VR with Metal 2

Session 603

Rav Dhiraj, GPU Software

VR Support in macOS

Developing VR Apps

External GPU support

What is Virtual Reality?

Immersive 360° 3D experience

Direct object manipulation

Interactive environment

Motion tracking

Enabled with Metal 2

Enabled with Metal 2

Direct to display capability for VR Headsets

Enabled with Metal 2

Direct to display capability for VR Headsets

Targeted features for VR

Enabled with Metal 2

Direct to display capability for VR Headsets

Targeted features for VR

Foundational support for External GPUs

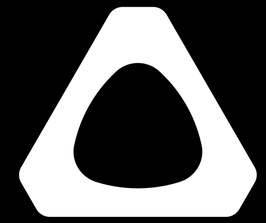
Platform Support

NEW

HTC Vive Head Mounted Display

Valve SteamVR runtime

Valve OpenVR Framework



VIVE™

VALVE®

VR Compositor

Image warping for HMD optics

VR Compositor

Image warping for HMD optics

Building a VR App

Building a VR App

Two options

Building a VR App

Two options

Game engine with VR support

- Hides VR compositor complexity
- Familiar toolchain

Building a VR App

Two options

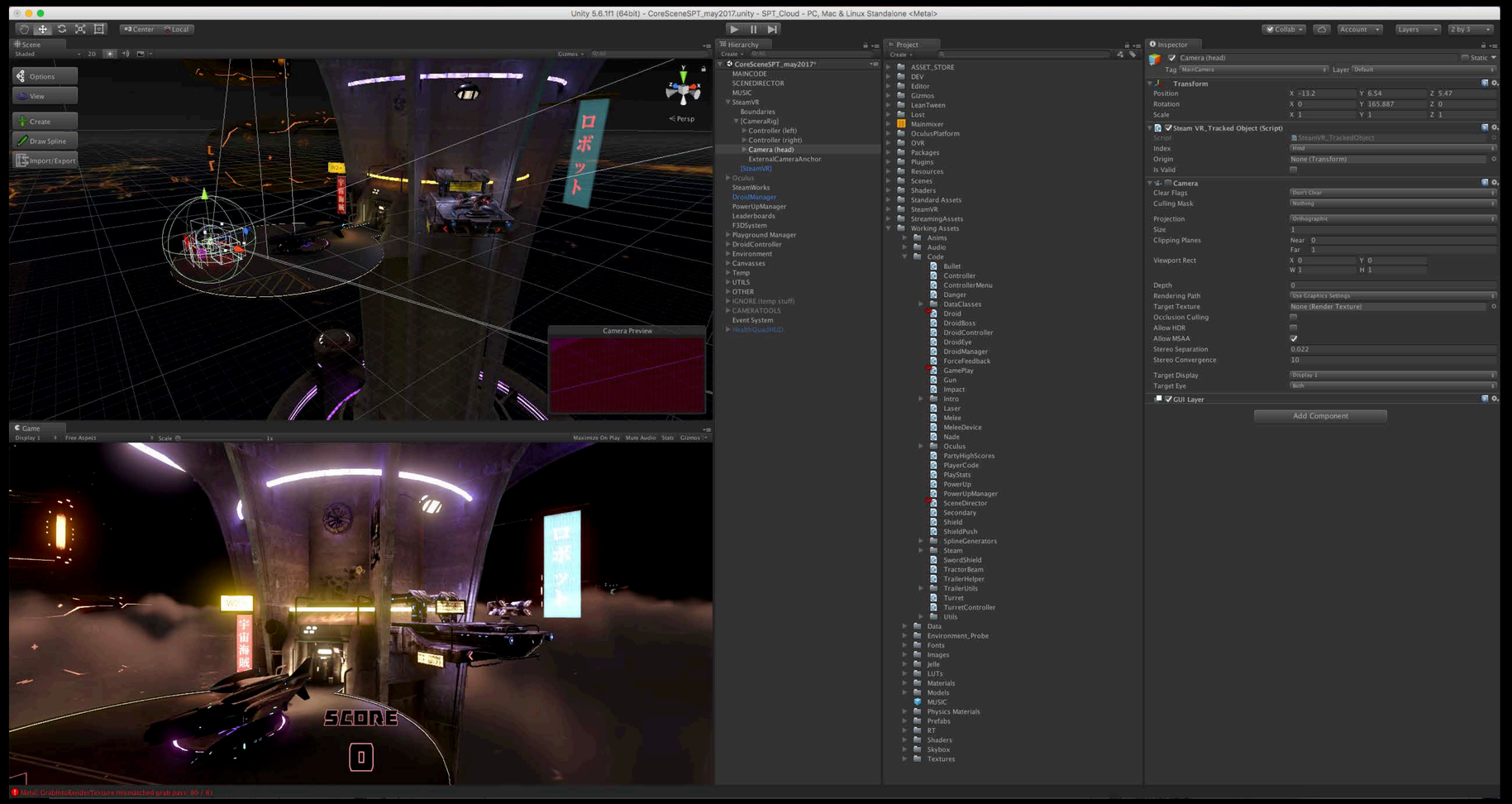
Game engine with VR support

- Hides VR compositor complexity
- Familiar toolchain

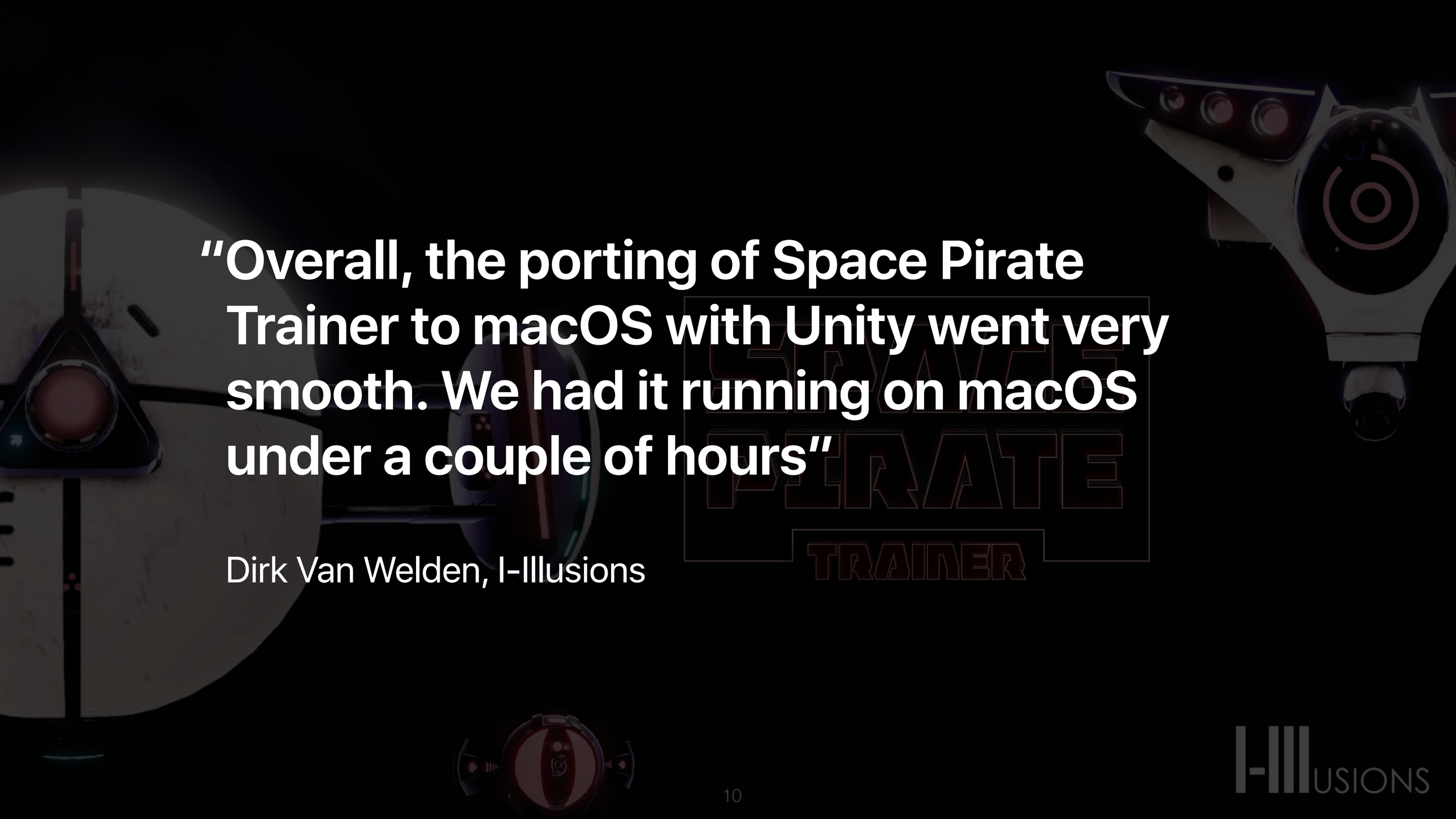
Build your own native VR app

- Full control of rendering and synchronization

UNREAL
ENGINE


Unity 5.6.1f1 (64bit) - CoreSceneSPT_may2017.unity - SPT_Cloud - PC, Mac & Linux Standalone <Metal>
Scene
Options
View
Create
Draw Spline
Import/Export
Persp
Camera Preview
Game
SCORE
0
Hierarchy
Project
Inspector
Camera (head)
Steam VR_Tracked Object (Script)
Camera
GUI Layer
Add Component


SPACE
PIRATE
TRAINER



“Overall, the porting of Space Pirate Trainer to macOS with Unity went very smooth. We had it running on macOS under a couple of hours”

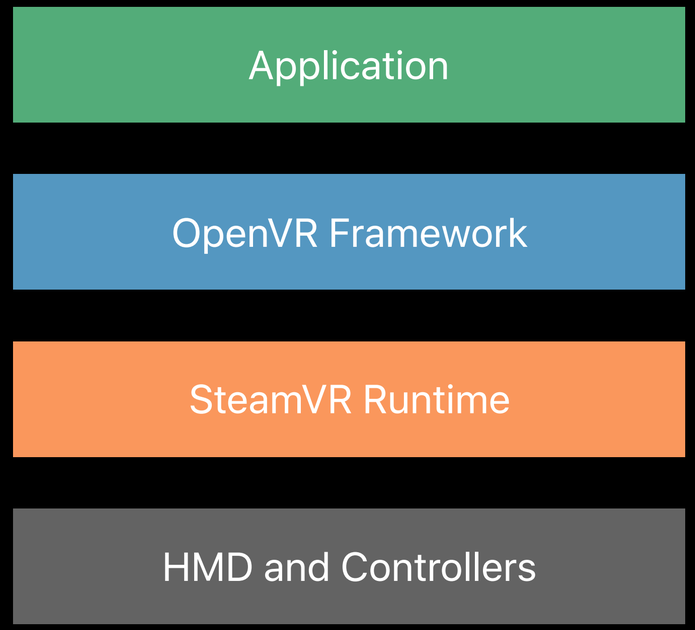
Dirk Van Welden, I-Illusions

Native SteamVR App

Custom app built using the OpenVR Framework

Binaries and documentation available on GitHub

macOS sample code available soon



VR App Building 101

Overview of VR development

macOS platform specifics

Anatomy of a VR frame

VR best practices

VR App Building 101

Overview of VR development

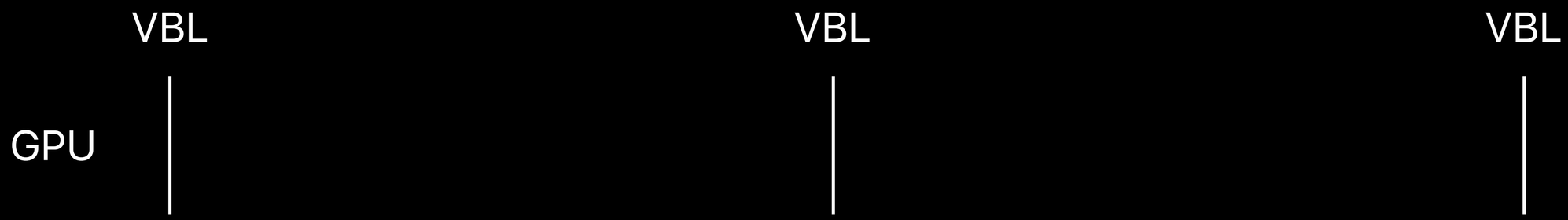
macOS platform specifics

Anatomy of a VR frame

VR best practices

Traditional Workload

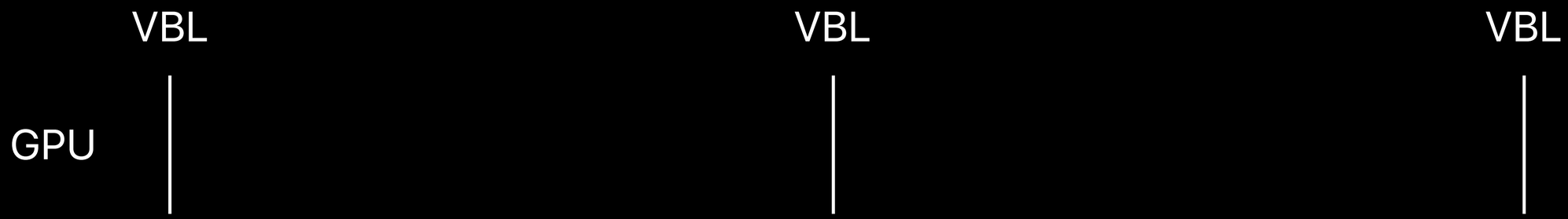
Non-VR



Traditional Workload

Non-VR

60 fps target

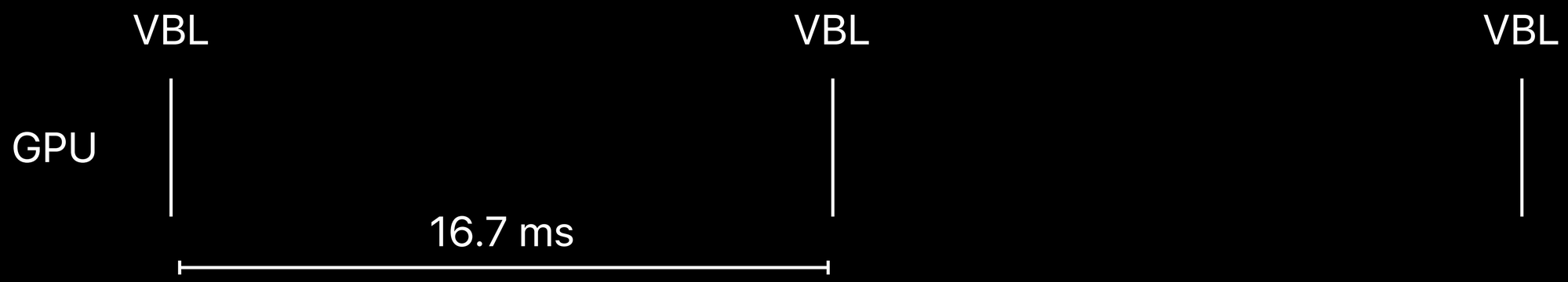


Traditional Workload

Non-VR

60 fps target

16.7 ms available frame time



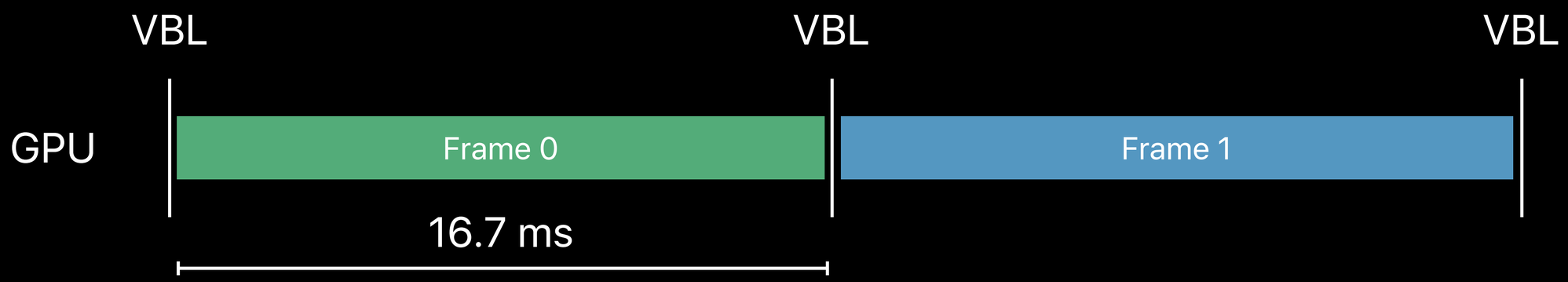
Traditional Workload

Non-VR

60 fps target

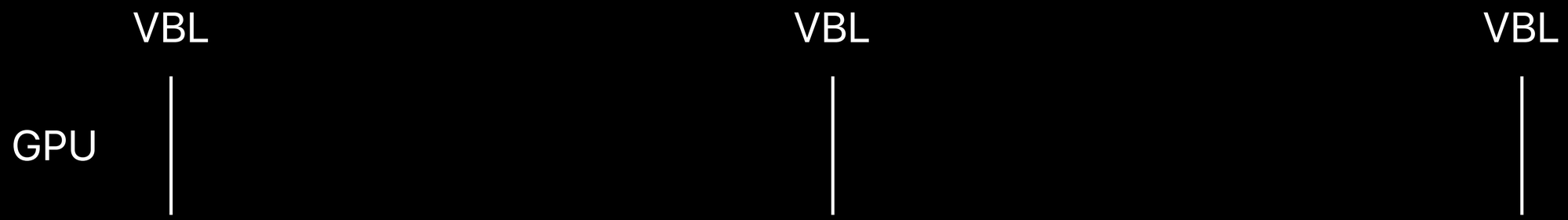
16.7 ms available frame time

Entire frame time available for GPU work



VR Workload

Frame time differences

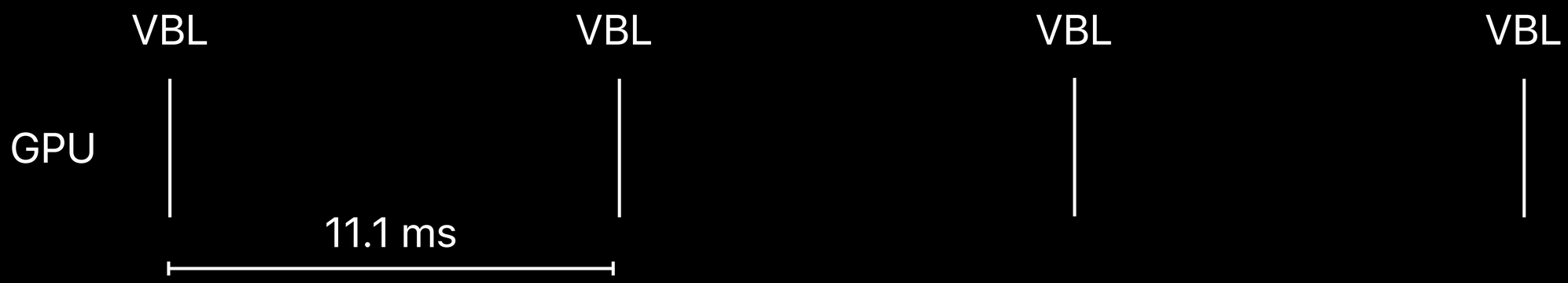


VR Workload

Frame time differences

90 fps target

- Reduces frame time to 11.1 ms



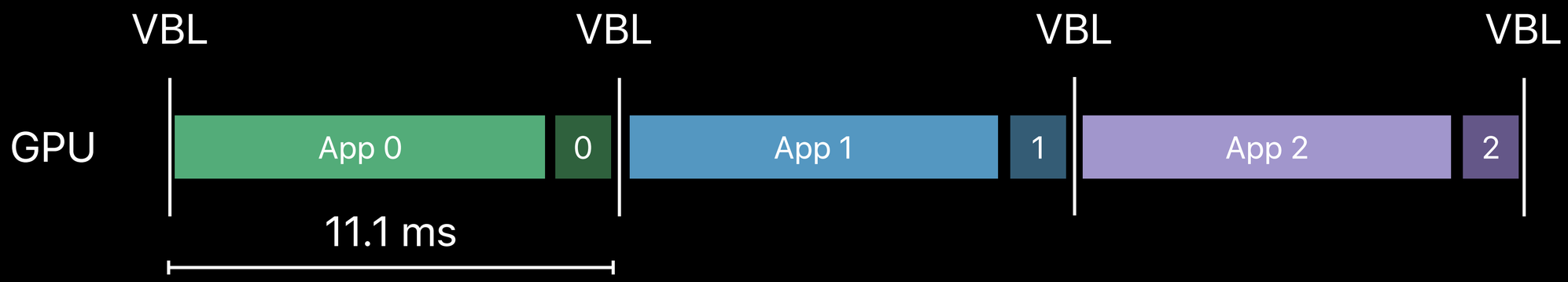
VR Workload

Frame time differences

90 fps target

- Reduces frame time to 11.1 ms

Additional GPU work by VR compositor



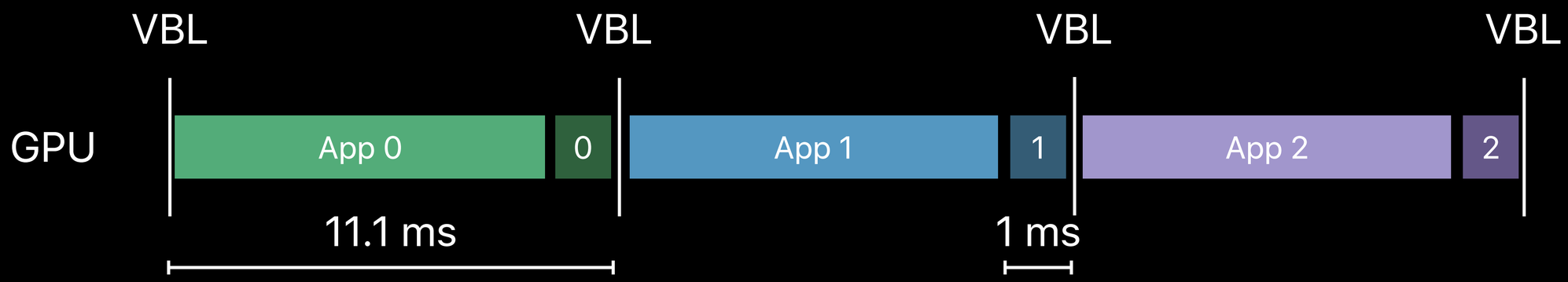
VR Workload

Frame time differences

90 fps target

- Reduces frame time to 11.1 ms

Additional GPU work by VR compositor



VR Workload

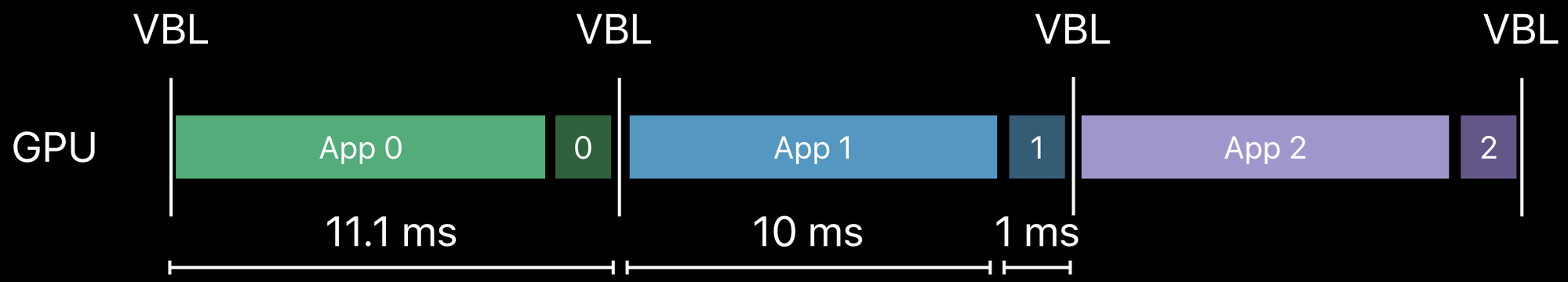
Frame time differences

90 fps target

- Reduces frame time to 11.1 ms

Additional GPU work by VR compositor

- ~10ms frame time budget for your app



VR Workload

More work every frame

VR Workload

More work every frame

Rendering the scene twice

- Left and right eye

VR Workload

More work every frame

Rendering the scene twice

- Left and right eye

Rendering at higher resolution

- HTC Vive headset: 2160x1200
- Commonly supersample at 1.2 to 1.4x when rendering

