ECS在Unity中的应用

樊松阳
UVP
微软MVP
腾讯高级工程师

- ECS的简介
- ECS的优势
- ECS的应用
- ECS的技巧

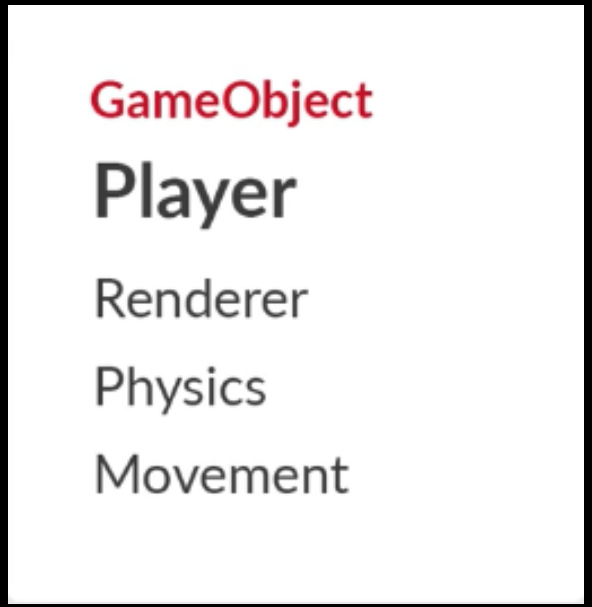
Entity Component System

- Entity – Component的挂载对象，本质上是Index
- Component - 数据集合，本质上是Data，无方法
- System - 行为集合，本质上是 Behaviour，无状态

ECS的本质：行为与状态分离

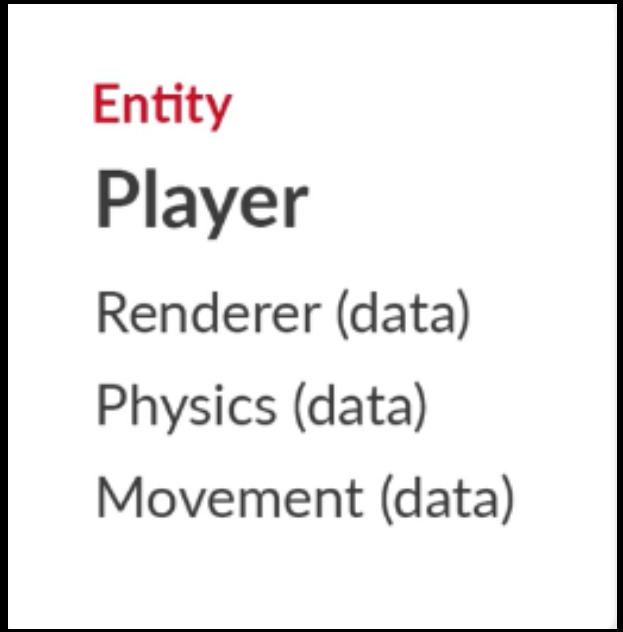
GameObject

数据与逻辑混在一起



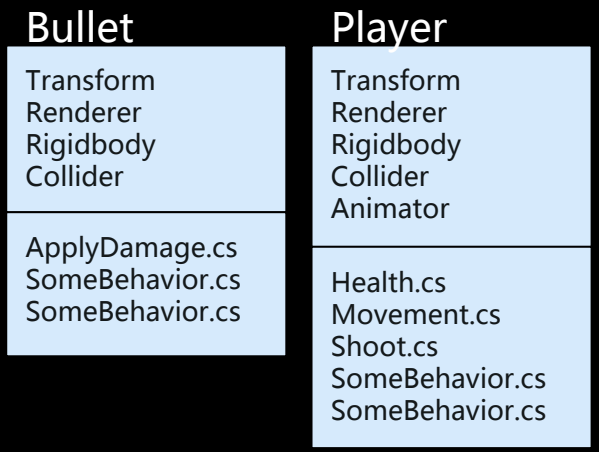
Entity

分离数据与逻辑

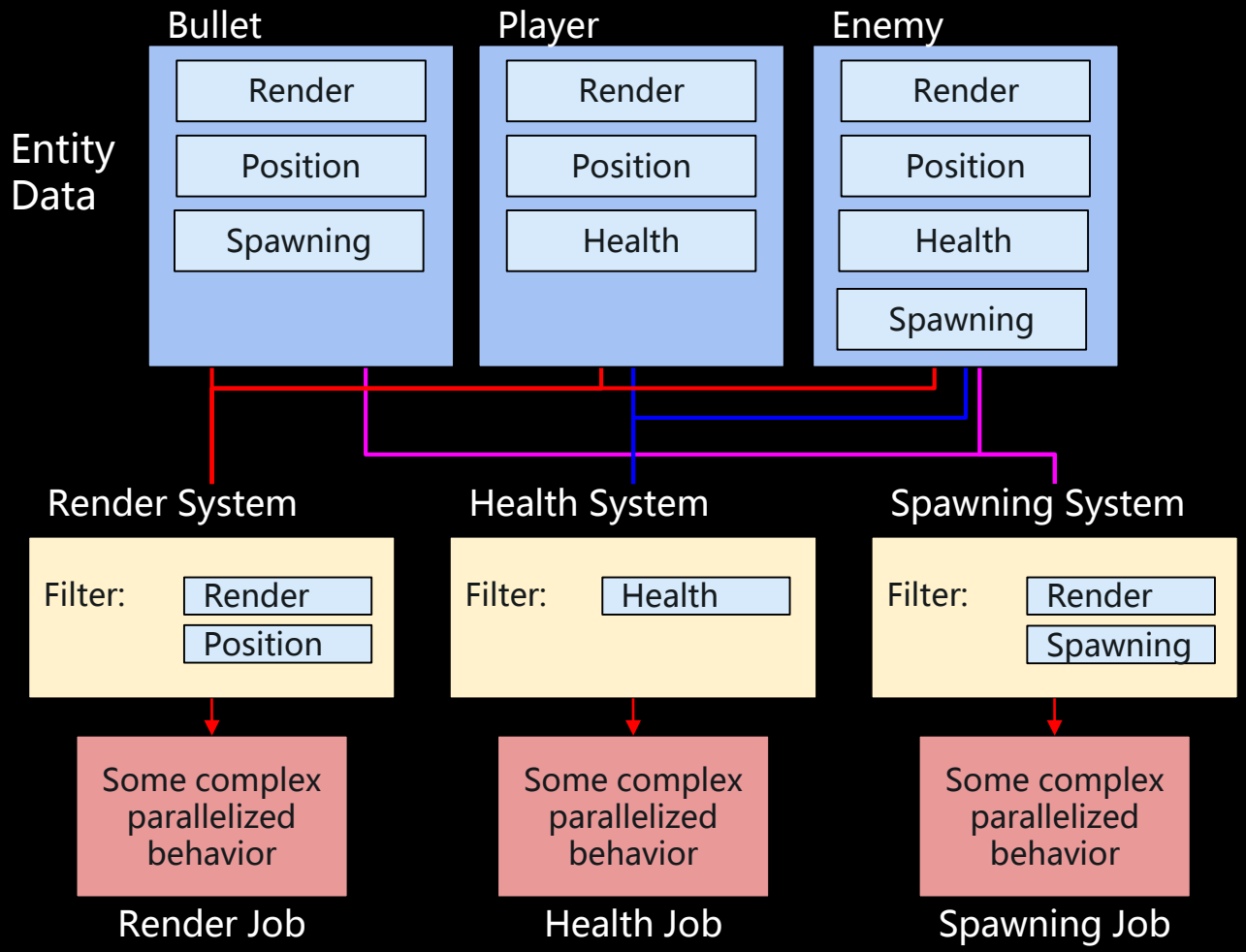


Render System
Physics System
Move System

GameObject & MonoBehaviour



Entity Component System



- ECS的简介
- ECS的优势
- ECS的应用
- ECS的技巧

Entity Component System 优势

- 更容易写出可重用的代码
- 更容易写出高性能的代码
- ECS数据（称为Archetype）会被紧密的排列在内存中
- 可以充分利用现代硬件架构
- Burst compiler

