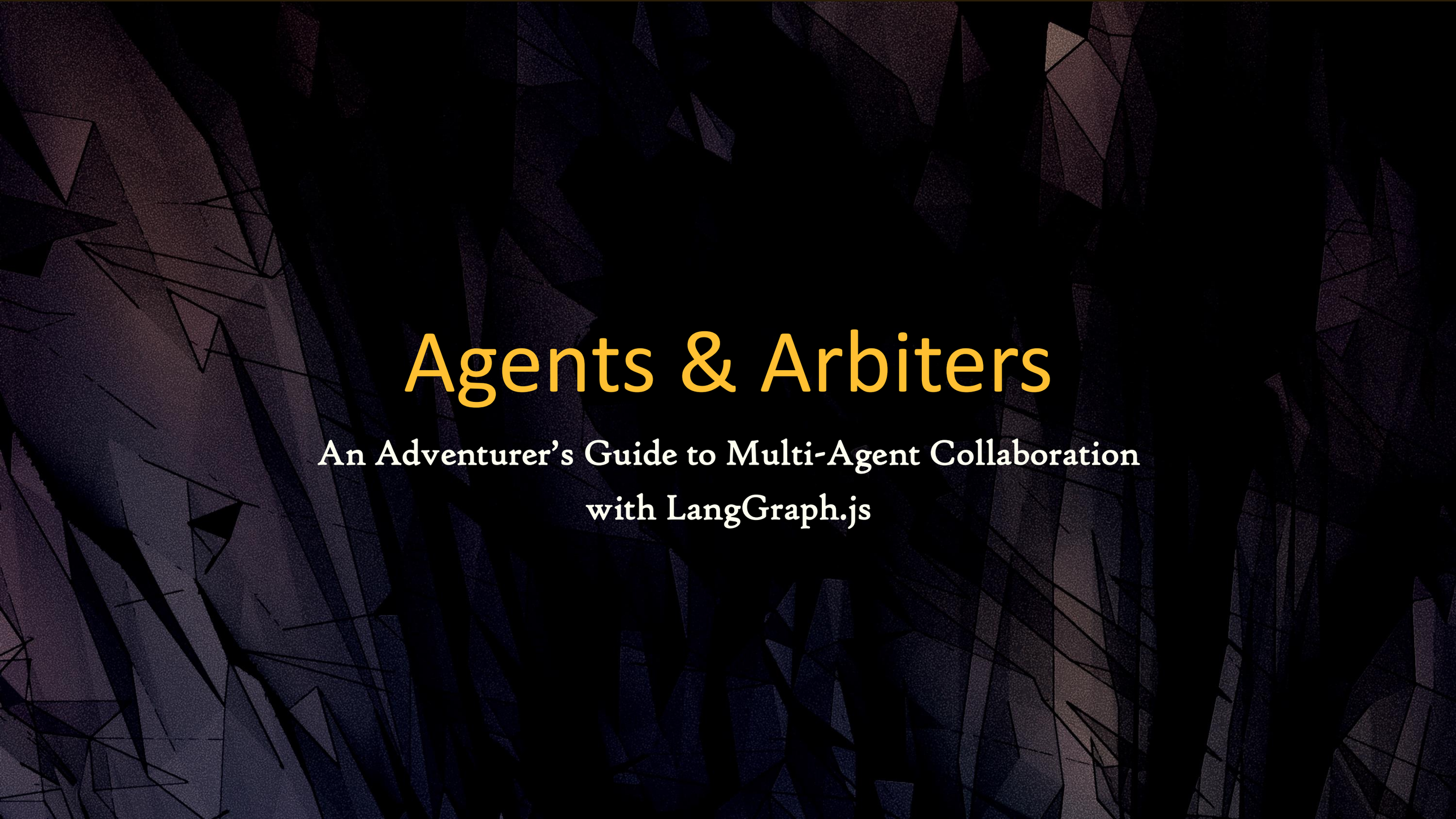

Agents & Arbiters
An Adventurer's Guide to Multi-Agent Collaboration
with LangGraph.js

Guy Royse

Developer Advocate

Redis

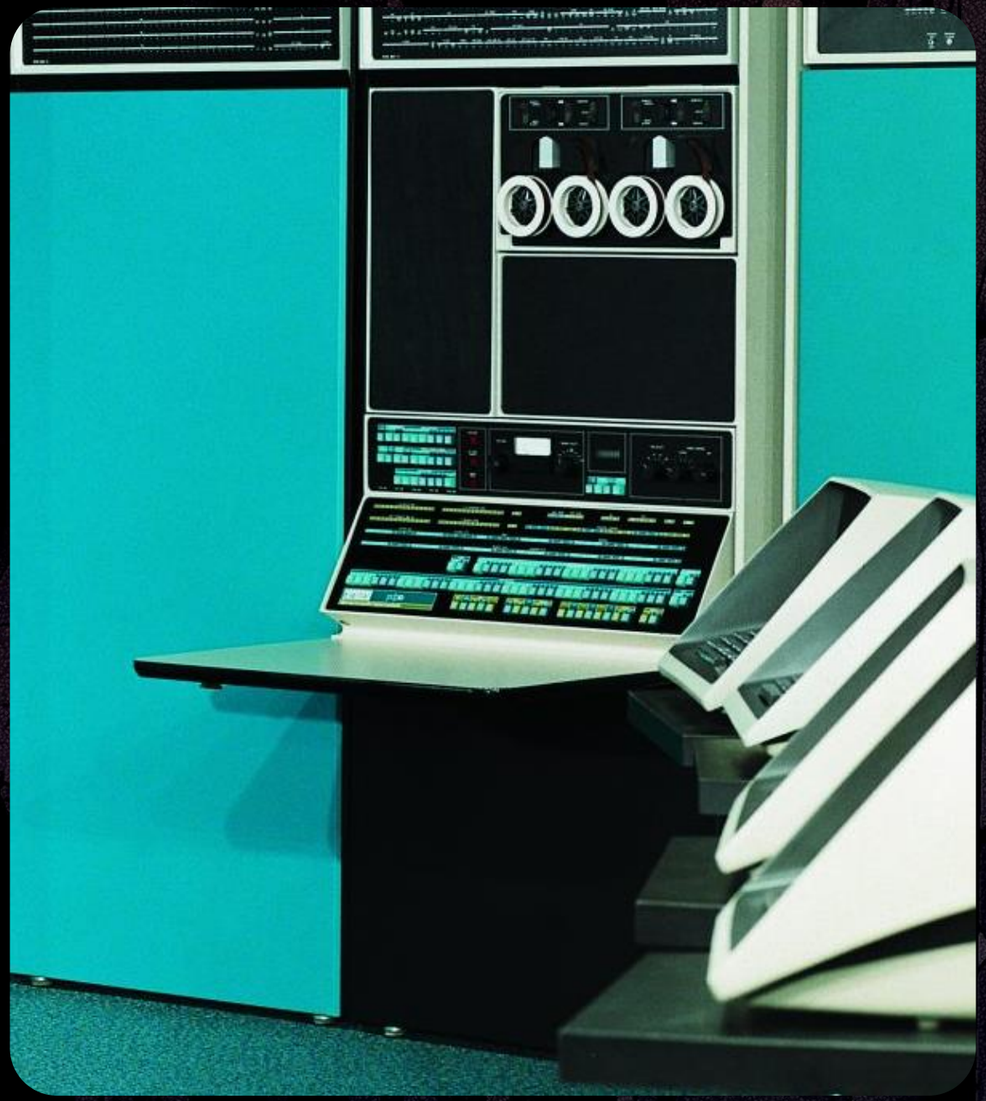
@guy.dev

@guyroyse

github.com/guyroyse

guy.dev


ZIDEK




CTRL
RUN STOP
SHIFT LOCK
SHIFT

West of House Score: 2 Moves: 5

>Look
West of House
You are standing in an open field west of a white house, with a boarded front door.
There is a small mailbox here.

>Look in mailbox
The small mailbox is closed.

>Open mailbox
Opening the small mailbox reveals a leaflet.

>Take leaflet
Taken.

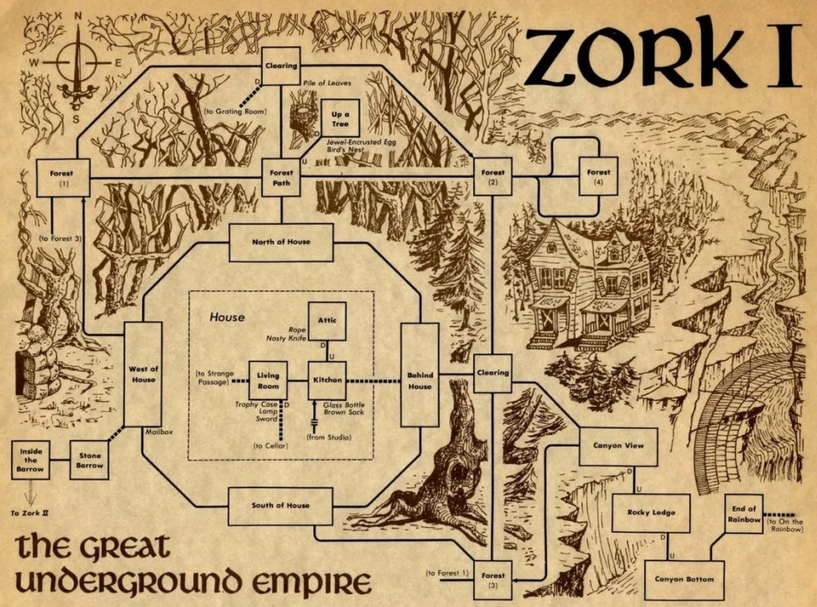
>Look at leaflet
"WELCOME TO ZORK!

ZORK is a game of adventure, danger, and low cunning. In it you will explore some of the most amazing territory ever seen by mortals. No computer should be without one!"

