



解密

GPU Resident Drawer

2023



GPU Resident Drawer

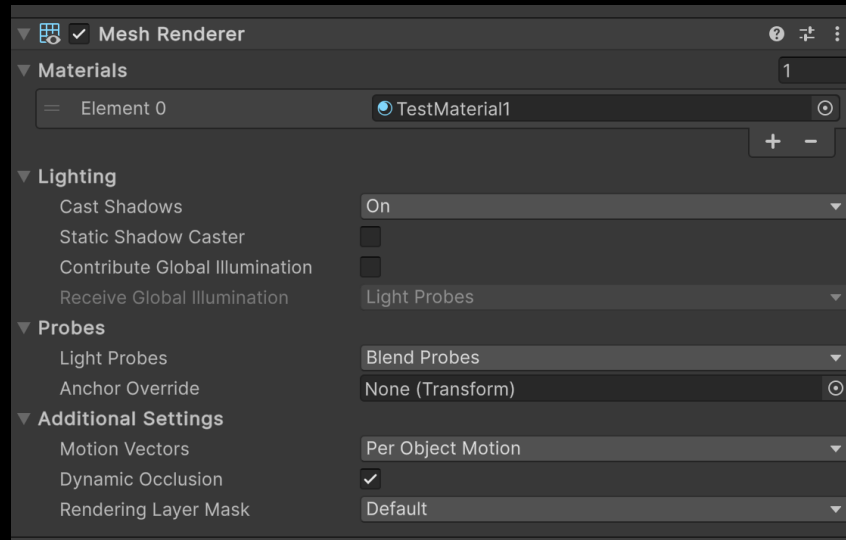
- 什么是GPU Resident Drawer
- GPU Resident Drawer workflow
 - 组织batch
 - 裁剪
 - 输出DrawCommands



