Who Am I?

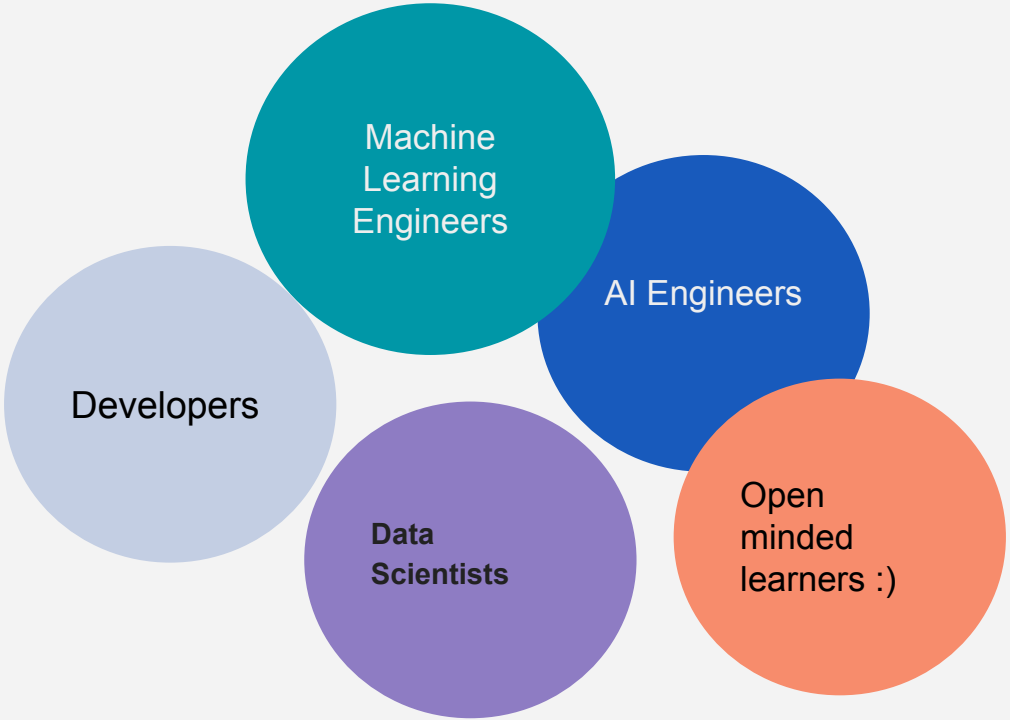
Paulo Dichone

Software, Cloud, AI Engineer
and Instructor

What Is This Course About?

- AutoGen
  - AutoGen Deep dive
    - Build and customize multi-agent systems
    - Agentic design patterns
  - Effectively implement multi-agent systems
  - Implement agents with different roles that collaborate to accomplish tasks
  - Best practices
  - Real-world, enterprise use cases

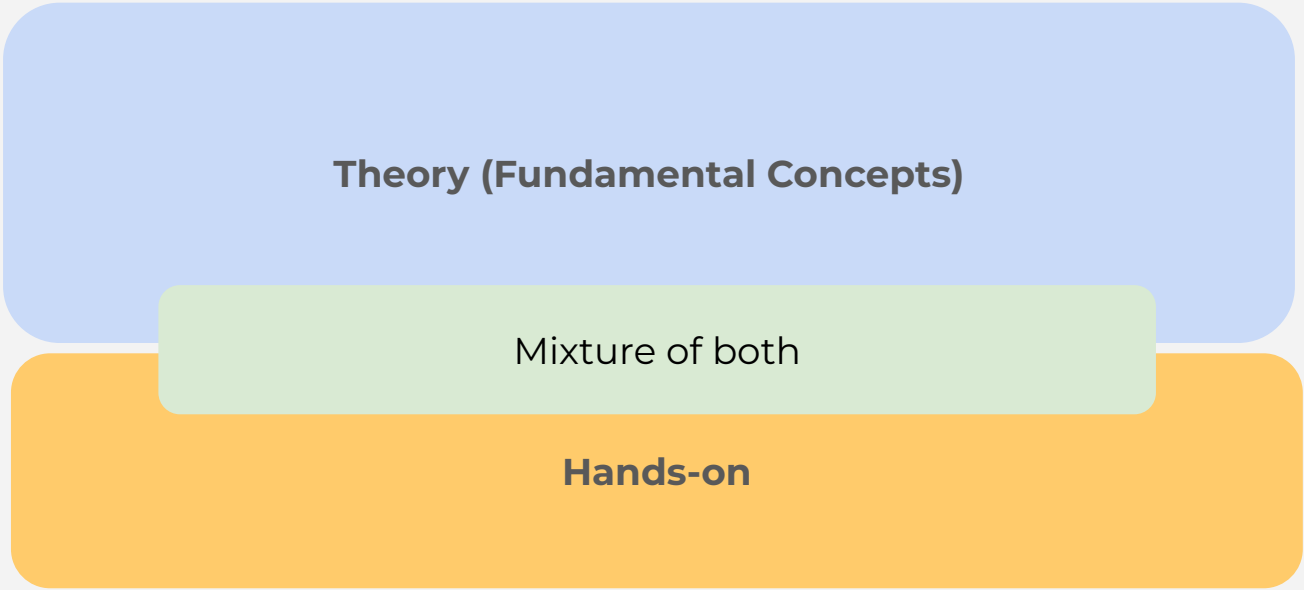
Who Is This Course For



Course Prerequisites

1. Know programming (highly *preferred… at least the basics)*
    a. *We will be using Python*
2. Basics of AI, ML, LLM
3. *This is not a programming course*
4. Willingness to learn :)

Course Structure



Development Environment setup

- Python
- VS Code (or any other code editor)
- OpenAI Account and an OpenAI API Key

Set up OpenAI API Account

*** Please note** that you will need an API key to use OpenAI services, and there may be some costs associated with using the API. However, these costs should be minimal.

Dev Environment Setup

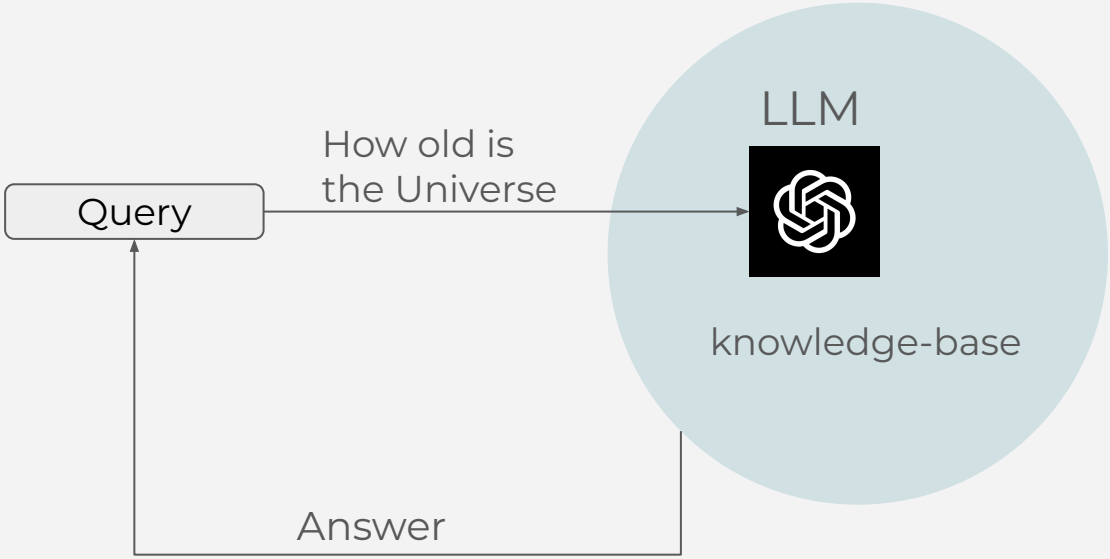
Python (Win, Mac, Linux)

