RADEON RX

Explicit DirectX12 Multi GPU rendering

Presented by Raul Aguaviva (AMD) & Dan Baker (Oxide Games)

AGENDA

- Intro/Motivation
- API introduction
- Adding MGPU support (AFR)
- Tools of the trade
- Case study, Ashes Of The Singularity

  By Dan Baker

WHAT IS EXPLICIT MGPU?

- 2+ GPUs
- Sharing a bus (16GB/s)
- Synchronization up to the user

WHAT IS EXPLICIT MGPU?

- 800 GPUs
- Sharing a bus (16GB/s bidirectional)
- Synchronization up to the user

WHAT IS EXPLICIT MGPU?

- 800 GPUs
- Sharing a bus (IMb/s
- Synchronization up to the user

YOU ARE ALREADY DOING IT ☺

MGPU IN DX11

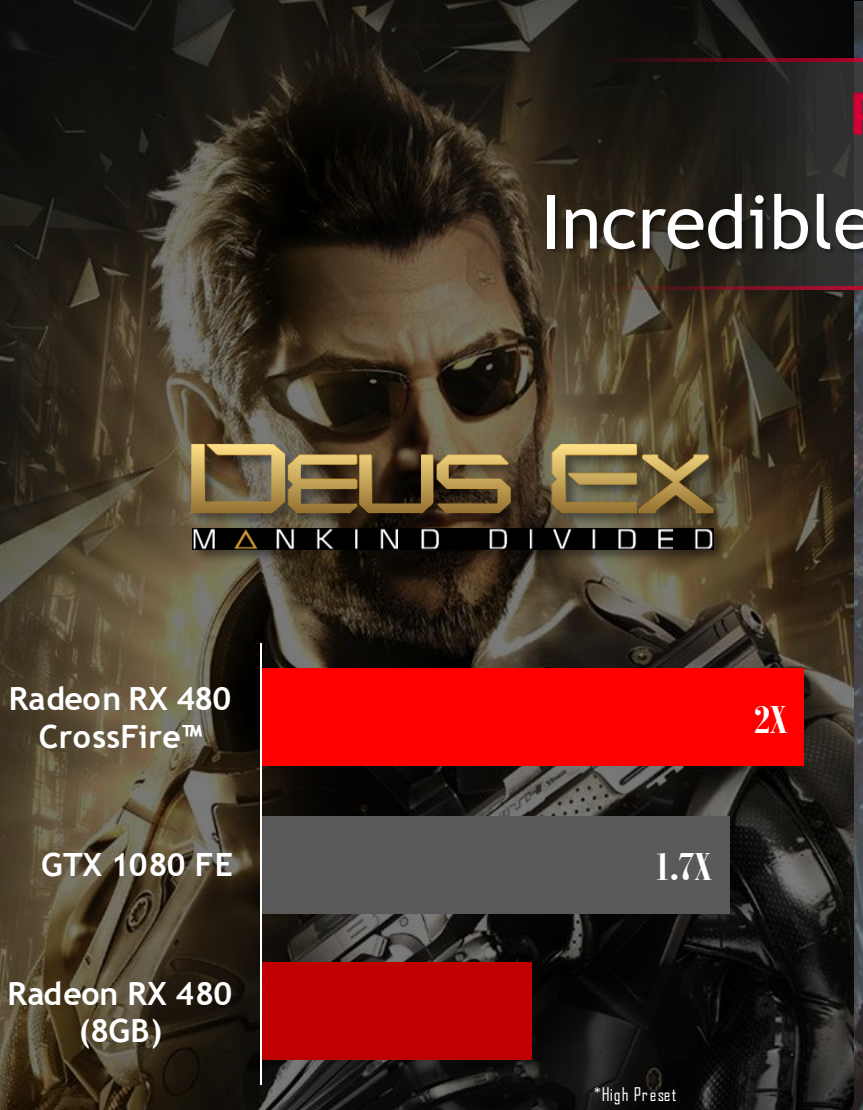
- Black box
- Heavy weight API (and driver)
- DX11 is single threaded

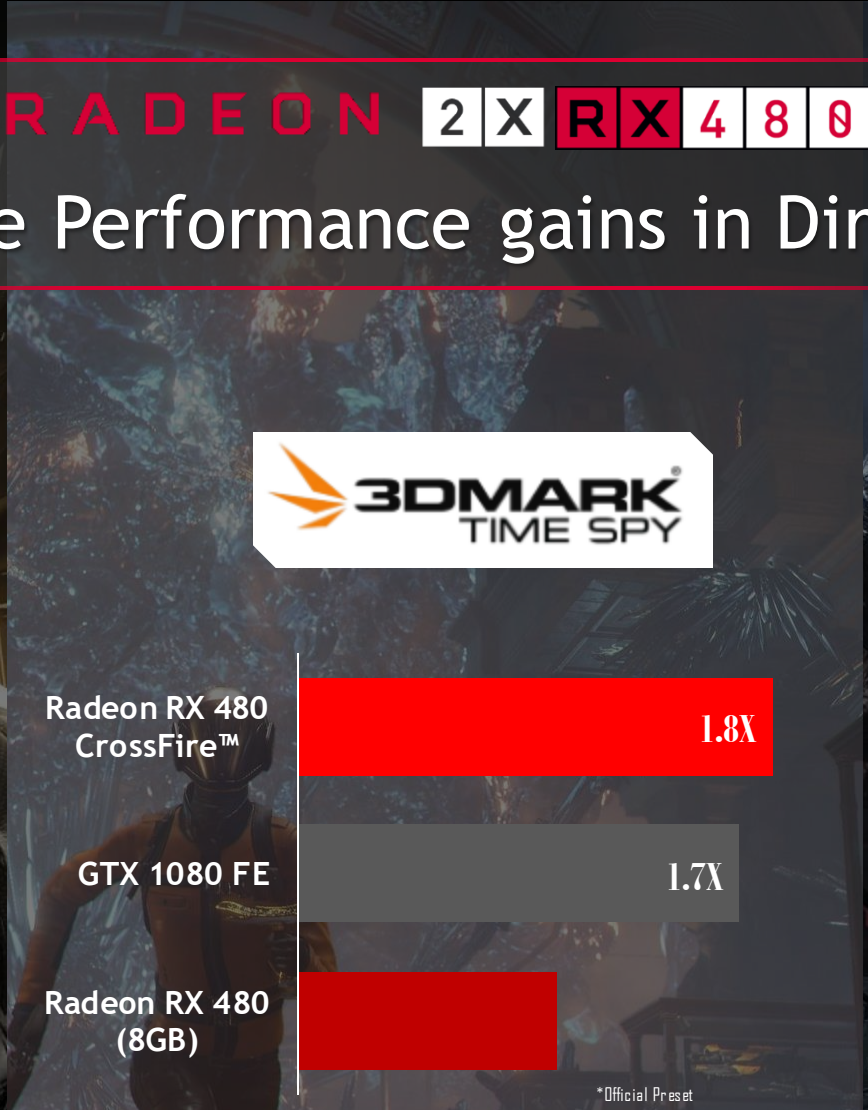
MGPU IN DX12 – WHY NOW?

- Explicit = full control
- Lightweight API
- Multithreading
- Needed in VR for one-GPU-per-eye configurations

RADEON 2X RX 480

Incredible Performance gains in DirectX® 12





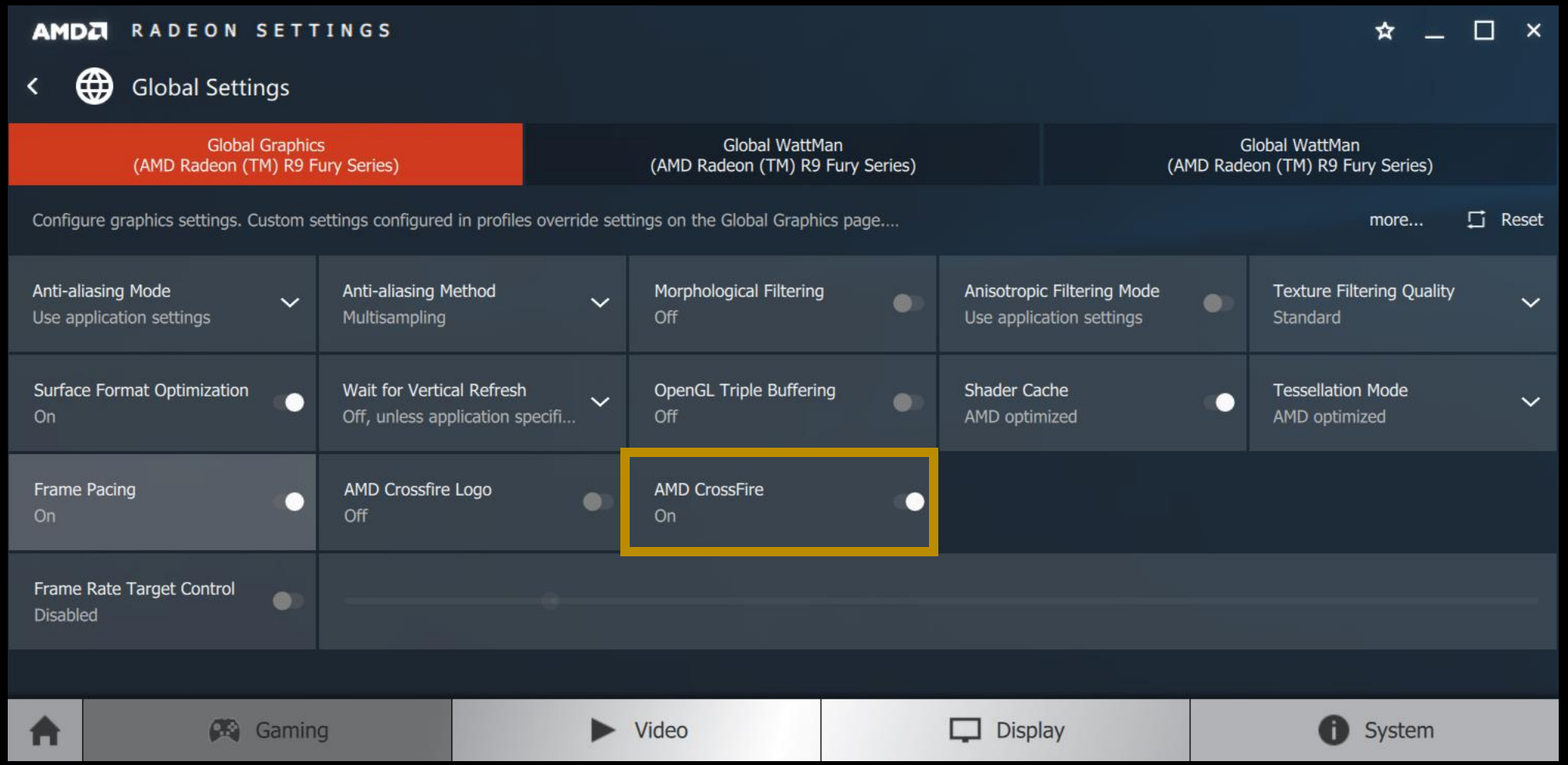
AMD | RADEON

* Tested at 1440p. See Endnotes for details. Use of third party marks/products is for informational purposes and no endorsement of or by AMD is

MGPU MODES

- Linked
  - Identical GPUs,
  - Enable in control panel
  - Optimal sharing of resources
- Unlinked
  - Different type and vendor GPUs
  - Resources shared through the system memory

ENABLING LINKED MODE



MGPU THE EASY WAY

- Typical rendering loop:

```
1  | Engine myengine1;
2  |
3  | myengine1.LoadResources(device);
4  |
5  |
6  | for(;;)
7  | {
8  |     myengine1.Render();
9  |     Present();
10 |
11 |
12 | }
```

MGPU THE EASY WAY

- Typical rendering loop:

```
1  | Engine myengine1, myengine2;
2  |
3  | myengine1.LoadResources(device1);
4  | myengine2.LoadResources(device2);
5  |
6  | for(;;)
7  | {
8  |     myengine1.Render();
9  |     present();
10 |     myengine2.Render();
11 |     present();
12 | }
```

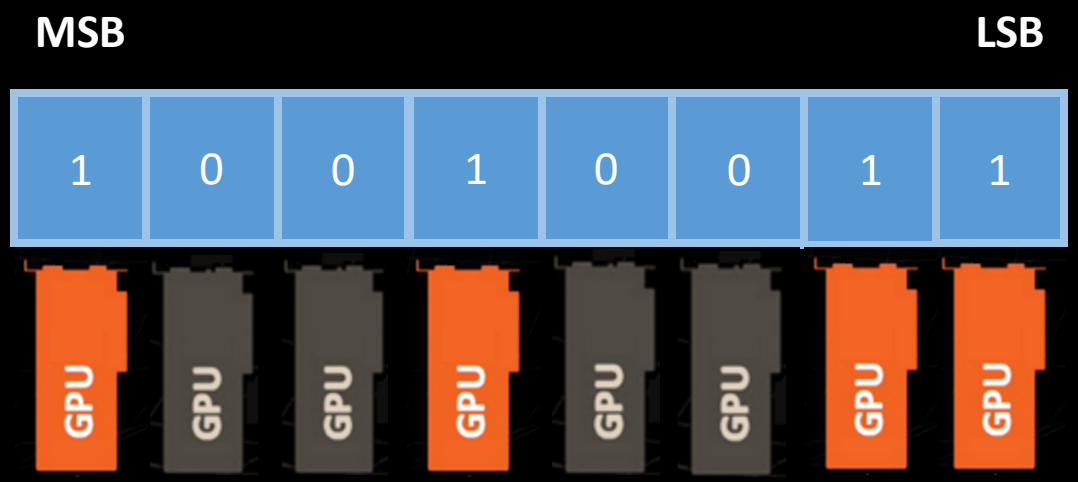
ADDRESSING GPUS

- Typical rendering loop:

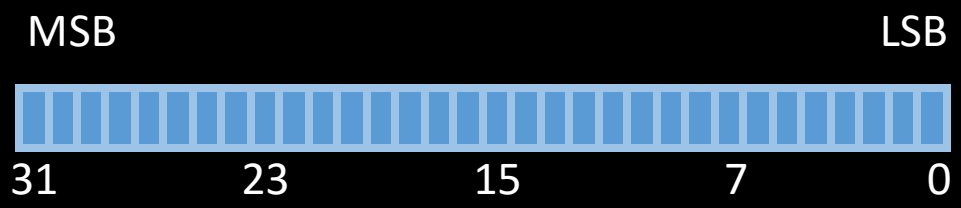
```
1  | Engine myengine1, myengine2;
2  |
3  | myengine1.LoadResources( node1 );
4  | myengine2.LoadResources( node2 );
5  |
6  | for(;;)
7  | {
8  |     myengine1.Render();
9  |     present();
10 |     myengine2.Render();
11 |     present();
12 | }
```

NODE MASK

- Each element of the bit field selects a GPU



NODE MASK



NOTICING THE NODE

```
CreateCommandList( UINT nodeMask,
                   D3D12_COMMAND_LIST_TYPE type,
                   ID3D12CommandAllocator  *pCommandAllocator,
                   ID3D12PipelineState     *pInitialState,
                   REFIID                  riid,
                   void                    **ppCommandList);
```

NOTICING THE NODE

```
HRESULT CreateCommandQueue(…);

struct D3D12_COMMAND_QUEUE_DESC
{
    D3D12_COMMAND_LIST_TYPE     Type;
    INT                         Priority;
    D3D12_COMMAND_QUEUE_FLAGS   Flags;
    UINT                        NodeMask;
};
```

NOTICING THE NODE

```
CreateRootSignature( UINT nodeMask,
                     const void      *pBlobWithRootSignature,
                     SIZE_T          blobLengthInBytes,
                     REFIID          riid,
                     void            **ppvRootSignature);
```

