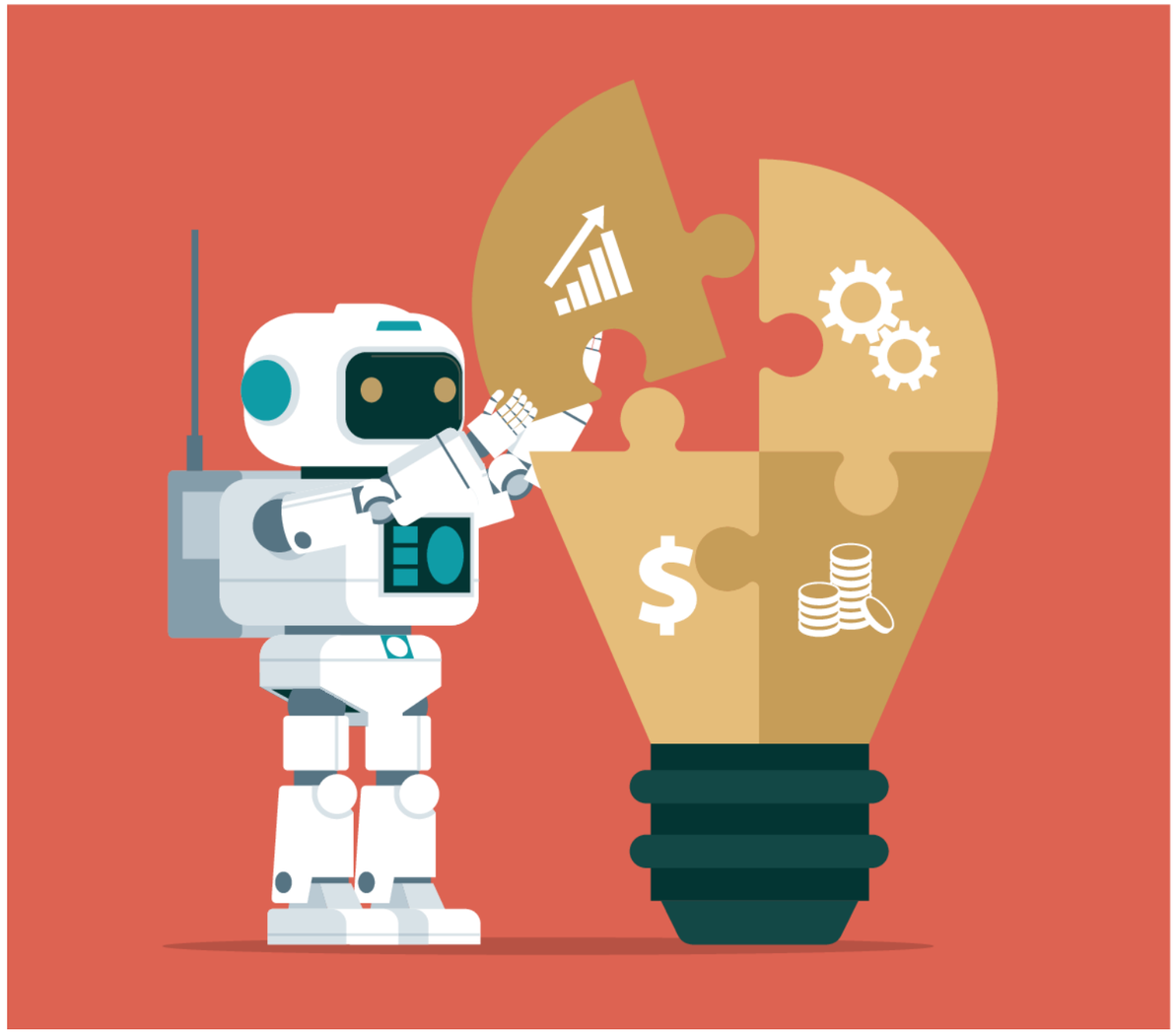
Defining multiple tools

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Dilini K. Sumanapala, PhD
Founder & AI Engineer, Genverv, Ltd.

Integrating multiple tools



- Add multiple custom tools
- Automate tool selection per query

Enhancing an education chatbot

- Look up historical events
- Check palindromes:
  - level = level
  - top spot = tops pot

Multiple ways to build tools

- **Invoke the LLM**

  Look up historical dates using natural language inputs, eg. "5th of November"

```
Agent: The 5th of November is famous for the Gunpowder treason and
plot...
```

- **Python code**

  Find palindromes by directly comparing strings, eg. `string == string[::-1]`

```
Agent: Yes, "madam" is a palindrome...
```

Historical events tool

```
# Use a decorator to label the tool and set the input format to string
@tool
def date_checker(date: str) -> str:
    """Provide a list of important historical events for a given date in any format."""
    try:
          # Invoke the LLM to interpret the date and generate historical events
        answer = llm.invoke(f"List important historical events that occurred on {date}.")

        # Return the response
        return answer.content

    # Set an exception block for errors in retrieval
    except Exception as e:
        return f"Error retrieving events: {str(e)}"
```

Palindrome tool

```
@tool
# Set input format to string
def check_palindrome(text: str):
    """Check if a word or phrase is a palindrome."""

    # Remove non-alphanumeric characters and convert to lowercase
    cleaned = ''.join(char.lower() for char in text if char.isalnum())

    # Check if the reversed text is the same as original text
    if cleaned == cleaned[::-1]:
        return f"The phrase or word '{text}' is a palindrome."
    else:
        return f"The phrase or word '{text}' is not a palindrome."
```

Binding multiple tools

```
# Import modules required for defining tool nodes
from langgraph.prebuilt import ToolNode

# List of tools
tools = [wikipedia_tool, date_checker, check_palindrome]

# Pass the tools to the ToolNode()
tool_node = ToolNode(tools)

# Bind tools to the LLM
model_with_tools = llm.bind_tools(tools)
```

Let's practice!

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Defining nodes and edges for flexible function calling