https://kinsta.com/knowledgebase/install-python/

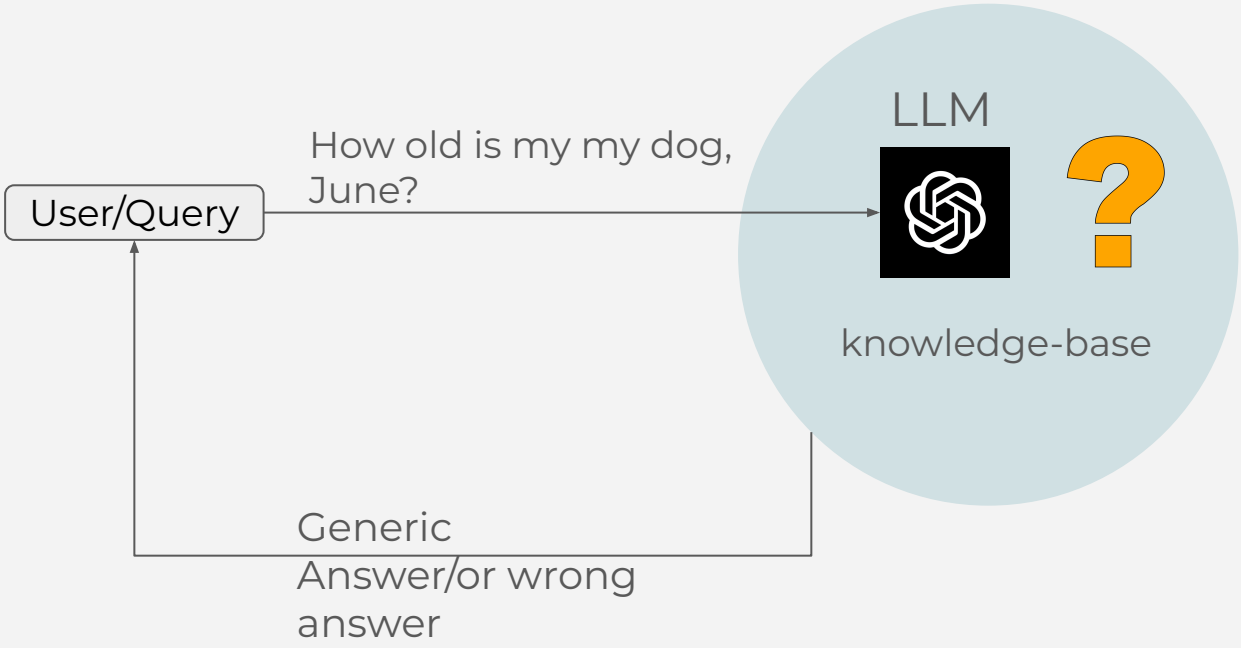
AI Agent Crash course

- What is it?
- Why (motivation)?
- Advantages

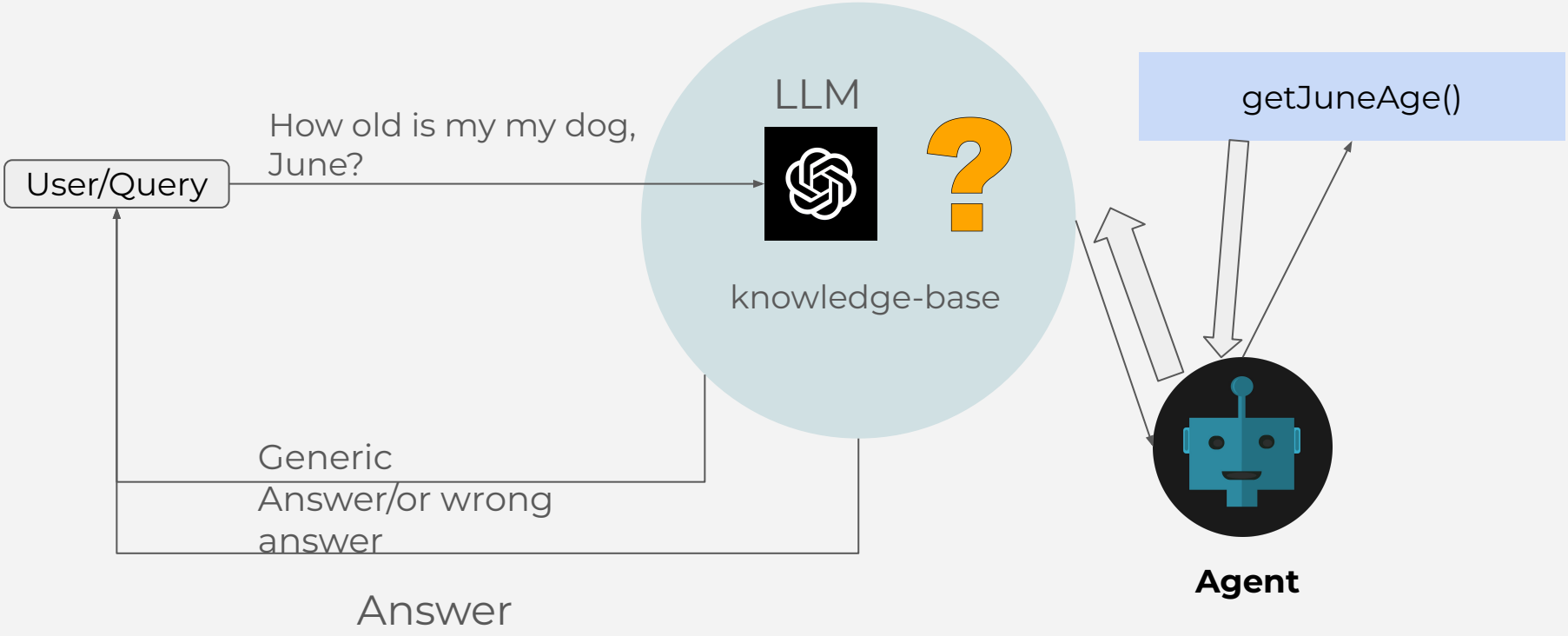
What is an (AI) agent (Motivation)?



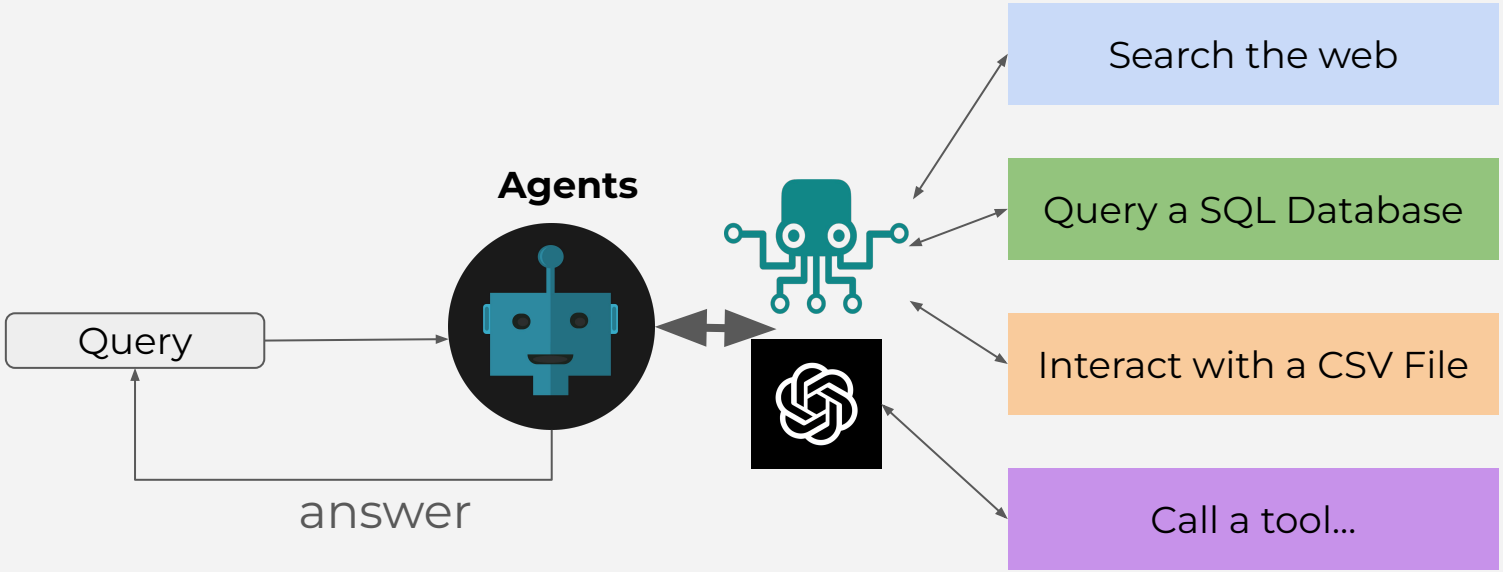
What is an (AI) agent (Motivation)?



What is an (AI) agent (Motivation)?

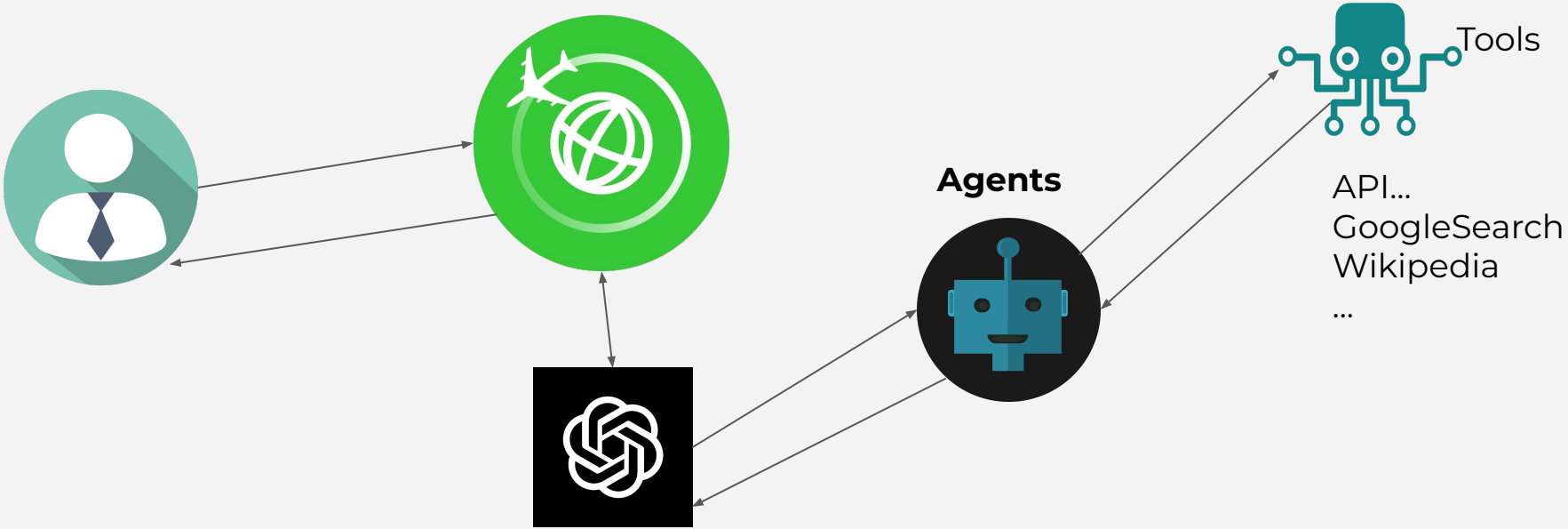


What is an (AI) agent?

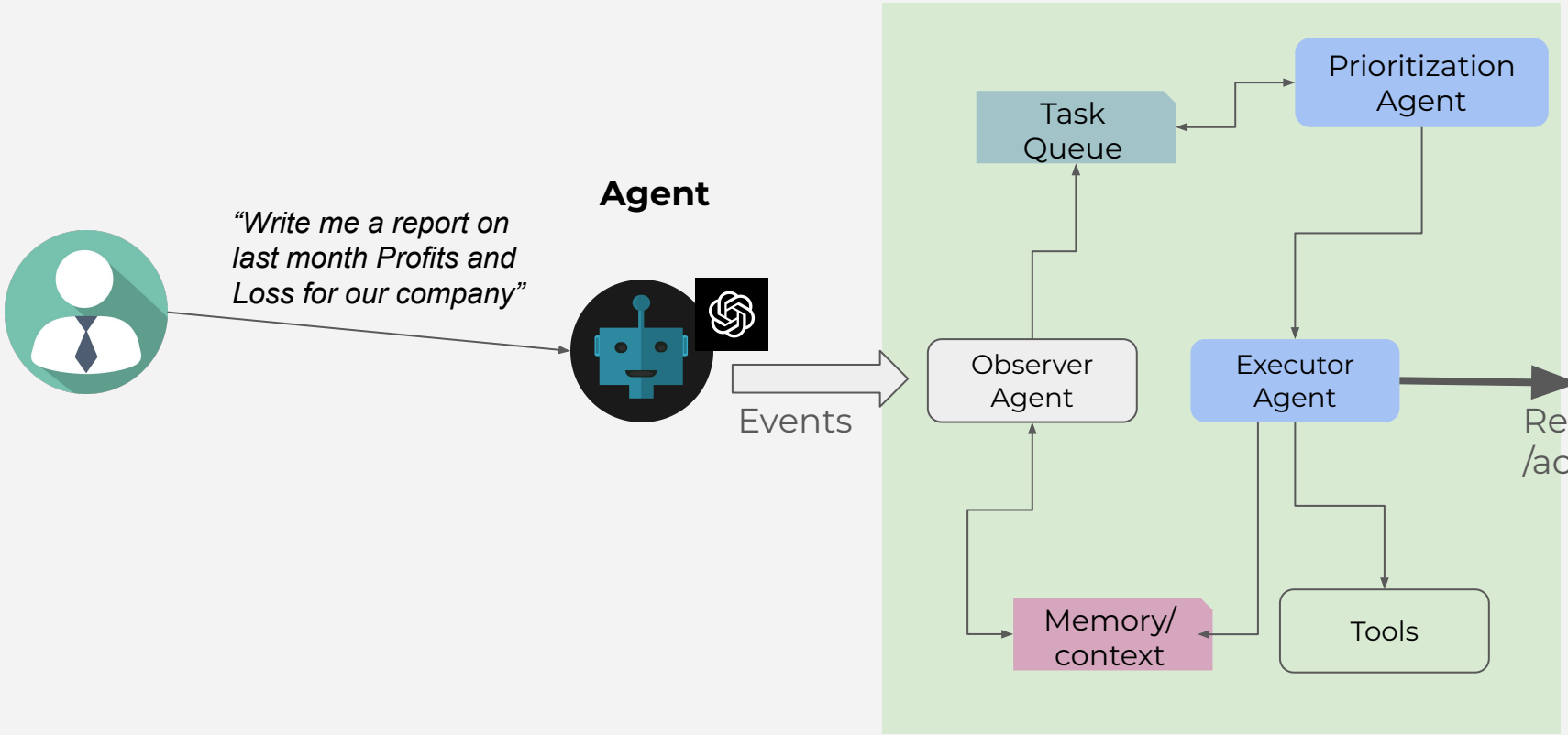


What is an agent & what they can do?

