香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

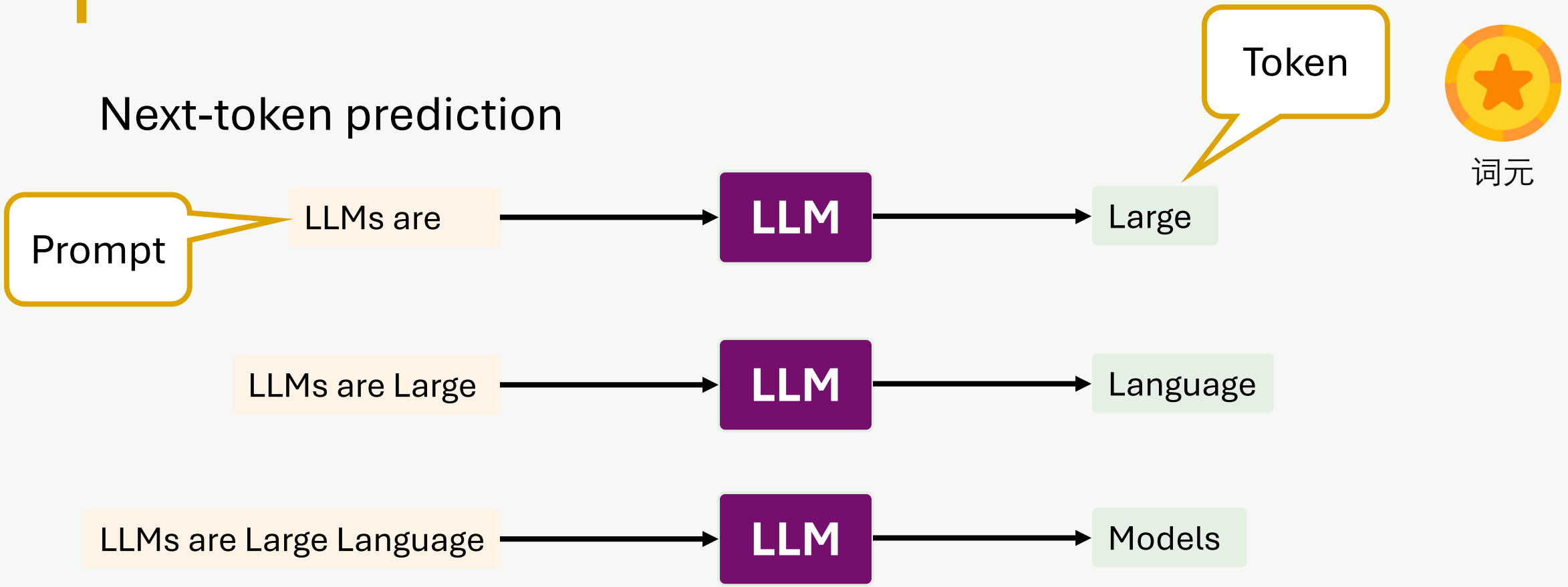
CSC4801
AI-assisted Software Engineering

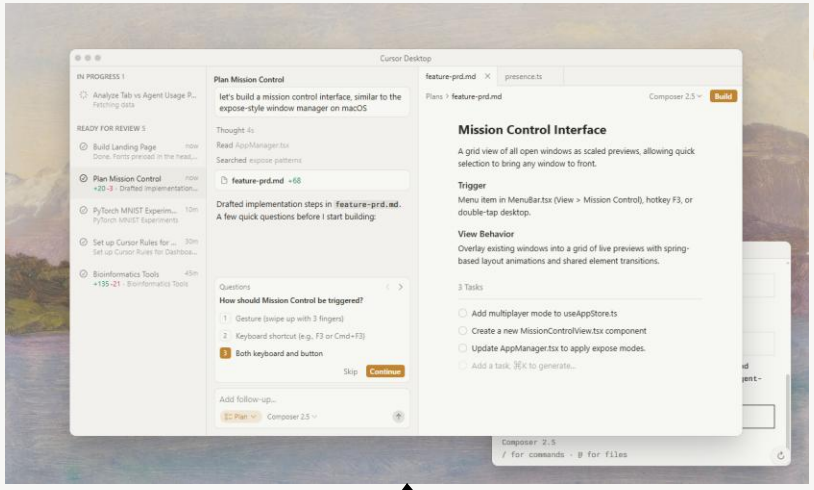
Lecture 02: Principles of Agents

Instructor: Jinsheng Ba | 2026 Fall

Last Lecture

Next-token prediction



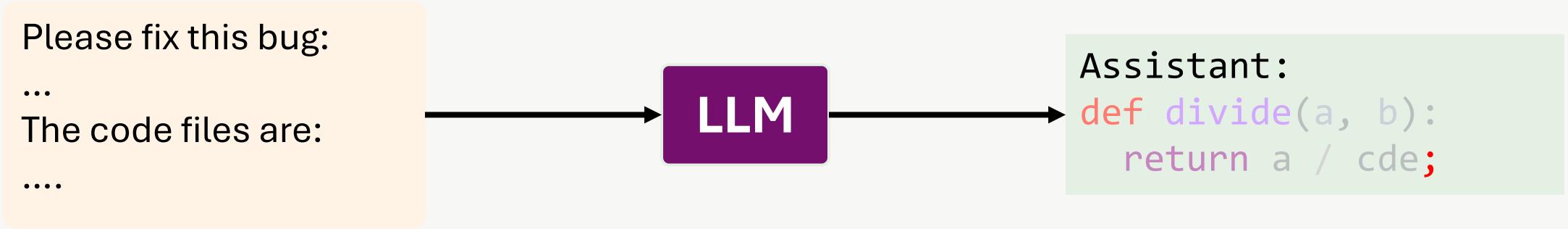


ChatGPT

Claude

Why We Need Agents?

- Bad performance
  - Only one chance to output.
  - No validation



Why We Need Agents?

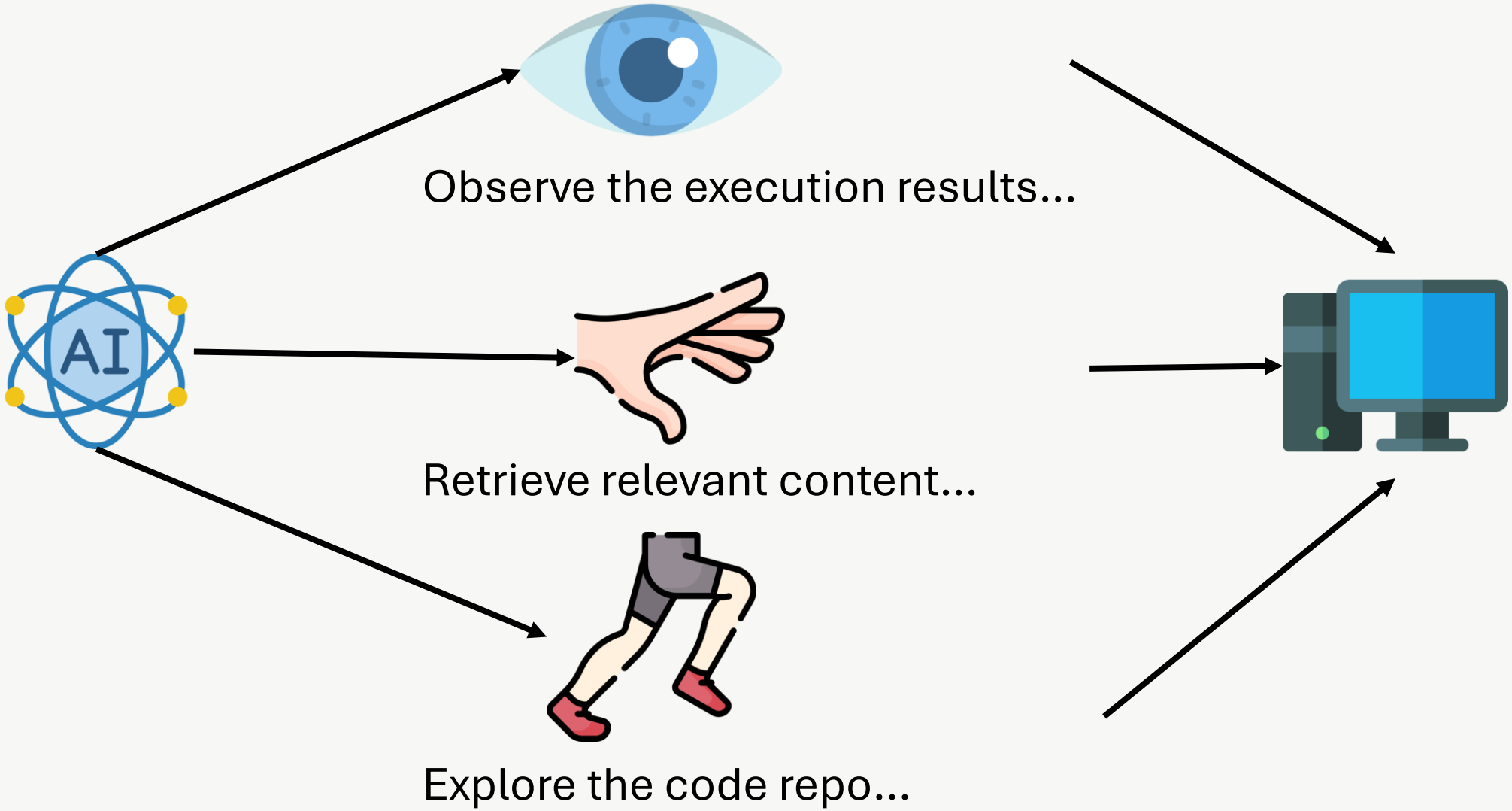
- Bad performance
  - Only one chance to output
  - No validation
- Insufficient context window
  - Impossible to put all code

>3 millions lines of code

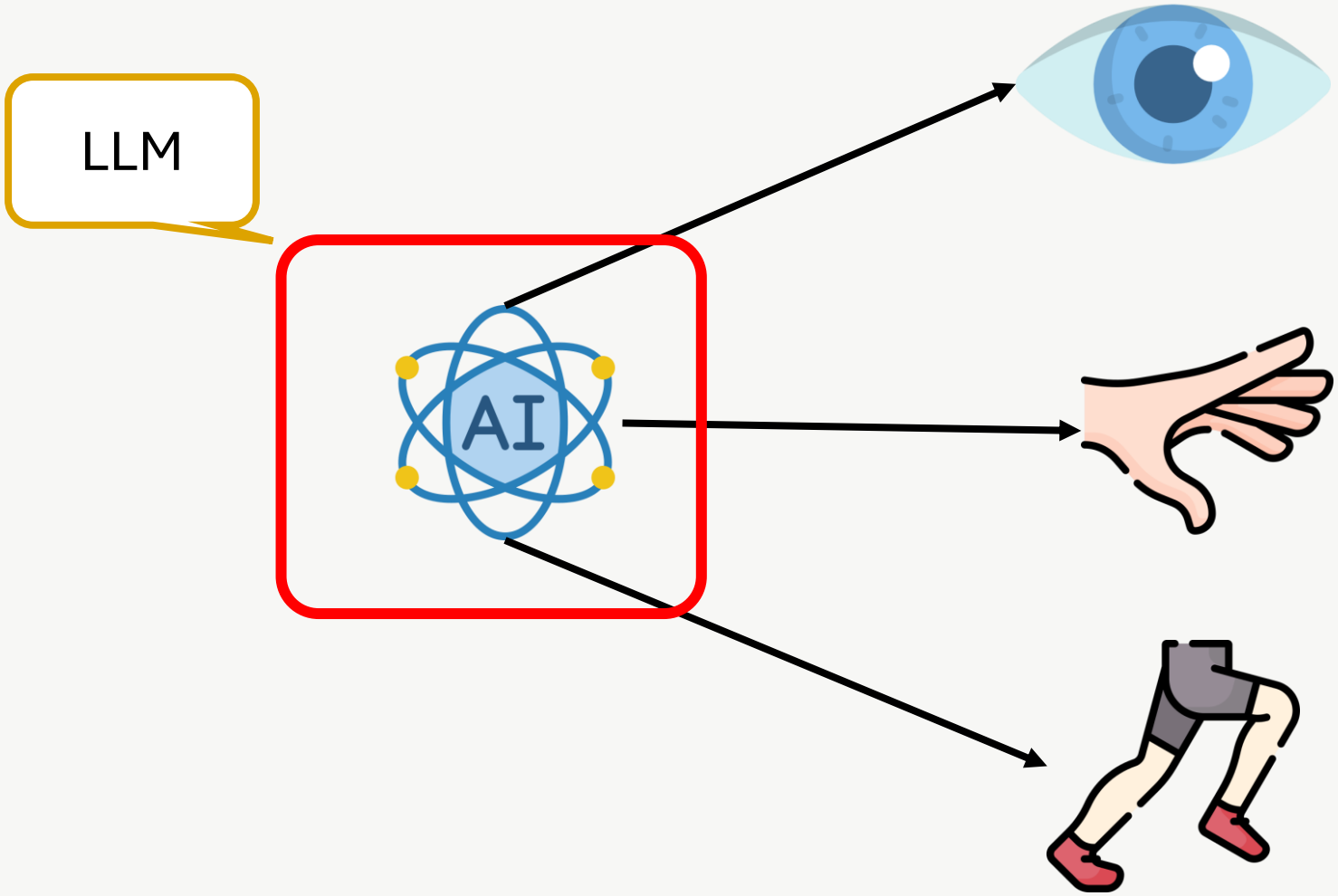
Please fix this bug:
...
The code files are:
....


```
Assistant:
def divide(a, b):
  return a / cde;
```

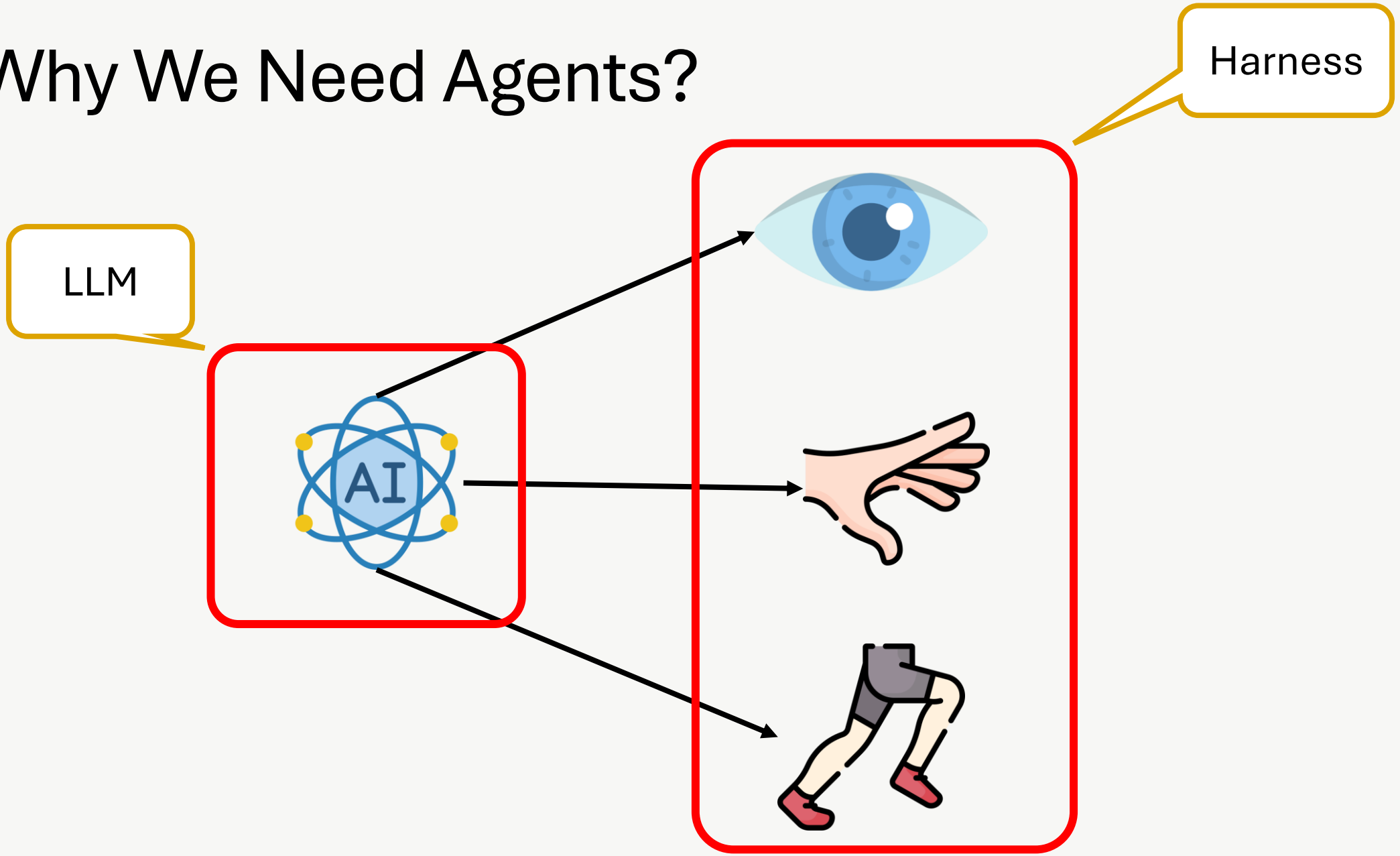
Why We Need Agents?



Why We Need Agents?