NOTICING THE NODE

```
HRESULT CreateDescriptorHeap (…);

struct D3D12_DESCRIPTOR_HEAP_DESC
{
        D3D12_DESCRIPTOR_HEAP_TYPE    Type;
        UINT                          NumDescriptors;
        D3D12_DESCRIPTOR_HEAP_FLAGS   Flags;
        UINT                          NodeMask;
};
```

NOTICING THE NODE

```
HRESULT CreateGraphicsPipelineState (…);

struct D3D12_GRAPHICS_PIPELINE_STATE_DESC
{
    D3D12_SHADER_BYTECODE               VS;
    D3D12_SHADER_BYTECODE               PS;
    […]
    DXGI_SAMPLE_DESC                    SampleDesc;
    UINT                                NodeMask;
    D3D12_CACHED_PIPELINE_STATE         CachedPSO;
    D3D12_PIPELINE_STATE_FLAGS          Flags;
};
```

NOTICING THE NODE

- Shared resource

```
pDevice->CreateCommittedResource(
    &CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_DEFAULT, CreationNode, VisibleNode),
    D3D12_HEAP_FLAG_NONE,
    &RDesc,
    D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE,
    nullptr,
    IID_PPV_ARGS(&m_pTexture2D));
```

NOTICING THE NODE

- Single node:

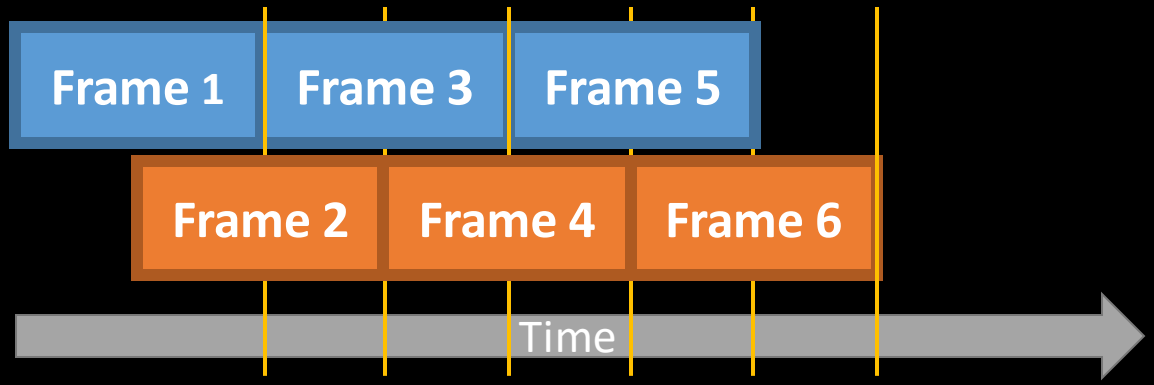

```
CreateCommandQueue( D3D12_COMMAND_QUEUE_DESC *,…);
CreateCommandList ( UINT nodeMask,…);
CreateDescriptorHeap( D3D12_DESCRIPTOR_HEAP_DESC ,…);
CreateQueryHeap( D3D12_QUERY_HEAP_DESC *,…);
```

- Multi-node:


```
CreateGraphicsPipelineState ( D3D12_GRAPHICS_PIPELINE_STATE_DESC *,…);
CreateComputePipelineState( D3D12_COMPUTE_PIPELINE_STATE_DESC * ,…);
CreateRootSignature( UINT nodeMask,… );
CreateCommandSignature( D3D12_COMMAND_SIGNATURE_DESC *,…);
CreateCommitedResource ( D3D12_HEAP_PROPERTIES *,…);
```

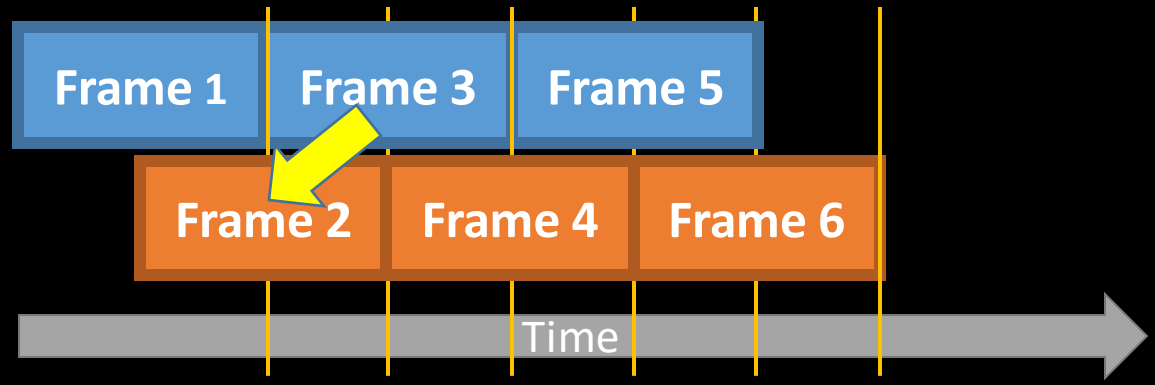
MGPU THE EASY WAY – BOTTOM LINE

- Use AFR
- Avoid frame dependencies



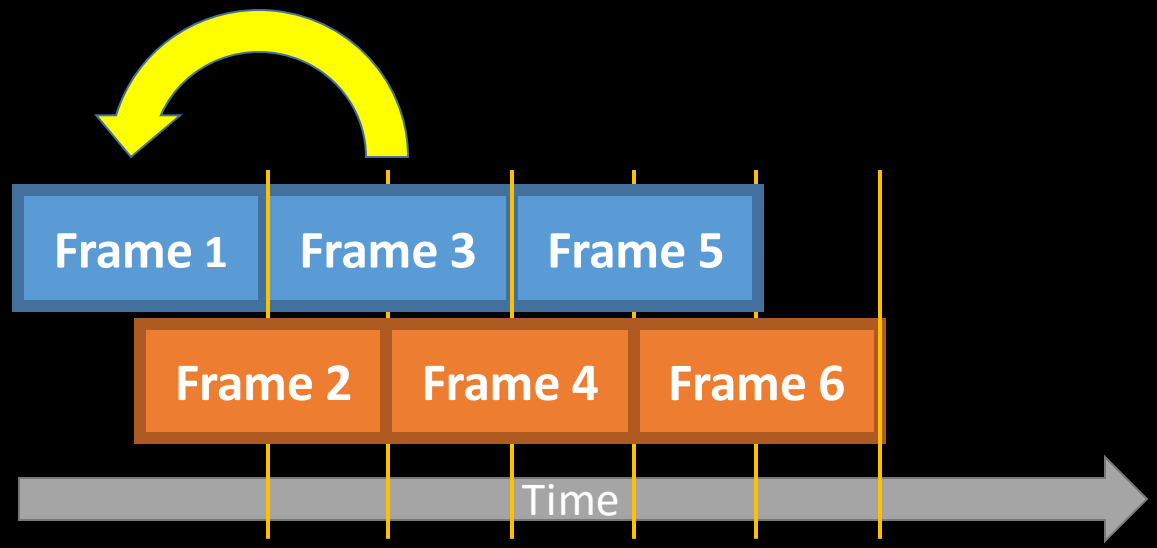
FRAME DEPENDENCIES

- Temporal effects
  - Motion blur
  - Antialiasing
  - Reprojection
  - Particle state

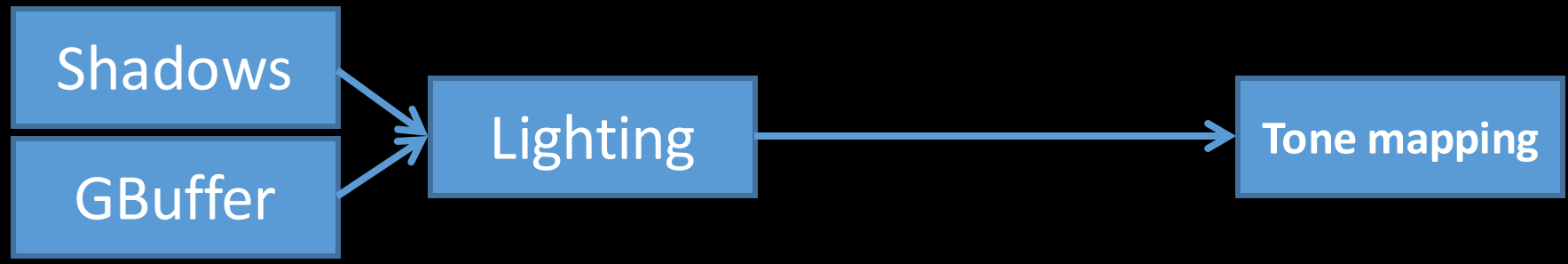


FRAME DEPENDENCIES

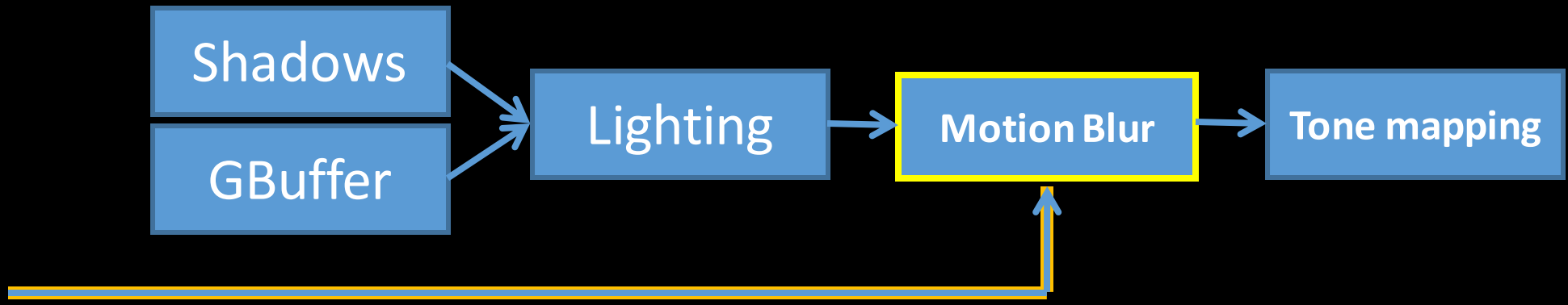
- Temporal effects
  - Motion blur
  - Antialiasing
  - Reprojection
  - Particle state



ADDING MGPU SUPPORT

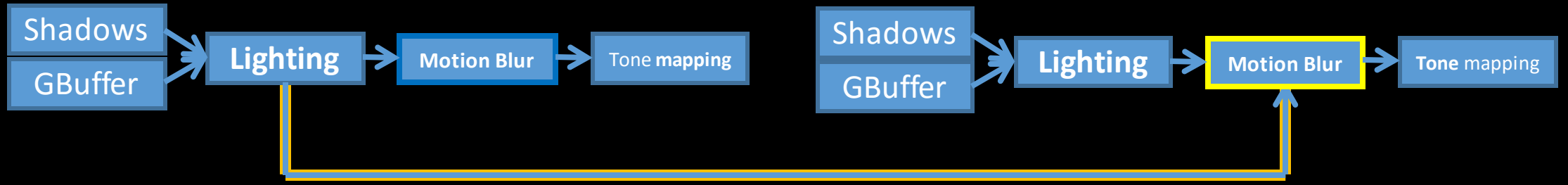


ADDING MGPU SUPPORT

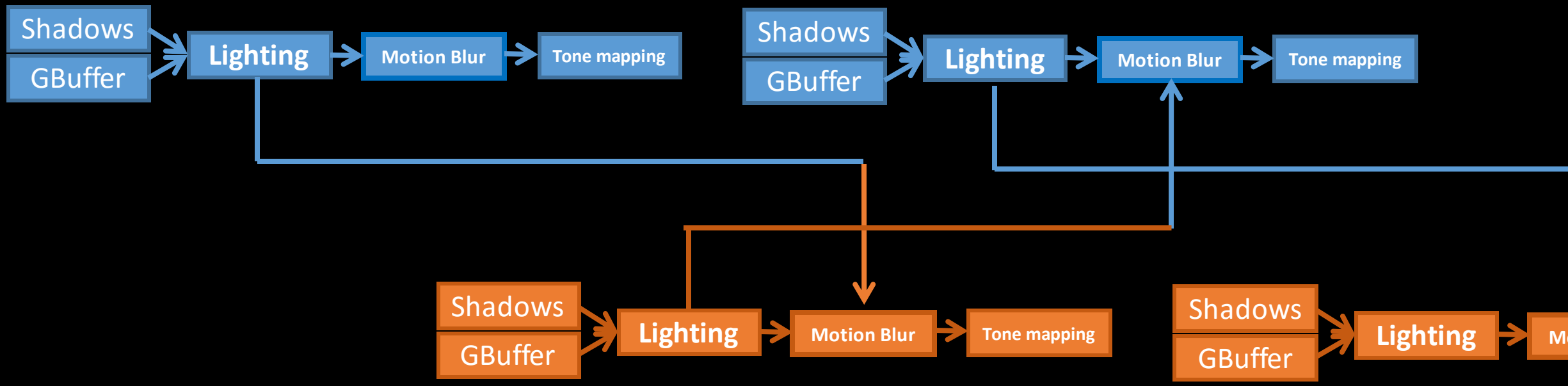


ADDING MGPU SUPPORT

- Retrieving the previous frame:

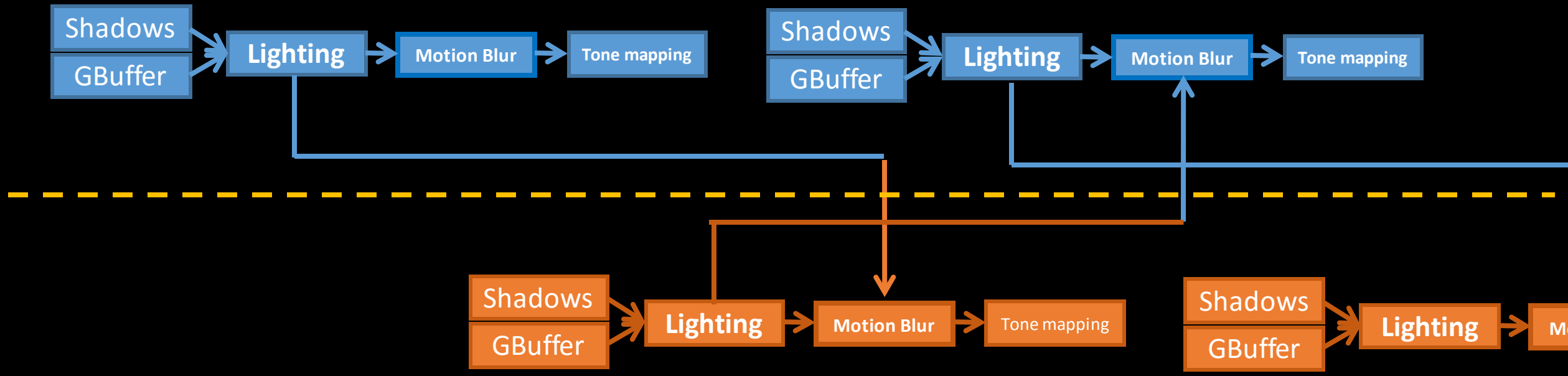


ADDING MGPU SUPPORT



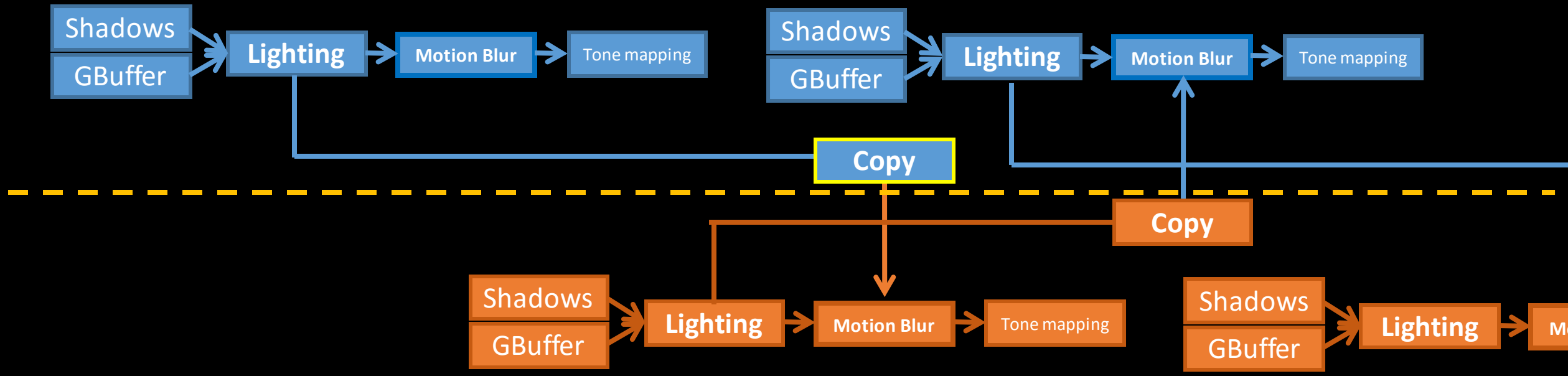
ADDING MGPU SUPPORT

- Copies need to go over the PCIE! Who are you gonna call?



