Graphics and Games

#WWDC17

Introducing Metal 2

Session 601

Michal Valient, GPU Software Engineer
Richard Schreyer, GPU Software Engineer

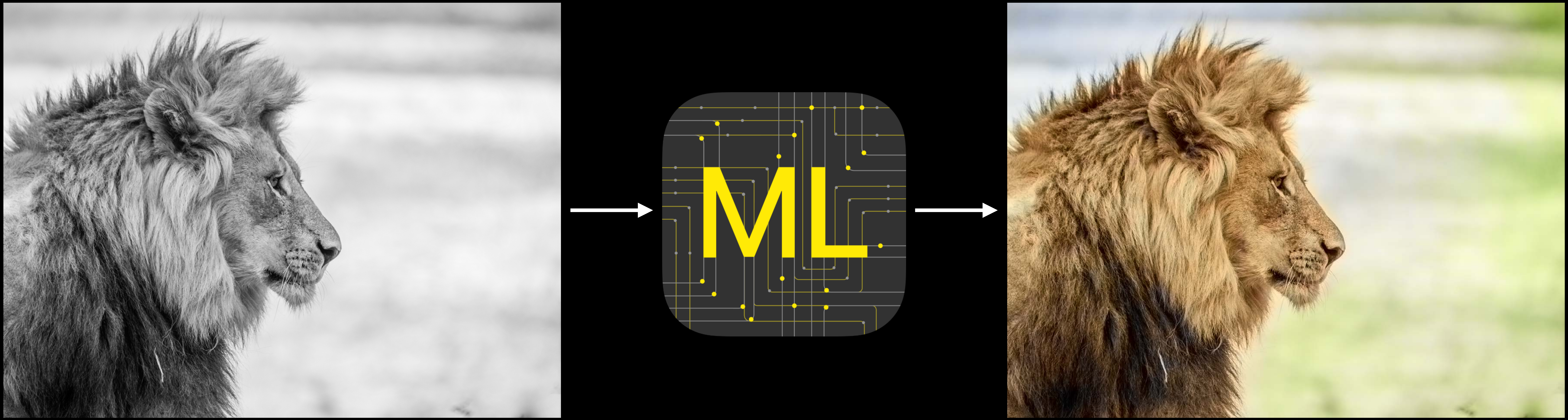
Make expensive things happen once

GPU in the driving seat

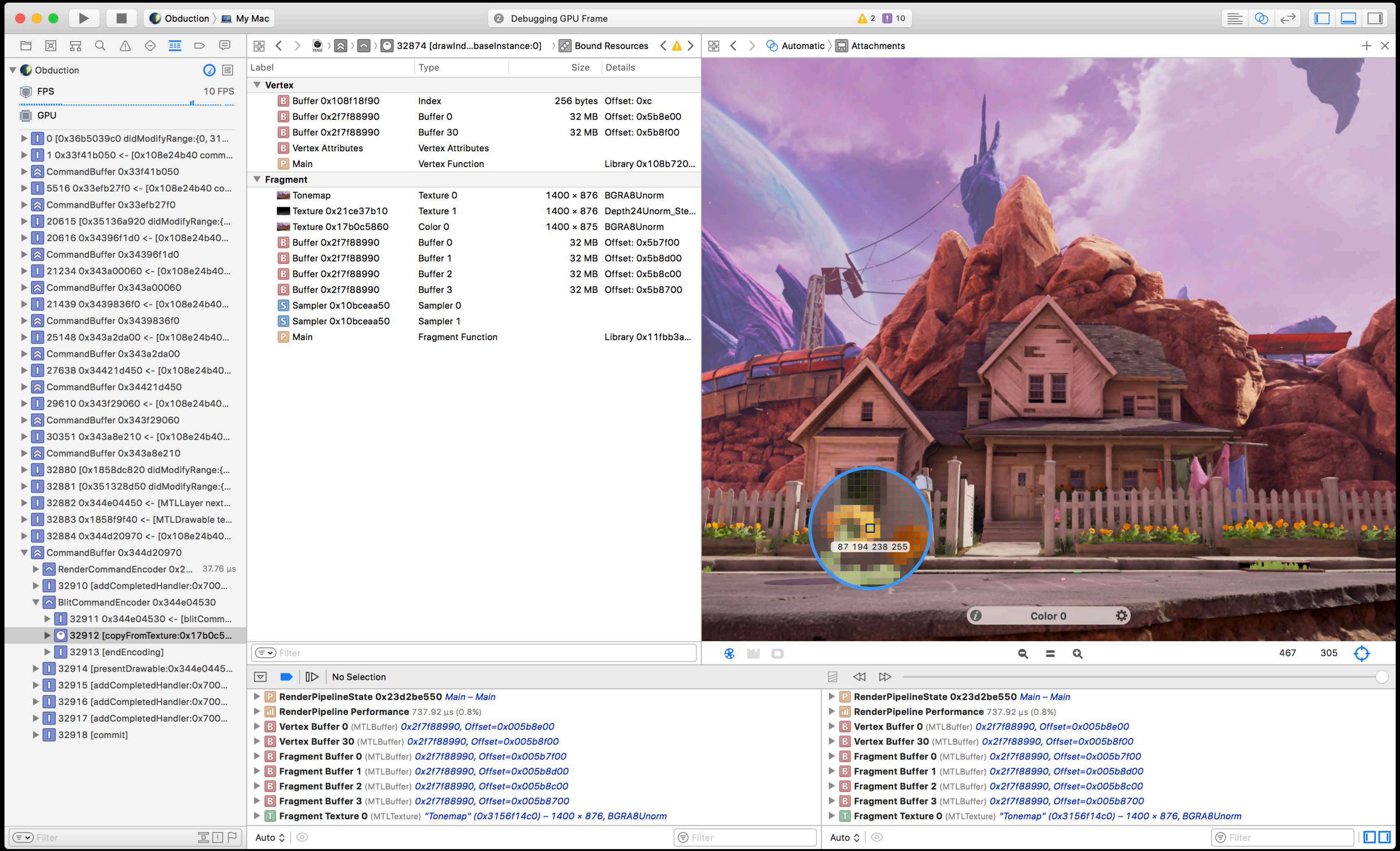
New experiences

VR with Metal 2 | Hall 3 | Wednesday 10:00AM

Using Metal 2 for Compute | Grand Ballroom A | Thursday 4:10PM



Using Metal 2 for Compute   Grand Ballroom A   Thursday 4:10PM



Metal 2 Optimization and Debugging

Executive Ballroom

Thursday 3:10PM

Introducing Metal 2

Agenda

Introducing Metal 2
Agenda

Argument Buffers

Introducing Metal 2
Agenda

Argument Buffers

Raster Order Groups

Introducing Metal 2

Agenda

Argument Buffers

Raster Order Groups

ProMotion Displays

Introducing Metal 2
Agenda

Argument Buffers

Raster Order Groups

ProMotion Displays

Direct to Display

Introducing Metal 2

Agenda

Argument Buffers

Raster Order Groups

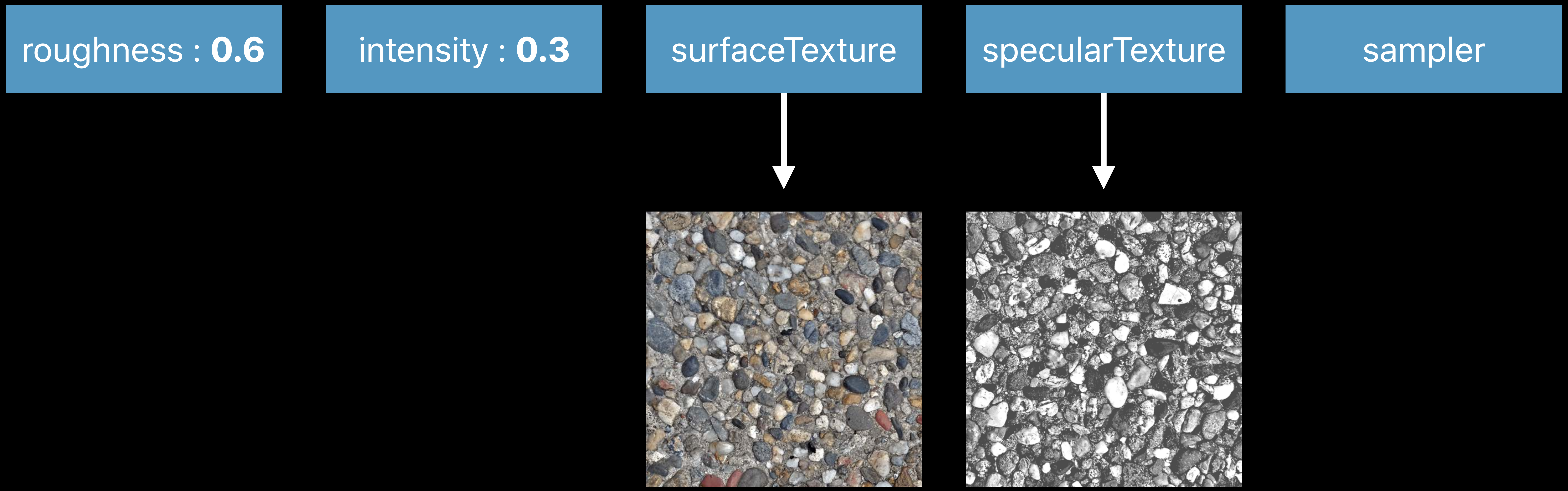
ProMotion Displays

Direct to Display

Everything Else

Argument Buffers

Material Example



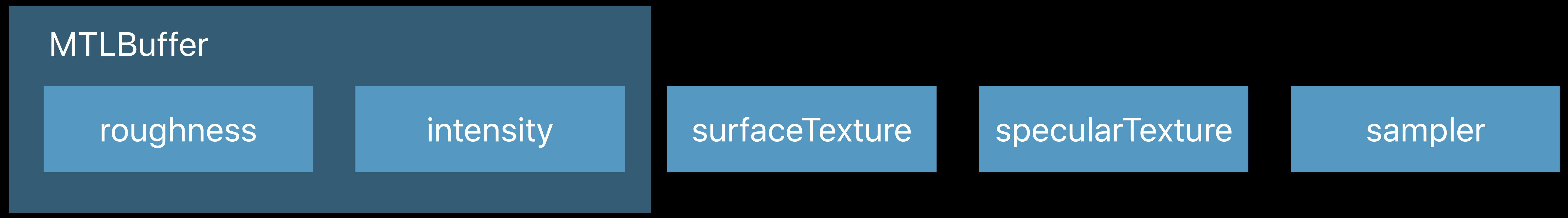
Material Example

roughness | intensity | surfaceTexture | specularTexture | sampler

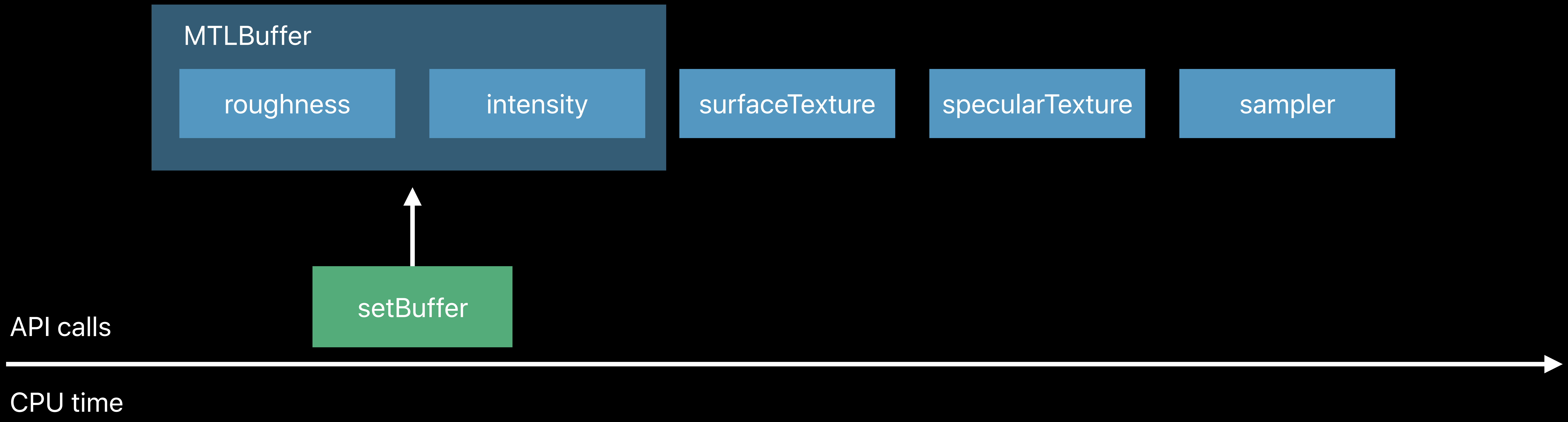
Traditional Argument Model

roughness	intensity	surfaceTexture	specularTexture	sampler

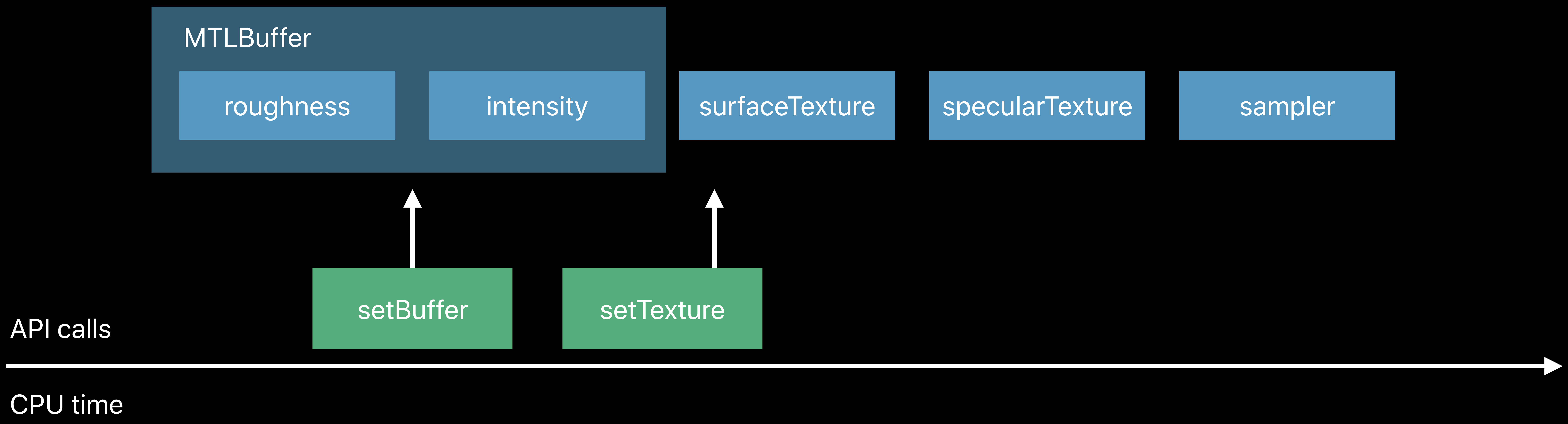
Traditional Argument Model



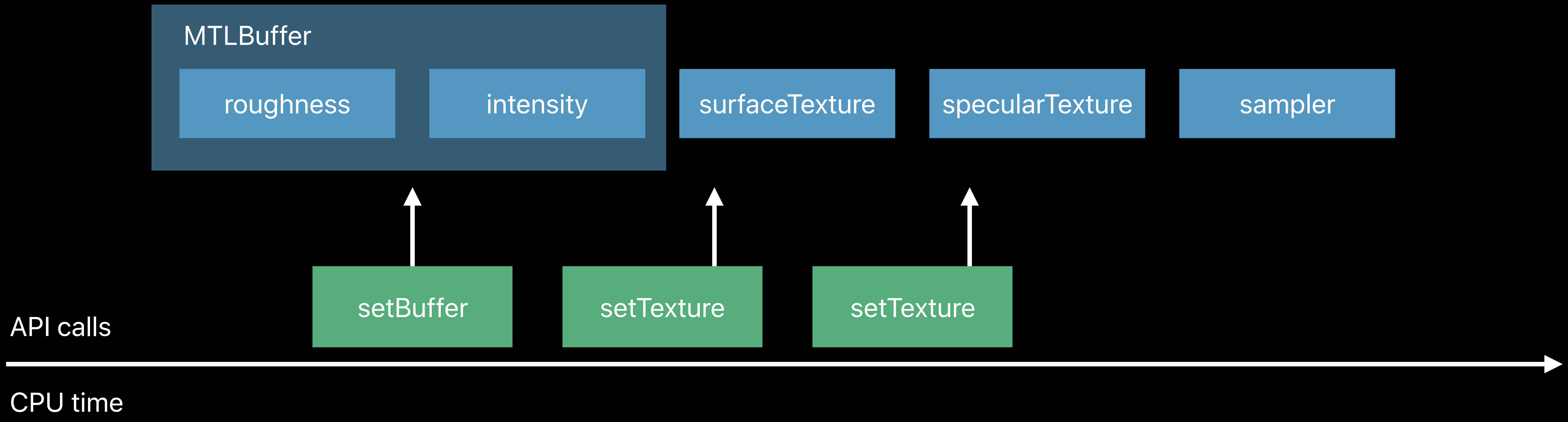
Traditional Argument Model



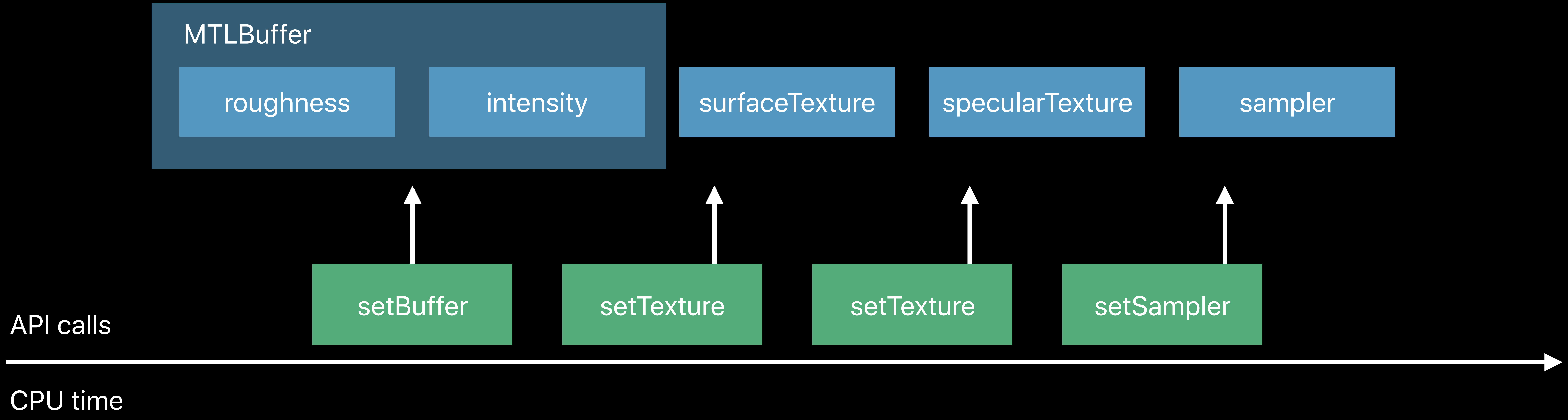
Traditional Argument Model



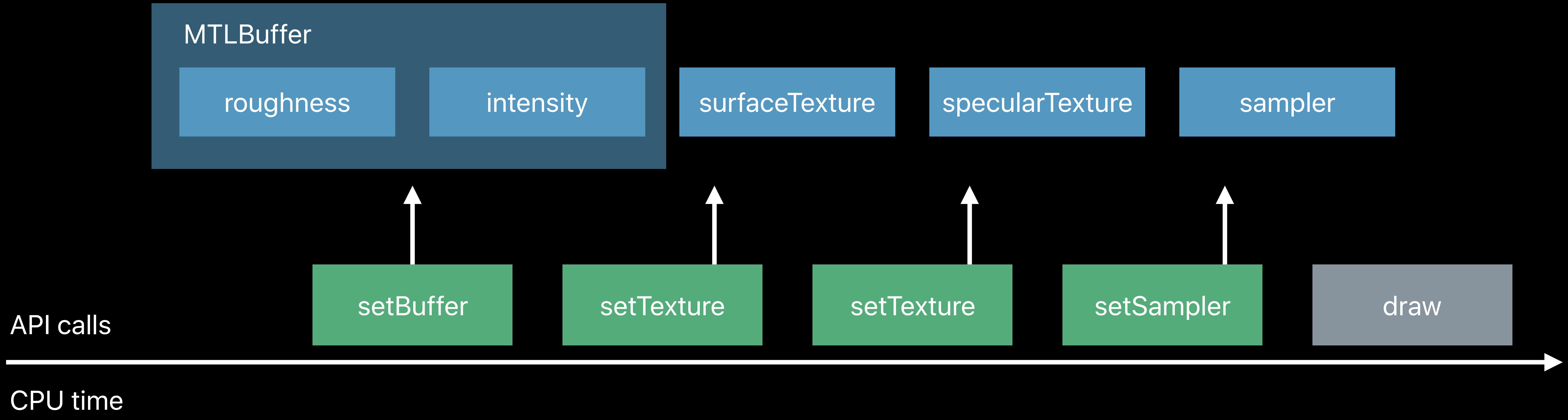
Traditional Argument Model



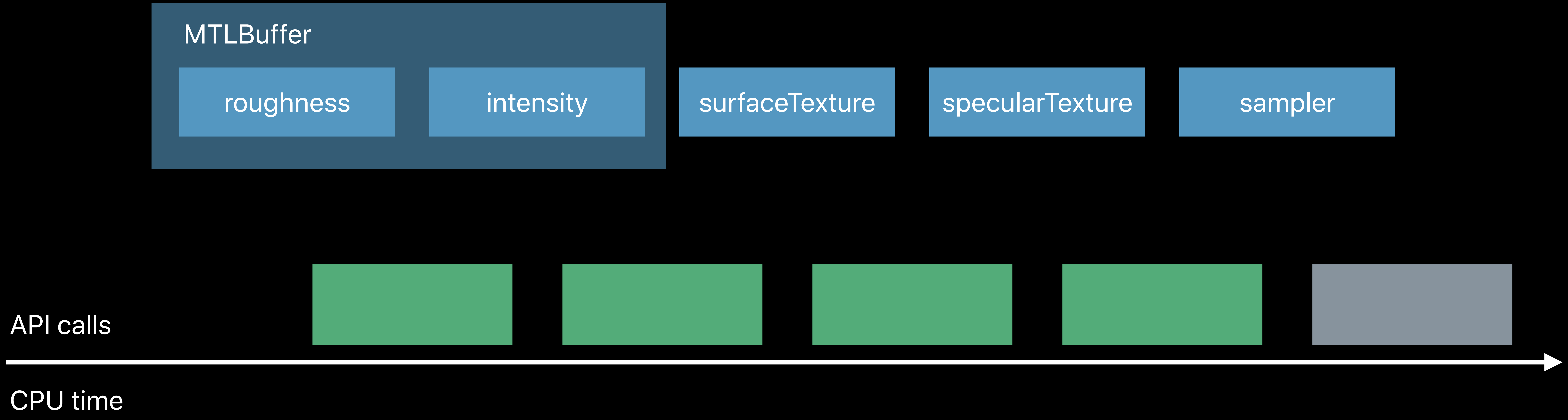
