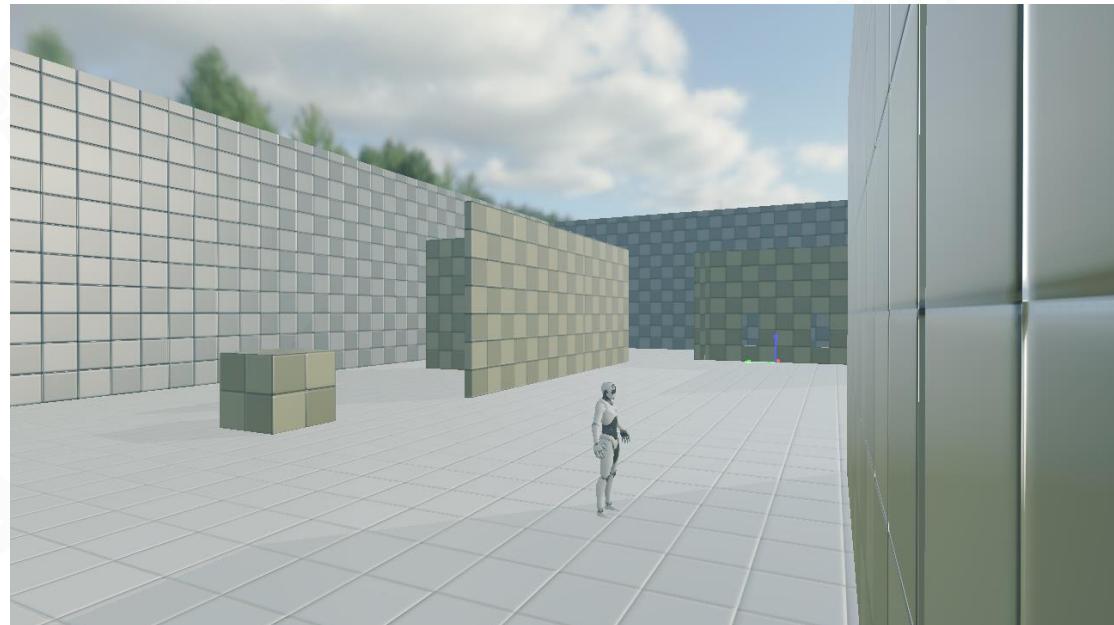


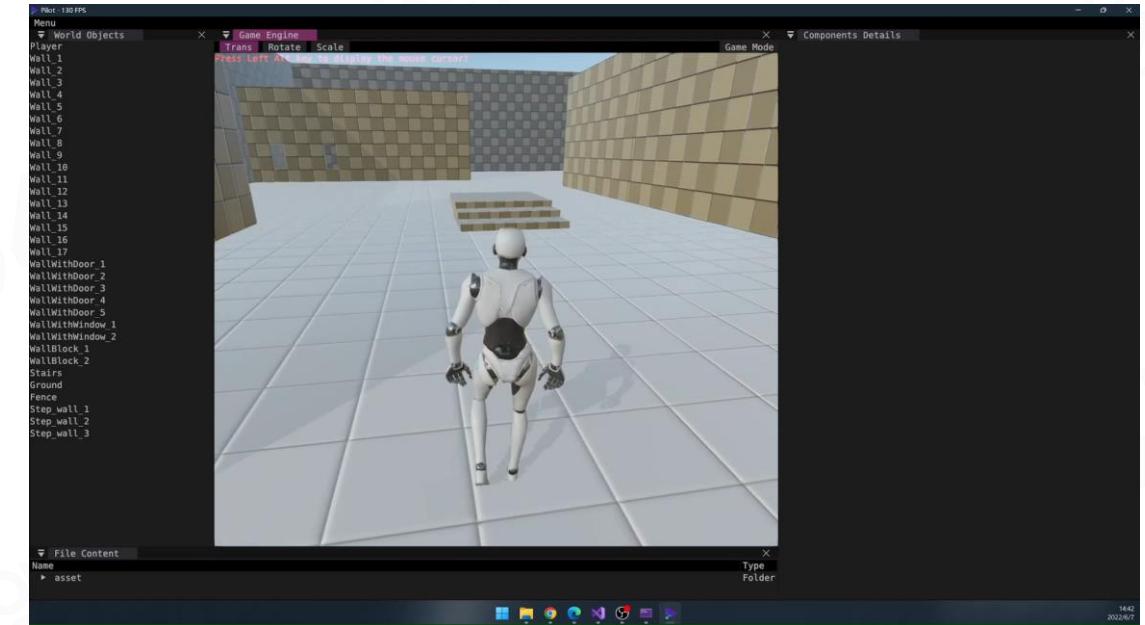


## Homework Submission Extended

- Will extend Homework #2 and Homework #3 deadline to Aug 31<sup>th</sup>



Homework #2 : Rendering



Homework #3 : Animation and Physics



## Voice from Community

- Some submissions are reported lost
- We've tested the submission system after received the reports
- We noticed one critical step may easily be omitted
- We have highlighted the critical step and updated our submission guide
- Please refer:  
[https://cdn.boomingtech.com/games104\\_static/upload/GAMES104\\_SmartChair\\_Submission\\_Guide.pdf](https://cdn.boomingtech.com/games104_static/upload/GAMES104_SmartChair_Submission_Guide.pdf), Page 9

点击“提交”（想要修改提交的文件可以点击“修改”）  
注意：此步骤非常关键，否则作业不会被上传到系统，依然会显示“未提交”。

思澈系统新表单格式测试 [ID: 11]

状态	未提交	请确认表单后提交。
file upload title	[homework_submit_test.txt.txt]	
<a href="#">修改</a>	<a href="#">提交</a>	



## Q&A about Piccolo Engine

- Q1: Is script system in Piccolo Engine's roadmap? Which script language will Piccolo Engine support?
- Q2: Why did Piccolo Engine use CMake as meta build system instead of XMake?
- Q3: Why some source code will be recompiled even no code is modified?



Lecture 15

# Gameplay System

Gameplay Complexity and Building Blocks



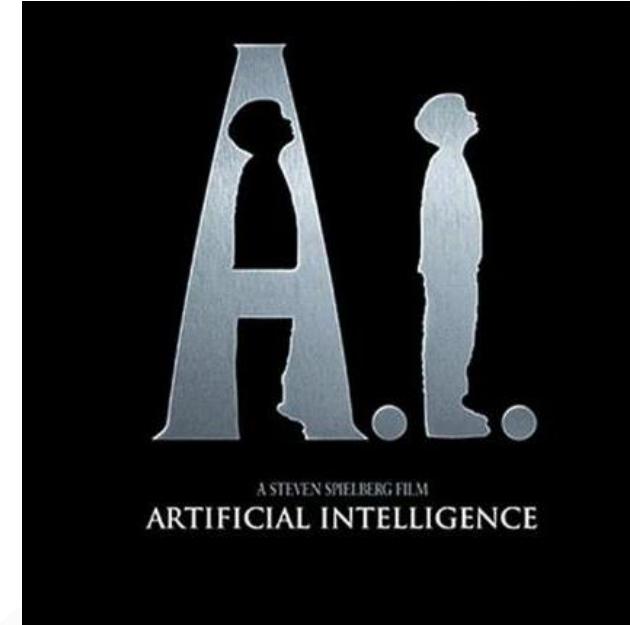
## Outline of Gameplay System

01.

### Gameplay Complexity and Building Blocks

- Overview
- Event Mechanism
- Script System
- Visual Script
- Character, Control and Camera

02.





## Challenges in GamePlay (1/3)

Cooperation among multiple systems



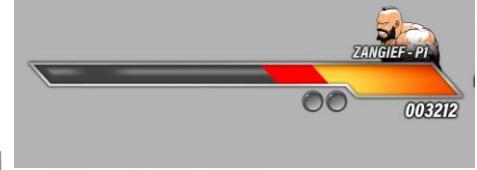
Animation



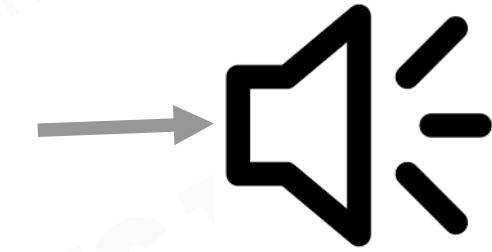
Effect



*Street Fighter*  
Attack Feedback



UI



Audio



Interface Devices



## Challenges in GamePlay (2/3)

Diversity of game play in the same game



*The Witcher 3: Wild Hunt*  
Combat Gameplay



*The Witcher 3: Wild Hunt*  
Card-playing Mechanic

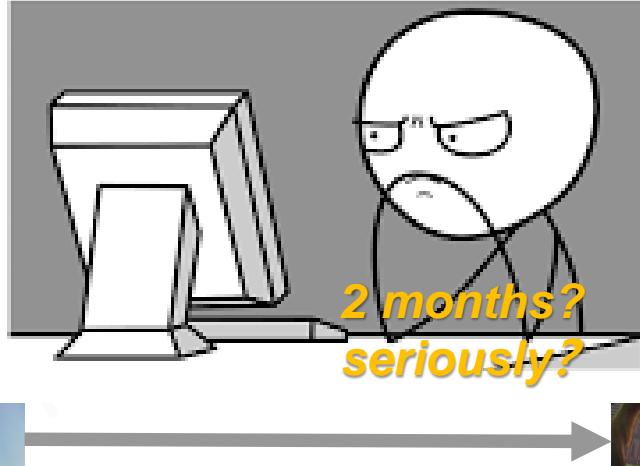


## Challenges in GamePlay (3/3)

Rapid iteration



*Fortnite: Save the World*  
TPS,tower defense, survival



*Fortnite: Battle Royale*  
battle royale

Epic acknowledged that within the Fortnite fundamentals, they could also do a battle royale mode, and rapidly developed their own version atop Fortnite in about **two months**.



## GamePlay

---

### Event Mechanism



# Let Objects Talk



Soldier



Tank

```
void Bomb::explode()
{
    ...
    switch(go_type)
    {
        case GoType.human_type:
            /* process soldier */
        case GoType.drone_type:
            /* process drone */
        ...
        case GoType.tank_type:
            /* process tank */
        ...
        case GoType.stone_type:
            /* process stone */
        ...
        default:
            break;
    }
}
```



Helicopter

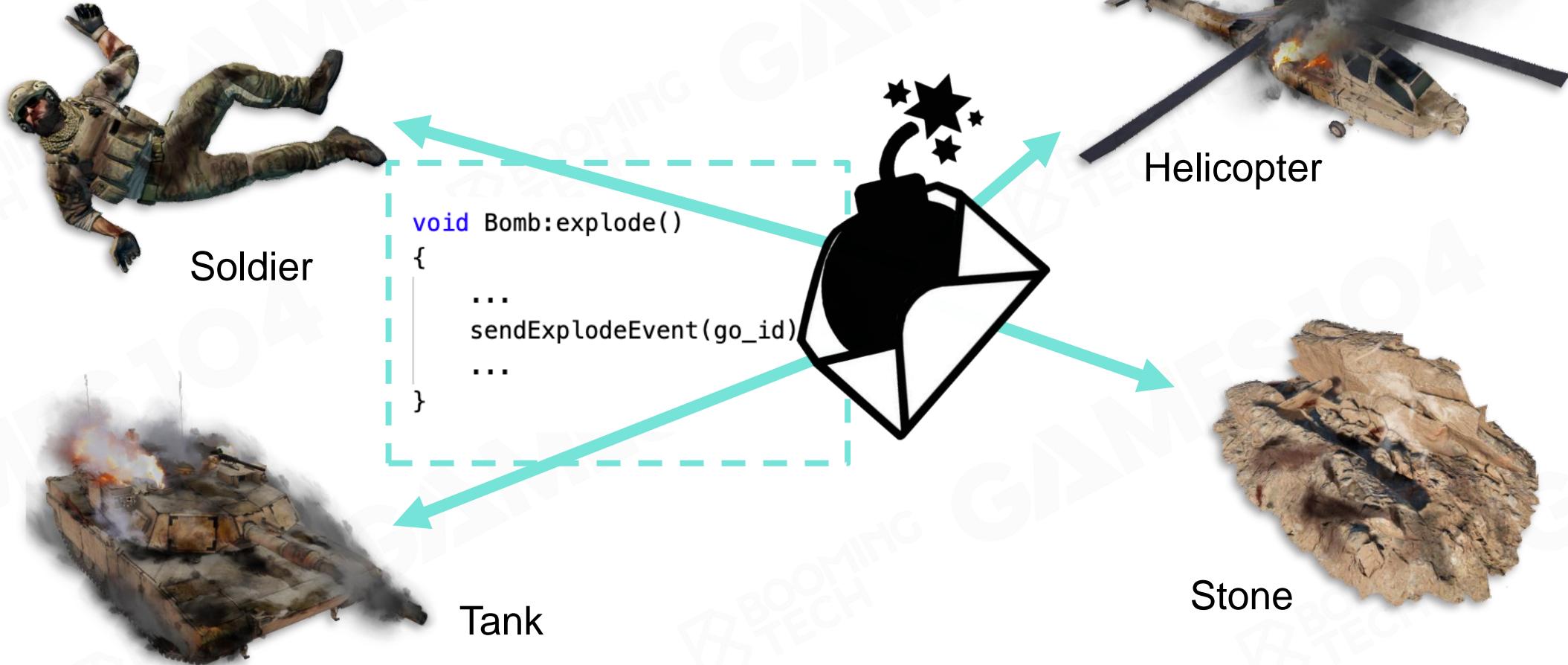


Stone



## Event/Message Mechanism

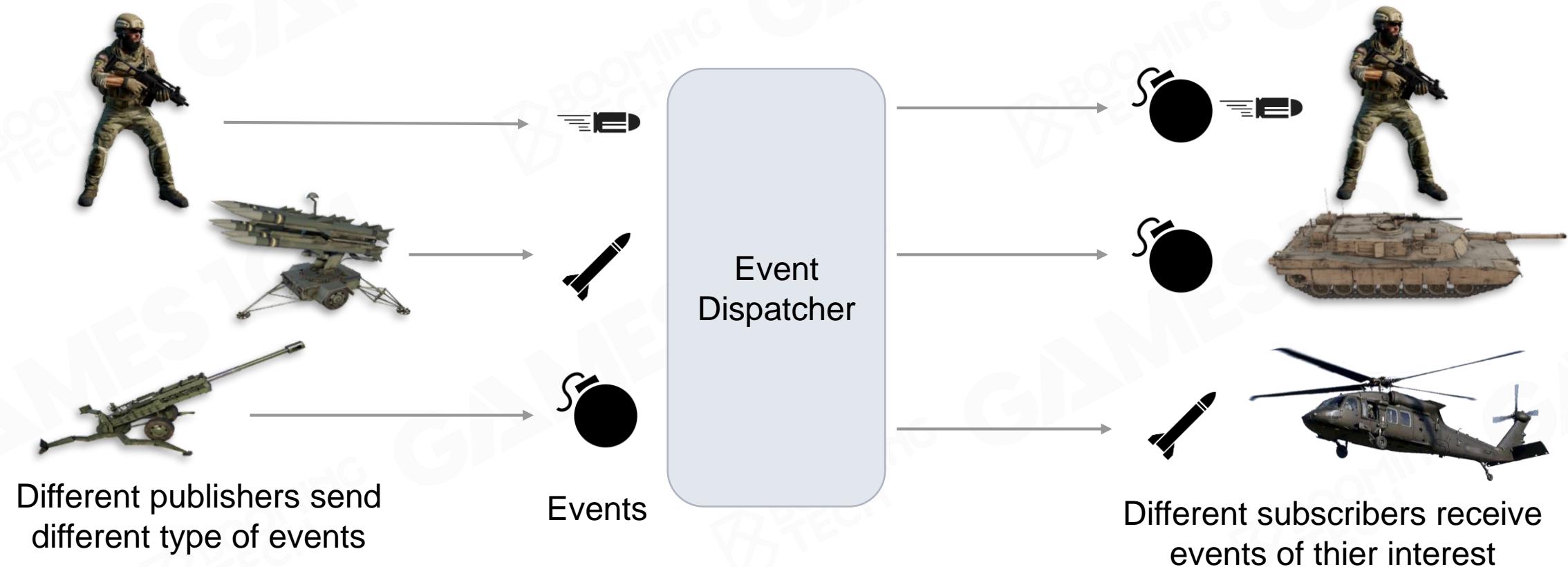
- Abstract the world communication to messages
- Decoupling event sending and handling





## Publish-subscribe Pattern

- Publisher categorizes published messages (events) into classes
- Subscriber receive messages (events) that are of interest without knowledge of which publishers

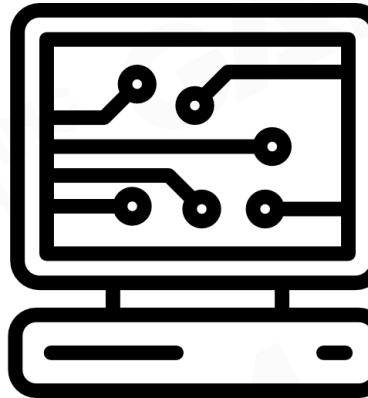




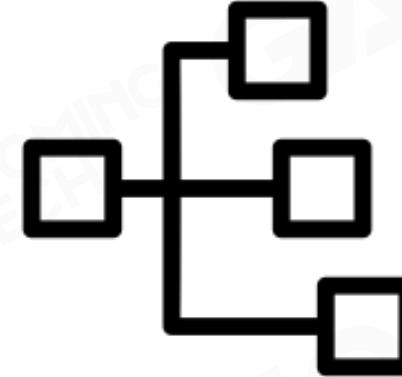
## 3 Key Components of Publish-subscribe Pattern



Event Definition



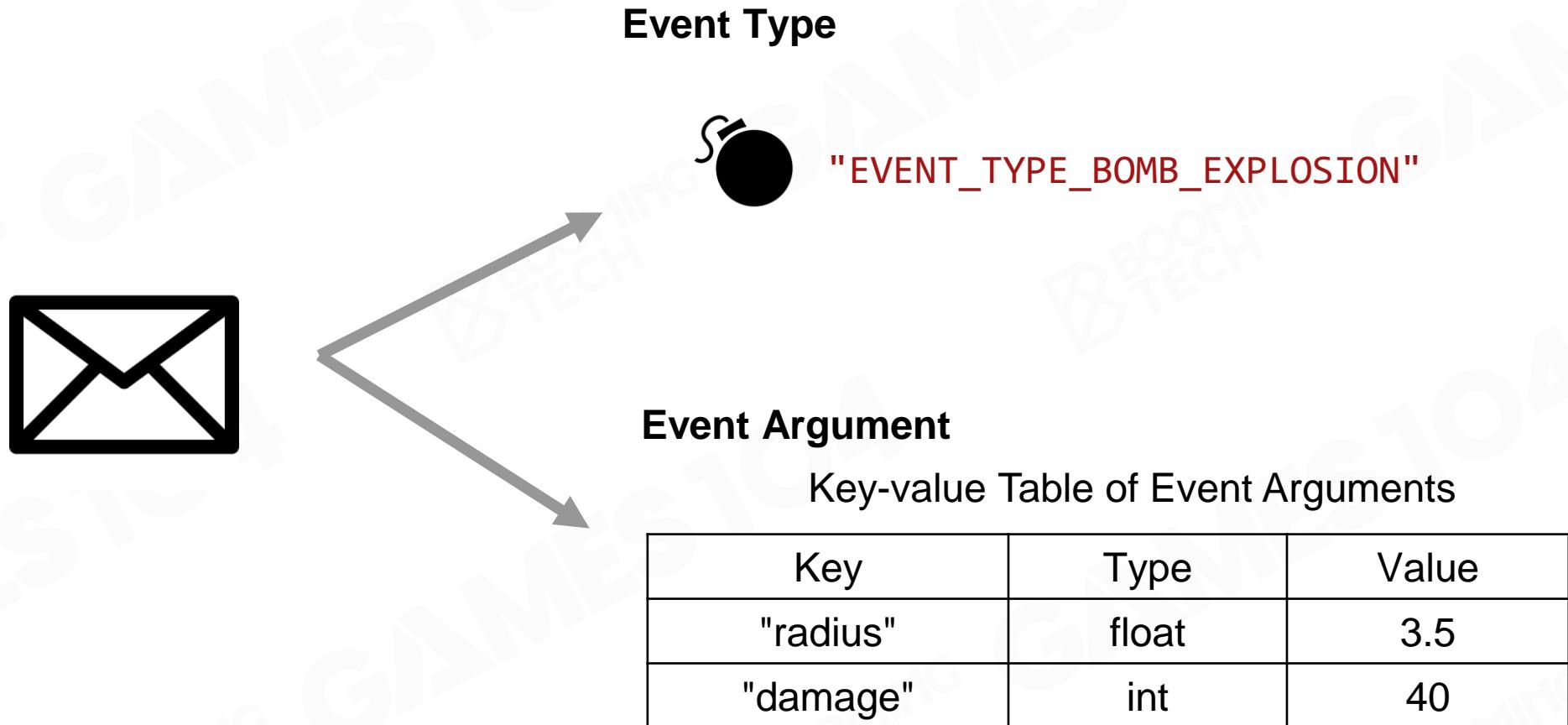
Callback Registration



Event Despatching



## Event Definition





# Event Definition

## Type and Arguments

```
class BombExplosionEvent : public Event
{
    Point m_center;
    float m_damage;
    float m_radius;
};
```

```
class BulletHitEvent : public Event
{
    float m_final_speed;
    float m_damage;
};
```

```
class MissileHitEvent : public Event
{
    float m_damage;
};
```

... ...

