Applications of Unity's DOTS in "Return to Empire"

Thousands of Soldiers Battle On One Mobile Screen

Jian Xiao

Principal Client Programmer

TiMi Studio Group of Tencent Games

jamesjxiao@tencent.com

Cangjian Hou

Principal Engine Programmer

TiMi Studio Group of Tencent Games

cangjianhou@tencent.com

Tencent Games

Tencent Games is established in 2003. We are a leading global platform for game development, operations and publishing, and the largest online game community in China.

Tencent Games has developed and operated over 140 games. We provide cross-platform interactive entertainment experience for more than 800 million users in over 200 countries and regions around the world. Honor of Kings, PUBG MOBILE, and League of Legends, are some of our most popular titles around the world.

While providing the best game experience, we strive to serve the community. We have taken the lead in creating a balanced and healthy game environment for underaged players, as ensuring the well-beings remains our priority.

Meanwhile, we actively promote the development of esports industry, work with global partners to build an open, collaborative and symbiotic industrial ecology, and create high-quality digital life experiences for players.

This is Tencent Games - an inspirer of happiness and explorer of game value. While bringing joy to players around the world, we are exploring more possibilities of video games.

Spark More.

Tencent Games TiMi TIMI STUDIOS GDC

01 Background and Introduction

Return to Empire

02 Use of DOTS

Practical experience and methodology

03 Performance Optimization

Mobile games performance sensitive

01 Background and Introduction

Return to Empire

Game: Return To Empire

- Full 3D SLG mobile game
- 1000+ character units
- Outside large battlefields

Challenges

Tencent Games TiMi TIMI STUDIOS GDC

More than one thousand soldiers fighting on a mobile screen?
(Targeting 30 FPS on mobile)

C# OOP?
C++ Plugin?
DOTS!

Unity DOTS (Data-Oriented Technology Stack)

data-based multi-threading
large-scale computing

- *In 2019, no preceding mobile games on the market have used DOTS technology on a large scale.*
- *Tight development schedule.*
- *This is an ADVENTURE!*

✓ *We have close cooperation with Unity official*

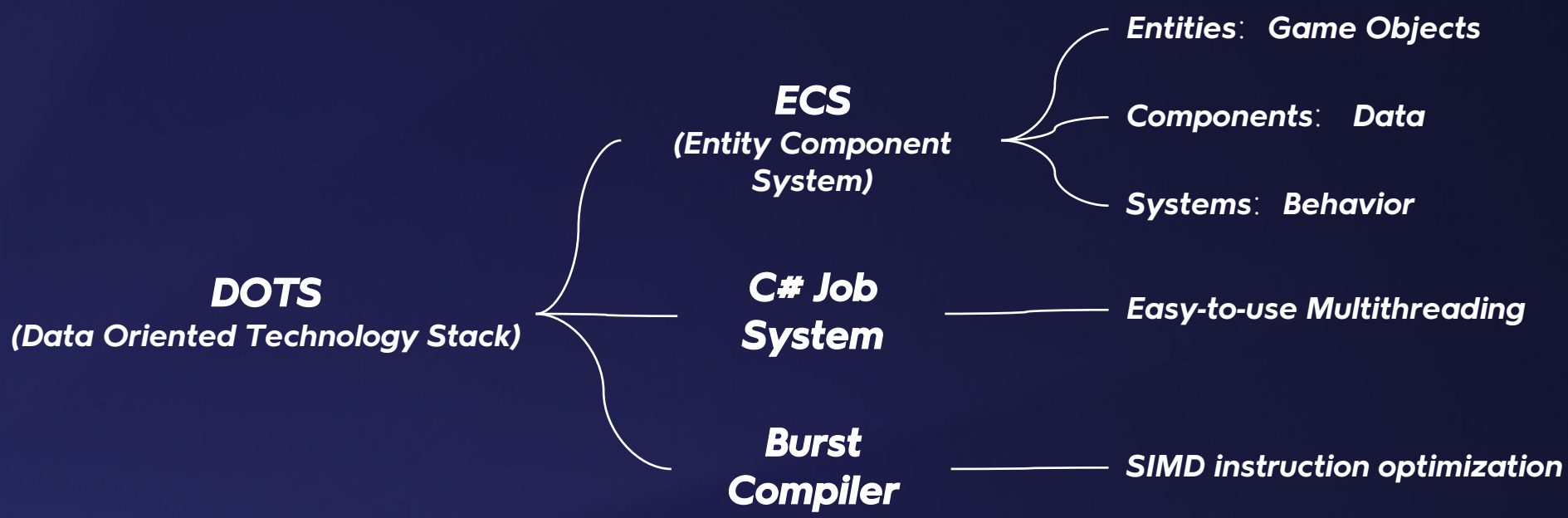
✓ *We are up for the challenge!*

02 Use of DOTS

Practical experience and methodology

Use Of DOTS – Concepts

A brief introduction to Unity DOTS



Code Design

ECS Code of "Return to Empire"

Job System

Burst Compiler

Systems

SoldierInitSystem	ArmyFollowPathSystem	P2PEffectSystem
SoldierAnimTriggerSystem	ArmyMoveSystem	ParabolaEffectSystem
SoldierDeadSystem	ArmyNormalAttackSystem	ParticleRePlaySystem
SoldierFightingStateSystem	ArmyRotateToTargetSystem	DestroyEntitySystem
SoldierFireSkillSystem	ArmySpawnSystem	LifeTimeEffectSystem
SoldierFlySystem	ArmyStateJobSystem	ParticleRePlaySystem
SoldierAnimSpeedLerpSystem	PositionLerpSystem	ParticleRePlaySystem
SteeringMoveSystem	RotationLerpSystem	...

Components

AnimLoopData	GroupAIData	ServerPropertyData	SoldierMoveData	EffectPropertyData
AnimSpeedLerpData	GroupFormationData	SoldierAIData	SoldierMoveNeedInfo	ParabolaEffectData
ArmyMoveTransitData	GroupHorseData	SoldierAnimNeedInfo	SoldierPropertyData	P2PEffectData
ArmyStateData	GroupLeaderData	SoldierAnimotarData	SoldierSkillData	RecycleEffectData
GroupUnitData	PositionLerpData	SoldierAnimTriggerData	SpawnSoldierData	LifeTimeData
LogicColliderData	ResourceCustomData	SoldierDeadData	SteeringMoveData	EffectVisualizationData
MeshLineData	RotationLerpData	SoldierFightingStateData	TagsMapEntity	SoundPlayData
PathPointData	SelectedStateData	SoldierFlyData	UniformMoveData	.....

Systems drive Data changes

Sample

A simple function using ECS

Related Components

- LineMoveData
- Translation
- Rotation
- RotationLerpData
- CityNpcPropertyData

→ **CityLineMoveSystem**

drive soldiers to walk in the city

```
public struct LineMoveData : IComponentData
{
    public bool     ReTarget;
    public float    MoveSpeed;
    public float    TargetYaw;

    public float3   TargetPos;
    public float3   StartPos;
    public float3   FinalPos;
}
```

```
[BurstCompile]
struct LineMoveJob : IJobChunk
{
    public float                        deltaTime;

    [ReadOnly] public ArchetypeChunkEntityType                  Archetype
    public          ArchetypeChunkComponentType<CityNpcPropertyData>
    public          ArchetypeChunkComponentType<LineMoveData>       Ar
    public          ArchetypeChunkComponentType<Translation>        Arch
    public ArchetypeChunkComponentType<Rotation> ArchetypeRotation
    public ArchetypeChunkComponentType<RotationLerpData> Archetyp
```

```
m_npcGroup = GetEntityQuery(new EntityQueryDesc()
{
    All = new ComponentType[]
    {
        ComponentType.ReadWrite<LineMoveData>(),
        ComponentType.ReadWrite<Translation>(),
        ComponentType.ReadWrite<Rotation>(),
        ComponentType.ReadWrite<RotationLerpData>(),
        ComponentType.ReadWrite<CityNpcPropertyData>(),
    },
});
```

Key points to make good use of DOTS

1. Job Dependency Optimization

Job dependencies, read and write order, determine the concurrency of the entire system.

2. ECS and non-ECS in parallelization

Further improve the overall concurrency of the game.

3. Split logic and visualization

Reduce stuttering and make the game running smoother

4. Reduce System Frequency

By controlling the frequency on demand for different systems, better game performance can be achieved .

