Games and Simulation

Game Engine Runtime Architecture

Game Engine

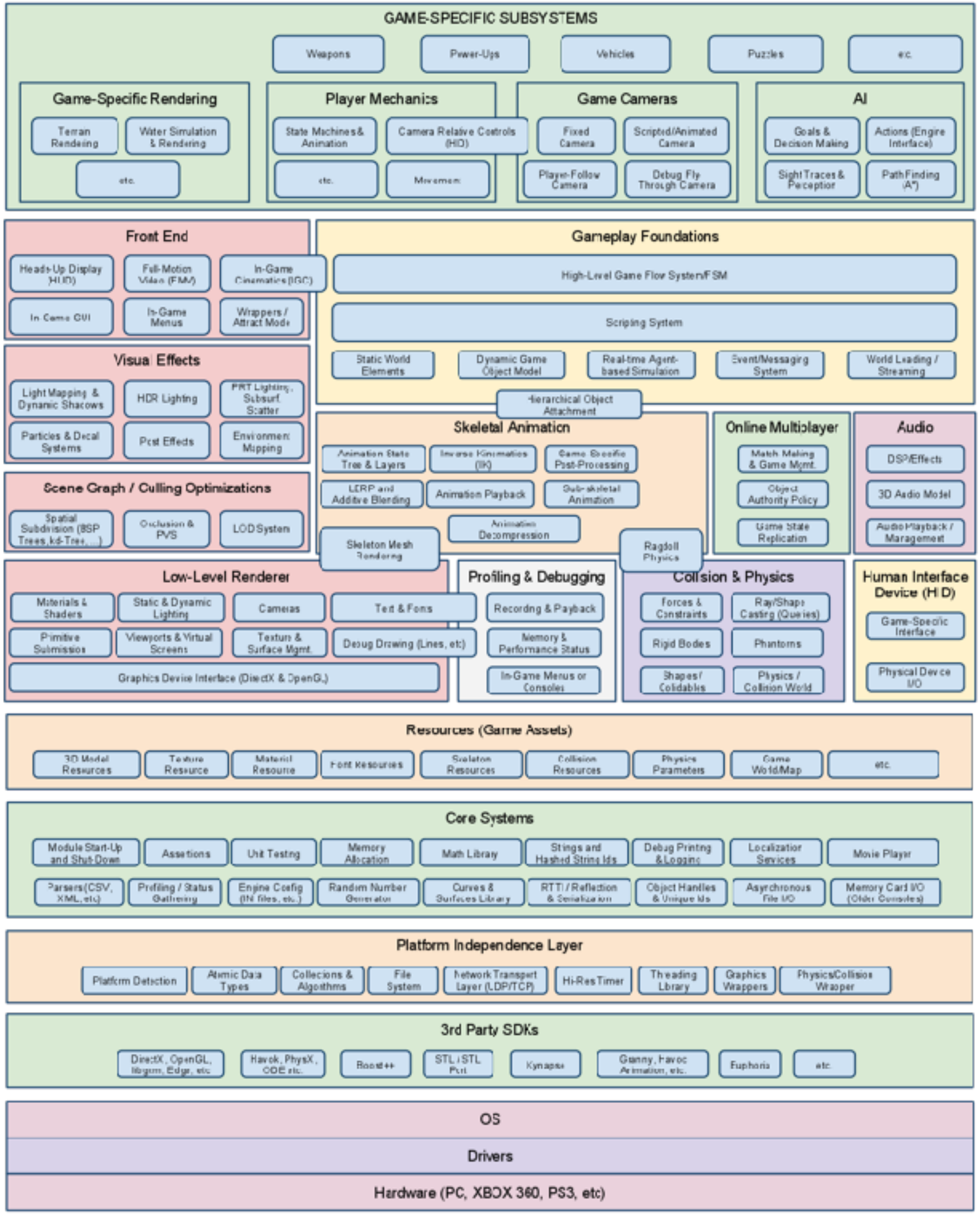
- Usually composed of:
  - A set of tools (tool suite)
  - A runtime component

game specific layer

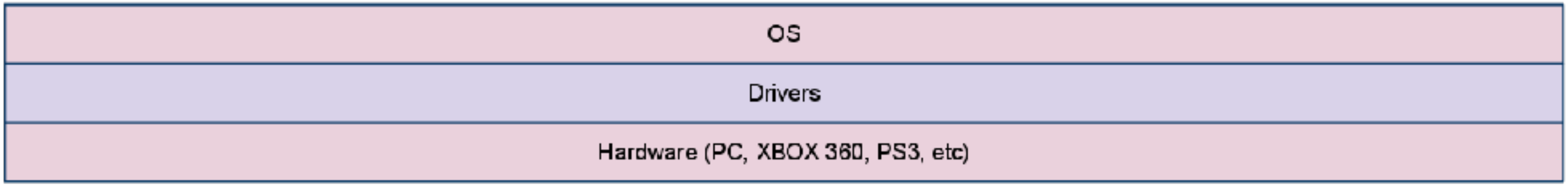
The runtime component of a game engine is a highly complex piece of software built using a layered approach