- https://github.com/Unity-Technologies/EntityComponentSystemSamples

Entity Component System 优势

- 更容易写出可重用的代码
- 更容易写出高性能的代码
- ECS数据（称为Archetype）会被紧密的排列在内存中
- 可以充分利用现代硬件架构
- Burst compiler

- https://github.com/Unity-Technologies/EntityComponentSystemSamples

Entity Component System 优势

- 更容易写出可重用的代码
- 更容易写出高性能的代码
- ECS数据（称为Archetype）会被紧密的排列在内存中
- 可以充分利用现代硬件架构
- Burst compiler

- https://github.com/Unity-Technologies/EntityComponentSystemSamples

现有方式的问题

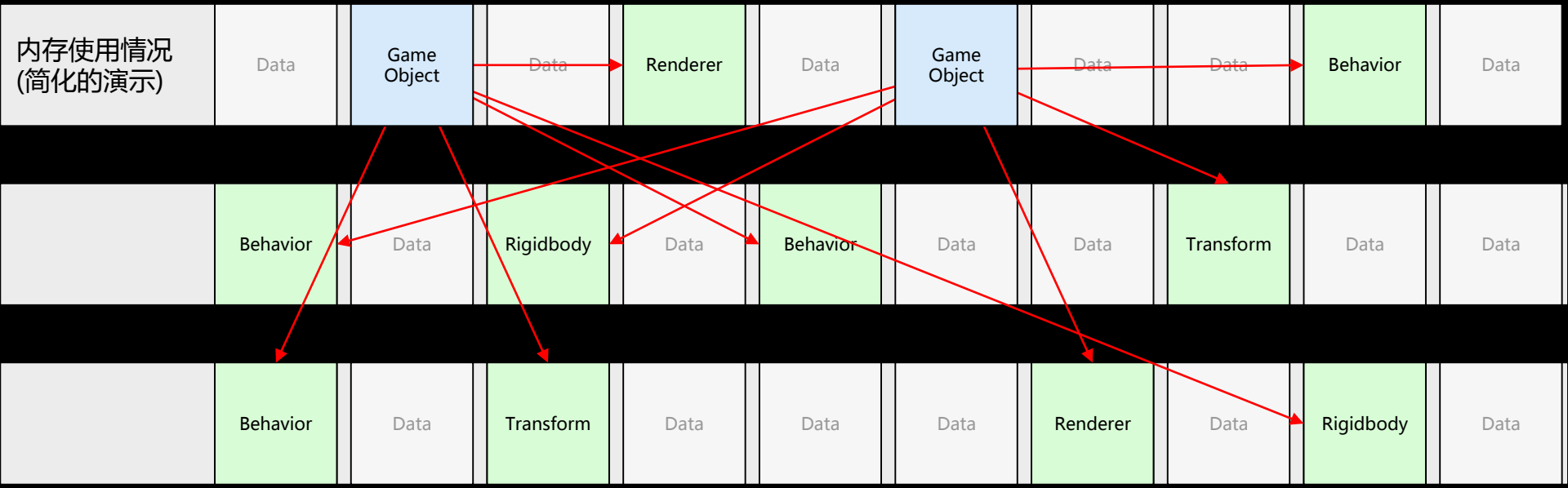
CPU Cache Miss

- 避免Cache Miss的理想情况就是**尽量使用小结构体的数组,且尽量连续的进行遍历处理**. 例如在游戏开发中, 把一类属性放在连续的内存中(数组),然后批量的进行处理.例如批量计算对象的位移.并且这样如果逻辑足够简单,数据量非常庞大,还能进一步的放入GPU进行运算.

现有方式的问题

数据分散在内存中...

从内存中加载数据速度很慢...



现有方式的问题

现有的GameObject上还绑定了很多多余的数据 ...

SomeMoveScript

speed :Float
transform :Transform

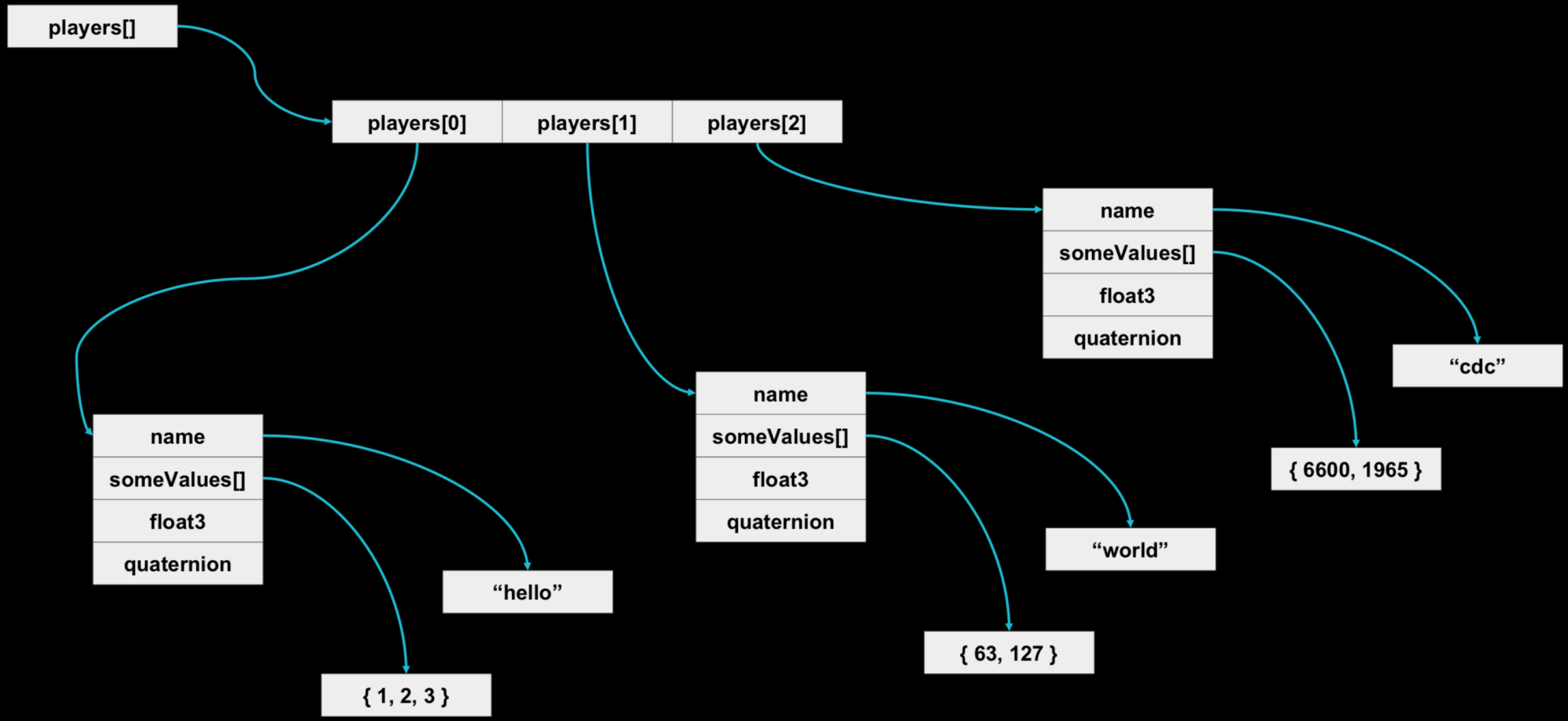
Transform

up :Vector3
right :Vector3
forward :Vector3
position :Vector3
localPosition :Vector3
rotation :Quaternion
localRotation :Quaternion
eulerAngles :Vector3
root :Transform
gameObject :GameObject
hasChanged :Bool