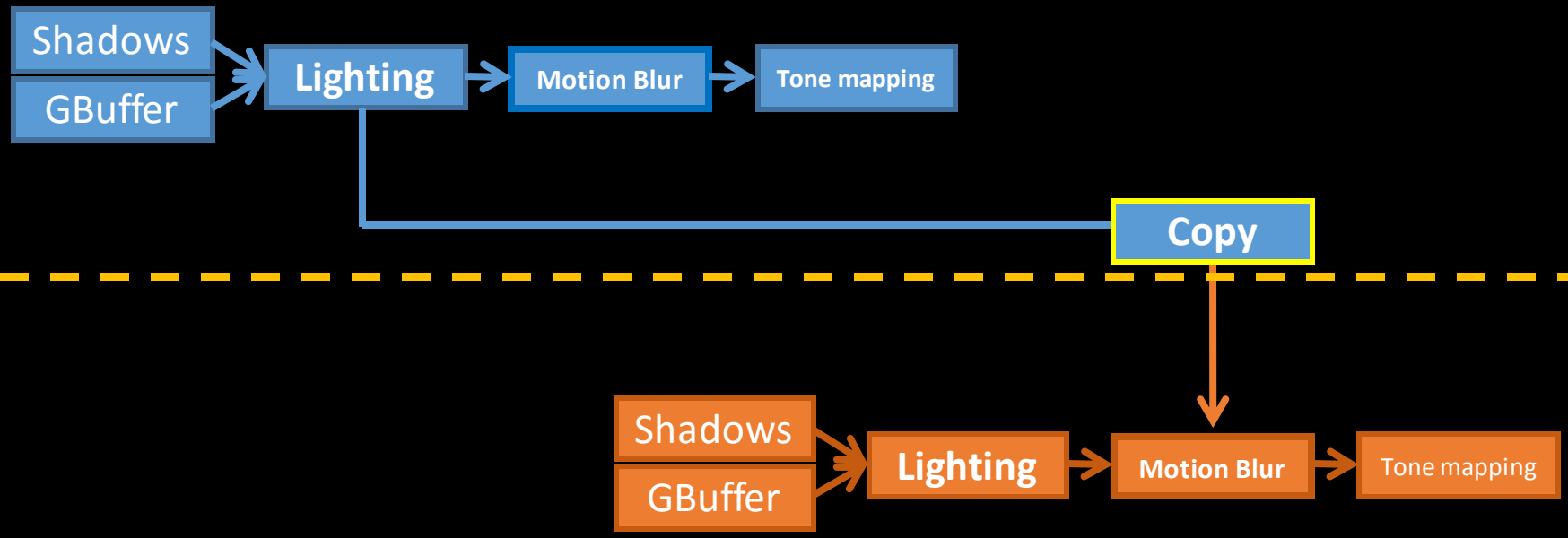
ADDING MGPU SUPPORT

- That is a job for the copy queue



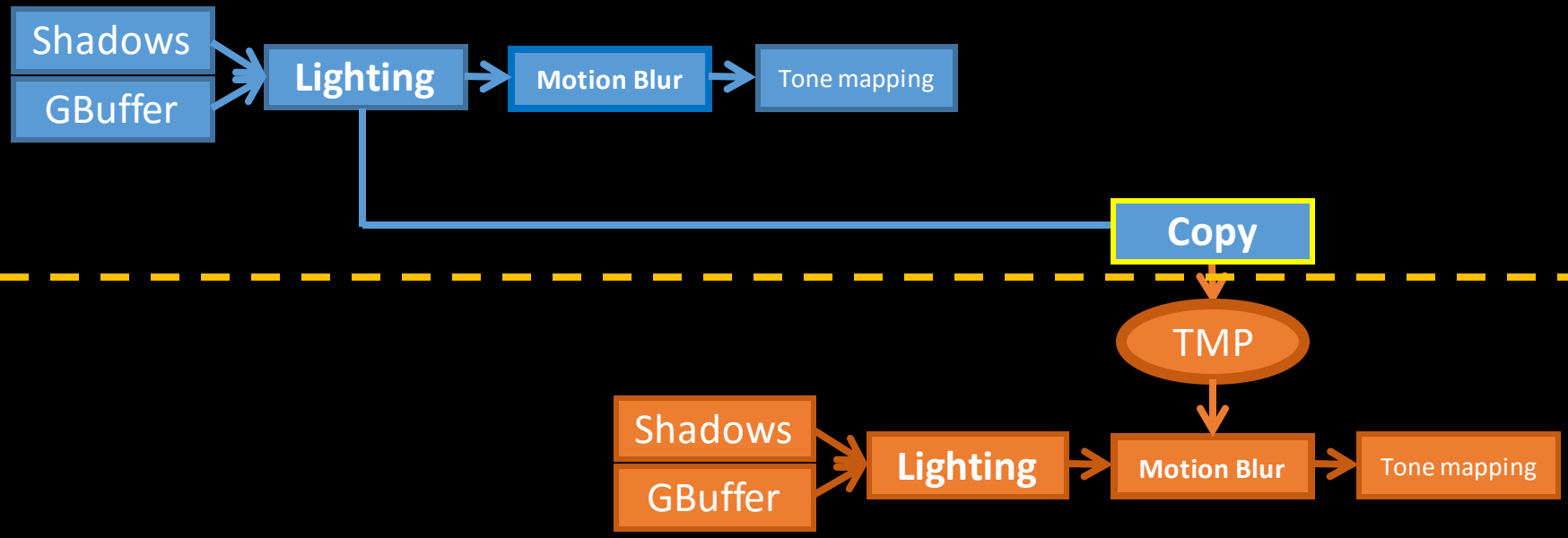
ADDING MGPU SUPPORT

- Let's focus on the copy



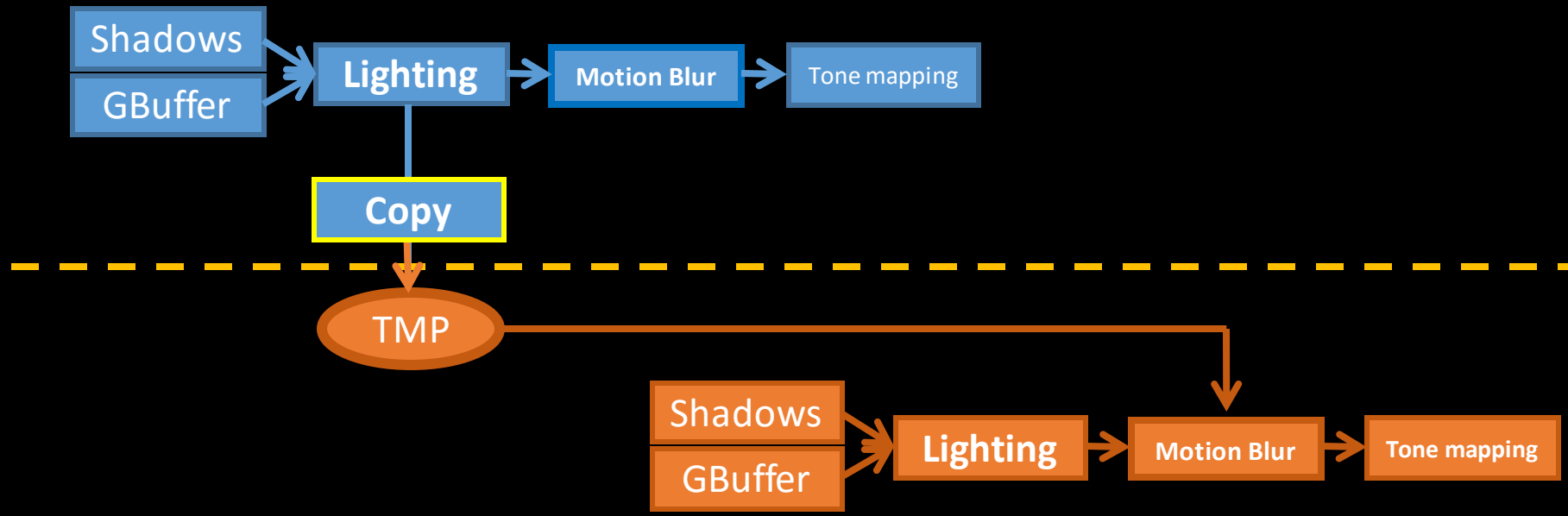
ADDING MGPU SUPPORT

- Copy into a temporary resource



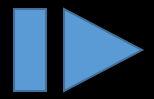
ADDING MGPU SUPPORT

- Copy as soon as possible!



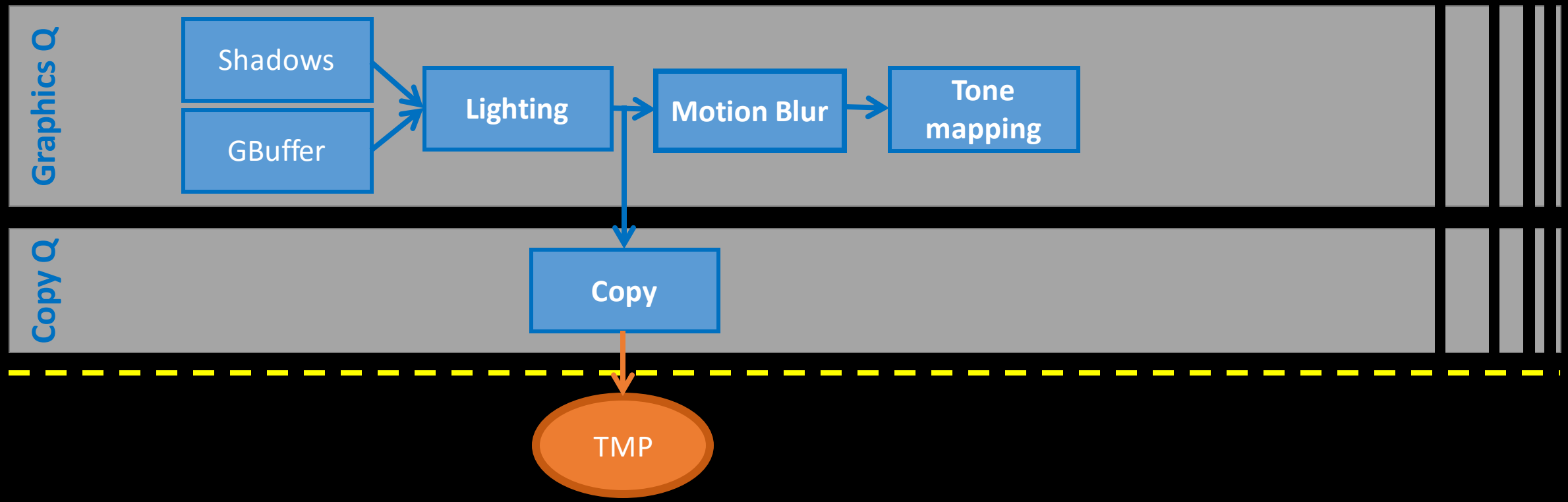
SYNCHRONIZATION

- Queue operations
  - Execute()
  - Wait()
  - Signal()



