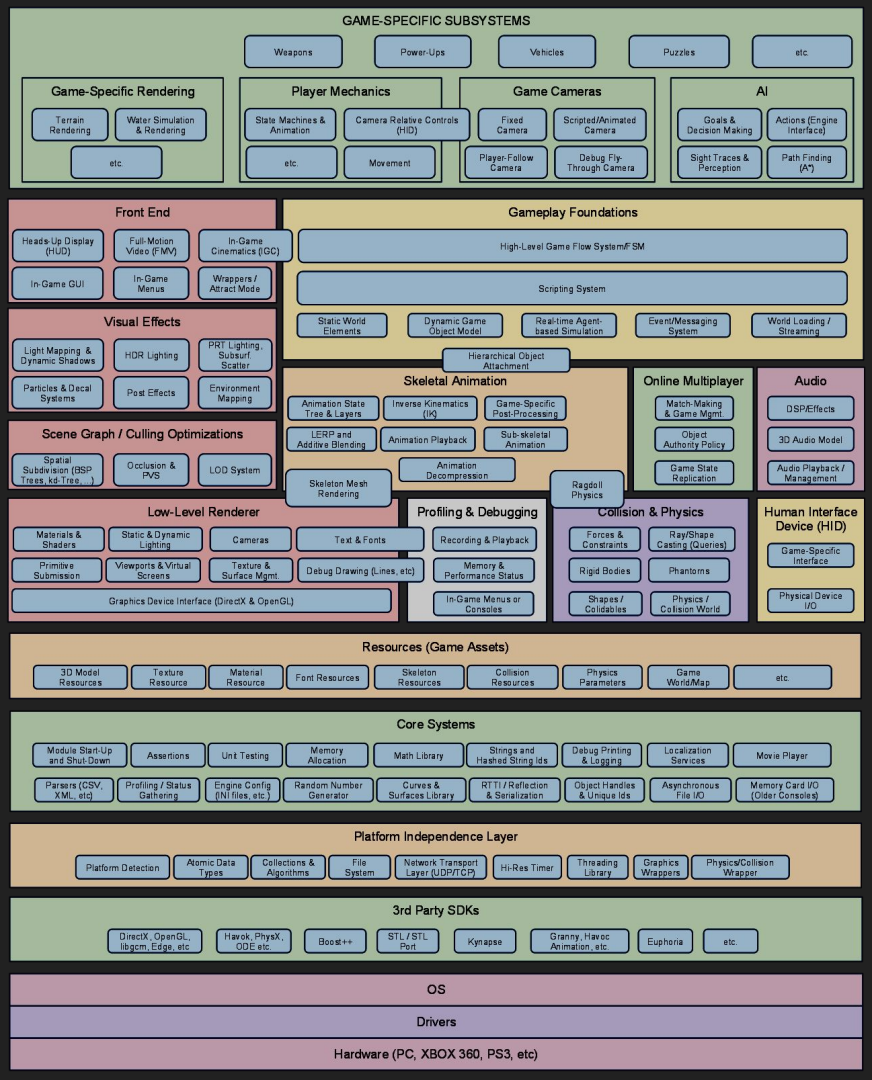
"I call architecture frozen music" :
- Johann Wolfgang von Goethe.

System Architecture

It is important to decide on a good architecture before starting your game.

Why?

Games are complex.


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
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
Scene Graph / Culling Optimizations
Spatial Subdivision (BSP Trees, kd-Tree, ...)
Occlusion & PVS
LOD System
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 / Collidables
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
etc.
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 / Status 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)
Platform Independence Layer
Platform Detection
Atomic Data Types
Collections & Algorithms
File System
Network Transport Layer (UDP/TCP)
Hi-Res Timer
Threading Library
Graphics Wrappers
Physics/Collision Wrapper
3rd Party SDKs
DirectX, OpenGL, libgcm, Edge, etc.
Havok, PhysX, ODE etc.
Boost++
STL / STL Port
Kynapse
Granny, Havoc Animation, etc.
Euphoria
etc.
OS
Drivers
Hardware (PC, XBOX 360, PS3, etc)

If you do not plan ahead, you will likely end up with code that is more complicated than it has to be.