Increasing abstraction layers

hardware and OS



Hardware and OS

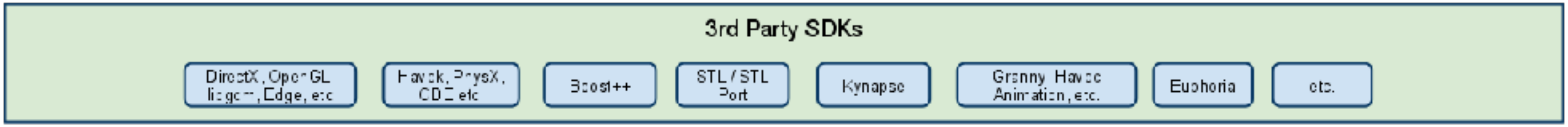

OS

Drivers

Hardware (PC, XBOX 360, PS3, etc)

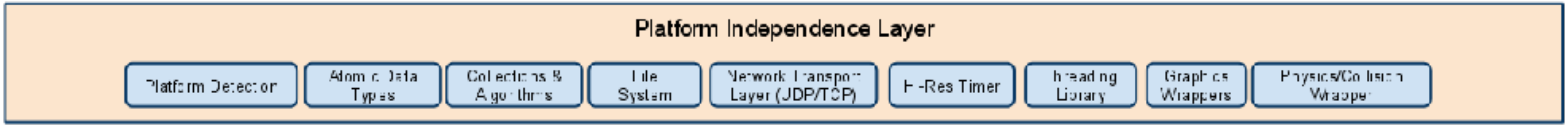
- The OS includes the drivers to access the specific hardware devices available.
- In consoles, the OS layer used to be a thin library linked with the game app.
- From Xbox and PS3 onwards, the OS is able to interrupt the game and display notification messages, reducing the gap between computers and console OS.

3rd Party SDK and middleware



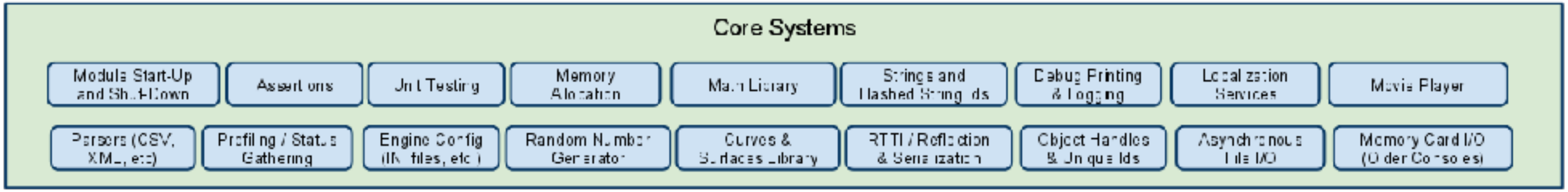
- Include:
  - Graphics API (DirectX, OpenGL, libgcm, …)
  - Collision and Physics (Havok, PhysX, ODE, …)
  - Data structures and algorithms libraries (Boost, STL)
  - Character Animation support (Granny 3D, Havoc Animation, …)
  - AI middleware (Kynapse, …)
  - Dynamic motion synthesis based on Biomechanics (Euphoria)
  - …

Platform Independence Layer



- Required for all game engines running on different hardware platforms.
- Most studios sell their games for several platforms (except first party studios)
- Shields the rest (top part) of the engine from knowledge of the underlying platform.

Core Systems



Core Systems

Module Start-Up and Shut-Down

Assertions

Unit Testing

Memory Allocation

Math Library

Strings and Hashed String Ids

Debug Printing & Logging

Localization Services

Movie Player

Parsers (CSV, XML, etc.)

Profiling / Stats Gathering

Engine Config (INI files, etc.)