VR App Building 101

Overview of VR development

macOS platform specifics

Anatomy of a VR frame

VR best practices

Direct to Display

NEW

Low latency path that bypasses the Window Server

VR Compositor presents directly to the HMD

Direct to Display

NEW

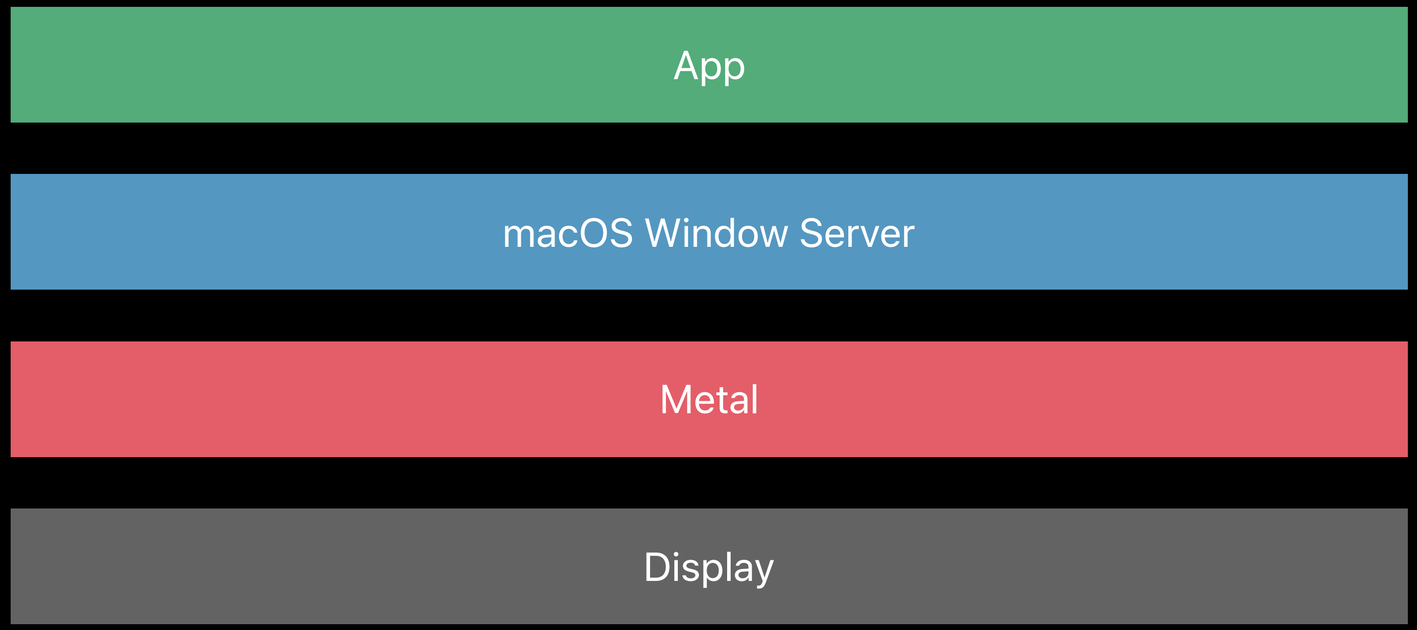
Low latency path that bypasses the Window Server

VR Compositor presents directly to the HMD

VR headsets not exposed as regular displays

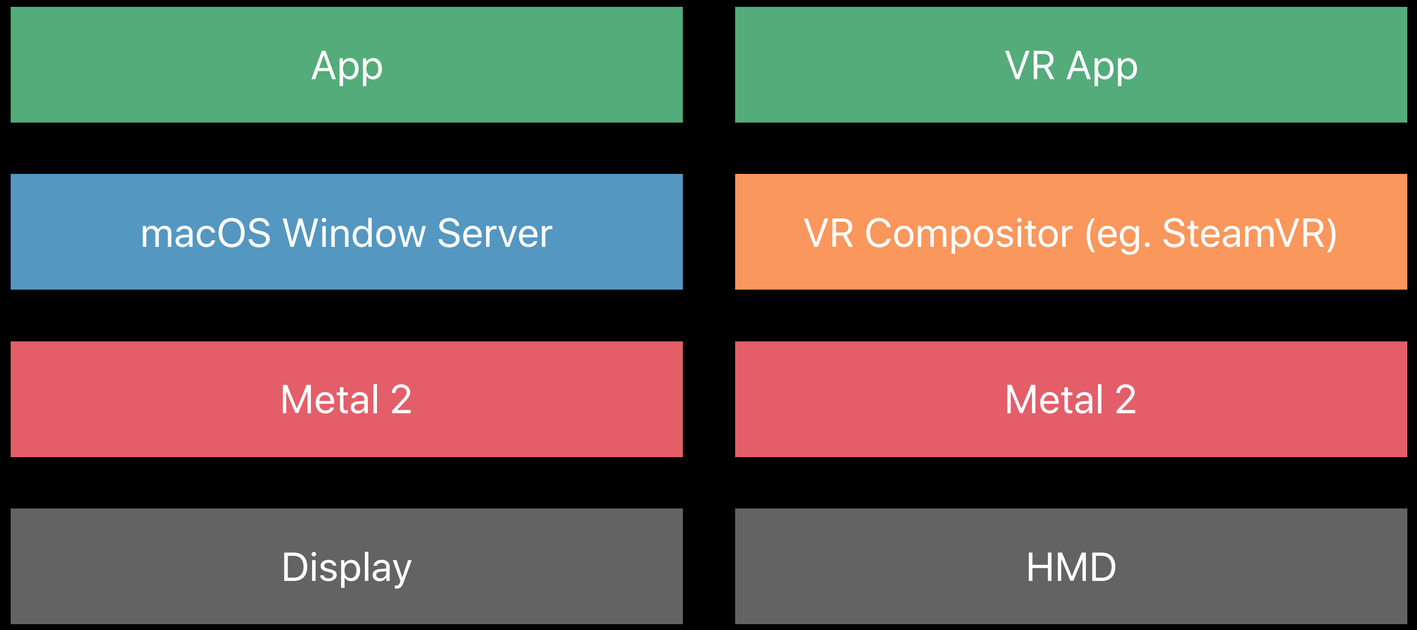
Present to Display

macOS 10.12



Present to Headset

macOS 10.13 with Metal 2



Selecting a Metal Device

Selecting a Metal Device

SteamVR selects the device attached to the VR Headset

Selecting a Metal Device

SteamVR selects the device attached to the VR Headset

Apps should select the same Metal device

Selecting a Metal Device

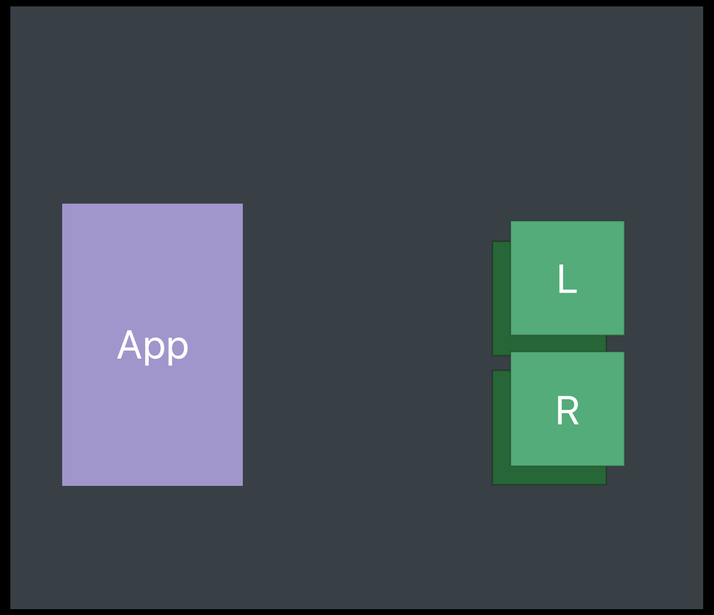
SteamVR selects the device attached to the VR Headset

Apps should select the same Metal device

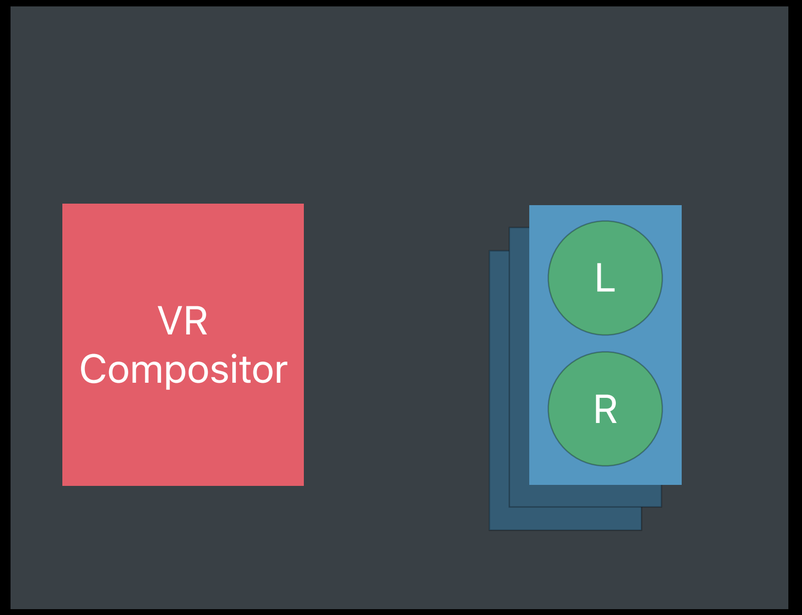
```
id<MTLDevice> vrDevice = nil;
vrSystem->GetOutputDevice((uint64_t*)&vrDevice, vr::TextureType_IOSurface);
```

Managing Surfaces

App

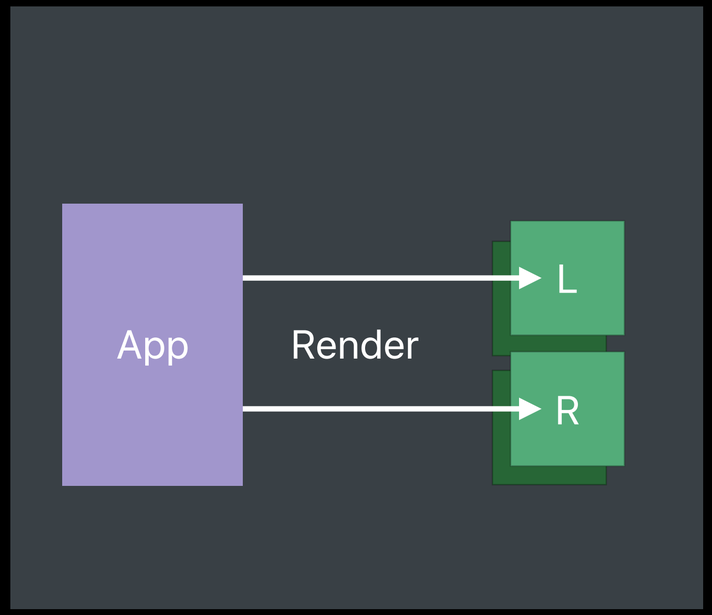


VR Compositor

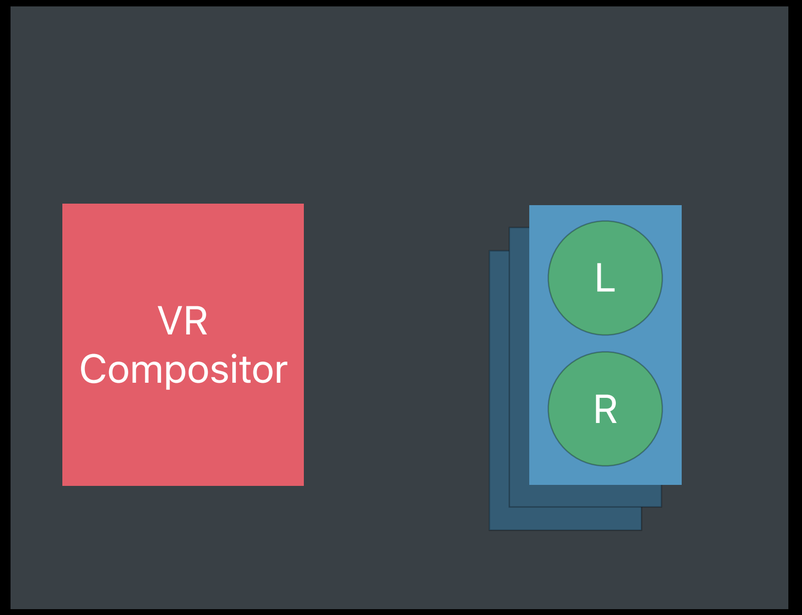


Managing Surfaces

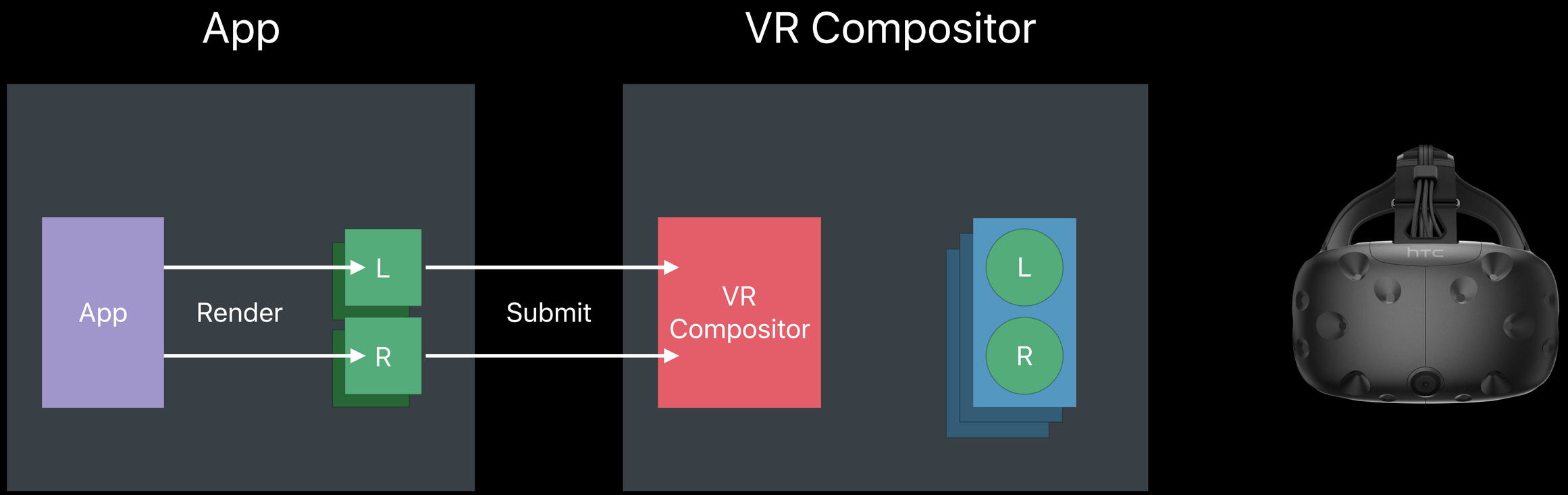
App



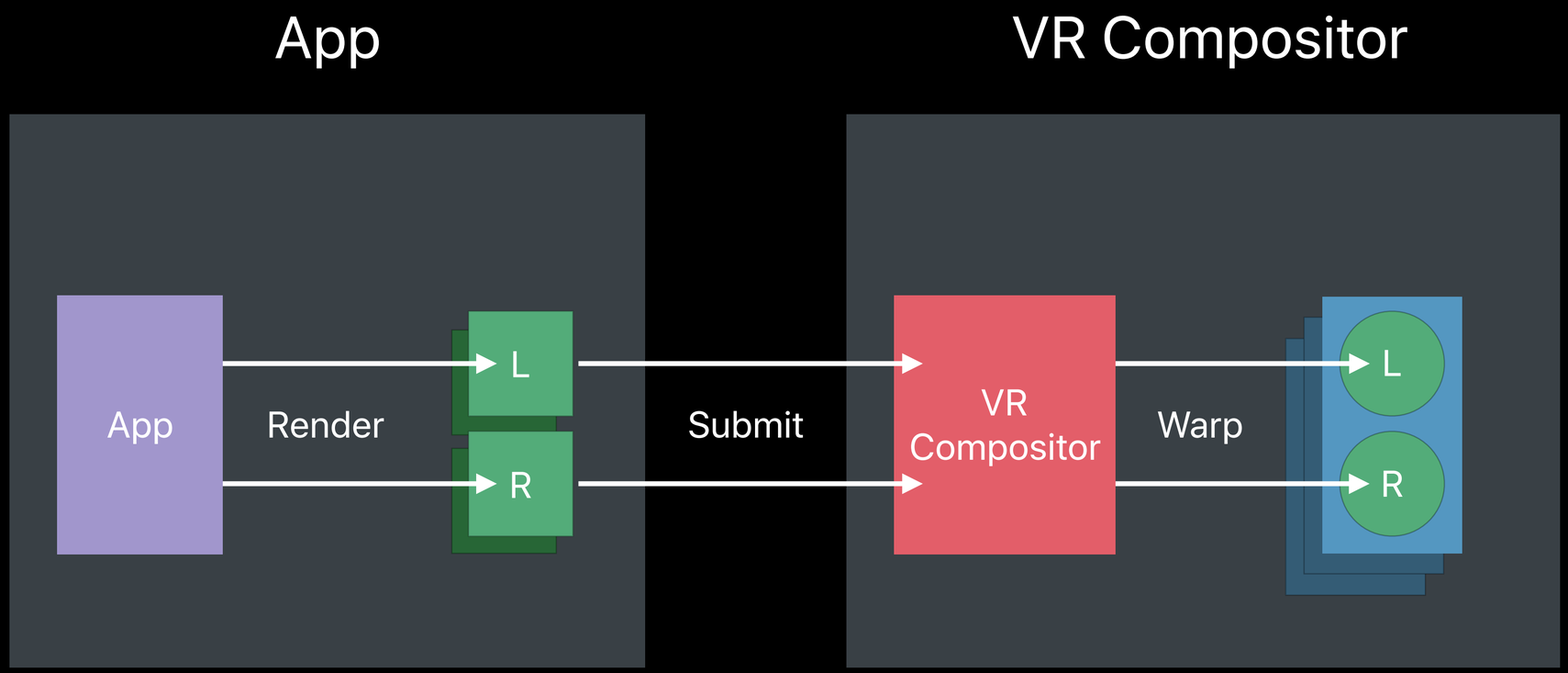
VR Compositor



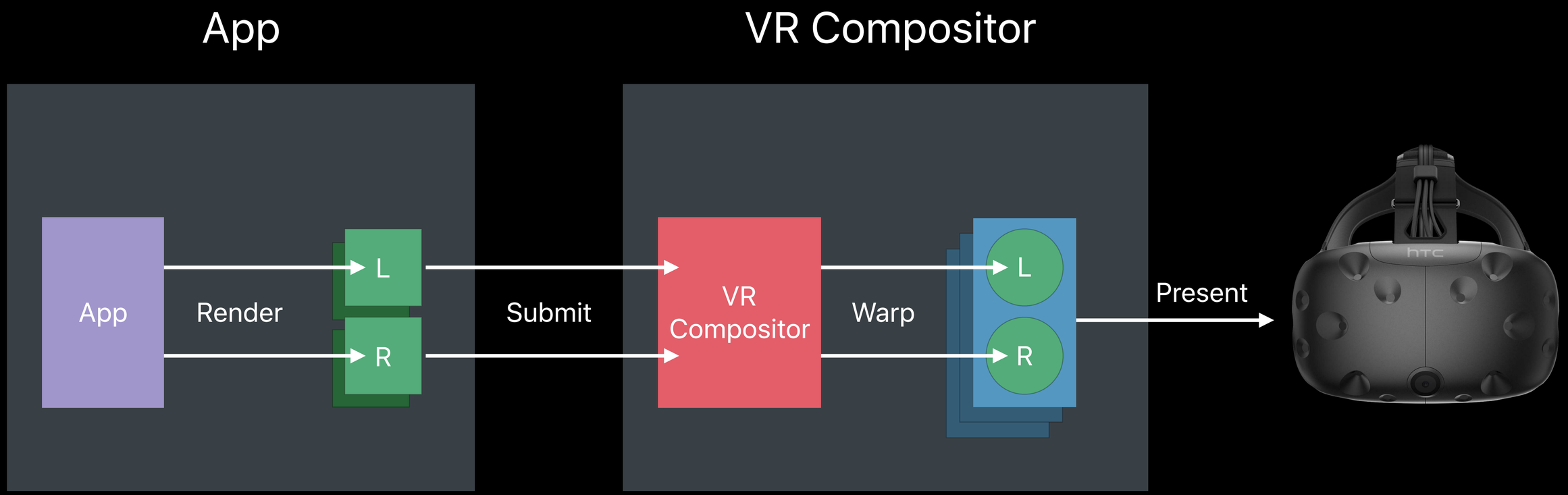
Managing Surfaces



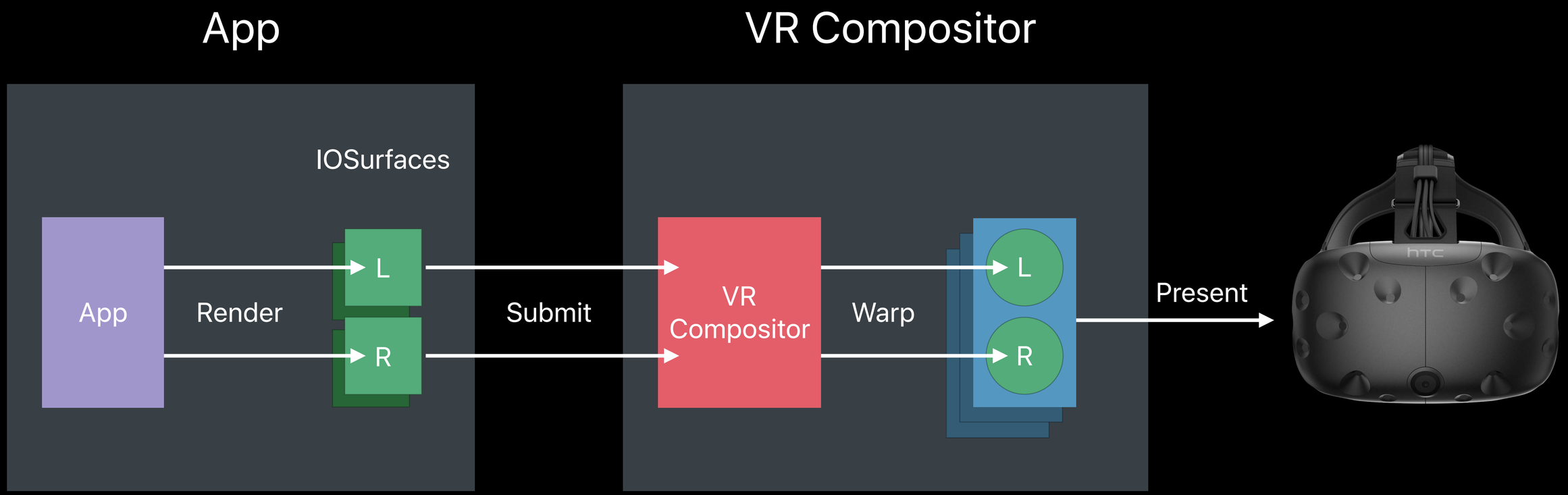
Managing Surfaces



Managing Surfaces



Managing Surfaces



```
// Creating Metal Textures

MTLTextureDescriptor *textureDesc = [MTLTextureDescriptor new];
textureDesc.width = vrWidth;
textureDesc.height = vrHeight;
textureDesc.pixelFormat = MTLPixelFormatRGBA8Unorm_sRGB;
textureDesc.storageMode = MTLStorageModeManaged;
textureDesc.usage = MTLTextureUsageRenderTarget | MTLTextureUsageShaderRead;

id <MTLTexture> right_tex =
   [device newTextureWithDescriptor:textureDesc iosurface:rightIOSurface plane:0];
id <MTLTexture> left_tex =
   [device newTextureWithDescriptor:textureDesc iosurface:leftIOSurface plane:0];
```

```
// Creating Metal Textures

MTLTextureDescriptor *textureDesc = [MTLTextureDescriptor new];
textureDesc.width = vrWidth;
textureDesc.height = vrHeight;
textureDesc.pixelFormat = MTLPixelFormatRGBA8Unorm_sRGB;
textureDesc.storageMode = MTLStorageModeManaged;
textureDesc.usage = MTLTextureUsageRenderTarget | MTLTextureUsageShaderRead;

id <MTLTexture> right_tex =
   [device newTextureWithDescriptor:textureDesc iosurface:rightIOSurface plane:0];
id <MTLTexture> left_tex =
   [device newTextureWithDescriptor:textureDesc iosurface:leftIOSurface plane:0];
```

```
// Creating Metal Textures

MTLTextureDescriptor *textureDesc = [MTLTextureDescriptor new];
textureDesc.width = vrWidth;
textureDesc.height = vrHeight;
textureDesc.pixelFormat = MTLPixelFormatRGBA8Unorm_sRGB;
textureDesc.storageMode = MTLStorageModeManaged;
textureDesc.usage = MTLTextureUsageRenderTarget | MTLTextureUsageShaderRead;

id <MTLTexture> right_tex =
   [device newTextureWithDescriptor:textureDesc iosurface:rightIOSurface plane:0];
id <MTLTexture> left_tex =
   [device newTextureWithDescriptor:textureDesc iosurface:leftIOSurface plane:0];
```

```
// Creating Metal Textures

MTLTextureDescriptor *textureDesc = [MTLTextureDescriptor new];
textureDesc.width = vrWidth;
textureDesc.height = vrHeight;
textureDesc.pixelFormat = MTLPixelFormatRGBA8Unorm_sRGB;
textureDesc.storageMode = MTLStorageModeManaged;
textureDesc.usage = MTLTextureUsageRenderTarget | MTLTextureUsageShaderRead;
```

```
id <MTLTexture> right_tex =
   [device newTextureWithDescriptor:textureDesc iosurface:rightIOSurface plane:0];
id <MTLTexture> left_tex =
   [device newTextureWithDescriptor:textureDesc iosurface:leftIOSurface plane:0];
```

