UNIVERSITÀ DEGLI STUDI
DI MILANO

What is a Game Engine?

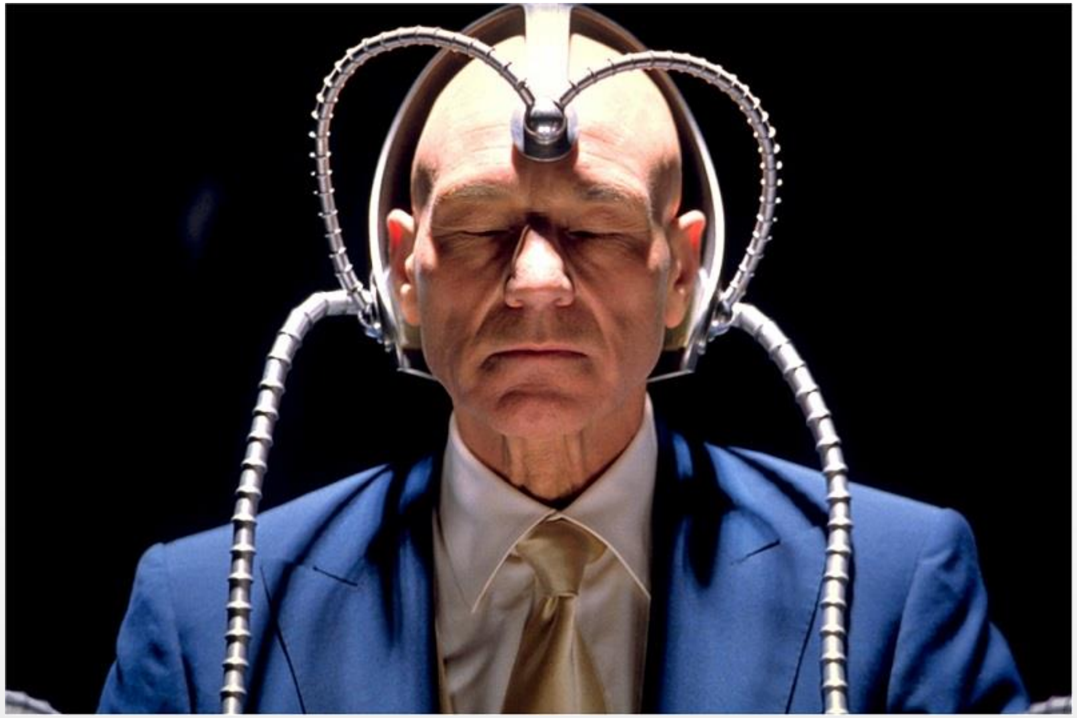
Lesson I01

Dario Maggiorini (dario@di.unimi.it)

Online Game Design - A.A. 2021-2022

Big Question: What is a Game Engine?

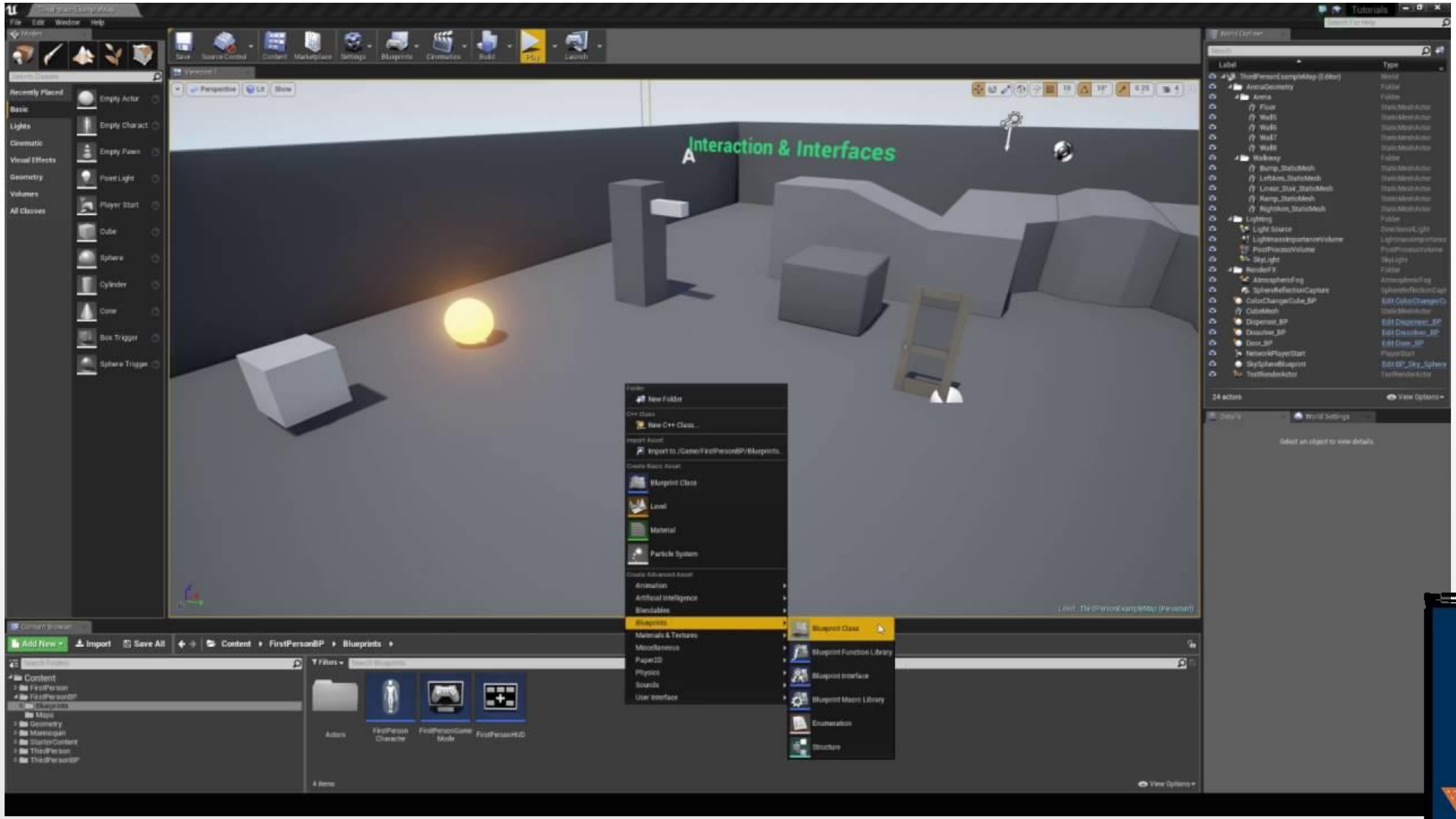
Reading Your Mind ...



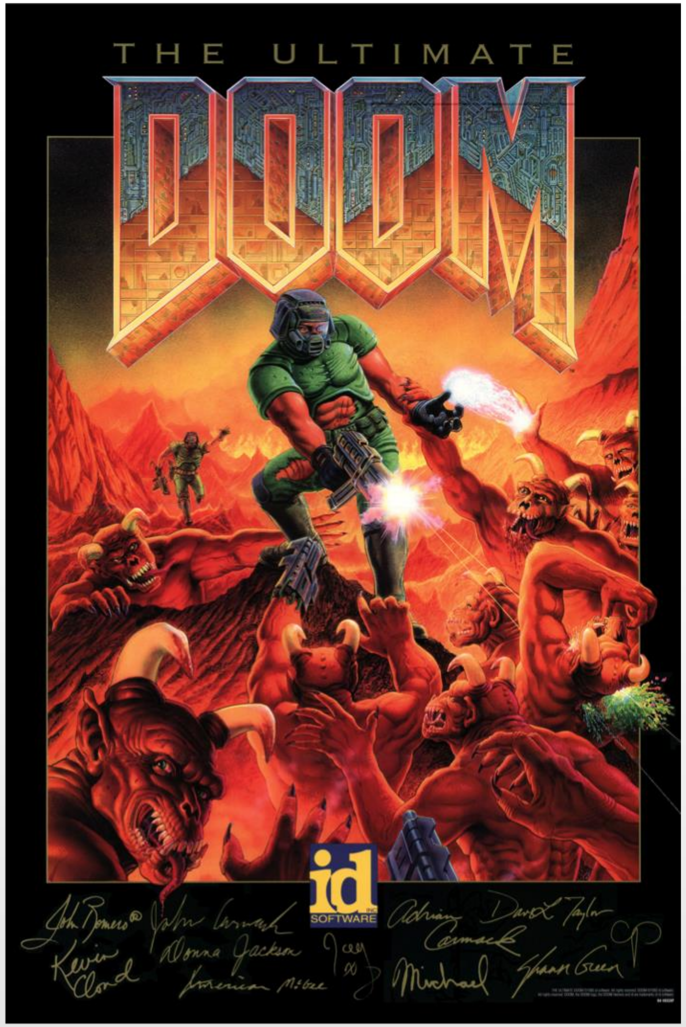
Most probably you though about Unity or Unreal

Wrong Answer!