Random Number Generator

Curves & Surfaces Library

RTTI / Reflection & Serialization

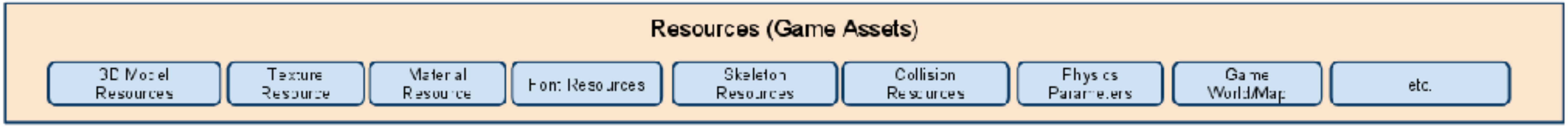
Object Handles & Unique Ids

Asynchronous File I/O

Memory Card I/O (Older Consoles)

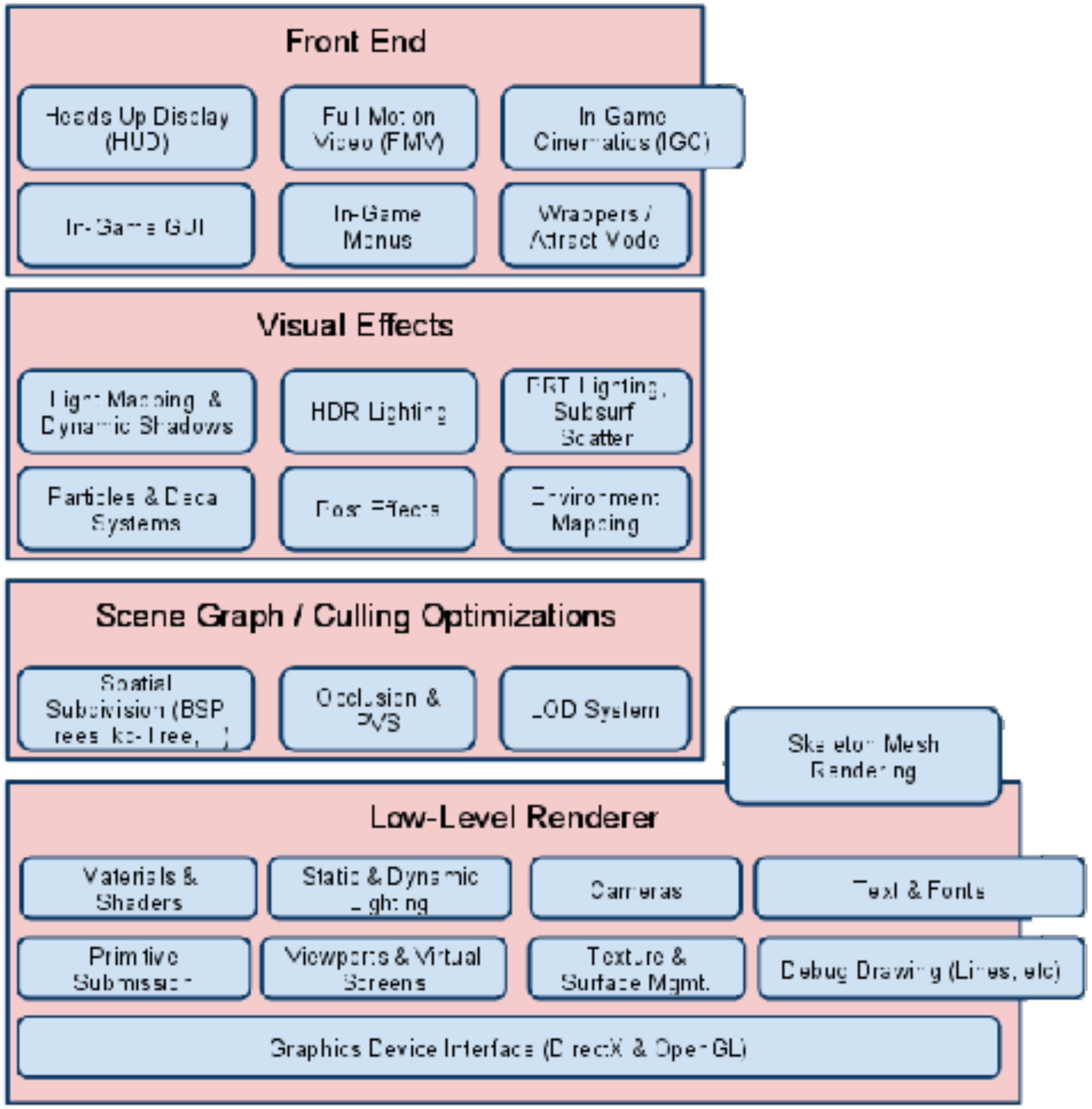
- Examples of “stuff” that go into Core System:
    - Assertions
    - Math library
    - Memory Management
    - Custom Data Structures and Algorithms