Impossible for  
hardcode



# Event Definition

## Type and Arguments

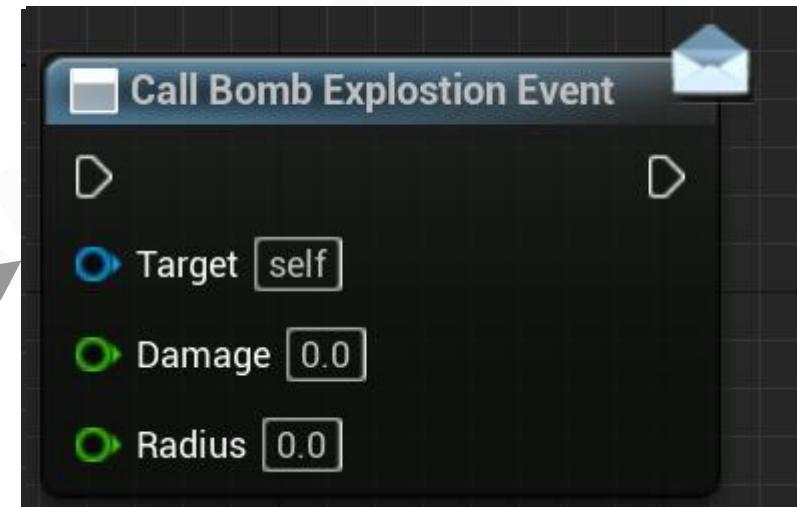
- Editable

```
class BombExplosionEvent
    : public Event
{
    float    m_damage;
    float    m_radius;
};
```

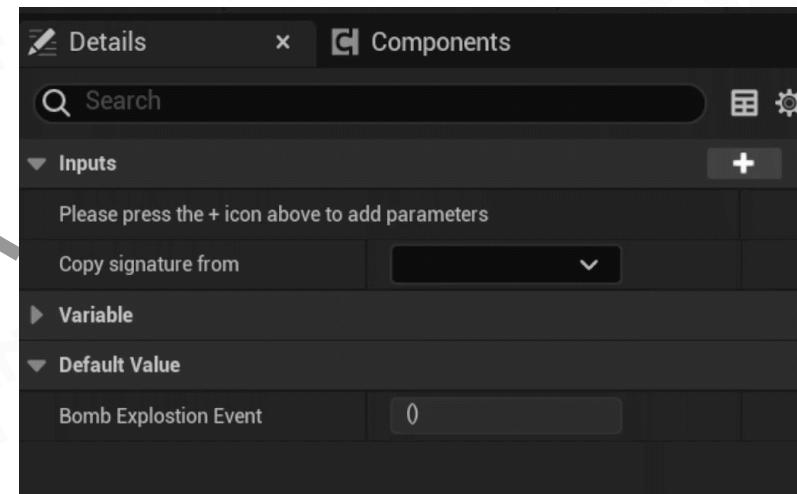
Hardcode

Reflection

Code Generator



View in Editor



Editable

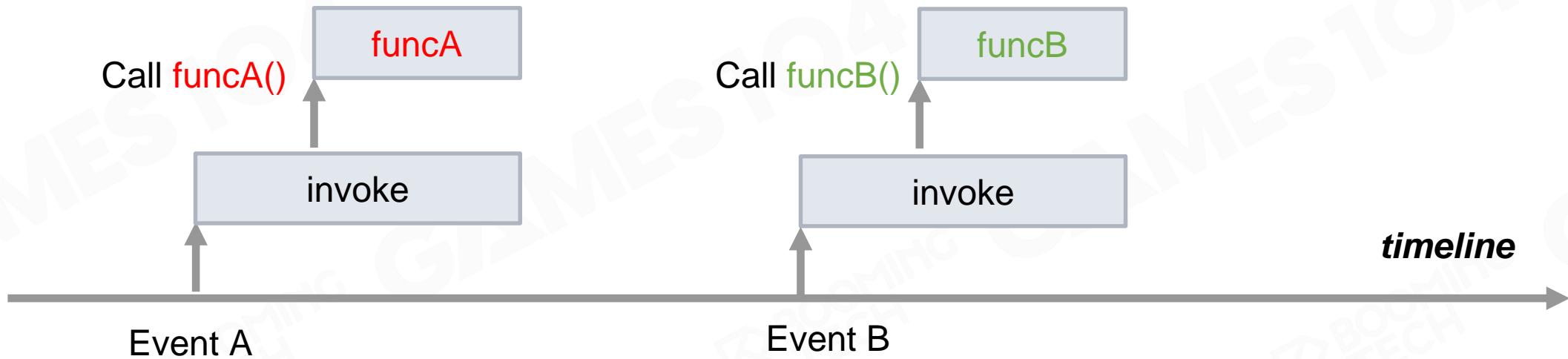


# Callback Registration

## Callback (function)

- Any reference to executable code that is passed as an argument to another piece of code

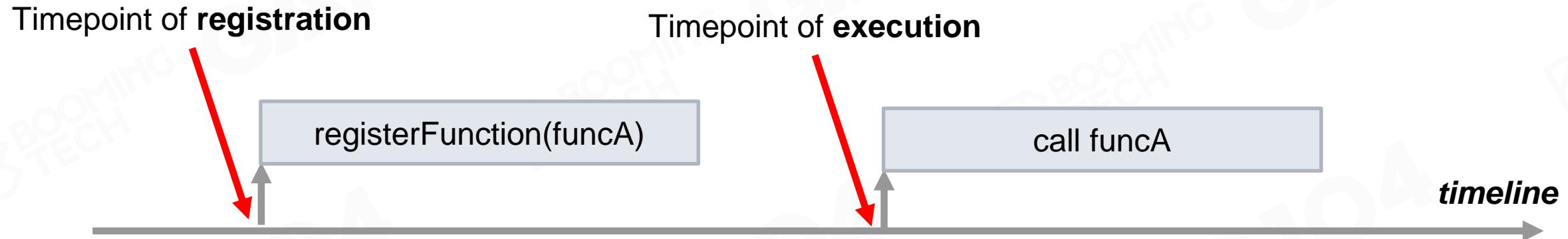
```
function invoke(call_back_function)
{
    // ...
    call_back_function()
}
```





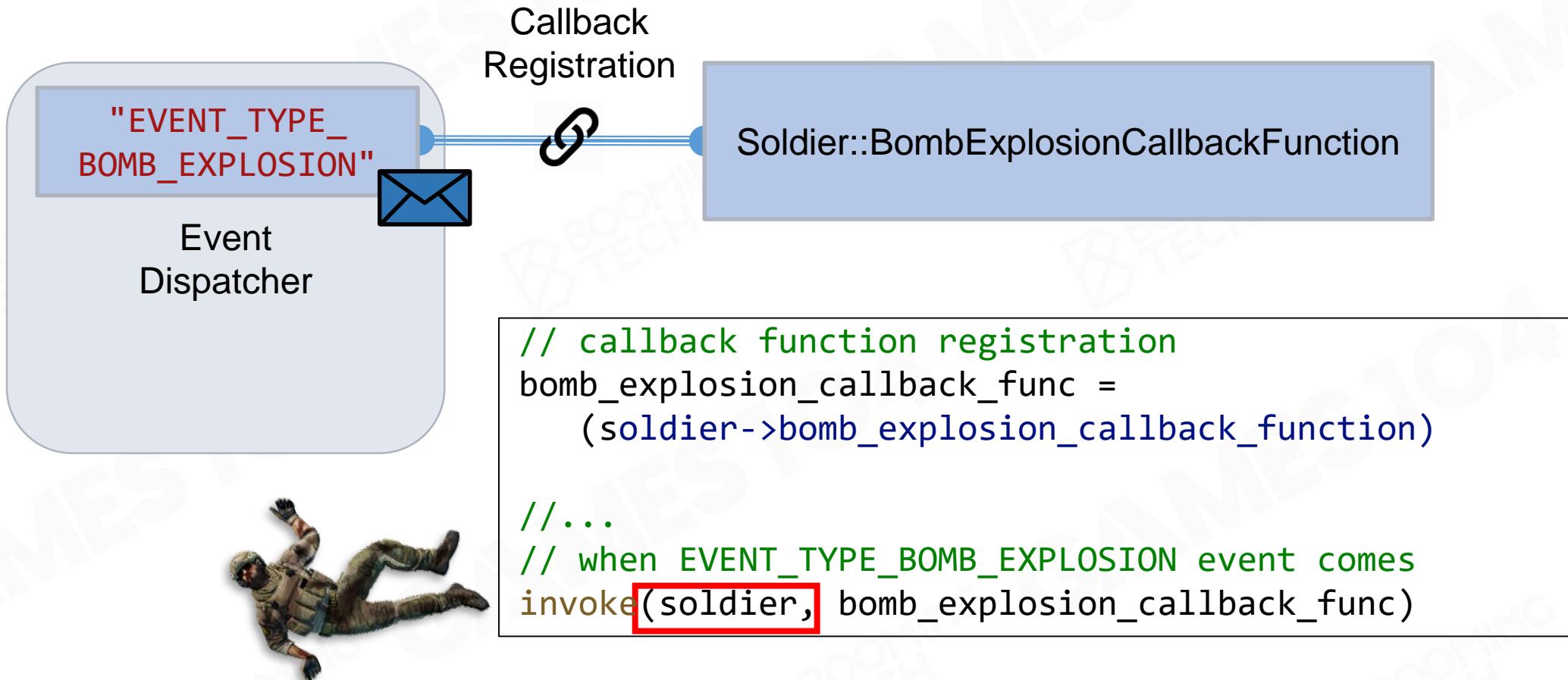
# Object Lifespan and Callback Safety

Time points of registration and execution differs





## Object Lifespan and Callback Safety

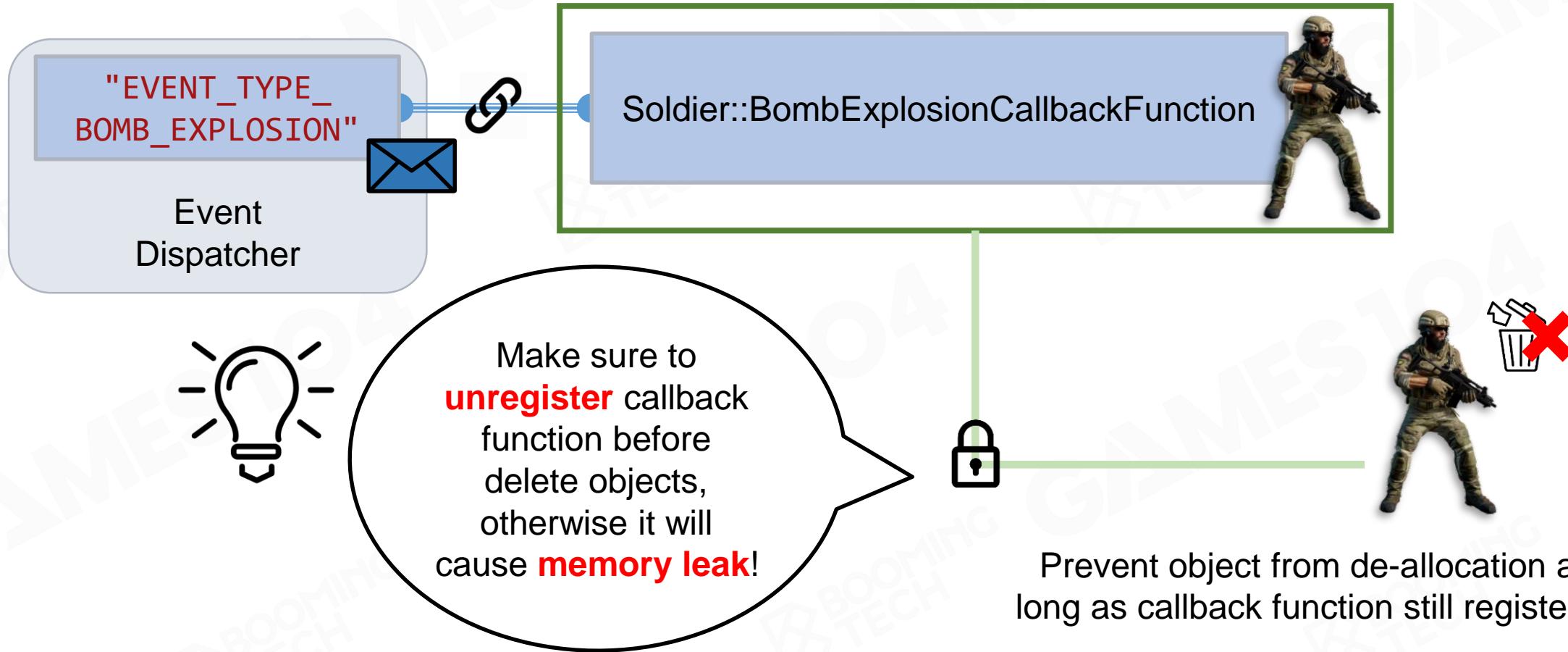


What if soldier already destroyed?

