What's New in Metal

Part 1

Session 604

Aaftab Munshi GPU Software Engineer
James Ding GPU Software Engineer
Jose Enrique D'Arnaude del Castillo GPU Software Engineer
Alp Yucebilgin GPU Software Engineer

Metal at WWDC This Year

A look at the sessions

Adopting Metal

Part One

- Fundamental Concepts
- Basic Drawing
- Lighting and Texturing

Part Two

- Dynamic Data Management
- CPU-GPU Synchronization
- Multithreaded Encoding

Metal at WWDC This Year

A look at the sessions

What's New in Metal

Part One

- Tessellation
- Resource Heaps and Memoryless Render Targets
- Improved Tools

Part Two

- Function Specialization and Function Resource Read-Writes
- Wide Color and Texture Assets
- Additions to Metal Performance Shaders

Metal at WWDC This Year

A look at the sessions

Advanced Shader Optimization

- Shader Performance Fundamentals
- Tuning Shader Code

Agenda

NEW

Agenda

NEW

Function Specialization and
Function Resource Read-Writes

Wide Color and Texture Assets

Additions to Metal Performance Shaders

Agenda

NEW

Function Specialization and
Function Resource Read-Writes

Wide Color and Texture Assets

Additions to Metal Performance Shaders

Agenda

NEW

Function Specialization and
Function Resource Read-Writes

Wide Color and Texture Assets

Additions to Metal Performance Shaders

Agenda

NEW

Tessellation

Resource Heaps and Memoryless Render Targets

Improved Tools

Function Specialization and
Function Resource Read-Writes

Wide Color and Texture Assets

Additions to Metal Performance Shaders

NEW

Tessellation

Aaftab Munshi GPU Software Engineer

Motivation

Motivation

Large increase in memory bandwidth required to render high resolution geometry

Motivation

Large increase in memory bandwidth required to render high resolution geometry

Describe input geometry as a coarser or lower resolution model

Motivation

Large increase in memory bandwidth required to render high resolution geometry

Describe input geometry as a coarser or lower resolution model

Generate high resolution model from the coarser model

- High resolution model not stored in graphics memory
- Method to generate high resolution model is programmable

Tessellation — A technique to amplify and refine the geometric details of an object

Tessellation in Action

GFXBench 4.0

Tessellation in Action

GFXBench 4.0

Tessellation in Action

GFXBench 4.0

Tessellation in Action

GFXBench 4.0

Tessellation in Action

GFXBench 4.0

Tessellation in Action

GFXBench 4.0

Tessellation in Metal

Metal Tessellation

A simpler and efficient approach

Metal Tessellation

A simpler and efficient approach

Modern clean-sheet approach

Easy to use

Efficient and performant

Tessellation in Metal

Availability

macOS Sierra

All Configurations

Tessellation in Metal

Availability

macOS Sierra

All Configurations

iOS

A9 Processor

Tessellation in Metal

Topics we'll cover

Tessellation in Metal

Topics we'll cover

Metal Graphics Pipeline

Rendering Geometry

Adopting Metal Tessellation

Metal Graphics Pipeline

With Tessellation

Tessellation Input Primitives

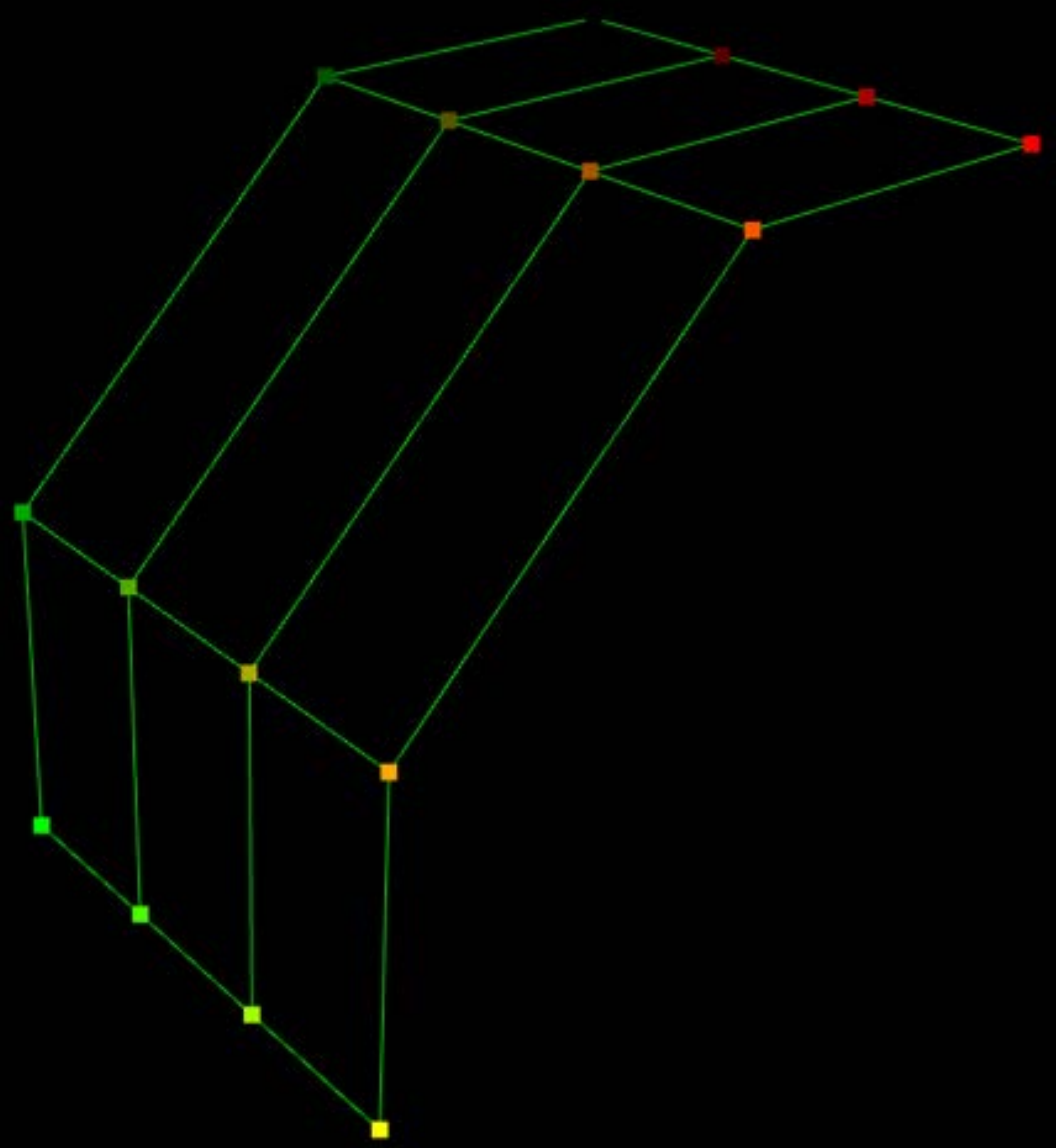
Patches

Tessellation Input Primitives

Patches

A patch is a parametric surface made from spline curves

Described by a set of control vertices



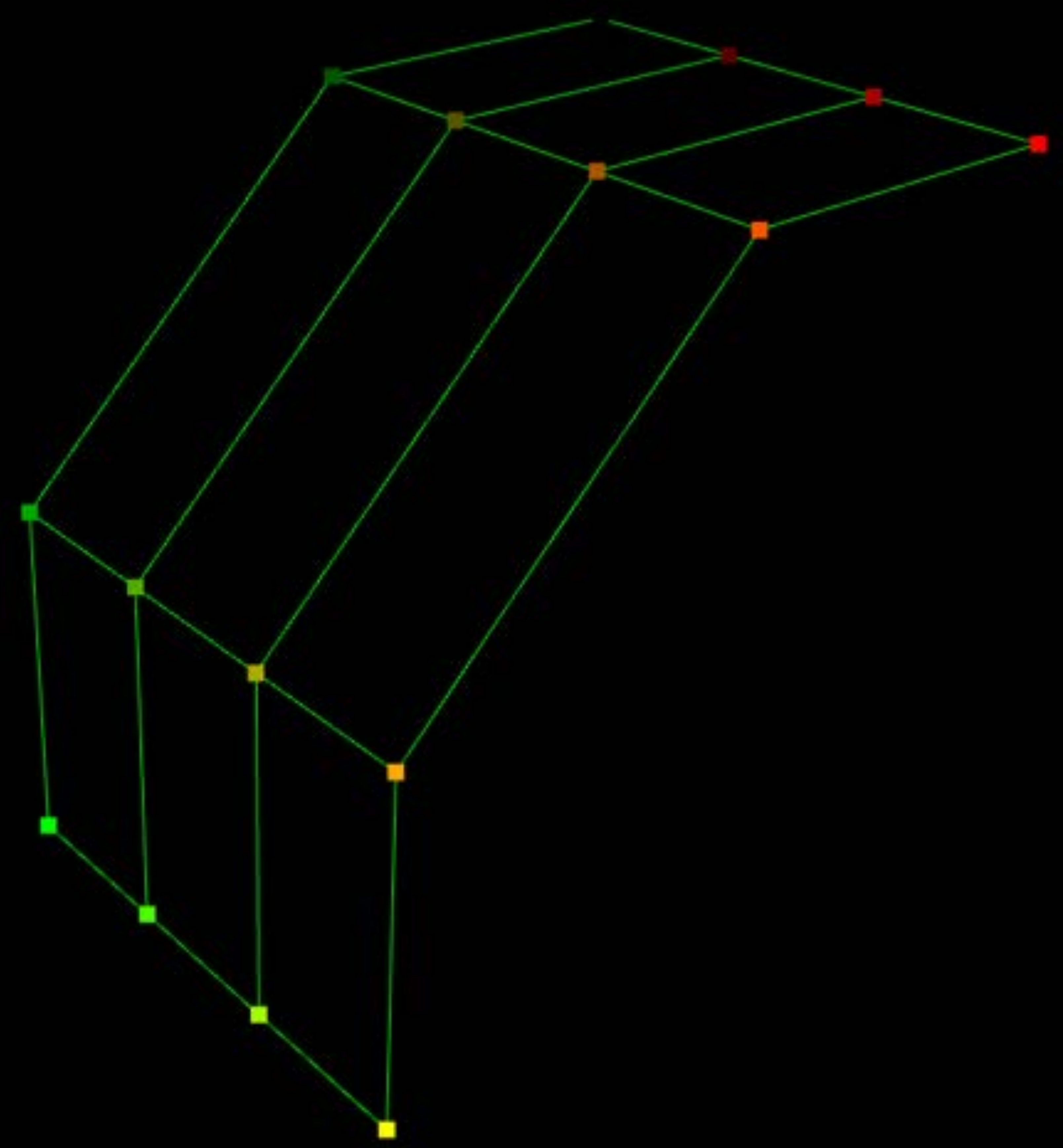
Tessellation Input Primitives

Patches

A patch is a parametric surface made from spline curves

Described by a set of control vertices

Tessellation controls how to render the patch as triangles



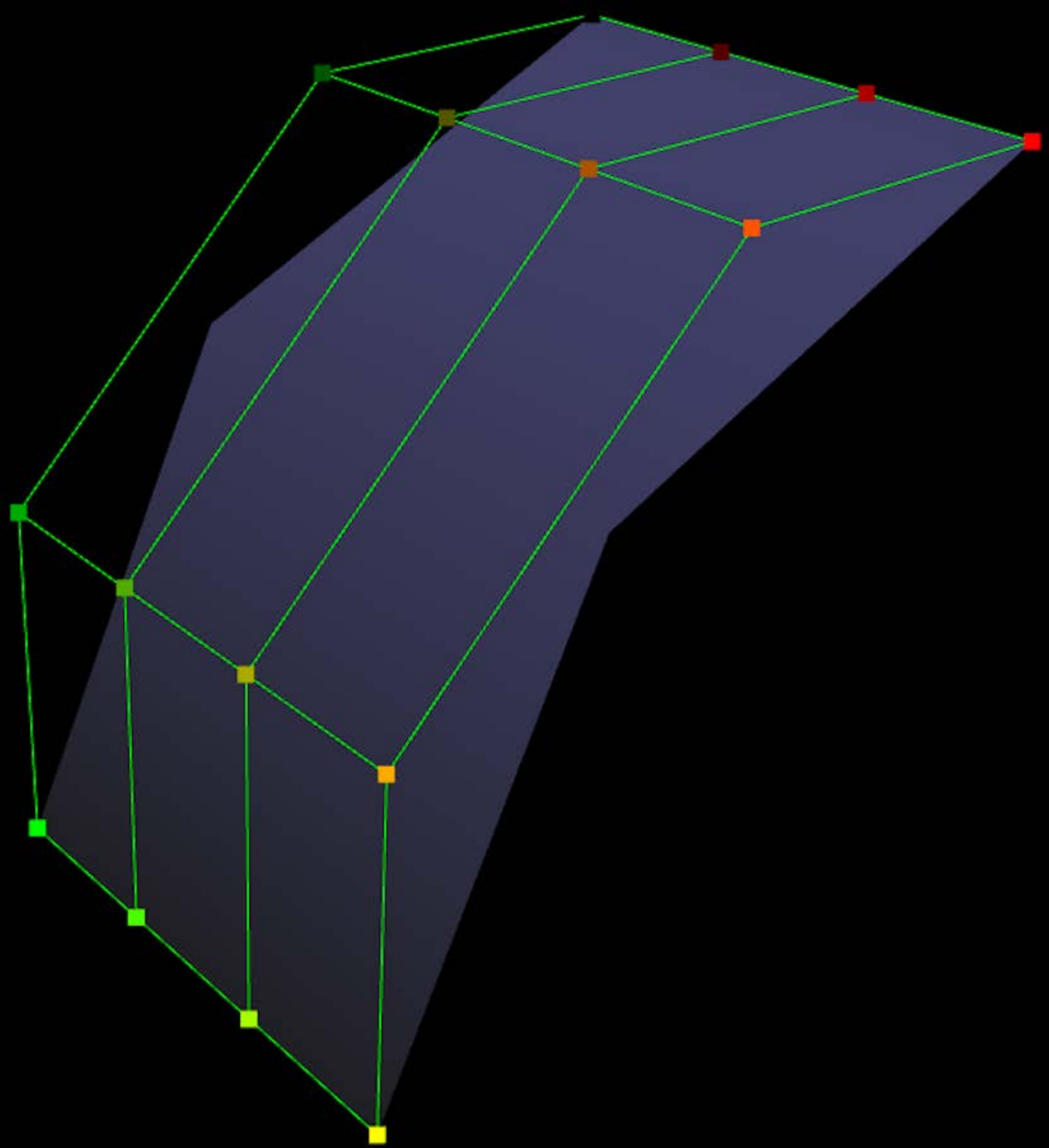
Tessellation Input Primitives

Patches

A patch is a parametric surface made from spline curves

Described by a set of control vertices

Tessellation controls how to render the patch as triangles



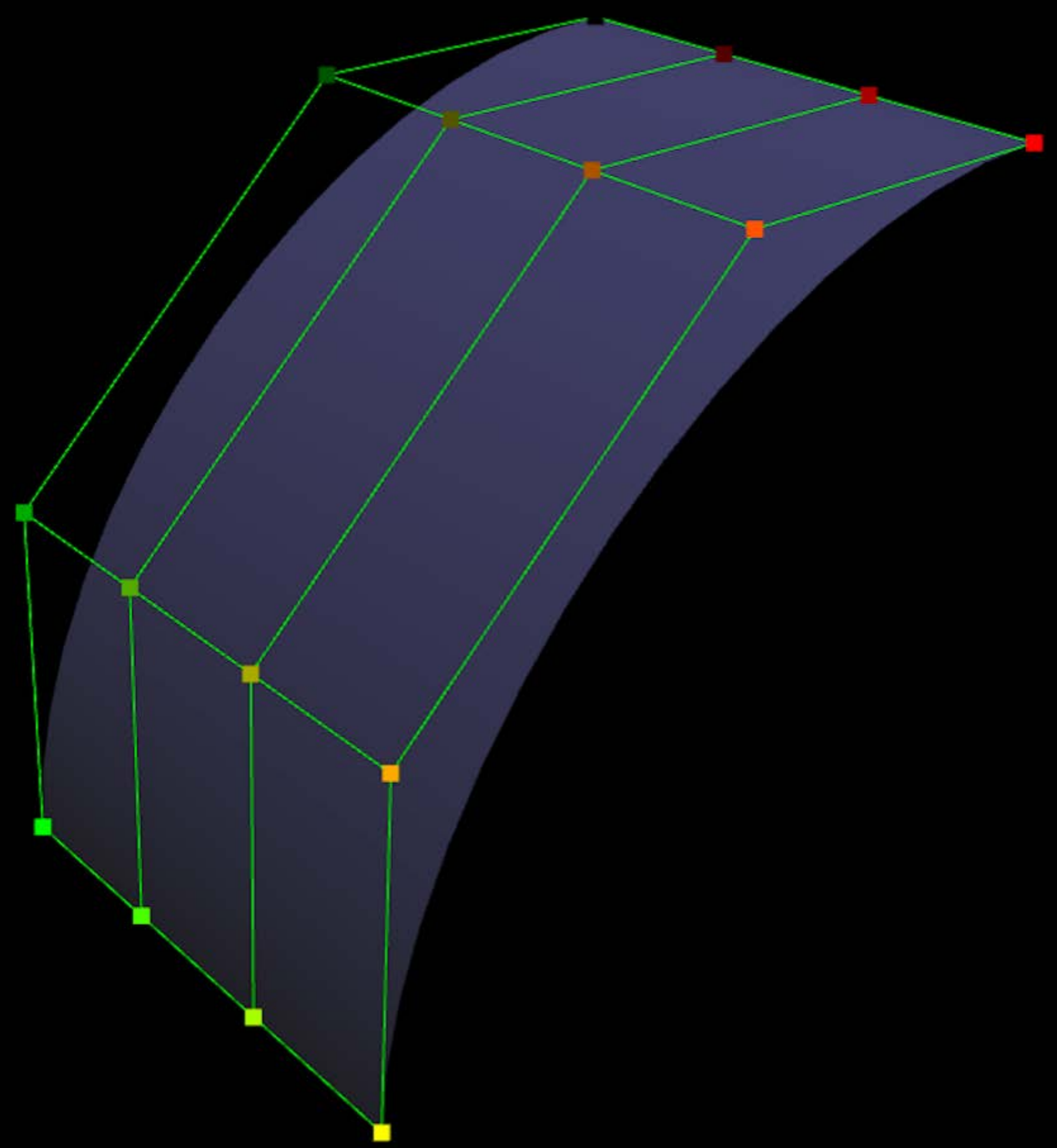
Tessellation Input Primitives

Patches

A patch is a parametric surface made from spline curves

Described by a set of control vertices

Tessellation controls how to render the patch as triangles



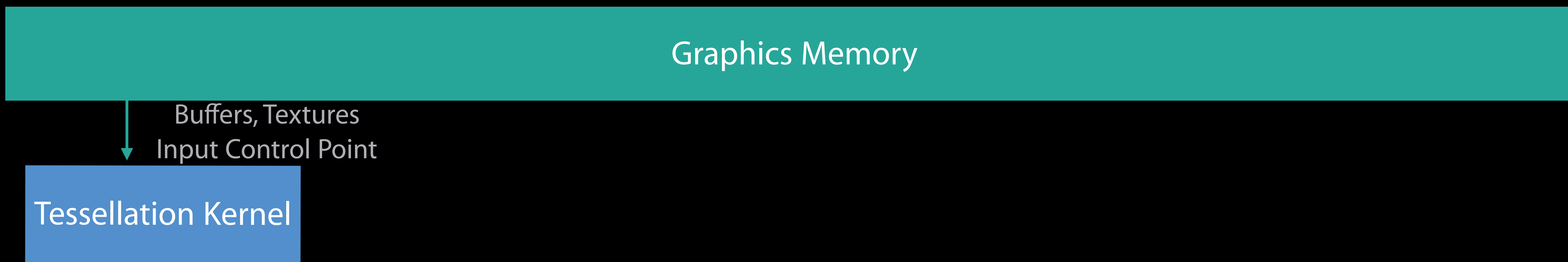
Metal Tessellation

First stage — Tessellation kernel

Graphics Memory

Metal Tessellation

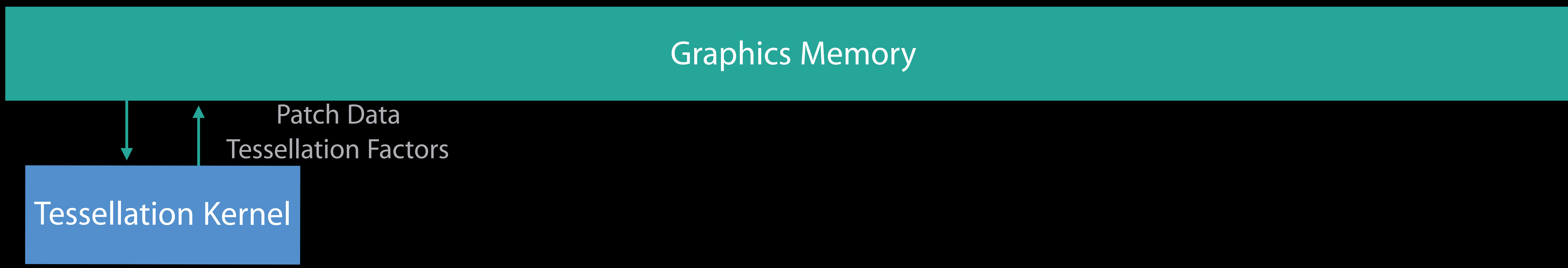
First stage — Tessellation kernel



A programmable stage

Metal Tessellation

First stage — Tessellation kernel



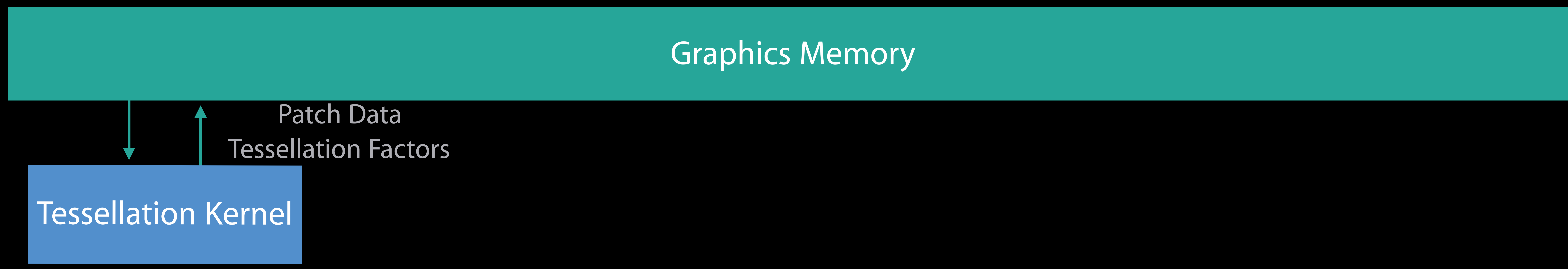
A programmable stage

Computations to generate

- Tessellation factors for a patch — determines how much to subdivide the patch
- Additional patch data, if required

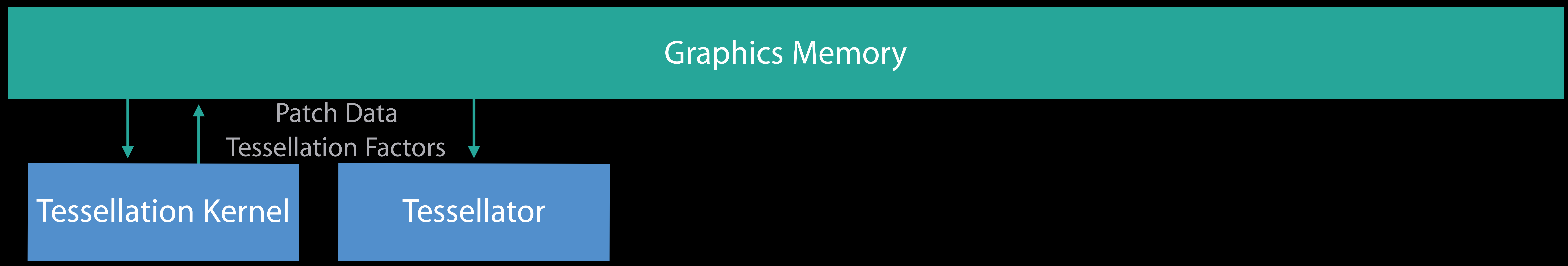
Metal Tessellation

Second stage — Tessellator



Metal Tessellation

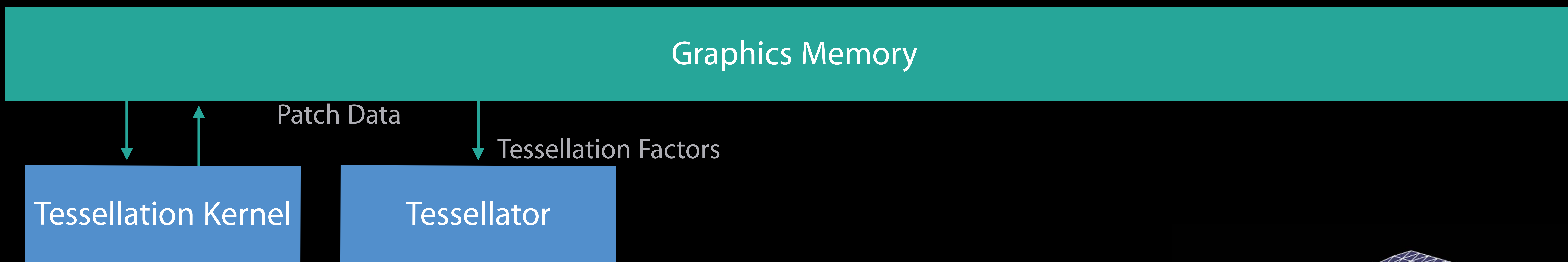
Second stage — Tessellator



A fixed-function stage but configurable

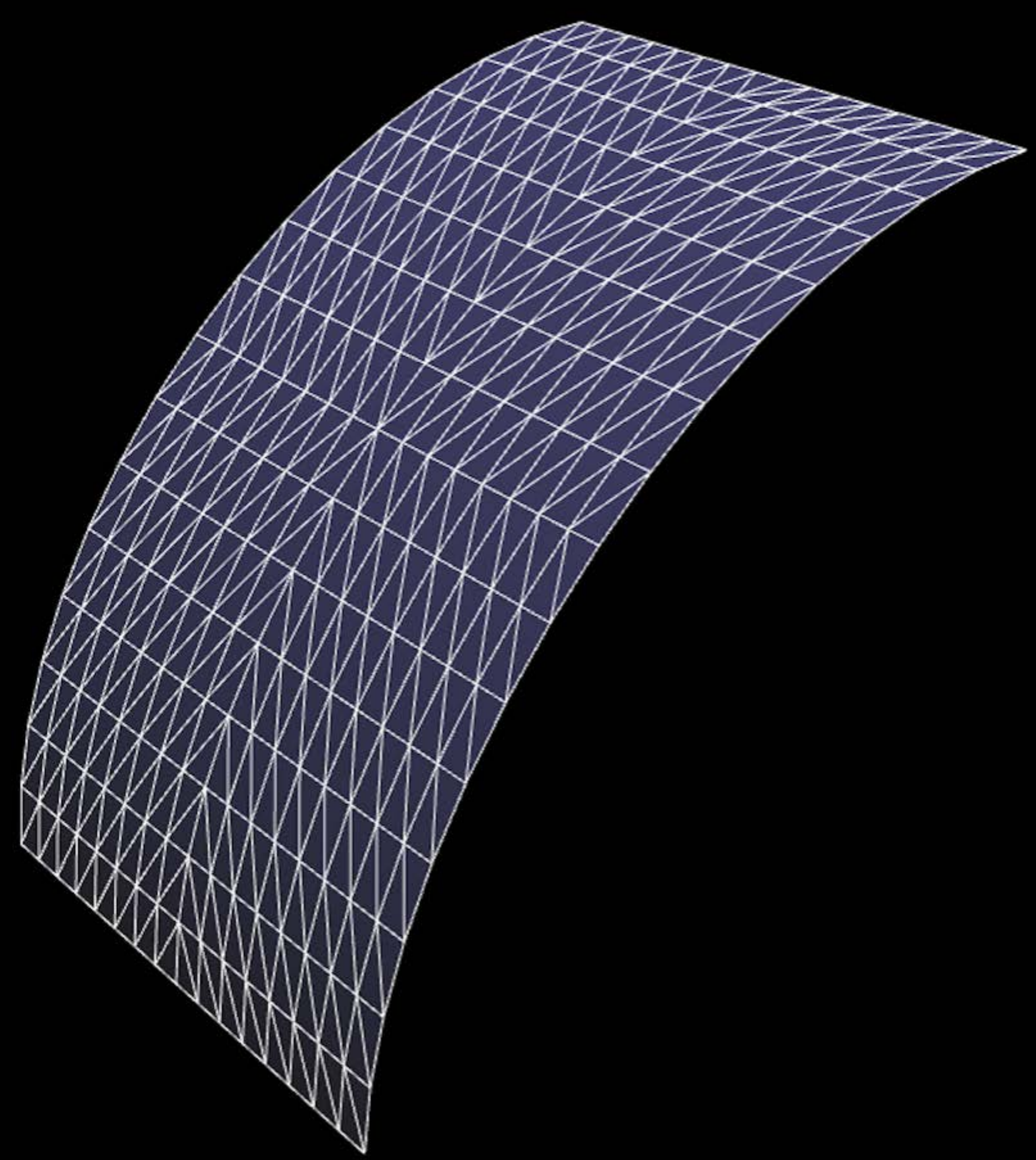
Metal Tessellation

Second stage — Tessellator



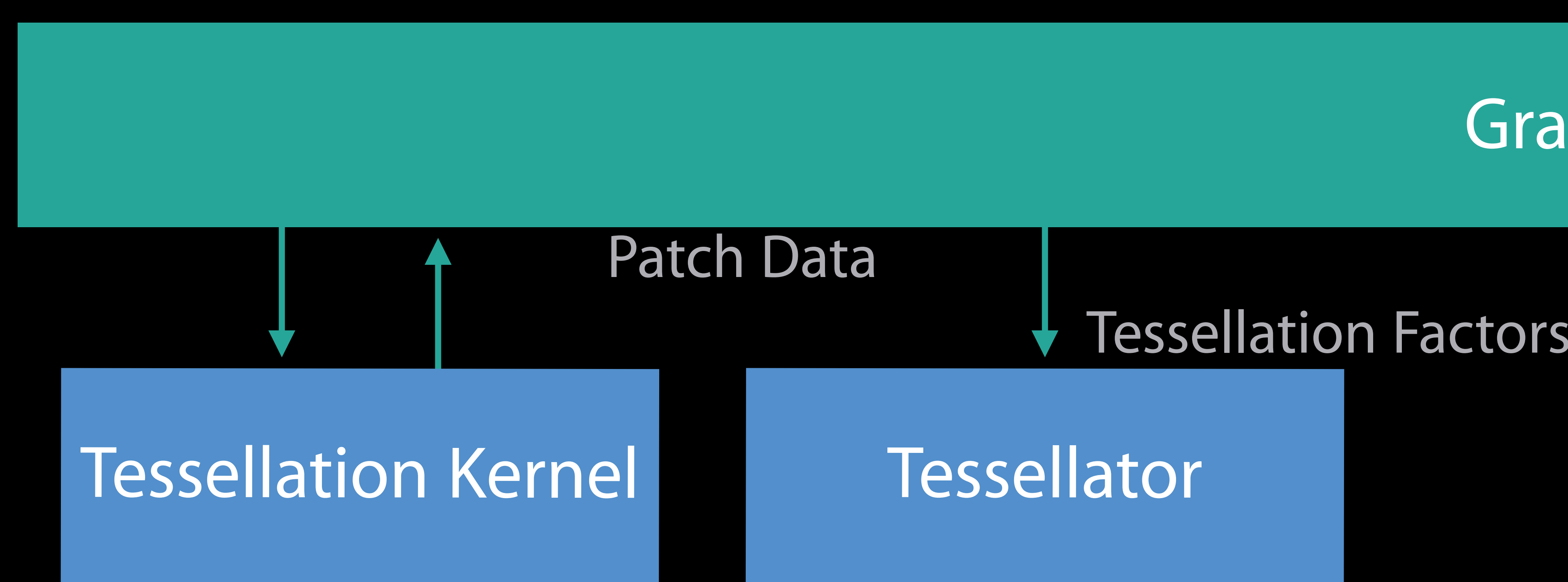
A fixed-function stage but configurable

- Subdivides the patch into triangles using the tessellation factors
- Triangles describe the high resolution geometry to draw



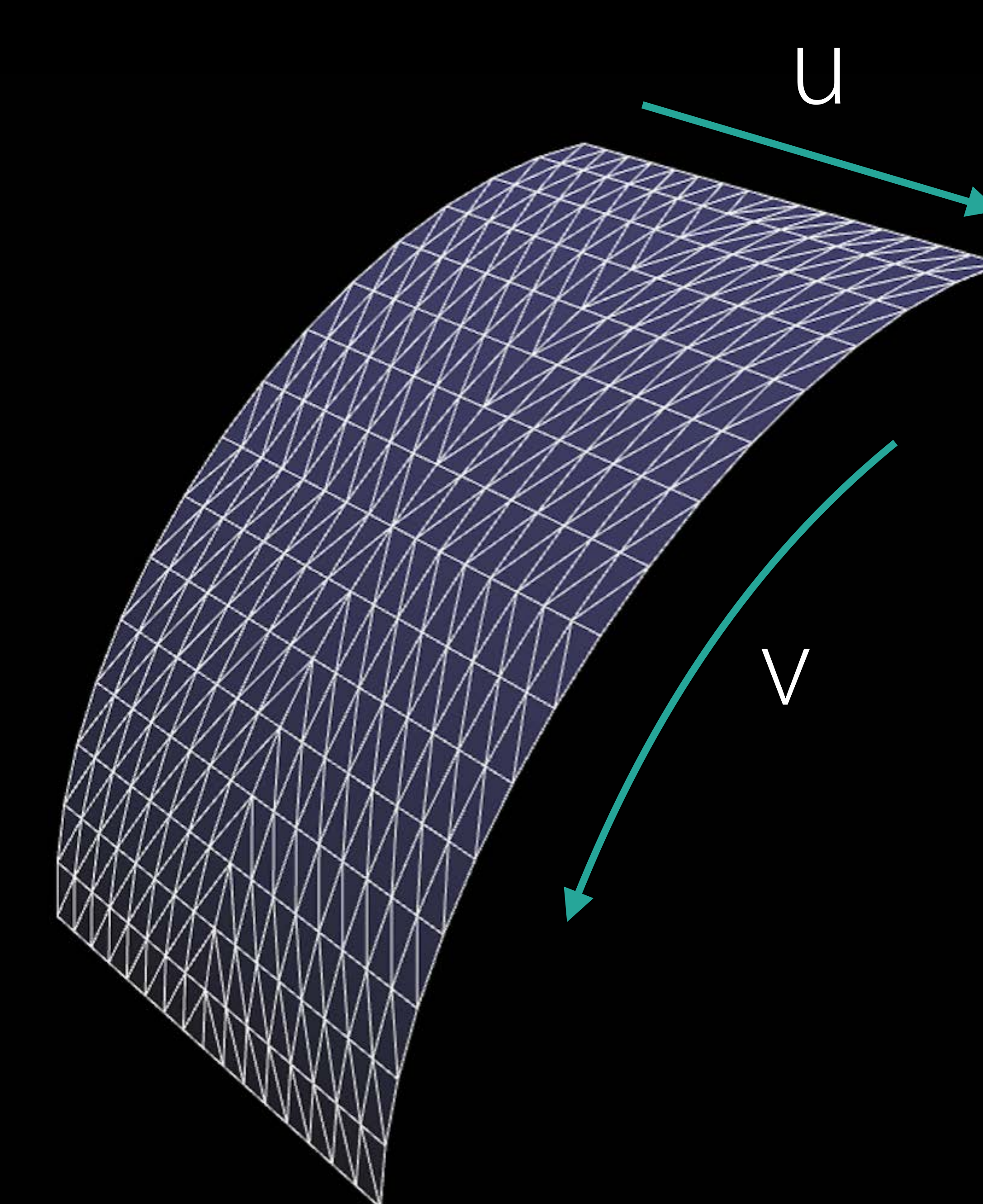
Metal Tessellation

Second stage — Tessellator



A fixed-function stage but configurable

- Subdivides the patch into triangles using the tessellation factors
- Triangles describe the high resolution geometry to draw
- Triangle list not stored in graphics memory
- Generates parametric (u, v) values for vertices on subdivided patch

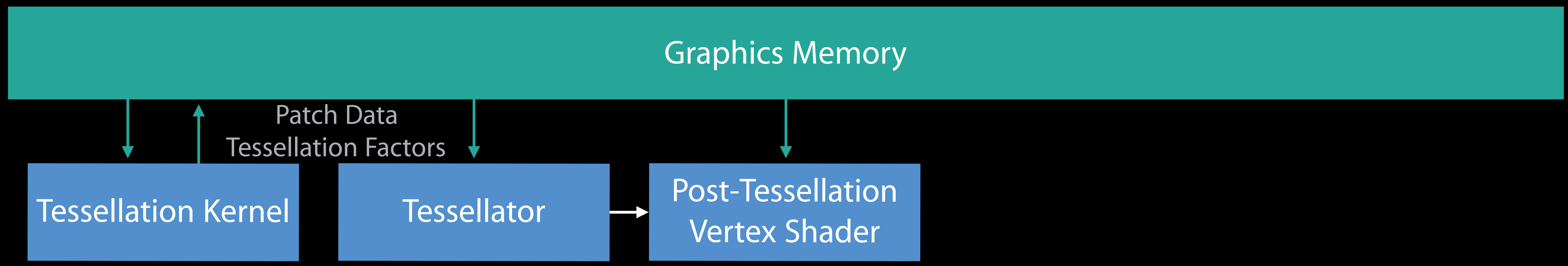


Metal Tessellation

Third stage — Post Tessellation Vertex Shader

Metal Tessellation

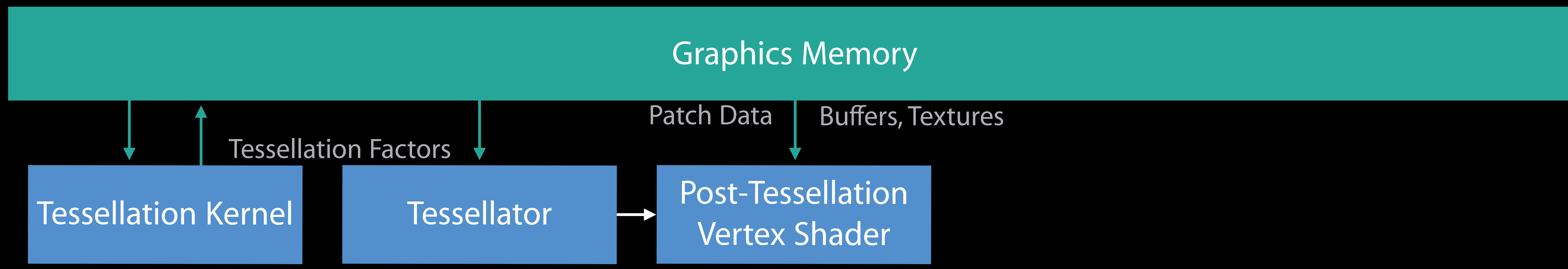
Third stage — Post Tessellation Vertex Shader



A programmable stage

Metal Tessellation

Third stage — Post Tessellation Vertex Shader

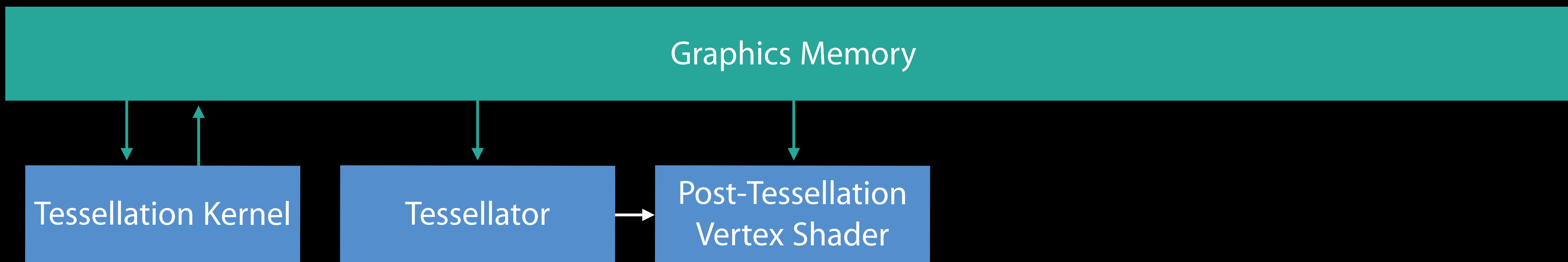


A programmable stage

- Executes for vertices generated by tessellator
- Evaluates position and other attributes on high resolution surface
- Similar role as the domain shader in DirectX

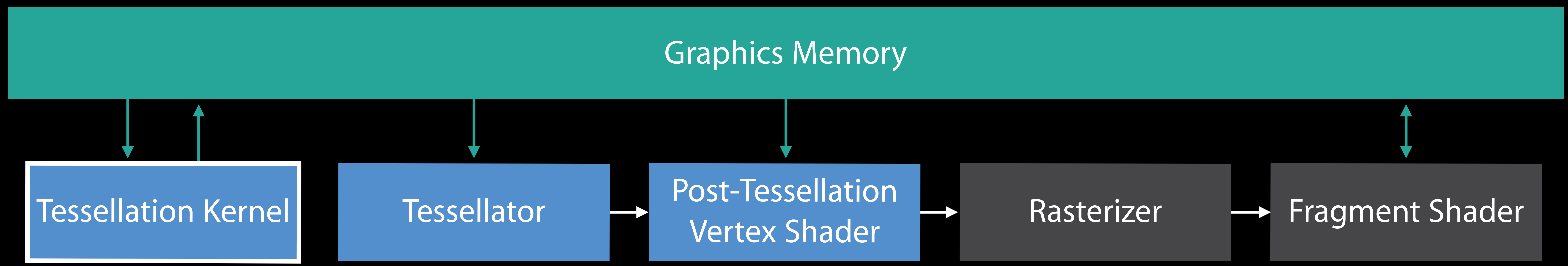
Metal Tessellation

Complete pipeline



Metal Tessellation

Complete pipeline

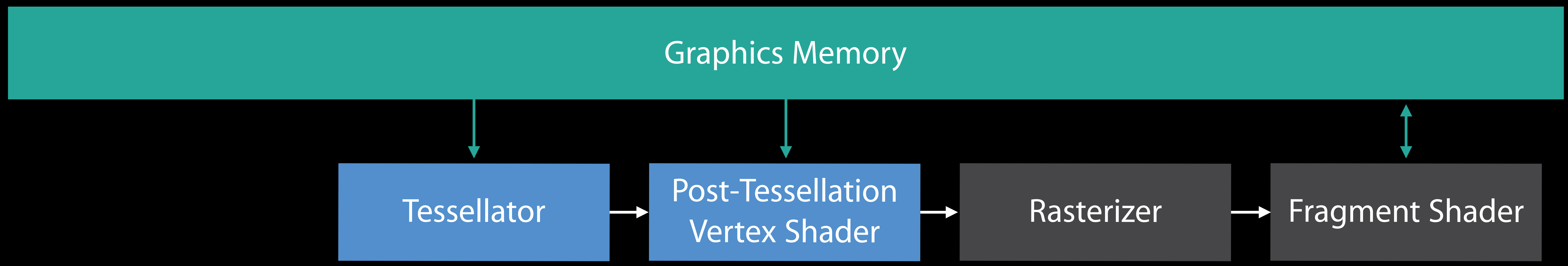


Tessellation factors and patch data can also be generated by vertex or fragment shader

- Compute kernel preferred as it can asynchronously execute with draw commands on GPU

Metal Tessellation

Complete pipeline



Tessellation factors and patch data can also be generated by vertex or fragment shader

- Compute kernel preferred as it can asynchronously execute with draw commands on GPU

Tessellation kernel does not need to execute every frame

- Apply a scale value to the tessellation factors

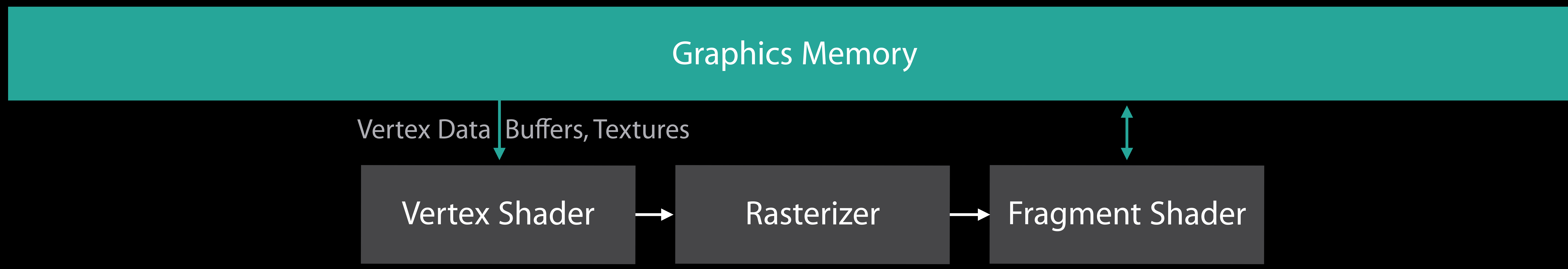
Metal Graphics Pipeline

Without Tessellation

Graphics Memory

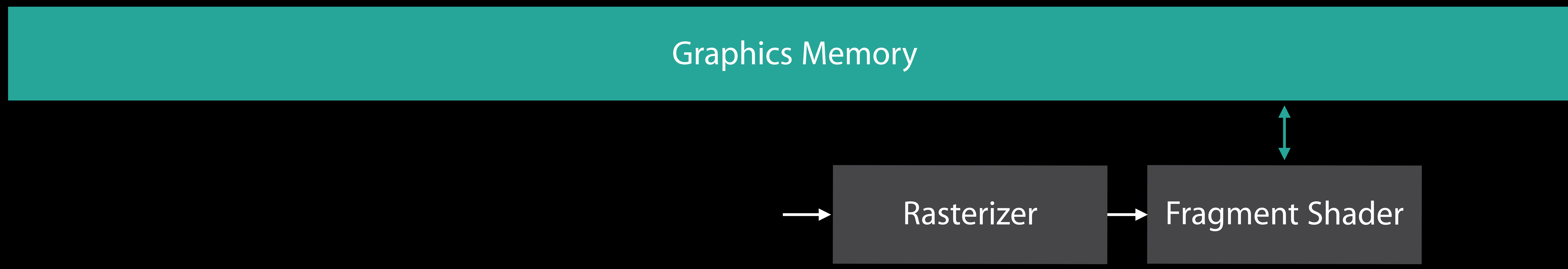
Metal Graphics Pipeline

Without Tessellation



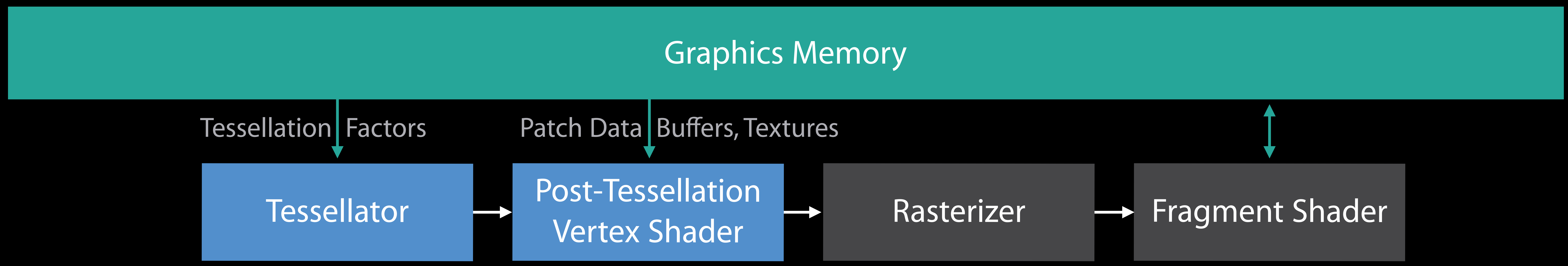
Metal Graphics Pipeline

With Tessellation



Metal Graphics Pipeline

With Tessellation



Rendering Geometry with Tessellation

Rendering Geometry with Tessellation

Rendering Geometry with Tessellation

Writing a Post-Tessellation Vertex Shader

Specifying Patch Data Inputs

Configuring the Tessellator

Drawing Patches

Writing a Post-Tessellation Vertex Shader

Meet the new shader, same as the old shader

```
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(uint patchID [[ patch_id ]],
                               float2 patchUV [[ position_in_patch ]],
                               MyPatchData patchData [[ stage_in ]],
                               …)
```

Writing a Post-Tessellation Vertex Shader

Meet the new shader, same as the old shader

A vertex shader that executes for vertices of a tessellated surface

```
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(uint patchID [[ patch_id ]],
                               float2 patchUV [[ position_in_patch ]],
                               MyPatchData patchData [[ stage_in ]],
                               …)
```

Writing a Post-Tessellation Vertex Shader

Meet the new shader, same as the old shader

A vertex shader that executes for vertices of a tessellated surface

```
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(uint patchID [[ patch_id ]],
                               float2 patchUV [[ position_in_patch ]],
                               MyPatchData patchData [[ stage_in ]],
                               …)
```

Writing a Post-Tessellation Vertex Shader

Meet the new shader, same as the old shader

A vertex shader that executes for vertices of a tessellated surface

Shader Inputs

- Patch Data vs. Vertex Data

```
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(uint patchID [[ patch_id ]],
                               float2 patchUV [[ position_in_patch ]],
                               MyPatchData patchData [[ stage_in ]],
                               …)
```

Writing a Post-Tessellation Vertex Shader

Meet the new shader, same as the old shader

A vertex shader that executes for vertices of a tessellated surface

Shader Inputs

- Patch Data vs. Vertex Data

```
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(uint patchID [[ patch_id ]],
                               float2 patchUV [[ position_in_patch ]],
                               MyPatchData patchData [[ stage_in ]],
                               …)
```

Writing a Post-Tessellation Vertex Shader

Meet the new shader, same as the old shader

A vertex shader that executes for vertices of a tessellated surface

Shader Inputs

- Patch Data vs. Vertex Data

```
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(uint patchID [[ patch_id ]],
                               float2 patchUV [[ position_in_patch ]],
                               MyPatchData patchData [[ stage_in ]],
                               …)
```

Writing a Post-Tessellation Vertex Shader

Meet the new shader, same as the old shader

A vertex shader that executes for vertices of a tessellated surface

Shader Inputs

- Patch Data vs. Vertex Data

Shader Outputs

- Similar to outputs from a vertex shader

```
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(uint patchID [[ patch_id ]],
                               float2 patchUV [[ position_in_patch ]],
                               MyPatchData patchData [[ stage_in ]],
                               …)
```

Patch Data Inputs

Two types of input data

Patch Data Inputs

Two types of input data

Per-patch and control-point data

- Multiple control-points per patch
  - Example: 16 control-points in a Bezier patch

Patch Data Inputs

Two types of input data

Per-patch and control-point data

- Multiple control-points per patch
  - Example: 16 control-points in a Bezier patch

Use `MTLVertexDescriptor` to specify the patch data layout in memory

Patch Data Inputs

Two types of input data

Per-patch and control-point data

- Multiple control-points per patch
  - Example: 16 control-points in a Bezier patch

Use `MTLVertexDescriptor` to specify the patch data layout in memory

Declared as a struct with `[[ stage_in ]]` qualifier in post-tessellation vertex shader

- Attribute index used to identify and match patch input data with `MTLVertexDescriptor`
- Control-point data specified as a templated type

```
// Control-Point Data
struct ControlPointData {
  float3 position [[ attribute(0) ]];
  uint   a        [[ attribute(1) ]];
};
```

```
// Per-Patch Data
struct PerPatchData {
  float b [[ attribute(2) ]];
  uint  c [[ attribute(3) ]];
};
```

```
// struct that stores the per-patch and control-point data
struct MyPatchData {
  PerPatchData patchData;
  patch_control_point<ControlPointData> controlPointData;
};

// post-tessellation vertex shader
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(MyPatchData patchData [[ stage_in ]], …)
```

```
// Control-Point Data
struct ControlPointData {
  float3 position [[ attribute(0) ]];
  uint   a        [[ attribute(1) ]];
};

// Per-Patch Data
struct PerPatchData {
  float b [[ attribute(2) ]];
  uint  c [[ attribute(3) ]];
};

// struct that stores the per-patch and control-point data
struct MyPatchData {
  PerPatchData patchData;
  patch_control_point<ControlPointData> controlPointData;
};

// post-tessellation vertex shader
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(MyPatchData patchData [[ stage_in ]], …)
```

```
// Control-Point Data
struct ControlPointData {
  float3 position [[ attribute(0) ]];
  uint   a        [[ attribute(1) ]];
};

// Per-Patch Data
struct PerPatchData {
  float b [[ attribute(2) ]];
  uint  c [[ attribute(3) ]];
};

// struct that stores the per-patch and control-point data
struct MyPatchData {
  PerPatchData patchData;
  patch_control_point<ControlPointData> controlPointData;
};

// post-tessellation vertex shader
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(MyPatchData patchData [[ stage_in ]], …)
```

```
// Control-Point Data
struct ControlPointData {
  float3 position [[ attribute(0) ]];
  uint   a        [[ attribute(1) ]];
};
```

```
// Per-Patch Data
struct PerPatchData {
  float b [[ attribute(2) ]];
  uint  c [[ attribute(3) ]];
};
```

```
// struct that stores the per-patch and control-point data
struct MyPatchData {
  PerPatchData patchData;
  patch_control_point<ControlPointData> controlPointData;
};

// post-tesselation vertex shader
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(MyPatchData patchData [[ stage_in ]], …)
```

```
// Control-Point Data
struct ControlPointData {
  float3 position [[ attribute(0) ]];
  uint   a        [[ attribute(1) ]];
};
```

```
// Per-Patch Data
struct PerPatchData {
  float b [[ attribute(2) ]];
  uint  c [[ attribute(3) ]];
};
```

```
// struct that stores the per-patch and control-point data
struct MyPatchData {
  PerPatchData patchData;
  patch_control_point<ControlPointData> controlPointData;
};

// post-tessellation vertex shader
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(MyPatchData patchData [[ stage_in ]], …)
```

```
// Control-Point Data
struct ControlPointData {
  float3 position [[ attribute(0) ]];
  uint   a        [[ attribute(1) ]];
};

// Per-Patch Data
struct PerPatchData {
  float b [[ attribute(2) ]];
  uint  c [[ attribute(3) ]];
};

// struct that stores the per-patch and control-point data
struct MyPatchData {
  PerPatchData patchData;
  patch_control_point<ControlPointData> controlPointData;
};

// post-tesselation vertex shader
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(MyPatchData patchData [[ stage_in ]], …)
```

```
// Control-Point Data
struct ControlPointData {
  float3 position [[ attribute(0) ]];
  uint   a        [[ attribute(1) ]];
};
```

```
// Per-Patch Data
struct PerPatchData {
  float b [[ attribute(2) ]];
  uint  c [[ attribute(3) ]];
};
```

```
// struct that stores the per-patch and control-point data
struct MyPatchData {
  PerPatchData patchData;
  patch_control_point<ControlPointData> controlPointData;
};

// post-tesselation vertex shader
[[ patch(quad, 16) ]]
vertex VertexOutput
myPostTessellationVertexShader(MyPatchData patchData [[ stage_in ]], …)
```

Configuring the Tessellator

Configuring the Tessellator

Tessellation Properties

- Specified in MTLRenderPipelineDescriptor

Configuring the Tessellator

Tessellation Properties

- Specified in MTLRenderPipelineDescriptor

Tessellation Factors

- Edge and Inside factors
- 16-bit floating-point values

```
// specify the tessellation factor buffer in the MTLRenderCommandEncoder
renderEncoder.setTessellationFactorBuffer(buffer, offset: offset, instanceStride: stride)
```

Drawing Patches

New APIs in MTLRenderCommandEncoder

Drawing Patches

New APIs in MTLRenderCommandEncoder

Non-indexed and indexed draw patch APIs

- Specify patch start, number of patches to draw, control-point index buffer . . .

```
renderEncoder.drawPatches(numberOfControlPoints,
                          patchStart: patchStart, numberOfPatches: n, …)
renderEncoder.drawIndexedPatches(numberOfControlPoints,
                                 patchStart: patchStart, numberOfPatches: n, …,
                                 controlPointIndexBuffer: buffer,
                                 controlPointIndexBufferOffset: offset, …)
```

Drawing Patches

New APIs in MTLRenderCommandEncoder

Non-indexed and indexed draw patch APIs

- Specify patch start, number of patches to draw, control-point index buffer . . .

```
renderEncoder.drawPatches(numberOfControlPoints,
                          patchStart: patchStart, numberOfPatches: n, …)
renderEncoder.drawIndexedPatches(numberOfControlPoints,
                                 patchStart: patchStart, numberOfPatches: n, …,
                                 controlPointIndexBuffer: buffer,
                                 controlPointIndexBufferOffset: offset, …)
```

DrawIndirect variants

- Allows previous draw or dispatch commands to generate draw parameters
- Draw parameters such as patch start, patch count specified in a buffer filled out on GPU

Adopting Metal Tessellation

unity

"Unity Technologies is proud to be working with Apple to make our Metal renderer the best of its kind, enabling Unity developers to harness the power of Metal. Later this year, we will be shipping support for Metal Tessellation, Metal Compute, and the ability to write native Metal shaders in Unity."

