Fall 2024

Concrete Architecture

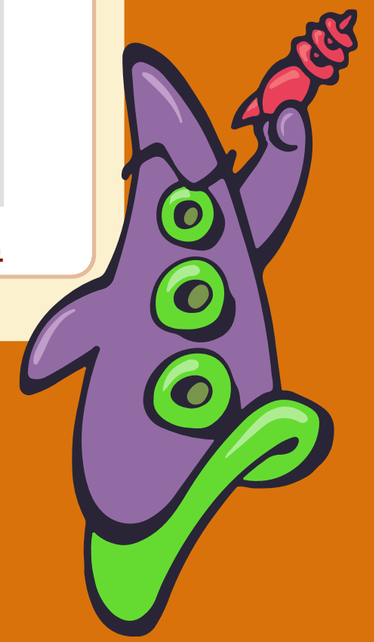
Oct 4, 2024

Team Null Terminators
CISC 322 A2
A Concrete Overview of ScummVM & The SCI Engine’s Architecture

https://youtu.be/EOSLWBhGbsw

High Resolution Architecture Diagrams Images

Explore Presentation...



Our Team

Hayden Jenkins

Presenter
ScummVM Concrete Arch
Presentation + Recording

Benjamin Leray

Presenter
Conclusion
SCI Engine Presentation Script

Ryan Van Drunen

Group Lead
Sequence Diagrams

Simon Nair

ScummVM Reflexion Analysis

Jeremy Strachan

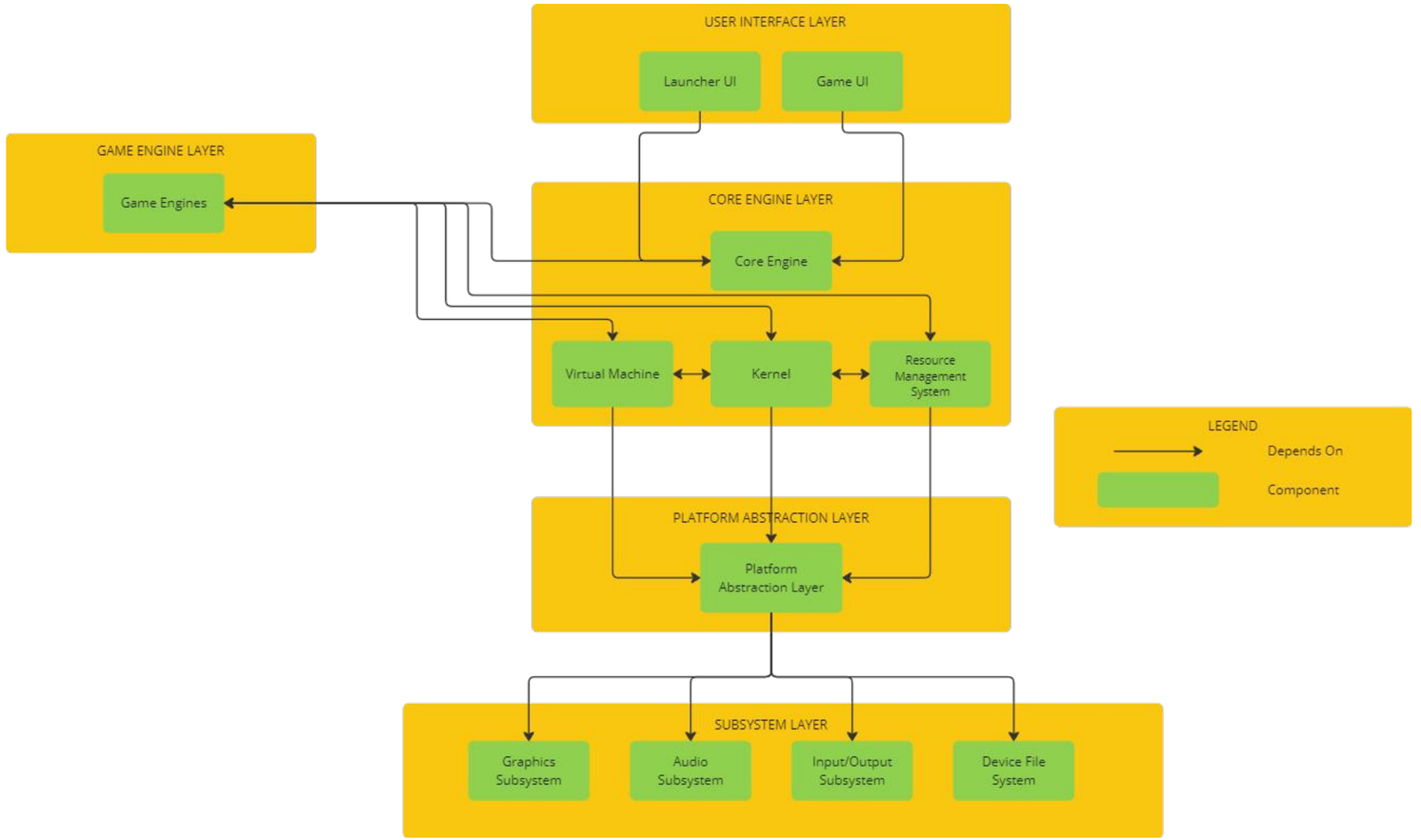
Intro + Abstract + Derivation

Corey Mccann

SCI Engine Arch + Reflexion

Review of Conceptual Architecture

Conceptual Architecture



Derivation Process

Conceptual Components and Subsystems

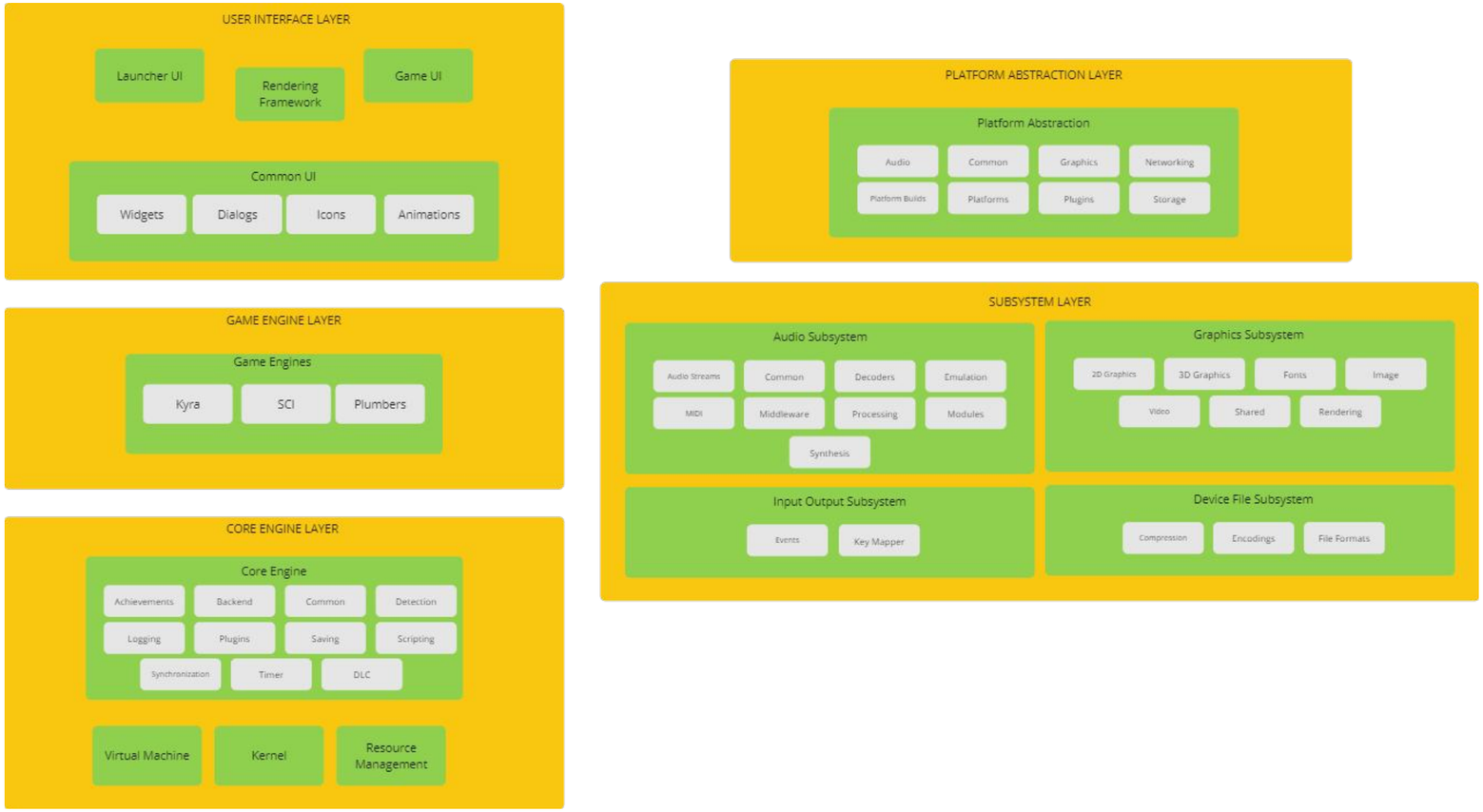
Launcher UI	Game UI	Game Engines	Core Engine
Virtual Machine	Kernel	Resource Management System	Platform Abstraction
Graphics Subsystem	Audio Subsystem	Input/Output Subsystem	Device File System