Resource Manager



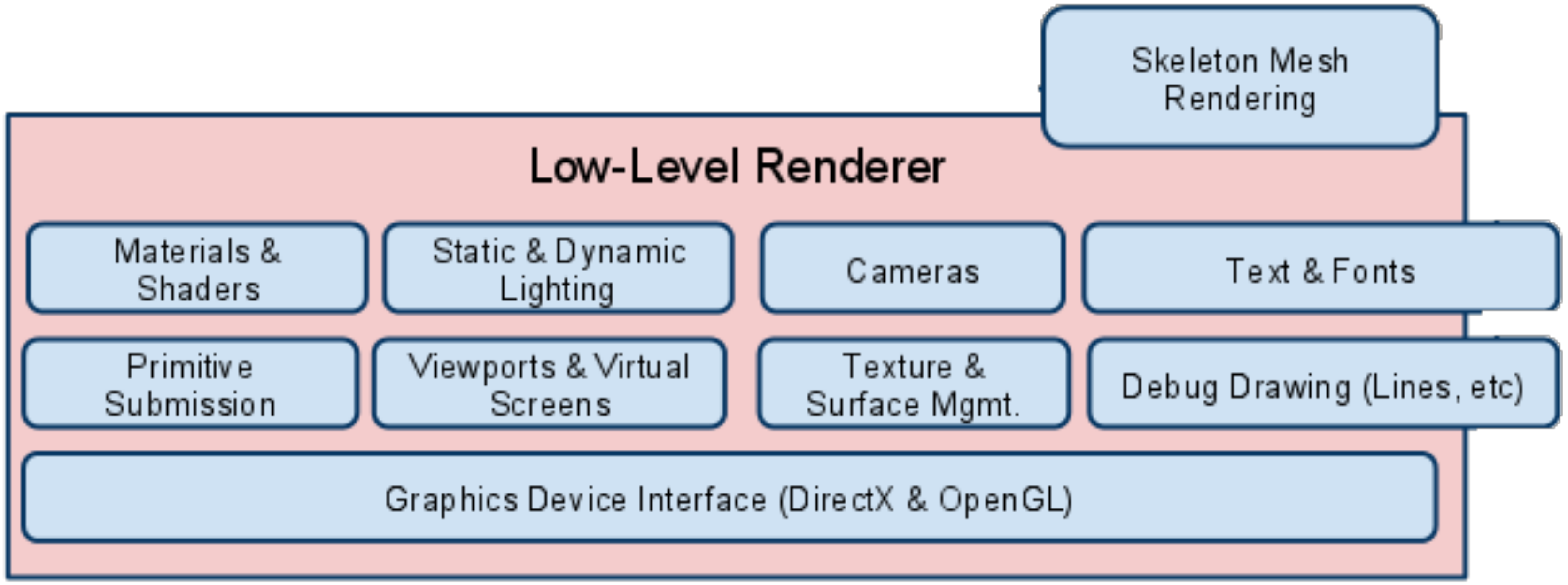
- Provide interfaces to access game assets and other input data:
  - 3D models, textures, fonts, audio files, skeletons, terrains, …
  - Ad-hoc access to single files vs. resource bundles

Rendering Engine



- One of the largest and most complex components
- Usually structured in several layers with upper levels providing higher abstraction features

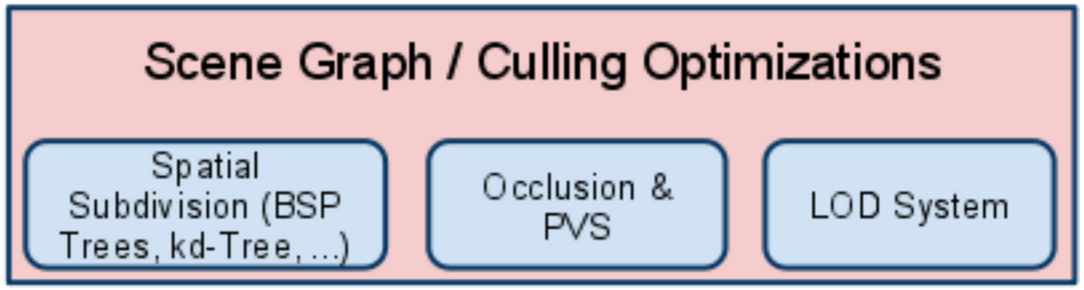
Low Level Rendering


Skeleton Mesh Rendering
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)

- The lower part deals with graphics API: enumerate the devices, buffer allocations (color, depth, stencil, etc.)