1 Dependency

Write-Write Conflict: split data

NO

MoveSystem — Write dstPos

RotateSystem — Write quaternion

```
public struct ParabolaEffectData
: IComponentData
{
    public quaternion quaternion;
    public float3 srcPos;
    public float3 dstPos;
}
```

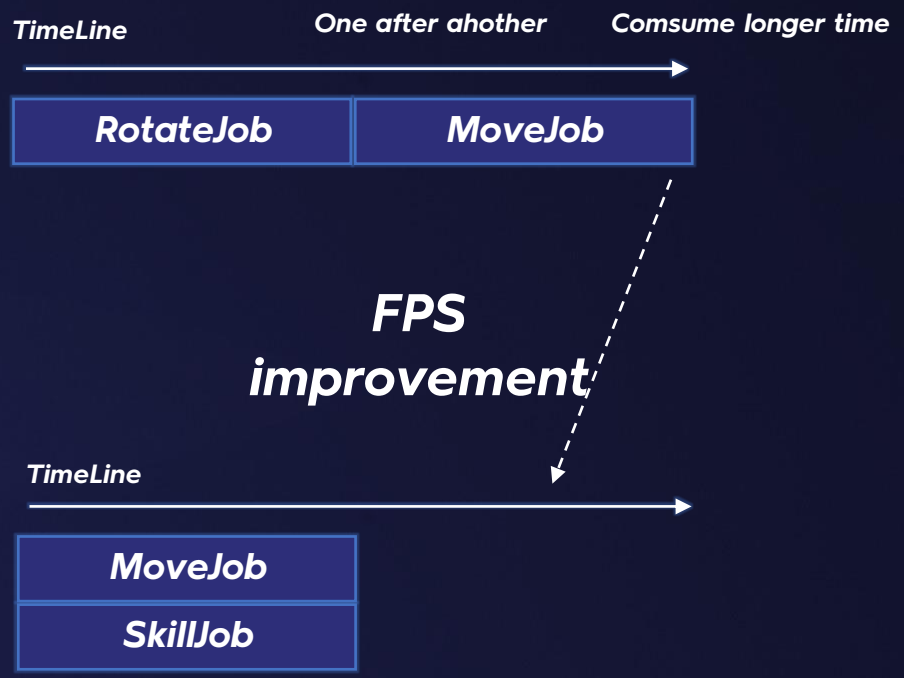
YES

RotateSystem — Write To

```
public struct ParabolaEffectRotateData
: IComponentData
{
    public quaternion quaternion;
}
```

MoveSystem — Write To

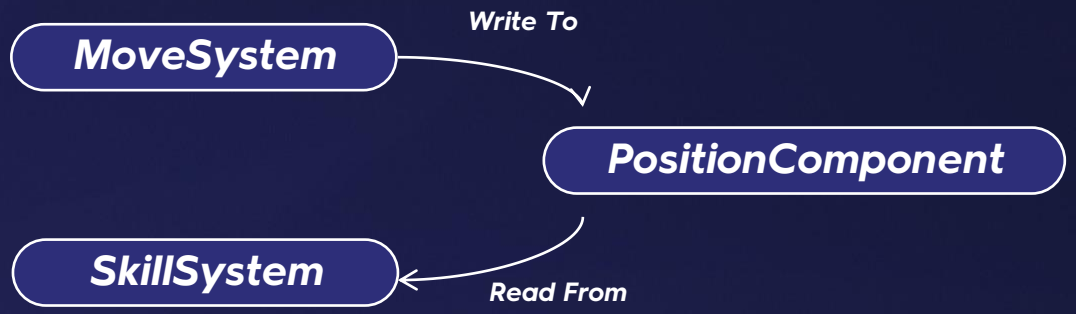
```
public struct ParabolaEffectMoveData
: IComponentData
{
    public float3 srcPos;
    public float3 dstPos;
}
```

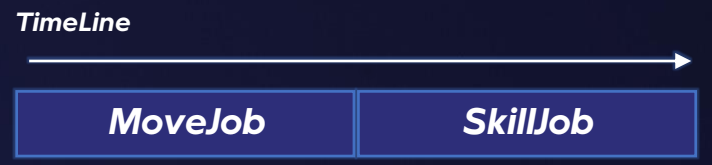


1 Dependency

Read-Write Conflict: data redundancy

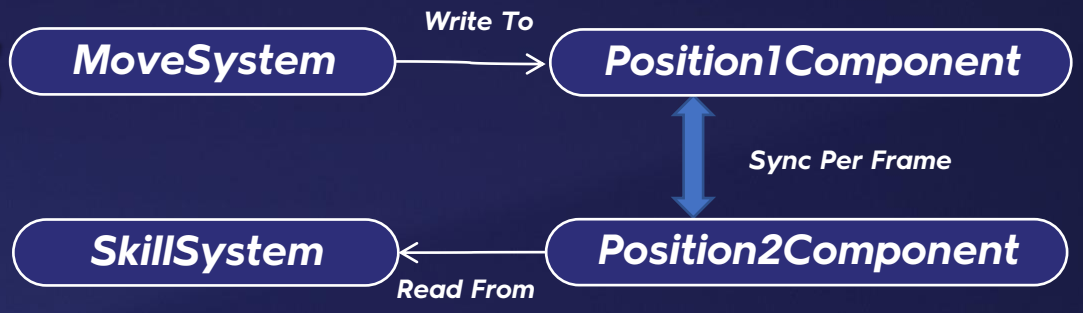
NO

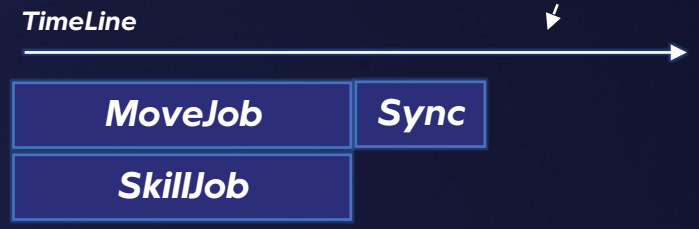




FPS improvement

YES





1 Dependency Analysis Tool

We developed a tool for visualizing dependencies between systems

By using this tool, we can easily find design bottleneck

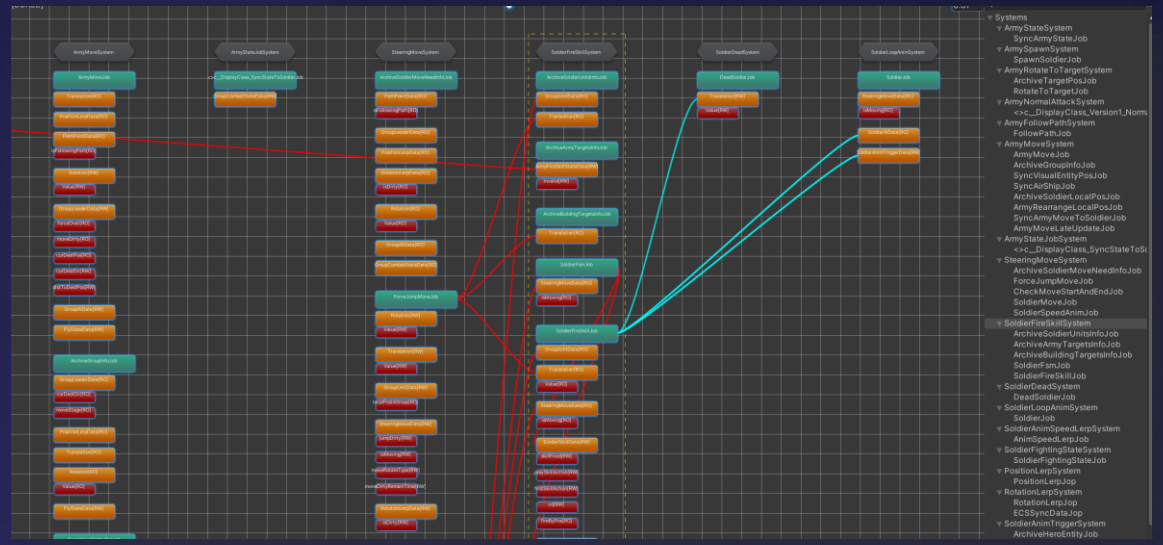
Without this tool, we can only analyze by reading code, which is very inefficient…