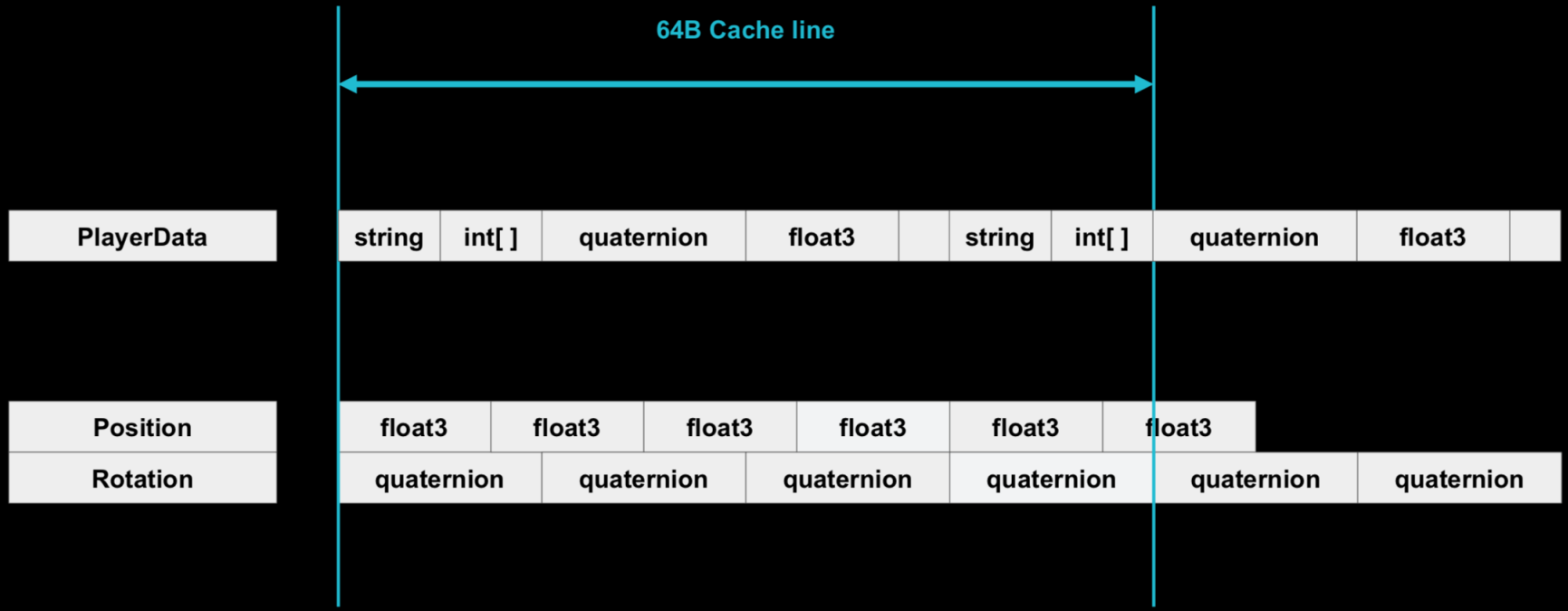
etc...

从内存到CPU缓存移动数据很慢。
但其实移动物体所需的数据其实很少

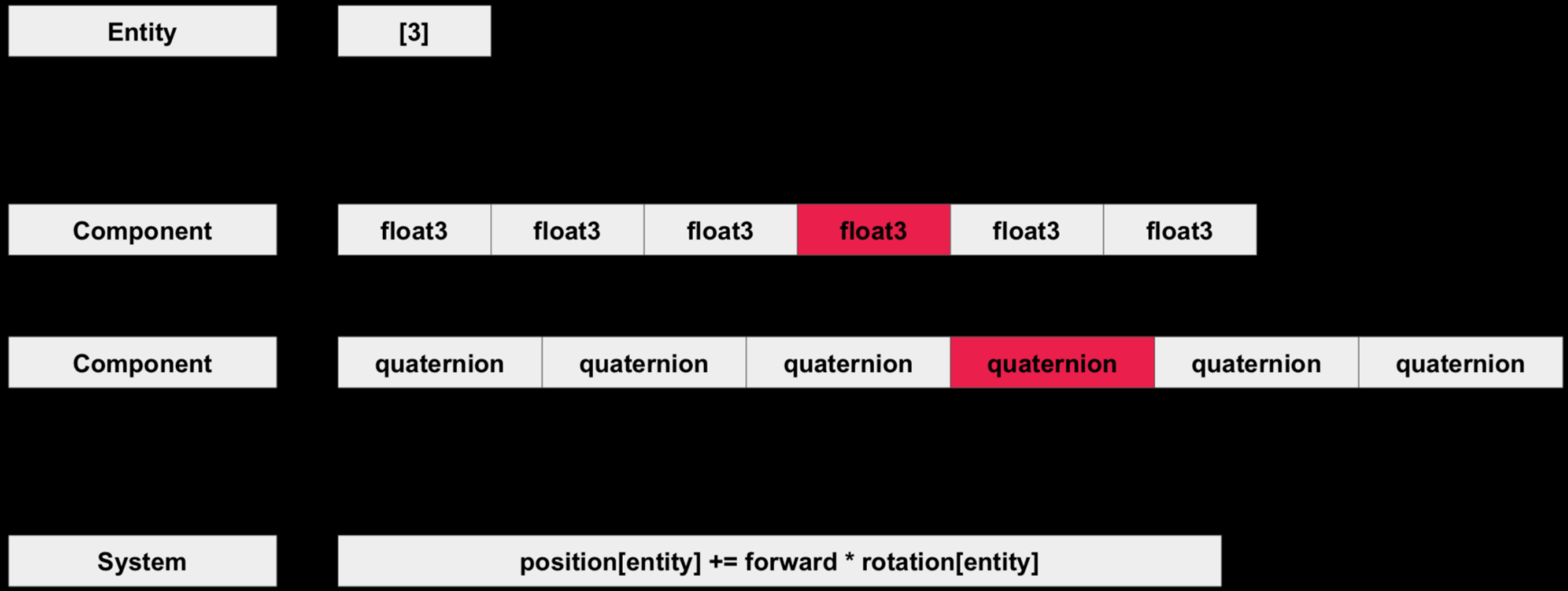
现有结构


players[]
players[0]
players[1]
players[2]
name
someValues[]
float3
quaternion
"hello"
{ 1, 2, 3 }
name
someValues[]
float3
quaternion
"world"
{ 63, 127 }
name
someValues[]
float3
quaternion
"cdc"
{ 6600, 1965 }

ECS结构


64B Cache line
PlayerData
string
int[ ]
quaternion
float3
string
int[ ]
quaternion
float3
Position
float3
float3
float3
float3
float3
float3
Rotation
quaternion
quaternion
quaternion
quaternion
quaternion
quaternion

ECS结构


Entity
[3]
Component
float3
float3
float3
float3
float3
float3
Component
quaternion
quaternion
quaternion
quaternion
quaternion
quaternion
System
position[entity] += forward * rotation[entity]

Entity Component System 优势

- 更容易写出可重用的代码
- 更容易写出高性能的代码
- ECS数据（称为Archetype）会被紧密的排列在内存中
- 可以充分利用现代硬件架构
- Burst compiler