Steps

- Leveraged the conceptual architecture from Assignment 1 as a foundation
- Recreated the conceptual components and subsystems in SciTools Understand to serve as a starting structure

Derivation Process

Initial Iteration

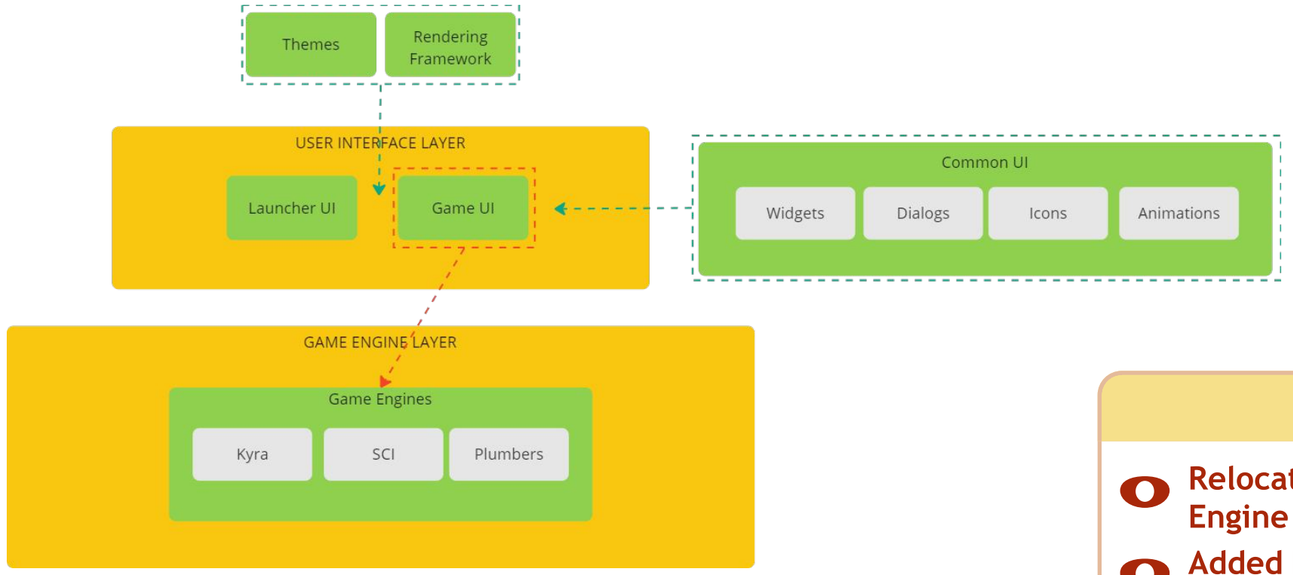


Steps

- Mapped files and modules from the codebase to the defined components and layers
- Used SciTools Understand to generate a dependency graph for the system
- Identified divergences and unexpected dependencies, refining the architecture accordingly

Concrete Architecture

Reflexion Analysis

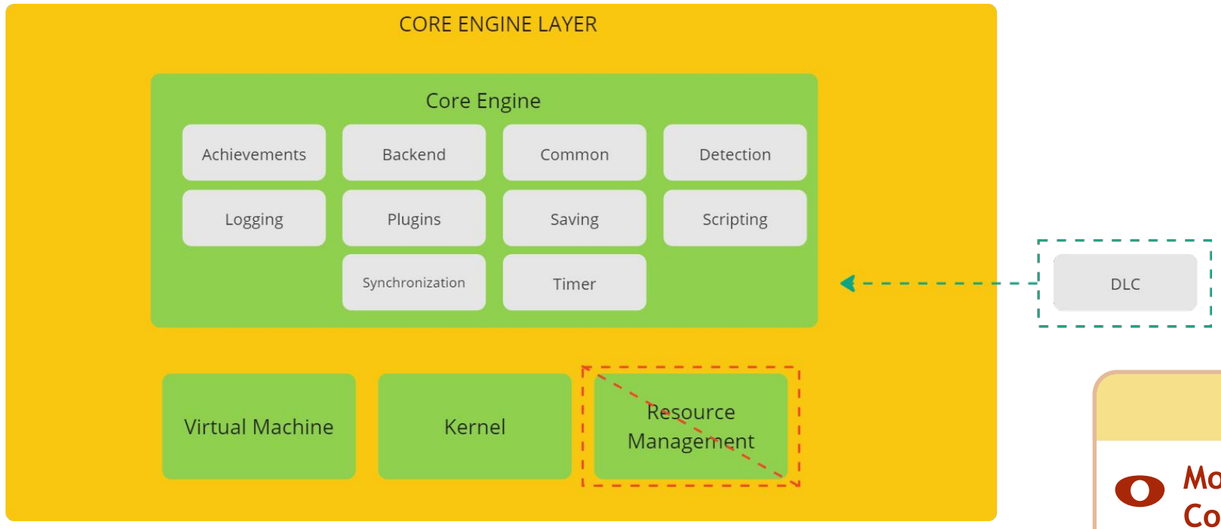


UI Layer Changes

- Relocated Game UI: Moved from User Interface Layer to Game Engine Layer for direct management by game engines
- Added Rendering Framework: Serves as a GUI manager, handling all UI rendering logic
- Introduced Common UI: Reduces code duplication with reusable elements like widgets, dialogs, and animations

Concrete Architecture

Reflexion Analysis

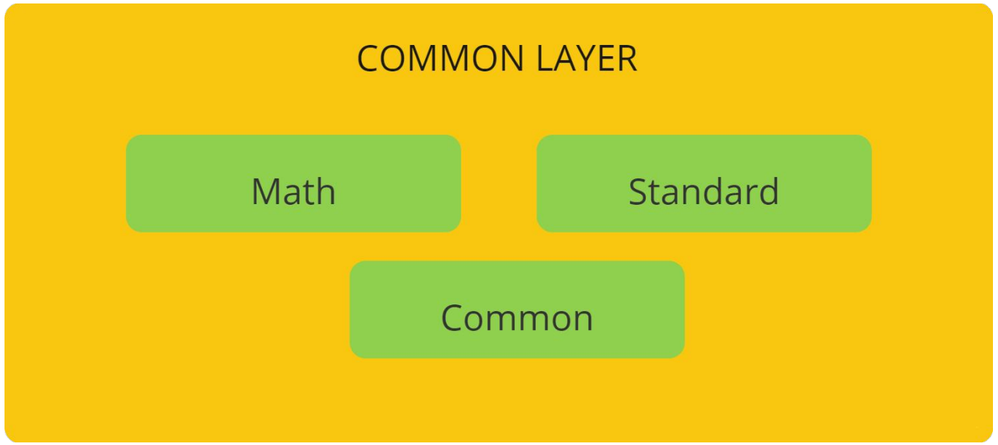


Core Engine Layer Changes

- Moved RMS: Moved resource management system into the Core Engine component, allowing games engines to manage resources directly
- Increased Emphasis on PAL: Shifted responsibility to the Platform Abstraction Layer for direct resource management with the host platform and other subsystems