什么是GPU Resident Drawer

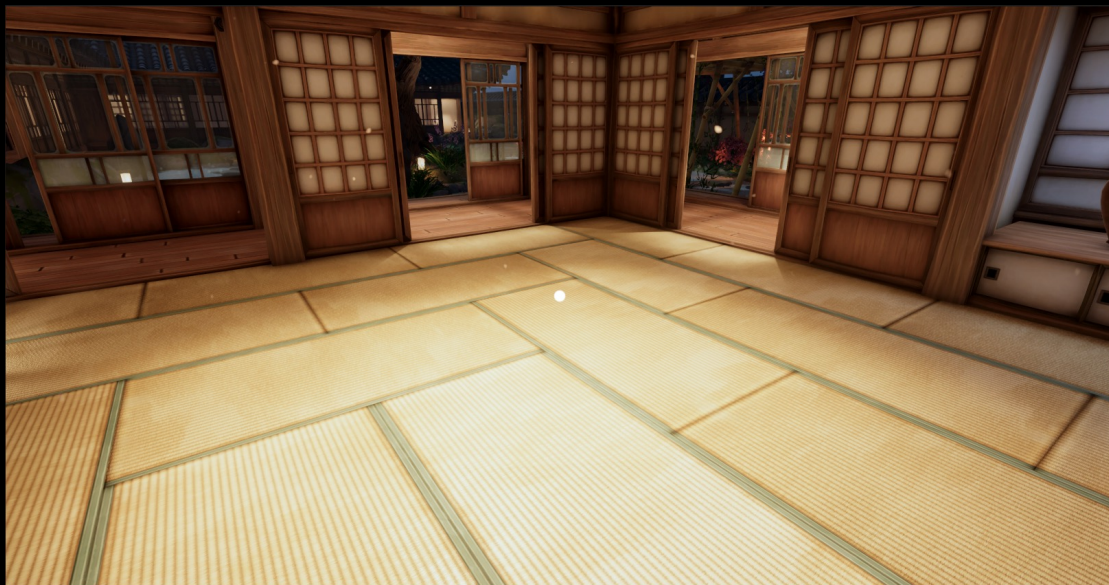
什么是GPU Resident Drawer

- 基于Batch Renderer Group
- 将MeshRenderer数据转换成BRG batch数据



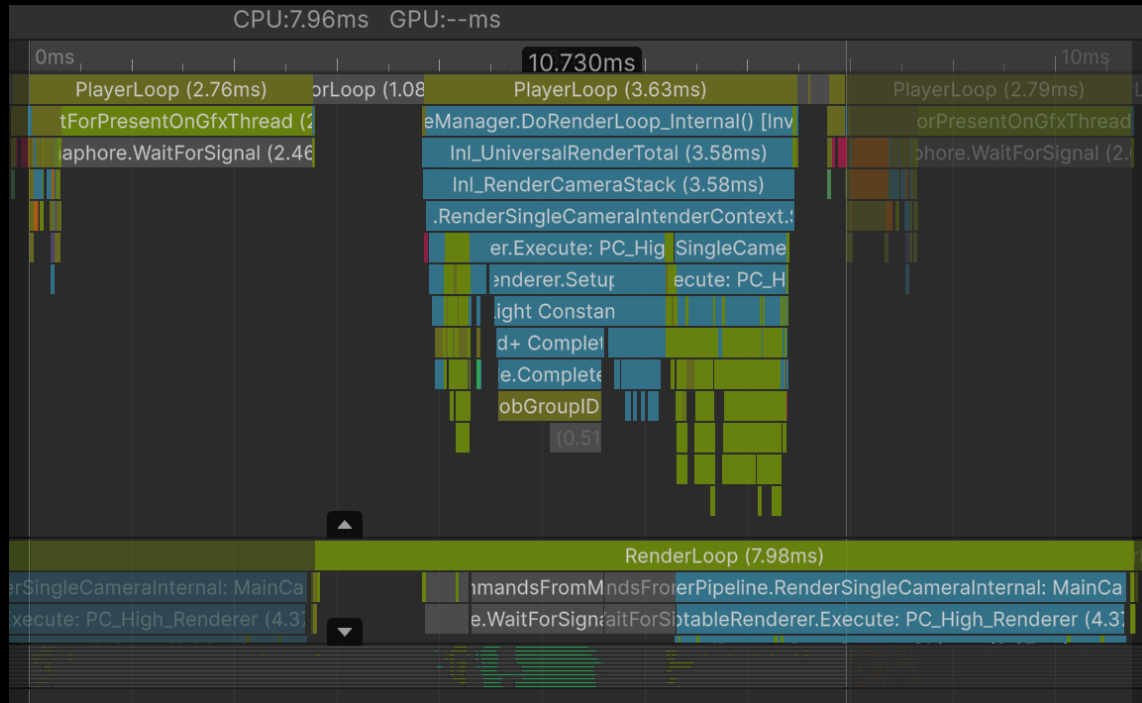
什么是GPU Resident Drawer

- 相比传统SRP好处
- Persistent batch
- GPU persistent data model



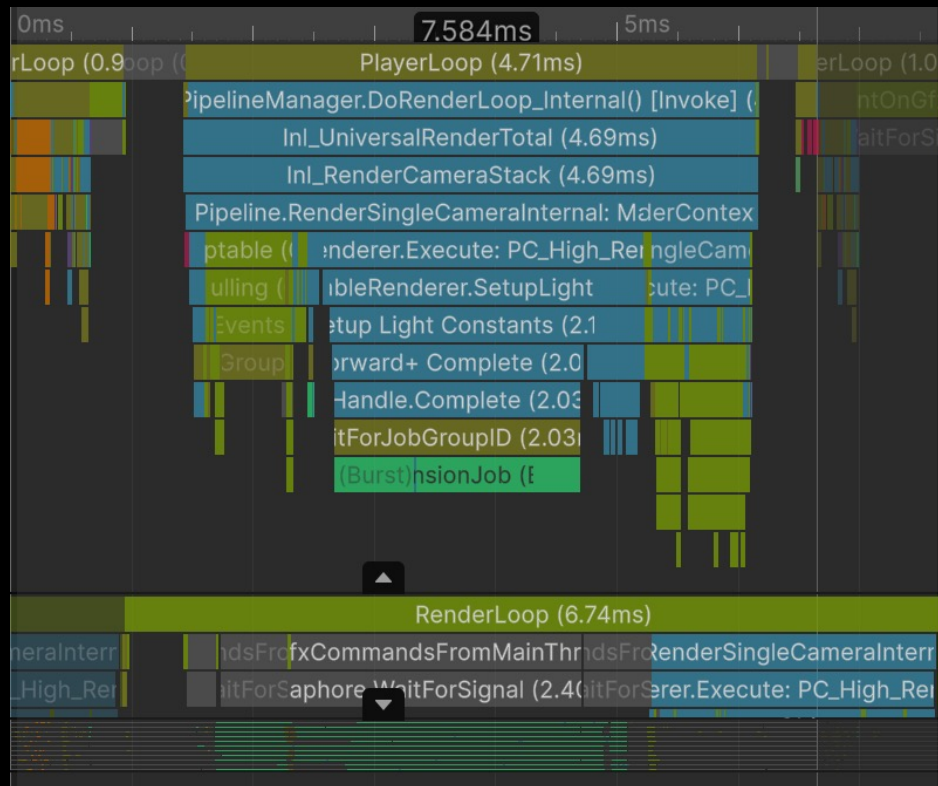
什么是GPU Resident Drawer

- URP
- 10.7ms



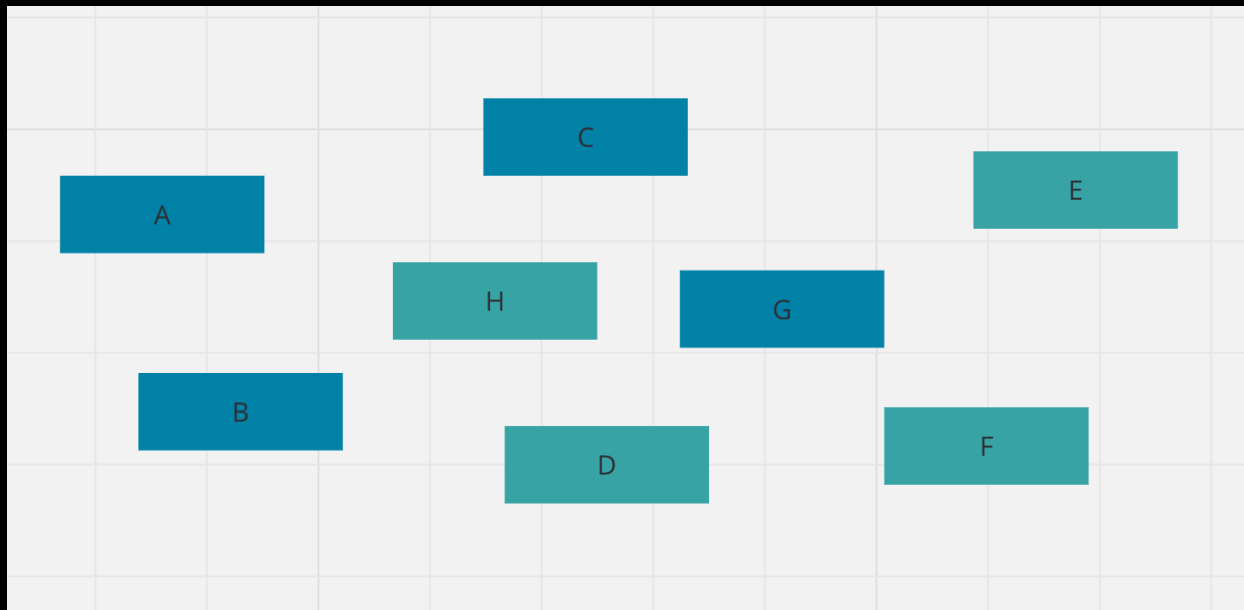
什么是GPU Resident Drawer

- GPU Resident Drawer
- ~7.5ms
- **1.5x faster**



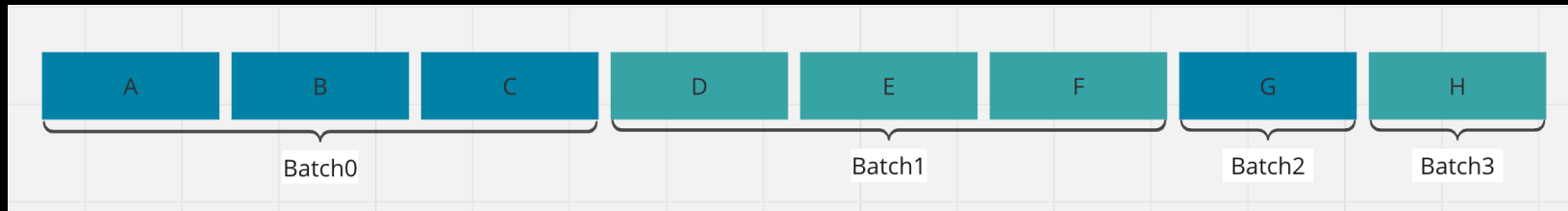
什么是GPU Resident Drawer

→ SRP如何组织数据



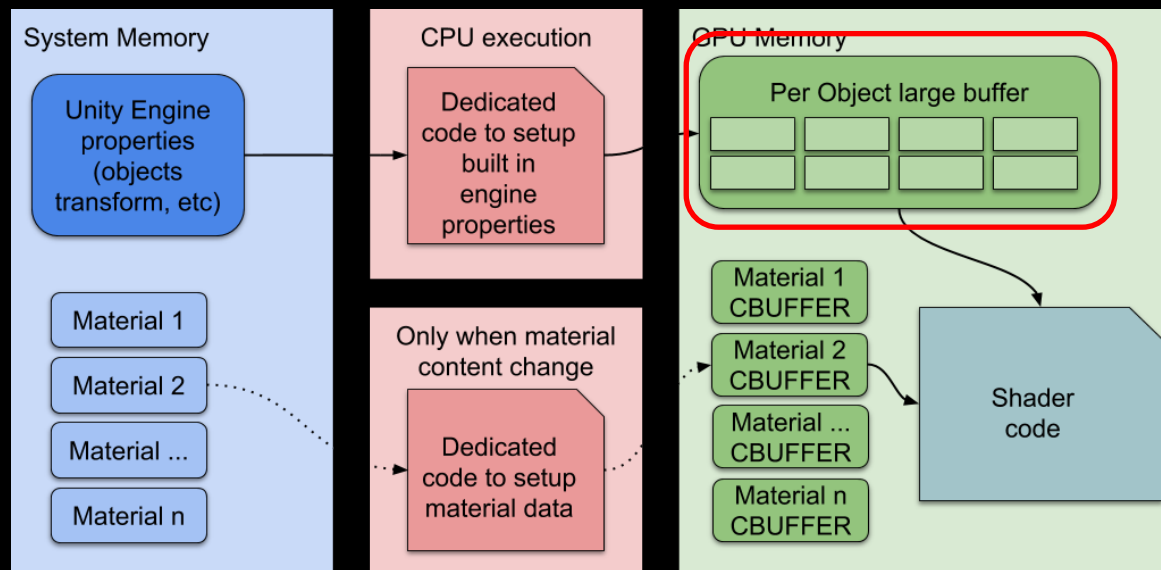
什么是GPU Resident Drawer

→ SRP Batcher如何组织数据



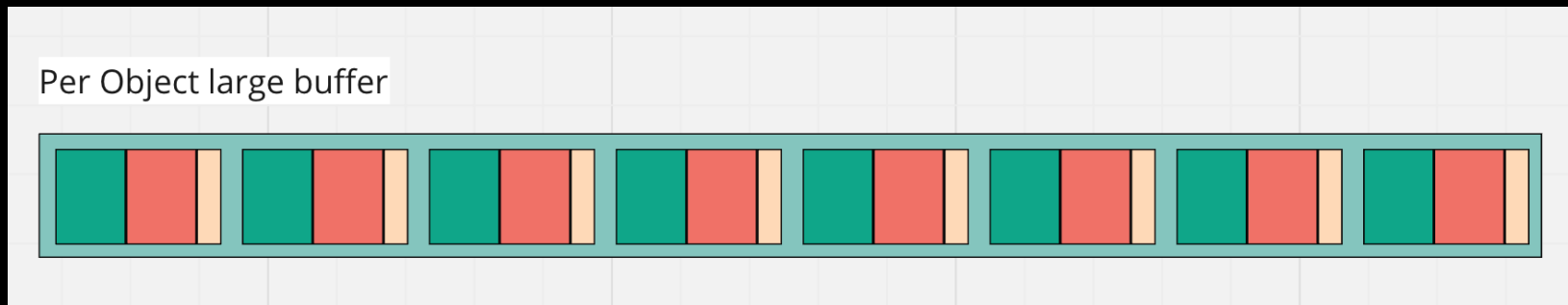
什么是GPU Resident Drawer

→ SRP如何组织数据



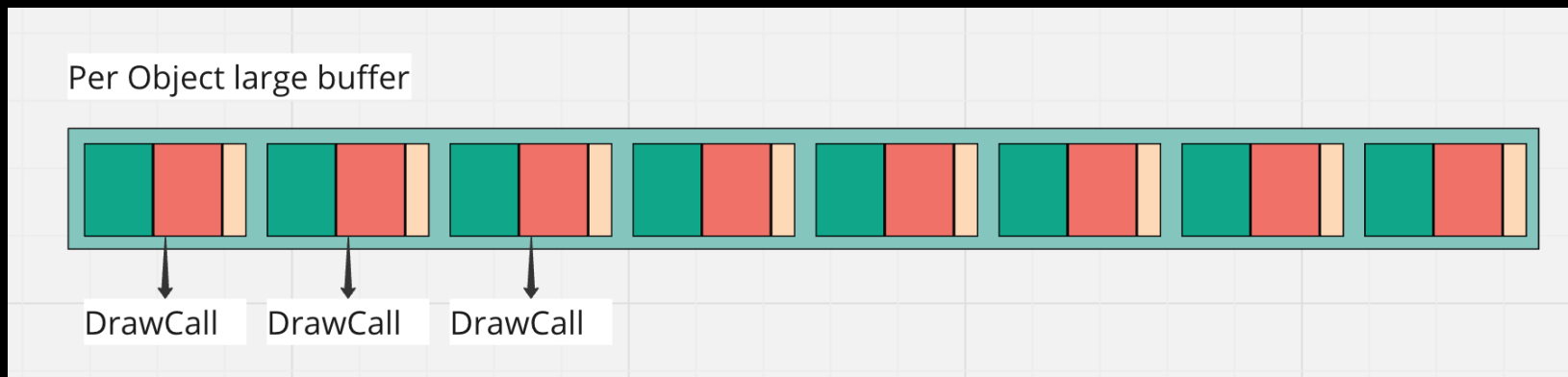
什么是GPU Resident Drawer

→ SRP如何组织数据



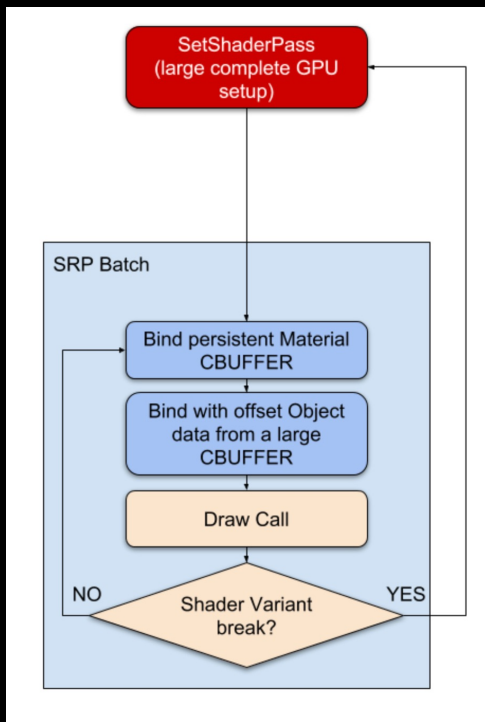
什么是GPU Resident Drawer

→ SRP如何组织数据



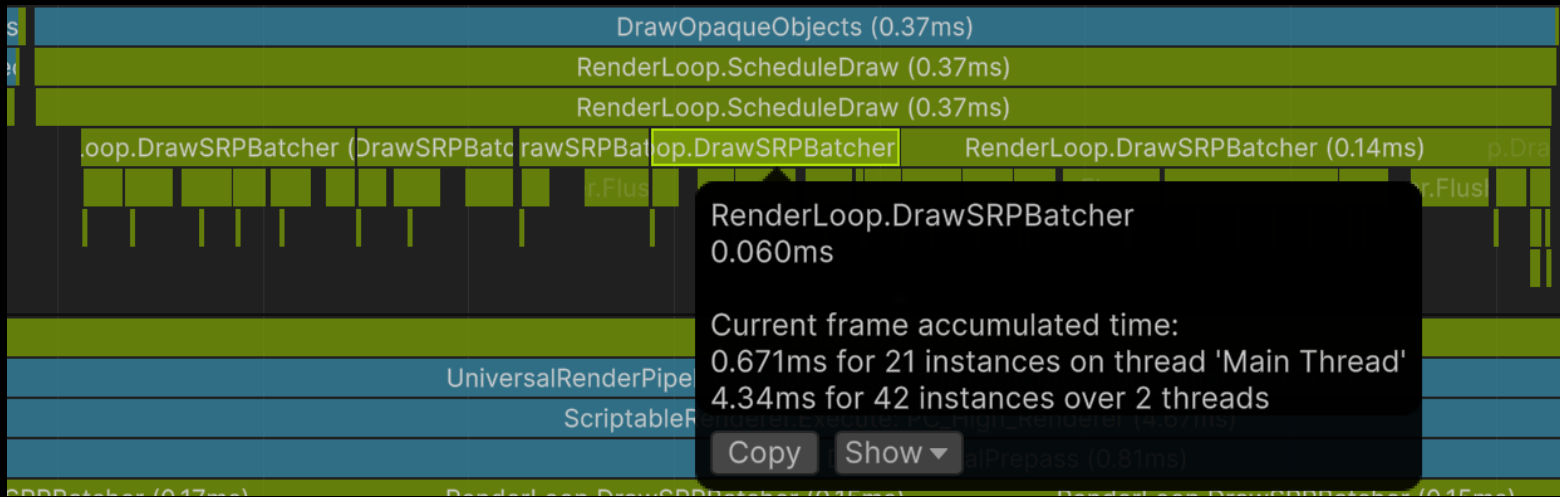
什么是GPU Resident Drawer

→ SRP如何组织数据



什么是GPU Resident Drawer

→ SRP Batcher单线程执行



什么是GPU Resident Drawer

- SRP Batcher问题
- Batch根据传入的Renderer的顺序确定Batch
- 单线程



GPU Resident Drawer

GPU Resident Drawer

→ FrameDebugger

The screenshot displays the Unity Frame Debugger's GPU Resident Drawer. The left pane shows a tree view of rendering stages, with 'RenderLoop.DrawSRPBatcher' expanded to show 21 sub-items, including multiple 'Hybrid Batch Group' entries. The right pane shows detailed statistics for the selected batch group, with a red box highlighting 'DrawInstanced Calls: 13', 'Instances: 233', and 'Draw Calls: -'. Below these are statistics for vertices (23856) and indices (73896). The 'Batch cause' is identified as 'SRP: Node have different