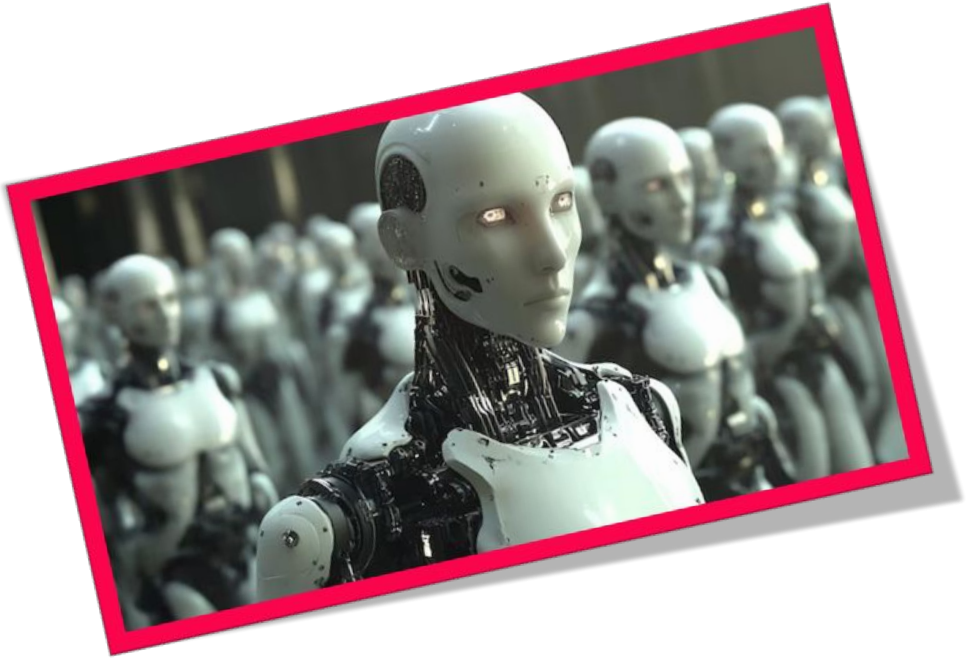
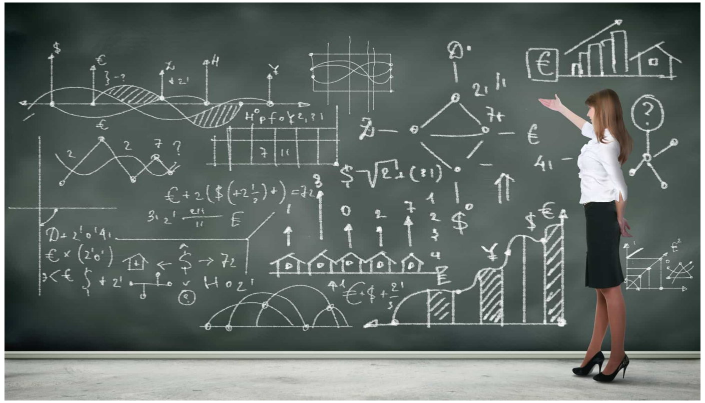
Agentic AI Patterns



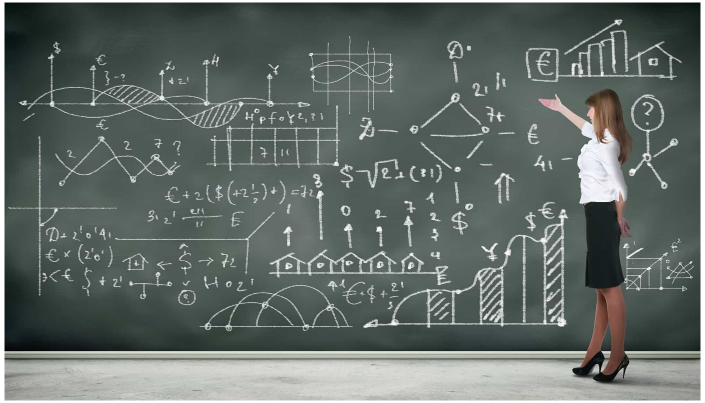
Mario Fusco, IBM

Java??? 😯 … no seriously … why not Python? 🤔



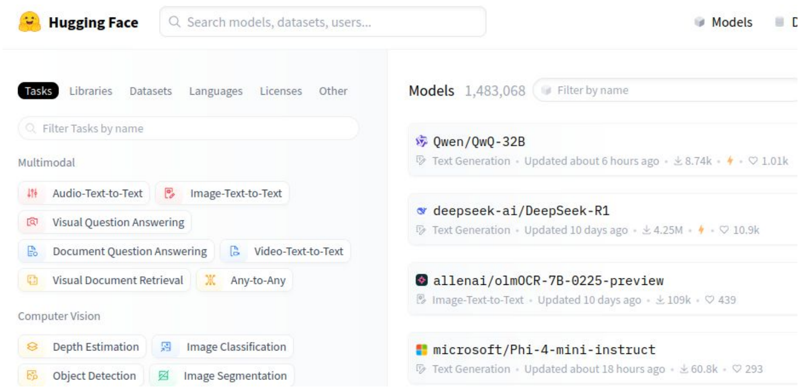
Because we are not data scientists

Java??? 😯 … no seriously … why not Python? 🤔



Because we are not data scientists

What we do is integrating existing models



Hugging Face | Search models, datasets, users... | Models

Tasks | Libraries | Datasets | Languages | Licenses | Other

Filter Tasks by name

Multimodal

- Audio-Text-to-Text
- Image-Text-to-Text
- Visual Question Answering
- Document Question Answering
- Video-Text-to-Text
- Visual Document Retrieval
- Any-to-Any

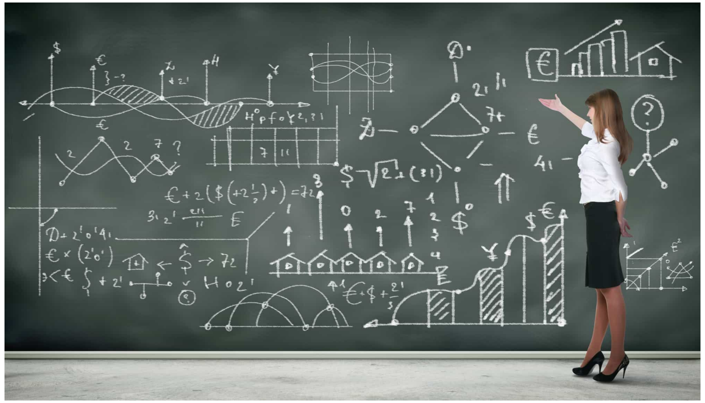
Computer Vision

- Depth Estimation
- Image Classification
- Object Detection
- Image Segmentation

Models 1,483,068 | Filter by name

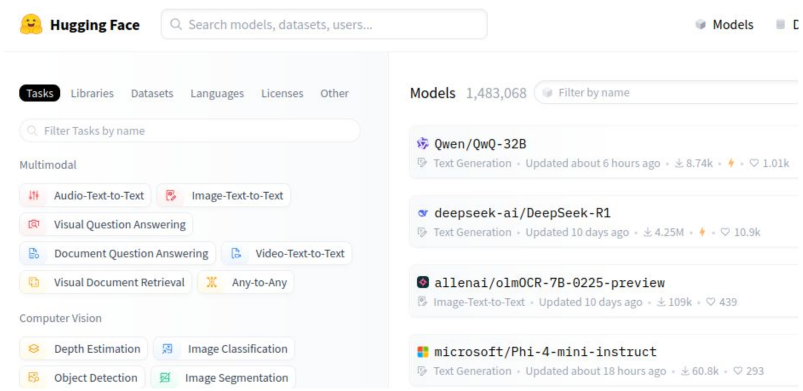
- Qwen/QwQ-32B — Text Generation • Updated about 6 hours ago • 8.74k • 1.01k
- deepseek-ai/DeepSeek-R1 — Text Generation • Updated 10 days ago • 4.25M • 10.9k
- allenai/olmOCR-7B-0225-preview — Image-Text-to-Text • Updated 10 days ago • 109k • 439
- microsoft/Phi-4-mini-instruct — Text Generation • Updated about 18 hours ago • 60.8k • 293

Java??? 😯 … no seriously … why not Python? 🤔



Because we are not data scientists

What we do is integrating existing models



Hugging Face — Search models, datasets, users... — Models

Tasks | Libraries | Datasets | Languages | Licenses | Other

Filter Tasks by name

Multimodal

- Audio-Text-to-Text
- Image-Text-to-Text
- Visual Question Answering
- Document Question Answering
- Video-Text-to-Text
- Visual Document Retrieval
- Any-to-Any