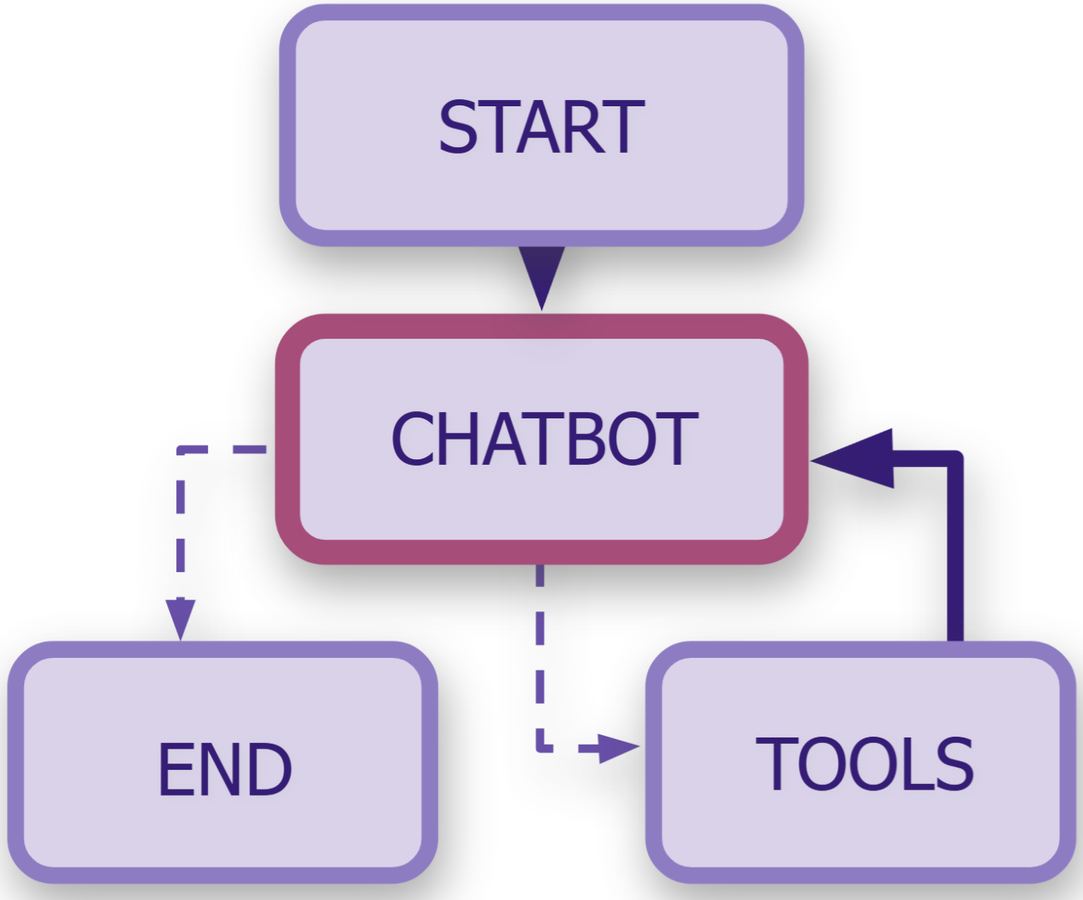
DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Dilini K. Sumanapala, PhD
Founder & AI Engineer, Genverv, Ltd.