**Wild pointer! Crashed!**

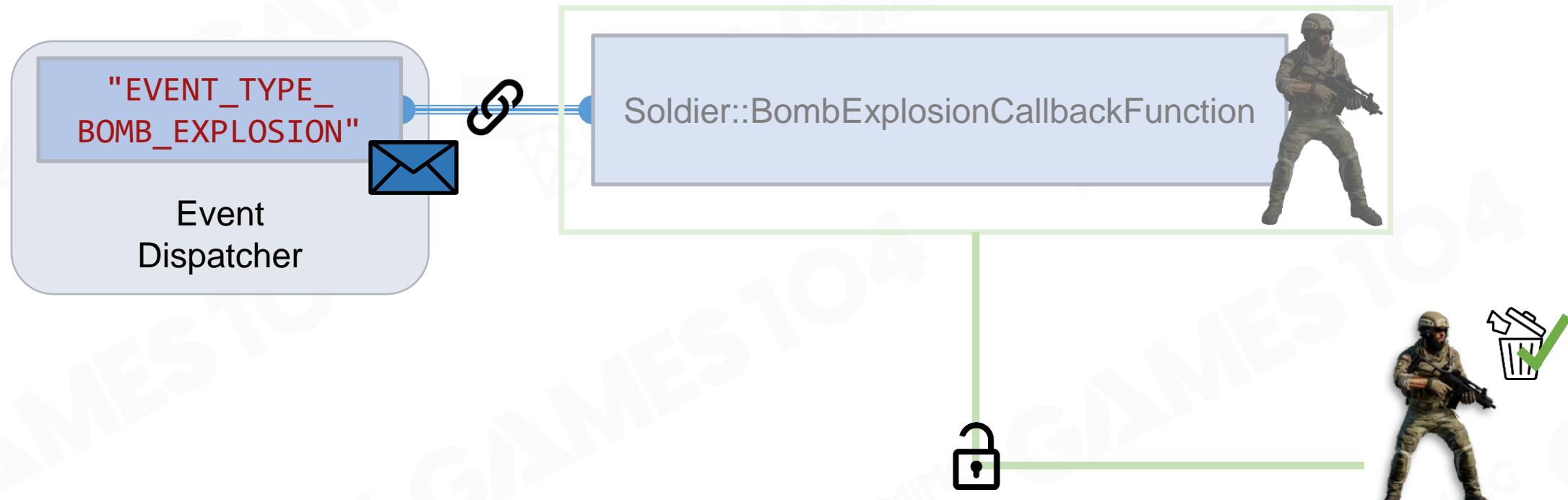


## Object Strong Reference





## Object Weak Reference

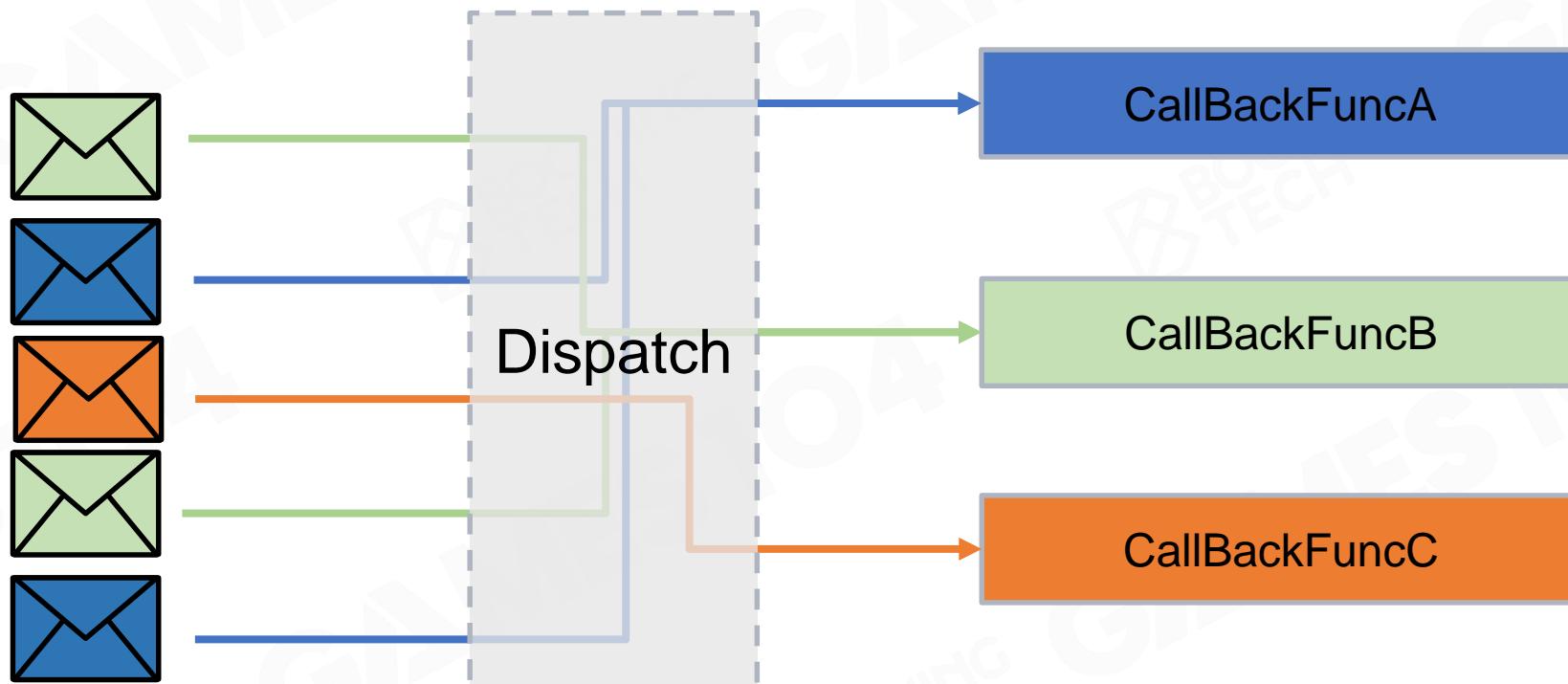


Object could be de-allocated, and will check callback function if valid



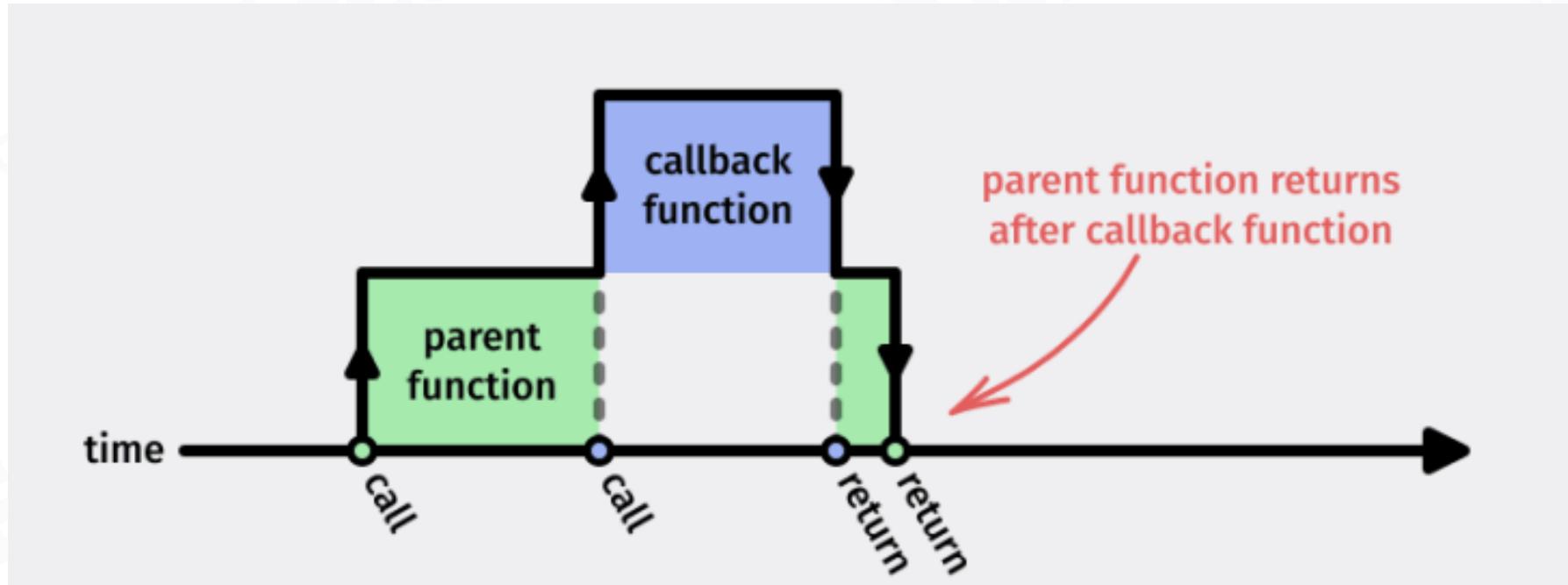
## Event Dispatch

- Send event to appropriate destination





## Event Dispatch : Immediate





# Event Dispatch : Immediate

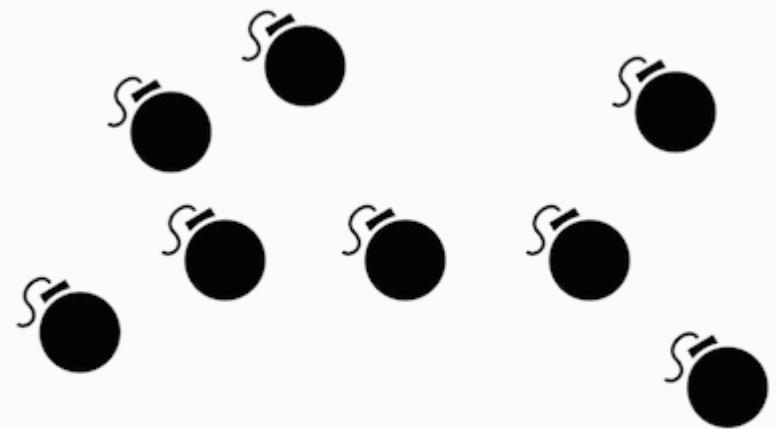
- Deep well of callbacks

When a bomb explodes near others.....

...

```
EventManager::sendEvent()
Bomb::explode()
Bomb::onEvent()
EventManager::sendEvent()
Bomb::explode()
Bomb::onEvent()
EventManager::sendEvent()
Bomb::explode()
Bomb::onEvent()
EventManager::sendEvent()
Bomb::explode()
...
Deep Callstack GameWorld::updateObjects()
Engine::gameLoop()
main()
```

Recursively call these functions





## Event Dispatch : Immediate

### Problem

- Blocked by function

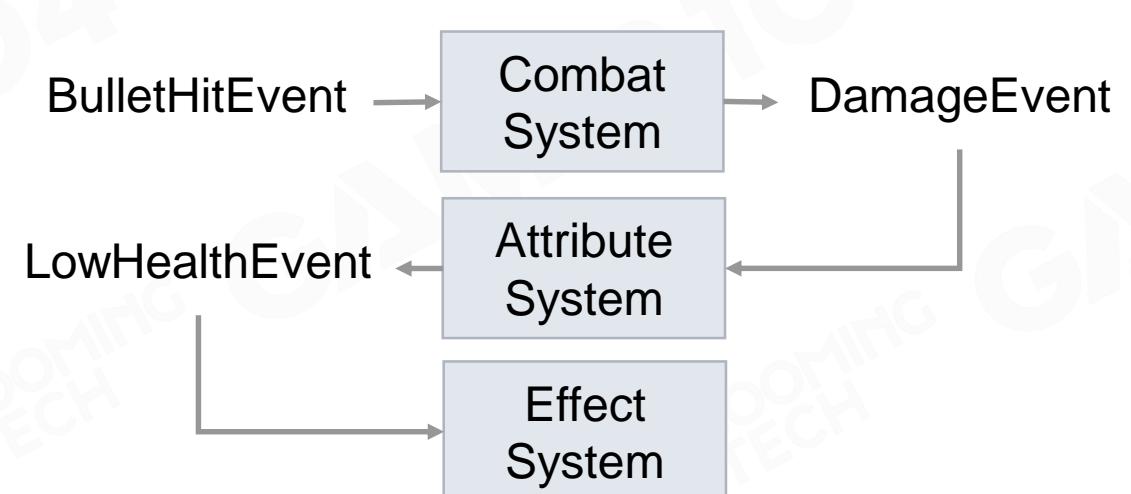
...

```
EventManager::sendEvent()  
EffectSystem::addEffect()  Blocked  
EffectSystem::onEvent()  
EventManager::sendEvent()  
AttributeSystem::updateHealth()  
AttributeSystem::onEvent()  
EventManager::sendEvent()  
CombatSystem::calculateDamage()  
CombatSystem::onEvent()  
GameWorld::updateSystem()  
Engine::gameLoop()  
main()
```

The bleeding effect should be loaded  
but cost plenty of time in this function call



Soldier begin  
bleeding after hit





# Event Dispatch : Immediate

## Problem

- Difficult for parallelization

EffectSystem



AttributeSystem



CombatSystem



AmmoSystem



*timeline*

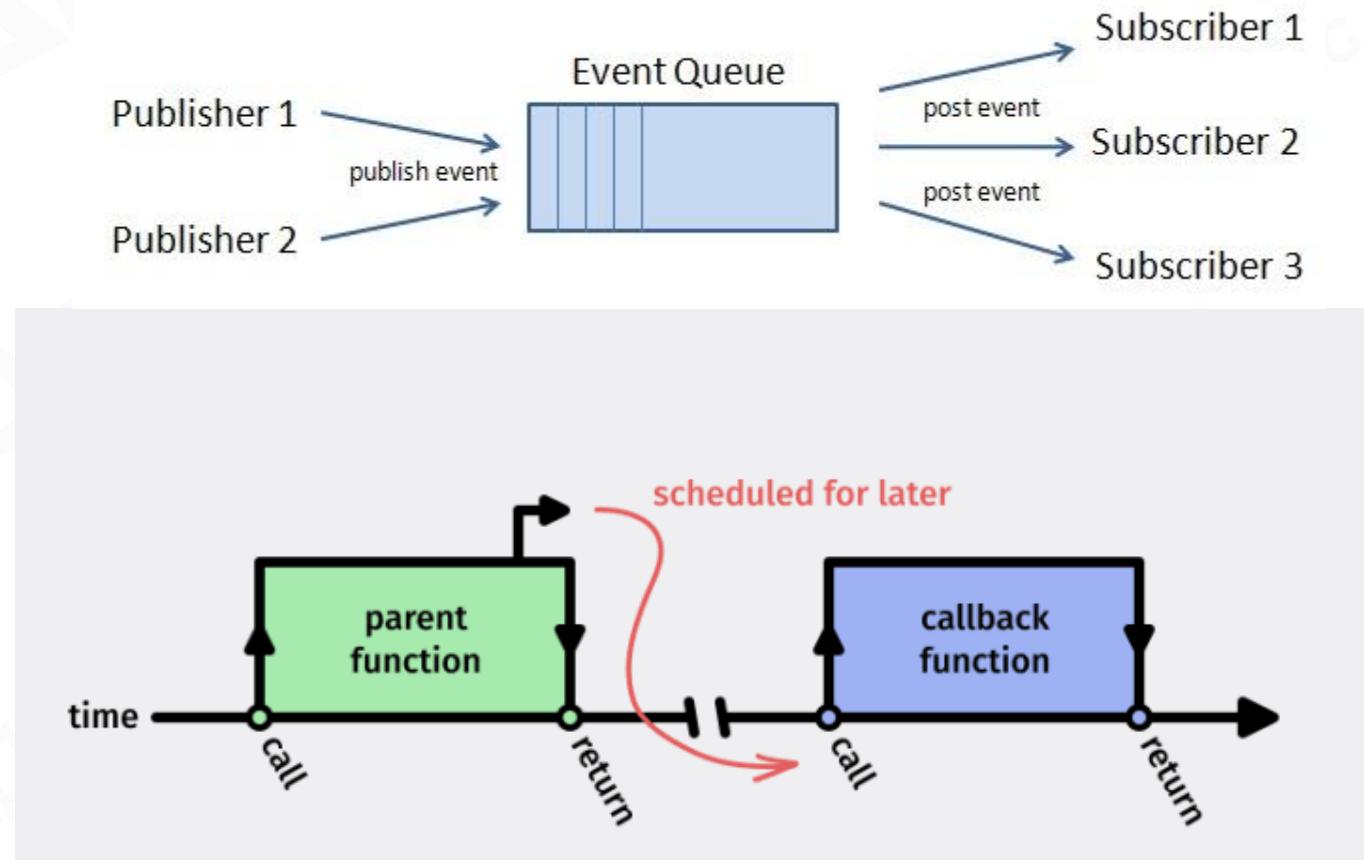




## Event Queue

### Basic Implementation

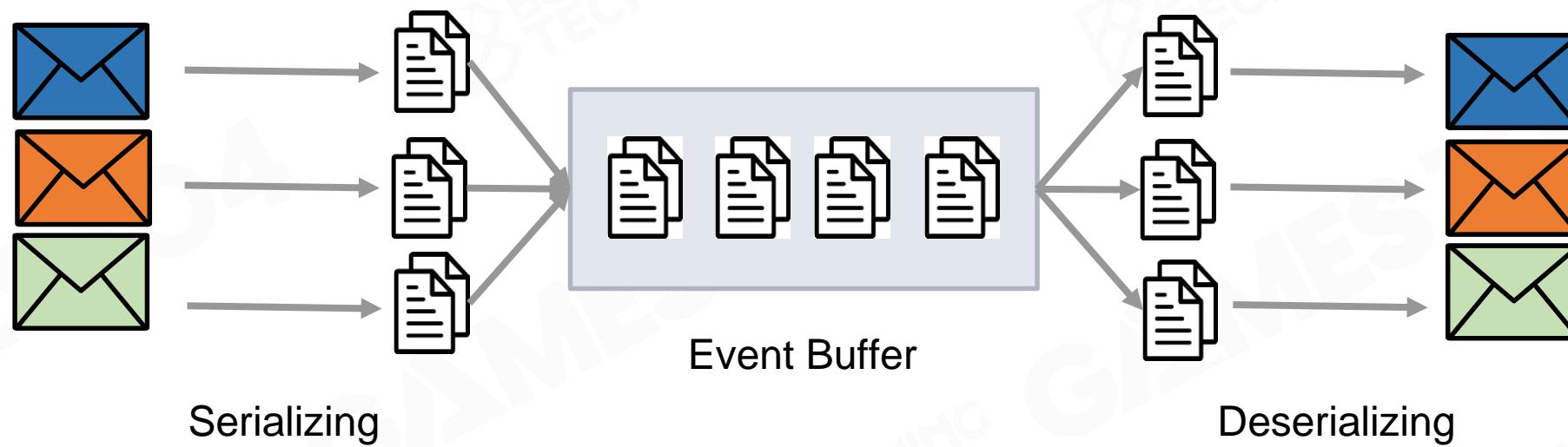
- Store events in queue for handling at an arbitrary future time





