Graphics and Games

#WWDC14

Working with Metal—Fundamentals

Session 604

Richard Schreyer
GPU Software Engineer

Aaftab Munshi
GPU Software Engineer

Metal Fundamentals

Building a Metal application

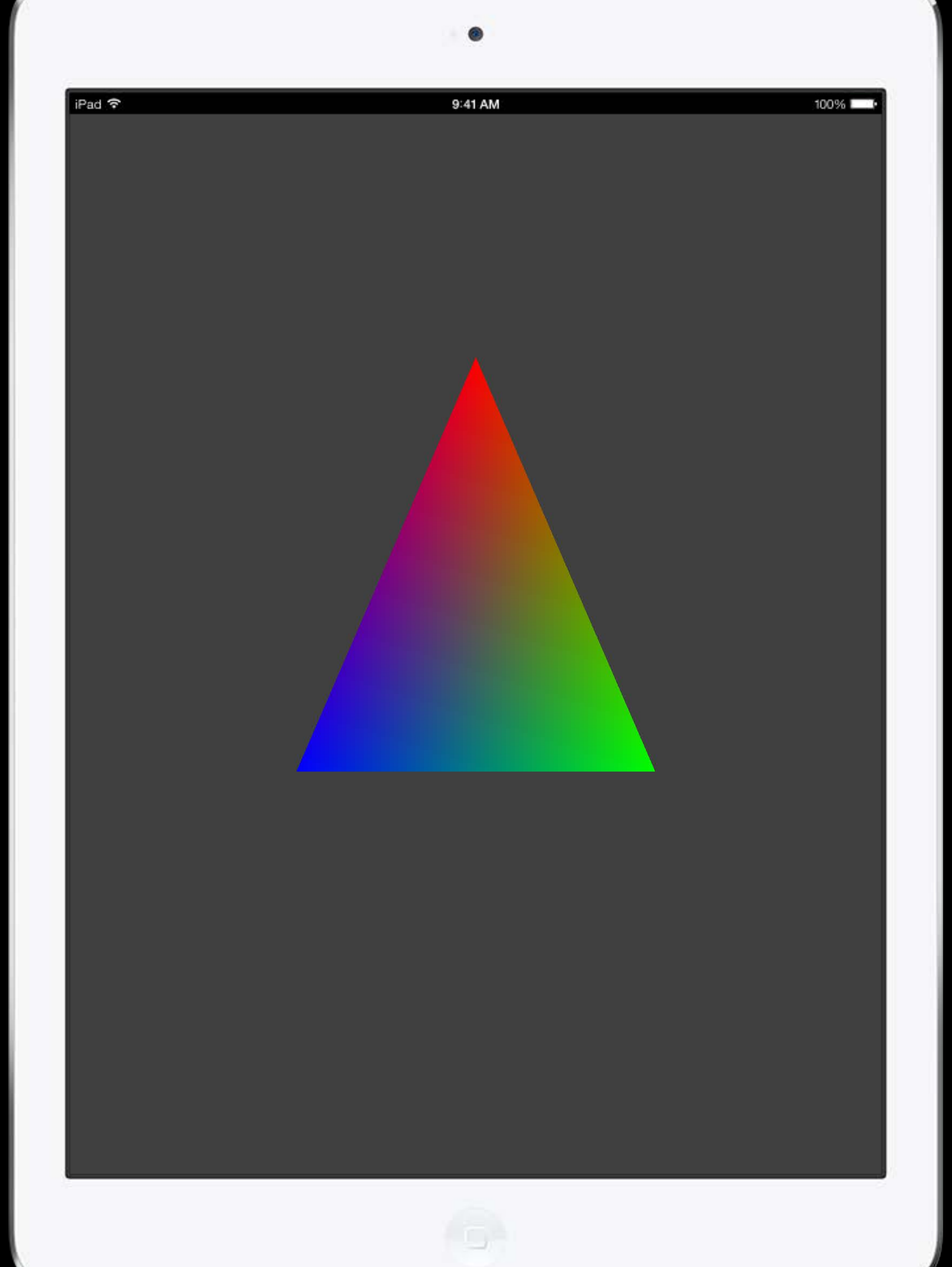
- Initialization
- Drawing
- Uniforms and synchronization

Metal shading language

- Writing shaders in Metal
- Data types in Metal
- Shader inputs, outputs, and matching rules

Building a Metal Application

Richard Schreyer
GPU Software


iPad
9:41 AM
100%

Building a Metal Application

Initialization

1. Get the Device
2. Create a CommandQueue
3. Create Resources (Buffers and Textures)
4. Create RenderPipelines
5. Create a View

Metal Device API

```
@protocol MTLDevice

- (id <MTLCommandQueue>)newCommandQueue…
- (id <MTLBuffer>)newBuffer…
- (id <MTLTexture>)newTexture…
- (id <MTLSampler>)newSamplerState…
- (id <MTLRenderPipelineState>)newRenderPipelineState…
//  and much more

@end
```

Initialization

```
//  Get the device
id <MTLDevice> device = MTLCreateSystemDefaultDevice();
```

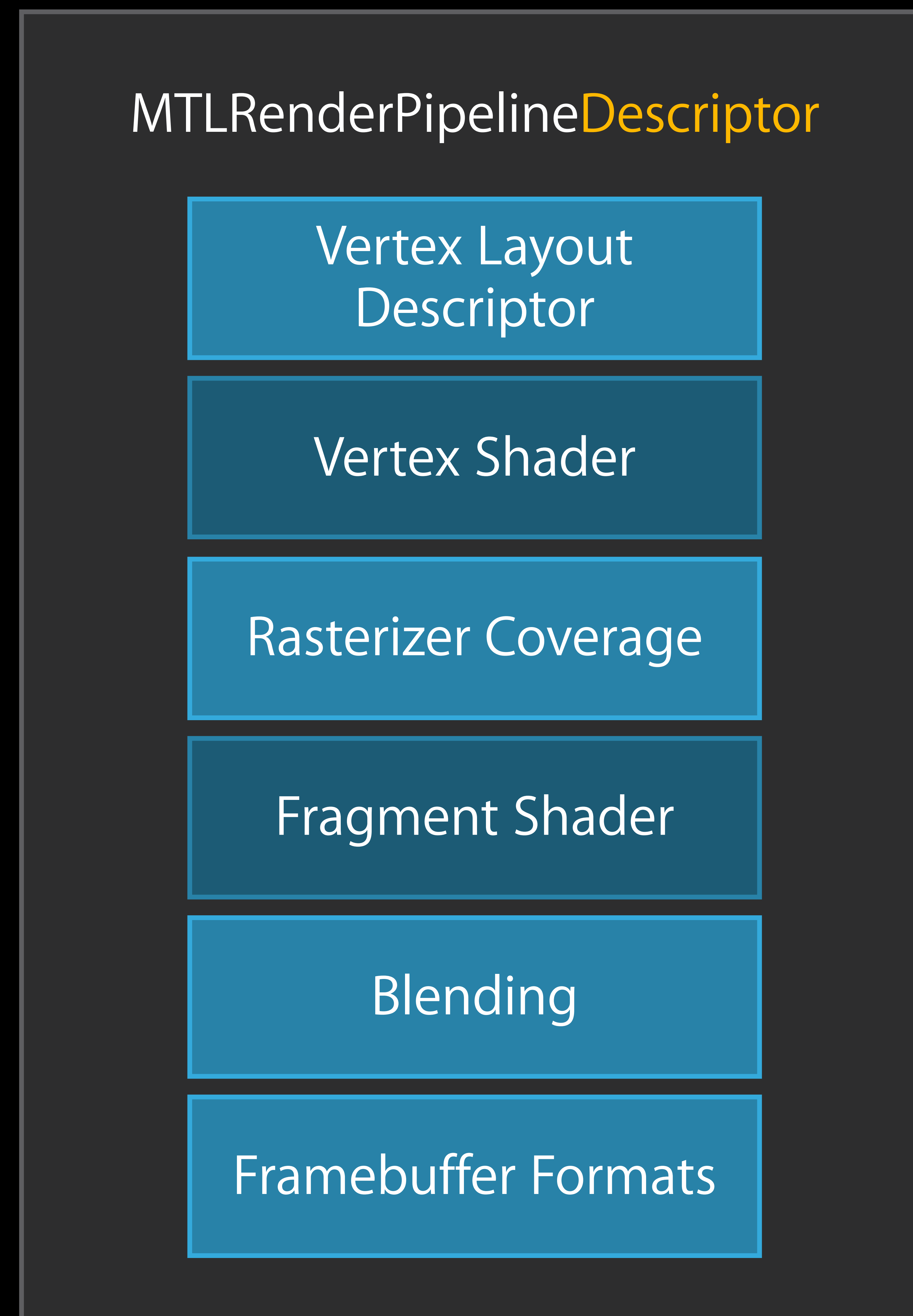
Initialization

```
//  Get the device
id <MTLDevice> device = MTLCreateSystemDefaultDevice();

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

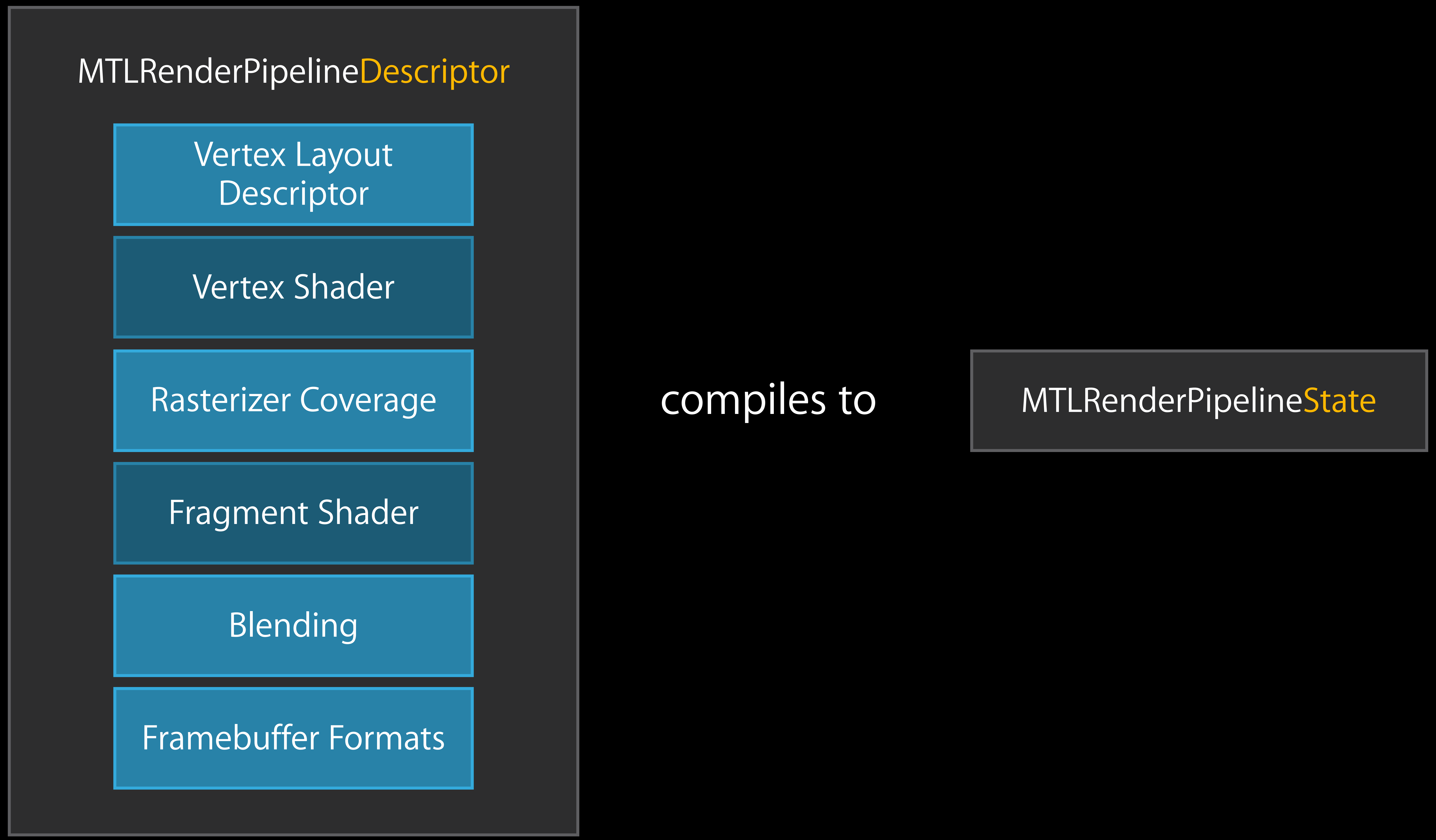
Initialization

```
//  Get the device
id <MTLDevice> device = MTLCreateSystemDefaultDevice();

//  Create a CommandQueue
id <MTLCommandQueue> commandQueue = [device newCommandQueue];

//  Create my Vertex Array
struct Vertex vertexArrayData[3] = { … };
id <MTLBuffer> vertexArray =
        [device newBufferWithBytes: vertexArrayData
                            length: sizeof(vertexArrayData)
                           options: 0];
```

Render Pipeline Descriptors



Render Pipeline Descriptors



Create a RenderPipeline

```
MTLRenderPipelineDescriptor* desc = [MTLRenderPipelineDescriptor new];

//  Set shaders
id <MTLLibrary> library = [device newDefaultLibrary];
desc.vertexFunction = [library newFunctionWithName: @"myVertexShader"];
desc.fragmentFunction = [library newFunctionWithName: @“myFragmentShader"];
```

Create a RenderPipeline

```
MTLRenderPipelineDescriptor* desc = [MTLRenderPipelineDescriptor new];

//  Set shaders
id <MTLLibrary> library = [device newDefaultLibrary];
desc.vertexFunction = [library newFunctionWithName: @"myVertexShader"];
desc.fragmentFunction = [library newFunctionWithName: @“myFragmentShader"];

//  Set framebuffer pixel format
desc.colorAttachments[0].pixelFormat = MTLPixelFormatBGRA8Unorm;
```

Create a RenderPipeline

```
MTLRenderPipelineDescriptor* desc = [MTLRenderPipelineDescriptor new];

//  Set shaders
id <MTLLibrary> library = [device newDefaultLibrary];
desc.vertexFunction = [library newFunctionWithName: @"myVertexShader"];
desc.fragmentFunction = [library newFunctionWithName: @“myFragmentShader"];

//  Set framebuffer pixel format
desc.colorAttachments[0].pixelFormat = MTLPixelFormatBGRA8Unorm;

//  Compile the RenderPipelineState
id <MTLRenderPipelineState> renderPipeline =
        [device newRenderPipelineStateWithDescriptor: desc error: &error];
```

Shader Input and Output

```
struct Vertex {
    float4 position;
    float4 color;
};

struct VertexOut {
    float4 position [[position]];
    float4 color;
};
```

Shader Input and Output

```
struct Vertex {
    float4 position;
    float4 color;
};

struct VertexOut {
    float4 position [[position]];
    float4 color;
};
```

Vertex and Fragment Shaders

```
vertex VertexOut myVertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    unsigned int vid                 [[ vertex_id ]])
{
    VSOut out;
    out.position = vertexArray[vid].position;
    out.color = vertexArray[vid].color;
    return out;
}
```

Vertex and Fragment Shaders

```
vertex VertexOut myVertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    unsigned int vid                 [[ vertex_id ]])
{
    VSOut out;
    out.position = vertexArray[vid].position;
    out.color = vertexArray[vid].color;
    return out;
}
```

Vertex and Fragment Shaders

```
vertex VertexOut myVertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    unsigned int vid                 [[ vertex_id ]])
{
    VSOut out;
    out.position = vertexArray[vid].position;
    out.color = vertexArray[vid].color;
    return out;
}
```

Vertex and Fragment Shaders

```
vertex VertexOut myVertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    unsigned int vid                 [[ vertex_id ]])
{
    VSOut out;
    out.position = vertexArray[vid].position;
    out.color = vertexArray[vid].color;
    return out;
}
```

Vertex and Fragment Shaders

```
vertex VertexOut myVertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    unsigned int vid                 [[ vertex_id ]])
{
    VSOut out;
    out.position = vertexArray[vid].position;
    out.color = vertexArray[vid].color;
    return out;
}

fragment float4 myFragmentShader(
    VertexOut interpolated [[stage_in]])
{
    return interpolated.color;
}
```

Vertex and Fragment Shaders

```
vertex VertexOut myVertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    unsigned int vid                 [[ vertex_id ]])
{
    VSOut out;
    out.position = vertexArray[vid].position;
    out.color = vertexArray[vid].color;
    return out;
}

fragment float4 myFragmentShader(
    VertexOut interpolated [[stage_in]])
{
    return interpolated.color;
}
```

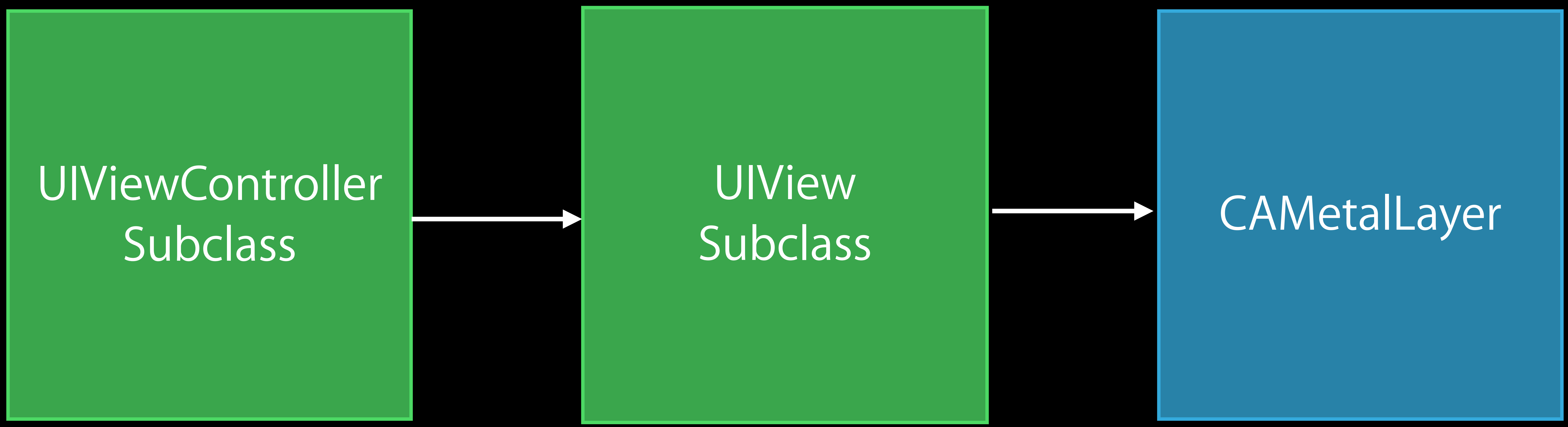
Vertex and Fragment Shaders

