#WWDC18

Metal for VR

Session 611

Karol Gasiński, GPU Software

What’s new in VR?

What’s new in VR?

New Metal features

What’s new in VR?

New Metal features

VR advanced techniques

VR on macOS

VR on macOS

VR on macOS

VR on macOS

VR on macOS

VR on macOS

Support for HTC Vive Pro

VR on macOS

Support for HTC Vive Pro

- Two 1440x1600 AMOLED displays, at 615 ppi

VR on macOS

Support for HTC Vive Pro

- Two 1440x1600 AMOLED displays, at 615 ppi
- 78% resolution increase

VR on macOS

Support for HTC Vive Pro

- Two 1440x1600 AMOLED displays, at 615 ppi
- 78% resolution increase
- 37% ppi increase

VR on macOS

Support for HTC Vive Pro

- Two 1440x1600 AMOLED displays, at 615 ppi
- 78% resolution increase
- 37% ppi increase
- Stereo cameras for MR

VR on macOS

Support for HTC Vive Pro

- Two 1440x1600 AMOLED displays, at 615 ppi
- 78% resolution increase
- 37% ppi increase
- Stereo cameras for MR

SteamVR Tracking System 2.0

VR on macOS

VR on macOS

Valve SteamVR runtime

VR on macOS

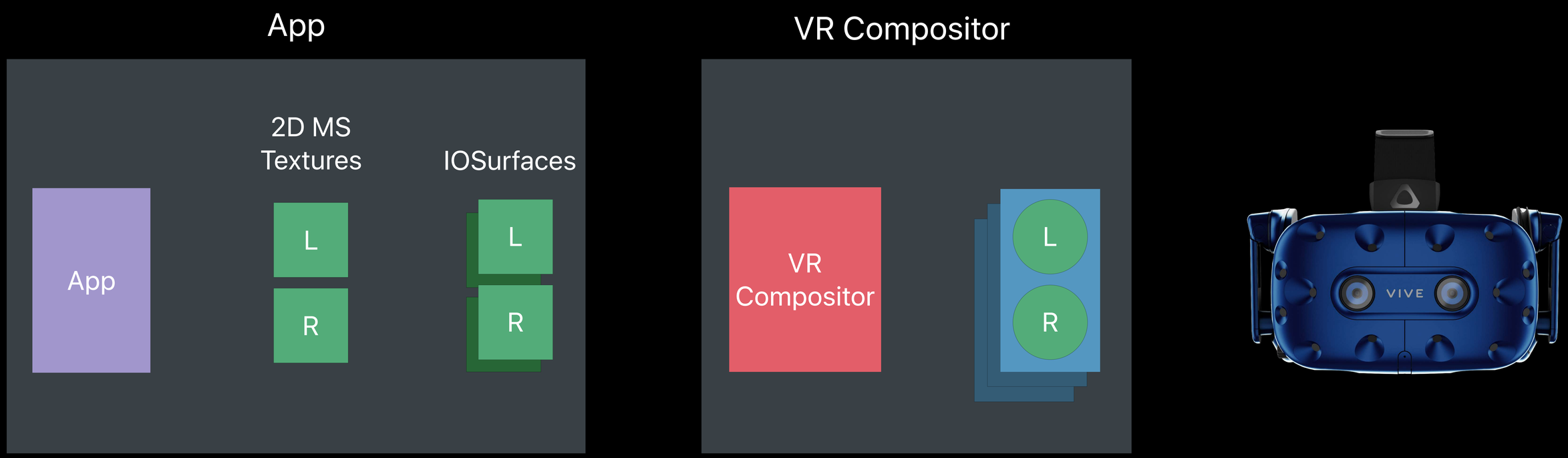
Valve SteamVR runtime

Valve OpenVR framework

Metal 2 Features for VR

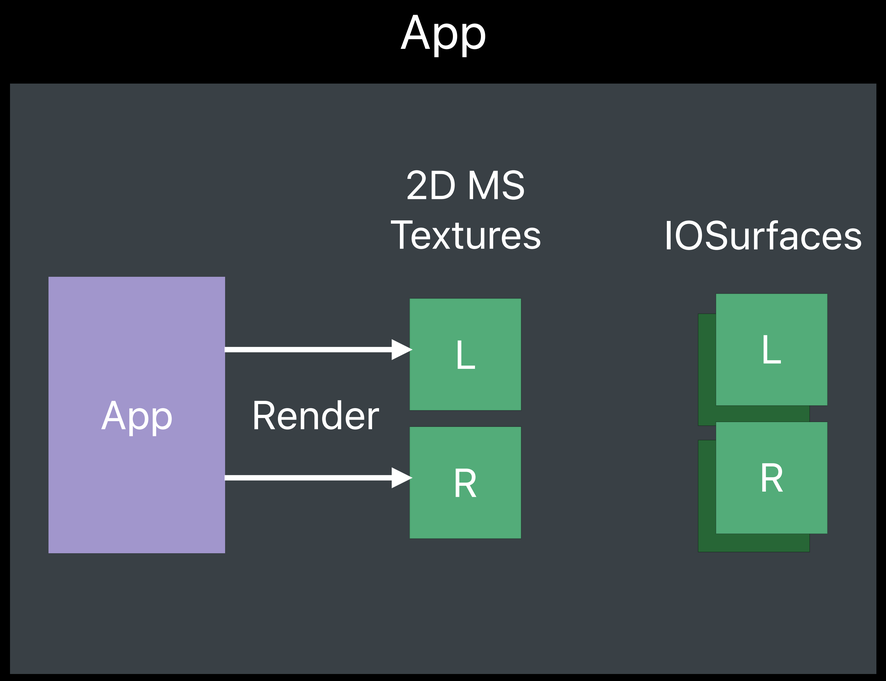
VR Application

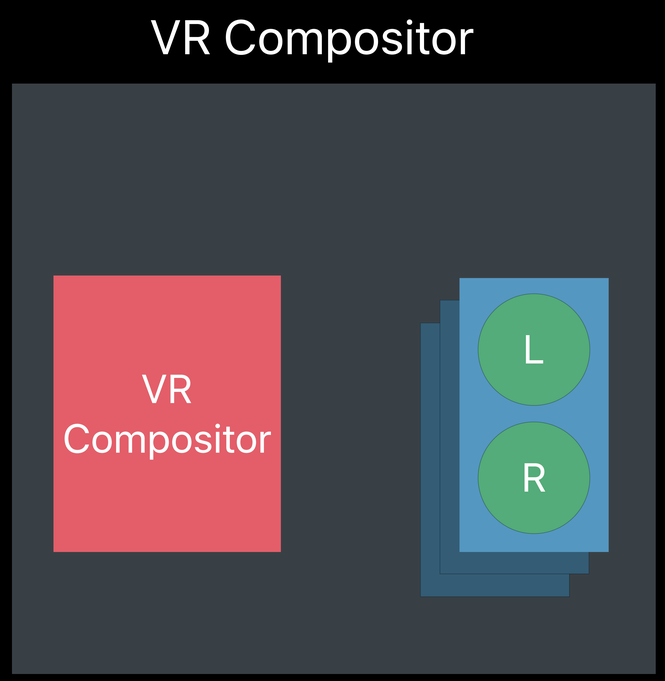
Rendering overview



VR Application

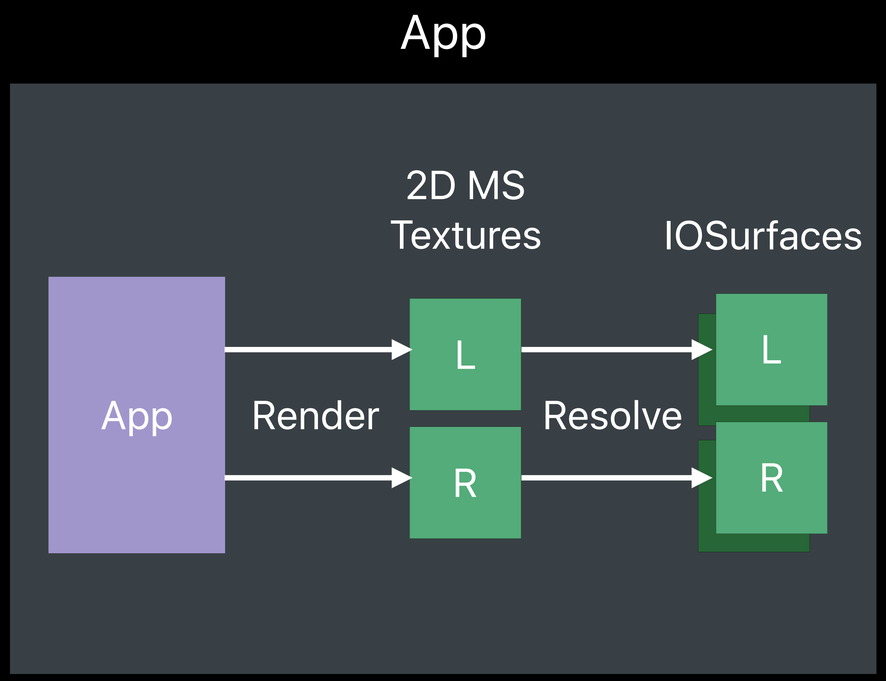
Rendering overview

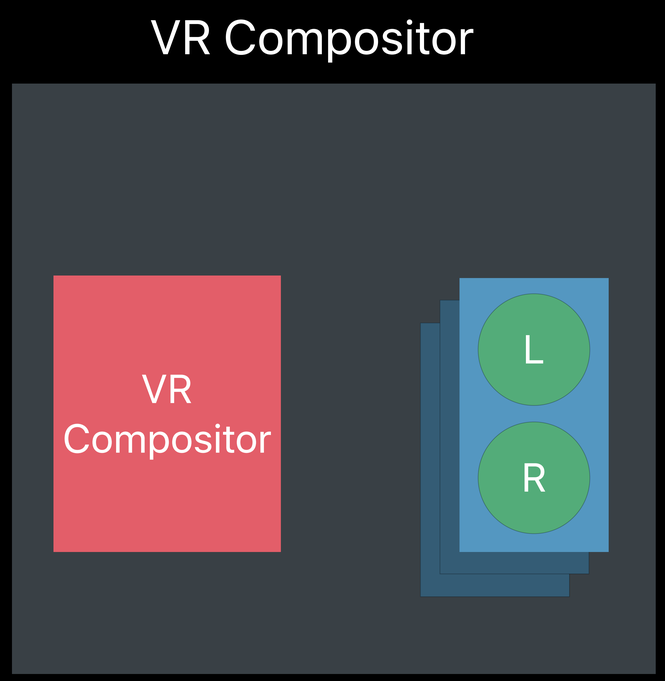




VR Application

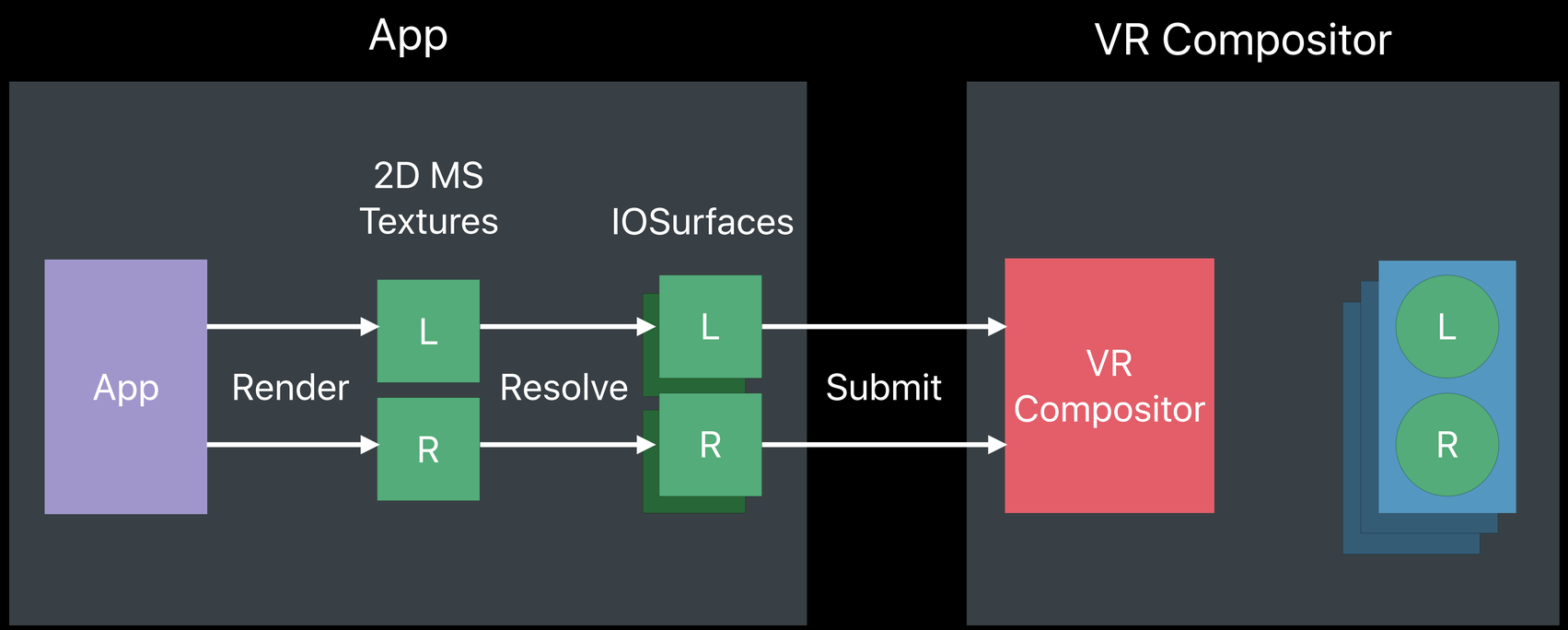
Rendering overview





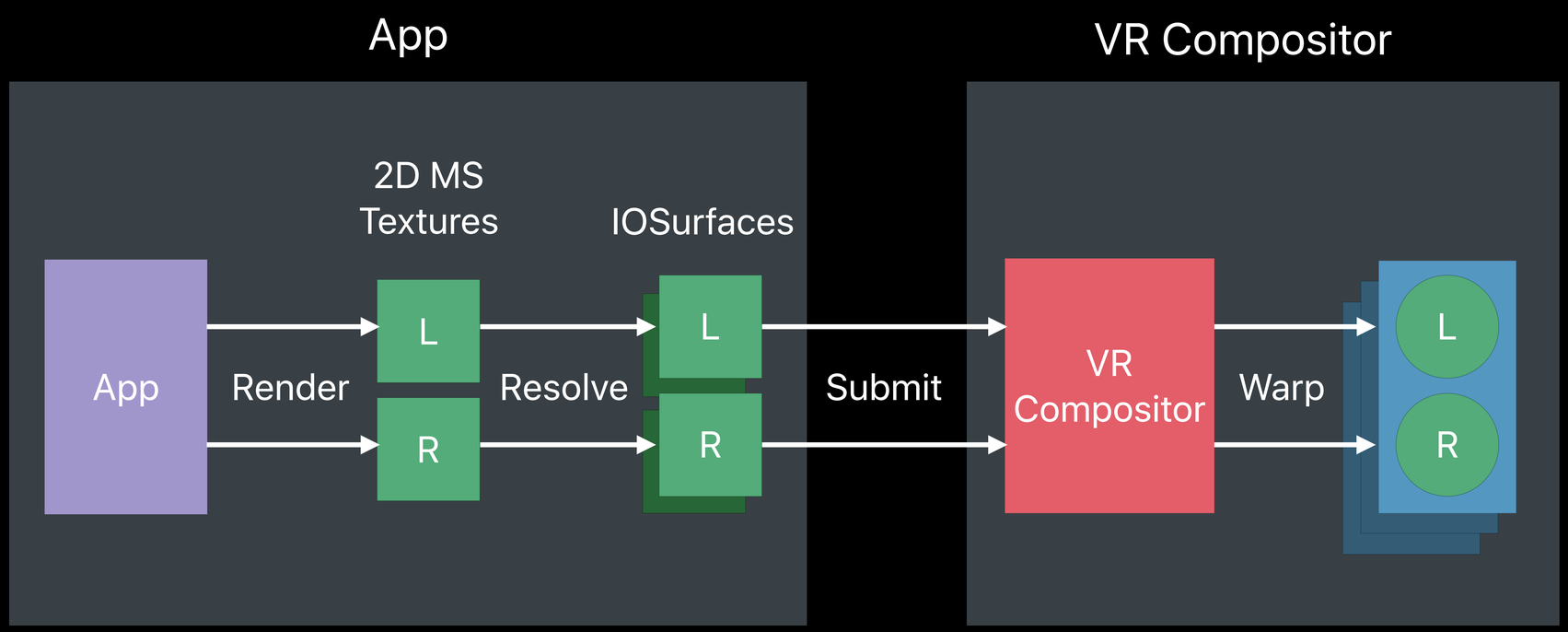
VR Application

Rendering overview



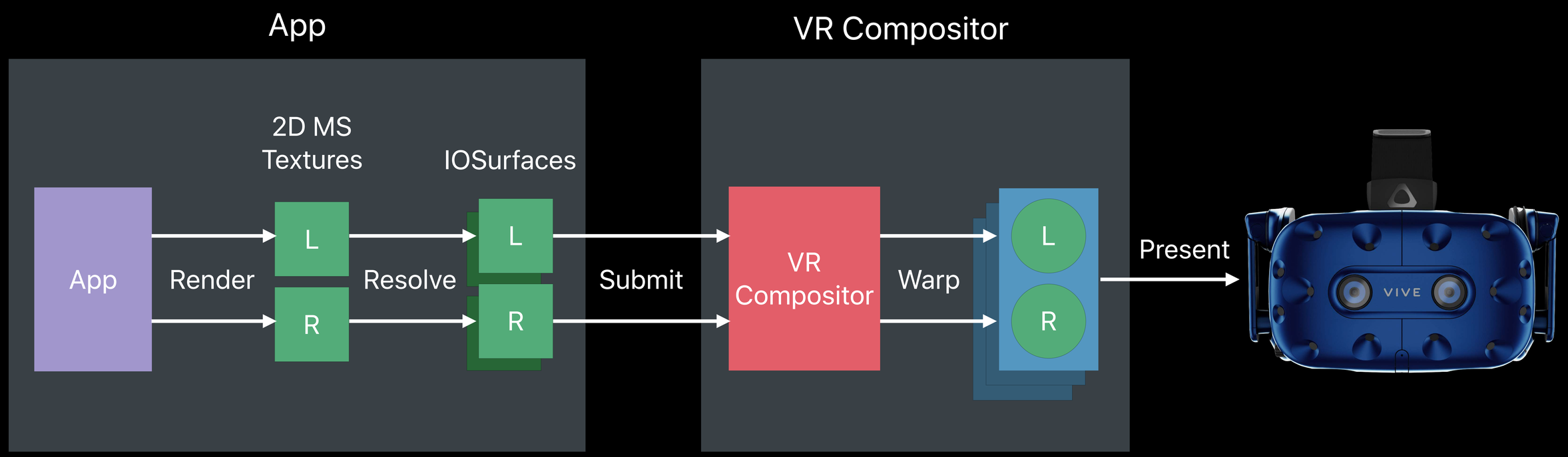
VR Application

Rendering overview



VR Application

Rendering overview



Rendering Layouts

Different rendering patterns depending on MSAA

Rendering Layouts

Different rendering patterns depending on MSAA

Dedicated

Drawbacks

2x Draw calls

2x Render passes

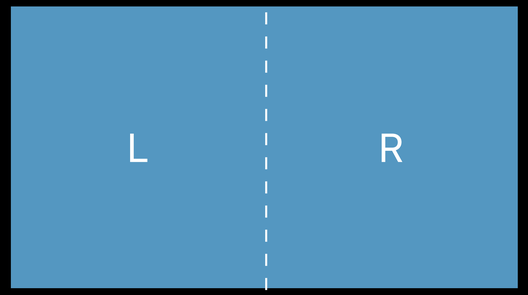
2x Resolves

Rendering Layouts

Different rendering patterns depending on MSAA

Dedicated

Shared



Drawbacks

2x Draw calls

2x Render passes

2x Resolves

Rendering Layouts

Different rendering patterns depending on MSAA

Dedicated

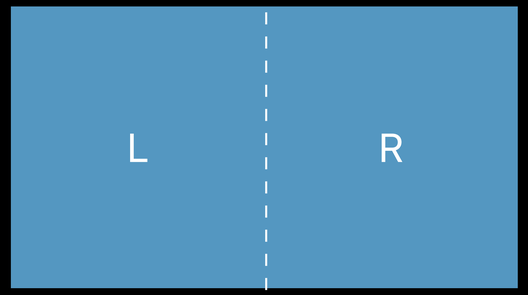
Drawbacks

2x Draw calls

2x Render passes

2x Resolves

Shared



Drawbacks

Issues with

Multi-resolution shading

Post-processes

Rendering Layouts

Different rendering patterns depending on MSAA

Dedicated

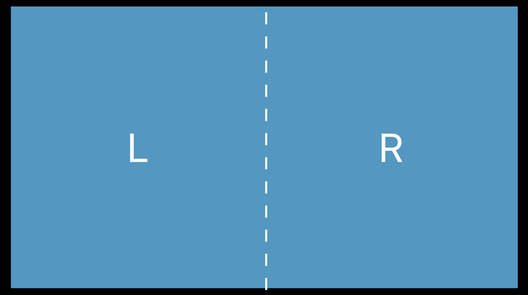
Drawbacks

2x Draw calls

2x Render passes

2x Resolves

Shared



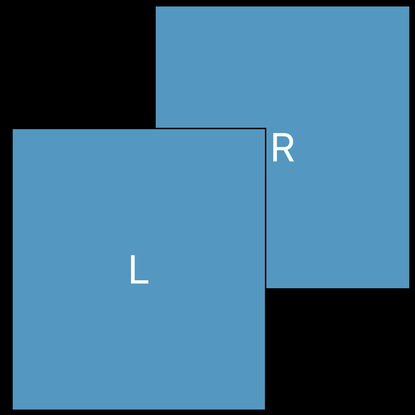
Drawbacks

Issues with

Multi-resolution shading

Post-processes

Layered



Rendering Layouts

Different rendering patterns depending on MSAA

Dedicated

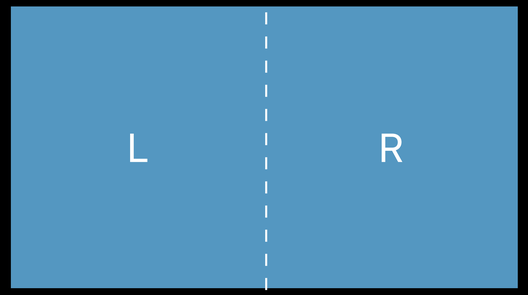
Drawbacks

2x Draw calls

2x Render passes

2x Resolves

Shared



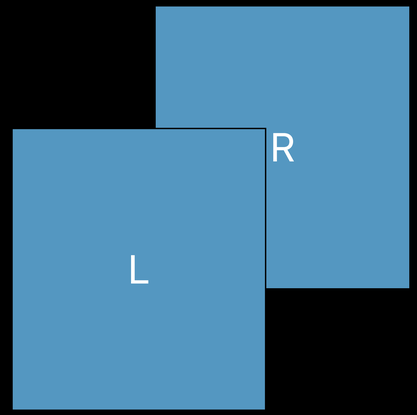
Drawbacks

Issues with

Multi-resolution shading

Post-processes

Layered



Drawbacks

Cannot be used

today for MSAA

Rendering Layouts

Different rendering patterns unified

NEW

Rendering Layouts

Different rendering patterns unified

NEW

New Metal texture type

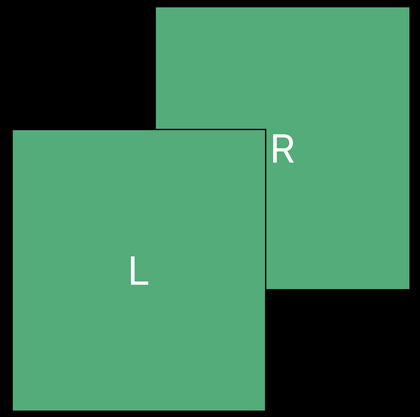
Rendering Layouts

Different rendering patterns unified

NEW

New Metal texture type

`MTLTextureType2DMultisampleArray`



2DMultisampleArray

Rendering Layouts

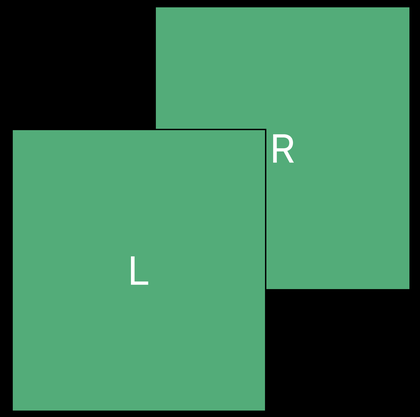
Different rendering patterns unified

NEW

New Metal texture type

`MTLTextureType2DMultisampleArray`

Separate control of



2DMultisampleArray

Rendering Layouts

Different rendering patterns unified

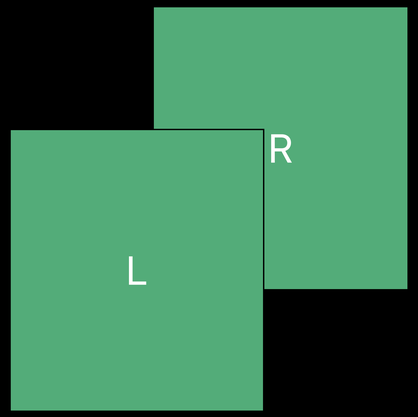
NEW

New Metal texture type

`MTLTextureType2DMultisampleArray`

Separate control of

- Rendering space



2DMultisampleArray

Rendering Layouts

Different rendering patterns unified

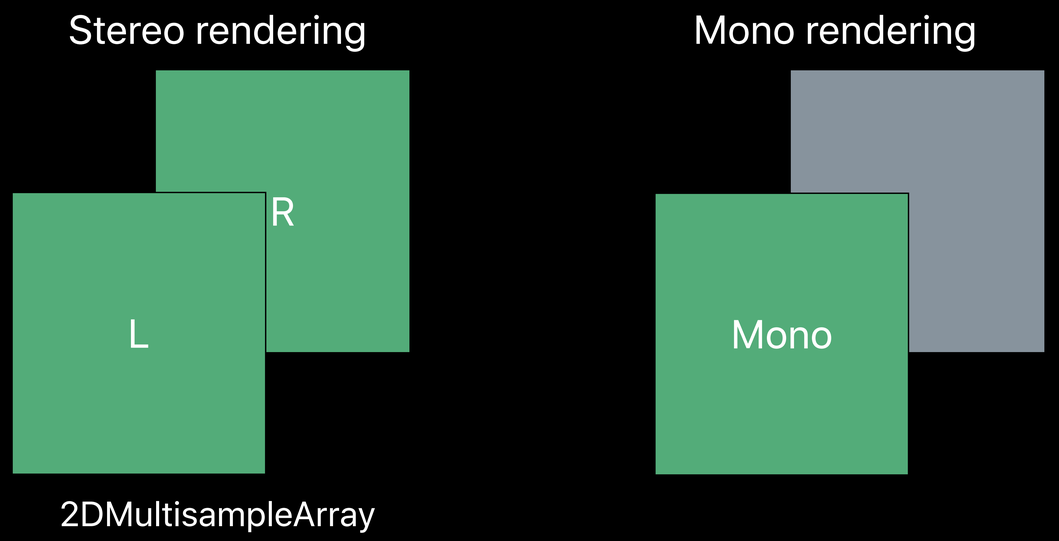
NEW

New Metal texture type

`MTLTextureType2DMultisampleArray`

Separate control of

- Rendering space
- Views count



Rendering Layouts

Different rendering patterns unified

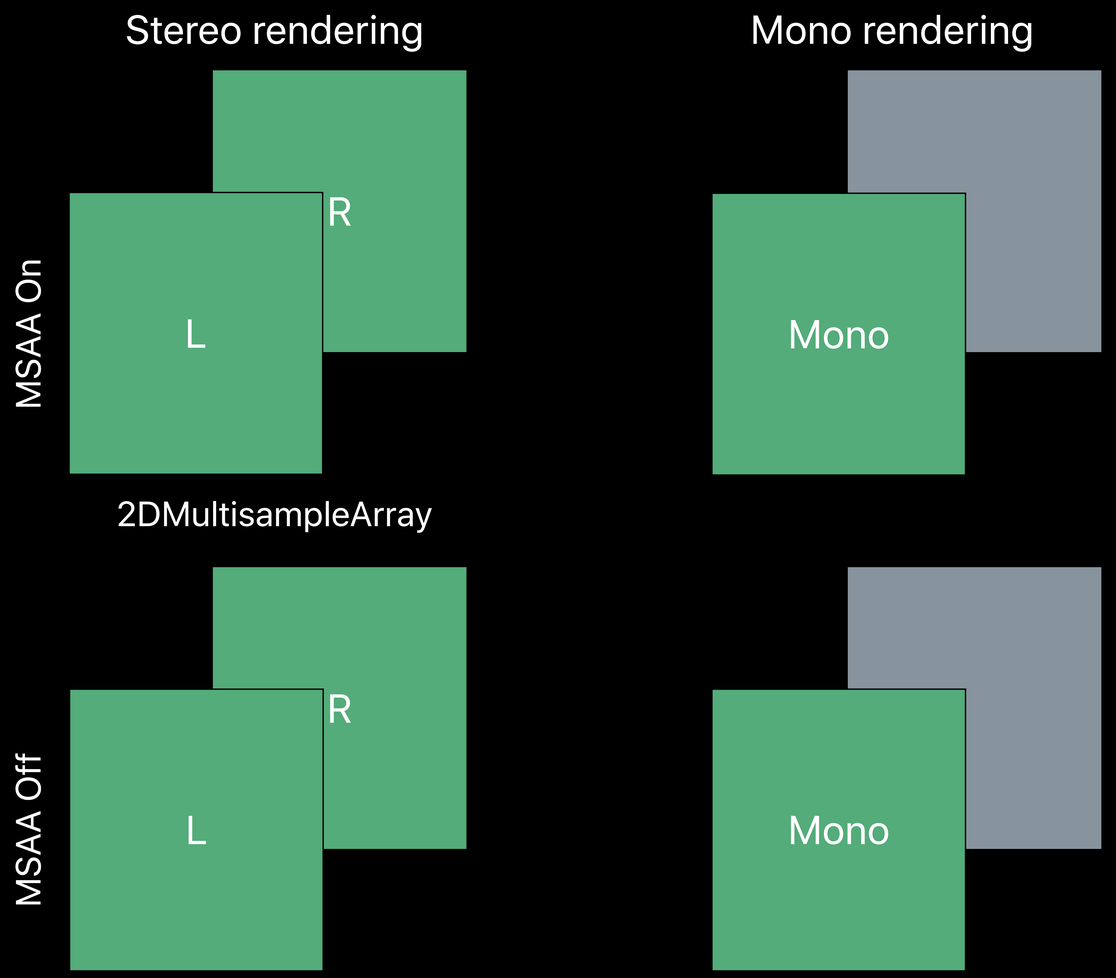
NEW

New Metal texture type

`MTLTextureType2DMultisampleArray`

Separate control of

- Rendering space
- Views count
- Anti-aliasing mode



Rendering Layouts

Different rendering patterns unified

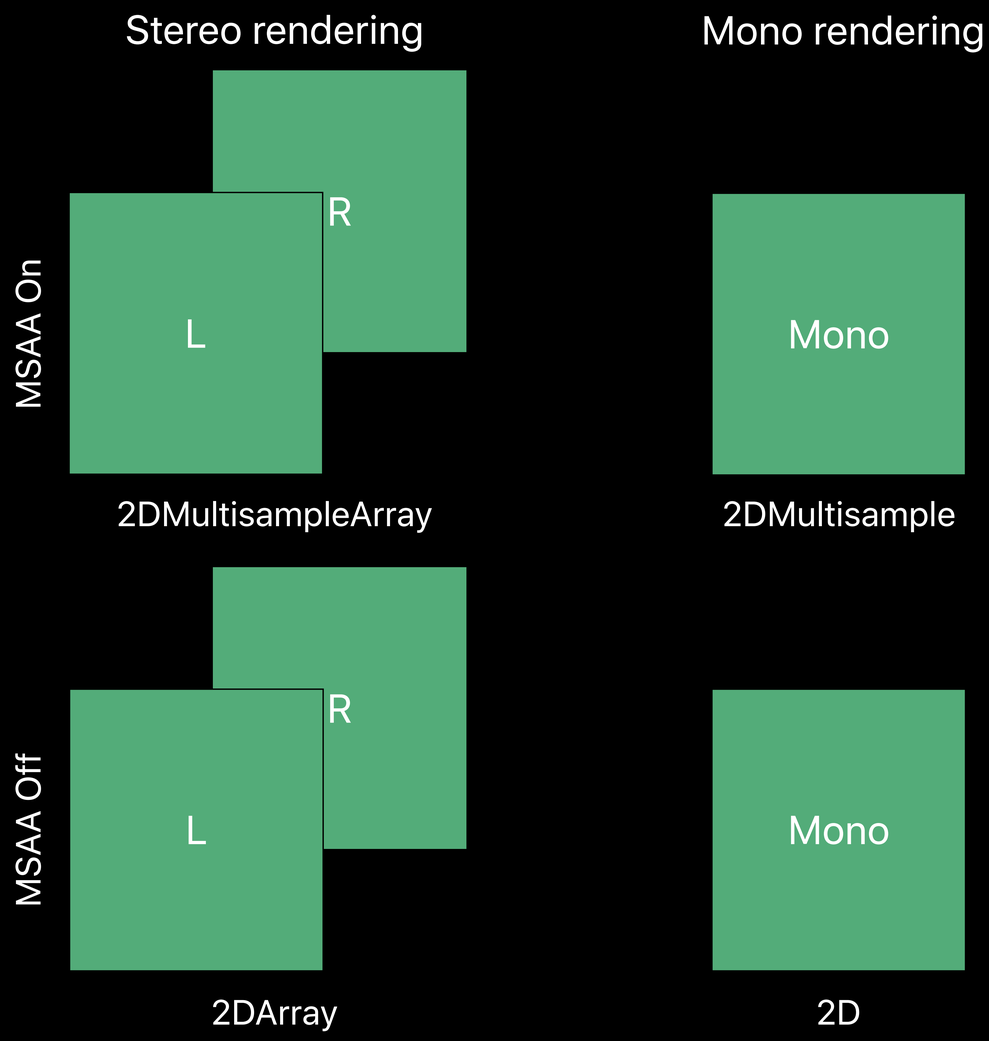
NEW

New Metal texture type

`MTLTextureType2DMultisampleArray`

Separate control of

- Rendering space
- Views count
- Anti-aliasing mode



Rendering Layouts

Different rendering patterns unified

NEW

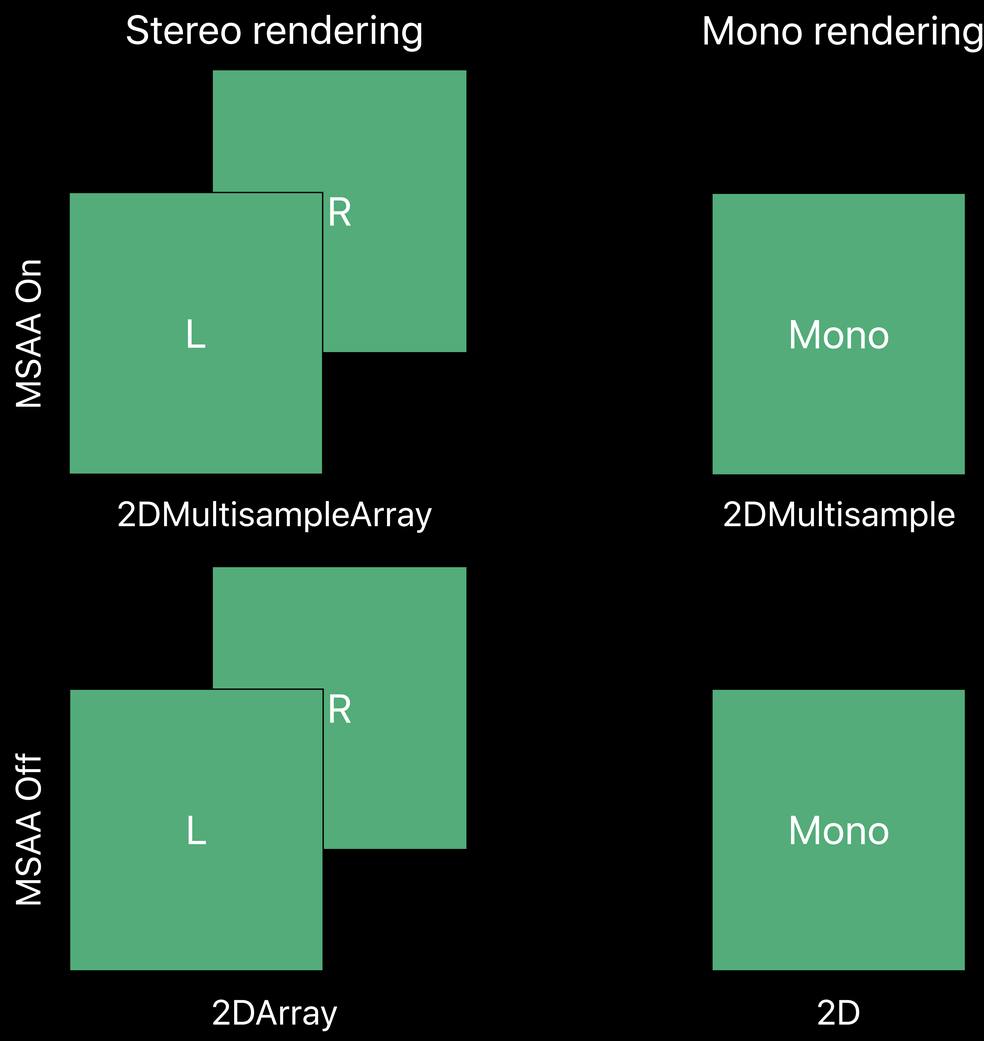
New Metal texture type

`MTLTextureType2DMultisampleArray`

Separate control of

- Rendering space
- Views count
- Anti-aliasing mode

Single draw, render, and resolve pass



Rendering Layouts

2D multisample array textures

NEW

Application creates 2D MS Array texture for rendering VR content

```
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType   = MTLTextureType2DMultisampleArray;
descriptor.pixelFormat   = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width         = width;
descriptor.height        = height;
descriptor.sampleCount   = 4u;
descriptor.arrayLength   = 2u;
descriptor.storageMode   = MTLStorageModePrivate;
descriptor.usage         = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

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

Rendering Layouts

2D multisample array textures

NEW

Application creates 2D MS Array texture for rendering VR content

```
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType   = MTLTextureType2DMultisampleArray;
descriptor.pixelFormat   = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width         = width;
descriptor.height        = height;
descriptor.sampleCount   = 4u;
descriptor.arrayLength   = 2u;
descriptor.storageMode   = MTLStorageModePrivate;
descriptor.usage         = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

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

Rendering Layouts

2D multisample array textures

NEW

Application creates 2D MS Array texture for rendering VR content

```
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType   = MTLTextureType2DMultisampleArray;
descriptor.pixelFormat   = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width         = width;
descriptor.height        = height;
descriptor.sampleCount   = 4u;
descriptor.arrayLength   = 2u;
descriptor.storageMode   = MTLStorageModePrivate;
descriptor.usage         = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

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

Rendering Layouts

2D multisample array textures

NEW

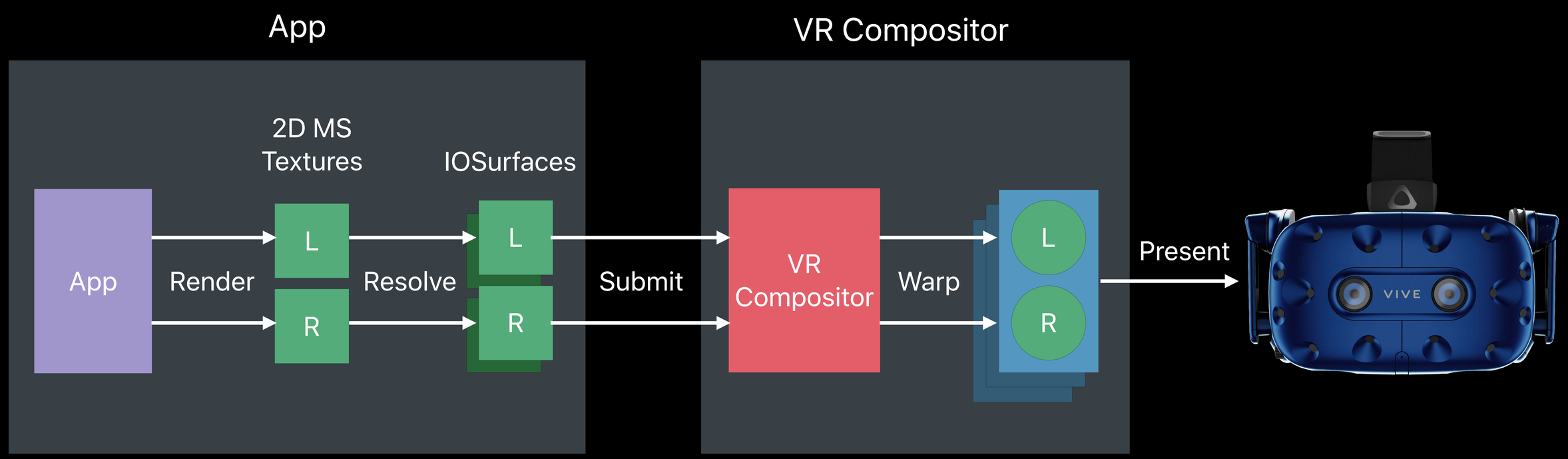
Application creates 2D MS Array texture for rendering VR content

```
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType   = MTLTextureType2DMultisampleArray;
descriptor.pixelFormat   = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width         = width;
descriptor.height        = height;
descriptor.sampleCount   = 4u;
descriptor.arrayLength   = 2u;
descriptor.storageMode   = MTLStorageModePrivate;
descriptor.usage         = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

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

VR Application

Rendering to 2D multisample array textures

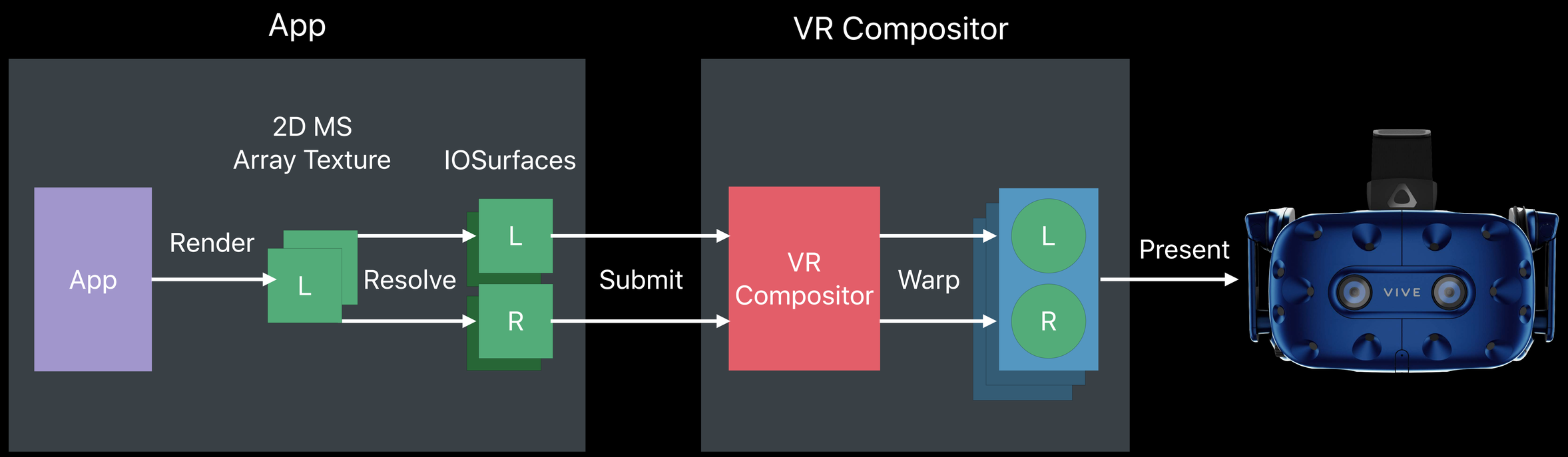
NEW



VR Application

Rendering to 2D multisample array textures

NEW



Cross-Process Texture Sharing

IOSurfaces versus shareable Metal textures

NEW

Cross-Process Texture Sharing

IOSurfaces versus shareable Metal textures

NEW

Ability to share Metal textures between processes

Cross-Process Texture Sharing

IOSurfaces versus shareable Metal textures

NEW

Ability to share Metal textures between processes

Shareable Metal textures can have complex structure

Cross-Process Texture Sharing

IOSurfaces versus shareable Metal textures

NEW

Ability to share Metal textures between processes

Shareable Metal textures can have complex structure

Shareable texture can be used only in scope of single GPU

Cross-Process Texture Sharing

IOSurfaces versus shareable Metal textures

NEW

Ability to share Metal textures between processes

Shareable Metal textures can have complex structure

Shareable texture can be used only in scope of single GPU

Enable advanced VR use cases, but are not limited to them

Cross-Process Texture Sharing

Standard versus shareable texture creation

NEW

Application creates 2D Array texture that can be shared with Compositor

```
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType   = MTLTextureType2DArray;
descriptor.pixelFormat   = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width         = width;
descriptor.height        = height;
descriptor.sampleCount   = 1u;
descriptor.arrayLength   = 2u;
descriptor.storageMode   = MTLStorageModePrivate;
descriptor.usage         = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
```

Cross-Process Texture Sharing

Standard versus shareable texture creation

NEW

Application creates 2D Array texture that can be shared with Compositor

```
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType   = MTLTextureType2DArray;
descriptor.pixelFormat   = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width         = width;
descriptor.height        = height;
descriptor.sampleCount   = 1u;
descriptor.arrayLength   = 2u;
descriptor.storageMode   = MTLStorageModePrivate;
descriptor.usage         = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
```

Cross-Process Texture Sharing

Standard versus shareable texture creation

NEW

Application creates 2D Array texture that can be shared with Compositor

```
MTLTextureDescriptor* descriptor = [[MTLTextureDescriptor alloc] init];
descriptor.textureType   = MTLTextureType2DArray;
descriptor.pixelFormat   = MTLPixelFormatBGRA8Unorm_sRGB;
descriptor.width         = width;
descriptor.height        = height;
descriptor.sampleCount   = 1u;
descriptor.arrayLength   = 2u;
descriptor.storageMode   = MTLStorageModePrivate;
descriptor.usage         = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;

id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
```

Cross-Process Texture Sharing

Passing texture to compositor

NEW

Passing frame for presentation, using shareable Metal texture

```
IOSurfaceRef   backingIOSurface[2];
id<MTLTexture> resolvedTexture[2]; // Two 2D textures (backed by IOSurfaces)
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle      = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
textureDesc[i].eType       = vr::ETextureType::TextureType_IOSurface;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i]           = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```

Cross-Process Texture Sharing

Passing texture to compositor

NEW

Passing frame for presentation, using shareable Metal texture

```
IOSurfaceRef   backingIOSurface[2];
id<MTLTexture> resolvedTexture[2]; // Two 2D textures (backed by IOSurfaces)
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle      = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
textureDesc[i].eType       = vr::ETextureType::TextureType_IOSurface;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i]           = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```

Cross-Process Texture Sharing

Passing texture to compositor

NEW

Passing frame for presentation, using shareable Metal texture

```
id<MTLTexture> resolvedTexture[2]; // Two 2D textures (backed by IOSurfaces)
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle       = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
textureDesc[i].eType        = vr::ETextureType::TextureType_IOSurface;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i]            = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```

Cross-Process Texture Sharing

Passing texture to compositor

NEW

Passing frame for presentation, using shareable Metal texture

```
id<MTLTexture> resolvedTexture[2]; // Two 2D textures (backed by IOSurfaces)
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle       = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
textureDesc[i].eType        = vr::ETextureType::TextureType_IOSurface;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i]            = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```

Cross-Process Texture Sharing

Passing texture to compositor

NEW

Passing frame for presentation, using shareable Metal texture

```
id<MTLTexture> resolvedTexture;    // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle       = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
textureDesc[i].eType        = vr::ETextureType::TextureType_IOSurface;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i]            = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```

Cross-Process Texture Sharing

Passing texture to compositor

NEW

Passing frame for presentation, using shareable Metal texture

```
id<MTLTexture> resolvedTexture;    // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle      = reinterpret_cast<void*>(intptr_t(backingIOSurface[i]));
textureDesc[i].eType       = vr::ETextureType::TextureType_IOSurface;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i]           = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```

Cross-Process Texture Sharing

Passing texture to compositor

NEW

Passing frame for presentation, using shareable Metal texture

```
id<MTLTexture> resolvedTexture;    // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle       = reinterpret_cast<void*>(intptr_t(resolvedTexture));
textureDesc[i].eType        = vr::ETextureType::TextureType_IOSurface;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i]            = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```

Cross-Process Texture Sharing

Passing texture to compositor

NEW

Passing frame for presentation, using shareable Metal texture

```
id<MTLTexture> resolvedTexture;    // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle      = reinterpret_cast<void*>(intptr_t(resolvedTexture));
textureDesc[i].eType       = vr::ETextureType::TextureType_Metal;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i]           = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```

Cross-Process Texture Sharing

Passing texture to compositor

NEW

Passing frame for presentation, using shareable Metal texture

```
id<MTLTexture> resolvedTexture;    // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle      = reinterpret_cast<void*>(intptr_t(resolvedTexture));
textureDesc[i].eType       = vr::ETextureType::TextureType_Metal;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i]           = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```

Cross-Process Texture Sharing

Passing texture to compositor

NEW

Passing frame for presentation, using shareable Metal texture

```
id<MTLTexture> resolvedTexture;    // Shareable 2D Array texture
vr::VRTexture_t textureDesc[2];
vr::VRTextureBounds_t textureBounds[2];
for(uint32 i=0; i<2; ++i) {
textureDesc[i].handle      = reinterpret_cast<void*>(intptr_t(resolvedTexture));
textureDesc[i].eType       = vr::ETextureType::TextureType_Metal;
textureDesc[i].eColorSpace = vr::EColorSpace::ColorSpace_Linear;
textureBounds[i]           = { .uMin = 0.0f, .uMax = 1.0f, .vMin = 0.0f, .vMax = 1.0f };
}

vr.VRCompositor()->Submit(vr::Eye_Left, &textureDesc[0], &textureBounds[0]);
vr.VRCompositor()->Submit(vr::Eye_Right, &textureDesc[1], &textureBounds[1]);
```

Cross-Process Texture Sharing

Passing across process boundary

Process A

```
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```

Process B

```
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```

Cross-Process Texture Sharing

Passing across process boundary

Process A

```
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```

Process B

```
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```

Cross-Process Texture Sharing

Passing across process boundary

Process A

```
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```

Process B

```
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```

Cross-Process Texture Sharing

Passing across process boundary

Process A

```
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```


XPC Connection

Process B

```
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```

Cross-Process Texture Sharing

Passing across process boundary

Process A

```
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```


XPC Connection

Process B

```
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```

Cross-Process Texture Sharing

Passing across process boundary

Process A

```
id<MTLTexture> texture = [device newSharedTextureWithDescriptor:descriptor];
MTLSharedTextureHandle* sharedHandle = [texture newSharedTextureHandle];
```


XPC Connection

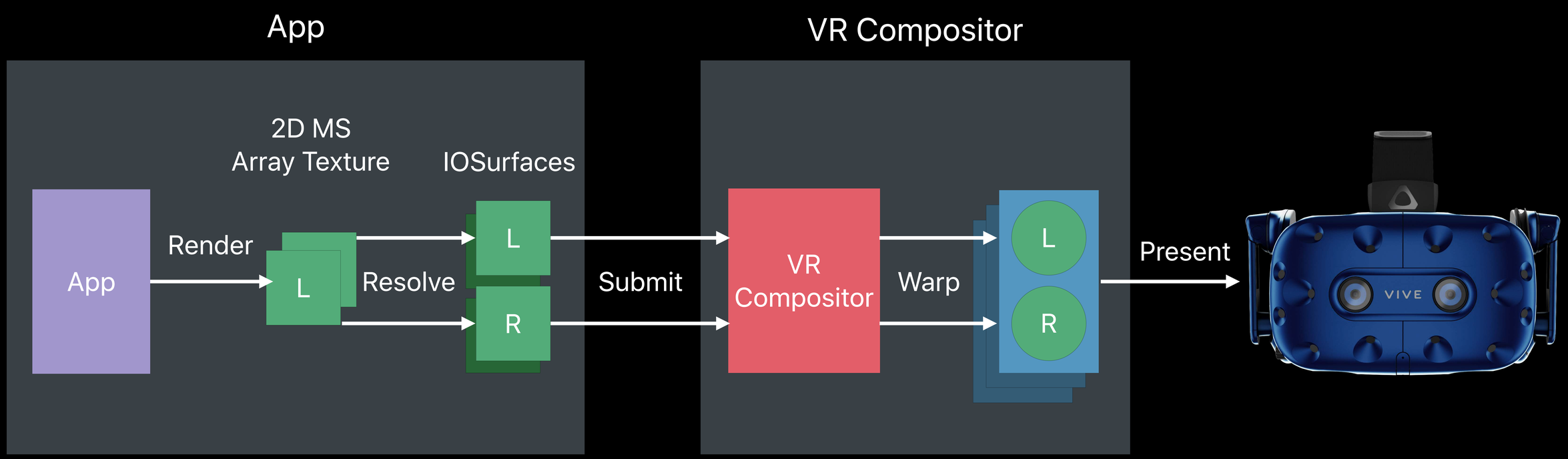
Process B

