BUILDING A SUPER RESOLUTION VIDEO COMPOSITOR

Thomas True, March 18. 2019

AGENDA

Motivation

Building Blocks

Putting the Pieces Together

Case Study

Results

Q & A

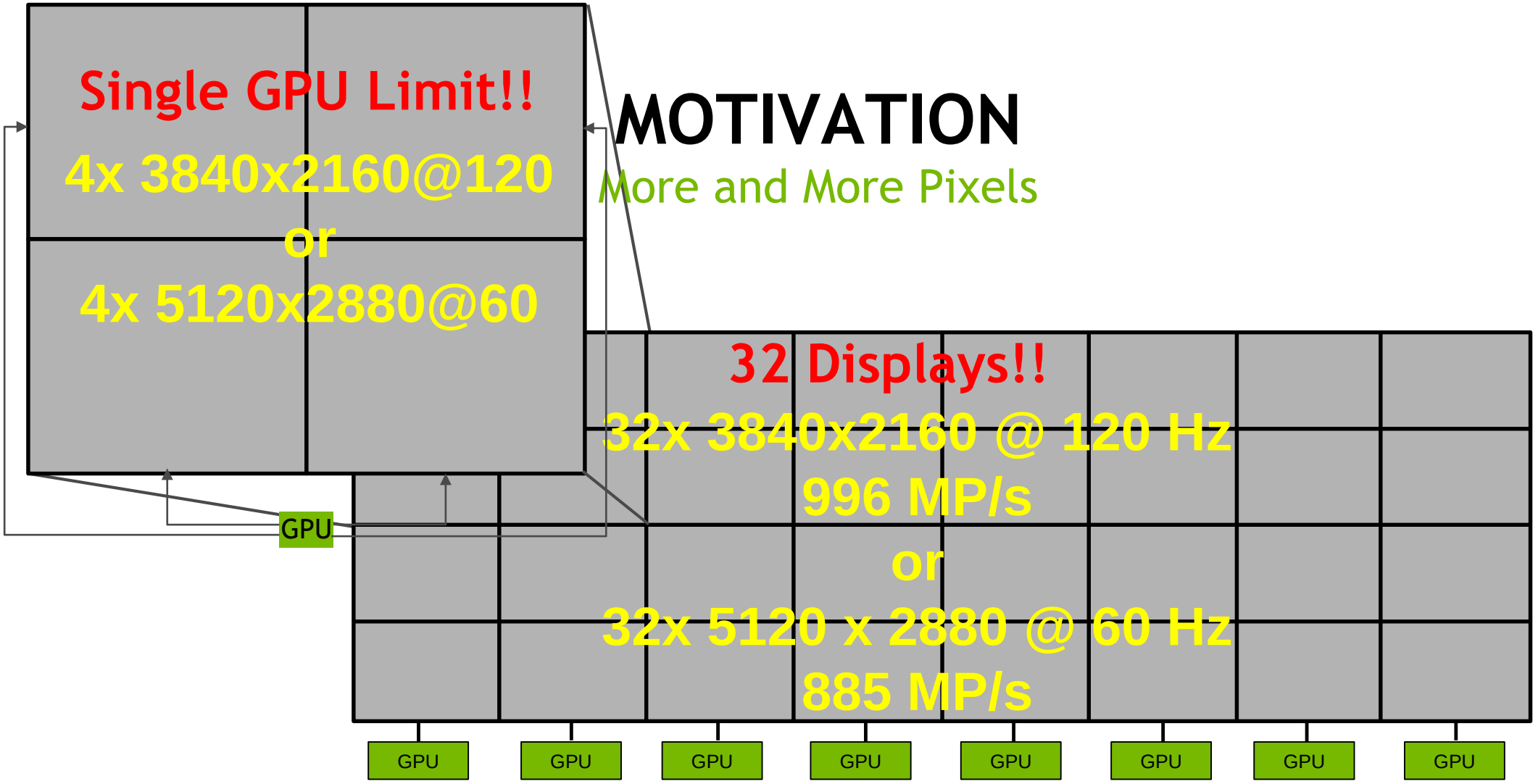
MOTIVATION

Create Large High-Resolution Displays

