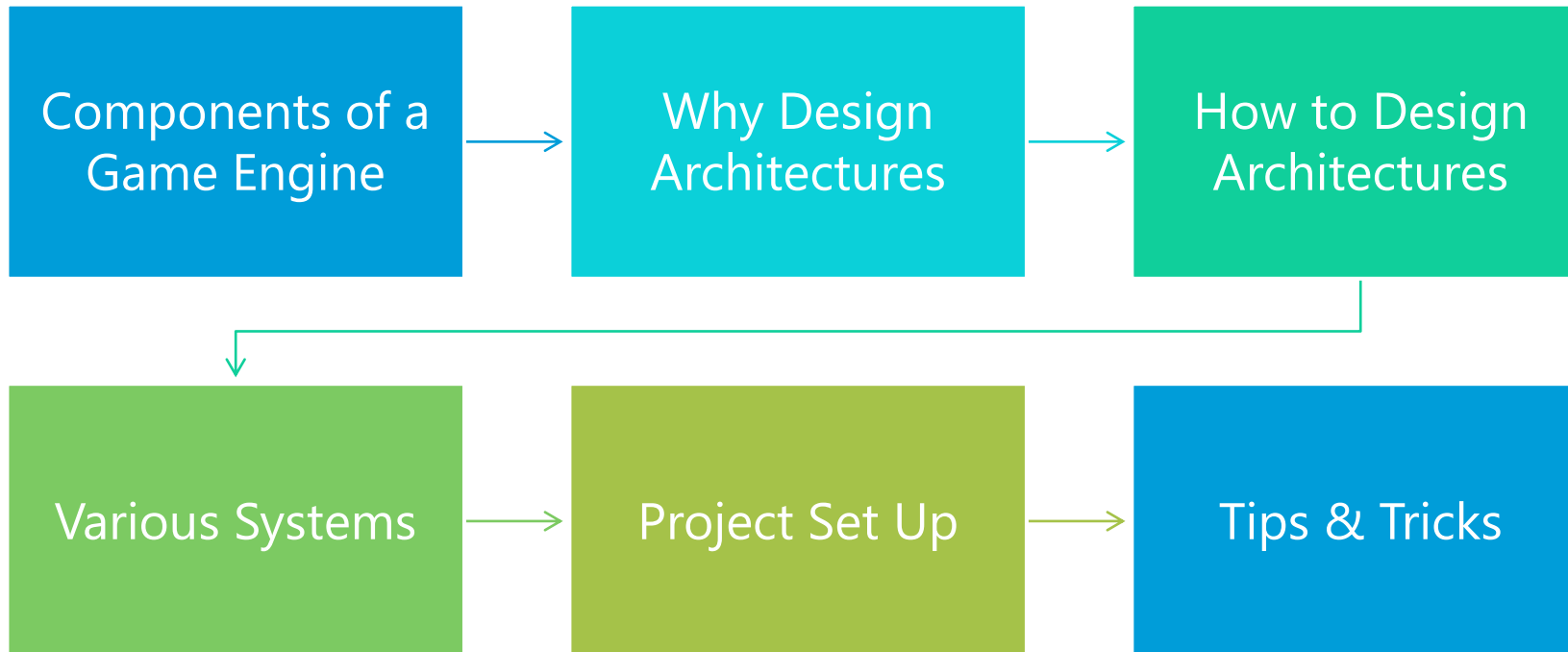


Intro to Game Engine Architecture

Architecture Engine Club Summer 2023

Topics



What's a game engine made of?



Graphics



Game Object
System



Event/Messaging
System



Audio



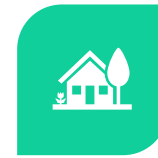
Resource
Management



Physics &
Collision



Frame Rate
Controller



Scene
Management



Animation



Input



Scripting



Editor



Serialization



Logging

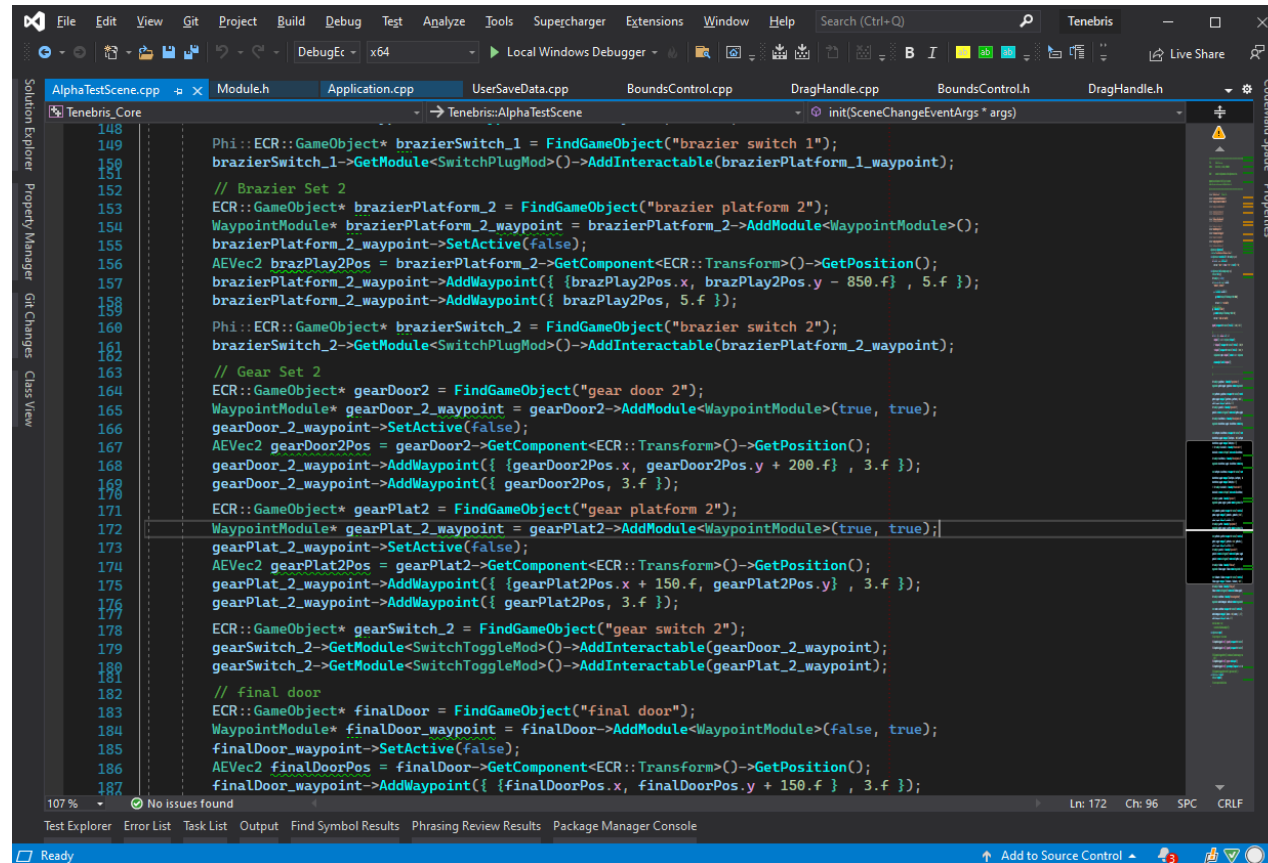


Memory
Manager



Etc.

