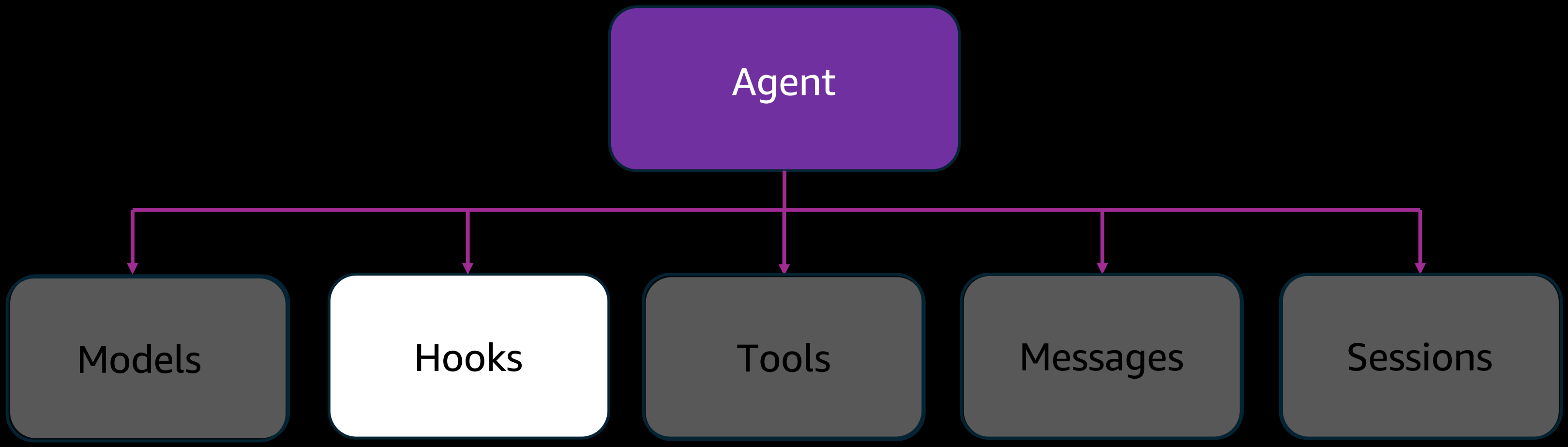
03 Advanced Response Processing with Hooks

Agenda

- Understand the hook system and event lifecycle
- Implement custom hooks for various events
- Build advanced agent behaviors with hooks

Strands Agent() Architecture



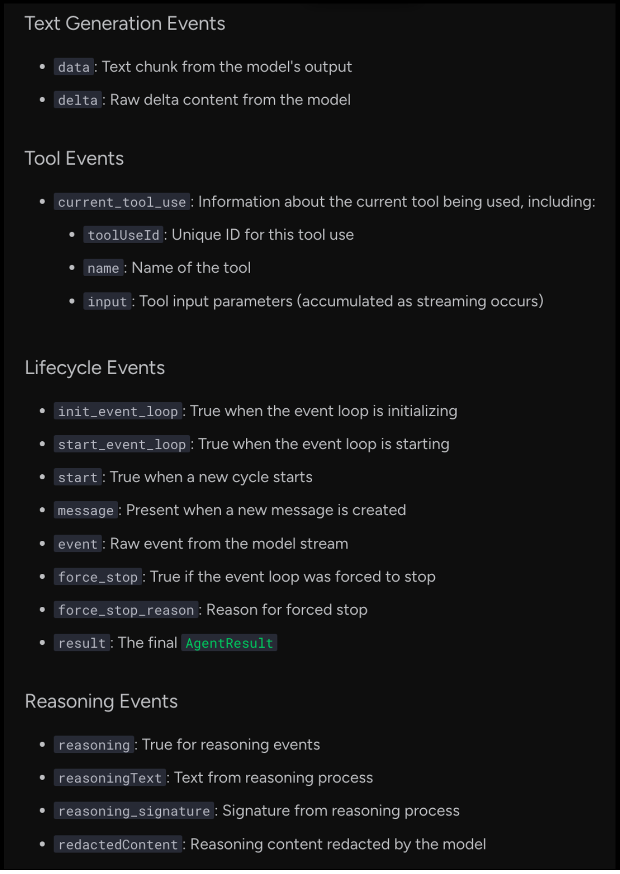
"Think of an Agent as an orchestrator that coordinates between your model, tools, and custom logic through hooks."