```
vertex VertexOut myVertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    unsigned int vid                 [[ vertex_id ]])
{
    VSOut out;
    out.position = vertexArray[vid].position;
    out.color = vertexArray[vid].color;
    return out;
}

fragment float4 myFragmentShader(
    VertexOut interpolated [[stage_in]])
{
    return interpolated.color;
}
```

Vertex and Fragment Shaders

```
vertex VertexOut myVertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    unsigned int vid                 [[ vertex_id ]])
{
    VSOut out;
    out.position = vertexArray[vid].position;
    out.color = vertexArray[vid].color;
    return out;
}

fragment float4 myFragmentShader(
    VertexOut interpolated [[stage_in]])
{
    return interpolated.color;
}
```

Creating a Metal View



Creating a Metal View

```
@interface MyView : UIView
@end

@implementation MyView

+ (id)layerClass {
    return [CAMetalLayer class];
}

@end
```

Building a Metal Application

Initialization

1. Get the Device
2. Create a CommandQueue
3. Create Resources (Buffers and Textures)
4. Create RenderPipelineState
5. Create a View

Building a Metal Application

Initialization

1. Get the Device
2. Create a CommandQueue
3. Create Resources (Buffers and Textures)
4. Create RenderPipelineState
5. Create a View

Building a Metal Application

Initialization

1. Get the Device
2. Create a CommandQueue
3. Create Resources (Buffers and Textures)
4. Create RenderPipelineState
5. Create a View

Building a Metal Application

Initialization

1. Get the Device
2. Create a CommandQueue
3. Create Resources (Buffers and Textures)
4. Create RenderPipelineState
5. Create a View

Building a Metal Application

Initialization

1. Get the Device
2. Create a CommandQueue
3. Create Resources (Buffers and Textures)
4. Create RenderPipelineState
5. Create a View

Building a Metal Application

Initialization

1. Get the Device
2. Create a CommandQueue
3. Create Resources (Buffers and Textures)
4. Create RenderPipelineState
5. Create a View

Building a Metal Application

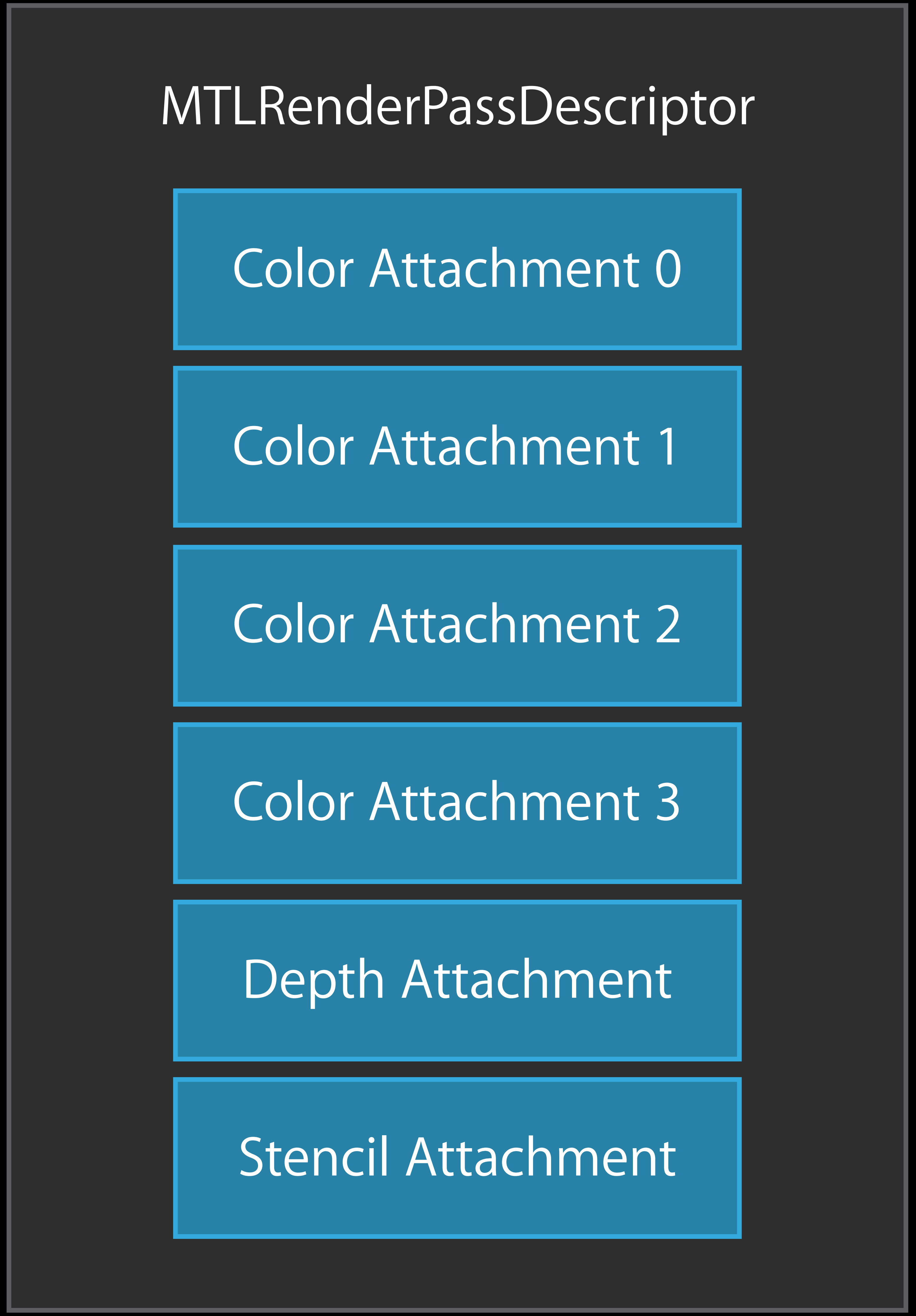
Drawing

1. Get a command buffer
2. Start a Render Pass
3. Draw
4. Commit the command buffer

Get a Command Buffer

```
//  Get an available CommandBuffer
commandBuffer = [queue commandBuffer];
```

Render Pass Configuration



Render Pass Configuration

```
//  Get this frame’s target drawable
drawable = [metalLayer nextDrawable];
```

Render Pass Configuration

```
//  Get this frame's target drawable
drawable = [metalLayer nextDrawable];

//  Configure the Color0 Attachment
renderDesc = [MTLRenderPassDescriptor new];
renderDesc.colorAttachments[0].texture = drawable.texture;
renderDesc.colorAttachments[0].loadAction = MTLLoadActionClear;
renderDesc.colorAttachments[0].clearValue = MTLClearValueMakeColor(…);
```

Render Pass Configuration

```
//  Get this frame’s target drawable
drawable = [metalLayer nextDrawable];

//  Configure the Color0 Attachment
renderDesc = [MTLRenderPassDescriptor new];
renderDesc.colorAttachments[0].texture = drawable.texture;
renderDesc.colorAttachments[0].loadAction = MTLLoadActionClear;
renderDesc.colorAttachments[0].clearValue = MTLClearValueMakeColor(…);

//  Start a Render command
id <MTLRenderCommandEncoder> render =
    [commandBuffer renderCommandEncoderWithDescriptor: renderDesc];
```

Drawing a Triangle

```
render = [commandBuffer renderCommandEncoderWithDescriptor: renderDesc];
[render setRenderPipelineState: renderPipeline];
[render setVertexBuffer: vertexArray offset: 0 atIndex: 0];
[render drawPrimitives: MTLPrimitiveTypeTriangle vertexStart:0 vertexCount:3];
[render endEncoding];
```

Committing a CommandBuffer

Committing a CommandBuffer

```
//  Tell CoreAnimation when to present this drawable
[commandBuffer addPresent: drawable];
```

Committing a CommandBuffer

```
//  Tell CoreAnimation when to present this drawable
[commandBuffer addPresent: drawable];

//  Put the command buffer into the queue
[commandBuffer commit];
```

Building a Metal Application

Drawing

1. Get a command buffer
2. Start a Render Pass
3. Draw
4. Commit the command buffer

Building a Metal Application

Drawing

1. Get a command buffer
2. Start a Render Pass
3. Draw
4. Commit the command buffer

Building a Metal Application

Drawing

1. Get a command buffer
2. Start a Render Pass
3. Draw
4. Commit the command buffer

Building a Metal Application

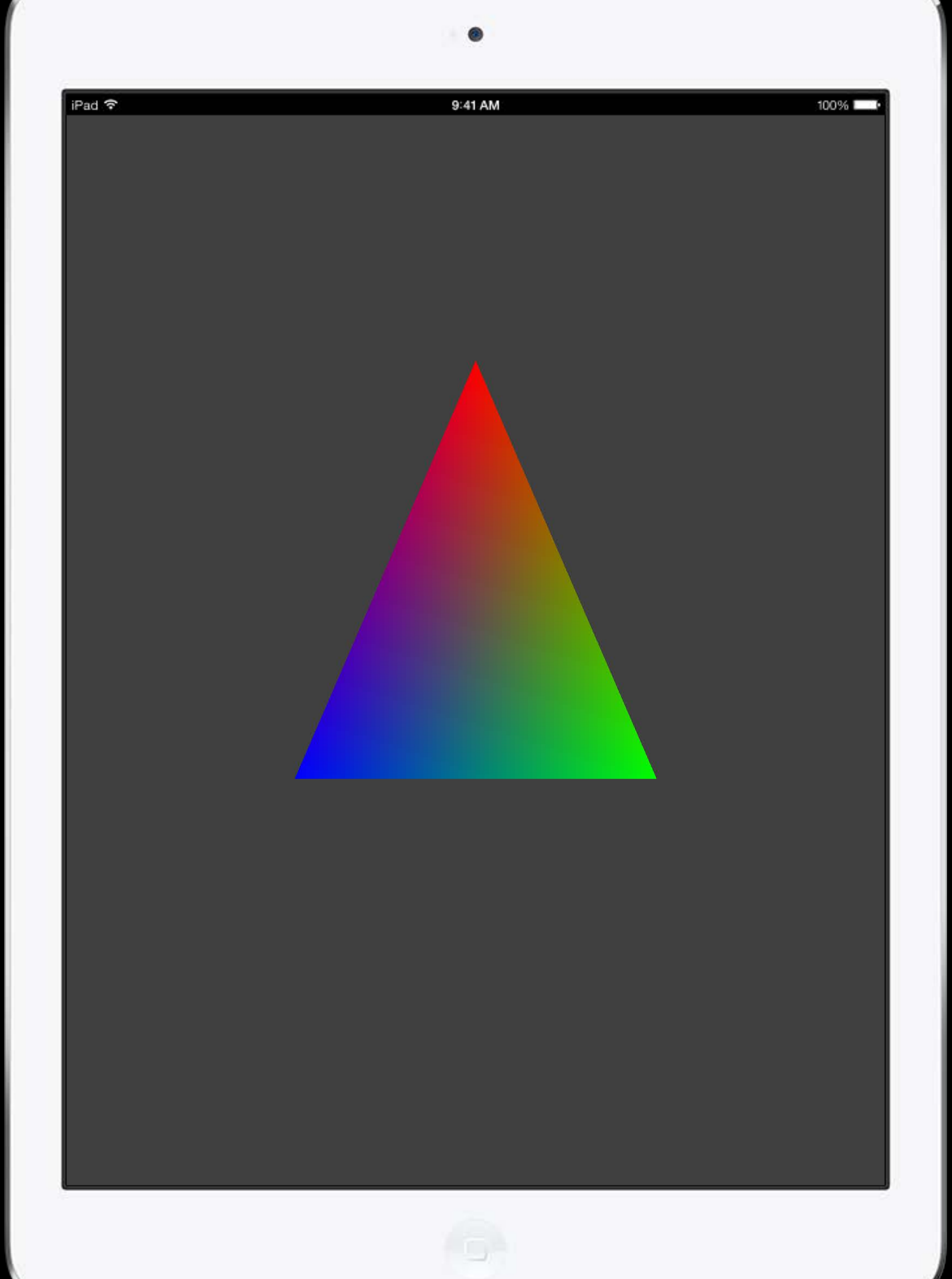
Drawing

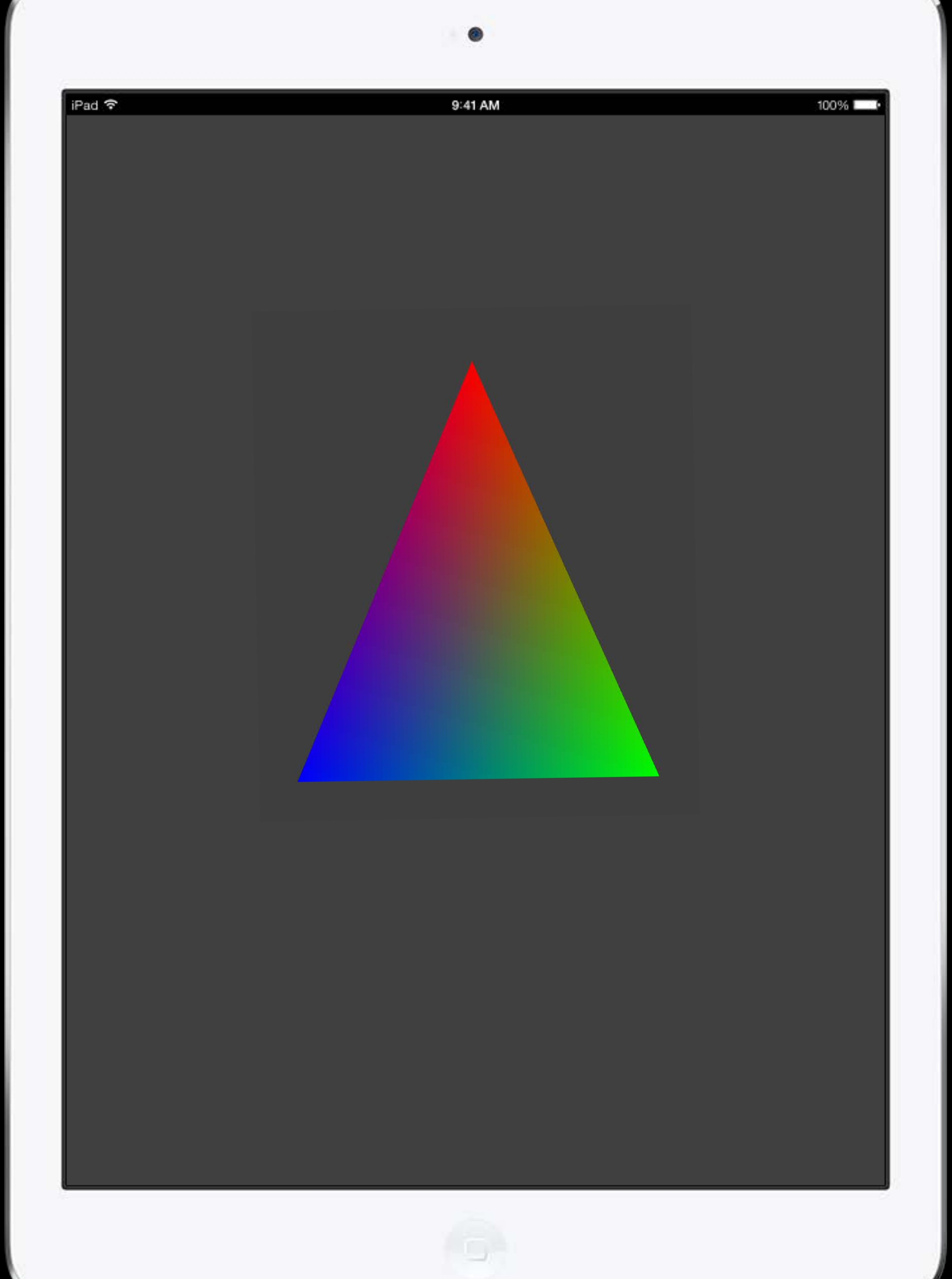
1. Get a command buffer
2. Start a Render Pass
3. Draw
4. Commit the command buffer

Building a Metal Application
Drawing

1. Get a command buffer
2. Start a Render Pass
3. Draw
4. Commit the command buffer

Uniforms and Synchronization


iPad
9:41 AM
100%


iPad
9:41 AM
100%

Vertex Shader with Uniforms

```
struct Uniforms {
    float4x4 mvp_matrix;
};

vertex VSOut vertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    constant Uniforms& uniforms      [[ buffer(1) ]],
    unsigned int vid [[ vertex_id]])
{
     VSOut out;
     out.position = uniforms.mvp_matrix * vertexArray[vid].position;
     out.color = half4(vertexArray[vid].color);
     return out;
}
```

Vertex Shader with Uniforms

```
struct Uniforms {
    float4x4 mvp_matrix;
};

vertex VSOut vertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    constant Uniforms& uniforms      [[ buffer(1) ]],
    unsigned int vid [[ vertex_id]])
{
     VSOut out;
     out.position = uniforms.mvp_matrix * vertexArray[vid].position;
     out.color = half4(vertexArray[vid].color);
     return out;
}
```

Vertex Shader with Uniforms