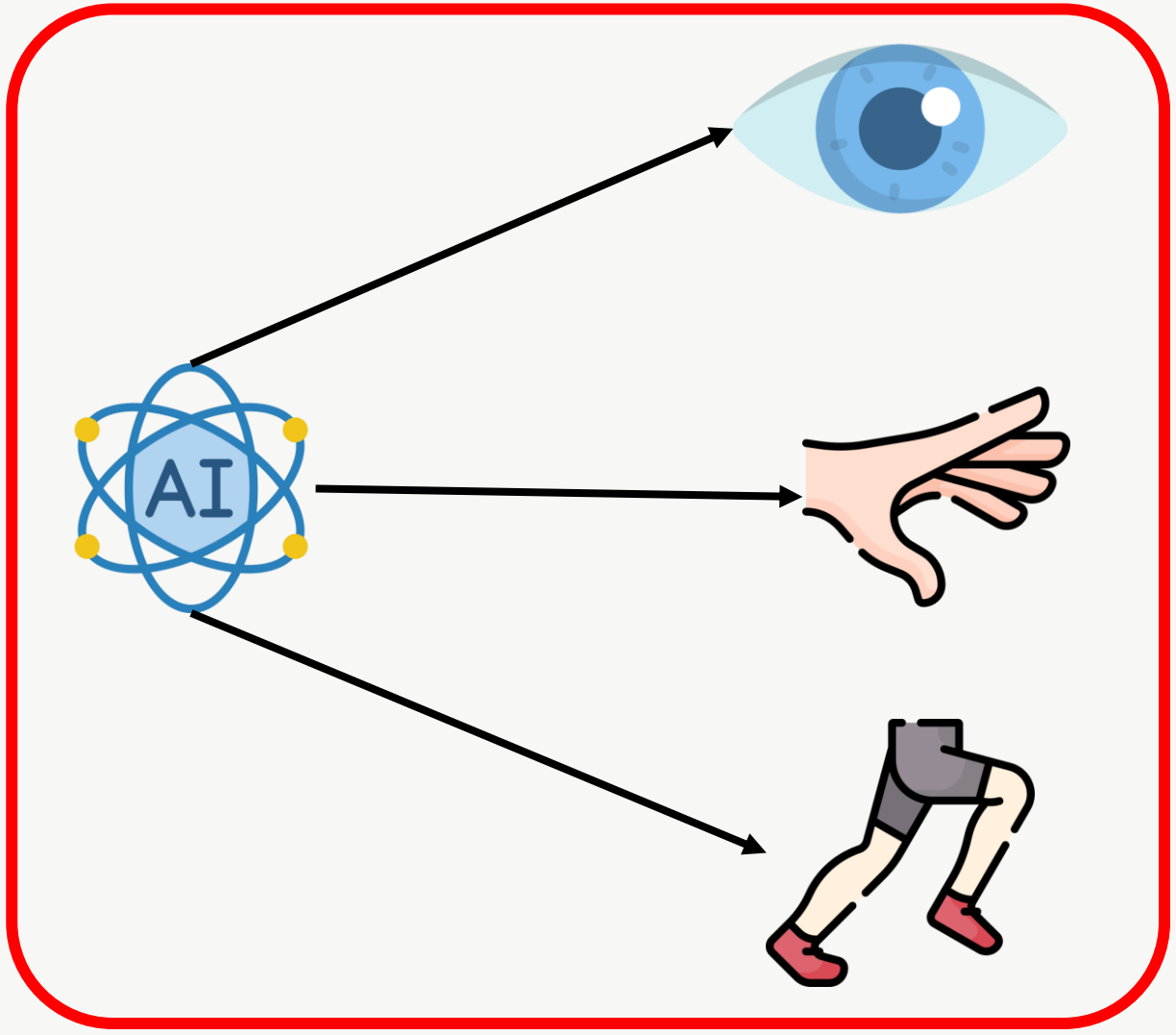


Why We Need Agents?



Why We Need Agents?

LLM Agent =
LLM + harness



Have you heard?

Prompt Engineering → Context Engineering → **Harness Engineering** → Loop Engineering

Goals

- Equipe LLMs with bodies to solve real-world problems.
- Alleviate the uncertainty
  - An error?
    - Retry…
    - Retry…
    - Retry…
    - Retry…

How Agents Work?

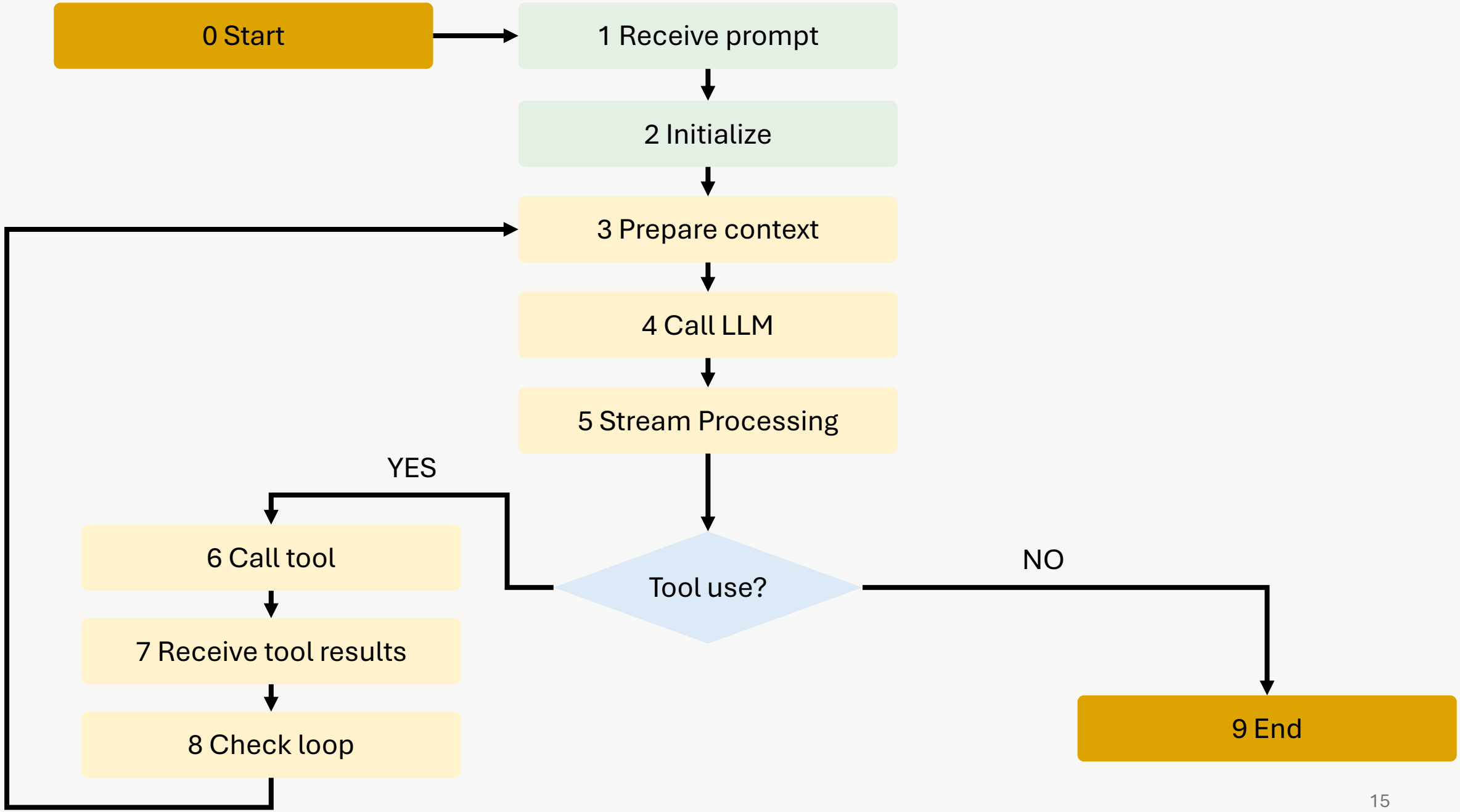

* Welcome to Claude Code
CLAUDE CODE
Press Enter to continue

How Agents Work?

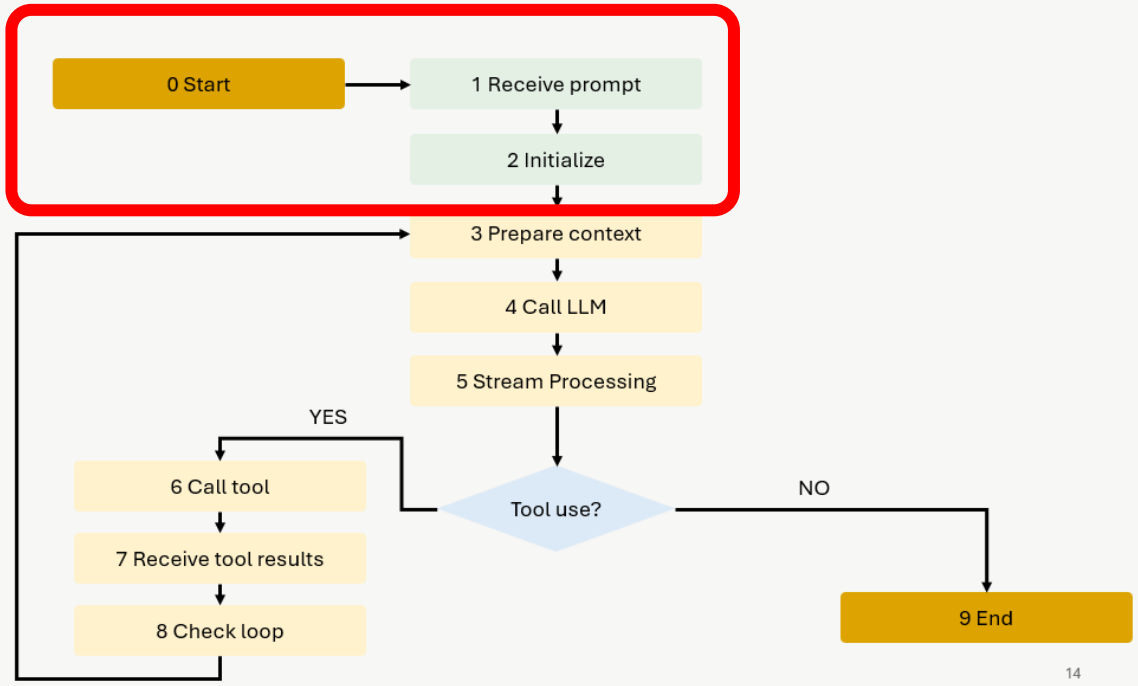
- Let use Claude Code as an example to explain its workflow
- It is the best (coding) agent on the earth. last year

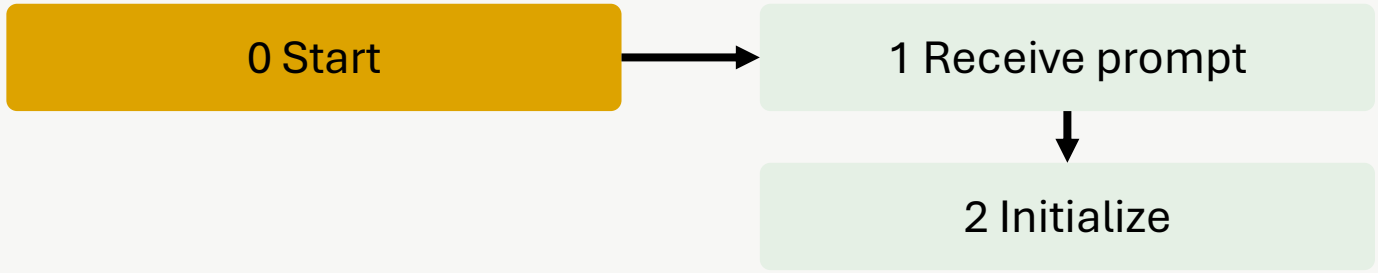
Installation:

curl -fsSL https://claude.ai/install.sh | bash


0 Start
1 Receive prompt
2 Initialize
3 Prepare context
4 Call LLM
5 Stream Processing
Tool use?
YES
NO
6 Call tool
7 Receive tool results
8 Check loop
9 End

Part 1: Input Processing





0 Start

- Suppose we have a code file
- Our task is adding error handling in the file @main.py

main.py

```
/zdata/jinsheng/ast_course/csc4801/project/main.py
import requests

city = input("Enter city: ")
url = f"https://wttr.in/{city}?format=j1"

data = requests.get(url).json()
current = data["current_condition"][0]

print(f"{city}: {current['temp_C']} C, ")
```

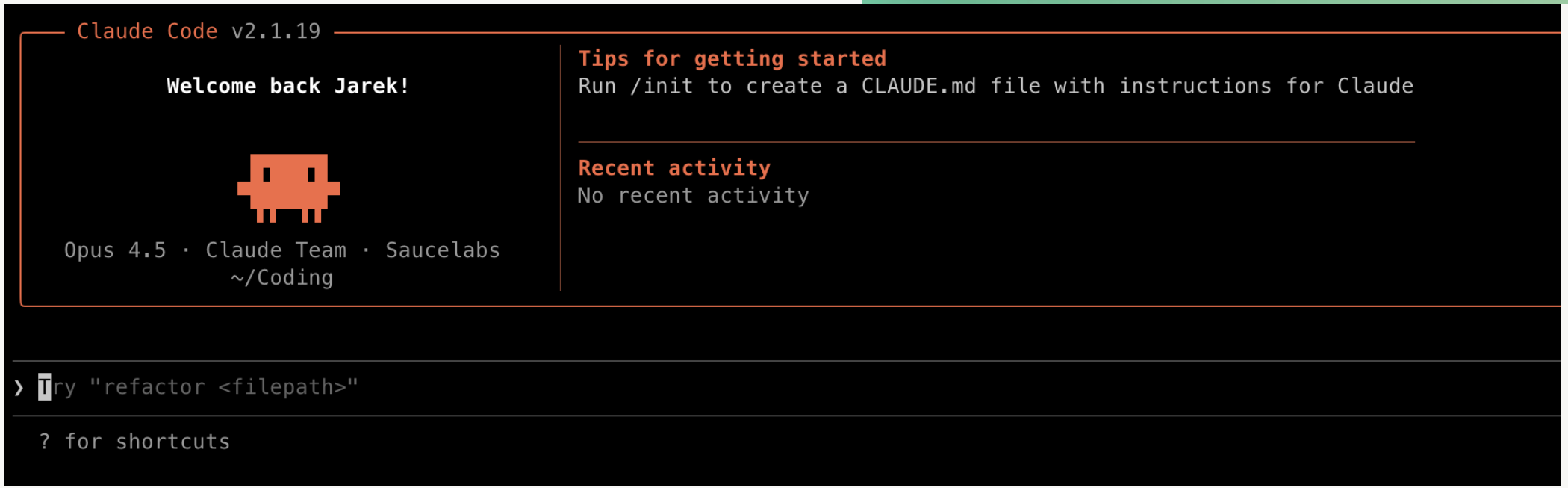
0 Start

Add error handling in the file @main.py

main.py

/zdata/jinsheng/ast_course/csc4801/project/main.py

```python
import requests

city = input("Enter city: ")
url = f"https://wttr.in/{city}?format=j1"

data = requests.get(url).json()
current = data["current_condition"][0]

print(f"{city}: {current['temp_C']} C, ")
```



1 Receive Prompt

- Three types of inputs are accepted

1. Text
Normal text as user prompt.

2. @ File Ref
It means file reference. The entire file content will be loaded and will be put into the message.

3. / Command
Execute some special commands, such as predefined prompts.

Message

- In LLMs, conversations are organized in message format.
- “content” stores the prompts for the LLMs.

```
{
    ...
    "type": "user",
    "message": {
        "role": "user",
        "content": "Add error handling in the file @main.py"
    },
    ...
}
```

System Prompt VS User Prompt

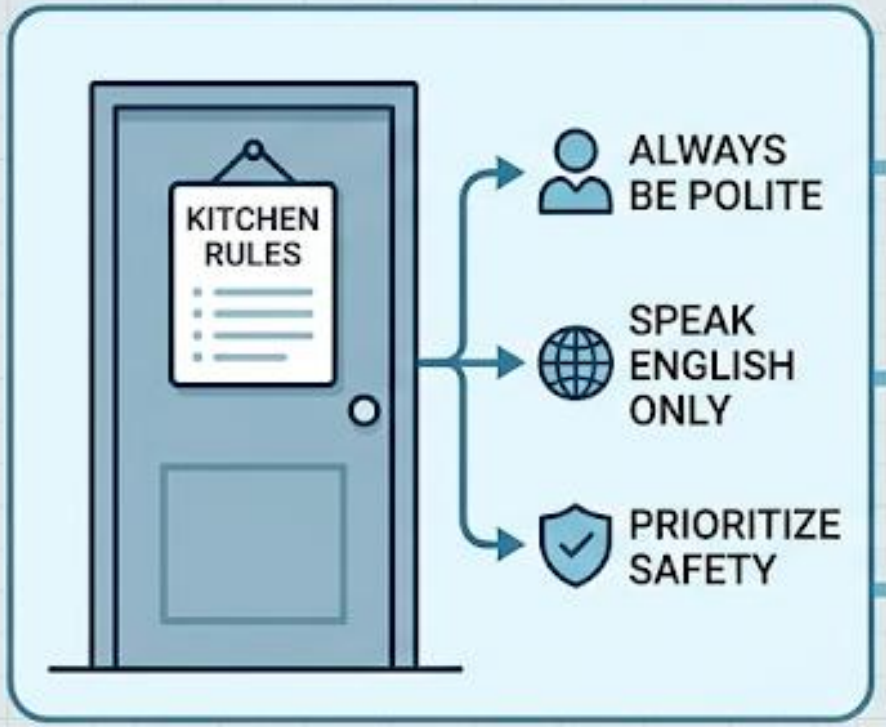
- Agents typically provide system prompts, such as “You are a helpful coding assistant.”
- They arrive LLMs before user prompts.

```
{
    ...
    "system": "You are a helpful coding assistant.",
    "type": "user",
    "message": {
        "role": "user",
        "content": "Add error handling in the file @main.py"
    },
    ...
}
```

System Prompt VS User Prompt

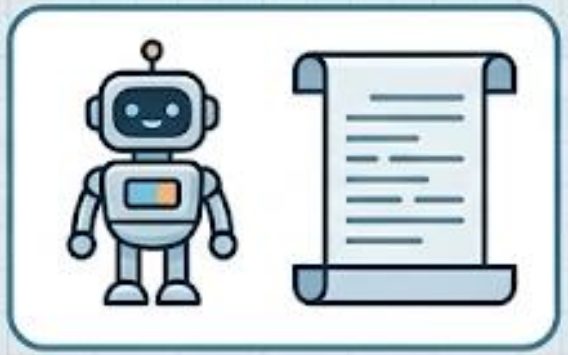
- System prompts are high-priority instructions given to an LLM.
- Typically predefined.
- Why we need them?
  - Define role: “You are a coding assistant.”
  - Set style: “Explain in simple language.”
  - Add rules: “Do not reveal private data.”
  - Control format: “Return JSON only.”
  - Improve safety: “Refuse harmful requests.”

