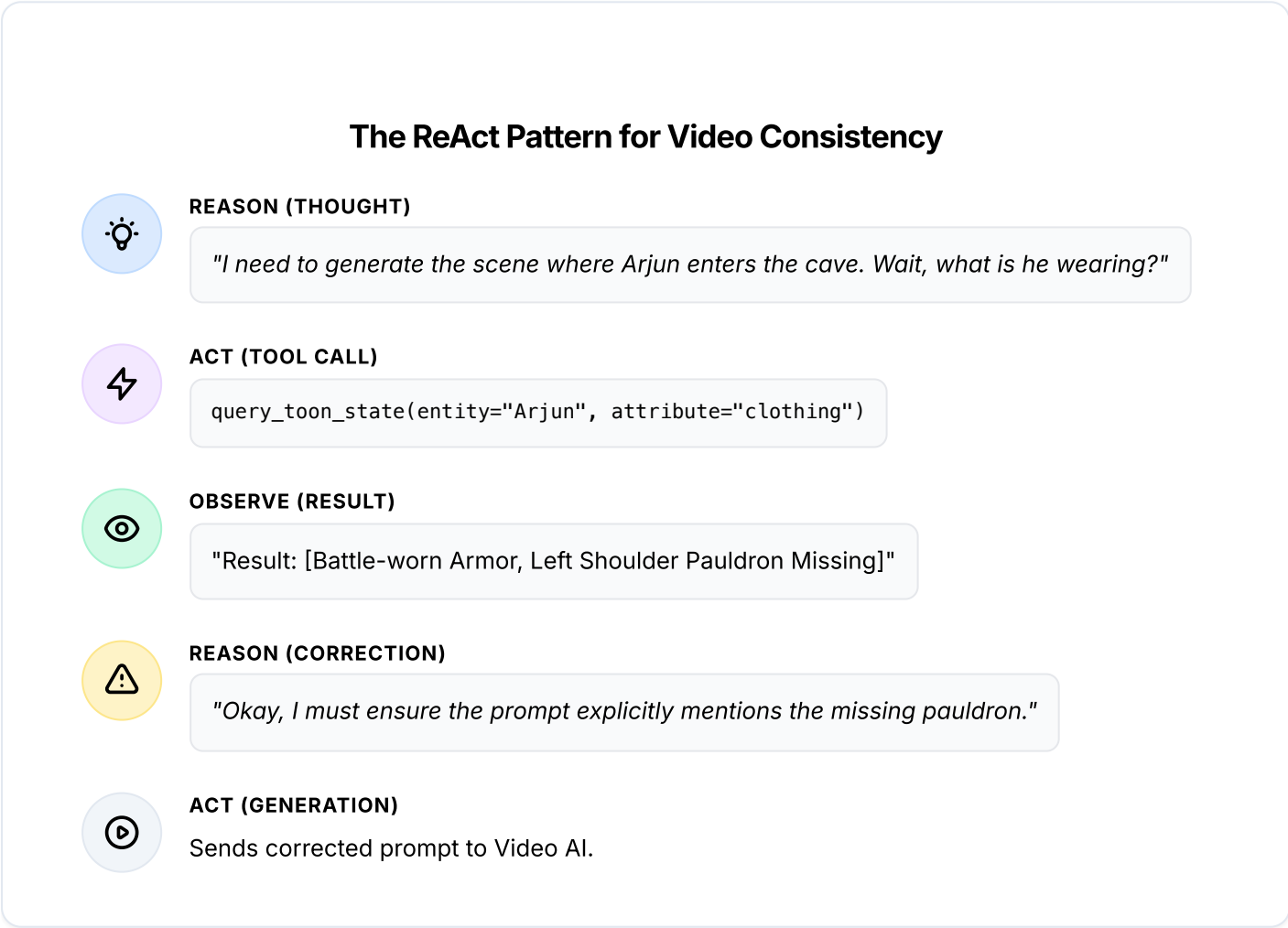


Orchestrator Architecture

If "TOON" is the language your app speaks, the Orchestrator is the Director who speaks it. We replace linear pipelines with a "Loop of Reasoning" that mimics a human film director checking their work.

The Core Concept: The "Showrunner" Loop

In a standard LLM script, you fire a prompt and hope for the best. In an Orchestrator (ReAct) model, we build a **Circular Dependency**. The Orchestrator (Showrunner) refuses to generate a video frame until the "Context State" is verified.



The Architecture: "The Supervisor Pattern"

We implement a Multi-Agent Orchestrator. You don't just have "One Bot"; you have a virtual film crew managed by a Supervisor.



Narrative Extractor

THE SCRIPTWRITER

Role: Reads the novel chunk.

Tool: Text_Compressor.

Output: Raw Scene Description.



Continuity Guard

THE SCRIPT SUPERVISOR

Role: Checks TOON database.

Tool: TOON_Retrieve.

Output: "Correction: Night time, Arjun bleeding."



