#WWDC18

Metal for OpenGL Developers

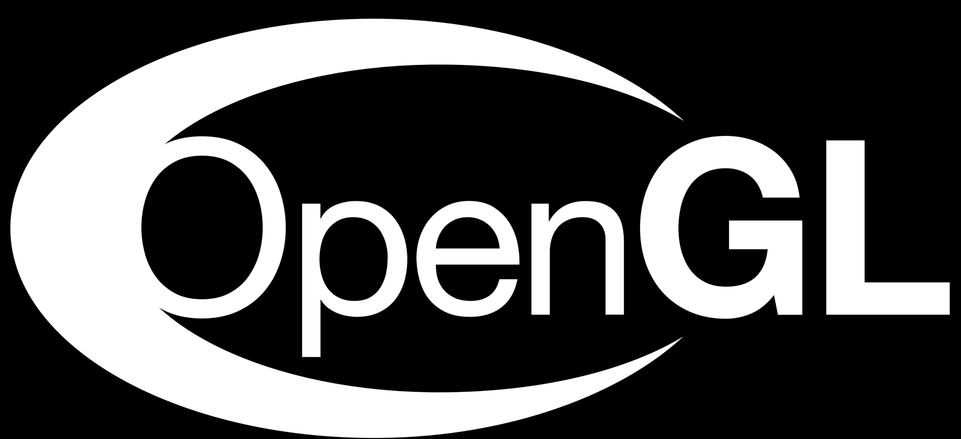
Dan Omachi, Metal Ecosystem Engineer
Sukanya Sudugu, GPU Software Engineer

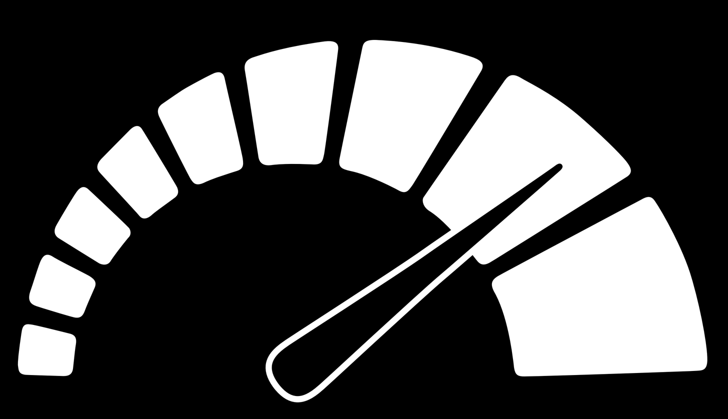
© 2018 Apple Inc. All rights reserved. Redistribution or public display not permitted without written permission from Apple.

These legacy APIs are deprecated

Still available in iOS 12,
macOS 10.14 and tvOS 12

Begin transitioning to Metal





OpenCL

Choosing an Approach

High-level Apple frameworks

- SpriteKit, SceneKit, and Core Image

Choosing an Approach

High-level Apple frameworks

- SpriteKit, SceneKit, and Core Image

Third-party engines

- Unity, Unreal, Lumberyard, etc.
- Update to latest version

Choosing an Approach

High-level Apple frameworks

- SpriteKit, SceneKit, and Core Image

Third-party engines

- Unity, Unreal, Lumberyard, etc.
- Update to latest version

Focus today—incrementally porting to Metal

Metal Concepts

Challenges with OpenGL

OpenGL designed more than 25 years ago

- Core architecture reflects the origins of 3D graphics
- Extensions retrofitted some GPU features

Challenges with OpenGL

OpenGL designed more than 25 years ago

- Core architecture reflects the origins of 3D graphics
- Extensions retrofitted some GPU features

Challenges with OpenGL

OpenGL designed more than 25 years ago

- Core architecture reflects the origins of 3D graphics
- Extensions retrofitted some GPU features

Fundamental design choices based on past principles

- GPU pipeline has changed
- Multithreaded operation not considered
- Asynchronous processing, not core

Challenges with OpenGL

OpenGL designed more than 25 years ago

- Core architecture reflects the origins of 3D graphics
- Extensions retrofitted some GPU features

Fundamental design choices based on past principles

- GPU pipeline has changed
- Multithreaded operation not considered
- Asynchronous processing, not core

Challenges with OpenGL

OpenGL designed more than 25 years ago

- Core architecture reflects the origins of 3D graphics
- Extensions retrofitted some GPU features

Fundamental design choices based on past principles

- GPU pipeline has changed
- Multithreaded operation not considered
- Asynchronous processing, not core

Design Goals for Metal

Efficient GPU interaction

Design Goals for Metal

Efficient GPU interaction

- Low CPU overhead

Design Goals for Metal

Efficient GPU interaction

- Low CPU overhead
- Multithreaded execution

Design Goals for Metal

Efficient GPU interaction

- Low CPU overhead
- Multithreaded execution
- Predictable operation

Design Goals for Metal

Efficient GPU interaction

- Low CPU overhead
- Multithreaded execution
- Predictable operation
- Resource and synchronization control

Design Goals for Metal

Efficient GPU interaction

- Low CPU overhead
- Multithreaded execution
- Predictable operation
- Resource and synchronization control

Approachable to OpenGL developers

Design Goals for Metal

Efficient GPU interaction

- Low CPU overhead
- Multithreaded execution
- Predictable operation
- Resource and synchronization control

Approachable to OpenGL developers

Built for modern and Apple-design GPUs

Key Conceptual Differences

Expensive operations less frequent

Expensive CPU operations performed less often

Key Conceptual Differences

Expensive operations less frequent

Expensive CPU operations performed less often

More GPU command generation during object creation

Key Conceptual Differences

Expensive operations less frequent

Expensive CPU operations performed less often

More GPU command generation during object creation

Less needed when rendering

Key Conceptual Differences

Modern GPU pipeline

Reflects the modern GPU architectures

Key Conceptual Differences

Modern GPU pipeline

Reflects the modern GPU architectures

Closer match yields less costly translation to GPU commands

Key Conceptual Differences

Modern GPU pipeline

Reflects the modern GPU architectures

Closer match yields less costly translation to GPU commands

State grouped more efficiently

Key Conceptual Differences

Multithreaded execution

Designed for multithreaded execution

Key Conceptual Differences

Multithreaded execution

Designed for multithreaded execution

Clear rules for multithreaded usage

Key Conceptual Differences

Multithreaded execution

Designed for multithreaded execution

Clear rules for multithreaded usage

Cross thread object usability

Key Conceptual Differences

Execution model

True interaction between software and GPU

Key Conceptual Differences

Execution model

True interaction between software and GPU

Predictable operation allows efficient designs

Key Conceptual Differences

Execution model

True interaction between software and GPU

Predictable operation allows efficient designs

Thinner stack between application and GPU

Application Renderer

OpenGL API

OpenGL Context

Application Renderer

Metal API

Device

Application Renderer

Metal API

Device

Application Renderer

Metal API

Device

GPU

Application Renderer

Metal API

Device

GPU

Application Renderer

Metal API

Device

GPU

Application Renderer

Metal API

Textures

Buffers

Pipelines

Render Objects

Device

GPU

Application Renderer

Metal API

Textures

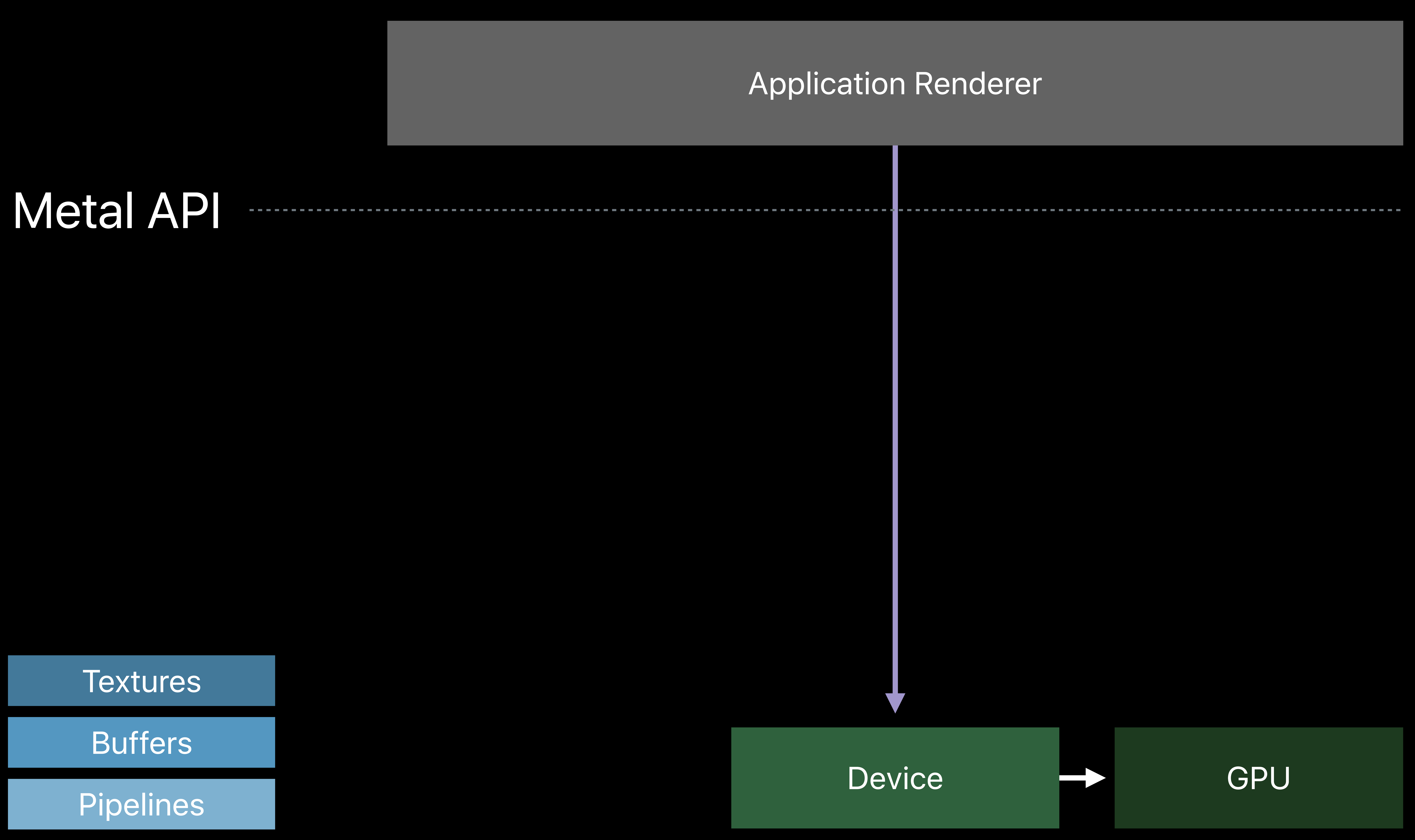
Buffers

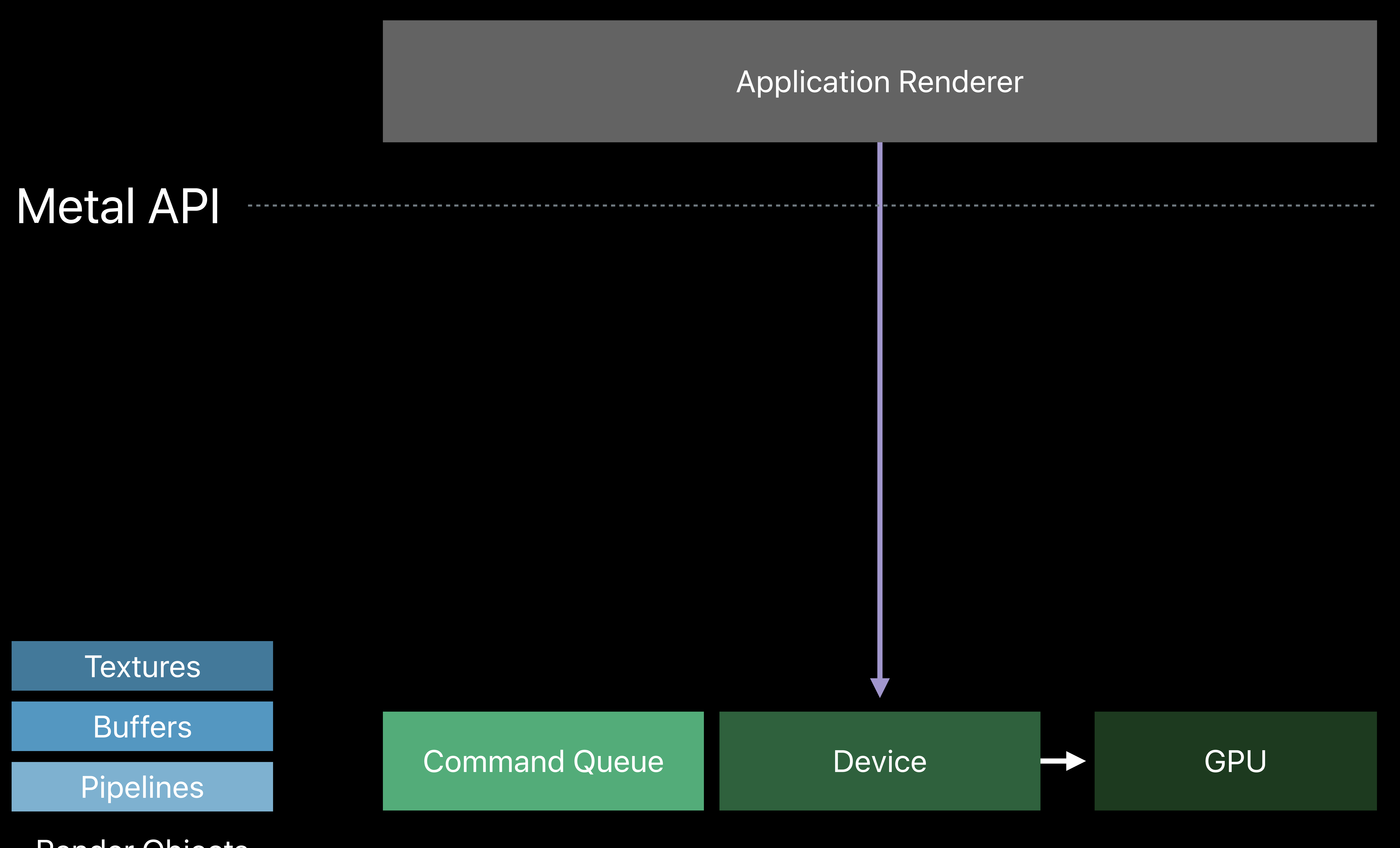
Pipelines

Render Objects

Device

GPU


Application Renderer
Metal API
Textures
Buffers
Pipelines
Render Objects
Device
GPU


Application Renderer
Metal API
Textures
Buffers
Pipelines
Render Objects
Command Queue
Device
GPU

Application Renderer

Metal API

Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Application Renderer

Metal API

Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Application Renderer

Metal API

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Buffers

Command Queue

Device → GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device → GPU

Metal API

Application Renderer

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

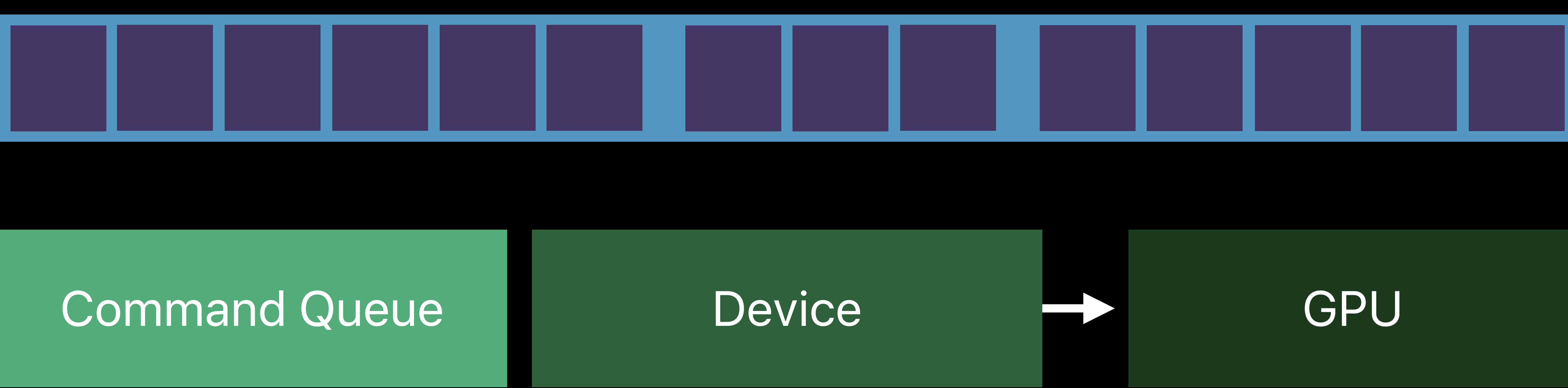
GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

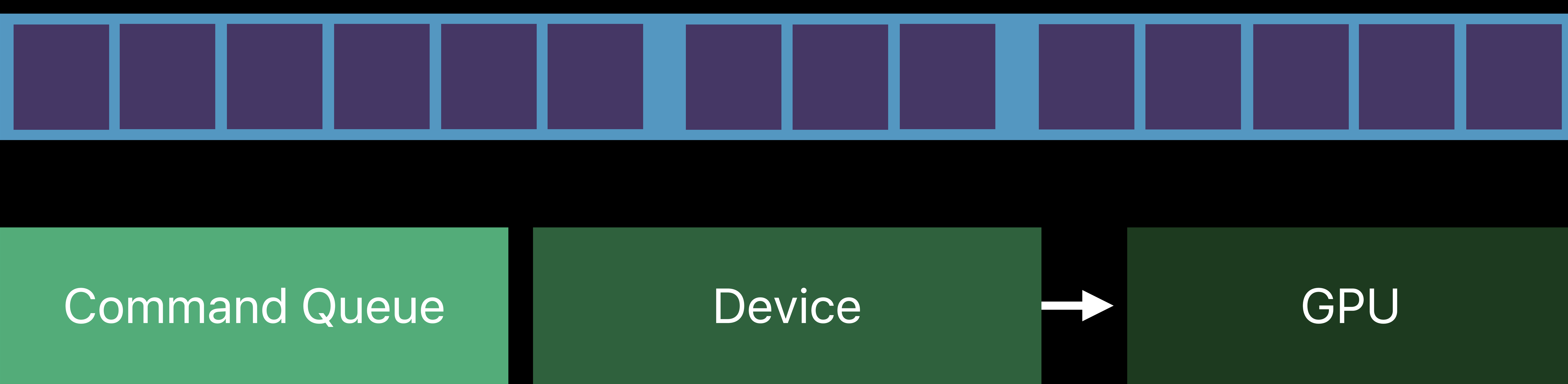


Application Renderer

Metal API

Command Encoders

Command Buffers



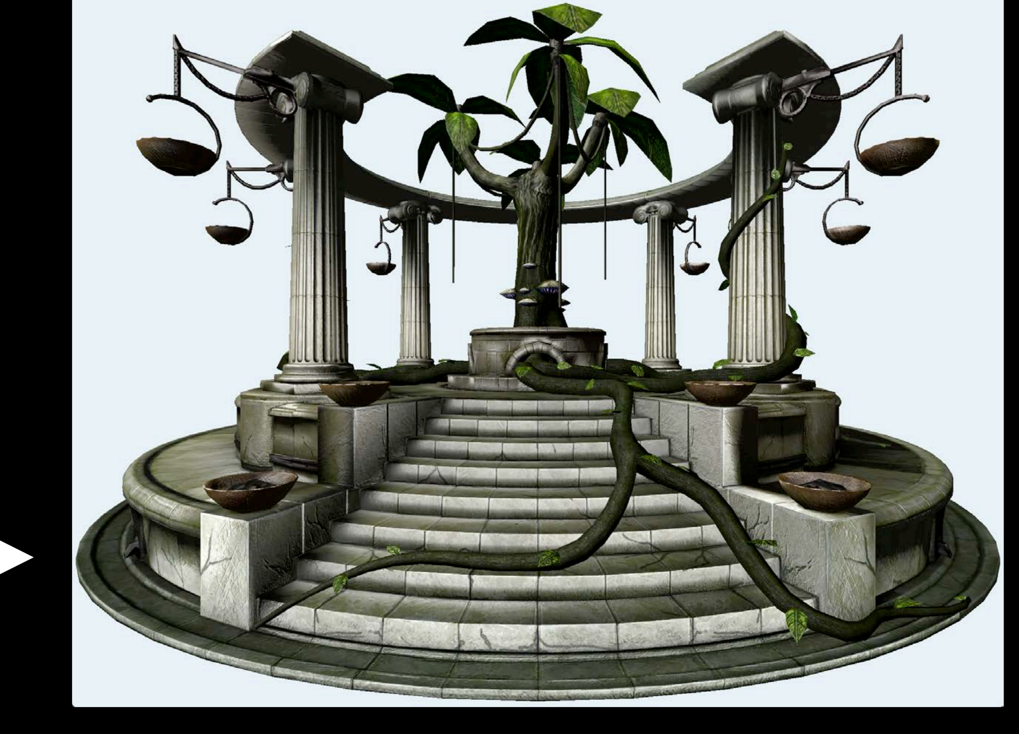
Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue → Device → GPU →



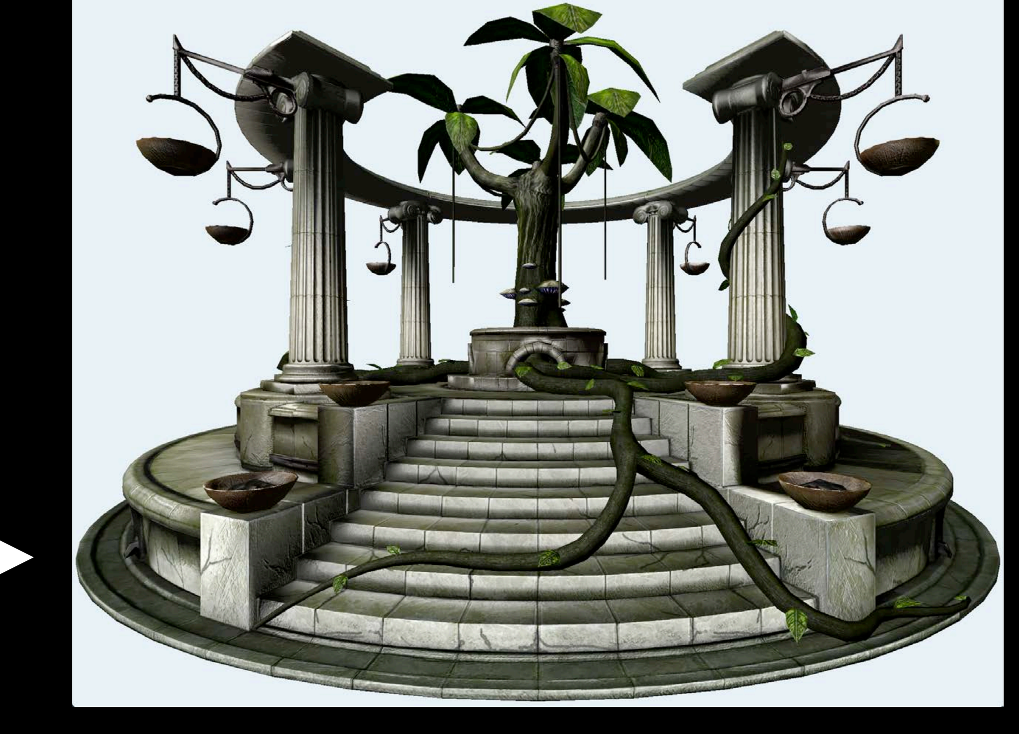
Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue → Device → GPU →



Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue | Device → GPU →

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

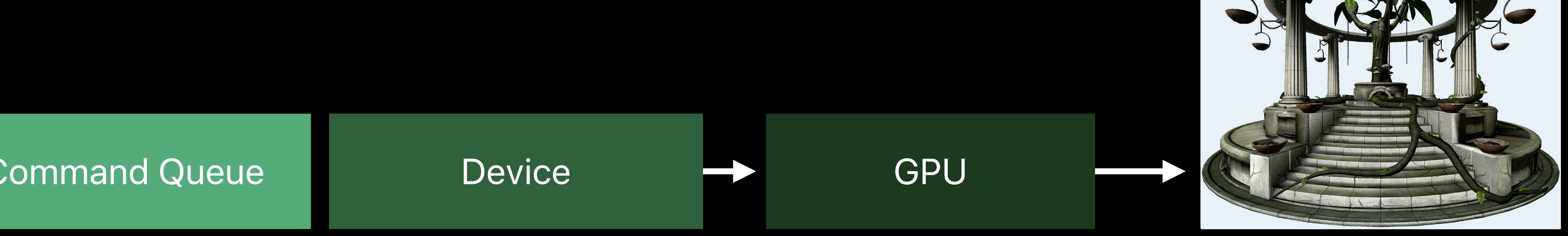
GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

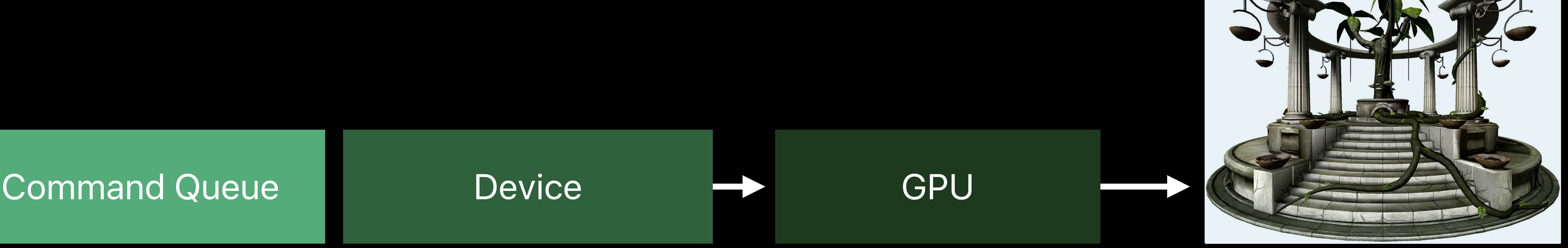


Application Renderer

Metal API

Command Encoders

Command Buffers

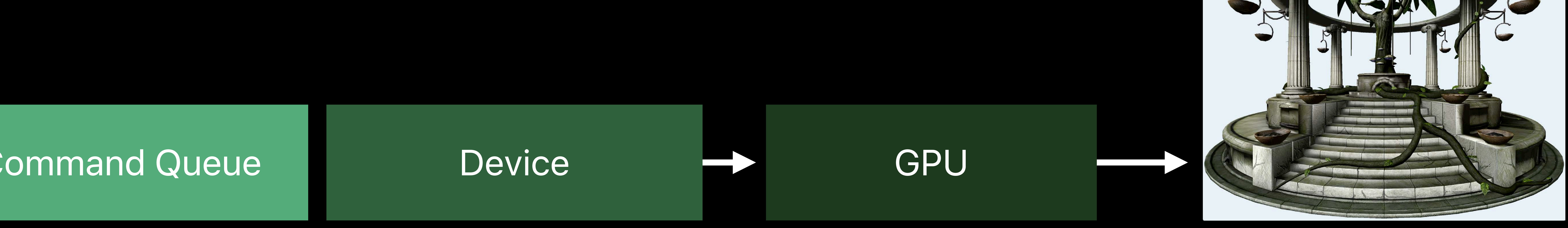


Application Renderer

Metal API

Command Encoders

Command Buffers

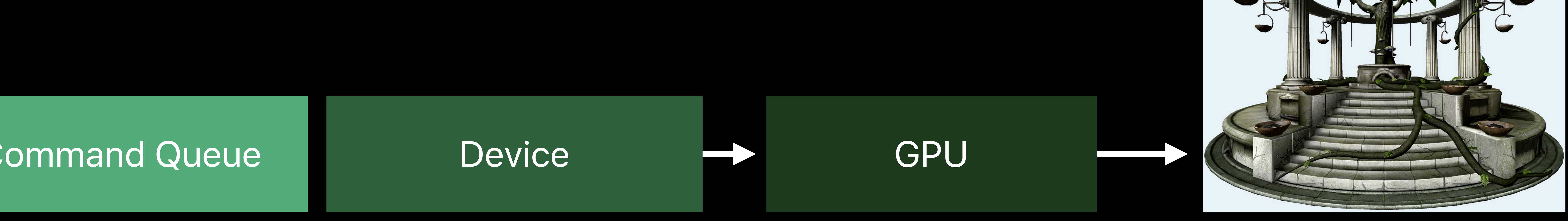


Application Renderer

Metal API

Command Encoders

Command Buffers

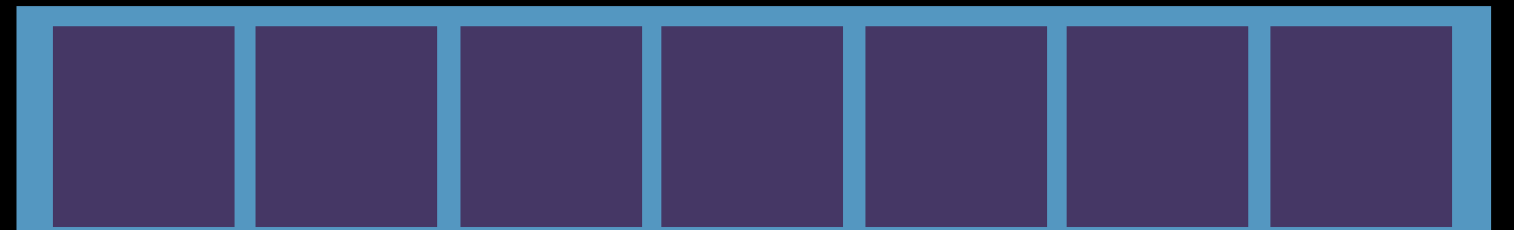


Application Renderer

Metal API

Command Encoders

Command Buffers



Command Queue

Device

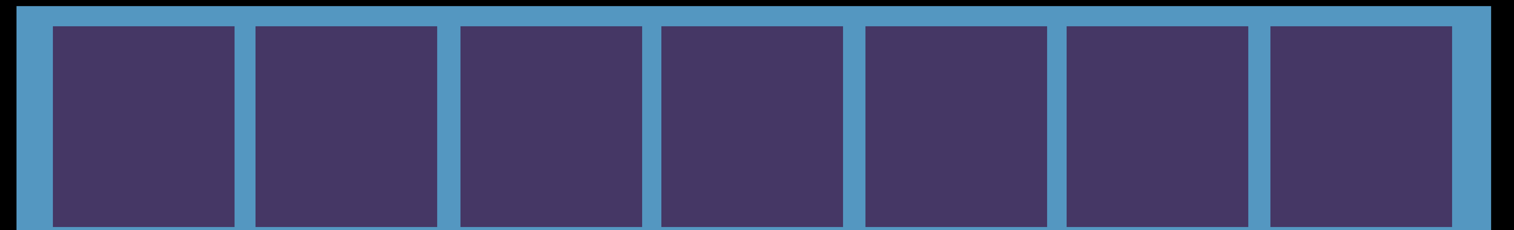
GPU

Application Renderer

Metal API

Command Encoders

Command Buffers



Command Queue

Device

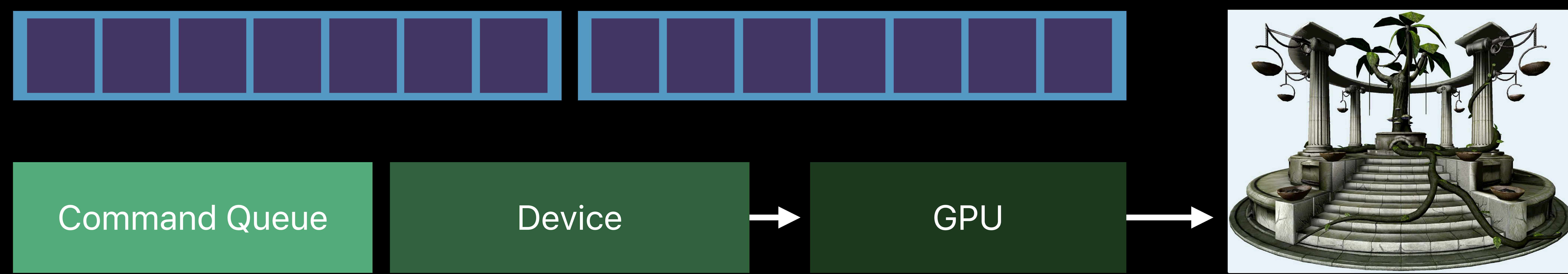
GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

