GAME ENGINE

Source

- The Cherno
- GAMES 104: Modern Game Engine – Theory and Practice

Framework

- Basic Elements
  - Structure and layer…
- Rendering
- Animation
- Physics
- Gameplay
  - Event system, scripts system, graph driven
- Misc. Systems
  - Effects, navigation, camera…
- Tool set
  - C++ reflection, data scheme (reflection: complex
- Online gaming
  - Synchronization, consistency

Advanced tech:
Motion matching
Procedural content generation (PCG

Data-oriented programming (DOP
Job system

(UE5 amazing systems)
Lumen
Nanite

Layered Architecture of Game Engine

- Tool Layer (chain of editors
- Function Layer (make it visible, movable and playable
- Resource Layer (data and file
- Core Layer (swiss knife of game engine
- Platform Layer (launch on different platforms

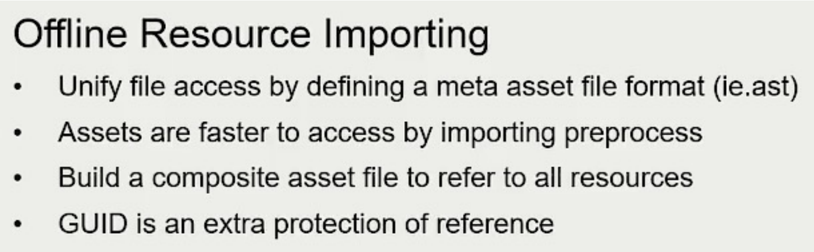
+Middleware and 3rd party libraries

Why:

- Decoupling and Reducing Complexity
- Response for Evolving Demands

Resource

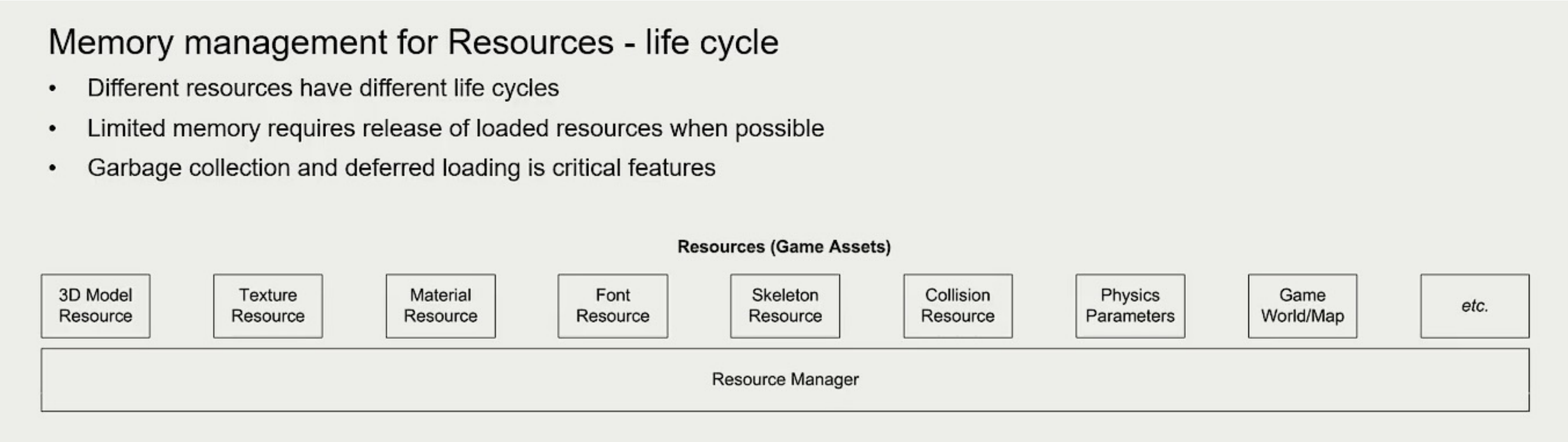
how to access my data



Offline Resource Importing

- Unify file access by defining a meta asset file format (ie.ast)
- Assets are faster to access by importing preprocess
- Build a composite asset file to refer to all resources
- GUID is an extra protection of reference

Manage asset life cycle



Memory management for Resources - life cycle

- Different resources have different life cycles
- Limited memory requires release of loaded resources when possible
- Garbage collection and deferred loading is critical features

Resources (Game Assets)

3D Model Resource

Texture Resource

Material Resource

Font Resource

Skeleton Resource

Collision Resource

Physics Parameters

Game World/Map

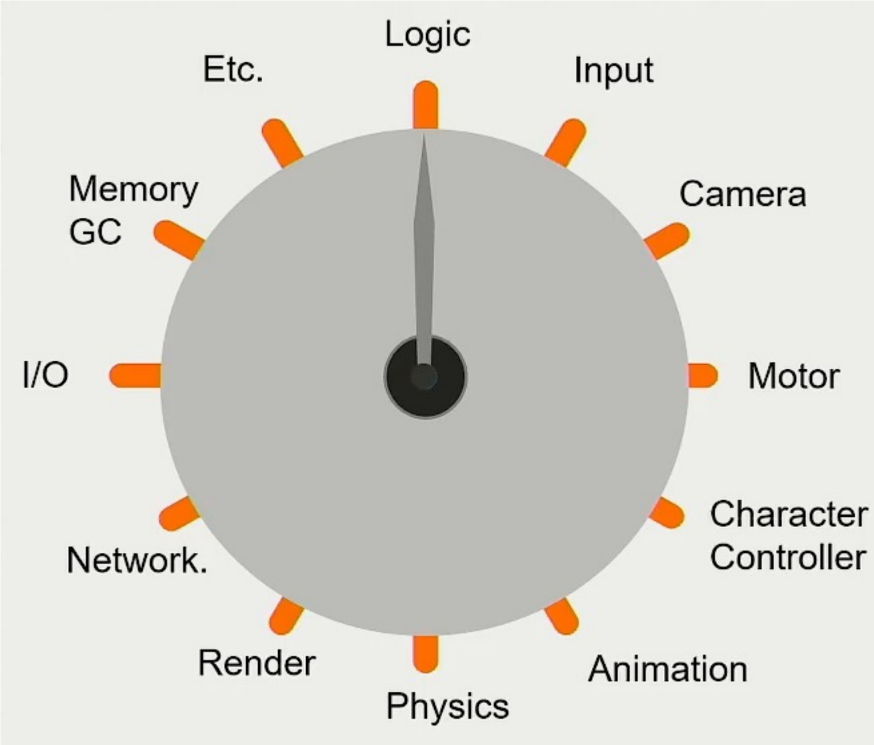
etc.

