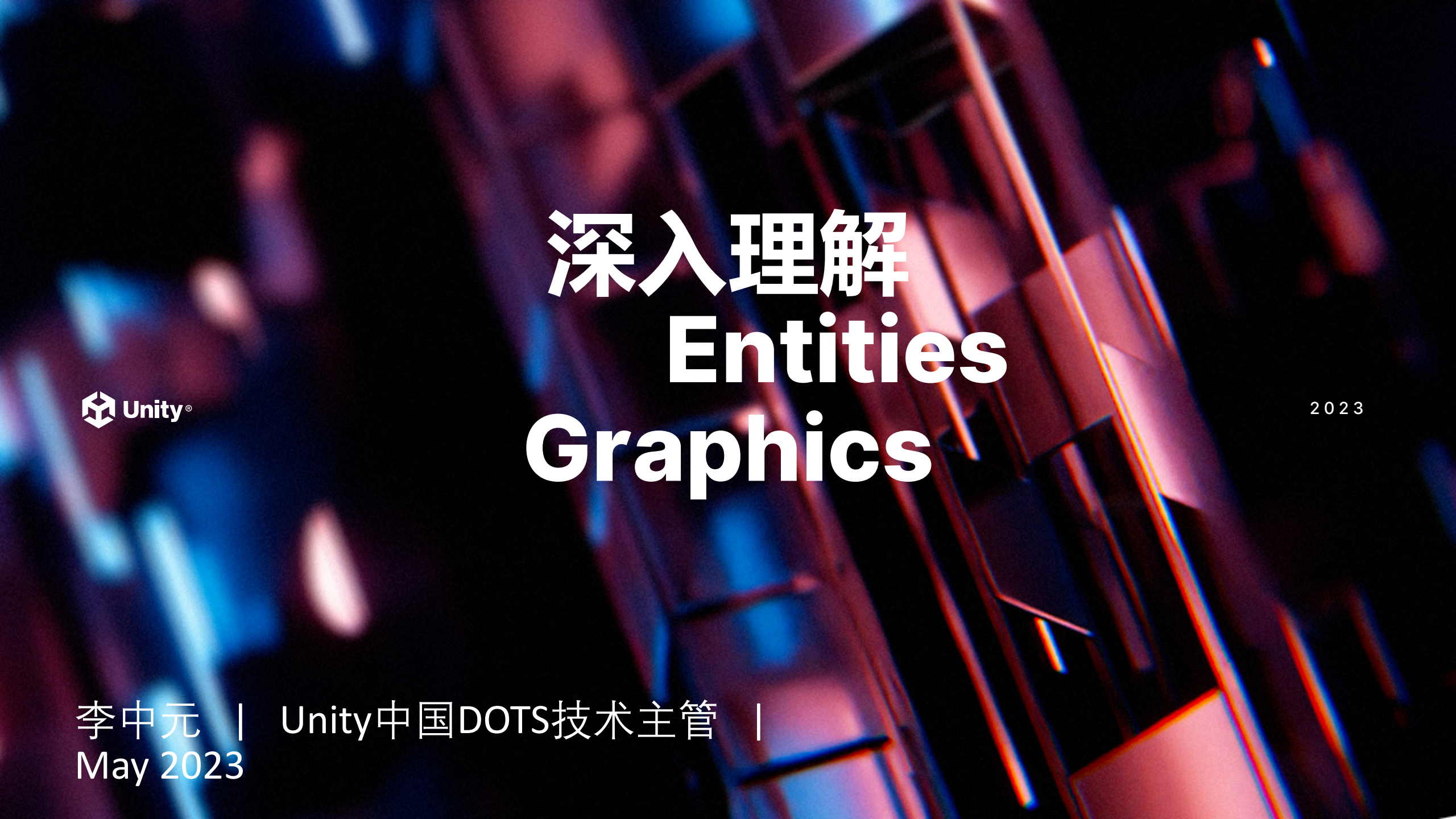


深入理解 Entities Graphics

Unity®

2023

李中元 | Unity中国DOTS技术主管 |

May 2023

Agenda

— Entities Graphics 简介

— Entities Graphics 架构设计

— Entities Graphics 底层原理

Entities Graphics简介

- Hybrid Renderer
  - V1 Ship with MegaCity Demo
  - V2 totally redesigned version
- 不是 “pure” DOTS renderer
  - Mesh
  - Material
  - Shader

Entities Graphics简介

— SRP & V1

Unity®

Entities Graphics简介

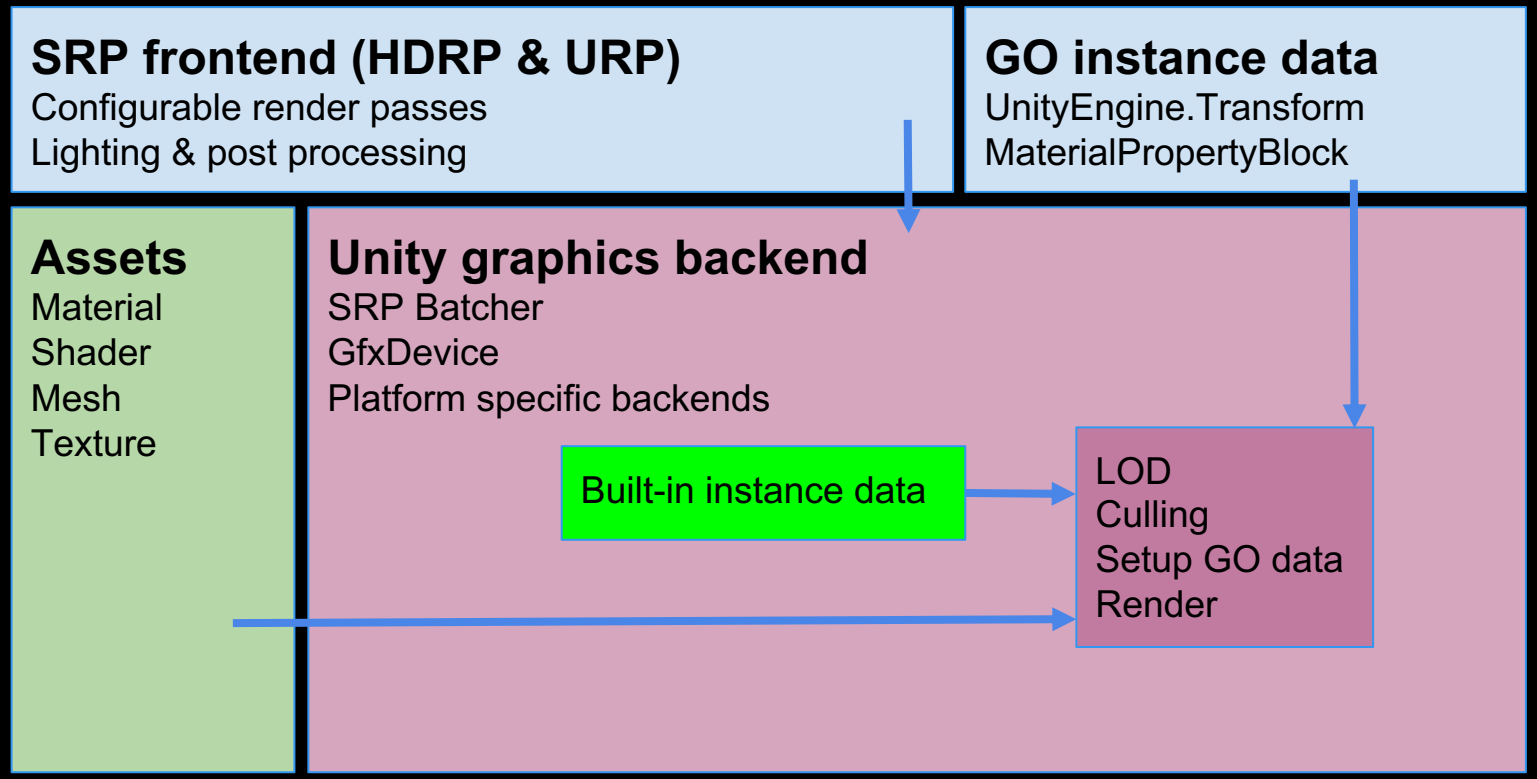
- SRP & V1
  - Main thread build batches
  - Slow constant buffer setup
- V1
  - Instancing shader variant broken
  - Specific ShaderGraph and material authoring workflow
  - Missing HDRP/URP features