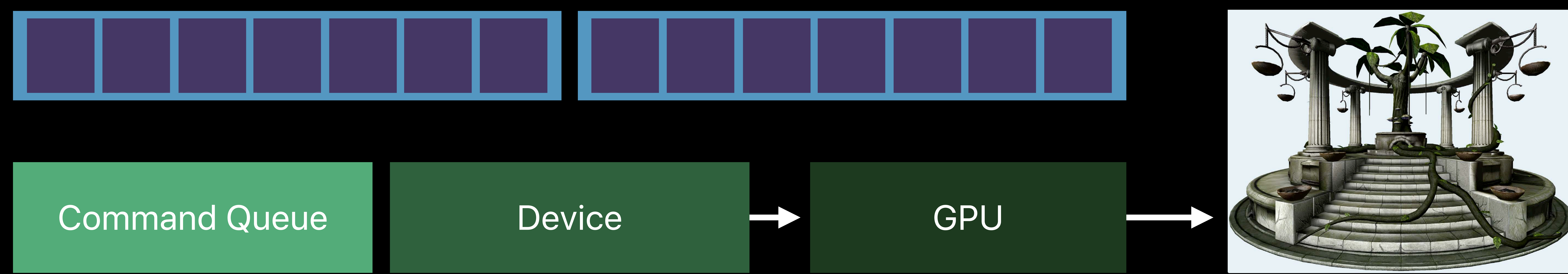


Application Renderer

Metal API

Command Encoders

Command Buffers

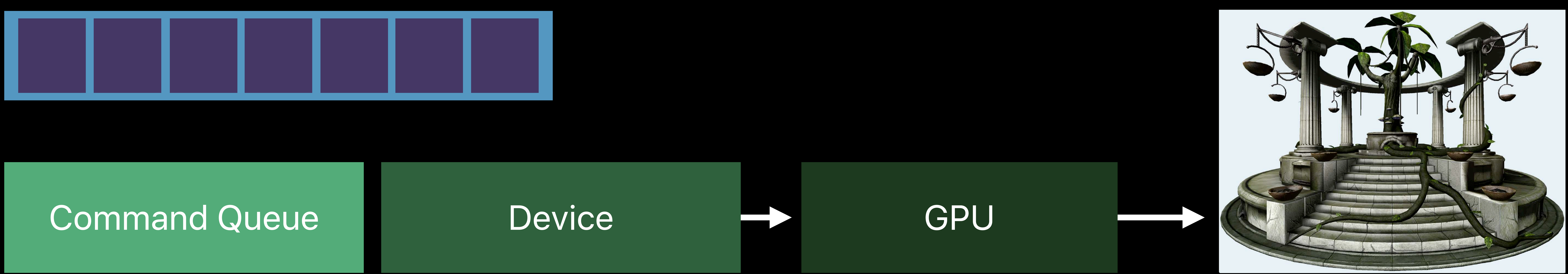


Application Renderer

Metal API

Command Encoders

Command Buffers

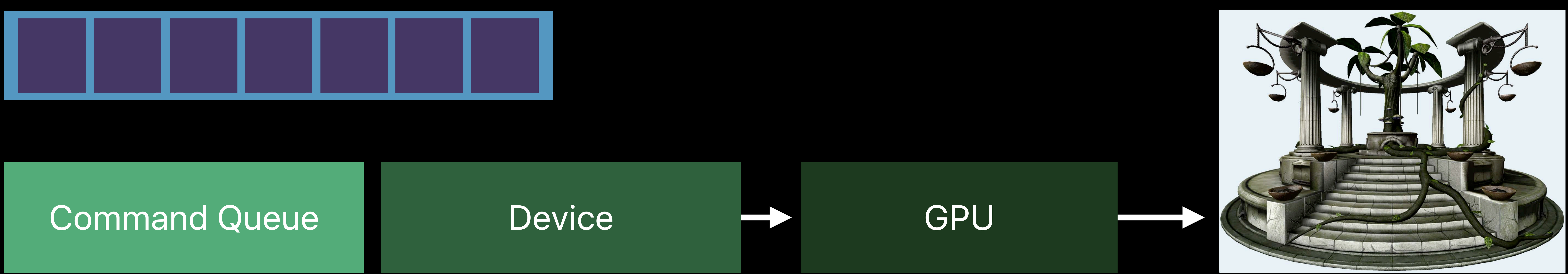


Application Renderer

Metal API

Command Encoders

Command Buffers



Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoders

Command Buffers

Command Queue

Device

GPU

Command Encoders

Render Command Encoder

Command Encoders

Render Command Encoder

Blit Command Encoder

Command Encoders

Render Command Encoder

Blit Command Encoder

Compute Command Encoder

Render Command Encoders

Commands for a render pass

Encodes a series of render commands

Render Command Encoders

Commands for a render pass

Encodes a series of render commands

Also called a Render Pass

Render Command Encoders

Commands for a render pass

Encodes a series of render commands

Also called a Render Pass

Set render object for the graphics pipeline

- Buffer, textures, shaders

Render Command Encoders

Commands for a render pass

Encodes a series of render commands

Also called a Render Pass

Set render object for the graphics pipeline

- Buffer, textures, shaders

Issue draw commands

- Draw primitives, draw index primitives, instanced draws

Render Command Encoders

Render targets

Associated with a set of render targets

Render Command Encoders

Render targets

Associated with a set of render targets

- Textures for rendering

Render Command Encoders

Render targets

Associated with a set of render targets

- Textures for rendering

Specify a set of render targets upon creation

Render Command Encoders

Render targets

Associated with a set of render targets

- Textures for rendering

Specify a set of render targets upon creation

All draw commands directed to these for lifetime of encoder

Render Command Encoders

Render targets

Associated with a set of render targets

- Textures for rendering

Specify a set of render targets upon creation

All draw commands directed to these for lifetime of encoder

New render targets need a new encoder

Render Command Encoders

Render targets

Associated with a set of render targets

- Textures for rendering

Specify a set of render targets upon creation

All draw commands directed to these for lifetime of encoder

New render targets need a new encoder

Clear delineation between sets of render targets

Render Objects

Textures

Render Objects

Textures

Buffers

Render Objects

Textures

Buffers

Samplers

Render Objects

Textures

Buffers

Samplers

Render pipeline states

Render Objects

Textures

Buffers

Samplers

Render pipeline states

Depth stencil states

Render Objects

Object creation

Created from a device

- Usable only on that device

Render Objects

Object creation

Created from a device

- Usable only on that device

Objects state set at creation

- Descriptor object specifies properties for render object

Render Objects

Object creation

Created from a device

- Usable only on that device

Objects state set at creation

- Descriptor object specifies properties for render object

State set at creation fixed for the lifetime of the object

- Image data of textures and values in buffers can change

Render Objects

Object creation

Metal compiles objects into GPU state once

- Never needs to check for changes and recompile

Render Objects

Object creation

Metal compiles objects into GPU state once

- Never needs to check for changes and recompile

Multithreaded usage more efficient

- Metal does not need to protect state from changes on other threads

Metal Porting

Build

Initialize

Render

Build Initialize Render

Build Initialize Render

Build

Initialize

Render

Build

Shaders

Initialize

Devices and Queues

Render Objects

Render

Command Buffers

Resource Updates

Render Encoders

Display

Build	Initialize	Render
Shaders	Devices and Queues	Command Buffers
	Render Objects	Resource Updates
		Render Encoders
		Display

Build	Initialize	Render
Shaders	Devices and Queues	Command Buffers
	Render Objects	Resource Updates
		Render Encoders
		Display

Metal Shading Language

Based on C++

- Classes, templates, structs, enums, namespaces

Metal Shading Language

Based on C++

- Classes, templates, structs, enums, namespaces

Built-in types for vectors and matrices

Metal Shading Language

Based on C++

- Classes, templates, structs, enums, namespaces

Built-in types for vectors and matrices

Built-in functions and operators

Metal Shading Language

Based on C++

- Classes, templates, structs, enums, namespaces

Built-in types for vectors and matrices

Built-in functions and operators

Built-in classes for textures and samplers

```
vertex VertexOutput myVertexShader(uint                vid      [[ vertex_id ]],
                                   device    Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput        in          [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                vid      [[ vertex_id ]],
                                   device    Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                vid      [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput       in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                     vid      [[ vertex_id ]],
                                   device    Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                vid      [[ vertex_id ]],
                                   device    Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                vid       [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput       in         [[ stage_in ]],
                                 constant Uniforms & uniforms  [[ buffer(3) ]],
                                 texture2d<float>    colorTex  [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                   vid        [[ vertex_id ]],
                                   device    Vertex     * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                vid       [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput        in          [[ stage_in ]],
                                 constant Uniforms & uniforms    [[ buffer(3) ]],
                                 texture2d<float>    colorTex    [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                 vid       [[ vertex_id ]],
                                   device    Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput        in          [[ stage_in ]],
                                 constant Uniforms & uniforms    [[ buffer(3) ]],
                                 texture2d<float>    colorTex    [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                vid       [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
struct VertexOutput
{
    float4 clipPos [[position]];
    float2 texCoord;
};

struct Vertex
{
    float4 modelPos;
    float2 texCoord;
};

vertex VertexOutput myVertexShader(uint               vid      [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
```

```
struct VertexOutput
{
    float4 clipPos [[position]];
    float2 texCoord;
};

struct Vertex
{
    float4 modelPos;
    float2 texCoord;
};

vertex VertexOutput myVertexShader(uint               vid      [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
```

```
struct VertexOutput
{
    float4 clipPos [[position]];
    float2 texCoord;
};

struct Vertex
{
    float4 modelPos;
    float2 texCoord;
};

vertex VertexOutput myVertexShader(uint                vid       [[ vertex_id ]],
                                   device   Vertex   * vertices  [[ buffer(0) ]],
                                   constant Uniforms & uniforms  [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
```

```
struct VertexOutput
{
    float4 clipPos [[position]];
    float2 texCoord;
};

struct Vertex
{
    float4 modelPos;
    float2 texCoord;
};

vertex VertexOutput myVertexShader(uint                    vid      [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
```

```
struct VertexOutput
{
    float4 clipPos [[position]];
    float2 texCoord;
};

struct Vertex
{
    float4 modelPos;
    float2 texCoord;
};

vertex VertexOutput myVertexShader(uint                vid      [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
```

```
struct VertexOutput
{
    float4 clipPos [[position]];
    float2 texCoord;
};

struct Vertex
{
    float4 modelPos;
    float2 texCoord;
};

vertex VertexOutput myVertexShader(uint                vid      [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
```

```
struct VertexOutput
{
    float4 clipPos [[position]];
    float2 texCoord;
};

struct Vertex
{
    float4 modelPos;
    float2 texCoord;
};

vertex VertexOutput myVertexShader(uint                 vid      [[ vertex_id ]],
                                   device   Vertex    * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
```

```
struct VertexOutput
{
    float4 clipPos [[position]];
    float2 texCoord;
};

struct Vertex
{
    float4 modelPos;
    float2 texCoord;
};

vertex VertexOutput myVertexShader(uint                vid      [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
```

```
struct VertexOutput
{
    float4 clipPos [[position]];
    float2 texCoord;
};

struct Vertex
{
    float4 modelPos;
    float2 texCoord;
};

vertex VertexOutput myVertexShader(uint                vid      [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
```

```
vertex VertexOutput myVertexShader(uint                vid      [[ vertex_id ]],
                                   device    Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput       in          [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                     vid       [[ vertex_id ]],
                                   device    Vertex    * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
vertex VertexOutput myVertexShader(uint                vid      [[ vertex_id ]],
                                   device   Vertex   * vertices [[ buffer(0) ]],
                                   constant Uniforms & uniforms [[ buffer(1) ]])
{
    VertexOutput out;
    out.clipPos = vertices[vid].modelPos * uniforms.mvp;
    out.texCoord = vertices[vid].texCoord;
    return out;
}

fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
[renderEncoder setFragmentBuffer:myUniformBuffer offset:0 atIndex:3];

[renderEncoder setFragmentTexture:myColorTexture atIndex:0];

[renderEncoder setFragmentSampler:mySampler atIndex:1];
```

```
fragment float4 myFragmentShader(VertexOutput        in          [[ stage_in ]],
                                 constant Uniforms & uniforms    [[ buffer(3) ]],
                                 texture2d<float>    colorTex    [[ texture(0) ]],
                                 sampler             texSampler  [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
[renderEncoder setFragmentBuffer:myUniformBuffer offset:0 atIndex:3];

[renderEncoder setFragmentTexture:myColorTexture atIndex:0];

[renderEncoder setFragmentSampler:mySampler atIndex:1];
```

```
fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
[renderEncoder setFragmentBuffer:myUniformBuffer offset:0 atIndex:3];

[renderEncoder setFragmentTexture:myColorTexture atIndex:0];

[renderEncoder setFragmentSampler:mySampler atIndex:1];
```

```
fragment float4 myFragmentShader(VertexOutput        in          [[ stage_in ]],
                                 constant Uniforms & uniforms    [[ buffer(3) ]],
                                 texture2d<float>    colorTex    [[ texture(0) ]],
                                 sampler             texSampler  [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
[renderEncoder setFragmentBuffer:myUniformBuffer offset:0 atIndex:3];

[renderEncoder setFragmentTexture:myColorTexture atIndex:0];

[renderEncoder setFragmentSampler:mySampler atIndex:1];
```

```
fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
[renderEncoder setFragmentBuffer:myUniformBuffer offset:0 atIndex:3];

[renderEncoder setFragmentTexture:myColorTexture atIndex:0];

[renderEncoder setFragmentSampler:mySampler atIndex:1];
```

```
fragment float4 myFragmentShader(VertexOutput        in          [[ stage_in ]],
                                 constant Uniforms & uniforms    [[ buffer(3) ]],
                                 texture2d<float>    colorTex    [[ texture(0) ]],
                                 sampler             texSampler  [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
[renderEncoder setFragmentBuffer:myUniformBuffer offset:0 atIndex:3];

[renderEncoder setFragmentTexture:myColorTexture atIndex:0];

[renderEncoder setFragmentSampler:mySampler atIndex:1];
```

```
fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
[renderEncoder setFragmentBuffer:myUniformBuffer offset:0 atIndex:3];

[renderEncoder setFragmentTexture:myColorTexture atIndex:0];

[renderEncoder setFragmentSampler:mySampler atIndex:1];
```

```
fragment float4 myFragmentShader(VertexOutput        in          [[ stage_in ]],
                                 constant Uniforms & uniforms    [[ buffer(3) ]],
                                 texture2d<float>    colorTex    [[ texture(0) ]],
                                 sampler             texSampler  [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

```
[renderEncoder setFragmentBuffer:myUniformBuffer offset:0 atIndex:3];

[renderEncoder setFragmentTexture:myColorTexture atIndex:0];

[renderEncoder setFragmentSampler:mySampler atIndex:1];
```

```
fragment float4 myFragmentShader(VertexOutput        in         [[ stage_in ]],
                                 constant Uniforms & uniforms   [[ buffer(3) ]],
                                 texture2d<float>    colorTex   [[ texture(0) ]],
                                 sampler             texSampler [[ sampler(1) ]])
{
    return colorTex.sample(texSampler, in.texCoord * uniforms.coordScale);
}
```

SIMD Type Library

Types for shader development

Vector and matrix types

SIMD Type Library

Types for shader development

Vector and matrix types

Usable with Metal shading language and application code

SIMD Type Library

Types for shader development

Vector and matrix types

Usable with Metal shading language and application code

```
struct MyUniforms
{
    matrix_float4x4 modelViewProjection;
    vector_float4   sunPosition;
};
```

SIMD Type Library

Types for shader development

Vector and matrix types

Usable with Metal shading language and application code

MyShaderTypes.h

```
struct MyUniforms
{
    matrix_float4x4 modelViewProjection;
    vector_float4   sunPosition;
};
```

SIMD Type Library

Types for shader development

MyShaderTypes.h

```
struct MyUniforms
{
    matrix_float4x4 modelViewProjection;
    vector_float4   sunPosition;
};
```

MyRenderer.m

```
#include "MyShaderTypes.h"
```

MyShaders.metal

```
#include "MyShaderTypes.h"
```

Shader Compilation

Building with Xcode

Xcode compiles shaders into a Metal library (.metallib)

Shader Compilation

Building with Xcode

Xcode compiles shaders into a Metal library (.metallib)

Front-end compilation to binary intermediate representation

Shader Compilation

Building with Xcode

Xcode compiles shaders into a Metal library (.metallib)

Front-end compilation to binary intermediate representation

Avoids parsing time on customer systems

Shader Compilation

Building with Xcode

Xcode compiles shaders into a Metal library (.metallib)

Front-end compilation to binary intermediate representation

Avoids parsing time on customer systems

By default, all shaders built into default.metallib

- Placed in app bundle for run time retrieval

Runtime Shader Compilation

Also can build shaders from source at runtime

Runtime Shader Compilation

Also can build shaders from source at runtime

Significant disadvantages

- Full shader compilation occurs at runtime
- Compilation errors less obvious
- No header sharing between application and runtime built shaders

Runtime Shader Compilation

Also can build shaders from source at runtime

Significant disadvantages

- Full shader compilation occurs at runtime
- Compilation errors less obvious
- No header sharing between application and runtime built shaders

Build time compilation recommended

Build	Initialize	Render
Shaders	Devices and Queues	Command Buffers
	Render Objects	Resource Updates
		Render Encoders
		Display

Build

Shaders

Initialize

Devices and Queues

Render Objects

Render

Command Buffers

Resource Updates

Render Encoders

Display

Application Renderer

Metal API

Command Encoder

Command Buffer

Display

Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Application Renderer

Metal API

Command Encoder

Command Buffer

Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Devices

A device represents one GPU

Devices

A device represents one GPU

Creates render objects

- Textures, buffers, pipelines

Devices

A device represents one GPU

Creates render objects

- Textures, buffers, pipelines

macOS multiple devices may be avaliable

Devices

A device represents one GPU

Creates render objects

- Textures, buffers, pipelines

macOS multiple devices may be avaliable

Default device suitable for most applications

```
// Getting a device
id<MTLDevice> device = MTLCreateSystemDefaultDevice();
```

Command Queues

Queue created from a device

Command Queues

Queue created from a device

Queues execute command buffers in order

- Create queue at initialization

Command Queues

Queue created from a device

Queues execute command buffers in order

- Create queue at initialization

Typically one queue sufficient

Command Queues

Queue created from a device

Queues execute command buffers in order

- Create queue at initialization

Typically one queue sufficient