Personalized recommendations
Browse history
Previous vacations and activities...



Motivation Behind AI Agents: Solving Real-World Problems

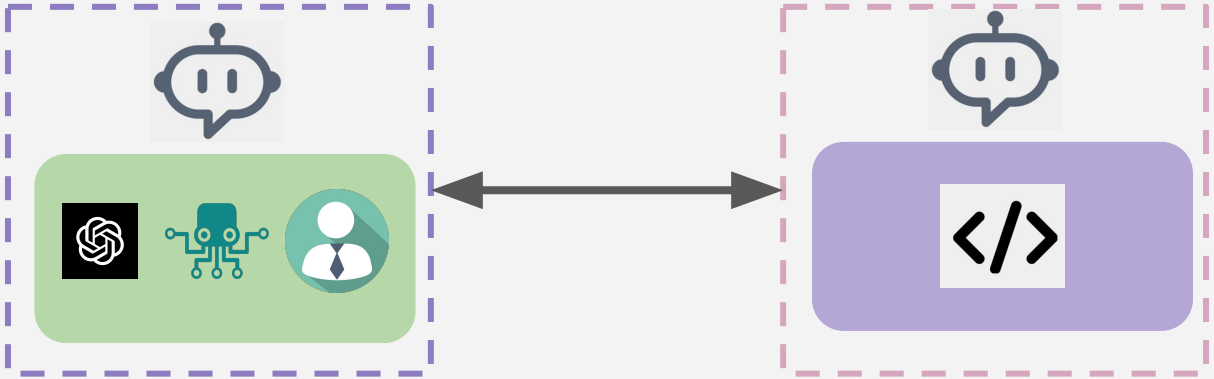


AutoGen Deep Dive

- What is it?
- Why (motivation)?
- How it works?
- Key concepts
- Hands-on

AutoGen

An open-source programming framework used for building AI agents that can **communicate** and **cooperate** with other agents to solve tasks.



AutoGen Main Features

Enables building next-gen LLM applications easier- we can build these applications based on **multi-agent conversations**

- Simplifies:
  - The orchestration
  - Automation
  - Optimization of complex LLM workflows
  - Heightens LLM models performance
  - Overcomes their weaknesses

Supports diverse conversation patterns (for complex workflows):

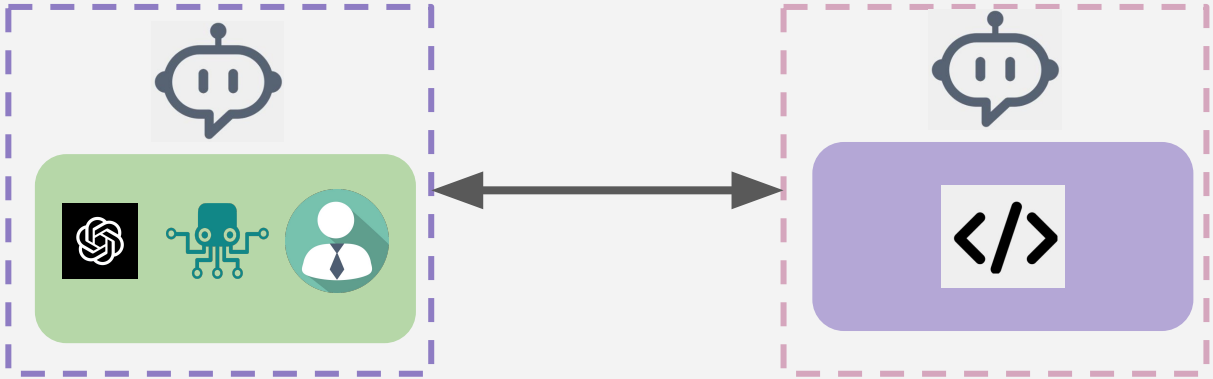
- Has customizable and conversational agents
- Developers can use AutoGen to build various types of conversation patterns

Provides a collection of working systems already:

- Developers can tap into systems that span a wide range of applications

AutoGen Building Blocks

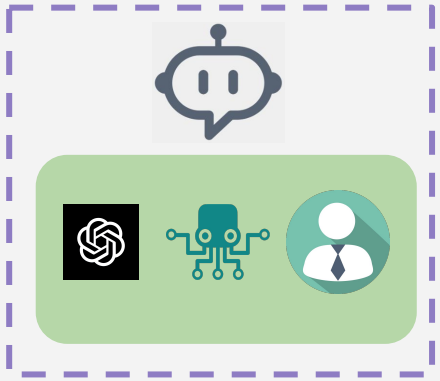
AutoGen **Agent** - an entity that can send and receive message to and from other agents.



AutoGen Building Blocks

Example of a AutoGen agent - *ConversableAgent*

ConversableAgent



Has the following components:

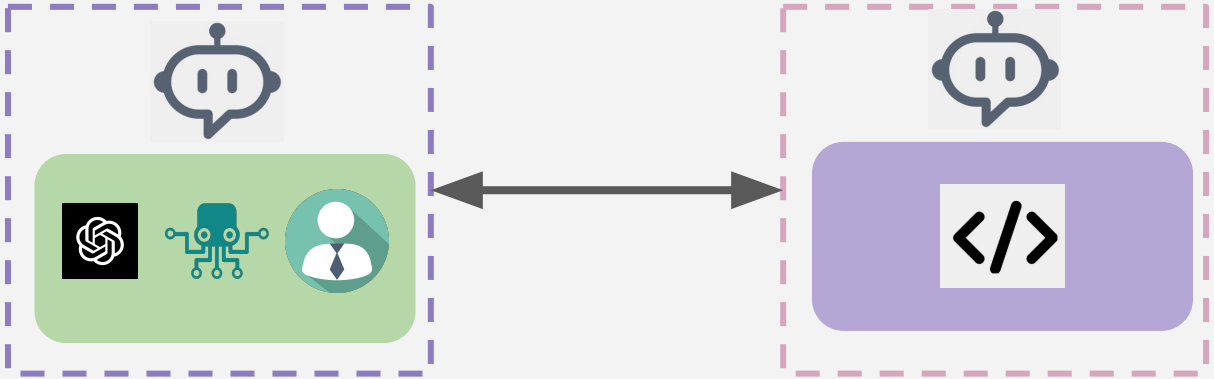
- A list of LLMs
- A code executor
- A function and tool executor
- A component for keeping human-in-the-loop

AutoGen Installation & Agent - Hands on

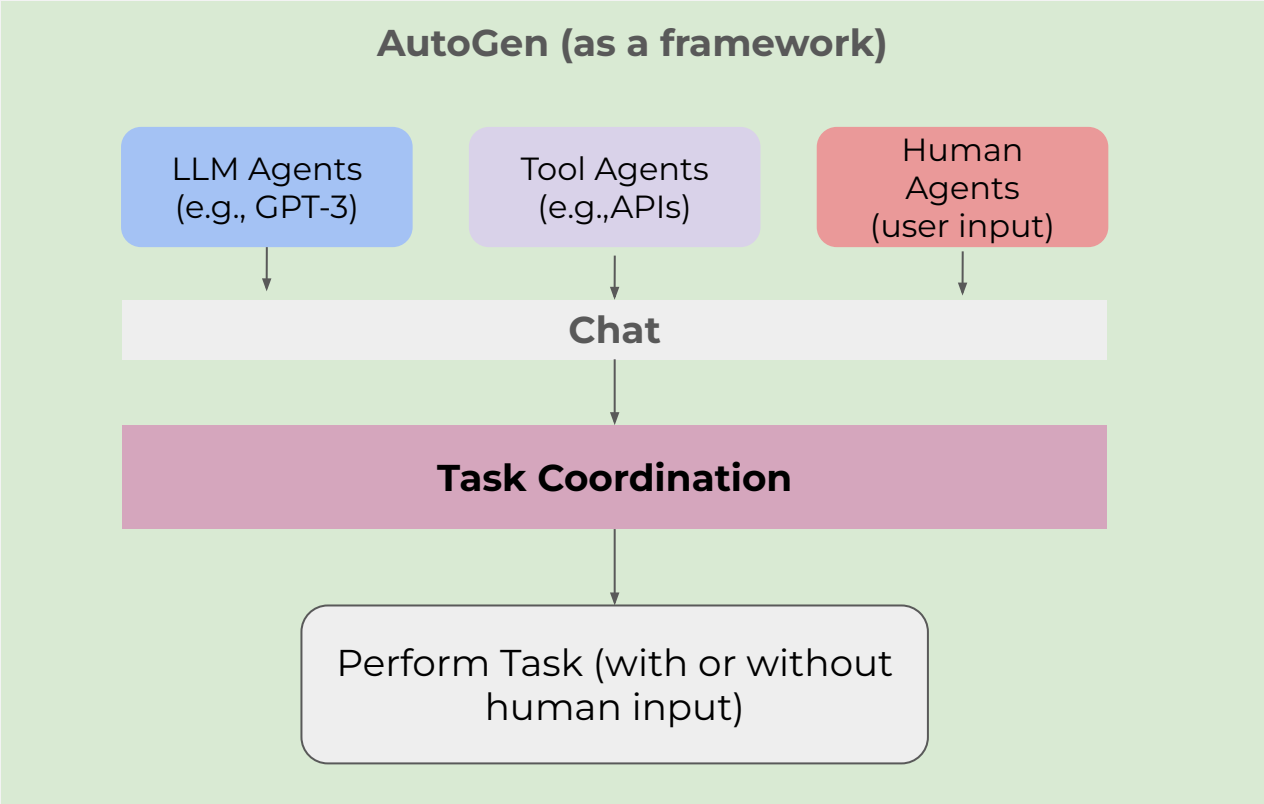
Must have Python set up in your environment

AutoGen Building Blocks

Multi-agents conversations
Conversation patterns



Multi-agents conversations



AutoGen

...makes it easy to create advanced applications using multiple agents, including AI models, tools and humans.