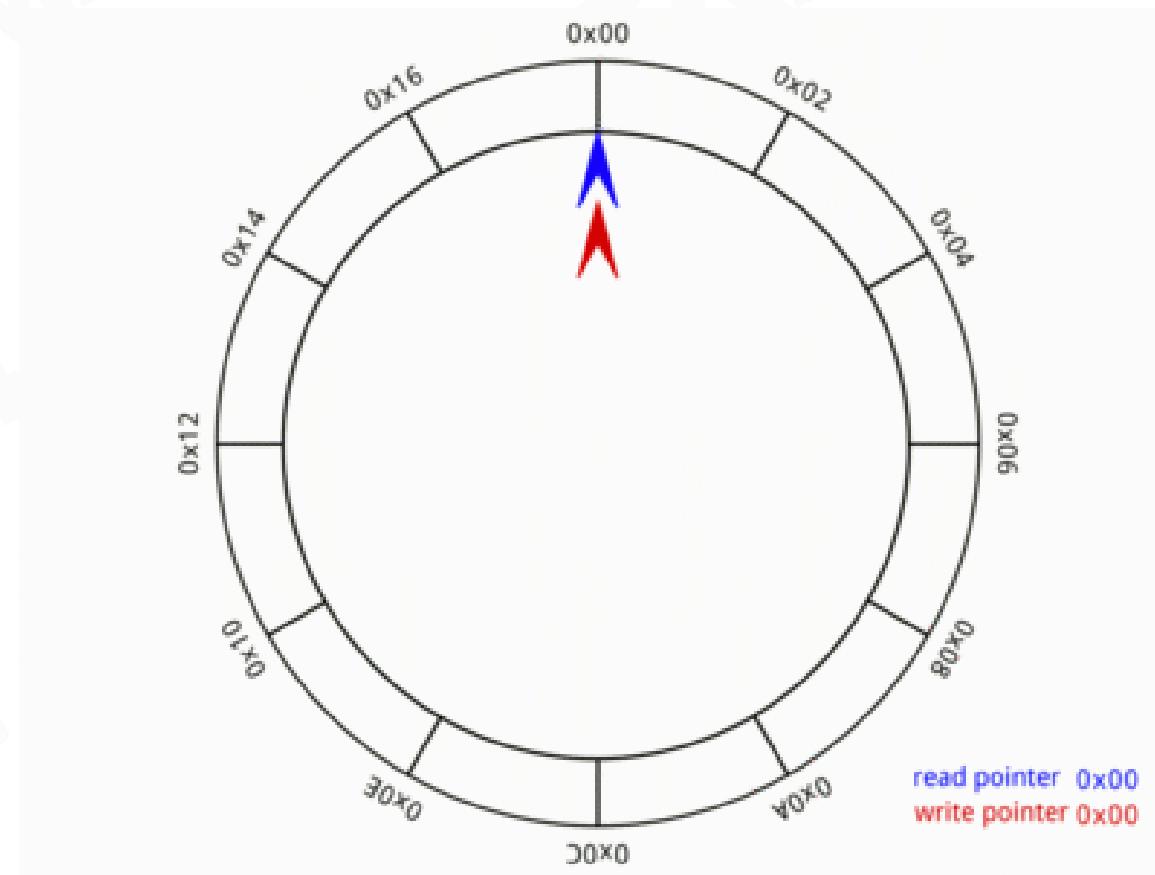
## Event Serializing and Deserializing

- To store various types of events



# Event Queue

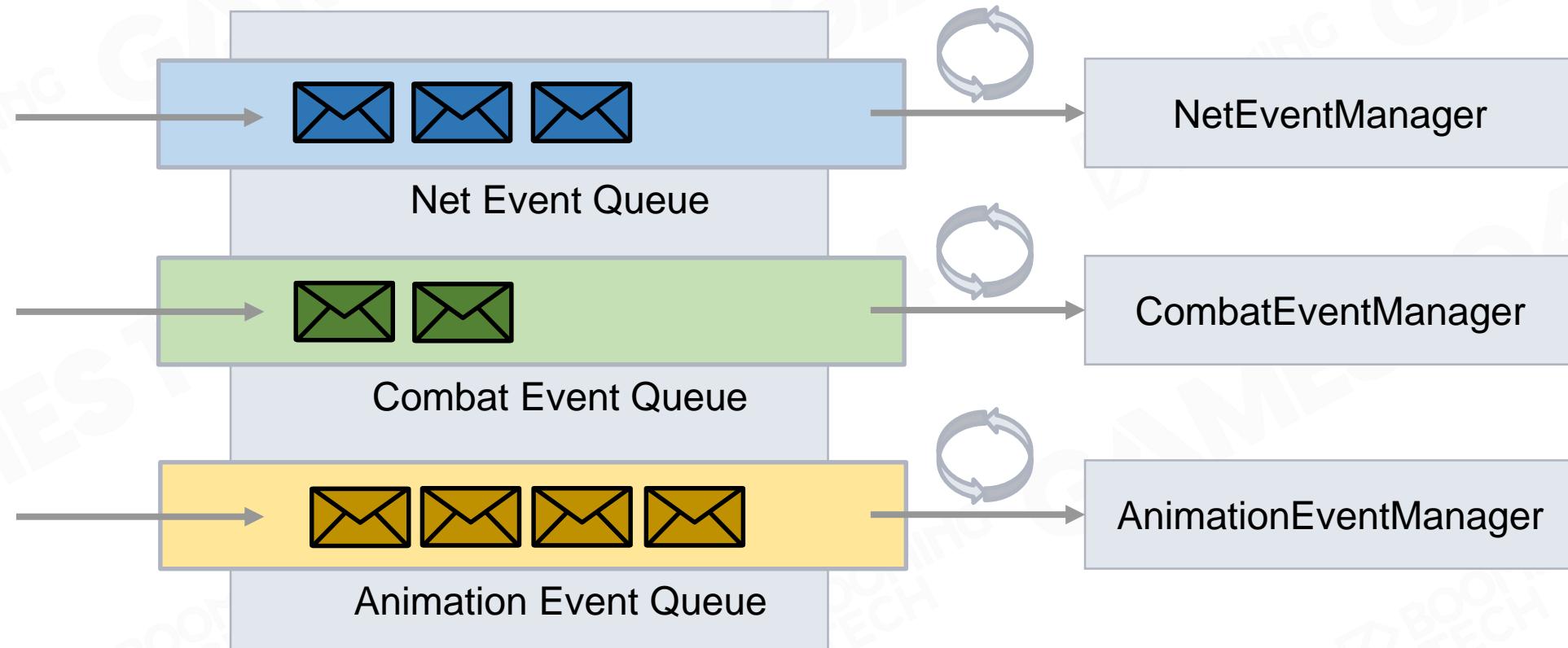
# Ring buffer





## Event Queue

### Batching



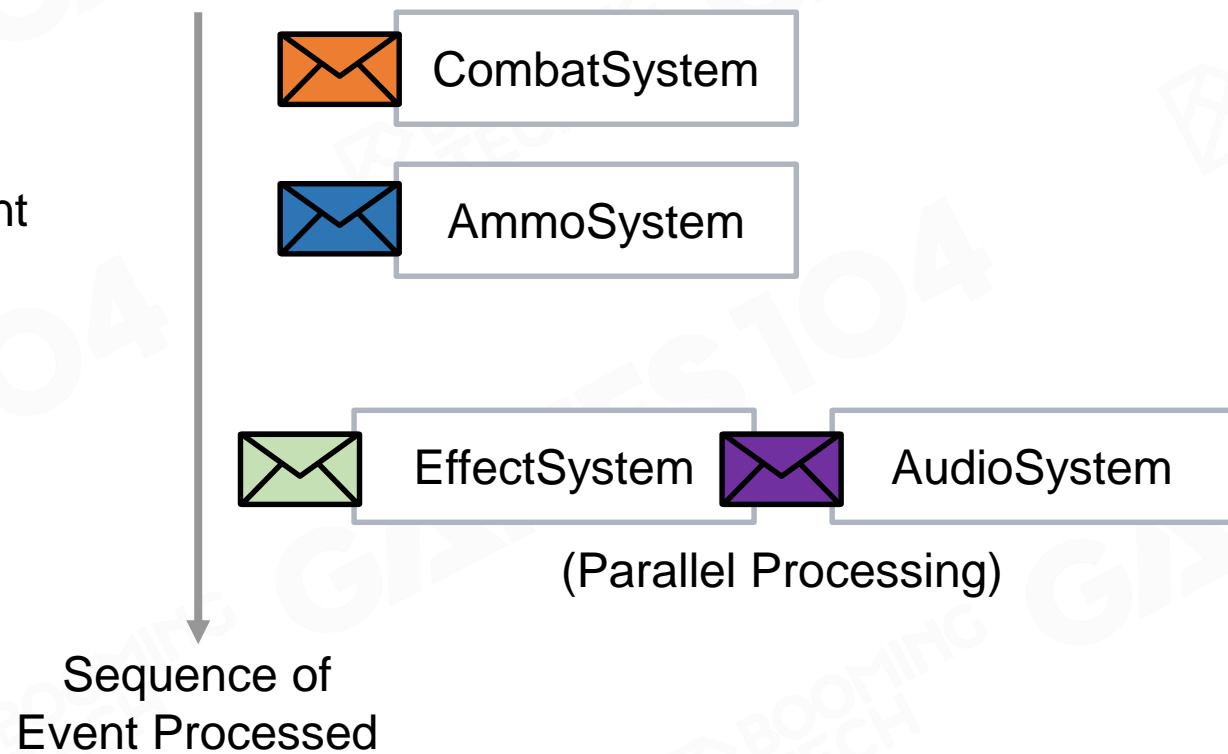
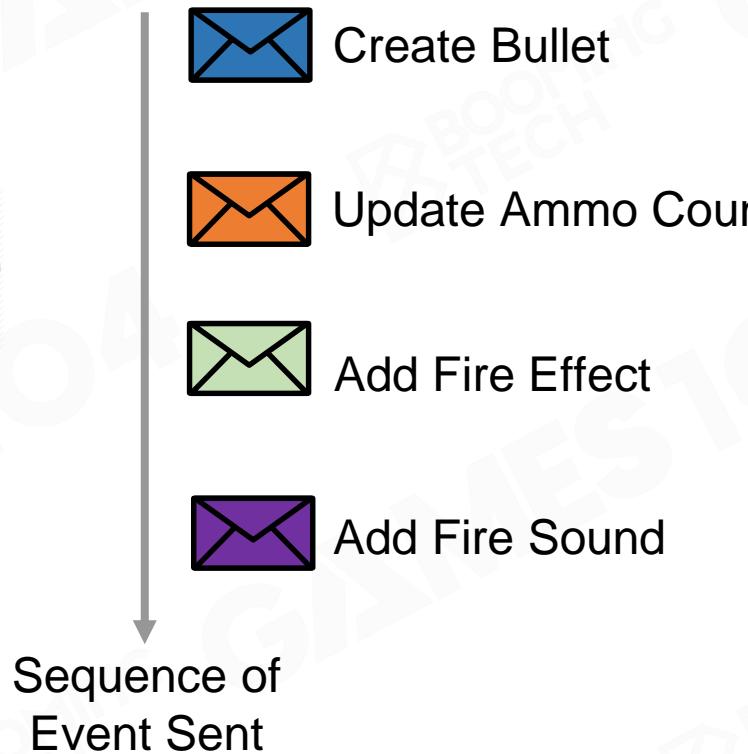


## Problems of Event Queue (1/2)

- Timeline not determined by publisher



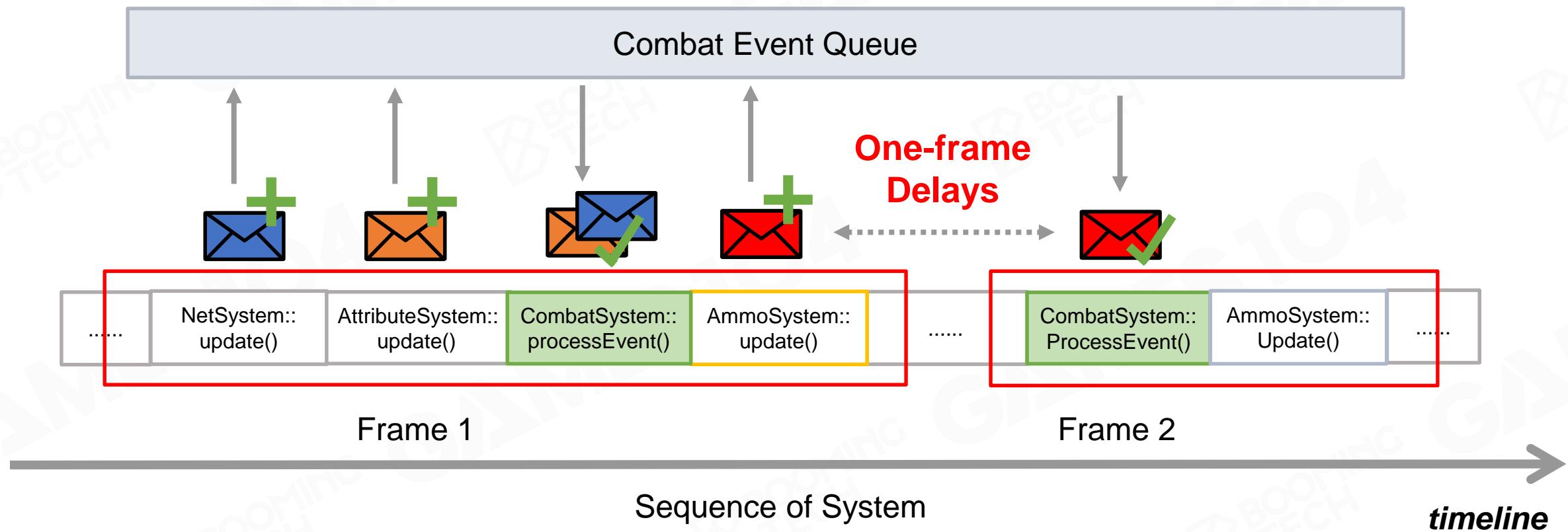
Fire a bullet





## Problems of Event Queue (2/2)

- One-frame delays





## GamePlay

---

Game Logic



# Early Stage Game Logic Programming

Compiled language(mostly C/C++)

- Compiled to machine code with high performance
- More easier to use than assembly language



```
void Player::tick(Float delta)
{
    updateDirection();

    if (isKeyPressed(MOUSE_LEFT))
    {
        fire();
    }

    if (isKeyDown(KEY_W))
    {
        moveForward(delta);
    }
    else if (isKeyDown(KEY_S))
    {
        moveBackward(delta);
    }
    if (isKeyDown(KEY_A))
    {
        moveLeftward(delta);
    }
    else if (isKeyDown(KEY_D))
    {
        moveRightward(delta);
    }

    ...
}
```



## Problem of Compiled Languages

Game requirements get complex as hardware evolves

- Need quick iterations of gameplay logic

Issues with compiled language

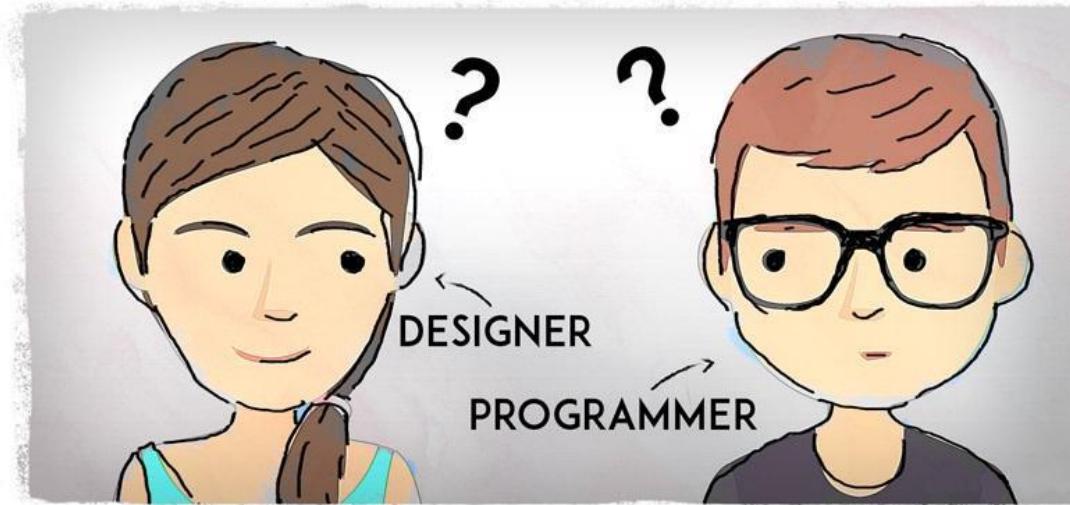
- Need recompilation with even a little modification
- Program can easily get crashed with incorrect codes





## Glue Designers and Programmers

- Get rid of inefficient communication between designers and programmers
- Designers need direct control of gameplay logic
- Artists need to quickly adjust assets at the runtime environment





## Scripting Languages

- Support for rapid iteration
- Easy to learn and write
- Support for hot update
- Stable, less crash by running in a sandbox

```
function tick(delta)
    if input_system.isKeyDown(KeyCode.W) then
        self:moveForward(delta)
    elseif input_system.isKeyDown(KeyCode.S) then
        self:moveBackward(delta)
    end

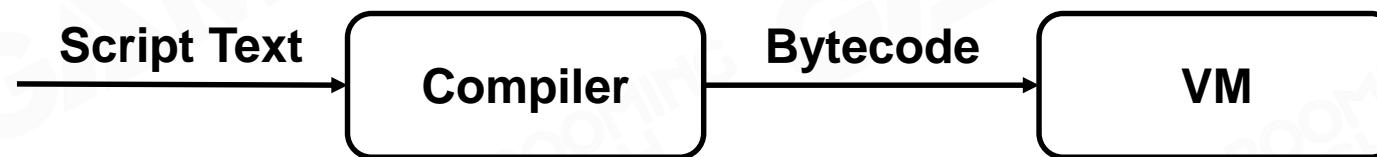
    if input_system.isKeyDown(KeyCode.MouseLeft) then
        self:fire(delta)
    end
    ...
end
```

Lua Script Example



# How Script Languages Work

Script is converted to **bytecode** by a **compiler** first, then run on a **virtual machine**



Instruction	Opcode	Description
NUM	0x00	Push a literal number
ADD	0x01	Pop two numbers and push the result of addition
PRT	0x02	Pop a value and print

Instruction Set Example

Script:	print(36 + 15)					
Bytecode:	0x00 0x24 0x00 0x0F 0x01 0x02					
Instruction:	NUM 36 NUM 15 ADD PRT					

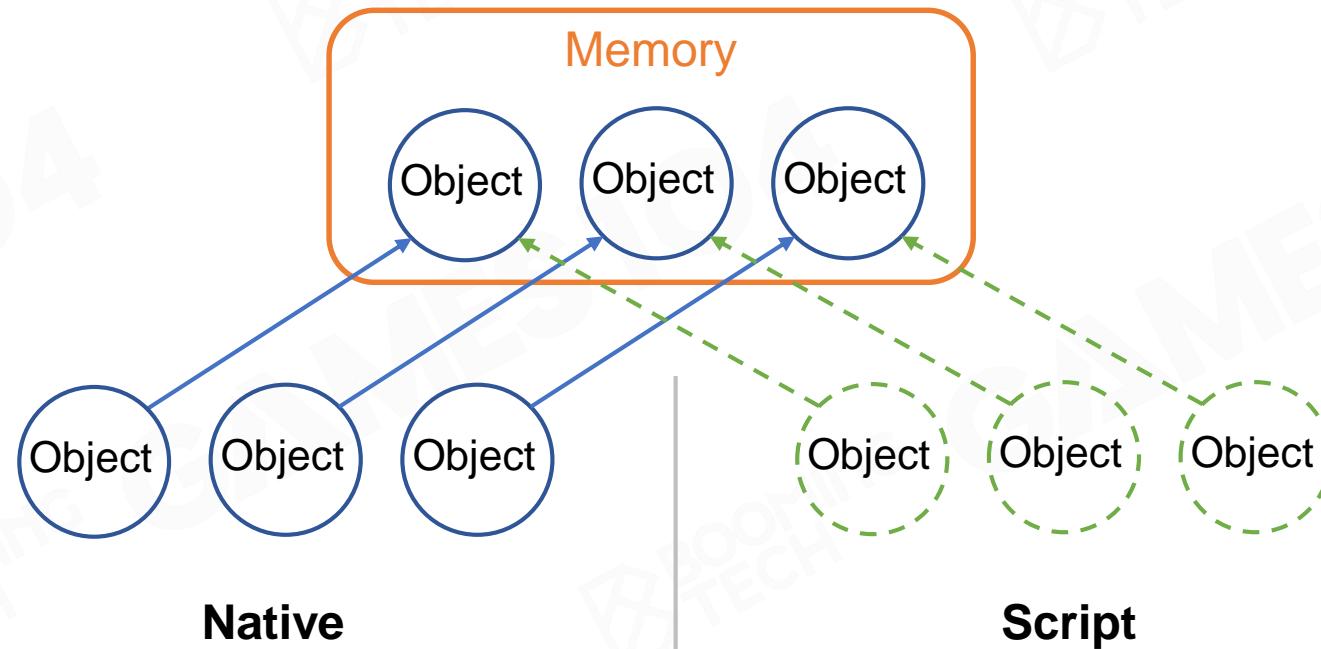
Bytecode Example



## Object Management between Scripts and Engine (1/2)

Object lifetime management in *native engine code*

- Need to provide an object lifetime management mechanism
- Not safe when script uses native objects (may have been destructed)

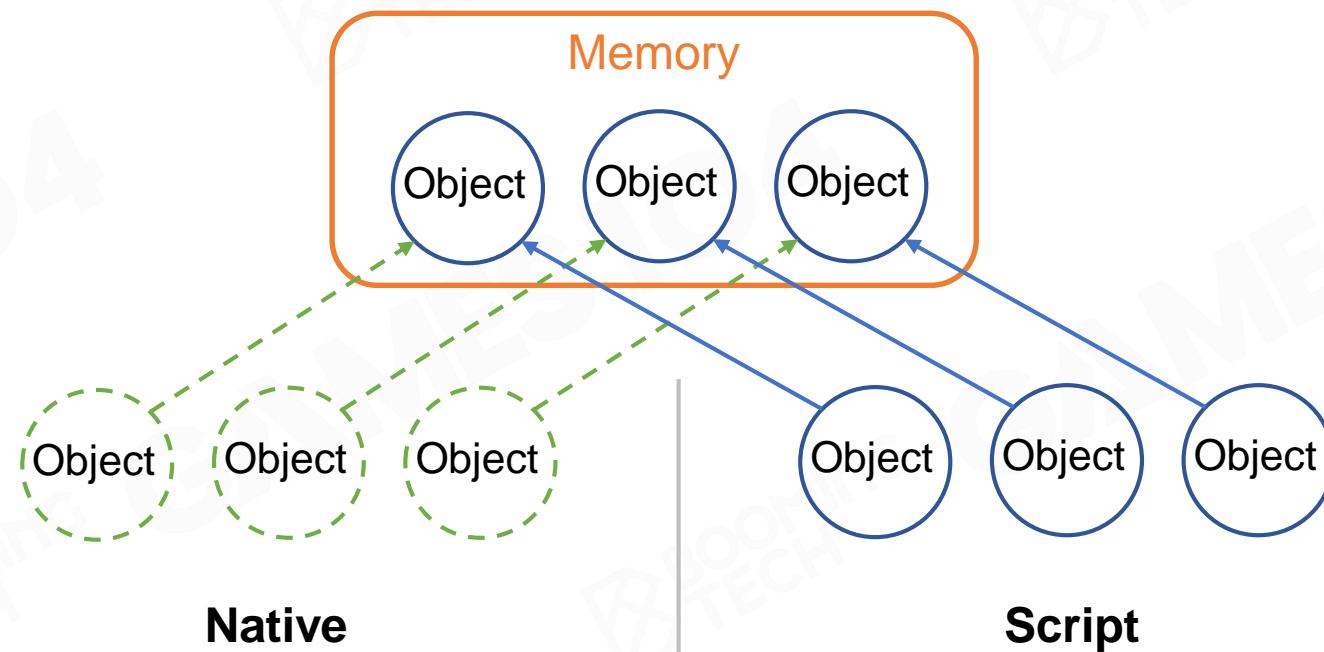




## Object Management between Scripts and Engine (2/2)

Object lifetime management in *script*

- The lifetime of objects are auto managed by script GC
- The time when object is deallocated is uncontrolled (controlled by GC)
- Easy to get memory leak if reference relations get complex in script

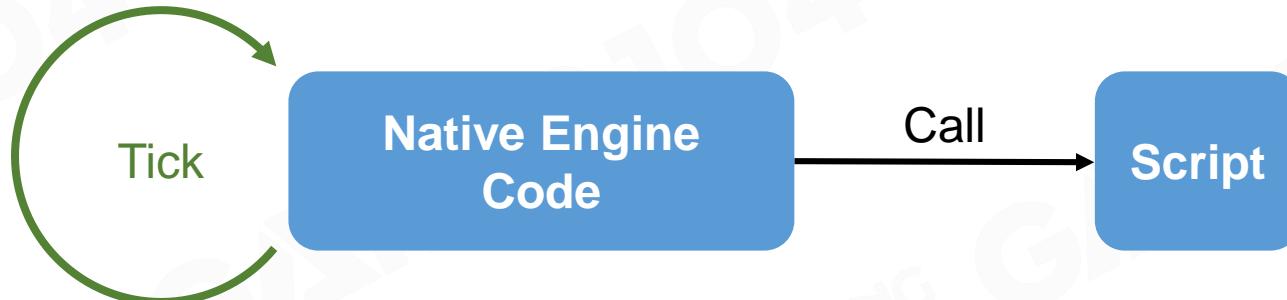




## Architectures for Scripting System (1/2)

Native language dominants the game world

- Most gameplay logic is in native code
- Script extends the functionality of native engine code
- High performance with compiled language

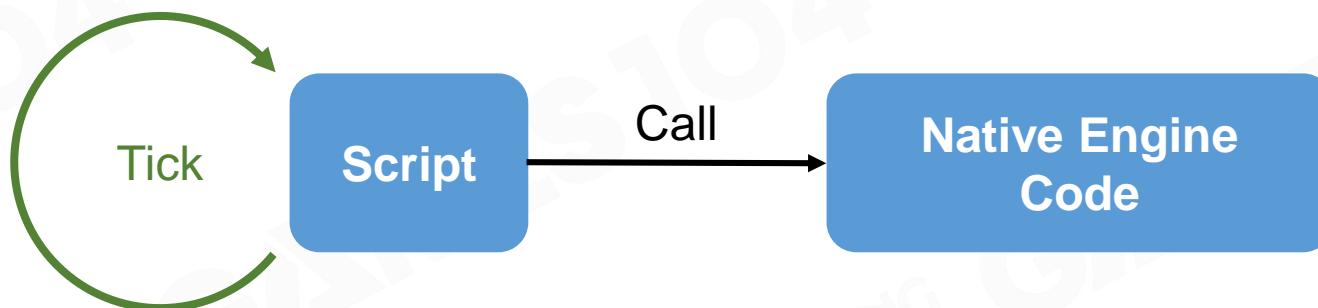




## Architectures for Scripting System (2/2)

Script language dominants the game world

- Most gameplay logic is in script
- Native engine code provides necessary functionality to script
- Quick development iteration with script language