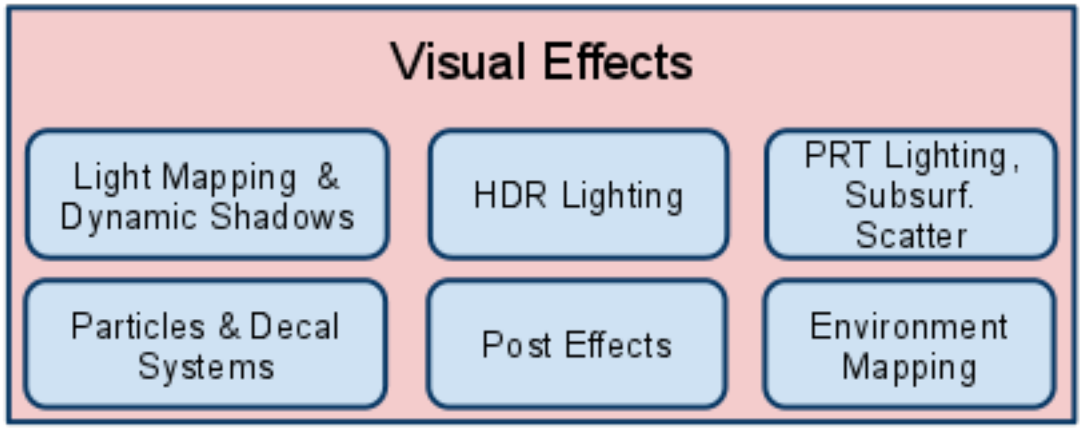
- The upper parts offer support commonly found in 3D graphics pipelines: primitive submission, materials and shader support, textures, lighting, viewports, cameras, …

Scene Graph handling


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

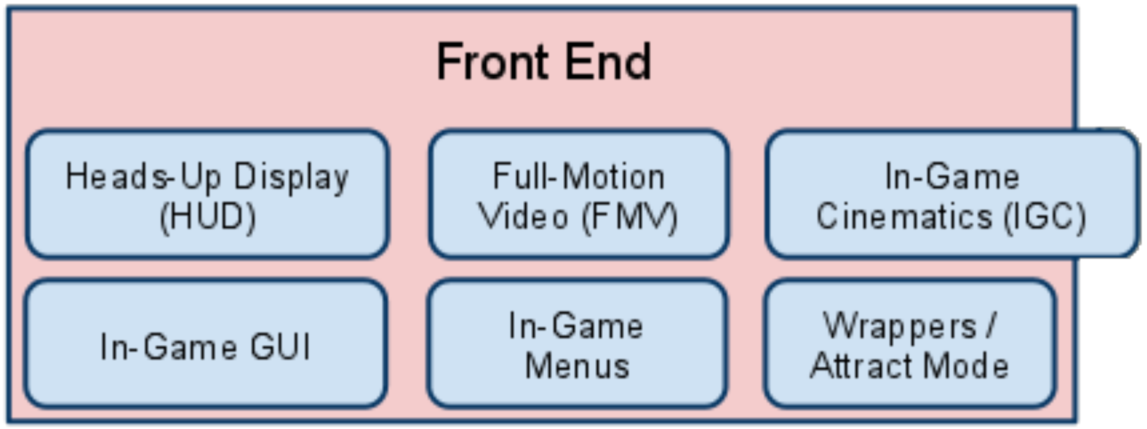
- Game action can potentially take place in large virtual worlds and/or include a very large number of actors/entities.
- Efficient handling of this large amount of model data is needed so not to stall the pipeline.
- Spatial data structures are employed to discard objects outside of view volume.
- Objects may occlude other objects and lead to further optimisations.
- Objects at different distances from the camera may be rendered with different levels of detail.

Visual Effects


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

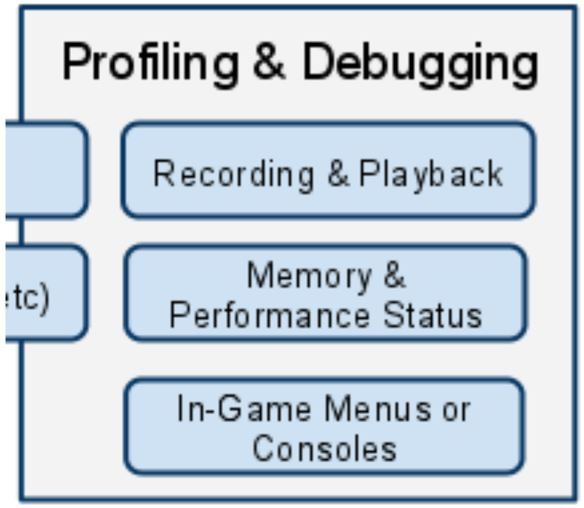
- Modern video games include highly advanced visual effects:
  - particle systems (water splashes, smoke, fire,…)
  - decals (bullet holes, footprints, …)
  - light mapping and environment mapping
  - dynamic shadows
  - full-screen post effects (HDR, FSAA, color correction/shift)

Front End



- Examples:
  - Heads-up display
  - In-game menus, console, development tools (may not ship with the game)
  - In-game GUI to manipulate the character's inventory, deploy units to battle, …
  - Video playback and scripted Cinematics Choreography using the Engine
- A large portion of these features require making 2D textures to the screen (orthographic view) or in 3D billboards (always facing the camera).