Unity Technology

UNREAL
ENGINE

“Metal provides great performance and efficiency improvements alongside a wonderfully clean programming model. We're very enthusiastic about the ongoing revolution in high-end graphics on iOS and Mac enabled by Metal.”

Epic Games

Demo

Tessellation in Action

Adopting Metal Tessellation

Digital content creation tools

OpenSubdiv

"OpenSubdiv is a key technology at Pixar that helps our artists create expressive performances and beautiful worlds. Pixar is thrilled to see the high performance and full fidelity of OpenSubdiv realized with Metal Tessellation on macOS and iOS devices and with Apple's contribution of a native Metal implementation to the OpenSubdiv open source project."

David Yu

Senior Software Engineer of Pixar's GPU Team

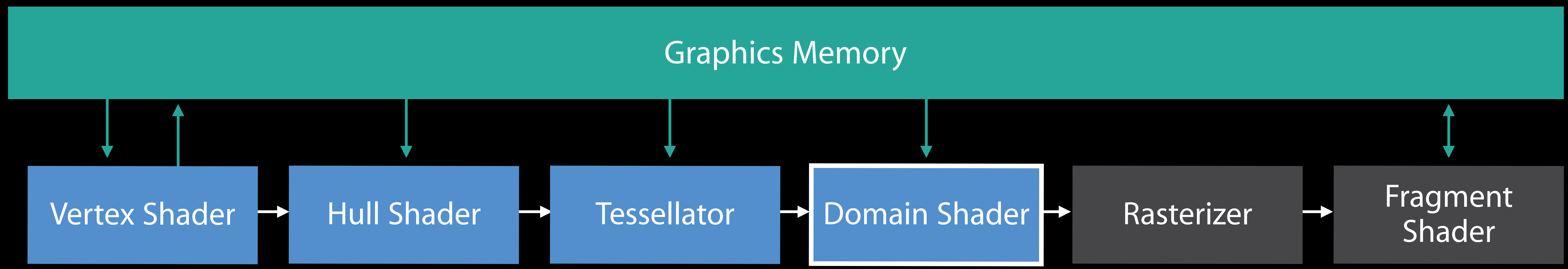
Adopting Metal Tessellation

From DirectX to Metal

Graphics Memory

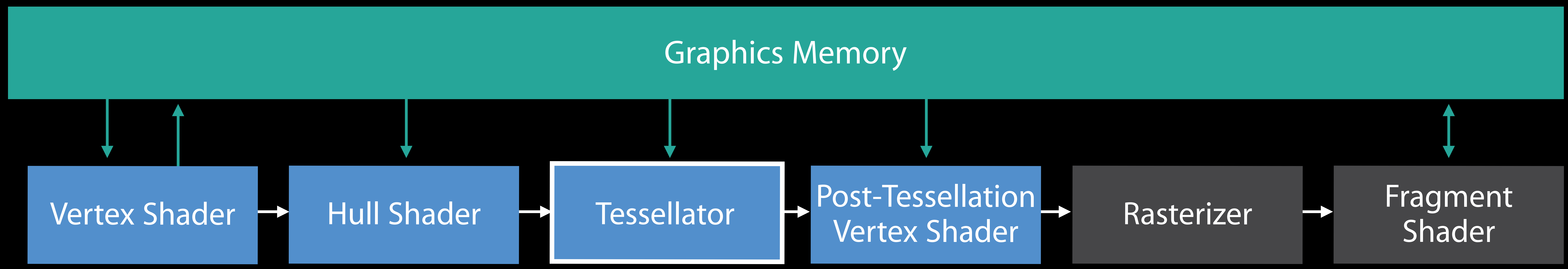
Adopting Metal Tessellation

From DirectX to Metal



Adopting Metal Tessellation

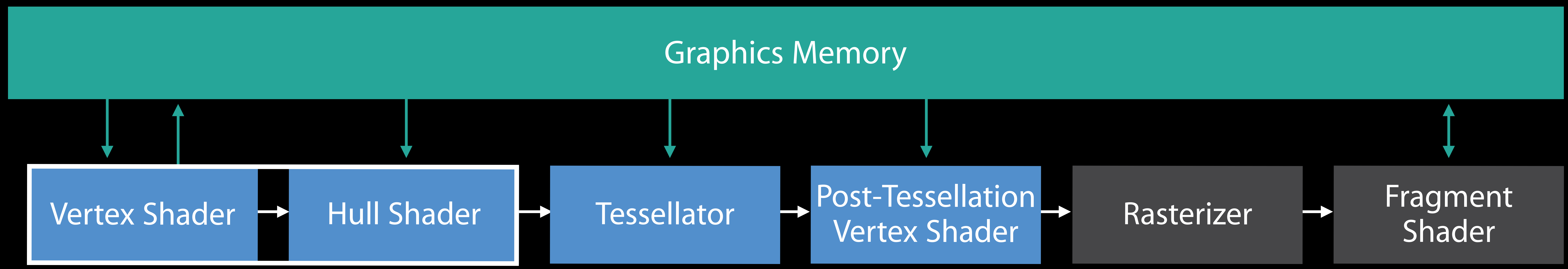
From DirectX to Metal



- One-to-one mapping between domain and post-tessellation vertex shader
- Tessellator remains the same

Adopting Metal Tessellation

From DirectX to Metal

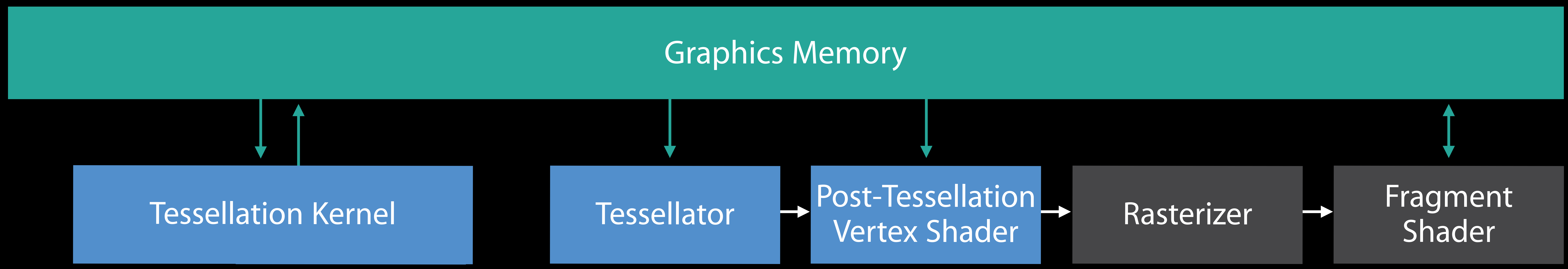


One-to-one mapping between domain and post-tessellation vertex shader

Tessellator remains the same

Adopting Metal Tessellation

From DirectX to Metal



One-to-one mapping between domain and post-tessellation vertex shader

Tessellator remains the same

Vertex and hull shader to be translated to a Metal kernel

Moving Existing Tessellation Shaders to Metal

Vertex + Hull Shader = Metal Tessellation Kernel

Moving Existing Tessellation Shaders to Metal

Vertex + Hull Shader = Metal Tessellation Kernel

Vertex descriptor used to describe vertex data layout

- `[[ stage_in ]]` in a Metal kernel
- Input data layout described in a `MTLStageInputOutputDescriptor`

Moving Existing Tessellation Shaders to Metal

Vertex + Hull Shader = Metal Tessellation Kernel

Vertex descriptor used to describe vertex data layout

- `[[ stage_in ]]` in a Metal kernel
- Input data layout described in a `MTLStageInputOutputDescriptor`

Vertex and hull shader observations

- Vertex shader executes for each control-point
- Hull shader described by control-point and per-patch functions

Moving Existing Tessellation Shaders to Metal

Vertex + Hull Shader = Metal Tessellation Kernel

Vertex descriptor used to describe vertex data layout

- `[[ stage_in ]]` in a Metal kernel
- Input data layout described in a `MTLStageInputOutputDescriptor`

Vertex and hull shader observations

- Vertex shader executes for each control-point
- Hull shader described by control-point and per-patch functions

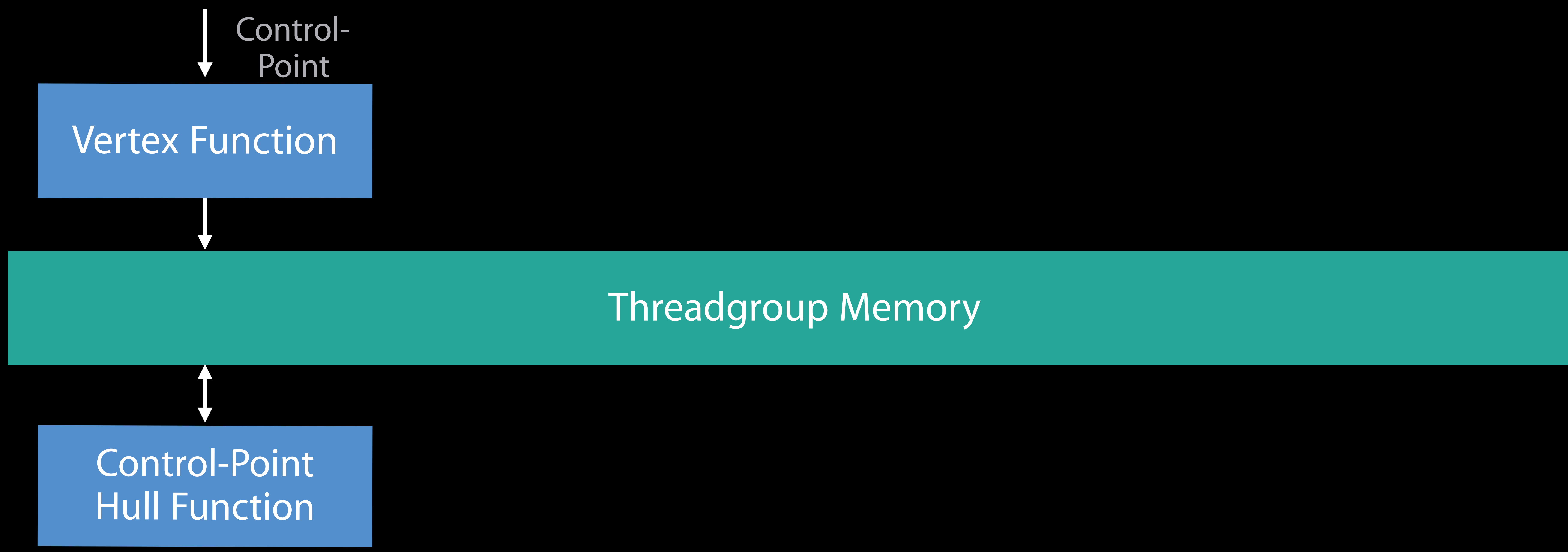
Translate vertex and hull shaders to Metal functions

Metal Tessellation

Vertex + Hull Shader = Metal Tessellation Kernel

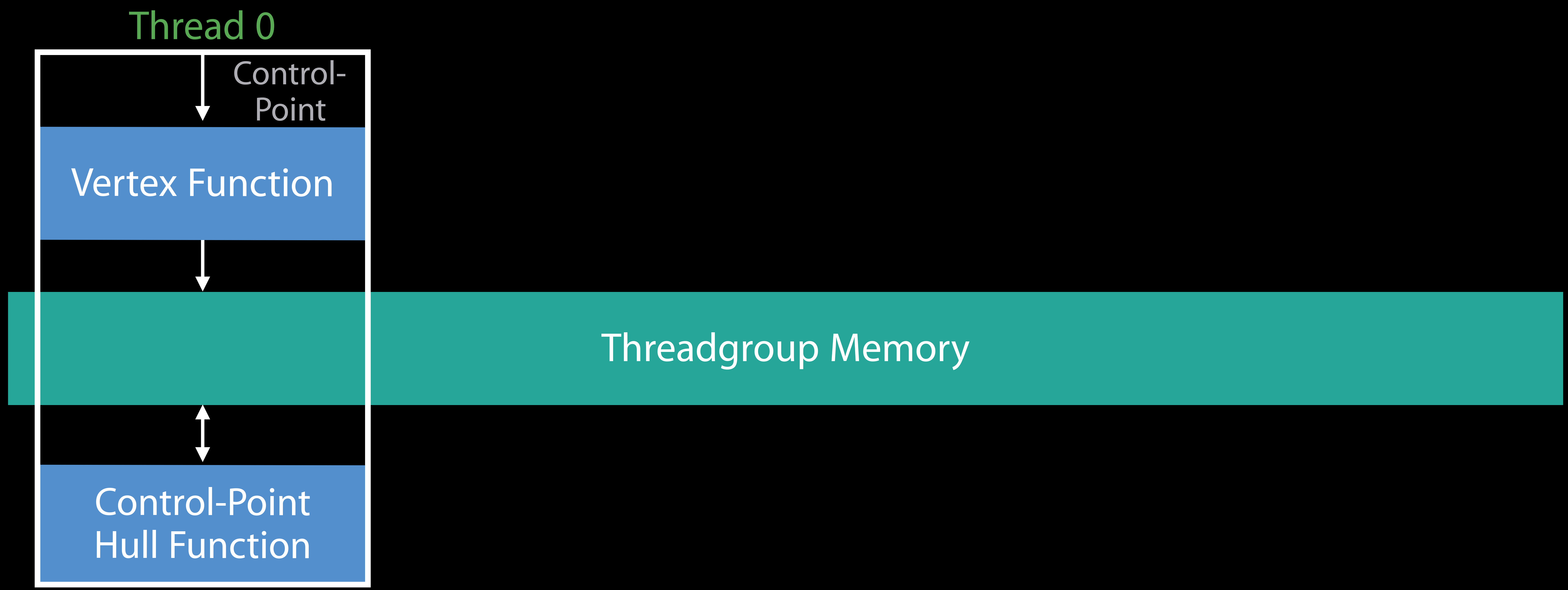
Metal Tessellation

Vertex + Hull Shader = Metal Tessellation Kernel



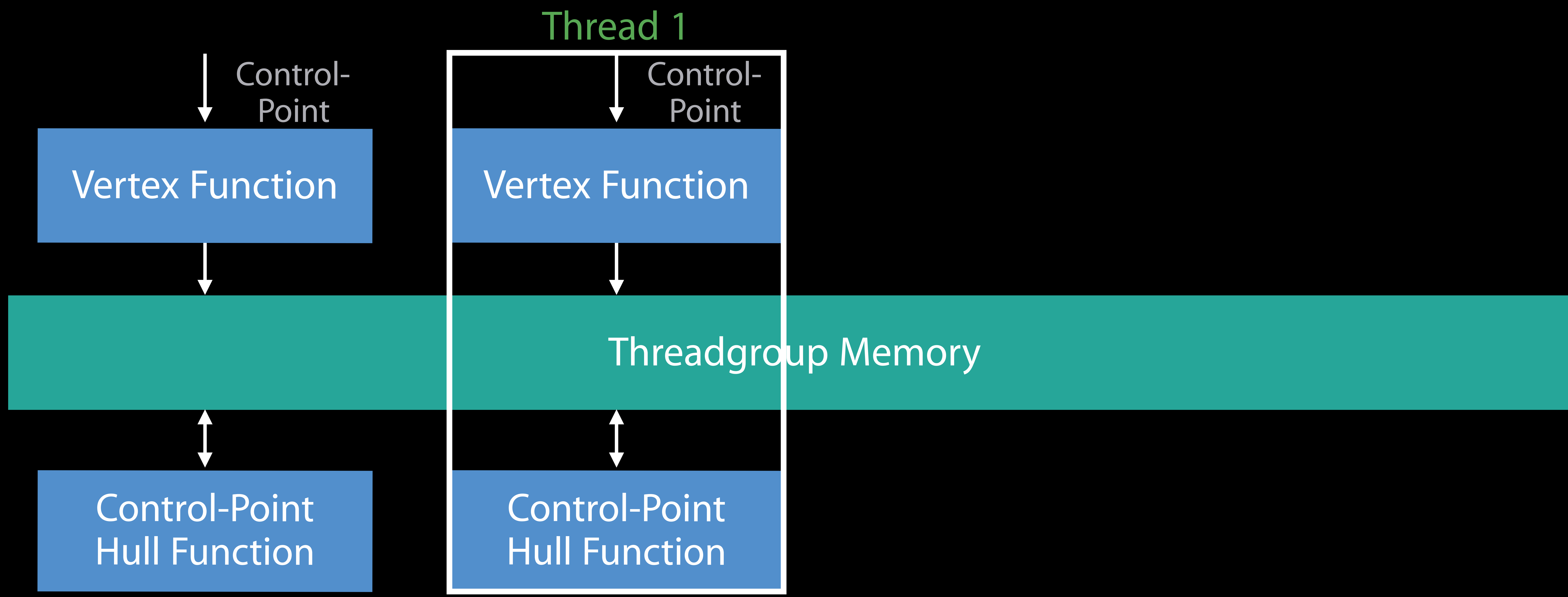
Metal Tessellation

Vertex + Hull Shader = Metal Tessellation Kernel



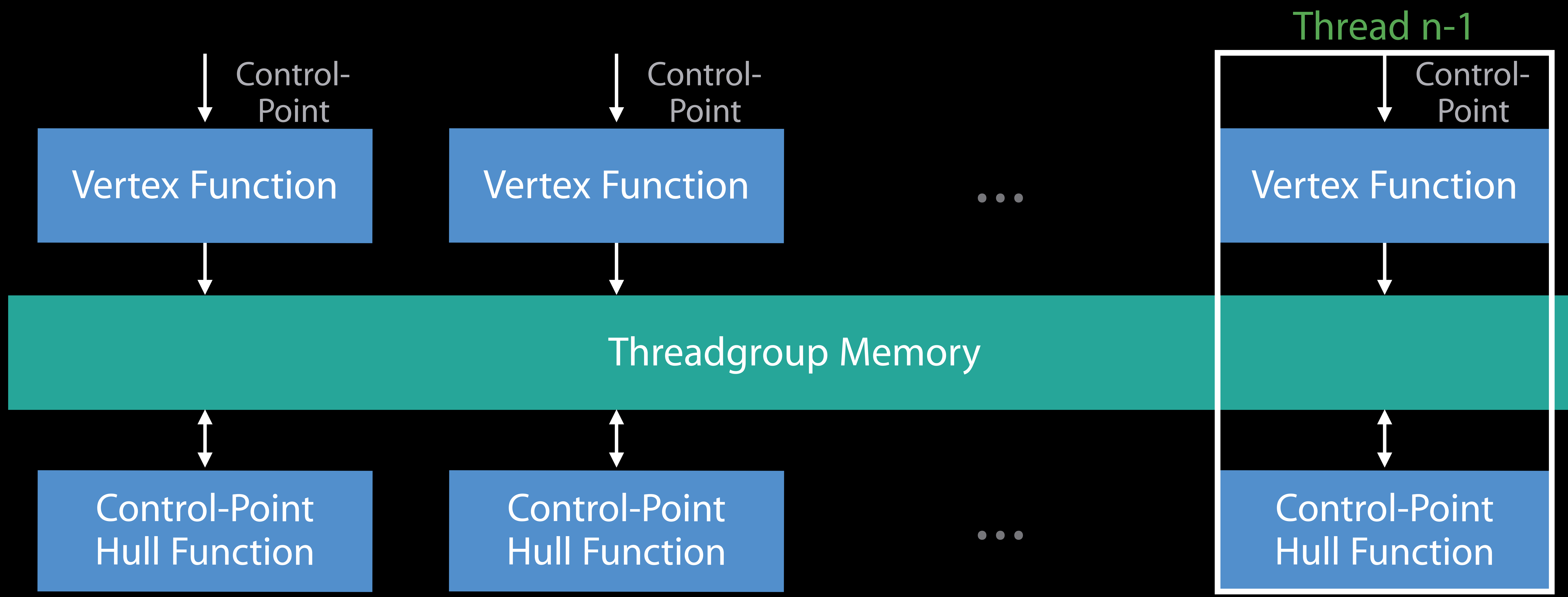
Metal Tessellation

Vertex + Hull Shader = Metal Tessellation Kernel



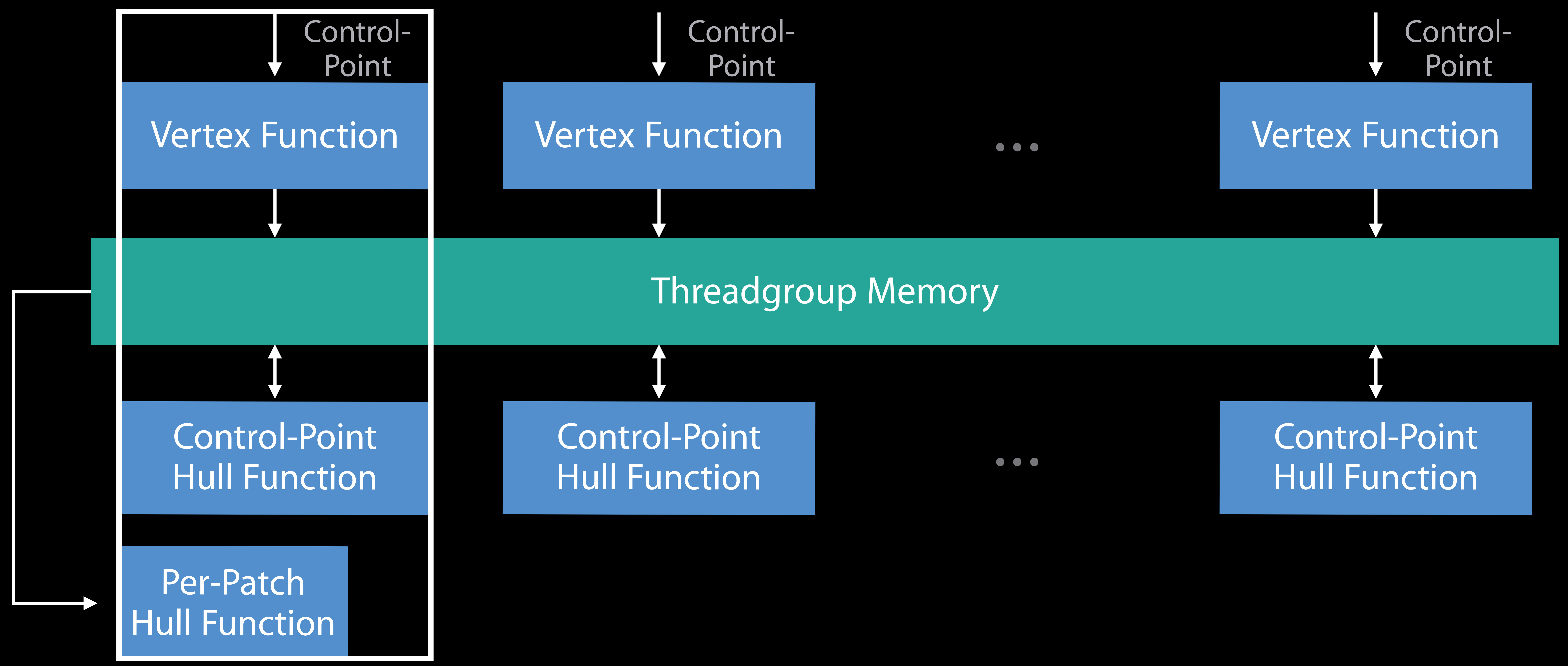
Metal Tessellation

Vertex + Hull Shader = Metal Tessellation Kernel



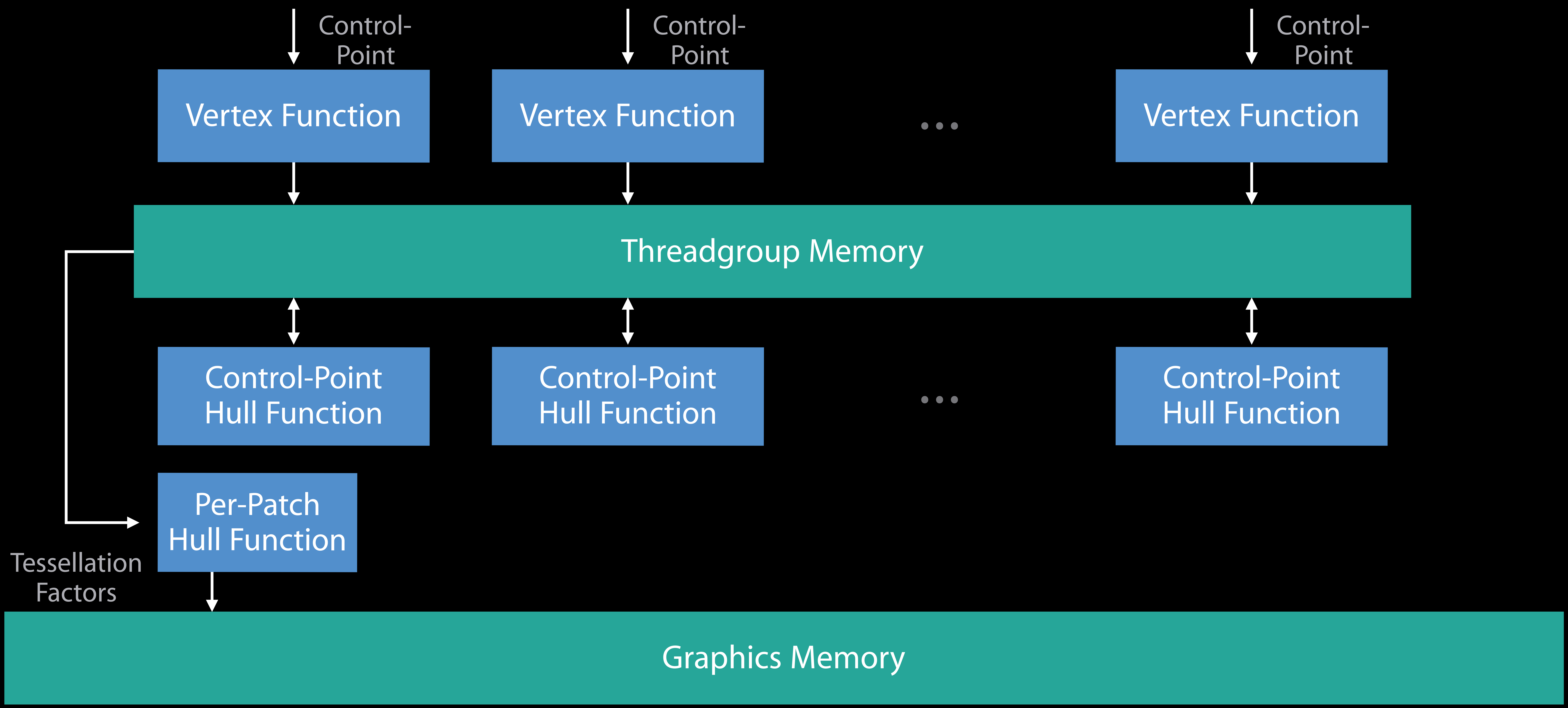
Metal Tessellation

Vertex + Hull Shader = Metal Tessellation Kernel



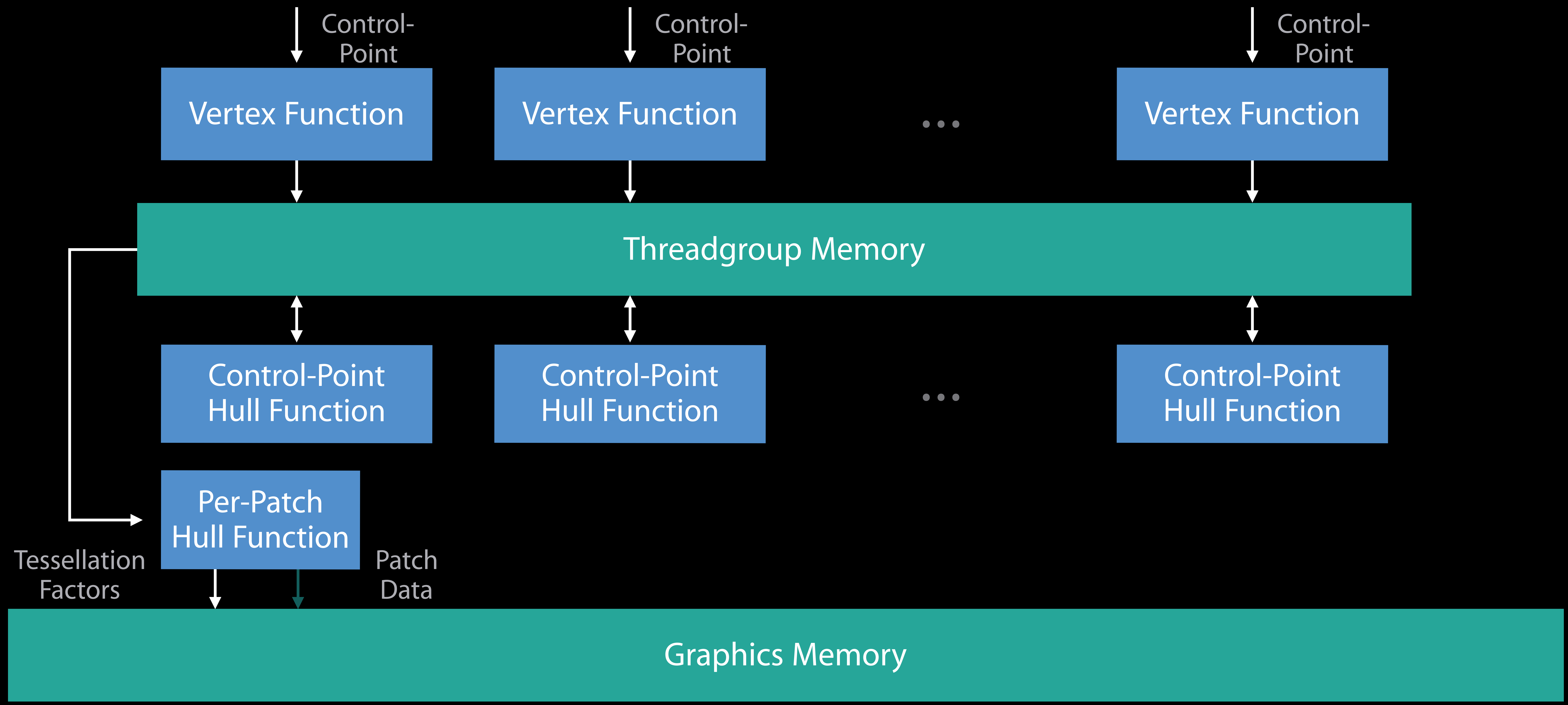
Metal Tessellation

Vertex + Hull Shader = Metal Tessellation Kernel



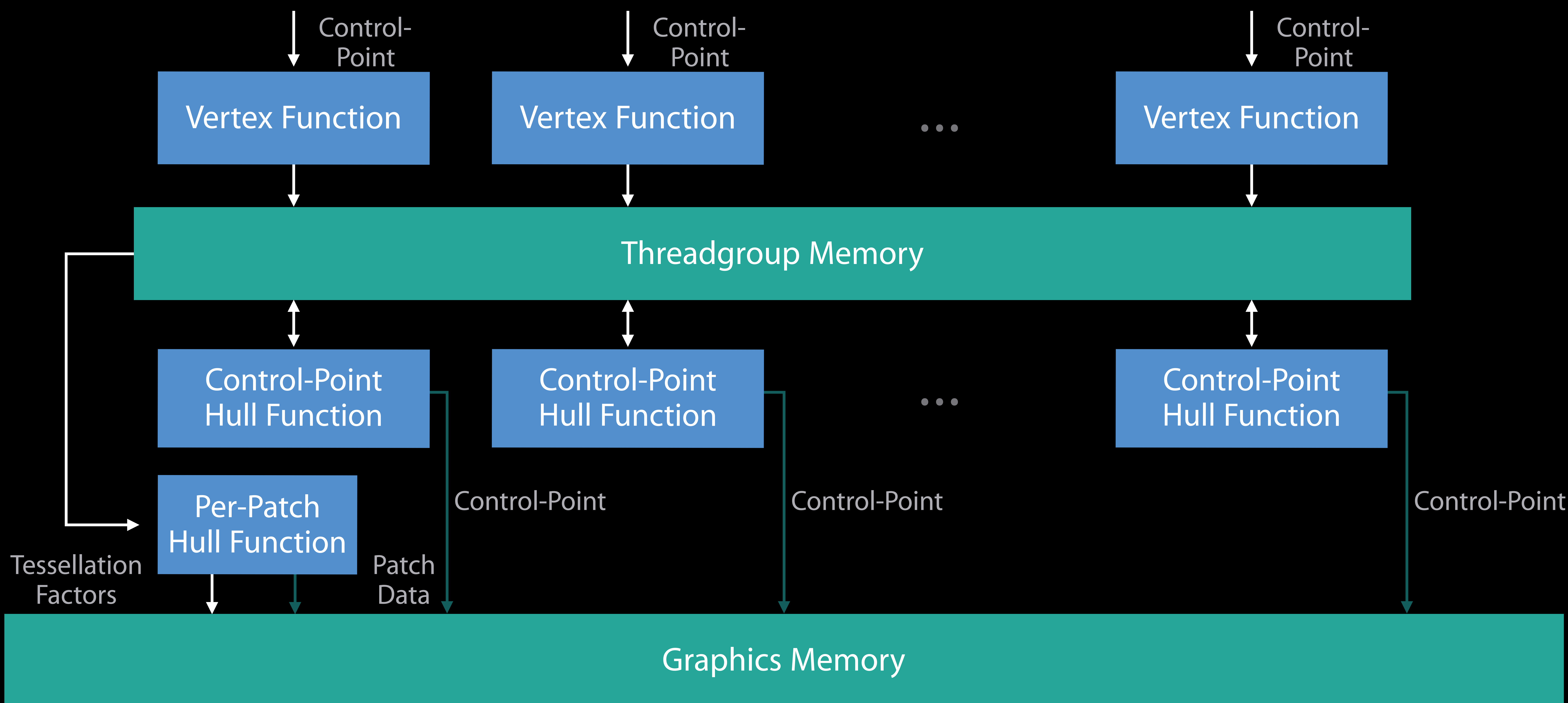
Metal Tessellation

Vertex + Hull Shader = Metal Tessellation Kernel



Metal Tessellation

Vertex + Hull Shader = Metal Tessellation Kernel



Tessellation

Summary

Tessellation

Summary

Simple to use and performant

Easy to adapt your existing tessellation code to Metal

Available on iOS and macOS

Tessellation

Summary

Simple to use and performant

Easy to adapt your existing tessellation code to Metal

Available on iOS and macOS

Call to Action

- Use tessellation to improve the visual content rendered by your application

Resource Heaps and Memoryless Render Targets

James Ding GPU Software Engineer

Resource Heaps

Lower overhead resource management

NEW

iOS

tvOS

Resource Heaps

Lower overhead resource management

NEW

Resource Sub-Allocation

iOS

tvOS

Resource Heaps

NEW

Lower overhead resource management

Resource Sub-Allocation

Resource Aliasing

iOS

tvOS

Resource Heaps

NEW

Lower overhead resource management

Resource Sub-Allocation

Resource Aliasing

Explicit Command Synchronization

iOS

tvOS

Resource Sub-Allocation

Resource Sub-Allocation

Faster resource creation and binding

Resource Sub-Allocation

Faster resource creation and binding

Resource creation with MTLDevice

- Heavy CPU operation

Resource Sub-Allocation

Faster resource creation and binding

Resource creation with MTLDevice

- Heavy CPU operation

Resource Sub-Allocation

Faster resource creation and binding

Resource creation with MTLDevice

- Heavy CPU operation

Resource binding

- Tracking costs add up for complex scenes

Resource Sub-Allocation

Faster resource creation and binding

Resource creation with MTLDevice

- Heavy CPU operation

Resource binding

- Tracking costs add up for complex scenes

Resource Sub-Allocation

Faster resource creation and binding

Resource creation with MTLDevice

- Heavy CPU operation

Resource binding

- Tracking costs add up for complex scenes

Resource Heaps address these performance issues

- Perform expensive memory operations ahead of time
- Only heaps are tracked, not their resources