```
// Make sure you recreate your texture handle on the same GPU
id<MTLDevice> device = [sharedHandle device];
id<MTLTexture> texture = [device newSharedTextureWithHandle:sharedHandle];
```

VR Application

Using shareable Metal textures

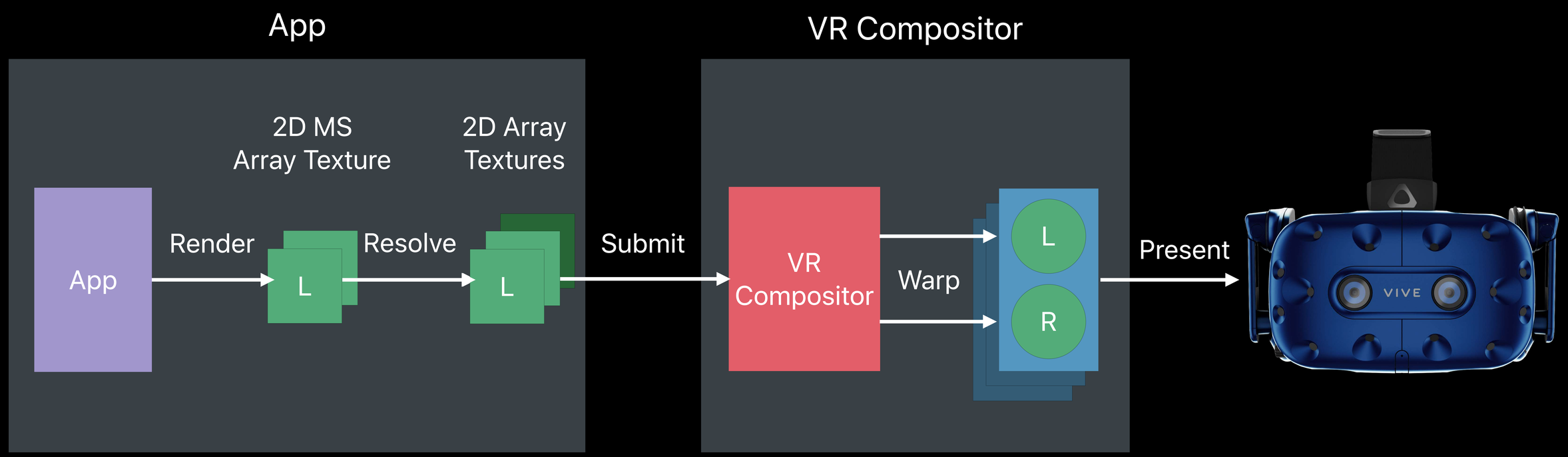
NEW



VR Application

Using shareable Metal textures

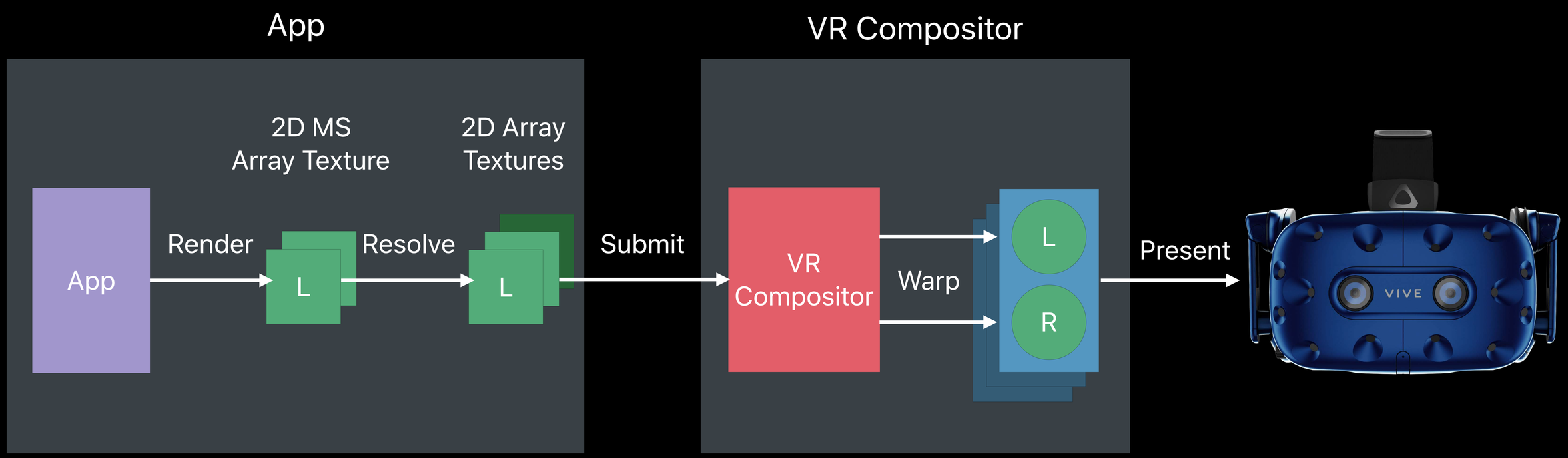
NEW



VR Application

Compositor optimizations

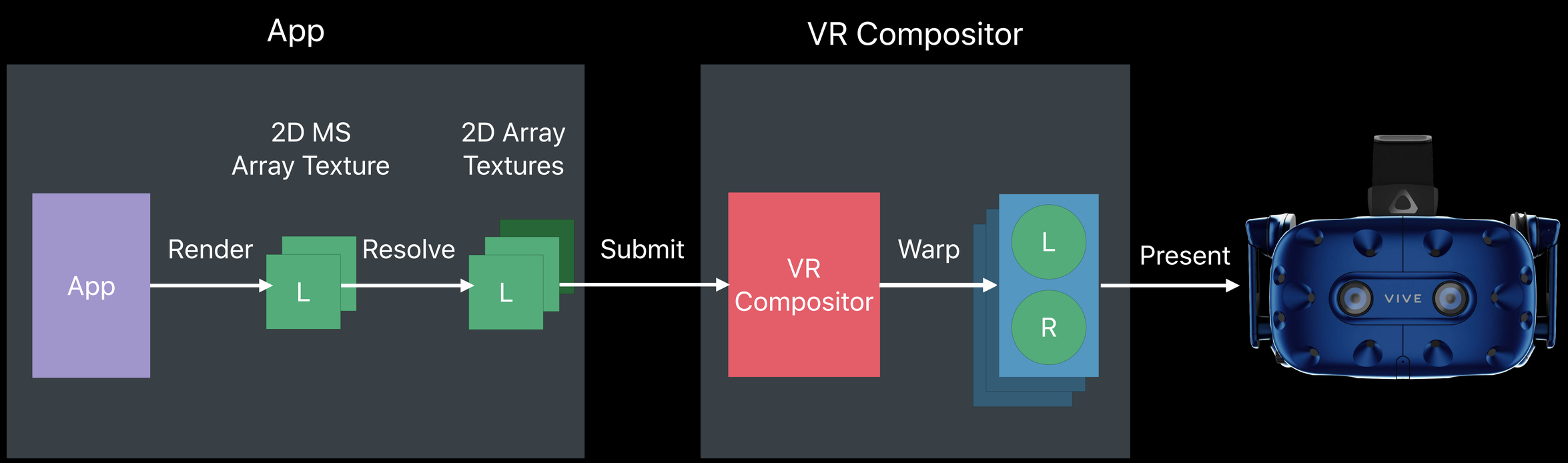
NEW



VR Application

Compositor optimizations

NEW



Metal 2 Features for VR

Recap

Shareable Metal textures

2D multisample array textures

OpenVR SDK support

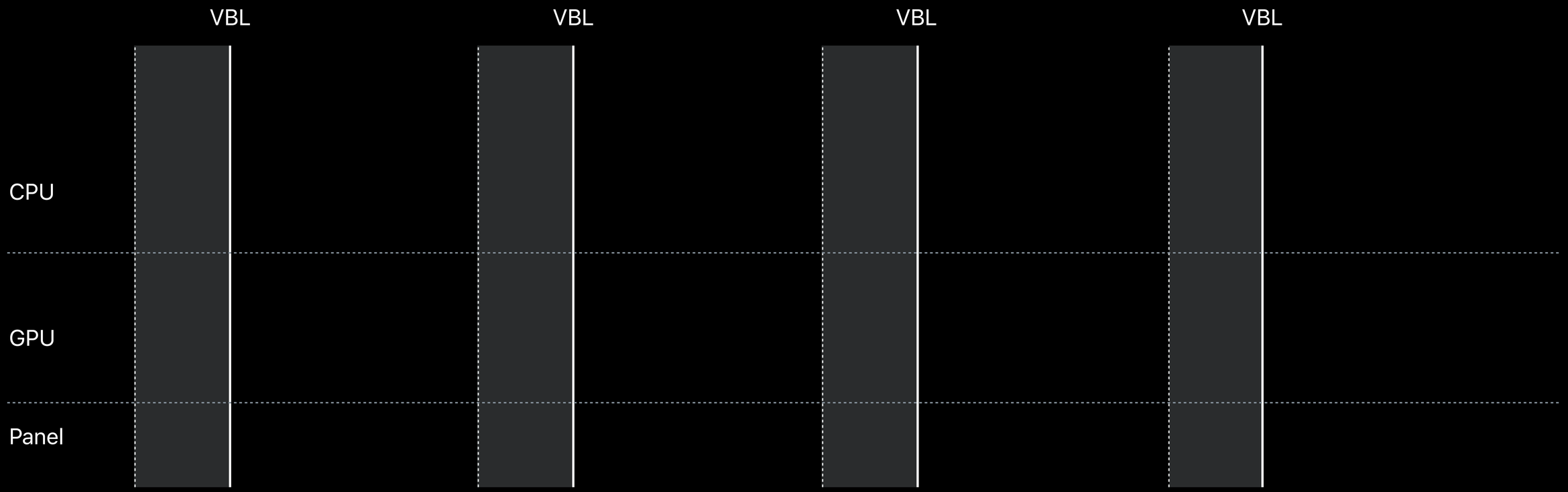
VR Advanced Techniques

Advanced Frame Pacing

Reducing Fill Rate

Frame Pacing

Single threaded application

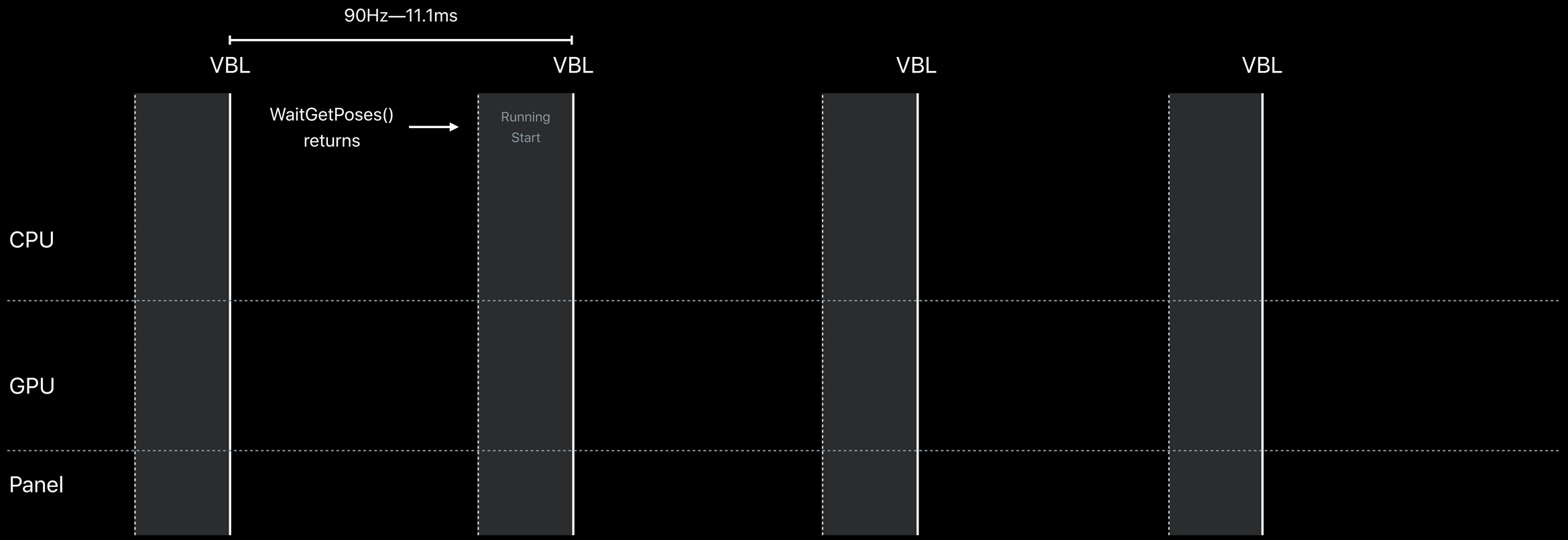


Frame Pacing

Single threaded application

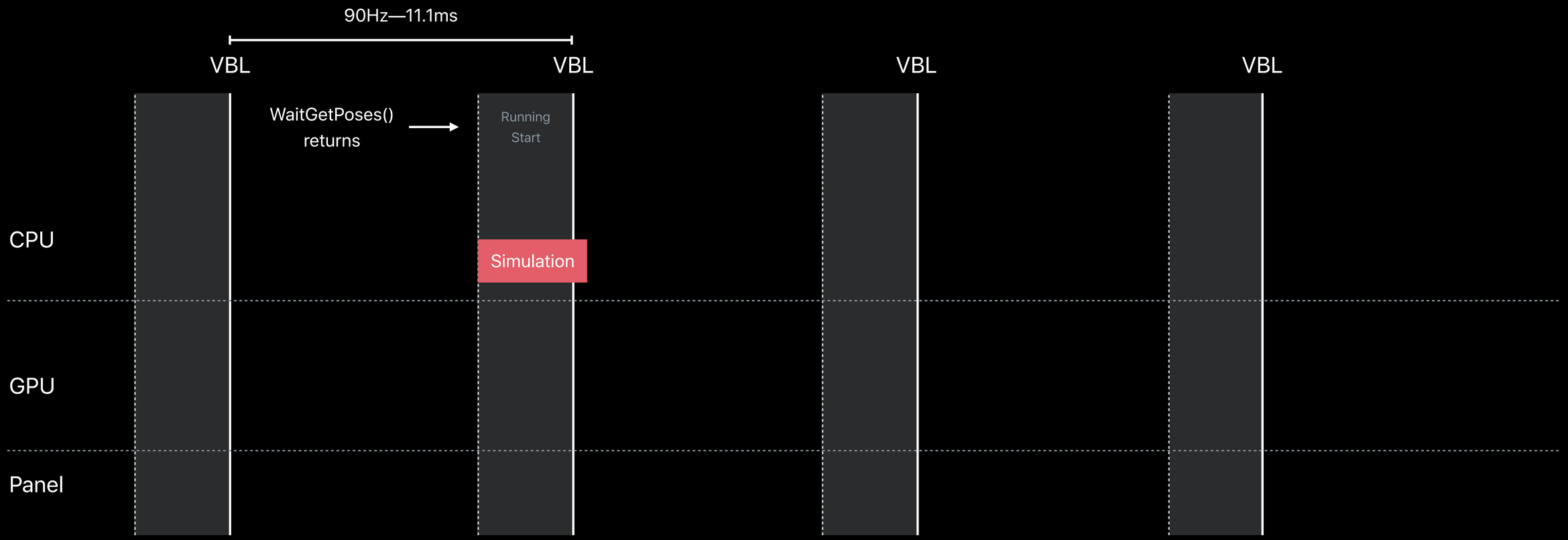
Frame Pacing

Single threaded application



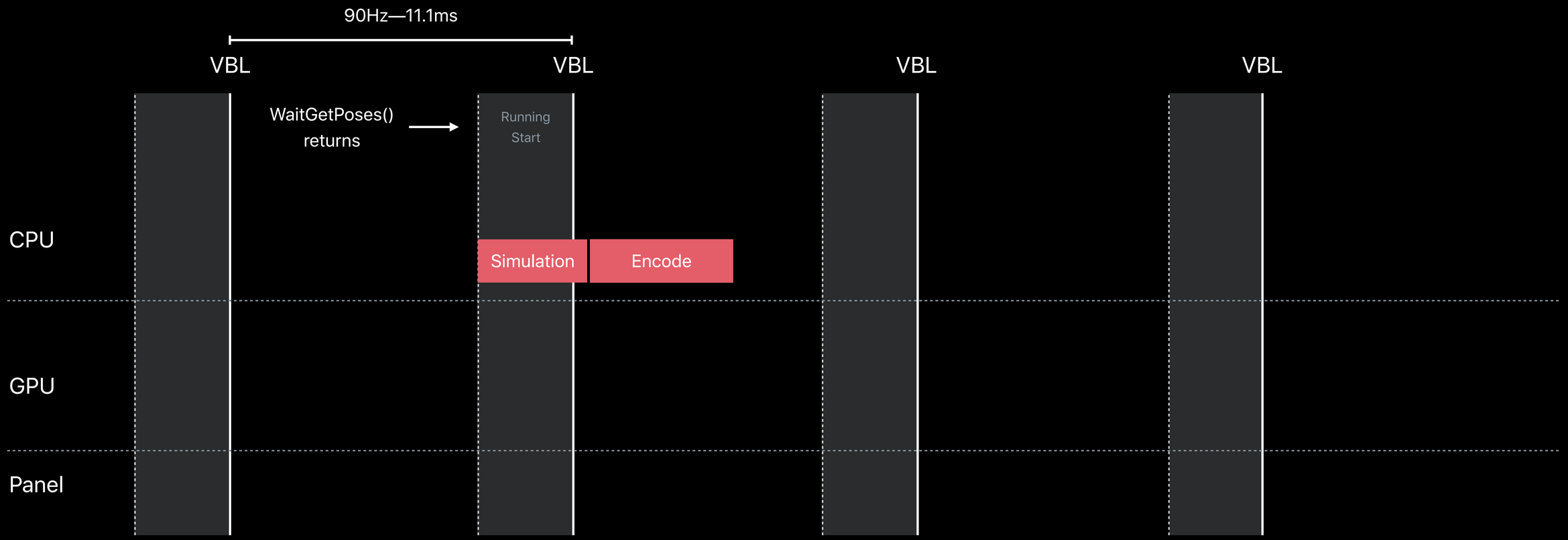
Frame Pacing

Single threaded application



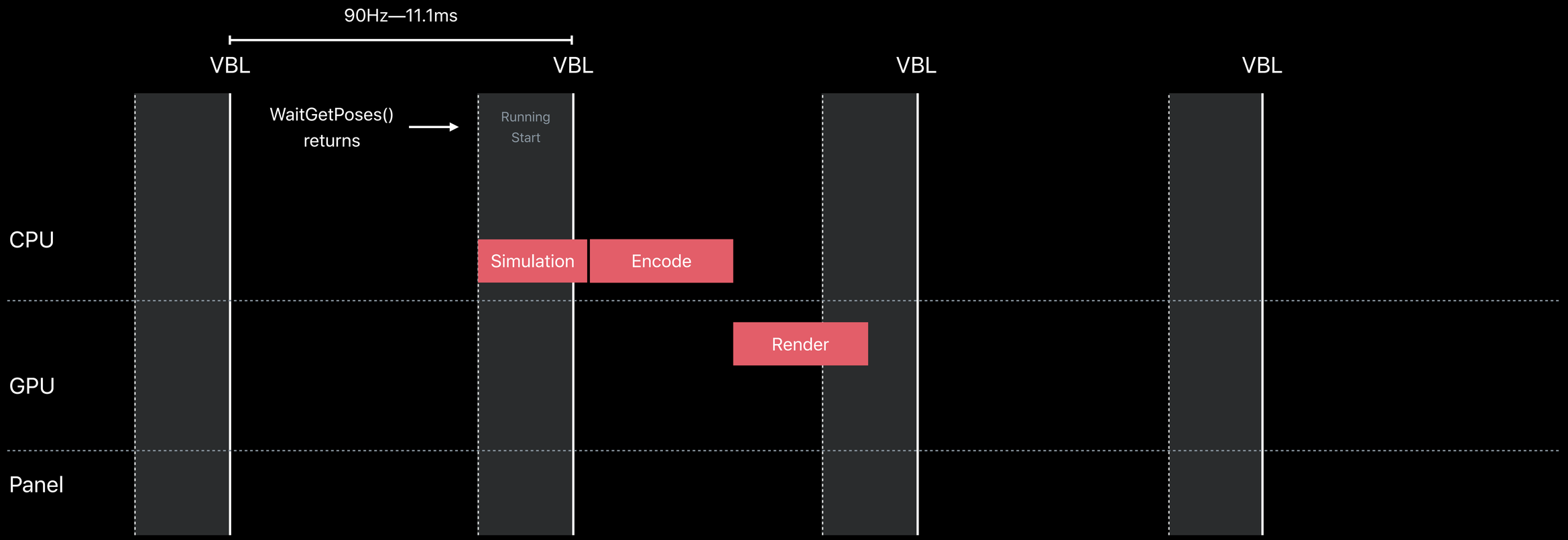
Frame Pacing

Single threaded application



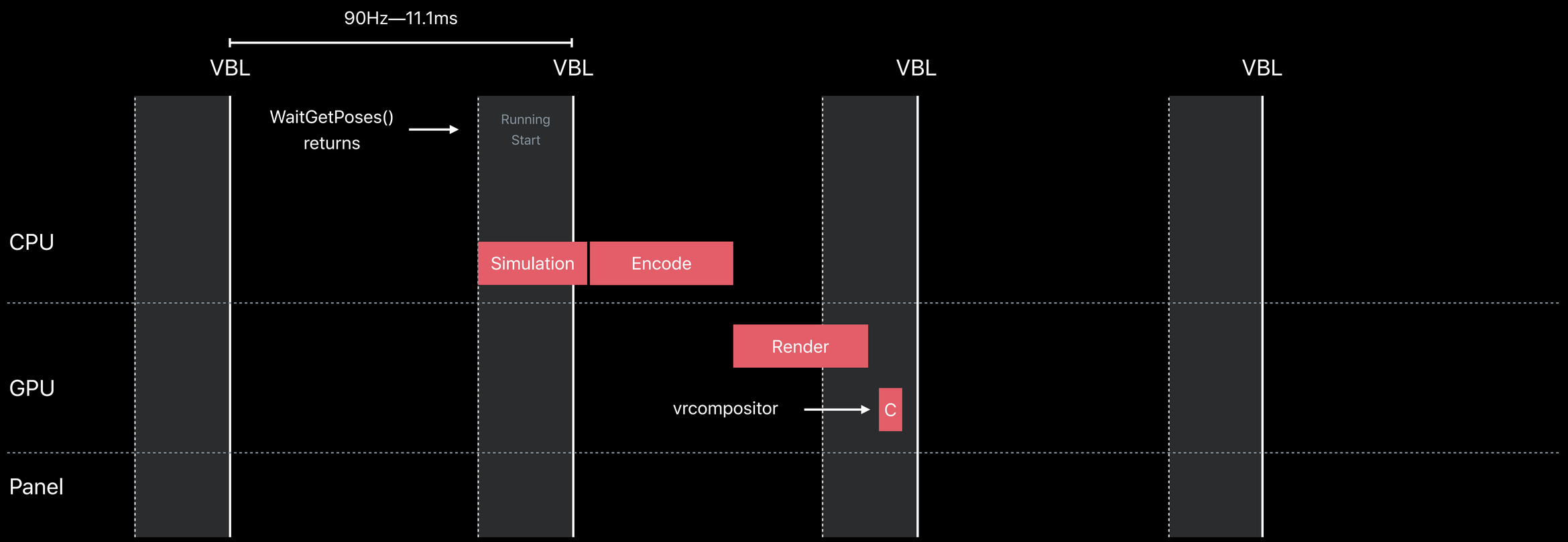
Frame Pacing

Single threaded application



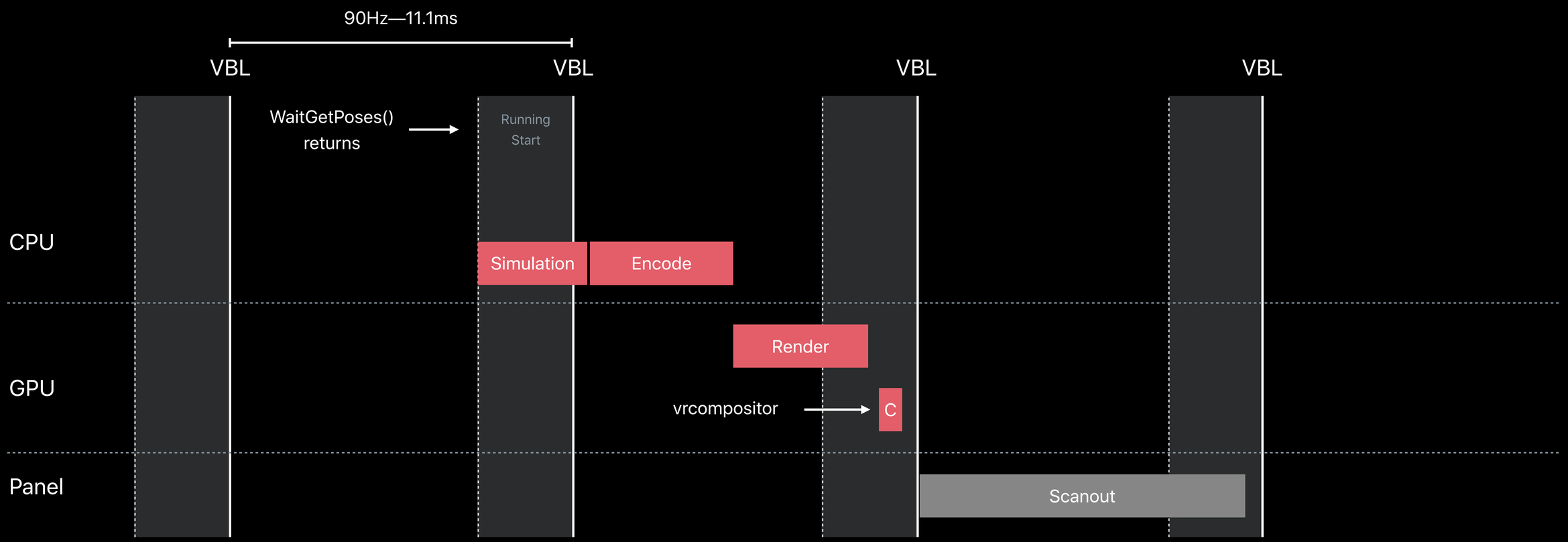
Frame Pacing

Single threaded application



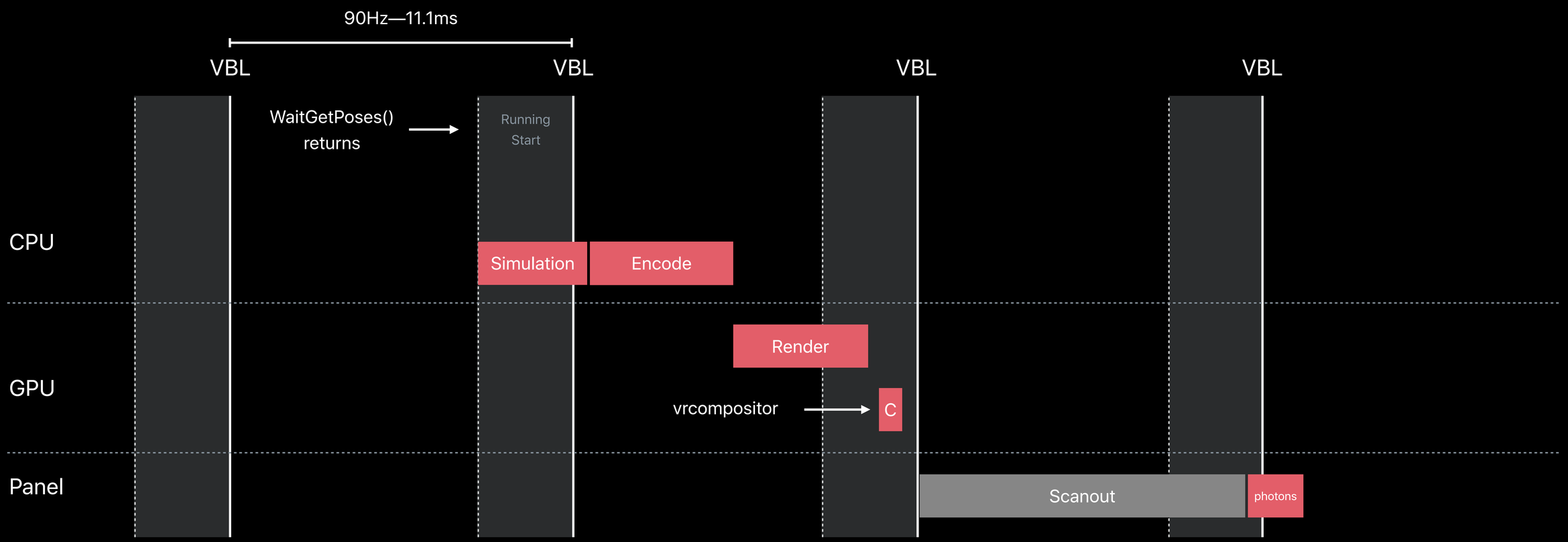
Frame Pacing

Single threaded application



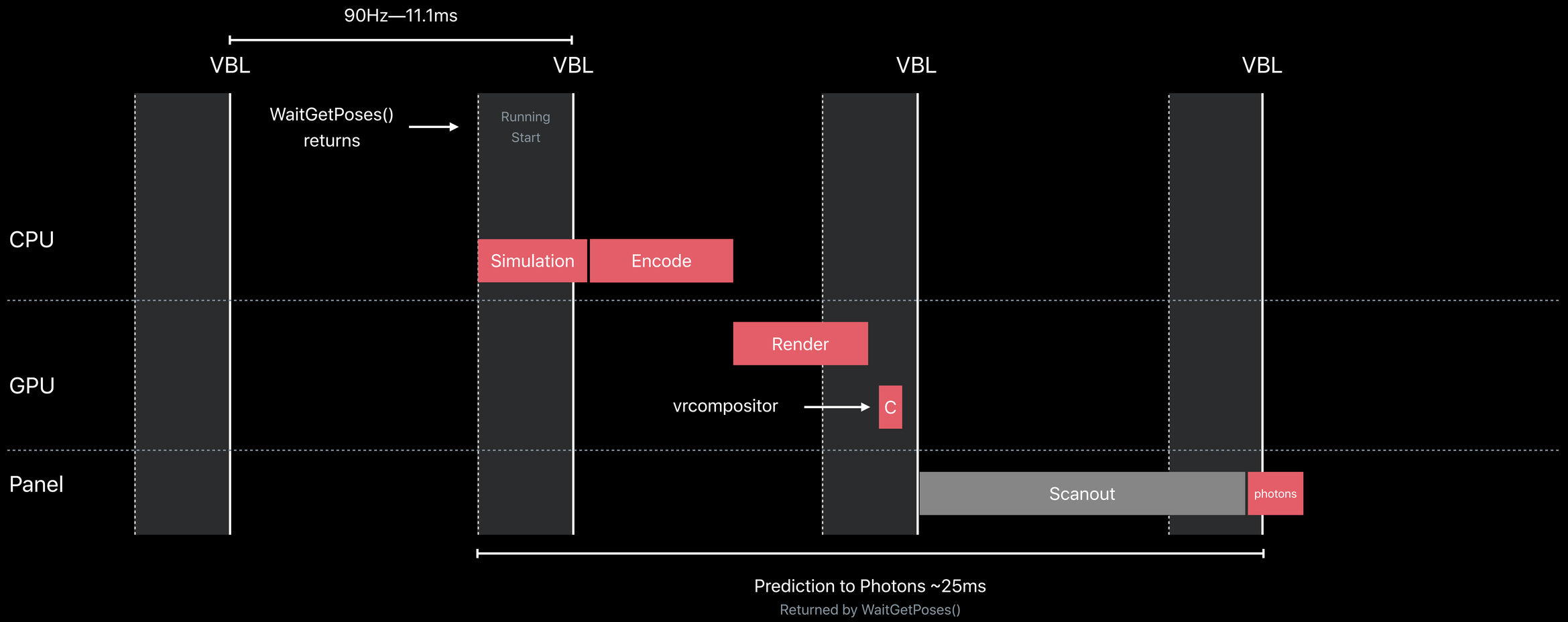
Frame Pacing

Single threaded application



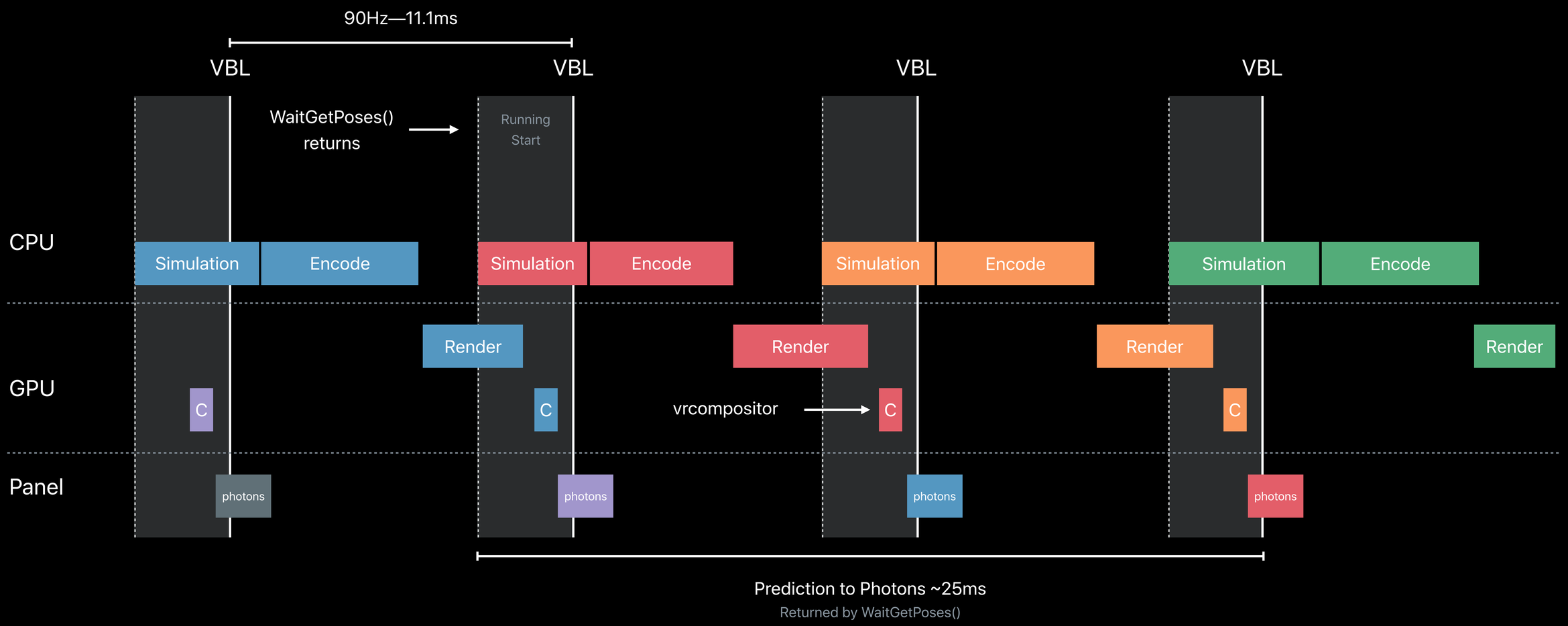
Frame Pacing

Single threaded application



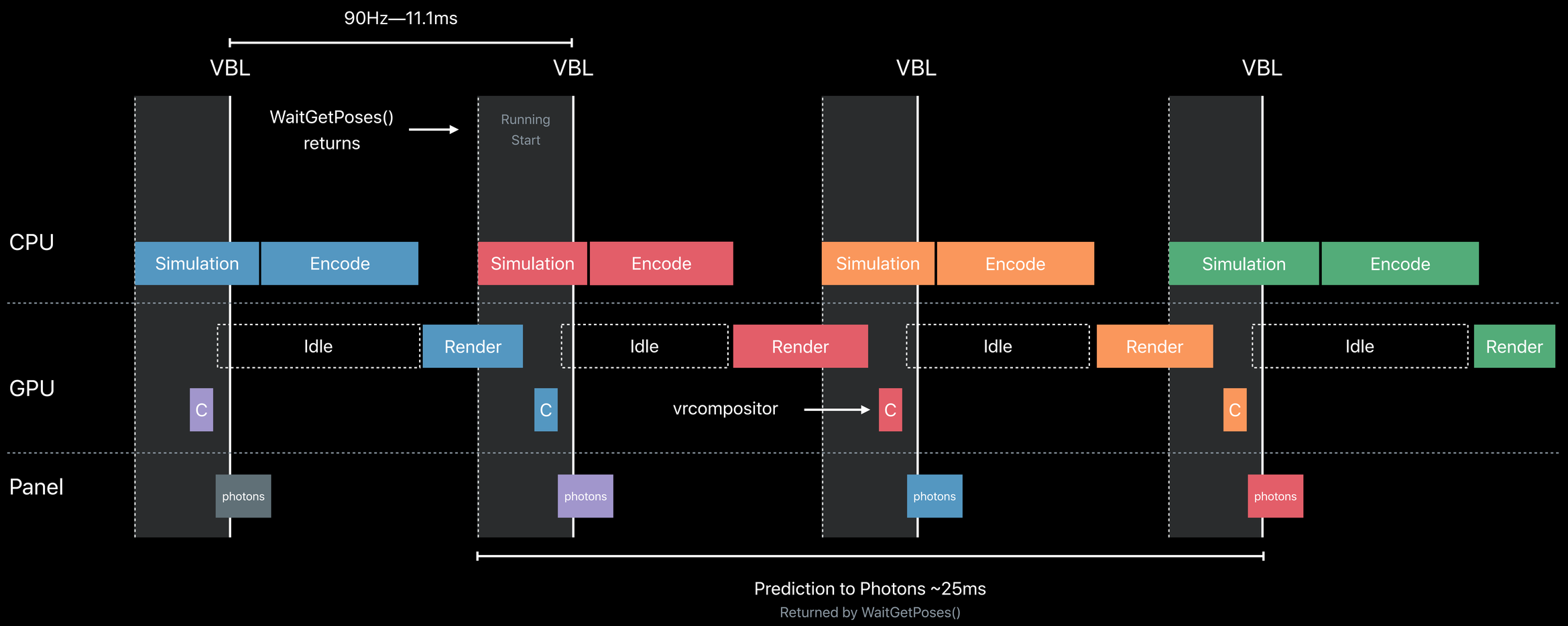
Frame Pacing

Single threaded application



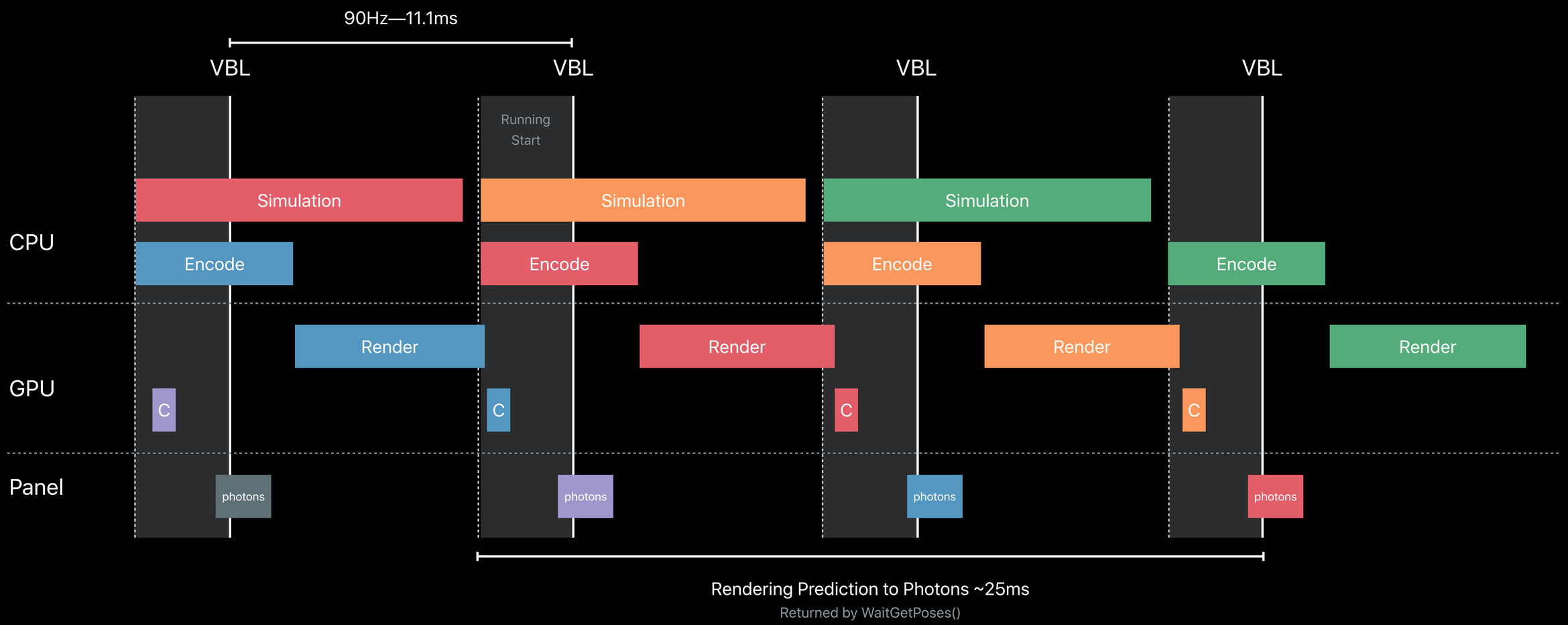
Frame Pacing

Single threaded application



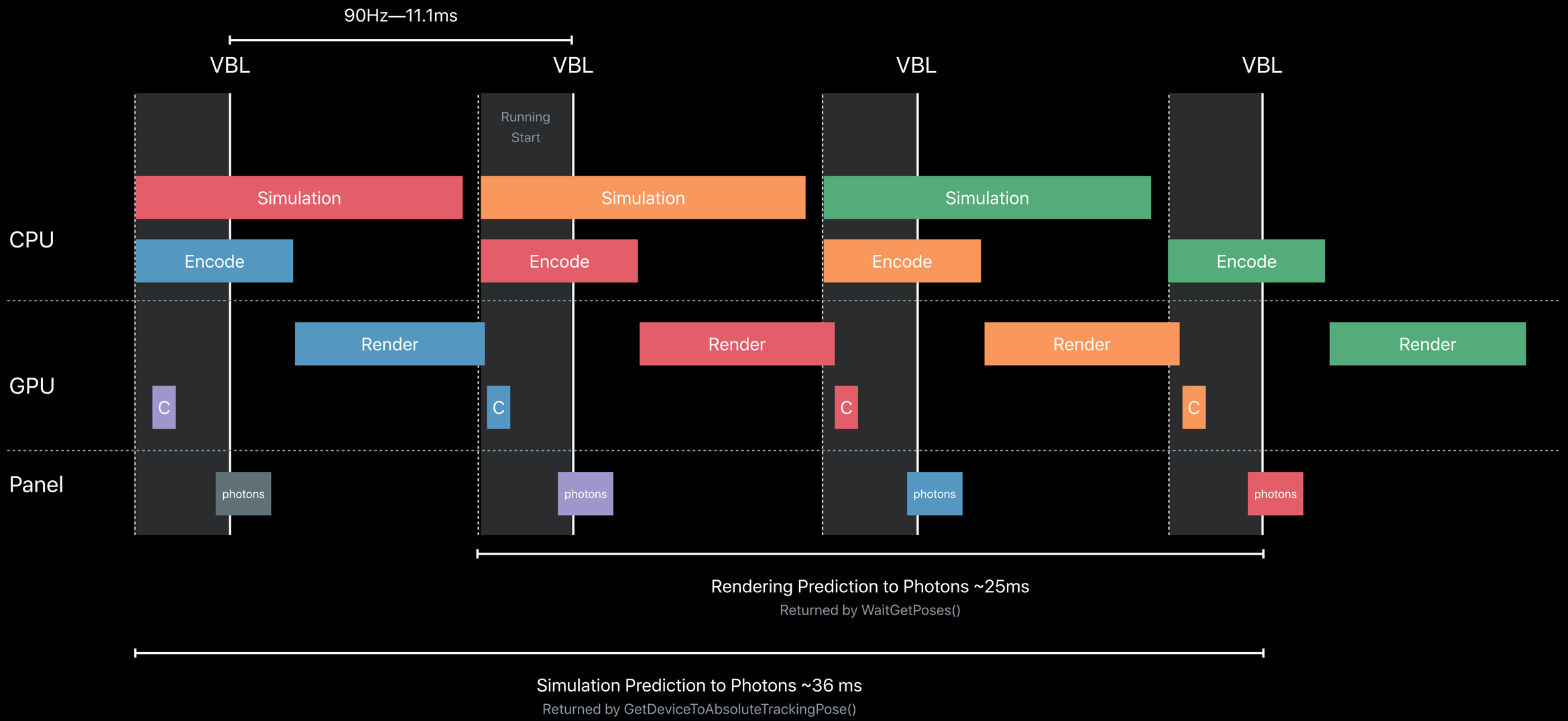
Frame Pacing

Multi-threaded application



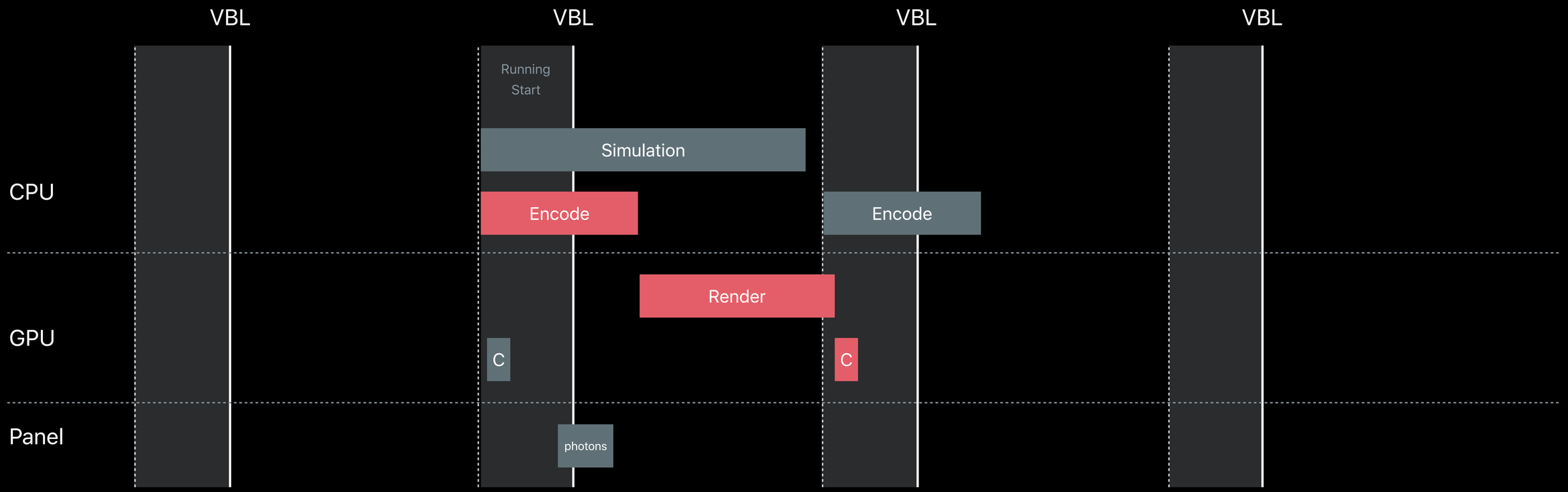
Frame Pacing

Multi-threaded application



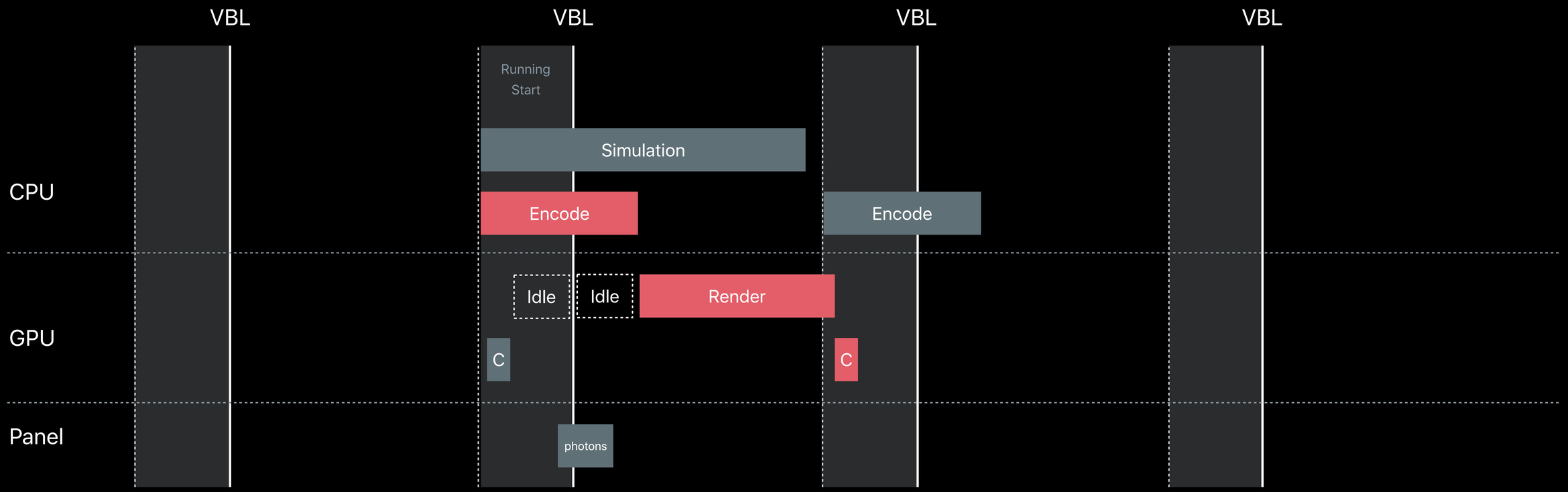
Frame Pacing

Splitting command buffer encoding



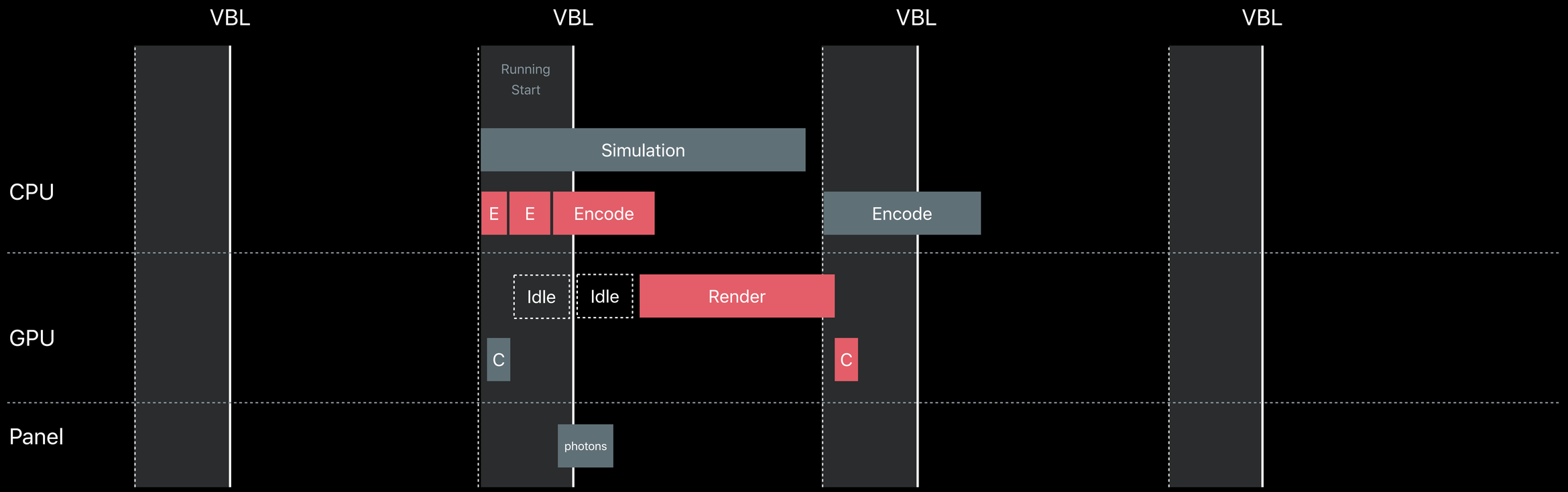