>_

```
"ZORK1 for
                Zork I: The Great Underground Empire
        (c) Copyright 1983 Infocom, Inc.  All Rights Reserved."

<VERSION ZIP>

<SETG ZORK-NUMBER 1>

<SET REDEFINE T>

<OR <GASSIGNED? ZILCH>
    <SETG WBREAKS <STRING !\" !,WBREAKS>>>

<PRINC "Renovated ZORK I: The Great Underground Empire
">

<FREQUENT-WORDS?>

<INSERT-FILE "GMACROS" T>
<INSERT-FILE "GSYNTAX" T>
<INSERT-FILE "1DUNGEON" T>
<INSERT-FILE "GGLOBALS" T>

<PROPDEF SIZE 5>
...
```

zork1.zil

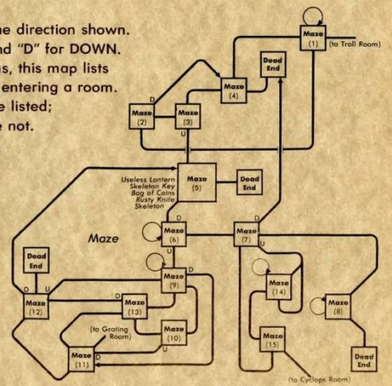


Map designed by D. Ardito and S. Meretzky

Legend

- Normal passageway
- One-way passageway
- Passageway requiring special equipment or problem-solving
- Narrow passageway (baggage limit)
- Passageway returning to room of origin

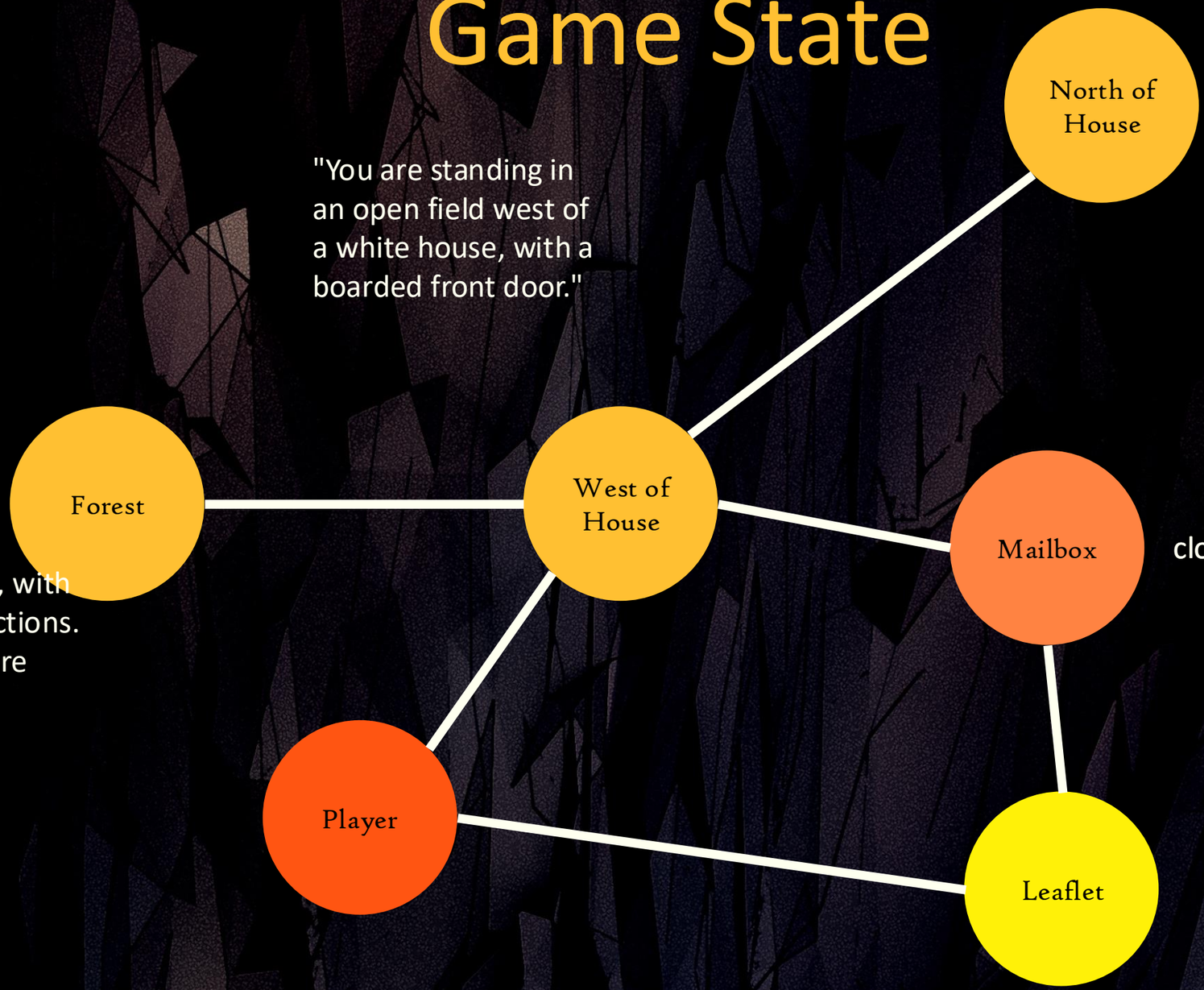
Notes: All horizontal passages leave the room in the direction shown. Vertical passages are labelled "U" for UP and "D" for DOWN. To avoid unnecessarily giving away problems, this map lists only those objects immediately visible upon entering a room. Only objects which can be taken or used are listed; objects which are merely part of a room are not. Where more than one direction leads to the same place, all are not necessarily shown.



For information and price lists for other games and products write:

ZORK USERS GROUP
P.O. BOX 20923
MILWAUKEE, WI 53220-0923

Game State


North of House
"You are facing the north side of a white house. There is no door here, and all the windows are boarded up. To the north a narrow path winds through the trees."
"You are standing in an open field west of a white house, with a boarded front door."
Forest
West of House
Mailbox
closed: false
"This is a forest, with trees in all directions. To the east, there appears to be sunlight."
Player
Leaflet

Main Game Loop

Describe

Parse

Process

Timers

You are standing in an open field west of a white house, with a boarded front door.

LOOK
OPEN MAILBOX
LOOK AT MAILBOX
TAKE LEAFLET FROM MAILBOX

West of House

Player

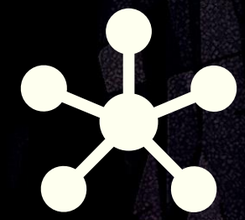
Leaflet

Mailbox

closed: false

Commands

Arbiter



Responses

Main Game Loop

Describe

You are standing in a field next to a white house. The front door is boarded shut. There is a rusty mailbox here leaning at an angle.

Parse

I open the mailbox. Is there anything in it?

Process

Remove
"Rusted Shut"

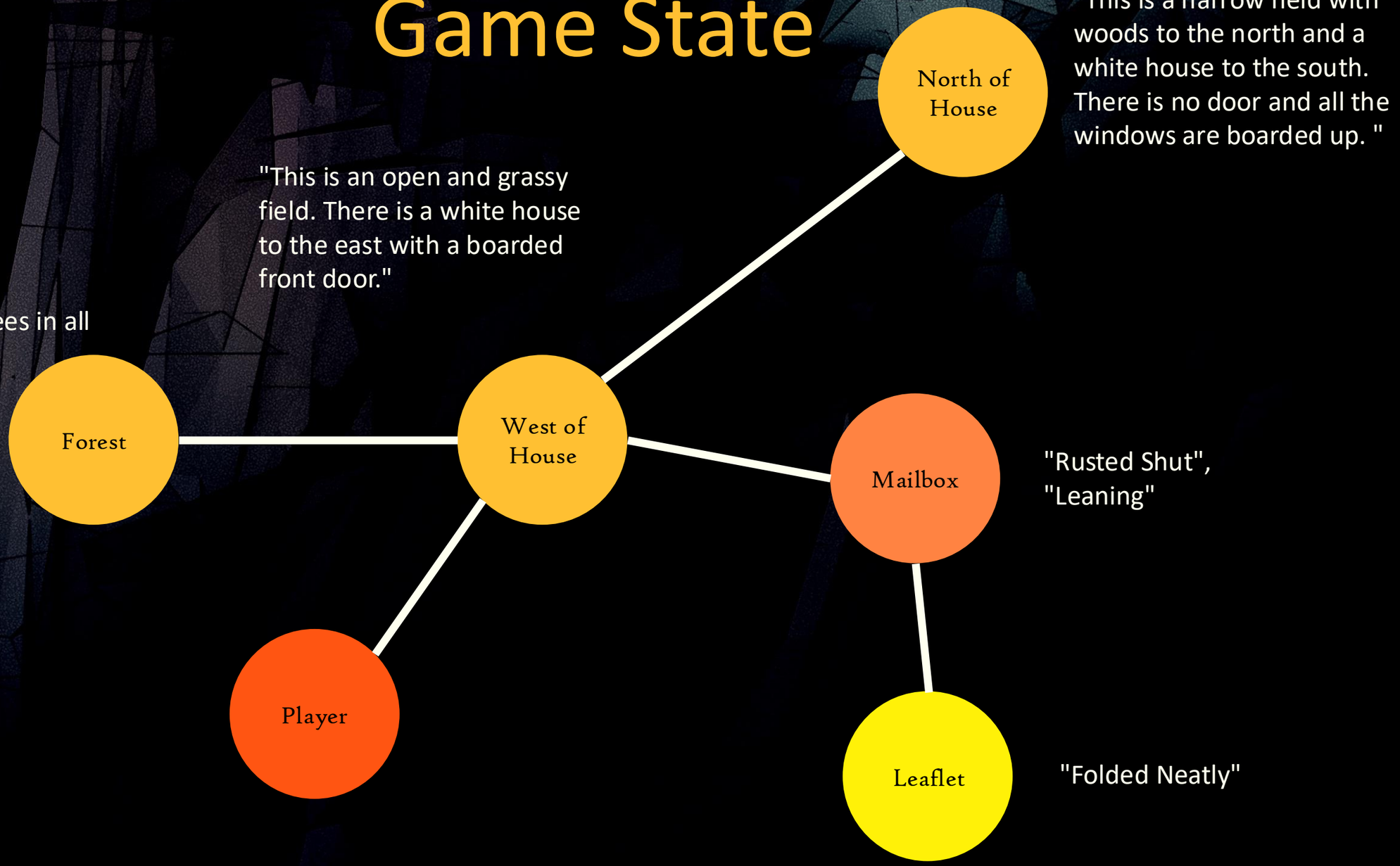
Add
"Open", "Rusty"

Timers

Player — West of House — Mailbox

"Leaning", "Open", "Rusty"

Game State


North of House
"This is a narrow field with woods to the north and a white house to the south. There is no door and all the windows are boarded up. "
"This is an open and grassy field. There is a white house to the east with a boarded front door."
"This forest has trees in all directions."
Forest
West of House
Mailbox
"Rusted Shut", "Leaning"
Player
Leaflet
"Folded Neatly"


Let's Build This


HIC HABITAT
GENIVS LOCI

Genii Singulorum Locorum

Forest

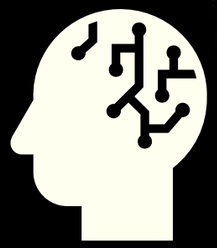
West of House

North of House

Mailbox

Leaflet

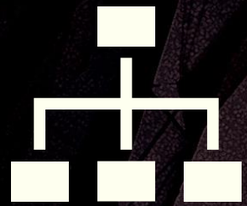
What's An Agent?