Resource Creation

Comparing MTLDevice and MTLHeap

NEW

Resource Creation

Comparing MTLDevice and MTLHeap

NEW

Creating Resources with MTLDevice

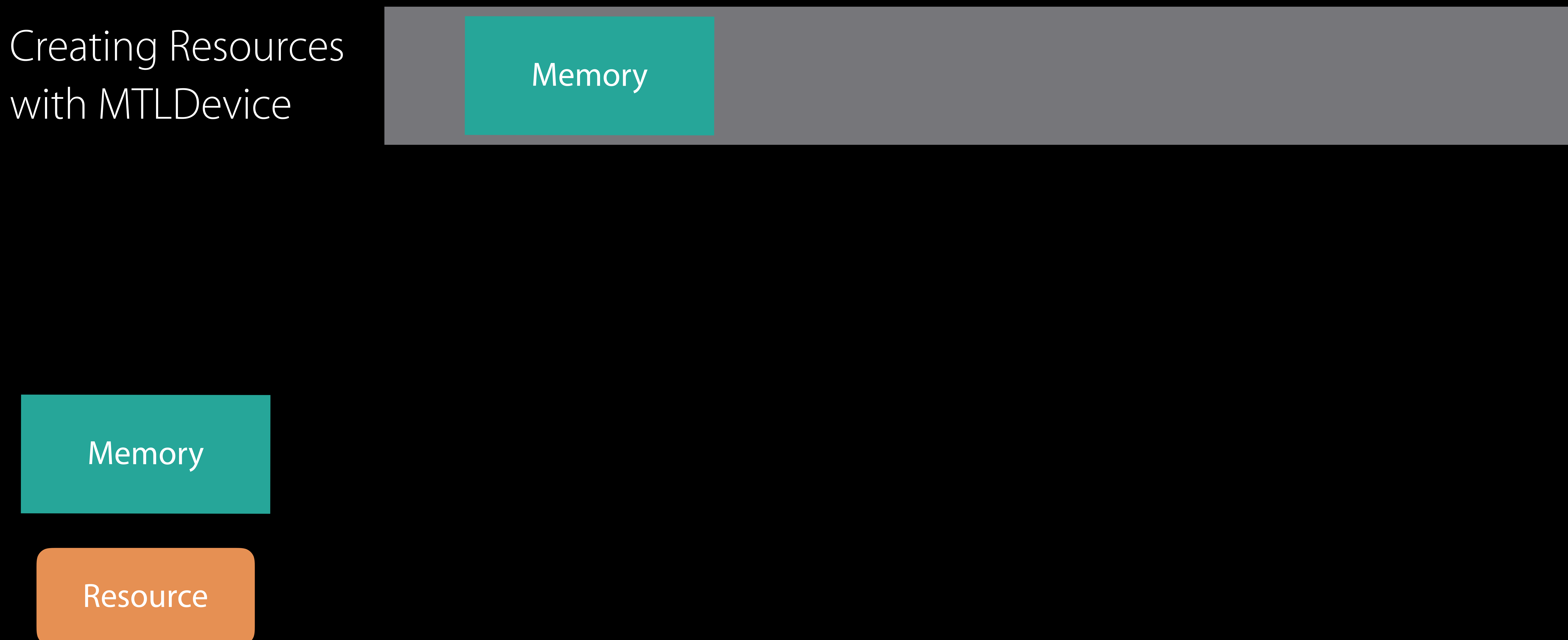
Memory

Resource

Resource Creation

Comparing MTLDevice and MTLHeap

NEW

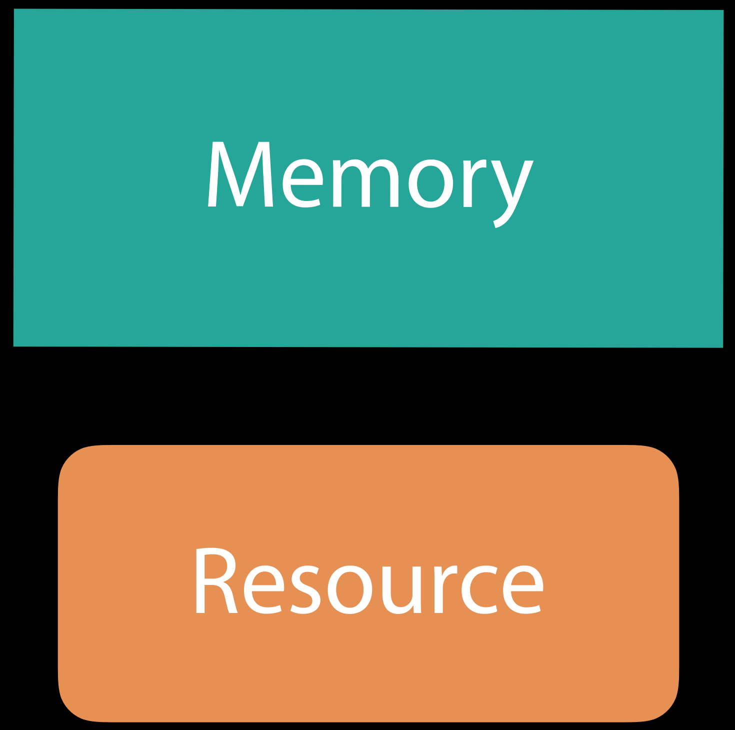


Resource Creation

NEW

Comparing MTLDevice and MTLHeap

Creating Resources with MTLDevice

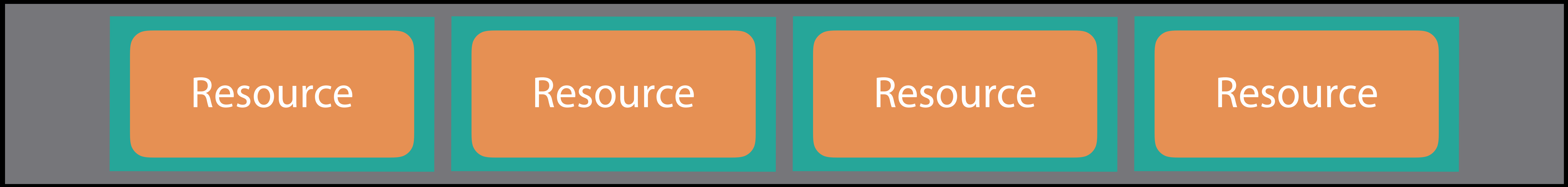


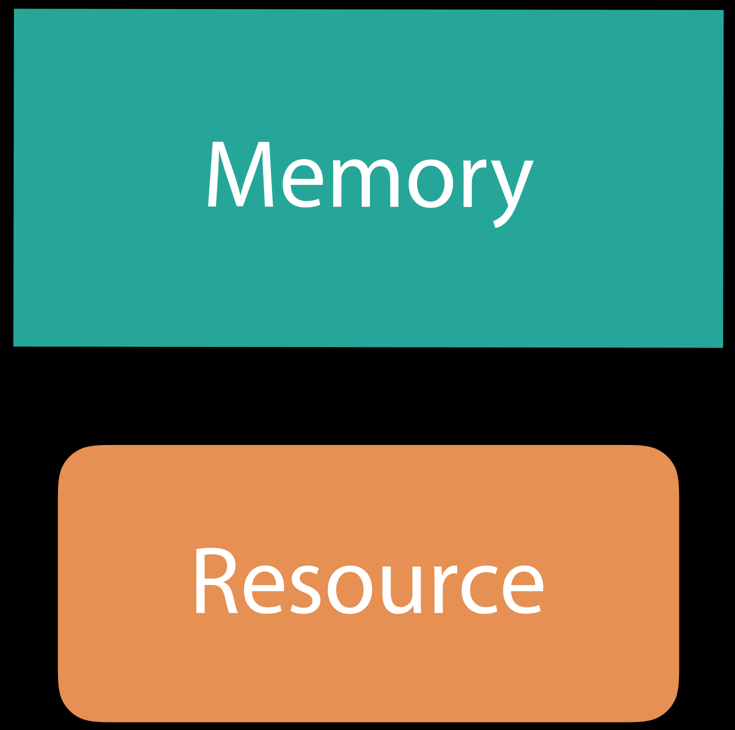
Resource Creation

Comparing MTLDevice and MTLHeap

NEW

Creating Resources
with MTLDevice





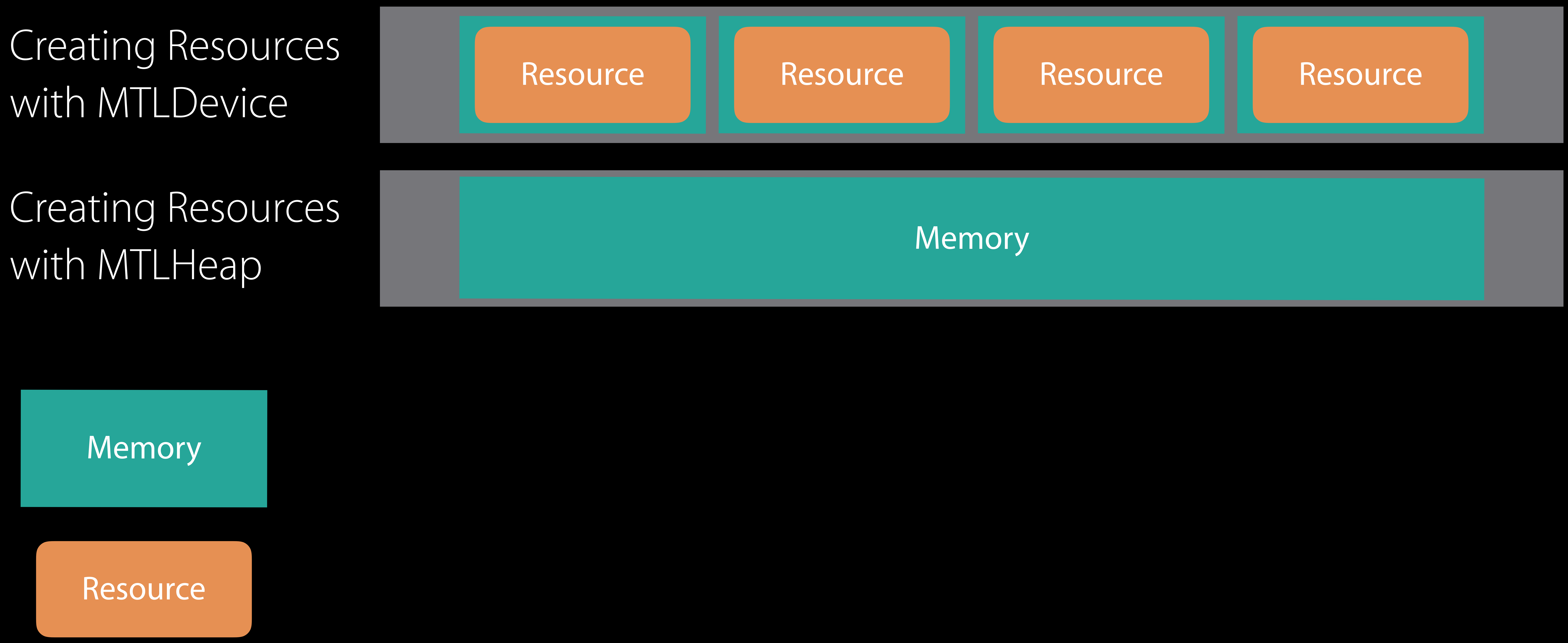
Resource Creation

Comparing MTLDevice and MTLHeap

NEW

Resource Creation

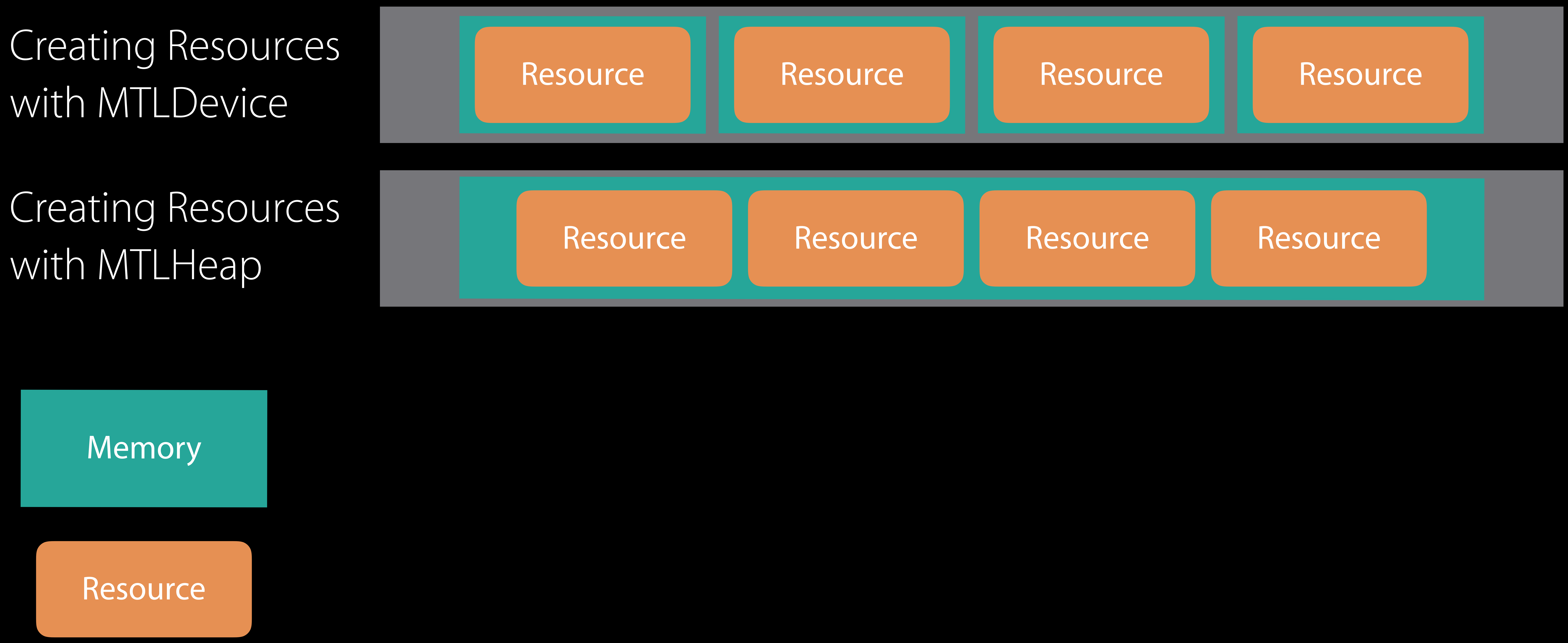
Comparing MTLDevice and MTLHeap



Resource Creation

NEW

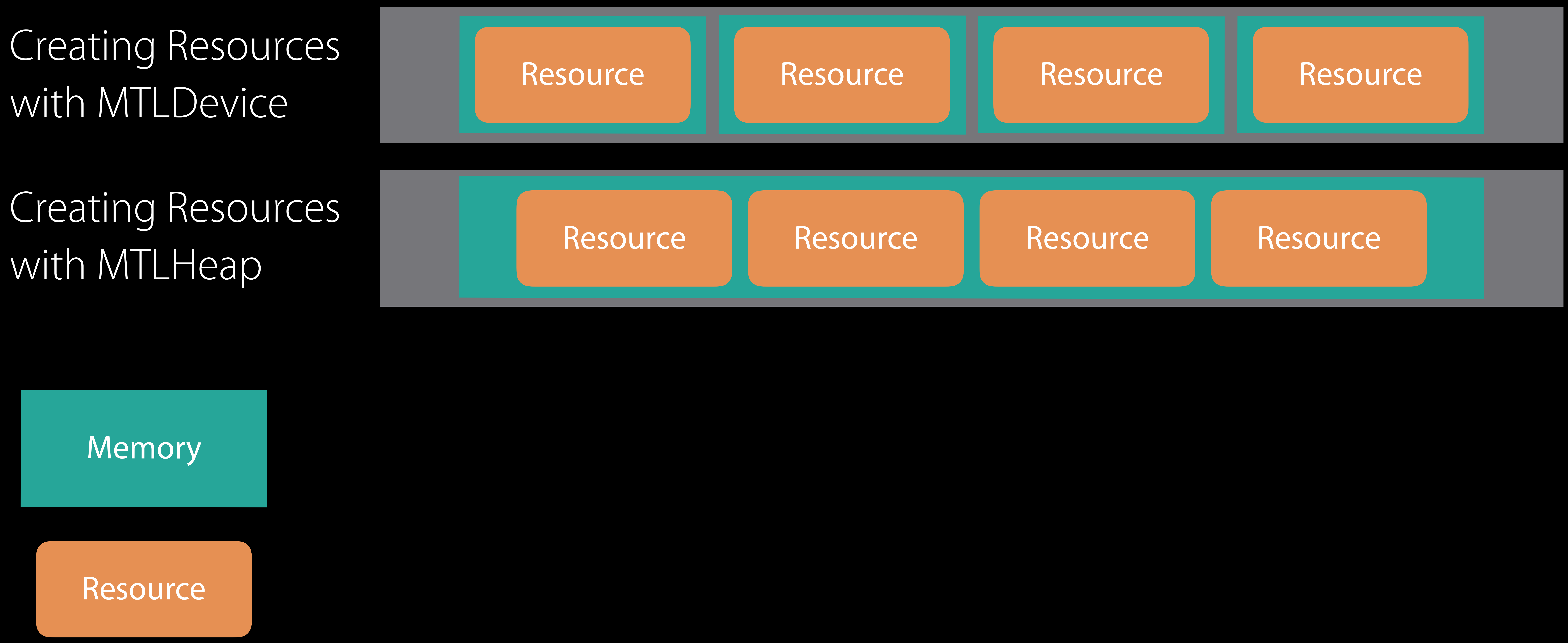
Comparing MTLDevice and MTLHeap



Resource Creation

NEW

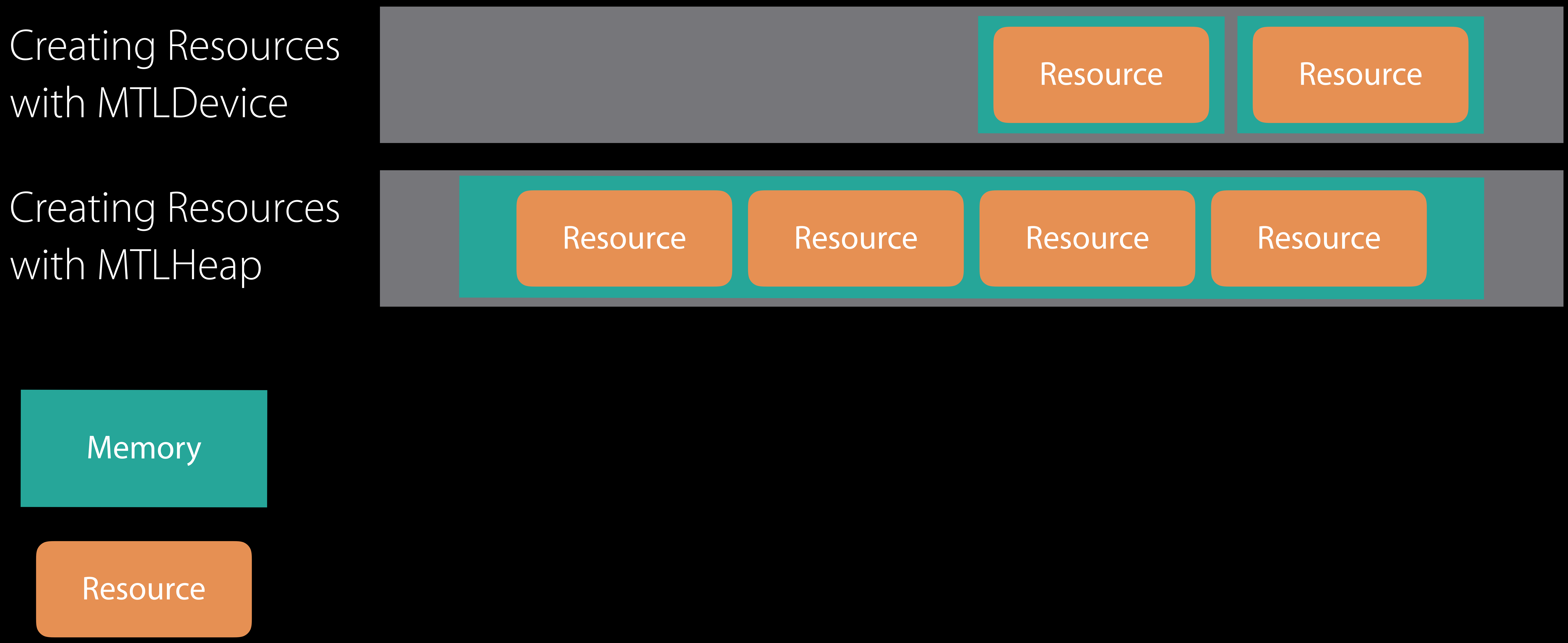
Comparing MTLDevice and MTLHeap



Resource Creation

Comparing MTLDevice and MTLHeap

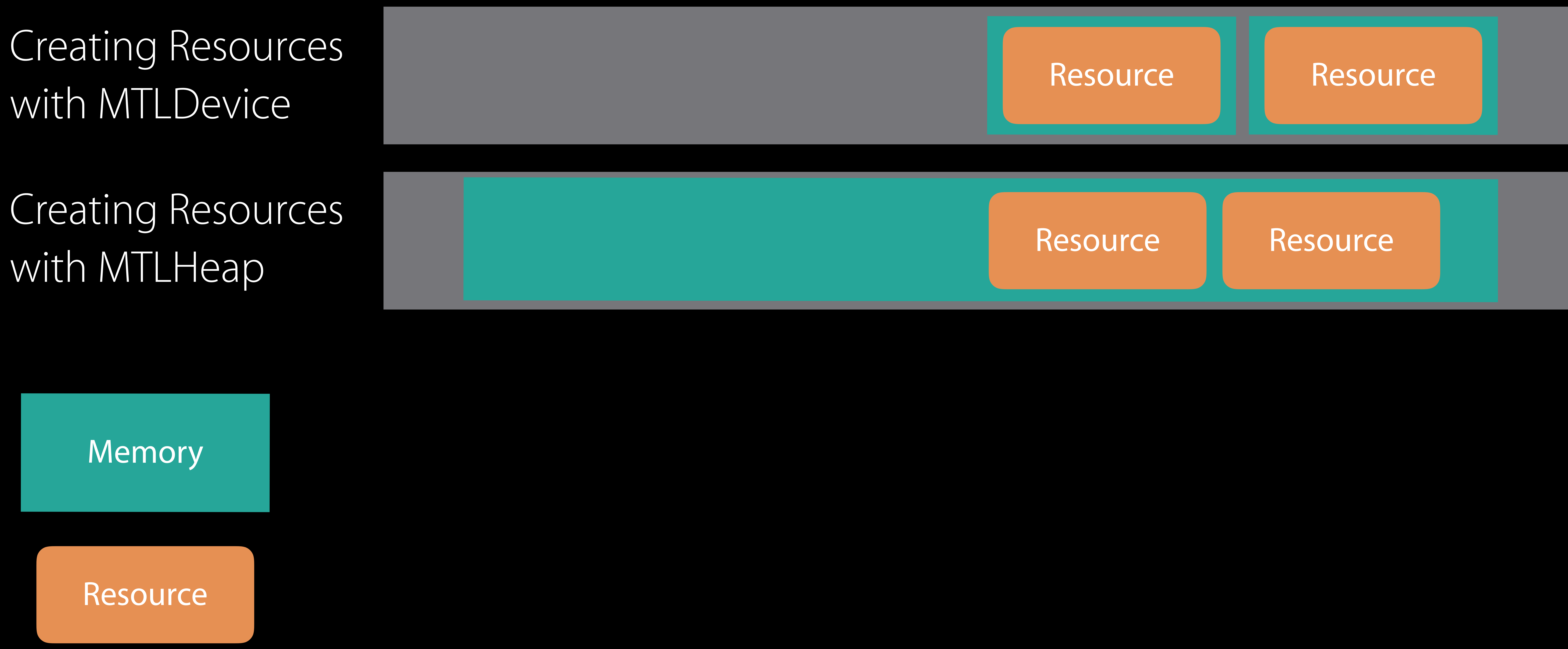
NEW



Resource Creation

Comparing MTLDevice and MTLHeap

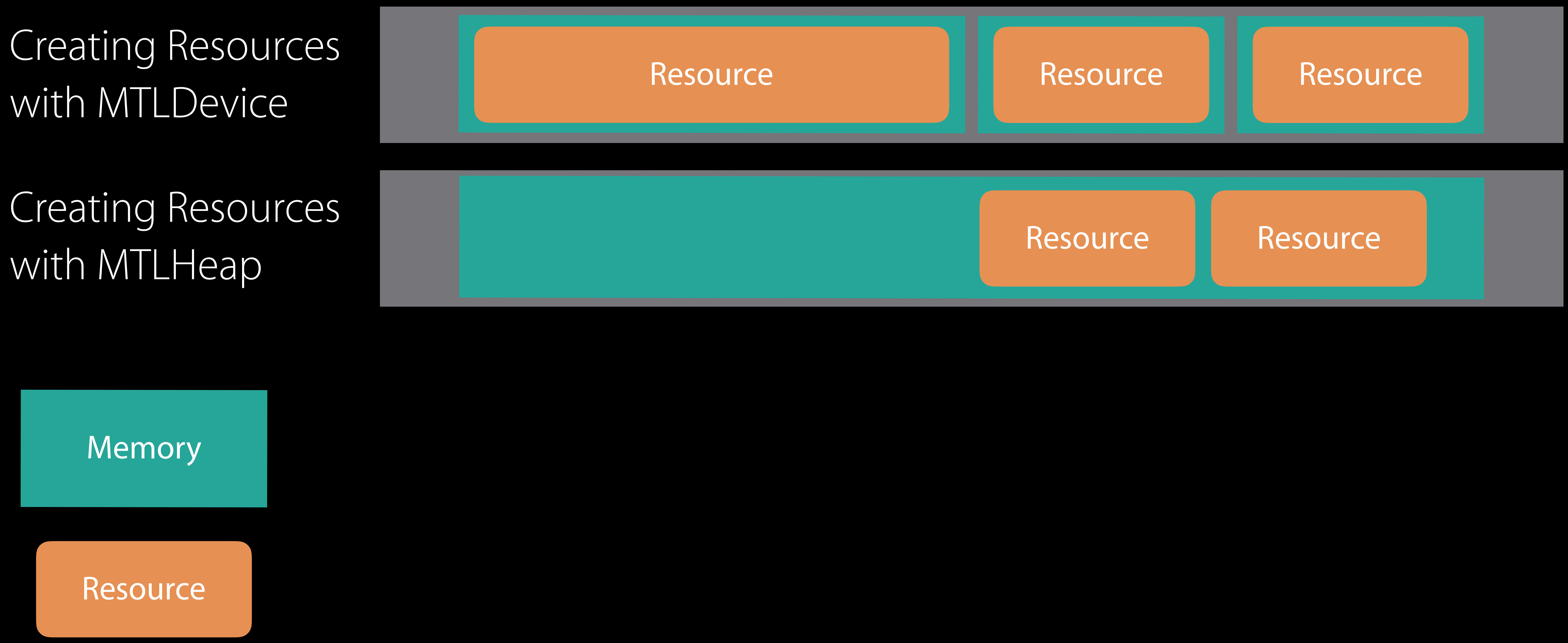
NEW



Resource Creation

NEW

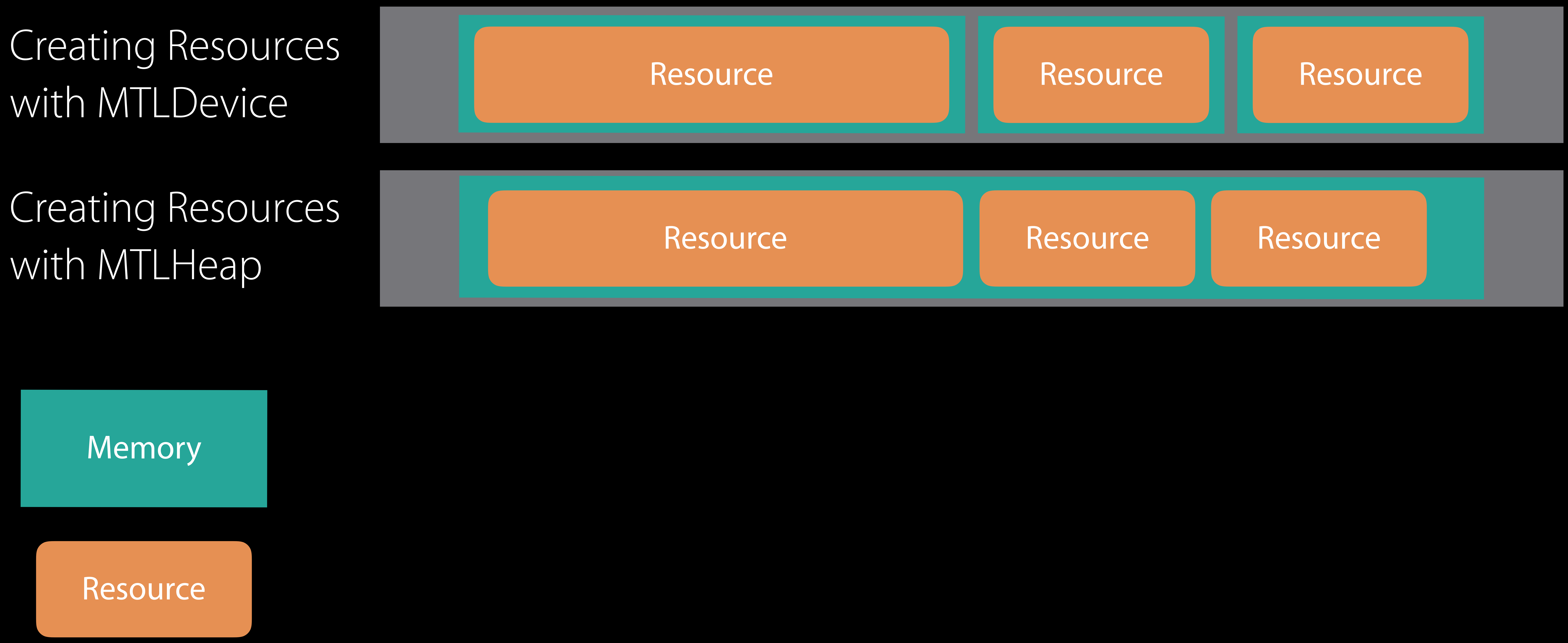
Comparing MTLDevice and MTLHeap



Resource Creation

Comparing MTLDevice and MTLHeap

NEW



```swift
// Creating Resources with MTLHeap

// Create heap descriptor, including the heap size.
let heapDesc = MTLHeapDescriptor()
heapDesc.size = heapSize

// Create the heap object ahead of time.
let heap = device.newHeap(with:heapDesc)

// Sub-allocate resources from heap - very fast!
let buffer = heap.newBuffer(withLength:bufferSize)
let texture = heap.newTexture(with:textureDescriptor)
```

```swift
// Creating Resources with MTLHeap

// Create heap descriptor, including the heap size.
let heapDesc = MTLHeapDescriptor()
heapDesc.size = heapSize

// Create the heap object ahead of time.
let heap = device.newHeap(with:heapDesc)

// Sub-allocate resources from heap – very fast!
let buffer = heap.newBuffer(withLength:bufferSize)
let texture = heap.newTexture(with:textureDescriptor)
```

```swift
// Creating Resources with MTLHeap

// Create heap descriptor, including the heap size.
let heapDesc = MTLHeapDescriptor()
heapDesc.size = heapSize

// Create the heap object ahead of time.
let heap = device.newHeap(with:heapDesc)

// Sub-allocate resources from heap - very fast!
let buffer = heap.newBuffer(withLength:bufferSize)
let texture = heap.newTexture(with:textureDescriptor)
```

```swift
// Creating Resources with MTLHeap

// Create heap descriptor, including the heap size.
let heapDesc = MTLHeapDescriptor()
heapDesc.size = heapSize

// Create the heap object ahead of time.
let heap = device.newHeap(with:heapDesc)

// Sub-allocate resources from heap - very fast!
let buffer = heap.newBuffer(withLength:bufferSize)
let texture = heap.newTexture(with:textureDescriptor)
```

Resource Sub-Allocation

Best practices

Resource Sub-Allocation

Best practices

Use Resource Heaps to create resources on a performance-critical path

Resource Sub-Allocation

Best practices

Use Resource Heaps to create resources on a performance-critical path

Resource Sub-Allocation

Best practices

Use Resource Heaps to create resources on a performance-critical path

Create multiple, size-bucketed heaps to reduce fragmentation

Resource Sub-Allocation

Best practices

Use Resource Heaps to create resources on a performance-critical path

Create multiple, size-bucketed heaps to reduce fragmentation

Resource Sub-Allocation

Best practices

Use Resource Heaps to create resources on a performance-critical path

Create multiple, size-bucketed heaps to reduce fragmentation

Use resource size and alignment queries to determine appropriate heap sizes

Resource Aliasing

Resource Aliasing

Multiple resources sharing memory

NEW

Resource Heaps allow multiple resources to occupy the same memory location

Aliasing resources together reduces total memory usage

Resource Aliasing

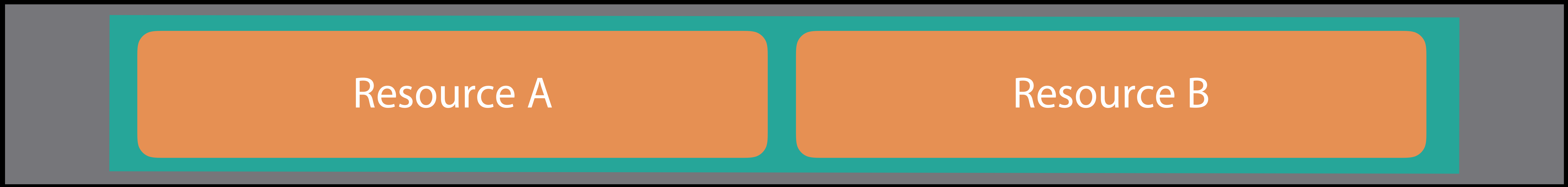
NEW

Multiple resources sharing memory

Resource Heaps allow multiple resources to occupy the same memory location

Aliasing resources together reduces total memory usage

Normal Heap
Resources



Resource Aliasing

Multiple resources sharing memory

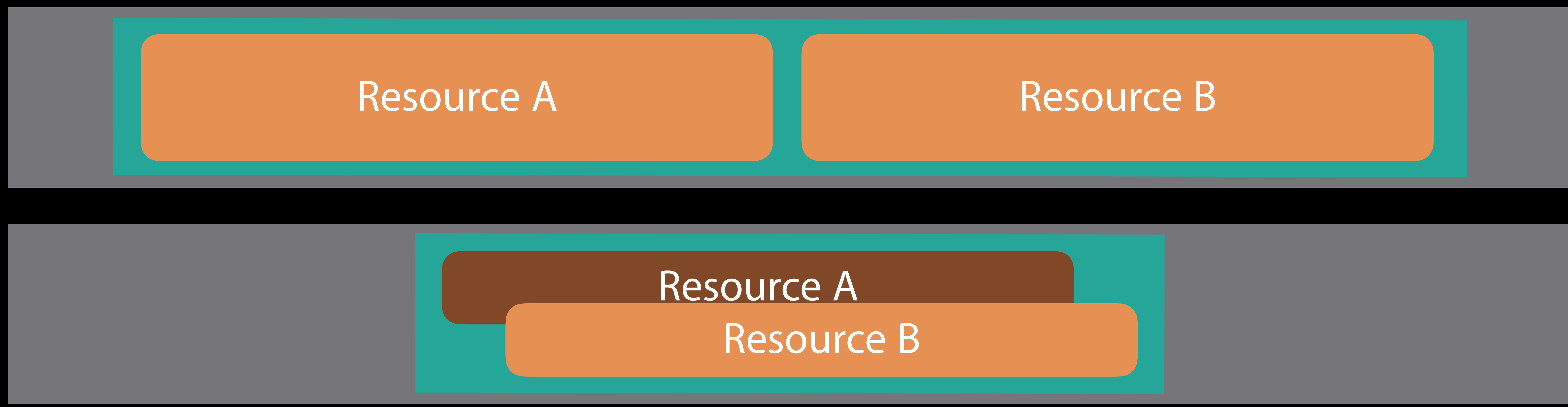
NEW

Resource Heaps allow multiple resources to occupy the same memory location

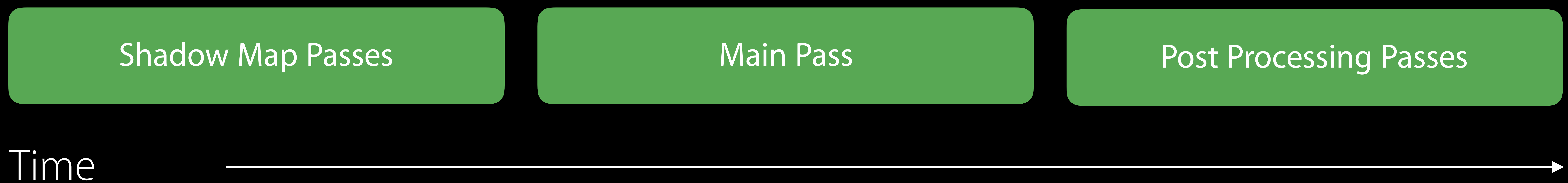
Aliasing resources together reduces total memory usage

Normal Heap Resources

Aliasing Heap Resources

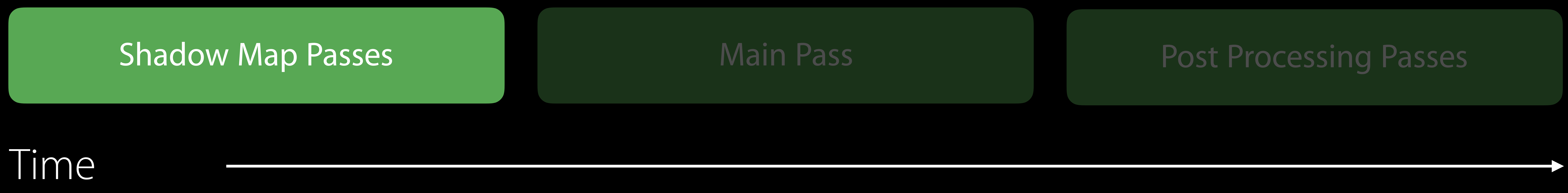


Example



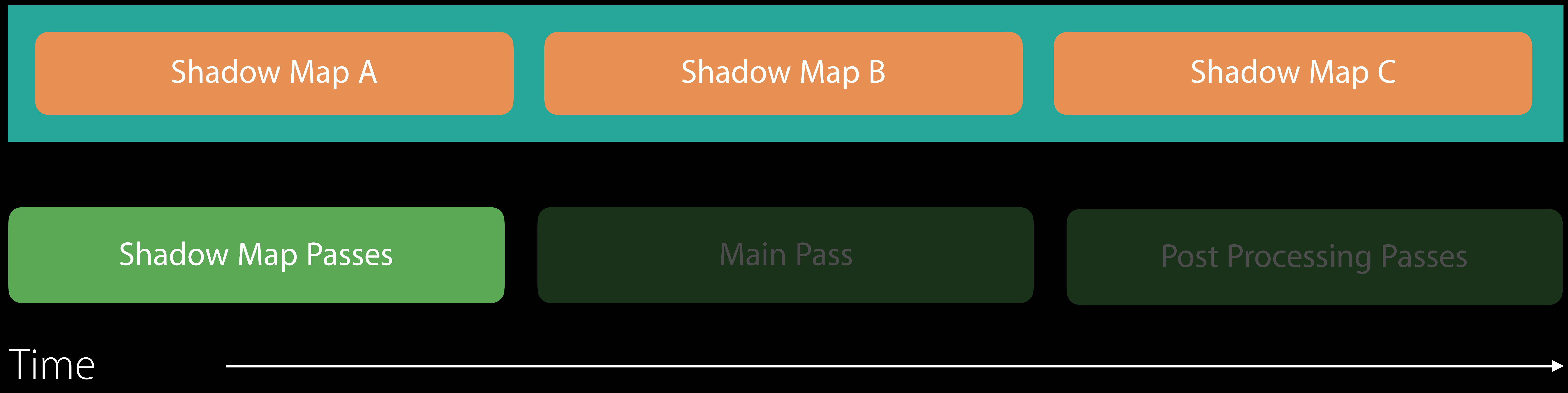
Example

Shadow maps are produced and consumed in consecutive passes



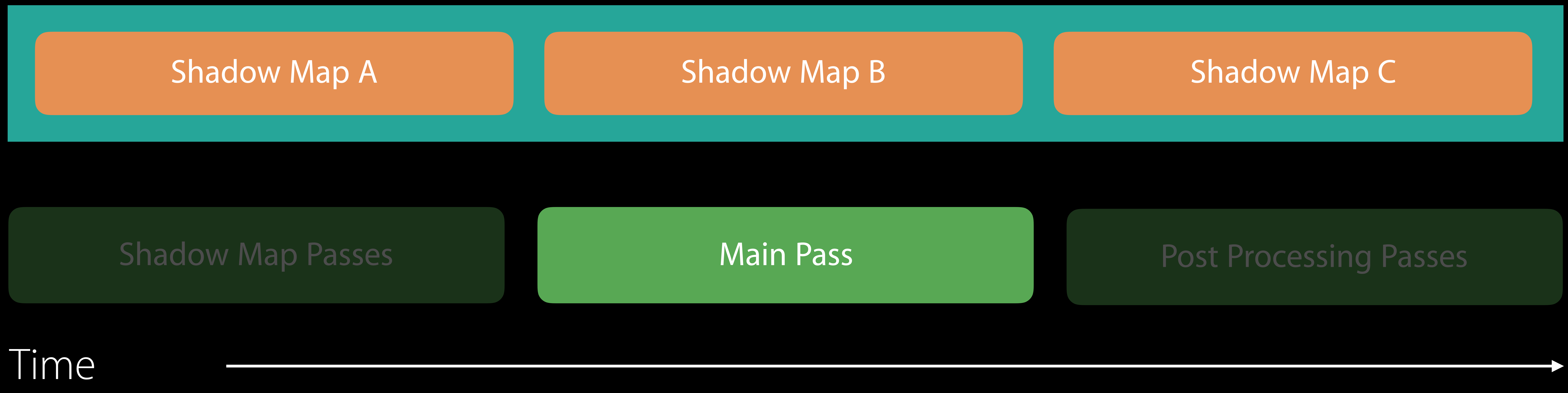
Example

Shadow maps are produced and consumed in consecutive passes



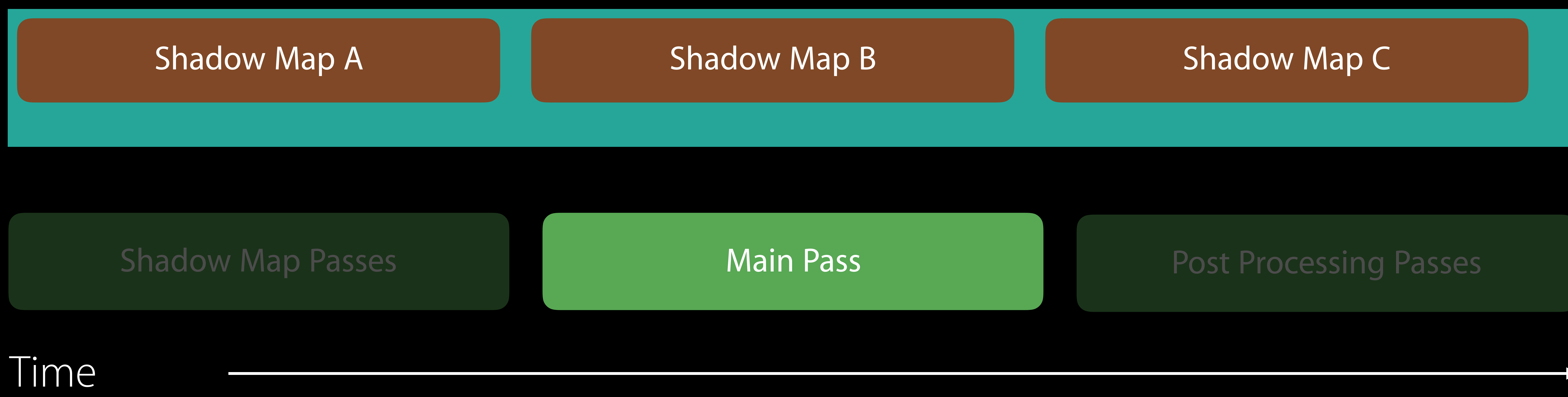
Example

Shadow maps are produced and consumed in consecutive passes



Example

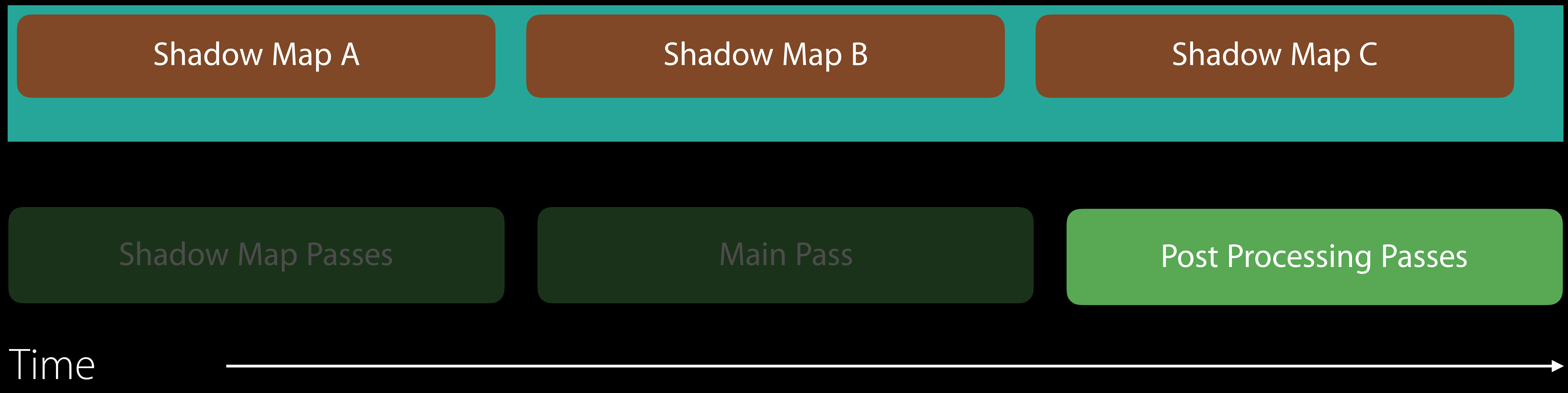
Shadow maps are produced and consumed in consecutive passes



Example

Shadow maps are produced and consumed in consecutive passes

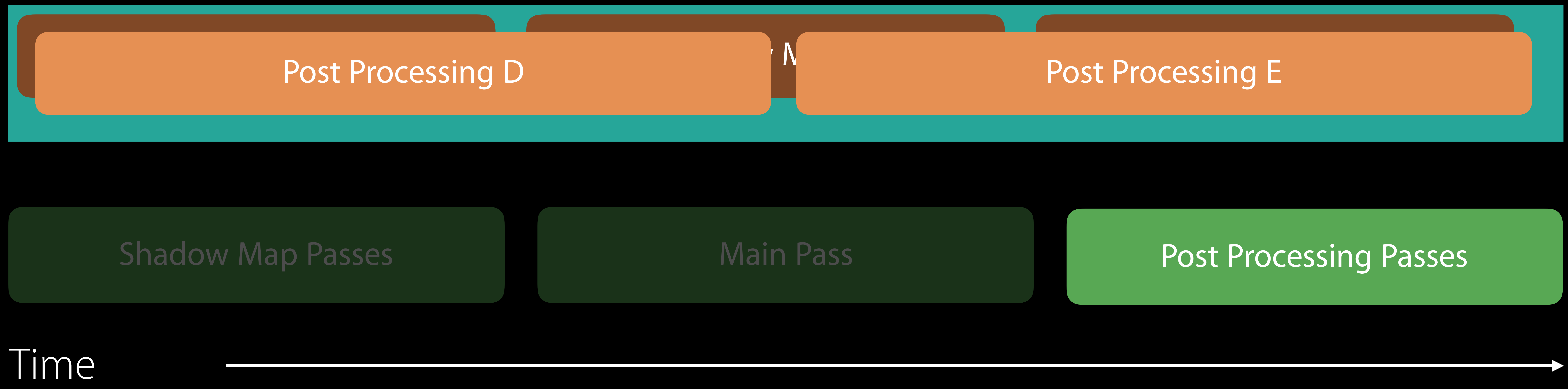
Post-processing uses temporary textures internally



Example

Shadow maps are produced and consumed in consecutive passes

Post-processing uses temporary textures internally

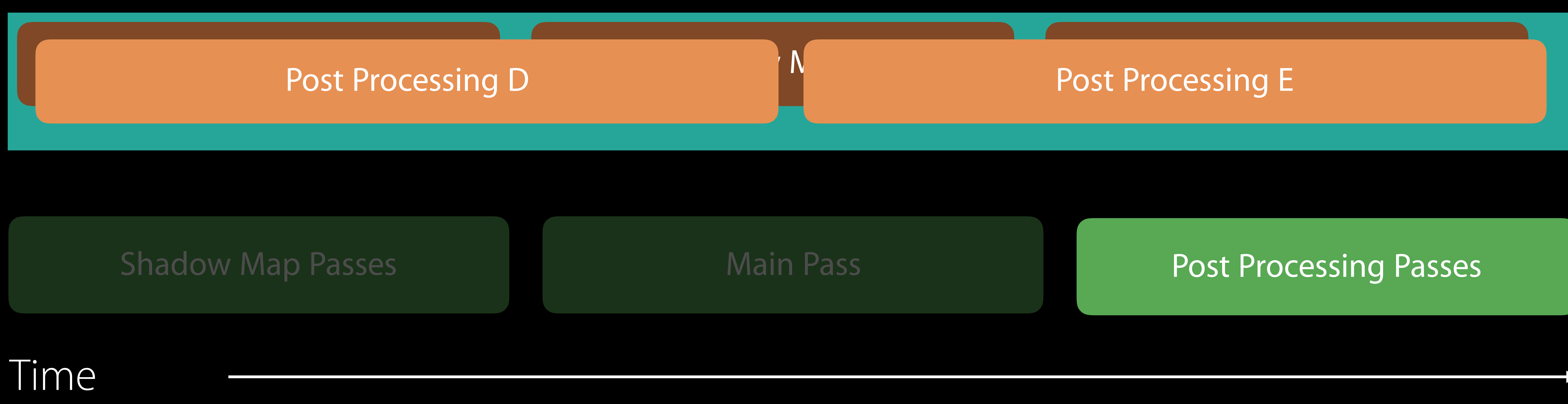


Example

Shadow maps are produced and consumed in consecutive passes

Post-processing uses temporary textures internally

Contents never used at the same time



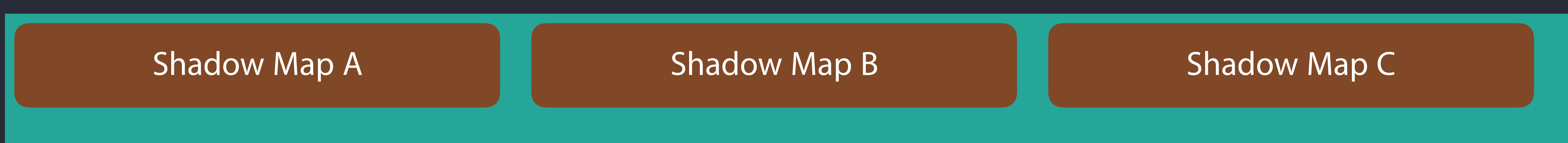
```swift
// Create Aliasing Textures

let heap = device.newHeap(with:heapDesc)
let shadowMapA = heap.newTexture(with:shadowDesc)
let shadowMapB = heap.newTexture(with:shadowDesc)
let shadowMapC = heap.newTexture(with:shadowDesc)

// Allow heap to reassign shadow map memory to new resources
shadowMapA.makeAliasable()
shadowMapB.makeAliasable()
shadowMapC.makeAliasable()

let postProcessingD = heap.newTexture(with:postProcessingDesc)
let postProcessingE = heap.newTexture(with:postProcessingDesc)
```

```swift
// Create Aliasing Textures

let heap = device.newHeap(with:heapDesc)
let shadowMapA = heap.newTexture(with:shadowDesc)
let shadowMapB = heap.newTexture(with:shadowDesc)
let shadowMapC = heap.newTexture(with:shadowDesc)

// Allow heap to reassign shadow map memory to new resources
shadowMapA.makeAliasable()
shadowMapB.makeAliasable()
shadowMapC.makeAliasable()

let postProcessingD = heap.newTexture(with:postProcessingDesc)
let postProcessingE = heap.newTexture(with:postProcessingDesc)
```

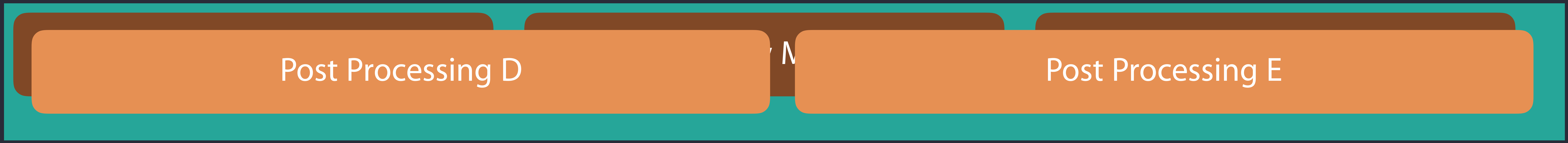
```swift
// Create Aliasing Textures

let heap = device.newHeap(with:heapDesc)
let shadowMapA = heap.newTexture(with:shadowDesc)
let shadowMapB = heap.newTexture(with:shadowDesc)
let shadowMapC = heap.newTexture(with:shadowDesc)

// Allow heap to reassign shadow map memory to new resources
shadowMapA.makeAliasable()
shadowMapB.makeAliasable()
shadowMapC.makeAliasable()

let postProcessingD = heap.newTexture(with:postProcessingDesc)
let postProcessingE = heap.newTexture(with:postProcessingDesc)
```

```swift
// Create Aliasing Textures

let heap = device.newHeap(with:heapDesc)
let shadowMapA = heap.newTexture(with:shadowDesc)
let shadowMapB = heap.newTexture(with:shadowDesc)
let shadowMapC = heap.newTexture(with:shadowDesc)

// Allow heap to reassign shadow map memory to new resources
shadowMapA.makeAliasable()
shadowMapB.makeAliasable()
shadowMapC.makeAliasable()

let postProcessingD = heap.newTexture(with:postProcessingDesc)
let postProcessingE = heap.newTexture(with:postProcessingDesc)
```

```swift
// Create Aliasing Textures

let heap = device.newHeap(with:heapDesc)
let shadowMapA = heap.newTexture(with:shadowDesc)
let shadowMapB = heap.newTexture(with:shadowDesc)
let shadowMapC = heap.newTexture(with:shadowDesc)

// Allow heap to reassign shadow map memory to new resources
shadowMapA.makeAliasable()
shadowMapB.makeAliasable()
shadowMapC.makeAliasable()

let postProcessingD = heap.newTexture(with:postProcessingDesc)
let postProcessingE = heap.newTexture(with:postProcessingDesc)
```



```swift
// Create Aliasing Textures

let heap = device.newHeap(with:heapDesc)
let shadowMapA = heap.newTexture(with:shadowDesc)
let shadowMapB = heap.newTexture(with:shadowDesc)
let shadowMapC = heap.newTexture(with:shadowDesc)

// Allow heap to reassign shadow map memory to new resources
shadowMapA.makeAliasable()
shadowMapB.makeAliasable()
shadowMapC.makeAliasable()

let postProcessingD = heap.newTexture(with:postProcessingDesc)
let postProcessingE = heap.newTexture(with:postProcessingDesc)
```



Resource Aliasing

Best practices

Resource Aliasing

Best practices

Calling sequence is important

- Create resources in the order they will be used in a frame
- Interleave with makeAliasable after resource contents are consumed

Resource Aliasing

Best practices

Calling sequence is important

- Create resources in the order they will be used in a frame
- Interleave with makeAliasable after resource contents are consumed