Computer Vision

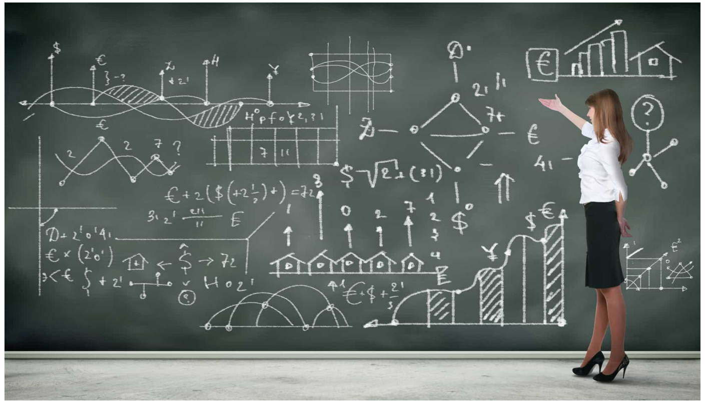
- Depth Estimation
- Image Classification
- Object Detection
- Image Segmentation

Models 1,483,068 — Filter by name

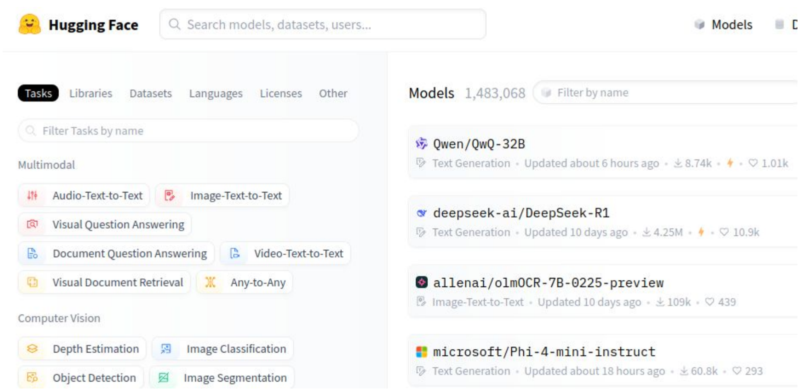
- Qwen/QwQ-32B — Text Generation · Updated about 6 hours ago · 8.74k · 1.01k
- deepseek-ai/DeepSeek-R1 — Text Generation · Updated 10 days ago · 4.25M · 10.9k
- allenai/olmOCR-7B-0225-preview — Image-Text-to-Text · Updated 10 days ago · 109k · 439
- microsoft/Phi-4-mini-instruct — Text Generation · Updated about 18 hours ago · 60.8k · 293

into enterprise-grade systems and applications

Java??? 😯 … no seriously … why not Python? 🤔



What we do is integrating existing models



Hugging Face | Search models, datasets, users... | Models | D

Tasks | Libraries | Datasets | Languages | Licenses | Other

Filter Tasks by name

Multimodal

Audio-Text-to-Text | Image-Text-to-Text | Visual Question Answering | Document Question Answering | Video-Text-to-Text | Visual Document Retrieval | Any-to-Any

Computer Vision

Depth Estimation | Image Classification | Object Detection | Image Segmentation

Models 1,483,068 | Filter by name

Qwen/QwQ-32B
Text Generation · Updated about 6 hours ago · 8.74k · · 1.01k

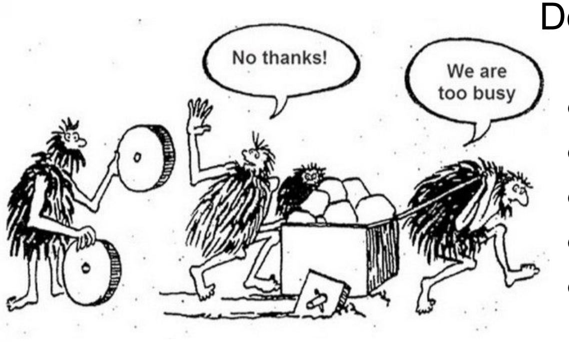
deepseek-ai/DeepSeek-R1
Text Generation · Updated 10 days ago · 4.25M · · 10.9k

allenai/olmOCR-7B-0225-preview
Image-Text-to-Text · Updated 10 days ago · 109k · 439

microsoft/Phi-4-mini-instruct
Text Generation · Updated about 18 hours ago · 60.8k · 293

Because we are not data scientists

into enterprise-grade systems and applications



Do you really want to do

- Transactions
- Security
- Scalability
- Observability
- … you name it

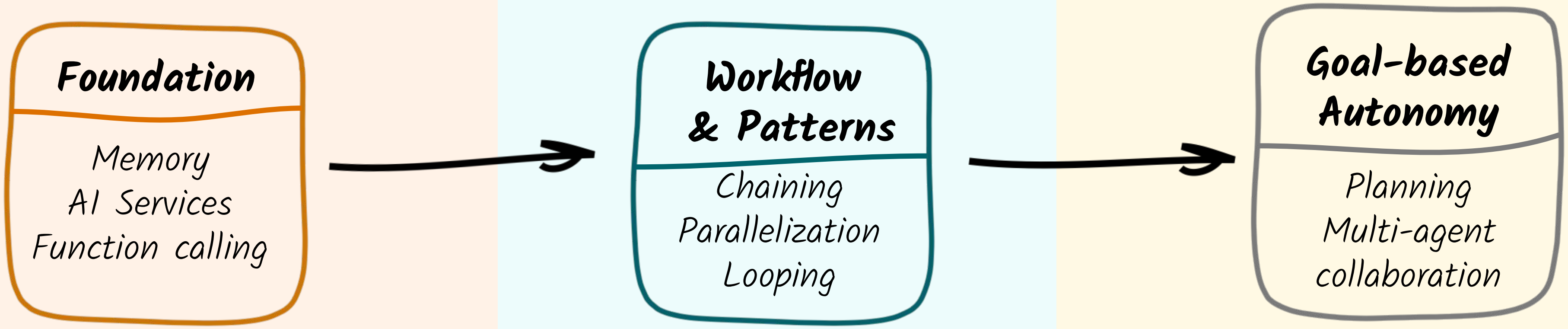
in Python???

I don't care if it works on your Jupyter notebook

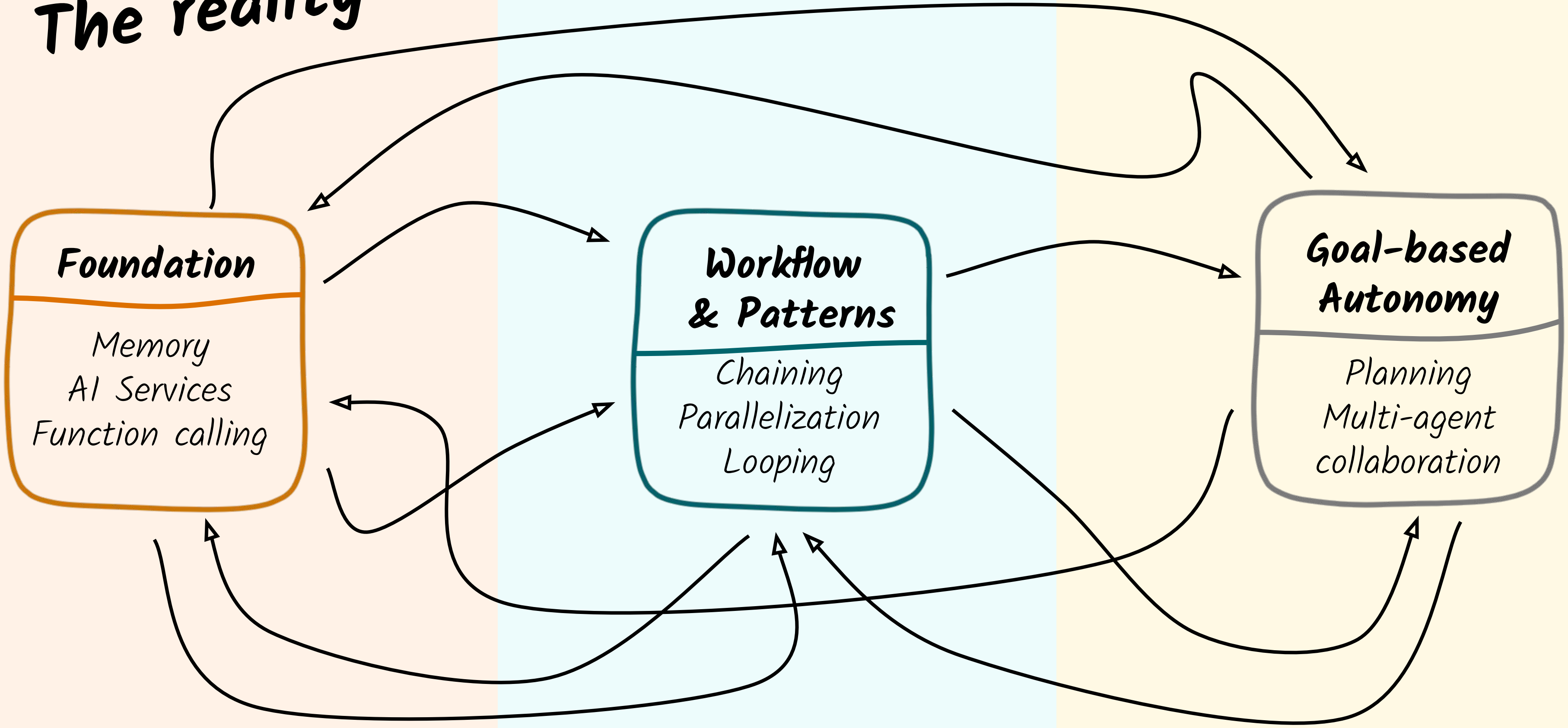
- it works on my computer
- yes but we are not going to give your computer to the client

We are not shipping your Jupyter notebook

What I will tell you...