LLM

System Prompt

Tools



Composable

A Simple Agent

What does the brass lantern do?

It keeps you from being eaten by a grue.

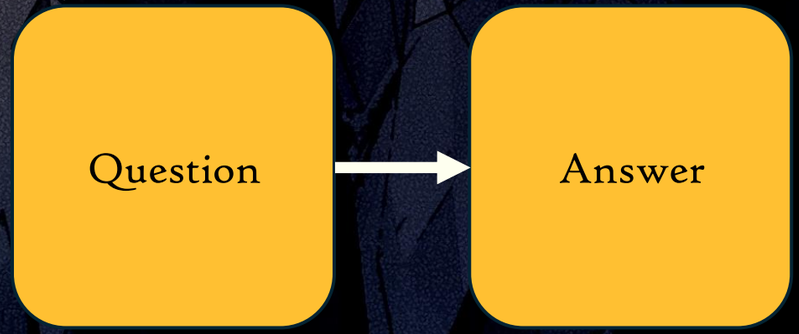
```
// create the graph
const graph = new StateGraph(MessagesAnnotation) as any

// add the node that is the agent
graph.addNode('zork_expert', async (state: typeof MessagesAnnotation.State) => {

  const SYSTEM_PROMPT = dedent`
    You are a helpful assistant that answers questions about the classic
    text adventure game Zork.`

  const llm = fetchLLM()

  const response = await llm.invoke([
    new SystemMessage(SYSTEM_PROMPT),
    state.messages[0]
  ])

  return { messages: [response] }

})

// add edges to start and stop the graph
graph.addEdge(START, 'zork_expert')
graph.addEdge('zork_expert', END)

// compile the graph
const workflow = graph.compile()
```

A Simple Agent

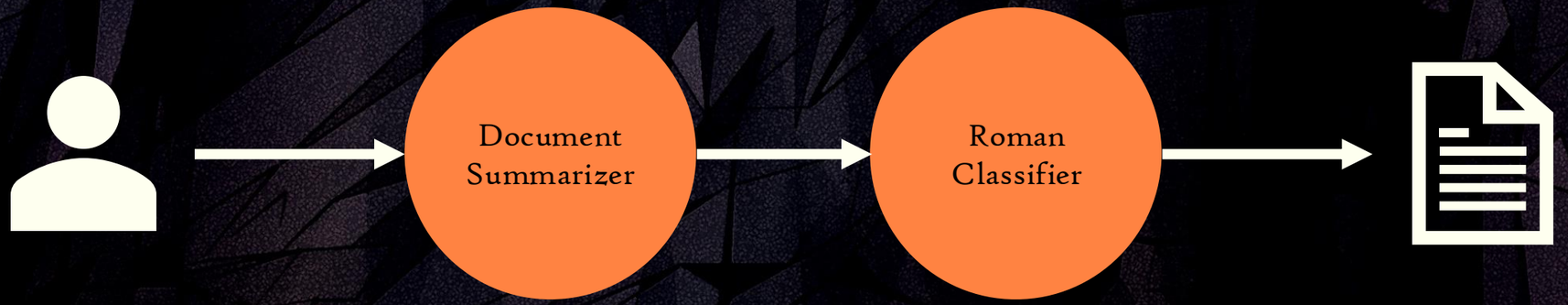
What does the brass lantern do?

It keeps you from being eaten by a grue.

```
// run the workflow
const response = await workflow.invoke({
  messages: [new HumanMessage('What does the brass lantern do?')]
})

// Return the answer which is the last message
const answer = response.messages [1] as AIMessage
```



Chained Agents



Before Julius Caesar came to power, he crossed to Rubicon. This was a symbolic act that marked the beginning of civil war.

Julius Caesar's crossing of the Rubicon River symbolized the start of a civil war before he came to power.

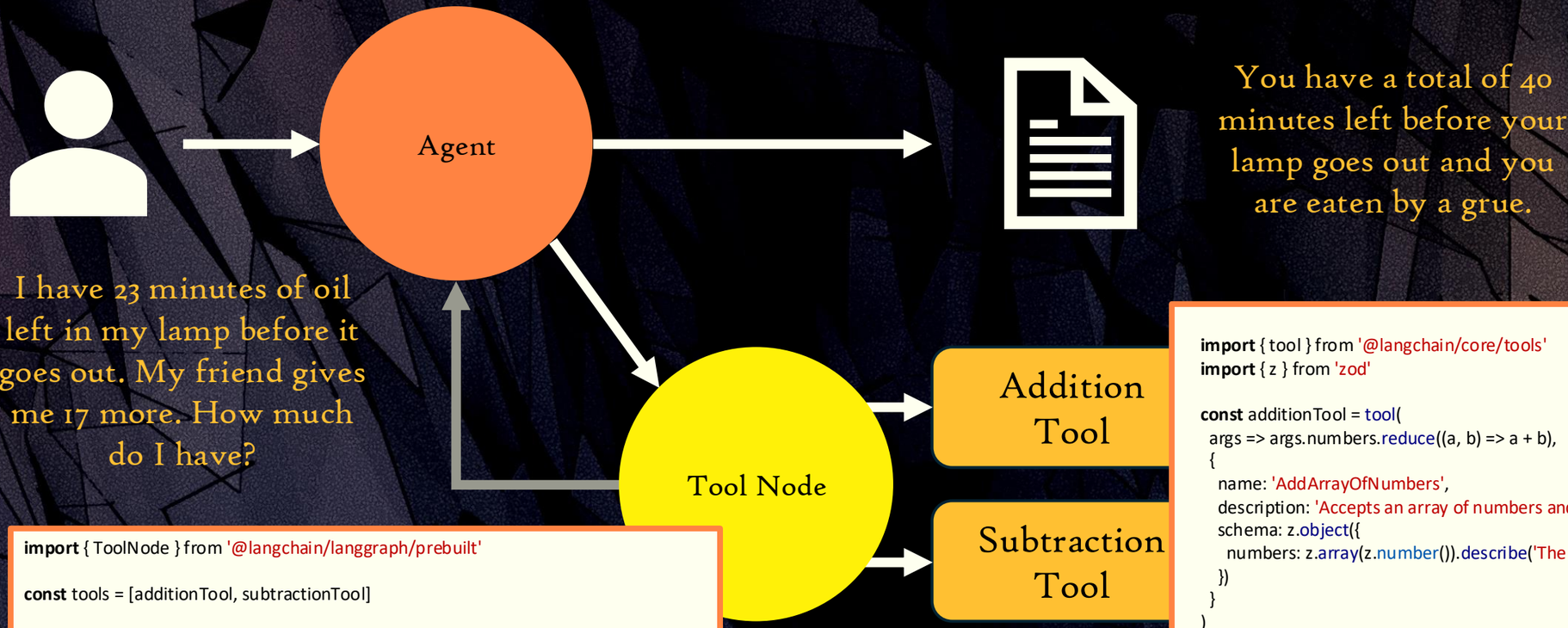
Yes, the civilization described in the passage is Ancient Rome.

Chained Agents

```
const graph = new StateGraph(MessagesAnnotation) as any

graph.addNode('summarizer', async (state: typeof MessagesAnnotation.State) => {
  const SYSTEM_PROMPT = dedent`
    You are a summarization AI. You will be provided with a text passage
    that contains historical information about ancient civilizations. Your
    job is to summarize the text passage.`

  const response = await llm.invoke([
    new SystemMessage(SYSTEM_PROMPT),
    state.messages[0]
  ])

  return { messages: [response] }
})

graph.addNode('classifier', async (state: typeof MessagesAnnotation.State) => {
  const SYSTEM_PROMPT = dedent`
    You are a civilizational classifier AI. You will be provided with a text
    passage that is contains information about an ancient civilization. You
    need to determine if that civilization is Ancient Rome or not.`

  const response = await llm.invoke([
    new SystemMessage(SYSTEM_PROMPT),
    state.messages[1]
  ])

  return { messages: [response] }
})
```

```
graph.addEdge(START, 'summarizer')
graph.addEdge('summarizer', 'classifier')
graph.addEdge('classifier', END)
```