```
// Getting a device
id<MTLCommandQueue> commandQueue = [device newCommandQueue];
```

Build

Shaders

Initialize

Devices and Queues

Render Objects

Render

Command Buffers

Resource Updates

Render Encoders

Display

Build

Shaders

Initialize

Devices and Queues

Render Objects

Render

Command Buffers

Resource Updates

Render Encoders

Display

Application Renderer

Metal API

Command Encoder

Command Buffer

Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Display

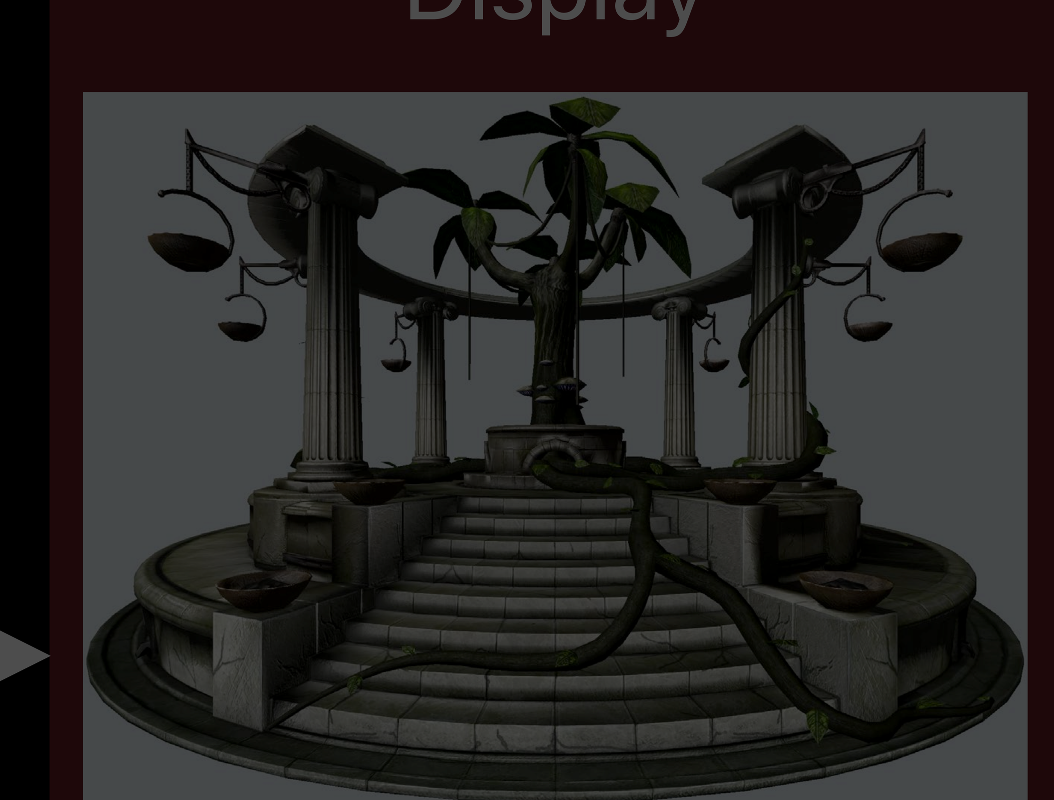
Application Renderer

Metal API

Command Encoder

Command Buffer

Display



Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Textures

Buffers

Pipelines

Textures

Buffers

Pipelines

Device

Descriptor

Device

Texture descriptor

Device

Texture descriptor

Texture Type	2D
Width	512
Height	512
Num Mipmaps	1
Pixel Format	RGBA8

Device

Texture descriptor

Texture Type	2D
Width	512
Height	512
Num Mipmaps	1
Pixel Format	RGBA8

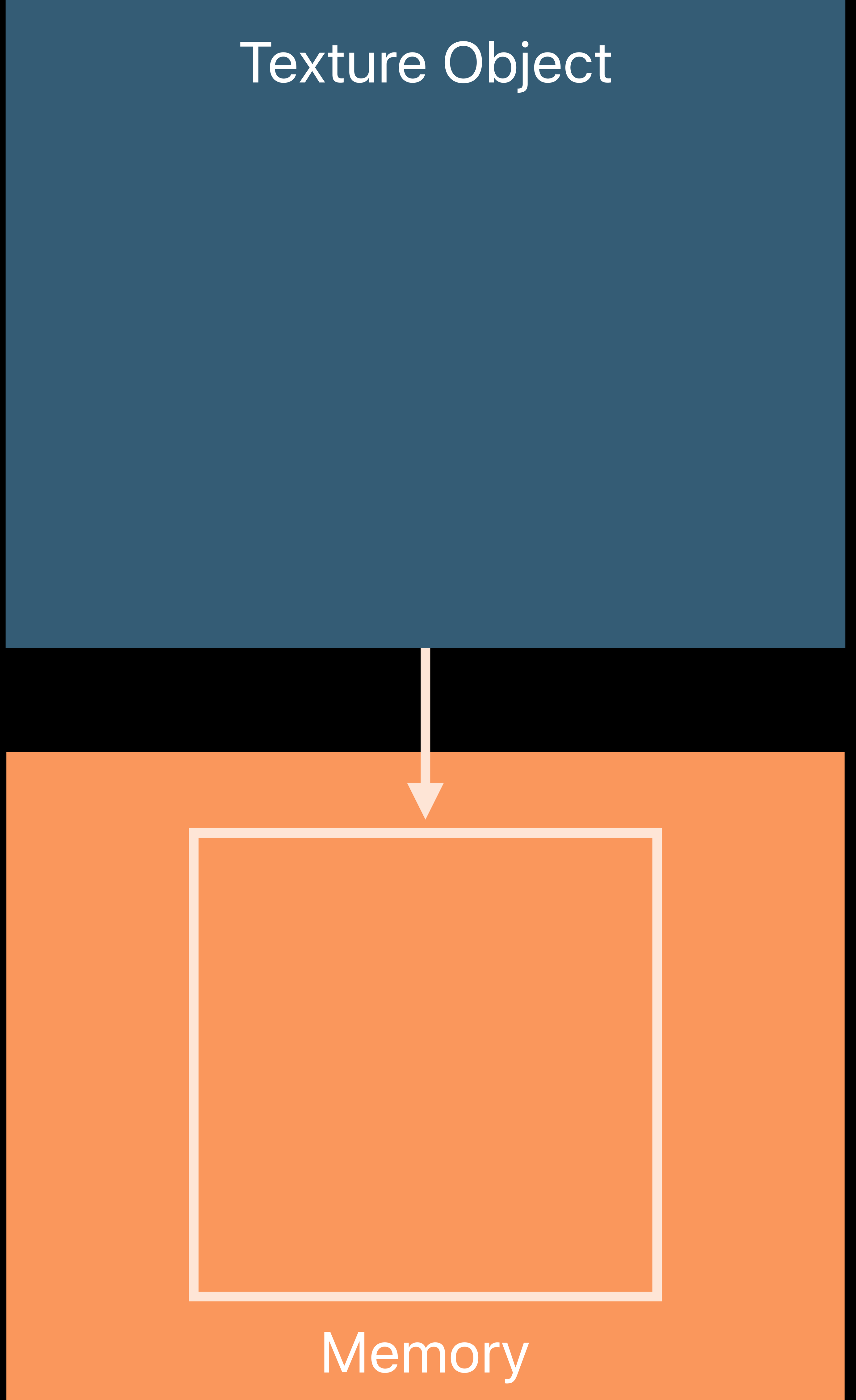
Device

Texture descriptor
Texture Type 2D
Width 512
Height 512
Num Mipmaps 1
Pixel Format RGBA8
Device
Texture Object

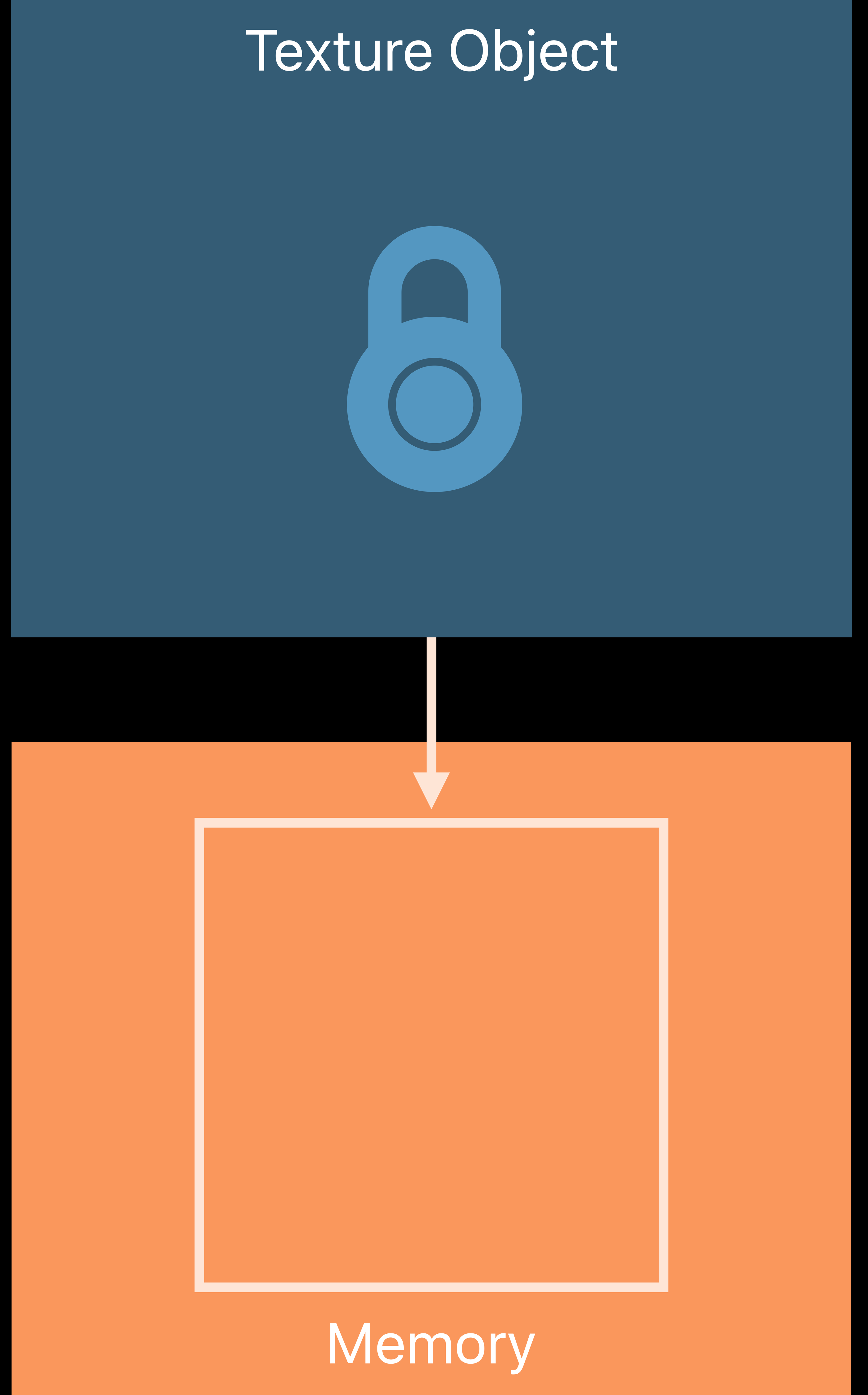
Texture descriptor
Texture Type 2D
Width 512
Height 512
Num Mipmaps 1
Pixel Format RGBA8
Device
Texture Object
Memory

Texture descriptor
Texture Type 2D
Width 512
Height 512
Num Mipmaps 1
Pixel Format RGBA8
Device
Texture Object
Memory

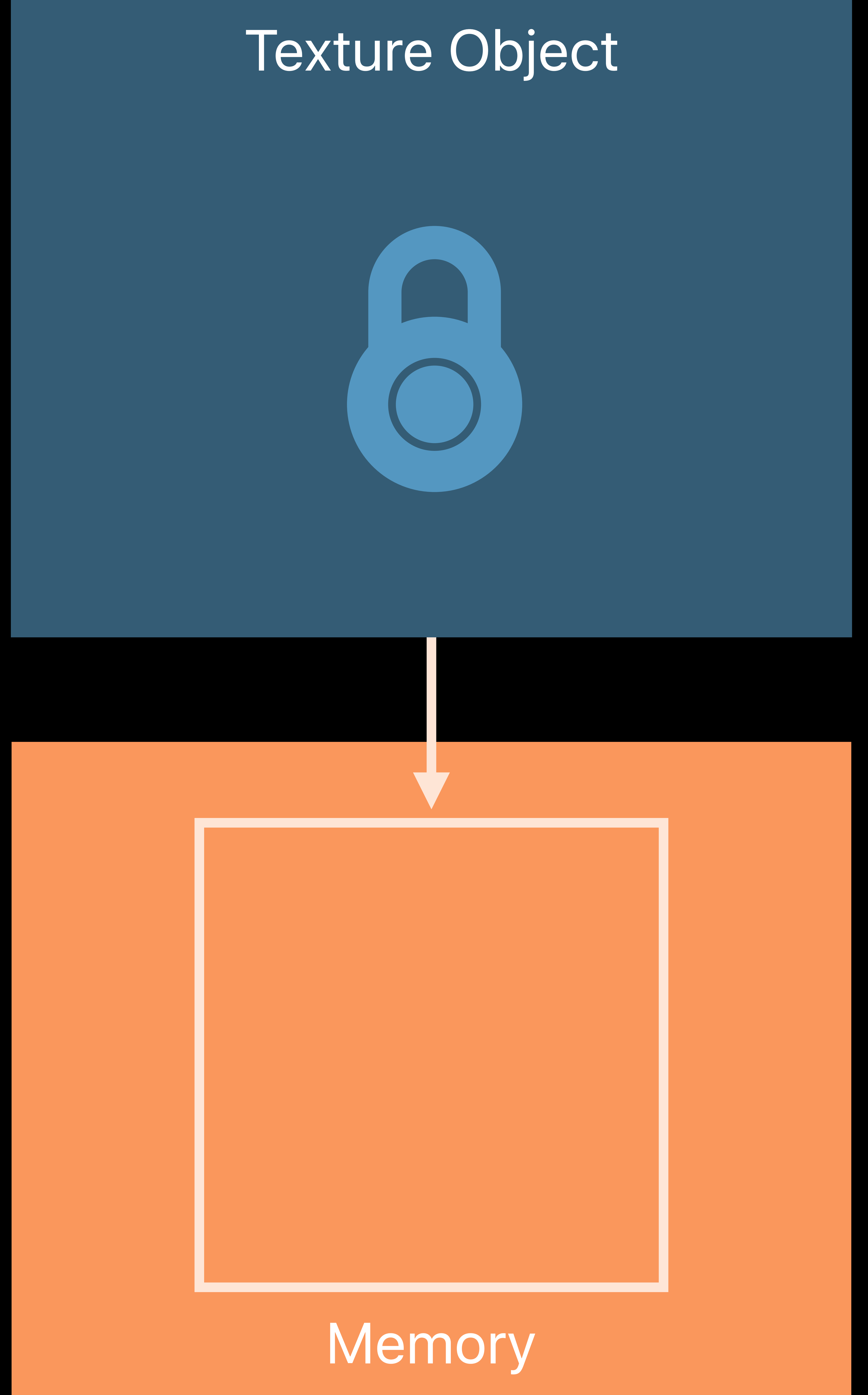
Device



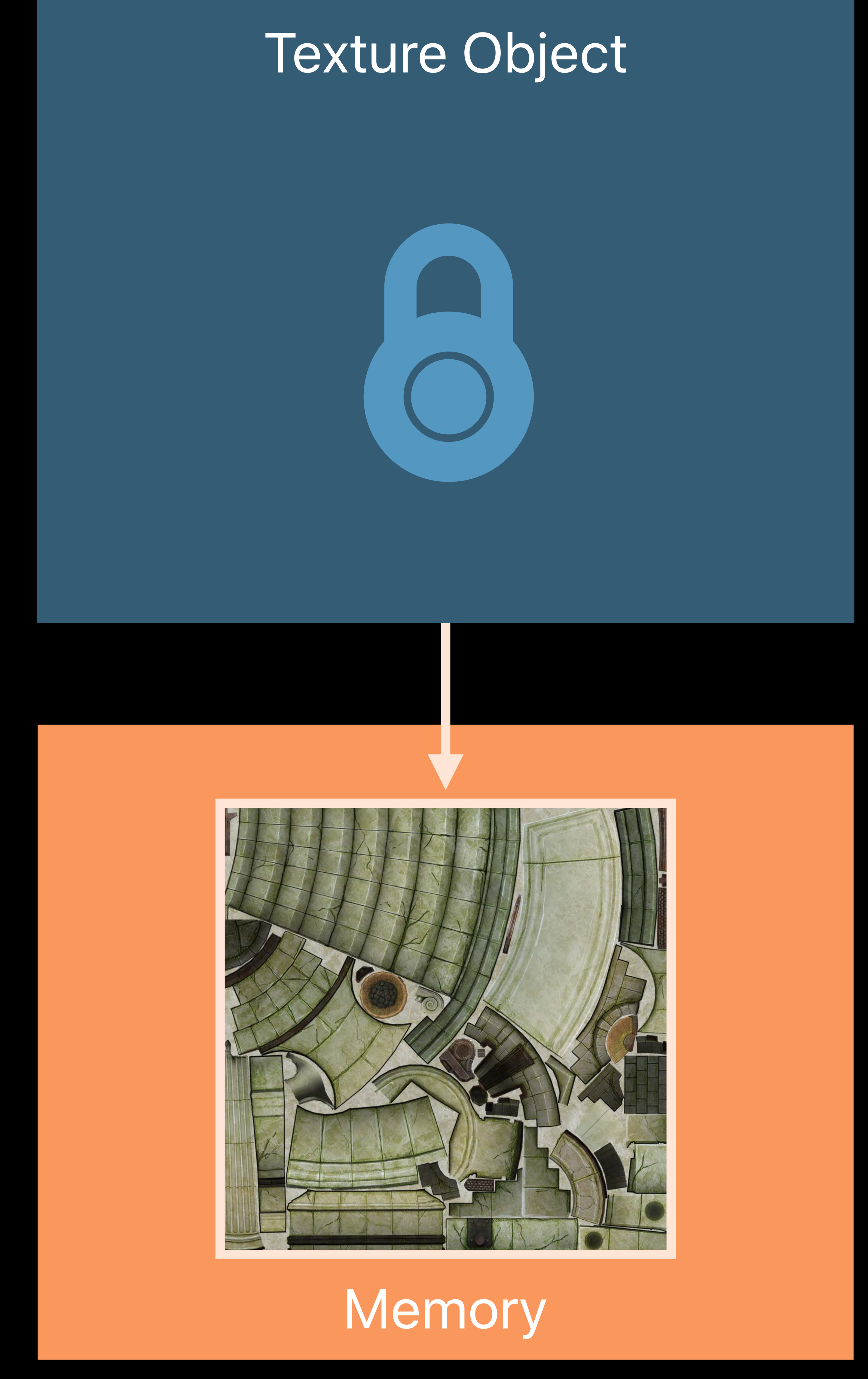
Device



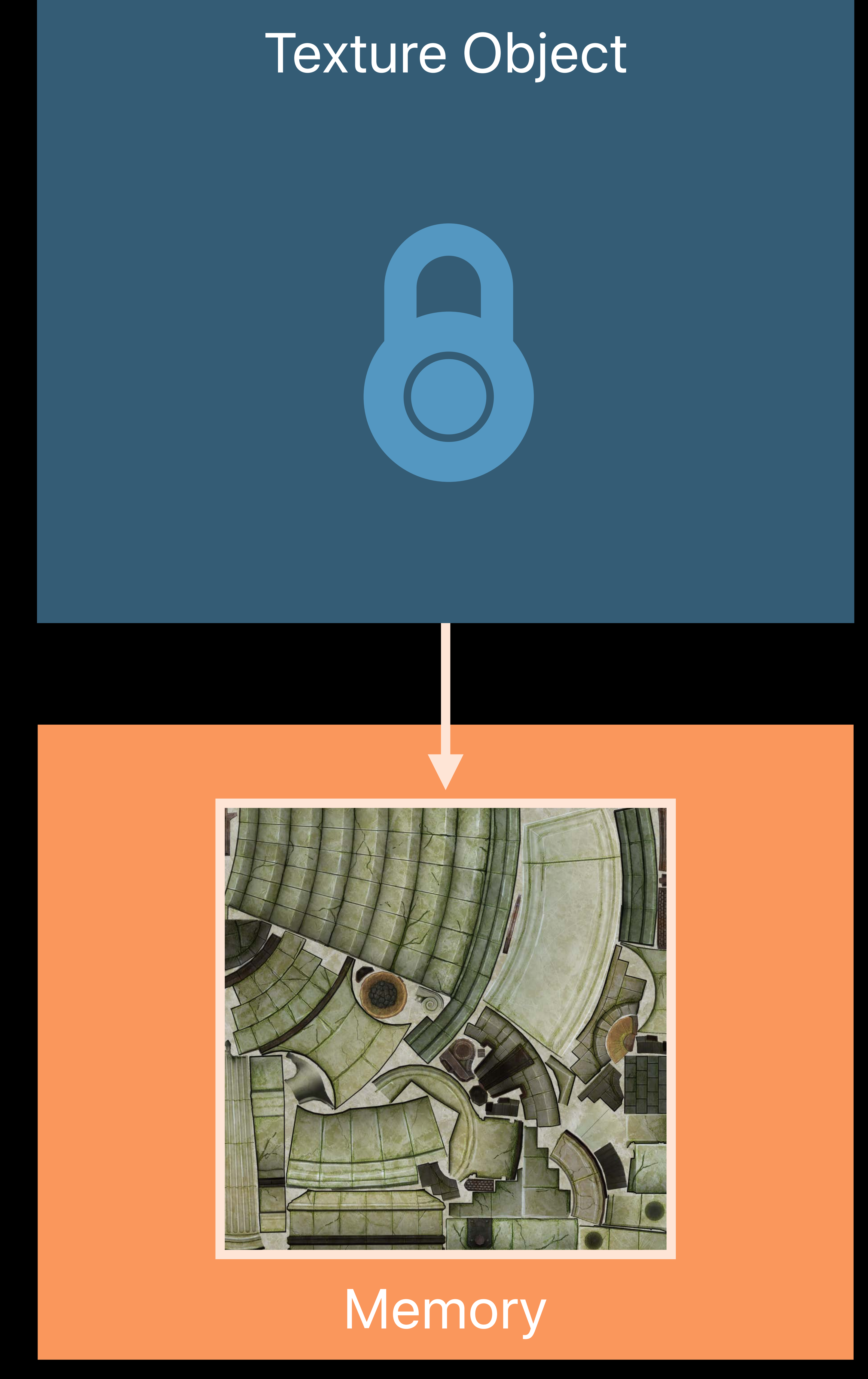
Device



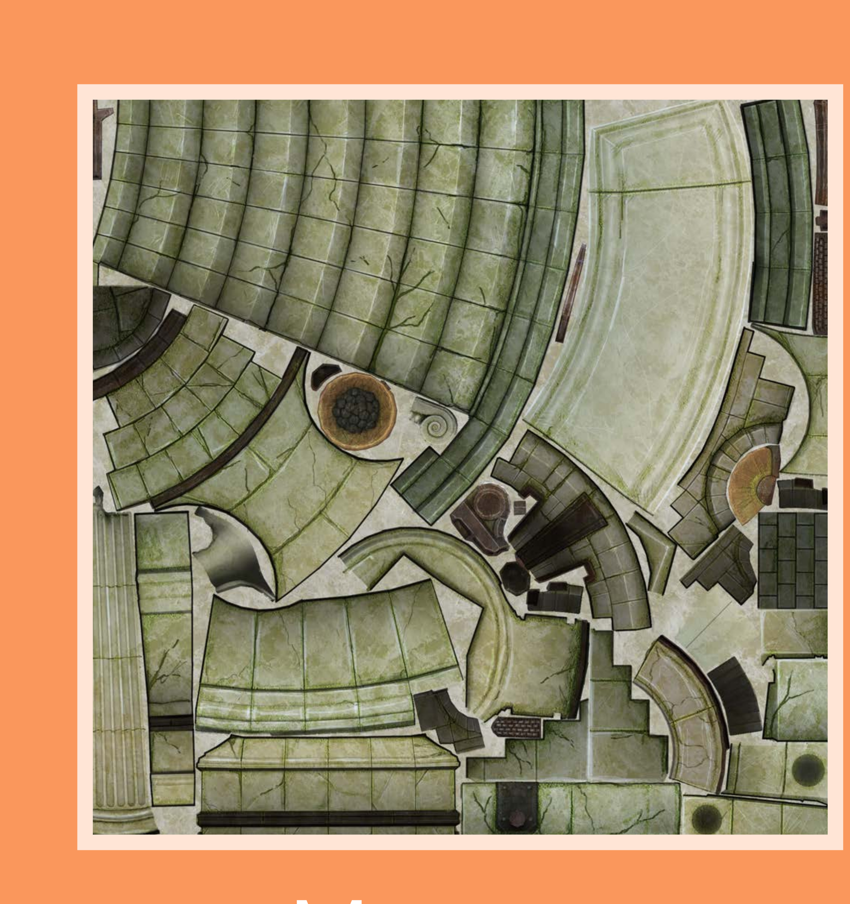
Device



Device



Storage Modes



Memory

Storage Modes

Shared storage

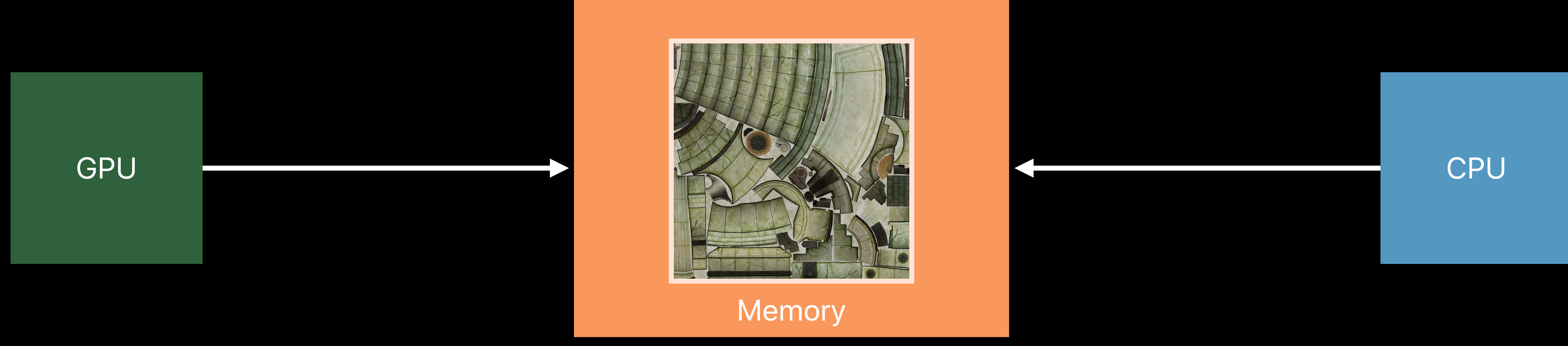
GPU

Memory

CPU

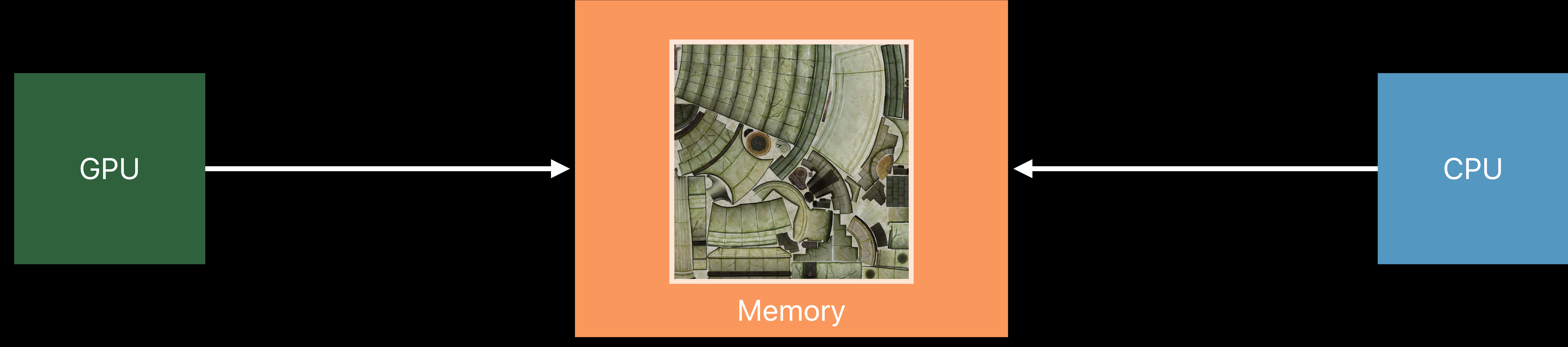
Storage Modes

Shared storage



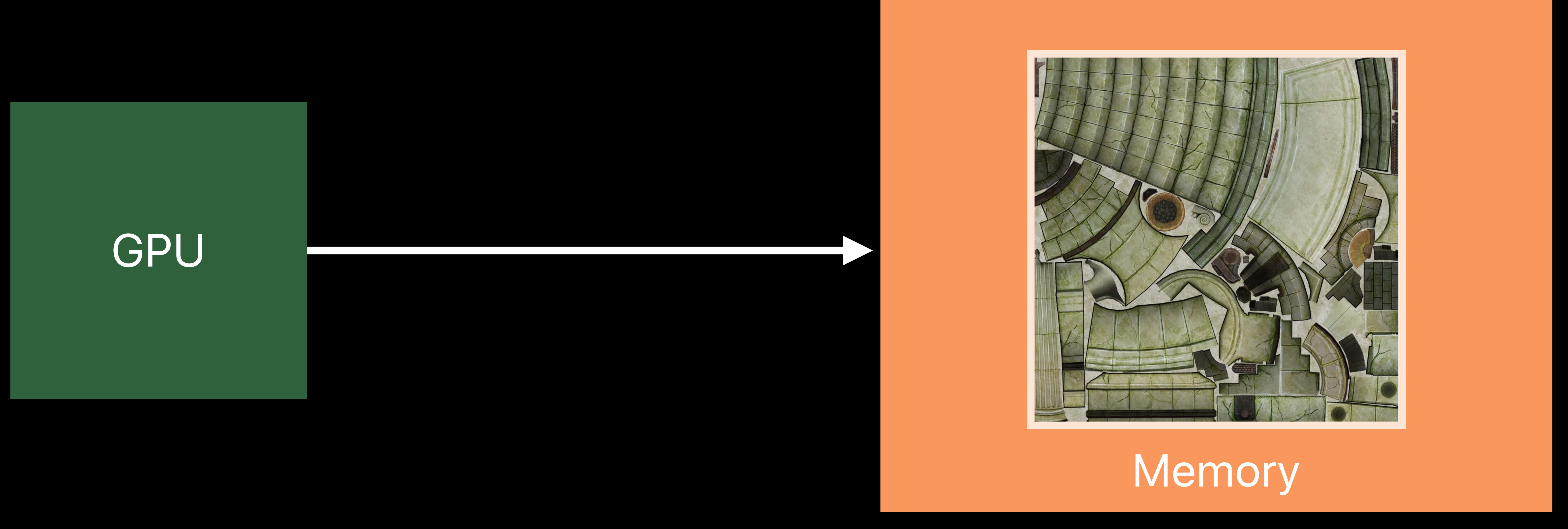
Storage Modes

Private storage



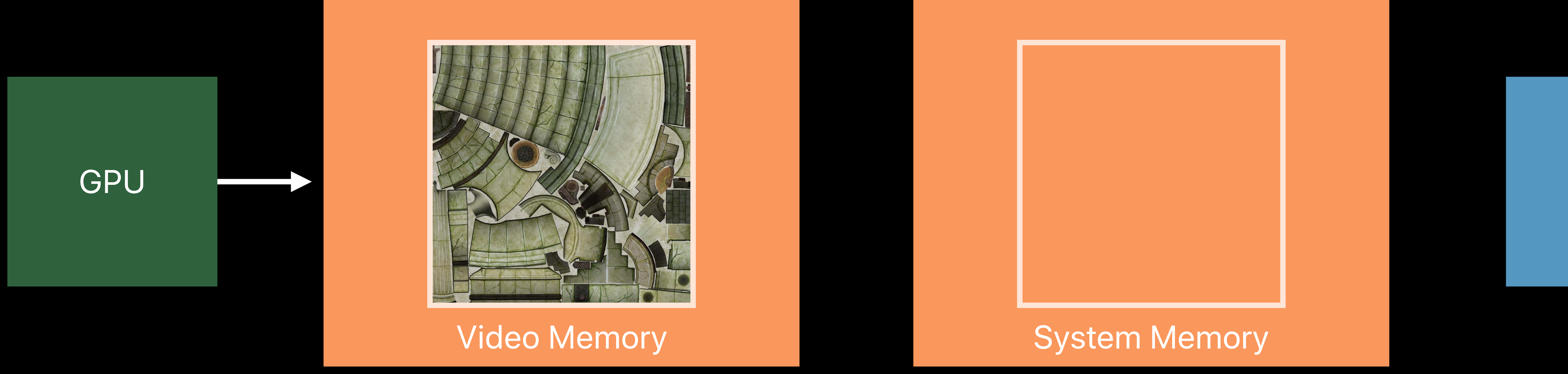
Storage Modes

Private storage



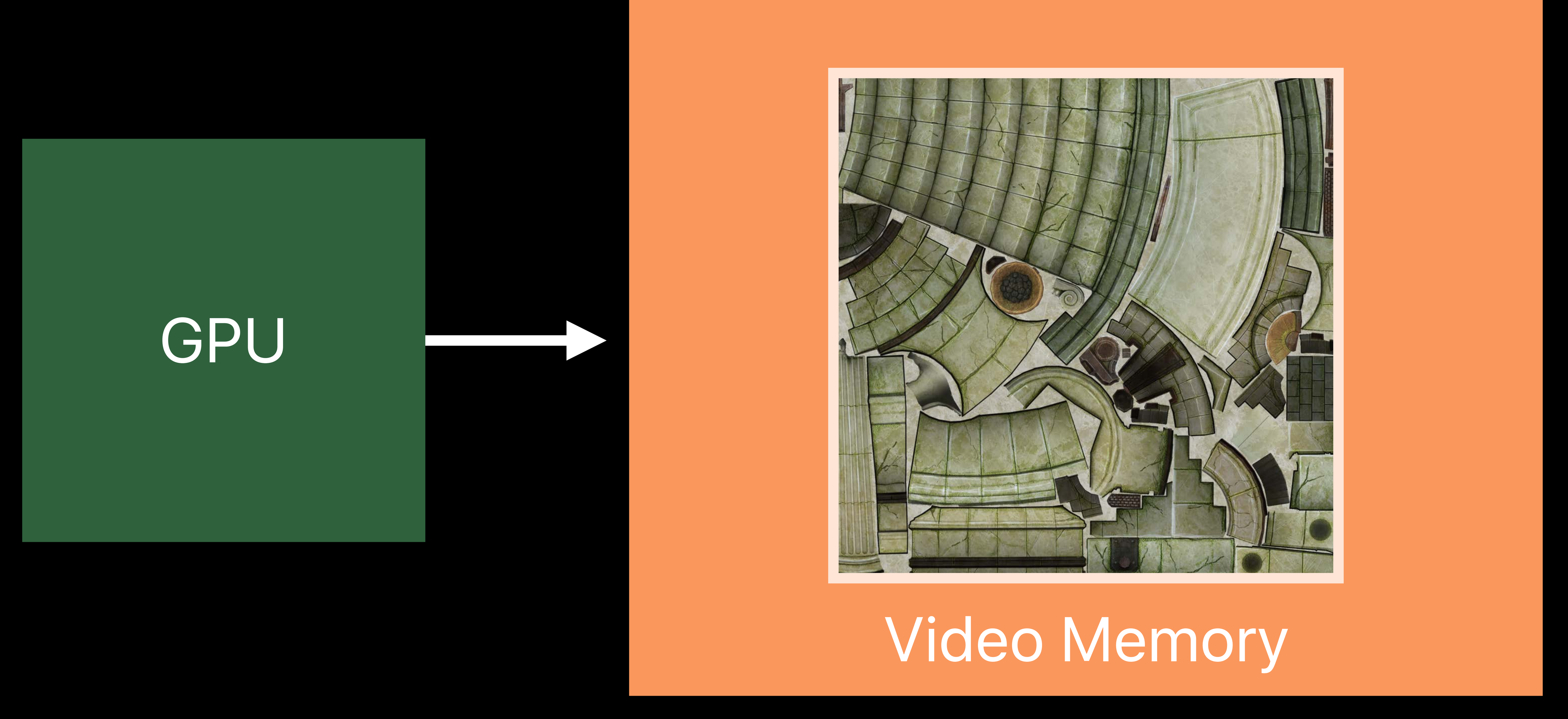
Storage Modes

Private storage



Storage Modes

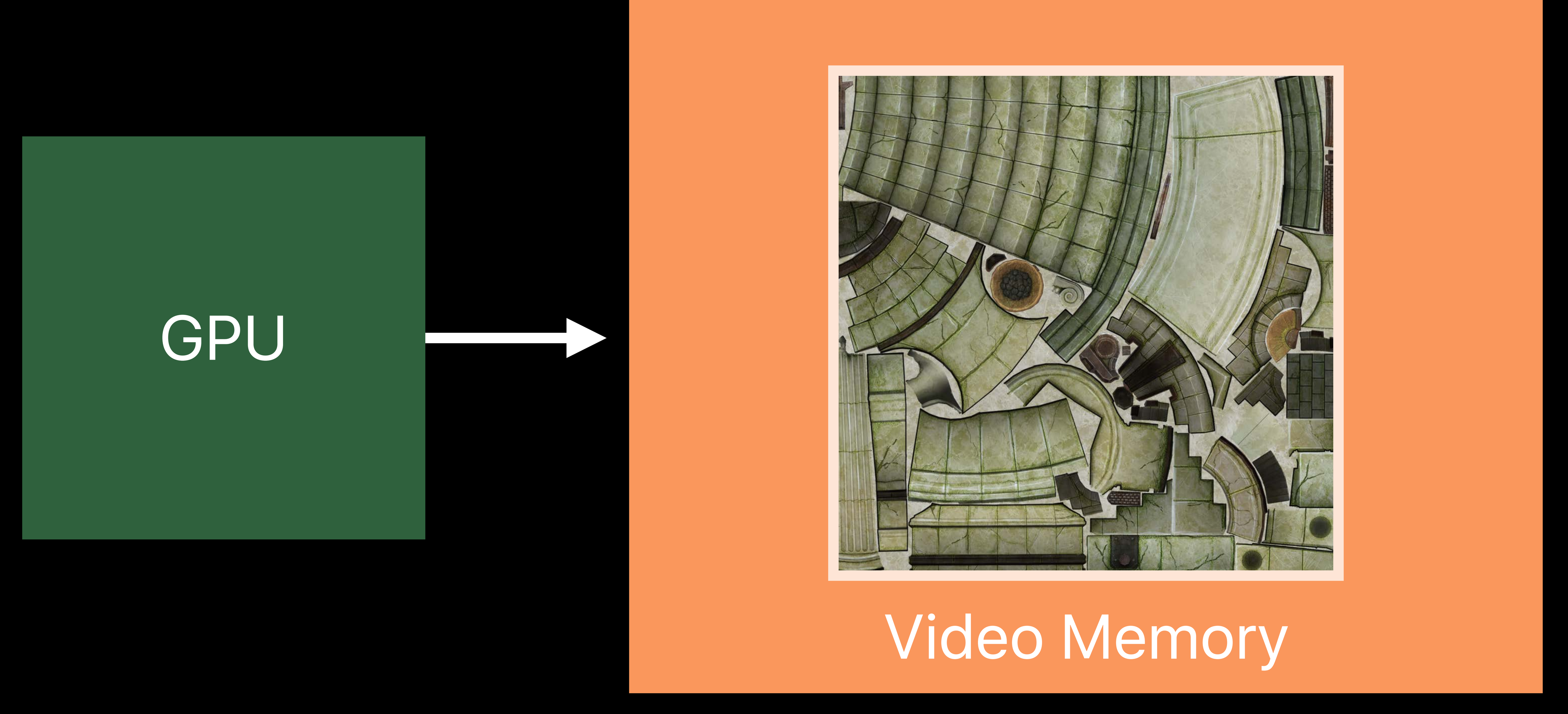
Managed storage

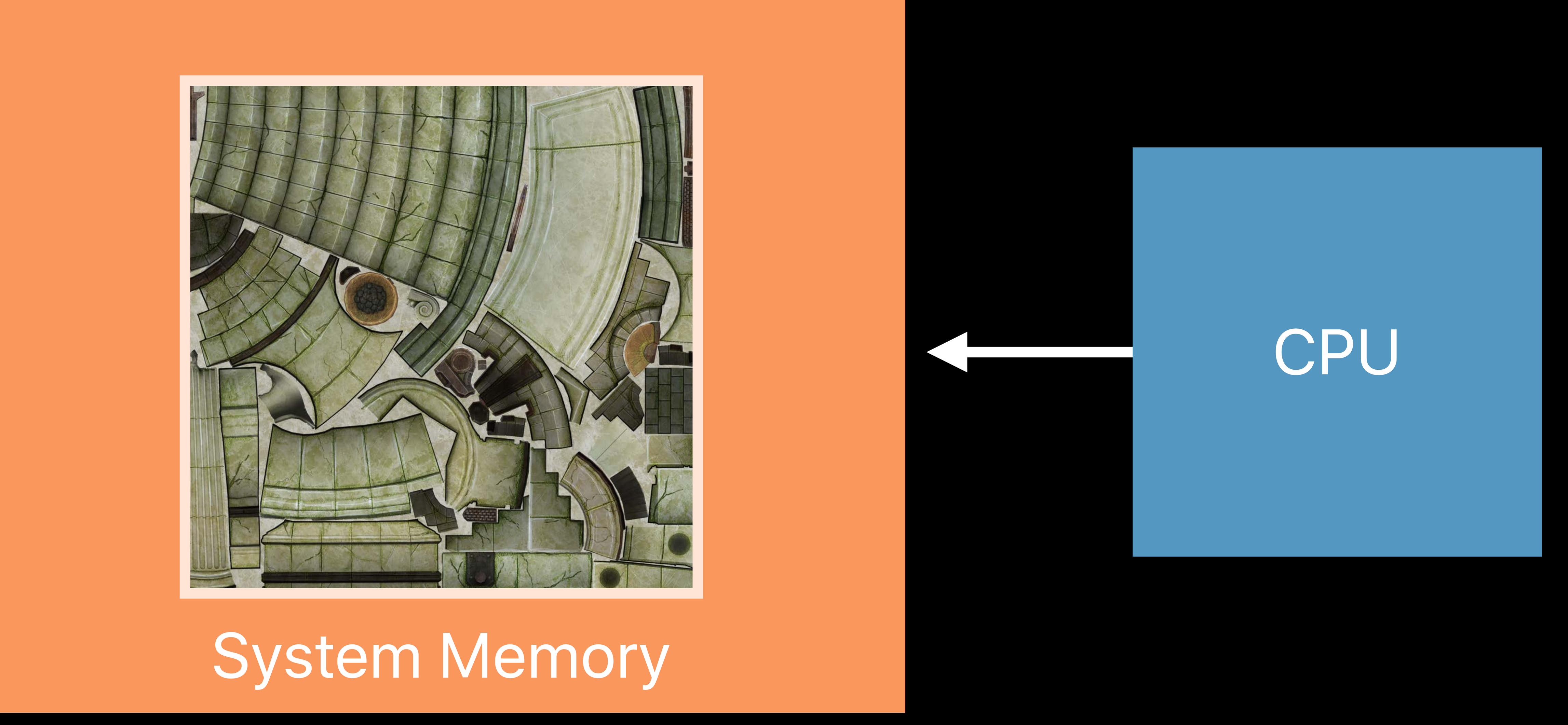


CPU

Storage Modes

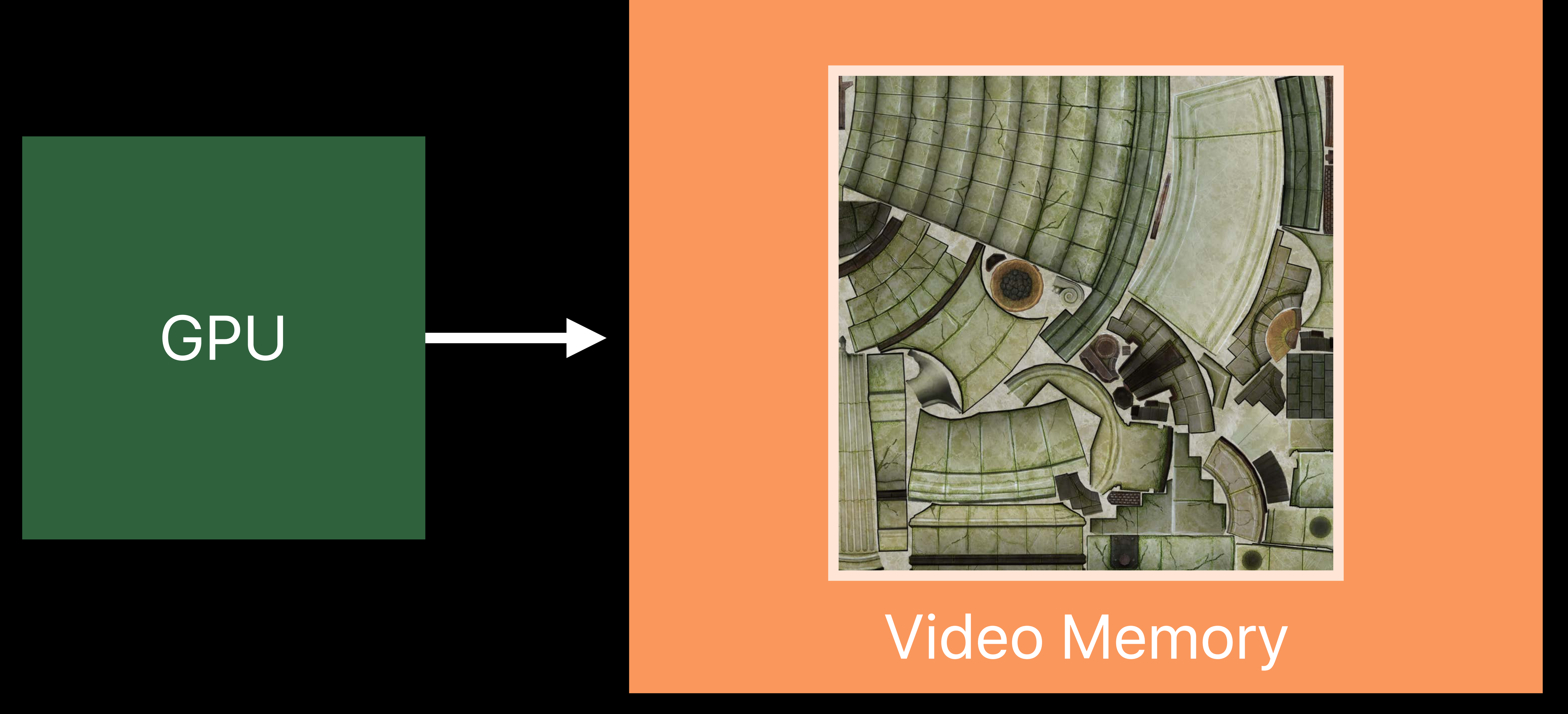
Managed storage

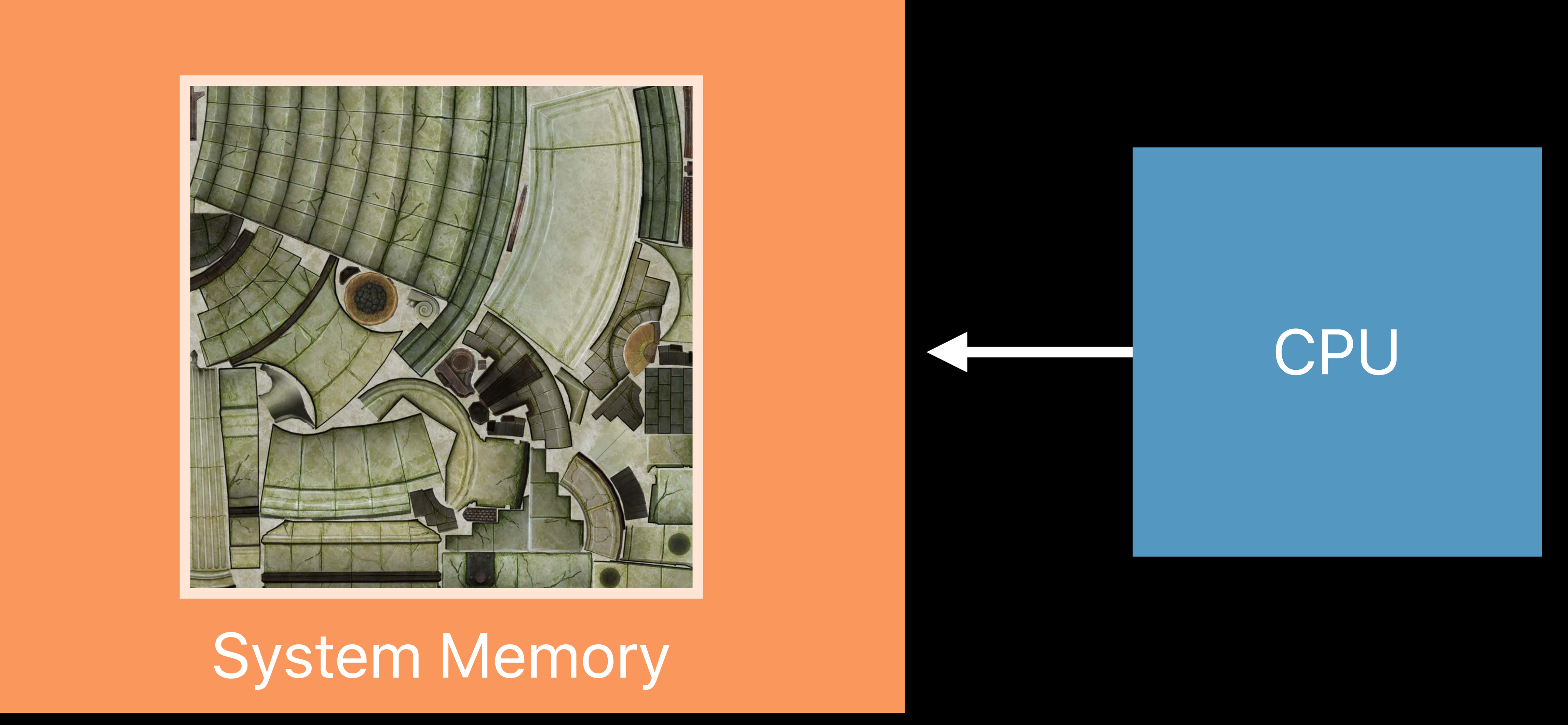




Storage Modes

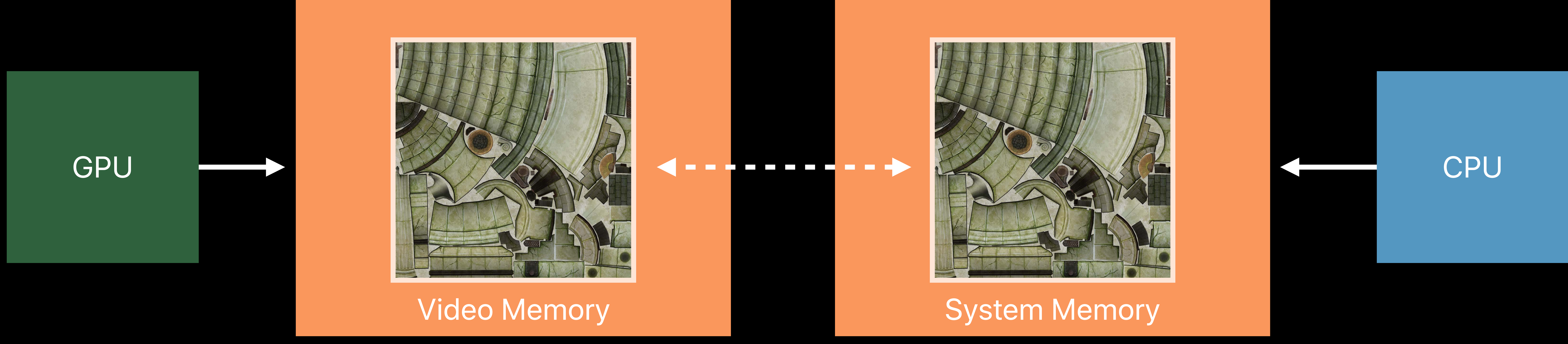
Managed storage





Storage Modes

Managed storage



