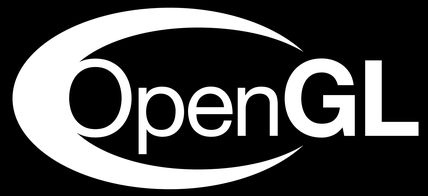
#WWDC19

Bringing OpenGL Apps to Metal

Lionel Lemarié GPU Software
Max Christ GPU Software
Sarah Clawson GPU Software

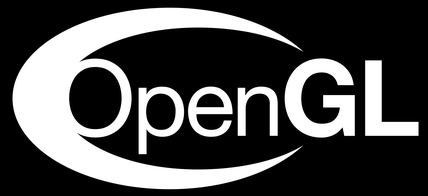
Reminder

Legacy APIs are deprecated

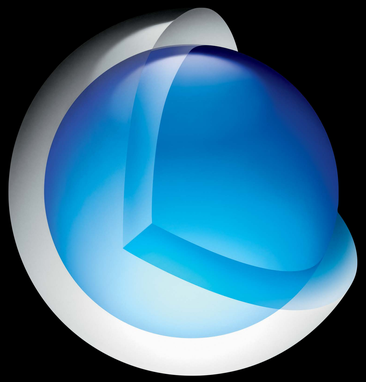


Reminder

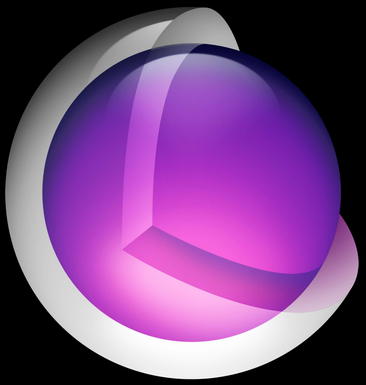
Legacy APIs are deprecated

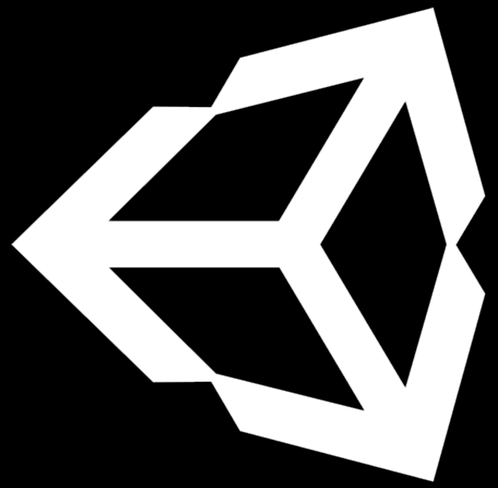






280




lumberyard BY amazon

Benefits of Metal

Benefits of Metal

Unified API for graphics and compute

Benefits of Metal

Unified API for graphics and compute

Multithreading

Benefits of Metal

Unified API for graphics and compute

Multithreading

Reduced overhead

Benefits of Metal

Unified API for graphics and compute

Multithreading

Reduced overhead

Benefits of Metal

Unified API for graphics and compute

Multithreading

Reduced overhead

Shading language based on C++

Benefits of Metal

Unified API for graphics and compute

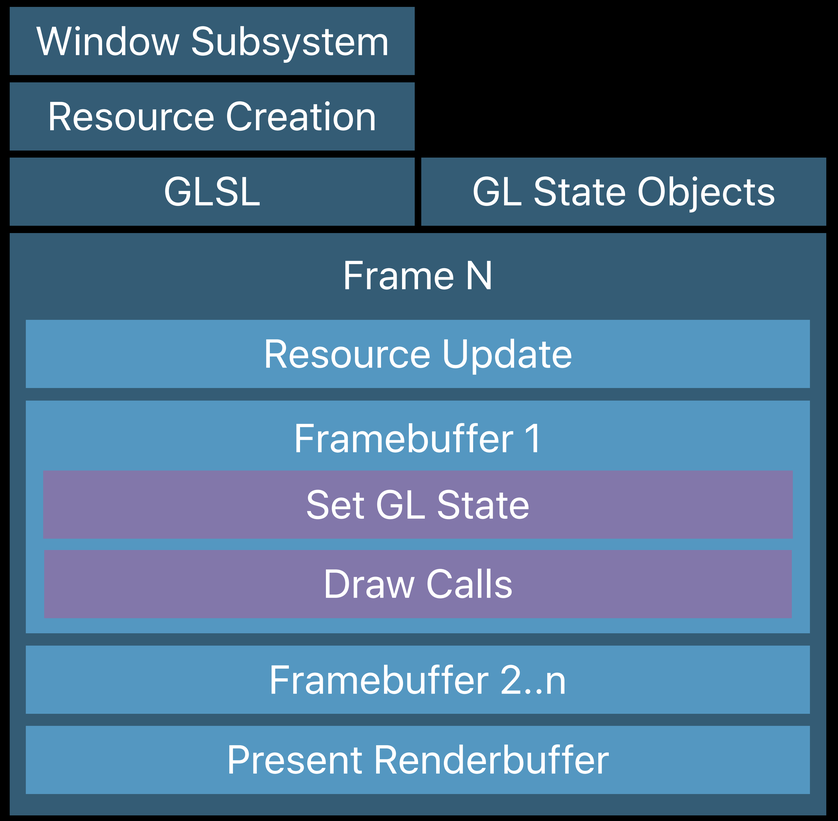
Multithreading

Reduced overhead

Shading language based on C++

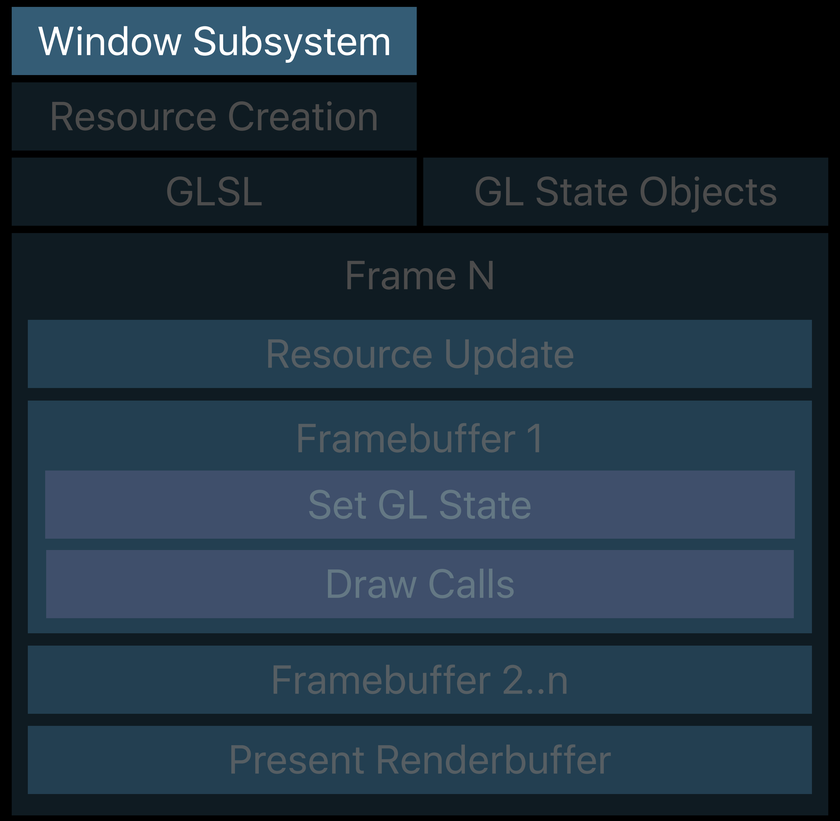
Deeply integrated tools

Life of a Graphics App



OpenGL

Life of a Graphics App



OpenGL

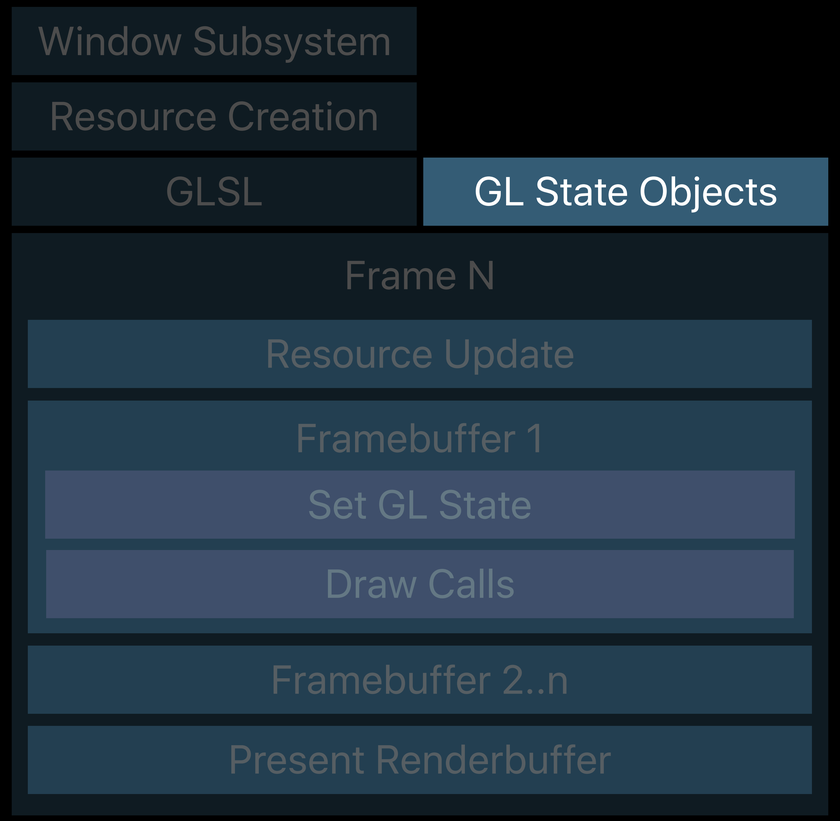
Life of a Graphics App

OpenGL

Life of a Graphics App

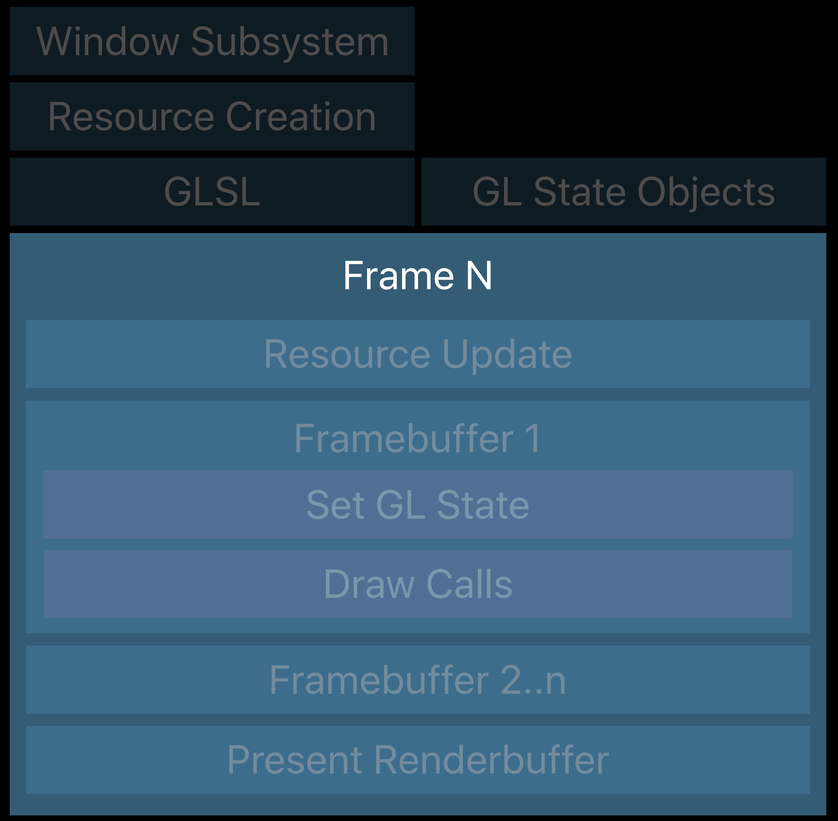
OpenGL

Life of a Graphics App



OpenGL

Life of a Graphics App

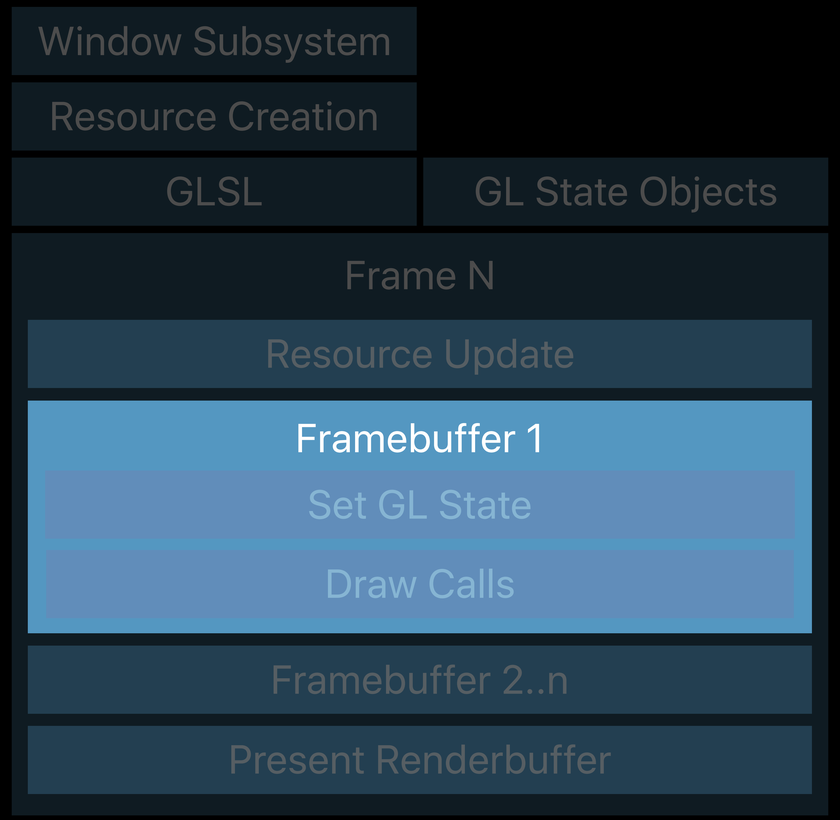


OpenGL

Life of a Graphics App

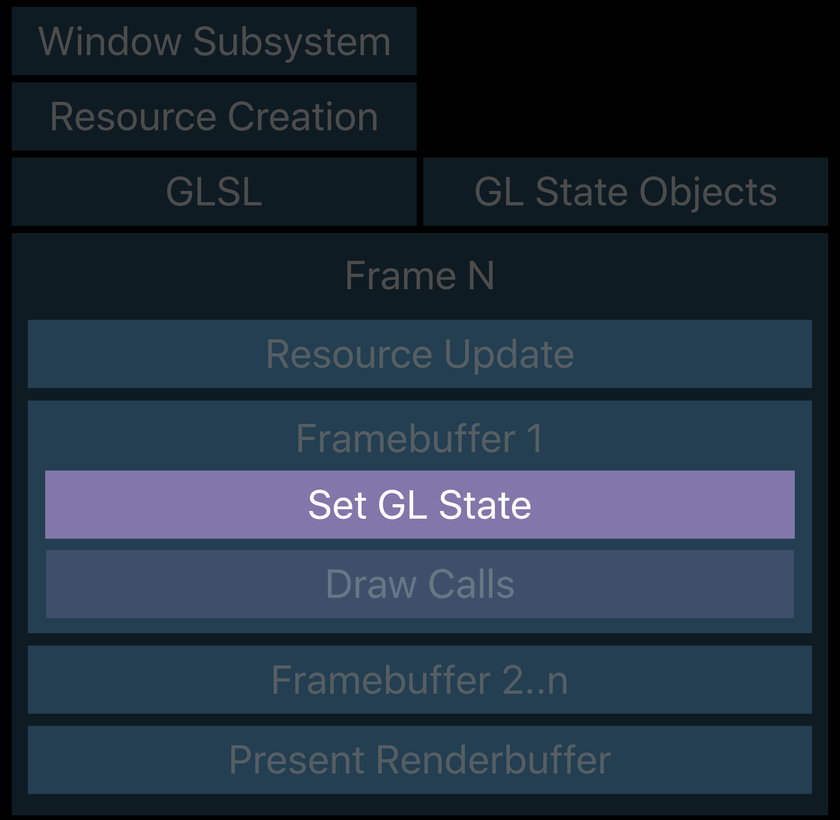
OpenGL

Life of a Graphics App



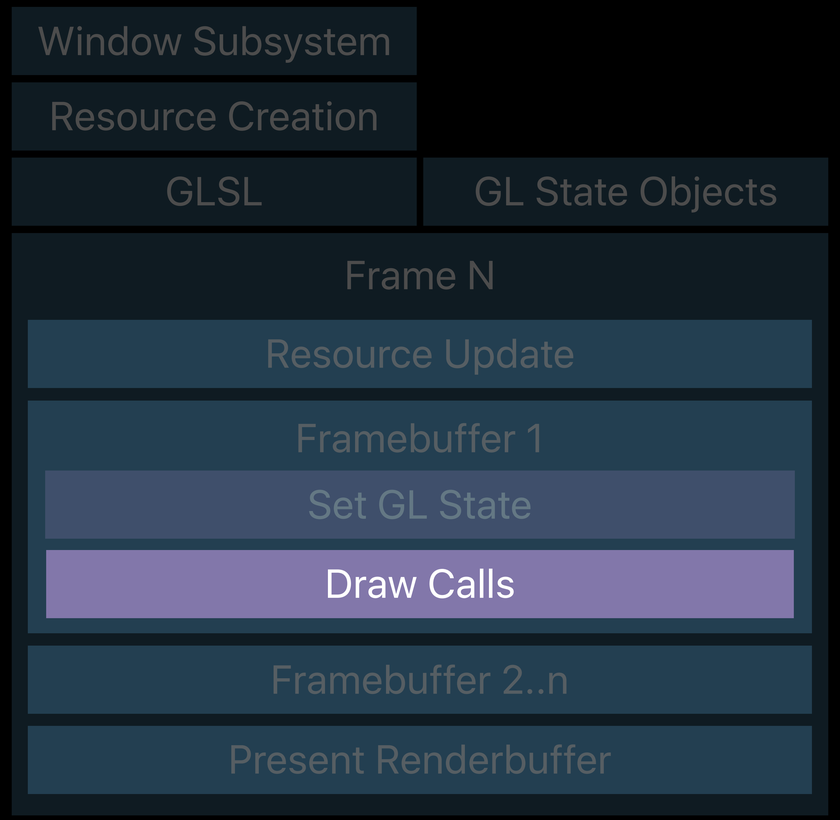
OpenGL

Life of a Graphics App



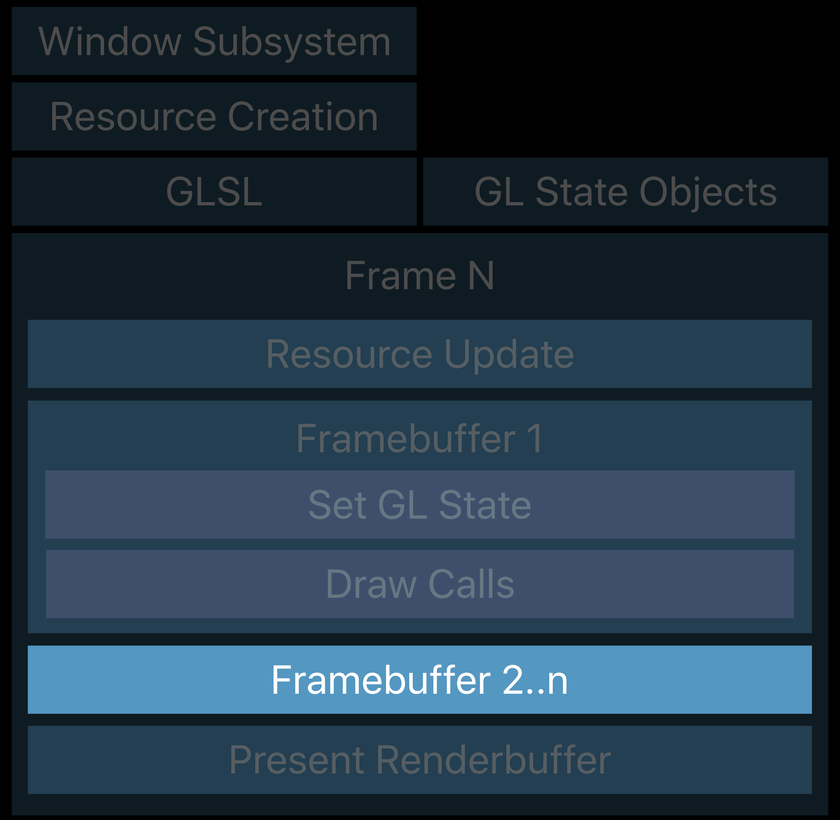
OpenGL

Life of a Graphics App



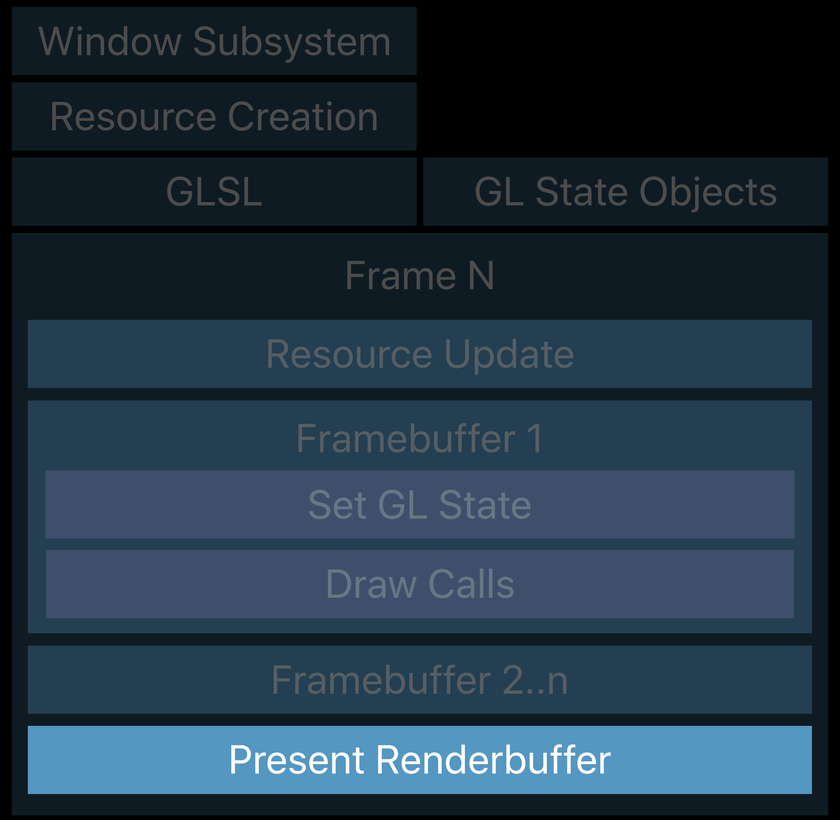
OpenGL

Life of a Graphics App



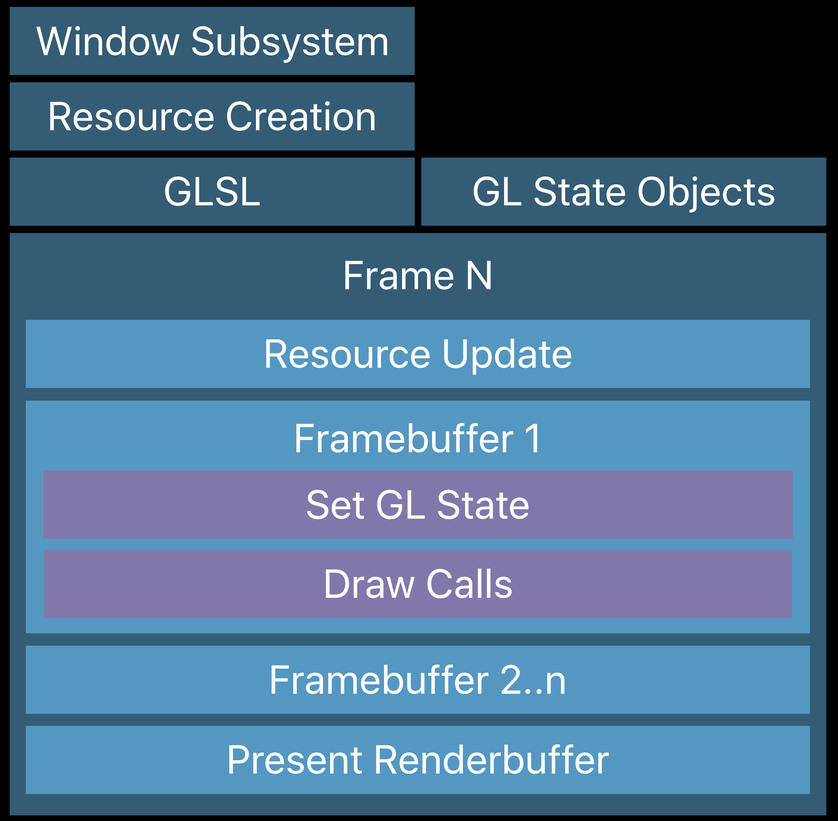
OpenGL

Life of a Graphics App



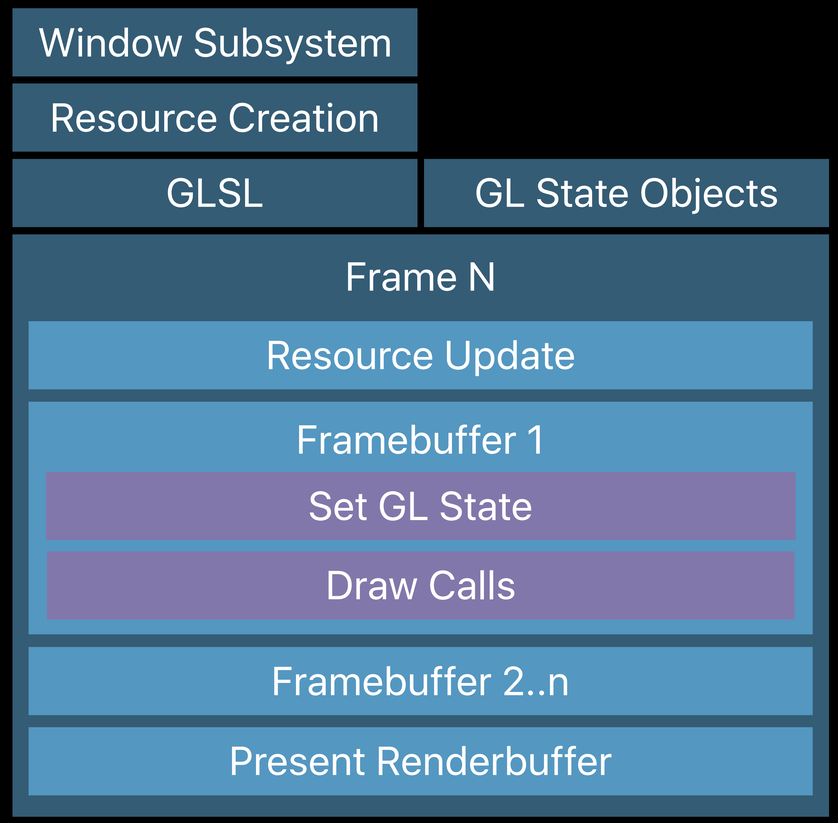
OpenGL

Life of a Graphics App

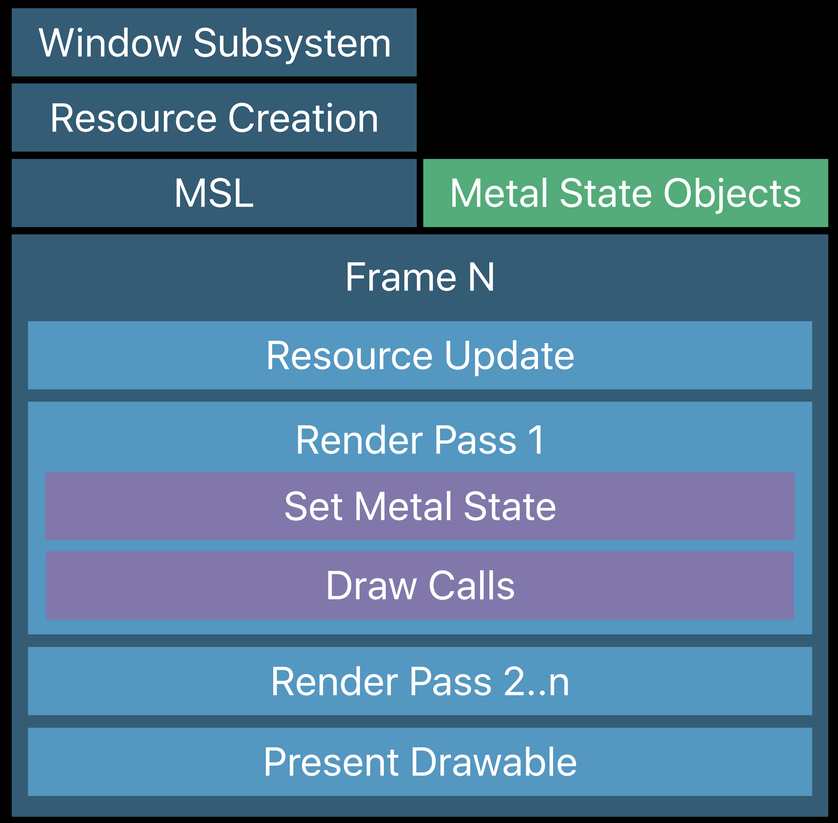


OpenGL

Life of a Graphics App

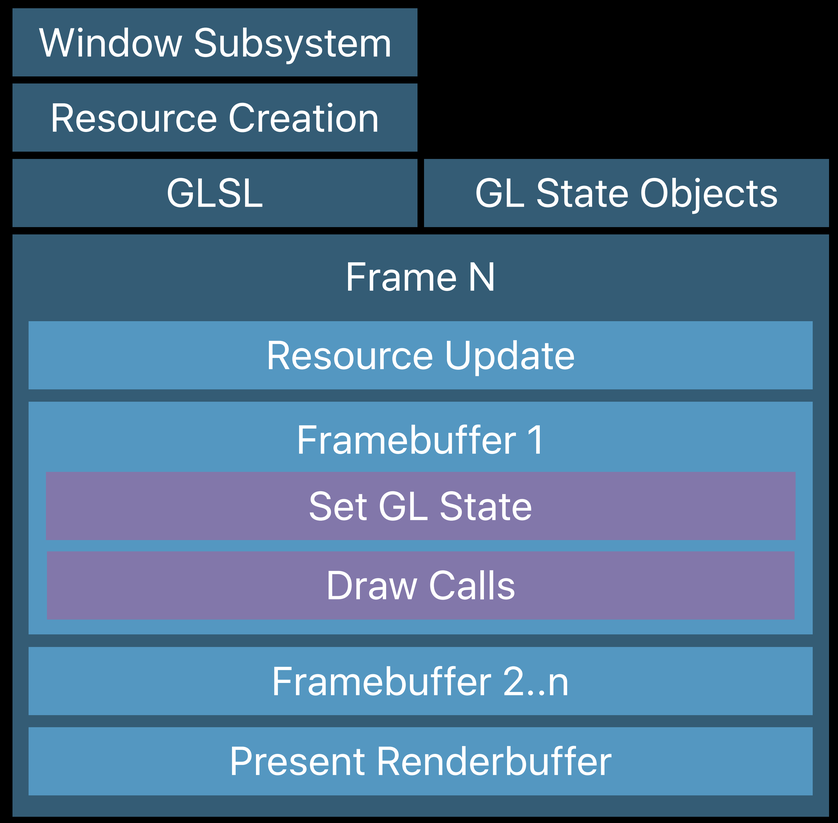


OpenGL

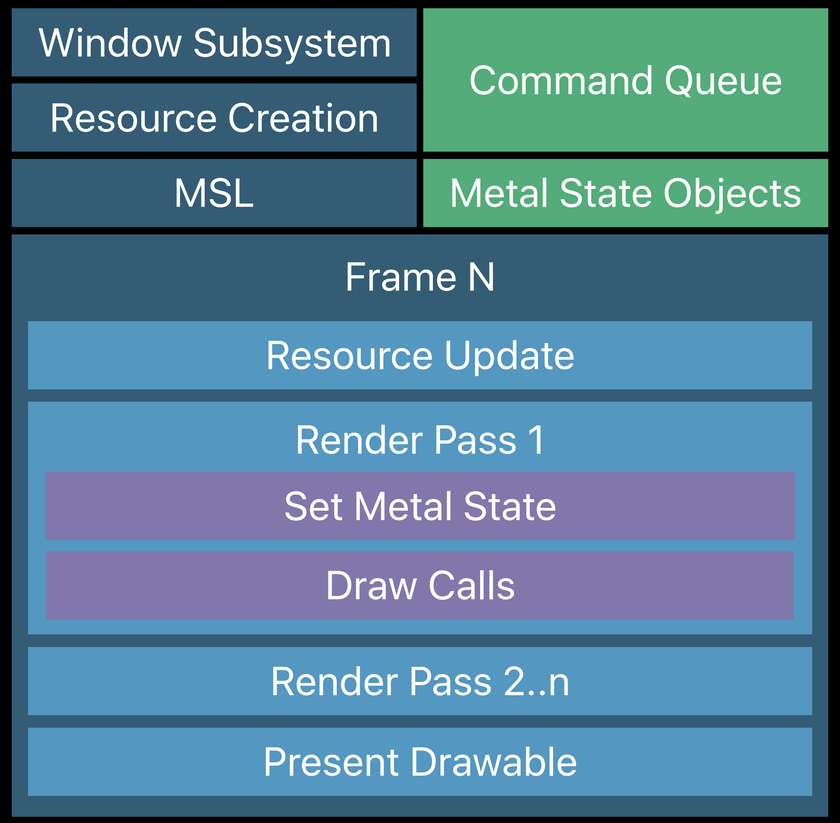


Metal

Life of a Graphics App



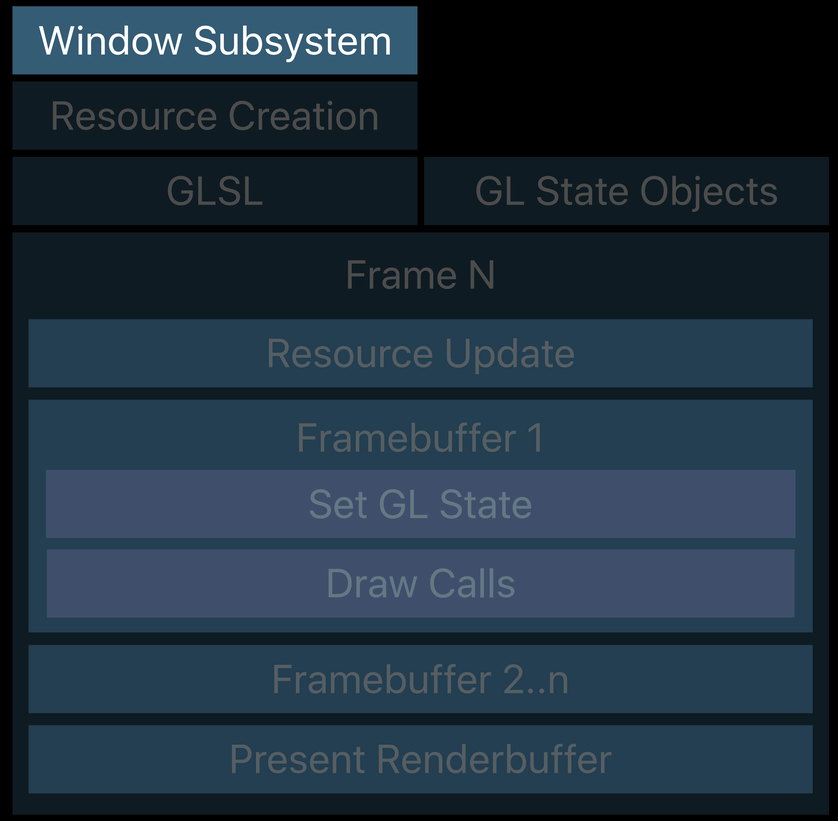
OpenGL



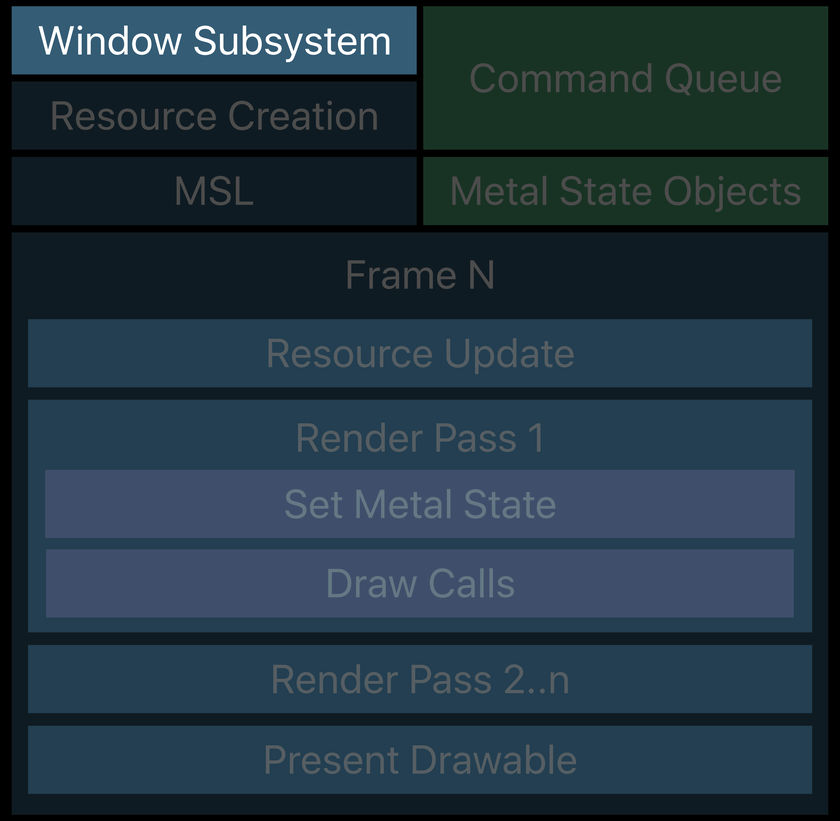
Metal

Life of a Graphics App

Window subsystem



OpenGL



Metal

Window Subsystem

Application is required to set up and present a drawing surface

Views and view delegates manage the underlying windowing system

Window Subsystem

OpenGL	Metal
NSOpenGLView GLKView	MTKView
CAEAGLLayer	CAMetalLayer

Window Subsystem

OpenGL	Metal
NSOpenGLView GLKView	MTKView
CAEAGLLayer	CAMetalLayer

Window Subsystem

OpenGL	Metal
NSOpenGLView GLKView	MTKView
CAEAGLLayer	CAMetalLayer

```
// GLKViewDelegate callback
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect;
```

OpenGL

Metal

```
// MTKViewDelegate callbacks
- (void) mtkView:(MTKView *)view drawableSizeWillChange:(CGSize)size;
- (void) drawInMTKView:(nonnull MTKView *)view;
```

```
// GLKViewDelegate callback
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect;
```

OpenGL

Metal

```
// MTKViewDelegate callbacks
- (void) mtkView:(MTKView *)view drawableSizeWillChange:(CGSize)size;
- (void) drawInMTKView:(nonnull MTKView *)view;
```

```
// GLKViewDelegate callback
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect;
```

OpenGL

Metal

```
// MTKViewDelegate callbacks
- (void) mtkView:(MTKView *)view drawableSizeWillChange:(CGSize)size;
- (void) drawInMTKView:(nonnull MTKView *)view;
```

```
// GLKViewDelegate callback
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect;
```

OpenGL

Metal

```
// MTKViewDelegate callbacks
- (void) mtkView:(MTKView *)view drawableSizeWillChange:(CGSize)size;
- (void) drawInMTKView:(nonnull MTKView *)view;
```

```
CAEAGLLayer *eaglLayer = (CAEAGLLayer *)[self layer];
[eaglLayer setOpaque:YES];

// Set other common layer properties
[eaglLayer setDrawableProperties:@{ kEAGLDrawablePropertyColorFormat,
kEAGLColorFormatSRGBA8 }];
```

OpenGL

Metal

```
CAMetalLayer *metalLayer = (CAMetalLayer *)[self layer];
[metalLayer setOpaque:YES];

// Set other common layer properties
[metalLayer setDevice:device]; // MTLCreateSystemDefaultDevice()
[metalLayer setPixelFormat:MTLPixelFormatBGRA8Unorm_sRGB];
[metalLayer setFramebufferOnly:YES]; // Specify NO if framebuffer is ever read from
```

```objc
CAEAGLLayer *eaglLayer = (CAEAGLLayer *)[self layer];
[eaglLayer setOpaque:YES];

// Set other common layer properties
[eaglLayer setDrawableProperties:@{ kEAGLDrawablePropertyColorFormat,
kEAGLColorFormatSRGBA8 }];
```

OpenGL

Metal

```objc
CAMetalLayer *metalLayer = (CAMetalLayer *)[self layer];
[metalLayer setOpaque:YES];

// Set other common layer properties
[metalLayer setDevice:device]; // MTLCreateSystemDefaultDevice()
[metalLayer setPixelFormat:MTLPixelFormatBGRA8Unorm_sRGB];
[metalLayer setFramebufferOnly:YES]; // Specify NO if framebuffer is ever read from
```

```
CAEAGLLayer *eaglLayer = (CAEAGLLayer *)[self layer];
[eaglLayer setOpaque:YES];

// Set other common layer properties
[eaglLayer setDrawableProperties:@{ kEAGLDrawablePropertyColorFormat,
kEAGLColorFormatSRGBA8 }];
```

OpenGL

Metal

```
CAMetalLayer *metalLayer = (CAMetalLayer *)[self layer];
[metalLayer setOpaque:YES];

// Set other common layer properties
[metalLayer setDevice:device]; // MTLCreateSystemDefaultDevice()
[metalLayer setPixelFormat:MTLPixelFormatBGRA8Unorm_sRGB];
[metalLayer setFramebufferOnly:YES]; // Specify NO if framebuffer is ever read from
```

```objc
CAEAGLLayer *eaglLayer = (CAEAGLLayer *)[self layer];
[eaglLayer setOpaque:YES];

// Set other common layer properties
[eaglLayer setDrawableProperties:@{ kEAGLDrawablePropertyColorFormat,
kEAGLColorFormatSRGBA8 }];
```

OpenGL

Metal

```objc
CAMetalLayer *metalLayer = (CAMetalLayer *)[self layer];
[metalLayer setOpaque:YES];

// Set other common layer properties
[metalLayer setDevice:device]; // MTLCreateSystemDefaultDevice()
[metalLayer setPixelFormat:MTLPixelFormatBGRA8Unorm_sRGB];
[metalLayer setFramebufferOnly:YES]; // Specify NO if framebuffer is ever read from
```

```objc
CAEAGLLayer *eaglLayer = (CAEAGLLayer *)[self layer];
[eaglLayer setOpaque:YES];

// Set other common layer properties
[eaglLayer setDrawableProperties:@{ kEAGLDrawablePropertyColorFormat,
kEAGLColorFormatSRGBA8 }];
```

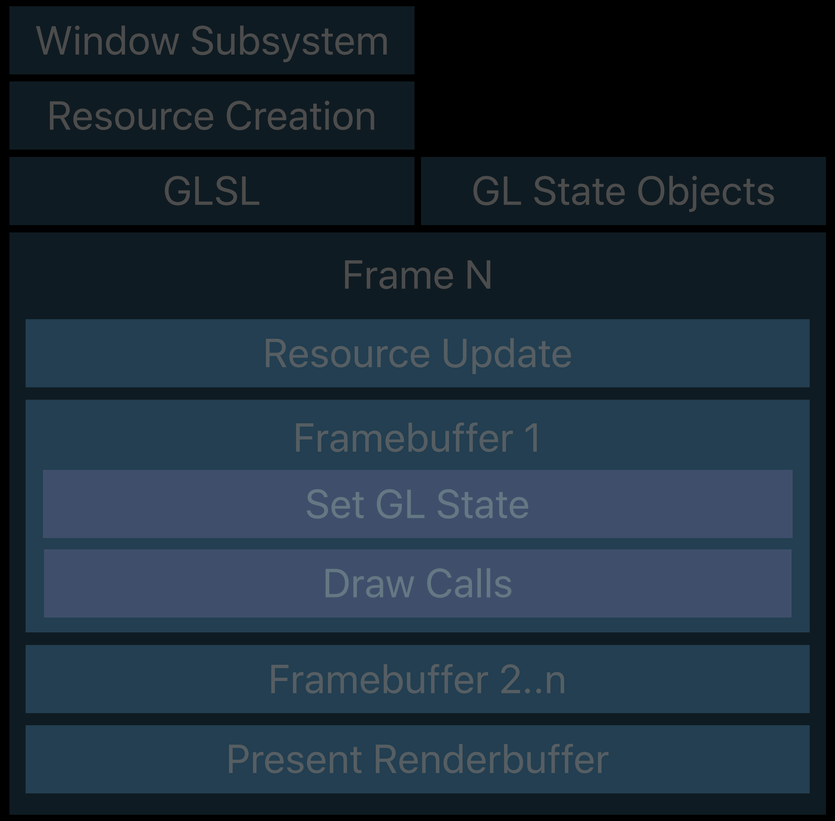
OpenGL

Metal

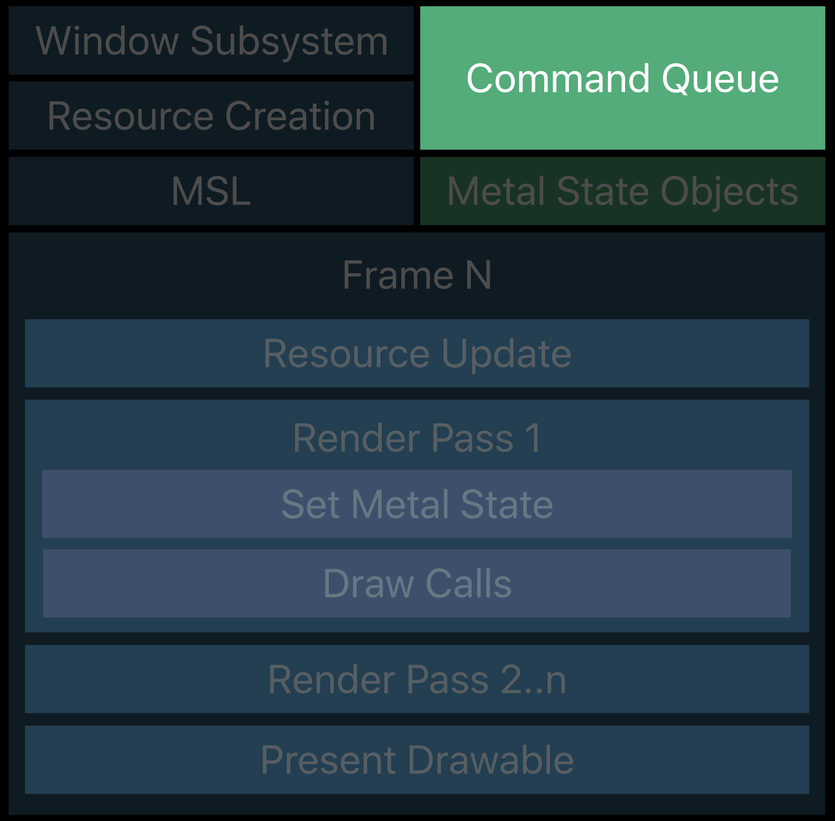
```objc
CAMetalLayer *metalLayer = (CAMetalLayer *)[self layer];
[metalLayer setOpaque:YES];

// Set other common layer properties
[metalLayer setDevice:device]; // MTLCreateSystemDefaultDevice()
[metalLayer setPixelFormat:MTLPixelFormatBGRA8Unorm_sRGB];
[metalLayer setFramebufferOnly:YES]; // Specify NO if framebuffer is ever read from
```

Life of a Graphics App

Command submission



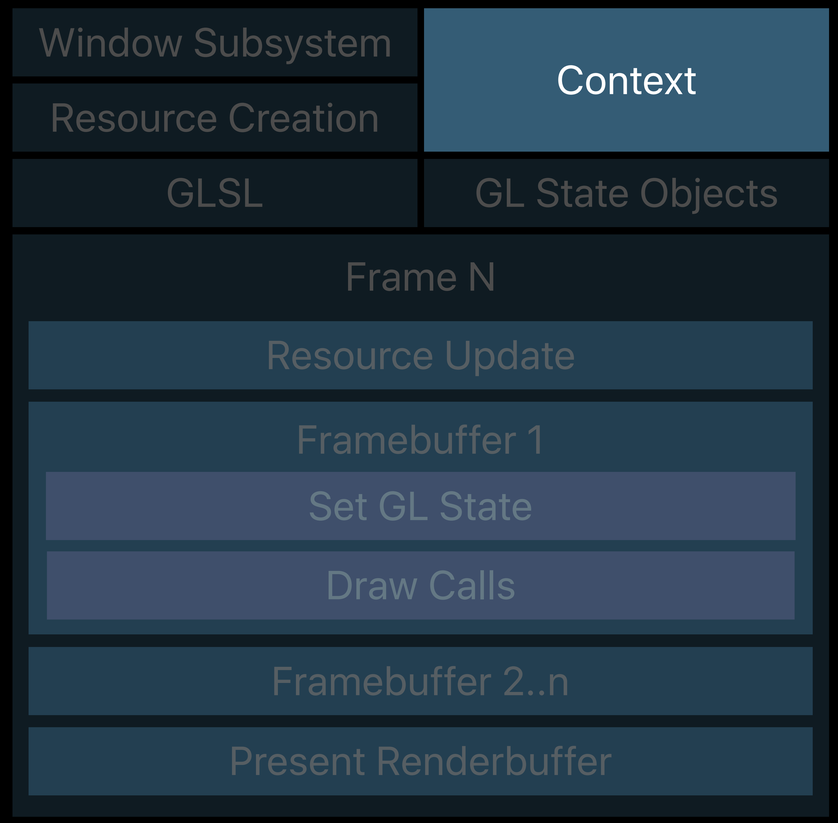
OpenGL



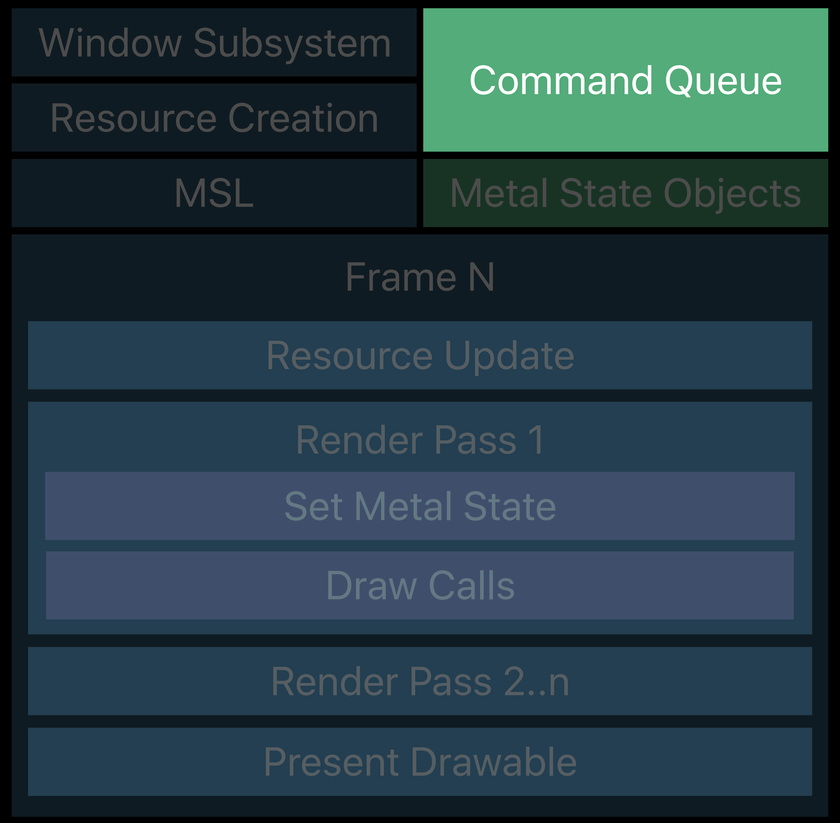
Metal

Life of a Graphics App

Command submission



OpenGL

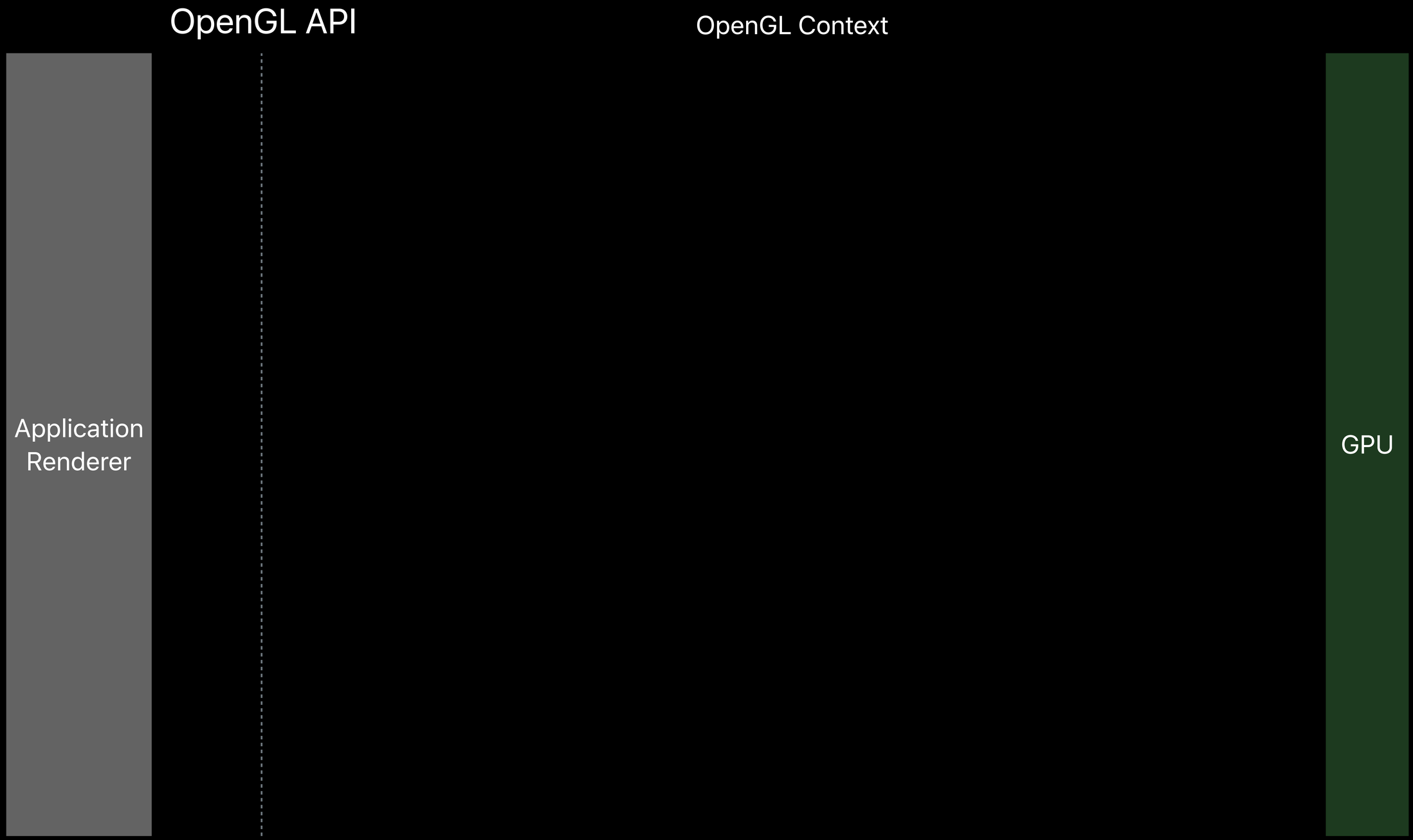


Metal

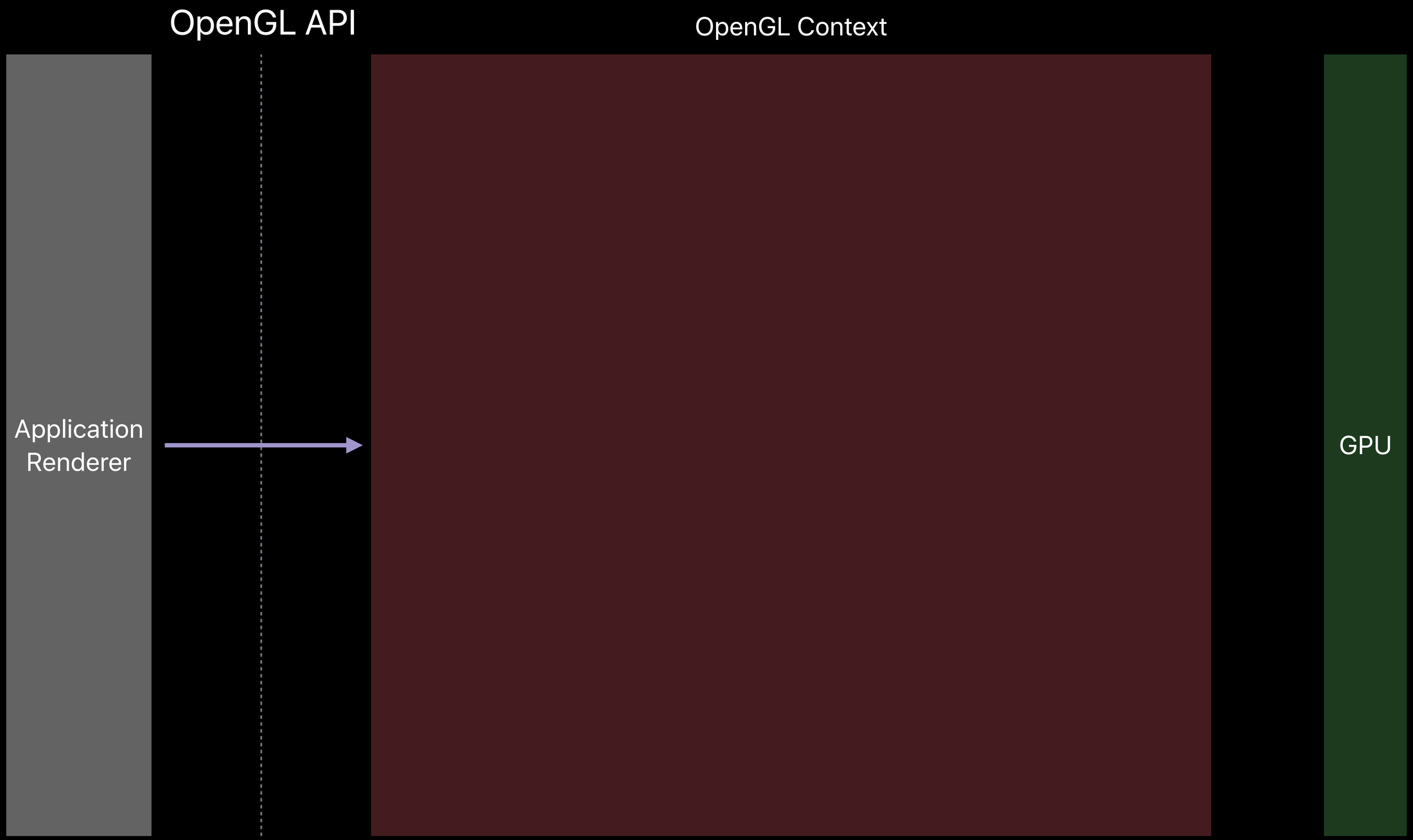
Submitting Work to the GPU

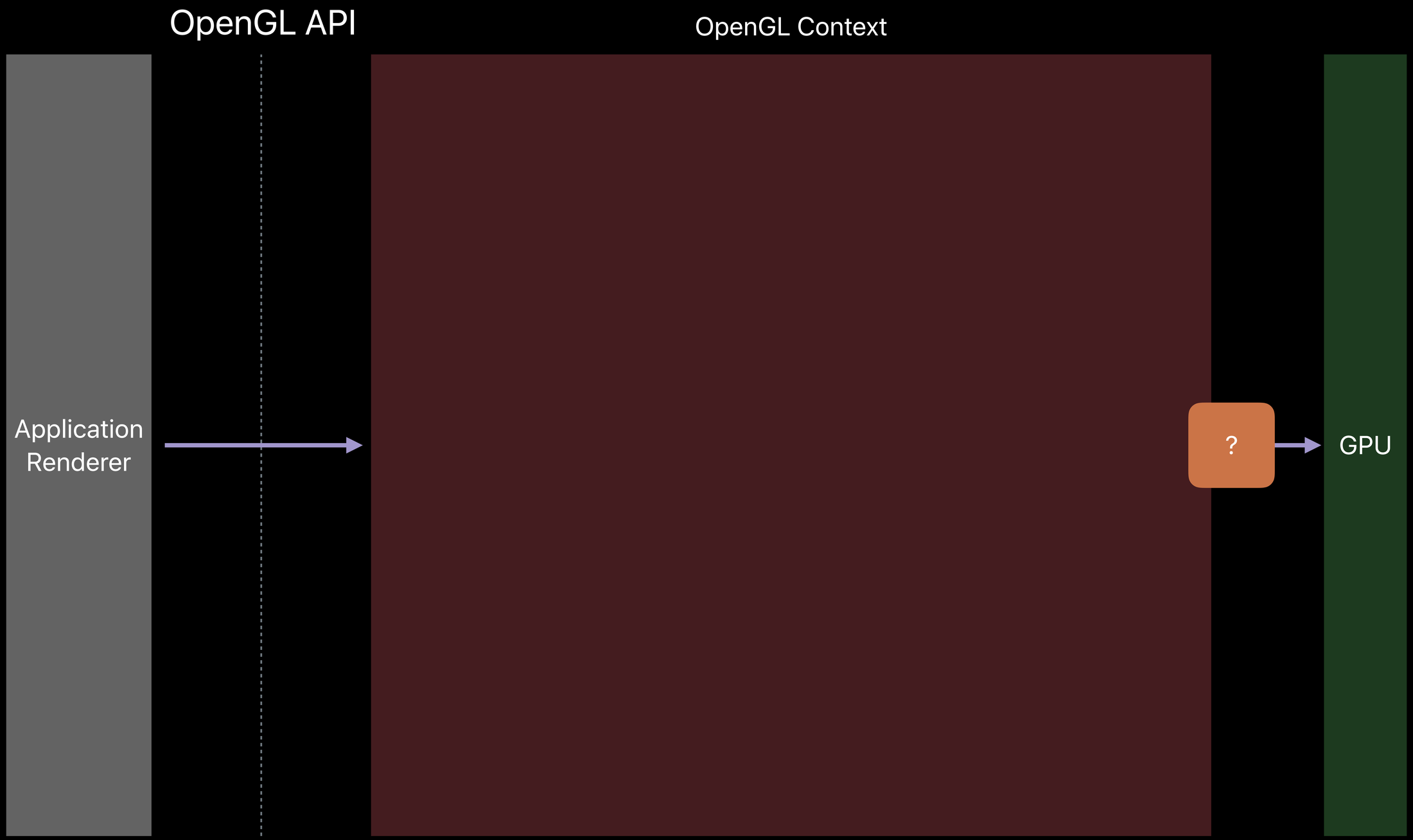
OpenGL — Implicit API

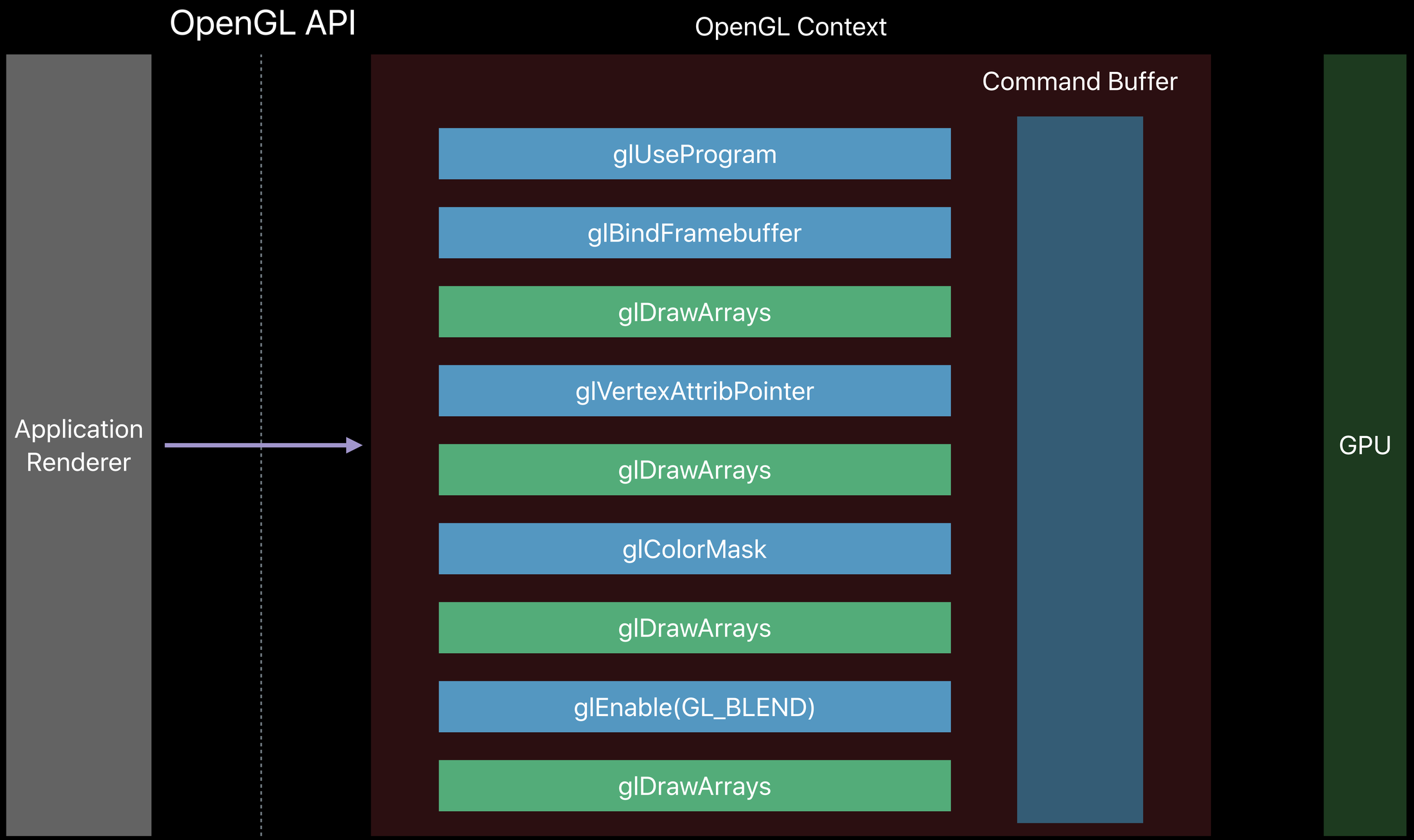
- Few mechanisms to enforce GPU execution
- OpenGL context takes the lead