Frame Pacing

Splitting command buffer encoding



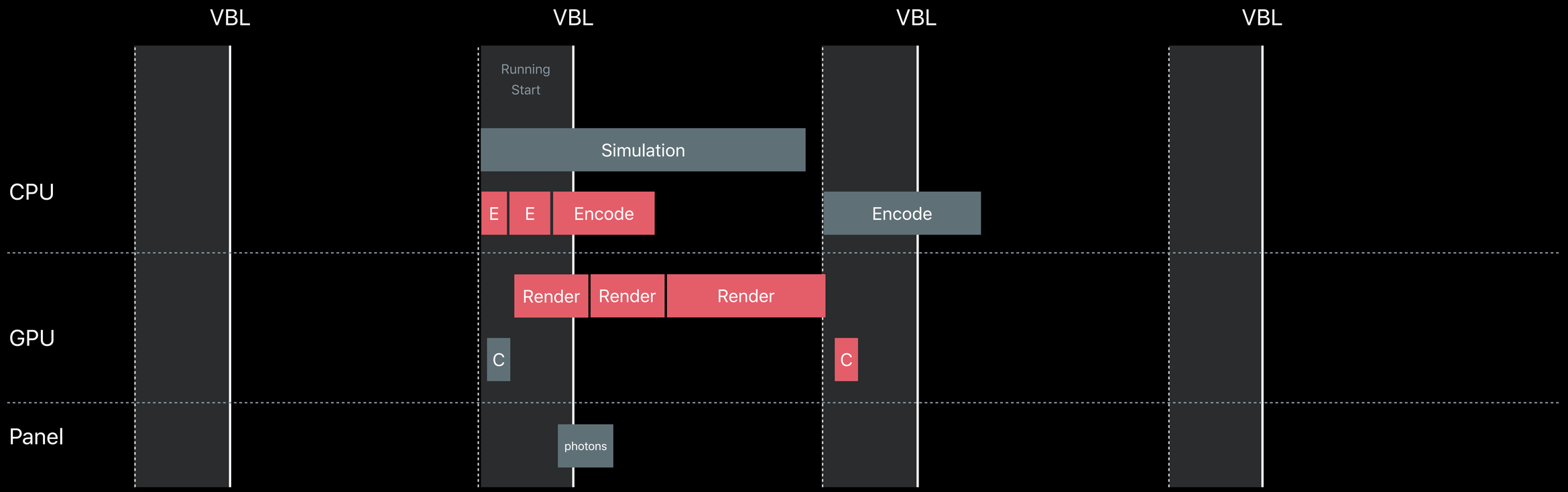
Frame Pacing

Splitting command buffer encoding



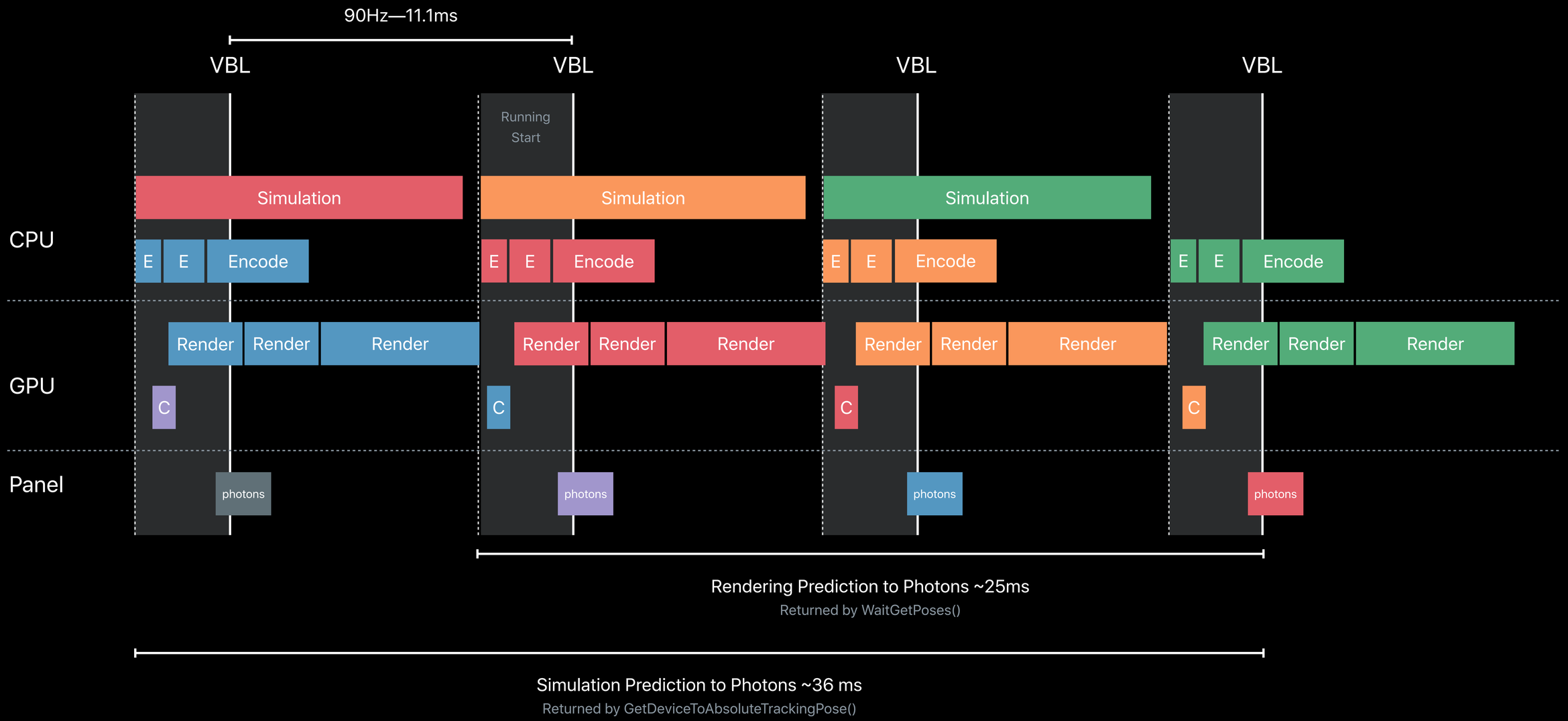
Frame Pacing

Splitting command buffer encoding



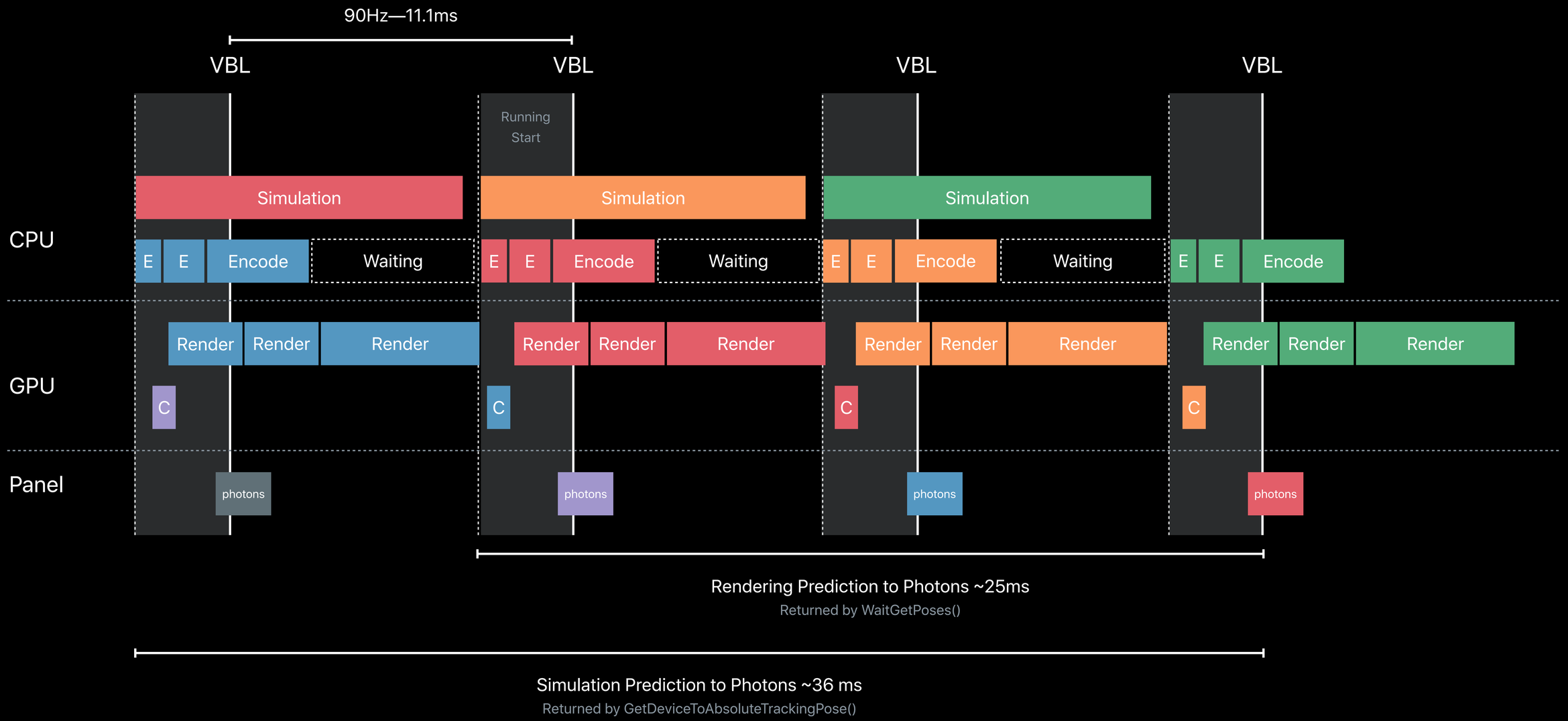
Advanced Frame Pacing

Multi-threaded application



Advanced Frame Pacing

Multi-threaded application



Advanced Frame Pacing

GPU workload examples

Companion / mirroring window

Physics, cloth, water simulation

Reflection maps

Frustum culling

Shadow maps

Occlusion queries

Rendering geometry

Final post-processing

Advanced Frame Pacing

GPU workload examples

Companion / mirroring window

Physics, cloth, water simulation

Reflection maps

Frustum culling

Shadow maps

Occlusion queries

Rendering geometry

Final post-processing

Advanced Frame Pacing

GPU workload examples

Companion / mirroring window

Physics, cloth, water simulation

Reflection maps

Pose-independent workloads

WaitGetPoses()

Frustum culling

Shadow maps

Occlusion queries

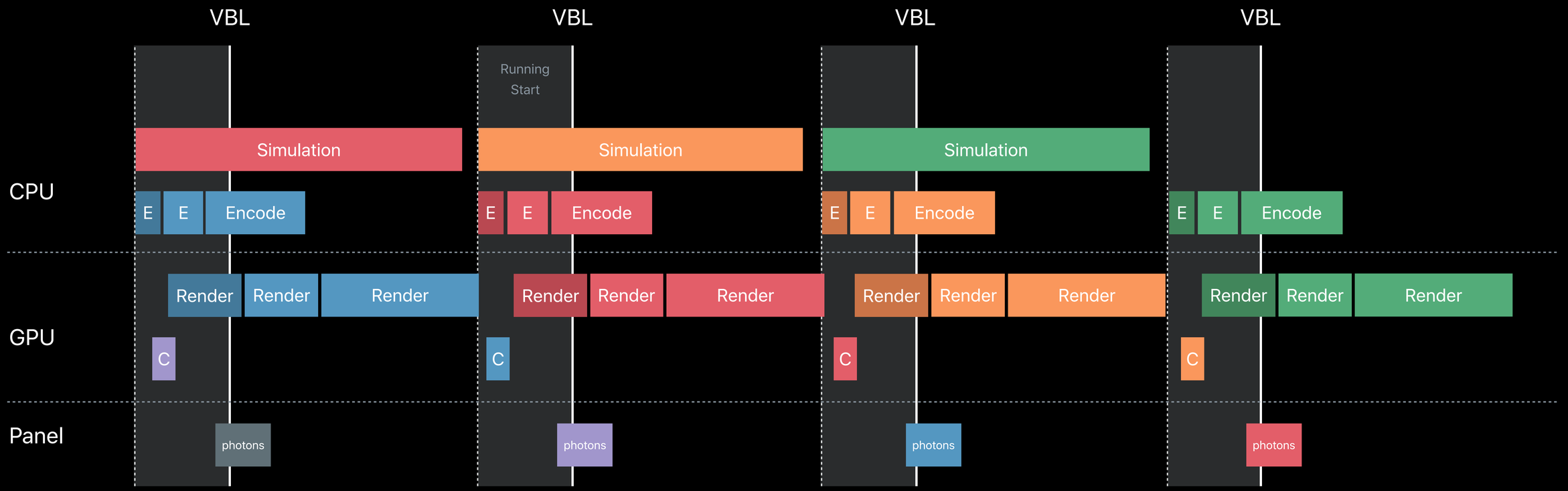
Pose-dependent workloads

Rendering geometry

Final post-processing

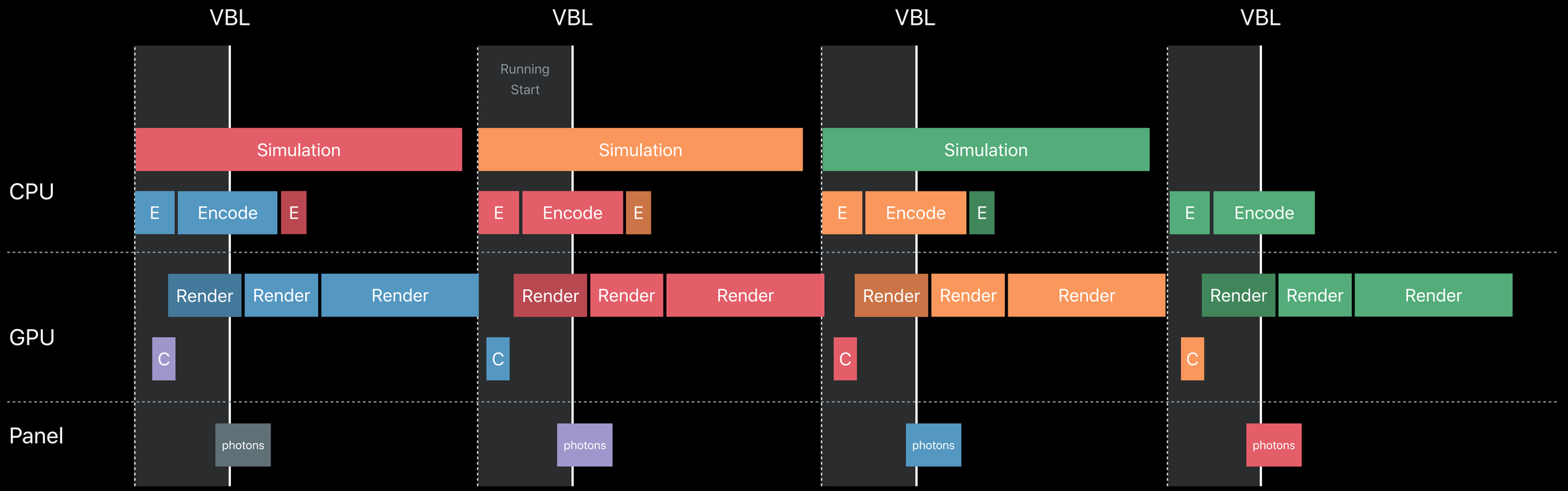
Advanced Frame Pacing

Pose independent work



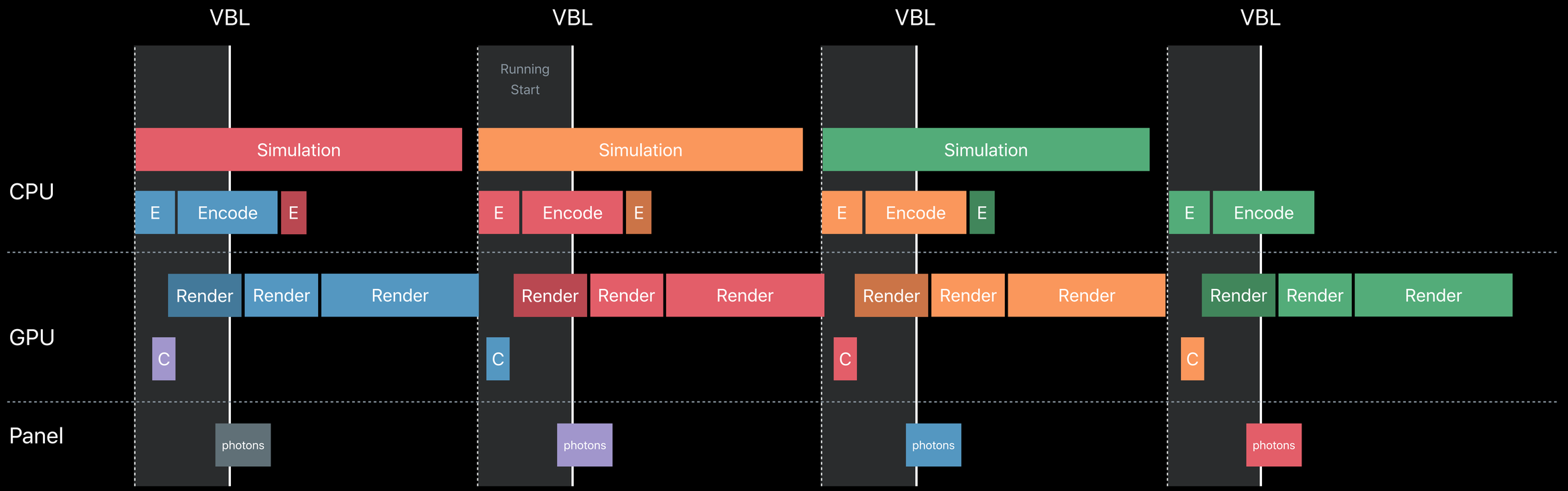
Advanced Frame Pacing

Pose independent work



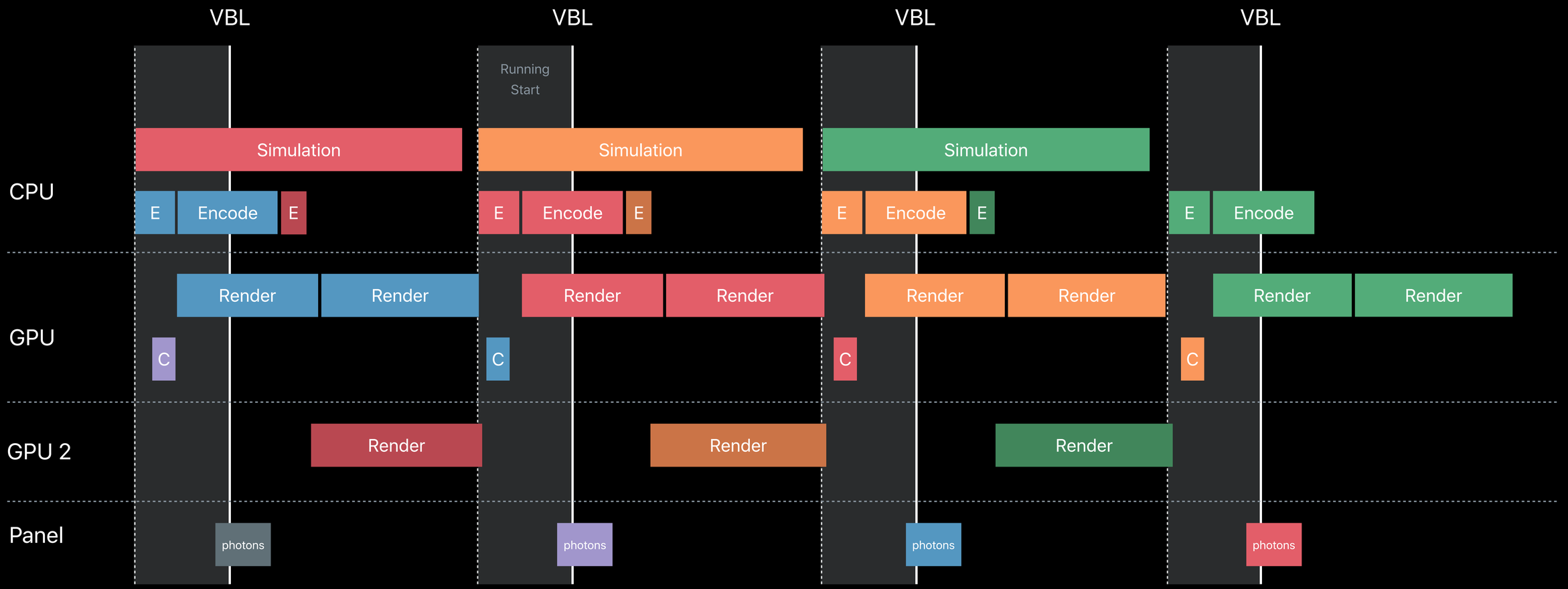
Advanced Frame Pacing

Multi-GPU workload distribution



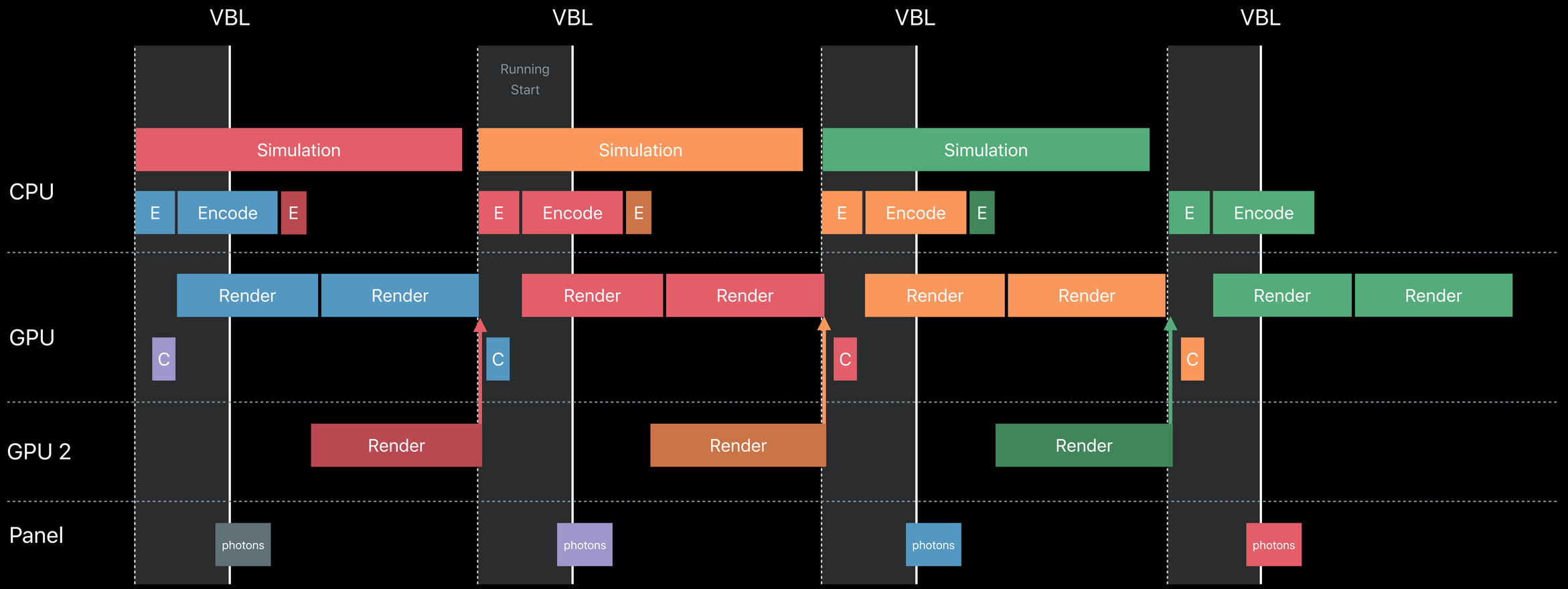
Advanced Frame Pacing

Multi-GPU workload distribution



Advanced Frame Pacing

Multi-GPU workload distribution



Multi-GPU Workload Distribution

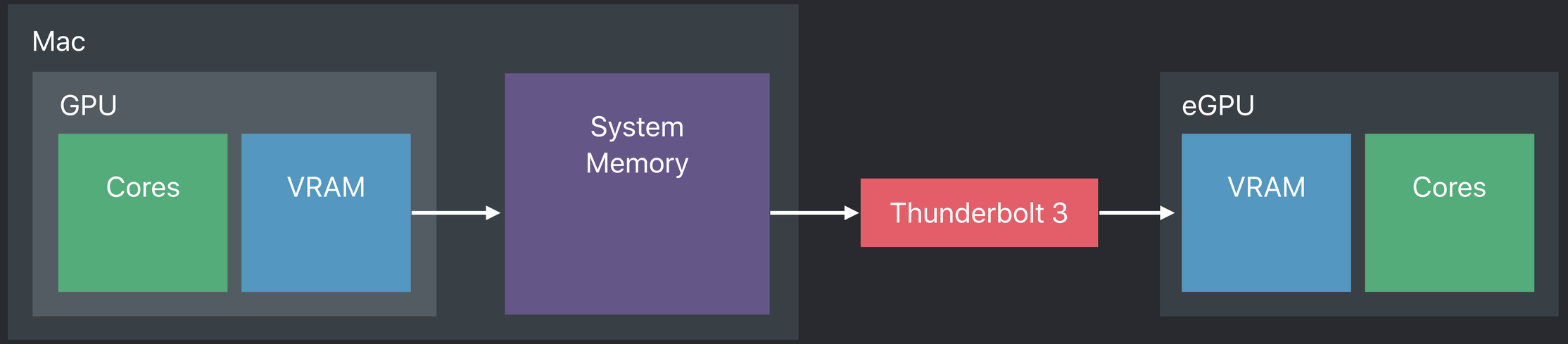
Synchronization primitives

MTLEvent

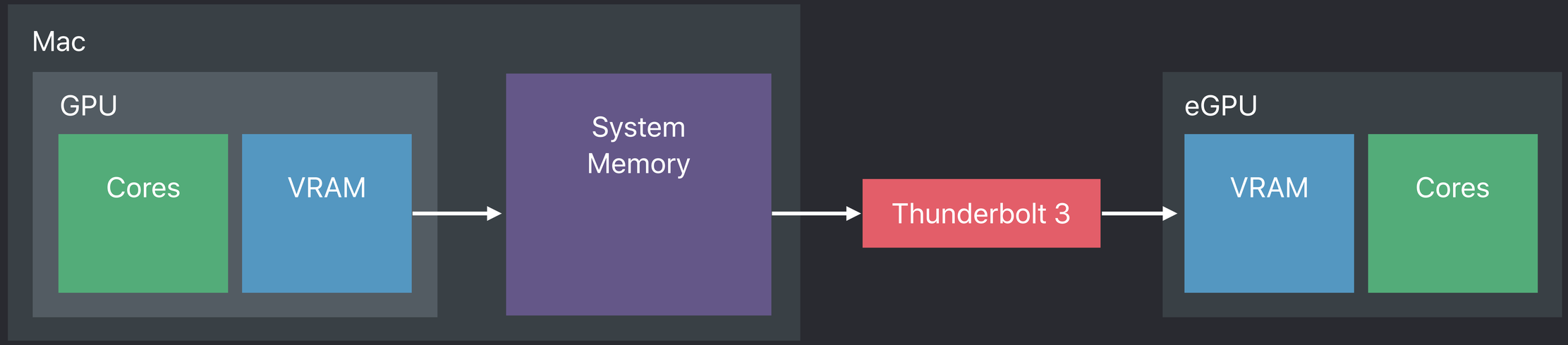
- Mechanism of synchronizing GPU workloads
- Synchronizes across Command Queues

MTLSharedEvent

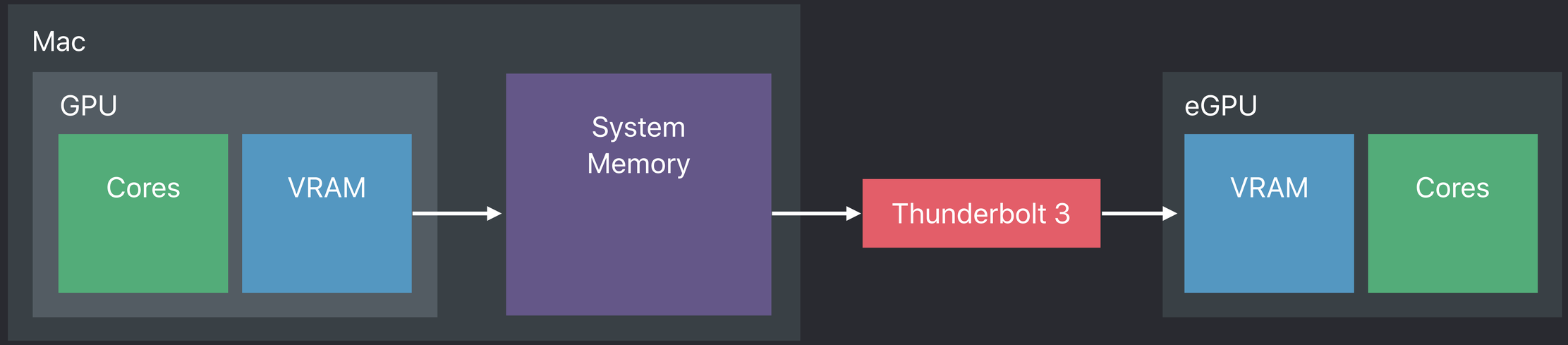
- Extends Event
- Synchronizes across GPU’s
- Can be shared between processes



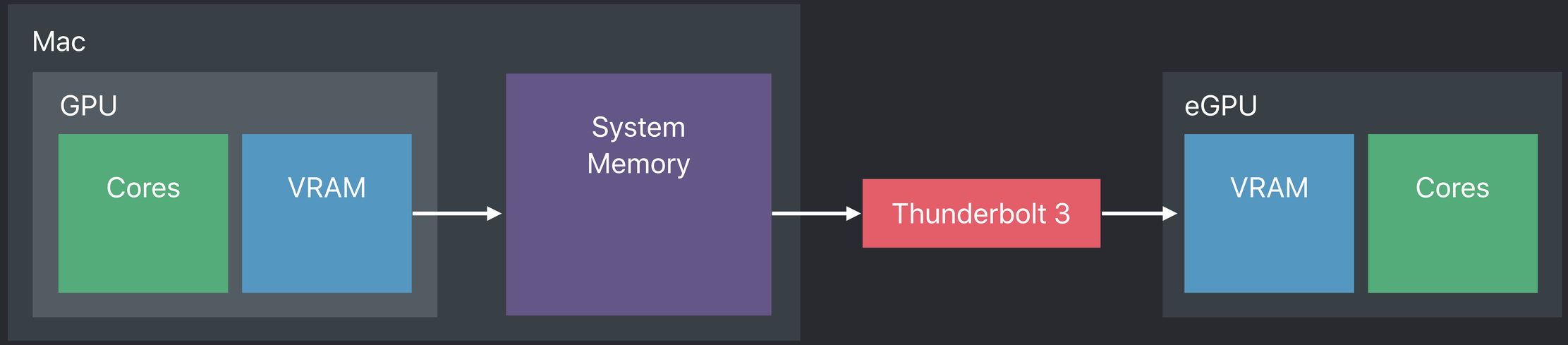