Traditional Argument Model



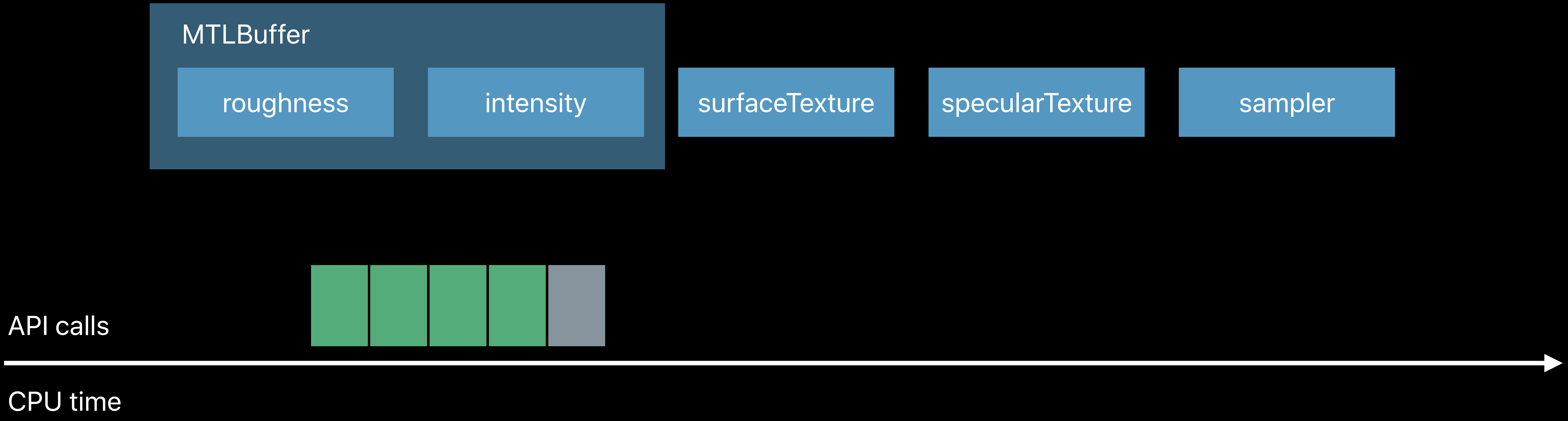
Traditional Argument Model



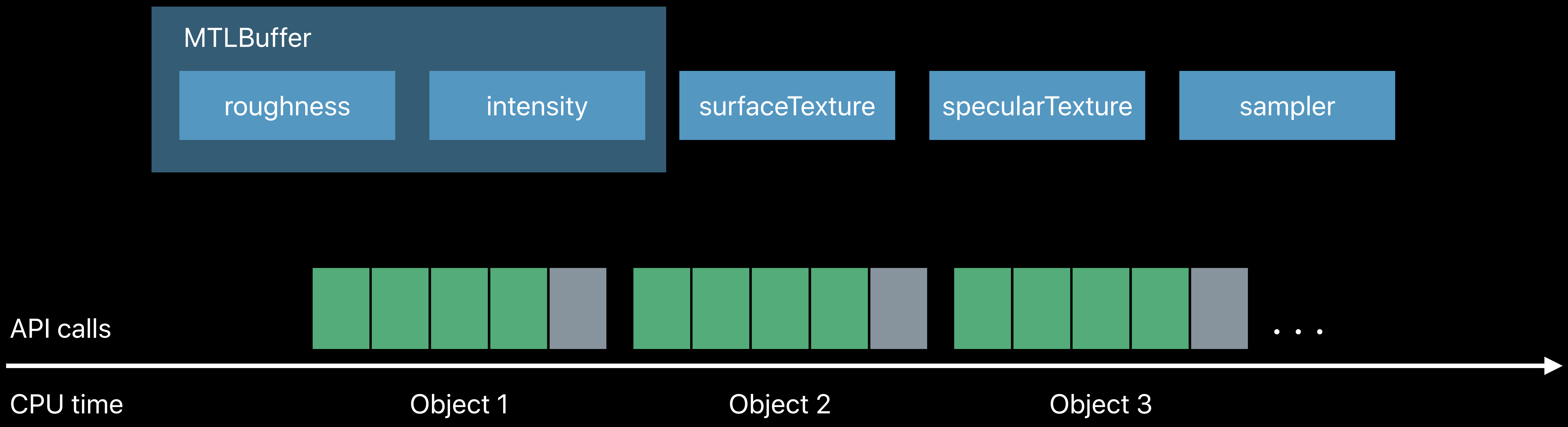
Traditional Argument Model



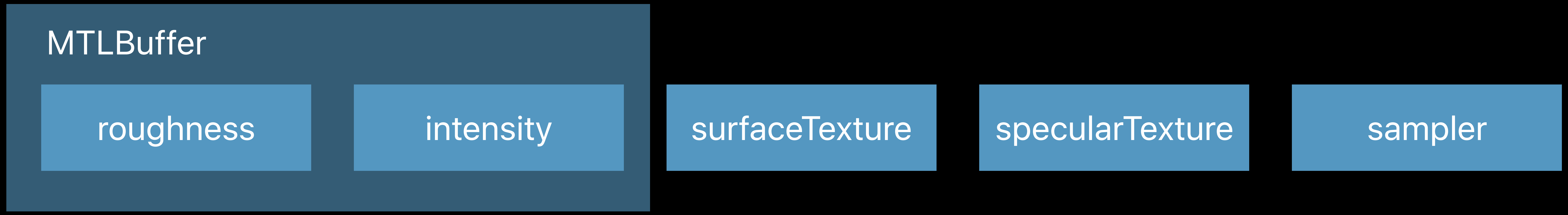
Traditional Argument Model



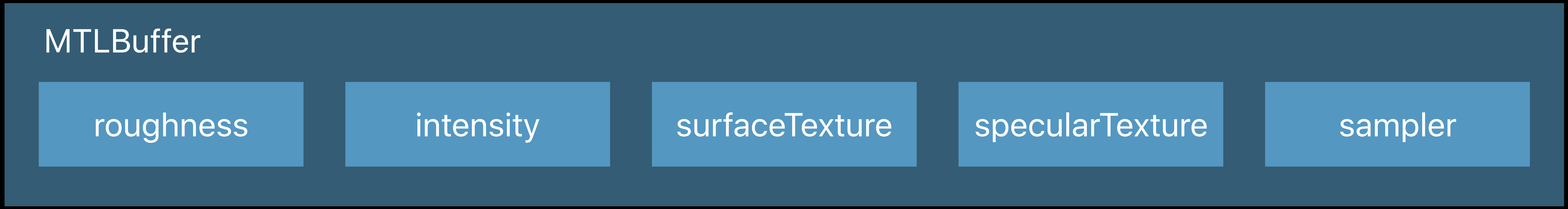
Traditional Argument Model



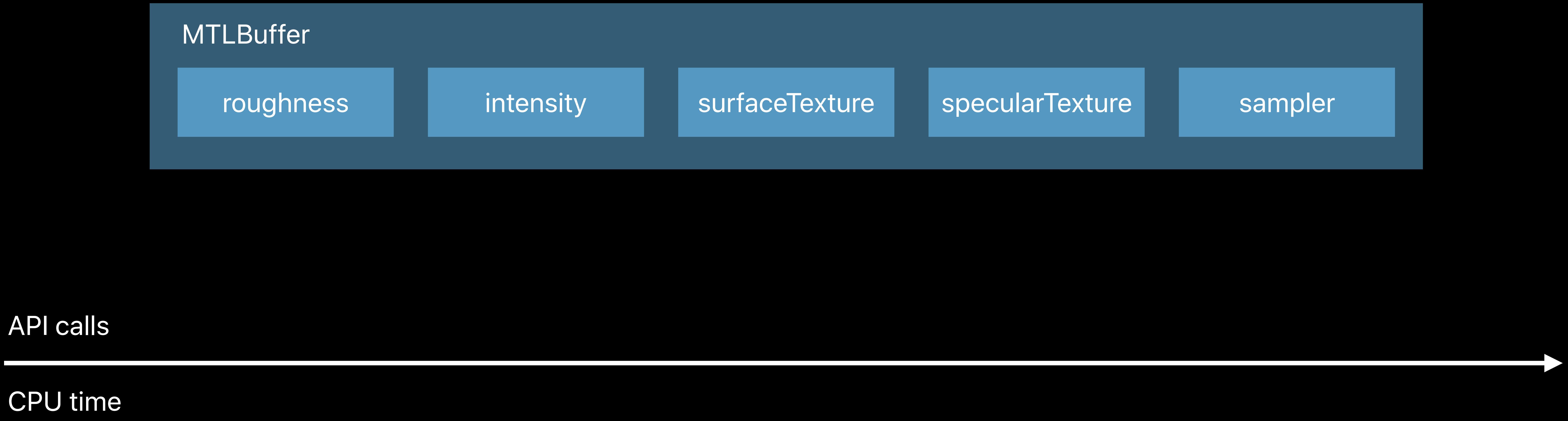
Argument Buffers



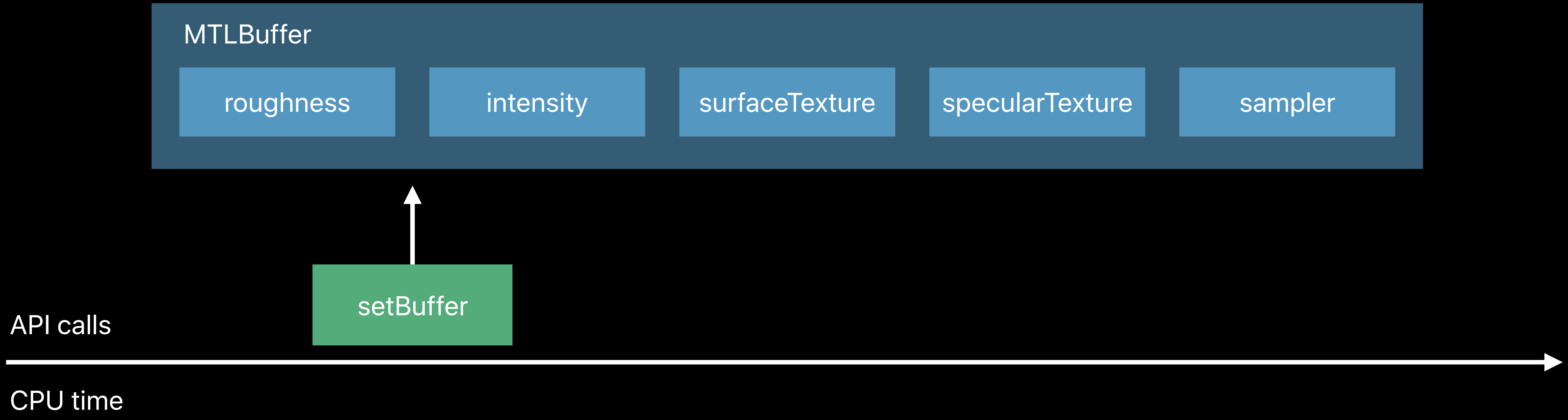
Argument Buffers



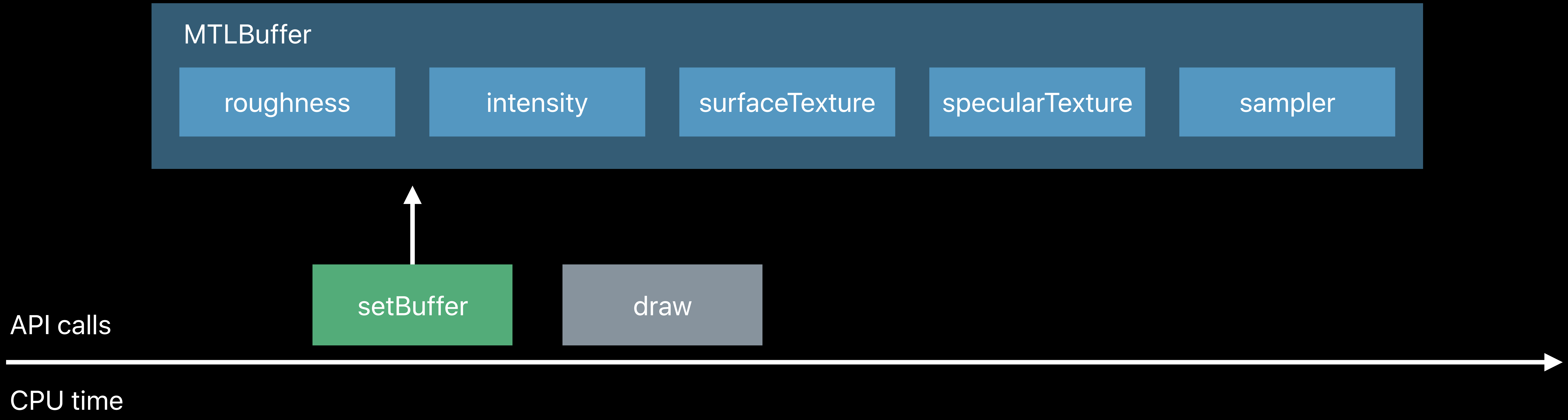
Argument Buffers



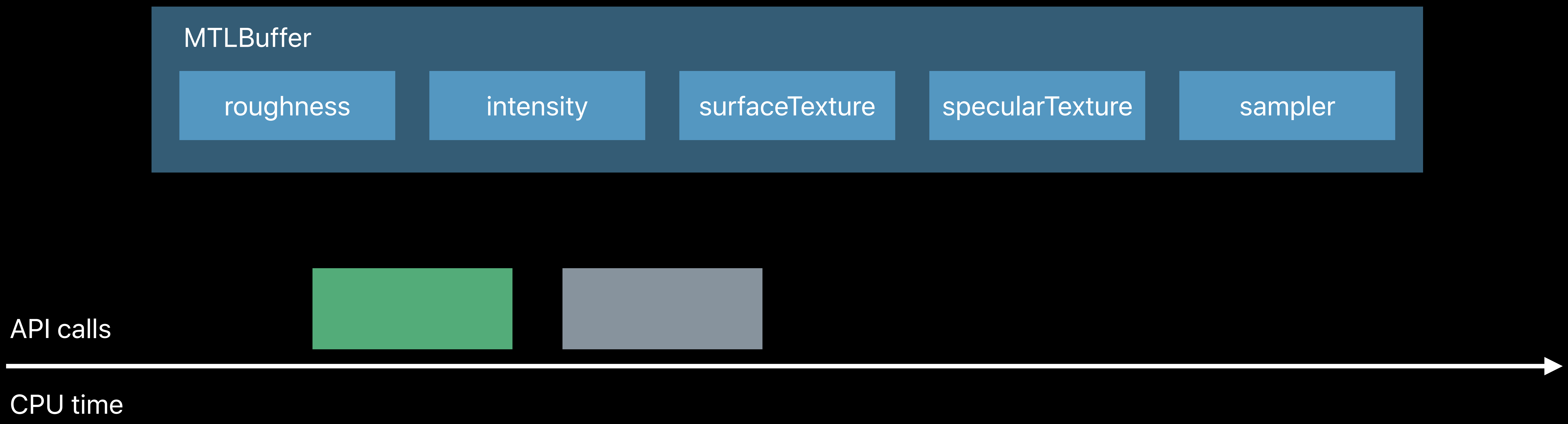
Argument Buffers



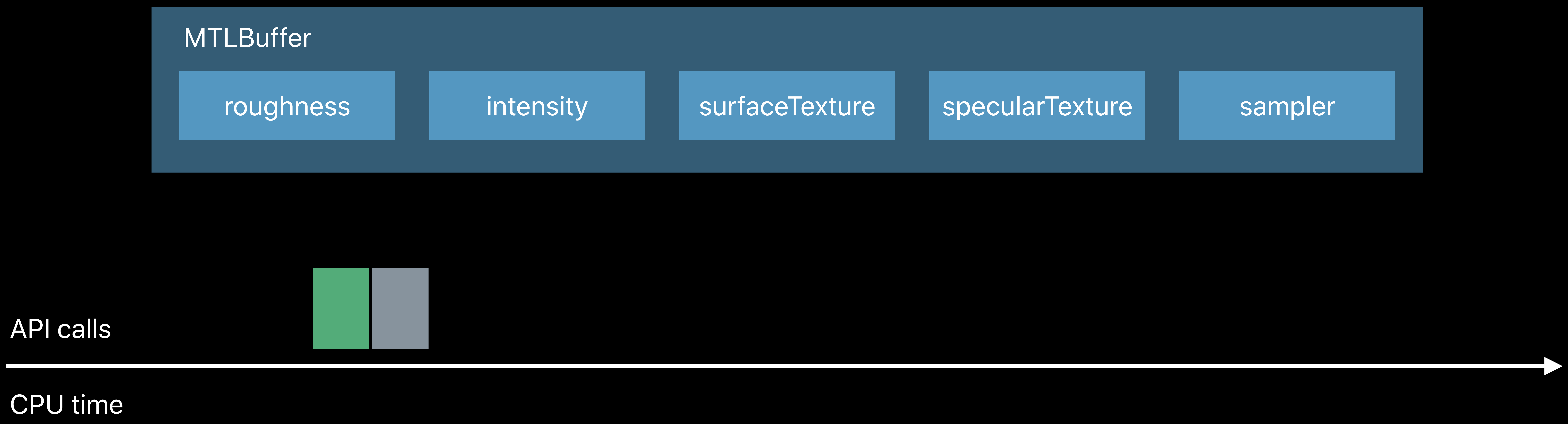
Argument Buffers



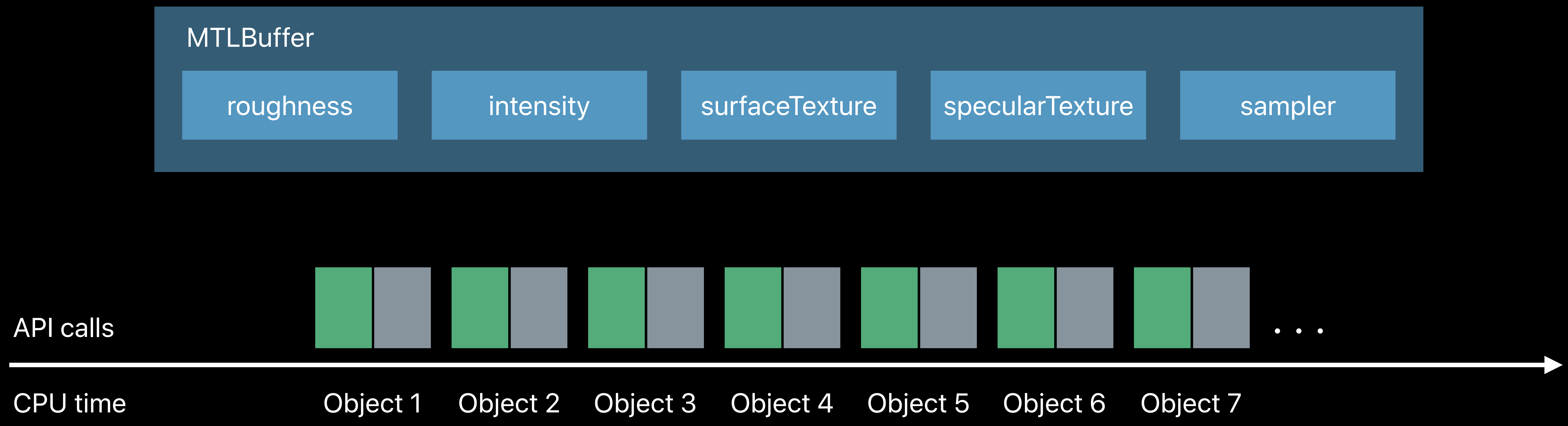
Argument Buffers



Argument Buffers

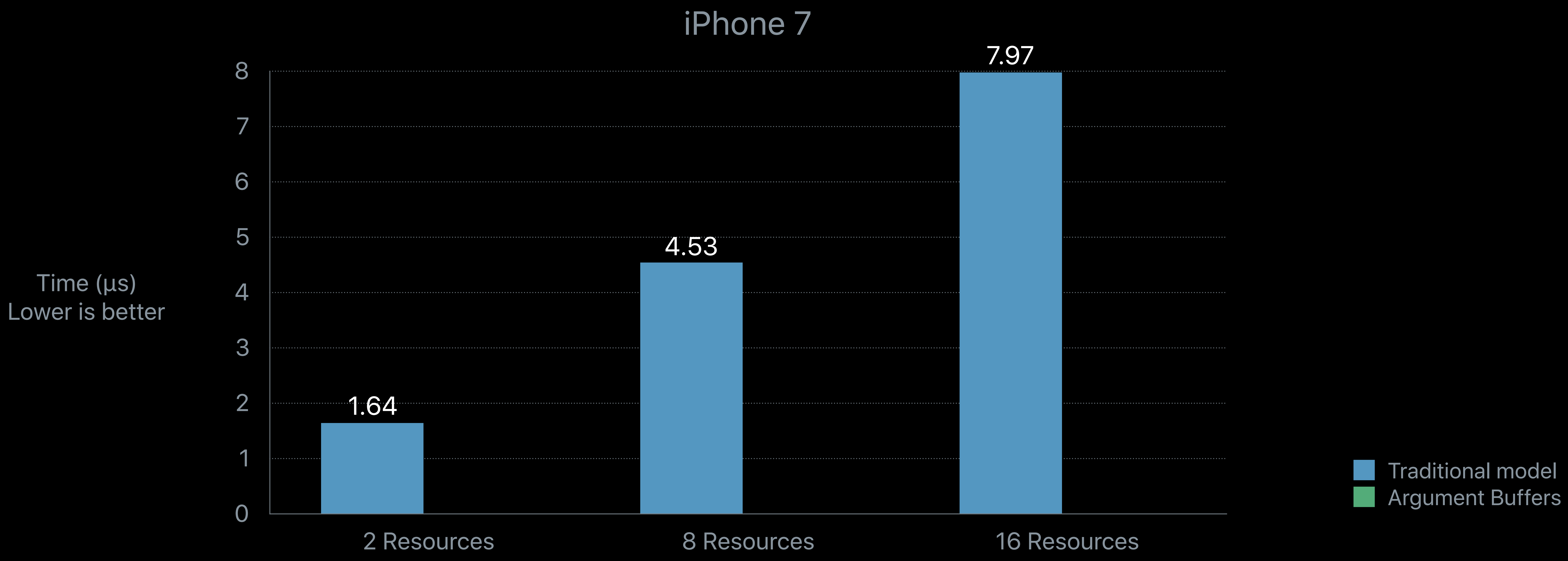


Argument Buffers

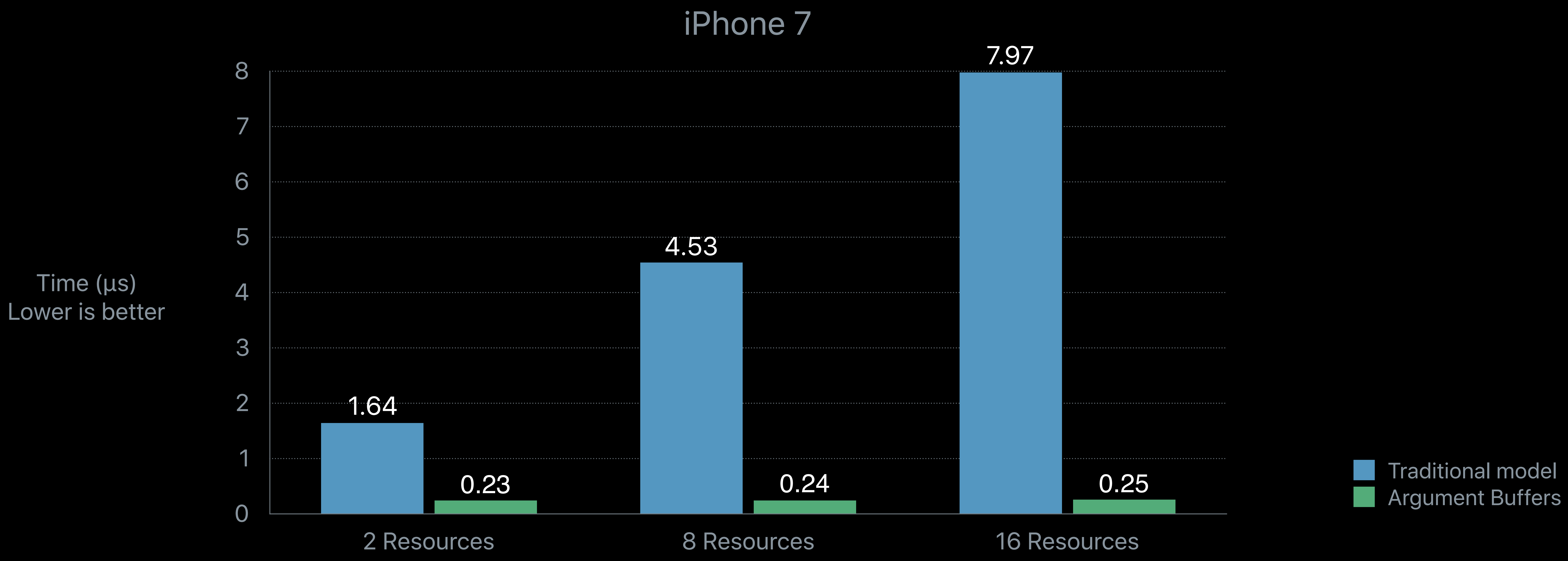


Reduced CPU Overhead

Reduced CPU Overhead



Reduced CPU Overhead



Argument Buffers
Benefits

Improve performance

Enable new use cases

Easy to use

Argument Buffers

Shader example

```
struct Material
{
    float              roughness;
    float              intensity;
    texture2d<float> surfaceTexture;
    texture2d<float> specularTexture;
    sampler            textureSampler;
};

kernel void my_kernel(constant Material &material [[buffer(0)]])
{
    ...
}
```

Argument Buffers

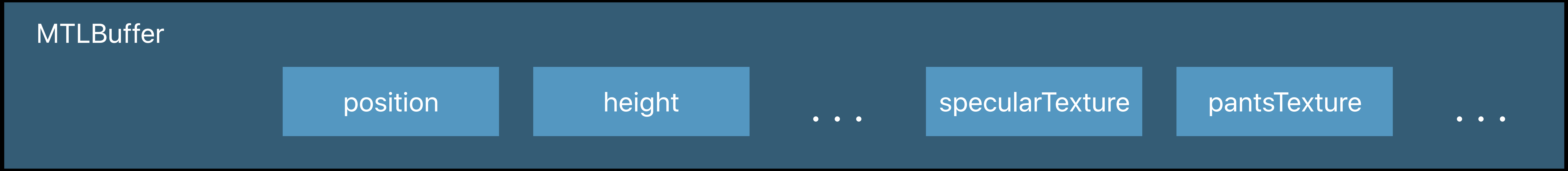
Shader example

```
struct Material
{
    float               roughness;
    float               intensity;
    texture2d<float> surfaceTexture;
    texture2d<float> specularTexture;
    sampler             textureSampler;
};

kernel void my_kernel(constant Material &material [[buffer(0)]])
{
    ...
}
```