Keep dynamic resources in their own heap

Explicit Command Synchronization

Explicit Command Synchronization

NEW

Synchronizing heap resource updates

Explicit Command Synchronization

NEW

Synchronizing heap resource updates

Metal does not track command updates to heap resources

- Modification of resource contents
- Reinterpretation of aliasing resources

Explicit Command Synchronization

NEW

Synchronizing heap resource updates

Metal does not track command updates to heap resources

- Modification of resource contents
- Reinterpretation of aliasing resources

Explicit Command Synchronization

NEW

Synchronizing heap resource updates

Metal does not track command updates to heap resources

- Modification of resource contents
- Reinterpretation of aliasing resources

Tell Metal when heap resources are read or updated

- Necessary for correct results

Explicit Command Synchronization

NEW

Synchronizing heap resource updates

Metal does not track command updates to heap resources

- Modification of resource contents
- Reinterpretation of aliasing resources

Tell Metal when heap resources are read or updated

- Necessary for correct results

Explicit Command Synchronization

NEW

Synchronizing heap resource updates

Metal does not track command updates to heap resources

- Modification of resource contents
- Reinterpretation of aliasing resources

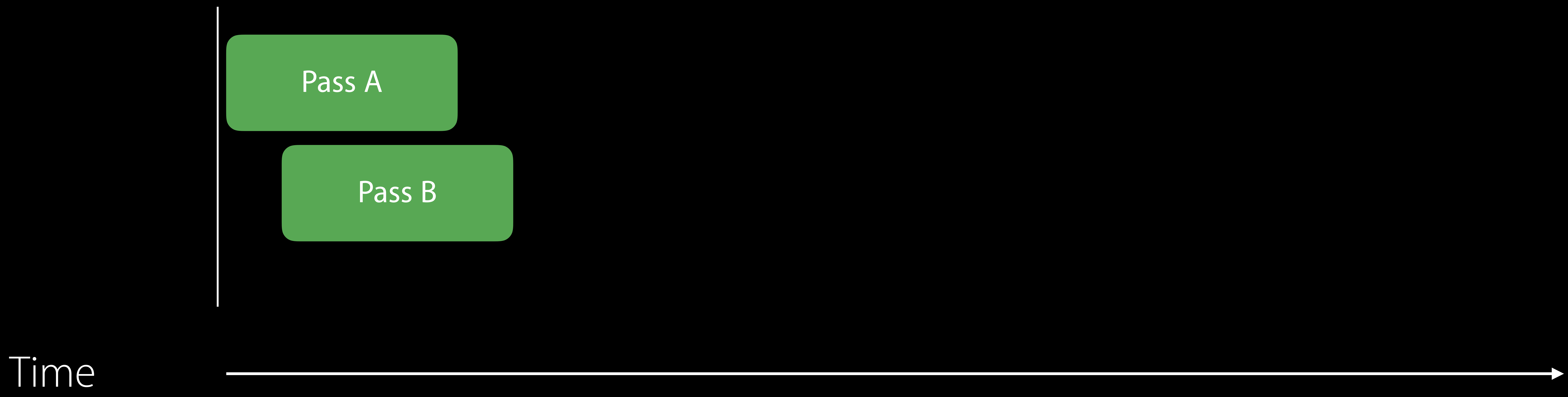
Tell Metal when heap resources are read or updated

- Necessary for correct results

Use GPU fences to explicitly communicate resource dependencies

GPU Fences

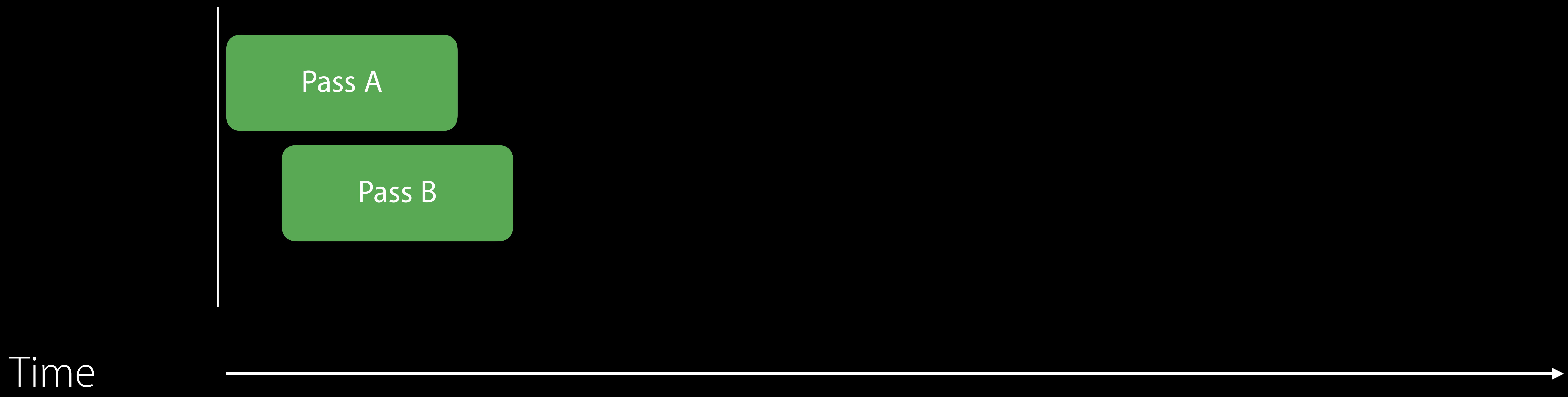
A GPU fence is a timestamp



GPU Fences

A GPU fence is a timestamp

Commands interact with fences with two operations

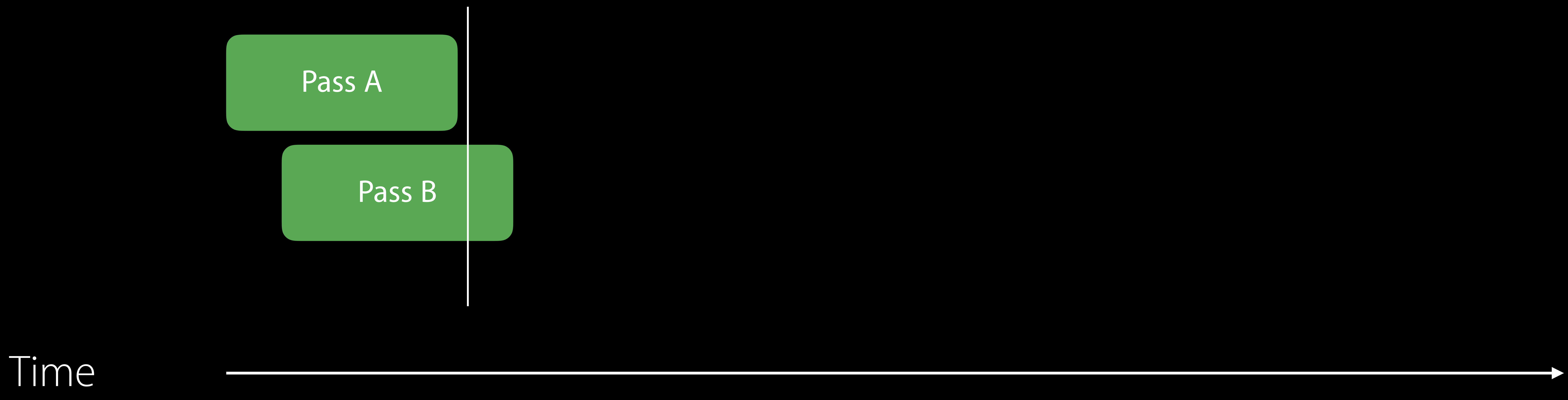


GPU Fences

A GPU fence is a timestamp

Commands interact with fences with two operations

- Update the fence with new timestamp

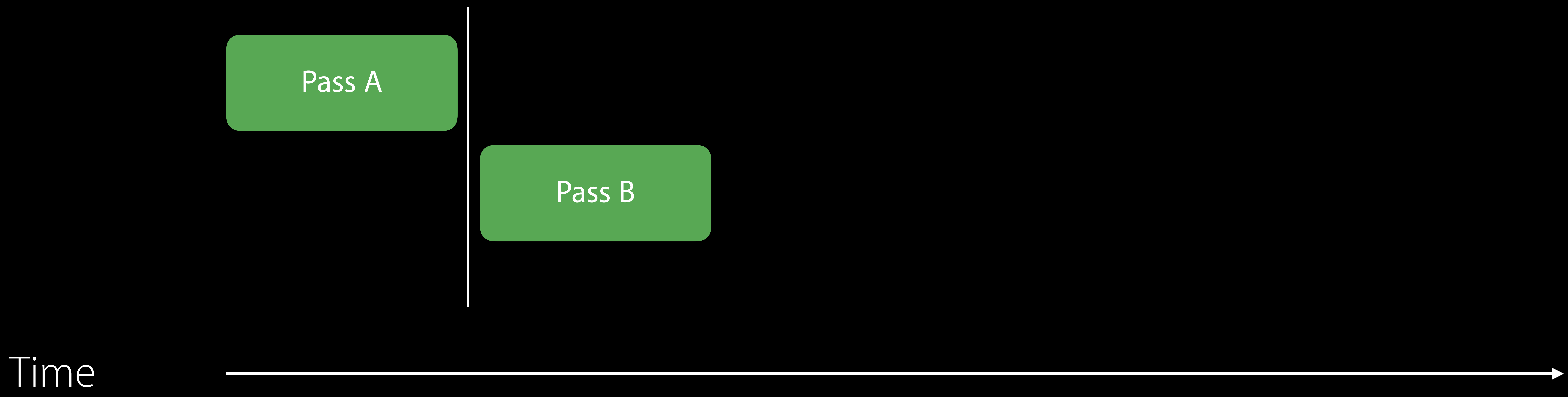


GPU Fences

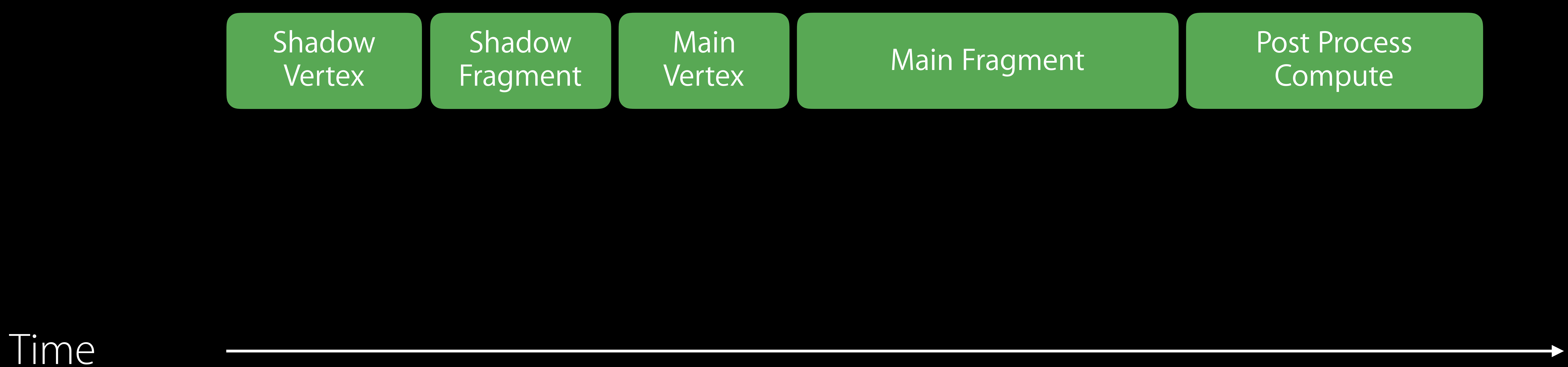
A GPU fence is a timestamp

Commands interact with fences with two operations

- Update the fence with new timestamp
- Wait for GPU to reach the current timestamp

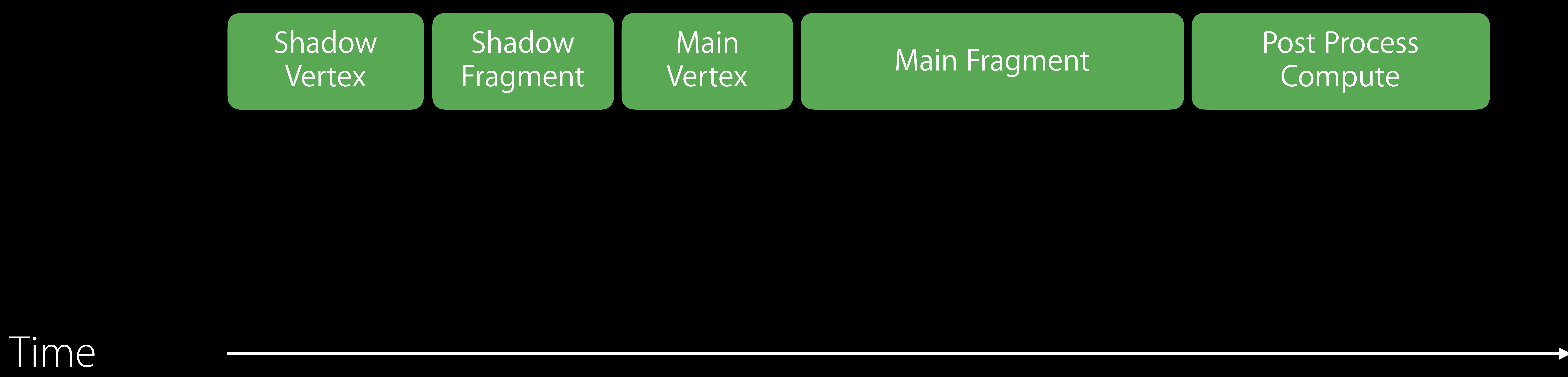


Parallel Command Execution



Parallel Command Execution

Metal commands are submitted in serial order



Parallel Command Execution

Metal commands are submitted in serial order

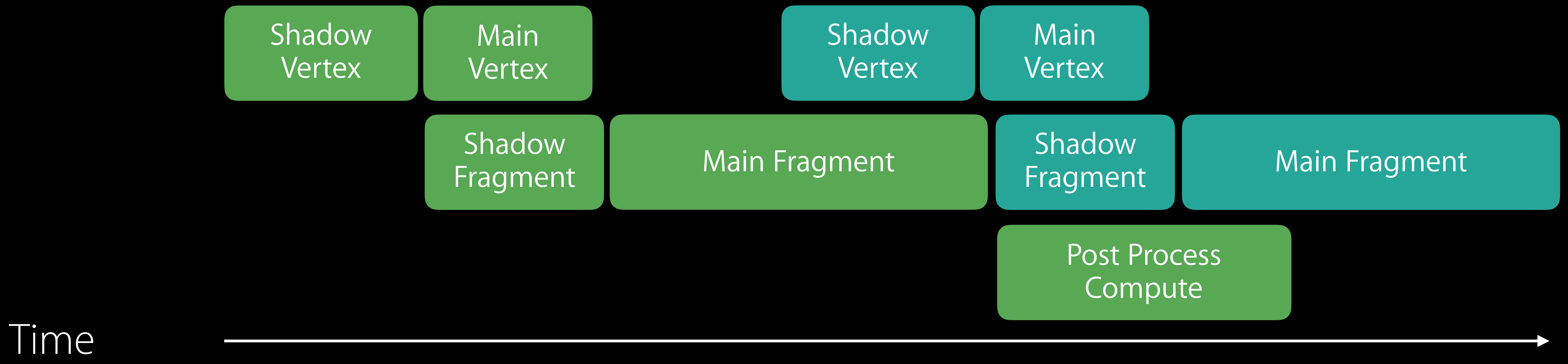
GPUs execute encoded commands in parallel

Parallel Command Execution

Metal commands are submitted in serial order

GPUs execute encoded commands in parallel

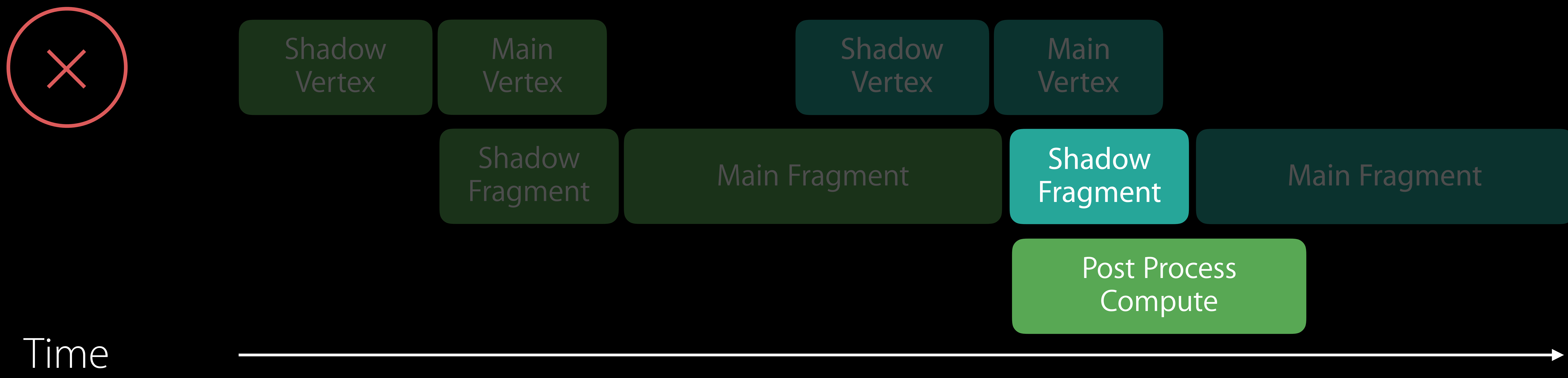
- Even across frames



Synchronizing Heap Updates

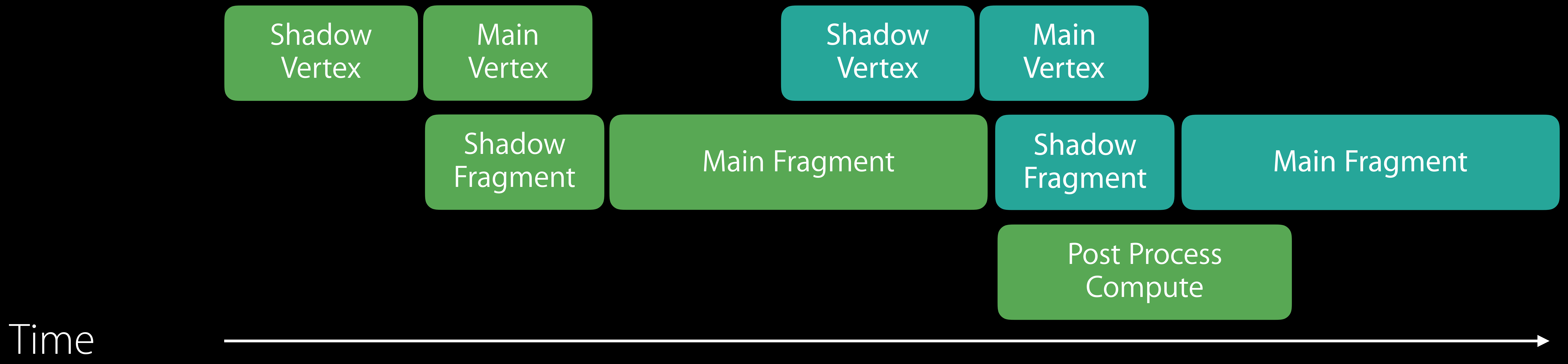
Incorrect results without synchronization

Two commands write to aliasing heap memory simultaneously



Synchronizing Heap Updates

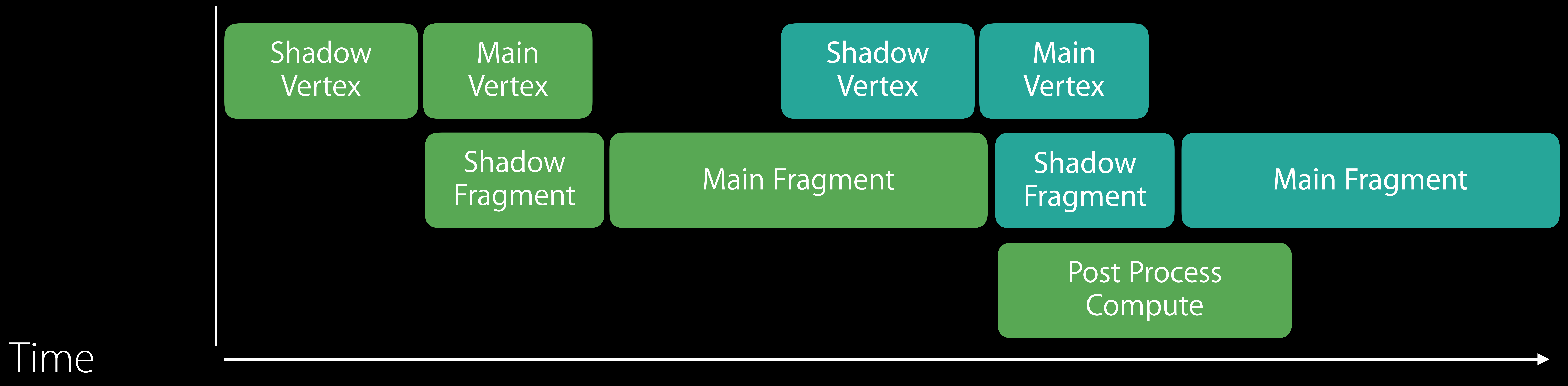
Serialize commands with a fence



Synchronizing Heap Updates

Serialize commands with a fence

Use a fence to serialize access to the aliasing heap resources

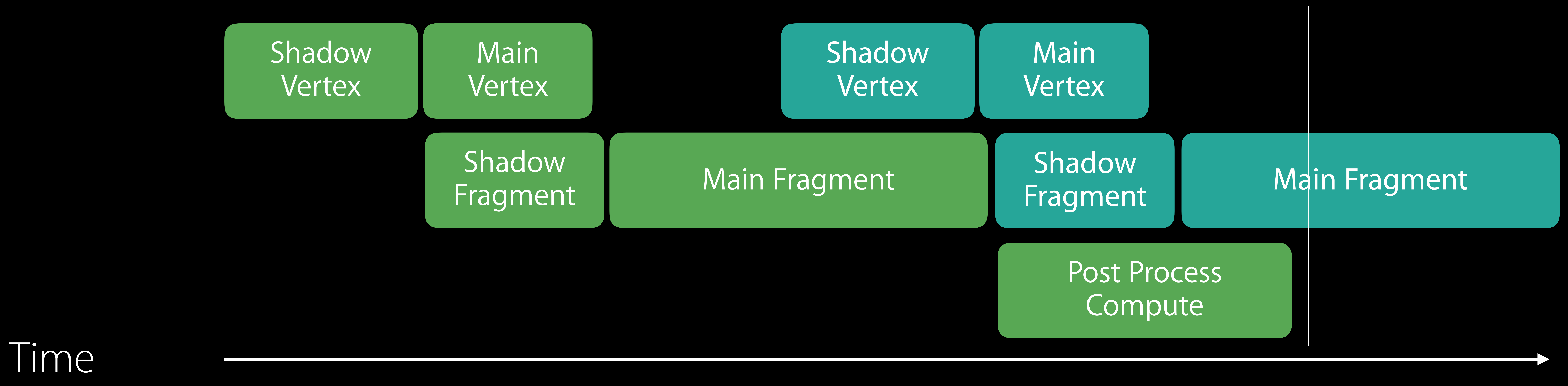


Synchronizing Heap Updates

Serialize commands with a fence

Use a fence to serialize access to the aliasing heap resources

- Post Process command updates the fence

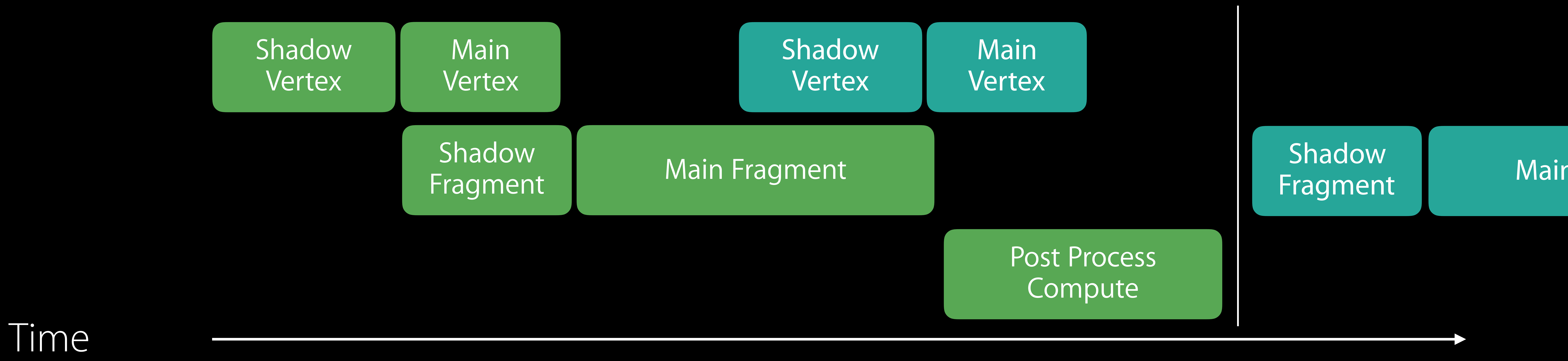


Synchronizing Heap Updates

Serialize commands with a fence

Use a fence to serialize access to the aliasing heap resources

- Post Process command updates the fence
- Shadow command waits on the fence

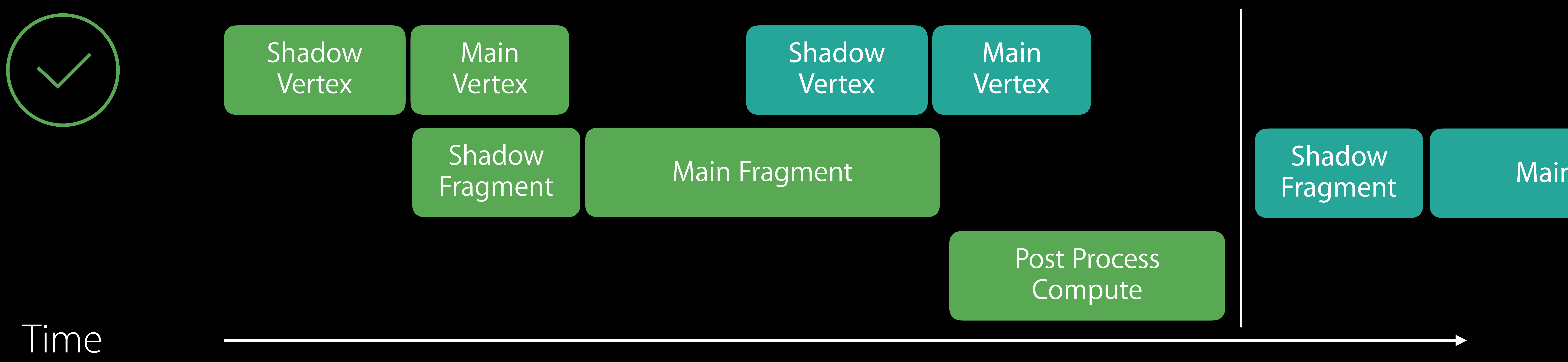


Synchronizing Heap Updates

Serialize commands with a fence

Use a fence to serialize access to the aliasing heap resources

- Post Process command updates the fence
- Shadow command waits on the fence



```swift
// Using Fences To Synchronize Heap Updates Across Commands

let shadowPostProcessingFence = device.newFence()

// Create post-processing encoder for frame A
let postProcessingEnc = commandBufA.computeCommandEncoder()
dispatchPostProcessingFilters(postProcessingEnc)
// Tell GPU to update fence when post-processing is complete
computeEnc.update(shadowPostProcessingFence)
computeEnc.endEncoding()

// Create shadow encoder for frame B
let shadowEnc = commandBufB.renderCommandEncoder(with:passDesc)
// Wait until compute encoder from frame A is complete
shadowEnc.wait(for:shadowPostProcessingFence, beforeStages:.fragment)
renderShadows(shadowEnc);
shadowEnc.endEncoding()
```

```swift
// Using Fences To Synchronize Heap Updates Across Commands

let shadowPostProcessingFence = device.newFence()

// Create post-processing encoder for frame A
let postProcessingEnc = commandBufA.computeCommandEncoder()
dispatchPostProcessingFilters(postProcessingEnc)
// Tell GPU to update fence when post-processing is complete
computeEnc.update(shadowPostProcessingFence)
computeEnc.endEncoding()

// Create shadow encoder for frame B
let shadowEnc = commandBufB.renderCommandEncoder(with:passDesc)
// Wait until compute encoder from frame A is complete
shadowEnc.wait(for:shadowPostProcessingFence, beforeStages:.fragment)
renderShadows(shadowEnc);
shadowEnc.endEncoding()
```

```swift
// Using Fences To Synchronize Heap Updates Across Commands

let shadowPostProcessingFence = device.newFence()

// Create post-processing encoder for frame A
let postProcessingEnc = commandBufA.computeCommandEncoder()
dispatchPostProcessingFilters(postProcessingEnc)
// Tell GPU to update fence when post-processing is complete
computeEnc.update(shadowPostProcessingFence)
computeEnc.endEncoding()

// Create shadow encoder for frame B
let shadowEnc = commandBufB.renderCommandEncoder(with:passDesc)
// Wait until compute encoder from frame A is complete
shadowEnc.wait(for:shadowPostProcessingFence, beforeStages:.fragment)
renderShadows(shadowEnc);
shadowEnc.endEncoding()
```

```swift
// Using Fences To Synchronize Heap Updates Across Commands

let shadowPostProcessingFence = device.newFence()

// Create post-processing encoder for frame A
let postProcessingEnc = commandBufA.computeCommandEncoder()
dispatchPostProcessingFilters(postProcessingEnc)
// Tell GPU to update fence when post-processing is complete
computeEnc.update(shadowPostProcessingFence)
computeEnc.endEncoding()

// Create shadow encoder for frame B
let shadowEnc = commandBufB.renderCommandEncoder(with:passDesc)
// Wait until compute encoder from frame A is complete
shadowEnc.wait(for:shadowPostProcessingFence, beforeStages:.fragment)
renderShadows(shadowEnc);
shadowEnc.endEncoding()
```

```swift
// Using Fences To Synchronize Heap Updates Across Commands

let shadowPostProcessingFence = device.newFence()

// Create post-processing encoder for frame A
let postProcessingEnc = commandBufA.computeCommandEncoder()
dispatchPostProcessingFilters(postProcessingEnc)
// Tell GPU to update fence when post-processing is complete
computeEnc.update(shadowPostProcessingFence)
computeEnc.endEncoding()

// Create shadow encoder for frame B
let shadowEnc = commandBufB.renderCommandEncoder(with:passDesc)
// Wait until compute encoder from frame A is complete
shadowEnc.wait(for:shadowPostProcessingFence, beforeStages:.fragment)
renderShadows(shadowEnc);
shadowEnc.endEncoding()
```

```
// Using Fences To Synchronize Heap Updates Across Commands

let shadowPostProcessingFence = device.newFence()

// Create post-processing encoder for frame A
let postProcessingEnc = commandBufA.computeCommandEncoder()
dispatchPostProcessingFilters(postProcessingEnc)
// Tell GPU to update fence when post-processing is complete
computeEnc.update(shadowPostProcessingFence)
computeEnc.endEncoding()

// Create shadow encoder for frame B
let shadowEnc = commandBufB.renderCommandEncoder(with:passDesc)
// Wait until compute encoder from frame A is complete
shadowEnc.wait(for:shadowPostProcessingFence, beforeStages:.fragment)
renderShadows(shadowEnc);
shadowEnc.endEncoding()
```

```swift
// Using Fences To Synchronize Heap Updates Across Commands

let shadowPostProcessingFence = device.newFence()

// Create post-processing encoder for frame A
let postProcessingEnc = commandBufA.computeCommandEncoder()
dispatchPostProcessingFilters(postProcessingEnc)
// Tell GPU to update fence when post-processing is complete
computeEnc.update(shadowPostProcessingFence)
computeEnc.endEncoding()

// Create shadow encoder for frame B
let shadowEnc = commandBufB.renderCommandEncoder(with:passDesc)
// Wait until compute encoder from frame A is complete
shadowEnc.wait(for:shadowPostProcessingFence, beforeStages:.fragment)
renderShadows(shadowEnc);
shadowEnc.endEncoding()
```

Explicit Command Synchronization

Best practices

Express dependencies at appropriate granularity

- Do not track static textures
- Track groups of resources with a single fence

Resource Heaps

Summary

Create resources faster with suballocation

Use memory more efficiently with resource aliasing

Synchronize heap updates across commands with GPU Fences

Memoryless Render Targets

Same Rendering, Less Memory

Memoryless Render Targets

Textures without storage

NEW

Memoryless Render Targets

NEW

Textures without storage

Memoryless render targets are not backed by system memory

Memoryless Render Targets

Textures without storage

NEW

Memoryless render targets are not backed by system memory

Memoryless Render Targets

NEW

Textures without storage

Memoryless render targets are not backed by system memory

Use for render pass attachments that are not stored

Memoryless Render Targets

Textures without storage

NEW

Memoryless render targets are not backed by system memory

Use for render pass attachments that are not stored

Memoryless Render Targets

Textures without storage

NEW

Memoryless render targets are not backed by system memory

Use for render pass attachments that are not stored

Create texture with new storage mode

- MTLStorageModeMemoryless

Memoryless Render Targets

NEW

Textures without storage

Memoryless render targets are not backed by system memory

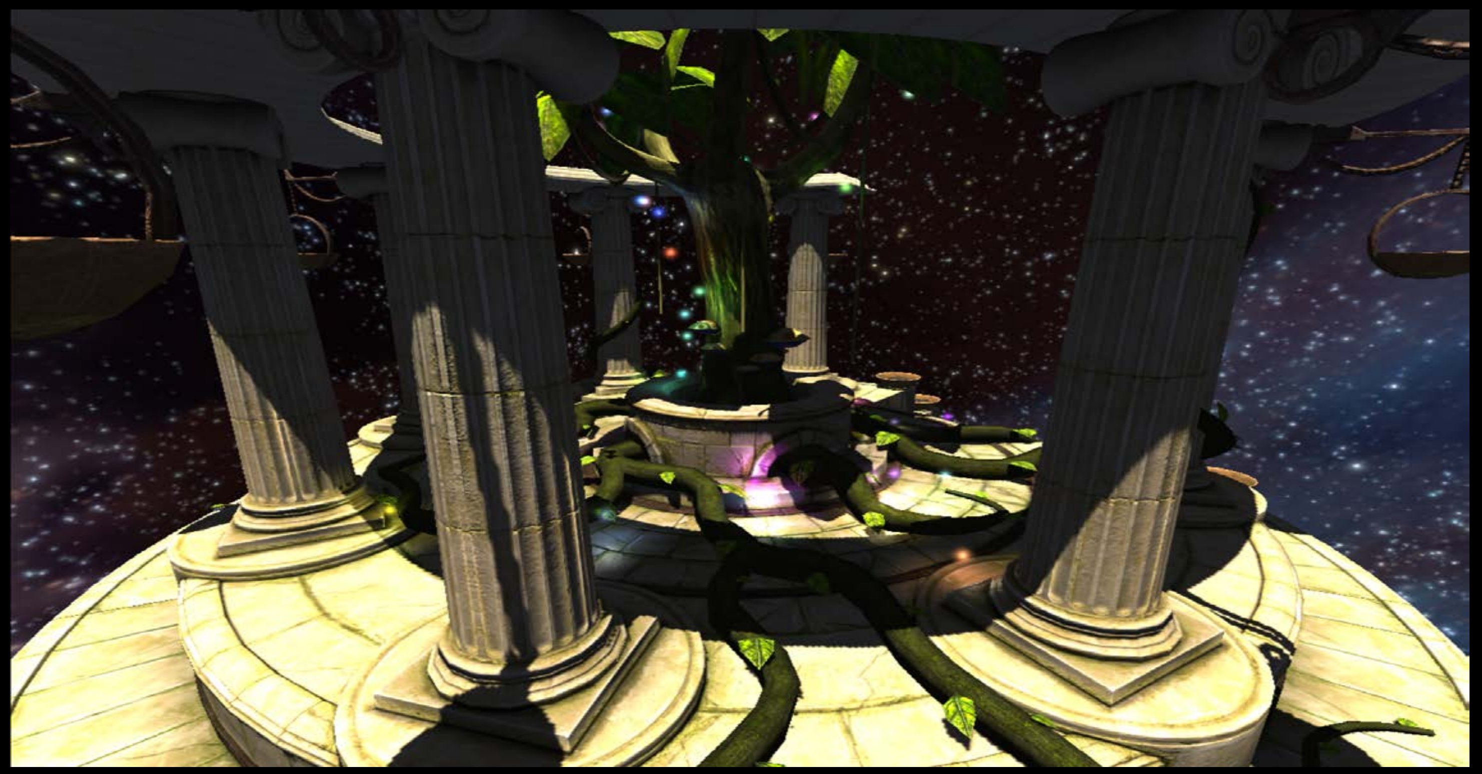
Use for render pass attachments that are not stored

Create texture with new storage mode

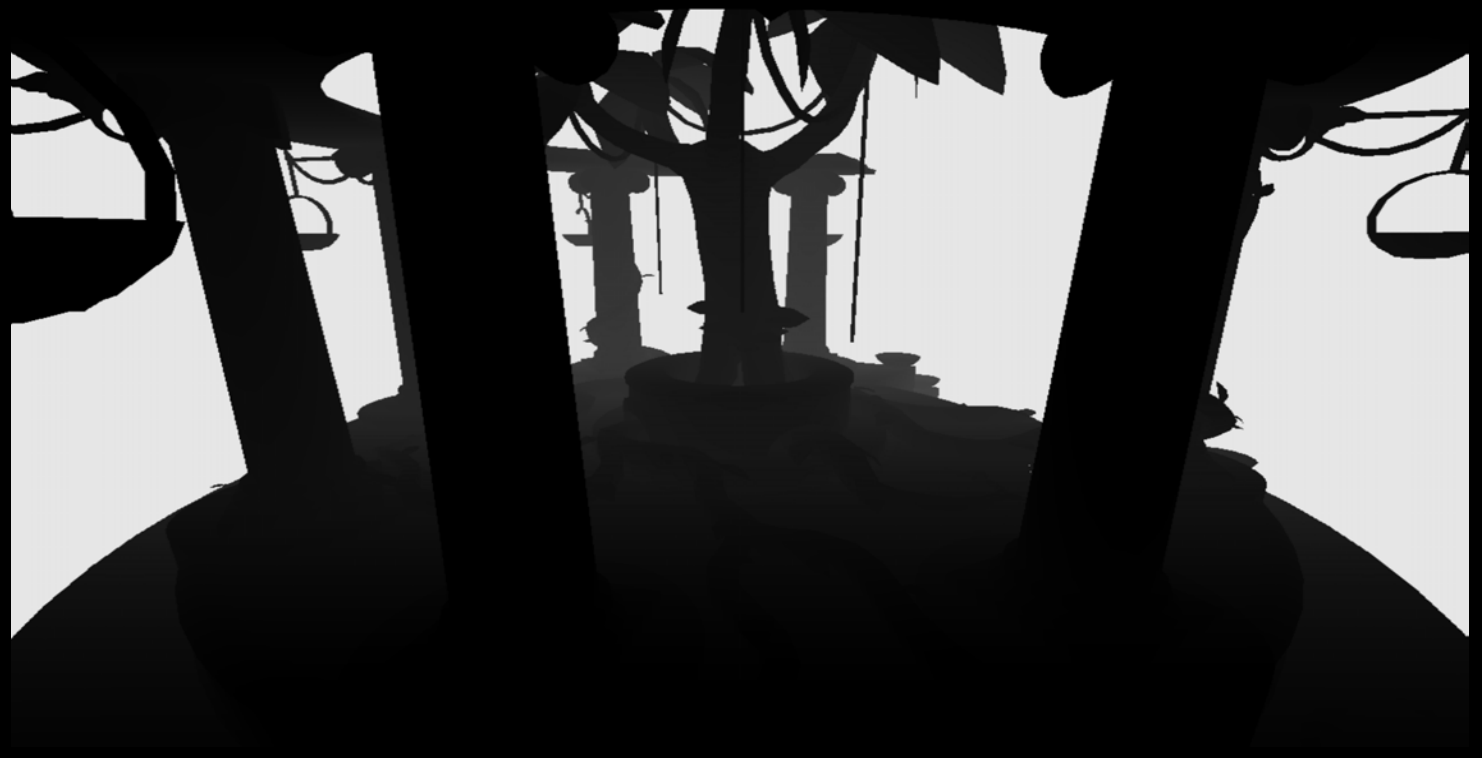
- MTLStorageModeMemoryless

iOS and tvOS only

Tile-Based Rendering



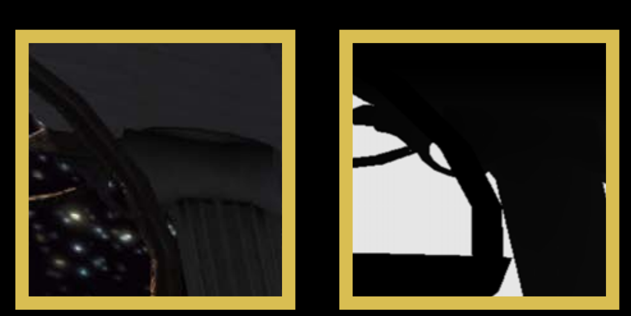
Color Attachment Texture



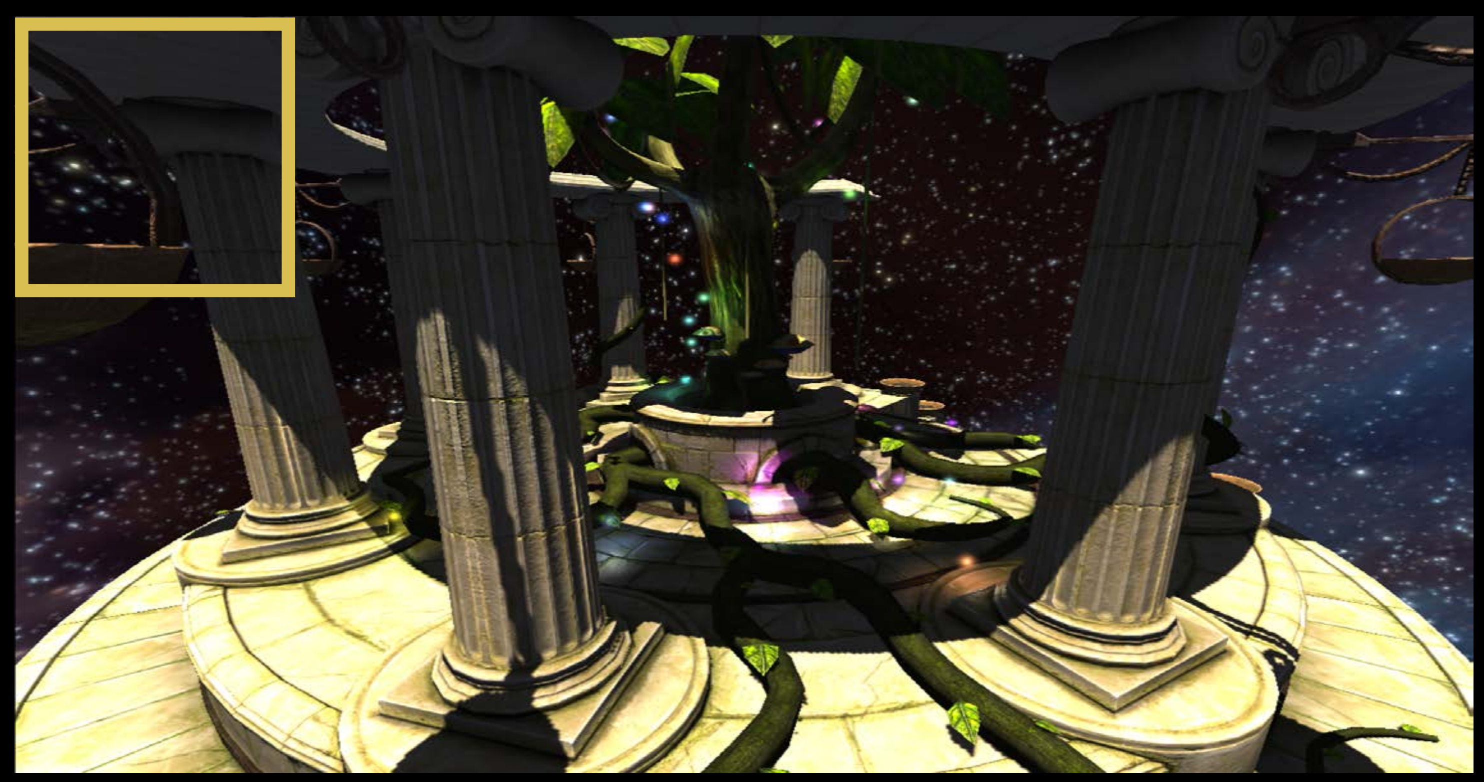
Depth Attachment Texture

Tile-Based Rendering

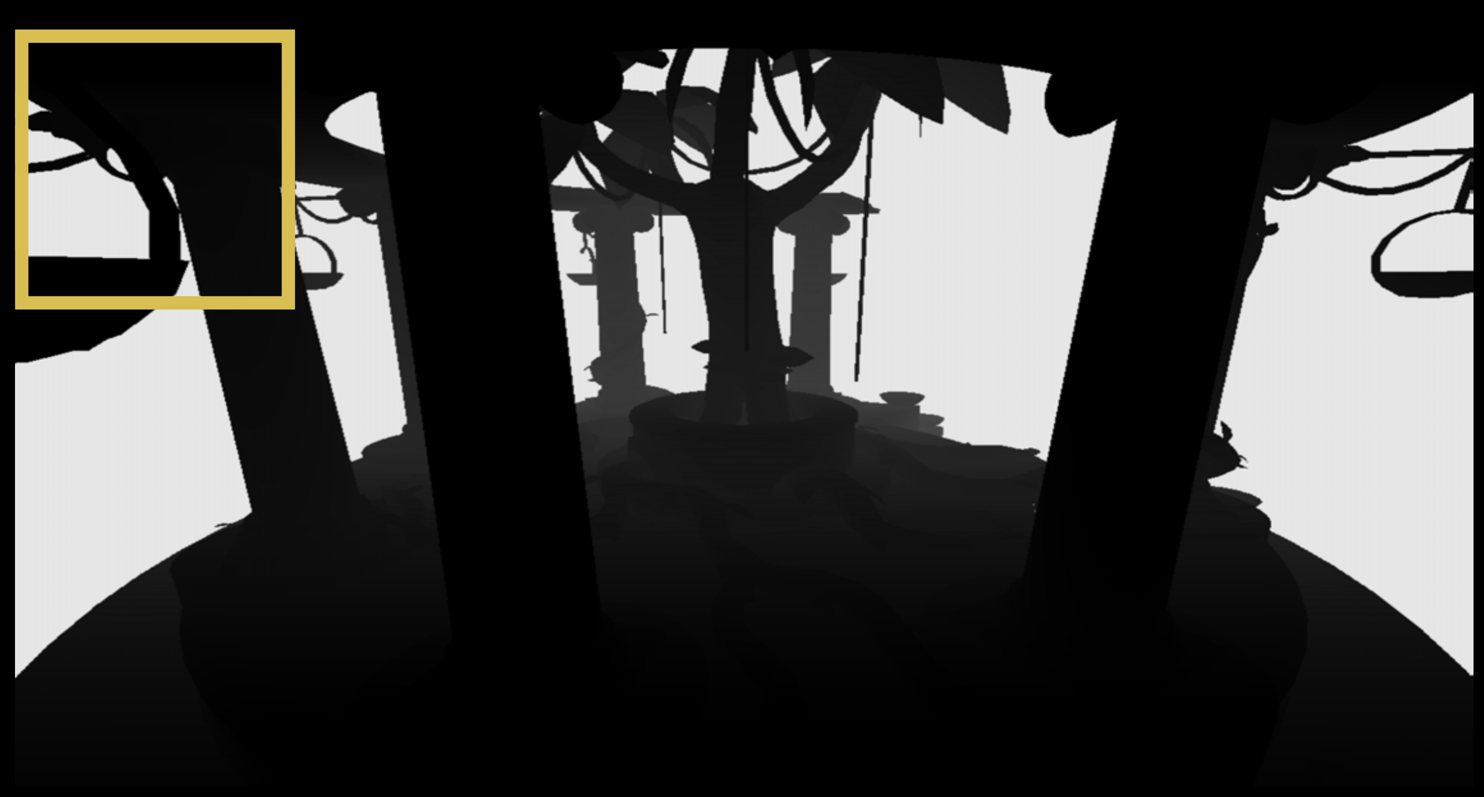
A7 and later GPUs render to fast GPU tile storage, one tile at a time