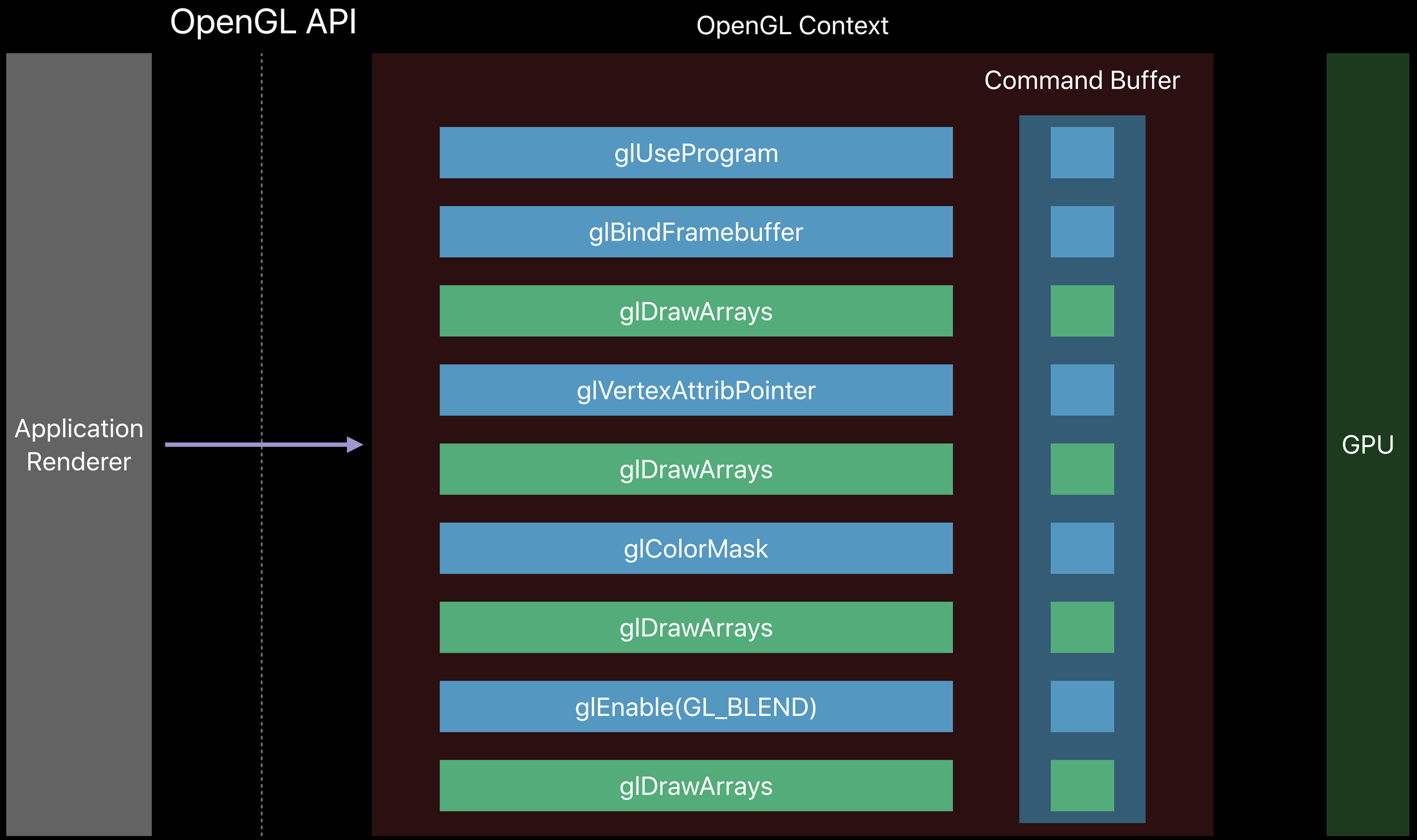

OpenGL API
OpenGL Context
Application
Renderer
GPU

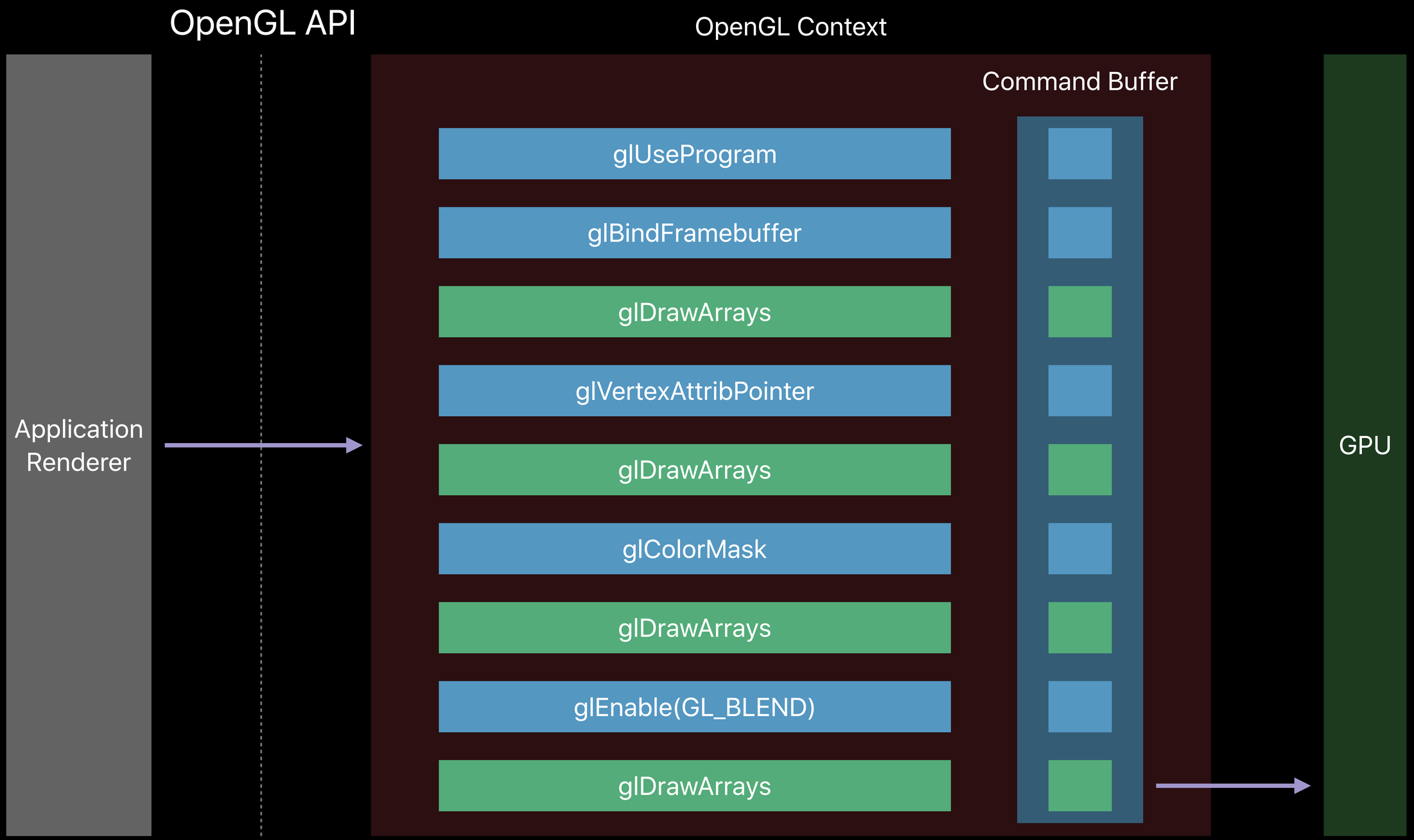

OpenGL API
OpenGL Context
Application
Renderer
[[EAGLContext alloc] initWithAPI:
kEAGLRenderingAPIOpenGLES3]
GPU

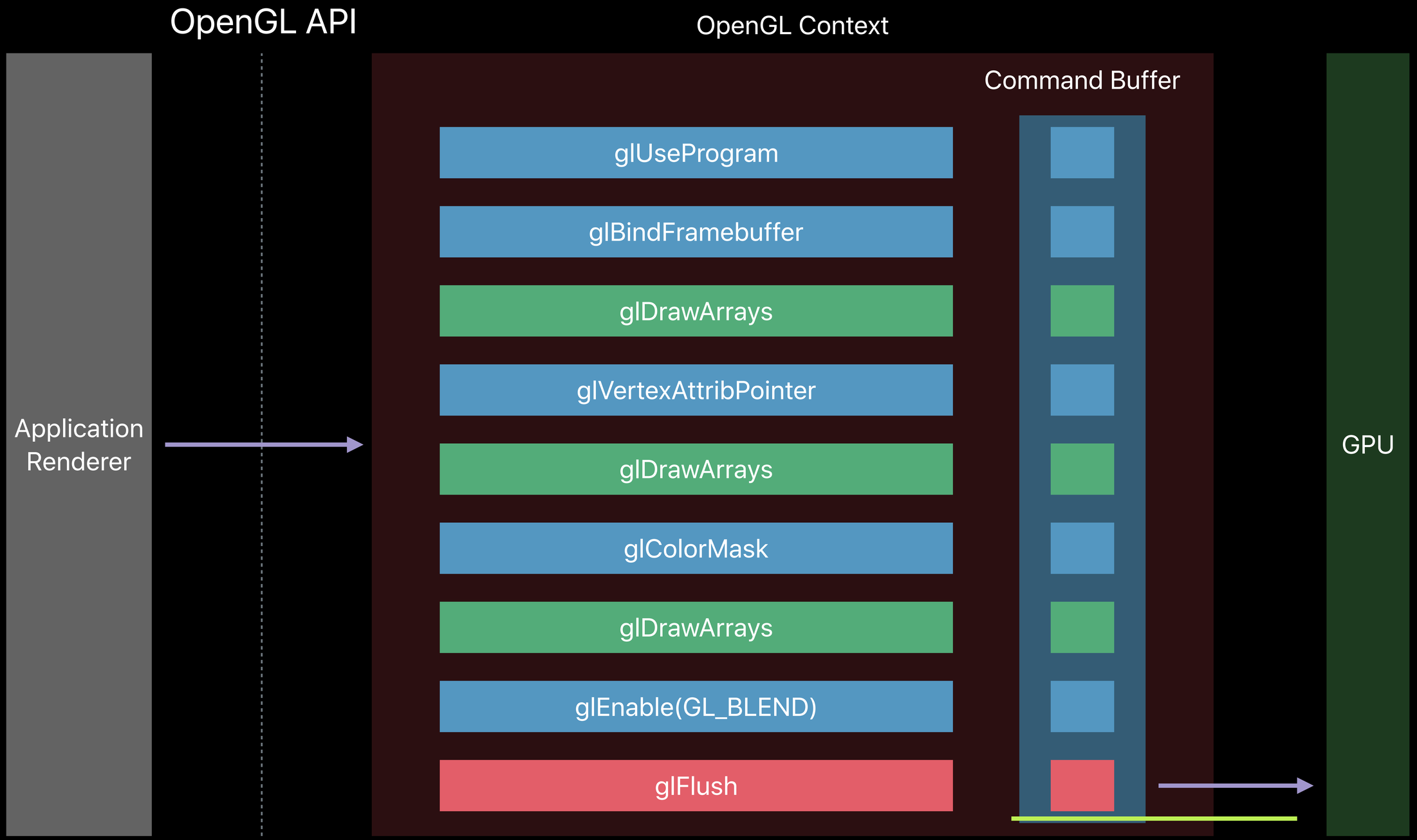

OpenGL API
OpenGL Context
Application
Renderer
GPU

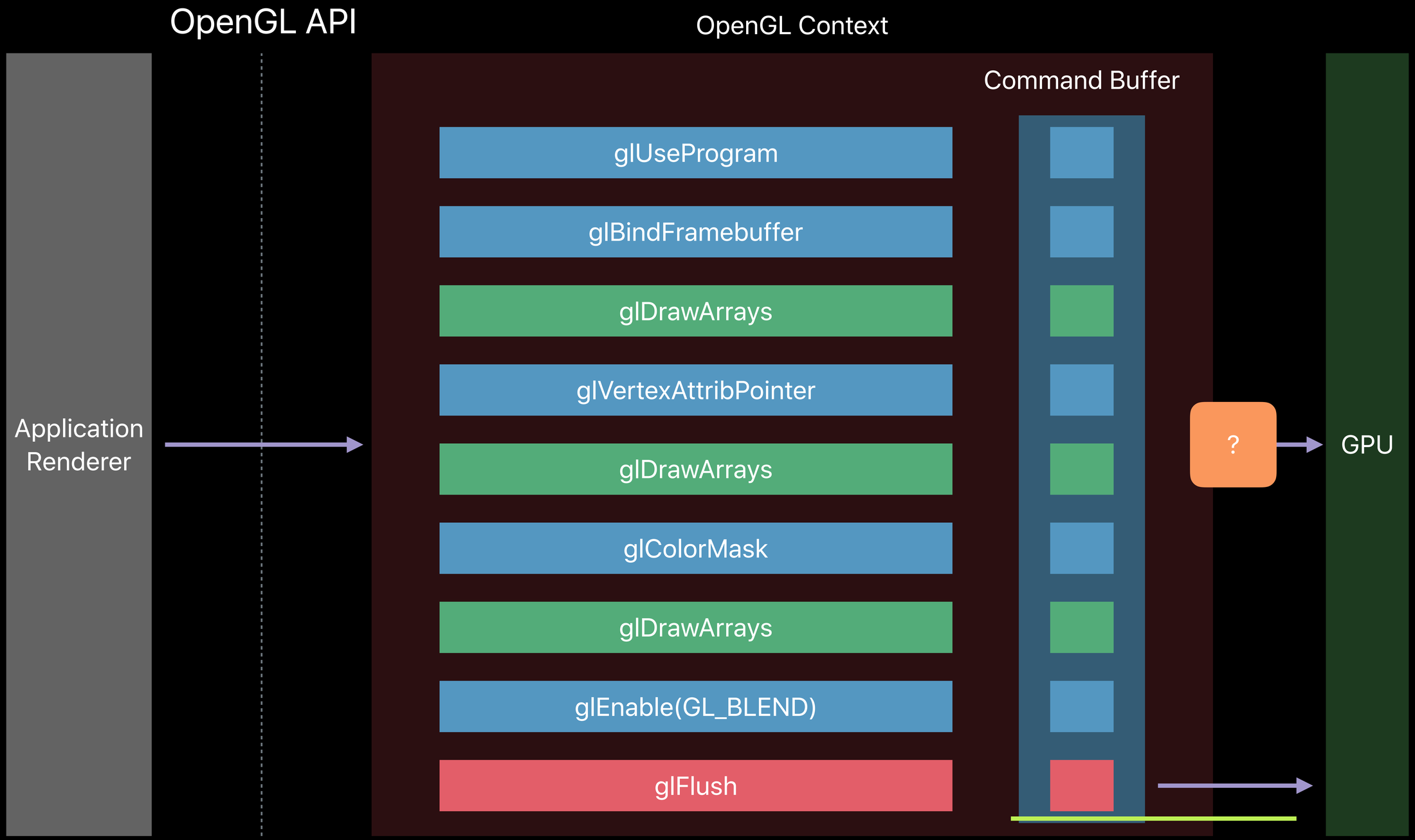

OpenGL API
OpenGL Context
Application
Renderer
?
GPU

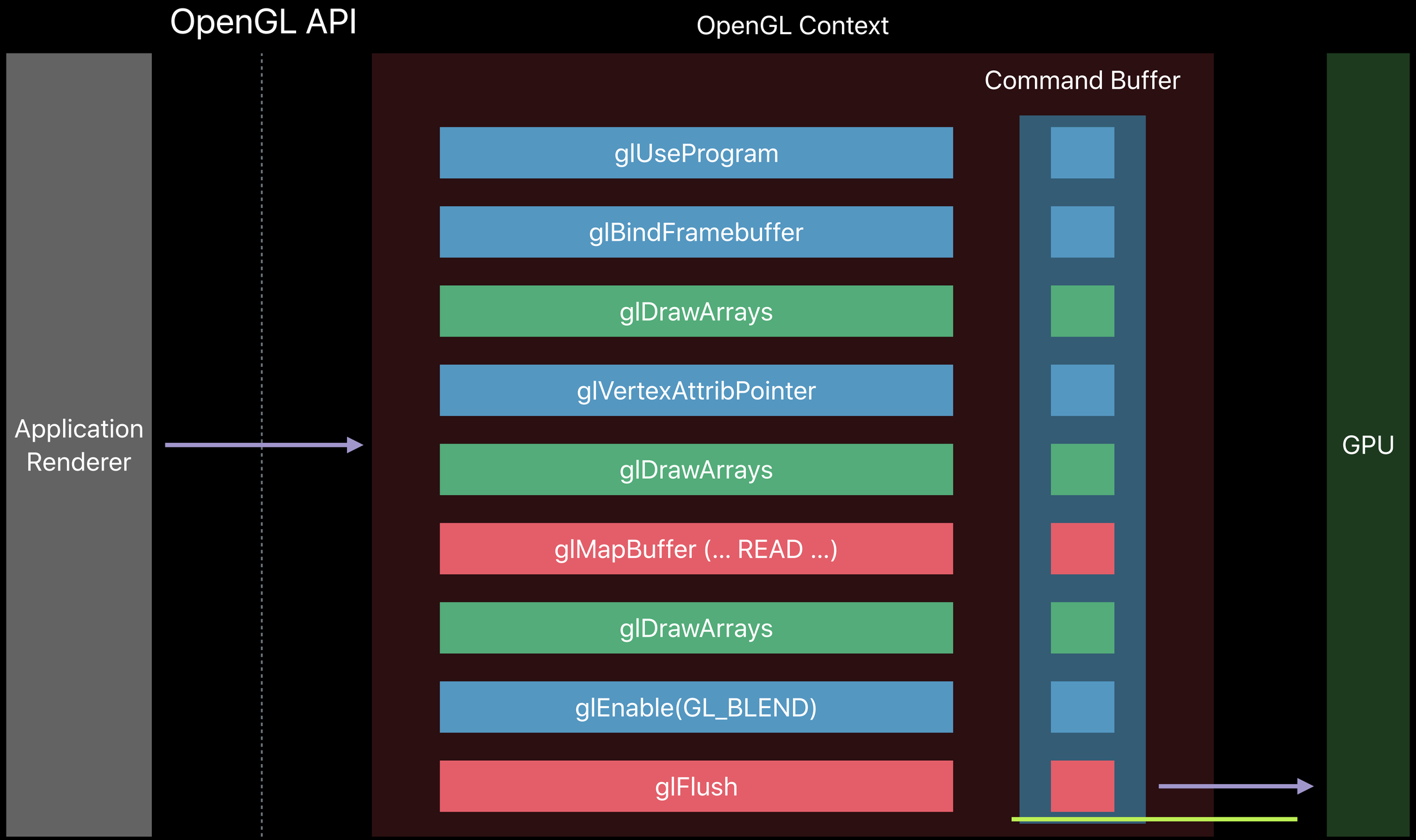

OpenGL API
OpenGL Context
Command Buffer
Application
Renderer
glUseProgram
glBindFramebuffer
glDrawArrays
glVertexAttribPointer
glDrawArrays
glColorMask
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
GPU

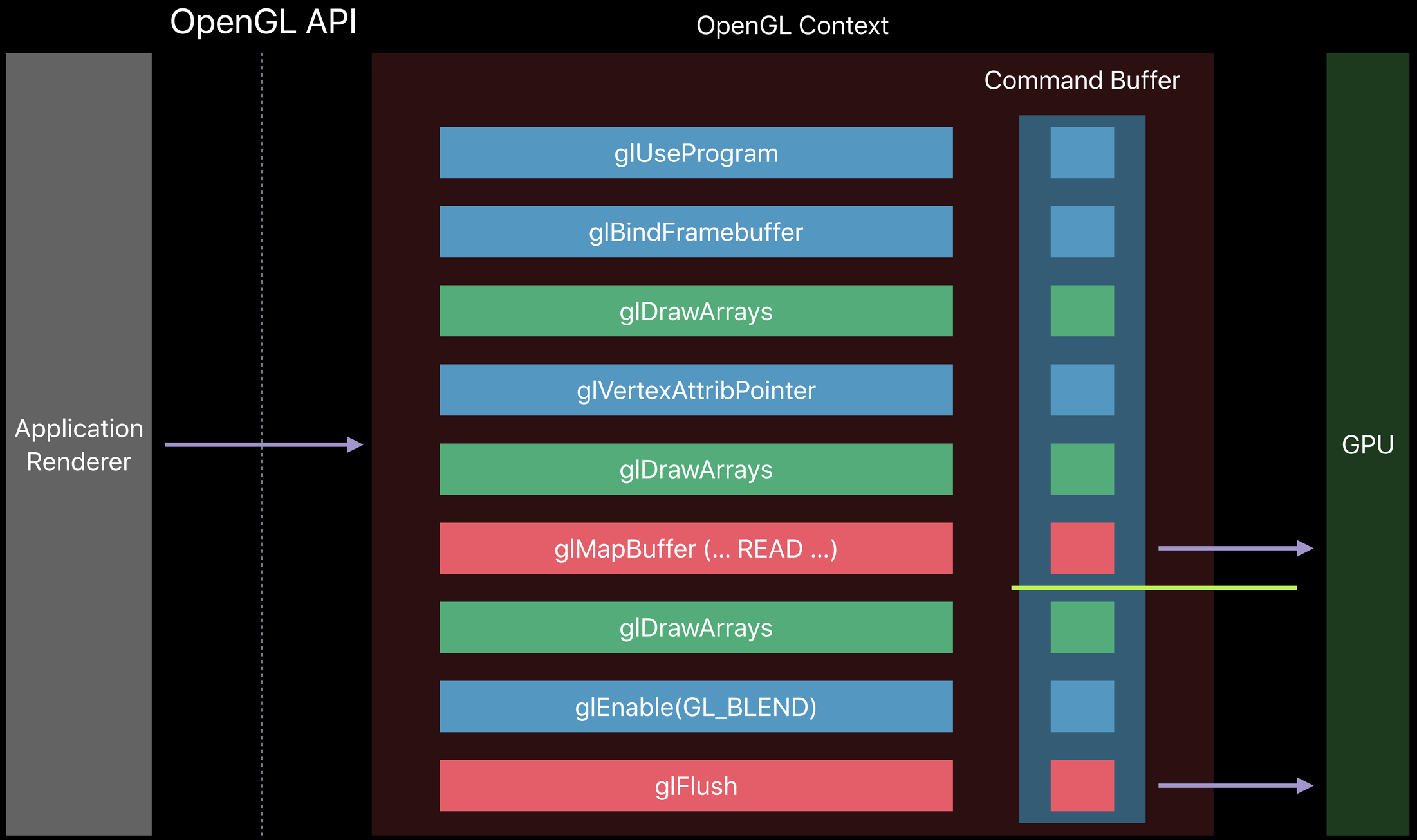

OpenGL API
OpenGL Context
Command Buffer
Application
Renderer
glUseProgram
glBindFramebuffer
glDrawArrays
glVertexAttribPointer
glDrawArrays
glColorMask
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
GPU

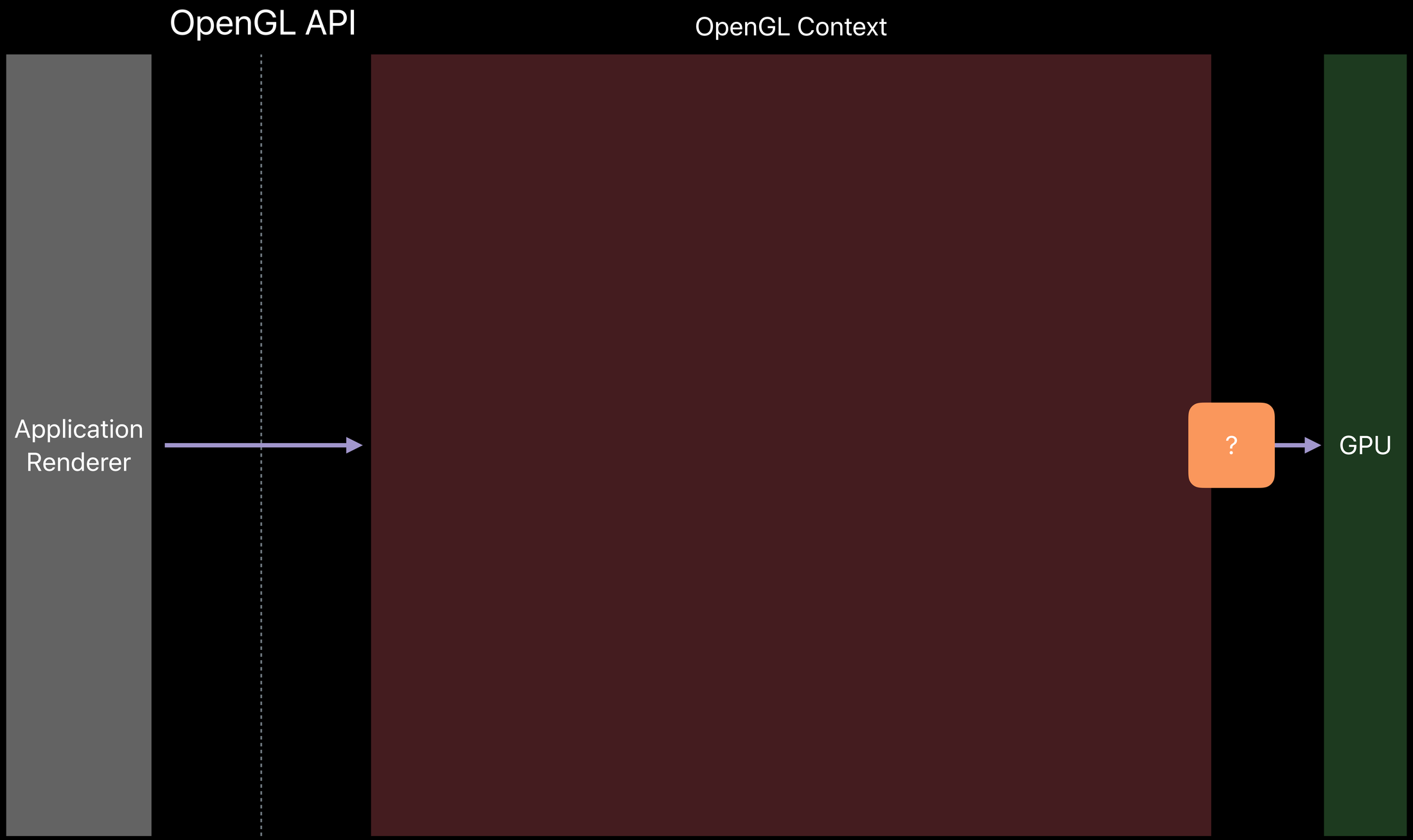

OpenGL API
OpenGL Context
Command Buffer
Application
Renderer
glUseProgram
glBindFramebuffer
glDrawArrays
glVertexAttribPointer
glDrawArrays
glColorMask
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
GPU


OpenGL API
OpenGL Context
Command Buffer
Application
Renderer
glUseProgram
glBindFramebuffer
glDrawArrays
glVertexAttribPointer
glDrawArrays
glColorMask
glDrawArrays
glEnable(GL_BLEND)
glFlush
GPU


OpenGL API
OpenGL Context
Command Buffer
Application
Renderer
glUseProgram
glBindFramebuffer
glDrawArrays
glVertexAttribPointer
glDrawArrays
glColorMask
glDrawArrays
glEnable(GL_BLEND)
glFlush
?
GPU


OpenGL API
OpenGL Context
Command Buffer
Application
Renderer
glUseProgram
glBindFramebuffer
glDrawArrays
glVertexAttribPointer
glDrawArrays
glMapBuffer (... READ ...)
glDrawArrays
glEnable(GL_BLEND)
glFlush
GPU

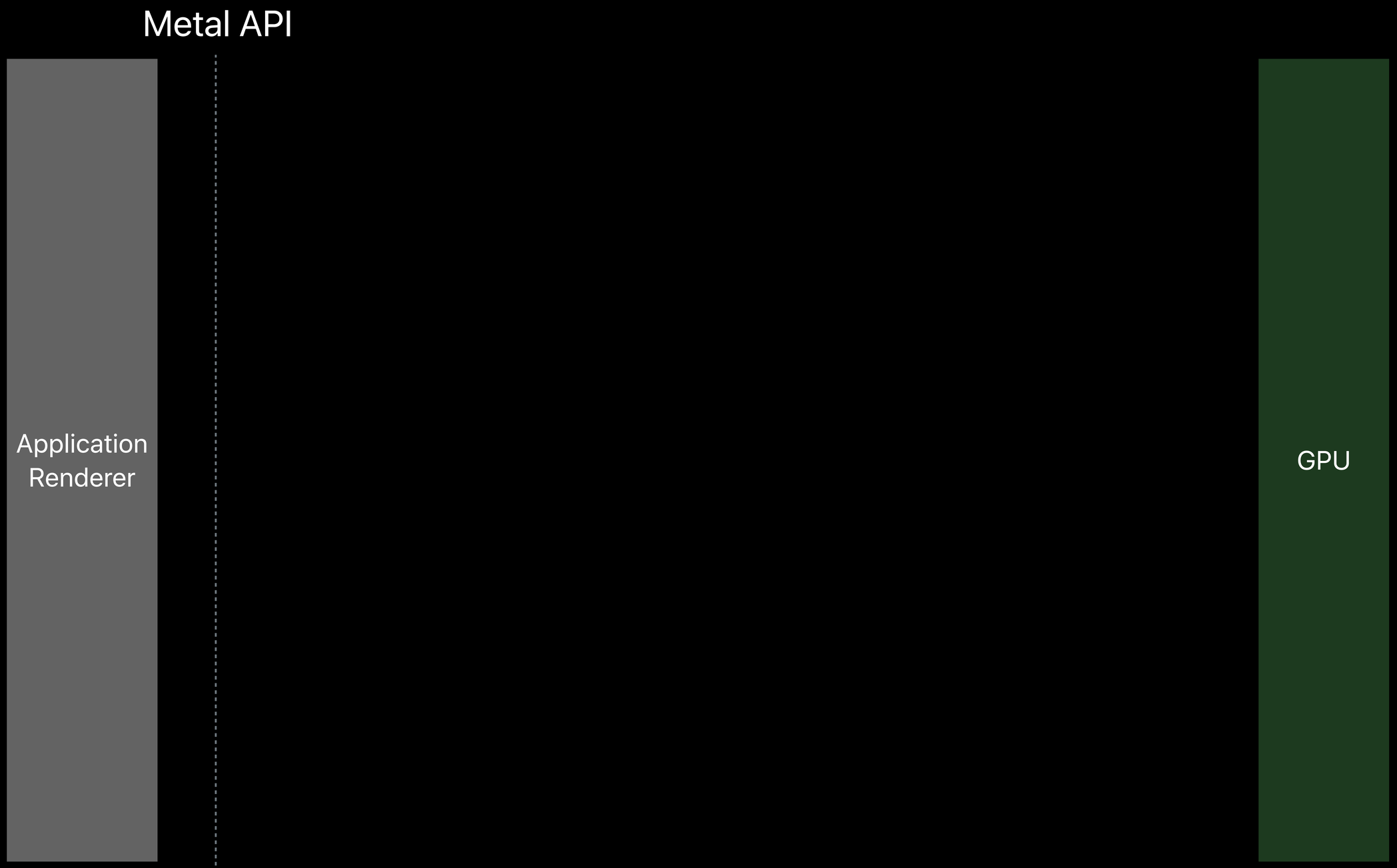

OpenGL API
OpenGL Context
Command Buffer
Application
Renderer
glUseProgram
glBindFramebuffer
glDrawArrays
glVertexAttribPointer
glDrawArrays
glMapBuffer (... READ ...)
glDrawArrays
glEnable(GL_BLEND)
glFlush
GPU


OpenGL API
OpenGL Context
Application
Renderer
?
GPU

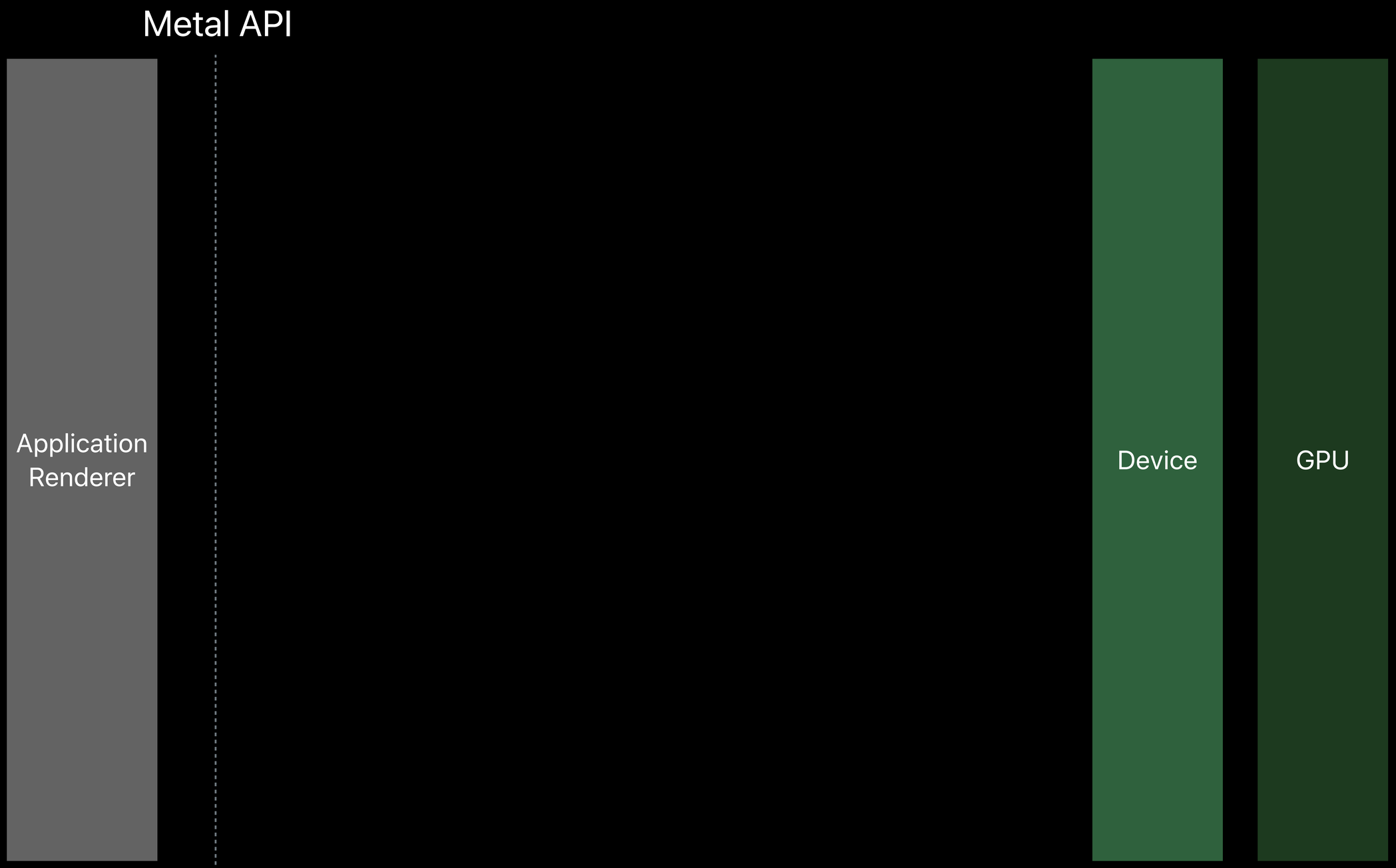
Submitting Work to the GPU

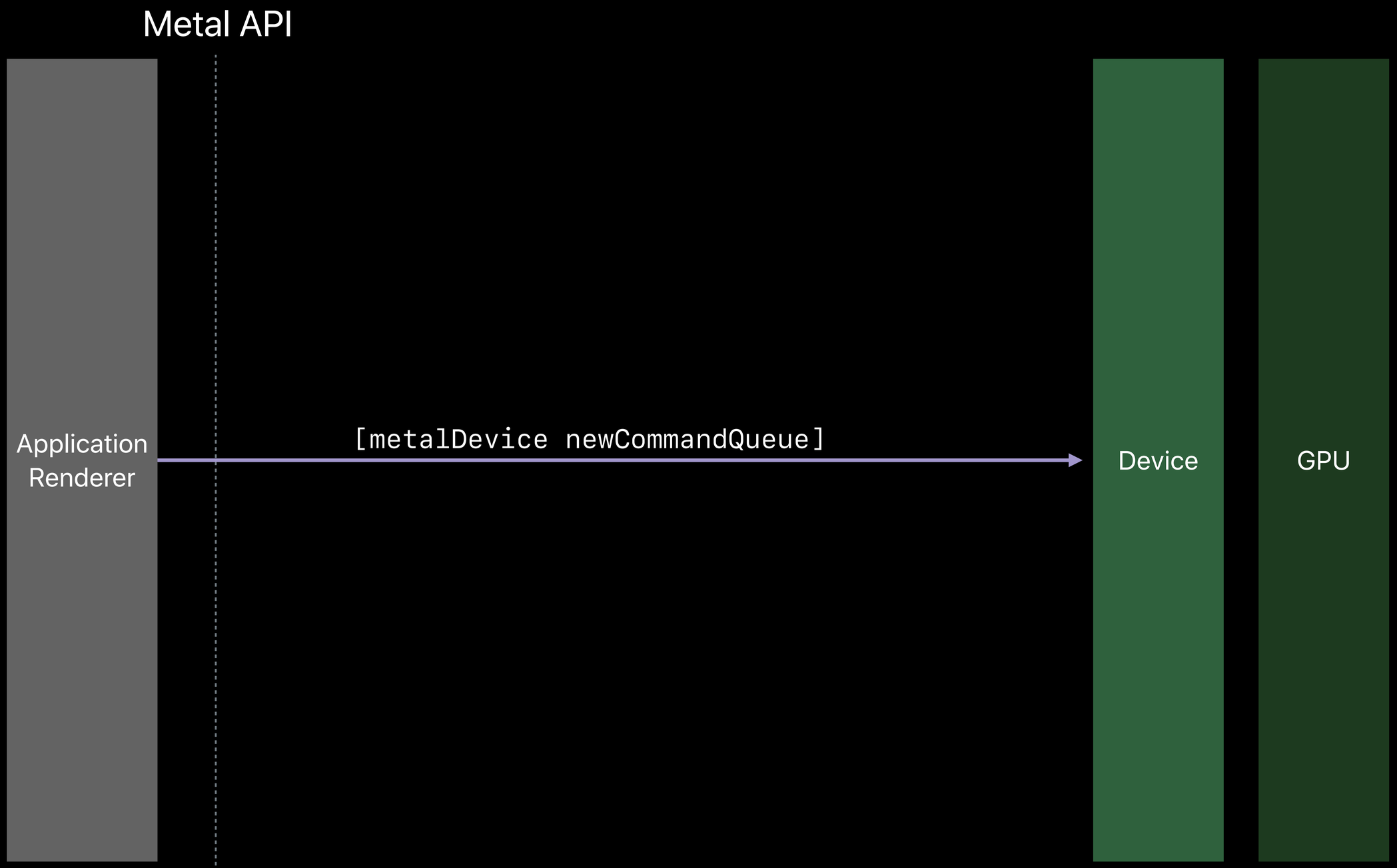
Metal — Explicit API

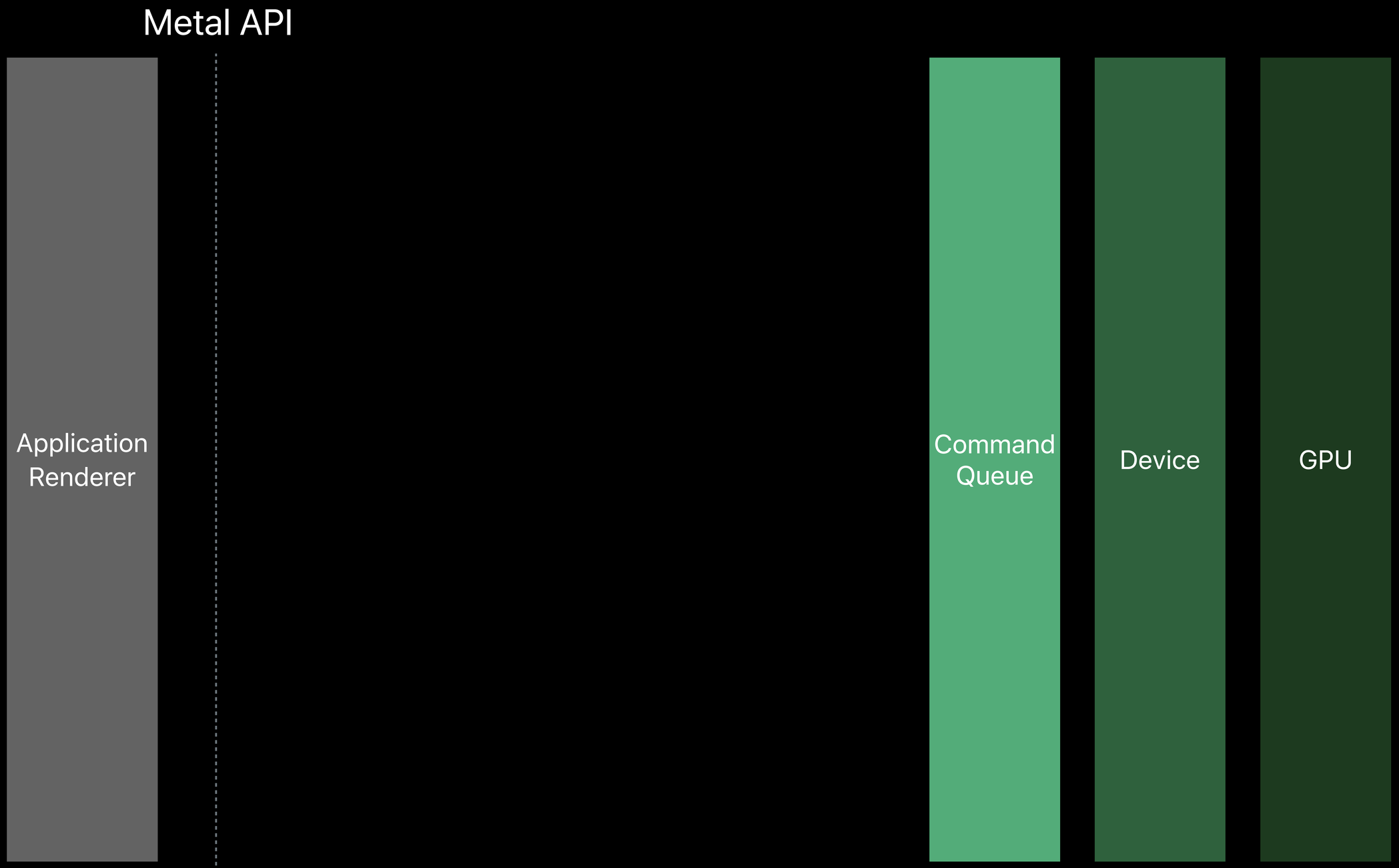
- Direct control over command submission
- Breaks “OpenGL context” concept into smaller objects

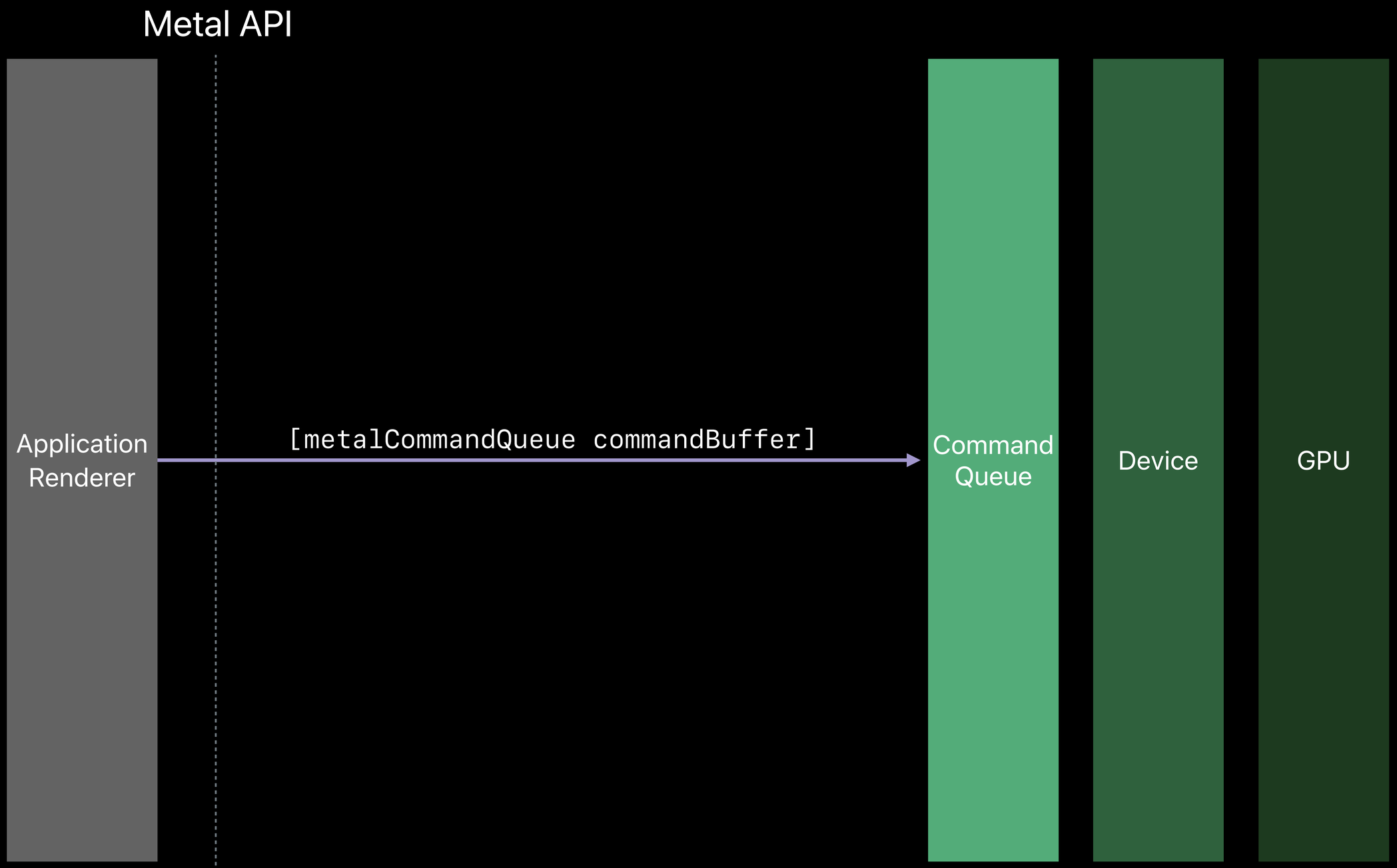

Metal API
Application
Renderer
GPU

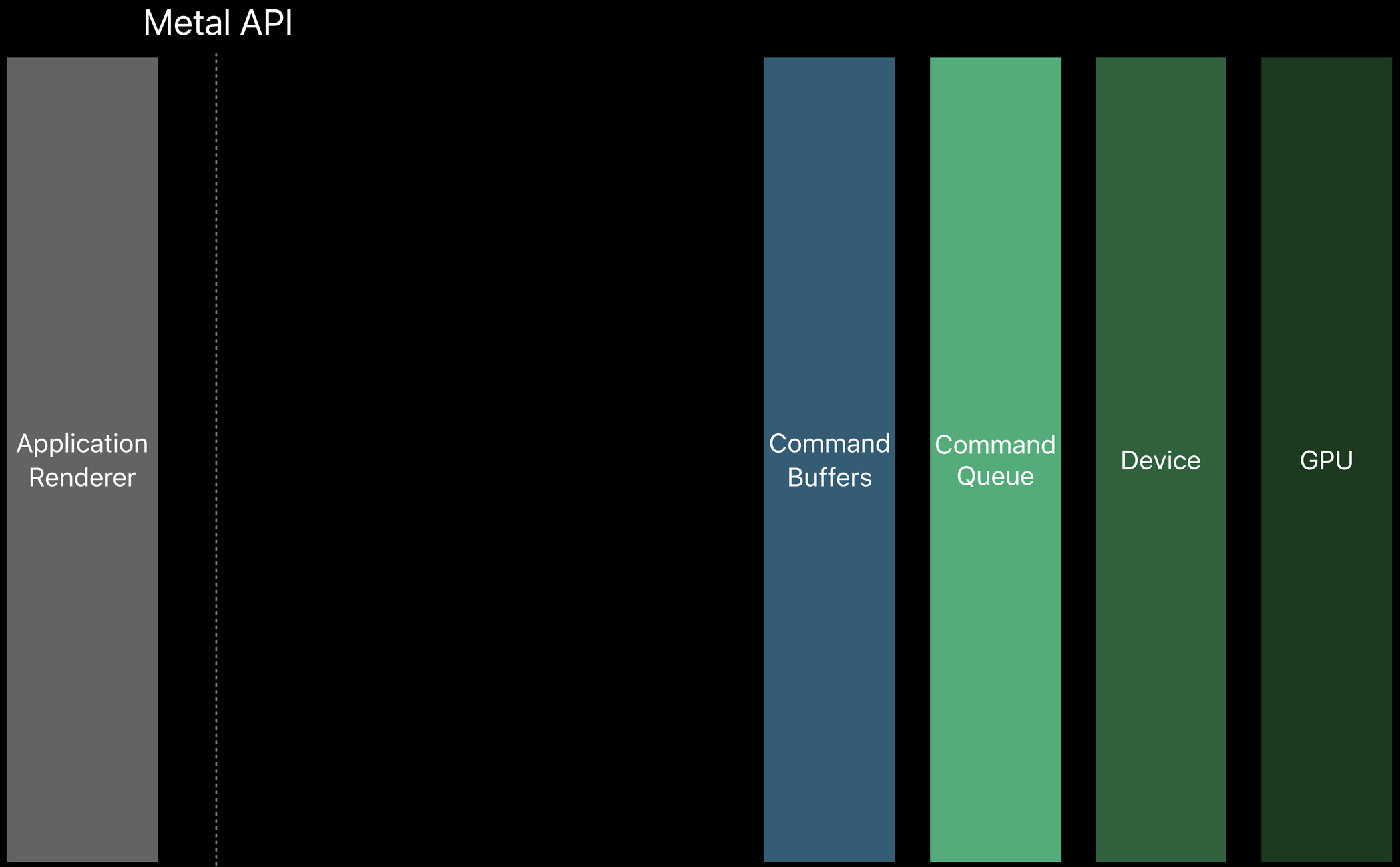

Metal API
Application
Renderer
MTLCreateSystemDefaultDevice()
Device
GPU

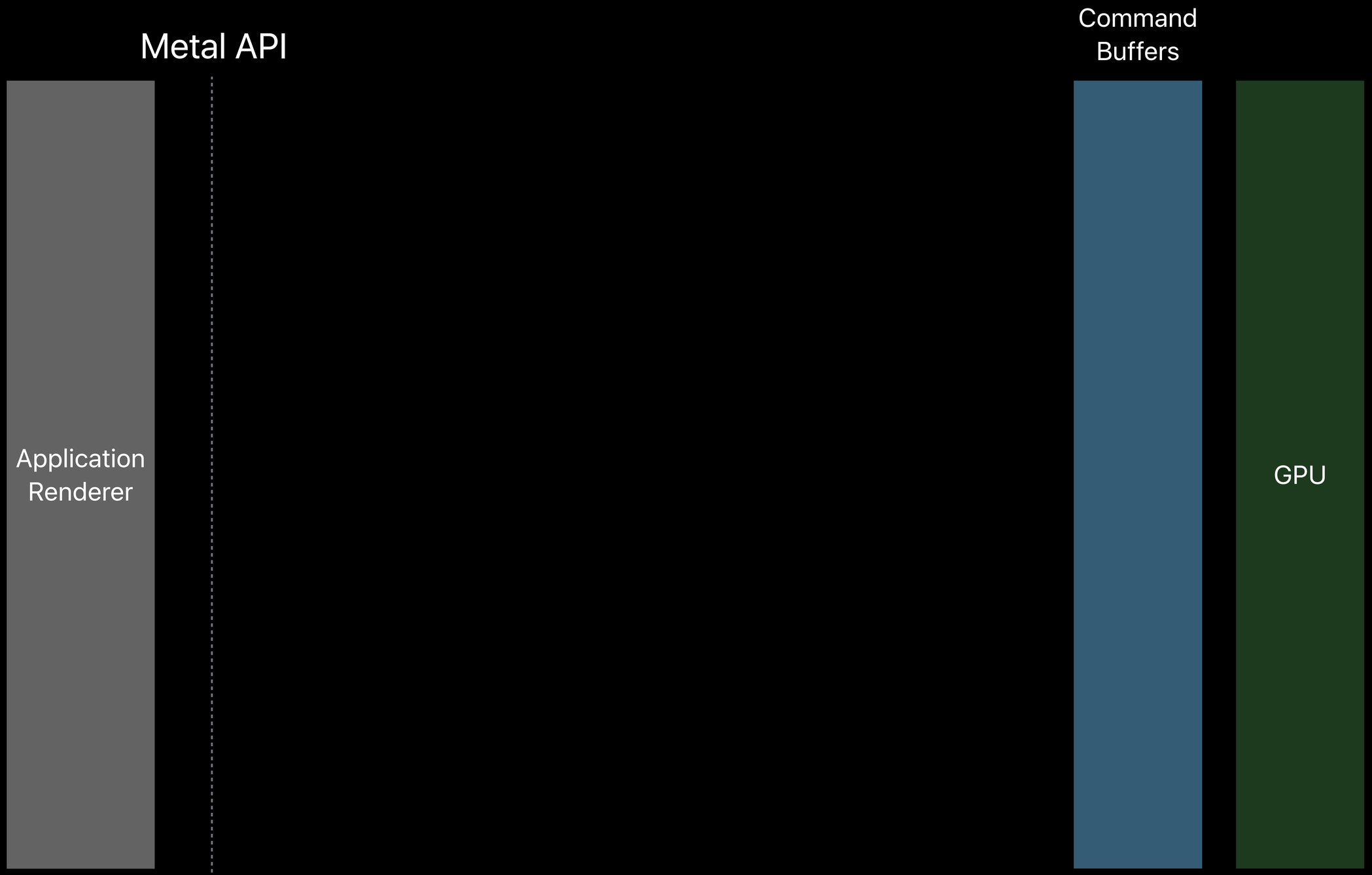

Metal API
Application
Renderer
Device
GPU

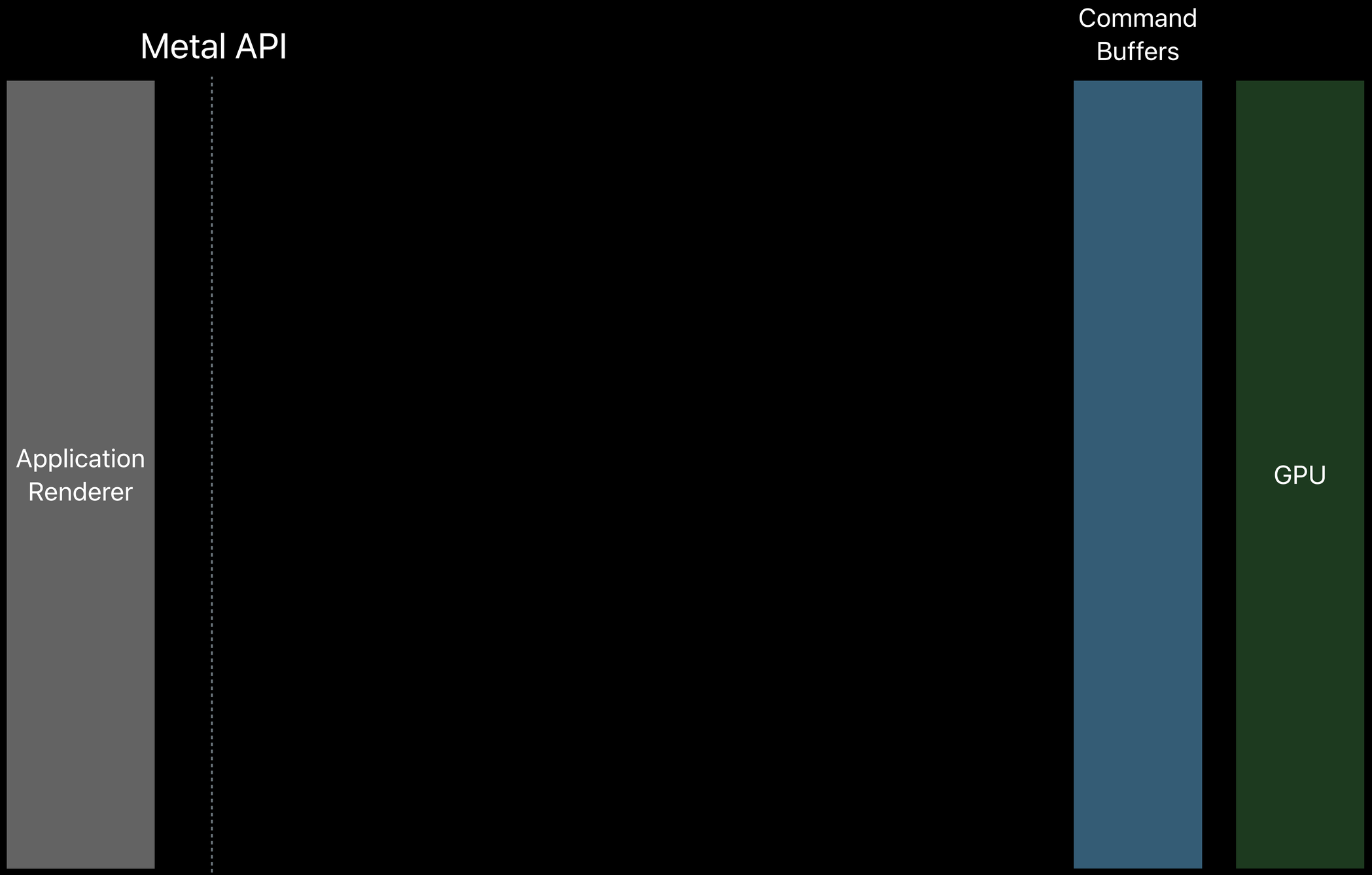

Metal API
Application
Renderer
[metalDevice newCommandQueue]
Device
GPU

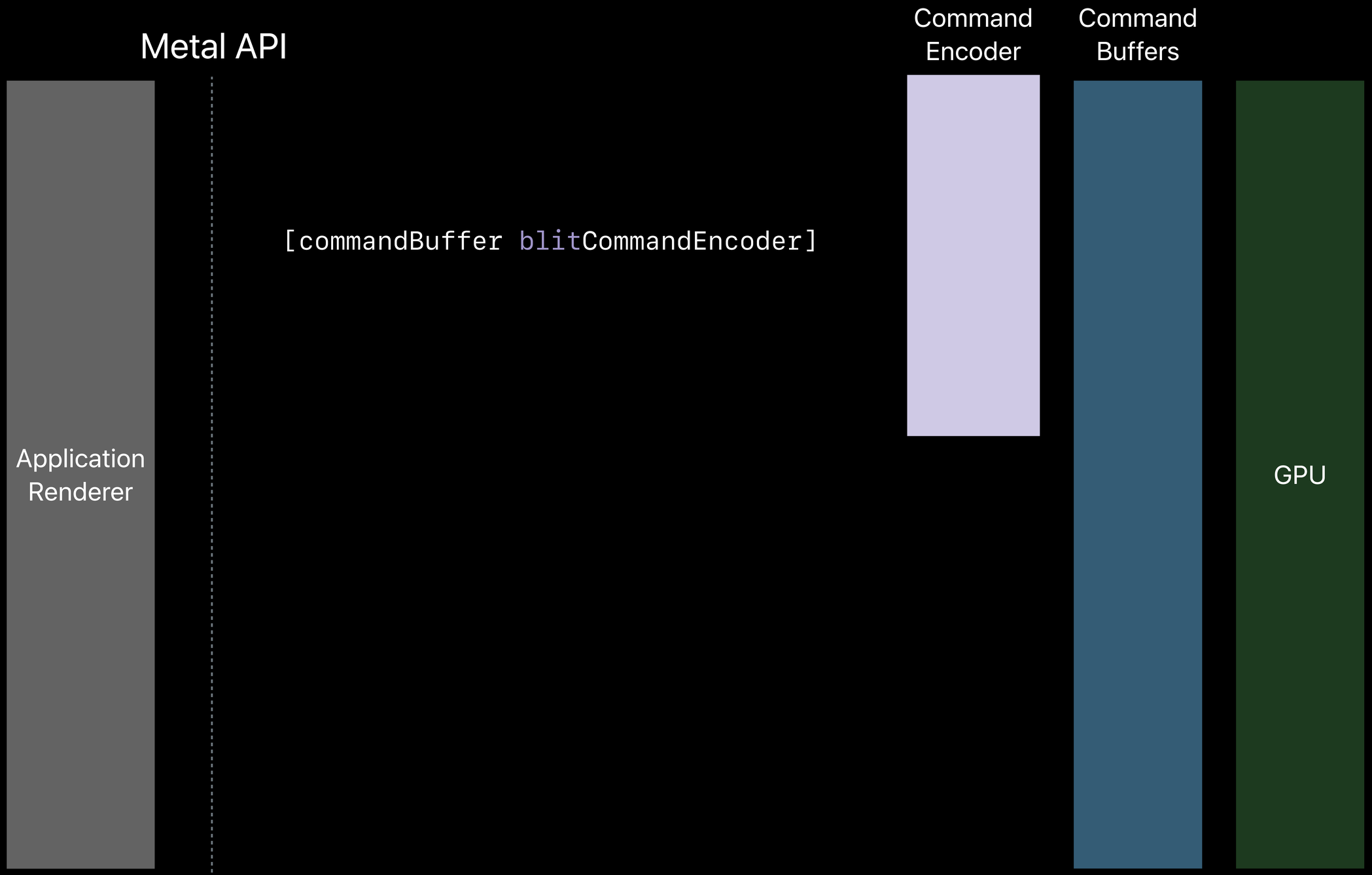

Metal API
Application
Renderer
Command
Queue
Device
GPU

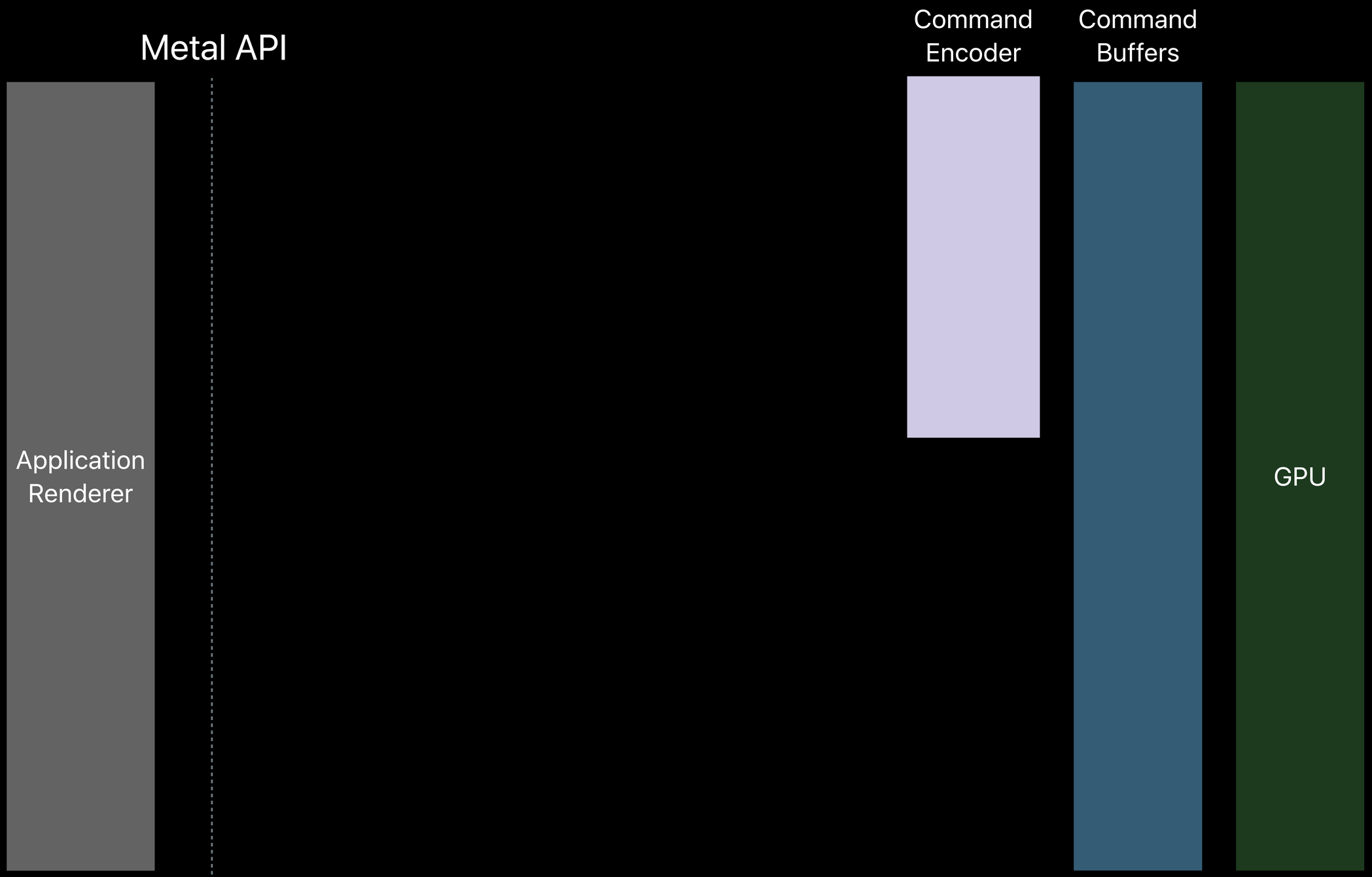

Metal API
Application
Renderer
[metalCommandQueue commandBuffer]
Command
Queue
Device
GPU