```
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA …]; // . . . Encode Pose-Independent work
[commandA …]; // . . . Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
```



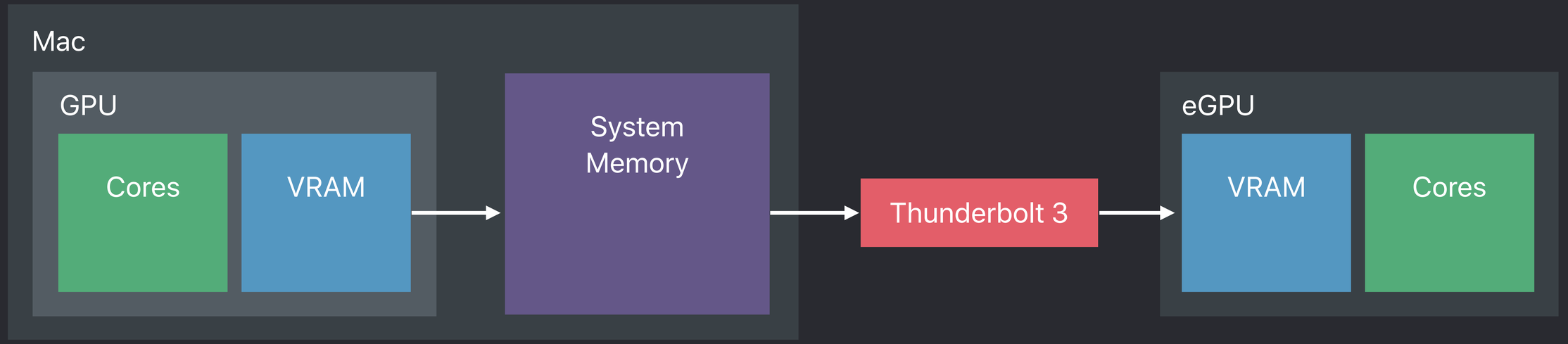