```
struct Uniforms {
    float4x4 mvp_matrix;
};

vertex VSOut vertexShader(
    const global Vertex* vertexArray [[ buffer(0) ]],
    constant Uniforms& uniforms      [[ buffer(1) ]],
    unsigned int vid [[ vertex_id]])
{
     VSOut out;
     out.position = uniforms.mvp_matrix * vertexArray[vid].position;
     out.color = half4(vertexArray[vid].color);
     return out;
}
```

Render Command with Uniforms

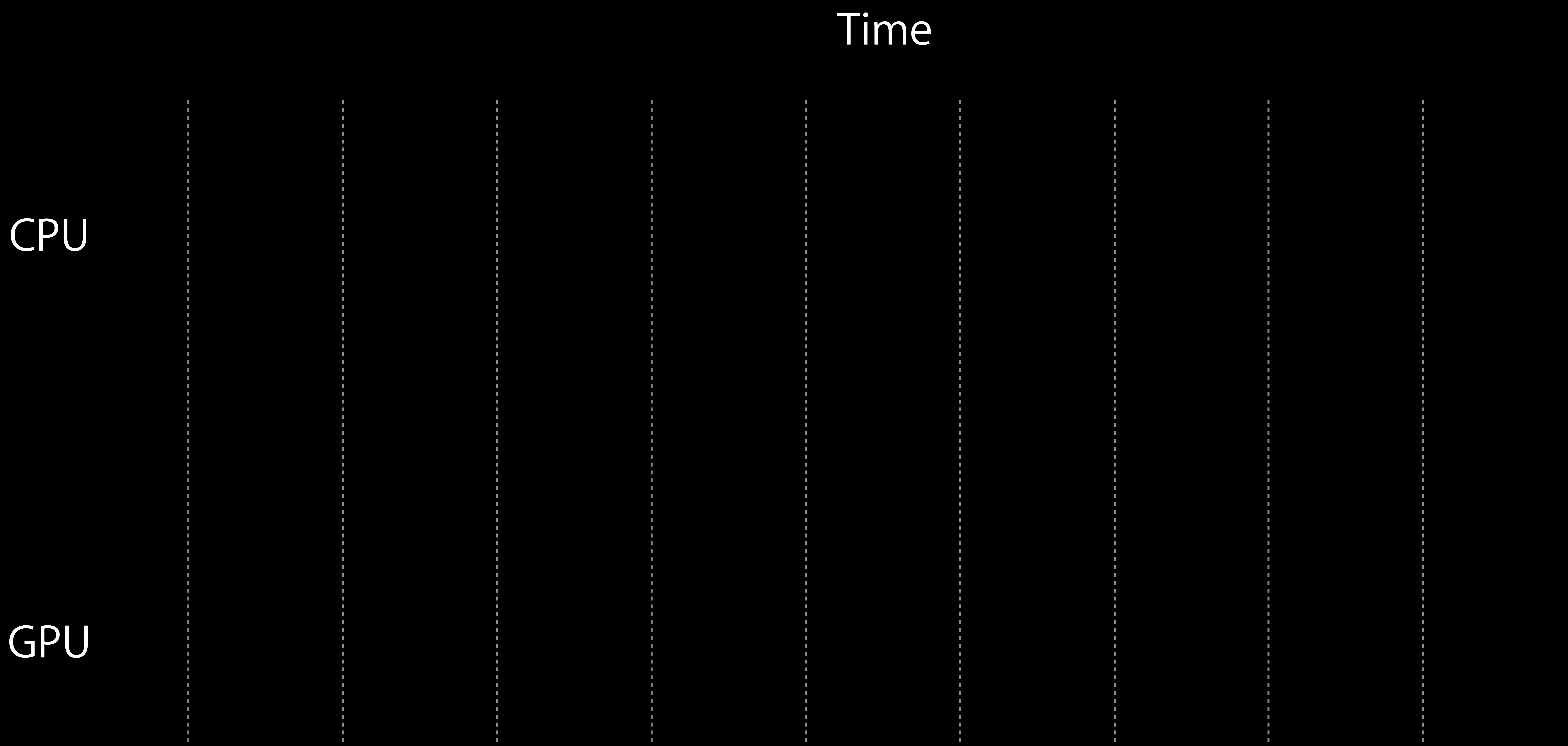
```
struct Uniforms* uniforms = [uniformBuffer contents];
uniforms->mvp_matrix = …;
```

```
[render setRenderPipelineState: renderPipeline];
[render setVertexBuffer: vertexArray   offset: 0 atIndex: 0];
[render setVertexBuffer: uniformBuffer offset: 0 atIndex: 1];
[render drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0 vertexCount:3];
```

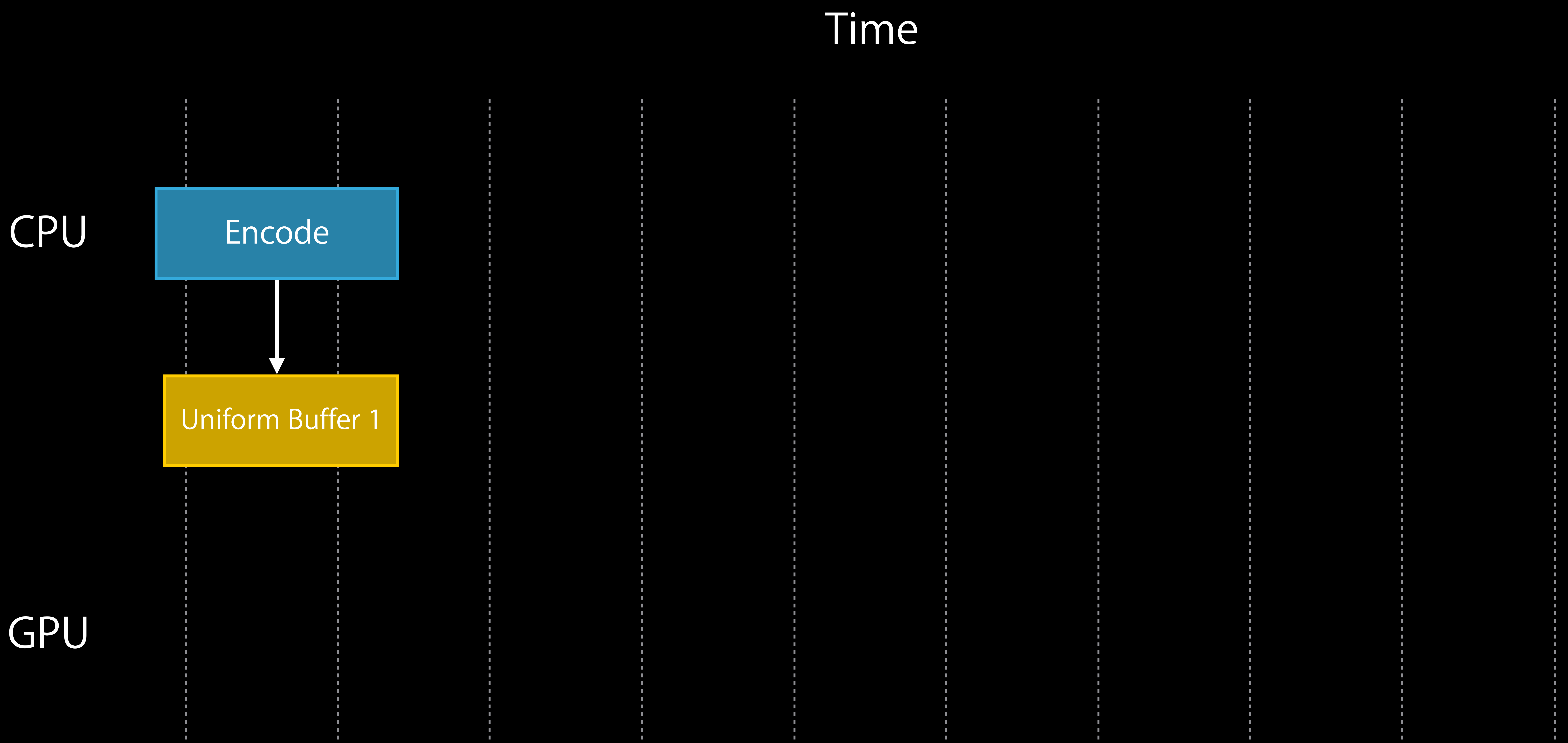
Render Command with Uniforms

```
struct Uniforms* uniforms = [uniformBuffer contents];
uniforms->mvp_matrix = …;