Profiling and Debugging



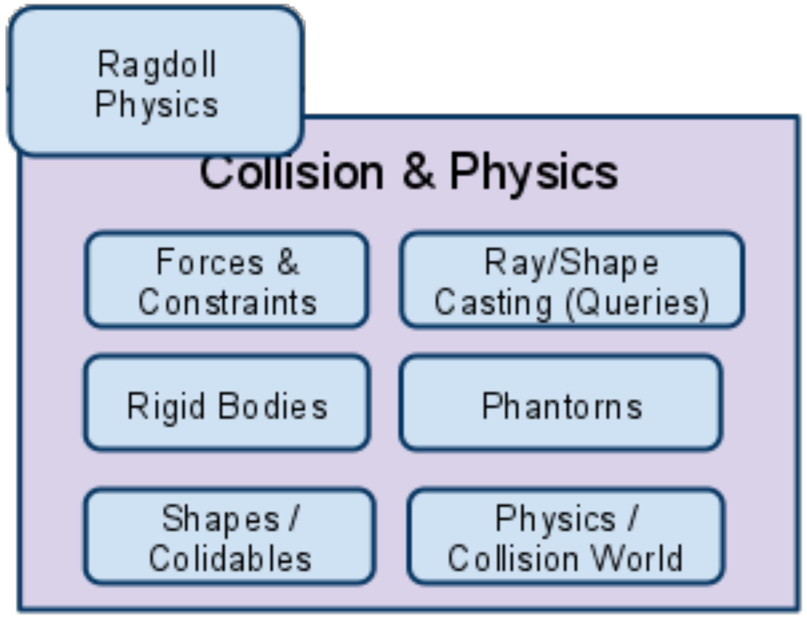
- Performance is a top priority in game development

- Console hardware quickly becomes outdated and resources are scarce.

- In-game debugging tools often include a development console and drawing facilities.

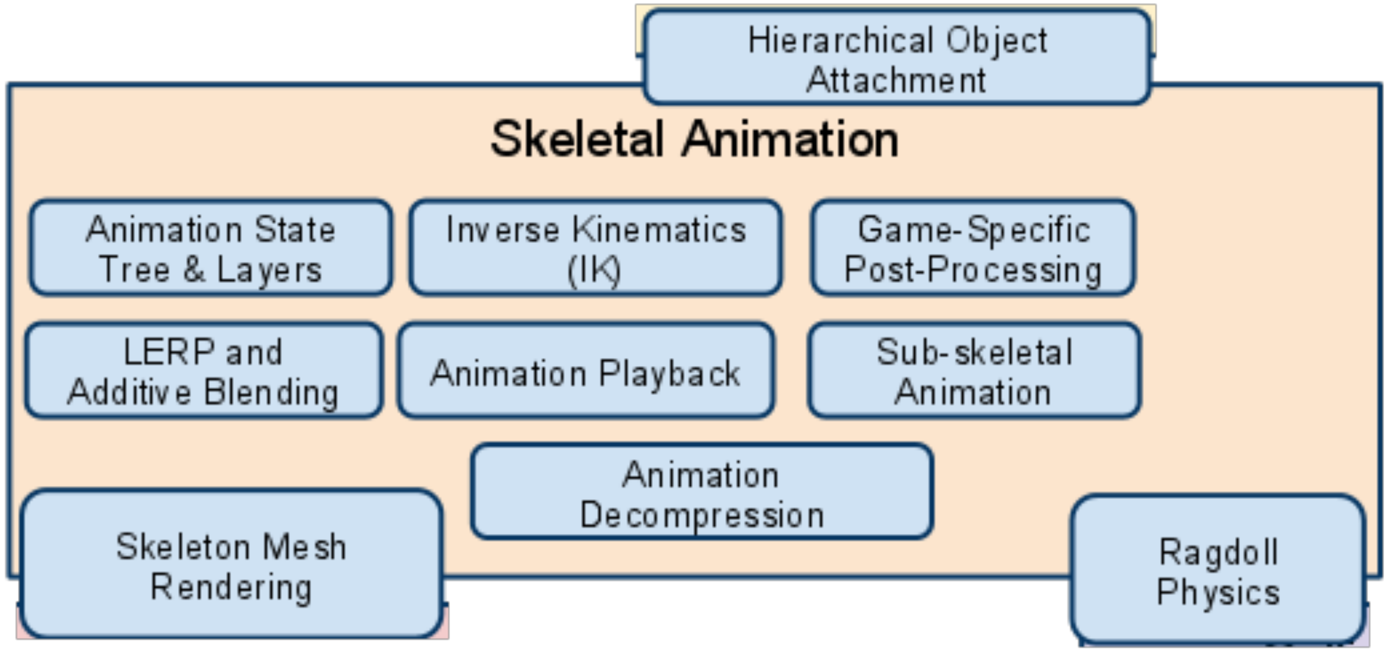
- Recording, playback, stopping the game clock, dumping stats, etc…