Processing responses in Strands

Three methods to intercept and process events:

- Async iterators
- Callback handlers *(default: PrintingCallbackHandler)*
- Hooks (*v1 release*)

Async Iterators



Text Generation Events

- `data`: Text chunk from the model's output
- `delta`: Raw delta content from the model

Tool Events

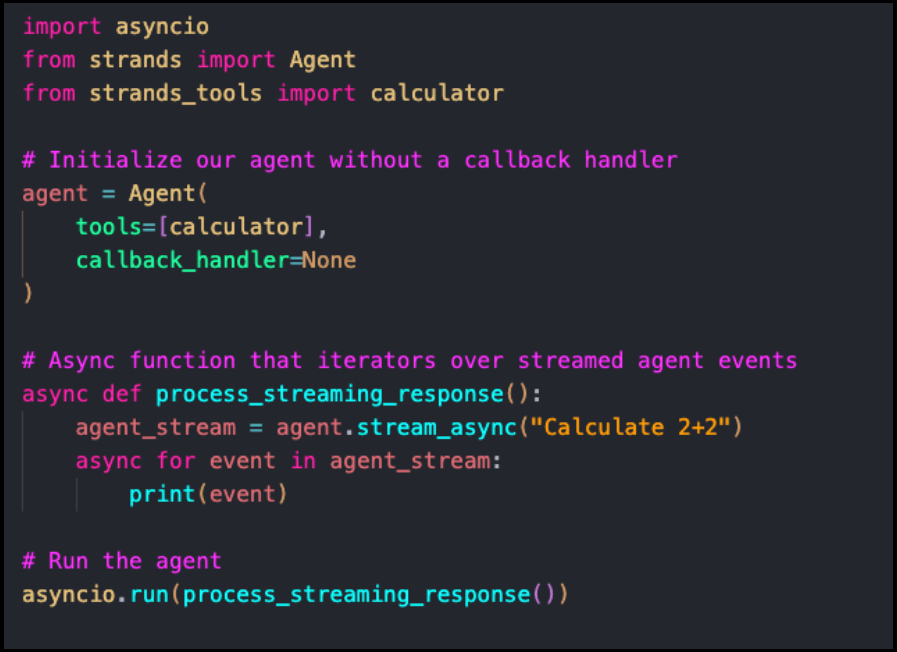
- `current_tool_use`: Information about the current tool being used, including:
  - `toolUseId`: Unique ID for this tool use
  - `name`: Name of the tool
  - `input`: Tool input parameters (accumulated as streaming occurs)

Lifecycle Events

- `init_event_loop`: True when the event loop is initializing
- `start_event_loop`: True when the event loop is starting
- `start`: True when a new cycle starts
- `message`: Present when a new message is created
- `event`: Raw event from the model stream
- `force_stop`: True if the event loop was forced to stop
- `force_stop_reason`: Reason for forced stop
- `result`: The final `AgentResult`

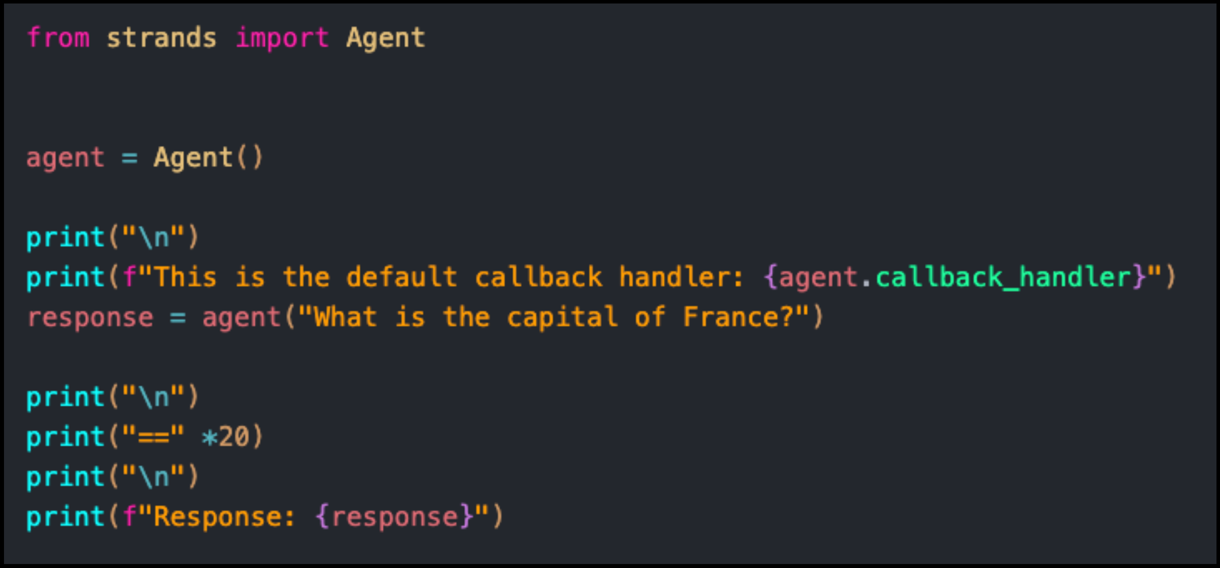
Reasoning Events

- `reasoning`: True for reasoning events
- `reasoningText`: Text from reasoning process
- `reasoning_signature`: Signature from reasoning process
- `redactedContent`: Reasoning content redacted by the model

