#WWDC19

Metal for Pro Apps

Eugene Zhidkov, GPU Software
Dileep Madhava, GPU Software
Brian Ross, GPU Software

What Are Pro Apps?

What Are Pro Apps?

Used by professional content creators for the media-making business

- Animated, live TV, and film
- Photography
- 3D animation and games
- Print media
- Audio production

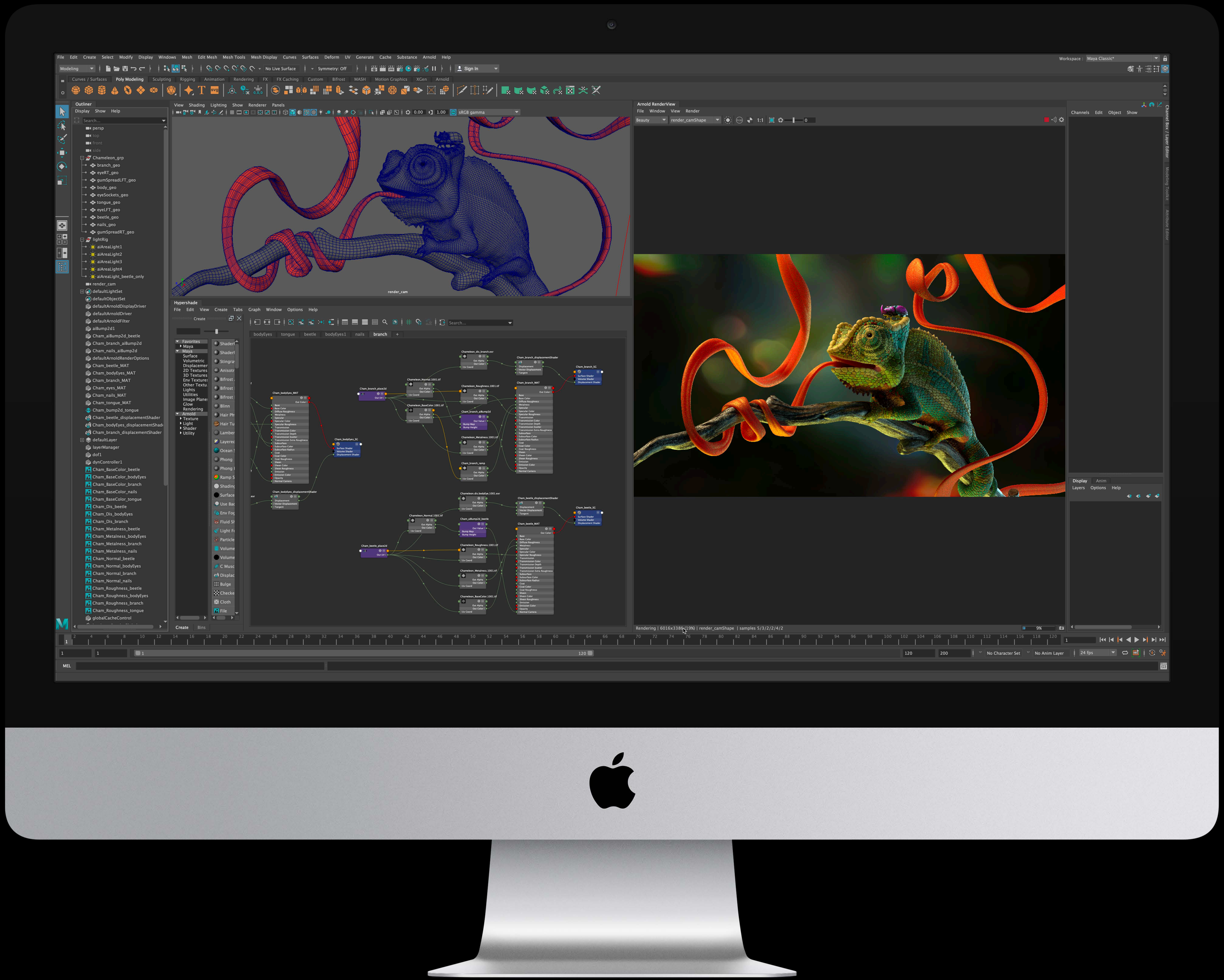
What Are Pro Apps?

Used by professional content creators for the media-making business

- Animated, live TV, and film
- Photography
- 3D animation and games
- Print media
- Audio production

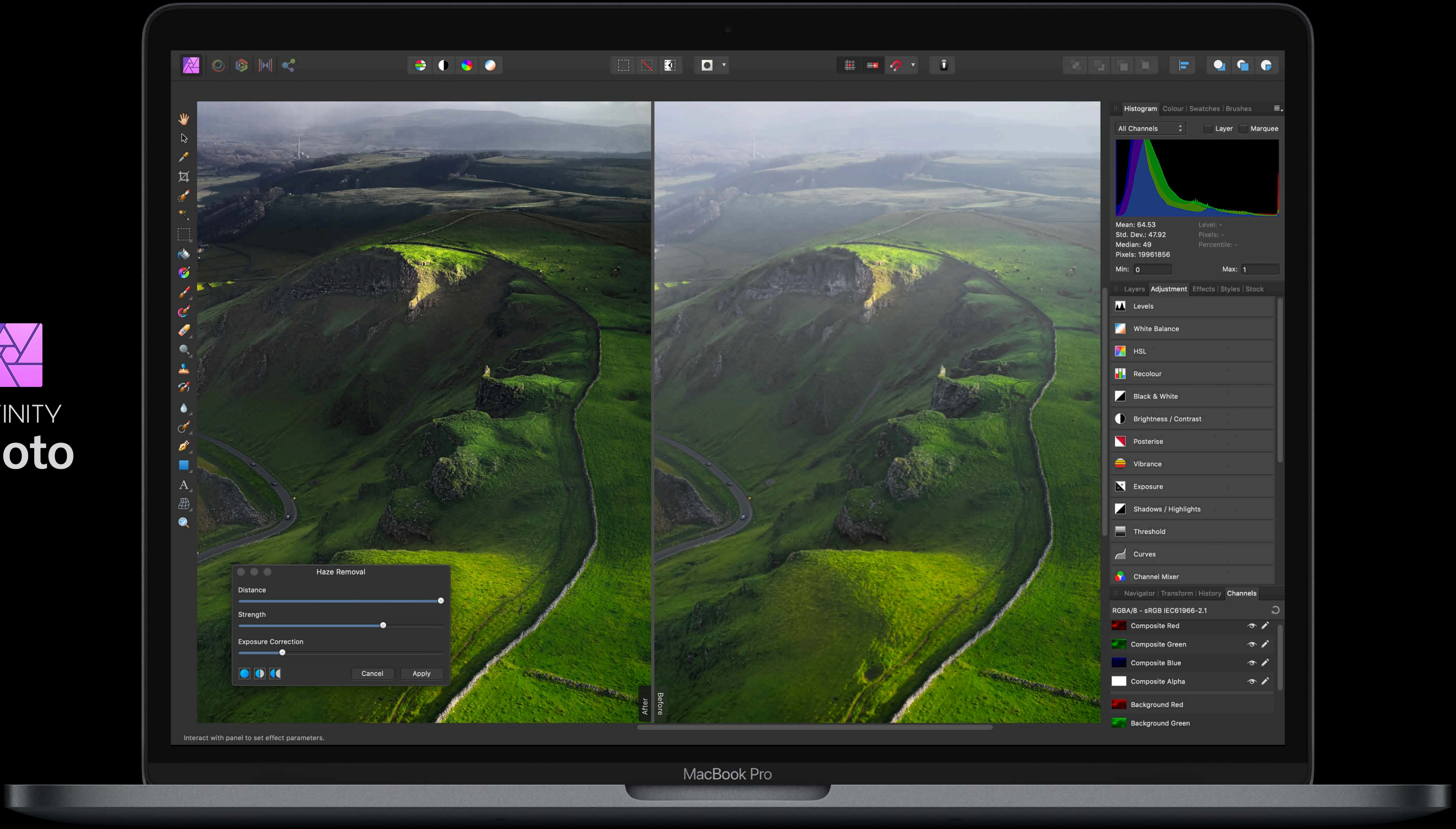
Apple and third party apps

AUTODESK®
MAYA®



01:01:31:03.60
49 1 4 178
127.0000
KEEP
4/4
/16
No In
No Out
Region: MIDI Defaults
Edit
Functions
View
Snap: Smart
Drag: No Overlap
Marker
Chorus
Pre-Verse
Verse
Groups
Groups active
Vocals
Synths
Drums
Guitars
Settings
Track: Morphing Lead
Big Room Kit
Morphing Lead
Synth Hook
Massive Bass
Synth Pad
Riser FX
Heavy Bass
Synth Lead
Crunchy Synth
Punch Bass
String Quartet
Super Saw Pad
Risers & Boomers
Morphing
Hyped Mix
Alchemy
Graphic EQ
Step FX
Chro...aVerb
Compressor
Linear EQ
Exciter
AdLimit
Bus 1
Bus 2
Stereo Out
Read
Bnce
Master Mix
Morphing Lead: Synth Lush
DAMPING EQ
Dense Room
MAIN
DETAILS
Frequency: 2250 Hz
Ratio: 1.30 x
Q: 1.00
Attack
0 %
Size
100 %
Density
100 %
Decay
5.60 s
Distance
55 %
Dry
100 %
Wet
70 %
Predelay
8 ms
Freeze
ChromaVerb

AFFINITY
Photo



Blackmagicdesign

Pro App Platforms

Pro Apps Requirements

Operate on large assets and data

Heavy compute demand

Real-time for creativity vs. offline for full fidelity

Optimizing for 8K video editing

Support for high dynamic range

Leveraging all platform resources

Efficient data transfers

Optimizing for 8K video editing

Support for high dynamic range

Leveraging all platform resources

Efficient data transfers

Before


Master
Master
8KProRes4444_01...
Timeline 1
Smart Bins
Keywords
30%
Timeline 1
01:00:35:05
01:00:35:06
Color Wheels
Primaries Wheels
Lift
Gamma
Gain
Offset
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
1.00 1.00 1.00 1.00
25.00 25.00 25.00
Curves
Custom
Edit
100
100
100
100
Soft Clip
Low
Low Soft
High
High Soft
Keyframes
00:00:35:05
Master
Corrector 1
Sizing

Before


Master
Master
8KProRes4444_01...
Timeline 1
Smart Bins
Keywords
30%
Timeline 1
01:00:35:05
01:00:35:06
Color Wheels
Primaries Wheels
Lift
Gamma
Gain
Offset
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
1.00 1.00 1.00 1.00
25.00 25.00 25.00
Curves
Custom
Edit
100
100
100
100
Soft Clip
Low
Low Soft
High
High Soft
Keyframes
00:00:35:05
Master
Corrector 1
Sizing

Current 8K Proxy Workflow

Current 8K Proxy Workflow

Raw camera footage

Current 8K Proxy Workflow

Raw camera footage

Import

Transcode

Current 8K Proxy Workflow

Current 8K Proxy Workflow

Current 8K Proxy Workflow

Goal — Realtime Edit of 8K Content

Raw camera footage

Import

Video editing app

Export

8K

Optimizing for 8K Video Editing

Video editing pipeline

Managing large asset sizes

Maintaining a predictable frame rate

Optimizing for 8K Video Editing

Video editing pipeline

Managing large asset sizes

Maintaining a predictable frame rate

Video Editing Pipeline

Video Editing Pipeline

Video Editing Pipeline

Video Editing Pipeline

Decode with VideoToolbox

Multiple decoders available via VTDecompressionSession

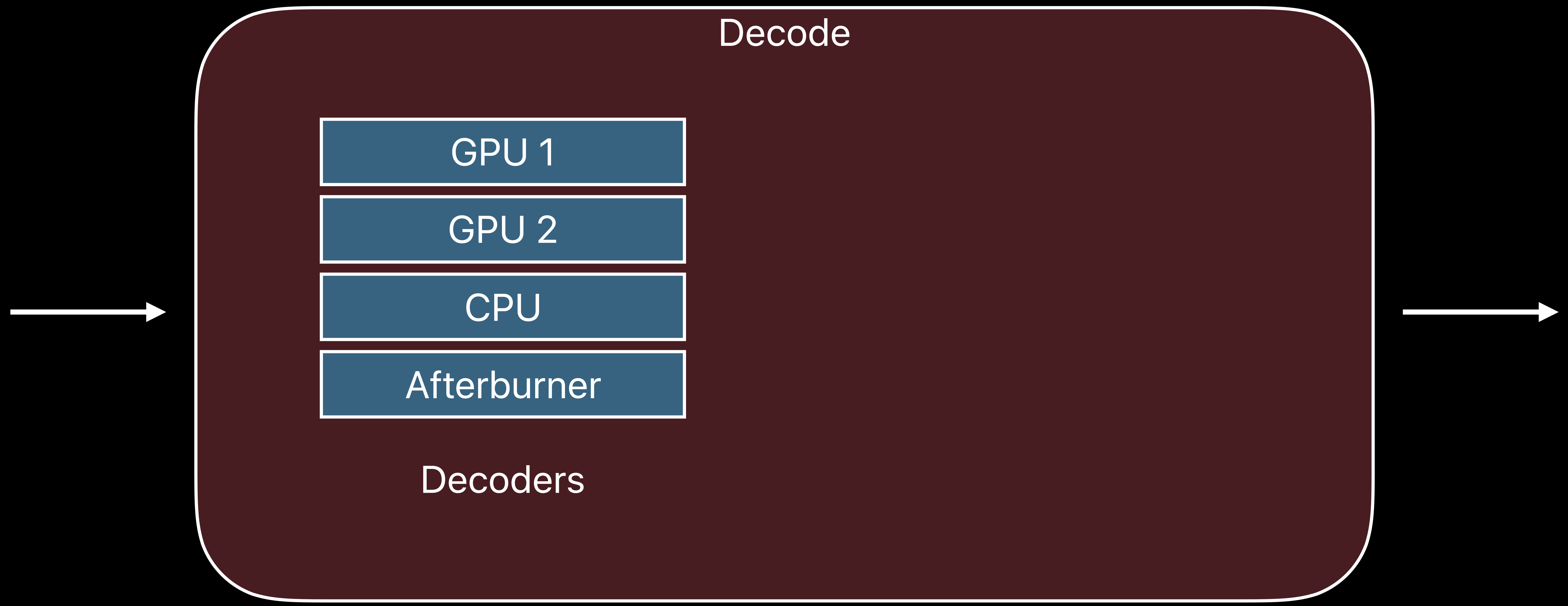
Afterburner card support

Video Editing Pipeline

Decode with VideoToolbox

Multiple decoders available via VTDecompressionSession

Afterburner card support



```
CFDictionarySetValue(decoderSpec,
    kVTVideoDecoderSpecification_EnableHardwareAcceleratedVideoDecoder,
    kCFBooleanTrue);

// This is how you set-up VTDecompressionSession
VTDecompressionSessionCreate(...,
    videoFormatDescription, // Input format
    decoderSpec,
    destinationImageBufferAttributes,
    ^(...) {...}, // didDecompress call for a single decompressed frame
    &session);
…
VTDecodeFrameFlags decodeFlags = kVTDecodeFrame_EnableAsynchronousDecompression;
VTDecompressionSessionDecodeFrame(session,..., decodeFlags, ...);
…
VTDecompressionSessionInvalidate(session);
```

```
CFDictionarySetValue(decoderSpec,
    kVTVideoDecoderSpecification_EnableHardwareAcceleratedVideoDecoder,
    kCFBooleanTrue);
```

```
// This is how you set-up VTDecompressionSession
VTDecompressionSessionCreate(...,
    videoFormatDescription, // Input format
    decoderSpec,
    destinationImageBufferAttributes,
    ^(...) {...}, // didDecompress call for a single decompressed frame
    &session);
…
VTDecodeFrameFlags decodeFlags = kVTDecodeFrame_EnableAsynchronousDecompression;
VTDecompressionSessionDecodeFrame(session,..., decodeFlags, ...);
…
VTDecompressionSessionInvalidate(session);
```

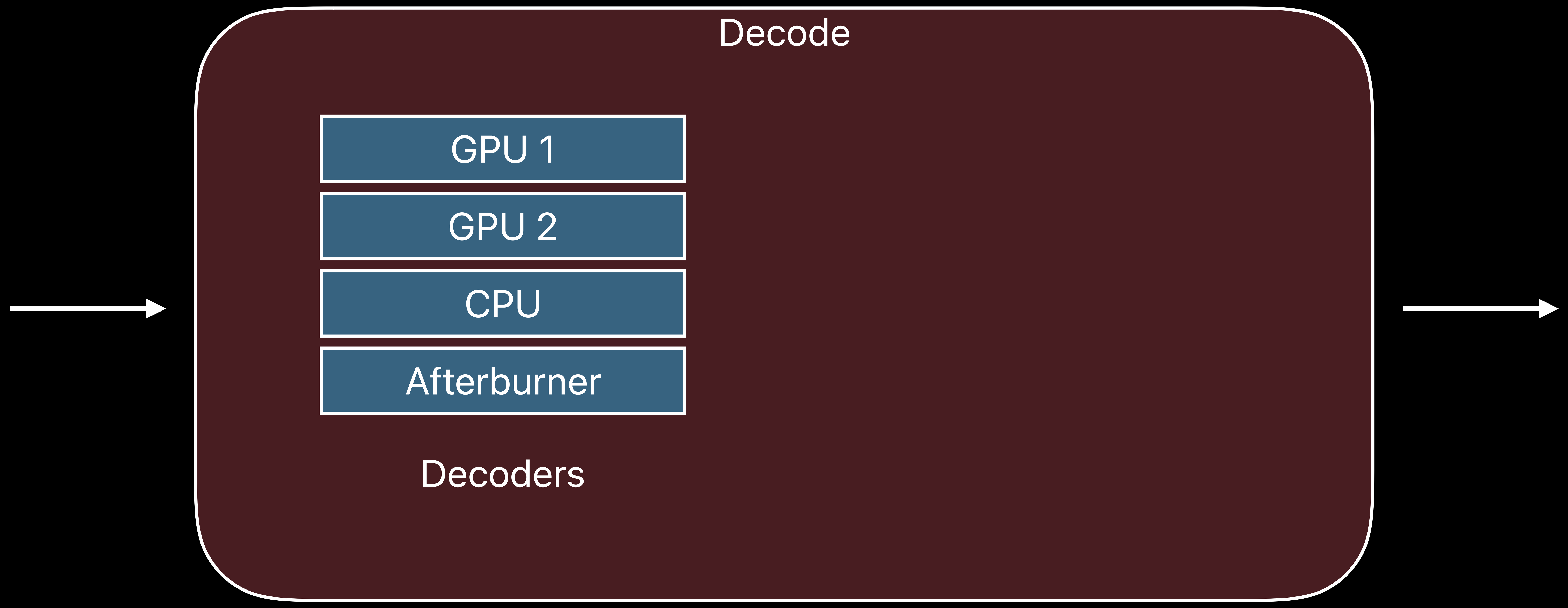
```
CFDictionarySetValue(decoderSpec,
    kVTVideoDecoderSpecification_EnableHardwareAcceleratedVideoDecoder,
    kCFBooleanTrue);

// This is how you set-up VTDecompressionSession
VTDecompressionSessionCreate(...,
    videoFormatDescription, // Input format
    decoderSpec,
    destinationImageBufferAttributes,
    ^(...) {...}, // didDecompress call for a single decompressed frame
    &session);
…
VTDecodeFrameFlags decodeFlags = kVTDecodeFrame_EnableAsynchronousDecompression;
VTDecompressionSessionDecodeFrame(session,..., decodeFlags, ...);
…
VTDecompressionSessionInvalidate(session);
```

```
CFDictionarySetValue(decoderSpec,
    kVTVideoDecoderSpecification_EnableHardwareAcceleratedVideoDecoder,
    kCFBooleanTrue);

// This is how you set-up VTDecompressionSession
VTDecompressionSessionCreate(...,
    videoFormatDescription, // Input format
    decoderSpec,
    destinationImageBufferAttributes,
    ^(...) {...}, // didDecompress call for a single decompressed frame
    &session);
…
VTDecodeFrameFlags decodeFlags = kVTDecodeFrame_EnableAsynchronousDecompression;
VTDecompressionSessionDecodeFrame(session,..., decodeFlags, ...);
…
VTDecompressionSessionInvalidate(session);
```

```
CFDictionarySetValue(decoderSpec,
    kVTVideoDecoderSpecification_EnableHardwareAcceleratedVideoDecoder,
    kCFBooleanTrue);

// This is how you set-up VTDecompressionSession
VTDecompressionSessionCreate(...,
    videoFormatDescription, // Input format
    decoderSpec,
    destinationImageBufferAttributes,
    ^(...) {...}, // didDecompress call for a single decompressed frame
    &session);
…
VTDecodeFrameFlags decodeFlags = kVTDecodeFrame_EnableAsynchronousDecompression;
VTDecompressionSessionDecodeFrame(session,..., decodeFlags, ...);
…
VTDecompressionSessionInvalidate(session);
```

Video Editing Pipeline

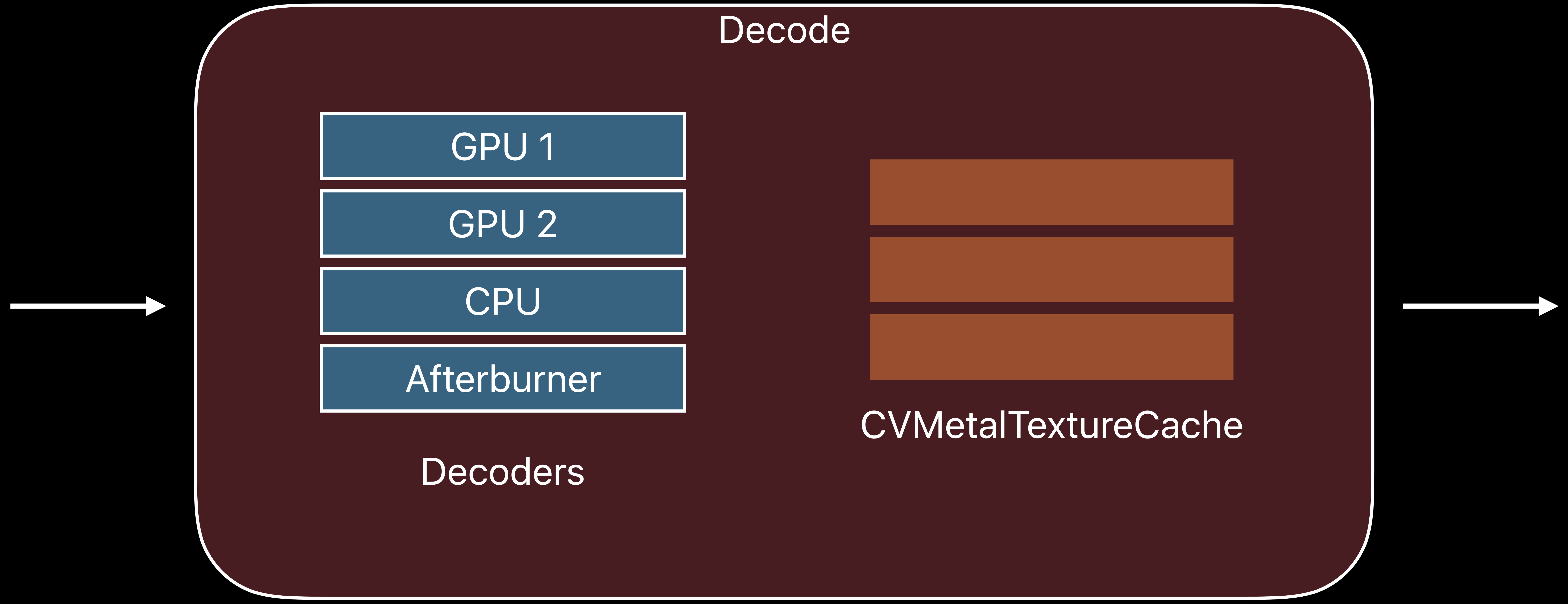
Metal with Core Video



Video Editing Pipeline
Metal with Core Video

Leverage IOSurface as a backing store

Use CVMetalTextureCache to directly access decoded frames



```
CVMetalTextureCacheRef sessionMetalCache;
CVMetalTextureCacheCreate(..., metalDevice, ..., &sessionMetalCache);

// On a new decoded pixelBuffer callback
CVMetalTextureRef textureOut; // Keep this around until rendering is done
CVMetalTextureCacheCreateTextureFromImage(...,
    sessionMetalCache,
    pixelBuffer,
    metalFormat,
    CVPixelBufferGetWidthOfPlane(pixelBuffer, 0),
    CVPixelBufferGetHeightOfPlane(pixelBuffer, 0),
    0, &textureOut);
id<MTLTexture> texture = CVMetalTextureGetTexture(textureOut);
…
CFRelease(textureOut); // Release when Metal processing is done
…
CVBufferRelease(pixelBuffer); // Crucial to do this to keep CVPixelBuffer recycled
```

```
CVMetalTextureCacheRef sessionMetalCache;
CVMetalTextureCacheCreate(..., metalDevice, ..., &sessionMetalCache);

// On a new decoded pixelBuffer callback
CVMetalTextureRef textureOut; // Keep this around until rendering is done
CVMetalTextureCacheCreateTextureFromImage(...,
    sessionMetalCache,
    pixelBuffer,
    metalFormat,
    CVPixelBufferGetWidthOfPlane(pixelBuffer, 0),
    CVPixelBufferGetHeightOfPlane(pixelBuffer, 0),
    0, &textureOut);
id<MTLTexture> texture = CVMetalTextureGetTexture(textureOut);
…
CFRelease(textureOut); // Release when Metal processing is done
…
CVBufferRelease(pixelBuffer); // Crucial to do this to keep CVPixelBuffer recycled
```

```
CVMetalTextureCacheRef sessionMetalCache;
CVMetalTextureCacheCreate(..., metalDevice, ..., &sessionMetalCache);

// On a new decoded pixelBuffer callback
CVMetalTextureRef textureOut; // Keep this around until rendering is done
CVMetalTextureCacheCreateTextureFromImage(...,
    sessionMetalCache,
    pixelBuffer,
    metalFormat,
    CVPixelBufferGetWidthOfPlane(pixelBuffer, 0),
    CVPixelBufferGetHeightOfPlane(pixelBuffer, 0),
    0, &textureOut);
id<MTLTexture> texture = CVMetalTextureGetTexture(textureOut);
…
CFRelease(textureOut); // Release when Metal processing is done
…
CVBufferRelease(pixelBuffer); // Crucial to do this to keep CVPixelBuffer recycled
```

```
CVMetalTextureCacheRef sessionMetalCache;
CVMetalTextureCacheCreate(..., metalDevice, ..., &sessionMetalCache);

// On a new decoded pixelBuffer callback
CVMetalTextureRef textureOut; // Keep this around until rendering is done
CVMetalTextureCacheCreateTextureFromImage(...,
    sessionMetalCache,
    pixelBuffer,
    metalFormat,
    CVPixelBufferGetWidthOfPlane(pixelBuffer, 0),
    CVPixelBufferGetHeightOfPlane(pixelBuffer, 0),
    0, &textureOut);
id<MTLTexture> texture = CVMetalTextureGetTexture(textureOut);
…
CFRelease(textureOut); // Release when Metal processing is done
…
CVBufferRelease(pixelBuffer); // Crucial to do this to keep CVPixelBuffer recycled
```