1 Dependency Analysis Tool

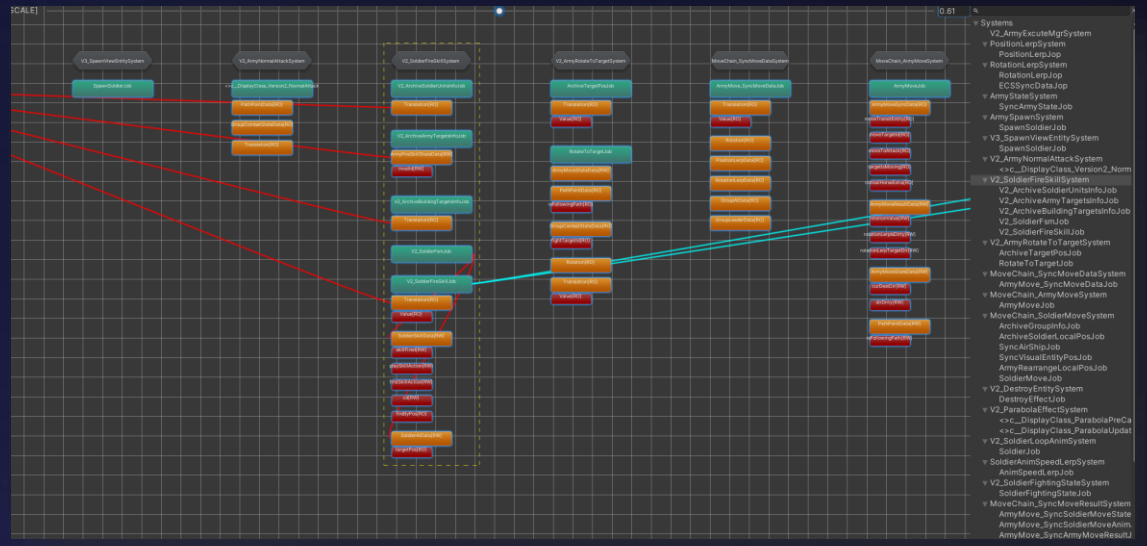
Tencent Games TiMi TIMI STUDIOS GDC

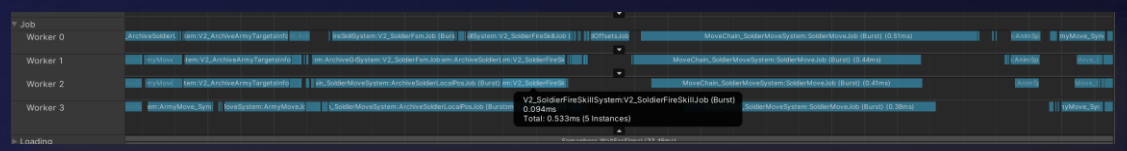
Before

SoldierFireSkillSystem



After

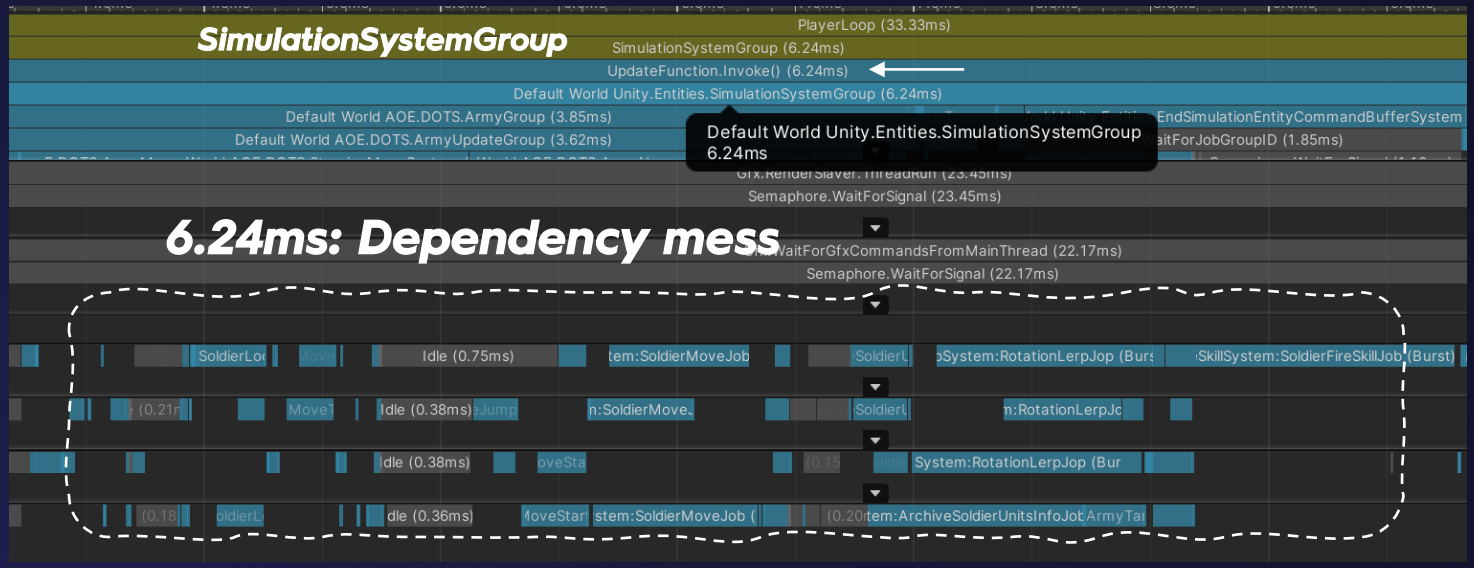


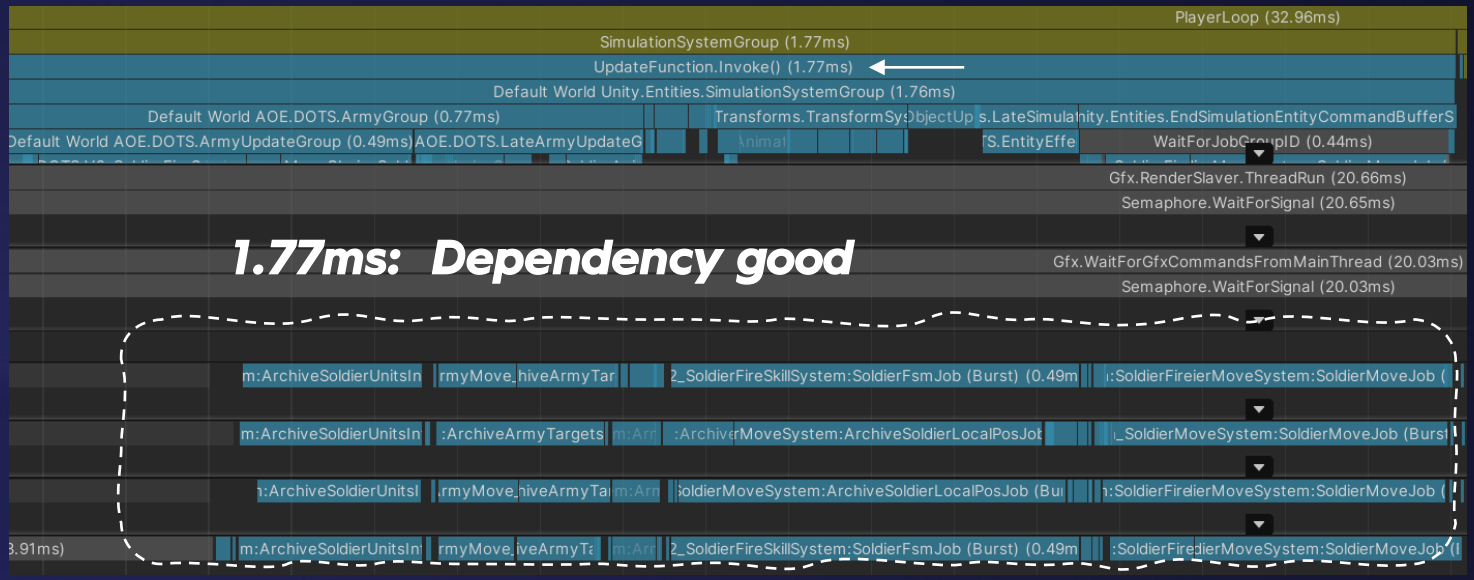


1 Dependency

significant effect!

