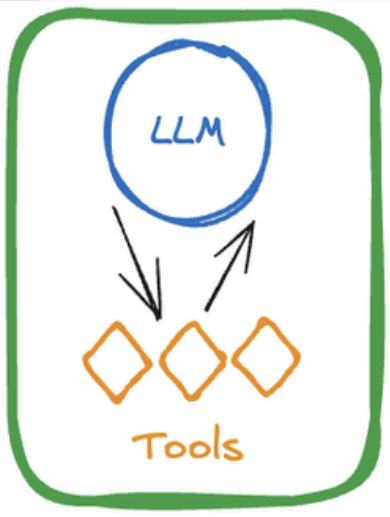
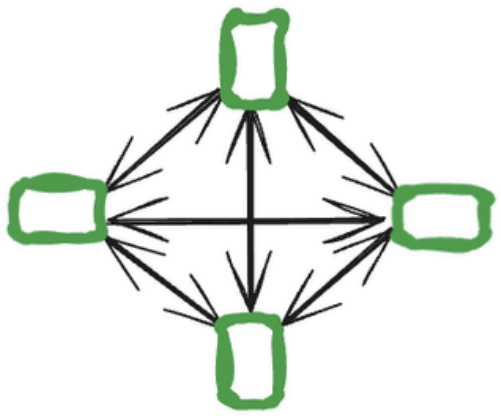
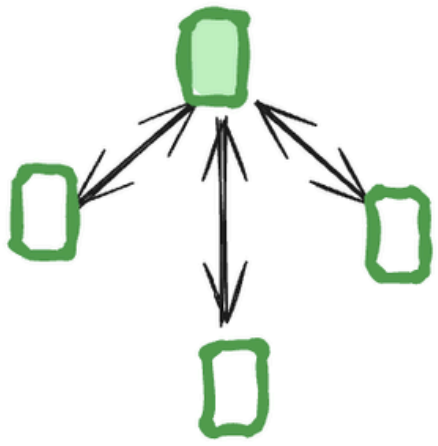
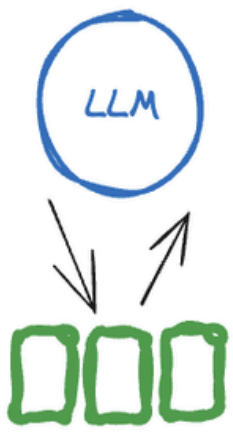
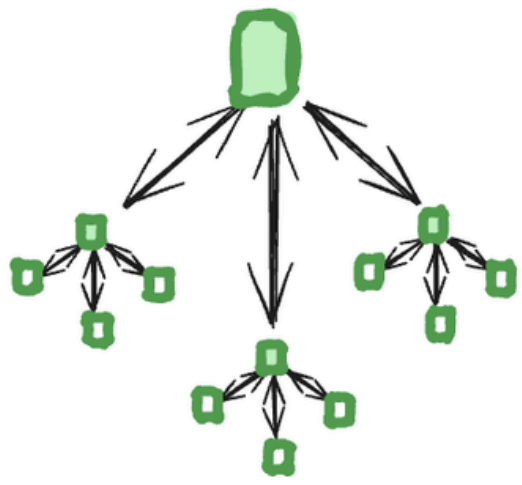
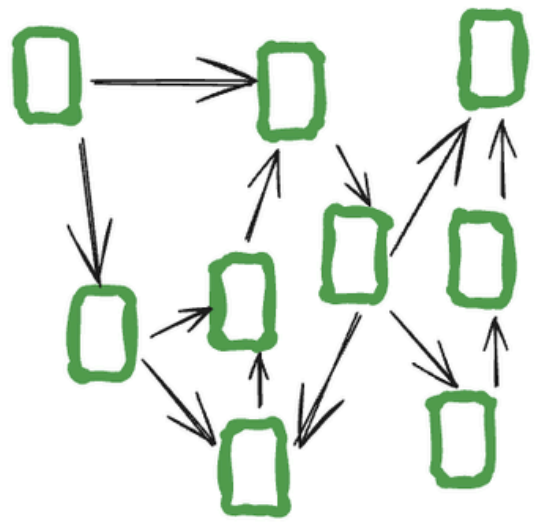
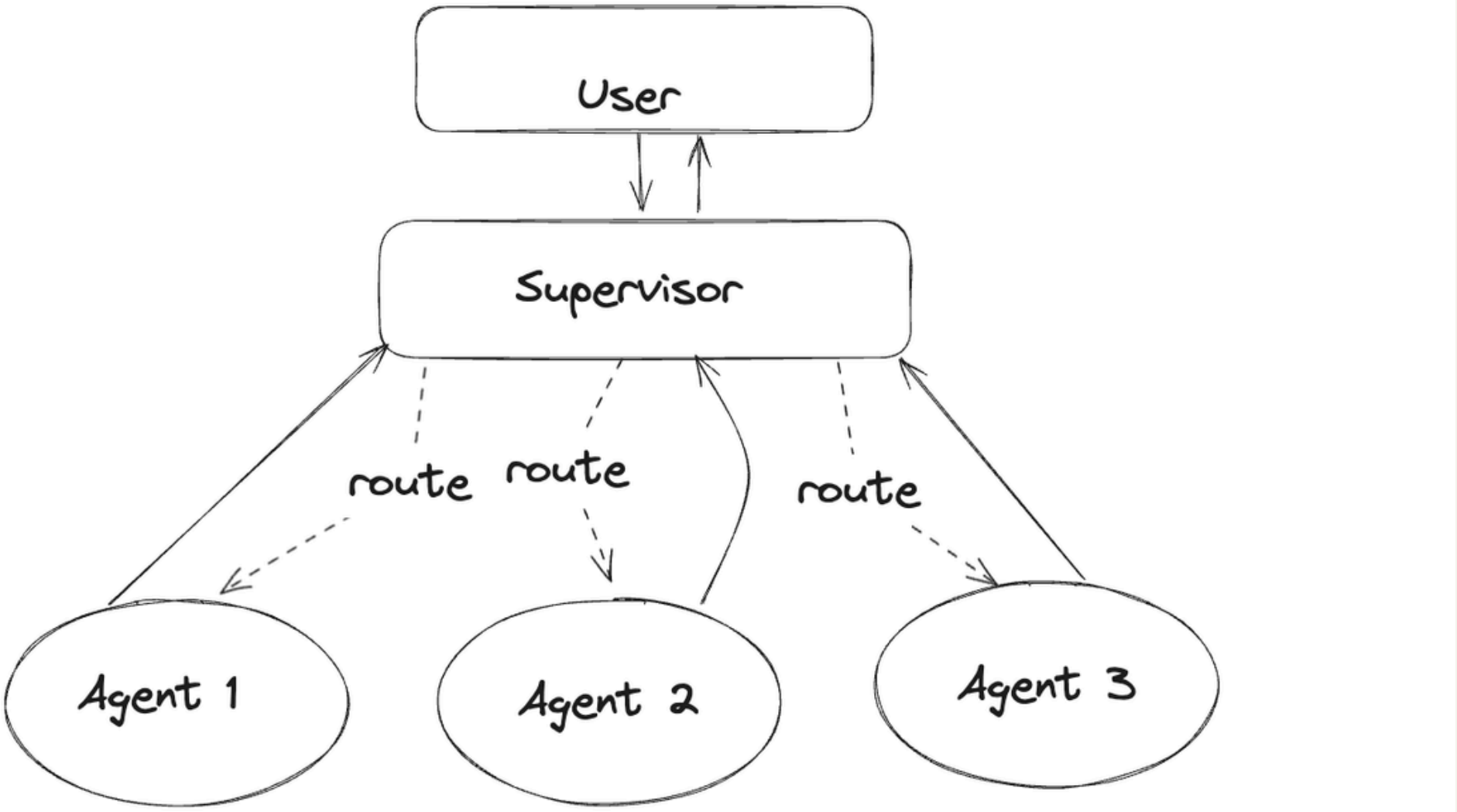
Guide to Multi-Agent Systems