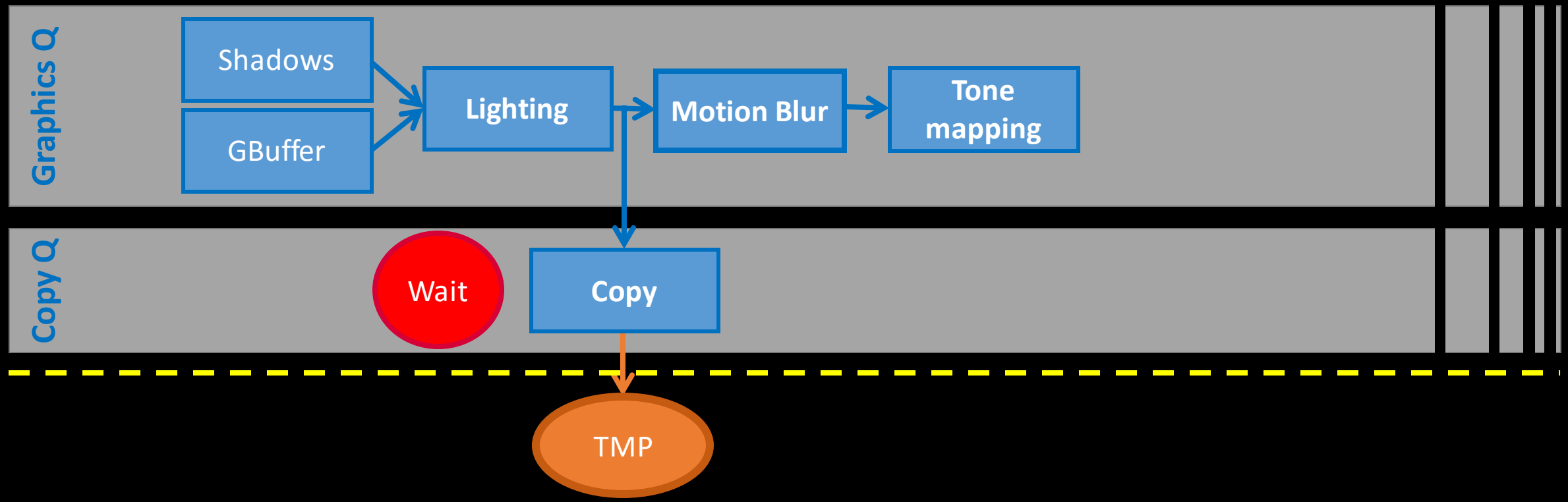
SYNCHRONIZATION

- Queues detail



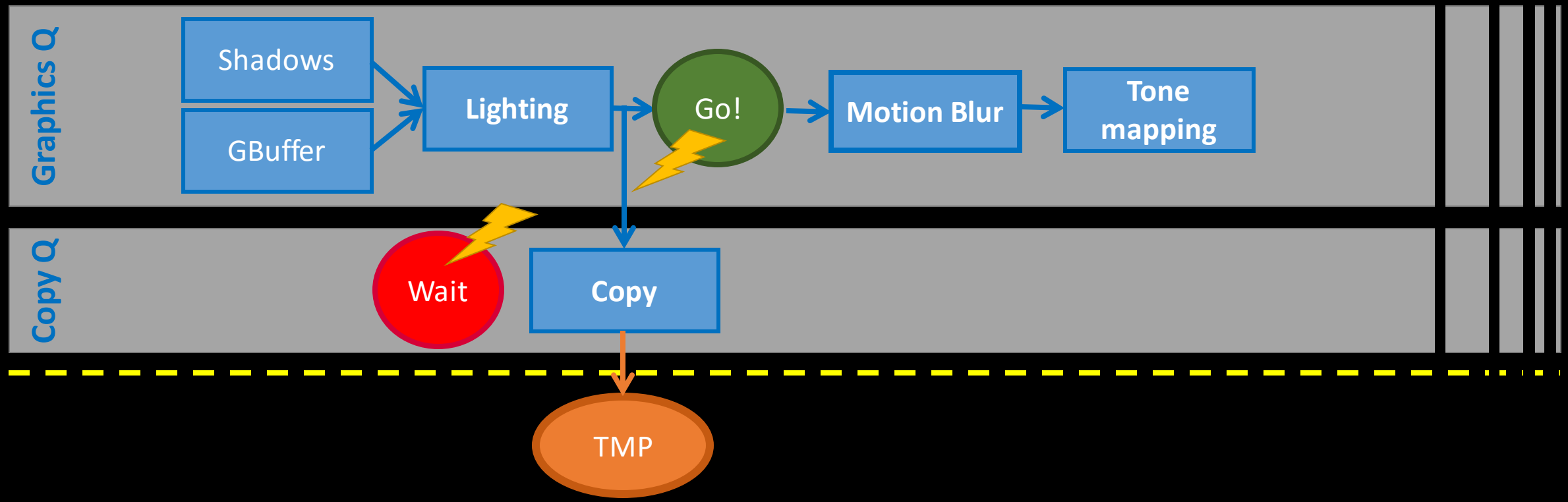
SYNCHRONIZATION

- Pausing the Copy Queue



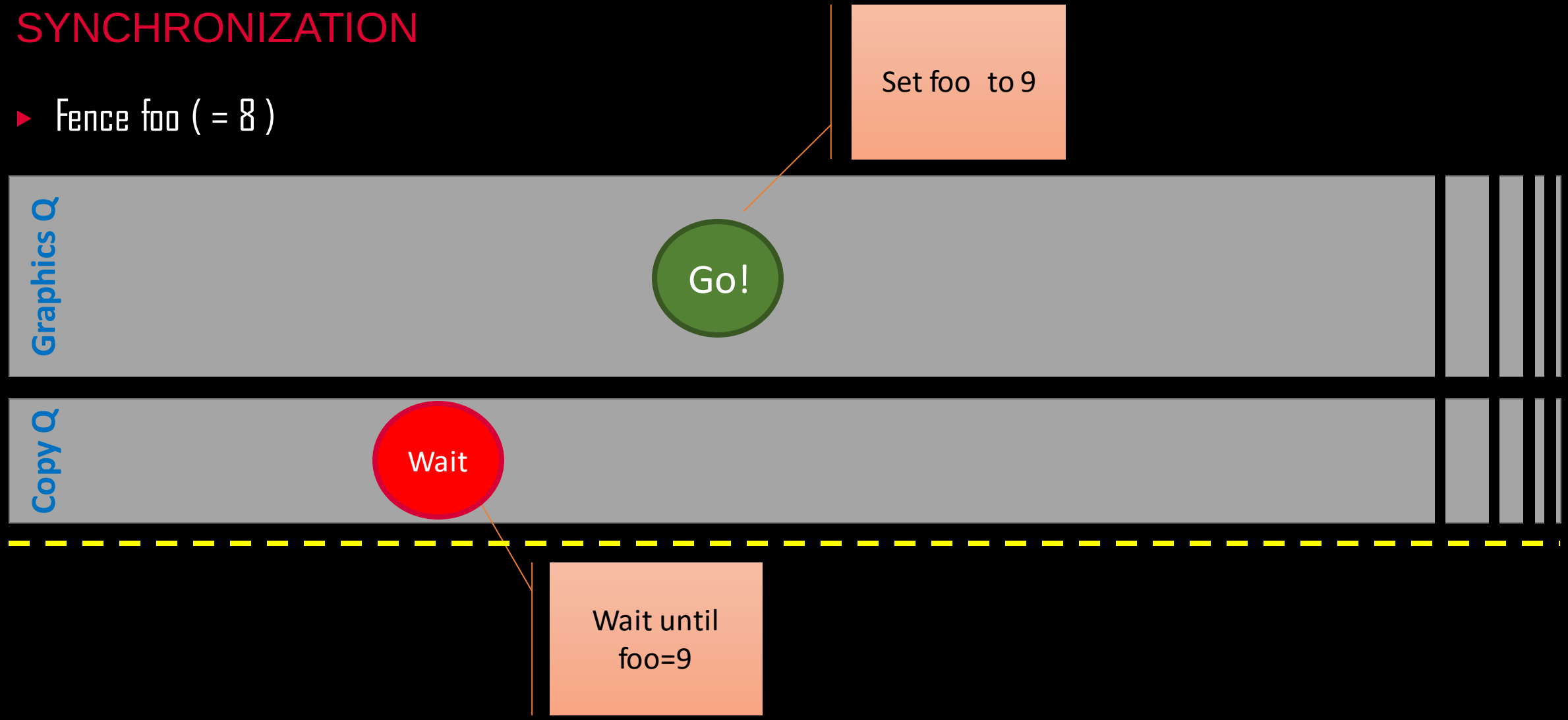
SYNCHRONIZATION

- Resuming the Copy Queue



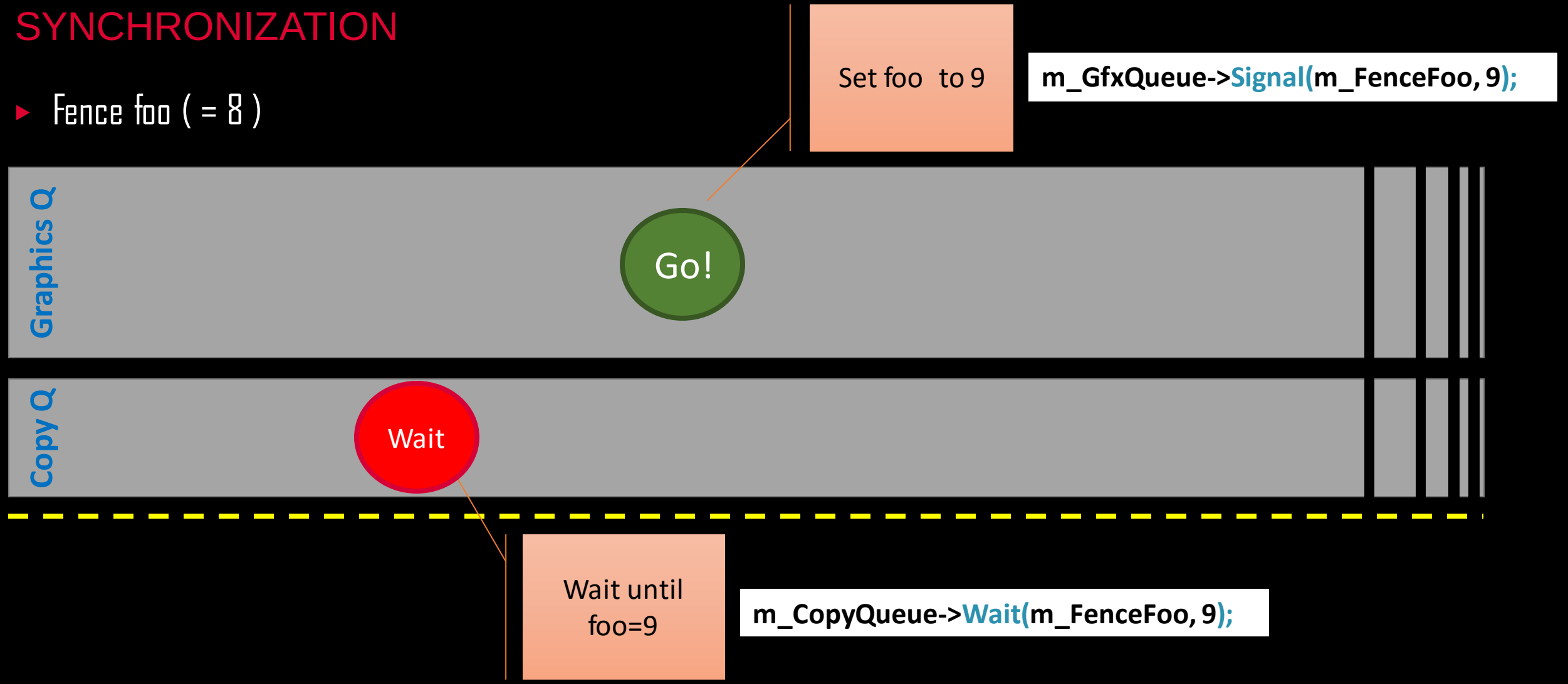
SYNCHRONIZATION

- Fence foo (= 8)



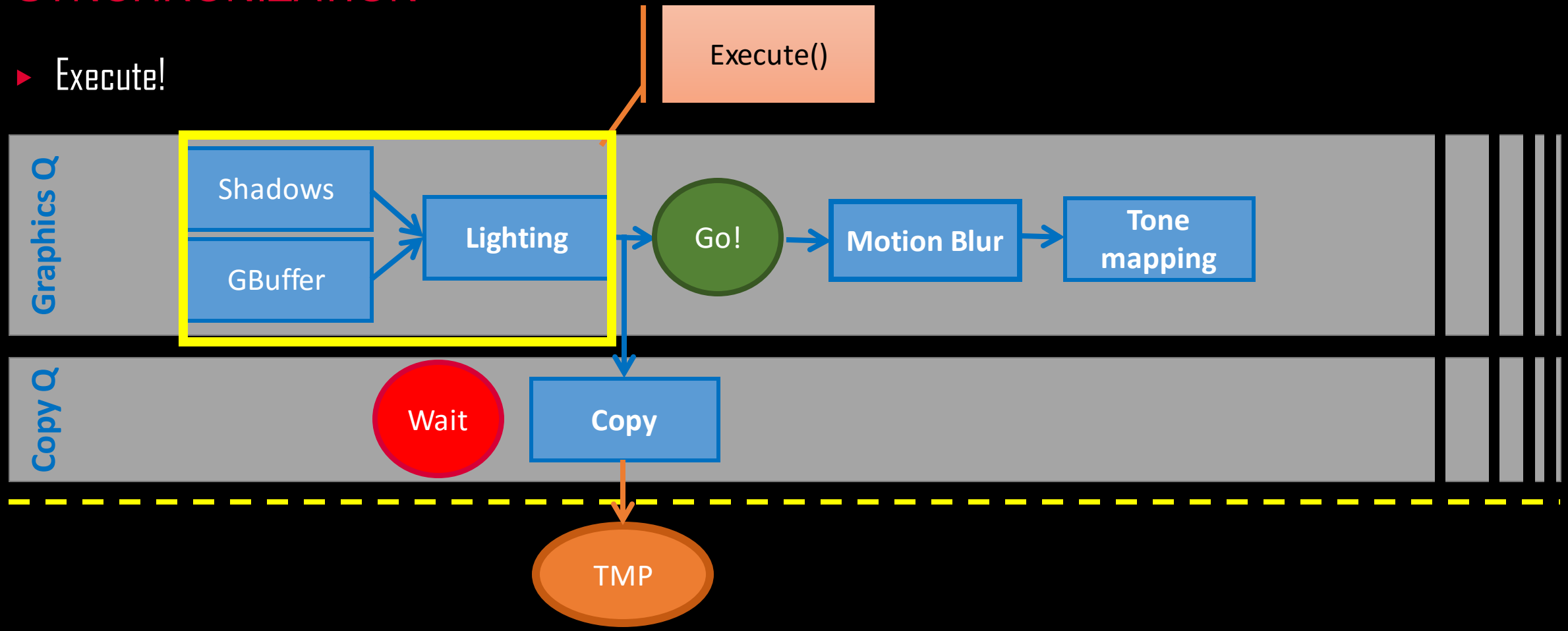
SYNCHRONIZATION

- Fence foo (= 8)



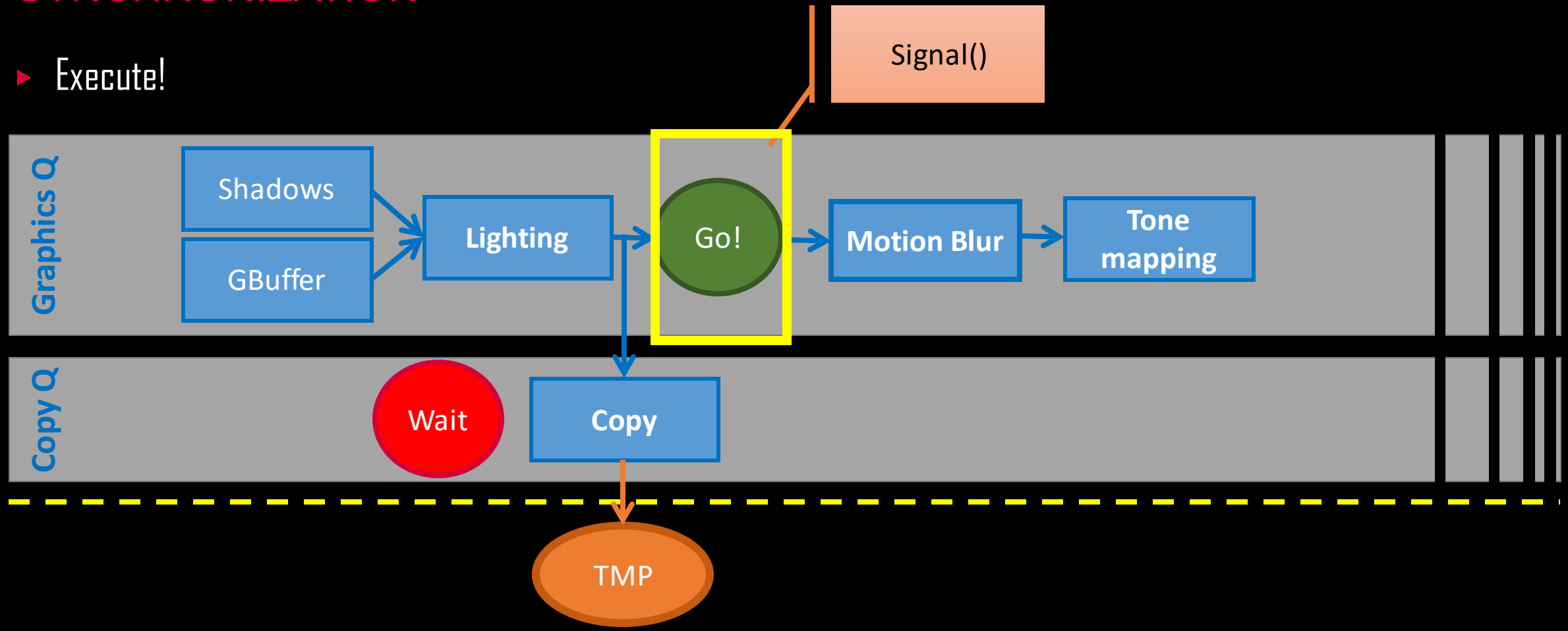
SYNCHRONIZATION

- Execute!