datacamp

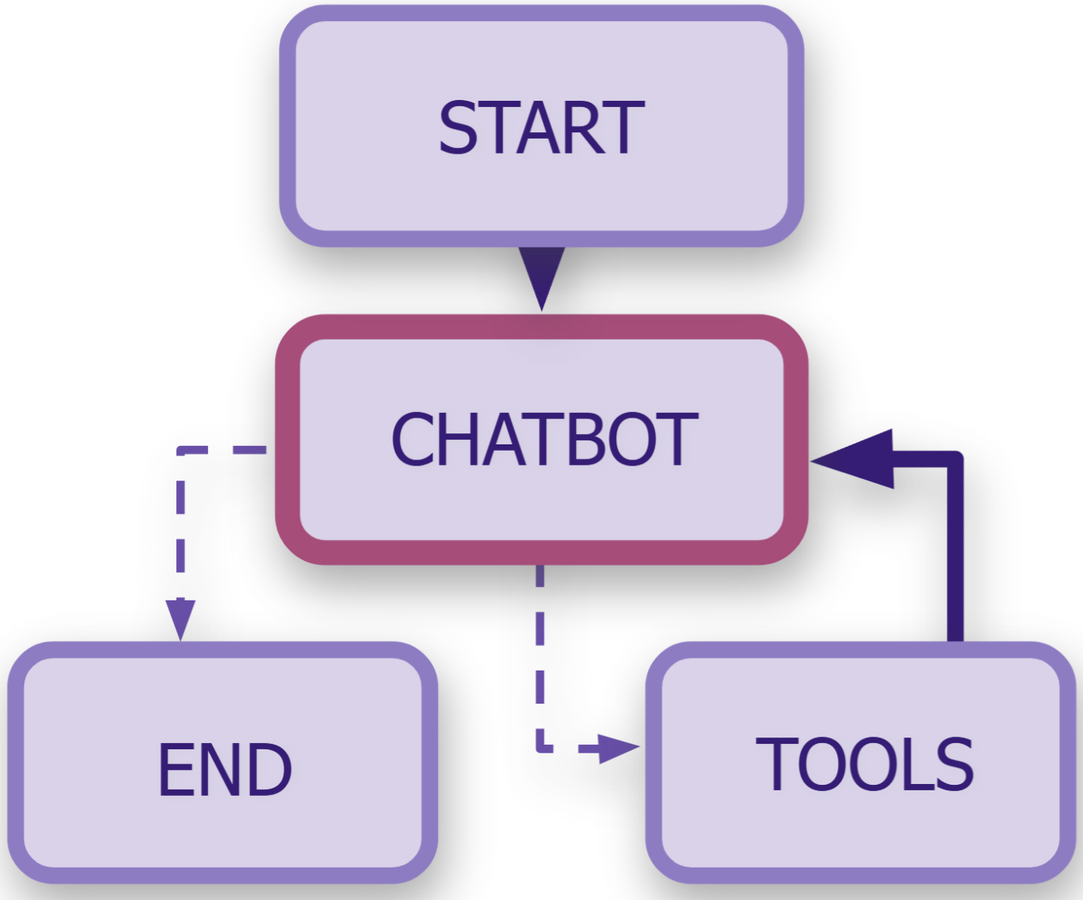
Building a multi-tool workflow

- Multiple available tools
  - Palindrome
  - Historical events
  - Wikipedia



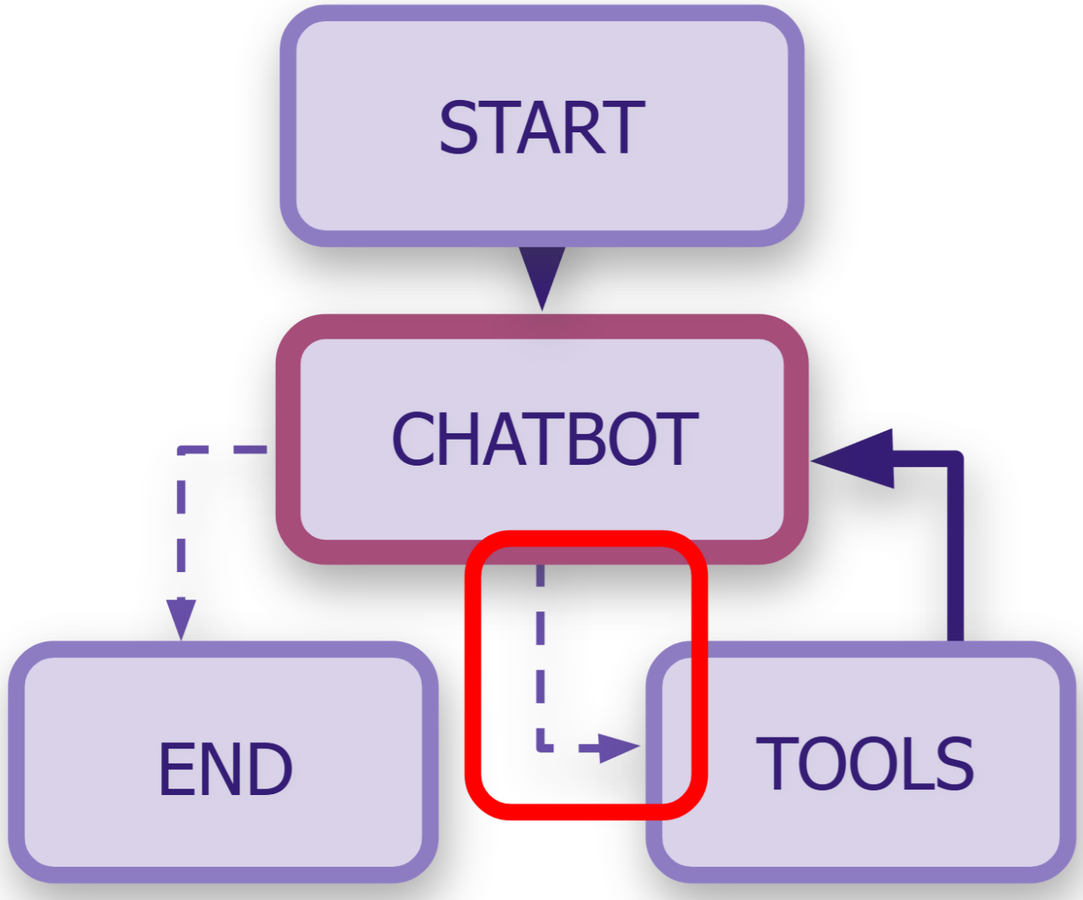
Define workflow functions

- Create a stopping function



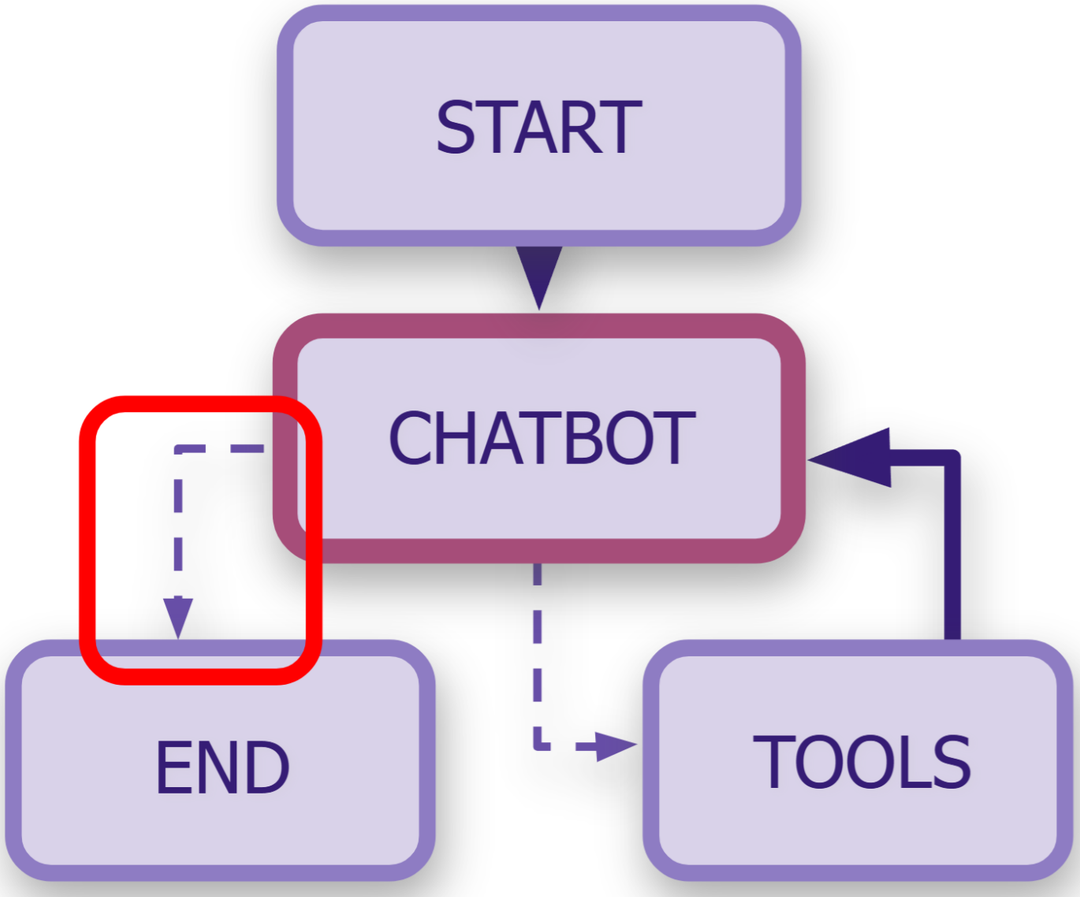
Define workflow functions

- **Create a stopping function**
  - Check for tool calls