Entities Graphics简介

- Entities Graphics Goals
  - New GPU data persistent model
    - Full access to all (HDRP and URP) shader input data from DOTS C# code
    - Automatic delta update of DOTS ECS chunks to GPU memory
  - Persistent batches
    - All batches are fully persistent.
    - Iterative batch changes: Added/removed entities/components

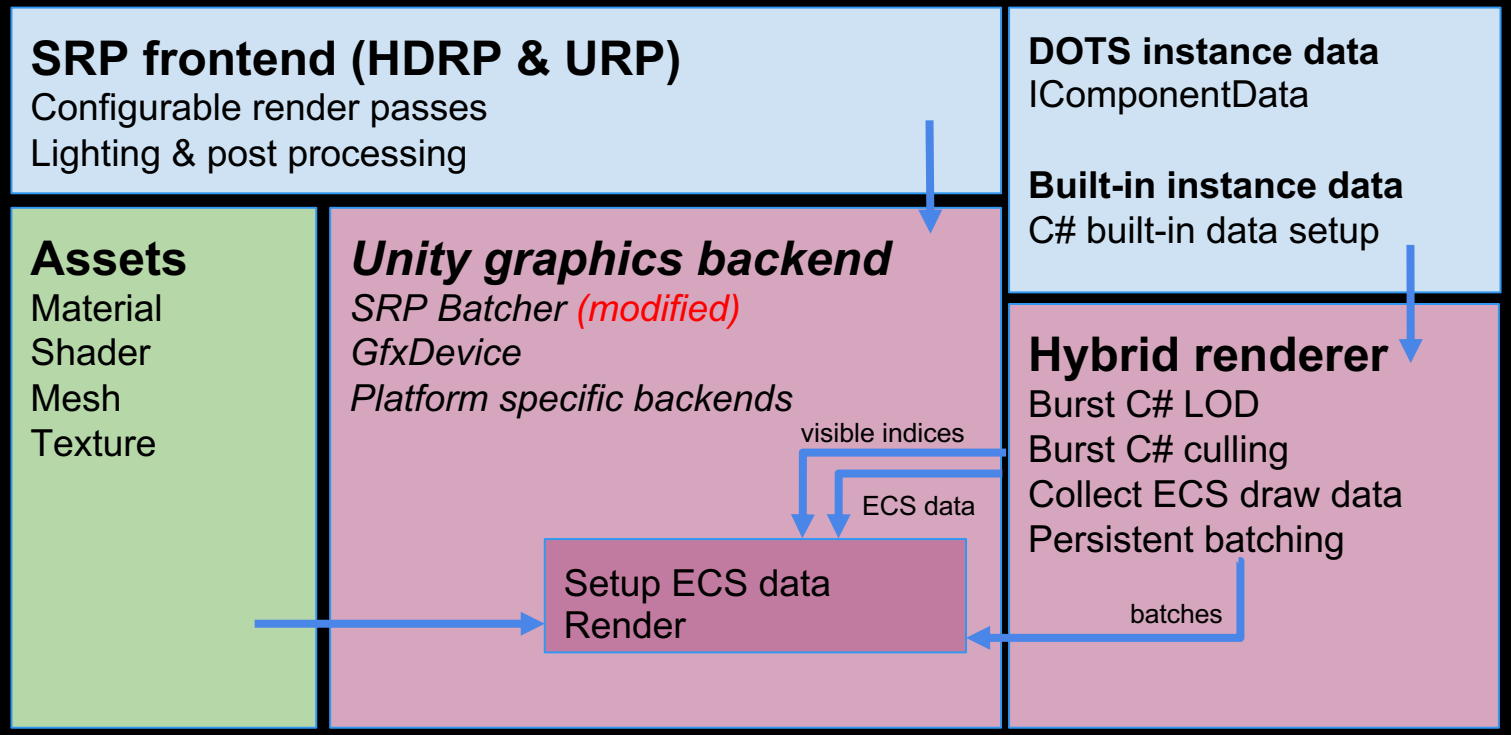