Not 100% wrong … but a game engine is NOT this!



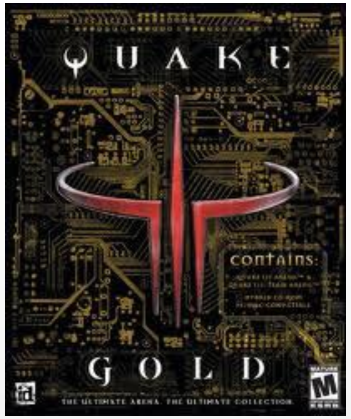
A (Very) Brief History of Game Engines

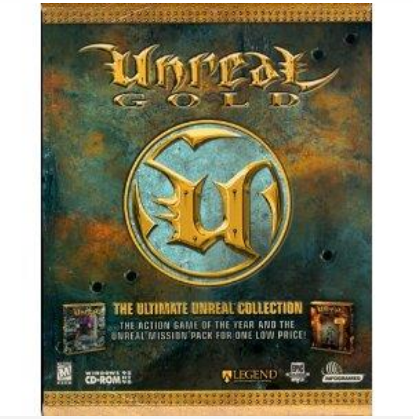


- … dates back to 1994 with DOOM
- Strict separation between core software components
- Strict separation between software and data assets
- Strong code reusability enforced during development
- Yet … The word “engine” was not there

Later on … in 90-Something

- Other games have been designed with with a modular architecture allowing modding and focusing on code reuse




Unreal
GOLD
THE ULTIMATE UNREAL COLLECTION
THE ACTION GAME OF THE YEAR AND THE UNREAL MISSION PACK FOR ONE LOW PRICE!

- Scripting language (such as quake C) started to be part of the distribution
- The game engine is now a standalone product (and a profitable one!)
  - E.g., Quake]I[can be considered as a (paid for) demo of the actual product (the Quake engine)
  - Customer are no longer players but developers!

What is a Game Engine

(Really) Technical description of game:

A real-time interactive agent-based computer simulation

A game engine is a software made to implement such a system

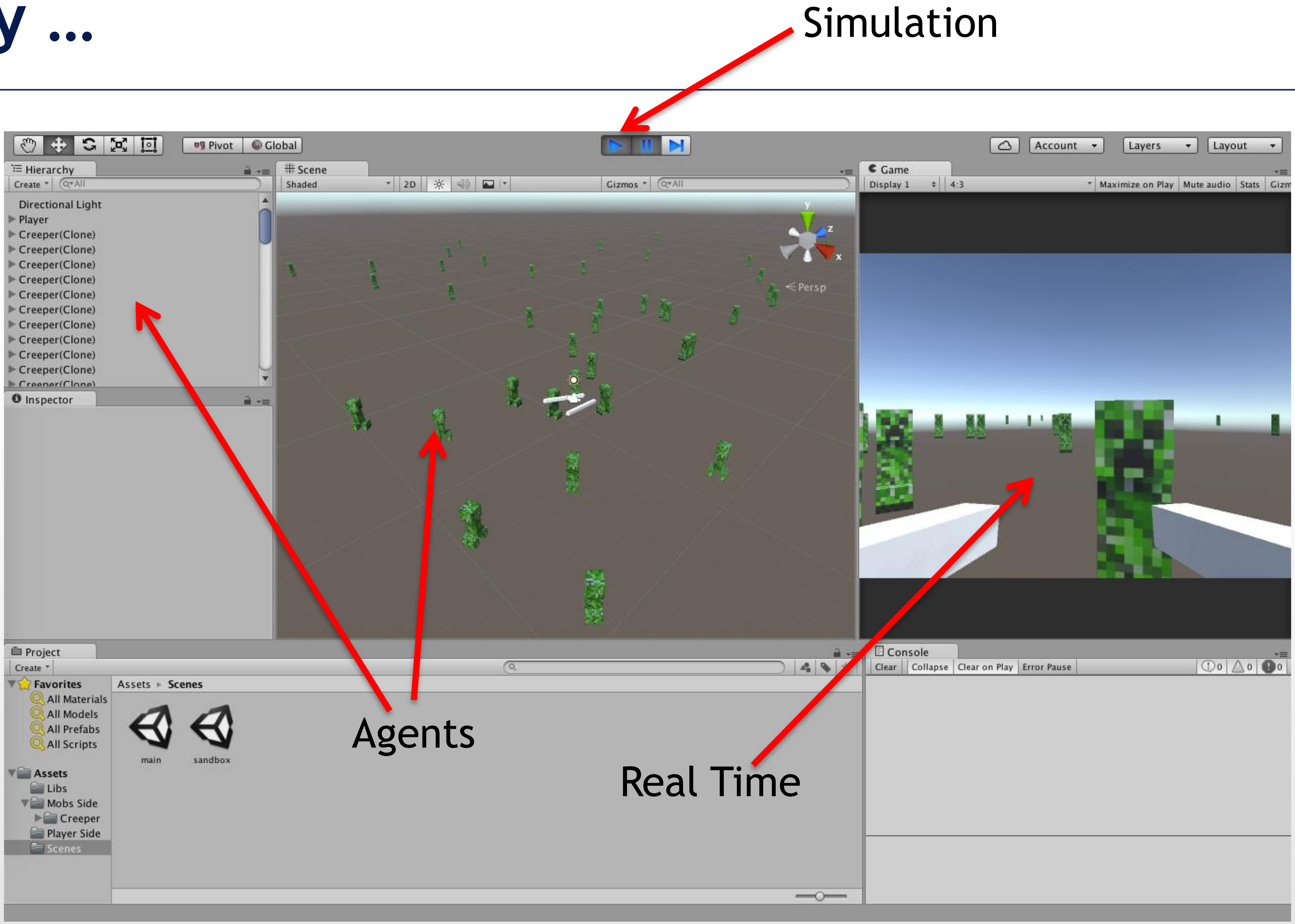
More Into Details

- Real-time (and interactive)
  - Must respond to player input in a timely-bound manner

- Agent-based
  - Independent entities (agents) live and interact with each other within the engine

- Simulation
  - It is capable to describe a model representation of a virtual world
  - It must be a MATHEMATICAL description