Concrete Architecture

Reflexion Analysis

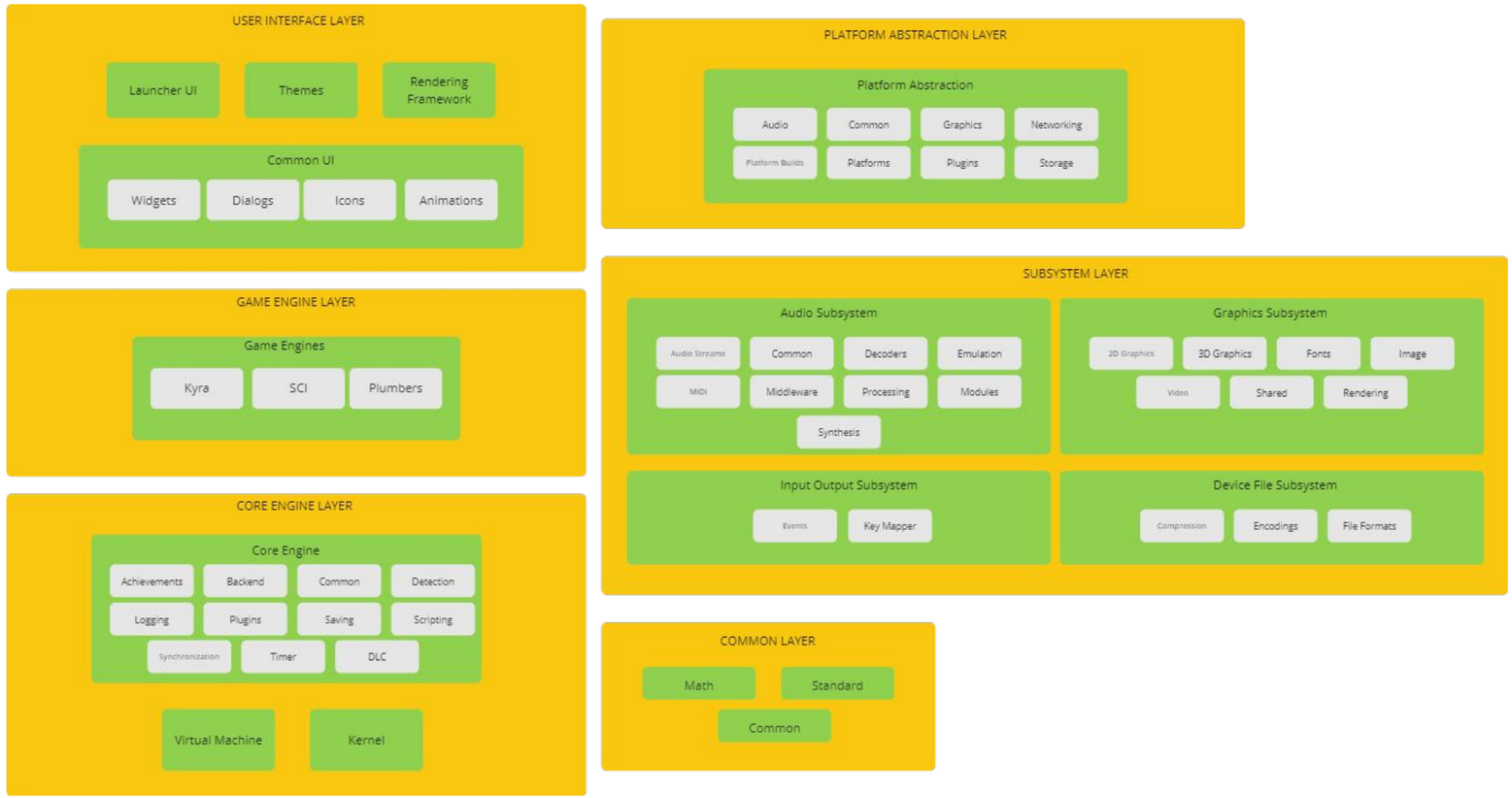


Common Layer

- **Centralized Utilities: Introduced the Common Layer for shared utilities and standardized data structures**
- **Subcomponents: Added a Math Library, C Standard Library, and Common Utility Component for separated and reusable implementations**