Photo Courtesy of Cinnemassive: http://www.cinnemassive.com/

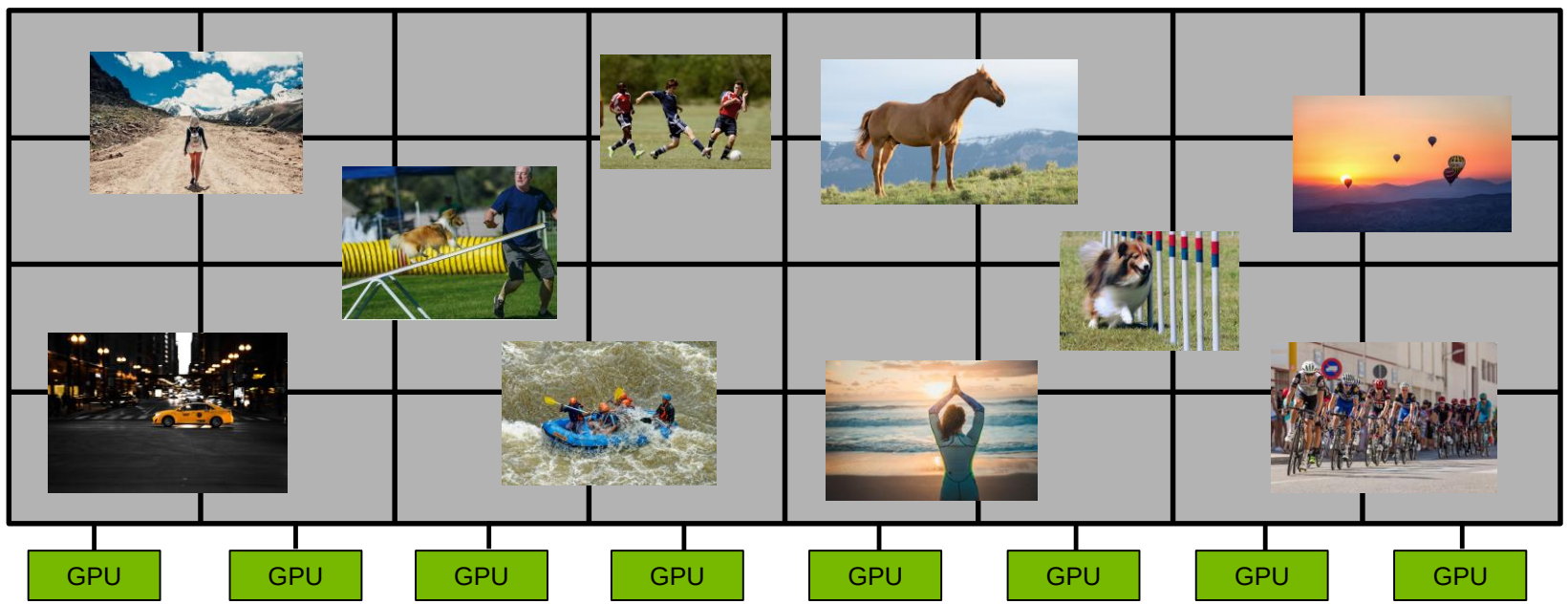
MOTIVATION

More and More Pixels



MOTIVATION

Render Video + Graphics



S8205- Multi-GPU Methods for Real-Time Graphics

S7352-See the Big Picture: How to Build Large Display Walls Using NVIDIA APIs/Tools