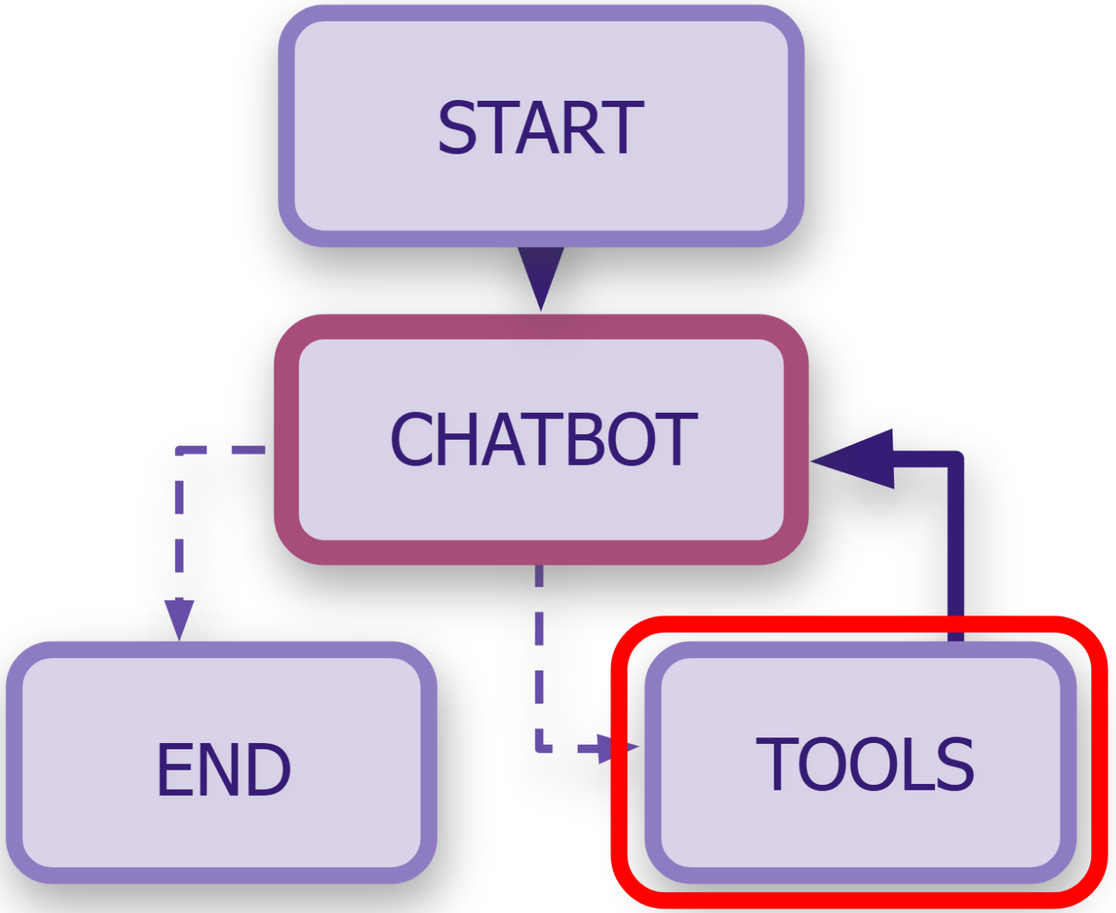
Define workflow functions

- **Create a stopping function**
  - Check for tool calls
  - End conversation if none



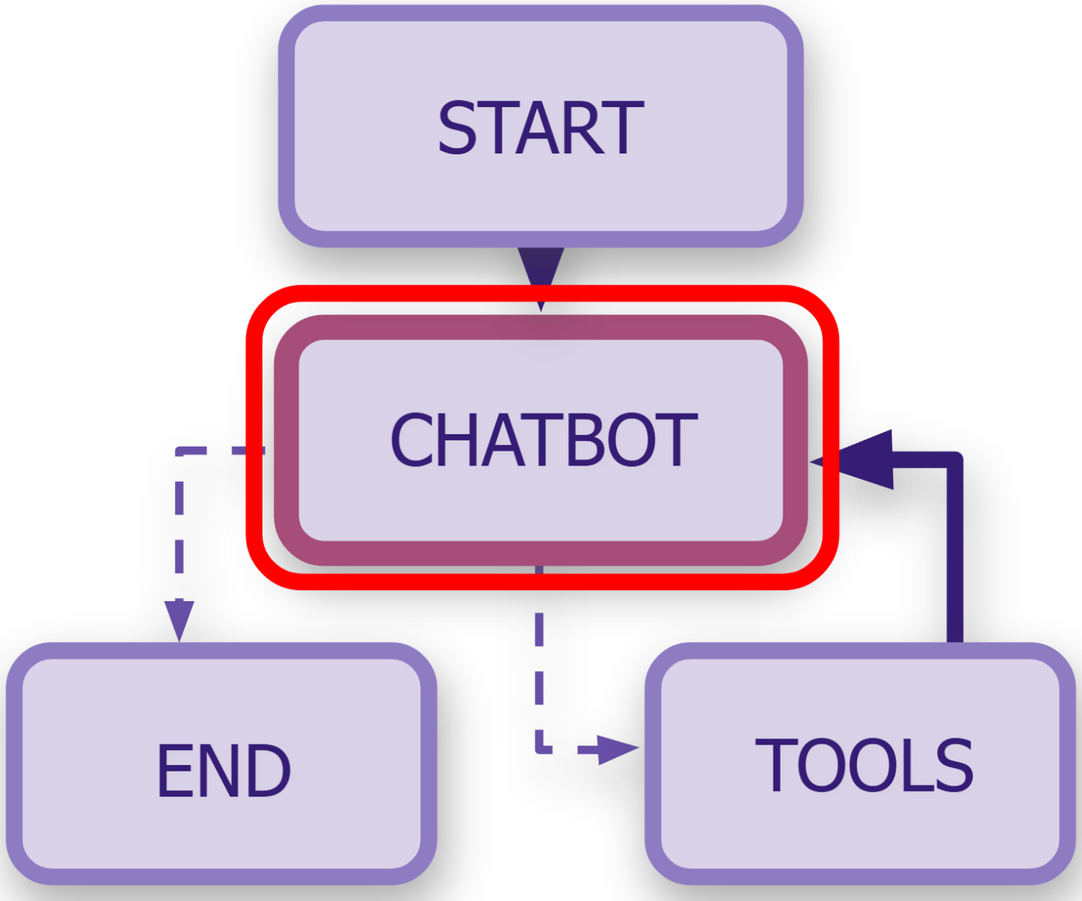
Define workflow functions

- **Create a stopping function**
  - Check for tool calls
  - End conversation if none
- **Create a dynamic tool caller**
  - Return a tool response if tool call present