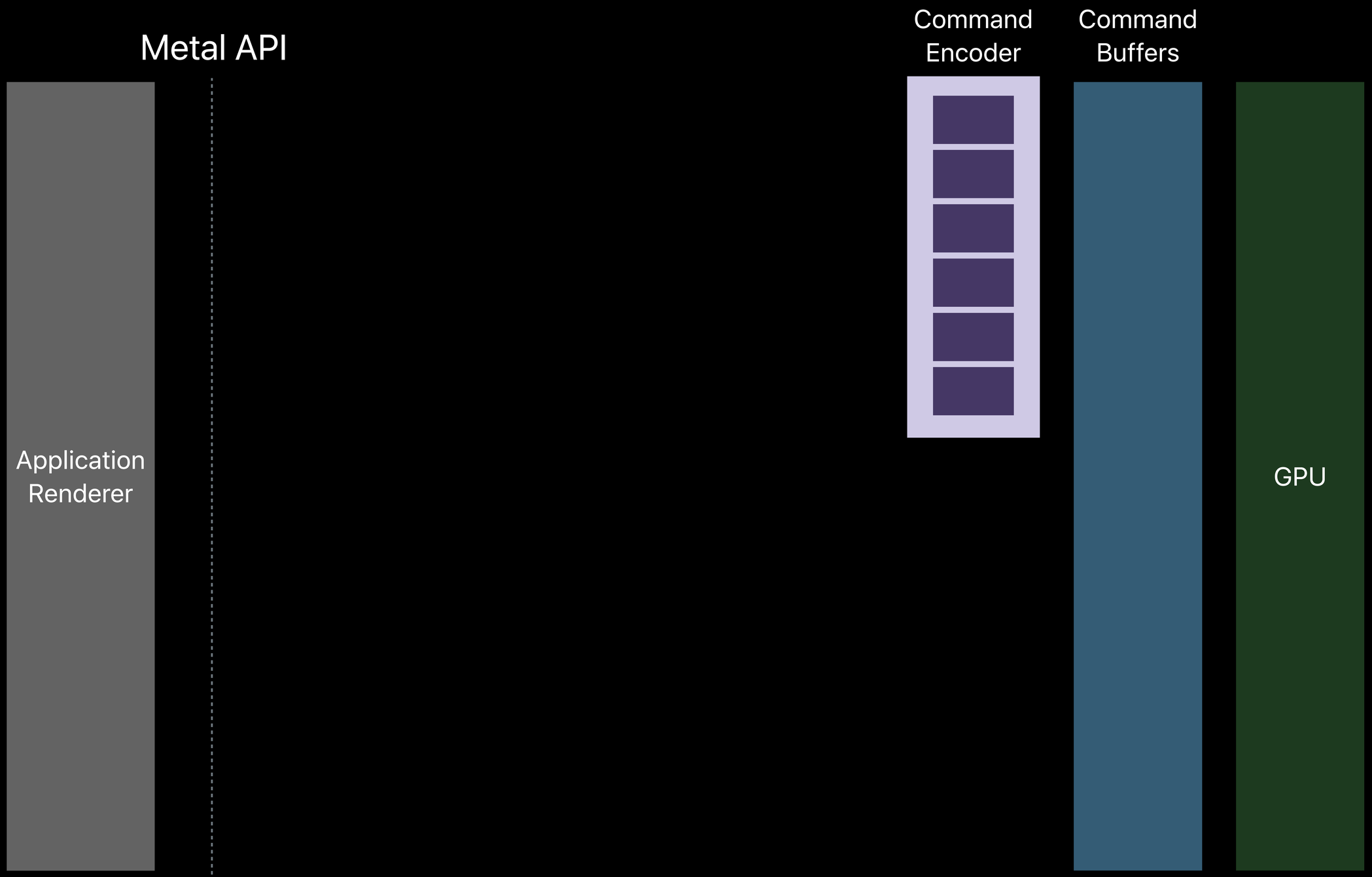

Metal API
Application
Renderer
Command
Buffers
Command
Queue
Device
GPU

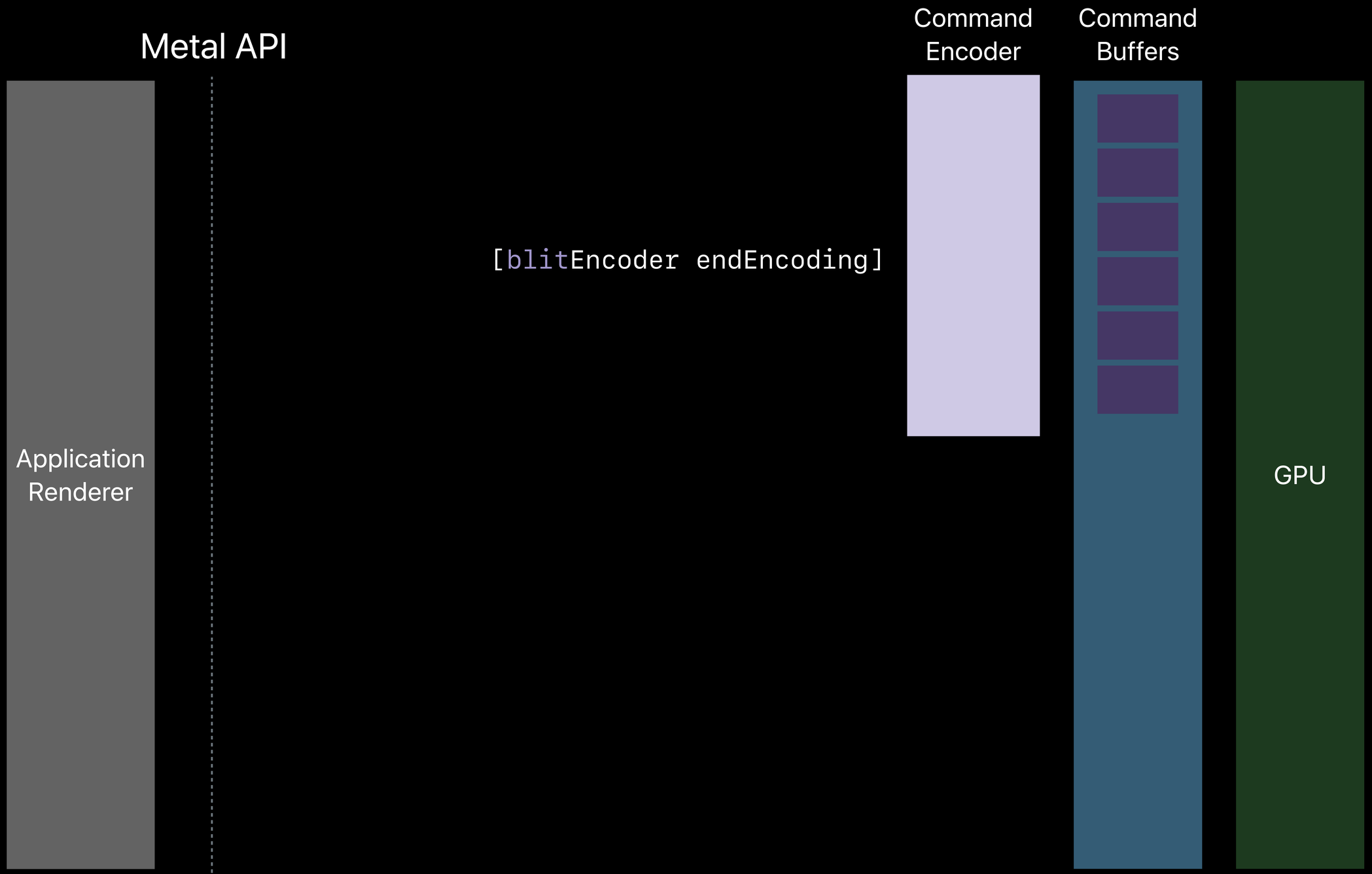

Metal API
Command Buffers
Application Renderer
GPU

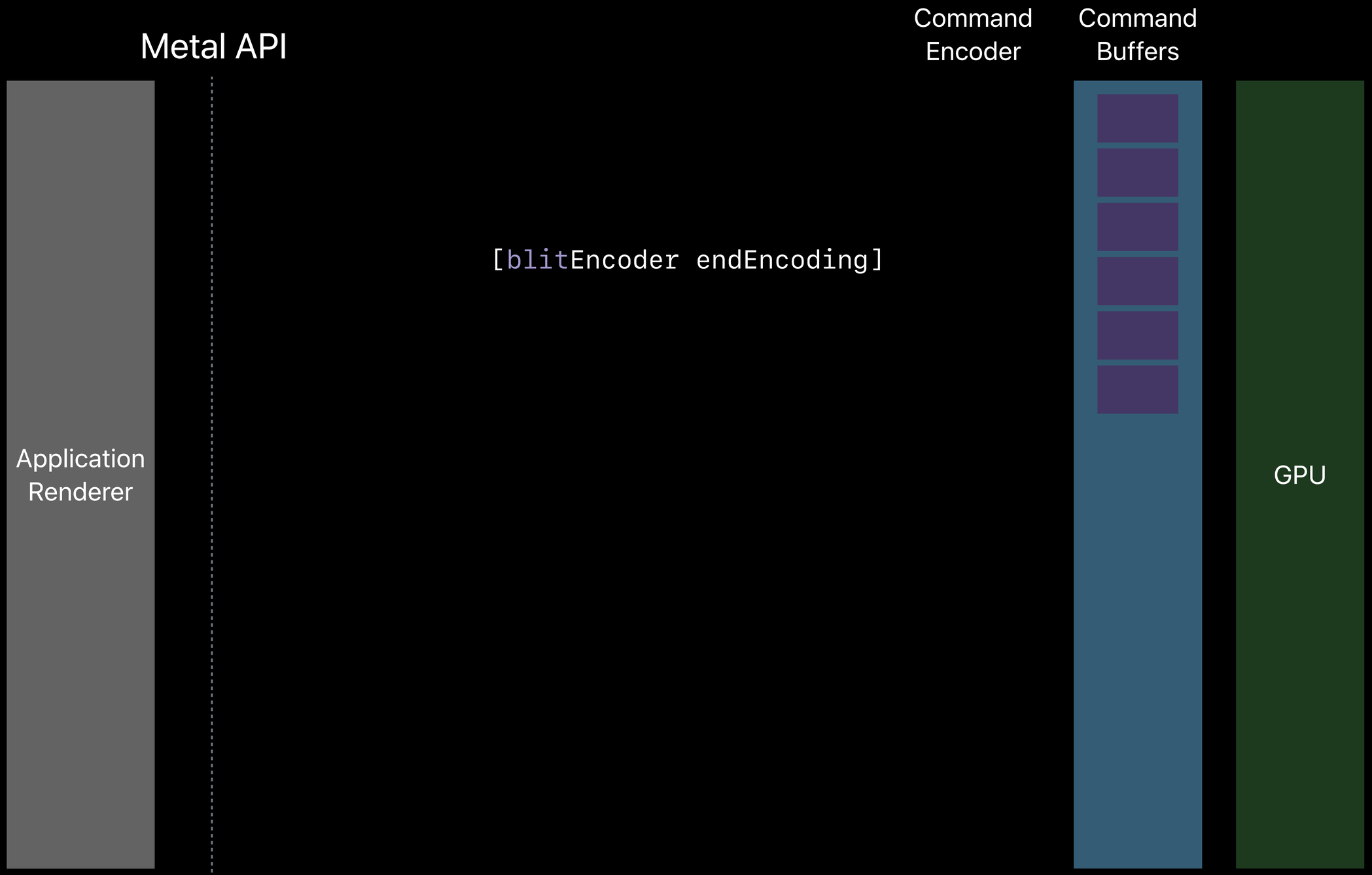

Metal API
Command Buffers
Application Renderer
GPU

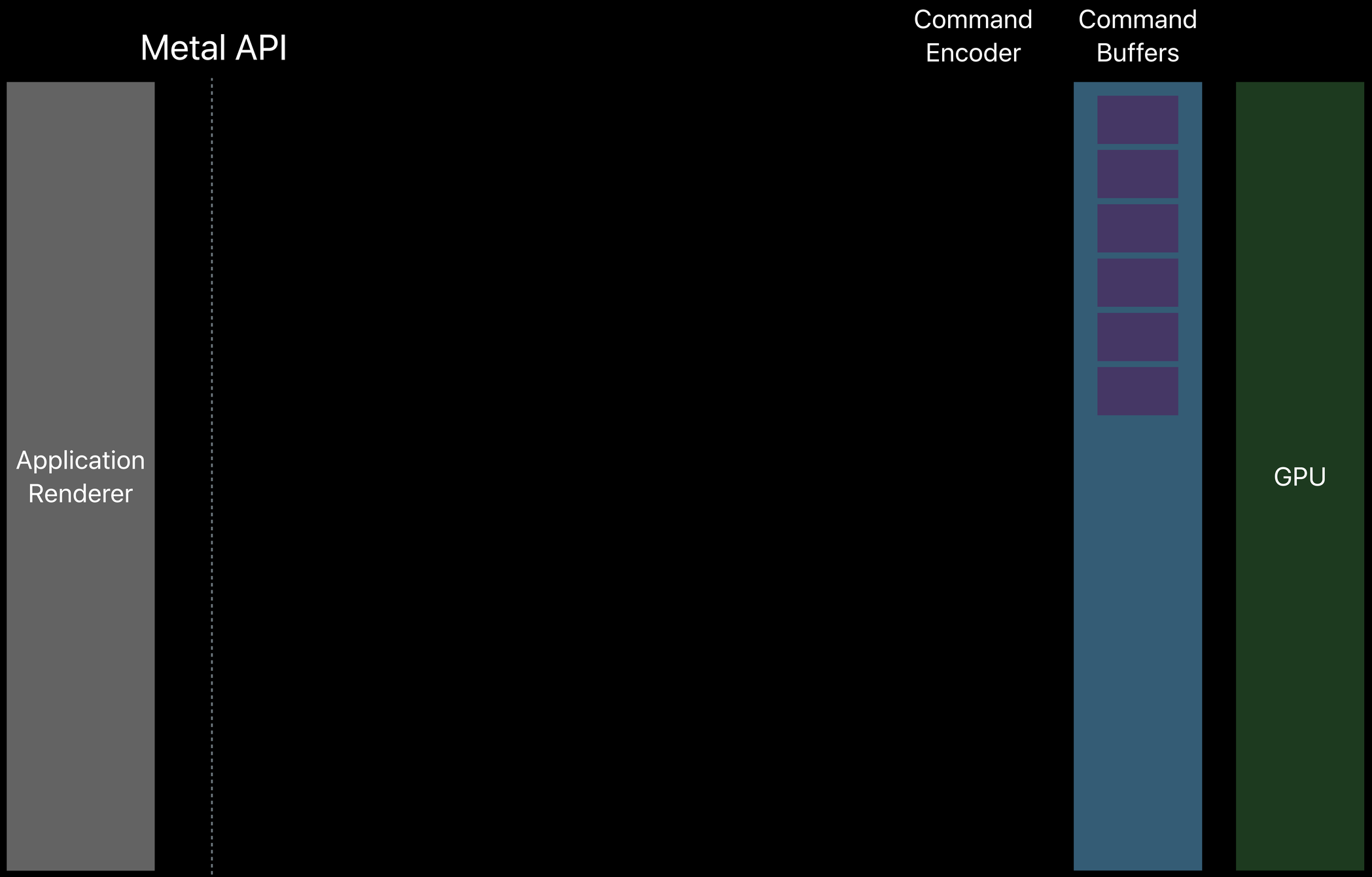

Metal API
Command Encoder
Command Buffers
[commandBuffer blitCommandEncoder]
Application Renderer
GPU

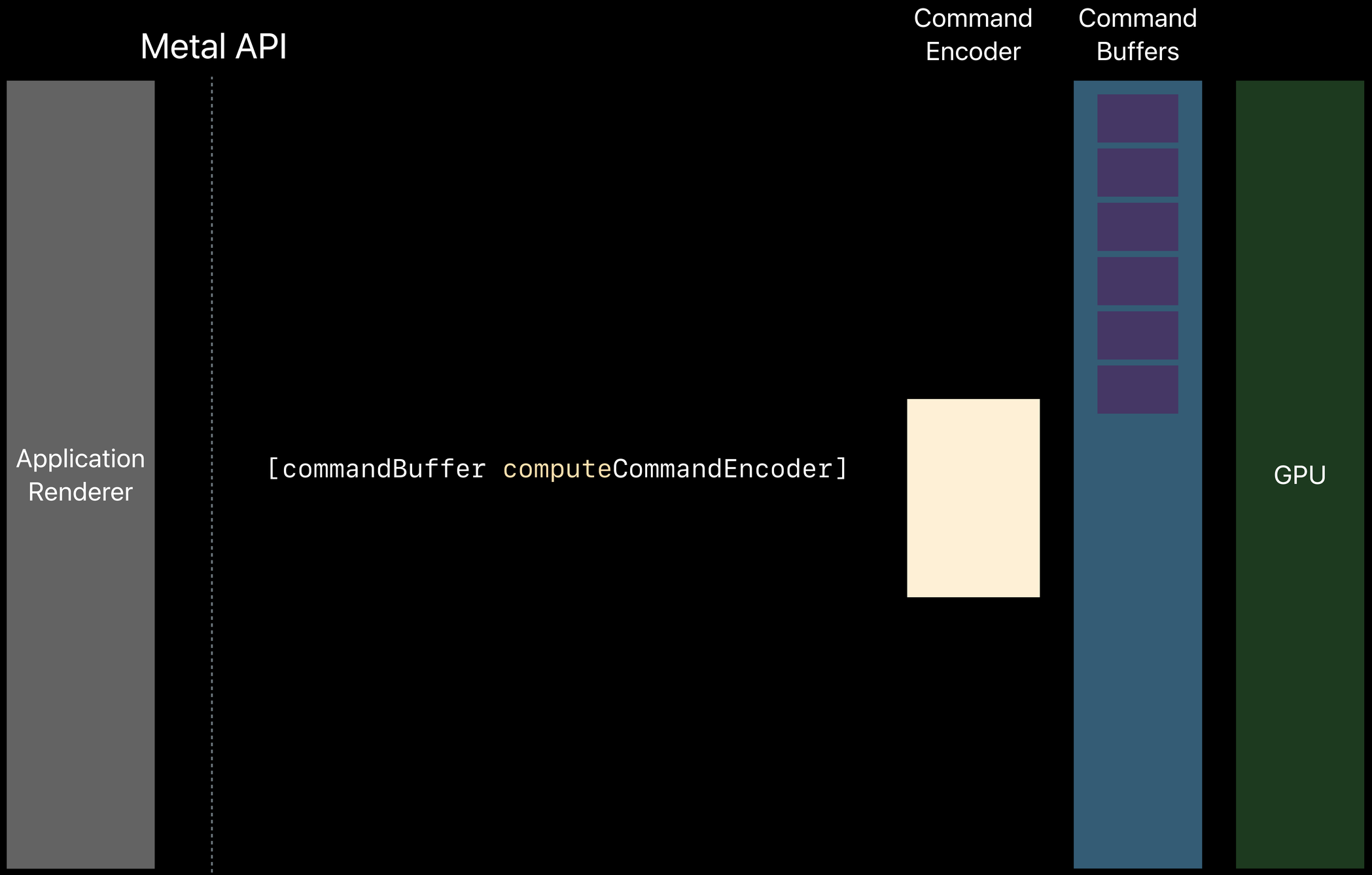

Metal API
Command Encoder
Command Buffers
Application Renderer
GPU

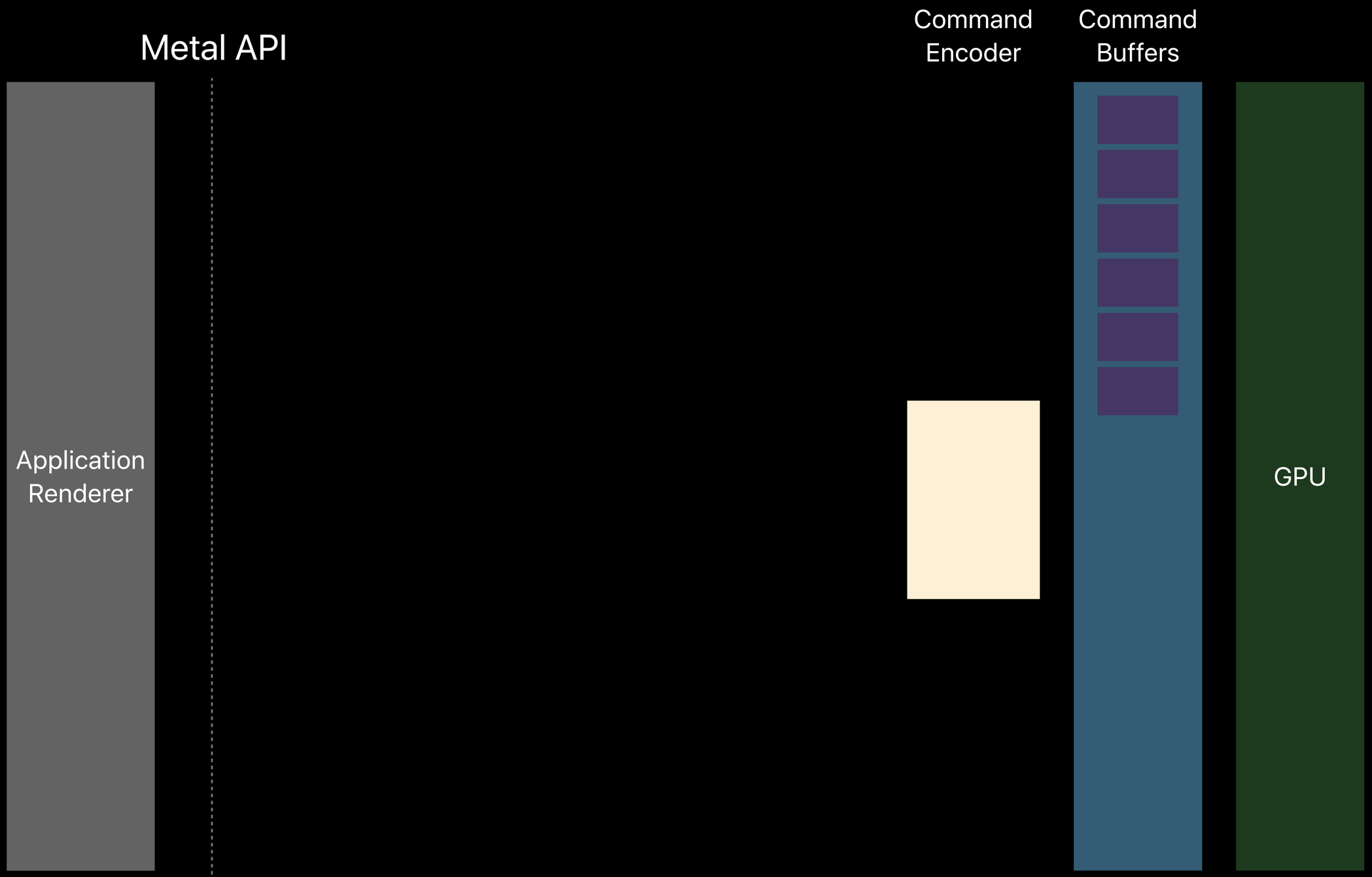

Metal API
Command Encoder
Command Buffers
Application Renderer
GPU

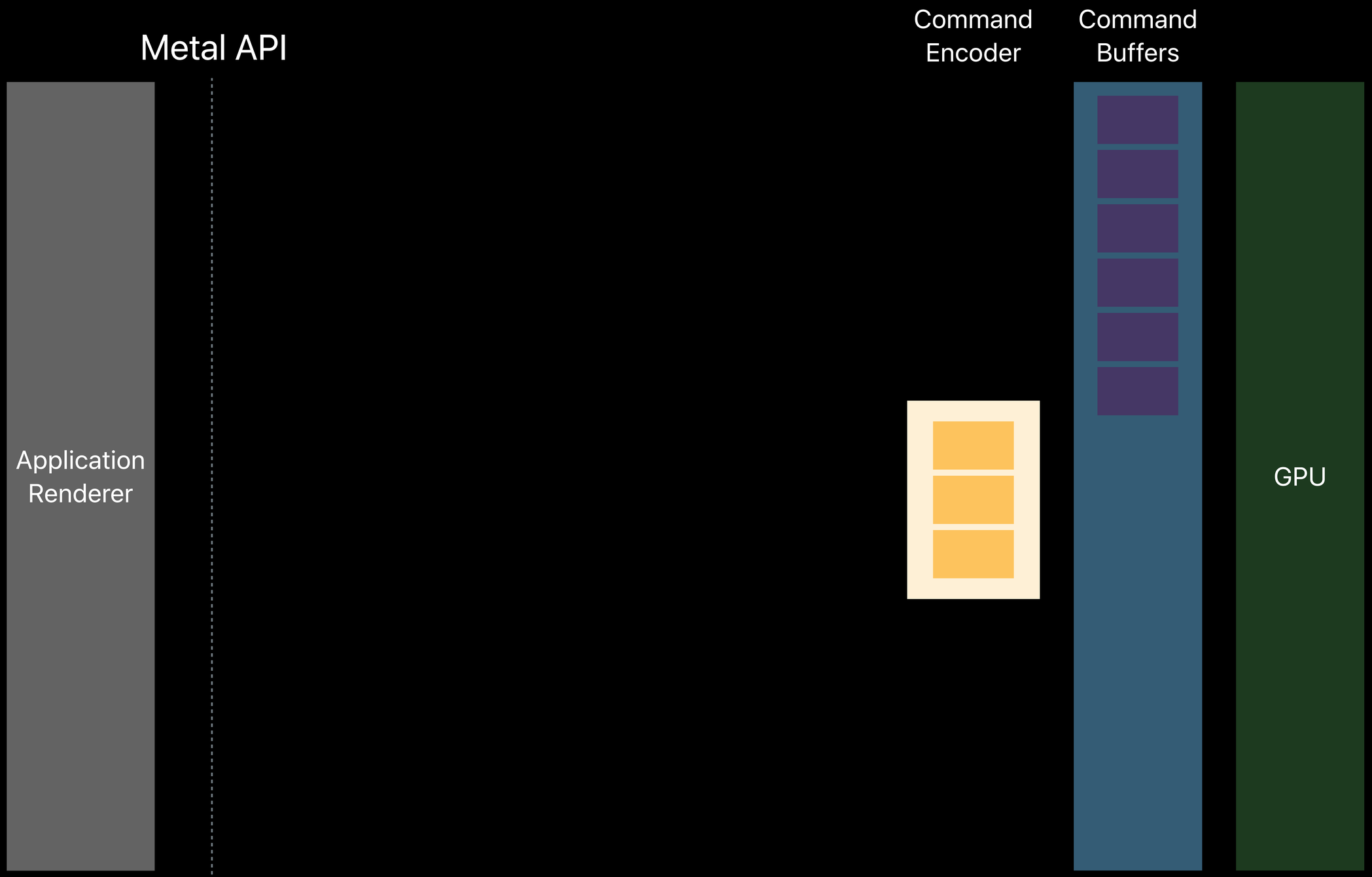

Metal API
Command Encoder
Command Buffers
[blitEncoder endEncoding]
Application Renderer
GPU

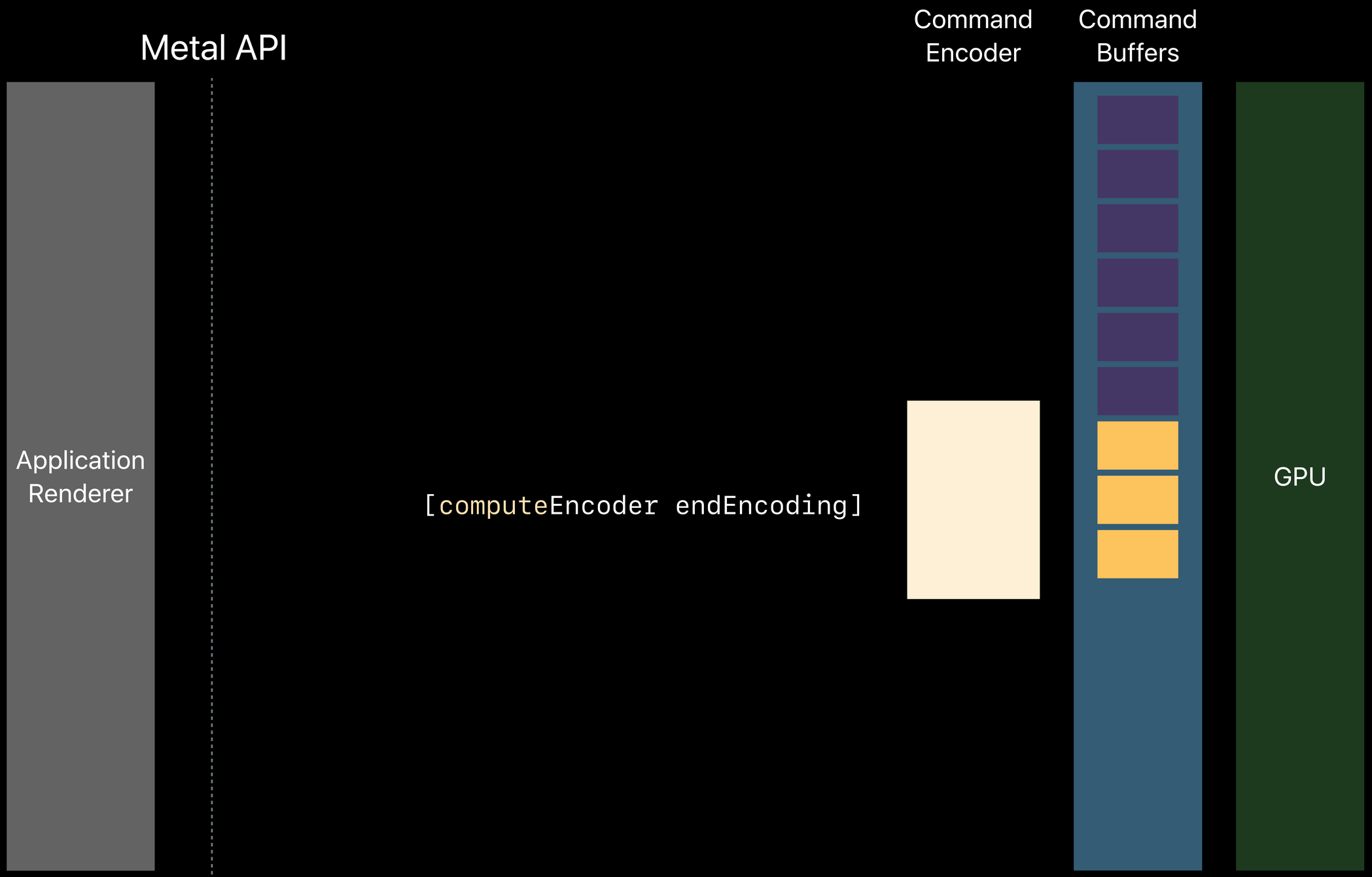

Metal API
Command Encoder
Command Buffers
[blitEncoder endEncoding]
Application Renderer
GPU

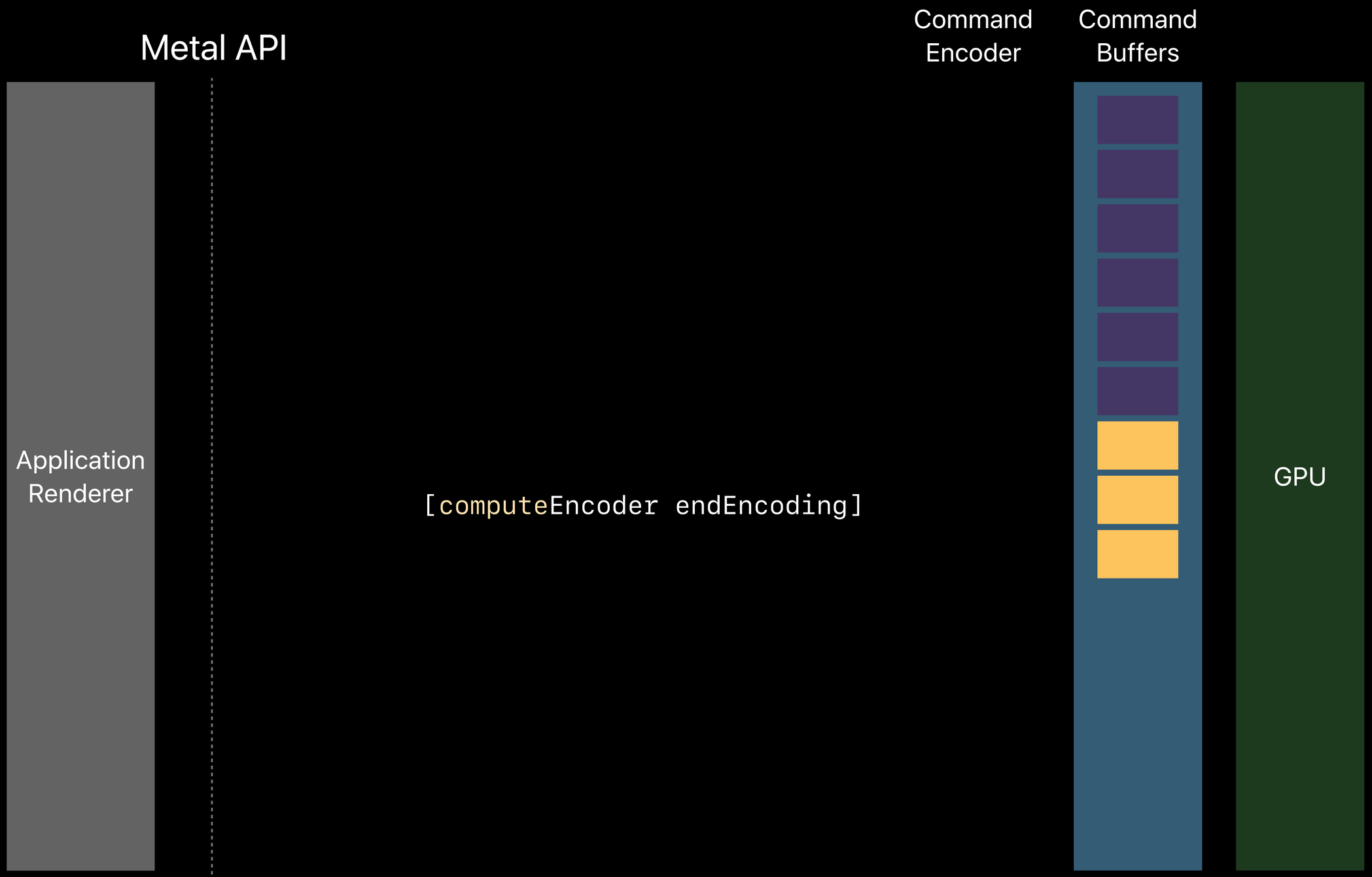

Metal API
Command Encoder
Command Buffers
Application Renderer
GPU

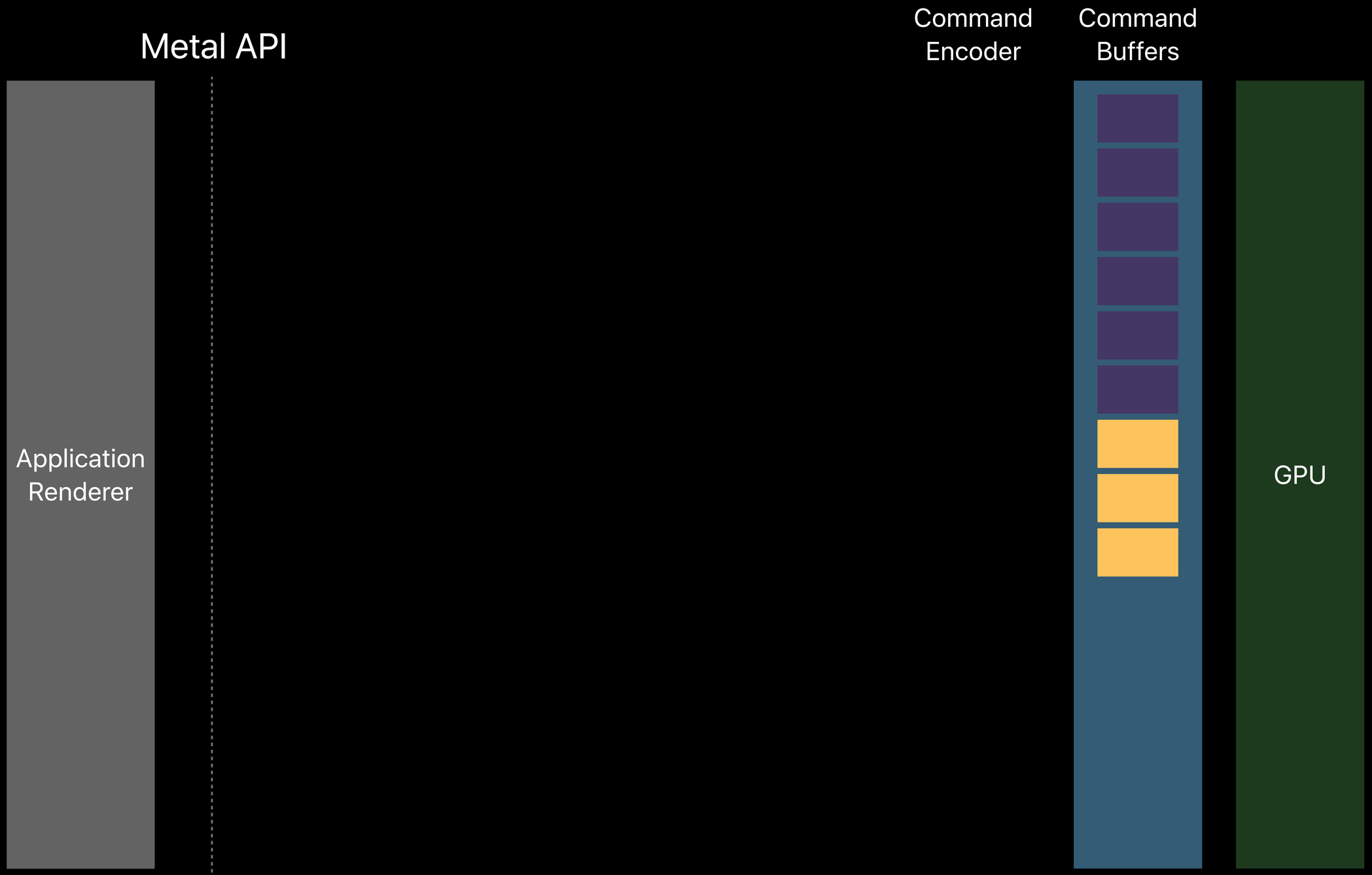

Metal API
Command Encoder
Command Buffers
Application Renderer
[commandBuffer computeCommandEncoder]
GPU

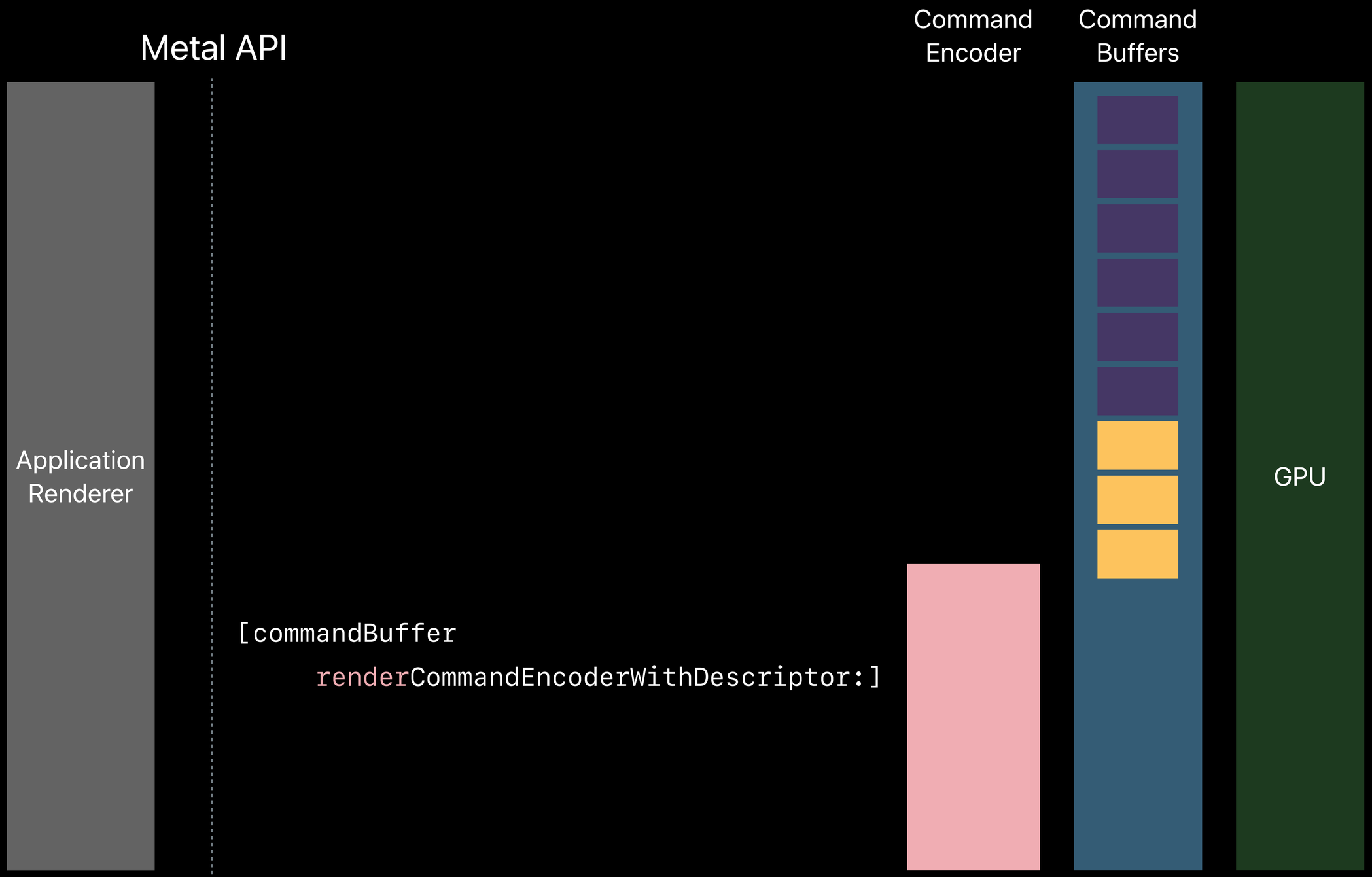

Metal API
Command Encoder
Command Buffers
Application Renderer
GPU

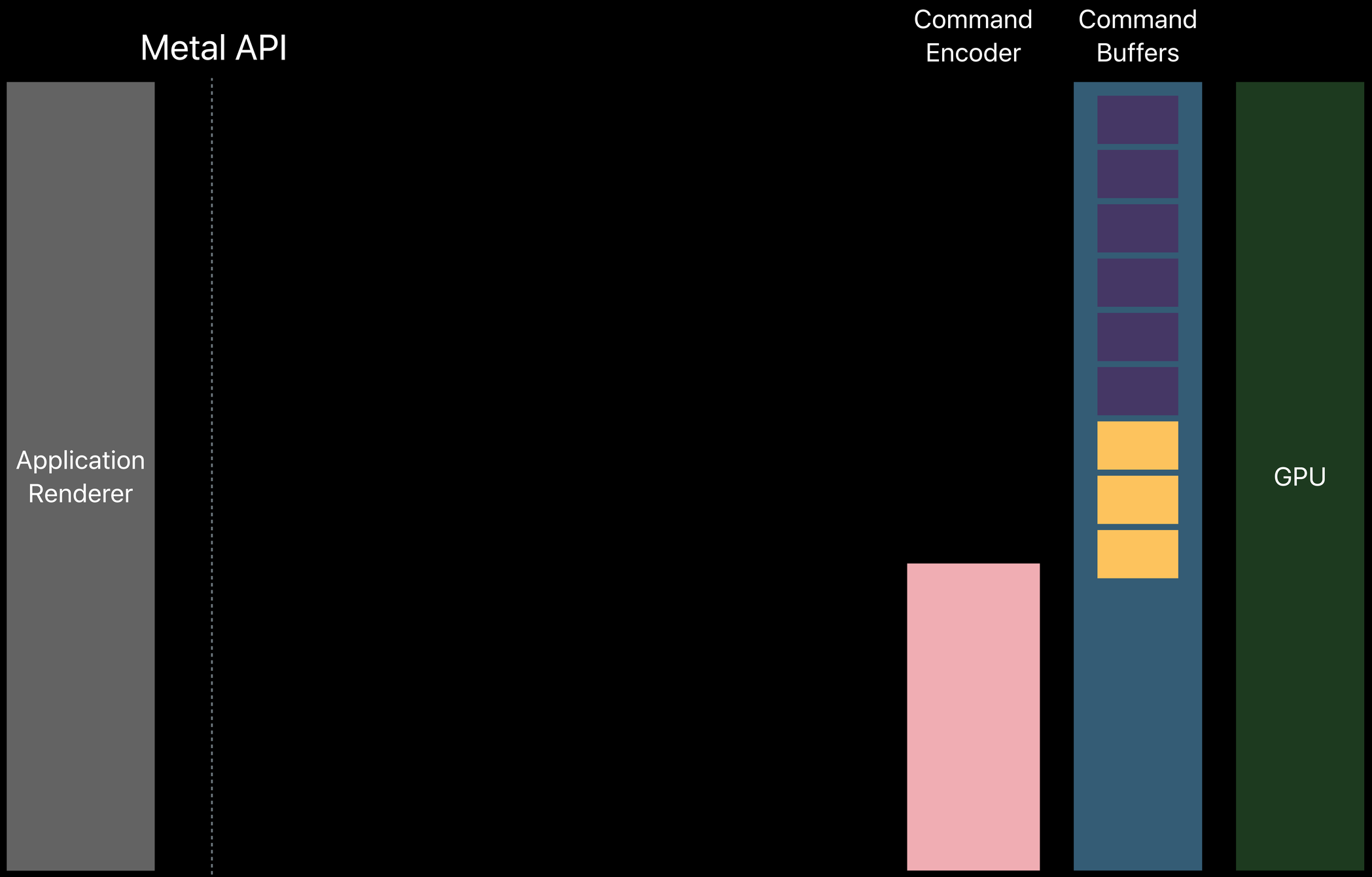

Metal API
Command Encoder
Command Buffers
Application Renderer
GPU

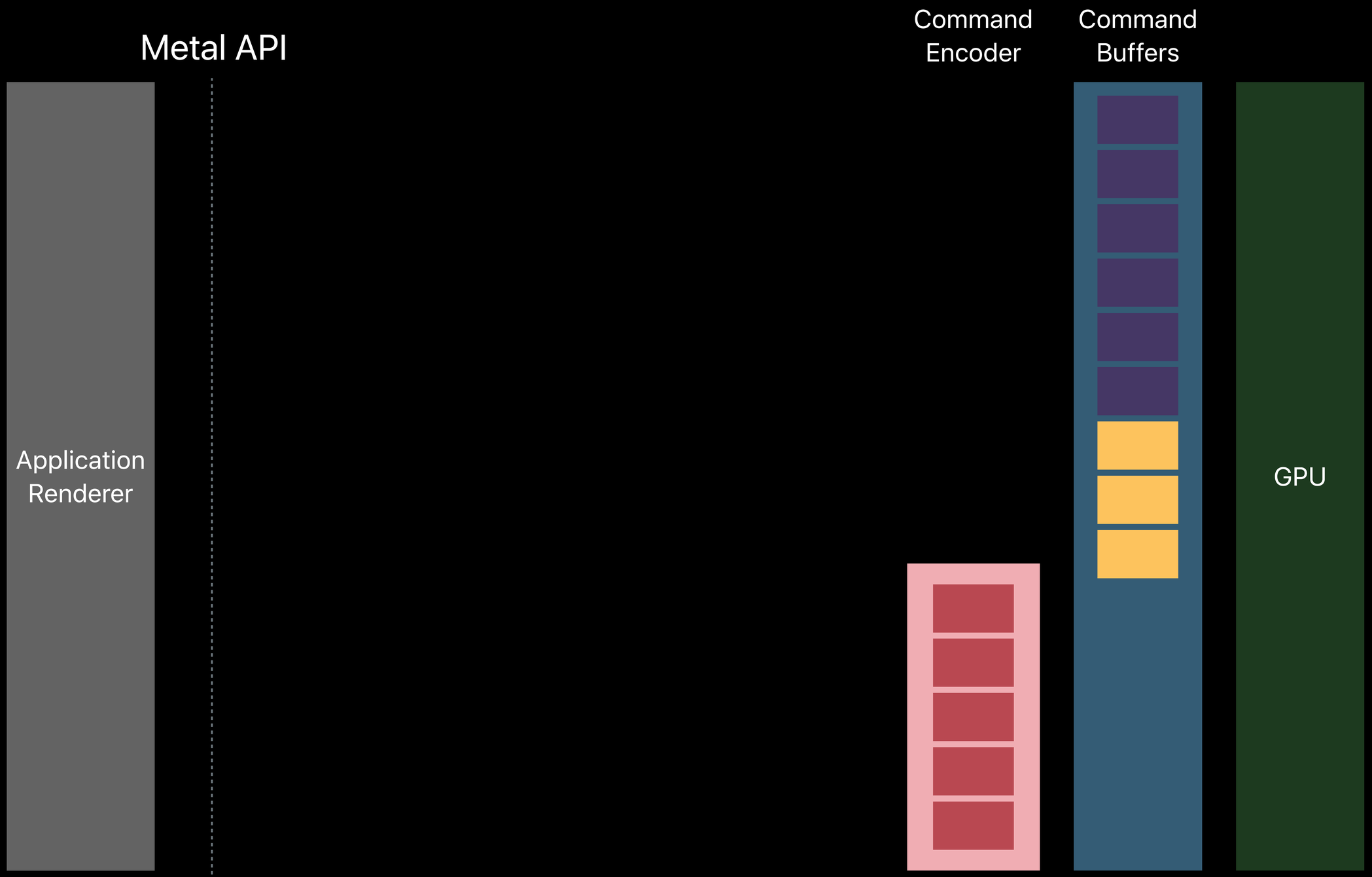

Metal API
Command Encoder
Command Buffers
Application Renderer
[computeEncoder endEncoding]
GPU

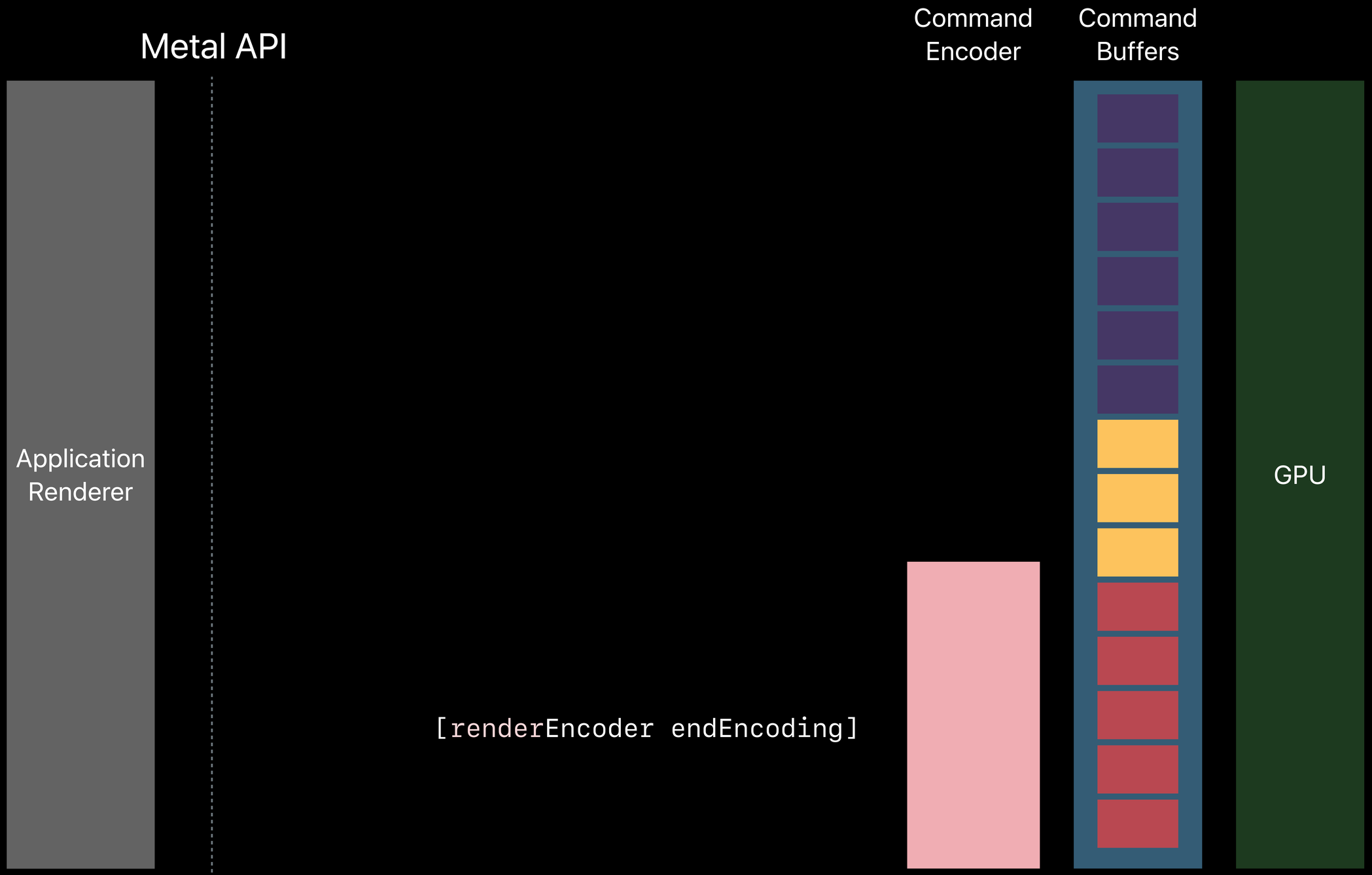

Metal API
Command Encoder
Command Buffers
Application Renderer
[computeEncoder endEncoding]
GPU