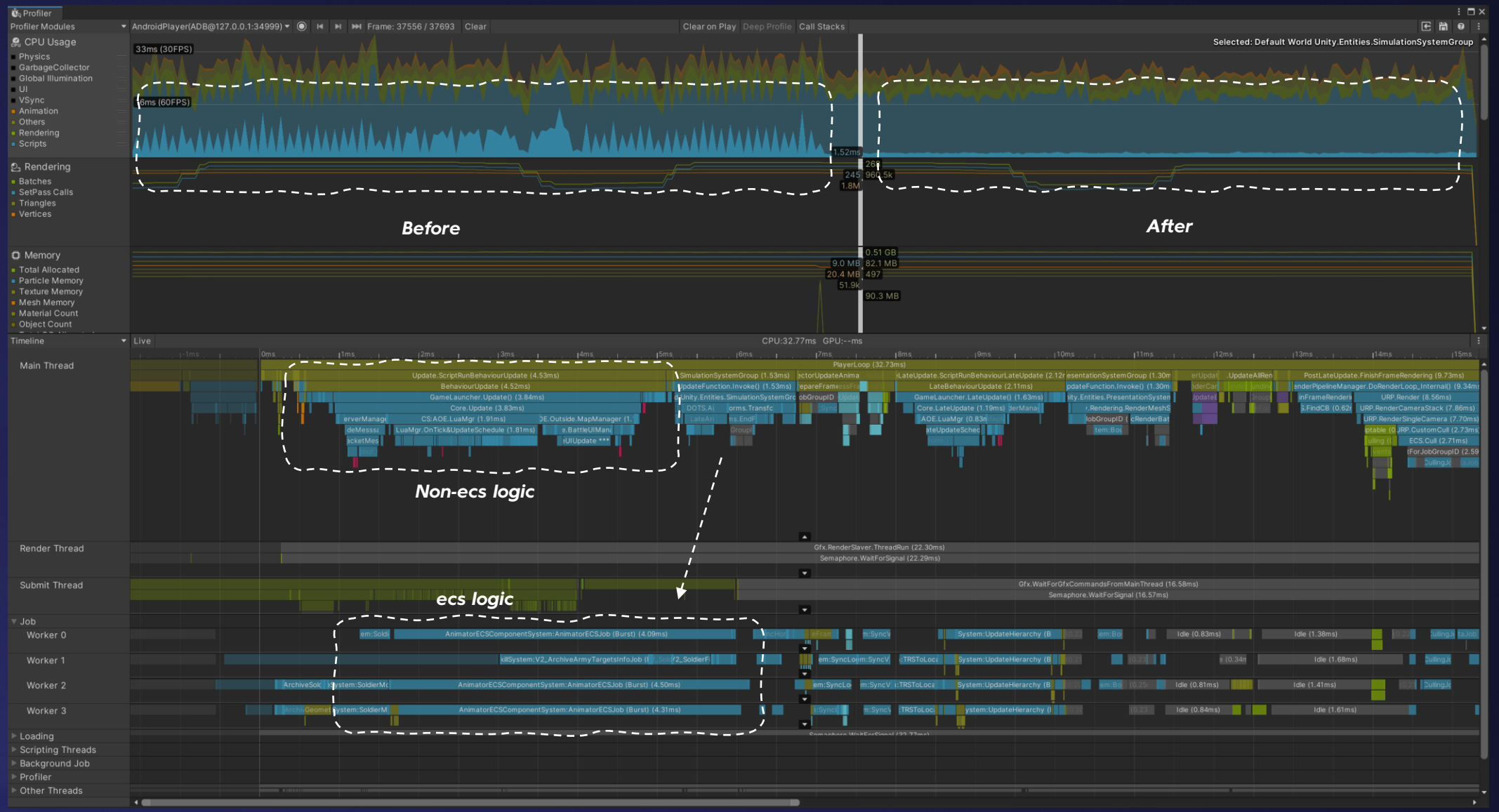
Dependencies Optimize





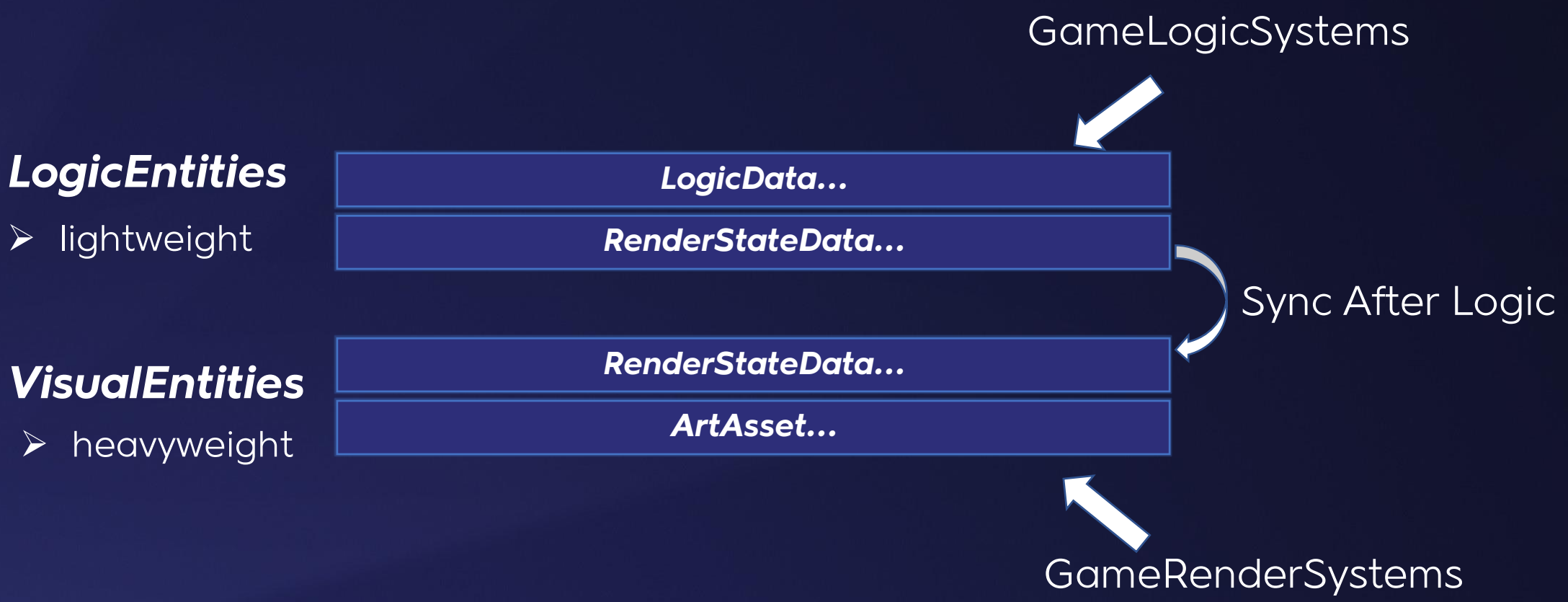
2 ECS and non-ECS in parallelization

2 ECS and non-ECS in parallelization



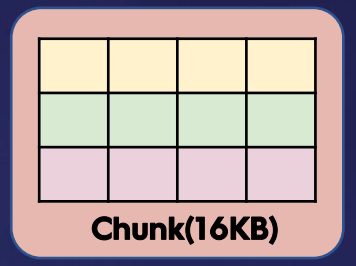
3 Split logic and visualization

One Entity = Logic Entity + Visual Entity



3 Split logic and visualization

- Data is arranged more tightly
- Async loading of ArtAsset
- Art-Asset Entities reuse



A Chunk = 14 Entities

A Chunk = 32 logic Entities

A Chunk = 24 visual Entities

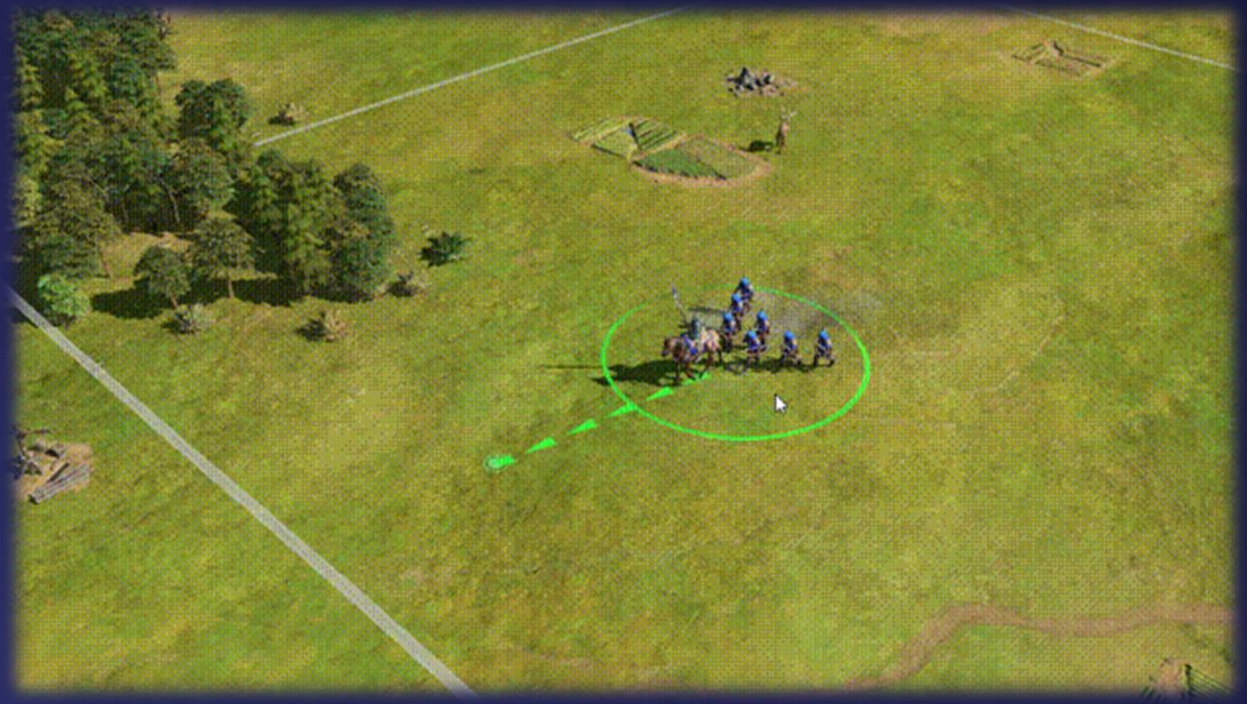
Game running smoothly
without assets-load blocking

4 Reduce System Frequency

Tencent Games TiMi TIMI STUDIOS GDC

Reduce frequency to reduce CPU overhead

Reduce MoveSystem freq to 12fps



barely noticeable changes!