Entities Graphics架构设计

— Traditional Unity GameObject rendering architecture

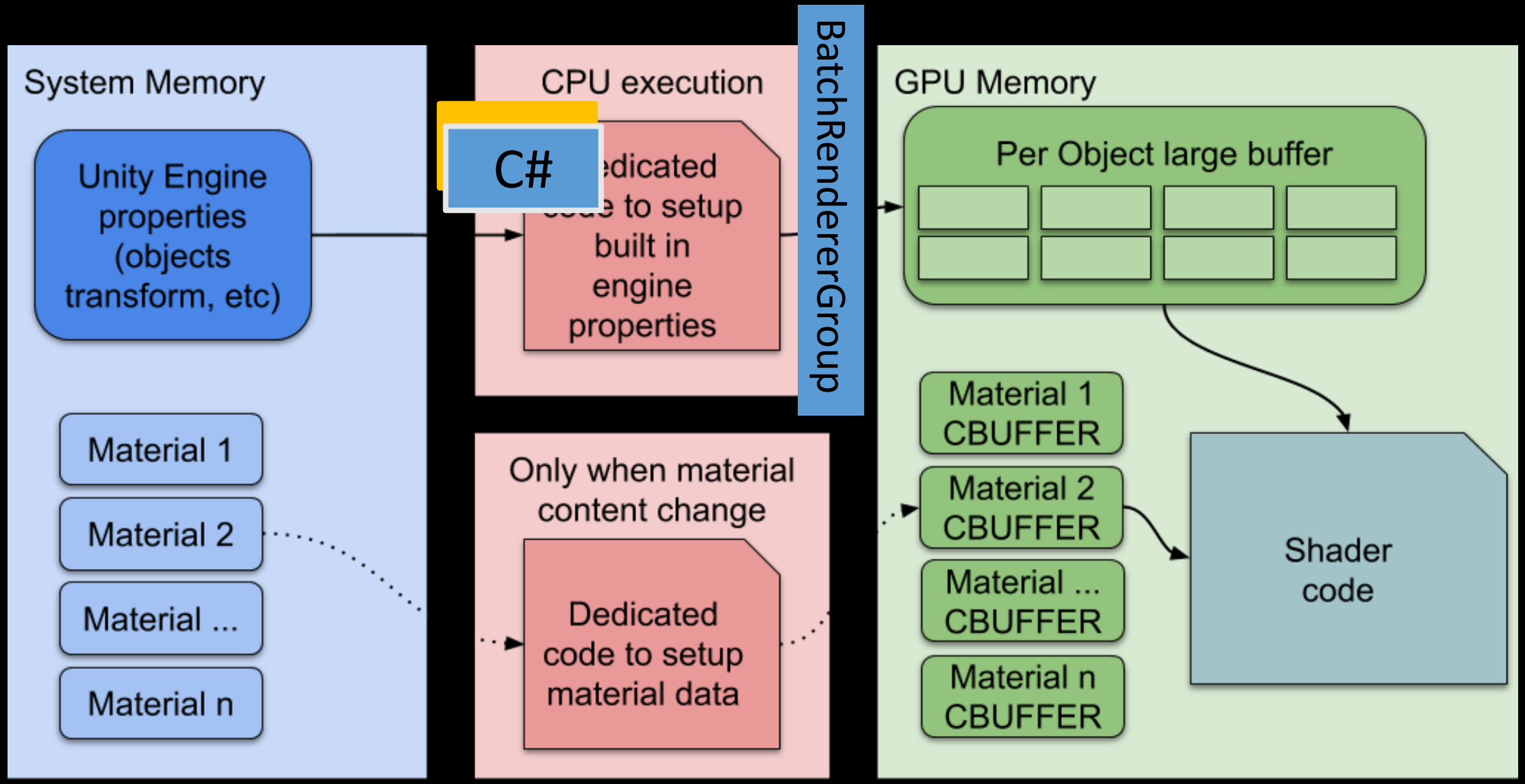


Entities Graphics架构设计

— Unity Entities Graphics architecture



Entities Graphics架构设计



unity

Entities Graphics架构设计

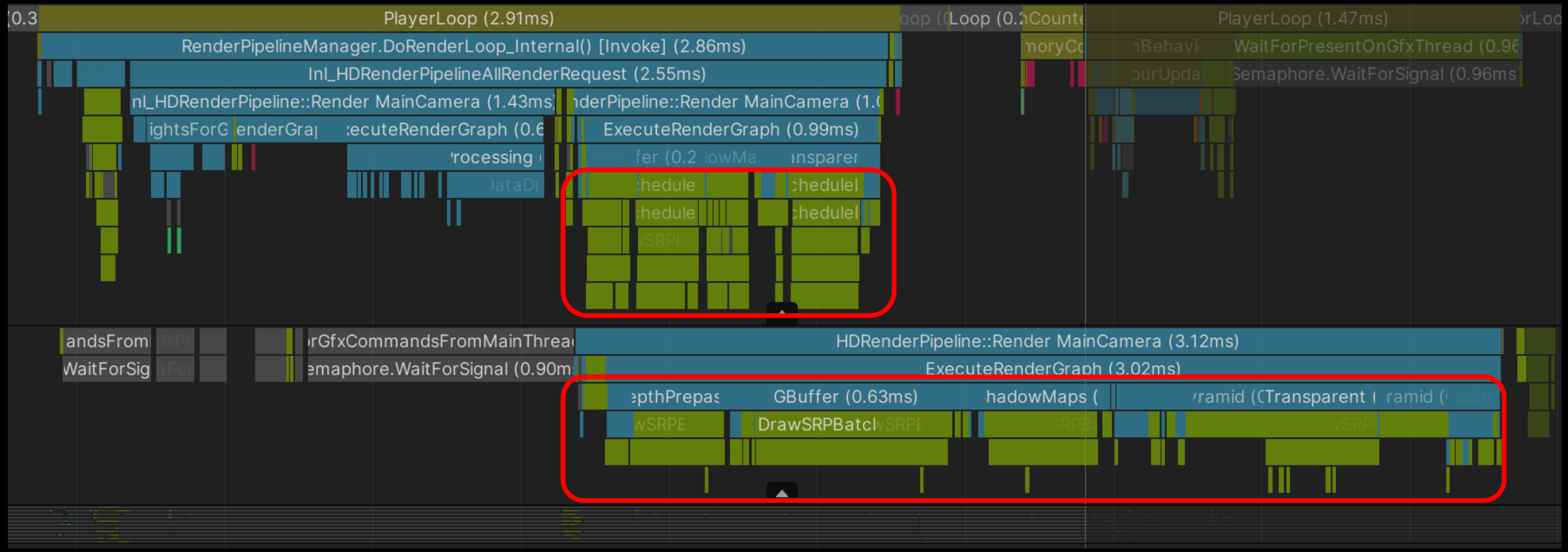
— Entities Graphics(Hybrid Renderer) is not a new renderer!
— It’s a new **data path** from DOTS to GPU
— Requires SRP Batcher