- **Organize and automate** - easily set up, automate, improve complex workflows that use LLMs
- **Boost performance -** enhances performance of LLMs and helps overcome their weaknesses
- **Easy to use -** minimal effort to build next-gen LLM applications
- **Flexible conversations -** support various conversation styles
- **Versatile Systems** - provides ready-to-use systems

Agents in AutoGen

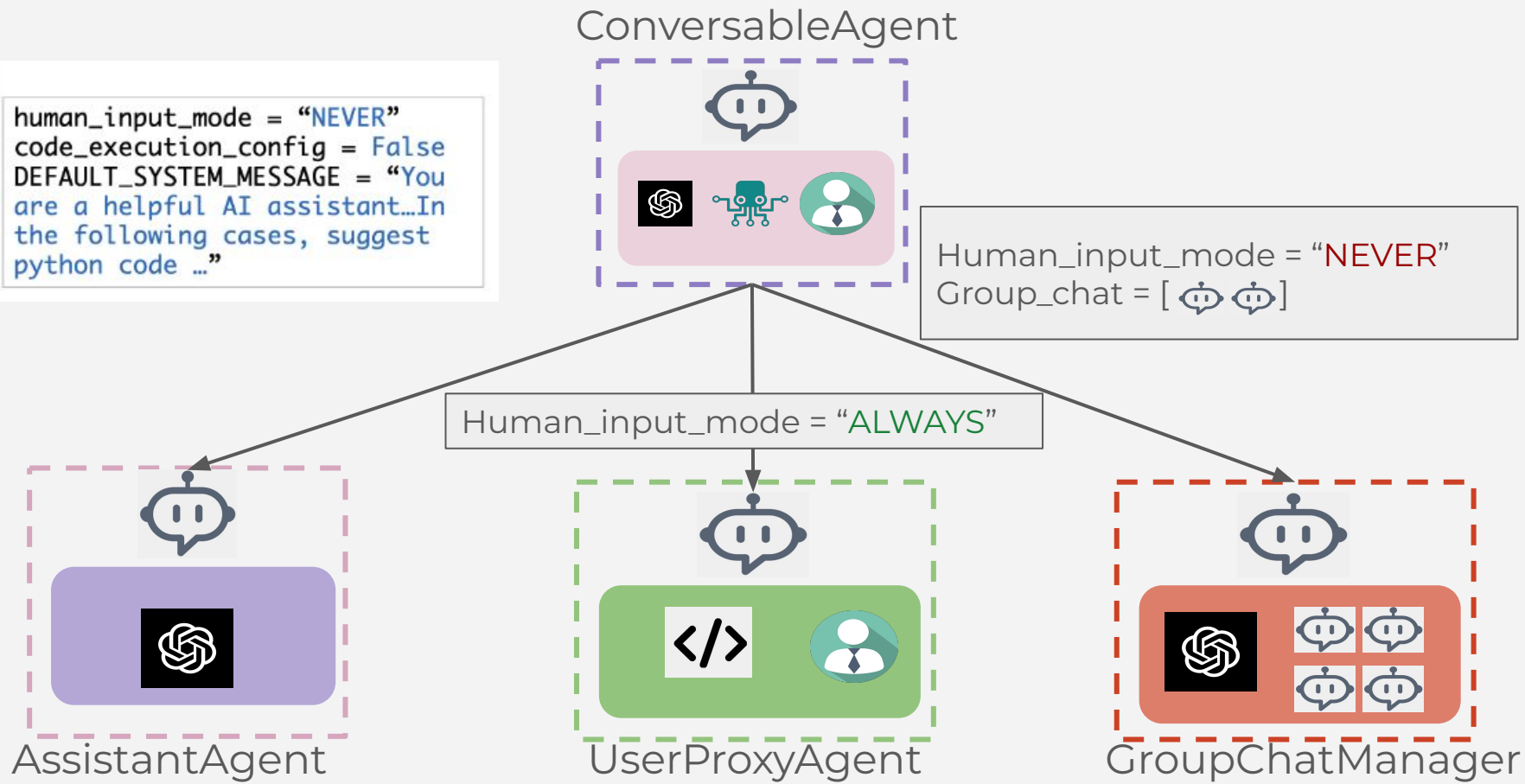
AutoGen abstracts out all intricacies about agents and how they work, and implements agents that communicate with each other to solve tasks.

These agents have two features:

- **Conversable** - agents can send and receive messages
- **Customizable** - agents can integrate with AI models, tools, humans or a combination of all.

Built-in Agents in AutoGen

```
human_input_mode = "NEVER"
code_execution_config = False
DEFAULT_SYSTEM_MESSAGE = "You
are a helpful AI assistant…In
the following cases, suggest
python code …"
```



ConversableAgent - Hands on

Set up a ConversableAgent

Multi-agent Conversation Framework Flow

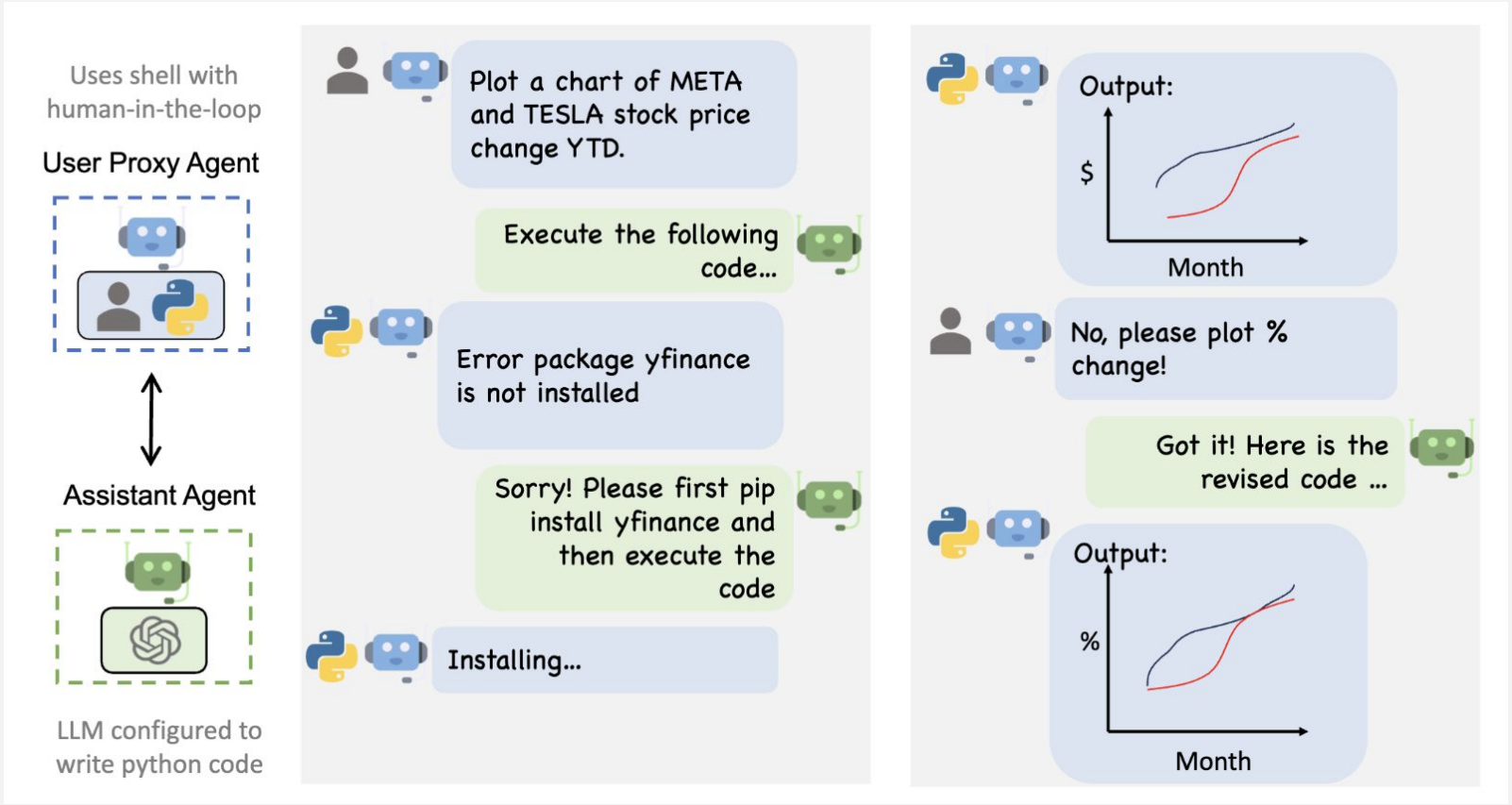


Image source: https://microsoft.github.io/autogen/docs/Getting-Started/

Multi-agent Conversation: Diverse Application Implementation

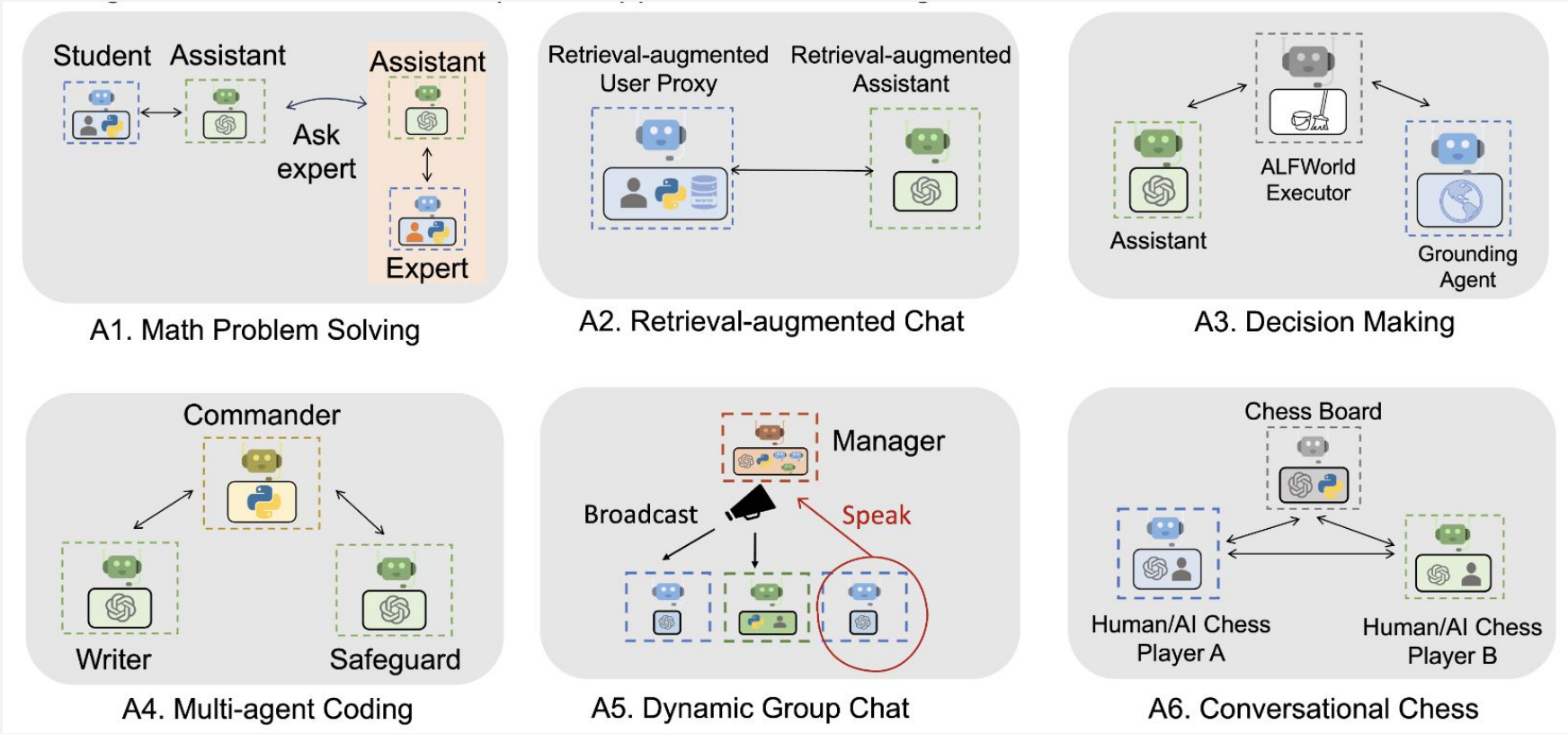
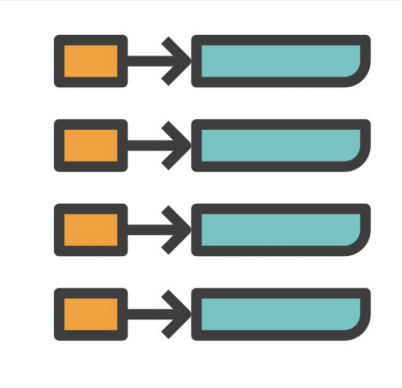