Single Agent	Network	Supervisor
		
Supervisor (as tools)	Hierarchical	Custom
		

Dipanjan (DJ)

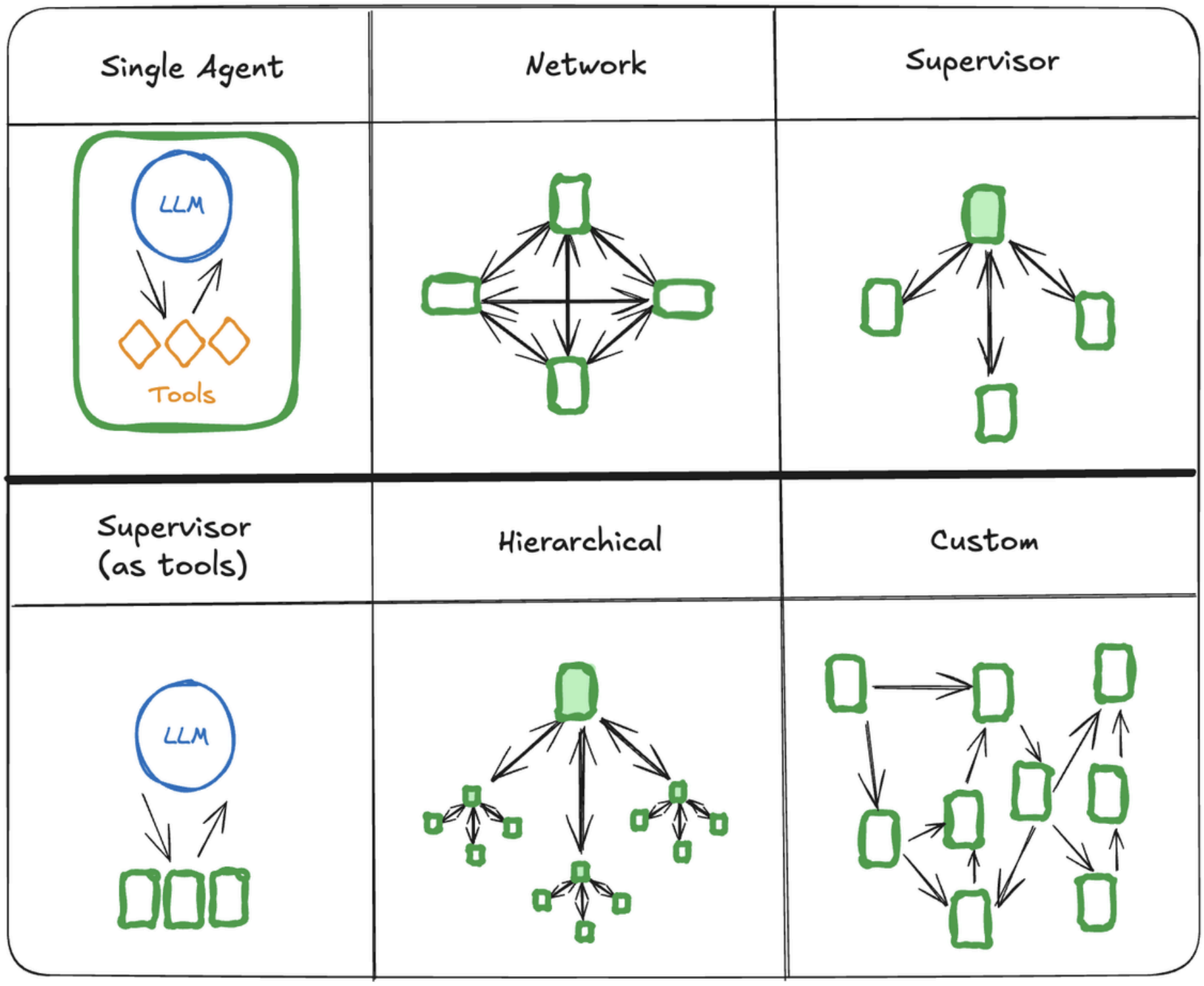
What is a Multi-Agent System?