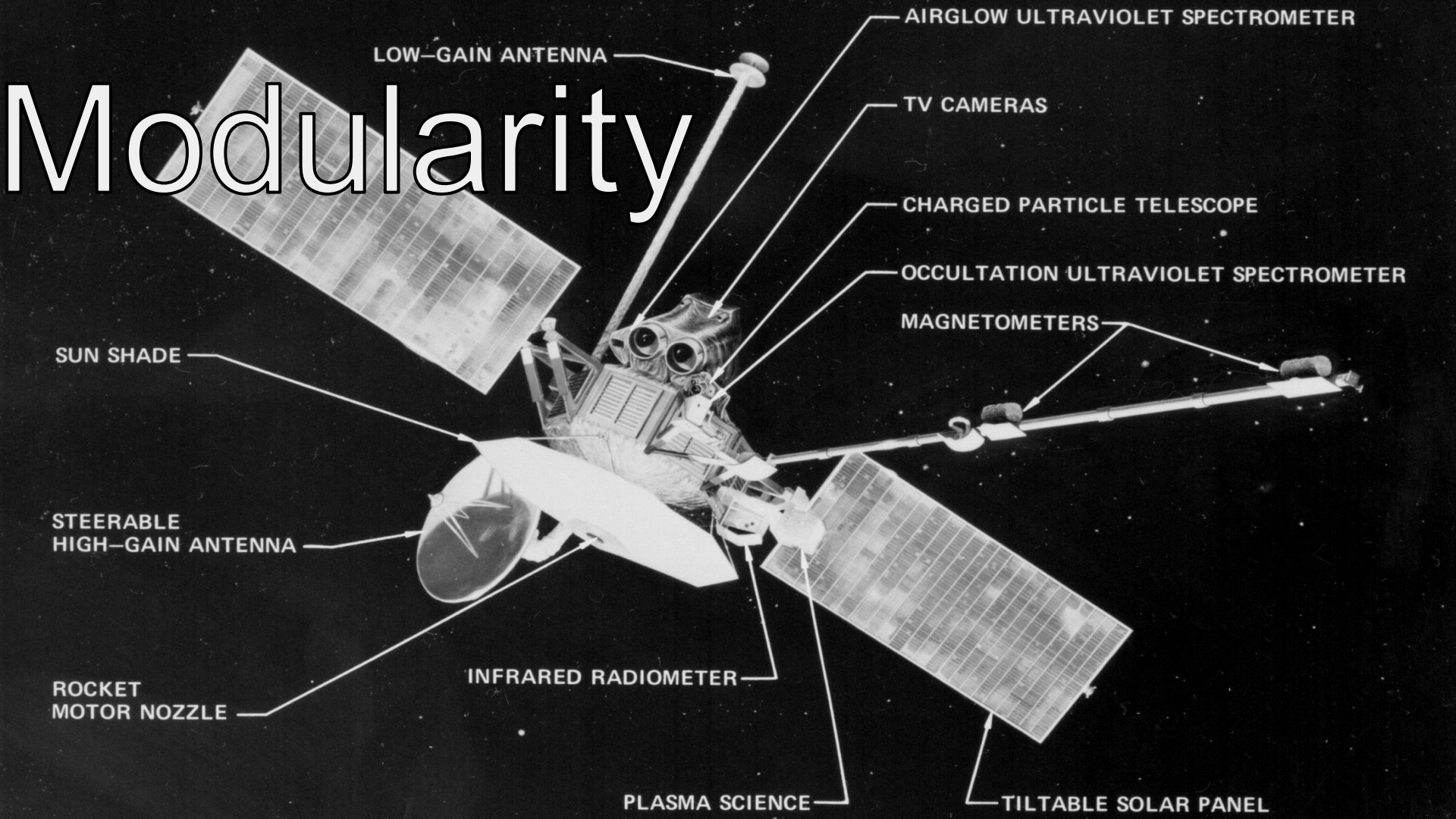
Game Loop

```java
public class Game
{
    public static void main(String[] args)
    {
        //Initialise
        World world = new World();
        Video video = new Video();
        Timer timer = new Timer();

        //Run

        while(world.isRunning())
        {
            timer.startFrame();

            world.frame(time.deltaTime());
            video.frame(world);

            timer.endFrame();
        }

        //Shutdown
        video.shutdown();
        world.shutdown();
    }
}
```

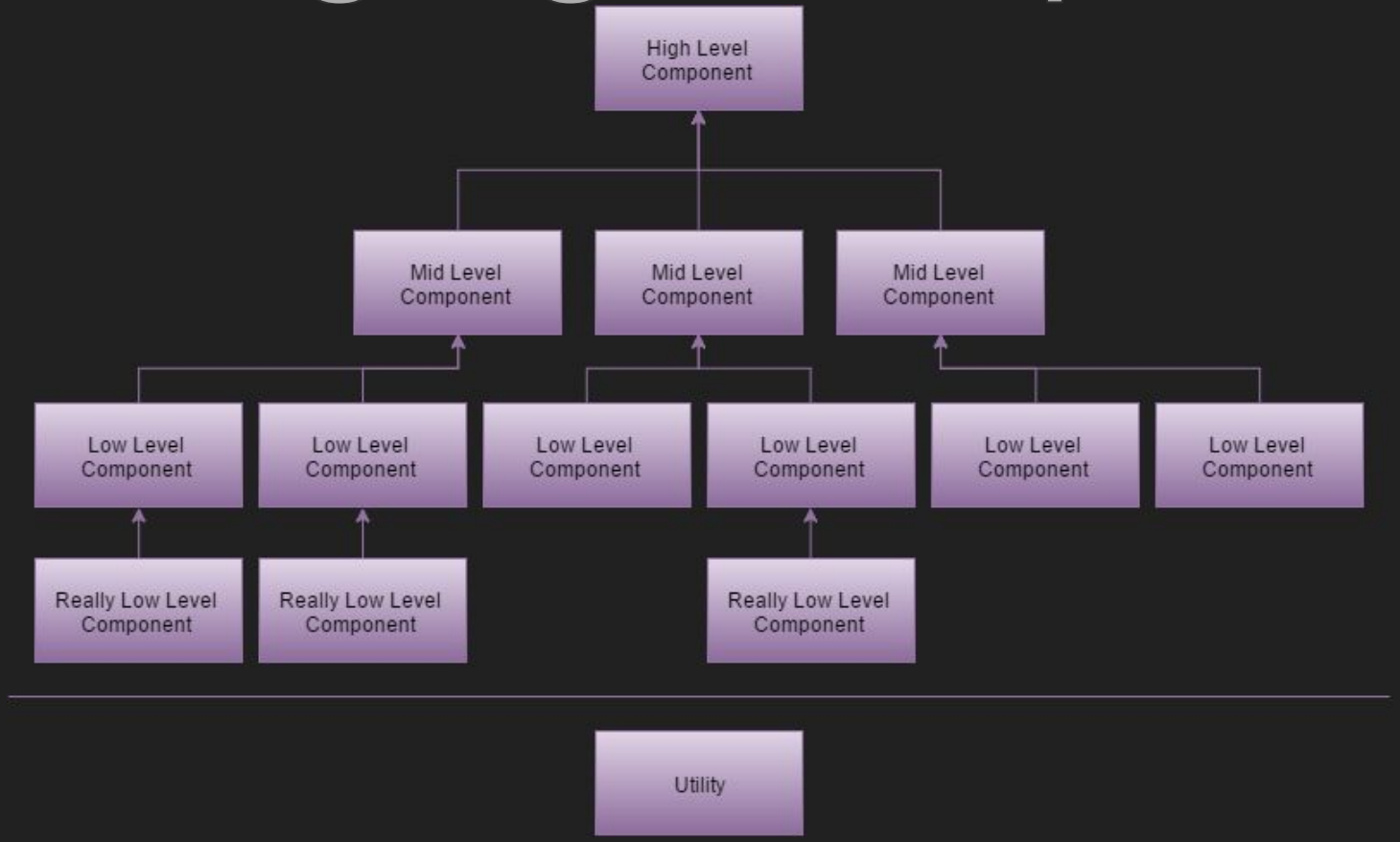
```
public class World
{
    public void frame(float deltaTime)
    {
        if(keyDown(Key.Esc))
        {
            running = false;
            return;
        }
        else
        {
            for(int i = 0; i < allWorldEntities.length; ++i)
                allWorldEntities[i].update(deltaTime);
        }
    }
```

```java
public class Video
{
    public void frame(World world)
    {
        for(int i = 0; i < world.getAllWorldEntities(); ++i)
            draw(world.getAllWorldEntities);

        applyVisualEffects();

        show();
    }
```