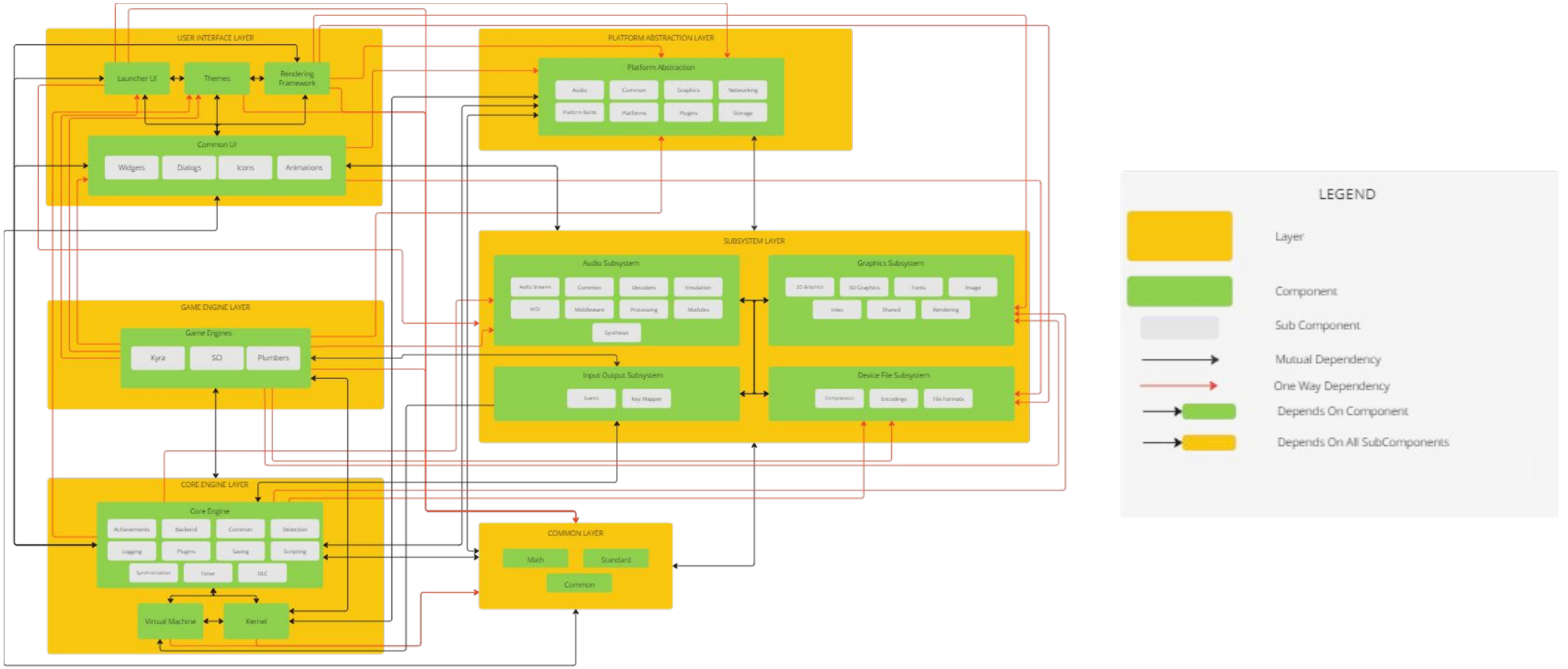
Derivation Process

Final Iteration



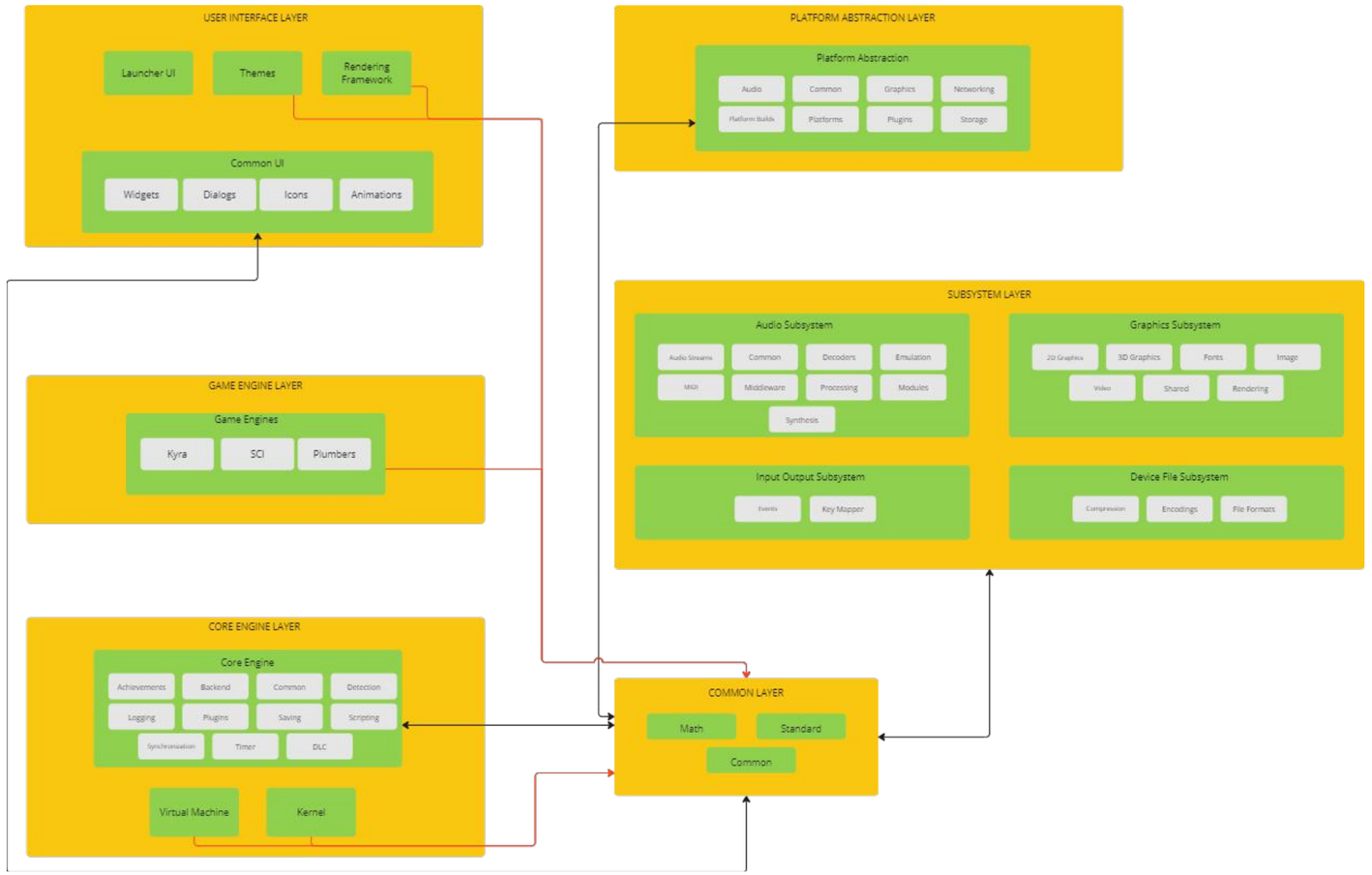
Concrete Architecture

Complete Concrete Architecture



Concrete Architecture

Reflexion Analysis

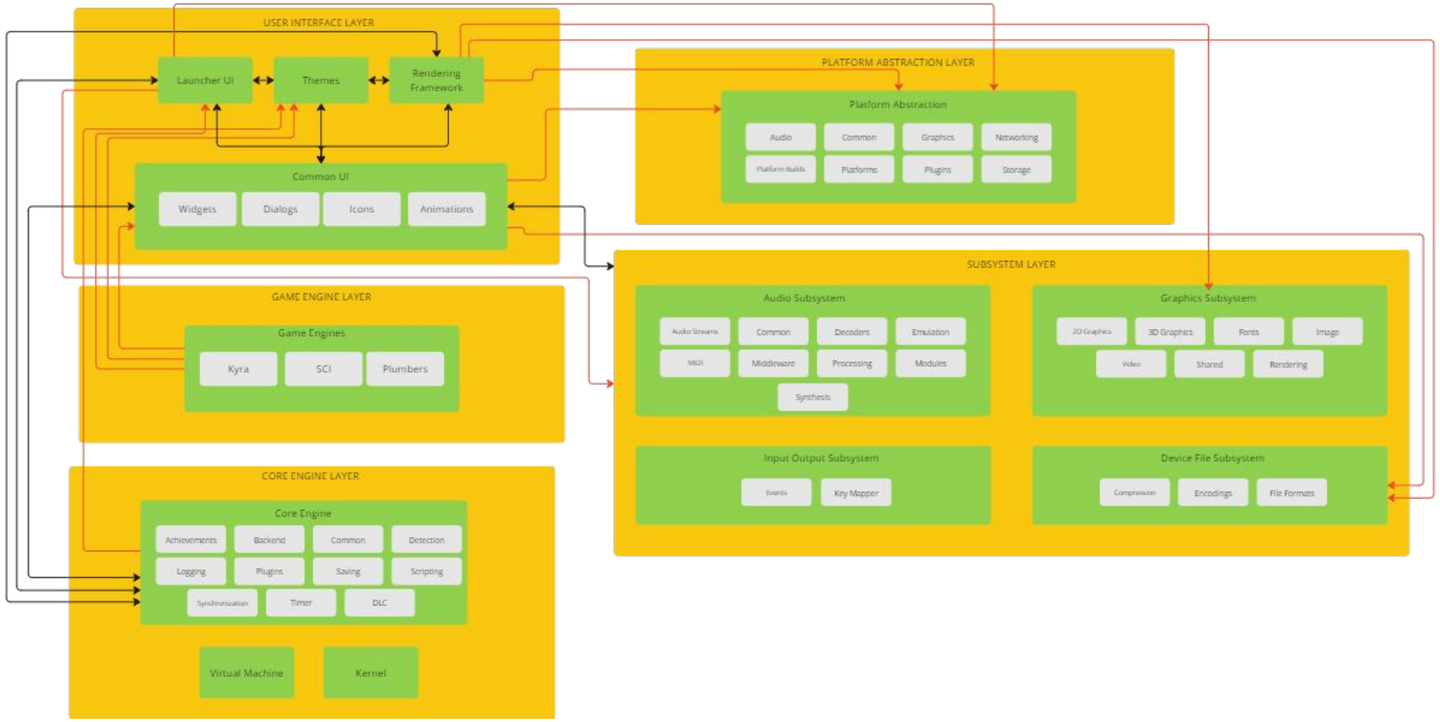


Common Layer

- One-Way Dependencies: Themes, Rendering Framework, Virtual Machine, Kernel and Game Engine
- Mutual Dependencies: Core Engine, Platform Abstraction, Subsystem Layer, Common UI