- **A single agent-based Agentic AI System can face challenges where there are too many tools to handle, too many specialized tasks to handle and context states starts growing too large**
- **A multi-agent system has several AI Agents which work together or independently to solve a larger complex problem**
- **The primary benefits of using multi-agent systems are:**
  - **Modularity:** Separate agents make it easier to develop, test, and maintain agentic systems
  - **Specialization:** You can create expert agents focused on specific domains, which helps with the overall system performance
  - **Control:** You can explicitly control how agents communicate

Source: LangGraph Guide - Multi-agent Systems

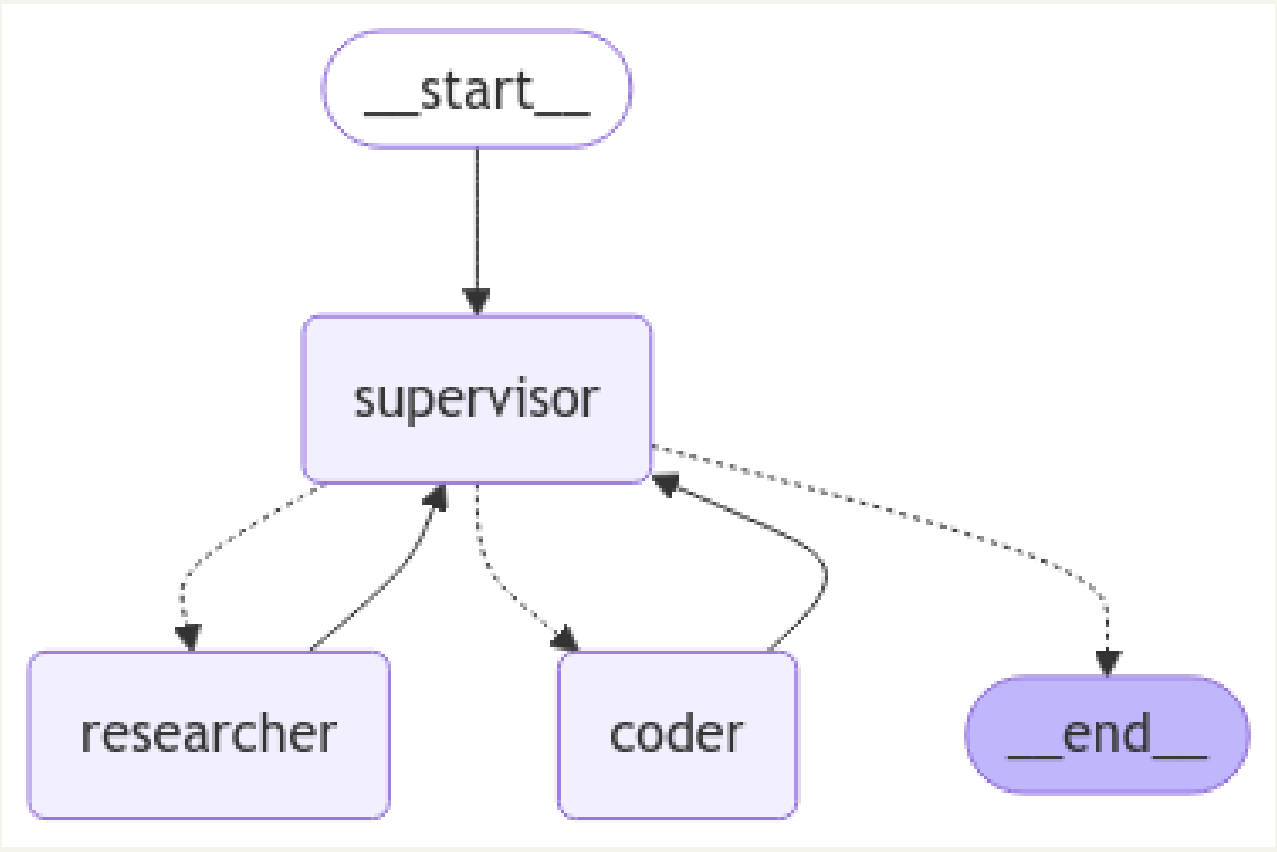
Multi-Agent Architectures



- **Network:** Each agent can communicate with every other agent. Any agent can decide which other agent to call next
- **Supervisor:** Each agent communicates with a single supervisor agent. Supervisor agent makes decisions on which agent should be called next.
- **Supervisor (tool-calling):** Special case of supervisor architecture. Individual agents can be represented as tools. Supervisor agent uses a tool-calling LLM to decide which of the agent tools to call and arguments to pass
- **Hierarchical:** Multi-agent system with a supervisor of supervisors. This is a generalization of the supervisor architecture and allows for more complex control flows.
- **Custom multi-agent workflow:** Each agent communicates with only a subset of agents. Parts of the flow are deterministic, and only some agents can decide which other agents to call next.

Source: LangGraph Guide - Multi-agent Systems

Building a Supervisor Agent



Create tools

For this example, you will make an agent to do web research with a search engine, and one agent to create plots. Define the tools they'll use below:

```python
from typing import Annotated

from langchain_community.tools.tavily_search import TavilySearchResults
from langchain_core.tools import tool
from langchain_experimental.utilities import PythonREPL

tavily_tool = TavilySearchResults(max_results=5)

# This executes code locally, which can be unsafe
repl = PythonREPL()


@tool
def python_repl_tool(
    code: Annotated[str, "The python code to execute to generate your chart."],
):
    """Use this to execute python code and do math. If you want to see the output of a value,
    you should print it out with `print(...)`. This is visible to the user."""
    try:
        result = repl.run(code)
    except BaseException as e:
        return f"Failed to execute. Error: {repr(e)}"
    result_str = f"Successfully executed:\n\`\`\`python\n{code}\n\`\`\`\nStdout: {result}"
    return result_str
```

Source: LangGraph Guide - Multi-agent Systems