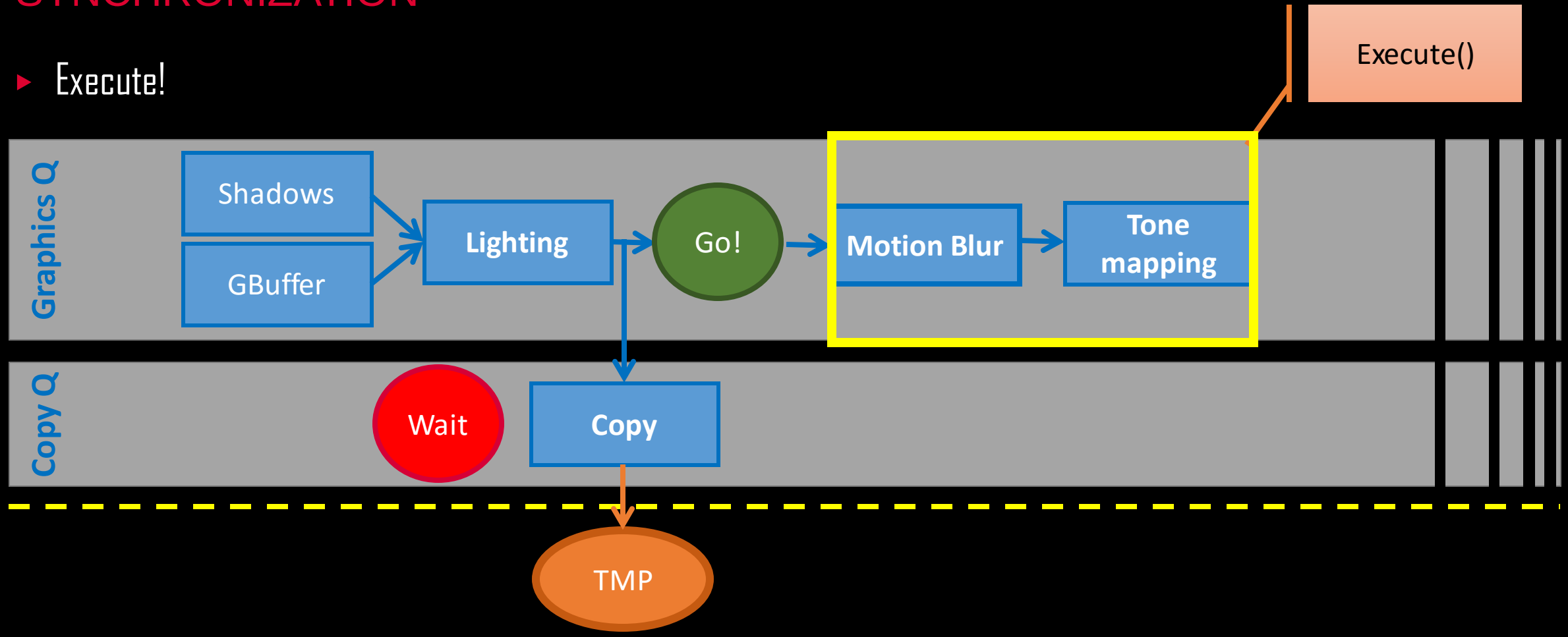
SYNCHRONIZATION

- Execute!



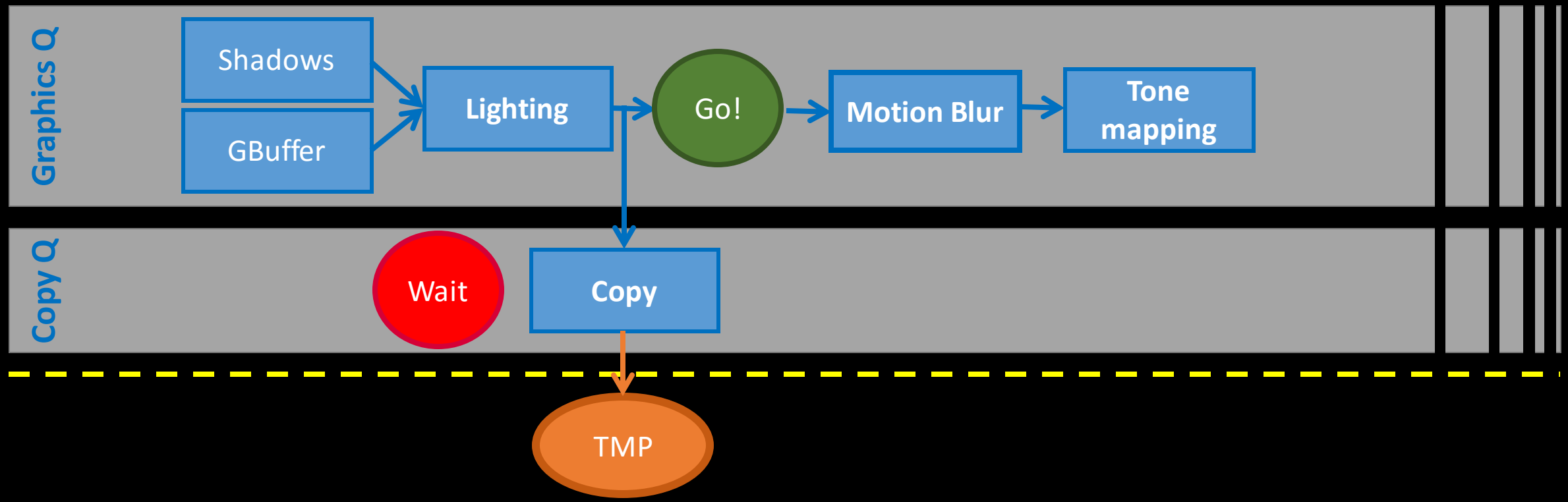
SYNCHRONIZATION

- Execute!



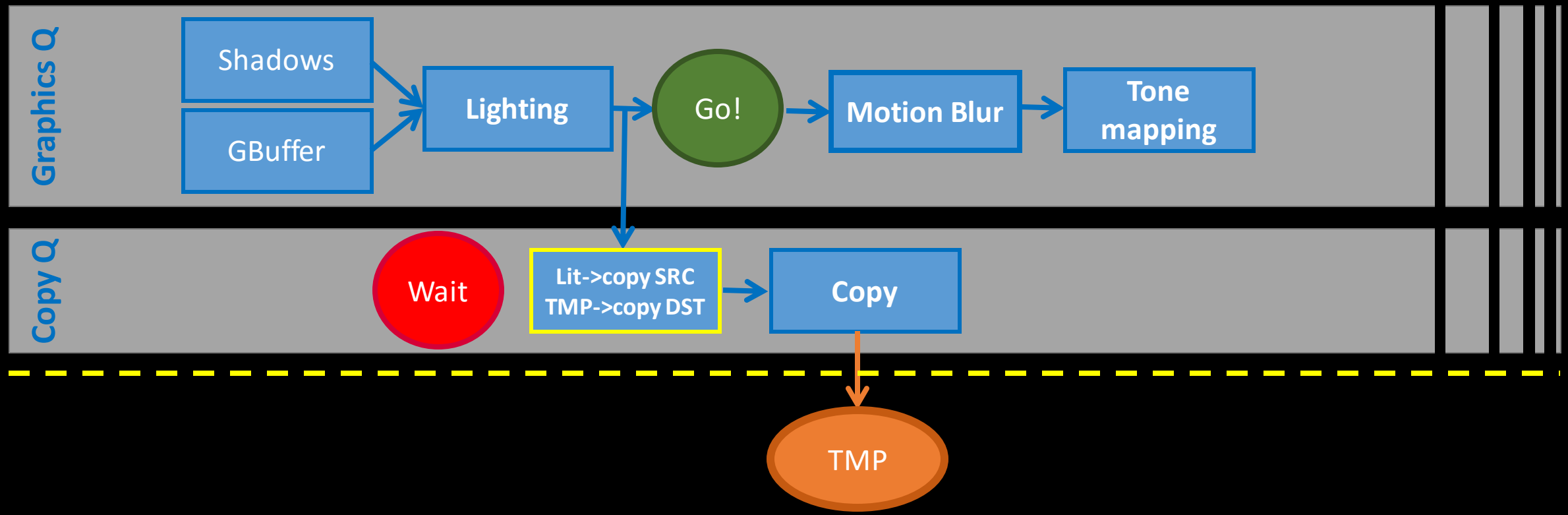
ADDING MGPU SUPPORT

- Now with barriers



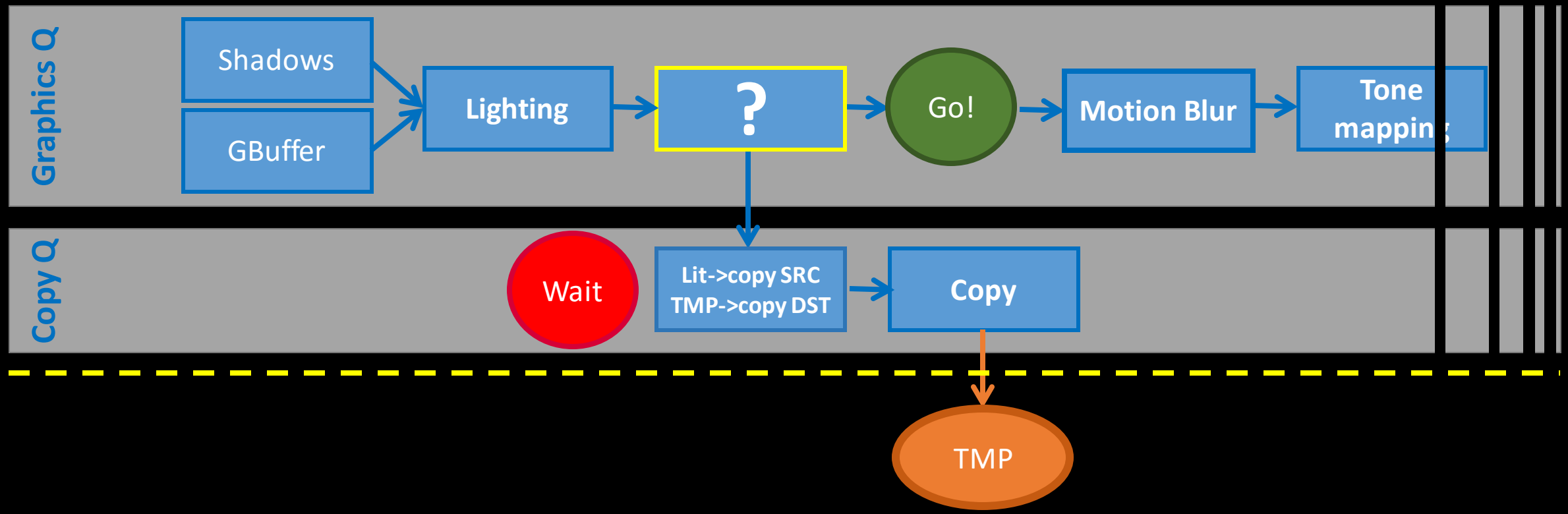
ADDING MGPU SUPPORT

- Now with barriers



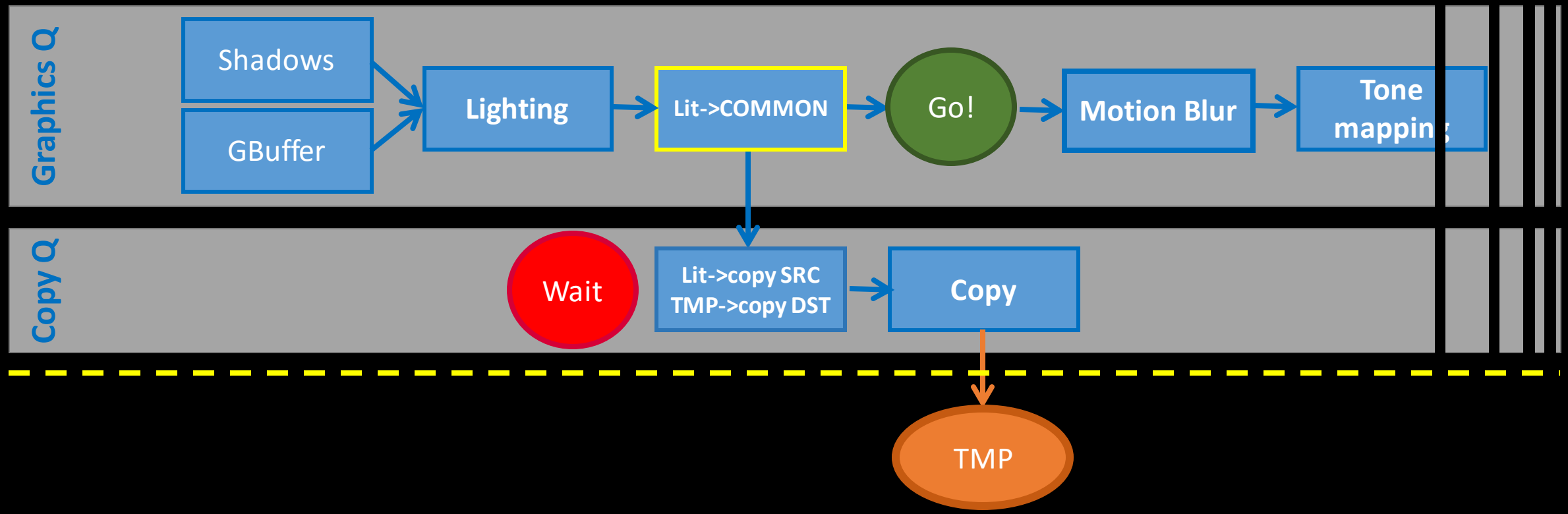
BARRIERS

- Let it go!