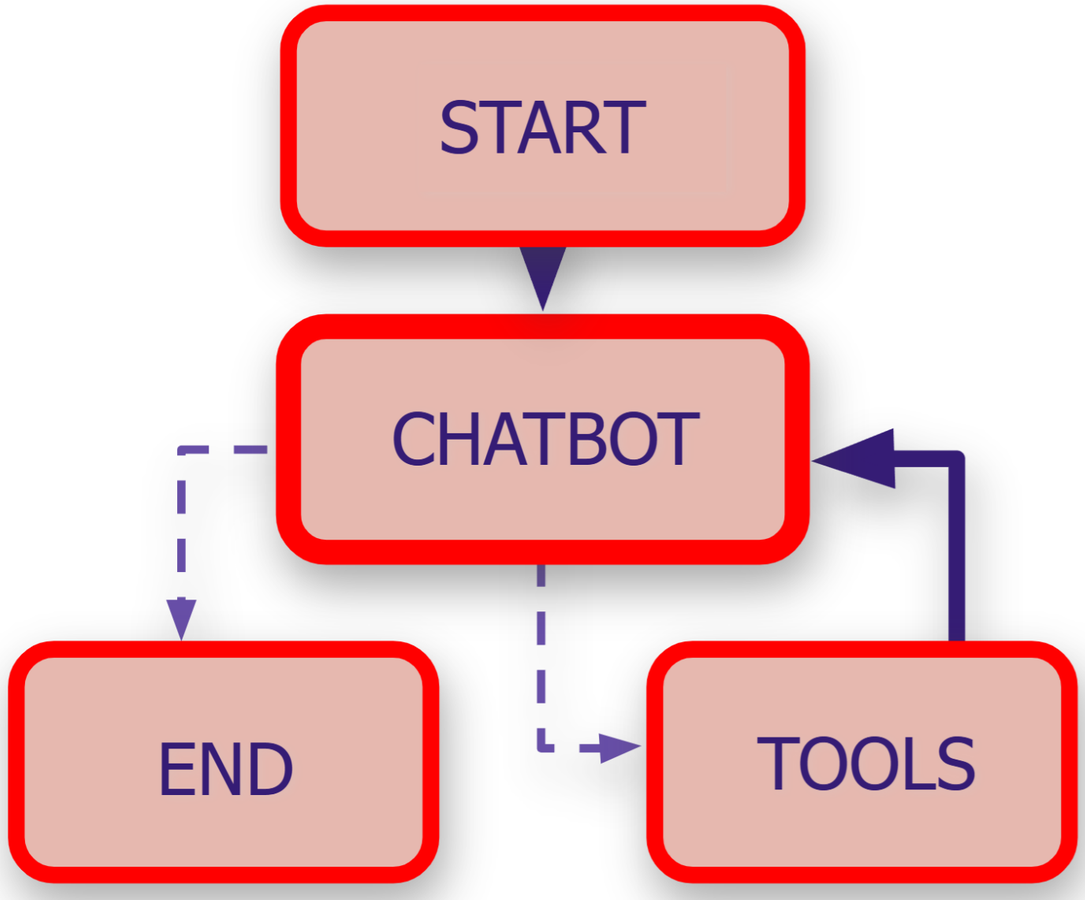
Define workflow functions

- **Create a stopping function**
  - Check for tool calls
  - End conversation if none
- **Create a dynamic tool caller**
  - Return a tool response if tool call present
  - Invoke the LLM with **just** the chatbot node if no tool calls



Define workflow functions

- **Create a stopping function**
  - Check for tool calls
  - End conversation if none
- **Create a dynamic tool caller**
  - Return a tool response if tool call present
  - Invoke the LLM with **just** the chatbot node if no tool calls
- **Compile the full graph**



Create a stop condition function

```
from langgraph.graph import MessagesState, START, END

# Use MessagesState to define the state of the stopping function
def should_continue(state: MessagesState):

    # Get the last message from the state
    last_message = state["messages"][-1]

    # Check if the last message includes tool calls
    if last_message.tool_calls:
        return "tools"

    # End the conversation if no tool calls are present
    return END
```

Create a dynamic tool caller

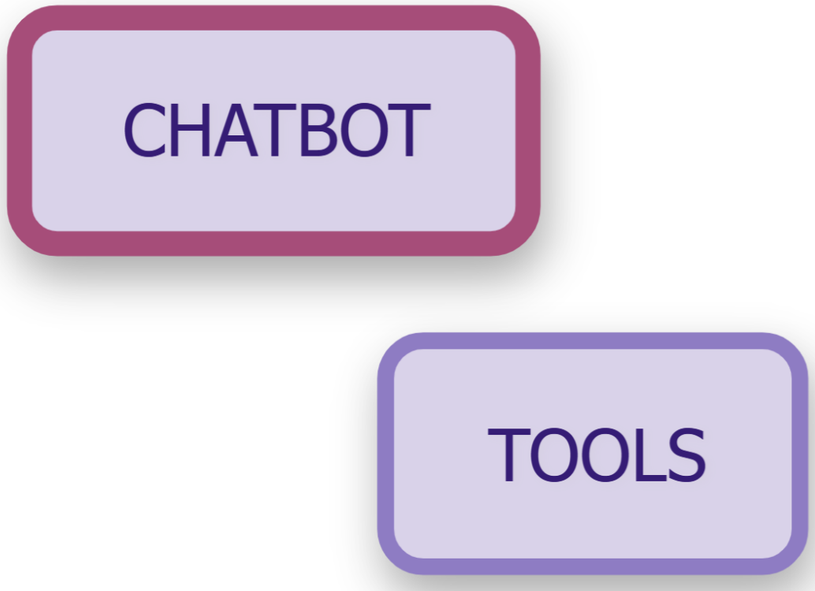
```
# Extract the last message from the history
def call_model(state: MessagesState):
    last_message = state["messages"][-1]

    # If the last message has tool calls, return the tool's response
    if isinstance(last_message, AIMessage) and last_message.tool_calls:

        # Return the messages from the tool call
        return {"messages": [AIMessage(content=last_message.tool_calls[0]["response"])]}

    # Otherwise, proceed with a regular LLM response
    return {"messages": [model_with_tools.invoke(state["messages"])]}
```

Create the graph

```
workflow = StateGraph(MessagesState)
```

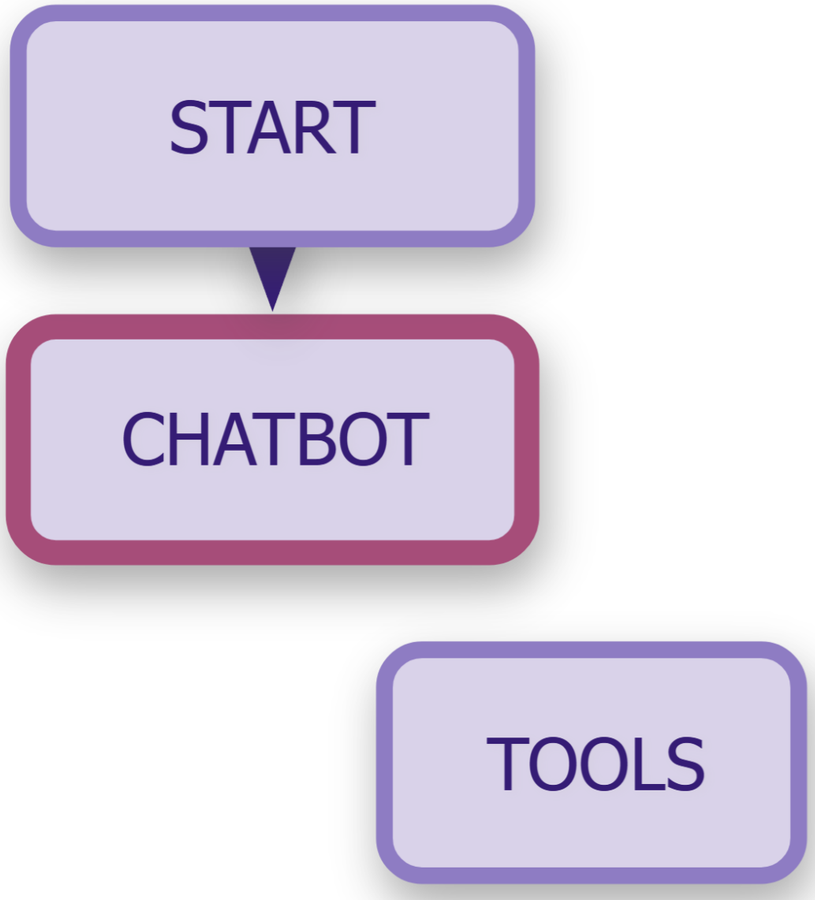
Create the graph

```
workflow = StateGraph(MessagesState)

# Add nodes for chatbot and tools
workflow.add_node("chatbot", call_model)
workflow.add_node("tools", tool_node)
```