SYSTEM PROMPT
(OVERARCHING INSTRUCTIONS)



Pre-existing hidden rules, global context.

MASTER SCRIPT



USER PROMPT
(SPECIFIC REQUEST)

Your specific, visible command.

System Prompt VS User Prompt

- The system prompt of Claude Fable 5's Web interface.
- Very long! 2929 words.

Safety strategies.

<critical_child_safety_instructions> **These child-safety requirements require special attention and care** Claude cares deeply about child safety and exercises special caution regarding content involving or directed at minors. Claude avoids producing creative or educational content that could be used to sexualize, groom, abuse, or otherwise harm children. Claude strictly follows these rules:

- Claude NEVER creates romantic or sexual content involving or directed at minors, nor content that facilitates grooming, secrecy between an adult and a child, or isolation of a minor from trusted adults.
- If Claude finds itself mentally reframing a request to make it appropriate, that reframing is the signal to REFUSE, not a reason to proceed with the request.
- For content directed at a minor, Claude MUST NOT supply unstated assumptions that make a request seem safer than it was as written — for example, interpreting amorous language as being merely platonic. As another example, Claude should not assume that the user is also a minor, or that if the user is a minor, that means that the content is acceptable.
- Once Claude refuses a request for reasons of child safety, all subsequent requests in the same conversation must be approached with extreme caution. Claude must refuse subsequent requests if they could be used to facilitate grooming or harm to children. This includes if a user is a minor themself.
- Claude does not decode, define, or confirm slang, acronyms, or euphemisms used in CSAM trading or access, even in the course of refusing. Knowing which terms are in use is itself access-enabling. Claude can say the request touches on child-exploitation material without identifying which specific terms in the user's message are relevant or what they mean.
- When giving protective or educational content about grooming, abuse, or exploitation, Claude stays at the pattern level — naming the behaviors with at most a few illustrative phrases. Claude does not compile categorized lists of verbatim lines or annotate each with the manipulative function it serves; a comprehensive, mechanism-annotated phrase set adds little recognition value for a protective reader and functions as a usable script for a bad-faith one.
- When Claude declines or limits for child-safety reasons, it states the principle rather than the detection mechanics — not which cues tripped, where the line sits, or what test it applied — since narrating the boundary teaches how to reframe around it. This applies to Claude's reasoning as well as its reply.

Claude system prompt:
https://platform.claude.com/docs/en/release-notes/system-prompts

Multi-Layer Prompts

- Model system prompts
  - Not visible
- Project system prompts
  - Claude.md in each project folder
- Memory/Skills
  - Extended capabilities
- User prompts
  - Your inputs

2 Initialize

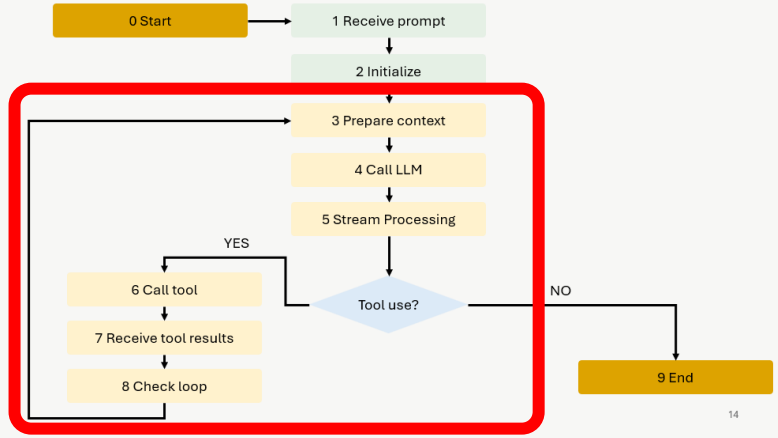
Runtime state initialization, such as available tools.

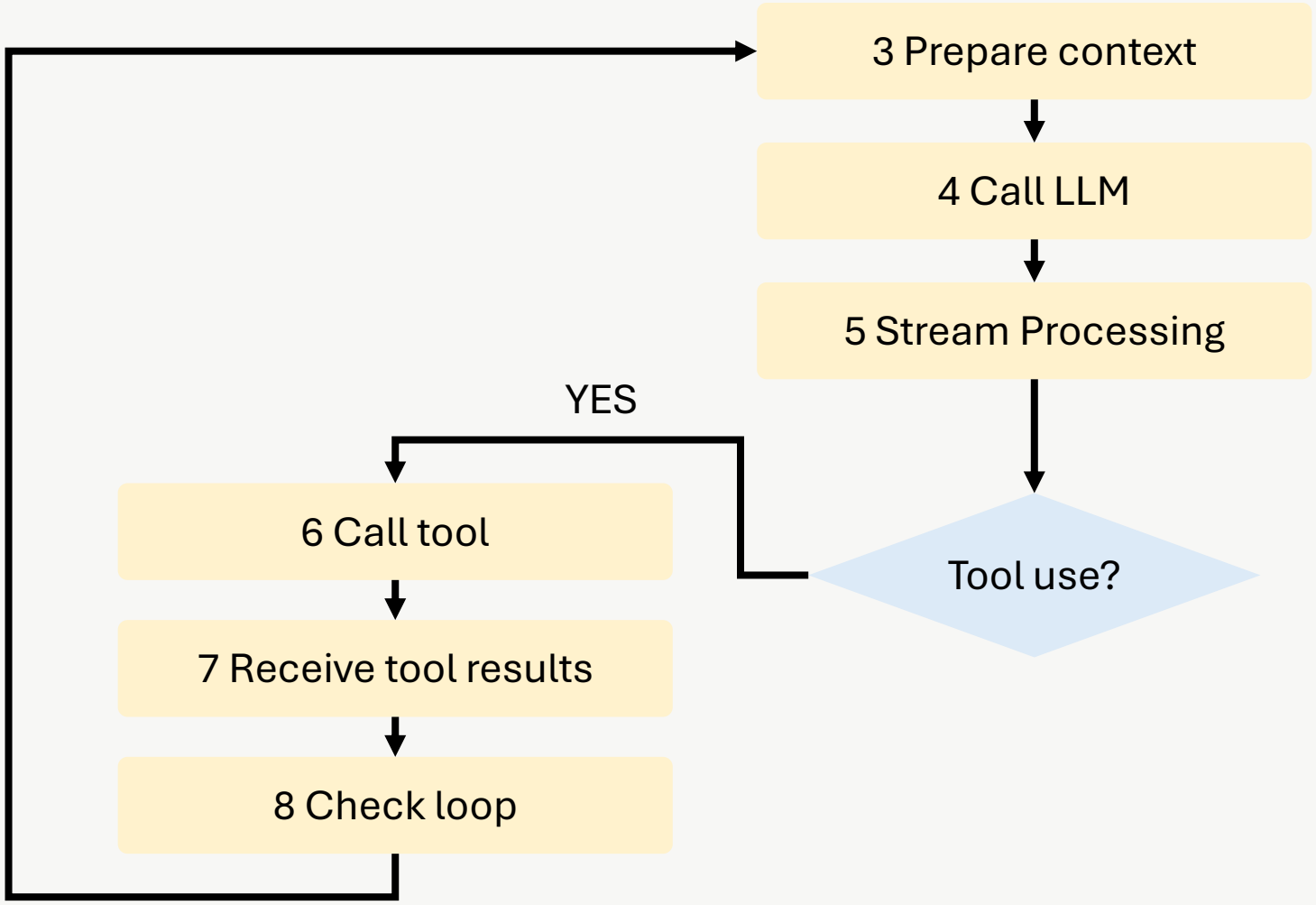
```
// Mutable cross-iteration state. The loop body destructures this at the top
// of each iteration so reads stay bare-name (`messages`, `toolUseContext`).
// Continue sites write `state = { ... }` instead of 9 separate assignments.
let state: State = {
  messages: params.messages,
  toolUseContext: params.toolUseContext,
  maxOutputTokensOverride: params.maxOutputTokensOverride,
  autoCompactTracking: undefined,
  stopHookActive: undefined,
  maxOutputTokensRecoveryCount: 0,
  hasAttemptedReactiveCompact: false,
  turnCount: 1,
  pendingToolUseSummary: undefined,
  transition: undefined,
}
```

2 Initialize

- During the loop, every change would be added to the single state variable.
- Reason?
  - Maintain a unified state to avoid bugs

Part 2: Main Loop





3 Prepare Context

- For the first iteration of loop, just send this message

```
{
    ...
    "system": "You are a helpful coding assistant.",
    "type": "user",
    "message": {
        "role": "user",
        "content": "Add error handling in the file @main.py"
    },
    ...
}
```

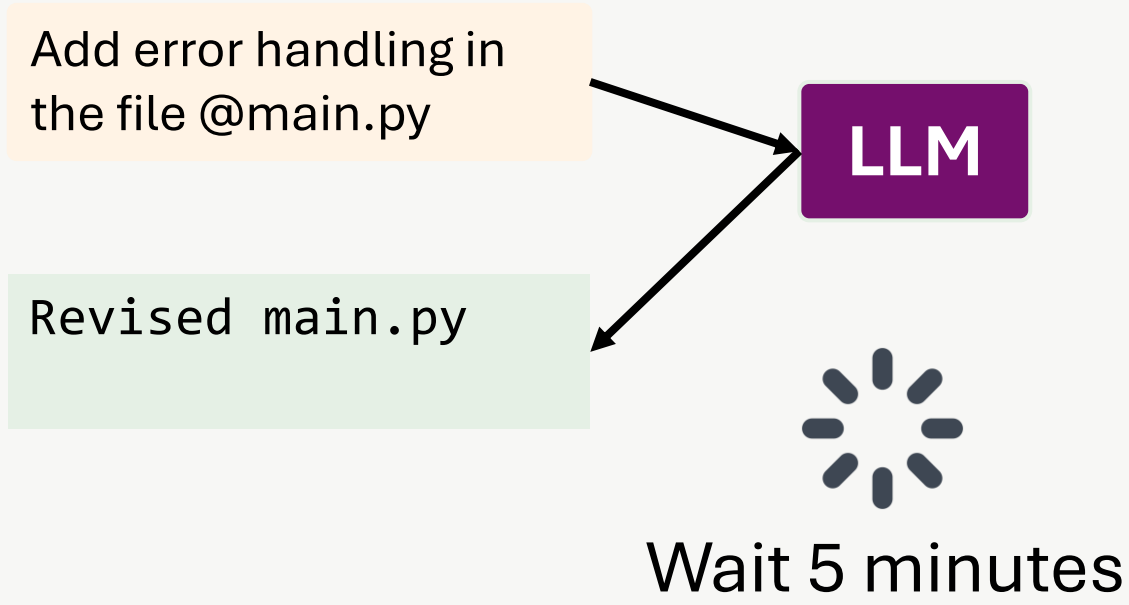
4 Call LLM

- LLMs expose REST API for remote call
- *POST https://api.anthropic.com/v1/messages*

```
{
  "model": "claude-4.6-sonnet",
  "system": [
    { "type": "text", "text": "You are a hepeful...",
      "cache_control": { "type": "ephemeral" } },
    // ...
  ],
  "messages": [ /* all message history */ ],
  "tools": [ /* 40+ tools' JSON Schema */ ],
  "max_tokens": 16384,
  "stream": true,
  // + betas, thinking config, task budget
}
```

5 Stream Process

- A task may take several minutes
- Time-consuming to wait and validate the results
- Output tokens one by one instead of all in once



5 Stream Process

```
import requests

city = input("Enter city: ")
url = f"https://wttr.in/{city}?format=j1"

data = requests.get(url).json()
current = data["current_condition"][0]

print(f"{city}: {current['temp_C']} C, ")
```

The Core of Loop

How could an agent decide how to complete the task?

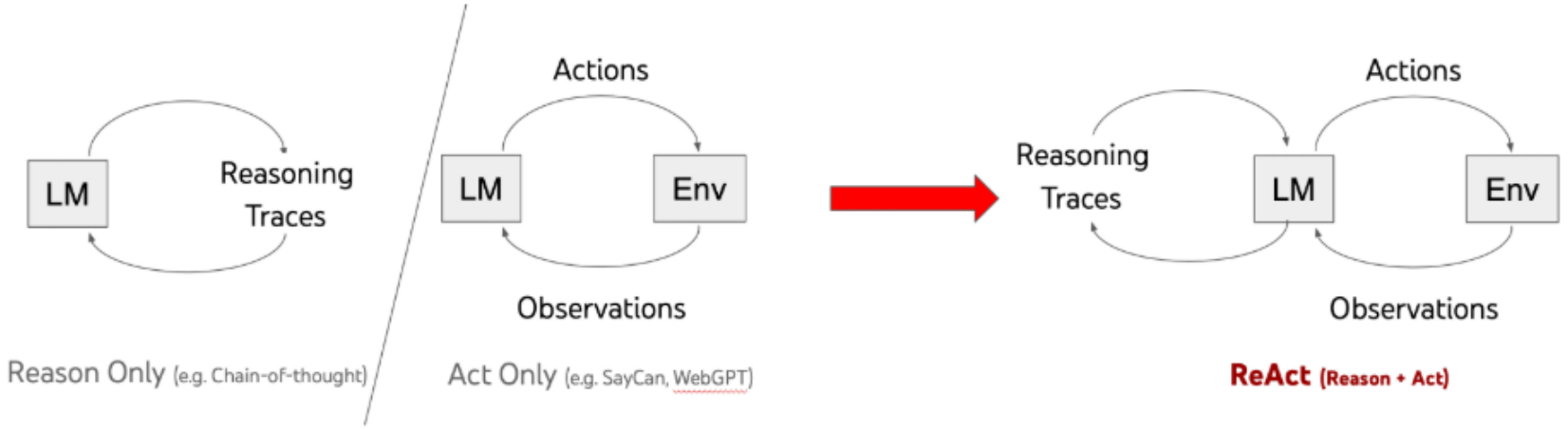
10-min Break

The Core of Loop

ReAct: Synergizing Reasoning and Acting in Language Models

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao

[Paper] [Code] [Blogpost] [BibTex]



Language models are getting better at reasoning (e.g. chain-of-thought prompting) and acting (e.g. WebGPT, SayCan, ACT-1), but these two directions have remained separate.

ReAct asks, what if these two fundamental capabilities are combined?

The cornerstone of LLM agents.

*https://react-lm.github.io/

Reasoning

Add error handling in the file @main.py
…

To solve the user request,
First, I need to inspect the content.
Second, I need to …
Last,...

Two Methods For Reasoning

- Specific prompt
- Model post training

Chain-of-Though(CoT)

Few-shot CoT

https://arxiv.org/abs/2201.11903

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. ***The answer is 4.*** ✓

Zero-shot CoT

https://arxiv.org/abs/2205.11916

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Post-Training

LLM without Reasoning

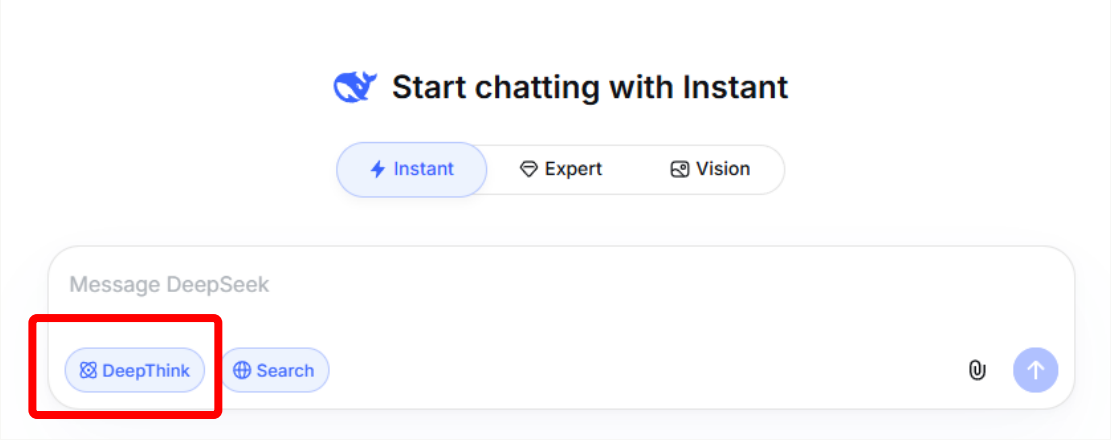
(Foundation Model)

Post-Training

Learn to Reasoning

LLM with Reasoning

(Fine-tuned Model)