shaders.' and the 'Meshes' list includes Stone_03_LOD2, Stone_04_LOD0, Stone_01_LOD2, Stone_01_LOD1, Stone_02_LOD2, Stone_04_LOD1, Stone_04_LOD2, and Stone_02_LOD1. The 'LightMode' is 'UniversalForward' and the 'Pass' is 'Universal Forward (0)'. The 'Used Shader' and 'Original Shader' are both 'Shader Graphs/MossRock_Graph'.

Item	Count
Blend Alpha	One / Zero
BlendOp Color	Add
BlendOp Alpha	Add
DrawInstanced Calls	13
Instances	233
Draw Calls	-
Vertices	23856
Indices	73896

Batch cause
SRP: Node have different shaders.

Meshes

- Stone_03_LOD2
- Stone_04_LOD0
- Stone_01_LOD2
- Stone_01_LOD1
- Stone_02_LOD2
- Stone_04_LOD1
- Stone_04_LOD2
- Stone_02_LOD1

LightMode: UniversalForward
Pass: Universal Forward (0)
Used Shader: Shader Graphs/MossRock_Graph
Original Shader: Shader Graphs/MossRock_Graph

GPU Resident Drawer

→ 基于Instancing技术

```
public static void DrawMeshInstanced(  
    Mesh mesh,  
    int submeshIndex,  
    Material material,  
    List<Matrix4x4> matrices,  
    MaterialPropertyBlock properties,  
    ShadowCastingMode castShadows,  
    bool receiveShadows,  
    int layer,  
    Camera camera,  
    LightProbeUsage lightProbeUsage,  
    LightProbeProxyVolume lightProbeProxyVolume) {...}
```

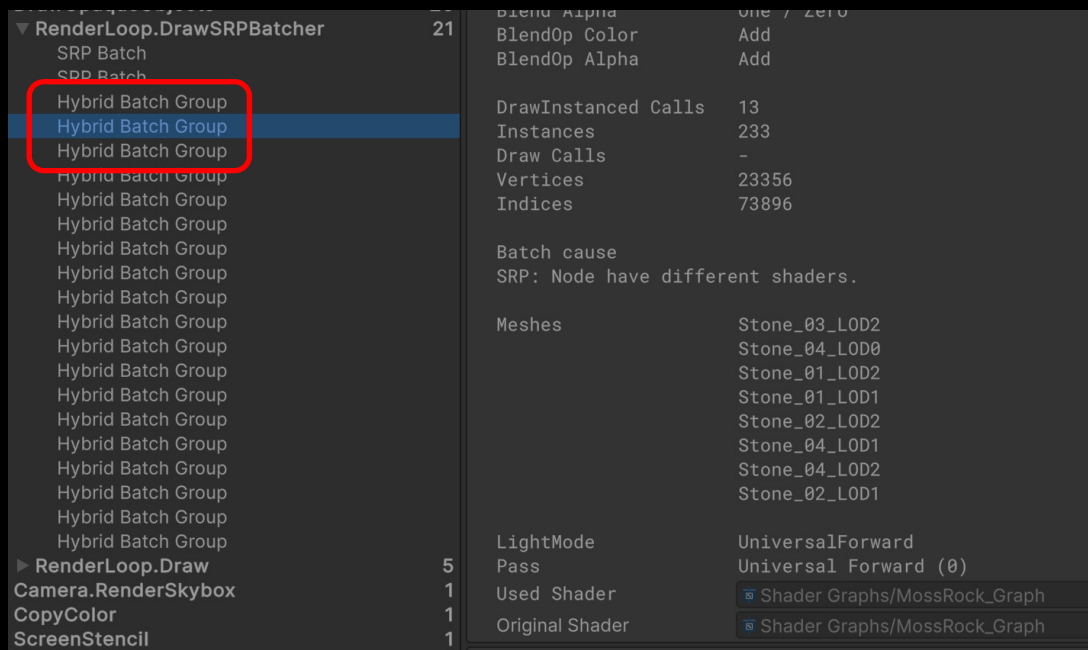
GPU Resident Drawer

→ 基于Instancing技术

```
public static void DrawMeshInstanced(  
    Mesh mesh,  
    int submeshIndex,  
    Material material,  
    List<Matrix4x4> matrices,  
    MaterialPropertyBlock properties,  
    ShadowCastingMode castShadows,  
    bool receiveShadows,  
    int layer,  
    Camera camera,  
    LightProbeUsage lightProbeUsage,  
    LightProbeProxyVolume lightProbeProxyVolume){...}
```