Video Editing Pipeline

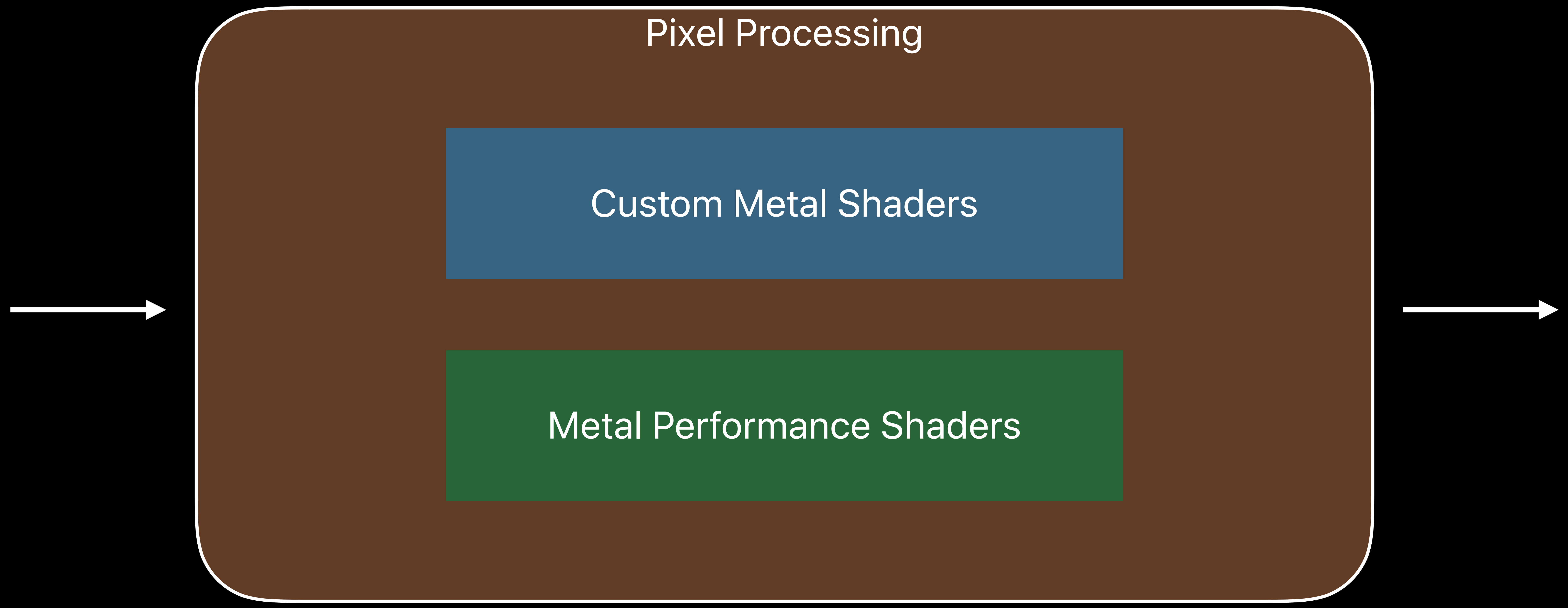
Video Editing Pipeline

Video Editing Pipeline

Pixel processing with Metal

Implement custom shaders for filters and effects

MPS provides optimized implementations for many common filters



```swift
func myBlurTextureInPlace(inTexture: MTLTexture, blurRadius: Float, queue: MTLCommandQueue)
{
    // Create the usual Metal objects
    let device = queue.device
    let buffer = queue.makeCommandBuffer()

    // Create a MPS filter
    let blur = MPSImageGaussianBlur(device, blurRadius)

    // Attempt to do the work in place
    let inPlaceTexture = UnsafeMutablePointer<MTLTexture>.allocate(capacity: 1)
    inPlaceTexture.initialize(inTexture)

    blur.encode(buffer, inPlaceTexture, myAllocator)

    // The usual Metal enqueue process
    buffer.commit()
}
```

```swift
func myBlurTextureInPlace(inTexture: MTLTexture, blurRadius: Float, queue: MTLCommandQueue)
{
    // Create the usual Metal objects
    let device = queue.device
    let buffer = queue.makeCommandBuffer()

    // Create a MPS filter
    let blur = MPSImageGaussianBlur(device, blurRadius)

    // Attempt to do the work in place
    let inPlaceTexture = UnsafeMutablePointer<MTLTexture>.allocate(capacity: 1)
    inPlaceTexture.initialize(inTexture)

    blur.encode(buffer, inPlaceTexture, myAllocator)

    // The usual Metal enqueue process
    buffer.commit()
}
```

```swift
func myBlurTextureInPlace(inTexture: MTLTexture, blurRadius: Float, queue: MTLCommandQueue)
{
    // Create the usual Metal objects
    let device = queue.device
    let buffer = queue.makeCommandBuffer()

    // Create a MPS filter
    let blur = MPSImageGaussianBlur(device, blurRadius)

    // Attempt to do the work in place
    let inPlaceTexture = UnsafeMutablePointer<MTLTexture>.allocate(capacity: 1)
    inPlaceTexture.initialize(inTexture)

    blur.encode(buffer, inPlaceTexture, myAllocator)

    // The usual Metal enqueue process
    buffer.commit()
}
```

```swift
func myBlurTextureInPlace(inTexture: MTLTexture, blurRadius: Float, queue: MTLCommandQueue)
{
    // Create the usual Metal objects
    let device = queue.device
    let buffer = queue.makeCommandBuffer()

    // Create a MPS filter
    let blur = MPSImageGaussianBlur(device, blurRadius)

    // Attempt to do the work in place
    let inPlaceTexture = UnsafeMutablePointer<MTLTexture>.allocate(capacity: 1)
    inPlaceTexture.initialize(inTexture)

    blur.encode(buffer, inPlaceTexture, myAllocator)

    // The usual Metal enqueue process
    buffer.commit()
}
```

```swift
func myBlurTextureInPlace(inTexture: MTLTexture, blurRadius: Float, queue: MTLCommandQueue)
{
    // Create the usual Metal objects
    let device = queue.device
    let buffer = queue.makeCommandBuffer()

    // Create a MPS filter
    let blur = MPSImageGaussianBlur(device, blurRadius)

    // Attempt to do the work in place
    let inPlaceTexture = UnsafeMutablePointer<MTLTexture>.allocate(capacity: 1)
    inPlaceTexture.initialize(inTexture)

    blur.encode(buffer, inPlaceTexture, myAllocator)

    // The usual Metal enqueue process
    buffer.commit()
}
```

Video Editing Pipeline

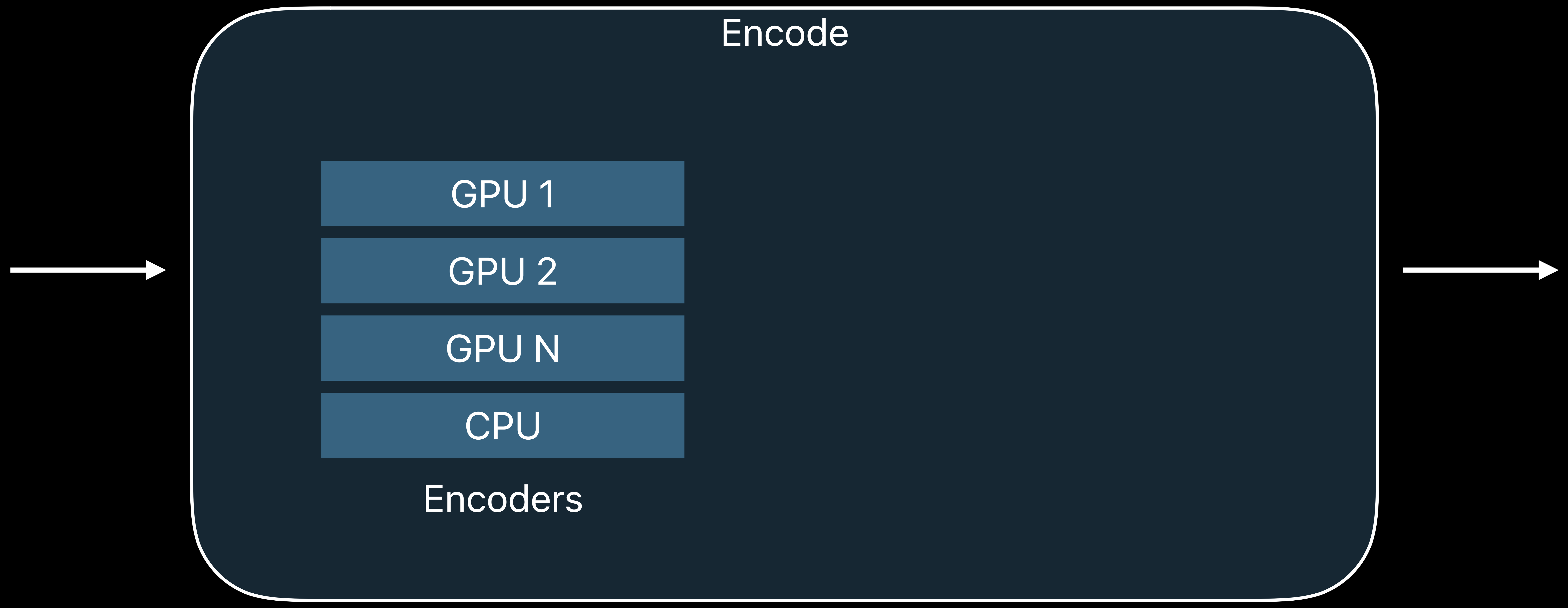
Video Editing Pipeline

Video Editing Pipeline

Encode with VideoToolbox

Similar to decode but using VTCompressionSession APIs

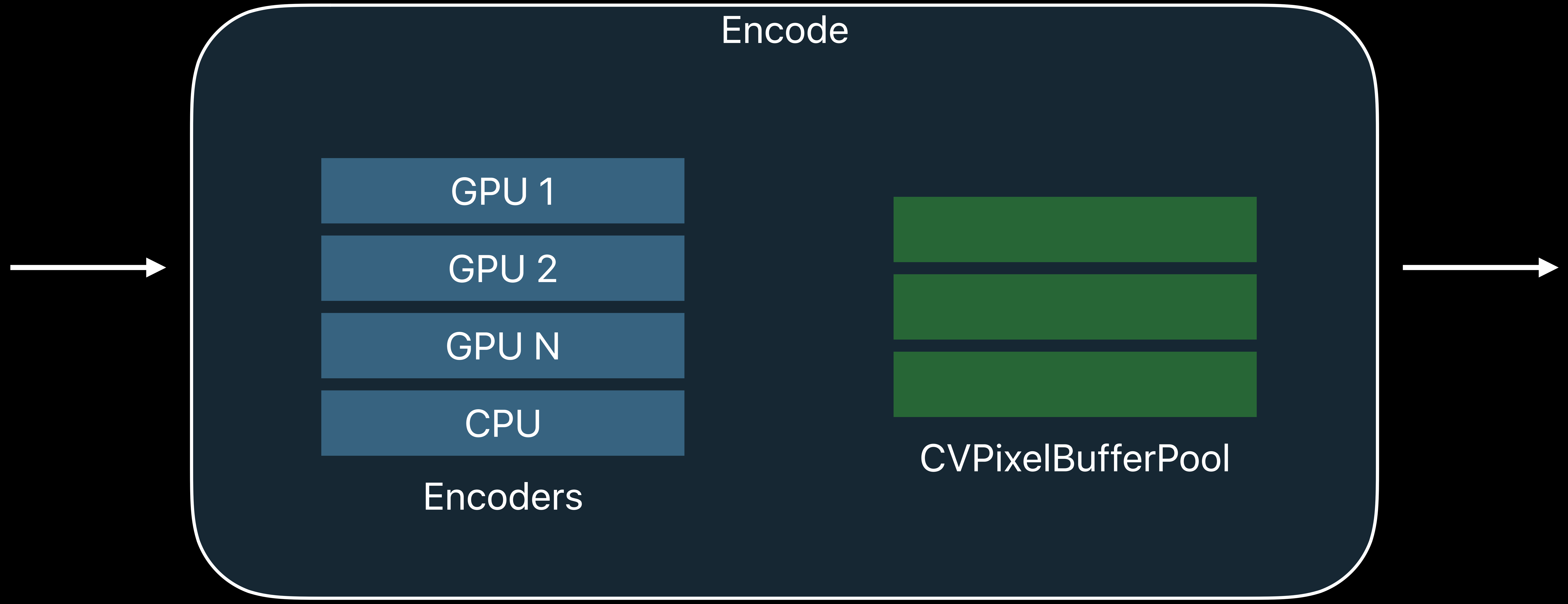
CVPixelBufferPool to get Metal textures



Video Editing Pipeline
Encode with VideoToolbox

Similar to decode but using VTCompressionSession APIs

CVPixelBufferPool to get Metal textures



```
VTCopyVideoEncoderList(…); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);
…
CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(…, pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);
…
CVBufferRelease(buffer); // Crucial to keep it recycled
```

```
VTCopyVideoEncoderList(…); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);
…
CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(…, pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);
…
CVBufferRelease(buffer); // Crucial to keep it recycled
```

```
VTCopyVideoEncoderList(…); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);
…
CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(…, pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);
…
CVBufferRelease(buffer); // Crucial to keep it recycled
```

```
VTCopyVideoEncoderList(…); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);
…
CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(…, pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);
…
CVBufferRelease(buffer); // Crucial to keep it recycled
```

```
VTCopyVideoEncoderList(…); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);
…
CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(…, pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);
…
CVBufferRelease(buffer); // Crucial to keep it recycled
```

```
VTCopyVideoEncoderList(…); // This is how you get the list of all available encoders

CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_EnableHardwareAcceleratedVideoEncoder,
    kCFBooleanTrue);
CFDictionarySetValue(encoderSpec,
    kVTVideoEncoderSpecification_PreferredEncoderGPURegistryID,
    requiredGPU);

CVPixelBufferPoolRef pixelBufferPool; // Pool to precisely match the format
pixelBufferPool = VTCompressionSessionGetPixelBufferPool(session);
…
CVPixelBufferRef buffer;
CVPixelBufferPoolCreatePixelBuffer(…, pixelBufferPool, &buffer);
CVMetalTextureCacheCreateTextureFromImage(...);
…
CVBufferRelease(buffer); // Crucial to keep it recycled
```