Modularity

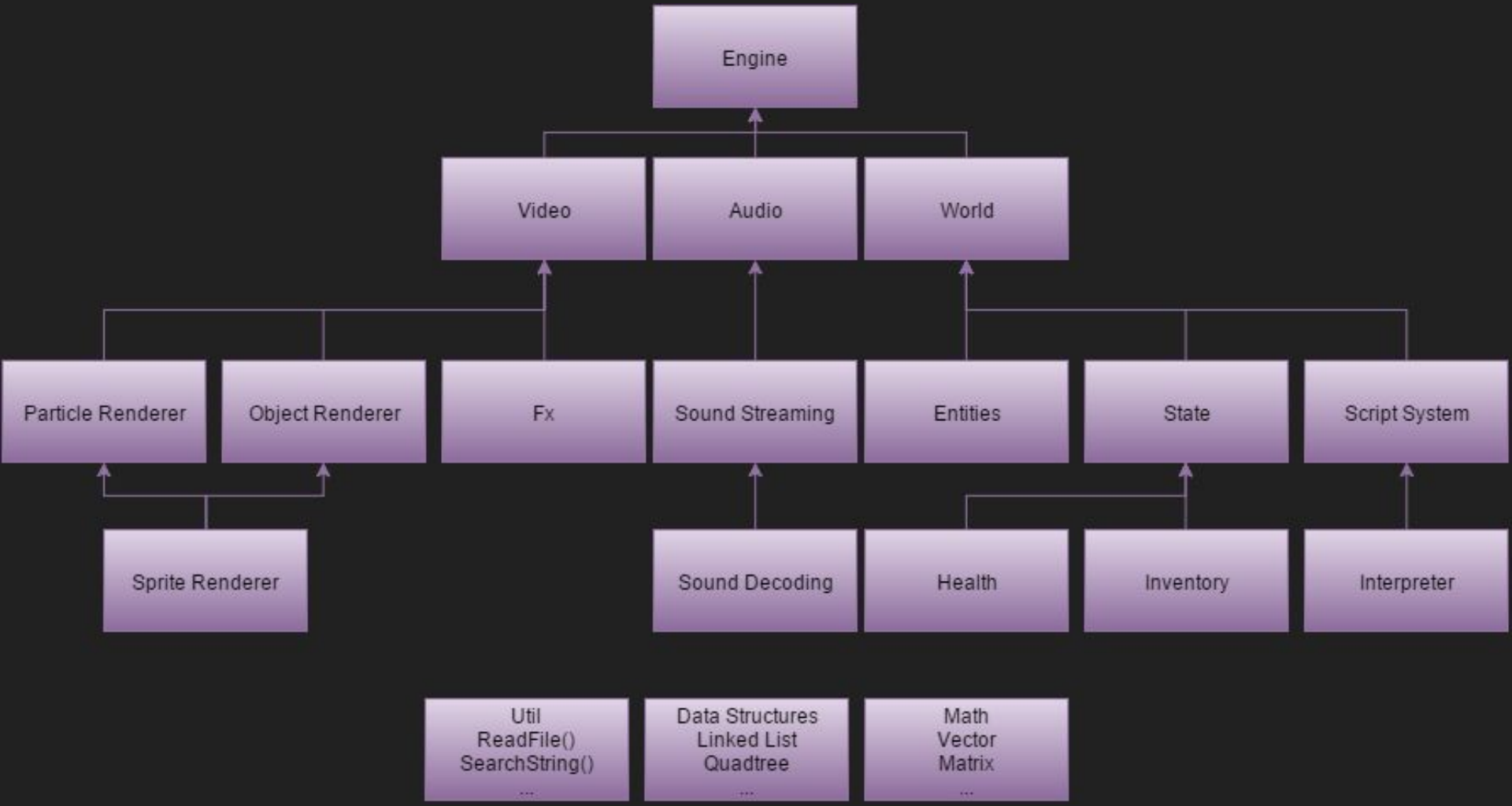

AIRGLOW ULTRAVIOLET SPECTROMETER
LOW–GAIN ANTENNA
TV CAMERAS
CHARGED PARTICLE TELESCOPE
OCCULTATION ULTRAVIOLET SPECTROMETER
MAGNETOMETERS
SUN SHADE
STEERABLE
HIGH–GAIN ANTENNA
ROCKET
MOTOR NOZZLE
INFRARED RADIOMETER
PLASMA SCIENCE
TILTABLE SOLAR PANEL