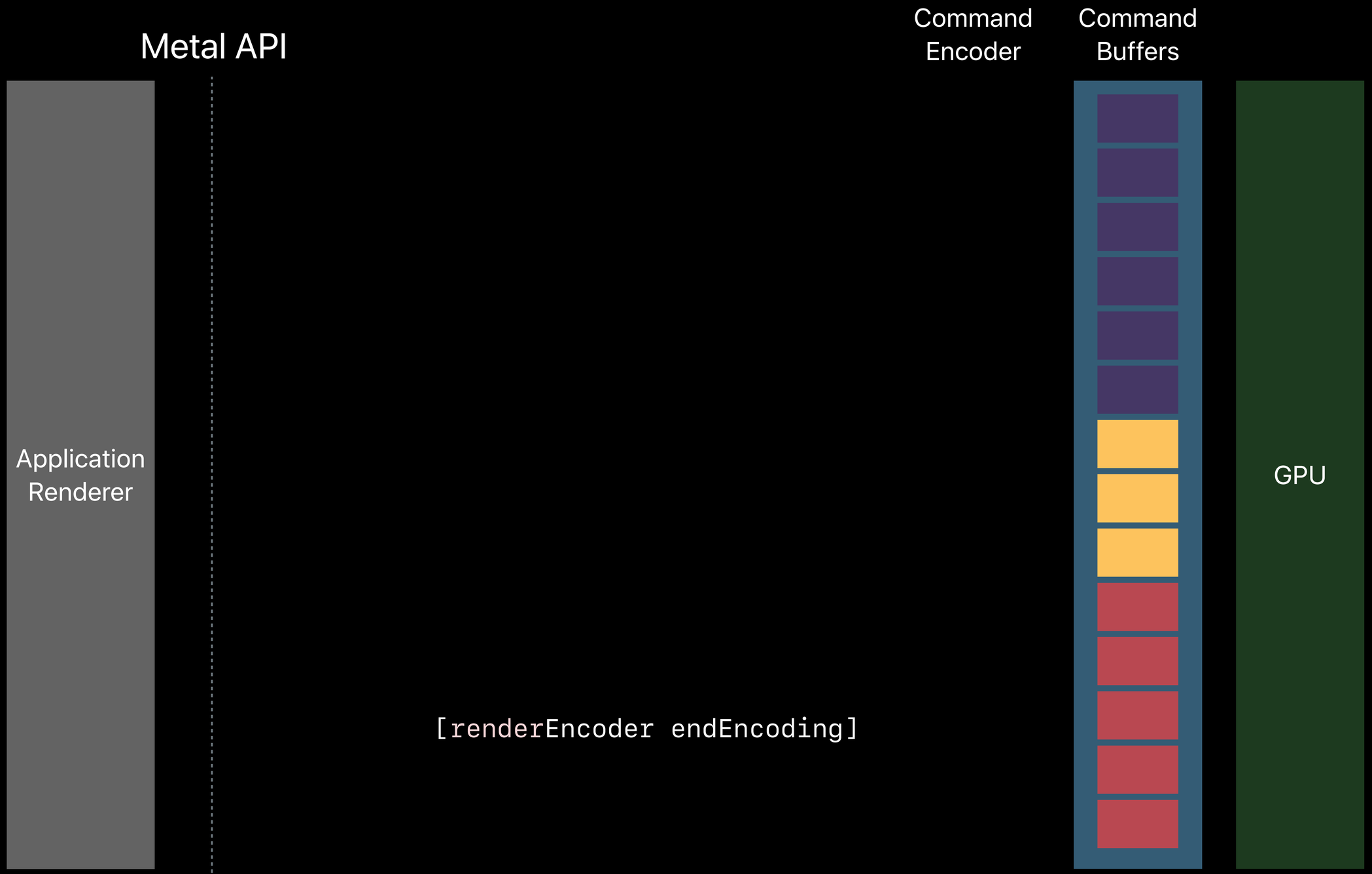

Metal API
Command Encoder
Command Buffers
Application Renderer
GPU

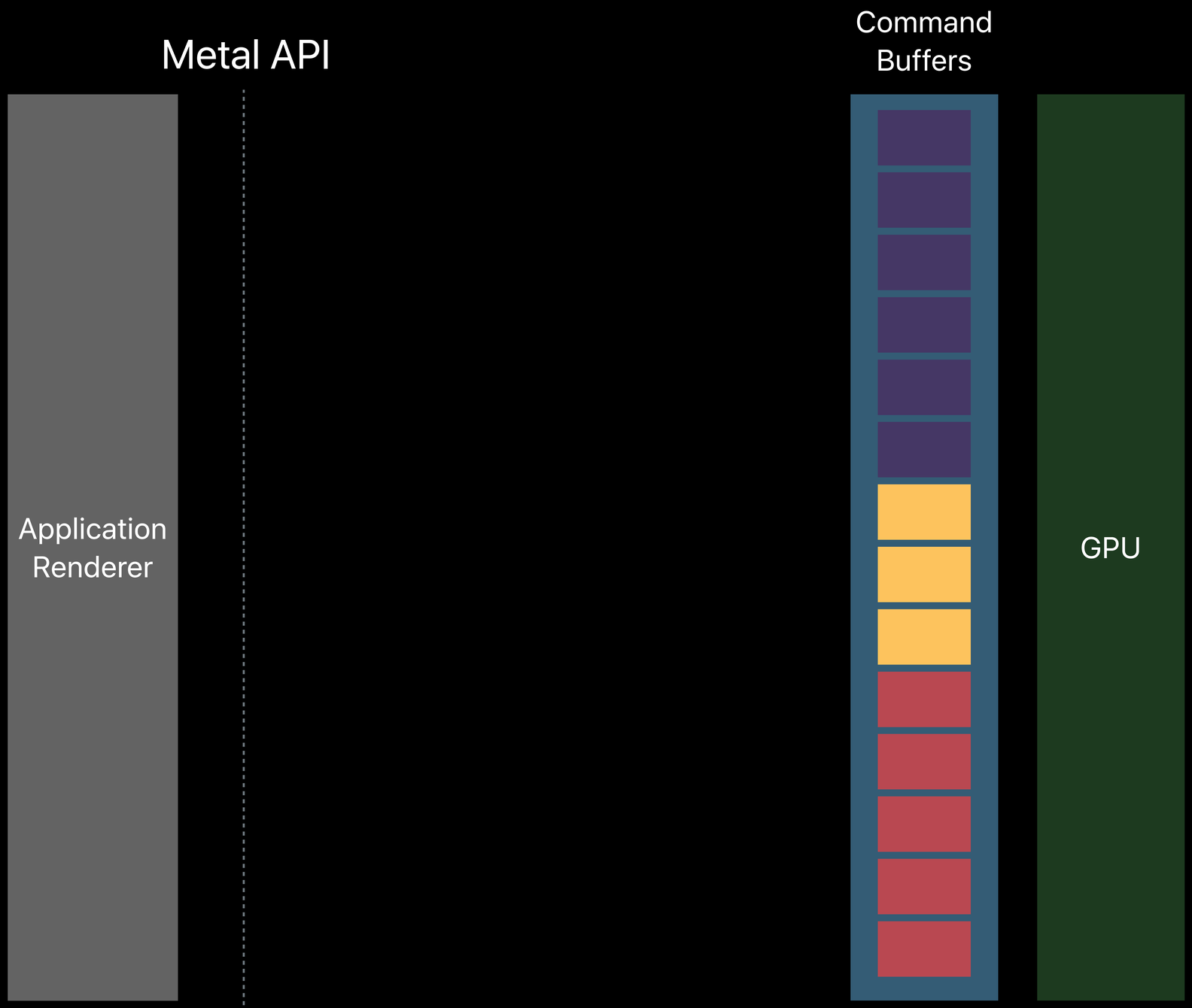

Metal API
Command Encoder
Command Buffers
Application Renderer
[commandBuffer
renderCommandEncoderWithDescriptor:]
GPU

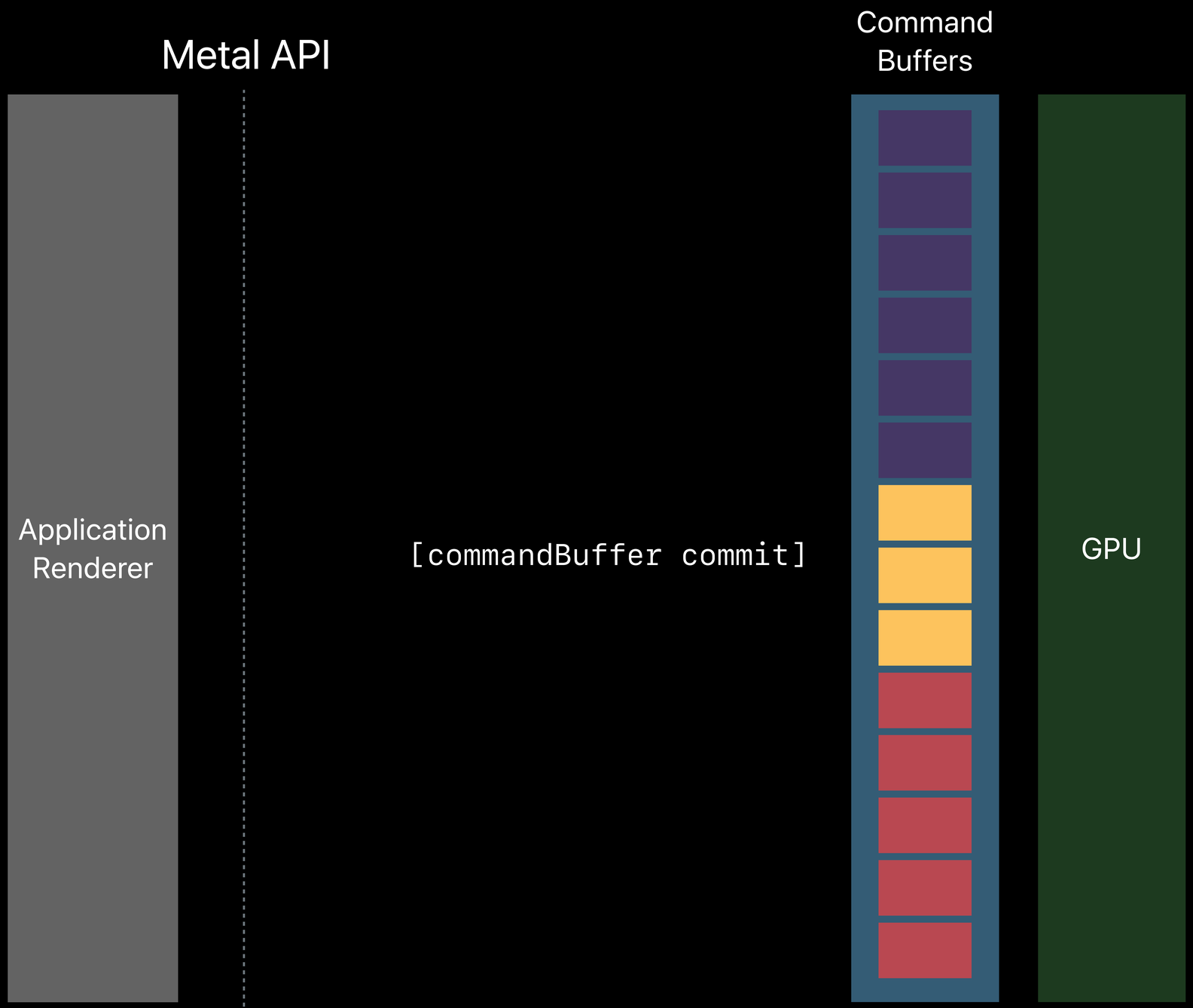

Metal API
Application
Renderer
Command
Encoder
Command
Buffers
GPU

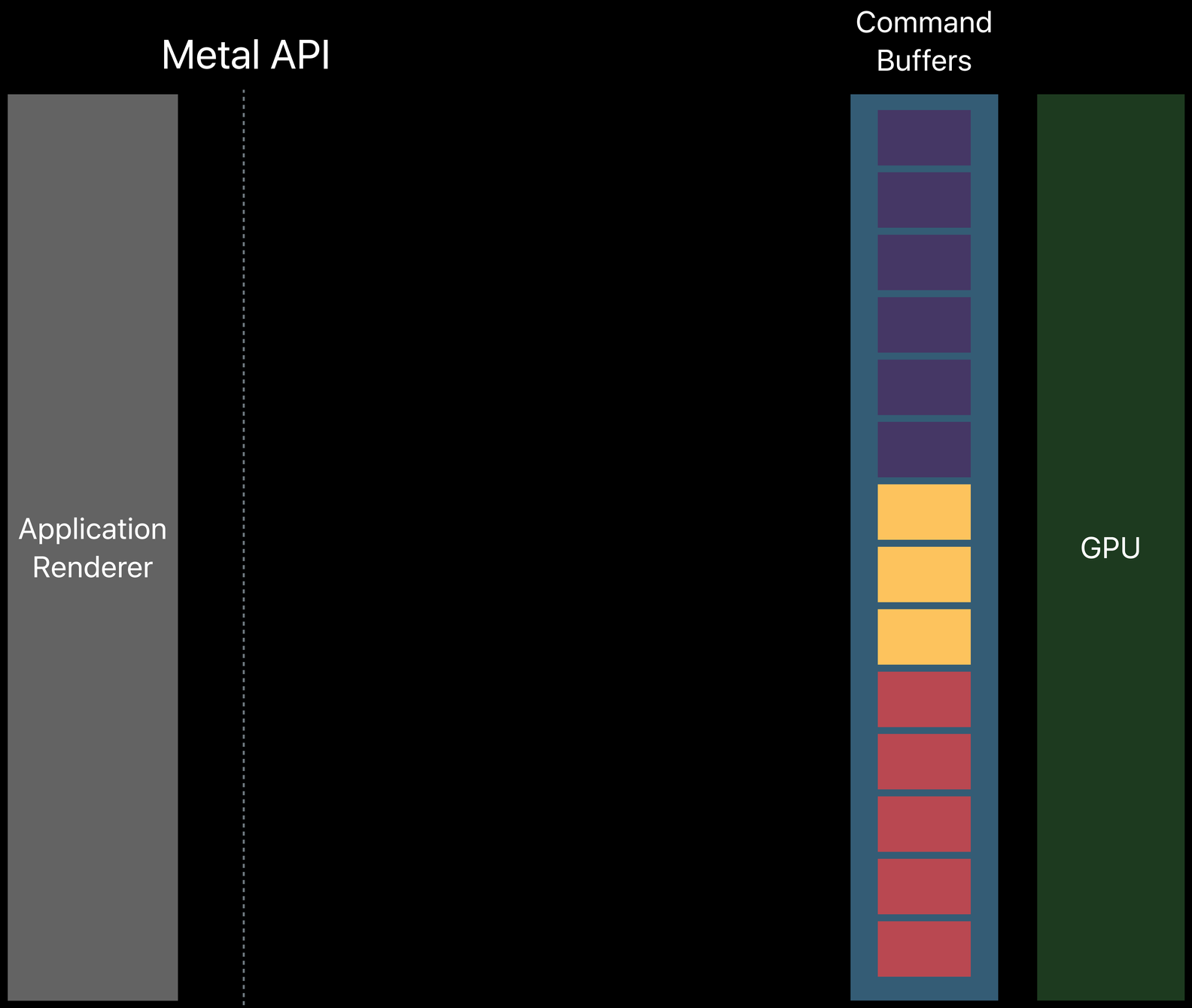

Metal API
Application
Renderer
Command
Encoder
Command
Buffers
GPU


Metal API
Command Encoder
Command Buffers
Application Renderer
[renderEncoder endEncoding]
GPU


Metal API
Command Encoder
Command Buffers
Application Renderer
[renderEncoder endEncoding]
GPU


Metal API
Command Buffers
Application Renderer
GPU


Metal API
Command Buffers
Application Renderer
[commandBuffer commit]
GPU


Metal API
Command Buffers
Application Renderer
GPU

Metal API

Command Buffers

Application Renderer

GPU

Submit Work

OpenGL	Metal
glFlush	[commandBuffer commit] [commandBuffer waitUntilScheduled]
glFinish	[commandBuffer commit] [commandBuffer waitUntilCompleted]
	[commandBuffer commit]
	[commandBuffer addCompletedHandler:] [commandBuffer commit]

Submit Work

OpenGL	Metal
glFlush	[commandBuffer commit] [commandBuffer waitUntilScheduled]
glFinish	[commandBuffer commit] [commandBuffer waitUntilCompleted]
	[commandBuffer commit]
	[commandBuffer addCompletedHandler:] [commandBuffer commit]

Submit Work

OpenGL		Metal
glFlush		[commandBuffer commit] [commandBuffer waitUntilScheduled]
glFinish		[commandBuffer commit] [commandBuffer waitUntilCompleted]
		[commandBuffer commit]
		[commandBuffer addCompletedHandler:] [commandBuffer commit]

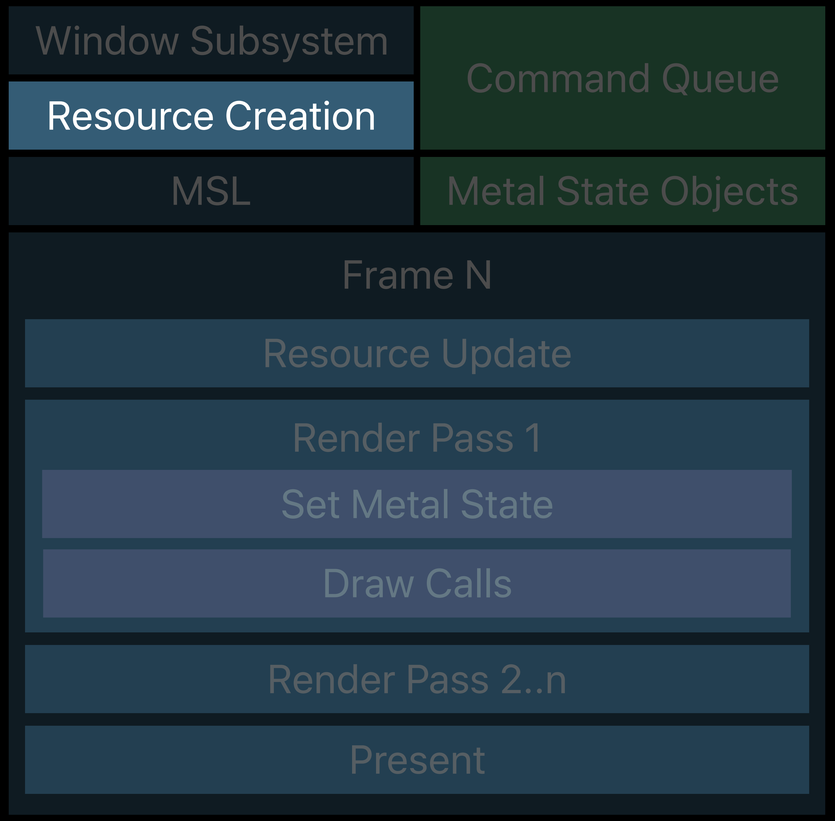
Submit Work

OpenGL		Metal
glFlush		[commandBuffer commit] [commandBuffer waitUntilScheduled]
glFinish		[commandBuffer commit] [commandBuffer waitUntilCompleted]
		[commandBuffer commit]
		[commandBuffer addCompletedHandler:] [commandBuffer commit]

Life of a Graphics App

Resource creation

OpenGL



Metal

Resources

Buffers

Textures

Samplers

Buffer Initialization

Buffer Initialization

Buffer Initialization

Buffer Initialization


```
glGenBuffers(1, &myBuffer);
glBindBuffer(GL_ARRAY_BUFFER, myBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeInBytes, dataPtr, usage, GL_DYNAMIC_DRAW);
```

OpenGL

Buffer Initialization


```
glGenBuffers(1, &myBuffer);
glBindBuffer(GL_ARRAY_BUFFER, myBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeInBytes, dataPtr, usage, GL_DYNAMIC_DRAW);
```

OpenGL

Buffer Initialization

OpenGL

```
glGenBuffers(1, &myBuffer);
glBindBuffer(GL_ARRAY_BUFFER, myBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeInBytes, dataPtr, usage, GL_DYNAMIC_DRAW);
```

Metal

```
id<MTLBuffer> myBuffer = [device newBufferWithBytes:dataPtr
                                             length:sizeInBytes
                                            options:MTLResourceStorageModeShared];
```

Buffer Initialization


```
glGenBuffers(1, &myBuffer);
glBindBuffer(GL_ARRAY_BUFFER, myBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeInBytes, dataPtr, usage, GL_DYNAMIC_DRAW);
```

OpenGL

Metal

```
id<MTLBuffer> myBuffer = [device newBufferWithBytes:dataPtr
                                             length:sizeInBytes
                                            options:MTLResourceStorageModeShared];
```

Buffer Initialization

OpenGL

```
glGenBuffers(1, &myBuffer);
glBindBuffer(GL_ARRAY_BUFFER, myBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeInBytes, dataPtr, usage, GL_DYNAMIC_DRAW);
```

Metal

```
id<MTLBuffer> myBuffer = [device newBufferWithBytes:dataPtr
                                             length:sizeInBytes
                                            options:MTLResourceStorageModeShared];
```

Buffer Initialization


```
glGenBuffers(1, &myBuffer);
glBindBuffer(GL_ARRAY_BUFFER, myBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeInBytes, dataPtr, usage, GL_DYNAMIC_DRAW);
```

OpenGL

Metal

```
id<MTLBuffer> myBuffer = [device newBufferWithBytes:dataPtr
                                             length:sizeInBytes
                                            options:MTLResourceStorageModeShared];
```

Storage Patterns

OpenGL — usage enum

- No direct control over storage
- Hint to the driver via [STREAM | STATIC | DYNAMIC] x [DRAW | READ | COPY]

Storage Patterns

OpenGL — usage enum

- No direct control over storage
- Hint to the driver via [STREAM | STATIC | DYNAMIC] x [DRAW | READ | COPY]

Metal — storage mode

- Specific storage — Shared | Managed | Private
- Specific performance characteristics and best practices

OpenGL Texture Initialization

OpenGL Texture Initialization

OpenGL Texture Initialization

OpenGL Texture Initialization


```
glActiveTexture(GL_TEXTURE0);
glGenTextures(1, &myTexture);
glBindTexture(GL_TEXTURE_2D, myTexture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
for (int mip : mipLevels)
    glTexImage2D(GL_TEXTURE_2D, mip, internalFormat, mipWidth, mipHeight);
for (int mip : mipLevels)
    glTexSubImage2D(GL_TEXTURE_2D, mip, x, y, width, height, format, type, dataPtr);
```

OpenGL Texture Initialization


```
glActiveTexture(GL_TEXTURE0);
glGenTextures(1, &myTexture);
glBindTexture(GL_TEXTURE_2D, myTexture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
for (int mip : mipLevels)
    glTexImage2D(GL_TEXTURE_2D, mip, internalFormat, mipWidth, mipHeight);
for (int mip : mipLevels)
    glTexSubImage2D(GL_TEXTURE_2D, mip, x, y, width, height, format, type, dataPtr);
```

OpenGL Texture Initialization


```
glActiveTexture(GL_TEXTURE0);
glGenTextures(1, &myTexture);
glBindTexture(GL_TEXTURE_2D, myTexture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
for (int mip : mipLevels)
    glTexImage2D(GL_TEXTURE_2D, mip, internalFormat, mipWidth, mipHeight);
for (int mip : mipLevels)
    glTexSubImage2D(GL_TEXTURE_2D, mip, x, y, width, height, format, type, dataPtr);
```

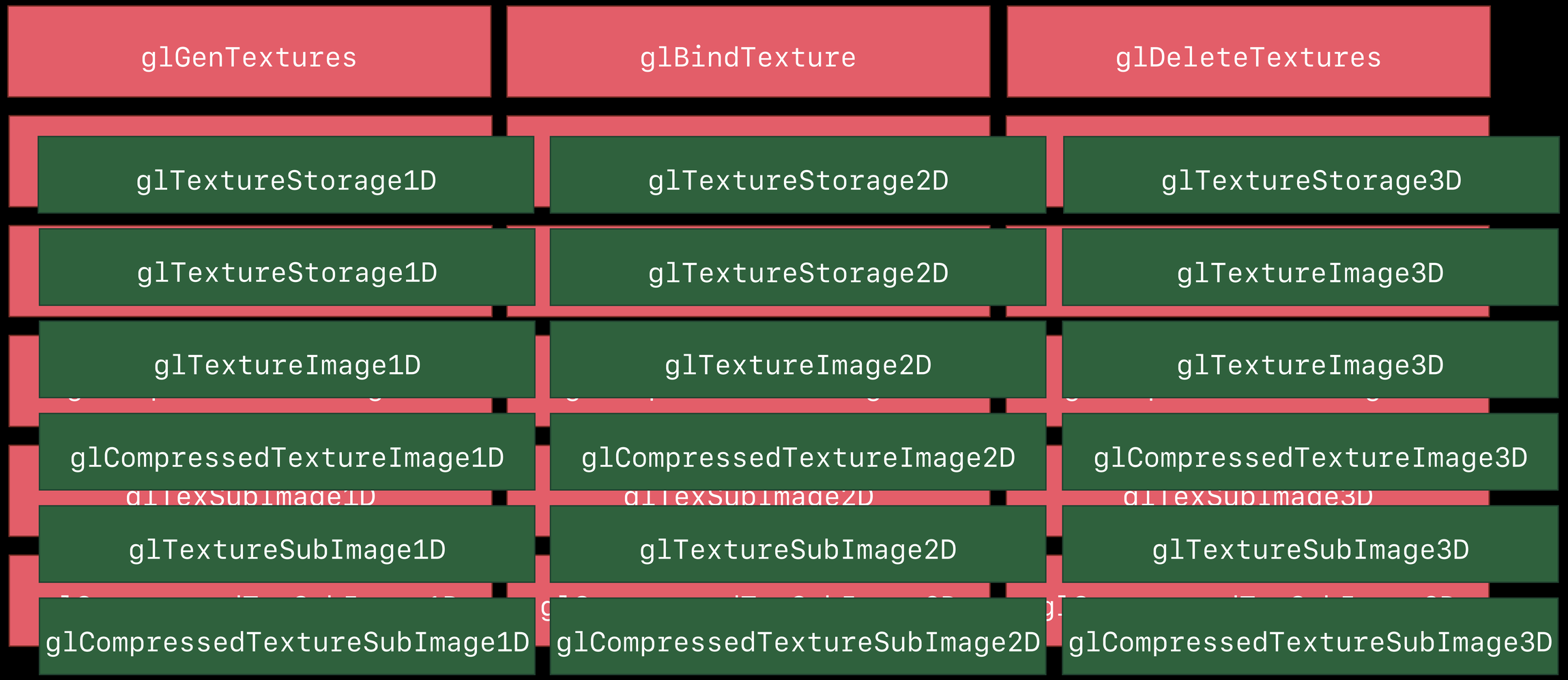
OpenGL Texture Initialization


```
glActiveTexture(GL_TEXTURE0);
glGenTextures(1, &myTexture);
glBindTexture(GL_TEXTURE_2D, myTexture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
for (int mip : mipLevels)
    glTexImage2D(GL_TEXTURE_2D, mip, internalFormat, mipWidth, mipHeight);
for (int mip : mipLevels)
    glTexSubImage2D(GL_TEXTURE_2D, mip, x, y, width, height, format, type, dataPtr);
```

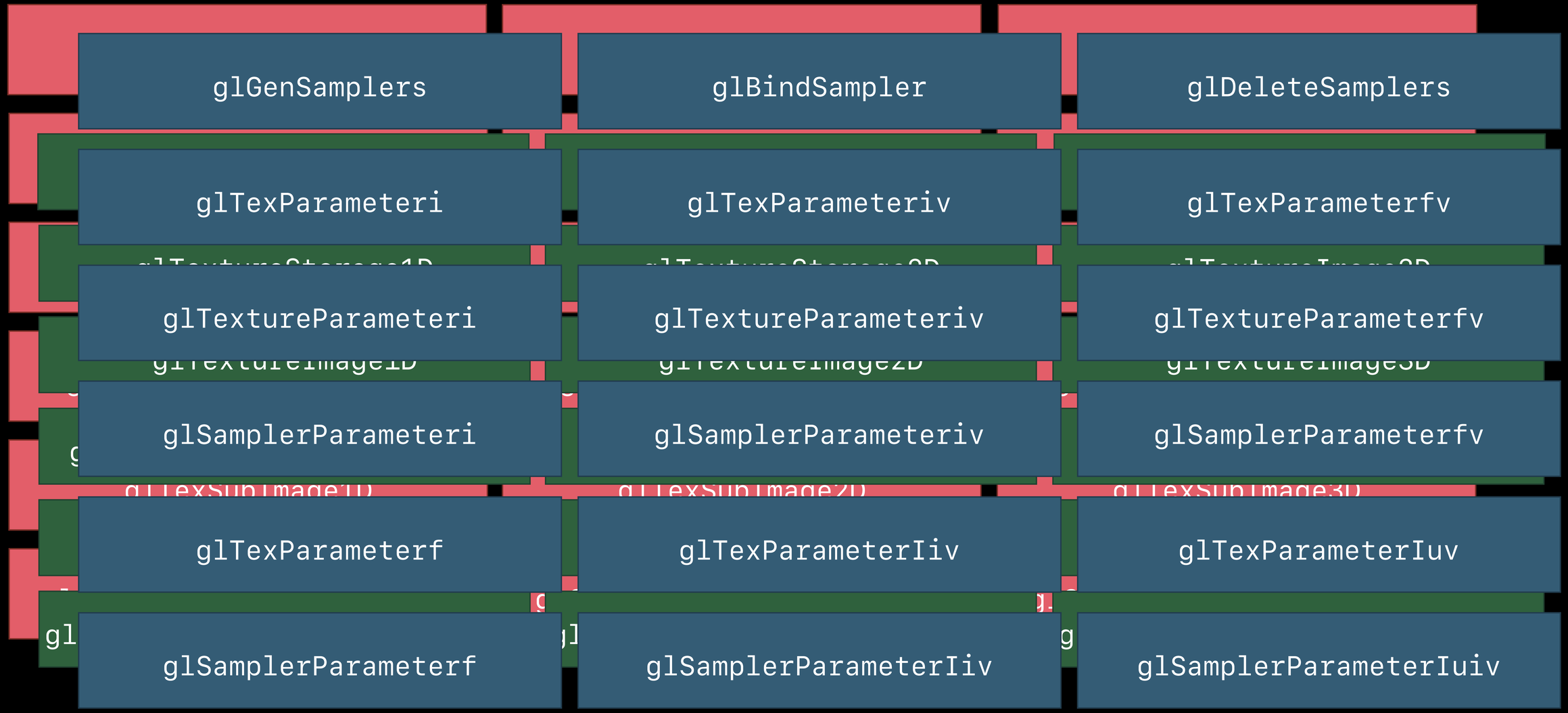
OpenGL Texture APIs

glGenTextures	glBindTexture	glDeleteTextures
glTexStorage1D	glTexStorage2D	glTexStorage3D
glTexImage1D	glTexImage2D	glTexImage3D
glCompressedTexImage1D	glCompressedTexImage2D	glCompressedTexImage3D
glTexSubImage1D	glTexSubImage2D	glTexSubImage3D
glCompressedTexSubImage1D	glCompressedTexSubImage2D	glCompressedTexSubImage3D

OpenGL Texture APIs



OpenGL Texture and Sampler APIs



Metal Texture Initialization

Metal Texture Initialization

Metal Texture Initialization


```
MTLTextureDescriptor *textureDescriptor = [[MTLTextureDescriptor alloc] init];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;
id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];
```

Metal Texture Initialization


```
MTLTextureDescriptor *textureDescriptor = [[MTLTextureDescriptor alloc] init];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;
id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];
```

Metal Texture Initialization


```
MTLTextureDescriptor *textureDescriptor = [[MTLTextureDescriptor alloc] init];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;
id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];
```

Metal Texture Initialization


```
MTLTextureDescriptor *textureDescriptor = [[MTLTextureDescriptor alloc] init];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;
id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];
```

Metal Texture Initialization


```
MTLTextureDescriptor *textureDescriptor = [[MTLTextureDescriptor alloc] init];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;
id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];
```

Metal Texture Initialization


```
MTLTextureDescriptor *textureDescriptor = [[MTLTextureDescriptor alloc] init];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;
id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];

MTLSamplerDescriptor *samplerDesc = [[MTLSamplerDescriptor alloc] init];
samplerDesc.mipFilter = MTLSamplerMipFilterNearest;
samplerDesc.magFilter = samplerDesc.minFilter = MTLSamplerMinMagFilterNearest;
id<MTLSamplerState> sampler = [device newSamplerStateWithDescriptor:samplerDesc];
```

Metal Texture Initialization


```
MTLTextureDescriptor *textureDescriptor = [[MTLTextureDescriptor alloc] init];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;
id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];

MTLSamplerDescriptor *samplerDesc = [[MTLSamplerDescriptor alloc] init];
samplerDesc.mipFilter = MTLSamplerMipFilterNearest;
samplerDesc.magFilter = samplerDesc.minFilter = MTLSamplerMinMagFilterNearest;
id<MTLSamplerState> sampler = [device newSamplerStateWithDescriptor:samplerDesc];
```

Metal Texture Initialization


```
MTLTextureDescriptor *textureDescriptor = [[MTLTextureDescriptor alloc] init];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;
id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];

MTLSamplerDescriptor *samplerDesc = [[MTLSamplerDescriptor alloc] init];
samplerDesc.mipFilter = MTLSamplerMipFilterNearest;
samplerDesc.magFilter = samplerDesc.minFilter = MTLSamplerMinMagFilterNearest;
id<MTLSamplerState> sampler = [device newSamplerStateWithDescriptor:samplerDesc];
```

Metal Texture Initialization


```
MTLTextureDescriptor *textureDescriptor = [[MTLTextureDescriptor alloc] init];
textureDescriptor.pixelFormat = MTLPixelFormatBGRA8Unorm;
textureDescriptor.width = textureDescriptor.height = 512;
textureDescriptor.storageMode = MTLStorageModeShared;
id<MTLTexture> texture = [device newTextureWithDescriptor:textureDescriptor];

MTLSamplerDescriptor *samplerDesc = [[MTLSamplerDescriptor alloc] init];
samplerDesc.mipFilter = MTLSamplerMipFilterNearest;
samplerDesc.magFilter = samplerDesc.minFilter = MTLSamplerMinMagFilterNearest;
id<MTLSamplerState> sampler = [device newSamplerStateWithDescriptor:samplerDesc];
```

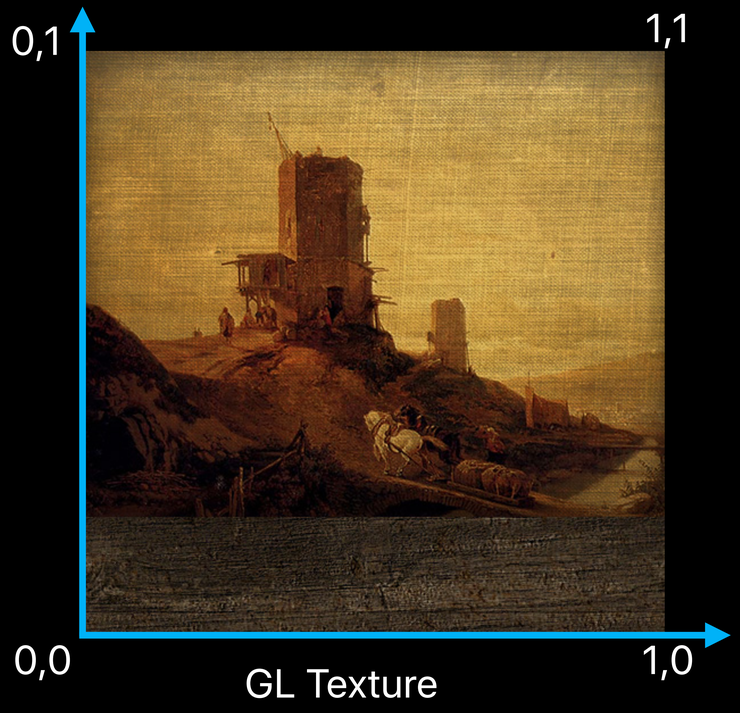
Metal Texture Initialization

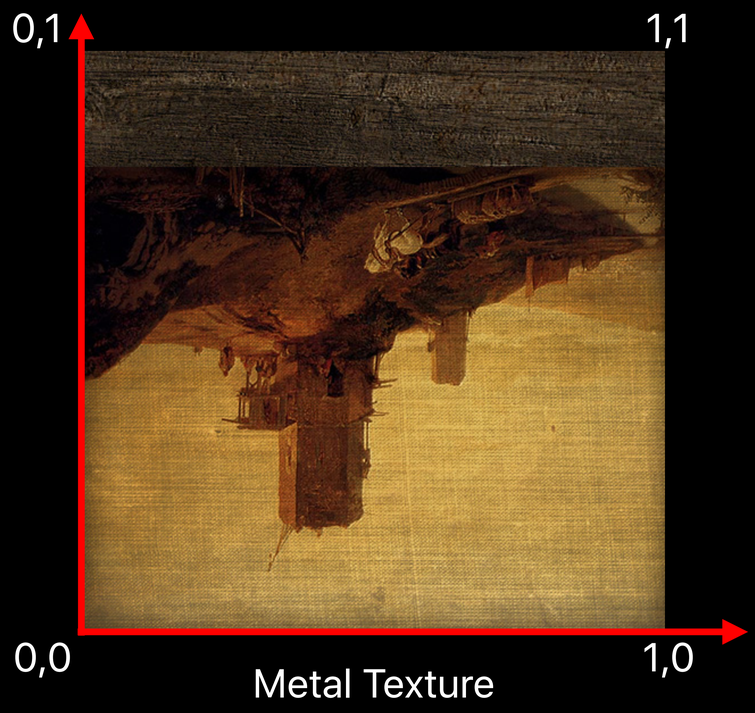

```
NSUInteger bytesPerRow = 4 * image.width;
MTLRegion region = MTLRegionMake2D(0, 0, image.width, image.width);
[texture replaceRegion:region
           mipmapLevel:0
             withBytes:image.dataPtr
           bytesPerRow:bytesPerRow];
```

Metal Texture Initialization

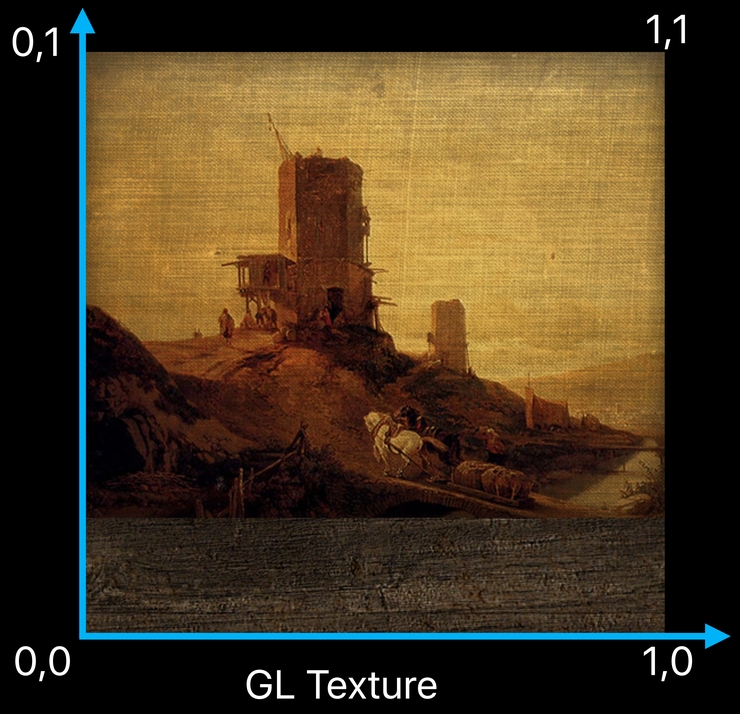

```
NSUInteger bytesPerRow = 4 * image.width;
MTLRegion region = MTLRegionMake2D(0, 0, image.width, image.width);
[texture replaceRegion:region
           mipmapLevel:0
             withBytes:image.dataPtr
           bytesPerRow:bytesPerRow];
```

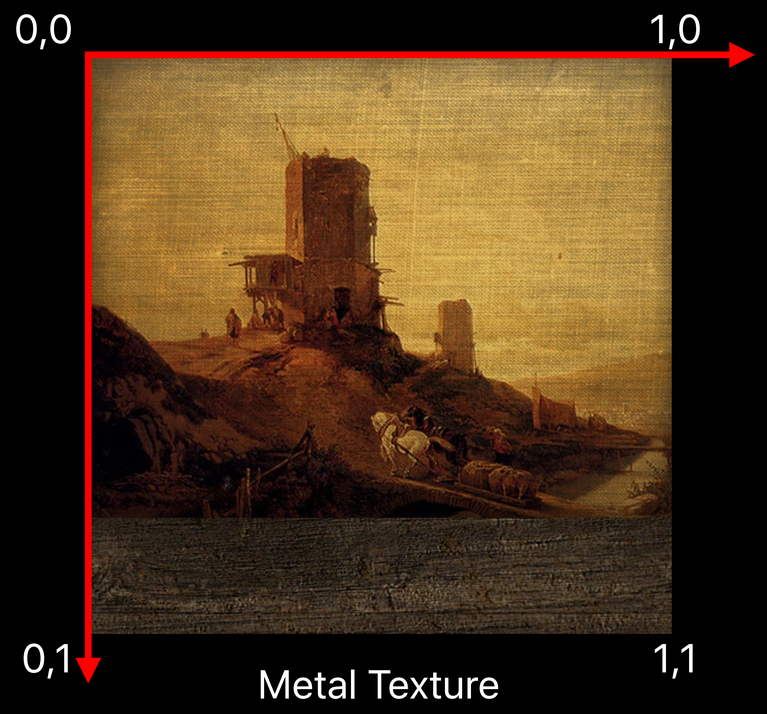
Coordinate Changes — Flip Your Y!





Coordinate Changes — Flip Your Y!





Storage Modes

Shared storage

Private storage

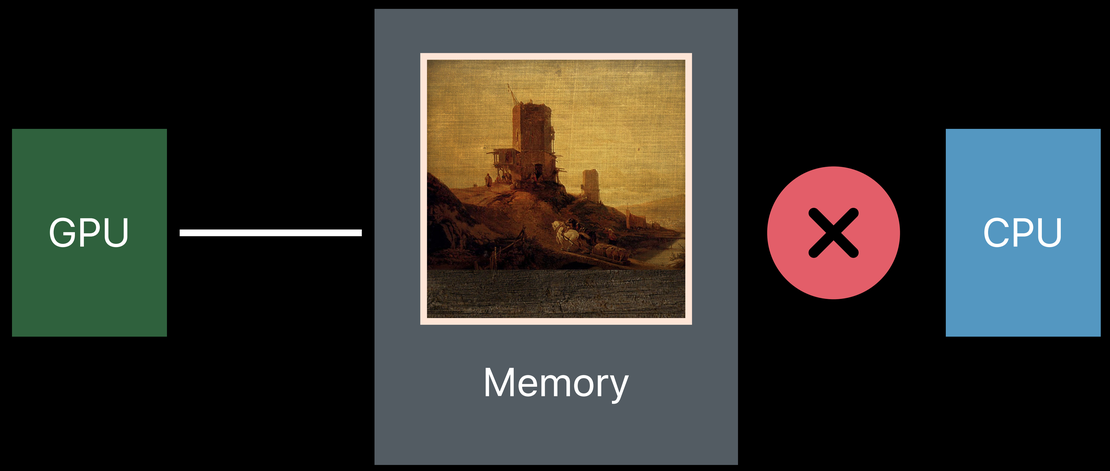
Managed storage

Shared Storage

CPU and GPU access

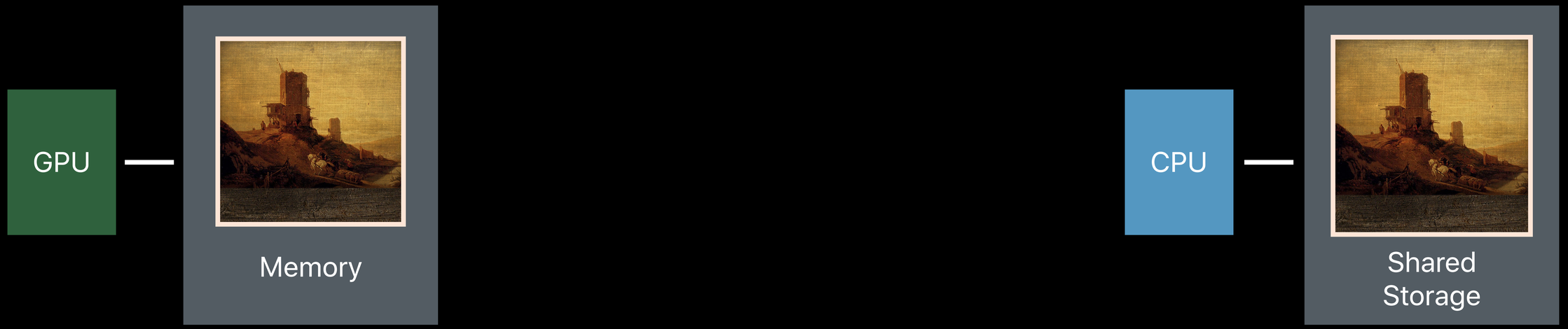
Private Storage

GPU only



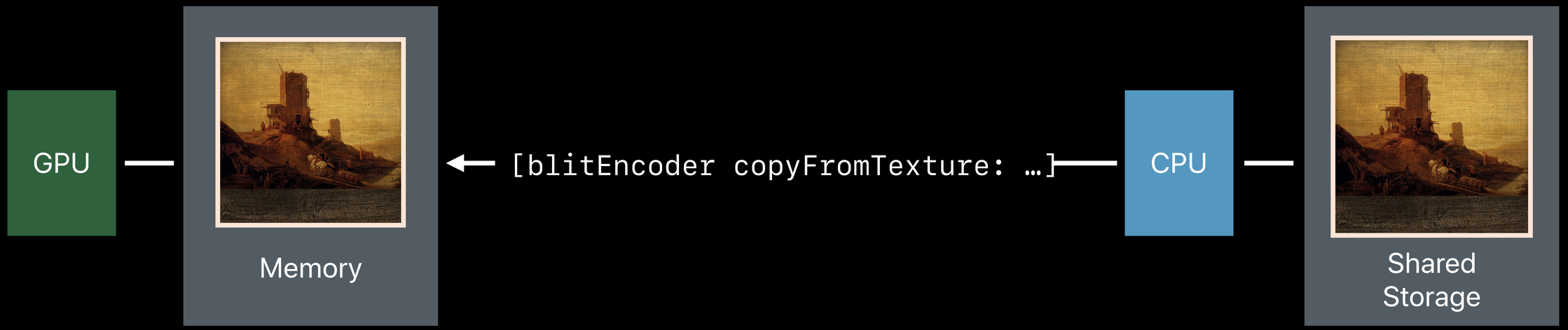
Private Storage

GPU only or blit from CPU



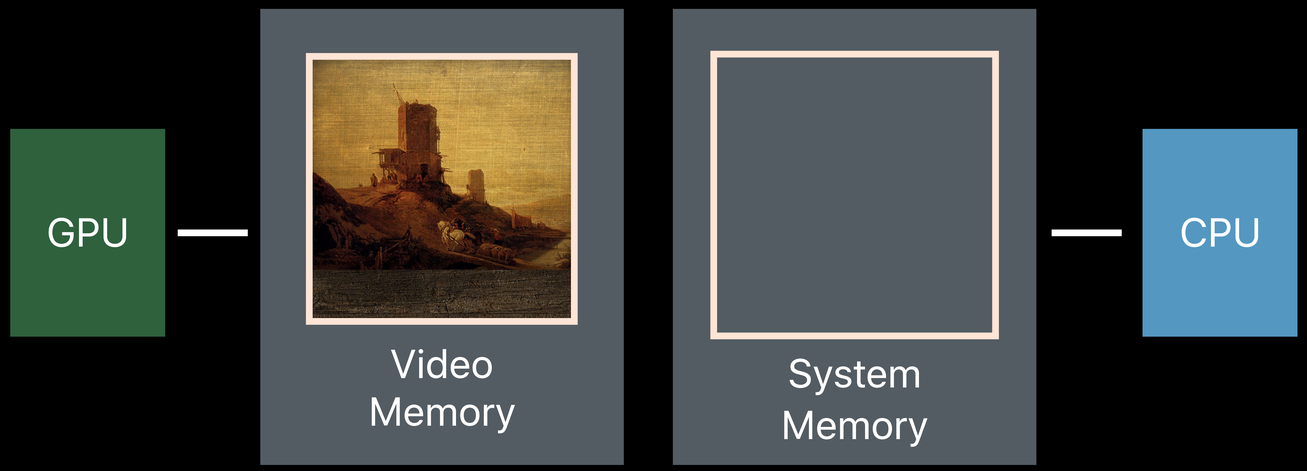
Private Storage

GPU only or blit from CPU



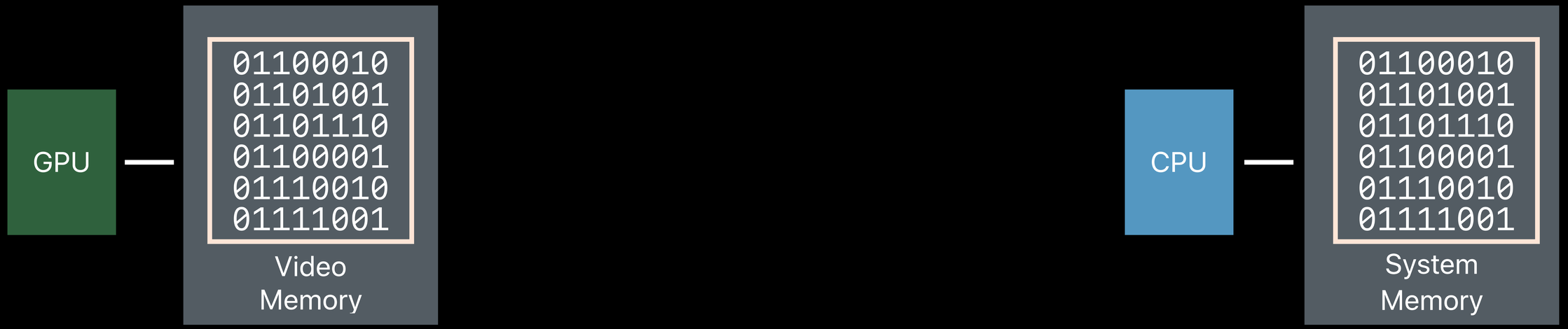
Private Storage

GPU video memory



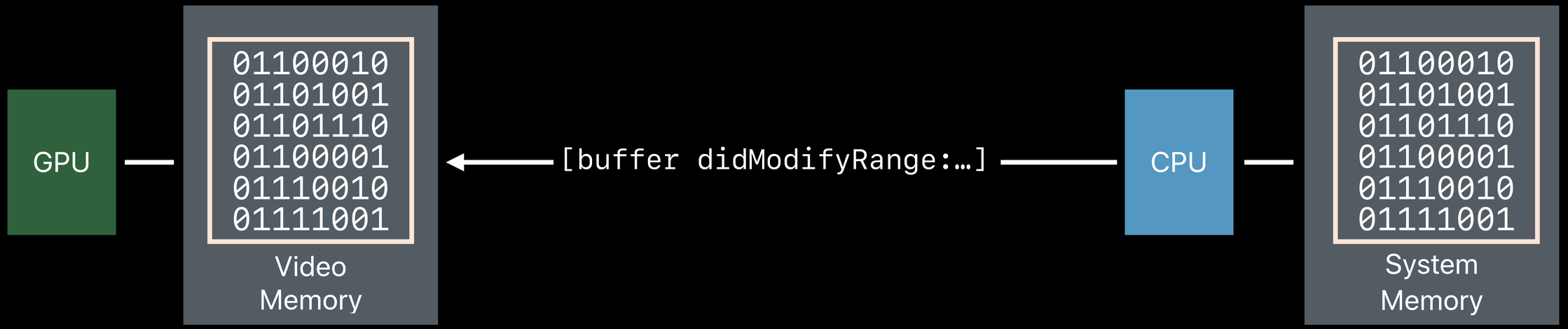
Managed Storage

CPU and GPU access with sync (macOS only)



Managed Storage

CPU and GPU access with sync (macOS only)



Shared Storage

Typical uses

	Private	Shared	Managed
macOS	Static buffers, textures Render targets	Small dynamic buffers	Dynamic buffers with small updates
iOS	Static buffers, textures Render targets	Dynamic buffers Dynamic textures	

Shared Storage

Typical uses

	Private	Shared	Managed
macOS	Static buffers, textures Render targets	Small dynamic buffers	Dynamic buffers with small updates
iOS	Static buffers, textures Render targets	Dynamic buffers Dynamic textures	

Shared Storage

Typical uses

	Private	Shared	Managed
macOS	Static buffers, textures Render targets	Small dynamic buffers	Dynamic buffers with small updates
iOS	Static buffers, textures Render targets	Dynamic buffers Dynamic textures	

Shared Storage

Typical uses

	Private	Shared	Managed
macOS	Static buffers, textures Render targets	Small dynamic buffers	Dynamic buffers with small updates
iOS	Static buffers, textures Render targets	Dynamic buffers Dynamic textures	

Shared Storage

Typical uses

	Private	Shared	Managed
macOS	Static buffers, textures Render targets	Small dynamic buffers	Dynamic buffers with small updates
iOS	Static buffers, textures Render targets	Dynamic buffers Dynamic textures	

Shared Storage

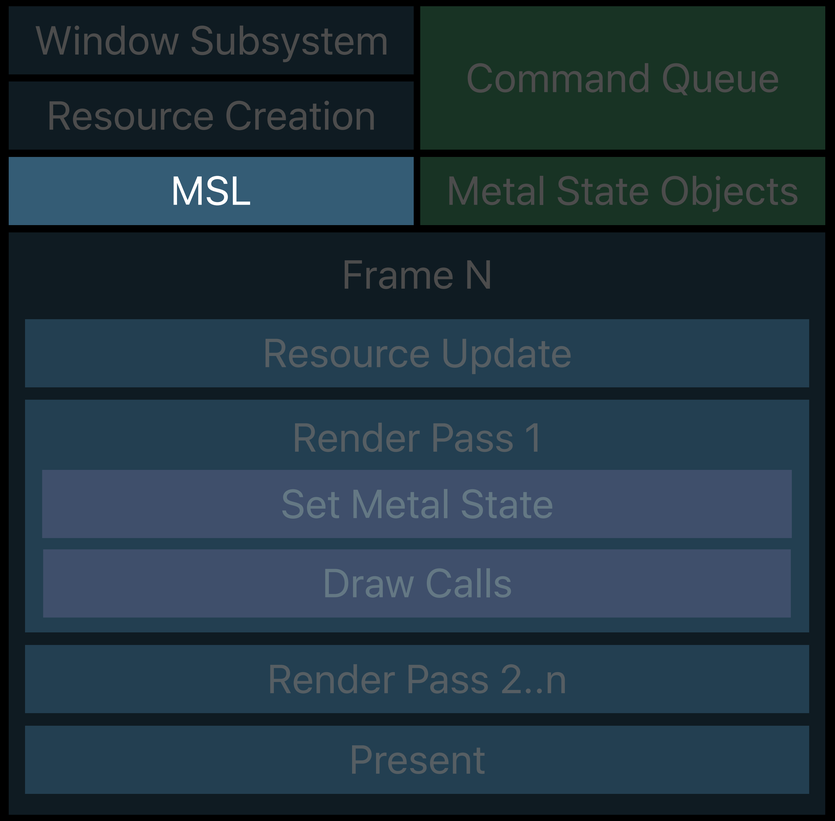
Typical uses

	Private	Shared	Managed
macOS	Static buffers, textures Render targets	Small dynamic buffers	Dynamic buffers with small updates
iOS	Static buffers, textures Render targets	Dynamic buffers Dynamic textures	

Life of a Graphics App

Shaders

OpenGL



Metal

Shader Compilation in OpenGL

Create	`myShader = glCreateShader(shaderType);`
Pass Source	`glShaderSource(myShader, numStrings, strings, lenStrings);`
JIT	`glCompileShader(myShader);`
Validate	`glGetShaderiv(myShader, GL_COMPILE_STATUS, &compiled);`

Shader Compilation in OpenGL

Create	`myShader = glCreateShader(shaderType);`
Pass Source	`glShaderSource(myShader, numStrings, strings, lenStrings);`
JIT	`glCompileShader(myShader);`
Validate	`glGetShaderiv(myShader, GL_COMPILE_STATUS, &compiled);`

Shader Compilation in OpenGL

Create	`myShader = glCreateShader(shaderType);`
Pass Source	`glShaderSource(myShader, numStrings, strings, lenStrings);`
JIT	`glCompileShader(myShader);`
Validate	`glGetShaderiv(myShader, GL_COMPILE_STATUS, &compiled);`

Shader Compilation in OpenGL

Create	`myShader = glCreateShader(shaderType);`
Pass Source	`glShaderSource(myShader, numStrings, strings, lenStrings);`
JIT	`glCompileShader(myShader);`
Validate	`glGetShaderiv(myShader, GL_COMPILE_STATUS, &compiled);`

Shader Compilation in OpenGL

Create	`myShader = glCreateShader(shaderType);`
Pass Source	`glShaderSource(myShader, numStrings, strings, lenStrings);`
JIT	`glCompileShader(myShader);`
Validate	`glGetShaderiv(myShader, GL_COMPILE_STATUS, &compiled);`

Shader Compilation in OpenGL

Create	`myShader = glCreateShader(shaderType);`
Pass Source	`glShaderSource(myShader, numStrings, strings, lenStrings);`
JIT	`glCompileShader(myShader);`
Validate	`glGetShaderiv(myShader, GL_COMPILE_STATUS, &compiled);`

Shader Compilation in Metal

Metal — Precompile into MTLLibrary at build time

Placed in app bundle for run time retrieval:

```
// Get default library that's bundled with the app
id<MTLLibrary> library = [device newDefaultLibrary];

// Get function from the library
id<MTLFunction> func = [library newFunctionWithName:@"myVertexShader"];
```

Metal Shading Language (MSL)

Metal Shading Language (MSL)

Based on C++

- Classes, templates, structures, enums, namespaces

Metal Shading Language (MSL)

Based on C++

- Classes, templates, structures, enums, namespaces

Built-in types for vectors and matrices

Metal Shading Language (MSL)

Based on C++

- Classes, templates, structures, enums, namespaces

Built-in types for vectors and matrices

Built-in functions and operators

Metal Shading Language (MSL)

Based on C++

- Classes, templates, structures, enums, namespaces

Built-in types for vectors and matrices

Built-in functions and operators

Built-in classes for textures and samplers

Metal Shading Language (MSL)

Based on C++

- Classes, templates, structures, enums, namespaces

Built-in types for vectors and matrices

Built-in functions and operators

Built-in classes for textures and samplers

Metal Shading Language (MSL)

Based on C++

- Classes, templates, structures, enums, namespaces

Built-in types for vectors and matrices

Built-in functions and operators

Built-in classes for textures and samplers

Unified for graphics and compute

Metal Shading Language (MSL)

Based on C++

- Classes, templates, structures, enums, namespaces

Built-in types for vectors and matrices

Built-in functions and operators

Built-in classes for textures and samplers

Unified for graphics and compute

Xcode can precompile, show syntax colors, and auto-complete

GLSL

MSL

```
void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(…)
```

```
void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(…)
```

```
void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(…)
```

```
void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(…)
```

```
void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(…)
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate

void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate

void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate

void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate

void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate

void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate

void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate

void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate

void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate

void main()
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
{
    VertexOutput out;
    out.clipPos = mvp * vertex.modelPos;
    out.texCoord = vertex.texCoord;
    return out;
}
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
{
    VertexOutput out;
    out.clipPos = mvp * vertex.modelPos;
    out.texCoord = vertex.texCoord;
    return out;
}
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                                Vertex   * vertex [[ stage_in ]])
{
    VertexOutput out;
    out.clipPos = mvp * vertex.modelPos;
    out.texCoord = vertex.texCoord;
    return out;
}
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
{
    VertexOutput out;
    out.clipPos = mvp * vertex.modelPos;
    out.texCoord = vertex.texCoord;
    return out;
}
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                   Vertex   * vertex [[ stage_in ]])
{
    VertexOutput out;
    out.clipPos = mvp * vertex.modelPos;
    out.texCoord = vertex.texCoord;
    return out;
}
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```
vertex VertexOutput myVertexShader(constant float4x4 & mvp    [[ buffer(1) ]],
                                                Vertex   * vertex [[ stage_in ]])
{
    VertexOutput out;
    out.clipPos = mvp * vertex.modelPos;
    out.texCoord = vertex.texCoord;
    return out;
}
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

```glsl
uniform mat4 mvp;
layout (location = 0) in vec4 aModelPos; // attribute for model space position
layout (location = 1) in vec4 aTexCoord; // attribute for model texture coordinate
out vec2 vTexCoord; // varying for output coordinate
void main()
    gl_Position = mvp * aModelPos;
    vTexCoord = aTexCoord;
}
```

GLSL

MSL