Video Editing Pipeline

Video Editing Pipeline

Optimizing for 8K Video Editing

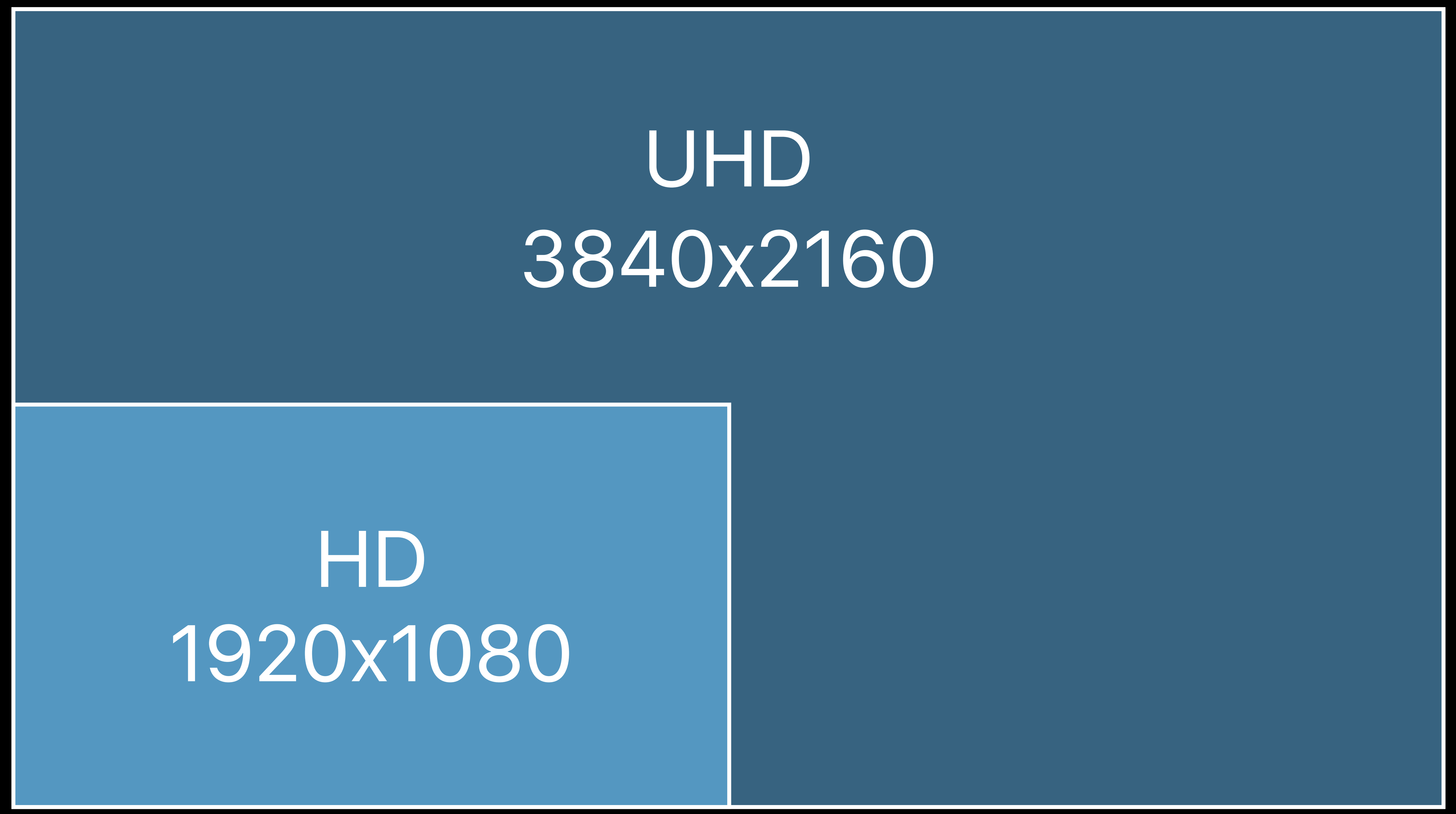
Video editing pipeline

Managing large asset sizes

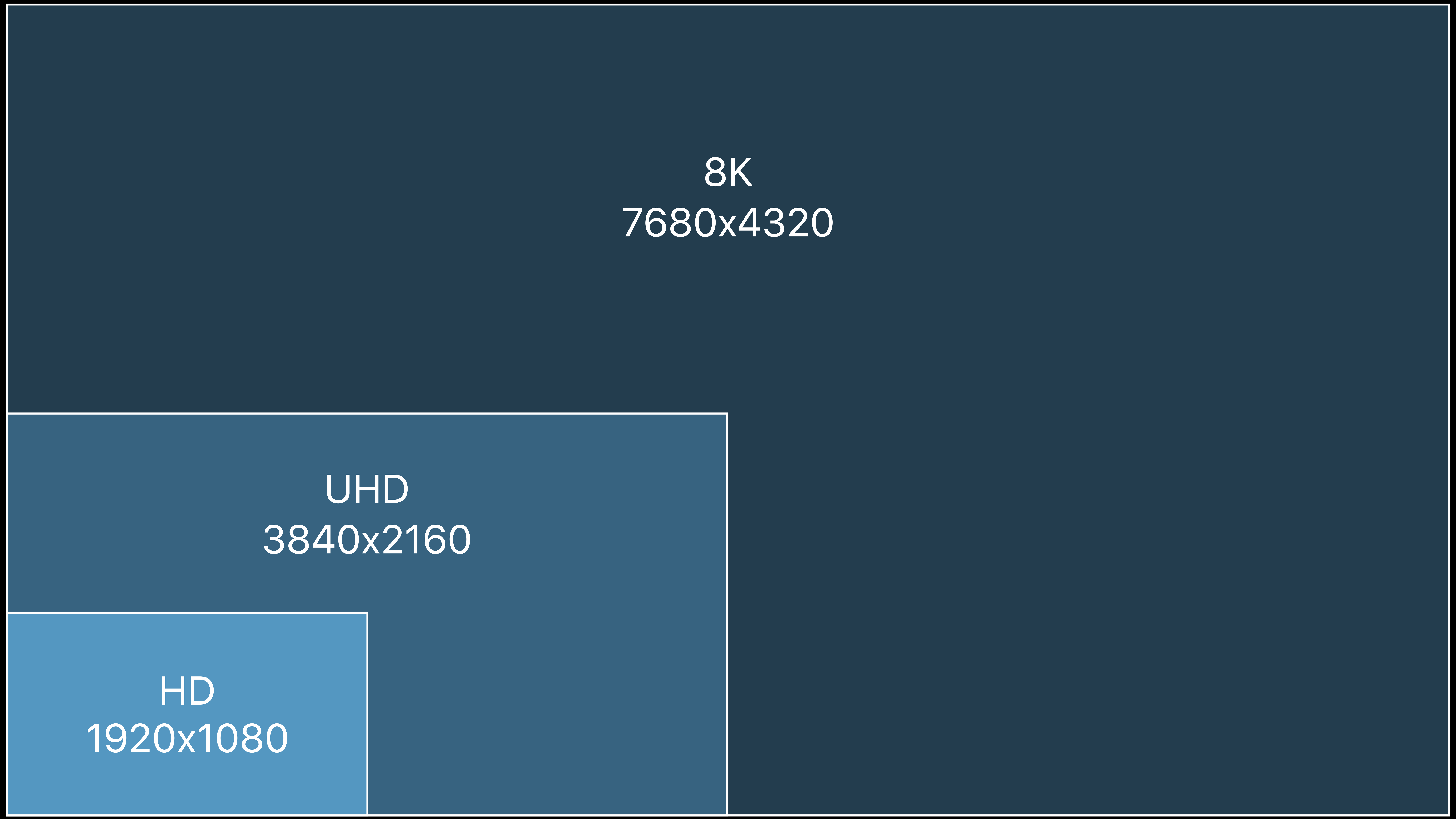
Maintaining a predictable frame rate

16x the Memory of HD Content

16x the Memory of HD Content



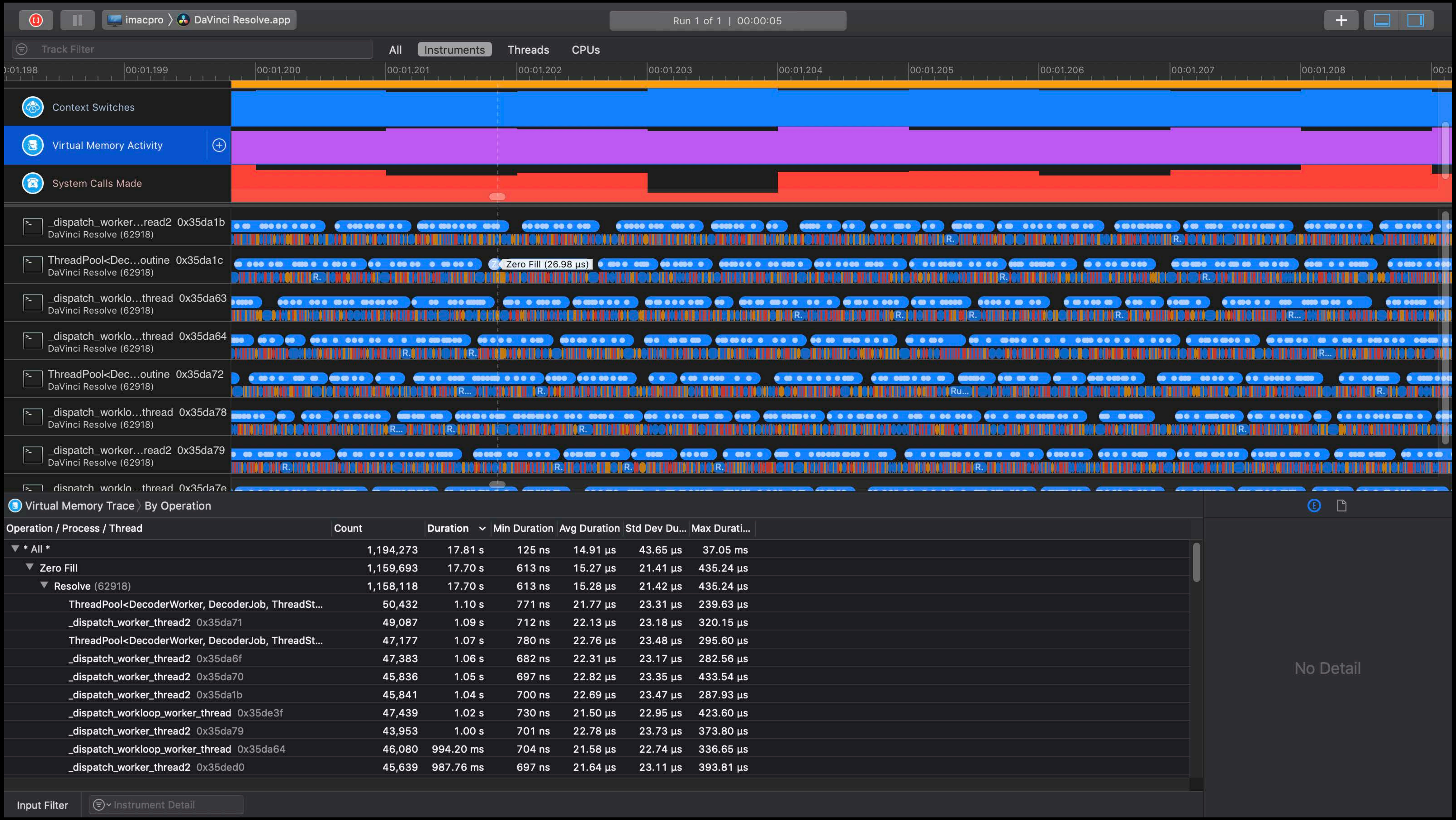
16x the Memory of HD Content



1TB

10 minute clip @ 24Hz

Virtual Memory Residency



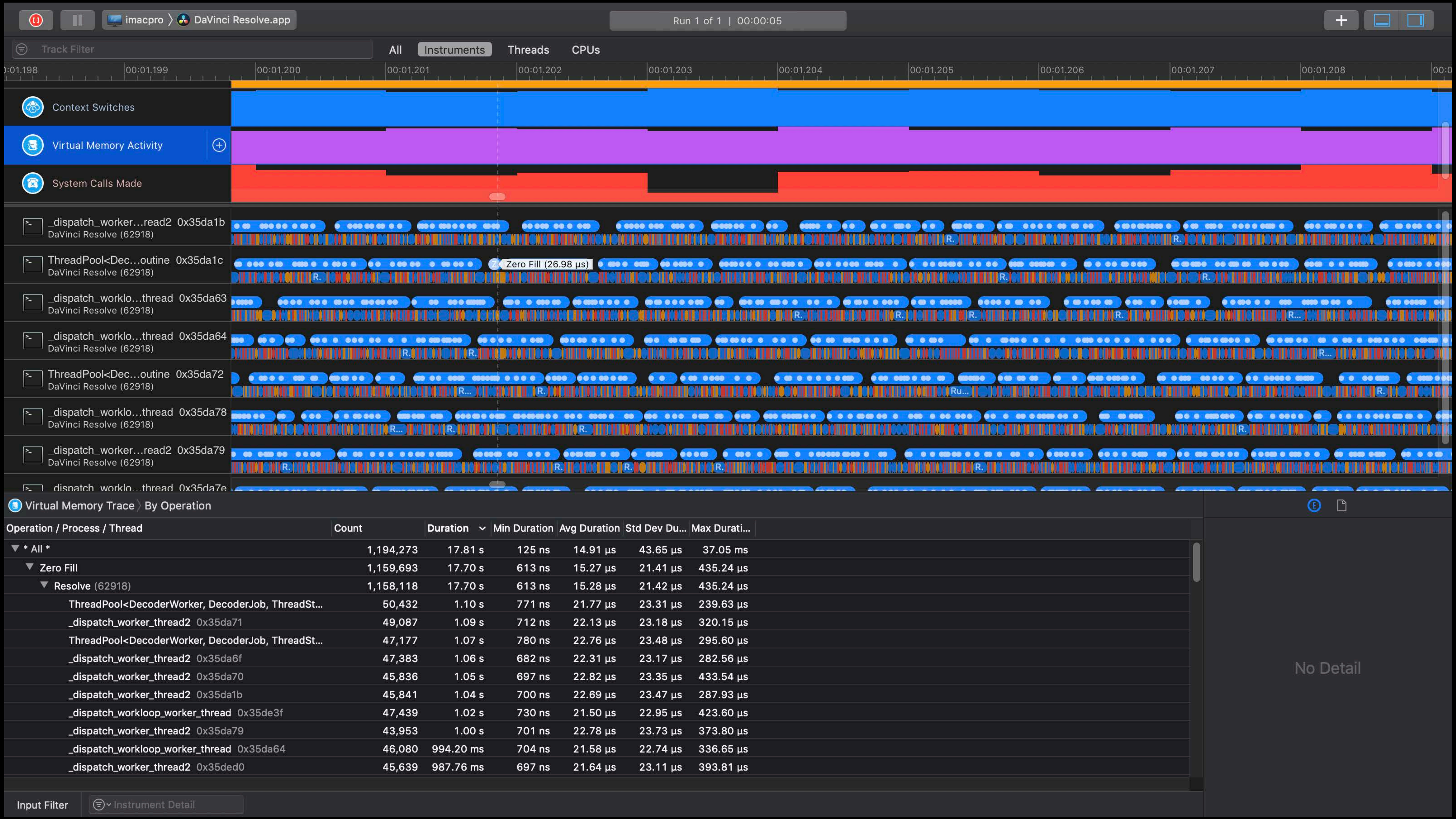
Virtual Memory Residency

Virtual Memory Residency

Operation / Process / Thread	Count	Duration	Min Duration	Avg Duration	Std Dev Du...	Max Durati...
* All *	1,194,273	17.81 s	125 ns	14.91 µs	43.65 µs	37.05 ms
Zero Fill	1,159,693	17.70 s	613 ns	15.27 µs	21.41 µs	435.24 µs
Resolve (62918)	1,158,118	17.70 s	613 ns	15.28 µs	21.42 µs	435.24 µs
ThreadPool<DecoderWorker, DecoderJob, ThreadSt...	50,432	1.10 s	771 ns	21.77 µs	23.31 µs	239.63 µs
_dispatch_worker_thread2 0x35da71	49,087	1.09 s	712 ns	22.13 µs	23.18 µs	320.15 µs
ThreadPool<DecoderWorker, DecoderJob, ThreadSt...	47,177	1.07 s	780 ns	22.76 µs	23.48 µs	295.60 µs
_dispatch_worker_thread2 0x35da6f	47,383	1.06 s	682 ns	22.31 µs	23.17 µs	282.56 µs
_dispatch_worker_thread2 0x35da70	45,836	1.05 s	697 ns	22.82 µs	23.35 µs	433.54 µs
_dispatch_worker_thread2 0x35da1b	45,841	1.04 s	700 ns	22.69 µs	23.47 µs	287.93 µs
_dispatch_workloop_worker_thread 0x35de3f	47,439	1.02 s	730 ns	21.50 µs	22.95 µs	423.60 µs
_dispatch_worker_thread2 0x35da79	43,953	1.00 s	701 ns	22.78 µs	23.73 µs	373.80 µs
_dispatch_workloop_worker_thread 0x35da64	46,080	994.20 ms	704 ns	21.58 µs	22.74 µs	336.65 µs
_dispatch_worker_thread2 0x35ded0	45,639	987.76 ms	697 ns	21.64 µs	23.11 µs	393.81 µs

Virtual Memory Residency

Virtual Memory Residency



Managing Large Asset Sizes

Virtual memory residency

High VM page fault cost with large 8K allocations

Can impact performance at the start of video playback

Managing Large Asset Sizes

Virtual memory residency

High VM page fault cost with large 8K allocations

Can impact performance at the start of video playback

Pre-warm your buffers before playback starts

Managing Large Asset Sizes

Allocation best practices

Allocate early to minimize allocation cost mid-workflow

Reuse memory by using buffer pools

Use Metal heaps for transient allocations

Managing Transient Allocations

MTLHeap advantages

Allocating from MTLHeap is cheap

Heap is made resident as a whole

MTLHeap uses memory more efficiently

Resources may be aliased and memory may be reused

```
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)
// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)
...
// Tell the heap we don't need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()
...
// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
executeCombinedOutput(output1, output2, intermediateBuffer, output3)
```

```swift
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)
// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)
...
// Tell the heap we don't need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()
...
// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
executeCombinedOutput(output1, output2, intermediateBuffer, output3)
```

```swift
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)
// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)
...
// Tell the heap we don't need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()
...
// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
executeCombinedOutput(output1, output2, intermediateBuffer, output3)
```

```
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)
// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)
...
// Tell the heap we don't need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()
...
// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
executeCombinedOutput(output1, output2, intermediateBuffer, output3)
```

```swift
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)
// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)
...
// Tell the heap we don't need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()
...
// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
executeCombinedOutput(output1, output2, intermediateBuffer, output3)
```

```
let heap = device.newHeapWithDescriptor()

// Allocate uniforms for blur kernel
let blurUniforms = heap.makeBuffer(length1, options1, offset1)
executeBlur(input, output1, blurUniforms)
// Allocate uniforms for color grading
let colorgradeUniforms = heap.makeBuffer(length2, options2, offset2)
executeColorgrade(input, output2, colorgradeUniforms)
...
// Tell the heap we don't need those from this point onwards
blurUniforms.makeAliasable()
colorgradeUniforms.makeAliasable()
...
// Allocate an intermediate buffer for combining the result (reuse the memory)
let intermediateBuffer = heap.makeBuffer(length3, options3, offset3)
executeCombinedOutput(output1, output2, intermediateBuffer, output3)
```

Optimizing for 8K Video Editing

Video editing pipeline

Managing large asset sizes

Maintaining a predictable frame rate

Uneven Frame Pacing

Uneven Frame Pacing

Uneven Frame Pacing

Uneven Frame Pacing

Uneven Frame Pacing

Uneven Frame Pacing

Predictable Frame Pacing with CVDisplayLink

```
// Set-up CVDisplayLink
CVDisplayLinkRef displayLink;
CVDisplayLinkCreateWithCGDisplay(display, &displayLink);

// Callback to control the cadence
__block int lastFrameIdx = -1;
CVDisplayLinkSetOutputHandler(displayLink, ^(…, inNow, inOutput, …)
{
   int outFrameIdx = floor(inOutputSec * frameDesiredFrequency);
   if (outFrameIdx > lastFrameIdx && !presentQueue.empty())
   {
      lastFrameIdx = outFrameIdx;
      id<MTLDrawable> toShow = presentQueue.pop();
      present(toShow);
   }
   return kCVReturnSuccess;
});
```

```
// Set-up CVDisplayLink
CVDisplayLinkRef displayLink;
CVDisplayLinkCreateWithCGDisplay(display, &displayLink);

// Callback to control the cadence
__block int lastFrameIdx = -1;
CVDisplayLinkSetOutputHandler(displayLink, ^(…, inNow, inOutput, …)
{
   int outFrameIdx = floor(inOutputSec * frameDesiredFrequency);
   if (outFrameIdx > lastFrameIdx && !presentQueue.empty())
   {
      lastFrameIdx = outFrameIdx;
      id<MTLDrawable> toShow = presentQueue.pop();
      present(toShow);
   }
   return kCVReturnSuccess;
});
```

```
// Set-up CVDisplayLink
CVDisplayLinkRef displayLink;
CVDisplayLinkCreateWithCGDisplay(display, &displayLink);

// Callback to control the cadence
__block int lastFrameIdx = -1;
CVDisplayLinkSetOutputHandler(displayLink, ^(…, inNow, inOutput, …)
{
   int outFrameIdx = floor(inOutputSec * frameDesiredFrequency);
   if (outFrameIdx > lastFrameIdx && !presentQueue.empty())
   {
      lastFrameIdx = outFrameIdx;
      id<MTLDrawable> toShow = presentQueue.pop();
      present(toShow);
   }
   return kCVReturnSuccess;
});
```

Content Rate Not Equal to Display Rate

Pro Display XDR

Matching refresh and content rate

Supports multiple common video playback rates (48Hz, 50Hz)

After


Master
Master
8KProRes4444_01...
Timeline 1
Smart Bins
Keywords
30%
23.976
Timeline 1
00:00:18:02
01:00:18:02
Color Wheels
Primaries Wheels
Lift
Gamma
Gain
Offset
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
1.00 1.00 1.00 1.00
25.00 25.00 25.00
Curves
Custom
Edit
100
100
100
100
Soft Clip
Low
Low Soft
High
High Soft
Keyframes
00:00:18:01
Master
Corrector 1
Sizing

After


Master
30%
23.976
Timeline 1
00:00:18:02
Master
8KProRes4444_01...
Timeline 1
Smart Bins
Keywords
01:00:18:02
Color Wheels
Primaries Wheels
Lift
Gamma
Gain
Offset
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
1.00 1.00 1.00 1.00
25.00 25.00 25.00
Curves
Custom
Edit
100
100
100
100
Soft Clip
Low
Low Soft
High
High Soft
Keyframes
00:00:18:01
Master
Corrector 1
Sizing

"With the recent DaVinci Resolve and macOS updates, we are able to get significant performance improvements for video playback and UI interactivity. This enables workflows like 8K editing, color correction, and finishing on Mac desktops and laptops."

Rohit Gupta, Director of DaVinci Software Engineering, Blackmagic Design

Optimizing for 8K video editing

Support for high dynamic range

Leveraging all platform resources

Efficient data transfers

Support for High Dynamic Range

Dileep Madhava, GPUSW

Common traits of HDR images

Apple's approach to HDR

HDR rendering with Metal

Best practices

Common Traits of HDR Images

Common Traits of HDR Images

Better contrast levels

Common Traits of HDR Images

Better contrast levels

More colors

Common Traits of HDR Images

Better contrast levels

More colors

Increased brightness

Common Traits of HDR Images

Better contrast levels

More colors

Increased brightness

Display that preserves artistic intent of image

Extended Dynamic Range (EDR)

Apple's approach to HDR

Brightness head room utilized for highlights and shadows

Extended Dynamic Range (EDR)

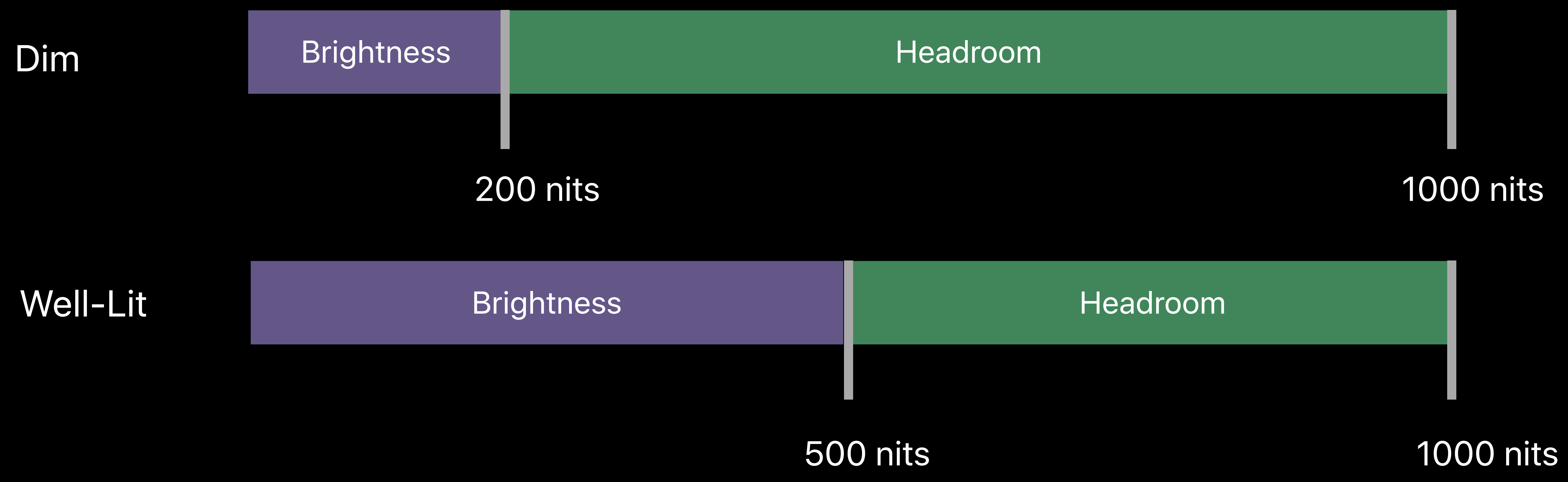
Apple’s approach to HDR

Brightness head room utilized for highlights and shadows

Extended Dynamic Range (EDR)

Apple’s approach to HDR

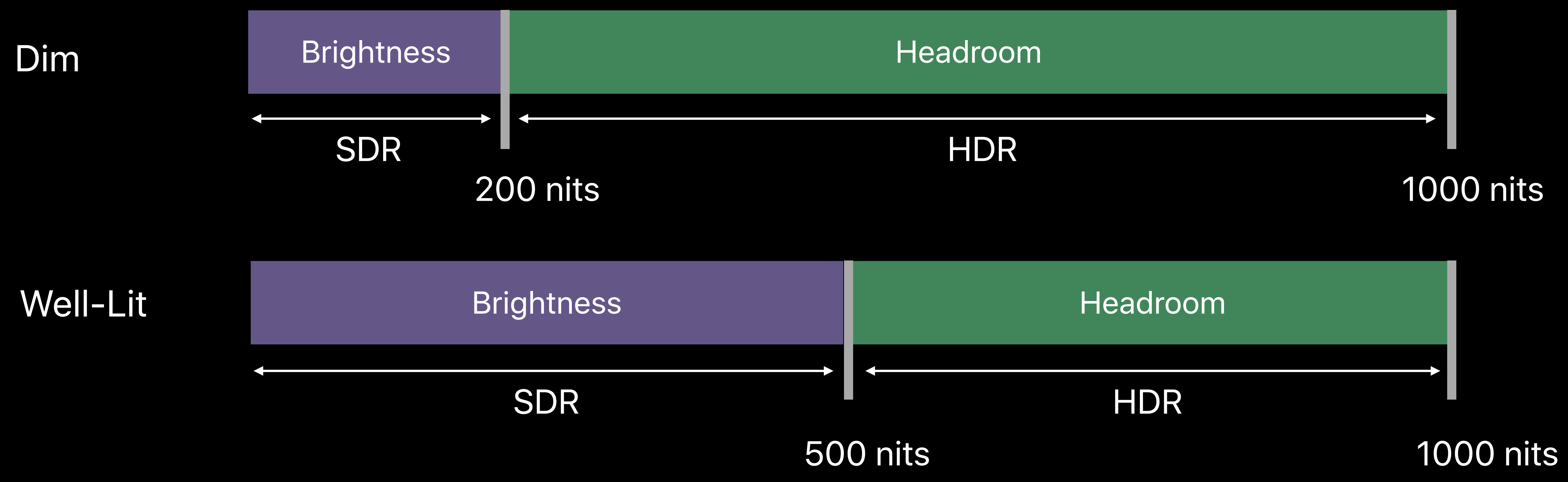
Brightness head room utilized for highlights and shadows



Extended Dynamic Range (EDR)

Apple's approach to HDR

Brightness head room utilized for highlights and shadows



Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness

Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness

Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness

Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness

Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness

Extended Dynamic Range (EDR)

HDR pixels values are scaled relative to SDR Display brightness

Extended Dynamic Range (EDR)

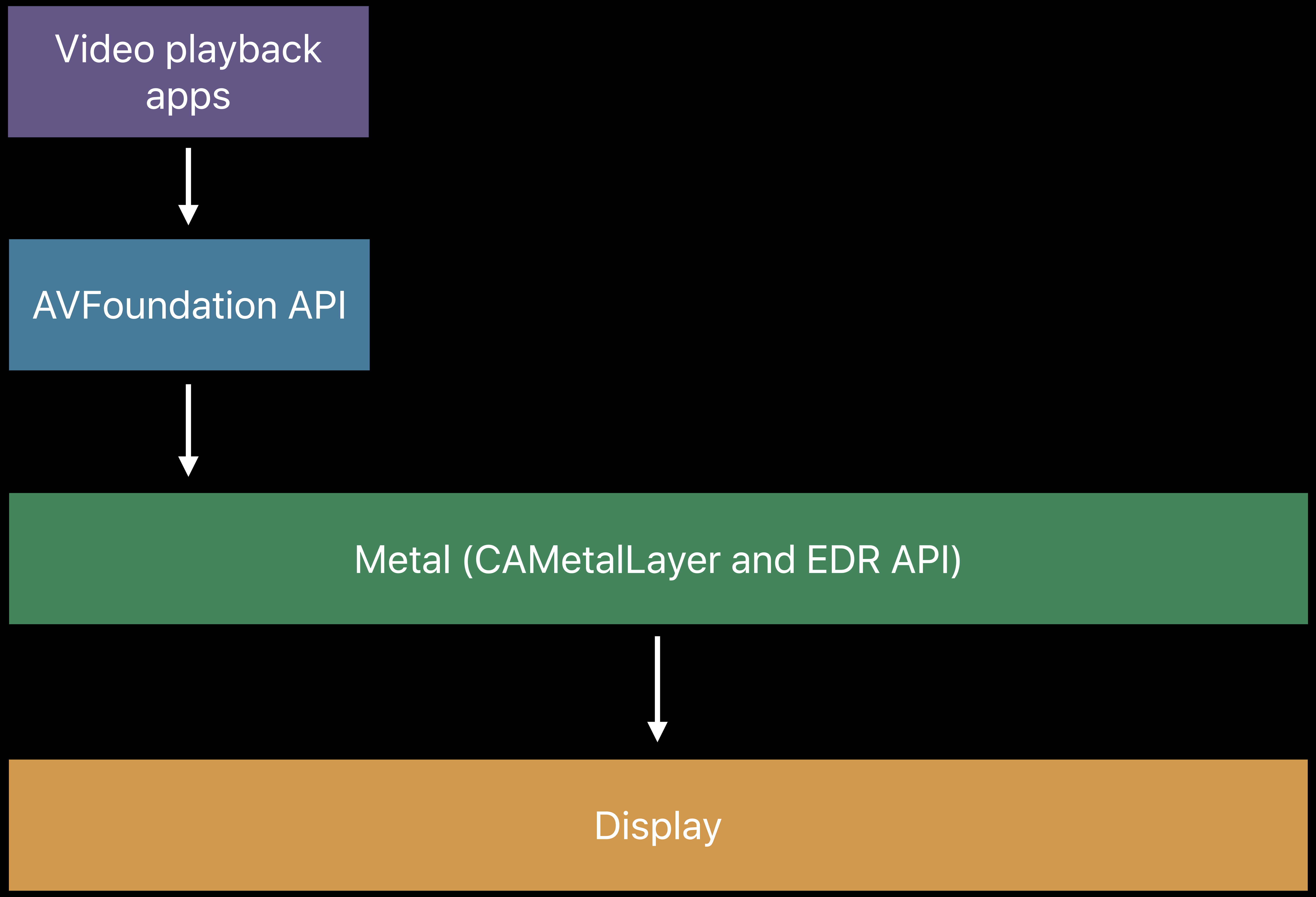
HDR pixels values are scaled relative to SDR Display brightness

Extended Dynamic Range (EDR)