VR App Building 101

Overview of VR development

macOS platform specifics

Anatomy of a VR frame

VR best practices

VR Compositor Synchronization

VR compositor and App work in lock step

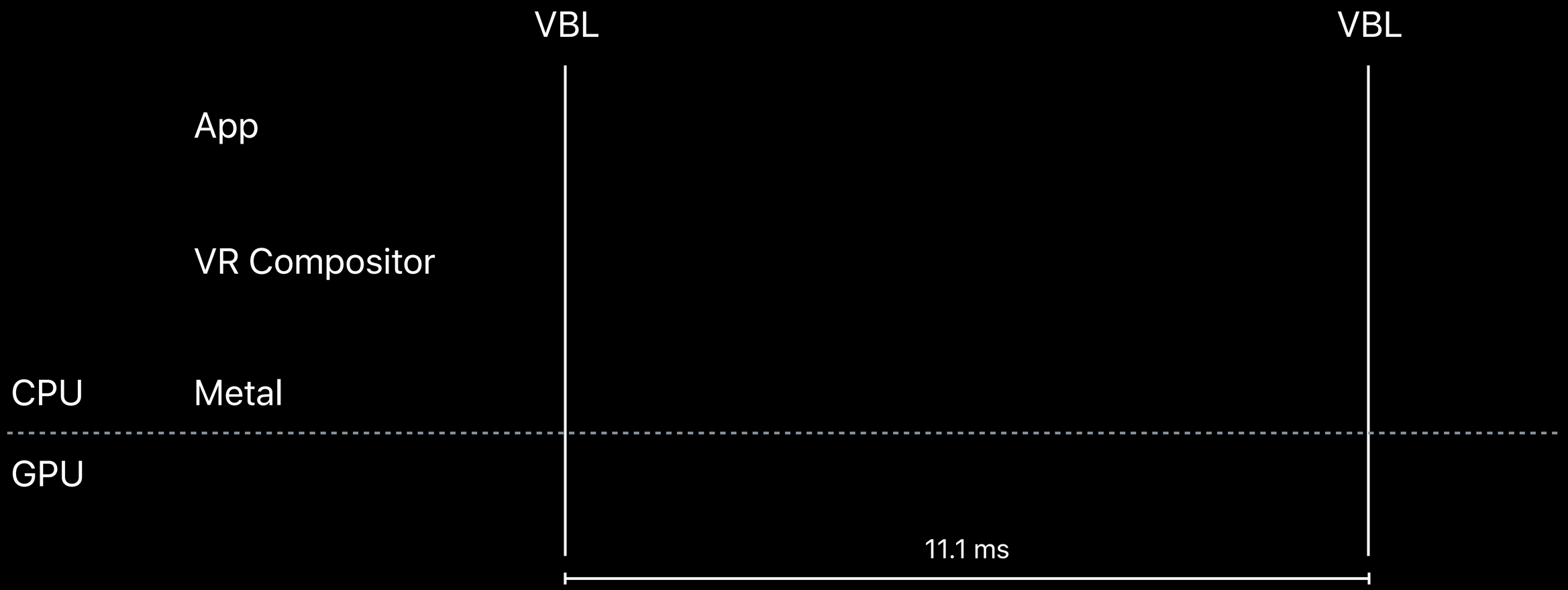
- Results of App rendering submitted to VR compositor

