? judgement call	→	**Model consortium**
Confirm before payment must never happen	→	**Guardrail hook**
Book 15 guests, limit is 10 can be fixed	→	**Steering**

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank You!

Elizabeth Fuentes Leone
Developer Advocate, GenAI @ AWS

Sandhya Subramani
Senior Developer Advocate, GenAI @ AWS

Laura Salinas
Developer Advocate, GenAI @ AWS

STRANDS

Resources
bit.ly/4wFPxNA

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.