In Unity …

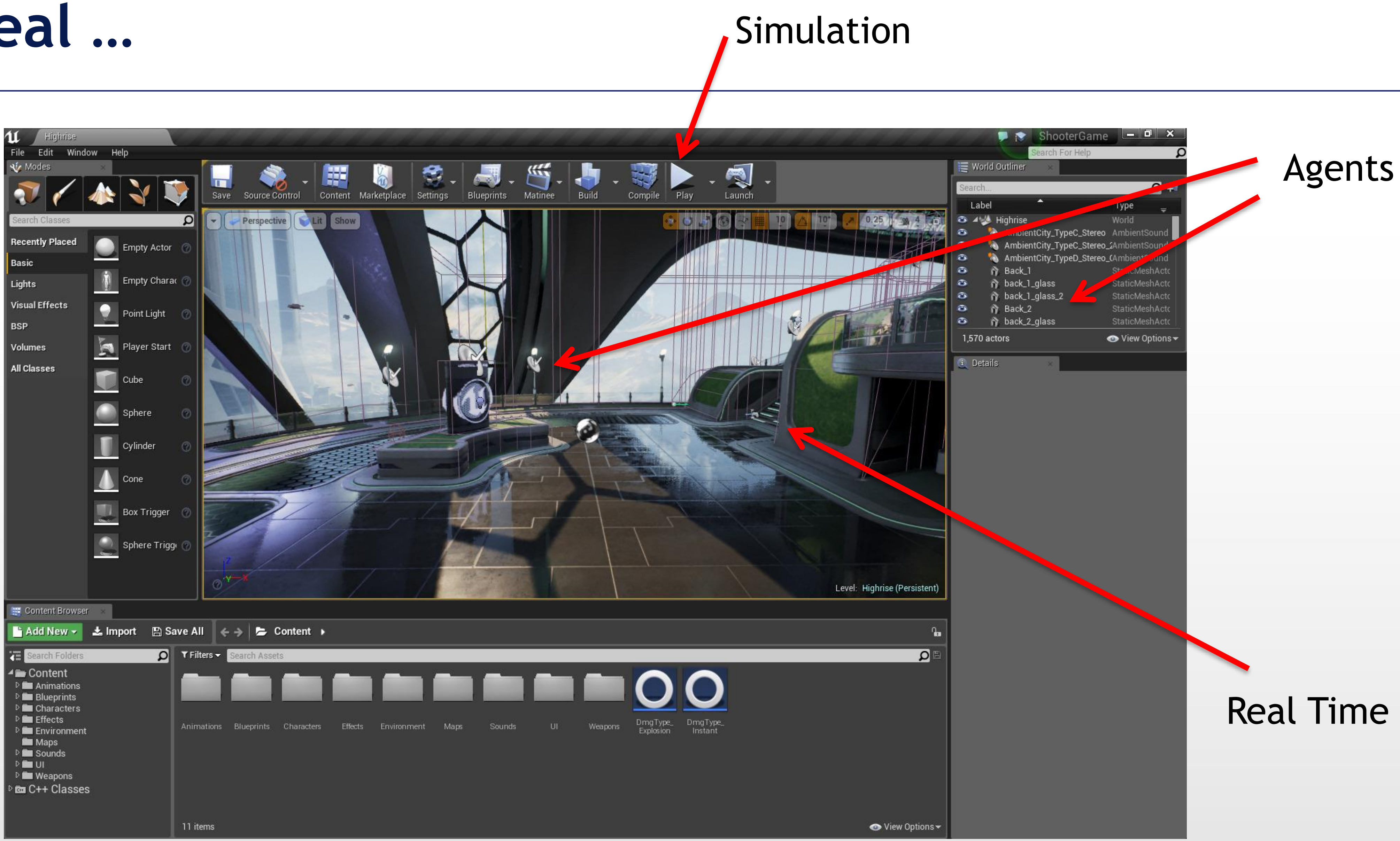


Simulation

Agents

Real Time

In Unreal …

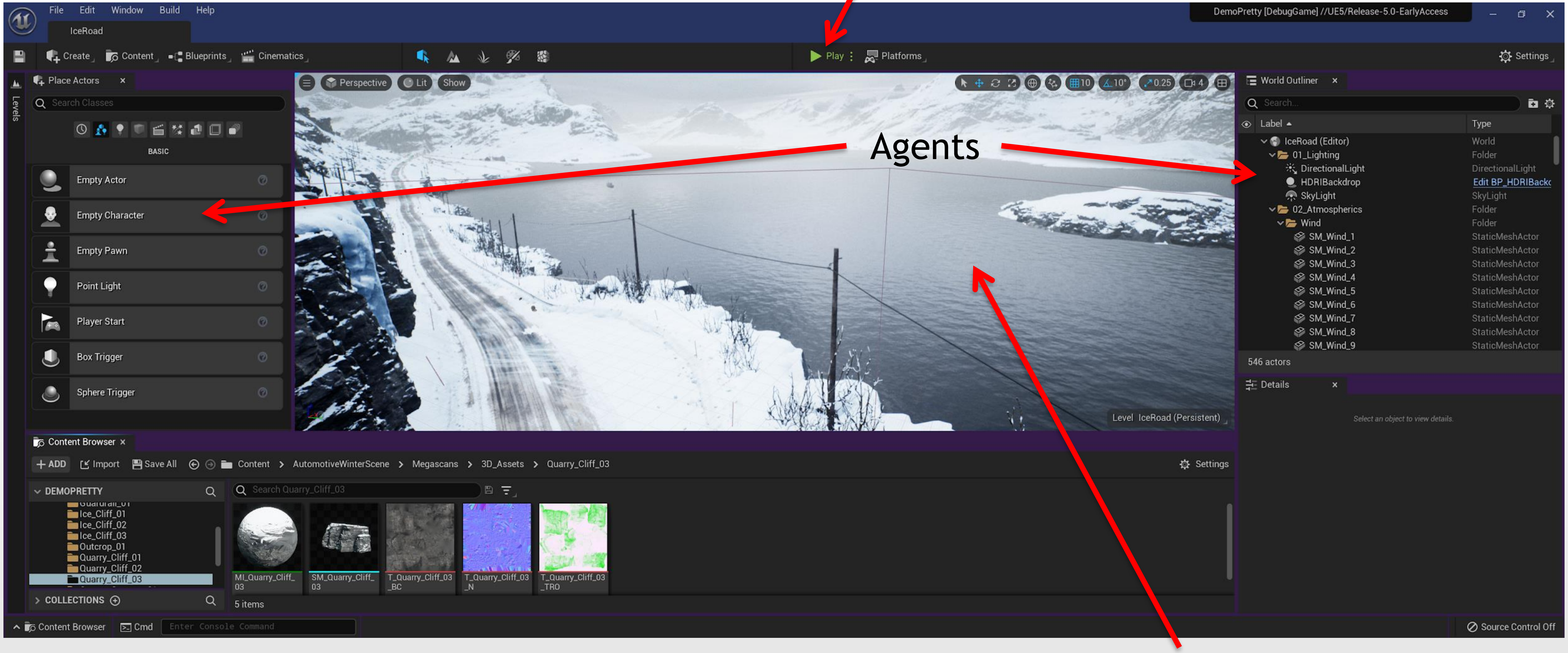


Simulation

Agents

Real Time

In Unreal 5 …



Simulation

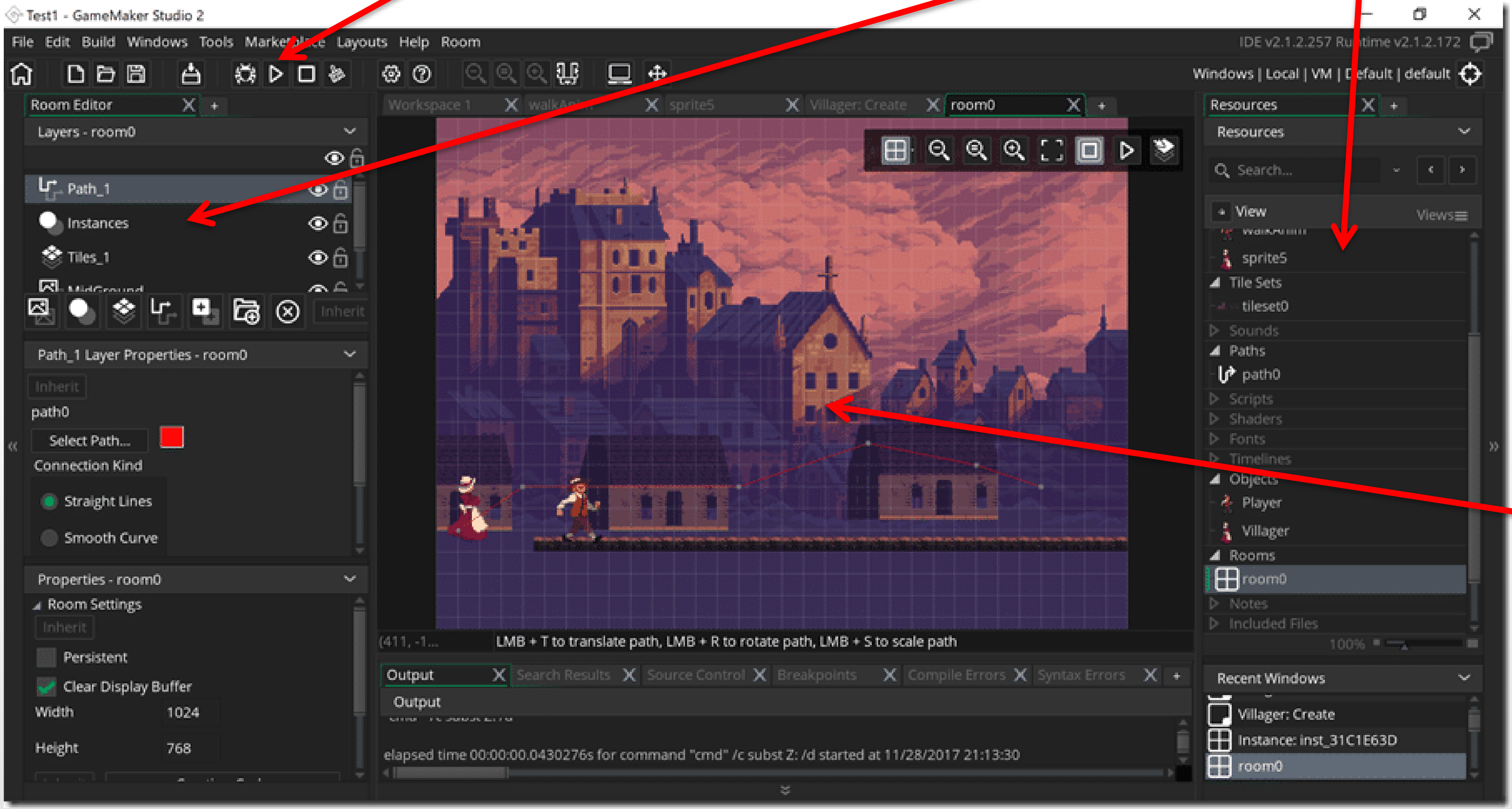
Agents

Real Time

In Game Maker …

Simulation

Agents



Path_1 Layer Properties - room0

Inherit

path0

Select Path... 🟥

Connection Kind

- ● Straight Lines
- ○ Smooth Curve

Properties - room0

Room Settings

Inherit

- □ Persistent
- ☑ Clear Display Buffer

Width	1024
Height	768

```
(411, -1...   LMB + T to translate path, LMB + R to rotate path, LMB + S to scale path
```