GPU is a shared resource

- Submission order matters

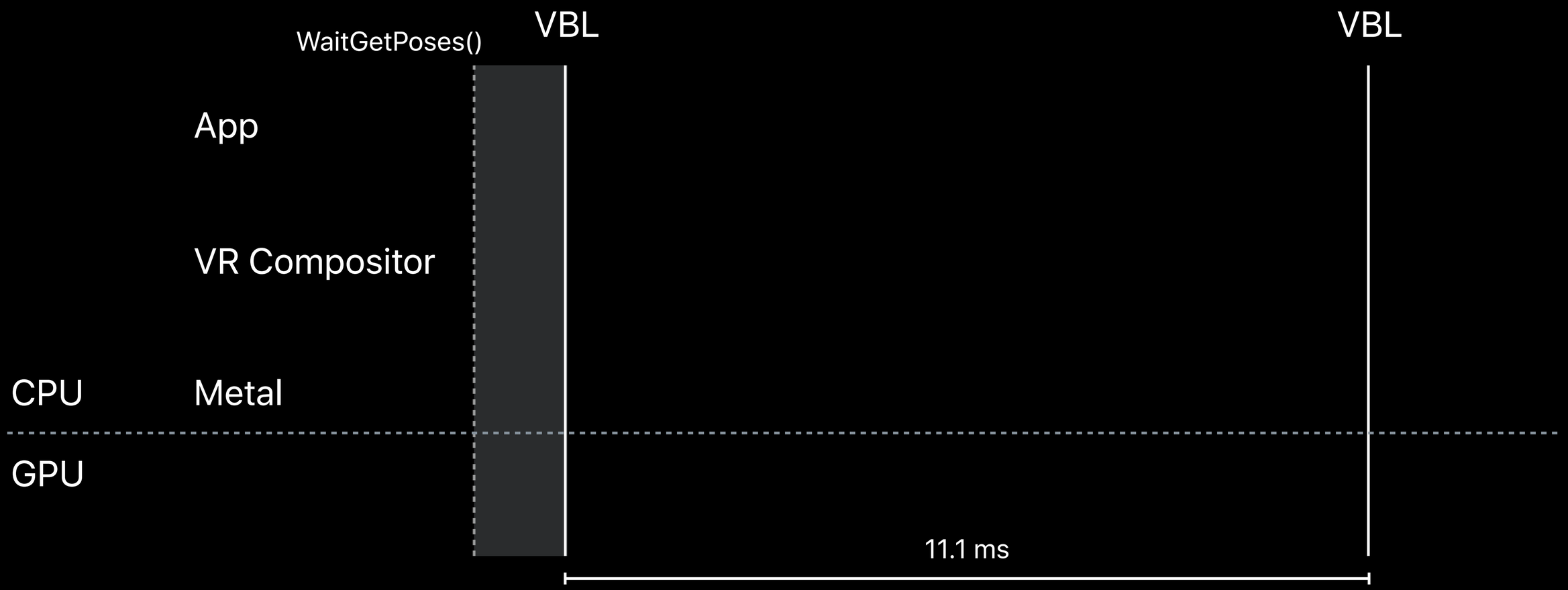
VR Compositor Synchronization

Start of frame



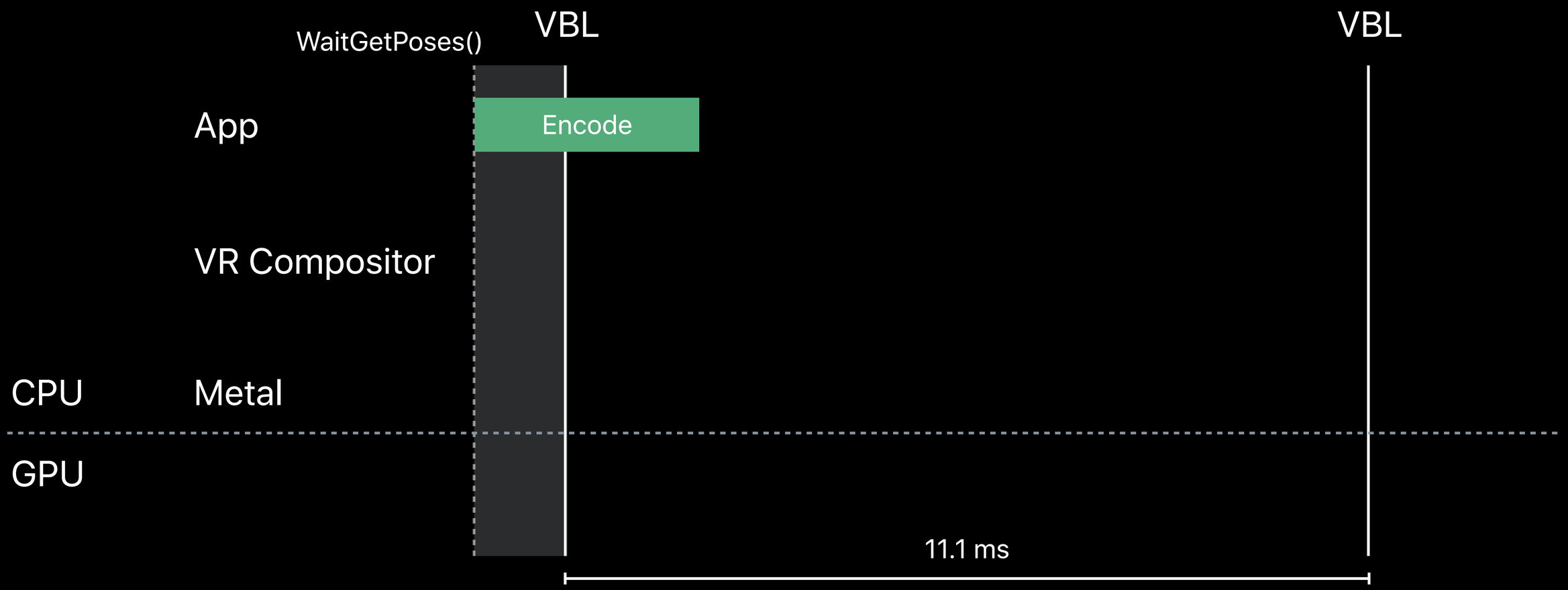
VR Compositor Synchronization

Start of frame



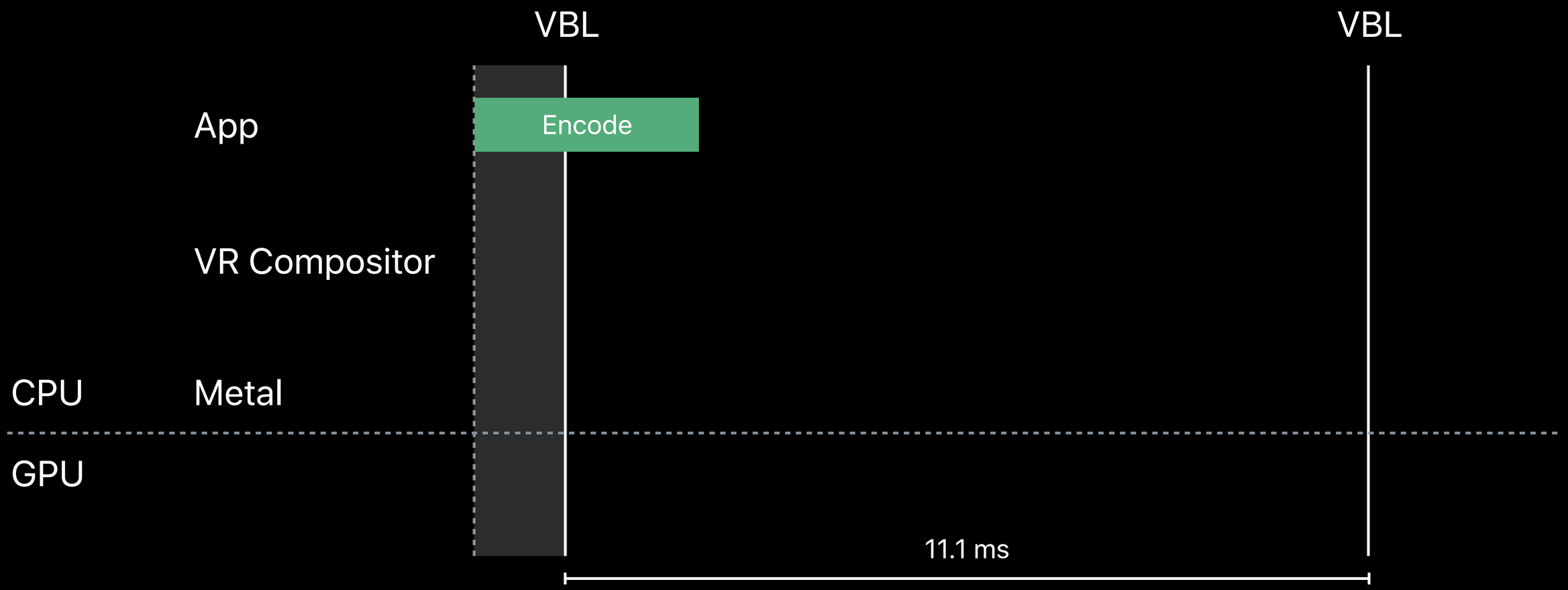
VR Compositor Synchronization

Start of frame



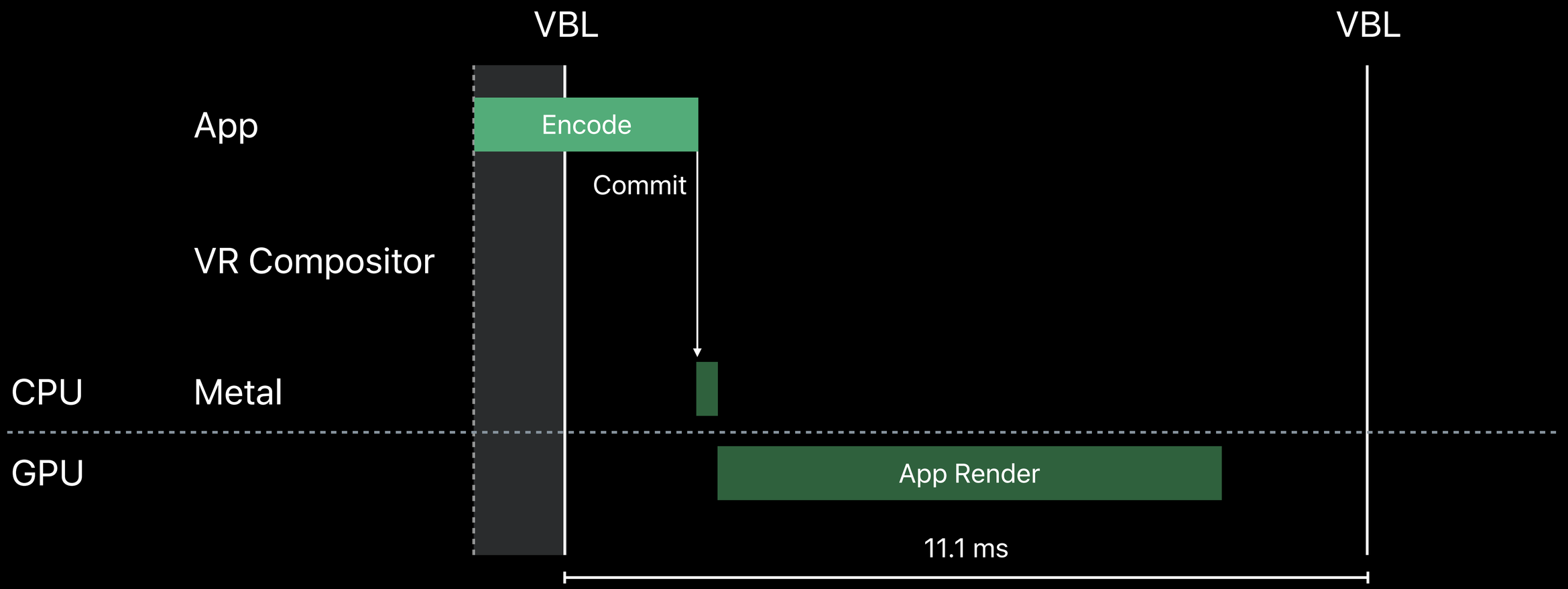
VR Compositor Synchronization

Submit to VR compositor



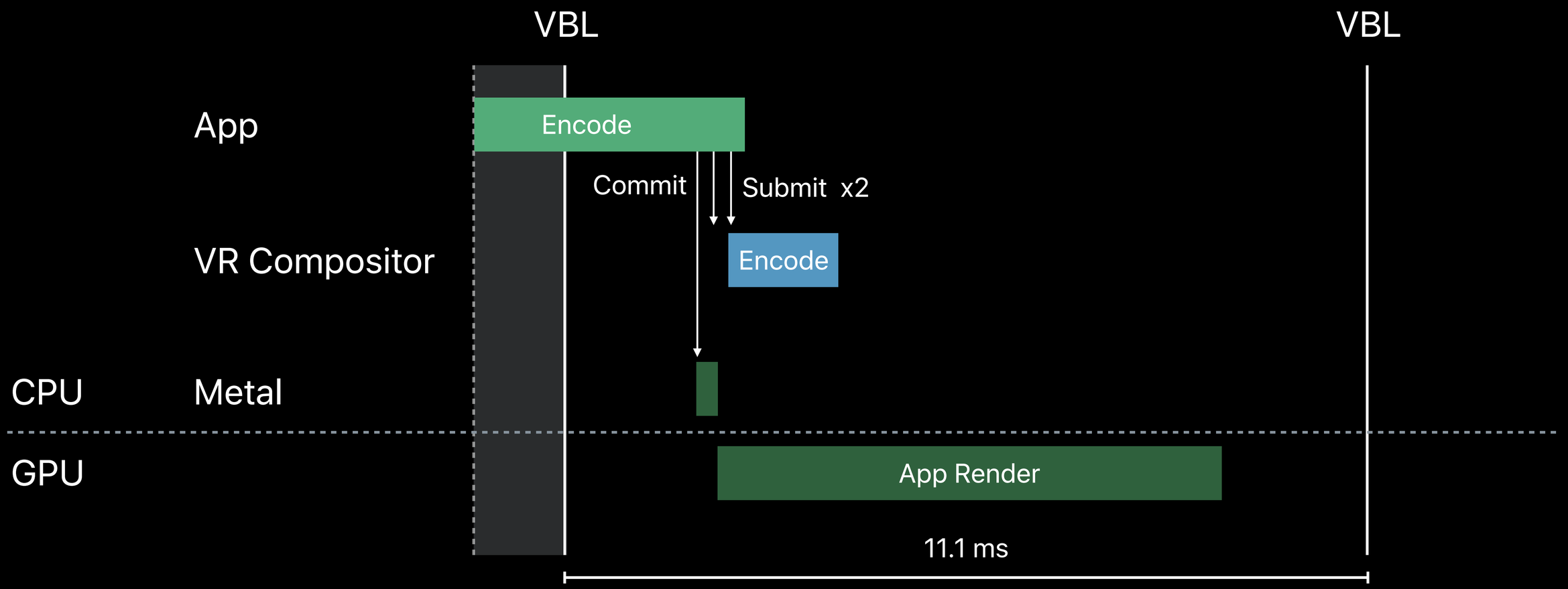
VR Compositor Synchronization

Submit to VR compositor



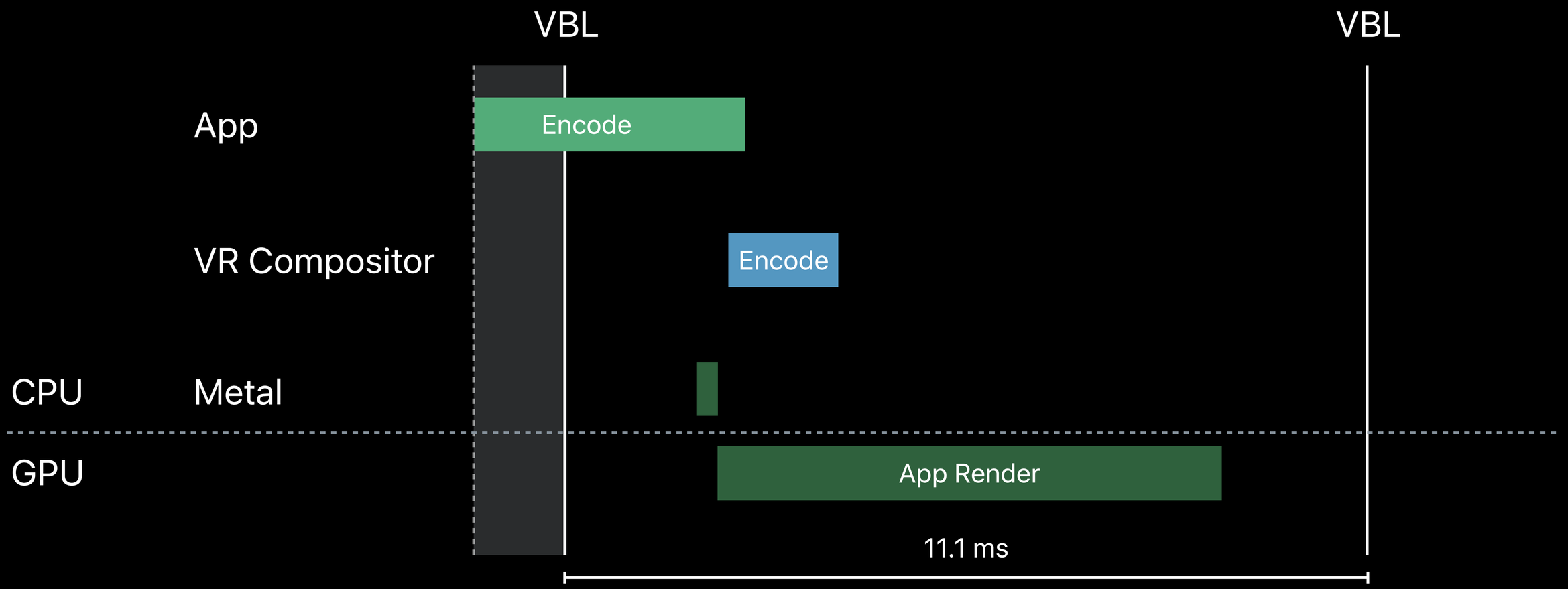
VR Compositor Synchronization

Submit to VR compositor



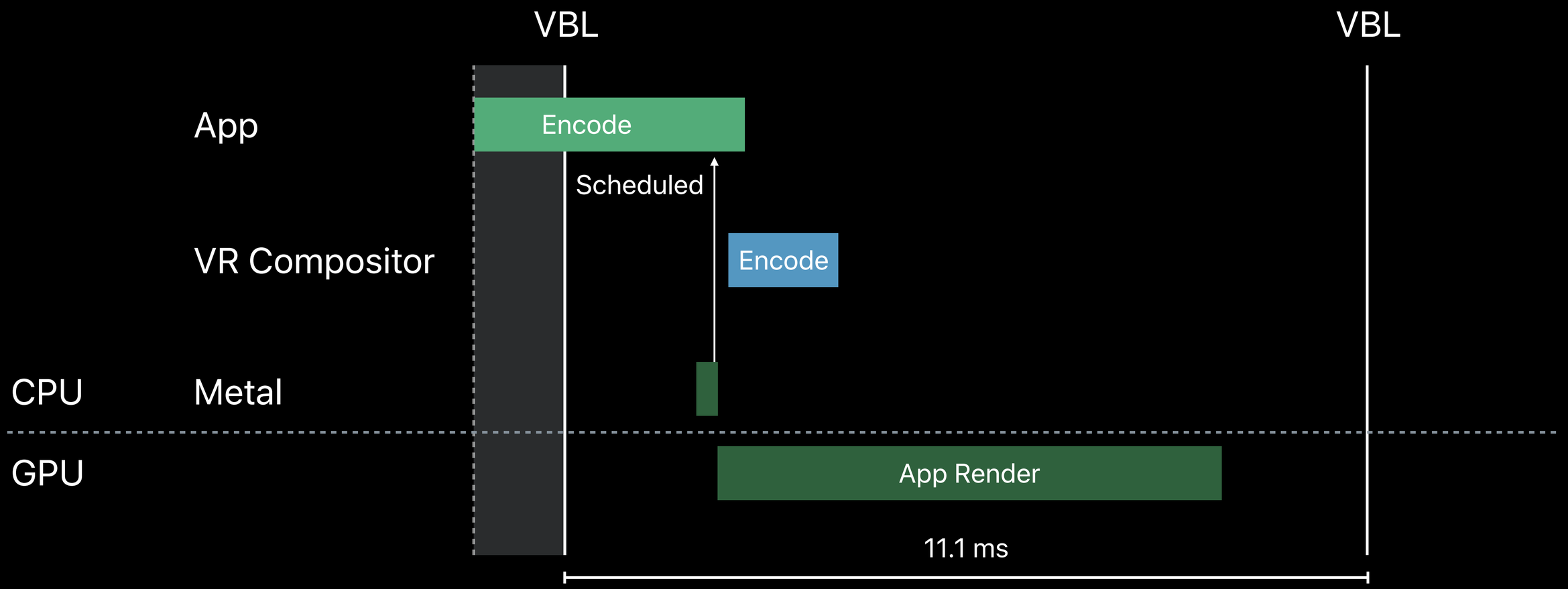
VR Compositor Synchronization

Guaranteeing submission order



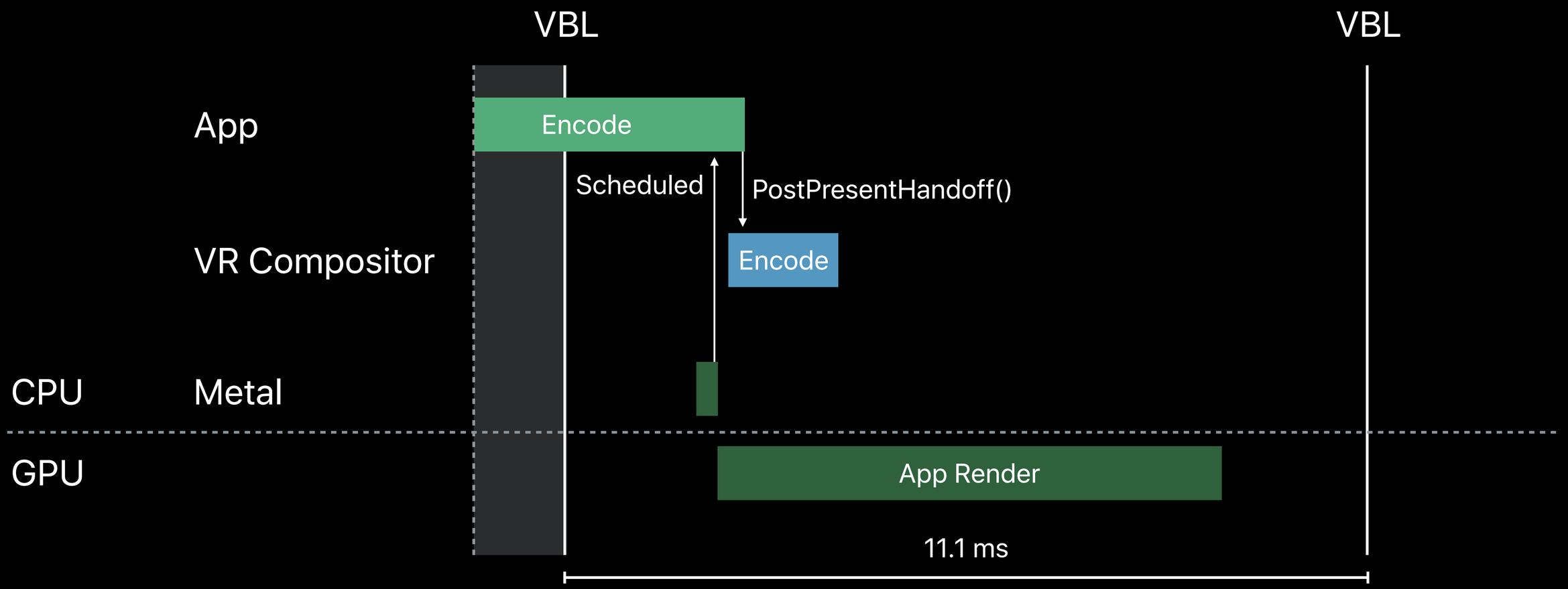
VR Compositor Synchronization

Guaranteeing submission order



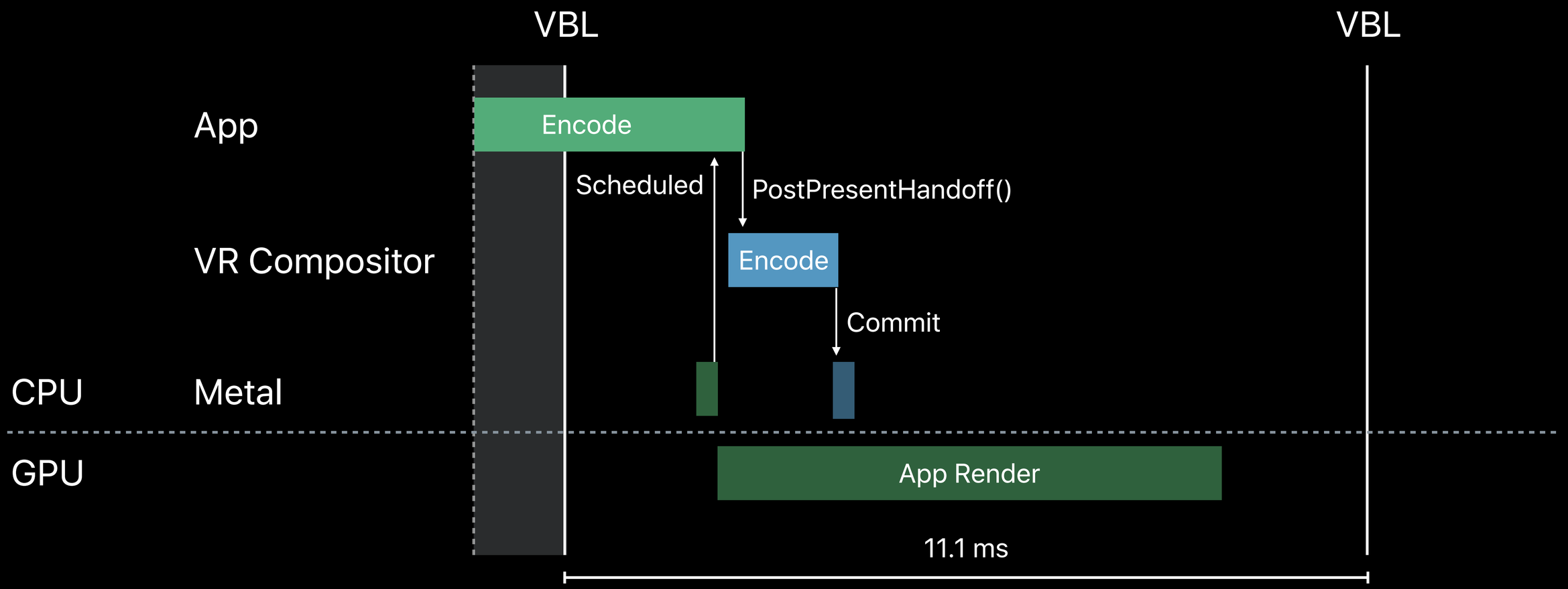
VR Compositor Synchronization

Guaranteeing submission order



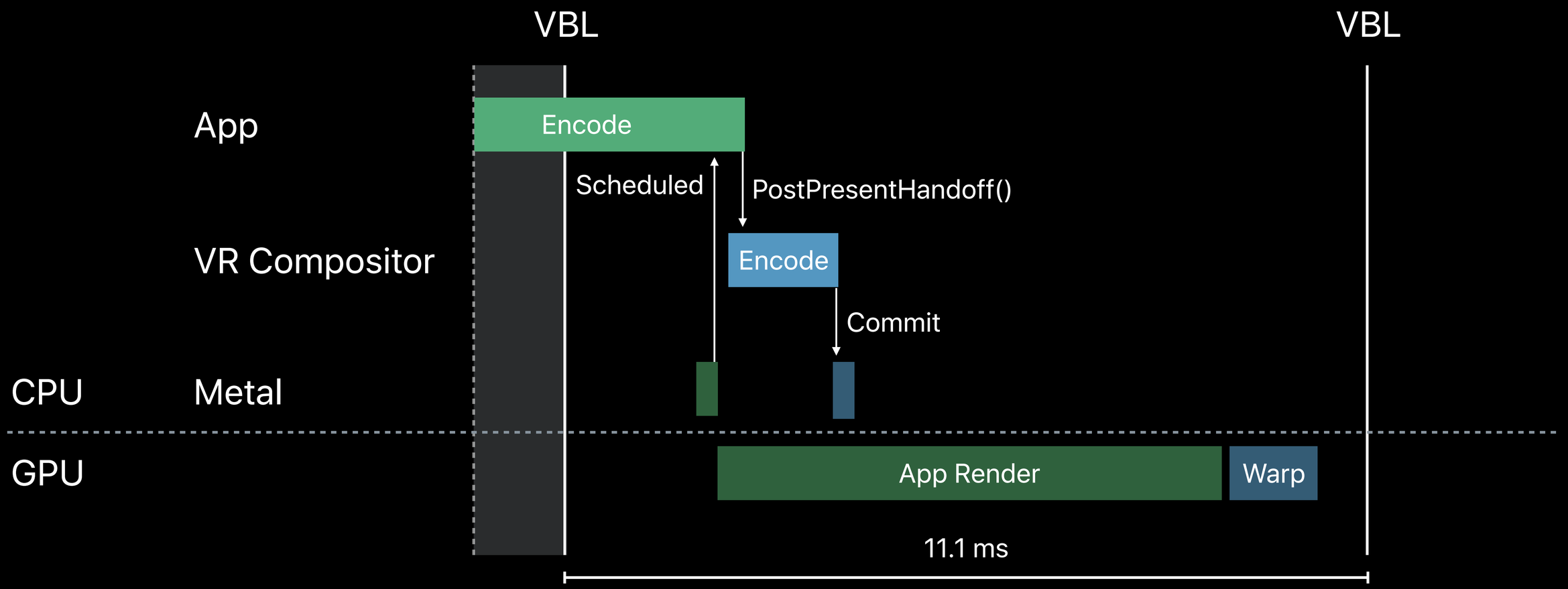
VR Compositor Synchronization

Guaranteeing submission order



VR Compositor Synchronization

Guaranteeing submission order



```
vr.VRCompositor()->WaitGetPoses(vrTrackedDevicePoses, vr::k_unMaxTrackedDeviceCount, nullptr, 0);

id<MTLCommandBuffer> commandBuffer = [sceneCommandQueue commandBuffer];

// render left and right eye images into eyeTextures
[self _drawSceneWithCommandBuffer:commandBuffer];

[commandBuffer commit];

vr.VRCompositor()->Submit(vr::Eye_Left, &eyeTextures[0], &vrEyeBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &eyeTextures[1], &vrEyeBounds[1]);

// wait until the GPU work is scheduled
[commandBuffer waitUntilScheduled];

// signal to the compositor that it can submit work to the GPU
vr.VRCompositor()->PostPresentHandoff();
```

```
vr.VRCompositor()->WaitGetPoses(vrTrackedDevicePoses, vr::k_unMaxTrackedDeviceCount, nullptr, 0);

id<MTLCommandBuffer> commandBuffer = [sceneCommandQueue commandBuffer];

// render left and right eye images into eyeTextures
[self _drawSceneWithCommandBuffer:commandBuffer];

[commandBuffer commit];

vr.VRCompositor()->Submit(vr::Eye_Left, &eyeTextures[0], &vrEyeBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &eyeTextures[1], &vrEyeBounds[1]);

// wait until the GPU work is scheduled
[commandBuffer waitUntilScheduled];

// signal to the compositor that it can submit work to the GPU
vr.VRCompositor()->PostPresentHandoff();
```

```
vr.VRCompositor()->WaitGetPoses(vrTrackedDevicePoses, vr::k_unMaxTrackedDeviceCount, nullptr, 0);

id<MTLCommandBuffer> commandBuffer = [sceneCommandQueue commandBuffer];

// render left and right eye images into eyeTextures
[self _drawSceneWithCommandBuffer:commandBuffer];

[commandBuffer commit];

vr.VRCompositor()->Submit(vr::Eye_Left, &eyeTextures[0], &vrEyeBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &eyeTextures[1], &vrEyeBounds[1]);

// wait until the GPU work is scheduled
[commandBuffer waitUntilScheduled];

// signal to the compositor that it can submit work to the GPU
vr.VRCompositor()->PostPresentHandoff();
```

```
vr.VRCompositor()->WaitGetPoses(vrTrackedDevicePoses, vr::k_unMaxTrackedDeviceCount, nullptr, 0);

id<MTLCommandBuffer> commandBuffer = [sceneCommandQueue commandBuffer];

// render left and right eye images into eyeTextures
[self _drawSceneWithCommandBuffer:commandBuffer];

[commandBuffer commit];

vr.VRCompositor()->Submit(vr::Eye_Left, &eyeTextures[0], &vrEyeBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &eyeTextures[1], &vrEyeBounds[1]);

// wait until the GPU work is scheduled
[commandBuffer waitUntilScheduled];

// signal to the compositor that it can submit work to the GPU
vr.VRCompositor()->PostPresentHandoff();
```

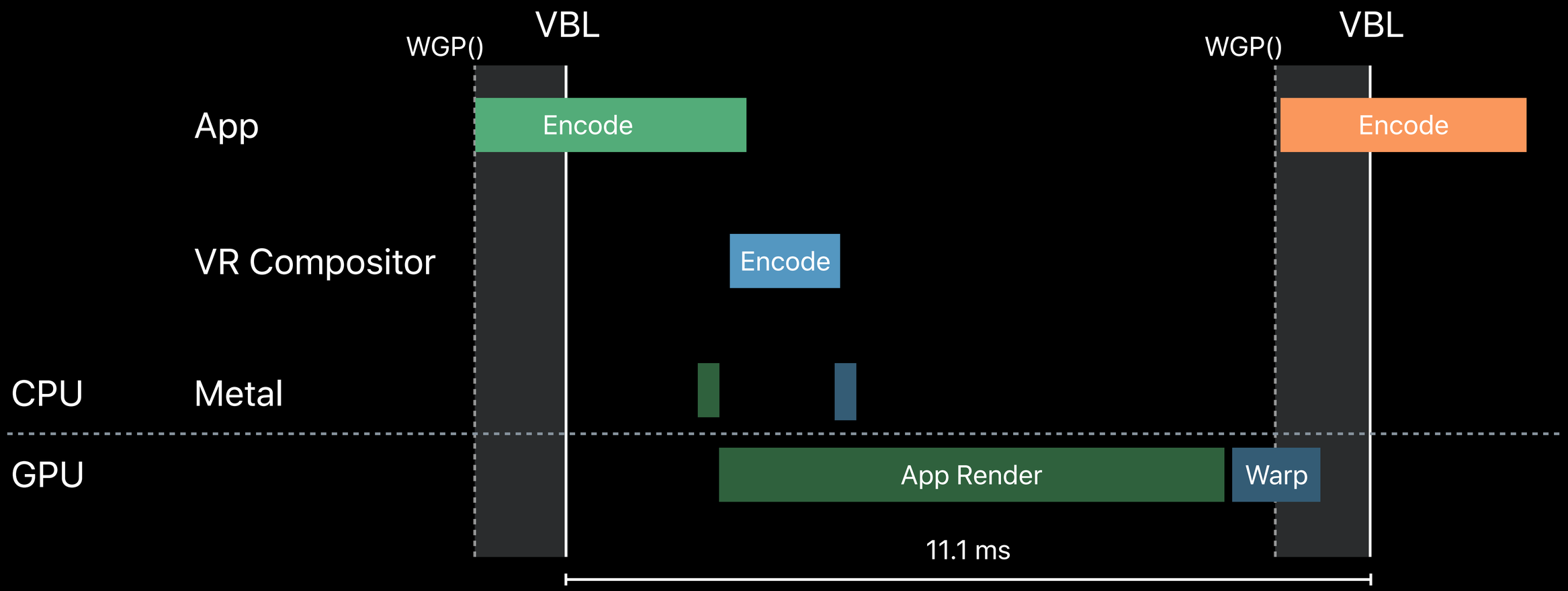
```
vr.VRCompositor()->WaitGetPoses(vrTrackedDevicePoses, vr::k_unMaxTrackedDeviceCount, nullptr, 0);

id<MTLCommandBuffer> commandBuffer = [sceneCommandQueue commandBuffer];

// render left and right eye images into eyeTextures
[self _drawSceneWithCommandBuffer:commandBuffer];

[commandBuffer commit];

vr.VRCompositor()->Submit(vr::Eye_Left, &eyeTextures[0], &vrEyeBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &eyeTextures[1], &vrEyeBounds[1]);

// wait until the GPU work is scheduled
[commandBuffer waitUntilScheduled];

// signal to the compositor that it can submit work to the GPU
vr.VRCompositor()->PostPresentHandoff();
```

```
vr.VRCompositor()->WaitGetPoses(vrTrackedDevicePoses, vr::k_unMaxTrackedDeviceCount, nullptr, 0);

id<MTLCommandBuffer> commandBuffer = [sceneCommandQueue commandBuffer];

// render left and right eye images into eyeTextures
[self _drawSceneWithCommandBuffer:commandBuffer];

[commandBuffer commit];

vr.VRCompositor()->Submit(vr::Eye_Left, &eyeTextures[0], &vrEyeBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &eyeTextures[1], &vrEyeBounds[1]);

// wait until the GPU work is scheduled
[commandBuffer waitUntilScheduled];

// signal to the compositor that it can submit work to the GPU
vr.VRCompositor()->PostPresentHandoff();
```

```
vr.VRCompositor()->WaitGetPoses(vrTrackedDevicePoses, vr::k_unMaxTrackedDeviceCount, nullptr, 0);

id<MTLCommandBuffer> commandBuffer = [sceneCommandQueue commandBuffer];

// render left and right eye images into eyeTextures
[self _drawSceneWithCommandBuffer:commandBuffer];

[commandBuffer commit];

vr.VRCompositor()->Submit(vr::Eye_Left, &eyeTextures[0], &vrEyeBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &eyeTextures[1], &vrEyeBounds[1]);

// wait until the GPU work is scheduled
[commandBuffer waitUntilScheduled];

// signal to the compositor that it can submit work to the GPU
vr.VRCompositor()->PostPresentHandoff();
```

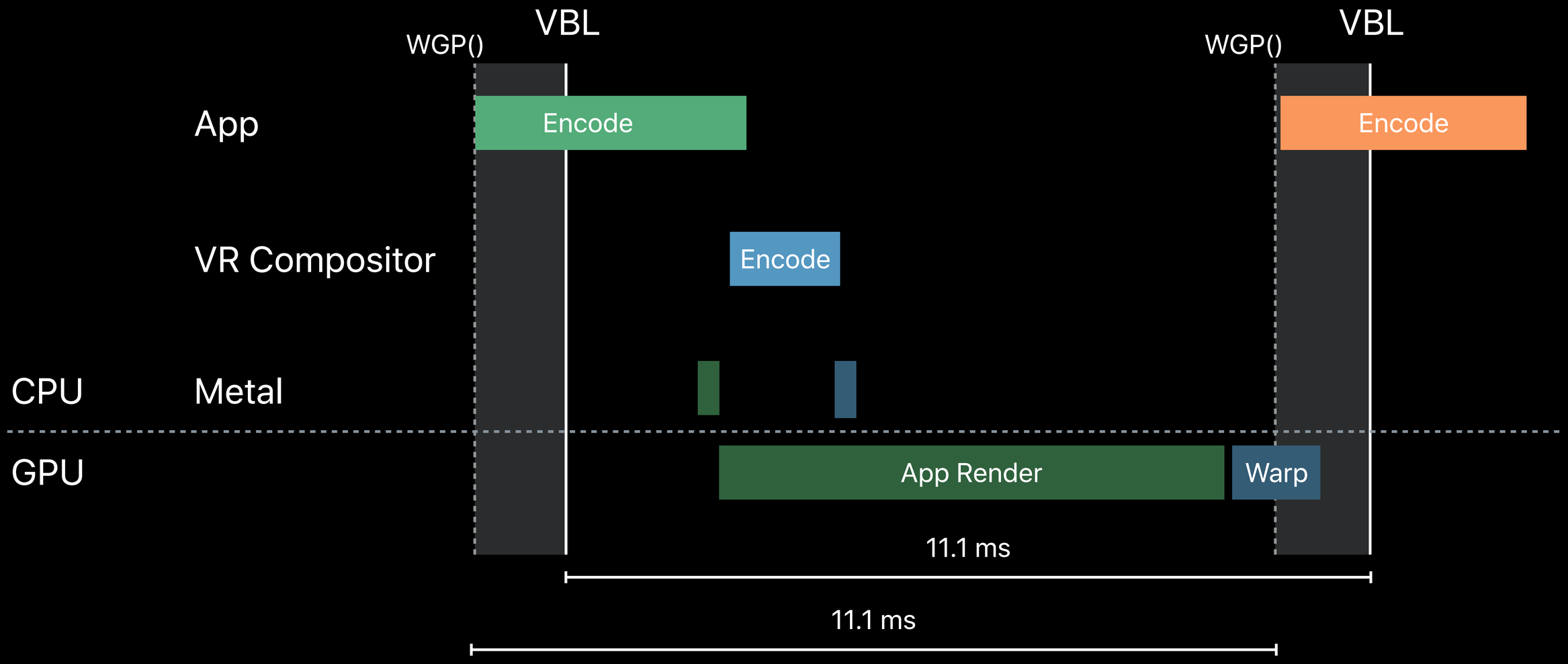
VR Compositor Synchronization

Frame cadence



VR Compositor Synchronization

Frame cadence



VR App Building 101

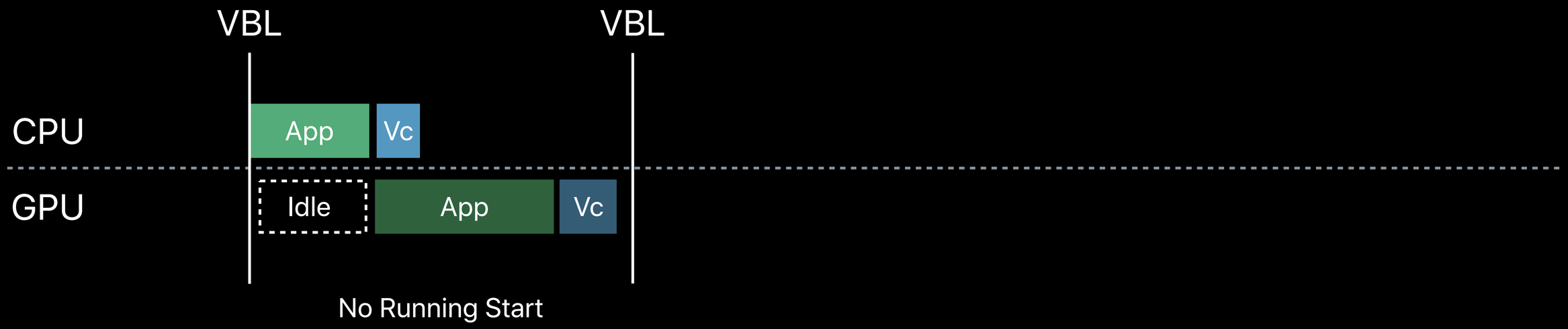
Overview of VR development

macOS platform specifics

Anatomy of a VR frame

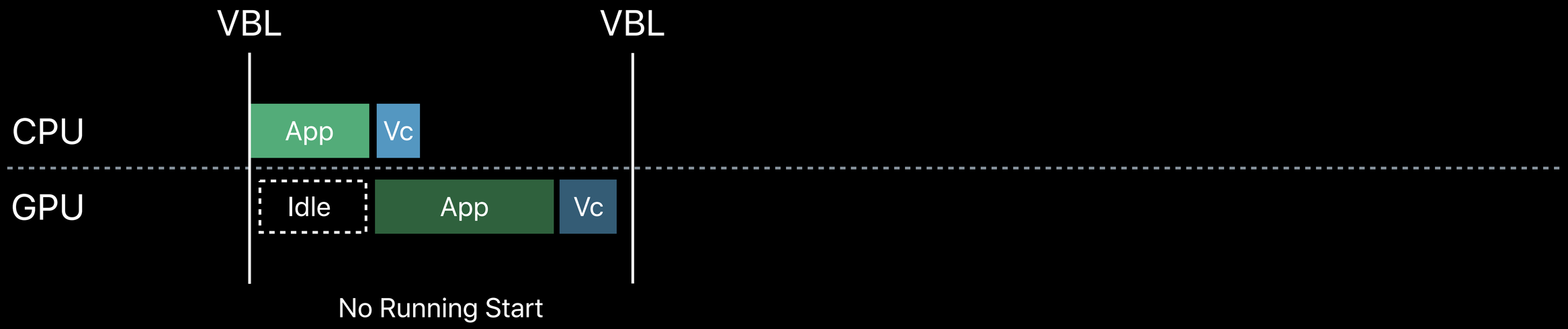
VR best practices

Start Early



Start Early

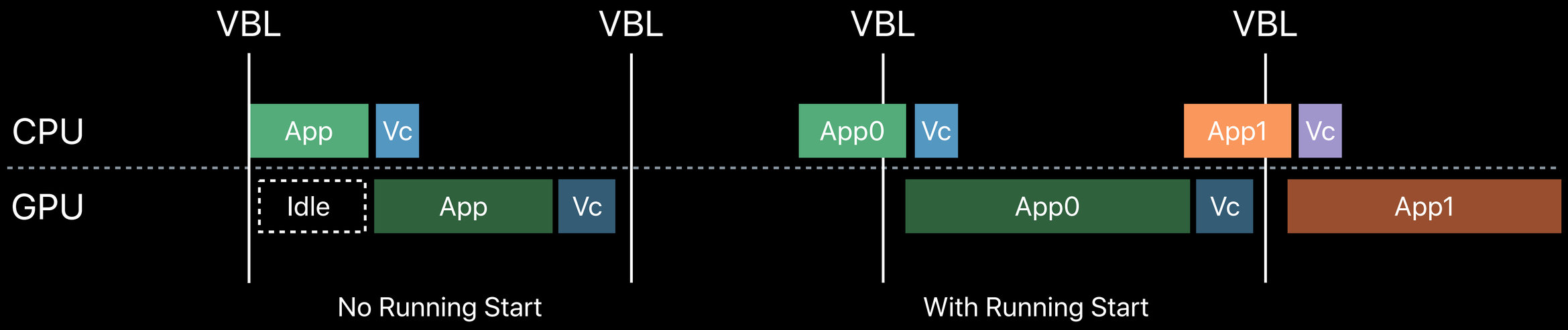
CPU work at frame start can introduce “GPU bubbles”