Managing Complexity

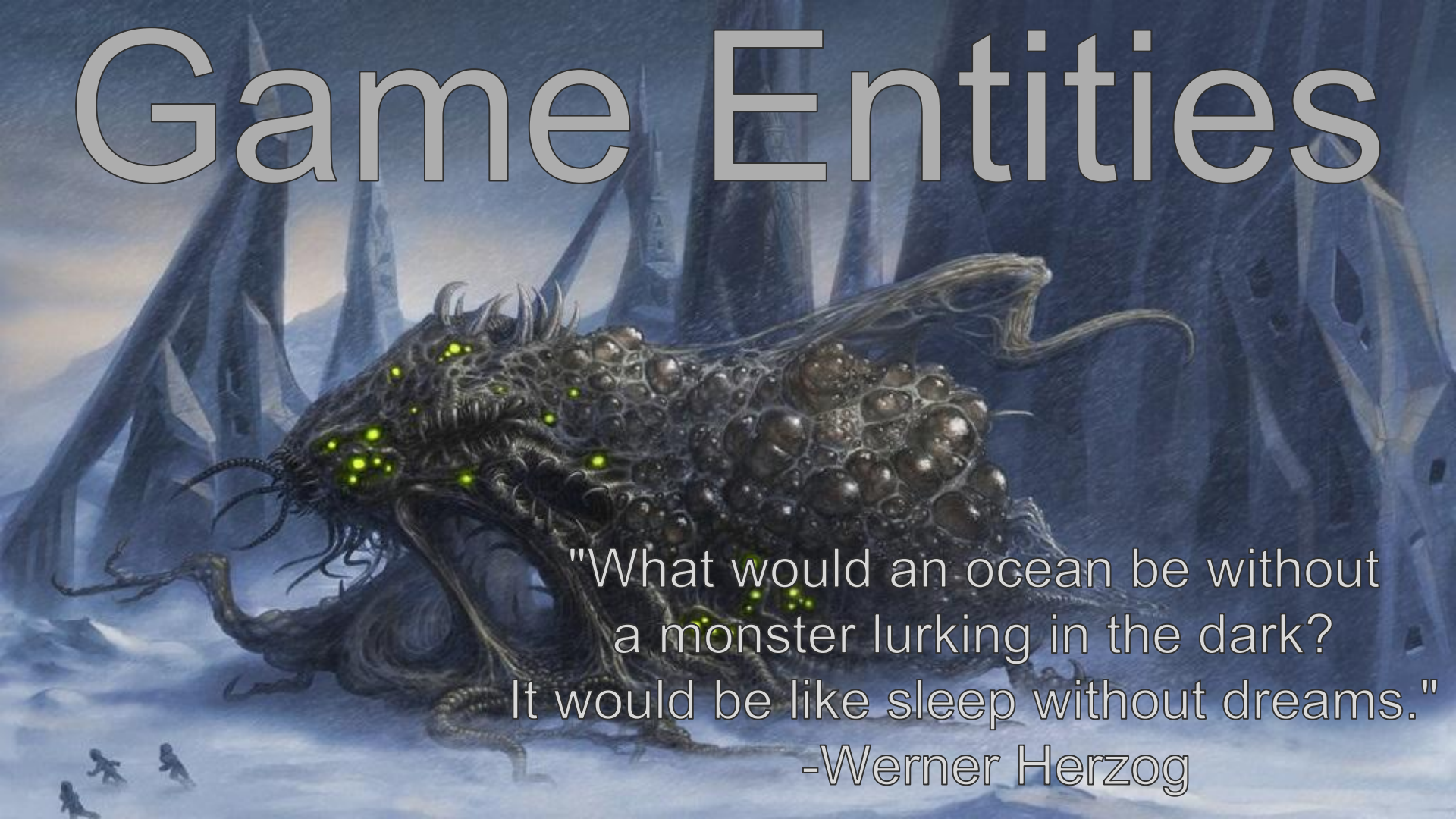


similar to a social hierarchy

manager
sub-managers
sub-sub-managers
sub-sub-sub-managers
...
workers
tools

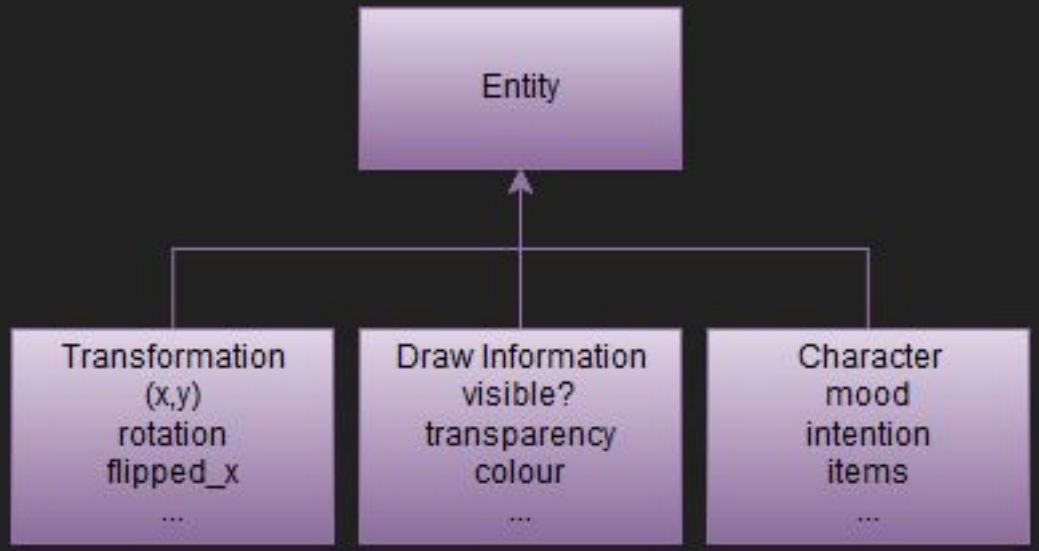

Engine
Video
Audio
World
Particle Renderer
Object Renderer
Fx
Sound Streaming
Entities
State
Script System
Sprite Renderer
Sound Decoding
Health
Inventory
Interpreter
Util
ReadFile()
SearchString()
...
Data Structures
Linked List
Quadtree
...
Math
Vector
Matrix
...

Game Entities



"What would an ocean be without
a monster lurking in the dark?
It would be like sleep without dreams."
-Werner Herzog

an entity is data and behaviour




```
void jump(float height)
{
    this.position.y += height;
}

void attack(Character target)
{
    this.state = State.Attack;
    this.target = target;
}

void update(float deltaTime)
{
    if(state == State.Idle)
        doNothing();
    else if (state == State.Attack)
    {
        moveTowards(target, deltaTile);
        thwap();
    }
    //else if (state == ...)
    //  ...
}
```

an interface declares that an entity can do something

```
interface ICanJump
{
    public void jump(float height);
}

interface ICanRun
{
    public void run(Vector2 direction, float speed);
}

public class RunningJumpingMonster implements ICanRun, ICanJump
{
    Vector3 position = new Vector3();

    @Override
    public void jump(float height)
    {
        position.z += height;
    }

    @Override
    public void run(Vector2 direction, float speed)
    {
        Vector2 vector = direction.normalised().multiply(speed);
        position.x += vector.x;
        position.y += vector.y;
    }
}
```

```
RunningJumpingMonster monster1 = new RunningJumpingMonster();
RunningJumpingMonster monster2 = new RunningJumpingMonster();
RunningJumpingMonster monster3 = new RunningJumpingMonster();

ICanRun  = arrayOfRunningThings = new ICanRun [10];
ICanJump = arrayOfJumpingThings = new ICanJump[10];

arrayOfJumpingThings[0] = monster1;
arrayOfJumpingThings[1] = monster2;

arrayOfRunningThings[0] = monster1;
arrayOfRunningThings[1] = monster3;

for (int i = 0; arrayOfJumpingThings.length; ++i)
    if(arrayOfJumpingThings[i]!=null)
        arrayOfJumpingThings[i].jump(0.5f);
```

Instancing

games have things
thousands of things

entities often contain redundant information

assume 300 16*32 sprites:

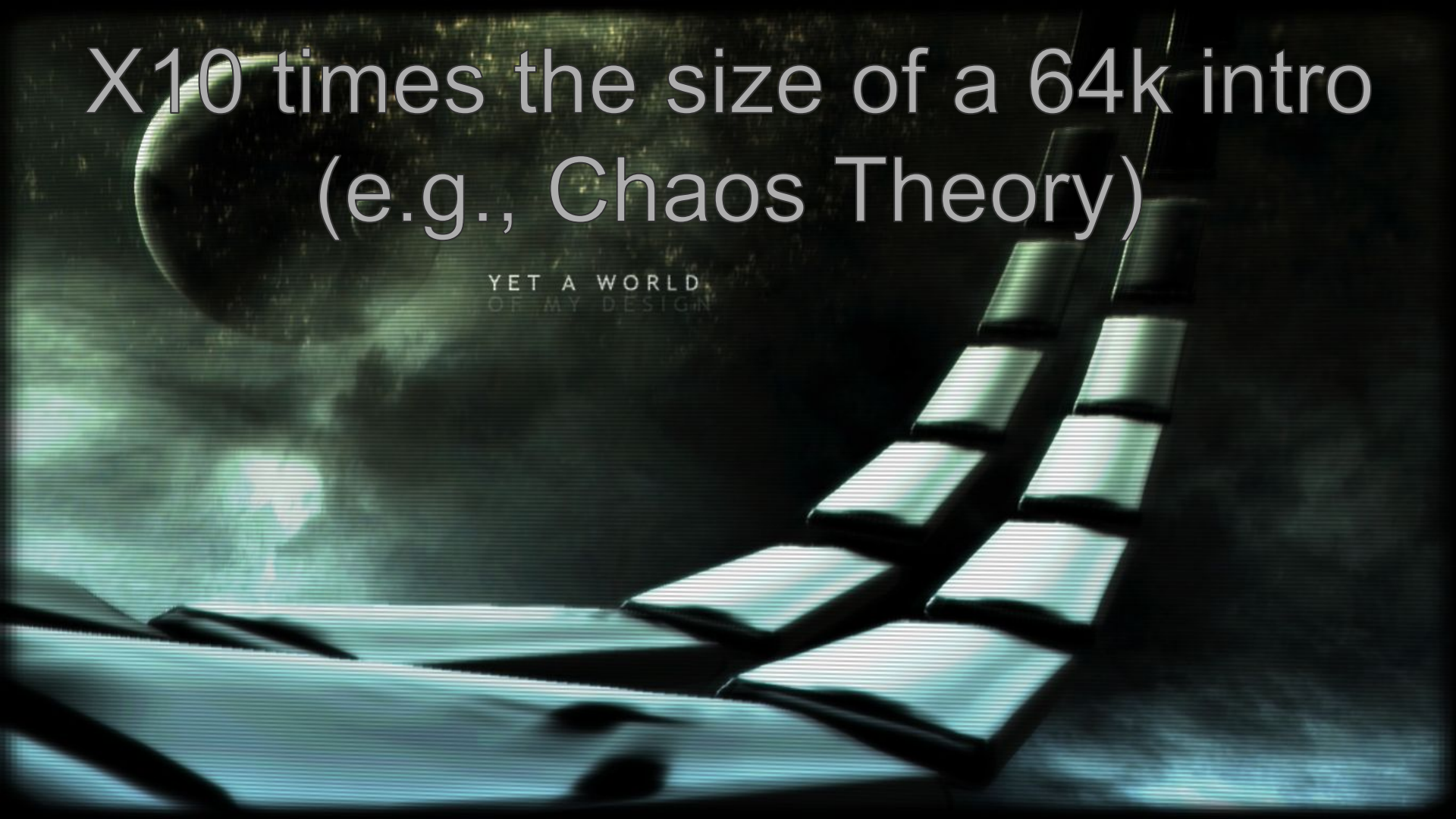
pixels*components*count

(16*32)*4*300

= 614400 bytes

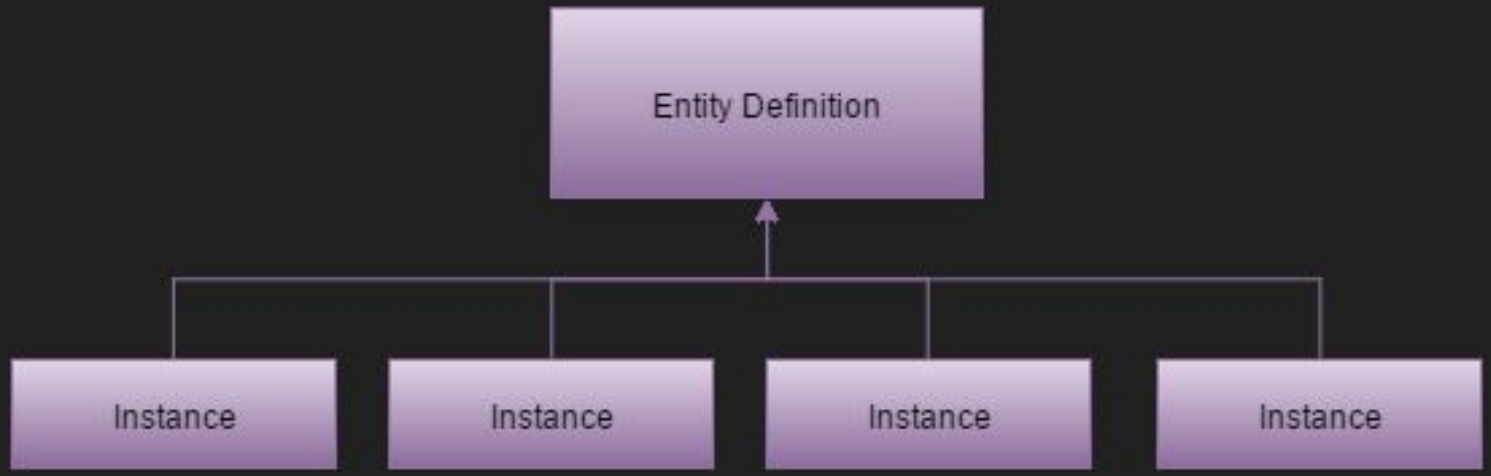
≈ 600 kilobytes

X10 times the size of a 64k intro (e.g., Chaos Theory)



solution: instancing

redundancy is minimised by storing everything only once




{Position, Flipped}
(5,32), false
(8,2), true
(2,3), true
(8,2), false
(12,54), true
(12,9), false
(3,12), true
.....

with instancing:

$$(16*32)*4 + (4+4+4)*300$$
$$= 5648 \text{ bytes}$$
$$\approx 5.5 \text{ kilobytes}$$