The reality

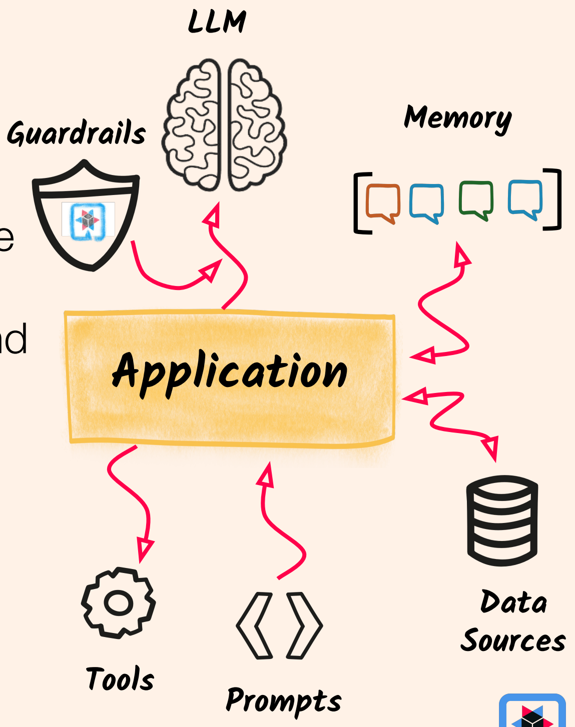

Foundation
Memory
AI Services
Function calling
Workflow & Patterns
Chaining
Parallelization
Looping
Goal-based Autonomy
Planning
Multi-agent collaboration

It all starts with a single AI Service

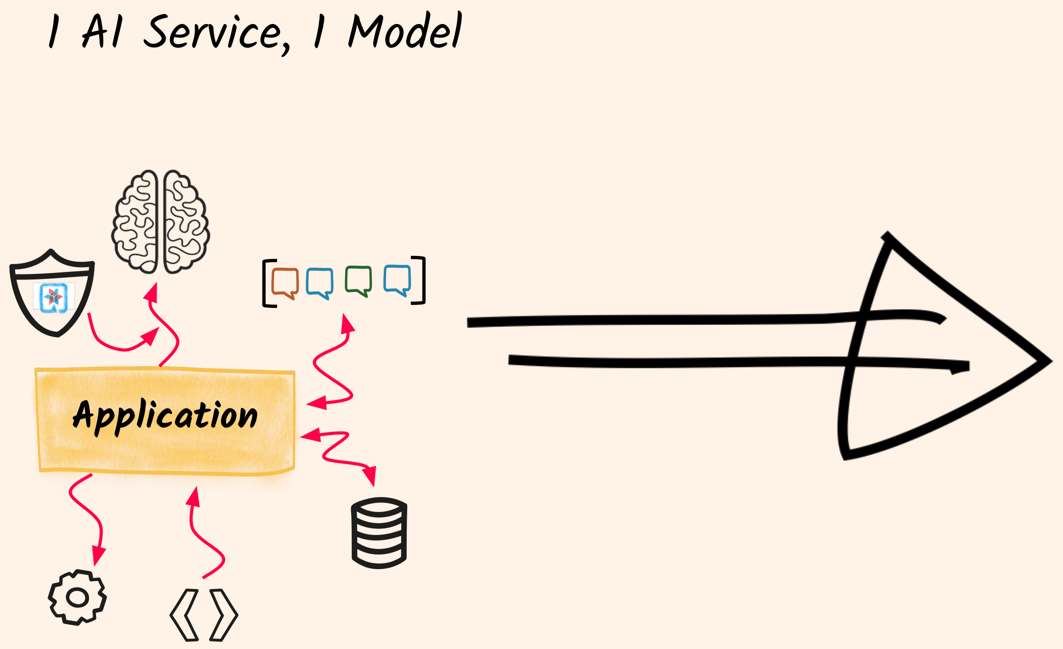
A **Large Language Model** is at the core of any **AI-Infused Application** … but this is not enough.

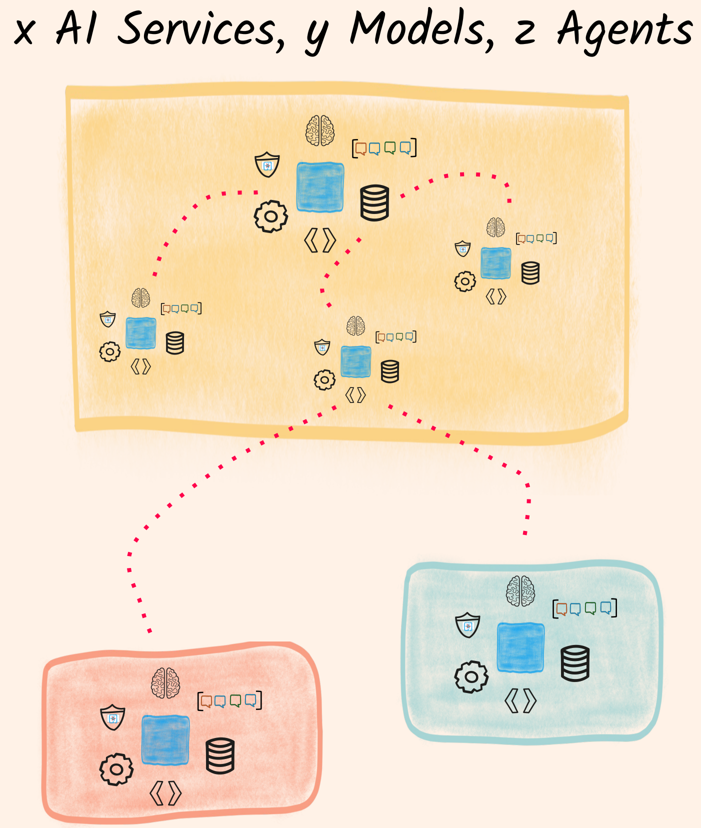
You also need:

- Well crafted **prompts** guiding the LLM in the most precise and least ambiguous possible ways
- A **chat memory** to "remember" previous interactions and make the AI service conversational
- External tools (**function calling**) expanding LLM capabilities and take responsibility for deterministic tasks where generative AI falls short
- **Data/Knowledge sources** to provide contextual information (RAG) and persist the LLM state
- **Guardrails** to prevent malicious input and block wrong or unacceptable responses



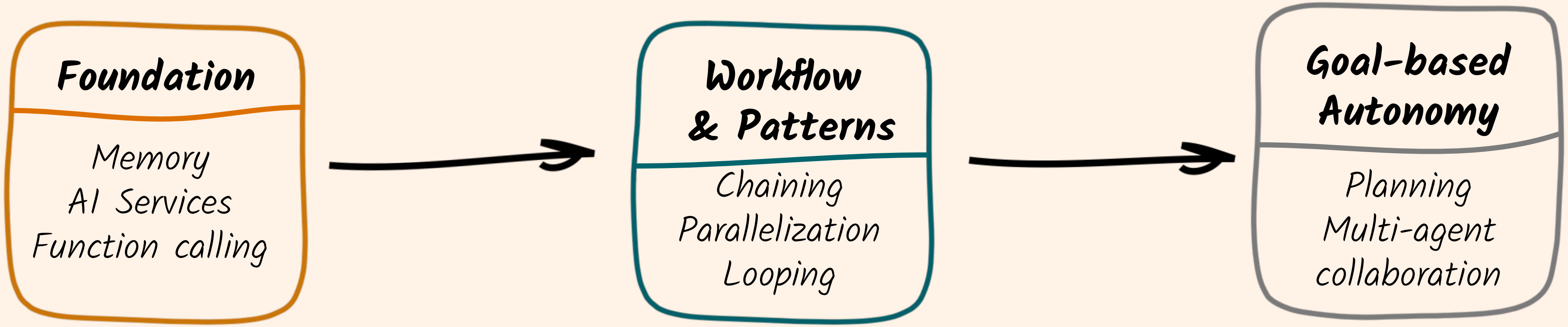
From a single AI service to Agentic Systems





From single AI Service to Agents and Agentic Systems

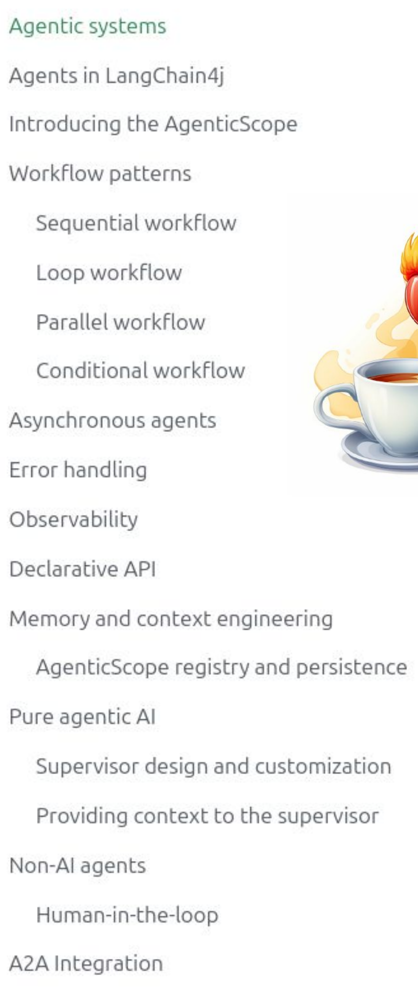
In essence what makes an **AI service** also an **Agent** is the capability to **collaborate** with other Agents in order to perform more complex tasks and pursue a common goal



The new langchain4j-agentic module

LangChain4j 1.3.0 introduces a new (experimental) agentic module.