GPU Tile Storage



Color Attachment Texture

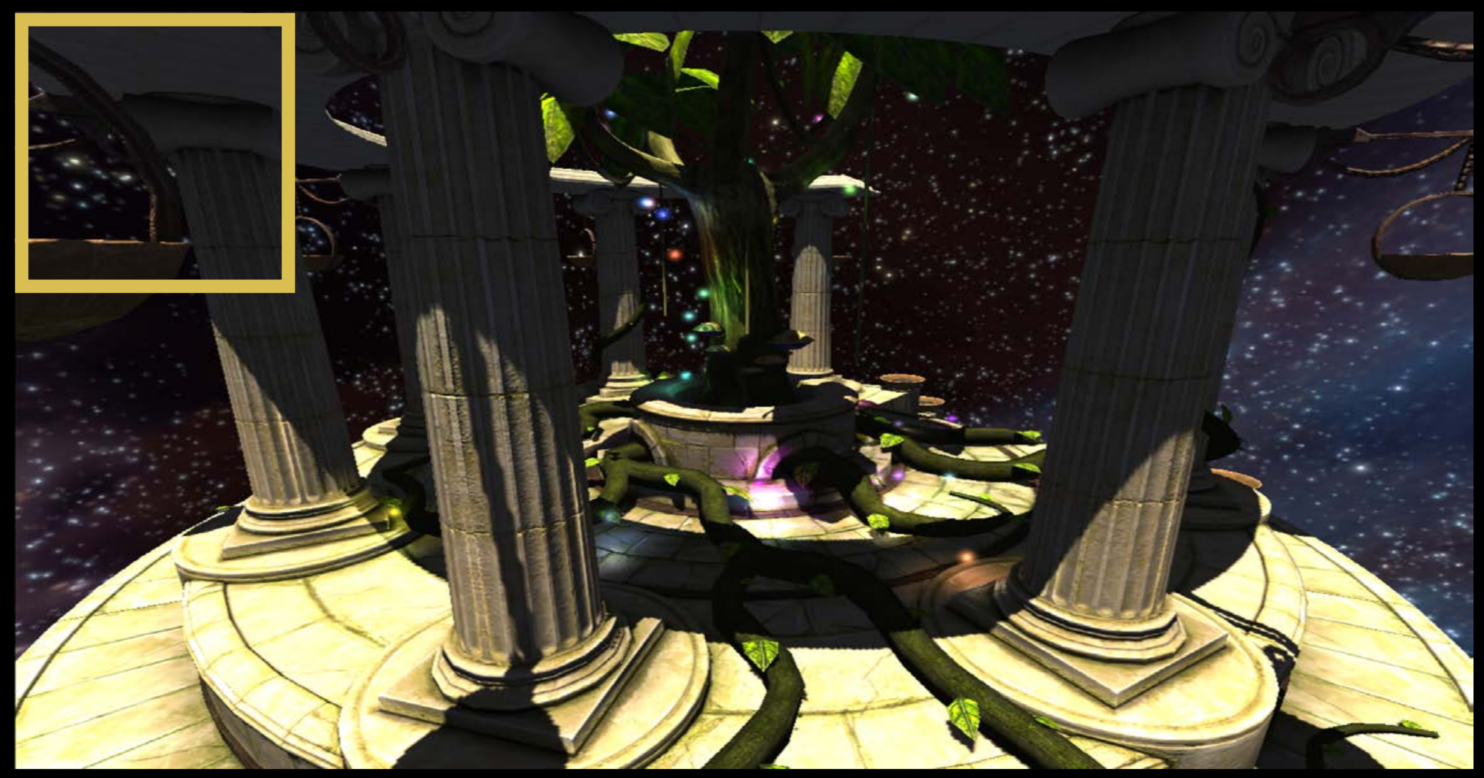


Depth Attachment Texture

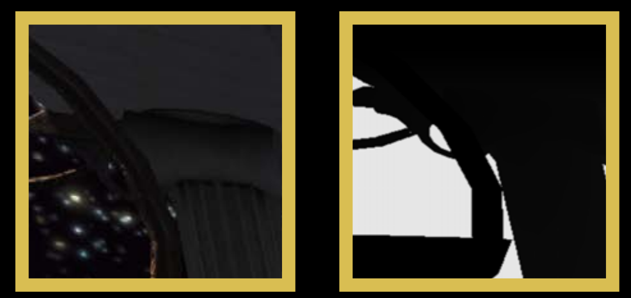
Tile-Based Rendering

A7 and later GPUs render to fast GPU tile storage, one tile at a time

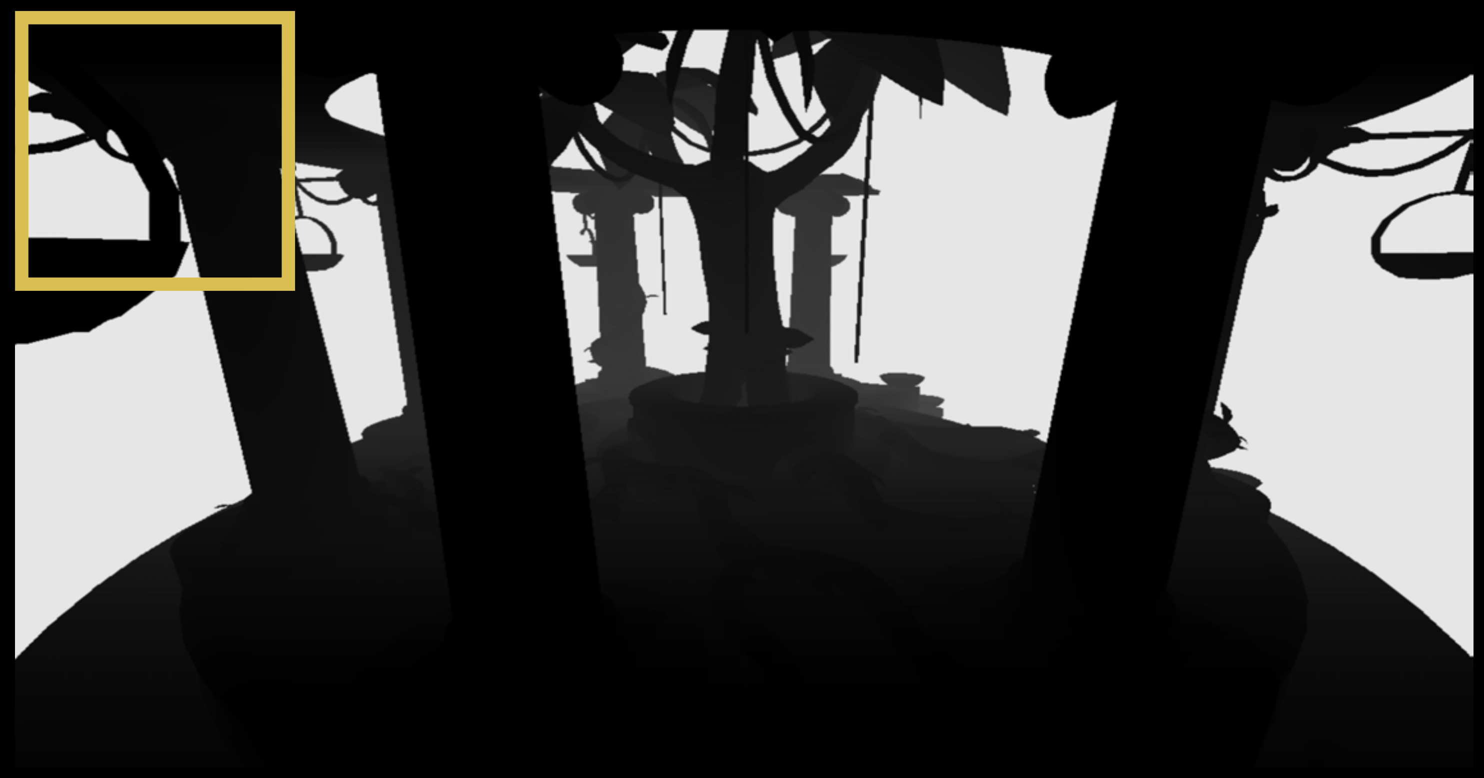
Store actions control whether to copy results to system memory



Color Attachment Texture



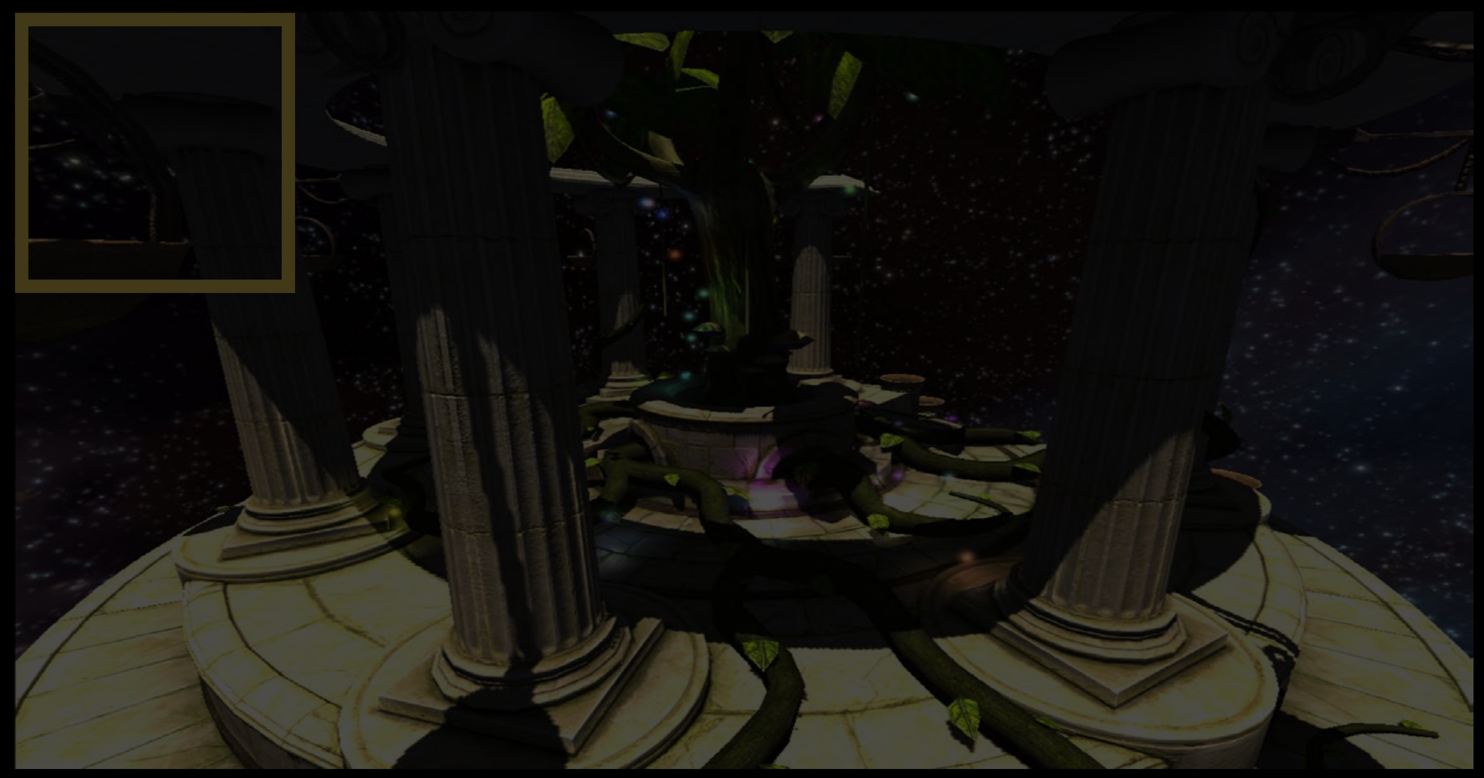
GPU Tile Storage



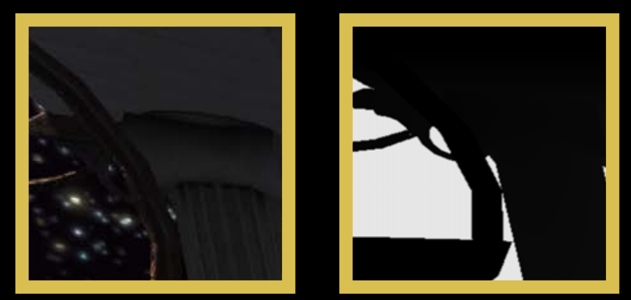
Depth Attachment Texture

Use Case

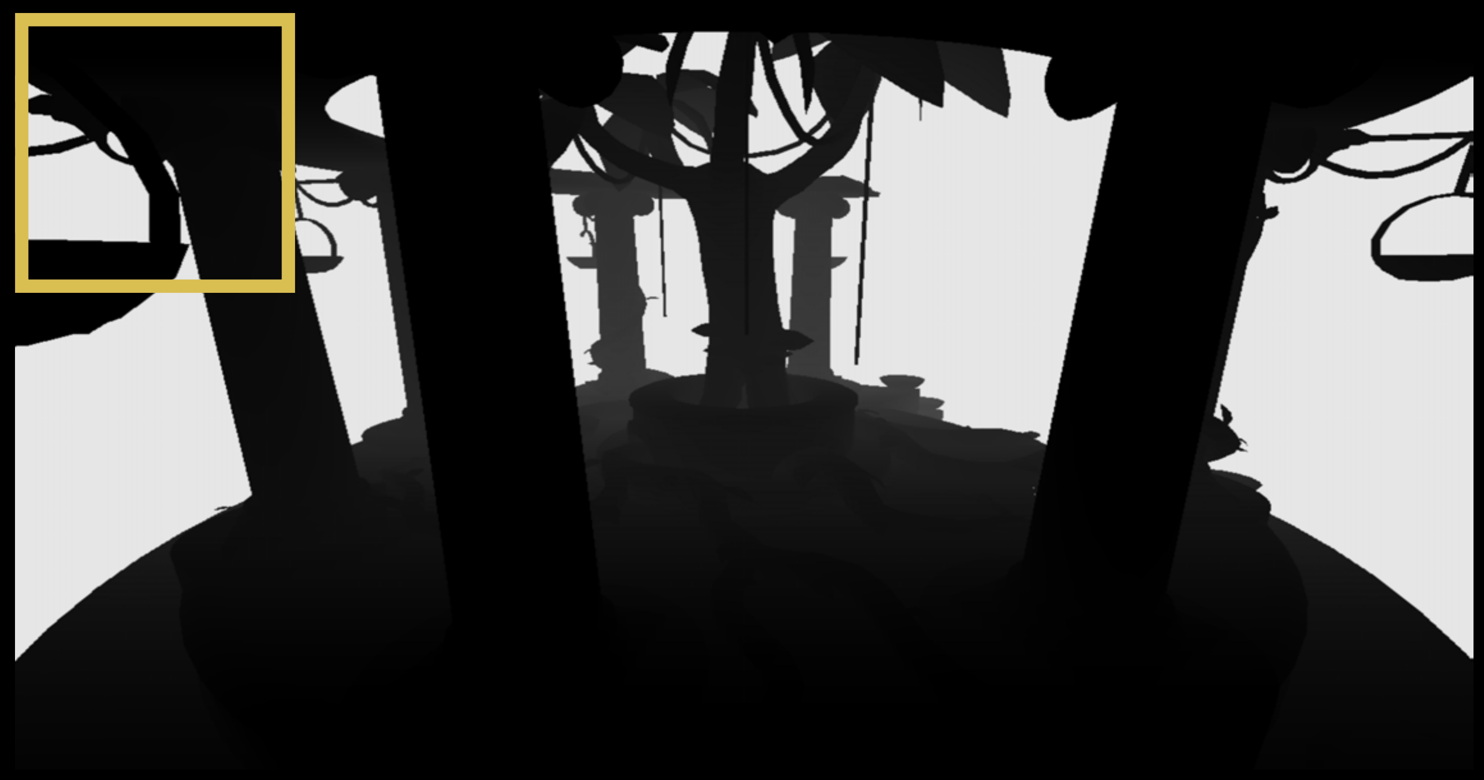
Depth attachments



Color Attachment Texture



GPU Tile Storage

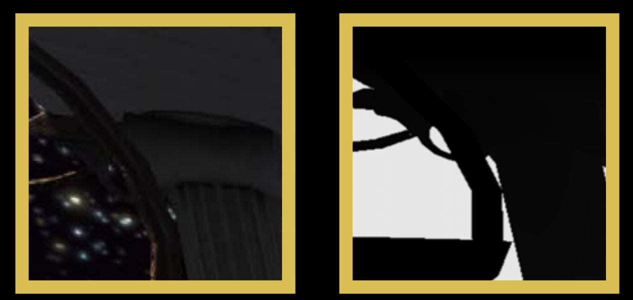


Depth Attachment Texture

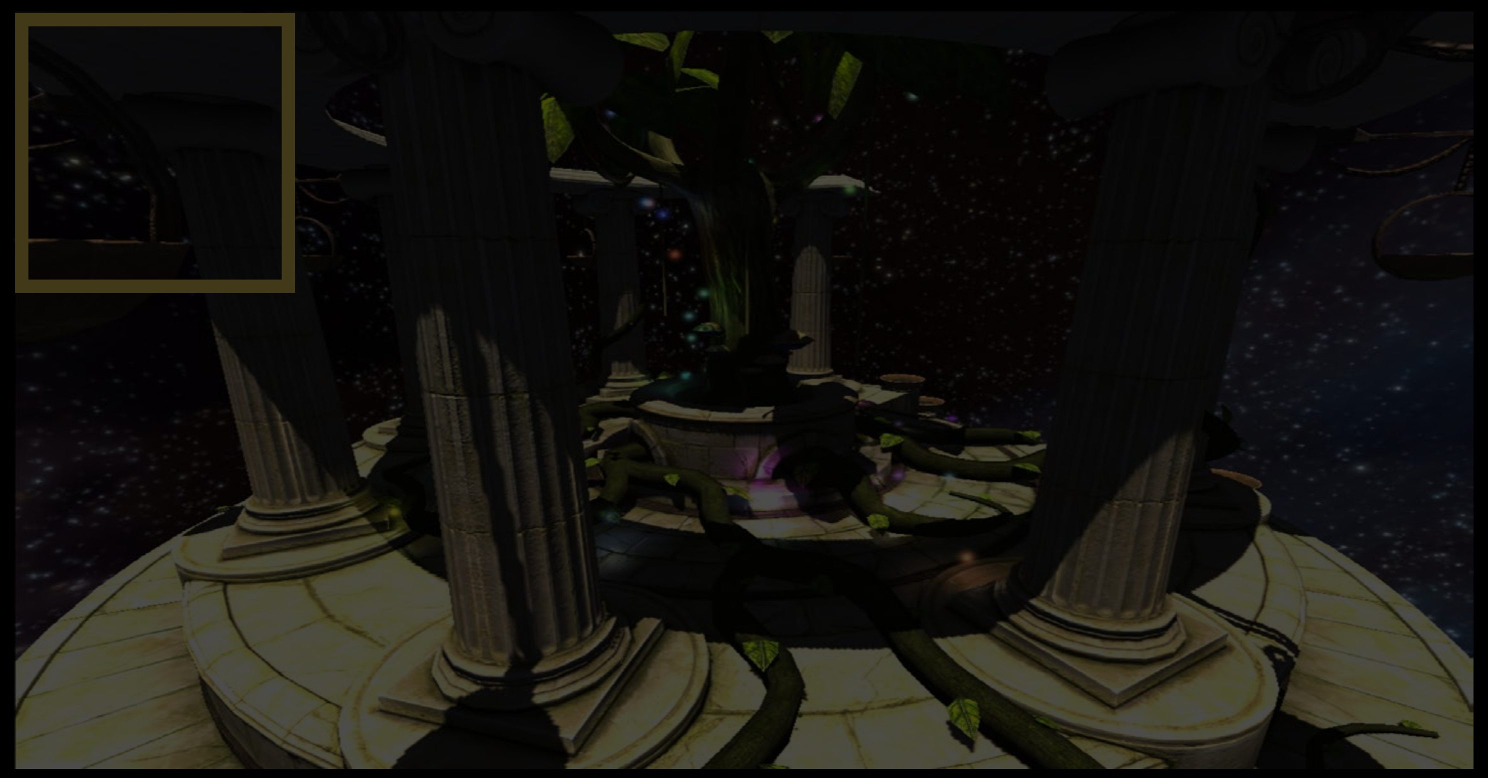
Use Case

Depth attachments

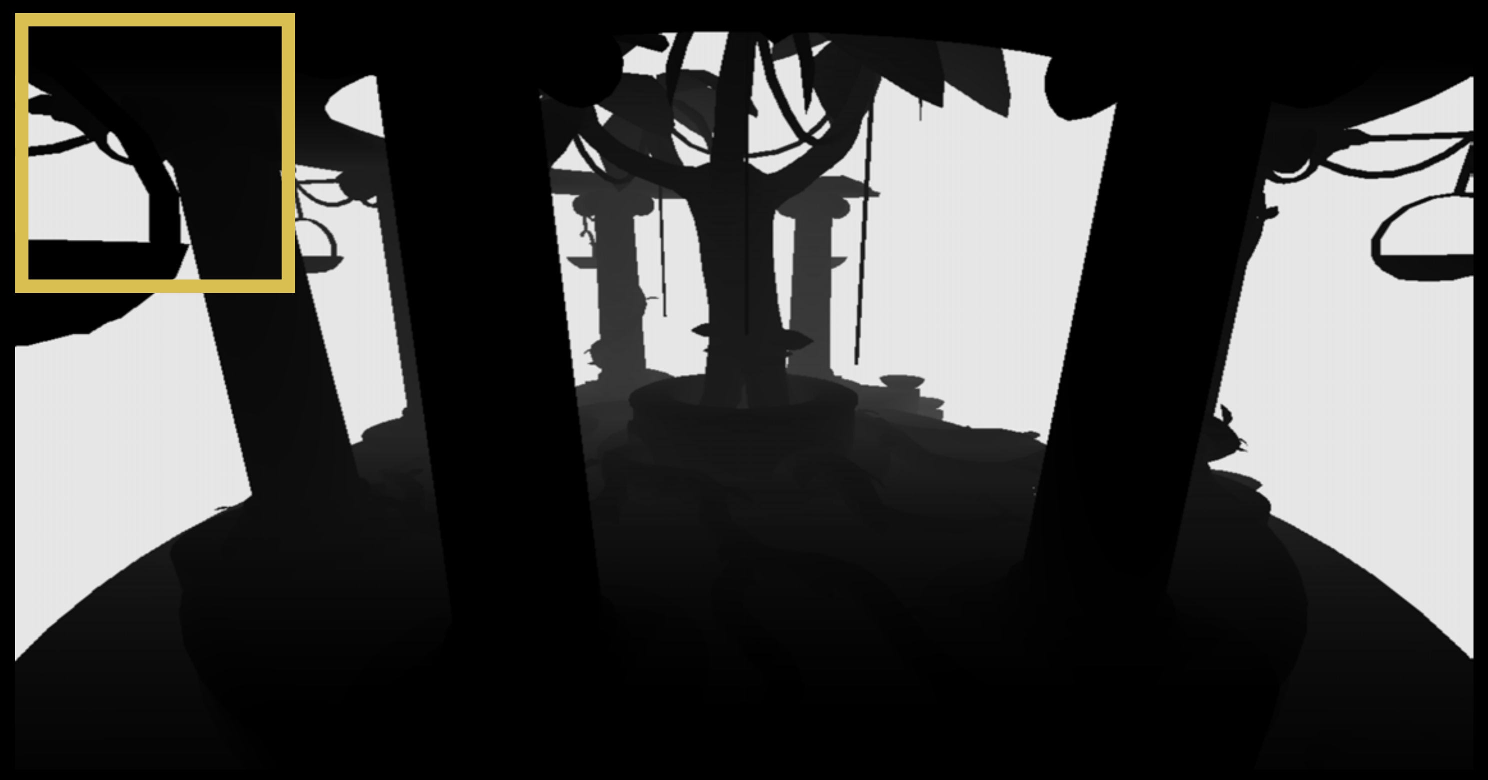
Depth attachments are required for depth testing



GPU Tile Storage



Color Attachment Texture



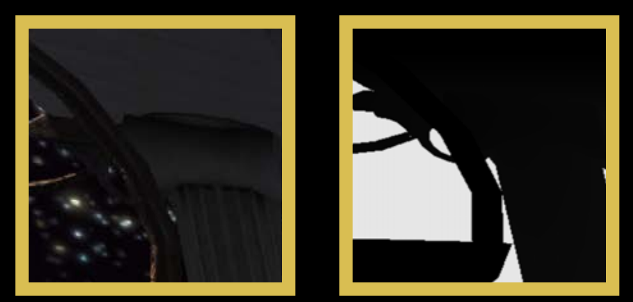
Depth Attachment Texture

Use Case

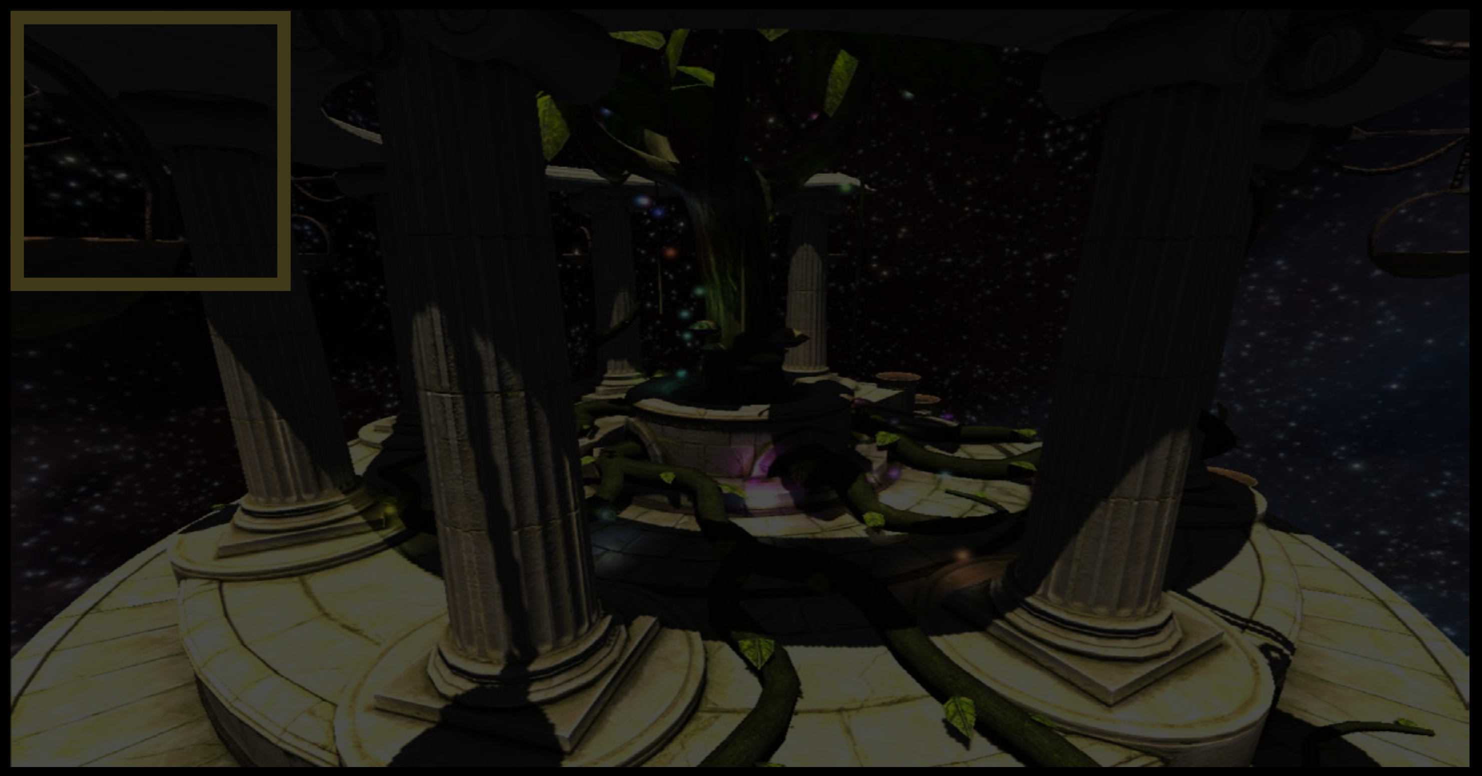
Depth attachments

Depth attachments are required for depth testing

If the depth attachment is not stored, make it memoryless



GPU Tile Storage



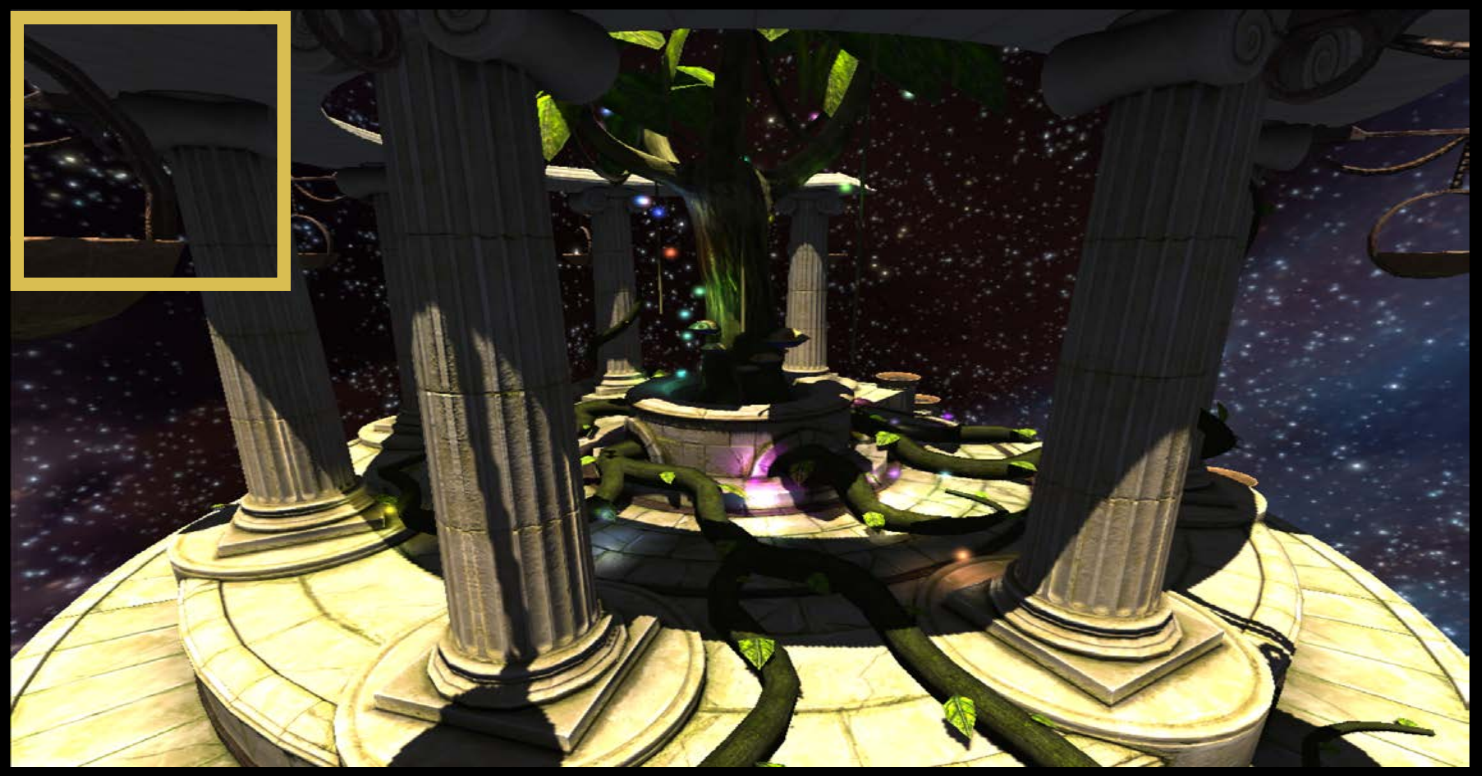
Color Attachment Texture

Depth Attachment Texture

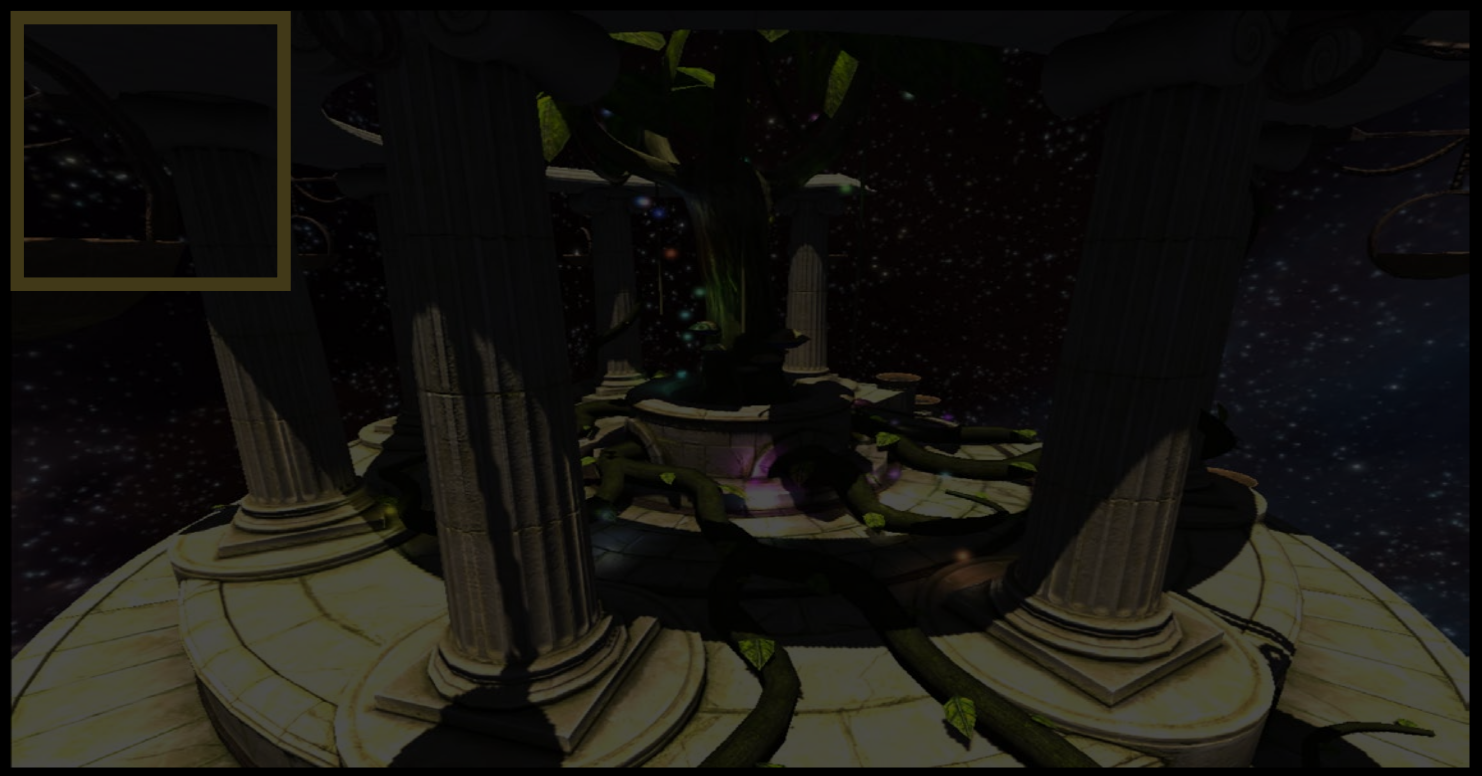
Use Case

Multisample color attachments

GPU Tile Storage



MSAA Color Attachment Texture

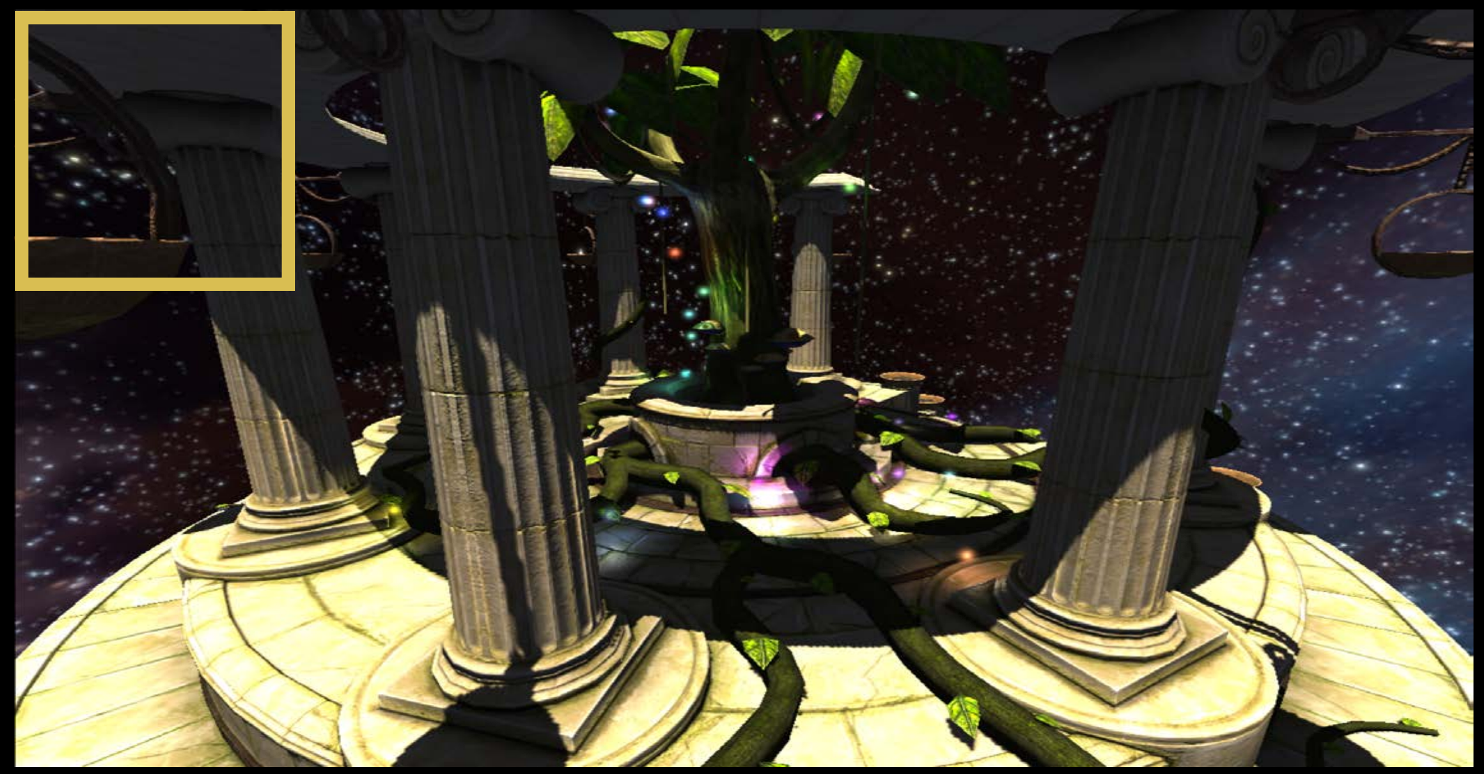


Resolve Color Attachment Texture

Use Case

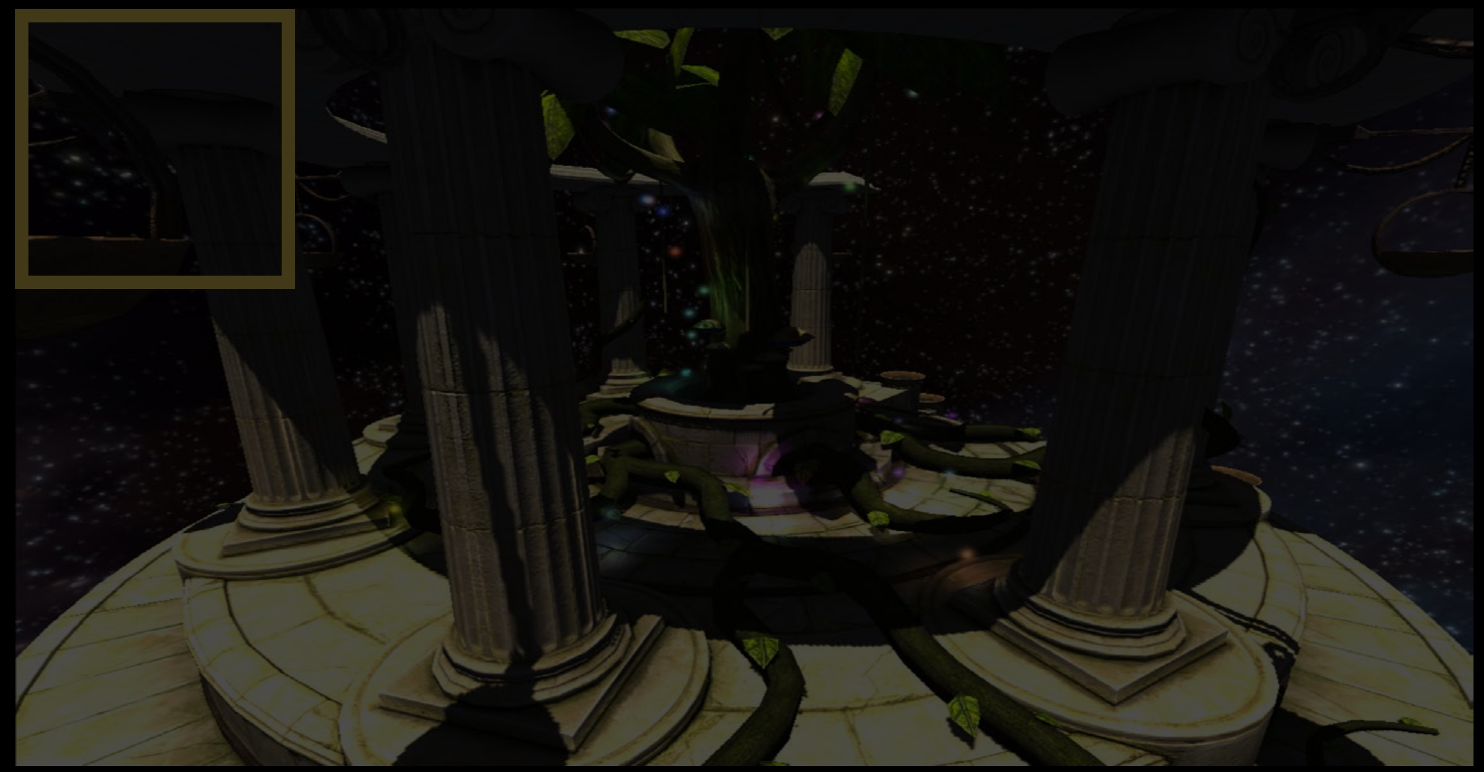
Multisample color attachments

MSAA attachments are required for multisample rendering



MSAA Color Attachment Texture

GPU Tile Storage



Resolve Color Attachment Texture

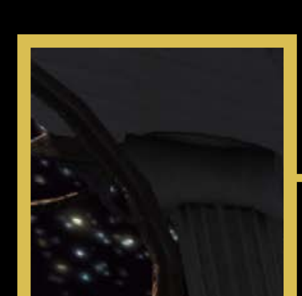
Use Case

Multisample color attachments

MSAA attachments are required for multisample rendering

If the color attachment is resolved, make the MSAA color attachment memoryless

MSAA Color Attachment Texture



GPU Tile Storage

Resolve Color Attachment Texture

Use Case

Memory savings

Texture Type	Potential Savings (MB)
Depth Texture on iPhone 6s Plus 1920x1080	7.9
Depth Texture on iPad Pro (12.9-inch) 2732x2048	21.3
4xMSAA Color Texture on iPhone 6s Plus 1920x1080	31.6
4xMSAA Color Texture on iPad Pro (12.9-inch) 2732x2048	85.4

Improved Tools

Jose Enrique D'Arnaude del Castillo GPU Software Engineer
Alp Yucebilgin GPU Software Engineer