Image source: https://microsoft.github.io/autogen/docs/Use-Cases/agent_chat

Code Executors



Code executor processes messages containing code:

- Executes the code
- Returns results

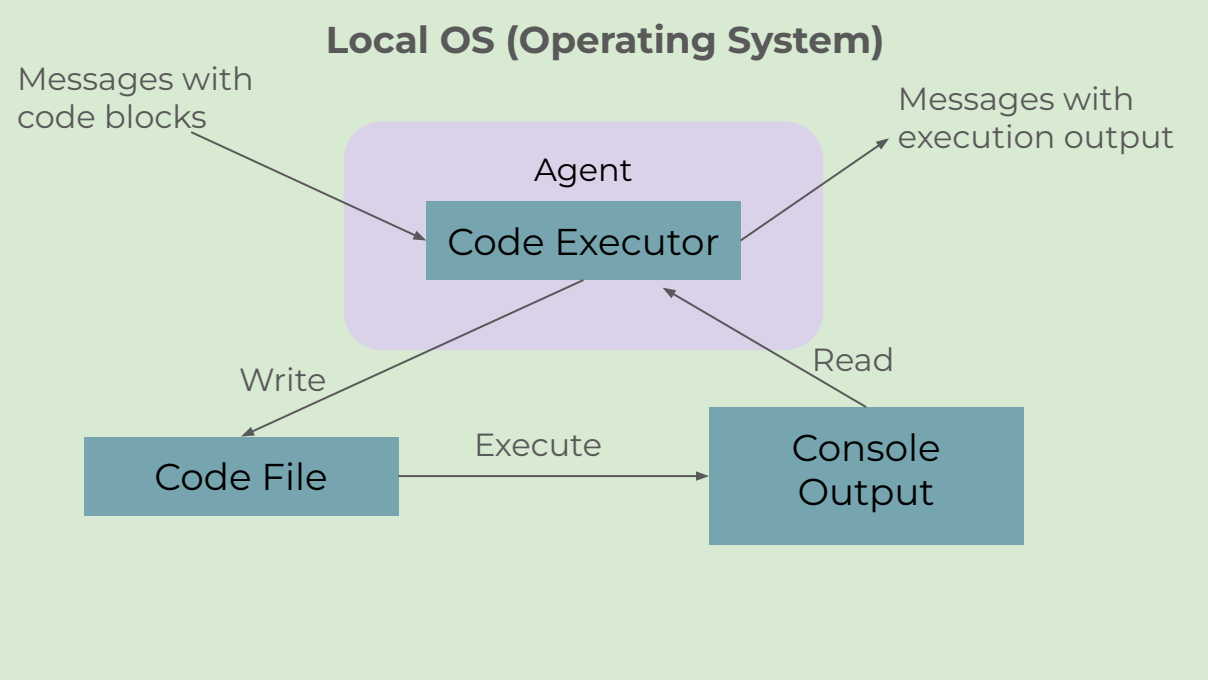
There are two built in:

- Command Line Code Executor: runs code in cmd env (Unix shell)
- Jupyter Executor: runs in a Jupyter env.

Code Executors

AutoGen can execute code:

- Locally - runs code directly on the host system (development)
- Docker container - runs code in an isolated container



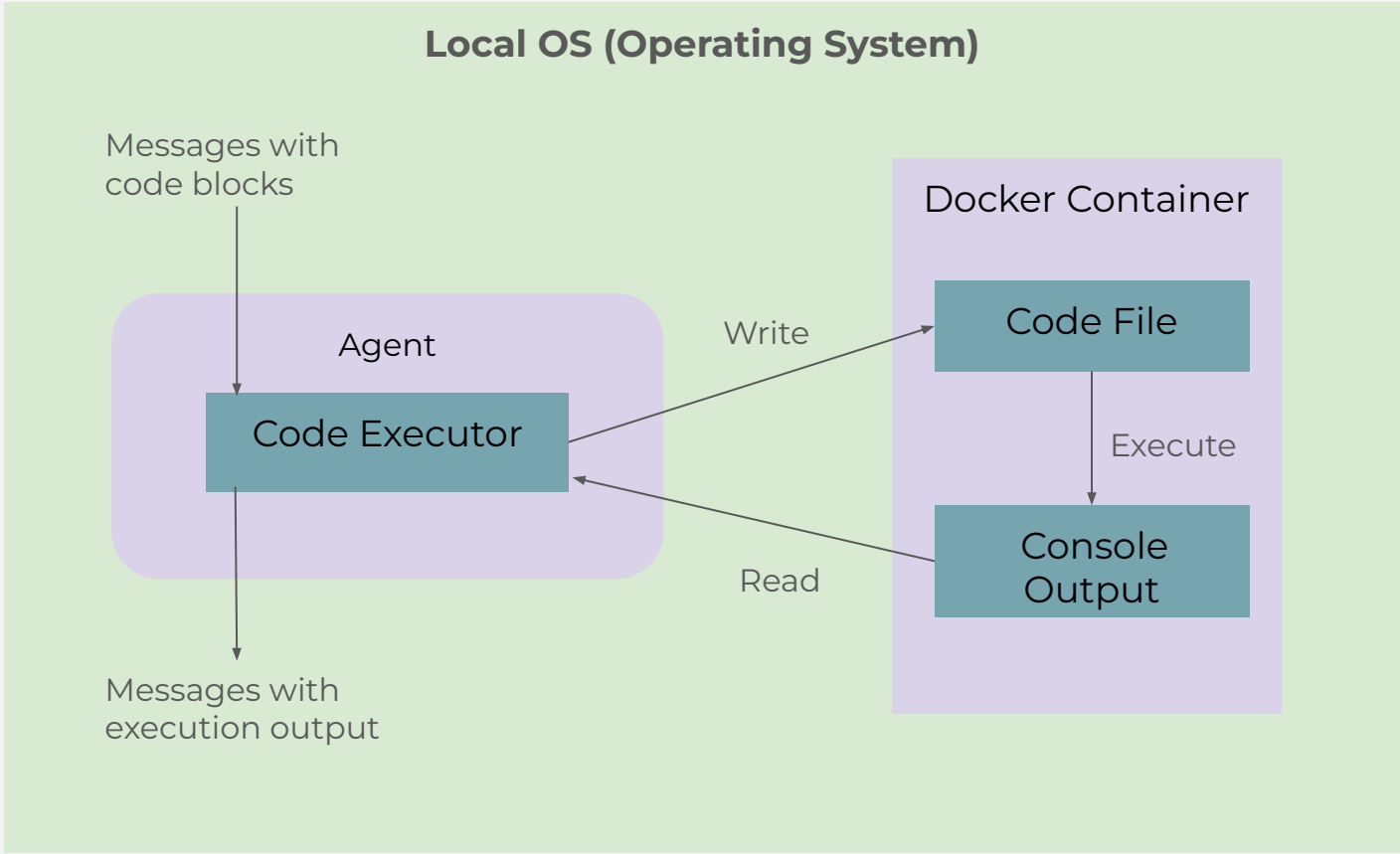
Code Executors & execution environments

AutoGen can execute code:

- Locally - runs code directly on the host system (development)
- Docker container - runs code in an isolated container (*must have Docker installed and set up*)

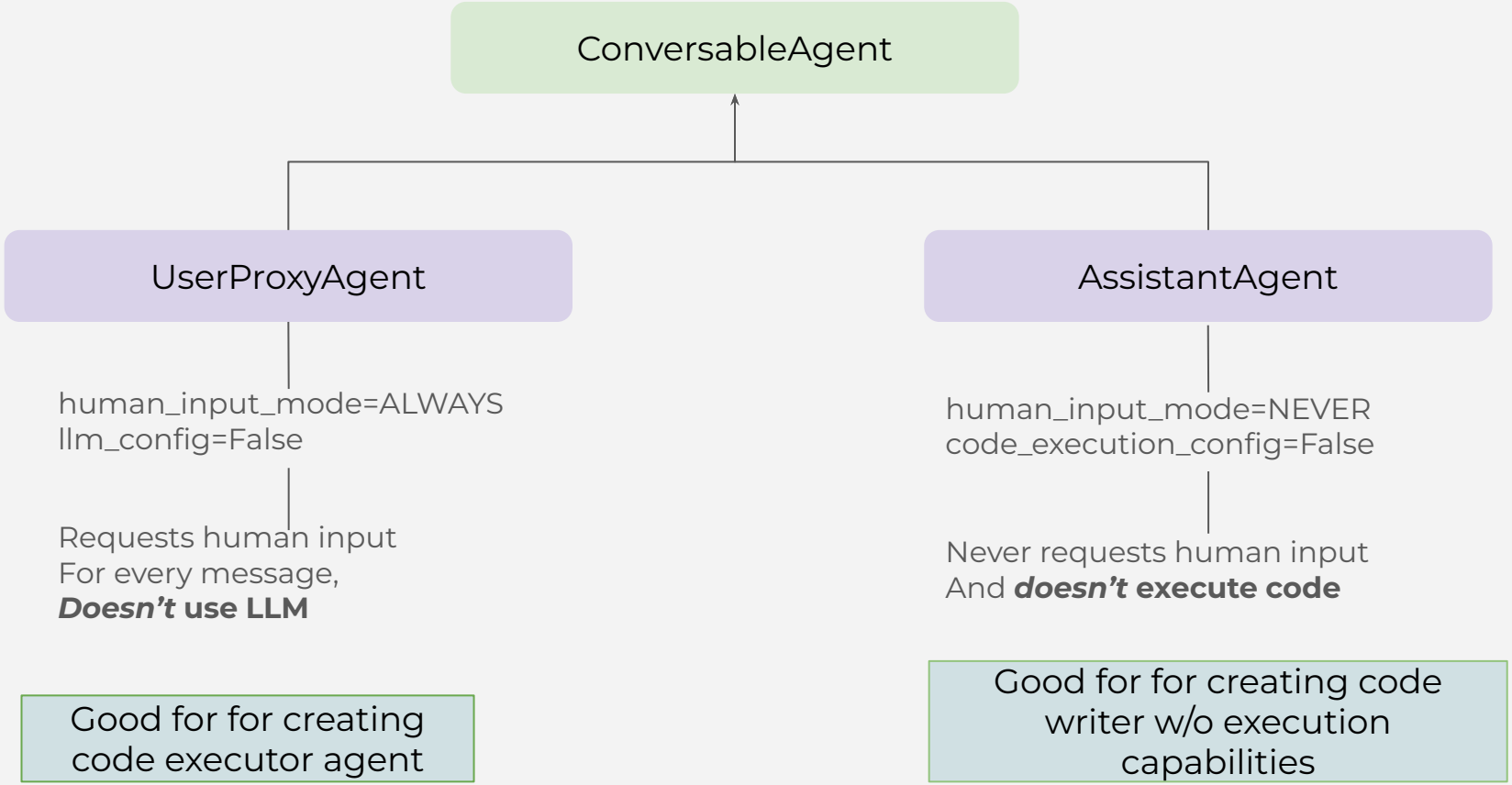
Code Executor	Environment	Platform
LocalCommandLineCodeExecutor	Shell	Local
DockerCommandLineCodeExecutor	Shell	Docker
jupyter.JupyterCodeExecutor	Jupyter Kernel (e.g., python3)	Local/Docker