Entities Graphics架构设计

— Data Persistent Model
— Persistent Batches

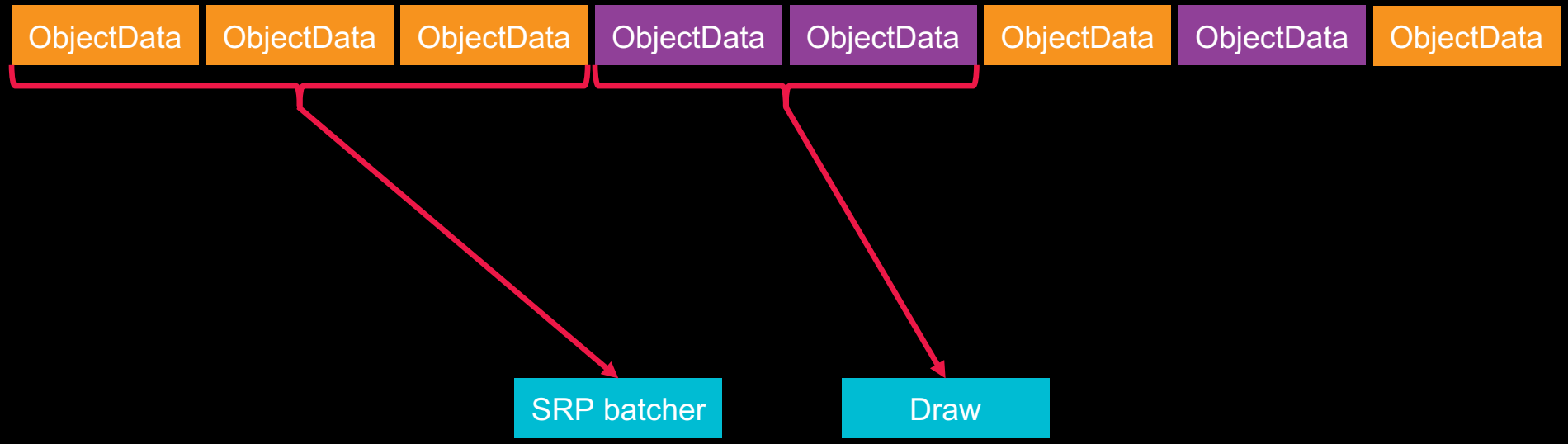
Data Persistent Model

Data Persistent Model

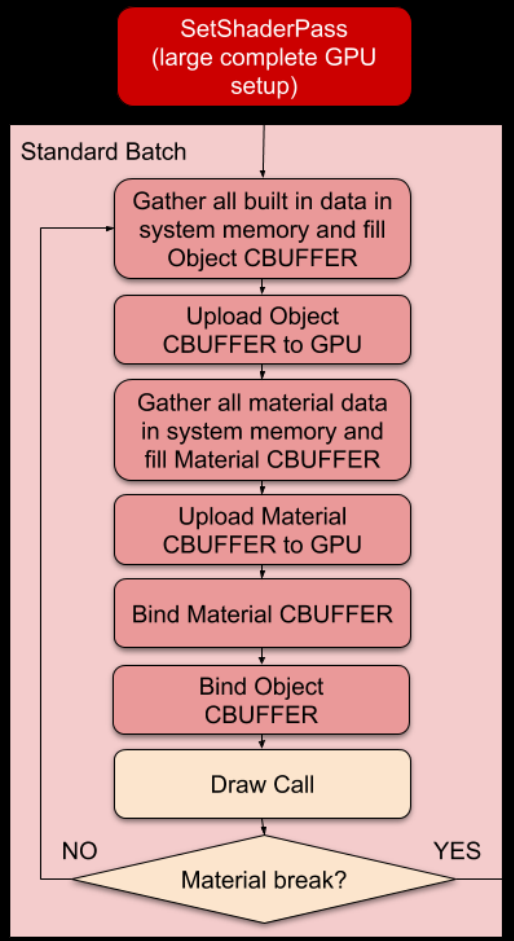


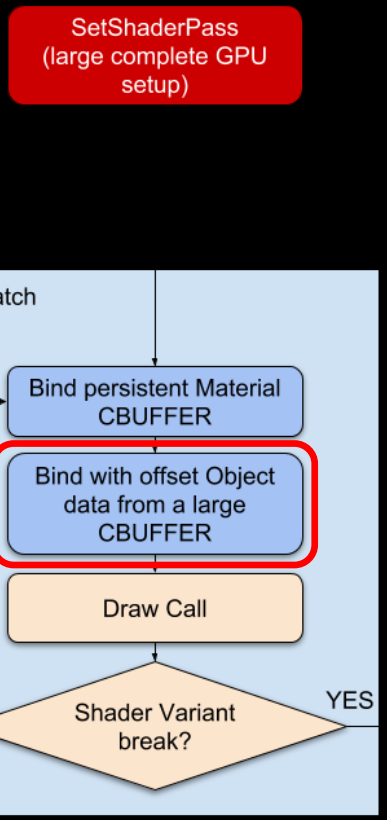
Scriptable Render Loop

– ScriptableRenderLoopDrawDispatch()



Data Persistent Model



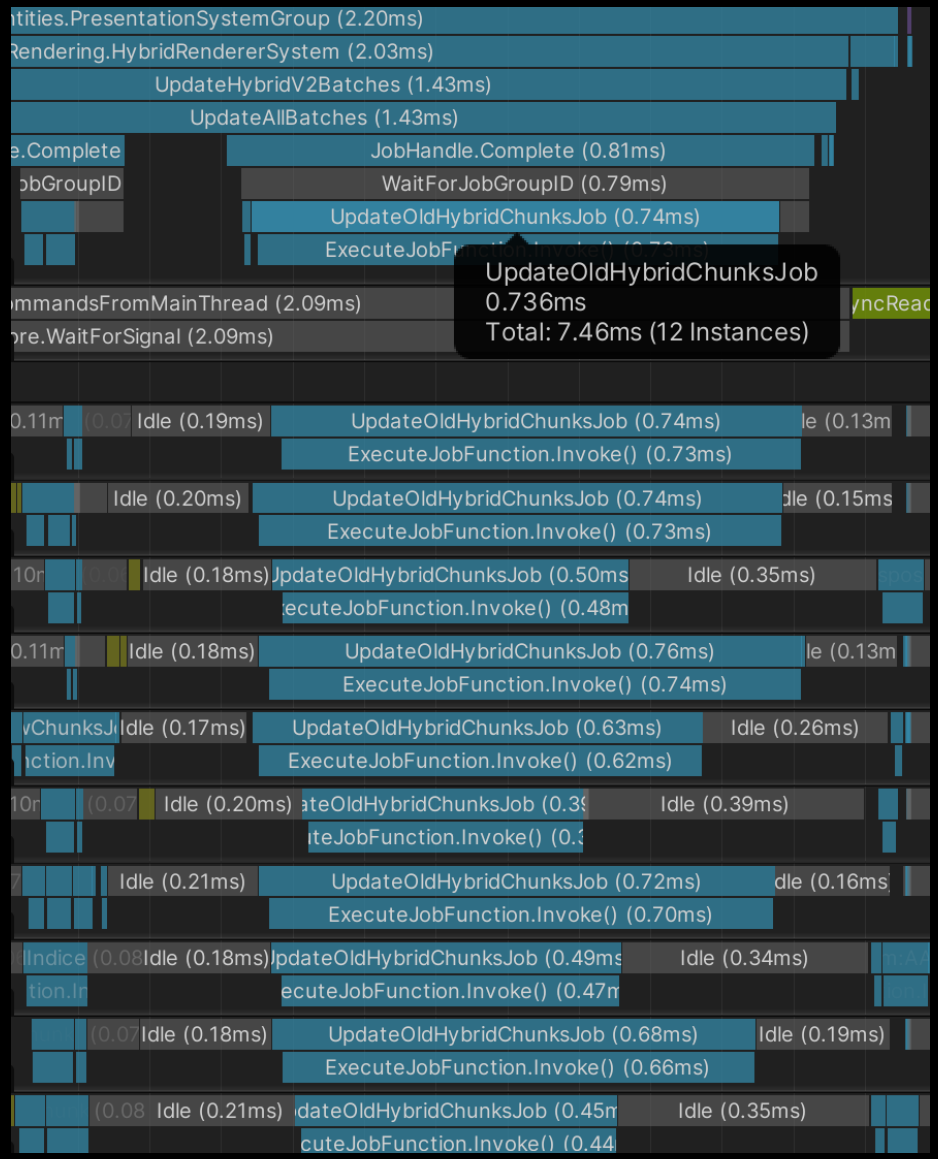


Data Persistent Model

- Parallel Burst jobs write directly to GPU memory
  - New ComputeBuffer SubUpdate mode: Pointer to GPU memory (upload heap)
  - No main/render thread processing of GPU instance data anymore
  - Data uploaded once per entity. No per viewport entity constant buffer setup