```
// Creating Textures

MTLTextureDescriptor *textureDescriptor = [MTLTextureDescriptor new];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = 512;
textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;

id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];
```

```
// Creating Textures

MTLTextureDescriptor *textureDescriptor = [MTLTextureDescriptor new];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = 512;
textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;;

id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];
```

```
// Creating Textures

MTLTextureDescriptor *textureDescriptor = [MTLTextureDescriptor new];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = 512;
textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;

id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];
```

```
// Loading Image Data

NSUInteger bytesPerRow = 4 * image.width;

MTLRegion region =
{
        { 0, 0, 0 },    // Origin
        { 512, 512, 1 } // Size
};

[texture replaceRegion:region
           mipmapLevel:0
             withBytes:imageData
           bytesPerRow:bytesPerRow];
```

```
// Loading Image Data

NSUInteger bytesPerRow = 4 * image.width;

MTLRegion region =
{
        { 0, 0, 0 },    // Origin
        { 512, 512, 1 } // Size
};

[texture replaceRegion:region
           mipmapLevel:0
             withBytes:imageData
           bytesPerRow:bytesPerRow];
```

```
// Loading Image Data

NSUInteger bytesPerRow = 4 * image.width;

MTLRegion region =
{
        { 0, 0, 0 },    // Origin
        { 512, 512, 1 } // Size
};

[texture replaceRegion:region
           mipmapLevel:0
             withBytes:imageData
           bytesPerRow:bytesPerRow];
```

Texture Differences

Sampler state never part of texture

- Wrap modes, filtering, min/max LOD

Texture Differences

Sampler state never part of texture

- Wrap modes, filtering, min/max LOD

Texture image data not flipped

- OpenGL uses bottom-left origin, Metal uses top-left origin

Texture Differences

Sampler state never part of texture

- Wrap modes, filtering, min/max LOD

Texture image data not flipped

- OpenGL uses bottom-left origin, Metal uses top-left origin

Metal does not perform format conversion

Textures

Buffers

Shaders

Textures

Buffers

Shaders

Buffers

Metal uses buffers for vertices, indices, and all uniform data

OpenGL's vertex, element, and uniform buffers are similar

- Easier to port apps that have adopted these

```
// Creating Buffers

id<MTLBuffer> buffer = [device newBufferWithLength:bufferDataByteSize
                                            options:MTLResourceStorageModeShared];

struct MyUniforms *uniforms =  (struct MyUniforms*) buffer.contents;

uniforms->modelViewProjection = modelViewProjection;
uniforms->sunPosition         = sunPosition;
```

```
// Creating Buffers

id<MTLBuffer> buffer = [device newBufferWithLength:bufferDataByteSize
                                           options:MTLResourceStorageModeShared];

struct MyUniforms *uniforms =  (struct MyUniforms*) buffer.contents;

uniforms->modelViewProjection = modelViewProjection;
uniforms->sunPosition         = sunPosition;
```

```
// Creating Buffers

id<MTLBuffer> buffer = [device newBufferWithLength:bufferDataByteSize
                                            options:MTLResourceStorageModeShared];

struct MyUniforms *uniforms =  (struct MyUniforms*) buffer.contents;

uniforms->modelViewProjection = modelViewProjection;
uniforms->sunPosition         = sunPosition;
```

Notes About Buffer Data

Pay attention to alignment

Type	Alignment
`float3, int3, uint3`	16 bytes
`float3x3, float4x3`	16 bytes
`half3, short3, ushore3,`	8 bytes
`half3x3, half4x3`	8 bytes
`structures`	4 bytes

Notes About Buffer Data

Pay attention to alignment

Type	Alignment
`float3, int3, uint3`	16 bytes
`float3x3, float4x3`	16 bytes
`half3, short3, ushore3,`	8 bytes
`half3x3, half4x3`	8 bytes
`structures`	4 bytes

Notes About Buffer Data

Pay attention to alignment

Type	Alignment
`float3, int3, uint3`	16 bytes
`float3x3, float4x3`	16 bytes
`half3, short3, ushore3,`	8 bytes
`half3x3, half4x3`	8 bytes
`structures`	4 bytes

Notes About Buffer Data

SIMD and packed types

SIMD libraries vector and matrix types follow same rules as Metal shaders

Special packed vector types available to shaders

- `packed_float3` consumes 12 bytes
- `packed_half3` consumes 6 bytes

Cannot directly operate on packed types

- Cast to non-packed type required

Storage Modes for Porting

Use most convient storage modes

- Easier access to data

Storage Modes for Porting

Use most convient storage modes

- Easier access to data

On iOS

- Create all textures and buffers with `MTLStorageModeShared`

Storage Modes for Porting

Use most convient storage modes

- Easier access to data

On iOS

- Create all textures and buffers with `MTLStorageModeShared`

On macOS

- Create all textures with `MTLStorageModeManaged`
- Make judicious use of `MTLStorageModeShared` for buffers
  - Separate GPU only data from CPU accessible data

MetalKit

Texture and buffer utilities

Texture Loading

- Textures from KTX, PVR, JPG, PNG, TIFF, etc.

Model Loading

- Vertex buffers from USD, OBJ, Alembic, etc.

Textures

Buffers

Pipelines

Textures

Buffers

Pipelines

Render Pipeline Descriptor

Render Pipeline
Descriptor

Vertex Shader

Fragment Shader

Device

Render Pipeline Descriptor

Vertex Layout

Vertex Shader

Fragment Shader

Device

Render Pipeline Descriptor

Vertex Layout

Vertex Shader

Fragment Shader

Blend State

Render Target Pixel Formats

Device

Render Pipeline Descriptor

Vertex Layout

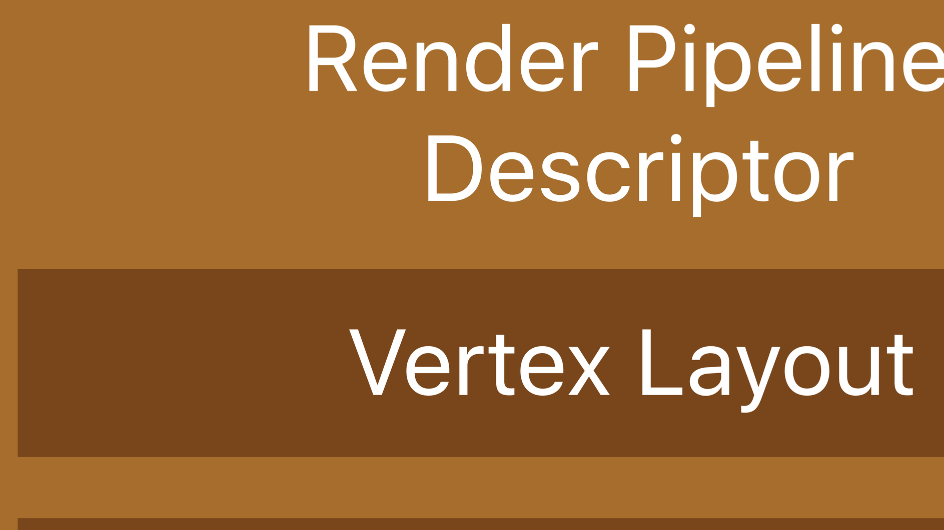
Vertex Shader

Fragment Shader

Blend State

Render Target Pixel Formats

Device


Render Pipeline Descriptor
Vertex Layout
Vertex Shader
Fragment Shader
Blend State
Render Target Pixel Formats

Device
Render State Pipeline Object

Device

Render State
Pipeline Object

```
// Creating Render Pipeline Objects

id<MTLLibrary> defaultLibrary = [device newDefaultLibrary];

id<MTLFunction> vertexFunction = [defaultLibrary newFunctionWithName:@"vertexShader"];
id<MTLFunction> fragmentFunction = [defaultLibrary newFunctionWithName:@"fragmentShader"];

MTLRenderPipelineDescriptor *pipelineStateDescriptor = [MTLRenderPipelineDescriptor new];
pipelineStateDescriptor.vertexFunction = vertexFunction;
pipelineStateDescriptor.fragmentFunction = fragmentFunction;
pipelineStateDescriptor.colorAttachments[0].pixelFormat = MTLPixelFormatRGBA8Unorm;

id<MTLRenderPipelineState> pipelineState;
pipelineState = [device newRenderPipelineStateWithDescriptor:pipelineStateDescriptor
                                                       error:nil];
```

```
// Creating Render Pipeline Objects

id<MTLLibrary> defaultLibrary = [device newDefaultLibrary];

id<MTLFunction> vertexFunction = [defaultLibrary newFunctionWithName:@"vertexShader"];
id<MTLFunction> fragmentFunction = [defaultLibrary newFunctionWithName:@"fragmentShader"];

MTLRenderPipelineDescriptor *pipelineStateDescriptor = [MTLRenderPipelineDescriptor new];
pipelineStateDescriptor.vertexFunction = vertexFunction;
pipelineStateDescriptor.fragmentFunction = fragmentFunction;
pipelineStateDescriptor.colorAttachments[0].pixelFormat = MTLPixelFormatRGBA8Unorm;

id<MTLRenderPipelineState> pipelineState;
pipelineState = [device newRenderPipelineStateWithDescriptor:pipelineStateDescriptor
                                                       error:nil];
```

```
// Creating Render Pipeline Objects

id<MTLLibrary> defaultLibrary = [device newDefaultLibrary];

id<MTLFunction> vertexFunction = [defaultLibrary newFunctionWithName:@"vertexShader"];
id<MTLFunction> fragmentFunction = [defaultLibrary newFunctionWithName:@"fragmentShader"];

MTLRenderPipelineDescriptor *pipelineStateDescriptor = [MTLRenderPipelineDescriptor new];
pipelineStateDescriptor.vertexFunction = vertexFunction;
pipelineStateDescriptor.fragmentFunction = fragmentFunction;
pipelineStateDescriptor.colorAttachments[0].pixelFormat = MTLPixelFormatRGBA8Unorm;

id<MTLRenderPipelineState> pipelineState;
pipelineState = [device newRenderPipelineStateWithDescriptor:pipelineStateDescriptor
                                                        error:nil];
```

```
// Creating Render Pipeline Objects

id<MTLLibrary> defaultLibrary = [device newDefaultLibrary];

id<MTLFunction> vertexFunction = [defaultLibrary newFunctionWithName:@"vertexShader"];
id<MTLFunction> fragmentFunction = [defaultLibrary newFunctionWithName:@"fragmentShader"];

MTLRenderPipelineDescriptor *pipelineStateDescriptor = [MTLRenderPipelineDescriptor new];
pipelineStateDescriptor.vertexFunction = vertexFunction;
pipelineStateDescriptor.fragmentFunction = fragmentFunction;
pipelineStateDescriptor.colorAttachments[0].pixelFormat = MTLPixelFormatRGBA8Unorm;

id<MTLRenderPipelineState> pipelineState;
pipelineState = [device newRenderPipelineStateWithDescriptor:pipelineStateDescriptor
                                                       error:nil];
```

```objc
// Creating Render Pipeline Objects

id<MTLLibrary> defaultLibrary = [device newDefaultLibrary];

id<MTLFunction> vertexFunction = [defaultLibrary newFunctionWithName:@"vertexShader"];
id<MTLFunction> fragmentFunction = [defaultLibrary newFunctionWithName:@"fragmentShader"];

MTLRenderPipelineDescriptor *pipelineStateDescriptor = [MTLRenderPipelineDescriptor new];
pipelineStateDescriptor.vertexFunction = vertexFunction;
pipelineStateDescriptor.fragmentFunction = fragmentFunction;
pipelineStateDescriptor.colorAttachments[0].pixelFormat = MTLPixelFormatRGBA8Unorm;

id<MTLRenderPipelineState> pipelineState;
pipelineState = [device newRenderPipelineStateWithDescriptor:pipelineStateDescriptor
                                                        error:nil];
```

```objc
// Creating Render Pipeline Objects

id<MTLLibrary> defaultLibrary = [device newDefaultLibrary];

id<MTLFunction> vertexFunction = [defaultLibrary newFunctionWithName:@"vertexShader"];
id<MTLFunction> fragmentFunction = [defaultLibrary newFunctionWithName:@"fragmentShader"];

MTLRenderPipelineDescriptor *pipelineStateDescriptor = [MTLRenderPipelineDescriptor new];
pipelineStateDescriptor.vertexFunction = vertexFunction;
pipelineStateDescriptor.fragmentFunction = fragmentFunction;
pipelineStateDescriptor.colorAttachments[0].pixelFormat = MTLPixelFormatRGBA8Unorm;

id<MTLRenderPipelineState> pipelineState;
pipelineState = [device newRenderPipelineStateWithDescriptor:pipelineStateDescriptor
                                                       error:nil];
```

Pipeline Differences

OpenGL Program Objects

OpenGL

- Vertex Shader
- Fragment Shader

Metal

- Vertex Layout
- Vertex Shader
- Fragment Shader
- Blend State
- Render Target Pixel Formats

Pipeline Building

Create at intitlization

Full compilation key advantage of state grouping

Choose a canonical vertex layout for meshes

Use a limited set of render target formats

Pipeline Building

Lazy creation at draw time

!

Store pipeline state objects in a dictionary using descriptor as key

Construct descriptor at draw time with current state

Retrieve existing pipeline from dictionary OR build new pipeline

Create Render Objects at Initialization

Object creation expensive

Create Render Objects at Initialization

Object creation expensive

- Pipelines require backend compilation

Create Render Objects at Initialization

Object creation expensive

- Pipelines require backend compilation
- Buffers and textures need allocations

Create Render Objects at Initialization

Object creation expensive

- Pipelines require backend compilation
- Buffers and textures need allocations

Once created, much faster usage during rendering

Porting the Render Loop

Sukanya Sudugu, GPU Software Engineer

Build	Initialize	Render
Shaders	Devices and Queues	Command Buffers
	Render Objects	Resource Updates
		Render Encoders
		Display

Build

Shaders

Initialize

Devices and Queues

Render Objects

Render

Command Buffers

Resource Updates

Render Encoders

Display

Build	Initialize	Render
Shaders	Devices and Queues	Command Buffers
	Render Objects	Resource Updates
		Render Encoders
		Display

Build	Initialize	Render
Shaders	Devices and Queues	Command Buffers
	Render Objects	Resource Updates
		Render Encoders
		Display

Build	Initialize	Render
Shaders	Devices and Queues	Command Buffers
	Render Objects	Resource Updates
		Render Encoders
		Display

Application Renderer

Metal API

Command Encoder

Command Buffer

Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Display

Application Renderer

Metal API

Command Encoder

Command Buffer

Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Display

Command Buffers

Explicit control over command buffer submission

Start with one command buffer per frame

Optionally split a frame into multiple command buffers to

- Submit early and get the GPU started
- Build commands on multiple threads

Command Buffers

Explicit control over command buffer submission

Start with one command buffer per frame

Optionally split a frame into multiple command buffers to

- Submit early and get the GPU started
- Build commands on multiple threads

Completion handler invoked when execution is finished

```
// Render Loop

// Obtaining a command buffer at the beginning of each frame
id<MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// Encoding commands
...

// Commit the command buffer to the GPU for execution
[commandBuffer commit];
```

```
// Render Loop

// Obtaining a command buffer at the beginning of each frame
id<MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// Encoding commands
...

// Commit the command buffer to the GPU for execution
[commandBuffer commit];
```

```
// Render Loop

// Obtaining a command buffer at the beginning of each frame
id<MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// Encoding commands
...

// Commit the command buffer to the GPU for execution
[commandBuffer commit];
```

```
// Render Loop

// Obtaining a command buffer at the beginning of each frame
id<MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// Encoding commands
...

// Commit the command buffer to the GPU for execution
[commandBuffer commit];
```

```objc
// Render Loop

// Obtaining a command buffer at the beginning of each frame
id<MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// Encoding commands
...

// Commit the command buffer to the GPU for execution
[commandBuffer commit];


// Wait until the GPU has finished execution
[commandBuffer waitUntilCompleted];
```

```
// Render Loop

// Obtaining a command buffer at the beginning of each frame
id<MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// Encoding commands
...

// Commit the command buffer to the GPU for execution
[commandBuffer commit];
```

```
// Render Loop

// Obtaining a command buffer at the beginning of each frame
id<MTLCommandBuffer> commandBuffer = [commandQueue commandBuffer];

// Encoding commands
...