Docker Execution



Hands-on: running Python code using code executor

UserProxyAgent and AssistantAgent



Hands-on: using UserProxyAgent & AssistantAgent

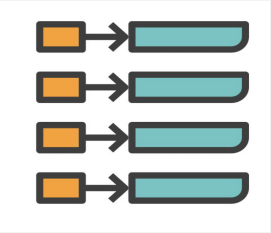
Best practice

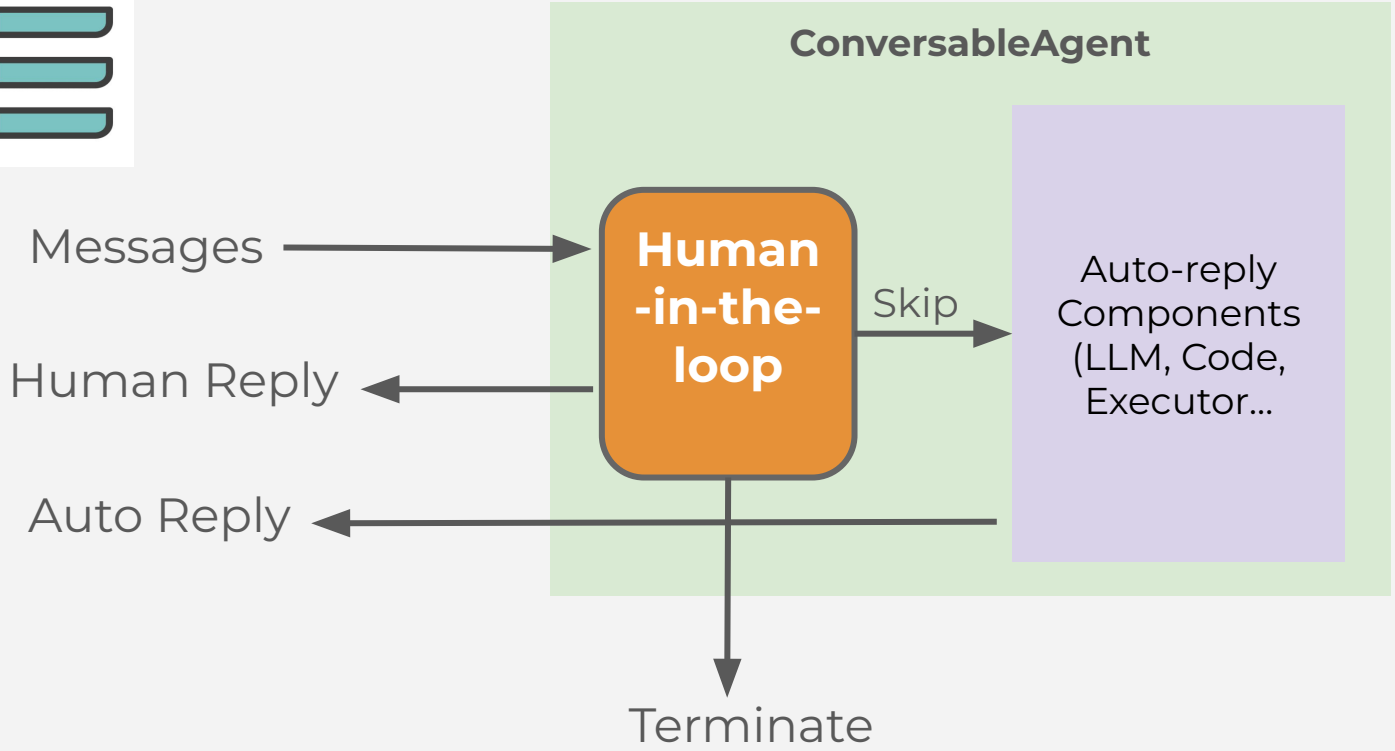
UserProxyAgent and AssystantAgent

- Always customize your agent's "system_message" instructions for your specific use case.
- Avoid subclassing "UserProxyAgent" and "AssistantAgent"

Human Feedback in Agents

Humans in-the-loop with agents





Human Input Modes

AutoGen supports 3 modes for human input:

- **NEVER** - human input is never requested
- **TERMINATE** (default) - human input is only requested when a termination condition is met
- **ALWAYS** - human input is always requested. Human skip and trigger an auto-reply.

Hands-on: Human input modes

Summary

Deep dive into AutoGen

- Building blocks and key features
- Create our first Agent
- Agent types
- UserProxyAgent and AssistantAgent
- Multi-conversation Framework
- Code Executors in AutoGen
- Adding Human input
- Best practices for UserProxyAgent and AssistantAgent

LLM Caching

AutoGen API Request Caching

AutoGen supports caching API requests for cost reduction and reusability of previously issued requests

Benefits:

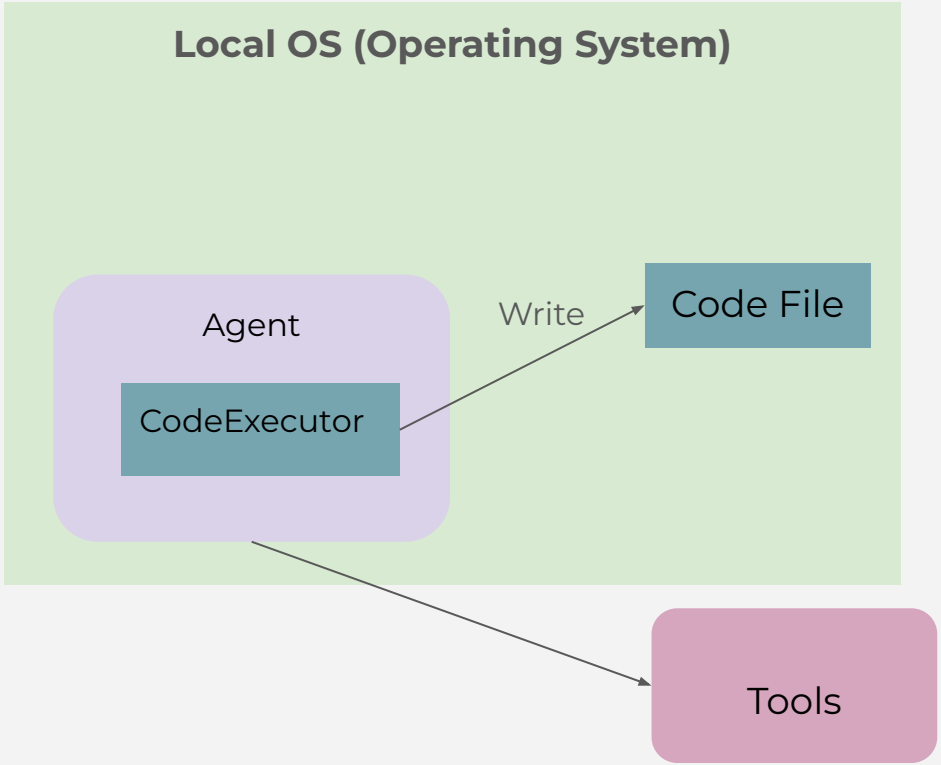
- Reproducibility: consistent results when experiments are repeated
- Cost savings: reduce expenses associated with repeated API calls.

LLM Caching

```
assistant = AssistantAgent(
    "coding_agent",
    llm_config={
        "cache_seed": None,
        "config_list": OAI_CONFIG_LIST,
        "max_tokens": 1024,
    },
)
```

None = disable caching
41 = default

AutoGen and Tools



We have no control over what an agent writes (code).

Tools in AutoGen:

- Pre-defined functions
- Controlled actions
- Controlled availability

Summary

AutoGen and tools

- What are tools
- How to use them with Agents
- Hands-on

Conversation patterns

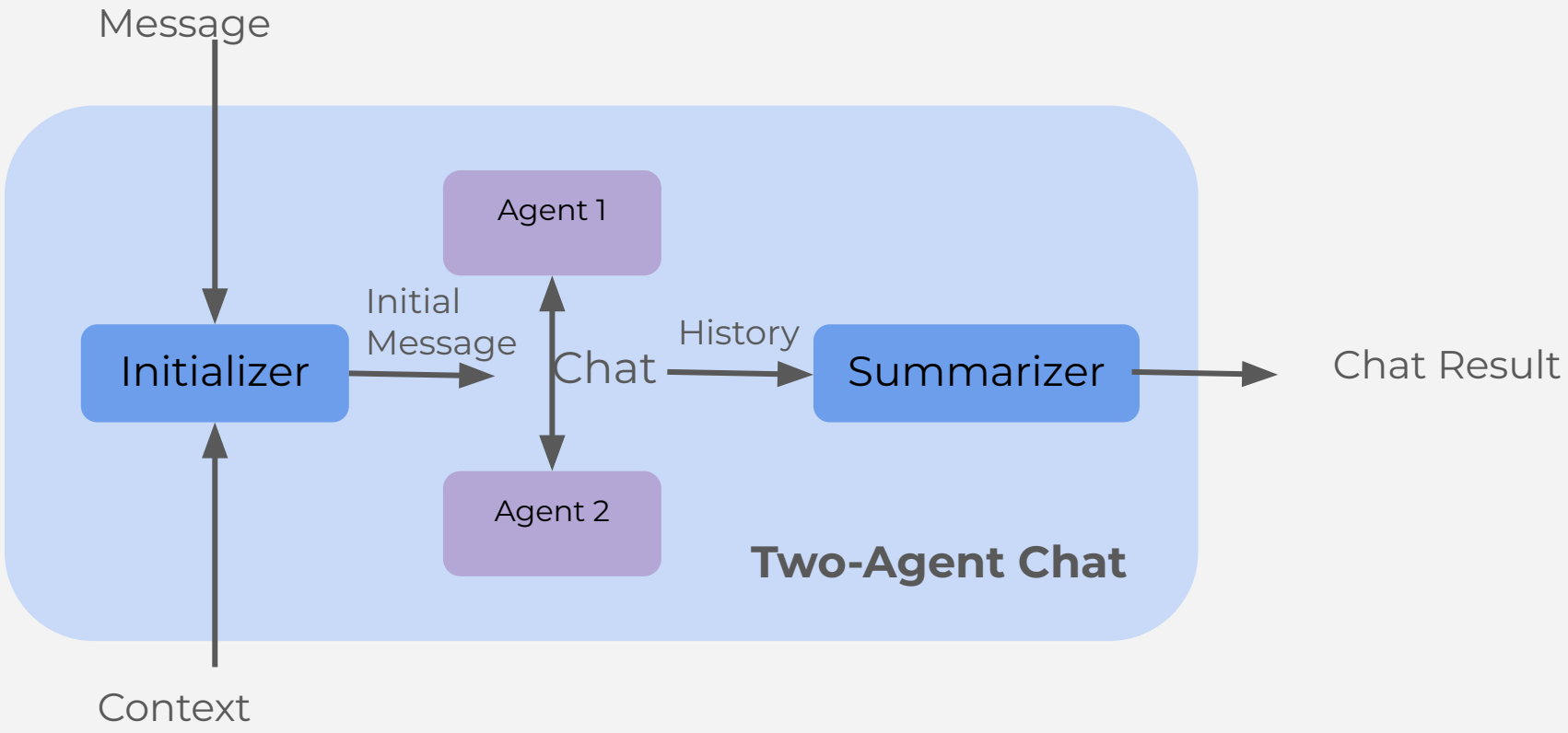
- Two-agent chat: simplest conversation pattern with two agents chatting **(used the *initiate_chat* method)**