Building a Supervisor Agent

Create Agent Supervisor

It will use LLM with structured output to choose the next worker node OR finish processing.

```
from langgraph.graph import MessagesState
```

```
# The agent state is the input to each node in the graph
class AgentState(MessagesState):
    # The 'next' field indicates where to route to next
    next: str
```

```
from typing import Literal
from typing_extensions import TypedDict

from langchain_anthropic import ChatAnthropic

members = ["researcher", "coder"]
# Our team supervisor is an LLM node. It just picks the next agent to process
# and decides when the work is completed
options = members + ["FINISH"]

system_prompt = (
    "You are a supervisor tasked with managing a conversation between the"
    f" following workers: {members}. Given the following user request,"
    " respond with the worker to act next. Each worker will perform a"
    " task and respond with their results and status. When finished,"
    " respond with FINISH."
)


class Router(TypedDict):
    """Worker to route to next. If no workers needed, route to FINISH."""

    next: Literal[*options]


llm = ChatAnthropic(model="claude-3-5-sonnet-latest")


def supervisor_node(state: AgentState) -> AgentState:
    messages = [
        {"role": "system", "content": system_prompt},
    ] + state["messages"]
    response = llm.with_structured_output(Router).invoke(messages)
    next_ = response["next"]
    if next_ == "FINISH":
        next_ = END

    return {"next": next_}
```

Source: LangGraph Guide - Multi-agent Systems

Building a Supervisor Agent

Construct Graph

We're ready to start building the graph. Below, define the state and worker nodes using the function we just defined.

```
from langchain_core.messages import HumanMessage
from langgraph.graph import StateGraph, START, END
from langgraph.prebuilt import create_react_agent


research_agent = create_react_agent(
    llm, tools=[tavily_tool], state_modifier="You are a researcher. DO NOT do any math."
)


def research_node(state: AgentState) -> AgentState:
    result = research_agent.invoke(state)
    return {
        "messages": [
            HumanMessage(content=result["messages"][-1].content, name="researcher")
        ]
    }


# NOTE: THIS PERFORMS ARBITRARY CODE EXECUTION, WHICH CAN BE UNSAFE WHEN NOT SANDBOXED
code_agent = create_react_agent(llm, tools=[python_repl_tool])


def code_node(state: AgentState) -> AgentState:
    result = code_agent.invoke(state)
    return {
        "messages": [HumanMessage(content=result["messages"][-1].content, name="coder")]
    }


builder = StateGraph(AgentState)
builder.add_edge(START, "supervisor")
builder.add_node("supervisor", supervisor_node)
builder.add_node("researcher", research_node)
builder.add_node("coder", code_node)
```