```msl
#include <simd/simd.h>
typedef struct {
    float4 modelPos [[ attribute(0) ]];
    float2 texCoord [[ attribute(1) ]];
} Vertex;
typedef struct {
    float4 pos [[ position ]];
    float2 texCoord;
} VertexOutput;
```

Notes About Buffer Data

Pay attention to alignment

Type	Alignment
`float3, int3, uint3`	16 bytes
`float3x3, float4x3`	16 bytes
`half3, short3, ushort3`	8 bytes
`half3x3, half4x3`	8 bytes
`structures`	4 bytes

Demo

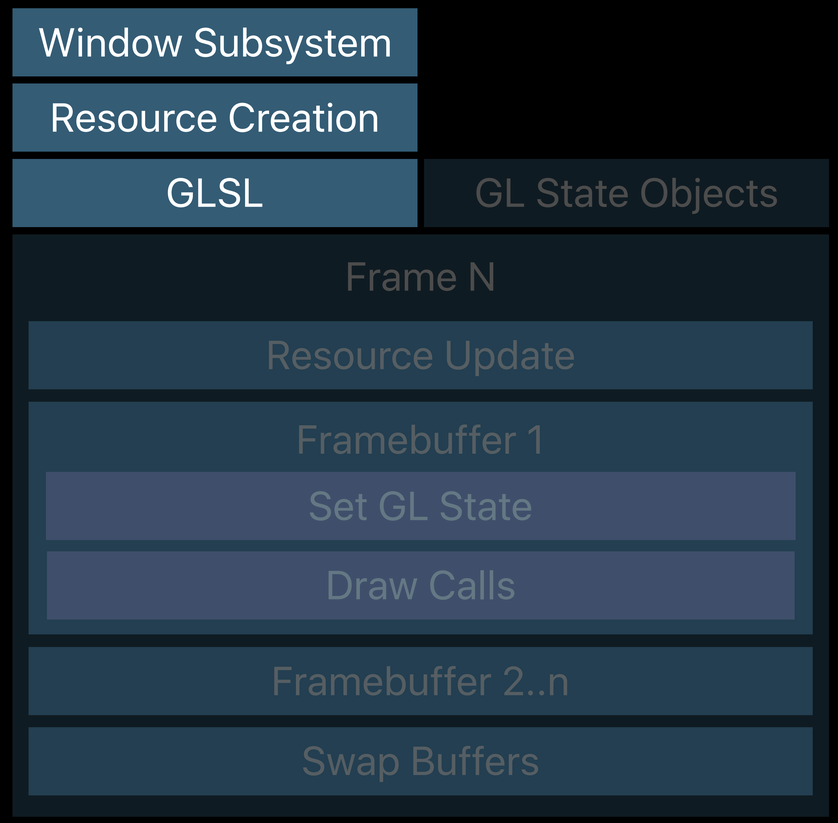
Shader development

Max Christ, GPU Software Tools Engineer

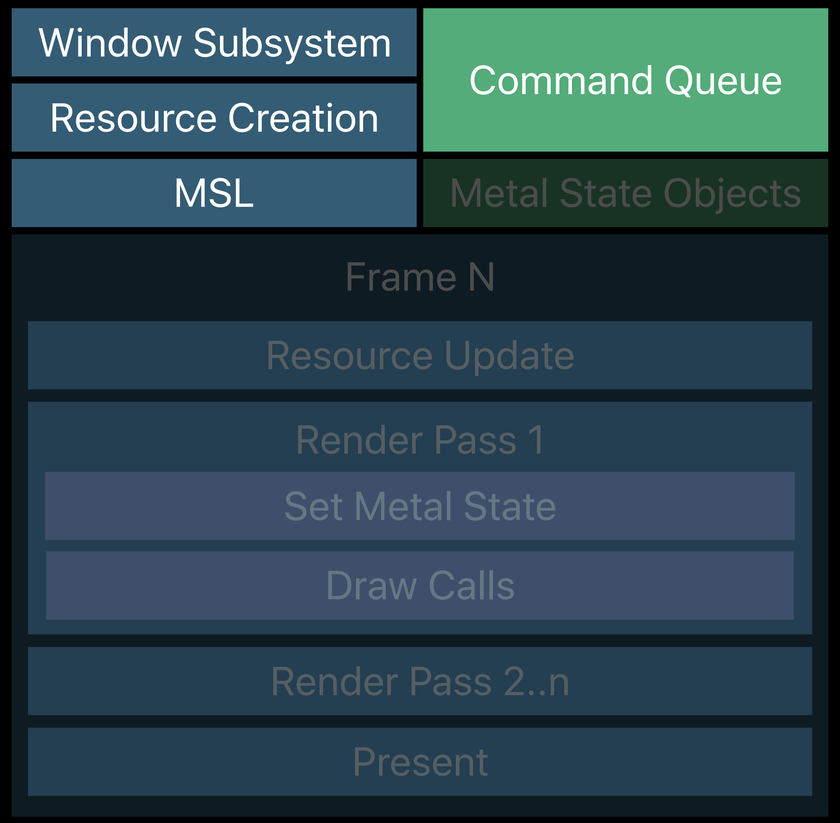
Complete the Port

Sarah Clawson, GPU Software Driver Engineer

Life of a Graphics App



OpenGL

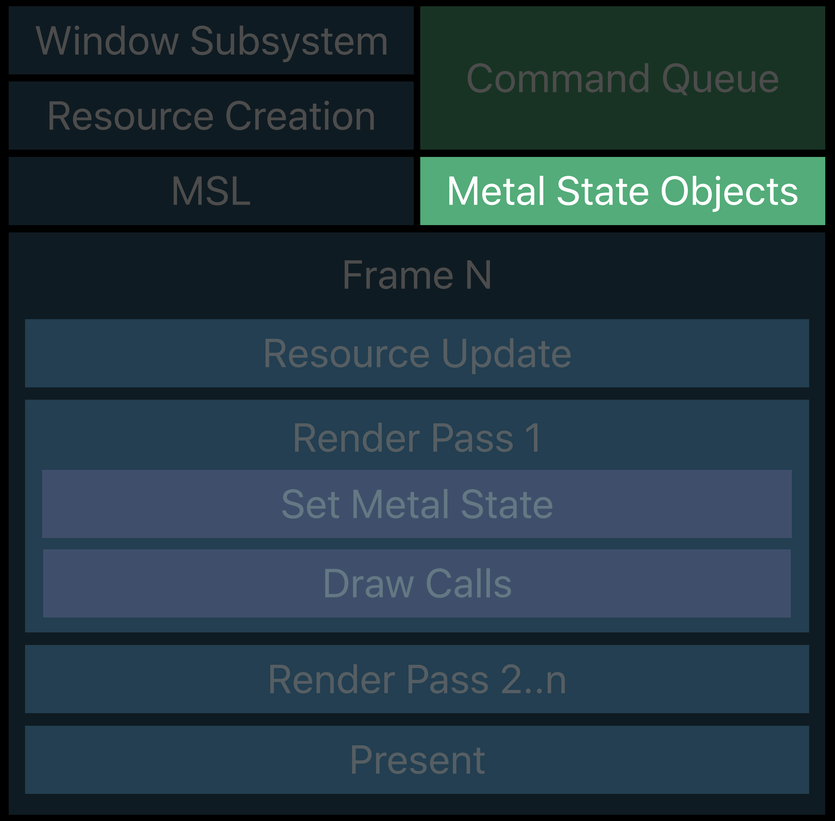


Metal

Life of a Graphics App

State management

OpenGL



Metal

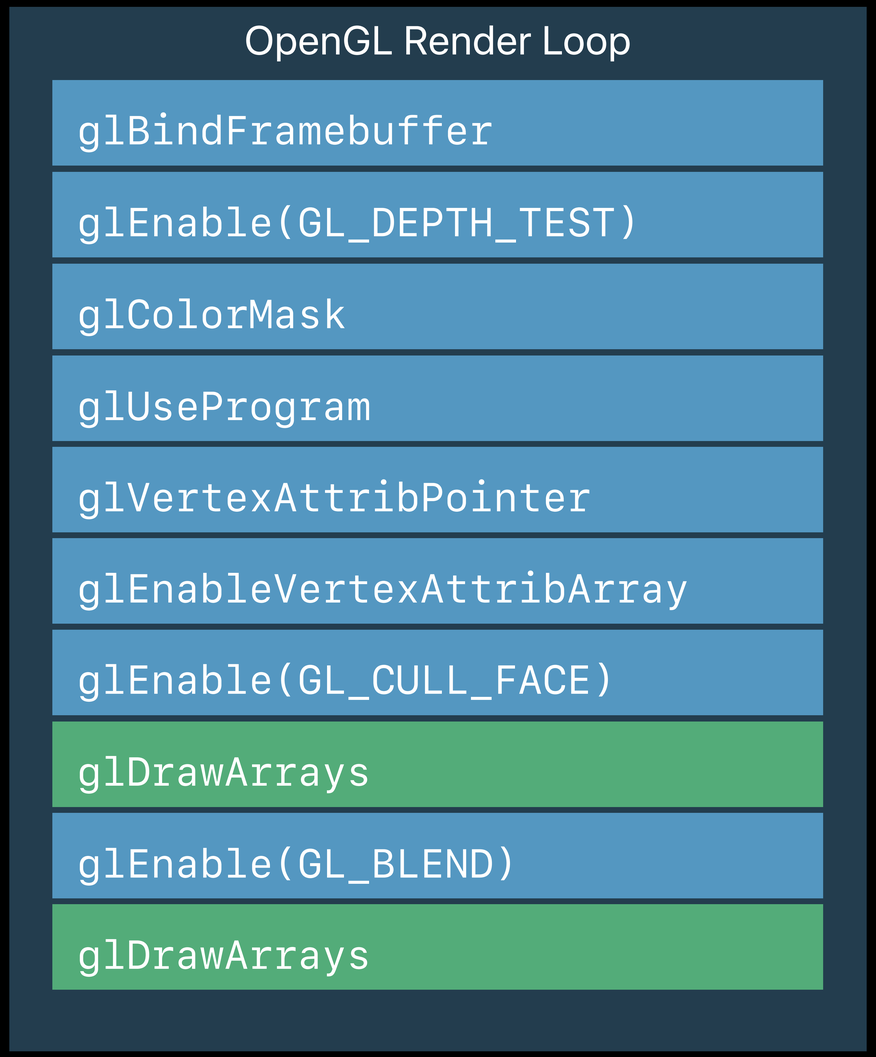
State Objects in OpenGL

Vertex Array Object

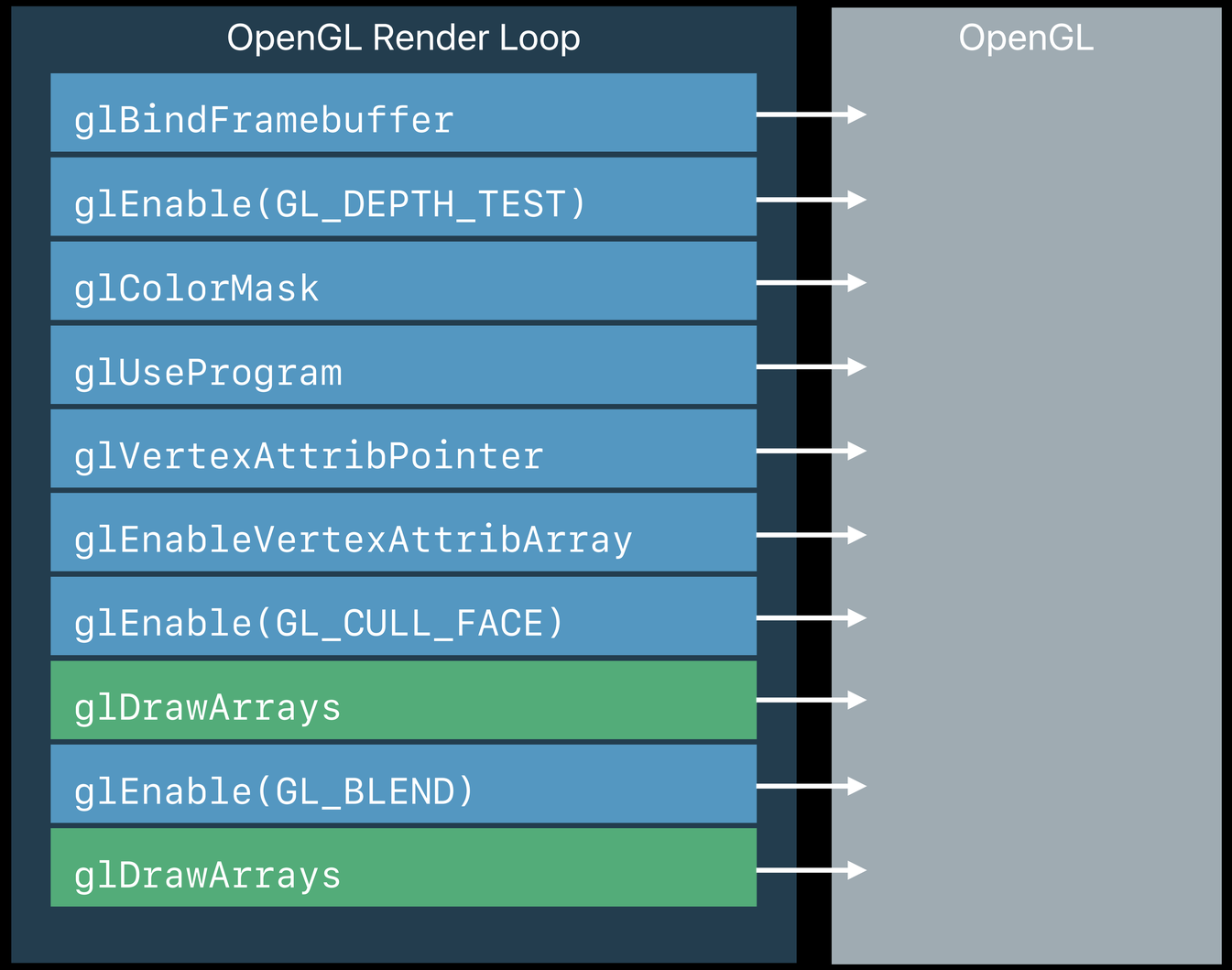
Program

Framebuffer

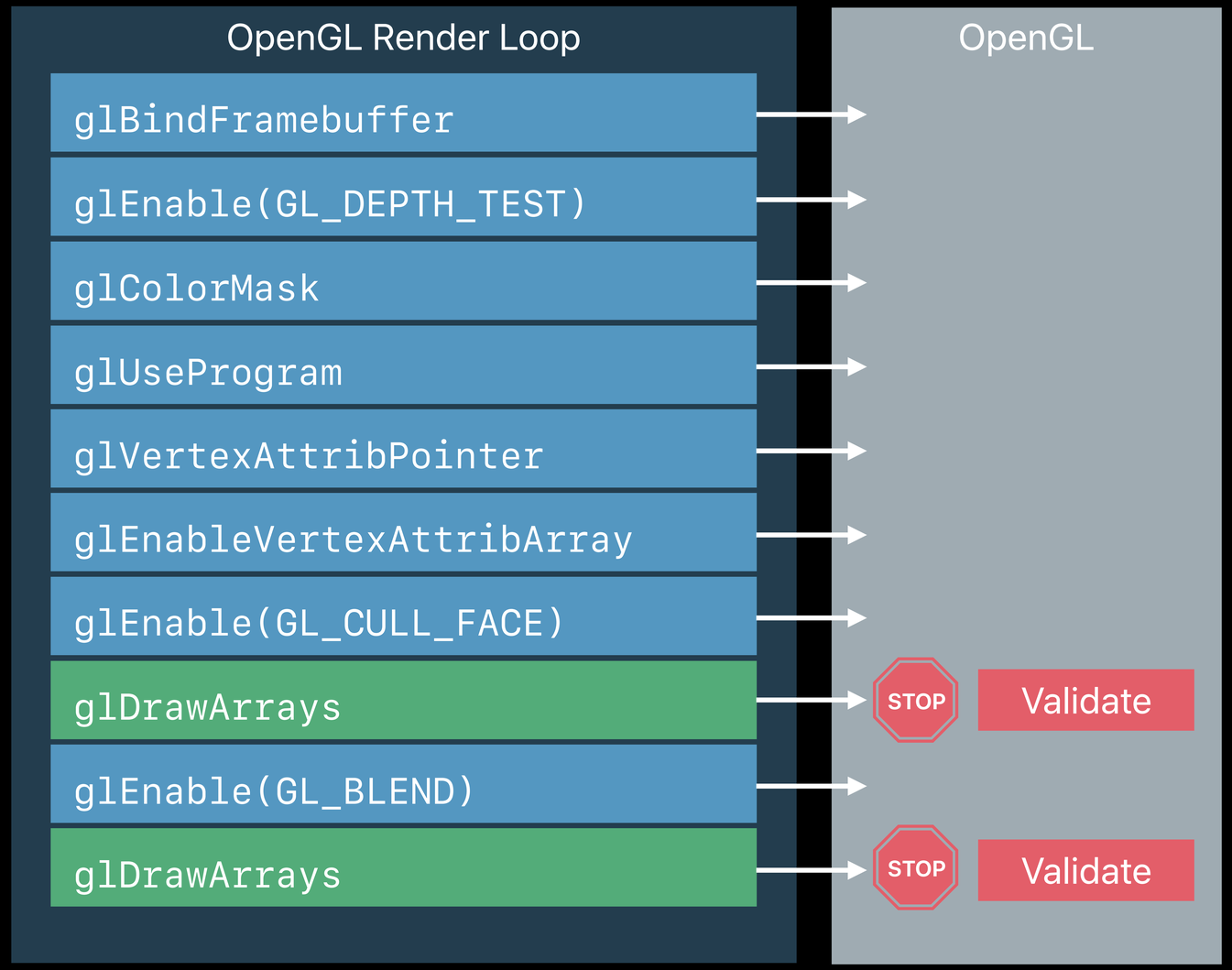
State Management in OpenGL



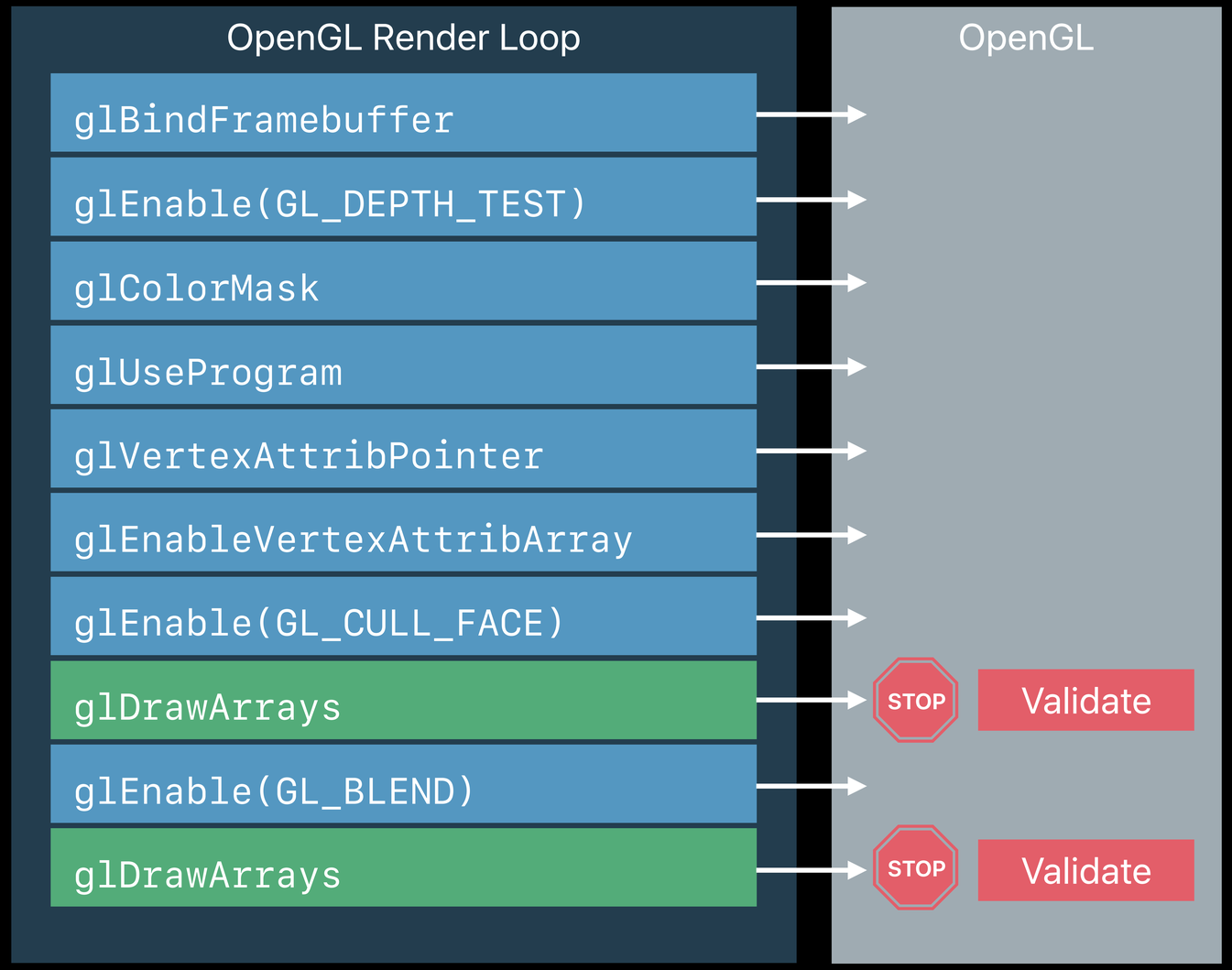
State Management in OpenGL



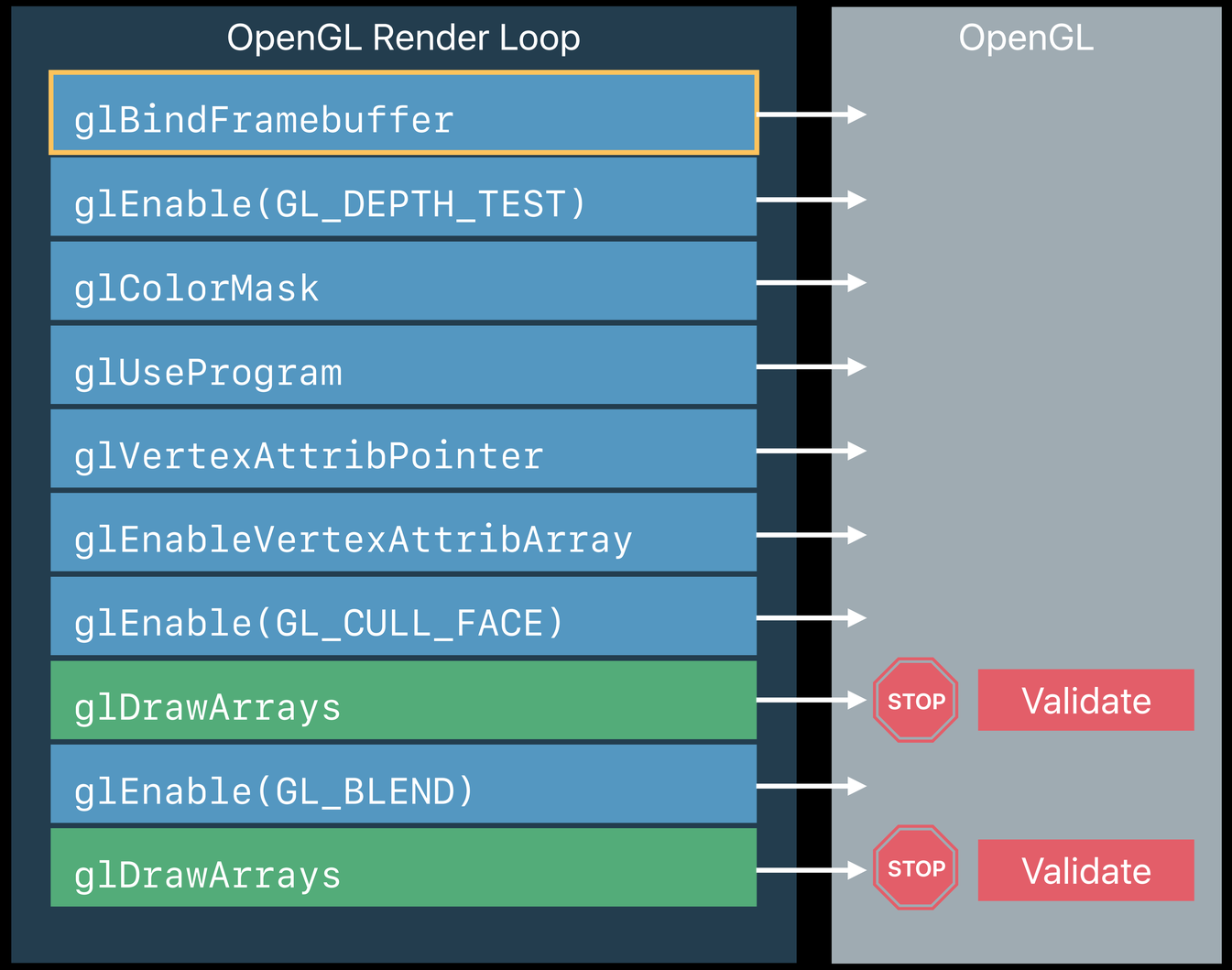
State Management in OpenGL



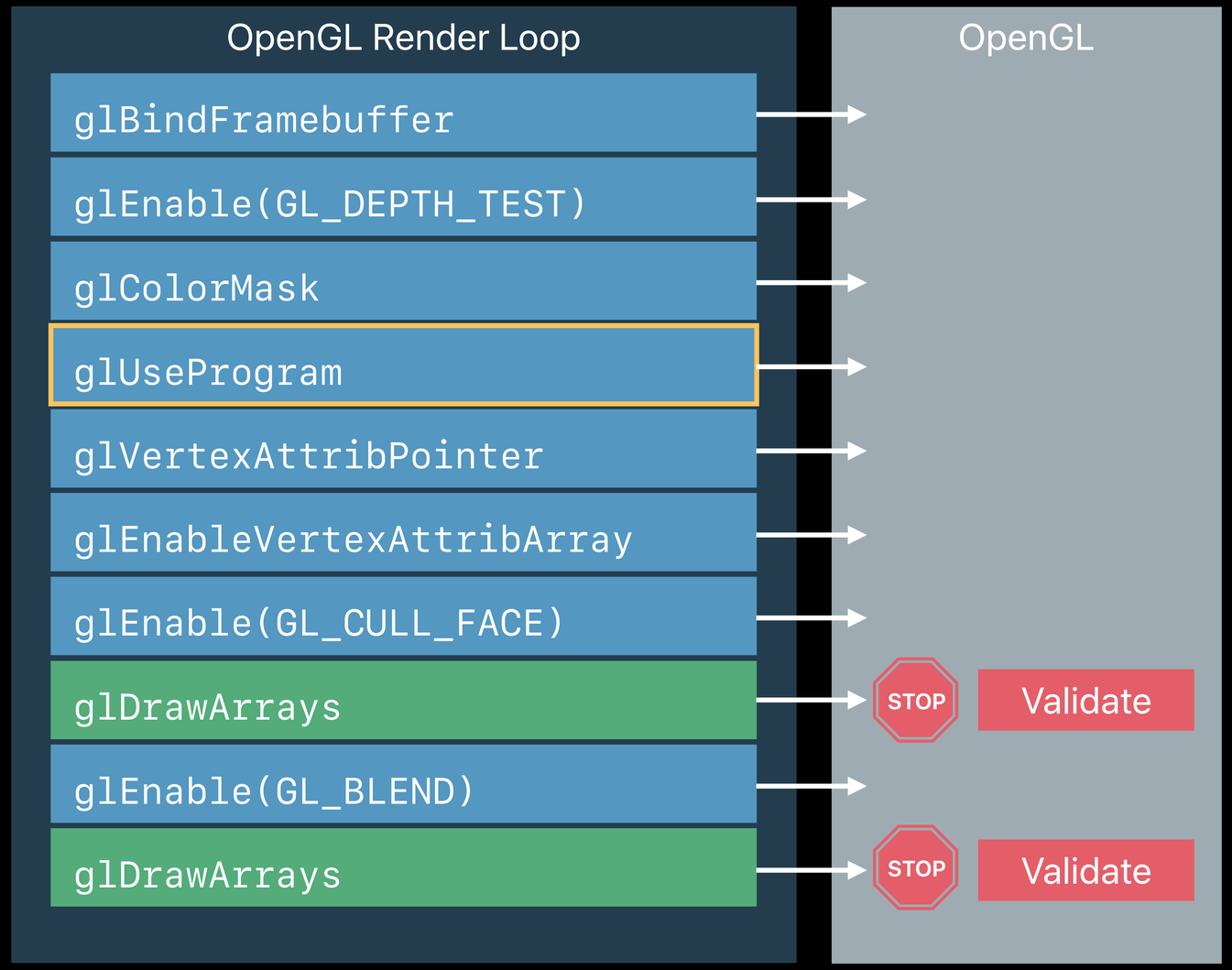
State Management in OpenGL



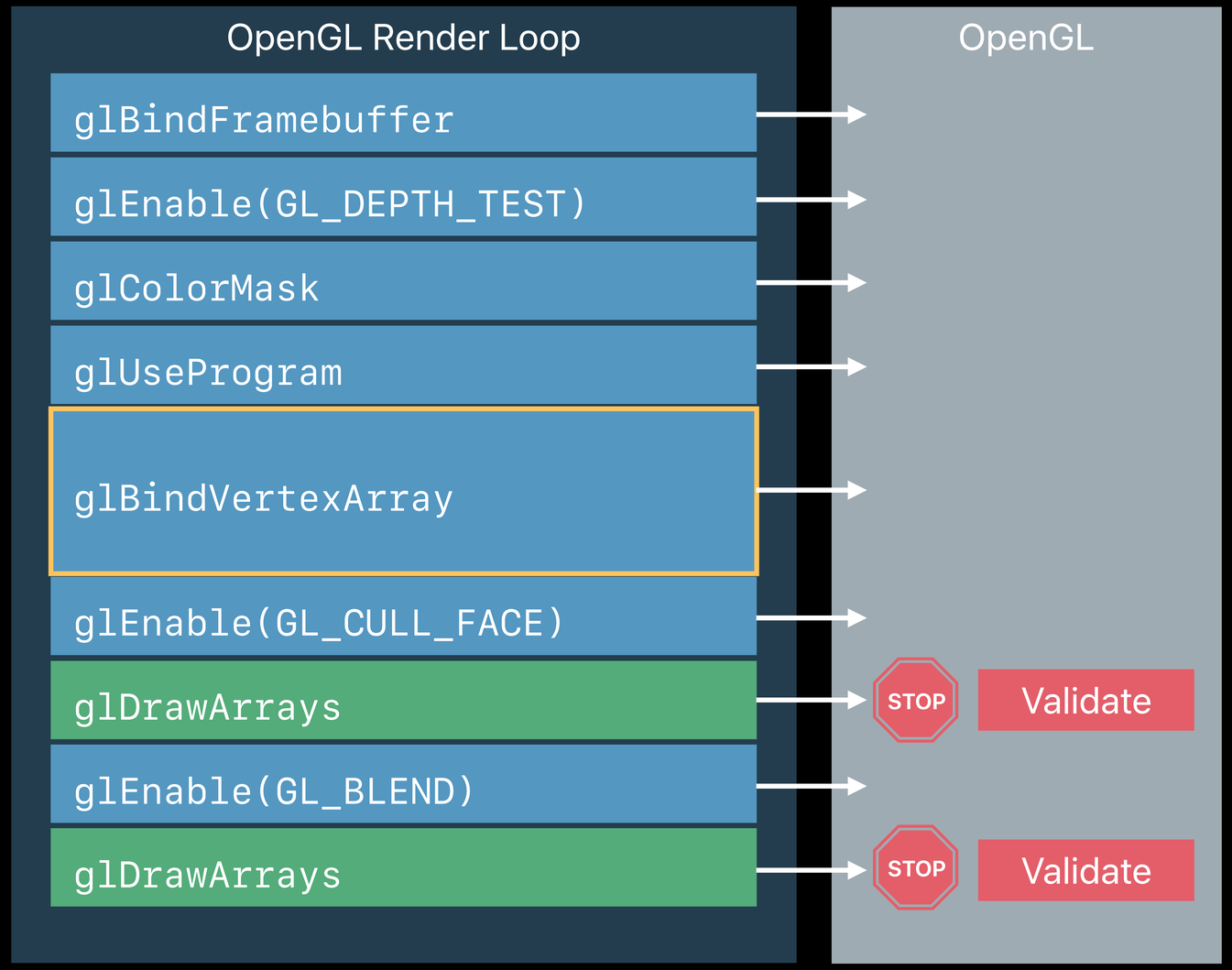
State Management in OpenGL



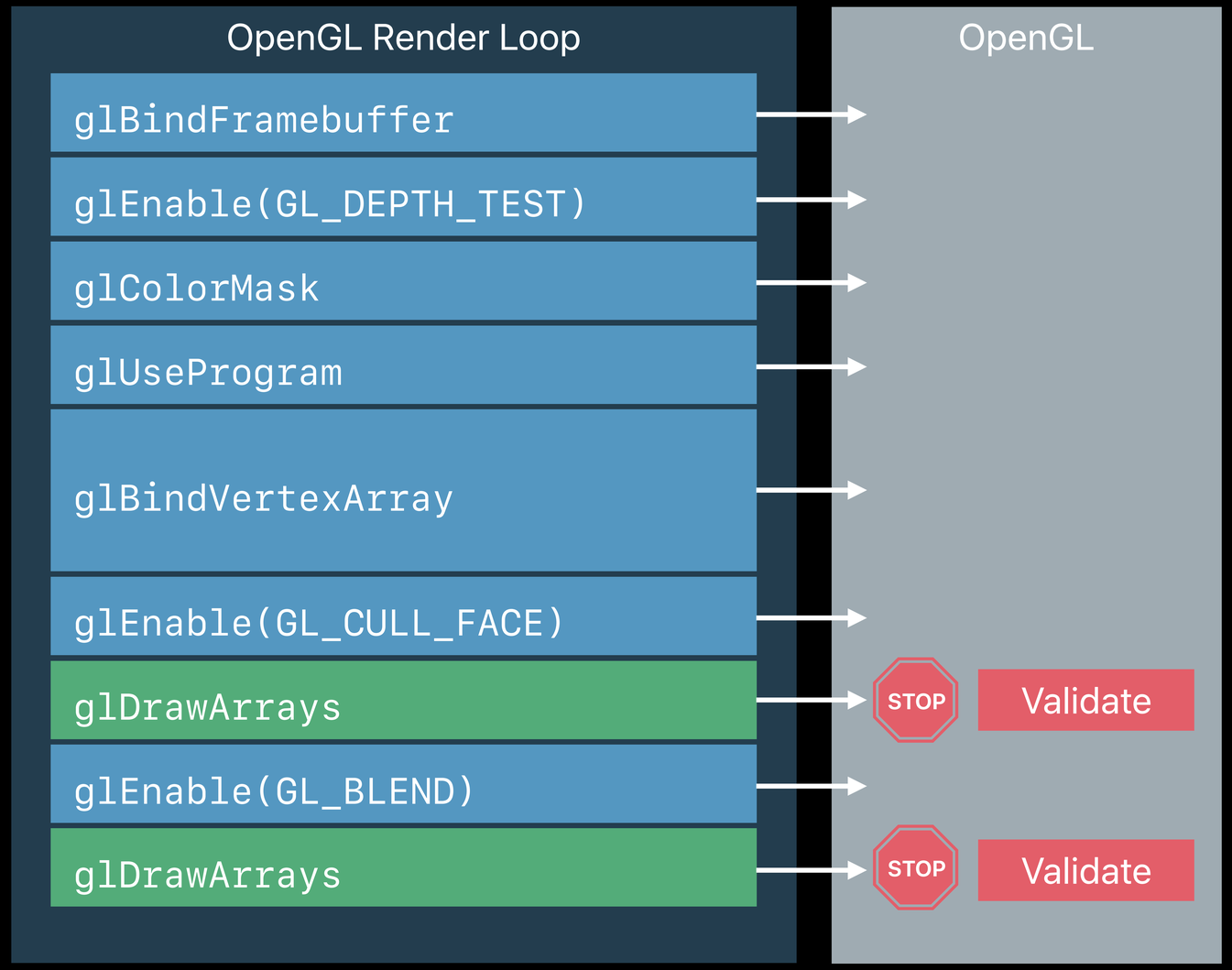
State Management in OpenGL



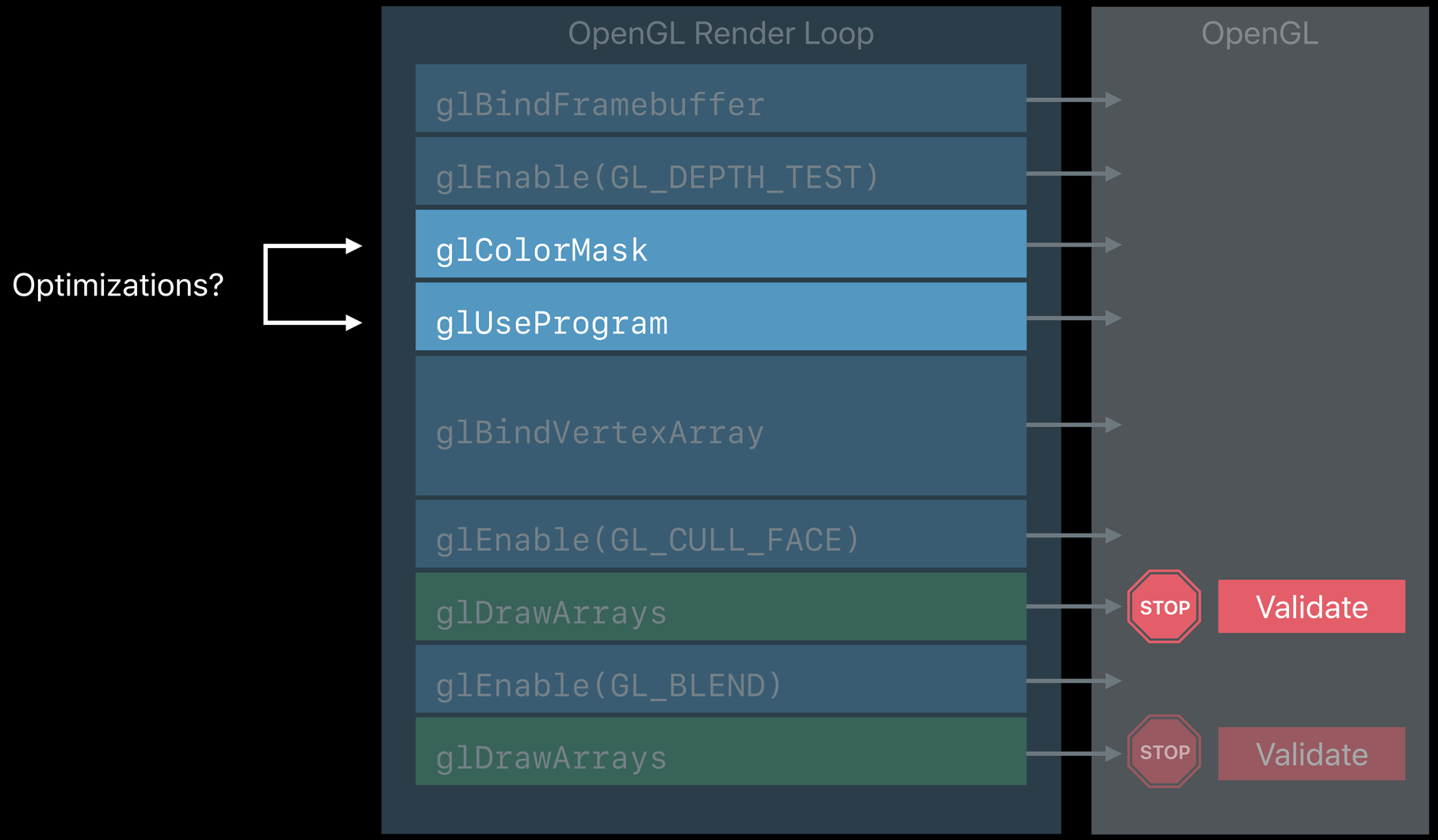
State Management in OpenGL



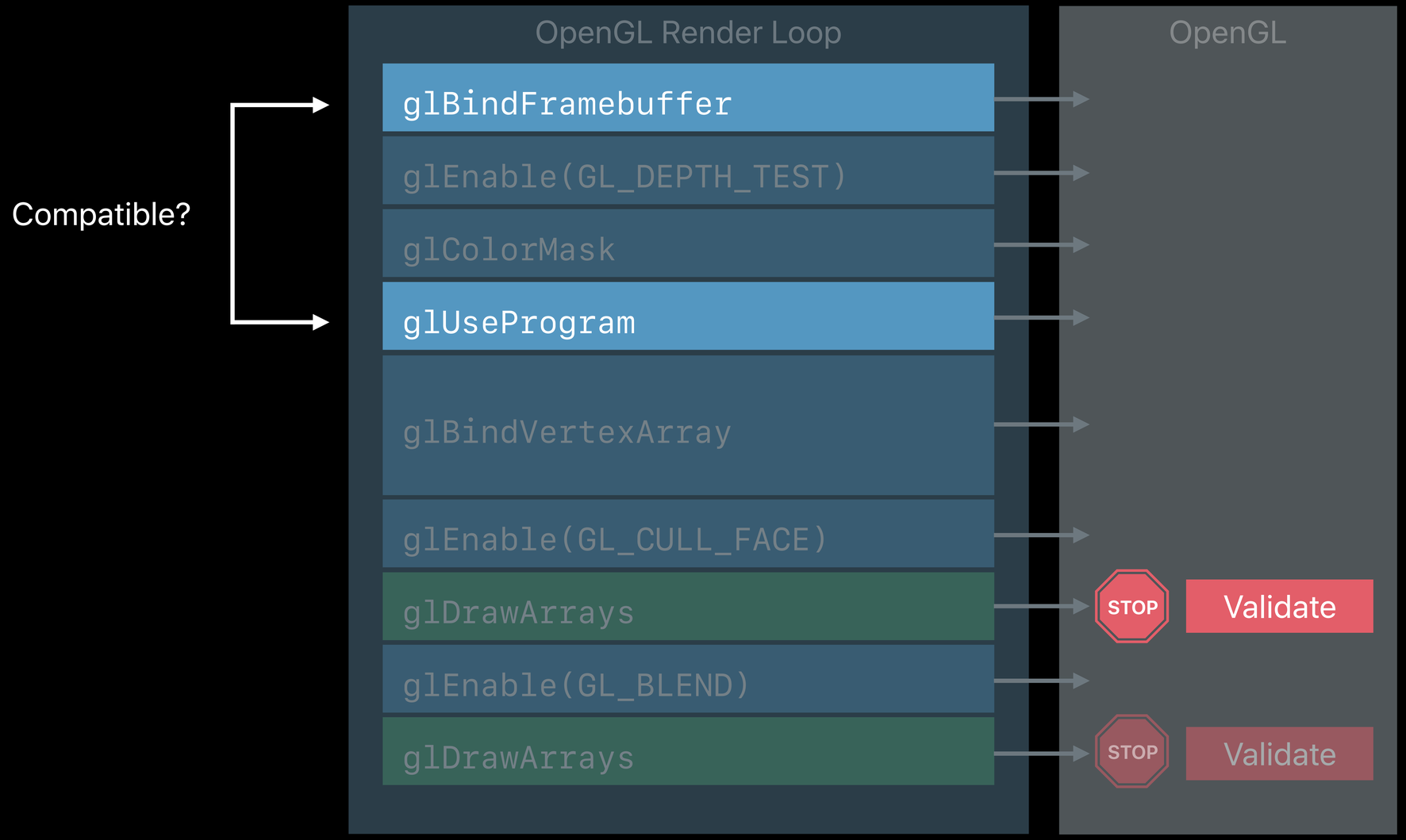
State Management in OpenGL



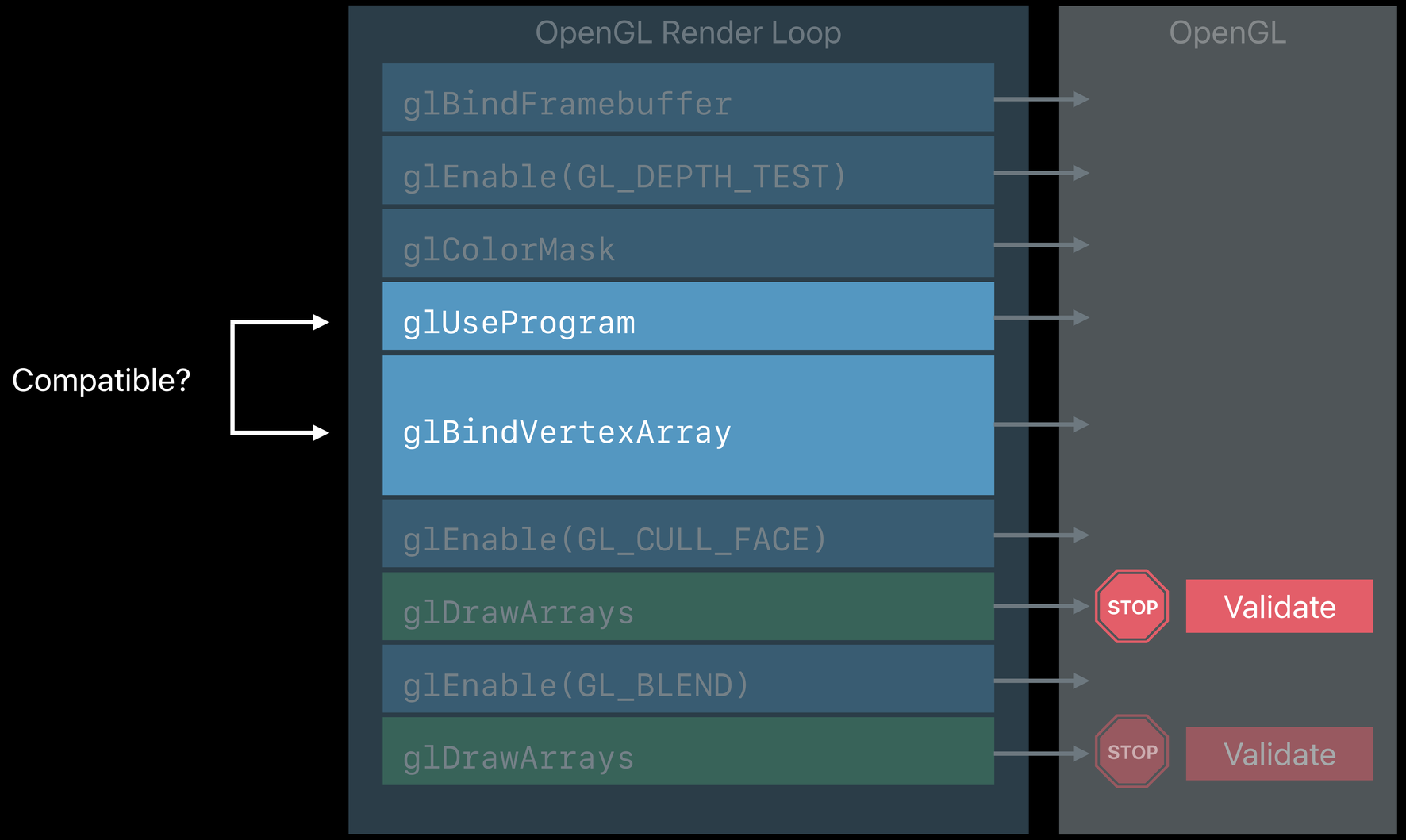
State Management in OpenGL



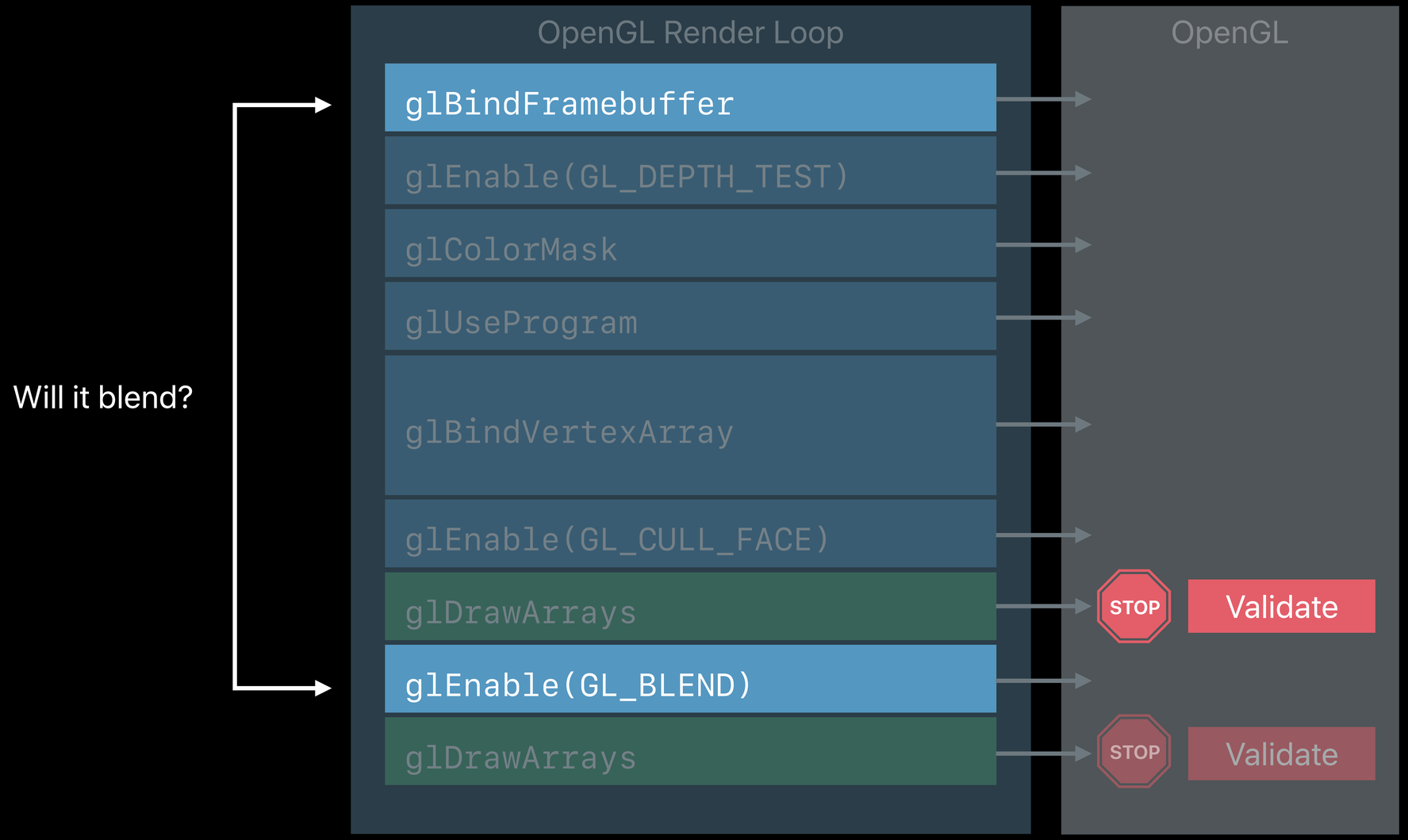
State Management in OpenGL



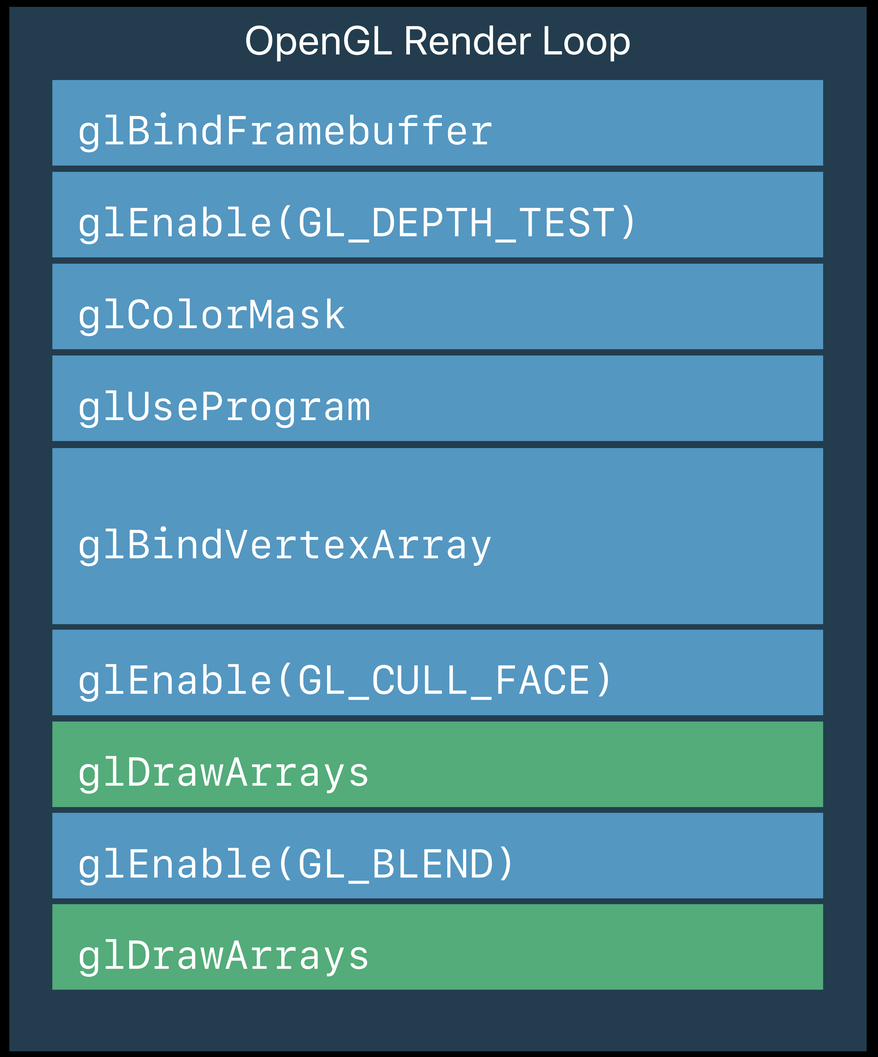
State Management in OpenGL



State Management in OpenGL



Improving Graphics State Management

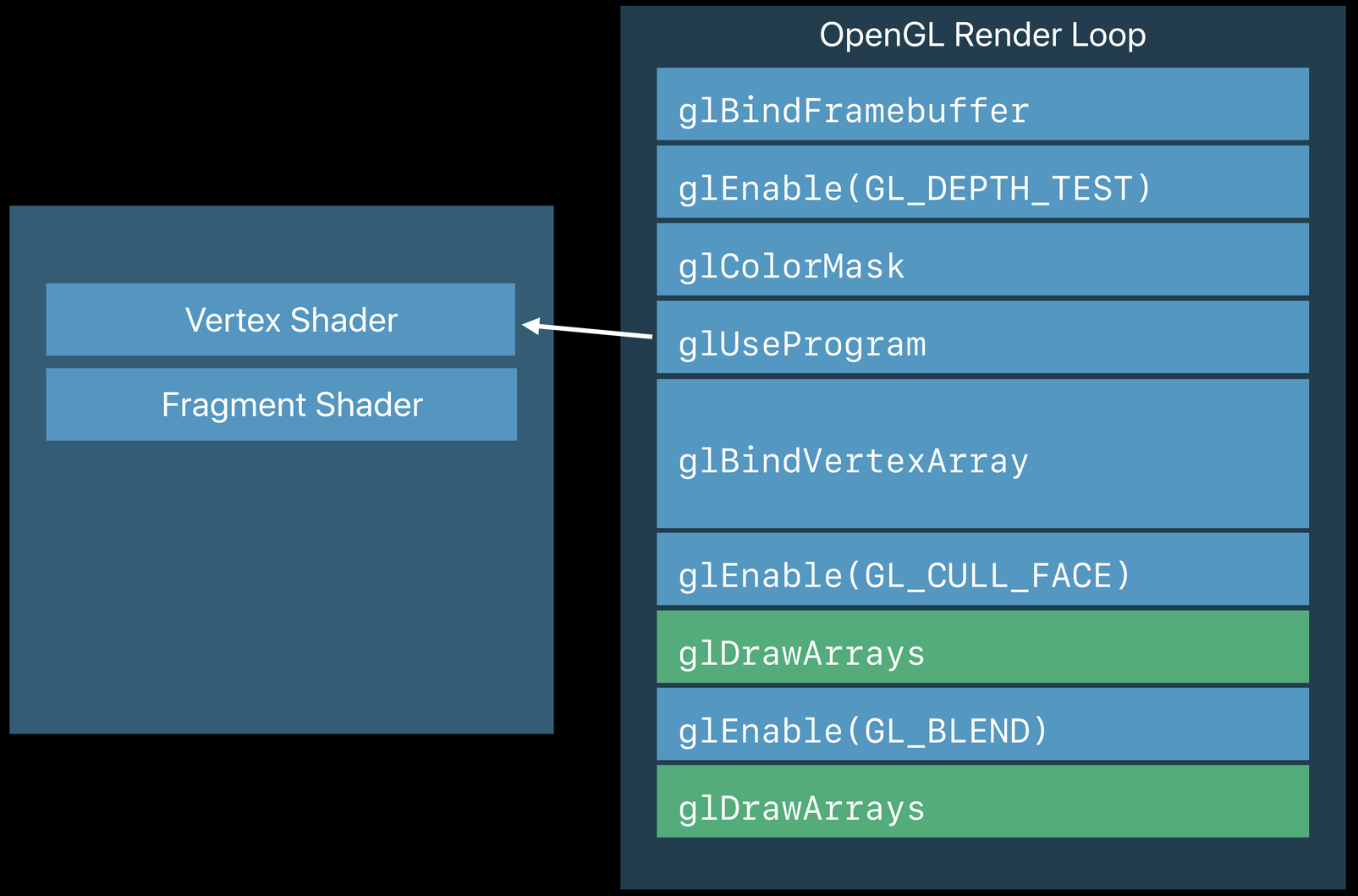


Improving Graphics State Management

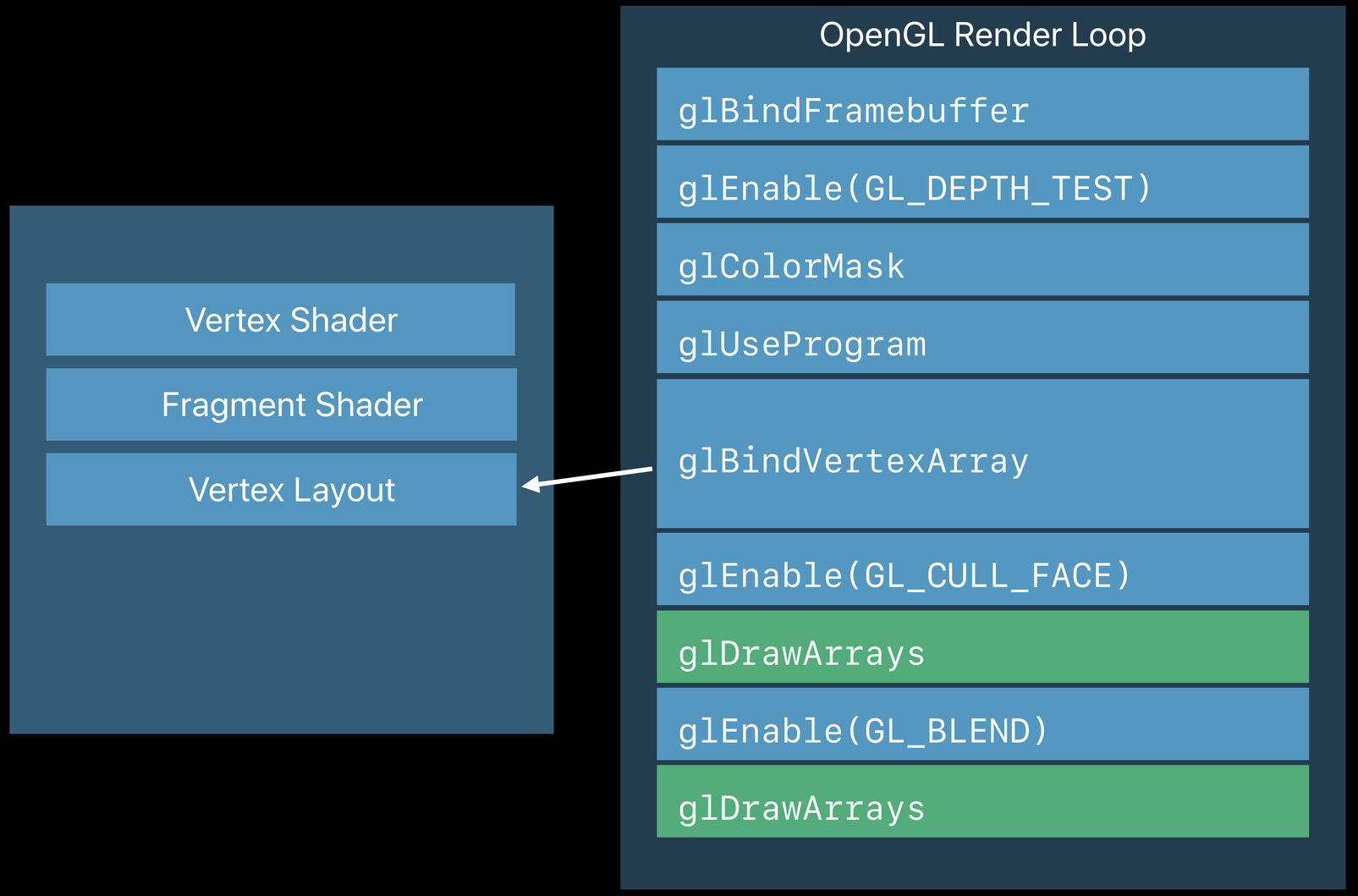
OpenGL Render Loop

```
glBindFramebuffer
glEnable(GL_DEPTH_TEST)
glColorMask
glUseProgram
glBindVertexArray
glEnable(GL_CULL_FACE)
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
```

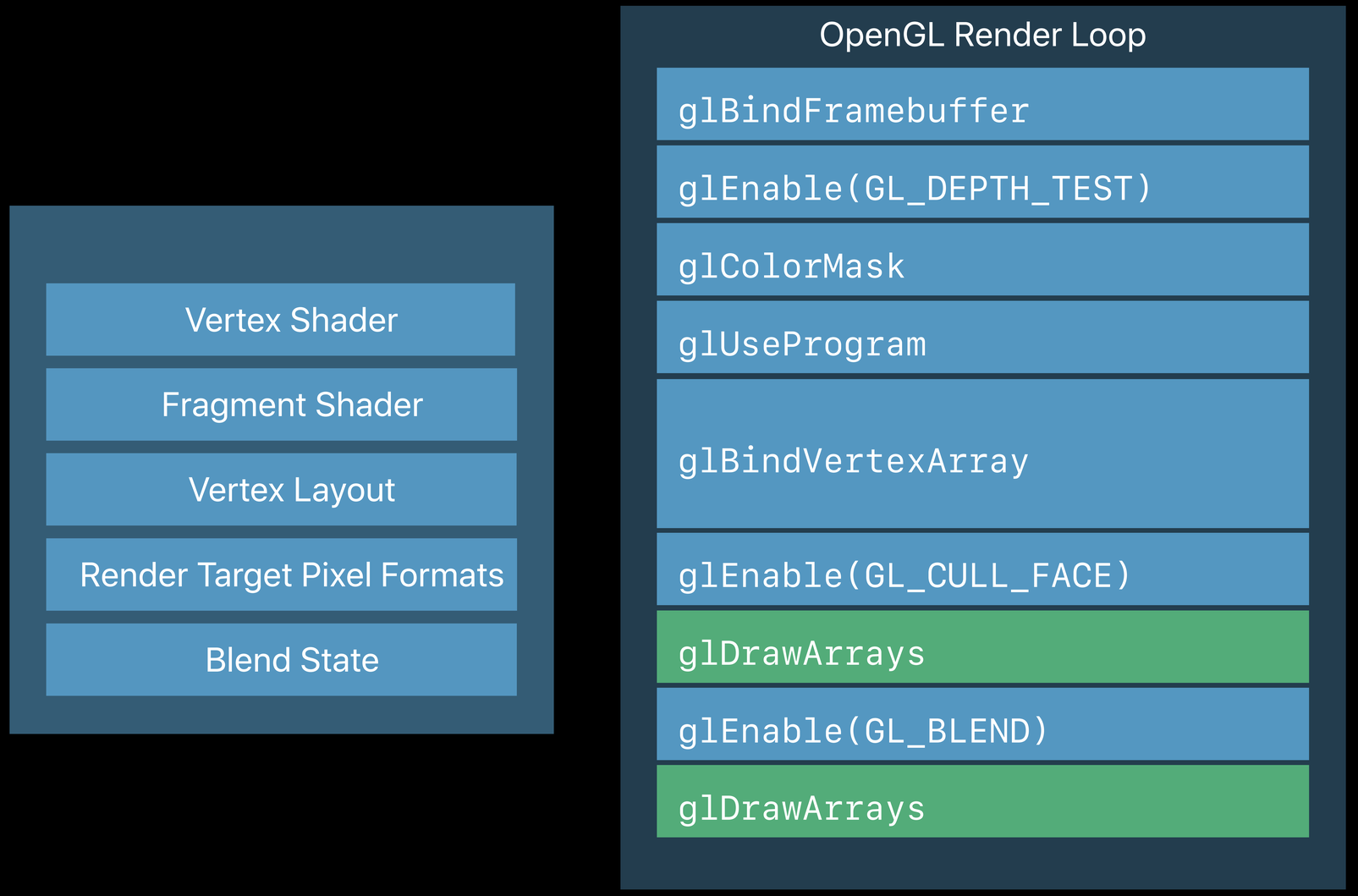
Improving Graphics State Management



Improving Graphics State Management

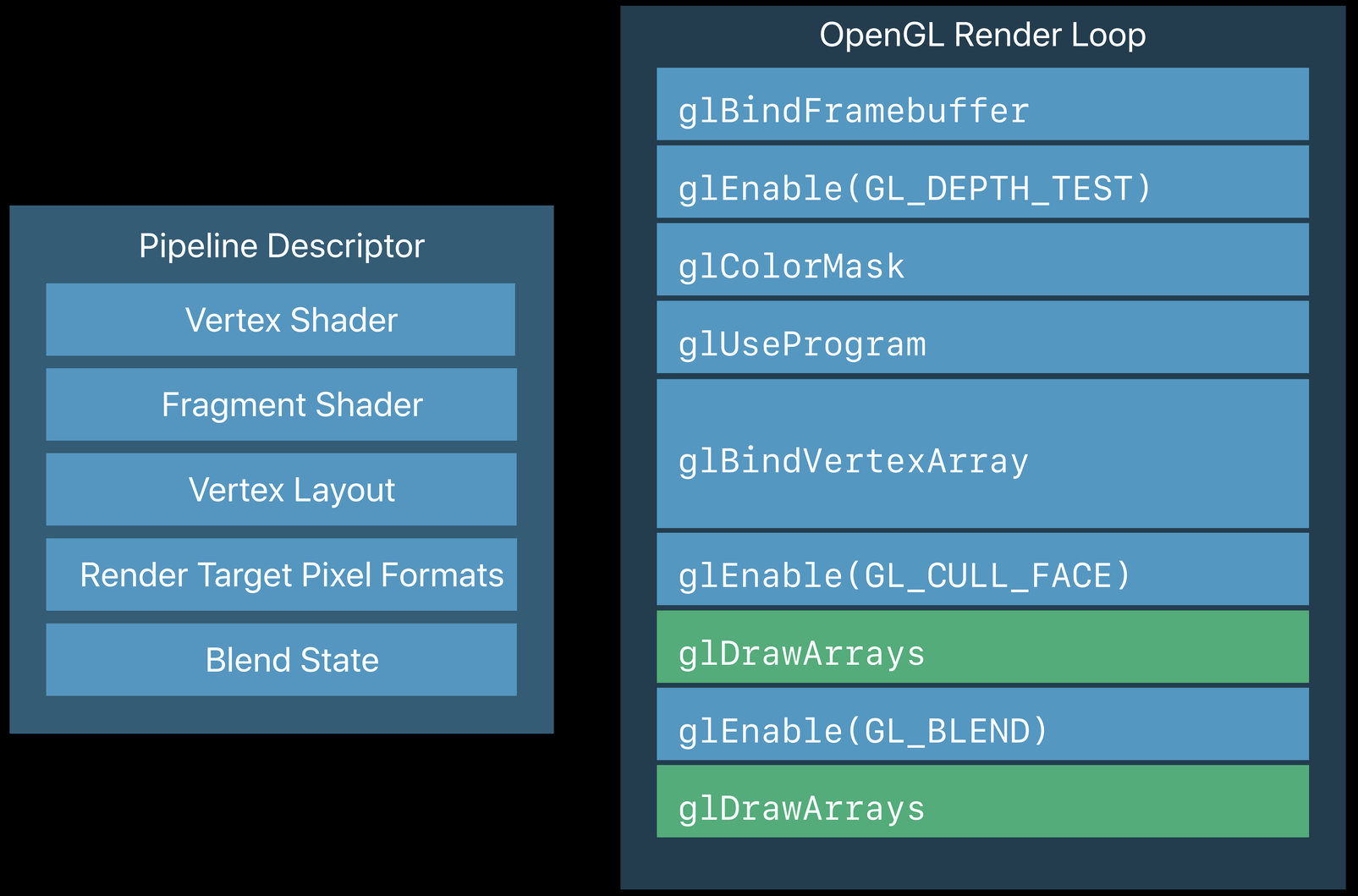


Improving Graphics State Management



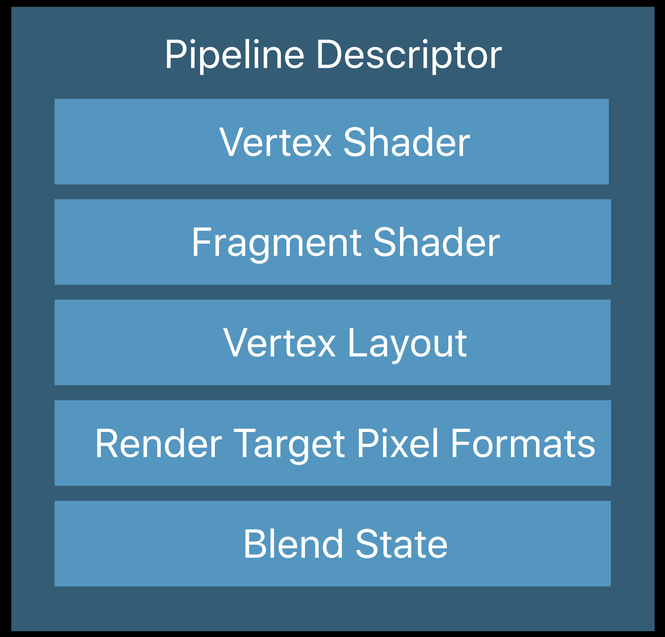
State Management in Metal

Render pipeline descriptor



State Management in Metal

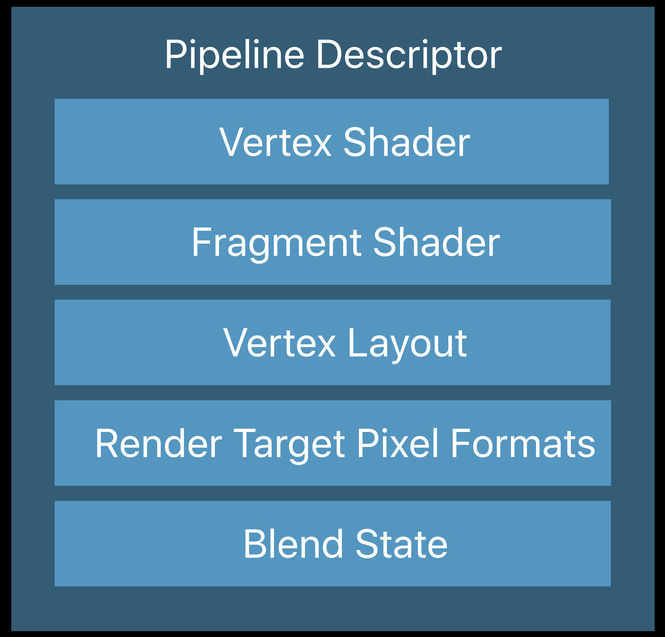
Render pipeline descriptor