AutoGen offers more:

- **Sequential chat**: a series of two-agents linked by a carryover mechanism
- **Group chat**: involves more than two agents
- **Nested chat**: combines a workflow into a single agent for reuse in larger workflows

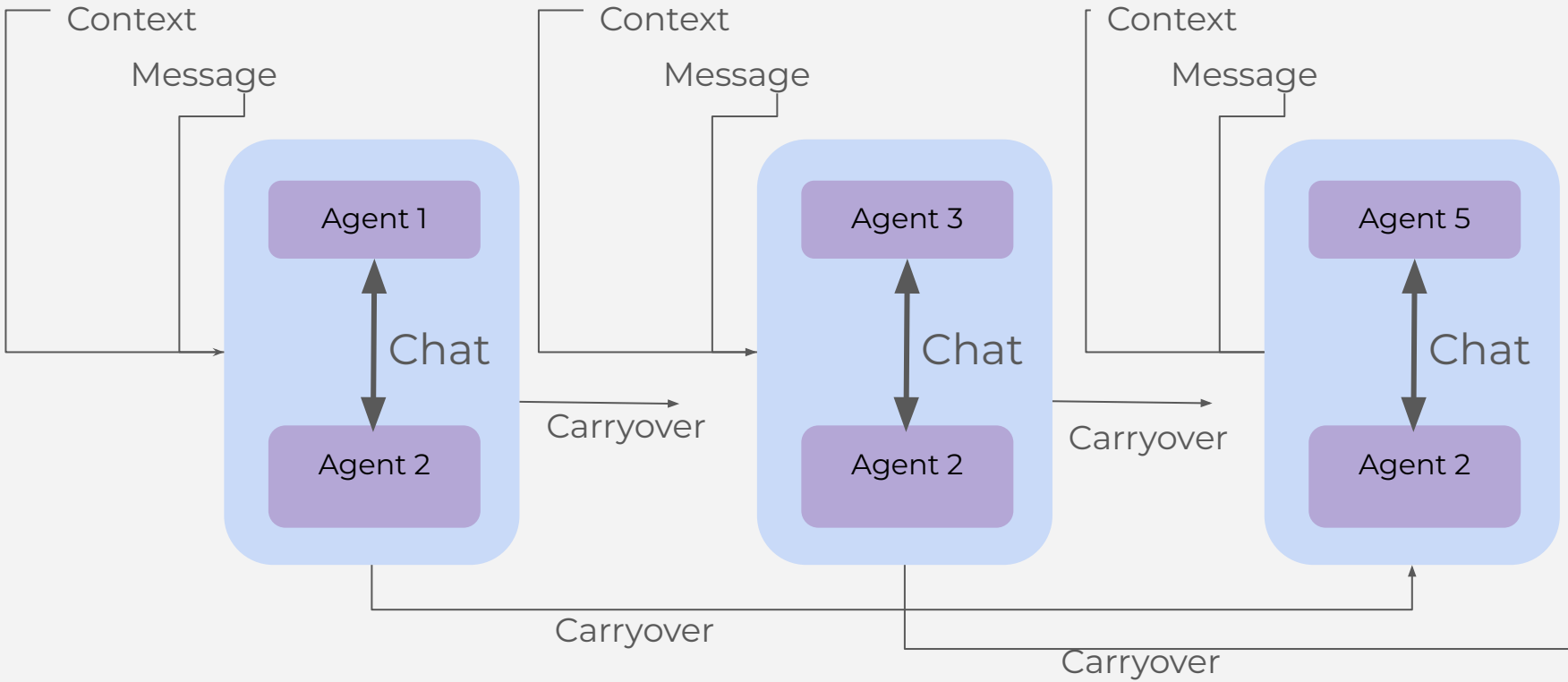
But let's first delve deeper into two-agent chats and their intricacies, then we'll explore the other ones...

Two-agent chat - how it works?



Hands-on - Travel Planner Agents

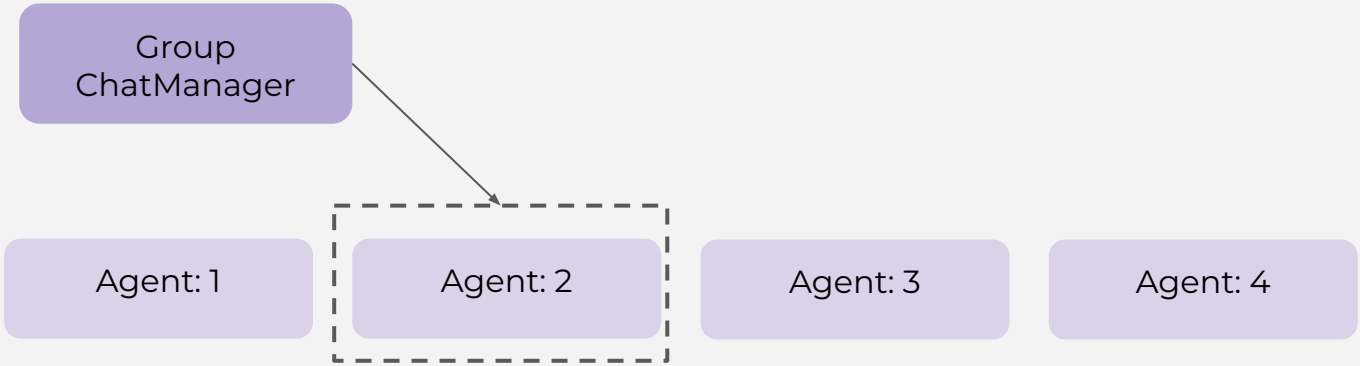
Sequential Chats



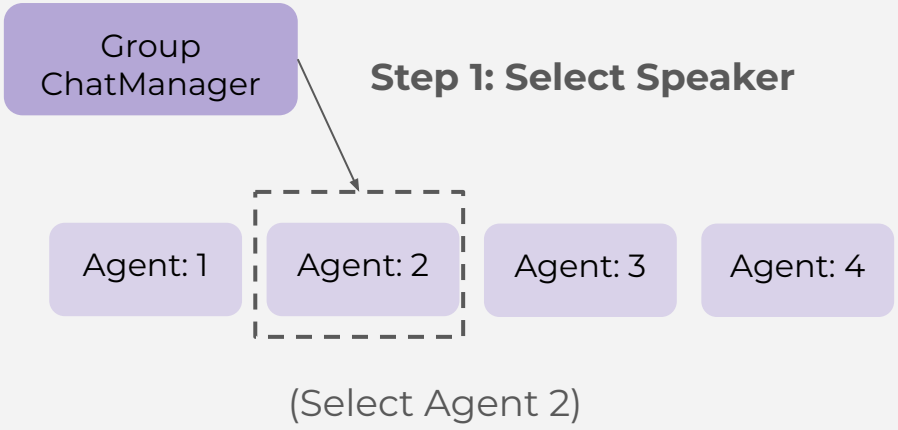
Hands-on - Sequential Chats *Document Processing Agents*

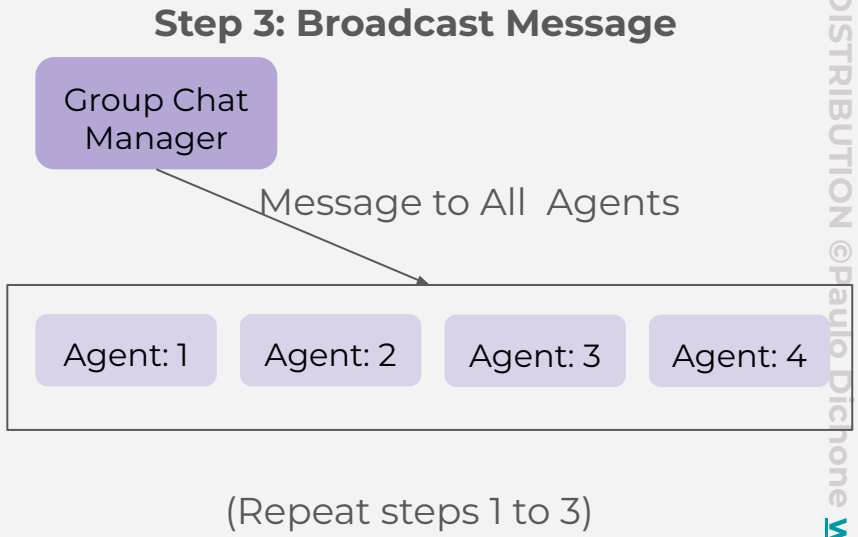
Group Chat

GroupChat - a conversation pattern involving more than two agent, directed by an agent called **GroupChatManager**