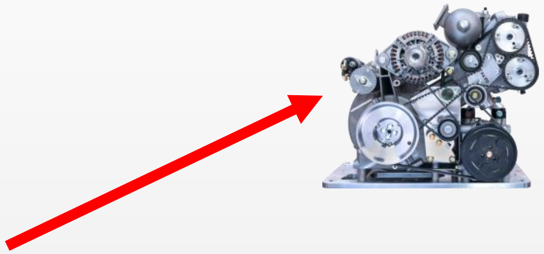
Output

```
elapsed time 00:00:00.0430276s for command "cmd" /c subst Z: /d started at 11/28/2017 21:13:30
```

Real Time

Building a Game Engine

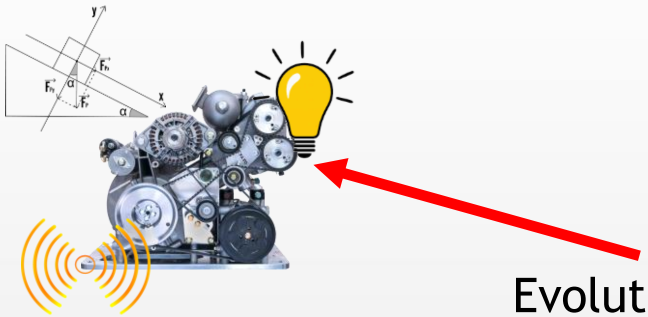
What is making your game “run” is a very small piece of code in charge to manage the basic simulation



Core
Resource management and coordination

Building a Game Engine

The core does not know how to make the system evolve.
It just applies “evolution rules” provided with the game



Evolution rules

Core is focused on performances.
It must be able to apply any kind of rule

Building a Game Engine

Core and basic evolution rules (e.g., physics and lighting) are bundled and hidden from the user. Like in a black box

Runtime

This black box (no one knows its content for commercial engines) is referenced as “the Runtime component”

Building a Game Engine

Easy to use interface

There is no need to know the content of the black box to create a game.

All we need to know is how to USE it! Possibly, by means of a convenient user interface

Runtime and Tool Set

- A game engine is made of two parts:
  - Tool Set
    - To compile software to work within the game engine
    - To help you describe rules
    - To manage assets
    - To create content

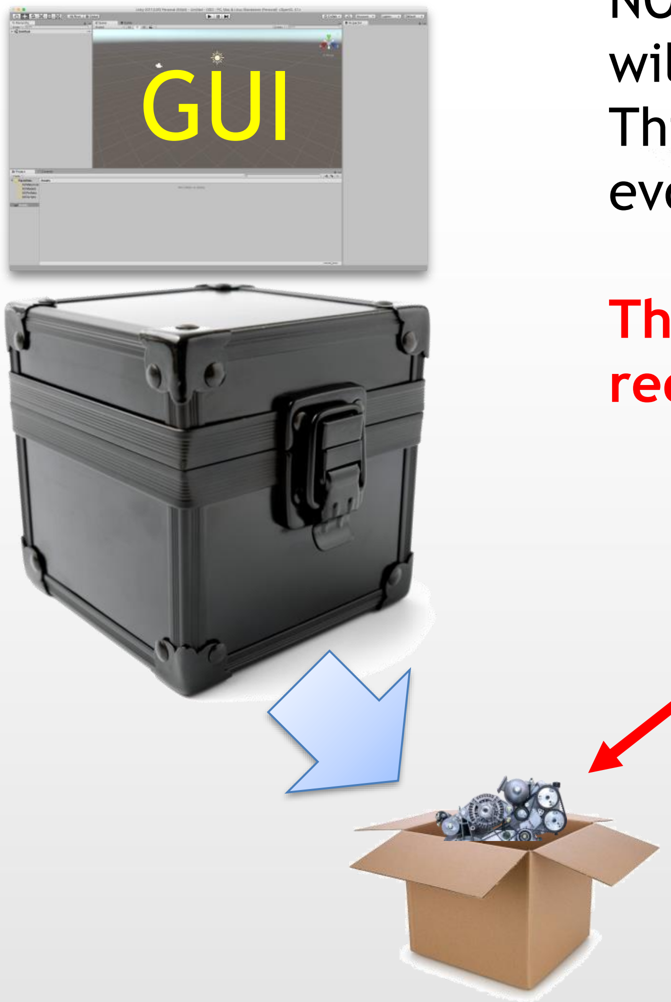
The GUI is usually the front-end to the toolset

And THIS was your (wrong) answer to the initial question

  - Runtime
    - A library/middleware/sandbox/virtual machine
    - This will run your rules on your assets
    - MUST be distributed with the game

Building a Game Engine

You use the GUI to put “stuff” (assets) inside the black box and then ask it to create an executable file



NOTE: a (large) piece of the core will be inside each executable. This is because the game must evolve also outside the black box

This is why we need redistributable licenses

Your game

Focus on the Black Box

- All we need to understand is how the black box is working
  - As a matter of fact, it is not required for a game engine to have a GUI (e.g., Source from Valve)

... Have been made with Source

A Black Box for Rules and Assets

- A Game engine is a container for RULES
  - You **explain** how the world is evolving
  - You **do not** create the code to make it evolve

- The black box will follow the rules and apply them on the assets you provided
  - At every step, these rules will change the box status and its assets, making your world magically evolve

What is an Asset?

- An asset is whatever you may think about to show, listen, or feel while playing
  - Texture
  - 3D meshes
  - Material definition
  - Particles
  - Visual effects (shaders)
  - Music
  - *Scraps of code* (?)

Asset

What is a Rule?

- A rule is the definition of a behavior you attach to an asset in order to define:

  1. How to interact with the user
     e.g., reacting to user controls
  2. How to interact with the environment
     e.g., falling and casting shadows
  3. How to interact with other assets
     e.g., collisions

- Creating a believable NPC means creating the "right" rules based on the surrounding context

Asset with rules

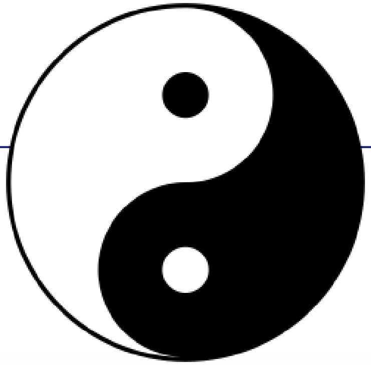
Script and Rules

- Of course, the easiest way (for a computer scientist) to describe a rule is by means of a scrap of code

… but

- A script is technically an asset (!)
- A script turns into a rule when:
  1. Is compiled
  2. Is run inside another asset