Why care about architecture?



The screenshot shows a Visual Studio IDE with a C++ project named 'Tenebris'. The code is organized into a single file, 'AlphaTestScene.cpp', which contains a large block of initialization code for a game scene. The code is divided into sections for different game objects, such as 'Brazier Set 2', 'Gear Set 2', and 'final door'. Each section contains multiple lines of code for finding game objects, adding waypoints, and setting up interactables. The code is not modular, with all logic contained within a single file, which is a sign of poor architecture. The IDE interface includes a menu bar, a toolbar, a Solution Explorer on the left, and a Properties window on the right. The status bar at the bottom shows 'Ready' and 'Add to Source Control'.

```
148 Phi::ECR::GameObject* brazierSwitch_1 = FindGameObject("brazier switch 1");
149 brazierSwitch_1->GetModule<SwitchPlugMod>()->AddInteractable(brazierPlatform_1_waypoint);
150
151 // Brazier Set 2
152 ECR::GameObject* brazierPlatform_2 = FindGameObject("brazier platform 2");
153 WaypointModule* brazierPlatform_2_waypoint = brazierPlatform_2->AddModule<WaypointModule>();
154 brazierPlatform_2_waypoint->SetActive(false);
155 AEVec2 brazPlay2Pos = brazierPlatform_2->GetComponent<ECR::Transform>()->GetPosition();
156 brazierPlatform_2_waypoint->AddWaypoint({ {brazPlay2Pos.x, brazPlay2Pos.y - 850.f}, 5.f });
157 brazierPlatform_2_waypoint->AddWaypoint({ brazPlay2Pos, 5.f });
158
159 Phi::ECR::GameObject* brazierSwitch_2 = FindGameObject("brazier switch 2");
160 brazierSwitch_2->GetModule<SwitchPlugMod>()->AddInteractable(brazierPlatform_2_waypoint);
161
162 // Gear Set 2
163 ECR::GameObject* gearDoor2 = FindGameObject("gear door 2");
164 WaypointModule* gearDoor_2_waypoint = gearDoor2->AddModule<WaypointModule>(true, true);
165 gearDoor_2_waypoint->SetActive(false);
166 AEVec2 gearDoor2Pos = gearDoor2->GetComponent<ECR::Transform>()->GetPosition();
167 gearDoor_2_waypoint->AddWaypoint({ {gearDoor2Pos.x, gearDoor2Pos.y + 200.f}, 3.f });
168 gearDoor_2_waypoint->AddWaypoint({ gearDoor2Pos, 3.f });
169
170 ECR::GameObject* gearPlat2 = FindGameObject("gear platform 2");
171 WaypointModule* gearPlat_2_waypoint = gearPlat2->AddModule<WaypointModule>(true, true);
172 gearPlat_2_waypoint->SetActive(false);
173 AEVec2 gearPlat2Pos = gearPlat2->GetComponent<ECR::Transform>()->GetPosition();
174 gearPlat_2_waypoint->AddWaypoint({ {gearPlat2Pos.x + 150.f, gearPlat2Pos.y}, 3.f });
175 gearPlat_2_waypoint->AddWaypoint({ gearPlat2Pos, 3.f });
176
177 ECR::GameObject* gearSwitch_2 = FindGameObject("gear switch 2");
178 gearSwitch_2->GetModule<SwitchToggleMod>()->AddInteractable(gearDoor_2_waypoint);
179 gearSwitch_2->GetModule<SwitchToggleMod>()->AddInteractable(gearPlat_2_waypoint);
180
181 // final door
182 ECR::GameObject* finalDoor = FindGameObject("final door");
183 WaypointModule* finalDoor_waypoint = finalDoor->AddModule<WaypointModule>(false, true);
184 finalDoor_waypoint->SetActive(false);
185 AEVec2 finalDoorPos = finalDoor->GetComponent<ECR::Transform>()->GetPosition();
186 finalDoor_waypoint->AddWaypoint({ {finalDoorPos.x, finalDoorPos.y + 150.f}, 3.f });
187
```

Key Parts of an Engine Architecture



Game Entities



Systems



Game Loops



Etc

Designing an Architecture

DESIGNING AN ARCHITECTURE

Mindset

No such thing as the best architecture

Architecture design is an art form

But there is bad art

DESIGNING AN ARCHITECTURE

Goals



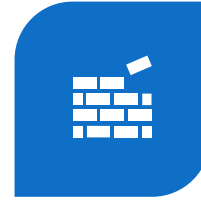
Easy to Use



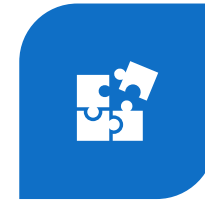
Data Driven



Clean



Not Obstructive



Extensible



Modular

DESIGNING AN ARCHITECTURE

Code Smells

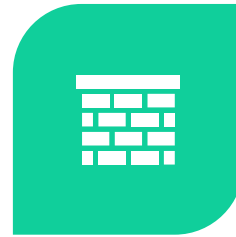
Symptoms of a problems in the architecture.



Rigidity



Fragility



Immobility



Viscosity

DESIGNING AN ARCHITECTURE

Conceptualise Usage



Conceptualise

How would you want to use the system?



Research

Is it possible to do it that way?



Plan

Would it get messy?

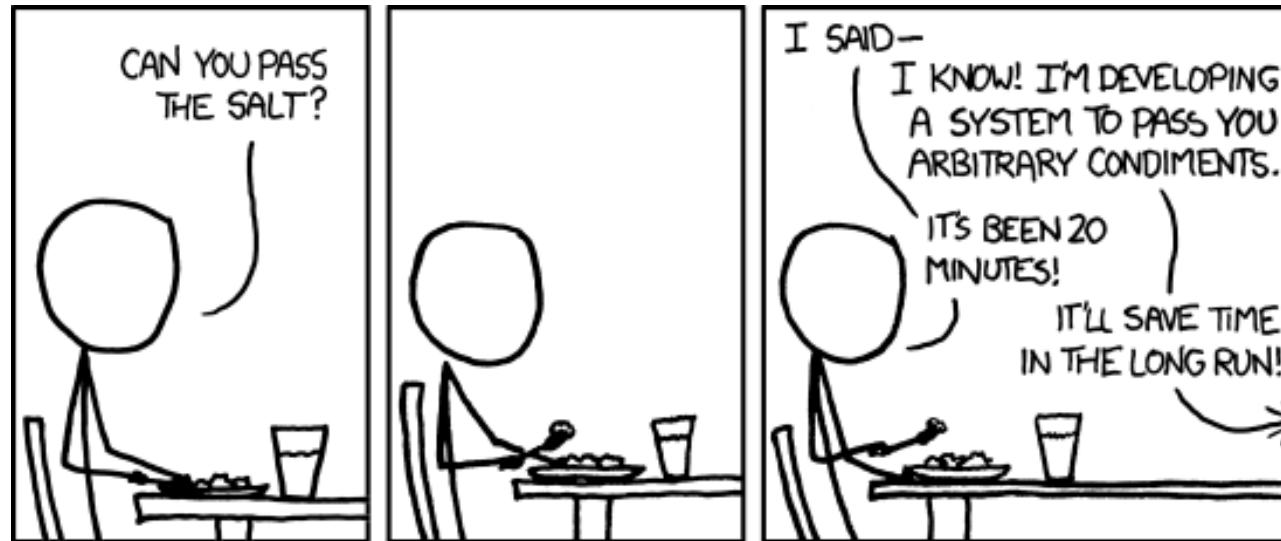


Test

Does it work in practice?