Data Persistent Model

— Parallel Burst jobs write directly to GPU memory

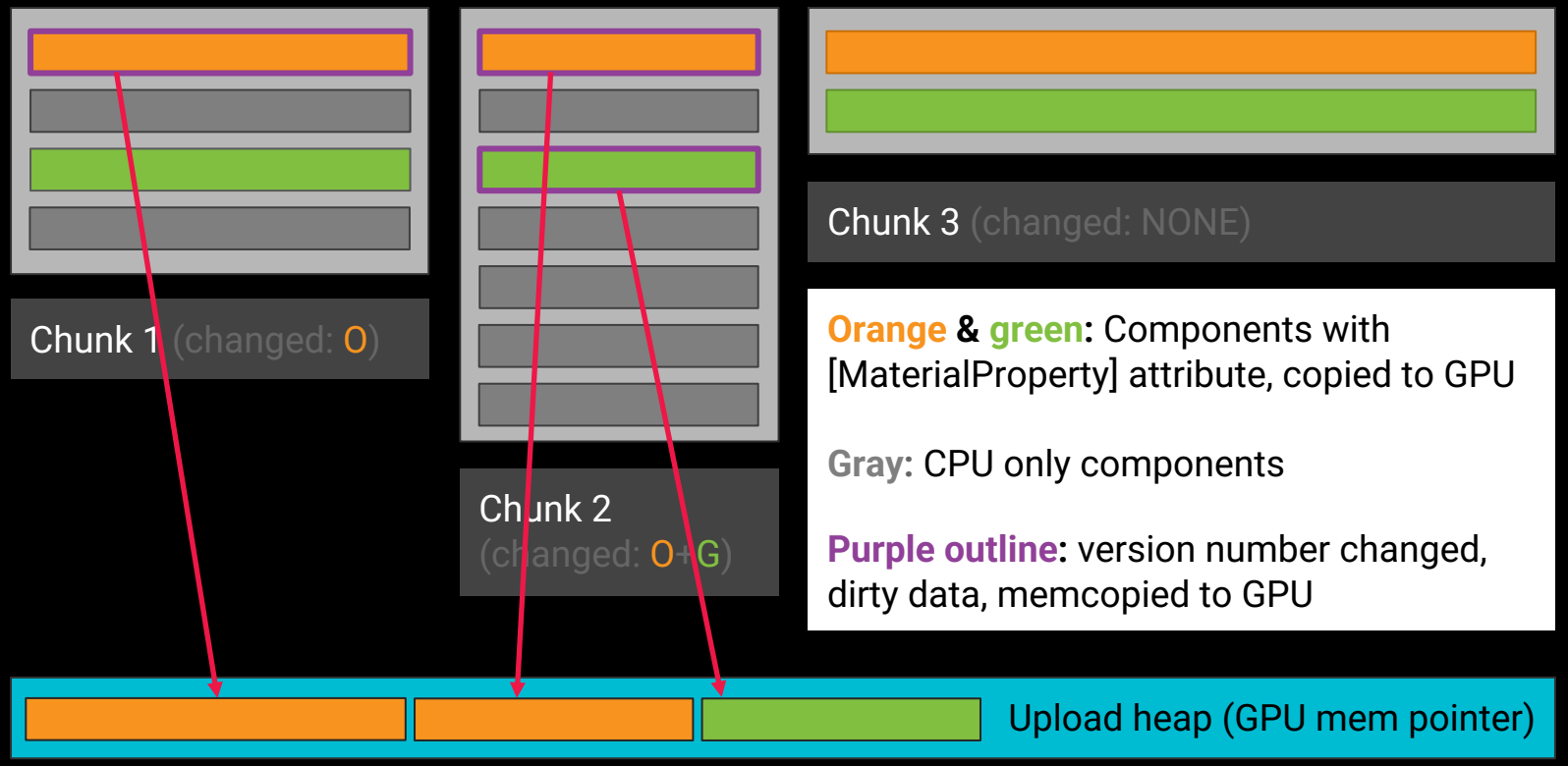


Unity®

Data Persistent Model

- Automatic delta update of DOTS ECS chunks to GPU
  - Automatic CPU->GPU update of all IComponentData with [MaterialProperty] attribute
  - Based on DOTS chunk.DidChange<T> write access tracking

Delta update



Persistent Batches

Persistent batches

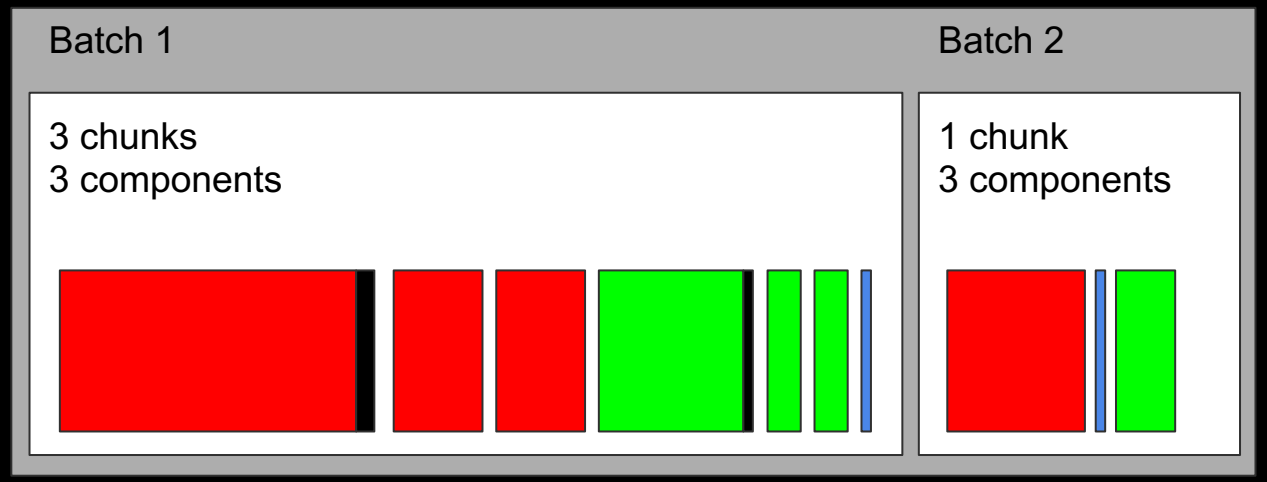
- All batches are fully persistent
  - Performance by default: Manual setup needed for Frozen batches is gone
  - Skip processing of chunks that didn’t change
- Iterative batch changes: Added/removed entities/components
  - Batch capacity = sum(chunks.capacity)
  - Empty slots at end of each chunk in GPU memory, just like in DOTS ECS
  - **Advantage**: Full batch rebuild is NOT needed when entity is added/removed from chunk

Persistent batches

- Burst C# parallel batch update jobs
  - Main thread bottleneck is gone!
- DisableRendering tag component
  - Use cases: sector based culling, optimizing cinematics, etc...

GPU ECS data layout

- All GPU ECS data is in a suballocated giant ComputeBuffer
- Each property of each batch gets a contiguous range from the buffer, with a contiguous sub-range for each chunk, one value per entity
- Data is persistent from frame to frame, no need to upload same values each frame



Red & green: Properties with per-entity IComponentData

Blue: Properties using shared material default value

Black: empty slots (chunk end)

Performance

Performance

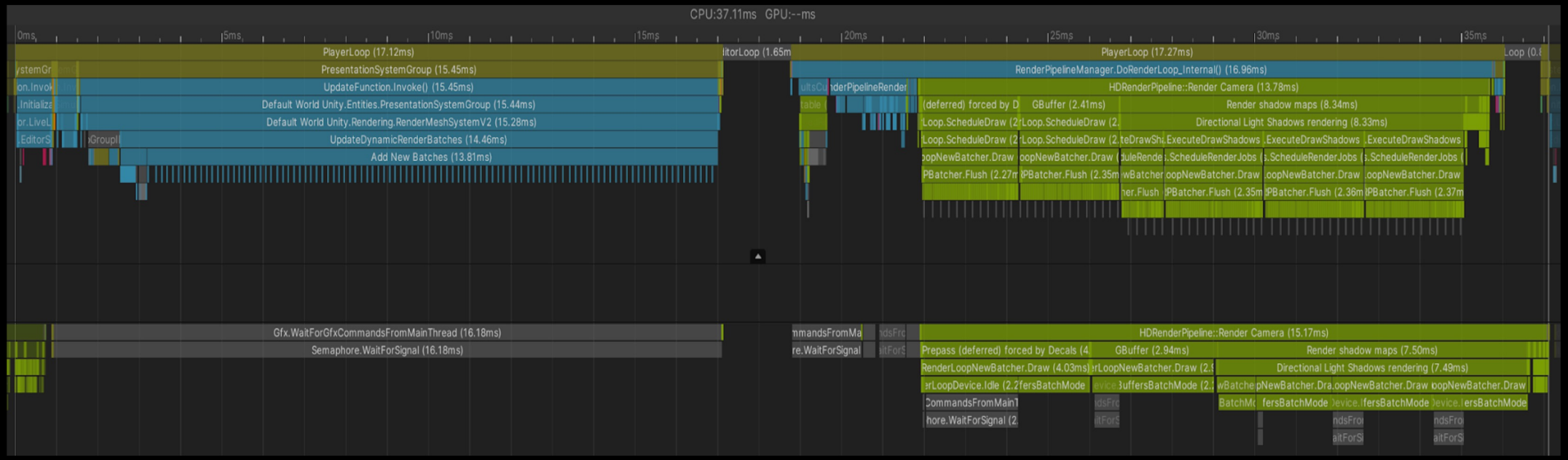
- Stress test scene
  - 100K dynamic entities
  - Animated: LocalToWorld and color
  - Perfectly batched (single shader and mesh)
- Test scene with more shader and mesh variation under development
  - Will give more realistic results. Gains will be smaller.