There are Two Kinds of Rules

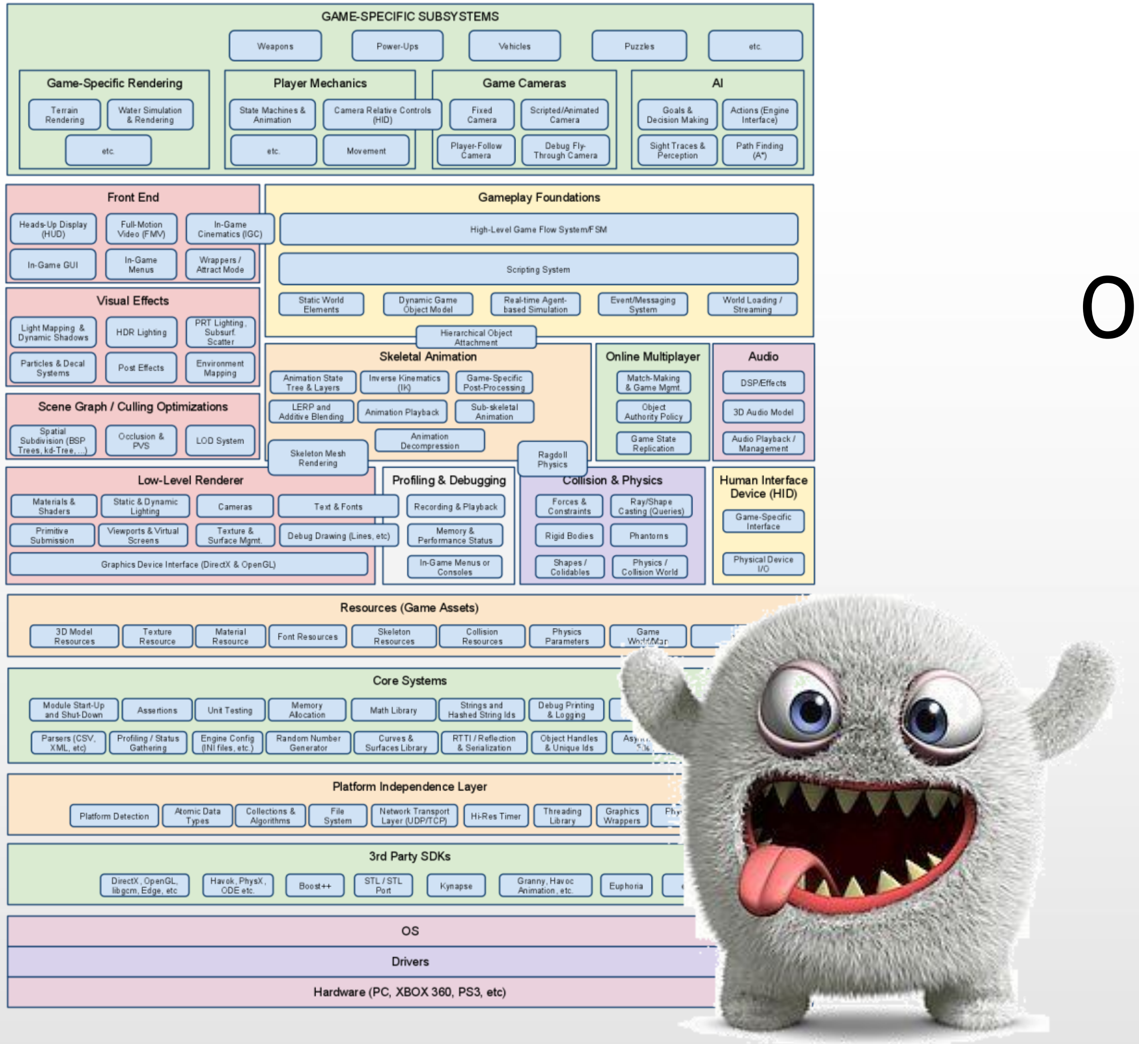


- Built-in
  - Wired in the black box for everyday swiss-army knife use

- Provided by the user
  - This is what is making you game truly unique
  - You will call then “gameplay programming”

- There is NO ACTUAL DIFFERENCE between the two
  - You can change built-in rules if you wish
    (e.g., switch to havok or bullet physic engines)

Runtime Architecture



GAME-SPECIFIC SUBSYSTEMS

Weapons | Power-Ups | Vehicles | Puzzles | etc.

Game-Specific Rendering: Terrain Rendering; Water Simulation & Rendering; etc.

Player Mechanics: State Machines & Animation; Camera Relative Controls (HID); etc.; Movement

Game Cameras: Fixed Camera; Scripted/Animated Camera; Player-Follow Camera; Debug Fly-Through Camera

AI: Goals & Decision Making; Actions (Engine Interface); Sight Traces & Perception; Path Finding (A*)

Front End: Heads-Up Display (HUD); Full-Motion Video (FMV); In-Game Cinematics (IGC); In-Game GUI; In-Game Menus; Wrappers / Attract Mode

Gameplay Foundations: High-Level Game Flow System/FSM; Scripting System; Static World Elements; Dynamic Game Object Model; Real-time Agent-based Simulation; Event/Messaging System; World Loading / Streaming; Hierarchical Object Attachment

Visual Effects: Light Mapping & Dynamic Shadows; HDR Lighting; PRT Lighting, Subsurf. Scatter; Particles & Decal Systems; Post Effects; Environment Mapping

Scene Graph / Culling Optimizations: Spatial Subdivision (BSP Trees, kd-Tree, ...); Occlusion & PVS; LOD System

Skeletal Animation: Animation State Tree & Layers; Inverse Kinematics (IK); Game-Specific Post-Processing; LERP and Additive Blending; Animation Playback; Sub-skeletal Animation; Animation Decompression; Skeleton Mesh Rendering; Ragdoll Physics

Online Multiplayer: Match-Making & Game Mgmt.; Object Authority Policy; Game State Replication

Audio: DSP/Effects; 3D Audio Model; Audio Playback / Management

Low-Level Renderer: Materials & Shaders; Static & Dynamic Lighting; Cameras; Text & Fonts; Primitive Submission; Viewports & Virtual Screens; Texture & Surface Mgmt.; Debug Drawing (Lines, etc); Graphics Device Interface (DirectX & OpenGL)

Profiling & Debugging: Recording & Playback; Memory & Performance Status; In-Game Menus or Consoles

Collision & Physics: Forces & Constraints; Ray/Shape Casting (Queries); Rigid Bodies; Phantoms; Shapes / Colidables; Physics / Collision World

Human Interface Device (HID): Game-Specific Interface; Physical Device I/O

Resources (Game Assets): 3D Model Resources; Texture Resource; Material Resource; Font Resources; Skeleton Resources; Collision Resources; Physics Parameters; Game World/Map

Core Systems: Module Start-Up and Shut-Down; Assertions; Unit Testing; Memory Allocation; Math Library; Strings and Hashed String Ids; Debug Printing & Logging; Parsers (CSV, XML, etc); Profiling / Status Gathering; Engine Config (INI files, etc.); Random Number Generator; Curves & Surfaces Library; RTTI / Reflection & Serialization; Object Handles & Unique Ids; Asy

Platform Independence Layer: Platform Detection; Atomic Data Types; Collections & Algorithms; File System; Network Transport Layer (UDP/TCP); Hi-Res Timer; Threading Library; Graphics Wrappers; Phy

3rd Party SDKs: DirectX, OpenGL, libgcm, Edge, etc; Havok, PhysX, ODE etc.; Boost++; STL / STL Port; Kynapse; Granny, Havoc Animation, etc.; Euphoria

OS

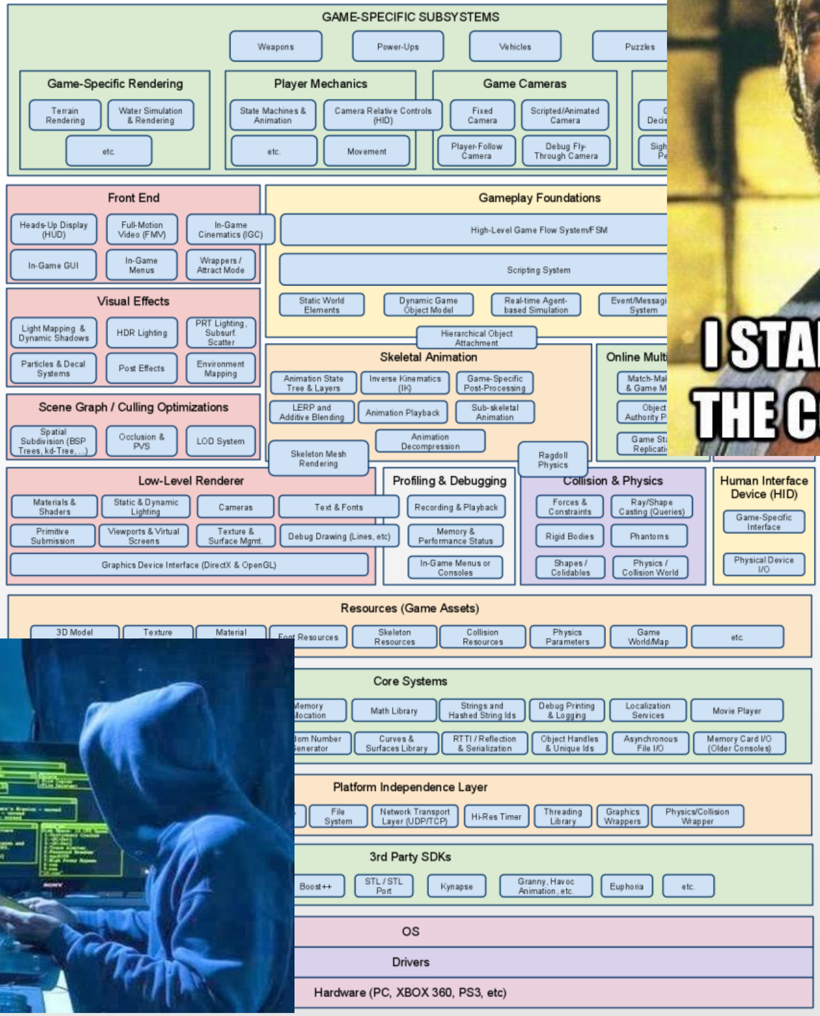
Drivers

Hardware (PC, XBOX 360, PS3, etc)

Ouch!

Runtime Architecture

Specialized developers are required for engine runtime development



GAME-SPECIFIC SUBSYSTEMS

Weapons

Power-Ups

Vehicles

Puzzles

Game-Specific Rendering

Terrain Rendering

Water Simulation & Rendering

etc.