Tool Using Agent

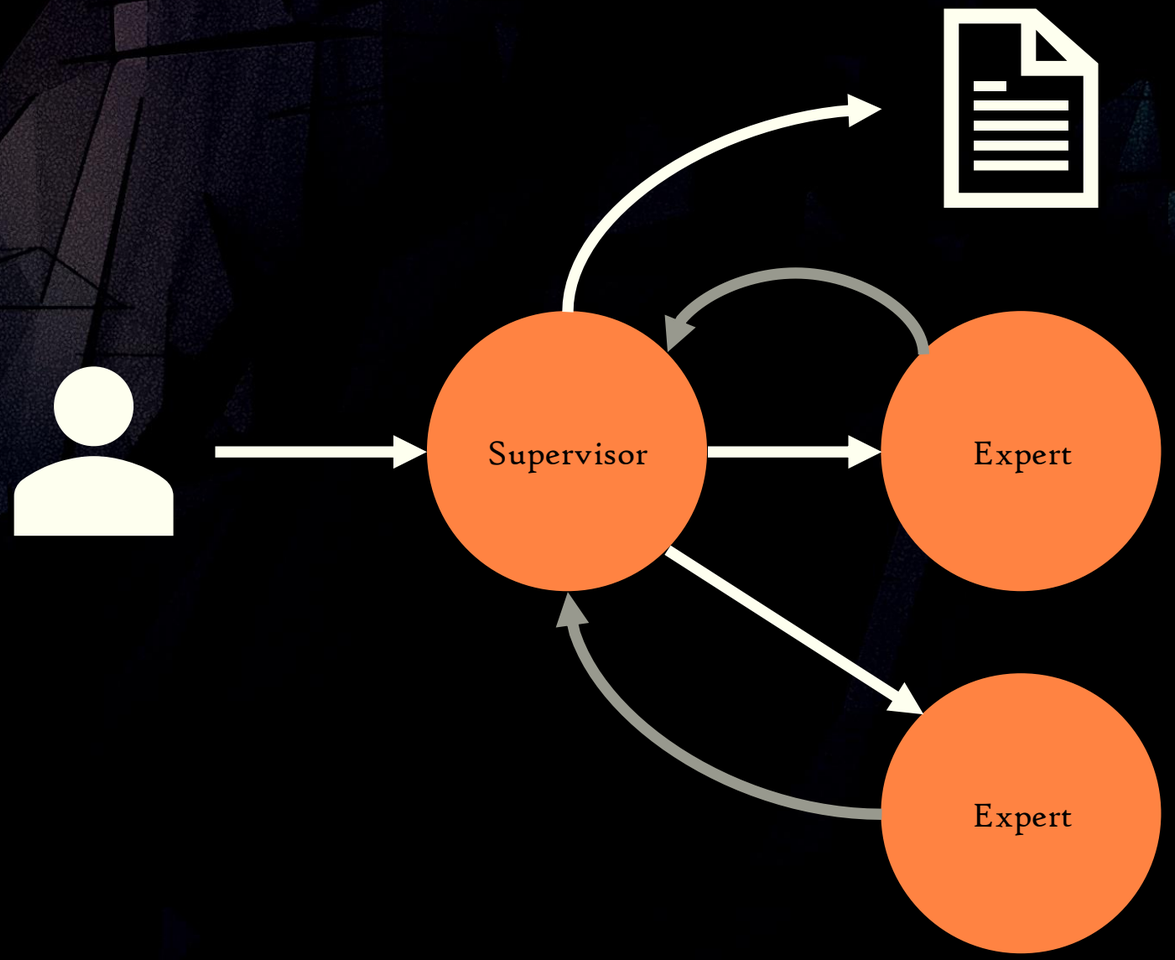



```
import { ToolNode } from '@langchain/langgraph/prebuilt'

const tools = [additionTool, subtractionTool]

// Bind the tool to the LLM
const llm = fetchLLM().bindTools(tools)

// Add the tools node that executes tool calls
graph.addNode('tools', new ToolNode(tools))
```

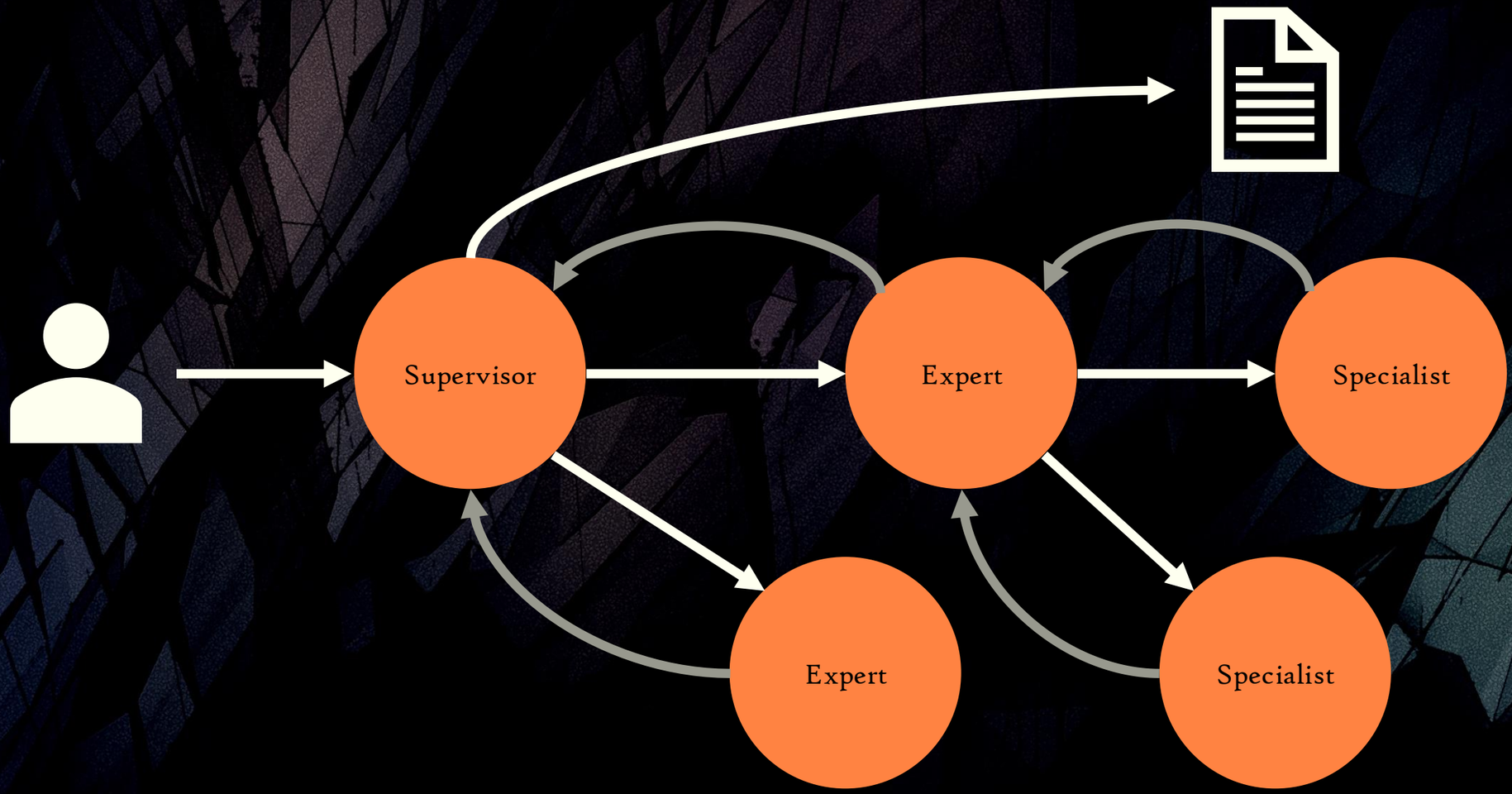
```
import { tool } from '@langchain/core/tools'
import { z } from 'zod'

const additionTool = tool(
  args => args.numbers.reduce((a, b) => a + b),
  {
    name: 'AddArrayOfNumbers',
    description: 'Accepts an array of numbers and adds them together',
    schema: z.object({
      numbers: z.array(z.number()).describe('The numbers to add together')
    })
  }
)
```

```
graph.addConditionalEdges(
  'agent',
  (state: typeof MessagesAnnotation.State) => {
    const lastMessage = state.messages[state.messages.length - 1] as AIMessage
    const areThereTools = lastMessage.tool_calls !== undefined && lastMessage.tool_calls.length > 0
    return areThereTools ? 'tools' : END
  },
  ['tools', END]
)
```