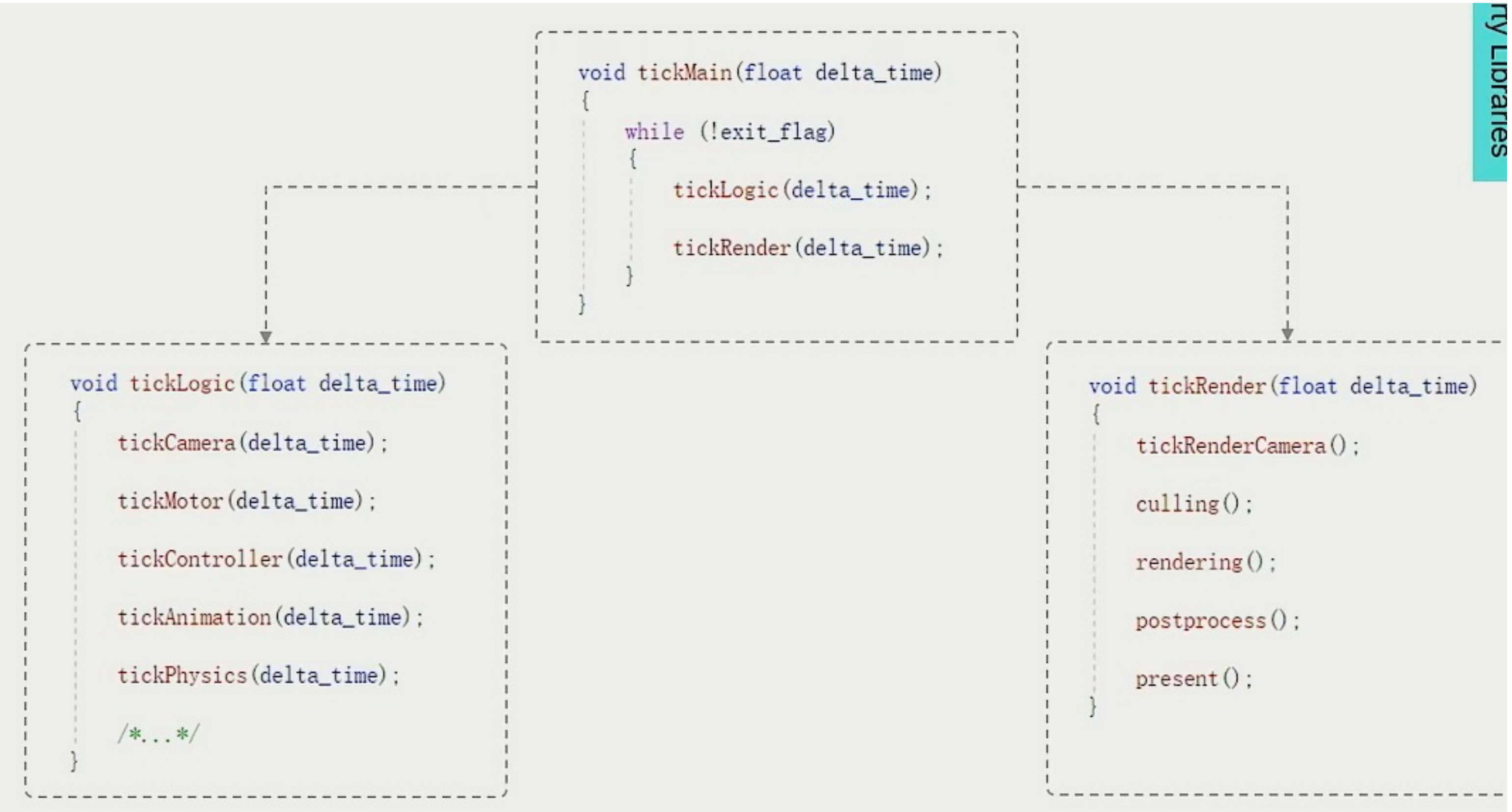
Resource Manager

Function

Heavy-duty hotchpotch
Multi-threading

Tick
--logic
--render





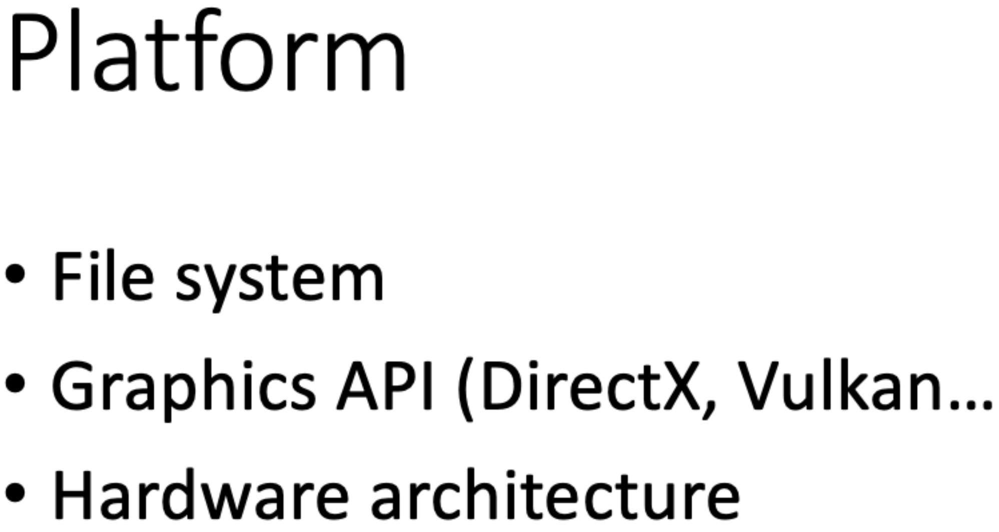
```
void tickMain(float delta_time)
{
    while (!exit_flag)
    {
        tickLogic(delta_time);

        tickRender(delta_time);
    }
}
```

```
void tickLogic(float delta_time)
{
    tickCamera(delta_time);

    tickMotor(delta_time);

    tickController(delta_time);

    tickAnimation(delta_time);

    tickPhysics(delta_time);

    /*...*/
}
```

```
void tickRender(float delta_time)
{
    tickRenderCamera();

    culling();

    rendering();

    postprocess();

    present();
}
```

Core

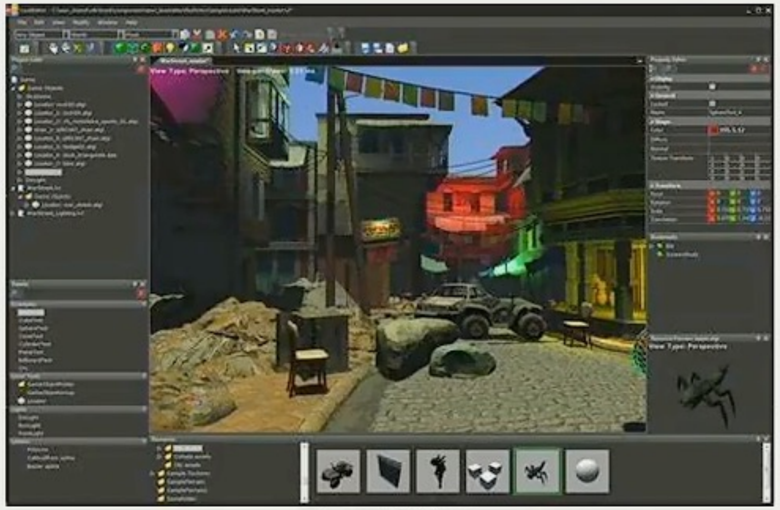
- Math Library
- Math Efficiency(quick and dirty hacks, SIMD
- Data structure and containers (customized STL
- Memory management (memory pool.



Platform

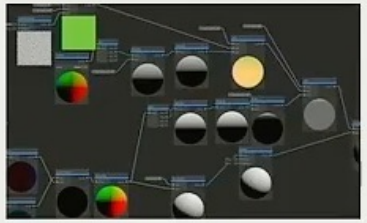
- File system
- Graphics API (DirectX, Vulkan…
- Hardware architecture