What's New in Metal Tools

Metal System Trace

What's New in Metal Tools

Metal System Trace

GPU Overrides

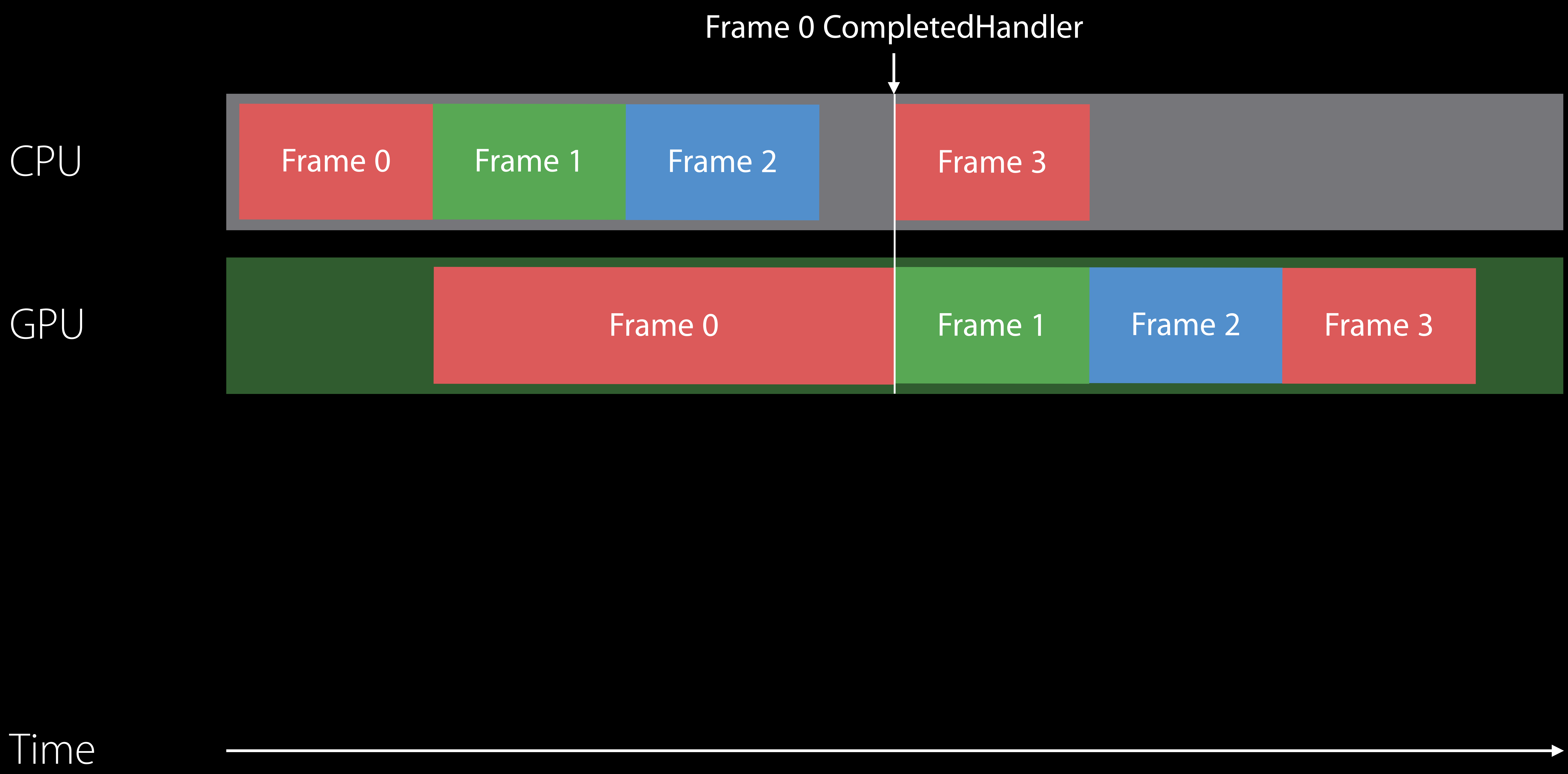
What's New in Metal Tools

Metal System Trace

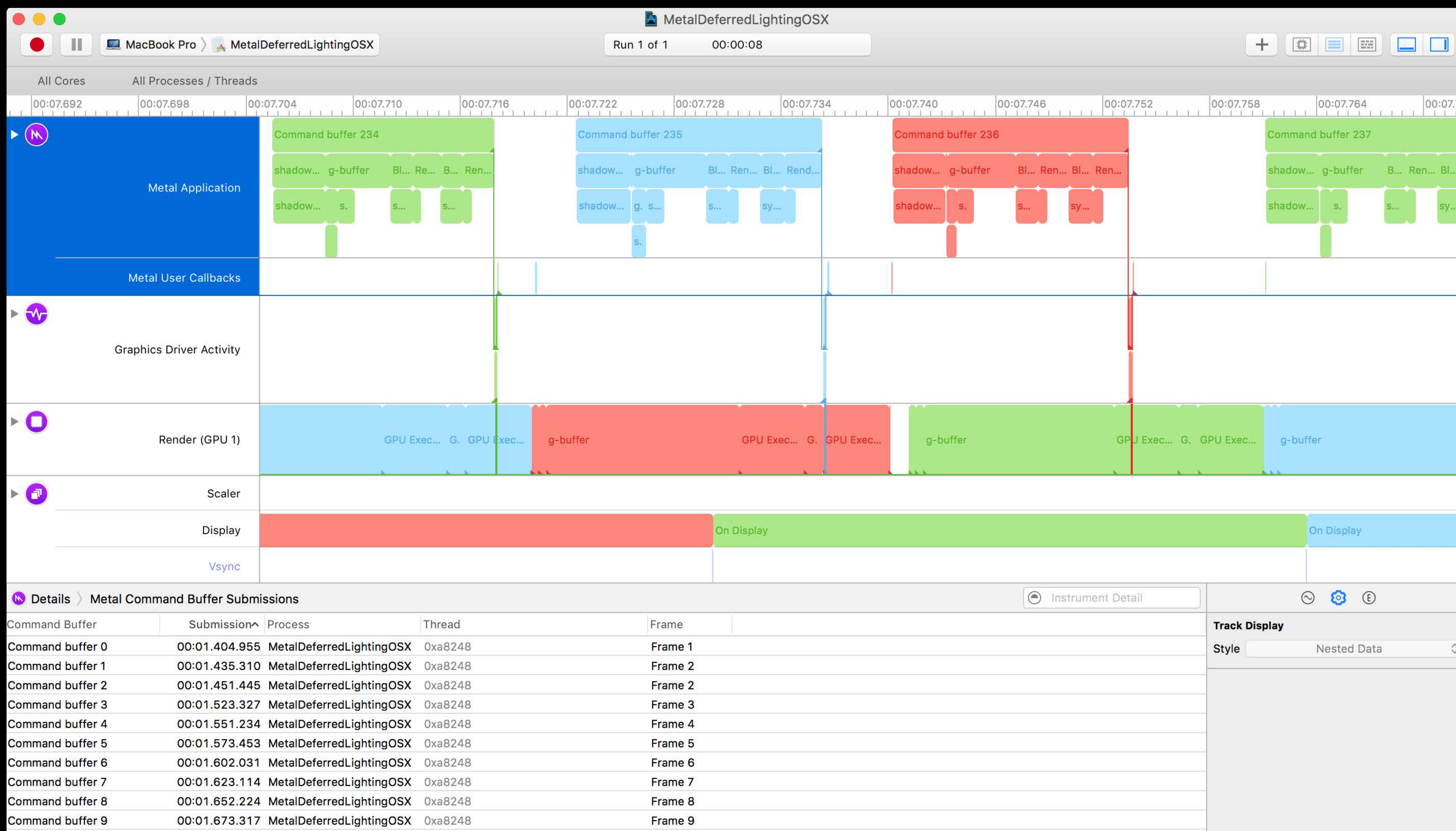
GPU Overrides

GPU Frame Debugger

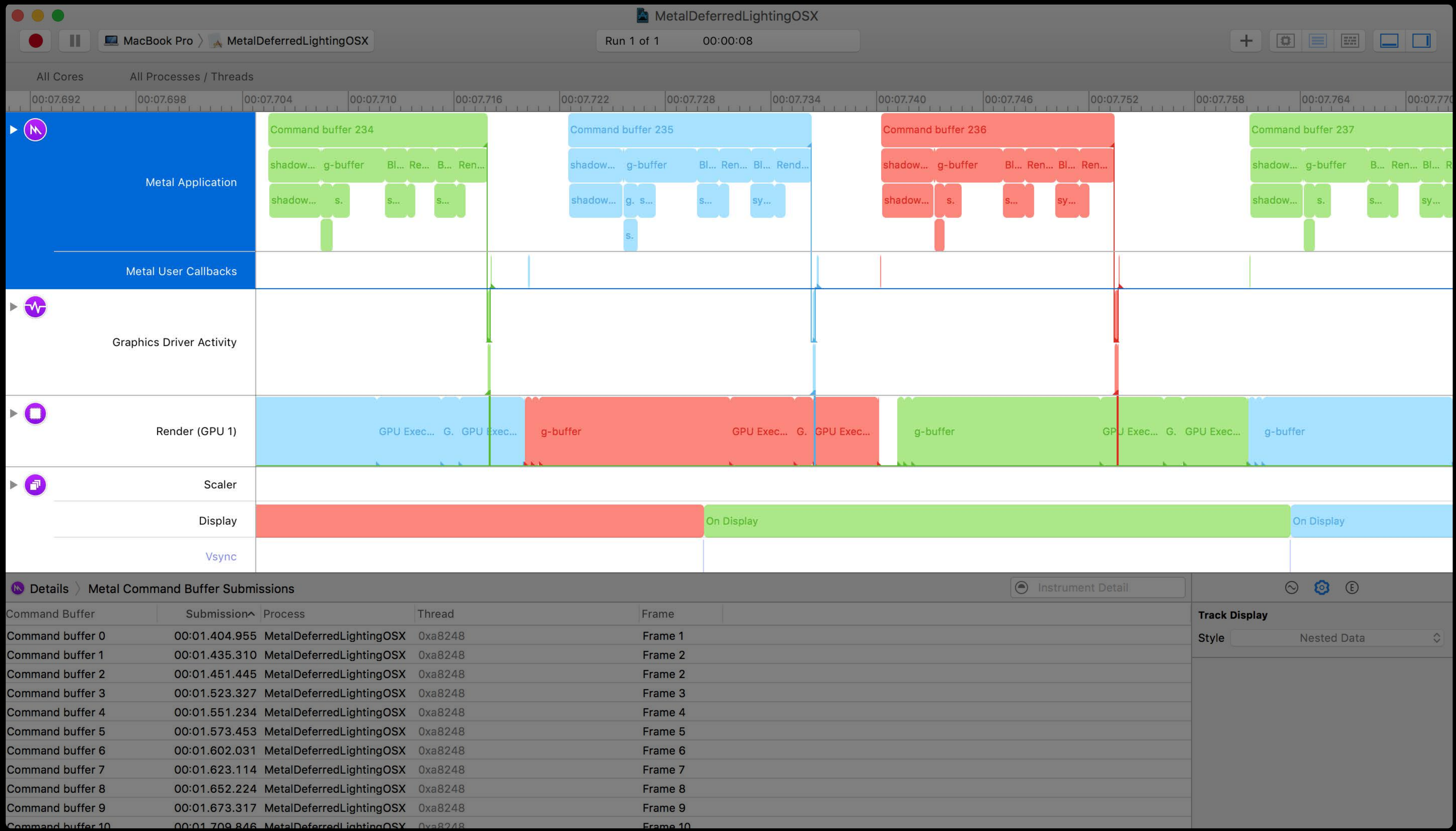
Metal System Trace



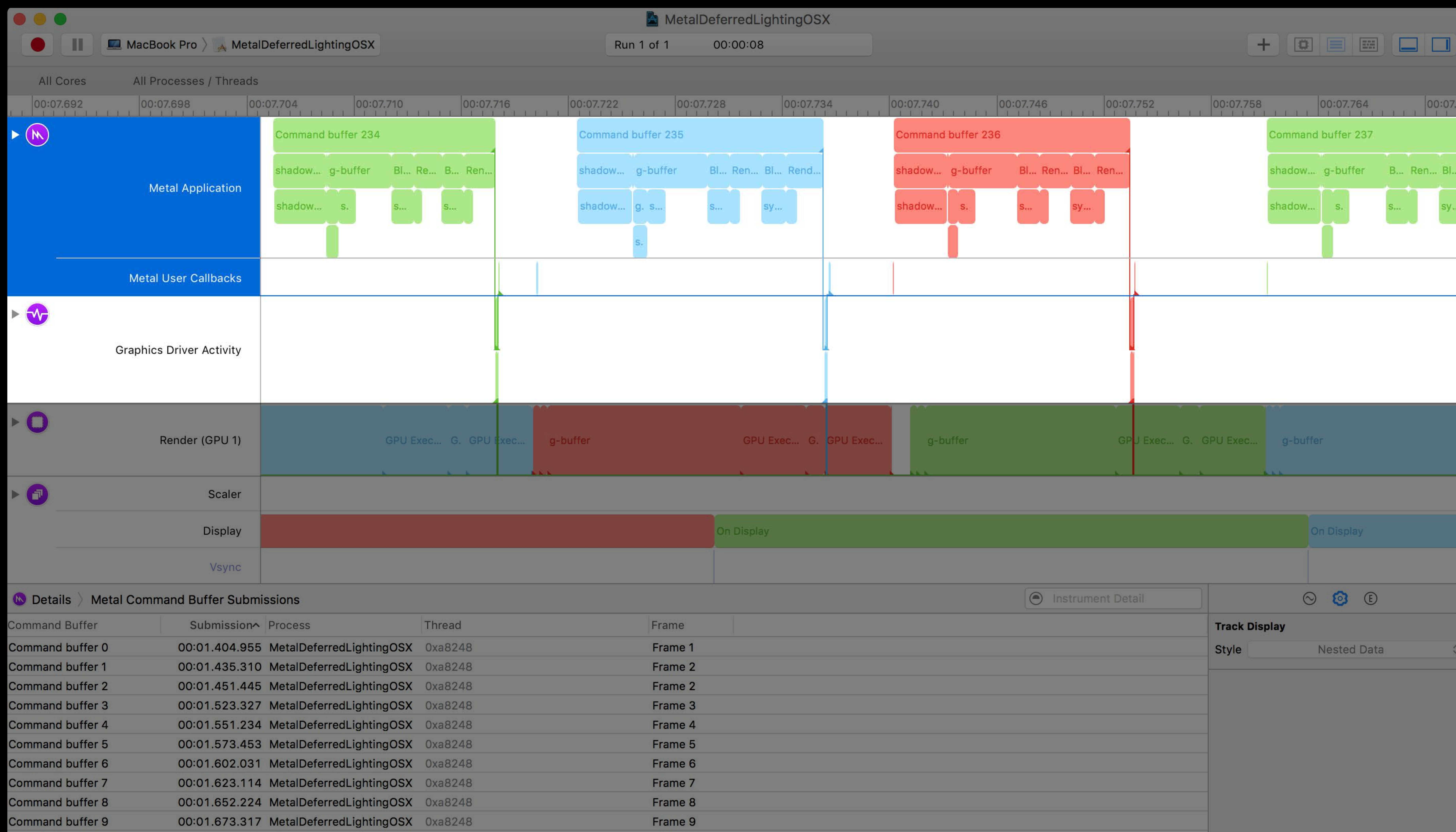
Metal System Trace



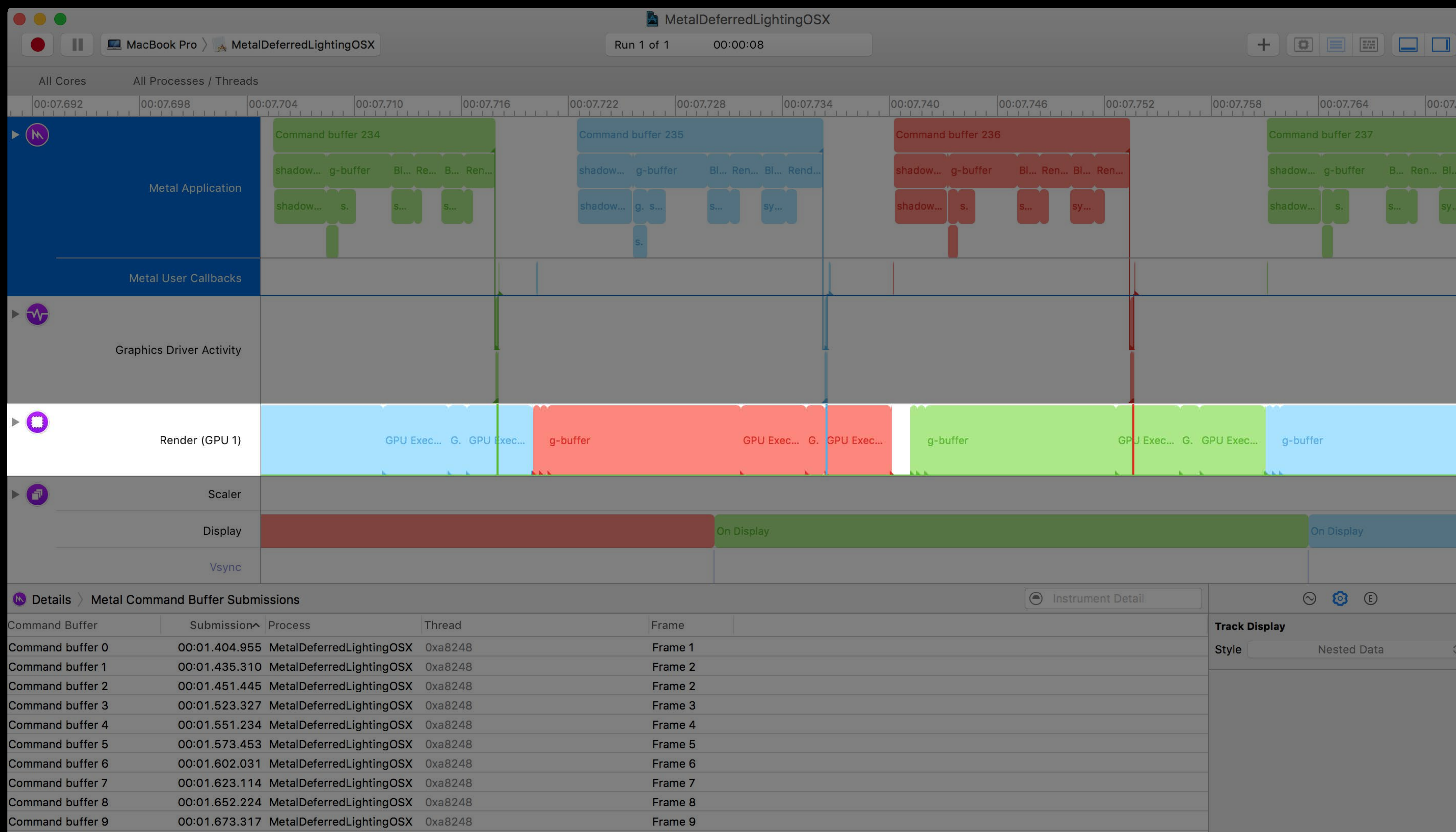
Metal System Trace



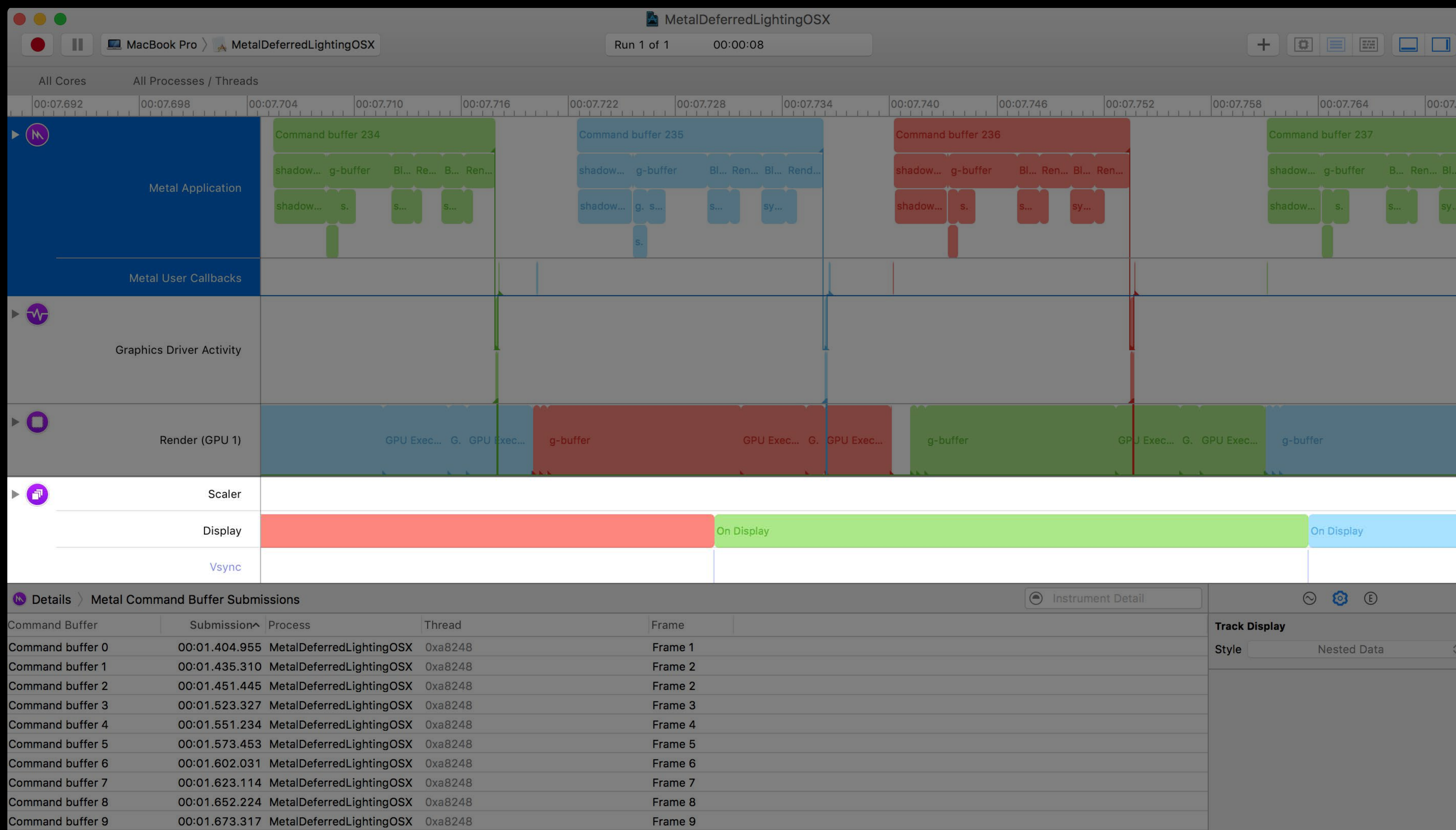
Metal System Trace



Metal System Trace



Metal System Trace



Metal System Trace

iOS

Metal System Trace

iOS

tvOS

Metal System Trace

NEW

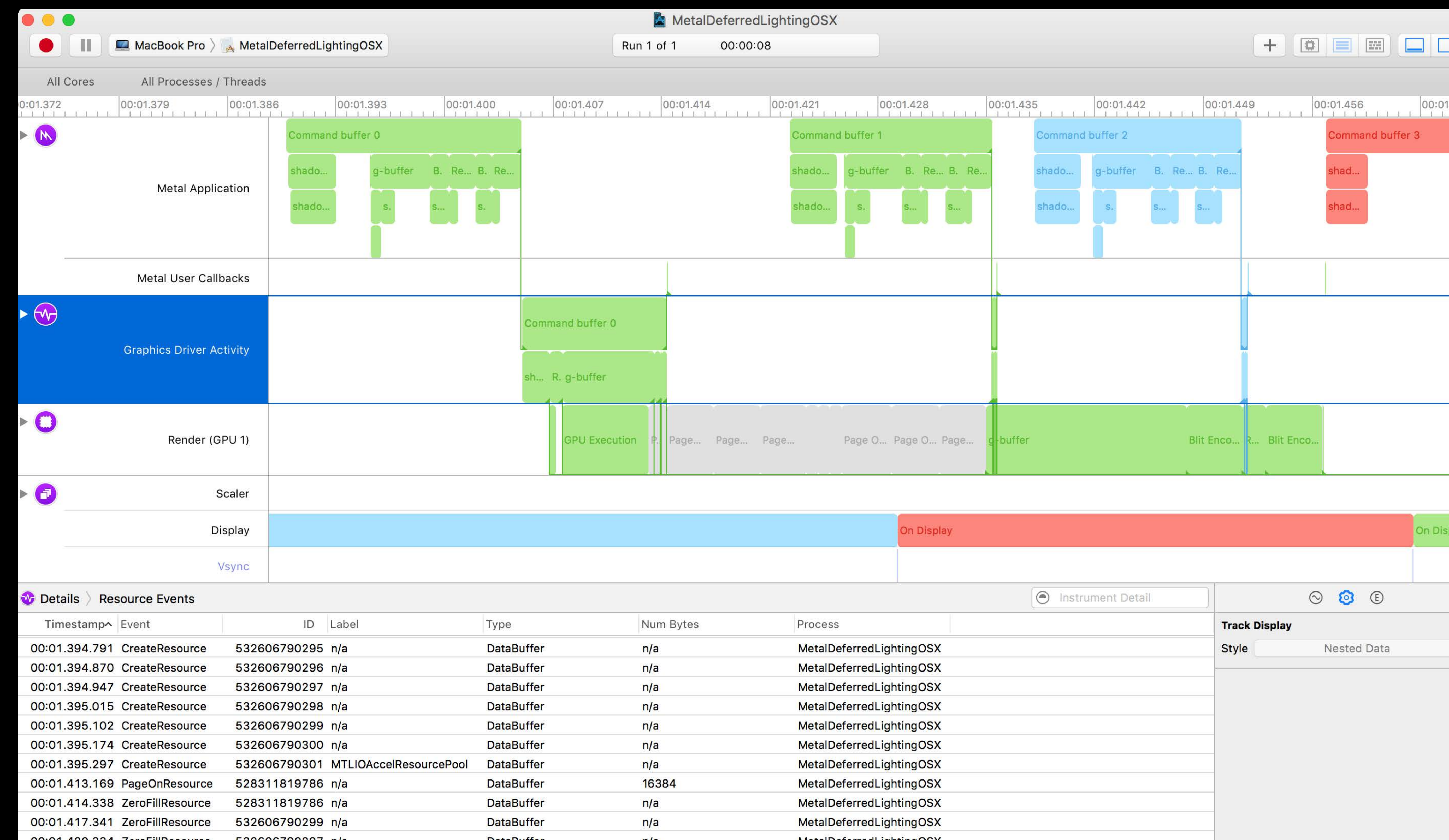
iOS tvOS macOS

Richer Metal Tracing

NEW

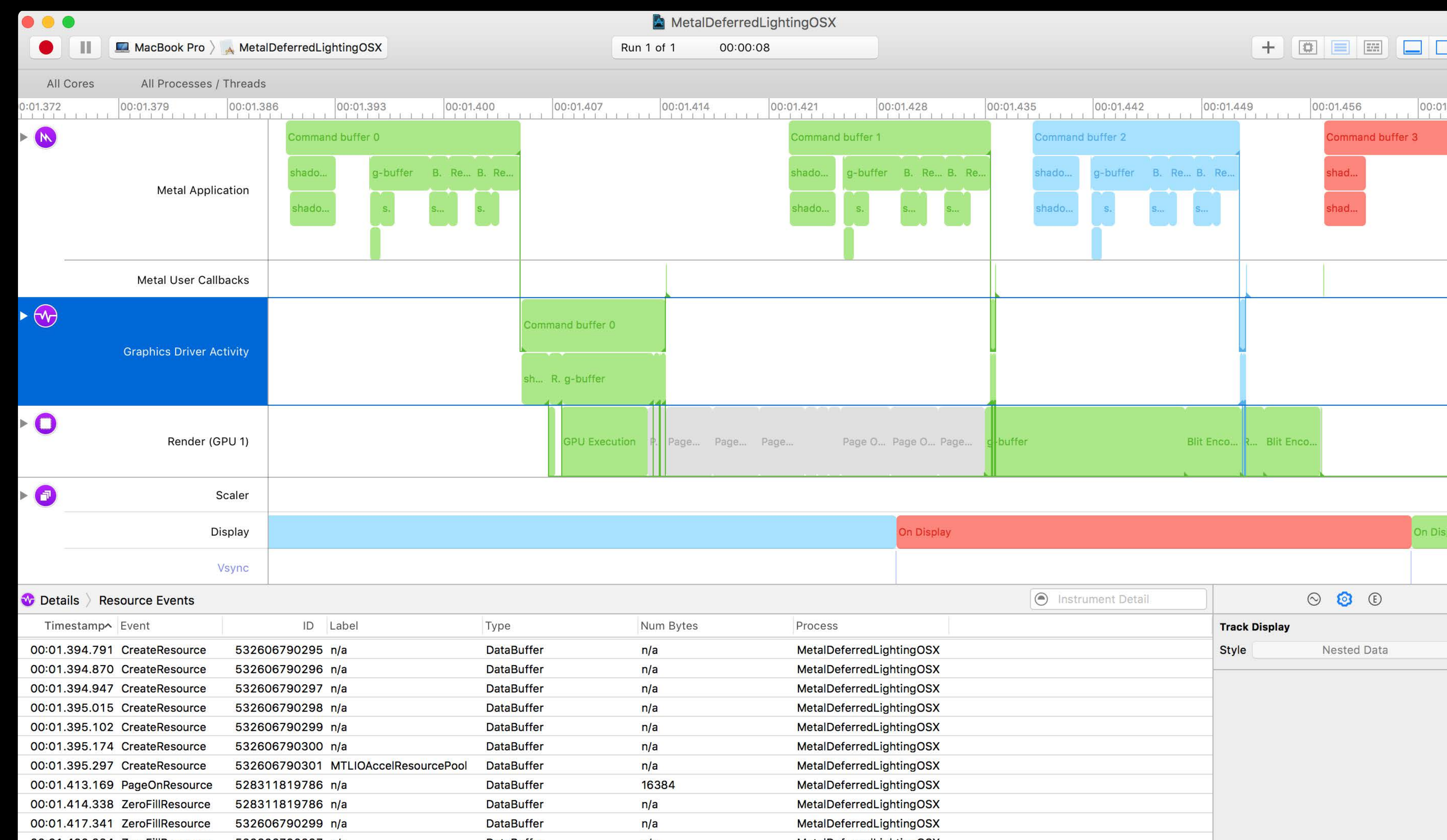
Richer Metal Tracing

Resource events



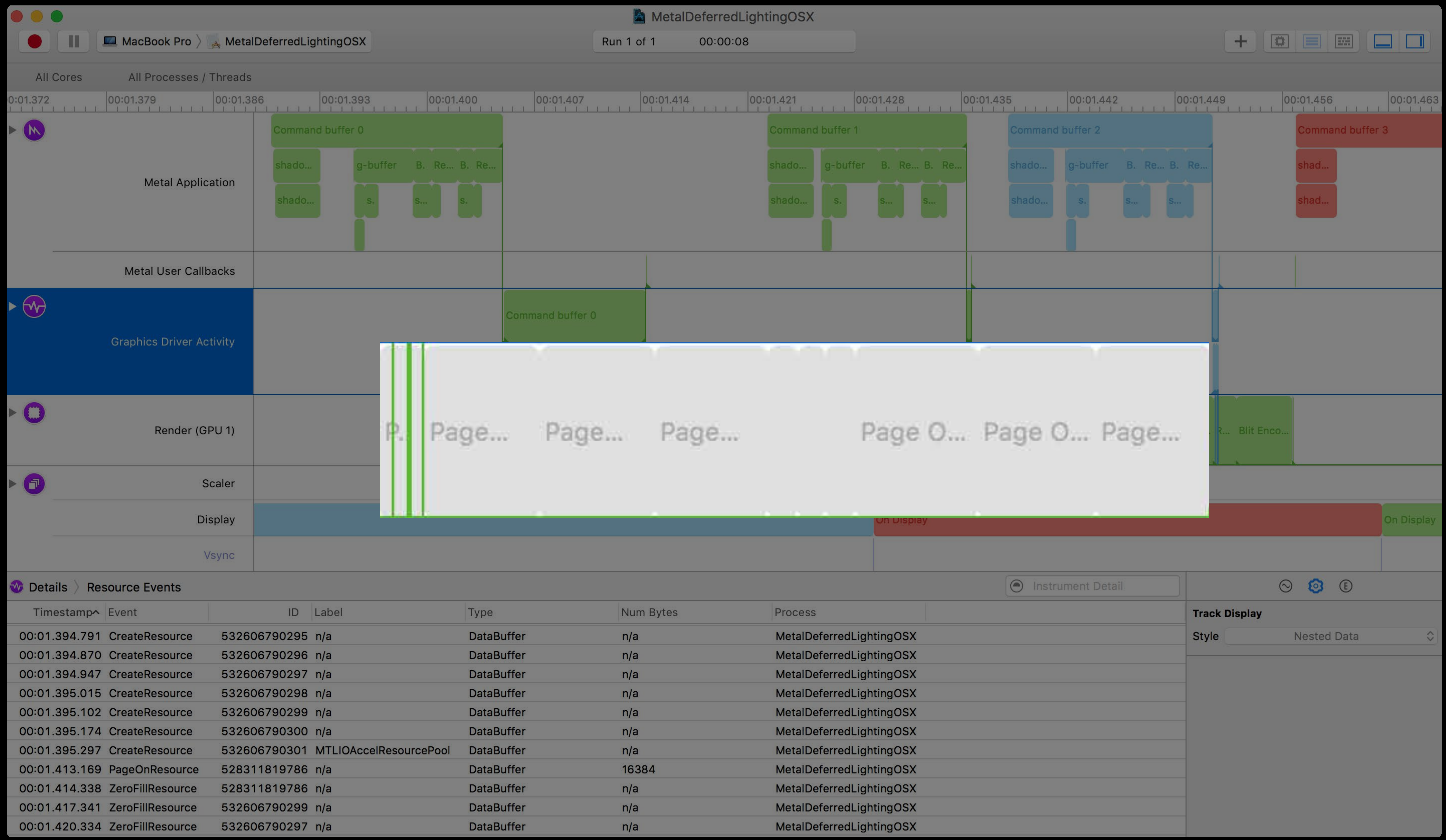
Richer Metal Tracing

Resource events



Richer Metal Tracing

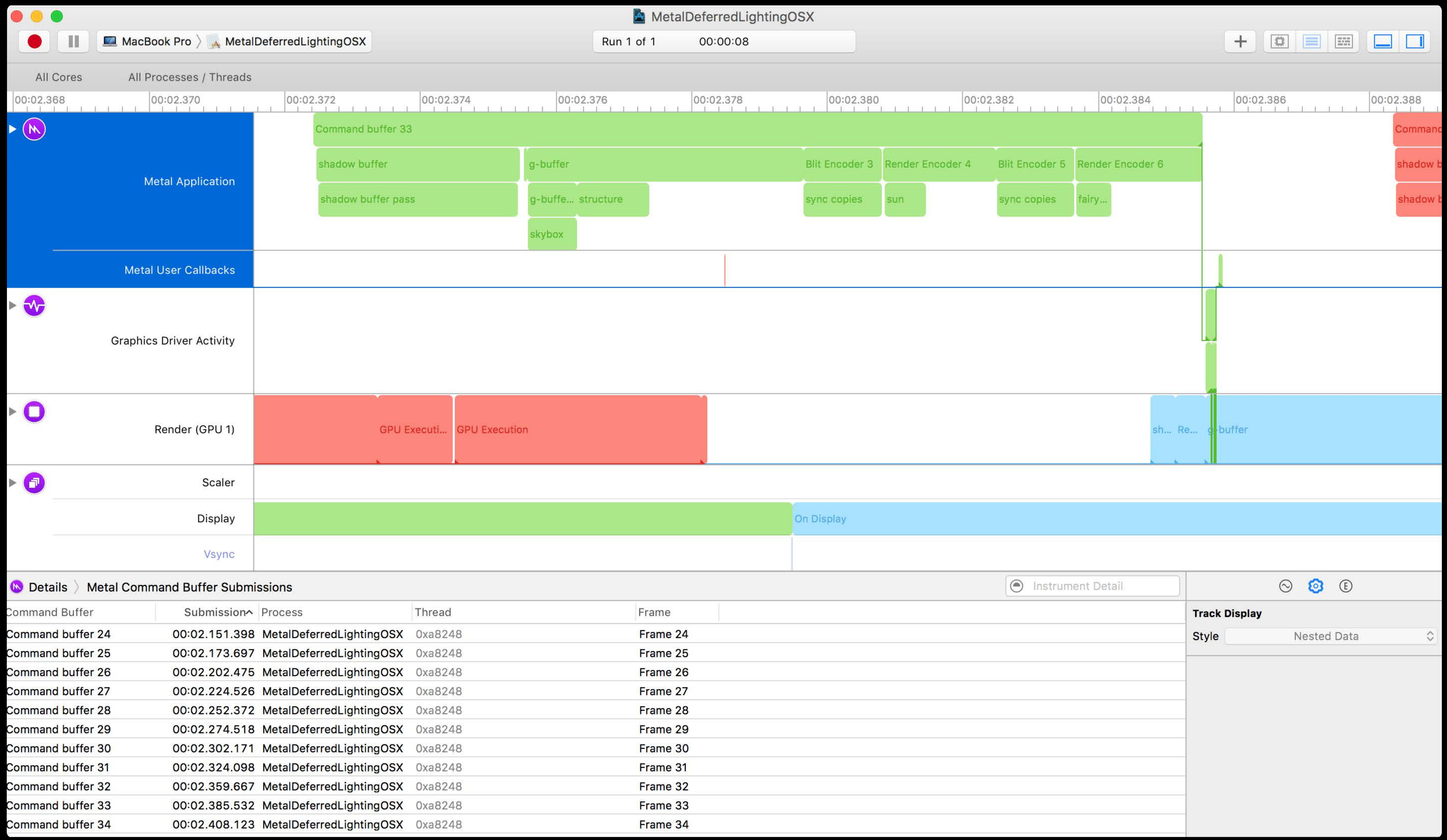
Resource events



Richer Metal Tracing

Resource events

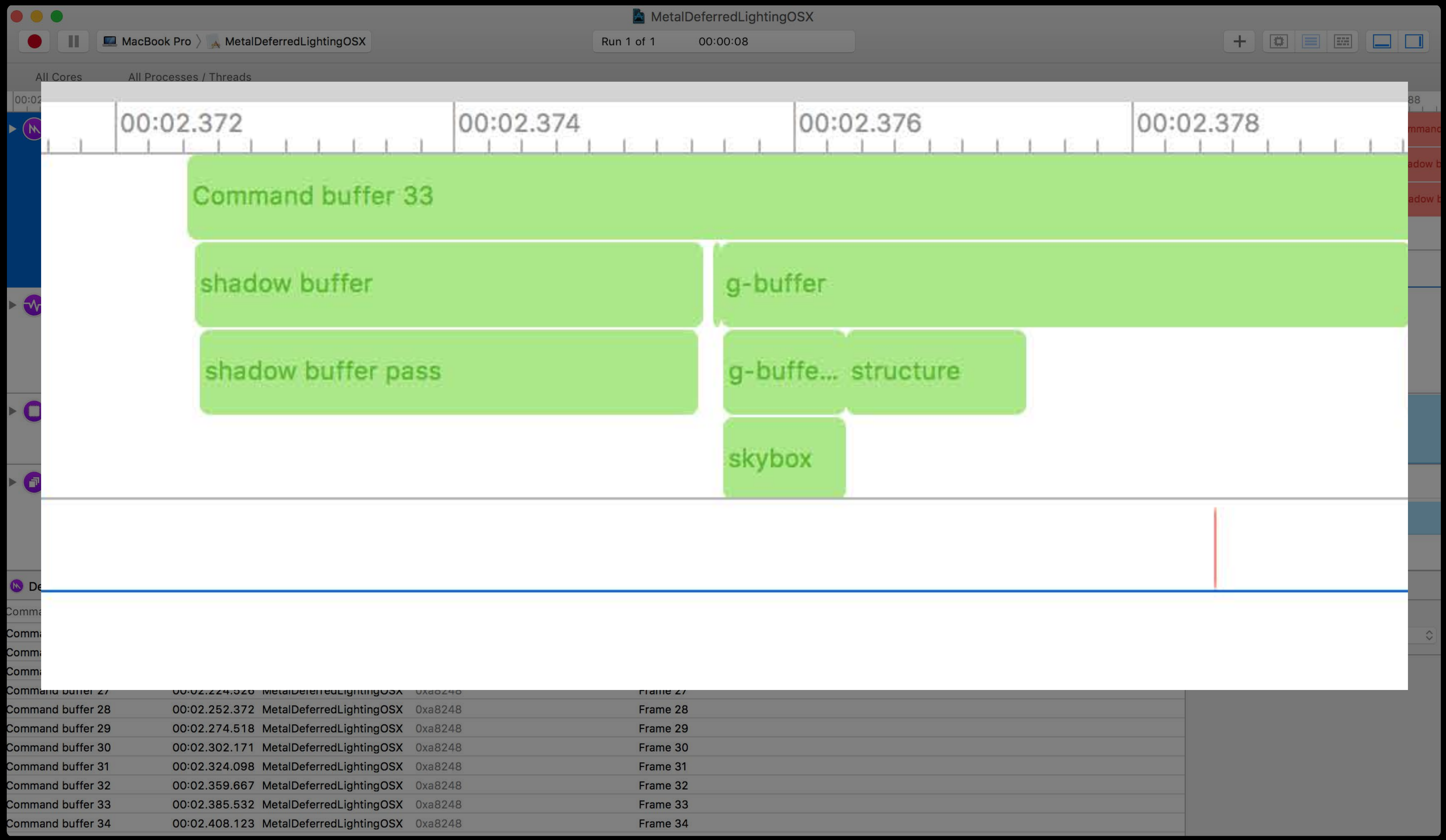
Debug groups



Richer Metal Tracing

Resource events

Debug groups

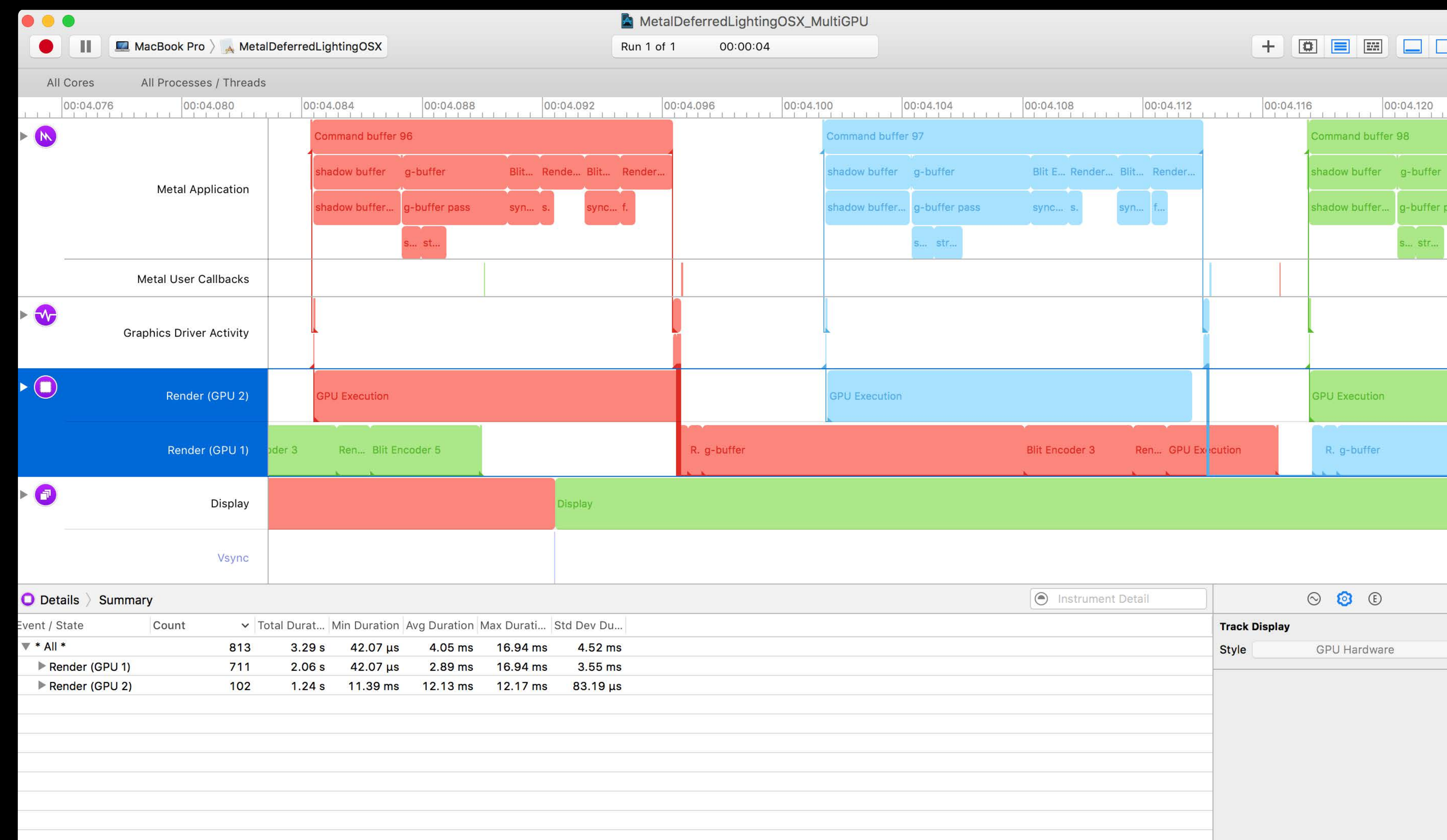


Richer Metal Tracing

Resource events

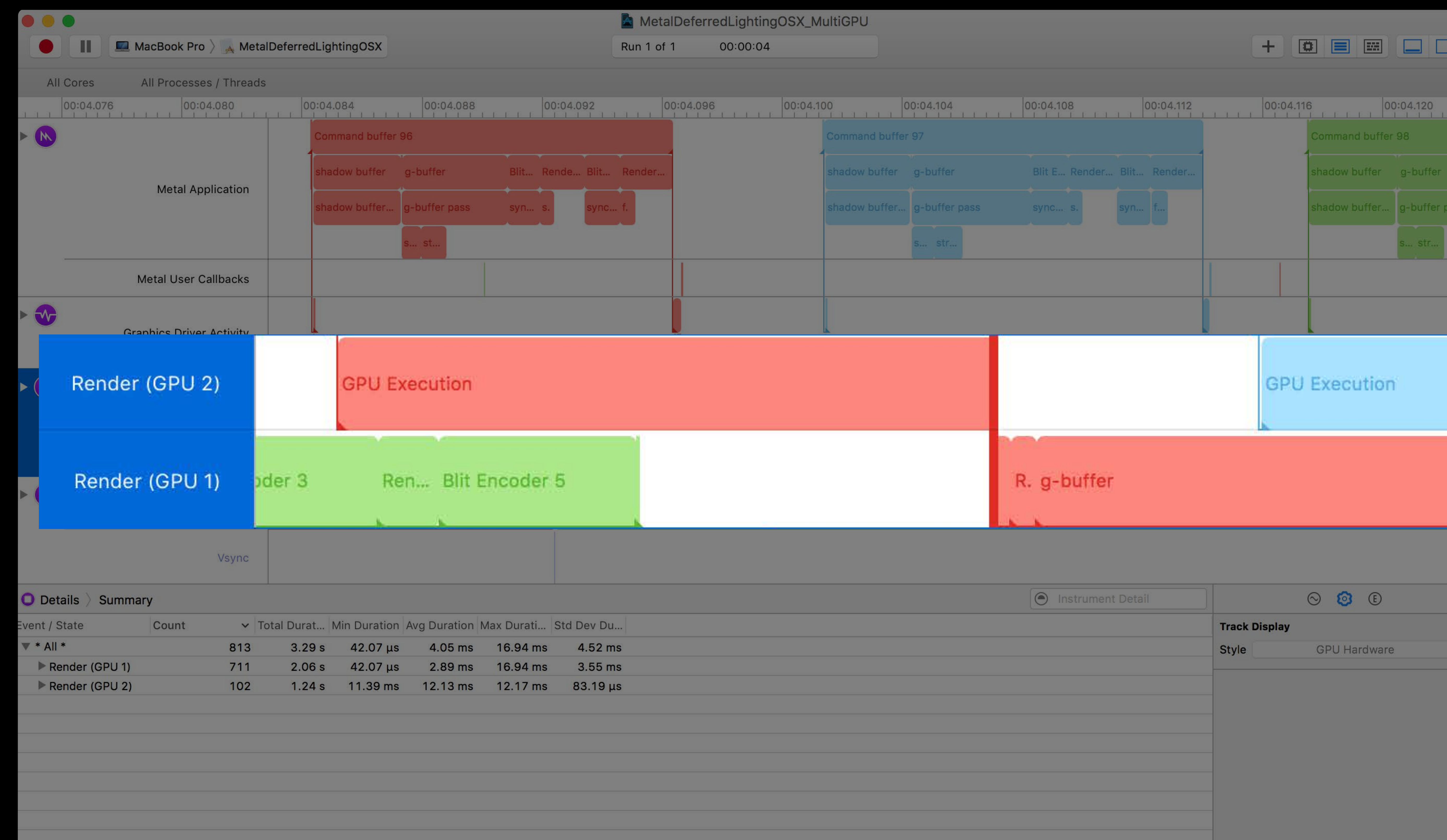
Debug groups

Multi GPU (macOS)



Richer Metal Tracing

- Resource events
- Debug groups
- Multi GPU (macOS)



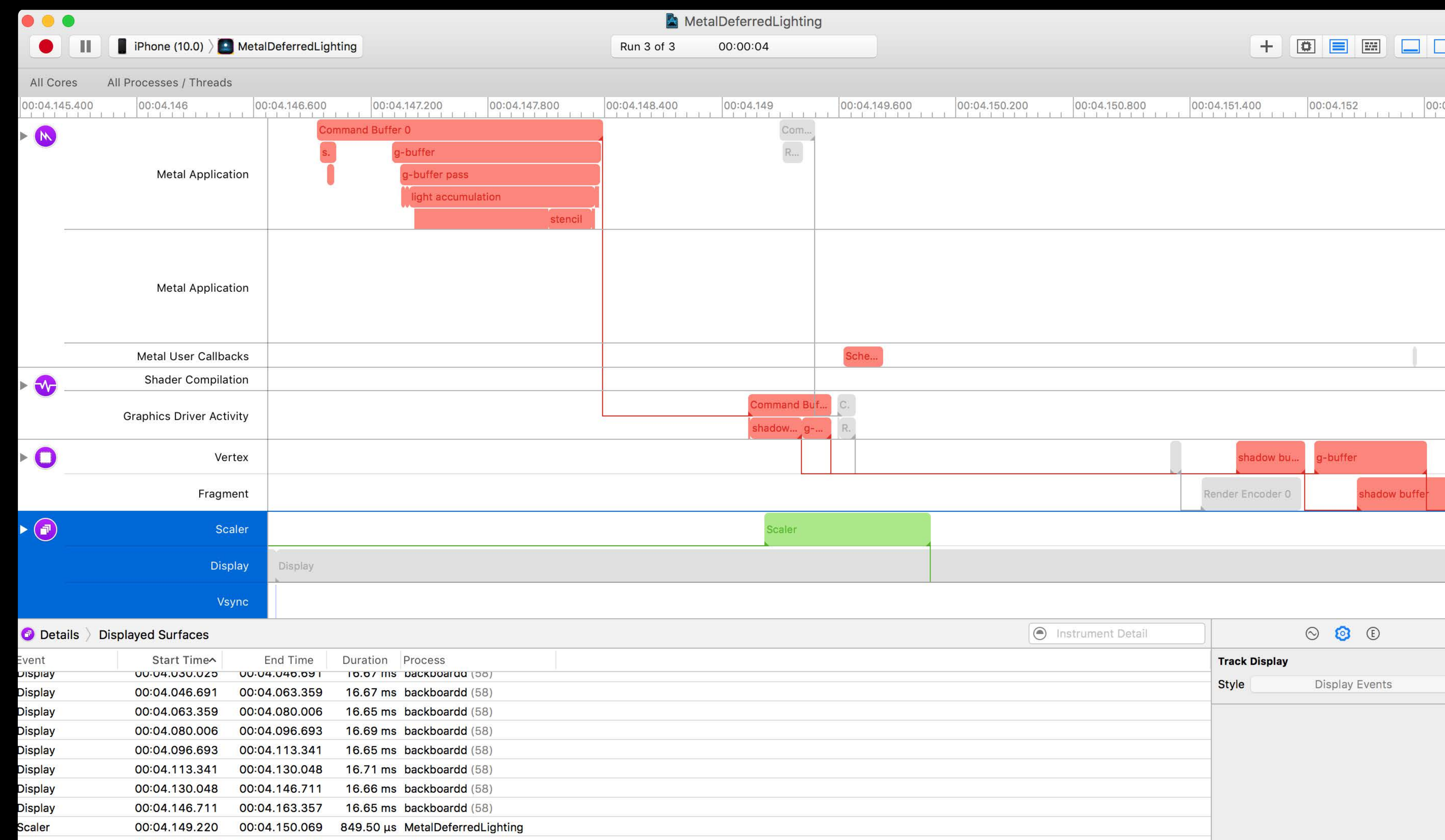
Richer Metal Tracing

Resource events

Debug groups

Multi GPU (macOS)

Scaler (iOS)



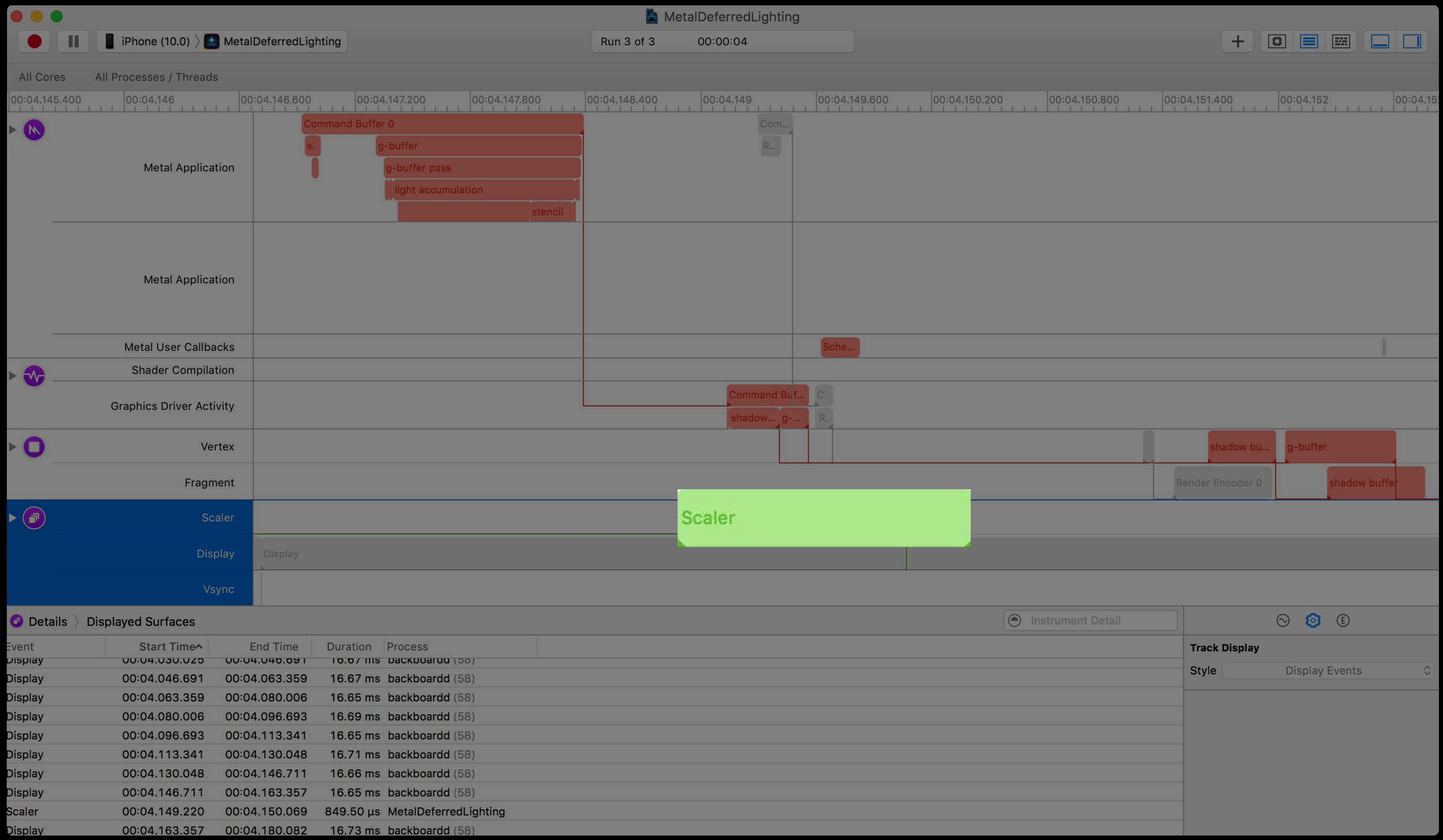
Richer Metal Tracing

Resource events

Debug groups

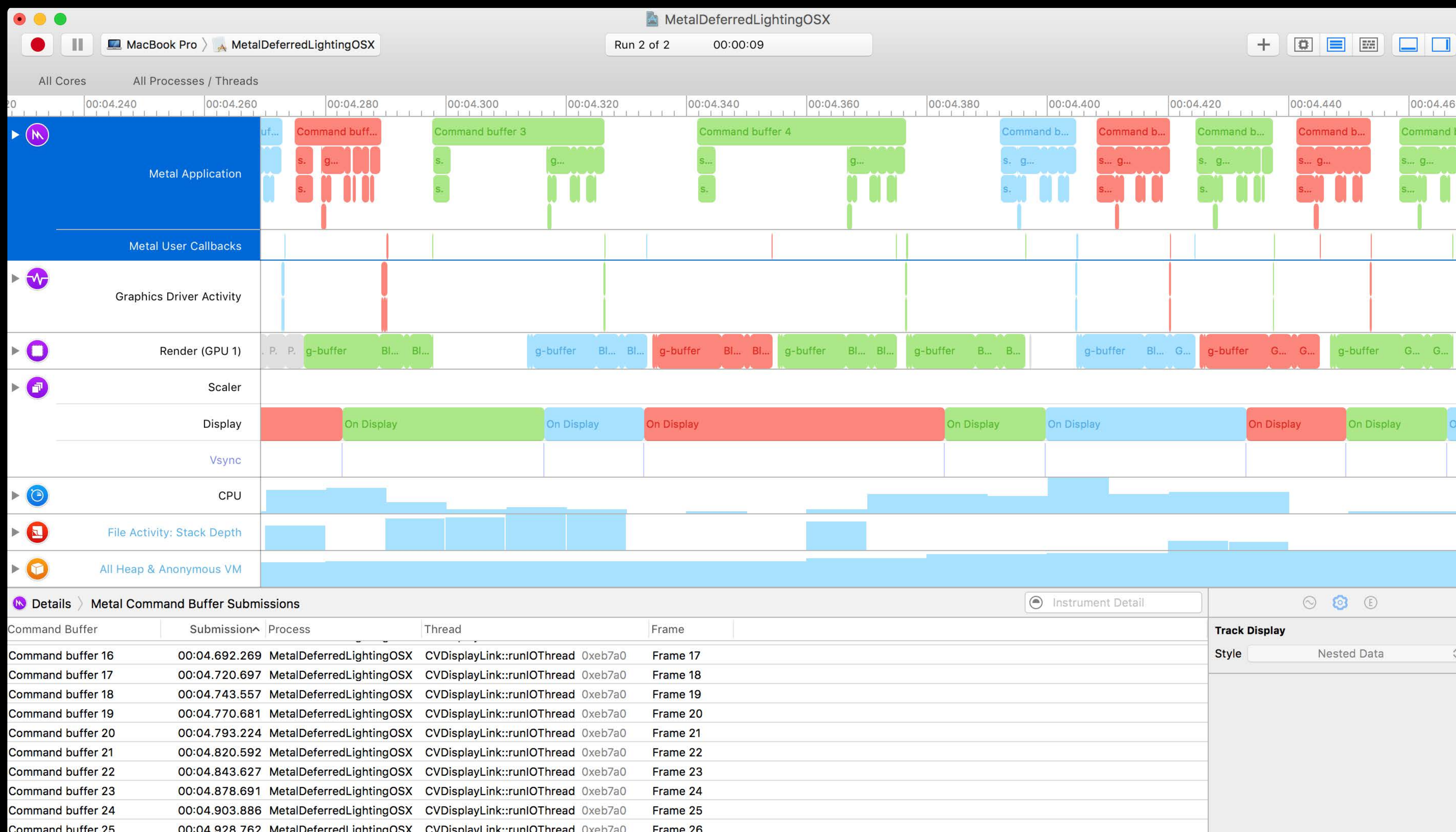
Multi GPU (macOS)