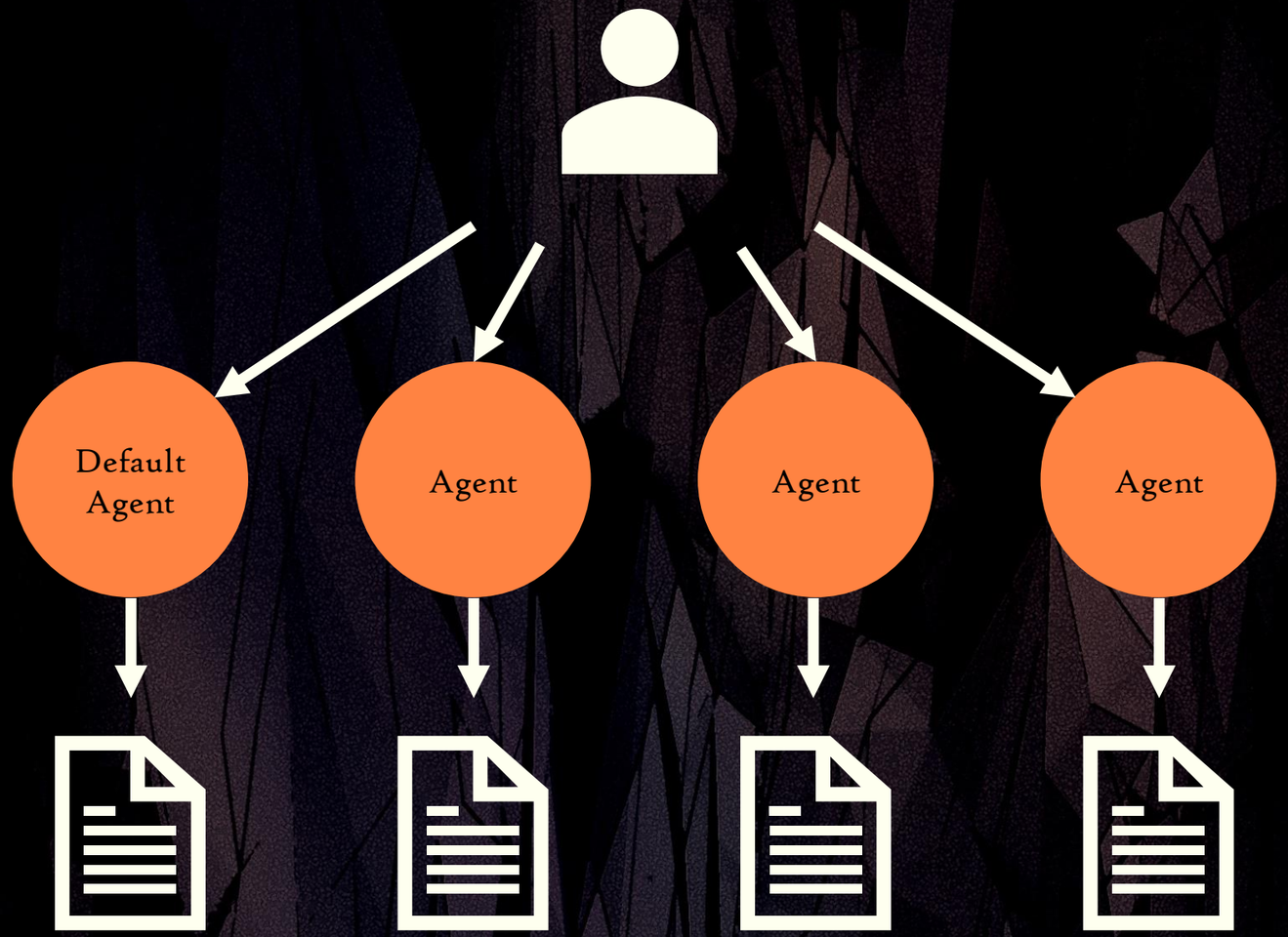
Supervisor



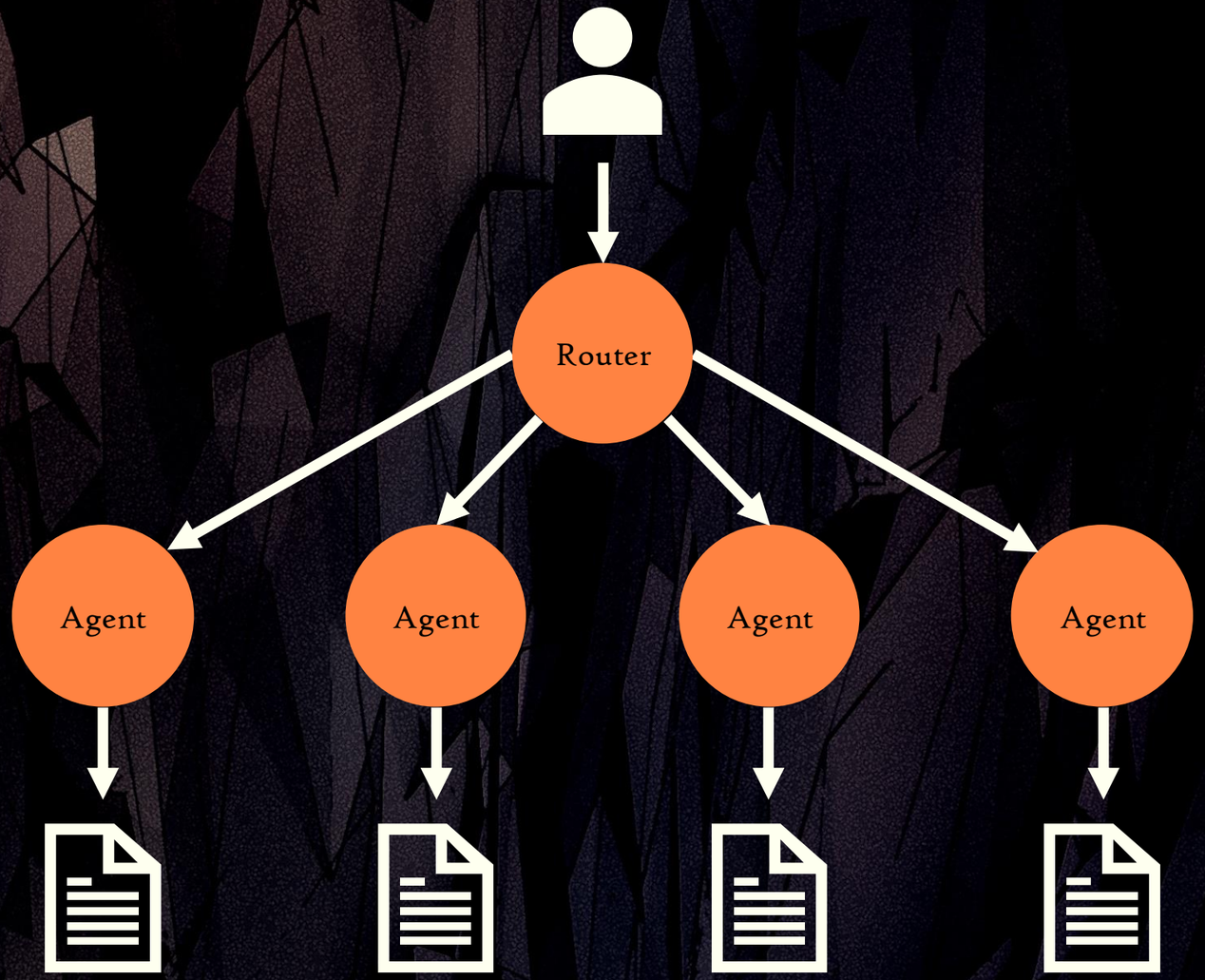
Hierarchical



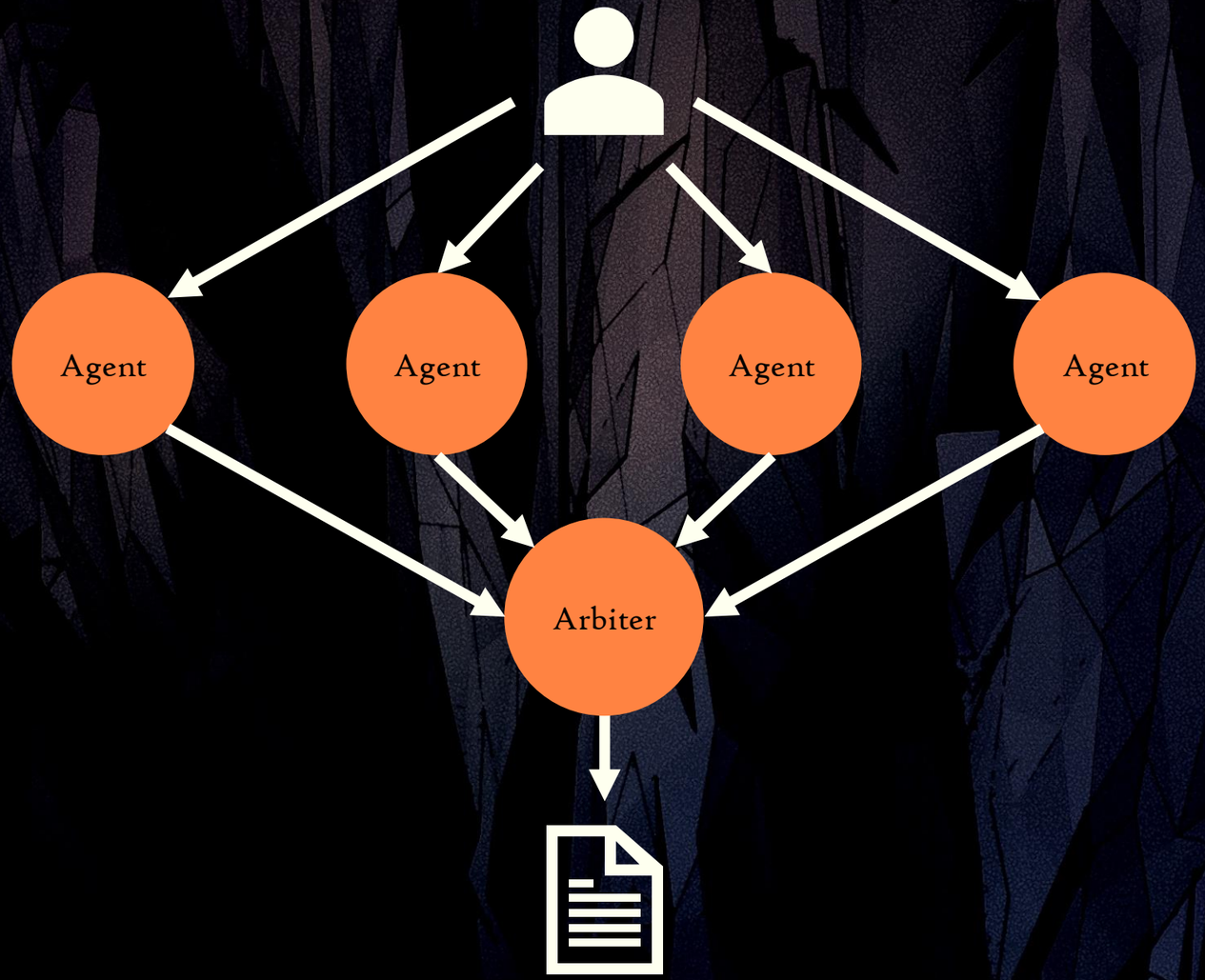
Swarm



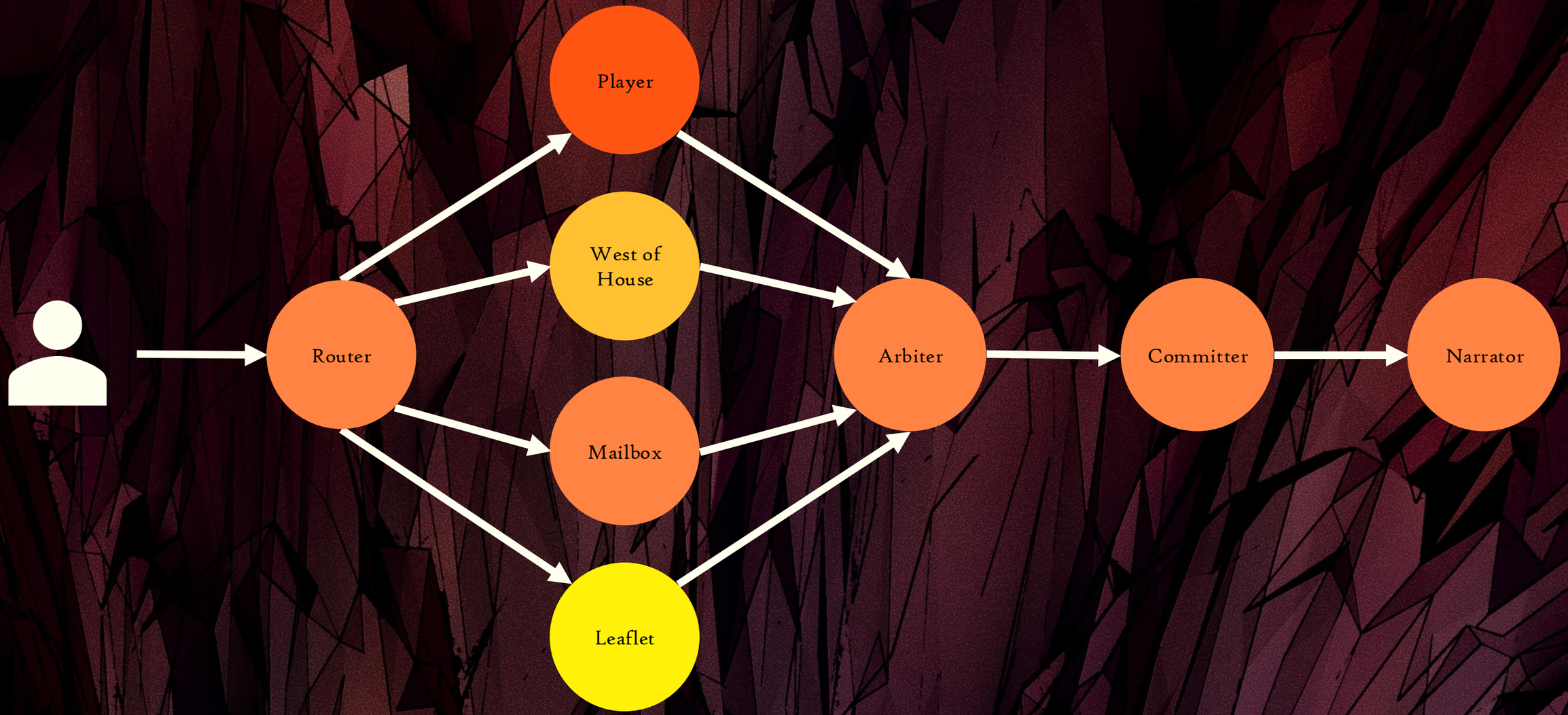
Router



COMMITTEE



Agents & Arbiters Workflow

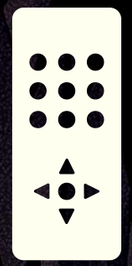



Annotations

What's An Annotation?