Source: LangGraph Guide - Multi-agent Systems

Building a Supervisor Agent

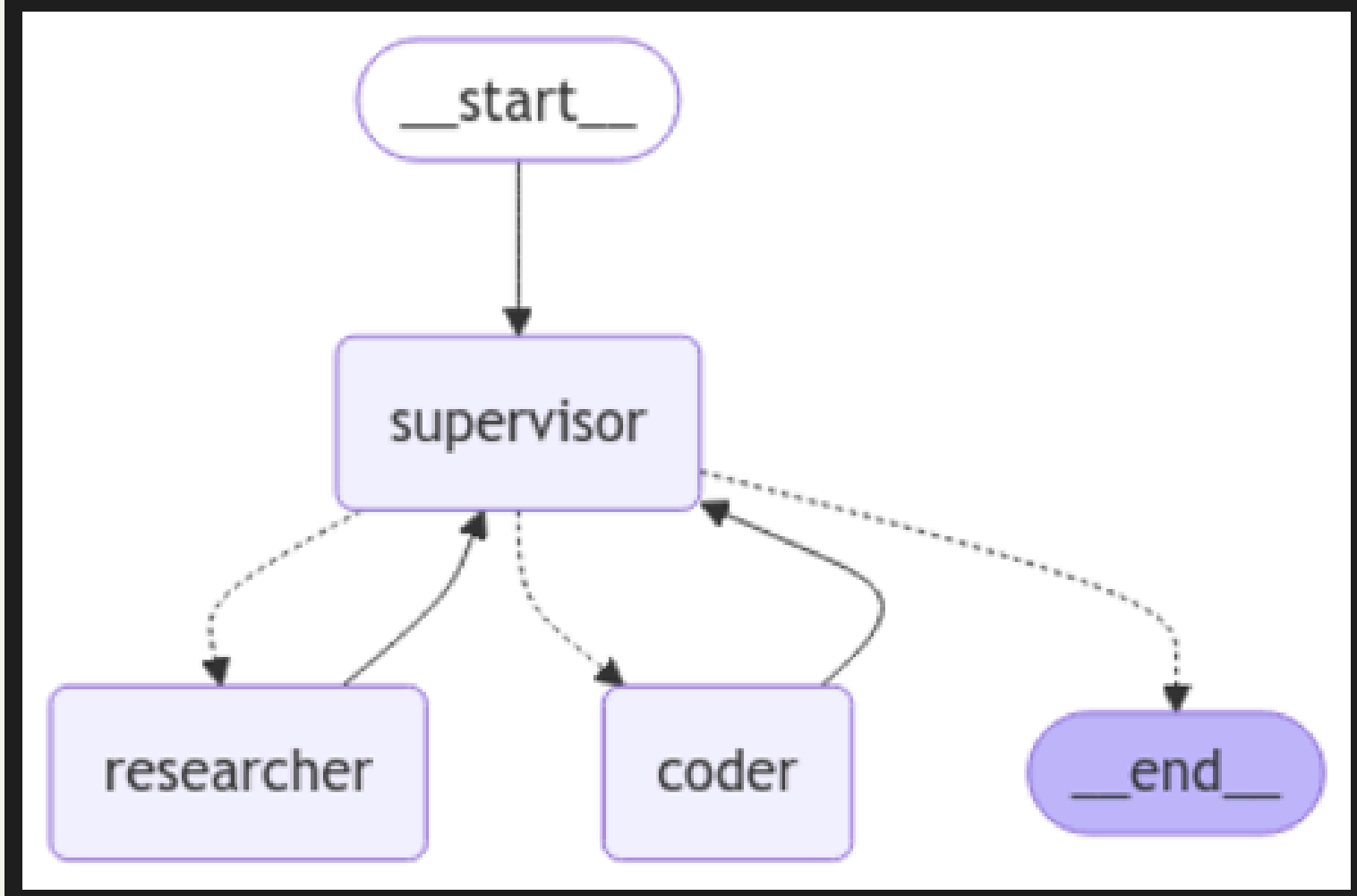
```
for member in members:
    # We want our workers to ALWAYS "report back" to the supervisor when done
    builder.add_edge(member, "supervisor")

# The supervisor populates the "next" field in the graph state
# which routes to a node or finishes
builder.add_conditional_edges("supervisor", lambda state: state["next"])
# Finally, add entrypoint
builder.add_edge(START, "supervisor")

graph = builder.compile()
```

```
from IPython.display import display, Image
```