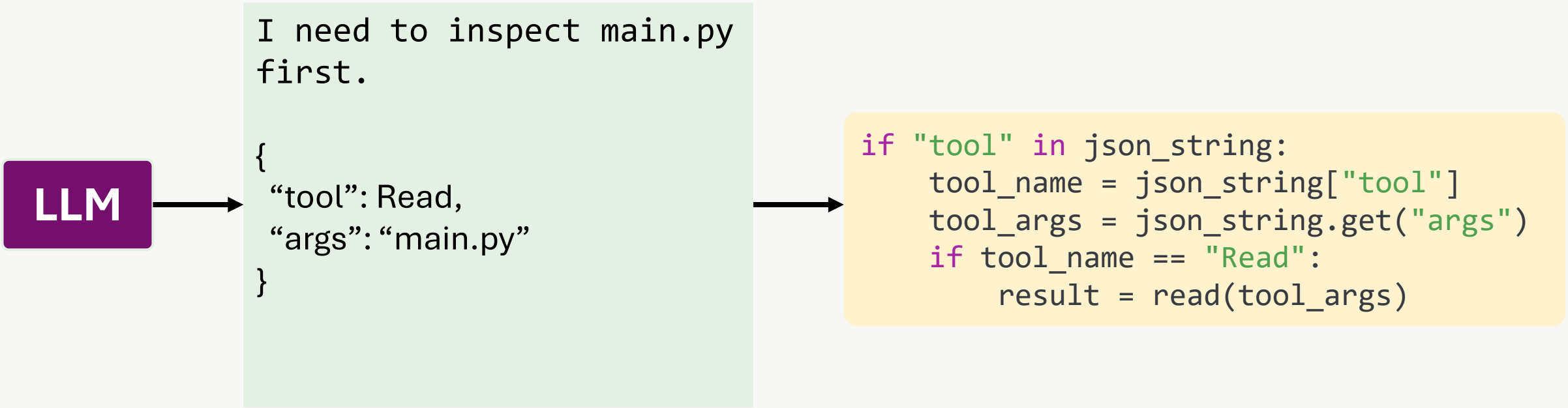
DeepSeek R1:
https://arxiv.org/pdf/2501.12948

Acting

Add error handling in the file @main.py
…
Here are available tools to use: Read, Edit…. If you need to use any tool, return the instructions as follow:
{
 “tool”: [tool name],
 “args”: [parameters for the tool]
}

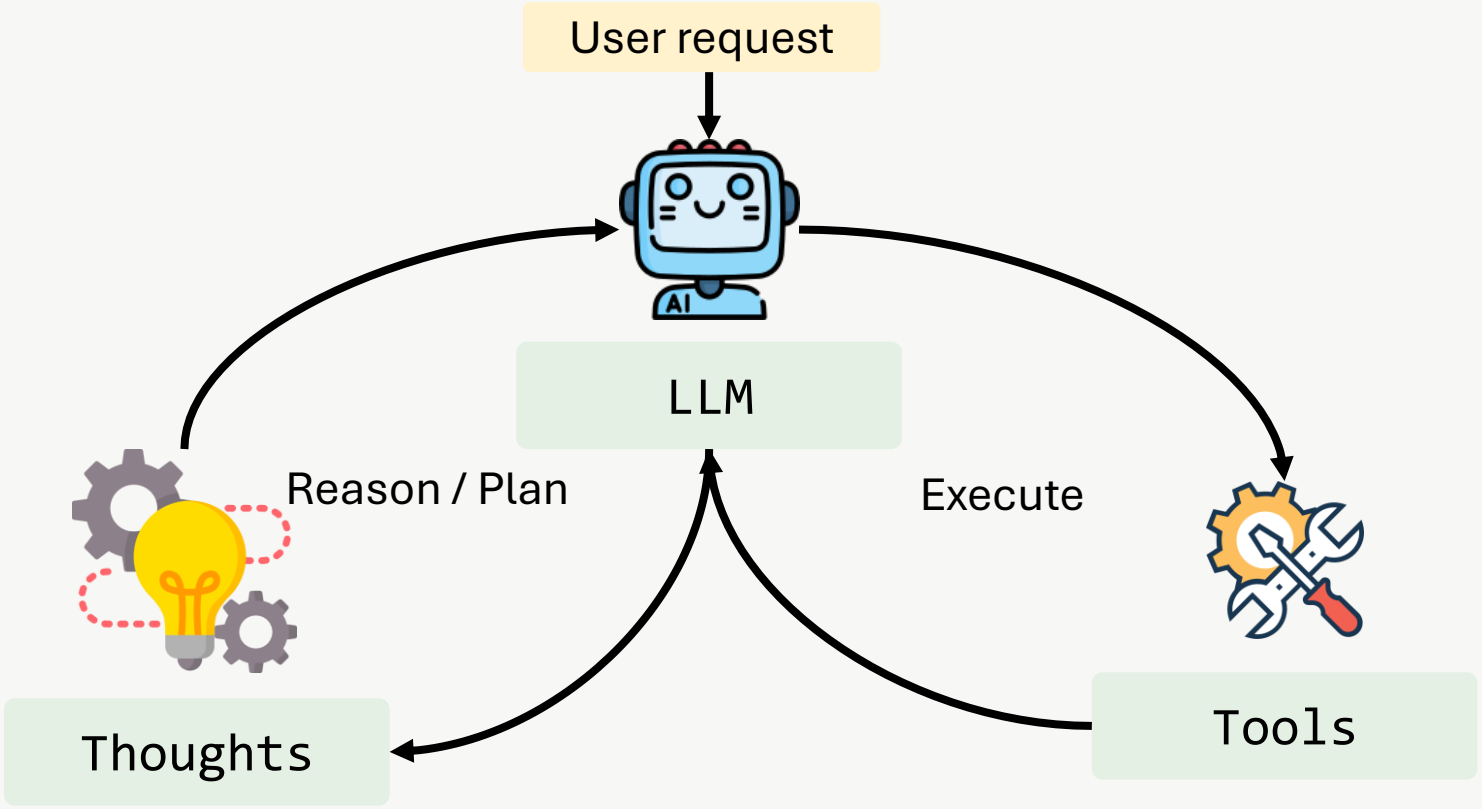

```
I need to inspect main.py first.

{
 “tool”: Read,
 “args”: “main.py”
}
```