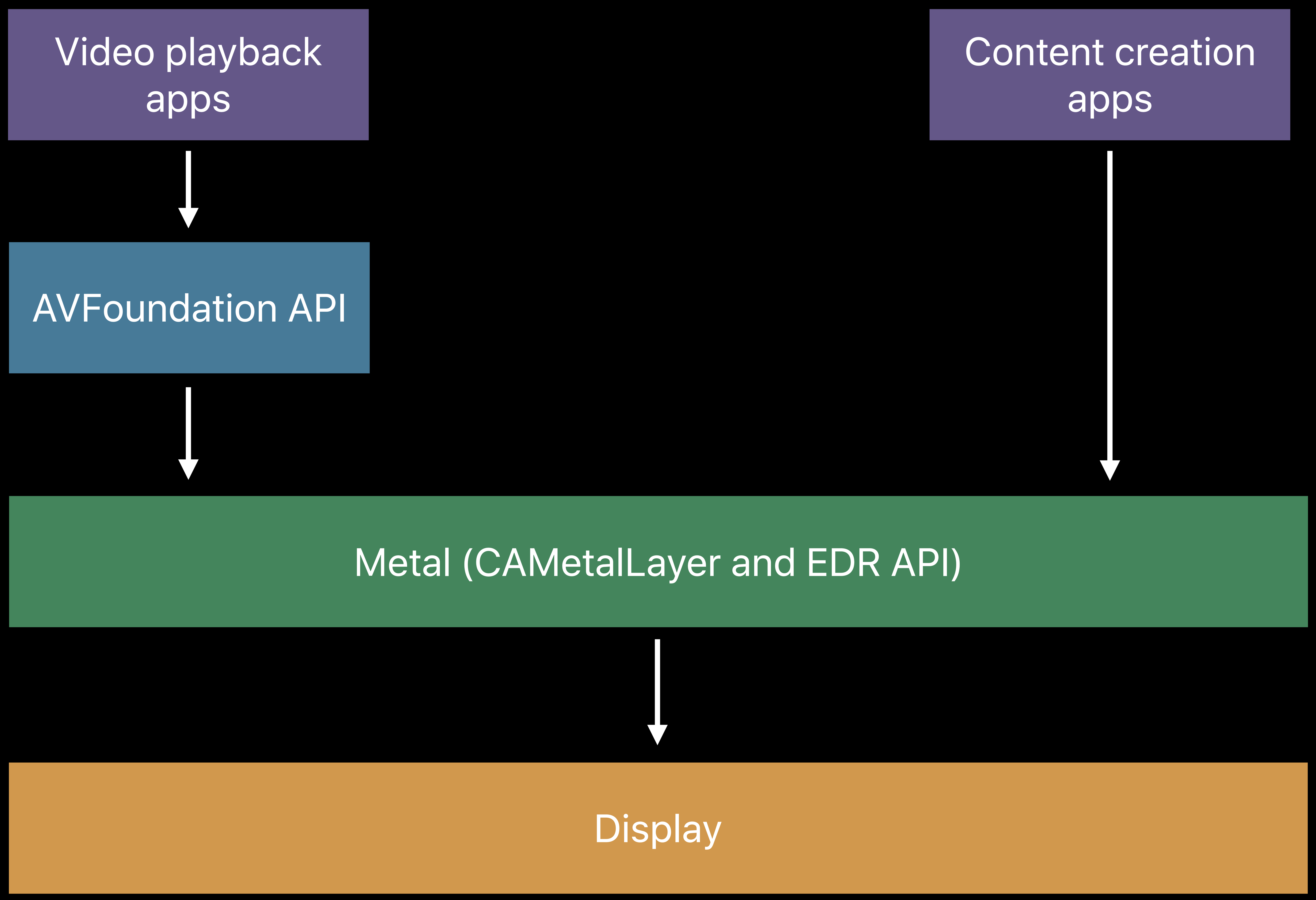
HDR pixels values are scaled relative to SDR Display brightness

HDR Rendering Options

HDR Rendering Options



HDR Rendering Options



```
// Check for EDR support on display
NSScreen * screen =  view.window.screen;
CGFloat edrSupport =  screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
…
```

```objc
// Check for EDR support on display
NSScreen * screen =  view.window.screen;
CGFloat edrSupport =  screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
…
```

```
// Check for EDR support on display
NSScreen * screen =  view.window.screen;
CGFloat edrSupport =  screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
…
```

```
// Check for EDR support on display
NSScreen * screen =  view.window.screen;
CGFloat edrSupport =  screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.col
CGColorSpaceRe

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
…
```

kCGColorSpaceDisplayBT2020

```
// Check for EDR support on display
NSScreen * screen =  view.window.screen;
CGFloat edrSupport =  screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
…
```

PQ_EOTF

```
// Check for EDR support on display
NSScreen * screen =  view.window.screen;
CGFloat edrSupport =  screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
…
```

```objc
// Check for EDR support on display
NSScreen * screen =  view.window.screen;
CGFloat edrSupport =  screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
…
```

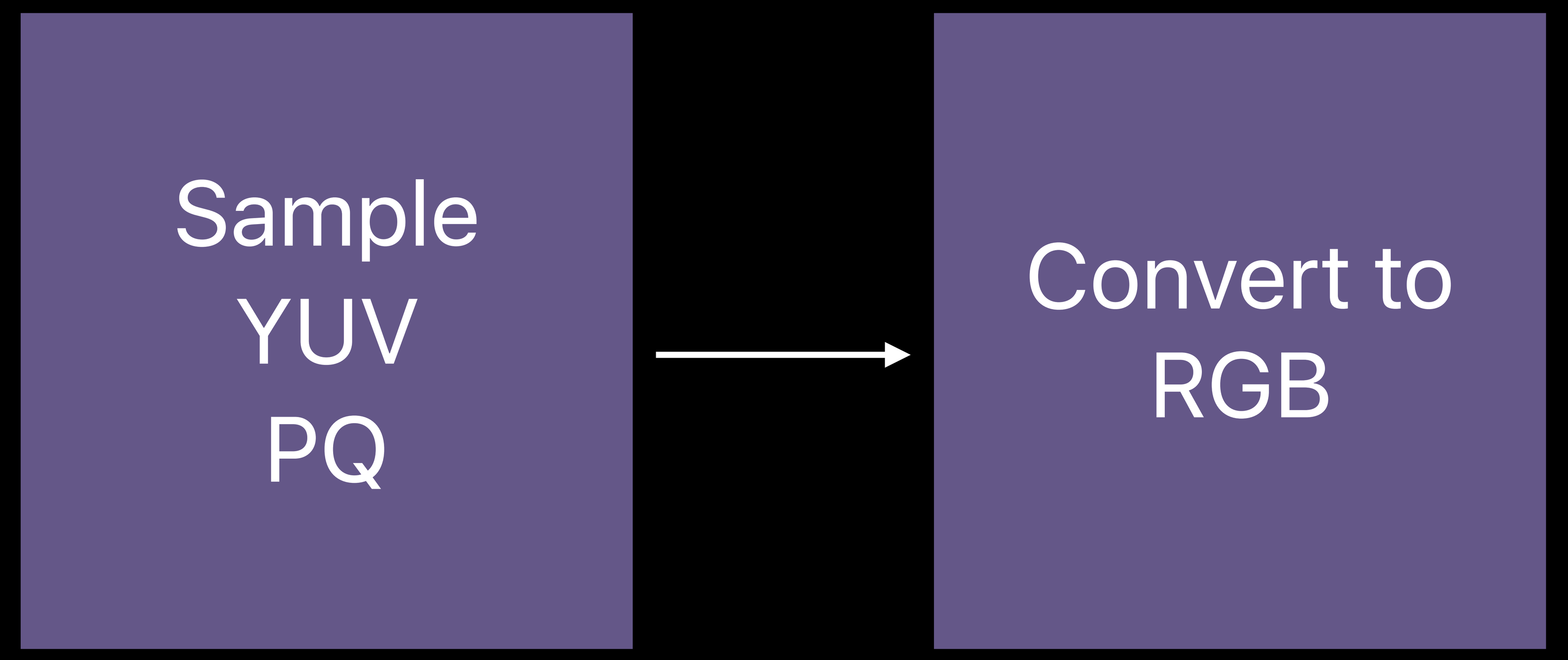
```
// Check for EDR support on display
NSScreen * screen =  view.window.screen;
CGFloat edrSupport =  screen.maximumPotentialExtendedDynamicRangeColorComponentValue;

// Set color space and transfer function
const CFStringRef name = kCGColorSpaceDisplayBT2020_PQ_EOTF;
CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(name);
metalLayer.colorspace = colorspace;
CGColorSpaceRelease(colorspace);

// Set pixel format to 16-bit floating point
metalLayer.pixelFormat = MTLPixelFormatRGBA16Float;

// Indicate contents need EDR
metalLayer.wantsExtendedDynamicRangeContent = YES;

// In main render loop, update maxEDR
float maxEDR = screen.maximumExtendedDynamicRangeColorComponentValue;
…
```

HDR Pixel Processing in Shaders

HDR Pixel Processing in Shaders

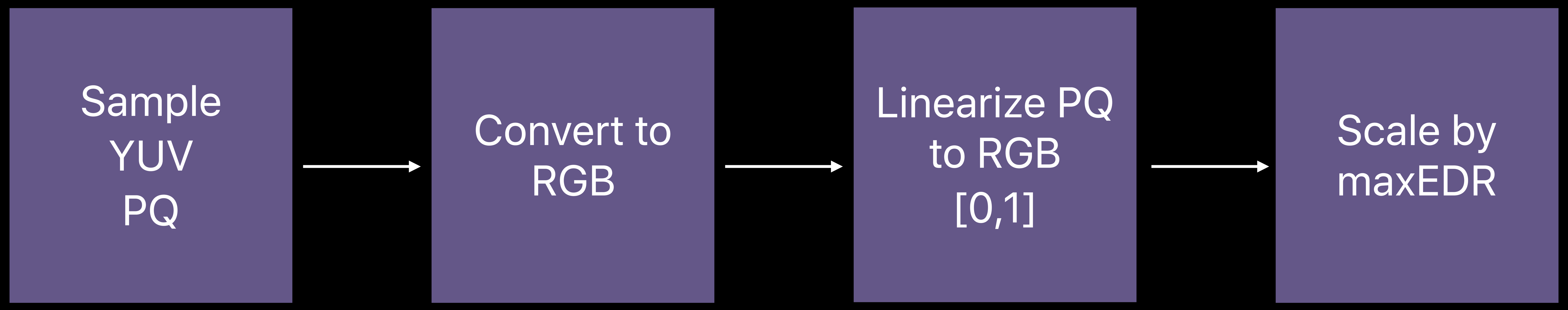
Sample
YUV
PQ

HDR Pixel Processing in Shaders

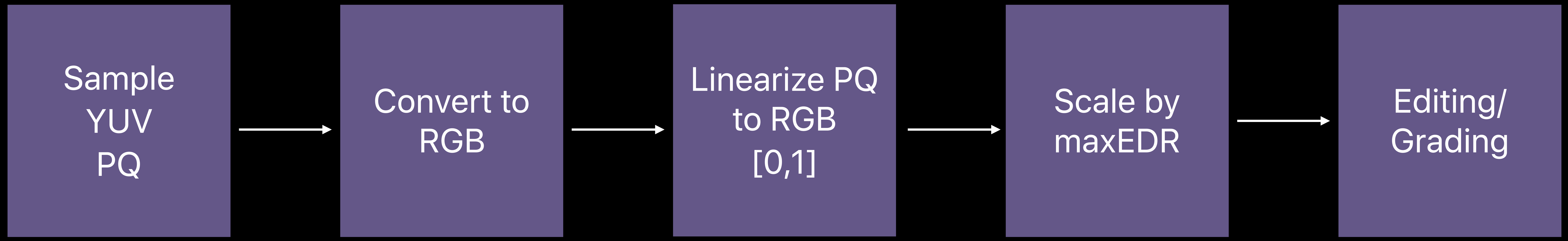


HDR Pixel Processing in Shaders

HDR Pixel Processing in Shaders



HDR Pixel Processing in Shaders



HDR Pixel Processing in Shaders

HDR Pixel Processing in Shaders

Sample YUV PQ → Convert to RGB → Linearize PQ to RGB [0,1] → Scale by maxEDR → Editing/ Grading → Tone Map

```
CAEDRMetadata *edrMetaData = CAEDRMetadata(minLuminance, maxLuminance, opticalOuputScale)
metalLayer.edrMetadata = edrMetaData;
```

Best Practices

Best Practices

Update content when display brightness changes

Best Practices

Update content when display brightness changes

MTLPixelFormatRGBA16Float recommended for HDR rendering

Best Practices

Update content when display brightness changes

MTLPixelFormatRGBA16Float recommended for HDR rendering

Select color space and transfer function that matches content

Best Practices

Update content when display brightness changes

MTLPixelFormatRGBA16Float recommended for HDR rendering

Select color space and transfer function that matches content

Bypass tone mapping if contents are already tone mapped

Optimizing for 8K video editing

Support for high dynamic range

Leveraging all platform resources

Efficient data transfers

Leveraging All Platform Resources

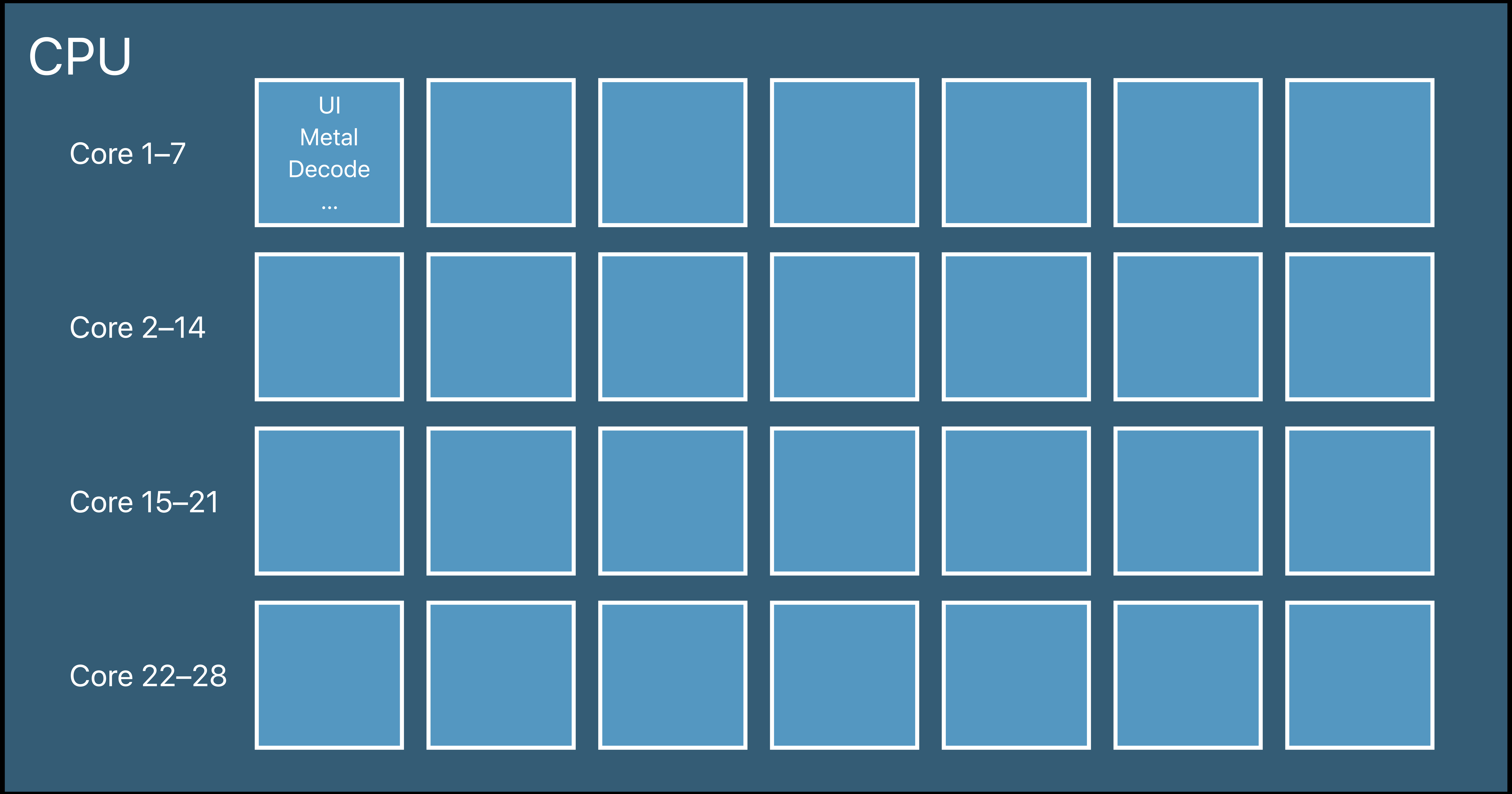
Brian Ross, GPU Software

Scale with CPU cores

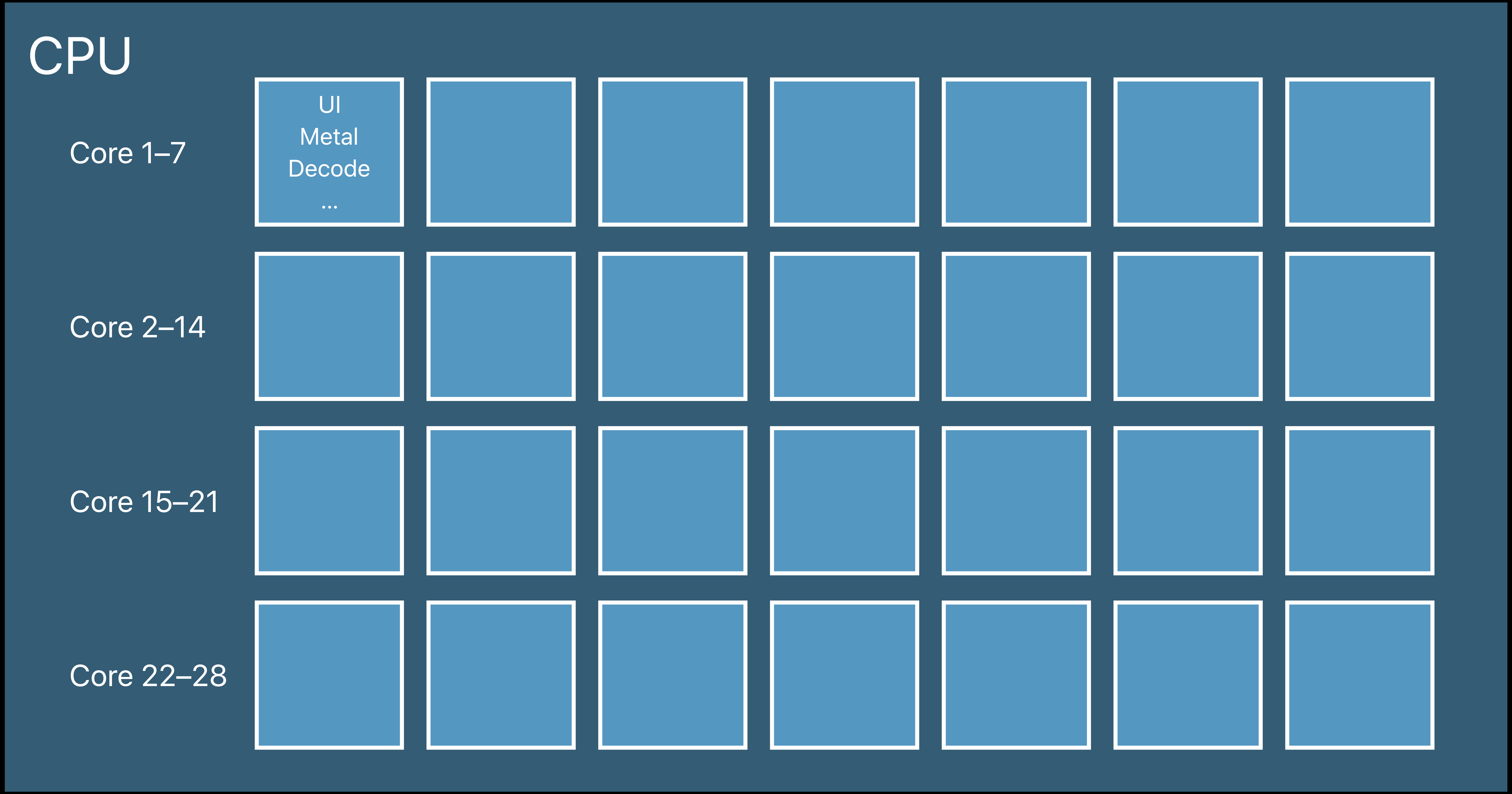
Scale with GPU channels

Scale with multiple GPUs

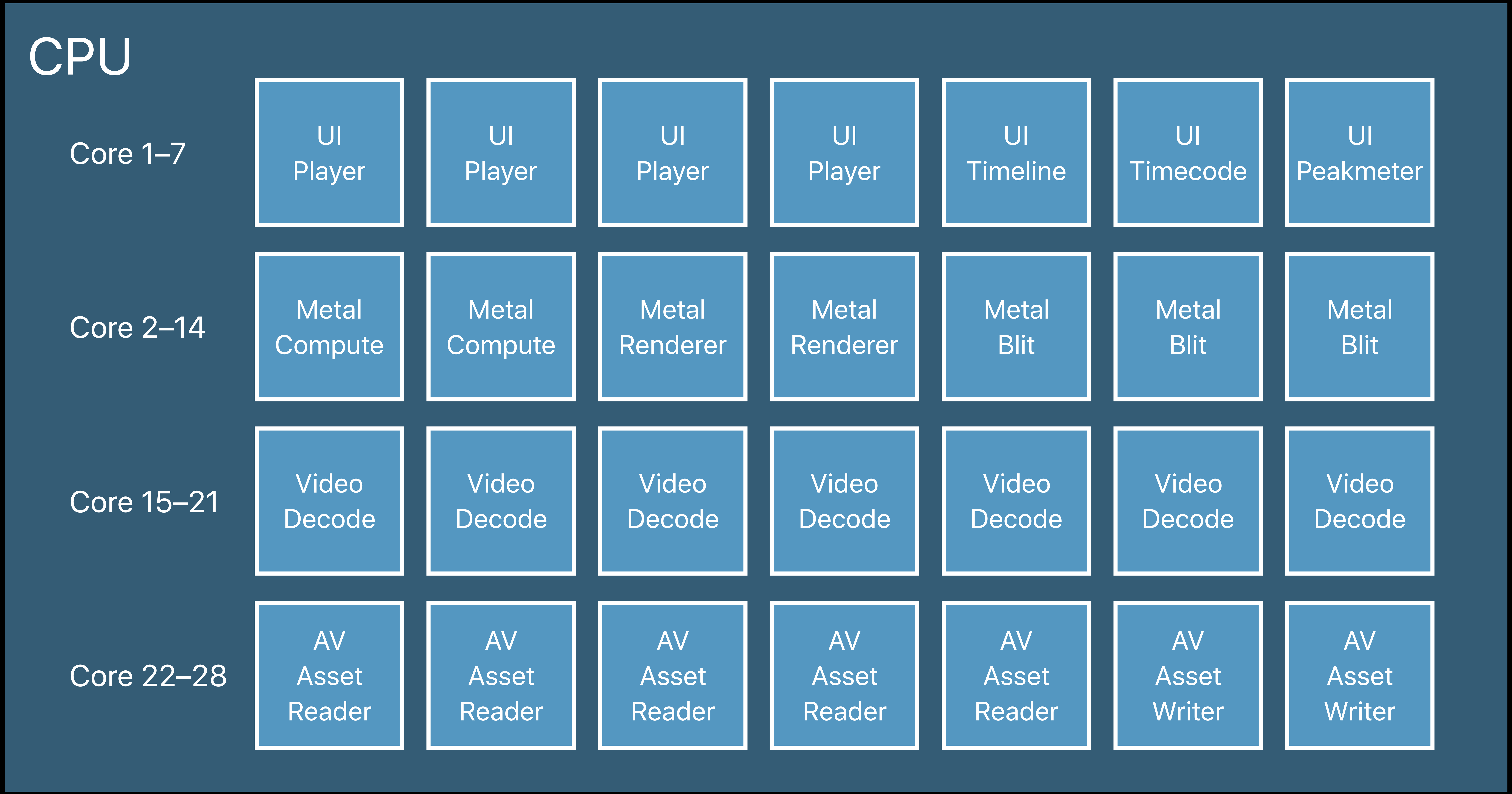
Scaling with CPU Cores



Scaling with CPU Cores



Scaling with CPU Cores



Single-Thread Rendering

Single-Thread Rendering

Single-Thread Rendering

Increased latency

Multi-Threaded Rendering

MTLCommandBuffer

Multi-Threaded Rendering

MTLCommandBuffer

Multi-Threaded Rendering

MTLCommandBuffer

```swift
// Create multiple command buffers
let commandBuffer1 = commandQueue.makeCommandBuffer()!
let commandBuffer2 = commandQueue.makeCommandBuffer()!

// Enqueue to define desired GPU execution order
commandBuffer1.enqueue()
commandBuffer2.enqueue()

// Dispatch encoding on separate threads
queue.async(group: group) {
    encodeComputeFilter( commandBuffer2, … )
    commandBuffer2.commit()
}
queue.async(group: group) {
    encodeRenderEffects( commandBuffer1, … )
    commandBuffer1.commit()
}
```

```swift
// Create multiple command buffers
let commandBuffer1 = commandQueue.makeCommandBuffer()!
let commandBuffer2 = commandQueue.makeCommandBuffer()!

// Enqueue to define desired GPU execution order
commandBuffer1.enqueue()
commandBuffer2.enqueue()

// Dispatch encoding on separate threads
queue.async(group: group) {
    encodeComputeFilter( commandBuffer2, … )
    commandBuffer2.commit()
}
queue.async(group: group) {
    encodeRenderEffects( commandBuffer1, … )
    commandBuffer1.commit()
}
```

```swift
// Create multiple command buffers
let commandBuffer1 = commandQueue.makeCommandBuffer()!
let commandBuffer2 = commandQueue.makeCommandBuffer()!

// Enqueue to define desired GPU execution order
commandBuffer1.enqueue()
commandBuffer2.enqueue()

// Dispatch encoding on separate threads
queue.async(group: group) {
    encodeComputeFilter( commandBuffer2, … )
    commandBuffer2.commit()
}
queue.async(group: group) {
    encodeRenderEffects( commandBuffer1, … )
    commandBuffer1.commit()
}
```

```swift
// Create multiple command buffers
let commandBuffer1 = commandQueue.makeCommandBuffer()!
let commandBuffer2 = commandQueue.makeCommandBuffer()!

// Enqueue to define desired GPU execution order
commandBuffer1.enqueue()
commandBuffer2.enqueue()

// Dispatch encoding on separate threads
queue.async(group: group) {
    encodeComputeFilter( commandBuffer2, … )
    commandBuffer2.commit()
}
queue.async(group: group) {
    encodeRenderEffects( commandBuffer1, … )
    commandBuffer1.commit()
}
```