Start Early

CPU work at frame start can introduce “GPU bubbles”

SteamVR provides a mechanism to start early

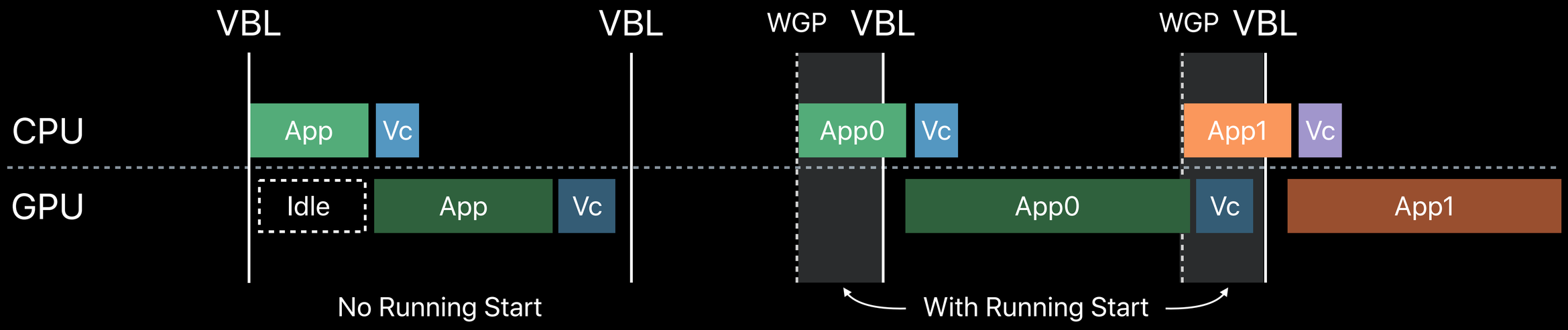


Start Early

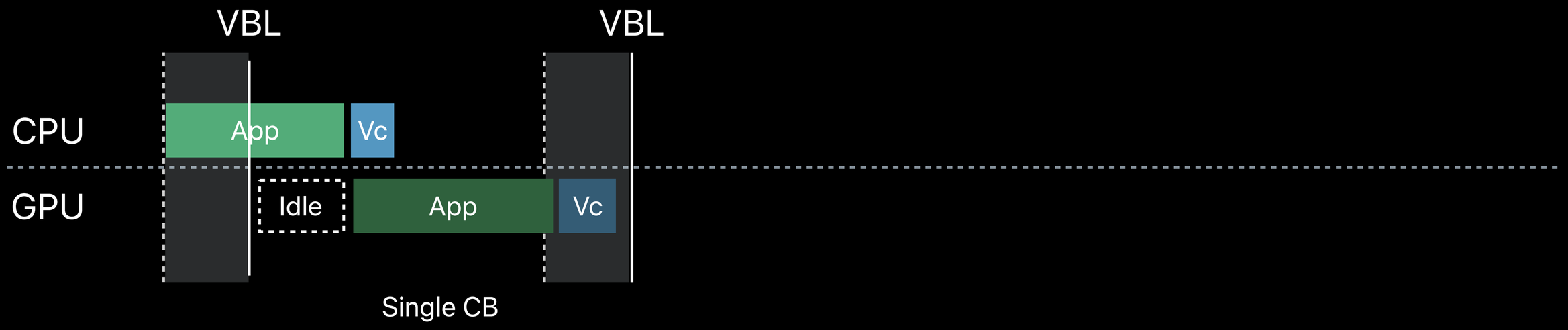
CPU work at frame start can introduce “GPU bubbles”

SteamVR provides a mechanism to start early

Encode your frame after WaitGetPoses returns

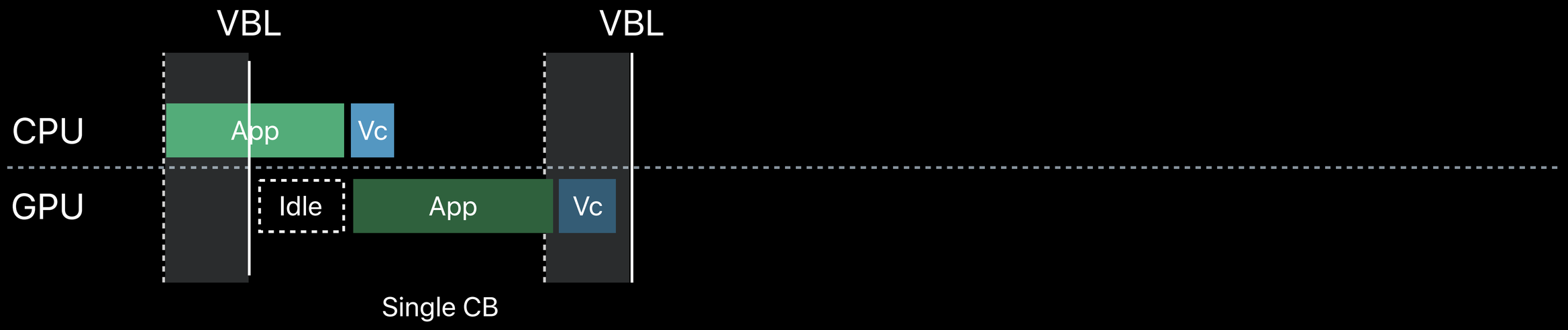


Split Your Command Buffers



Split Your Command Buffers

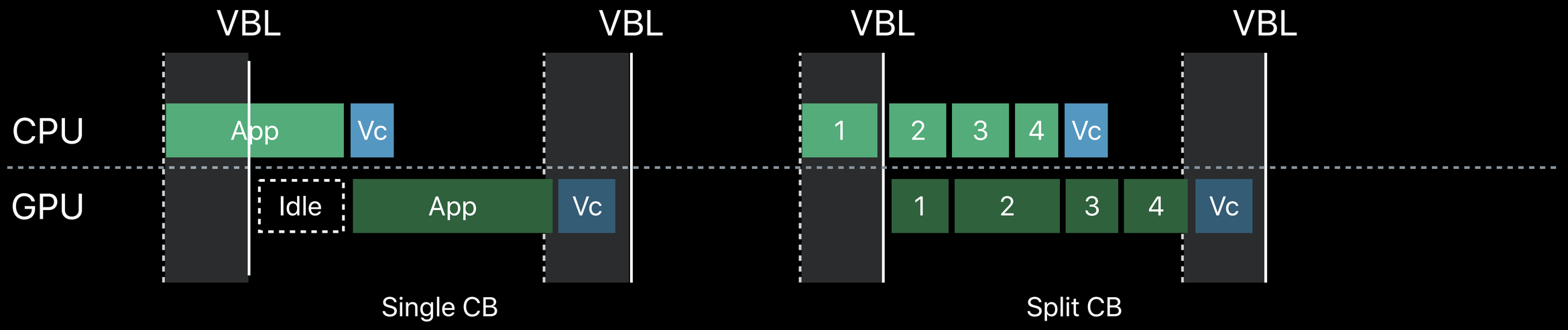
Avoid submitting monolithic command buffers



Split Your Command Buffers

Avoid submitting monolithic command buffers

Split and submit as you go to maximize GPU utilization



Coalesce Left and Right Eye Draws

Coalesce Left and Right Eye Draws

NEW

Use the Metal 2 Viewport Array feature

- Per-primitive viewport selection in the vertex shader

Coalesce Left and Right Eye Draws

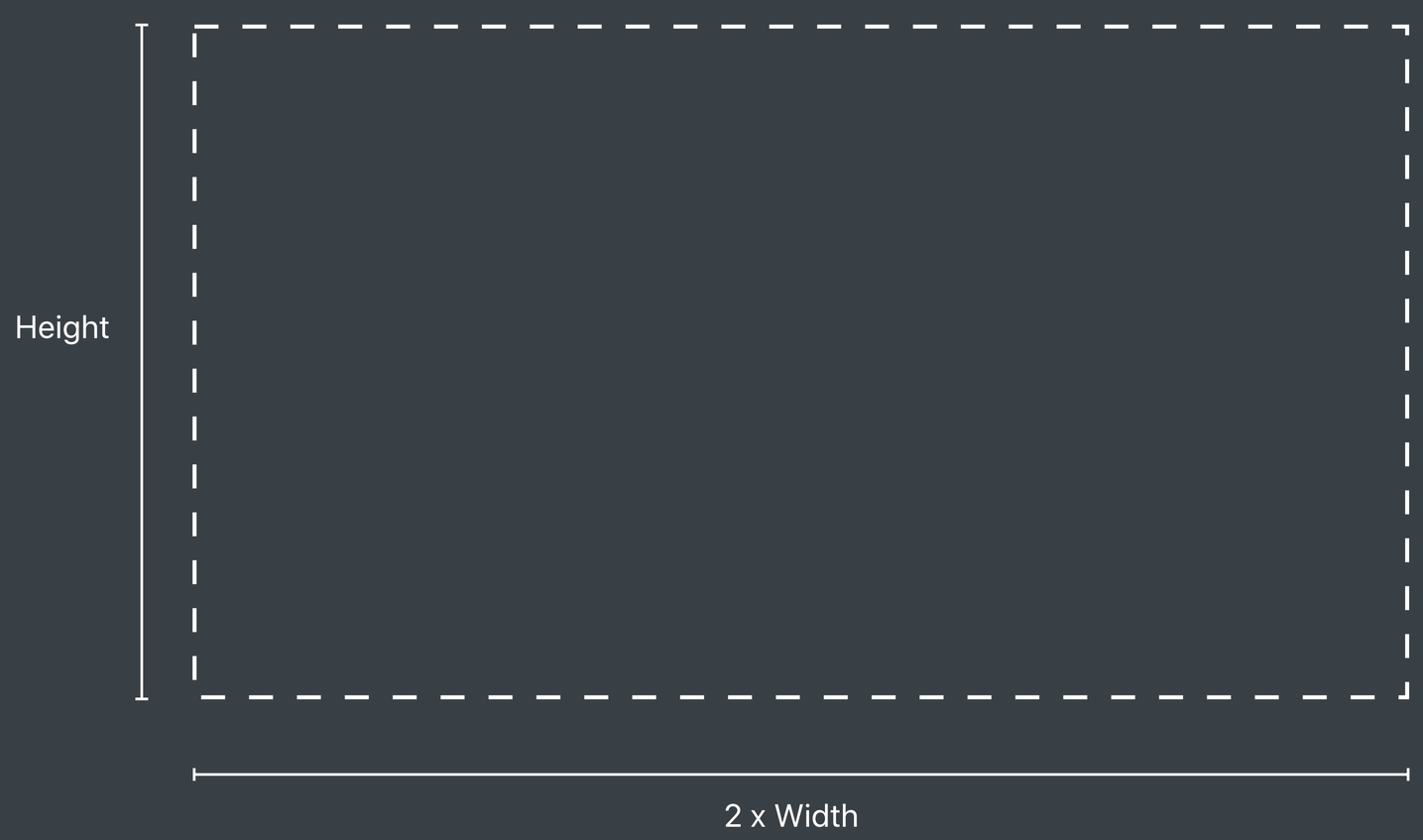
NEW

Use the Metal 2 Viewport Array feature

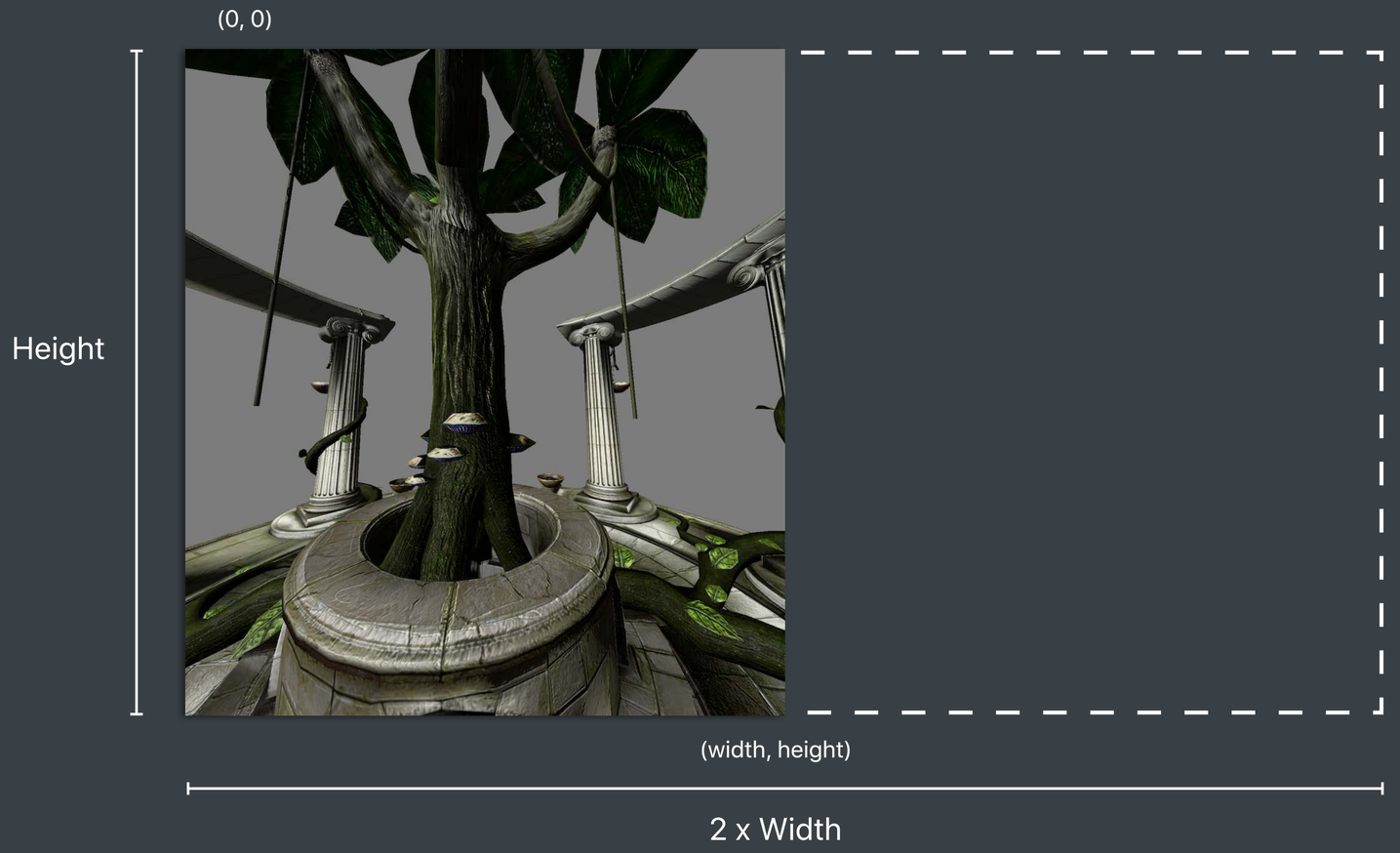
- Per-primitive viewport selection in the vertex shader

Render to left and right eye with a single draw call

```
// Each viewport will cover half the texture size
MTLViewport vrViewports[2];

vrViewports[0].originX = 0;
vrViewports[0].originY = 0;
vrViewports[0].width = width;
vrViewports[0].height = height;
vrViewports[0].znear = 0.0;
vrViewports[0].zfar = 1.0;

vrViewports[1].originX = width;
vrViewports[1].originY = 0;
vrViewports[1].width = width;
vrViewports[1].height = height;
vrViewports[1].znear = 0.0;
vrViewports[1].zfar = 1.0;
```



```
// Each viewport will cover half the texture size
MTLViewport vrViewports[2];

vrViewports[0].originX = 0;
vrViewports[0].originY = 0;
vrViewports[0].width = width;
vrViewports[0].height = height;
vrViewports[0].znear = 0.0;
vrViewports[0].zfar = 1.0;

vrViewports[1].originX = width;
vrViewports[1].originY = 0;
vrViewports[1].width = width;
vrViewports[1].height = height;
vrViewports[1].znear = 0.0;
vrViewports[1].zfar = 1.0;
```



```
// Each viewport will cover half the texture size
MTLViewport vrViewports[2];

vrViewports[0].originX = 0;
vrViewports[0].originY = 0;
vrViewports[0].width = width;
vrViewports[0].height = height;
vrViewports[0].znear = 0.0;
vrViewports[0].zfar = 1.0;

vrViewports[1].originX = width;
vrViewports[1].originY = 0;
vrViewports[1].width = width;
vrViewports[1].height = height;
vrViewports[1].znear = 0.0;
vrViewports[1].zfar = 1.0;
```

```
// Set the viewports on your render command encoder for that render pass
[renderEncoder setViewports:vrViewports count:2];


// The instance_id will be used as the "eye" index
[renderEncoder drawIndexedPrimitives:primitiveType
                          indexCount:count
                           indexType:type
                         indexBuffer:buffer
                   indexBufferOffset:offset
                       instanceCount:2];
```

```
// Set the viewports on your render command encoder for that render pass
[renderEncoder setViewports:vrViewports count:2];


// The instance_id will be used as the "eye" index
[renderEncoder drawIndexedPrimitives:primitiveType
                          indexCount:count
                           indexType:type
                         indexBuffer:buffer
                   indexBufferOffset:offset
                       instanceCount:2];
```

```
// Set the viewports on your render command encoder for that render pass
[renderEncoder setViewports:vrViewports count:2];


// The instance_id will be used as the "eye" index
[renderEncoder drawIndexedPrimitives:primitiveType
                          indexCount:count
                           indexType:type
                         indexBuffer:buffer
                   indexBufferOffset:offset
                       instanceCount:2];
```

```
typedef struct
{
    float4 position [[ position ]];
    …
    ushort viewport [[ viewport_array_index ]];
} ColorInOut;

// Vertex Shader treats instance_id as an eye index
vertex ColorInOut VS(Vertex in [[ stage_in ]],
                     constant AAPLUniforms& uniforms [[ buffer(kBufferIndexUniforms) ]],
                     ushort iid [[ instance_id ]])
{
    ColorInOut out;
    out.position = uniforms.modelViewProjectionMatrix[iid] * float4(in.position, 1.0);
    …
    out.viewport = iid;
    return out;
}
```

```
typedef struct
{
    float4 position [[ position ]];
    …
    ushort viewport [[ viewport_array_index ]];
} ColorInOut;

// Vertex Shader treats instance_id as an eye index
vertex ColorInOut VS(Vertex in [[ stage_in ]],
                     constant AAPLUniforms& uniforms [[ buffer(kBufferIndexUniforms) ]],
                     ushort iid [[ instance_id ]])
{
    ColorInOut out;
    out.position = uniforms.modelViewProjectionMatrix[iid] * float4(in.position, 1.0);
    …
    out.viewport = iid;
    return out;
}
```

```
typedef struct
{
    float4 position [[ position ]];
    …
    ushort viewport [[ viewport_array_index ]];
} ColorInOut;

// Vertex Shader treats instance_id as an eye index
vertex ColorInOut VS(Vertex in [[ stage_in ]],
                     constant AAPLUniforms& uniforms [[ buffer(kBufferIndexUniforms) ]],
                     ushort iid [[ instance_id ]])
{
    ColorInOut out;
    out.position = uniforms.modelViewProjectionMatrix[iid] * float4(in.position, 1.0);
    …
    out.viewport = iid;
    return out;
}
```

```
typedef struct
{
    float4 position [[ position ]];
    …
    ushort viewport [[ viewport_array_index ]];
} ColorInOut;

// Vertex Shader treats instance_id as an eye index
vertex ColorInOut VS(Vertex in [[ stage_in ]],
                     constant AAPLUniforms& uniforms [[ buffer(kBufferIndexUniforms) ]],
                     ushort iid [[ instance_id ]])
{
    ColorInOut out;
    out.position = uniforms.modelViewProjectionMatrix[iid] * float4(in.position, 1.0);
    …
    out.viewport = iid;
    return out;
}
```

```
typedef struct
{
    float4 position [[ position ]];
    …
    ushort viewport [[ viewport_array_index ]];
} ColorInOut;

// Vertex Shader treats instance_id as an eye index
vertex ColorInOut VS(Vertex in [[ stage_in ]],
                     constant AAPLUniforms& uniforms [[ buffer(kBufferIndexUniforms) ]],
                     ushort iid [[ instance_id ]])
{
    ColorInOut out;
    out.position = uniforms.modelViewProjectionMatrix[iid] * float4(in.position, 1.0);
    …
    out.viewport = iid;
    return out;
}
```