Collisions and Physics



- Collisions is at the heart of every game. Without them, objects would overlap and no interaction with the virtual world would be possible.
- Collision is commonly done with low detail, simpler, shapes
- The Physics system comprises rigid body dynamic simulation based on forces, torques and restrictions.

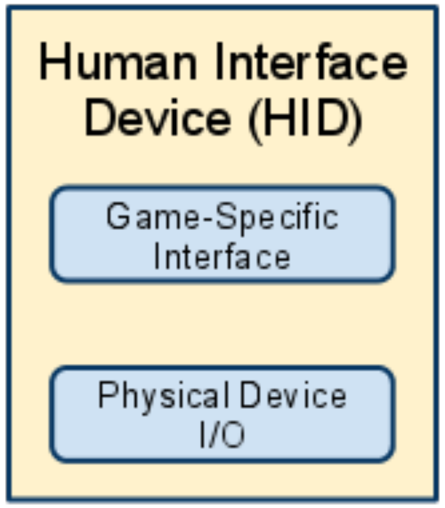
Animation



- Animation systems are needed for modelling humans, animals, cartoon characters, robots,…
- types of animation systems:
    - sprite/texture animation (2D)
    - rigid body hierarchy animation (machines, robots)
    - skeletal animation (humans, animals, cartoon characters,…)
    - vertex animation
    - morph targets

Human Interface Devices

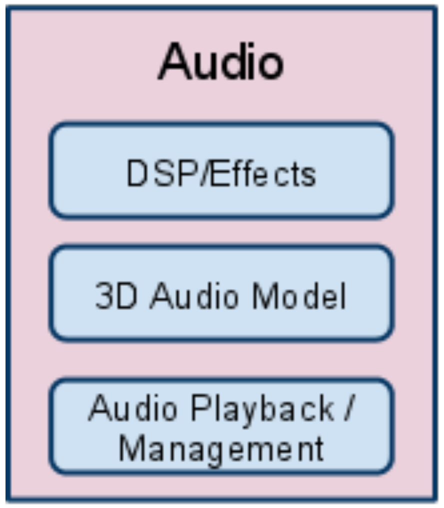
- Process user input from HID:
  - keyboard, mouse, joypad, specialised game controllers
- process raw data:
  - dead zone around center position of joypad stick
  - debounce button presses
  - interpret and smooth signals from accelerometer
  - allow for user customised mappings
  - detect chords, sequences and gestures



Audio

- Just as important as rendering and physics

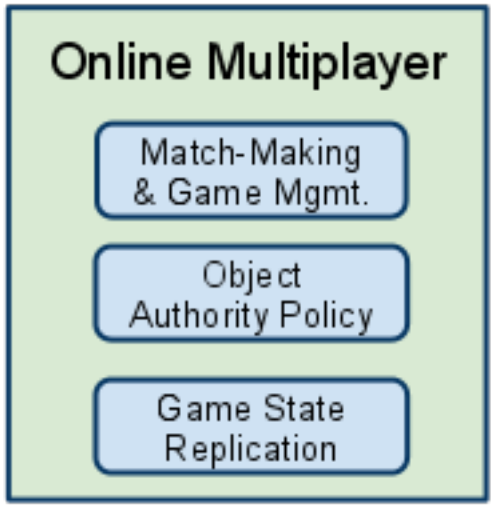
- Essential to create immersive experiences



- Can be as simple as basic playback but often includes 3D spatialisation and DSP/Effects