Player Mechanics

State Machines & Animation

Camera Relative Controls (HID)

etc.

Movement

Game Cameras

Fixed Camera

Scripted/Animated Camera

Player-Follow Camera

Debug Fly-Through Camera

Deci

Sigh Pe

Front End

Heads-Up Display (HUD)

Full-Motion Video (FMV)

In-Game Cinematics (IGC)

In-Game GUI

In-Game Menus

Wrappers / Attract Mode

Visual Effects

Light Mapping & Dynamic Shadows

HDR Lighting

PRT Lighting, Subsurf. Scatter

Particles & Decal Systems

Post Effects

Environment Mapping

Scene Graph / Culling Optimizations

Spatial Subdivision (BSP Trees, kd-Tree, ...)

Occlusion & PVS

LOD System

Gameplay Foundations

High-Level Game Flow System/FSM

Scripting System

Static World Elements

Dynamic Game Object Model

Real-time Agent-based Simulation

Event/Messagi System

Hierarchical Object Attachment

Skeletal Animation

Animation State Tree & Layers

Inverse Kinematics (IK)

Game-Specific Post-Processing

LERP and Additive Blending

Animation Playback

Sub-skeletal Animation

Animation Decompression

Skeleton Mesh Rendering

Ragdoll Physics

Online Multi

Match-Mal & Game M

Object Authority P

Game Sta Replicati

Low-Level Renderer

Materials & Shaders

Static & Dynamic Lighting

Cameras

Text & Fonts

Primitive Submission

Viewports & Virtual Screens

Texture & Surface Mgmt.

Debug Drawing (Lines, etc)

Graphics Device Interface (DirectX & OpenGL)

Profiling & Debugging

Recording & Playback

Memory & Performance Status

In-Game Menus or Consoles

Collision & Physics

Forces & Constraints

Ray/Shape Casting (Queries)

Rigid Bodies

Phantoms

Shapes / Colidables

Physics / Collision World

Human Interface Device (HID)

Game-Specific Interface

Physical Device I/O

Resources (Game Assets)

3D Model

Texture

Material

Font Resources

Skeleton Resources

Collision Resources

Physics Parameters

Game World/Map

etc.

Core Systems

lemory llocation

Math Library

Strings and Hashed String Ids

Debug Printing & Logging

Localization Services

Movie Player

dom Number enerator

Curves & Surfaces Library

RTTI / Reflection & Serialization

Object Handles & Unique Ids

Asynchronous File I/O

Memory Card I/O (Older Consoles)

Platform Independence Layer

File System

Network Transport Layer (UDP/TCP)

Hi-Res Timer

Threading Library

Graphics Wrappers

Physics/Collision Wrapper

3rd Party SDKs

Boost++

STL / STL Port

Kynapse

Granny, Havoc Animation, etc.

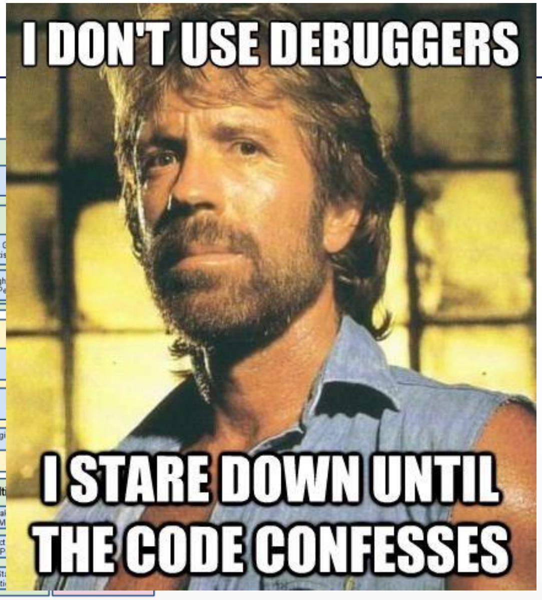
Euphoria

etc.

OS

Drivers

Hardware (PC, XBOX 360, PS3, etc)



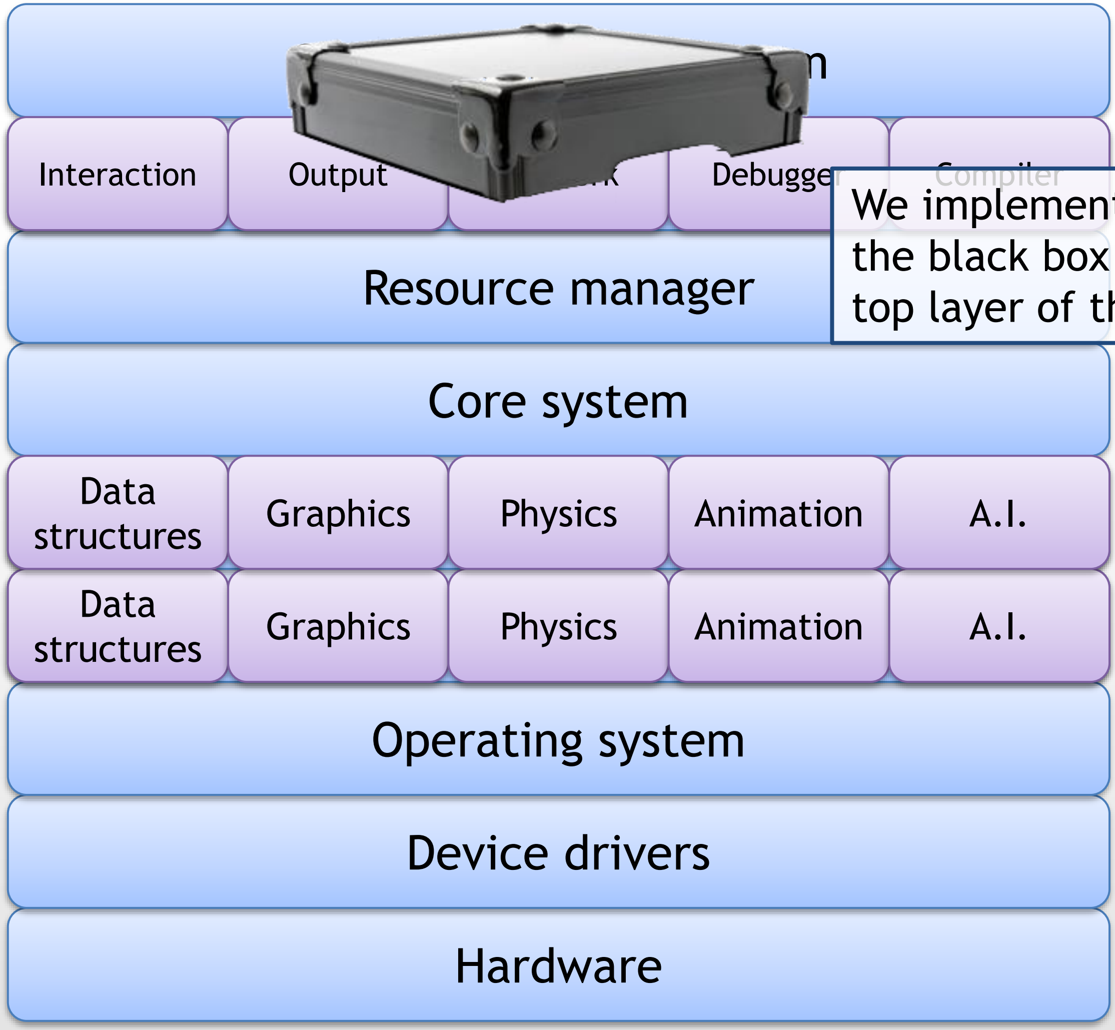
Very difficult to debug

You do not need to do that, really!