All use cases discussed in this presentation are based on it.



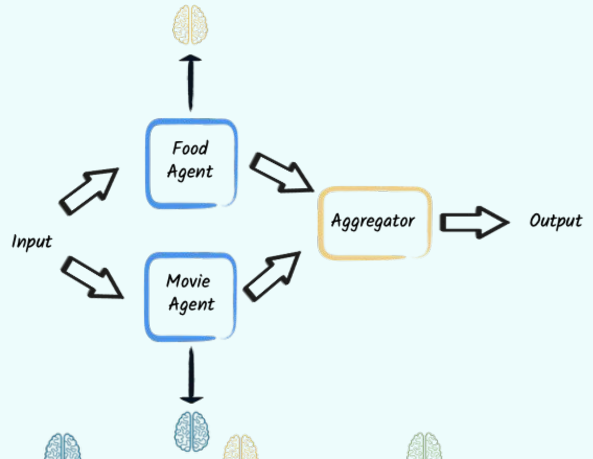
- Agentic systems
- Agents in LangChain4j
- Introducing the AgenticScope
- Workflow patterns
  - Sequential workflow
  - Loop workflow
  - Parallel workflow
  - Conditional workflow
- Asynchronous agents
- Error handling
- Observability
- Declarative API
- Memory and context engineering
  - AgenticScope registry and persistence
- Pure agentic AI
  - Supervisor design and customization
  - Providing context to the supervisor
- Non-AI agents
  - Human-in-the-loop
- A2A Integration

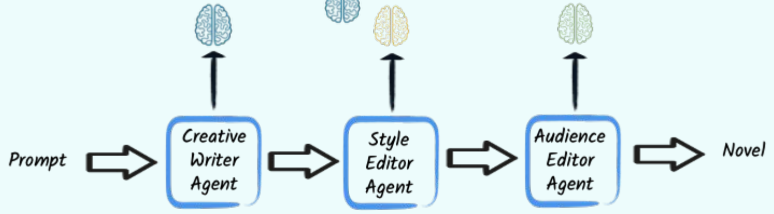
Agents programmatic orchestration

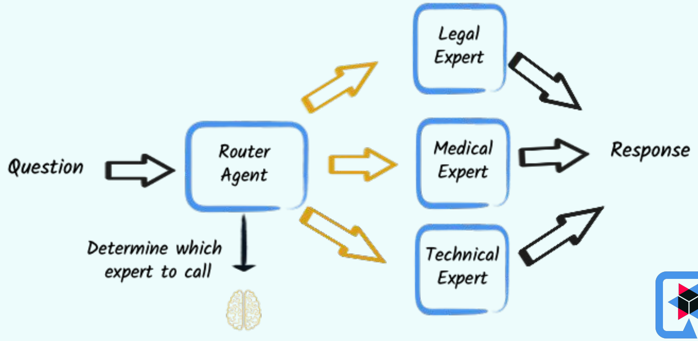
The simplest way to glue agents together is programmatically orchestrating them in fixed and predetermined workflows

4 basic patterns that can be used as building blocks to create more complex interactions

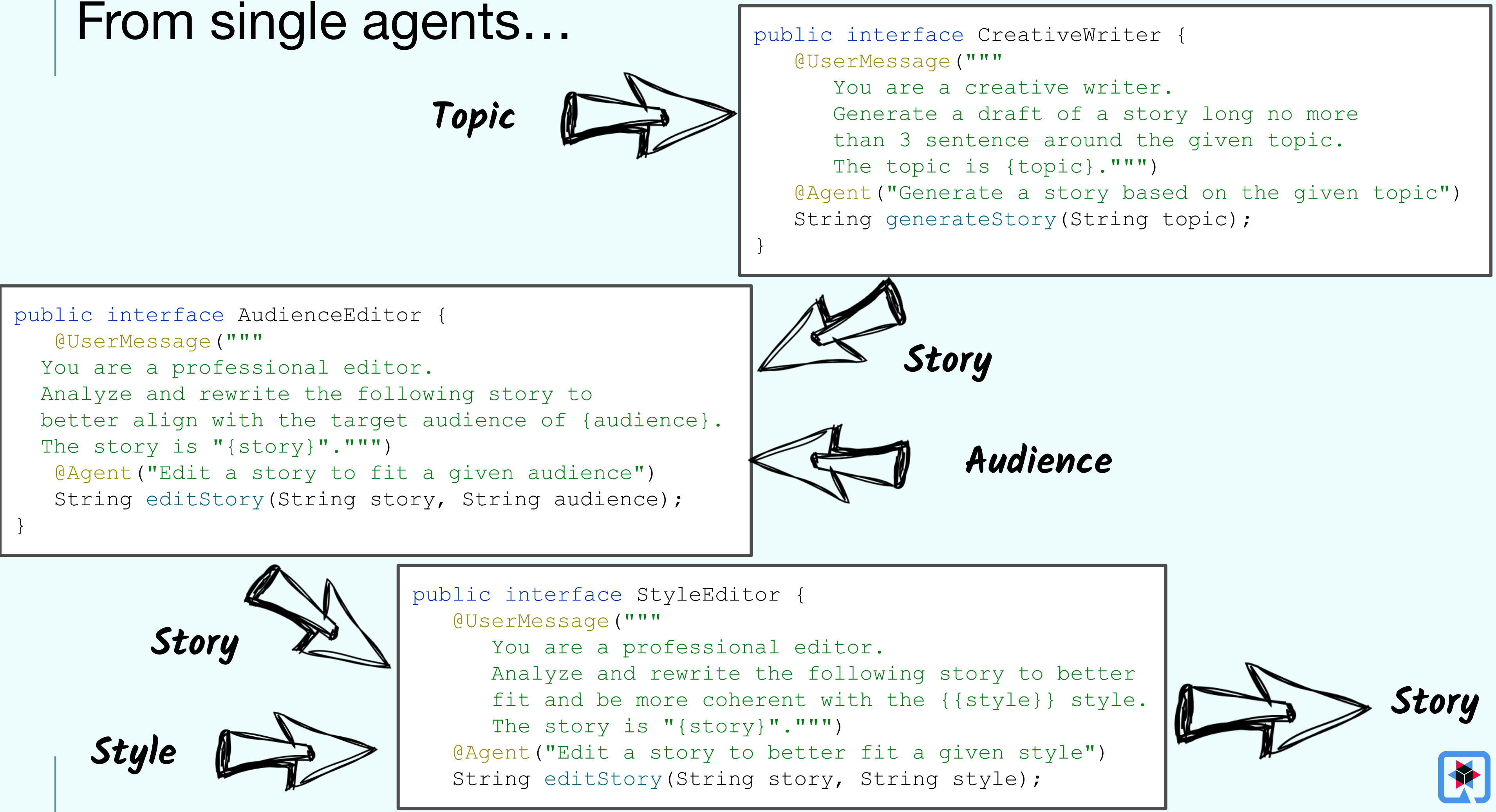
- Sequence / Prompt chaining
- Loop / Reflection
- Parallelization
- Conditional / Routing







From single agents…



```java
public interface CreativeWriter {
   @UserMessage("""
      You are a creative writer.
      Generate a draft of a story long no more
      than 3 sentence around the given topic.
      The topic is {topic}.""")
   @Agent("Generate a story based on the given topic")
   String generateStory(String topic);
}
```

```java
public interface AudienceEditor {
   @UserMessage("""
  You are a professional editor.
  Analyze and rewrite the following story to
  better align with the target audience of {audience}.
  The story is "{story}".""")
   @Agent("Edit a story to fit a given audience")
   String editStory(String story, String audience);
}
```

```java
public interface StyleEditor {
   @UserMessage("""
      You are a professional editor.
      Analyze and rewrite the following story to better
      fit and be more coherent with the {{style}} style.
      The story is "{story}".""")
   @Agent("Edit a story to better fit a given style")
   String editStory(String story, String style);
```