```
MTLRenderPipelineDescriptor *psoDesc;
```

State Management in Metal

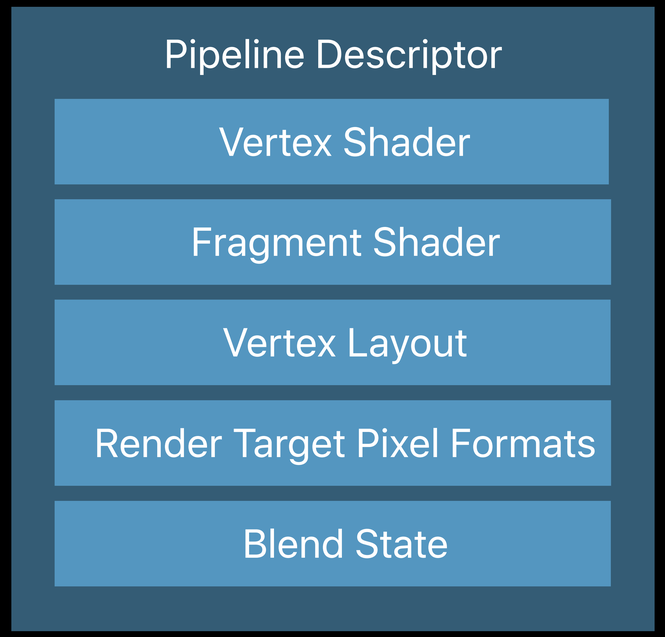
Render pipeline descriptor




```
MTLRenderPipelineDescriptor *psoDesc;
```

State Management in Metal

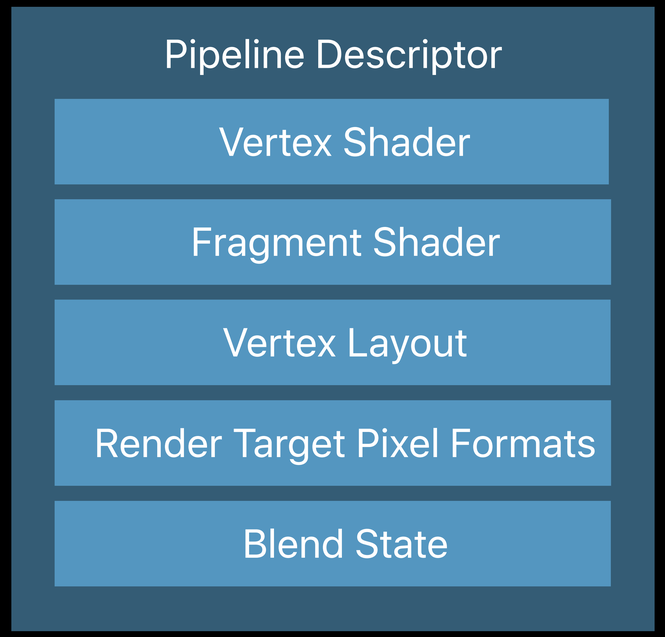
Render pipeline descriptor




```
MTLRenderPipelineDescriptor *psoDesc;

psoDesc = [[MTLRenderPipelineDescriptor alloc] init];
```

State Management in Metal

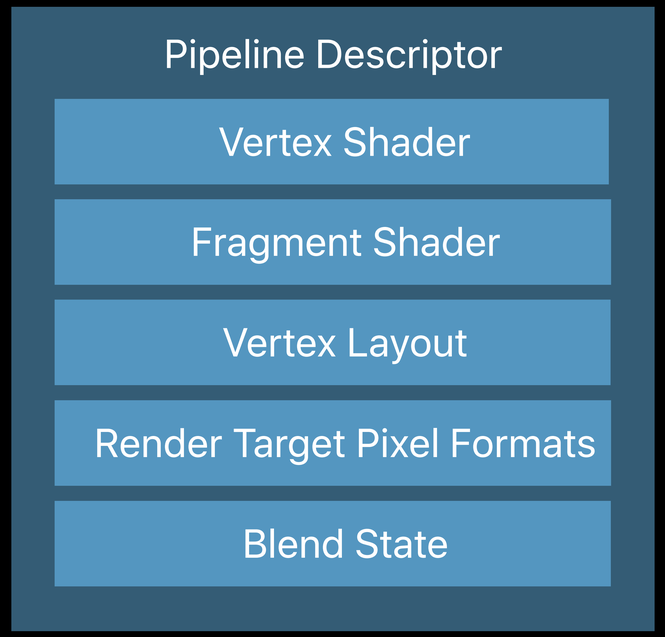
Render pipeline descriptor




```
MTLRenderPipelineDescriptor *psoDesc;

psoDesc = [[MTLRenderPipelineDescriptor alloc] init];

psoDesc.vertexFunction   = vertexFunction;
psoDesc.fragmentFunction = fragmentFunction;
```

State Management in Metal

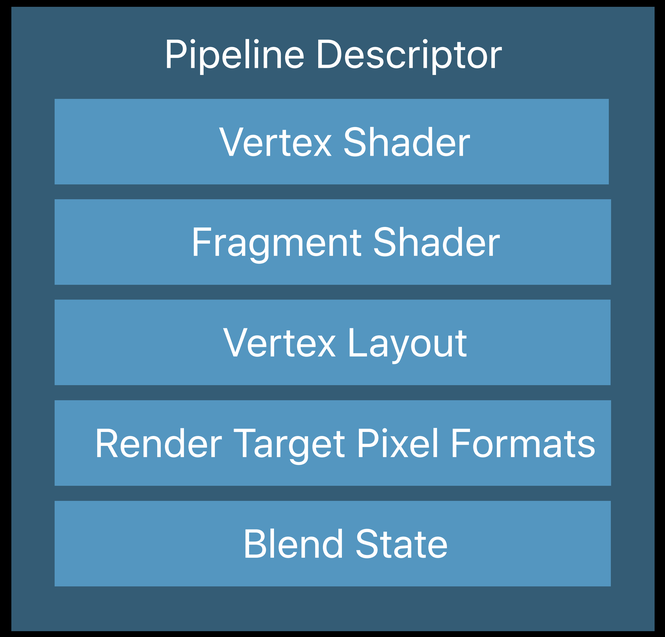
Render pipeline descriptor




```
MTLRenderPipelineDescriptor *psoDesc;

psoDesc = [[MTLRenderPipelineDescriptor alloc] init];

psoDesc.vertexFunction   = vertexFunction;
psoDesc.fragmentFunction = fragmentFunction;
psoDesc.vertexDescriptor = mtlVertexDescriptor;
```

State Management in Metal

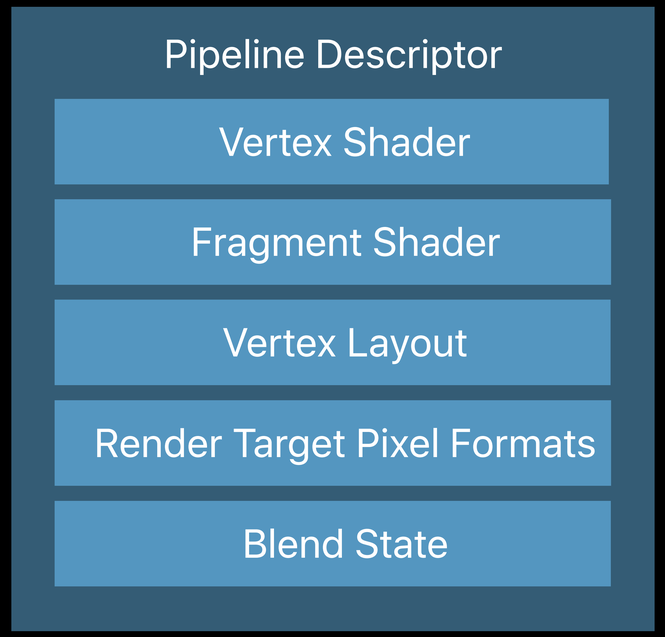
Render pipeline descriptor




```
MTLRenderPipelineDescriptor *psoDesc;

psoDesc = [[MTLRenderPipelineDescriptor alloc] init];

psoDesc.vertexFunction   = vertexFunction;
psoDesc.fragmentFunction = fragmentFunction;
psoDesc.vertexDescriptor = mtlVertexDescriptor;

psoDesc.sampleCount = view.sampleCount;
psoDesc.colorAttachments[0].pixelFormat = view.colorPixelFormat;
psoDesc.depthAttachmentPixelFormat = view.depthStencilPixelFormat;
psoDesc.stencilAttachmentPixelFormat = view.depthStencilPixelFormat;
```

State Management in Metal

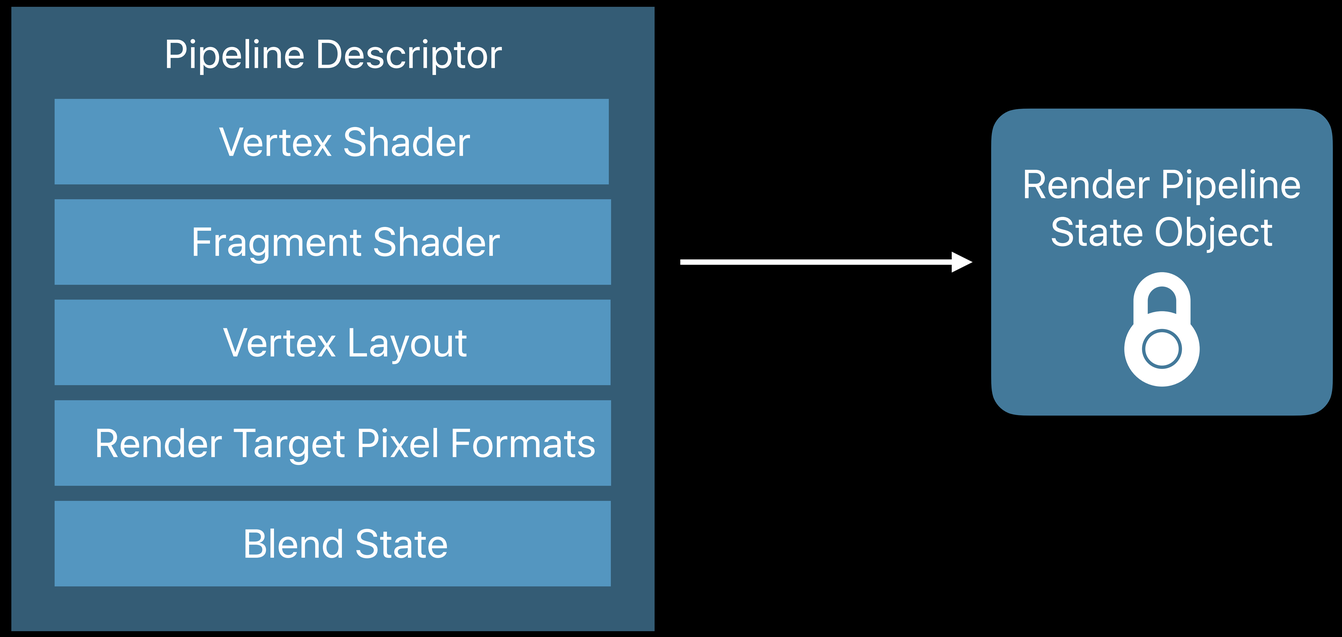
Render pipeline state object




```
MTLRenderPipelineDescriptor *psoDesc;

psoDesc = [[MTLRenderPipelineDescriptor alloc] init];

psoDesc.vertexFunction   = vertexFunction;
psoDesc.fragmentFunction = fragmentFunction;
psoDesc.vertexDescriptor = mtlVertexDescriptor;

psoDesc.sampleCount = view.sampleCount;
psoDesc.colorAttachments[0].pixelFormat = view.colorPixelFormat;
psoDesc.depthAttachmentPixelFormat = view.depthStencilPixelFormat;
psoDesc.stencilAttachmentPixelFormat = view.depthStencilPixelFormat;
```

State Management in Metal

Render pipeline state object




```
id<MTLRenderPipelineState> pipelineState;
pipelineState = [device newRenderPipelineStateWithDescriptor:psoDesc error:&error];
```

State Management in Metal

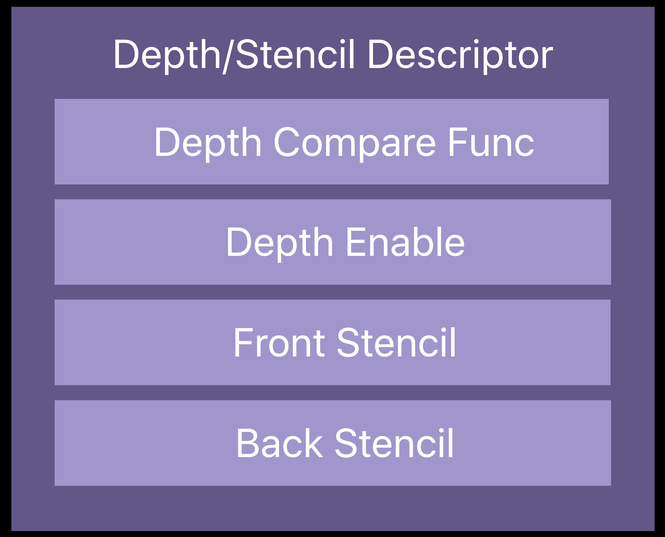
Depth/Stencil descriptor

Depth/Stencil Descriptor

```
MTLDepthStencilDescriptor *dssDesc;

dssDesc = [[MTLDepthStencilDescriptor alloc] init];
```

State Management in Metal

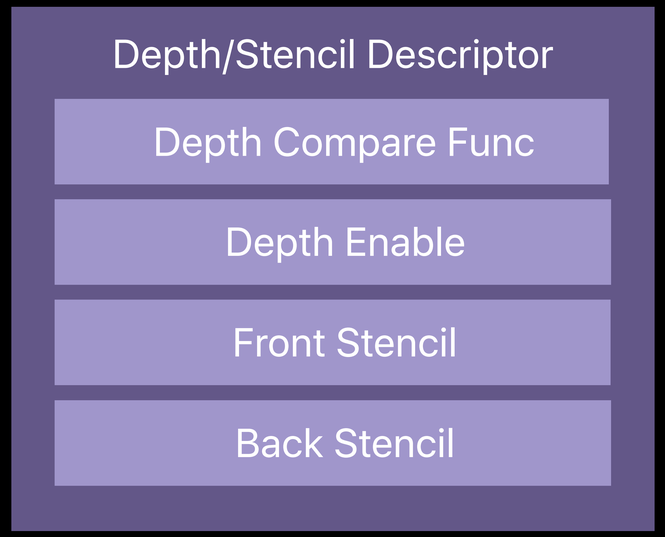
Depth/Stencil descriptor




```
MTLDepthStencilDescriptor *dssDesc;

dssDesc = [[MTLDepthStencilDescriptor alloc] init];

dssDesc.depthCompareFunction = MTLCompareFunctionLess;
dssDesc.depthWriteEnabled    = YES;

dssDesc.frontFaceStencil     = nil;
```

State Management in Metal

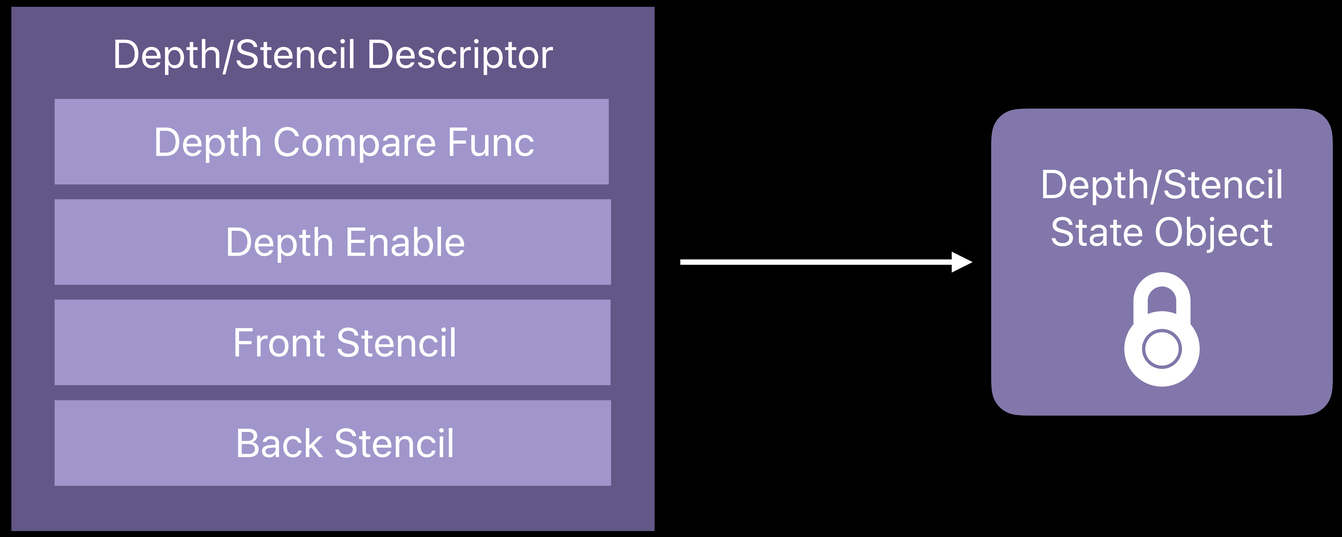
Depth/Stencil state object




```
MTLDepthStencilDescriptor *dssDesc;

dssDesc = [[MTLDepthStencilDescriptor alloc] init];

dssDesc.depthCompareFunction = MTLCompareFunctionLess;
dssDesc.depthWriteEnabled    = YES;

dssDesc.frontFaceStencil     = nil;
```

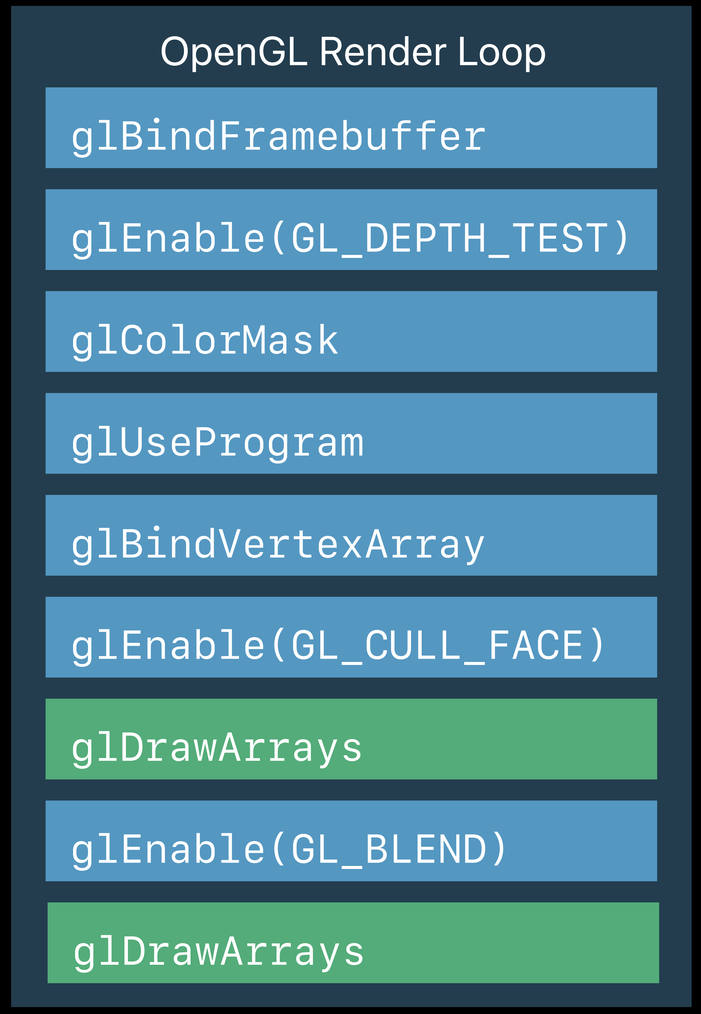
State Management in Metal

Depth/Stencil state object




```
id<MTLDepthStencilState> depthState;
depthState = [device newDepthStencilStateWithDescriptor:dssDesc];
```

State Management in Metal



State Management in Metal

OpenGL Render Loop

```
glBindFramebuffer
glEnable(GL_DEPTH_TEST)
glColorMask
glUseProgram
glBindVertexArray
glEnable(GL_CULL_FACE)
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
```

Metal Render Loop

```
id <MTLRenderCommandEncoder> renderEncoder = [commandBuffer
renderCommandEncoderWithDescriptor: renderPassDescriptor];

[renderEncoder setDepthStencilState:depthState];

[renderEncoder setRenderPipelineState:pipelineState_1];

[renderEncoder setCullMode:MTLCullModeBack];

[renderEncoder drawIndexedPrimitives]:

[renderEncoder setRenderPipelineState:pipelineState_2];

[renderEncoder drawIndexedPrimitives];
```

State Management in Metal

OpenGL Render Loop

```
glBindFramebuffer
glEnable(GL_DEPTH_TEST)
glColorMask
glUseProgram
glBindVertexArray
glEnable(GL_CULL_FACE)
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
```

Metal Render Loop

```
id <MTLRenderCommandEncoder> renderEncoder = [commandBuffer
renderCommandEncoderWithDescriptor: renderPassDescriptor];

[renderEncoder setDepthStencilState:depthState];

[renderEncoder setRenderPipelineState:pipelineState_1];

[renderEncoder setCullMode:MTLCullModeBack];

[renderEncoder drawIndexedPrimitives]:

[renderEncoder setRenderPipelineState:pipelineState_2];

[renderEncoder drawIndexedPrimitives];
```

State Management in Metal

OpenGL Render Loop

```
glBindFramebuffer
glEnable(GL_DEPTH_TEST)
glColorMask
glUseProgram
glBindVertexArray
glEnable(GL_CULL_FACE)
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
```

Metal Render Loop

```
id <MTLRenderCommandEncoder> renderEncoder = [commandBuffer
renderCommandEncoderWithDescriptor: renderPassDescriptor];

[renderEncoder setDepthStencilState:depthState];

[renderEncoder setRenderPipelineState:pipelineState_1];

[renderEncoder setCullMode:MTLCullModeBack];

[renderEncoder drawIndexedPrimitives]:

[renderEncoder setRenderPipelineState:pipelineState_2];

[renderEncoder drawIndexedPrimitives];
```

State Management in Metal

OpenGL Render Loop

```
glBindFramebuffer
glEnable(GL_DEPTH_TEST)
glColorMask
glUseProgram
glBindVertexArray
glEnable(GL_CULL_FACE)
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
```

Metal Render Loop

```
id <MTLRenderCommandEncoder> renderEncoder = [commandBuffer
renderCommandEncoderWithDescriptor: renderPassDescriptor];

[renderEncoder setDepthStencilState:depthState];

[renderEncoder setRenderPipelineState:pipelineState_1];

[renderEncoder setCullMode:MTLCullModeBack];

[renderEncoder drawIndexedPrimitives]:

[renderEncoder setRenderPipelineState:pipelineState_2];

[renderEncoder drawIndexedPrimitives];
```

State Management in Metal

OpenGL Render Loop

```
glBindFramebuffer
glEnable(GL_DEPTH_TEST)
glColorMask
glUseProgram
glBindVertexArray
glEnable(GL_CULL_FACE)
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
```

Metal Render Loop

```
id <MTLRenderCommandEncoder> renderEncoder = [commandBuffer
renderCommandEncoderWithDescriptor: renderPassDescriptor];

[renderEncoder setDepthStencilState:depthState];

[renderEncoder setRenderPipelineState:pipelineState_1];

[renderEncoder setCullMode:MTLCullModeBack];

[renderEncoder drawIndexedPrimitives]:

[renderEncoder setRenderPipelineState:pipelineState_2];

[renderEncoder drawIndexedPrimitives];
```

Precompiled State versus “Any Time” State

Set on Pipeline State
Vertex and Fragment Function
Alpha Blending
MSAA Sample Count
Pixel Formats for Render Targets
Vertex Descriptor

Precompiled State versus “Any Time” State

Set on Pipeline State	Set While Drawing
Vertex and Fragment Function	Front Face Winding
Alpha Blending	Cull Mode
MSAA Sample Count	Fill Mode
Pixel Formats for Render Targets	Scissor
Vertex Descriptor	Viewport

State Management in Metal

OpenGL Render Loop

```
glBindFramebuffer
glEnable(GL_DEPTH_TEST)
glColorMask
glUseProgram
glBindVertexArray
glEnable(GL_CULL_FACE)
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
```

Metal Render Loop

```
id <MTLRenderCommandEncoder> renderEncoder = [commandBuffer
renderCommandEncoderWithDescriptor: renderPassDescriptor];

[renderEncoder setDepthStencilState:depthState];

[renderEncoder setRenderPipelineState:pipelineState_1];

[renderEncoder setCullMode:MTLCullModeBack];

[renderEncoder drawIndexedPrimitives]:

[renderEncoder setRenderPipelineState:pipelineState_2];

[renderEncoder drawIndexedPrimitives];
```

State Management in Metal

OpenGL Render Loop

```
glBindFramebuffer
glEnable(GL_DEPTH_TEST)
glColorMask
glUseProgram
glBindVertexArray
glEnable(GL_CULL_FACE)
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
```

Metal Render Loop

```
id <MTLRenderCommandEncoder> renderEncoder = [commandBuffer
renderCommandEncoderWithDescriptor: renderPassDescriptor];

[renderEncoder setDepthStencilState:depthState];

[renderEncoder setRenderPipelineState:pipelineState_1];

[renderEncoder setCullMode:MTLCullModeBack];

[renderEncoder drawIndexedPrimitives]:

[renderEncoder setRenderPipelineState:pipelineState_2];

[renderEncoder drawIndexedPrimitives];
```

State Management in Metal

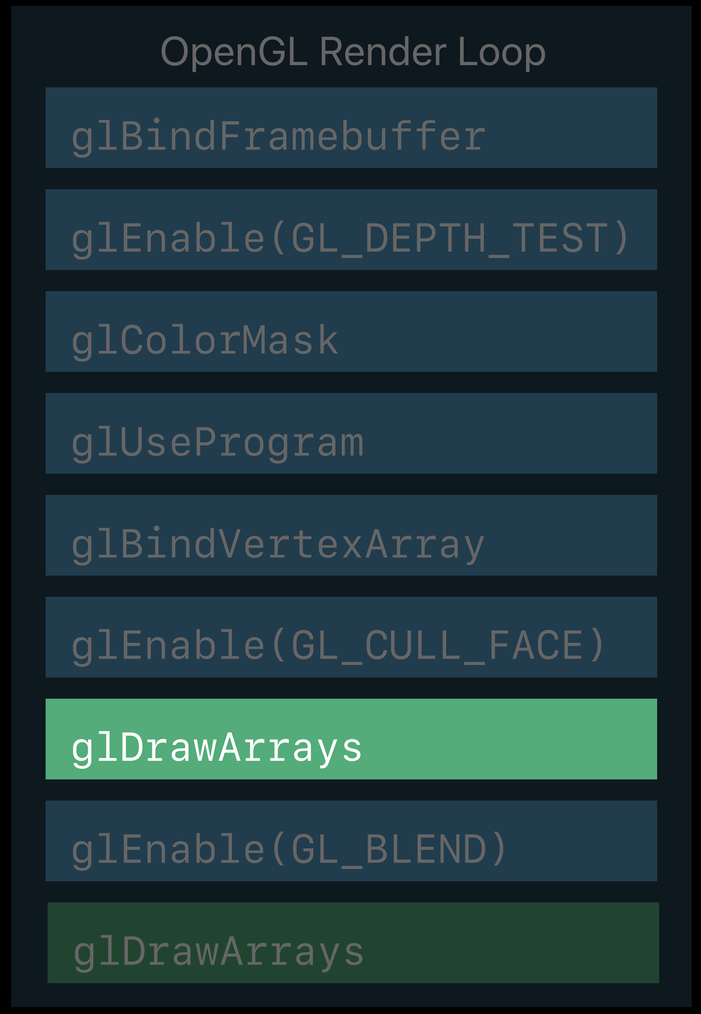
OpenGL Render Loop

```
glBindFramebuffer
glEnable(GL_DEPTH_TEST)
glColorMask
glUseProgram
glBindVertexArray
glEnable(GL_CULL_FACE)
glDrawArrays
glEnable(GL_BLEND)
glDrawArrays
```

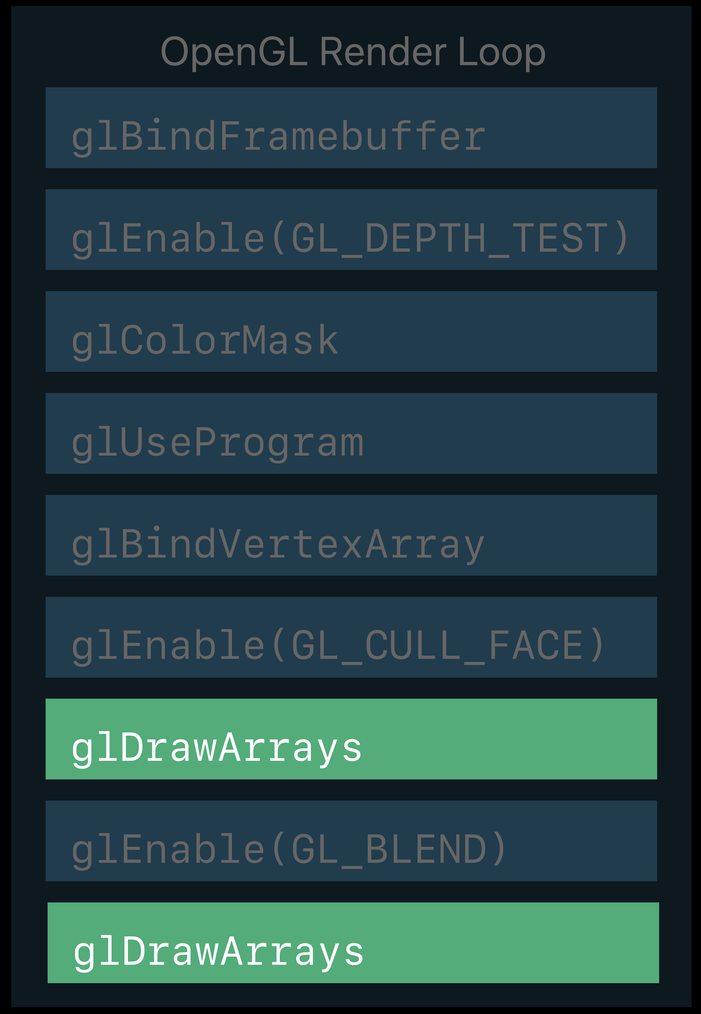
Metal Render Loop

```
id <MTLRenderCommandEncoder> renderEncoder = [commandBuffer
renderCommandEncoderWithDescriptor: renderPassDescriptor];

[renderEncoder setDepthStencilState:depthState];

[renderEncoder setRenderPipelineState:pipelineState_1];

[renderEncoder setCullMode:MTLCullModeBack];

[renderEncoder drawIndexedPrimitives]:

[renderEncoder setRenderPipelineState:pipelineState_2];

[renderEncoder drawIndexedPrimitives];
```

OpenGL Optimization



Shader Pre-Warming



Shader Pre-Warming in Metal

Shader Pre-Warming in Metal

```
psoDesc = [[MTLRenderPipelineDescriptor alloc] init];
psoDesc.vertexFunction   = myVertexFunction;
psoDesc.fragmentFunction = myFragmentFunction;
psoDesc.vertexDescriptor = myVertexDescriptor;
psoDesc.sampleCount = view.sampleCount;
psoDesc.colorAttachments[0].pixelFormat = view.colorPixelFormat;
pso_1 = [device newRenderPipelineStateWithDescriptor:psoDesc error:&error];
```

pso_1

Shader Pre-Warming in Metal

```
psoDesc = [[MTLRenderPipelineDescriptor alloc] init];
psoDesc.vertexFunction   = myVertexFunction;
psoDesc.fragmentFunction = myFragmentFunction;
psoDesc.vertexDescriptor = myVertexDescriptor;
psoDesc.sampleCount = view.sampleCount;
psoDesc.colorAttachments[0].pixelFormat = view.colorPixelFormat;
pso_1 = [device newRenderPipelineStateWithDescriptor:psoDesc error:&error];
```

pso_1

```
psoDesc.colorAttachments[0].blendingEnabled = YES;
// Set all blend state
pso_2 = [device newRenderPipelineStateWithDescriptor:psoDesc error:&error];
```

pso_2

Benefit — Do Expensive Work Less Often

When	Frequency
Build	Once
Load	Rare
Draw	Many times per second

Benefit — Do Expensive Work Less Often

When	Frequency	OpenGL
Build	Once	—
Load	Rare	—
Draw	Many times per second	Shader Compilation State Validation Encode Work for GPU

Benefit — Do Expensive Work Less Often

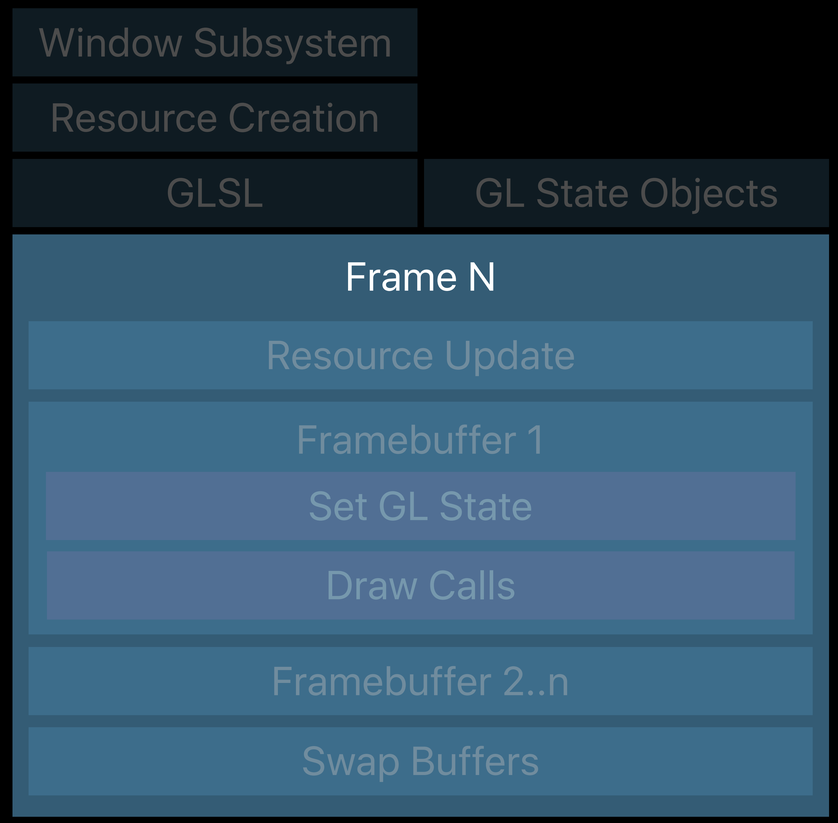
When	Frequency	OpenGL	Metal
Build	Once	—	
Load	Rare	—	
Draw	Many times per second	Shader Compilation State Validation Encode Work for GPU	

Benefit — Do Expensive Work Less Often

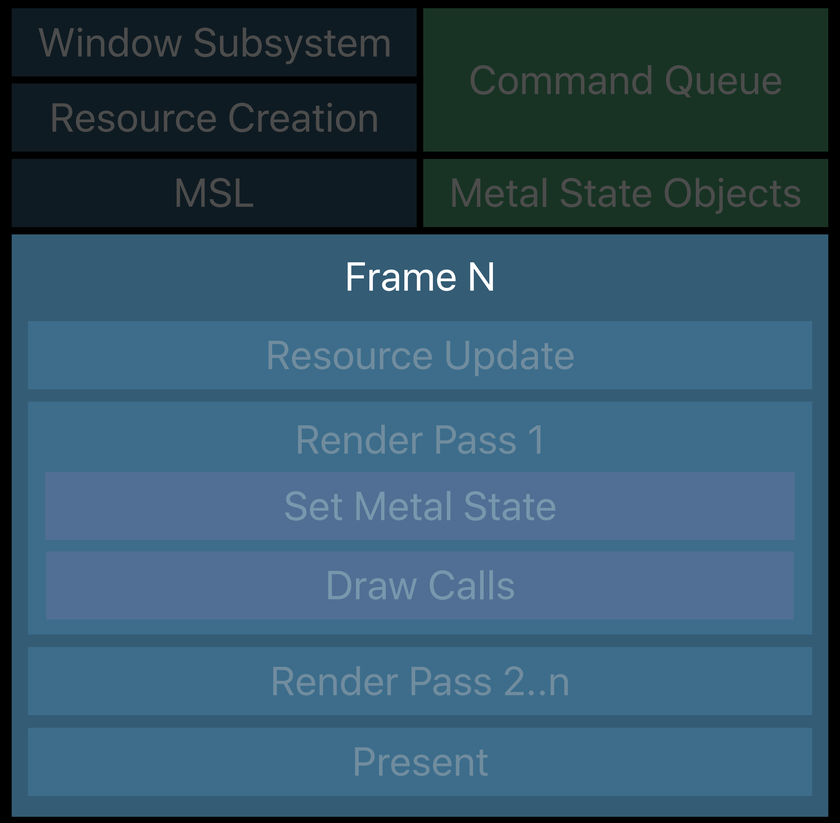
When	Frequency	OpenGL	Metal
Build	Once	—	Shader Compilation
Load	Rare	—	State Validation
Draw	Many times per second	Shader Compilation State Validation Encode Work for GPU	Encode Work for GPU

Life of a Graphics App

Render loop



OpenGL



Metal

Drawing a Frame

Update resources

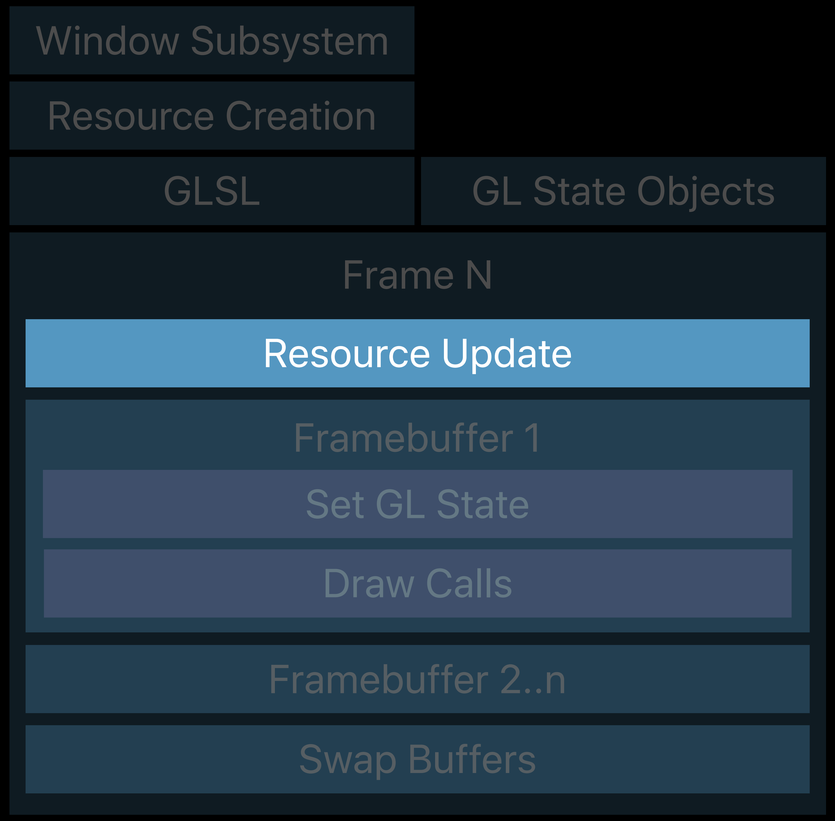
Set the render target

Perform render passes

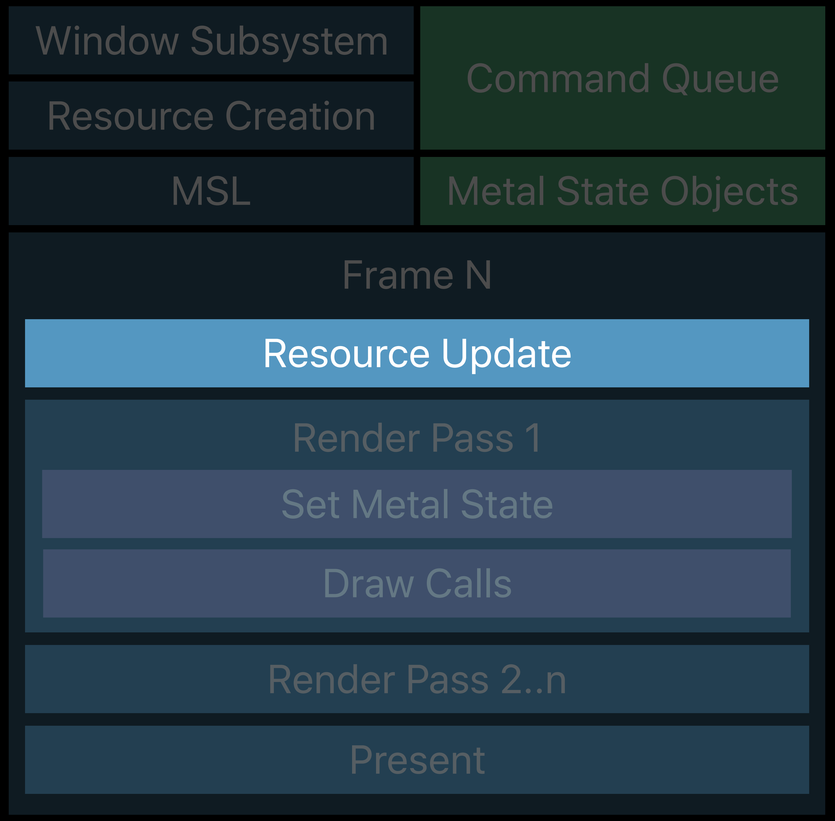
Present

Life of a Graphics App

Resource updates



OpenGL



Metal

Resource Updates

Shader constants

Vertex and index buffers

Textures

Resource Updates in OpenGL

CPU Buffer Update	`glBufferSubData()`
GPU Buffer Update	`glCopyBufferSubData()`
CPU Texture Update	`glTexSubImage2D()`
GPU Texture Update	`glCopyTexSubImage2D()`

Resource Updates in OpenGL

CPU Buffer Update	`glBufferSubData()`
GPU Buffer Update	`glCopyBufferSubData()`
CPU Texture Update	`glTexSubImage2D()`
GPU Texture Update	`glCopyTexSubImage2D()`

Resource Updates in OpenGL

CPU Buffer Update	`glBufferSubData()`
GPU Buffer Update	`glCopyBufferSubData()`
CPU Texture Update	`glTexSubImage2D()`
GPU Texture Update	`glCopyTexSubImage2D()`

Resource Updates in OpenGL

CPU Buffer Update	`glBufferSubData()`
GPU Buffer Update	`glCopyBufferSubData()`
CPU Texture Update	`glTexSubImage2D()`
GPU Texture Update	`glCopyTexSubImage2D()`

Resource Updates in OpenGL

CPU Buffer Update	`glBufferSubData()`
GPU Buffer Update	`glCopyBufferSubData()`
CPU Texture Update	`glTexSubImage2D()`
GPU Texture Update	`glCopyTexSubImage2D()`

Resource Updates in OpenGL

CPU Buffer Update	`glBufferSubData()`
GPU Buffer Update	`glCopyBufferSubData()`
CPU Texture Update	`glTexSubImage2D()`
GPU Texture Update	`glCopyTexSubImage2D()`

Resource Updates in Metal

CPU Buffer Update	`memcpy(mtlBuffer.contents, srcData, updateBytes)`
GPU Buffer Update	`[blitEncoder copyFromBuffer: … ]`
CPU Texture Update	`[mtlTexture replaceRegion: … ]`
GPU Texture Update	`[blitEncoder copyFromTexture: … ]`

Resource Updates in Metal

CPU Buffer Update	`memcpy(mtlBuffer.contents, srcData, updateBytes)`
GPU Buffer Update	`[blitEncoder copyFromBuffer: … ]`
CPU Texture Update	`[mtlTexture replaceRegion: … ]`
GPU Texture Update	`[blitEncoder copyFromTexture: … ]`

Resource Updates in Metal

CPU Buffer Update	`memcpy(mtlBuffer.contents, srcData, updateBytes)`
GPU Buffer Update	`[blitEncoder copyFromBuffer: … ]`
CPU Texture Update	`[mtlTexture replaceRegion: … ]`
GPU Texture Update	`[blitEncoder copyFromTexture: … ]`

Resource Updates in Metal

CPU Buffer Update	`memcpy(mtlBuffer.contents, srcData, updateBytes)`
GPU Buffer Update	`[blitEncoder copyFromBuffer: … ]`
CPU Texture Update	`[mtlTexture replaceRegion: … ]`
GPU Texture Update	`[blitEncoder copyFromTexture: … ]`

Resource Updates in Metal

CPU Buffer Update	`memcpy(mtlBuffer.contents, srcData, updateBytes)`
GPU Buffer Update	`[blitEncoder copyFromBuffer: … ]`
CPU Texture Update	`[mtlTexture replaceRegion: … ]`
GPU Texture Update	`[blitEncoder copyFromTexture: … ]`

Resource Updates in Metal

CPU Buffer Update	`memcpy(mtlBuffer.contents, srcData, updateBytes)`
GPU Buffer Update	`[blitEncoder copyFromBuffer: … ]`
CPU Texture Update	`[mtlTexture replaceRegion: … ]`
GPU Texture Update	`[blitEncoder copyFromTexture: … ]`

Resource Updates in Metal

GPU

Memory

CPU

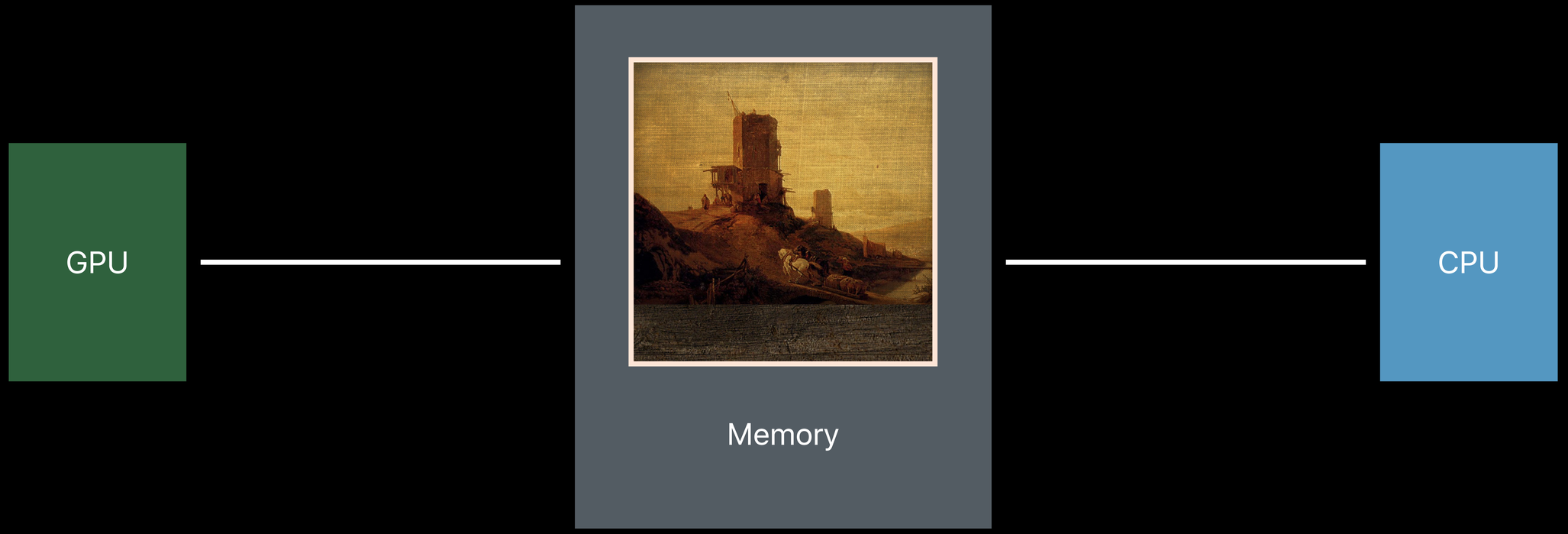
Resource Updates in Metal

GPU

Memory

CPU

Resource Updates in Metal



Resource Updates

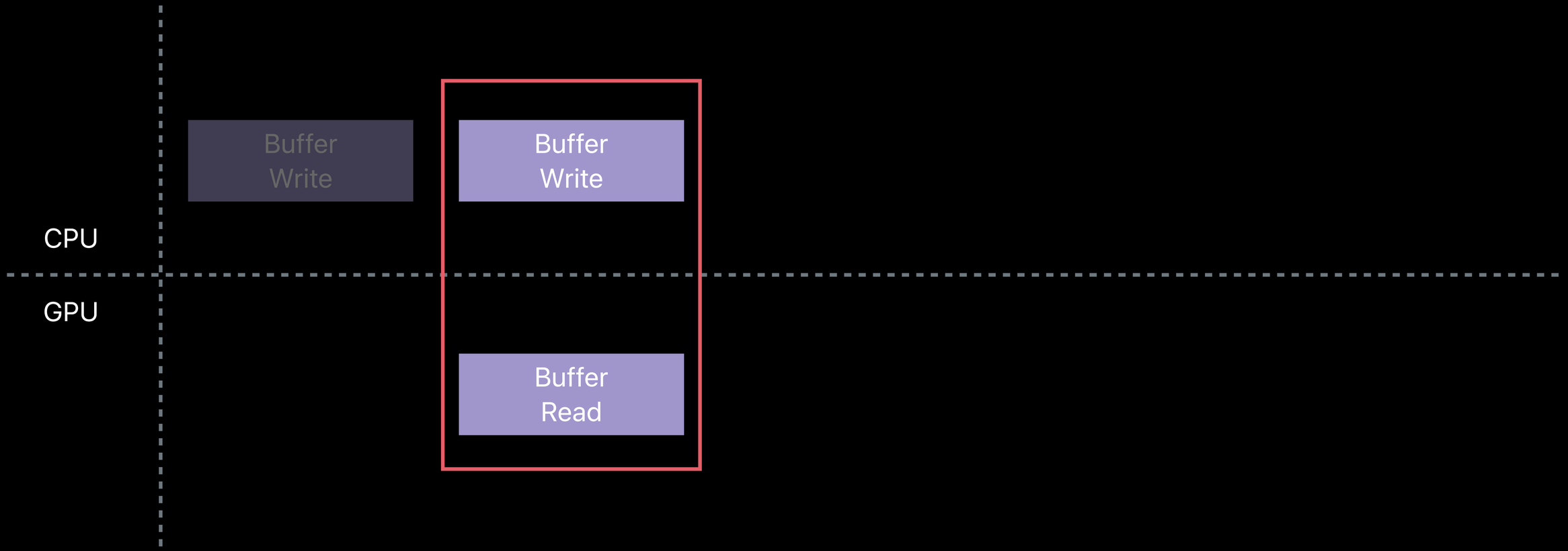
Single buffer — No synchronization

Resource Updates

Single buffer — No synchronization

Resource Updates

Single buffer — No synchronization



Resource Updates

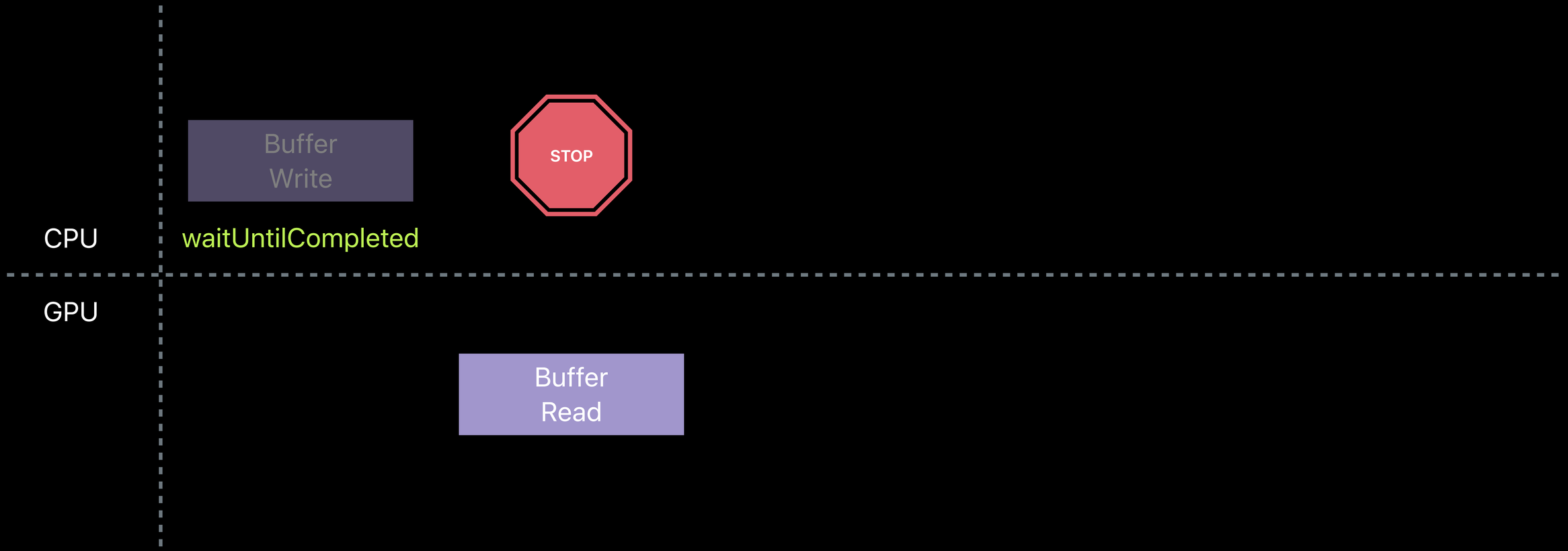
Single buffer — waitUntilCompleted

Resource Updates

Single buffer — waitUntilCompleted

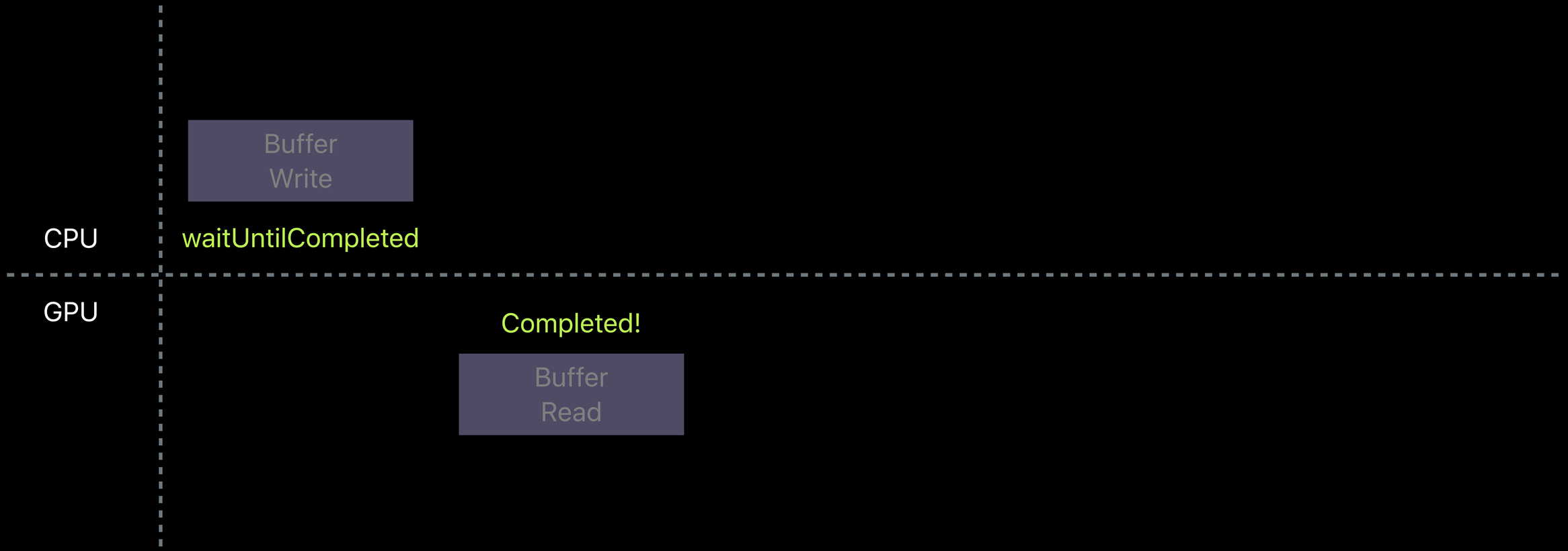
Resource Updates

Single buffer — waitUntilCompleted



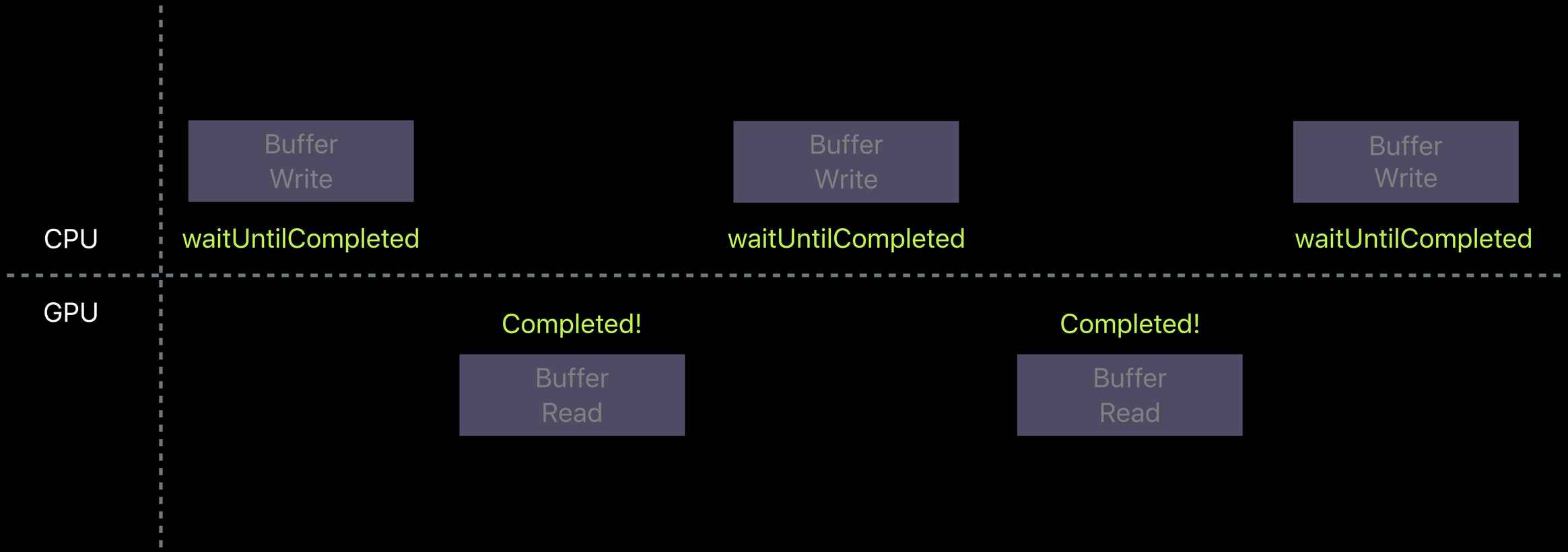
Resource Updates

Single buffer — waitUntilCompleted



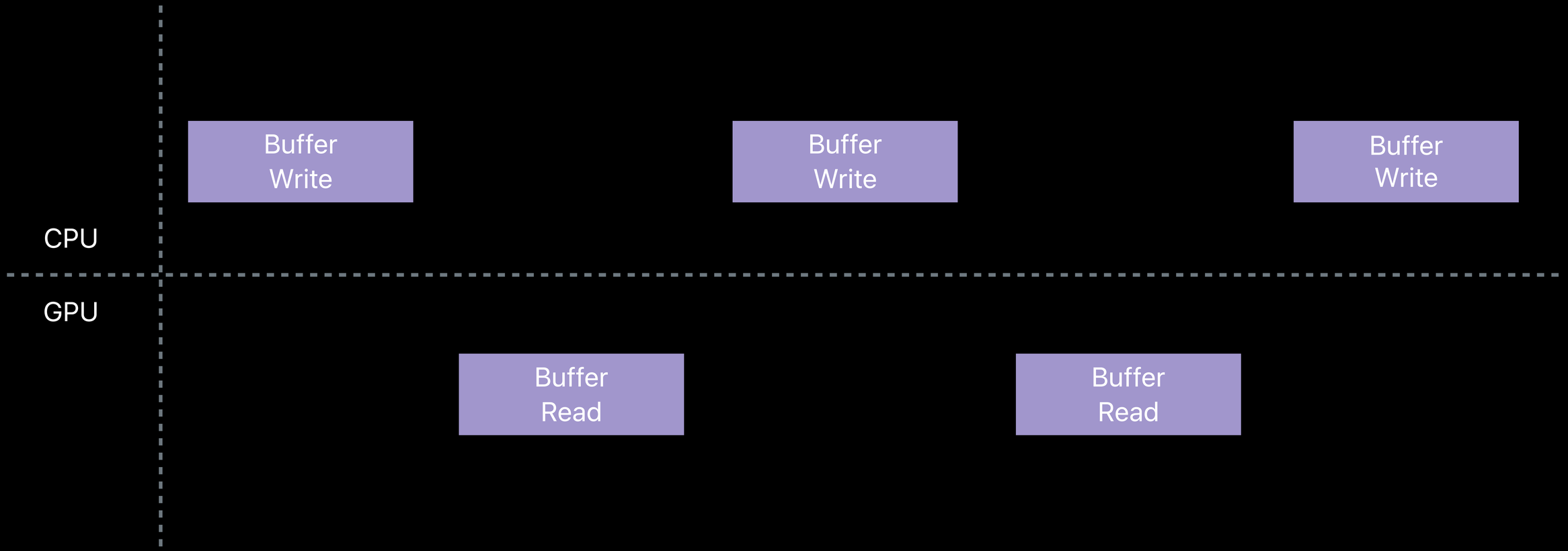
Resource Updates

Single buffer — waitUntilCompleted



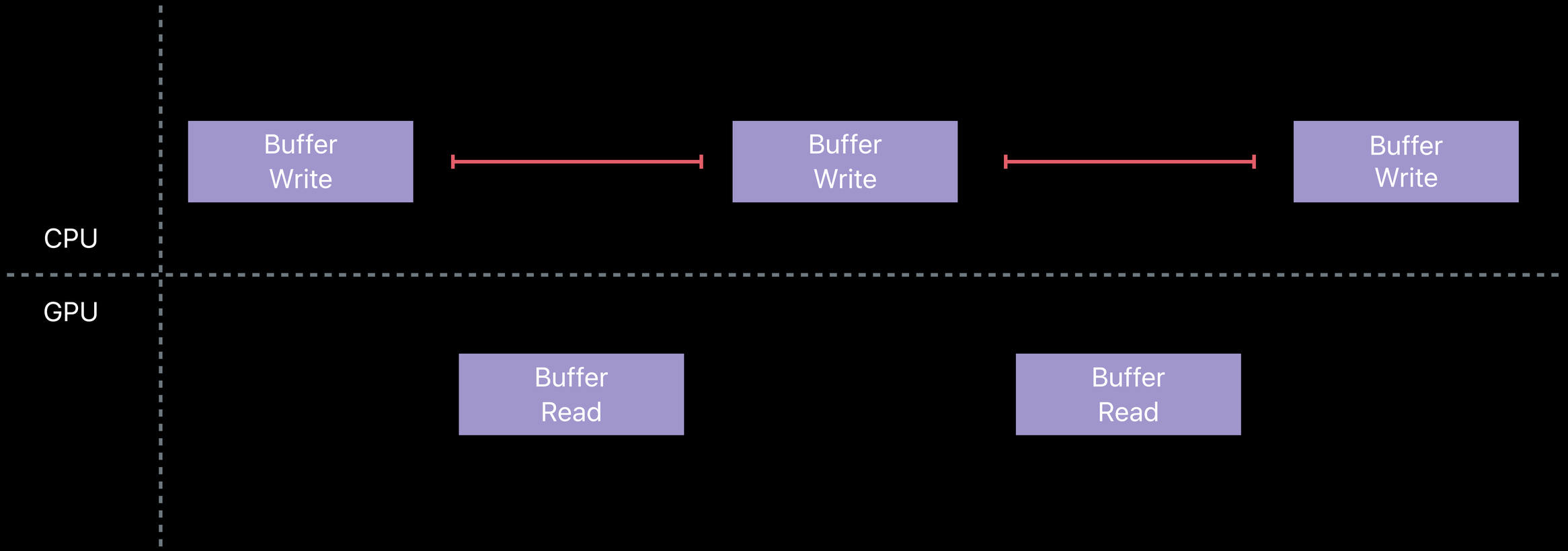
Resource Updates

Single buffer — waitUntilCompleted



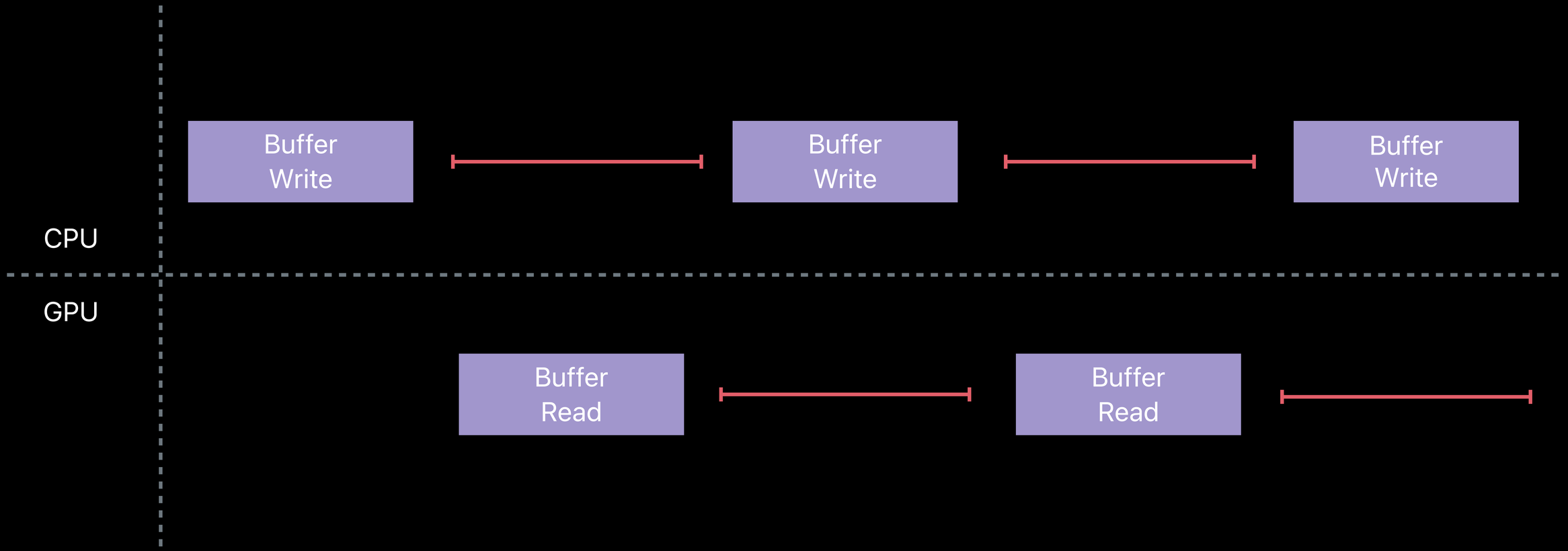
Resource Updates

Single buffer — waitUntilCompleted



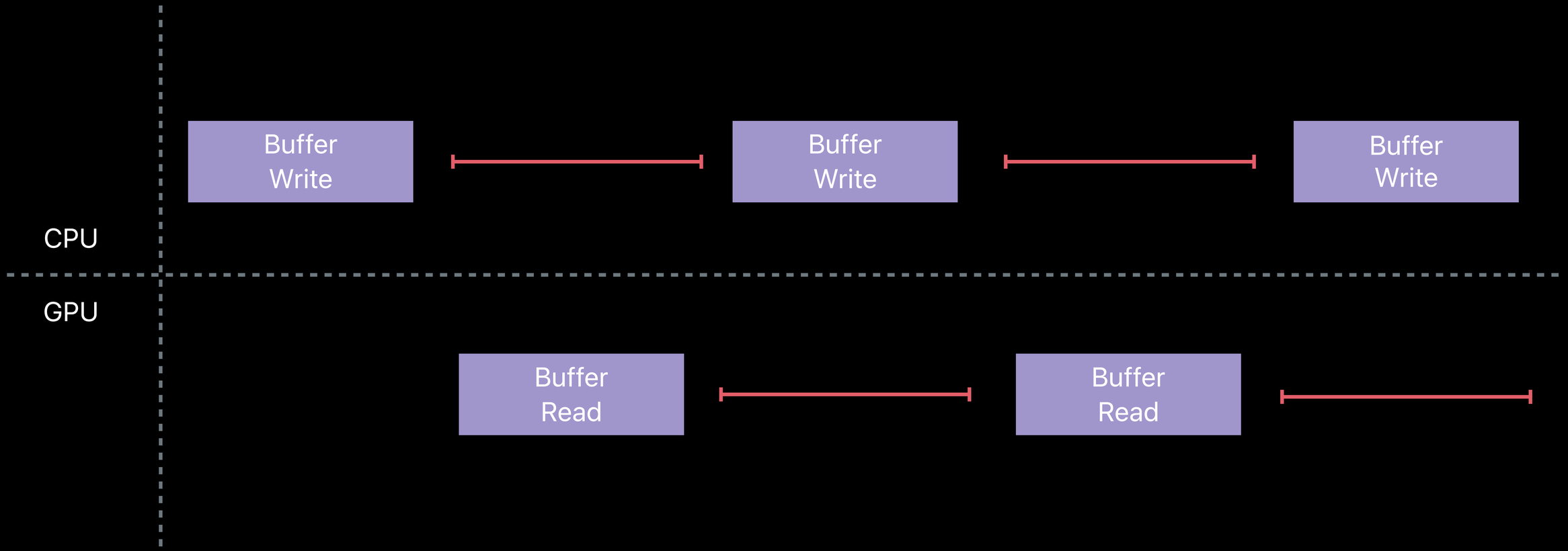
Resource Updates

Single buffer — waitUntilCompleted



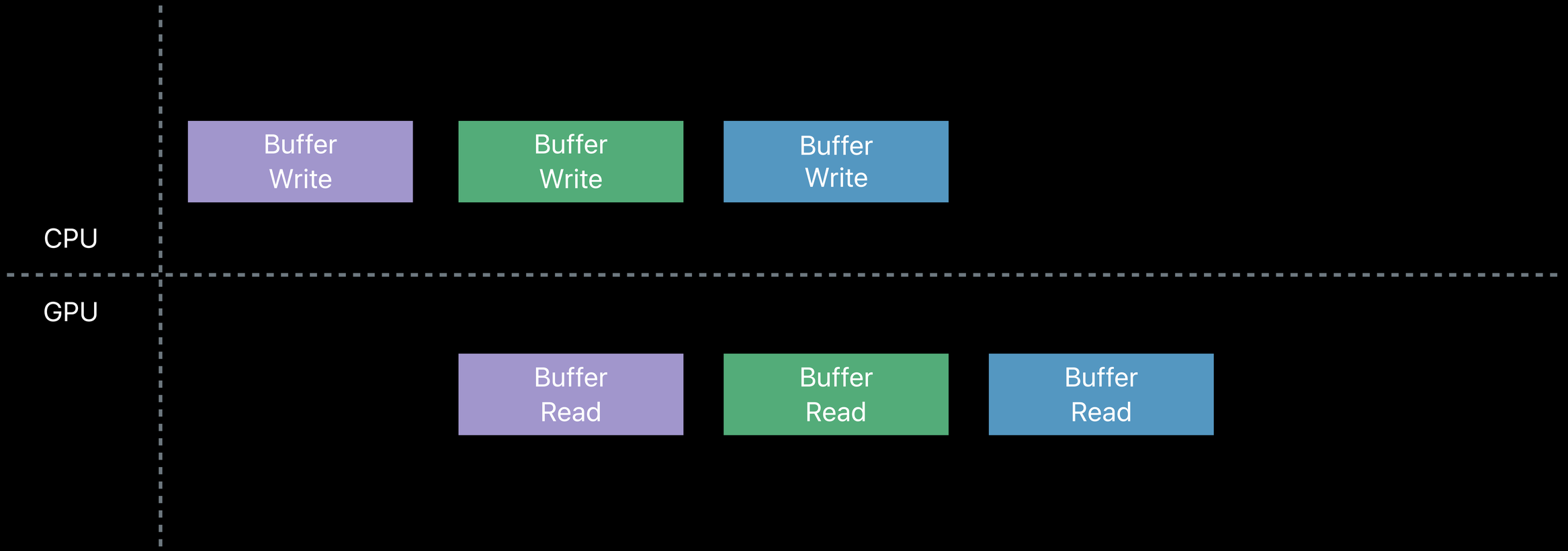
Resource Updates

Single buffer — waitUntilCompleted



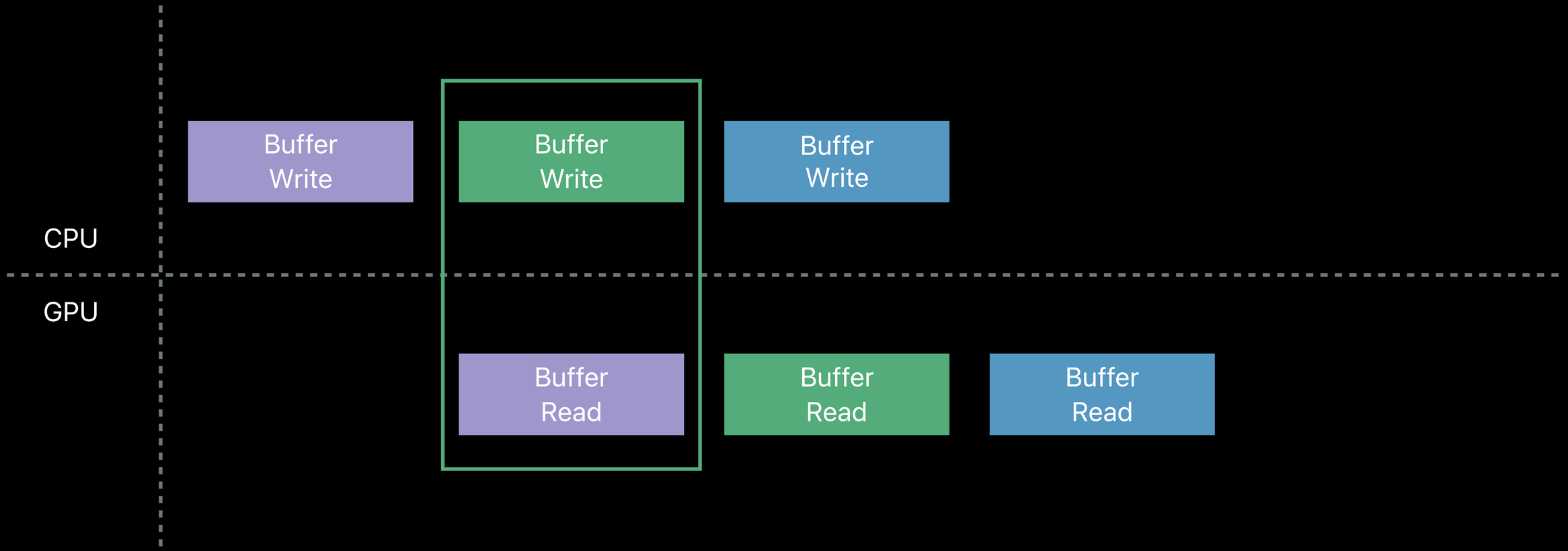
Resource Updates

Multiple buffers



Resource Updates

Multiple buffers



Resource Updates

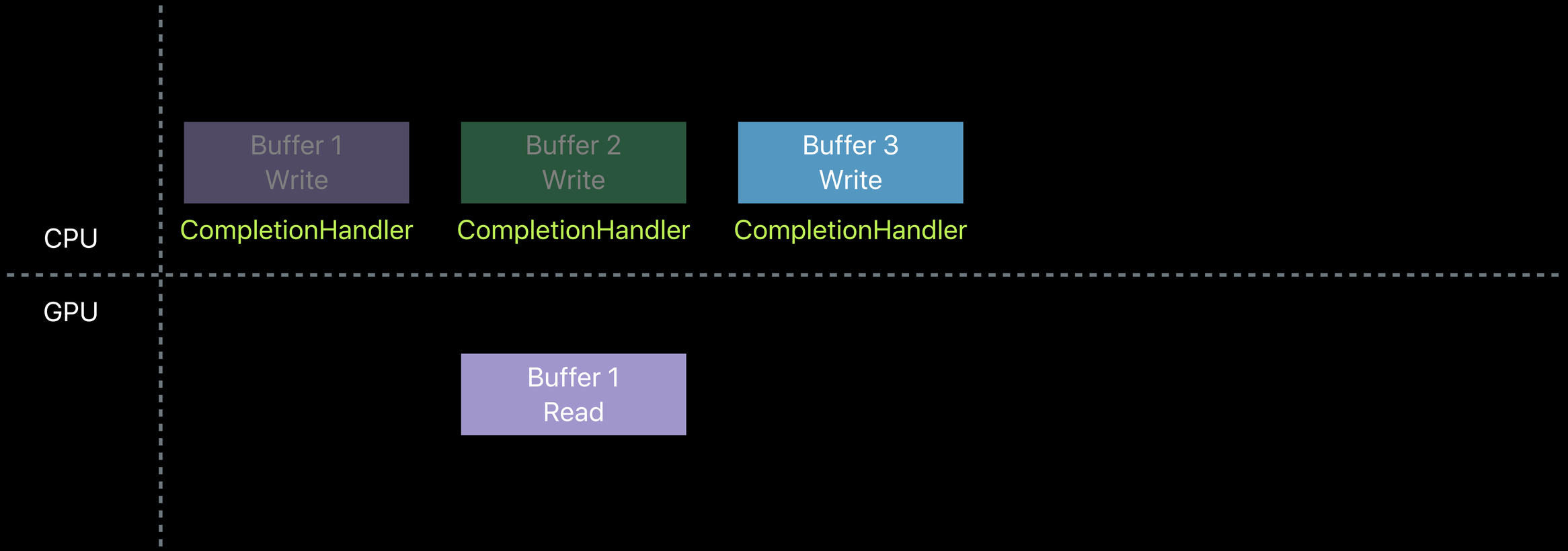
Triple buffer

Resource Updates

Triple buffer

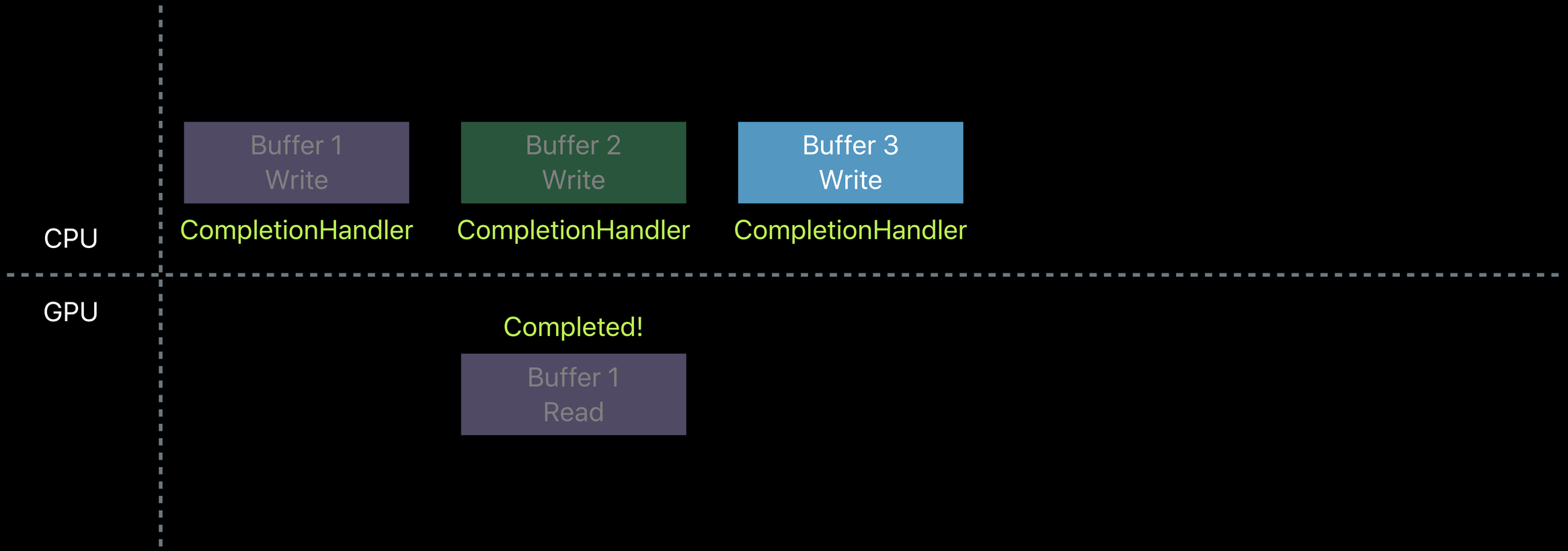
Resource Updates

Triple buffer



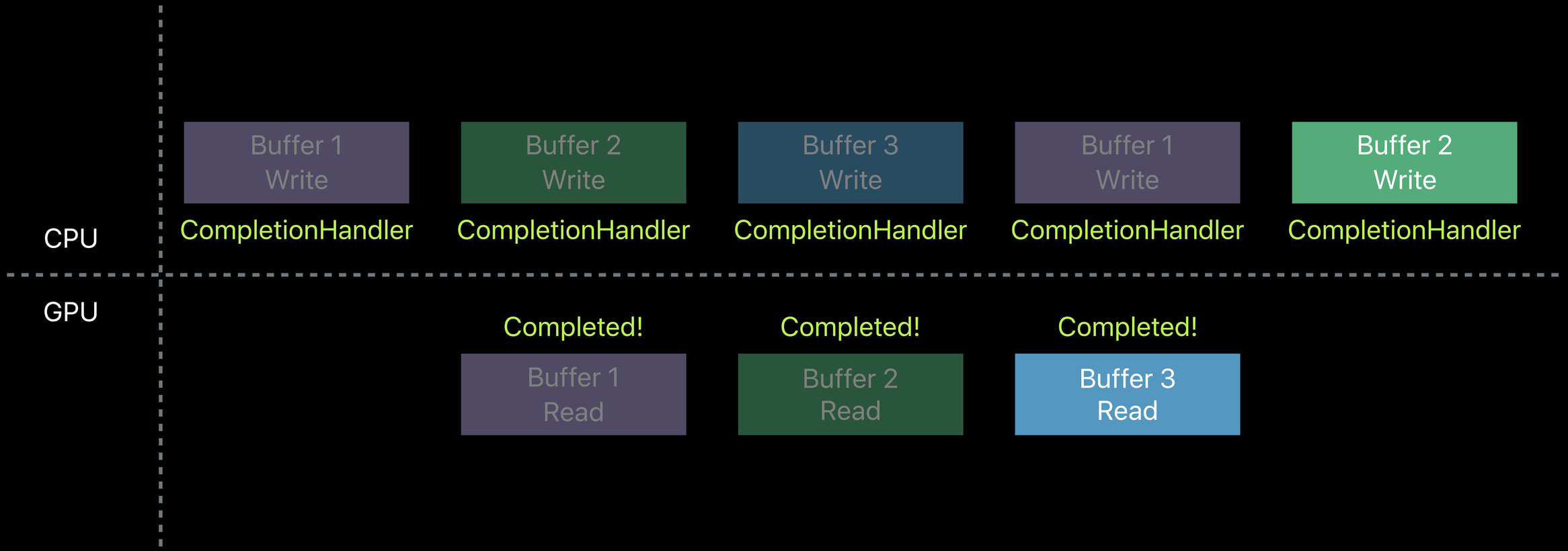
Resource Updates

Triple buffer



Resource Updates

Triple buffer