// Add a completion hander to tell me when the GPU is done
[commandBuffer addCompletedHandler:^(id<MTLCommandBuffer> commandBuffer)
{
    // GPU is done with my buffer!
    ...
}];

// Commit the command buffer to the GPU for execution
[commandBuffer commit];
```

Build	Initialize	Render
Shaders	Devices and Queues	Command Buffers
	Render Objects	Resource Updates
		Render Encoders
		Display

Resource Updates

Resources are explicitly managed in Metal

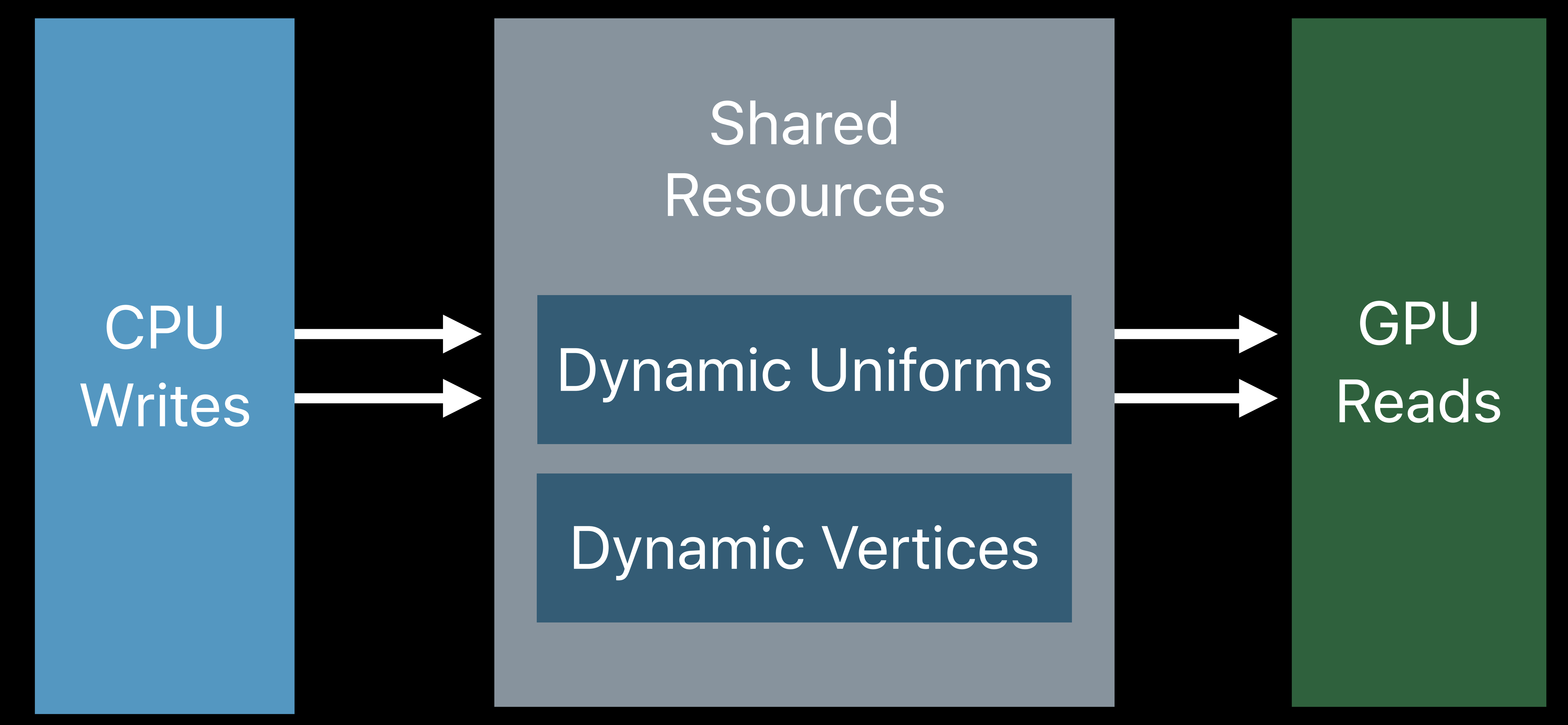
No implicit synchronization like OpenGL

Allows for fine gained synchronization

Application has complete control

Best model dependent on usage

- Triple buffering recommended



Resource Updates

Without synchronization

CPU

Write

Buffer

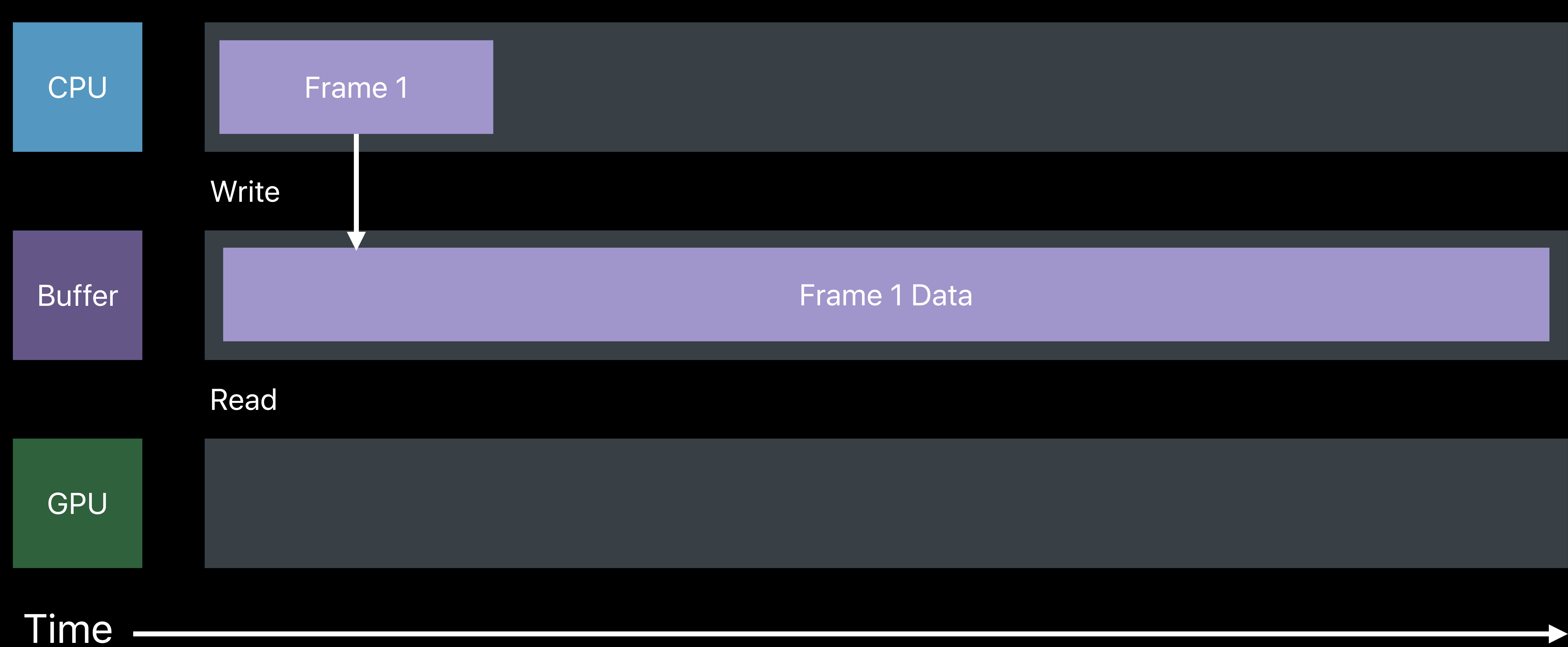
Read

GPU

Time

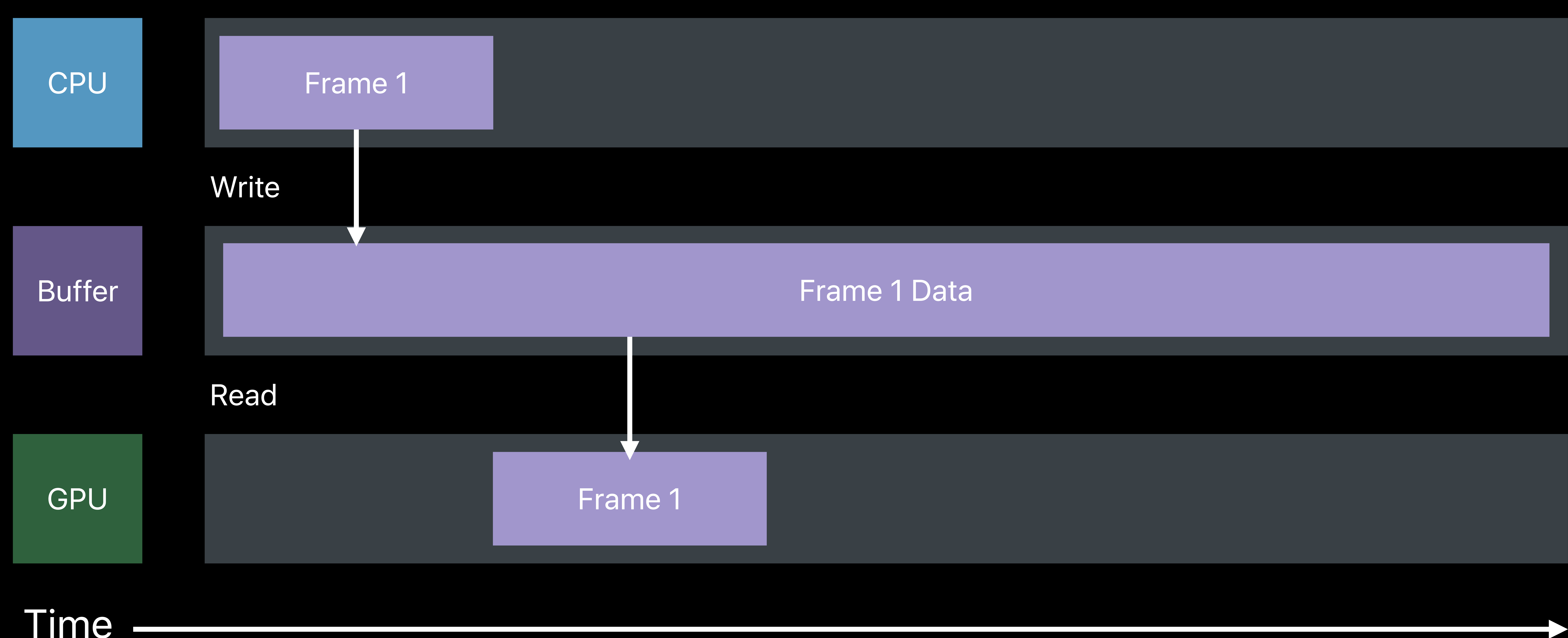
Resource Updates

Without synchronization



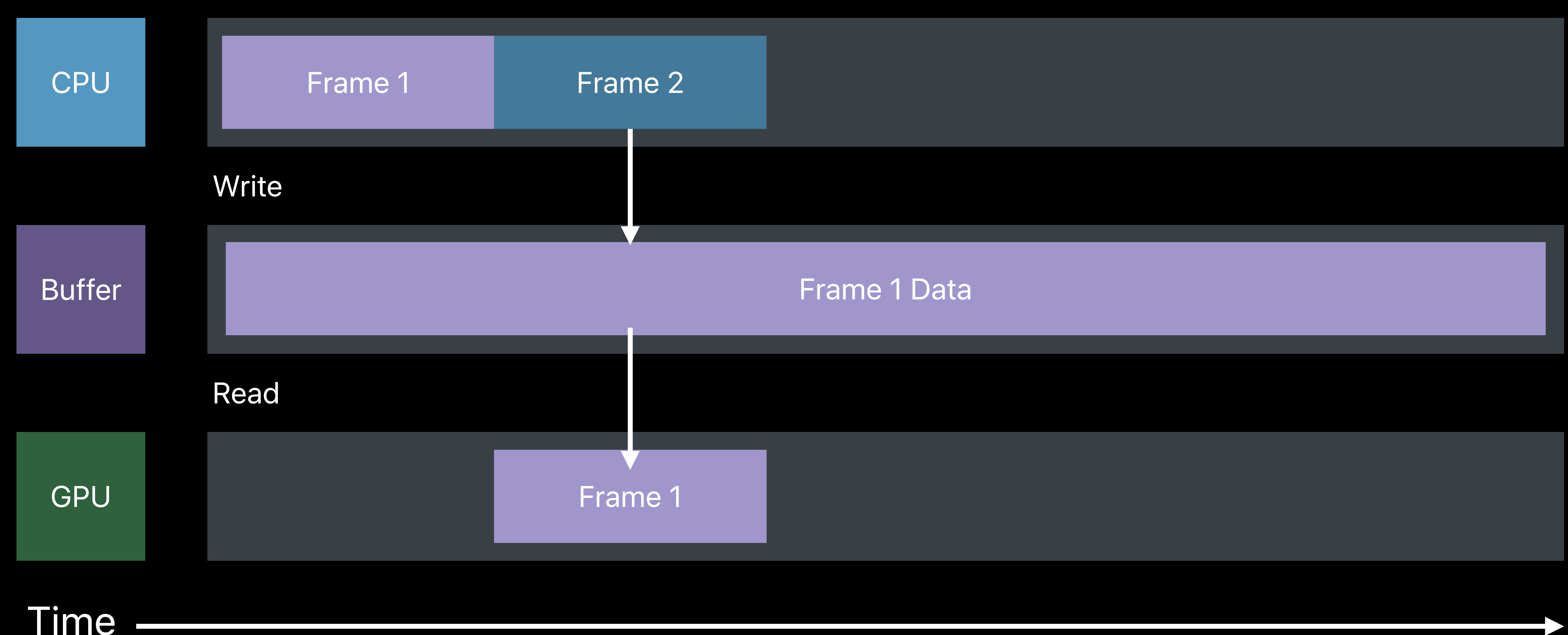
Resource Updates

Without synchronization



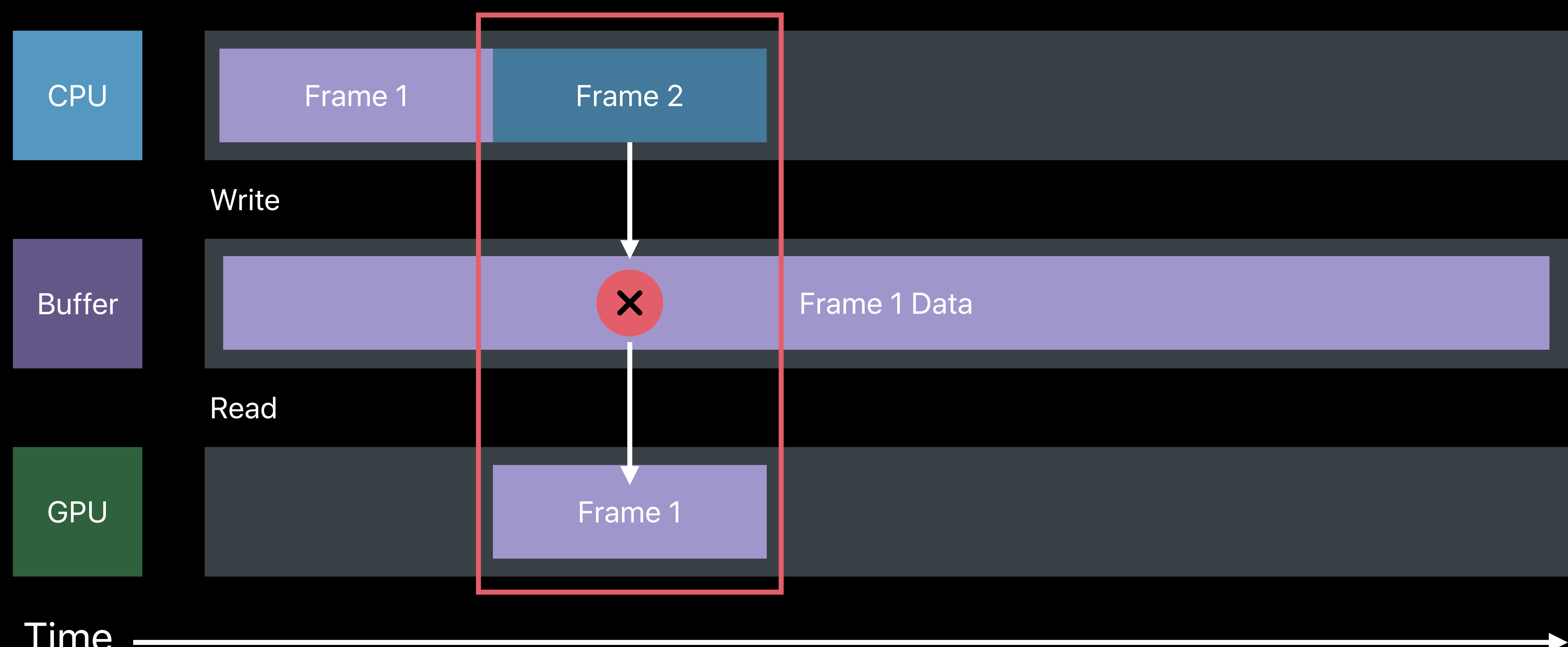
Resource Updates

Without synchronization



Resource Updates

Without synchronization



Temporary Solution

Synchronous wait after every frame

CPU	Frame 1	Wait	Frame 2
Write			
Buffer	Frame 2 Data		
Read			
GPU		Frame 1	Idle

Time →

Temporary Solution

Synchronous wait after every frame

CPU	Frame 1	Wait	Frame 2

Write

Buffer	Frame 2 Data

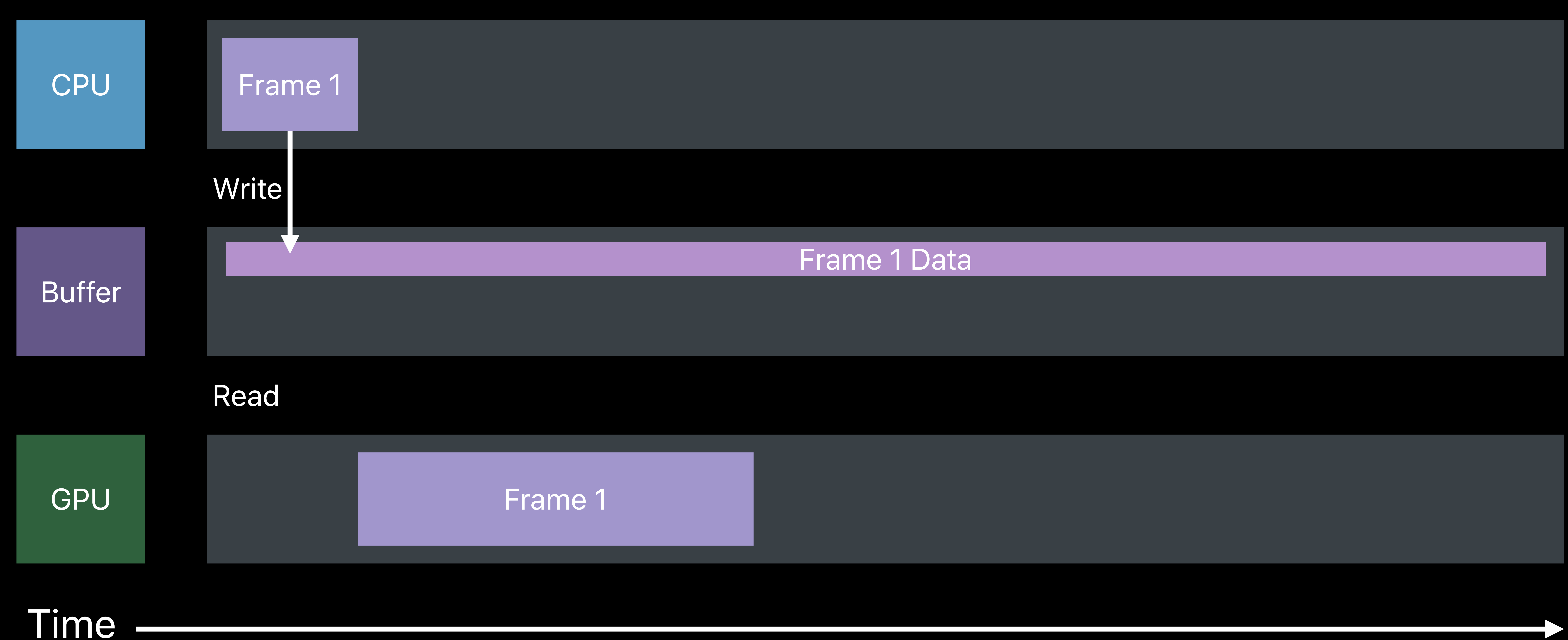
Read

GPU	Frame 1	Idle

Time →

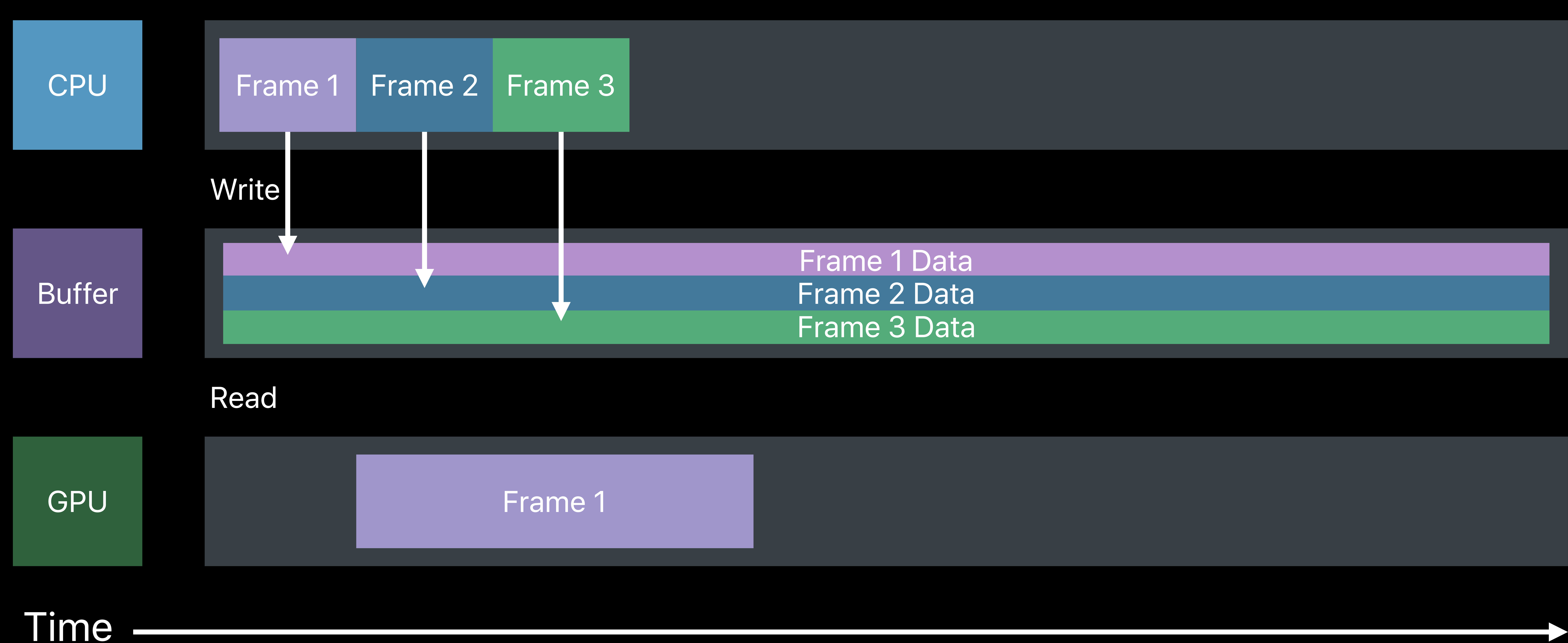
Triple Buffering

Shared buffer pool



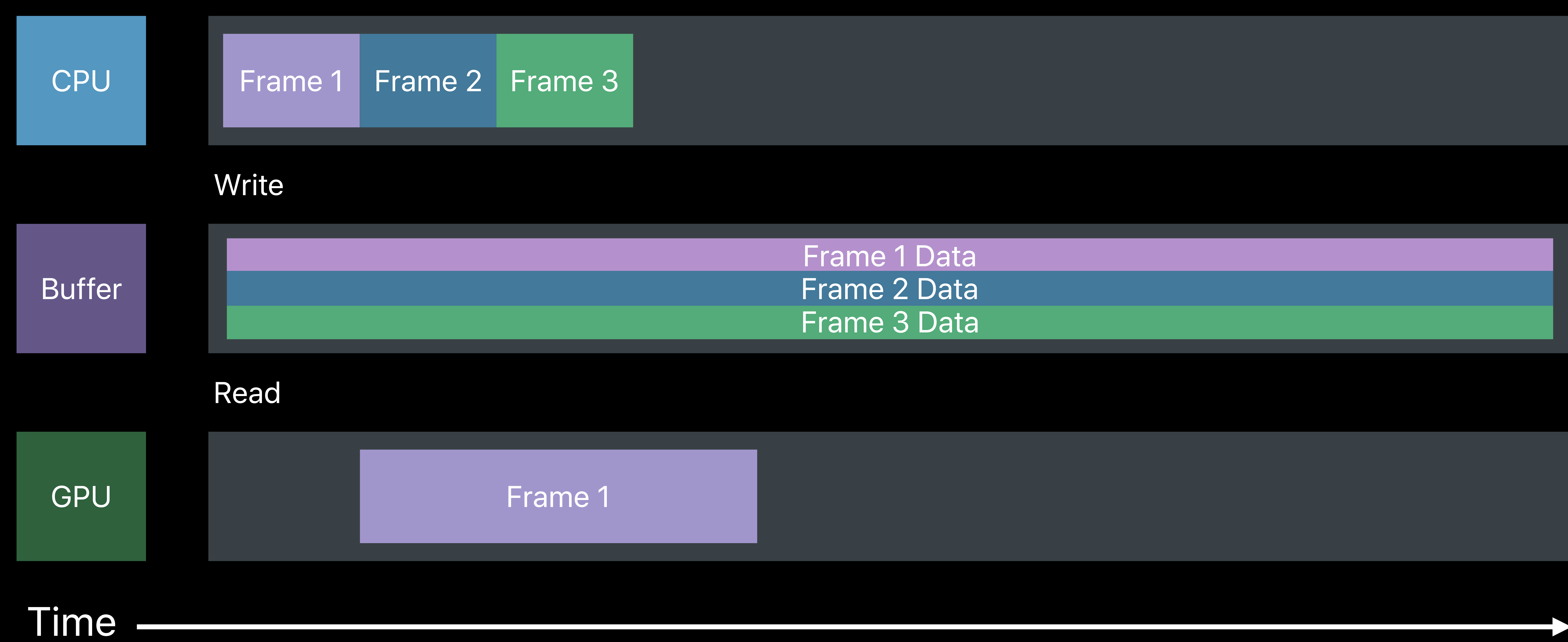
Triple Buffering

Shared buffer pool



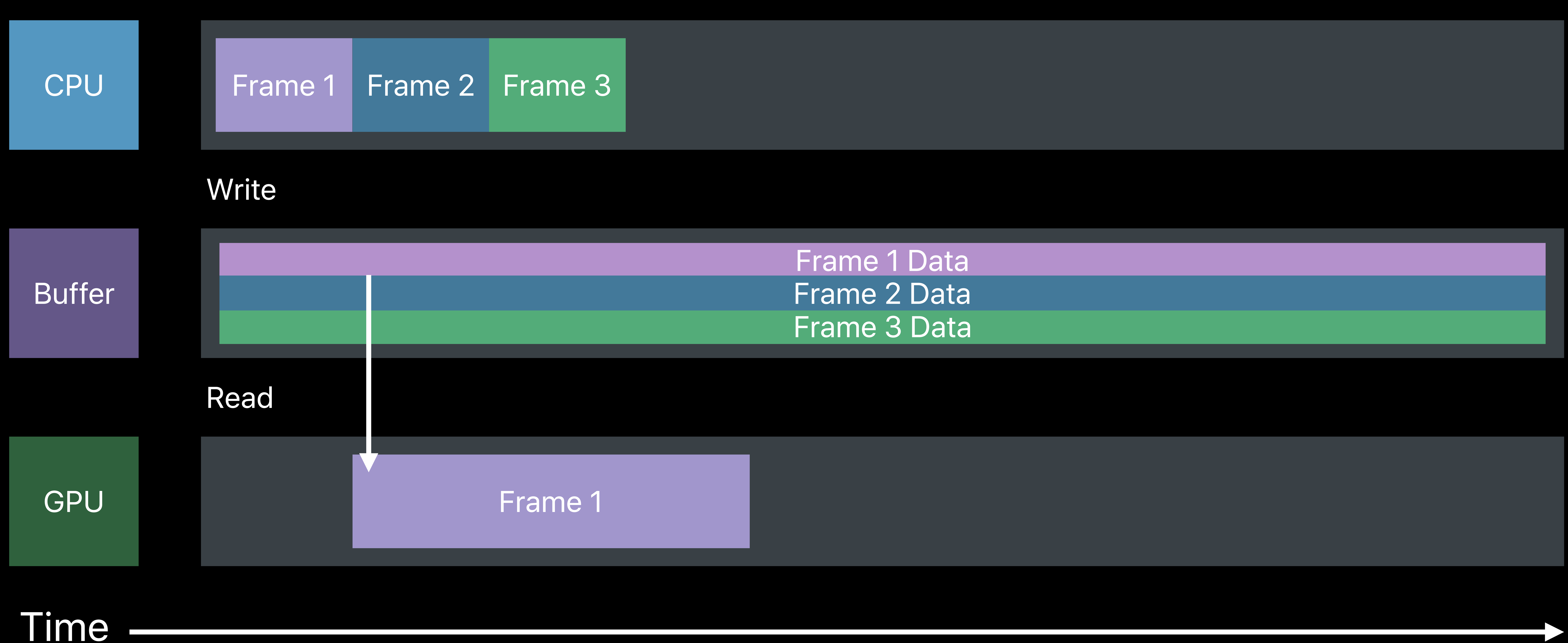
Triple Buffering

Shared buffer pool



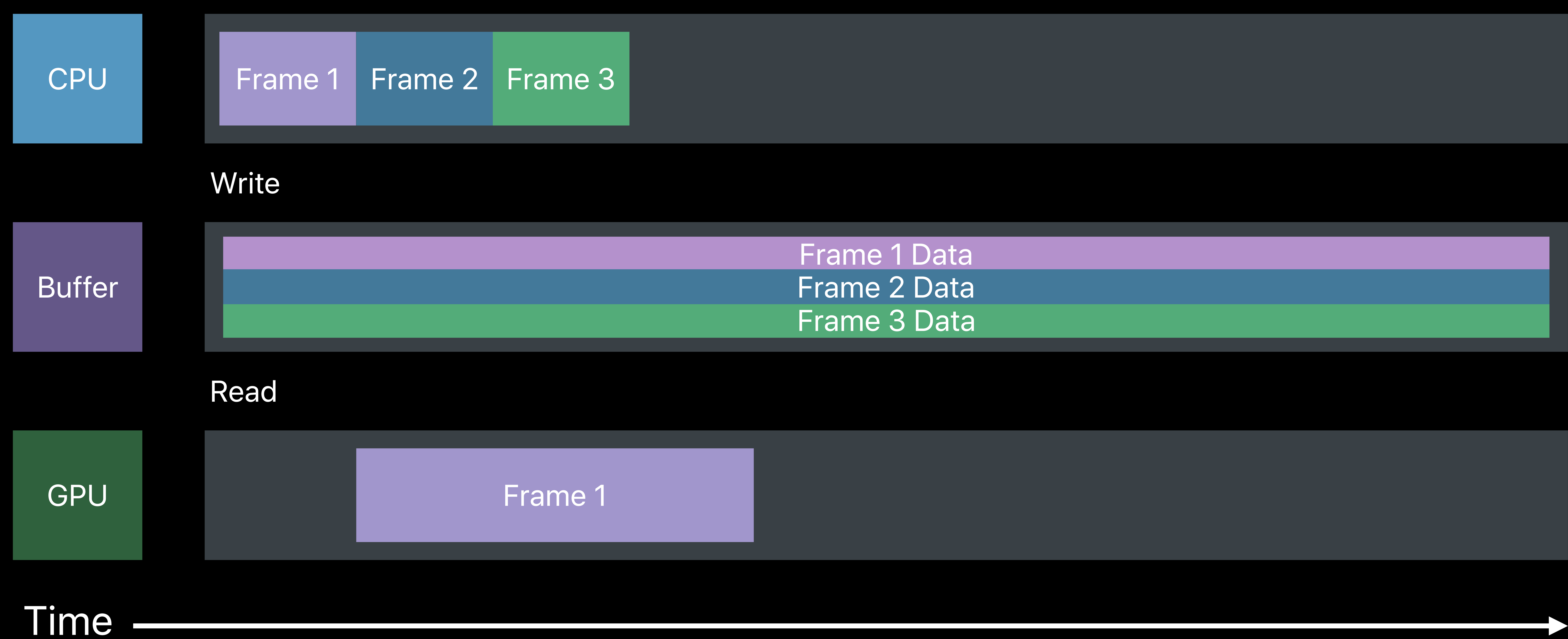
Triple Buffering

Shared buffer pool



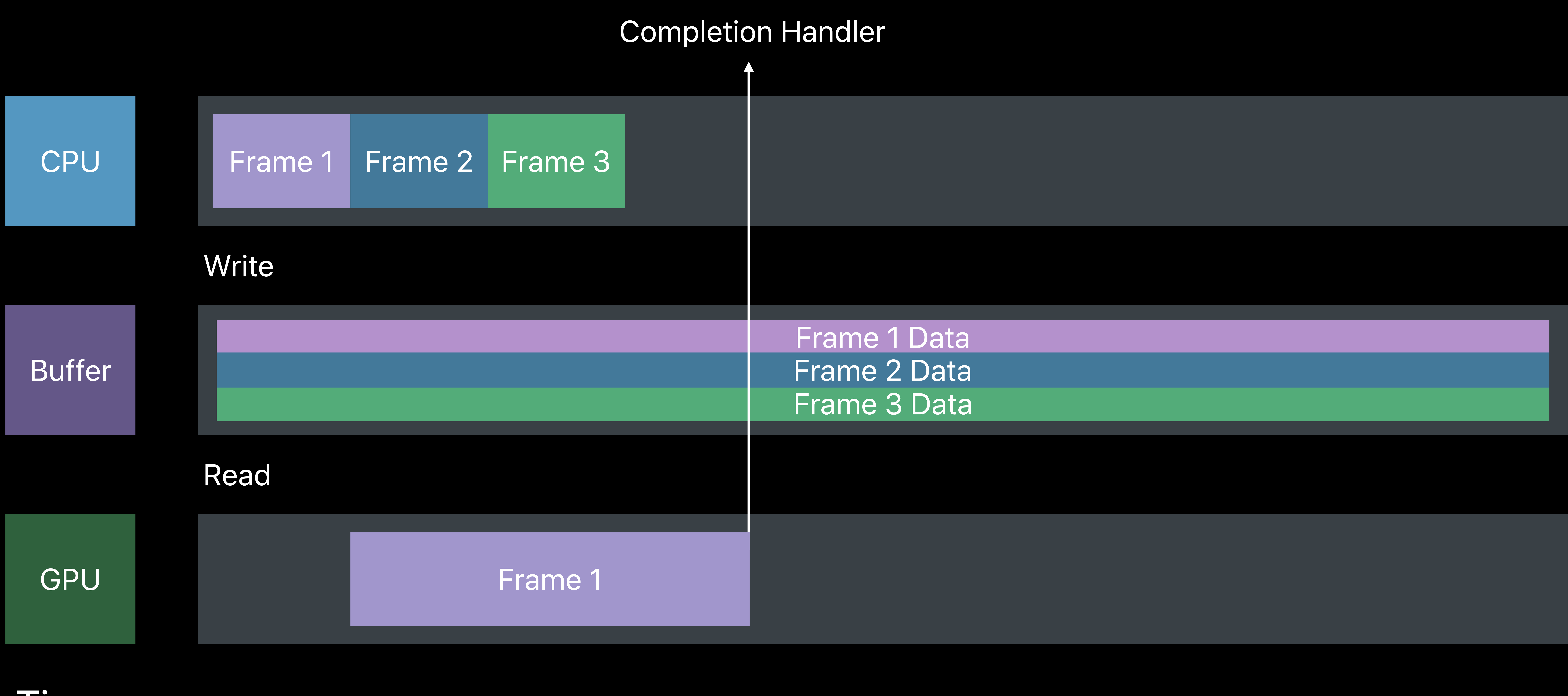
Triple Buffering

Shared buffer pool



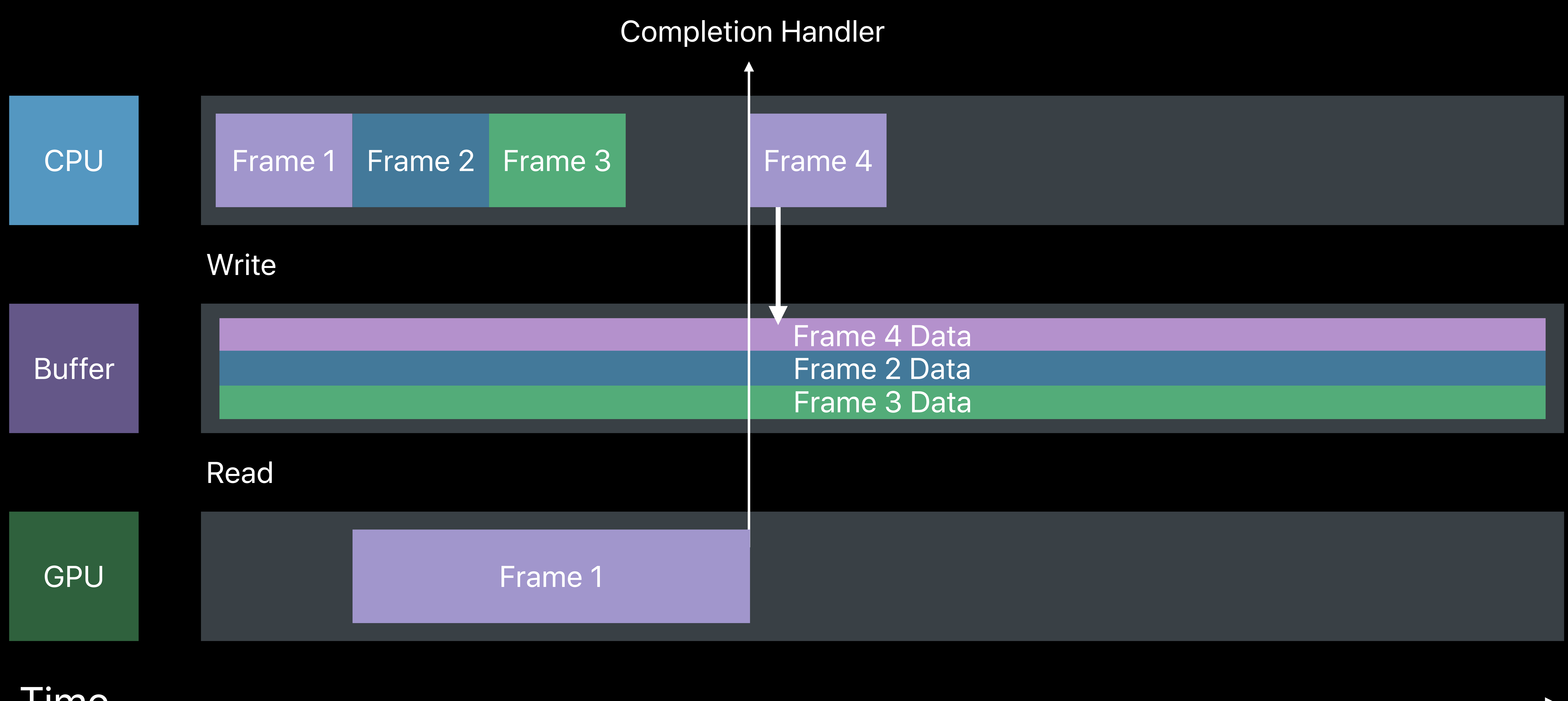
Triple Buffering

Shared buffer pool



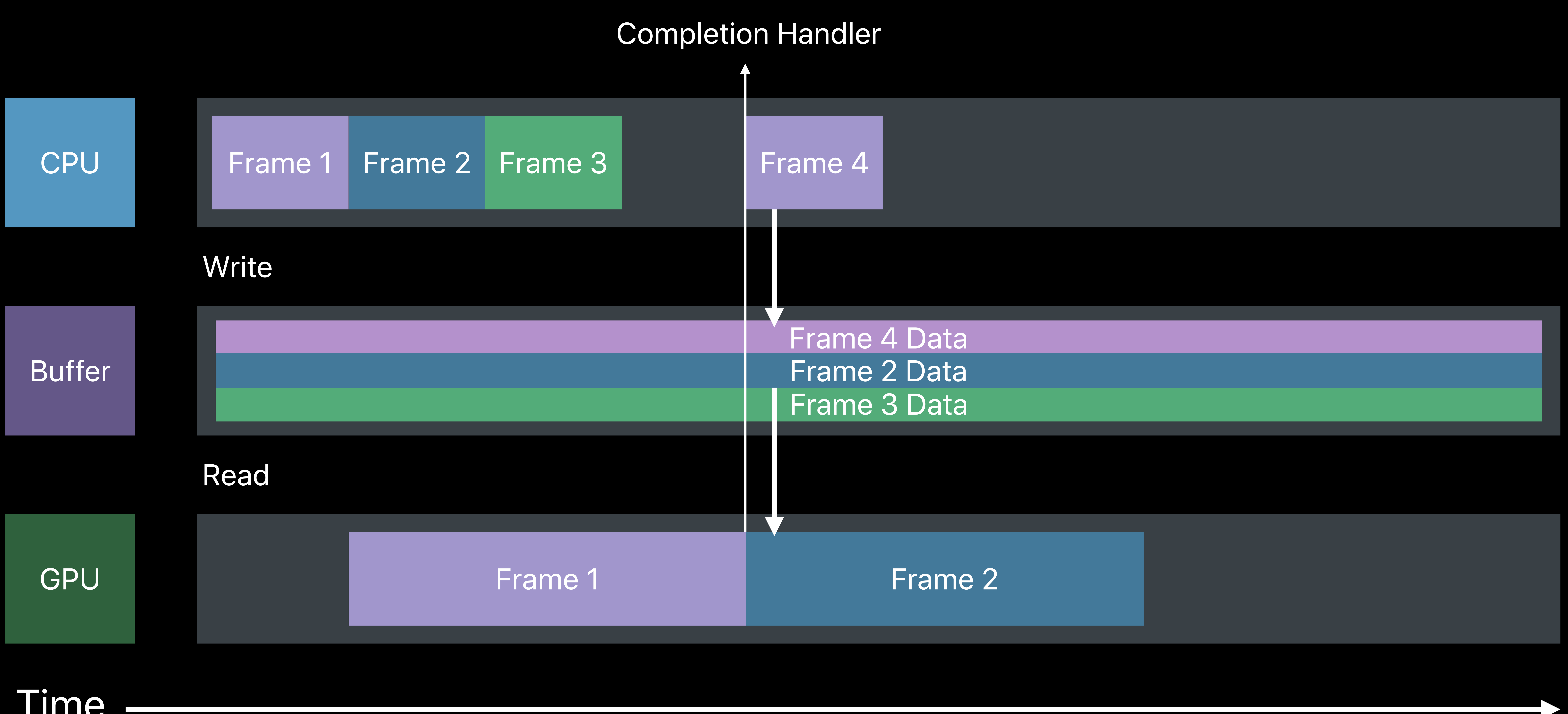
Triple Buffering

Shared buffer pool



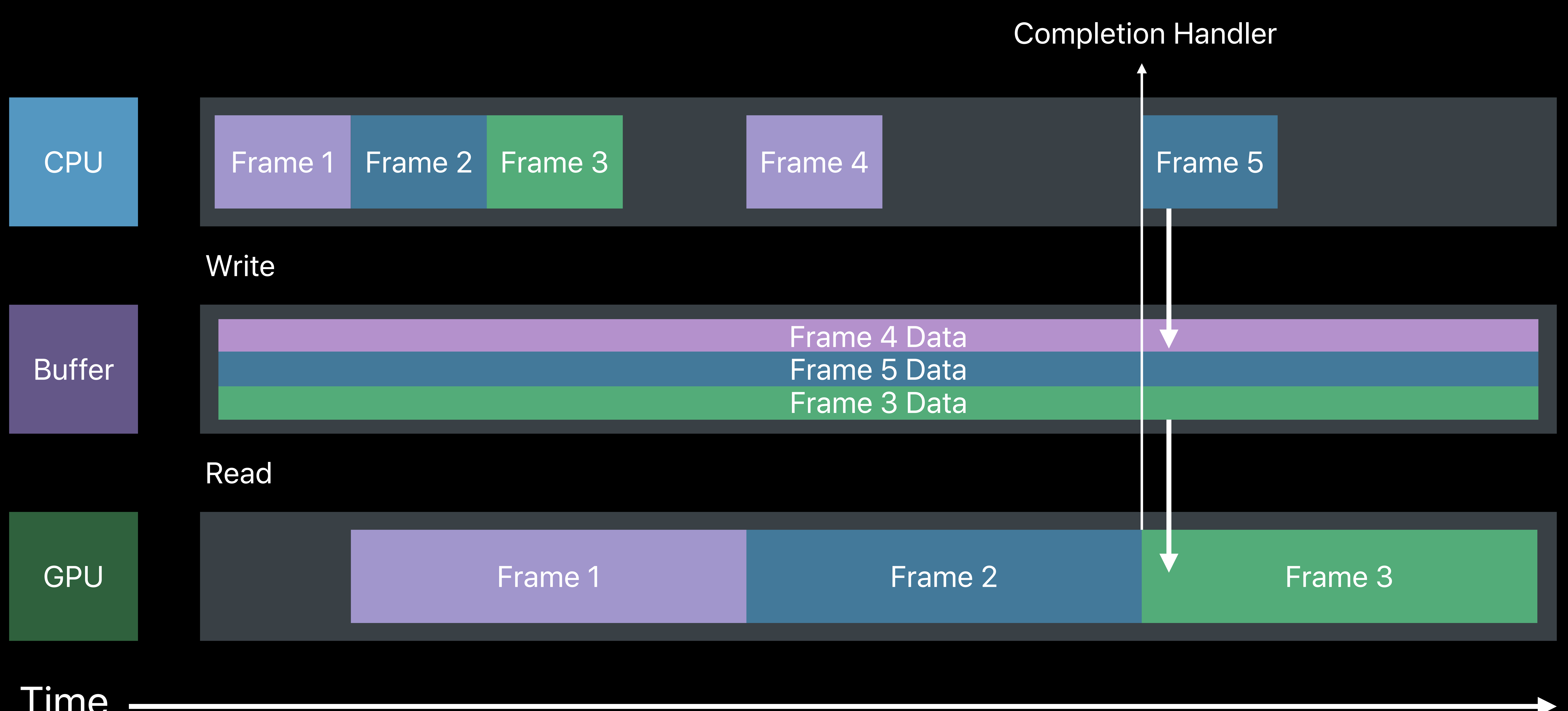
Triple Buffering

Shared buffer pool



Triple Buffering

Shared buffer pool