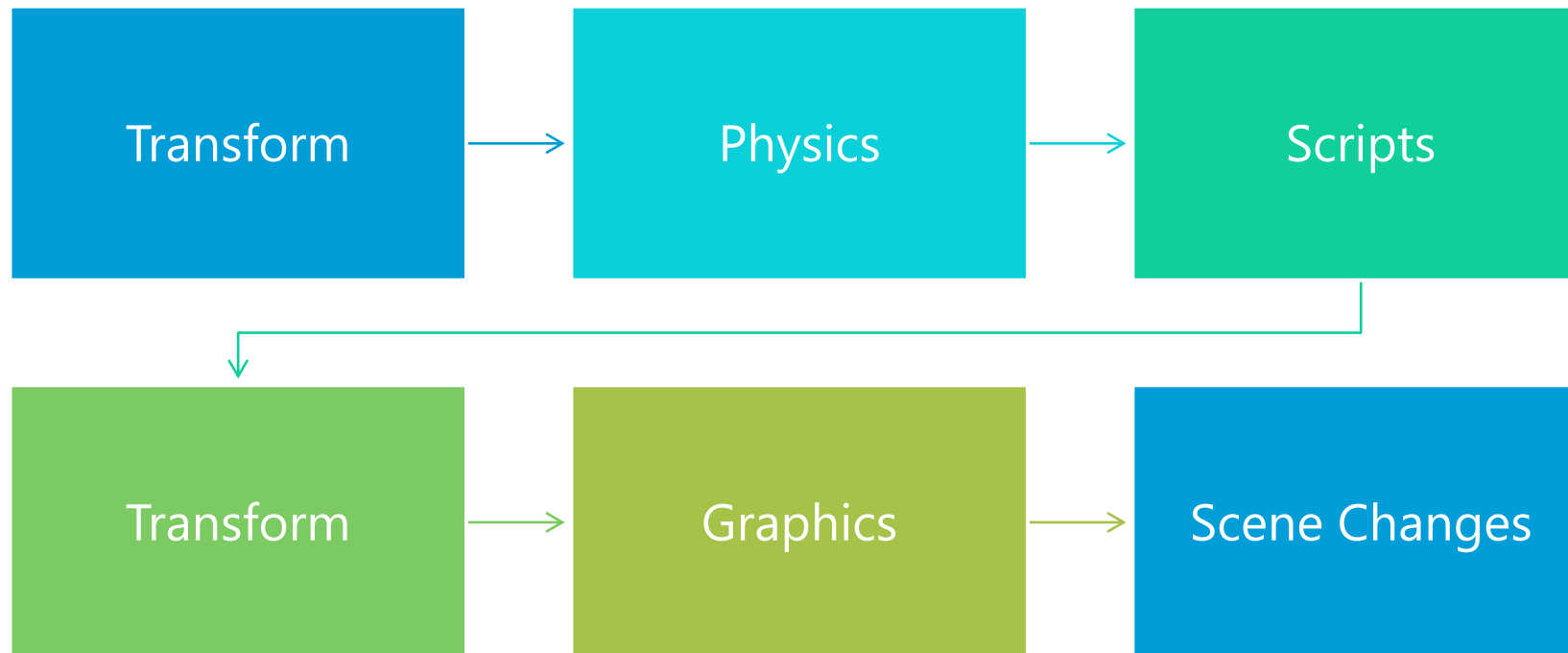
DESIGNING AN ARCHITECTURE

Abstractions & Overengineering



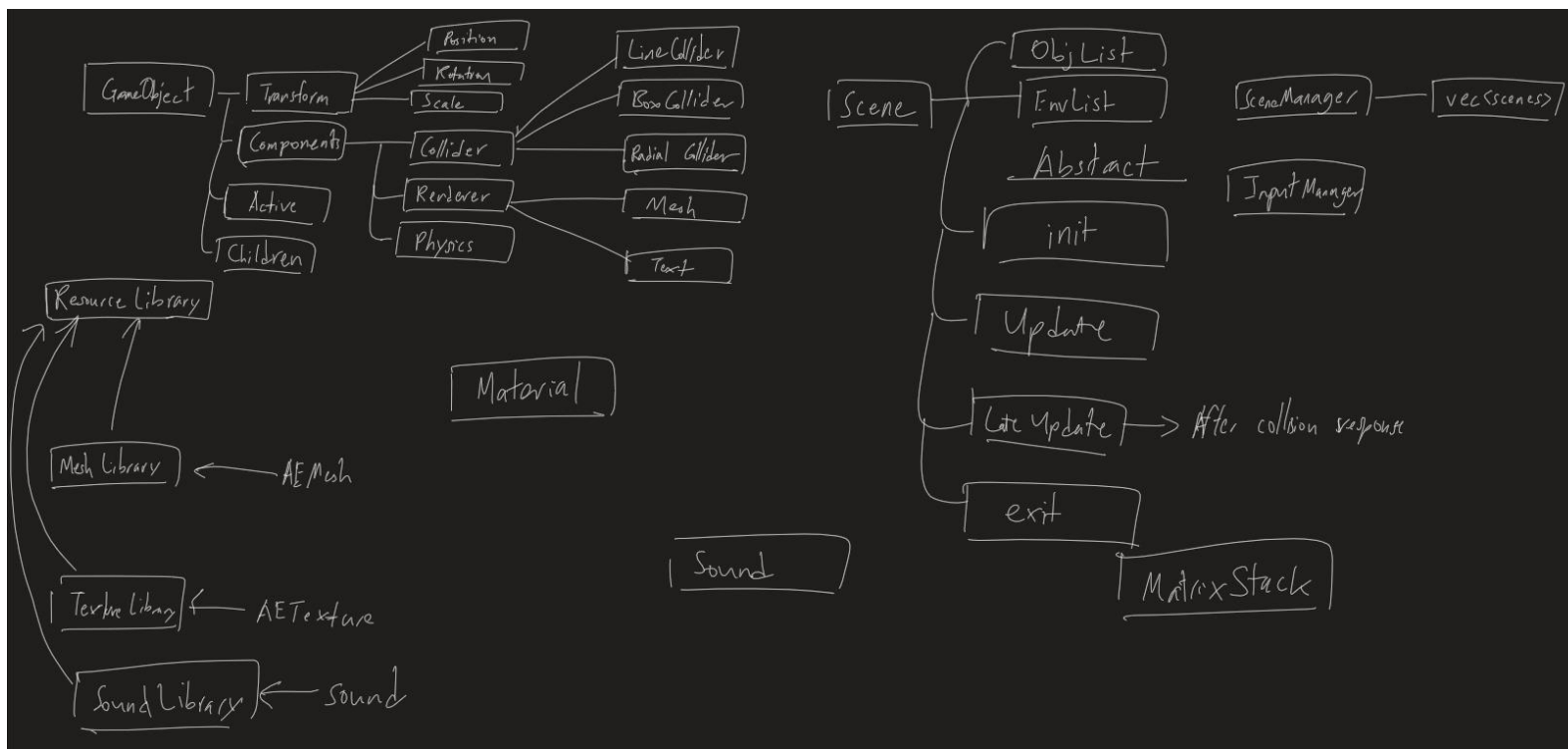
DESIGNING AN ARCHITECTURE

Sequence of System Updates



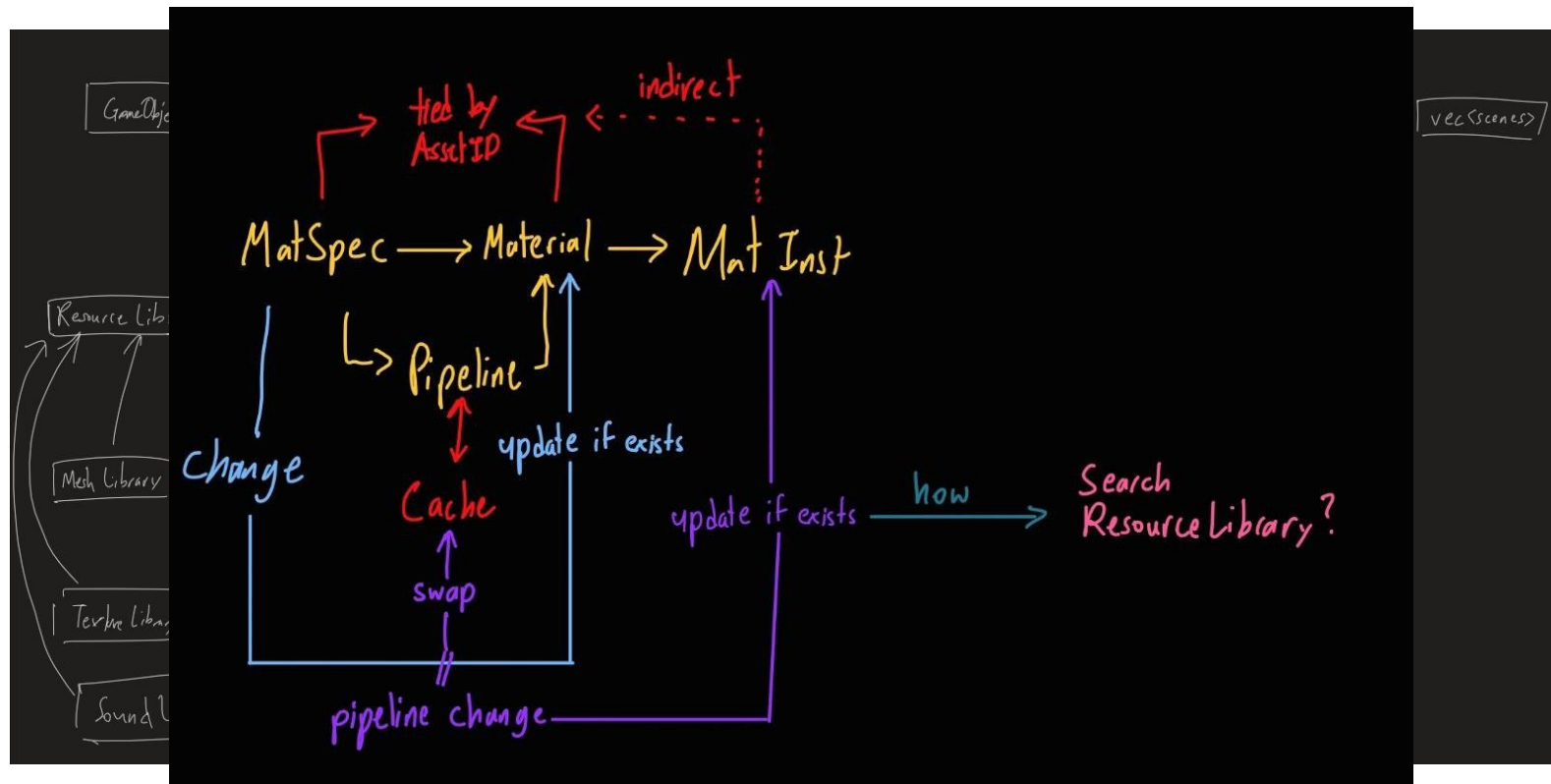