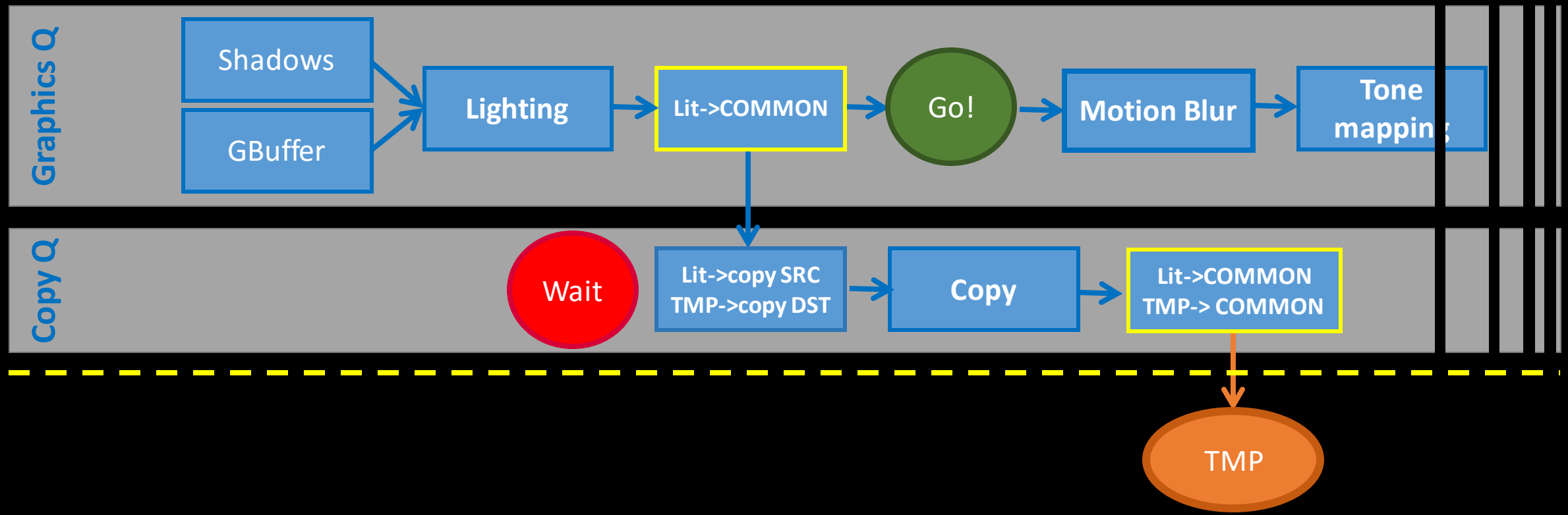
BARRIERS

- Release ownership by transitioning to COMMON



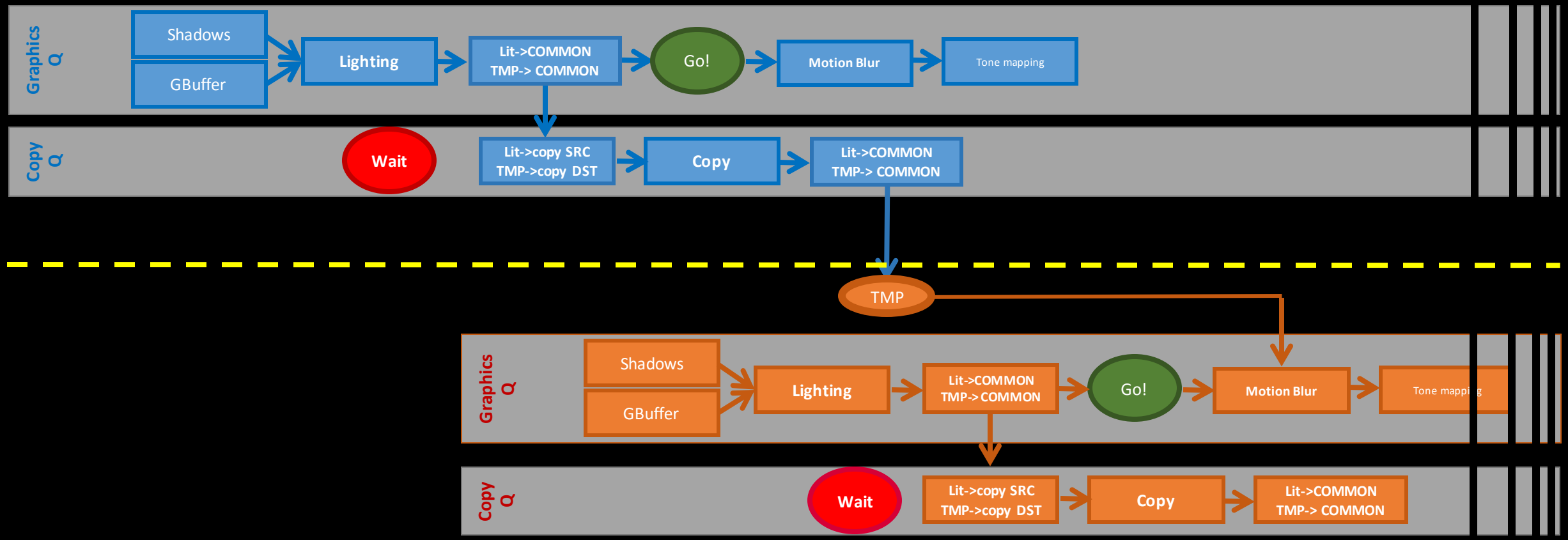
BARRIERS

- Release ownership by transitioning to COMMON



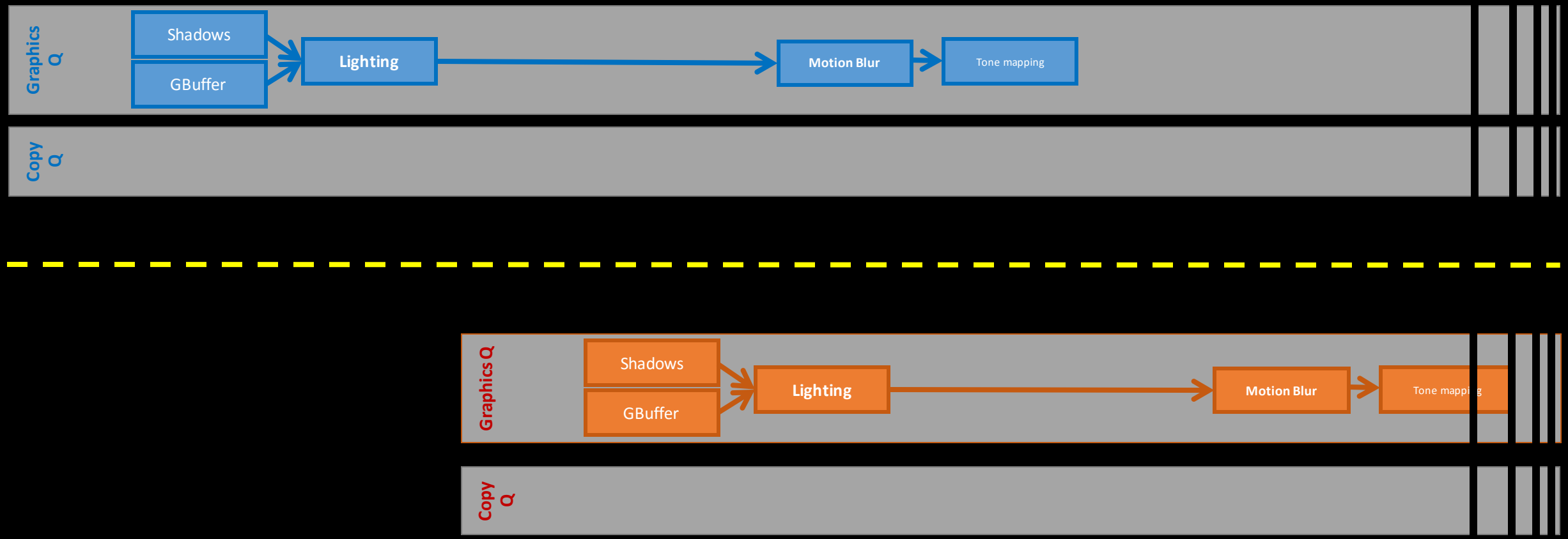
ADDING MGPU SUPPORT

- All together now!



ADDING MGPU SUPPORT

- After and before



COPY QUEUE CODE

```
// Wait for signal from the GFX queue
m_CopyQueue->Wait(m_FenceFoo, 9);

// Taking over resources from the Graphics queue
D3D12_RESOURCE_BARRIER barriersPreCopy[] = {
   CD3DX12_RESOURCE_BARRIER::Transition(pDest, D3D12_RESOURCE_STATE_COMMON, D3D12_RESOURCE_STATE_COPY_DEST),
   CD3DX12_RESOURCE_BARRIER::Transition(pSrc, D3D12_RESOURCE_STATE_COMMON, D3D12_RESOURCE_STATE_COPY_SOURCE),
};
m_pCopyCmdLst->ResourceBarrier(2, barriersPreCopy);

// Do Copy
m_pCopyCmdLst->CopyResource(pDest, pSrc);

// Release ownership so the direct queue can use these resources
D3D12_RESOURCE_BARRIER barriersPostCopy[] = {
   CD3DX12_RESOURCE_BARRIER::Transition(pSrc, D3D12_RESOURCE_STATE_COPY_SOURCE, D3D12_RESOURCE_STATE_COMMON),
   CD3DX12_RESOURCE_BARRIER::Transition(pDest, D3D12_RESOURCE_STATE_COPY_DEST, D3D12_RESOURCE_STATE_COMMON),
};
pCopyCmdLst->ResourceBarrier(2, barriersPostCopy);
```

PSEUDO CODE

- Graphics Queue A

```
Shadows(A.shadowRT);
GBuffer(A.gbufferRT);
Lighting(A.litRT, A.shadowRT, A.gbufferRT);
Signal(A.CopyQueue);
MotionBlur(A.hdr, A.litRT, A.lastFrame);
ToneMapping(swapchain, A.hdr);
Present();
```

- Copy queue A

```
Wait(A.DirectQueue)
Copy( B.lastFrame , A.litRT);
```

PSEUDO CODE

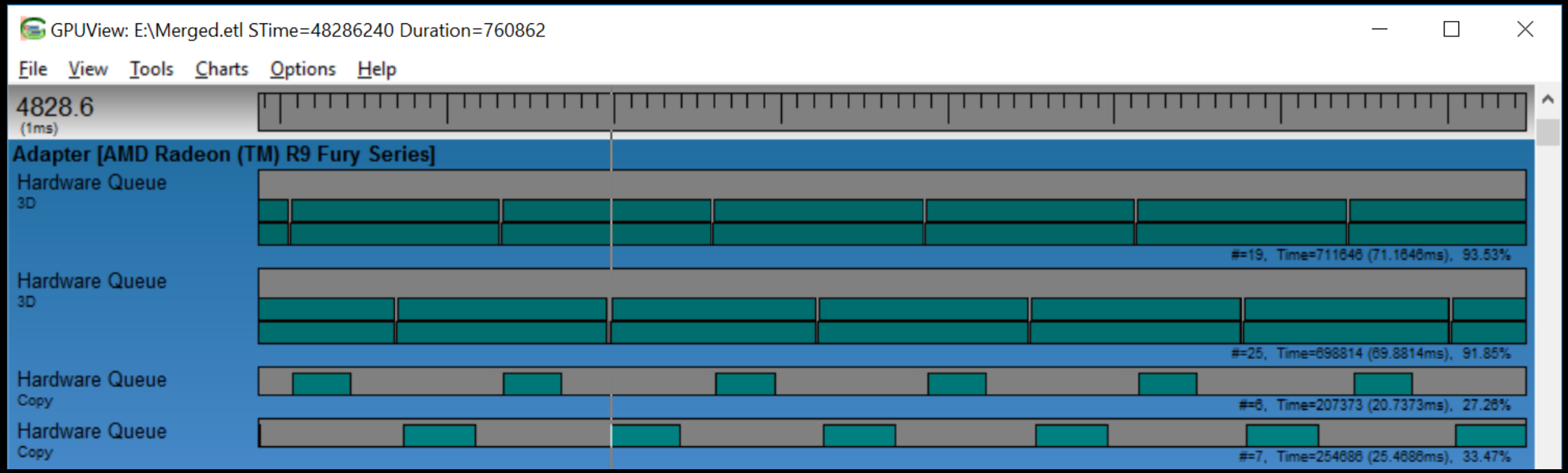
- Graphics Queue B

```
Shadows(B.shadowRT);
GBuffer(B.gbufferRT);
Lighting(B.litRT, B.shadowRT, B.gbufferRT);
Signal(B.CopyQueue);
MotionBlur(B.hdr, B.litRT, B.lastFrame);
ToneMapping(swapchain, B.hdr);
Present();
```

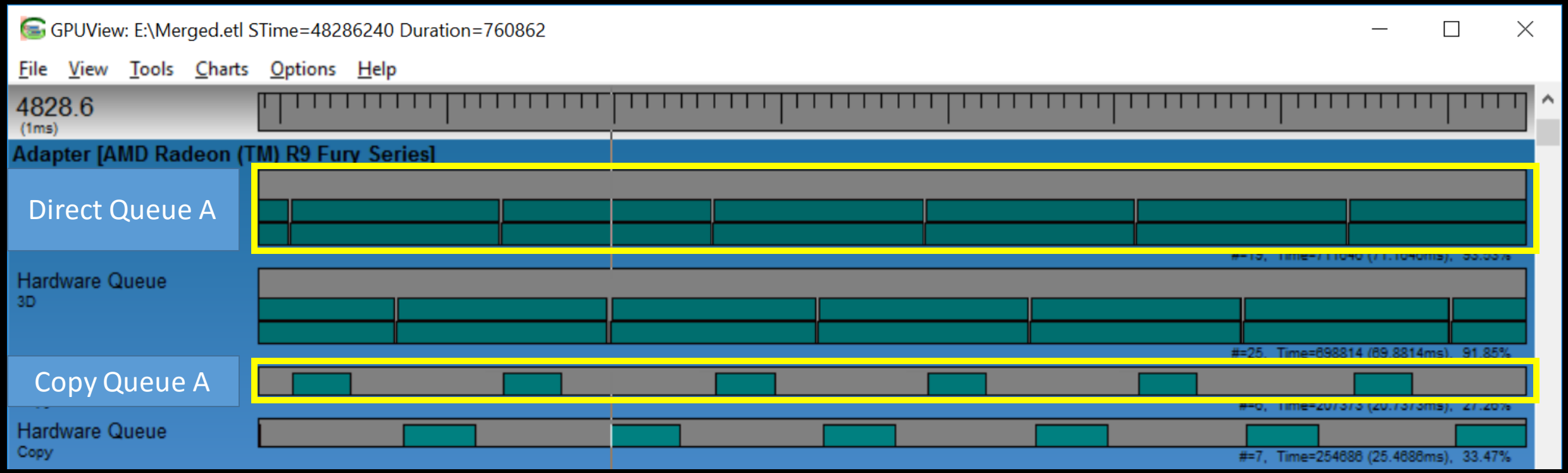
- Copy queue B

```
Wait(B.DirectQueue)
Copy( A.lastFrame , B.litRT);
```