```
// Triple Buffering Implementation

// Create FIFO queue of three dynamic data uniform buffers
id <MTLBuffer> myUniformBuffers[3];

// Create a semaphore that gets signaled at each frame boundary.
// The GPU signals the semaphore once it completes a frame’s work,
// allowing CPU To work on a new frame
dispatch_semaphore_t frameBoundarySemaphore = dispatch_semaphore_create(3);

// Current frame Index
NSUInteger currentUniformIndex = 0;
```

```
// Triple Buffering Implementation

// Create FIFO queue of three dynamic data uniform buffers
id <MTLBuffer> myUniformBuffers[3];

// Create a semaphore that gets signaled at each frame boundary.
// The GPU signals the semaphore once it completes a frame's work,
// allowing CPU To work on a new frame
dispatch_semaphore_t frameBoundarySemaphore = dispatch_semaphore_create(3);

// Current frame Index
NSUInteger currentUniformIndex = 0;
```

```
// Triple Buffering Implementation

// Create FIFO queue of three dynamic data uniform buffers
id <MTLBuffer> myUniformBuffers[3];

// Create a semaphore that gets signaled at each frame boundary.
// The GPU signals the semaphore once it completes a frame's work,
// allowing CPU To work on a new frame
dispatch_semaphore_t frameBoundarySemaphore = dispatch_semaphore_create(3);

// Current frame Index
NSUInteger currentUniformIndex = 0;
```

```
// Wait until inflight frame is completed
dispatch_semaphore_wait(frameBoundarySemaphore, DISPATCH_TIME_FOREVER);

// Grab current frame and update its buffer
currentUniformIndex = (currentUniformIndex + 1) % 3;
[self updateUniformResource: myUniformBuffers[currentUniformIndex]];

// Encode commands and bind uniform buffer for GPU access

// Schedule frame completion handler
[commandBuffer addCompletedHandler:^(id<MTLCommandBuffer> commandBuffer) {
   // GPU work is complete. Signal the Semaphore to start CPU work
   dispatch_semaphore_signal(frameBoundarySemaphore);
}];

// Finalize and commit frame to GPU
[commandBuffer commit];
```

```
// Wait until inflight frame is completed
dispatch_semaphore_wait(frameBoundarySemaphore, DISPATCH_TIME_FOREVER);

// Grab current frame and update its buffer
currentUniformIndex = (currentUniformIndex + 1) % 3;
[self updateUniformResource: myUniformBuffers[currentUniformIndex]];

// Encode commands and bind uniform buffer for GPU access

// Schedule frame completion handler
[commandBuffer addCompletedHandler:^(id<MTLCommandBuffer> commandBuffer) {
   // GPU work is complete. Signal the Semaphore to start CPU work
   dispatch_semaphore_signal(frameBoundarySemaphore);
}];

// Finalize and commit frame to GPU
[commandBuffer commit];
```

```
// Wait until inflight frame is completed
dispatch_semaphore_wait(frameBoundarySemaphore, DISPATCH_TIME_FOREVER);

// Grab current frame and update its buffer
currentUniformIndex = (currentUniformIndex + 1) % 3;
[self updateUniformResource: myUniformBuffers[currentUniformIndex]];

// Encode commands and bind uniform buffer for GPU access

// Schedule frame completion handler
[commandBuffer addCompletedHandler:^(id<MTLCommandBuffer> commandBuffer) {
   // GPU work is complete. Signal the Semaphore to start CPU work
   dispatch_semaphore_signal(frameBoundarySemaphore);
}];

// Finalize and commit frame to GPU
[commandBuffer commit];
```

```
// Wait until inflight frame is completed
dispatch_semaphore_wait(frameBoundarySemaphore, DISPATCH_TIME_FOREVER);

// Grab current frame and update its buffer
currentUniformIndex = (currentUniformIndex + 1) % 3;
[self updateUniformResource: myUniformBuffers[currentUniformIndex]];

// Encode commands and bind uniform buffer for GPU access

// Schedule frame completion handler
[commandBuffer addCompletedHandler:^(id<MTLCommandBuffer> commandBuffer) {
   // GPU work is complete. Signal the Semaphore to start CPU work
   dispatch_semaphore_signal(frameBoundarySemaphore);
}];

// Finalize and commit frame to GPU
[commandBuffer commit];
```

```
// Wait until inflight frame is completed
dispatch_semaphore_wait(frameBoundarySemaphore, DISPATCH_TIME_FOREVER);

// Grab current frame and update its buffer
currentUniformIndex = (currentUniformIndex + 1) % 3;
[self updateUniformResource: myUniformBuffers[currentUniformIndex]];

// Encode commands and bind uniform buffer for GPU access

// Schedule frame completion handler
[commandBuffer addCompletedHandler:^(id<MTLCommandBuffer> commandBuffer) {
   // GPU work is complete. Signal the Semaphore to start CPU work
   dispatch_semaphore_signal(frameBoundarySemaphore);
}];

// Finalize and commit frame to GPU
[commandBuffer commit];
```

```objc
// Wait until inflight frame is completed
dispatch_semaphore_wait(frameBoundarySemaphore, DISPATCH_TIME_FOREVER);

// Grab current frame and update its buffer
currentUniformIndex = (currentUniformIndex + 1) % 3;
[self updateUniformResource: myUniformBuffers[currentUniformIndex]];

// Encode commands and bind uniform buffer for GPU access

// Schedule frame completion handler
[commandBuffer addCompletedHandler:^(id<MTLCommandBuffer> commandBuffer) {
    // GPU work is complete. Signal the Semaphore to start CPU work
    dispatch_semaphore_signal(frameBoundarySemaphore);
}];

// Finalize and commit frame to GPU
[commandBuffer commit];
```

Build

- Shaders

Initialize

- Devices and Queues
- Render Objects

Render

- Command Buffers
- Resource Updates
- Render Encoders
- Display

Application Renderer

Metal API

Command Encoder

Command Buffer

Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Display

Application Renderer

Metal API

Command Encoder

Command Buffer

Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Display

Render Pass Descriptor

Render Pass Descriptor

Color

Depth

Stencil

Command Buffer

Render Pass Descriptor

Render Pass Setup

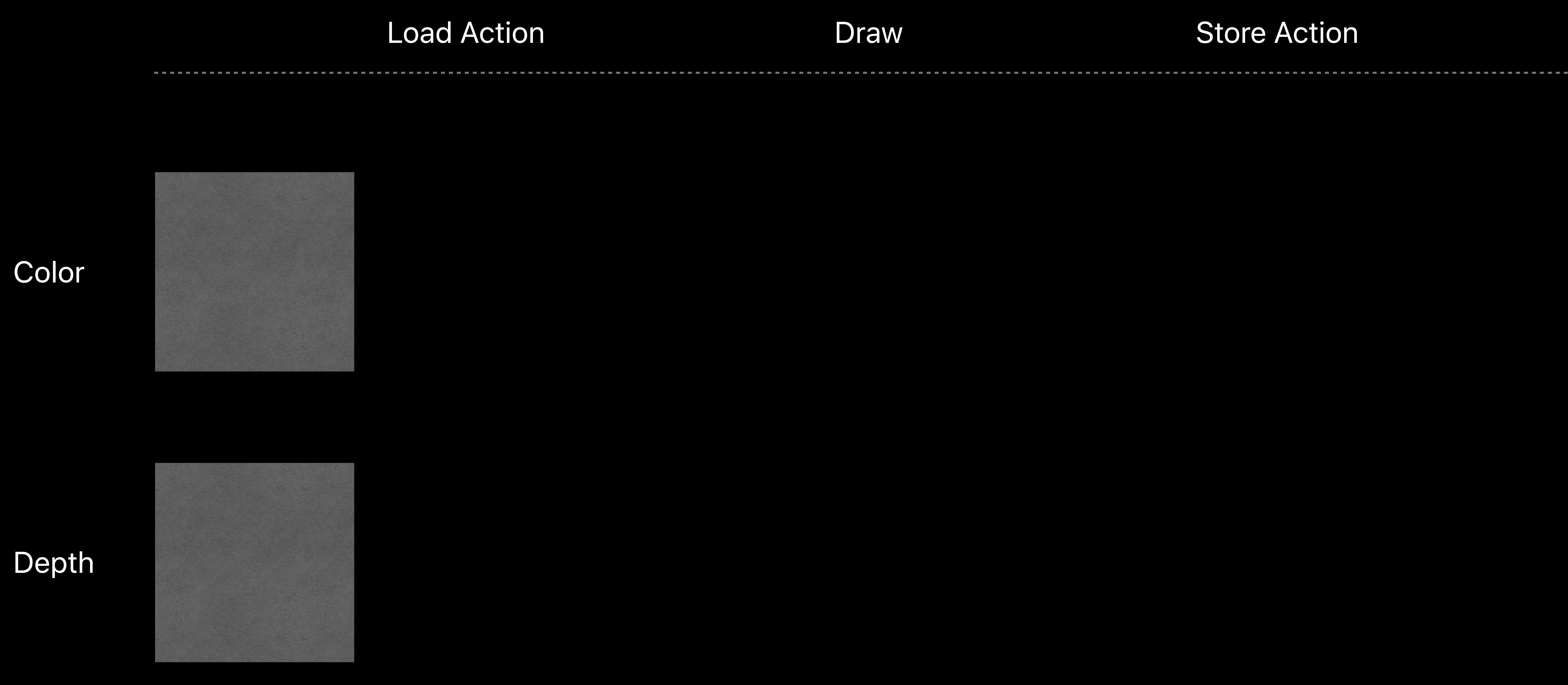
```
// Metal Render Pass descriptor
MTLRenderPassDescriptor * desc = [MTLRenderPassDescriptor new];
desc.colorAttachment[0].texture = myColorTexture;
desc.depthAttachment.texture = myDepthTexture;
id <MTLRenderCommandEncoder> encoder =  [commandBuffer renderCommandEncoderWithDescriptor: desc];
```

Render Pass Setup

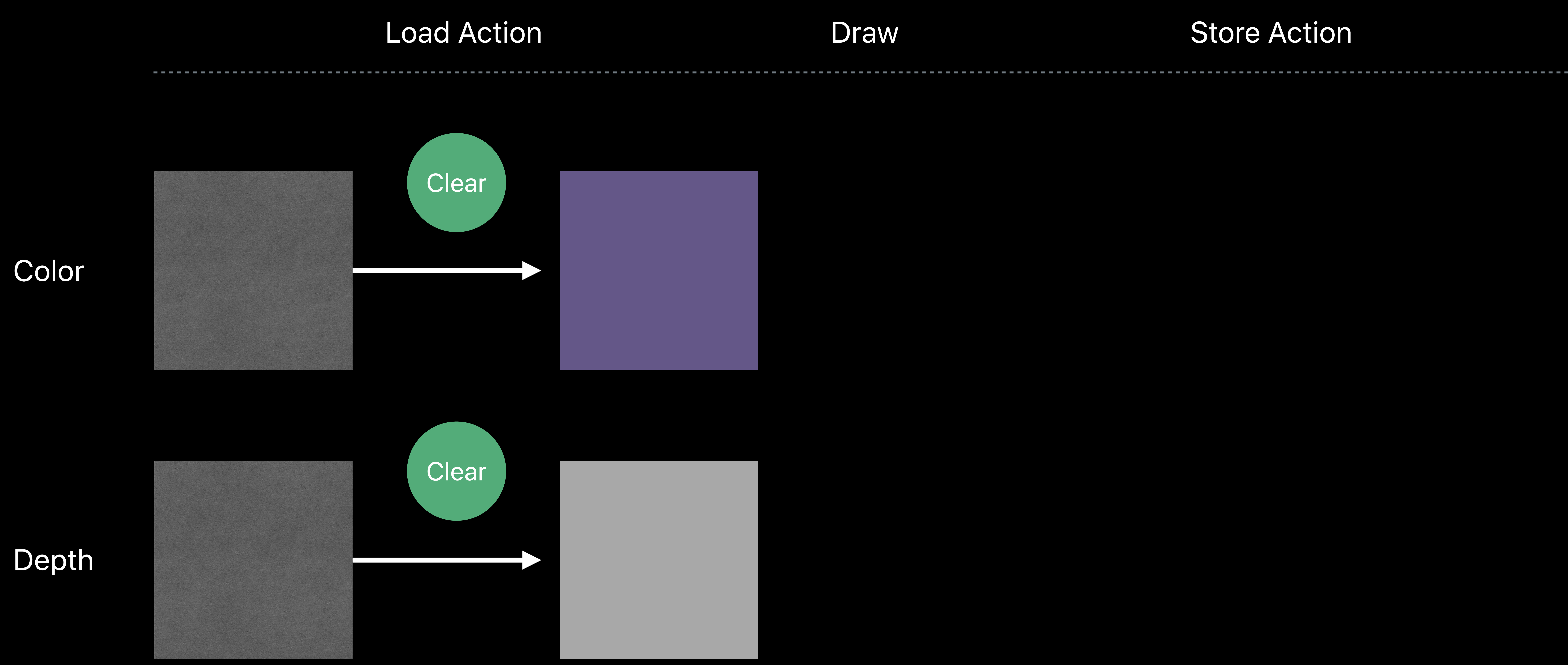
```
// Metal Render Pass descriptor
MTLRenderPassDescriptor * desc = [MTLRenderPassDescriptor new];
desc.colorAttachment[0].texture = myColorTexture;
desc.depthAttachment.texture = myDepthTexture;
id <MTLRenderCommandEncoder> encoder =  [commandBuffer renderCommandEncoderWithDescriptor: desc];
```

Render Pass Load and Store Actions

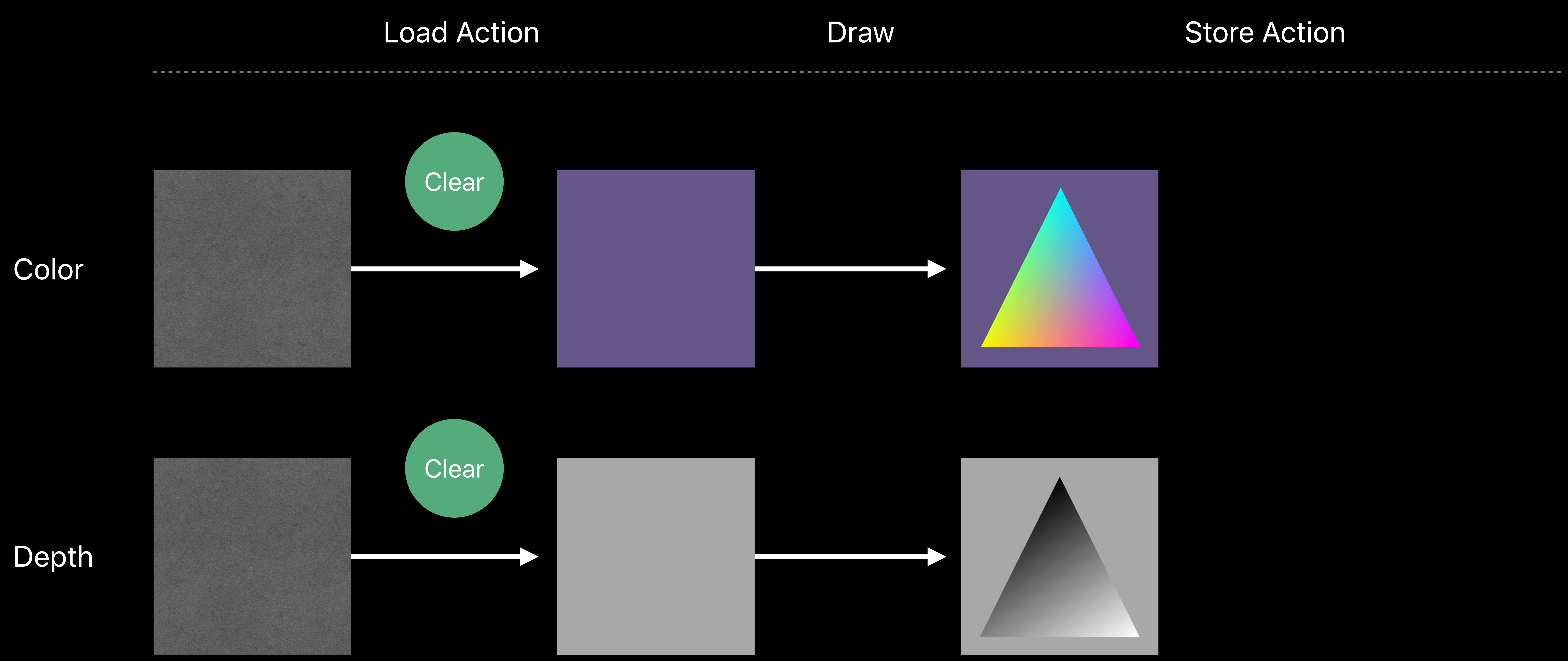
Render Pass Load and Store Actions



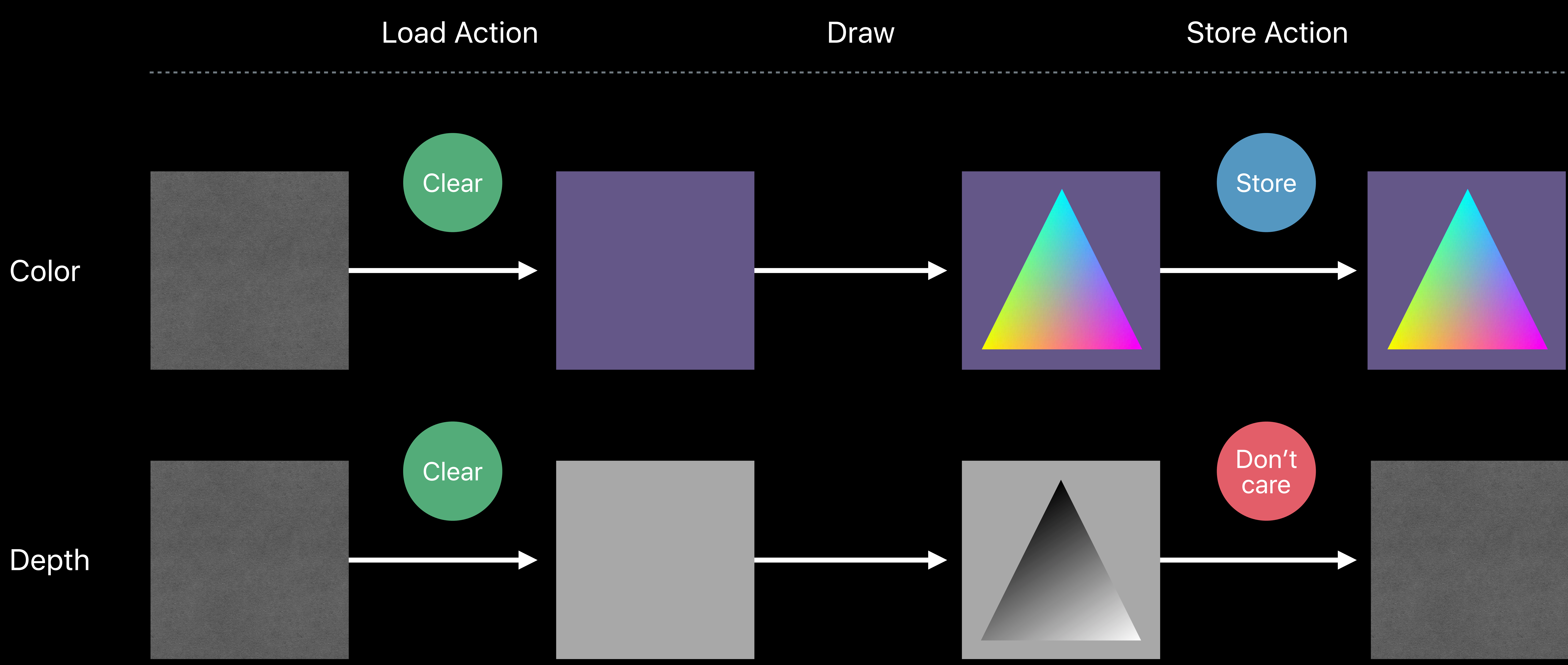
Render Pass Load and Store Actions



Render Pass Load and Store Actions



Render Pass Load and Store Actions



Render Pass Load and Store Actions