GPU Resident Drawer

→ 使用了Batch Renderer Group API



GPU Resident Drawer

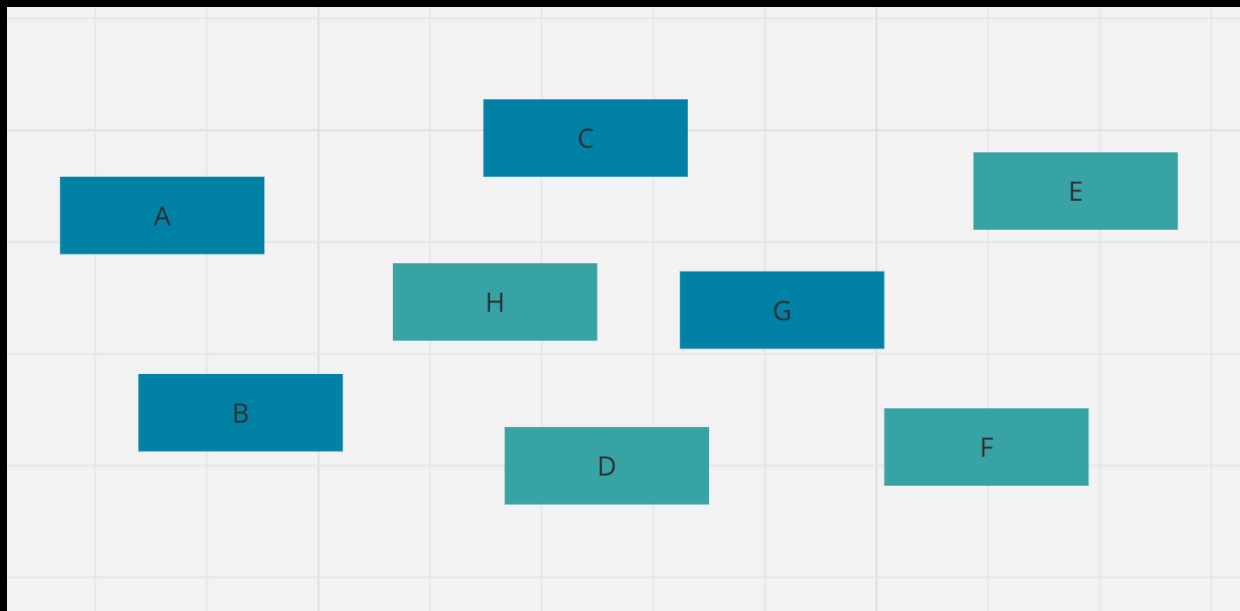
→ GPU Resident Drawer是Batch Renderer Group API的上层封装

→ 做了下面几项工作

1. 把Mesh Renderer的数据组织成BRG Batch
2. 上传Renderer数据到GPU
3. 对Renderer进行裁剪
4. 提交draw call

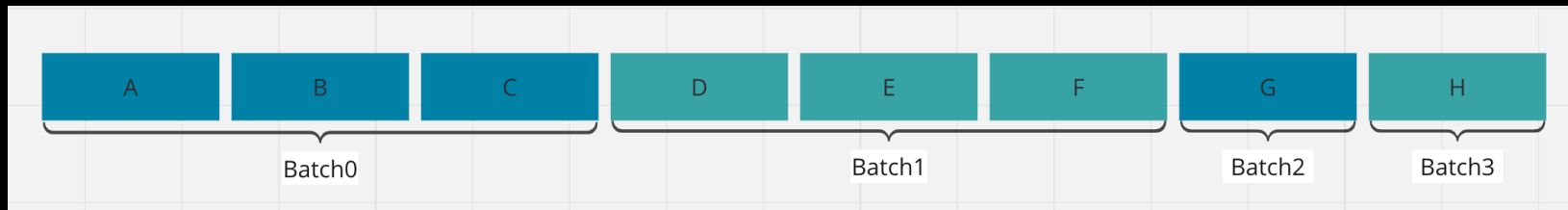
GPU Resident Drawer

- 组织Batch
- SRP Batcher VS Resident Drawer



什么是GPU Resident Drawer

- SRP Batcher
- 按照Renderer顺序尝试合批



什么是GPU Resident Drawer

- Resident Drawer
- 对Renderer进行重映射 (remapping) , 获得更少的批次

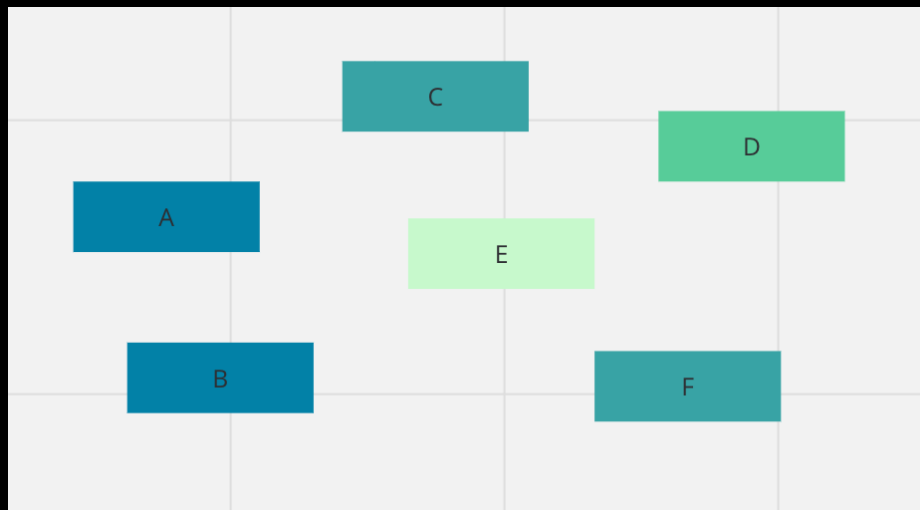


GPU Resident Drawer

→ Resident Drawer如何做到更合理的组织Batch?

GPU Resident Drawer

→ Resident Drawer如何做到更合理的组织Batch?



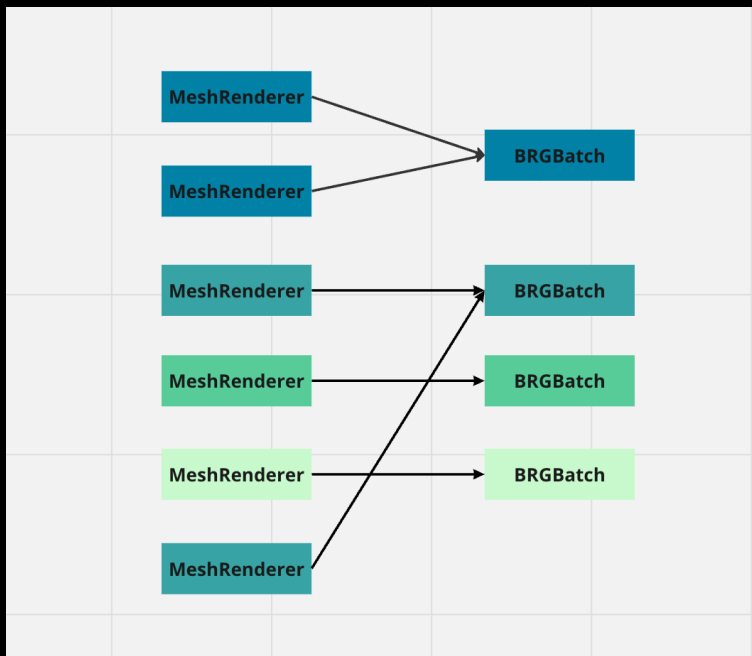
GPU Resident Drawer

- Instancing技术限制
- 一次instanced call, 需要传入一个mesh和一个material

```
public static void DrawMeshInstanced(  
    Mesh mesh,  
    int submeshIndex,  
    Material material,  
    List<Matrix4x4> matrices,  
    MaterialPropertyBlock properties,  
    ShadowCastingMode castShadows,  
    bool receiveShadows,  
    int layer,  
    Camera camera,  
    LightProbeUsage lightProbeUsage,  
    LightProbeProxyVolume lightProbeProxyVolume) {...}
```

GPU Resident Drawer

→ Resident Drawer如何做到更合理的组织Batch?