Dynamic Indexing

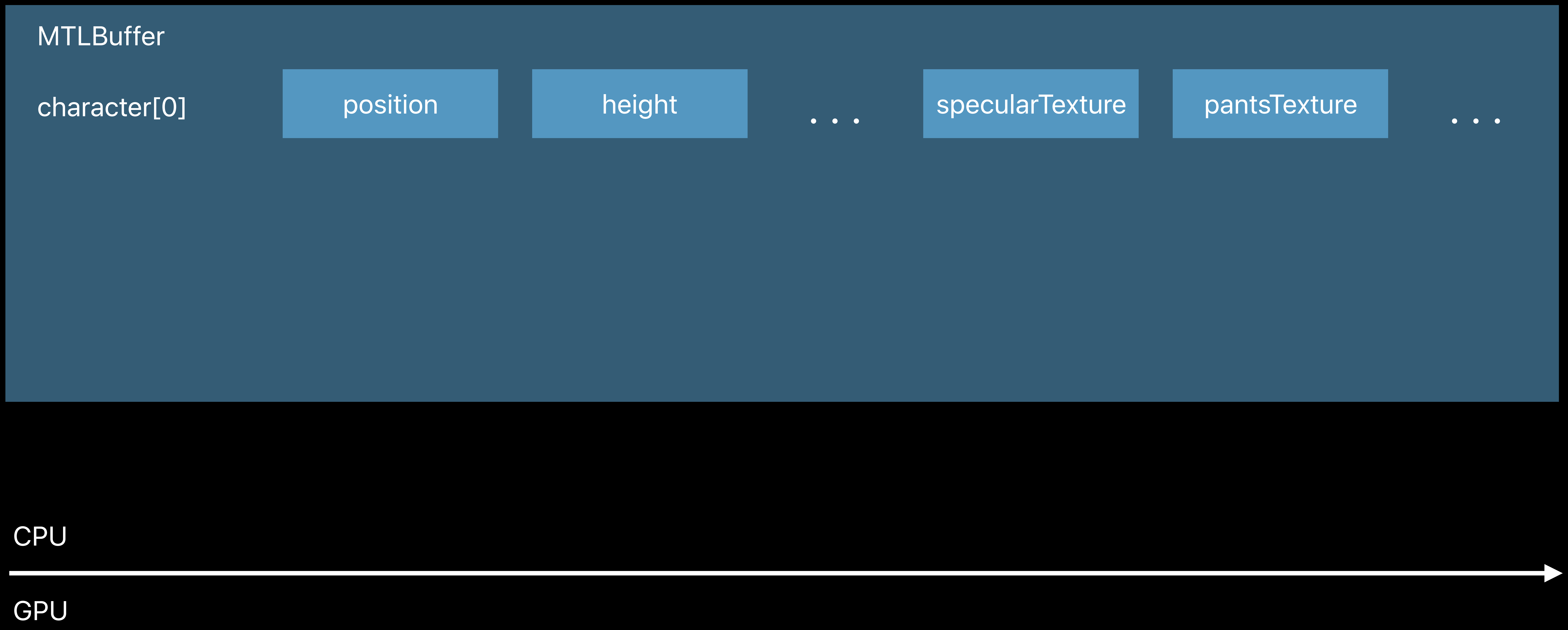
Crowd rendering

Dynamic Indexing
Crowd rendering



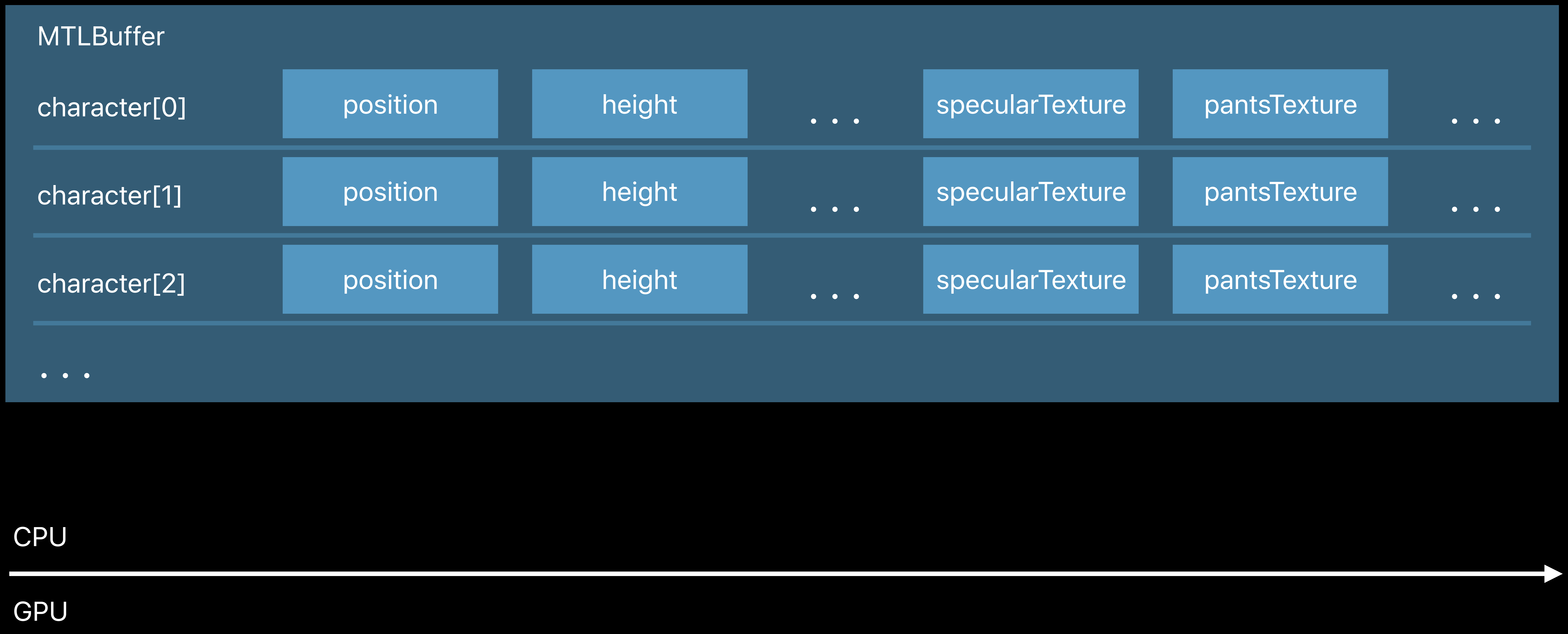
Dynamic Indexing

Crowd rendering



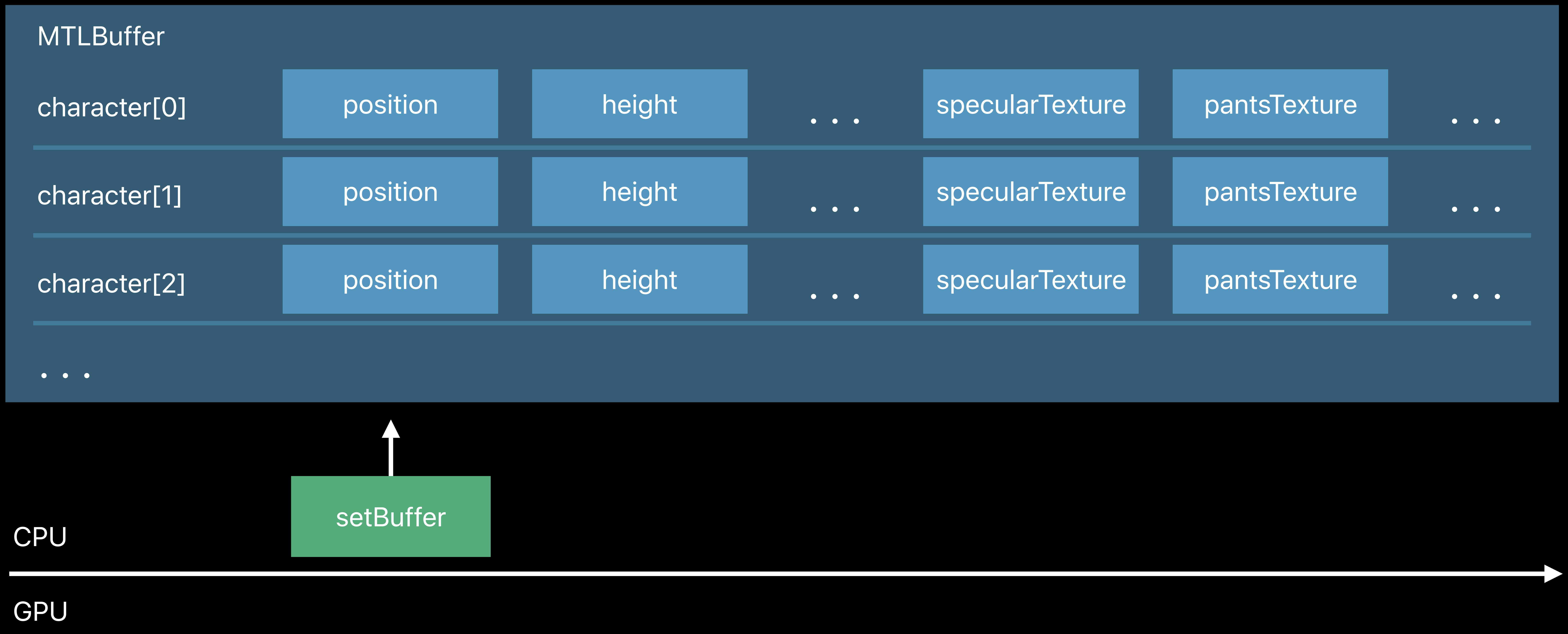
Dynamic Indexing

Crowd rendering



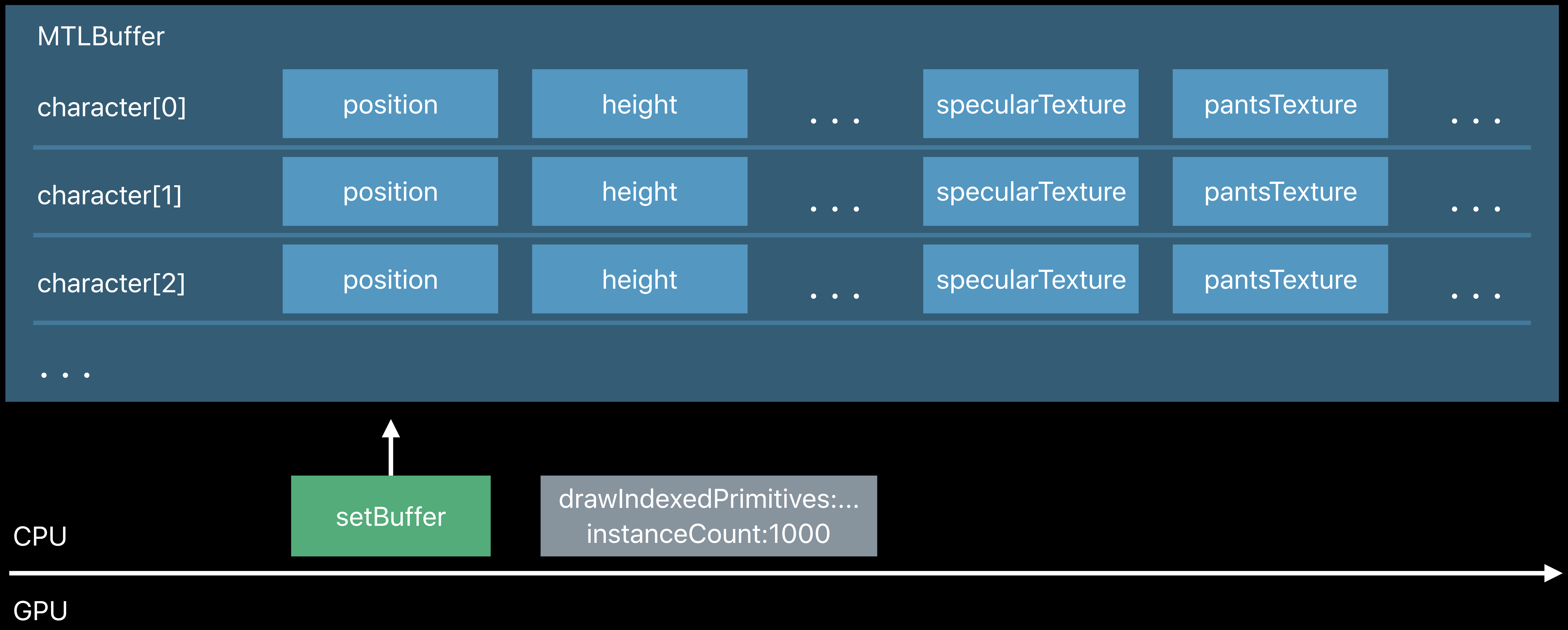
Dynamic Indexing

Crowd rendering



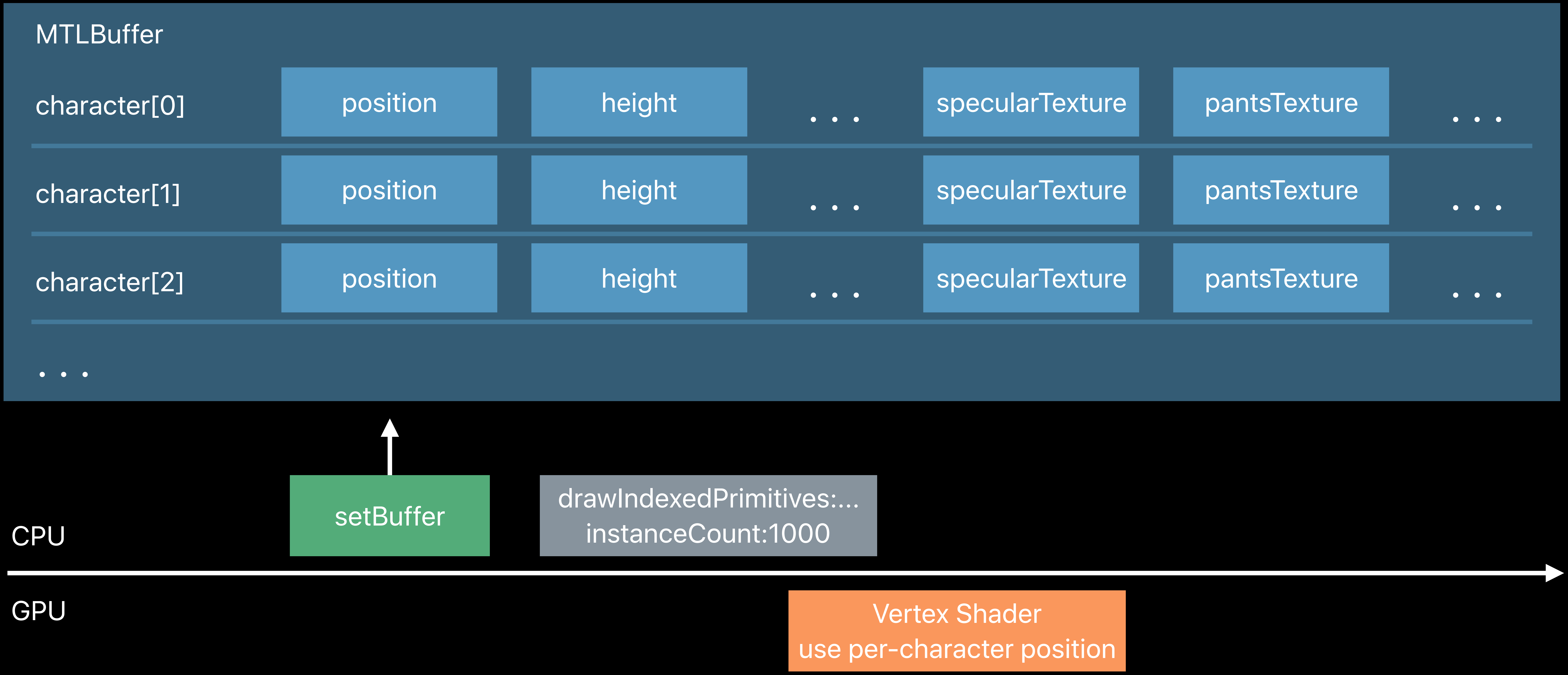
Dynamic Indexing

Crowd rendering

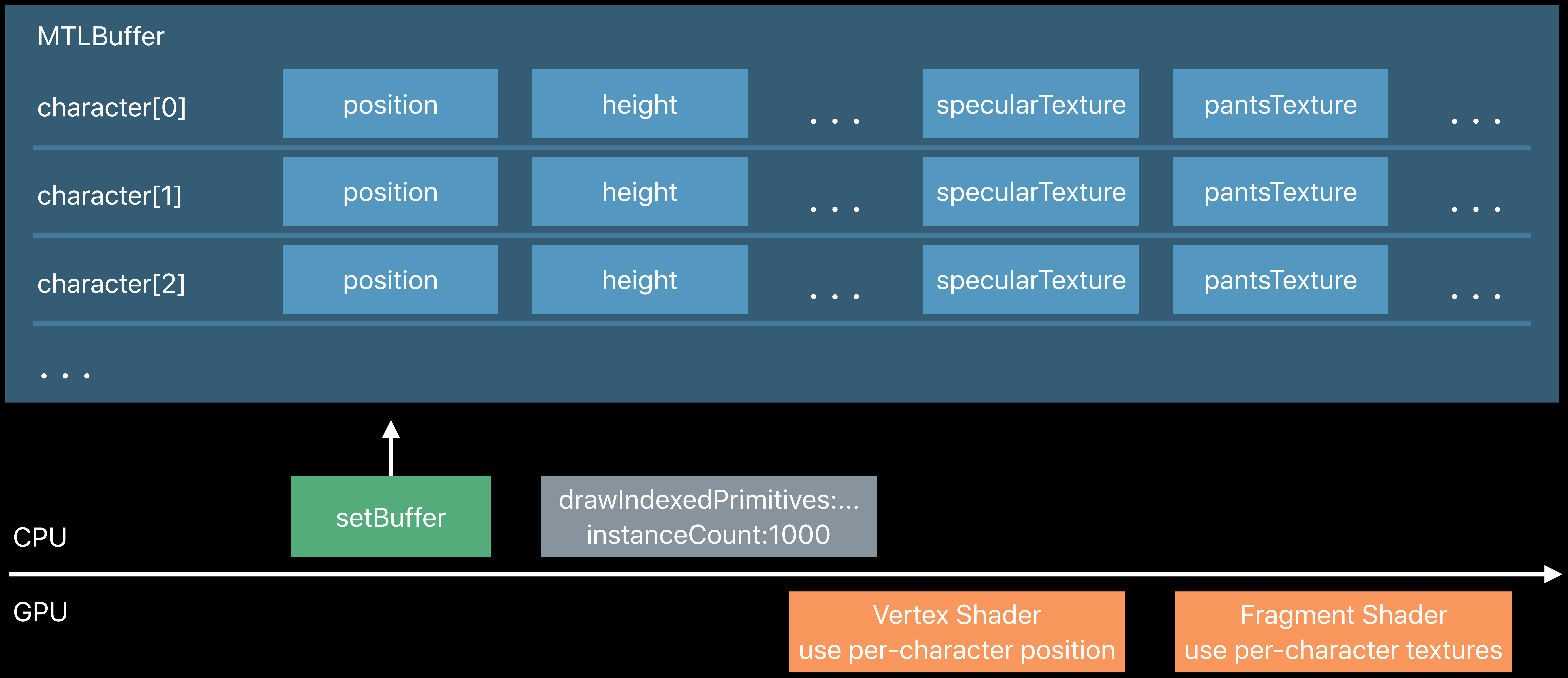


Dynamic Indexing

Crowd rendering



Dynamic Indexing
Crowd rendering



Dynamic Indexing

Vertex shader

```
vertex VertexOutput instancedShader(uint             id    [[instance_id]],
                                    constant Character *crowd [[buffer(0)]])
{
    // Dynamically pick the character for given instance index
    constant Character &obj = crowd[id];

    return transformCharacter(obj);
}
```

Dynamic Indexing

Vertex shader

```
vertex VertexOutput instancedShader(uint               id    [[instance_id]],
                                    constant Character *crowd [[buffer(0)]])
{
    // Dynamically pick the character for given instance index
    constant Character &obj = crowd[id];

    return transformCharacter(obj);
}
```

Dynamic Indexing

Vertex shader

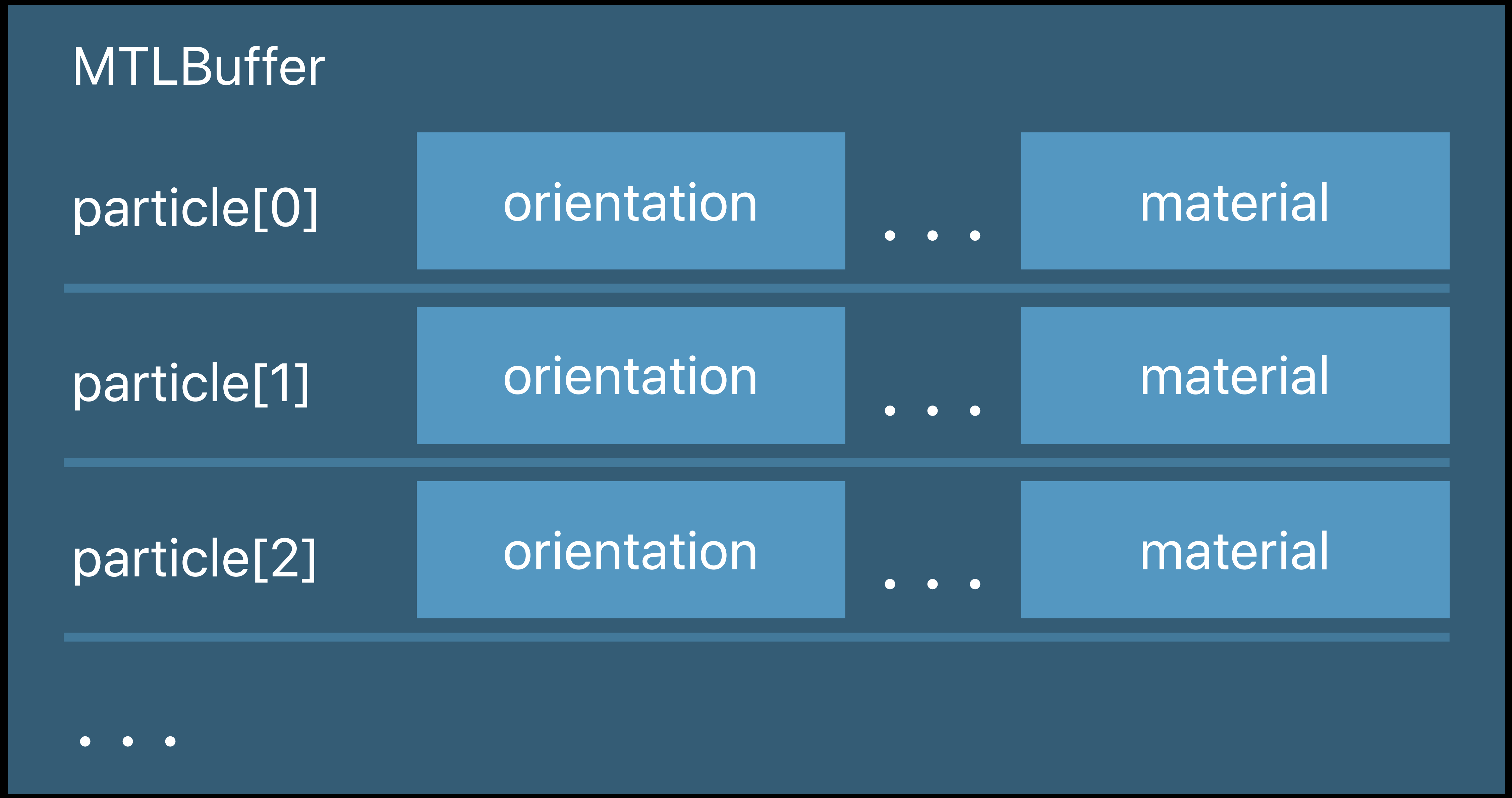
```
vertex VertexOutput instancedShader(uint                  id     [[instance_id]],
                                    constant Character *crowd [[buffer(0)]])
{
    // Dynamically pick the character for given instance index
    constant Character &obj = crowd[id];

    return transformCharacter(obj);
}
```