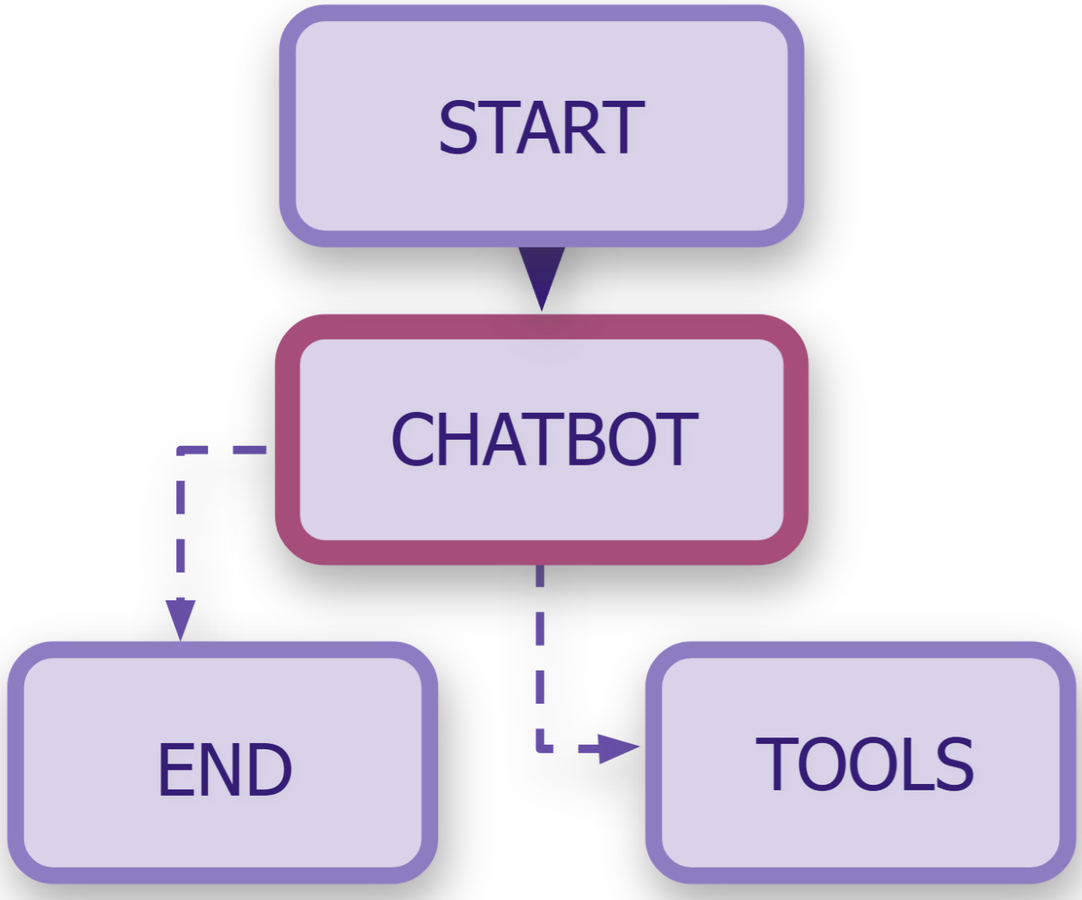
Create the graph

```
workflow = StateGraph(MessagesState)

# Add nodes for chatbot and tools
workflow.add_node("chatbot", call_model)
workflow.add_node("tools", tool_node)

# Connect the START node to the chatbot
workflow.add_edge(START, "chatbot")
```



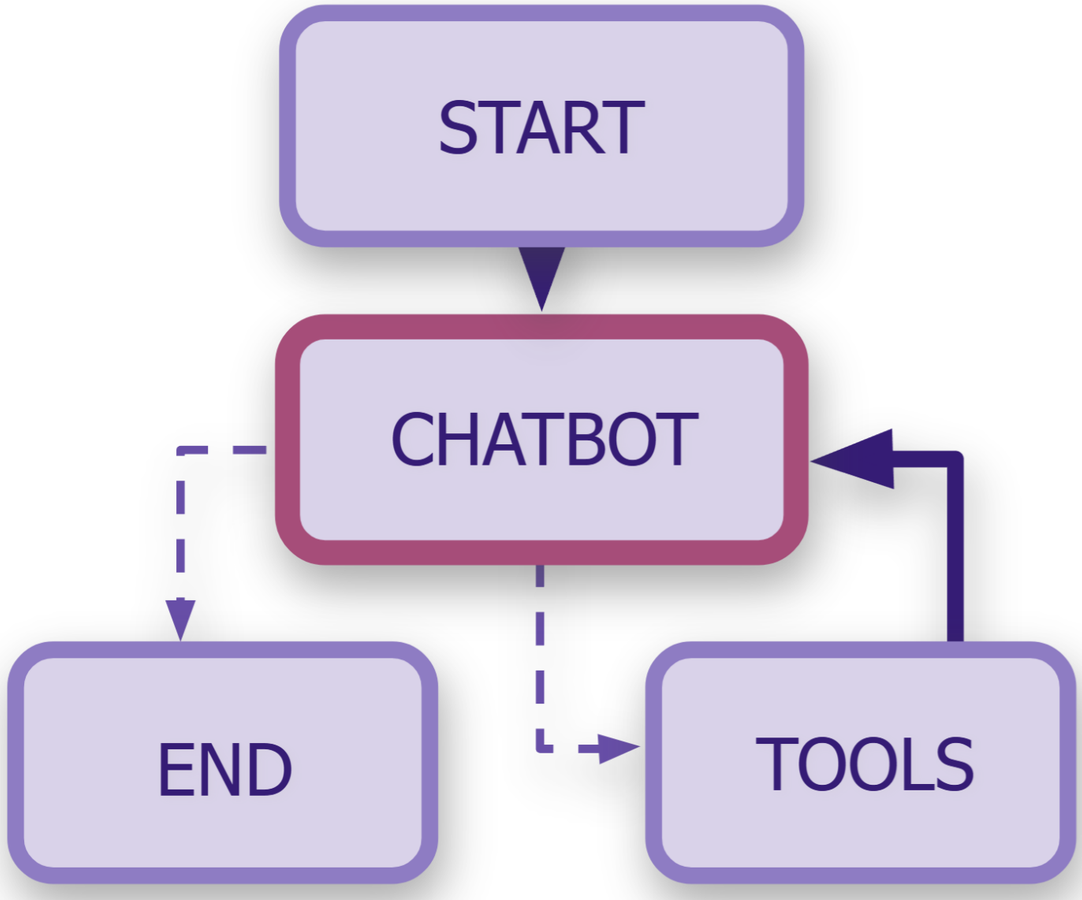
Create the graph

```
workflow = StateGraph(MessagesState)

# Add nodes for chatbot and tools
workflow.add_node("chatbot", call_model)
workflow.add_node("tools", tool_node)

# Connect the START node to the chatbot
workflow.add_edge(START, "chatbot")

# Define conditions, then loop back to chatbot
workflow.add_conditional_edges("chatbot",
         should_continue, ["tools", END])
```