DESIGNING AN ARCHITECTURE

Draw Diagrams



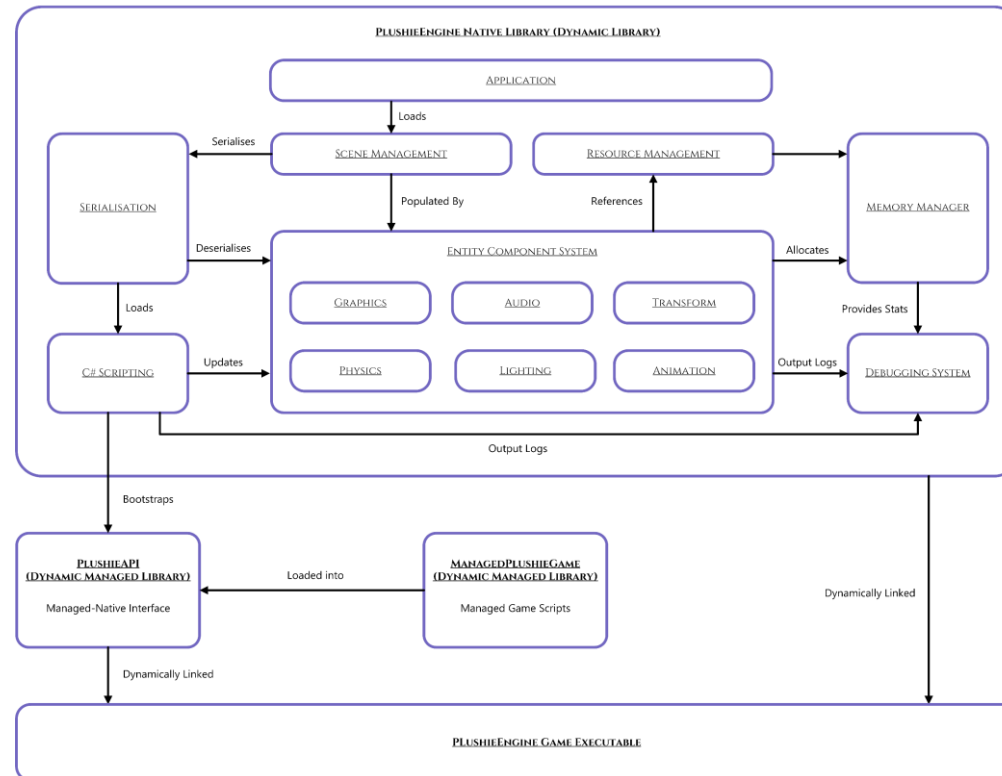
DESIGNING AN ARCHITECTURE

Draw Diagrams



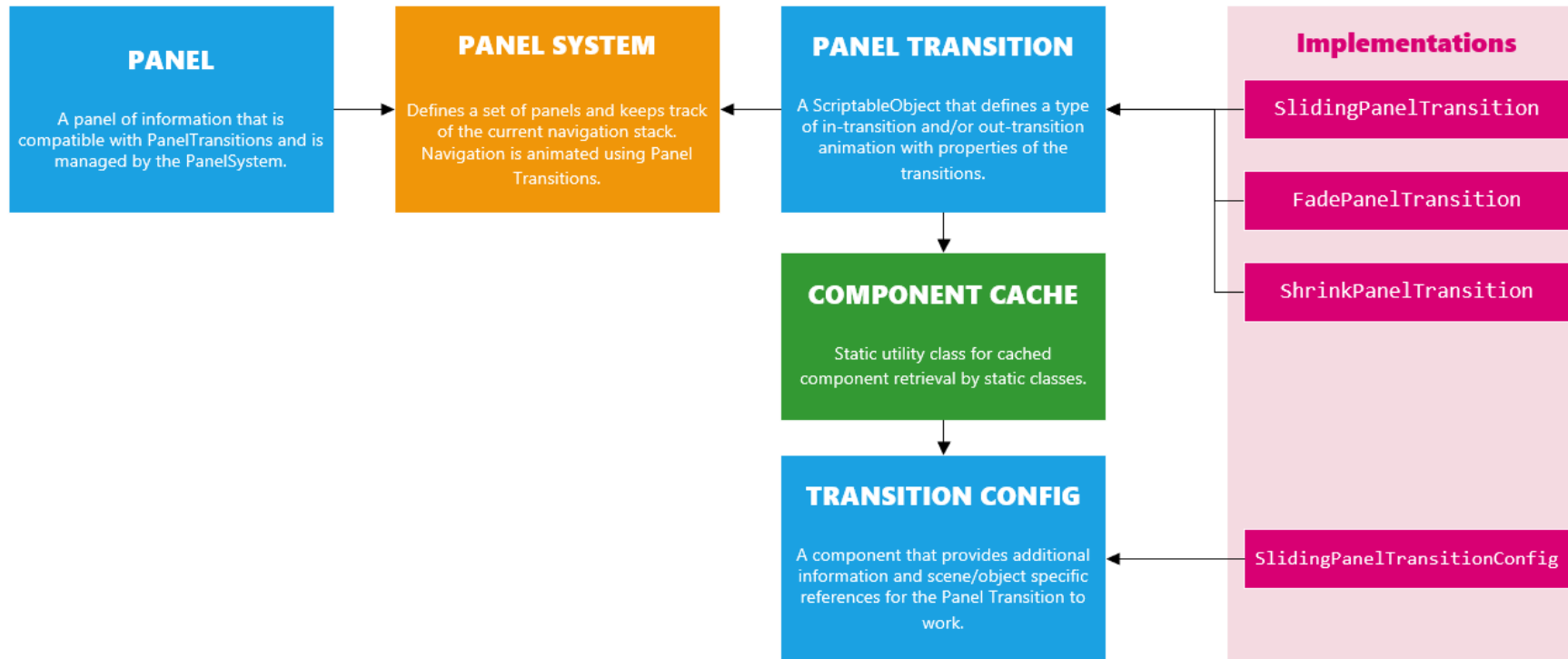
DESIGNING AN ARCHITECTURE

Draw Diagrams