From single agents…

Topic, Audience, Style

```java
public interface CreativeWriter {
   @UserMessage("""
      You are a creative writer.
      Generate a draft of a story long no more
      than 3 sentence around the given topic.
      The topic is {topic}.""")
   @Agent("Generate a story based on the given topic")
   String generateStory(String topic);
}
```

```java
public interface AudienceEditor {
   @UserMessage("""
  You are a professional editor.
  Analyze and rewrite the following story to
  better align with the target audience of {audience}.
  The story is "{story}".""")
   @Agent("Edit a story to fit a given audience")
   String editStory(String story, String audience);
}
```

Story

```java
public interface StyleEditor {
   @UserMessage("""
      You are a professional editor.
      Analyze and rewrite the following story to better
      fit and be more coherent with the {{style}} style.
      The story is "{story}".""")
   @Agent("Edit a story to better fit a given style")
   String editStory(String story, String style);
```

Defining the typed Agentic System

```java
public interface StoryGenerator {

    @Agent("Generate a story based on the given topic,
              for a specific audience and in a specific style")
    String generateStory(String topic, String audience, String style);
}
```

Our Agent System Interface (API):

```java
var story = storyGenerator.generateStory("dragons and wizards",
         "young adults", "fantasy");
```

Sequence Workflow - Defining *agents*

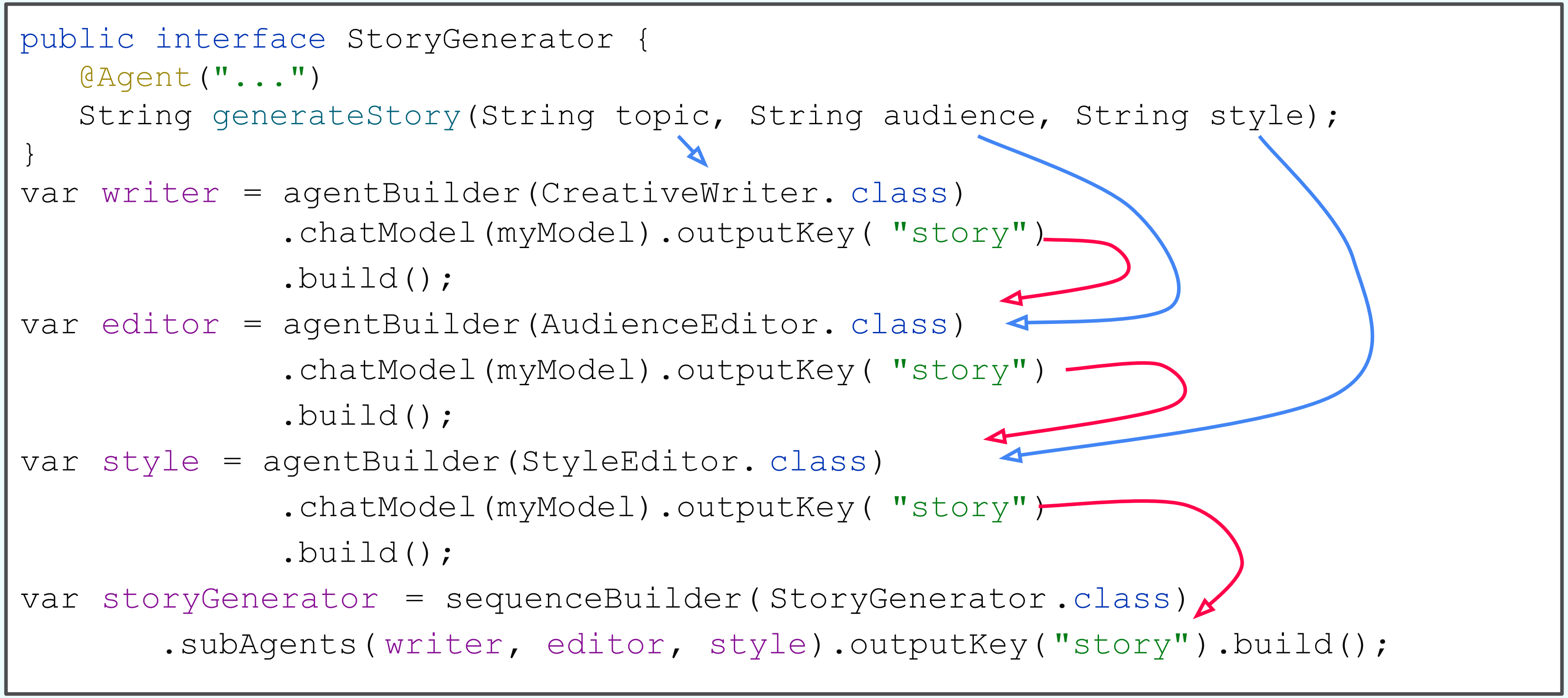
```
var creativeWriter =
            AgenticServices.agentBuilder(CreativeWriter. class)
                .chatModel(myModel).outputKey( "story")
                .build();

var audienceEditor = agentBuilder(AudienceEditor. class)
         .chatModel(myModel).outputKey( "story").build();

var styleEditor = agentBuilder(StyleEditor. class)
         .chatModel(myModel).outputKey( "story").build();
```

Sequence Workflow - Composing Agents

```
var creativeWriter =
          AgenticServices.agentBuilder(CreativeWriter. class)
             .chatModel(myModel).outputKey( "story")
             .build();

var audienceEditor = agentBuilder(AudienceEditor. class)
       .chatModel(myModel).outputKey( "story").build();

var styleEditor = agentBuilder(StyleEditor. class)
       .chatModel(myModel).outputKey( "story").build();

var storyGenerator = AgenticServices.sequenceBuilder( StoryGenerator.class)
       .subAgents(creativeWriter, audienceEditor, styleEditor)
       .outputKey("story").build();
```

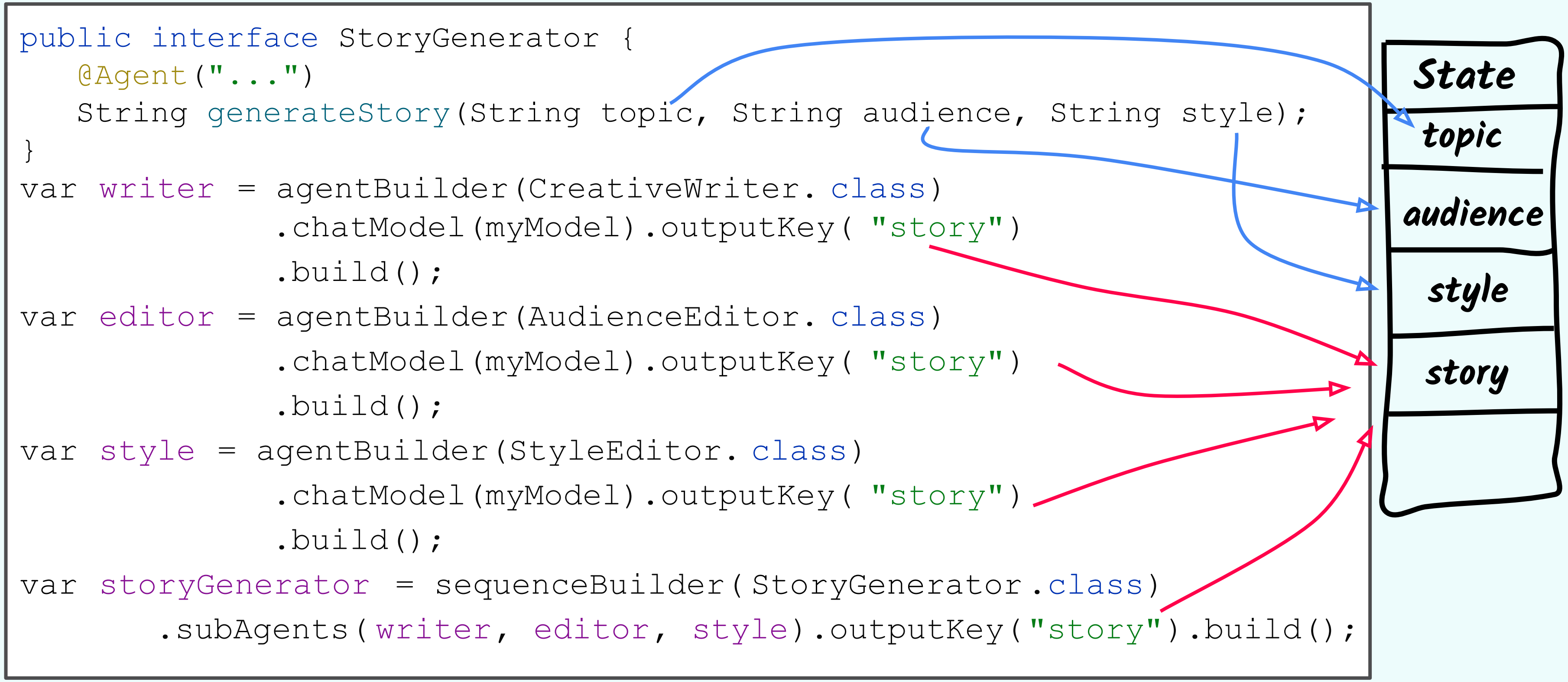
Invoke the system using the StoryGenerator API