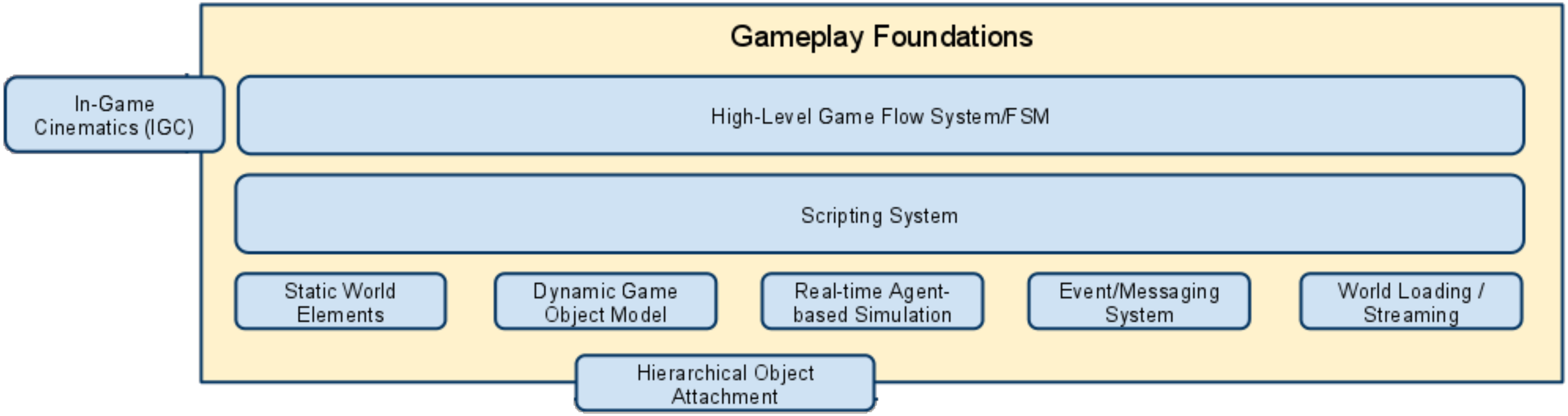
- Examples of Audio engines: X3DAudio, SoundR!ot, Scream

Online Multiplayer/ Networking



- Allow for multiple human players in the same virtual world
- 4 flavours:
  - Single-screen multiplayer
  - Split-screen multiplayer
  - Networked multiplayer - Each machine hosts one player
  - Massively multiplayer online games (MMOG) - A battery of central servers host a common and persistent game world.
- Easy to change a multiplayer game into a single player, harder the other way around.

Gameplay Foundation Systems

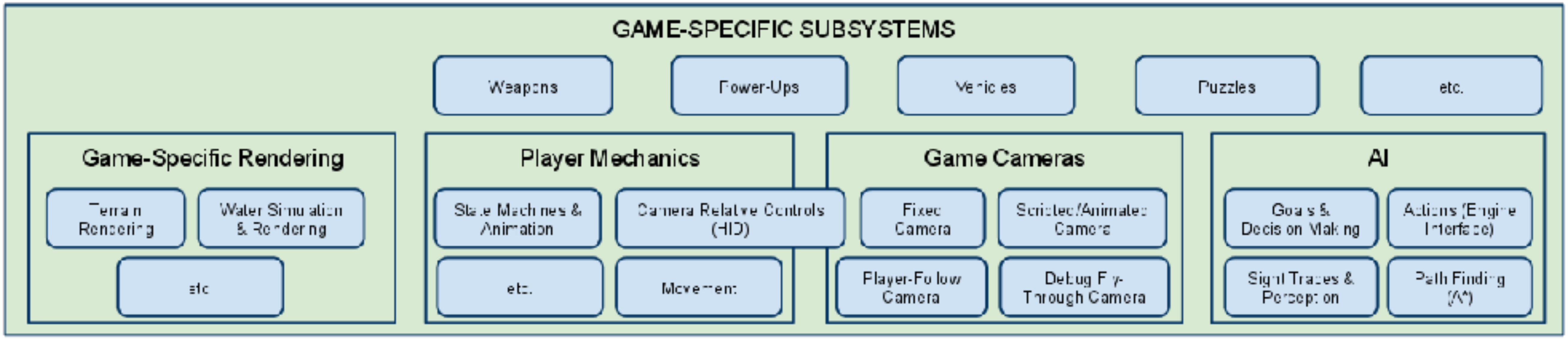

Gameplay Foundations
In-Game Cinematics (IGC)
High-Level Game Flow System/FSM
Scripting System
Static World Elements
Dynamic Game Object Model
Real-time Agent-based Simulation
Event/Messaging System
World Loading / Streaming
Hierarchical Object Attachment

- Rules of the game world/character abilities,…
- Implemented in native or scripting language

Gameplay Foundation Systems

- Includes:
  - Game world and object models for static background, dynamic rigid bodies, PC and NPC, weapons, projectiles, vehicles, lights, cameras,…
  - Event system for inter object communication
  - Scripting system for easier and more rapid development of game rules
  - AI foundations

Game Specific Subsystems



Further readings and resources

- Cap. 1 Game Engine Architecture - Jason Gregory (adopted book)
- http://www.gameenginebook.com/index.html