BUILDING BLOCKS

A Four Legged Stool

DISPLAY SYNCHRONIZATION

- Mosaic
- Quadro Sync

GPU VIDEO PROCESSING

NVIDIA Codec SDK

LOW-LATENCY VIDEO INGEST

- GPU Direct for Video
- GPU Direct RDMA

EFFICIENT RENDERING

OpenGL

NVIDIA CUDA

Microsoft DirectX

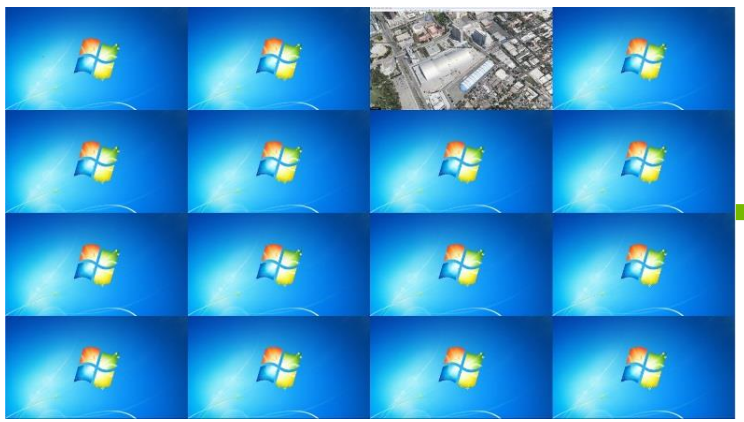
Vulkan