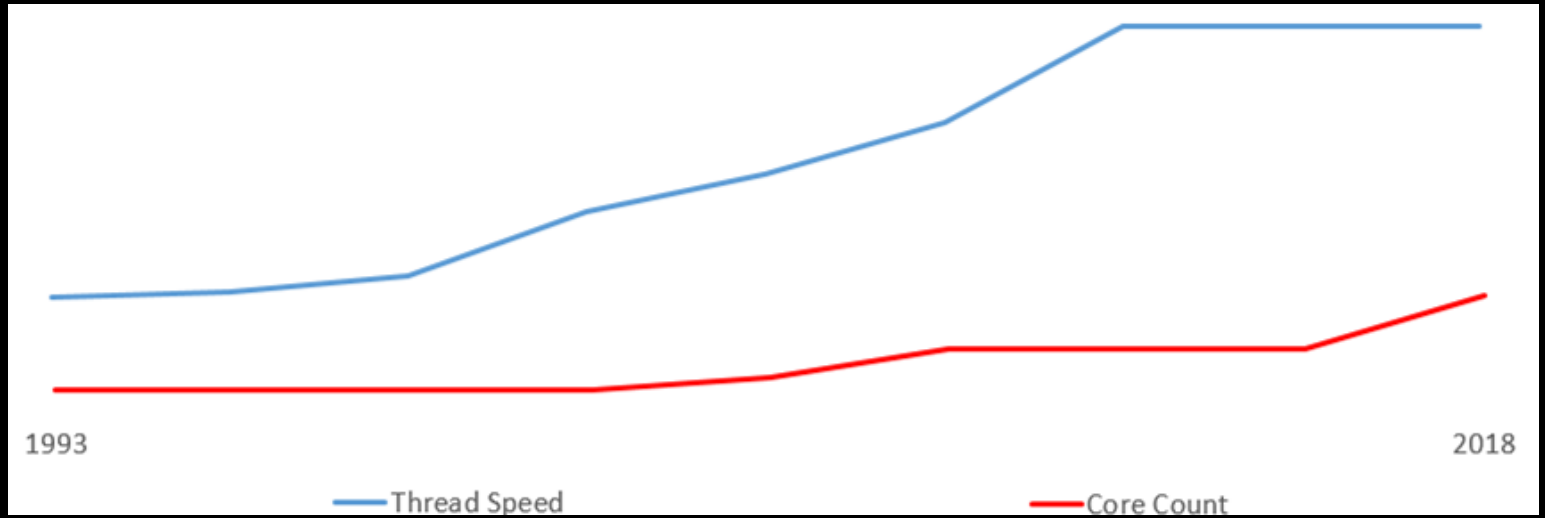
- https://github.com/Unity-Technologies/EntityComponentSystemSamples

Job System

CPU平均核数一直在增加

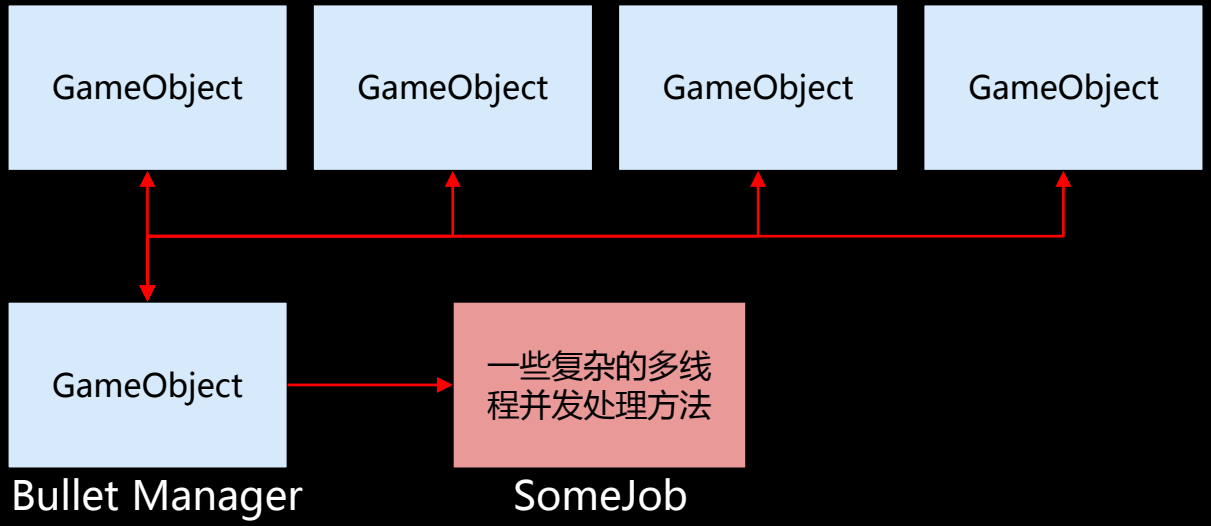
- 主流CPU的物理核心数为4-6个, 8-12个逻辑核心
- 发烧级CPU的物理核心数最多可达到16个, 32逻辑核心