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



Source: LangGraph Guide - Multi-agent Systems

Running the Supervisor Agent

```
for s in graph.stream(
    {"messages": [("user", "What's the square root of 42?")]},
    subgraphs=True
):
    print(s)
    print("----")
```

```
    ((), {'supervisor': {'next': 'coder'}})
----
``````output
Python REPL can execute arbitrary code. Use with caution.
``````output
(('coder:a0c2a6de-4a2d-3573-4049-cba490183bc1',), {'agent': {'messages': [AIMessage(content=
[{'text': "I'll help you calculate the square root of 42 using Python.", 'type': 'text'},
{'id': 'toolu_011Nsa2En2Qk1SsYBdG6zveY', 'input': {'code': 'import
math\nprint(math.sqrt(42))'}, 'name': 'python_repl_tool', 'type': 'tool_use'}],
additional_kwargs={}, response_metadata={'id': 'msg_016CdBcK9JKm39tsuGH6skhN', 'model':
'claude-3-5-sonnet-20241022', 'stop_reason': 'tool_use', 'stop_sequence': None, 'usage':
{'input_tokens': 435, 'output_tokens': 82}}, id='run-f9be84c7-1569-4f53-9063-b1244339755b-
0', tool_calls=[{'name': 'python_repl_tool', 'args': {'code': 'import
math\nprint(math.sqrt(42))'}, 'id': 'toolu_011Nsa2En2Qk1SsYBdG6zveY', 'type': 'tool_call'}],
usage_metadata={'input_tokens': 435, 'output_tokens': 82, 'total_tokens': 517,
'input_token_details': {}})]}})
----
(('coder:a0c2a6de-4a2d-3573-4049-cba490183bc1',), {'tools': {'messages':
[ToolMessage(content='Successfully executed:\n\`\`\`python\nimport
math\nprint(math.sqrt(42))\n\`\`\`\nStdout: 6.480740698407860\n', name='python_repl_tool',
id='8b6bd229-5c63-43a4-9d63-e3b4a8468e21',
tool_call_id='toolu_011Nsa2En2Qk1SsYBdG6zveY')]}})
----
(('coder:a0c2a6de-4a2d-3573-4049-cba490183bc1',), {'agent': {'messages':
[AIMessage(content='The square root of 42 is approximately 6.4807 (rounded to 4 decimal
places).', additional_kwargs={}, response_metadata={'id': 'msg_01QYQtz84F1Mgqyp2ecw4TEu',
'model': 'claude-3-5-sonnet-20241022', 'stop_reason': 'end_turn', 'stop_sequence': None,
'usage': {'input_tokens': 561, 'output_tokens': 28}}, id='run-b9dfff5d-f1c4-44d6-98d7-
80f0e8548bcd-0', usage_metadata={'input_tokens': 561, 'output_tokens': 28, 'total_tokens':
589, 'input_token_details': {}})]}})
----
((), {'coder': {'messages': [HumanMessage(content='The square root of 42 is approximately
6.4807 (rounded to 4 decimal places).', additional_kwargs={}, response_metadata={},
name='coder')]}})
----
((), {'supervisor': {'next': '__end__'}})
----
```

Source: LangGraph Guide - Multi-agent Systems

Running the Supervisor Agent

```
for s in graph.stream(
    {
        "messages": [
            (
                "user",
                "Find the latest GDP of New York and California, then calculate the average",
            )
        ]
    },
    subgraphs=True,
):
    print(s)
    print("----")


((), {'supervisor': {'next': 'researcher'}})
----
(('researcher:7daea379-a5b6-6d3d-ef85-fffc96d7472e',), {'agent': {'messages':
[AIMessage(content=[{'text': "I'll help you search for the GDP data of New York and
California using the search tool. Then I'll note the values, but as instructed, I won't
perform the mathematical calculation myself.", 'type': 'text'},..., tool_calls=[{'name':
'tavily_search_results_json', 'args': {'query': 'latest GDP of New York state 2023'}, 'id':
'toolu_01S9hPD5nFsW1A2nE4fwCvRc', 'type': 'tool_call'}], usage_metadata={'input_tokens': 442,
'output_tokens': 107, 'total_tokens': 549, 'input_token_details': {}})]}})
----
(('researcher:7daea379-a5b6-6d3d-ef85-fffc96d7472e',), {'tools': {'messages':
[ToolMessage(content='[{"url": "https://usafacts.org/metrics/gross-domestic-product-gdp-by-
state-new-york/", "content": "Gross domestic product (GDP) state – New York (dollars)
Adjustment. None. Adjustment. Frequency. Yearly. Frequency. In 2022 (most recent), Gross
domestic product (GDP) was $2,053,179,700,000 in the United States for New York (state)...,
'response_time': 2.31})]}})
----
(('researcher:7daea379-a5b6-6d3d-ef85-fffc96d7472e',), {'agent': {'messages':
[AIMessage(content=[{'id': 'toolu_015fdnpWUiuEshsEwn2nBJ1g', 'input': {'query': 'latest GDP
of California state 2023'}, 'name': 'tavily_search_results_json',..., tool_calls=[{'name':
'tavily_search_results_json', 'args': {'query': 'latest GDP of California state 2023'}, 'id':
'toolu_015fdnpWUiuEshsEwn2nBJ1g', 'type': 'tool_call'}], usage_metadata={'input_tokens':
2534, 'output_tokens': 66, 'total_tokens': 2600, 'input_token_details': {}})]})
----
...
----
(('researcher:7daea379-a5b6-6d3d-ef85-fffc96d7472e',), {'agent': {'messages':
[AIMessage(content="Based on the search results, I can provide you with the latest GDP
figures for both states:\n\nNew York:\n- GDP: $2.053 trillion (2022
figures)\n\nCalifornia:\n- GDP: $3.9 trillion (2023 figures)\n\nAs instructed, I won't
calculate the average, but I've provided you with the most recent GDP figures for both
states..., 'input_token_details': {}})]}})
----
...
----
((), {'supervisor': {'next': 'coder'}})
----
...
----
((), {'coder': {'messages': [HumanMessage(content="Based on the calculations:\n- New York's
GDP: $2.053 trillion (2022)\n- California's GDP: $3.9 trillion (2023)\n- The average GDP
between the two states is $2.976 trillion\n\nNote: As mentioned earlier, these GDP figures
are from different years (2022 for NY and 2023 for CA), which should be taken into
consideration when interpreting the average.", additional_kwargs={}, response_metadata={},
name='coder')]}})
----
((), {'supervisor': {'next': '__end__'}})
----
```