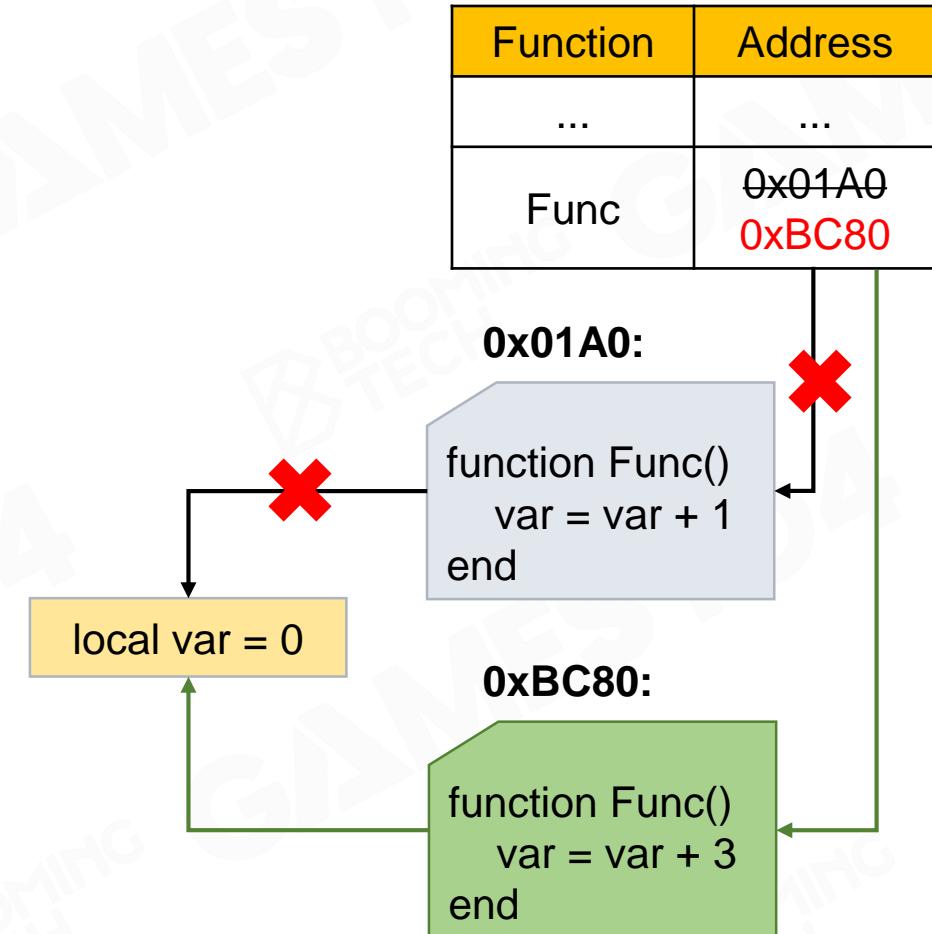
## Advanced Script Features - Hot Update

Allow modifications of script while game is running

- Quick iteration for some specific logic
- Enable to fix bugs in script while game is online

A troublesome problem with hot update

- All variables reference to old functions should be updated too



Hot update workflow example

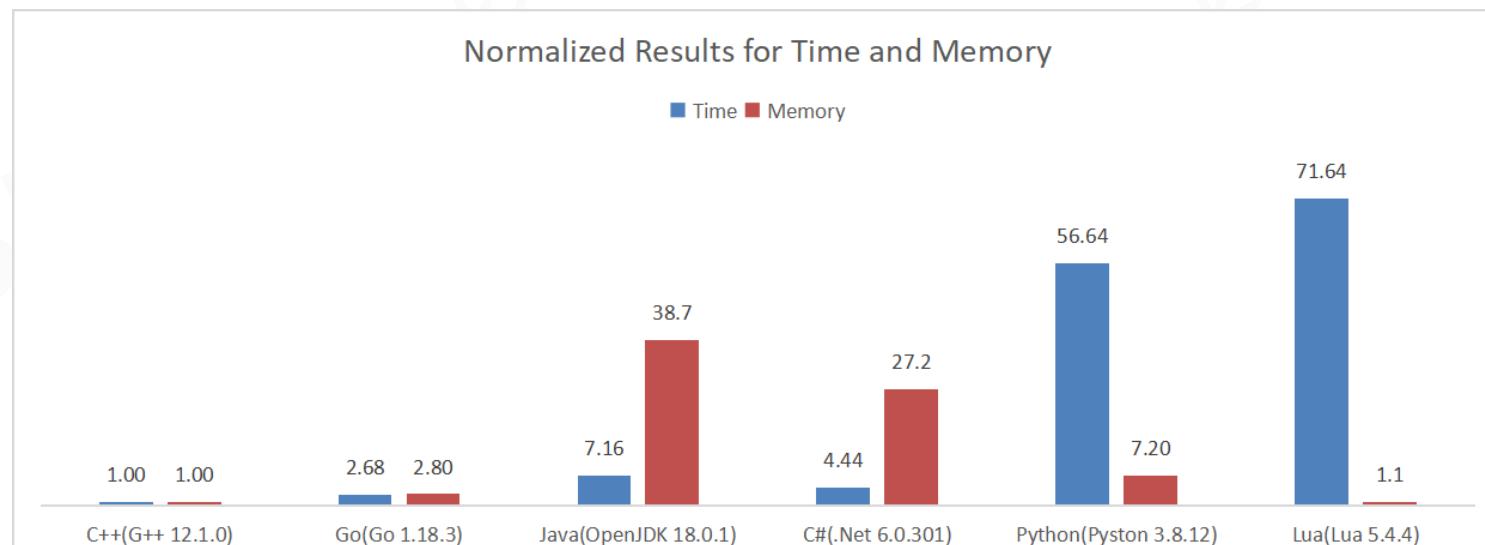


## Issues with Script Language

The **performance** is usually lower than compiled language

- Weakly typed language is usually harder to optimize when compile
- Need a virtual machine to run the bytecode
- JIT is a solution for optimization

Weakly typed language is usually harder to refactor



N-body problem benchmark of popular languages



## Make a Right Choice of Scripting Language

Things need to be considered

- Language performance
- Built-in features, e.g. object-oriented programming support

Select the proper architecture of scripting

- Object lifetime management in native engine code or script
- Which one is dominant, native language or script



## Popular Script Languages (1/2)

Lua (used in *World of Warcraft*, *Civilization V*)

- Robust and mature
- Excellent runtime performance
- Light-weighted and highly extensible

Python (used in *The Sims 4*, *EVE Online*)

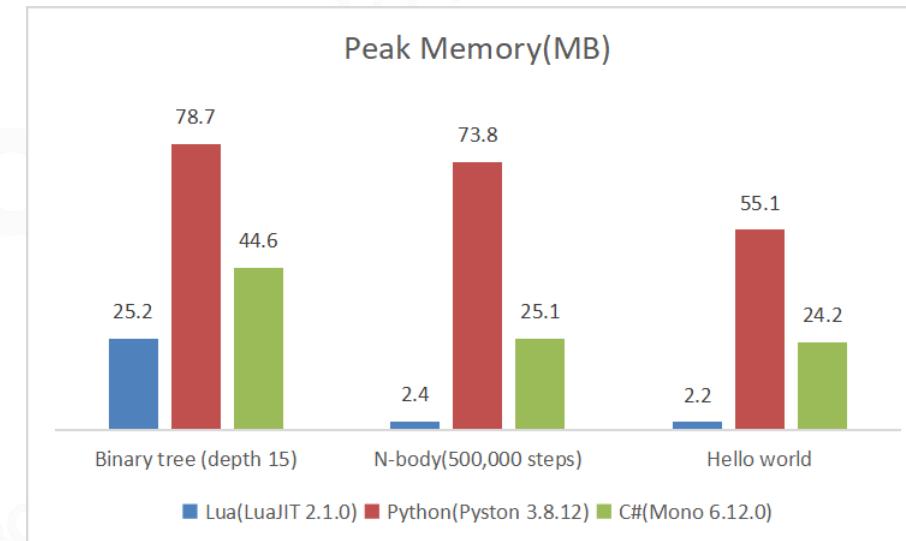
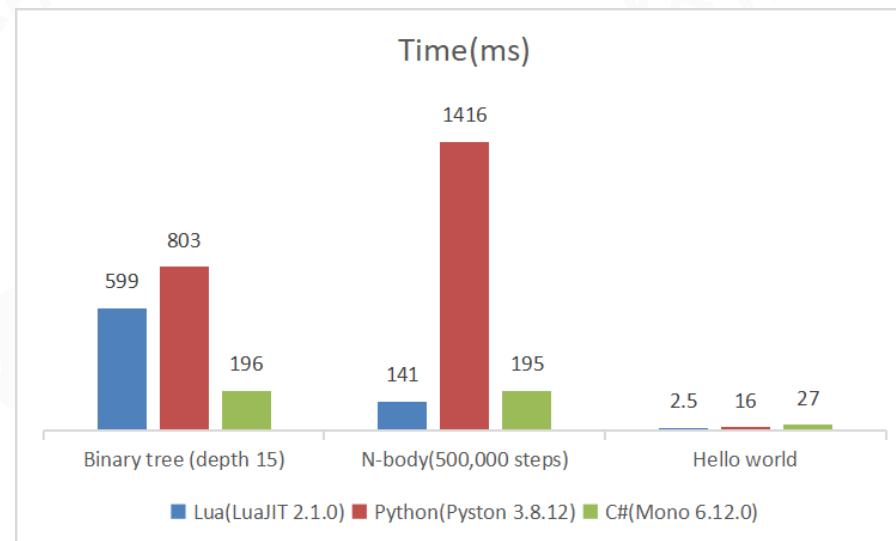
- Reflection support
- Built-in object-oriented support
- Extensive standard libraries and third-party modules



## Popular Script Languages (2/2)

C# (to bytecode offline, used in *Unity*)

- Low learning curve, easy to read and understand
- Built-in object-oriented support
- Great community with lots of active developers



Benchmark of 3 popular script languages



## GamePlay

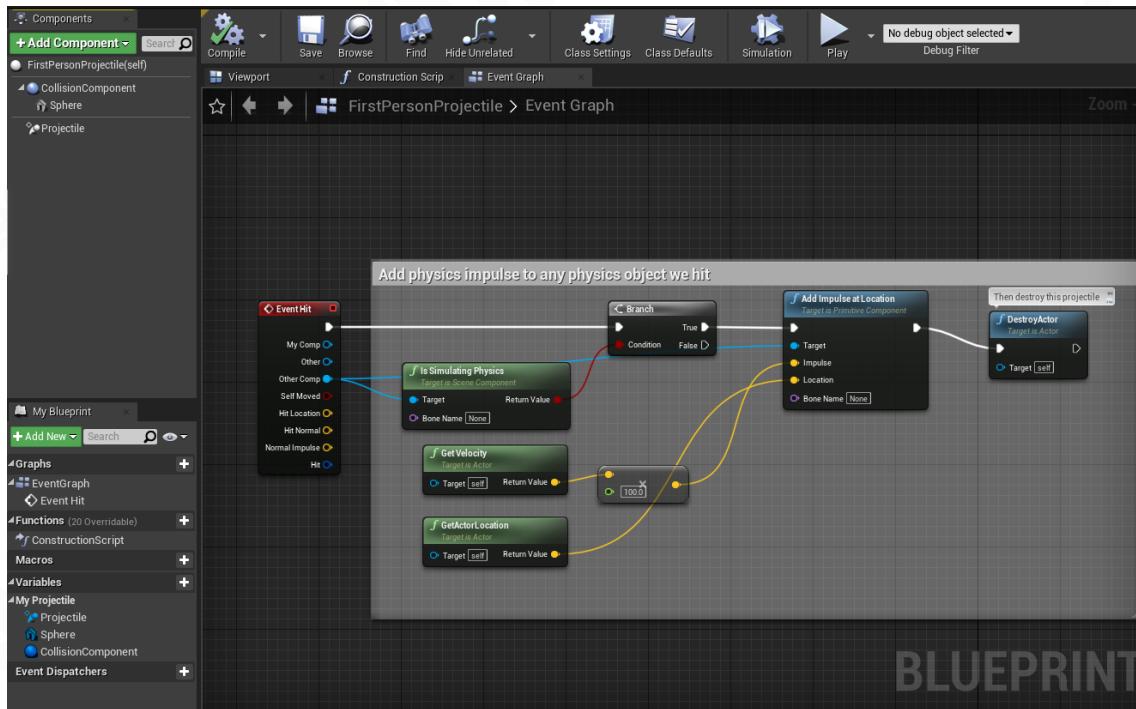
---

Visual Scripting

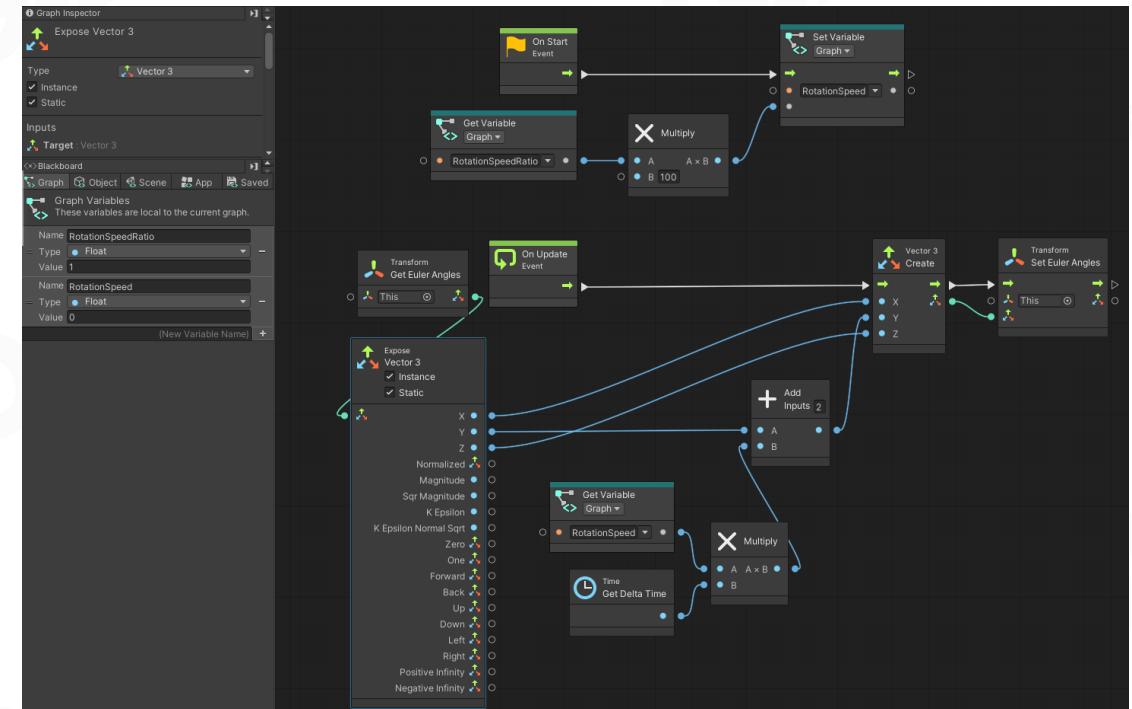


## Why We Need Visual Scripting

- Friendly to non-programmers, especially designers and artists
- Less error-prone with drag-drop operations instead of code writing



Unreal Blueprint



Unity Visual Scripting



# Visual Script is a Program Language

Visual script is also a programming language, which usually needs

- Variable
- Statement and Expression
- Control Flow
- Function
- Class (for object-oriented programming language)

```
class Class
{
public:
    int m_a;
};

void Function(int a)
{
    Class c;           Variable
    if (a >= 0)       Expression
    {
        c.m_a = 3 * a + 4;   Statement
    }
    else
    {
        c.m_a = 0;           Statement
    }
    ...
}
```

Control Flow



# Variable

Preserve the data to be processed or output

- Type
  - Basic type, e.g. integer, floating
  - Complex type, e.g. structure
- Scope
  - Local variable
  - Member variable
  - ...

Complex type  
`struct Complex`  
{  
|      Basic type     Member Variable  
|      int            a;  
|      float          b;  
|      char           c;  
};

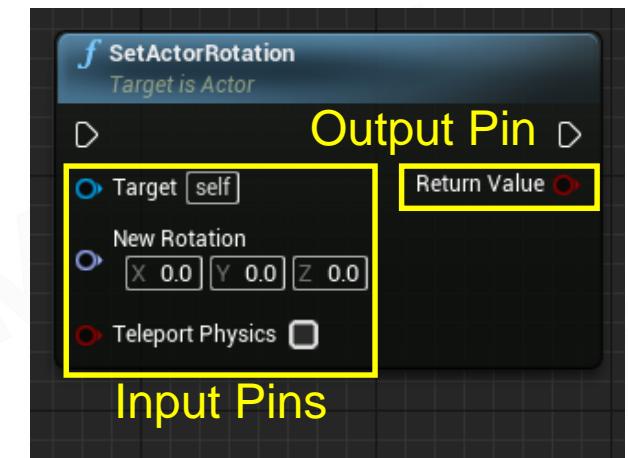
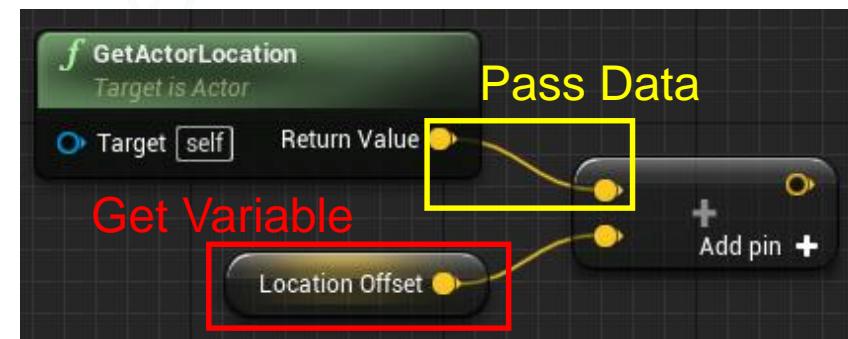
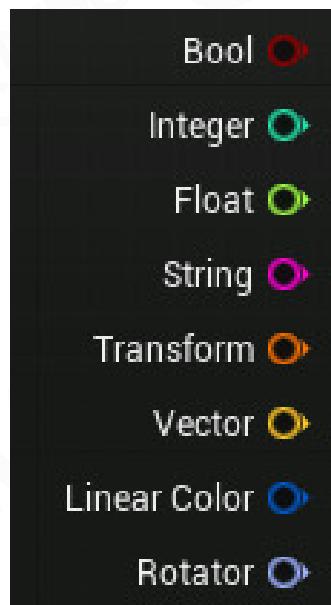
`void Example()`  
{  
|      Local Variable  
|      double d;  
|      ...  
}



# Variable Visualization - Data Pin and Wire

Use **data wires** through **data pins** to pass **variables** (parameters)

- Each data type uses a unique pin color





# Statement and Expression

Control how to process data

- Statement: expresses some action to be carried out
  - Assignment Statement
  - Function Statement
  - ...
- Expression: to be evaluated to determine its value
  - Function Expression
  - Math Expression
  - ...