很多CPU核都没有被充分利用到



Job System

- 数据和处理逻辑分离
- 充分利用CPU多核性能
- 安全利用多核性能

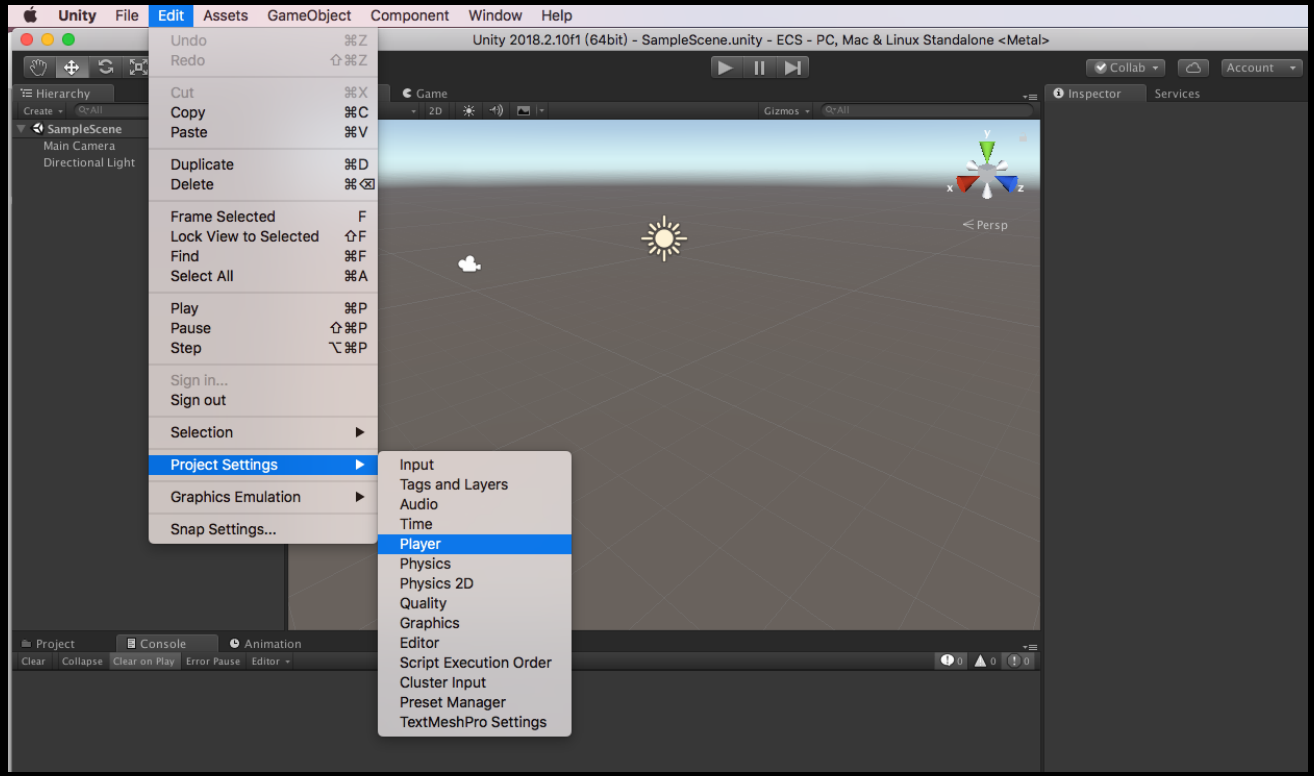


- ECS的简介
- ECS的优势
- ECS的应用
- ECS的技巧

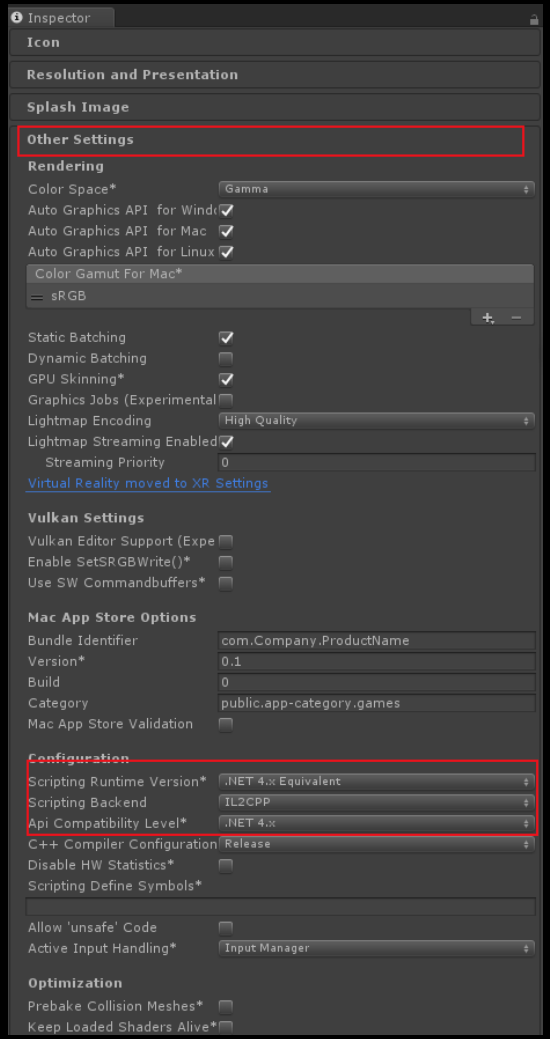
使用方法：

安装 Unity2018.1.1 之后的版本

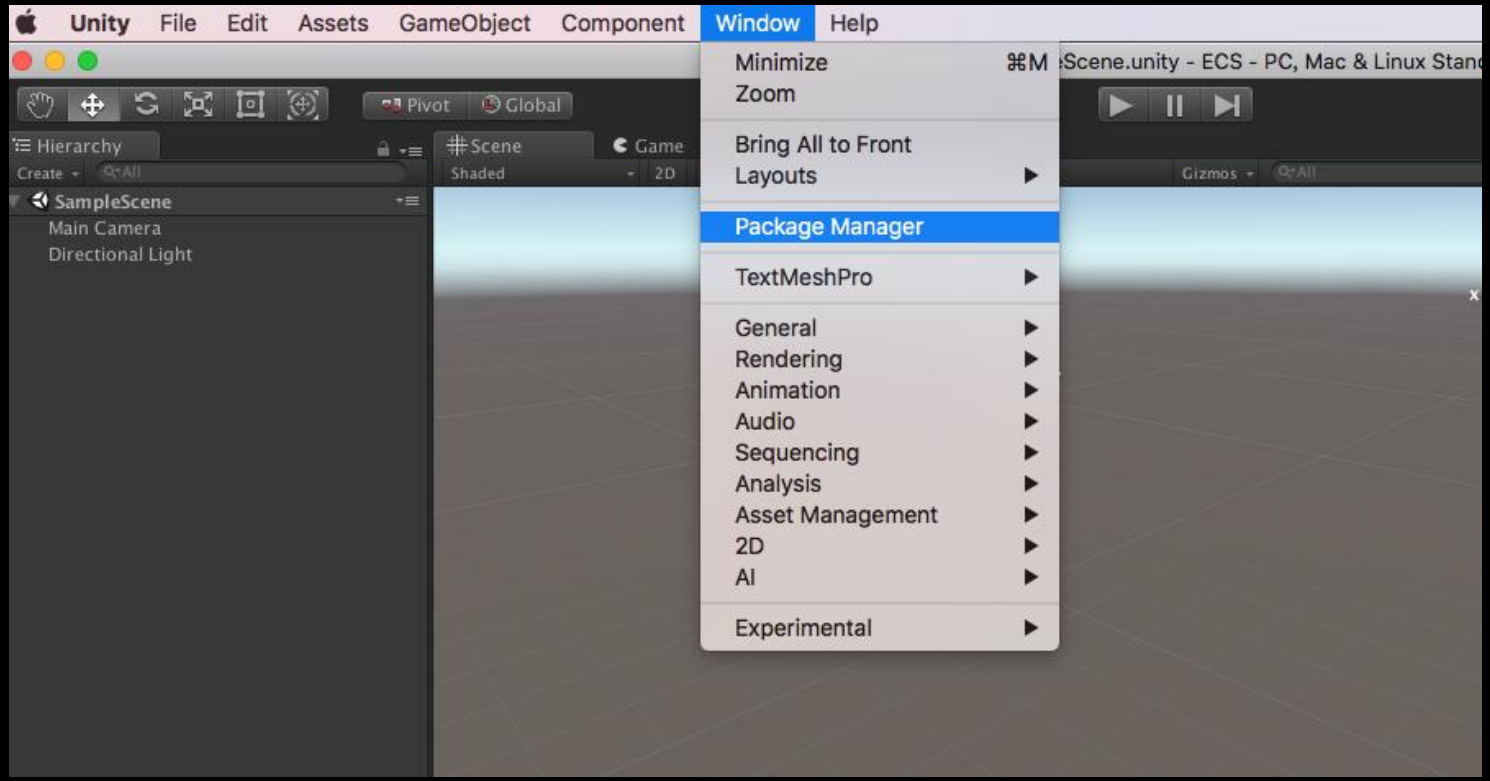
使用方法：


Unity File Edit Assets GameObject Component Window Help
Unity 2018.2.10f1 (64bit) - SampleScene.unity - ECS - PC, Mac & Linux Standalone <Metal>
Undo ⌘Z
Redo ⇧⌘Z
Cut ⌘X
Copy ⌘C
Paste ⌘V
Duplicate ⌘D
Delete ⌘⌫
Frame Selected F
Lock View to Selected ⇧F
Find ⌘F
Select All ⌘A
Play ⌘P
Pause ⇧⌘P
Step ⌥⌘P
Sign in...
Sign out
Selection
Project Settings
Graphics Emulation
Snap Settings...
Input
Tags and Layers
Audio
Time
Player
Physics
Physics 2D
Quality
Graphics
Editor
Script Execution Order
Cluster Input
Preset Manager
TextMeshPro Settings
Hierarchy
SampleScene
Main Camera
Directional Light
Game
Inspector
Services
Persp
Project
Console
Animation