```
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA …]; // . . . Encode Pose-Independent work
[commandA …]; // . . . Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
```



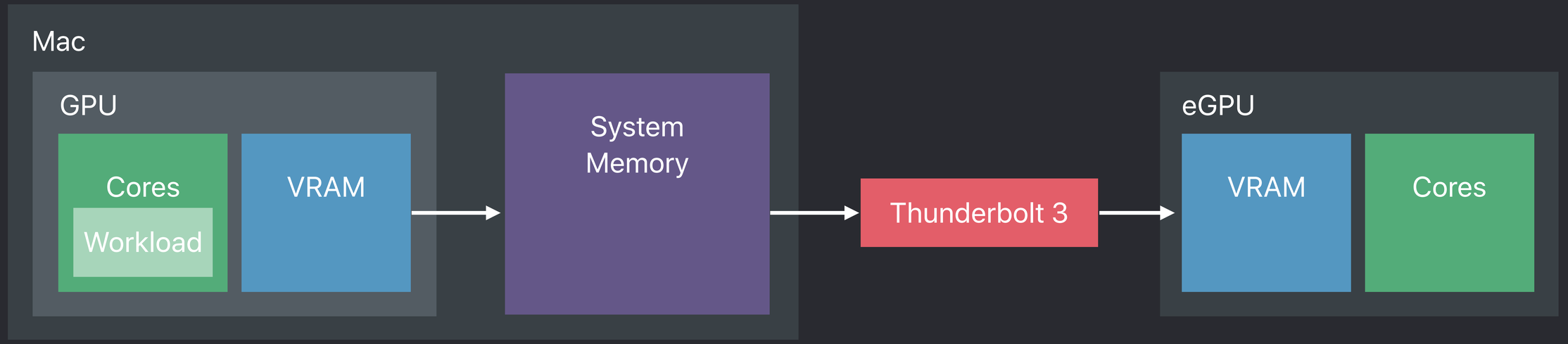

```
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA …]; // . . . Encode Pose-Independent work
[commandA …]; // . . . Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
```



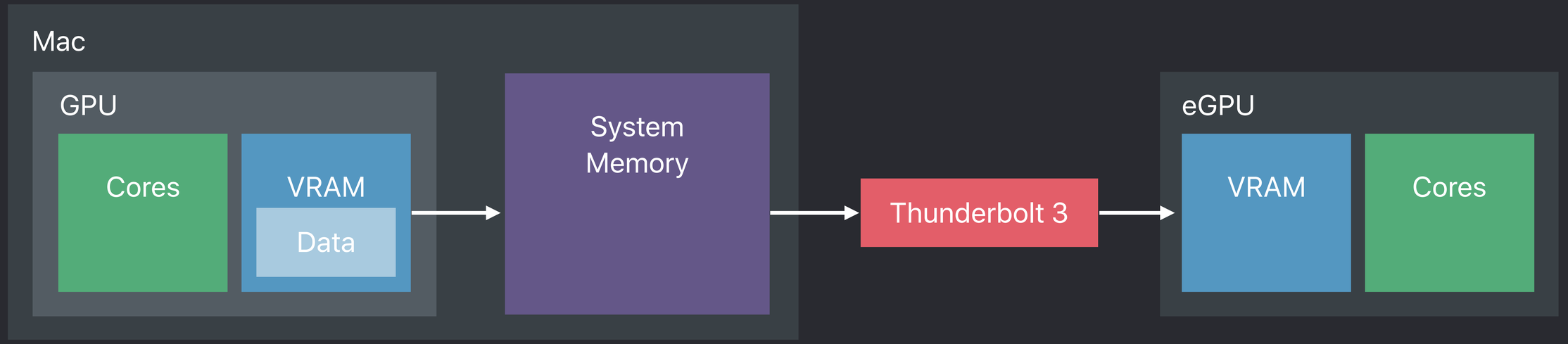