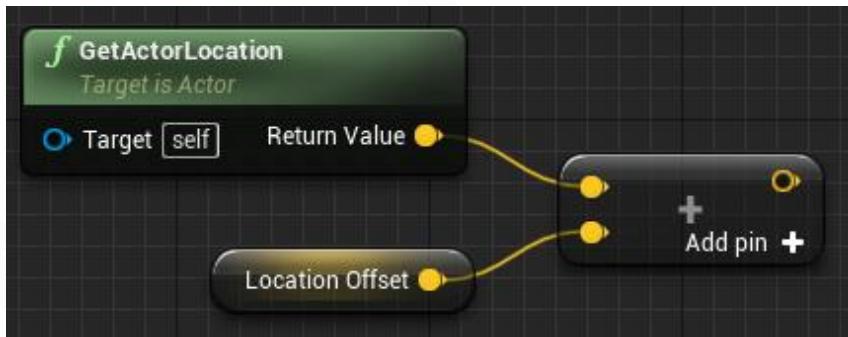
```
void Example()
{
    ...
Assignment Statement
int a = 3;
Function Statement
doSomething();
Function Expression
int b = getValue();
Math Expression
int sum = a + b;
...
```



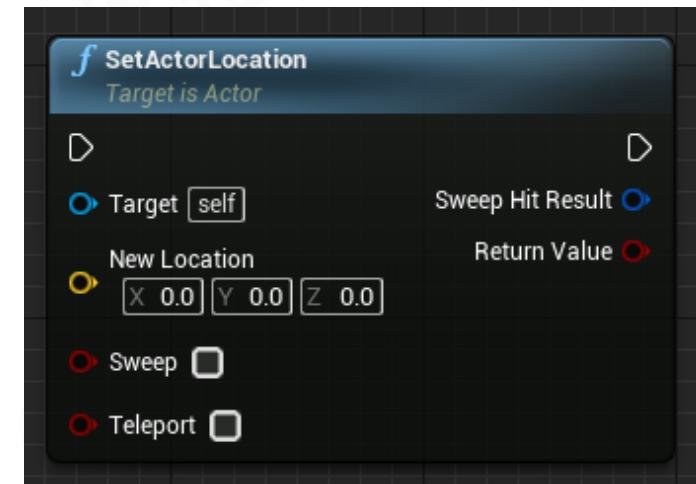
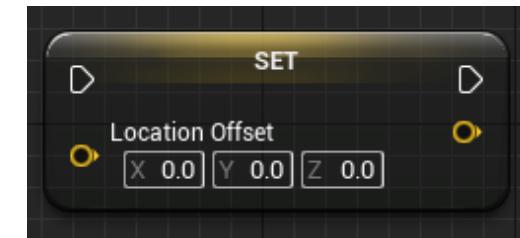
# Statement and Expression Visualization - Node

Use **nodes** to represent **statements** and **expressions**

- Statement Node
- Expression Node



Expression Nodes



Statement Nodes



## Control Flow

Control the statement execution order

- Sequence
  - By default statements are executed one by one
- Conditional
  - Next statement is decided by a condition
- Loop
  - Statements are executed iteratively until the condition is not true

```
void Example()  
{
```

```
    first();  
    then();
```

Sequence

```
if (condition)  
{  
    doIfTrue();  
}  
else  
{  
    doIfFalse();  
}
```

Conditional

```
for (int i = 0; i < loop_count; ++i)  
{  
    doIteration();  
}
```

Loop



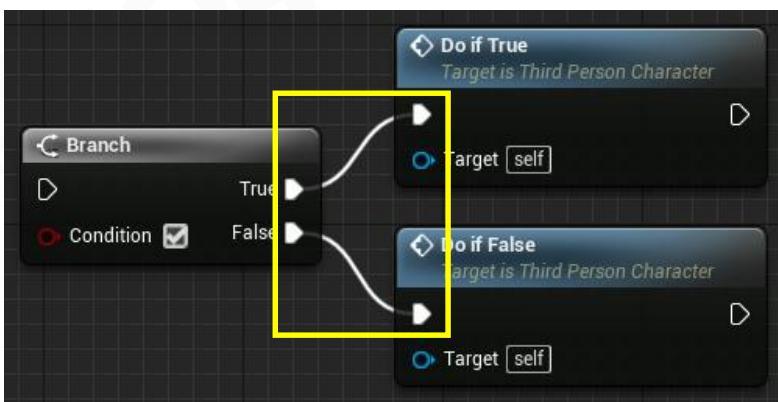
# Control Flow Visualization - Execution Pin and Wire

Use **execution wires** through **execution pins** to make **statements** sequence

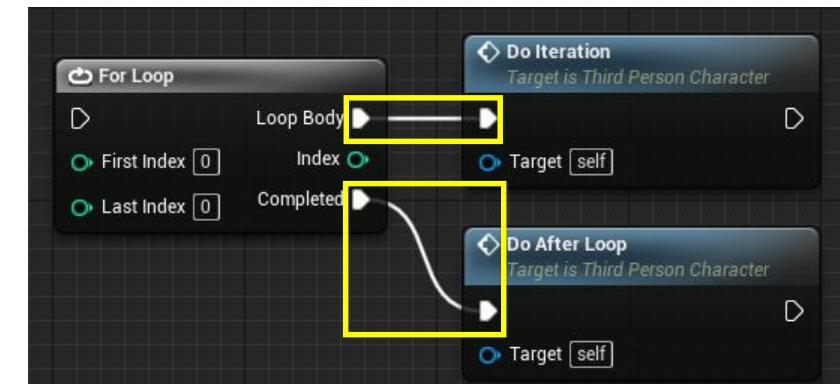
- Use control statement nodes to make different control flow



Sequence



Conditional



Loop



# Function

A logic module which take in data, process it and return result(s)

- Input Parameter
  - The data required input to be processed
- Function Body
  - Control how to process data
- Return value(s)
  - The data to be returned

```
float functionExample(float input)
{
    doSomething();
    float result = calculateResult();
    return result; Return value
}
```

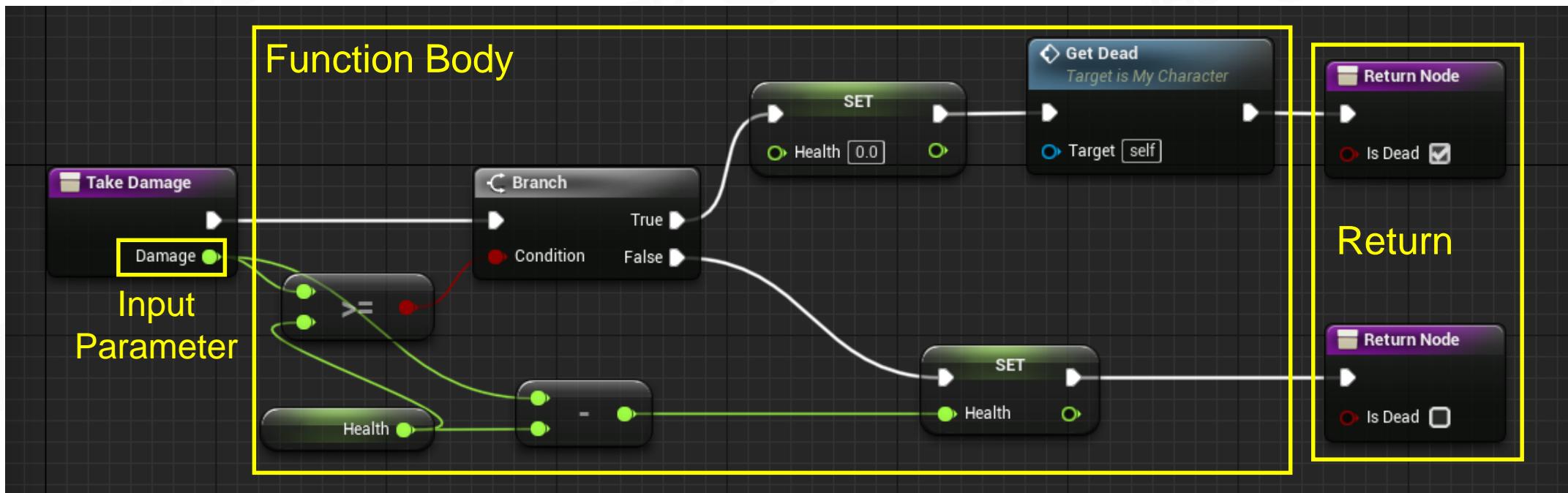
Input Parameter

Function Body



# Function Visualization - Function Graph

Use a **graph** with connected nodes to make a function



Example: Define a Function with Graph



# Class

A prototype for a kind of objects

- Member Variable
  - The lifetime is managed by the object instance
- Member Function
  - Can access member variables directly
  - Maybe overrided by derived classes

```
class ClassExample
{
public:
    int sum()
    {
        return m_a + m_b;
    }

private:
    int m_a;
    int m_b;
};
```

Member Function

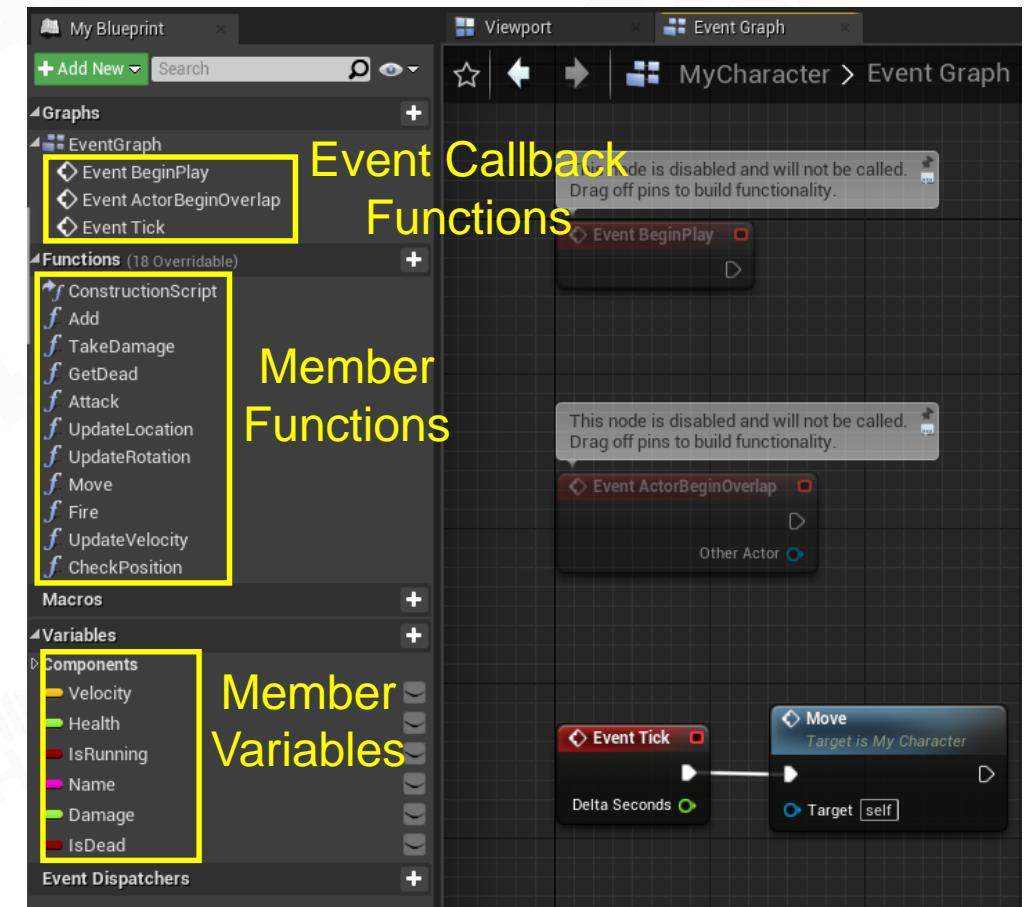
Member Variables



# Class Visualization - Blueprint

Use **blueprint** to define a class that inherits from a native class

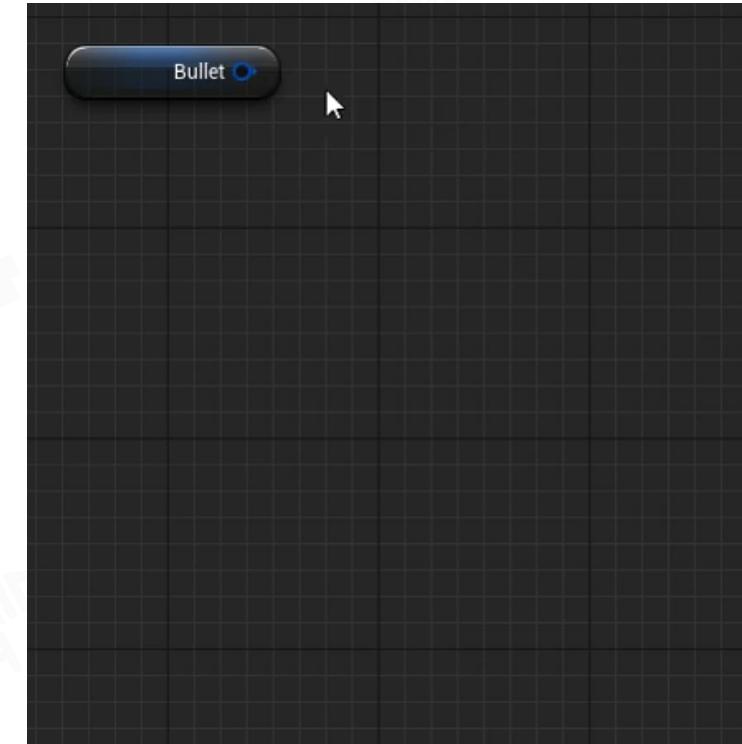
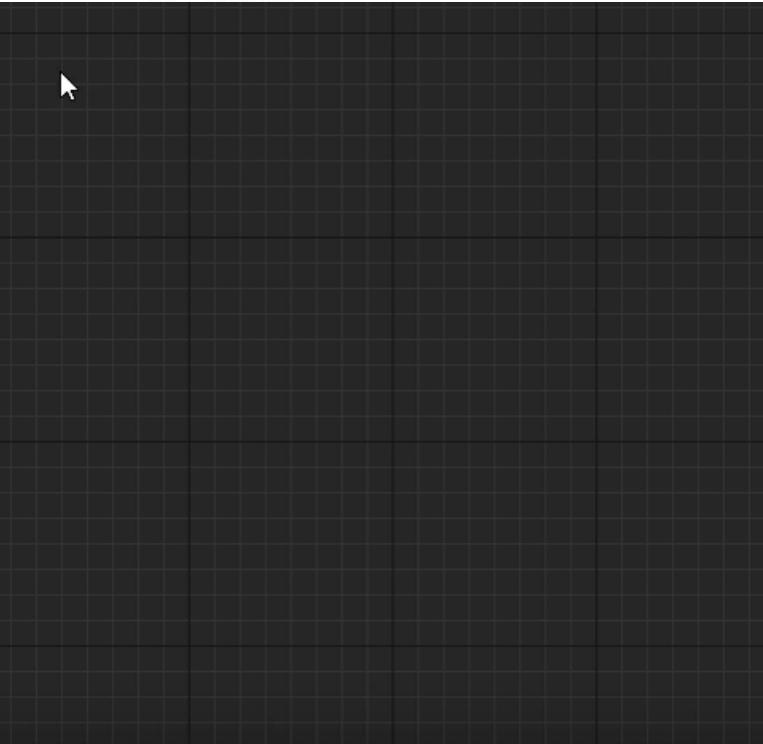
- Event Callback Functions
- Member Functions
- Member Variables
- ...





## Make Graph User Friendly

- Fuzzy finding
- Accurate suggestions by type

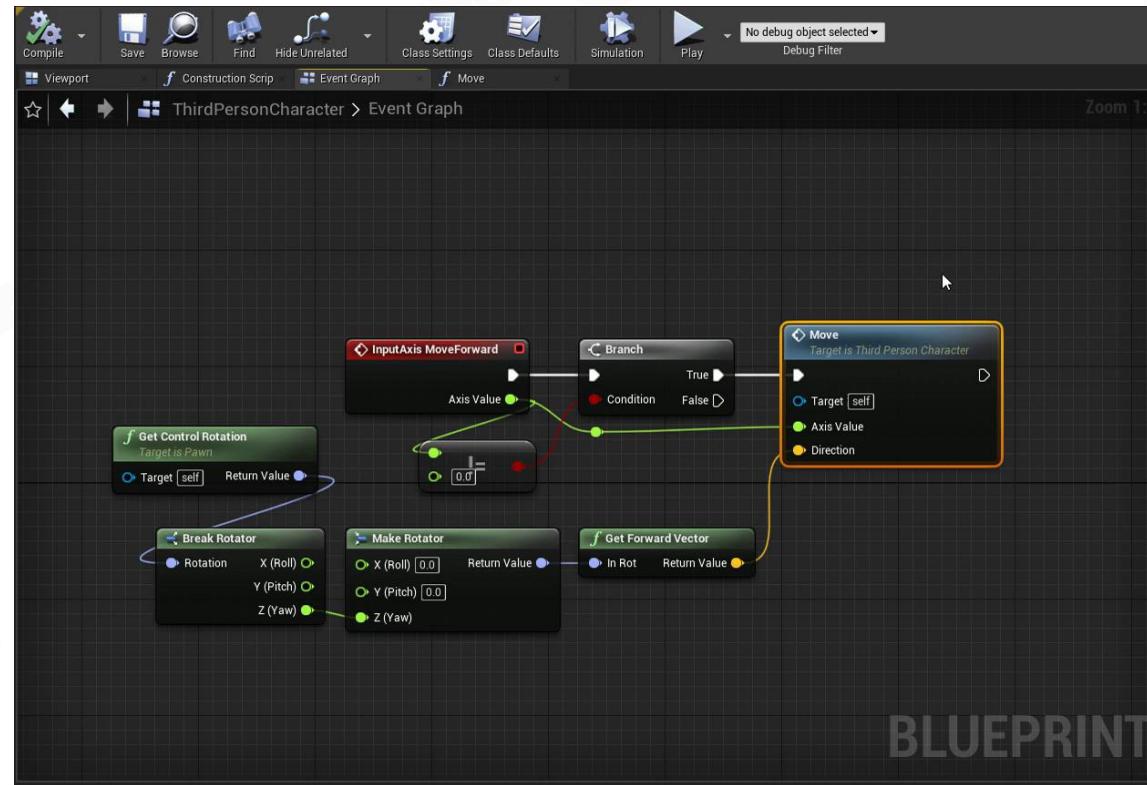




## Visual Script Debugger

Debug is an important step among development

- Provide user-friendly debug tools for visual scripting

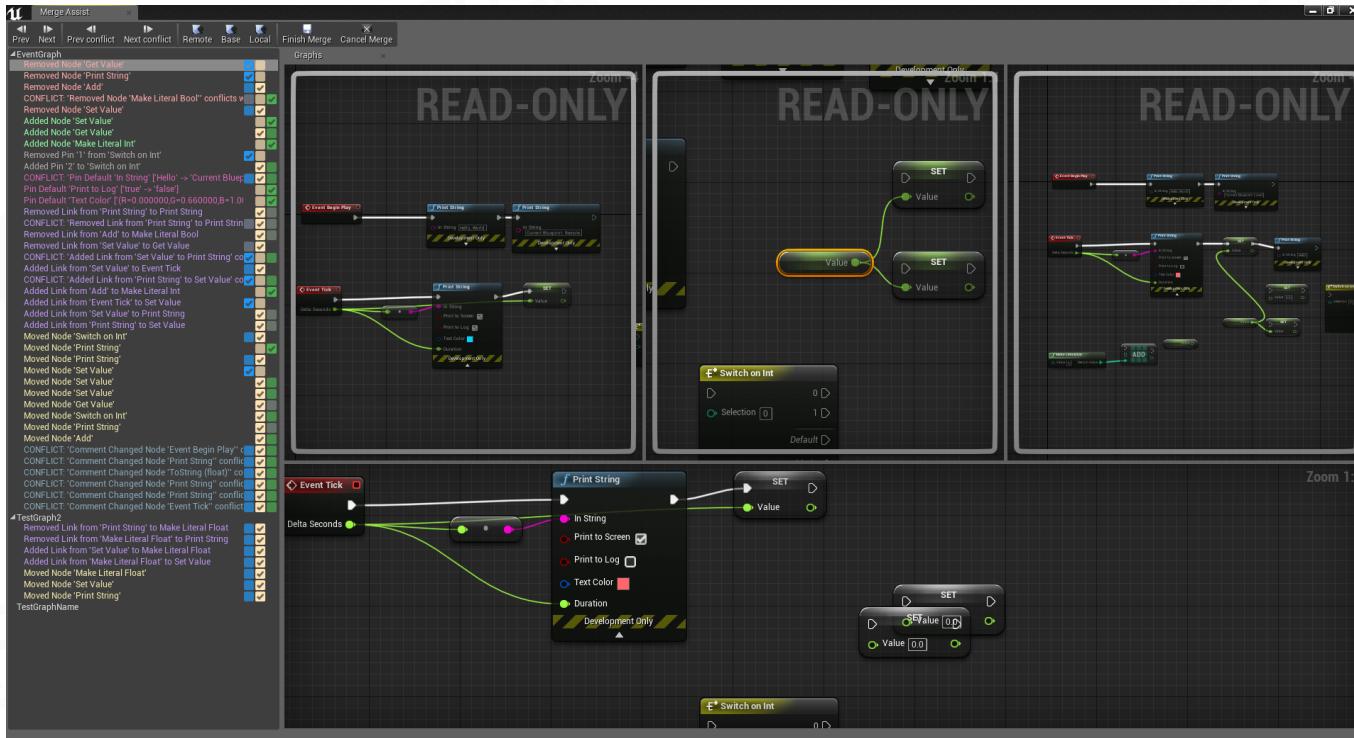




## Issues with Visual Scripting (1/2)

Visual script is hard to merge for a team work

- Usually a visual script is stored as a binary file
- Manually reorder script graph is inefficient and error-prone even with a merge tool