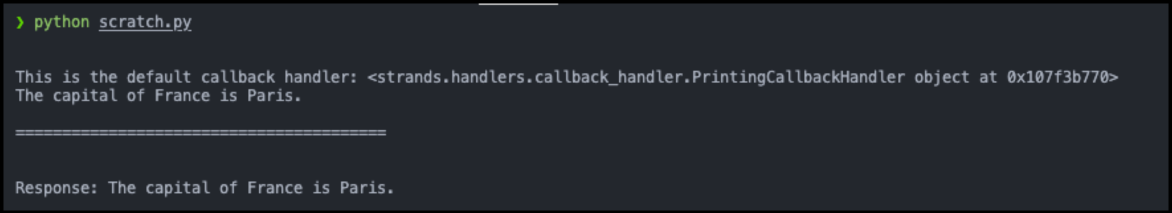


```python
import asyncio
from strands import Agent
from strands_tools import calculator

# Initialize our agent without a callback handler
agent = Agent(
    tools=[calculator],
    callback_handler=None
)

# Async function that iterators over streamed agent events
async def process_streaming_response():
    agent_stream = agent.stream_async("Calculate 2+2")
    async for event in agent_stream:
        print(event)

# Run the agent
asyncio.run(process_streaming_response())
```

Callback Handlers



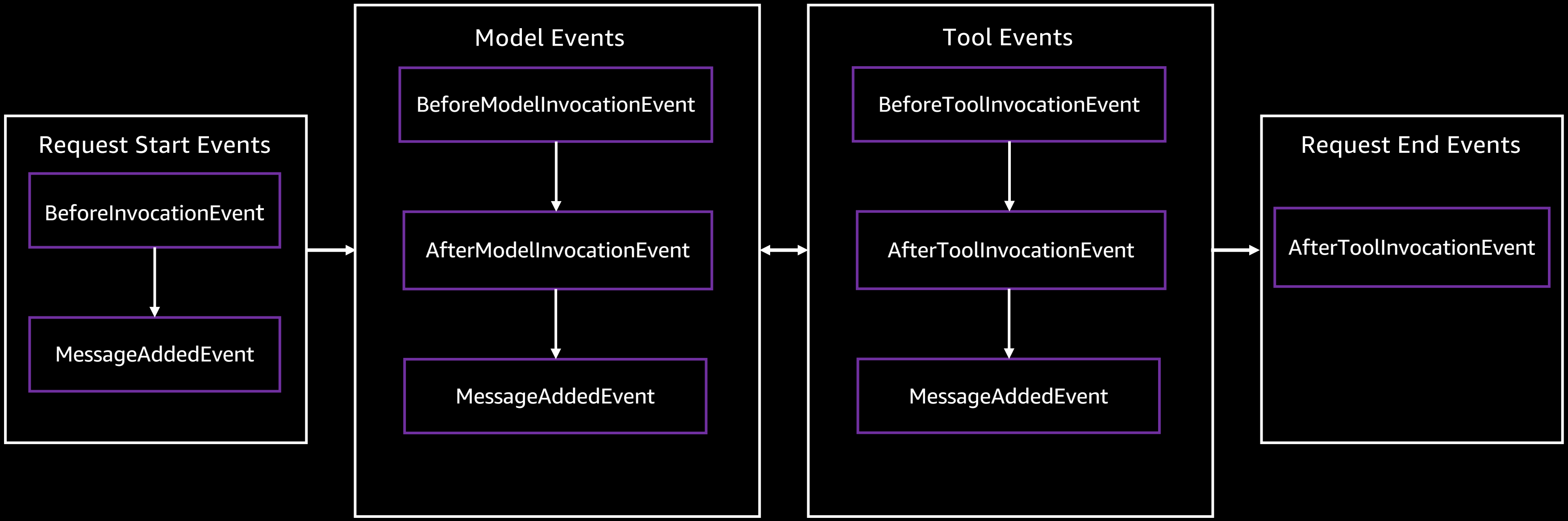
```python
from strands import Agent


agent = Agent()

print("\n")
print(f"This is the default callback handler: {agent.callback_handler}")
response = agent("What is the capital of France?")

print("\n")
print("==" *20)
print("\n")
print(f"Response: {response}")
```



```
❯ python scratch.py


This is the default callback handler: <strands.handlers.callback_handler.PrintingCallbackHandler object at 0x107f3b770>
The capital of France is Paris.

========================================


Response: The capital of France is Paris.
```

Hooks

The hook system enables you to inject custom logic at specific points in the agent lifecycle to react to or modify agent behavior through strongly-typed event callbacks.



Demo

Key Takeaways

Best Practices

- Keep hook callbacks lightweight since they execute synchronously
- Design hooks to be composable and reusable
- When modifying event properties, log the changes for debugging and audit purposes

Where’s the code?

https://s12d.com/advanced-strands-agents