Tool



Level Editor

Logical Editor



Shader Editor

Animation Editor

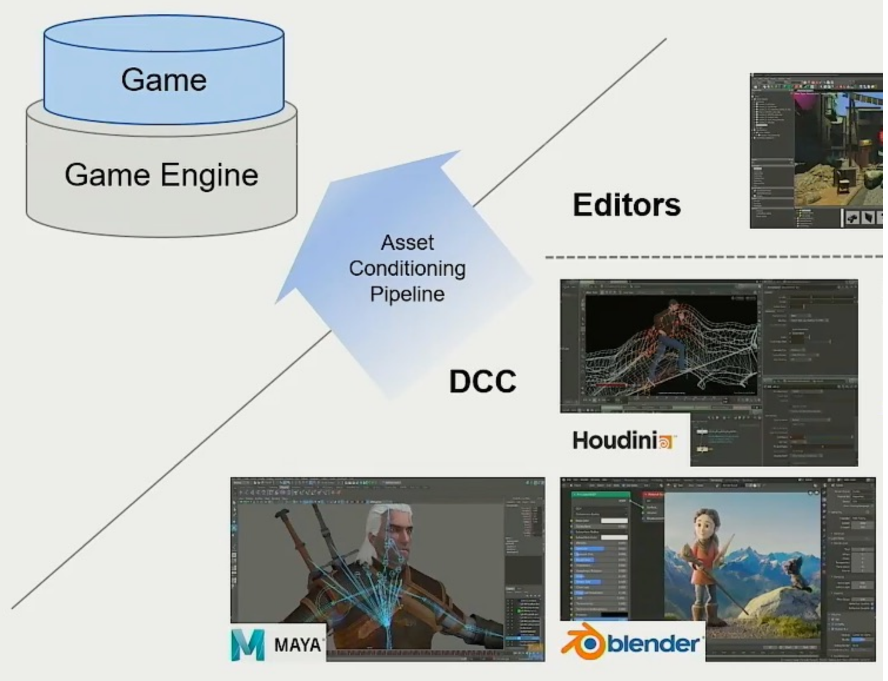
UI Editor

Allow everyone to create game

Unleash the Creativity

- Build upon game engine
- Create, edit and exchange game play assets

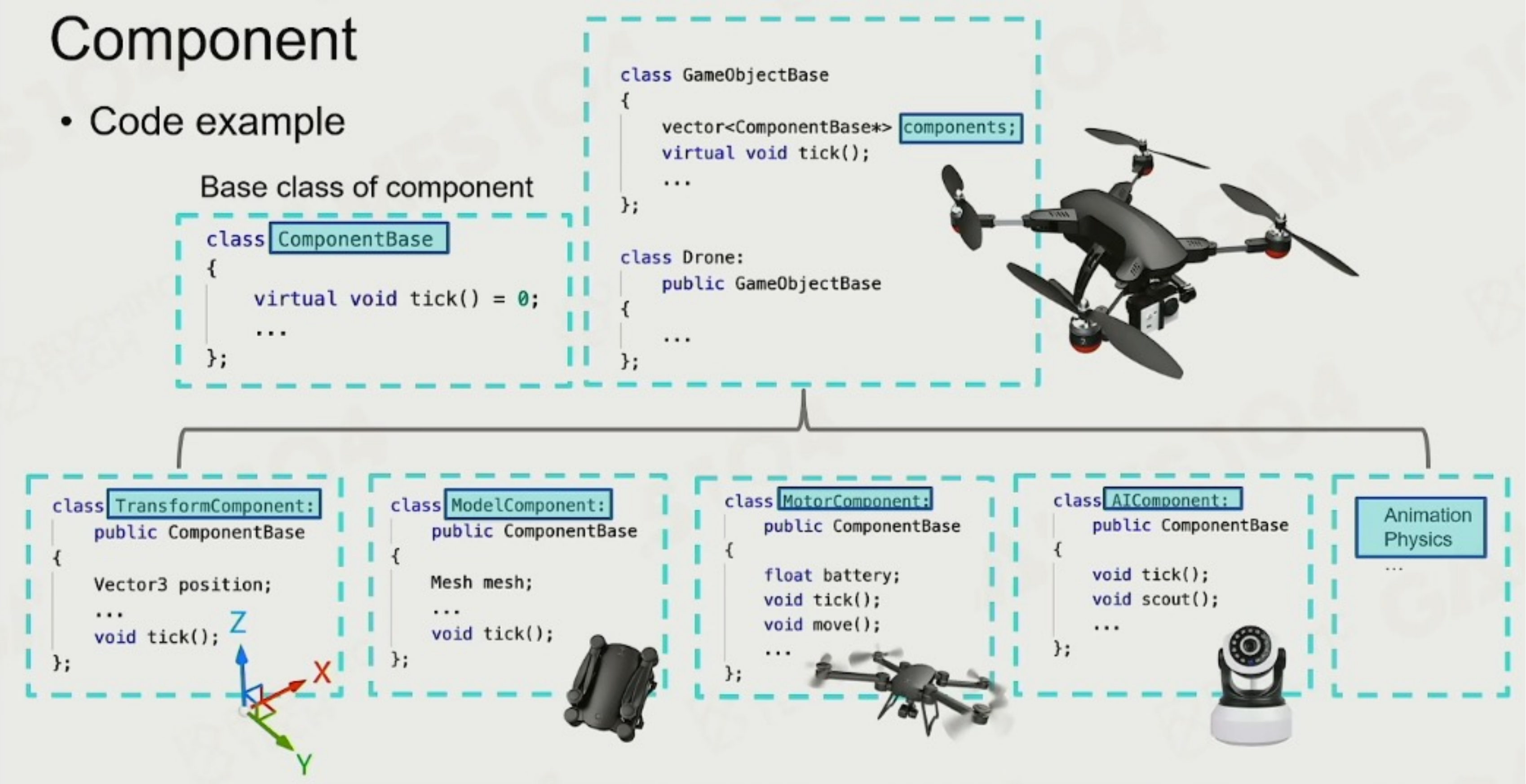
Flexible of coding languages



- Digital content creation (DCC

How to Build a Game World

- Game Object (GO)
  - Name, property, behavior.
  - Inheritance -> component base



Component

- Code example

Base class of component

```
class ComponentBase
{
    virtual void tick() = 0;
    ...
};
```

```
class GameObjectBase
{
    vector<ComponentBase*> components;
    virtual void tick();
    ...
};

class Drone:
    public GameObjectBase
{
    ...
};
```

```
class TransformComponent:
    public ComponentBase
{
    Vector3 position;
    ...
    void tick();
};
```