Set Resources on GPU

Particle simulation

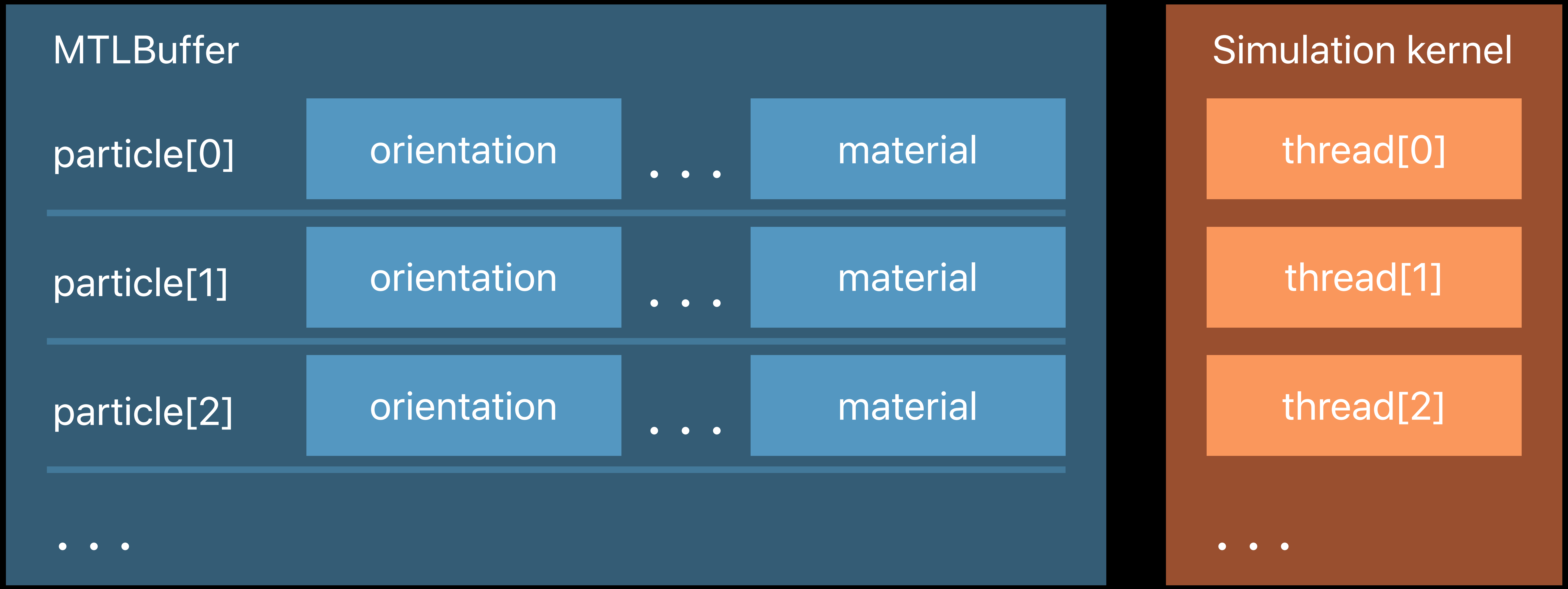
Set Resources on GPU

Particle simulation



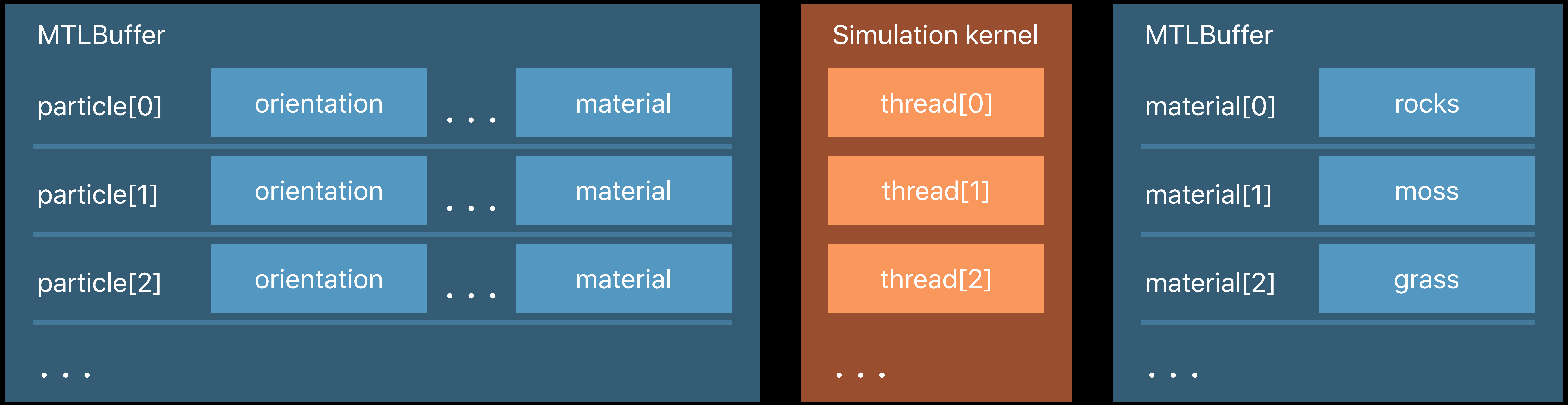
Set Resources on GPU

Particle simulation



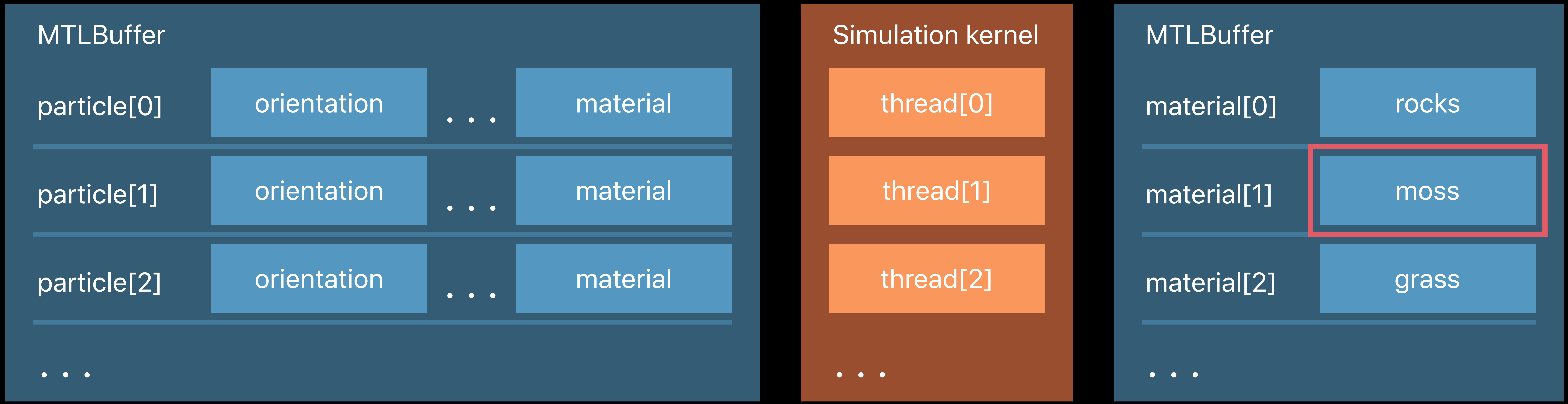
Set Resources on GPU

Particle simulation



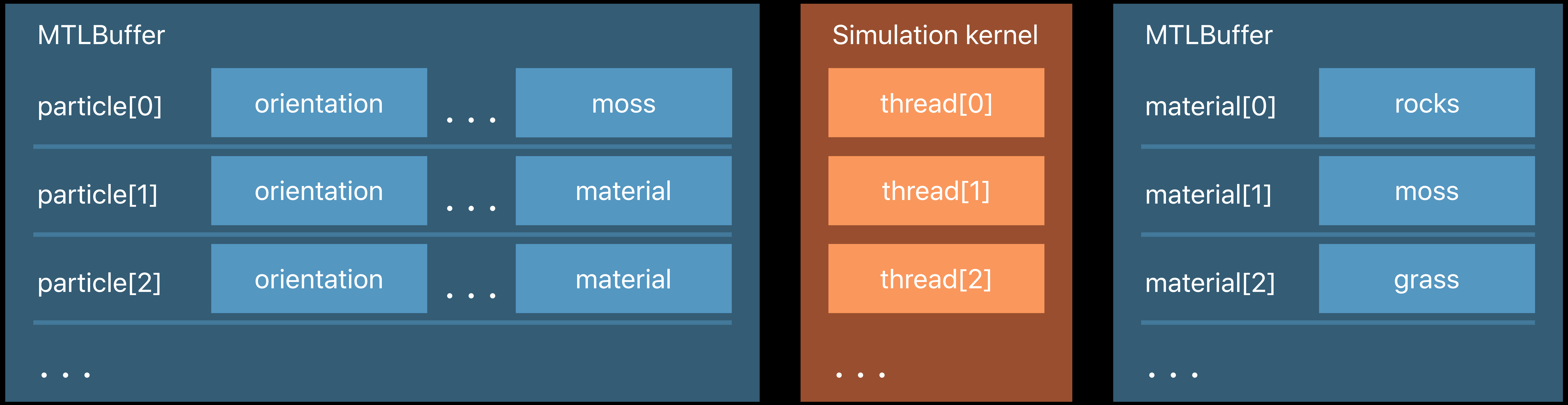
Set Resources on GPU

Particle simulation



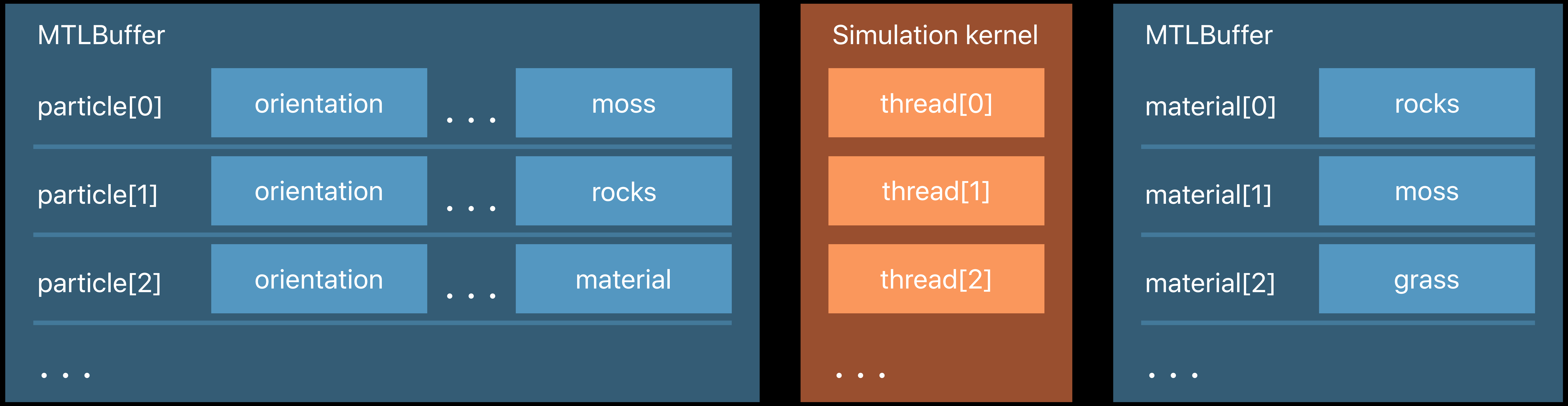
Set Resources on GPU

Particle simulation



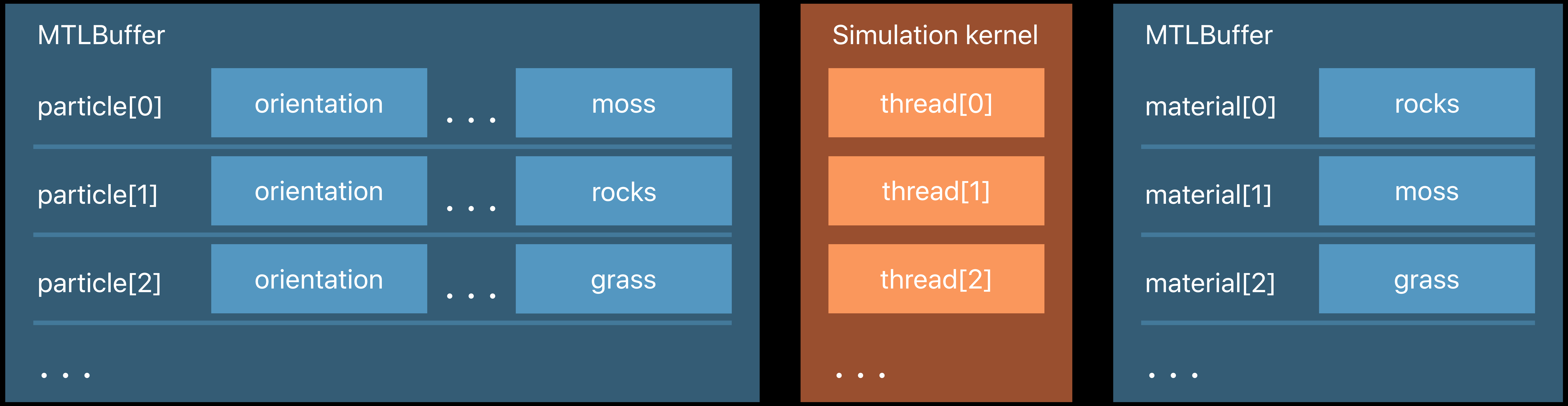
Set Resources on GPU

Particle simulation



Set Resources on GPU

Particle simulation



Set Resources on GPU

Shader example

```
struct Data {
    texture2d<float> tex;
    float            value;
};

kernel void copy(constant Data &src [[buffer(0)]],
                 device   Data &dst [[buffer(1)]])
{
    dst.value = 1.0f;     //< Assign constants
    dst.tex   = src.tex;  //< Copy just a texture
    dst       = src;      //< Copy entire structure
}
```

Set Resources on GPU

Shader example

```
struct Data {
    texture2d<float> tex;
    float            value;
};

kernel void copy(constant Data &src [[buffer(0)]],
                 device   Data &dst [[buffer(1)]])
{
    dst.value = 1.0f;     //< Assign constants
    dst.tex   = src.tex;  //< Copy just a texture
    dst       = src;      //< Copy entire structure
}
```

Set Resources on GPU

Shader example

```
struct Data {
    texture2d<float> tex;
    float            value;
};

kernel void copy(constant Data &src [[buffer(0)]],
                 device   Data &dst [[buffer(1)]])
{
    dst.value = 1.0f;     //< Assign constants
    dst.tex   = src.tex;  //< Copy just a texture
    dst       = src;      //< Copy entire structure
}
```

Set Resources on GPU

Shader example

```
struct Data {
    texture2d<float> tex;
    float            value;
};

kernel void copy(constant Data &src [[buffer(0)]],
                 device   Data &dst [[buffer(1)]])
{
    dst.value = 1.0f;     //< Assign constants
    dst.tex   = src.tex;  //< Copy just a texture
    dst       = src;      //< Copy entire structure
}
```

Set Resources on GPU

Shader example

```
struct Data {
    texture2d<float> tex;
    float            value;
};

kernel void copy(constant Data &src [[buffer(0)]],
                 device   Data &dst [[buffer(1)]])
{
    dst.value = 1.0f;     //< Assign constants
    dst.tex   = src.tex;  //< Copy just a texture
    dst       = src;      //< Copy entire structure
}
```

Multiple Indirections

Argument Buffers can reference other Argument Buffers

Create and reuse complex object hierarchies

```
struct Object {
    float4           position;
    device Material *material;     ///< Many objects can point to the same material
};

struct Tree {
    device Tree   *children[2];
    device Object *objects;    ///< Array of objects in the node
};
```

Multiple Indirections

Argument Buffers can reference other Argument Buffers

Create and reuse complex object hierarchies

```
struct Object {
    float4           position;
    device Material *material;     ///< Many objects can point to the same material
};

struct Tree {
    device Tree   *children[2];
    device Object *objects;    ///< Array of objects in the node
};
```

Multiple Indirections

Argument Buffers can reference other Argument Buffers

Create and reuse complex object hierarchies

```
struct Object {
    float4           position;
    device Material *material;     ///< Many objects can point to the same material
};

struct Tree {
    device Tree   *children[2];
    device Object *objects;    ///< Array of objects in the node
};
```

Support Tiers

CPU overhead reduction
Set resources on GPU
Multiple indirections
Arrays of Argument Buffers
Resources per draw call

Support Tiers

	Tier 1
CPU overhead reduction	Yes
Set resources on GPU	No
Multiple indirections	No
Arrays of Argument Buffers	No
Resources per draw call	Unchanged

Support Tiers

	Tier 1	Tier 2
CPU overhead reduction	Yes	Yes
Set resources on GPU	No	Yes
Multiple indirections	No	Yes
Arrays of Argument Buffers	No	Yes
Resources per draw call	Unchanged	500,000 textures and buffers

Argument Buffers

Terrain rendering

Material changes with terrain

Trees placed by GPU

Particle materials generated by GPU

Available as sample code

API Highlights

Argument Buffers are stored in plain `MTLBuffer`

Use `MTLArgumentEncoder` to fill Indirect Argument Buffers

Abstracts platform differences behind simple interface

Up to eight Argument Buffers per stage

```
// Shader syntax
struct Particle {
    texture2d<float> surface;
    float4           position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }
```

```
// Shader syntax
struct Particle {
    texture2d<float> surface;
    float4           position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }
```

```
// Shader syntax
struct Particle {
    texture2d<float> surface;
    float4           position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }


// Create encoder for first indirect argument buffer in Metal function 'simulate'
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)
```

```
// Shader syntax
struct Particle {
    texture2d<float> surface;
    float4           position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }


// Create encoder for first indirect argument buffer in Metal function 'simulate'
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)
```

```
// Shader syntax
struct Particle {
    texture2d<float> surface;
    float4           position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }
```

```
// Create encoder for first indirect argument buffer in Metal function 'simulate'
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)
```

```
// Shader syntax
struct Particle {
    texture2d<float> surface;
    float4           position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }


// Create encoder for first indirect argument buffer in Metal function 'simulate'
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)


// API calls to fill the indirect argument buffer
particleEncoder.setTexture(mySurfaceTexture, at: 0)
particleEncoder.constantData(at: 1).storeBytes(of: myPosition, as: float4.self)
```

```
// Shader syntax
struct Particle {
    texture2d<float> surface;
    float4           position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }
```

```
// Create encoder for first indirect argument buffer in Metal function 'simulate'
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)
```

```
// API calls to fill the indirect argument buffer
particleEncoder.setTexture(mySurfaceTexture, at: 0)
particleEncoder.constantData(at: 1).storeBytes(of: myPosition, as: float4.self)
```

```
// Shader syntax
struct Particle {
    texture2d<float> surface;
    float4           position;
};

kernel void simulate(constant Particle &particle [[buffer(0)]]) { ... }


// Create encoder for first indirect argument buffer in Metal function 'simulate'
let simulateFunction = library.makeFunction(name:"simulate")!
let particleEncoder = simulateFunction.makeArgumentEncoder(bufferIndex: 0)


// API calls to fill the indirect argument buffer
particleEncoder.setTexture(mySurfaceTexture, at: 0)
particleEncoder.constantData(at: 1).storeBytes(of: myPosition, as: float4.self)
```

Managing Resource Usage

Tell Metal what resources you plan to use

Use Metal Heaps for best performance

```
// Use for textures with sample access or buffers
commandEncoder.use(myTextureHeap)

// Used for all render targets, views or read/write access to texture
commandEncoder.use(myRenderTarget, usage: .write)
```

Managing Resource Usage

Tell Metal what resources you plan to use

Use Metal Heaps for best performance

```
// Use for textures with sample access or buffers
commandEncoder.use(myTextureHeap)

// Used for all render targets, views or read/write access to texture
commandEncoder.use(myRenderTarget, usage: .write)
```

Managing Resource Usage

Tell Metal what resources you plan to use

Use Metal Heaps for best performance

```
// Use for textures with sample access or buffers
commandEncoder.use(myTextureHeap)

// Used for all render targets, views or read/write access to texture
commandEncoder.use(myRenderTarget, usage: .write)
```

Best Practices

Organize based on usage pattern

- Per-view vs. per-object vs. per-material
- Dynamically changing vs. Static

Favor data locality

Use traditional model where appropriate

Raster Order Groups

Richard Schreyer

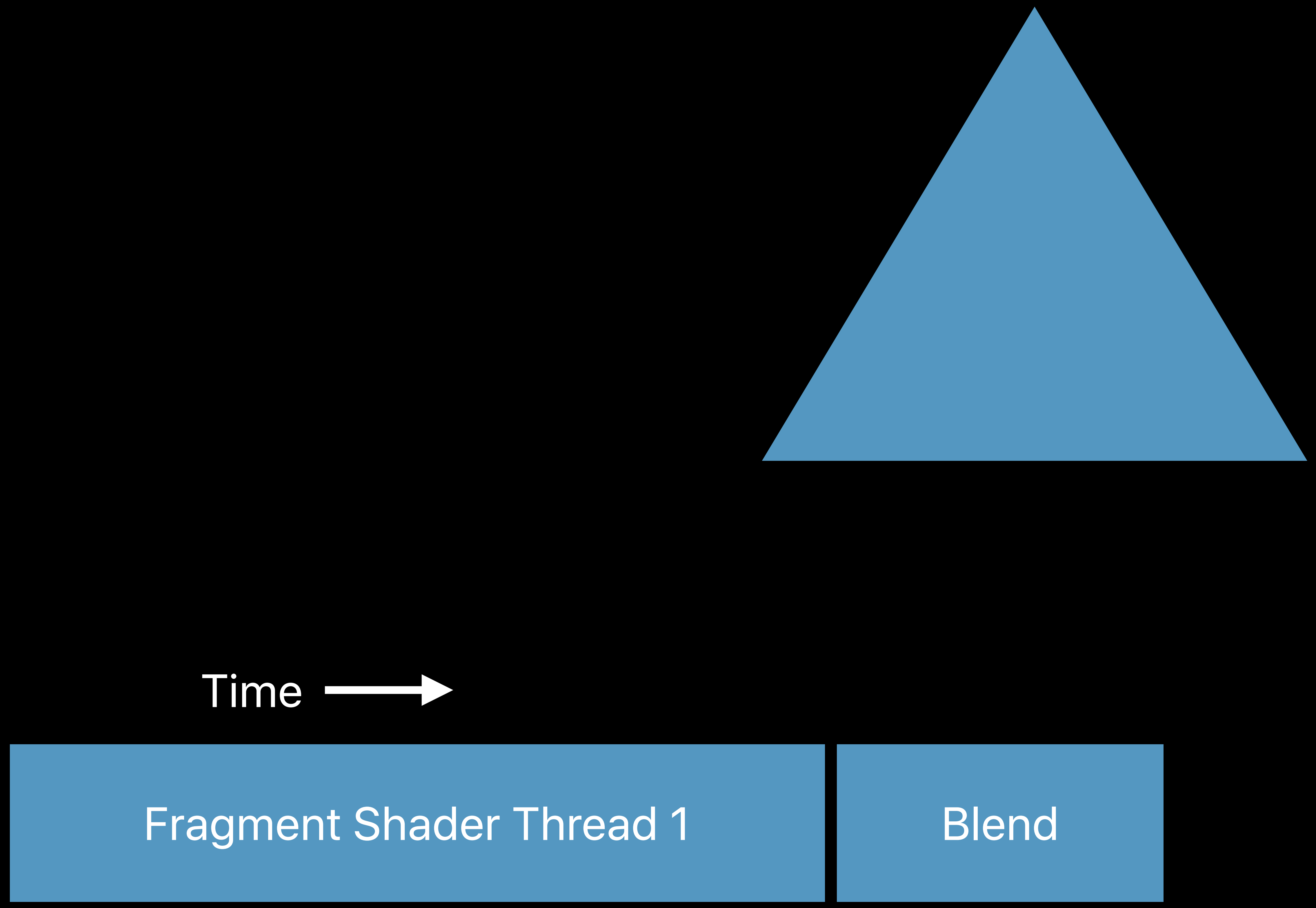
Raster Order Groups

Ordered memory access from fragment shaders

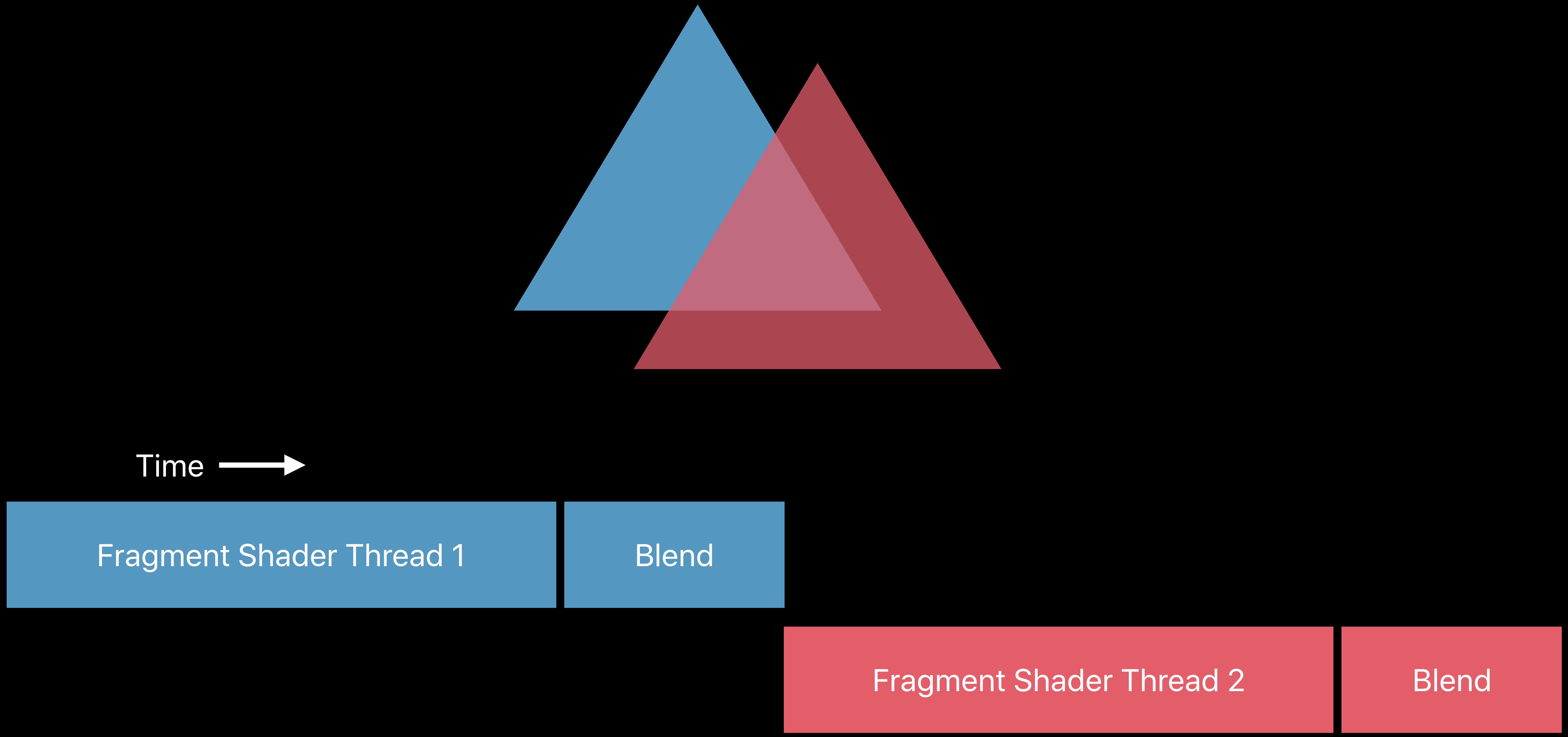
Enables new rendering techniques

- Order-independent transparency
- Dual-layer GBuffers
- Voxelization
- Custom blending