[render setRenderPipelineState: renderPipeline];
[render setVertexBuffer: vertexArray   offset: 0 atIndex: 0];
[render setVertexBuffer: uniformBuffer offset: 0 atIndex: 1];
[render drawPrimitives:MTLPrimitiveTypeTriangle vertexStart:0 vertexCount:3];
```

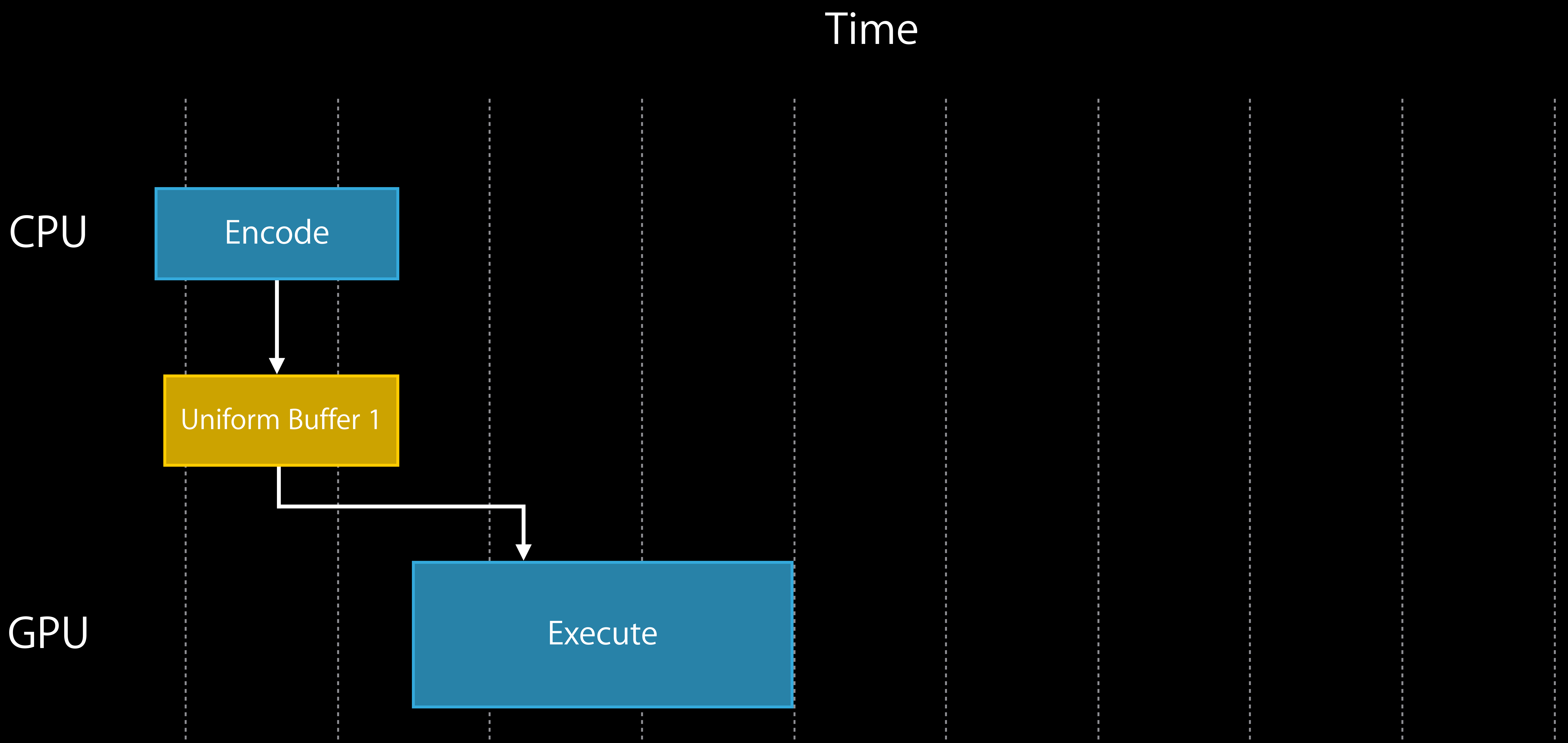
CPU and GPU Pipelining



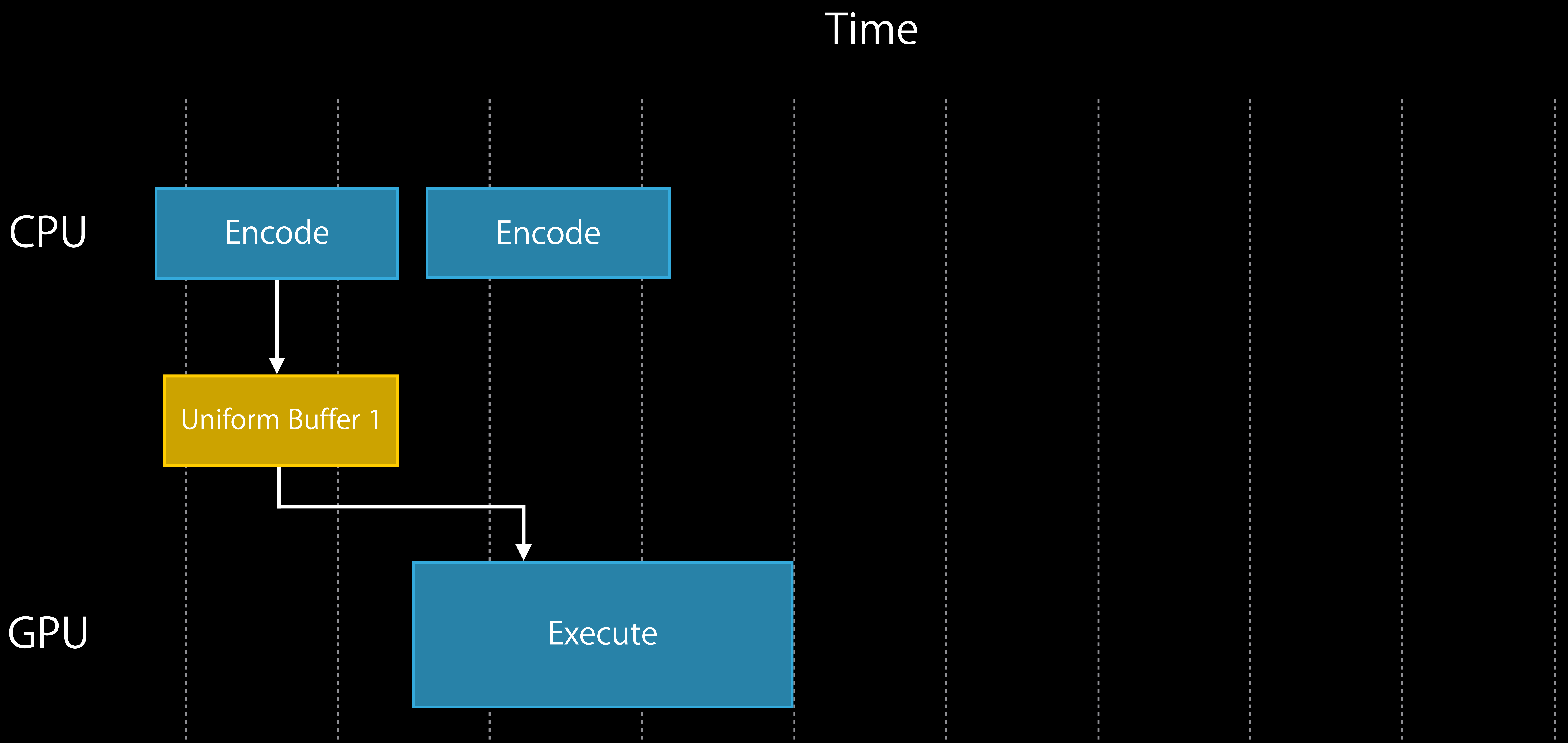
CPU and GPU Pipelining



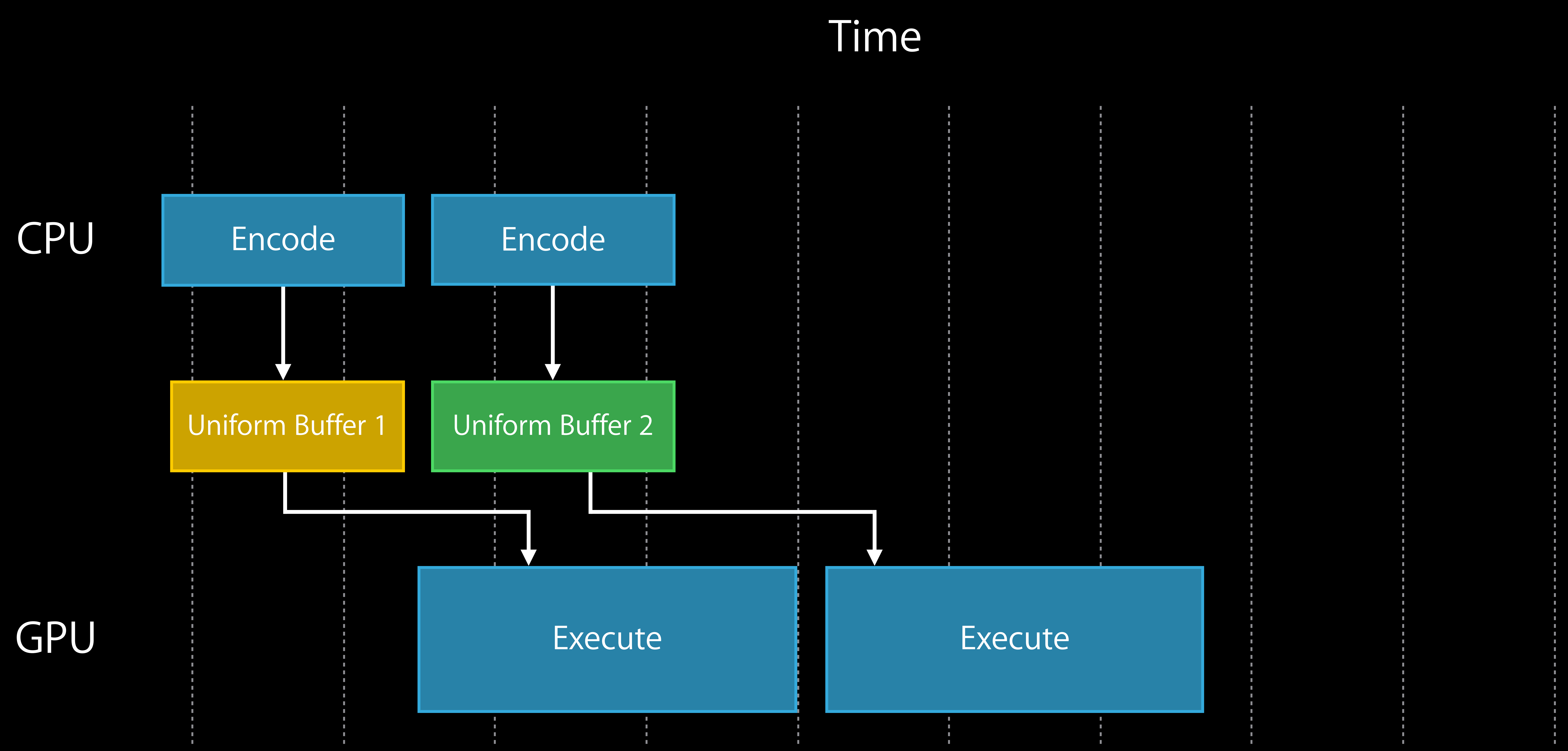
CPU and GPU Pipelining



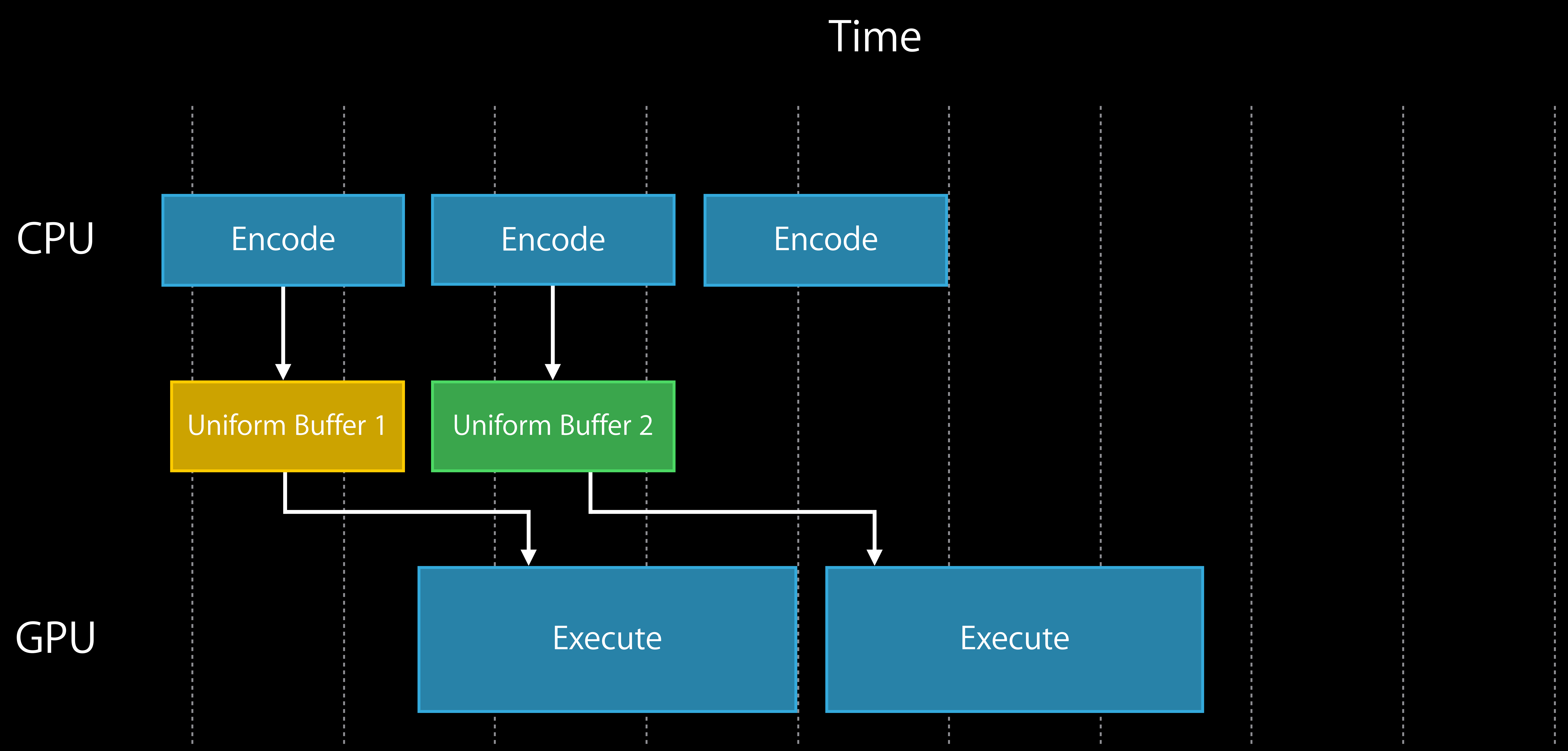
CPU and GPU Pipelining



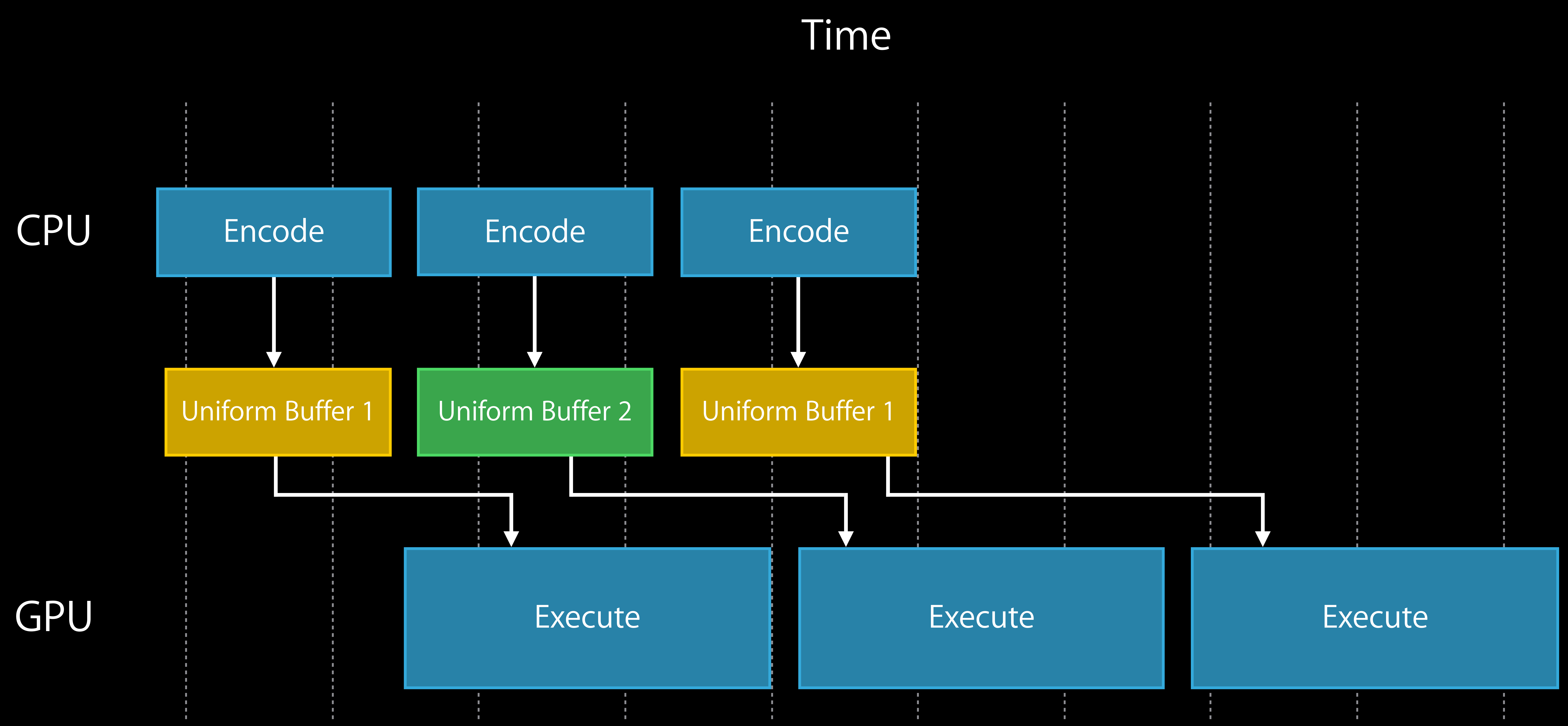
CPU and GPU Pipelining



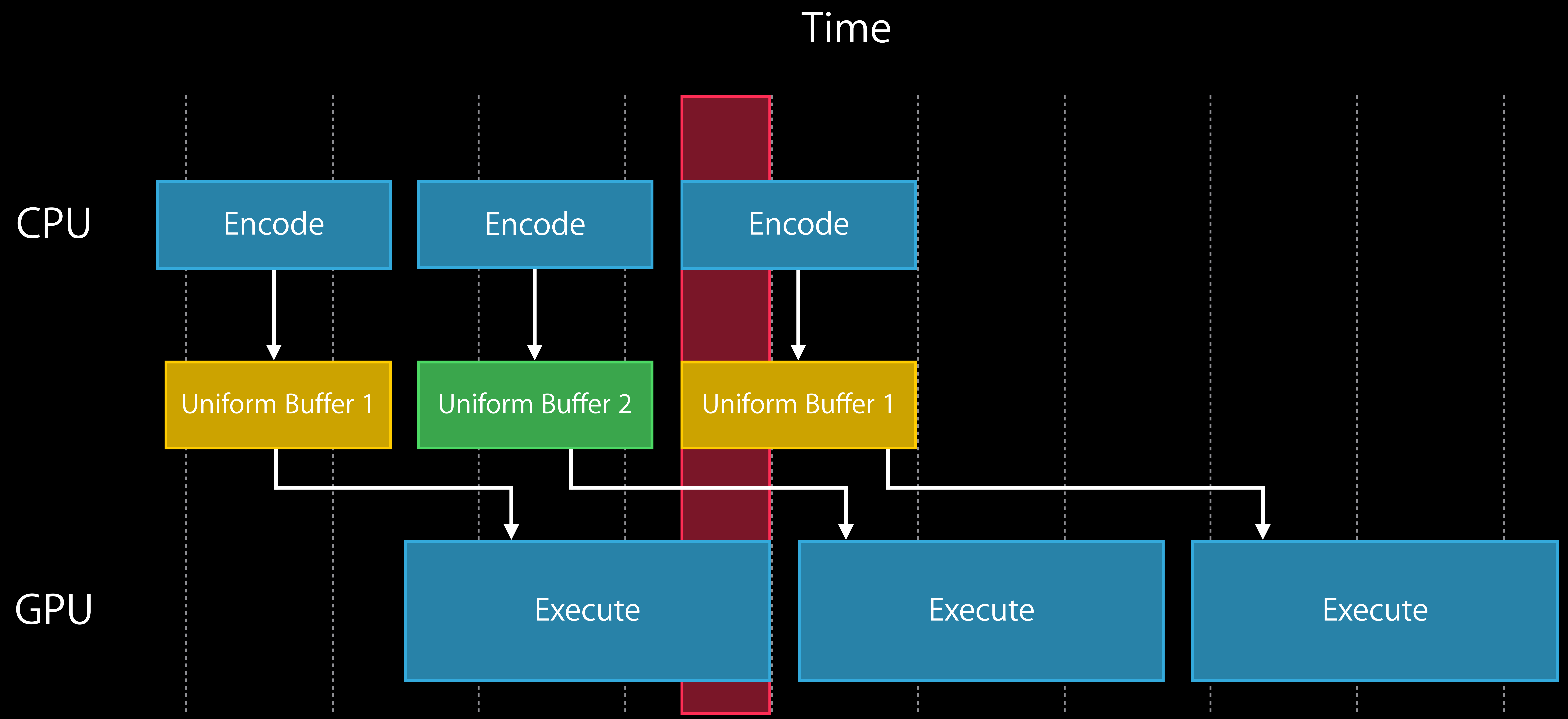
CPU and GPU Pipelining



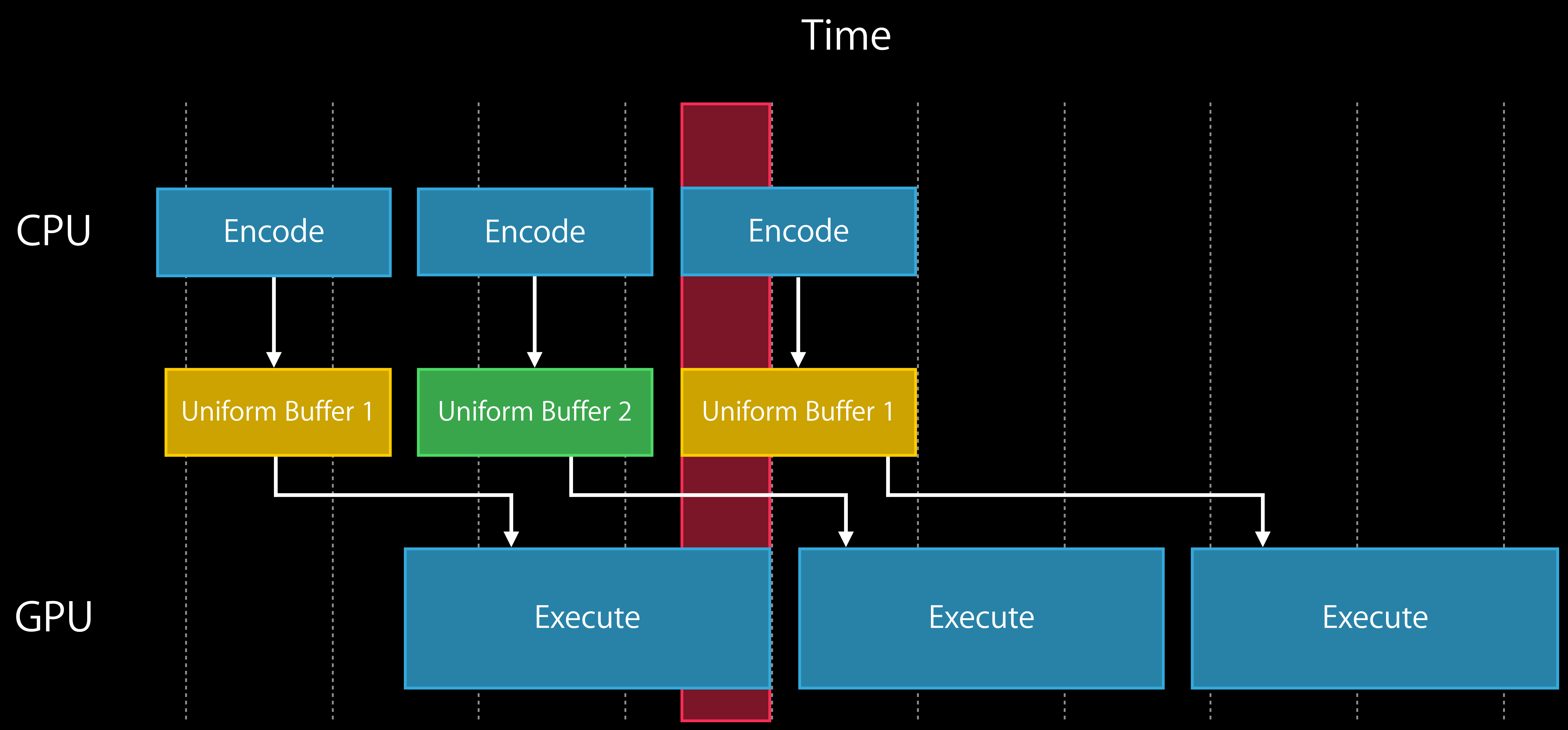
CPU and GPU Pipelining



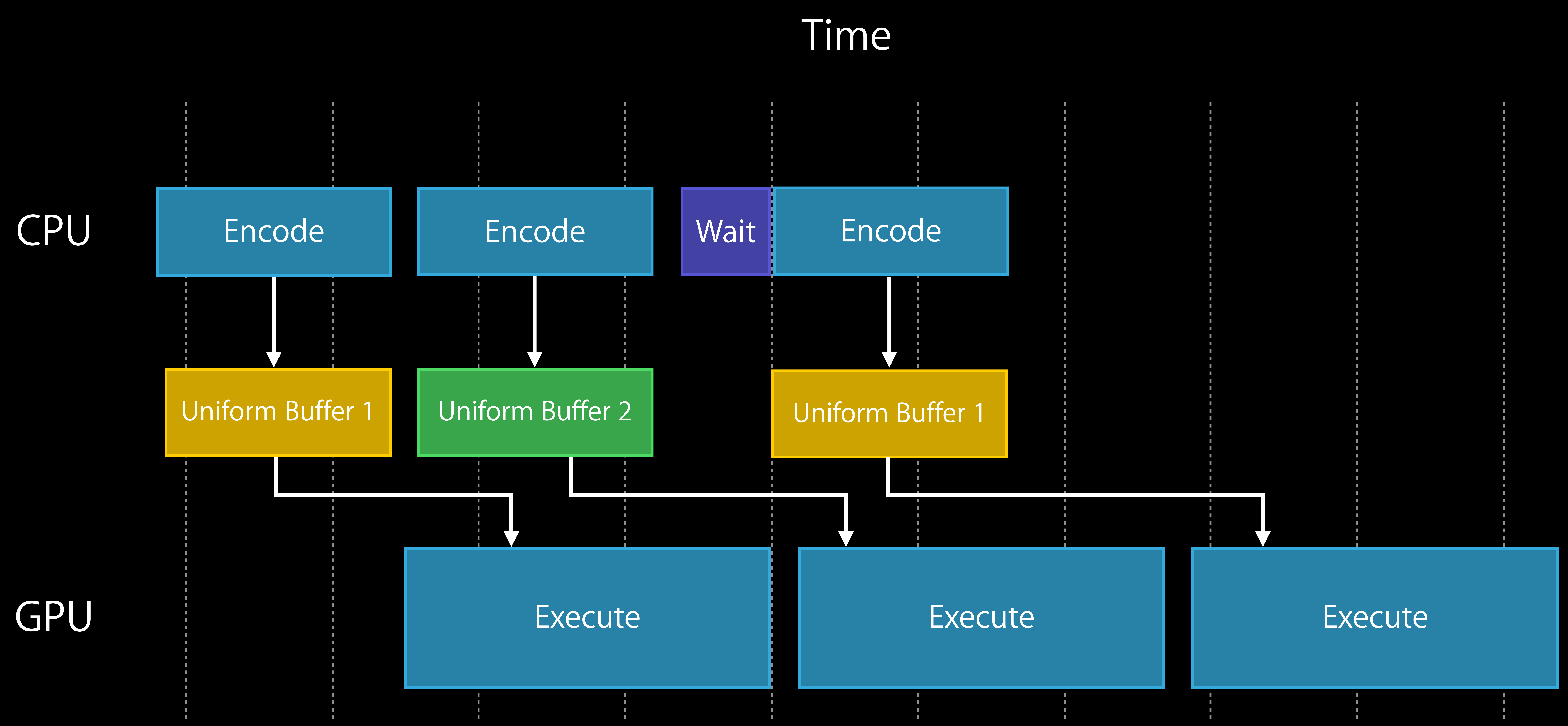
CPU and GPU Pipelining



CPU and GPU Pipelining



CPU and GPU Pipelining



Waiting for Command Buffers

Waiting for Command Buffers