Sequence Workflow - Composing Agents



```
public interface StoryGenerator {
    @Agent("...")
    String generateStory(String topic, String audience, String style);
}
var writer = agentBuilder(CreativeWriter. class)
                .chatModel(myModel).outputKey( "story")
                .build();
var editor = agentBuilder(AudienceEditor. class)
                .chatModel(myModel).outputKey( "story")
                .build();
var style = agentBuilder(StyleEditor. class)
                .chatModel(myModel).outputKey( "story")
                .build();
var storyGenerator = sequenceBuilder(StoryGenerator.class)
        .subAgents(writer, editor, style).outputKey("story").build();
```

Sequence Workflow - Composing Agents



```java
public interface StoryGenerator {
   @Agent("...")
   String generateStory(String topic, String audience, String style);
}
var writer = agentBuilder(CreativeWriter. class)
                .chatModel(myModel).outputKey( "story")
                .build();
var editor = agentBuilder(AudienceEditor. class)
                .chatModel(myModel).outputKey( "story")
                .build();
var style = agentBuilder(StyleEditor. class)
                .chatModel(myModel).outputKey( "story")
                .build();
var storyGenerator = sequenceBuilder(StoryGenerator.class)
       .subAgents(writer, editor, style).outputKey("story").build();
```

Introducing the `AgenticScope`

A collection of data shared among the agents participating in the same agentic system

Stores **shared variables**

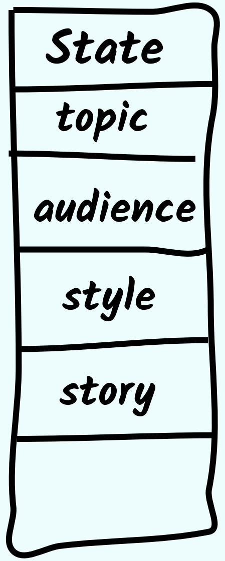
written by an agent to communicate the results it produced

read by another agent to retrieve the necessary to perform its task

Records the sequence of **invocations of all agents** with their responses

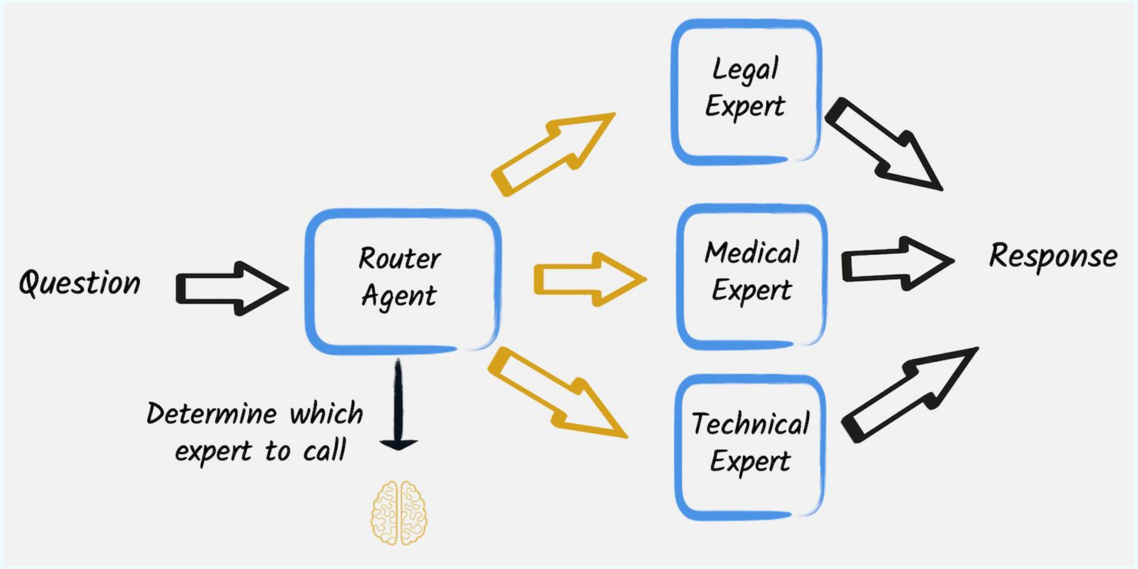
Provides **agentic system wide context** to an agent based on former agent executions

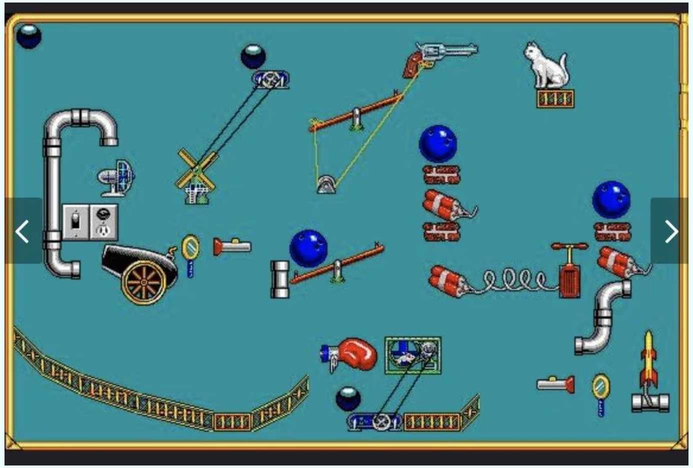
Persistable via a pluggable SPI



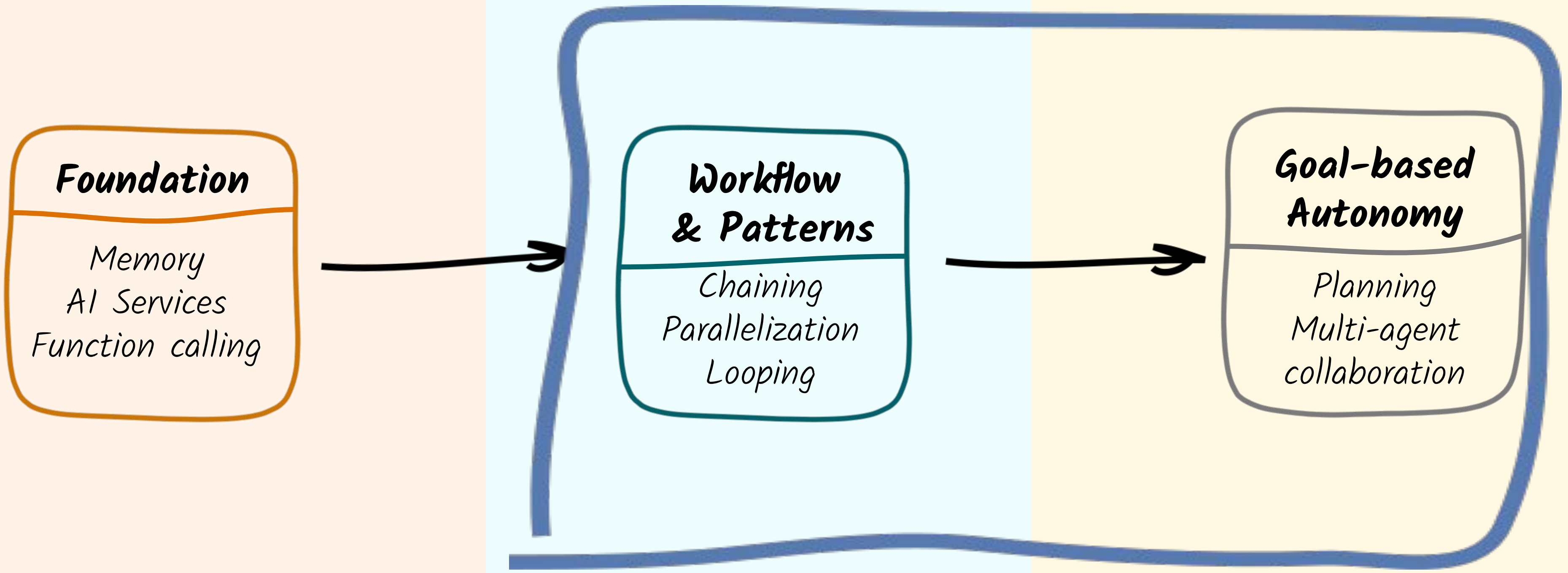
Memory and context engineering

- All agents discussed so far are **stateless**, meaning that they **do not maintain any context** or memory of previous interactions
- AI Services can be provided with a ChatMemory, but this is **local** to the single agent, so in many cases **not enough** in a complex agentic system
- In general an agent requires a **broader context**, carrying information about everything it happened in the agentic system before its invocation
- That's another task for the **AgenticScope**


Question
Router Agent
Determine which expert to call
Legal Expert
Medical Expert
Technical Expert
Response

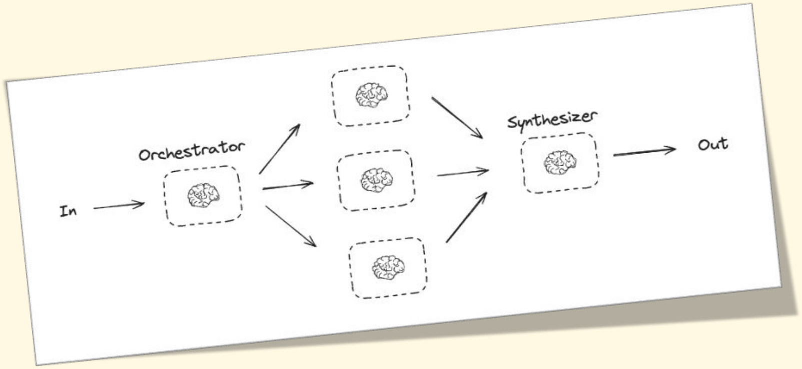


DEMO TIME !!!