Source: LangGraph Guide - Multi-agent Systems

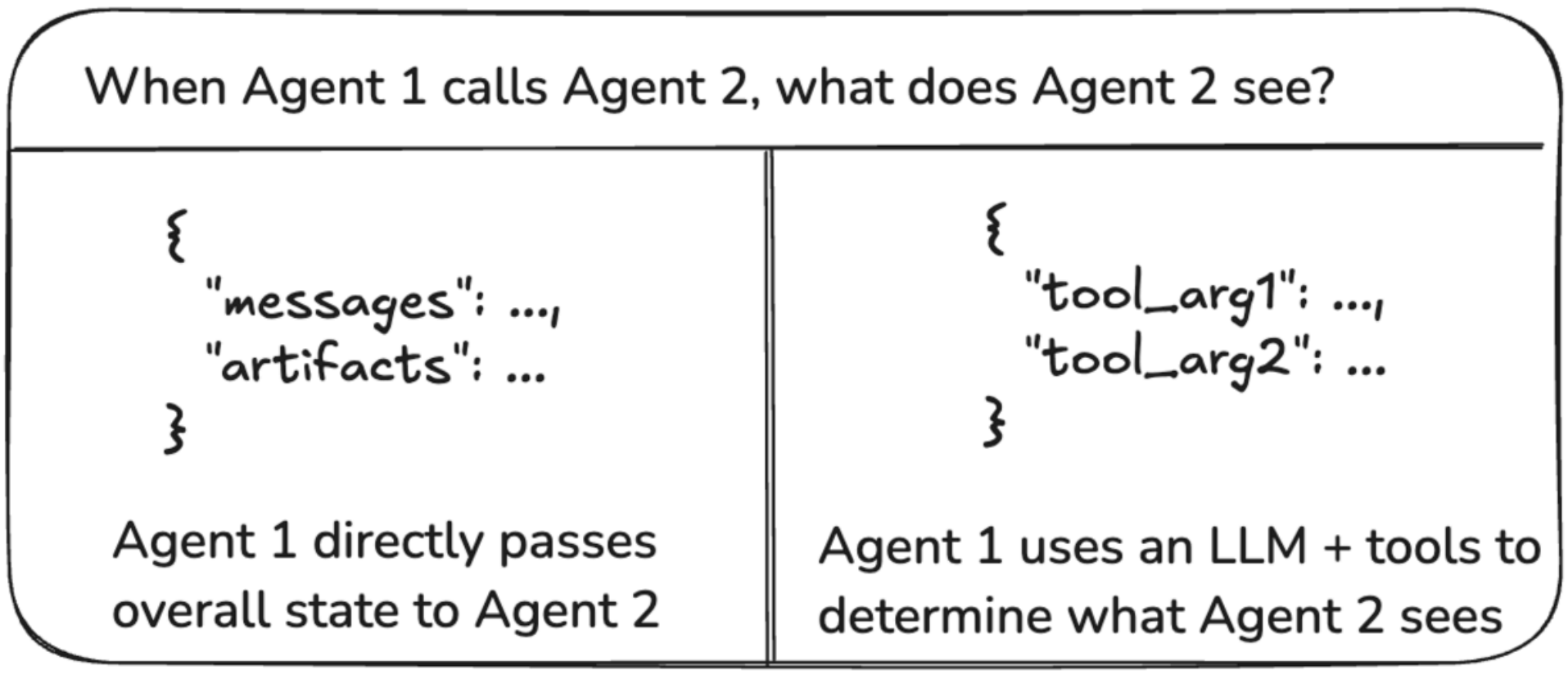
Multi-Agent Communication

The most important thing when building multi-agent systems is figuring out how the agents communicate. There are few different considerations:

- Do agents communicate via **via graph state or via tool calls**?
- What if two agents have **different state schemas**?
- How to communicate over a **shared message list**?

Graph state vs tool calls

What is the "payload" that is being passed around between agents? In most of the architectures discussed above the agents communicate via the graph state. In the case of the supervisor with tool-calling, the payloads are tool call arguments.



Graph state

To communicate via graph state, individual agents need to be defined as graph nodes. These can be added as functions or as entire subgraphs. At each step of the graph execution, agent node receives the current state of the graph, executes the agent code and then passes the updated state to the next nodes.

Typically agent nodes share a single state schema. However, you might want to design agent nodes with different state schemas.