DESIGNING AN ARCHITECTURE

Draw Diagrams

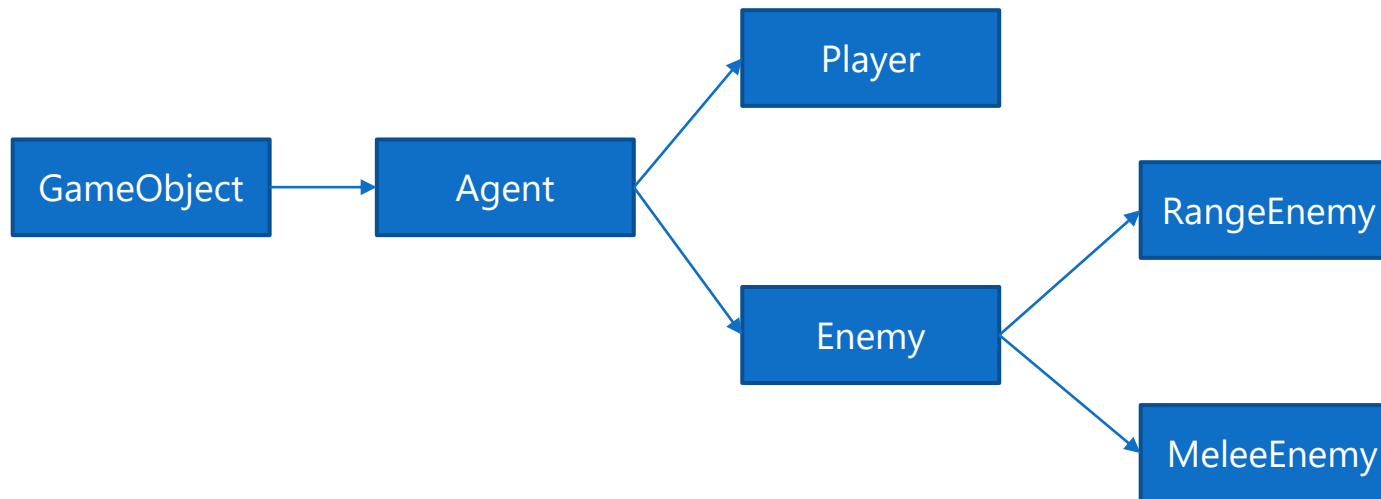


Individual Systems

GAME OBJECTS SYSTEM

Inheritance-Based

Simplest implementation.



GAME OBJECTS SYSTEM

Inheritance-Based

Simplest implementation.