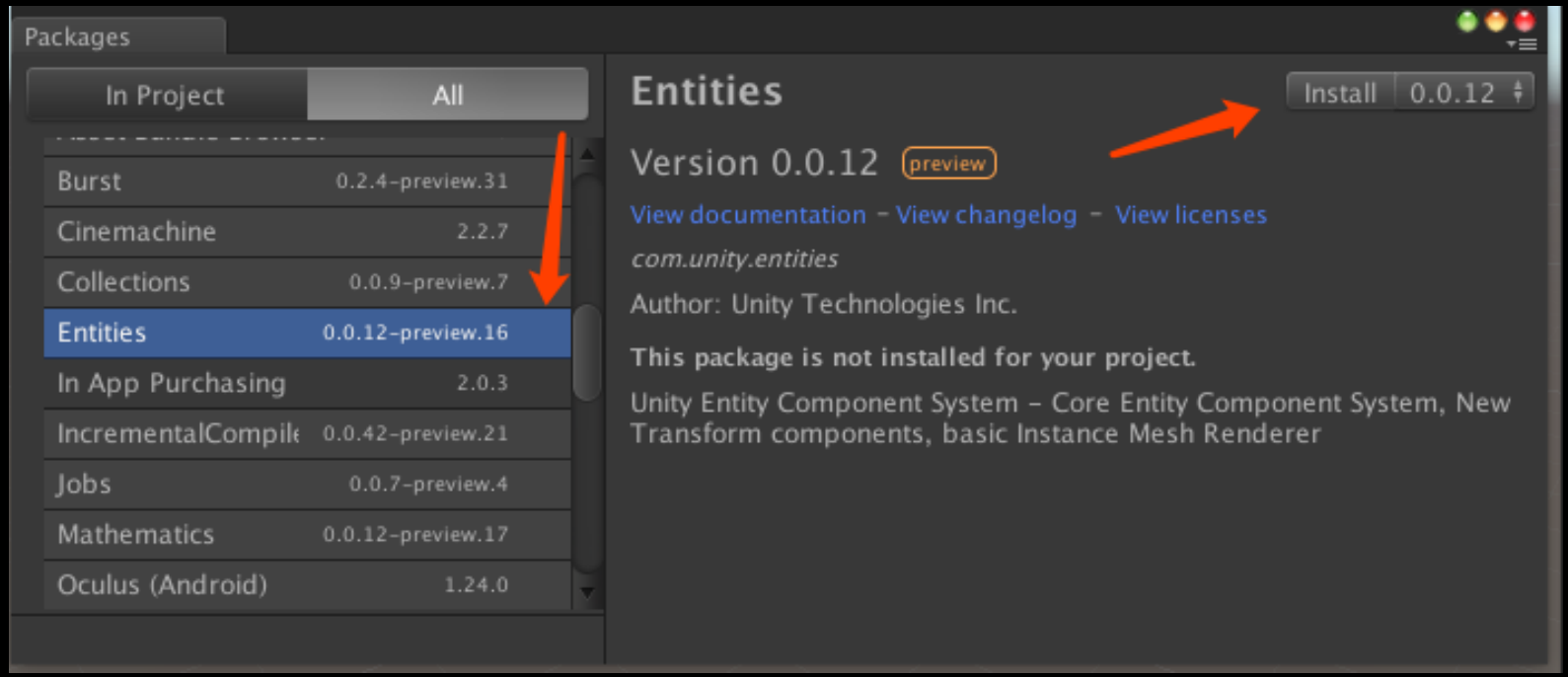
使用方法：


Inspector
Icon
Resolution and Presentation
Splash Image
Other Settings
Rendering
Color Space*
Gamma
Auto Graphics API for Wind
Auto Graphics API for Mac
Auto Graphics API for Linux
Color Gamut For Mac*
sRGB
Static Batching
Dynamic Batching
GPU Skinning*
Graphics Jobs (Experimental
Lightmap Encoding
High Quality
Lightmap Streaming Enabled
Streaming Priority
0
Virtual Reality moved to XR Settings
Vulkan Settings
Vulkan Editor Support (Expe
Enable SetSRGBWrite()*
Use SW Commandbuffers*
Mac App Store Options
Bundle Identifier
com.Company.ProductName
Version*
0.1
Build
0
Category
public.app-category.games
Mac App Store Validation
Configuration
Scripting Runtime Version*
.NET 4.x Equivalent
Scripting Backend
IL2CPP
Api Compatibility Level*
.NET 4.x
C++ Compiler Configuration
Release
Disable HW Statistics*
Scripting Define Symbols*
Allow 'unsafe' Code
Active Input Handling*
Input Manager
Optimization
Prebake Collision Meshes*
Keep Loaded Shaders Alive*

使用方法：


Unity
File
Edit
Assets
GameObject
Component
Window
Help
Minimize ⌘M
Zoom
Bring All to Front
Layouts
Package Manager
TextMeshPro
General
Rendering
Animation
Audio
Sequencing
Analysis
Asset Management
2D
AI
Experimental
Pivot
Global
Hierarchy
Create
SampleScene
Main Camera
Directional Light
Scene
Game
Shaded
2D
Gizmos

使用方法：


Packages
In Project
All
Burst 0.2.4-preview.31
Cinemachine 2.2.7
Collections 0.0.9-preview.7
Entities 0.0.12-preview.16
In App Purchasing 2.0.3
IncrementalCompile 0.0.42-preview.21
Jobs 0.0.7-preview.4
Mathematics 0.0.12-preview.17
Oculus (Android) 1.24.0
Entities
Install 0.0.12
Version 0.0.12 preview
View documentation - View changelog - View licenses
com.unity.entities
Author: Unity Technologies Inc.
This package is not installed for your project.
Unity Entity Component System - Core Entity Component System, New Transform components, basic Instance Mesh Renderer

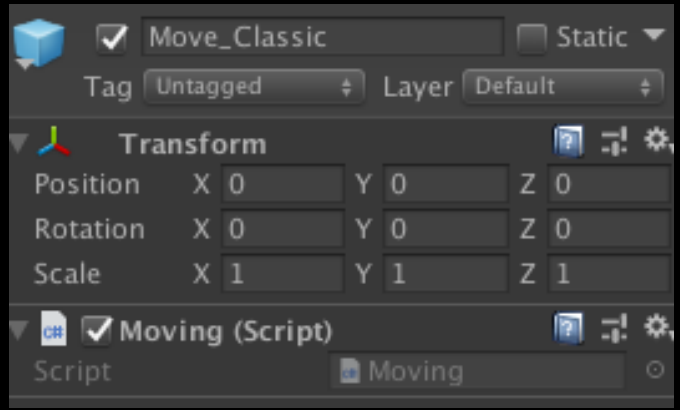
实例演示 & 性能测试

传统方式：

```
public class Moving : MonoBehaviour {

    private float speed;
    private float minX;

    void Start () {
        speed = Random.Range(2f, 5f);
        minX = transform.position.x;
    }

    void Update()
    {
        Vector3 cur = transform.position;
        if(cur.x+speed *Time.deltaTime > -minX)
        {
            transform.position = new Vector3(minX, 0,cur.z);
        }
        else
        {
            transform.position = new Vector3(cur.x + speed * Time.fixedDeltaTime, 0, cur.z);
        }
        transform.rotation = Quaternion.Euler(-90, 90, 0);
    }
}
```

传统方式：

```
public void AddGameObjectClassic(int addCount)
{
    for (int i = 0; i < addCount; ++i)
    {
        var go = Instantiate(moveClassic);
        go.transform.position = randomInitPosLeft();
        go.transform.rotation = Quaternion.Euler(0, 0, 0);
        if (willRender)
        {
            var mr = go.AddComponent<MeshRenderer>();
            mr.material = renderMat;
            var mesh = go.AddComponent<MeshFilter>();
            mesh.mesh = renderMesh;
        }
    }
    fps.AddElementCount(addCount);
}
```