```
//  Initialization
available_resources = dispatch_semaphore_create(3);
```

Waiting for Command Buffers

```
//  Initialization
available_resources = dispatch_semaphore_create(3);

//  Per frame
{

    //  Build a CommandBuffer




    [commandBuffer commit];
}
```

Waiting for Command Buffers

```
//  Initialization
available_resources = dispatch_semaphore_create(3);

//  Per frame
{
    dispatch_semaphore_wait(available_resources, DISPATCH_TIME_FOREVER);

    //  Build a CommandBuffer





    [commandBuffer commit];
}
```

Waiting for Command Buffers

```
//  Initialization
available_resources = dispatch_semaphore_create(3);

//  Per frame
{
    dispatch_semaphore_wait(available_resources, DISPATCH_TIME_FOREVER);

    //  Build a CommandBuffer

    //  Register a completion callback, unblock any waiting threads
    [commandBuffer addCompletedHandler:^(id<MTLCommandBuffer> cb) {
        dispatch_semaphore_signal(available_resources);
    }];
    [commandBuffer commit];
}
```

Metal Fundamentals

Building a Metal application

- Initialization
- Drawing
- Uniforms and synchronization

Metal shading language

- Writing shaders in Metal
- Data types in Metal
- Shader inputs, outputs, and matching rules

Metal Fundamentals

Building a Metal application

- Initialization
- Drawing
- Uniforms and synchronization

Metal shading language

- Writing shaders in Metal
- Data types in Metal
- Shader inputs, outputs, and matching rules

Metal Fundamentals

Building a Metal application

- Initialization
- Drawing
- Uniforms and synchronization

Metal shading language

- Writing shaders in Metal
- Data types in Metal
- Shader inputs, outputs, and matching rules

Metal Fundamentals

Building a Metal application

- Initialization
- Drawing
- Uniforms and synchronization

Metal shading language

- Writing shaders in Metal
- Data types in Metal
- Shader inputs, outputs, and matching rules

Metal Fundamentals

Building a Metal application

- Initialization
- Drawing
- Uniforms and synchronization

Metal shading language

- Writing shaders in Metal
- Data types in Metal
- Shader inputs, outputs, and matching rules

Metal Shading Language

A unified language for graphics and compute

Aaftab Munshi
GPU Software Engineer

Metal Fundamentals

Building a Metal application

- Initialization
- Drawing
- Uniforms and synchronization

Metal shading language

- Writing shaders in Metal
- Data types in Metal
- Shader inputs, outputs, and matching rules

Metal Fundamentals

Building a Metal application

- Initialization
- Drawing
- Uniforms and synchronization

Metal shading language

- Writing shaders in Metal
- Data types in Metal
- Shader inputs, outputs, and matching rules

Writing Shaders in Metal

Pseudo Code for a Vertex Shader

```
struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};
VertexOutput
texturedQuadVertex(const float4* vtx_data,
                   const float2* uv_data,
                   uint vid)
{
    VertexOutput v_out;
    v_out.pos = vtx_data[vid];
    v_out.uv = uv_data[vid];
    return v_out;
}
```

Pseudo Code for a Vertex Shader

```
struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};

VertexOutput
texturedQuadVertex(const float4* vtx_data,
                   const float2* uv_data,
                   uint vid)
{
    VertexOutput v_out;
    v_out.pos = vtx_data[vid];
    v_out.uv = uv_data[vid];
    return v_out;
}
```

Pseudo Code for a Vertex Shader

```
#include <metal_stdlib>
using namespace metal;

struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};

VertexOutput
texturedQuadVertex(const float4* vtx_data,
                   const float2* uv_data,
                   uint vid)
{
    VertexOutput v_out;
    v_out.pos = vtx_data[vid];
    v_out.uv = uv_data[vid];
    return v_out;
}
```

Pseudo Code for a Vertex Shader

```
#include <metal_stdlib>
using namespace metal;

struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};

vertex VertexOutput
texturedQuadVertex(const float4* vtx_data,
                   const float2* uv_data,
                   uint vid)
{
    VertexOutput v_out;
    v_out.pos = vtx_data[vid];
    v_out.uv = uv_data[vid];
    return v_out;
}
```

Pseudo Code for a Vertex Shader

```
#include <metal_stdlib>
using namespace metal;

struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};

vertex VertexOutput
texturedQuadVertex(const global float4* vtx_data, [[ buffer(0) ]],
                   const global float2* uv_data, [[ buffer(1) ]],
                   uint vid)
{
    VertexOutput v_out;
    v_out.pos = vtx_data[vid];
    v_out.uv = uv_data[vid];
    return v_out;
}
```

Metal Vertex Shader

```
#include <metal_stdlib>
using namespace metal;

struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};

vertex VertexOutput
texturedQuadVertex(const global float4* vtx_data, [[ buffer(0) ]],
                   const global float2* uv_data, [[ buffer(1) ]],
                   uint vid [[ vertex_id ]])
{
    VertexOutput v_out;
    v_out.pos = vtx_data[vid];
    v_out.uv = uv_data[vid];
    return v_out;
}
```

Metal Vertex Shader

```
#include <metal_stdlib>
using namespace metal;

struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};

vertex VertexOutput
texturedQuadVertex(const global float4* vtx_data, [[ buffer(0) ]],
                   const global float2* uv_data, [[ buffer(1) ]],
                   uint vid [[ vertex_id ]])
{
    VertexOutput v_out;
    v_out.pos = vtx_data[vid];
    v_out.uv = uv_data[vid];
    return v_out;
}
```

Pseudo Code for a Fragment Shader

```
#include <metal_stdlib>
using namespace metal;

struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};
float4
texturedQuadFragment(VertexOutput frag_input,
                   texture2d<float> tex [[ texture(0) ]],
                   sampler s [[ sampler(0) ]]

{
   return tex.sample(s, frag_input.uv);
}
```

Pseudo Code for a Fragment Shader

```
#include <metal_stdlib>
using namespace metal;

struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};
fragment float4
texturedQuadFragment(VertexOutput frag_input,
                     texture2d<float> tex [[ texture(0) ]],
                     sampler s [[ sampler(0) ]]

{
   return tex.sample(s, frag_input.uv);
}
```

Metal Fragment Shader

```
#include <metal_stdlib>
using namespace metal;

struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};
fragment float4
texturedQuadFragment(VertexOutput frag_input, [[ stage_in ]],
                   texture2d<float> tex [[ texture(0) ]],
                   sampler s [[ sampler(0) ]]

{
   return tex.sample(s, frag_input.uv);
}
```

Metal Fragment Shader

```
#include <metal_stdlib>
using namespace metal;

struct VertexOutput {
    float4 pos [[ position ]];
    float2 uv;
};
fragment float4
texturedQuadFragment(VertexOutput frag_input, [[ stage_in ]],
                     texture2d<float> tex [[ texture(0) ]],
                     sampler s [[ sampler(0) ]]

{
   return tex.sample(s, frag_input.uv);
}
```

Data Types

Scalars, vectors, matrices, and atomics

Scalars

Scalars

C++11 scalar types

Scalars

C++11 scalar types

The half type

Scalars

C++11 scalar types

The half type

Use the half type wherever you can

Vectors and Matrices

More than just a big scalar

Vectors and Matrices

More than just a big scalar

Vectors

- Two-, three-, and four-component integer and floating-point types
- char2, int3, float4, half2, etc.

Vectors and Matrices

More than just a big scalar

Vectors

- Two-, three-, and four-component integer and floating-point types
- char2, int3, float4, half2, etc.

Matrices

- float*nxm*, half*nxm*
- Column major order

Vectors and Matrices

More than just a big scalar

Vectors

- Two-, three-, and four-component integer and floating-point types
- char2, int3, float4, half2, etc.

Matrices

- float*nxm*, half*nxm*
- Column major order

Vector and Matrix Constructors and Operators

- Similar to GLSL

Vectors and Matrices

More than just a big scalar

Vectors

- Two-, three-, and four-component integer and floating-point types
- char2, int3, float4, half2, etc.

Matrices

- float*nxm*, half*nxm*
- Column major order

Vector and Matrix Constructors and Operators

- Similar to GLSL

Types defined by simd/simd.h

Vectors and Matrices

More than just a big scalar

Vectors

- Two-, three-, and four-component integer and floating-point types
- char2, int3, float4, half2, etc.

Matrices

- float*nxm*, half*nxm*
- Column major order

Vector and Matrix Constructors and Operators

- Similar to GLSL

Types defined by simd/simd.h

Use the half*n* and half*nxm* types wherever you can

Vectors

Aligned at vector length

```
struct Foo {
    float a;
    float2 b;
    float4 c;
};
```

Vectors

Aligned at vector length

```
struct Foo {
    float a;
    float2 b;
    float4 c;
};
```

alignment = 8 bytes

Vectors

Aligned at vector length

```
struct Foo {
    float a;
    float2 b;   ← alignment = 8 bytes
    float4 c;   ← alignment = 16 bytes
};
```

Vectors

Aligned at vector length

```
struct Foo {
    float a;
    float pad;
    float2 b;
    float4 c;
};
```

Potential impact to both allocation size and memory b/w

Vectors

Aligned at vector length

```
struct Foo {
    float a;
    float pad;   ← generated by compiler
    float2 b;
    float4 c;
};
```

Potential impact to both allocation size and memory b/w

Vectors

Aligned at vector length

```
struct Foo {
    float a;
    float pad;  ←  generated by compiler
    float2 b;
    float4 c;
};
```

```
sizeof(Foo) = 32 bytes
```

Potential impact to both allocation size and memory b/w

Vectors

Aligned at vector length

What if we declare them in order of decreasing size?

```
struct Foo {
    float4 c;
    float2 b;
    float  a;
};
```

Vectors

Aligned at vector length

What if we declare them in order of decreasing size?

```
struct Foo {
    float4 c;
    float2 b;
    float  a;
};
```

```
sizeof(Foo) is still 32 bytes
```

Vectors

Packed vector types

Vectors

Packed vector types

packed_float3, packed_char4, …

Vectors

Packed vector types

packed_float3, packed_char4, …

Always aligned at scalar type length

Vectors

Packed vector types

packed_float3, packed_char4, …

Always aligned at scalar type length

```
struct Foo {
    float a;
    packed_float2 b;
    packed_float4 c;
};
```

Vectors

Packed vector types

packed_float3, packed_char4, …

Always aligned at scalar type length

```
struct Foo {
    float a;
    packed_float2 b; ← alignment = 4 bytes
    packed_float4 c; ← alignment = 4 bytes
};
```

Vectors

Packed vector types

packed_float3, packed_char4, …

Always aligned at scalar type length

```
struct Foo {
    float a;
    packed_float2 b;  ←  alignment = 4 bytes
    packed_float4 c;  ←  alignment = 4 bytes
};

        sizeof(Foo) = 28 bytes
```

Vectors

Packed vector types

packed_float3, packed_char4, …

Always aligned at scalar type length

```
struct Foo {
    float a;
    packed_float2 b; ← alignment = 4 bytes
    packed_float4 c; ← alignment = 4 bytes
};
```

```
sizeof(Foo) = 28 bytes
```

Not a good fit for CPU as CPUs prefer aligned vector types

Atomic

Atomic

Supported atomic types

- `atomic_int` and `atomic_uint`

Atomic

Supported atomic types

- `atomic_int` and `atomic_uint`

Operations on atomic types are race-free

- Subset of C++11 atomic functions
- Guaranteed to be performed without interference from other threads

Data Types

Textures, samplers, and buffers

Textures

A templated type

Textures

A templated type

Template parameters

Textures

A templated type

Template parameters

- Color type
  - Float, half, int, or uint

Textures

A templated type

Template parameters

- Color type
  - Float, half, int, or uint
- Access mode
  - Sample, read, or write

Textures

A templated type

Template parameters

- Color type
  - Float, half, int, or uint
- Access mode
  - Sample, read, or write

Separate type for depth textures

Textures

A templated type

```
fragment FragOutput
my_fragment_shader(
   texture2d<float> tA [[ texture(0) ]],
   texture2d<half, access::write> tB [[ texture(1) ]],
   depth2d<float> tC [[ texture(2) ]],
   …)
{
}
```

Textures

A templated type

```
fragment FragOutput
my_fragment_shader(
    texture2d<float> tA [[ texture(0) ]],
    texture2d<half, access::write> tB [[ texture(1) ]],
    depth2d<float> tC [[ texture(2) ]],
    …)
{
}
```

Textures

A templated type

```
fragment FragOutput
my_fragment_shader(
   texture2d<float> tA [[ texture(0) ]],
   texture2d<half, access::write> tB [[ texture(1) ]],
   depth2d<float> tC [[ texture(2) ]],
   …)
{
}
```

Textures

A templated type