Channelized state **with merge semantics**

State

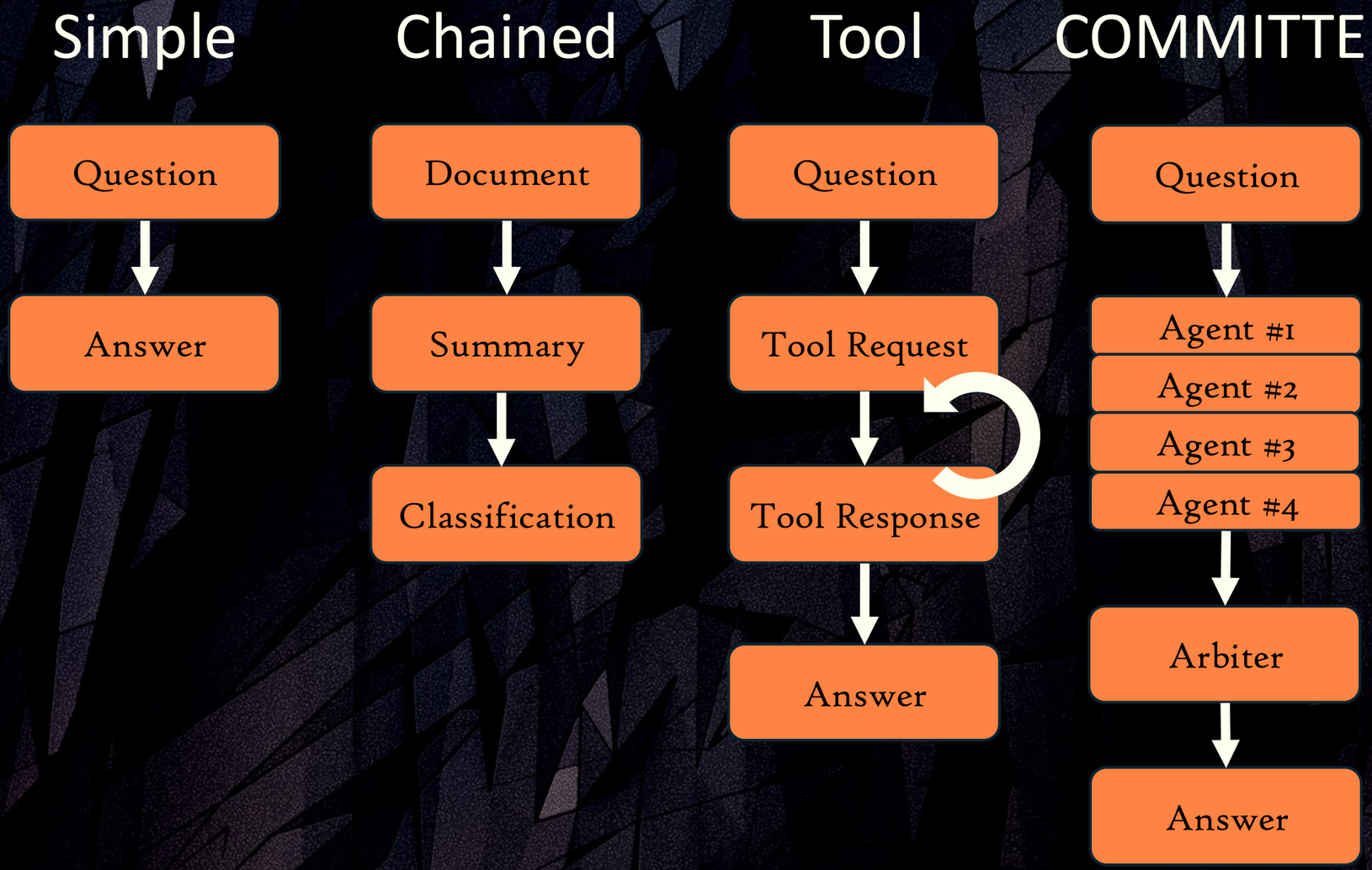


Channel

Accumulator

Reducer

Message Annotation



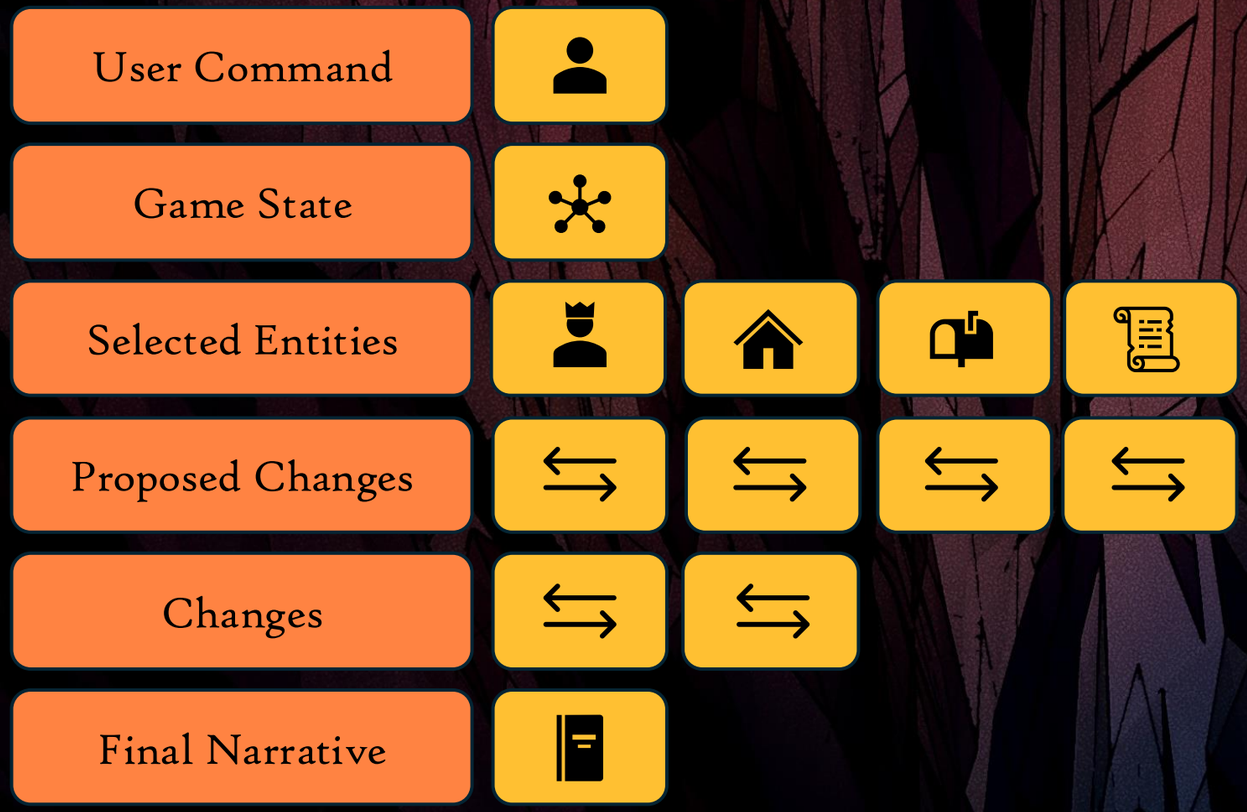
Message Annotation

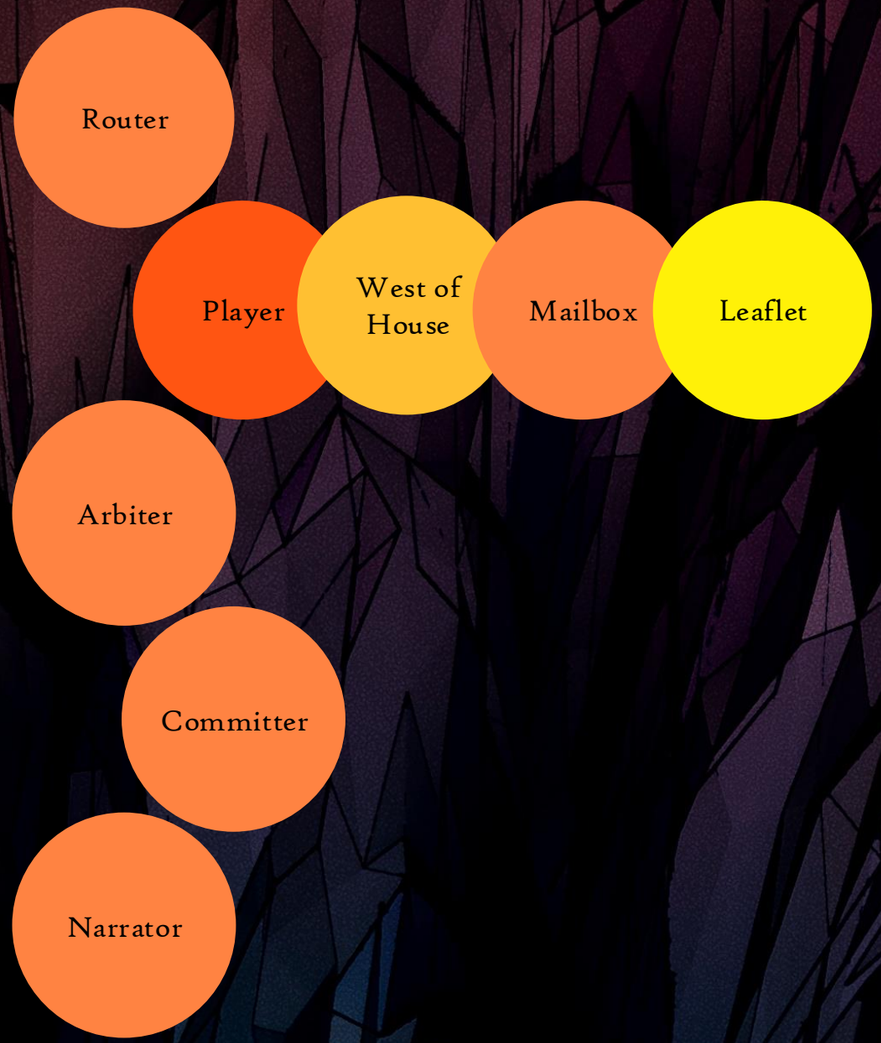
```
graph.addNode('zork_expert', async (state: typeof MessagesAnnotation.State) => {

  const SYSTEM_PROMPT = dedent`
    You are a helpful assistant that answers questions about the classic
    text adventure game Zork.`

  const llm = fetchLLM()

  const response = await llm.invoke([
    new SystemMessage(SYSTEM_PROMPT),
    state.messages[0]
  ])

  return { messages: [response] }

})

// add edges to start and stop the graph
graph.addEdge(START, 'zork_expert')
graph.addEdge('zork_expert', END)

// compile the graph
const workflow = graph.compile()

// run the workflow
const response = await workflow.invoke({
  messages: [new HumanMessage('What does the brass lantern do?')]
})

// Return the answer which is the last message
const answer = response.messages[1] as AIMessage
```

```
const graph = new StateGraph(MessagesAnnotation)
```

```
const MessagesAnnotation = Annotation.Root({
  messages: Annotation<BaseMessage[]>({
    default: () => [],
    reducer: (prev, next) => [...prev, ...next]
  })
})
```

Multi-Channel Annotation





```ts
export const GameTurnAnnotation = Annotation.Root({
  userCommand: Annotation<string | null>({
    reducer: (_prev, next) => next,
    default: () => null
  }),

  gameState: Annotation<GameState | null>({
    reducer: (_prev, next) => next,
    default: () => null
  }),

  selectedEntities: Annotation<SelectedEntity[]>({