机器配置：


显示适配器
NVIDIA GeForce GTX 1060 6GB


查看有关计算机的基本信息
Windows 版本
Windows 10 专业版
© 2017 Microsoft Corporation。保留所有权利。
Windows 10
系统
处理器:
Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz 3.70 GHz
已安装的内存(RAM):
16.0 GB
系统类型:
64 位操作系统，基于 x64 的处理器

运行效果：

ECS：

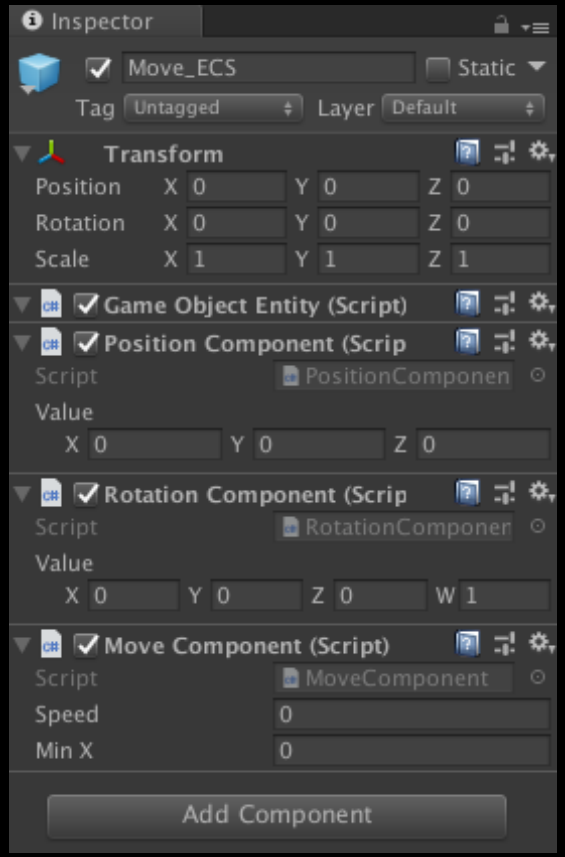
```
using UnityEngine;
using Unity.Entities;

[System.Serializable]
public struct Move :  IComponentData
{
    public float Speed;
    public float MinX;

    public Move(float x)
    {
        Speed = Random.Range(2f, 5f);
        MinX = x;
    }
}

public class MoveComponent : ComponentDataWrapper<Move> {}
```

ECS :

```
using UnityEngine;
using Unity.Entities;
using Unity.Transforms;
using Unity.Collections;
using Unity.Mathematics;

public class MoveSystem : ComponentSystem {

    struct Group
    {
        public ComponentDataArray<Position> position;
        public ComponentDataArray<Rotation> rotation;
        [ReadOnly] public ComponentDataArray<Move> move;
        public readonly int Length;
    }

    [Inject]
    Group entities;

    protected override void OnUpdate()
    {
        for (int i = 0; i < entities.Length; ++i)
        {
            Position pos = entities.position[i];
            Move move = entities.move[i];

            if (pos.Value.x + move.Speed * Time.deltaTime > -move.MinX)
            {
                pos.Value.x = move.MinX;
            }
            else
            {
                pos.Value.x = pos.Value.x + move.Speed * Time.fixedDeltaTime;
            }

            entities.position[i] = pos;
            Rotation r = new Rotation();
            r.Value = quaternion.EulerXYZ(new float3(-90,90,0));
            entities.rotation[i] = r;
        }
    }
}
```

ECS :

```
public void AddECS(int addCount)
{
    NativeArray<Entity> entities = new NativeArray<Entity>(addCount, Allocator.Temp);

    EntityManager manager = World.Active.GetOrCreateManager<EntityManager>();
    manager.Instantiate(moveECS, entities);
    for (int i = 0; i < addCount; ++i)
    {
        var pos = randomInitPosLeftF3();
        manager.SetComponentData(entities[i], new Position { Value = pos });
        manager.SetComponentData(entities[i], new Move(pos.x));
        if (willRender)
        {
            manager.AddSharedComponentData(entities[i], new MeshInstanceRenderer() {
                mesh = renderMesh,
                material = renderMat
                });
        }
    }

    entities.Dispose();
    fps.AddElementCount(addCount);
}
```



运行效果：

x 1.12

ECS With Job System :

```
using UnityEngine;
using Unity.Entities;

[System.Serializable]
public struct MoveWithJob : IComponentData
{
    public float Speed;
    public float MinX;

    public MoveWithJob(float x)
    {
        Speed = Random.Range(2f, 5f);
        MinX = x;
    }
}

public class MoveWithJobComponent : ComponentDataWrapper<MoveWithJob> { }
```

ECS With Job System :

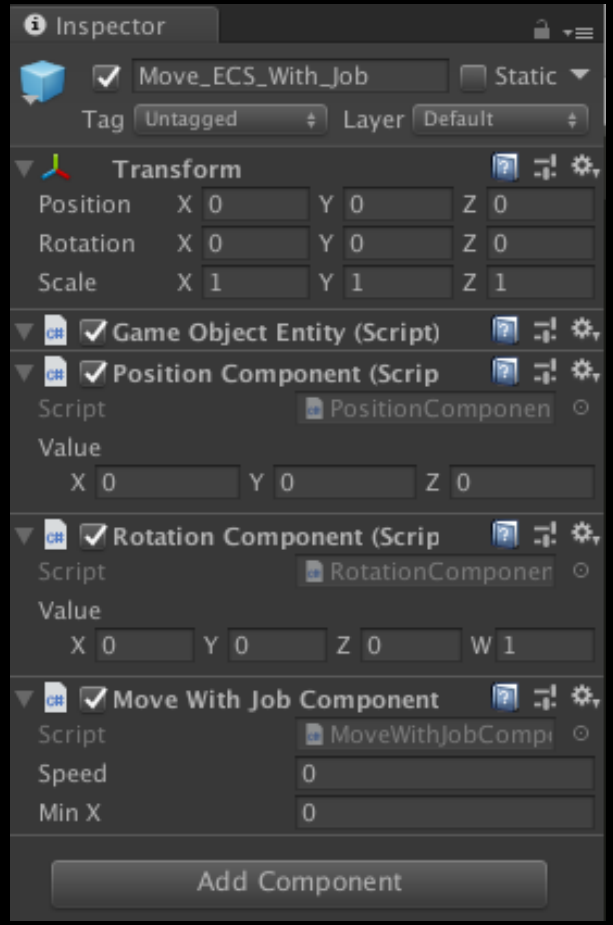
```
using UnityEngine;
using Unity.Entities;

[System.Serializable]
public struct MoveWithJob : IComponentData
{
    public float Speed;
    public float MinX;

    public MoveWithJob(float x)
    {
        Speed = Random.Range(2f, 5f);
        MinX = x;
    }
}

public class MoveWithJobComponent : ComponentDataWrapper<MoveWithJob> { }
```

ECS With Job System：

```
using UnityEngine;
using Unity.Burst;
using Unity.Jobs;
using Unity.Entities;
using Unity.Transforms;
using Unity.Collections;
using Unity.Mathematics;

public class MoveJobSystem : JobComponentSystem {

    [BurstCompile]
    struct MoveJob : IJobProcessComponentData<Position, Rotation, MoveWithJob>
    {
        public float deltaTimeFromMainThread;

        public void Execute(ref Position pos, ref Rotation rot, [ReadOnly]ref MoveWithJob move)
        {

            float3 posf = pos.Value;
            if (posf.x + move.Speed * deltaTimeFromMainThread > -move.MinX)
            {
                posf.x = move.MinX;
            }
            else
            {
                posf.x = posf.x + move.Speed * deltaTimeFromMainThread;
            }
            pos.Value = posf;

            rot.Value = quaternion.EulerXYZ(new float3(-90, 90, 0));
        }
    }

    protected override JobHandle OnUpdate(JobHandle inputDeps)
    {
        MoveJob mj = new MoveJob()
        {
            deltaTimeFromMainThread = Time.fixedDeltaTime
        };
        JobHandle moveHandle = mj.Schedule(this, inputDeps);
        return moveHandle;
    }
}
```