Scaler (iOS)

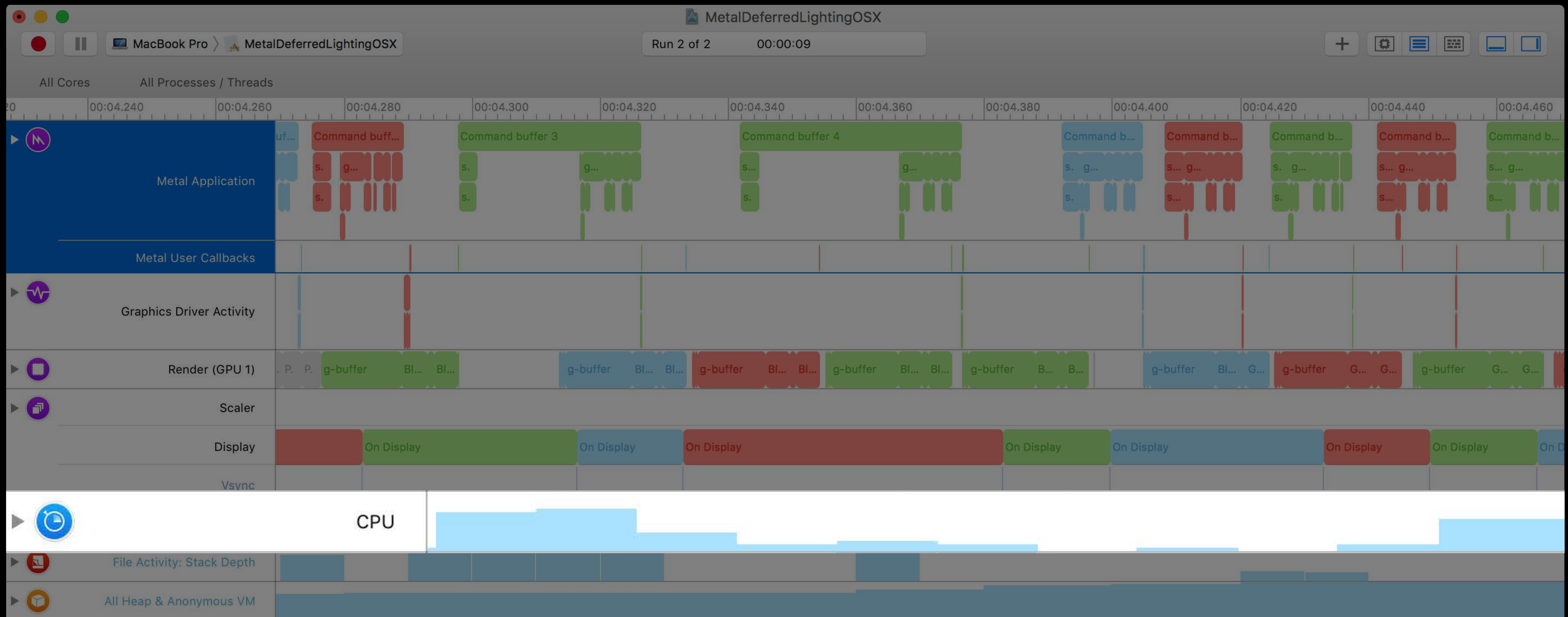


Integration with More Instruments

NEW

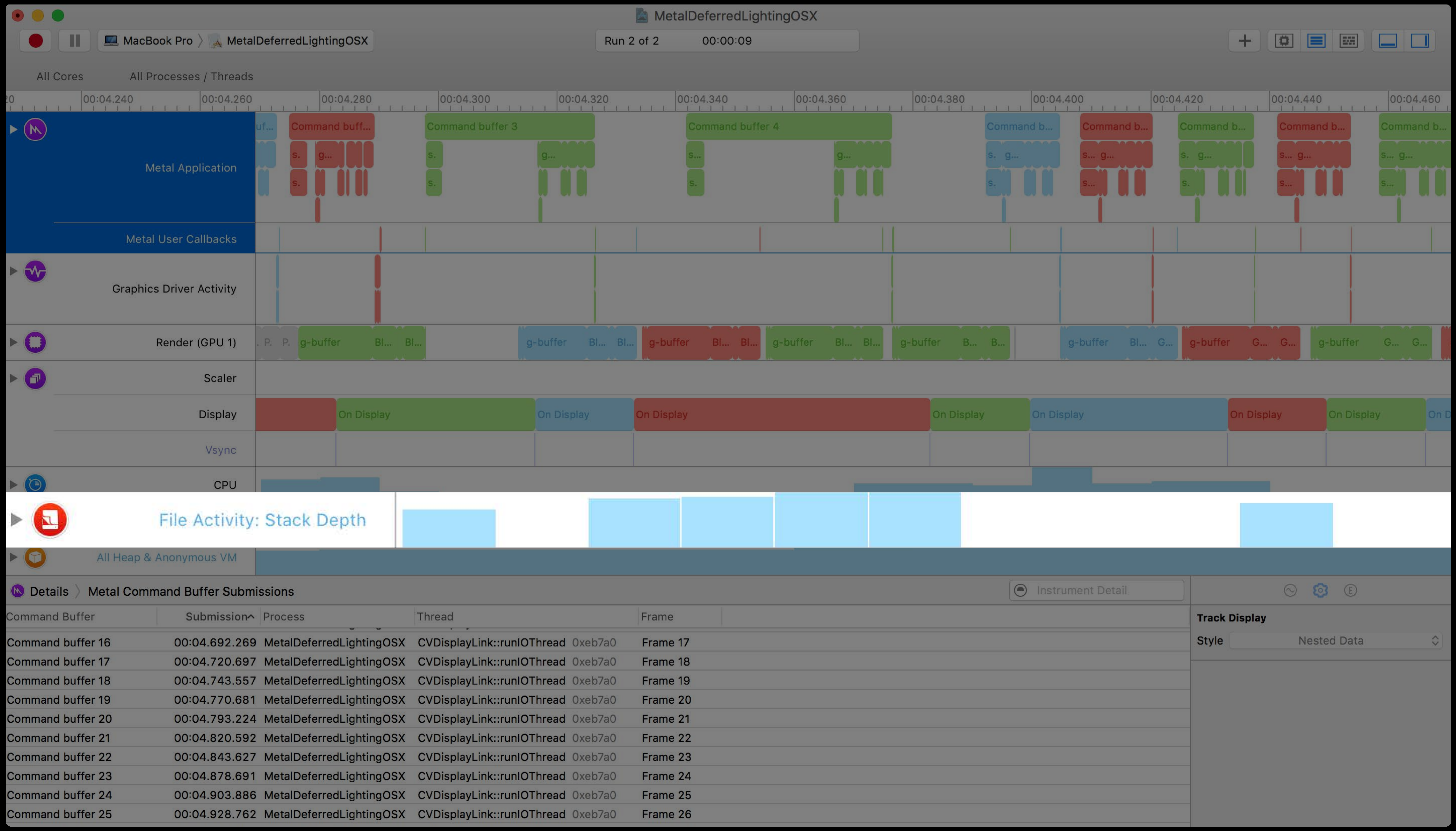


Integration with More Instruments

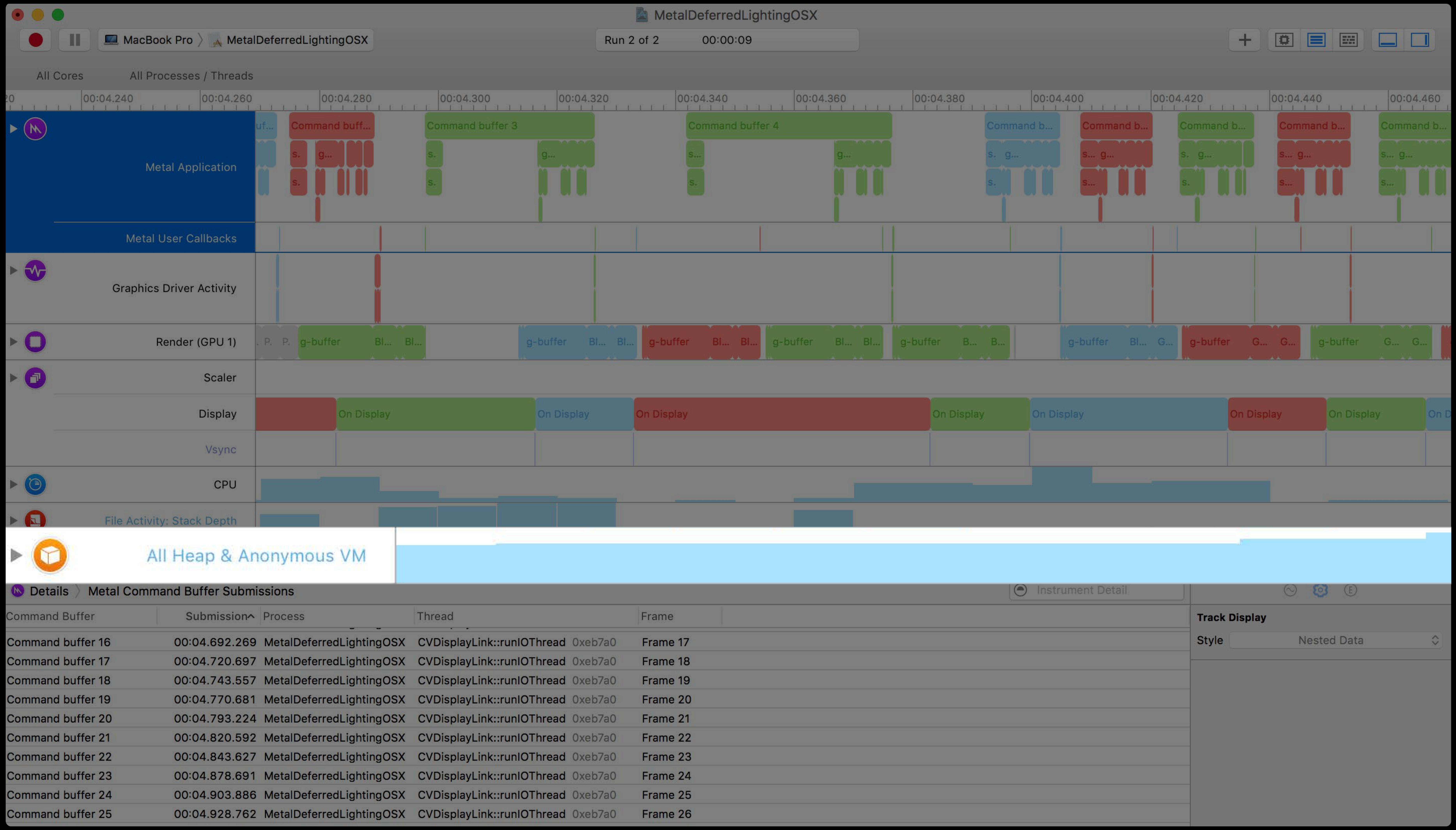


Command Buffer	Submission	Process	Thread	Frame
Command buffer 16	00:04.692.269	MetalDeferredLightingOSX	CVDisplayLink::runIOThread 0xeb7a0	Frame 17
Command buffer 17	00:04.720.697	MetalDeferredLightingOSX	CVDisplayLink::runIOThread 0xeb7a0	Frame 18
Command buffer 18	00:04.743.557	MetalDeferredLightingOSX	CVDisplayLink::runIOThread 0xeb7a0	Frame 19
Command buffer 19	00:04.770.681	MetalDeferredLightingOSX	CVDisplayLink::runIOThread 0xeb7a0	Frame 20
Command buffer 20	00:04.793.224	MetalDeferredLightingOSX	CVDisplayLink::runIOThread 0xeb7a0	Frame 21
Command buffer 21	00:04.820.592	MetalDeferredLightingOSX	CVDisplayLink::runIOThread 0xeb7a0	Frame 22
Command buffer 22	00:04.843.627	MetalDeferredLightingOSX	CVDisplayLink::runIOThread 0xeb7a0	Frame 23
Command buffer 23	00:04.878.691	MetalDeferredLightingOSX	CVDisplayLink::runIOThread 0xeb7a0	Frame 24
Command buffer 24	00:04.903.886	MetalDeferredLightingOSX	CVDisplayLink::runIOThread 0xeb7a0	Frame 25
Command buffer 25	00:04.928.762	MetalDeferredLightingOSX	CVDisplayLink::runIOThread 0xeb7a0	Frame 26