Concrete Architecture

Reflexion Analysis

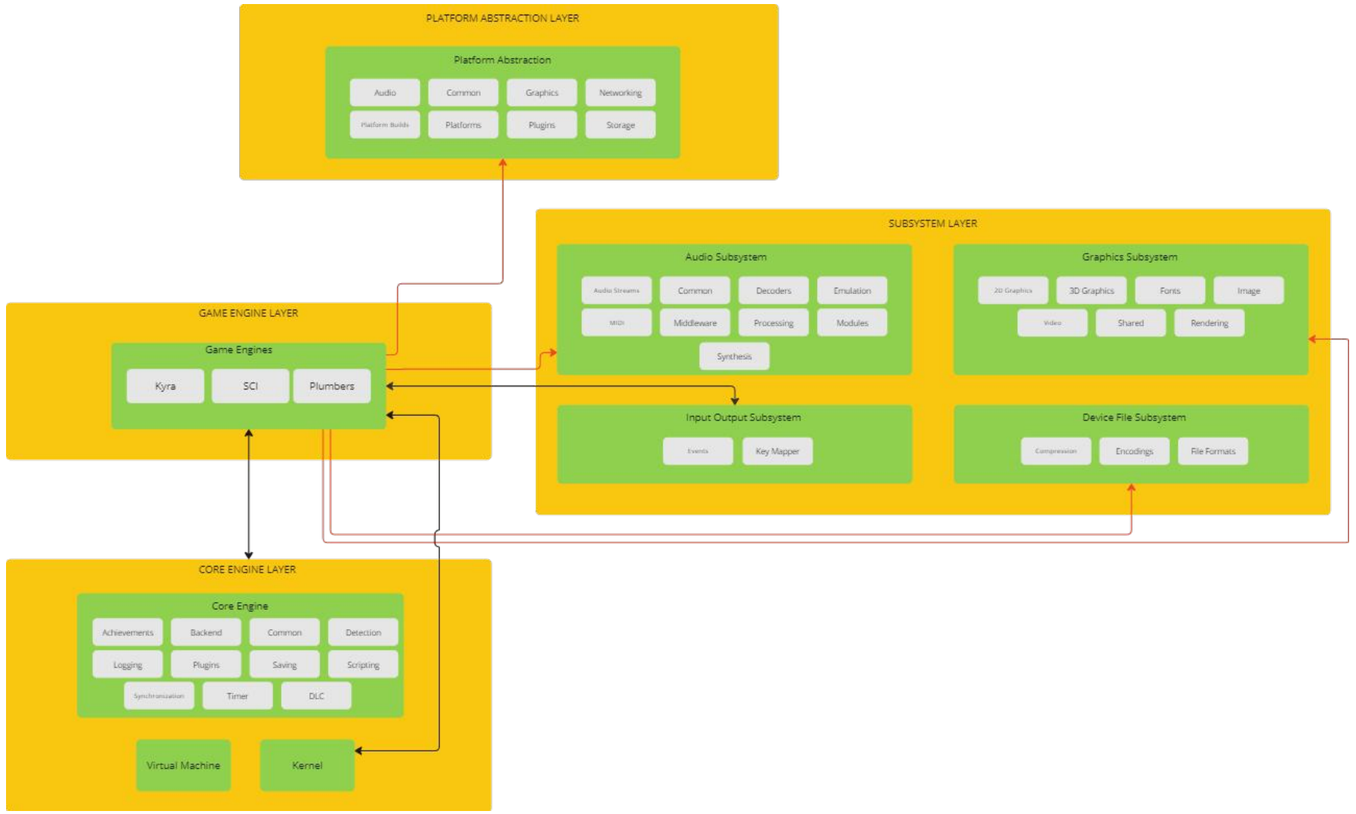


UI Layer

- Mutual Dependencies: Tightly coupled components like Launcher UI ↔️ Common UI, Rendering Framework ↔️ Core Engine
- New One-Way Dependencies: Launcher UI ➡️ PAL and Subsystems (Graphics, Audio, File System)

Concrete Architecture

Reflexion Analysis

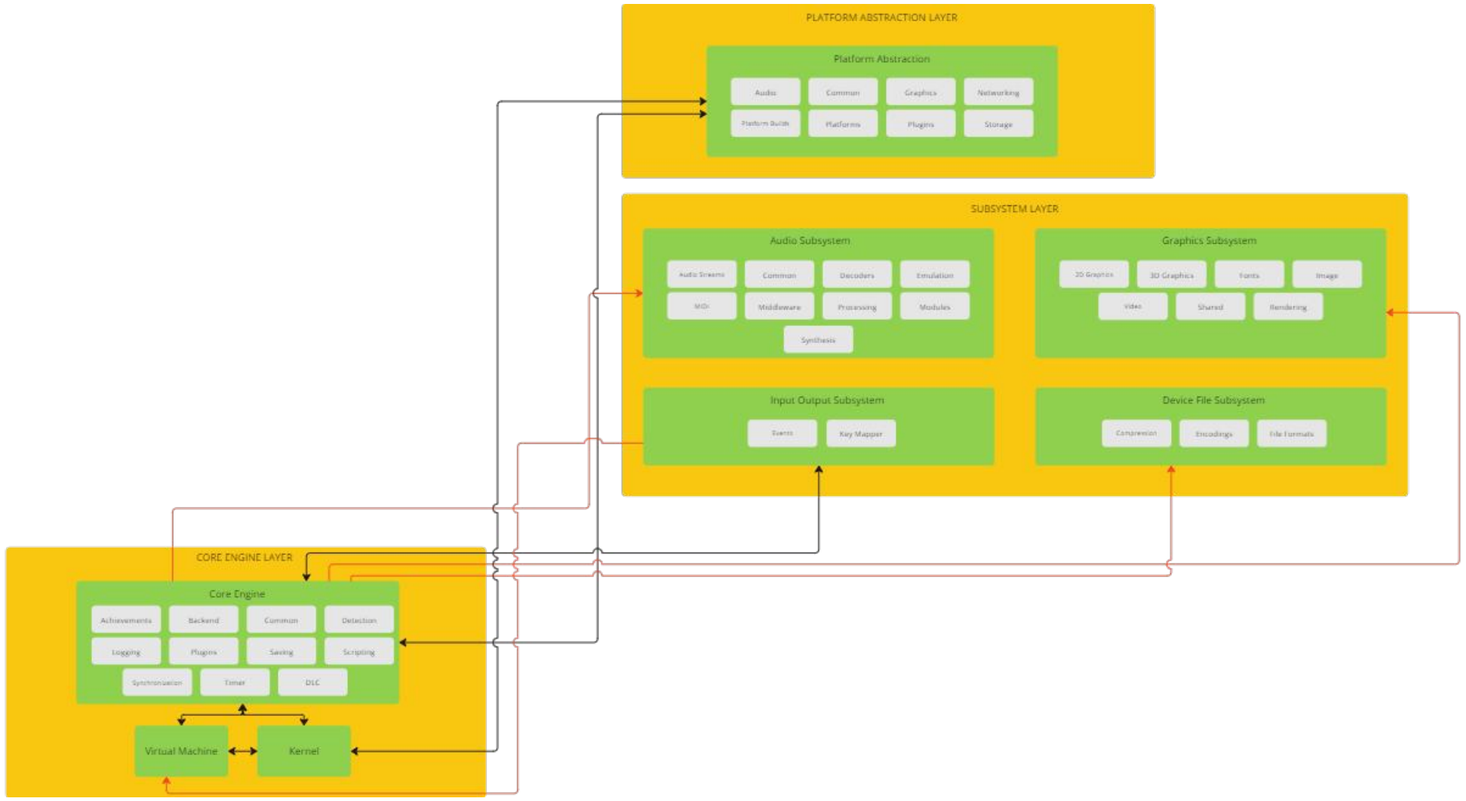


Game Engine Layer

- New One-Way Dependencies: Device Files, Graphics, Audio and PAL
- New Mutual Dependencies: Game Engine ➡️ Input Output
- Missing Dependency: No dependencies between Game Engine and VM