Group Chat

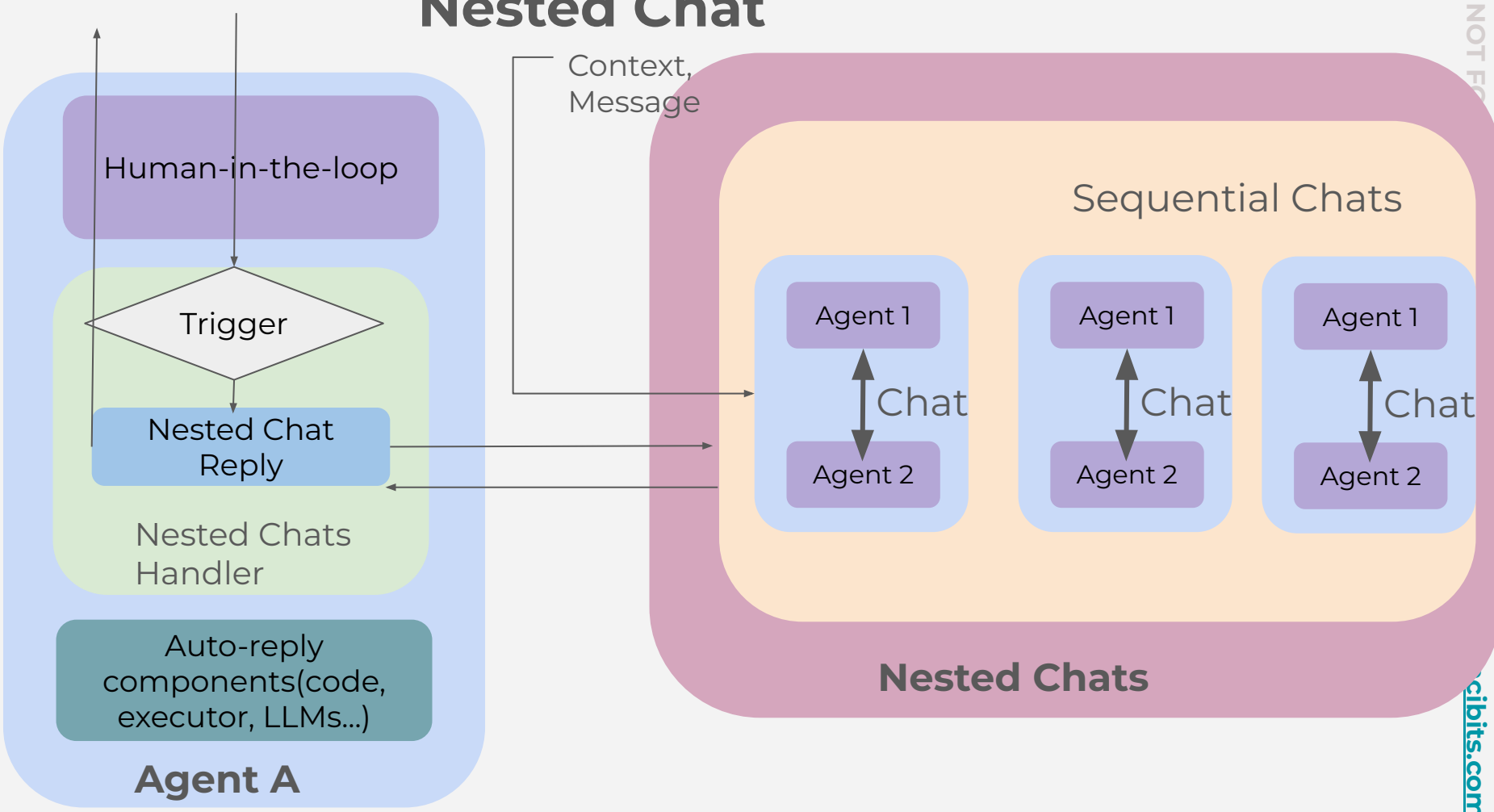




Hands-on - GroupChat

Hands-on - GroupChat in a Sequential Chat

Nested Chat


Context,
Message
Human-in-the-loop
Trigger
Nested Chat
Reply
Nested Chats
Handler
Auto-reply
components(code,
executor, LLMs...)
Agent A
Sequential Chats
Agent 1
Chat
Agent 2
Agent 1
Chat
Agent 2
Agent 1
Chat
Agent 2
Nested Chats

Hands-on - Nested Chats

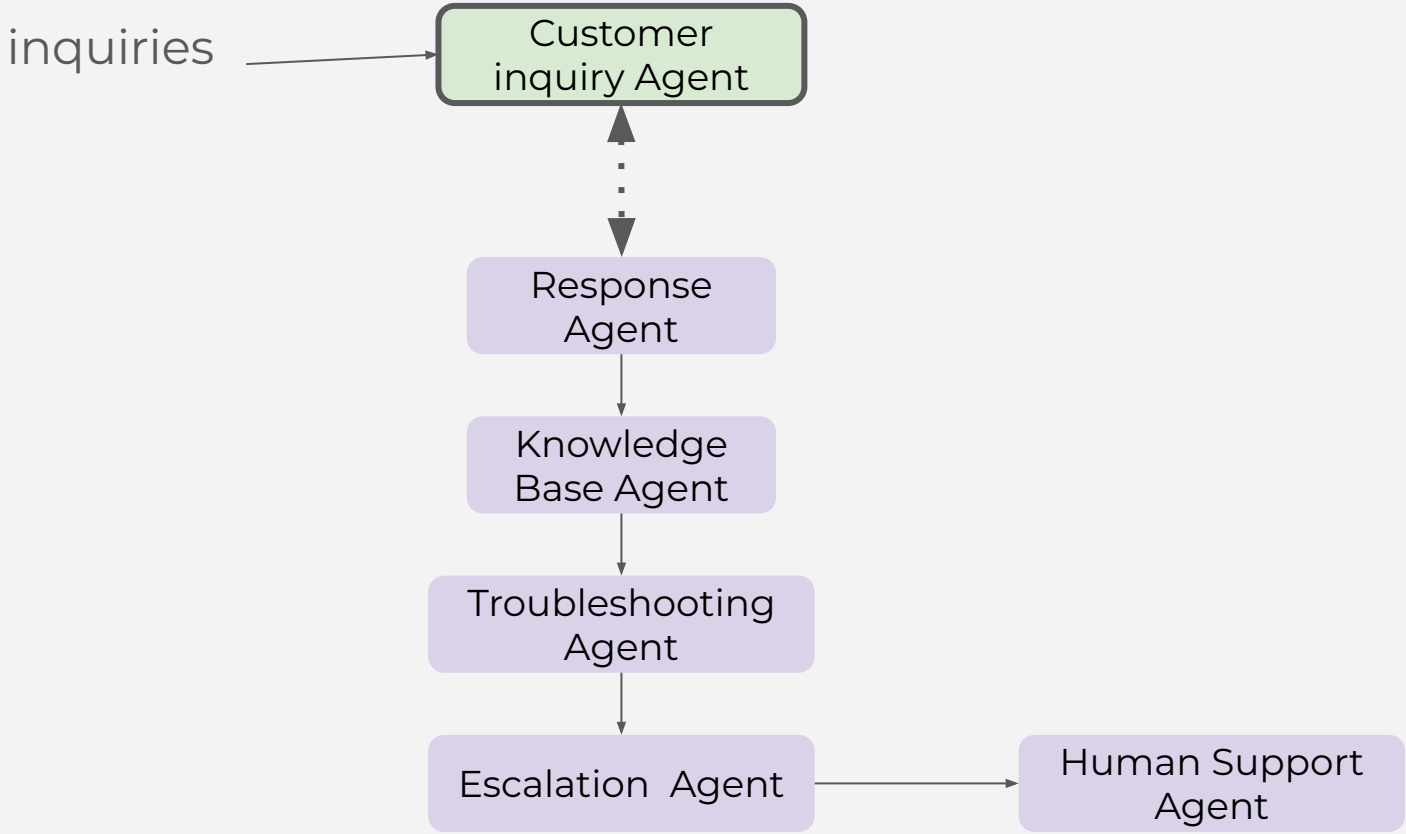
Summary

Conversation patterns:

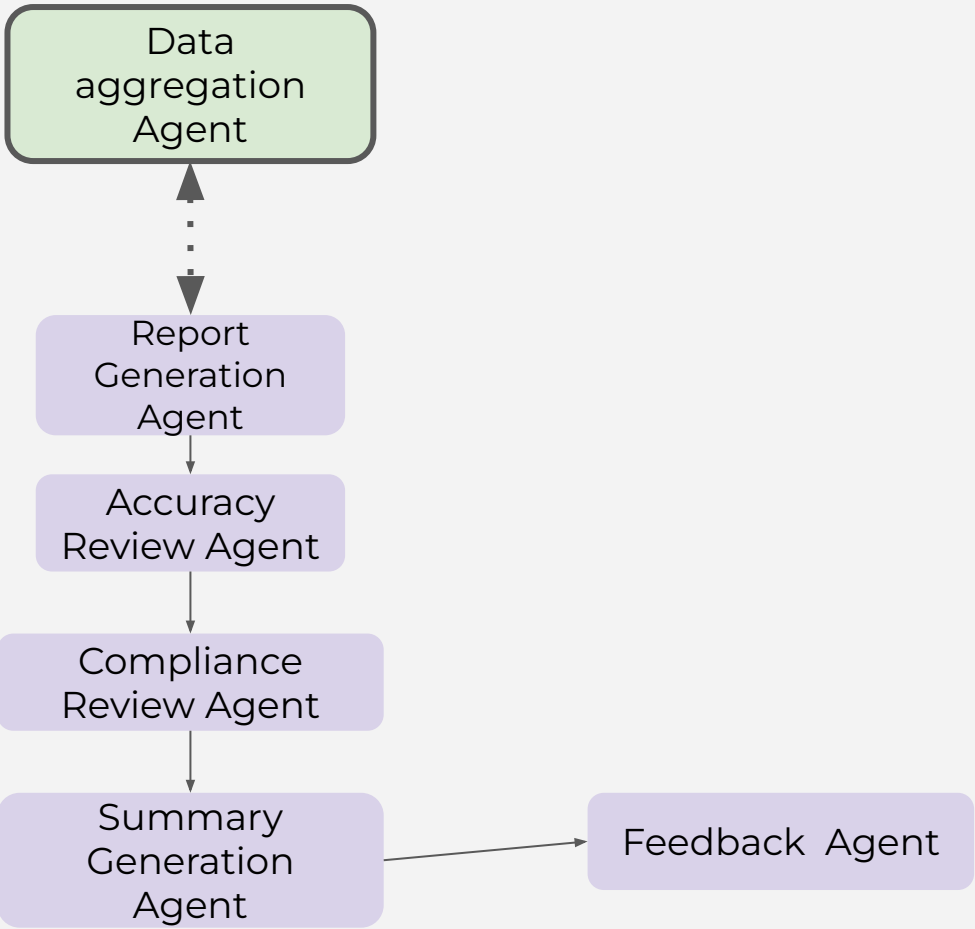
- Two-agent chat
- Sequential chat
- Group chat
- Nested chat

Use these patterns to compose blocks to create complex workflows.

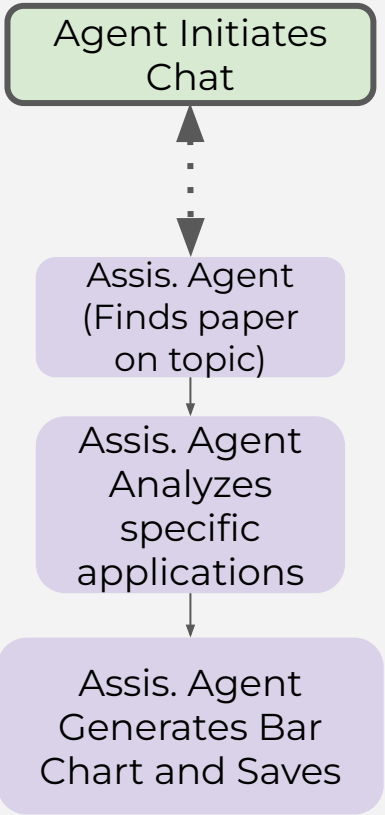
Hands-on: Customer Support Automation



Hands-on: Automated Financial Report Generation and Review



Hands-on: Automating Research Paper Analysis and Visualization



AutoGen and RAG (Retrieval-Augmented Generation

- A way of combining language models with external knowledge retrieval to improve the quality and relevance of generated response.

We can do that by:

- Constructing agent chats with *RetrieveAssistantAgent*
- *REtrieveUerProxyAgent* classes

AutoGen Studio

- A way of combining language models with external knowledge retrieval to improve the quality and relevance of generated response.

We can do that by:

- Constructing agent chats with *RetrieveAssistantAgent*
- *REtrieveUerProxyAgent* classes

Congratulations!

You made it to the end!

- Next steps...

Course Summary

- AutoGen
- AutoGen Deep dive
- Build and customize multi-agent systems
- Agentic design patterns
    - Two-Agent Chat
    - Sequential Chat
    - Group Chat
    - Nested Chat
- Effectively implement multi-agent systems
- Implement agents with different roles that collaborate to accomplish tasks
- Best practices
- Real-world, enterprise use cases (hands-on)

Wrap up - Where to Go From Here?

- Keep learning
  - Extend the projects we worked on in this course
  - Design and implement your own agents
- https://microsoft.github.io/autogen/docs/Getting-Started

Thank you!