```
fragment FragOutput
my_fragment_shader(
   texture2d<float> tA [[ texture(0) ]],
   texture2d<half, access::write> tB [[ texture(1) ]],
   depth2d<float> tC [[ texture(2) ]],
   …)
{
}
```

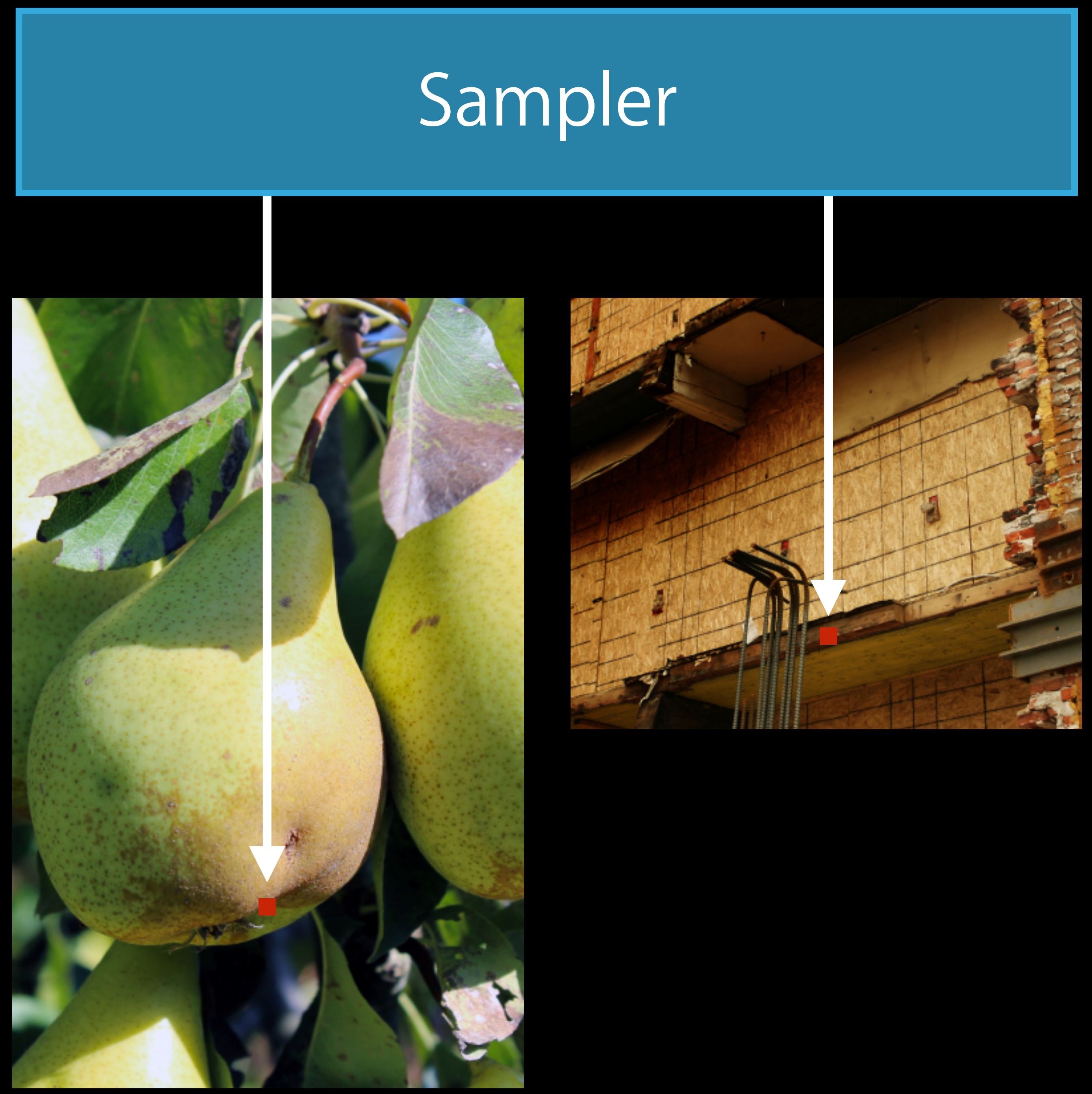
Samplers

Samplers independent from textures

Samplers

Samplers independent from textures

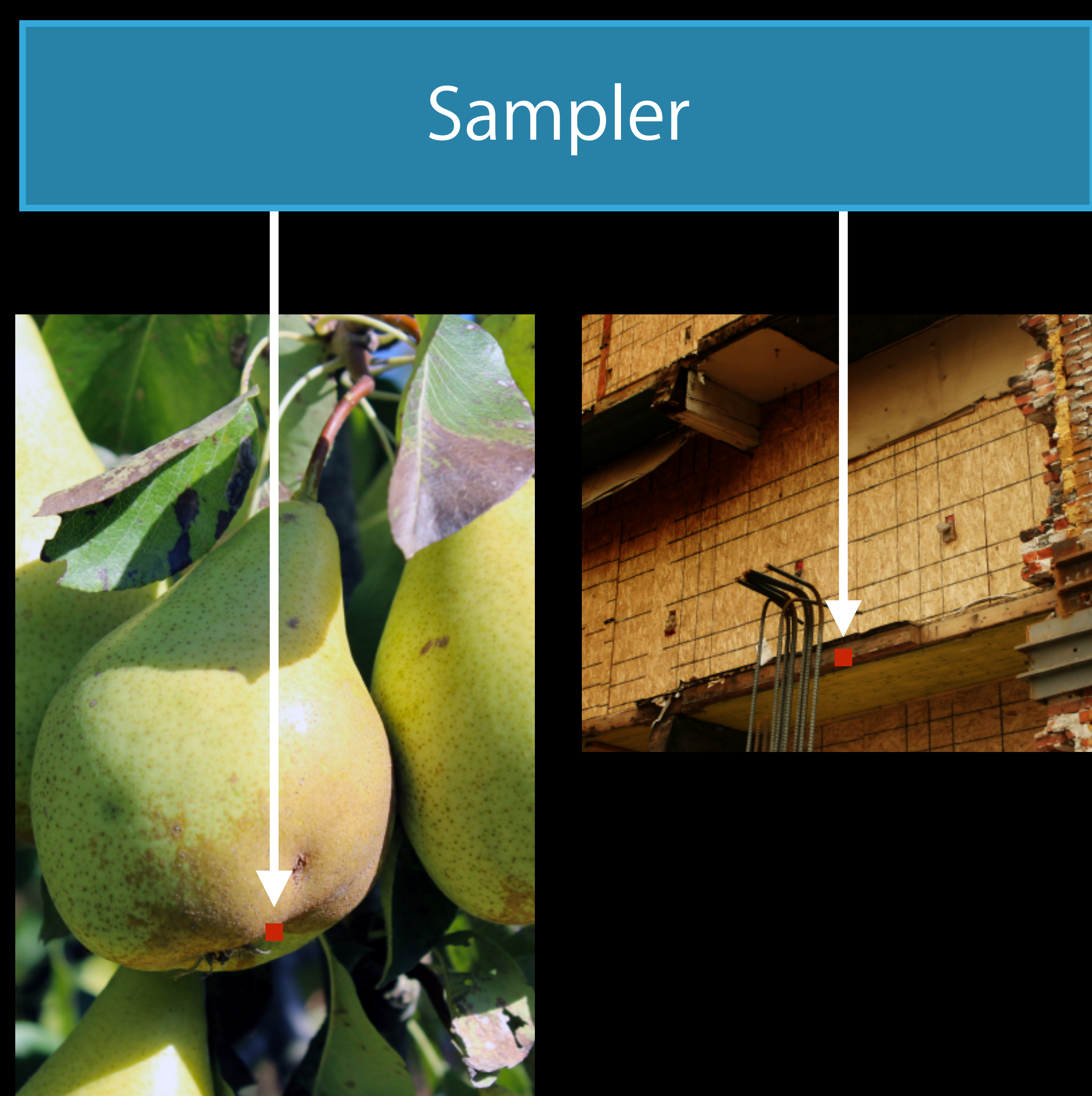
One sampler, multiple textures



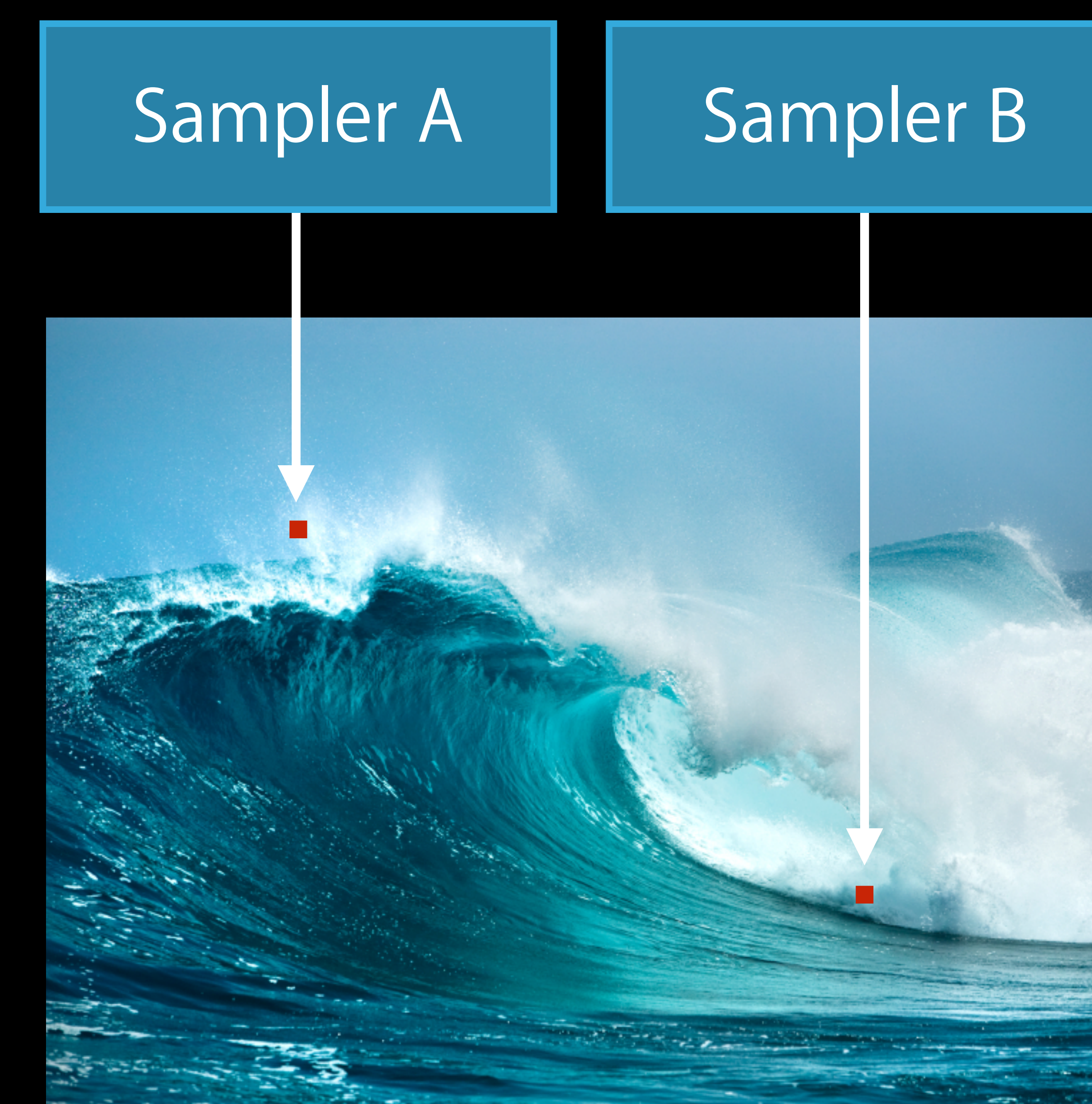
Samplers

Samplers independent from textures

One sampler, multiple textures



Multiple samplers, one texture



Samplers

Argument to a graphics or kernel function

```
fragment float4
texturedQuadFragment(VertexOutput frag_input [[ stage_in ]],
                     texture2d<float> tex [[ texture(0) ]],
                     sampler s [[ sampler(0) ]])
{
    return tex.sample(s, frag_input.texcoord);
}
```

Samplers

Argument to a graphics or kernel function

```
fragment float4
texturedQuadFragment(VertexOutput frag_input [[ stage_in ]],
                     texture2d<float> tex [[ texture(0) ]],
                     sampler s [[ sampler(0) ]])
{
    return tex.sample(s, frag_input.texcoord);
}
```

Samplers

Declared in Metal source

Samplers

Declared in Metal source

```
// Defined as a variadic template
   constexpr sampler s(coord::normalized,
                       filter::linear,
                       address::clamp_to_edge);
```

Samplers

Declared in Metal source

```
// Defined as a variadic template
   constexpr sampler s(coord::normalized,
                       filter::linear,
                       address::clamp_to_edge);

// Defaults for sampler properties not specified
   constexpr sampler s(address::clamp_to_zero);
```

Buffers

Show me the memory

Buffers

Show me the memory

A pointer or a reference to a type

Buffers

Show me the memory

A pointer or a reference to a type

Must be declared in an address space

- global
- constant

Buffers

When to use global

Buffers

When to use global

When buffers are indexed dynamically such as with:

- vertex ID
- global ID

Buffers

When to use constant

Buffers

When to use constant

Should be used when multiple instances index the same location

Buffers

When to use constant

Should be used when multiple instances index the same location

For data structures such as:

- Light descriptors, material properties
- Skinning matrices
- Filter weights

Buffers

When to use constant

Should be used when multiple instances index the same location

For data structures such as:

- Light descriptors, material properties
- Skinning matrices
- Filter weights

Pass by reference

Global vs. Constant

Example

```
vertex VertexOutput
my_vertex(const float3* position_data [[ buffer(0) ]],
           const float3* normal_data [[ buffer(1) ]],
           TransformMatrices& matrices [[ buffer(2) ]],
           uint vid [[ vertex_id ]])
{
    VertexOutput out;

    float3 n_d = normal_data[vid];
    float3 transformed_normal = matrices.normal_matrix * n_d;
    float4 p_d = float4(position_data[vid], 1.0f);
    out.position = matrices.modelview_projection_matrix * p_d;
    float4 eye_vector = matrices.modelview_matrix * p_d;
    ...
    return out;
}
```

Global vs. Constant

Example

```
vertex VertexOutput
my_vertex(const float3* position_data [[ buffer(0) ]],
          const float3* normal_data [[ buffer(1) ]],
          TransformMatrices& matrices [[ buffer(2) ]],
          uint vid [[ vertex_id ]])
{
    VertexOutput out;

    float3 n_d = normal_data[vid];
    float3 transformed_normal = matrices.normal_matrix * n_d;
    float4 p_d = float4(position_data[vid], 1.0f);
    out.position = matrices.modelview_projection_matrix * p_d;
    float4 eye_vector = matrices.modelview_matrix * p_d;
    ...
    return out;
}
```

Global vs. Constant

Example

```
vertex VertexOutput
my_vertex(const float3* position_data [[ buffer(0) ]],
          const float3* normal_data [[ buffer(1) ]],
          TransformMatrices& matrices [[ buffer(2) ]],
          uint vid [[ vertex_id ]])
{
    VertexOutput out;

    float3 n_d = normal_data[vid];
    float3 transformed_normal = matrices.normal_matrix * n_d;
    float4 p_d = float4(position_data[vid], 1.0f);
    out.position = matrices.modelview_projection_matrix * p_d;
    float4 eye_vector = matrices.modelview_matrix * p_d;
    ...
    return out;
}
```

Global vs. Constant

Example

```
vertex VertexOutput
my_vertex(const global float3* position_data [[ buffer(0) ]],
           const global float3* normal_data [[ buffer(1) ]],
           TransformMatrices& matrices [[ buffer(2) ]],
           uint vid [[ vertex_id ]])
{
    VertexOutput out;

    float3 n_d = normal_data[vid];
    float3 transformed_normal = matrices.normal_matrix * n_d;
    float4 p_d = float4(position_data[vid], 1.0f);
    out.position = matrices.modelview_projection_matrix * p_d;
    float4 eye_vector = matrices.modelview_matrix * p_d;
    ...
    return out;
}
```

Global vs. Constant

Example

```
vertex VertexOutput
my_vertex(const global float3* position_data [[ buffer(0) ]],
           const global float3* normal_data [[ buffer(1) ]],
           TransformMatrices& matrices [[ buffer(2) ]],
           uint vid [[ vertex_id ]])
{
    VertexOutput out;

    float3 n_d = normal_data[vid];
    float3 transformed_normal = matrices.normal_matrix * n_d;
    float4 p_d = float4(position_data[vid], 1.0f);
    out.position = matrices.modelview_projection_matrix * p_d;
    float4 eye_vector = matrices.modelview_matrix * p_d;
    ...
    return out;
}
```

Global vs. Constant

Example

```
vertex VertexOutput
my_vertex(const global float3* position_data [[ buffer(0) ]],
           const global float3* normal_data [[ buffer(1) ]],
           constant TransformMatrices& matrices [[ buffer(2) ]],
           uint vid [[ vertex_id ]])
{
    VertexOutput out;

    float3 n_d = normal_data[vid];
    float3 transformed_normal = matrices.normal_matrix * n_d;
    float4 p_d = float4(position_data[vid], 1.0f);
    out.position = matrices.modelview_projection_matrix * p_d;
    float4 eye_vector = matrices.modelview_matrix * p_d;
    ...
    return out;
}
```

Per-Vertex Inputs

Two methods for reading vertex data

Per-Vertex Inputs—Option One

Vertex data layout is known by the shader

Per-Vertex Inputs—Option One

Vertex data layout is known by the shader

Pass pointers to vertex input buffers in global address space

Use vertex ID and instance ID to index into vertex buffers

Per-Vertex Inputs—Option One

Vertex data layout is known by the shader

Pass pointers to vertex input buffers in global address space

Use vertex ID and instance ID to index into vertex buffers

```
vertex VertexOutput
my_vertex_shader(vertexInputA* inputA [[ buffer(0) ]],
                 vertexInputB* inputB [[ buffer(1) ]],
                 uint vid [[ vertex_id ]],
                 uint instid [[ instance_id ]])
{
   float a = inputA[vid].a;

   half4 b = inputB[instid].b;

   ...
}
```

Per-Vertex Inputs—Option One

Vertex data layout is known by the shader

Pass pointers to vertex input buffers in global address space

Use vertex ID and instance ID to index into vertex buffers

```
vertex VertexOutput
my_vertex_shader(vertexInputA* inputA [[ buffer(0) ]],
                 vertexInputB* inputB [[ buffer(1) ]],
                 uint vid [[ vertex_id ]],
                 uint instid [[ instance_id ]])
{
    float a = inputA[vid].a;

    half4 b = inputB[instid].b;