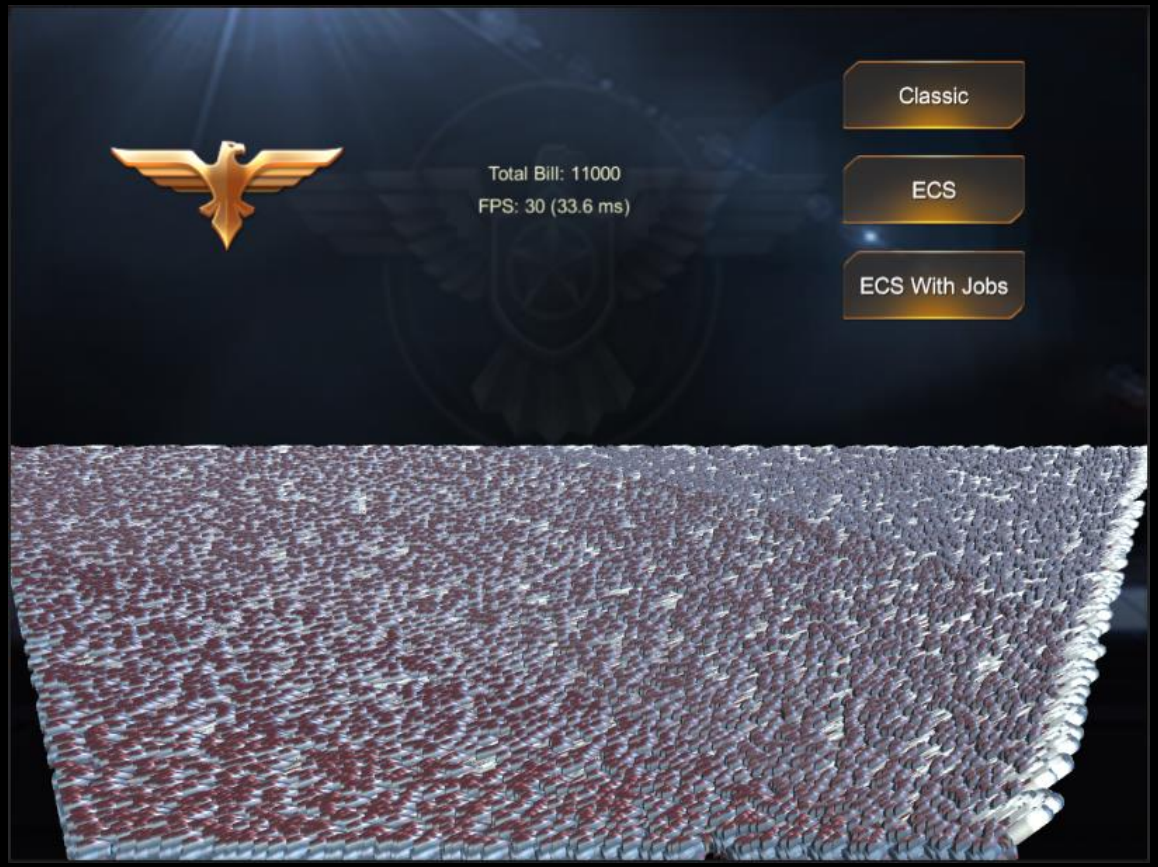
ECS With Job System :

```
public void AddECSWithJobs(int addCount)
{
    if(willRender && addCount > 1000)
    {
        addCount = 1000;
    }
    NativeArray<Entity> entities = new NativeArray<Entity>(addCount, Allocator.Temp);
    EntityManager manager = World.Active.GetOrCreateManager<EntityManager>();
    manager.Instantiate(moveECSWithJobs, entities);
    for (int i = 0; i < addCount; ++i)
    {
        var pos = randomInitPosLeftF3();
        manager.SetComponentData(entities[i], new Position { Value = pos });
        manager.SetComponentData(entities[i], new MoveWithJob(pos.x));
        if (willRender)
        {
            manager.AddSharedComponentData(entities[i], new MeshInstanceRenderer() {
                mesh = renderMesh,
                material = renderMat
                });
        }
    }

    entities.Dispose();
    fps.AddElementCount(addCount);
}
```



运行效果：

x 1.38



纯计算性能统计：

Classic

纯计算性能统计：

ECS

x 1.2

纯计算性能统计：

ECS & Job

x 93.5

开源代码：

https://github.com/fansongy/SampleECS

- ECS的简介
- ECS的优势
- ECS的使用
- ECS的技巧

单例组件（Singleton Component）

这些组件属于单一的匿名实体，可以通过World直接访问。System中的大部分状态都可以移到单例组件。

另一方面只需要被一个System访问的状态其实是很罕见的。如果发现这个系统需要依赖一些状态。就做一个单例来存储，几乎每一次都会发现其他一些System也同样需要这些状态。

单帧延迟组件（Deferment）

它包含了一个未决的记录数组，每个记录都有足够的信息，来在本帧的晚些时候执行逻辑。等运行到当前帧的后期，例如，进行场景更新和准备渲染的时候，再由对应System遍历数组，并执行逻辑。

这样的话，即使有严重的副作用，在每一帧也只是发生在一个调用点而已。从优化的角度说，这种设计可以使非关键性能热点有更大的优化空间。

公用行为（Utility Function）

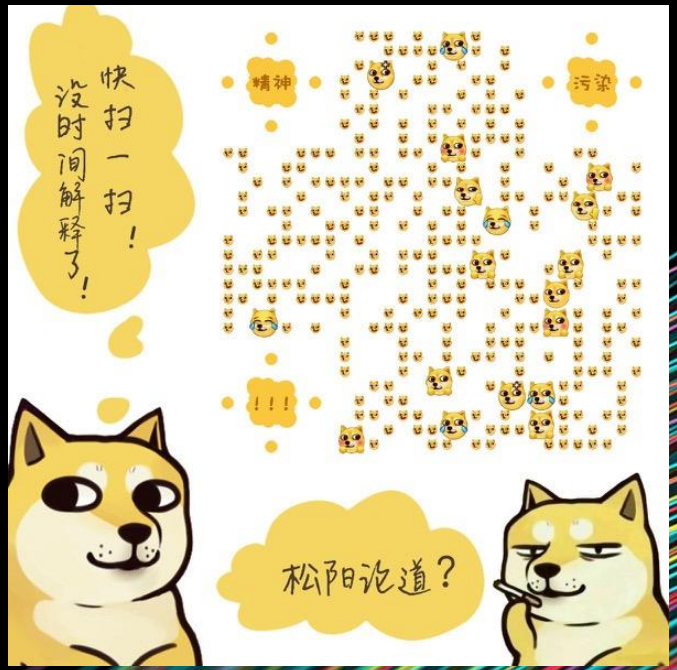
公用行为通常是被多个System用到的同一个功能。因此可被抽离成一个调用函数，可以认为是Static方法的ECS版本。

从代码复杂度上看，如果是广泛使用的公共行为，那么这个函数就应该依赖很少的组件，而且不应该带副作用或者很少的副作用。 如果公共行为函数依赖很多组件，就要限制它的调用点数量。

参考和资料

- Unity2018官方路演
- https://github.com/Unity-Technologies/EntityComponentSystemSamples
- https://unity3d.com/unity/features/job-system-ECS
- Intro To The Entity Component System And C# Job System
- New way of CODING in Unity! ECS Tutorial
- 《守望先锋》架构设计与网络同步 (http://gad.qq.com/article/detail/28682)

谢谢



unity