GPU Resident Drawer

→ Resident Drawer如何做到更合理的组织Batch?

```
public static void DrawMeshInstanced(  
    Mesh mesh,  
    int submeshIndex,  
    Material material,  
    List<Matrix4x4> matrices,  
    MaterialPropertyBlock properties,  
    ShadowCastingMode castShadows,  
    bool receiveShadows,  
    int layer,  
    Camera camera,  
    LightProbeUsage lightProbeUsage,  
    LightProbeProxyVolume lightProbeProxyVolume) { ... }
```

GPU Resident Drawer

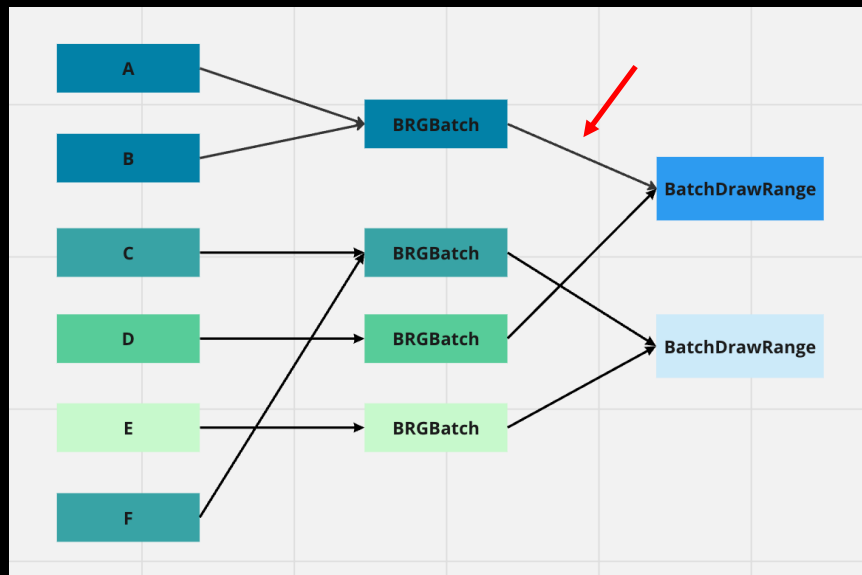
- DrawRange
- 一组连续的拥有共同过滤属性的Batch
- 类似SRP->FilteringSettings

```
public struct BatchDrawRange
{
    <> IL code
    public BatchDrawCommandType drawCommandsType;
    <> IL code
    public uint drawCommandsBegin;
    <> IL code
    public uint drawCommandsCount;
    <> IL code
    public BatchFilterSettings filterSettings;
}
```

```
var range = new BatchDrawRange();
range.filterSettings.shadowCastingMode = ShadowCastingMode.Off;
```

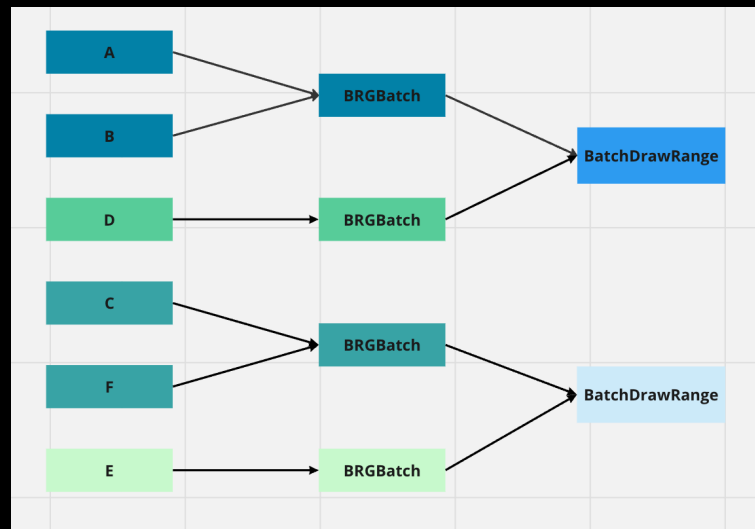
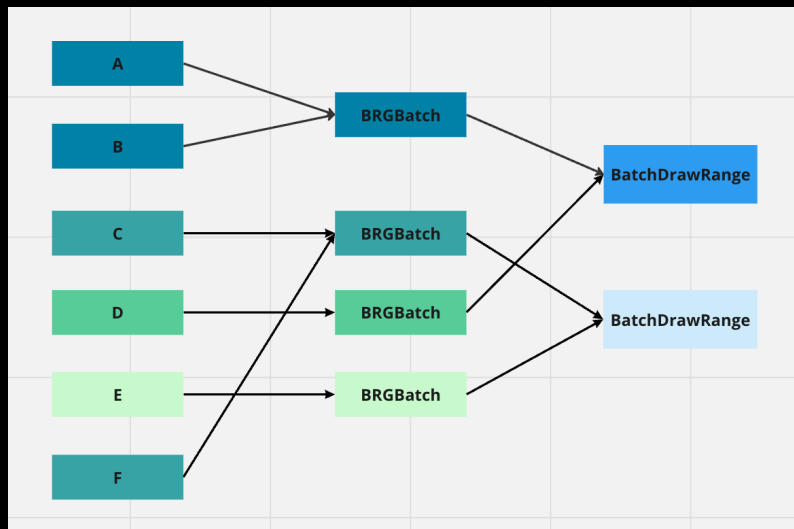
GPU Resident Drawer