Render Fewer Pixels

Render Fewer Pixels

15% of rendered pixels not displayed

Render Fewer Pixels

15% of rendered pixels not displayed

Use SteamVR stencil mask to clip these pixels

VR App Building 101

Overview of VR development

macOS platform specifics

Anatomy of a VR frame

VR best practices

STEAM® VR on macOS

Nat Brown, Valve Software

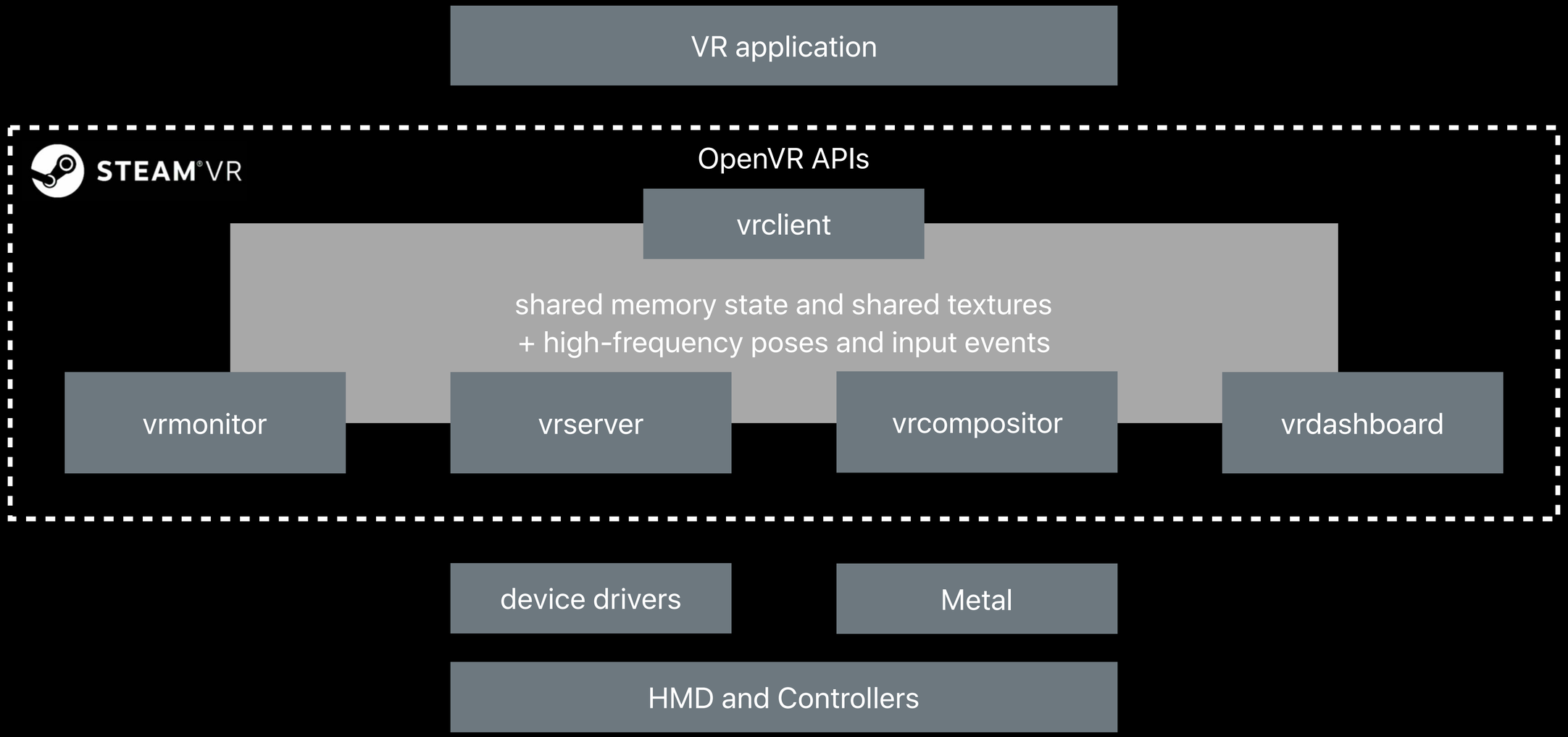
VR Motivation

VR is a long-term investment for Valve

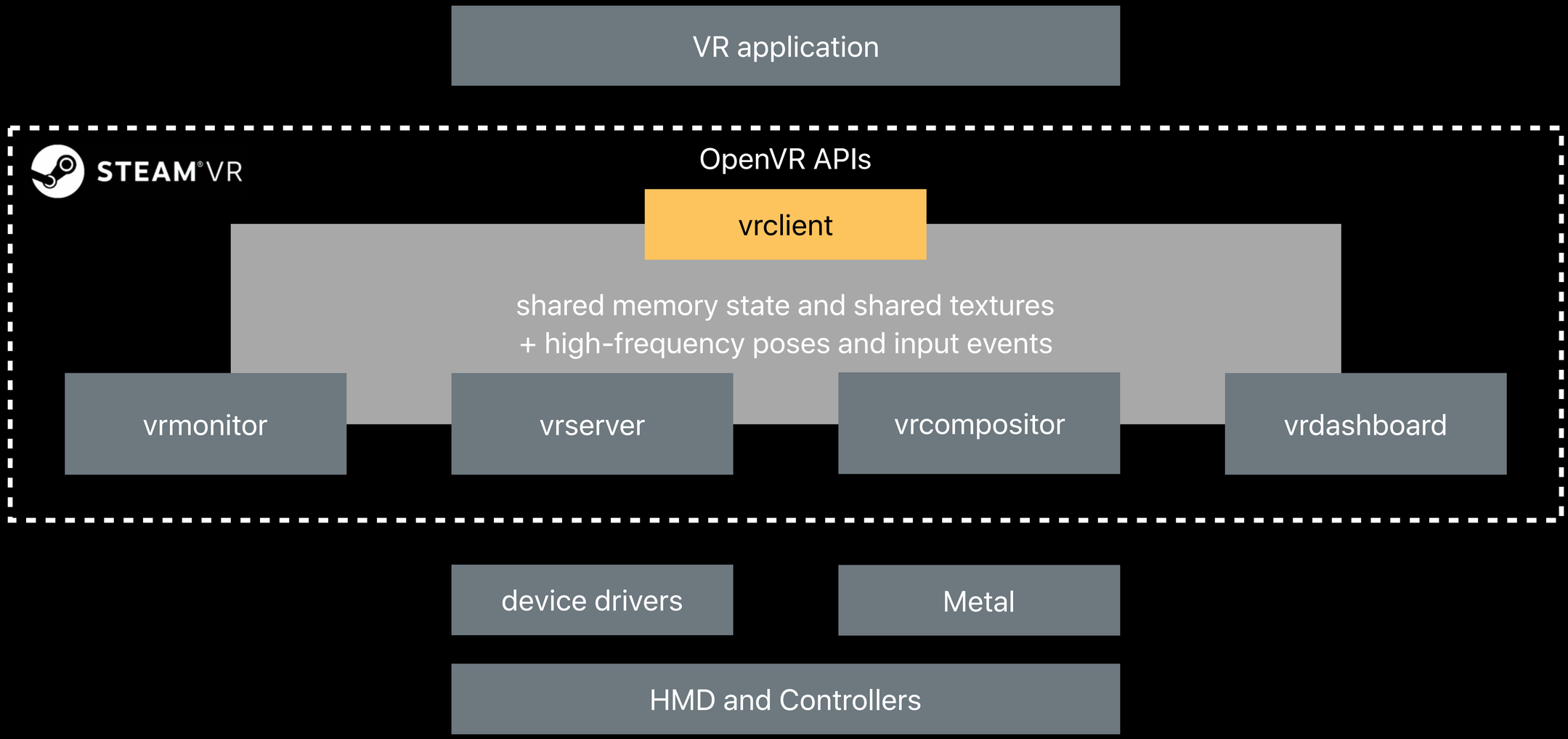
360 + room scale + input = magical

Valve licenses VR tech non-exclusively

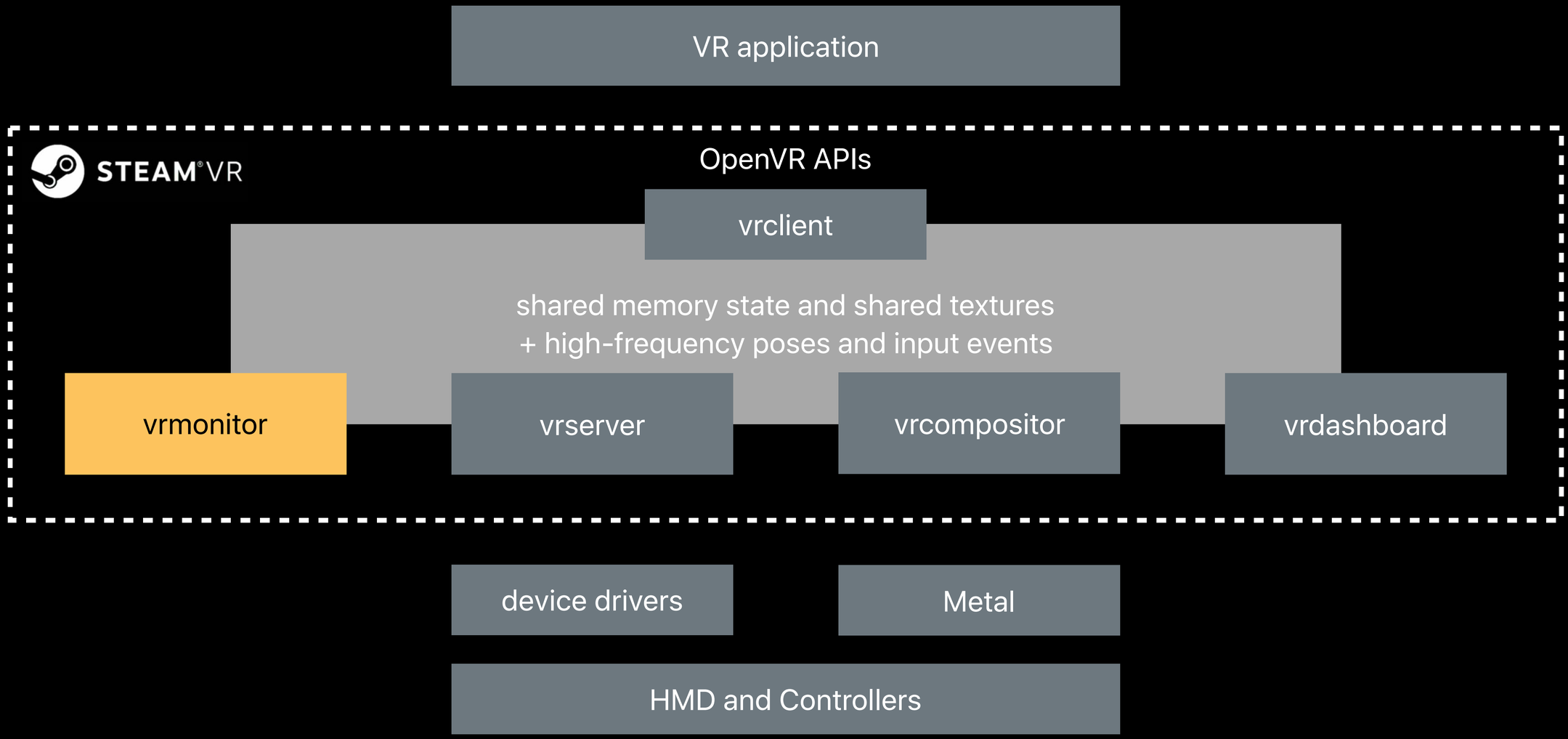
SteamVR Architecture



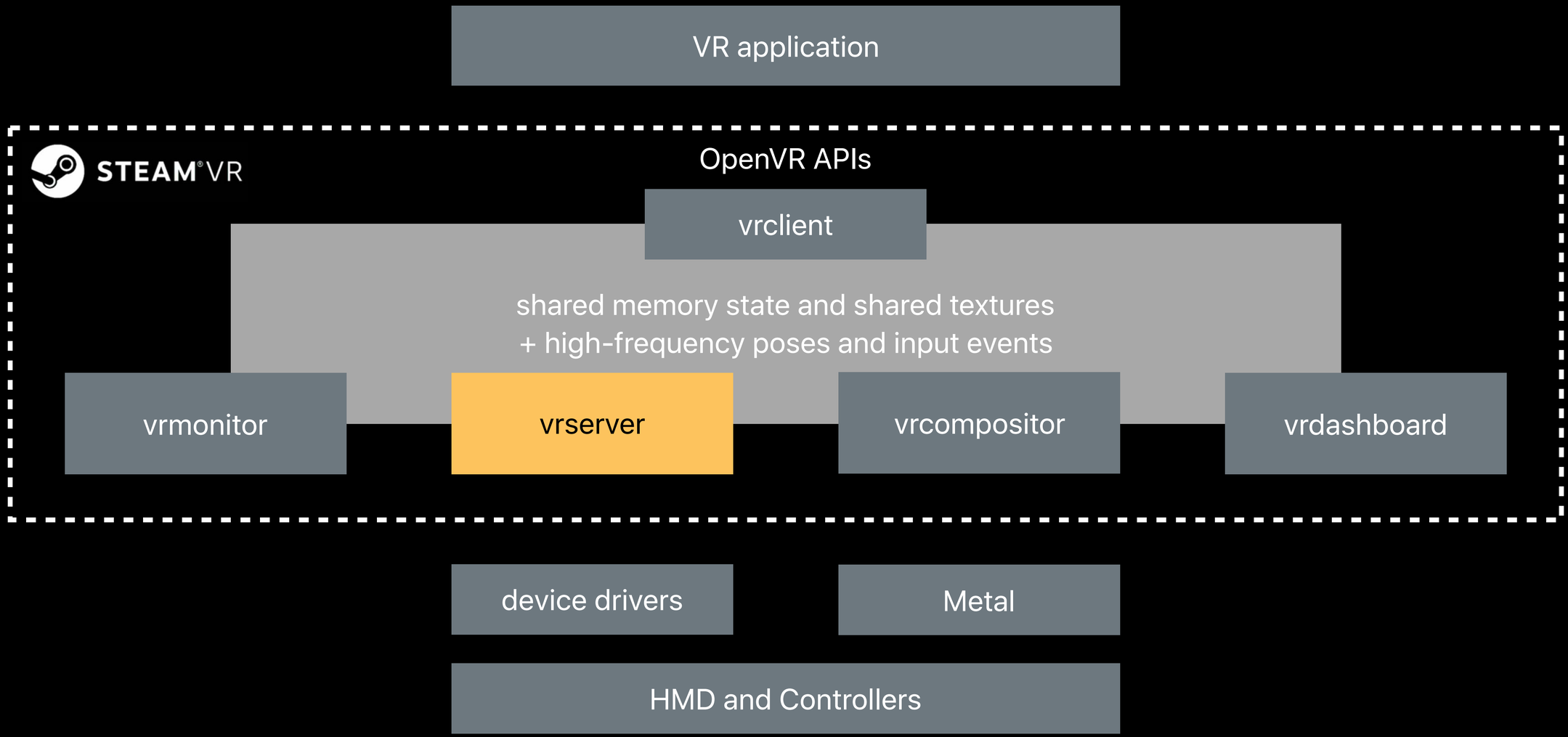
SteamVR Architecture



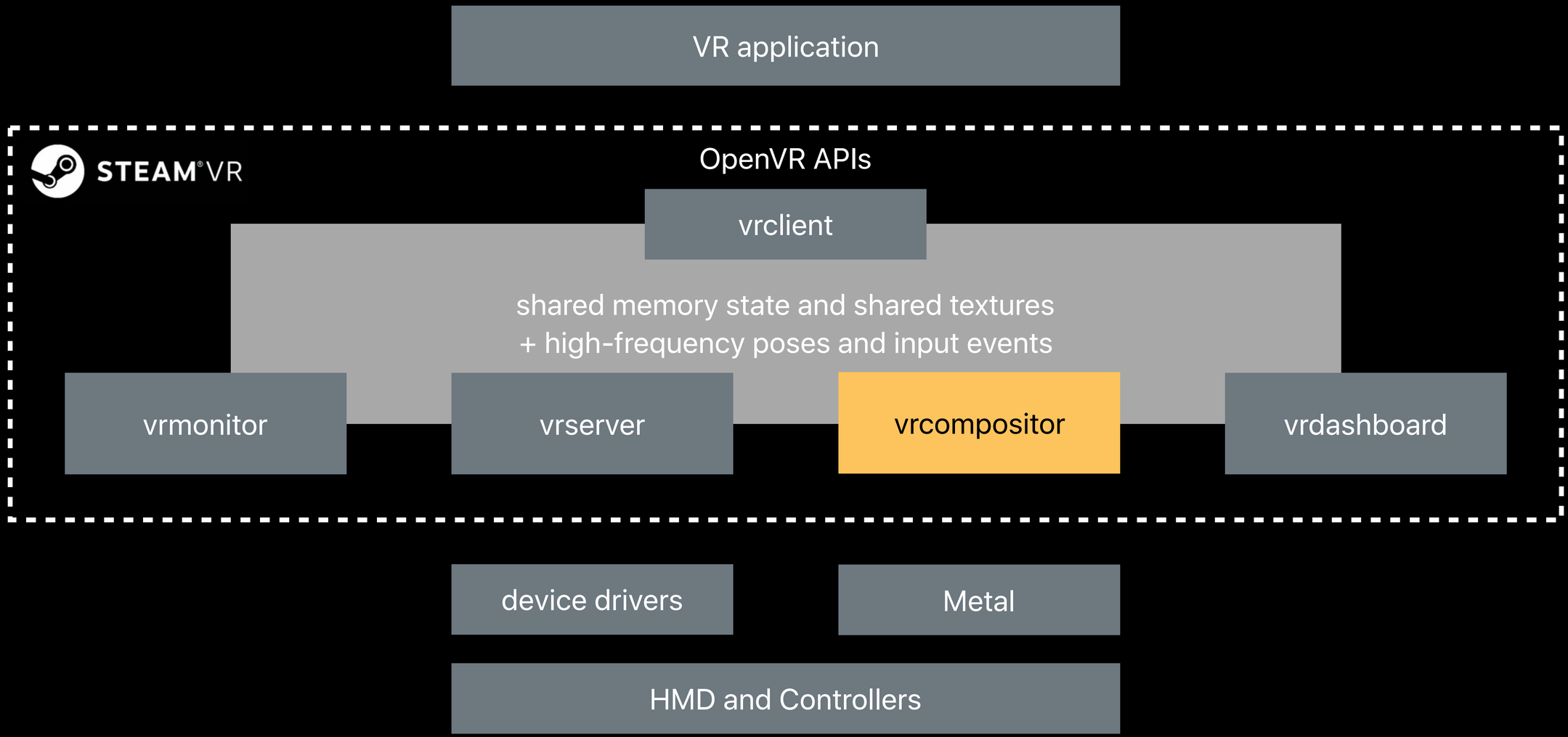
SteamVR Architecture



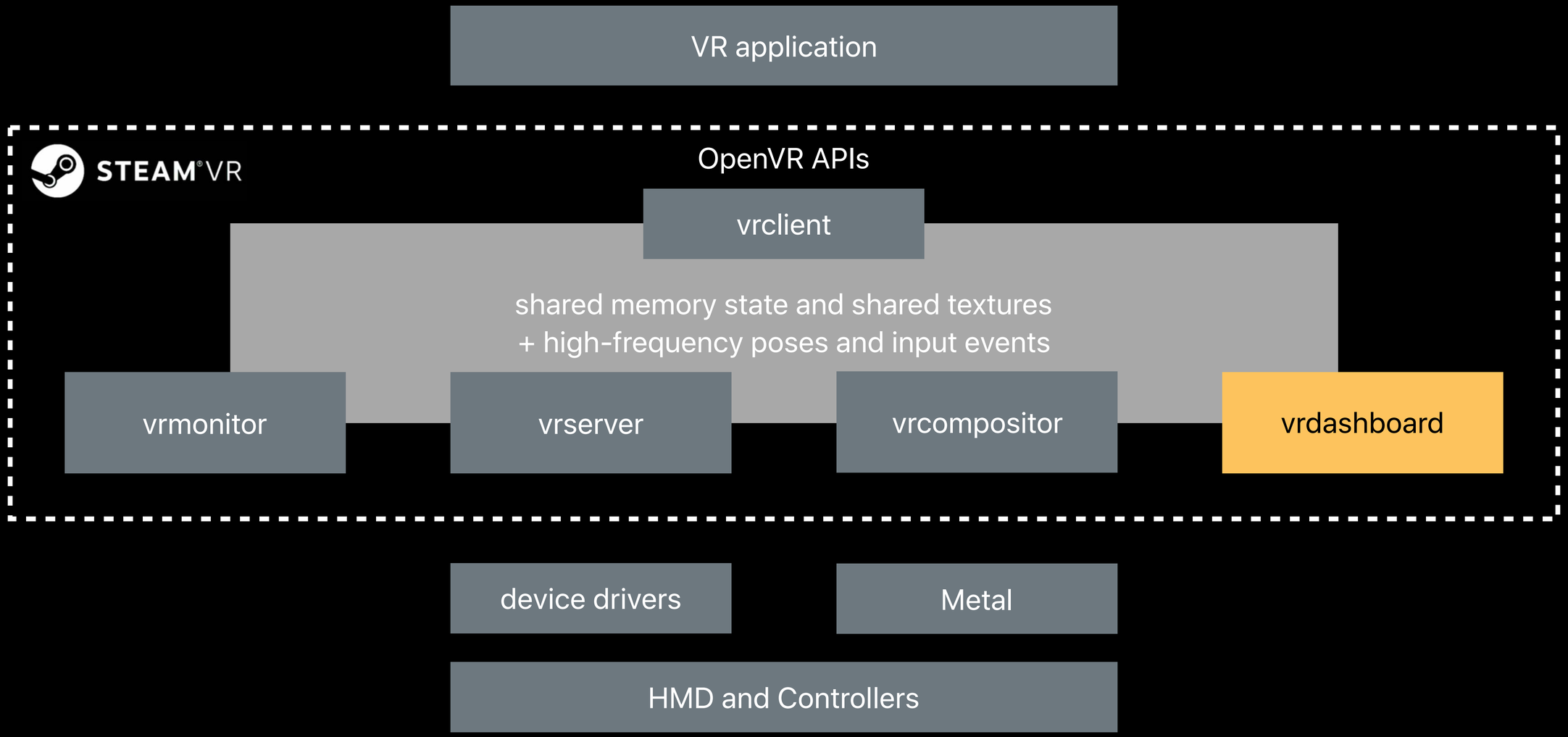
SteamVR Architecture



SteamVR Architecture



SteamVR Architecture



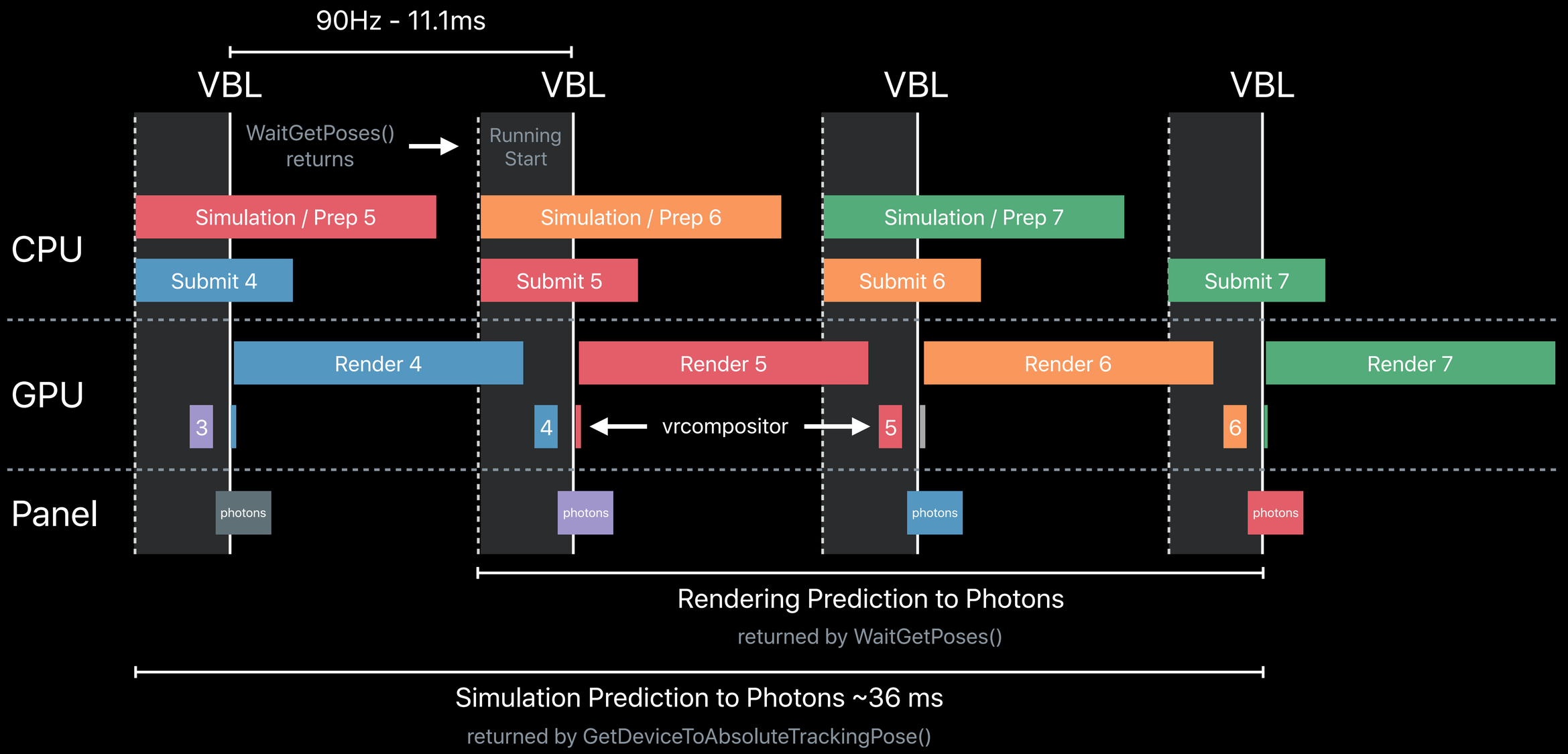
SteamVR on macOS

Closer engagement with Apple started Summer 2016

Bringing the compositor to Metal only took a few weeks

Adopted the Metal 2 Direct to Display APIs

SteamVR on macOS



Where to get SteamVR

Install Steam and create a free account

- http://store.steampowered.com/

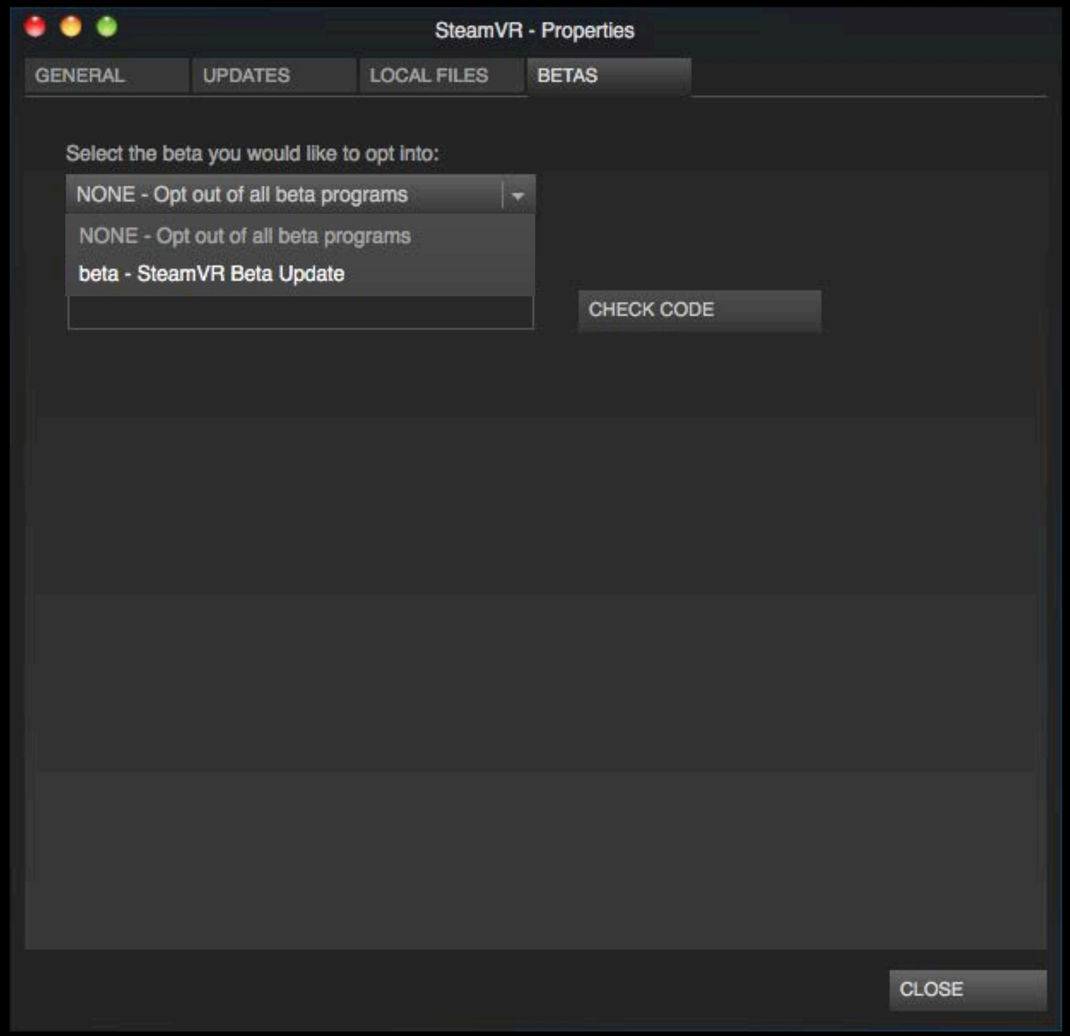
Under Library / Tools, install SteamVR

Opt into the SteamVR Beta

- Right-Click SteamVR, choose “Properties”
- Choose “beta - SteamVR Beta Update”