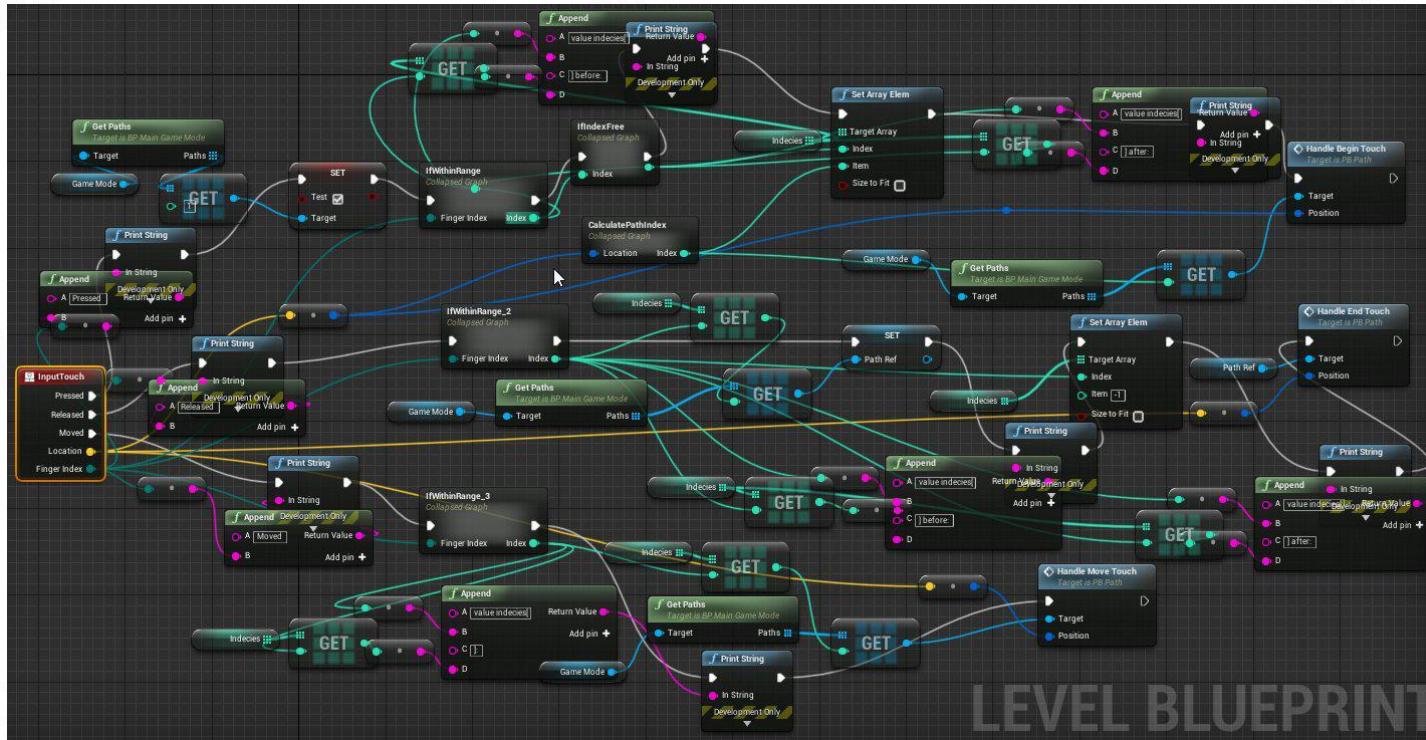




## Issues with Visual Scripting (2/2)

The graph can get pretty messy with complex logic

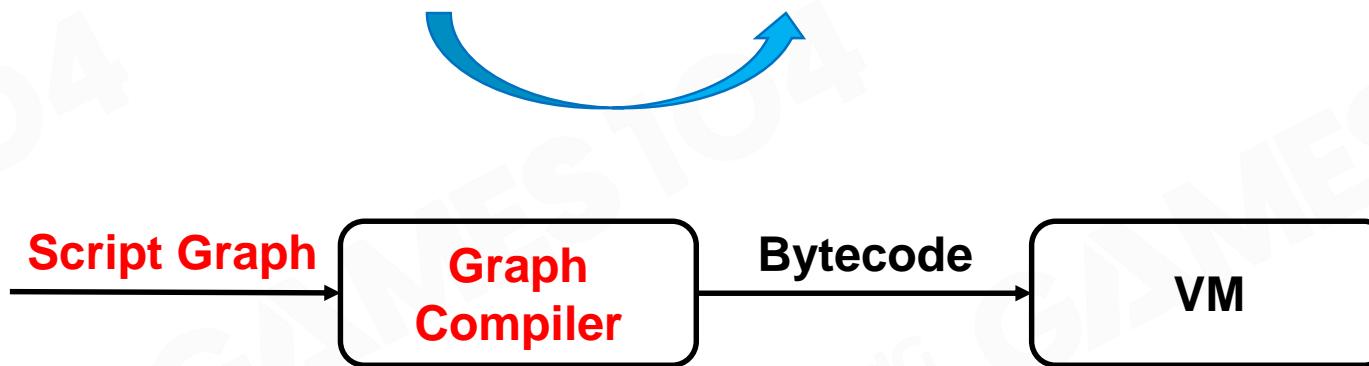
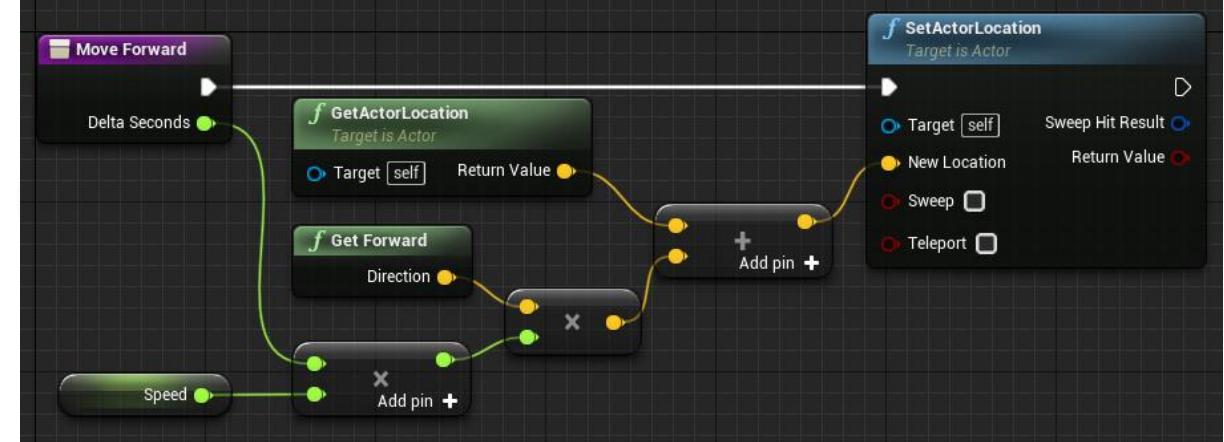
- Need uniform graph layout rules for a team work





# Script and Graph are Twins

```
function moveForward(delta_seconds)
    local location = self:getLocation()
    local direction = self:getForward()
    local speed = self.speed
    local movement = delta_seconds * speed * direction
    self:setLocation(location + movement)
end
```





## Game Play

---

“3C” in Game Play



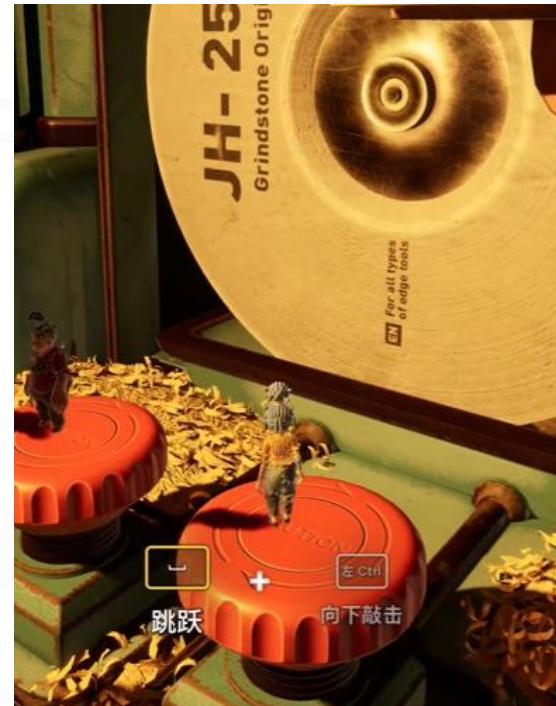
**It takes two**



## What is 3C?

3C: Character, Control & Camera

3C is the primary element that determines the gameplay experience





## Character

In-game character, both player and npc.

Include character movement, combat, health, mana, what skills and talents they have, etc.

One most basic element of a character is **movement**.





## Character: Well-designed Movement

Movement looks simple, but it's hard to do well.

In AAA games, every basic state of action needs to be broken down into detailed states.

several state changes in a few seconds



Idle



Start



Walk



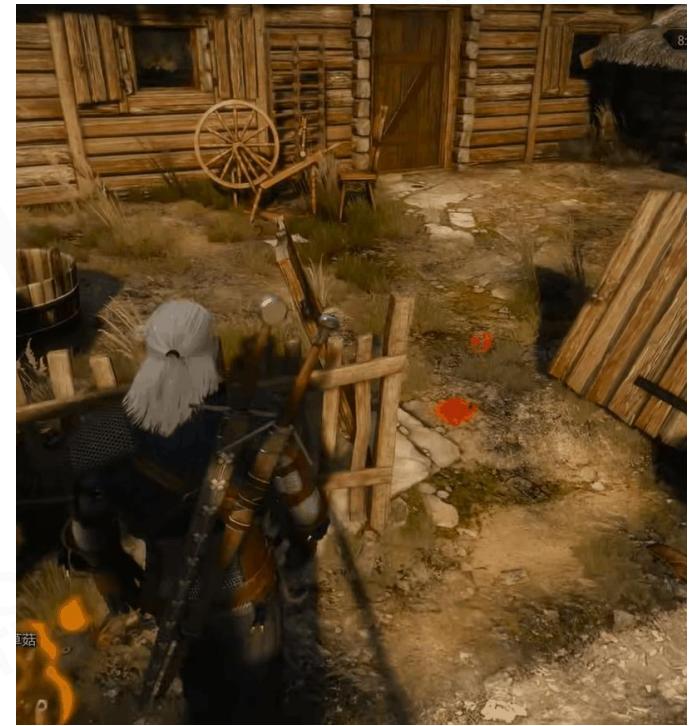
Accelerate



Run



Brake





## Extended Character: More complex and varied states



Hanging



Skating

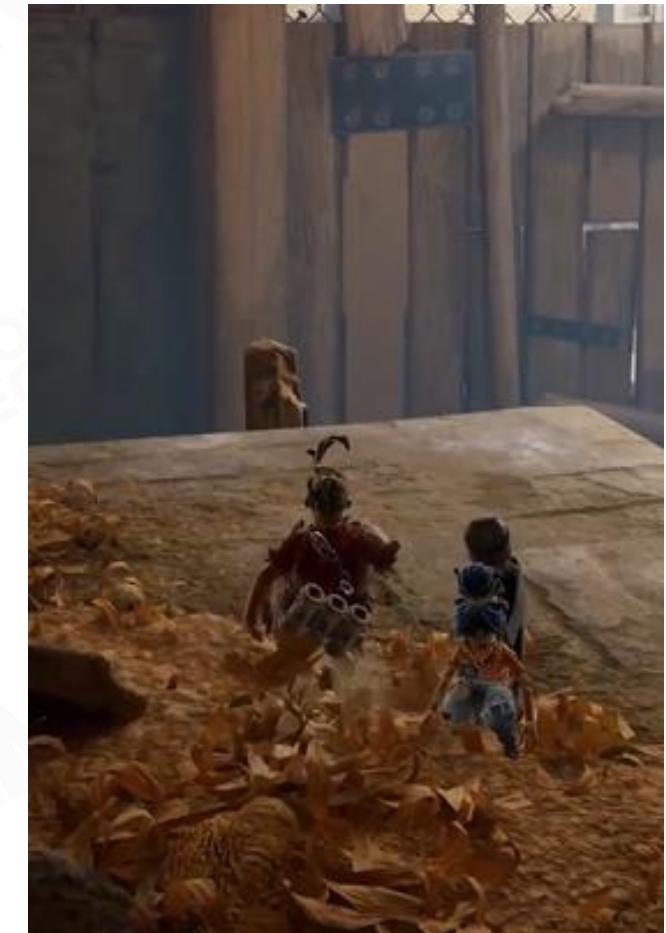


Diving



## Extended Character: Cooperate with other systems

Game effects, sound, environment interaction.





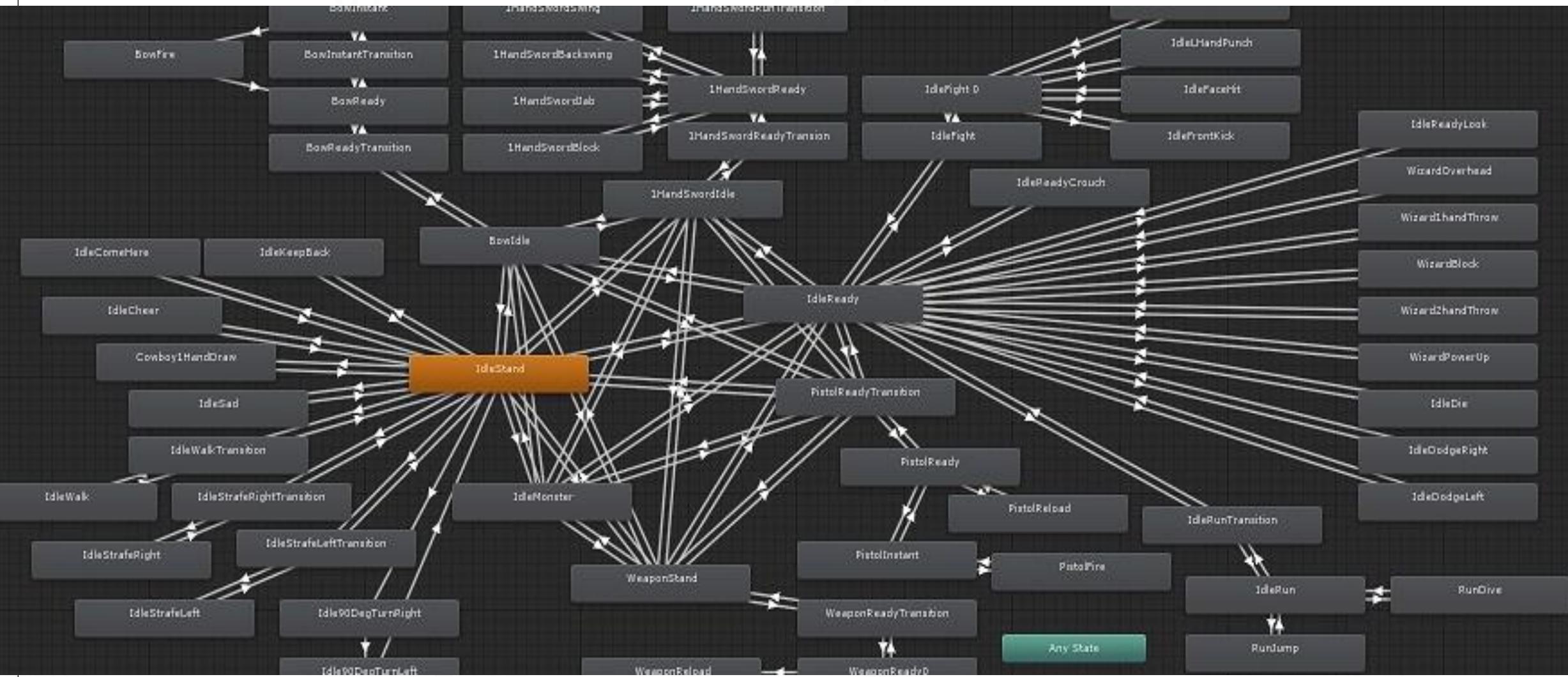
## Extended Character: More realistic motion with Physics

- Airflow
- Inertia tensor
- Torque
- ...





# Movement State Machine





## Control

Different input device  
Different game play



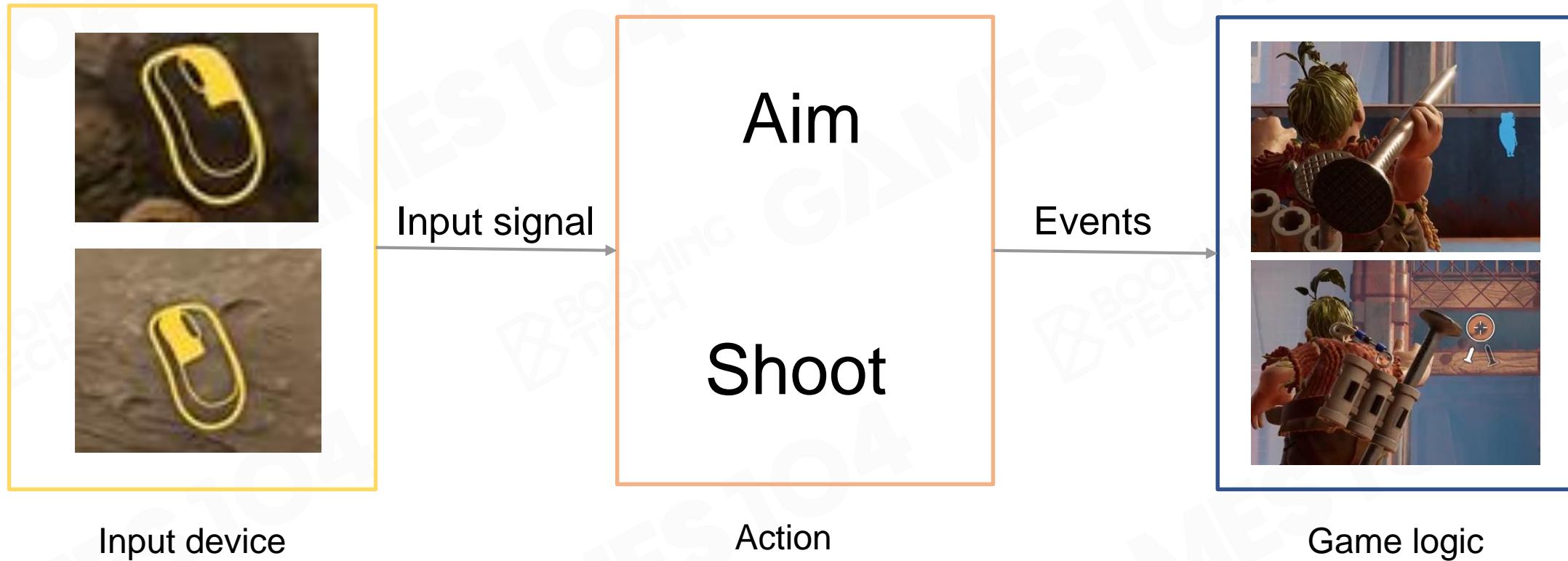


## A Good Example of Control





## From Input to Game Logic



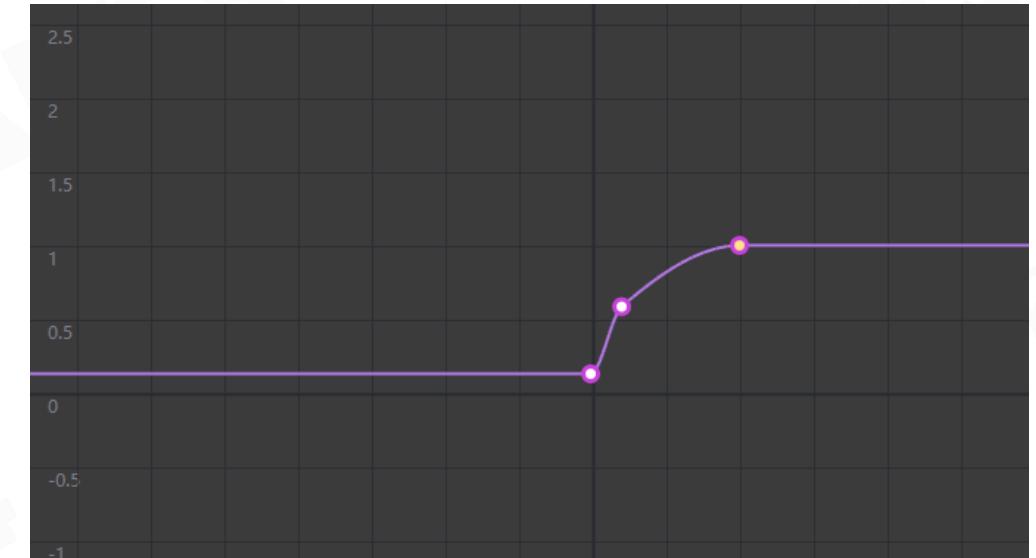


## Control: Zoom in and out





## Control: Aim Assist





## Control: Feedback

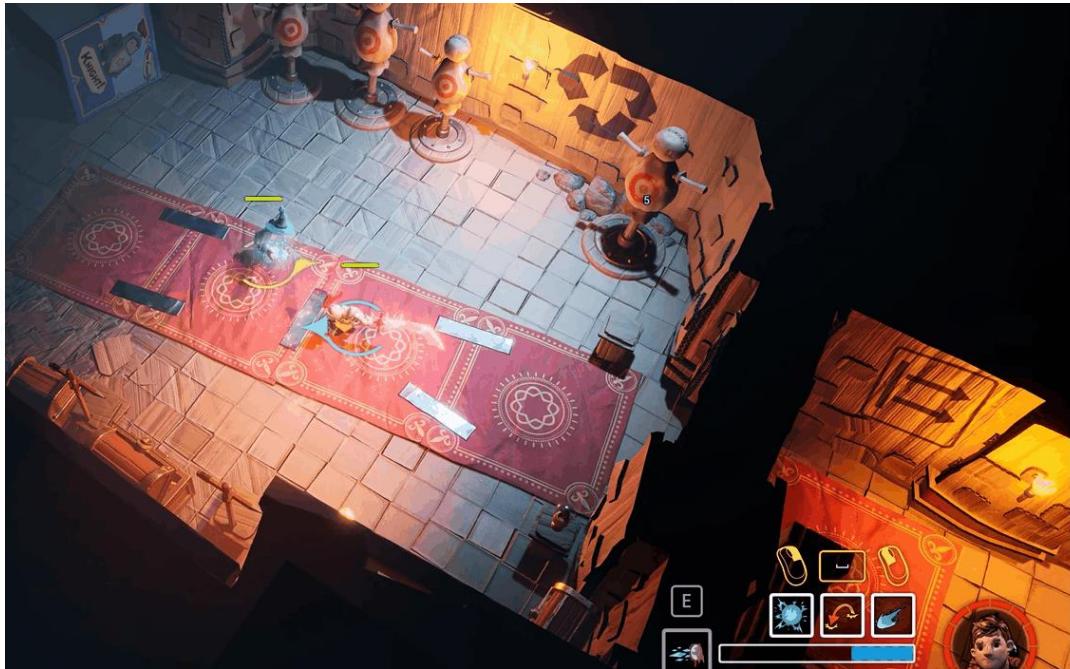




## Control: Context Awareness

### Context-sensitive controls

- The same input button produces different effects in different game scenarios.