Performance

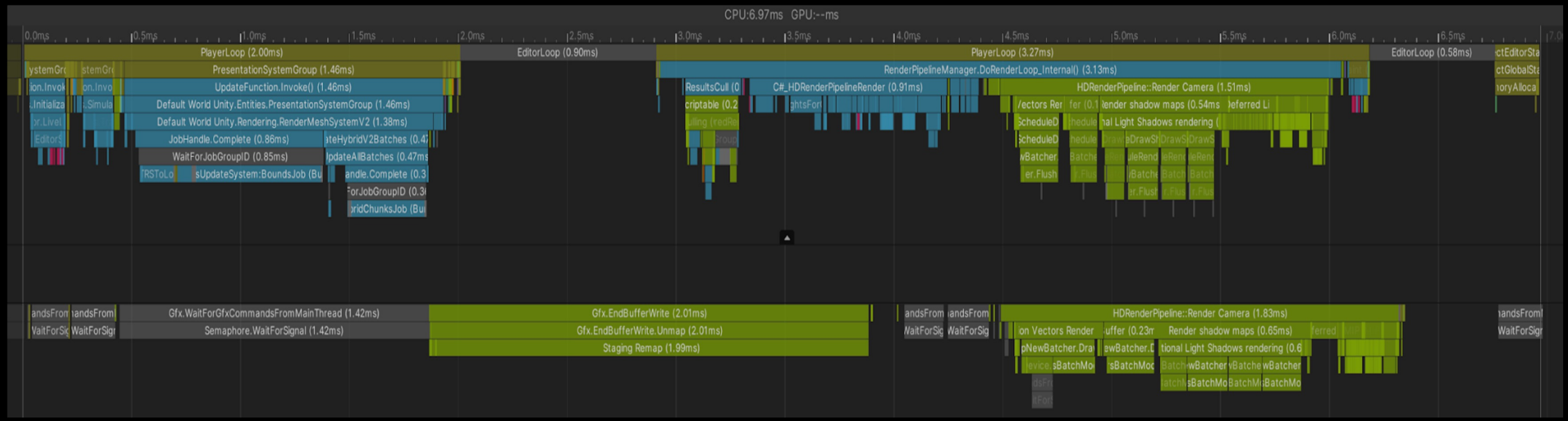
- Preliminary performance results
  - Around 5x faster frame time on high end laptops
  - Around 30% faster frame time on Qualcomm/ARM mobile GPUs
    - While using 2x-3x less CPU cycles
- Further optimizations in development

Hybrid V1 performance (100K dynamic entities)

using single mesh and single material



Hybrid V2 performance (100K dynamic entities) using single mesh and single material



Unity®

Entities Graphics底层原理

- Batch Renderer Group
  - Rewritten in unity 2022.1
  - OpenGL ES 3 supported int 2022.2

BatchRendererGroup API

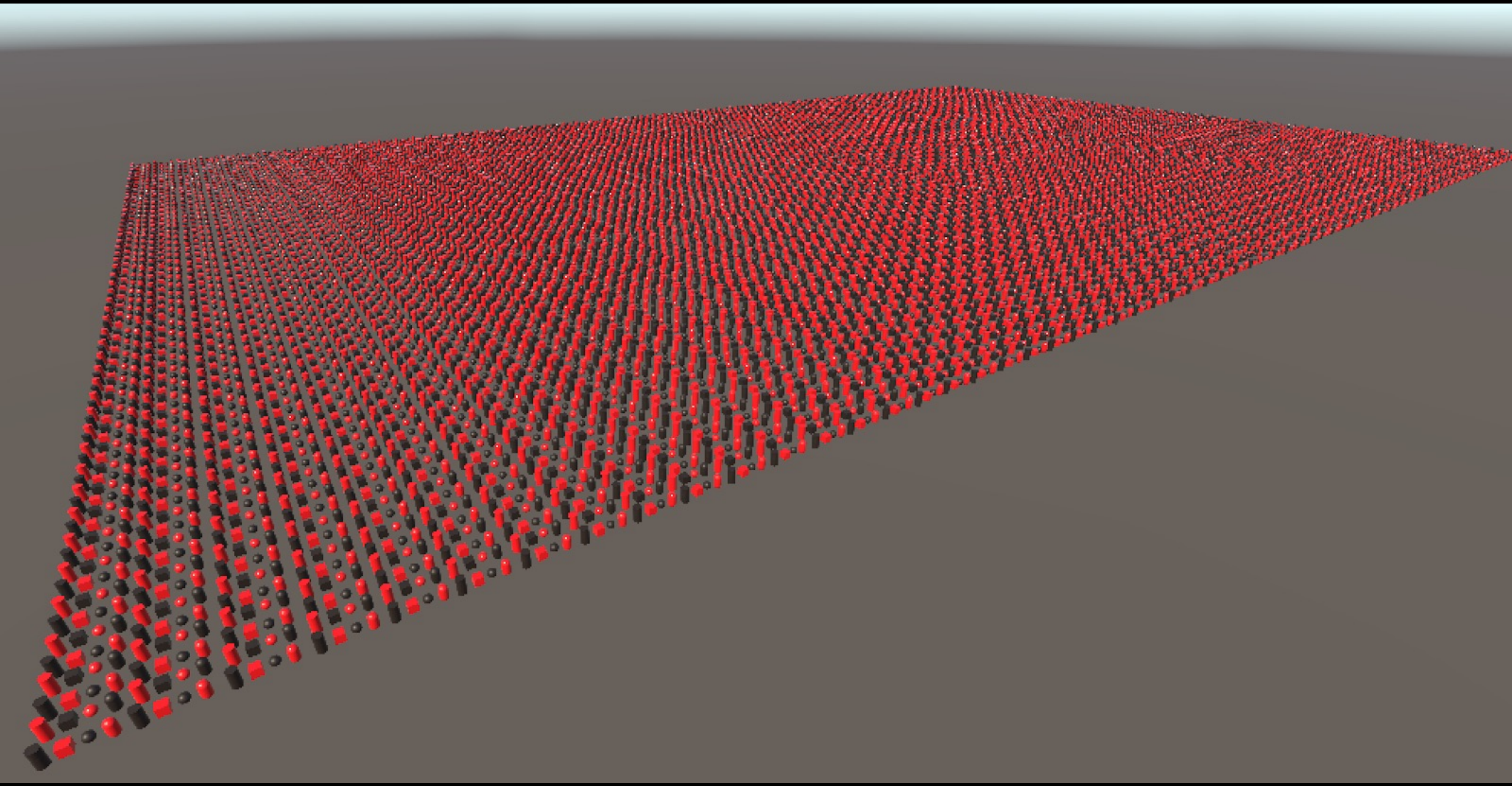
```
// Create the BatchRendererGroup and register assets
m_BRG = new BatchRendererGroup(this.OnPerformCulling, userContext: IntPtr.Zero);
m_MeshID = m_BRG.RegisterMesh(mesh);
m_MaterialID = m_BRG.RegisterMaterial(material);
```