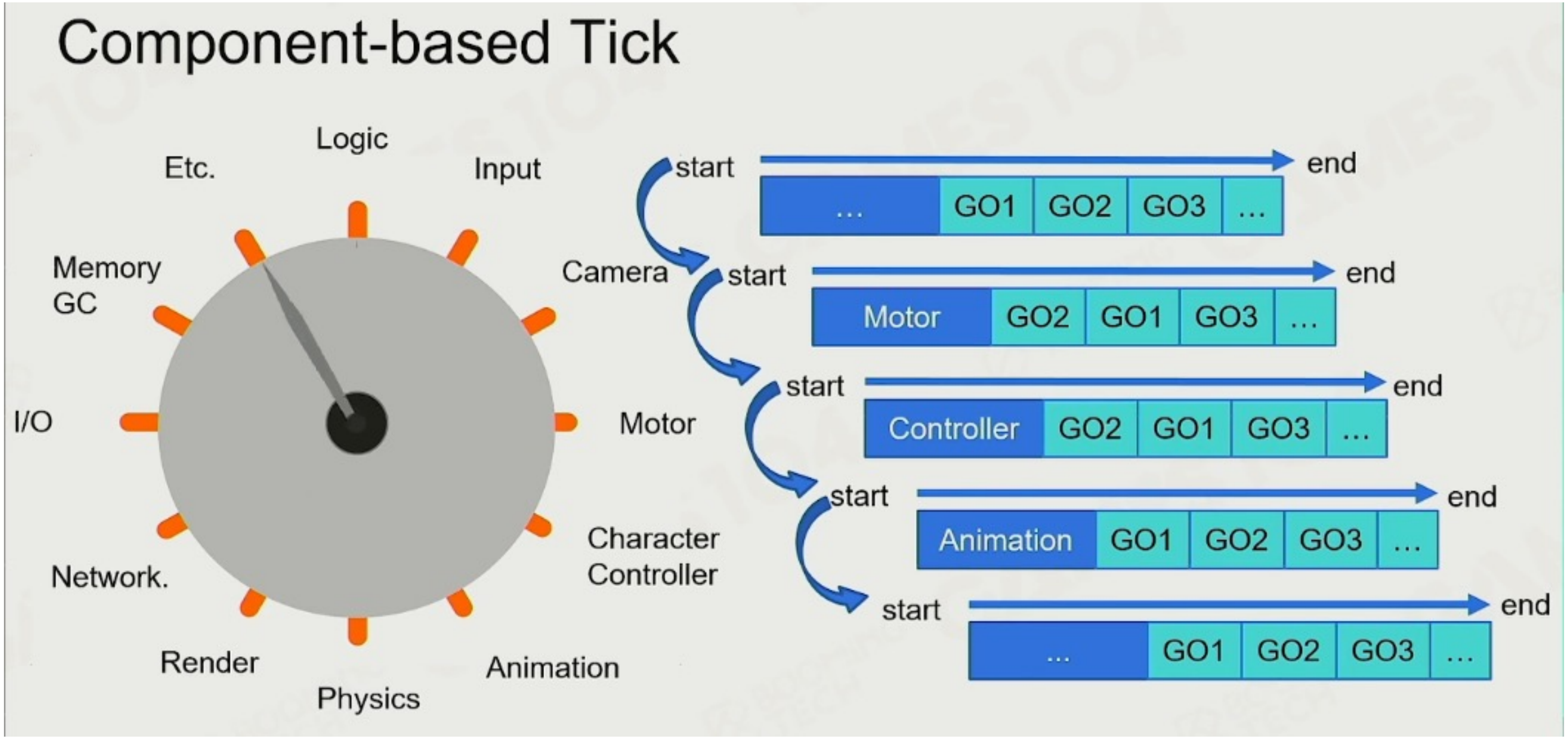
```
class ModelComponent:
    public ComponentBase
{
    Mesh mesh;
    ...
    void tick();
};
```

```
class MotorComponent:
    public ComponentBase
{
    float battery;
    void tick();
    void move();
    ...
};
```

```
class AIComponent:
    public ComponentBase
{
    void tick();
    void scout();
    ...
};
```

Animation
Physics
...