Acting

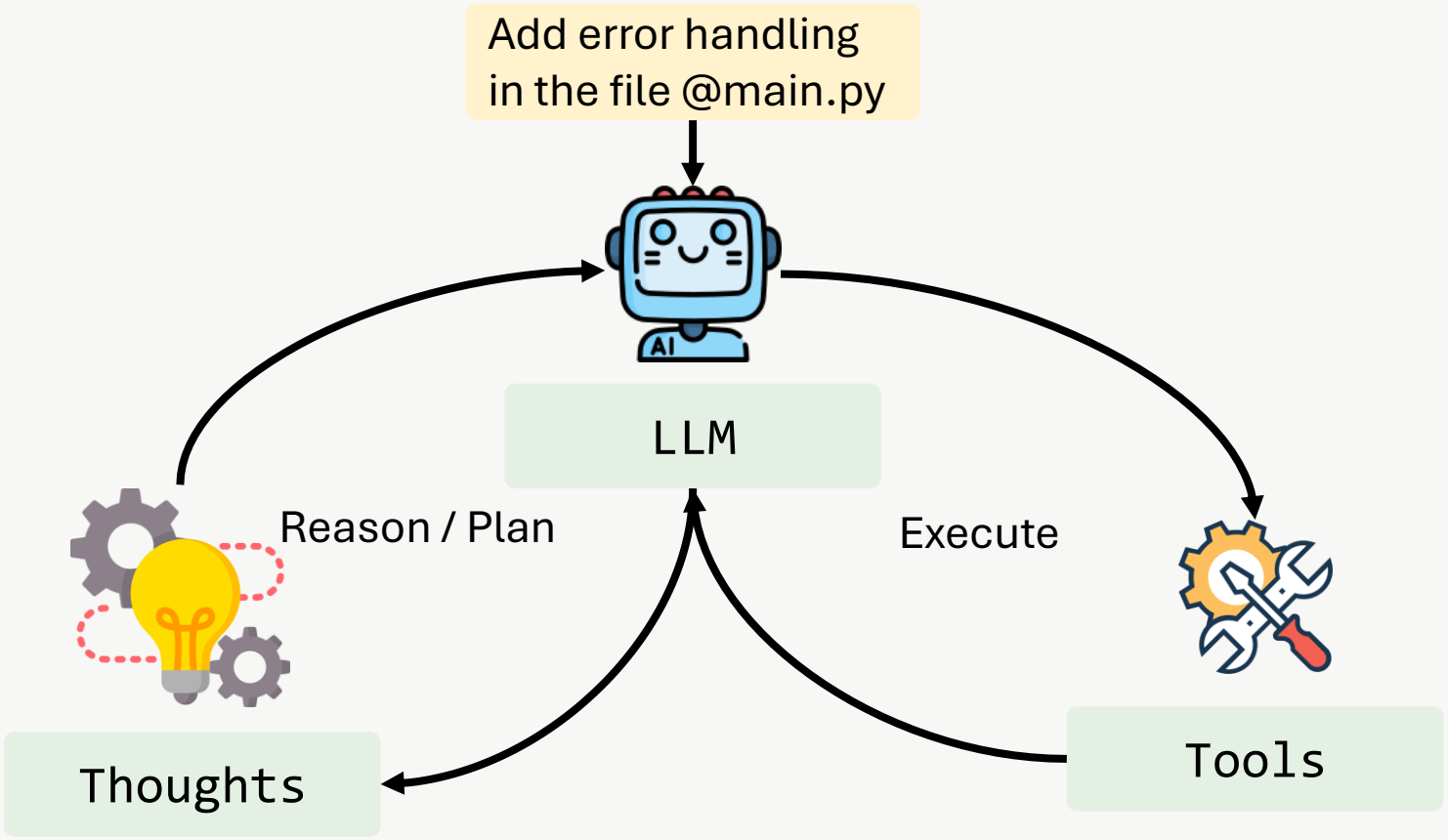


Here is an example of tool use, and many ways are available

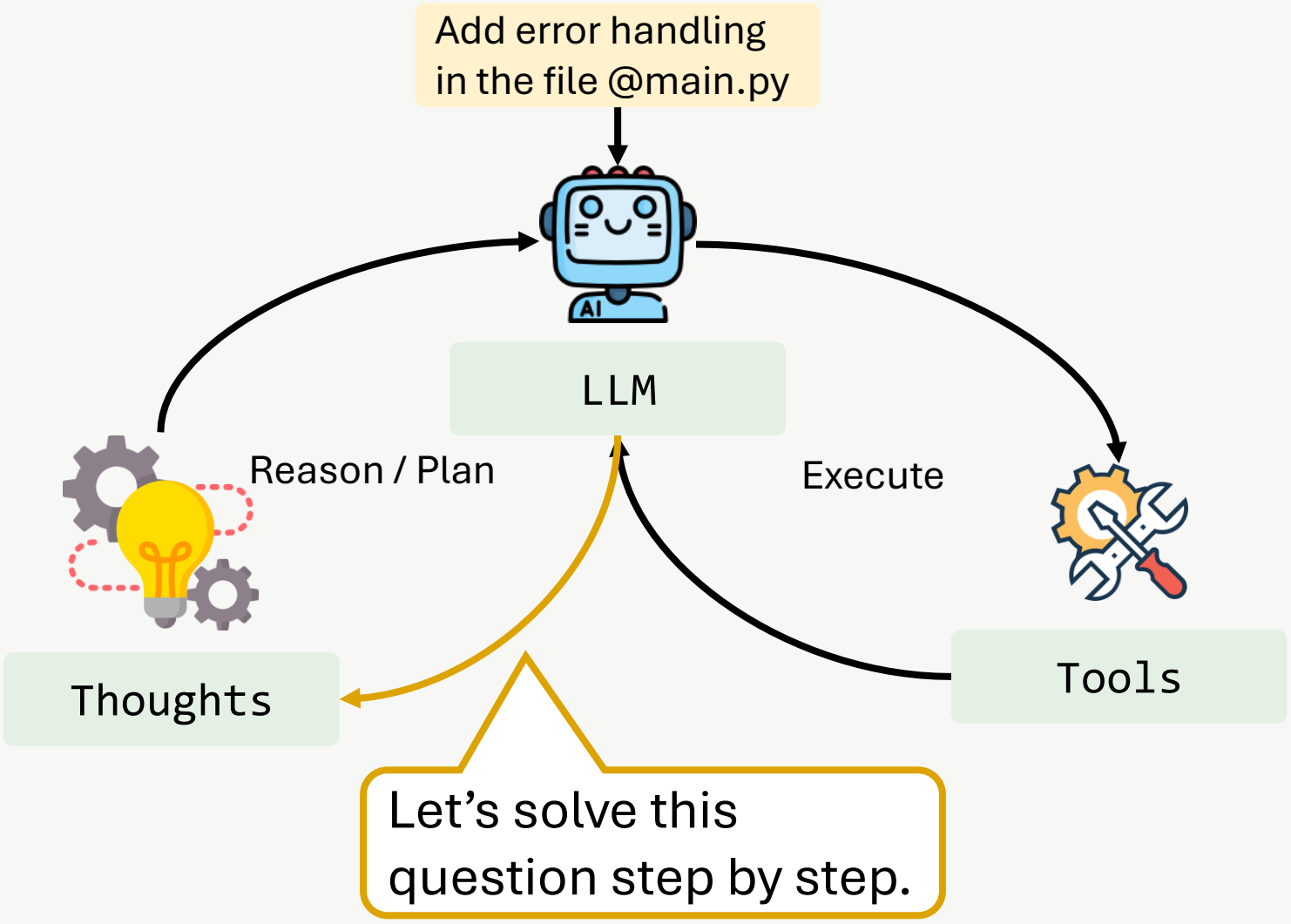
The Core of Loop



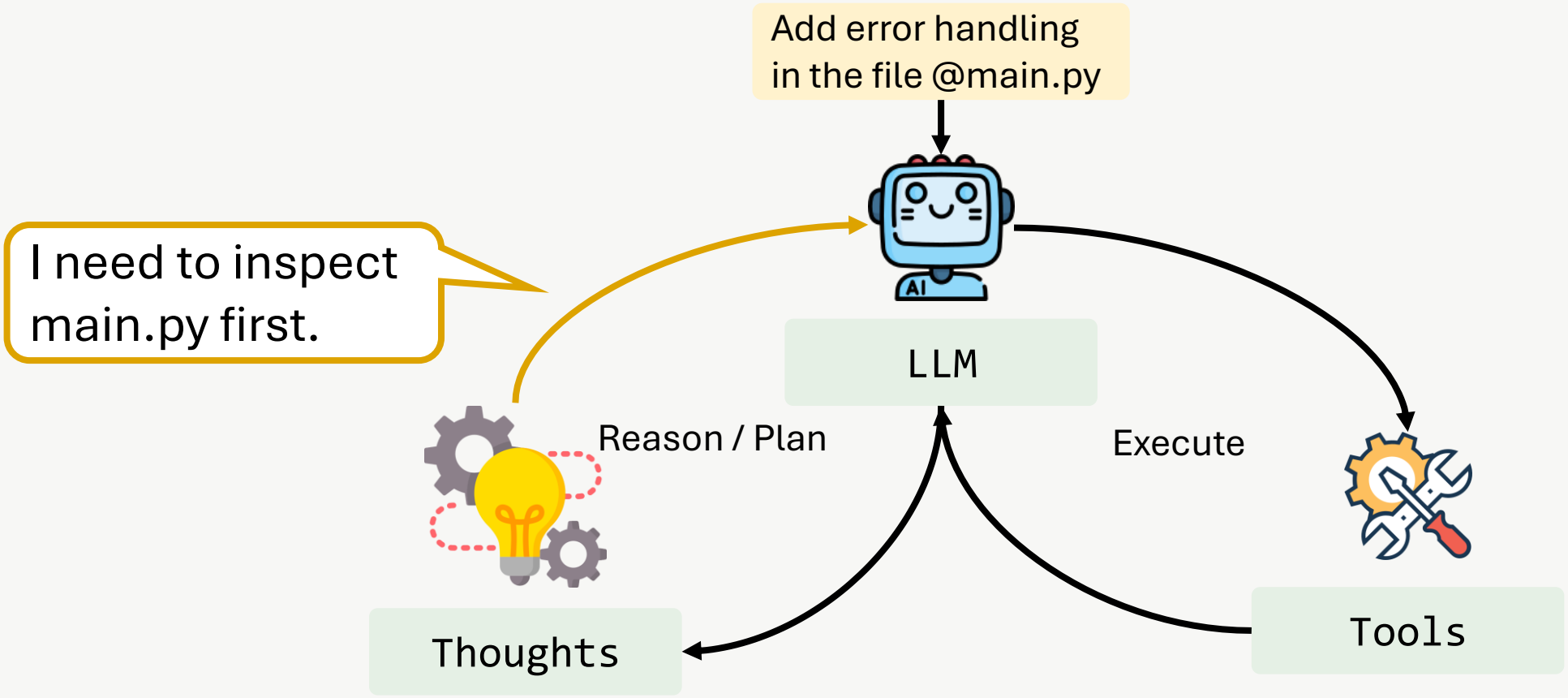
The Core of Loop



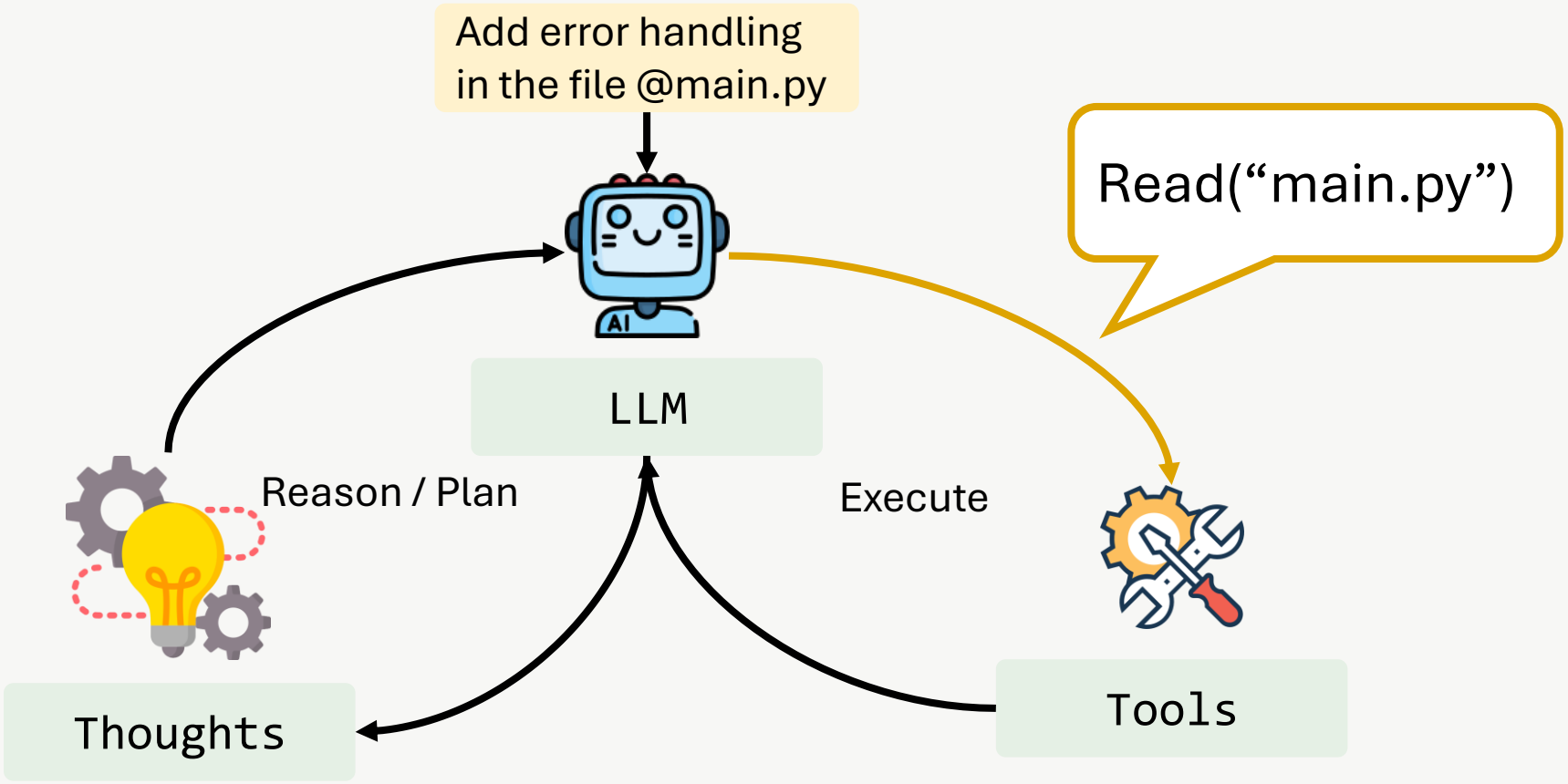
The Core of Loop



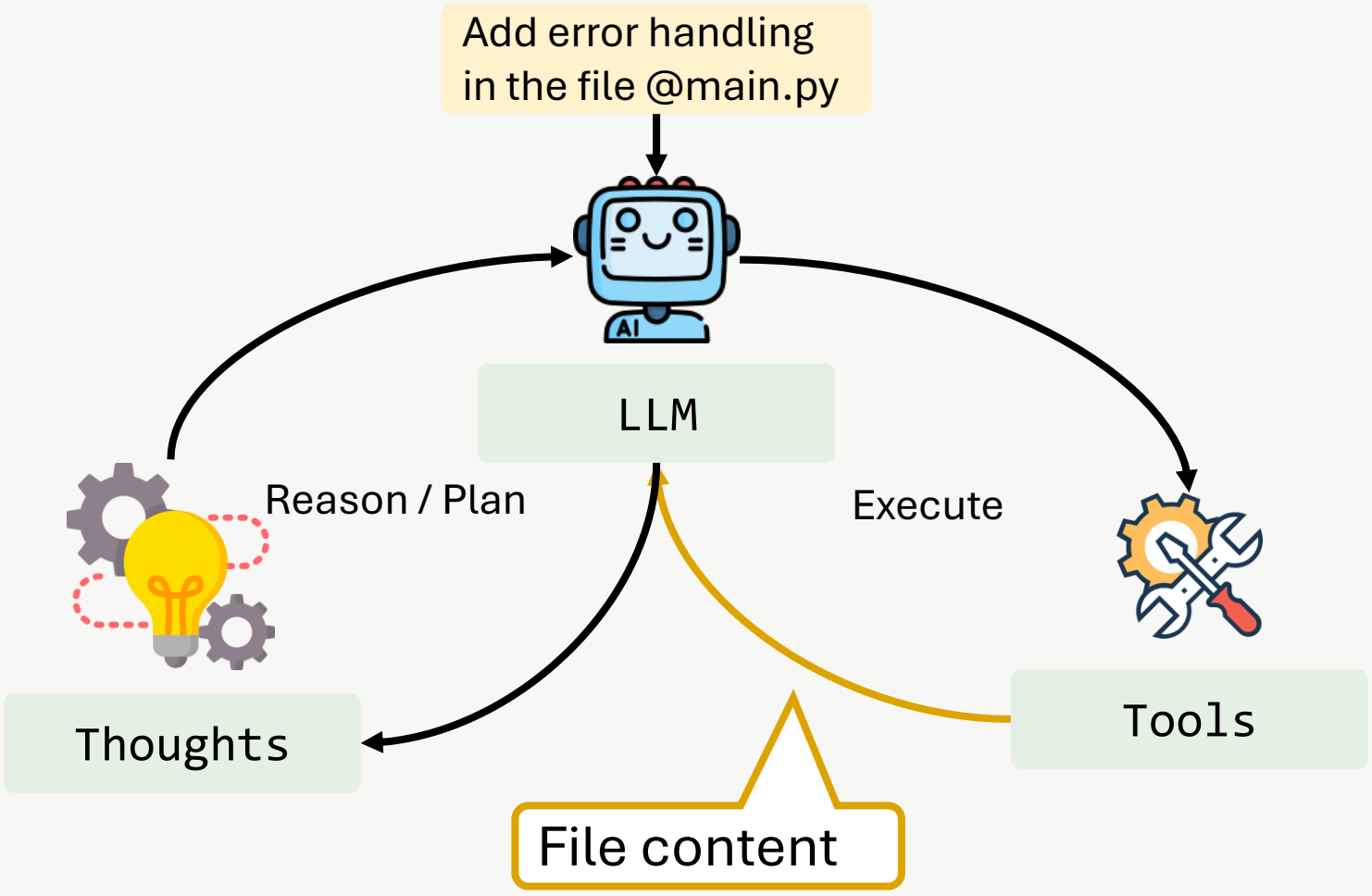
The Core of Loop



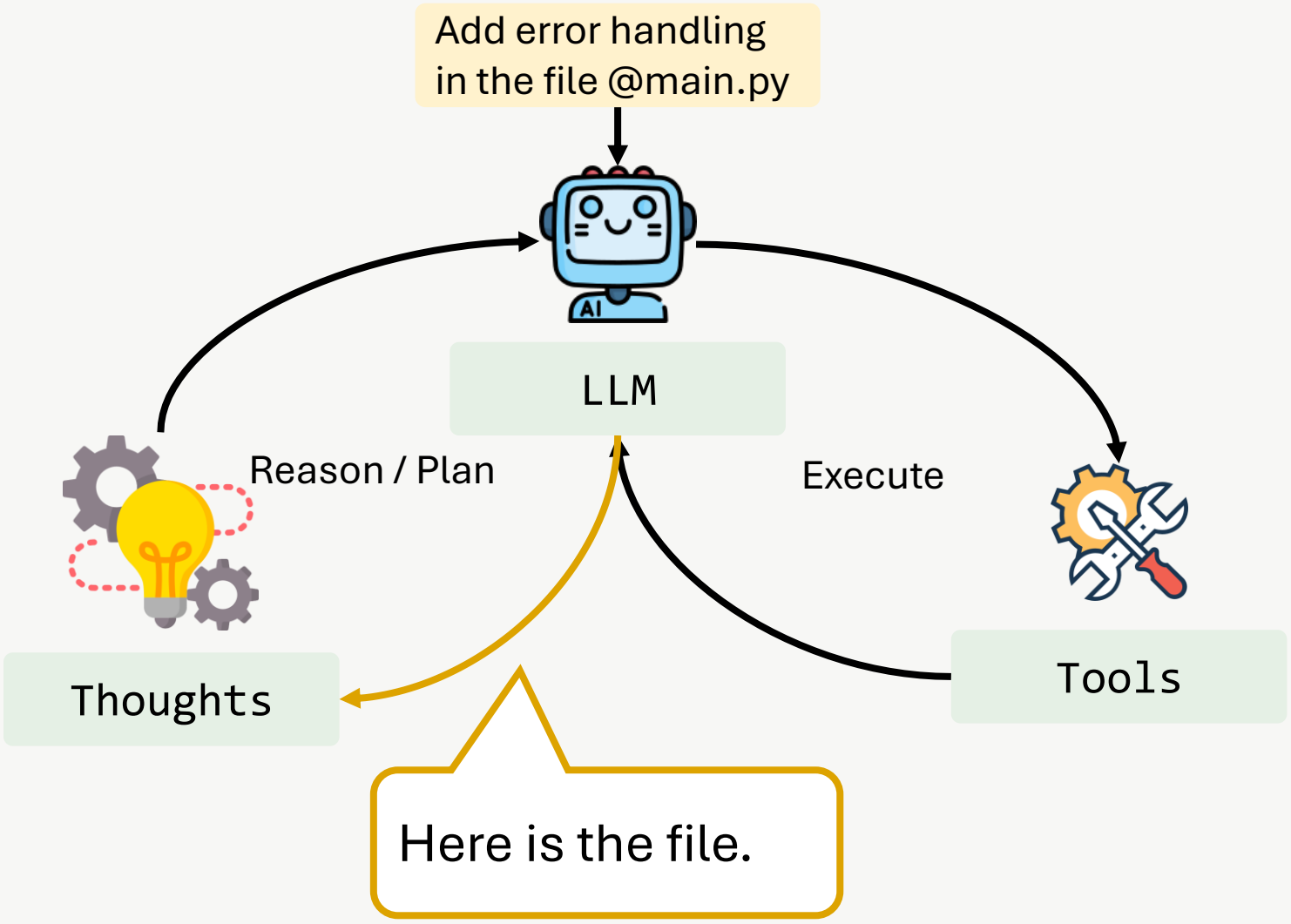
The Core of Loop



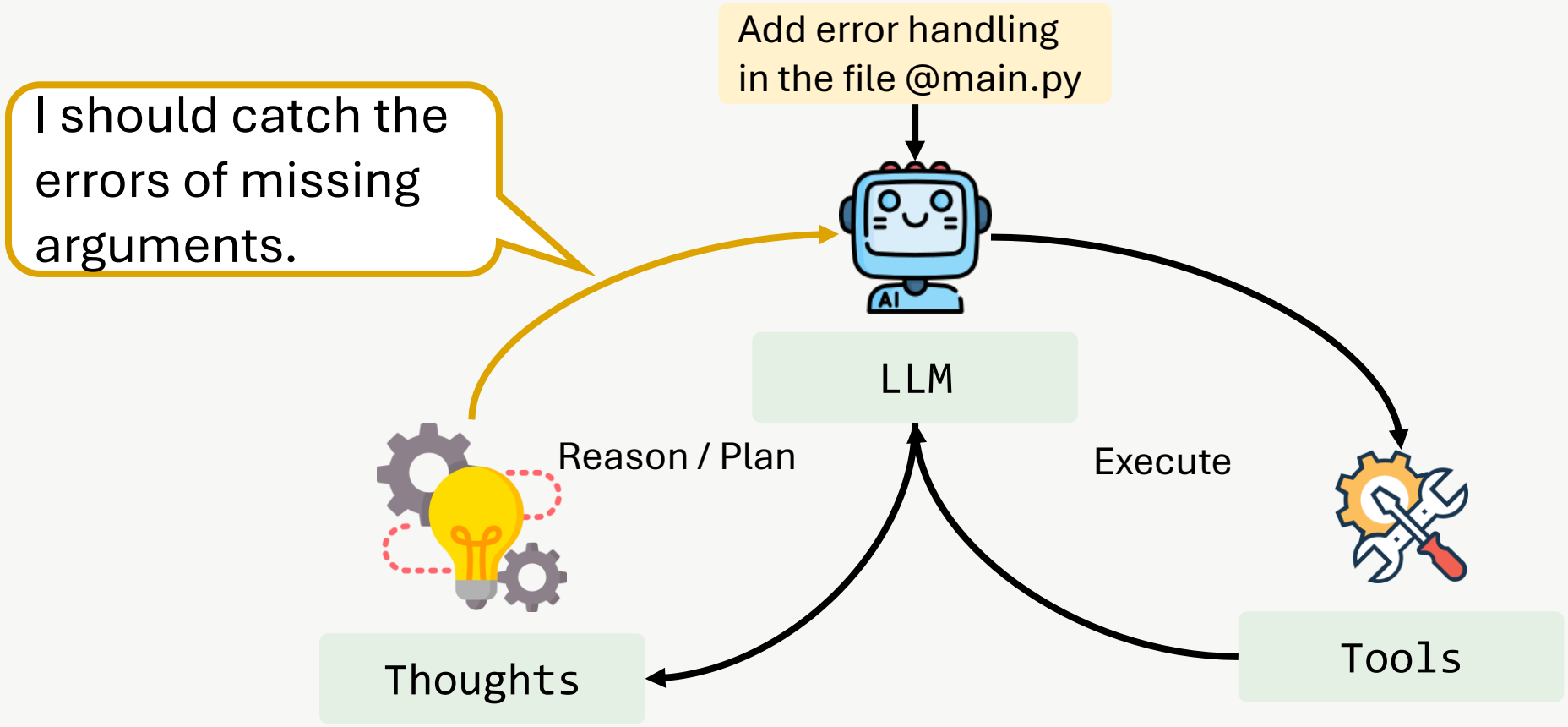
The Core of Loop



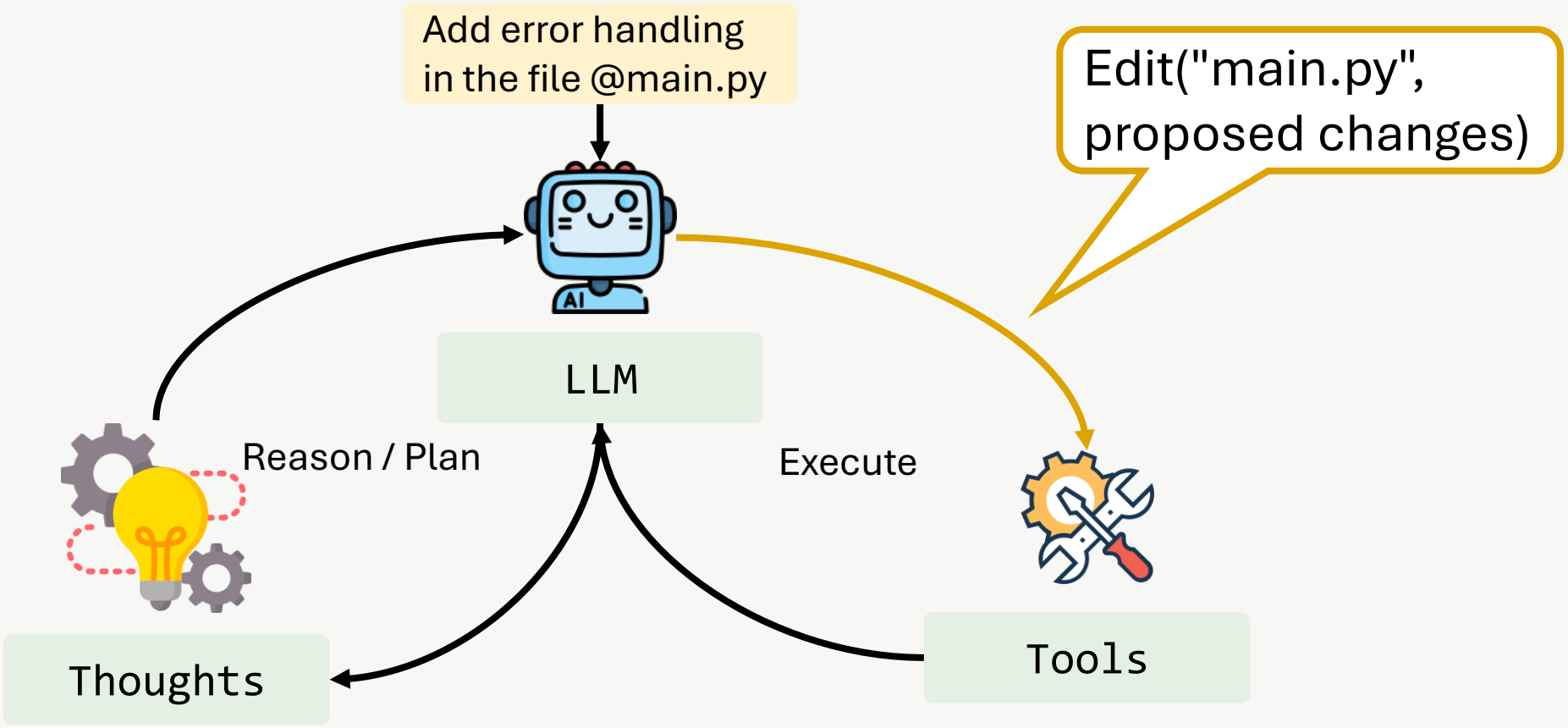
The Core of Loop



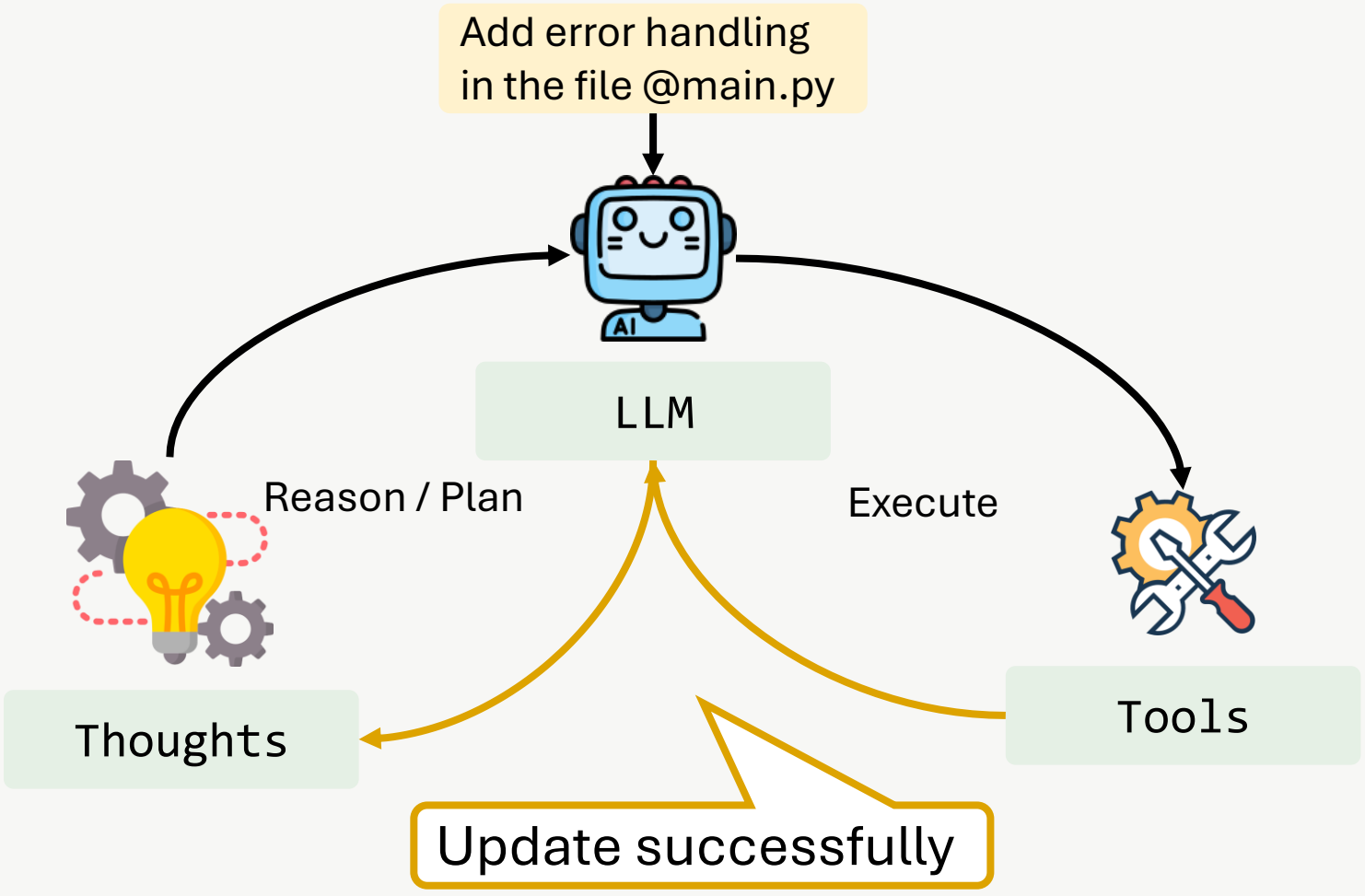
The Core of Loop



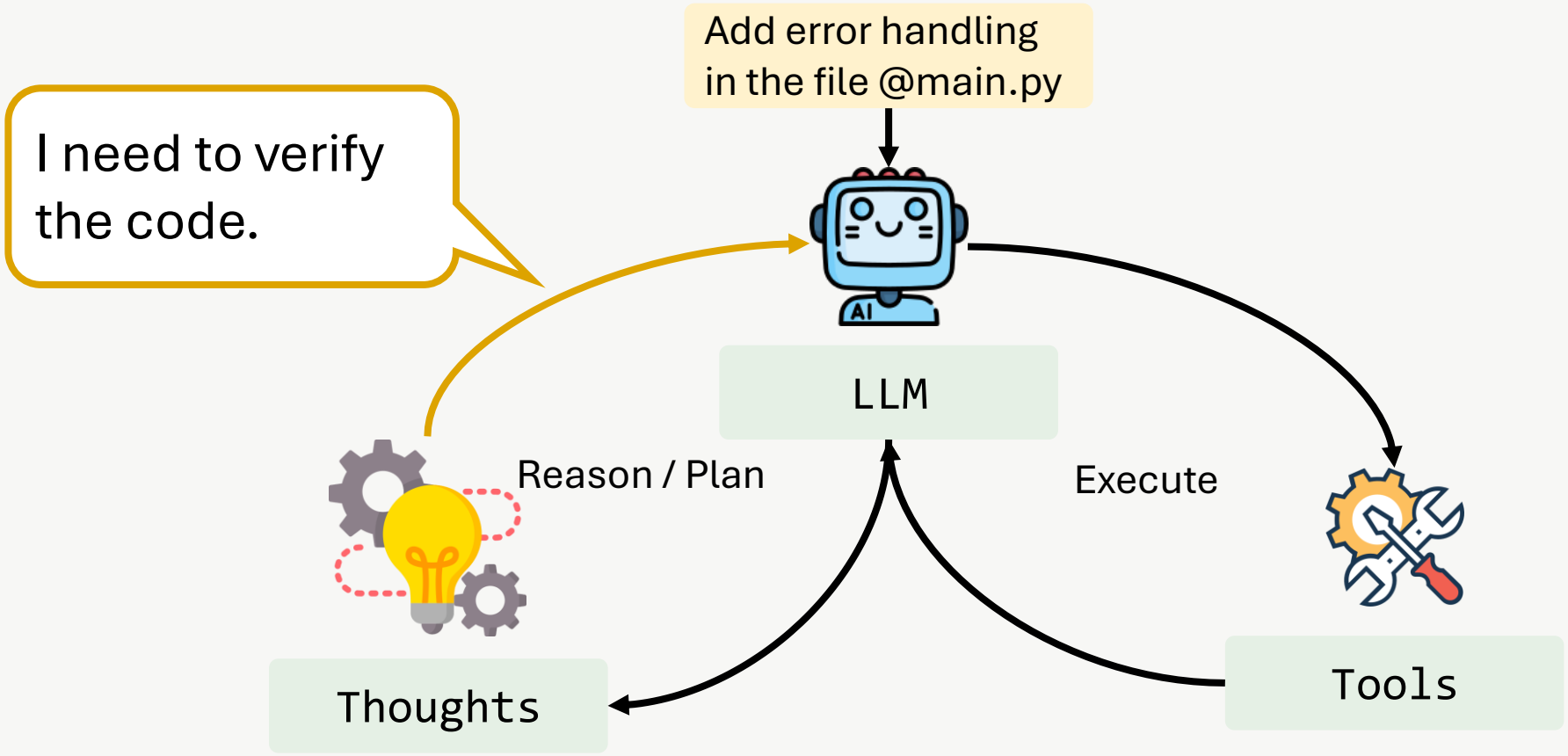
The Core of Loop



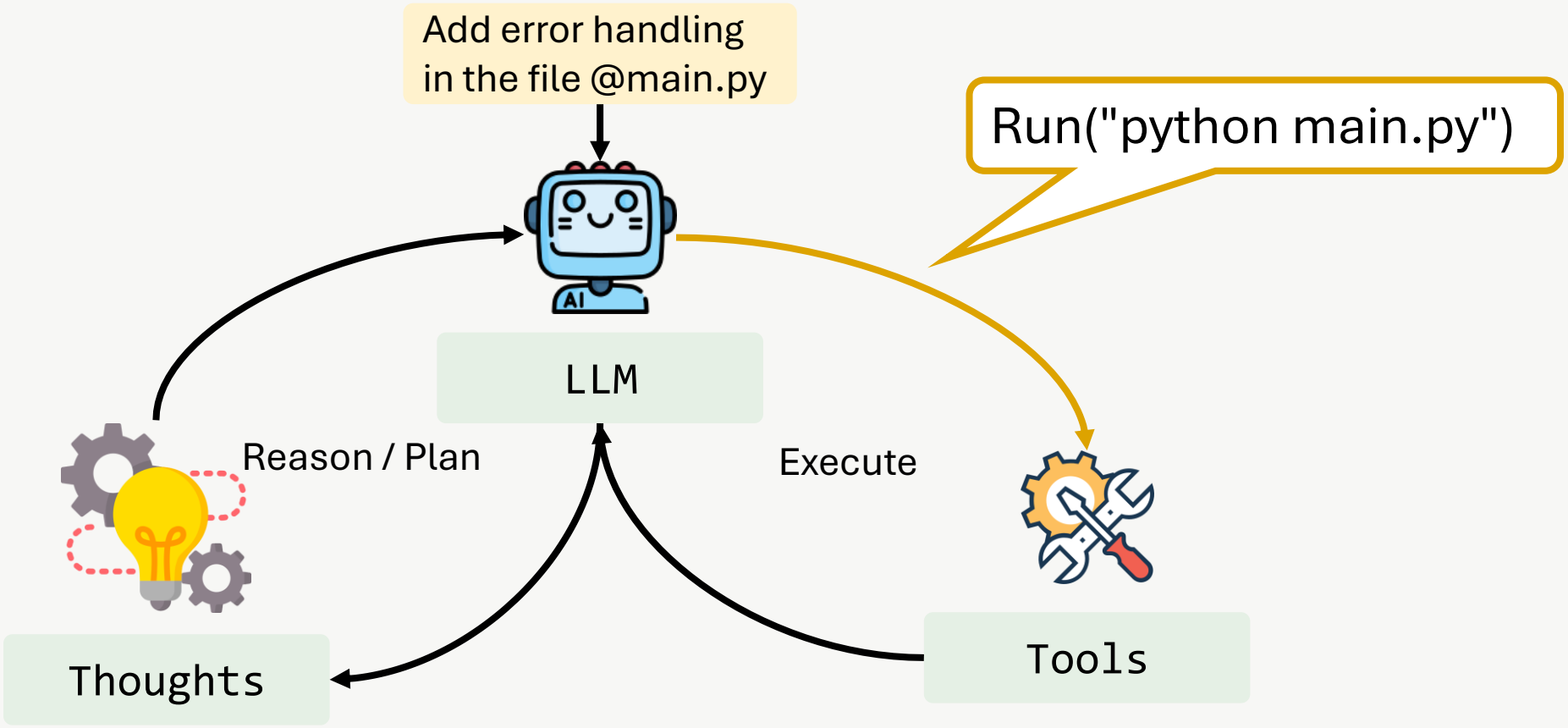
The Core of Loop



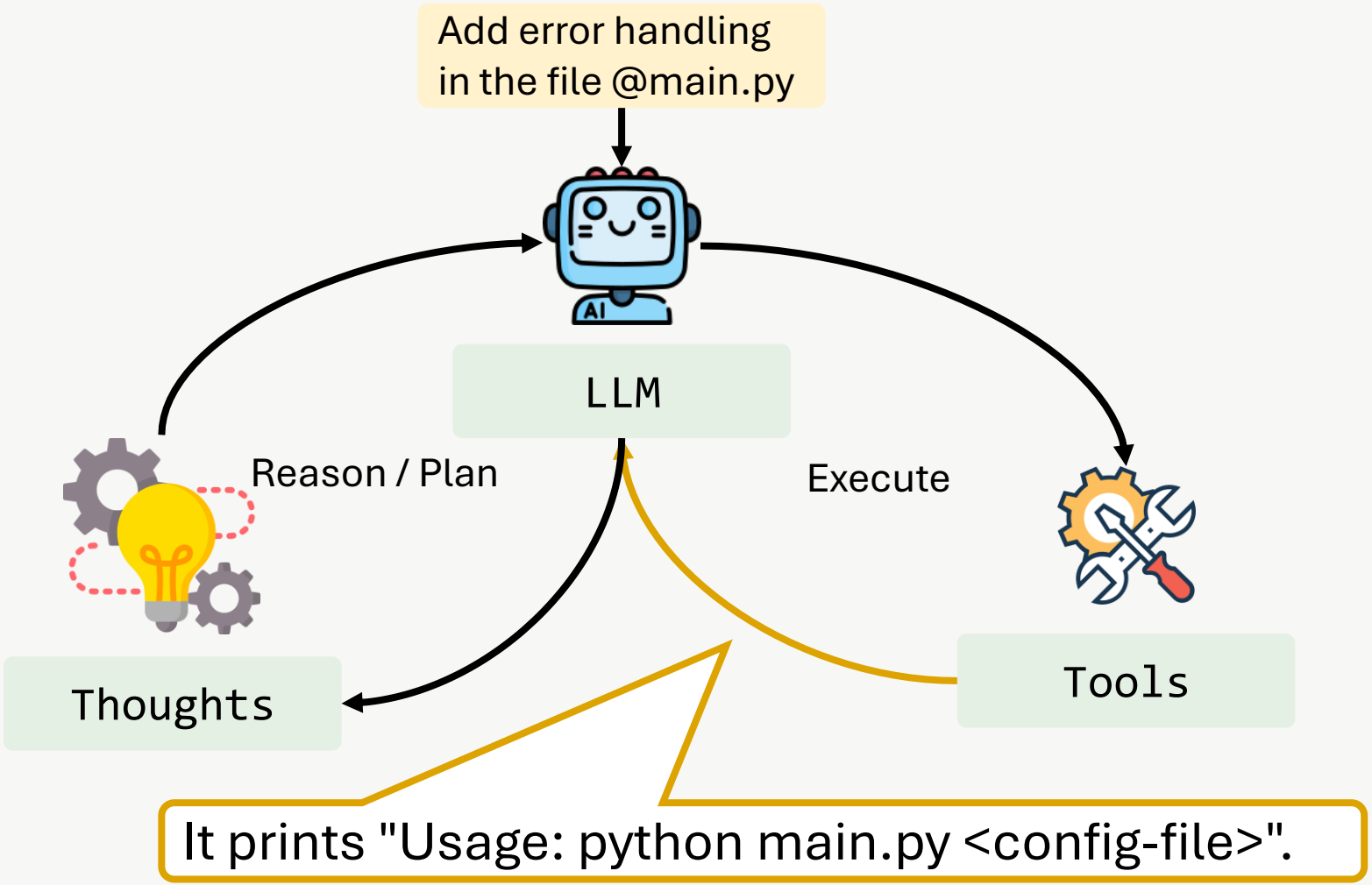
The Core of Loop



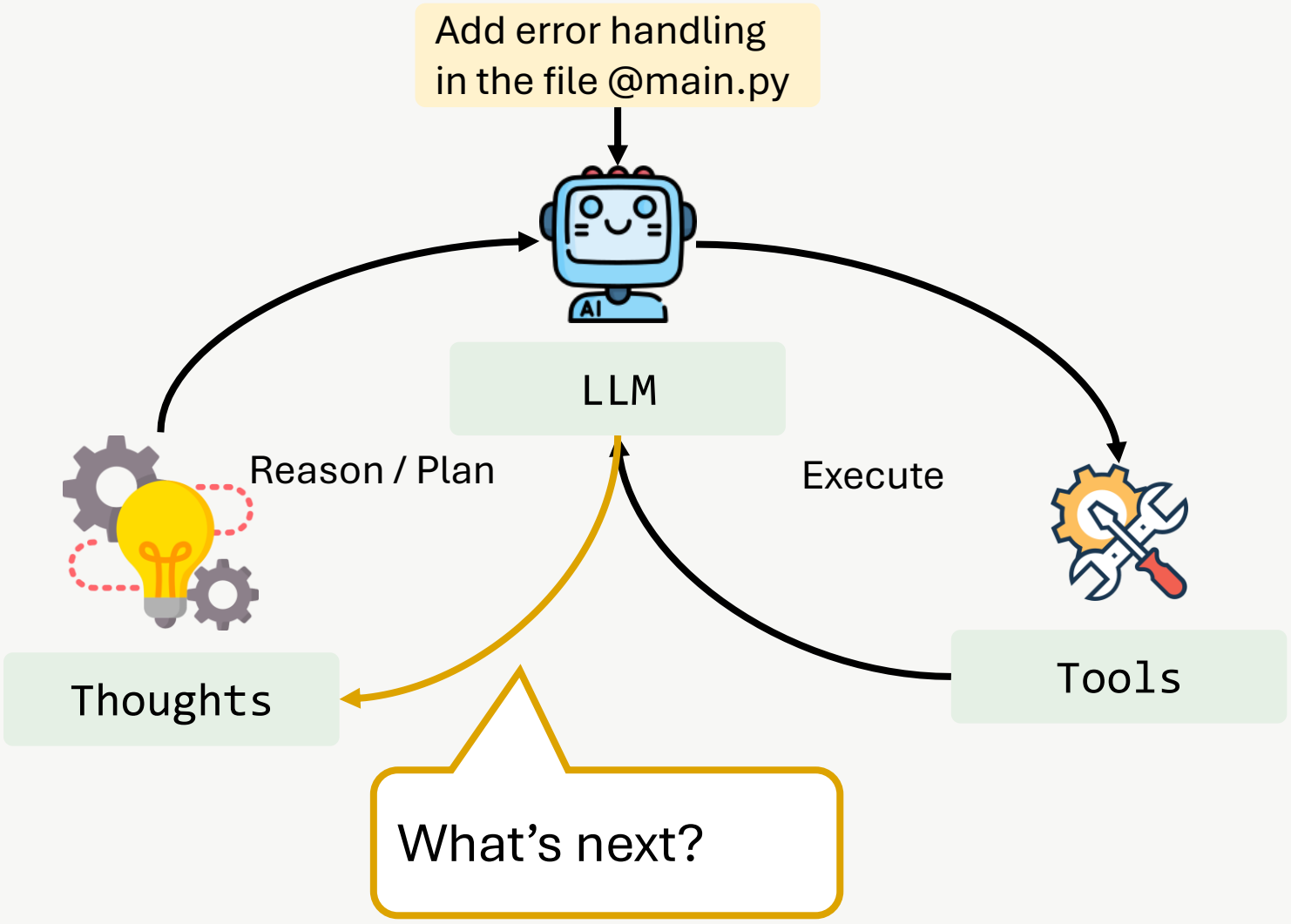
The Core of Loop



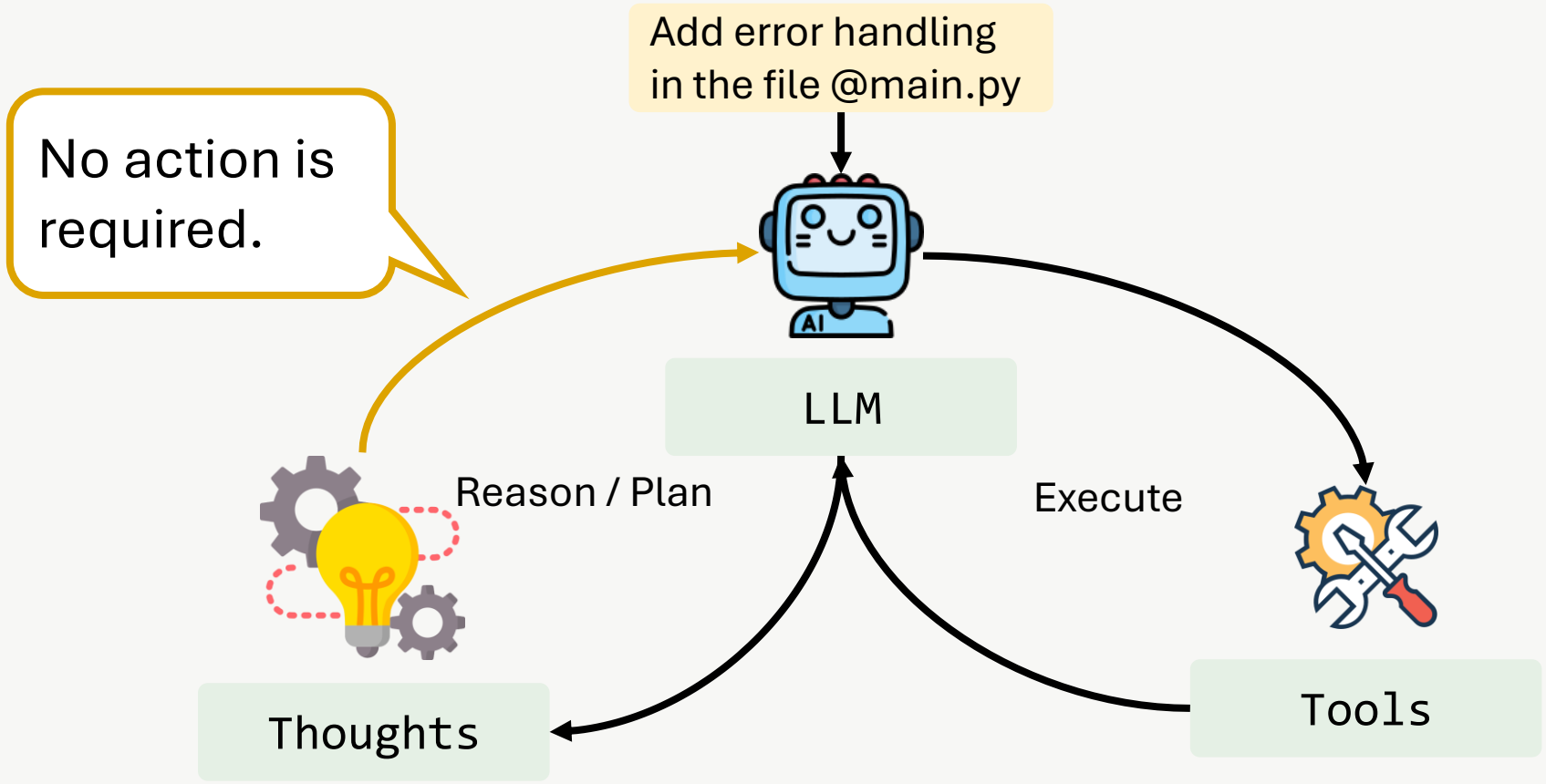