    reducer: (_prev, next) => next,
    default: () => []
  }),

  entityChangeRecommendations: Annotation<EntityChangeRecommendation[] | EntityChangeRecommendation>({
    reducer: (prev, next) => {
      const result = []
      Array.isArray(prev) ? result.push(...prev) : result.push(prev)
      Array.isArray(next) ? result.push(...next) : result.push(next)
      return result
    },
    default: () => []
  }),

  entityChanges: Annotation<EntityChange[]>({
    reducer: (_prev, next) => next,
    default: () => []
  }),

  finalNarrative: Annotation<string | null>({
    reducer: (_prev, next) => next,
    default: () => null
  })
})
```

```ts
const graph = new StateGraph(GameTurnAnnotation)
```

```ts
const gameState = await GameState.fetch(gameId)
const command = 'I open the mailbox and look inside'

const result = await workflow.invoke({
  userCommand: command
  gameState: gameState,
})

async function router(state: typeof GameTurnAnnotation.State): Promise<ClassifierReturnType> {

  const gameState = state.gameState
  const userCommand = state.userCommand

  const llm = await fetchLLMClient()
  const prompt = buildPrompt(gameState, userCommand)
  const selection = await llm.invoke(prompt)

  return { selectedEntities: selection.selectedEntities }
}
```


Memory

Without Memory

You are West of the House. There is a rusty mailbox on a post leaning at an angle.

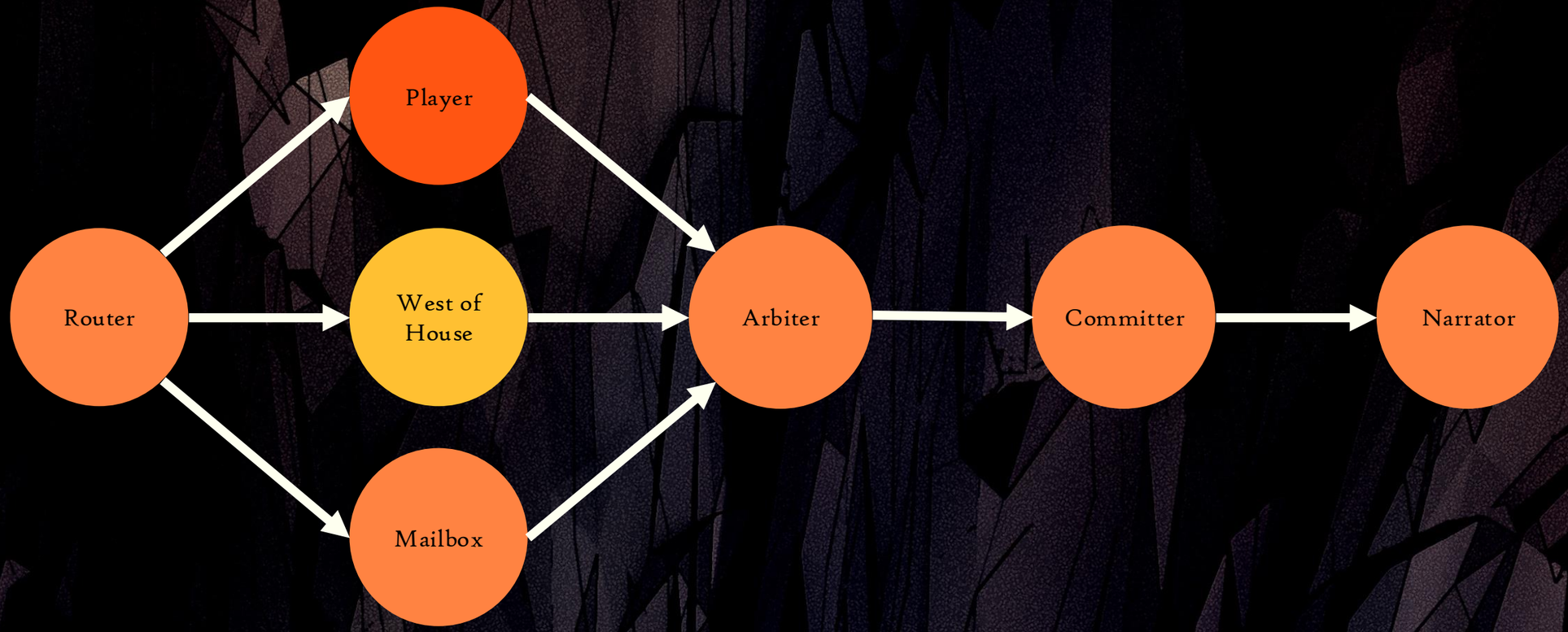
I open the mailbox.

You yank on the mailbox and manage to break through years of rust. The mailbox is now open.

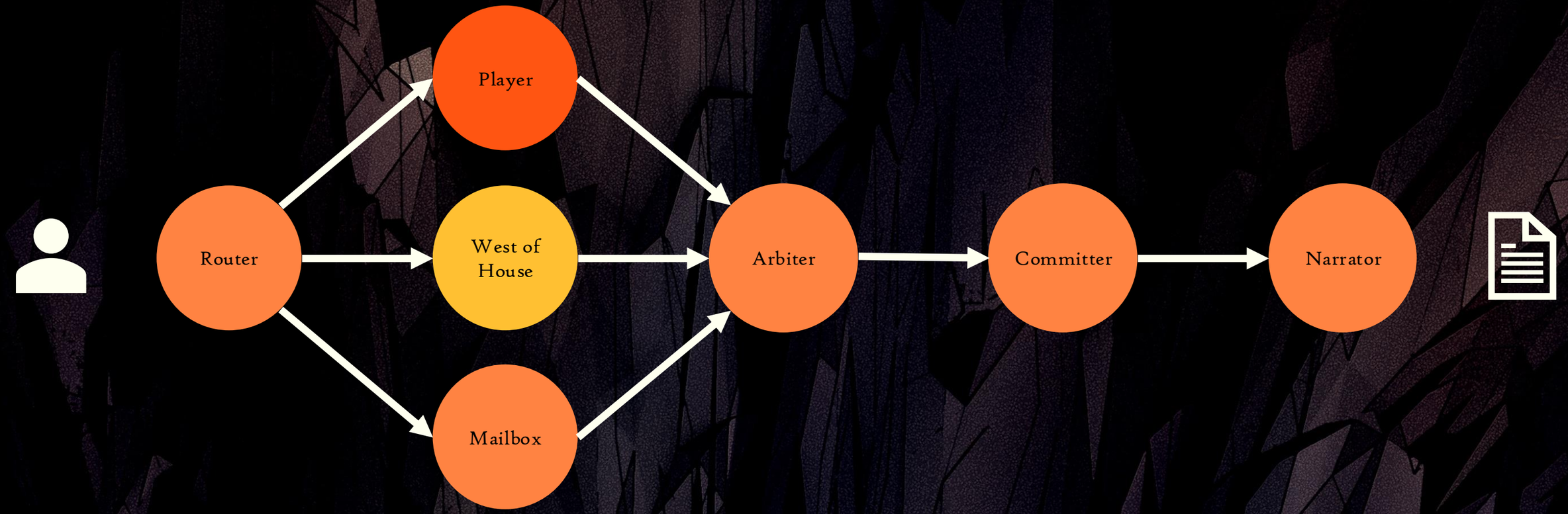
Is there anything in it?

In what?