Foundation
Memory
AI Services
Function calling
Workflow & Patterns
Chaining
Parallelization
Looping
Goal-based Autonomy
Planning
Multi-agent collaboration

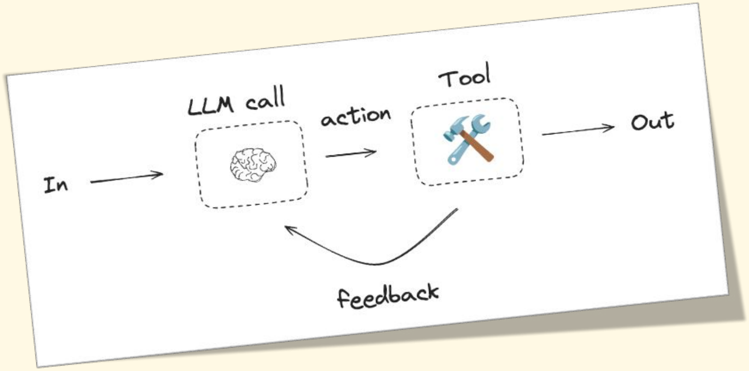
From AI Orchestration to Pure Agentic AI

Workflow



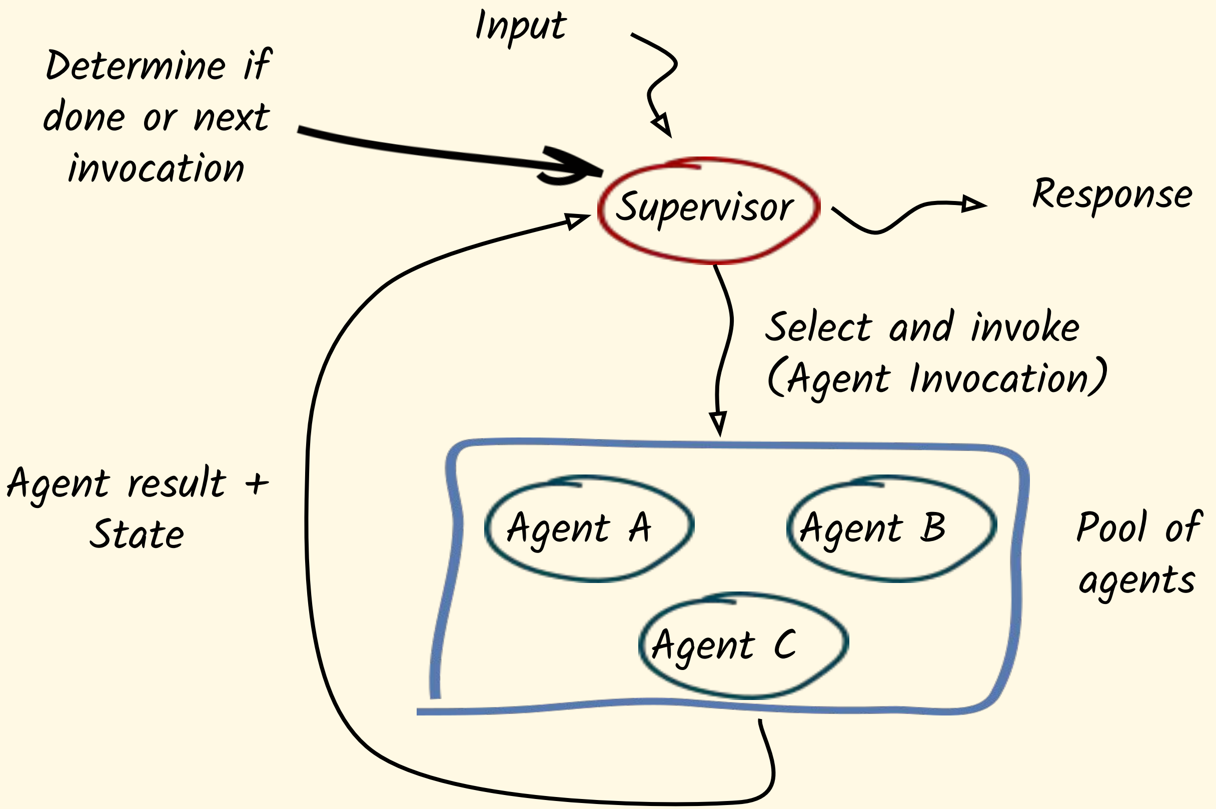
LLMs and tools are **programmatically orchestrated** through predefined code paths and workflows

Agents



LLMs dynamically direct their own processes and tool usage, **maintaining control** over how they execute tasks

A Pure Agentic AI case study – Supervisor pattern

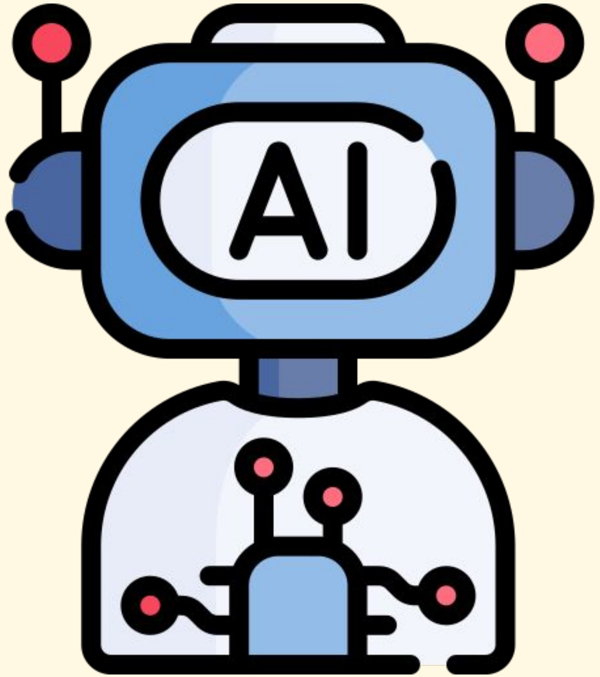


A Pure Agentic AI case study – Supervisor pattern

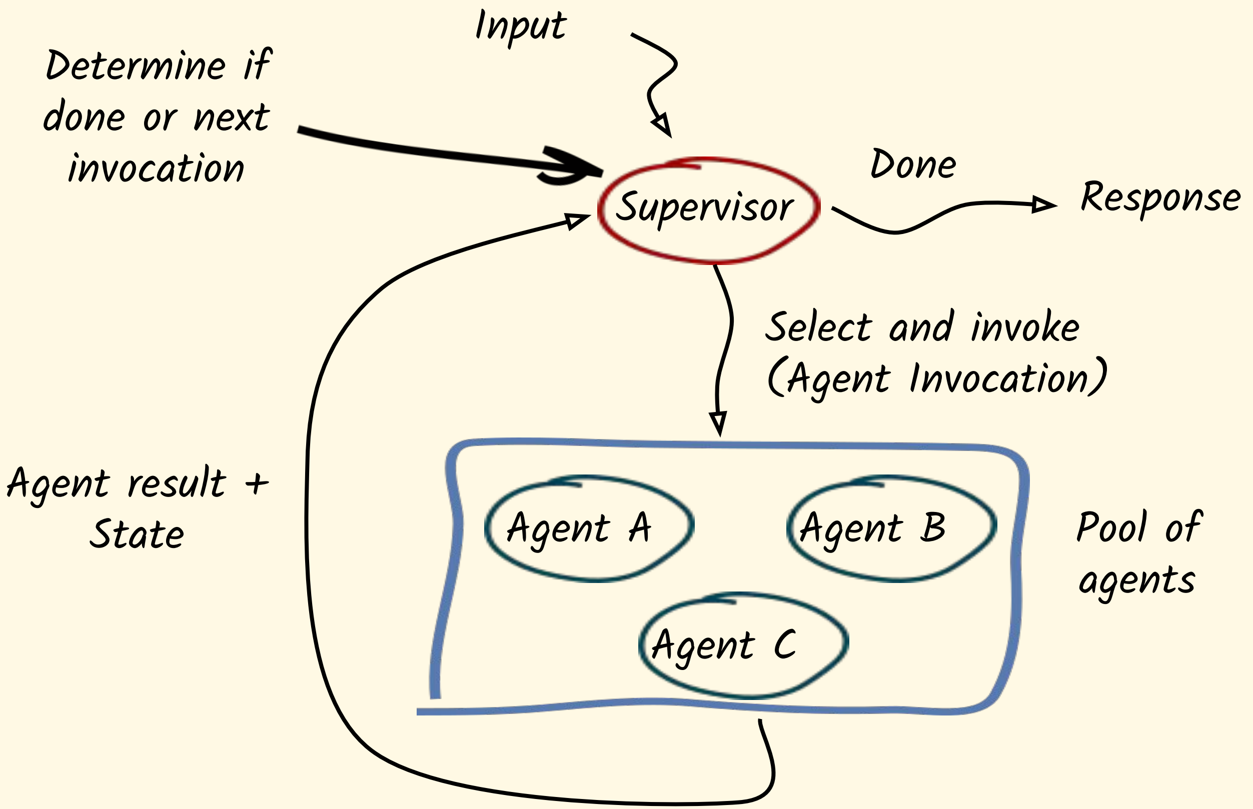
- All agentic systems explored so far orchestrated agents programmatically in a **fully deterministic** way