MOSAIC

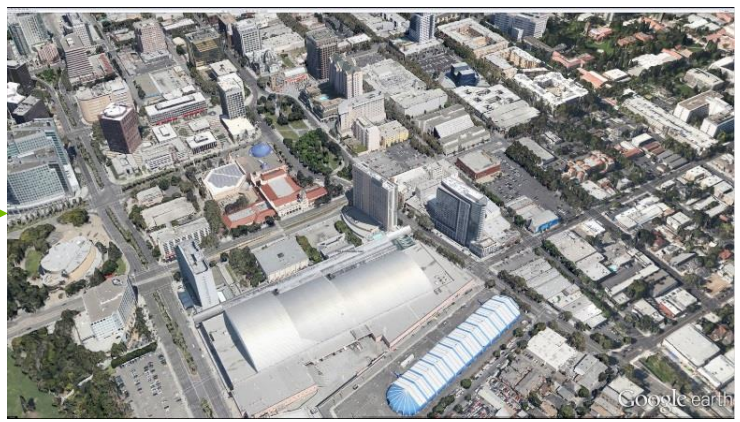
Create a Seamless Desktop

Supported on all Quadro GPUs
Supported in single and multi-GPU configurations

Without Mosaic



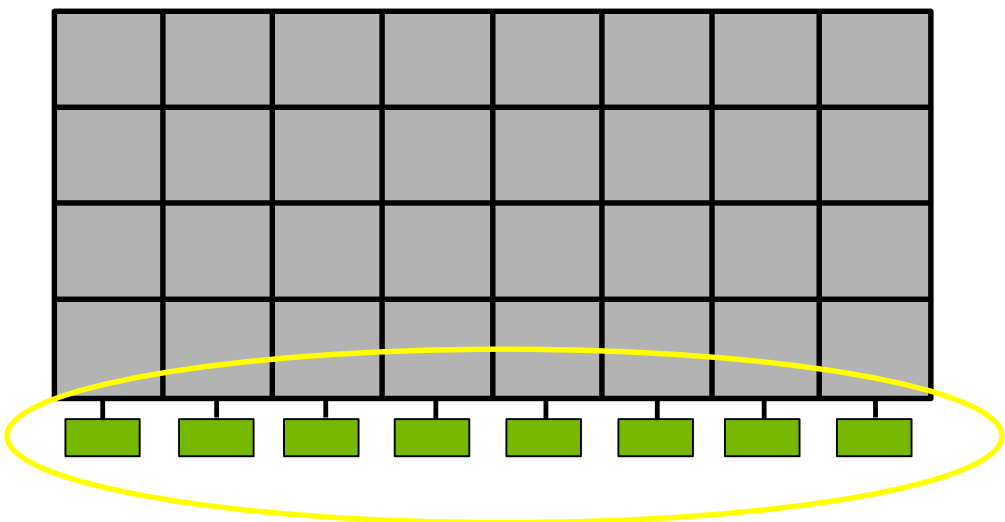
With Mosaic



MOSAIC

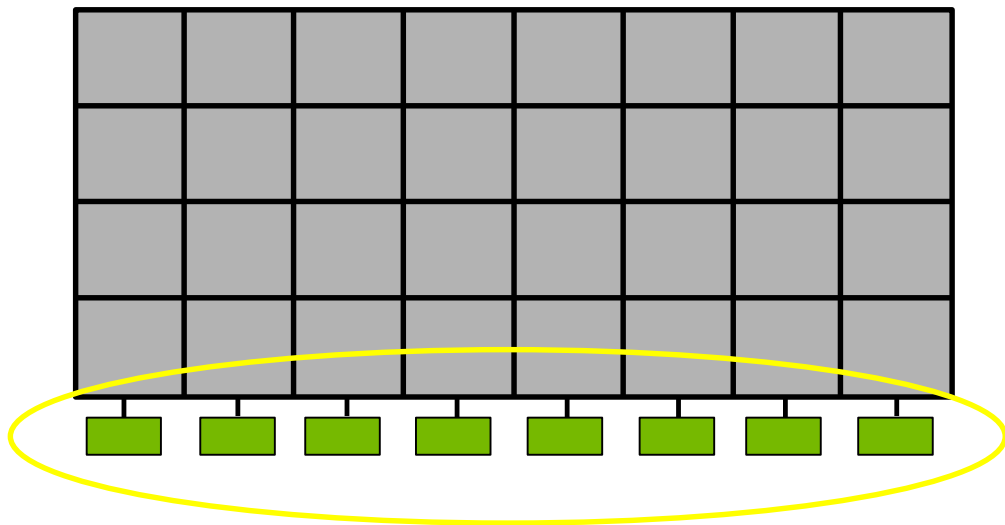
Creates a Single Logical GPU