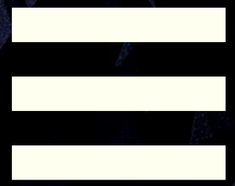
Checkpointer



Checkpointer

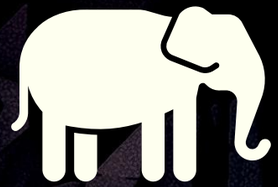


Agent Memory Server



Store

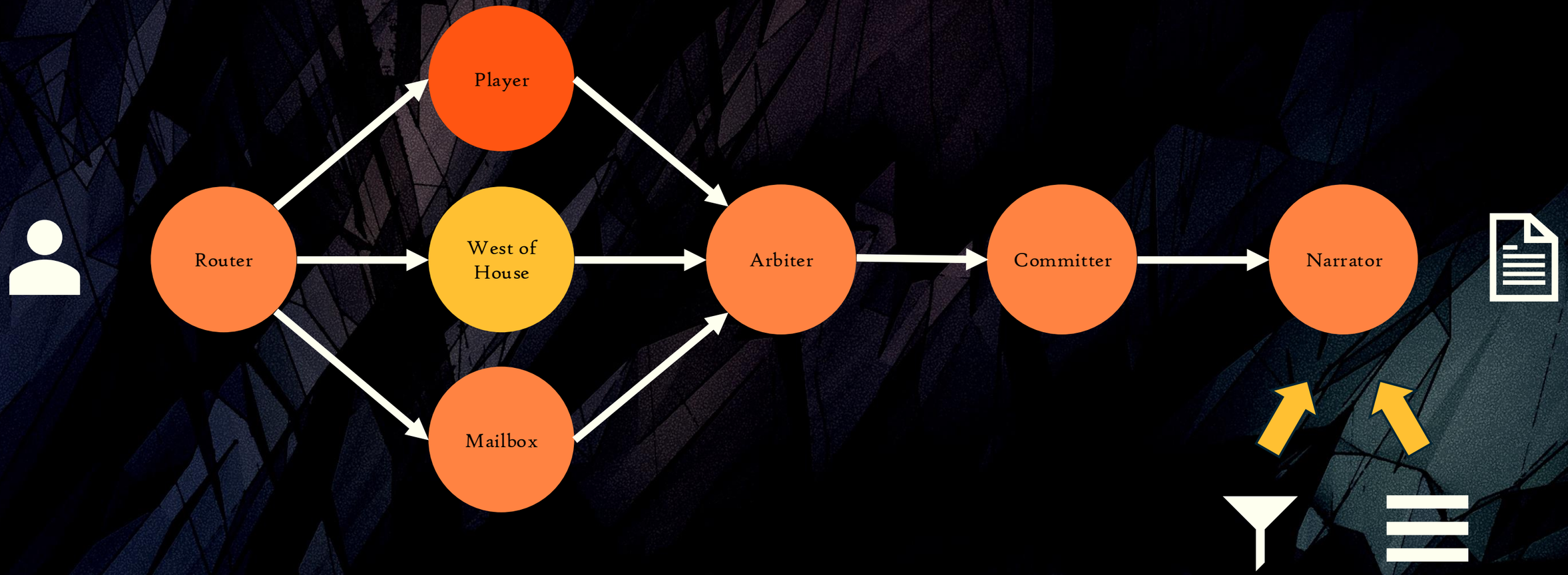
COMPACT



Extract Memories

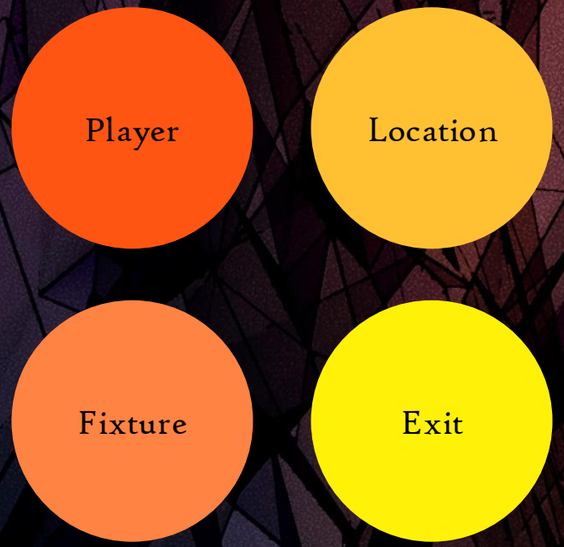
AUTOMATIC

Agent Memory Server



Architecture

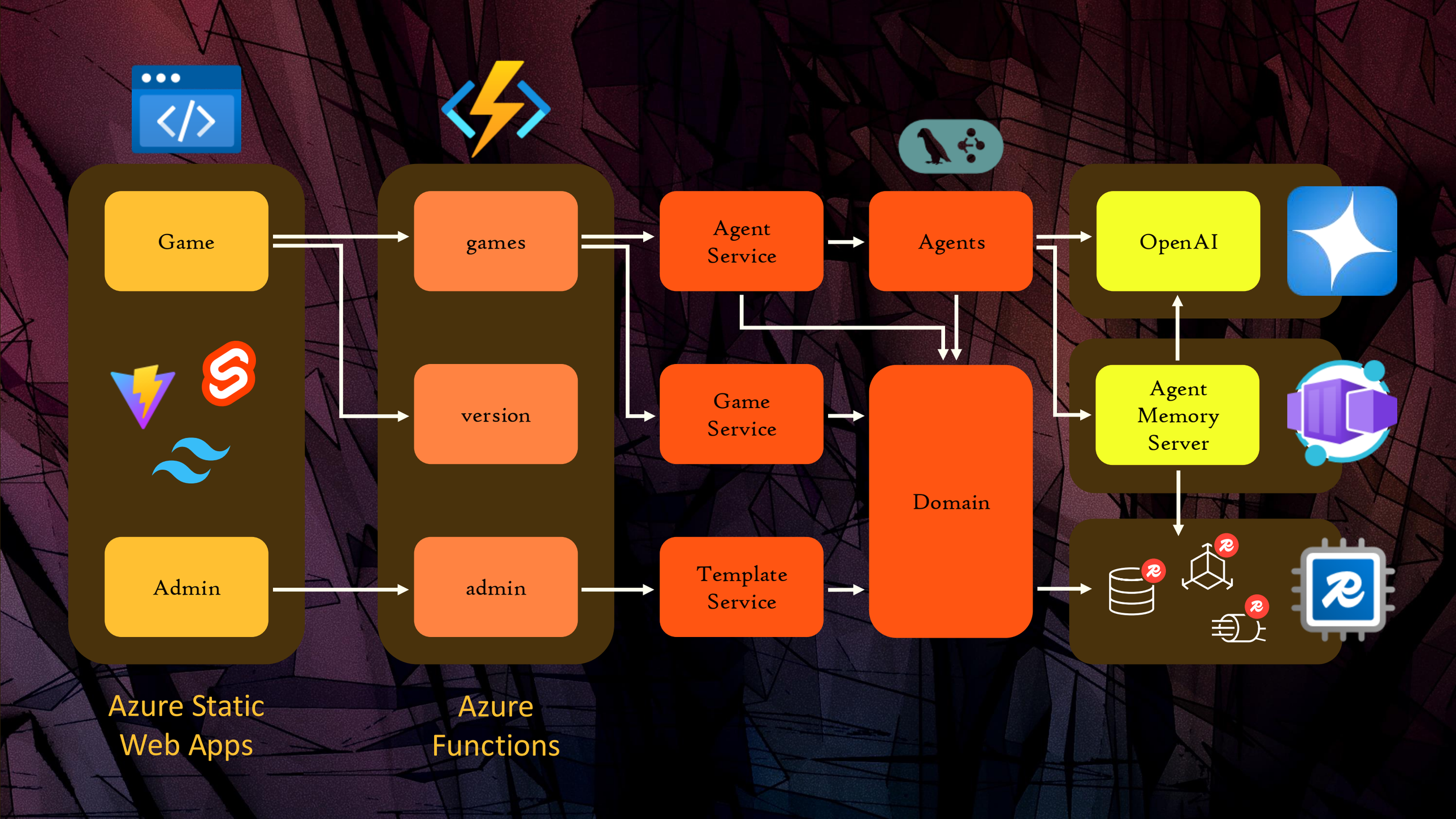
Storing JSON in Redis




```
{
  "id": "01KEJ2Y2QTZ1H08RCPTD04V1ZZ",
  "name": "Mailbox",
  "description": "It's a mailbox on a post.",
  "statuses": [
    "Rusty",
    "Leaning",
    "Rusted Shut"
  ],
  "location": "01KEJ2GTMKT1Z6JRNZJPY3YMH3",
  "fixtureIds": [
    "01KEJ2P51HZXXTGD1VWJ5ZD4RE",
    "01KEJ2P51K2ZA2Y8Y8B1D6FE4D",
    "01KEJ2P51KQD9T21MB8CHCZSN7"
  ],
  "exitIds": [
    "01KEJ2P51MP7P61CVQAB8NBTR0",
    "01KEJ2P51MT3BQNQTFRQ9TFK5C"
  ],
  "destinationId": " 01KEJ2P51K9GZG96GW2PF3H596"
}
```

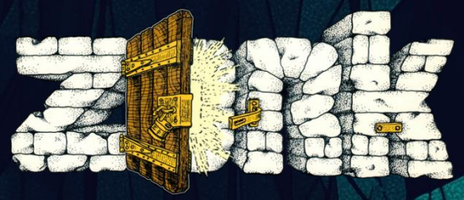
```
await redisClient.json.merge(key, '$', json)
```

```
JSON.MERGE \
  game:01KEJ2Y2QTZ1H08RCPTD04V1ZZ:entity:01KEJ2Y2QTZ1H08RCPTD04V1ZZ \
  '$' '{"id":"01KEJ2Y2QTZ1H08RCPTD04V1ZZ","name":...
```


Game
Admin
games
version
admin
Agent Service
Agents
Game Service
Template Service
Domain
OpenAI
Agent Memory Server
Azure Static Web Apps
Azure Functions

Demo

Resources



zilf.io
github.com/historicalsource/zork1
github.com/historicalsource/zork2
github.com/historicalsource/zork3

github.com/langchain-ai/langgraphjs
docs.langchain.com/oss/javascript/langgraph/overview

Redis

redis.io
discord.gg/redis
university.redis.io
redis.github.io/agent-memory-server

github.com/guyroyse/
agents-n-arbiters

Guy Royse

Developer Advocate

Redis

@guy.dev

@guyroyse

github.com/guyroyse

guy.dev


Thanks