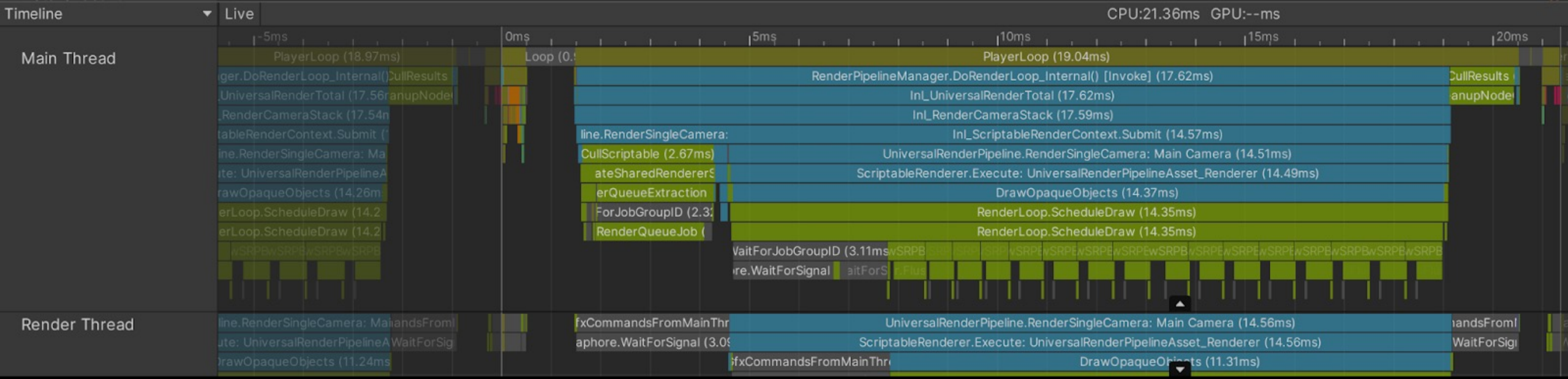
BatchRendererGroup API

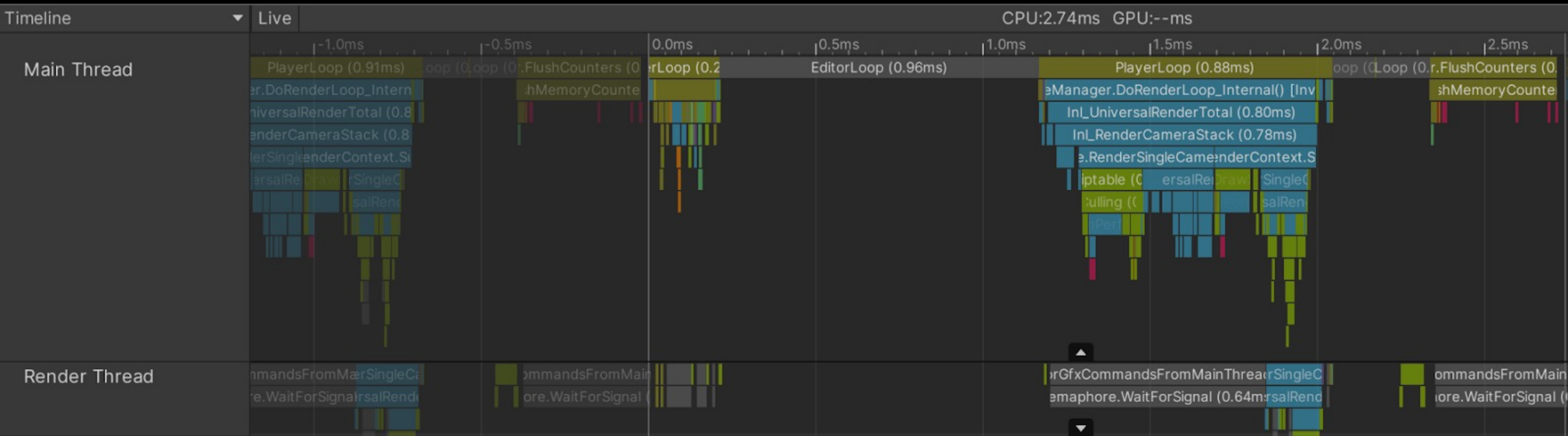
```
var metadata = new NativeArray<MetadataValue>(length:3, Allocator.Temp);
metadata[0] = new MetadataValue {NameID = Shader.PropertyToID(name:"unity_ObjectToWorld"), Value = 0x80000000 | _byteAddressObjectToWorld,};
metadata[1] = new MetadataValue {NameID = Shader.PropertyToID(name:"unity_WorldToObject"), Value = 0x80000000 | _byteAddressWorldToObject,};
metadata[2] = new MetadataValue {NameID = Shader.PropertyToID(name:"_BaseColor"), Value = 0x80000000 | _byteAddressColor,};

m_BatchID = m_BRG.AddBatch(metadata, m_InstanceData.bufferHandle, (uint) BufferOffset, (uint) BufferWindowSize);
```

BatchRendererGroup API

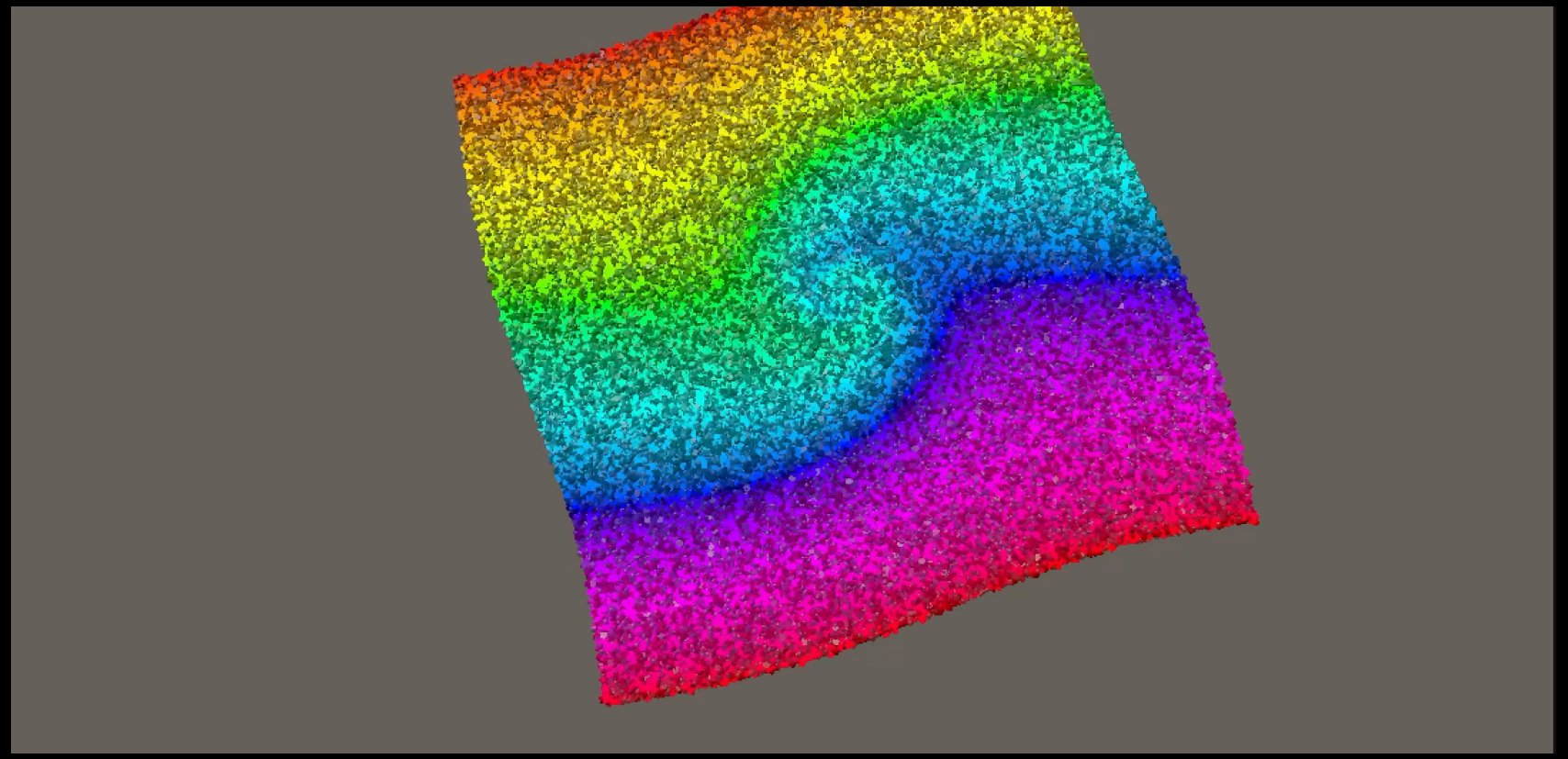
```
public unsafe JobHandle OnPerformCulling(
    BatchRendererGroup rendererGroup,
    BatchCullingContext cullingContext,
    BatchCullingOutput cullingOutput,
    IntPtr userContext){...}
```