Runtime Engine Architecture for Beginners

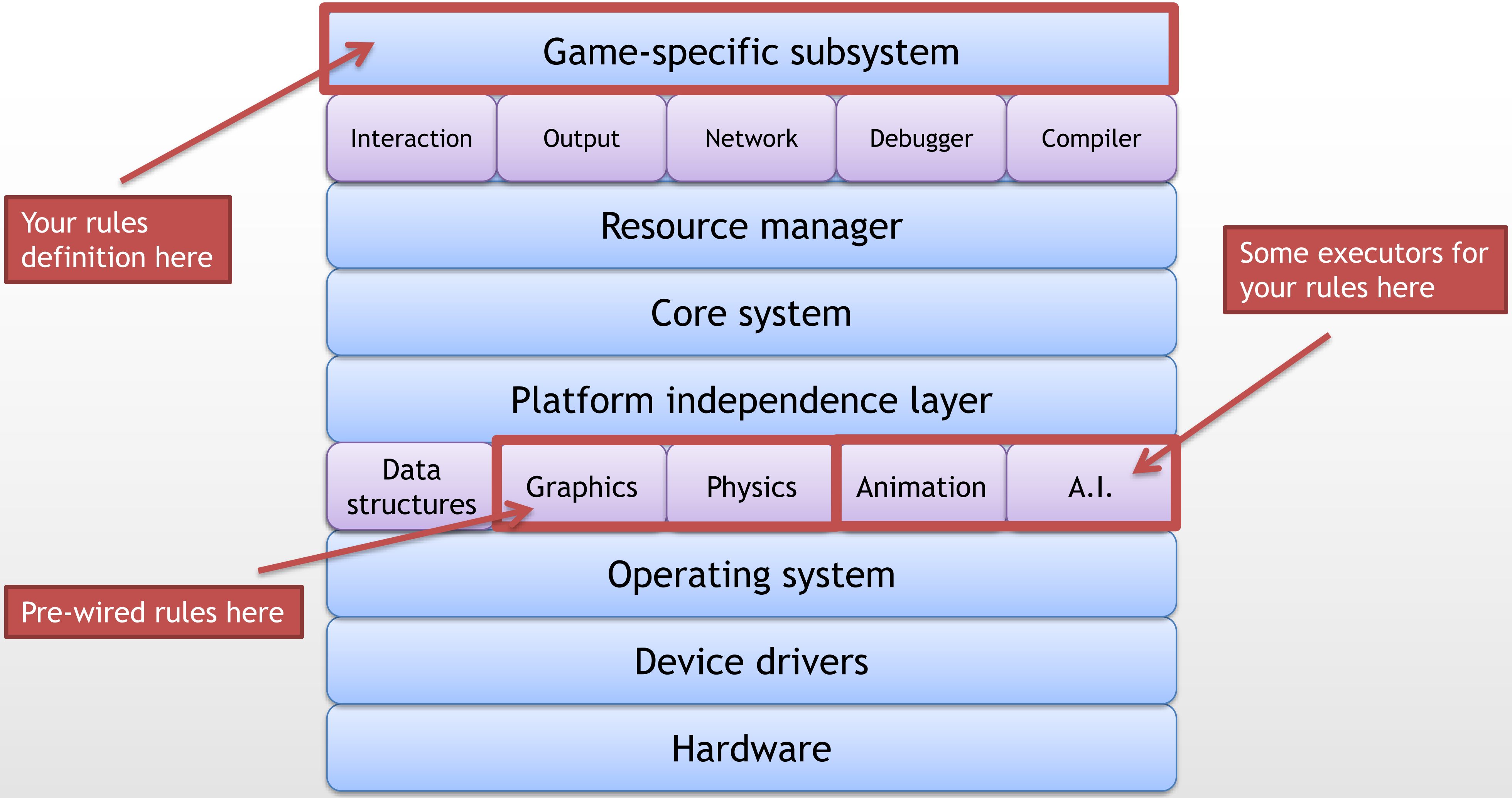
- A game engine is:
  - huge
  - complex
  - made of layers
    - Like many other complex softwares, such as O.S. kernels

Let’s Try Again



We implement the game from outside the black box using the toolset in the top layer of this architecture

Where are the Rules?

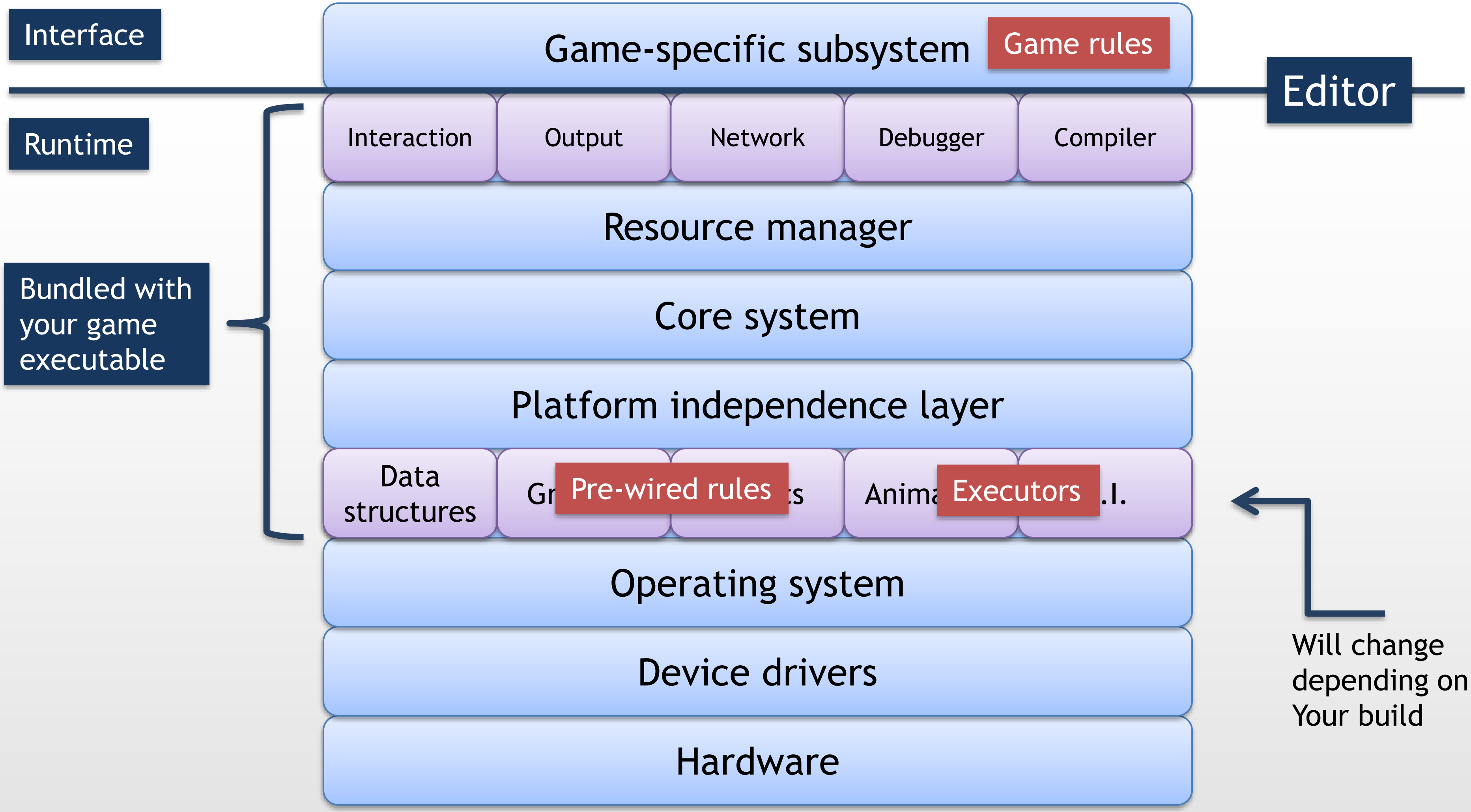


Your rules definition here

Some executors for your rules here

Pre-wired rules here

How do the Engine Uses Them?



Interface

Game-specific subsystem

Game rules

Editor

Runtime

Interaction

Output

Network

Debugger

Compiler

Resource manager

Bundled with your game executable

Core system

Platform independence layer

Data structures

Gr

Pre-wired rules

ts

Anima

Executors

.I.