Prompt Engineer

THE CINEMATOGRAPHER

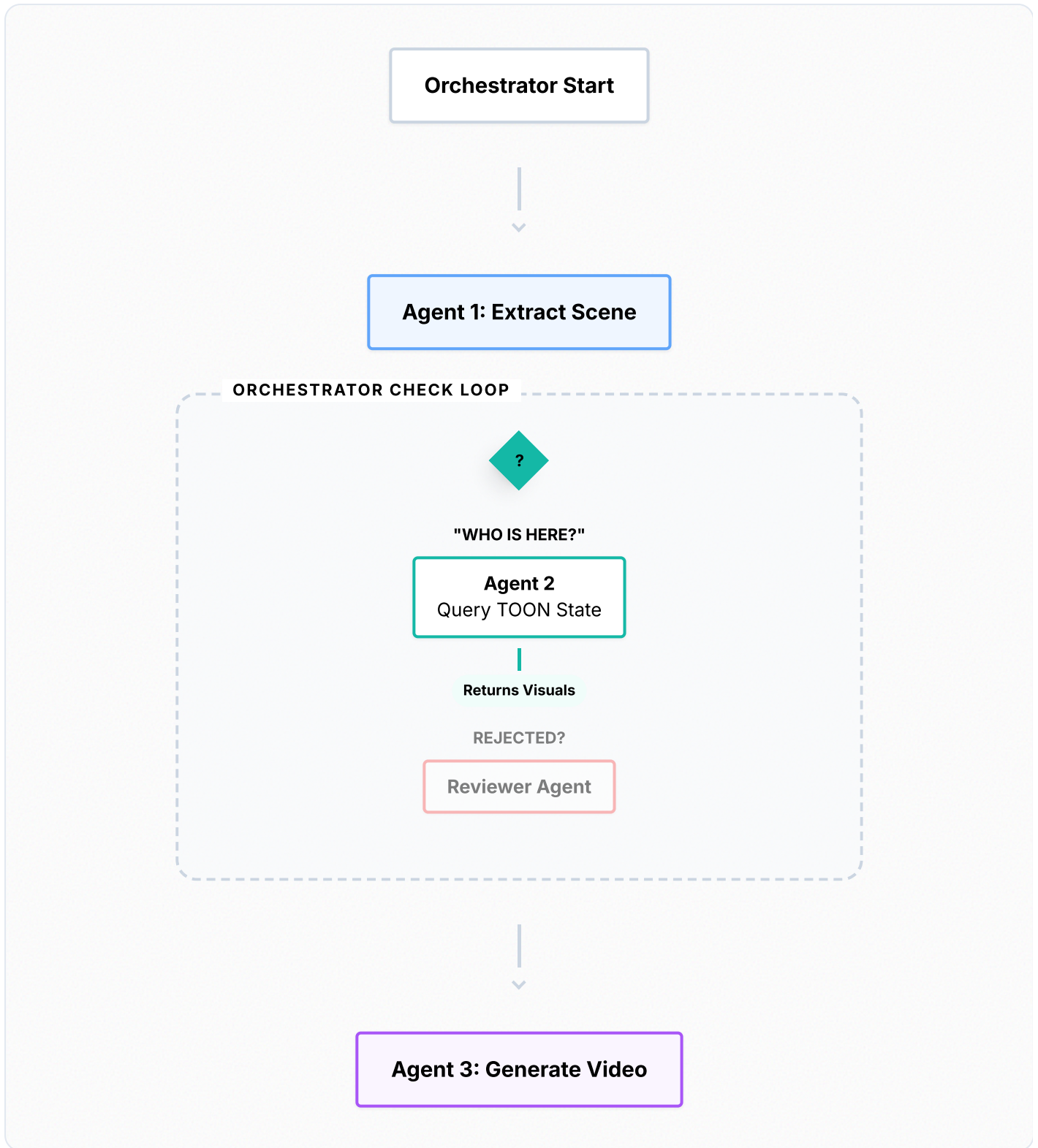
Role: Merges Script + Consistency Check.

Tool: Video_Prompt_Generator.

Output: Final Stable Diffusion Prompt.

Logic Flow: The State Graph

The Orchestrator code binds them together. It runs a loop that blocks generation until consistency is met.



Implementation Details

The Orchestrator State

Using TypeScript interfaces to define the "Showrunner's Clipboard".

orchestrator.ts

```
interface ShowrunnerState {  
  current_chunk: string;  
  toon_history: string; // The compressed context  
  visual_plan: string | null;  
  video_url: string | null;  
  retry_count: number;  
}
```

Why This Solves "Hallucination"

In standard LLM apps, the model forgets that a character lost their sword three scenes ago.

- **Without Orchestrator:** The model guesses.
- **With Orchestrator:** The "Continuity Guard" agent forces the "Cinematographer" agent to include "No Sword" in the negative prompt or description before the request is ever sent to the Video AI.