→ 按照DrawRange进行分组



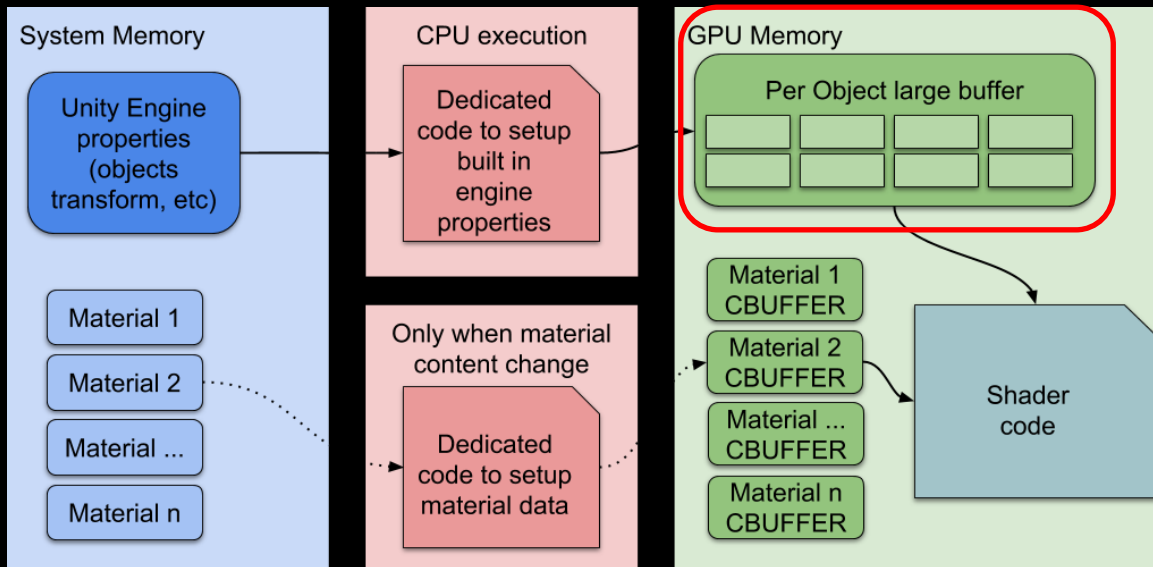
GPU Resident Drawer

→ 重映射Remap



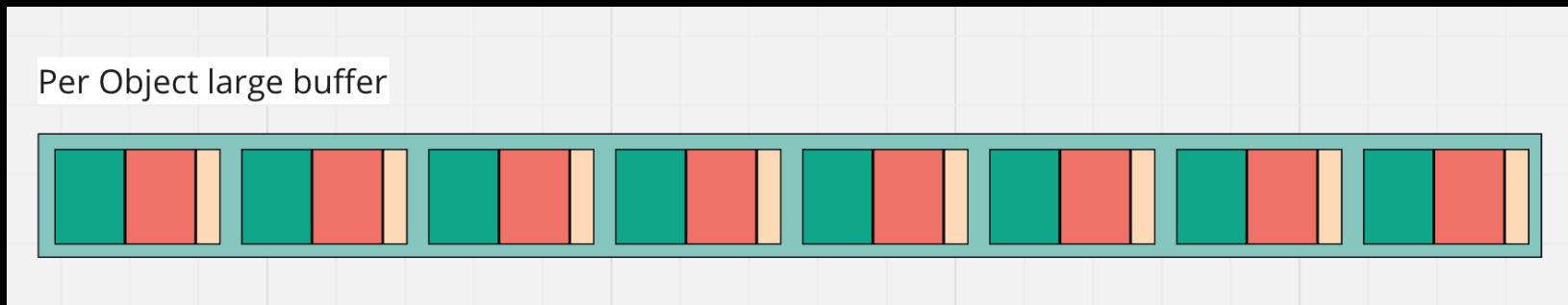
GPU Resident Drawer

→ 上传Renderer数据



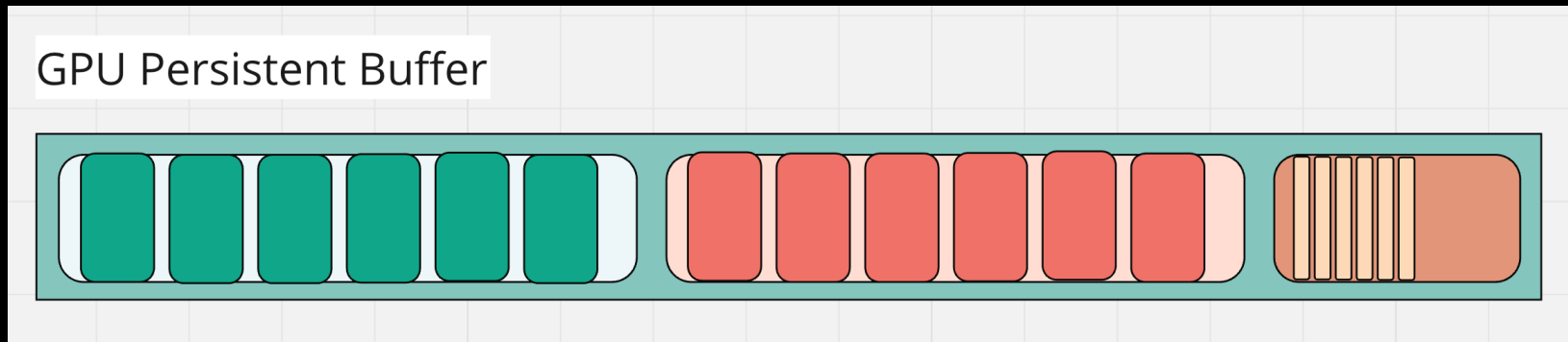
GPU Resident Drawer

→ SRP Batcher



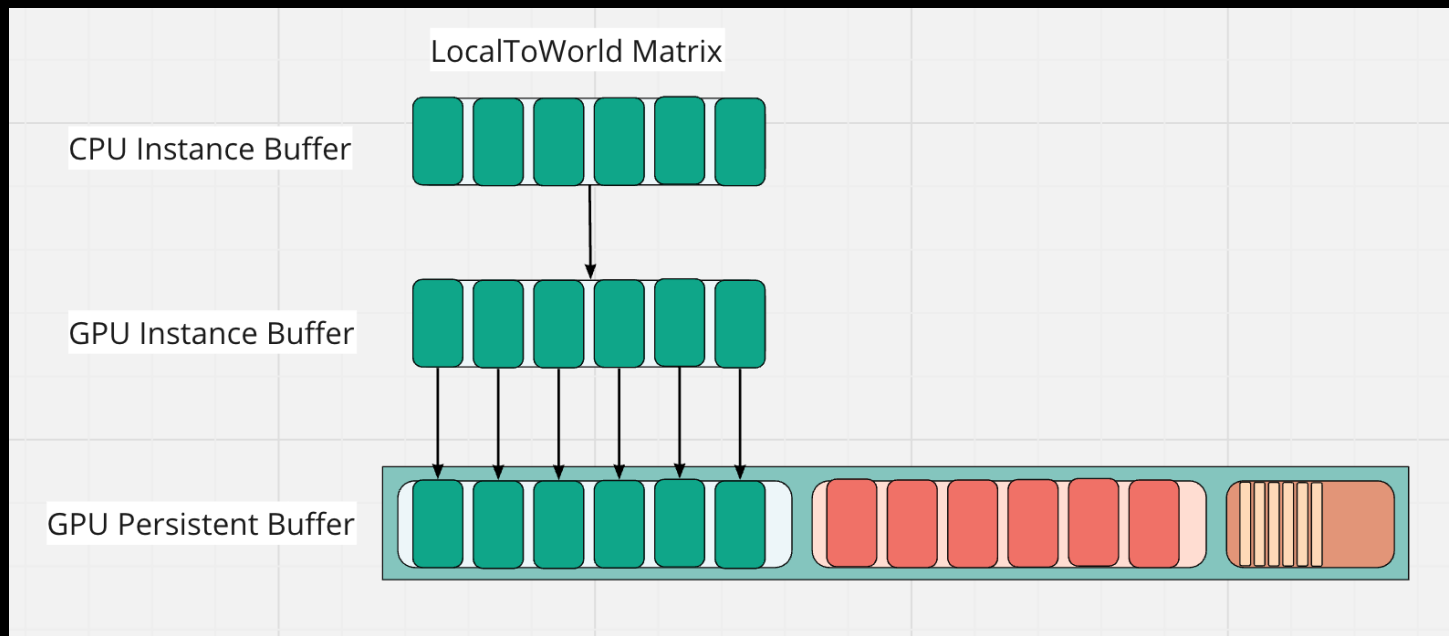
GPU Resident Drawer

- Resident Drawer
- SOA



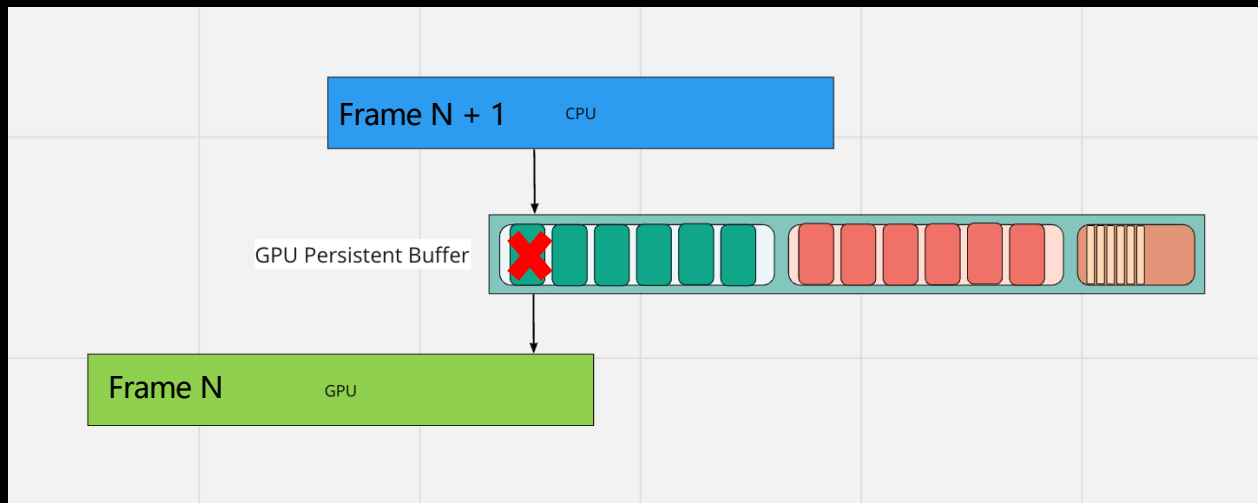
GPU Resident Drawer

- 上传数据
- 采用compute shader并行拷贝数据



GPU Resident Drawer

- 类似双缓冲设计
- 为什么?