Multi-Threaded Rendering

MTLCommandBuffer

Multi-Threaded Rendering

Without MTLParallelRenderCommandEncoder

Multi-Threaded Rendering

With MTLParallelRenderCommandEncoder

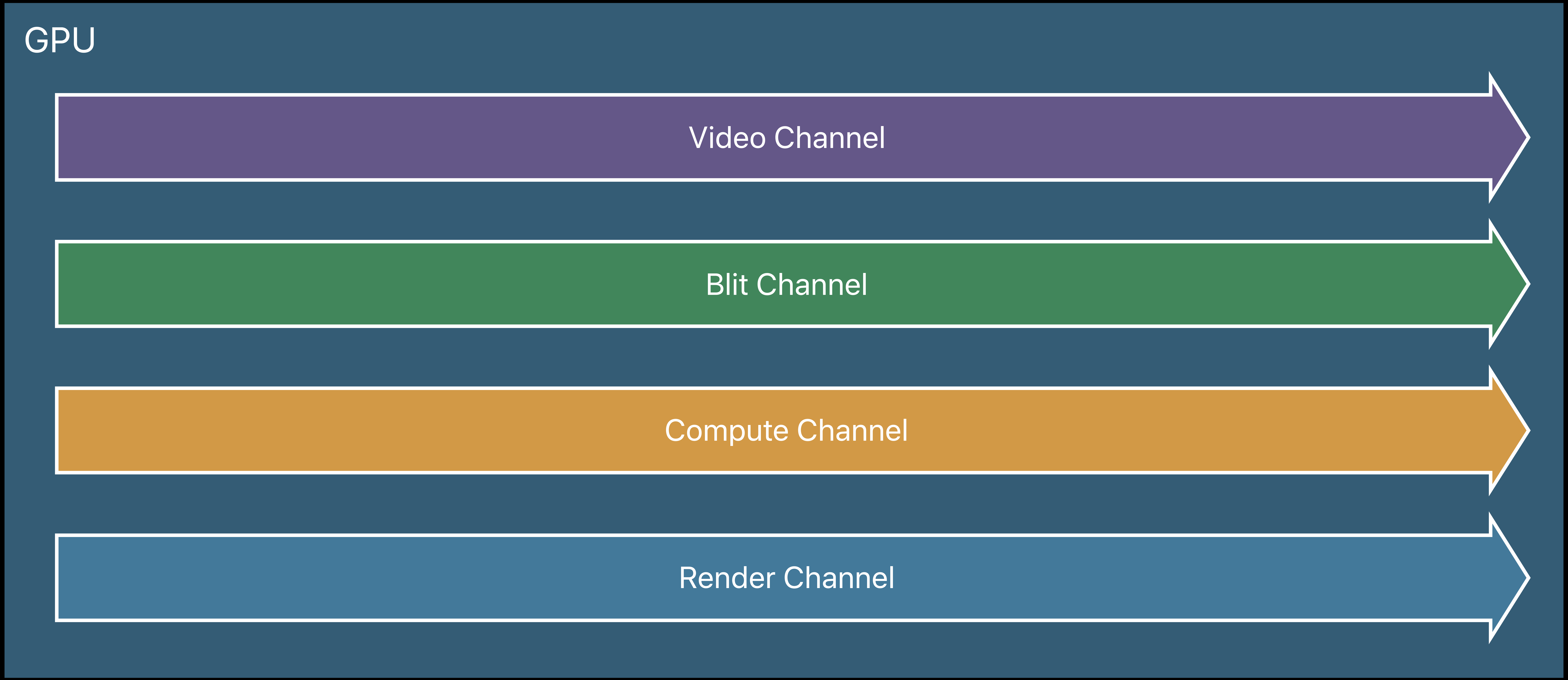
```swift
// Create parallel encoder and subordinate render command encoder objects
let parallelRenderEncoder = commandBuffer.makeParallelRenderCommandEncoder(renderPassDesc)!
let renderEncoder1 = parallelRenderEncoder.makeRenderCommandEncoder()!
let renderEncoder2 = parallelRenderEncoder.makeRenderCommandEncoder()!

// Encode different portions of render effects (in any order) on separate threads
queue.async(group: group) {
    encodeRenderEffectsPart1(renderEncoder2)
}
queue.async(group: group) {
    encodeRenderEffectsPart2(renderEncoder1)
}

// Notify when encoding complete and end the parallel encoder
group.notify(queue: queue) {
    parallelRenderEncoder.endEncoding()
}
```

```swift
// Create parallel encoder and subordinate render command encoder objects
let parallelRenderEncoder = commandBuffer.makeParallelRenderCommandEncoder(renderPassDesc)!
let renderEncoder1 = parallelRenderEncoder.makeRenderCommandEncoder()!
let renderEncoder2 = parallelRenderEncoder.makeRenderCommandEncoder()!

// Encode different portions of render effects (in any order) on separate threads
queue.async(group: group) {
    encodeRenderEffectsPart1(renderEncoder2)
}
queue.async(group: group) {
    encodeRenderEffectsPart2(renderEncoder1)
}

// Notify when encoding complete and end the parallel encoder
group.notify(queue: queue) {
    parallelRenderEncoder.endEncoding()
}
```

```swift
// Create parallel encoder and subordinate render command encoder objects
let parallelRenderEncoder = commandBuffer.makeParallelRenderCommandEncoder(renderPassDesc)!
let renderEncoder1 = parallelRenderEncoder.makeRenderCommandEncoder()!
let renderEncoder2 = parallelRenderEncoder.makeRenderCommandEncoder()!

// Encode different portions of render effects (in any order) on separate threads
queue.async(group: group) {
    encodeRenderEffectsPart1(renderEncoder2)
}
queue.async(group: group) {
    encodeRenderEffectsPart2(renderEncoder1)
}

// Notify when encoding complete and end the parallel encoder
group.notify(queue: queue) {
    parallelRenderEncoder.endEncoding()
}
```

```swift
// Create parallel encoder and subordinate render command encoder objects
let parallelRenderEncoder = commandBuffer.makeParallelRenderCommandEncoder(renderPassDesc)!
let renderEncoder1 = parallelRenderEncoder.makeRenderCommandEncoder()!
let renderEncoder2 = parallelRenderEncoder.makeRenderCommandEncoder()!

// Encode different portions of render effects (in any order) on separate threads
queue.async(group: group) {
    encodeRenderEffectsPart1(renderEncoder2)
}
queue.async(group: group) {
    encodeRenderEffectsPart2(renderEncoder1)
}

// Notify when encoding complete and end the parallel encoder
group.notify(queue: queue) {
    parallelRenderEncoder.endEncoding()
}
```

Scaling with GPU Channels



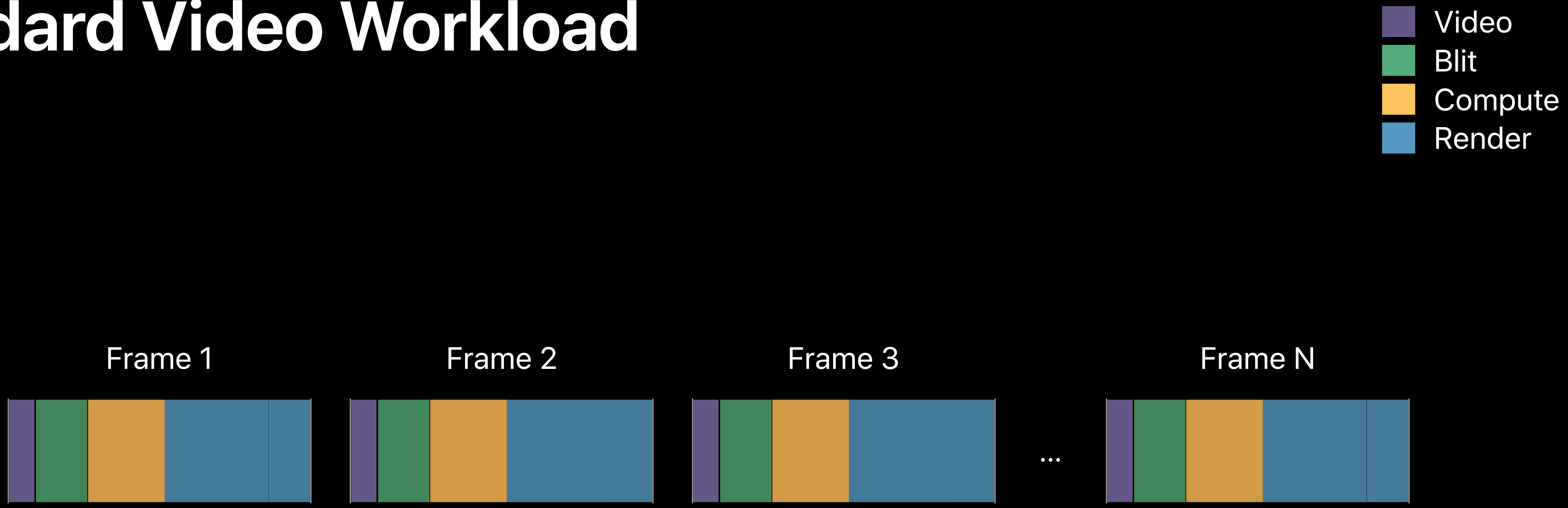
Standard Video Workload

Standard Video Workload

Standard Video Workload

Standard Video Workload

Standard Video Workload



Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps


```
commandBuffer.waitUntilCompleted()

commandBuffer.addCompletionHandler {
    // Code to execute after GPU completion
    ...
}
```

Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps

Parallel GPU Channels with Gaps

Parallel GPU Channels with Less Gaps

Parallel GPU Channels with Less Gaps

Parallel GPU Channels with Less Gaps

Parallel GPU Channels with Minimal Gaps

Scaling with Multiple GPUs

Multi GPU support in Metal

Load balancing strategies

Synchronization

Multi GPU Support in Metal

Detect multi GPU capabilities

Easy management with Metal device objects

Transfer data between GPUs NEW

Powerful synchronization support

Metal GPU Detection

MTLDevice properties

```swift
// Query for GPU properties
let gpuLocation = device.location
let gpuLocationNumber = device.locationNumber
let gpuMaxTransferRate = device.maxTransferRate
let gpuIsLowPower = device.isLowPower
let gpuIsHeadless = device.isHeadless
let gpuIsRemovable = device.isRemovable
```

	Location	LocationNumber	isRemovable	isLowPower
Integrated GPU	MTLDeviceLocationBuiltin	0	FALSE	TRUE
Discrete GPU	MTLDeviceLocationBuiltin	1	FALSE	FALSE
External GPU	MTLDeviceLocationExternal	Thunderbolt Port (1..N)	TRUE	FALSE
Mac Pro GPU	MTLDeviceLocationSlot	PCI Slot (0..N)	FALSE	FALSE