```
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA …]; // . . . Encode Pose-Independent work
[commandA …]; // . . . Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
```



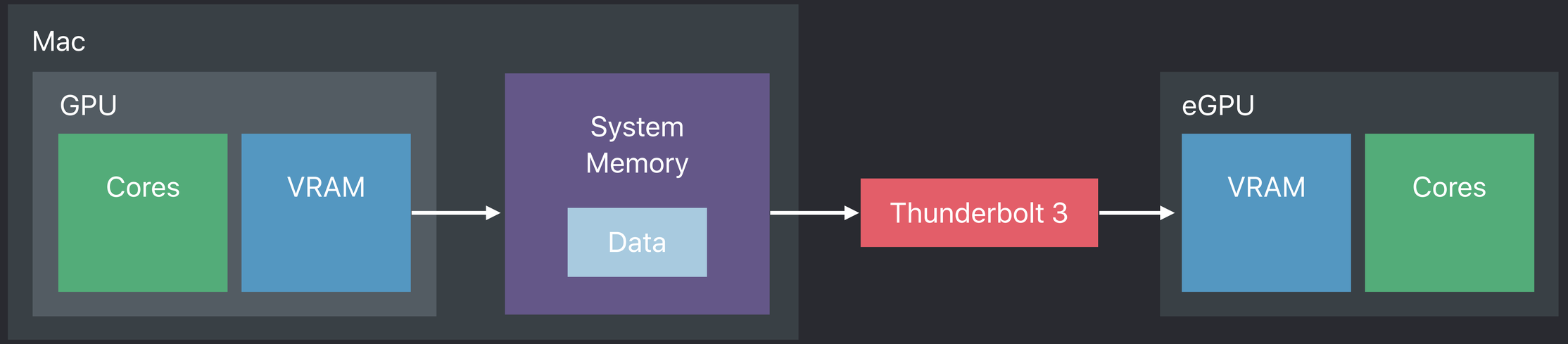

```
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA …]; // . . . Encode Pose-Independent work
[commandA …]; // . . . Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
```



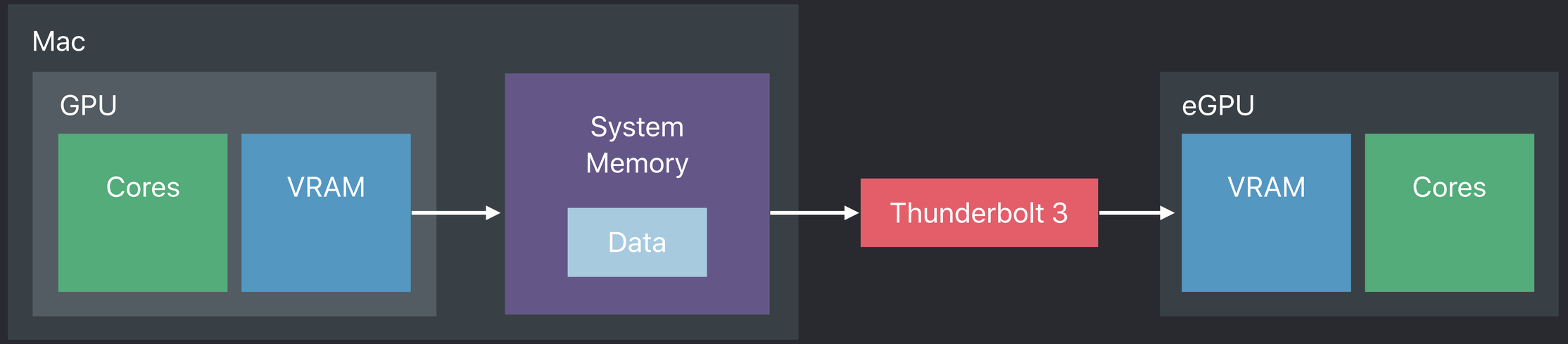

```
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA …]; // . . . Encode Pose-Independent work
[commandA …]; // . . . Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
```



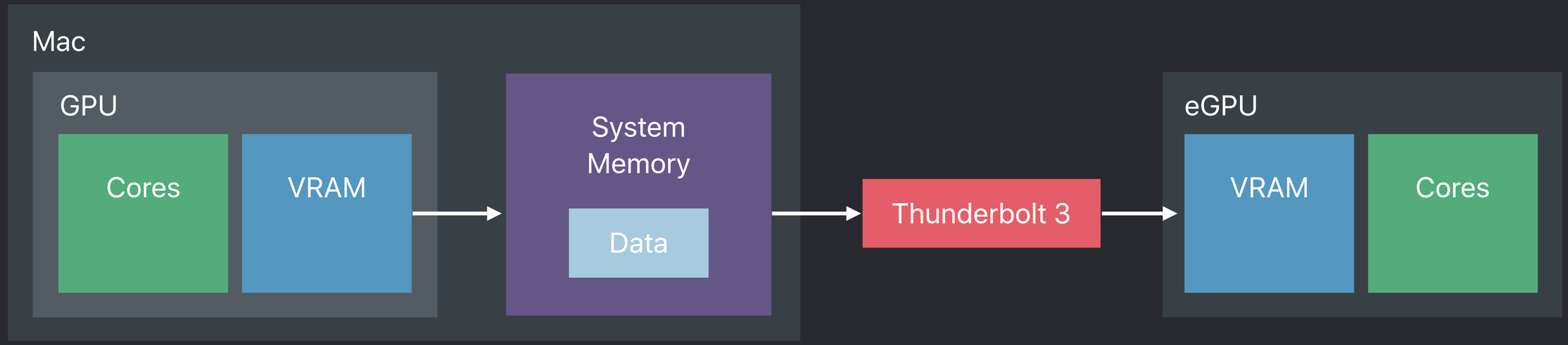

```
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA …]; // . . . Encode Pose-Independent work
[commandA …]; // . . . Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
```



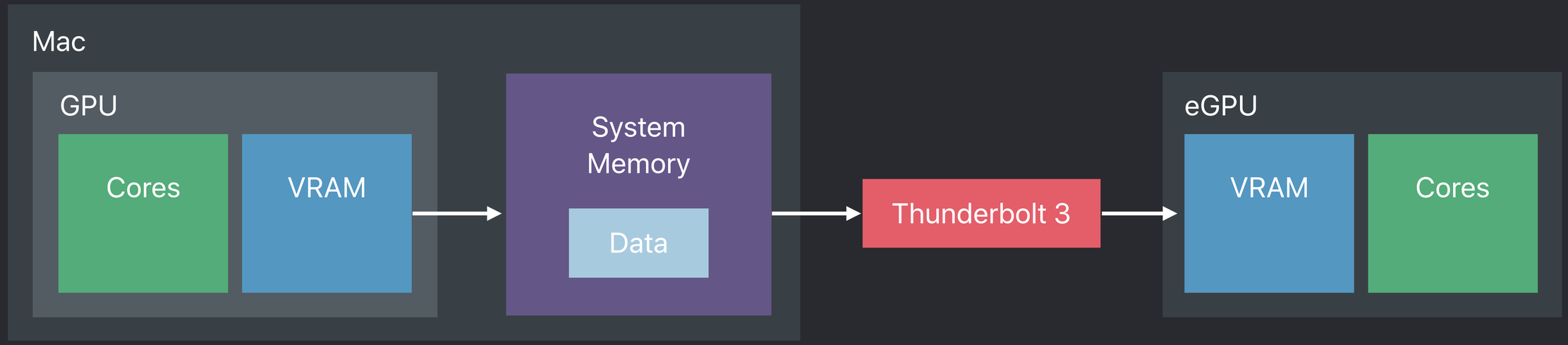

```
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA …]; // . . . Encode Pose-Independent work
[commandA …]; // . . . Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
```



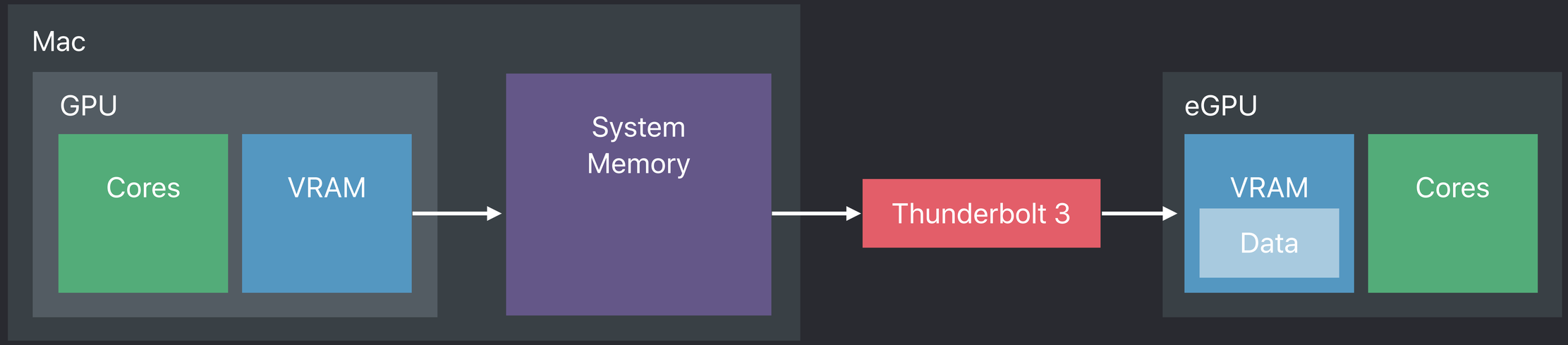

```
id<MTLSharedEvent> syncTransferToRAM = [mainDevice newSharedEvent];
renderingFrame++;

id<MTLCommandBuffer> commandA = [supportingDeviceQueue commandBuffer];
[commandA …]; // . . . Encode Pose-Independent work
[commandA …]; // . . . Encode transfer of computed results from VRAM to RAM using blit encoder
[commandA encodeSignalEvent:syncTransferToRAM value:renderingFrame];
[commandA commit];
```



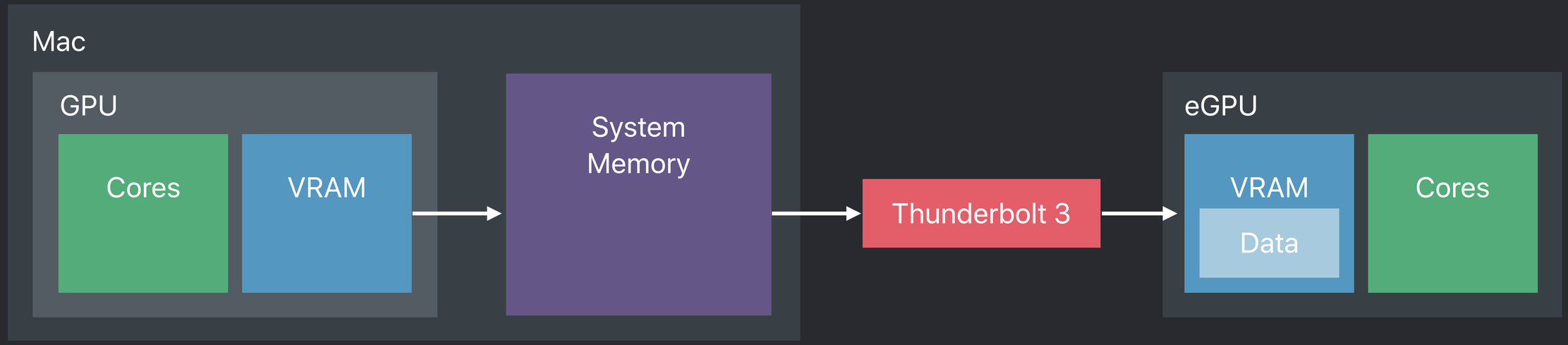

```
id<MTLEvent> syncTransferToGPU = [mainDevice newEvent];

id<MTLCommandBuffer> commandB = [mainDeviceQueue commandBuffer];
[commandB encodeWaitForEvent:syncTransferToRAM value:renderingFrame];
[commandB …]; // . . . Encode transfer of computed results from RAM to main GPU VRAM
[commandB encodeSignalEvent:syncTransferToGPU value:renderingFrame];
[commandB commit];

vr::VRCompositor()->WaitGetPoses();
```



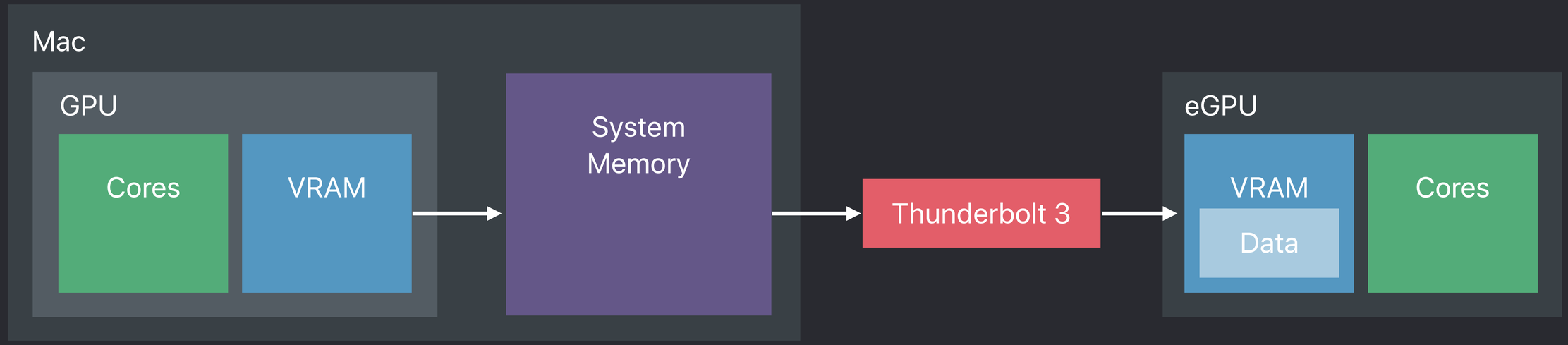

```
id<MTLEvent> syncTransferToGPU = [mainDevice newEvent];

id<MTLCommandBuffer> commandB = [mainDeviceQueue commandBuffer];
[commandB encodeWaitForEvent:syncTransferToRAM value:renderingFrame];
[commandB …]; // . . . Encode transfer of computed results from RAM to main GPU VRAM
[commandB encodeSignalEvent:syncTransferToGPU value:renderingFrame];
[commandB commit];

vr::VRCompositor()->WaitGetPoses();
```




```
id<MTLEvent> syncTransferToGPU = [mainDevice newEvent];

id<MTLCommandBuffer> commandB = [mainDeviceQueue commandBuffer];
[commandB encodeWaitForEvent:syncTransferToRAM value:renderingFrame];
[commandB …]; // . . . Encode transfer of computed results from RAM to main GPU VRAM
[commandB encodeSignalEvent:syncTransferToGPU value:renderingFrame];
[commandB commit];

vr::VRCompositor()->WaitGetPoses();
```




```
id<MTLEvent> syncTransferToGPU = [mainDevice newEvent];

id<MTLCommandBuffer> commandB = [mainDeviceQueue commandBuffer];
[commandB encodeWaitForEvent:syncTransferToRAM value:renderingFrame];
[commandB …]; // . . . Encode transfer of computed results from RAM to main GPU VRAM
[commandB encodeSignalEvent:syncTransferToGPU value:renderingFrame];
[commandB commit];

vr::VRCompositor()->WaitGetPoses();
```

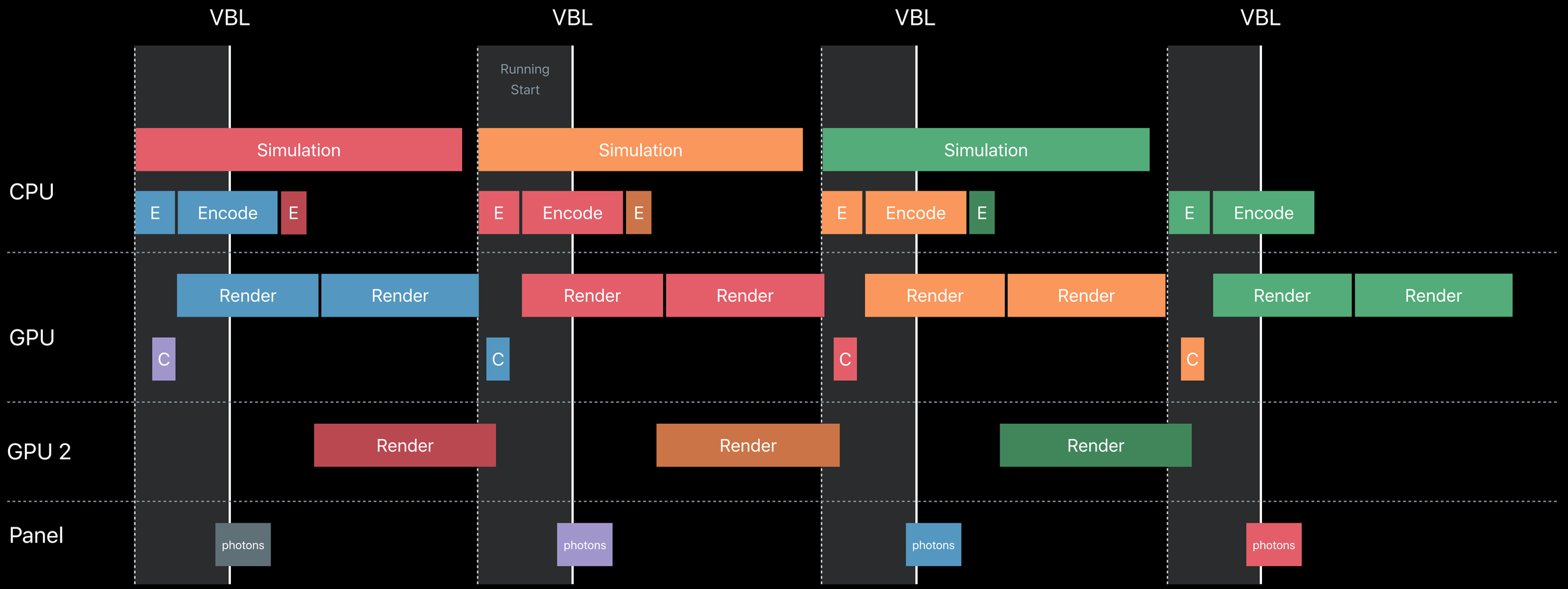



```
id<MTLEvent> syncTransferToGPU = [mainDevice newEvent];

id<MTLCommandBuffer> commandB = [mainDeviceQueue commandBuffer];
[commandB encodeWaitForEvent:syncTransferToRAM value:renderingFrame];
[commandB …]; // . . . Encode transfer of computed results from RAM to main GPU VRAM
[commandB encodeSignalEvent:syncTransferToGPU value:renderingFrame];
[commandB commit];
```