The Core of Loop



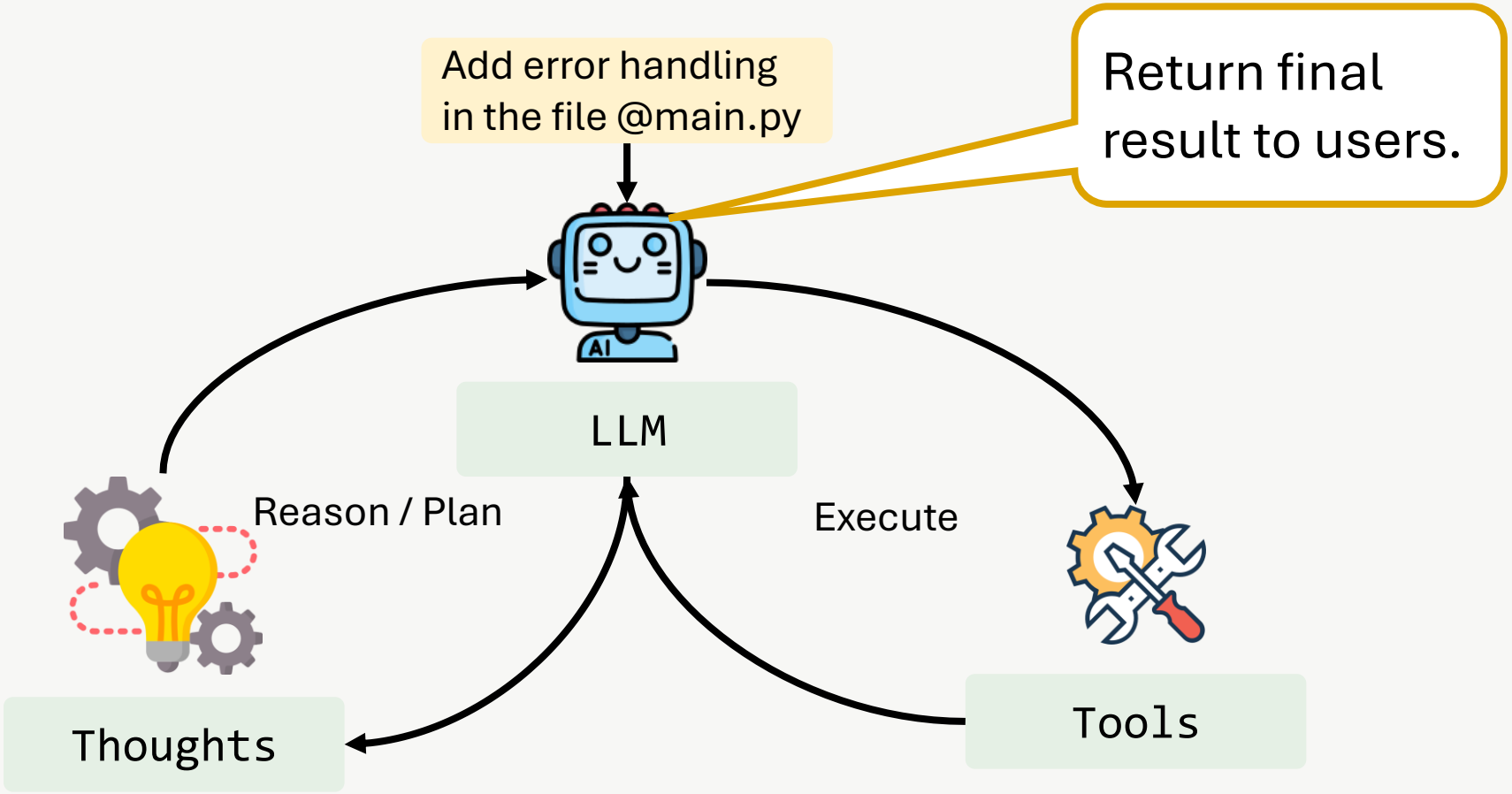
The Core of Loop



The Core of Loop



The Core of Loop



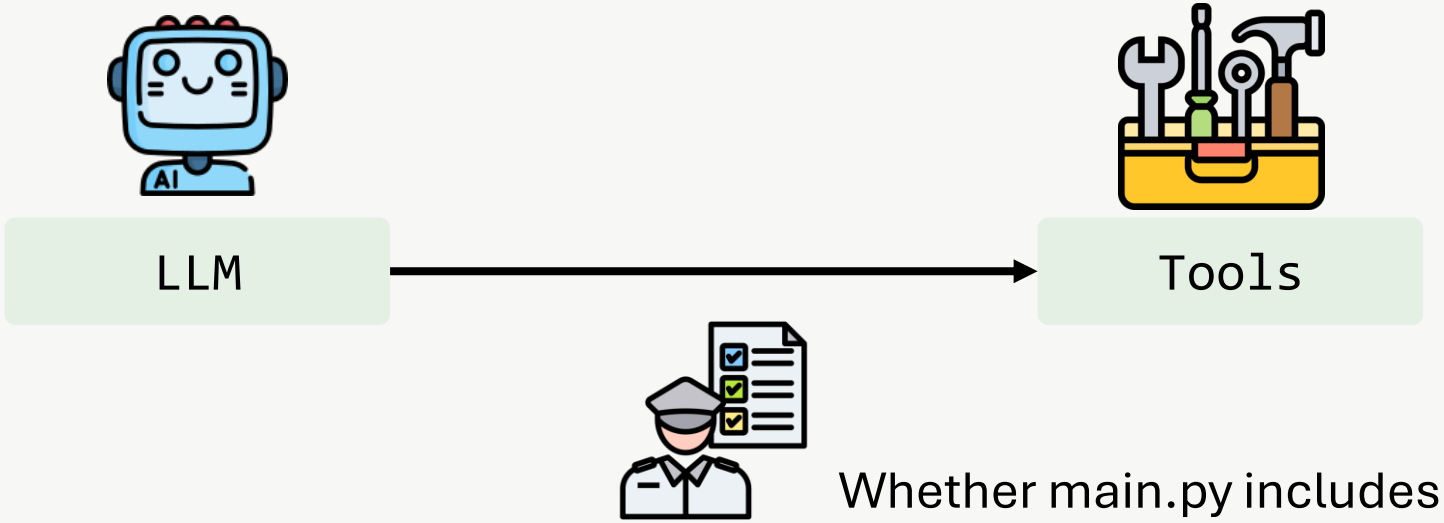
6 Call Tools

Tool	Description	Permission Required
Bash	Executes shell commands in your environment. See Bash tool behavior	Yes
Edit	Makes targeted edits to specific files. See Edit tool behavior	Yes
Glob	Finds files based on pattern matching. See Glob tool behavior	No
Grep	Searches for patterns in file contents. See Grep tool behavior	No
Read	Reads the contents of files. See Read tool behavior	No
Write	Creates or overwrites files. See Write tool behavior	Yes

Some common tools. For full list of tools:
https://code.claude.com/docs/en/tools-reference

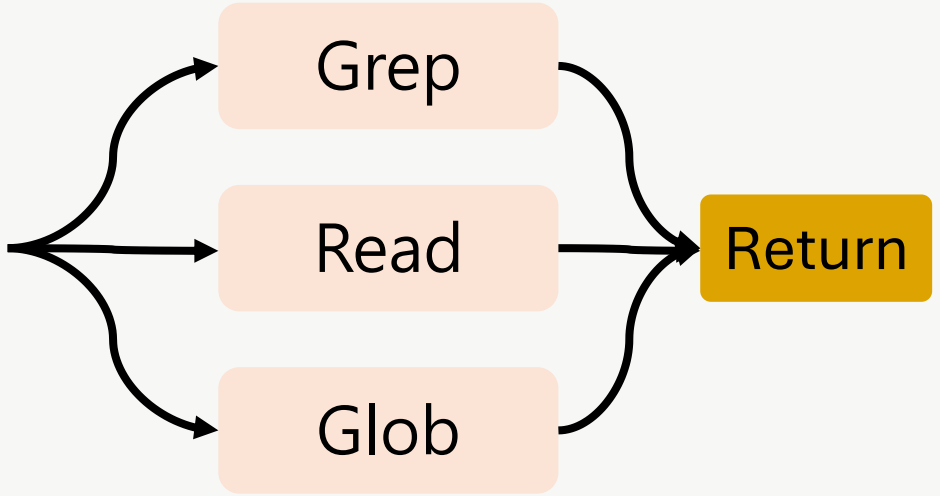
6 Call Tools

Tools are the interface for environment, so it is critical to make sure each tool call is safe and valid.