AND

1000+ soldiers move

Power consumption reduced by 0.23w

03 Performance Optimization

Mobile games performance sensitive

Cangjian Hou

Principal Engine Programmer

TiMi Studio Group of Tencent Games

cangjianhou@tencent.com

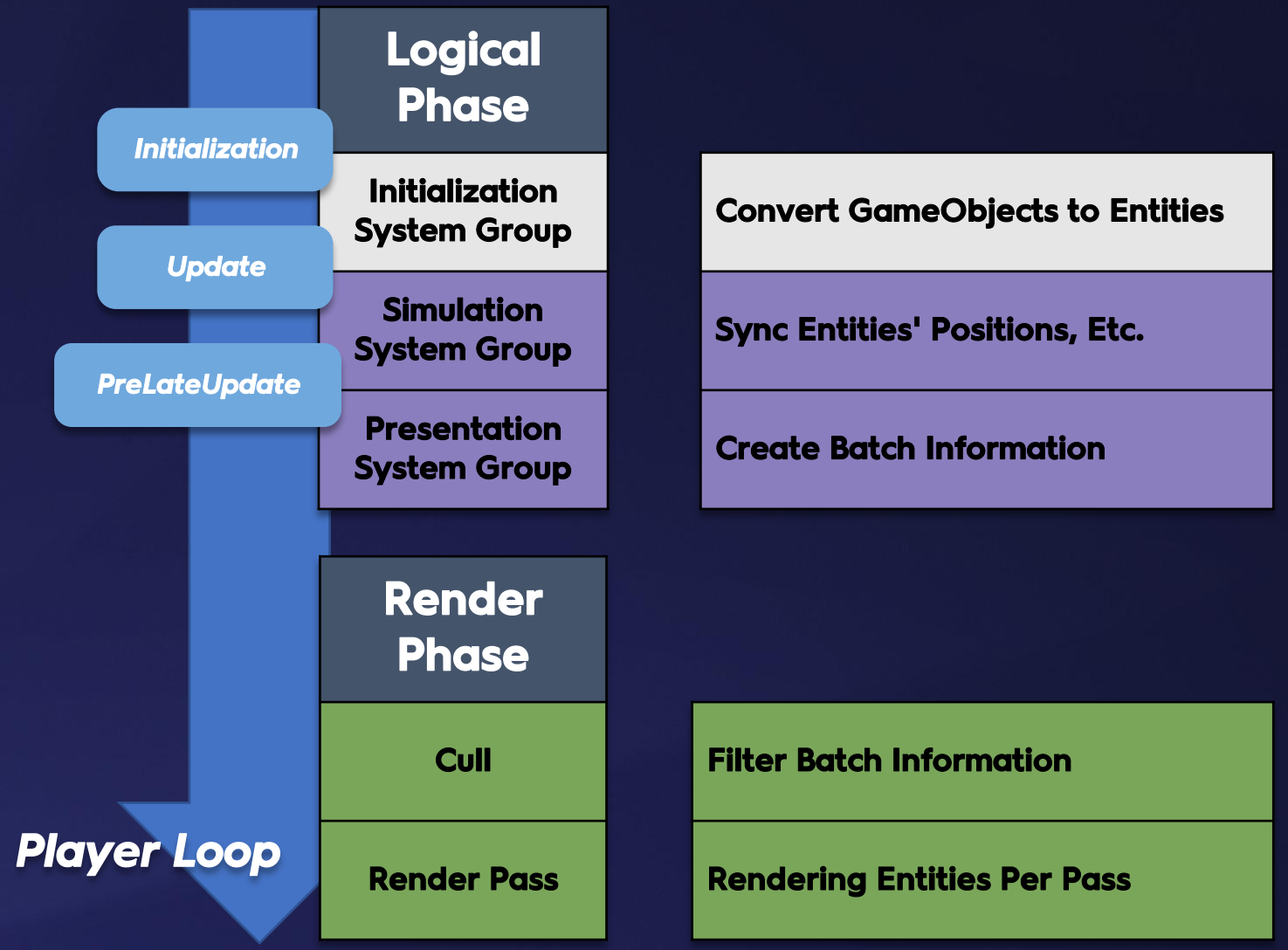
3D Top-Down View Strategy Game

- *Freely control*
- *1000+ character units*
- *Completed the preliminary game logic development and did some optimization work*
- *Start using **DOTS** to improve game performance*

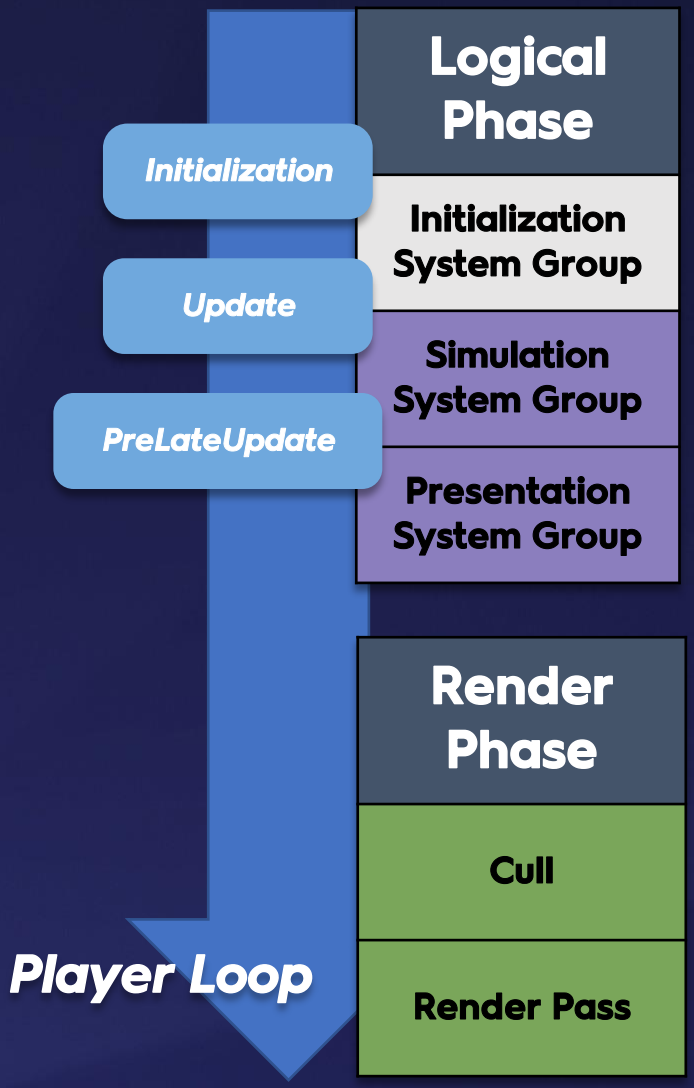
Technical Target

- Compatible with **OpenGL ES3.0** (need support **GPU instancing**)
- **Custom** Unity 2019.4.x (source access)
- **Custom** Unity HybridRender 0.5.2-preview.4 with **Hybrid Render V1**