```
// Color attachment Load and Store Actions
MTLRenderPassDescriptor * desc = [MTLRenderPassDescriptor new];
desc.colorAttachment[0].texture = myColorTexture;
desc.colorAttachment[0].loadAction  = MTLLoadActionClear;
desc.colorAttachment[0].clearColor  = MTLClearColorMake(0.39f, 0.34f, 0.53f, 1.0f);
desc.colorAttachment[0].storeAction = MTLStoreActionStore;
id <MTLRenderCommandEncoder> encoder =  [commandBuffer renderCommandEncoderWithDescriptor: desc];
```

Render Pass Load and Store Actions

```
// Color attachment Load and Store Actions
MTLRenderPassDescriptor * desc = [MTLRenderPassDescriptor new];
desc.colorAttachment[0].texture = myColorTexture;
desc.colorAttachment[0].loadAction  = MTLLoadActionClear;
desc.colorAttachment[0].clearColor  = MTLClearColorMake(0.39f, 0.34f, 0.53f, 1.0f);
desc.colorAttachment[0].storeAction = MTLStoreActionStore;
id <MTLRenderCommandEncoder> encoder =  [commandBuffer renderCommandEncoderWithDescriptor: desc];
```

Render Pass Setup

```
// Color attachment Load and Store Actions
MTLRenderPassDescriptor * desc = [MTLRenderPassDescriptor new];
desc.colorAttachment[0].texture = myColorTexture;
desc.colorAttachment[0].loadAction  = MTLLoadActionClear;
desc.colorAttachment[0].clearColor  = MTLClearColorMake(0.39f, 0.34f, 0.53f, 1.0f);
desc.colorAttachment[0].storeAction = MTLStoreActionStore;
id <MTLRenderCommandEncoder> encoder =  [commandBuffer renderCommandEncoderWithDescriptor: desc];
```

Rendering with OpenGL

Render Targets	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Shaders	`glUseProgram(myProgram);`
Vertex Buffers	`glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);`
Uniforms	`glBindBuffer(GL_UNIFORM_BUFFER, myUniforms);`
Textures	`glBindTexture(GL_TEXTURE_2D, myColorTexture);`
Draws	`glDrawArrays(GL_TRIANGLES, 0, numVertices);`

Rendering with OpenGL

- Render Targets
- Shaders
- Vertex Buffers
- Uniforms
- Textures
- Draws

```
glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);

glUseProgram(myProgram);

glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);

glBindBuffer(GL_UNIFORM_BUFFER, myUniforms);

glBindTexture(GL_TEXTURE_2D, myColorTexture);

glDrawArrays(GL_TRIANGLES, 0, numVertices);
```

Rendering with OpenGL

- Render Targets
- Shaders
- Vertex Buffers
- Uniforms
- Textures
- Draws

```
glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);

glUseProgram(myProgram);

glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);

glBindBuffer(GL_UNIFORM_BUFFER, myUniforms);

glBindTexture(GL_TEXTURE_2D, myColorTexture);

glDrawArrays(GL_TRIANGLES, 0, numVertices);
```

Rendering with OpenGL

- Render Targets
- Shaders
- Vertex Buffers
- Uniforms
- Textures
- Draws

```
glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);

glUseProgram(myProgram);

glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);

glBindBuffer(GL_UNIFORM_BUFFER, myUniforms);

glBindTexture(GL_TEXTURE_2D, myColorTexture);

glDrawArrays(GL_TRIANGLES, 0, numVertices);
```

Rendering with OpenGL

- Render Targets
- Shaders
- Vertex Buffers
- Uniforms
- Textures
- Draws

```
glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);

glUseProgram(myProgram);

glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);

glBindBuffer(GL_UNIFORM_BUFFER, myUniforms);

glBindTexture(GL_TEXTURE_2D, myColorTexture);

glDrawArrays(GL_TRIANGLES, 0, numVertices);
```

Rendering with OpenGL

- Render Targets
- Shaders
- Vertex Buffers
- Uniforms
- **Textures**
- Draws

```
glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);

glUseProgram(myProgram);

glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);

glBindBuffer(GL_UNIFORM_BUFFER, myUniforms);

glBindTexture(GL_TEXTURE_2D, myColorTexture);

glDrawArrays(GL_TRIANGLES, 0, numVertices);
```

Rendering with OpenGL

Render Targets

Shaders

Vertex Buffers

Uniforms

Textures

Draws

```
glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);

glUseProgram(myProgram);

glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);

glBindBuffer(GL_UNIFORM_BUFFER, myUniforms);

glBindTexture(GL_TEXTURE_2D, myColorTexture);

glDrawArrays(GL_TRIANGLES, 0, numVertices);
```

Rendering with Metal

- Render Targets
- Shaders
- Vertex Buffers
- Uniforms
- Textures
- Draws

```
encoder = [commandBuffer renderCommandEncoderWithDescriptor:descriptor];

[encoder setPipelineState:myPipeline];

[encoder setVertexBuffer:myVertexData offset:0 atIndex:0];

[encoder setVertexBuffer:myUniforms offset:0 atIndex:1];
[encoder setFragmentBuffer:myUniforms offset:0 atIndex:1];

[encoder setFragmentTexture:myColorTexture atIndex:0];

[encoder drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0
            vertexCount:numVertices];

[encoder endEncoding];
```

Rendering with Metal

- Render Targets
- Shaders
- Vertex Buffers
- Uniforms
- Textures
- Draws

```
encoder = [commandBuffer renderCommandEncoderWithDescriptor:descriptor];

[encoder setPipelineState:myPipeline];

[encoder setVertexBuffer:myVertexData offset:0 atIndex:0];

[encoder setVertexBuffer:myUniforms offset:0 atIndex:1];
[encoder setFragmentBuffer:myUniforms offset:0 atIndex:1];

[encoder setFragmentTexture:myColorTexture atIndex:0];

[encoder drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0
            vertexCount:numVertices];

[encoder endEncoding];
```

Rendering with Metal

- Render Targets
- **Shaders**
- Vertex Buffers
- Uniforms
- Textures
- Draws

```
encoder = [commandBuffer renderCommandEncoderWithDescriptor:descriptor];

[encoder setPipelineState:myPipeline];

[encoder setVertexBuffer:myVertexData offset:0 atIndex:0];

[encoder setVertexBuffer:myUniforms offset:0 atIndex:1];
[encoder setFragmentBuffer:myUniforms offset:0 atIndex:1];

[encoder setFragmentTexture:myColorTexture atIndex:0];

[encoder drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0
            vertexCount:numVertices];

[encoder endEncoding];
```

Rendering with Metal

- Render Targets
- Shaders
- **Vertex Buffers**
- Uniforms
- Textures
- Draws

```
encoder = [commandBuffer renderCommandEncoderWithDescriptor:descriptor];

[encoder setPipelineState:myPipeline];

[encoder setVertexBuffer:myVertexData offset:0 atIndex:0];

[encoder setVertexBuffer:myUniforms offset:0 atIndex:1];
[encoder setFragmentBuffer:myUniforms offset:0 atIndex:1];

[encoder setFragmentTexture:myColorTexture atIndex:0];

[encoder drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0
            vertexCount:numVertices];

[encoder endEncoding];
```

Rendering with Metal

- Render Targets
- Shaders
- Vertex Buffers
- **Uniforms**
- Textures
- Draws

```
encoder = [commandBuffer renderCommandEncoderWithDescriptor:descriptor];

[encoder setPipelineState:myPipeline];

[encoder setVertexBuffer:myVertexData offset:0 atIndex:0];

[encoder setVertexBuffer:myUniforms offset:0 atIndex:1];
[encoder setFragmentBuffer:myUniforms offset:0 atIndex:1];

[encoder setFragmentTexture:myColorTexture atIndex:0];

[encoder drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0
            vertexCount:numVertices];

[encoder endEncoding];
```

Rendering with Metal

- Render Targets
- Shaders
- Vertex Buffers
- Uniforms
- **Textures**
- Draws

```
encoder = [commandBuffer renderCommandEncoderWithDescriptor:descriptor];

[encoder setPipelineState:myPipeline];

[encoder setVertexBuffer:myVertexData offset:0 atIndex:0];

[encoder setVertexBuffer:myUniforms offset:0 atIndex:1];
[encoder setFragmentBuffer:myUniforms offset:0 atIndex:1];

[encoder setFragmentTexture:myColorTexture atIndex:0];

[encoder drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0
            vertexCount:numVertices];

[encoder endEncoding];
```

Rendering with Metal

Render Targets

Shaders

Vertex Buffers

Uniforms

Textures

Draws

```
encoder = [commandBuffer renderCommandEncoderWithDescriptor:descriptor];

[encoder setPipelineState:myPipeline];

[encoder setVertexBuffer:myVertexData offset:0 atIndex:0];

[encoder setVertexBuffer:myUniforms offset:0 atIndex:1];
[encoder setFragmentBuffer:myUniforms offset:0 atIndex:1];

[encoder setFragmentTexture:myColorTexture atIndex:0];

[encoder drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0
            vertexCount:numVertices];

[encoder endEncoding];
```

Rendering with Metal

- Render Targets
- Shaders
- Vertex Buffers
- Uniforms
- Textures
- Draws

```
encoder = [commandBuffer renderCommandEncoderWithDescriptor:descriptor];

[encoder setPipelineState:myPipeline];

[encoder setVertexBuffer:myVertexData offset:0 atIndex:0];

[encoder setVertexBuffer:myUniforms offset:0 atIndex:1];
[encoder setFragmentBuffer:myUniforms offset:0 atIndex:1];

[encoder setFragmentTexture:myColorTexture atIndex:0];

[encoder drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0
            vertexCount:numVertices];

[encoder endEncoding];
```

Build	Initialize	Render
Shaders	Devices and Queues	Command Buffers
	Render Objects	Resource Updates
		Render Encoders
		Display

Application Renderer

Metal API

Command Encoder

Command Buffer

Textures

Buffers

Pipelines

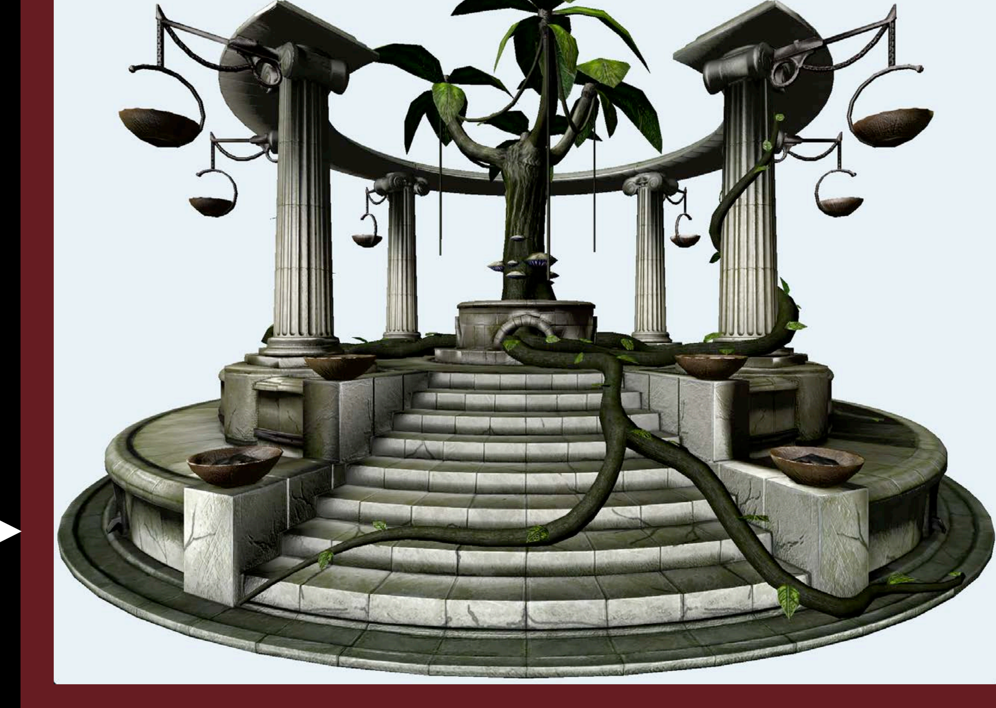
Render Objects

Command Queue

Device

GPU

Display

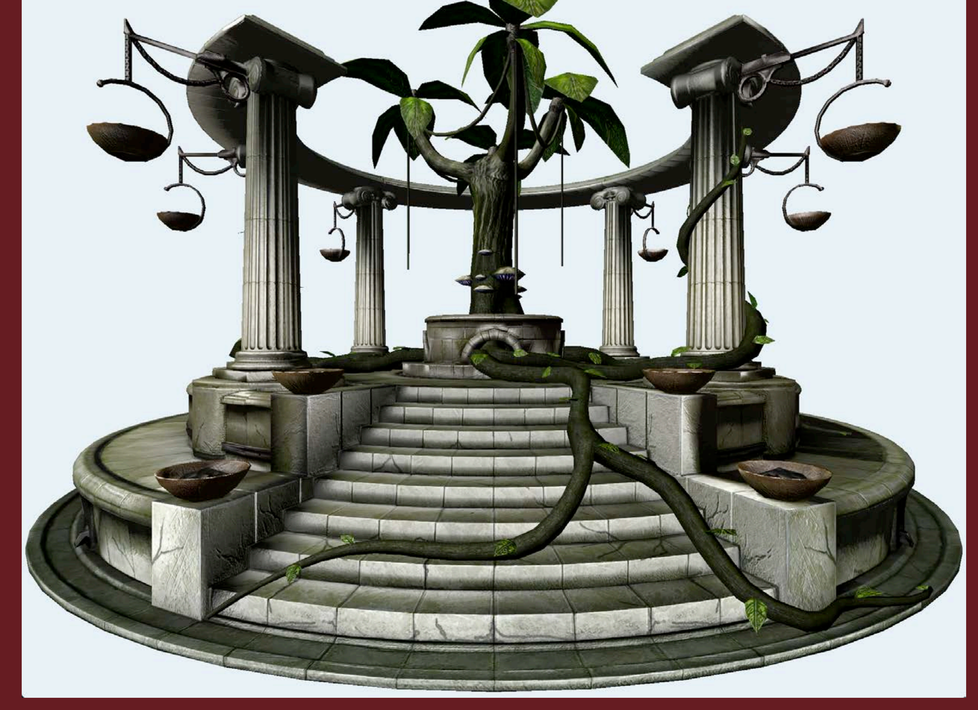


Application Renderer

Metal API

Command Encoder

Command Buffer



Textures

Buffers

Pipelines

Render Objects

Command Queue

Device

GPU

Display

Drawables and presentation

Drawables—Textures for on screen display

Each frame MTKView provides

- Drawable texture
- Render pass descriptor setup with the drawable

Render to drawables like any other texture

Present drawable when done rendering

```
// Render offscreen passes
...

// Acquire a render pass descriptor generated from the drawable's texture
MTLRenderPassDescriptor* renderPassDescriptor = view.currentRenderPassDescriptor;

// encode your on-screen render passes
id <MTLRenderCommandEncoder> renderCommandEncoder =
[commandBuffer renderCommandEncoderWithDescriptor:renderPassDescriptor];

// Encode render commands
...

[renderCommandEncoder endEncoding];

// Register the drawable presentation
[commandBuffer presentDrawable:view.currentDrawable];
[commandBuffer commit];
```

```
// Render offscreen passes
...

// Acquire a render pass descriptor generated from the drawable's texture
MTLRenderPassDescriptor* renderPassDescriptor = view.currentRenderPassDescriptor;

// encode your on-screen render passes
id <MTLRenderCommandEncoder> renderCommandEncoder =
[commandBuffer renderCommandEncoderWithDescriptor:renderPassDescriptor];

// Encode render commands
...

[renderCommandEncoder endEncoding];

// Register the drawable presentation
[commandBuffer presentDrawable:view.currentDrawable];
[commandBuffer commit];
```

```
// Render offscreen passes
...

// Acquire a render pass descriptor generated from the drawable's texture
MTLRenderPassDescriptor* renderPassDescriptor = view.currentRenderPassDescriptor;

// encode your on-screen render passes
id <MTLRenderCommandEncoder> renderCommandEncoder =
[commandBuffer renderCommandEncoderWithDescriptor:renderPassDescriptor];

// Encode render commands
...

[renderCommandEncoder endEncoding];