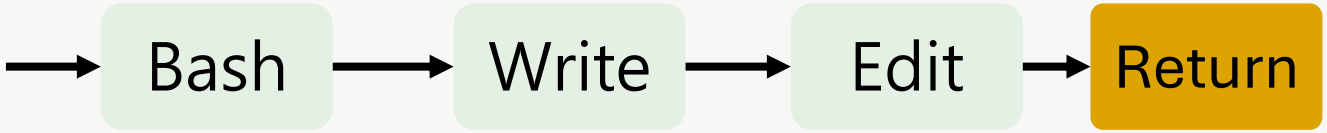


7 Receive tool results

- Read tools are executed in parallel



- Write tools are executed in sequence

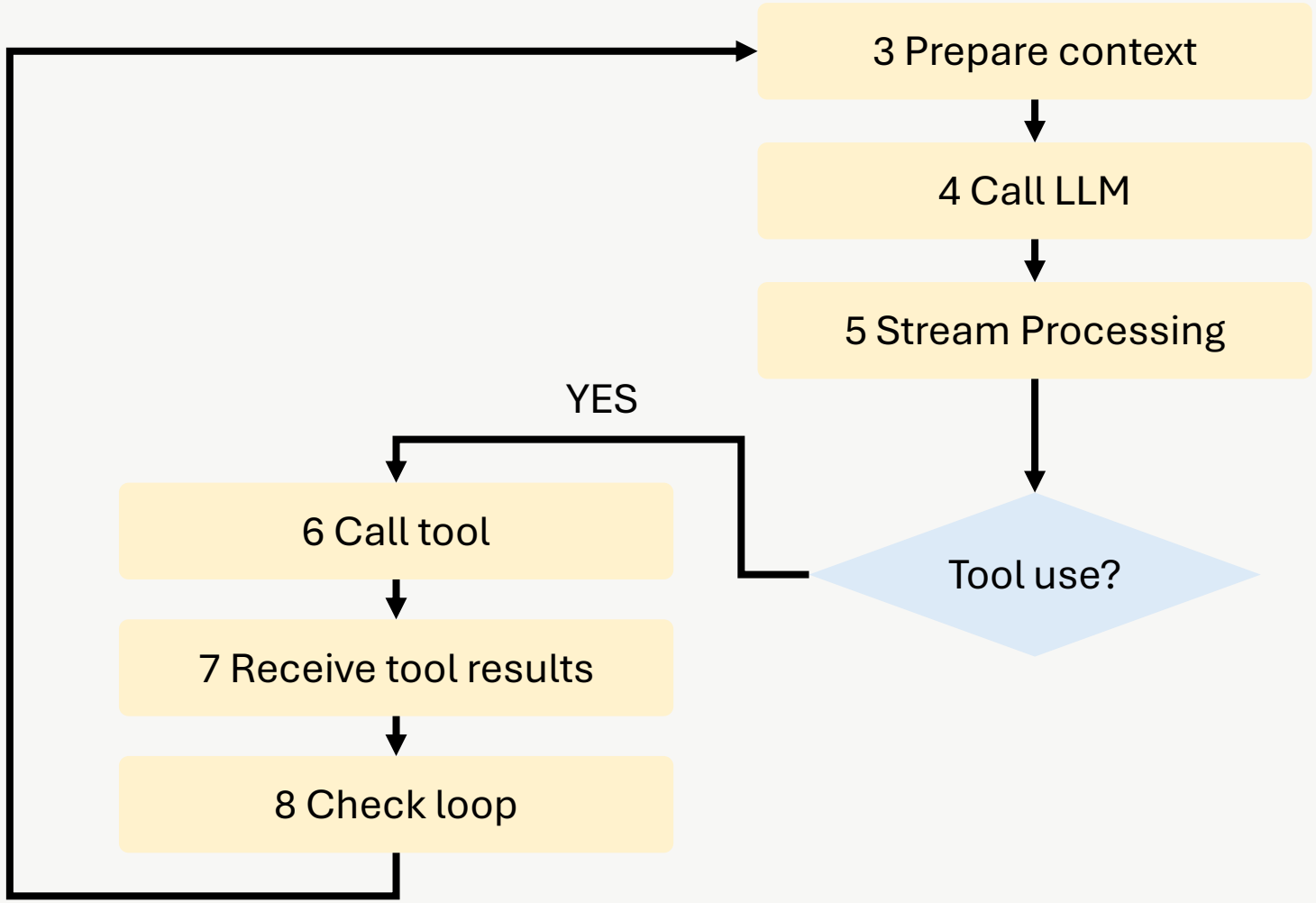


7 Receive tool results

Append the tool (Read) result into message list

```
{
    ...
    "message": {
        "role": "user",
        "content": [
            {
                "tool_use_id": "call_bZnZbCZ1smrb7oIXEFUqQy8u",
                "type": "tool_result",
                "content": "import request..."
            }
        ]
    }
}
```

8 Check loop


3 Prepare context
4 Call LLM
5 Stream Processing
Tool use?
YES
6 Call tool
7 Receive tool results
8 Check loop

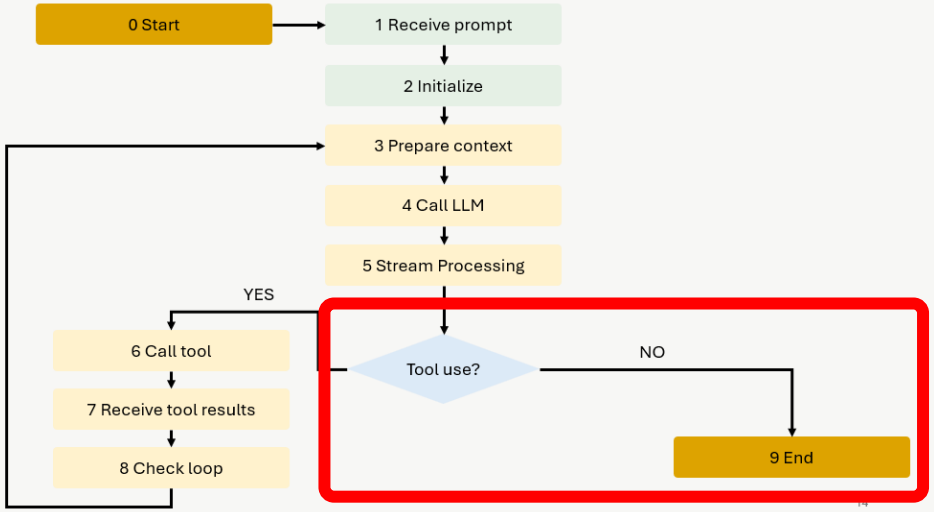
8 Check Loop

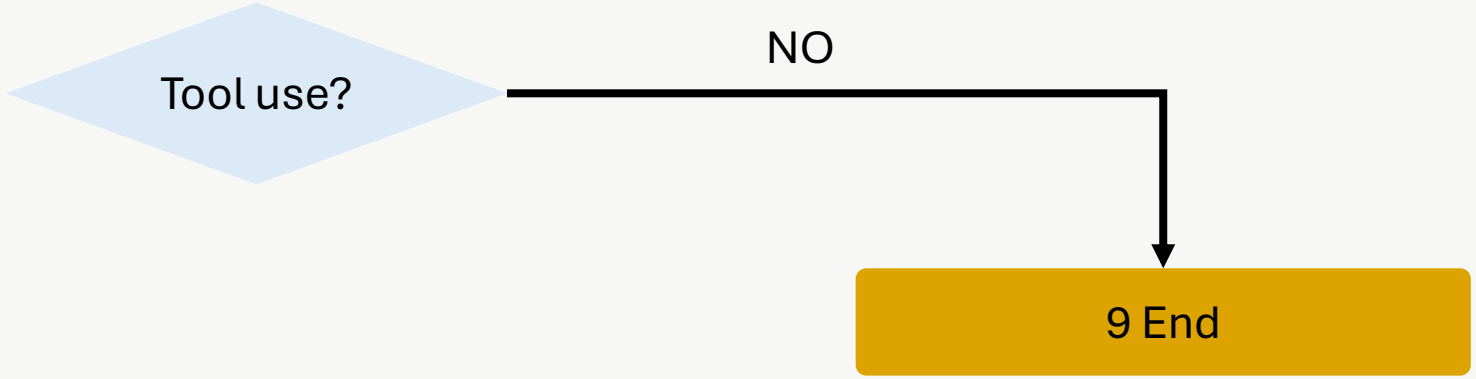
Passing all messages to LLMs to check more tool calls are required.