```
// 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;
```

```
// 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;
```

```objc
- (void)drawInMTKView:(nonnull MTKView *)view {
    // 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];
}
```

```
- (void)drawInMTKView:(nonnull MTKView *)view {
    // 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
- (void)drawInMTKView:(nonnull MTKView *)view {
    // 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
- (void)drawInMTKView:(nonnull MTKView *)view {
    // 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];
}
```

```
- (void)drawInMTKView:(nonnull MTKView *)view {
    // 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];
}
```

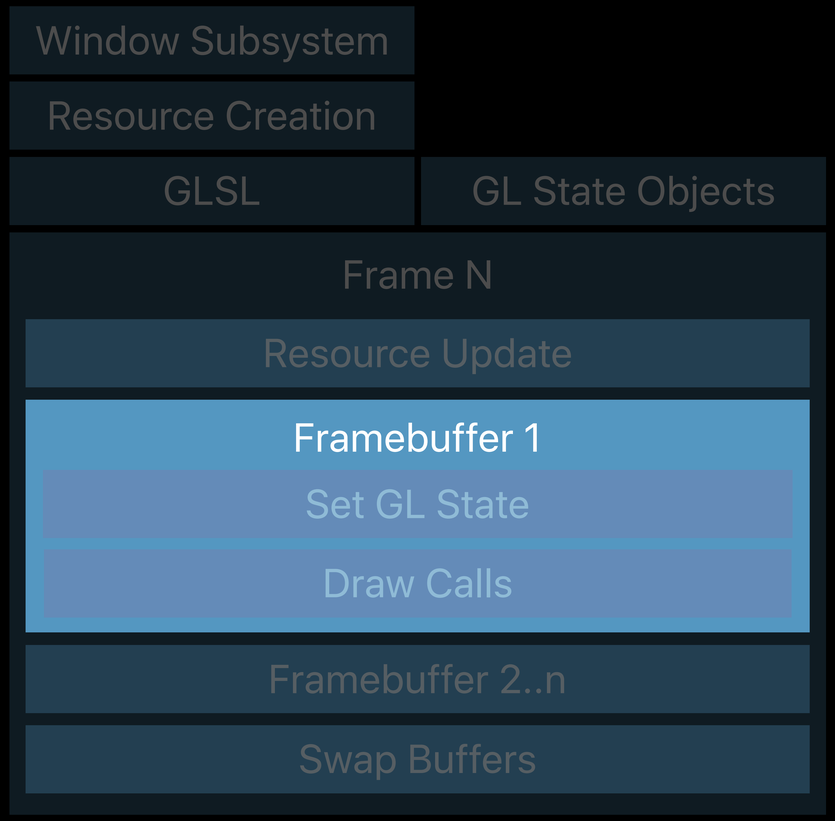
```objc
- (void)drawInMTKView:(nonnull MTKView *)view {
    // Wait until inflight frame is completed
    dispatch_semaphore_wait(frameBoundarySemaphore, DISPATCH_TIME_FOREVER);

    // 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];
}
```

```
- (void)drawInMTKView:(nonnull MTKView *)view {
    // 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];
}
```

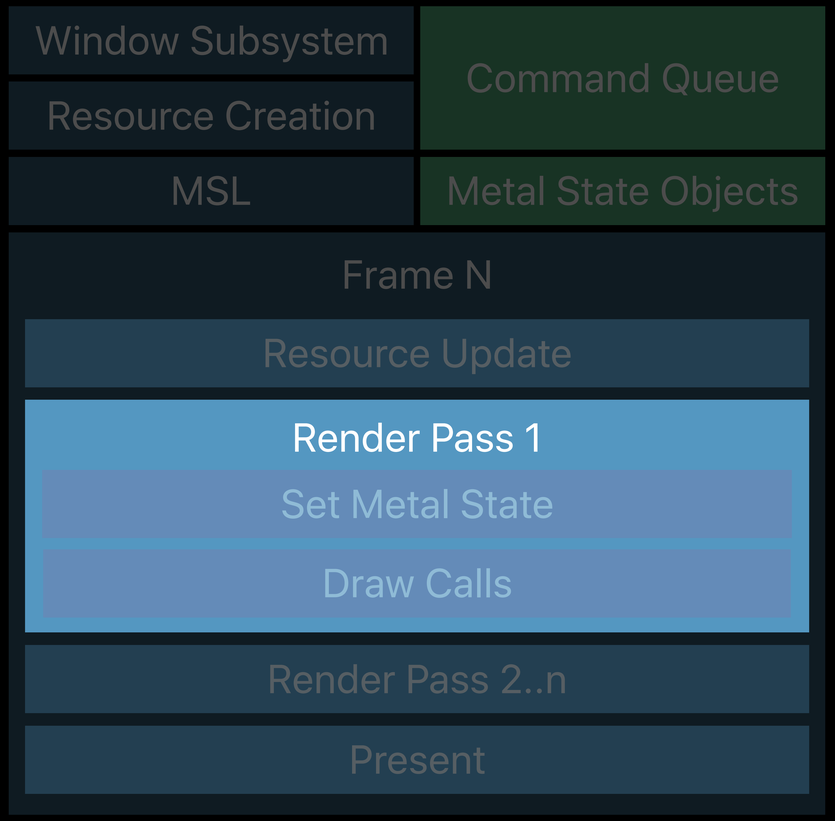
```objc
- (void)drawInMTKView:(nonnull MTKView *)view {
    // Wait until inflight frame is completed
    dispatch_semaphore_wait(frameBoundarySemaphore, DISPATCH_TIME_FOREVER);

    // 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];
}
```

Life of a Graphics App

Inside the render pass



OpenGL



Metal

Render Targets in OpenGL

Framebuffer object (FBO)

Collection of objects

Mutable

Bind and present

OpenGL — Framebuffers

Init

Create	`glGenFramebuffers(1, &myFramebuffer);`
Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Bind Attachment i	`glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 + i, …);`
Validate	`glCheckFramebufferStatus(GL_FRAMEBUFFER);`

Draw

Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Clear	`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
Draw	`// . . .`
Discard	`glDiscardFramebuffer(GL_FRAMEBUFFER, numAttachments, pAttachments);`

OpenGL — Framebuffers

Init

Create	`glGenFramebuffers(1, &myFramebuffer);`
Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Bind Attachment i	`glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 + i, …);`
Validate	`glCheckFramebufferStatus(GL_FRAMEBUFFER);`

Draw

Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Clear	`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
Draw	`// . . .`
Discard	`glDiscardFramebuffer(GL_FRAMEBUFFER, numAttachments, pAttachments);`

OpenGL — Framebuffers

Init

Create	`glGenFramebuffers(1, &myFramebuffer);`
Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Bind Attachment i	`glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 + i, …);`
Validate	`glCheckFramebufferStatus(GL_FRAMEBUFFER);`

Draw

Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Clear	`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
Draw	`// . . .`
Discard	`glDiscardFramebuffer(GL_FRAMEBUFFER, numAttachments, pAttachments);`

OpenGL — Framebuffers

Init

Create	`glGenFramebuffers(1, &myFramebuffer);`
Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Bind Attachment i	`glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 + i, …);`
Validate	`glCheckFramebufferStatus(GL_FRAMEBUFFER);`

Draw

Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Clear	`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
Draw	`// . . .`
Discard	`glDiscardFramebuffer(GL_FRAMEBUFFER, numAttachments, pAttachments);`

OpenGL — Framebuffers

Init

Create	`glGenFramebuffers(1, &myFramebuffer);`
Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Bind Attachment i	`glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 + i, …);`
Validate	`glCheckFramebufferStatus(GL_FRAMEBUFFER);`

Draw

Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Clear	`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
Draw	`// . . .`
Discard	`glDiscardFramebuffer(GL_FRAMEBUFFER, numAttachments, pAttachments);`

OpenGL — Framebuffers

Init

Create	`glGenFramebuffers(1, &myFramebuffer);`
Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Bind Attachment i	`glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 + i, …);`
Validate	`glCheckFramebufferStatus(GL_FRAMEBUFFER);`

Draw

Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Clear	`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
Draw	`// . . .`
Discard	`glDiscardFramebuffer(GL_FRAMEBUFFER, numAttachments, pAttachments);`

OpenGL — Framebuffers

Init

Create	`glGenFramebuffers(1, &myFramebuffer);`
Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Bind Attachment i	`glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 + i, …);`
Validate	`glCheckFramebufferStatus(GL_FRAMEBUFFER);`

Draw

Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Clear	`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
Draw	`// . . .`
Discard	`glDiscardFramebuffer(GL_FRAMEBUFFER, numAttachments, pAttachments);`

OpenGL — Framebuffers

Init

Create	`glGenFramebuffers(1, &myFramebuffer);`
Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Bind Attachment i	`glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 + i, …);`
Validate	`glCheckFramebufferStatus(GL_FRAMEBUFFER);`

Draw

Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Clear	`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
Draw	`// . . .`
Discard	`glDiscardFramebuffer(GL_FRAMEBUFFER, numAttachments, pAttachments);`

OpenGL — Framebuffers

Init

Create	`glGenFramebuffers(1, &myFramebuffer);`
Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Bind Attachment i	`glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 + i, …);`
Validate	`glCheckFramebufferStatus(GL_FRAMEBUFFER);`

Draw

Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Clear	`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
Draw	`// . . .`
Discard	`glDiscardFramebuffer(GL_FRAMEBUFFER, numAttachments, pAttachments);`

OpenGL — Framebuffers

Init

Create	`glGenFramebuffers(1, &myFramebuffer);`
Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Bind Attachment i	`glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0 + i, …);`
Validate	`glCheckFramebufferStatus(GL_FRAMEBUFFER);`

Draw

Make Current	`glBindFramebuffer(GL_FRAMEBUFFER, myFramebuffer);`
Clear	`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
Draw	`// . . .`
Discard	`glDiscardFramebuffer(GL_FRAMEBUFFER, numAttachments, pAttachments);`

Render Targets in Metal

Render Command encoder

Created from a Render Pass Descriptor

Generates hardware commands

Explicitly starts and ends

Metal — Render Encoders

Create Descriptor

```
desc = [MTLRenderPassDescriptor renderPassDescriptor];
```

Specify Attachment i

```
desc.colorAttachments[i].texture = renderTargetTexture;
```

Specify Clear Color

Specify Load/ Store Action

```
desc.colorAttachments[i].clearColor = MTLClearColorMake(0, 0, 0, 1);
desc.colorAttachments[i].loadAction = MTLLoadActionClear;
desc.colorAttachments[i].storeAction = MTLStoreActionDontCare;
```

Init

Draw

Start Render Pass

```
renderEncoder= [mtlCommandBuffer renderCommandEncoderWithDescriptor:desc];
```

Draw

```
//…
```

End Render Pass

```
[renderEncoder endEncoding];
```

Metal — Render Encoders

Create Descriptor

```
desc = [MTLRenderPassDescriptor renderPassDescriptor];
```

Specify Attachment i

```
desc.colorAttachments[i].texture = renderTargetTexture;
```

Specify Clear Color

```
desc.colorAttachments[i].clearColor = MTLClearColorMake(0, 0, 0, 1);
```

Specify Load/ Store Action

```
desc.colorAttachments[i].loadAction = MTLLoadActionClear;
desc.colorAttachments[i].storeAction = MTLStoreActionDontCare;
```

Init

Draw

Start Render Pass

```
renderEncoder= [mtlCommandBuffer renderCommandEncoderWithDescriptor:desc];
```

Draw

```
//…
```

End Render Pass

```
[renderEncoder endEncoding];
```

Metal — Render Encoders

Create Descriptor

```
desc = [MTLRenderPassDescriptor renderPassDescriptor];
```

Specify Attachment i

```
desc.colorAttachments[i].texture = renderTargetTexture;
```

Specify Clear Color

Specify Load/ Store Action

```
desc.colorAttachments[i].clearColor = MTLClearColorMake(0, 0, 0, 1);
desc.colorAttachments[i].loadAction = MTLLoadActionClear;
desc.colorAttachments[i].storeAction = MTLStoreActionDontCare;
```

Init

Draw

Start Render Pass

```
renderEncoder= [mtlCommandBuffer renderCommandEncoderWithDescriptor:desc];
```

Draw

```
//…
```

End Render Pass

```
[renderEncoder endEncoding];
```

Metal — Render Encoders

Create Descriptor

```
desc = [MTLRenderPassDescriptor renderPassDescriptor];
```

Specify Attachment i

```
desc.colorAttachments[i].texture = renderTargetTexture;
```

Specify Clear Color

```
desc.colorAttachments[i].clearColor = MTLClearColorMake(0, 0, 0, 1);
```

Specify Load/ Store Action

```
desc.colorAttachments[i].loadAction = MTLLoadActionClear;
desc.colorAttachments[i].storeAction = MTLStoreActionDontCare;
```

Init

Draw

Start Render Pass

```
renderEncoder= [mtlCommandBuffer renderCommandEncoderWithDescriptor:desc];
```

Draw

```
//…
```

End Render Pass

```
[renderEncoder endEncoding];
```

Metal — Render Encoders

Create Descriptor

```
desc = [MTLRenderPassDescriptor renderPassDescriptor];
```

Specify Attachment i

```
desc.colorAttachments[i].texture = renderTargetTexture;
```

Specify Clear Color

```
desc.colorAttachments[i].clearColor = MTLClearColorMake(0, 0, 0, 1);
```

Specify Load/ Store Action

```
desc.colorAttachments[i].loadAction = MTLLoadActionClear;
desc.colorAttachments[i].storeAction = MTLStoreActionDontCare;
```

Init

Draw

Start Render Pass

```
renderEncoder= [mtlCommandBuffer renderCommandEncoderWithDescriptor:desc];
```

Draw

```
//…
```

End Render Pass

```
[renderEncoder endEncoding];
```

Metal — Render Encoders

Create Descriptor

```
desc = [MTLRenderPassDescriptor renderPassDescriptor];
```

Specify Attachment i

```
desc.colorAttachments[i].texture = renderTargetTexture;
```

Specify Clear Color

```
desc.colorAttachments[i].clearColor = MTLClearColorMake(0, 0, 0, 1);
```

Specify Load/ Store Action

```
desc.colorAttachments[i].loadAction = MTLLoadActionClear;
desc.colorAttachments[i].storeAction = MTLStoreActionDontCare;
```

Init

Draw

Start Render Pass

```
renderEncoder= [mtlCommandBuffer renderCommandEncoderWithDescriptor:desc];
```

Draw

```
//…
```

End Render Pass

```
[renderEncoder endEncoding];
```

Metal — Render Encoders

Create Descriptor

```
desc = [MTLRenderPassDescriptor renderPassDescriptor];
```

Specify Attachment i

```
desc.colorAttachments[i].texture = renderTargetTexture;
```

Specify Clear Color

Specify Load/ Store Action

```
desc.colorAttachments[i].clearColor = MTLClearColorMake(0, 0, 0, 1);
desc.colorAttachments[i].loadAction = MTLLoadActionClear;
desc.colorAttachments[i].storeAction = MTLStoreActionDontCare;
```

Init

Draw

Start Render Pass

```
renderEncoder= [mtlCommandBuffer renderCommandEncoderWithDescriptor:desc];
```

Draw

```
//…
```

End Render Pass

```
[renderEncoder endEncoding];
```

Metal — Render Encoders

Create Descriptor

```
desc = [MTLRenderPassDescriptor renderPassDescriptor];
```

Specify Attachment i

```
desc.colorAttachments[i].texture = renderTargetTexture;
```

Specify Clear Color

Specify Load/ Store Action

```
desc.colorAttachments[i].clearColor = MTLClearColorMake(0, 0, 0, 1);
desc.colorAttachments[i].loadAction = MTLLoadActionClear;
desc.colorAttachments[i].storeAction = MTLStoreActionDontCare;
```

Init

Draw

Start Render Pass

```
renderEncoder= [mtlCommandBuffer renderCommandEncoderWithDescriptor:desc];
```

Draw

```
//…
```

End Render Pass

```
[renderEncoder endEncoding];
```

Metal — Render Encoders

Create Descriptor

```
desc = [MTLRenderPassDescriptor renderPassDescriptor];
```

Specify Attachment i

```
desc.colorAttachments[i].texture = renderTargetTexture;
```

Specify Clear Color

Specify Load/ Store Action

```
desc.colorAttachments[i].clearColor = MTLClearColorMake(0, 0, 0, 1);
desc.colorAttachments[i].loadAction = MTLLoadActionClear;
desc.colorAttachments[i].storeAction = MTLStoreActionDontCare;
```

Init

Draw

Start Render Pass

```
renderEncoder= [mtlCommandBuffer renderCommandEncoderWithDescriptor:desc];
```

Draw

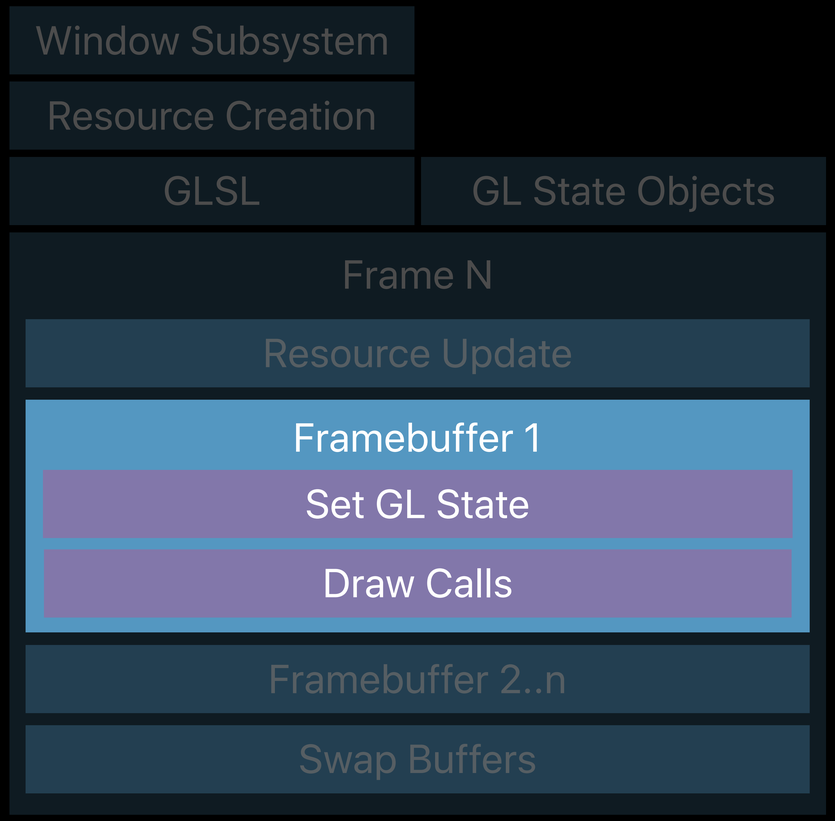
```
//…
```

End Render Pass

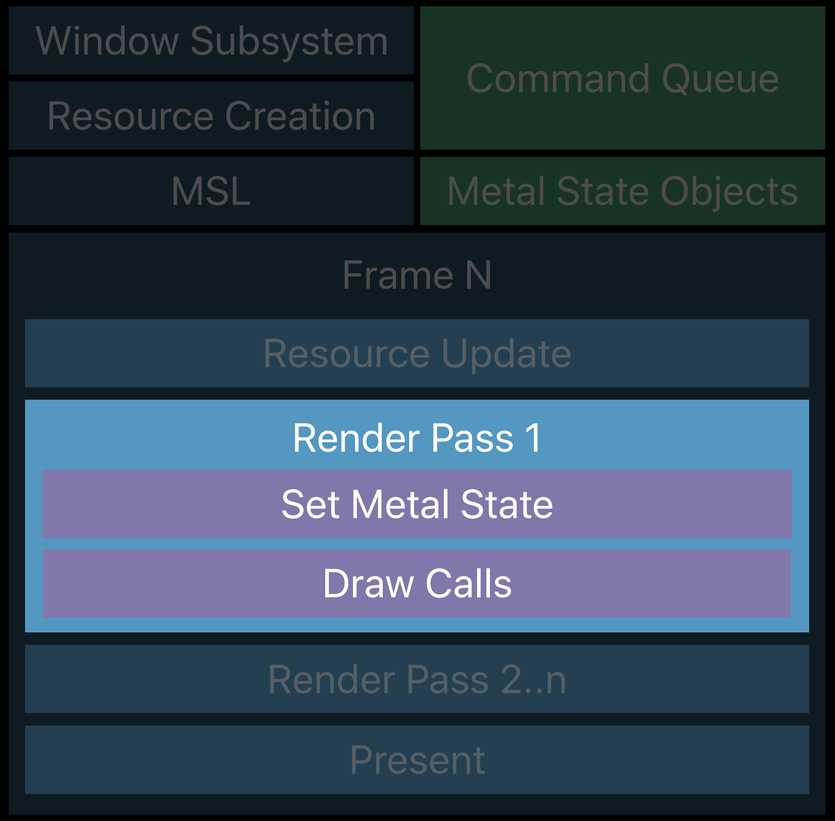
```
[renderEncoder endEncoding];
```

Life of a Graphics App

Inside the render pass



OpenGL



Metal

Drawing a Frame

Render pass

Set up state and draw call inputs

Issue draw commands

Rendering with OpenGL

GL State	`// glEnable(); glVertexAttribPointer(); glColorMask(); glBlendFunc(); …`
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

GL State	`// glEnable(); glVertexAttribPointer(); glColorMask(); glBlendFunc(); …`
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

GL State	`// glEnable(); glVertexAttribPointer(); glColorMask(); glBlendFunc(); …`
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

GL State	`// glEnable(); glVertexAttribPointer(); glColorMask(); glBlendFunc(); …`
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

GL State	`// glEnable(); glVertexAttribPointer(); glColorMask(); glBlendFunc(); …`
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

GL State	`// glEnable(); glVertexAttribPointer(); glColorMask(); glBlendFunc(); …`
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

GL State	`// glEnable(); glVertexAttribPointer(); glColorMask(); glBlendFunc(); …`
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);`

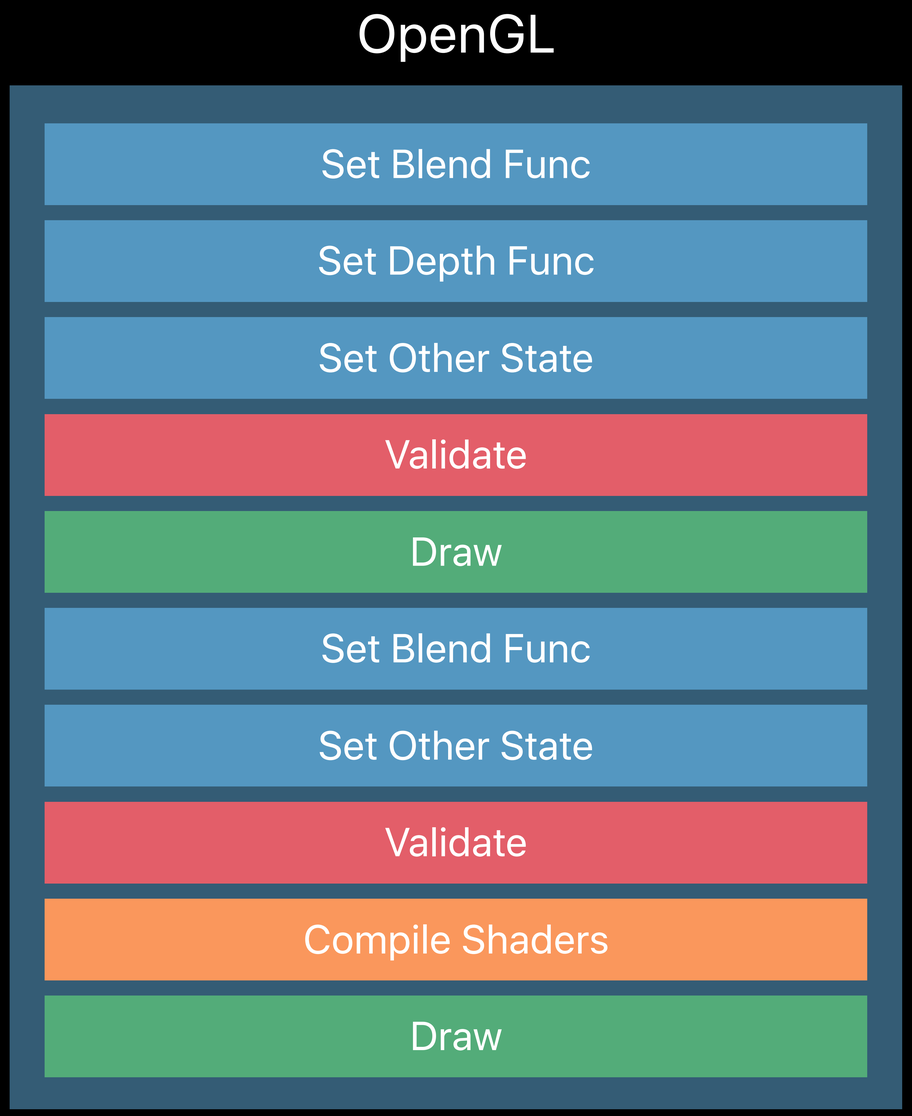
Rendering with OpenGL

GL State	`// glEnable(); glVertexAttribPointer(); glColorMask(); glBlendFunc(); …`
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

GL State	`// glEnable(); glVertexAttribPointer(); glColorMask(); glBlendFunc(); …`
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);`

The Main Difference — Setting State



The Main Difference — Setting State

Rendering with Metal

Render Targets

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

PSO

```
[encoder setPipelineState:myPipeline];
```

Vertex Buffers

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

Uniforms

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

Textures

```
[encoder setFragmentTexture:myColorTexture atIndex:0];
```

Draws

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

Rendering with Metal

Render Targets

PSO

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];
```

Rendering with Metal

Render Targets

PSO

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];
```

Rendering with Metal

Render Targets

PSO

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];
```

Rendering with Metal

Render Targets

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

PSO

```
[encoder setPipelineState:myPipeline];
```

Vertex Buffers

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

Uniforms

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

Textures

```
[encoder setFragmentTexture:myColorTexture atIndex:0];
```

Draws

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

Rendering with Metal

- Render Targets
- PSO
- 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];
```

Rendering with Metal

Render Targets

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

PSO

```
[encoder setPipelineState:myPipeline];
```

Vertex Buffers

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

Uniforms

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

Textures

```
[encoder setFragmentTexture:myColorTexture atIndex:0];
```

Draws

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

Rendering with Metal

Render Targets

PSO

Vertex Buffers

Uniforms

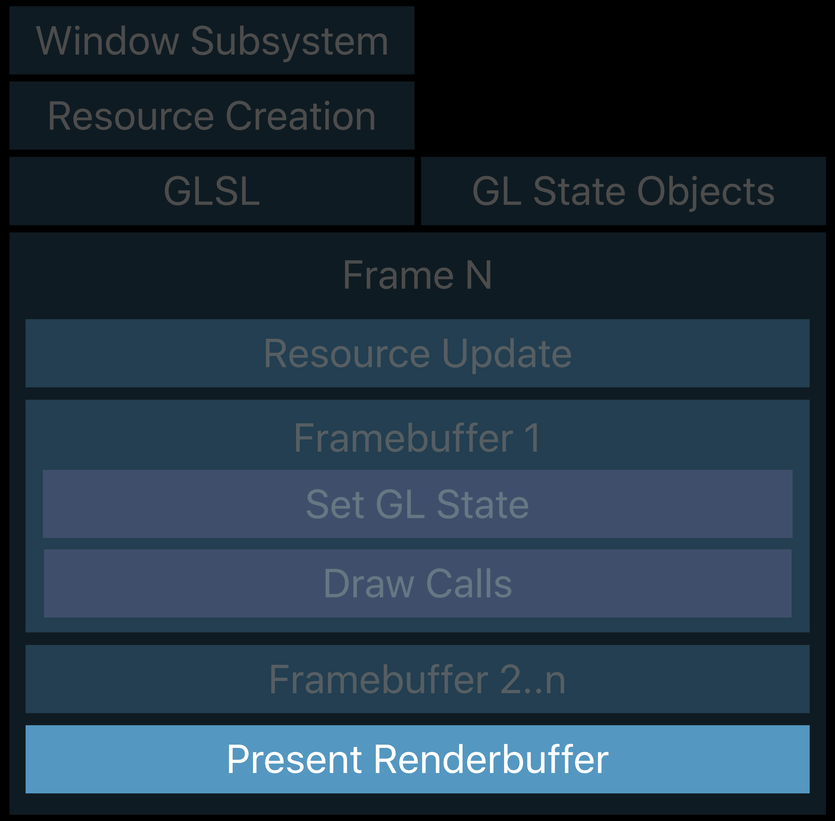
Textures

Draws

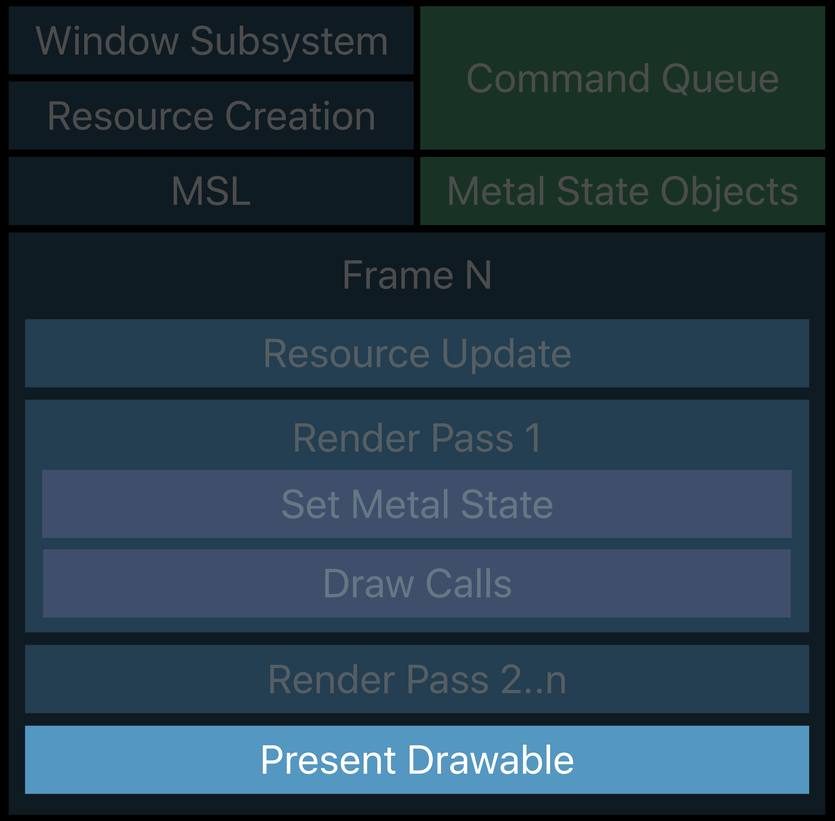
```
encoder = [commandBuffer renderCommandEncoderWithDescriptor:descriptor];

[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];
```

Life of a Graphics App

Presenting



OpenGL



Metal

Presenting

OpenGL

- Framebuffer

```
[context presentRenderbuffer:GL_RENDERBUFFER]
```

Metal

- Drawables — textures for on-screen display

```
[commandBuffer presentDrawable:drawable]
```

```
// GLKViewDelegate callback

- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect;
```

OpenGL

Metal

```
// MTKViewDelegate callbacks

- (void) mtkView:(MTKView *)view drawableSizeWillChange:(CGSize)size;
- (void) drawInMTKView:(nonnull MTKView *)view;
```

```
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect
```

OpenGL

```
- (void) drawInMTKView:(nonnull MTKView *)view
```

Metal

```objc
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect {
    // Bind Framebuffer
    // Render loop source

    return; // [context presentRenderbuffer:GL_RENDERBUFFER]; is called for you
}
```

OpenGL

Metal

```objc
- (void) drawInMTKView:(nonnull MTKView *)view {
    id<MTLCommandBuffer> mtlCommandBuffer = [mtlCommandQueue commandBuffer];

    // Render loop source (encoders)

    [mtlCommandBuffer presentDrawable:[view currentDrawable]];
    [mtlCommandBuffer commit];
}
```

```objc
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect {
    // Bind Framebuffer
    // Render loop source

    return; // [context presentRenderbuffer:GL_RENDERBUFFER]; is called for you
}
```

OpenGL

Metal

```objc
- (void) drawInMTKView:(nonnull MTKView *)view {
    id<MTLCommandBuffer> mtlCommandBuffer = [mtlCommandQueue commandBuffer];

    // Render loop source (encoders)

    [mtlCommandBuffer presentDrawable:[view currentDrawable]];
    [mtlCommandBuffer commit];
}
```

```objc
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect {
    // Bind Framebuffer
    // Render loop source

    return; // [context presentRenderbuffer:GL_RENDERBUFFER]; is called for you
}
```

OpenGL

Metal

```objc
- (void) drawInMTKView:(nonnull MTKView *)view {
    id<MTLCommandBuffer> mtlCommandBuffer = [mtlCommandQueue commandBuffer];

    // Render loop source (encoders)

    [mtlCommandBuffer presentDrawable:[view currentDrawable]];
    [mtlCommandBuffer commit];
}
```

```objc
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect {
    // Bind Framebuffer
    // Render loop source

    return; // [context presentRenderbuffer:GL_RENDERBUFFER]; is called for you
}
```

OpenGL

Metal

```objc
- (void) drawInMTKView:(nonnull MTKView *)view {
    id<MTLCommandBuffer> mtlCommandBuffer = [mtlCommandQueue commandBuffer];

    // Render loop source (encoders)

    [mtlCommandBuffer presentDrawable:[view currentDrawable]];
    [mtlCommandBuffer commit];
}
```

```objc
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect {
    // Bind Framebuffer
    // Render loop source

    return; // [context presentRenderbuffer:GL_RENDERBUFFER]; is called for you
}
```

OpenGL

Metal

```objc
- (void) drawInMTKView:(nonnull MTKView *)view {
    id<MTLCommandBuffer> mtlCommandBuffer = [mtlCommandQueue commandBuffer];

    // Render loop source (encoders)

    [mtlCommandBuffer presentDrawable:[view currentDrawable]];
    [mtlCommandBuffer commit];
}
```

```objc
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect {
    // Bind Framebuffer
    // Render loop source

    return; // [context presentRenderbuffer:GL_RENDERBUFFER]; is called for you
}
```

OpenGL

Metal

```objc
- (void) drawInMTKView:(nonnull MTKView *)view {
    id<MTLCommandBuffer> mtlCommandBuffer = [mtlCommandQueue commandBuffer];

    // Render loop source (encoders)

    [mtlCommandBuffer presentDrawable:[view currentDrawable]];
    [mtlCommandBuffer commit];
}
```

```objc
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect {
    // Bind Framebuffer
    // Render loop source

    return; // [context presentRenderbuffer:GL_RENDERBUFFER]; is called for you
}
```

OpenGL

Metal

```objc
- (void) drawInMTKView:(nonnull MTKView *)view {
    id<MTLCommandBuffer> mtlCommandBuffer = [mtlCommandQueue commandBuffer];

    // Render loop source (encoders)

    [mtlCommandBuffer presentDrawable:[view currentDrawable]];
    [mtlCommandBuffer commit];
}
```

```objc
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect {
    // Bind Framebuffer
    // Render loop source

    return; // [context presentRenderbuffer:GL_RENDERBUFFER]; is called for you
}
```

OpenGL

Metal

```objc
- (void) drawInMTKView:(nonnull MTKView *)view {
    id<MTLCommandBuffer> mtlCommandBuffer = [mtlCommandQueue commandBuffer];

    // Render loop source (encoders)

    [mtlCommandBuffer presentDrawable:[view currentDrawable]];
    [mtlCommandBuffer commit];
}
```