Pros

- Simple to implement
- Intuitive to work with
- Easy to debug

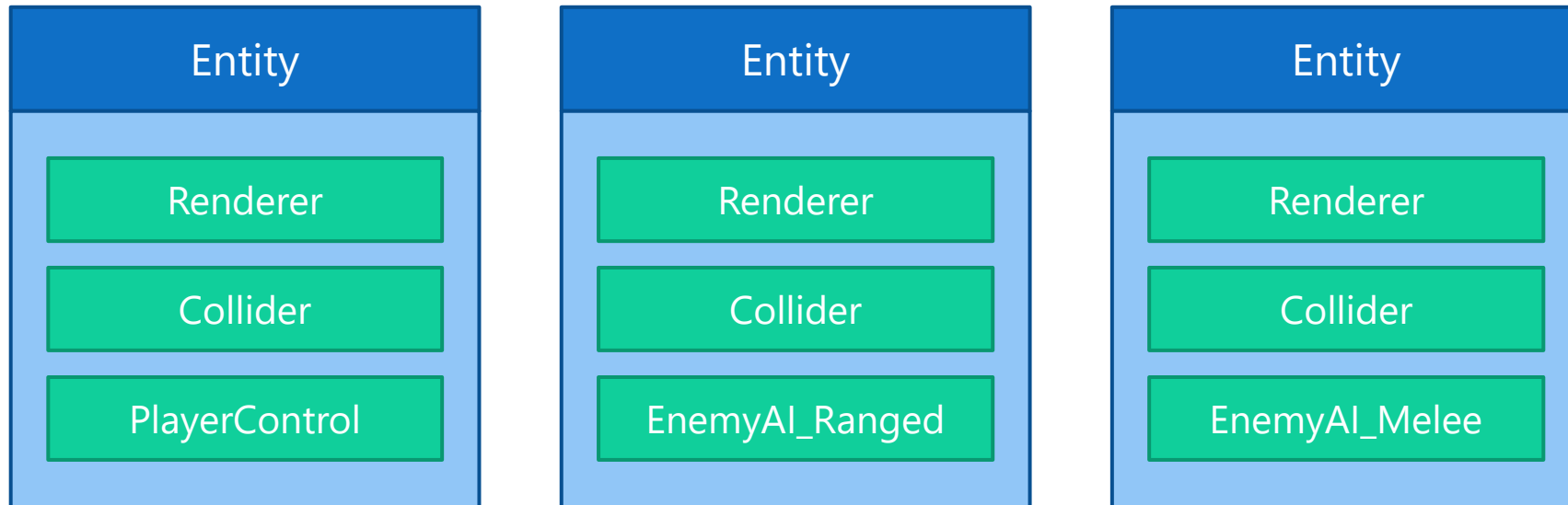
Cons

- Hard to extend
- Multiple virtual calls
 - Slow
- Spaghetti
- Hard to visualise relations

GAME OBJECTS SYSTEM

Entity Component

Composition based system.



GAME OBJECTS SYSTEM

Entity Component

Composition based system.

Pros

- Slightly harder to implement
- Intuitive to work with
- Code reuse via components
- Cleaner design

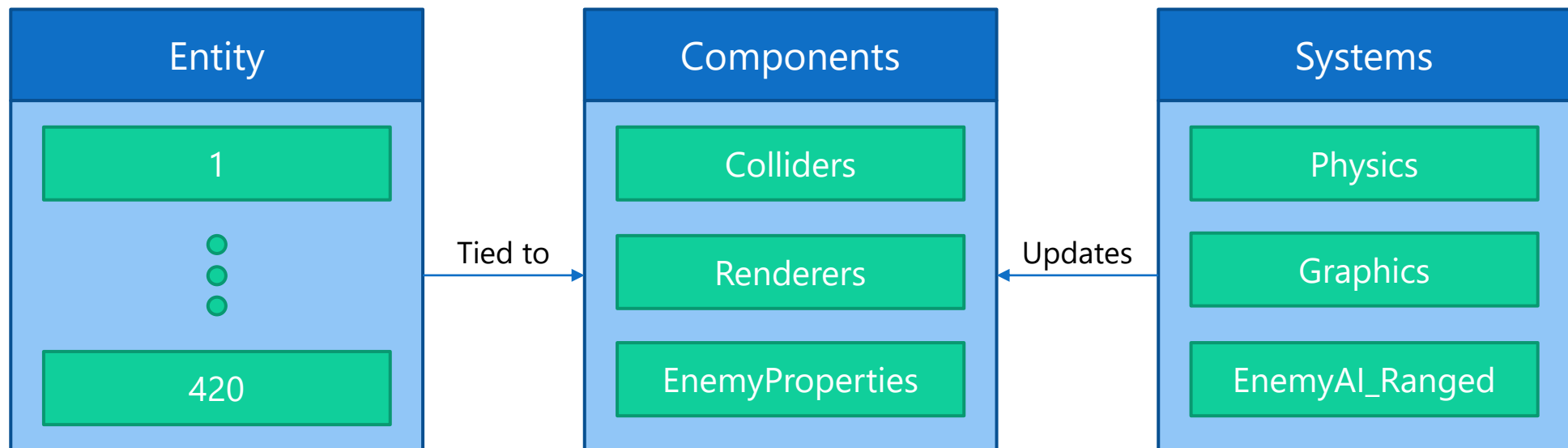
Cons

- Multiple virtual calls
- Bad cache locality
- Slow

GAME OBJECTS SYSTEM

Entity Component System

Data-driven component-based architecture.



Will be talked in-depth about in a future lecture.

GAME OBJECTS SYSTEM

Entity Component System

Data-driven component-based architecture.

Pros

- Maximizes cache locality
 - Very fast
- Clear separation of concerns
- Very clean design

Cons

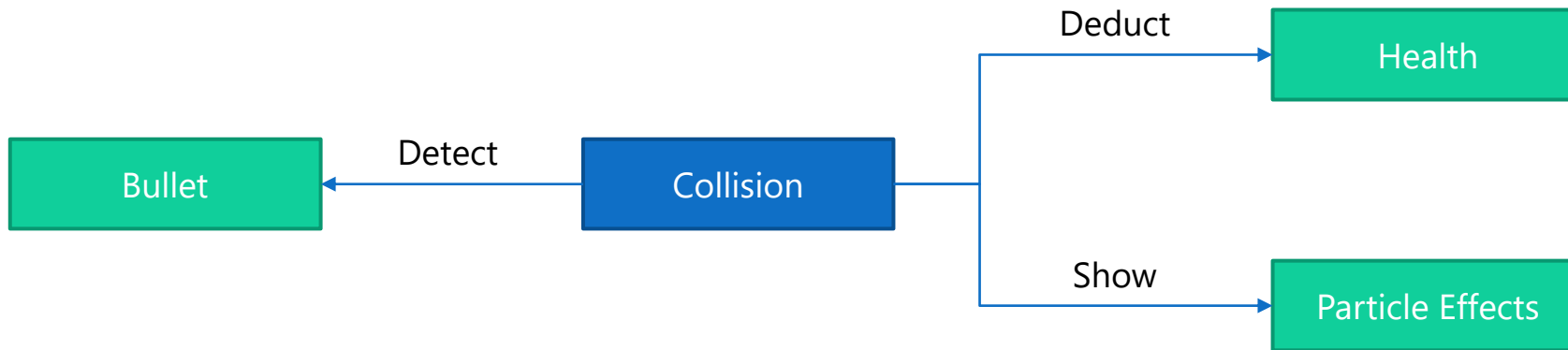
- Can be hard to debug
- Difficult to implement
- Complicated to understand

Will be talked in-depth about in a future lecture.

EVENT/MESSAGING SYSTEM

Why have one?