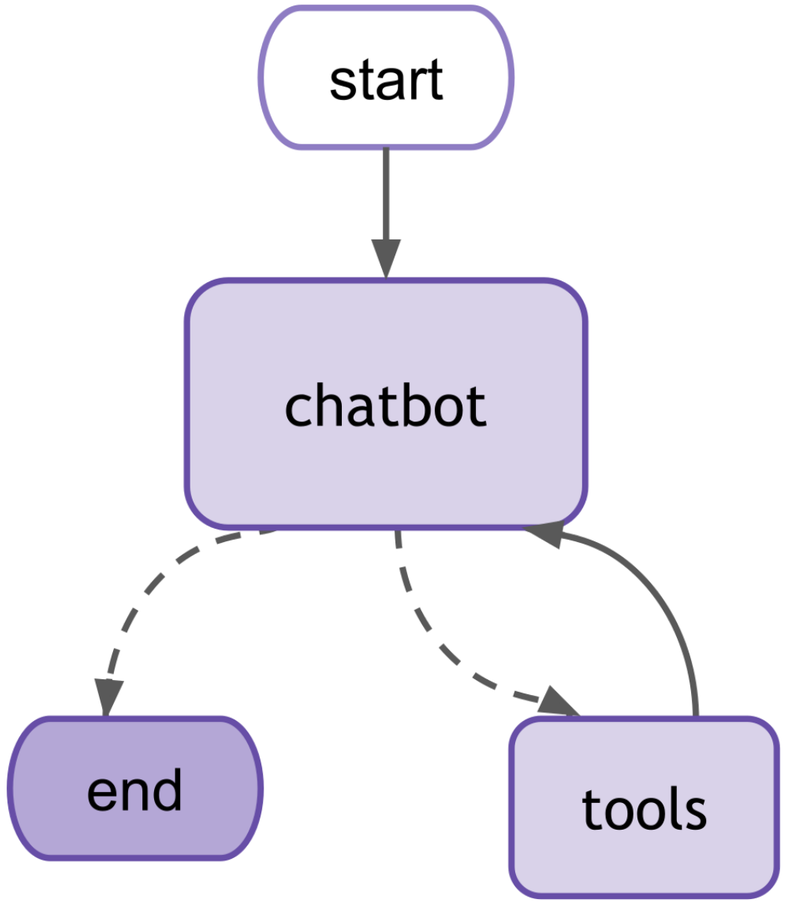
Create the graph

```
workflow = StateGraph(MessagesState)

# Add nodes for chatbot and tools
workflow.add_node("chatbot", call_model)
workflow.add_node("tools", tool_node)

# Connect the START node to the chatbot
workflow.add_edge(START, "chatbot")

# Define conditions, then loop back to chatbot
workflow.add_conditional_edges("chatbot",
         should_continue, ["tools", END])
workflow.add_edge("tools", "chatbot")
```



Adding memory

```
# Set up memory and compile the workflow
memory = MemorySaver()
app = workflow.compile(
                checkpointer=memory)

display(Image(app.get_graph()
        .draw_mermaid_png()))
```



Let's practice!

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

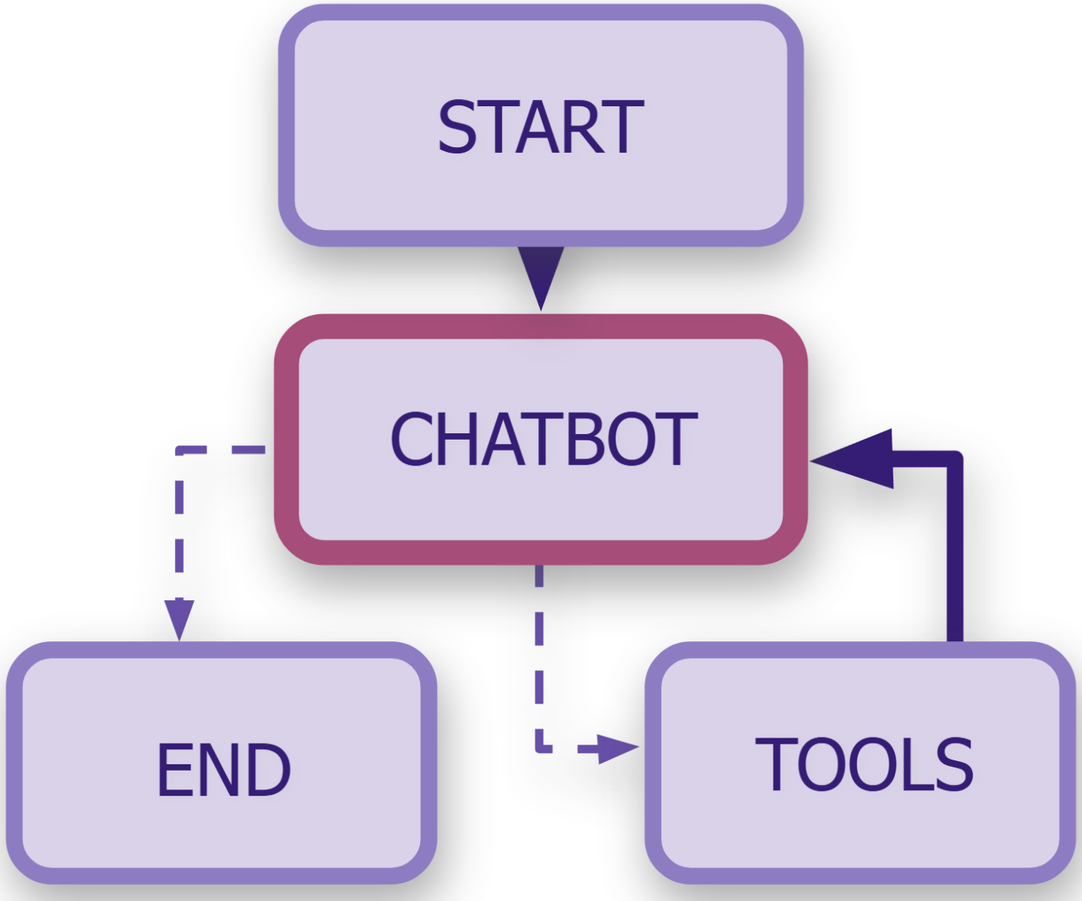
Organize chatbot outputs with memory

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Dilini K. Sumanapala, PhD
Founder & AI Engineer, Genverv, Ltd.

datacamp

Streaming multiple tool outputs



- **Print outputs with multiple tools**

```
- Agent: [Tool 1: answer]
- Agent: [Tool 2: answer]
```

- **Print user query and enable memory**

```
- User: [Query]
- Agent: [Tool 1: answer]

- User: [Follow-up query]
- Agent: [Tool 1: follow-up answer]
```

Streaming multiple tool outputs

```
from langchain_core.messages import AIMessage, HumanMessage

config = {"configurable": {"thread_id": "1"}}

# Create input message with the user's query
def multi_tool_output(query):
    inputs = {"messages": [HumanMessage(content=query)]}

    # Stream messages and metadata from the chatbot application
    for msg, metadata in app.stream(inputs, config, stream_mode="messages"):

        # Check if the message has content and is not from a human
        if msg.content and not isinstance(msg, HumanMessage):
            print(msg.content, end="", flush=True)
    print("\n")
```