Load Balancing

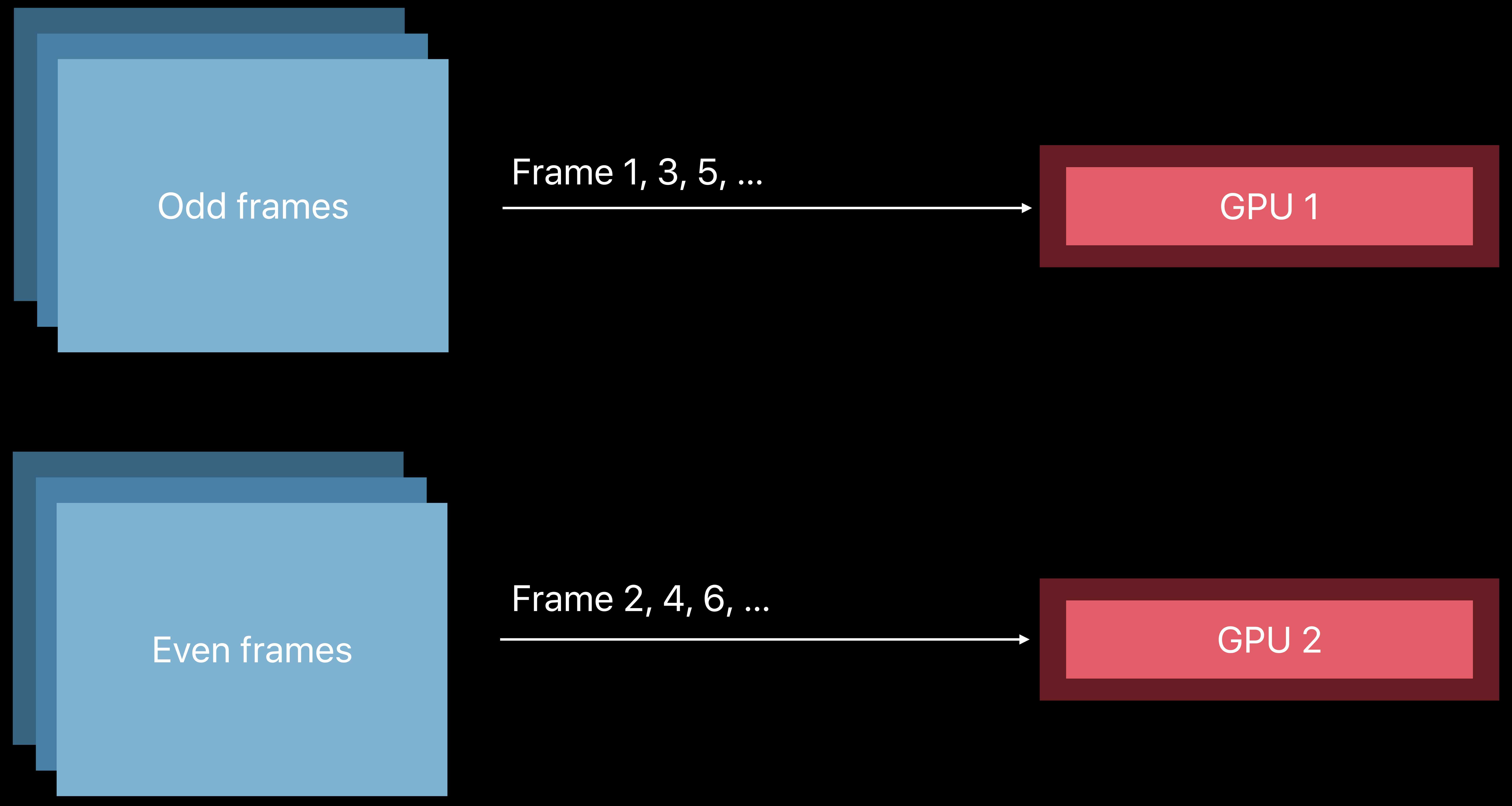
Many load balancing schemes

Consider your software architecture

Goal is high scaling efficiency

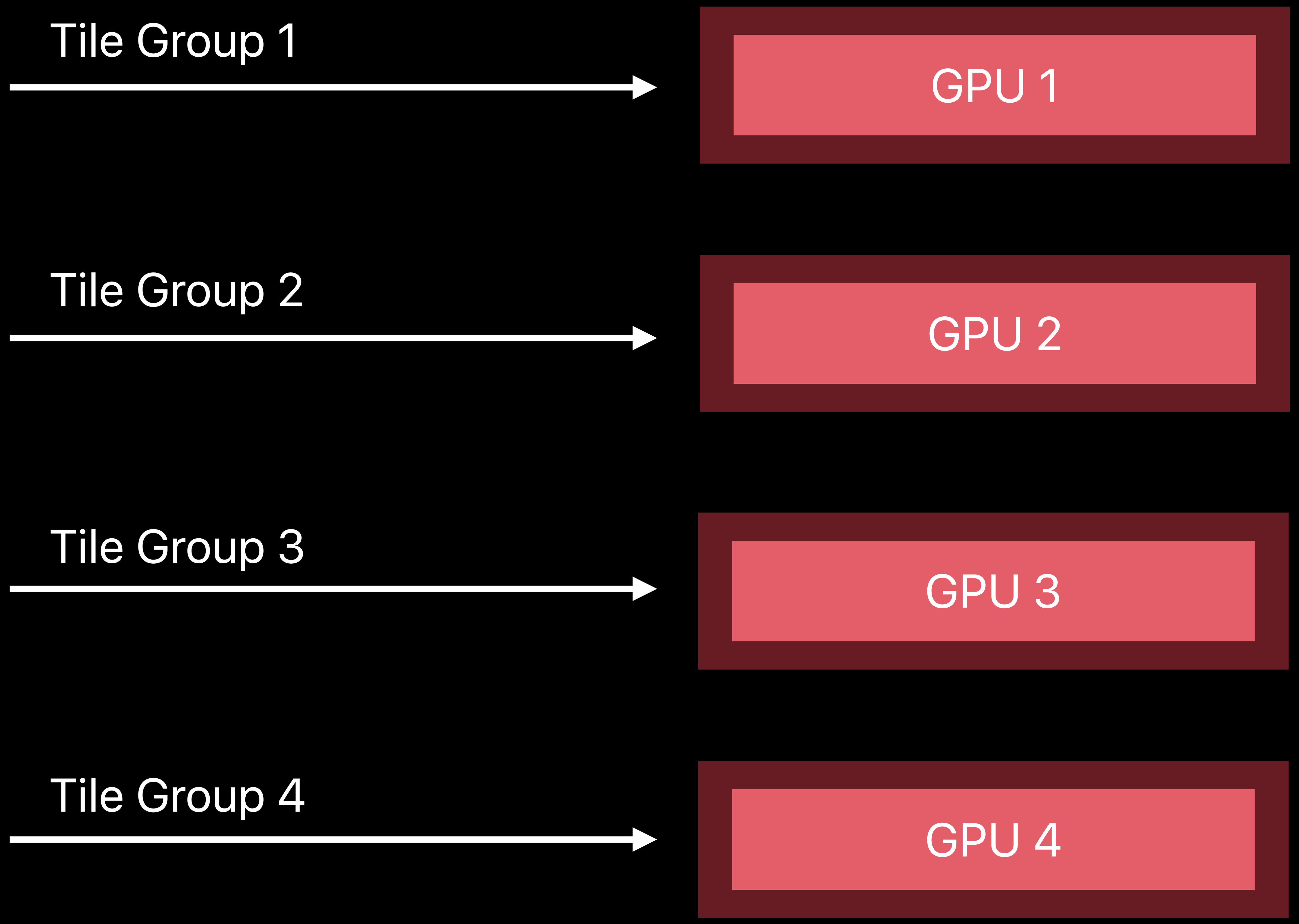
Load Balancing Strategies

Alternating frames



Load Balancing Strategies

Interleaved tiling



Load Balancing Strategies

Interleaved tiling

Load Balancing Strategies

Interleaved tiling

Load Balancing Strategies

Interleaved tiling

Tile Group 1 → GPU 1

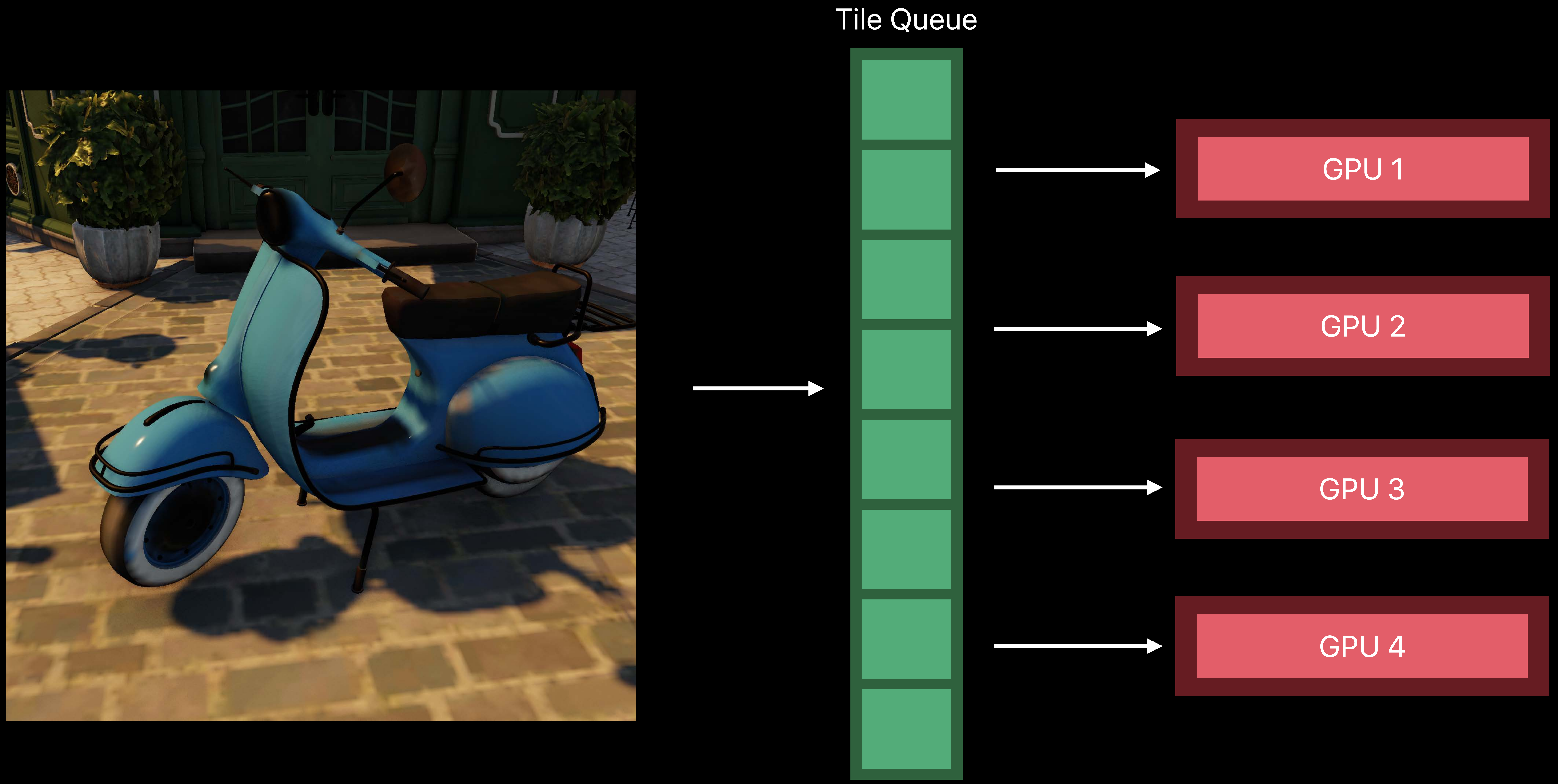
Tile Group 2 → GPU 2

Tile Group 3 → GPU 3

Tile Group 4 → GPU 4

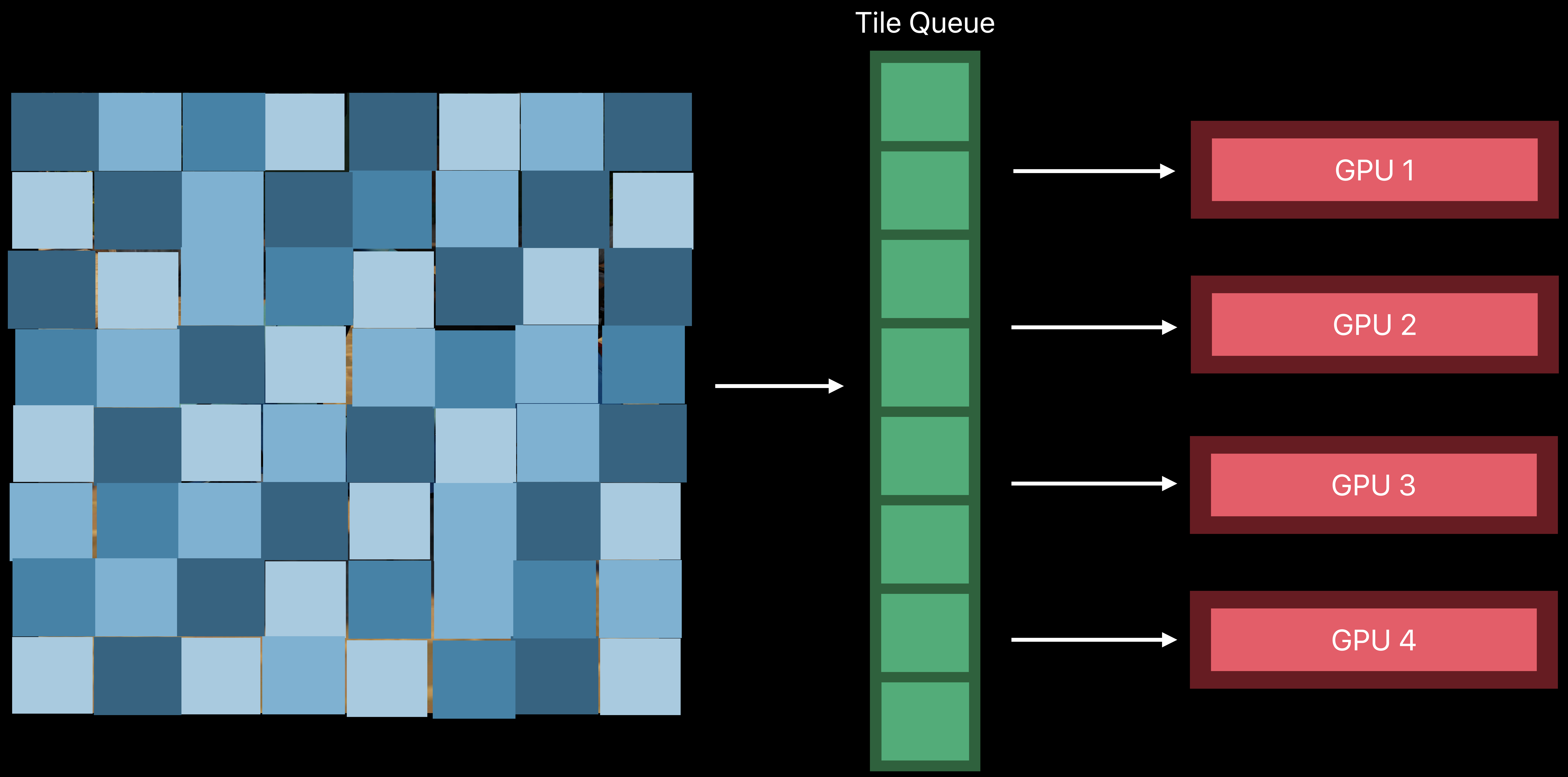
Load Balancing Strategies

Interleaved tiling with queue



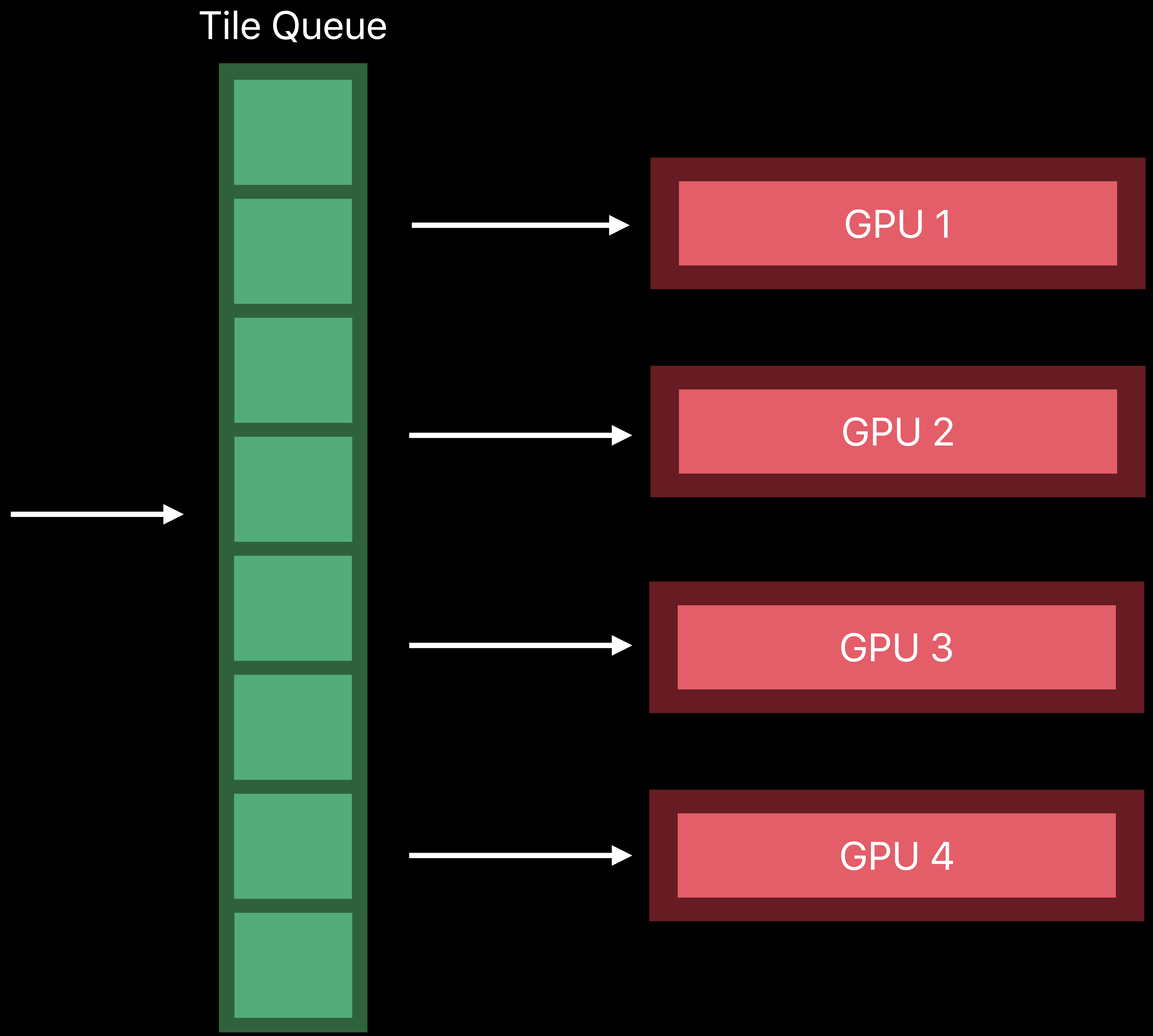
Load Balancing Strategies

Interleaved tiling with queue



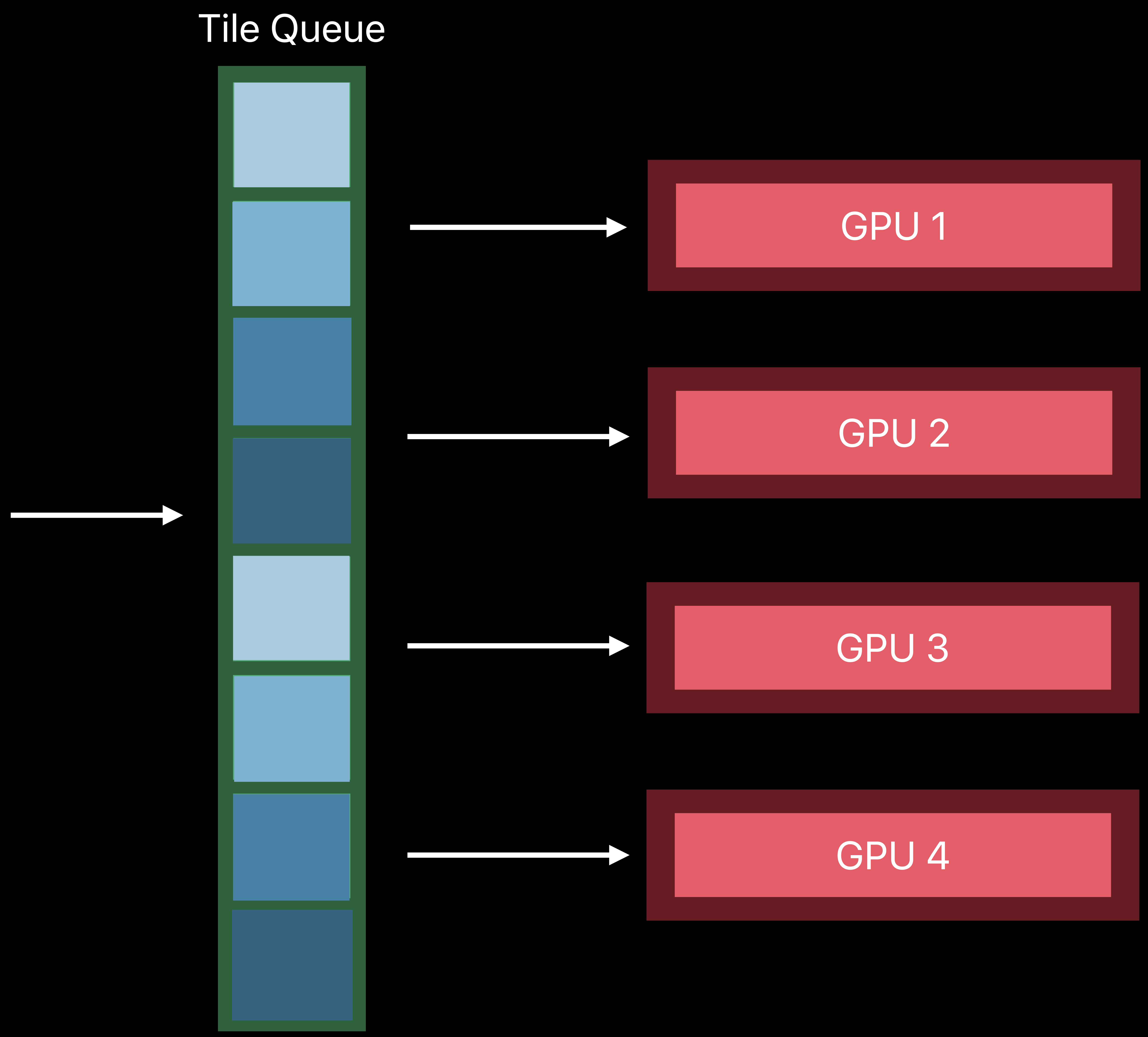
Load Balancing Strategies

Interleaved tiling with queue



Load Balancing Strategies

Interleaved tiling with queue



Multi GPU Synchronization

MTLSharedEvent

Synchronize between

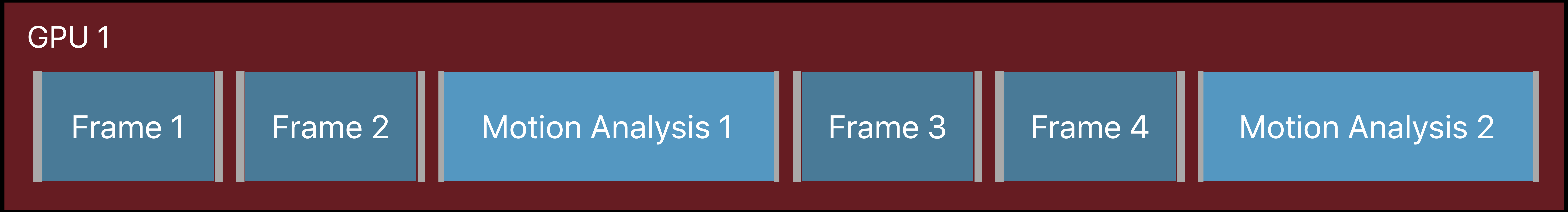
- Multiple GPUs
- CPU and GPU
- Processes

Multi GPU Synchronization

MTLSharedEvent

Multi GPU Synchronization

MTLSharedEvent

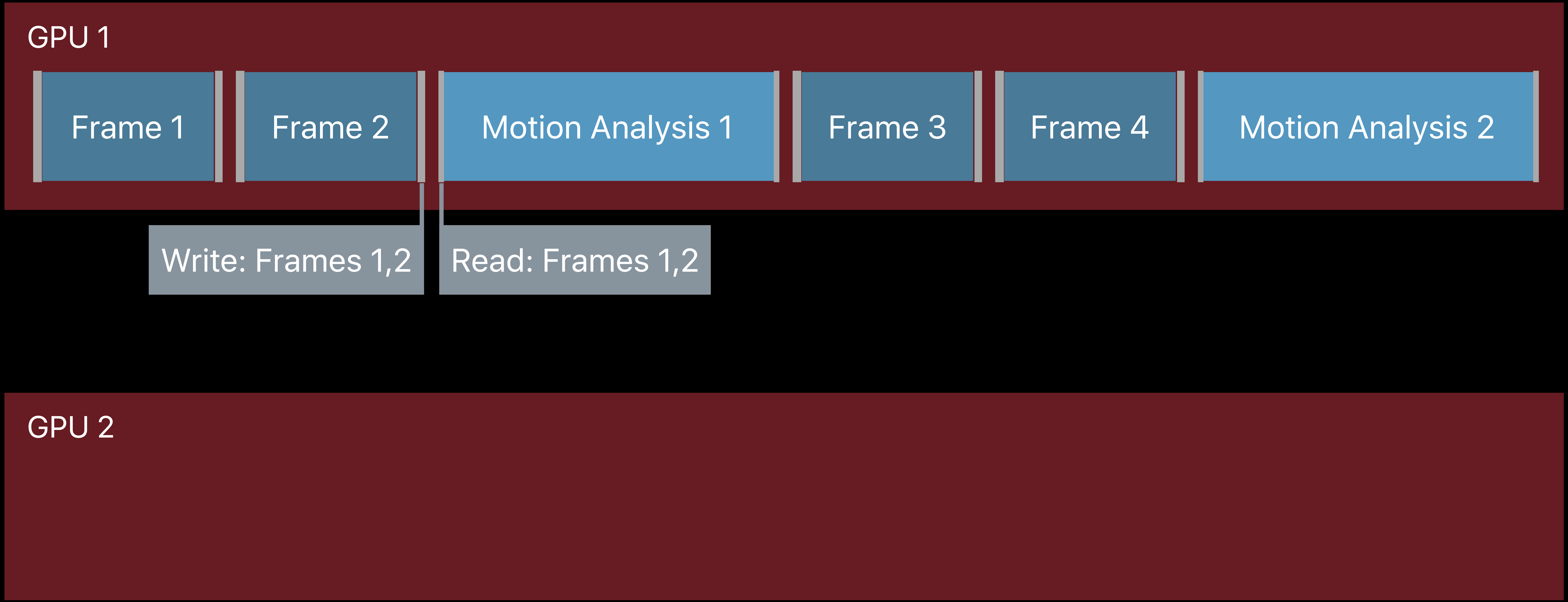


Multi GPU Synchronization

MTLSharedEvent

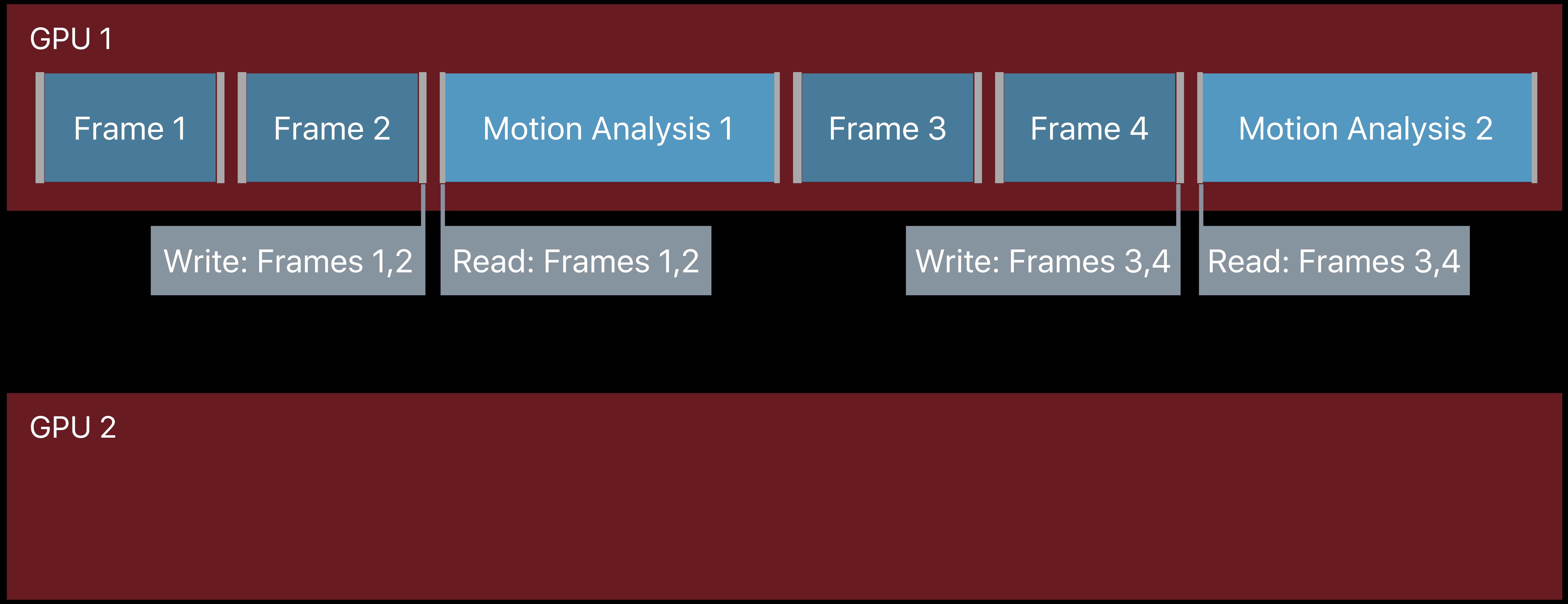
Multi GPU Synchronization

MTLSharedEvent



Multi GPU Synchronization

MTLSharedEvent



Multi GPU Synchronization

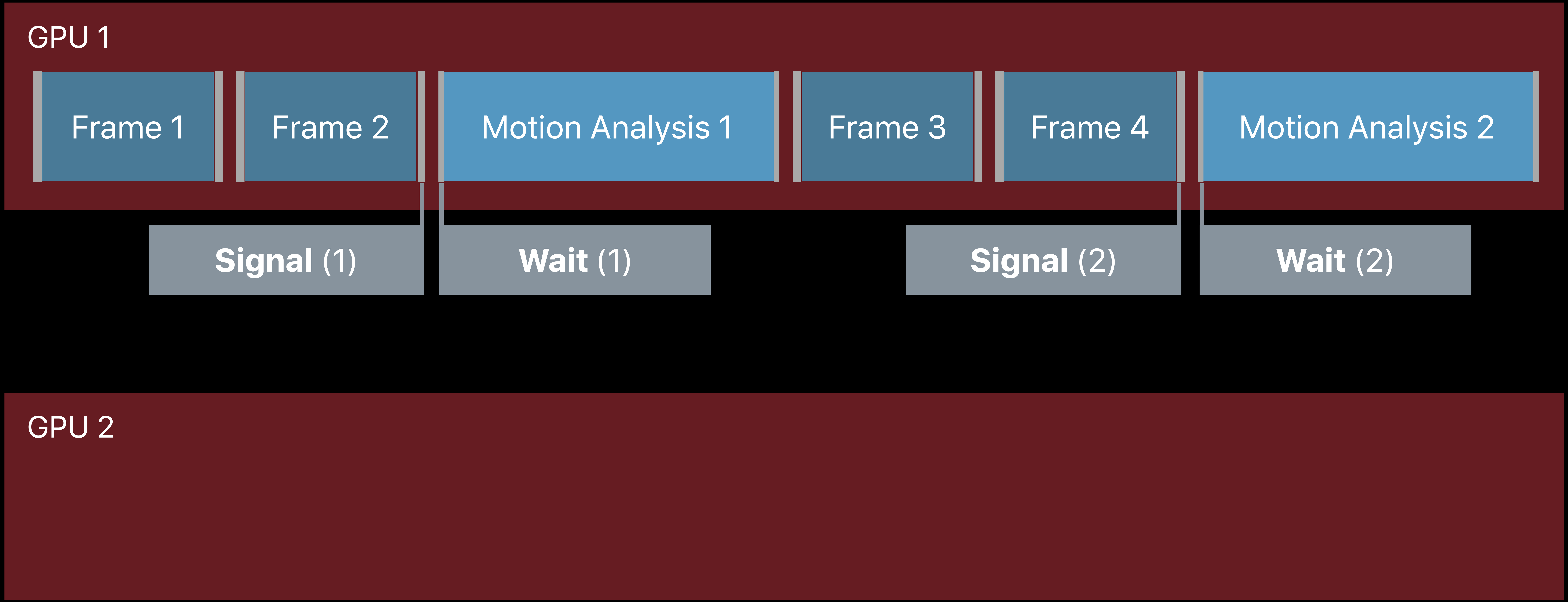
MTLSharedEvent

Multi GPU Synchronization

MTLSharedEvent

Multi GPU Synchronization

MTLSharedEvent



Multi GPU Synchronization

MTLSharedEvent

Multi GPU Synchronization

MTLSharedEvent

```swift
// Create shared event and command queues
let sharedEvent = deviceA.makeSharedEvent()!
let commandQueueA = deviceA.makeCommandQueue()!
let commandQueueB = deviceB.makeCommandQueue()!

// Encode Frame Rendering
let commandBufferA = commandQueueA.makeCommandBuffer()!
encodeRenderFrames(commandBufferA)
commandBufferA.encodeSignalEvent(sharedEvent)
commandBufferA.commit()

// Encode motion analysis (Optical Flow)
let commandBufferB = commandQueueB.makeCommandBuffer()!
commandBufferB.encodeWaitEvent(sharedEvent)
encodeMotionAnalysis(commandBufferB)
commandBufferB.commit()
```

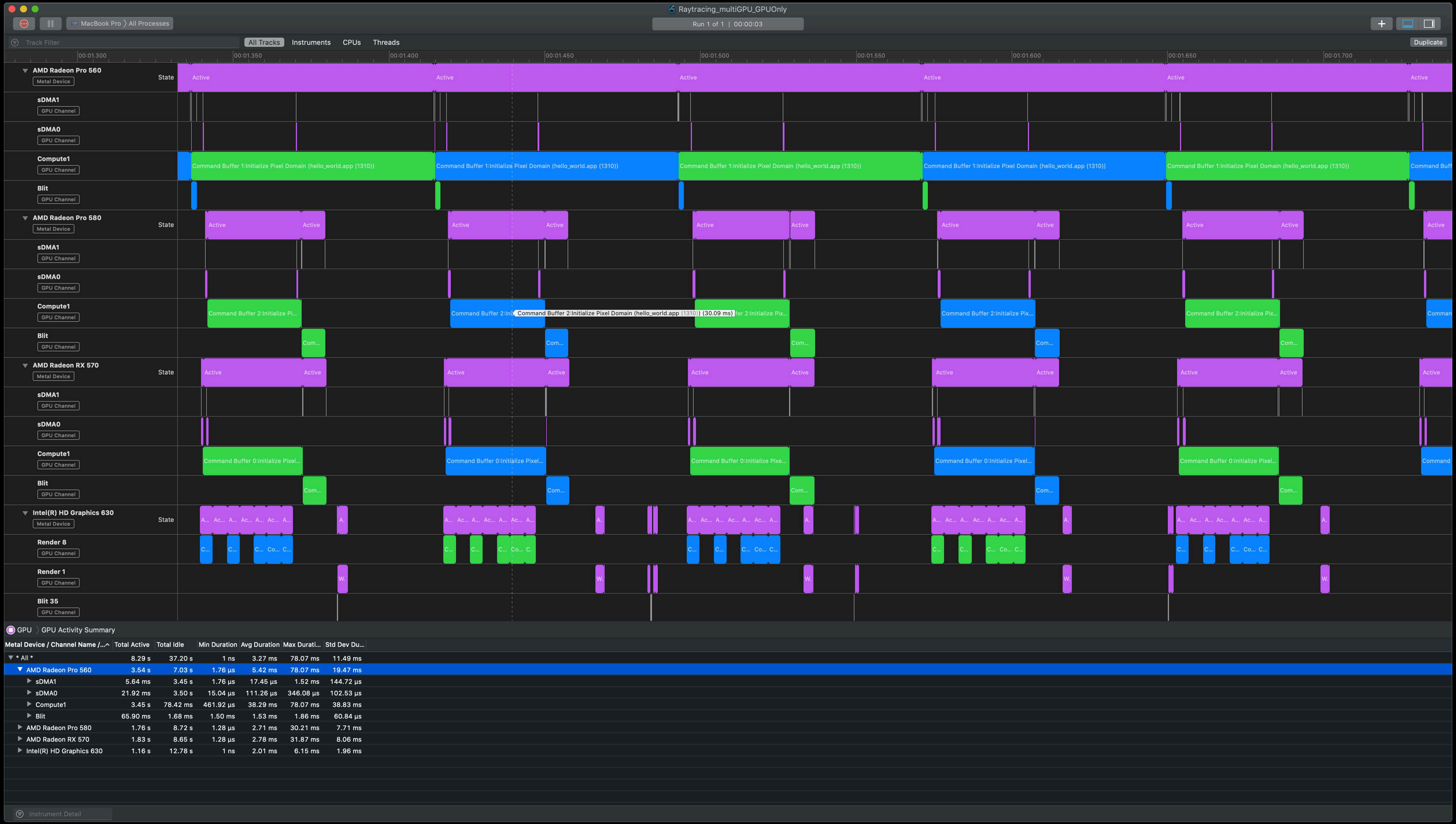
```swift
// Create shared event and command queues
let sharedEvent = deviceA.makeSharedEvent()!
let commandQueueA = deviceA.makeCommandQueue()!
let commandQueueB = deviceB.makeCommandQueue()!

// Encode Frame Rendering
let commandBufferA = commandQueueA.makeCommandBuffer()!
encodeRenderFrames(commandBufferA)
commandBufferA.encodeSignalEvent(sharedEvent)
commandBufferA.commit()

// Encode motion analysis (Optical Flow)
let commandBufferB = commandQueueB.makeCommandBuffer()!
commandBufferB.encodeWaitEvent(sharedEvent)
encodeMotionAnalysis(commandBufferB)
commandBufferB.commit()
```

```swift
// Create shared event and command queues
let sharedEvent = deviceA.makeSharedEvent()!
let commandQueueA = deviceA.makeCommandQueue()!
let commandQueueB = deviceB.makeCommandQueue()!

// Encode Frame Rendering
let commandBufferA = commandQueueA.makeCommandBuffer()!
encodeRenderFrames(commandBufferA)
commandBufferA.encodeSignalEvent(sharedEvent)
commandBufferA.commit()

// Encode motion analysis (Optical Flow)
let commandBufferB = commandQueueB.makeCommandBuffer()!
commandBufferB.encodeWaitEvent(sharedEvent)
encodeMotionAnalysis(commandBufferB)
commandBufferB.commit()
```

```swift
// Create shared event and command queues
let sharedEvent = deviceA.makeSharedEvent()!
let commandQueueA = deviceA.makeCommandQueue()!
let commandQueueB = deviceB.makeCommandQueue()!

// Encode Frame Rendering
let commandBufferA = commandQueueA.makeCommandBuffer()!
encodeRenderFrames(commandBufferA)
commandBufferA.encodeSignalEvent(sharedEvent)
commandBufferA.commit()

// Encode motion analysis (Optical Flow)
let commandBufferB = commandQueueB.makeCommandBuffer()!
commandBufferB.encodeWaitEvent(sharedEvent)
encodeMotionAnalysis(commandBufferB)
commandBufferB.commit()
```

```swift
// Create shared event and command queues
let sharedEvent = deviceA.makeSharedEvent()!
let commandQueueA = deviceA.makeCommandQueue()!
let commandQueueB = deviceB.makeCommandQueue()!

// Encode Frame Rendering
let commandBufferA = commandQueueA.makeCommandBuffer()!
encodeRenderFrames(commandBufferA)
commandBufferA.encodeSignalEvent(sharedEvent)
commandBufferA.commit()

// Encode motion analysis (Optical Flow)
let commandBufferB = commandQueueB.makeCommandBuffer()!
commandBufferB.encodeWaitEvent(sharedEvent)
encodeMotionAnalysis(commandBufferB)
commandBufferB.commit()
```