Pipeline – batch process



Component-based Tick

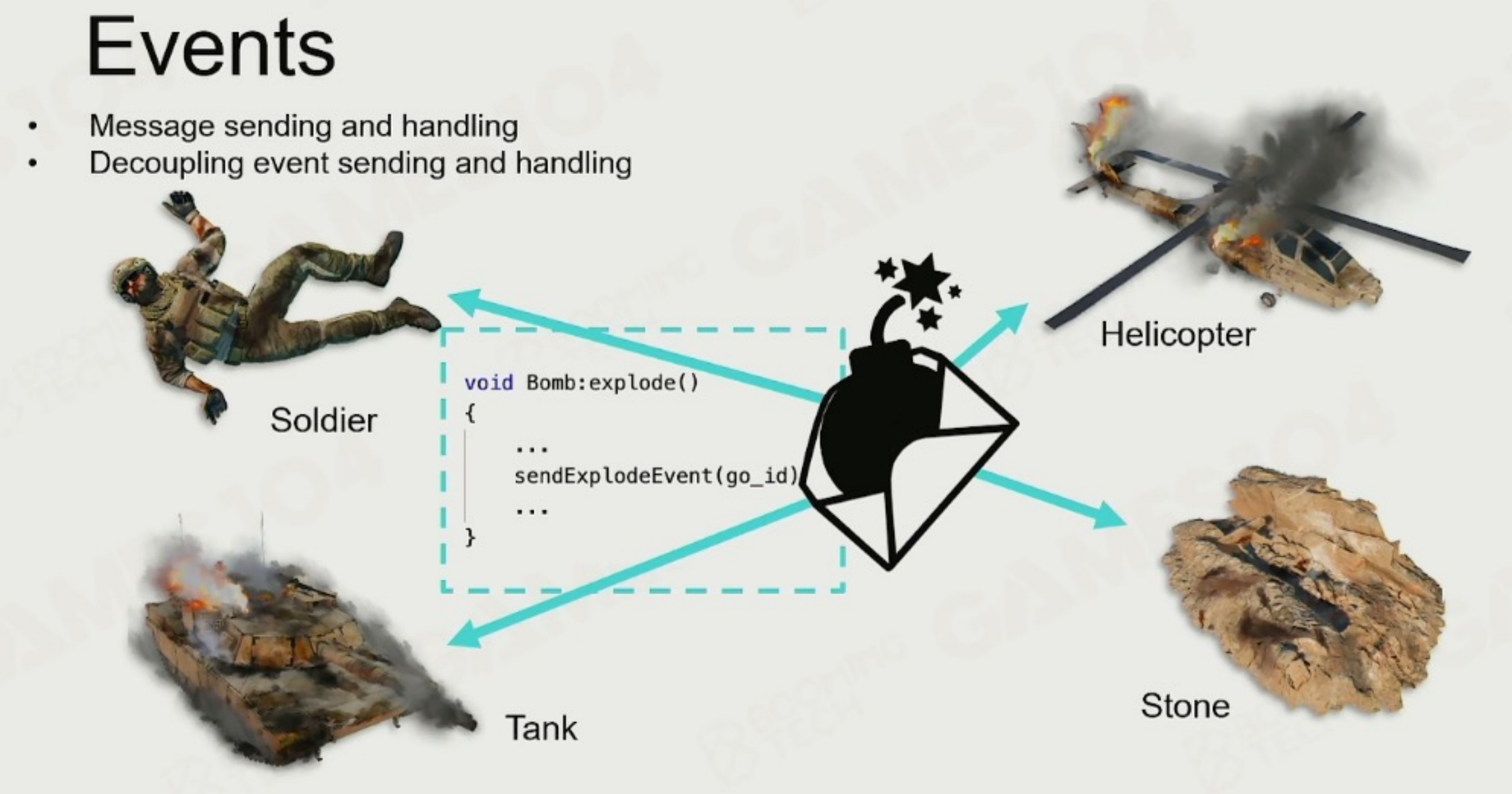
start → end: … | GO1 | GO2 | GO3 | …

start → end: Motor | GO2 | GO1 | GO3 | …

start → end: Controller | GO2 | GO1 | GO3 | …

start → end: Animation | GO1 | GO2 | GO3 | …