Without Mosaic



8 Physical GPUs
8 Logical GPUs

With Mosaic

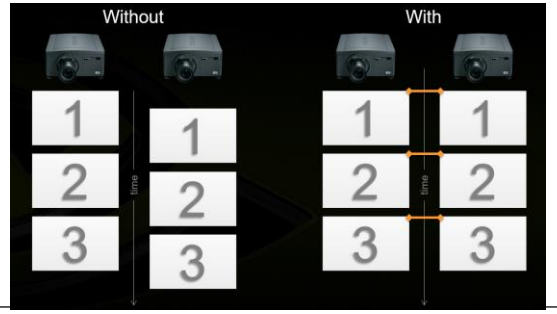


8 Physical GPUs
1 Logical GPU

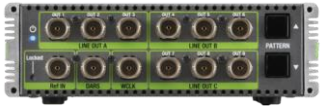
QUADRO SYNC II

Hardware Features Provide Tear-Free Mosaic Display

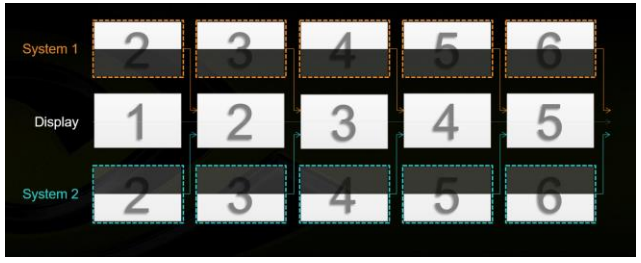
FRAMELOCK MULTIPLE DISPLAYS