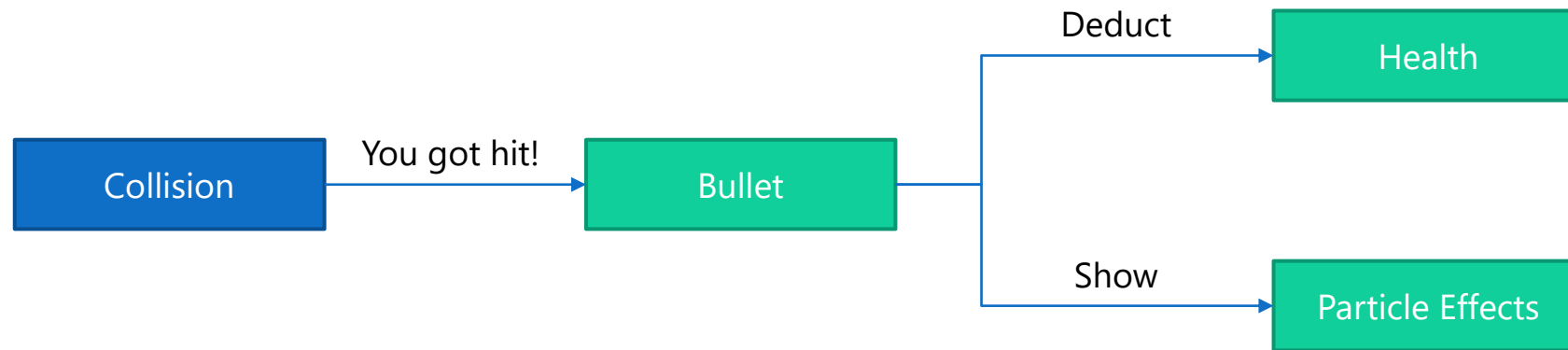
A way to separate gameplay logic from systems.



EVENT/MESSAGING SYSTEM

Observer Pattern

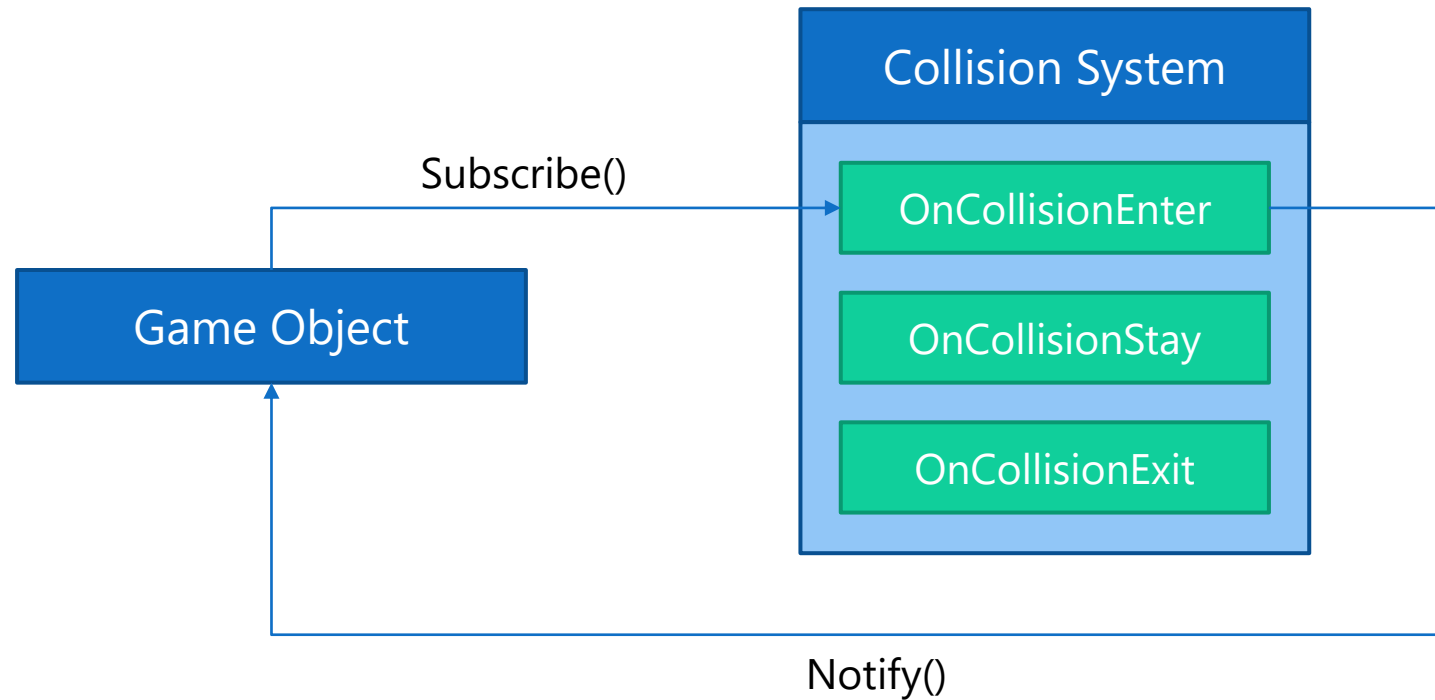
"Let me know when it happens"



EVENT/MESSAGING SYSTEM

Observer Pattern

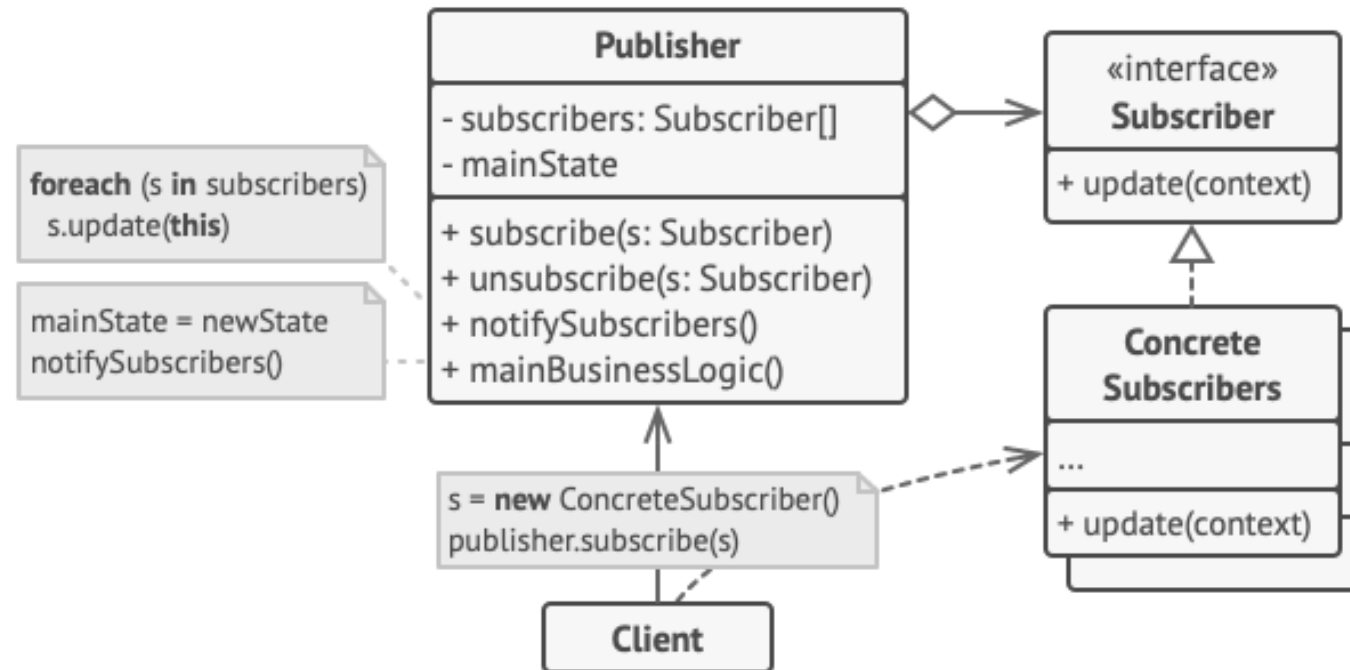
“Let me know when it happens”