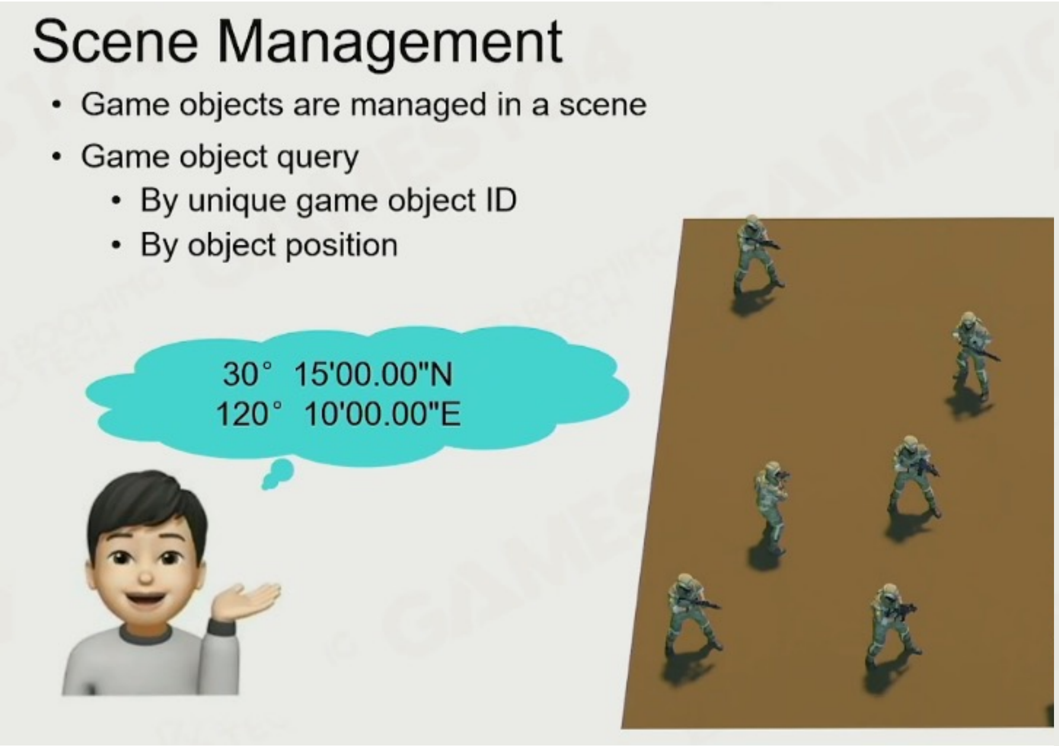
start → end: … | GO1 | GO2 | GO3 | …



Events

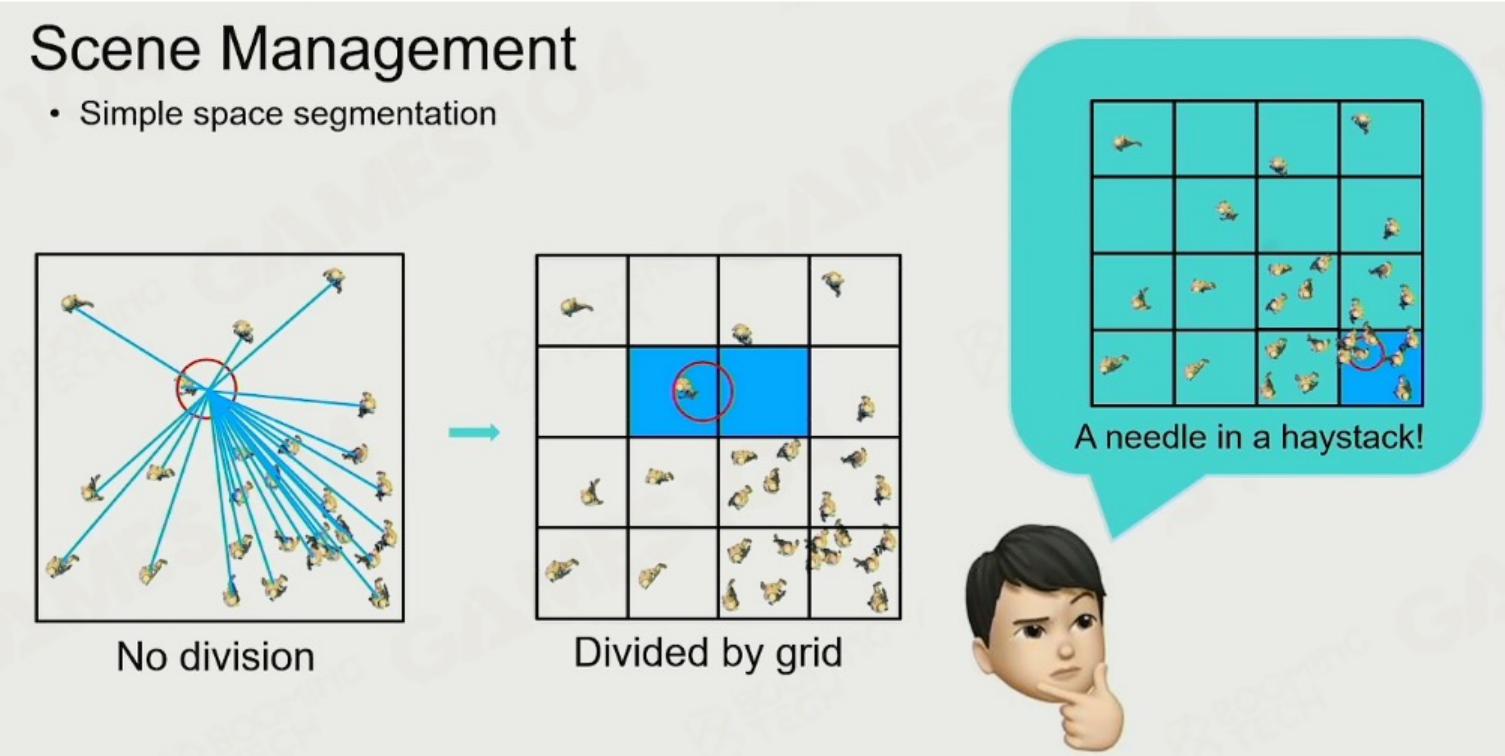
- Message sending and handling
- Decoupling event sending and handling

```
void Bomb:explode()
{
    ...
    sendExplodeEvent(go_id)
    ...
}
```