Hybrid Render Pipeline

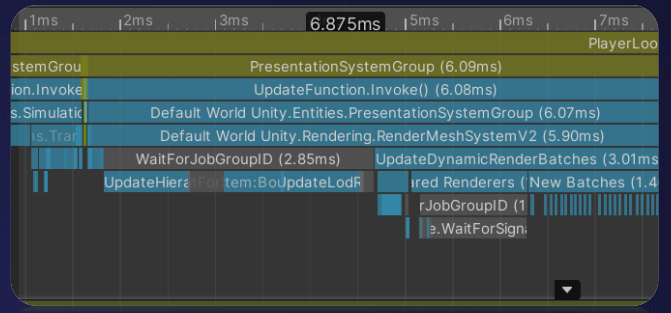


Challenges



>> Asset Compatibility

>> Base Load too High



>> No Custom Shader Material Overrides

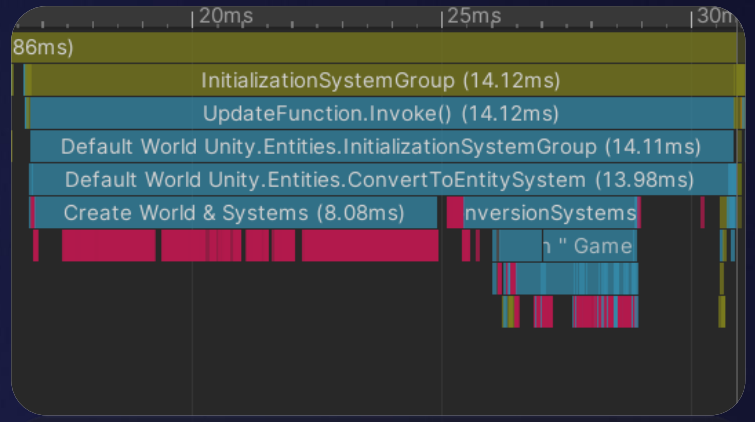
Unity's suggestions

SubScene: store a large number of objects

- Conflict with our existing asset production workflow
- Conversion is unnecessary or not supported in the scene

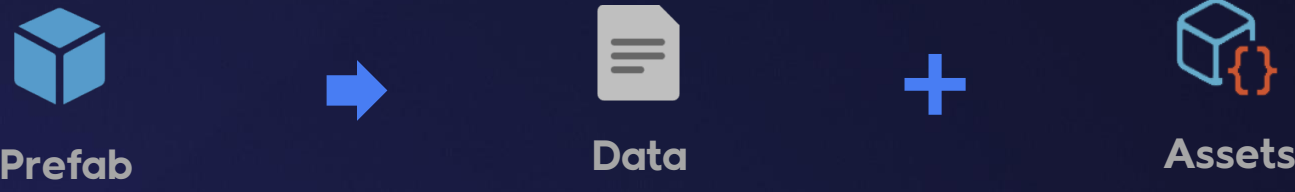
Runtime Conversion

- Slow: Take 10ms+ to convert a single gameObject
- Risky: Component types may not be supported