Metal System Trace




00:01.300
00:01.350
00:01.400
00:01.450
00:01.500
00:01.550
AMD Radeon Pro 560
Metal Device
State
Active
sDMA1
GPU Channel
sDMA0
GPU Channel
Compute1
GPU Channel
Command Buffer 1:Initialize Pixel Domain (hello_world.app (1310))
Blit
GPU Channel
AMD Radeon Pro 580
Metal Device
State
Active
sDMA1
GPU Channel
sDMA0
GPU Channel
Compute1
GPU Channel
Command Buffer 2:Initialize Pi...
Command Buffer 2:Initialize Pixel Domain (hello_world.app (1310)) (30.09 ms)
Blit
GPU Channel
Com...
AMD Radeon RX 570
Metal Device
State
Active
sDMA1
GPU Channel
sDMA0
GPU Channel
Compute1
GPU Channel
Command Buffer 0:Initialize Pixel...
Blit
GPU Channel
Com...
Intel(R) HD Graphics 630
Metal Device
State
Render 8
GPU Channel
Render 1
GPU Channel
Blit 35
GPU Channel
GPU
GPU Activity Summary

Blit 35

GPU Channel

GPU 〉 GPU Activity Summary

Metal Device / Channel Name /...	Total Active	Total Idle	Min Duration	Avg Duration	Max Durati...	Std Dev Du...
* All *	8.29 s	37.20 s	1 ns	3.27 ms	78.07 ms	11.49 ms
AMD Radeon Pro 560	3.54 s	7.03 s	1.76 µs	5.42 ms	78.07 ms	19.47 ms
sDMA1	5.64 ms	3.45 s	1.76 µs	17.45 µs	1.52 ms	144.72 µs
sDMA0	21.92 ms	3.50 s	15.04 µs	111.26 µs	346.08 µs	102.53 µs
Compute1	3.45 s	78.42 ms	461.92 µs	38.29 ms	78.07 ms	38.83 ms
Blit	65.90 ms	1.68 ms	1.50 ms	1.53 ms	1.86 ms	60.84 µs
AMD Radeon Pro 580	1.76 s	8.72 s	1.28 µs	2.71 ms	30.21 ms	7.71 ms
AMD Radeon RX 570	1.83 s	8.65 s	1.28 µs	2.78 ms	31.87 ms	8.06 ms
Intel(R) HD Graphics 630	1.16 s	12.78 s	1 ns	2.01 ms	6.15 ms	1.96 ms

Instrument Detail

Blit 35

GPU Channel

GPU 〉 GPU Activity Summary 〉 AMD Radeon Pro 560

Channel Name	Creation^	Duration	Frame	Label
Compute1	00:00.162.640	77.82 ms	Frame 1	Command Buffer 1:Initialize...
sDMA1	00:00.162.699	1.92 µs	n/a	hello_world.app (1310)
sDMA1	00:00.162.721	1.52 ms	n/a	Unknown 0x1d01ff04
sDMA1	00:00.164.242	4.32 µs	n/a	Unknown 0x1d01ff07
sDMA1	00:00.166.396	4.16 µs	n/a	Unknown 0x1d01ff28
sDMA1	00:00.166.454	1.76 µs	n/a	Unknown 0x0
sDMA0	00:00.166.478	204.96 µs	n/a	Page On 1,048,576 bytes (...
sDMA0	00:00.166.684	19.20 µs	n/a	Page On 65,536 bytes (hell...
sDMA1	00:00.197.134	5.28 µs	n/a	Unknown 0x1d01ff61
sDMA1	00:00.197.209	1.76 µs	n/a	Unknown 0x0
sDMA0	00:00.197.239	20.32 µs	n/a	Page On 65,536 bytes (hell...
sDMA0	00:00.197.261	185.92 µs	n/a	Page On 1,048,576 bytes (...
Compute1	00:00.240.462	463.52 µs	Frame 1	Command Buffer 5:Compac...

Instrument Detail

Blit 35

GPU Channel

GPU 〉IO Surface Accesses

Timestamp^	Process	Surface ID	Is Write	Width	Height	Pixel Format
00:00.398.385	WindowServer (163)	1	Yes	3360	2100	r01l
00:00.400.645	hello_world.app (1310)	181	Yes	2696	1988	ARGB
00:00.415.237	WindowServer (163)	2	Yes	3360	2100	r01l
00:00.415.445	WindowServer (163)	1	Yes	3360	2100	r01l
00:00.415.445	WindowServer (163)	2	Yes	3360	2100	r01l
00:00.449.681	WindowServer (163)	181	No	2696	1988	ARGB
00:00.449.681	WindowServer (163)	60	Yes	473	473	ARGB
00:00.449.681	WindowServer (163)	12	Yes	3360	2100	r01l
00:00.449.681	WindowServer (163)	32	Yes	297	297	ARGB
00:00.449.865	WindowServer (163)	2	Yes	3360	2100	r01l
00:00.449.865	WindowServer (163)	12	Yes	3360	2100	r01l
00:00.465.730	WindowServer (163)	60	Yes	473	473	ARGB
00:00.465.730	WindowServer (163)	32	Yes	297	297	ARGB

Instrument Detail


Raytracing_multiGPU_Everything
MacBook Pro 〉 Choose Target...
Run 1 of 1 | 00:00:03
Track Filter
All Tracks
Instruments
CPUs
Threads
Duplicate
AMD Radeon Pro 580
Metal Device
State
Active
Allocation
Metal Resource
Main Thread
Thread 0x310d
hello_world...
Driver Processing
GPU Driver
MTLEvent
Signal
Wait
sDMA1
GPU Channel
sDMA0
Compute1
Command Buffer 2:Initialize Pixel Domain (hello_world (1310))
Blit
AMD Radeon RX 570
Command Buffer 4 (hello_world (1310))
Blit Command 0
Shared Event 1: Waiting for signal value 639
Shared Event 1: Waiting for signal value 639 (30.08 ms)
Shared Event 1: Waiting for signal value 640
Command Buffer 0:Initialize Pixel Domain (hello_world (1310))
Intel(R) HD Graphics 630
MTLEvent 〉 MTLEvent Activities
Timestamp Event ID Event Type State Value Duration Process
00:00.050.435 72,057,59... Shared Signal 630 n/a hello_world (1310)
00:00.050.435 1 Shared Signal 630 n/a hello_world (1310)
00:00.050.486 1 Shared Wait 630 30.04 ms hello_world (1310)
00:00.128.401 72,057,59... Shared Signal 631 n/a hello_world (1310)
00:00.128.401 1 Shared Signal 631 n/a hello_world (1310)
00:00.128.422 1 Shared Wait 631 30.22 ms hello_world (1310)
00:00.206.850 72,057,59... Shared Signal 632 n/a hello_world (1310)
00:00.206.850 1 Shared Signal 632 n/a hello_world (1310)
00:00.206.886 1 Shared Wait 632 30.24 ms hello_world (1310)
00:00.284.989 72,057,59... Shared Signal 633 n/a hello_world (1310)
00:00.284.989 1 Shared Signal 633 n/a hello_world (1310)
No Detail
Input Filter
Instrument Detail


GPU Channel
Command Buffer 2:Initialize Pixel Domain (hello_world (1310))
Blit
GPU Channel
Command Buffer 8:...
AMD Radeon RX 570
Metal Device
State
Active
Active
Allocation
Metal Resource
Main Thread
Thread
0x310d
hello_world...
C.
Driver Processing
GPU Driver
Command Buffer 4 (hello_world (1310))
Blit Command 0
Shared Event 1: Waiting for signal value 639
Shared Event 1: Waiting for signal value 639 (30.08 ms)
MTLEvent
GPU Driver
Signal
Wait
sDMA1
GPU Channel
sDMA0
GPU Channel
Compute1
GPU Channel
Command Buffer 0:Initialize Pixel Domain (hello_world (1310))
Blit
GPU Channel
Command Buffer 4:...
Intel(R) HD Graphics 630
Metal Device
State
Active
Active
Active
Active
Active
Active
Active
Active
Allocation
Metal Resource
MTLEvent
MTLEvent Activities
Timestamp
Event ID
Event Type
State
Value
Duration
Process
00:00.050.435
72,057,59...
Shared
Signal
630
n/a
hello_world (1310)

Intel(R) HD Graphics 630

Metal Device

State

Active Active Active Activ

Allocation

Metal Resource

MTLEvent 〉 MTLEvent Activities

Timestamp	Event ID	Event Type	State	Value	Duration	Process
00:00.050.435	72,057,59...	Shared	Signal	630	n/a	hello_wo
00:00.050.435	1	Shared	Signal	630	n/a	hello_wo
00:00.050.486	1	Shared	Wait	630	30.04 ms	hello_wo
00:00.128.401	72,057,59...	Shared	Signal	631	n/a	hello_wo
00:00.128.401	1	Shared	Signal	631	n/a	hello_wo
00:00.128.422	1	Shared	Wait	631	30.22 ms	hello_wo
00:00.206.850	72,057,59...	Shared	Signal	632	n/a	hello_wo
00:00.206.850	1	Shared	Signal	632	n/a	hello_wo
00:00.206.886	1	Shared	Wait	632	30.24 ms	hello_wo
00:00.284.989	72,057,59...	Shared	Signal	633	n/a	hello_wo
00:00.284.989	1	Shared	Signal	633	n/a	hello_wo

Input Filter

Instrument Detail

Optimizing for 8K video editing

Support for high dynamic range

Leveraging all platform resources

Efficient data transfers

Efficient Data Transfers

Bandwidth and Mac Pro configurations

Transfer strategies with Infinity Fabric Link NEW

Unlocking challenging workflows

Baseline Transfer Rates

Baseline Transfer Rates

Baseline Transfer Rates

Baseline Transfer Rates

Baseline Transfer Rates

Mac Pro Dual GPU Configuration

Heavy bandwidth

NEW

Mac Pro Dual GPU Configuration

Heavy bandwidth

NEW

Mac Pro Dual GPU Configuration

Heavy bandwidth

NEW

Mac Pro Dual GPU Configuration

Heavy bandwidth

NEW

Mac Pro Quad GPU Configuration

Heavy compute

NEW

Mac Pro Quad GPU Configuration

Heavy compute

NEW

Mac Pro Quad GPU Configuration

Heavy compute

NEW

Transfer Strategies with Infinity Fabric

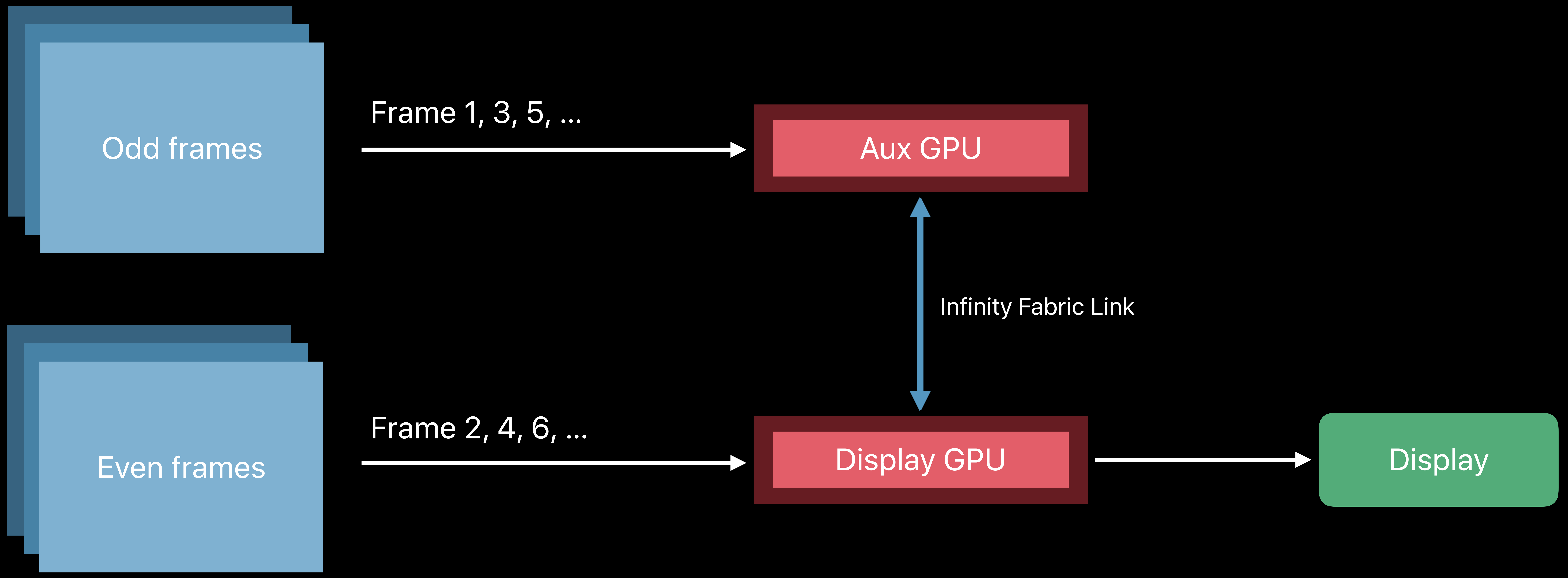
Many transfer schemes

Consider your software architecture

Goal is high bandwidth efficiency

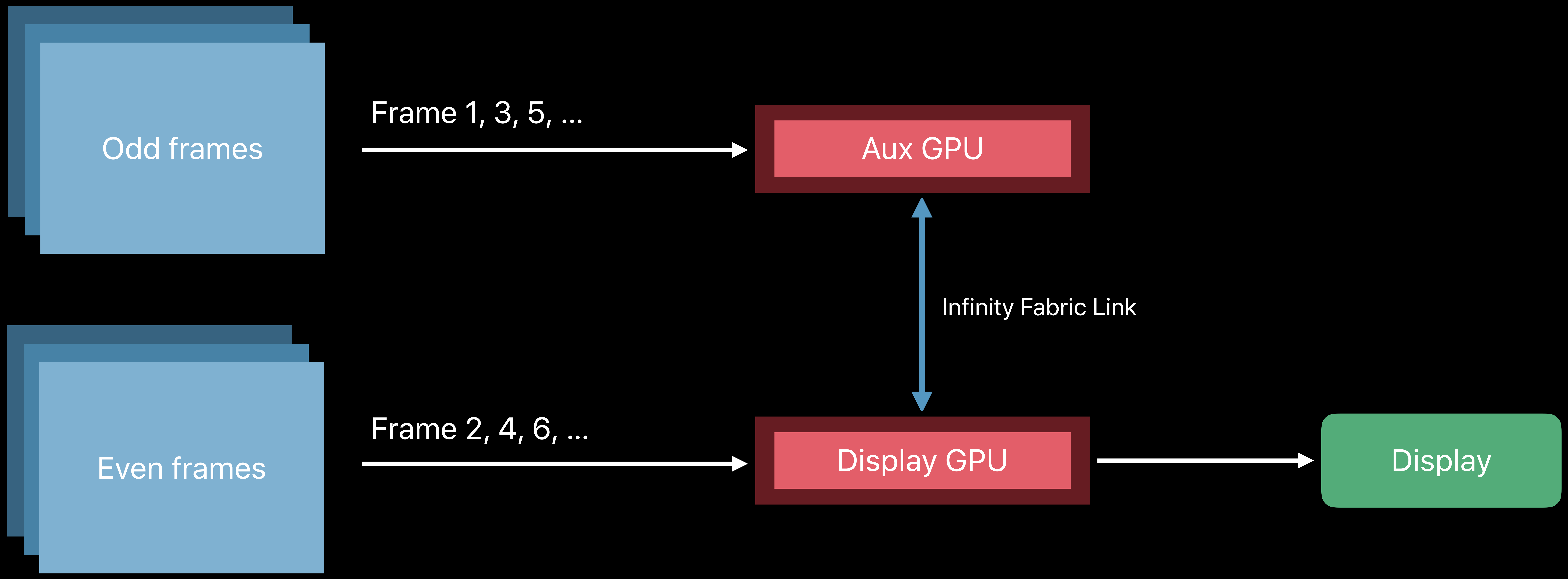
Infinity Fabric Transfer Strategies

Transfer entire frames



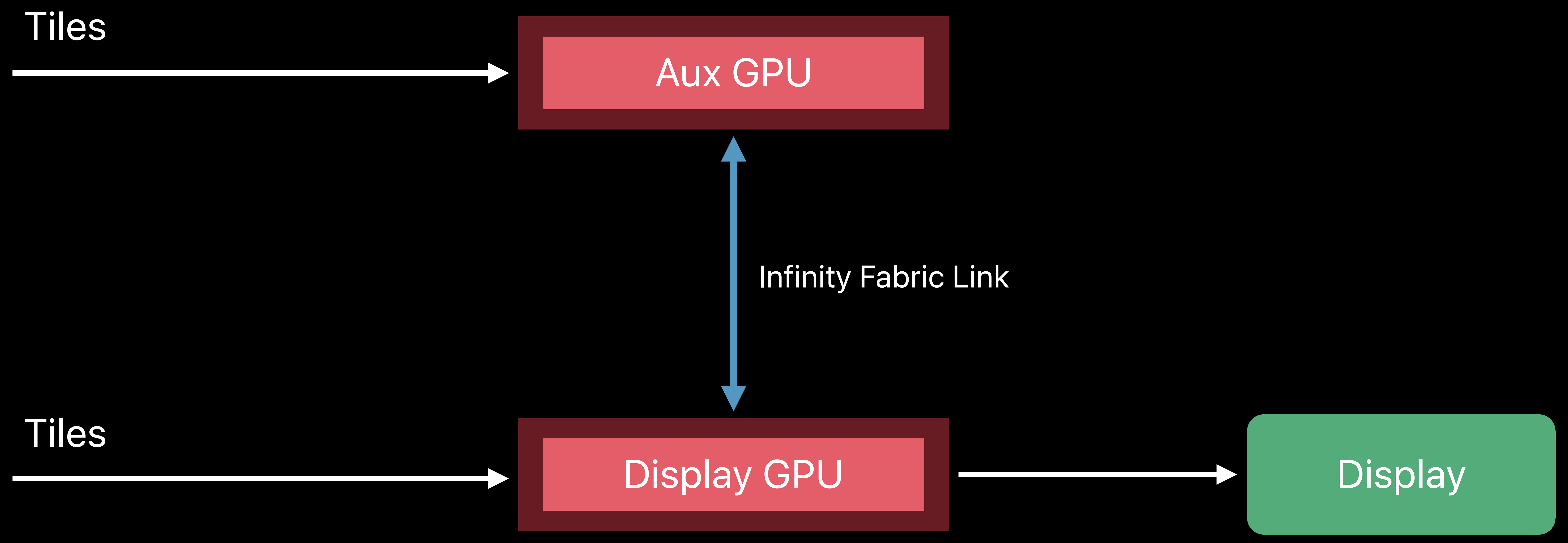
Infinity Fabric Transfer Strategies

Transfer entire frames



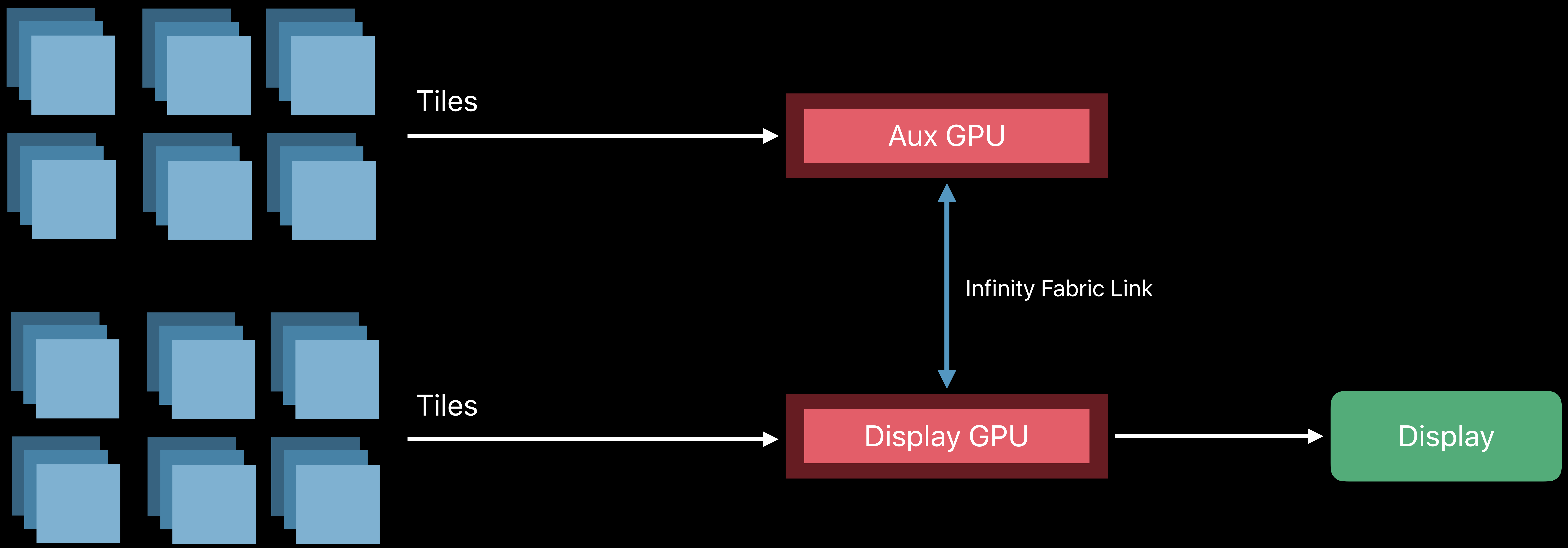
Infinity Fabric Transfer Strategies

Transfer tiles



Infinity Fabric Transfer Strategies

Transfer tiles



Final Cut Pro X

Efficient scaling on CPU and GPU

Making full use of Metal Peer Group API

More streams of 8K ProRes video

Single GPU PCIe Transfer

Single GPU PCIe Transfer

Single GPU PCIe Transfer

Single GPU PCIe Transfer

Single GPU PCIe Transfer

Single GPU PCIe Transfer

Single GPU PCIe Transfer

Dual GPU PCIe Transfer

Dual GPU PCIe Transfer

Dual GPU PCIe Transfer

Dual GPU PCIe Transfer

Dual GPU PCIe Transfer

Dual GPU PCIe Transfer

Dual GPU PCIe Transfer

Dual GPU PCIe Transfer

Dual GPU PCIe Transfer

Infinity Fabric Link

Dual GPU PCIe Transfer

Infinity Fabric Link

Dual GPU PCIe Transfer

Infinity Fabric Link

Dual GPU PCIe Transfer

Infinity Fabric Link

Dual GPU PCIe Transfer

Infinity Fabric Link


8K 29.97p, Stereo
Namunyak Kenya
44%
View
1:00:44:15
Index
Namunyak Kenya 01:14:04
01:00:00:00
01:00:10:00
01:00:20:00
01:00:30:00
01:00:40:00
01:00:50:00
01:01:00:00
01:01:10:00
01:01:20:00
Zebras at play
CANON8K_S001_S001_T050
D004_C001_20190501_PR00000-8K_PAD
Namunyak Title
Sunrise
Mountains
Zebras i...
Galloping
Giraffes in the bush
Broad smile
Kenya Time Lapse
C004_C021_0502EC_001_15_RS
Namunyak Score
Namunyak Score
Effects
4K Only
Installed Effects
All Video & Audio
VIDEO
All
360°
Basics
Blur
Color
Color Presets
Comic Looks
Custom
Distortion
Favorites
Color Corrections
Lens Flare
Search
2 items


8K 29.97p, Stereo
Namunyak Kenya
44%
View
1:00:44:15
Index
Namunyak Kenya 01:14:04
Zebras at play
CANON8K_S001_S001_T050
D004_C001_20190501_PR00000-8K_PAD
Namunyak Title
Sunrise
Mountains
Galloping
Giraffes in the bush
Broad smile
Kenya Time Lapse
C004_C021_0502EC_001_15_RS
Namunyak Score
Namunyak Score
Effects
4K Only
Installed Effects
All Video & Audio
VIDEO
All
360°
Basics
Blur
Color
Color Presets
Comic Looks
Custom
Distortion
Favorites
Color Corrections
Lens Flare
Search
2 items