    ...
}
```

Per-Vertex Inputs—Option Two

Decouple vertex input data from type used in shader

Per-Vertex Inputs—Option Two

Decouple vertex input data from type used in shader

Good match to OpenGL's Vertex Array API

Per-Vertex Inputs—Option Two

Decouple vertex input data from type used in shader

Good match to OpenGL's Vertex Array API

A vertex descriptor for fetching data in the API

- Data type in shader can be different from the input data format
- One or more buffers can be used to describe vertex inputs

Per-Vertex Inputs—Option Two

Decouple vertex input data from type used in shader

Good match to OpenGL's Vertex Array API

A vertex descriptor for fetching data in the API

- Data type in shader can be different from the input data format
- One or more buffers can be used to describe vertex inputs

Per-vertex inputs to shader

- Declared as a struct
- Described with the [[`stage_in`]] qualifier

Per-Vertex Inputs—Option Two

Decouple vertex input data from type used in shader

Good match to OpenGL's Vertex Array API

A vertex descriptor for fetching data in the API

- Data type in shader can be different from the input data format
- One or more buffers can be used to describe vertex inputs

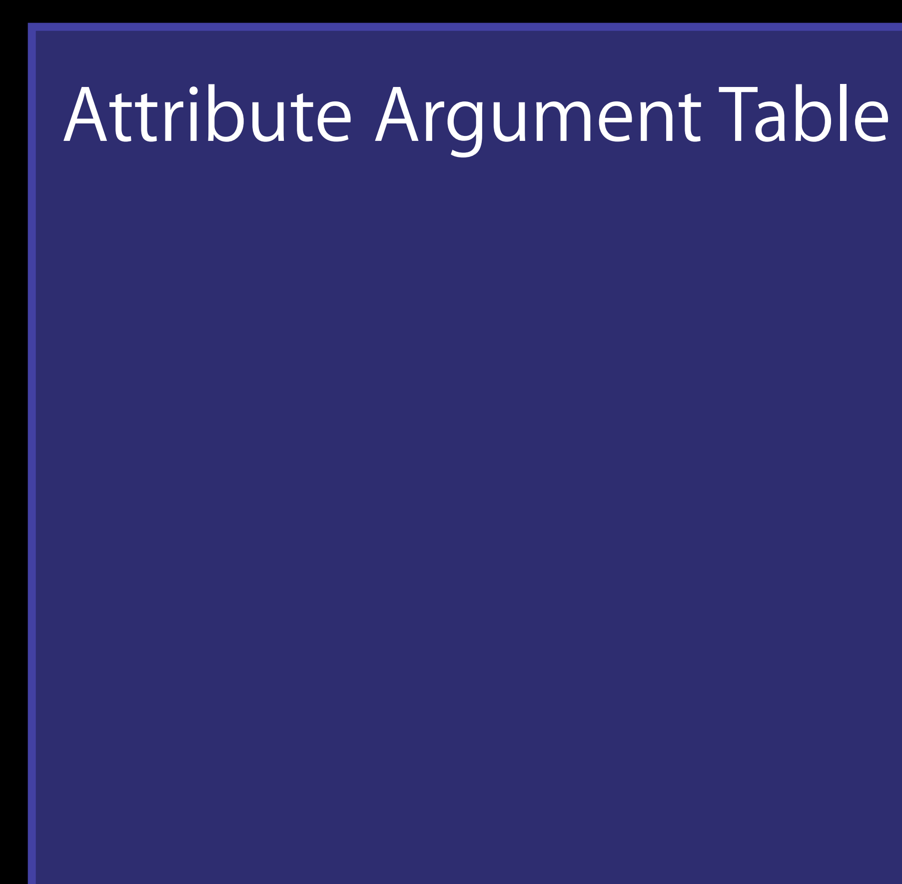
Per-vertex inputs to shader

- Declared as a struct
- Described with the [[`stage_in`]] qualifier

Attribute index to identify each vertex input

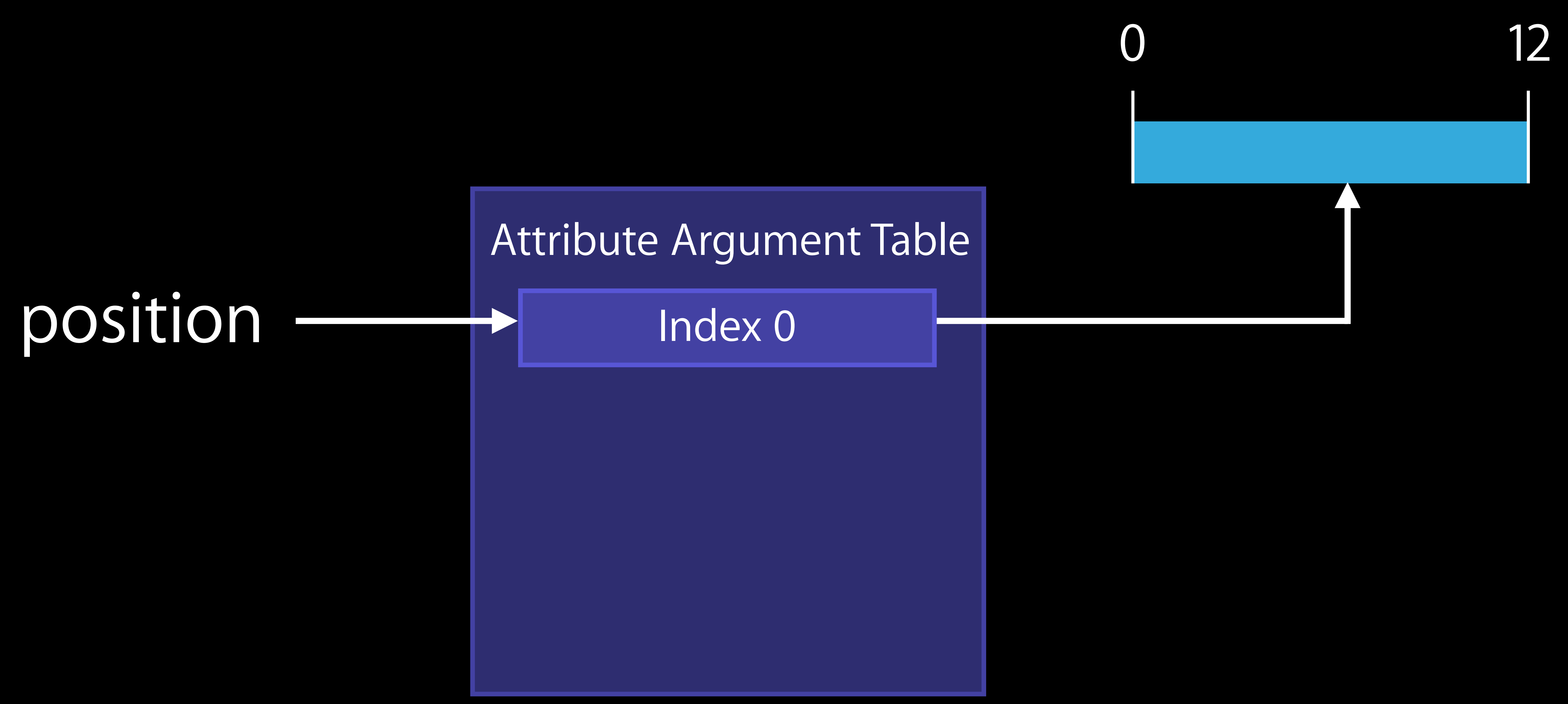
Per-Vertex Inputs—Option 2

Decouple vertex input data from type used in shader



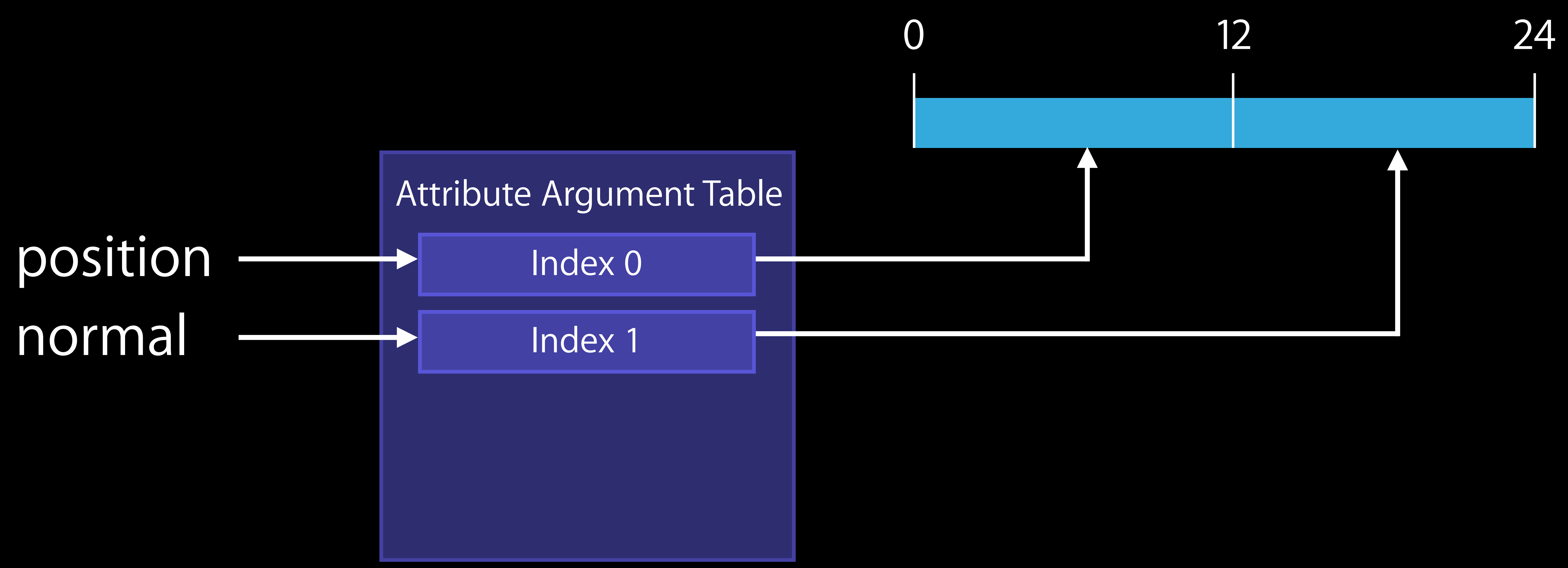
Per-Vertex Inputs—Option 2

Decouple vertex input data from type used in shader



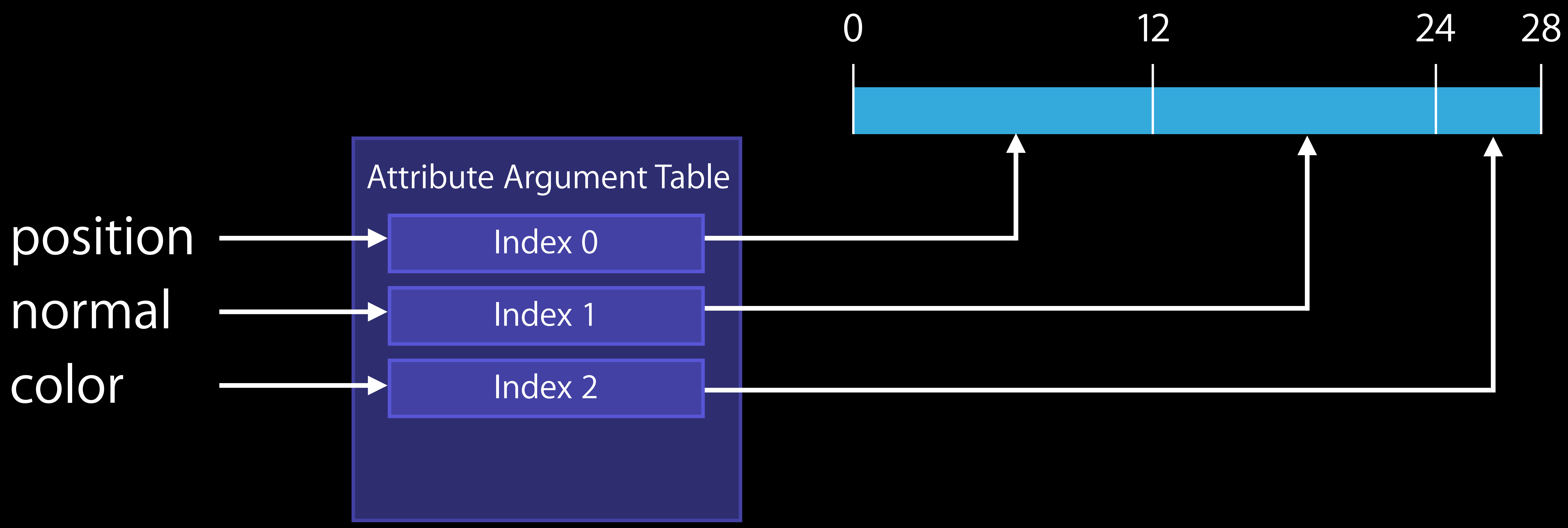
Per-Vertex Inputs—Option 2

Decouple vertex input data from type used in shader



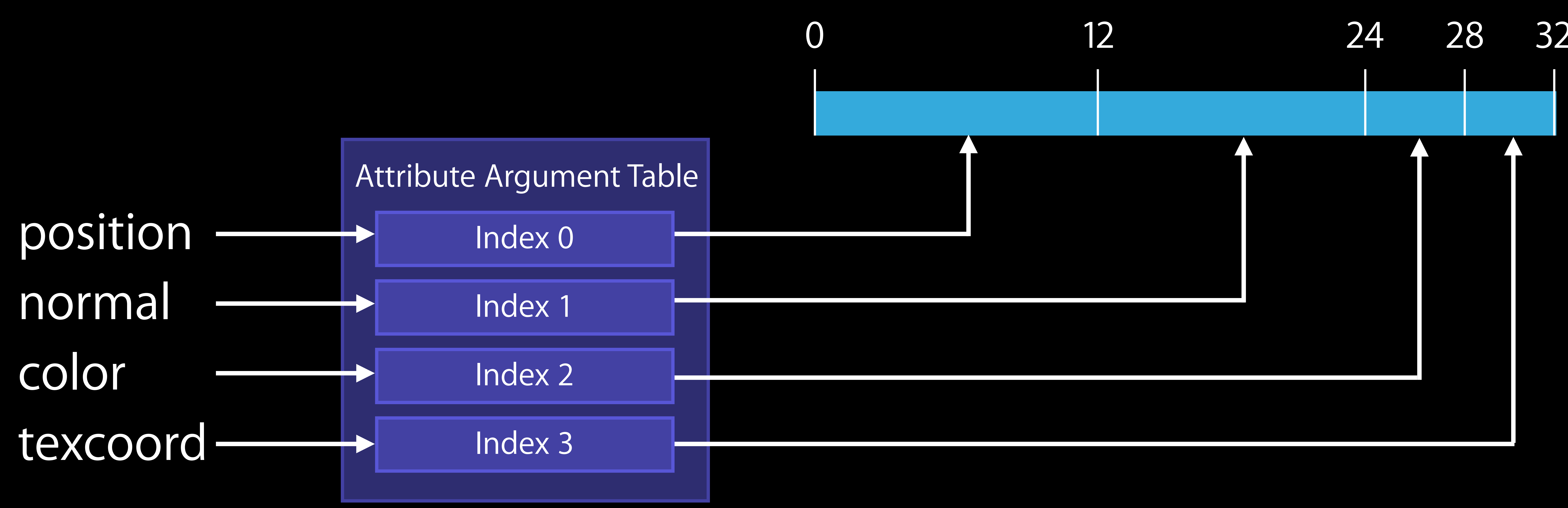
Per-Vertex Inputs—Option 2

Decouple vertex input data from type used in shader



Per-Vertex Inputs—Option 2

Decouple vertex input data from type used in shader



Per-Vertex Inputs—Option 2

Specifying vertex attribute indices in a shader

```
struct VertexInput {
    float4 position [[ attribute(0) ]];
    float3 normal [[ attribute(1) ]];
    half4 color [[ attribute(2) ]];
    half2 texcoord [[ attribute(3) ]];
};

vertex VertexOutput
my_vertex_shader(VertexInput v_in [[ stage_in ]], …)
```

Per-Vertex Inputs—Option 2

Specifying vertex attribute indices in a shader

```
struct VertexInput {
    float4 position [[ attribute(0) ]];
    float3 normal [[ attribute(1) ]];
    half4 color [[ attribute(2) ]];
    half2 texcoord [[ attribute(3) ]];
};

vertex VertexOutput
my_vertex_shader(VertexInput v_in [[ stage_in ]], …)
```

Per-Vertex Inputs—Option 2

Building the vertex descriptor

```
MTLVertexDescriptor* vertexDesc = [[MTLVertexDescriptor alloc] init];
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                 offset:0 vertexBufferIndex:0 atAttributeIndex:0]
```

Per-Vertex Inputs—Option 2

Building the vertex descriptor

```
MTLVertexDescriptor* vertexDesc = [[MTLVertexDescriptor alloc] init];
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                 offset:0 vertexBufferIndex:0 atAttributeIndex:0]
```

Per-Vertex Inputs—Option 2

Building the vertex descriptor

```
MTLVertexDescriptor* vertexDesc = [[MTLVertexDescriptor alloc] init];

[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                     offset:0 vertexBufferIndex:0 atAttributeIndex:0]
```

Per-Vertex Inputs—Option 2

Building the vertex descriptor

```
MTLVertexDescriptor* vertexDesc = [[MTLVertexDescriptor alloc] init];
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                 offset:0 vertexBufferIndex:0 atAttributeIndex:0]
```

Per-Vertex Inputs—Option 2

Building the vertex descriptor

```
MTLVertexDescriptor* vertexDesc = [[MTLVertexDescriptor alloc] init];

[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                 offset:0 vertexBufferIndex:0 atAttributeIndex:0]
```

Per-Vertex Inputs—Option 2

Building the vertex descriptor

```
MTLVertexDescriptor* vertexDesc = [[MTLVertexDescriptor alloc] init];
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                  offset:0 vertexBufferIndex:0 atAttributeIndex:0]
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                  offset:12 vertexBufferIndex:0 atAttributeIndex:1]
```

Per-Vertex Inputs—Option 2

Building the vertex descriptor

```
MTLVertexDescriptor* vertexDesc = [[MTLVertexDescriptor alloc] init];
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                 offset:0 vertexBufferIndex:0 atAttributeIndex:0]
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                 offset:12 vertexBufferIndex:0 atAttributeIndex:1]
[vertexDesc setVertexFormat:MTLVertexFormatUChar4Normalized
                 offset:24 vertexBufferIndex:0 atAttributeIndex:2]
```

Per-Vertex Inputs—Option 2

Building the vertex descriptor

```
MTLVertexDescriptor* vertexDesc = [[MTLVertexDescriptor alloc] init];
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                  offset:0 vertexBufferIndex:0 atAttributeIndex:0]
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                  offset:12 vertexBufferIndex:0 atAttributeIndex:1]
[vertexDesc setVertexFormat:MTLVertexFormatUChar4Normalized
                  offset:24 vertexBufferIndex:0 atAttributeIndex:2]
[vertexDesc setVertexFormat:MTLVertexFormatUShort2Normalized
                  offset:28 vertexBufferIndex:0 atAttributeIndex:3];
```

Per-Vertex Inputs—Option 2

Building the vertex descriptor

```
MTLVertexDescriptor* vertexDesc = [[MTLVertexDescriptor alloc] init];
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                 offset:0 vertexBufferIndex:0 atAttributeIndex:0]
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                 offset:12 vertexBufferIndex:0 atAttributeIndex:1]
[vertexDesc setVertexFormat:MTLVertexFormatUChar4Normalized
                 offset:24 vertexBufferIndex:0 atAttributeIndex:2]
[vertexDesc setVertexFormat:MTLVertexFormatUShort2Normalized
                 offset:28 vertexBufferIndex:0 atAttributeIndex:3];
[vertexDesc setStride:32 atVertexBufferIndex:0];
```

Per-Vertex Inputs—Option 2

Building the vertex descriptor

```
MTLVertexDescriptor* vertexDesc = [[MTLVertexDescriptor alloc] init];
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                 offset:0 vertexBufferIndex:0 atAttributeIndex:0]
[vertexDesc setVertexFormat:MTLVertexFormatFloat3
                 offset:12 vertexBufferIndex:0 atAttributeIndex:1]
[vertexDesc setVertexFormat:MTLVertexFormatUChar4Normalized
                 offset:24 vertexBufferIndex:0 atAttributeIndex:2]
[vertexDesc setVertexFormat:MTLVertexFormatUShort2Normalized
                 offset:28 vertexBufferIndex:0 atAttributeIndex:3];
[vertexDesc setStride:32 atVertexBufferIndex:0];
// add vertex descriptor to the MTLRenderPipelineDescriptor
pipelineDescriptor.vertexDescriptor = vertexDesc;
```

Per-Vertex Outputs

Two methods for writing vertex data

Per-Vertex Outputs

Return type of vertex shader

Per-Vertex Outputs

Return type of vertex shader

A float4 or a user-defined struct

Per-Vertex Outputs

Return type of vertex shader

A float4 or a user-defined struct

Elements of a user-defined struct

Per-Vertex Outputs

Return type of vertex shader

A float4 or a user-defined struct

Elements of a user-defined struct

- A scalar, vector, or matrix type

Per-Vertex Outputs

Return type of vertex shader

A float4 or a user-defined struct

Elements of a user-defined struct

- A scalar, vector, or matrix type
- Built-in variables