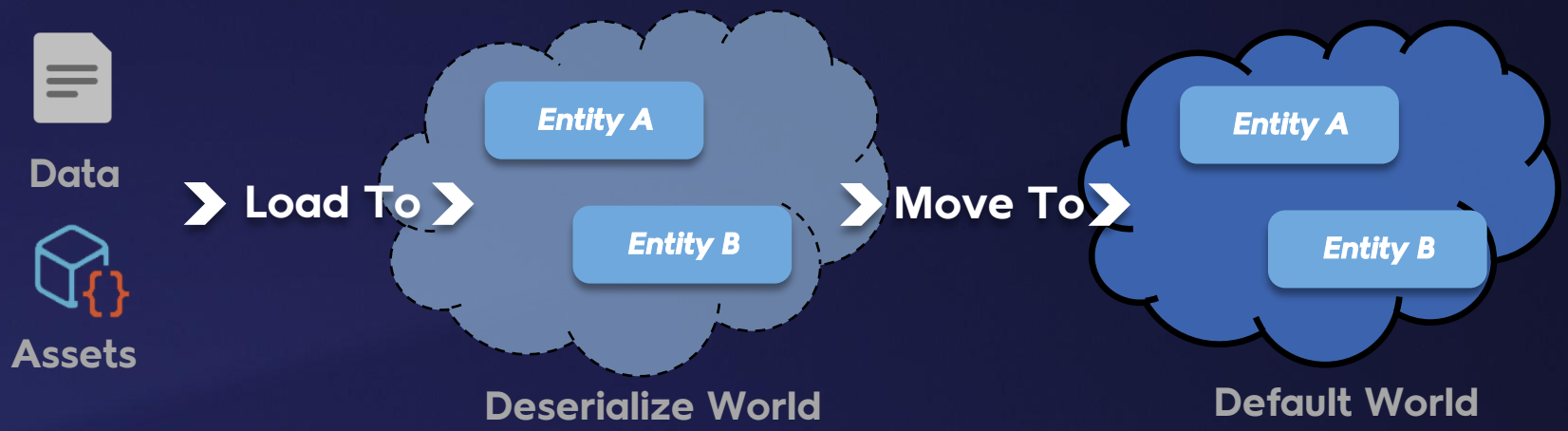
Our Solution

Offline



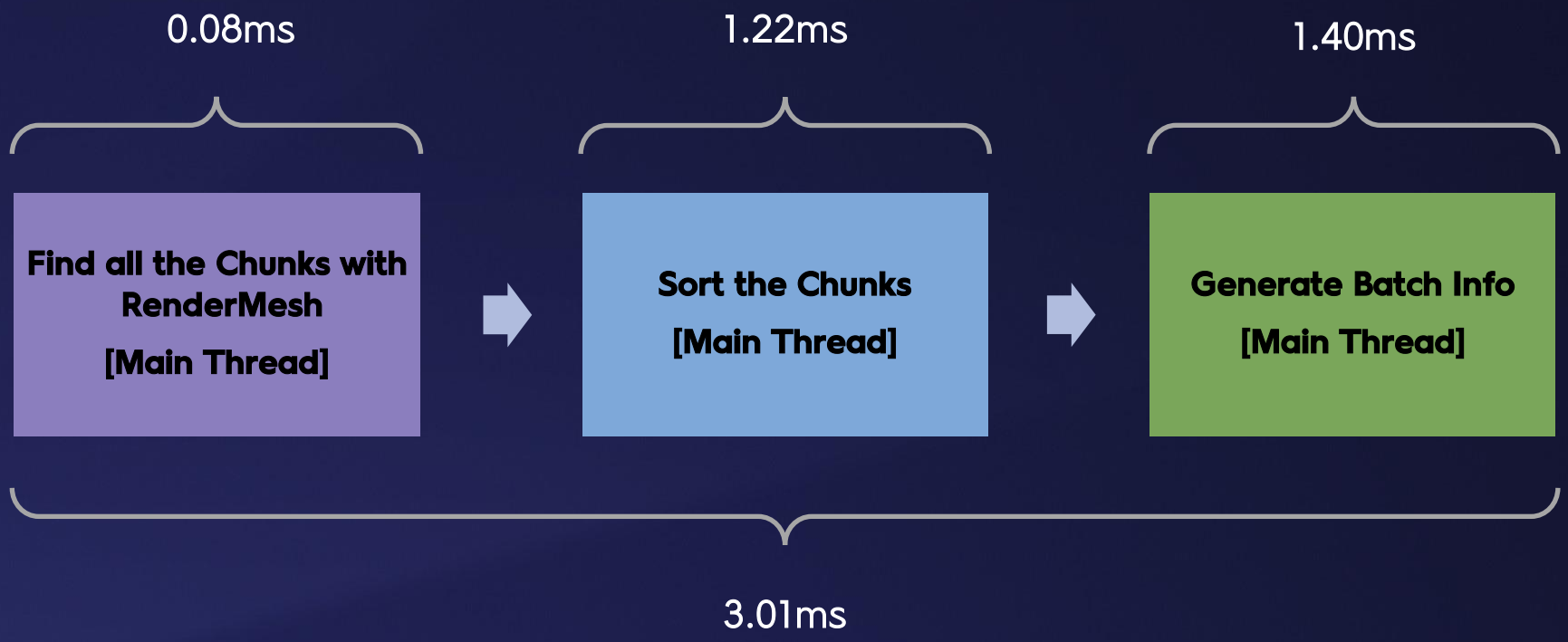
Runtime

~1+ms

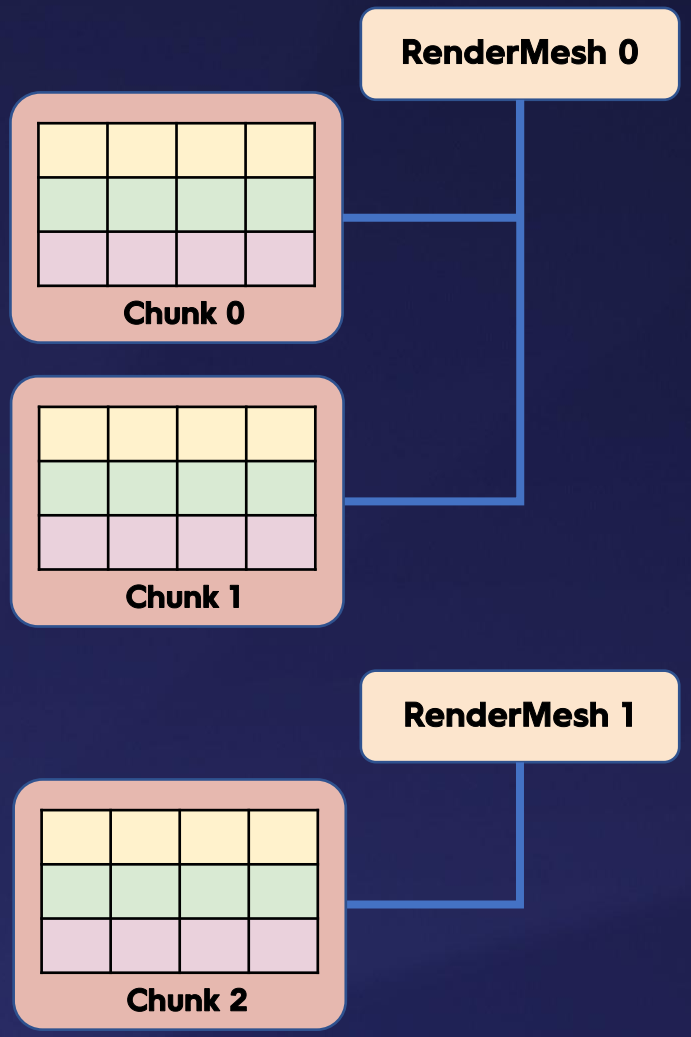


Create Batch Info with Multi-Threading

A typical entity batch processing workflow

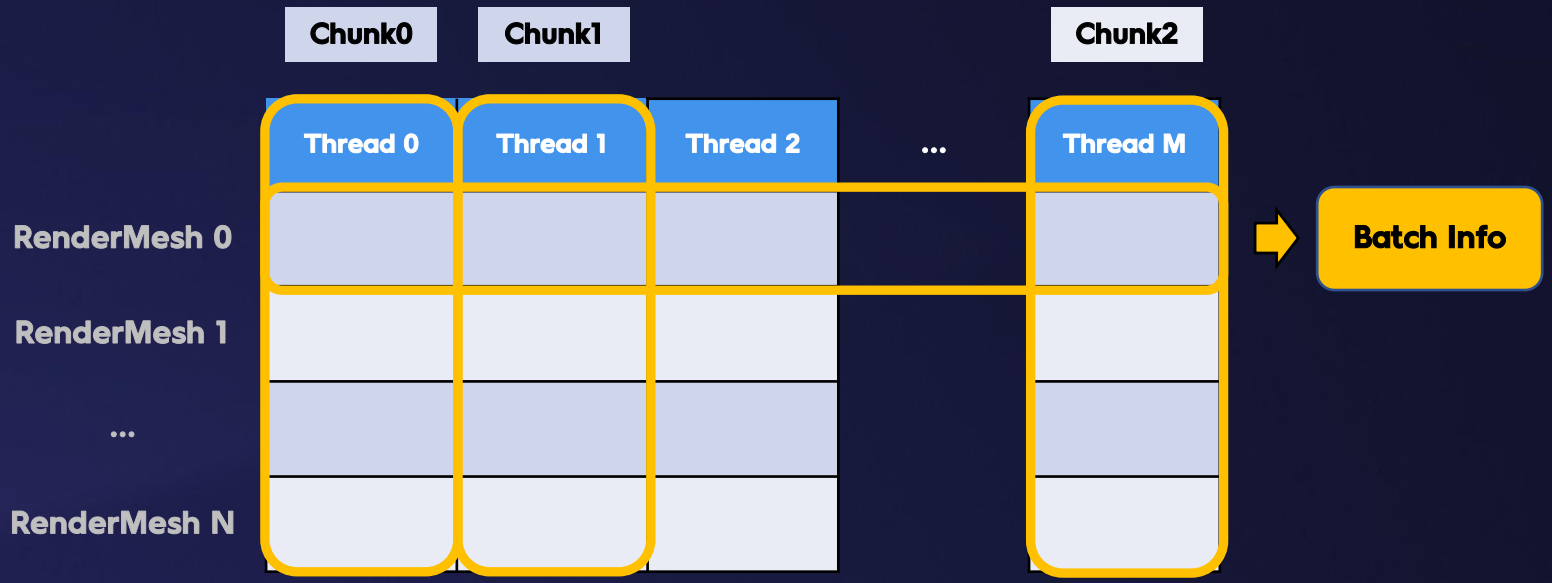


Create Batch Info with Multi-Threading



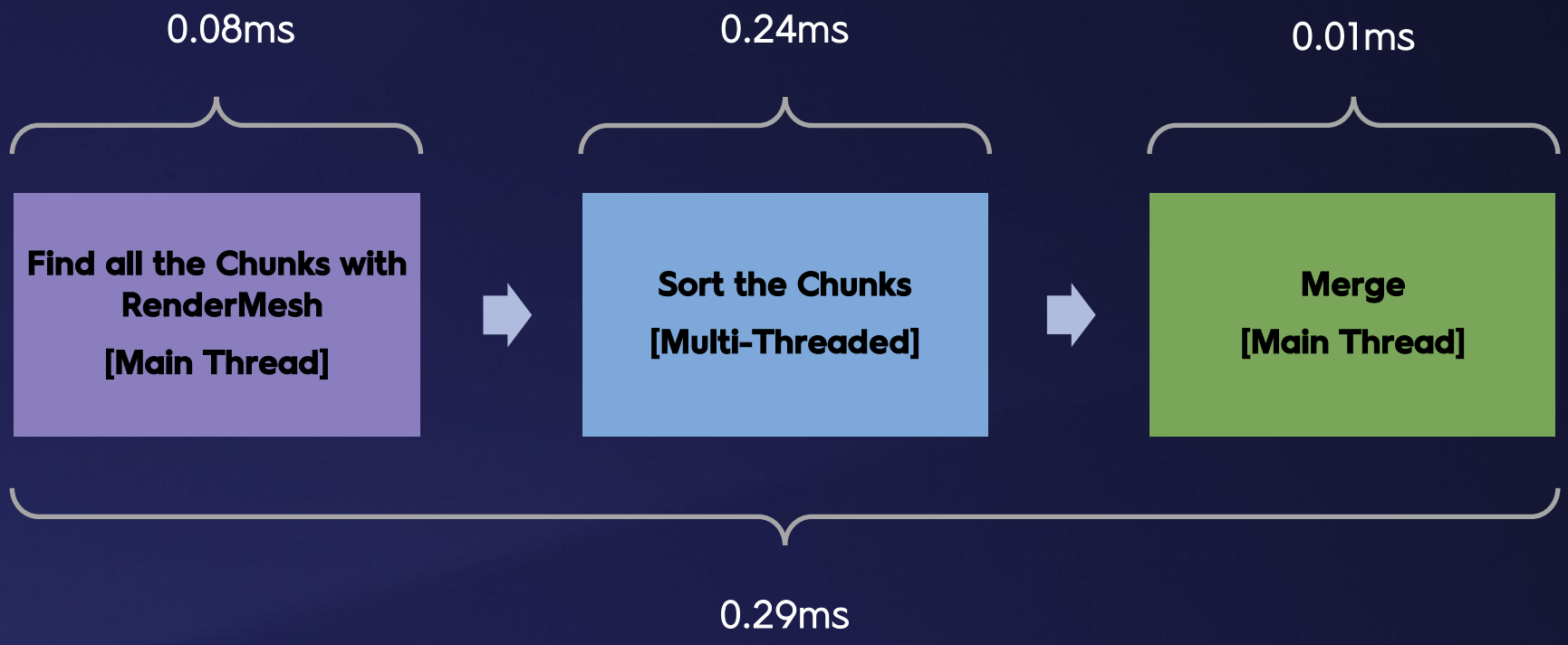
*Different **RenderMeshes** mean Different **Batches***

*Using **multi-threading** to find all chunks that share the **same RenderMesh***