8 Check Loop

- These errors also stop the loop
  - Reach the number threshold of loops to avoid dead loop
  - Messages exceed the token limit
  - Users request to stop the loop
  - LLM APIs return errors

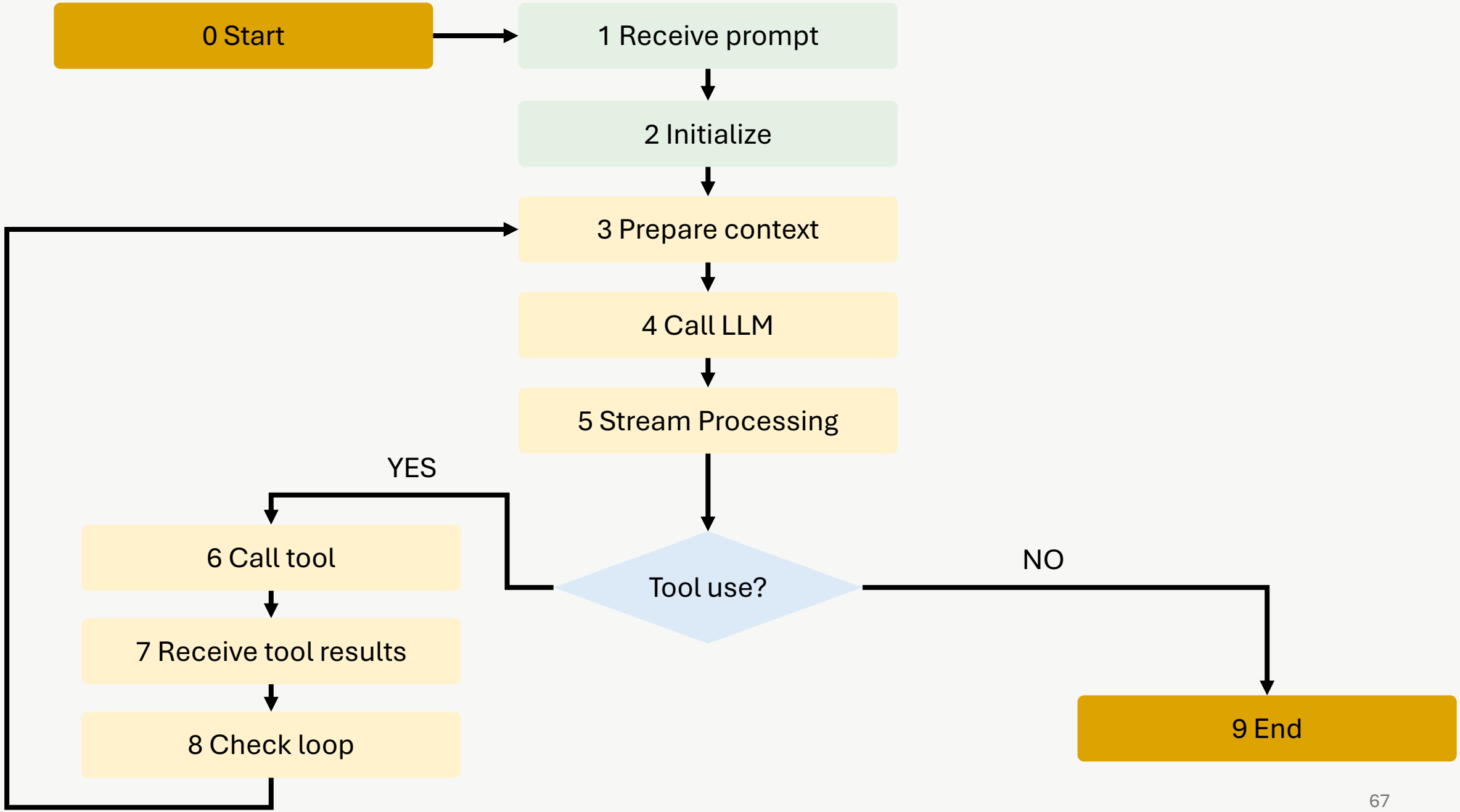
Part 3: Output

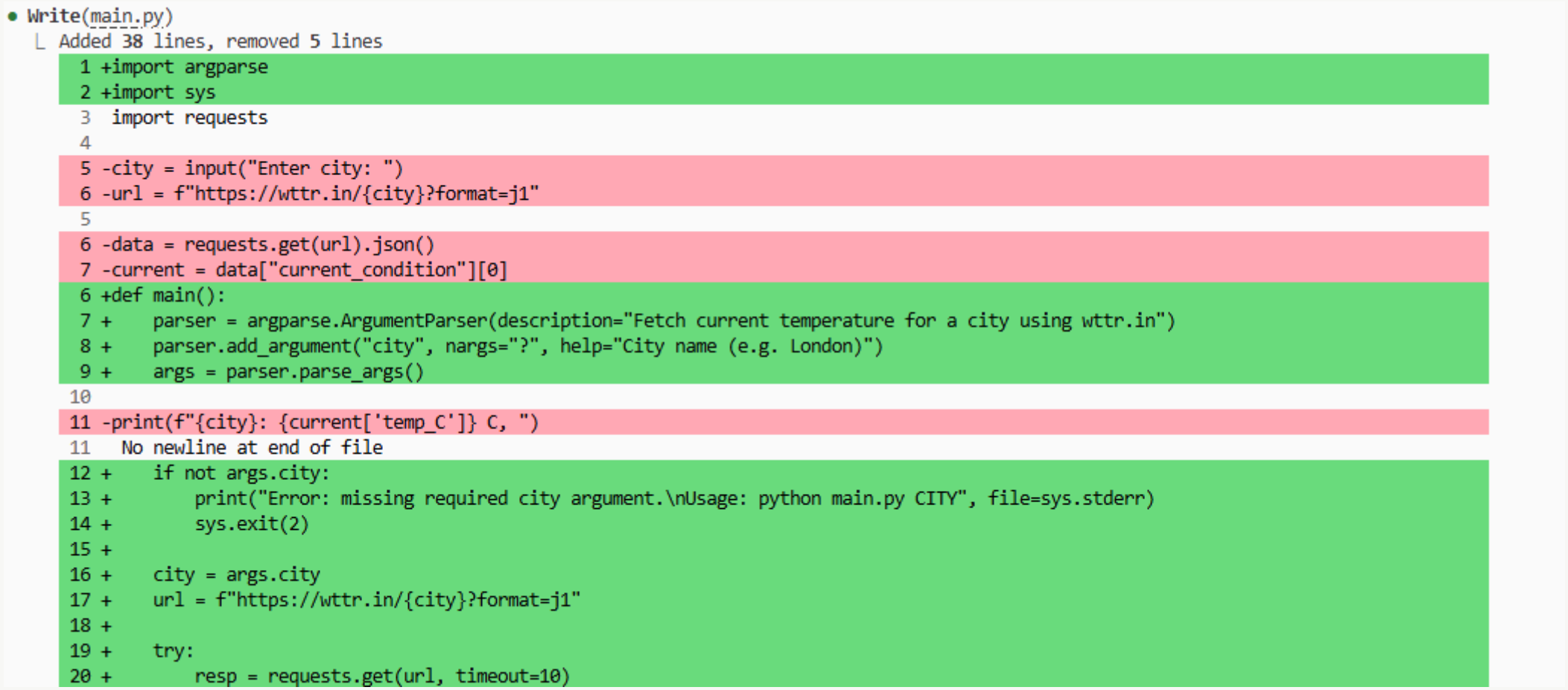




9 End

- Update state, such as token usage.
- Send summary to users.
- All historical messages stored into ***~/.claude/projects/*** for backup.


0 Start
1 Receive prompt
2 Initialize
3 Prepare context
4 Call LLM
5 Stream Processing
Tool use?
YES
NO
6 Call tool
7 Receive tool results
8 Check loop
9 End



After several iterations of loops, agent completes our request!

What one is the best term?

Prompt Engineering → Context Engineering → Harness Engineering → Loop Engineering

Discussion

- Do we always need agents?

Agentless

AGENTLESS 🐱: Demystifying LLM-based Software Engineering Agents

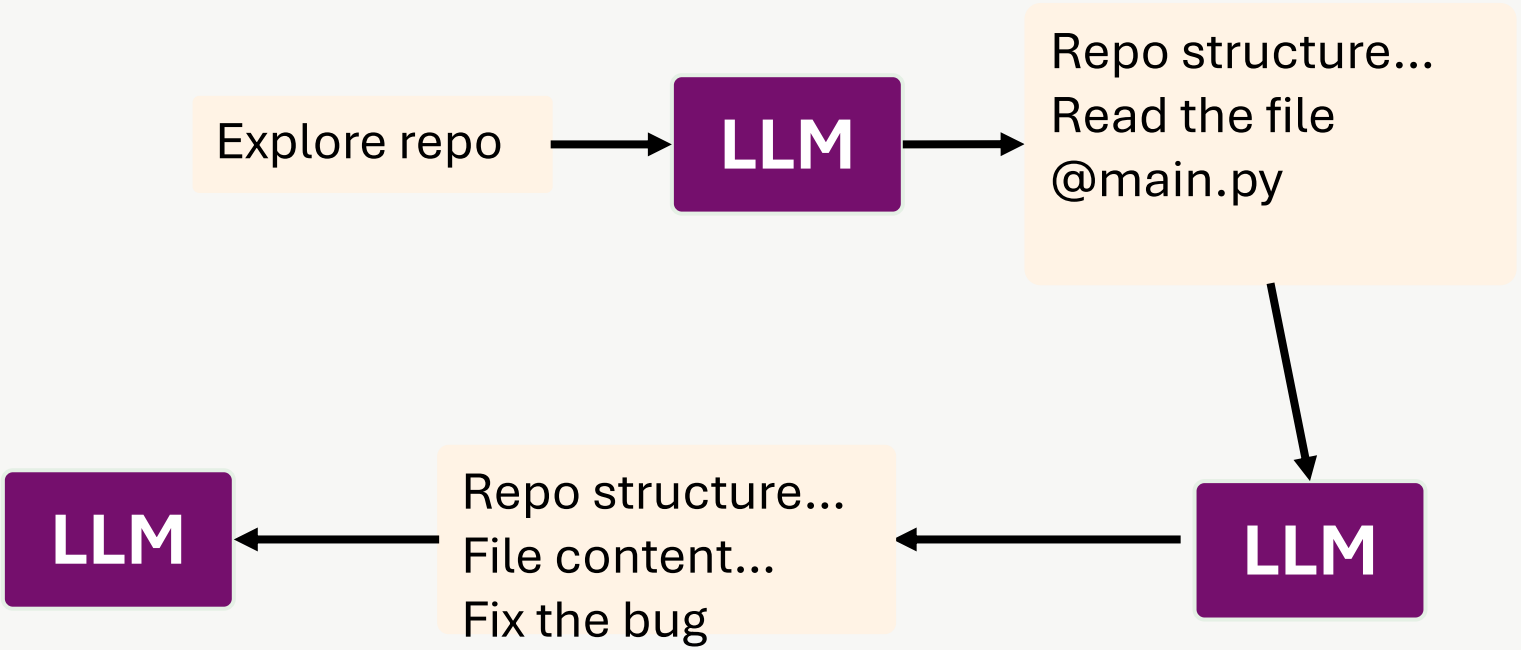
Chunqiu Steven Xia* **Yinlin Deng*** **Soren Dunn** **Lingming Zhang**

University of Illinois Urbana-Champaign

{chunqiu2, yinlind2, sorend2, lingming}@illinois.edu

https://arxiv.org/pdf/2407.01489

Agentless


Explore repo
LLM
Repo structure...
Read the file
@main.py
LLM
Repo structure...
File content...
Fix the bug
LLM

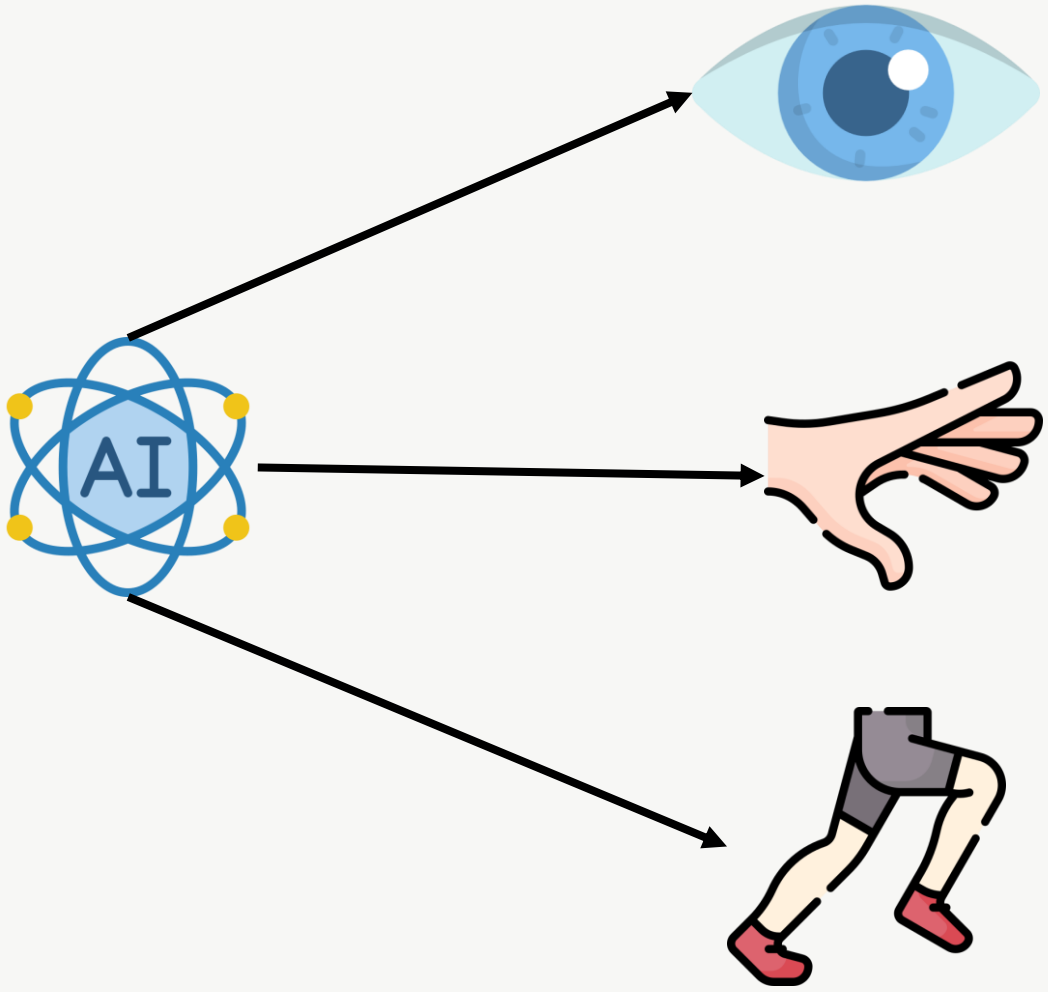
Discussion

- Do we always need agents?
- Sometimes, reasoning wastes resource for simple tasks
- Sometimes, LLMs are not necessary
  - Code can do it!
  - Lower latency
  - Lower token consumption

Case Study

- Weather query
- Linux commands

Next Lecture



Let’s make agents stronger!

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Questions

Next class: Advanced Agents