Detecting Infinity Fabric Configurations

NEW

```
// Query for Infinity Fabric connections
let gpuPeerGroupID = device.peerGroupID
let gpuPeerIndex = device.peerIndex
let gpuPeerCount = device.peerCount
let gpuLocationNumber = device.locationNumber
```

	peerGroupID	peerIndex	peerCount	locationNumber
Linked GPUs, dual PCIe AMD Radeon Pro Vega II	Equal	(0..peerCount-1)	2	Different
Linked GPUs, shared PCIe AMD Radeon Pro Vega II Duo	Equal	(0..peerCount-1)	2	Equal

```swift
// Create shared event and command queues for auxiliary and display connected GPUs
let sharedEvent = deviceAux.makeSharedEvent()!
let renderTexture = deviceAux.makeTexture()!
let renderTextureView = renderTexture.makeRemoteTextureViewForDevice(deviceDisp)!

// Encode rendering of video frame on auxiliary device
let renderCommandBuffer = commandQueueAux.makeCommandBuffer()!
let renderCommandEncoder = renderCommandBuffer.makeRenderCommandEncoder()!
renderCommandEncoder.drawPrimitives()
renderCommandEncoder.endEncoding()
renderCommandBuffer.encodeSignalEvent(sharedEvent)

// Encode blit from auxiliary device to display device
let blitCommandBuffer = commandQueueDisp.makeCommandBuffer()!
let blitCommandEncoder = blitCommandBuffer.makeBlitCommandEncoder()!
blitCommandBuffer.encodeWaitEvent(sharedEvent)
blitCommandEncoder.copy(remoteTextureView, …)
blitCommandEncoder.endEncoding()
```

```swift
// Create shared event and command queues for auxiliary and display connected GPUs
let sharedEvent = deviceAux.makeSharedEvent()!
let renderTexture = deviceAux.makeTexture()!
let renderTextureView = renderTexture.makeRemoteTextureViewForDevice(deviceDisp)!

// Encode rendering of video frame on auxiliary device
let renderCommandBuffer = commandQueueAux.makeCommandBuffer()!
let renderCommandEncoder = renderCommandBuffer.makeRenderCommandEncoder()!
renderCommandEncoder.drawPrimitives()
renderCommandEncoder.endEncoding()
renderCommandBuffer.encodeSignalEvent(sharedEvent)

// Encode blit from auxiliary device to display device
let blitCommandBuffer = commandQueueDisp.makeCommandBuffer()!
let blitCommandEncoder = blitCommandBuffer.makeBlitCommandEncoder()!
blitCommandBuffer.encodeWaitEvent(sharedEvent)
blitCommandEncoder.copy(remoteTextureView, …)
blitCommandEncoder.endEncoding()
```

```swift
// Create shared event and command queues for auxiliary and display connected GPUs
let sharedEvent = deviceAux.makeSharedEvent()!
let renderTexture = deviceAux.makeTexture()!
let renderTextureView = renderTexture.makeRemoteTextureViewForDevice(deviceDisp)!

// Encode rendering of video frame on auxiliary device
let renderCommandBuffer = commandQueueAux.makeCommandBuffer()!
let renderCommandEncoder = renderCommandBuffer.makeRenderCommandEncoder()!
renderCommandEncoder.drawPrimitives()
renderCommandEncoder.endEncoding()
renderCommandBuffer.encodeSignalEvent(sharedEvent)

// Encode blit from auxiliary device to display device
let blitCommandBuffer = commandQueueDisp.makeCommandBuffer()!
let blitCommandEncoder = blitCommandBuffer.makeBlitCommandEncoder()!
blitCommandBuffer.encodeWaitEvent(sharedEvent)
blitCommandEncoder.copy(remoteTextureView, …)
blitCommandEncoder.endEncoding()
```

```swift
// Create shared event and command queues for auxiliary and display connected GPUs
let sharedEvent = deviceAux.makeSharedEvent()!
let renderTexture = deviceAux.makeTexture()!
let renderTextureView = renderTexture.makeRemoteTextureViewForDevice(deviceDisp)!

// Encode rendering of video frame on auxiliary device
let renderCommandBuffer = commandQueueAux.makeCommandBuffer()!
let renderCommandEncoder = renderCommandBuffer.makeRenderCommandEncoder()!
renderCommandEncoder.drawPrimitives()
renderCommandEncoder.endEncoding()
renderCommandBuffer.encodeSignalEvent(sharedEvent)

// Encode blit from auxiliary device to display device
let blitCommandBuffer = commandQueueDisp.makeCommandBuffer()!
let blitCommandEncoder = blitCommandBuffer.makeBlitCommandEncoder()!
blitCommandBuffer.encodeWaitEvent(sharedEvent)
blitCommandEncoder.copy(remoteTextureView, …)
blitCommandEncoder.endEncoding()
```

```swift
// Create shared event and command queues for auxiliary and display connected GPUs
let sharedEvent = deviceAux.makeSharedEvent()!
let renderTexture = deviceAux.makeTexture()!
let renderTextureView = renderTexture.makeRemoteTextureViewForDevice(deviceDisp)!

// Encode rendering of video frame on auxiliary device
let renderCommandBuffer = commandQueueAux.makeCommandBuffer()!
let renderCommandEncoder = renderCommandBuffer.makeRenderCommandEncoder()!
renderCommandEncoder.drawPrimitives()
renderCommandEncoder.endEncoding()
renderCommandBuffer.encodeSignalEvent(sharedEvent)

// Encode blit from auxiliary device to display device
let blitCommandBuffer = commandQueueDisp.makeCommandBuffer()!
let blitCommandEncoder = blitCommandBuffer.makeBlitCommandEncoder()!
blitCommandBuffer.encodeWaitEvent(sharedEvent)
blitCommandEncoder.copy(remoteTextureView, …)
blitCommandEncoder.endEncoding()
```

Metal Peer Group API

Transfer between GPUs at high speed

Take advantage of parallel channel

Reduce PCIe traffic

Enable challenging workflows

AFFINITY

Photo


Affinity Photo Beta - Rabbit [Locked] [Modified] (200.0%)
Ungroup
Histogram
Colour
Swatches
Brushes
All Channels
Layer
Marquee
Mean: 27.59
Std. Dev.: 36.06
Median: 15
Pixels: 466944
Level: -
Pixels: -
Percentile: -
Min: 0
Max: 1
Layers
Adj
FX
Sty
Stock
Dbg
Opacity: 100 %
Passthrough
(Vibrance Adjustment)
Lighting (Pixel)
(Colour Balance Adjustment)
(Shadows / Highlights Ac
Audience Ears (Group)
Confetti (Group)
Sparks (Group)
Doves and Cage (Group)
Hat (Group)
Table (Group)
Smoke (Pixel)
Smoke (Pixel)
Rabbit (Group)
Stage Shadow (Pixel)
Smoke (Group)
Confetti (Group)
Smoke (Pixel)
Curtains (Pixel)
Curtain Shadow (Pixel)
Xfm
His
Chn
Nvg
32-bit Preview
Extended Dynamic Range
Enable EDR
Show EDR Clipping
Clip to maximum
Preview Exposure


Affinity Photo Beta - Rabbit [Locked] [Modified] (200.0%)
Ungroup
Histogram | Colour | Swatches | Brushes
All Channels
Layer
Marquee
Mean: 27.59
Std. Dev.: 36.06
Median: 15
Pixels: 466944
Level: -
Pixels: -
Percentile: -
Min: 0
Max: 1
Layers | Adj | FX | Sty | Stock | Dbg
Opacity: 100 %
Passthrough
(Vibrance Adjustment)
Lighting (Pixel)
(Colour Balance Adjustment)
(Shadows / Highlights Ac
Audience Ears (Group)
Confetti (Group)
Sparks (Group)
Doves and Cage (Group)
Hat (Group)
Table (Group)
Smoke (Pixel)
Smoke (Pixel)
Rabbit (Group)
Stage Shadow (Pixel)
Smoke (Group)
Confetti (Group)
Smoke (Pixel)
Curtains (Pixel)
Curtain Shadow (Pixel)
Xfm | His | Chn | Nvg | 32-bit Preview
Extended Dynamic Range
Enable EDR
Show EDR Clipping
Clip to maximum
Preview Exposure


Affinity Photo Beta File Edit Text Document Layer Select Arrange Filters View Window Help
Tue 14:42
Affinity Photo Beta - Rabbit [Locked] [Modified] (200.0%)
Wheel (Group) Fill: Stroke: None Ungroup
18 Core CPU
Histogram Colour Swatches Brushes
All Channels Layer Marquee
Mean: 83.44
Std. Dev.: 103.51
Median: 22
Pixels: 466944
Min: 0 Max: 1
Layers Adj FX Sty Stock Dbg
Opacity: 100 % Passthrough
Hat (Group)
Table (Group)
Smoke (Pixel)
Smoke (Pixel)
Rabbit (Group)
Stage Shadow (Pixel)
Smoke (Group)
Confetti (Group)
Smoke (Pixel)
Curtains (Pixel)
Curtain Shadow (Pixel)
Suitcases (Pixel)
Suitcase (Group)
Wheel (Group)
Lighting (Pixel)
Rabbit Shadow (Group)
Stage (Group)
Backdrop (Pixel)
Sketch (Pixel)
Xfm His Chn Nvg 32-bit Preview
Extended Dynamic Range
Enable EDR Show EDR Clipping
Clip to maximum
Preview Exposure
Preview Gamma
Display Transform
ICC Display Transform
Unmanaged
OCIO Display Transform
'Wheel' selected. Drag to move selection. Click another object to select it. Click on an empty area to deselect selection. ⇧ to constrain. ⌘ to clone selected objects. ⌥ to ignore snapping.


Affinity Photo Beta File Edit Text Document Layer Select Arrange Filters View Window Help
Tue 14:42
Affinity Photo Beta - Rabbit [Locked] [Modified] (200.0%)
Wheel (Group) Fill: Stroke: None Ungroup
Histogram Colour Swatches Brushes
All Channels Layer Marquee
Mean: 83.44
Std. Dev.: 103.51
Median: 22
Pixels: 466944
Min: 0 Max: 1
Layers Adj FX Sty Stock Dbg
Opacity: 100 % Passthrough
Hat (Group)
Table (Group)
Smoke (Pixel)
Smoke (Pixel)
Rabbit (Group)
Stage Shadow (Pixel)
Smoke (Group)
Confetti (Group)
Smoke (Pixel)
Curtains (Pixel)
Curtain Shadow (Pixel)
Suitcases (Pixel)
Suitcase (Group)
Wheel (Group)
Lighting (Pixel)
Rabbit Shadow (Group)
Stage (Group)
Backdrop (Pixel)
Sketch (Pixel)
Xfm His Chn Nvg 32-bit Preview
Extended Dynamic Range
Enable EDR Show EDR Clipping
Clip to maximum
Preview Exposure
Preview Gamma
Display Transform
ICC Display Transform
Unmanaged
OCIO Display Transform
'Wheel' selected. Drag to move selection. Click another object to select it. Click on an empty area to deselect selection. ⇧ to constrain. ⌘ to clone selected objects. ⌥ to ignore snapping.


Affinity Photo Beta - Rabbit [Locked] [Modified] (200.0%)
Wheel (Group)
Fill:
Stroke:
None
Ungroup
GPU
0.00 FPS
Telemetry...
Manual
AMD Radeon RX Vega 64
Assignments: 0
Allocated buffers: 1082
Allocated memory (expected): 1750.36MB
Allocated memory (reported): 1053.80MB
Kernel cache items: 45
Pipelines (compute): 0
Pipelines (demoted): 0
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
AMD Radeon RX Vega 64
Assignments: 0
Allocated buffers: 727
Allocated memory (expected): 1715.59MB
Allocated memory (reported): 1488.54MB
Kernel cache items: 45
Pipelines (compute): 0
Pipelines (demoted): 0
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
AMD Radeon Pro Vega 56 (Root)
Assignments: 6
Allocated buffers: 1777
Allocated memory (expected): 2458.22MB
Allocated memory (reported): 3132.32MB
Kernel cache items: 109
Pipelines (compute): 0
Pipelines (demoted): 0
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
AMD Radeon Pro 580
Assignments: 0
Allocated buffers: 1042
Allocated memory (expected): 1696.29MB
Allocated memory (reported): 1427.21MB
Kernel cache items: 45
Pipelines (compute): 0
Pipelines (demoted): 0
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
1 External GPU

Affinity Photo Beta - Rabbit [Locked] [Modified] (200.0%)
Wheel (Group)
Fill:
Stroke:
None
Ungroup
GPU
0.00 FPS
Telemetry...
Manual
AMD Radeon RX Vega 64
Assignments: 0
Allocated buffers: 1082
Allocated memory (expected): 1750.36MB
Allocated memory (reported): 1053.80MB
Kernel cache items: 45
Pipelines (compute): 0
Pipelines (demoted): 0
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
AMD Radeon RX Vega 64
Assignments: 0
Allocated buffers: 727
Allocated memory (expected): 1715.59MB
Allocated memory (reported): 1488.54MB
Kernel cache items: 45
Pipelines (compute): 0
Pipelines (demoted): 0
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
AMD Radeon Pro Vega 56 (Root)
Assignments: 6
Allocated buffers: 1777
Allocated memory (expected): 2458.22MB
Allocated memory (reported): 3132.32MB
Kernel cache items: 109
Pipelines (compute): 0
Pipelines (demoted): 0
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
AMD Radeon Pro 580
Assignments: 0
Allocated buffers: 1042
Allocated memory (expected): 1696.29MB
Allocated memory (reported): 1427.21MB
Kernel cache items: 45
Pipelines (compute): 0
Pipelines (demoted): 0
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB

Affinity Photo Beta File Edit Text Document Layer Select Arrange Filters View Window Help
Tue 14:44
Affinity Photo Beta - Rabbit [Locked] [Modified] (200.0%)
Wheel (Group) Fill: Stroke: None Ungroup
GPU
24.10 FPS Telemetry... Manual
AMD Radeon RX Vega 64
Assignments: 2
Allocated buffers: 1110
Allocated memory (expected): 1770.74MB
Allocated memory (reported): 1637.53MB
Kernel cache items: 45
Pipelines (compute): 1101
Pipelines (demoted): 252
Upstream bytes: 0.00MB
Downstream bytes: 40.00MB
AMD Radeon RX Vega 64
Assignments: 2
Allocated buffers: 755
Allocated memory (expected): 1873.67MB
Allocated memory (reported): 1901.88MB
Kernel cache items: 45
Pipelines (compute): 832
Pipelines (demoted): 183
Upstream bytes: 0.00MB
Downstream bytes: 40.00MB
AMD Radeon Pro Vega 56 (Root)
Assignments: 1
Allocated buffers: 1747
Allocated memory (expected): 2172.50MB
Allocated memory (reported): 3180.04MB
Kernel cache items: 109
Pipelines (compute): 735
Pipelines (demoted): 163
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
AMD Radeon Pro 580
Assignments: 1
Allocated buffers: 1047
Allocated memory (expected): 1696.29MB
Allocated memory (reported): 1755.74MB
Kernel cache items: 45
Pipelines (compute): 351
Pipelines (demoted): 86
Upstream bytes: 0.00MB
Downstream bytes: 4.00MB
4 External GPUs
Histogram Colour Swatches Brushes
All Channels Layer Marquee
Mean: 27.59
Std. Dev.: 36.06
Median: 15
Pixels: 466944
Layers Adj FX Sty Stock Dbg
Opacity: 100 % Passthrough
(Vibrance Adjustment)
Lighting (Pixel)
(Colour Balance Adjustment)
(Shadows / Highlights Ad
Audience Ears (Group)
Confetti (Group)
Sparks (Group)
Doves and Cage (Group)
Hat (Group)
Table (Group)
Smoke (Pixel)
Smoke (Pixel)
Rabbit (Group)
Stage Shadow (Pixel)
Smoke (Group)
Confetti (Group)
Smoke (Pixel)
Curtains (Pixel)
Curtain Shadow (Pixel)
Xfm His Chn Nvg 32-bit Preview
Extended Dynamic Range
Enable EDR Show EDR Clipping
Clip to maximum
Preview Exposure
Preview Gamma
Display Transform
ICC Display Transform
Unmanaged
OCIO Display Transform
'Wheel' selected. Drag to move selection. Click another object to select it. Click on an empty area to deselect selection. ⇧ to constrain. ⌘ to clone selected objects. ⌥ to ignore snapping.


Affinity Photo Beta File Edit Text Document Layer Select Arrange Filters View Window Help
Tue 14:44
Affinity Photo Beta - Rabbit [Locked] [Modified] (200.0%)
Wheel (Group) Fill: Stroke: None Ungroup
GPU
24.10 FPS Telemetry... Manual
AMD Radeon RX Vega 64
Assignments: 2
Allocated buffers: 1110
Allocated memory (expected): 1770.74MB
Allocated memory (reported): 1637.53MB
Kernel cache items: 45
Pipelines (compute): 1101
Pipelines (demoted): 252
Upstream bytes: 0.00MB
Downstream bytes: 40.00MB
AMD Radeon RX Vega 64
Assignments: 2
Allocated buffers: 755
Allocated memory (expected): 1873.67MB
Allocated memory (reported): 1901.88MB
Kernel cache items: 45
Pipelines (compute): 832
Pipelines (demoted): 183
Upstream bytes: 0.00MB
Downstream bytes: 40.00MB
AMD Radeon Pro Vega 56 (Root)
Assignments: 1
Allocated buffers: 1747
Allocated memory (expected): 2172.50MB
Allocated memory (reported): 3180.04MB
Kernel cache items: 109
Pipelines (compute): 735
Pipelines (demoted): 163
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
AMD Radeon Pro 580
Assignments: 1
Allocated buffers: 1047
Allocated memory (expected): 1696.29MB
Allocated memory (reported): 1755.74MB
Kernel cache items: 45
Pipelines (compute): 351
Pipelines (demoted): 86
Upstream bytes: 0.00MB
Downstream bytes: 4.00MB
Histogram Colour Swatches Brushes
All Channels Layer Marquee
Mean: 27.59
Std. Dev.: 36.06
Median: 15
Pixels: 466944
Min: 0 Max: 1
Layers Adj FX Sty Stock Dbg
Opacity: 100 % Passthrough
(Vibrance Adjustment)
Lighting (Pixel)
(Colour Balance Adjustment)
(Shadows / Highlights Ad
Audience Ears (Group)
Confetti (Group)
Sparks (Group)
Doves and Cage (Group)
Hat (Group)
Table (Group)
Smoke (Pixel)
Smoke (Pixel)
Rabbit (Group)
Stage Shadow (Pixel)
Smoke (Group)
Confetti (Group)
Smoke (Pixel)
Curtains (Pixel)
Curtain Shadow (Pixel)
Xfm His Chn Nvg 32-bit Preview
Extended Dynamic Range
Enable EDR Show EDR Clipping
Clip to maximum
Preview Exposure
Preview Gamma
Display Transform
ICC Display Transform
Unmanaged
OCIO Display Transform
'Wheel' selected. Drag to move selection. Click another object to select it. Click on an empty area to deselect selection. ⇧ to constrain. ⌘ to clone selected objects. ⌥ to ignore snapping.


Affinity Photo Beta File Edit Text Document Layer Select Arrange Filters View Window Help
Tue 14:25
Affinity Photo Beta - bike.jpg (189.5%)
18 Core CPU
SHOWA
elf
HONDA
Histogram Colour Swatches Brushes
All Channels Layer Marquee
Mean: 133.92
Std. Dev.: 51.94
Median: 136
Pixels: 385452
Level: -
Pixels: -
Percentile: -
Min: 0 Max: 1
Layers Adj FX Sty Stock Dbg
Opacity: 100 % Normal
Background (Pixel)
Xfm His Chn Nvg 32-bit Preview
Extended Dynamic Range
Enable EDR Show EDR Clipping
Clip to maximum
Preview Exposure
Preview Gamma
Display Transform
ICC Display Transform
Unmanaged
OCIO Display Transform
Depth of Field
Mode Elliptical
Radius 25.9 px
Vibrance 0 %
Clarity 0 px
Cancel Apply
Drag handles to position blur.


Affinity Photo Beta File Edit Text Document Layer Select Arrange Filters View Window Help
Tue 14:25
Affinity Photo Beta - bike.jpg (189.5%)
Histogram Colour Swatches Brushes
All Channels
Layer
Marquee
Mean: 133.92
Std. Dev.: 51.94
Median: 136
Pixels: 385452
Level: -
Pixels: -
Percentile: -
Min: 0
Max: 1
Layers Adj FX Sty Stock Dbg
Opacity: 100 %
Normal
Background (Pixel)
Xfm His Chn Nvg 32-bit Preview
Extended Dynamic Range
Enable EDR
Show EDR Clipping
Clip to maximum
Preview Exposure
0
Preview Gamma
1
Display Transform
ICC Display Transform
Unmanaged
OCIO Display Transform
Depth of Field
Mode Elliptical
Radius
25.9 px
Vibrance
0 %
Clarity
0 px
Cancel
Apply
Drag handles to position blur.


Affinity Photo Beta File Edit Text Document Layer Select Arrange Filters View Window Help
Tue 14:31
Affinity Photo Beta - bike.jpg (200.0%)
GPU
40.66 FPS
Telemetry...
Manual
AMD Radeon RX Vega 64
Assignments: 2
Allocated buffers: 15
Allocated memory (expected): 385.57MB
Allocated memory (reported): 459.64MB
Kernel cache items: 5
Pipelines (compute): 12
Pipelines (demoted): 4
Upstream bytes: 0.00MB
Downstream bytes: 8.00MB
AMD Radeon RX Vega 64
Assignments: 2
Allocated buffers: 12
Allocated memory (expected): 311.80MB
Allocated memory (reported): 274.56MB
Kernel cache items: 5
Pipelines (compute): 12
Pipelines (demoted): 4
Upstream bytes: 0.00MB
Downstream bytes: 24.00MB
AMD Radeon Pro Vega 56 (Root)
Assignments: 1
Allocated buffers: 28
Allocated memory (expected): 530.45MB
Allocated memory (reported): 1468.16MB
Kernel cache items: 69
Pipelines (compute): 6
Pipelines (demoted): 2
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
AMD Radeon Pro 580
Assignments: 1
Allocated buffers: 7
Allocated memory (expected): 165.73MB
Allocated memory (reported): 182.66MB
Kernel cache items: 4
Pipelines (compute): 6
Pipelines (demoted): 2
Upstream bytes: 0.00MB
Downstream bytes: 4.00MB
4 External GPUs
GPU History
AMD Radeon RX
AMD Radeon RX
AMD Radeon Pro
AMD Radeon Pro 580
Histogram Colour Swatches Brushes
All Channels
Layer
Marquee
Mean: 134.97
Std. Dev.: 46.06
Median: 134
Pixels: 385452
Min: 0
Max: 1
Layers Adj FX Sty Stock Dbg
Opacity: 100 %
Normal
Background (Pixel)
Xfm His Chn Nvg 32-bit Preview
Extended Dynamic Range
Enable EDR
Show EDR Clipping
Clip to maximum
Preview Exposure
Preview Gamma
Display Transform
ICC Display Transform
Unmanaged
OCIO Display Transform
Depth of Field
Mode Elliptical
Radius
100 px
Vibrance
0 %
Clarity
0 px
Cancel
Apply
Drag handles to position blur.

Affinity Photo Beta File Edit Text Document Layer Select Arrange Filters View Window Help
Tue 14:31
Affinity Photo Beta - bike.jpg (200.0%)
GPU
40.66 FPS
Telemetry...
Manual
AMD Radeon RX Vega 64
Assignments: 2
Allocated buffers: 15
Allocated memory (expected): 385.57MB
Allocated memory (reported): 459.64MB
Kernel cache items: 5
Pipelines (compute): 12
Pipelines (demoted): 4
Upstream bytes: 0.00MB
Downstream bytes: 8.00MB
AMD Radeon RX Vega 64
Assignments: 2
Allocated buffers: 12
Allocated memory (expected): 311.80MB
Allocated memory (reported): 274.56MB
Kernel cache items: 5
Pipelines (compute): 12
Pipelines (demoted): 4
Upstream bytes: 0.00MB
Downstream bytes: 24.00MB
AMD Radeon Pro Vega 56 (Root)
Assignments: 1
Allocated buffers: 28
Allocated memory (expected): 530.45MB
Allocated memory (reported): 1468.16MB
Kernel cache items: 69
Pipelines (compute): 6
Pipelines (demoted): 2
Upstream bytes: 0.00MB
Downstream bytes: 0.00MB
AMD Radeon Pro 580
Assignments: 1
Allocated buffers: 7
Allocated memory (expected): 165.73MB
Allocated memory (reported): 182.66MB
Kernel cache items: 4
Pipelines (compute): 6
Pipelines (demoted): 2
Upstream bytes: 0.00MB
Downstream bytes: 4.00MB
GPU History
AMD Radeon RX
AMD Radeon RX
AMD Radeon Pro
AMD Radeon Pro 580
Histogram Colour Swatches Brushes
All Channels
Layer
Marquee
Mean: 134.97
Std. Dev.: 46.06
Median: 134
Pixels: 385452
Level: -
Pixels: -
Percentile: -
Min: 0
Max: 1
Layers Adj FX Sty Stock Dbg
Opacity: 100 %
Normal
Background (Pixel)
Xfm His Chn Nvg 32-bit Preview
Extended Dynamic Range
Enable EDR
Show EDR Clipping
Clip to maximum
Preview Exposure
0
Preview Gamma
1
Display Transform
ICC Display Transform
Unmanaged
OCIO Display Transform
Depth of Field
Mode Elliptical
Radius
100 px
Vibrance
0 %
Clarity
0 px
Cancel
Apply
Drag handles to position blur.

Metal for Pro Apps

Takeaways

Metal for Pro Apps

Takeaways

Optimize your apps for 8K content

Metal for Pro Apps

Takeaways

Optimize your apps for 8K content

Support HDR TVs and displays

Metal for Pro Apps

Takeaways

Optimize your apps for 8K content

Support HDR TVs and displays

Scale performance based on platform resources

Metal for Pro Apps

Takeaways

Optimize your apps for 8K content

Support HDR TVs and displays

Scale performance based on platform resources

Leverage Infinity Fabric Link with Metal Peer Group API

More Information

developer.apple.com/wwdc19/608

Metal for Pro Apps Lab	Thursday, 9:00
Metal Lab	Friday, 9:00
Metal for Machine Learning and Ray Tracing Lab	Friday, 12:00

WWDC19