## Control : Chord & Key Sequences

### Chords

- when pressed at the same time, produce a unique behavior in the game

### Key Sequences

- Gesture detection is generally implemented by keeping a brief history of the HID actions performed by the player





## Camera: Subjective Feelings





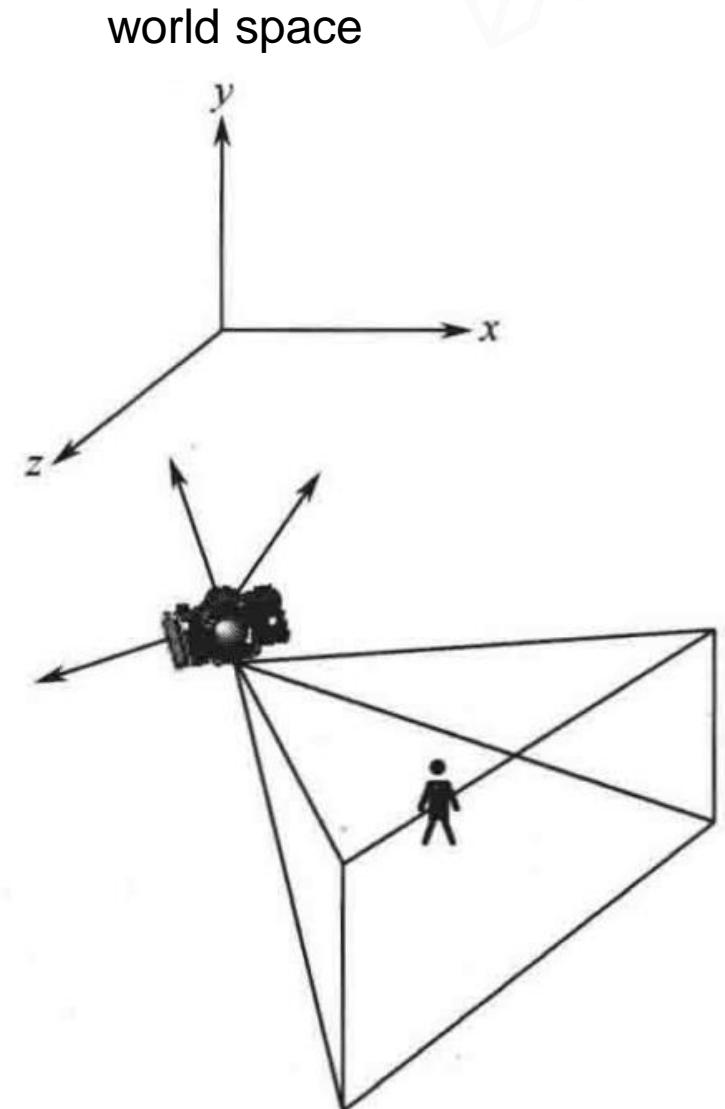
## Camera Basic: POV & FOV

### POV (point of view)

- determines the position of the player to observe

### FOV (field of view)

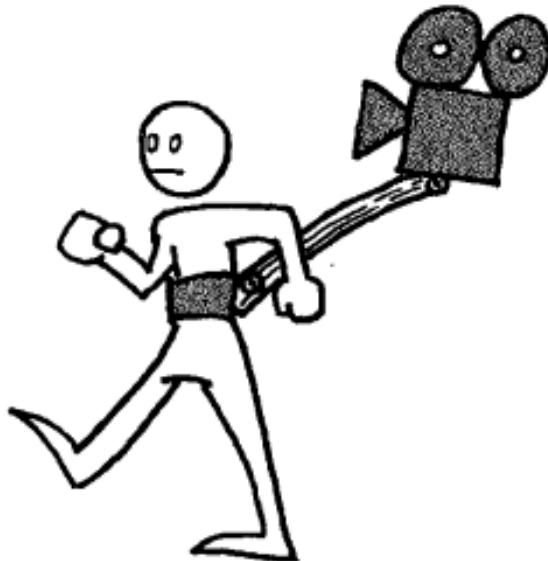
- determines the size of the player's viewing Angle





## Camera Binding

Using POV and rotation to bind.





## Camera Control



Spring Arm



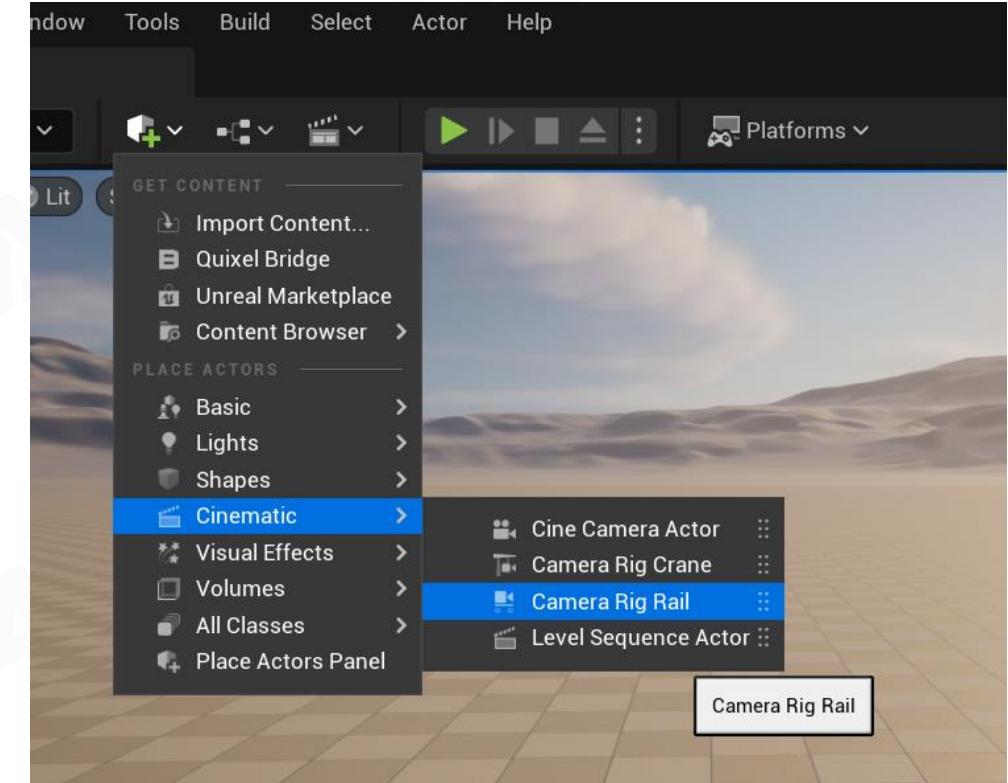
focusing  
FOV&distance Curve



## Camera Track



Camera Track



Scene Editor



## Camera Effects

Provide the camera with more post-visual effects, such as filters and shake.



Camera Shake

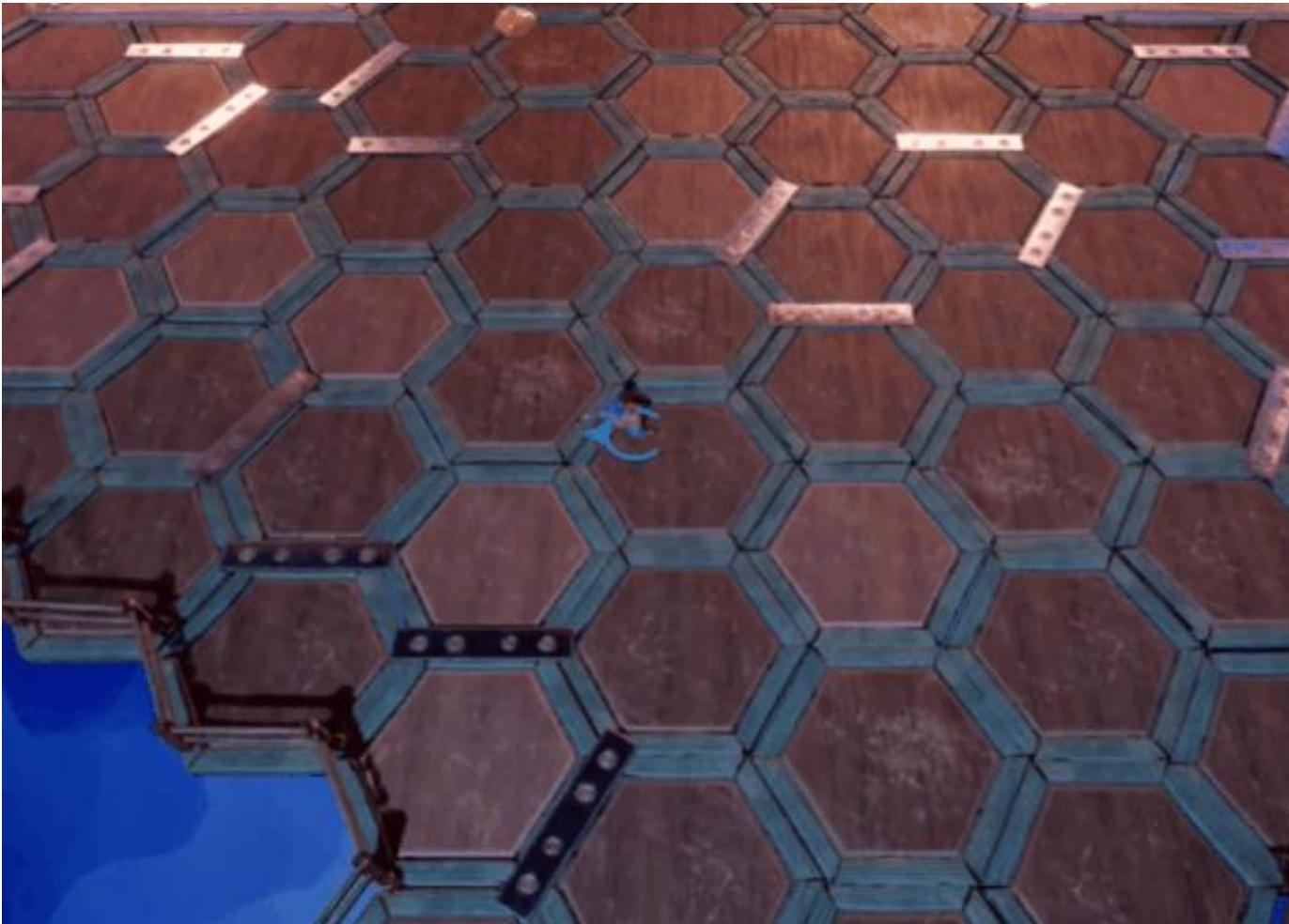


Camera Filter





## Many Cameras : Camera Manager



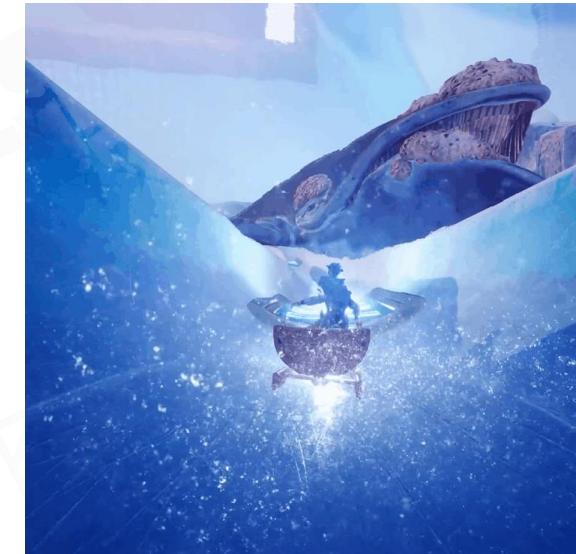
Camera Switch



## Camera: Subjective Feelings

Complex effects are often achieved by multiple base adjustments.  
To create a sense of speed as an example, we can do:

- Add lines in the speed direction
- The character falls backwards
- The dynamic fuzzy
- Zoom in FOV (to speed up changes in screen content)



Speed  
Motion blur, magnify FOV



## Camera: Subjective Feelings

### loose feeling

- Relax camera movement



### Cinematic

- filter, motion, sound, narrator, model, animation, camera movement,...

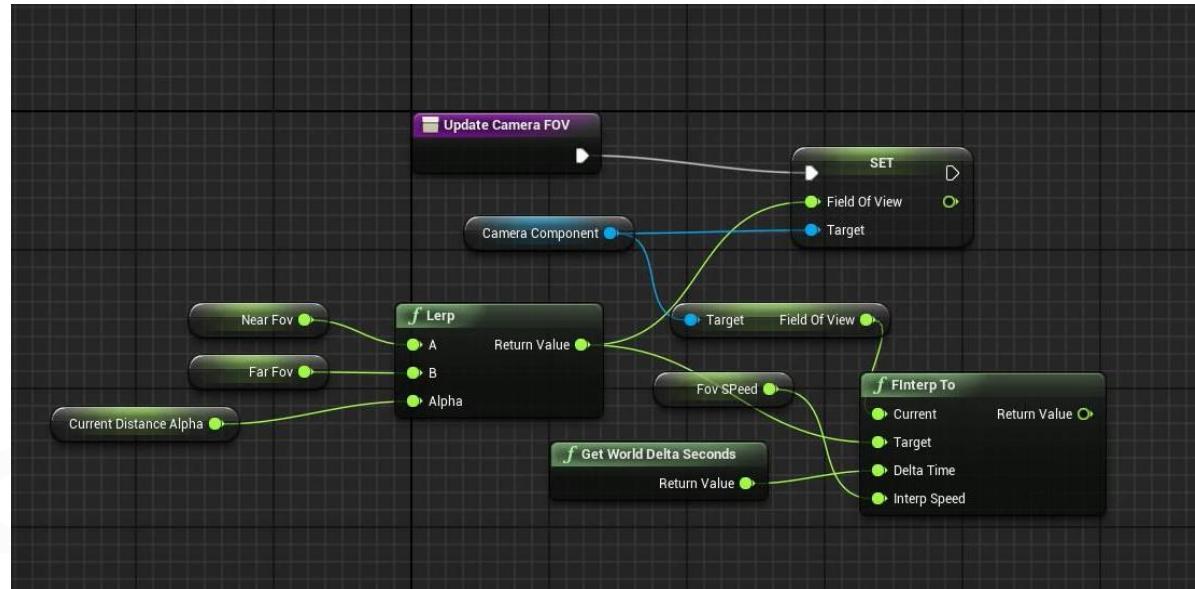




## Camera

For artists and designers to optimize the effect:

- Inheritable classes
- Function that can be accessed by Blueprint
- Adjustable parameters



**Actor Component**  
An ActorComponent is a reusable component added to any actor.

**Scene Component**  
A Scene Component is a component that has a transform and can be attached to other scene

**ALL CLASSES**

Search: CameraModifier\_CameraShake

CameraShakeBase

- DefaultCameraShakeBase
- MatineeCameraShake
- ChaosPhysicalMaterial
- CheatManager

All Actions for this Blueprint    Context Sensitive

Search

Camera

- f Deproject Screen to World
- f Get Camera View (FollowCamera)
- f Get View Projection Matrix
- f Play World Camera Shake
- f Project World to Screen
- f Set Aspect Ratio (FollowCamera)
- f Set Constraint Aspect Ratio (FollowCamera)
- f Set Field Of View (FollowCamera)
- f Set Ortho Far Clip Plane (FollowCamera)
- f Set Ortho Near Clip Plane (FollowCamera)
- f Set Ortho Width (FollowCamera)
- f Set Post Process Blend Weight (FollowCamera)
- f Set Projection Mode (FollowCamera)

Post Process

Lens

Mobile Depth of Field

- High Quality... (checkbox)
- Focal Region (0.0)
- Near Transition (300.0)
- Far Transition (500.0)
- Scale (0.0)
- Near Blur Size (15.0)
- Far Blur Size (15.0)

Advanced

Bloom

Exposure

Chromatic Aberration

Dirt Mask

Camera



Everything is Gameplay



## GamePlay

---

### References



## Event Mechanism

- Are Callbacks Always Asynchronous: <https://dev.to/marek/are-callbacks-always-asynchronous-bah>
- Unreal Engine Documentation Actor Communication:  
<https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ActorCommunication/>
- Publish-subscribe pattern: <https://ably.com/topic/pub-sub>
- Unreal Engine Event Dispatcher: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/UserGuide/EventDispatcher/>



## Script

- A Classification of Scripting Systems for Entertainment and Serious Computer Games, 2011:  
<http://eprints.bournemouth.ac.uk/20555/4/vsgames2011postprint.pdf>
- Programming Language and Compiler Benchmarks: <https://programming-language-benchmarks.vercel.app>
- Lua official site: <https://www.lua.org/>
- LuaJIT official site: <https://luajit.org/>
- Unity C# Scripting Manual: <https://docs.unity3d.com/Manual/ScriptingSection.html>
- Implementing a Scripting Engine:  
[https://www.flipcode.com/archives/Implementing\\_A\\_Scripting\\_Engine-Part\\_1\\_Overview.shtml](https://www.flipcode.com/archives/Implementing_A_Scripting_Engine-Part_1_Overview.shtml)
- Scripting Virtual Machine and Bytecode: <http://gameprogrammingpatterns.com/byticode.html>



## Visual Scripting

- Unreal Engine Blueprints Visual Scripting: <https://docs.unrealengine.com/5.0/en-US/blueprints-visual-scripting-in-unreal-engine/>
- Unity Visual Scripting: <https://unity.com/features/unity-visual-scripting>
- Programming Language Design: <https://cs.lmu.edu/~ray/notes/languagedesignnotes/>
- Blueprint Merge Tool: <https://github.com/KennethBuijsen/MergeAssist>



## Gameplay and 3C

- *Level Up! The Guide to Great Video Game Design 2nd Edition*, Scott Rogers, 2014
- *Game Engine Architecture Third Edition*, Jason Gregory, 2018
- The 3Cs of Game Development: <https://www.pluralsight.com/blog/film-games/character-controls-camera-3cs-game-development>



## Lecture 15 Contributor

- |        |           |         |        |
|--------|-----------|---------|--------|
| - 一将   | - 爵爷      | - 喵小君   | - QIUU |
| - 蓑笠翁  | - Jason   | - 果蝇    | - C佬   |
| - 烛哥   | - 砚书      | - 梨叔    | - 阿乐   |
| - 玉林   | - BOOK    | - Shine | - 阿熊   |
| - 小老弟  | - MANDY   | - 邓导    | - CC   |
| - 建辉   | - Unicorn | - Judy  | - 大喷   |
| - Hoya | - 灰灰      | - Leon  | - 金大壮  |



# Q&A



# Enjoy ;) Coding



Course Wechat

*Follow us for  
further information*