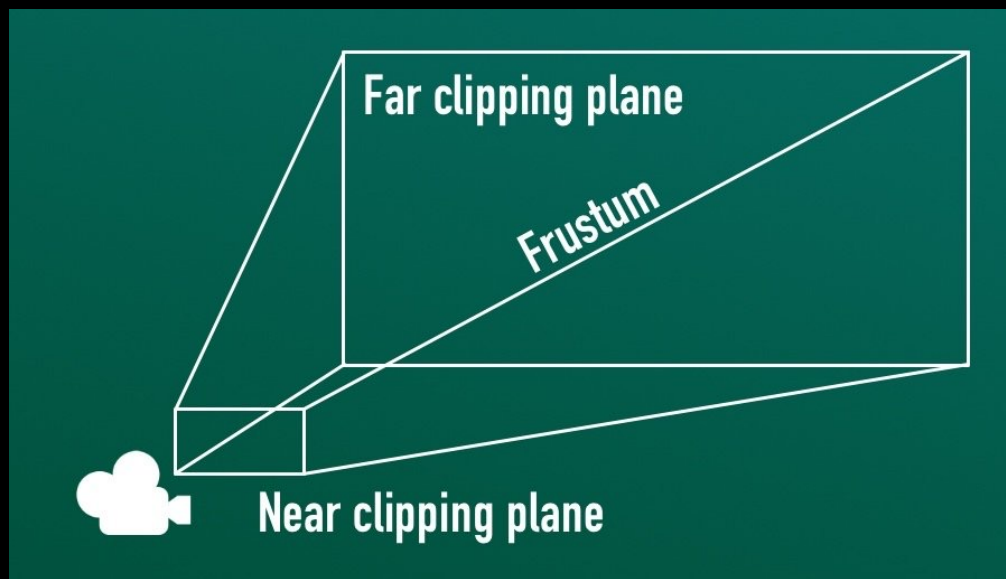


GPU Resident Drawer

- `DrawMeshInstanced()`
- 不支持裁剪，会渲染相机视口外的物体

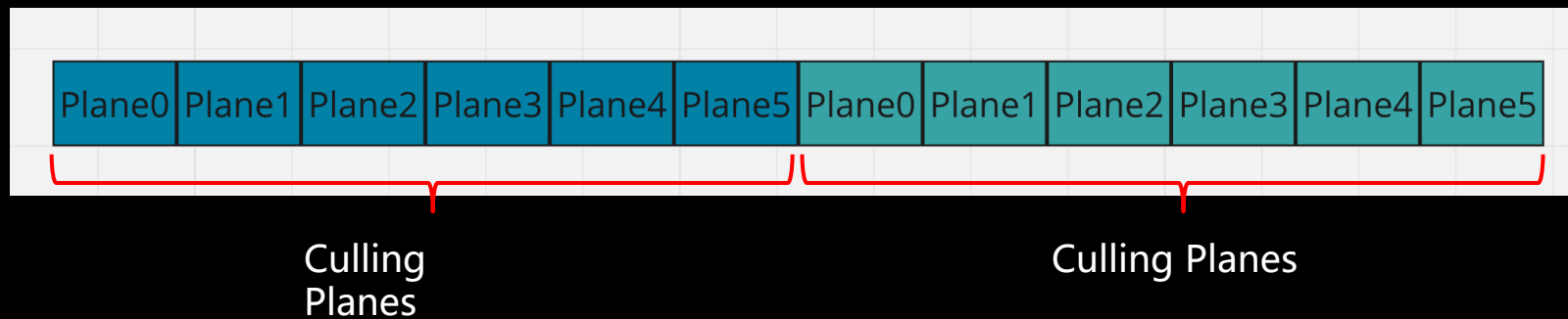
GPU Resident Drawer

→ 裁剪



GPU Resident Drawer

→ 裁剪



GPU Resident Drawer

裁剪会面临什么问题？

1. 一帧裁剪不只发生一次
2. 每个相机+每个灯光各裁剪一次
3. 当场景中绘制物体增多时，裁剪会成为性能瓶颈

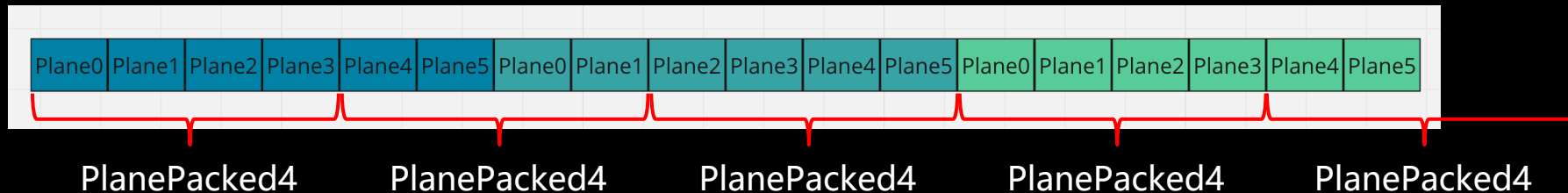
GPU Resident Drawer

如何解决？

1. 多线程
2. SIMD!

GPU Resident Drawer

→ 裁剪



GPU Resident Drawer

[7 usages](#)

```
public struct PlanePacket4
{
    public float4 Xs;   ✎ Serializable
    public float4 Ys;   ✎ Serializable
    public float4 Zs;   ✎ Serializable
    public float4 Distances; ✎ Serializable
}
```

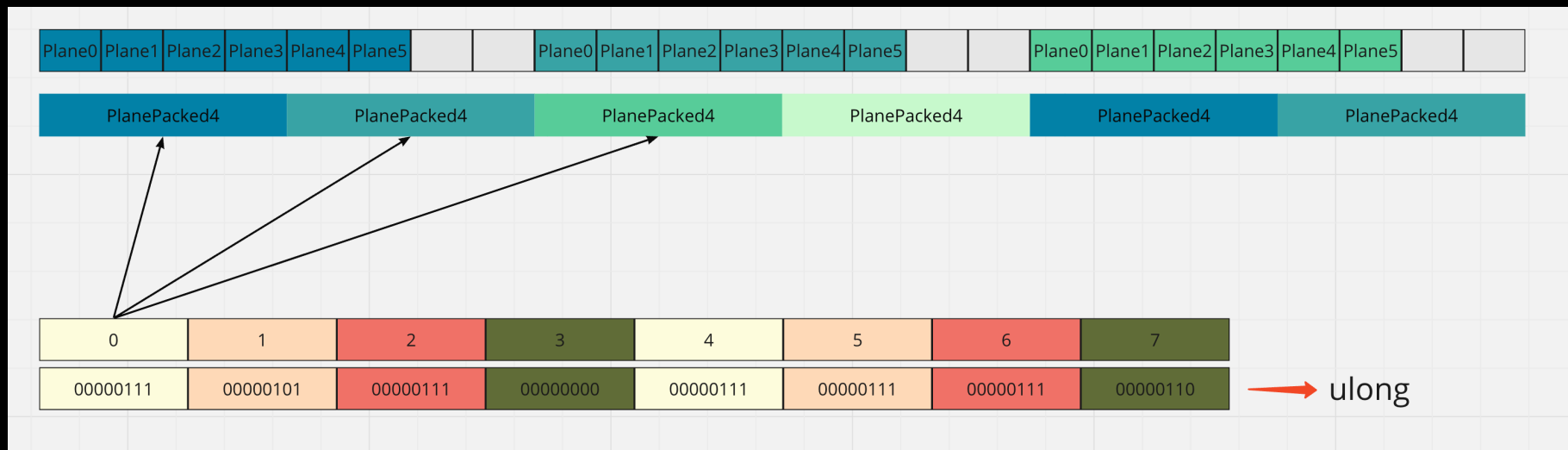
GPU Resident Drawer

→ SIMD!

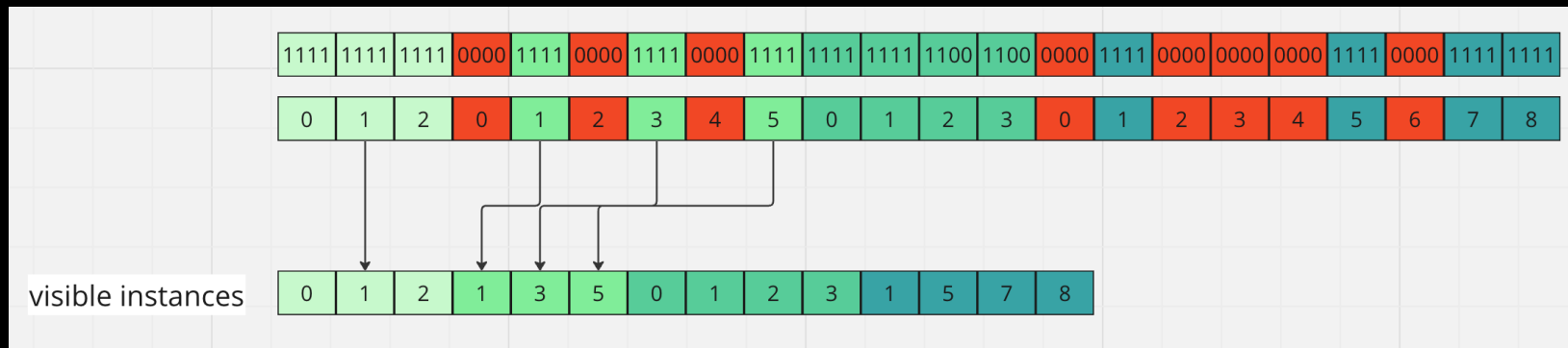
```
int4 masks = (int4) 0;
for (int i = 0; i < packetCount; i++)
{
    var p:PlanePacket4 = cullingPlanePackets[offset + i];
    float4 distances = dot4(p.Xs, p.Ys, p.Zs, mx, my, mz) + p.Distances;
    float4 radii = dot4(xs: ex, ys: ey, zs: ez, mx: math.abs(p.Xs), my: math.abs(p.Ys), mz: math.abs(p.Zs));
    masks += (int4)(distances + radii <= 0);
}

int outCount = math.csum(masks);
if (outCount == 0) outMask |= 1u << s;
```