ChunkPosition = RenderMeshIndex * maxThreadIndex + currentThreadIndex

Optimized multi-threaded batch processing workflow



Prefab LOD Structure

- Prefabs in our game have four levels of LOD

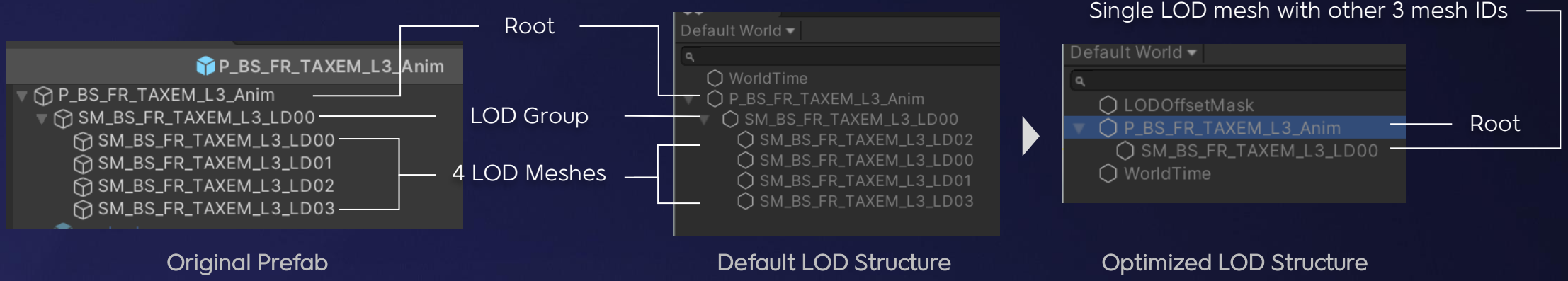
LOD 0

LOD 1

LOD 2

LOD 3

LOD Group Structure Optimization



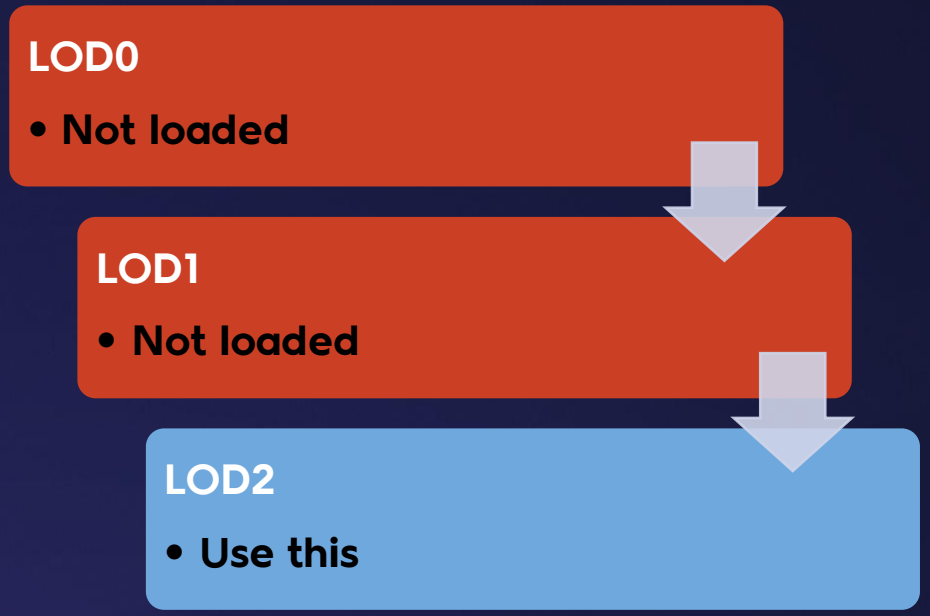
Original Prefab | Default LOD Structure | Optimized LOD Structure

Less entities, fewer chunks, better performance

- **Sync Positions : 2.2ms → 0.94ms**
- **Create Batch Information: 0.29ms → 0.18ms**

LOD Streaming

- *Higher LODs are replaced with lower LODs when they are **not loaded** or when there is rendering pressure*



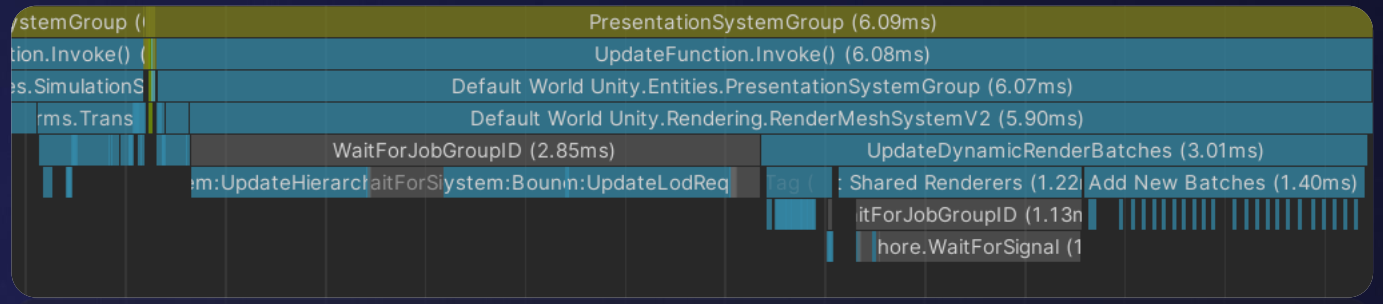
Faster loading time, lower memory usage, higher rendering efficiency

Base Load Optimization

No more main thread bottleneck

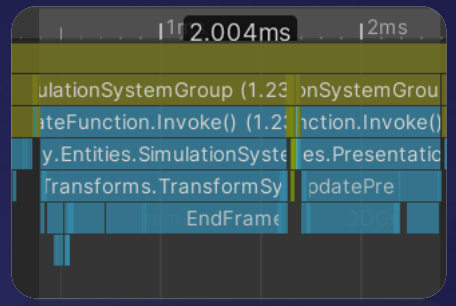
Before

~6ms



After

~2ms



Custom Shader Material Overrides

GPU-friendly data

- *Create a matching IComponentData struct for every custom shader instanced property*
- *All member types in the struct are float4, conforming with the std140 layout*

C# Define

```
[MaterialInstanceBuffer( materialPropertyName: "UnityInstancing_AnimProps", strideInBytes: 32)]
public struct AnimPropsComponentData : IComponentData
{
    public float4 FrameIndices;
    public float4 LerpWeights;
}
```

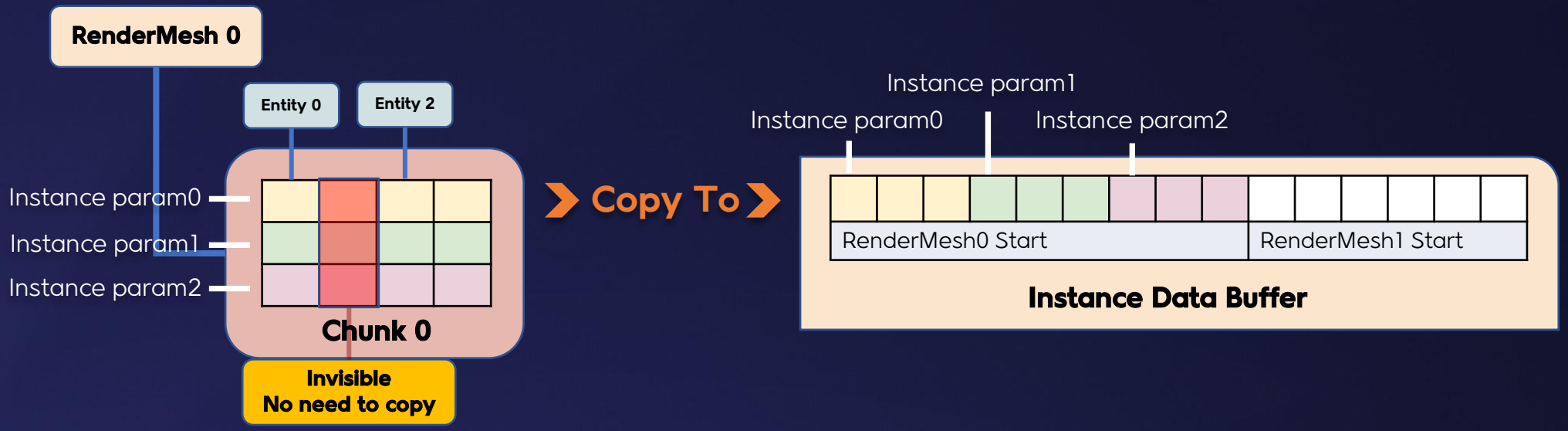
Shader Define

```
UNITY_INSTANCING_BUFFER_START(AnimProps)
    UNITY_DEFINE_INSTANCED_PROP(float4, FrameIndices)
    UNITY_DEFINE_INSTANCED_PROP(float4, LerpWeights)
UNITY_INSTANCING_BUFFER_END(AnimProps)
```

Compiled Shader

```
struct AnimPropsArray_Type {
    vec4 FrameIndices;
    vec4 LerpWeights;
};
layout(std140) uniform UnityInstancing_AnimProps {
    UNITY_UNIFORM AnimPropsArray_Type AnimPropsArray[UNITY_RUNTIME_INSTANCING_ARRAY_SIZE];
};
```

Optimized Instance param copy



- *Pre-allocate a large buffer based on the number of entities*
- *Copy the Instance data of the RenderMeshes to be displayed into the allocated buffer*

URP Compatibility

Filter RenderMeshes

- Filter RenderMeshes to be drawn based on the current pass configuration, with caching

Render RenderMeshes

- Submit the collected data
- Disregard camera distance sorting (based on TBR)

Thank You

Jian Xiao

jamesjxiao@tencent.com

Cangjian Hou

cangjianhou@tencent.com