0.9% of original size

Video

"In order to carry a positive action we must develop here a positive vision."
- Dalai Lama.

```
public class Video
{
    public Video()
    {
        initialiseWindow();
        prepareForDrawing();
    }

    public void startFrame()
    {
        clearScreen();
        beginDrawBatch();
    }

    public void endFrame()
    {
        endDrawBatch();
        presentToPlayer();
    }

    public void draw(Sprite sprite)
    {
        setDrawTexture(sprite.texture);
        setDrawPosition(sprite.position);
        drawWithState();
    }
}
```

```
Video video = new Video();

Sprite s1 = new Sprite("BadGuy.png");
Sprite s2 = new Sprite("GoodGuy.png");
Sprite s3 = new Sprite("Background.png");

while(game.isRunning)
{
    video.startFrame();

    video.draw(s3);
    video.draw(s1);
    video.draw(s2);

    video.endFrame();
}
```

Audio

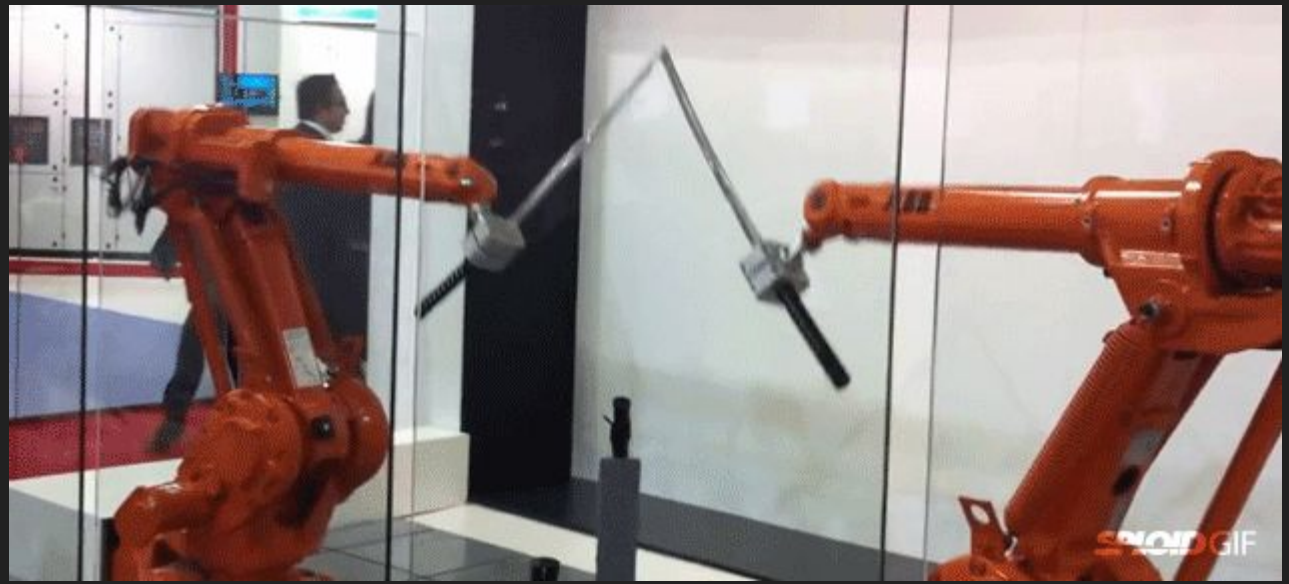
"After silence, that which comes nearest to expressing the inexpressible is music."
- Aldous Huxley.

Sound vs Music

loaded all at once
small in size
played many times

loaded piece by piece
large in size
only one piece of music ever active

Transform Hierarchy



"Lead, follow,
or get out
of the way."
- Laurence J. Peter.

```
void moveMario(float xAmount, float yAmount)
{
    position.x += xAmount;
    position.y += yAmount;
}
```

how should Mario know that he should move with the platform if standing on it?