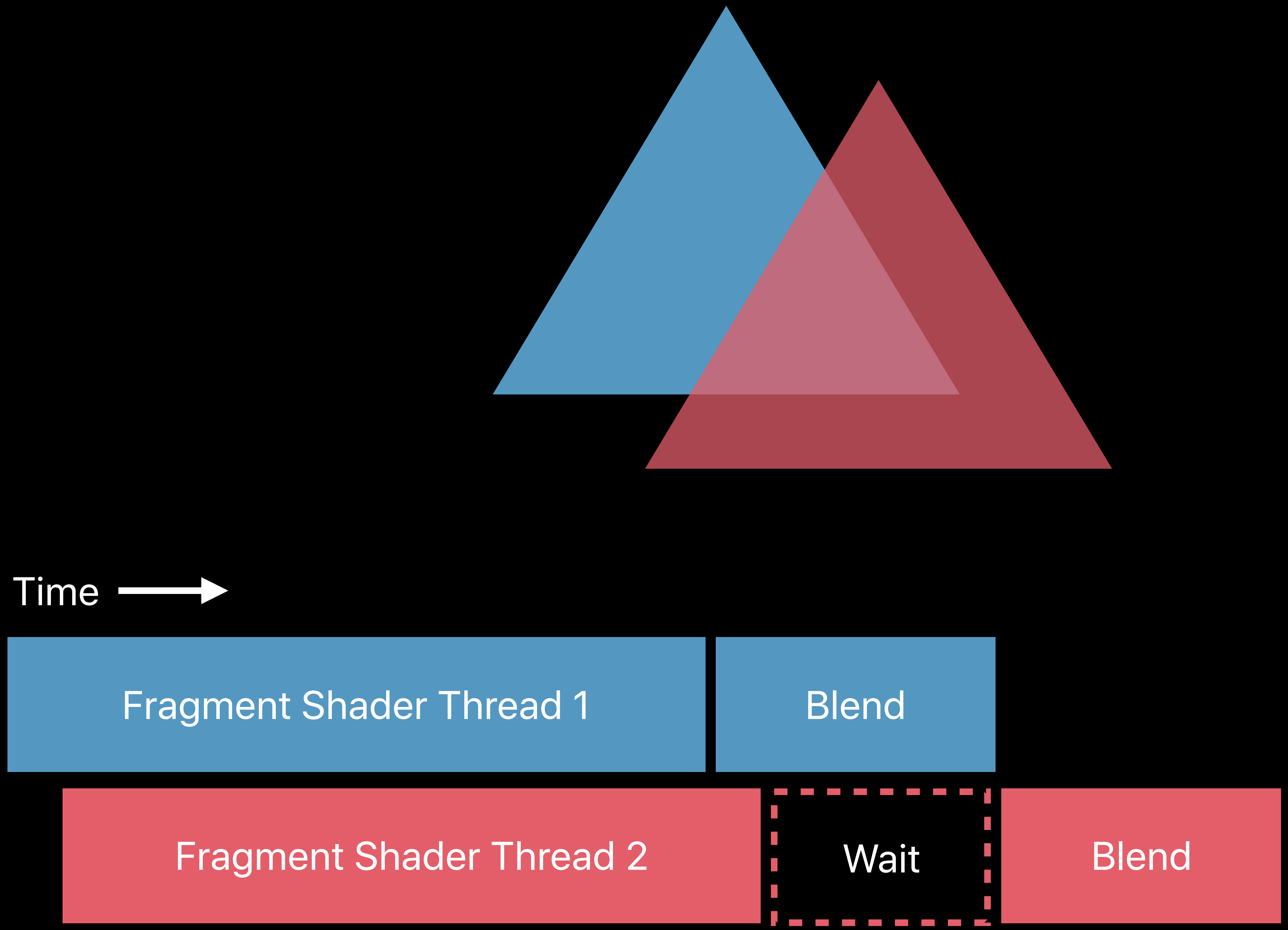
Fragment Shaders with Blending



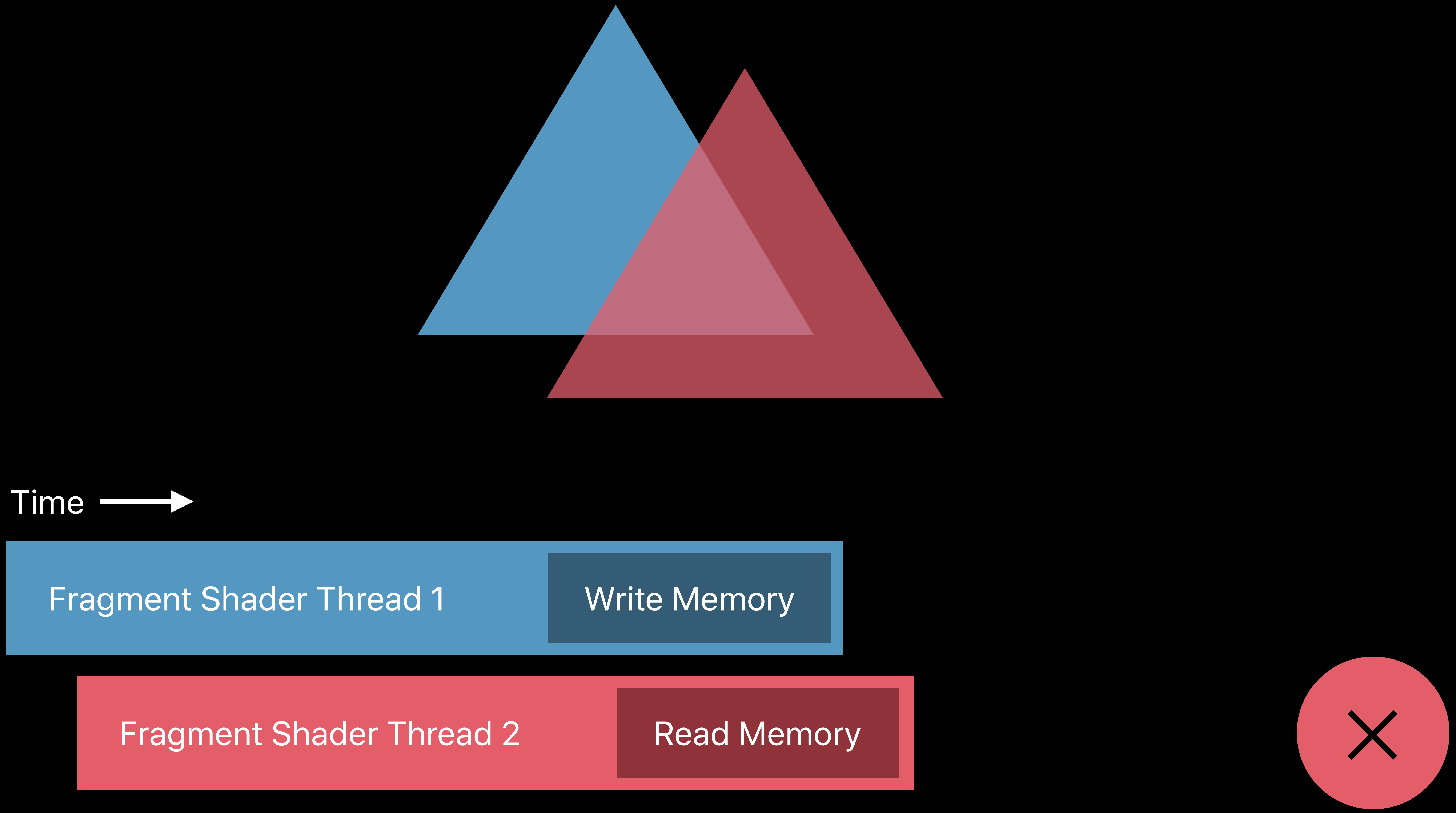
Fragment Shaders with Blending



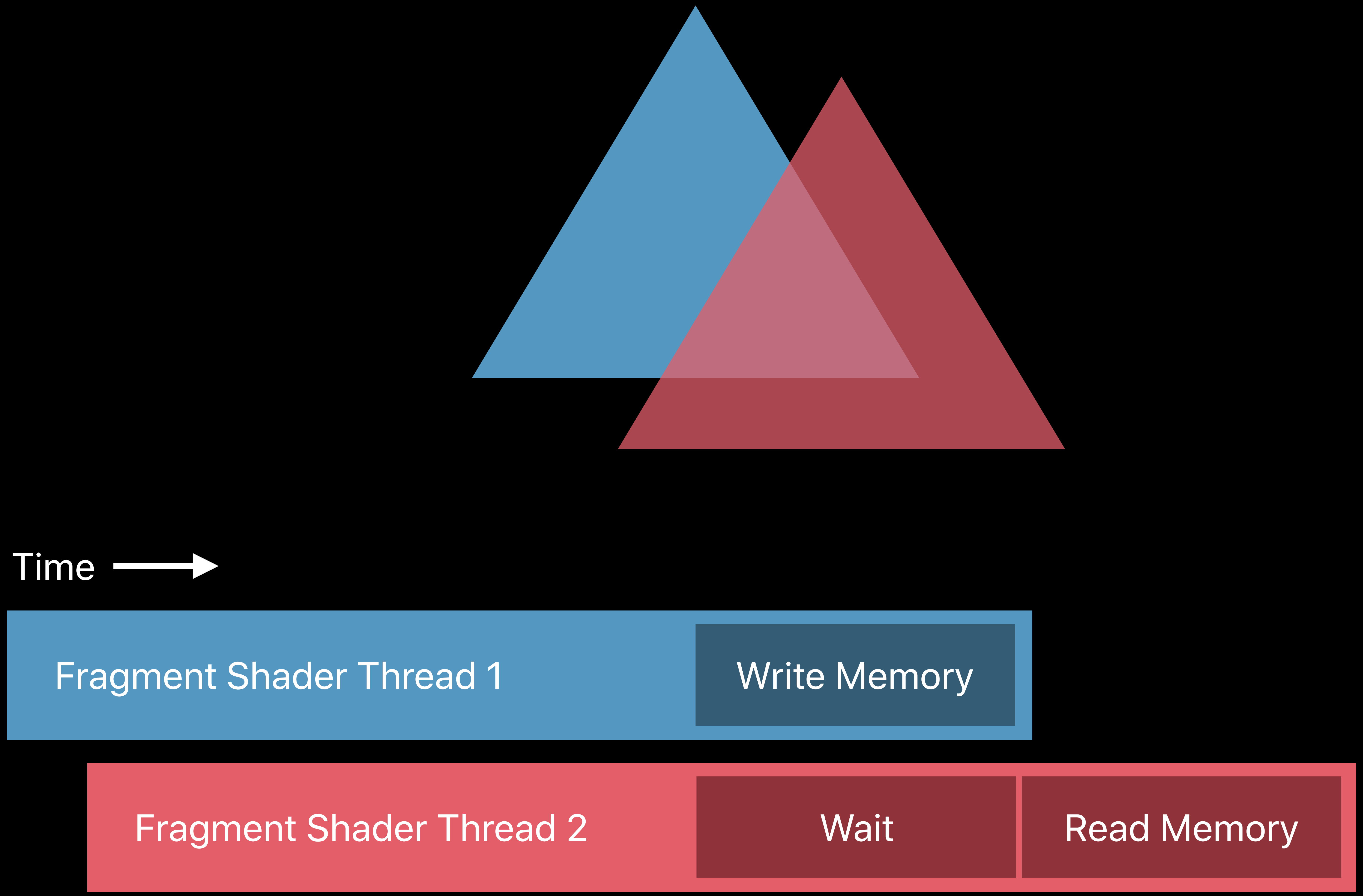
Fragment Shaders with Blending



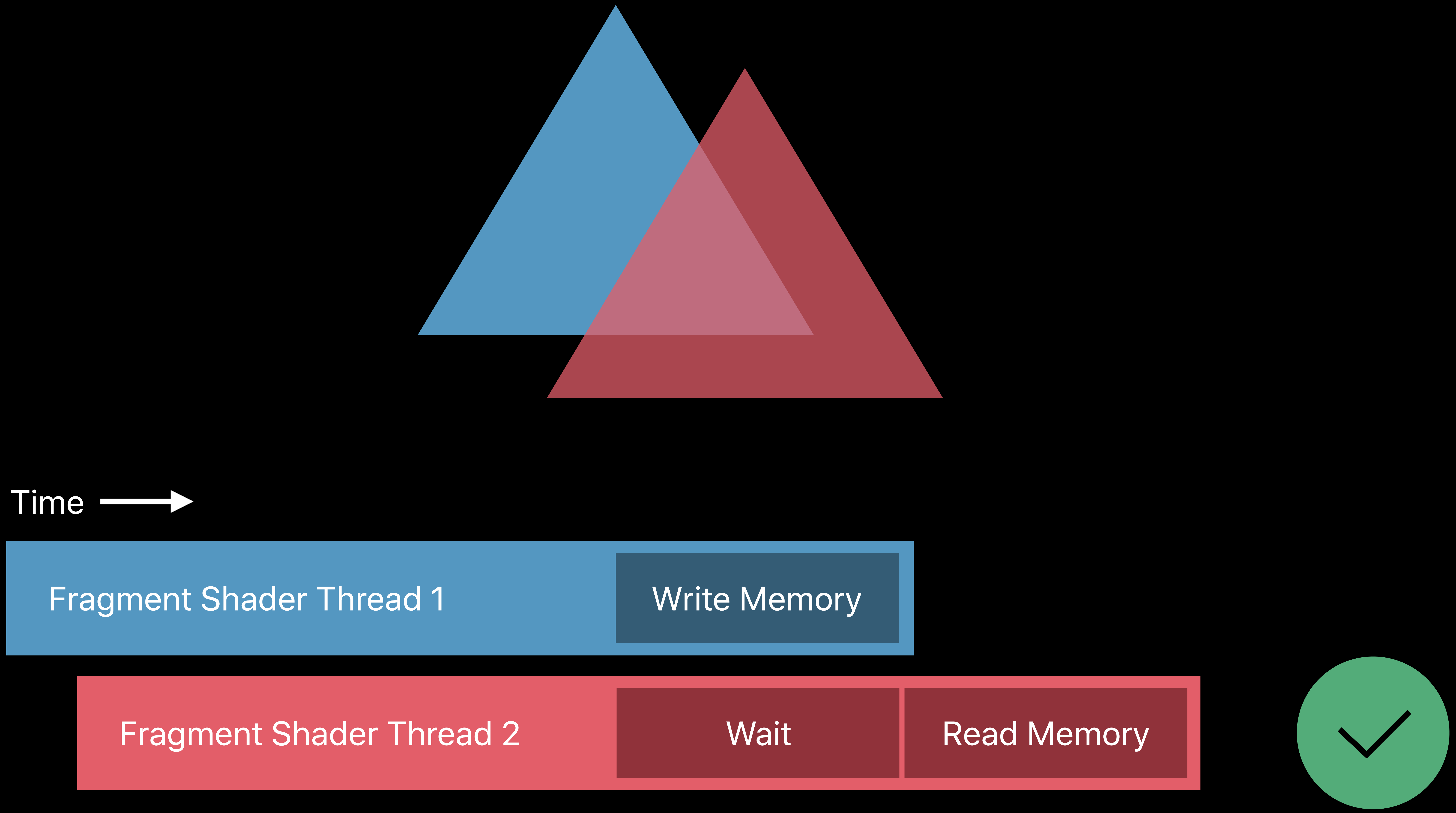
Mid-Shader Memory Access



With Raster Order Groups



With Raster Order Groups



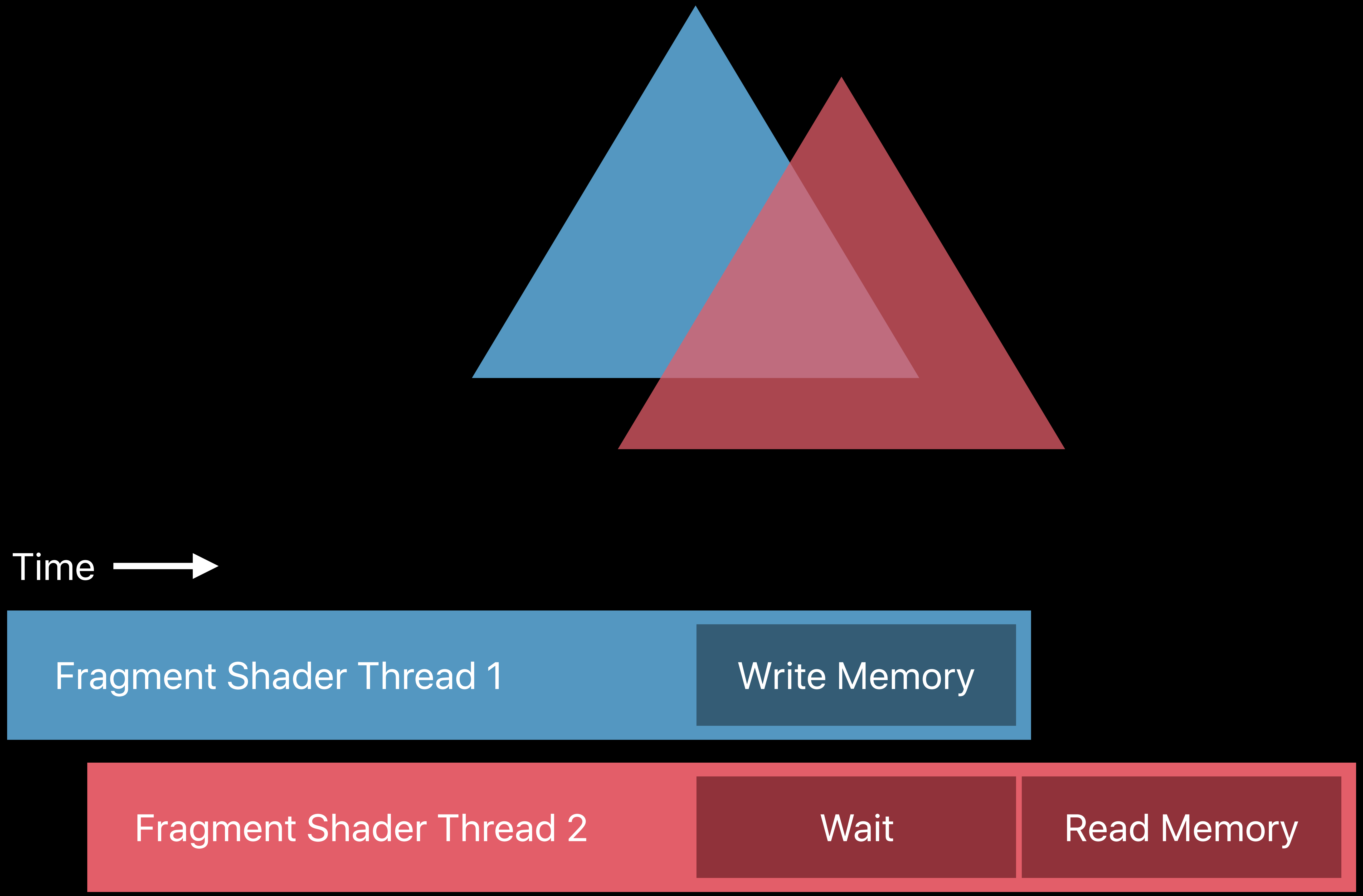
```
//  Blending manually to a pointer in memory
fragment void BlendSomething(
    texture2d<float, access::read_write> framebuffer [[texture(0)]]) {

   float4 newColor = …

    //  Non-atomic access to memory without synchronization
    float4 priorColor = framebuffer.read(framebufferPosition);
    float4 blended = custom_blend(newColor, priorColor);
    framebuffer.write(blended, framebufferPosition);
}
```

```
//  Blending manually to a pointer in memory
fragment void BlendSomething(
   texture2d<float, access::read_write> framebuffer [[texture(0)]]) {

   float4 newColor = …

    //  Non-atomic access to memory without synchronization
    float4 priorColor = framebuffer.read(framebufferPosition);
    float4 blended = custom_blend(newColor, priorColor);
    framebuffer.write(blended, framebufferPosition);
}
```

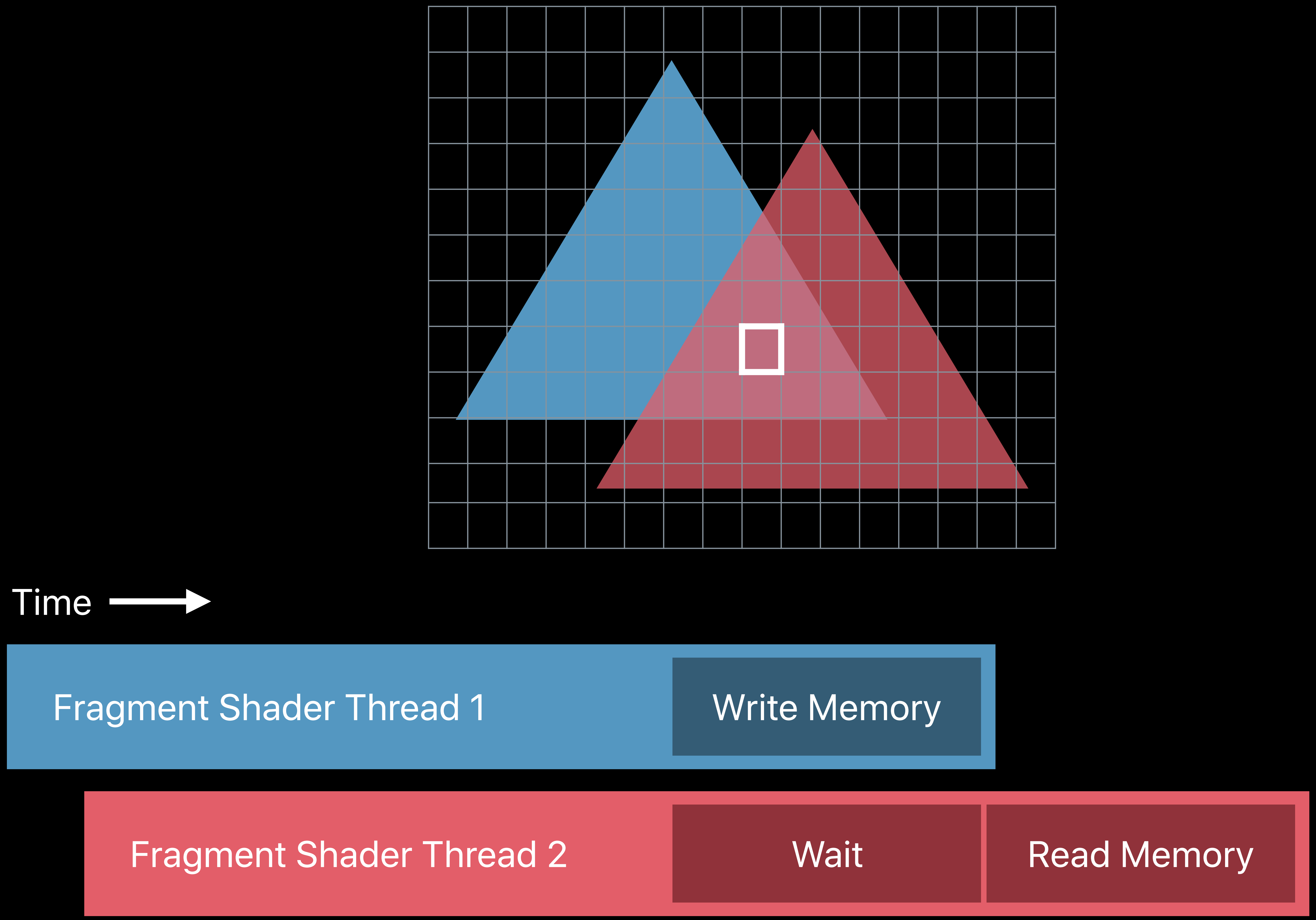
```
//  Blending manually to a pointer in memory
fragment void BlendSomething(
    texture2d<float, access::read_write> framebuffer [[texture(0)]]) {

   float4 newColor = …

    //  Non-atomic access to memory without synchronization
    float4 priorColor = framebuffer.read(framebufferPosition);
    float4 blended = custom_blend(newColor, priorColor);
    framebuffer.write(blended, framebufferPosition);
}
```

```
//  Blending manually to a pointer in memory
fragment void BlendSomething(
    texture2d<float, access::read_write> framebuffer [[texture(0), raster_order_group(0)]]) {

   float4 newColor = …

    //  Hardware waits on first access to raster ordered memory
    float4 priorColor = framebuffer.read(framebufferPosition);
    float4 blended = custom_blend(newColor, priorColor);
    framebuffer.write(blended, framebufferPosition);
}
```