GPU Resident Drawer



GPU Resident Drawer

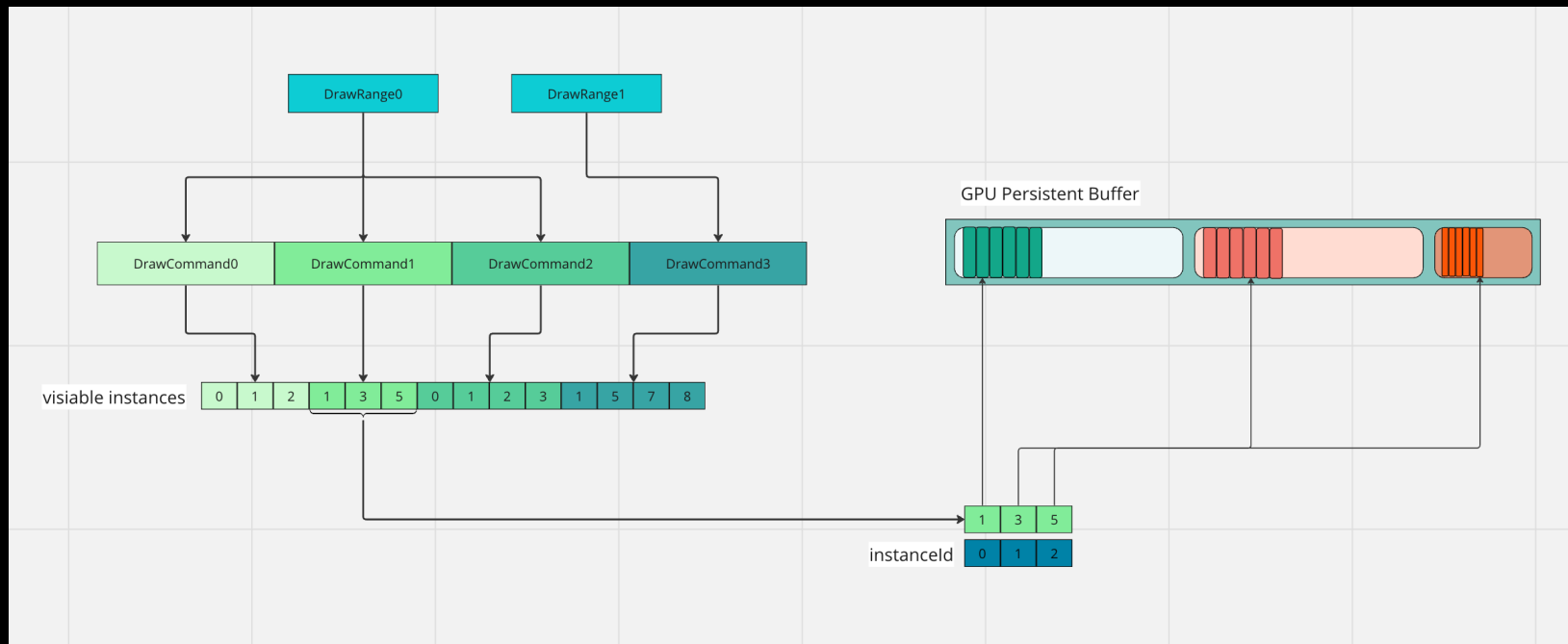


GPU Resident Drawer

到此为止

1. Batch组织 ✓
2. Renderer数据上传GPU ✓
3. Batch内的instance可见性 ✓

GPU Resident Drawer



GPU Resident Drawer

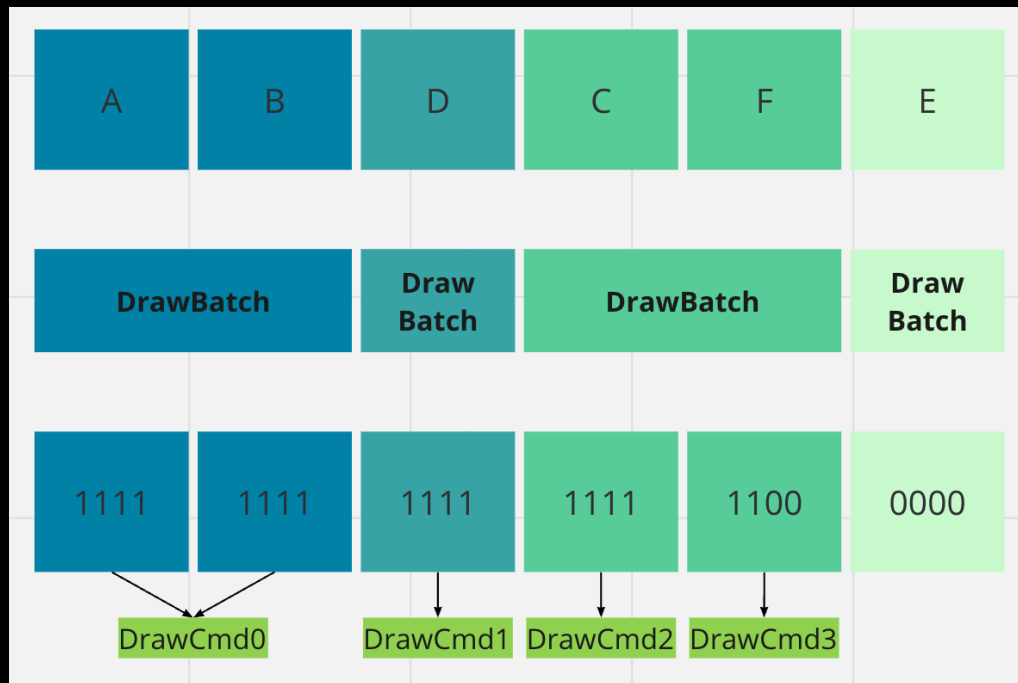
→ 提交DrawCall

```
public struct BatchDrawCommand
{
    public BatchID batchID;
    public BatchMaterialID materialID;
    public BatchMeshID meshID;
    public uint visibleOffset;
    public uint visibleCount;
}
```

```
public struct BatchDrawRange
{
    public BatchDrawCommandType drawCommandsType;
    public uint drawCommandsBegin;
    public uint drawCommandsCount;
    public BatchFilterSettings filterSettings;
}
```

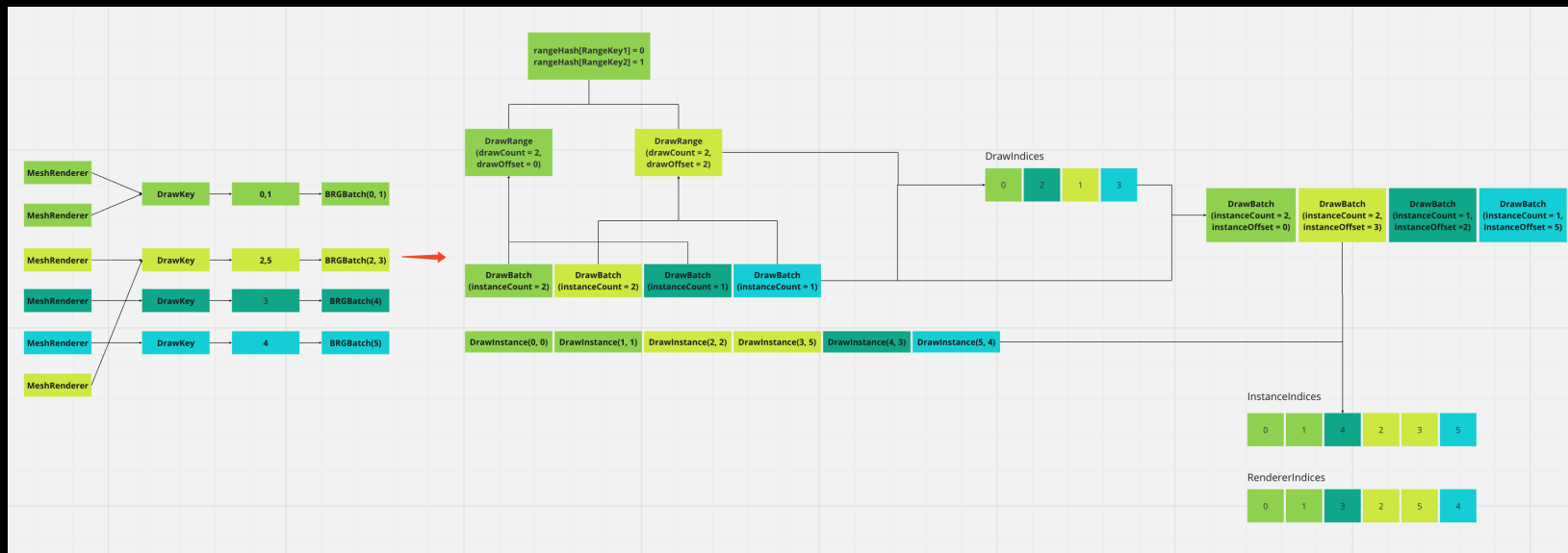

GPU Resident Drawer

→ 提交DrawCall



GPU Resident Drawer

→ 组织batch



GPU Resident Drawer

→ 提交DrawCall



GPU Resident Drawer

- 总结
- 基于BatchRendererGroup API
- 把MeshRenderer转换成BRG batch
- 相比SRP Batcher可以提供更好的合批
- 并且优化了CPU->GPU的数据传输过程

GPU Resident Drawer

- Unity GPU Driven Tier0
- 主要优化批次组织和CPU->GPU的数据传输
- Unity6 BRG扩展了indirect和procedural两种新DrawCommand
- 当前Unity6 alpha版本支持GPU Driven Tier1 => GPU Culling