```
vr::VRCompositor()->WaitGetPoses();
```

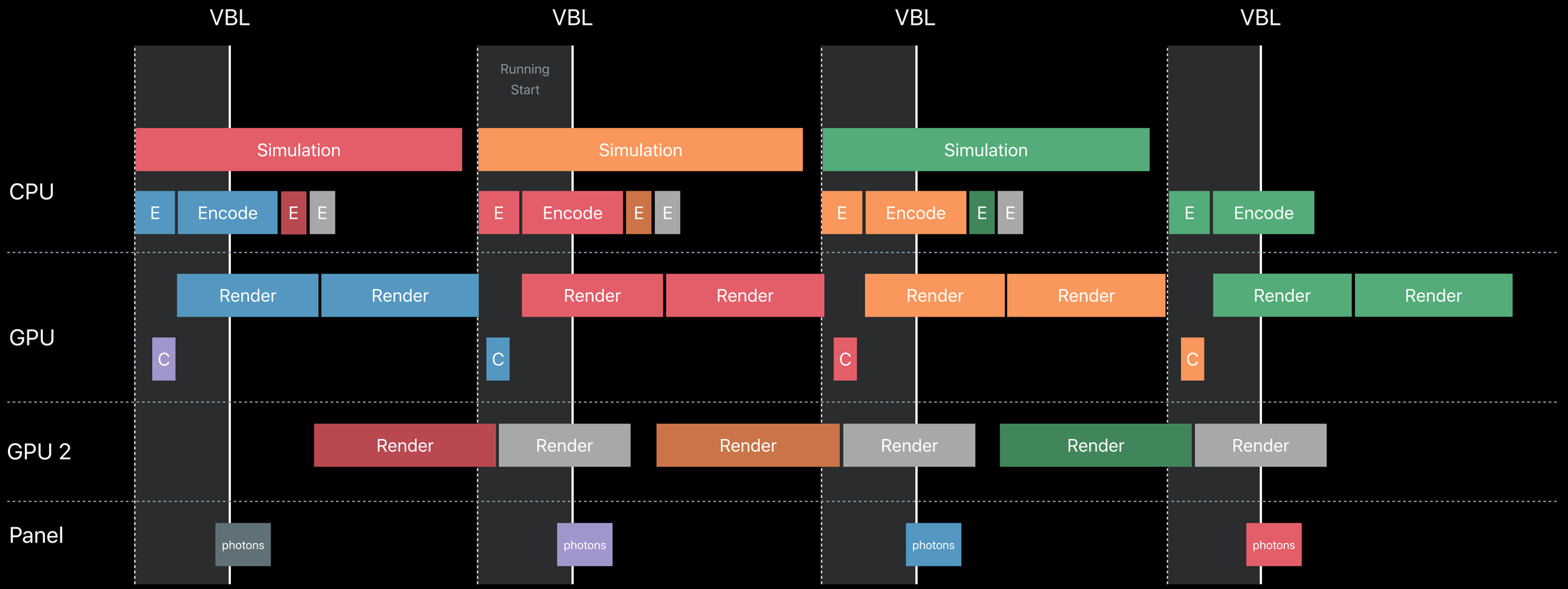
Advanced Frame Pacing

Multi-GPU workload distribution



Advanced Frame Pacing

Multi-GPU workload distribution



Advanced Frame Pacing

Summary

Advanced Frame Pacing

Summary

Multi-thread application

Advanced Frame Pacing

Summary

Multi-thread application

Split command buffers

Advanced Frame Pacing

Summary

Multi-thread application

Split command buffers

Separate pose-independent from pose-dependent workloads

Advanced Frame Pacing

Summary

Multi-thread application

Split command buffers

Separate pose-independent from pose-dependent workloads

Separate workloads by frequency of update

Advanced Frame Pacing

Summary

Multi-thread application

Split command buffers

Separate pose-independent from pose-dependent workloads

Separate workloads by frequency of update

Benefit from multi-GPU configurations

Advanced Frame Pacing

Summary

Multi-thread application

Split command buffers

Separate pose-independent from pose-dependent workloads

Separate workloads by frequency of update

Benefit from multi-GPU configurations

Rendering thread per GPU

Advanced Frame Pacing

Reducing Fill Rate

Increasing Resolutions

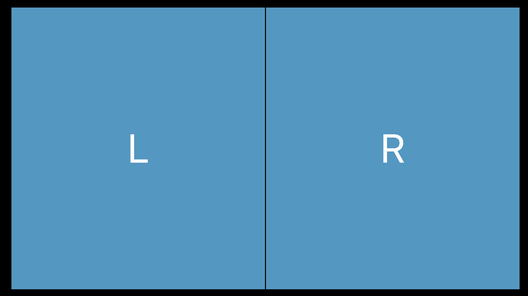
Optimizing for Vive Pro

Rendering at higher resolution

Increasing Resolutions

Optimizing for Vive Pro

Rendering at higher resolution

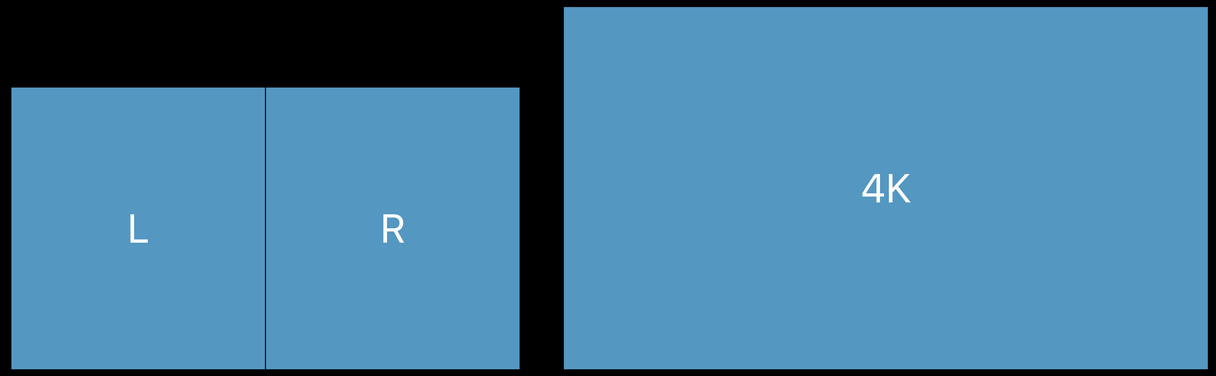


Vive @ 90Hz
436MP/s

Increasing Resolutions

Optimizing for Vive Pro

Rendering at higher resolution



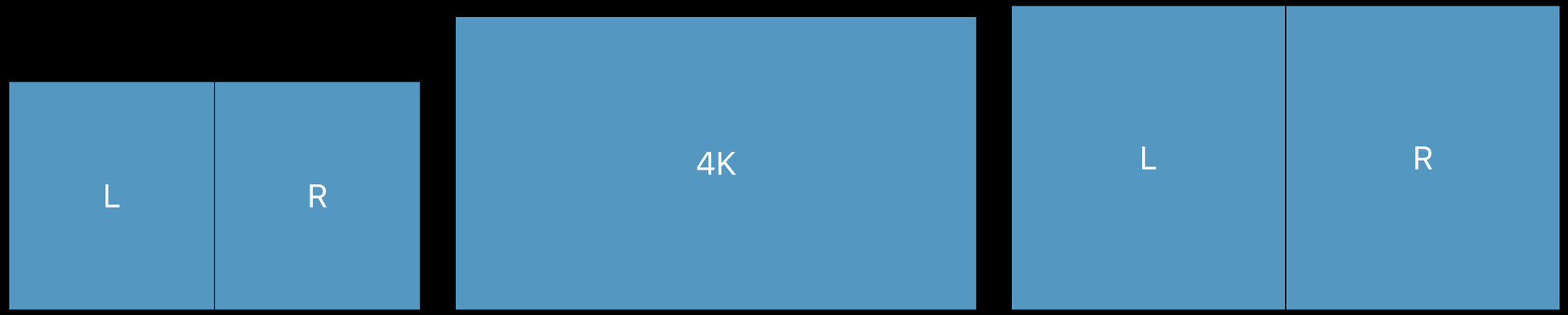
Vive @ 90Hz
436MP/s

UHD TV @ 60Hz
475MP/s

Increasing Resolutions

Optimizing for Vive Pro

Rendering at higher resolution



Vive @ 90Hz
436MP/s

UHD TV @ 60Hz
475MP/s

Vive Pro @ 90Hz
775MP/s

Reducing Fill Rate

Clipping invisible pixels

Multi-resolution shading

Monoscopic far-field rendering

Stereo shading reprojection

Reducing Fill Rate

Clipping invisible pixels

Multi-resolution shading

Monoscopic far-field rendering

Stereo shading reprojection

Reducing Fill Rate

Cliping invisible pixels

Loss of information
in periphery due to lens
distortion correction

Reducing Fill Rate

Cliping invisible pixels

Loss of information
in periphery due to lens
distortion correction

Reducing Fill Rate

Cliping invisible pixels

Use SteamVR stencil mask to clip invisible pixels

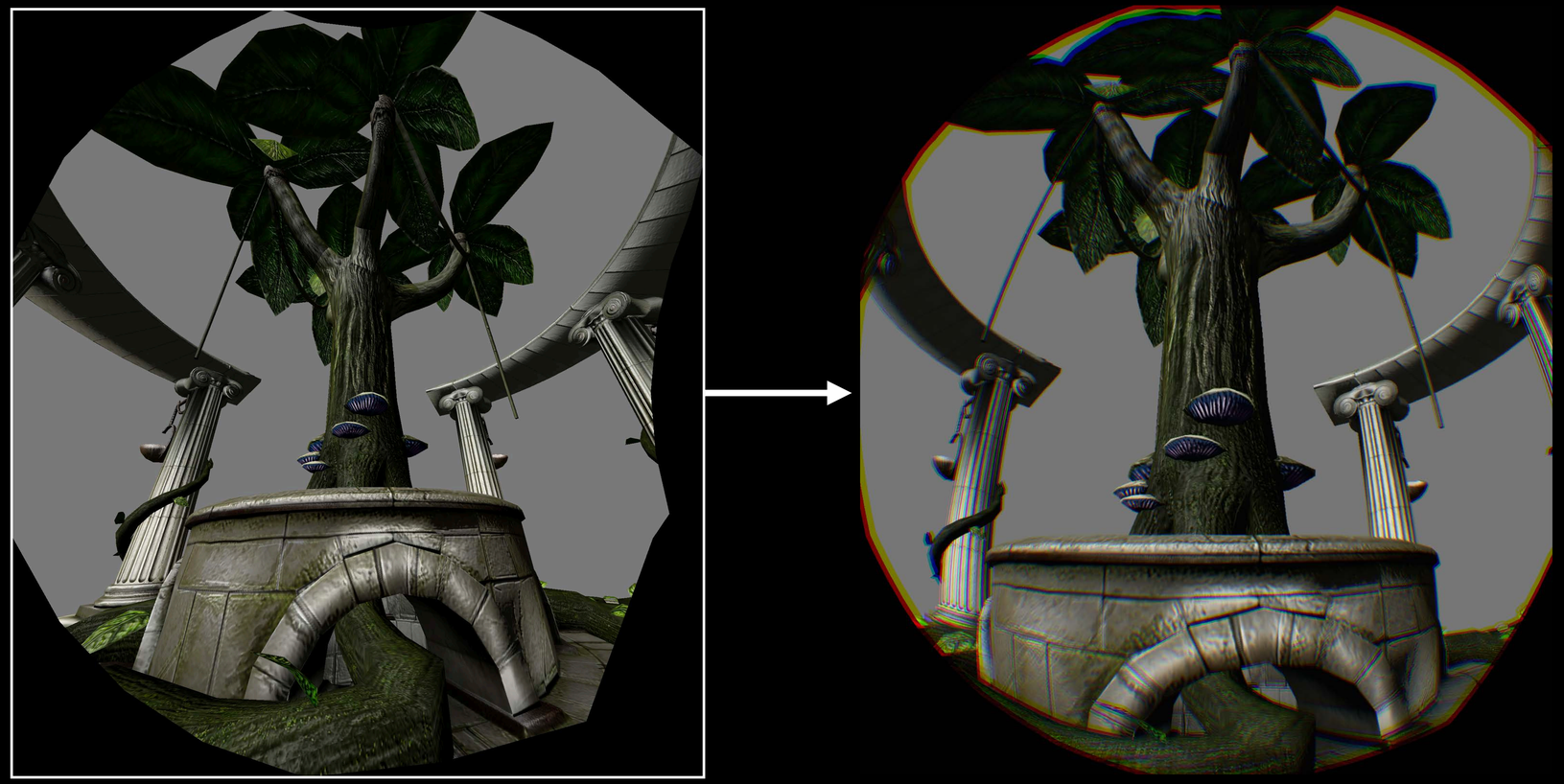
VivePro
775MP/s

Reducing Fill Rate

Cliping invisible pixels

Use SteamVR stencil mask to clip invisible pixels

VivePro
775MP/s
620MP/s



Reducing Fill Rate

Multi-resolution shading

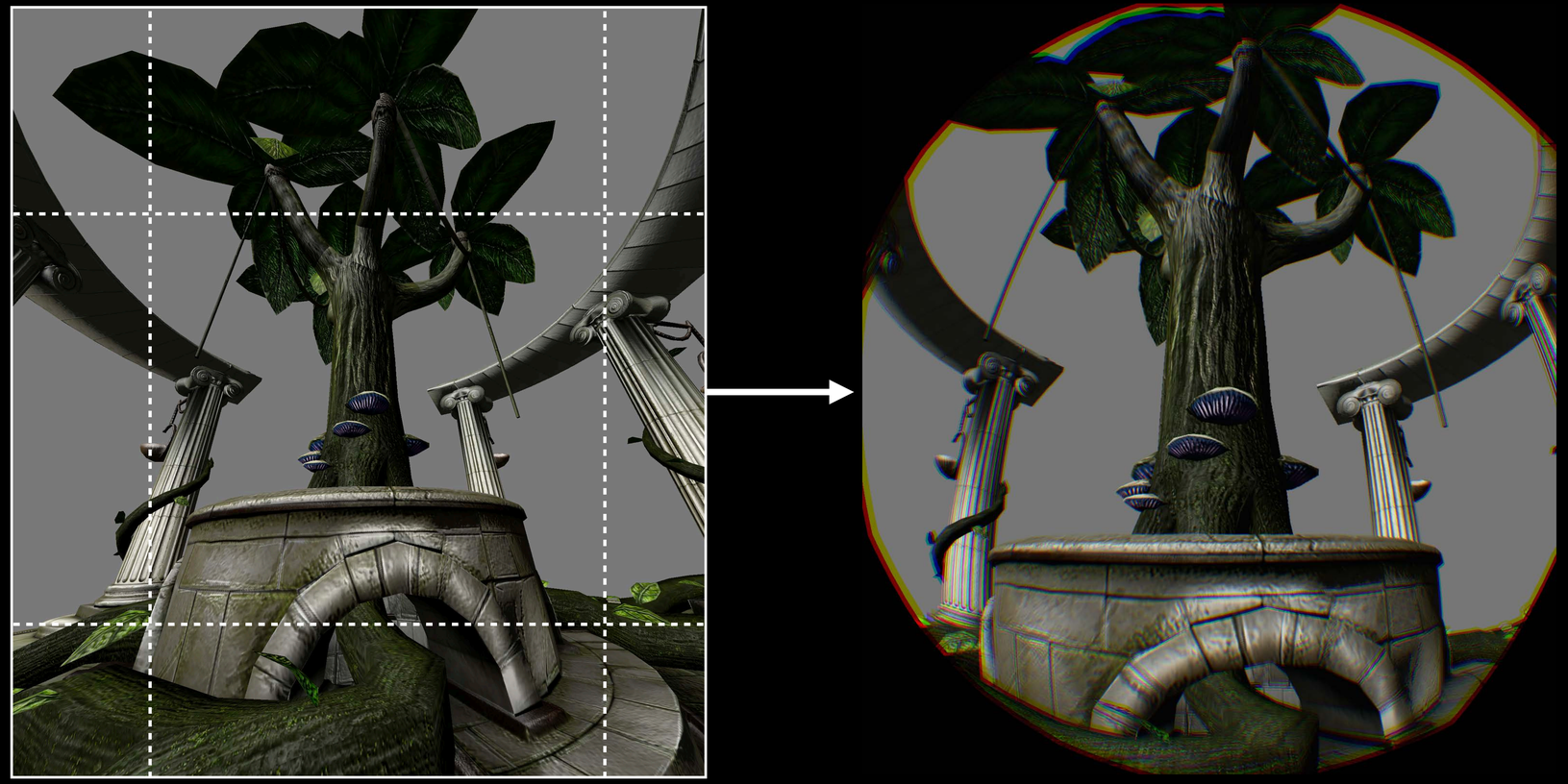
Contribution of edge and corner regions

Reducing Fill Rate

Multi-resolution shading

Contribution of edge and corner regions

80° x 80°

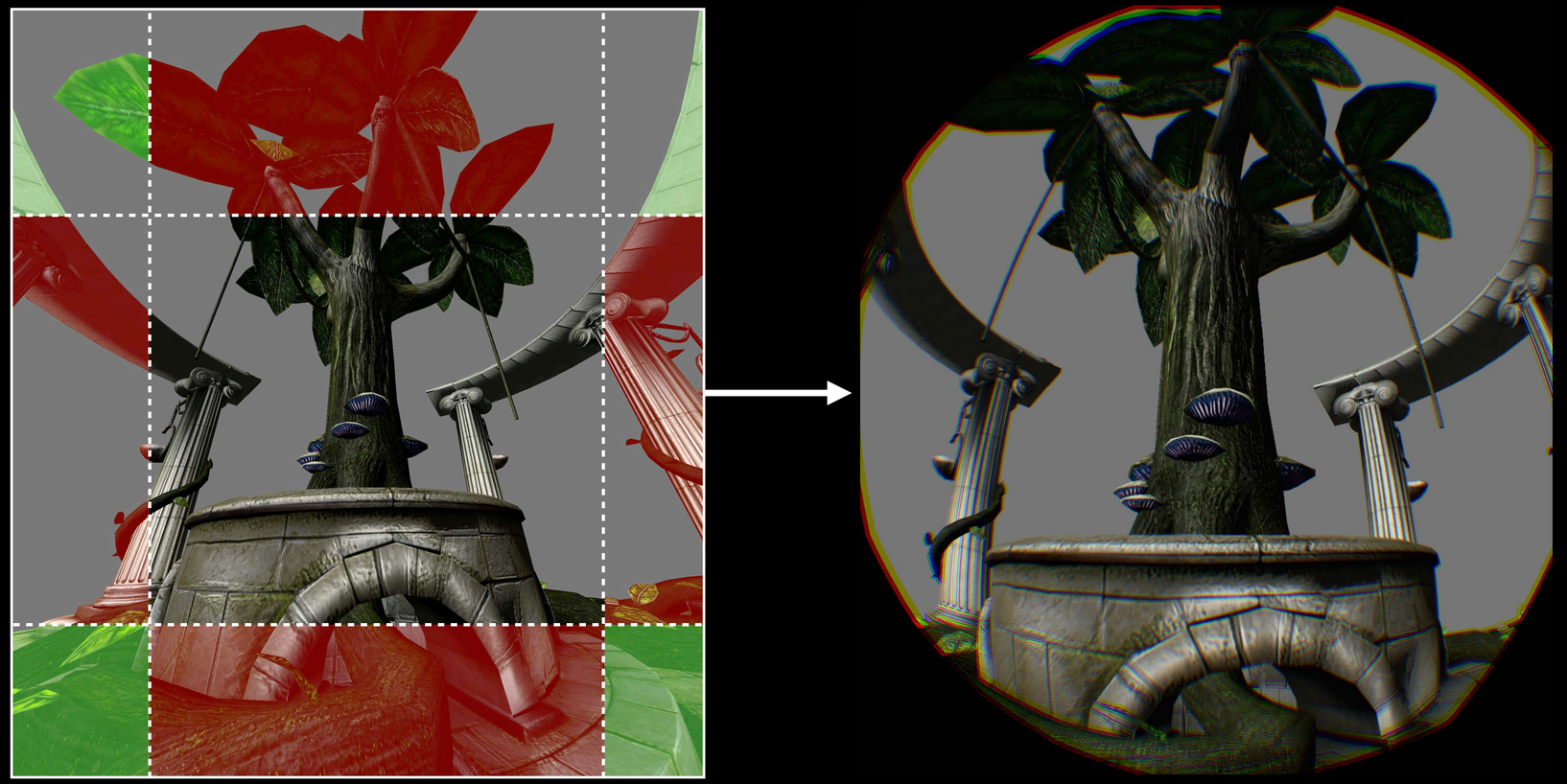


Reducing Fill Rate

Multi-resolution shading

Contribution of edge and corner regions

80° x 80°

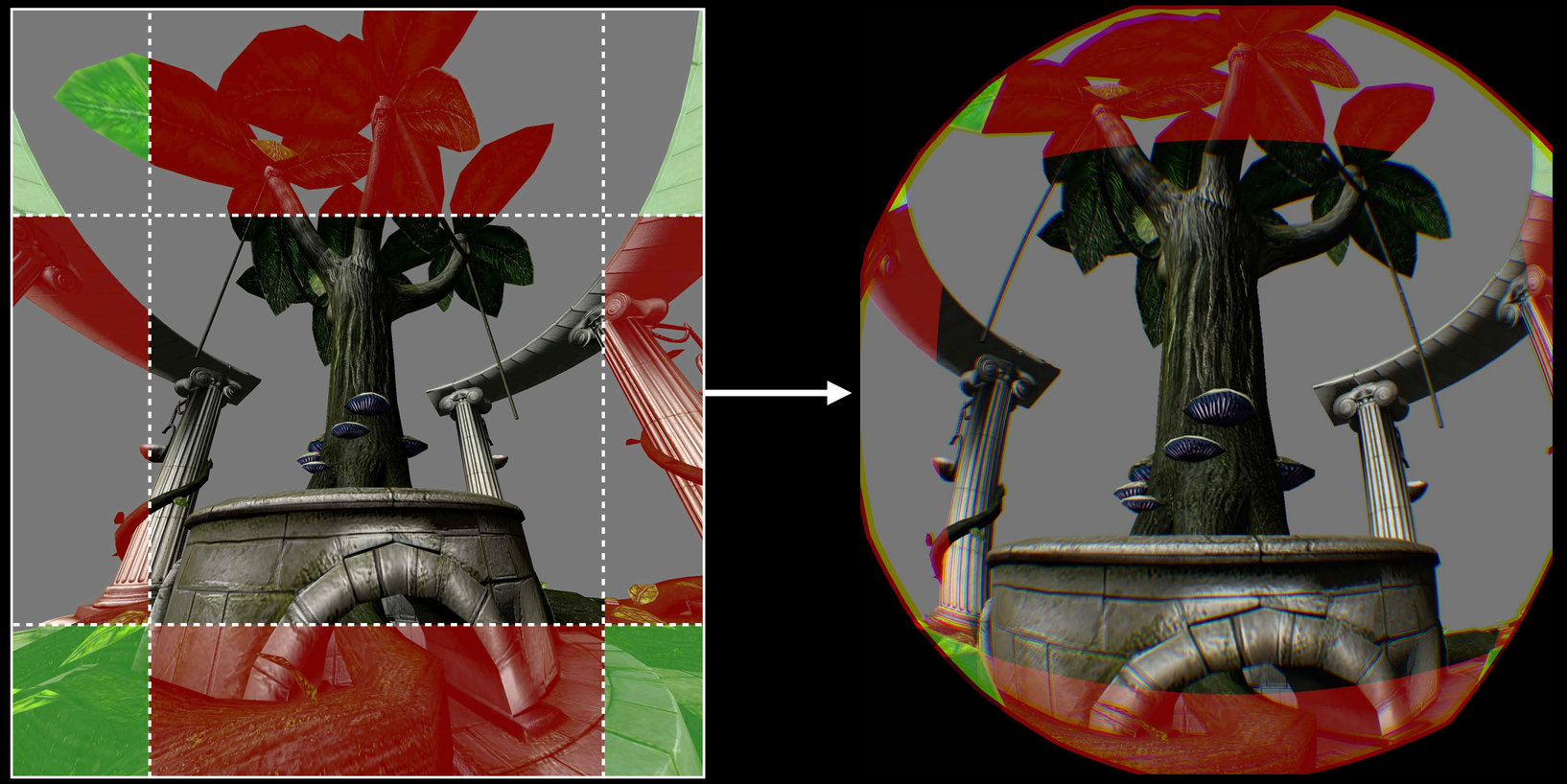


Reducing Fill Rate