With Raster Order Groups



With Raster Order Groups



Raster Order Groups

Summary

Synchronization between overlapping fragment shader threads

Check for support with `MTLDevice.rasterOrderGroupsSupported`

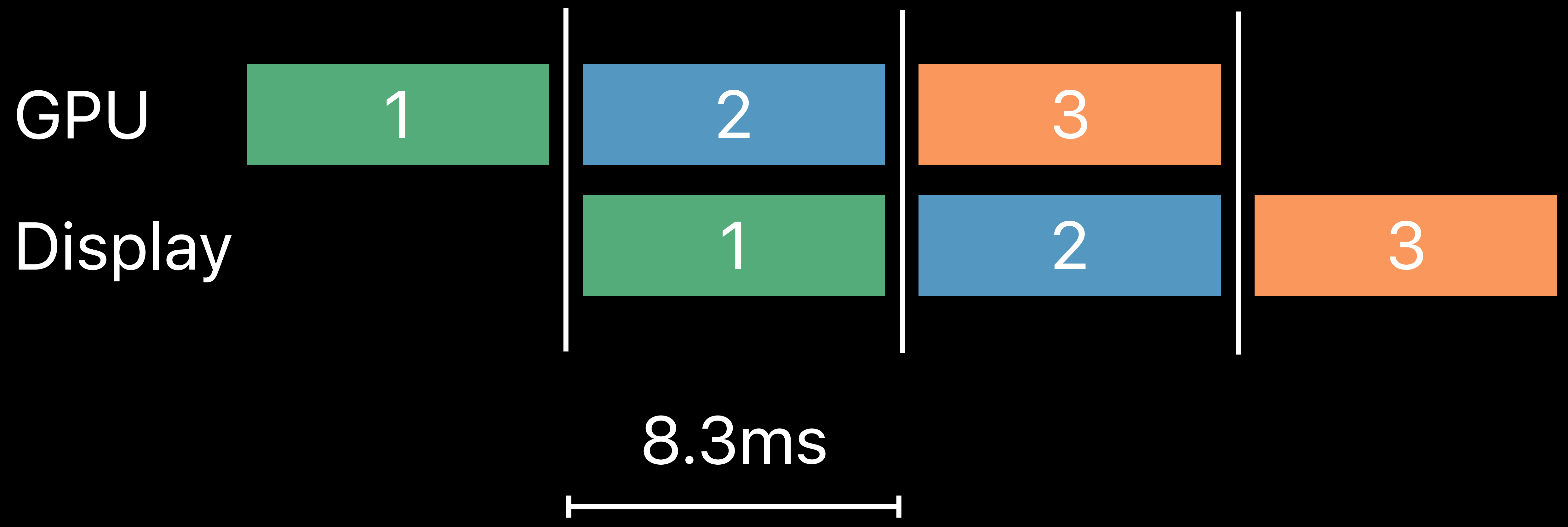
ProMotion Displays

Without ProMotion

60 FPS

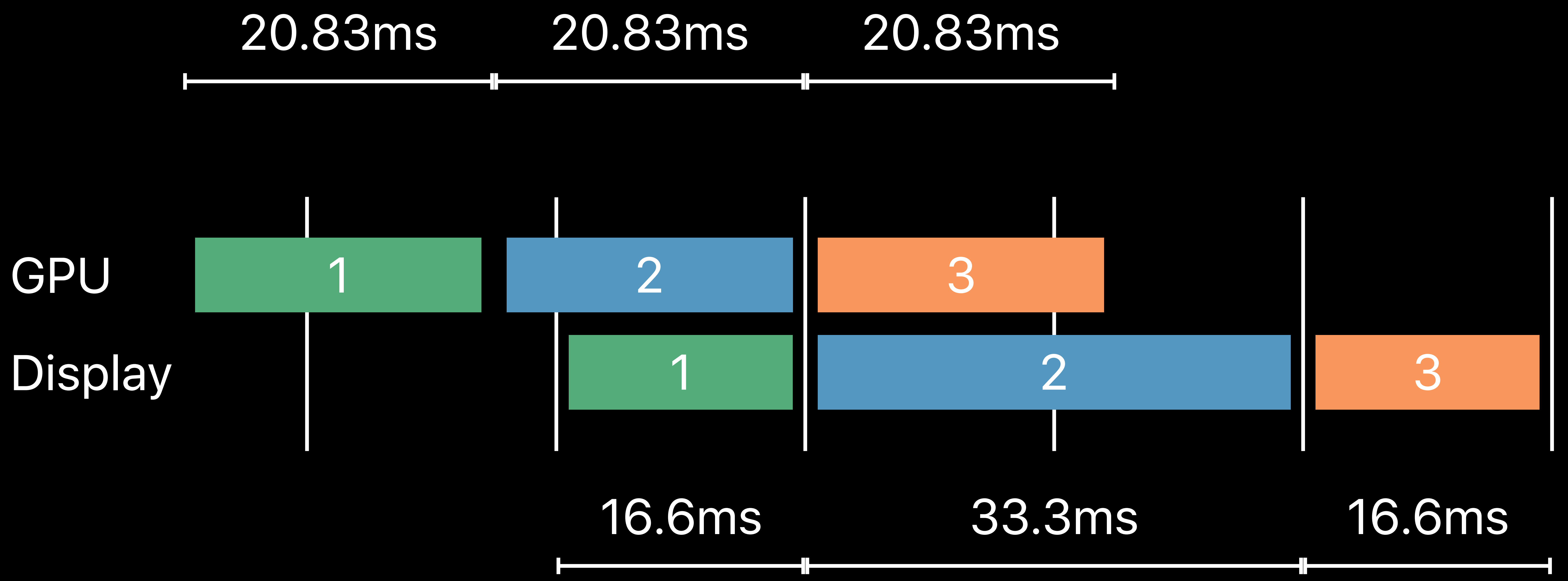
With ProMotion

120 FPS



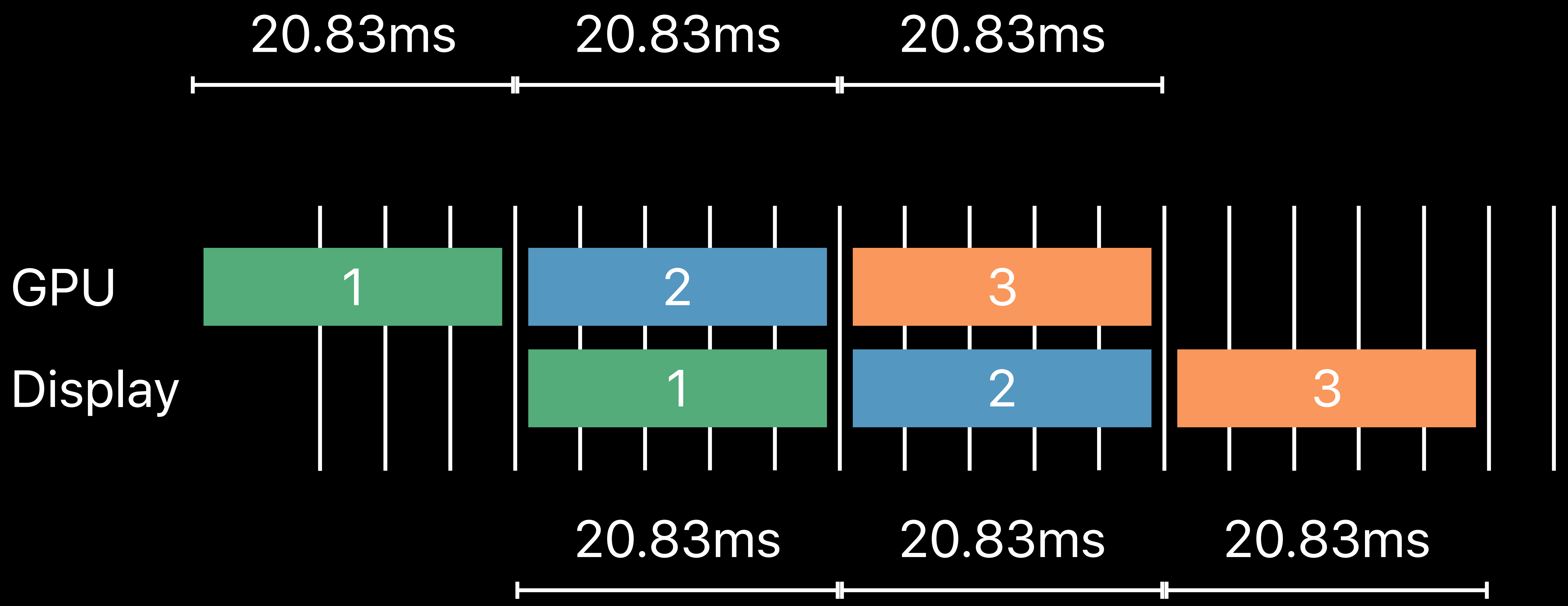
Without ProMotion

48 FPS



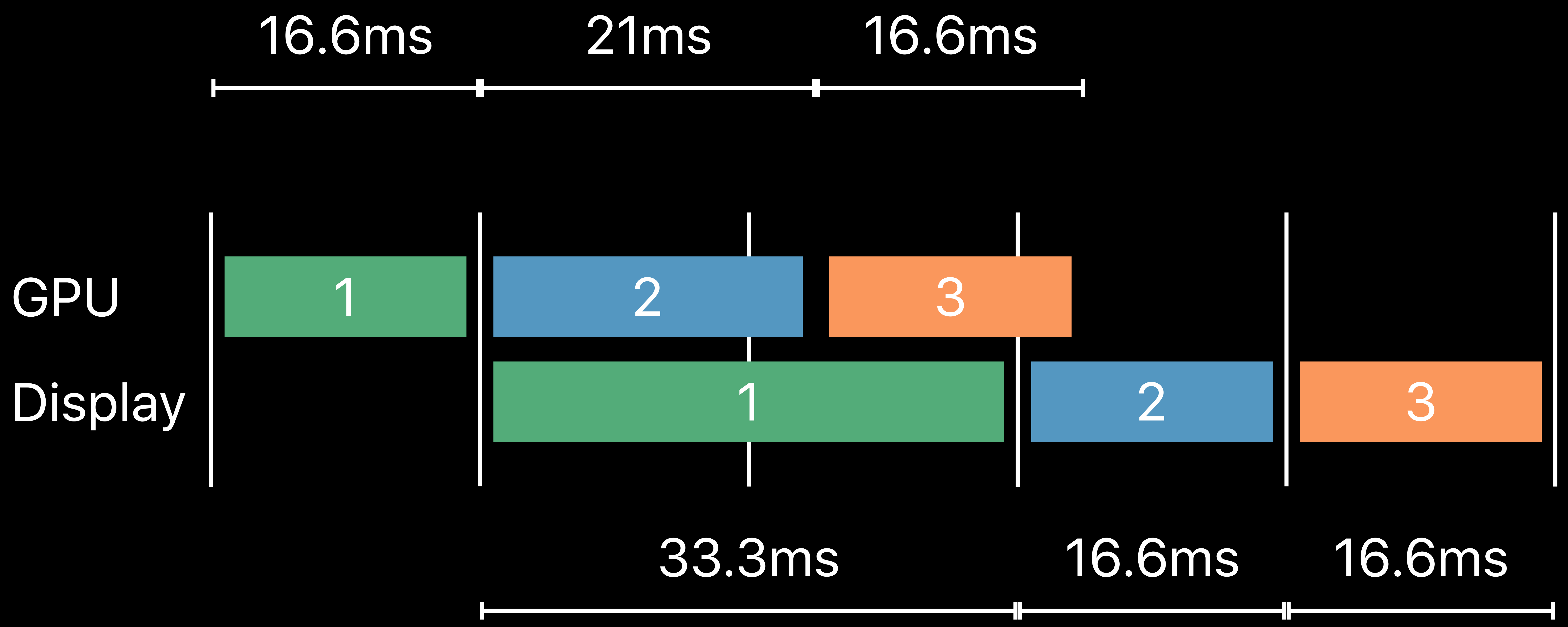
With ProMotion

48 FPS



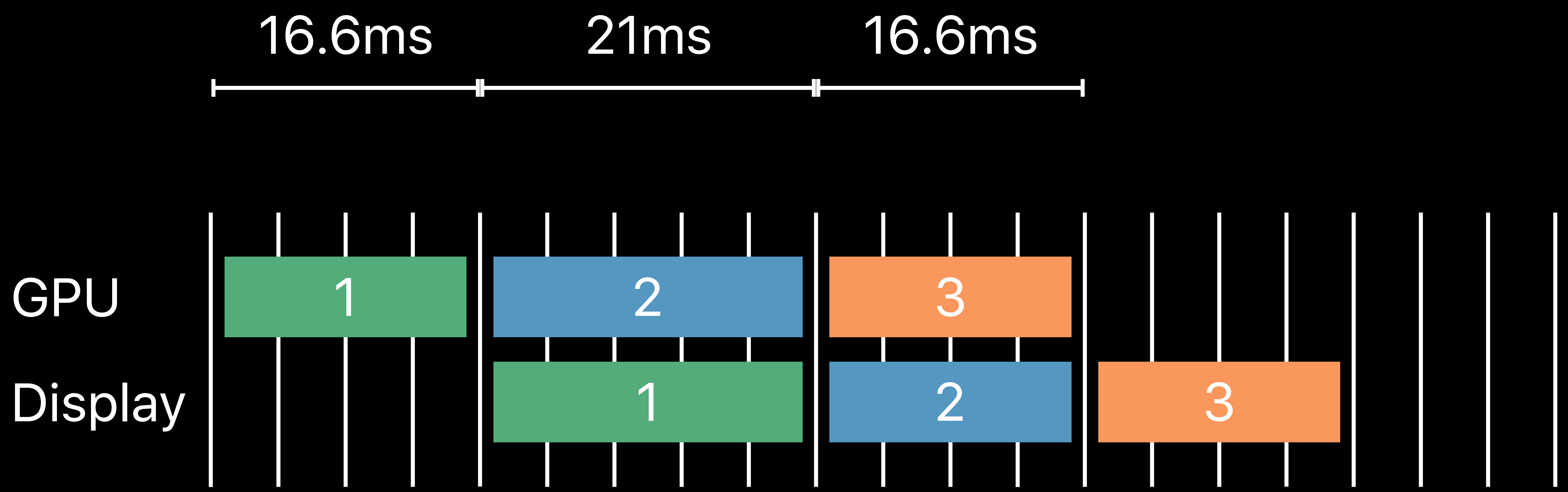
Without ProMotion

Dropped frame



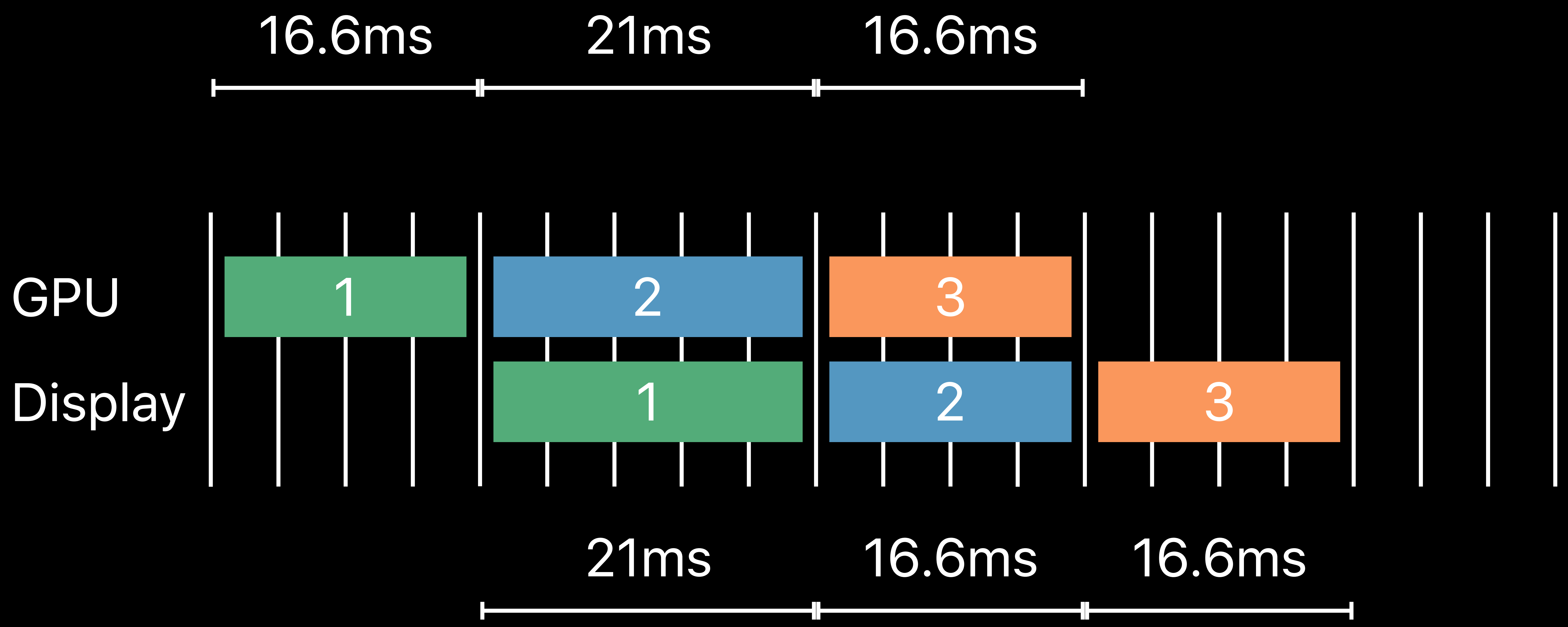
With ProMotion

Dropped frame



With ProMotion

Dropped frame



Opting in to ProMotion

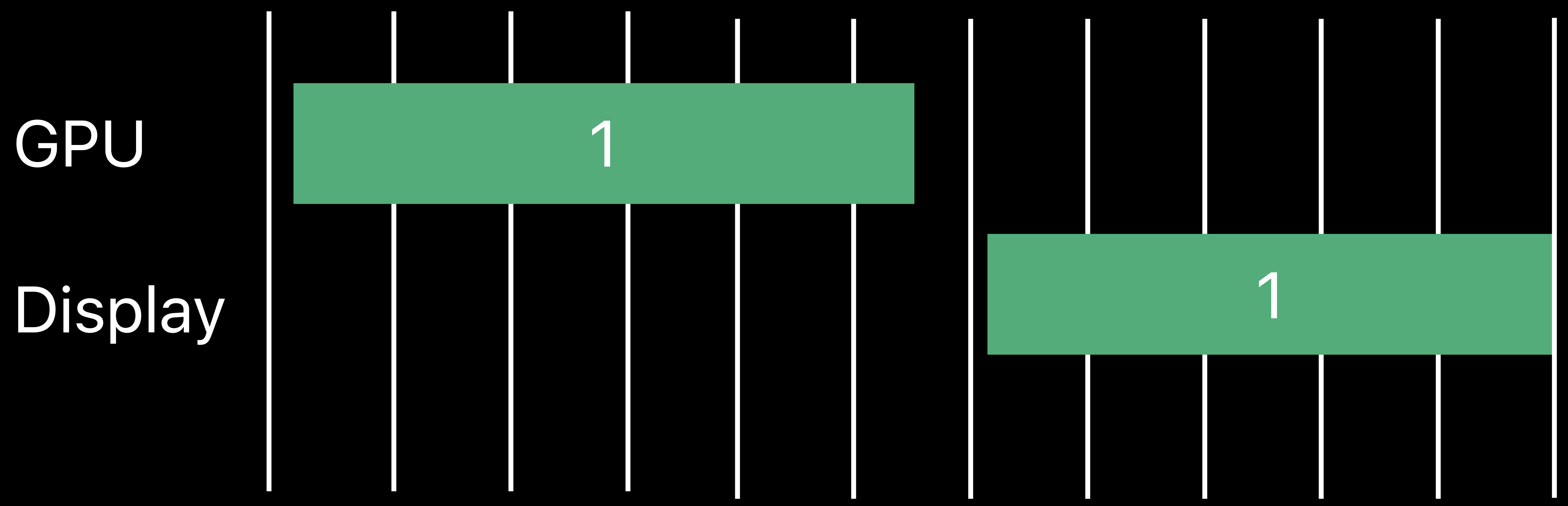
UIKit animations use ProMotion automatically

Metal views require opt-in with Info.plist key

```
<key>CADisableMinimumFrameDuration</key>
<true/>
```

Metal Presentation APIs

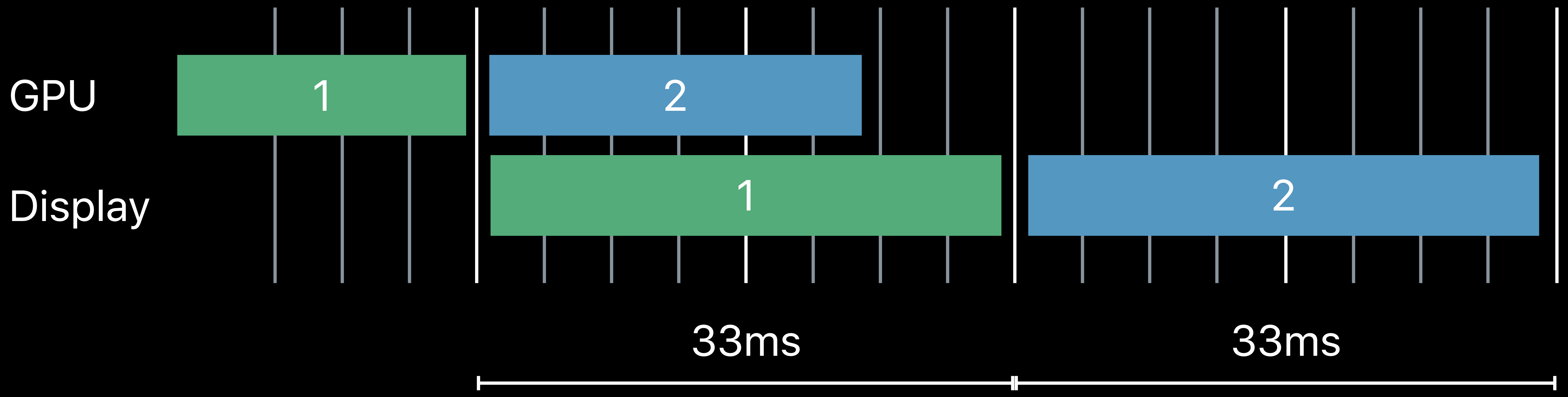
Present immediately




```
present(drawable)
```

Metal Presentation APIs

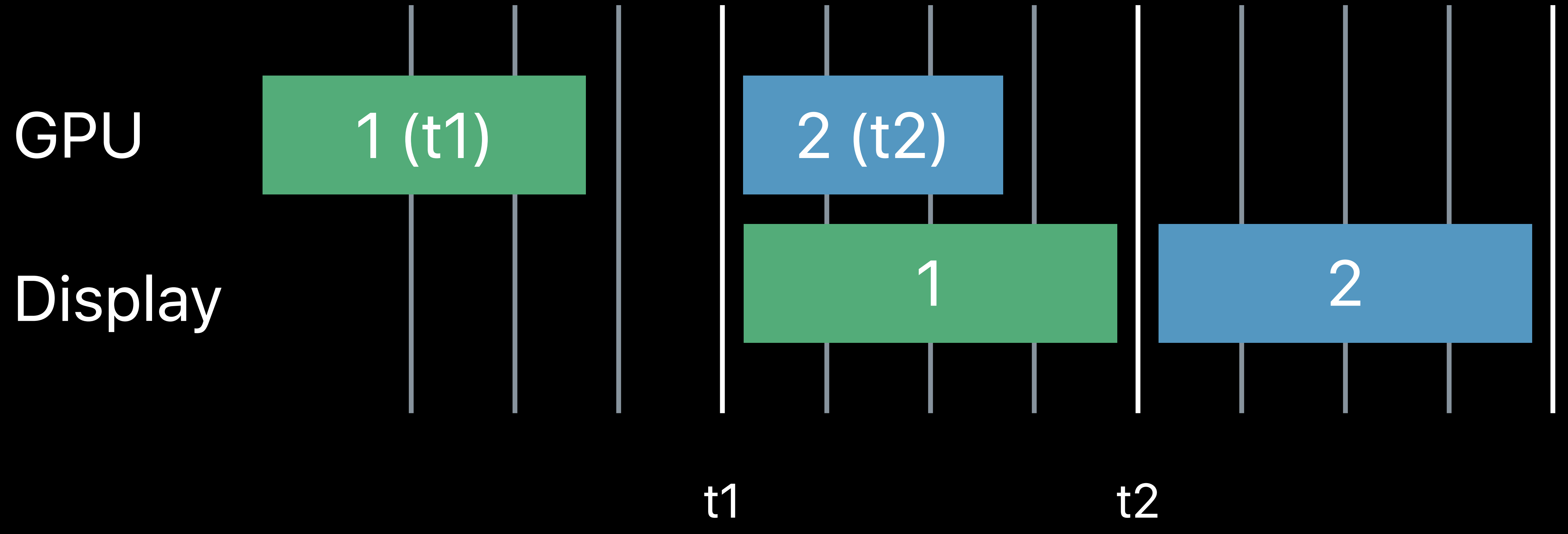
Present after minimum duration




```
present(drawable, afterMinimumDuration: 1000.0 / 30.0)
```

Metal Presentation APIs

Present at specific time




```
let time : CFTimeInterval = projectNextDisplayTime();
present(drawable, atTime: time)
```

```swift
let targetTime = //  project when intend to display this drawable
//  render your scene into a command buffer for ‘targetTime’
let drawable = metalLayer.nextDrawable()
commandBuffer.present(drawable, atTime: targetTime)



// after a frame or two…


let presentationDelay = drawable.presentedTime - targetTime
//  Examine presentationDelay and adjust future frame timing
```

ProMotion Displays

Summary

120 FPS

Reduced latency

Improved framerate consistency

Reduced stuttering from missed display deadlines

Direct to Display

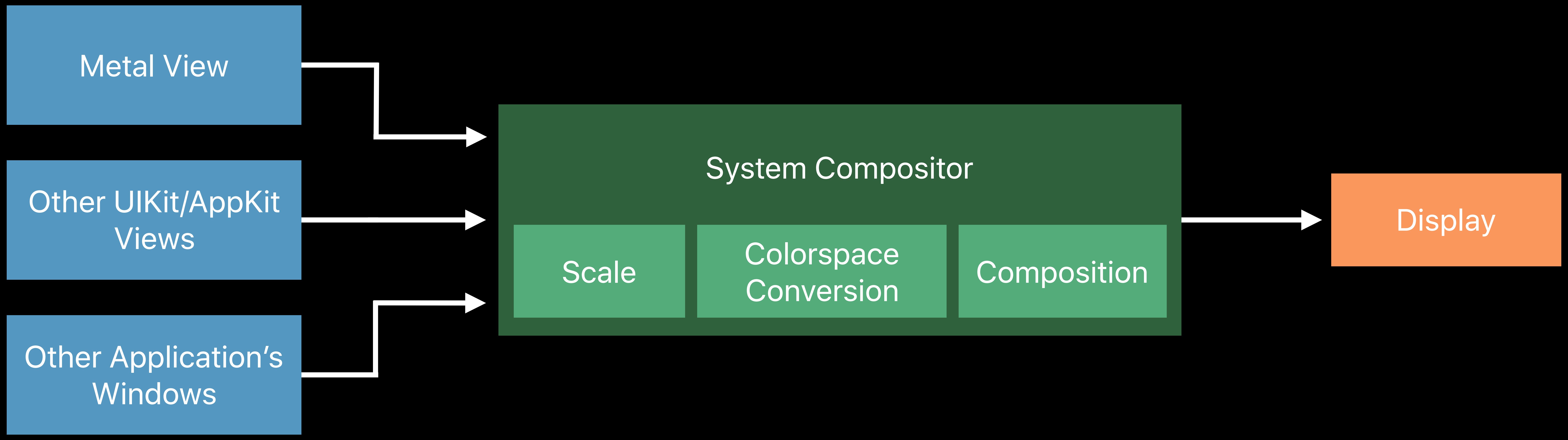
Displaying Metal Content

Two paths

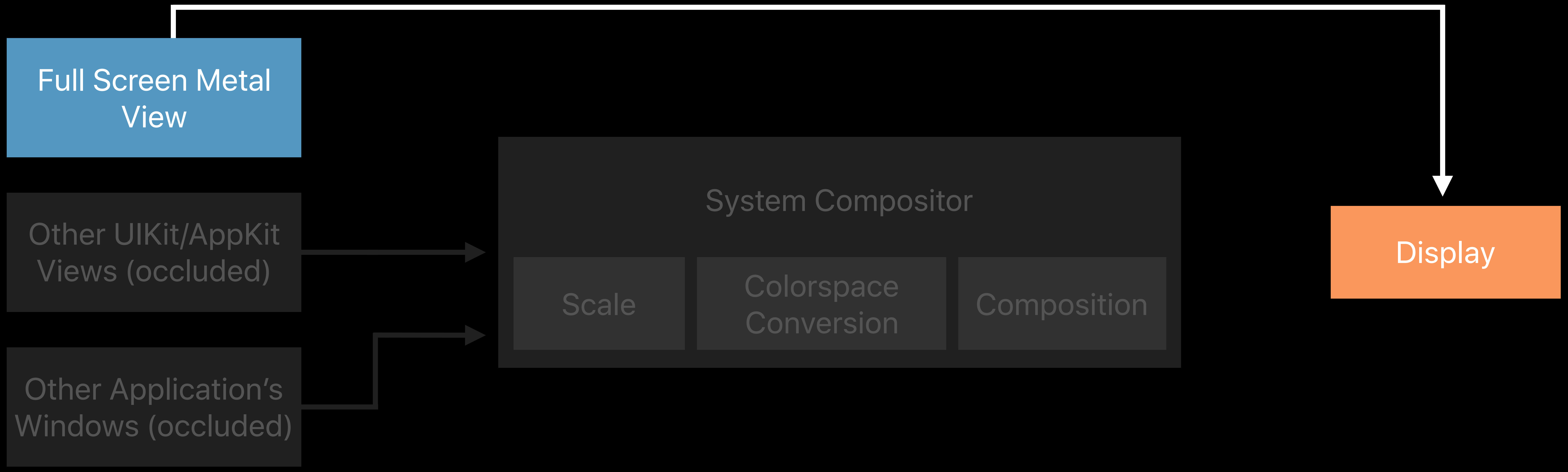
GPU Composition

Direct to Display

GPU Composition



Direct to Display



Direct to Display Requirements

Direct to Display Requirements

- ✓ Opaque Layer

Direct to Display Requirements

- Opaque Layer
- No masking, rounded corners, and so on

Direct to Display Requirements

- Opaque Layer
- No masking, rounded corners, and so on
- Full screen (or with “black bars” via an opaque black background color)

Direct to Display Requirements

- Opaque Layer
- No masking, rounded corners, and so on
- Full screen (or with “black bars” via an opaque black background color)
- Dimensions matching the display or smaller

Direct to Display Requirements

- Opaque Layer
- No masking, rounded corners, and so on
- Full screen (or with “black bars” via an opaque black background color)
- Dimensions matching the display or smaller
- Color Space and Pixel Format compatible with the display

Colorspace Requirements

Color Space	Metal Pixel Format	P3 Display	sRGB Display
sRGB	bgra8Unorm_srgb	Direct	Direct

Colorspace Requirements

Color Space	Metal Pixel Format	P3 Display	sRGB Display
sRGB	`bgra8Unorm_srgb`	Direct	Direct
Display P3 (macOS)	`bgr10a2Unorm`	Direct	GPU Composited

Colorspace Requirements

Color Space	Metal Pixel Format	P3 Display	sRGB Display
sRGB	`bgra8Unorm_srgb`	Direct	Direct
Display P3 (macOS)	`bgr10a2Unorm`	Direct	GPU Composited
Extended sRGB (iOS)	`bgr10_xr_srgb`	Direct	GPU Composited

Colorspace Requirements

Color Space	Metal Pixel Format	P3 Display	sRGB Display
sRGB	`bgra8Unorm_srgb`	Direct	Direct
Display P3 (macOS)	`bgr10a2Unorm`	Direct	GPU Composited
Extended sRGB (iOS)	`bgr10_xr_srgb`	Direct	GPU Composited
Linear Extended sRGB, or Display P3	`rgba16Float`	GPU Composited	GPU Composited