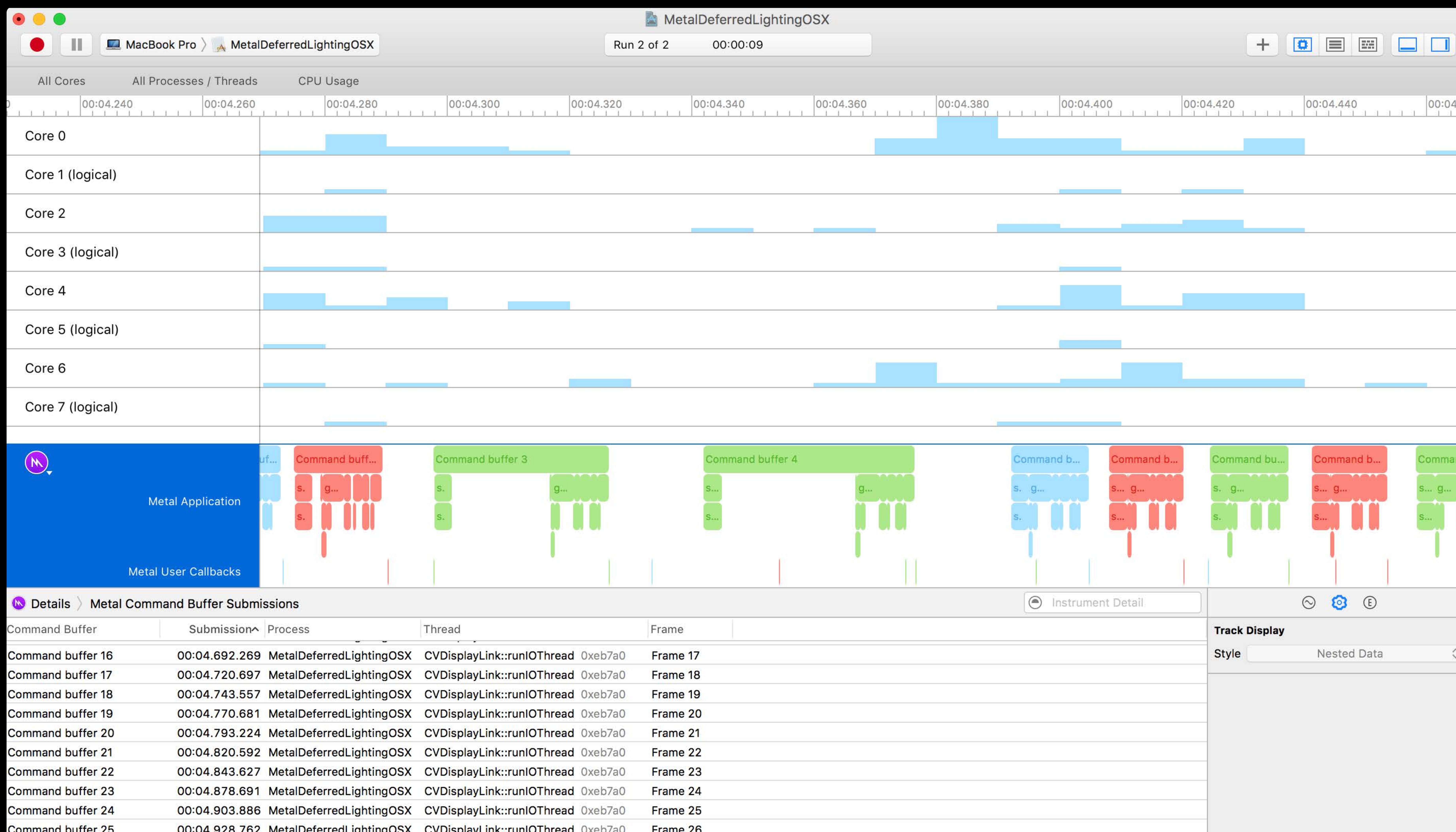
Integration with More Instruments



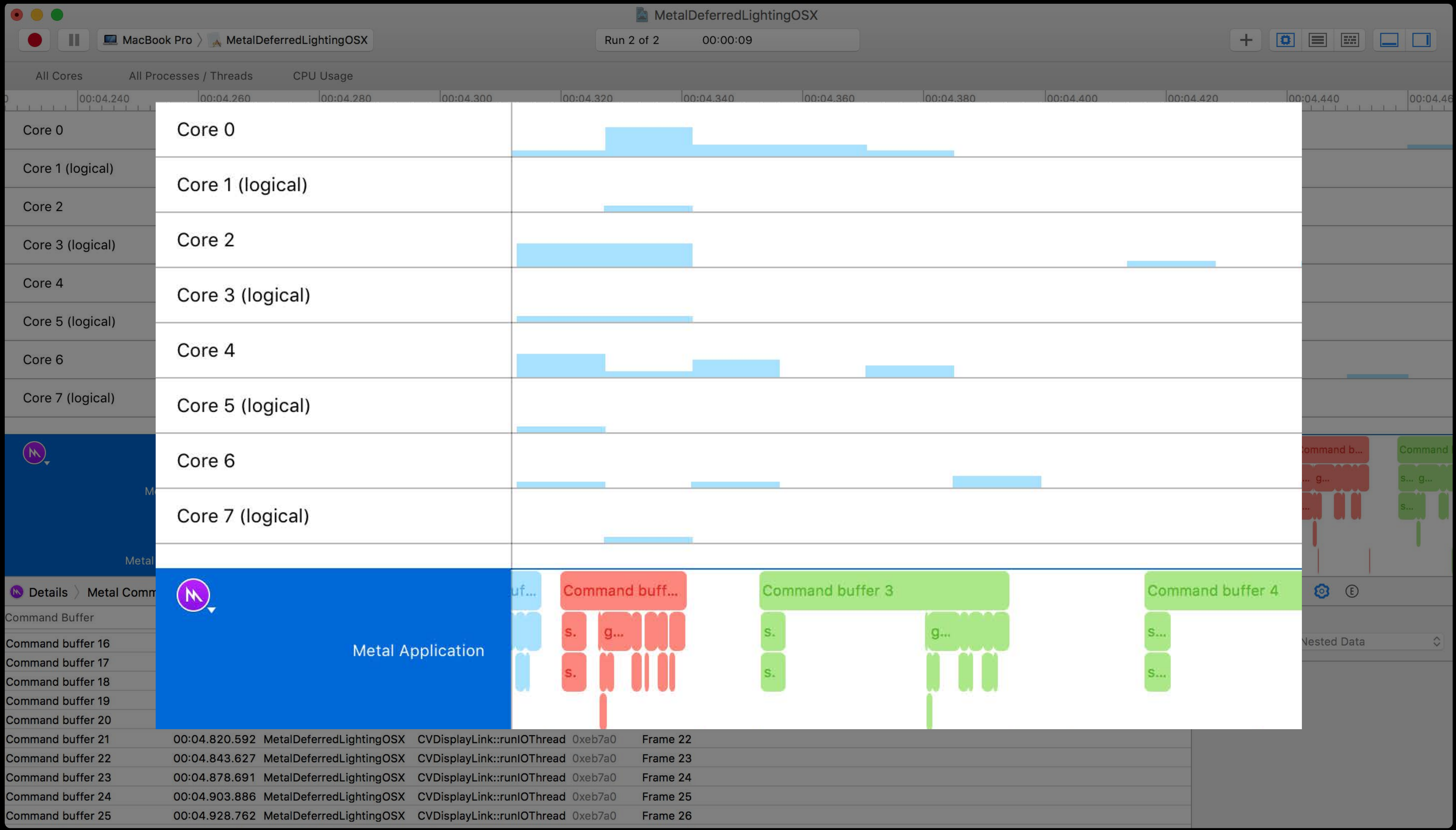
Integration with More Instruments



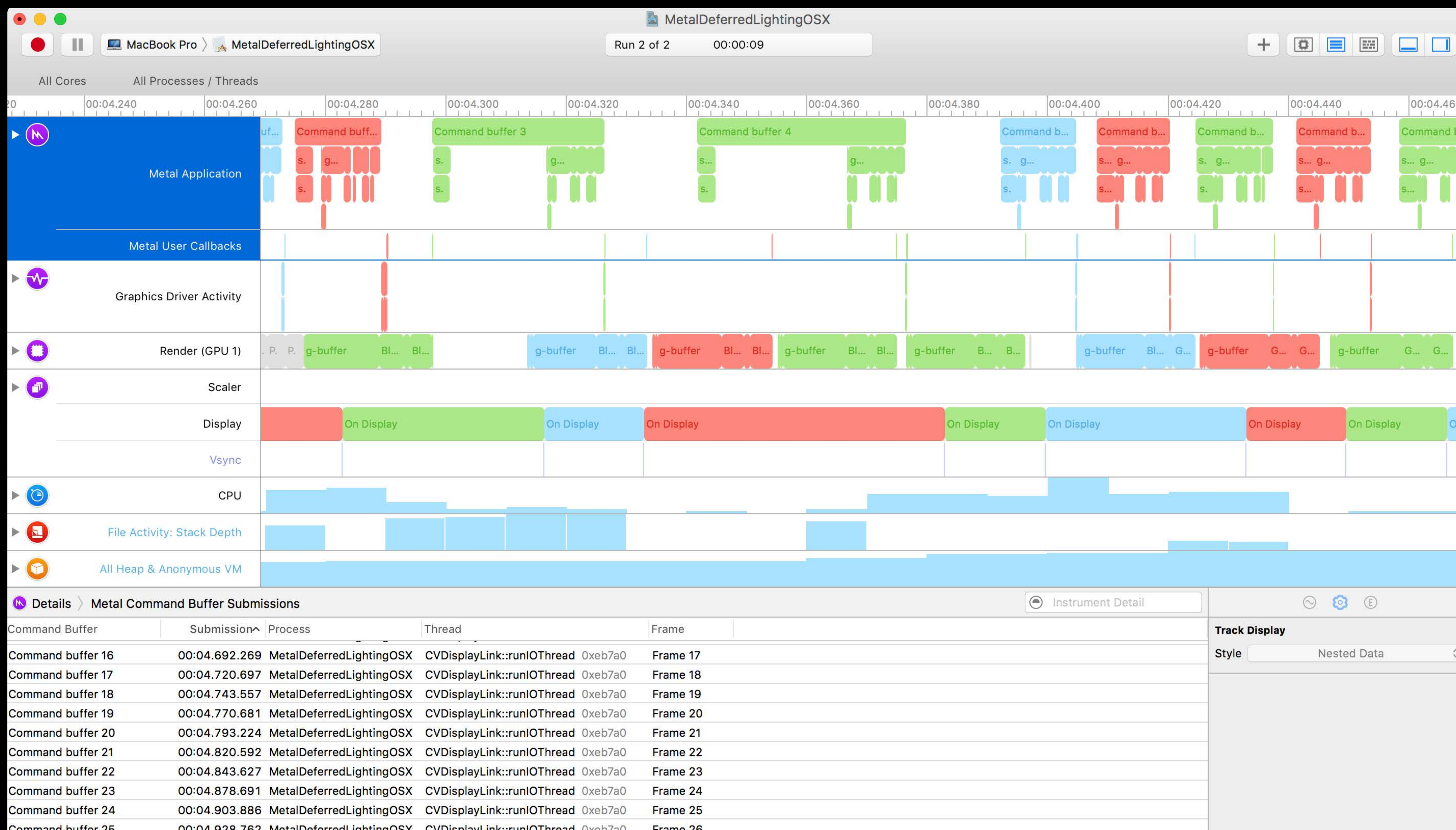
Integration with More Instruments



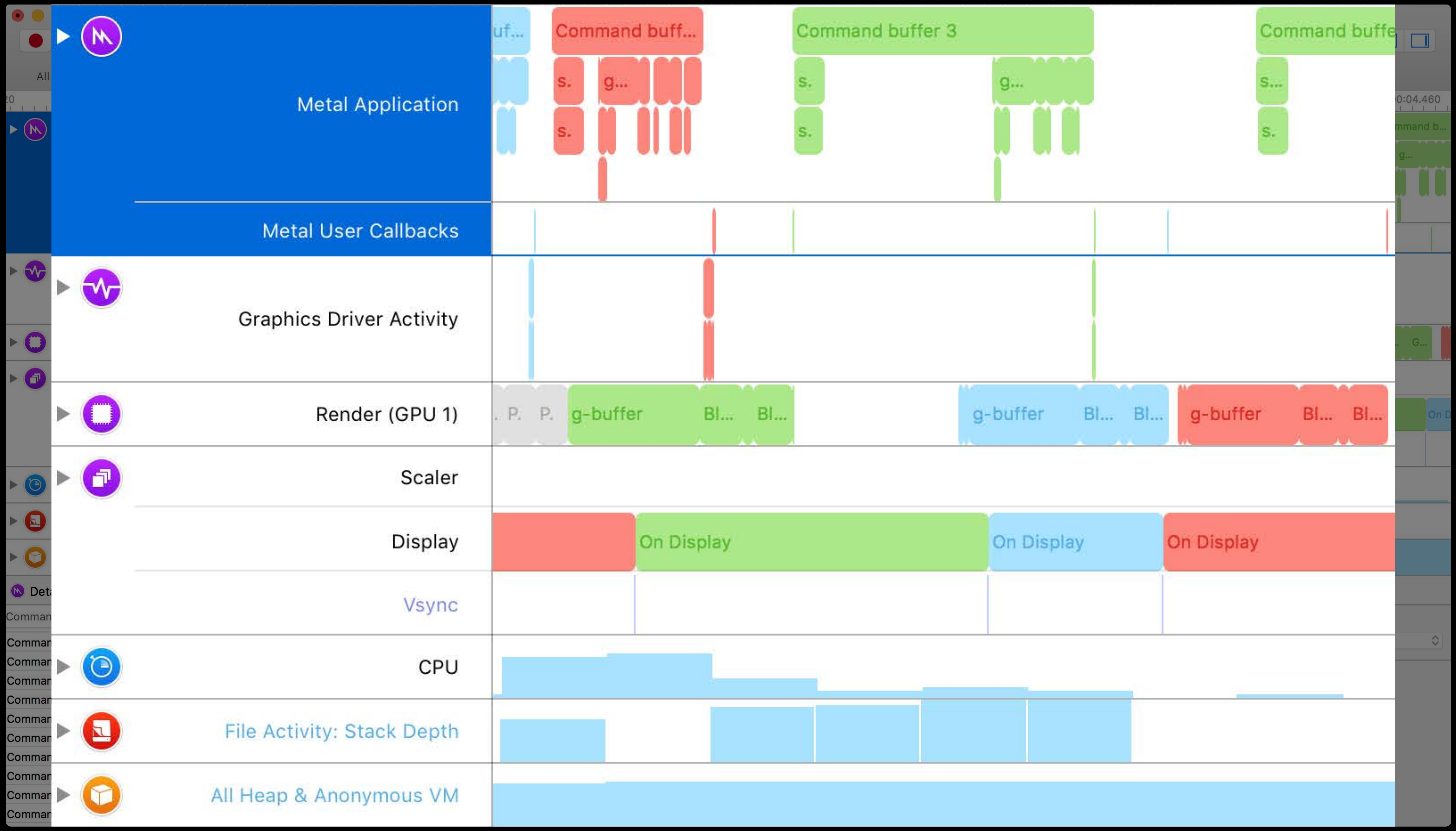
Integration with More Instruments



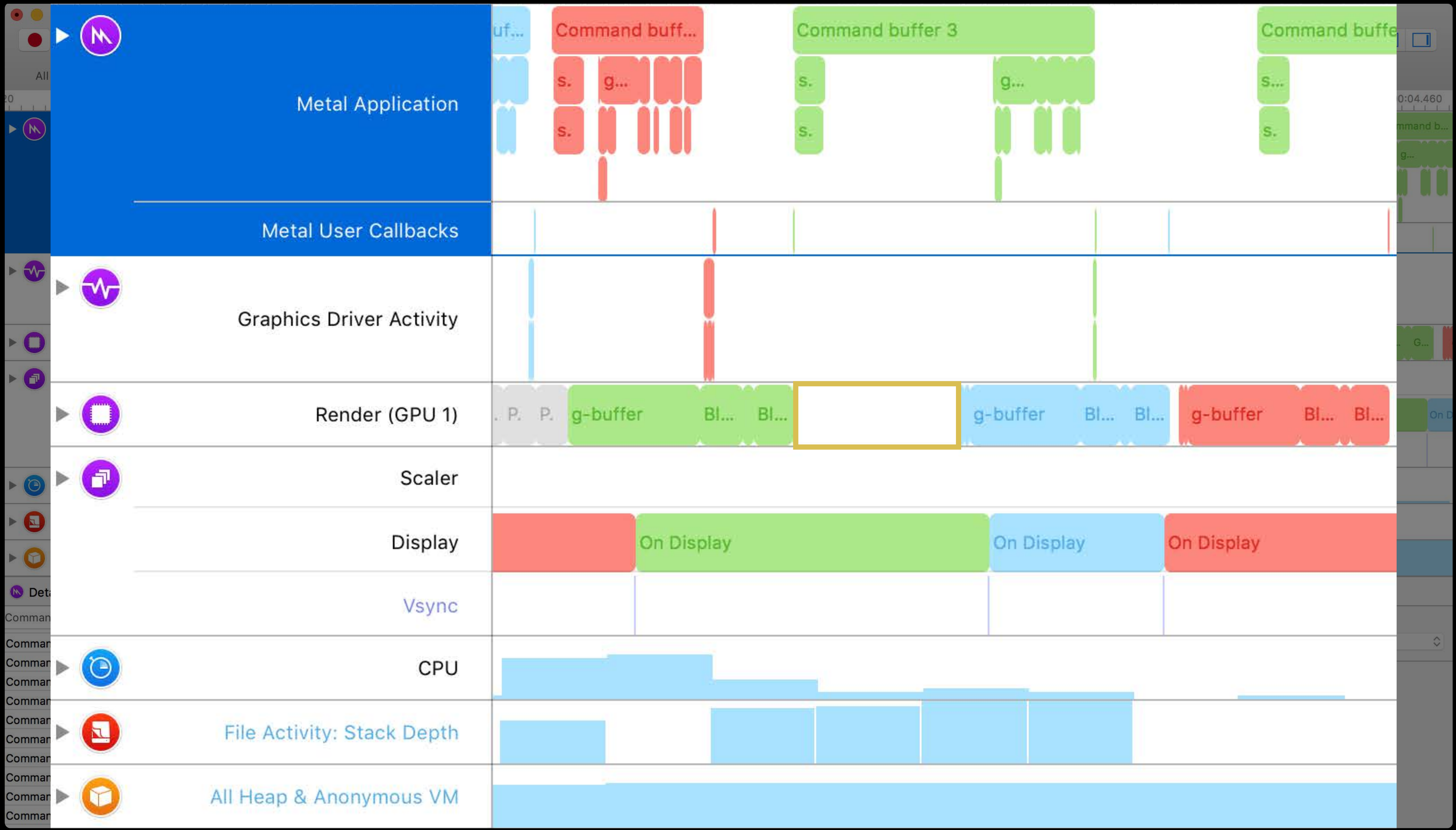
Integration with More Instruments



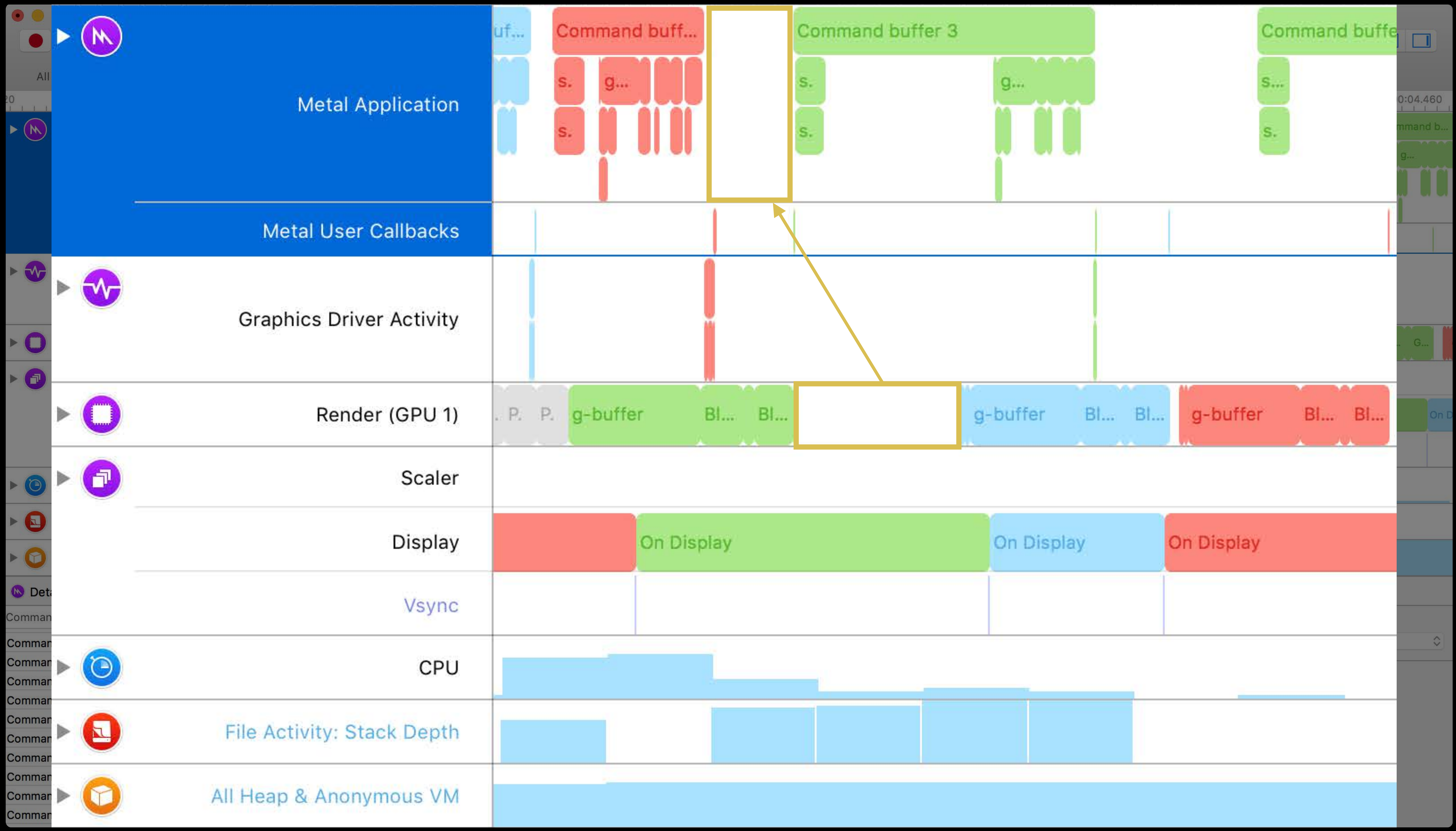
Integration with More Instruments



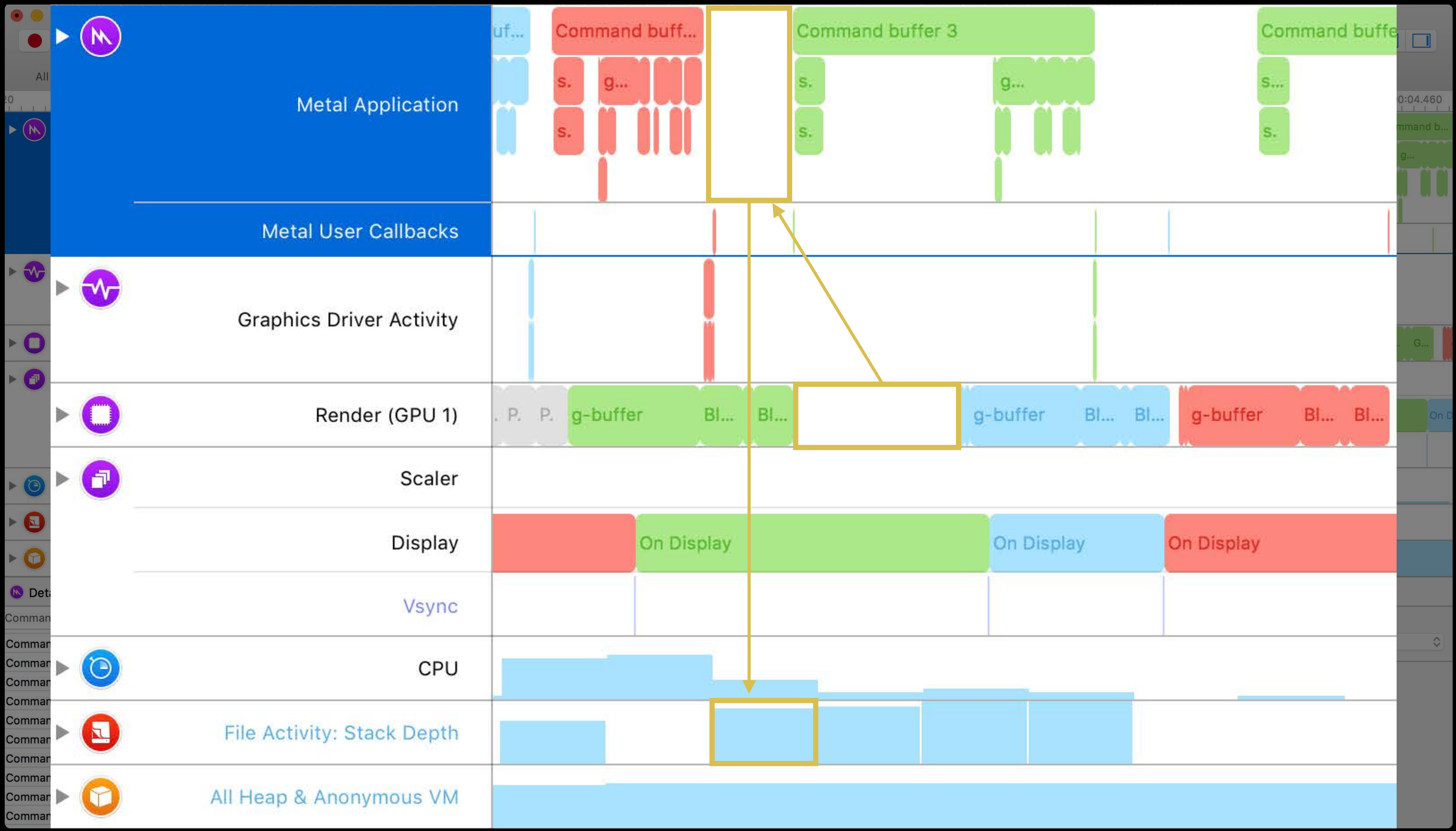
Integration with More Instruments



Integration with More Instruments

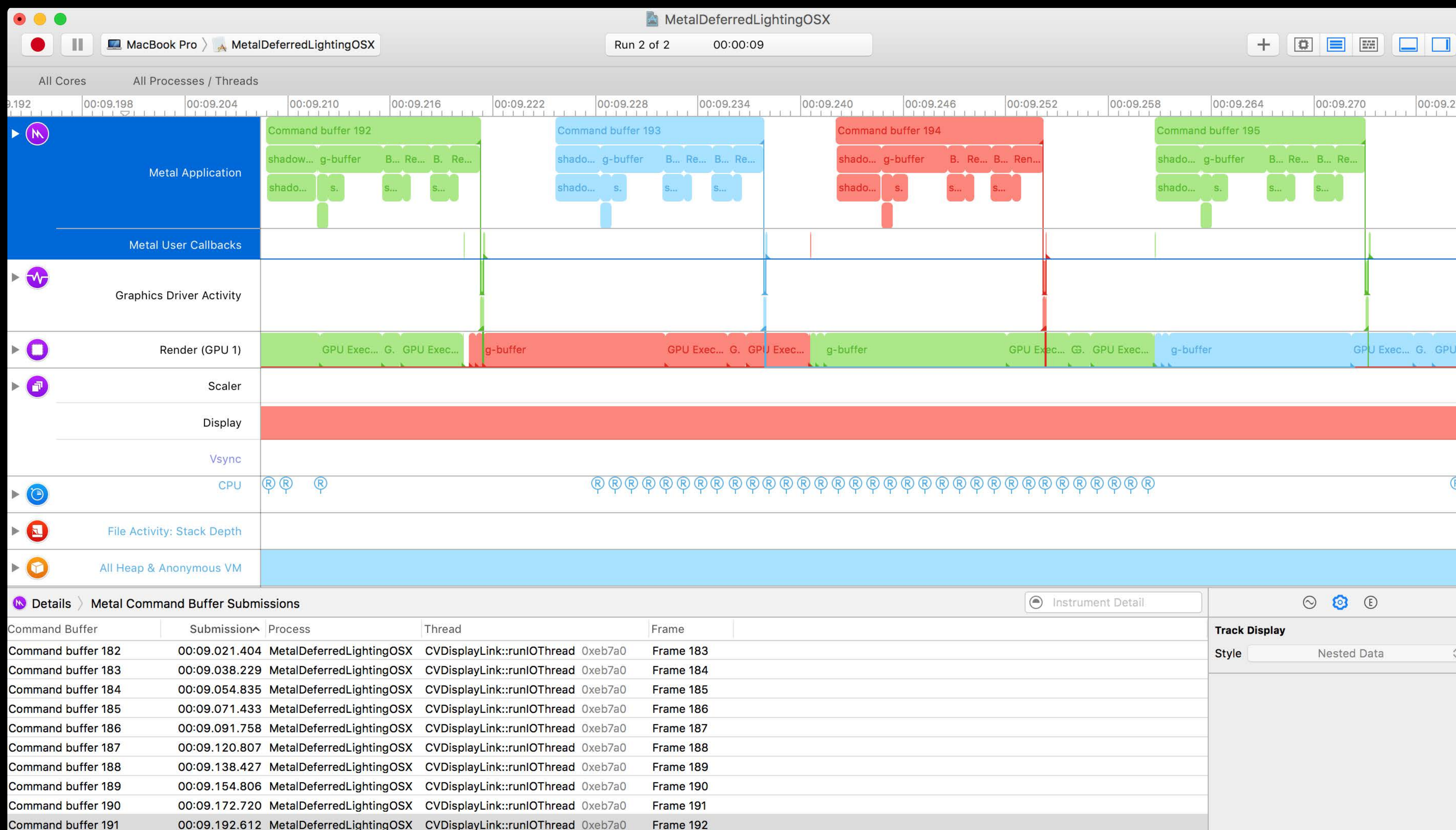


Integration with More Instruments



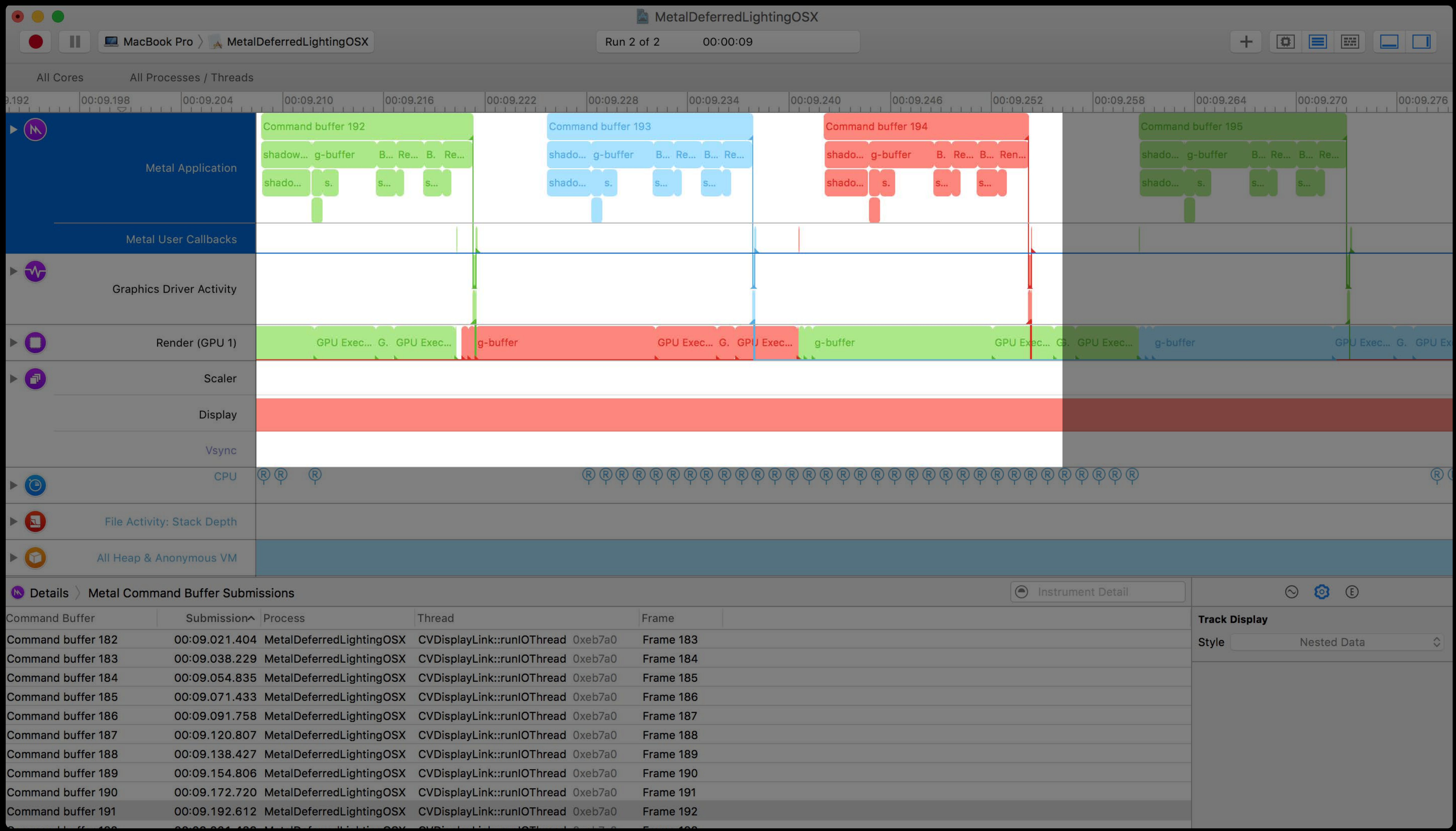
Navigation

NEW

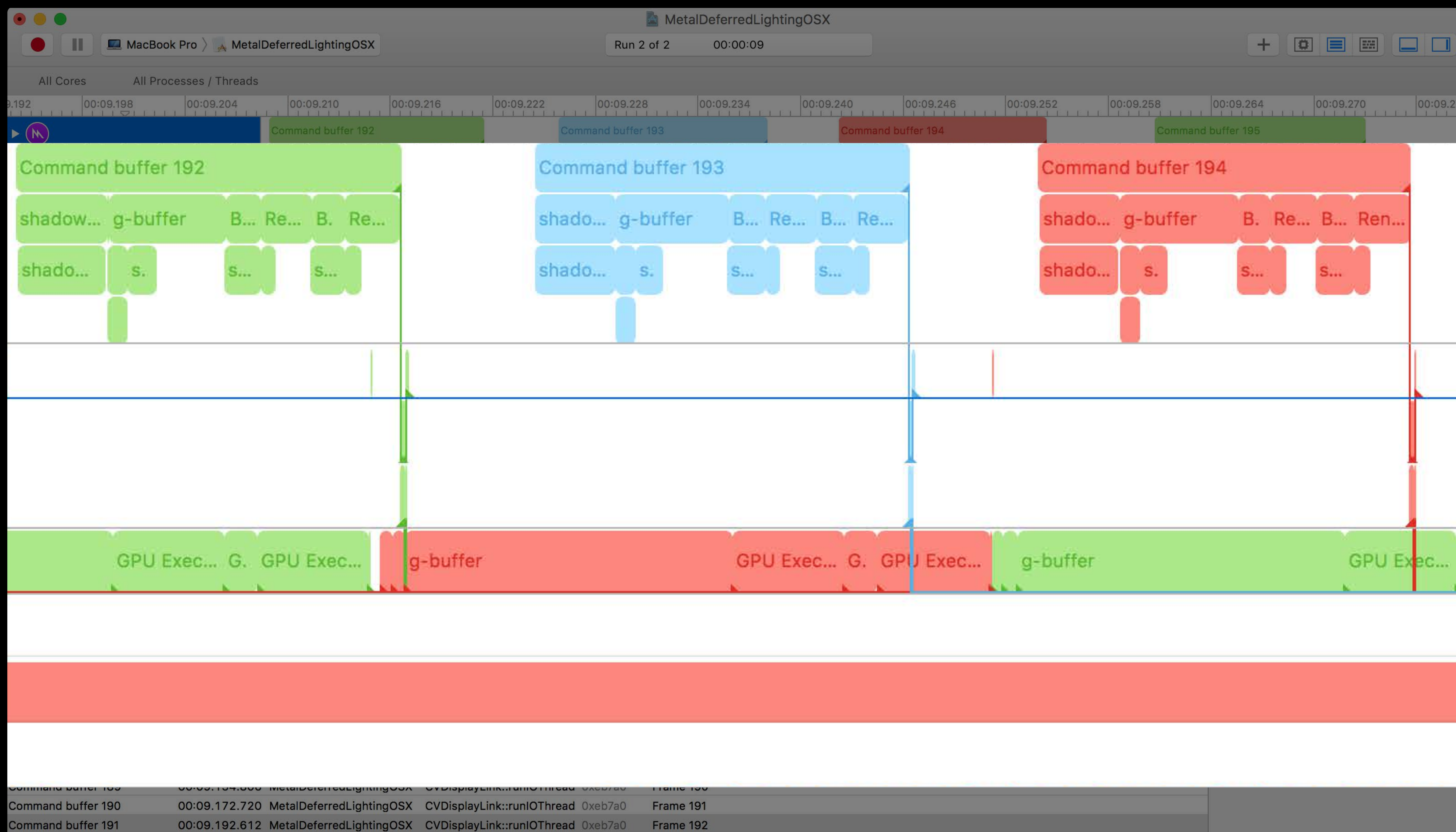


Navigation

NEW

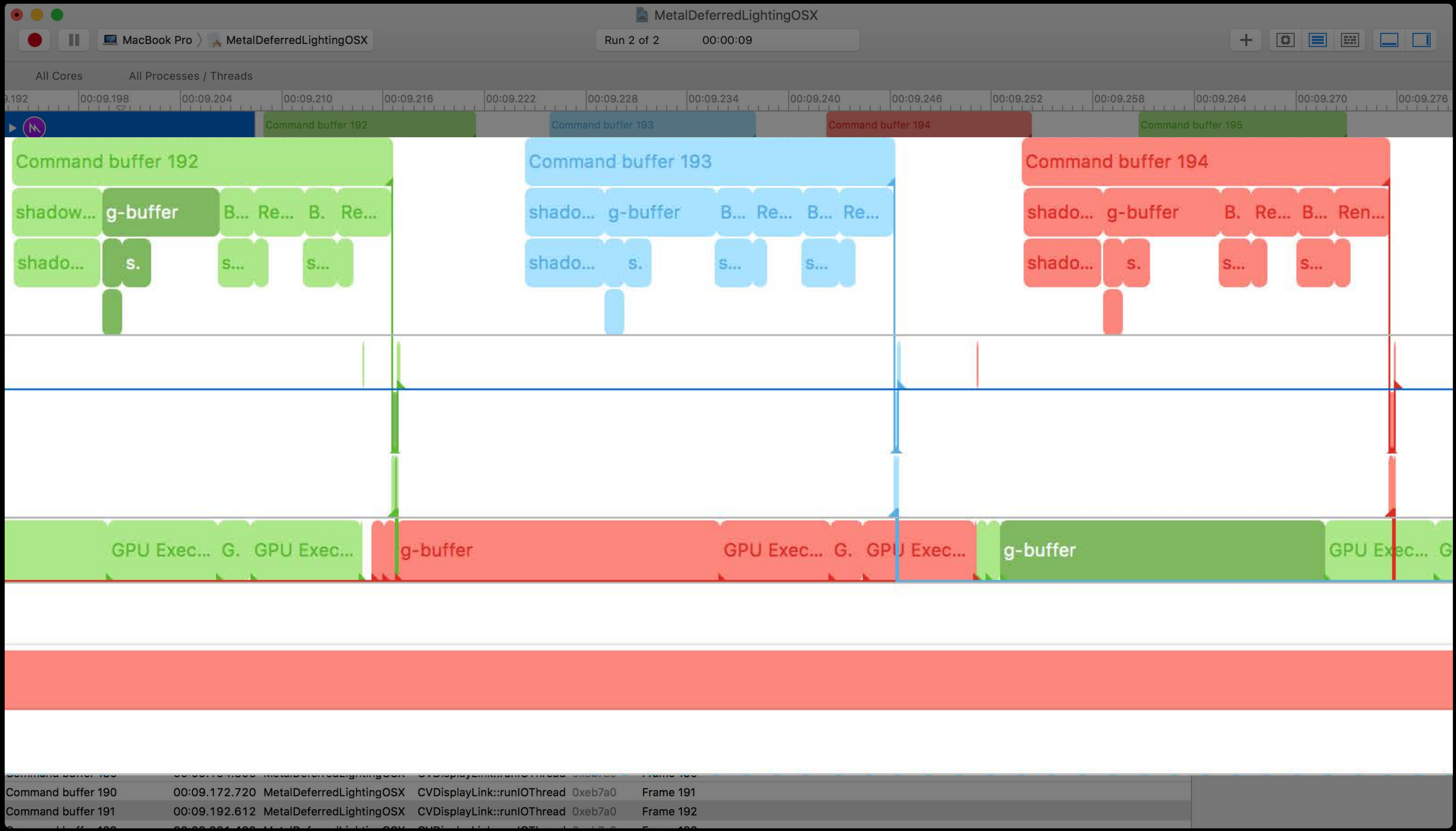


Navigation

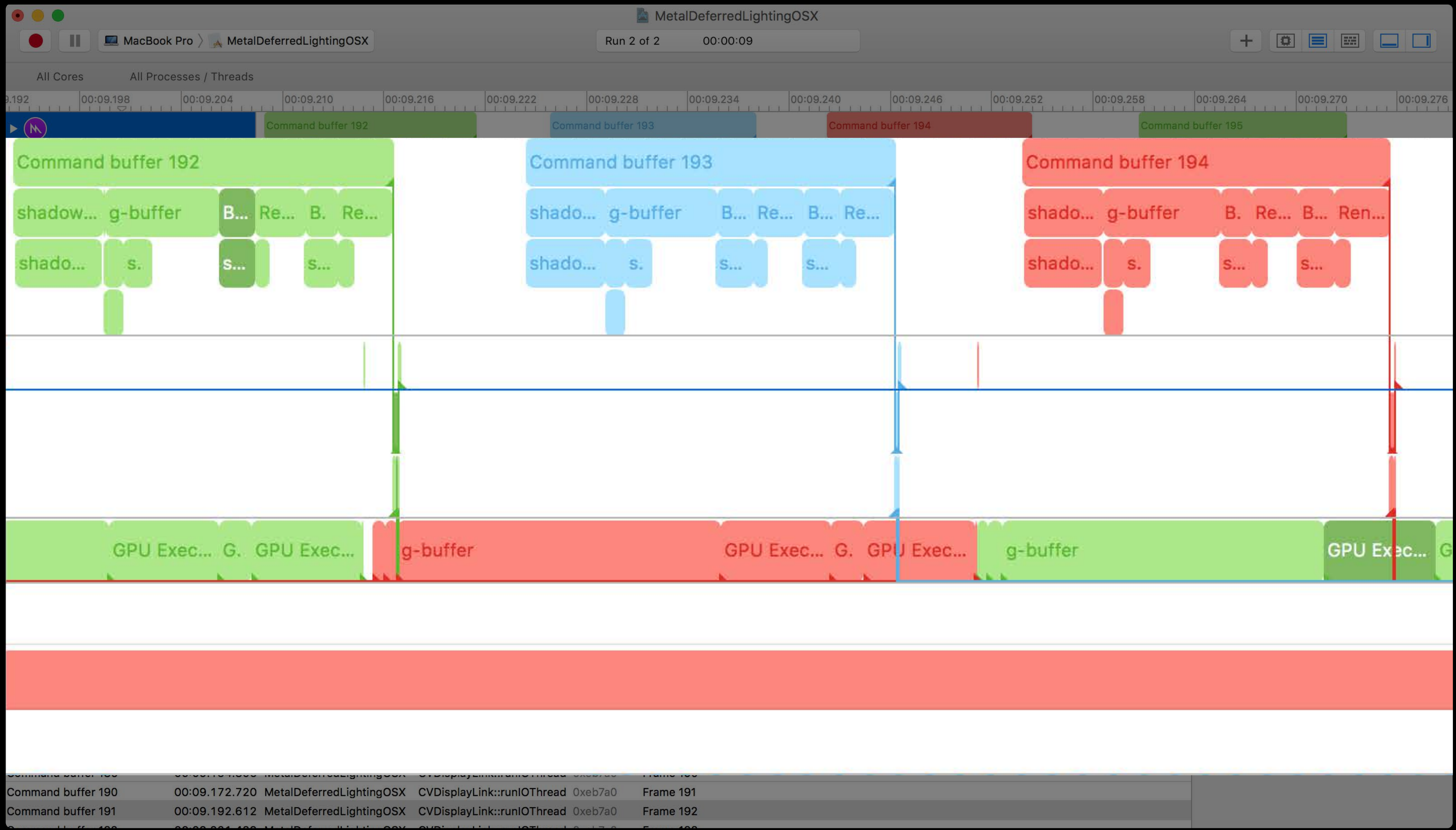


MetalDeferredLightingOSX
MacBook Pro
MetalDeferredLightingOSX
Run 2 of 2
00:00:09
All Cores
All Processes / Threads
Command buffer 192
Command buffer 193
Command buffer 194
Command buffer 195
g-buffer
GPU Exec...
Command buffer 190
00:09.172.720
MetalDeferredLightingOSX
CVDisplayLink::runIOThread
0xeb7a0
Frame 191
Command buffer 191
00:09.192.612
MetalDeferredLightingOSX
CVDisplayLink::runIOThread
0xeb7a0
Frame 192

Navigation

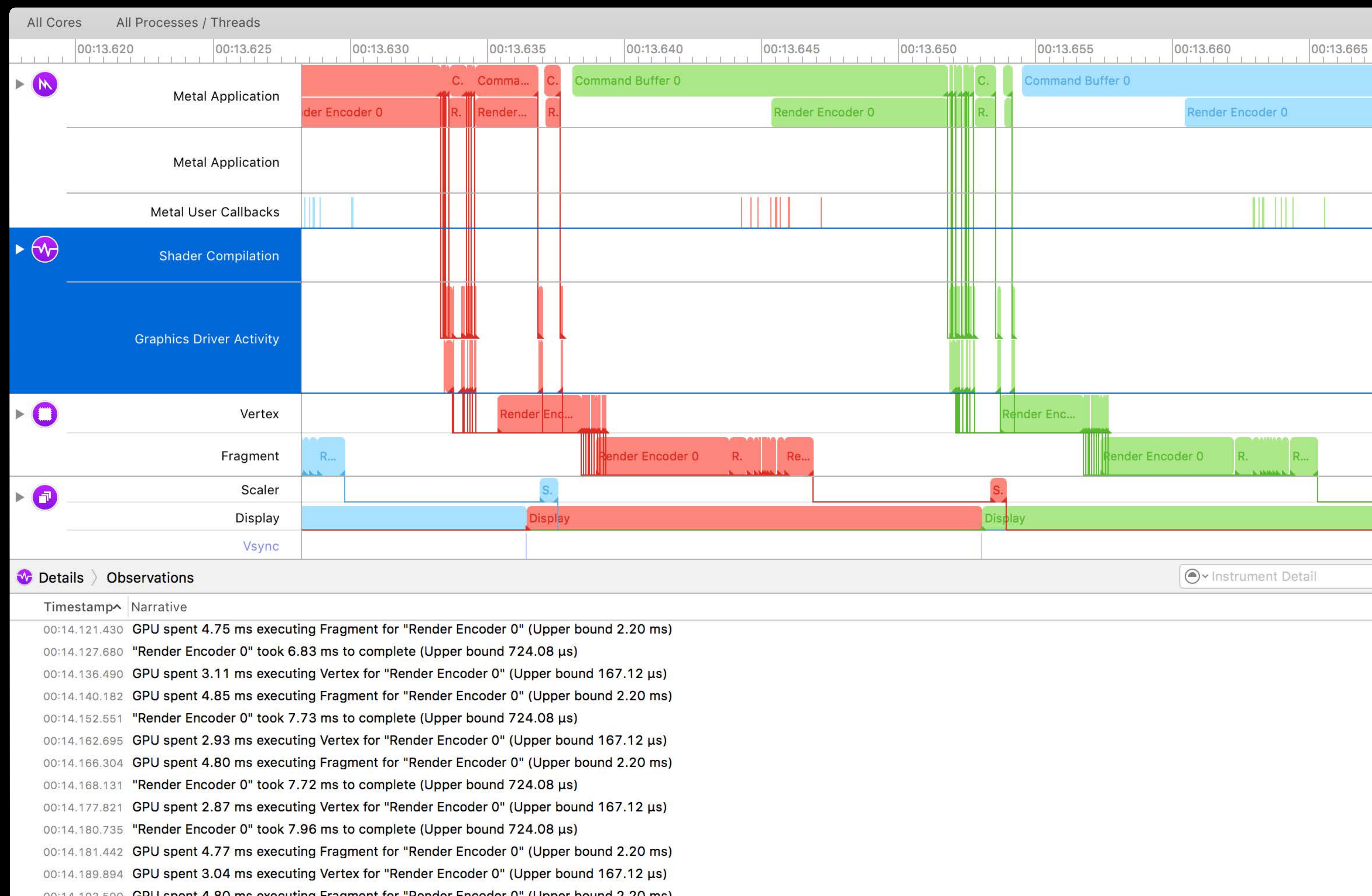

MetalDeferredLightingOSX
MacBook Pro
MetalDeferredLightingOSX
Run 2 of 2
00:00:09
All Cores
All Processes / Threads
Command buffer 192
Command buffer 193
Command buffer 194
Command buffer 195
shadow...
g-buffer
GPU Exec...
Command buffer 190 00:09.172.720 MetalDeferredLightingOSX CVDisplayLink::runIOThread 0xeb7a0 Frame 191
Command buffer 191 00:09.192.612 MetalDeferredLightingOSX CVDisplayLink::runIOThread 0xeb7a0 Frame 192

Navigation


MetalDeferredLightingOSX
MacBook Pro
MetalDeferredLightingOSX
Run 2 of 2
00:00:09
All Cores
All Processes / Threads
Command buffer 192
Command buffer 193
Command buffer 194
Command buffer 195
shadow...
g-buffer
GPU Exec...
Command buffer 190
00:09.172.720
MetalDeferredLightingOSX
CVDisplayLink::runIOThread
0xeb7a0
Frame 191
Command buffer 191
00:09.192.612
MetalDeferredLightingOSX
CVDisplayLink::runIOThread
0xeb7a0
Frame 192

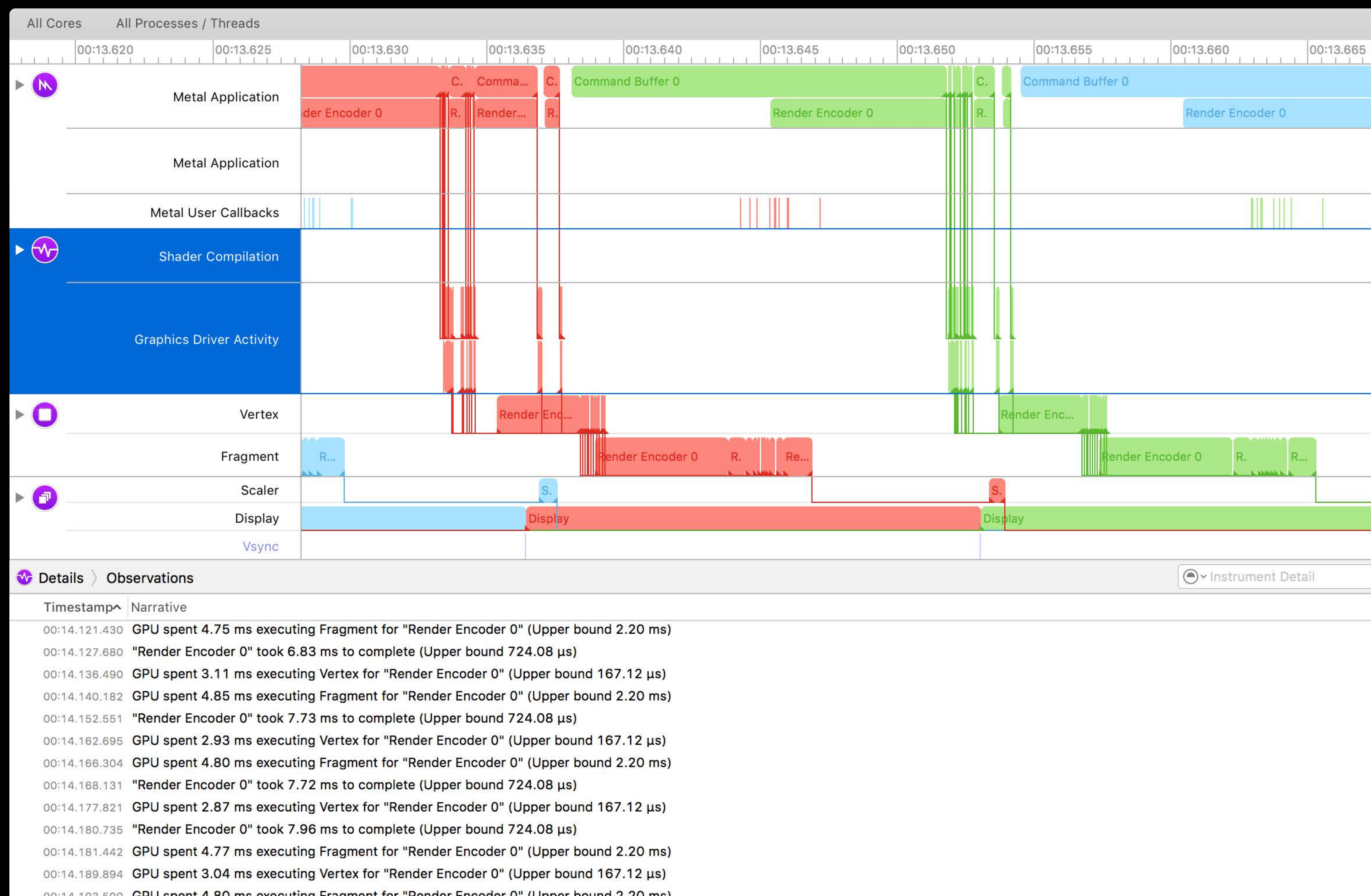
Performance Observations

NEW



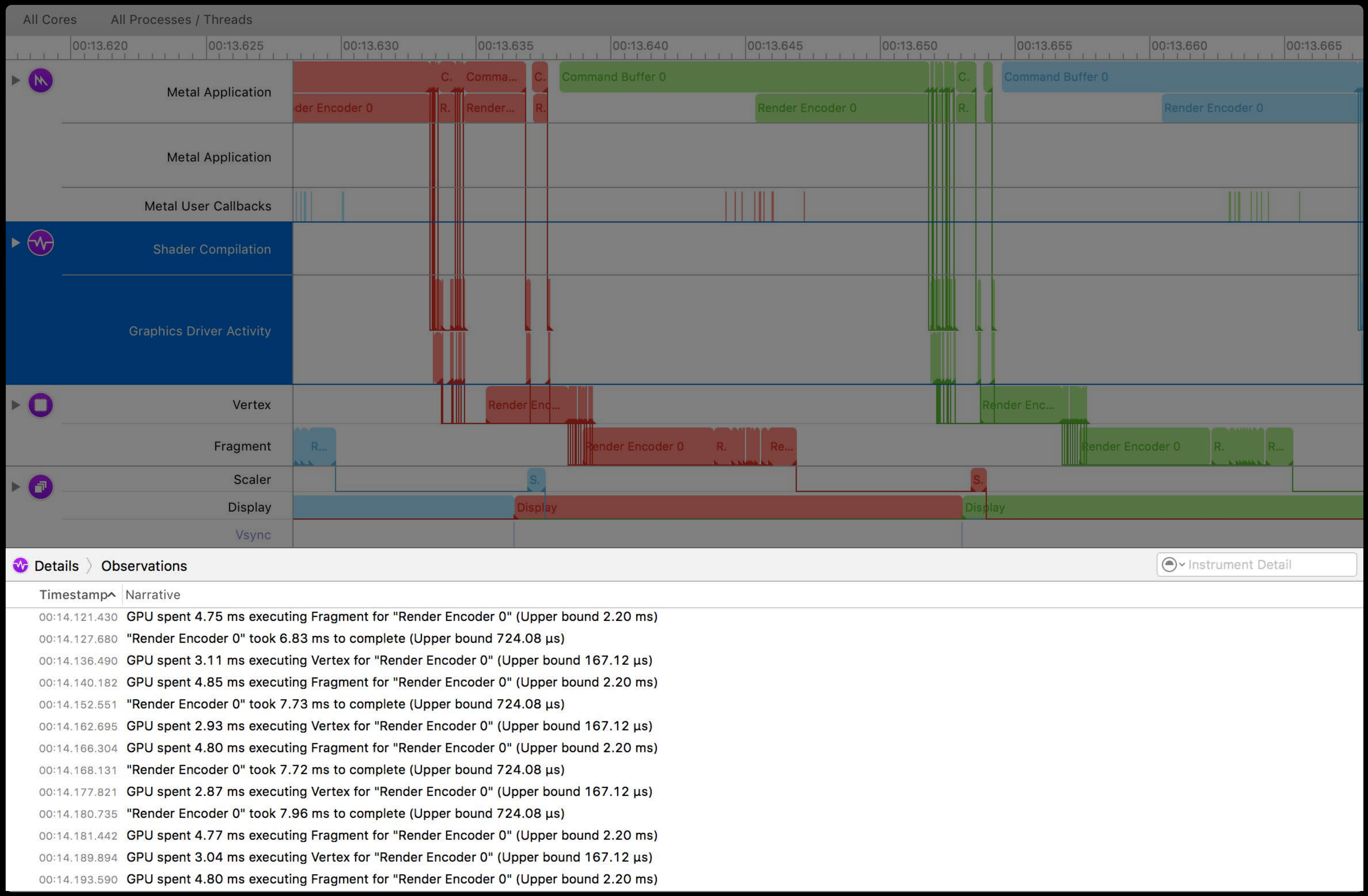
Performance Observations

NEW

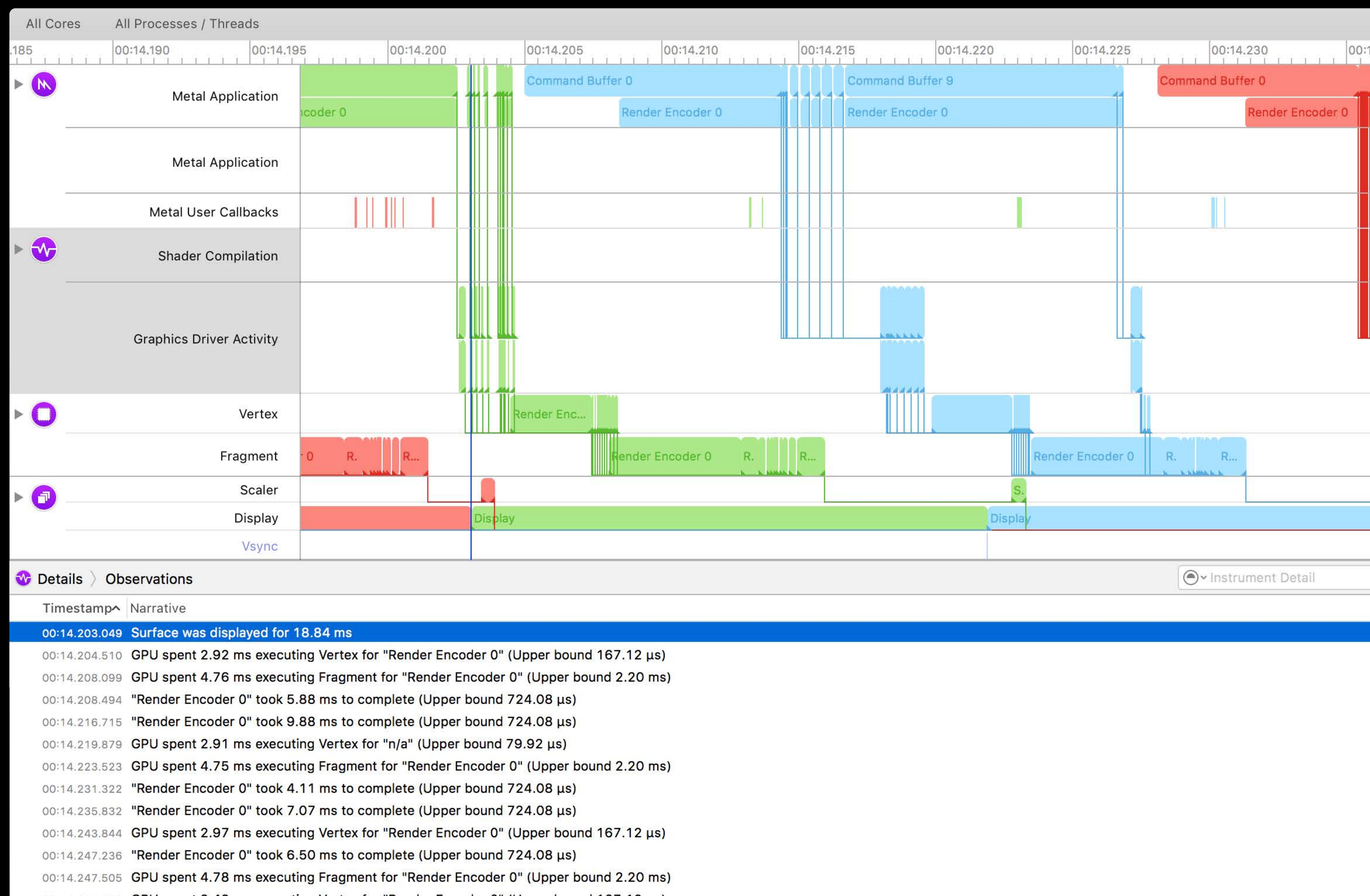


Performance Observations

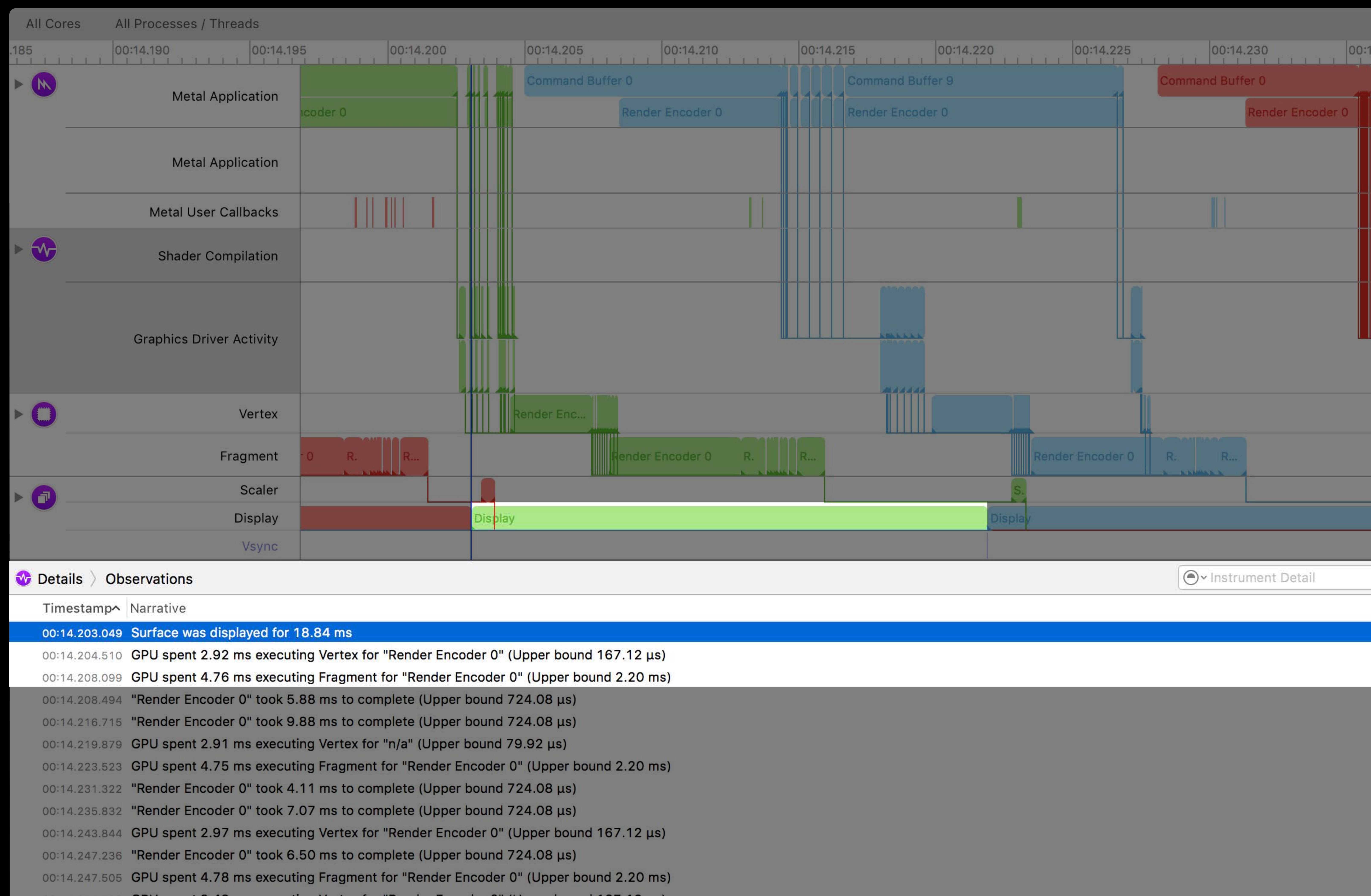
NEW



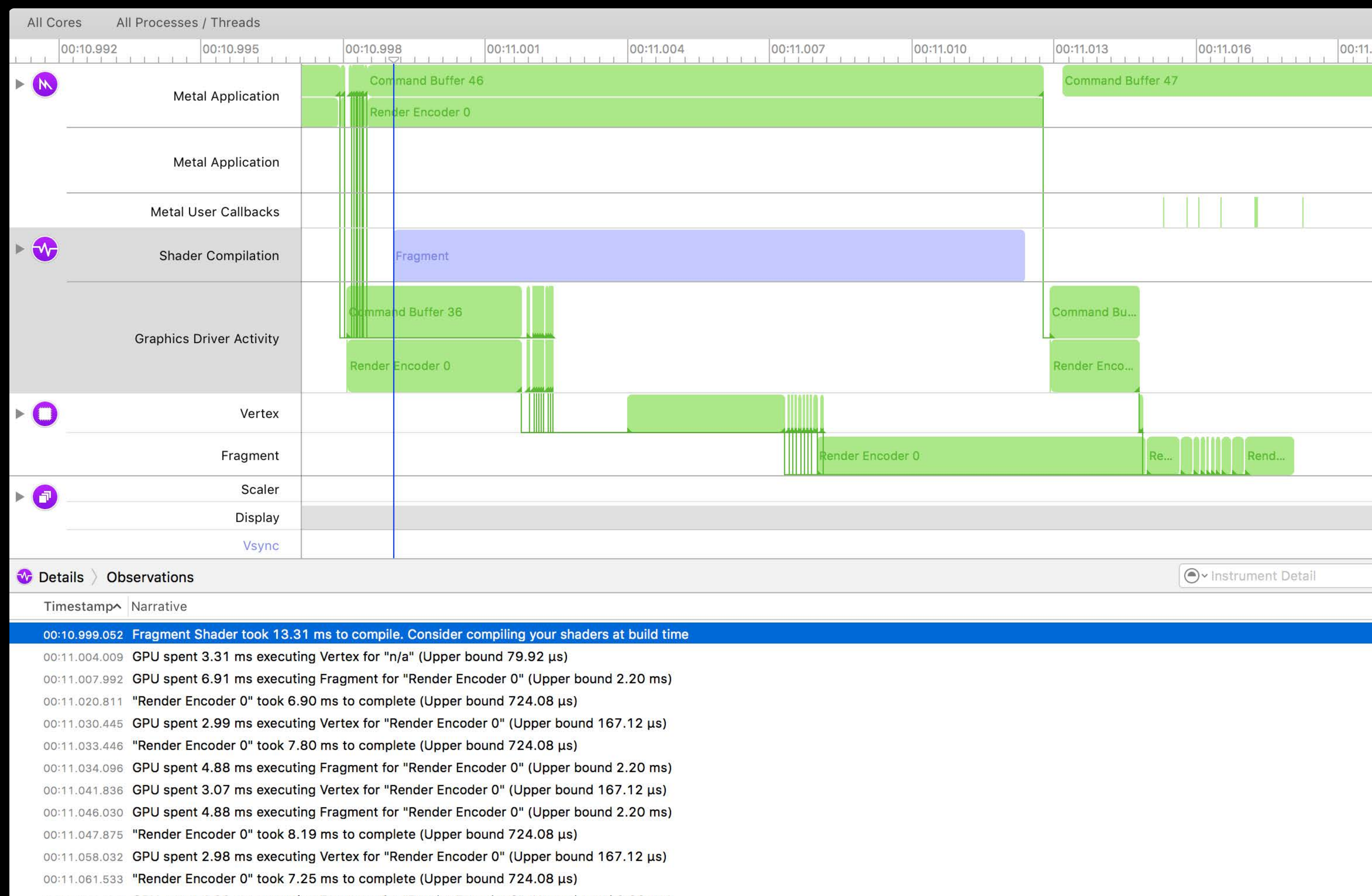
Performance Observations



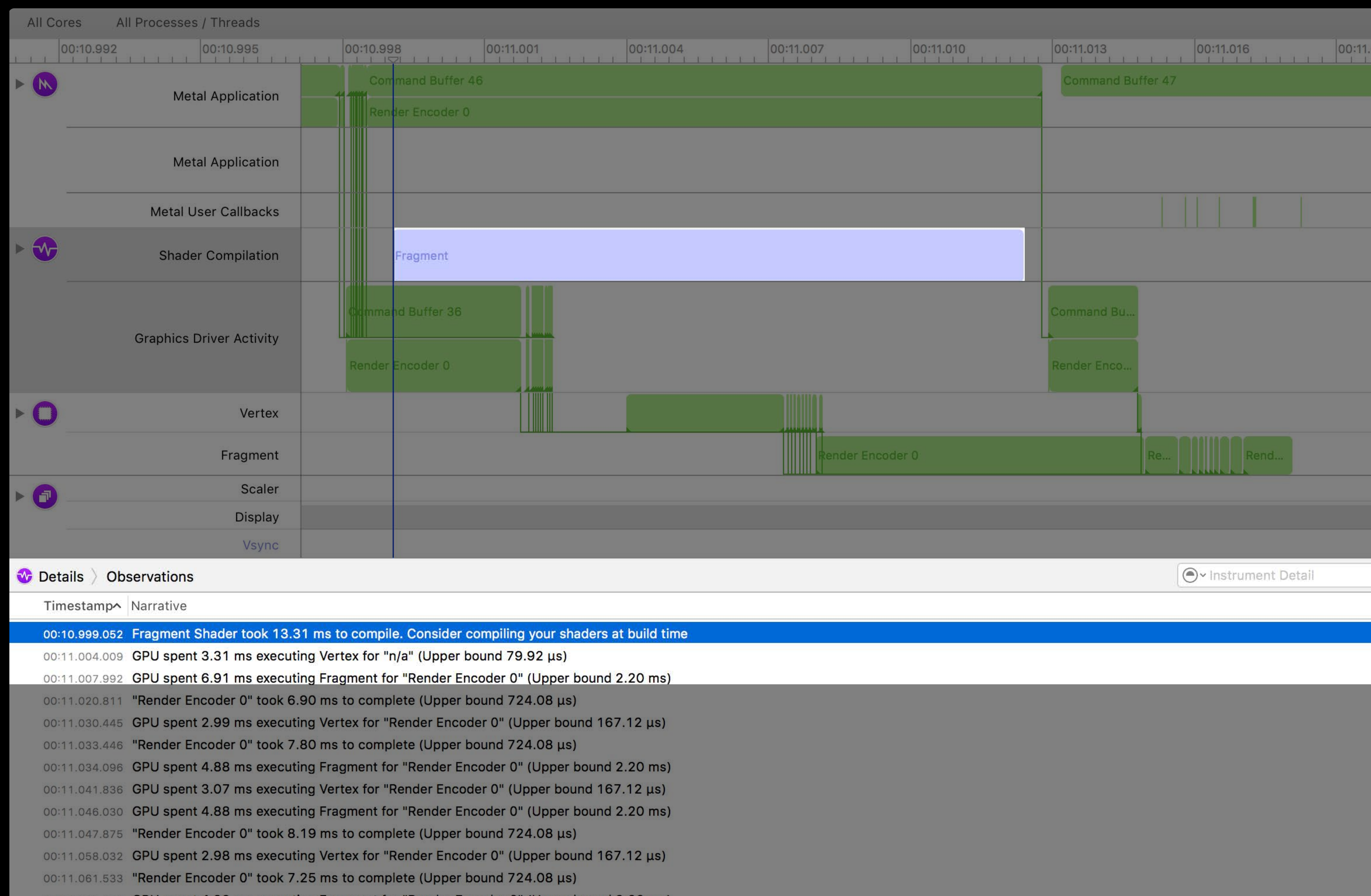
Performance Observations



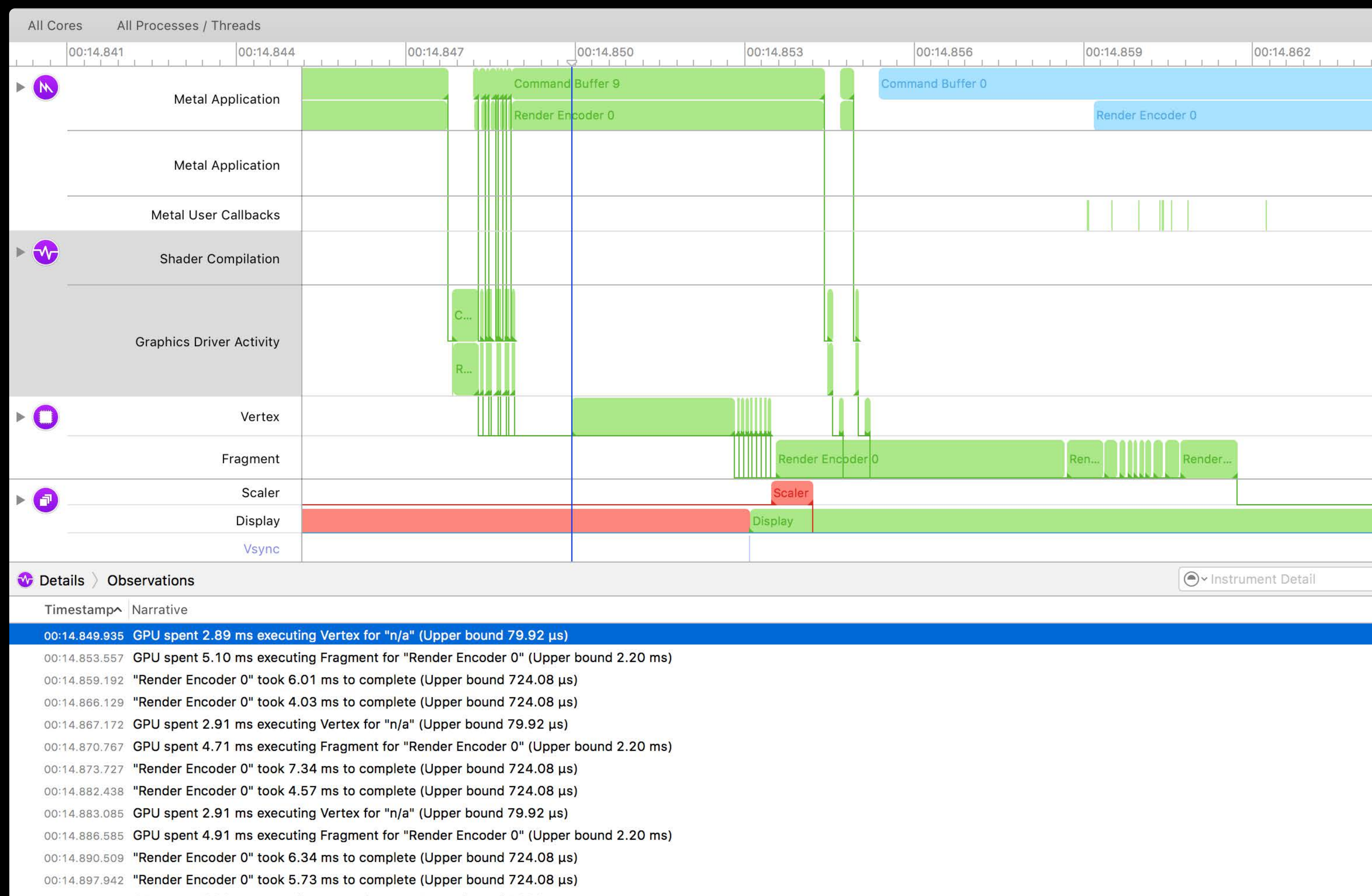
Performance Observations



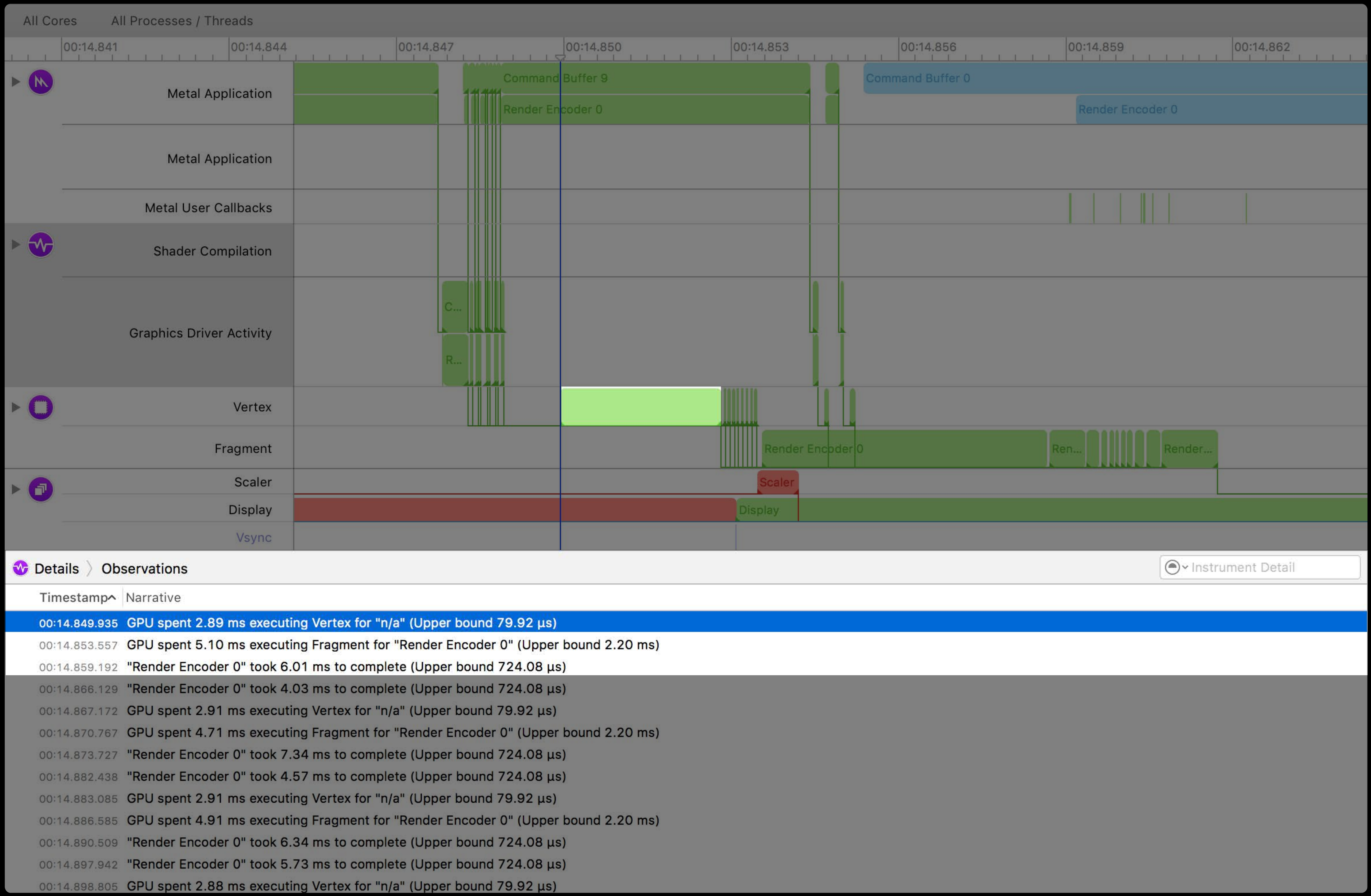
Performance Observations



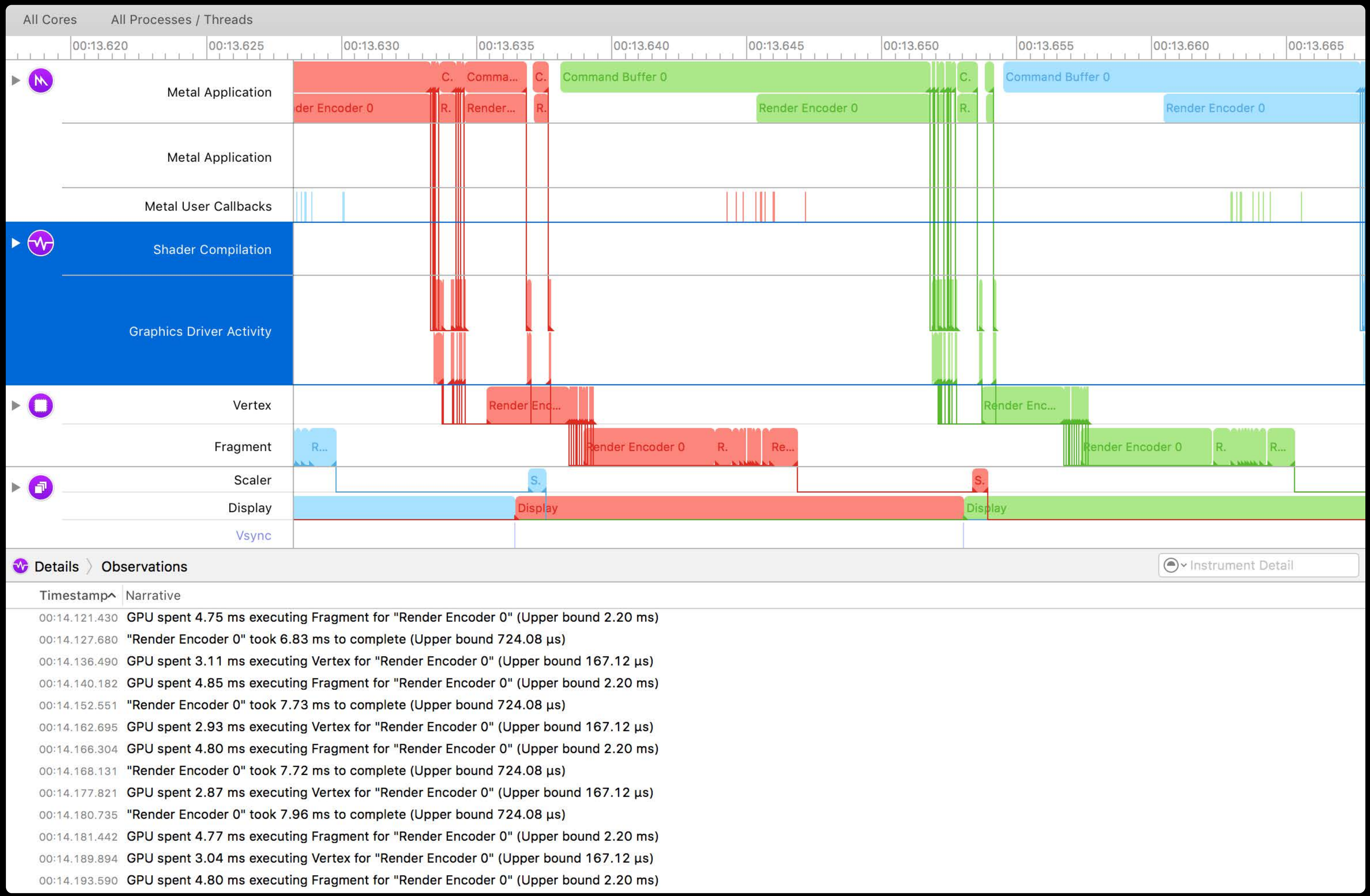
Performance Observations



Performance Observations



Performance Observations



Demo

GPU Frame Debugger

What's New in Metal Tools

Metal System Trace

GPU Overrides

GPU Frame Debugger

What's New in Metal Tools

Metal System Trace

GPU Overrides

GPU Frame Debugger

- Extended Validation

What’s New in Metal Tools

Metal System Trace

GPU Overrides

GPU Frame Debugger

- Extended Validation
- Metal Library Projects

Summary

Summary

Tessellation

Resource Heaps and Memoryless Render Targets

Improved Tools

Summary

Tessellation

Resource Heaps and Memoryless Render Targets

Improved Tools

Summary

Tessellation

Resource Heaps and Memoryless Render Targets

Improved Tools

Function Specialization and
Function Resource Read-Writes

Wide Color and Texture Assets

Additions to Metal Performance Shaders

More Information

https://developer.apple.com/wwdc16/604

Related Sessions

Adopting Metal, Part 1	Nob Hill	Tuesday 1:40PM
Adopting Metal, Part 2	Nob Hill	Tuesday 3:00PM
What's New in Metal, Part 2	Pacific Heights	Wednesday 1:40PM
Advanced Metal Shader Optimization	Pacific Heights	Wednesday 3:00PM

Labs

Metal Lab	Graphics, Games, and Media Lab B	Thursday 12:00PM

WWDC16