Multi-resolution shading

Contribution of edge and corner regions

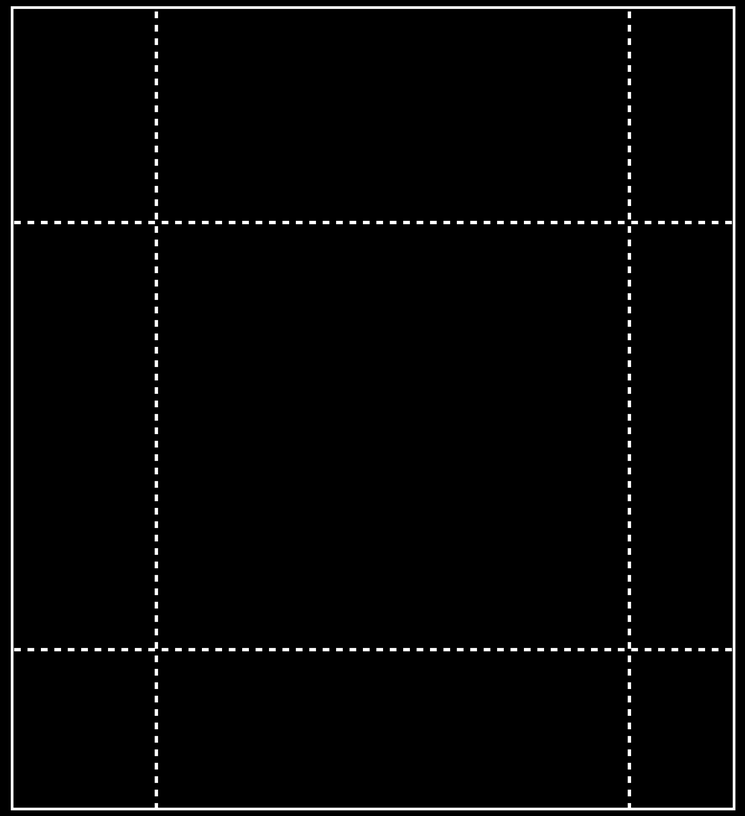
80° x 80°

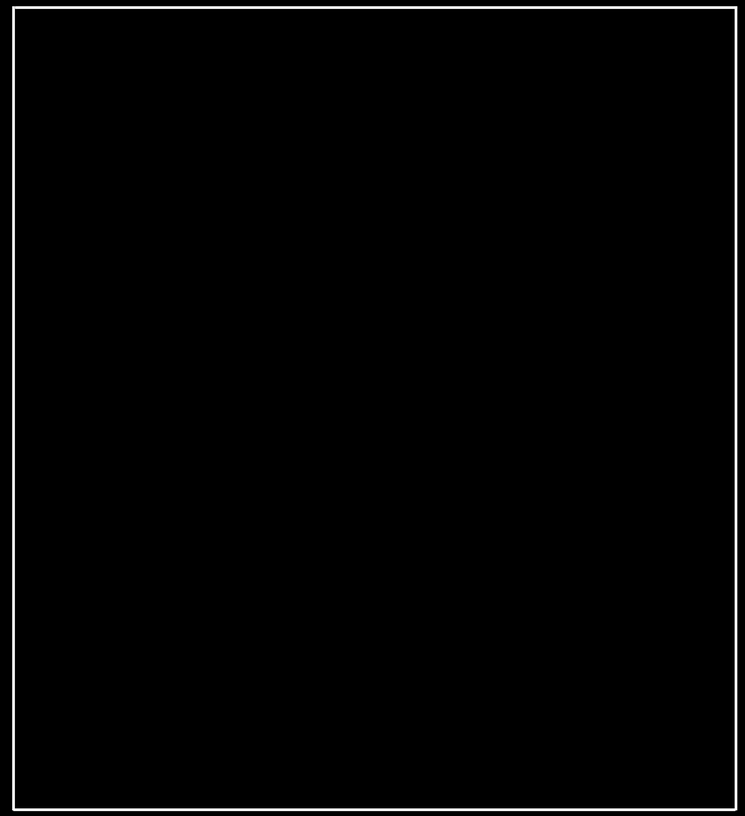


Reducing Fill Rate

Multi-resolution shading

Rendering edge and corner regions at reduced resolution

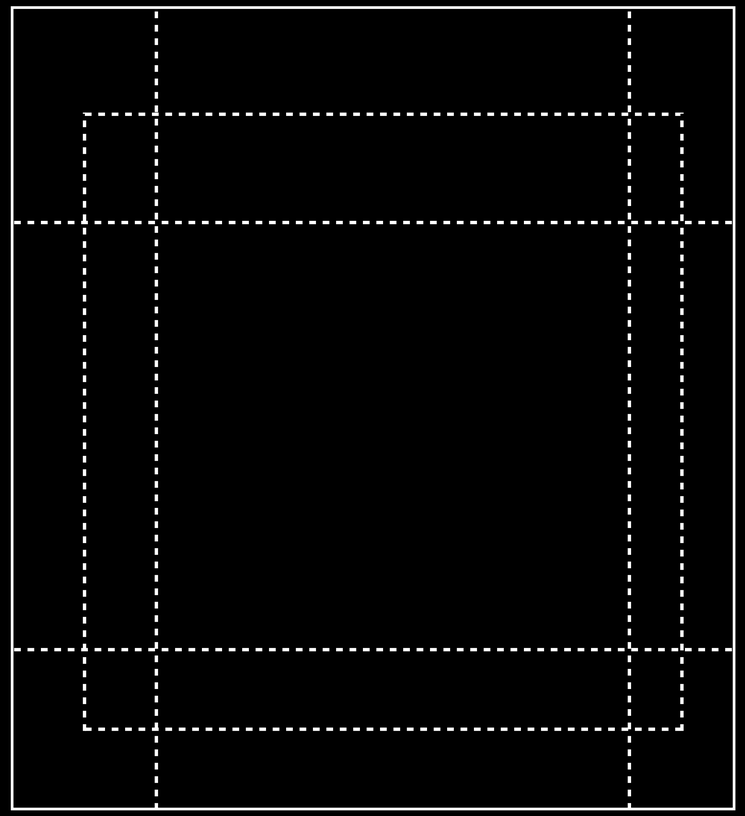


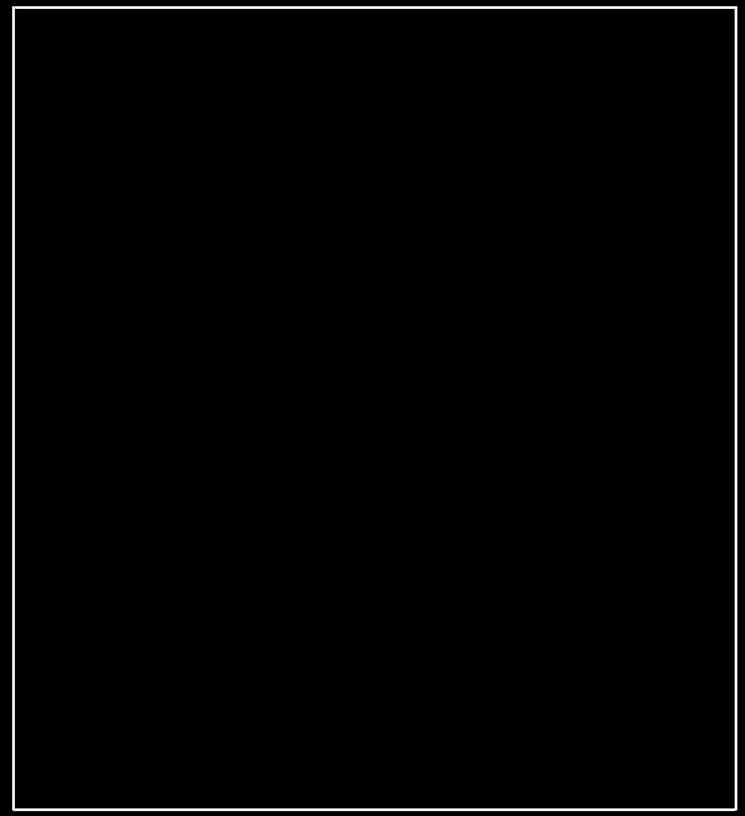


Reducing Fill Rate

Multi-resolution shading

Rendering edge and corner regions at reduced resolution

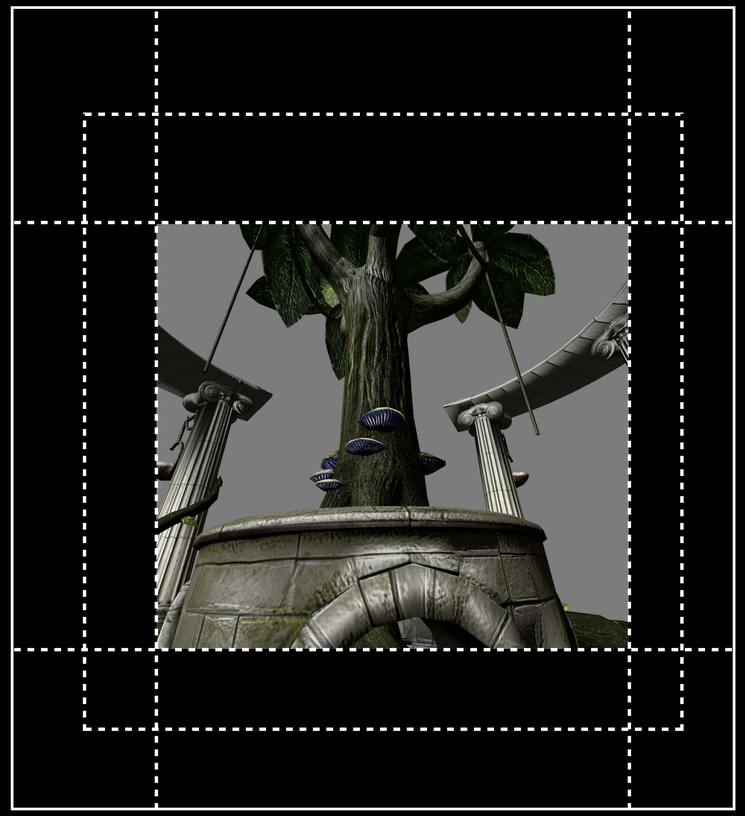


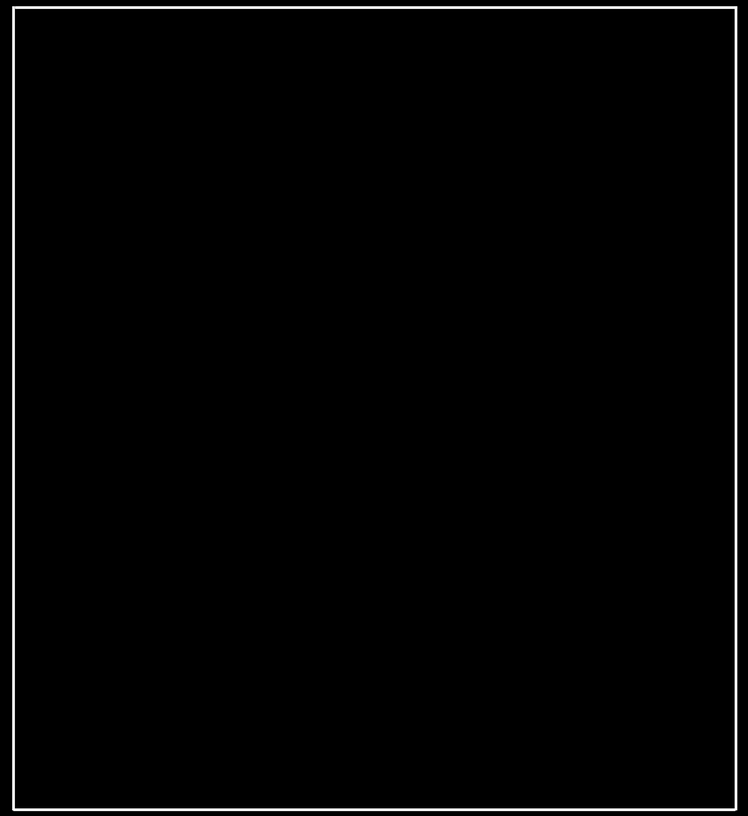


Reducing Fill Rate

Multi-resolution shading

Rendering edge and corner regions at reduced resolution

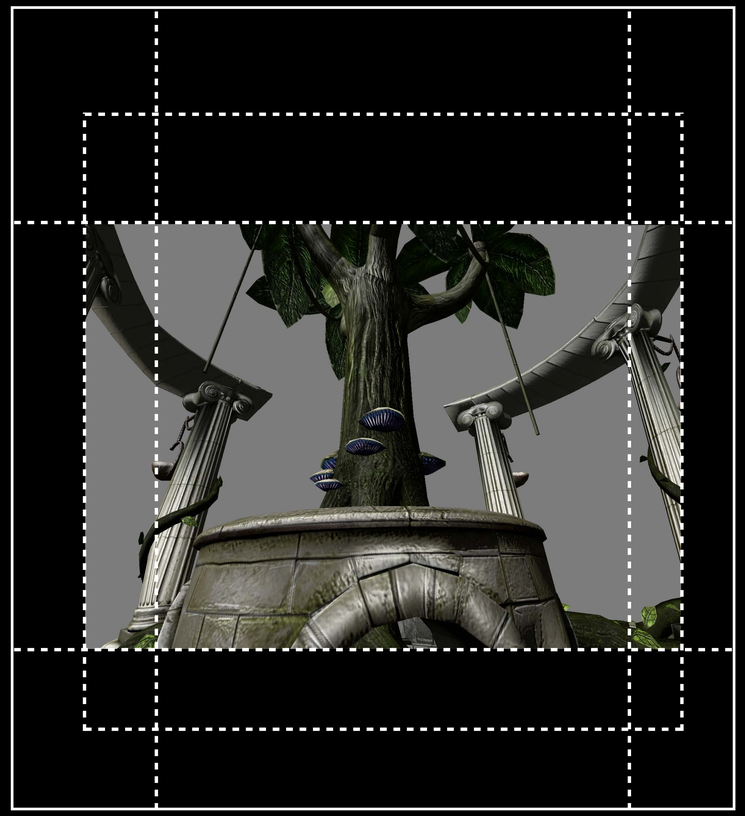


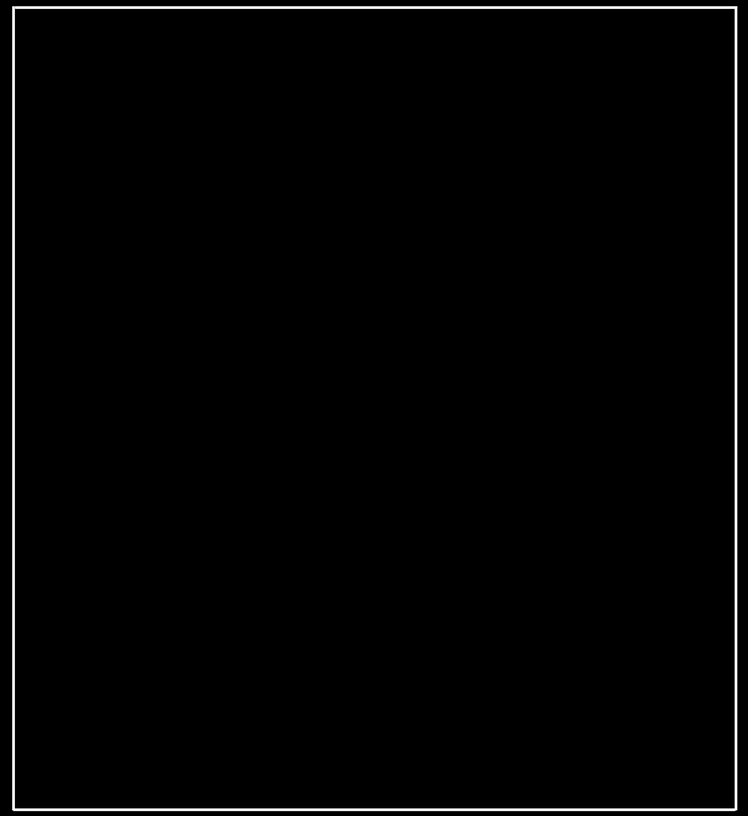


Reducing Fill Rate

Multi-resolution shading

Rendering edge and corner regions at reduced resolution

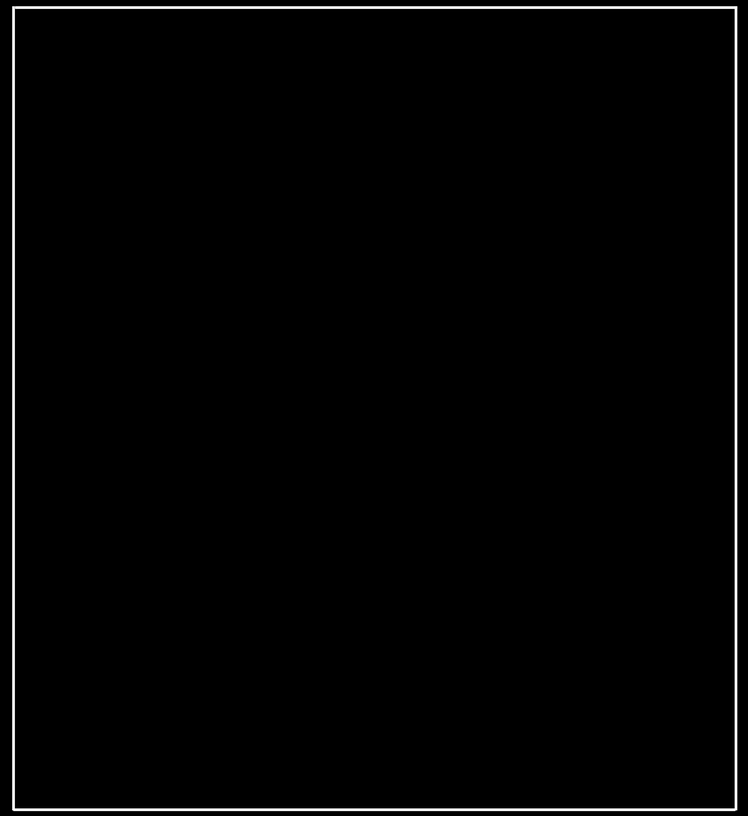




Reducing Fill Rate

Multi-resolution shading

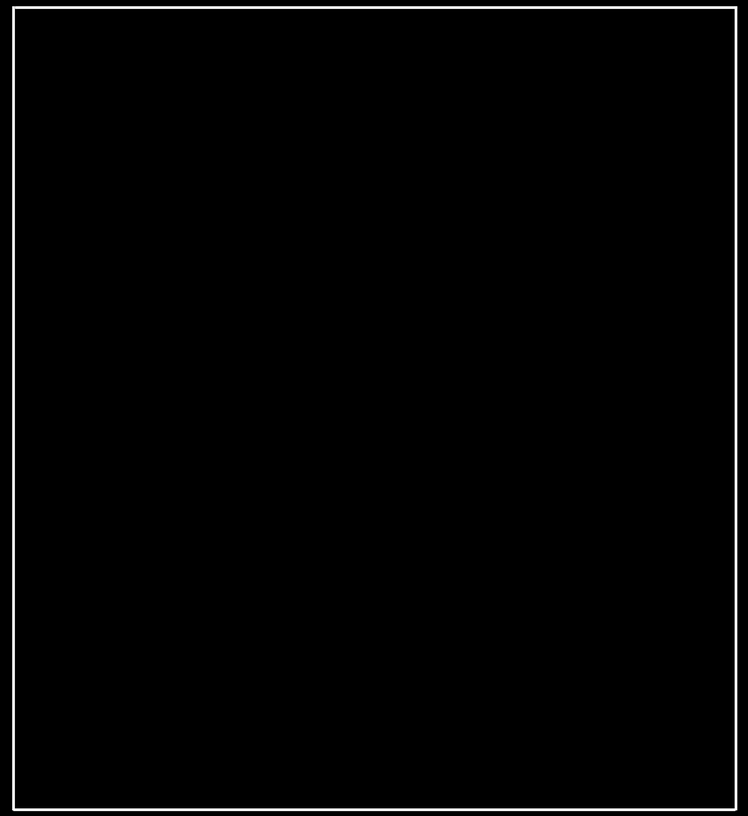
Rendering edge and corner regions at reduced resolution



Reducing Fill Rate

Multi-resolution shading

Rendering edge and corner regions at reduced resolution



Reducing Fill Rate

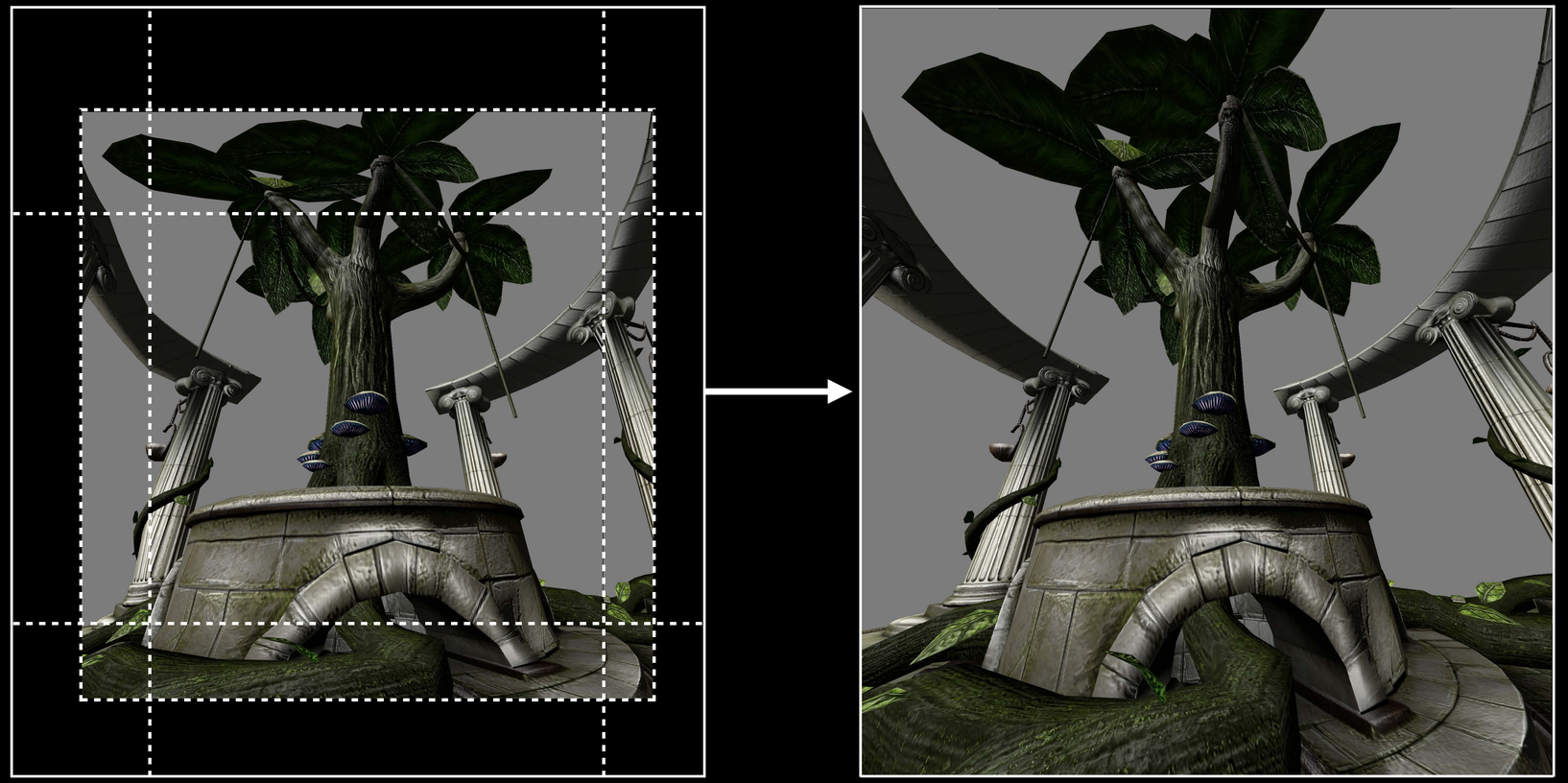
Multi-resolution shading

Rendering edge and corner regions at reduced resolution

Reducing Fill Rate

Multi-resolution shading

Upscaling rendered regions to final resolution for submission



Reducing Fill Rate

Multi-resolution shading

Fill rate reduction

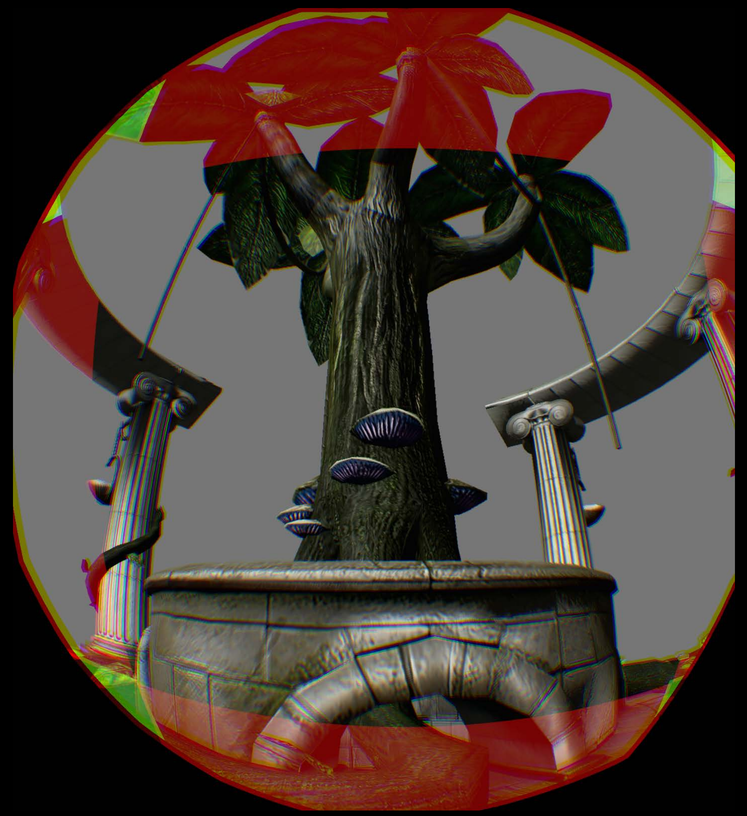
VivePro
775MP/s

Reducing Fill Rate

Multi-resolution shading

Fill rate reduction

	VivePro
	775MP/s
80° x 80°	491MP/s

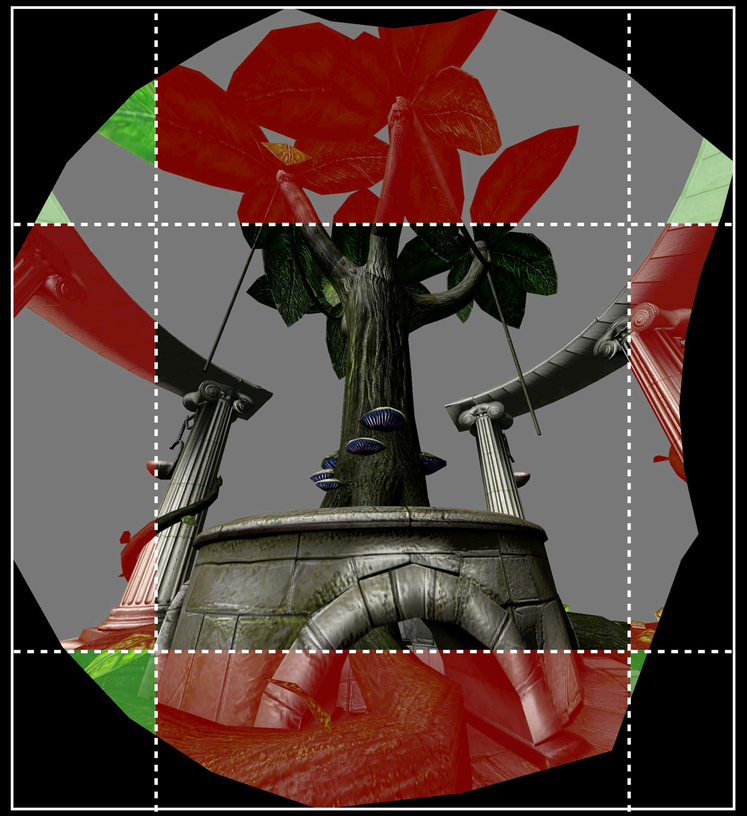


Reducing Fill Rate

Multi-resolution shading

Fill rate reduction

	VivePro	(Clipped)
	775MP/s	620MP/s
80° x 80°	491MP/s	436MP/s

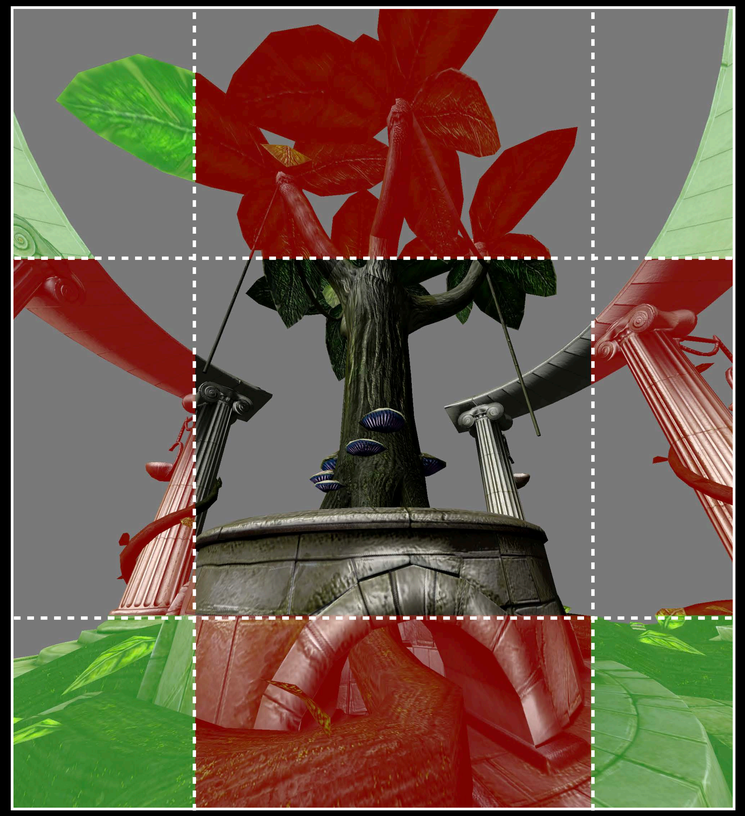


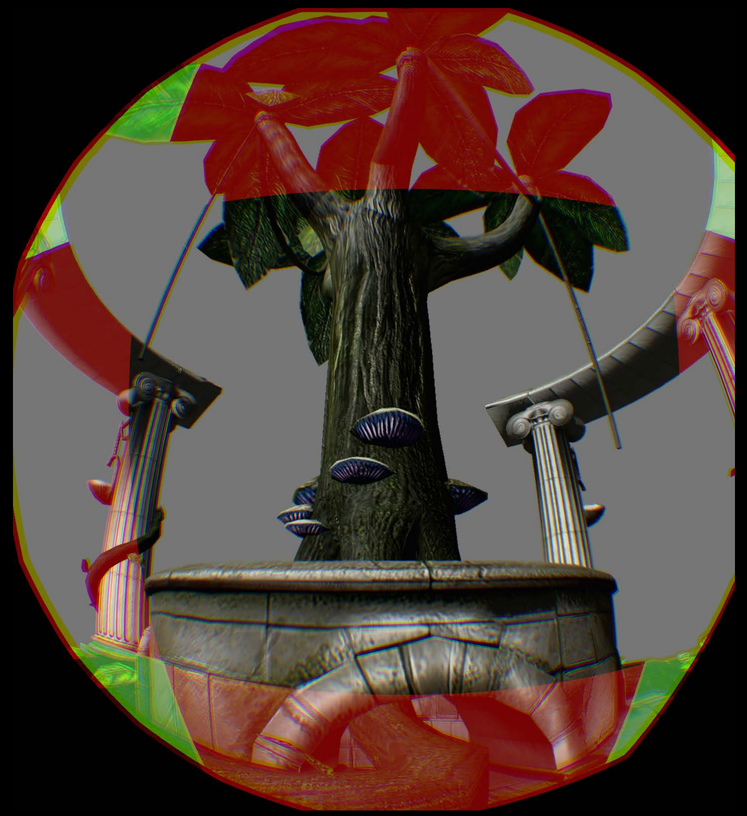
Reducing Fill Rate

Multi-resolution shading

Fill rate reduction

	VivePro	(Clipped)
	775MP/s	620MP/s
80° x 80°	491MP/s	436MP/s
70° x 70°	433MP/s	



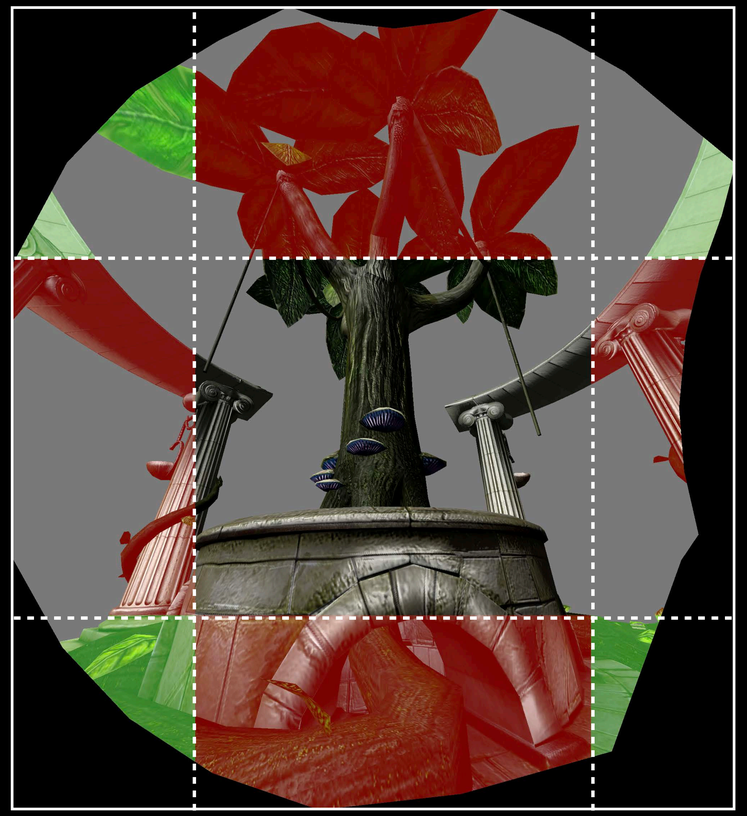


Reducing Fill Rate

Multi-resolution shading

Fill rate reduction

	VivePro	(Clipped)
	775MP/s	620MP/s
80° x 80°	491MP/s	436MP/s
70° x 70°	433MP/s	382MP/s

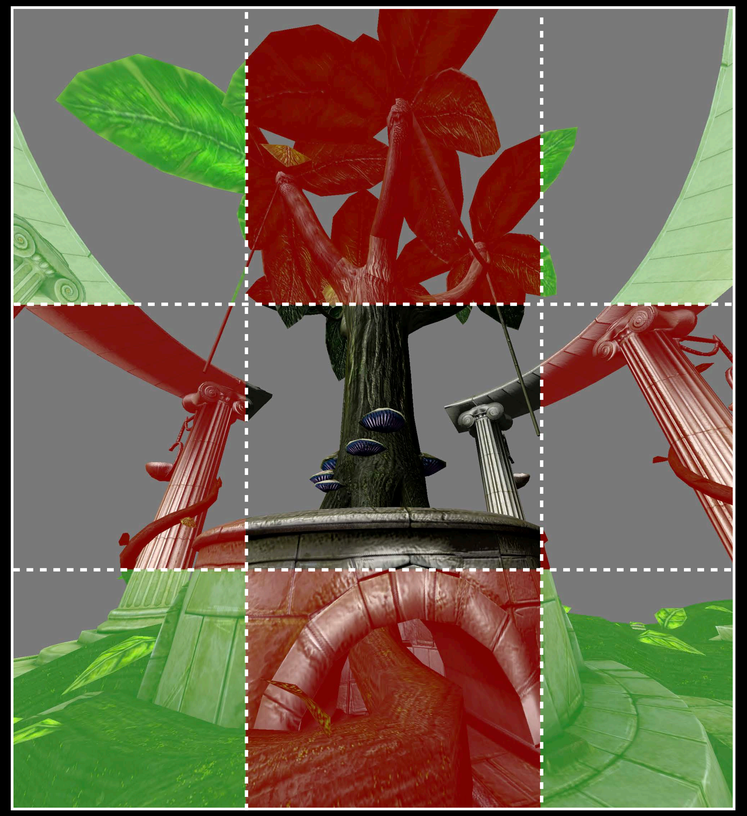


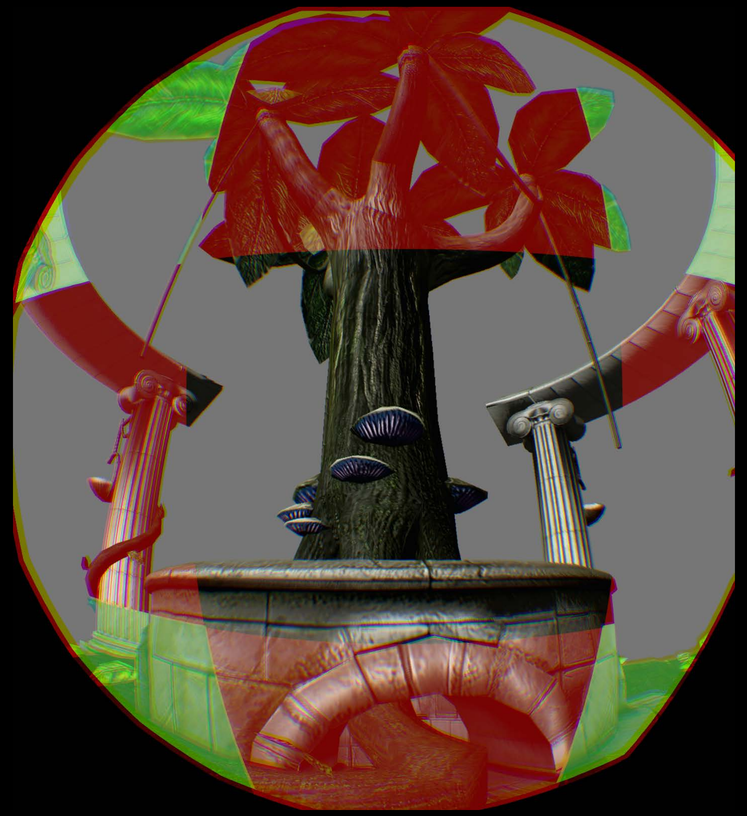
Reducing Fill Rate

Multi-resolution shading

Fill rate reduction

	VivePro	(Clipped)
	775MP/s	620MP/s
80° x 80°	491MP/s	436MP/s
70° x 70°	433MP/s	382MP/s
55° x 55°	362MP/s	



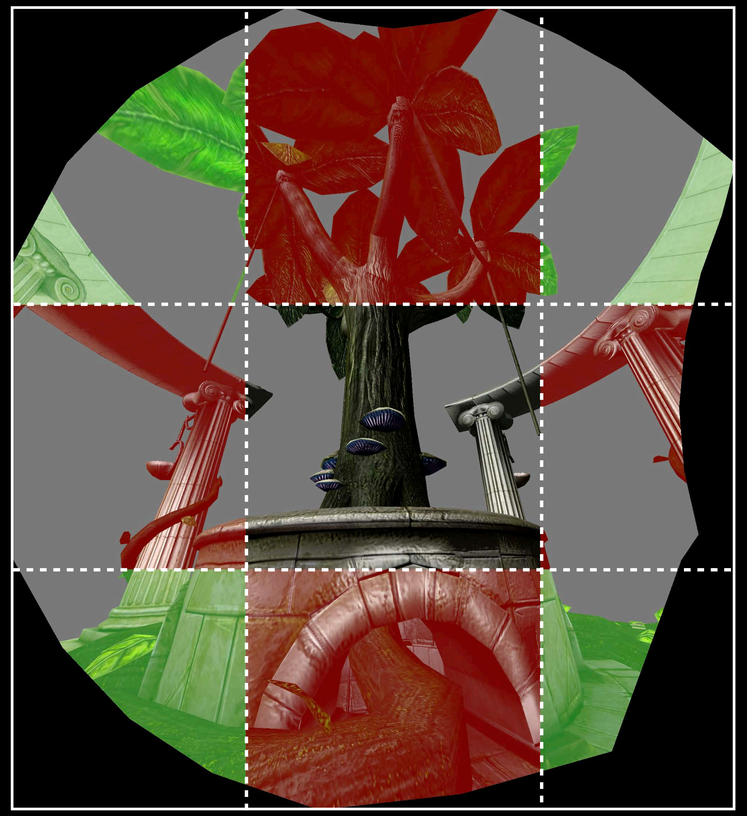


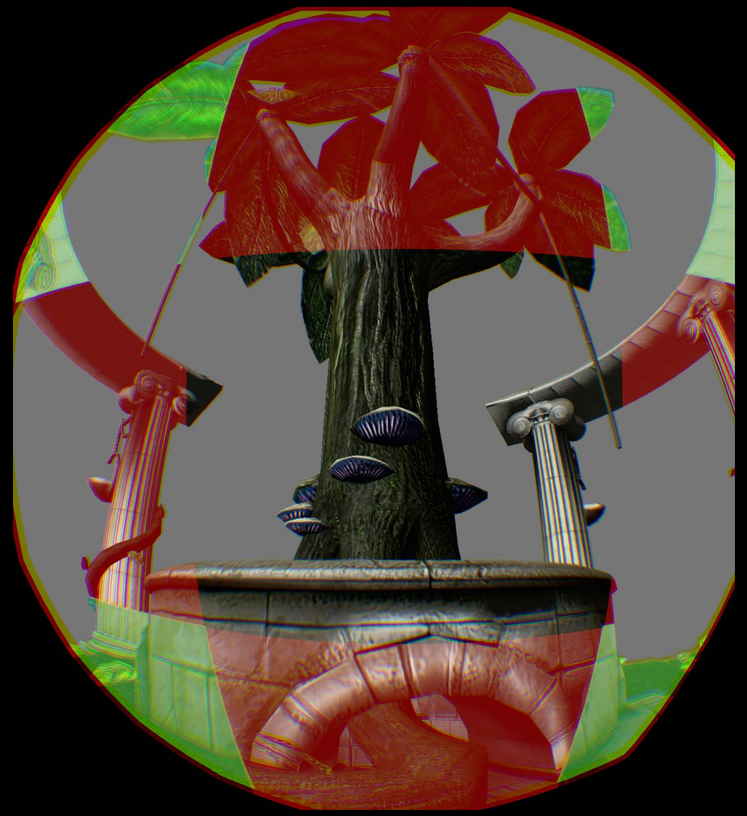
Reducing Fill Rate

Multi-resolution shading

Fill rate reduction

	VivePro	(Clipped)
	775MP/s	620MP/s
80° x 80°	491MP/s	436MP/s
70° x 70°	433MP/s	382MP/s
55° x 55°	362MP/s	316MP/s





Summary

Support for HTC VivePro

Advanced VR development enabled with Metal 2

Take advantage of multi-GPU

More Information

https://developer.apple.com/wwdc18/611

Metal for VR Lab	Technology Lab 6	Friday 12:00PM


WWDC18