```
[[ position ]]
[[ point_size ]]
[[ clip_distance ]]
```

Per-Vertex Outputs

Return type of vertex shader

A float4 or a user-defined struct

Elements of a user-defined struct

- A scalar, vector, or matrix type
- Built-in variables

```
[[ position ]]
[[ point_size ]]
[[ clip_distance ]]
```

- Position must always be returned

Per-Vertex Outputs

Return type of vertex shader

```
struct VertexOutput {
    float4 pos [[ position ]];
    half4  color;
    float  pt [[ point_size ]];
    float2 texcoord;
}

vertex VertexOutput
my_vertex_shaderA(...)
{
}
```

Per-Vertex Outputs

Anyway you want, just the way you need…to write

Per-Vertex Outputs

Anyway you want, just the way you need…to write

Output to a buffer(s) using your vertex ID

Per-Vertex Outputs

Anyway you want, just the way you need…to write

Output to a buffer(s) using your vertex ID

```
struct VertexOutput {
    float4 pos;
    half4  color;
    float2 texcoord;
};

vertex void
my_vertex_shaderA(global VertexOutput* output_buffer [[ buffer(0) ]],
                  uint vid [[ vertex_id ]], ...)
{
    VertexOutput v_out;

    ...

    output_buffer[vid] = v_out;
}
```

Per-Vertex Outputs

Anyway you want, just the way you need…to write

Output to a buffer(s) using your vertex ID

```
struct VertexOutput {
    float4 pos;
    half4  color;
    float2 texcoord;
};

vertex void
my_vertex_shaderA(global VertexOutput* output_buffer [[ buffer(0) ]],
                  uint vid [[ vertex_id ]], ...)
{
    VertexOutput v_out;

    ...

    output_buffer[vid] = v_out;
}
```

Per-Vertex Outputs

Anyway you want, just the way you need…to write

Output to a buffer(s) using your vertex ID

```
struct VertexOutput {
    float4 pos;
    half4  color;
    float2 texcoord;
};

vertex void
my_vertex_shaderA(global VertexOutput* output_buffer [[ buffer(0) ]],
                  uint vid [[ vertex_id ]], ...)
{
    VertexOutput v_out;

    ...

    output_buffer[vid] = v_out;
}
```

Per-Vertex Outputs

Anyway you want, just the way you need…to write

Output to a buffer(s) using your vertex ID

```
struct VertexOutput {
    float4 pos;
    half4  color;
    float2 texcoord;
};

vertex void
my_vertex_shaderA(global VertexOutput* output_buffer [[ buffer(0) ]],
                  uint vid [[ vertex_id ]], ...)
{
    VertexOutput v_out;

    ...

    output_buffer[vid] = v_out;
}
```

Per-Vertex Outputs

Anyway you want, just the way you need…to write

Output to a buffer(s) using your vertex ID

```
struct VertexOutput {
    float4 pos;
    half4  color;
    float2 texcoord;
};

vertex void
my_vertex_shaderA(global VertexOutput* output_buffer [[ buffer(0) ]],
                  uint vid [[ vertex_id ]], ...)
{
    VertexOutput v_out;

    ...

    output_buffer[vid] = v_out;
}
```

Per-Fragment

Inputs and outputs

Per-Fragment Inputs

Output of a vertex shader

Per-Fragment Inputs

Output of a vertex shader

Declared with the [[`stage_in`]] qualifier

Per-Fragment Inputs

Output of a vertex shader

Declared with the [[`stage_in`]] qualifier

Built-in variables generated by the rasterizer

- Front facing
- Point coordinate
- Sample ID and sample mask

Per-Fragment Inputs

Output of a vertex shader

Declared with the [[`stage_in`]] qualifier

Built-in variables generated by the rasterizer

- Front facing
- Point coordinate
- Sample ID and sample mask

Frame-buffer color values

- For programmable blending

Per-Fragment Inputs

Output of a vertex shader

```
struct MyFragmentInput {
    half3 normal;
    float2 texcoord;
};

fragment float4
my_fragment_shader(
        MyFragmentInput fragIn [[ stage_in ]],
        bool is_front_face [[ front_facing ]],
        half4 fb_color [[ color(0) ]])
{
   ...
}
```

Per-Fragment Inputs

Output of a vertex shader

```
struct MyFragmentInput {
    half3 normal;
    float2 texcoord;
};

fragment float4
my_fragment_shader(
        MyFragmentInput fragIn [[ stage_in ]],
        bool is_front_face [[ front_facing ]],
        half4 fb_color [[ color(0) ]])
{
   ...
}
```

Per-Fragment Inputs

Output of a vertex shader

```
struct MyFragmentInput {
    half3 normal;
    float2 texcoord;
};

fragment float4
my_fragment_shader(
        MyFragmentInput fragIn [[ stage_in ]],
        bool is_front_face [[ front_facing ]],
        half4 fb_color [[ color(0) ]])
{
   ...
}
```

Per-Fragment Inputs

Output of a vertex shader

```
struct MyFragmentInput {
    half3 normal;
    float2 texcoord;
};

fragment float4
my_fragment_shader(
        MyFragmentInput fragIn [[ stage_in ]],
        bool is_front_face [[ front_facing ]],
        half4 fb_color [[ color(0) ]])
{
   ...
}
```

Per-Fragment Inputs

Output of a vertex shader

```
struct MyFragmentInput {
    half3 normal;
    float2 texcoord;
};

fragment float4
my_fragment_shader(
        MyFragmentInput fragIn [[ stage_in ]],
        bool is_front_face [[ front_facing ]],
        half4 fb_color [[ color(0) ]])
{
   ...
}
```

Per-Fragment Inputs

Output of a vertex shader

```
struct MyFragmentInput {
    half3 normal;
    float2 texcoord;
};

fragment float4
my_fragment_shader(
        MyFragmentInput fragIn [[ stage_in ]],
        bool is_front_face [[ front_facing ]],
        half4 fb_color [[ color(0) ]])
{
   ...
}
```

Per-Fragment Inputs

Output of a vertex shader

```
struct MyFragmentInput {
    half3 normal;
    float2 texcoord;
};

fragment float4
my_fragment_shader(
        MyFragmentInput fragIn [[ stage_in ]],
        bool is_front_face [[ front_facing ]],
        half4 fb_color [[ color(0) ]])
{
   ...
}
```

Per-Fragment Outputs

Return type of fragment shader

Per-Fragment Outputs

Return type of fragment shader

A scalar, vector, or user-defined struct

Per-Fragment Outputs

Return type of fragment shader

A scalar, vector, or user-defined struct

Color, depth, or sample mask

Per-Fragment Outputs

Return type of fragment shader

A scalar, vector, or user-defined struct

Color, depth, or sample mask

Identified with attributes

- [[`color(m)`]]
- [[`depth(qualifier)`]]
- [[`sample_mask`]]

Per-Fragment Outputs

Return type of fragment shader

Per-Fragment Outputs

Return type of fragment shader

```
fragment float4
my_fragment_shader(…)
```

Per-Fragment Outputs

Return type of fragment shader

```
fragment float4
my_fragment_shader(…)

struct MyFragmentOutput {
    half4 clrA  [[ color(0) ]];
    int4  clrB  [[ color(2) ]];
    uint4 clrC  [[ color(1) ]];
};

fragment MyFragmentOutput
my_fragment_shader(…)
{
    MyFragmentOutput v;
    ...
    return v;
}
```

Shader Signature Matching

A match made in heaven

Shader Signature Matching

Types match

Shader Signature Matching

Types match

```
struct VertexOutput {
    float4 pos [[ position ]];
    float3 normal;
    float2 texcoord;
};
```

Shader Signature Matching

Types match

```
struct VertexOutput {
    float4 pos [[ position ]];
    float3 normal;
    float2 texcoord;
};
```

```
vertex VertexOutput
my_vertex_shader(…)
{
    VertexOutput v;
    ...
    return v;
}
```

```
fragment float4
my_fragment_shader(VertexOutput frag_in [[ stage_in ]],…)
{
    float4 f;
    ...
    return f;
}
```

Shader Signature Matching

Flexible pairing

```
struct VertexOutput {
  float4 pos [[ position ]];
  float3 normal [[ user(N)]];
  float2 texcoord [[ user(T) ]];
};
```

```
struct FragmentInput {
  float4 pos [[ position ]];
  float2 texcoord [[ user(T) ]];
};
```

Shader Signature Matching

Flexible pairing

```
struct VertexOutput {
  float4 pos [[ position ]];
  float3 normal [[ user(N)]];
  float2 texcoord [[ user(T) ]];
};
```

```
struct FragmentInput {
  float4 pos [[ position ]];
  float2 texcoord [[ user(T) ]];
};
```

Shader Signature Matching

Flexible pairing

```
struct VertexOutput {
  float4 pos [[ position ]];
  float3 normal [[ user(N)]];
  float2 texcoord [[ user(T) ]];
};
```

```
struct FragmentInput {
  float4 pos [[ position ]];
  float2 texcoord [[ user(T) ]];
};
```

Shader Signature Matching

Flexible pairing

```
struct VertexOutput {
  float4 pos [[ position ]];
  float3 normal [[ user(N)]];
  float2 texcoord [[ user(T) ]];
};
```

```
struct FragmentInput {
  float4 pos [[ position ]];
  float2 texcoord [[ user(T) ]];
};
```

```
vertex VertexOutput
my_vertex_shader(…)
{
    VertexOutput v;
    ...
    return v;
}
```

```
fragment float4
my_fragment_shader(FragmentInput frag_in [[ stage_in ]],…)
{
    float4 f;
    ...
    return f;
}
```

Math in Shaders

Fast or precise, maybe both

Math

Math

By default, all math operations in fast mode

Math

By default, all math operations in fast mode

Why would you want to choose precise mode?

Math

By default, all math operations in fast mode

Why would you want to choose precise mode?

- Handling of NaNs is undefined in fast mode
  - e.g., how should clamp (NaN, min, max) behave?

Math

By default, all math operations in fast mode

Why would you want to choose precise mode?

- Handling of NaNs is undefined in fast mode
  - e.g., how should clamp (NaN, min, max) behave?
- Math functions only operate over a limited range in fast mode

Math

By default, all math operations in fast mode

Why would you want to choose precise mode?

- Handling of NaNs is undefined in fast mode
  - e.g., how should clamp (NaN, min, max) behave?
- Math functions only operate over a limited range in fast mode

Compiler option -fno-fast-math to change default to precise math

Math

By default, all math operations in fast mode

Why would you want to choose precise mode?

- Handling of NaNs is undefined in fast mode
  - e.g., how should clamp (NaN, min, max) behave?
- Math functions only operate over a limited range in fast mode

Compiler option -fno-fast-math to change default to precise math

- Be careful as this may impact performance of your shader

Math

Precise math in fast mode

Math

Precise math in fast mode

Nested name spaces—metal::precise and metal::fast

Math

Precise math in fast mode

Nested name spaces—metal::precise and metal::fast

Use explicit math function name

- precise::clamp, precise::sin

Metal Standard Library

Quite a nice list of functions if I do say so myself

Metal Standard Library Functions

Metal Standard Library Functions

Common Functions
T clamp(T x, T minval, T maxval)
T mix(T x, T y, T a)
T saturate(T x)
T sign(T x)
T smoothstep(T edge0, T edge1, T x)
T step(T edge, T x)
Integer Functions
T abs(T x)
T_u absdiff(T x, T y)
T addsat(T x, T y)
T clamp(T x, T minval, T maxval)
T clz(T x)
T ctz(T x)
T hadd(T x, T y)
T madhi(T a, T b, T c)
T madsat(T a, T b, T c)
T max(T x, T y)
T min(T x, T y)
T mulhi(T x, T y)
T popcount(T x)
T rhadd(T x, T y)
T rotate(T v, T i)
T subsat(T x, T y)
Relational Functions
bool all(T_b x)
bool any(T_b x)
T_b isfinite(T x)
T_b isinf(T x)
T_b isnan(T x)
T_b isnormal(T x)
T_b isordered(T x, T y)
T_b isunordered(T x, T y)
T_b not(T_b x)
T select(T a, T b, T_b c)
T_i select(T_i a, T_i b, T_b c)
T_b signbit(T x)
Math Functions
T acos(T x)
T acosh(T x)
T asin(T x)

Math Functions contd…
T atanh(T x)
T ceil(T x)
T copysign(T x, T y)
T cos(T x)
T cosh(T x)
T exp(T x)
T exp2(T x)
T exp10(T x)
T fabs(T x)
T abs(T x)
T fdim(T x, T y)
T floor(T x)
T fmax(T x, T y)
T max(T x, T y)
T fmin(T x, T y)
T min(T x, T y)
T fmod(T x, T y)
T fract(T x)
T frexp(T x, Ti& exponent)
Ti ilogb(T x)
T ldexp(T x, Ti k)
T log(T x)
T log2(T x)
T log10(T x)
T modf(T x, T& intval)
T pow(T x, T y)
T powr(T x, T y)
T rint(T x)
T round(T x)
T rsqrt(T x)
T sin(T x)
T sincos(T x, T& cosval)
T sinh(T x)
T sqrt(T x)
T tan(T x)
T tanh(T x)
T trunc(T x)
Geometric Functions
T cross(T x, T y)
T_s distance(T x, T y)

Geometric Functions contd…
T_s length(T x)
T_s length_squared(T x)
T normalize(T x)
T reflect(T I, T N)
T refract(T I, T N, T_s eta)
Compute Functions
void work_group_barrier(mem_flags)
Fragment Functions - Derivatives
T dfdx(T p)
T dfdy(T p)
T fwidth(T p)
Fragment Functions - Samples
uint get_num_samples()
float2 get_sample_position(uint indx)
Fragment Functions - Flow Control
void discard_fragment(void)
Unpack Functions
float4 unpack_unorm4x8_to_float(uint x)
float4 unpack_snorm4x8_to_float(uint x)
half4 unpack_unorm4x8_to_half(uint x)
half4 unpack_snorm4x8_to_half(uint x)
float4 unpack_unorm4x8_srgb_to_float(uint x)
half4 unpack_unorm4x8_srgb_to_half(uint x)
float2 unpack_unorm2x16_to_float(uint x)
float2 unpack_snorm2x16_to_float(uint x)
half2 unapck_unorm2x16_to_half(uint x)
half2 unpack_snorm2x16_to_half(uint x)
float4 unpack_unorm10a2_to_float(uint x)
float3 unpack_unorm565_to_float(ushort x)
half4 unpack_unorm10a2_to_half(uint x)
half3 unpck_unorm565_to_half(ushort x)
Pack Functions
uint pack_float_to_unorm4x8(float4 x)
uint pack_float_to_snorm4x8(float4 x)
uint pack_half_to_unorm4x8(half4 x)
uint pack_half_tosnorm4x8(half4 x)
uint pack_float_to_srgb_unorm4x8(float4 x)
uint pack_half_to_srgb_unorm4x8(half4 x)
uint pack_float_to_unorm2x16(float2 x)
uint pack_float_to_snorm2x16(float2 x)

Pack Functions contd…
ushort pack_float_to_unorm565(float3 x)
uint pack_half_to_unorm10a2(half4 x)
ushort pack_half_to_unorm565(half3 x)
Atomic Functions
void atomic_store_explicit(…)
void atomic_load_explicit(…)
void atomic_exchange_explicit(…)
void atomic_compare_exchange_weak_explicit(…)
void atomic_fetch_key_explicit(…)
Texture Functions
T_v sample(sampler s, float*n* coord, int*n* offset=0)
T_v sample(sampler s, float*n* coord, uint array, int*n* offset=0)
T_v sample(sampler s, float*n* coord,
lod_options options, int*n* offset=0)
T_v sample(sampler s, float*n* coord, uint array,
lod_options options, int*n* offset=0)
T_v read(uint*n* coord, uint lod=0)
T_v read(uint*n* coord, uint array, uint lod=0)
void write(T_v color, uint*n* coord, uint lod=0)
void write(T_v color, uint*n* coord, uint array, uint lod=0)
T_v gather(sampler s, floatn coord,
int2 offset=0, component c=component::x)
T_v gather(sampler s, float2 coord, uint array,
int2 offset=0, component c=component::x)
T sample_compare(sampler s, float2 coord,
float compare-val, int2 offset=0)
T sample_compare(sampler s, float2 coord,
float compare-val, lod_options options,
int2 offset=0)
T sample_compare(sampler s, float2 coord, uint array,
float compare-val, int2 offset=0)
T sample_compare(sampler s, float2 coord, uint array,
float compare-val, lod_options options,
int2 offset=0)
T_v gather_compare(sampler s, float2 coord,
float compare_val, int2 offset=0)
T_v gather_compare(sampler s, float2 coord, uint array,
float compare_val, int2 offset=0)
uint get_width(uint lod=0)
uint get_height(uint lod=0)

Metal Fundamentals

Building a Metal application

- Initialization
- Drawing
- Uniforms and synchronization

Metal shading language

- Writing shaders in Metal
- Data types in Metal
- Shader inputs, outputs, and matching rules

Metal Fundamentals

Building a Metal application

- Initialization
- Drawing
- Uniforms and synchronization

Metal shading language

- Writing shaders in Metal
- Data types in Metal
- Shader inputs, outputs, and matching rules

Call to action

- Amaze us with how you use Metal
- Let us know how we can improve Metal

More Information

Filip Iliescu
Graphics and Games Technologies Evangelist
filiescu@apple.com

Allan Schaffer
Graphics and Games Technologies Evangelist
aschaffer@apple.com

Documentation
http://developer.apple.com

Apple Developer Forums
http://devforums.apple.com

Related Sessions

What's New in the Accelerate Framework	Nob Hill	Tuesday 10:15AM
Working with Metal—Overview	Pacific Heights	Wednesday 9:00AM
Working with Metal—Advanced	Pacific Heights	Wednesday 11:30AM

Labs

Metal Lab	Graphics and Games Lab A	Wednesday 2:00PM
Metal Lab	Graphics and Games Lab B	Thursday 10:15AM

WWDC14