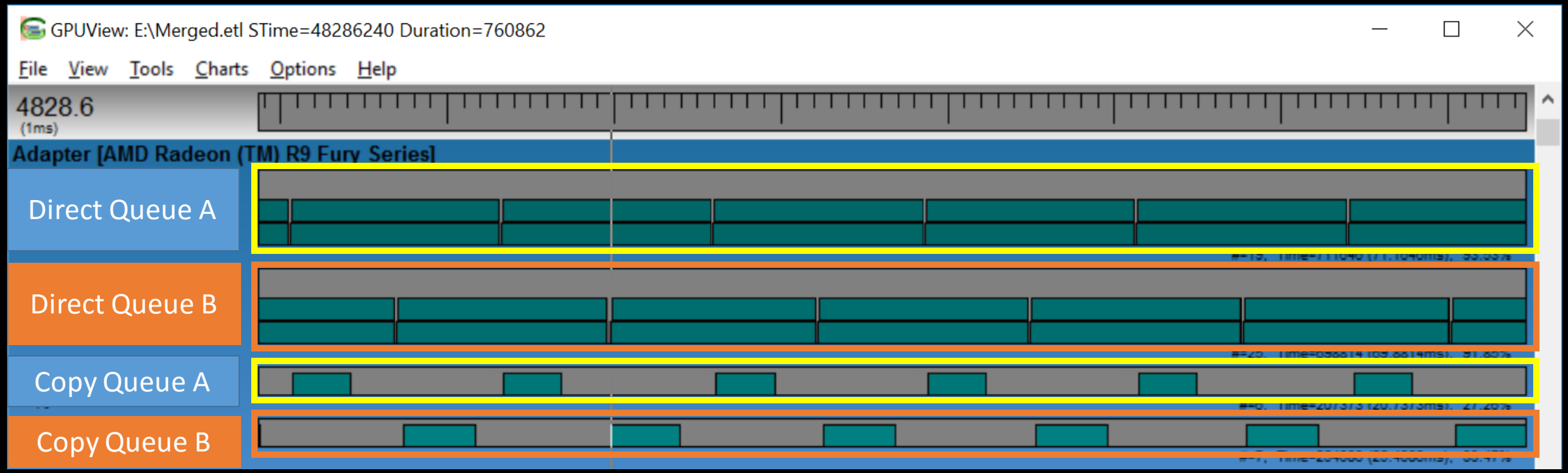
TOOLS OF THE TRADE - GPUVIEW



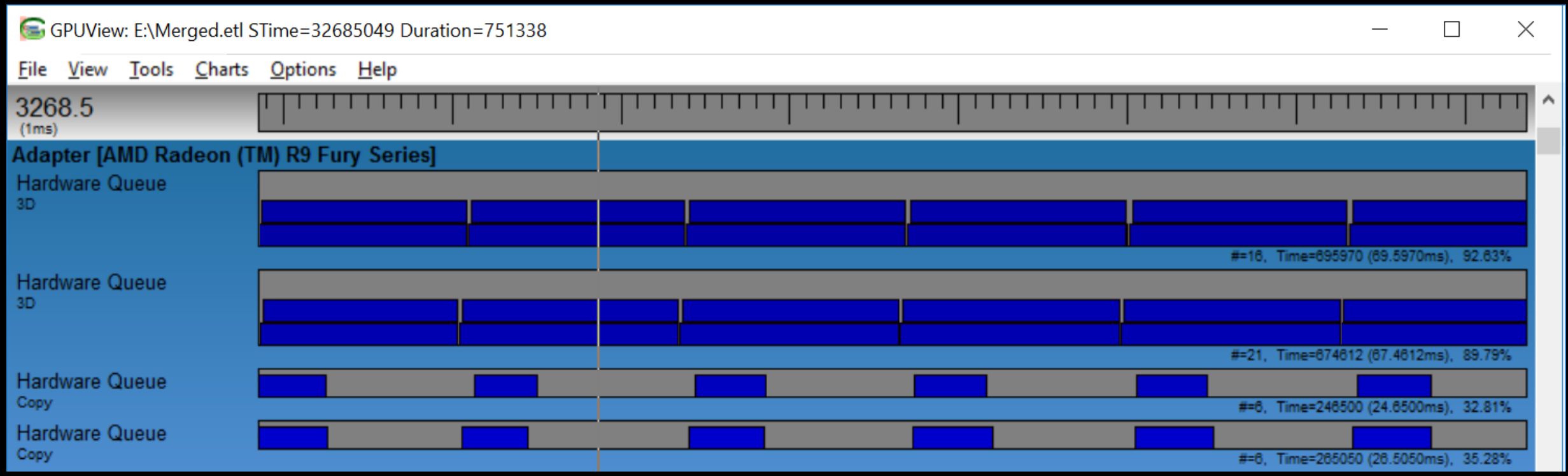
TOOLS OF THE TRADE - GPUVIEW



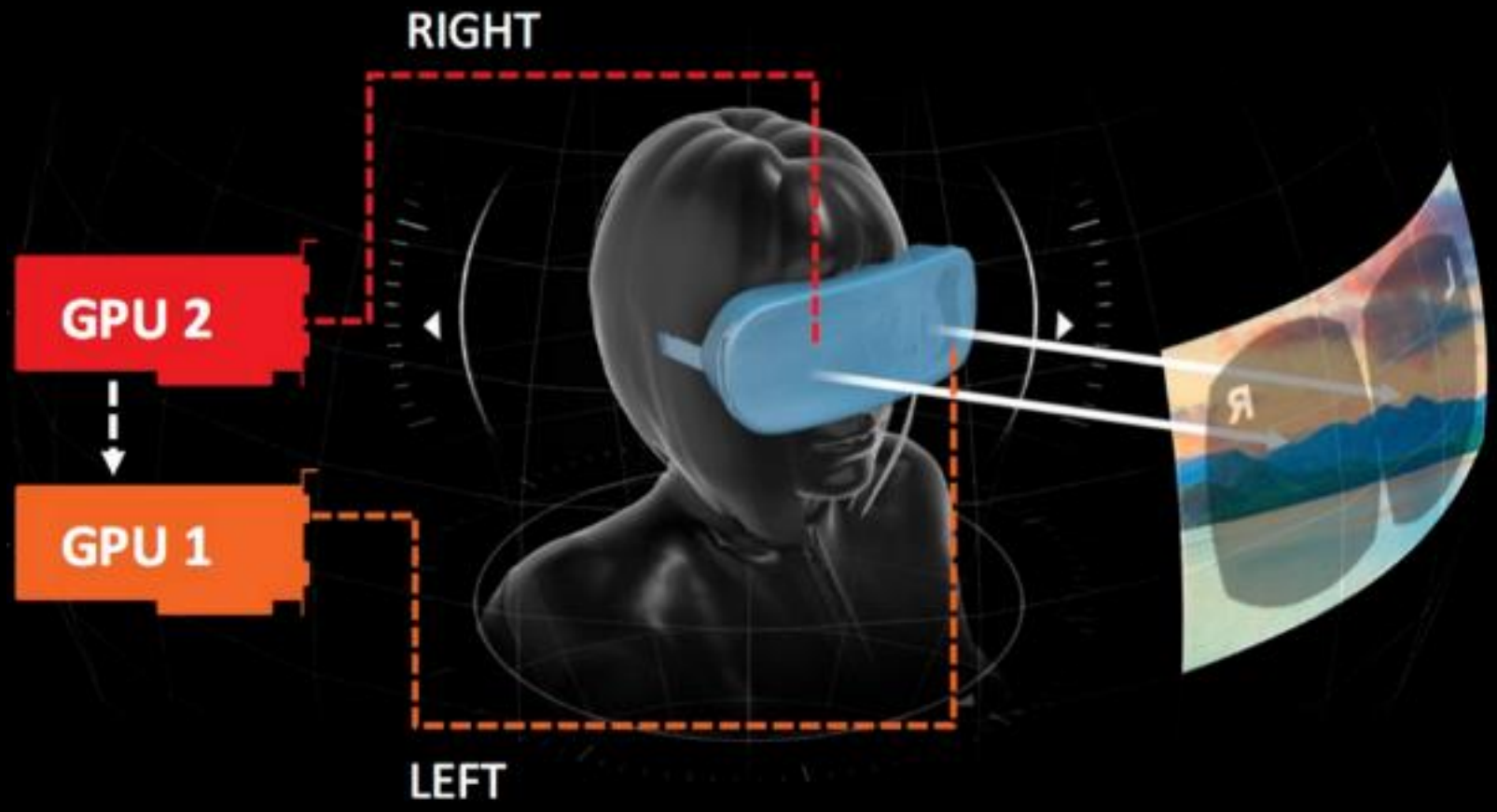
TOOLS OF THE TRADE - GPUVIEW



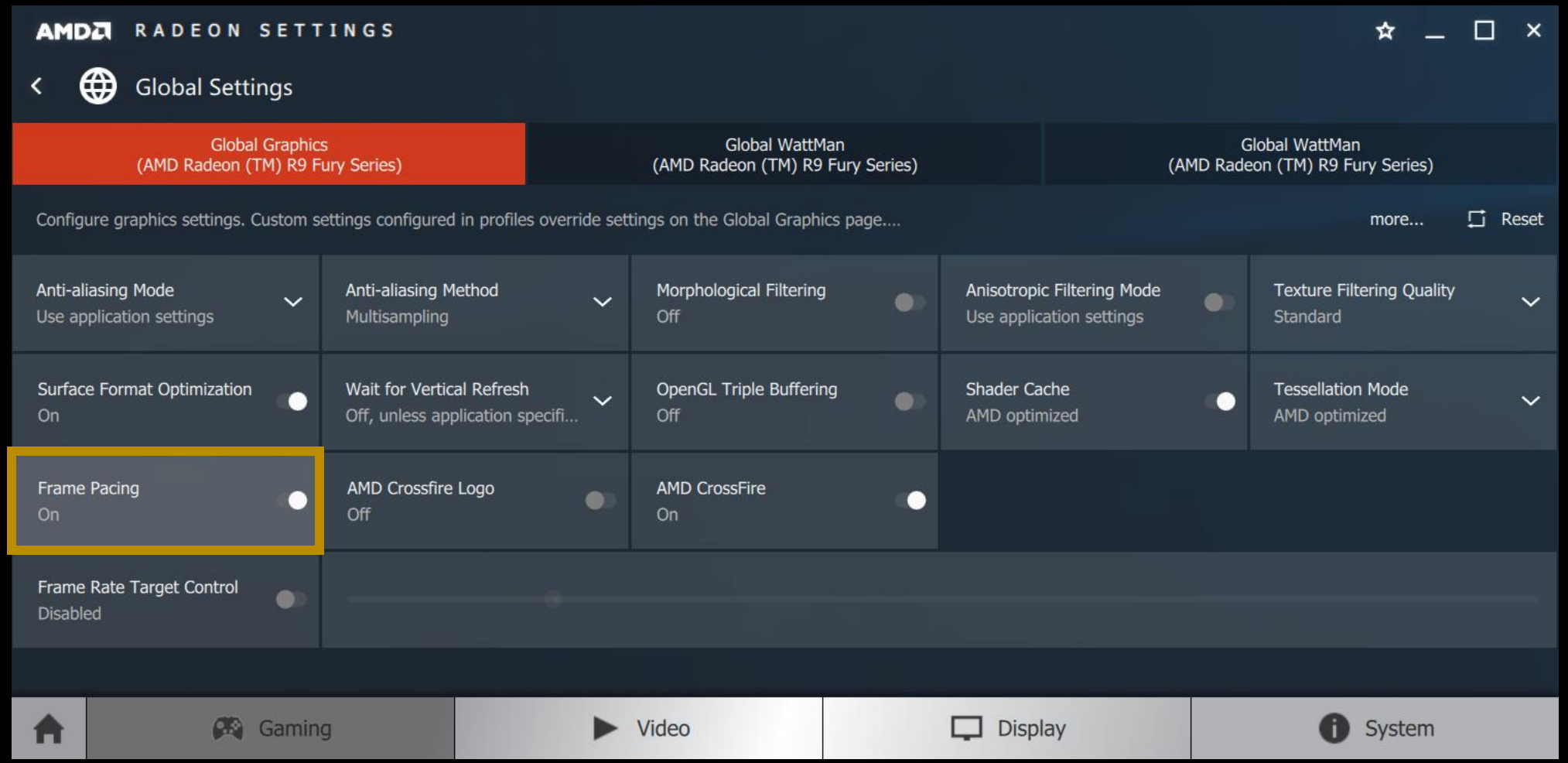
TOOLS OF THE TRADE - GPUVIEW



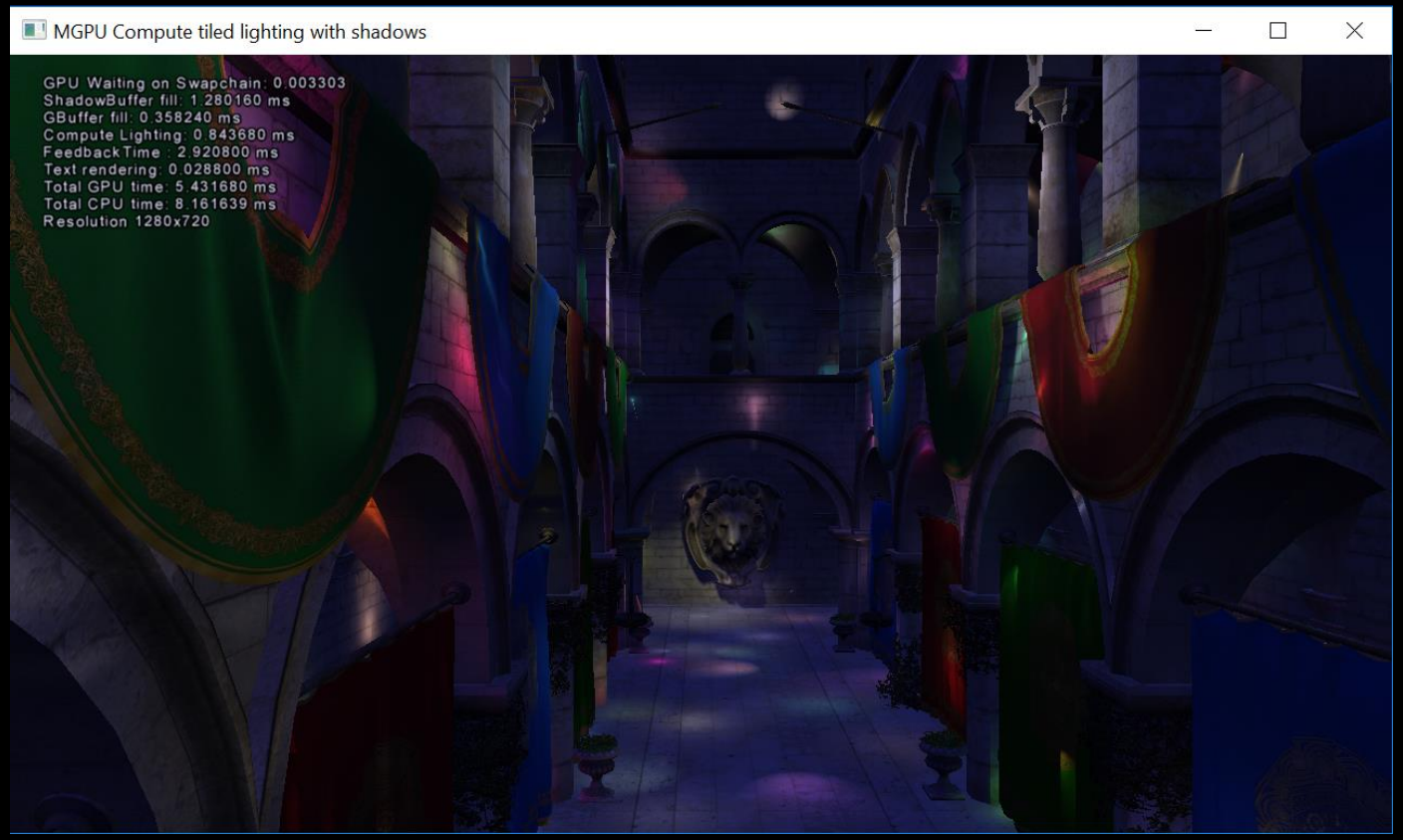
TOOLS OF THE TRADE - GPUVIEW



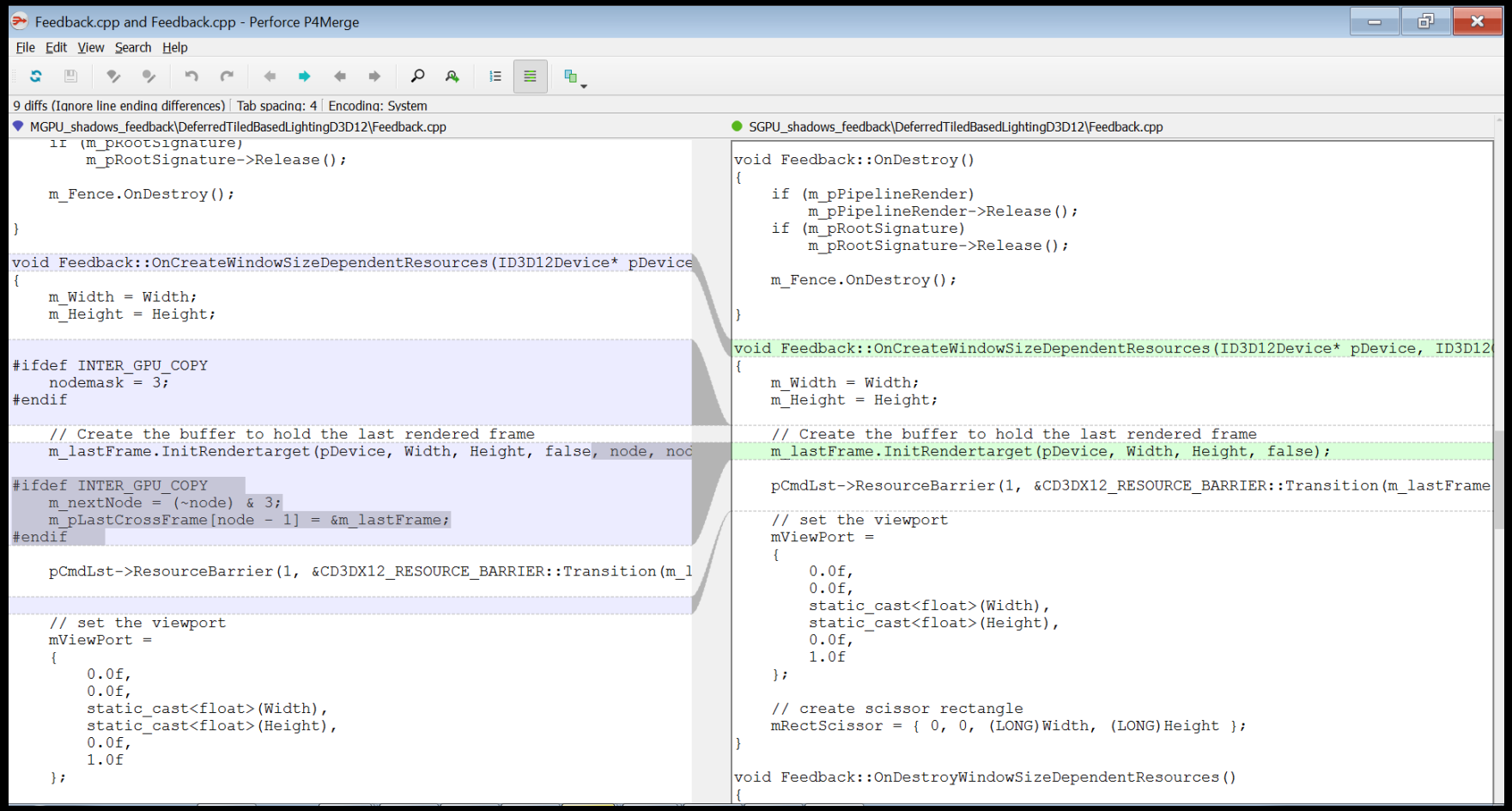
FRAME PACING



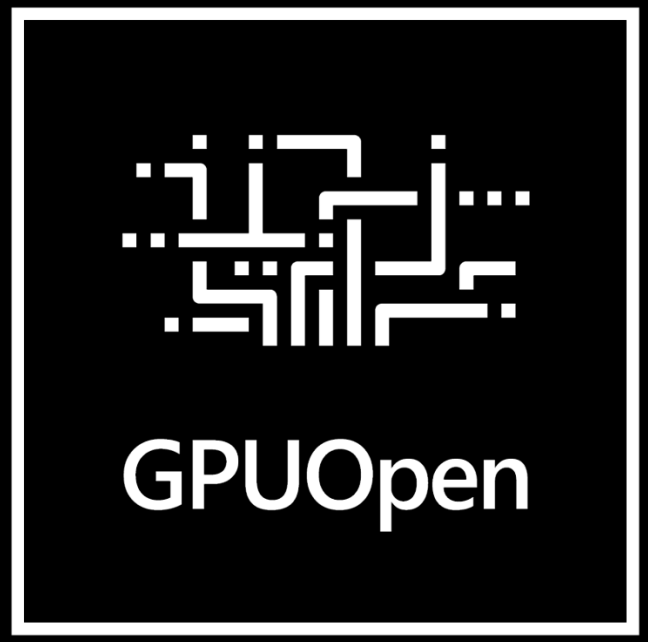
SHOW ME THE CODE



SHOW ME THE CODE



SHOW ME THE CODE



WRAPPING UP

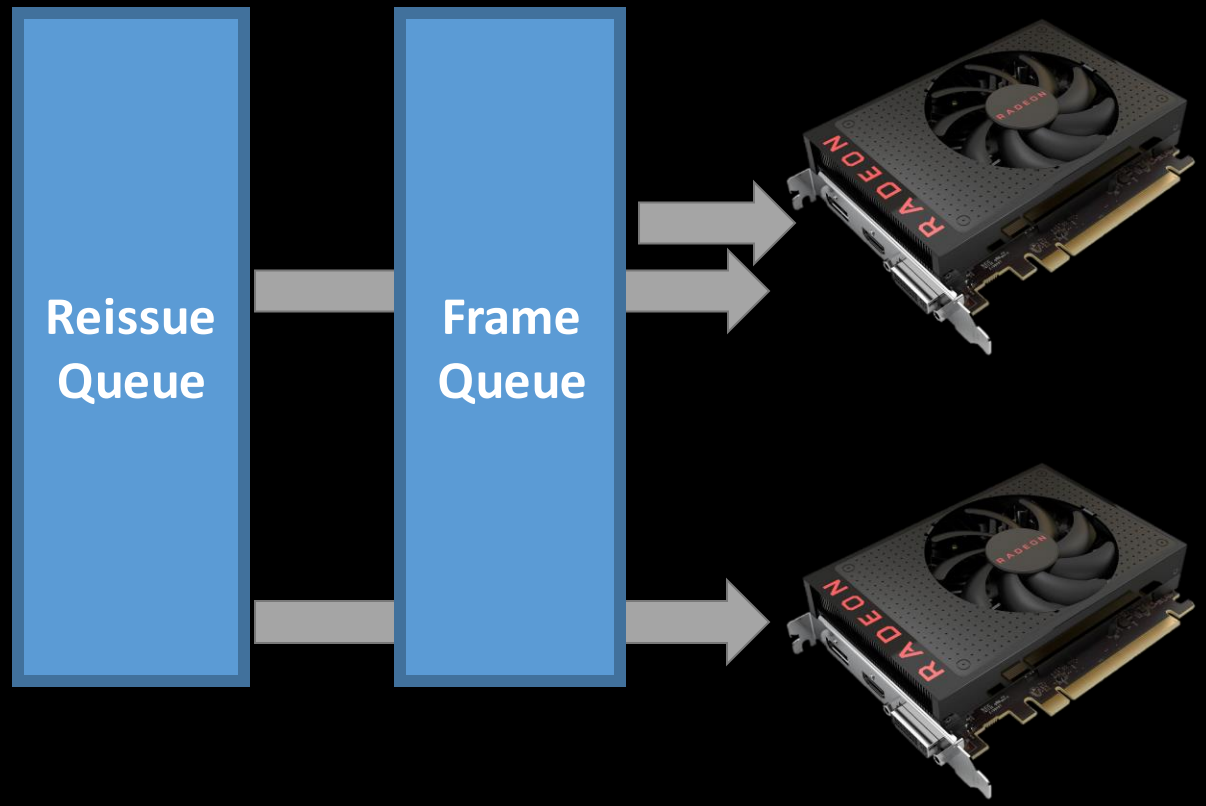
- Quick AFR
- MGPU API
- Synchronization
- Queue ownership
- Sample code
- Lucky winner!

CASE STUDY, EXPLICIT MGPU IN NITROUS

- Ashes of the Singularity supports Multi-GPU
  - Regardless of hardware supported linked mode
- Essentially AFR support, alternate frame rendering
- Nitrous 2.0 will do something different

BASICS

- Eliminated all cross frame dependencies
  - Rendering technique does not require previous frames contents
- Reissue queue which will issue command to both GPUS (even if one GPU is not 'active' for a frame)
- Frame Queue which only generates commands for active frame

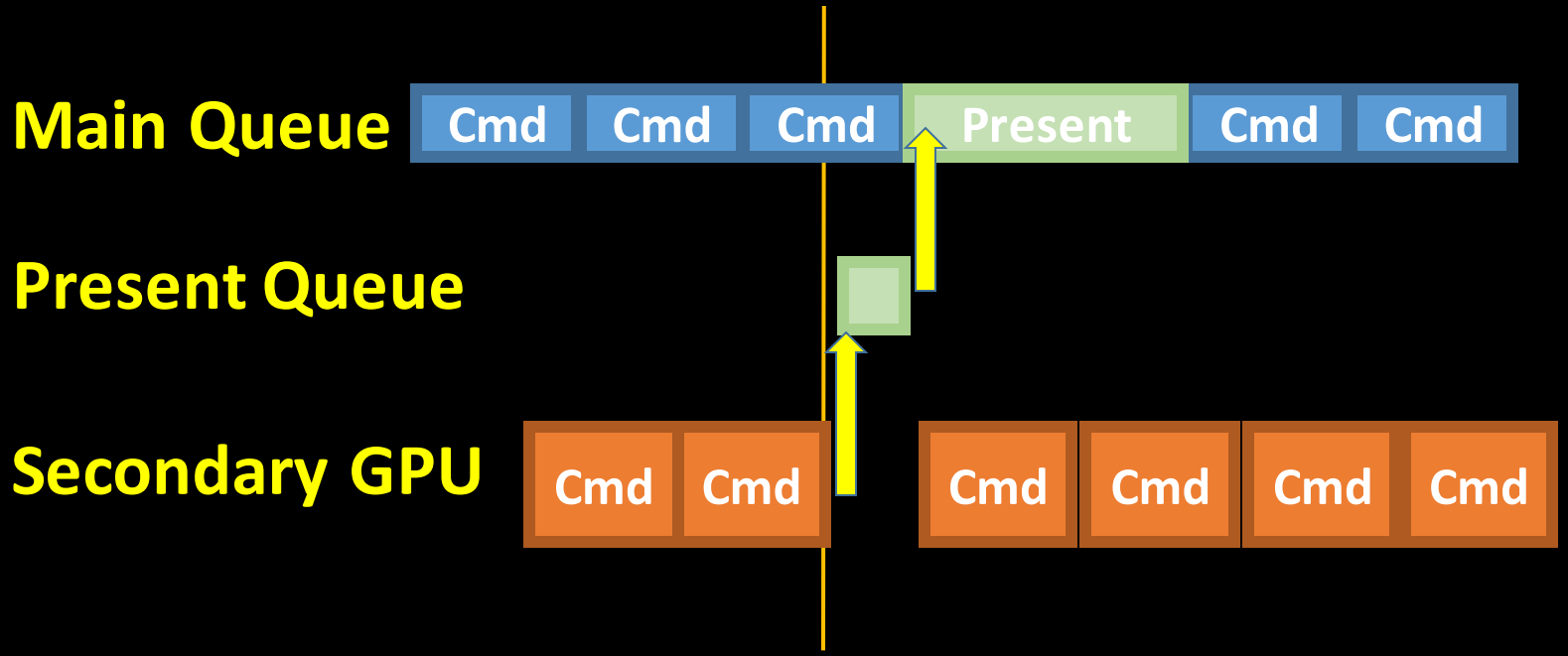


BACKBUFFER

- Nitrous 1.0 does not use backbuffer present on different GPUS
  - Backbuffers exist on "main" GPU
  - Backbuffer is manually copied via queues
- Frame pacing is done manually
- Linked mode enables more slick operation, but requires hardware level support

FRAME PACING DONE MANUALLY

- Chop main queue into < 1ms command buffers
- Allow windows 10 scheduler to insert present when second GPU is done