EVENT/MESSAGING SYSTEM

Observer Pattern

“Let me know when it happens”



EVENT/MESSAGING SYSTEM

Observer Pattern

"Let me know when it happens"

Pros

- Systems are decoupled from gameplay logic
- Easily extensible

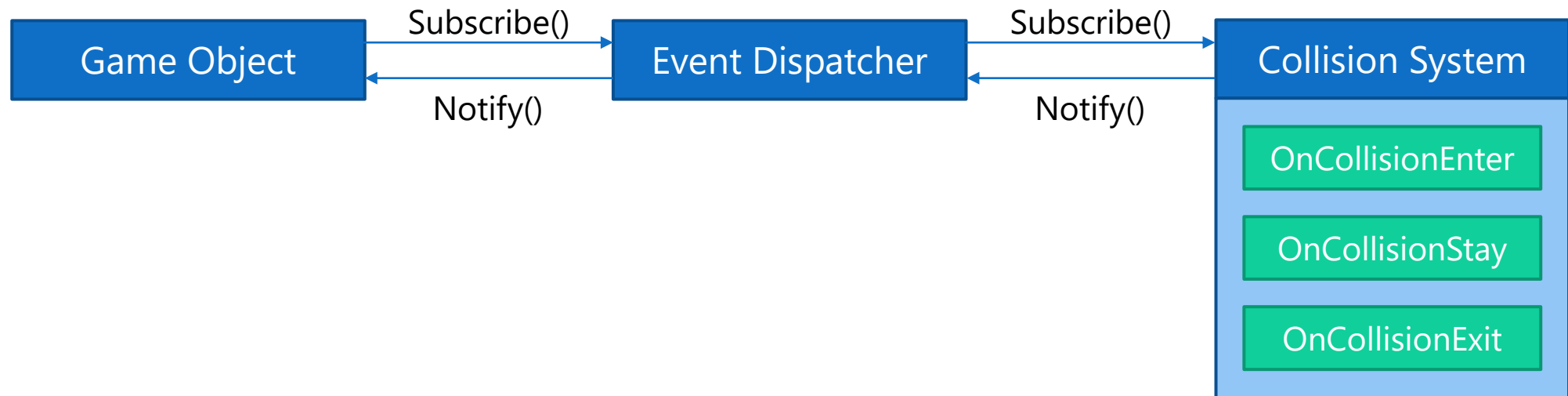
Cons

- Cannot ensure execution order
- Requires dynamic dispatch
- Gameplay logic still needs to be able to interact with the system

EVENT/MESSAGING SYSTEM

Dispatcher

Additional layer to the observer pattern.



EVENT/MESSAGING SYSTEM

Dispatcher

Additional layer to the observer pattern.

Pros

- Systems are decoupled from gameplay logic **and systems**
- Easily extensible
- Events can be deferred
- Order can be controlled

Cons

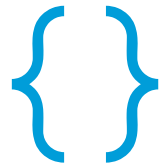
- Requires dynamic dispatch

Setting Up Your Project

SETTING UP YOUR PROJECT

Decide Your SDK

Best to figure these out **before writing any code.**



C++ Standard



Types of Projects



Dependencies

SETTING UP YOUR PROJECT

Standardise Your Conventions

Assets in 4 different folder paths? Not fun.

Source Control

Build Directories

Intermediate
Directories

Asset
Directories

Extern
Dependencies
Directories

File Names

Folder Names

.h and .cpp in
the same
folder?

SETTING UP YOUR PROJECT

Source Control

Are you using it effectively?



Remote Host



Client



Branches



Tags



Merging

IN SUMMARY

Key Takeaways



Start Planning



Start
Discussing



Start
Prototyping



Ask Questions
Early

Tips & Tricks

TIPS & TRICKS

Document, Document, Document

For your team **and yourself** who will probably forget.

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y;                // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 );        // what the f***?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
    // y = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be removed

    return y;
}
```

TIPS & TRICKS

Document, Document, Document

For your team **and yourself** who will probably forget.

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5f;

    x2 = number * 0.5f;
    y  = number;
    i  = * ( long * ) &y;
    i  = 0x5f3759df - i;
    y  = * ( float * ) &i;
    y  = y * ( threehalfs * ( 1.0f - y * y ) );
    // y = y * ( threehalfs * ( 1.0f - y * y ) );

    return y;
}
```

What

- What it is
- How to use it
- Side effects
- What to watch out for

at bit level hacking

is can be removed

TIPS & TRICKS

Const When You Can

Prevents things from being changed if they aren't supposed to.

```
const const const int const const const AN_INT = 420;
```


TIPS & TRICKS

Read Up on Design Patterns



<https://gameprogrammingpatterns.com/>

<https://refactoring.guru/>

TIPS & TRICKS

Use `<chrono>`

Stop using `<ctime>`

```
#include <iostream>
#include <chrono>

using Time      = std::chrono::steady_clock;
using Milliseconds = std::chrono::duration<float, std::milli>;
using TimePoint  = std::chrono::time_point<Time, Milliseconds>;

int main()
{
    const TimePoint START_TIME = Time::now();
    while (true)
    {
        const Milliseconds TIME_PASSED = Time::now() - START_TIME;
        std::cout << "Time since start: " << TIME_PASSED.count() << " ms" << std::endl;
    }
}
```

TIPS & TRICKS

Unit Types

Makes converting between units easy.

```
struct Degrees
{
    public:
        Degrees(float f);
        Degrees(Radians r);

        explicit operator Radians() const;
        operator float() const;

    private:
        float angle = 0.0f;
};
```

```
struct Radians
{
    public:
        Radians(float f);
        Radians(Degrees r);

        explicit operator Degrees() const;
        operator float() const;

    private:
        float angle = 0.0f;
};
```

TIPS & TRICKS

Unit Types

Makes converting between units easy.

```
struct Degrees
{
    public:
        Degrees(float f)
        Degrees(Radians r)

        explicit operator float() const;
        operator float() const;

    private:
        float angle = 0.0f;
};
```

```
void RotateObject(Radians r);

const Degrees ANGLE = 45.0f;
RotateObject(ANGLE);
```

```
struct Radians
{
    public:
        Radians(float f)
        Radians(Degrees d)

        explicit operator float() const;
        operator float() const;

    private:
        float angle = 0.0f;
};
```

TIPS & TRICKS

Loading Screens

You don't need multi-threading.

Frame 1	Frame 2	Frame 3	Frame 4
Texture1	Texture3	Audio2	Texture6
Texture2	Texture4	Audio3	Texture7
Audio1	Texture5	Audio4	Audio5

Thanks for Listening

Any Questions?