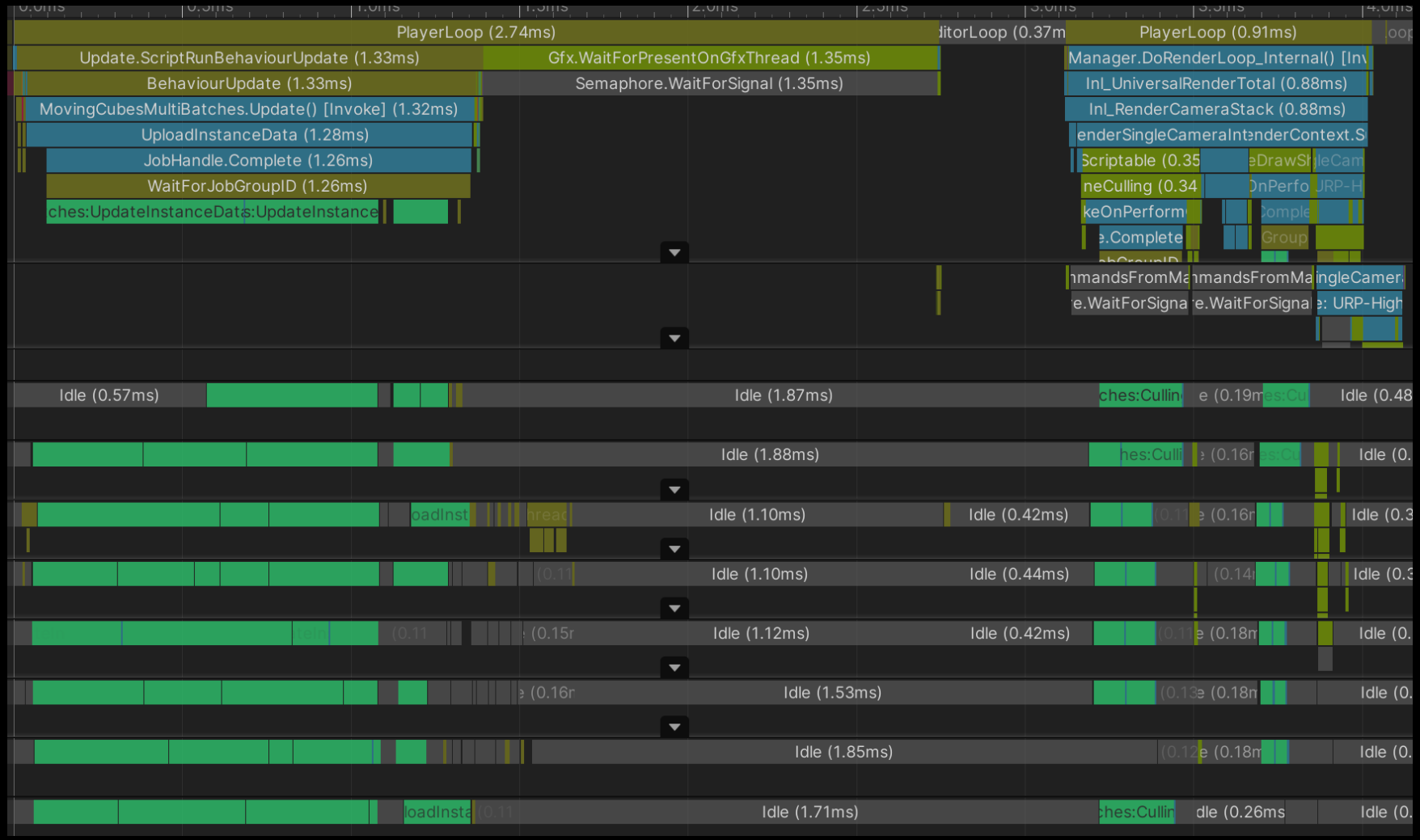

Timeline
Live
CPU:21.36ms GPU:--ms
Main Thread
-5ms
0ms
5ms
10ms
15ms
20ms
PlayerLoop (18.97ms)
PlayerLoop (19.04ms)
RenderPipelineManager.DoRenderLoop_Internal() [Invoke] (17.62ms)
InI_UniversalRenderTotal (17.62ms)
InI_RenderCameraStack (17.59ms)
InI_ScriptableRenderContext.Submit (14.57ms)
UniversalRenderPipeline.RenderSingleCamera: Main Camera (14.51ms)
ScriptableRenderer.Execute: UniversalRenderPipelineAsset_Renderer (14.49ms)
DrawOpaqueObjects (14.37ms)
RenderLoop.ScheduleDraw (14.35ms)
RenderLoop.ScheduleDraw (14.35ms)
Render Thread
UniversalRenderPipeline.RenderSingleCamera: Main Camera (14.56ms)
ScriptableRenderer.Execute: UniversalRenderPipelineAsset_Renderer (14.56ms)
DrawOpaqueObjects (11.31ms)


Timeline
Live
CPU:2.74ms GPU:--ms
Main Thread
-1.0ms
-0.5ms
0.0ms
0.5ms
1.0ms
1.5ms
2.0ms
2.5ms
PlayerLoop (0.91ms)
EditorLoop (0.96ms)
PlayerLoop (0.88ms)
InI_UniversalRenderTotal (0.80ms)
InI_RenderCameraStack (0.78ms)
Render Thread

Entities Graphics底层原理



DOTS Graphics底层原理



Unity®

DOTS Graphics底层原理

- Batch Renderer Group
  - Repo: https://github.com/vinsli/batch-renderer
  - Doc: https://docs.unity.cn/2022.1/Documentation/Manual/batch-renderer-group.html
  - Forum: https://forum.unity.com/threads/new-batchrenderergroup-api-for-2022-1.1230669/

DOTS Graphics底层原理

- Difference with DrawMeshInstanced
  - DrawMeshInstanced
    - Upload matrices every frame
    - Custom Shaders
    - One Mesh/Material per
  - BRG
    - Upload once
    - Support URP/Lit and HDRP/Lit out of box
    - Muti Mesh/Material support
    - Custom Culling