```objc
- (void) glkView:(GLKView *)view drawInRect:(CGRect)rect {
    // Bind Framebuffer
    // Render loop source

    return; // [context presentRenderbuffer:GL_RENDERBUFFER]; is called for you
}
```

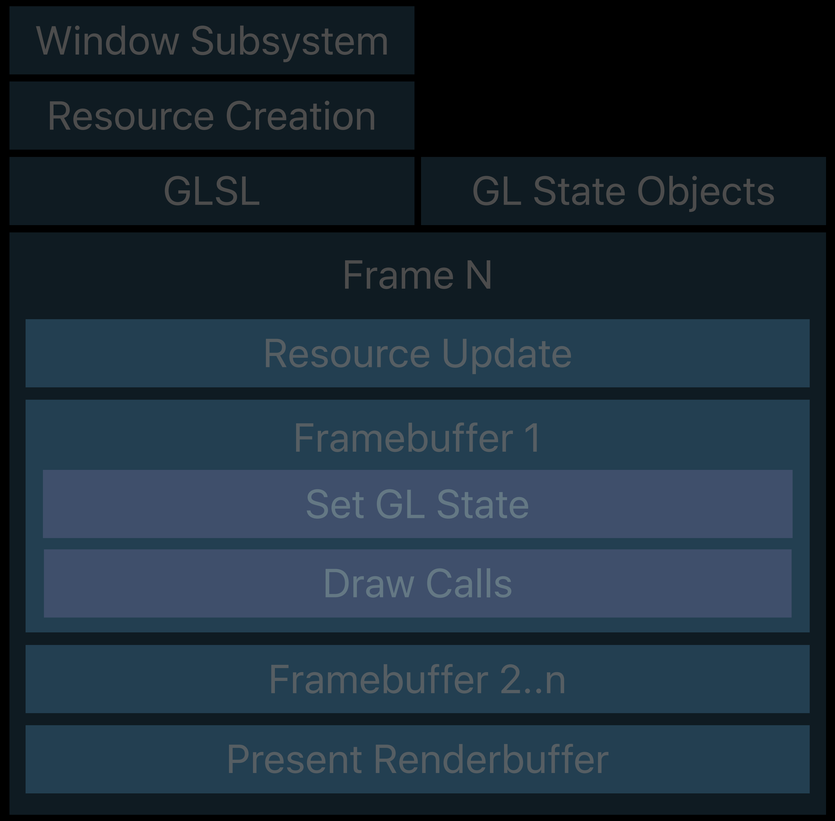
OpenGL

Metal

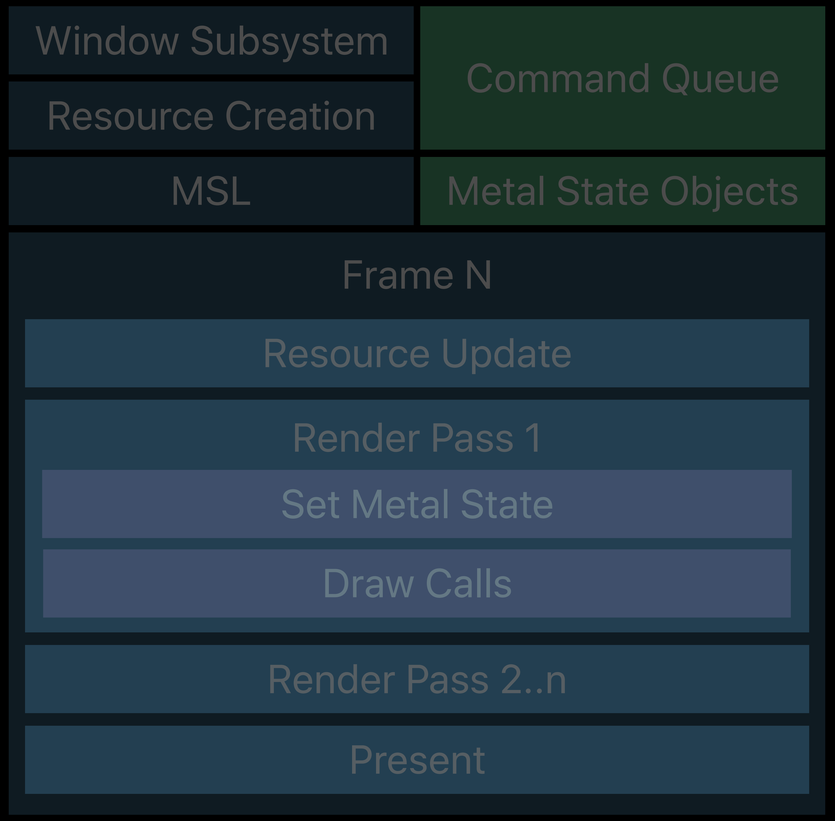
```objc
- (void) drawInMTKView:(nonnull MTKView *)view {
    id<MTLCommandBuffer> mtlCommandBuffer = [mtlCommandQueue commandBuffer];

    // Render loop source (encoders)

    [mtlCommandBuffer presentDrawable:[view currentDrawable]];
    [mtlCommandBuffer commit];
}
```

Life of a Graphics App

Recap



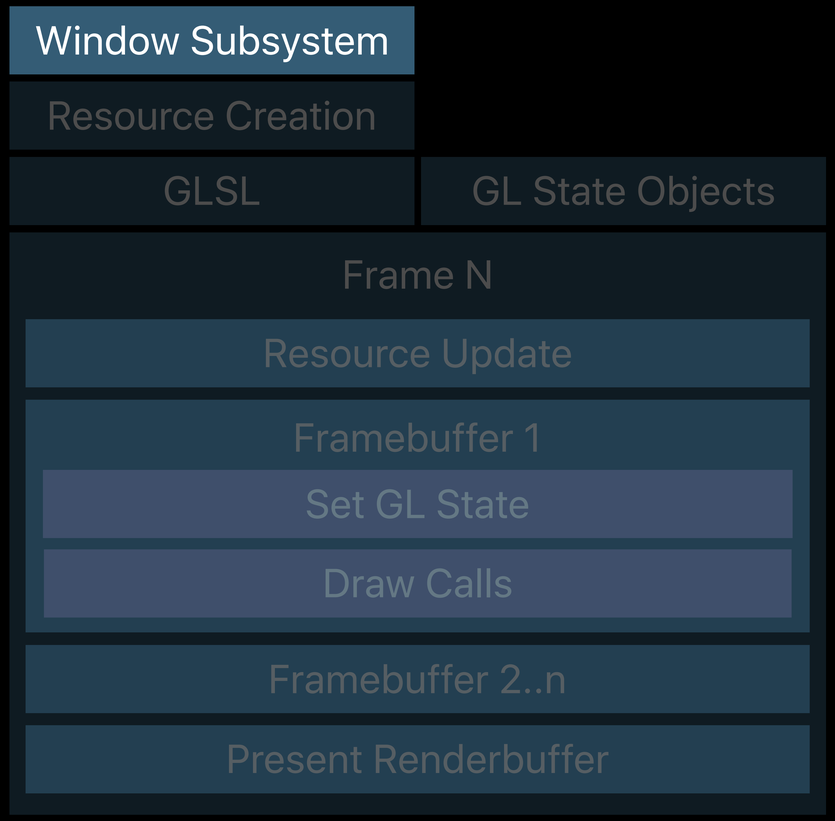
OpenGL



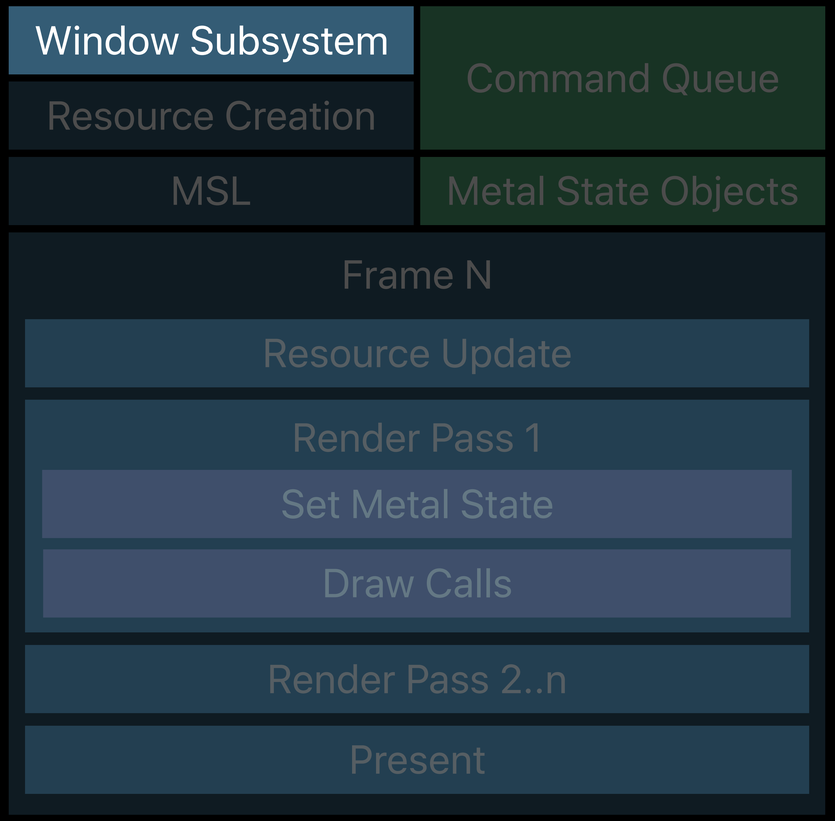
Metal

Life of a Graphics App

Recap



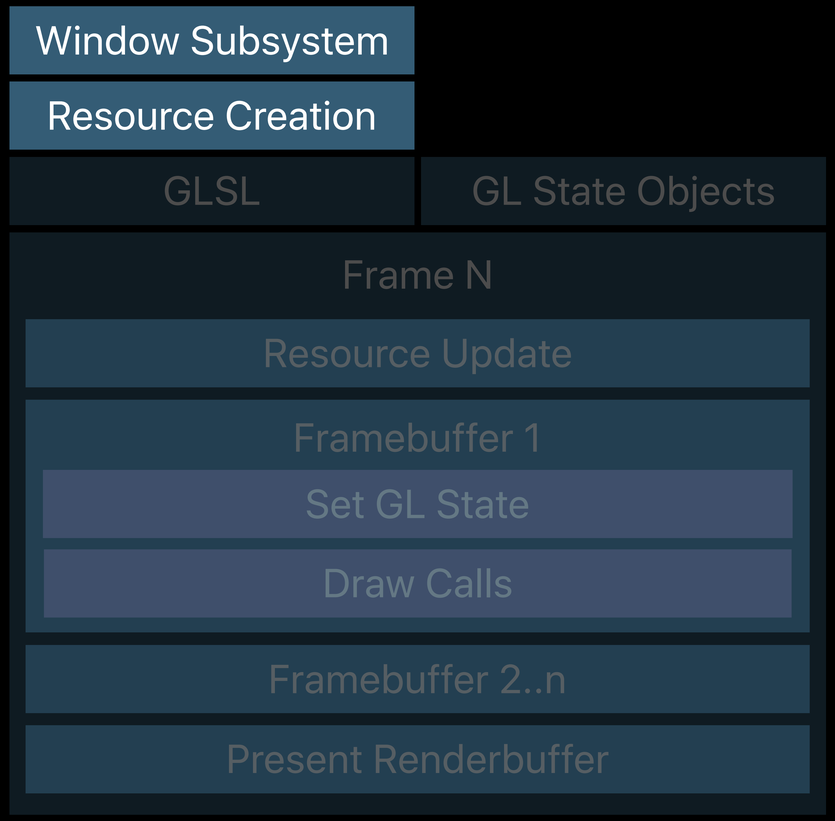
OpenGL



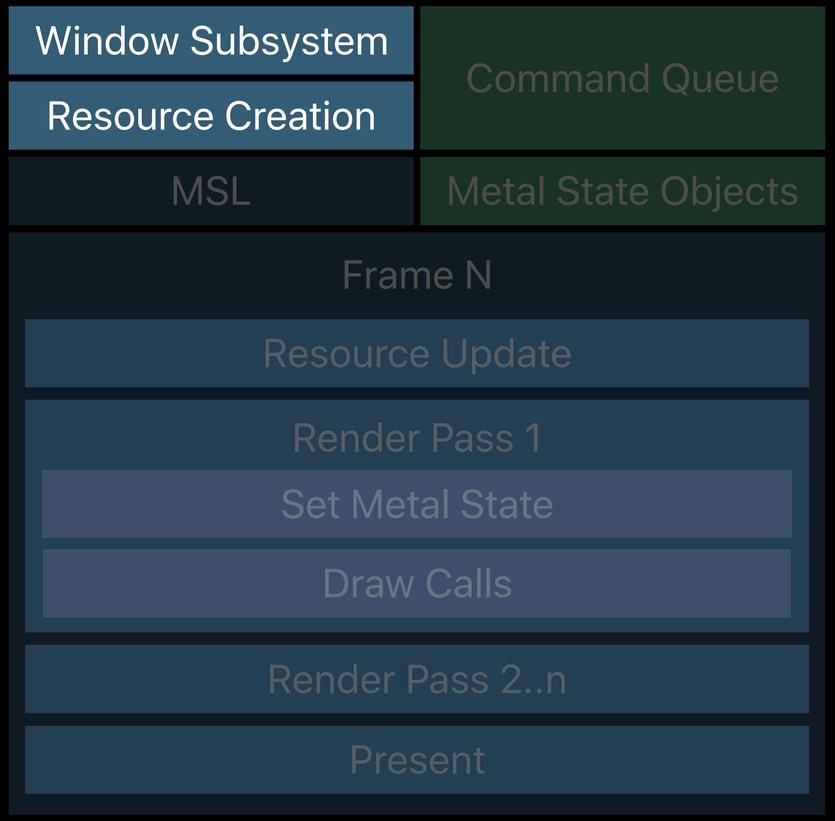
Metal

Life of a Graphics App

Recap



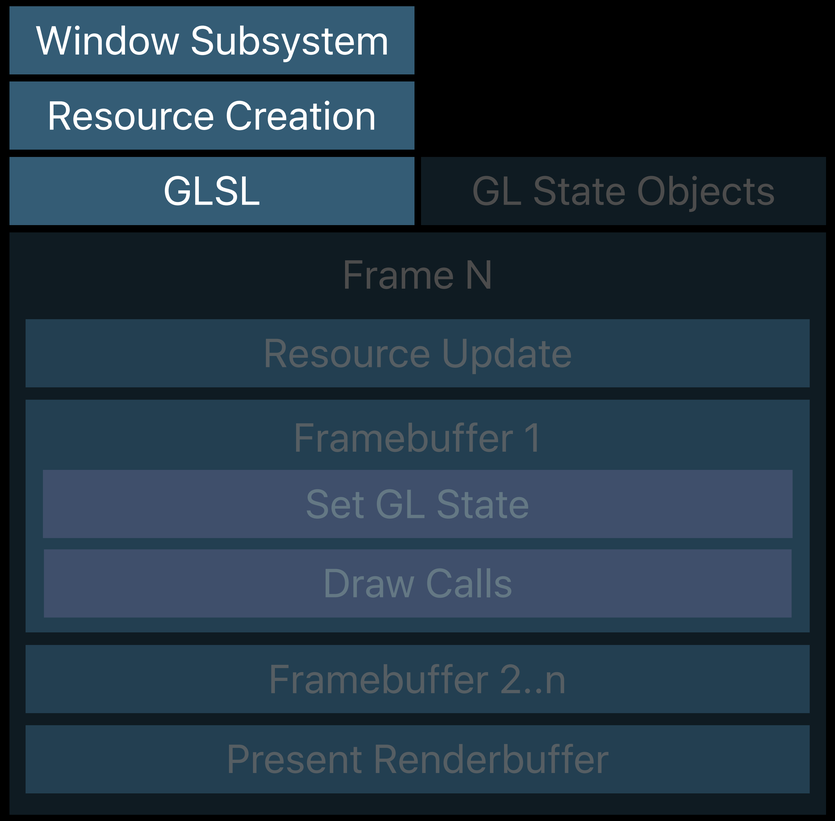
OpenGL



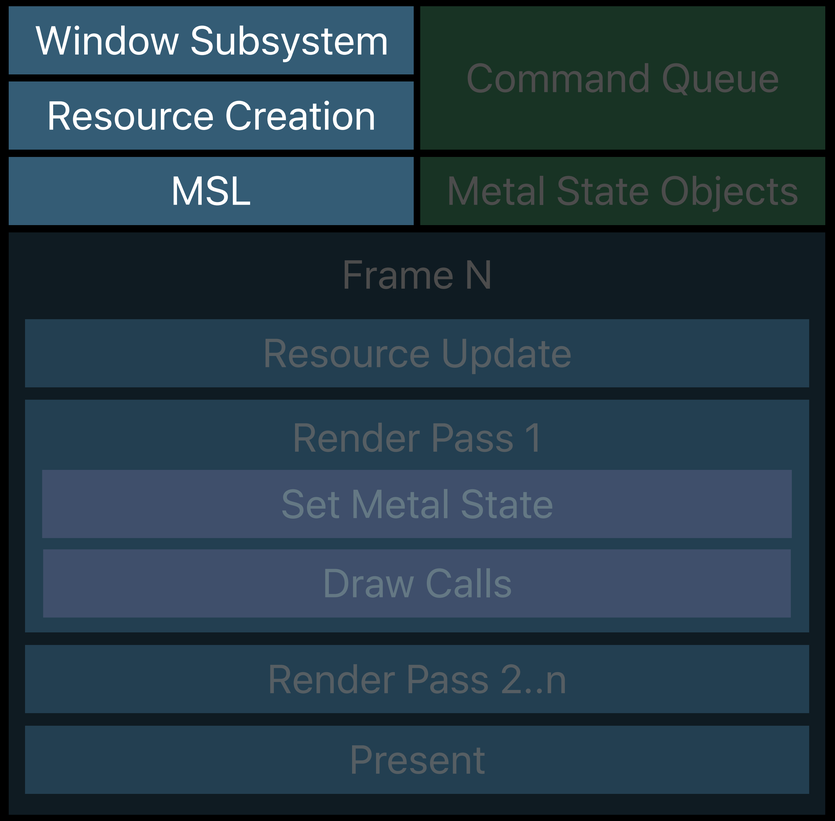
Metal

Life of a Graphics App

Recap



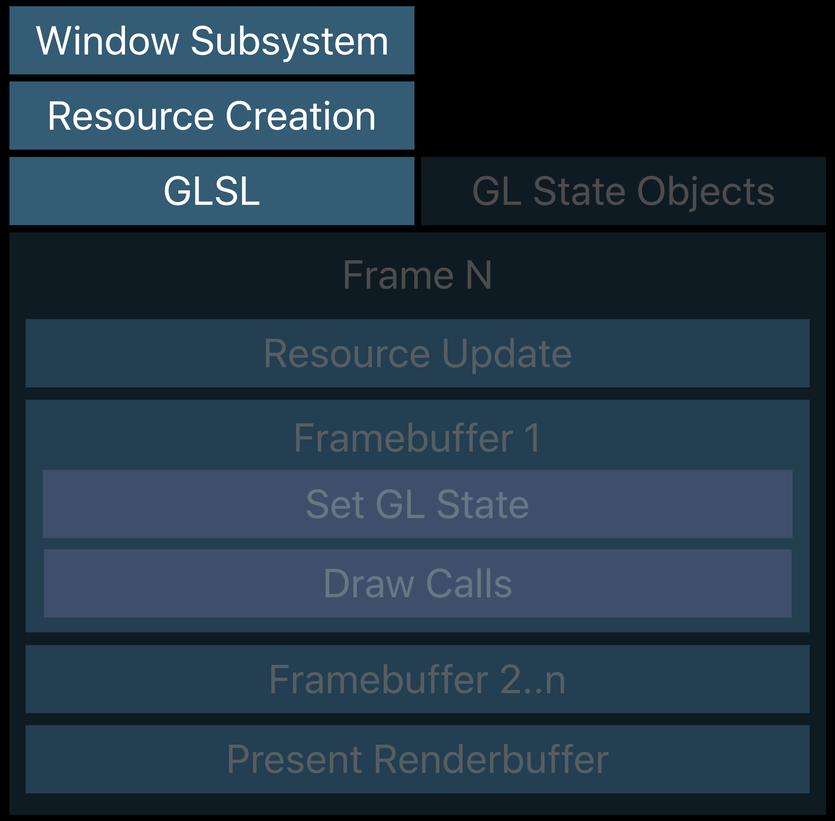
OpenGL



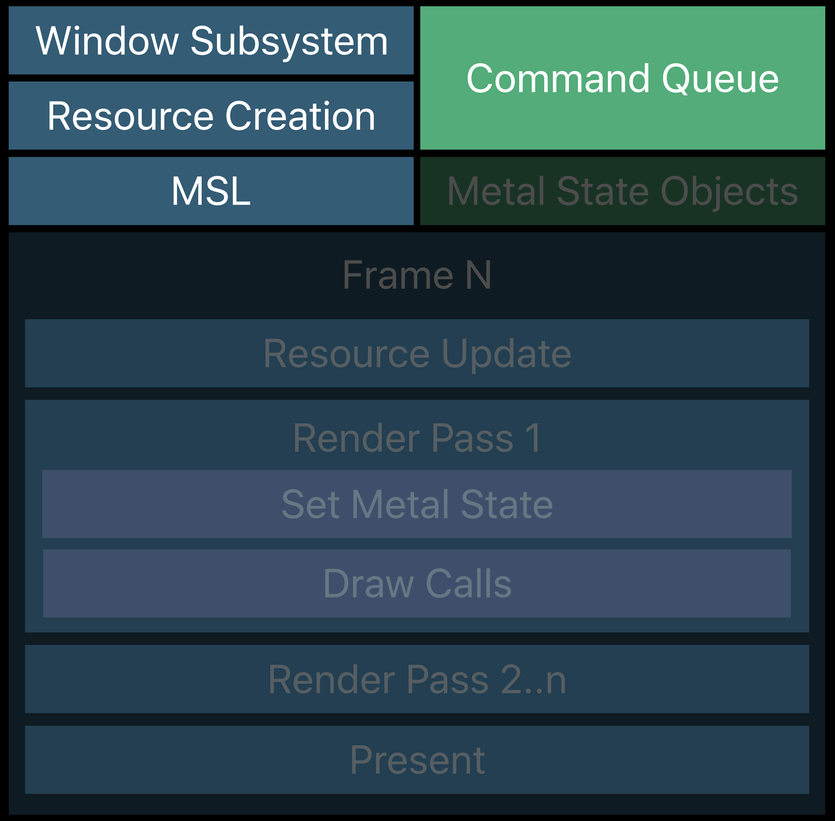
Metal

Life of a Graphics App

Recap



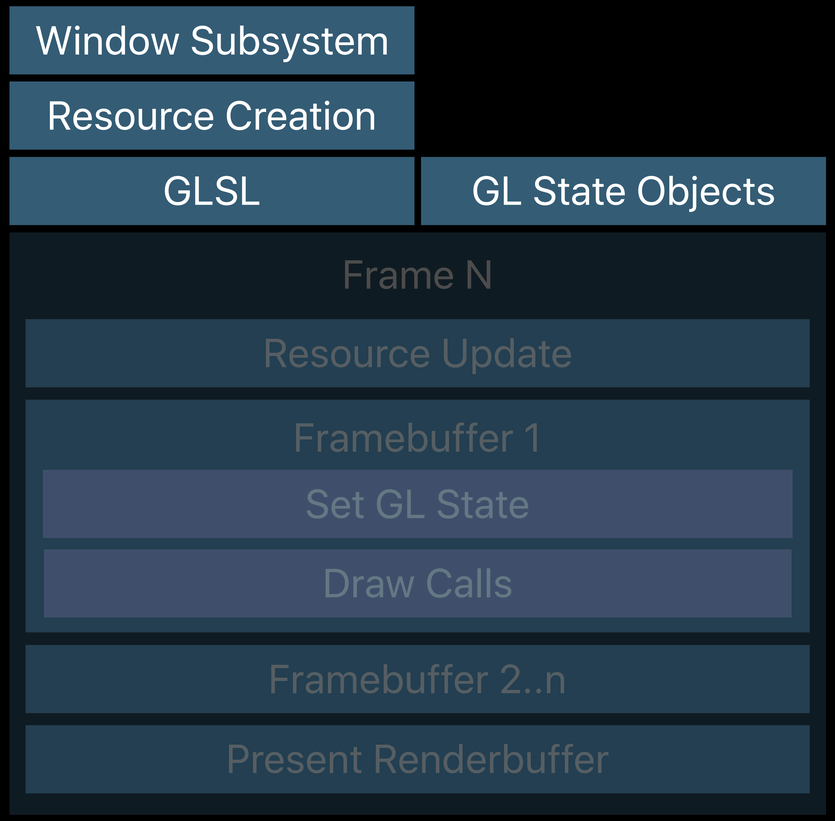
OpenGL



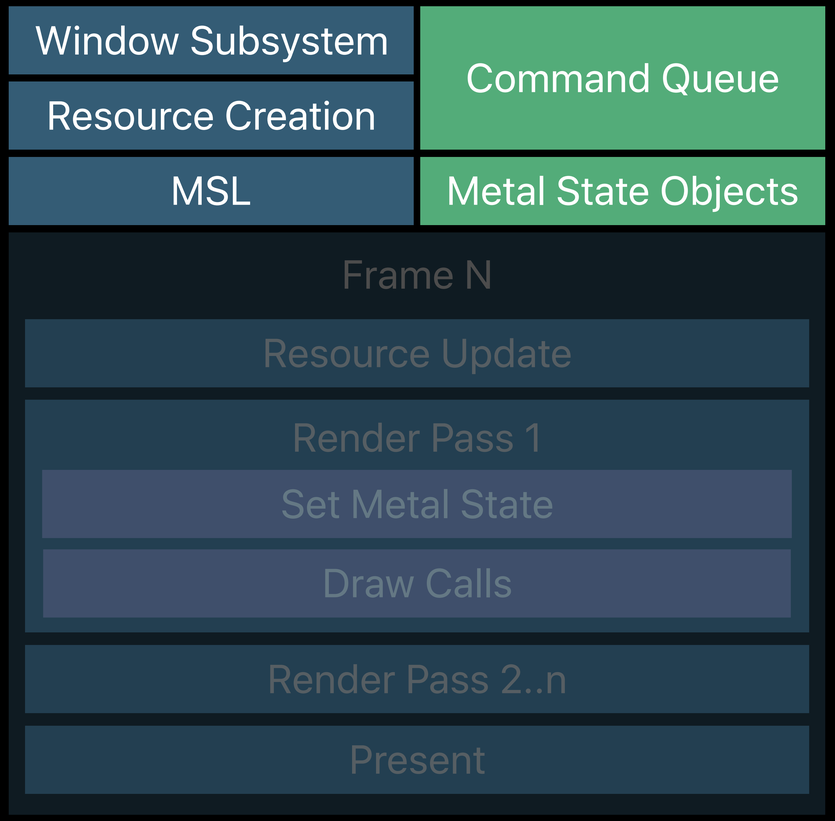
Metal

Life of a Graphics App

Recap



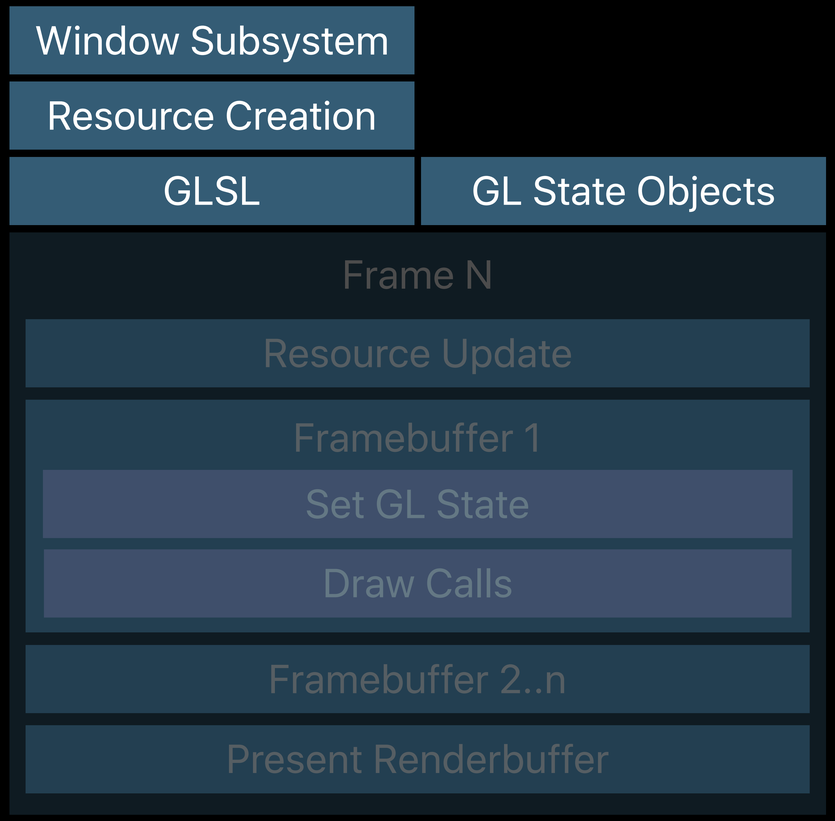
OpenGL



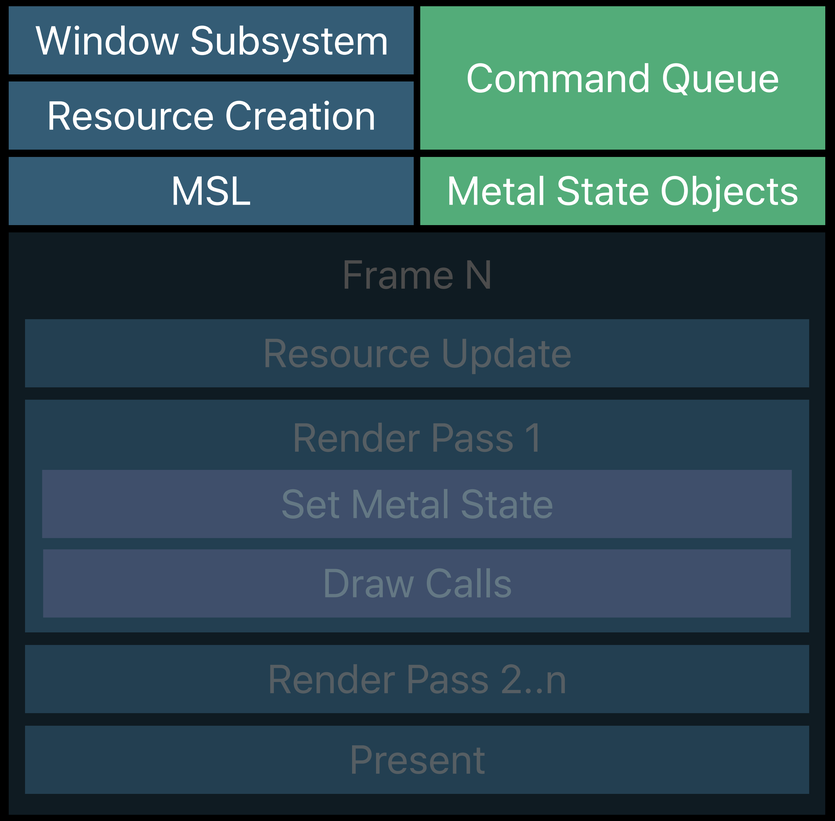
Metal

Life of a Graphics App

Recap



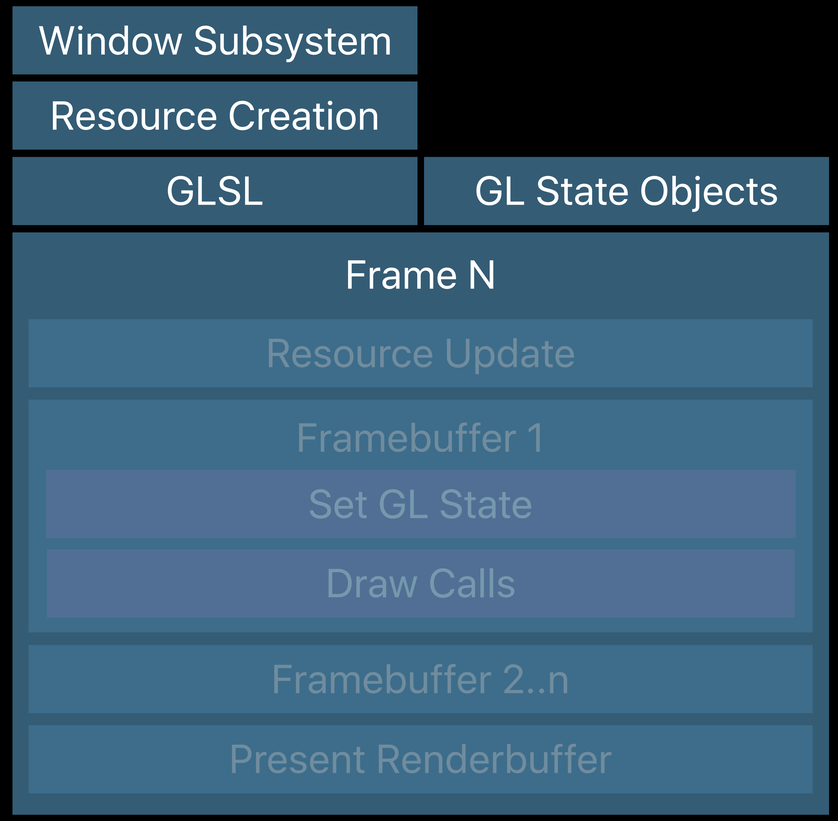
OpenGL



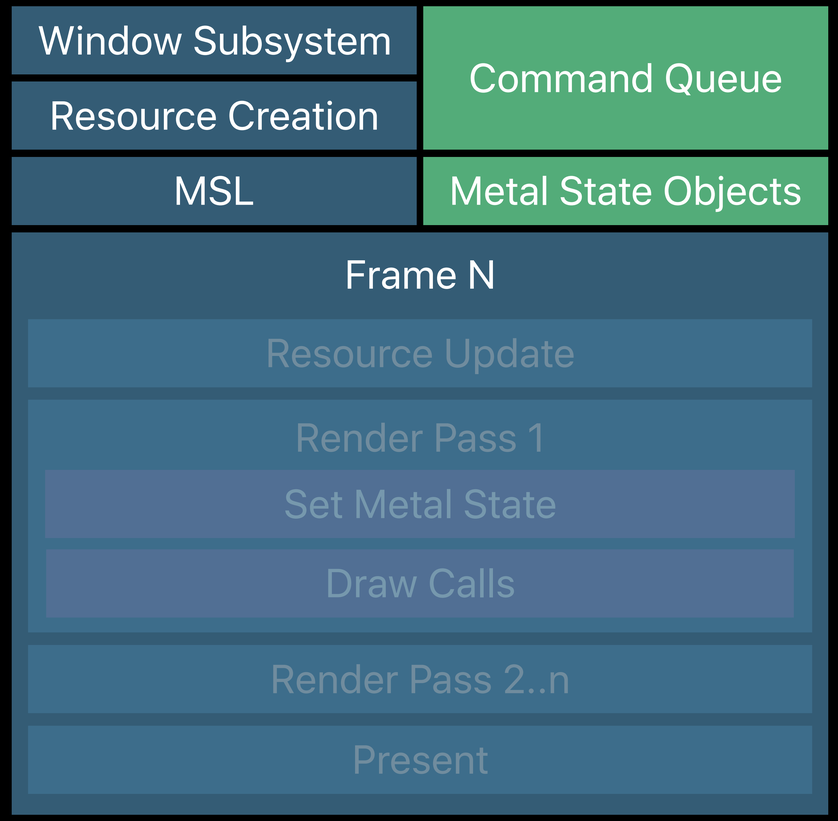
Metal

Life of a Graphics App

Recap



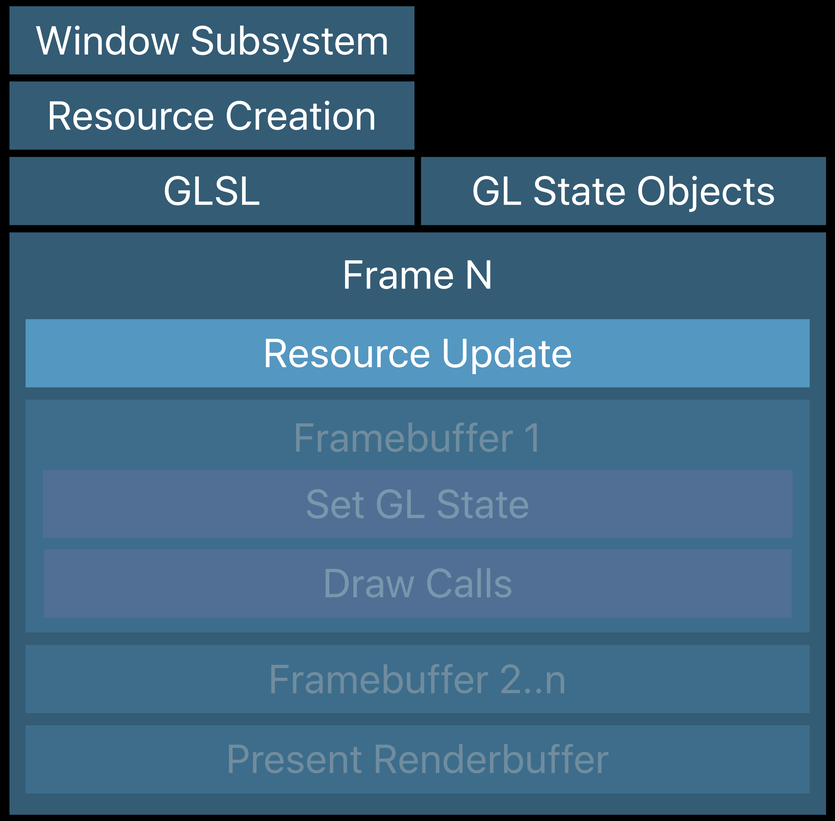
OpenGL



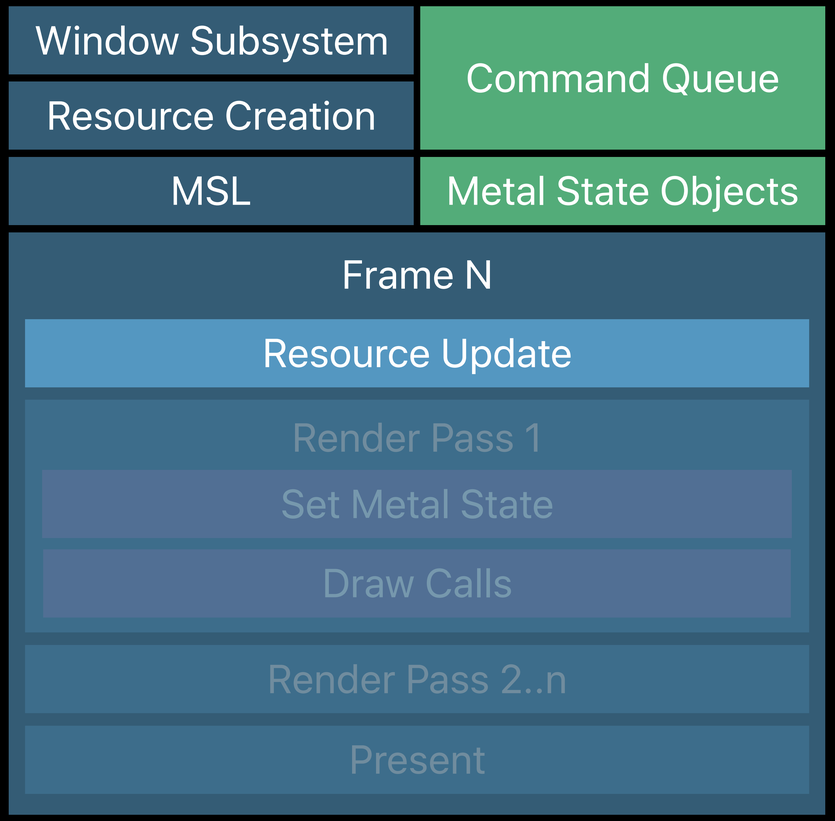
Metal

Life of a Graphics App

Recap



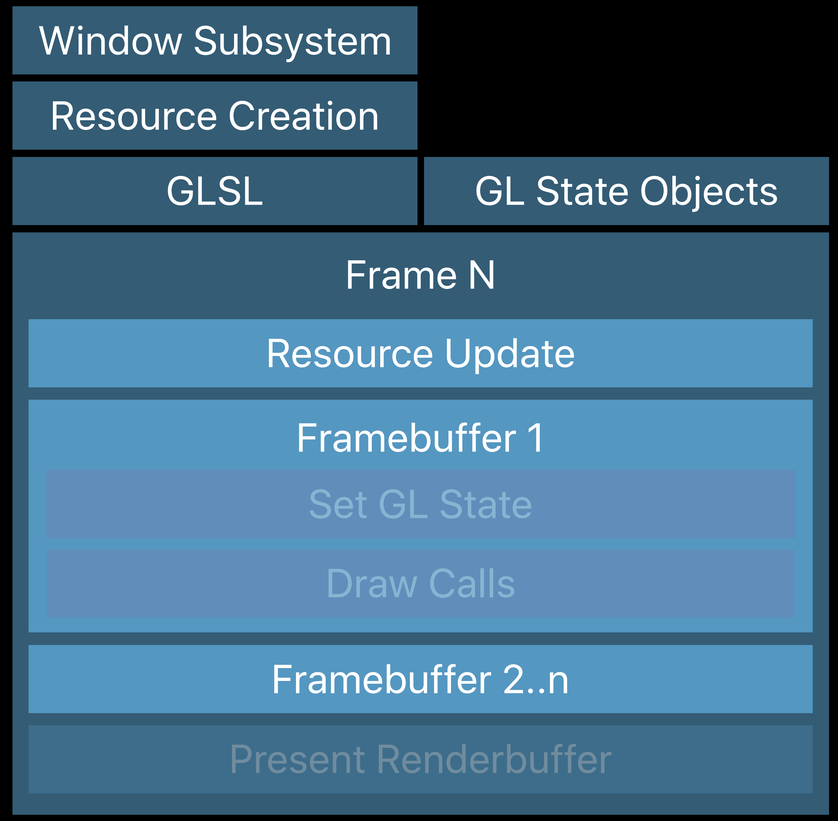
OpenGL



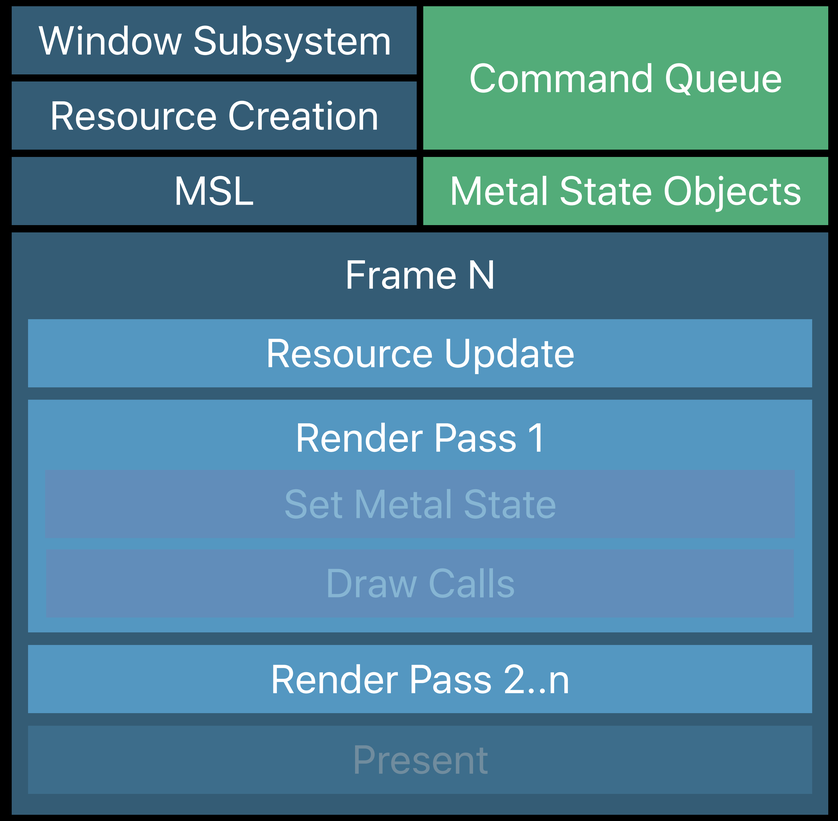
Metal

Life of a Graphics App

Recap



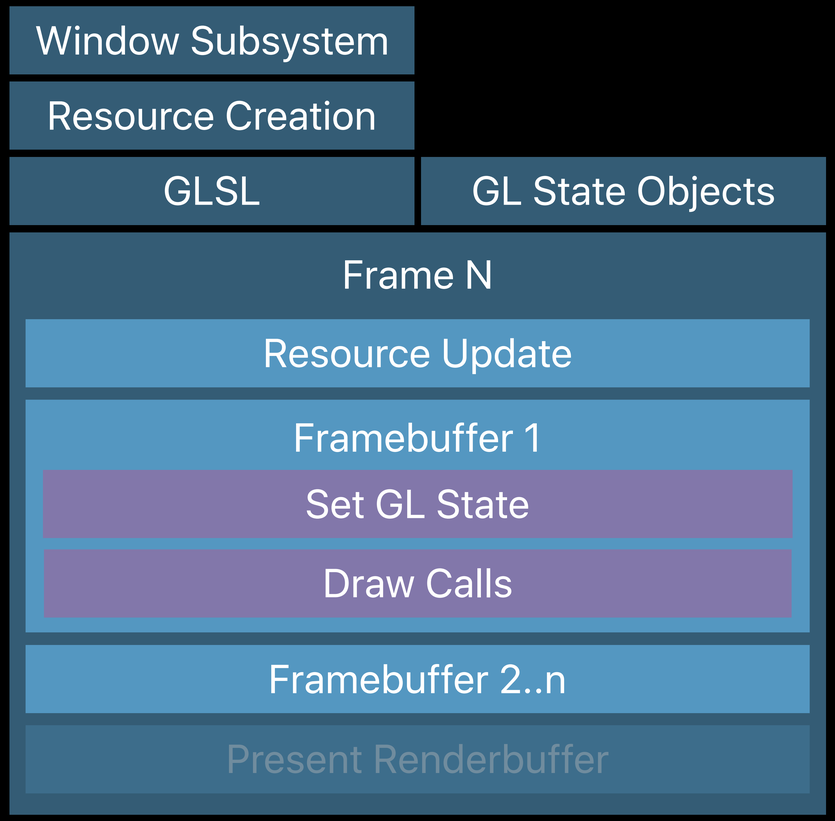
OpenGL



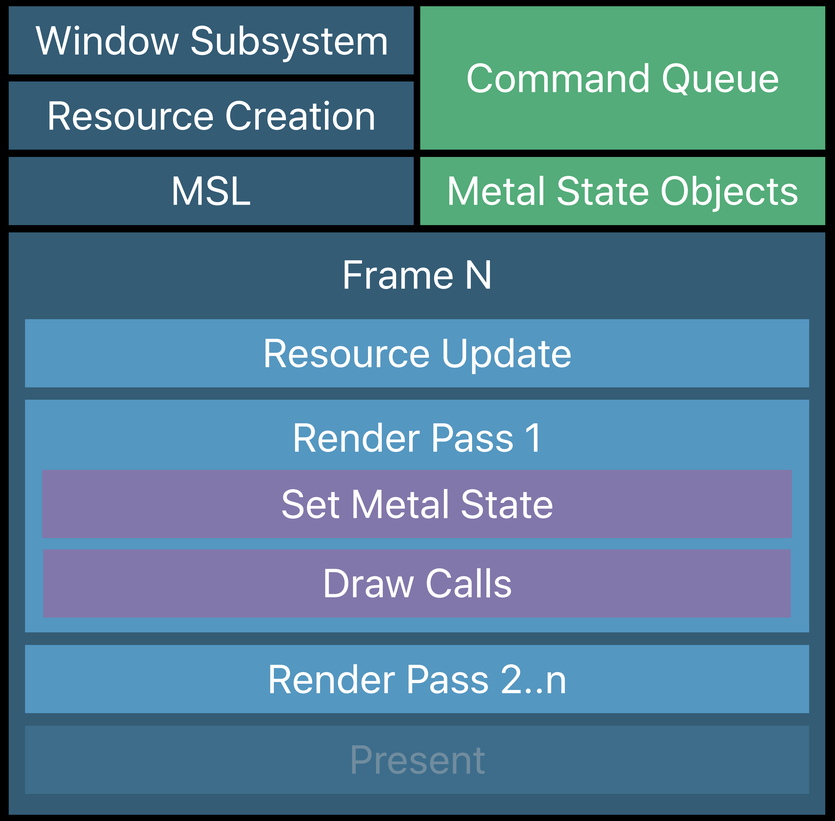
Metal

Life of a Graphics App

Recap



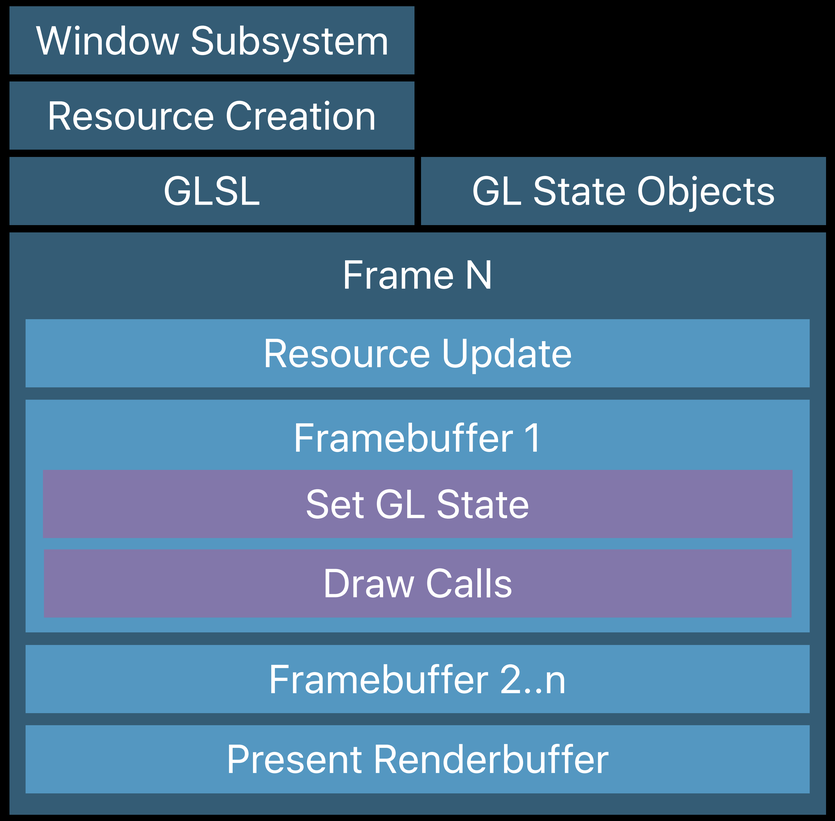
OpenGL



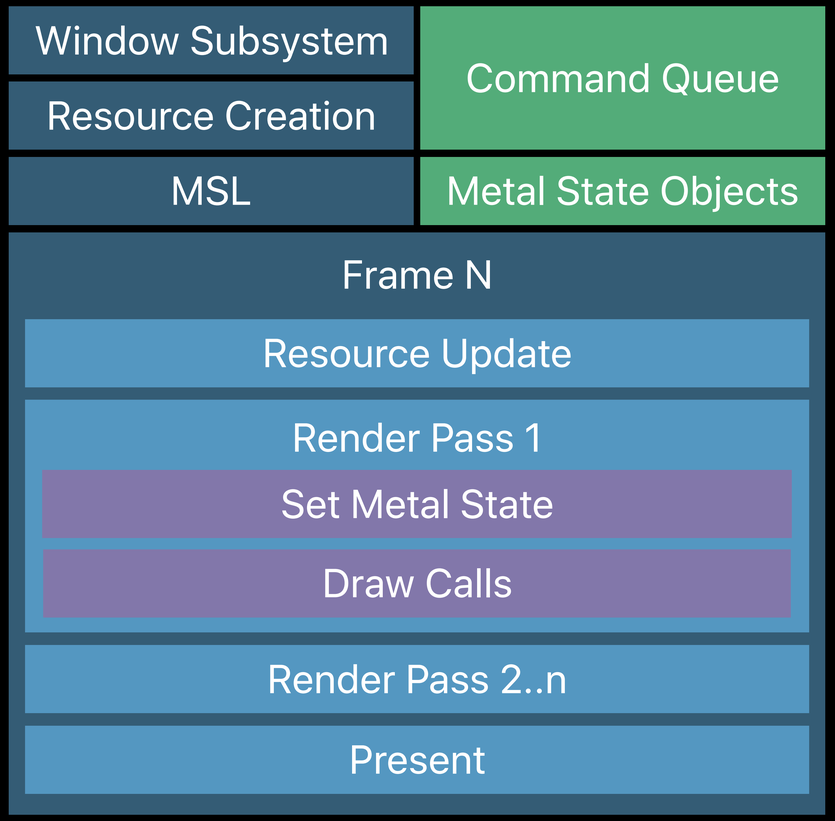
Metal

Life of a Graphics App

Recap



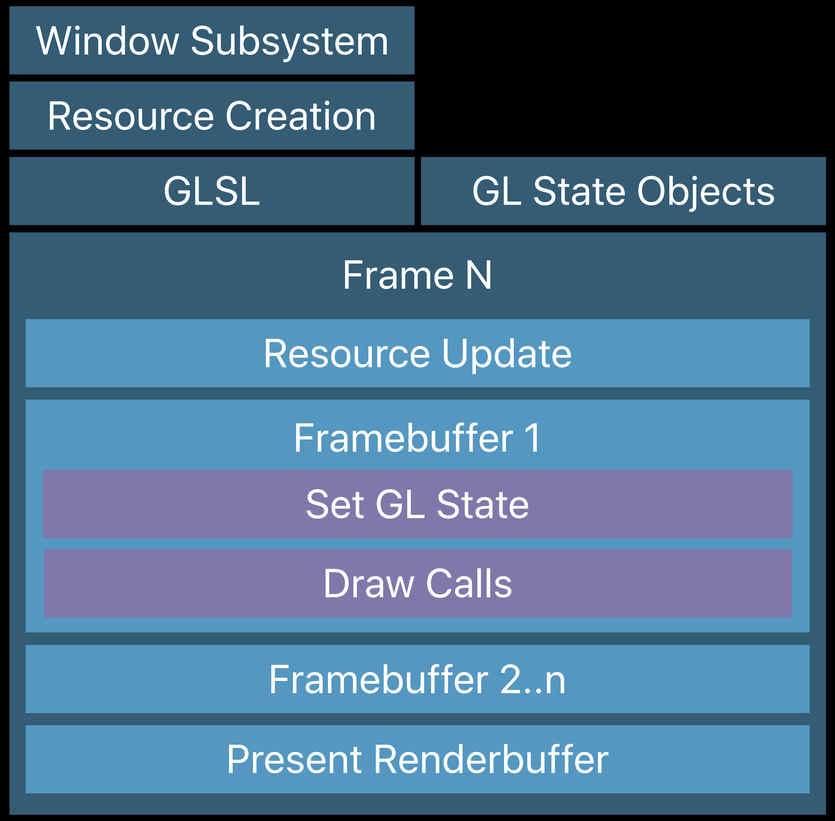
OpenGL



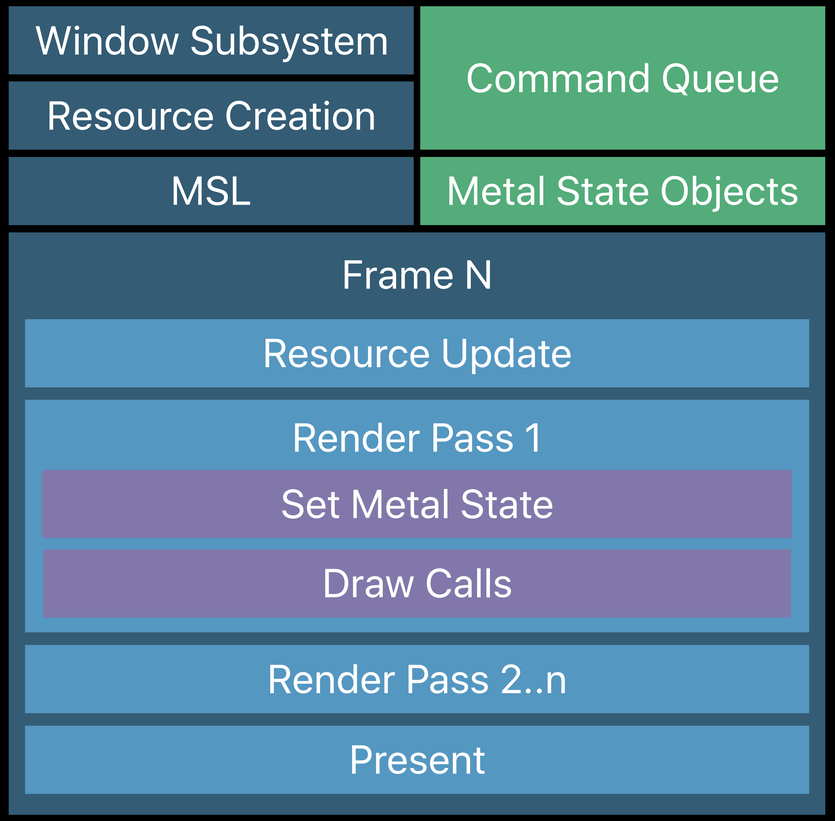
Metal

Life of a Graphics App

Recap



OpenGL



Metal

Metal Tools

Xcode

- Frame Capture
- Shader Debugger
- GPU Memory Viewer

Instruments

- Metal System Trace
- Game Performance Template

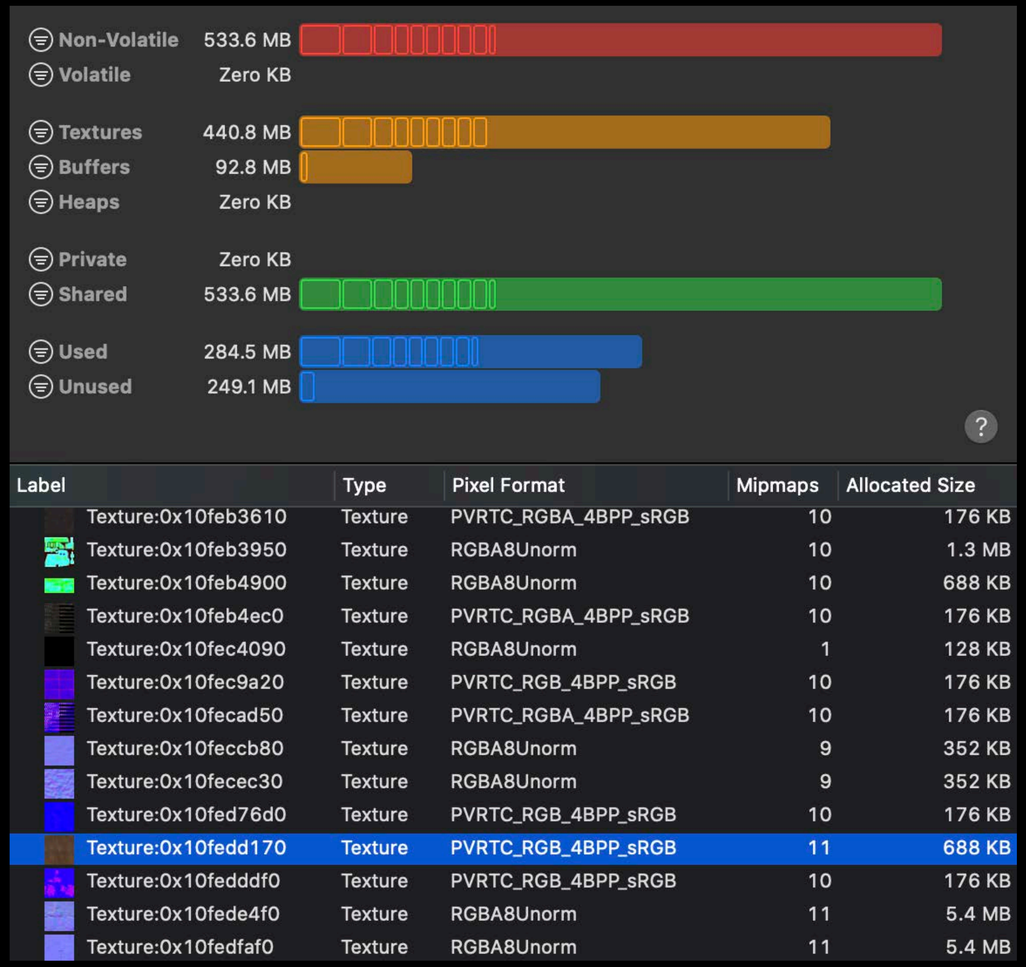
Metal Tools

Xcode

- Frame Capture
- Shader Debugger
- GPU Memory Viewer

Instruments

- Metal System Trace
- Game Performance Template



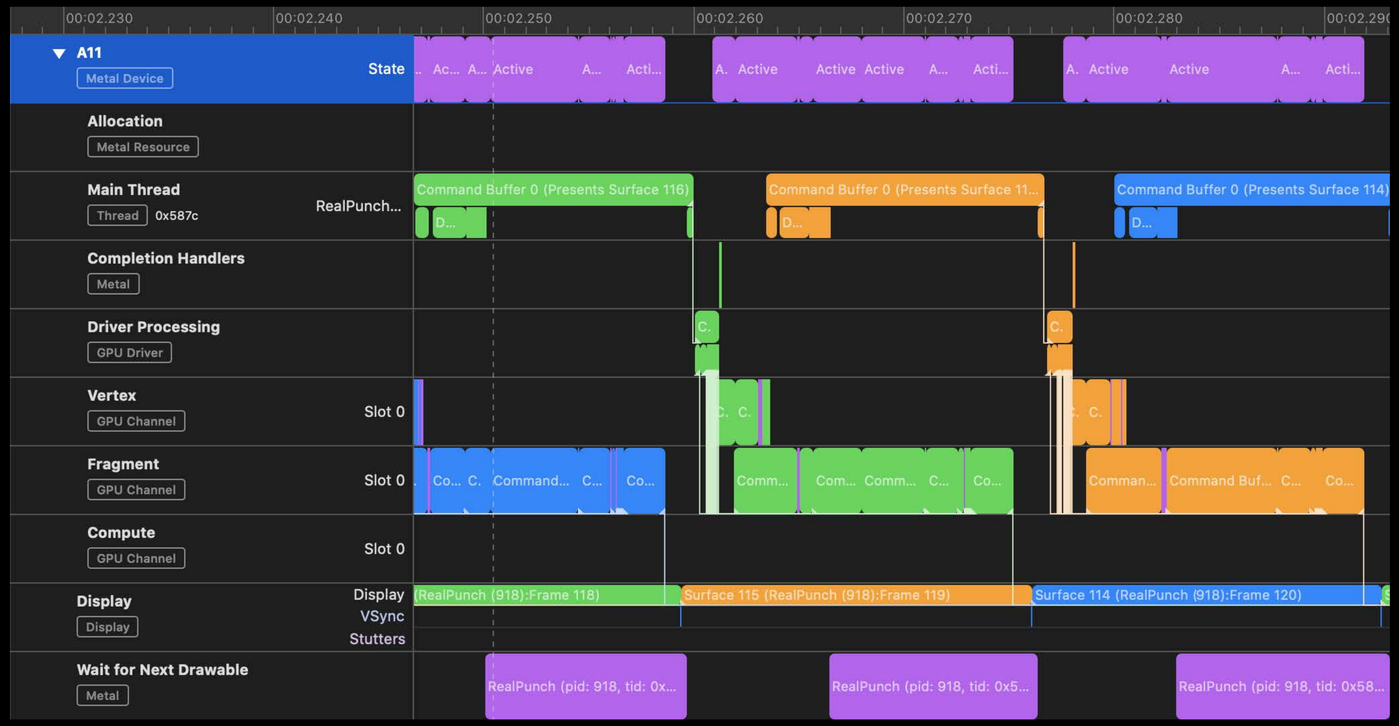
Metal Tools

Xcode

- Frame Capture
- Shader Debugger
- GPU Memory Viewer

Instruments

- Metal System Trace
- Game Performance Template



Metal Tools

Xcode

- Frame Capture
- Shader Debugger
- GPU Memory Viewer

Instruments

- Metal System Trace
- Game Performance template

Metal Tools

NEW

Xcode

- Frame Capture
- Shader Debugger
- GPU Memory Viewer

Instruments

- Metal System Trace
- Game Performance template

Metal Tools

NEW

Xcode

- Frame Capture
- Shader Debugger
- GPU Memory Viewer

Instruments

- Metal System Trace
- Game Performance template

Simulator supports Metal

Metal Simulator

New on macOS Catalina

NEW

Metal Simulator

New on macOS Catalina

NEW

Metal Simulator

New on macOS Catalina

NEW

MTLGPUFamilyApple2 feature set

More Information

developer.apple.com/wwdc19/611

Getting Most Out of the Simulator	Friday 9:00
Modern Rendering with Metal	WWDC 2019
Delivering Optimized Metal Apps and Games	WWDC 2019

More Information

developer.apple.com/wwdc19/611

Metal Lab	Friday 9:00


WWDC19