Scene Management

- Game objects are managed in a scene
- Game object query
  - By unique game object ID
  - By object position

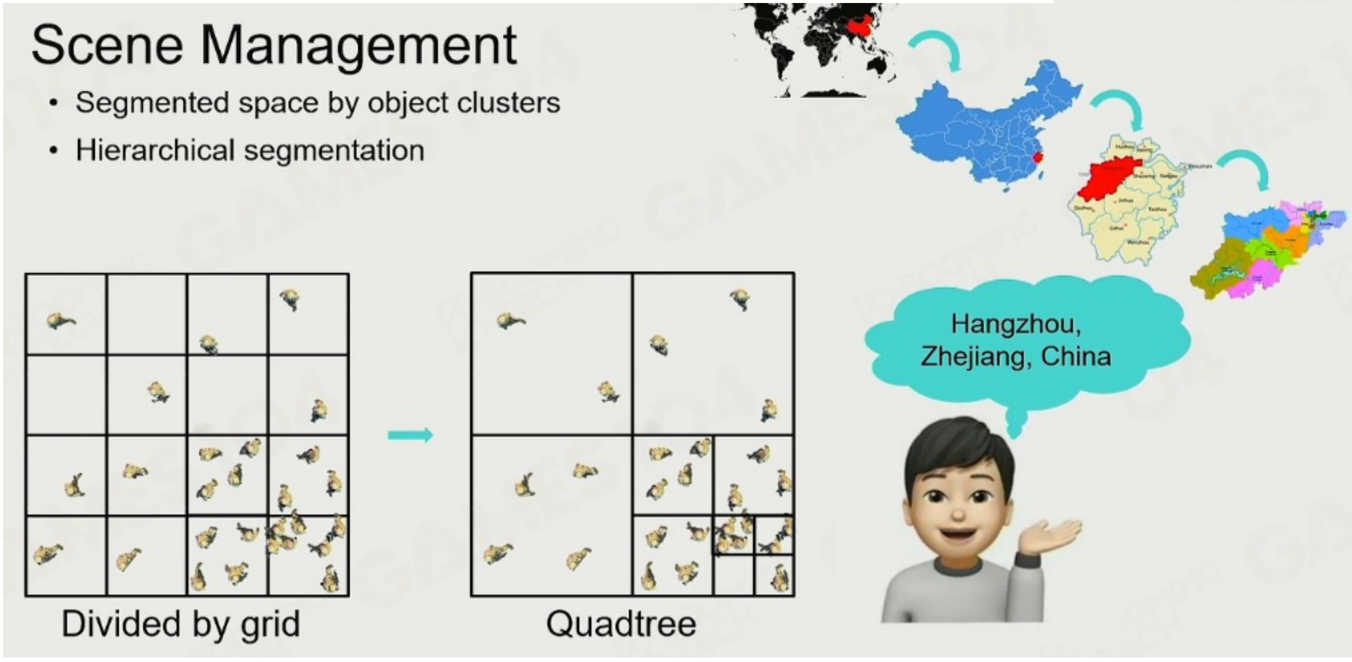


Scene Management

- Simple space segmentation

No division

Divided by grid

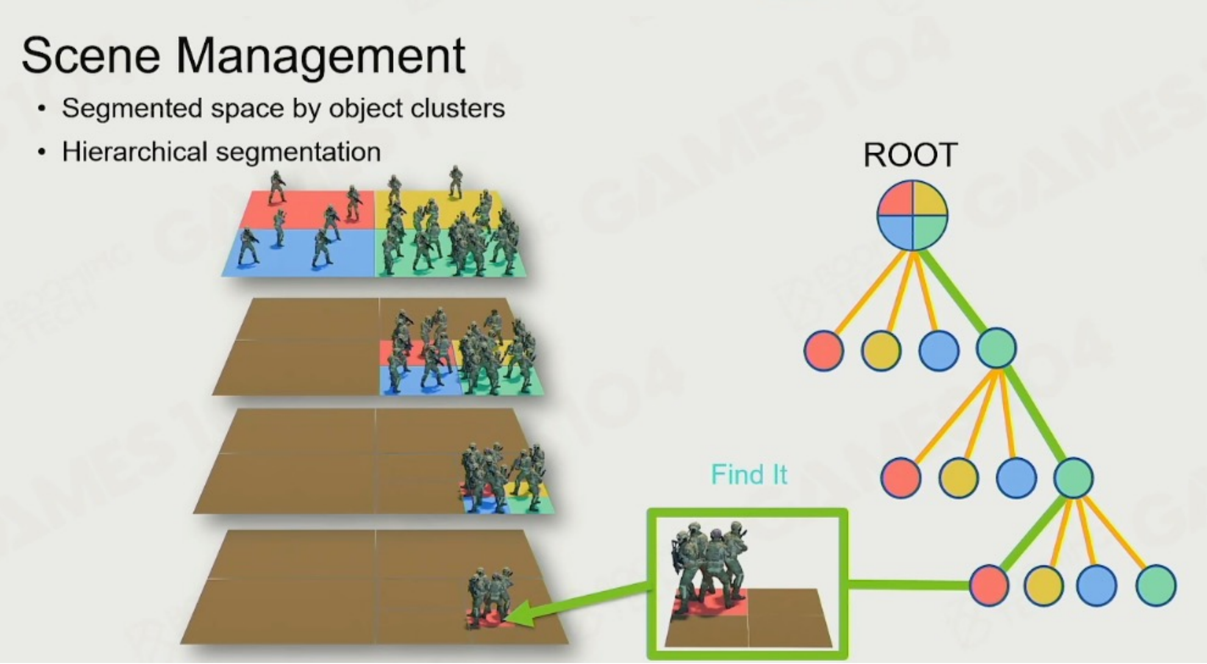


Scene Management

- Segmented space by object clusters
- Hierarchical segmentation

Divided by grid

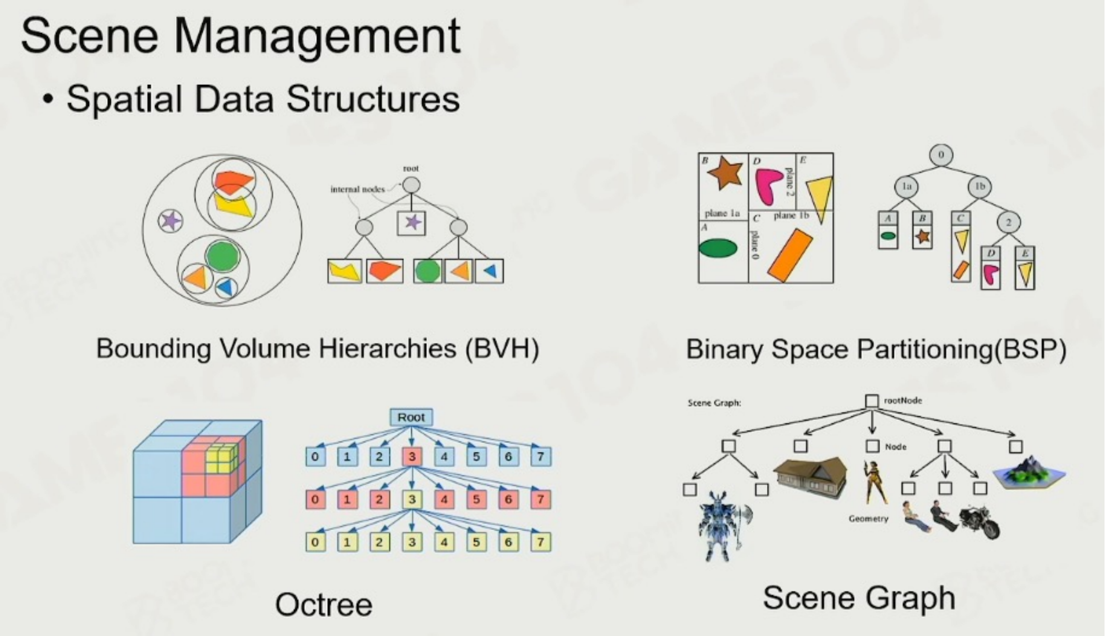
Quadtree



Scene Management

- Segmented space by object clusters
- Hierarchical segmentation

Bounding box



Scene Management

- Spatial Data Structures