Concrete Architecture

Reflexion Analysis

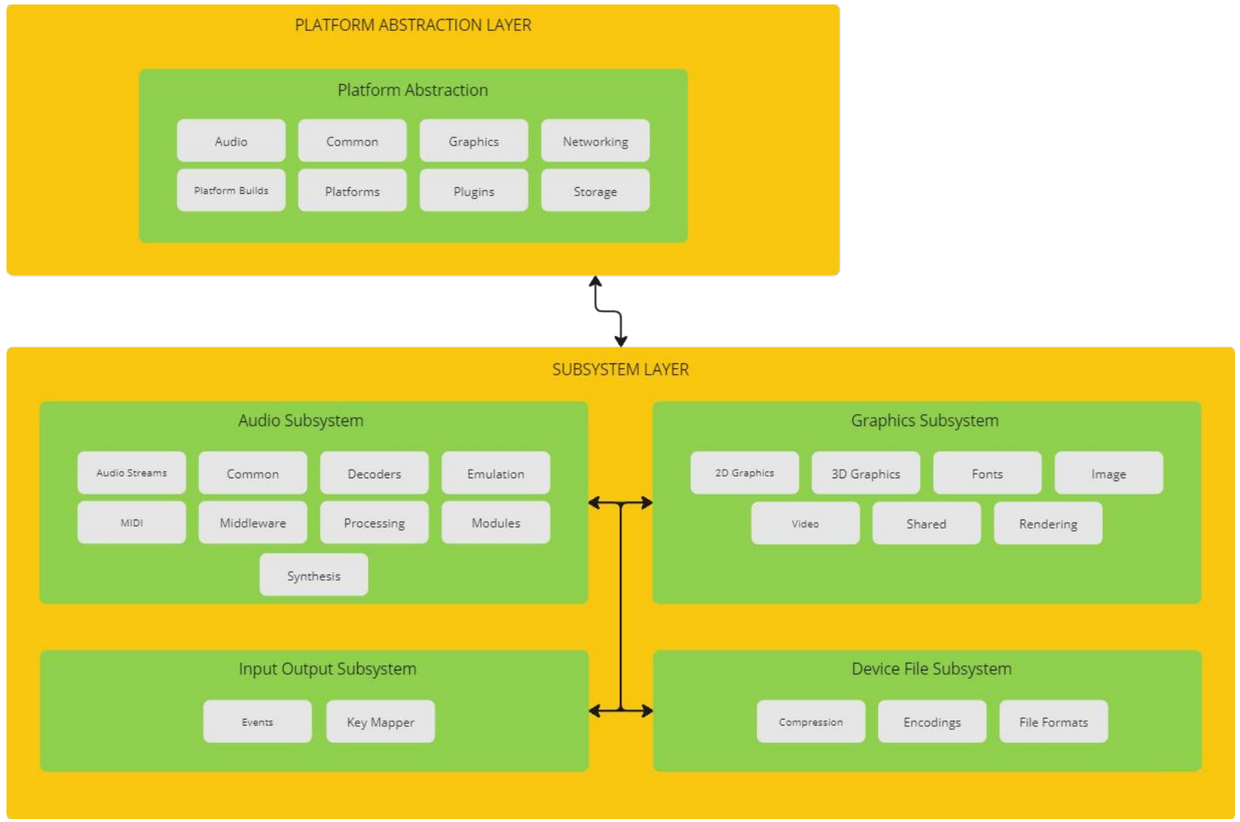


Core Engine Layer

- New One Way Dependencies: IO ➡️ VM, Core Engine ➡️ File System, Graphics and Audio
- New Mutual Dependencies: Core Engine ↔️ PAL, Kernel ↔️ PAL
- Internal Core Engine Components were closely coupled as expected

Concrete Architecture

Reflexion Analysis



Platform Abstraction Layer

- Audio, Graphics, File System and Input Output subcomponents were more coupled then initially expected
- Mutual Dependency between PAL and Subsystem Layer was exactly as expected

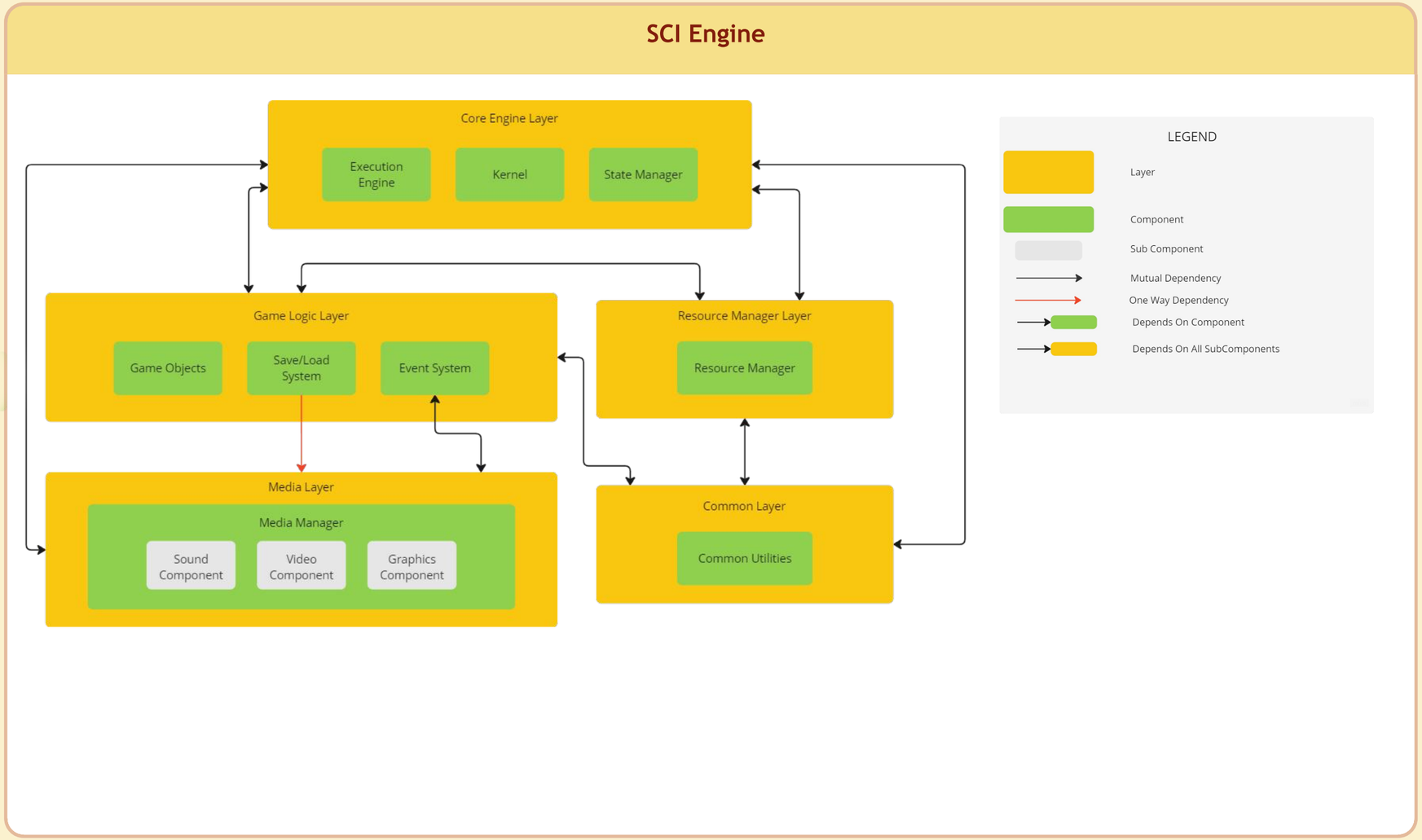
Derivation Process

Conceptual Components and Subsystems Of The SCI Engine

Steps

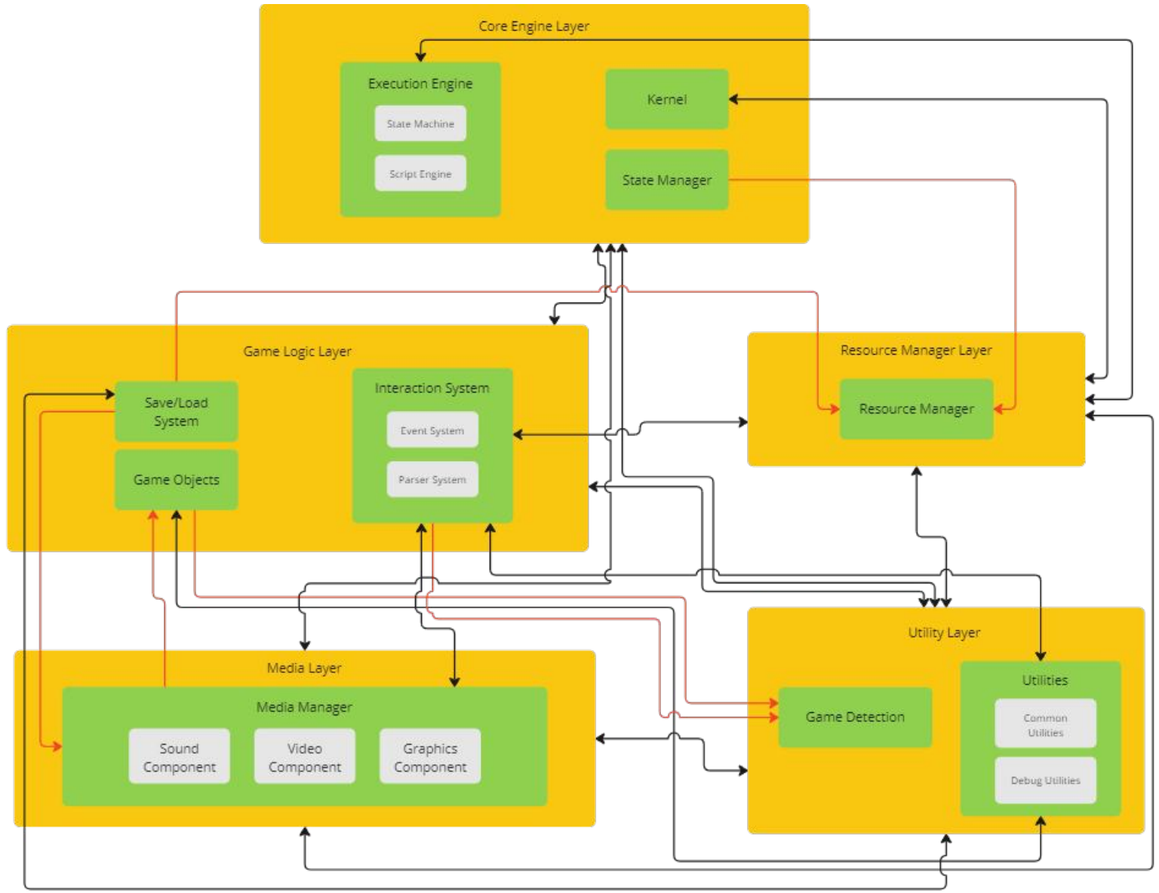
- Took the same approach as in the conceptual architecture from A1
- Derived using online resources and reference architectures of similar emulator
- Structured components using a layered architecture approach