Download OpenVR headers & framework

- https://github.com/ValveSoftware/openvr



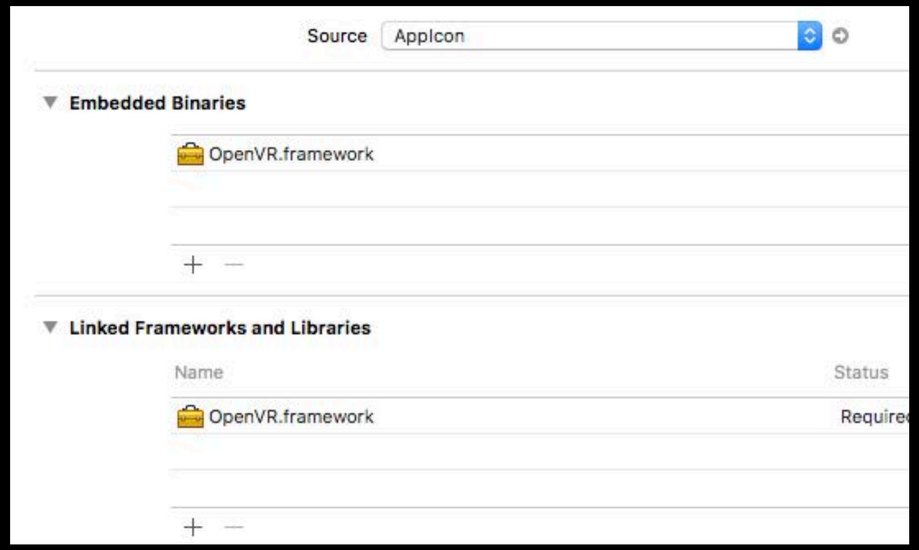
Using OpenVR in Your App

Include OpenVR.framework inside your app bundle

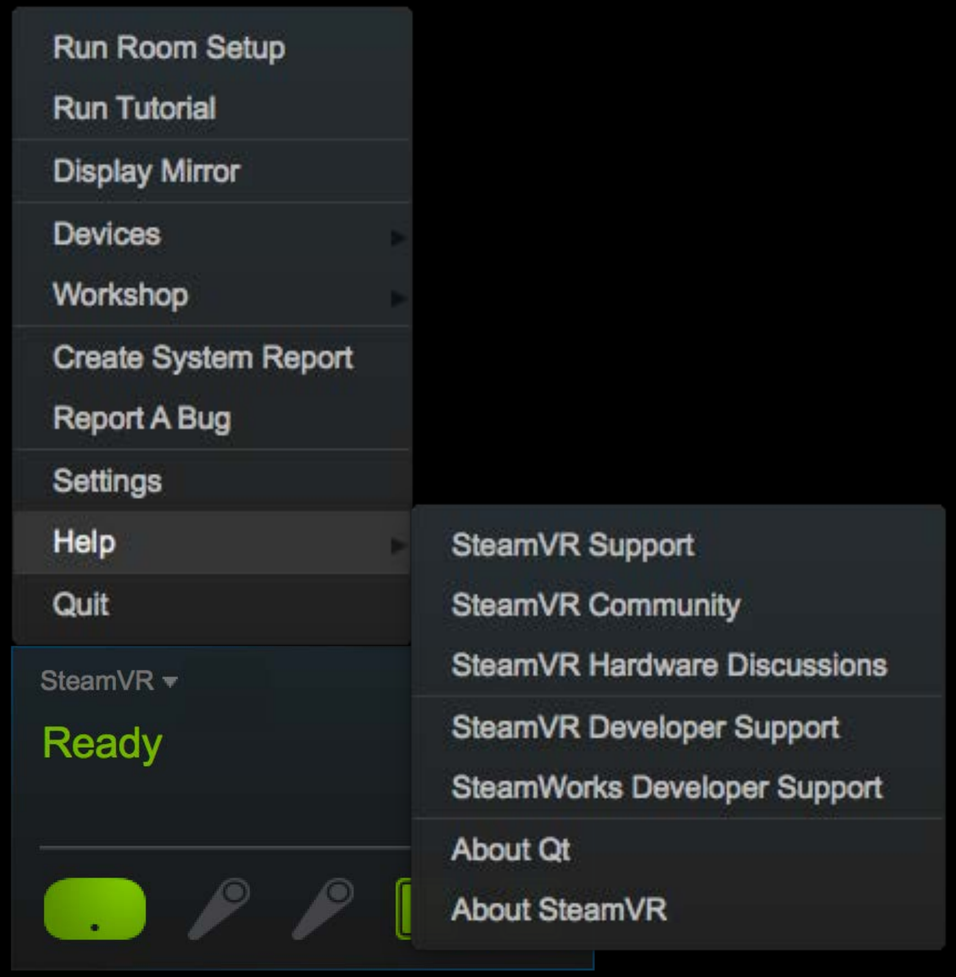
- Conveys version to the SteamVR runtime

- Add OpenVR.Framework to Embedded Binaries under General settings

- Xcode will automatically set the Runtime Search Path appropriately for your bundle



Feedback and Support



Nat Brown, Valve Software

External GPU Support

Background

External chassis with a desktop class GPU

Connected to host via Thunderbolt

Goals

Enable VR development

Goals

Enable VR development

Performance improvement in other GPU bound cases

External Graphics Developer Kit

Sonnet 350W external GPU Chassis

AMD Radeon RX 580 GPU

Optimized for Thunderbolt 3 capable Macs

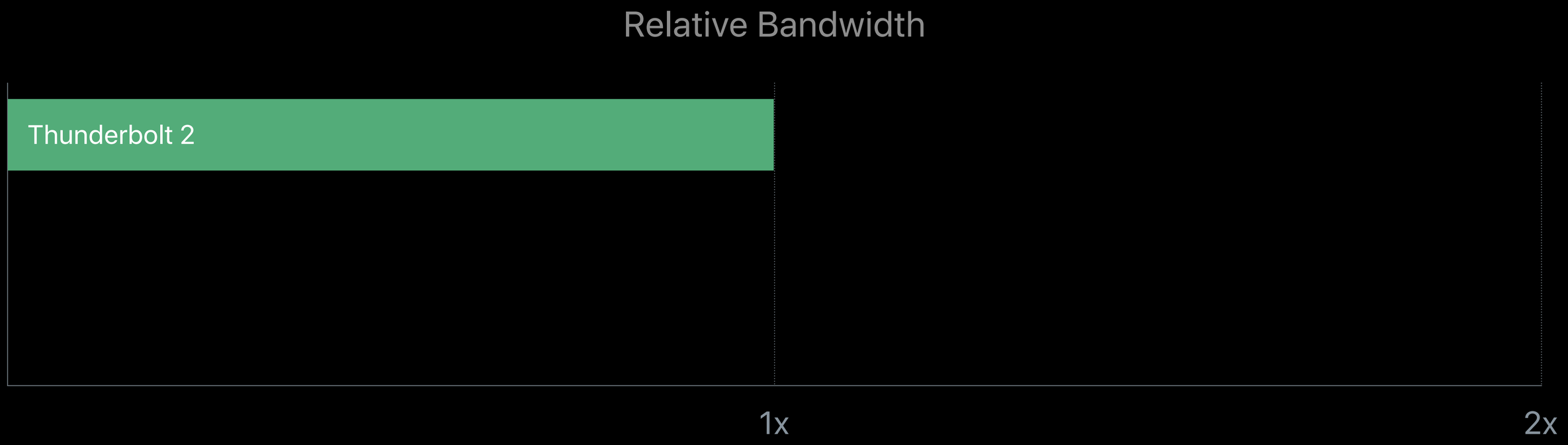
Available for purchase today

```
// Identifying the External GPU

id<MTLDevice> externalGPU = nil;
NSArray<id<MTLDevice>> * availableDevices = MTLCopyAllDevices();
for (id <MTLDevice> device in availableDevices)
{
    if (device.removable)
    {
        externalGPU = device;
       return;
    }
}
```

```
// Identifying the External GPU

id<MTLDevice> externalGPU = nil;
NSArray<id<MTLDevice>> * availableDevices = MTLCopyAllDevices();
for (id <MTLDevice> device in availableDevices)
{
    if (device.removable)
    {
        externalGPU = device;
        return;
    }
}
```

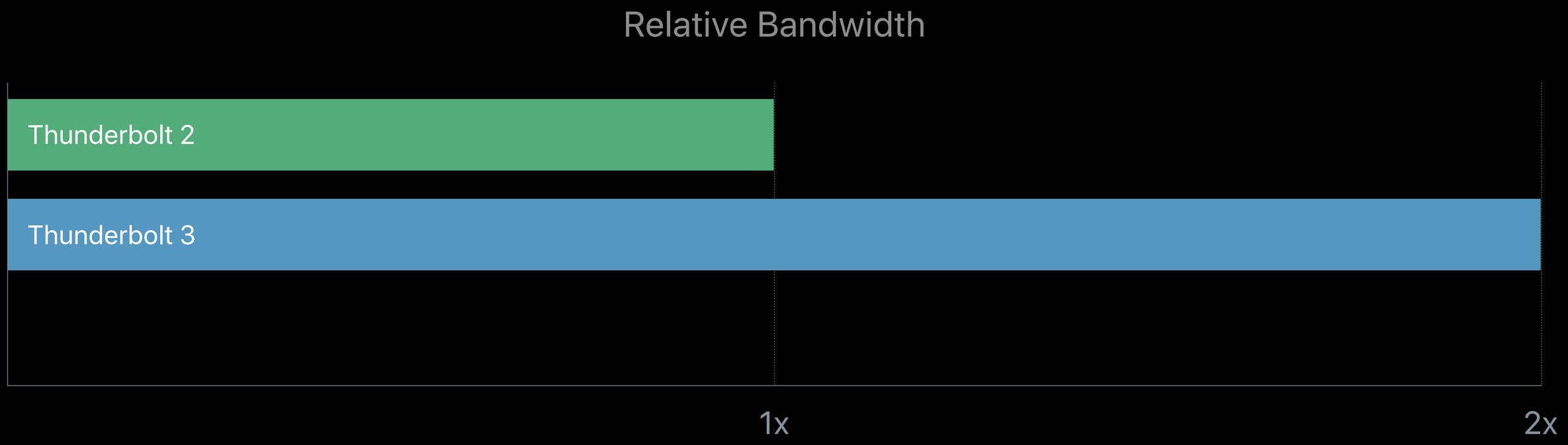
```
// Identifying the External GPU

id<MTLDevice> externalGPU = nil;
NSArray<id<MTLDevice>> * availableDevices = MTLCopyAllDevices();
for (id <MTLDevice> device in availableDevices)
{
    if (device.removable)
    {
        externalGPU = device;
        return;
    }
}
```

Thunderbolt 3 Bandwidth



Thunderbolt 3 Bandwidth

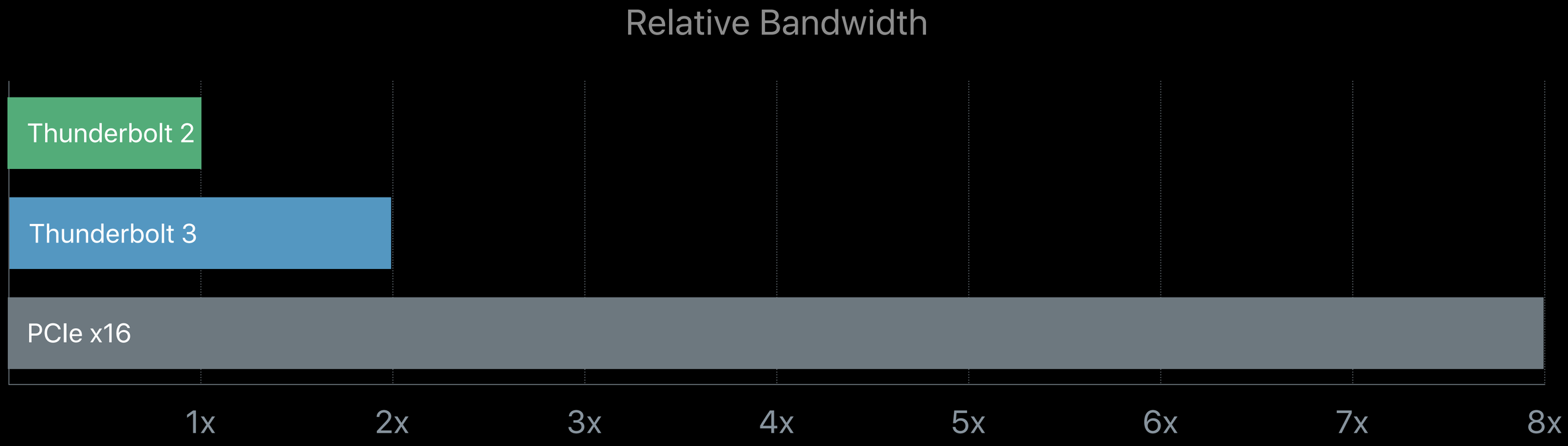
Twice the bandwidth capability of Thunderbolt 2