Test with multiple tools

Check dynamic tool assignment

```
multi_tool_output("Is `Stella won no wallets` a palindrome?")
multi_tool_output("What happened on April 12th, 1955?")
```

```
The phrase or word 'Stella won no wallets' is a palindrome.
Yes, the phrase "Stella won no wallets" is a palindrome.

April 12th, 1955, is notable for several reasons, particularly in the context of science and
technology. On this date, the first successful polio vaccine developed by Dr. Jonas Salk
was announced to the public. This vaccine was a significant breakthrough in the fight against
poliomyelitis, a disease that had caused widespread fear and numerous outbreaks, particularly
affecting children. The announcement marked a turning point in public health and led to
widespread vaccination campaigns that ultimately contributed to the near-eradication of polio...
```

Follow-up questions with multiple tools

```
# Print the user query first for every interaction
def user_agent_multiturn(queries):
    for query in queries:
        print(f"User: {query}")

        # Stream through messages corresponding to queries, excluding metadata
        print("Agent: " + "".join(msg.content for msg, metadata in app.stream(
                {"messages": [HumanMessage(content=query)]}, config, stream_mode="messages")

            # Filter out the human messages to print agent messages
            if msg.content and not isinstance(msg, HumanMessage)) + "\n")

queries = ["What happened on the 12 April 1961?", "What about 10 December 1948?",
"Is `Mr. Owl ate my metal worm?` a palindrome?", "What about 'palladium stadium?'"]
user_agent_multiturn(queries)
```

Full conversation output

- Historical events

```
User: What happened on the 12 April 1961?
Agent: On April 12, 1961, Yuri Gagarin, a Soviet cosmonaut, became the first human to travel
into space aboard the Vostok 1 spacecraft...

User: What about 22 November 1963?
Agent: December 10, 1948... marks the Universal Declaration of Human Rights (UDHR).
```

- Palindrome check

```
User: Is `Mr. Owl ate my metal worm?` a palindrome?
Agent: The phrase or word 'Mr. Owl ate my metal worm?' is a palindrome...

User: What about 'palladium stadium'?
Agent: No, the phrase `palladium stadium` is not a palindrome...
```

Let's practice!

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Congratulations!

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN

Dilini K. Sumanapala, PhD
Founder & AI Engineer, Genverv, Ltd.

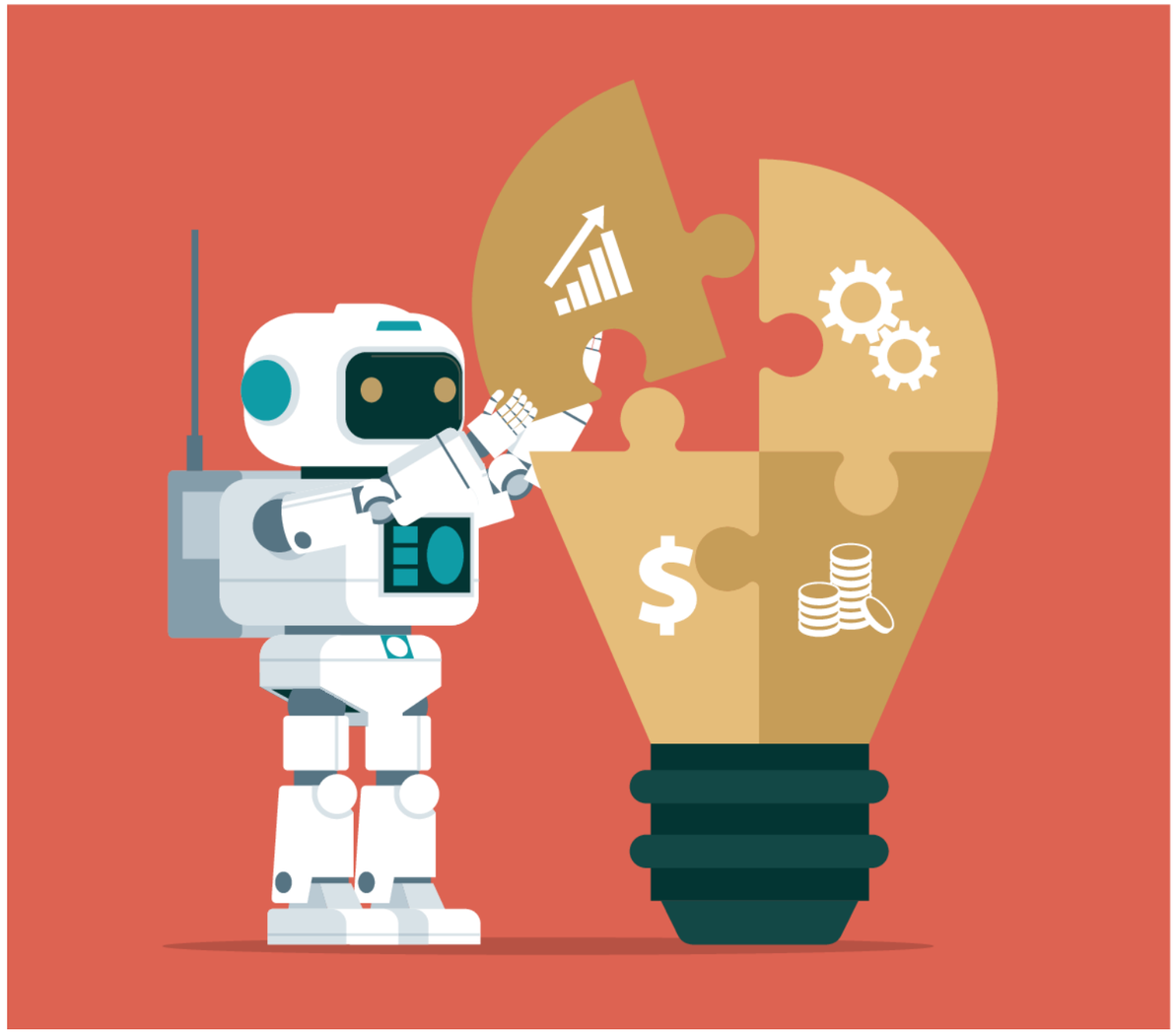
Essentials of LangChain agents

- Created ReAct agents
- Enhanced agents with tools
- Enabled follow-up questions
- Reasoning and LLMs
- Natural language responses

Building chatbots with LangGraph

- **Integrate external APIs**
  - eg. Wikipedia
- **Define**
  - Graph and agent states
  - Nodes
  - Edges
- **Memory**
- **Conversation**

Building dynamic chatbots



Advanced features

- Switching between tools
- Dynamic LLM calls
- Create multiple tools
- Flexible workflows
- Multiple tool memory
- Multi-turn conversation
- Large workloads, eg.
  - School curriculum

The LangChain ecosystem

- AI workflows
  - Scalable
  - Flexible
- Building foundational agents
- Integrating tools
- Advanced memory

The LangChain ecosystem

- **Real world settings**
  - **LangSmith**
    debugging and evaluation
  - **LangGraph**
    agent customization
  - **LangGraph Platform**
    agent deployment
- **Documentation**
  - LangChain products
  - LangGraph and LangGraph Platform

Thank you!

DESIGNING AGENTIC SYSTEMS WITH LANGCHAIN