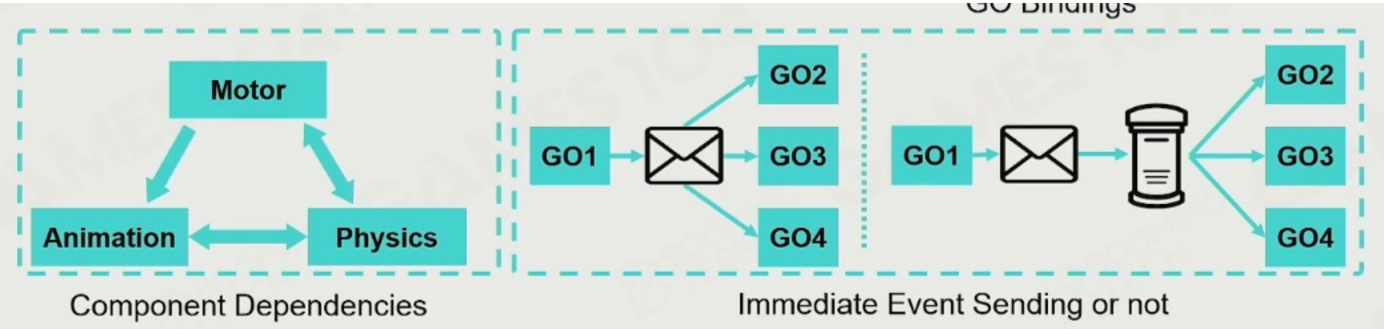
Bounding Volume Hierarchies (BVH)

Binary Space Partitioning(BSP)

Octree

Scene Graph

parallel



GO Bindings

Component Dependencies

Immediate Event Sending or not

Designing

- Entry point
- Application layout
- Window layout
    - Input
    - ->Events
- Renderer
- Render API abstraction
- Debugging support
- Scripting language
- Memory systems
- Entity-component systems (ECS)
- Physics
- File I/O, VFS
- Build System

Game engine is actually dll library
In a solution, add two projects engine(dll) and sandbox(exe)
Sandbox add reference to engine. (link)