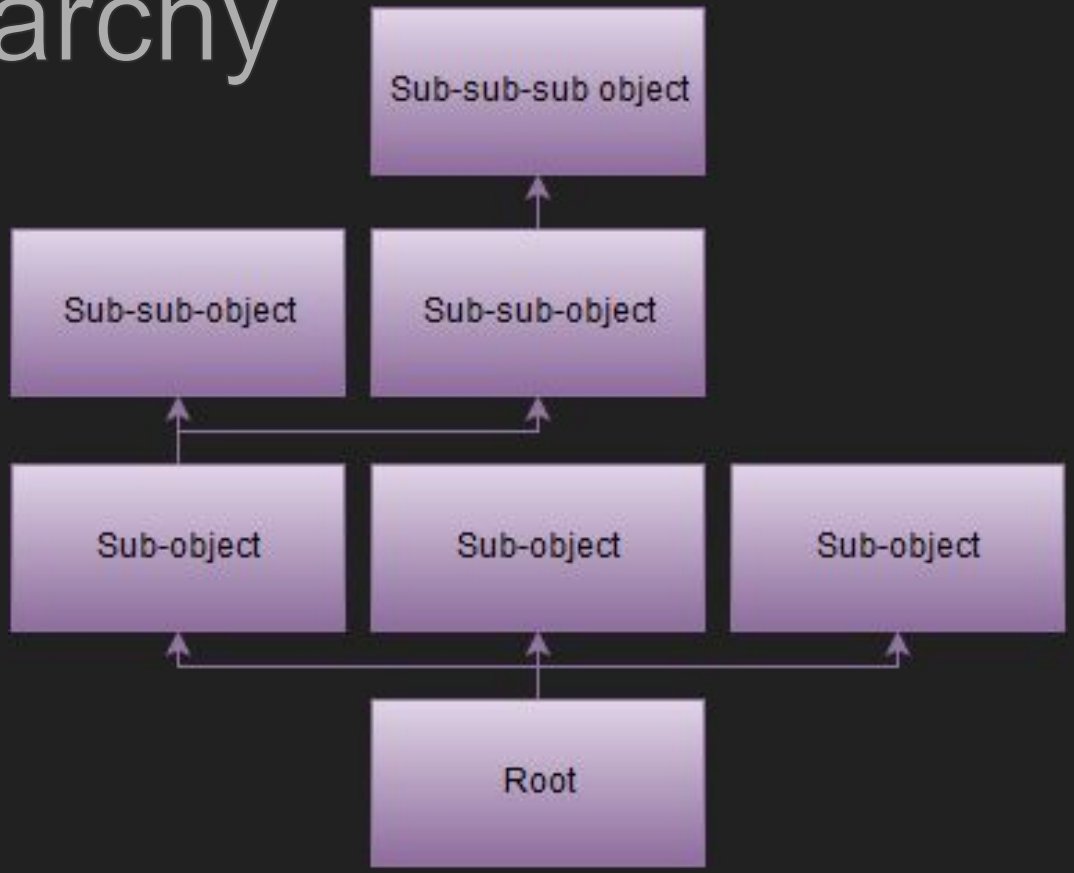
```
void moveMario(float xAmount, float yAmount)
{
    position.x += xAmount;
    position.y += yAmount;

    if(touchingPlatform())
    {
        position.x += platform.movingAmountX;
        position.y += platform.movingAmountY;
    }
}
```

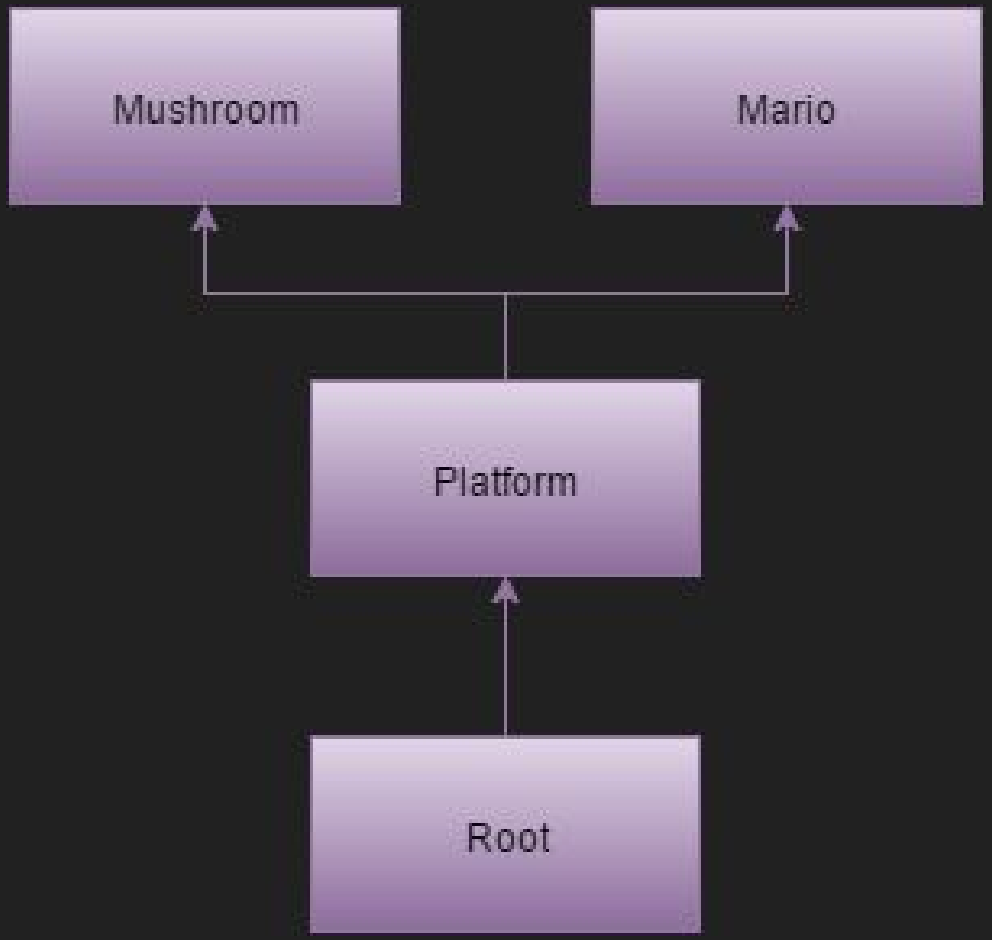

then this happens

```
void moveMario(float xAmount, float yAmount)
{
    position.x += xAmount;
    position.y += yAmount;

    for(int i = 0; i < platforms.length; ++i)
    {
        if(touchingPlatform(platforms[i]))
        {
            position.x += platforms[i].movingAmountX;
            position.y += platforms[i].movingAmountY;
        }
    }
}
```

```
void moveMushroom(float xAmount, float yAmount)
{
    ???
}
```

solution:
Transform Hierarchy




Mushroom
Mario
Platform
Root