Conceptual Architecture



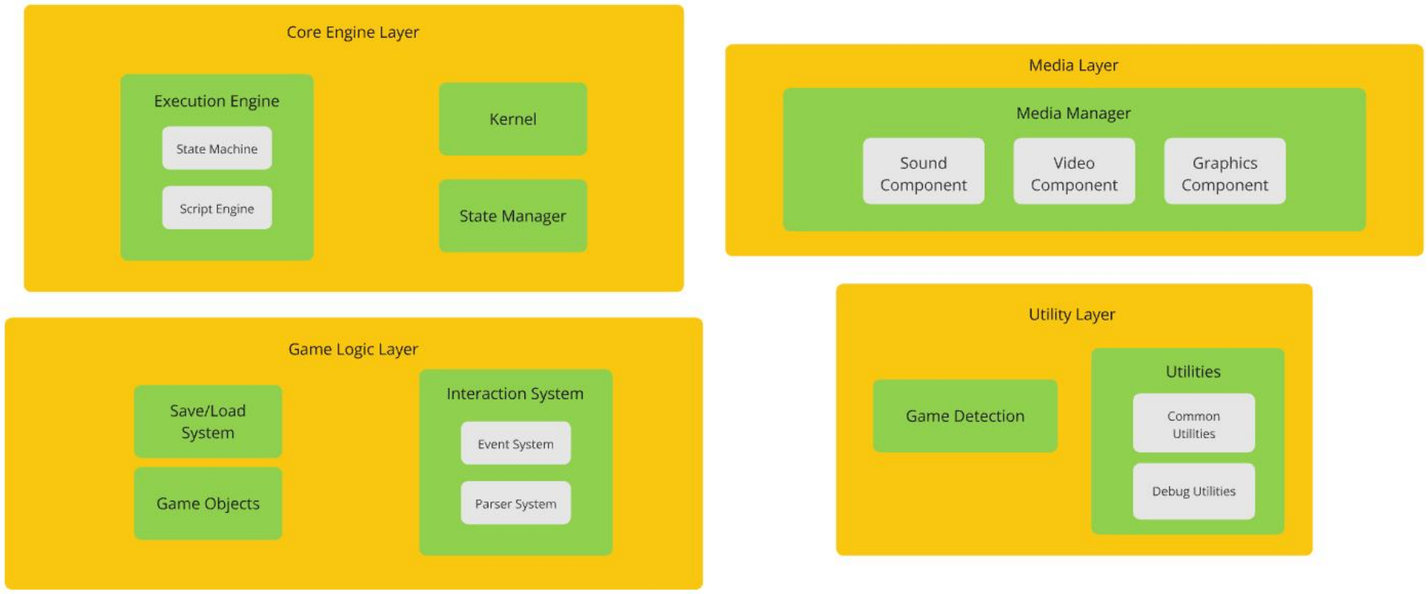
Concrete Architecture

SCI Engine



Concrete Architecture

SCI Engine

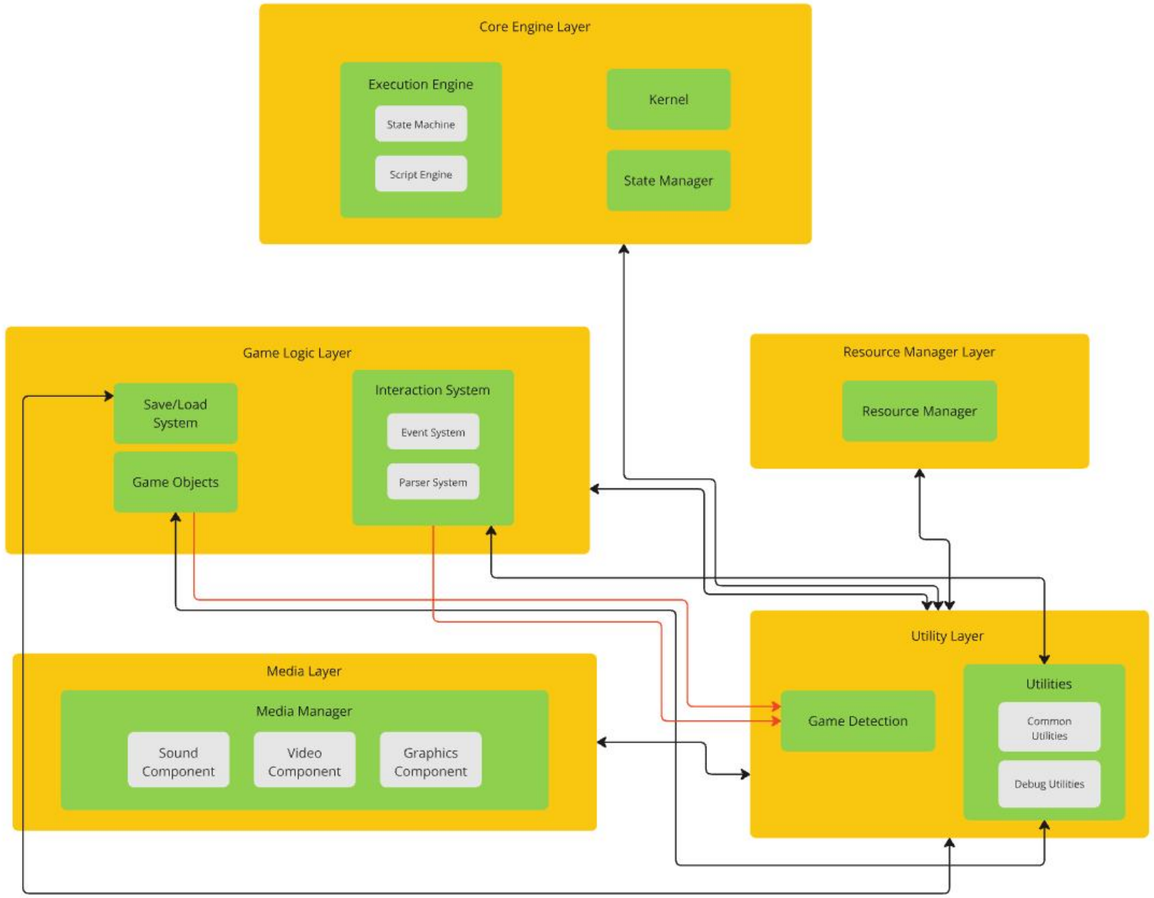


Game Engine Layer

- Most source files mapped well to the conceptual layers
- Added VM and Script Engine subcomponents
- Added Interaction, Event and Parser Systems to the Game Logic layer
- Added Game Detection, Utility Component, Debug and Common utilities to the Utility Layer