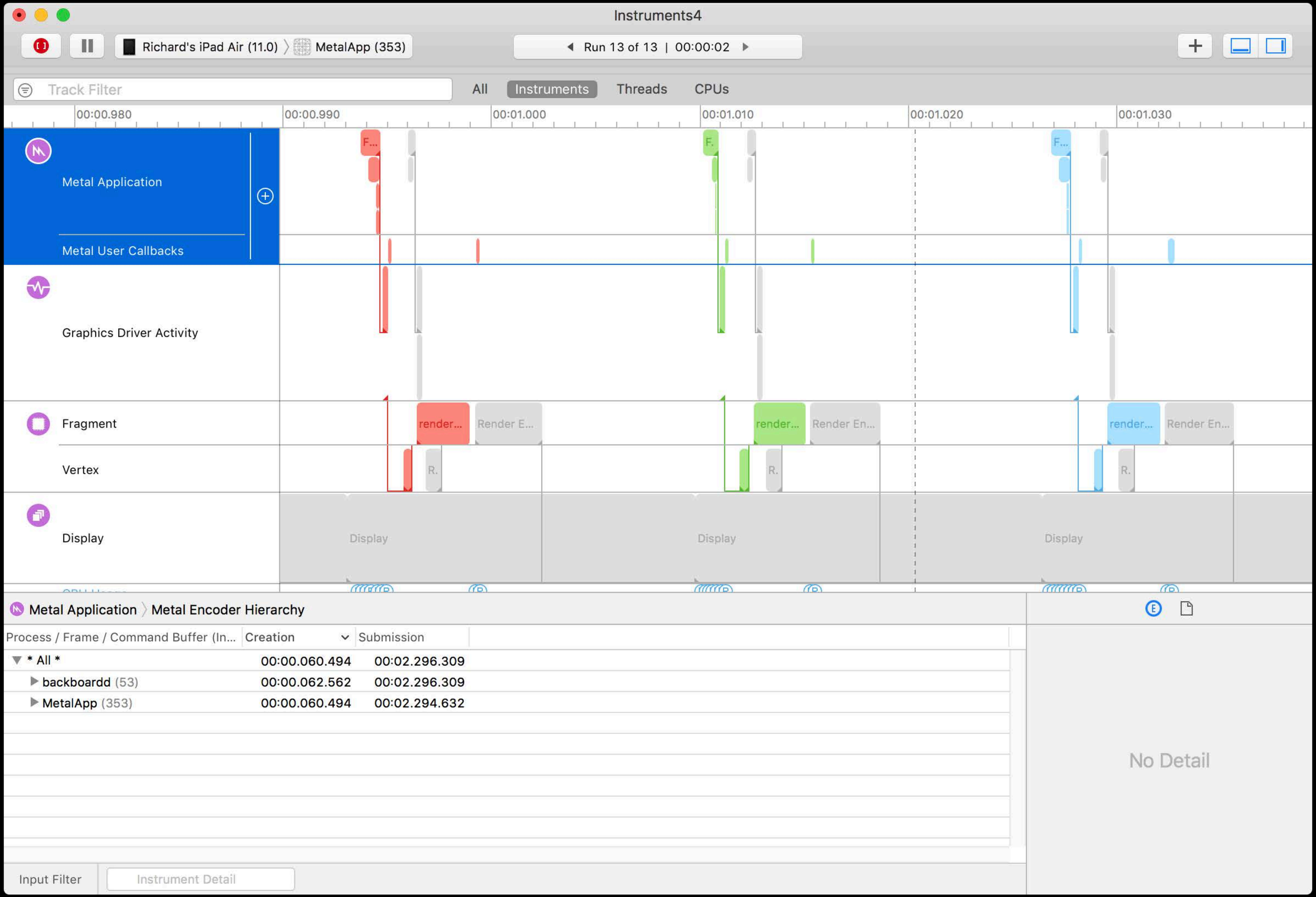
Detecting P3 Display Gamut

UIKit

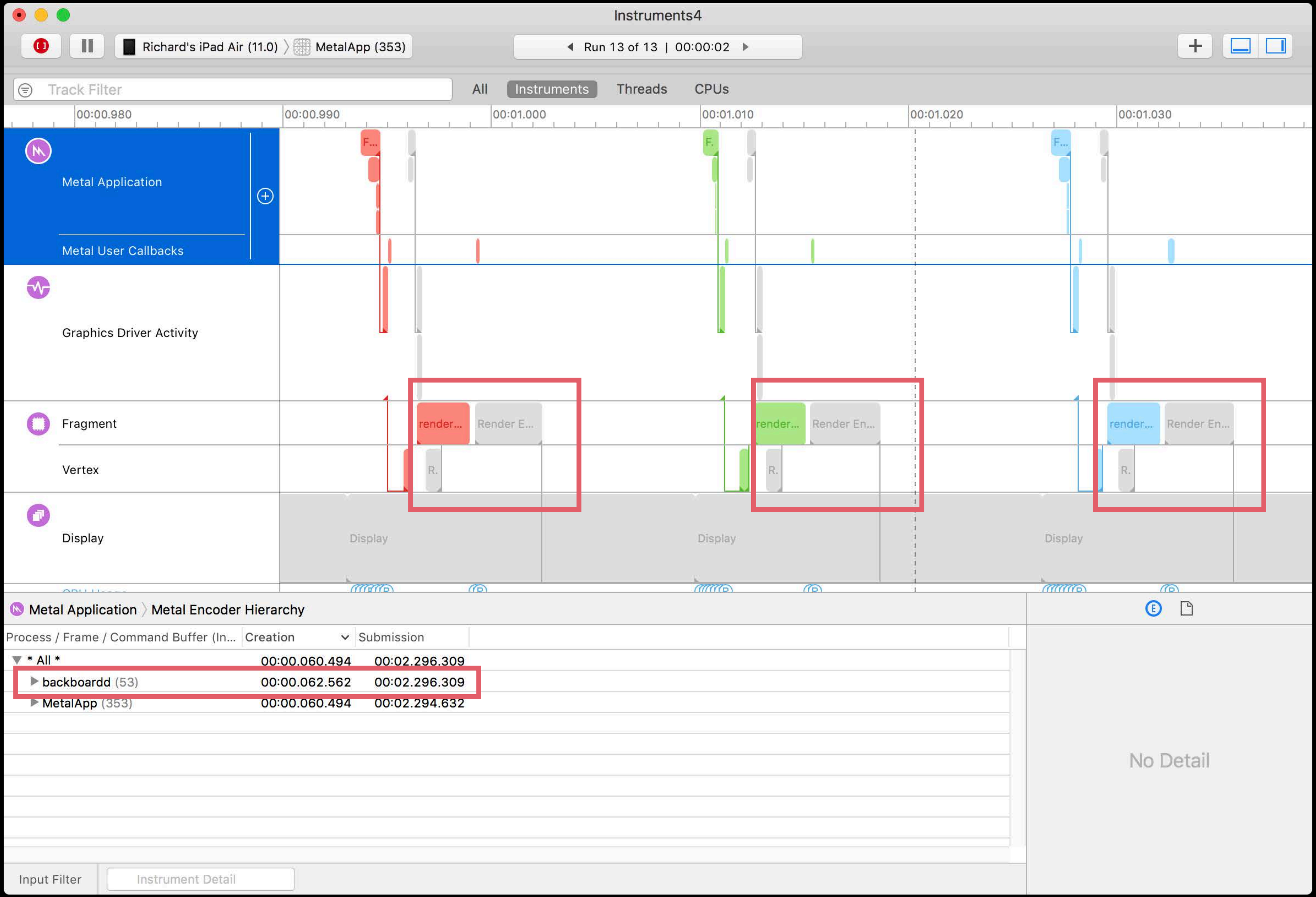
```
UITraitCollection.displayGamut == .P3
```

AppKit

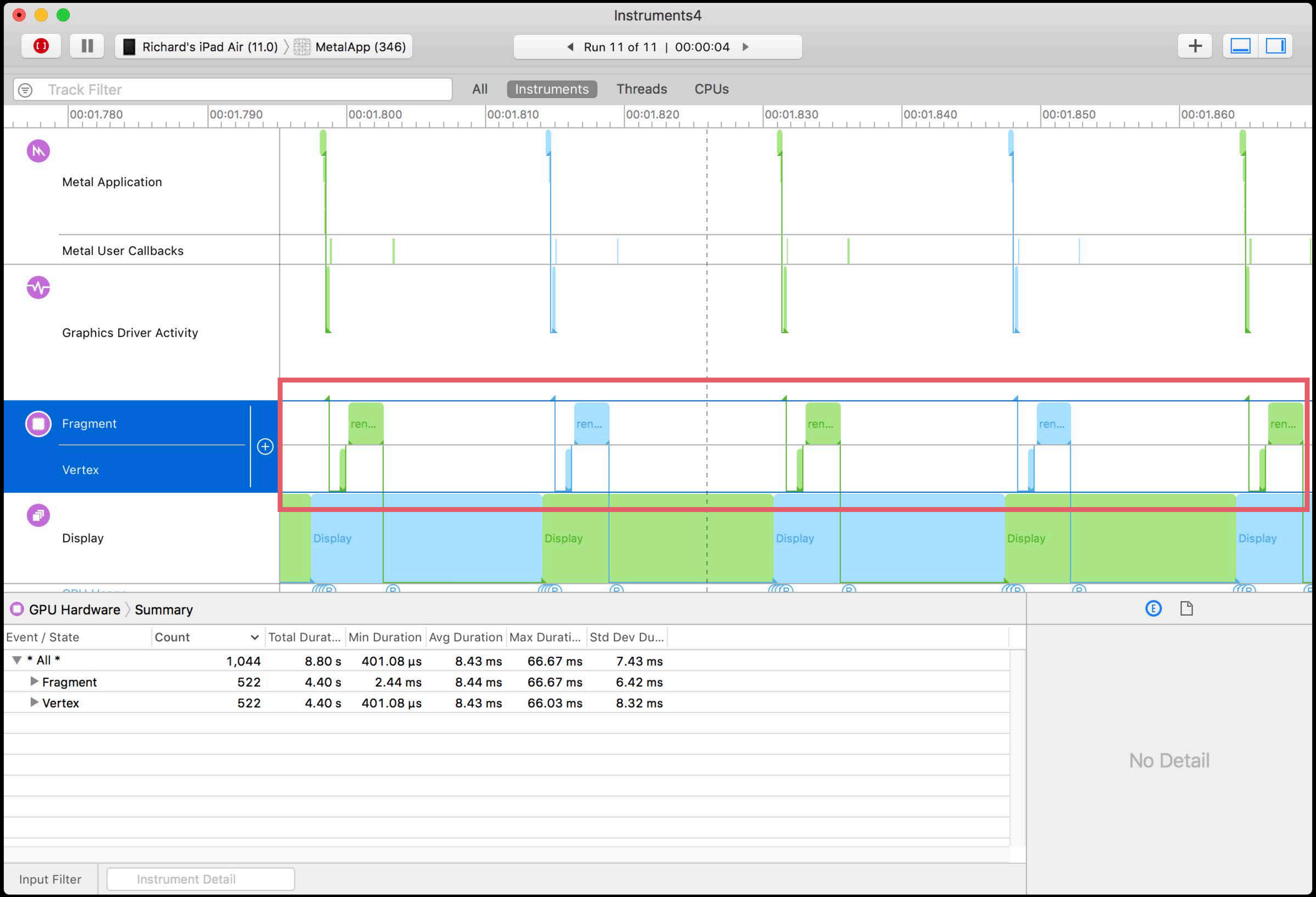
```
NSScreen.canRepresent(.p3)
```



Instruments4
Richard's iPad Air (11.0)
MetalApp (353)
Run 13 of 13 | 00:00:02
Track Filter
All
Instruments
Threads
CPUs
00:00.980
00:00.990
00:01.000
00:01.010
00:01.020
00:01.030
Metal Application
Metal User Callbacks
Graphics Driver Activity
Fragment
Vertex
Display
render...
Render E...
Render En...
Display
Metal Application
Metal Encoder Hierarchy
Process / Frame / Command Buffer (In...
Creation
Submission
* All *
00:00.060.494
00:02.296.309
backboardd (53)
00:00.062.562
00:02.296.309
MetalApp (353)
00:00.060.494
00:02.294.632
No Detail
Input Filter
Instrument Detail



Instruments4
Richard's iPad Air (11.0)
MetalApp (353)
Run 13 of 13 | 00:00:02
Track Filter
All
Instruments
Threads
CPUs
00:00.980
00:00.990
00:01.000
00:01.010
00:01.020
00:01.030
Metal Application
Metal User Callbacks
Graphics Driver Activity
Fragment
Vertex
Display
render...
Render E...
Render En...
Metal Application
Metal Encoder Hierarchy
Process / Frame / Command Buffer (In...
Creation
Submission
* All *
00:00.060.494
00:02.296.309
backboardd (53)
00:00.062.562
00:02.296.309
MetalApp (353)
00:00.060.494
00:02.294.632
No Detail
Input Filter
Instrument Detail


Instruments4
Richard's iPad Air (11.0)
MetalApp (346)
Run 11 of 11 | 00:00:04
Track Filter
All
Instruments
Threads
CPUs
00:01.780
00:01.790
00:01.800
00:01.810
00:01.820
00:01.830
00:01.840
00:01.850
00:01.860
Metal Application
Metal User Callbacks
Graphics Driver Activity
Fragment
Vertex
ren...
Display
GPU Hardware
Summary
Event / State
Count
Total Durat...
Min Duration
Avg Duration
Max Durati...
Std Dev Du...
* All * 1,044 8.80 s 401.08 µs 8.43 ms 66.67 ms 7.43 ms
Fragment 522 4.40 s 2.44 ms 8.44 ms 66.67 ms 6.42 ms
Vertex 522 4.40 s 401.08 µs 8.43 ms 66.03 ms 8.32 ms
No Detail
Input Filter
Instrument Detail

Direct to Display

Summary

Eliminate compositor usage of GPU

Useful for full-screen scenes

Supported on iOS, tvOS, and macOS

Use Metal System Trace to verify

Everything Else

Memory Usage Queries

New APIs to query memory usage per allocation

```
MTLResource.allocatedSize
MTLHeap.currentAllocatedSize
```

Query total GPU memory allocated by the device

```
MTLDevice.currentAllocatedSize
```

SIMDGroup-scoped Data Sharing

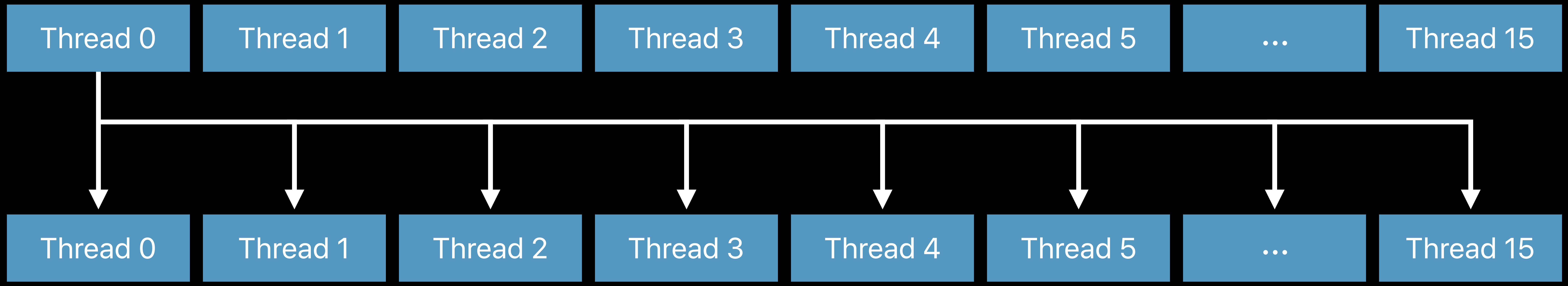
Share data across a SIMDGroup

```
simd_broadcast(data, simd_lane_id)
simd_shuffle(data, simd_lane_id)
simd_shuffle_up(data, simd_lane_id)
simd_shuffle_down(data, simd_lane_id)
simd_shuffle_xor(data, simd_lane_id)
```

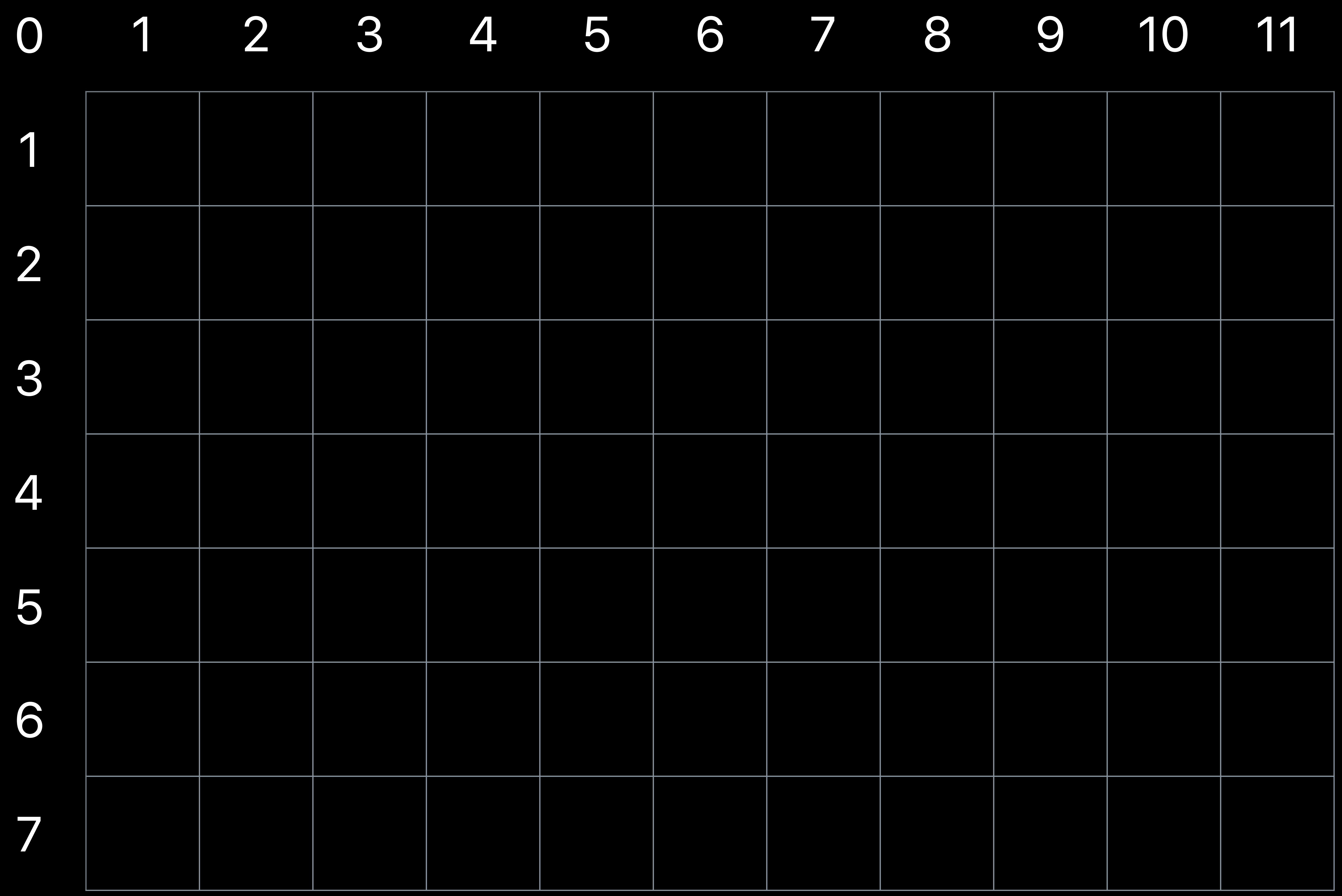
SIMDGroup-scoped Data Sharing

Share data across a SIMDGroup

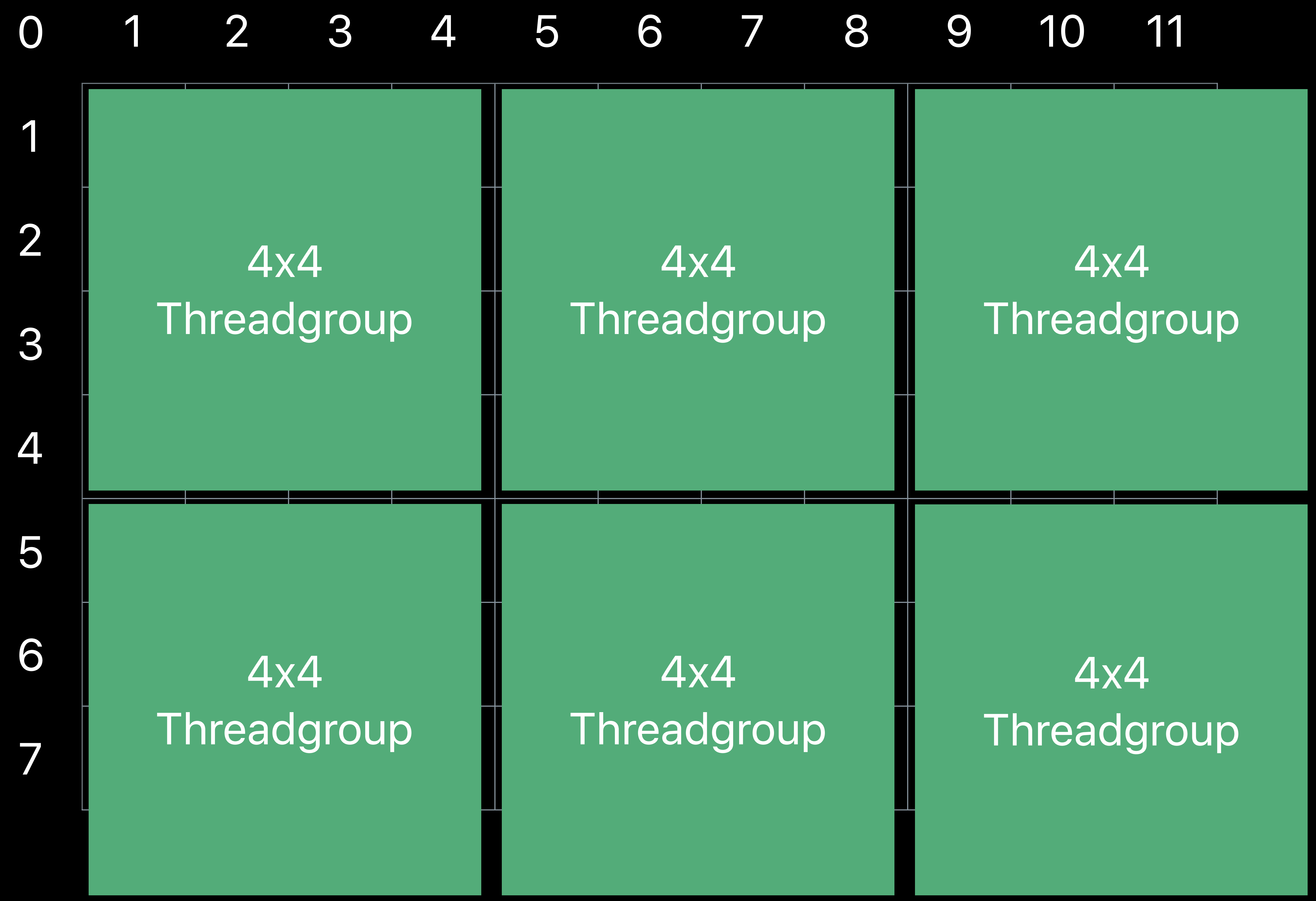
```
simd_broadcast(data, simd_lane_id)
simd_shuffle(data, simd_lane_id)
simd_shuffle_up(data, simd_lane_id)
simd_shuffle_down(data, simd_lane_id)
simd_shuffle_xor(data, simd_lane_id)
```