// Register the drawable presentation
[commandBuffer presentDrawable:view.currentDrawable];
[commandBuffer commit];
```

Build

Shaders

Initialize

Devices and Queues

Render Objects

Render

Command Buffers

Resource Updates

Render Encoders

Display

Incrementally Porting

Create shared Metal/OpenGL textures using IOSurface or CVPixelBuffer

- Render to texture on one API and read in the other

Can enable mixed Metal/OpenGL applications

Sample code available

Going Further

Multithreading

Metal is designed to facilitate Multithreading

- Consider multithreading if application is CPU bound
- Encode multiple command buffers simultaneously
- Split single render pass using MTLParalllelCommandEncoder

Going Further

Staying on the GPU

Metal natively supports compute

Performance benefits

- Reduces CPU utilization
- Reduces GPU-CPU synchronization points
- Free’s data bandwidth to the GPU

New algorithms possible

- Particle systems, physics, object culling

More Metal Features

More Metal Features

Sharable Textures
Tile shaders
Tessellation
Resource Heaps

Indirect Command Buffers
Raster Order Groups
Programmable Sample Positions

Events
Layered Rendering
Argument Buffers
Image Blocks

Compute
Indirect Dispatch

SIMD Group operators
Typed Buffers
Framebuffer Fetch
Metal Performance Shaders

Function Specialization
Multi Viewport Rendering
Texture Arrays

Array of Samplers
Wide Color
Resource Views
Memoryless Render Targets

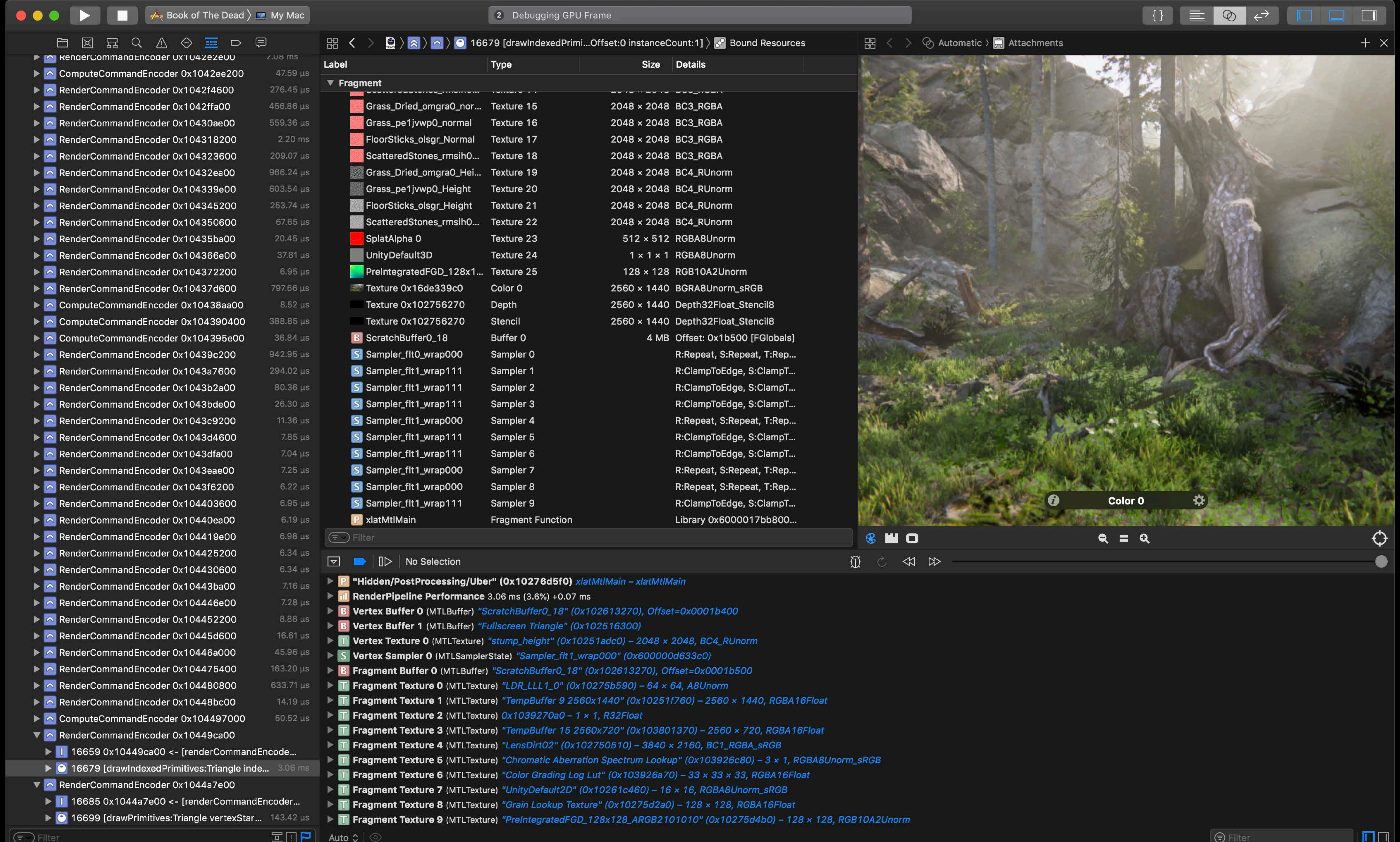
Developer Tools

Debug and optimize your applications

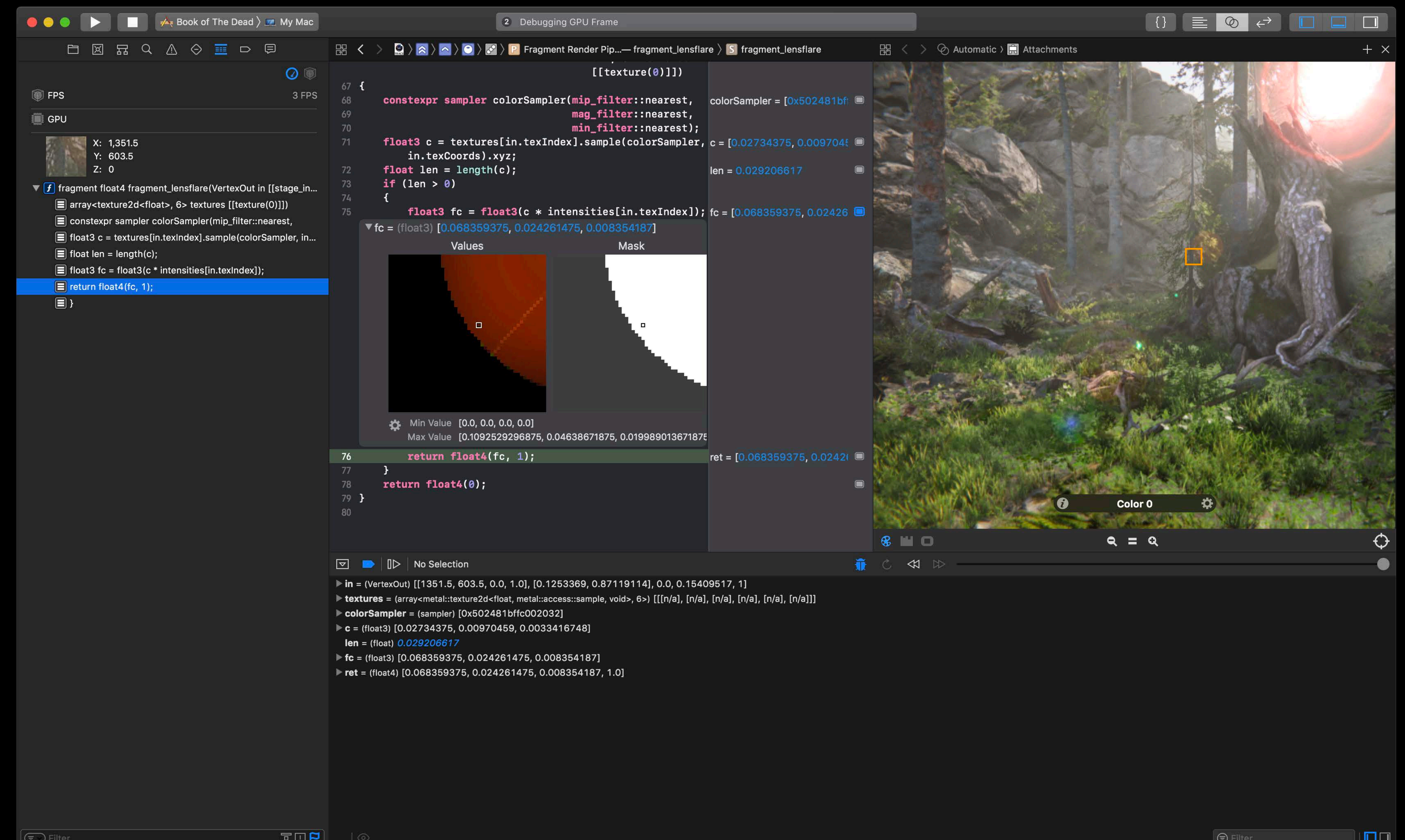
Xcode contains an advanced set of GPU tools

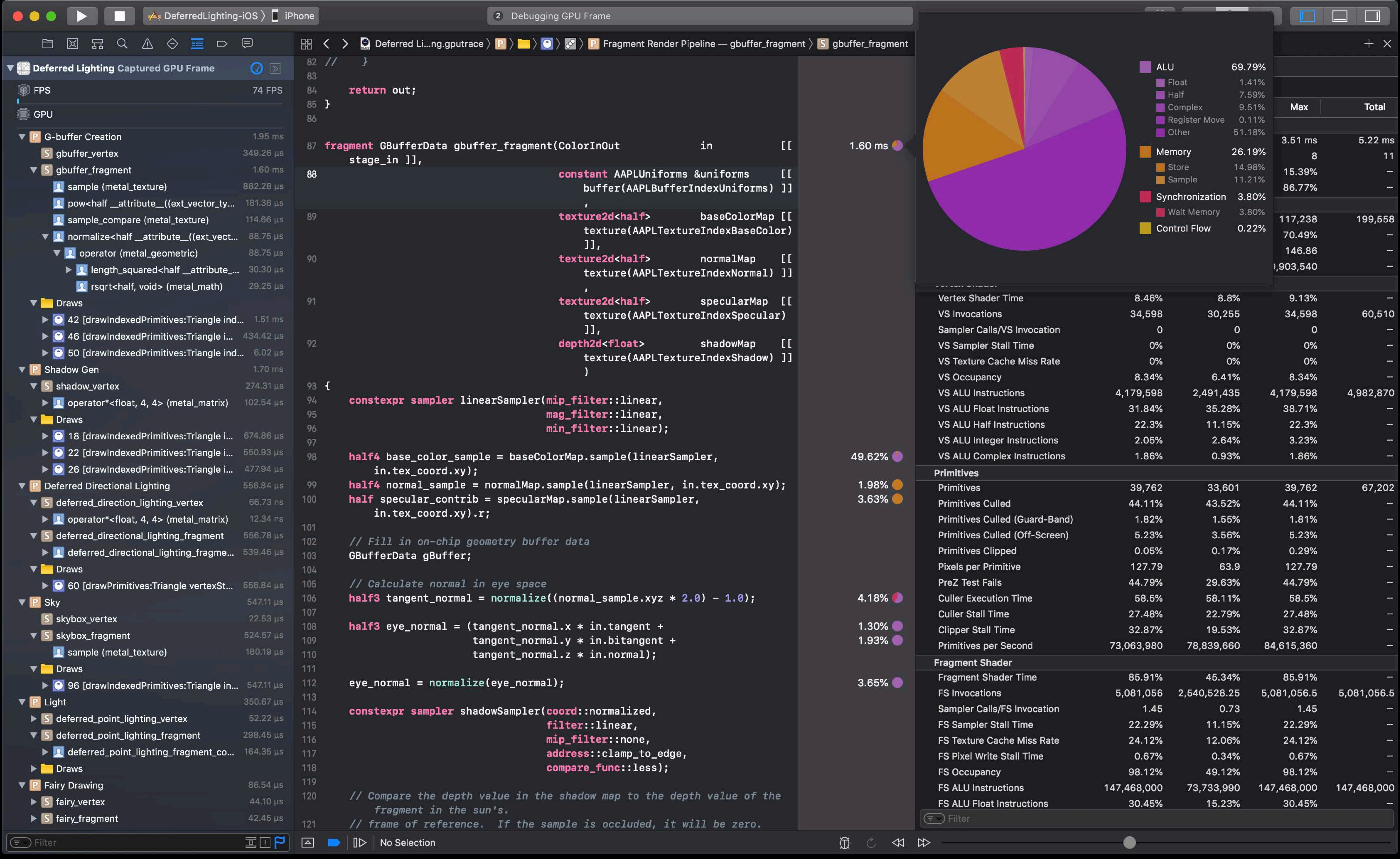
Enable Metal's API validation layer

- On by default when target run from Xcode

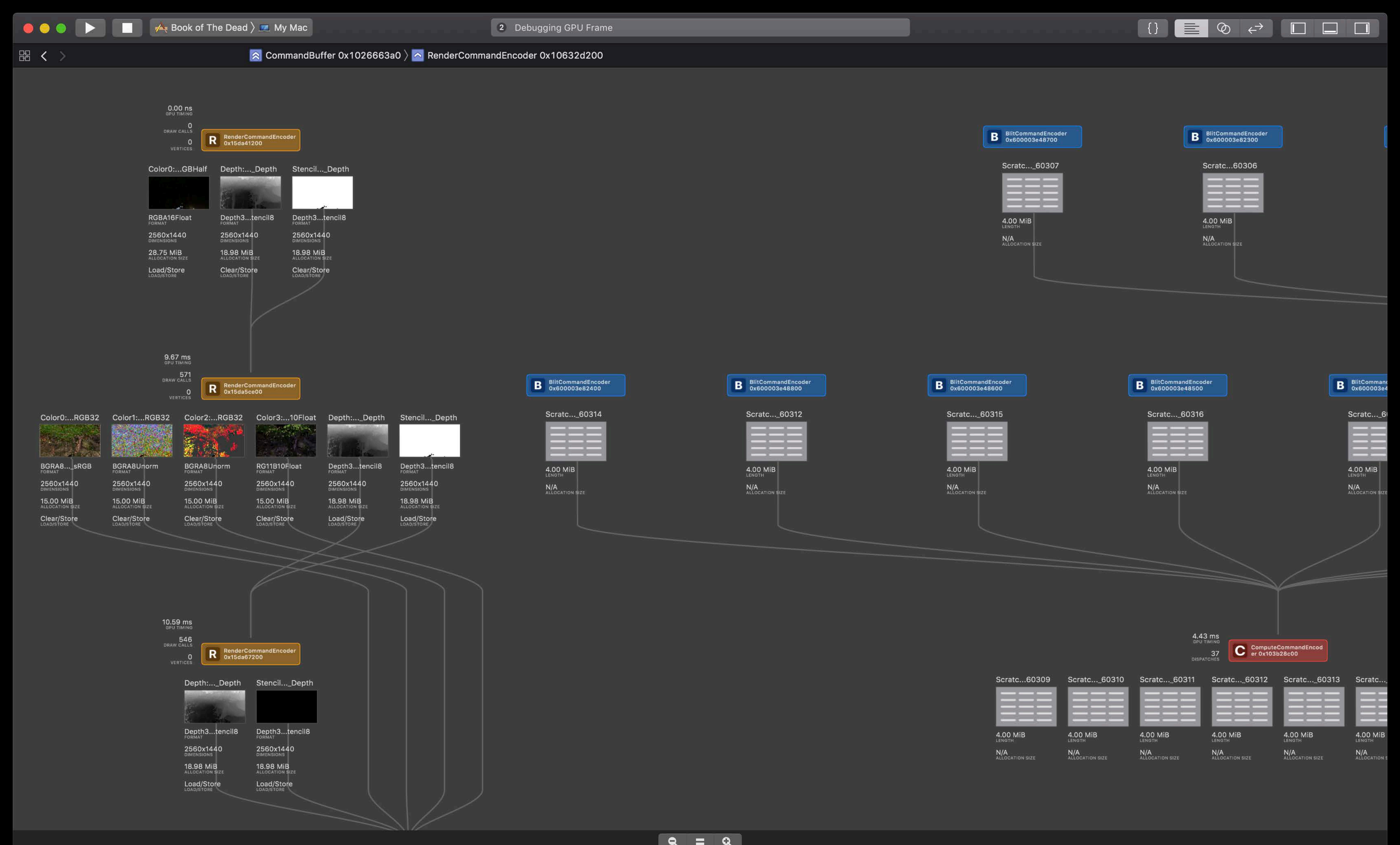


Book of The Dead
My Mac
Debugging GPU Frame
16679 [drawIndexedPrimi...Offset:0 instanceCount:1]
Bound Resources
Automatic
Attachments
Label
Type
Size
Details
Fragment
Grass_Dried_omgra0_nor... Texture 15 2048 × 2048 BC3_RGBA
Grass_pe1jvwp0_normal Texture 16 2048 × 2048 BC3_RGBA
FloorSticks_olsgr_Normal Texture 17 2048 × 2048 BC3_RGBA
ScatteredStones_rmsih0... Texture 18 2048 × 2048 BC3_RGBA
Grass_Dried_omgra0_Hei... Texture 19 2048 × 2048 BC4_RUnorm
Grass_pe1jvwp0_Height Texture 20 2048 × 2048 BC4_RUnorm
FloorSticks_olsgr_Height Texture 21 2048 × 2048 BC4_RUnorm
ScatteredStones_rmsih0... Texture 22 2048 × 2048 BC4_RUnorm
SplatAlpha 0 Texture 23 512 × 512 RGBA8Unorm
UnityDefault3D Texture 24 1 × 1 × 1 RGBA8Unorm
PreIntegratedFGD_128x1... Texture 25 128 × 128 RGB10A2Unorm
Texture 0x16de339c0 Color 0 2560 × 1440 BGRA8Unorm_sRGB
Texture 0x102756270 Depth 2560 × 1440 Depth32Float_Stencil8
Texture 0x102756270 Stencil 2560 × 1440 Depth32Float_Stencil8
ScratchBuffer0_18 Buffer 0 4 MB Offset: 0x1b500 [FGlobals]
Sampler_flt0_wrap000 Sampler 0 R:Repeat, S:Repeat, T:Rep...
Sampler_flt1_wrap111 Sampler 1 R:ClampToEdge, S:ClampT...
Sampler_flt1_wrap111 Sampler 2 R:ClampToEdge, S:ClampT...
Sampler_flt1_wrap111 Sampler 3 R:ClampToEdge, S:ClampT...
Sampler_flt1_wrap000 Sampler 4 R:Repeat, S:Repeat, T:Rep...
Sampler_flt1_wrap111 Sampler 5 R:ClampToEdge, S:ClampT...
Sampler_flt1_wrap111 Sampler 6 R:ClampToEdge, S:ClampT...
Sampler_flt1_wrap000 Sampler 7 R:Repeat, S:Repeat, T:Rep...
Sampler_flt1_wrap000 Sampler 8 R:Repeat, S:Repeat, T:Rep...
Sampler_flt1_wrap111 Sampler 9 R:ClampToEdge, S:ClampT...
xlatMtlMain Fragment Function Library 0x6000017bb800...
Filter
Color 0
No Selection
"Hidden/PostProcessing/Uber" (0x10276d5f0) xlatMtlMain – xlatMtlMain
RenderPipeline Performance 3.06 ms (3.6%) +0.07 ms
Vertex Buffer 0 (MTLBuffer) "ScratchBuffer0_18" (0x102613270), Offset=0x0001b400
Vertex Buffer 1 (MTLBuffer) "Fullscreen Triangle" (0x102516300)
Vertex Texture 0 (MTLTexture) "stump_height" (0x10251adc0) – 2048 × 2048, BC4_RUnorm
Vertex Sampler 0 (MTLSamplerState) "Sampler_flt1_wrap000" (0x600000d633c0)
Fragment Buffer 0 (MTLBuffer) "ScratchBuffer0_18" (0x102613270), Offset=0x0001b500
Fragment Texture 0 (MTLTexture) "LDR_LLL1_0" (0x10275b590) – 64 × 64, A8Unorm
Fragment Texture 1 (MTLTexture) "TempBuffer 9 2560x1440" (0x10251f760) – 2560 × 1440, RGBA16Float
Fragment Texture 2 (MTLTexture) 0x1039270a0 – 1 × 1, R32Float
Fragment Texture 3 (MTLTexture) "TempBuffer 15 2560x720" (0x103801370) – 2560 × 720, RGBA16Float
Fragment Texture 4 (MTLTexture) "LensDirt02" (0x102750510) – 3840 × 2160, BC1_RGBA_sRGB
Fragment Texture 5 (MTLTexture) "Chromatic Aberration Spectrum Lookup" (0x103926c80) – 3 × 1, RGBA8Unorm_sRGB
Fragment Texture 6 (MTLTexture) "Color Grading Log Lut" (0x103926a70) – 33 × 33 × 33, RGBA16Float
Fragment Texture 7 (MTLTexture) "UnityDefault2D" (0x10261c460) – 16 × 16, RGBA8Unorm_sRGB
Fragment Texture 8 (MTLTexture) "Grain Lookup Texture" (0x10275d2a0) – 128 × 128, RGBA16Float
Fragment Texture 9 (MTLTexture) "PreIntegratedFGD_128x128_ARGB2101010" (0x10275d4b0) – 128 × 128, RGB10A2Unorm
ComputeCommandEncoder 0x1042ee200 47.59 µs
RenderCommandEncoder 0x1042f4600 276.45 µs
RenderCommandEncoder 0x1042ffa00 456.86 µs
RenderCommandEncoder 0x10430ae00 559.36 µs
RenderCommandEncoder 0x104318200 2.20 ms
RenderCommandEncoder 0x104323600 209.07 µs
RenderCommandEncoder 0x10432ea00 966.24 µs
RenderCommandEncoder 0x104339e00 603.54 µs
RenderCommandEncoder 0x104345200 253.74 µs
RenderCommandEncoder 0x104350600 67.65 µs
RenderCommandEncoder 0x10435ba00 20.45 µs
RenderCommandEncoder 0x104366e00 37.81 µs
RenderCommandEncoder 0x104372200 6.95 µs
RenderCommandEncoder 0x10437d600 797.66 µs
ComputeCommandEncoder 0x10438aa00 8.52 µs
ComputeCommandEncoder 0x104390400 388.85 µs
ComputeCommandEncoder 0x104395e00 36.84 µs
RenderCommandEncoder 0x10439c200 942.95 µs
RenderCommandEncoder 0x1043a7600 294.02 µs
RenderCommandEncoder 0x1043b2a00 80.36 µs
RenderCommandEncoder 0x1043bde00 26.30 µs
RenderCommandEncoder 0x1043c9200 11.36 µs
RenderCommandEncoder 0x1043d4600 7.85 µs
RenderCommandEncoder 0x1043dfa00 7.04 µs
RenderCommandEncoder 0x1043eae00 7.25 µs
RenderCommandEncoder 0x1043f6200 6.22 µs
RenderCommandEncoder 0x104403600 6.95 µs
RenderCommandEncoder 0x10440ea00 6.19 µs
RenderCommandEncoder 0x104419e00 6.98 µs
RenderCommandEncoder 0x104425200 6.34 µs
RenderCommandEncoder 0x104430600 6.34 µs
RenderCommandEncoder 0x10443ba00 7.16 µs
RenderCommandEncoder 0x104446e00 7.28 µs
RenderCommandEncoder 0x104452200 8.88 µs
RenderCommandEncoder 0x10445d600 16.61 µs
RenderCommandEncoder 0x10446a000 45.96 µs
RenderCommandEncoder 0x104475400 163.20 µs
RenderCommandEncoder 0x104480800 633.71 µs
RenderCommandEncoder 0x10448bc00 14.19 µs
ComputeCommandEncoder 0x104497000 50.52 µs
RenderCommandEncoder 0x10449ca00
16659 0x10449ca00 <- [renderCommandEncode...
16679 [drawIndexedPrimitives:Triangle inde... 3.06 ms
RenderCommandEncoder 0x1044a7e00
16685 0x1044a7e00 <- [renderCommandEncoder...
16699 [drawPrimitives:Triangle vertexStar... 143.42 µs
Filter
Auto
Filter

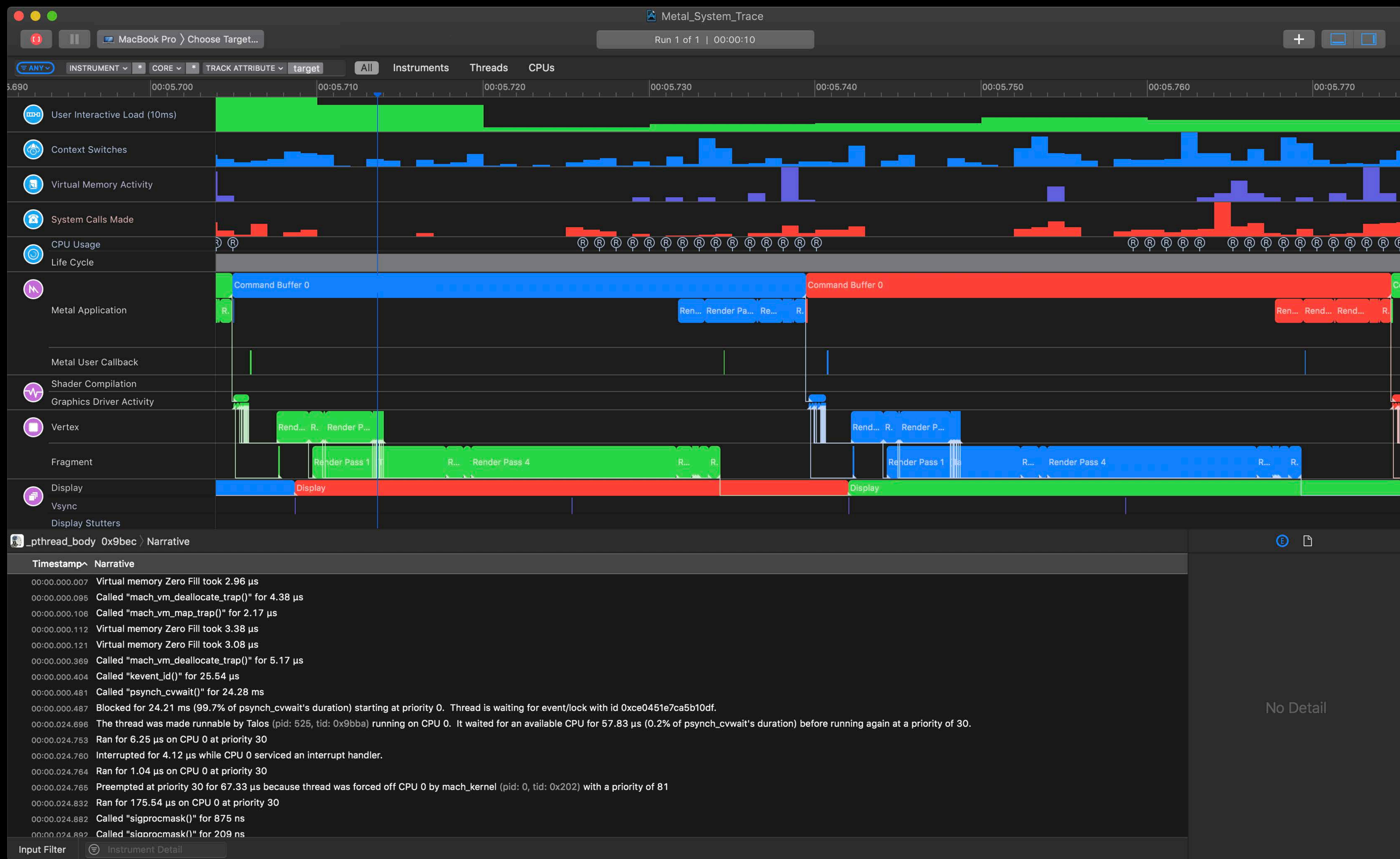

Book of The Dead
My Mac
Debugging GPU Frame
Fragment Render Pip...— fragment_lensflare
fragment_lensflare
Automatic
Attachments
FPS
3 FPS
GPU
X: 1,351.5
Y: 603.5
Z: 0
fragment float4 fragment_lensflare(VertexOut in [[stage_in...
array<texture2d<float>, 6> textures [[texture(0)]])
constexpr sampler colorSampler(mip_filter::nearest,
float3 c = textures[in.texIndex].sample(colorSampler, in...
float len = length(c);
float3 fc = float3(c * intensities[in.texIndex]);
return float4(fc, 1);
}
[[texture(0)]])
{
constexpr sampler colorSampler(mip_filter::nearest,
mag_filter::nearest,
min_filter::nearest);
float3 c = textures[in.texIndex].sample(colorSampler,
in.texCoords).xyz;
float len = length(c);
if (len > 0)
{
float3 fc = float3(c * intensities[in.texIndex]);
fc = (float3) [0.068359375, 0.024261475, 0.008354187]
Values
Mask
Min Value [0.0, 0.0, 0.0, 0.0]
Max Value [0.1092529296875, 0.04638671875, 0.01998901367187
return float4(fc, 1);
}
return float4(0);
}
colorSampler = [0x502481bf
c = [0.02734375, 0.009704
len = 0.029206617
fc = [0.068359375, 0.02426
ret = [0.068359375, 0.0242
Color 0
No Selection
in = (VertexOut) [[1351.5, 603.5, 0.0, 1.0], [0.1253369, 0.87119114], 0.0, 0.15409517, 1]
textures = (array<metal::texture2d<float, metal::access::sample, void>, 6>) [[[n/a], [n/a], [n/a], [n/a], [n/a], [n/a]]]
colorSampler = (sampler) [0x502481bffc002032]
c = (float3) [0.02734375, 0.00970459, 0.0033416748]
len = (float) 0.029206617
fc = (float3) [0.068359375, 0.024261475, 0.008354187]
ret = (float4) [0.068359375, 0.024261475, 0.008354187, 1.0]
Filter
Filter



DeferredLighting-iOS
iPhone
Debugging GPU Frame
Deferred Lighting Captured GPU Frame
FPS 74 FPS
GPU
G-buffer Creation 1.95 ms
gbuffer_vertex 349.26 µs
gbuffer_fragment 1.60 ms
sample (metal_texture) 882.28 µs
sample_compare (metal_texture) 114.66 µs
Draws
Shadow Gen 1.70 ms
shadow_vertex 274.31 µs
Deferred Directional Lighting 556.84 µs
deferred_direction_lighting_vertex 66.73 ns
deferred_directional_lighting_fragment 556.78 µs
Sky 547.11 µs
skybox_vertex 22.53 µs
skybox_fragment 524.57 µs
sample (metal_texture) 180.19 µs
Light 350.67 µs
deferred_point_lighting_vertex 52.22 µs
deferred_point_lighting_fragment 298.45 µs
Fairy Drawing 86.54 µs
fairy_vertex 44.10 µs
fairy_fragment 42.45 µs
Filter
Deferred Li...ng.gputrace
Fragment Render Pipeline — gbuffer_fragment
gbuffer_fragment
fragment GBufferData gbuffer_fragment(ColorInOut in [[stage_in]],
constant AAPLUniforms &uniforms [[buffer(AAPLBufferIndexUniforms)]],
texture2d<half> baseColorMap [[texture(AAPLTextureIndexBaseColor)]],
texture2d<half> normalMap [[texture(AAPLTextureIndexNormal)]],
texture2d<half> specularMap [[texture(AAPLTextureIndexSpecular)]],
depth2d<float> shadowMap [[texture(AAPLTextureIndexShadow)]])
constexpr sampler linearSampler(mip_filter::linear, mag_filter::linear, min_filter::linear);
half4 base_color_sample = baseColorMap.sample(linearSampler, in.tex_coord.xy);
half4 normal_sample = normalMap.sample(linearSampler, in.tex_coord.xy);
half specular_contrib = specularMap.sample(linearSampler, in.tex_coord.xy).r;
// Fill in on-chip geometry buffer data
GBufferData gBuffer;
// Calculate normal in eye space
half3 tangent_normal = normalize((normal_sample.xyz * 2.0) - 1.0);
half3 eye_normal = (tangent_normal.x * in.tangent + tangent_normal.y * in.bitangent + tangent_normal.z * in.normal);
eye_normal = normalize(eye_normal);
constexpr sampler shadowSampler(coord::normalized, filter::linear, mip_filter::none, address::clamp_to_edge, compare_func::less);
// Compare the depth value in the shadow map to the depth value of the fragment in the sun's.
// frame of reference. If the sample is occluded, it will be zero.
1.60 ms
49.62%
1.98%
3.63%
4.18%
1.30%
1.93%
3.65%
ALU 69.79%
Float 1.41%
Half 7.59%
Complex 9.51%
Register Move 0.11%
Other 51.18%
Memory 26.19%
Store 14.98%
Sample 11.21%
Synchronization 3.80%
Wait Memory 3.80%
Control Flow 0.22%
Max
Total
Vertex Shader Time 8.46% 8.8% 9.13%
VS Invocations 34,598 30,255 34,598 60,510
VS ALU Instructions 4,179,598 2,491,435 4,179,598 4,982,870
Primitives
Primitives 39,762 33,601 39,762 67,202
Primitives Culled 44.11% 43.52% 44.11%
Primitives per Second 73,063,980 78,839,660 84,615,360
Fragment Shader
Fragment Shader Time 85.91% 45.34% 85.91%
FS Invocations 5,081,056 2,540,528.25 5,081,056.5 5,081,056.5
FS Occupancy 98.12% 49.12% 98.12%
FS ALU Instructions 147,468,000 73,733,990 147,468,000 147,468,000
FS ALU Float Instructions 30.45% 15.23% 30.45%
No Selection



Book of The Dead
My Mac
Debugging GPU Frame
CommandBuffer 0x1026663a0
RenderCommandEncoder 0x10632d200
0.00 ns
RenderCommandEncoder 0x15da41200
Color0:...GBHalf
Depth:..._Depth
Stencil..._Depth
RGBA16Float
Depth3...tencil8
2560x1440
28.75 MiB
18.98 MiB
Load/Store
Clear/Store
9.67 ms
571
RenderCommandEncoder 0x15da5ce00
Color0:...RGB32
Color1:...RGB32
Color2:...RGB32
Color3:...10Float
BGRA8..._sRGB
BGRA8Unorm
RG11B10Float
15.00 MiB
10.59 ms
546
RenderCommandEncoder 0x15da67200
BlitCommandEncoder 0x600003e48700
BlitCommandEncoder 0x600003e82300
Scratc..._60307
Scratc..._60306
BlitCommandEncoder 0x600003e82400
BlitCommandEncoder 0x600003e48800
BlitCommandEncoder 0x600003e48600
BlitCommandEncoder 0x600003e48500
Scratc..._60314
Scratc..._60312
Scratc..._60315
Scratc..._60316
4.00 MiB
N/A
4.43 ms
37
ComputeCommandEncoder 0x103b28c00
Scratc...60309
Scratc..._60310
Scratc..._60311
Scratc..._60312
Scratc..._60313



Metal_System_Trace
MacBook Pro 〉 Choose Target...
Run 1 of 1 | 00:00:10
ANY
INSTRUMENT
CORE
TRACK ATTRIBUTE
target
All
Instruments
Threads
CPUs
00:05.700
00:05.710
00:05.720
00:05.730
00:05.740
00:05.750
00:05.760
00:05.770
User Interactive Load (10ms)
Context Switches
Virtual Memory Activity
System Calls Made
CPU Usage
Life Cycle
Metal Application
Command Buffer 0
Command Buffer 0
Metal User Callback
Shader Compilation
Graphics Driver Activity
Vertex
Fragment
Render Pass 1
Render Pass 4
Render Pass 1
Render Pass 4
Display
Vsync
Display Stutters
Display
Display
_pthread_body 0x9bec 〉 Narrative
Timestamp
Narrative
00:00.000.007 Virtual memory Zero Fill took 2.96 µs
00:00.000.095 Called "mach_vm_deallocate_trap()" for 4.38 µs
00:00.000.106 Called "mach_vm_map_trap()" for 2.17 µs
00:00.000.112 Virtual memory Zero Fill took 3.38 µs
00:00.000.121 Virtual memory Zero Fill took 3.08 µs
00:00.000.369 Called "mach_vm_deallocate_trap()" for 5.17 µs
00:00.000.404 Called "kevent_id()" for 25.54 µs
00:00.000.481 Called "psynch_cvwait()" for 24.28 ms
00:00.000.487 Blocked for 24.21 ms (99.7% of psynch_cvwait's duration) starting at priority 0. Thread is waiting for event/lock with id 0xce0451e7ca5b10df.
00:00.024.696 The thread was made runnable by Talos (pid: 525, tid: 0x9bba) running on CPU 0. It waited for an available CPU for 57.83 µs (0.2% of psynch_cvwait's duration) before running again at a priority of 30.
00:00.024.753 Ran for 6.25 µs on CPU 0 at priority 30
00:00.024.760 Interrupted for 4.12 µs while CPU 0 serviced an interrupt handler.
00:00.024.764 Ran for 1.04 µs on CPU 0 at priority 30
00:00.024.765 Preempted at priority 30 for 67.33 µs because thread was forced off CPU 0 by mach_kernel (pid: 0, tid: 0x202) with a priority of 81
00:00.024.832 Ran for 175.54 µs on CPU 0 at priority 30
00:00.024.882 Called "sigprocmask()" for 875 ns
No Detail
Input Filter
Instrument Detail

Summary

OpenGL and OpenCL are deprecated

Now it’s time to adopt Metal

Incremental port is possible

Full suite of developer tools available

More Information

https://developer.apple.com/wwdc18/604

OpenGL to Metal Porting Lab	Technology Lab 5	Wednesday 12:00PM
Metal Shader Debugging and Profiling	Hall 3	Thursday 3:00PM
Metal Game Performance Optimization	Hall 1	Friday 10:00AM

WWDC18