Concrete Architecture

SCI Engine Reflexion Analysis

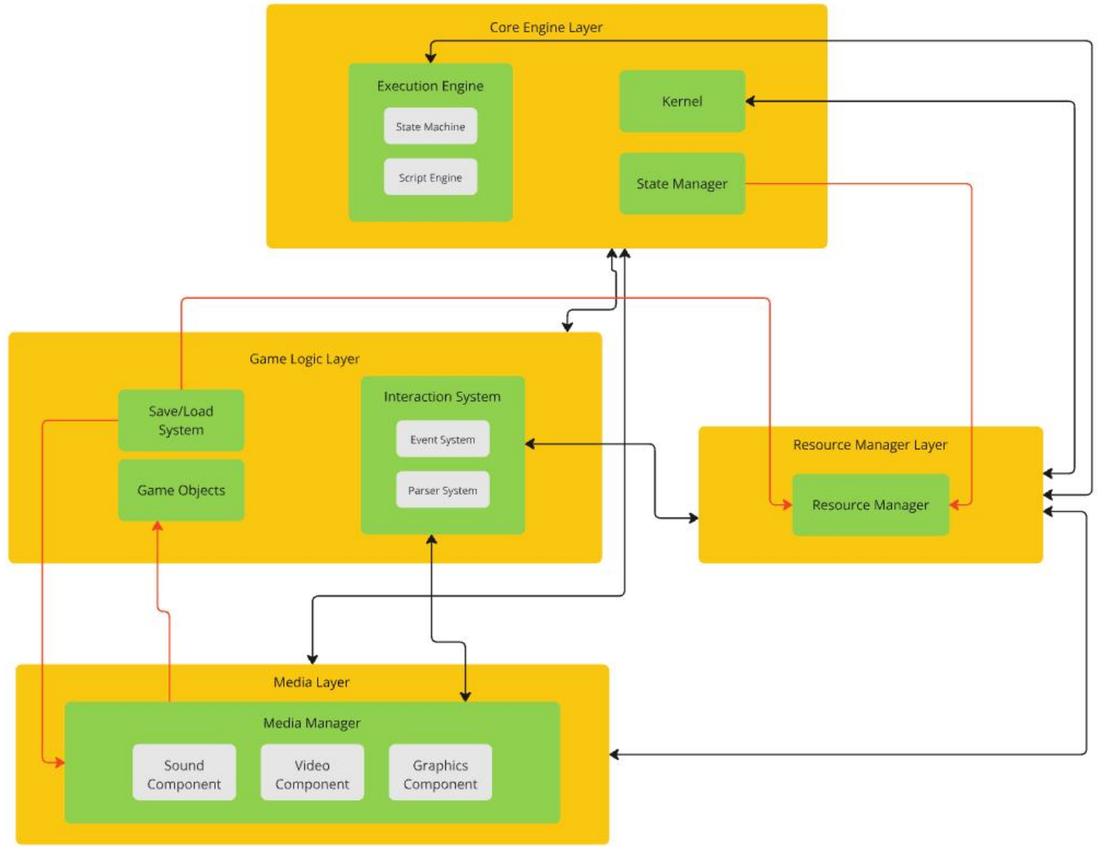


Utility Layer

- New Mutual Dependencies: Utility Layer ↔️ Media Layer
- New One-Way Dependencies: Game Logic Layer ➡️ Game Detection

Concrete Architecture

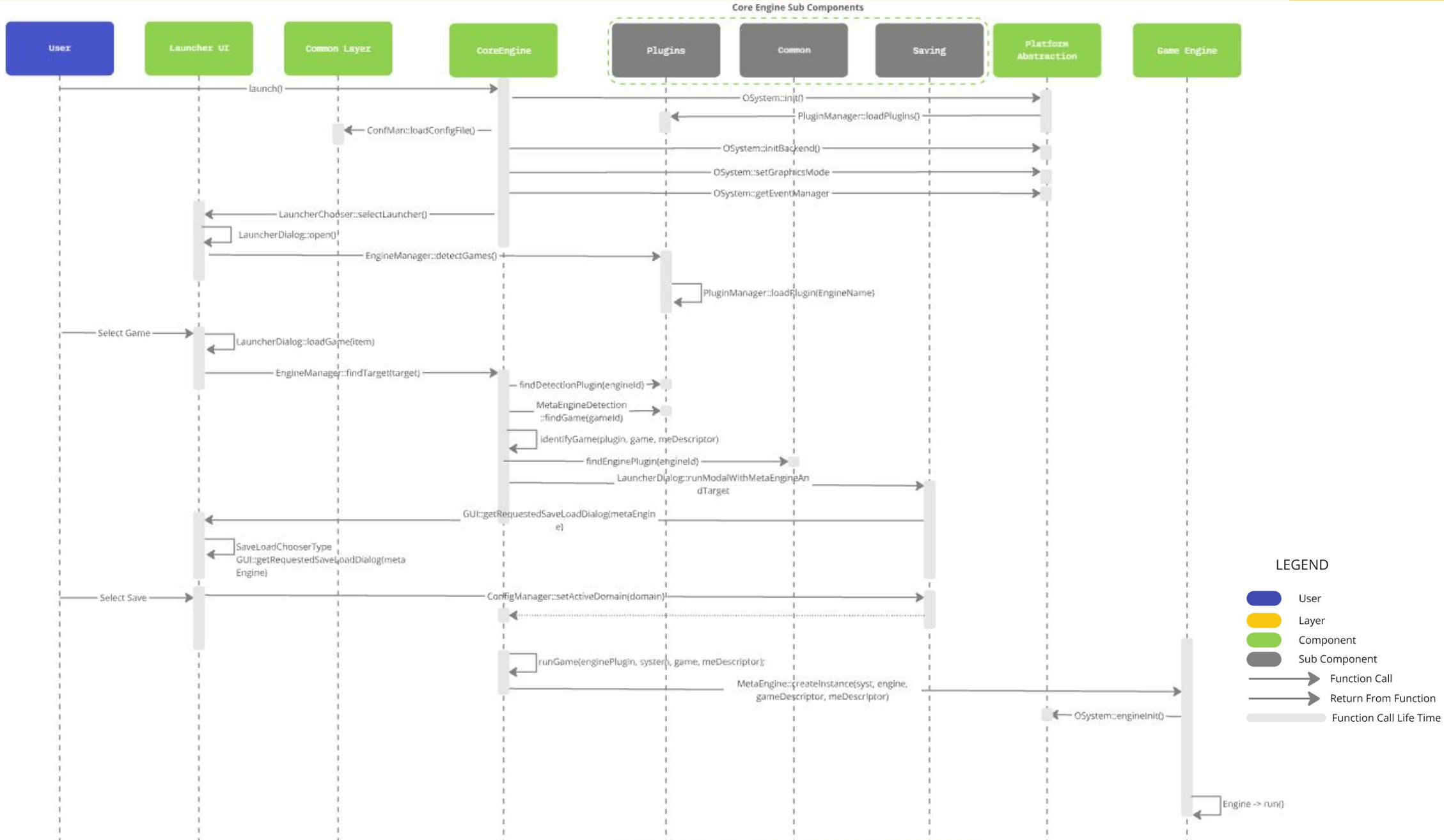
SCI Engine Reflexion Analysis



Remaining Dependencies

- Dependency between Media Layer and Resource management Layer as Media layer must dynamically load resources
- Save/Load depends on Resource Management for game state access
- Interaction System and Resource Management have mutual dependency
- Game Objects lack direct dependency on Resource Management Layer

Game Startup Use Case



Limitations, Lessons Learned, and Conclusion

Limitations and Lessons Learned

- Difficulty understanding functionality and execution flow of source code files
- Large codebase with extensive platform-specific conditional statements
- Complexity in tracing execution paths for sequence diagrams

Summary

- Multiple architectural styles (e.g., layered, object-oriented, interpreter), but many component dependencies create architectural ambiguity in some areas
- Open-source contributions bring new features but introduce risks of technical debt, architectural inconsistencies, and regressions
- High coupling due to platform-specific optimizations and size of codebase