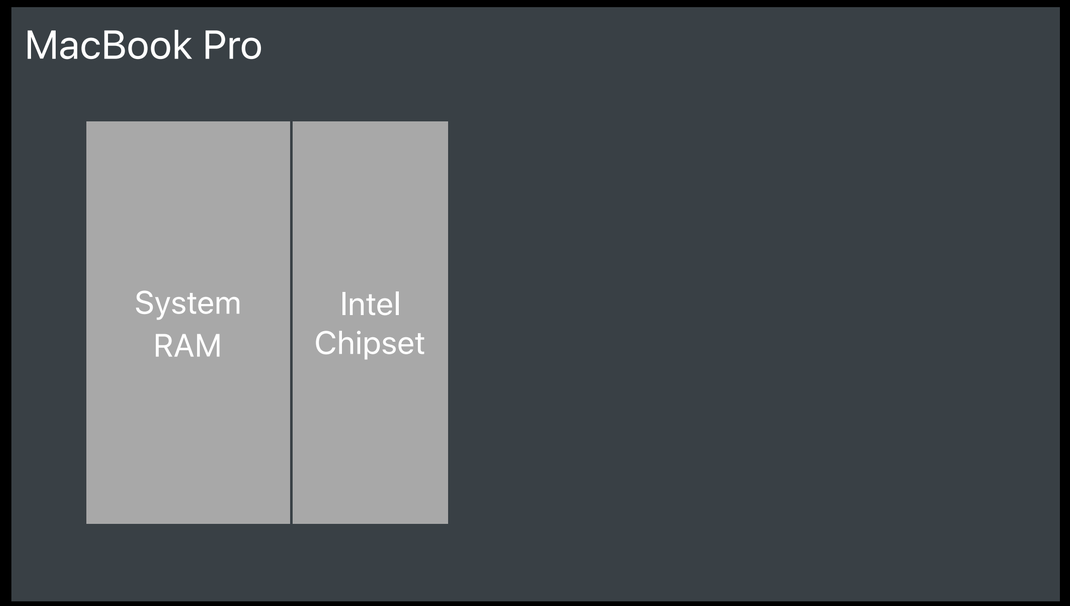
Thunderbolt 3 Bandwidth

Twice the bandwidth capability of Thunderbolt 2

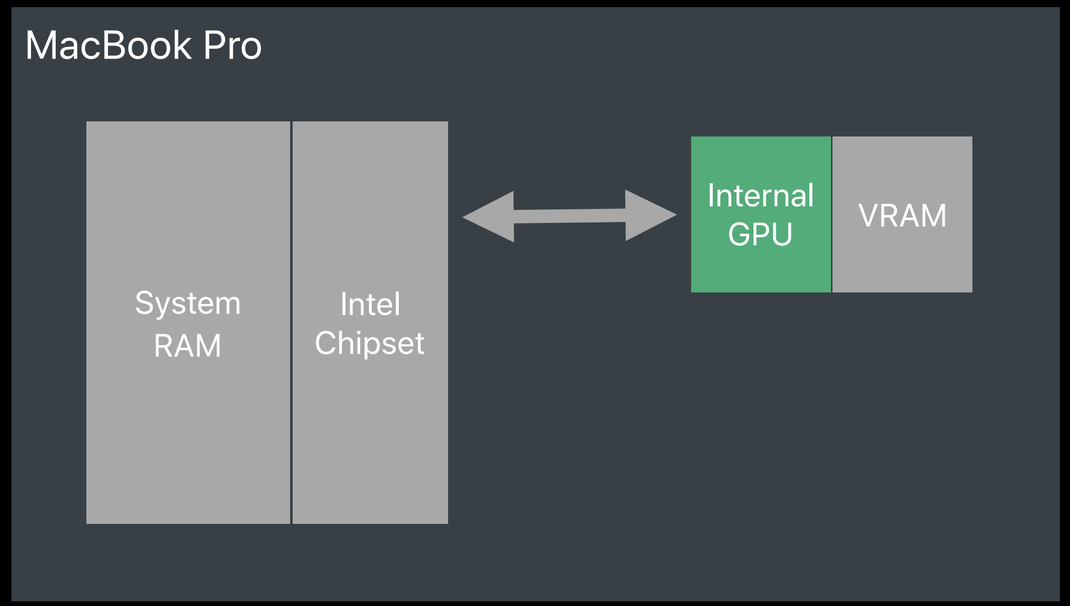
25% the bandwidth of PCIe x16



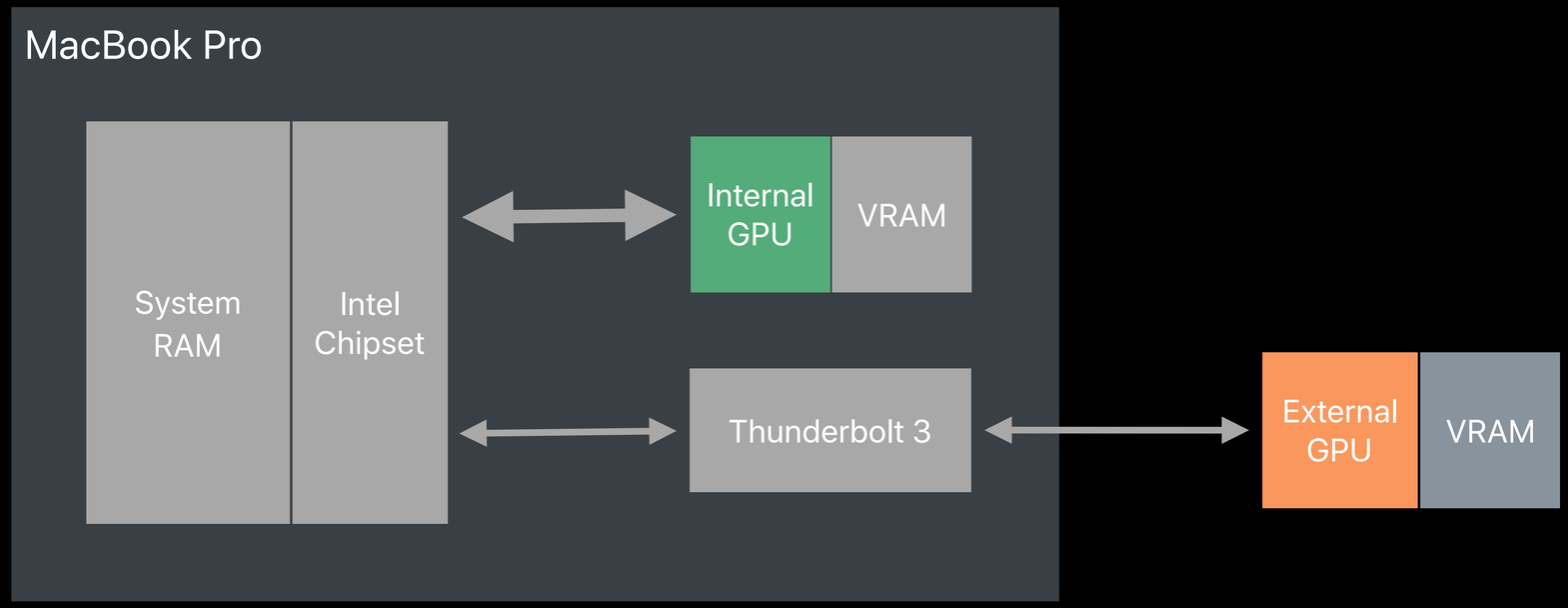
GPU Power vs. Bandwidth



GPU Power vs. Bandwidth



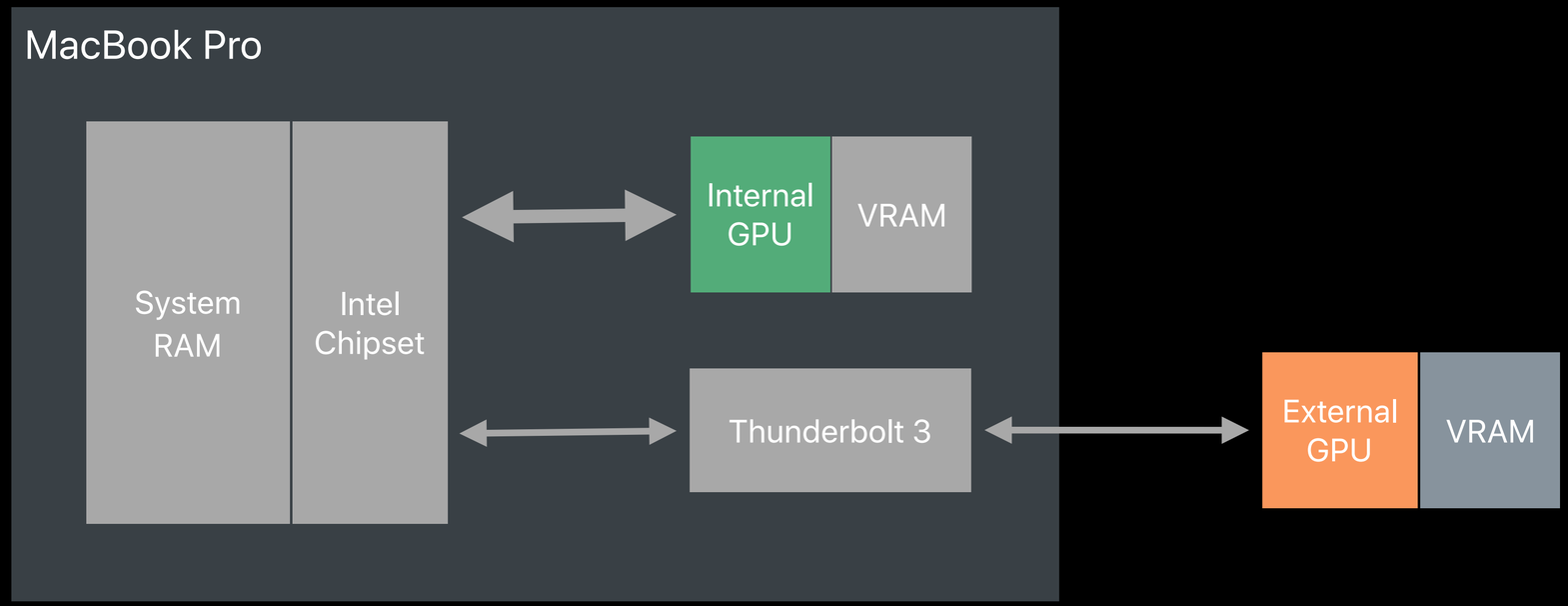
GPU Power vs. Bandwidth



GPU Power vs. Bandwidth

Treat the GPU and link as a pair

Optimal combination will depend on workload



Complex Display Topology

Complex Display Topology

Displays connected to different GPUs

Complex Display Topology

Displays connected to different GPUs

Performance impact to render on one GPU and display on another

Complex Display Topology

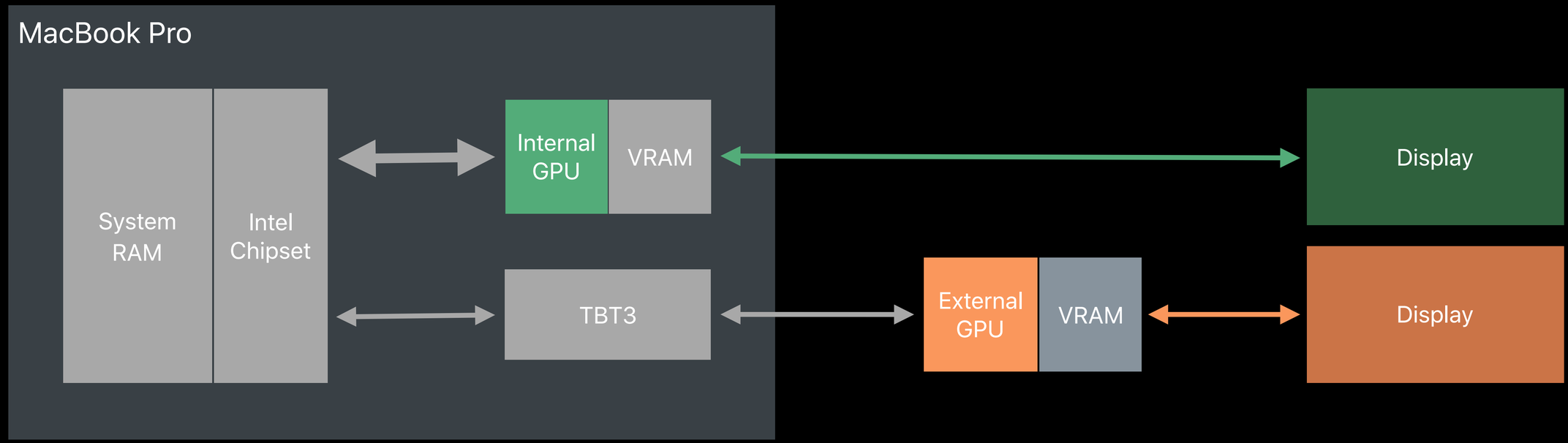
Displays connected to different GPUs

Performance impact to render on one GPU and display on another

Where your content is displayed matters

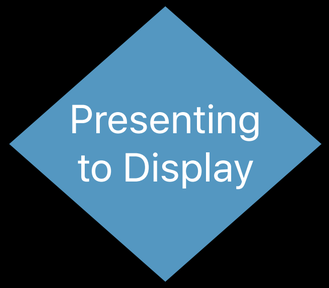
Golden Rule for GPU Selection

Render on the same GPU your app displays on

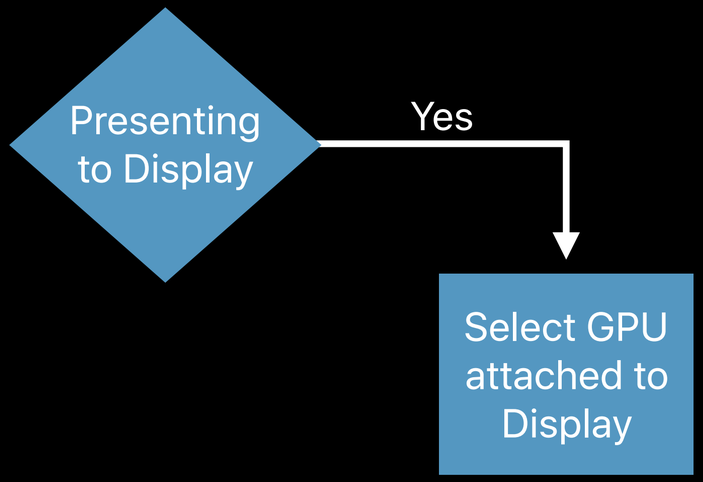


GPU Selection Decision Tree

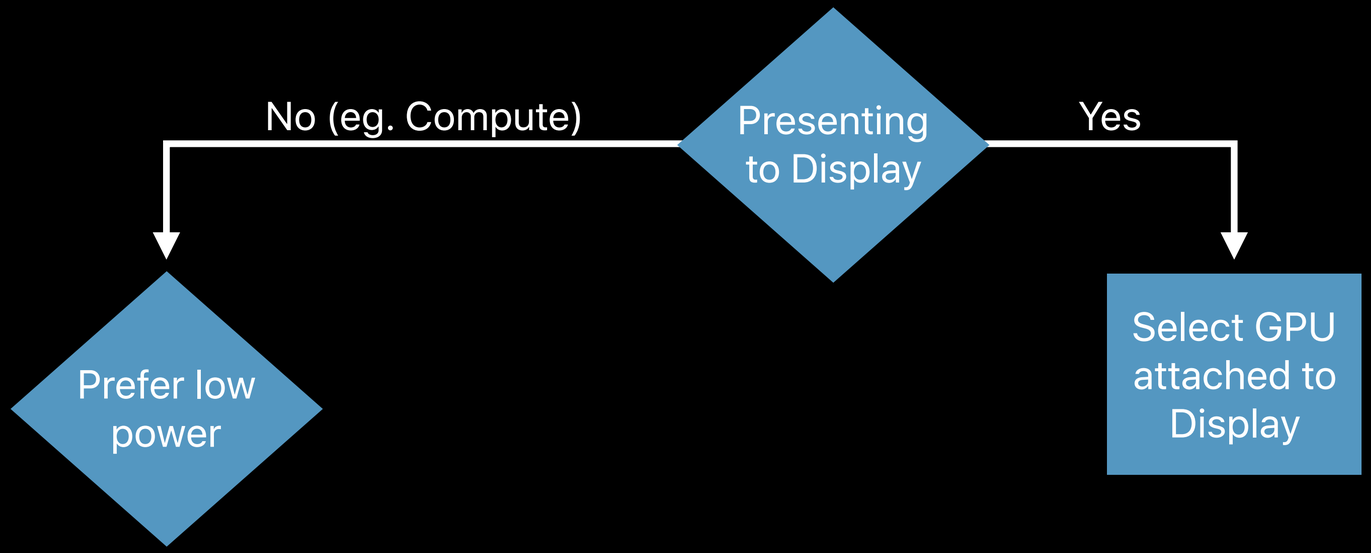
GPU Selection Decision Tree



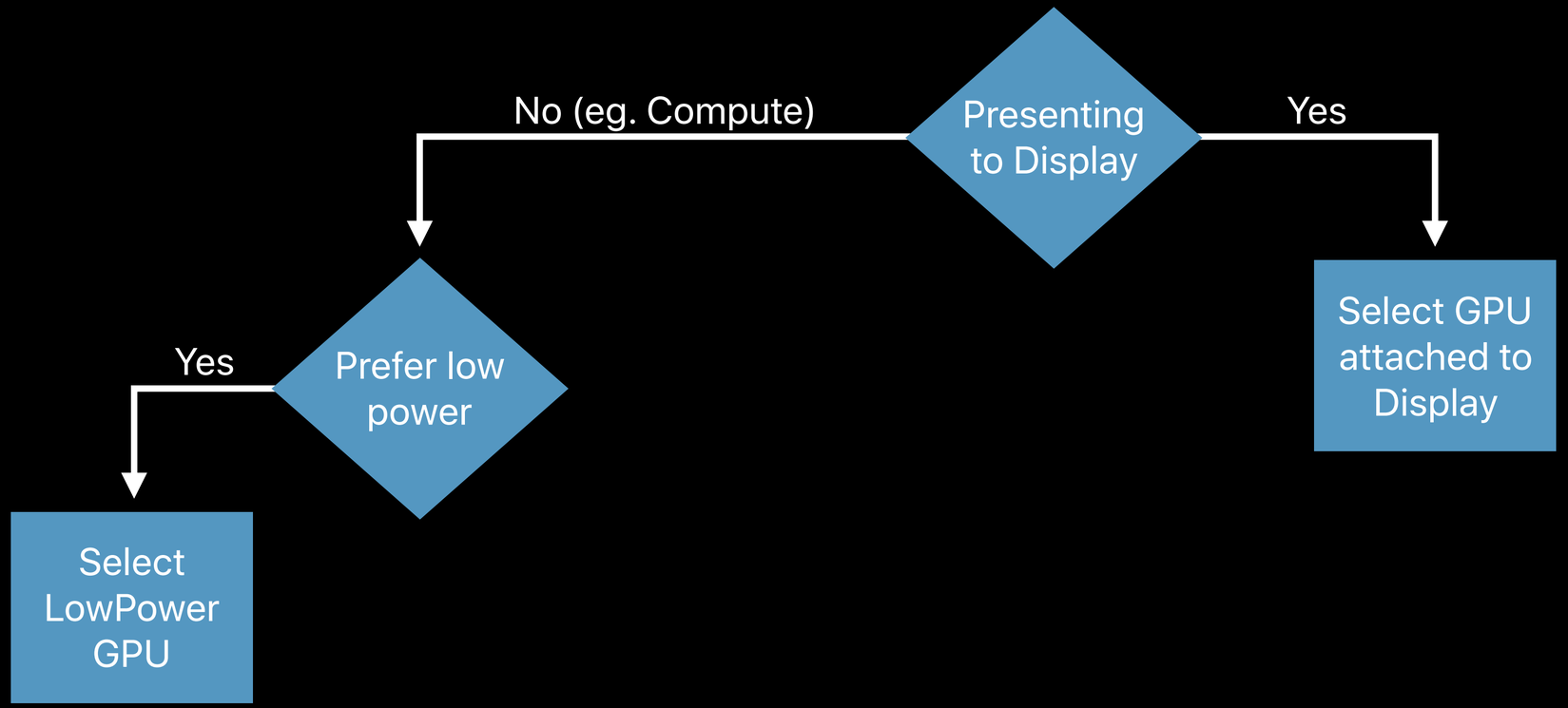
GPU Selection Decision Tree



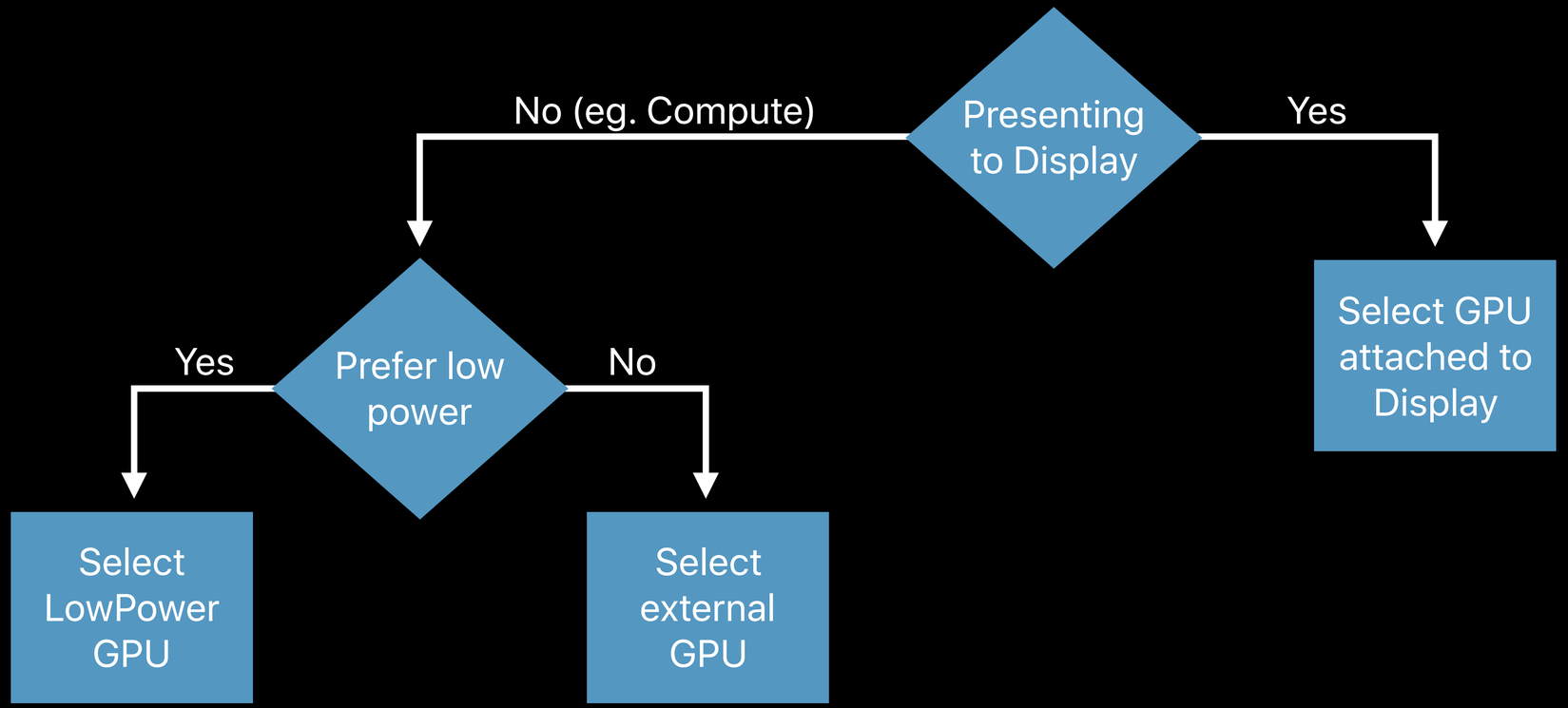
GPU Selection Decision Tree



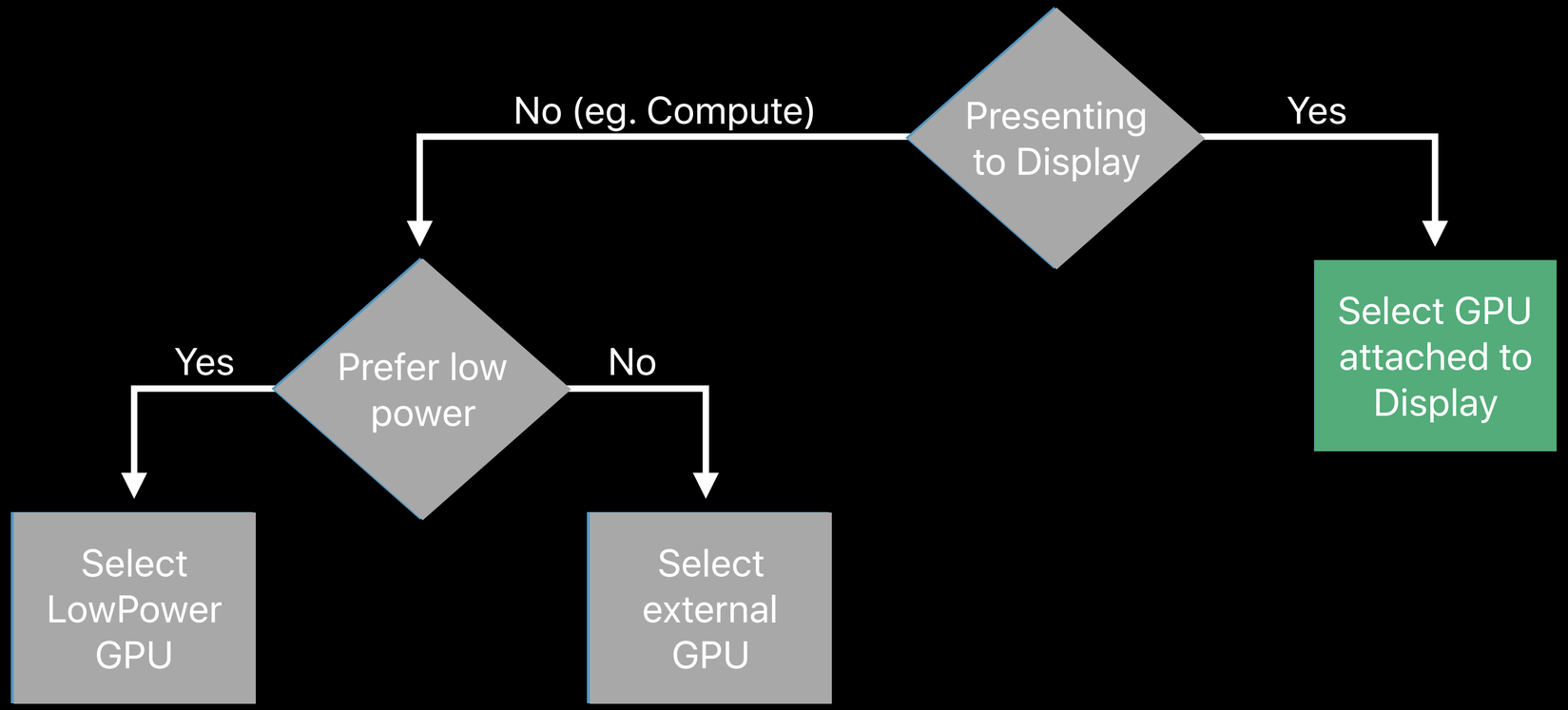
GPU Selection Decision Tree



GPU Selection Decision Tree



GPU Selection Decision Tree



Metal Device from Display

Use existing CoreGraphics API

```
// Get the CGDirectDisplayID for the display your app window is on
NSNumber* num = view.window.screen.deviceDescription[@"NSScreenNumber"];
CGDirectDisplayID viewDisplayID = [num unsignedIntegerValue];

// query CG for the metal device
id<MTLDevice> newPreferredDevice = CGDirectDisplayCopyCurrentMetalDevice(viewDisplayID);
```

Metal Device from Display

Use existing CoreGraphics API

```
// Get the CGDirectDisplayID for the display your app window is on
NSNumber* num = view.window.screen.deviceDescription[@"NSScreenNumber"];
CGDirectDisplayID viewDisplayID = [num unsignedIntegerValue];

// query CG for the metal device
id<MTLDevice> newPreferredDevice = CGDirectDisplayCopyCurrentMetalDevice(viewDisplayID);
```

Metal Device from Display

Use existing CoreGraphics API

```
// Get the CGDirectDisplayID for the display your app window is on
NSNumber* num = view.window.screen.deviceDescription[@"NSScreenNumber"];
CGDirectDisplayID viewDisplayID = [num unsignedIntegerValue];

// query CG for the metal device
id<MTLDevice> newPreferredDevice = CGDirectDisplayCopyCurrentMetalDevice(viewDisplayID);
```

GPU Migration

GPU Migration

Display migration may require GPU migration

GPU Migration

Display migration may require GPU migration

Register for NSWindowDidChangeScreenNotification

- Triggered when windows move across displays

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(handleScreenChanges:)
                                             name:NSWindowDidChangeScreenNotification
                                           object:nil];
```

```
// Get the Metal device for the GPU driving the display your app is now on
NSNumber* num = view.window.screen.deviceDescription[@"NSScreenNumber"];
CGDirectDisplayID viewDisplayID = [num unsignedIntegerValue];
id<MTLDevice> newPreferredDevice = CGDirectDisplayCopyCurrentMetalDevice(viewDisplayID);

// Early out if this display is being driven by the same GPU
if (currentDevice == newPreferredDevice)
    return;

// switch view to new device
view.device = newPreferredDevice;

// handle App migration to the new GPU

// Call draw on this new device
[MetalKitRenderer draw:view];
```

```objc
// Get the Metal device for the GPU driving the display your app is now on
NSNumber* num = view.window.screen.deviceDescription[@"NSScreenNumber"];
CGDirectDisplayID viewDisplayID = [num unsignedIntegerValue];
id<MTLDevice> newPreferredDevice = CGDirectDisplayCopyCurrentMetalDevice(viewDisplayID);

// Early out if this display is being driven by the same GPU
if (currentDevice == newPreferredDevice)
    return;

// switch view to new device
view.device = newPreferredDevice;

// handle App migration to the new GPU

// Call draw on this new device
[MetalKitRenderer draw:view];
```

```
// Get the Metal device for the GPU driving the display your app is now on
NSNumber* num = view.window.screen.deviceDescription[@"NSScreenNumber"];
CGDirectDisplayID viewDisplayID = [num unsignedIntegerValue];
id<MTLDevice> newPreferredDevice = CGDirectDisplayCopyCurrentMetalDevice(viewDisplayID);

// Early out if this display is being driven by the same GPU
if (currentDevice == newPreferredDevice)
    return;

// switch view to new device
view.device = newPreferredDevice;

// handle App migration to the new GPU

// Call draw on this new device
[MetalKitRenderer draw:view];
```

```
// Get the Metal device for the GPU driving the display your app is now on
NSNumber* num = view.window.screen.deviceDescription[@"NSScreenNumber"];
CGDirectDisplayID viewDisplayID = [num unsignedIntegerValue];
id<MTLDevice> newPreferredDevice = CGDirectDisplayCopyCurrentMetalDevice(viewDisplayID);

// Early out if this display is being driven by the same GPU
if (currentDevice == newPreferredDevice)
    return;

// switch view to new device
view.device = newPreferredDevice;

// handle App migration to the new GPU

// Call draw on this new device
[MetalKitRenderer draw:view];
```

Attach and Removal Notifications

New with Metal 2

NEW

MTLDeviceWasAddedNotification

MTLDeviceWasRemovedNotification

MTLDeviceRemovalRequestedNotification

NEW

```
NSArray< id<MTLDevice> >* devices = nil;
id<NSObject> metalDeviceObserver = nil;
devices = MTLCopyAllDevicesWithObserver(&metalDeviceObserver,
                                        ^(id<MTLDevice> device, MTLNotificationName name)
                                        {
                                            [self handleGPUHotPlug:device notifier:name];
                                        });
```

NEW

```
NSArray< id<MTLDevice> >* devices = nil;
id<NSObject> metalDeviceObserver = nil;
devices = MTLCopyAllDevicesWithObserver(&metalDeviceObserver,
                                        ^(id<MTLDevice> device, MTLNotificationName name)
                                        {
                                            [self handleGPUHotPlug:device notifier:name];
                                        });
```

NEW

```
NSArray< id<MTLDevice> >* devices = nil;
id<NSObject> metalDeviceObserver = nil;
devices = MTLCopyAllDevicesWithObserver(&metalDeviceObserver,
                                        ^(id<MTLDevice> device, MTLNotificationName name)
                                        {
                                            [self handleGPUHotPlug:device notifier:name];
                                        });
```

NEW

```
- (void)handleGPUHotPlug:(id<MTLDevice>)device notifier:(MTLNotificationName)notifier
{
    if (notifier == MTLDeviceWasAddedNotification)
        // Device plugged in
    else if (notifier == MTLDeviceRemovalRequestedNotification)
        // Device Removal Requested. Cleanup and switch to preferred device
    else if (notifier == MTLDeviceWasRemovedNotification)
        // additional handling of surprise removal
}
```

NEW

```
- (void)handleGPUHotPlug:(id<MTLDevice>)device notifier:(MTLNotificationName)notifier
{
    if (notifier == MTLDeviceWasAddedNotification)
        // Device plugged in
    else if (notifier == MTLDeviceRemovalRequestedNotification)
        // Device Removal Requested. Cleanup and switch to preferred device
    else if (notifier == MTLDeviceWasRemovedNotification)
        // additional handling of surprise removal
}
```

NEW

```
- (void)handleGPUHotPlug:(id<MTLDevice>)device notifier:(MTLNotificationName)notifier
{
    if (notifier == MTLDeviceWasAddedNotification)
        // Device plugged in
    else if (notifier == MTLDeviceRemovalRequestedNotification)
        // Device Removal Requested. Cleanup and switch to preferred device
    else if (notifier == MTLDeviceWasRemovedNotification)
        // additional handling of surprise removal
}
```

Unexpected GPU Removal

Unexpected GPU Removal

Metal API will return errors on GPU removal

Survive until a GPU migration notification

Regenerate data in GPU local memory

Best Practices

Best Practices

Retain devices after migration

Best Practices

Retain devices after migration

Manage GPU migration for each window

Best Practices

Retain devices after migration

Manage GPU migration for each window

Avoid transferring data between GPUs

VR Best Practices

VR Best Practices

Attach the VR headset to the external GPU

VR Best Practices

Attach the VR headset to the external GPU

Present to a display driven by the external GPU

VR Best Practices

Attach the VR headset to the external GPU

Present to a display driven by the external GPU

Cache resources on the external GPU

Summary

VR development enabled with Metal 2

Support for HTC Vive and SteamVR

VR enabled game engines or build a native VR app

External GPU support

Related Sessions

Introducing Metal 2	Executive Ballroom	Tuesday 1:50PM
Metal 2 Optimization and Debugging	Executive Ballroom	Thursday 3:15PM
Using Metal 2 for Compute	Grand Ballroom A	Thursday 4:10PM

Metal Labs

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


WWDC17