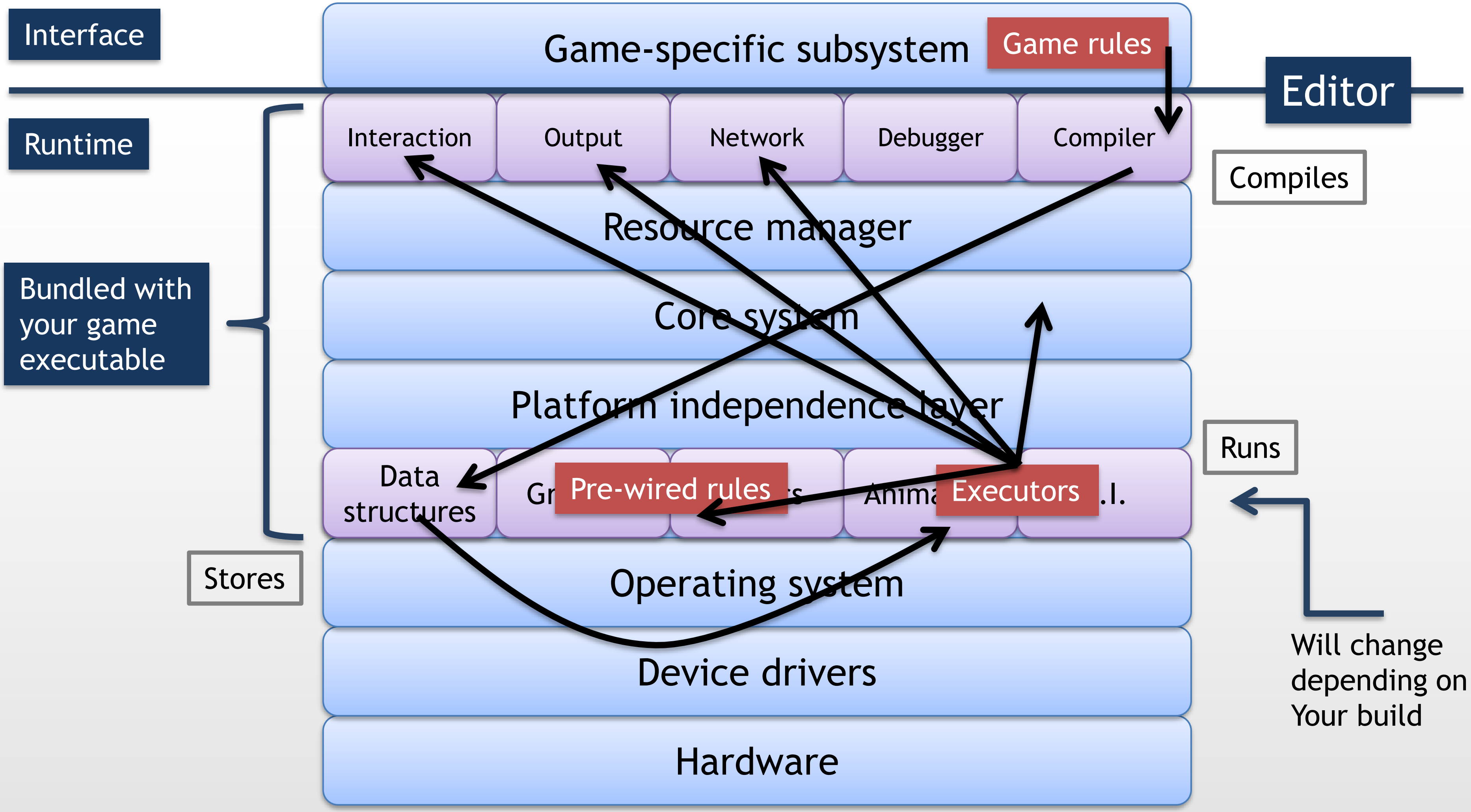
Operating system

Will change depending on Your build

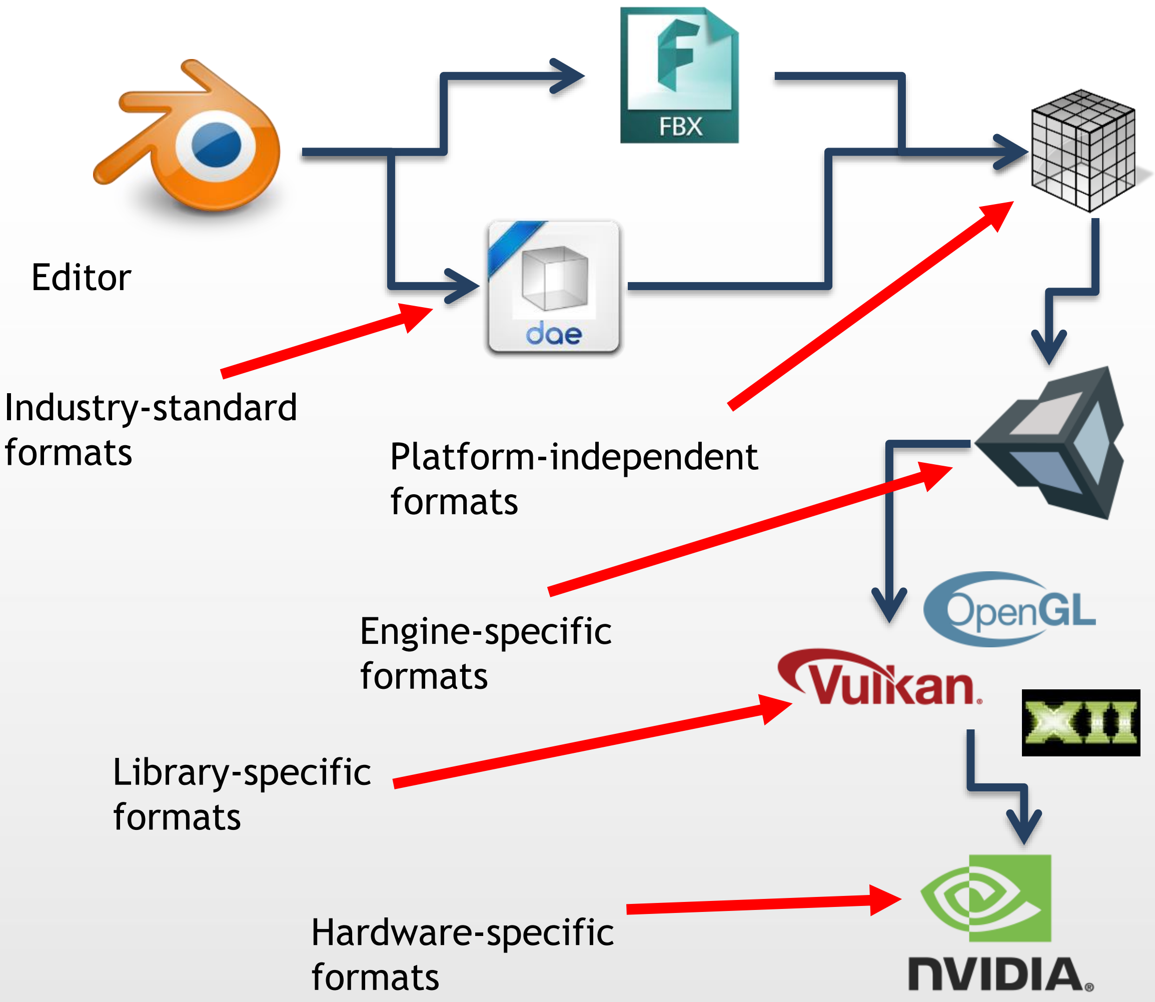
Device drivers

Hardware

How do the Engine Uses Them?



A Word About Graphics (Conversion Pipeline)



Editor

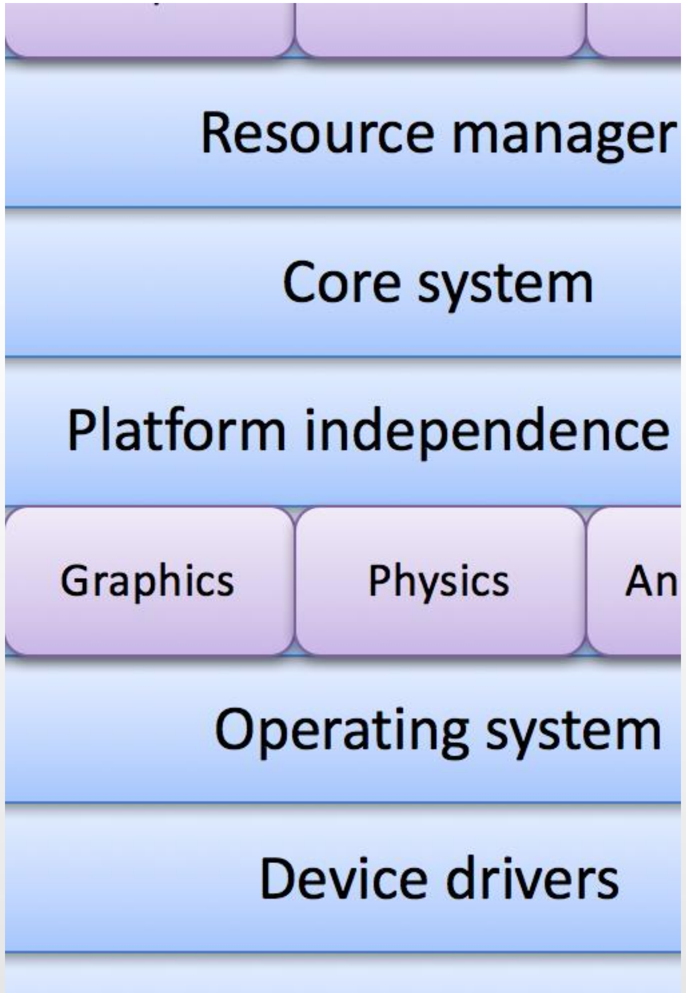
Industry-standard formats

Platform-independent formats

Engine-specific formats

Library-specific formats

Hardware-specific formats



Beware … distinction is blurry

- There is no strict definition of engine modules

  - The rendered might know how to render a full fledge ogre or may just provide basic functionalities

  - The network manager may implement SOAP or provide just sockets

Study Material

- Game Engine Architecture
  3rd edition, ISBN 1138035459
  by Jason Gregory
  Chapter 1, up to § 1.7.4 included