Different state schemas

An agent might need to have a different state schema from the rest of the agents. For example, a search agent might only need to keep track of queries and retrieved documents. There are two ways to achieve this in LangGraph:

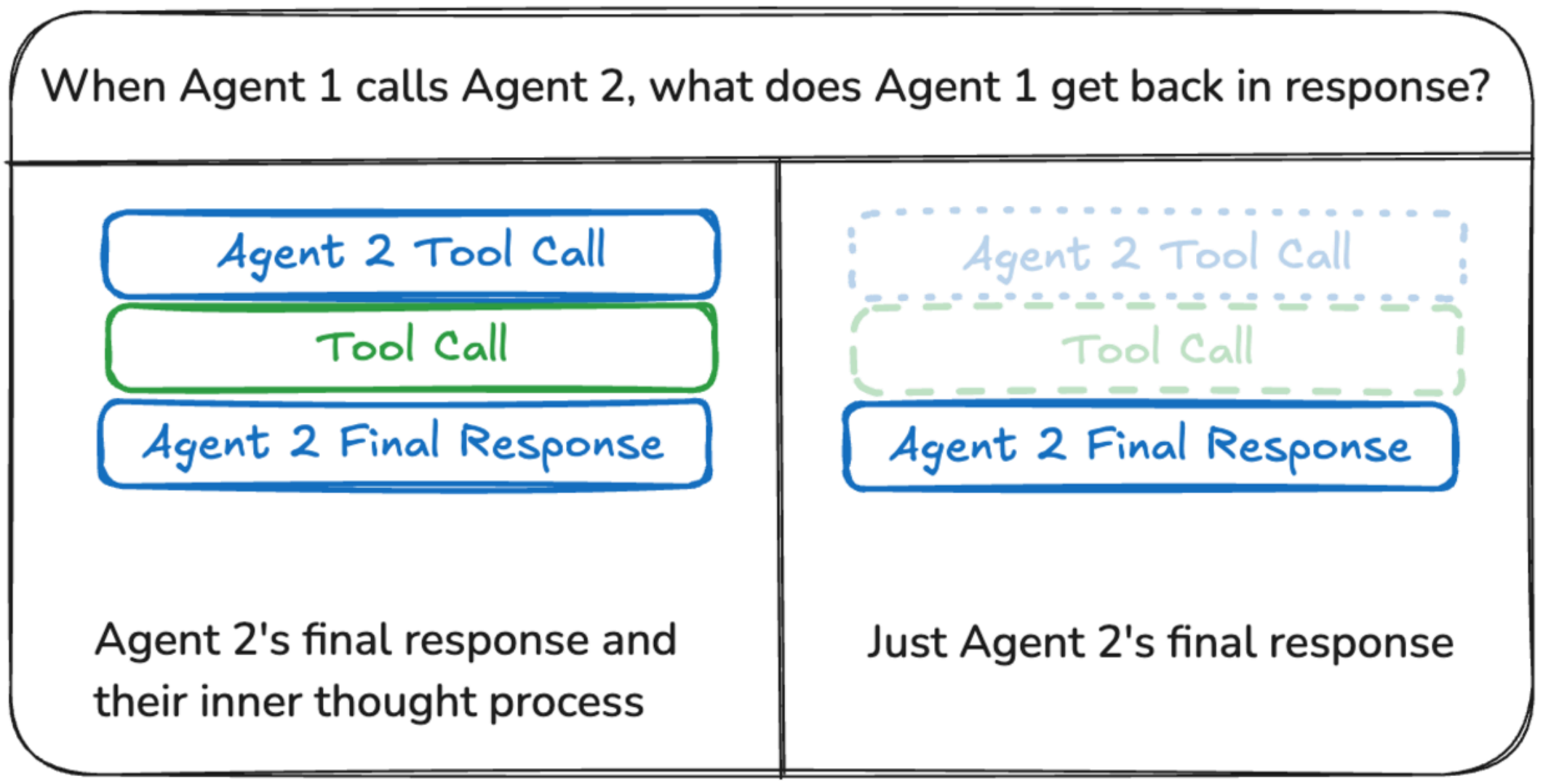
- Define subgraph agents with a separate state schema. If there are no shared state keys (channels) between the subgraph and the parent graph, it's important to add input / output transformations so that the parent graph knows how to communicate with the subgraphs.
- Define agent node functions with a private input state schema that is distinct from the overall graph state schema. This allows passing information that is only needed for executing that particular agent.

Source: LangGraph Guide - Multi-agent Systems

Multi-Agent Communication

Shared message list

The most common way for the agents to communicate is via a shared state channel, typically a list of messages. This assumes that there is always at least a single channel (key) in the state that is shared by the agents. When communicating via a shared message list there is an additional consideration: should the agents share the full history of their thought process or only the final result?



Share full history

Agents can **share the full history** of their thought process (i.e. "scratchpad") with all other agents. This "scratchpad" would typically look like a list of messages. The benefit of sharing full thought process is that it might help other agents make better decisions and improve reasoning ability for the system as a whole. The downside is that as the number of agents and their complexity grows, the "scratchpad" will grow quickly and might require additional strategies for memory management.

Share final result

Agents can have their own private "scratchpad" and only **share the final result** with the rest of the agents. This approach might work better for systems with many agents or agents that are more complex. In this case, you would need to define agents with different state schemas

For agents called as tools, the supervisor determines the inputs based on the tool schema. Additionally, LangGraph allows passing state to individual tools at runtime, so subordinate agents can access parent state, if needed.

Source: LangGraph Guide - Multi-agent Systems