- In many cases agentic system have to be more **flexible and adaptive**

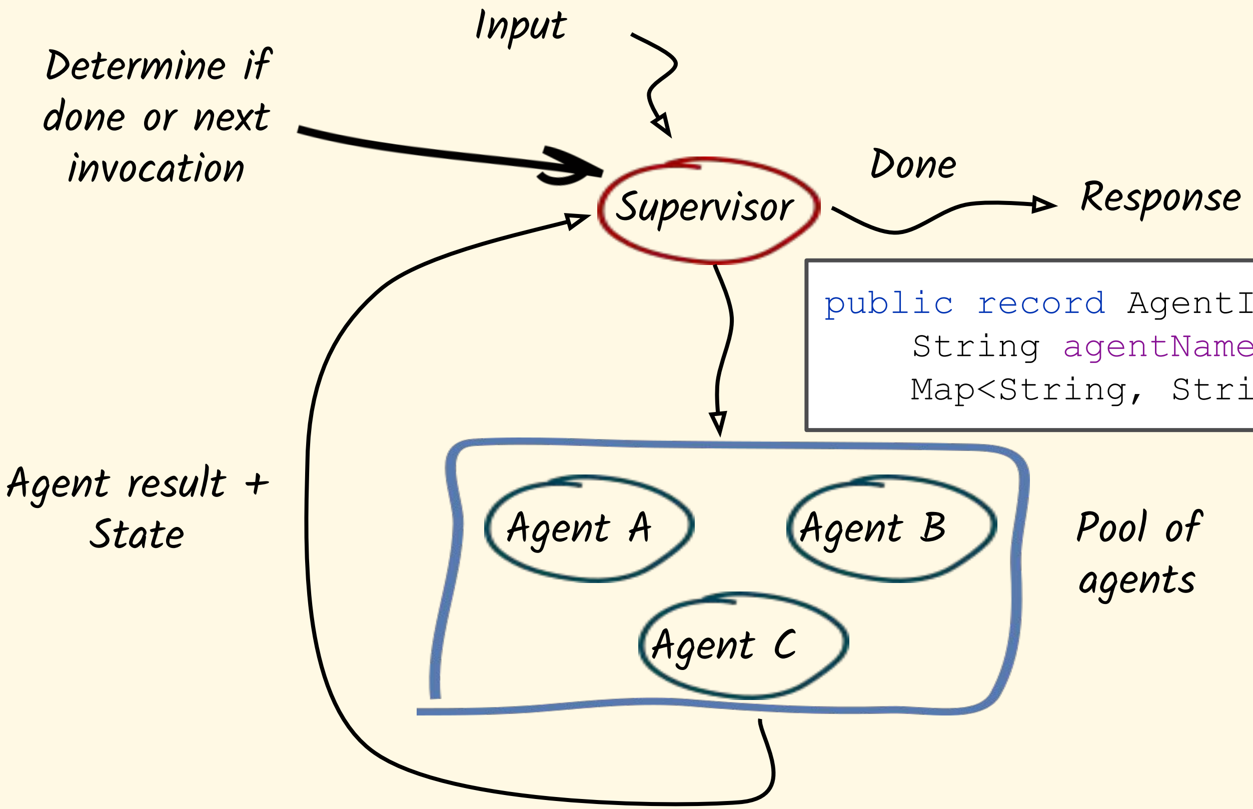
- A pure agentic AI system
  - Takes **autonomous** decisions
  - Decides **iteratively** which agent has to be invoked next
  - Uses the result of previous interactions to determine **if it is done** and achieved its final goal
  - Uses the context and state to generate the **arguments** to be passed to the selected agent



A Pure Agentic AI case study – Supervisor pattern



A Pure Agentic AI case study – Supervisor pattern



```java
public record AgentInvocation(
    String agentName,
    Map<String, String> arguments) { }
```

Supervisor pattern at work - Pool of agents

```java
public interface WithdrawAgent {
   @SystemMessage("You are a banker that can only withdraw US dollars (USD) from a user account.")
   @UserMessage("Withdraw {amountInUSD} USD from {withdrawUser}'s account and return the new balance.")
   @Agent("A banker that withdraw USD from an account")
   String withdraw(String withdrawUser, Double amountInUSD);
}
```

```java
public interface CreditAgent {
   @SystemMessage("You are a banker that can only credit US dollars (USD) to a user account.")
   @UserMessage("Credit {amountInUSD} USD to {creditUser}'s account and return the new balance.")
   @Agent("A banker that credit USD to an account")
   String credit(String creditUser, Double amountInUSD);
}
```

```java
public interface ExchangeAgent {
   @UserMessage("""
   You are an operator exchanging money in different currencies.
   Use the tool to exchange {amount} {originalCurrency} into {targetCurrency}
   returning only the final amount provided by the tool as it is and nothing else.
   """)
   @Agent("A money exchanger that converts a given amount from the original to the target currency")
   Double exchange(String originalCurrency, Double amount, String targetCurrency);
}
```

Supervisor pattern at work - Creating the system

```
BankTool bankTool = new BankTool();
bankTool.createAccount("Mario",1000.0);
bankTool.createAccount("Kevin",1000.0);


WithdrawAgent withdrawAgent = AgenticServices.agentBuilder(WithdrawAgent.class)
       .chatModel(myModel).tools(bankTool).build();

CreditAgent creditAgent = AgenticServices.agentBuilder(CreditAgent.class)
       .chatModel(myModel).tools(bankTool).build();

ExchangeAgent exchange = AgenticServices.agentBuilder(ExchangeAgent.class)
       .chatModel(myModel).tools(new ExchangeTool()).build();


SupervisorAgent bankSupervisor = AgenticServices.supervisorBuilder()
       .chatModel(plannerModel).subAgents(withdrawAgent, creditAgent, exchange).build();
```

Supervisor pattern at work

```java
var result = bankSupervisor.invoke("Transfer 100 EUR from Mario's account to
                                    Kevin's one" );
System.out.println(result);
```

```
100 EUR has been transferred from Mario's account to Kevin's account. Kevin's account
has been credited with 115.0 USD, and the new balance is 1115.0 USD. The withdrawal of
115.0 USD from Mario's account has been completed, and the new balance is 885.0 USD.
```

Supervisor pattern - Agent Invocation Sequence

```
public interface PlannerAgent {
   @SystemMessage(
           """
           …
           """)
   @UserMessage("The user request is: '{req}'. The last received response is: '{lastResponse}'.")
   AgentInvocation plan(@MemoryId Object userId, String agents, String req, String lastResponse, String ctx);
}
```

```
AgentInvocation{agentName='exchange', arguments={originalCurrency=EUR, amount=100, targetCurrency=USD}}

AgentInvocation{agentName='credit', arguments={creditUser=Kevin, amountInUSD=115.0}}

AgentInvocation{agentName='withdraw', arguments={withdrawUser=Mario, amountInUSD=115.0}}

AgentInvocation{agentName='done', arguments={response=100 EUR has been transferred from Mario's account to
Kevin's account. Kevin's account has been credited with 115.0 USD, and the new balance is 1115.0 USD. The
withdrawal of 115.0 USD from Mario's account has been completed, and the new balance is 885.0 USD.}
```

Jetzt Session bewerten!

Einfach QR-Code scannen, Session aus der Liste wählen und bewerten. **Vielen Dank!**

red.ht/rhsc-darmstadt-feedback