- Works ok, except precision of windows timers means +- 2-3ms accuracy
- Hopefully will be improved in future DX versions



SETTING QUEUED FRAMES HIGHER

```
SwapChainDesc.Flags = DXGI_SWAP_CHAIN_FLAG_FRAME_LATENCY_WAITABLE_OBJECT;
```

```
// Set number of frames to 4 if using MGPU AFR mode
g_pDevice->QueryInterface(IID_IDXGISwapChain2, (void**)&g_pSwapChain2);
g_pSwapChain2->SetMaximumFrameLatency(4);
```

```
// If using maximum frame latency, need to manually block on present
if(g_pSwapChain2)
{
    HANDLE SwapChainHandle = g_pSwapChain2->GetFrameLatencyWaitableObject();
    if(WaitForSingleObject(SwapChainHandle,0)== WAIT_OBJECT_0)
        g_bWaitOnPresent = true;
    CloseHandle(SwapChainHandle);
}
```

MULTIGPU IN NITROUS 2.0

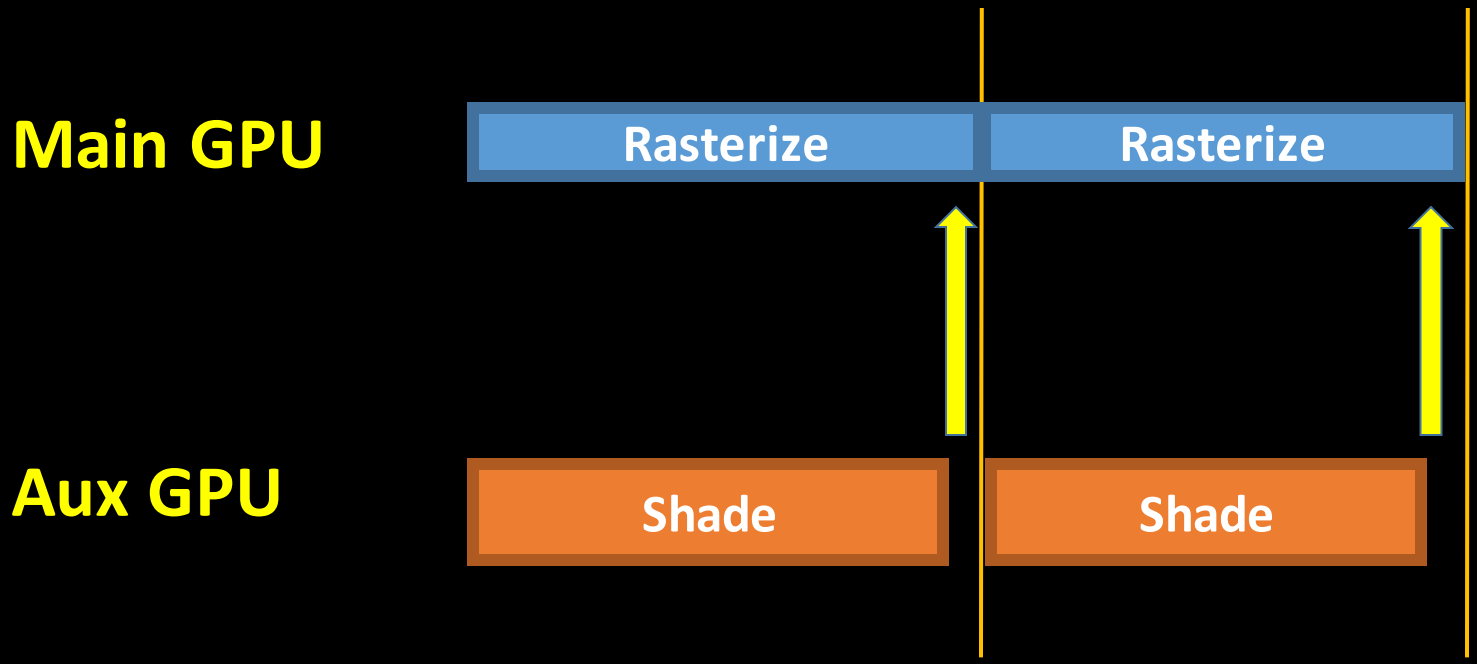
- AFR is limited approach, increases latency
- Alternatives include split screen rendering
- What about VR? One card per eye?
  - Sounds simple, but not necessarily the best choice
  - Some rendering can be shared between the eyes
- Oxide looking at something else

OBJECT SPACE RENDERING

- Opposite of deferred, shading happens before rasterization
- Shading can be shared between eyes!

COMPLETELY DIFFERENT MGPU SETUP

- Can shade on 1 GPU, and raster on other GPU
- Shading data can have (Quite a bit) latency without noticeable visual effect
- AXU GPU can be set to never block main GPU
- Mismatch of GPU performance possible
- Challenge: Load balancing, possible arbitrary # of GPUs could be supported



THANK YOU!

- GPU raffle
- Questions and Answers

- Please fill the feedback form ☺