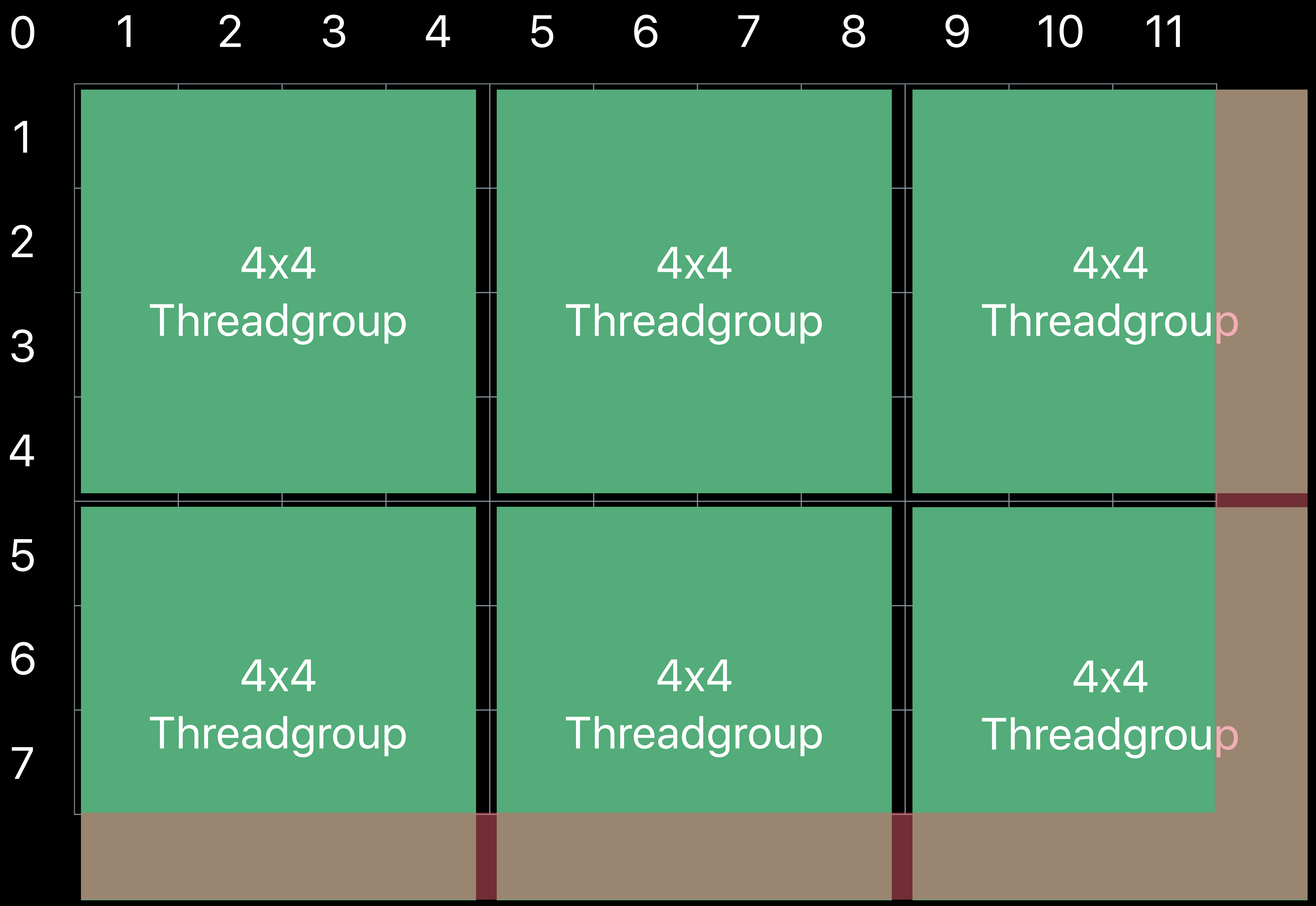
Non-Uniform Threadgroup Sizes



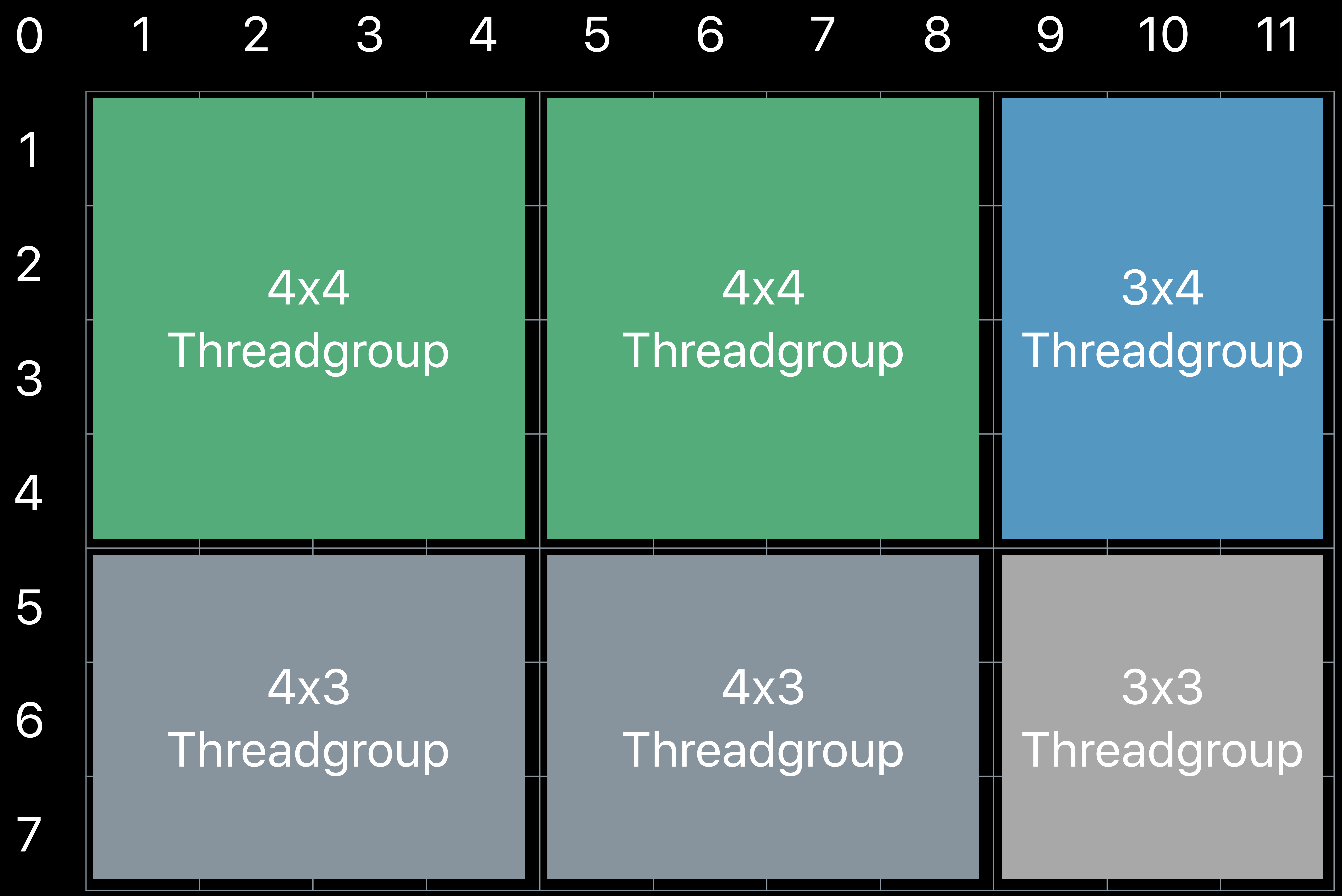
Non-Uniform Threadgroup Sizes



Non-Uniform Threadgroup Sizes



Non-Uniform Threadgroup Sizes



Viewport Arrays

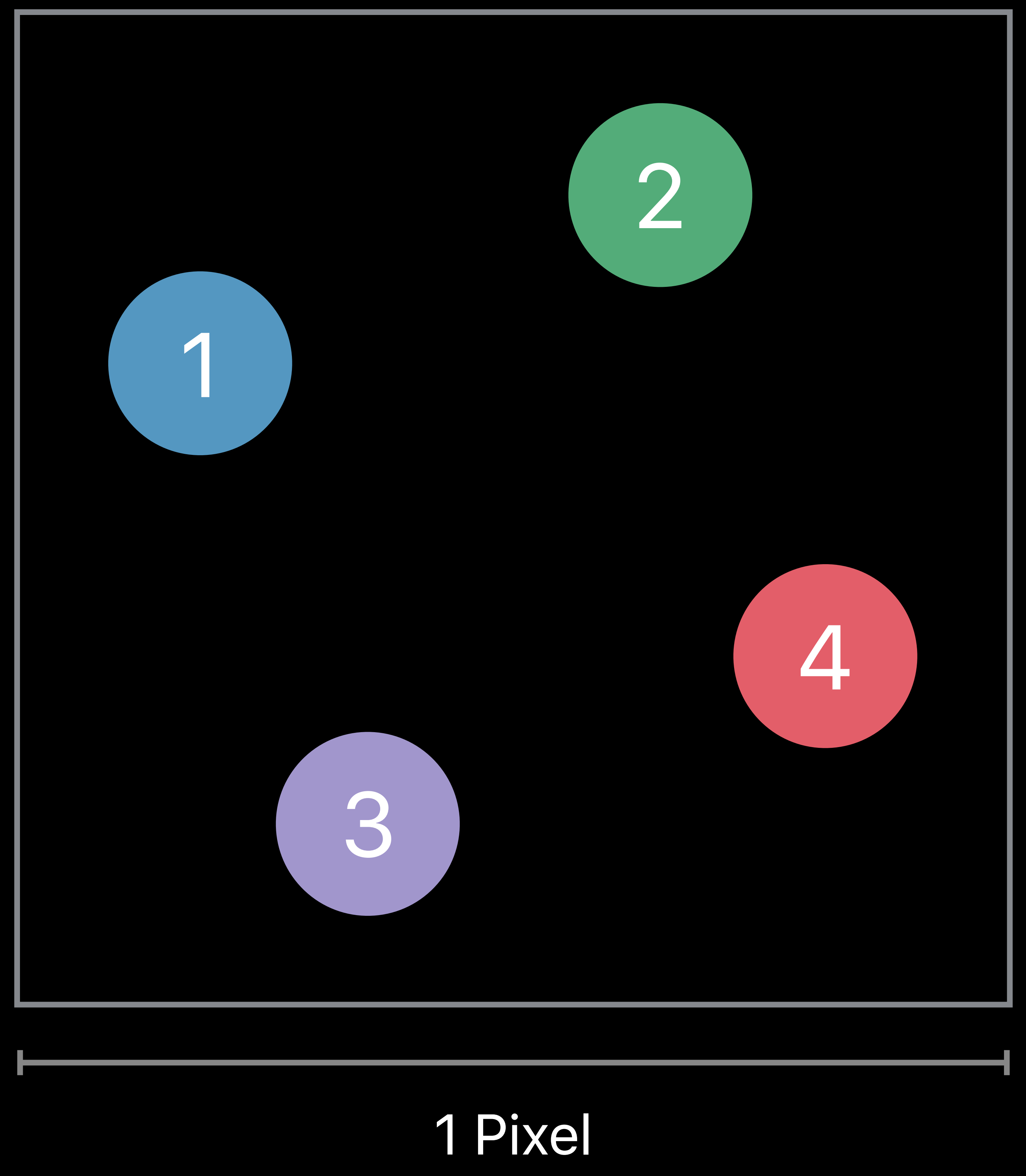
Vertex Function selects which viewport

Useful for VR when combined with instancing

Multisample Pattern Control

Select where within a pixel the MSAA Sample Patterns are located

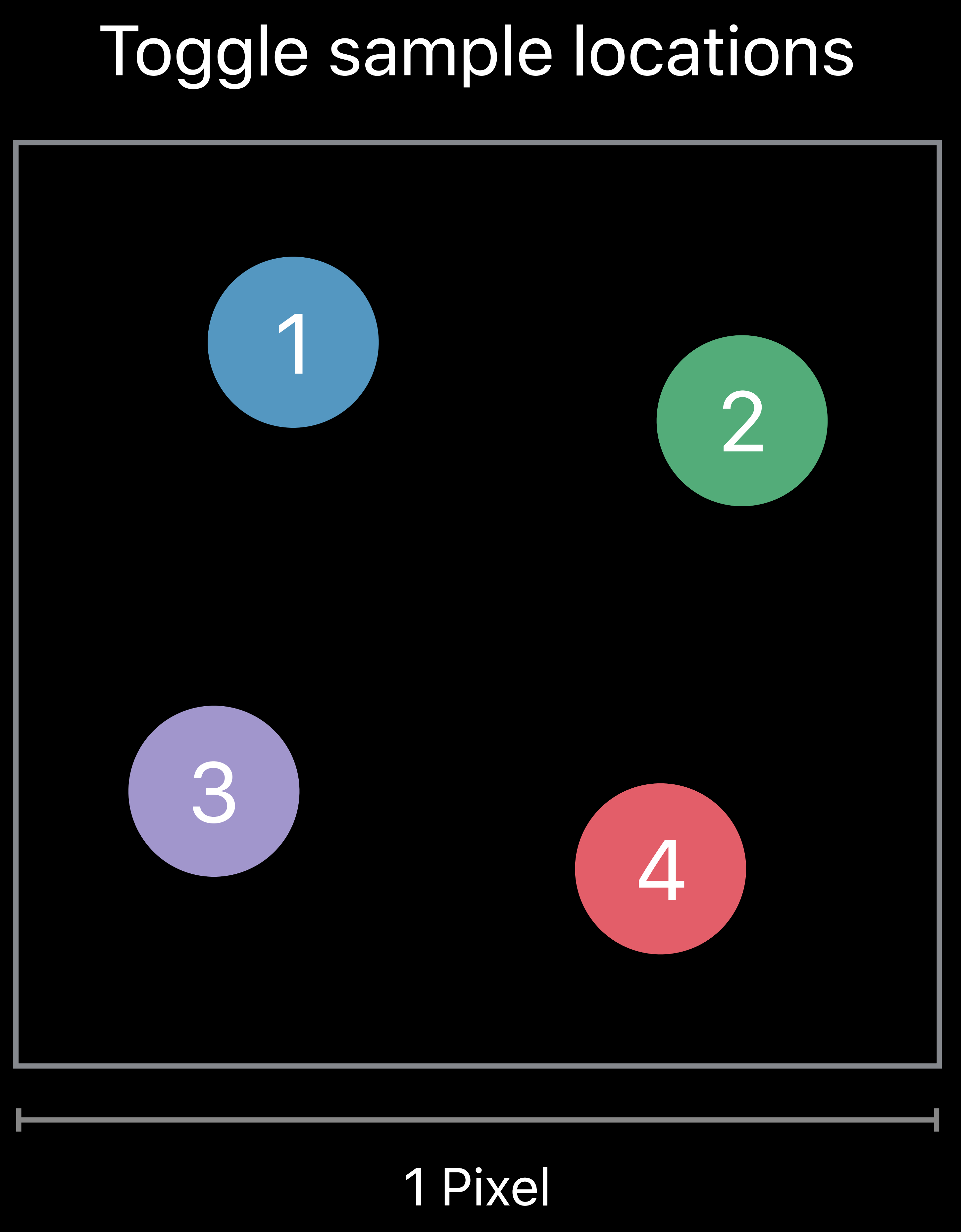
Useful for custom anti-aliasing



Multisample Pattern Control

Select where within a pixel the MSAA Sample Patterns are located

Useful for custom anti-aliasing



Resource Heaps

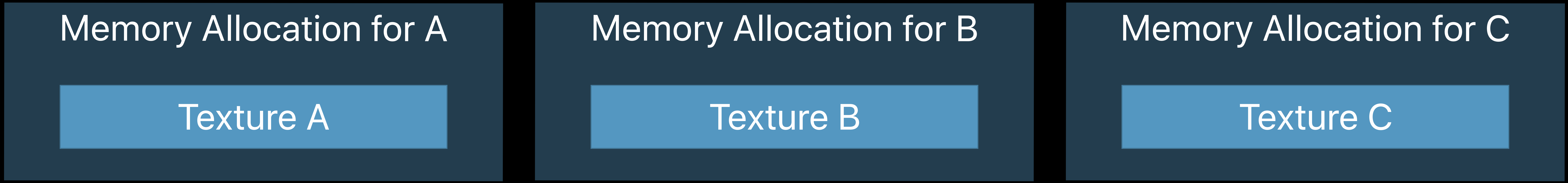
Now available on macOS

Control time of memory allocation

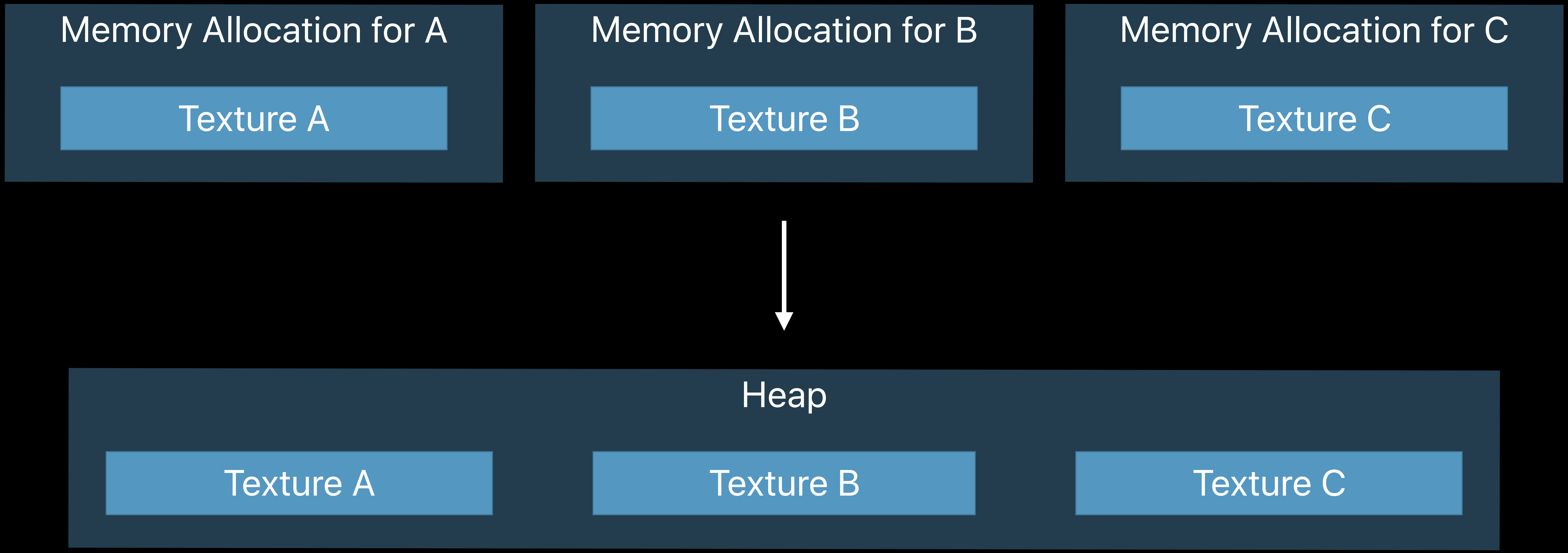
Fast reallocation and aliasing of resources

Group related resources for faster binding

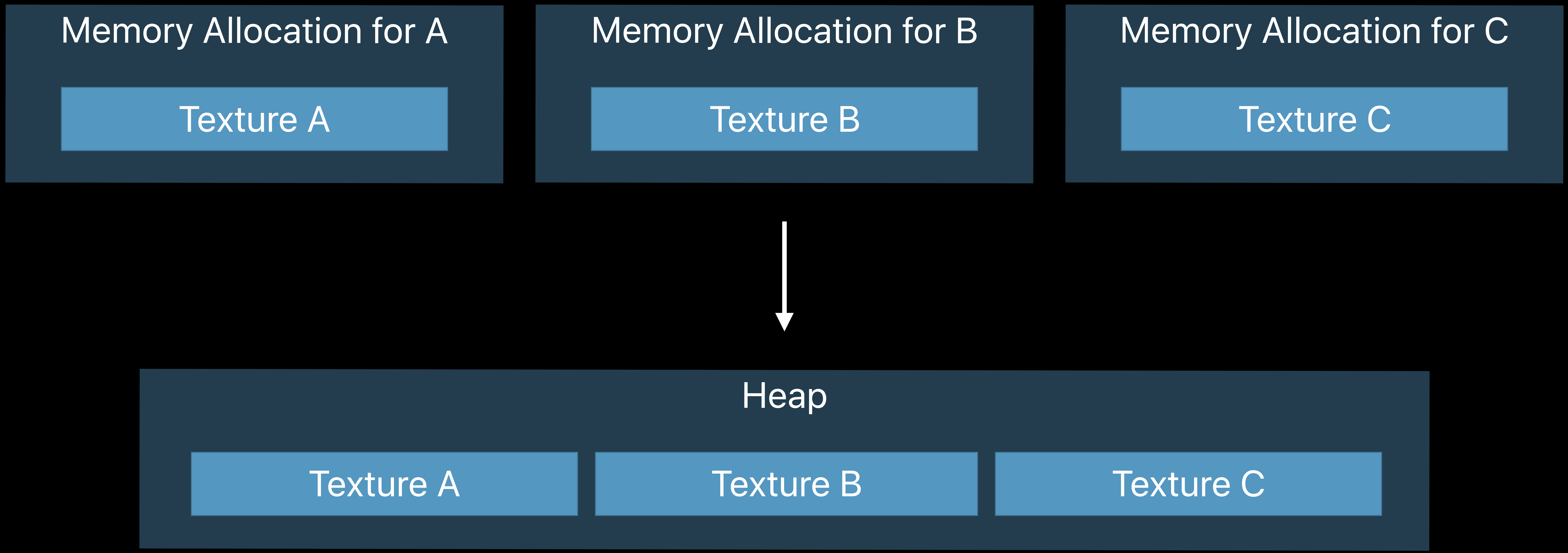
Resource Heaps



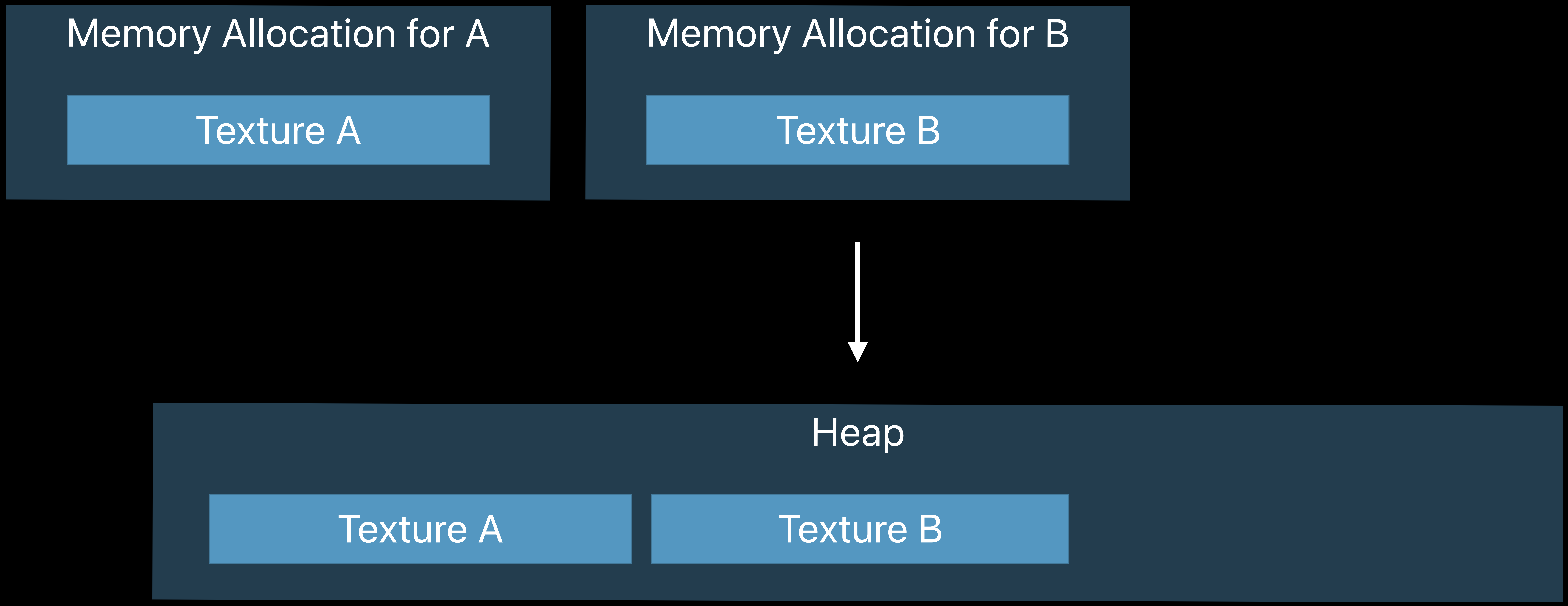
Resource Heaps



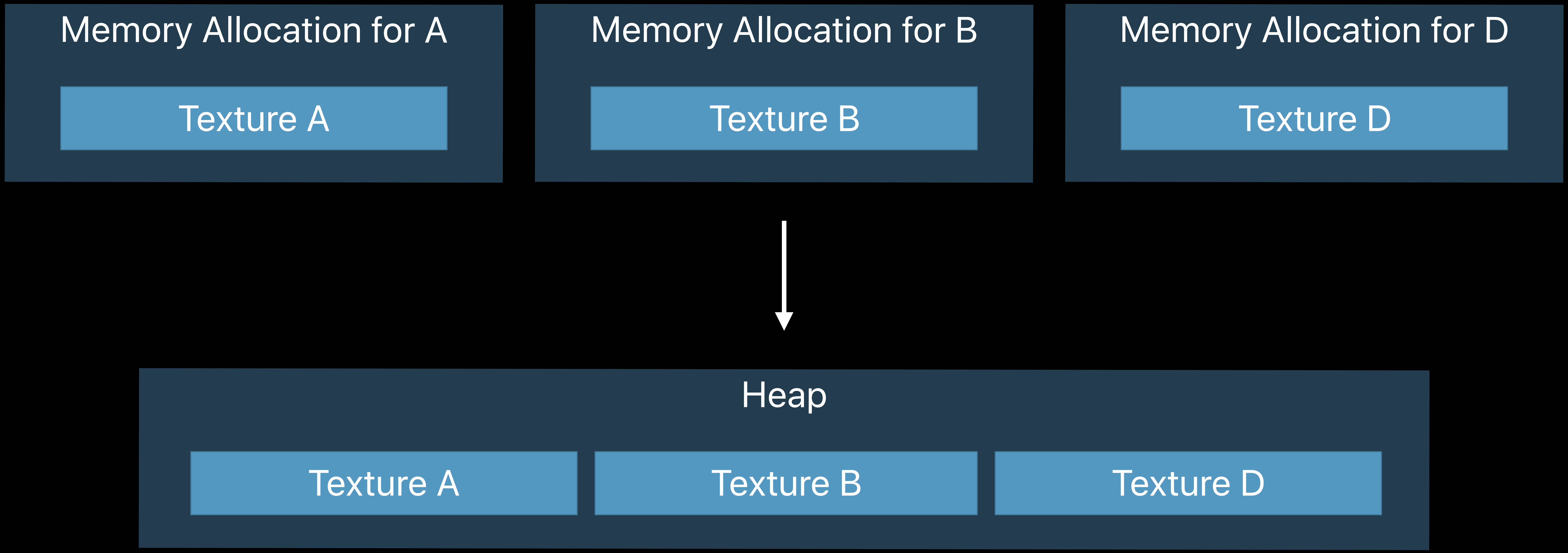
Resource Heaps



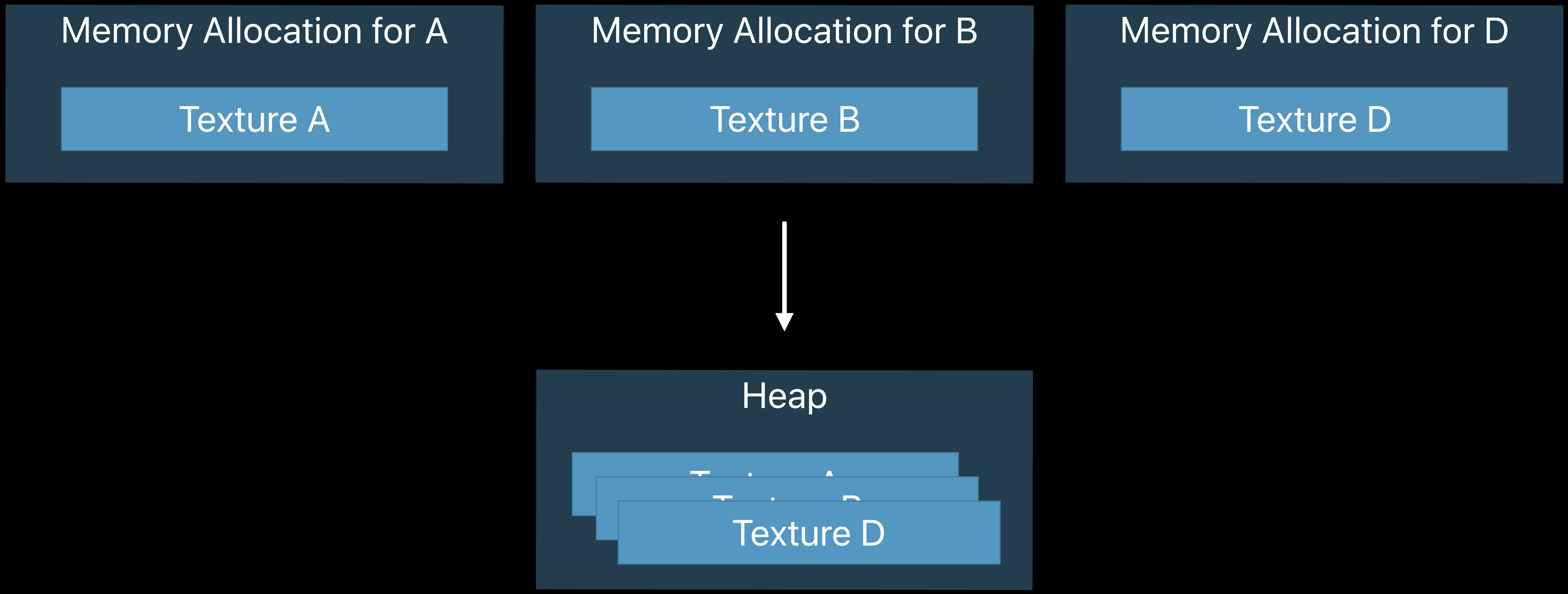
Resource Heaps



Resource Heaps



Resource Heaps



Other Features

Feature	Summary

Other Features

Feature	Summary
Linear Textures	Create textures from a MTLBuffer without copying

Other Features

Feature	Summary
Linear Textures	Create textures from a MTLBuffer without copying
Function Constant for Argument Indexes	Specialize bytecodes to change the binding index for shader arguments

Other Features

Feature	Summary
Linear Textures	Create textures from a MTLBuffer without copying
Function Constant for Argument Indexes	Specialize bytecodes to change the binding index for shader arguments
Additional Vertex Array Formats	Add some additional 1 component and 2 component vertex formats, and a BGRA8 vertex format

Other Features

Feature	Summary
Linear Textures	Create textures from a MTLBuffer without copying
Function Constant for Argument Indexes	Specialize bytecodes to change the binding index for shader arguments
Additional Vertex Array Formats	Add some additional 1 component and 2 component vertex formats, and a BGRA8 vertex format
IOSurface Textures	Create MTLTextures from IOSurfaces on iOS

Other Features

Feature	Summary
Linear Textures	Create textures from a MTLBuffer without copying
Function Constant for Argument Indexes	Specialize bytecodes to change the binding index for shader arguments
Additional Vertex Array Formats	Add some additional 1 component and 2 component vertex formats, and a BGRA8 vertex format
IOSurface Textures	Create MTLTextures from IOSurfaces on iOS
Dual Source Blending	Additional blending modes with two source parameters

Summary

Argument Buffers

Raster Order Groups

ProMotion Displays

Direct to Display

Everything Else

More Information

https://developer.apple.com/wwdc17/601

Related Sessions

VR with Metal 2	Hall 3	Wednesday 10:00AM
Metal 2 Optimization and Debugging	Executive Ballroom	Thursday 3:10PM
Using Metal 2 for Compute	Grand Ballroom A	Thursday 4:10PM

Previous Sessions

What's New in Metal, Part 1	WWDC 2016
Working with Wide Color	WWDC 2016

Labs

Metal 2 Lab	Technology Lab A	Tues 3:10–6:10PM
VR with Metal 2 Lab	Technology Lab A	Wed 3:10–6:00PM
Metal 2 Lab	Technology Lab F	Fri 9:00AM–12:00PM

WWDC17