EXTERNAL/HOUSE SYNC



SWAP SYNCHRONIZATION

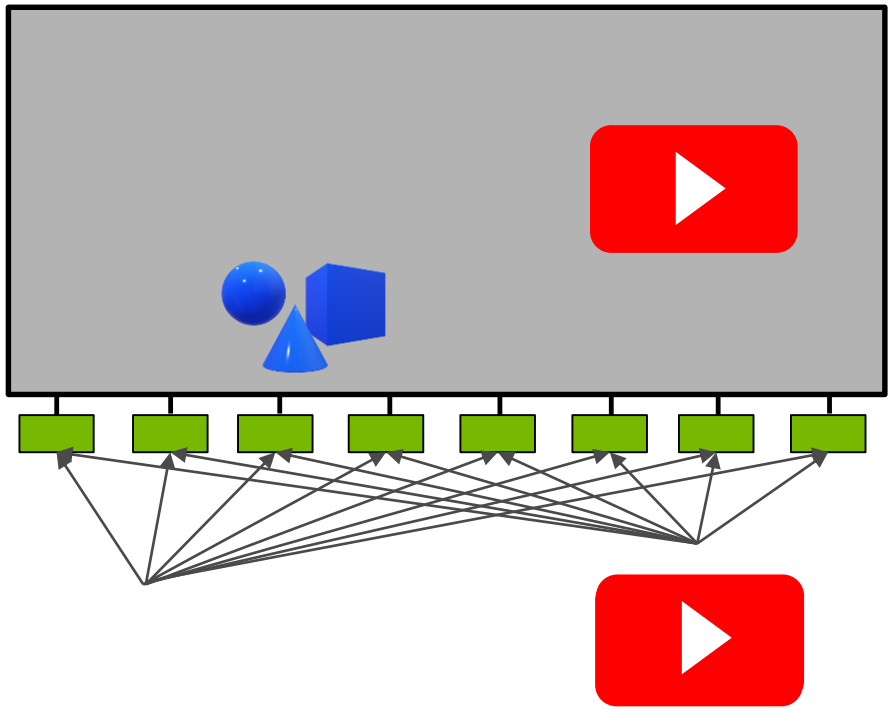


MOSAIC WITH SYNC

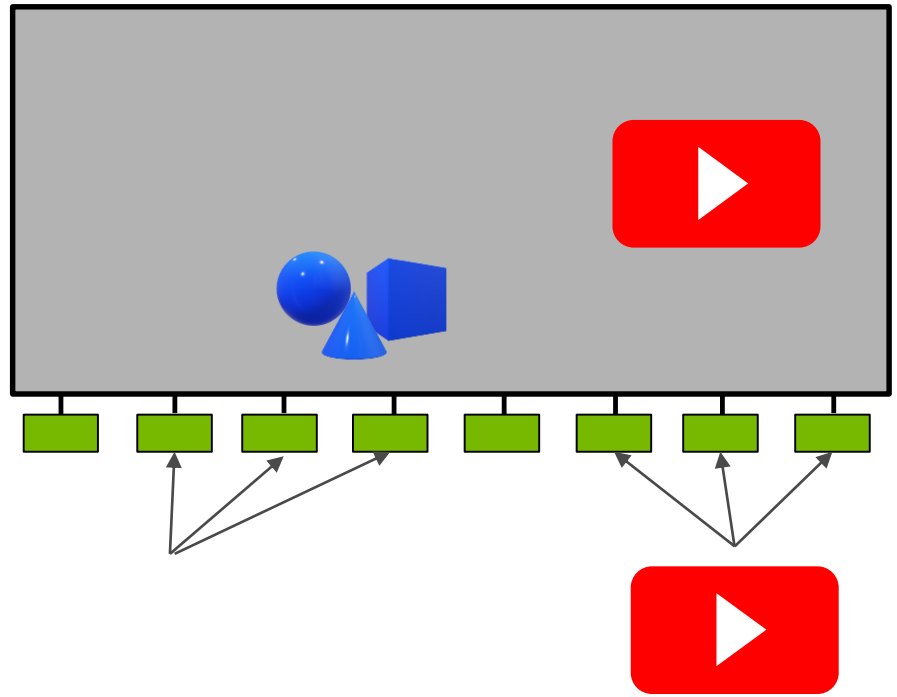
EFFICIENT RENDERING

Explicit GPU Addressing

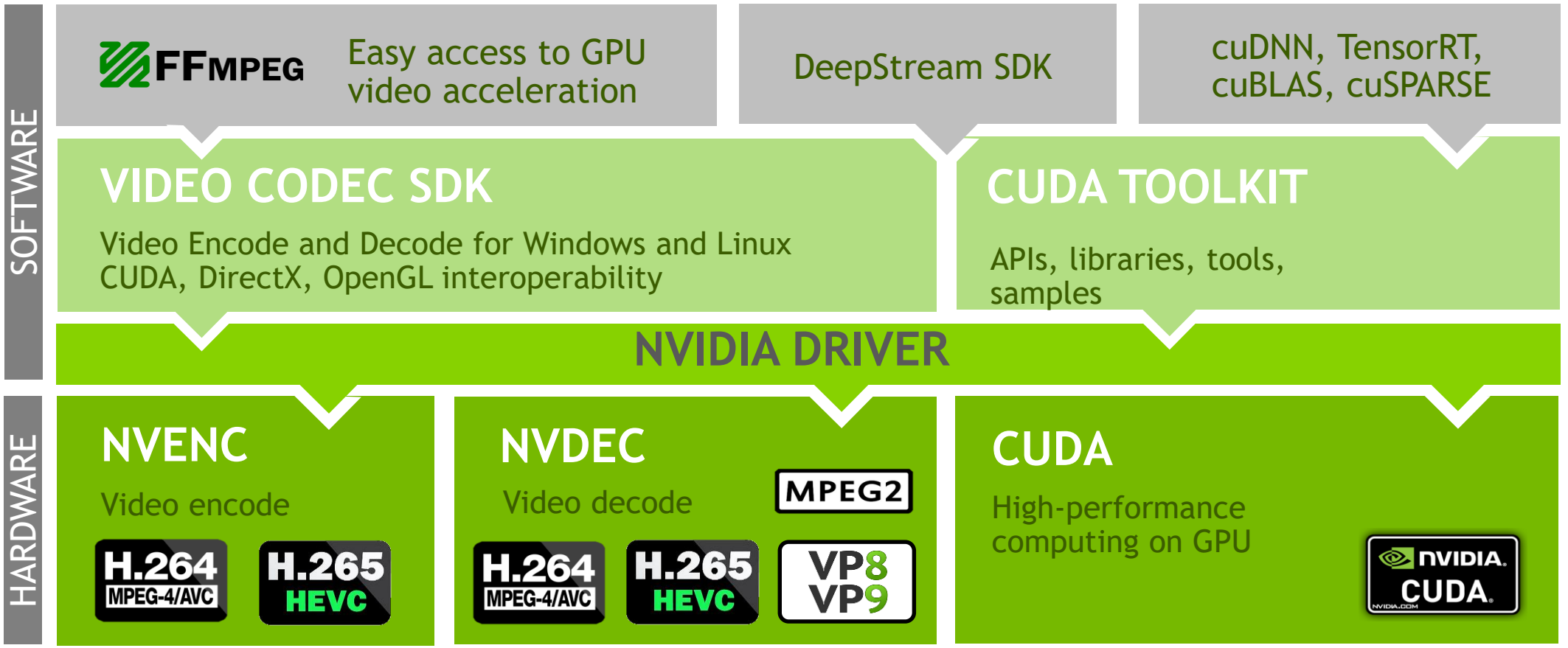
Without Directed Rendering



With Directed Rendering



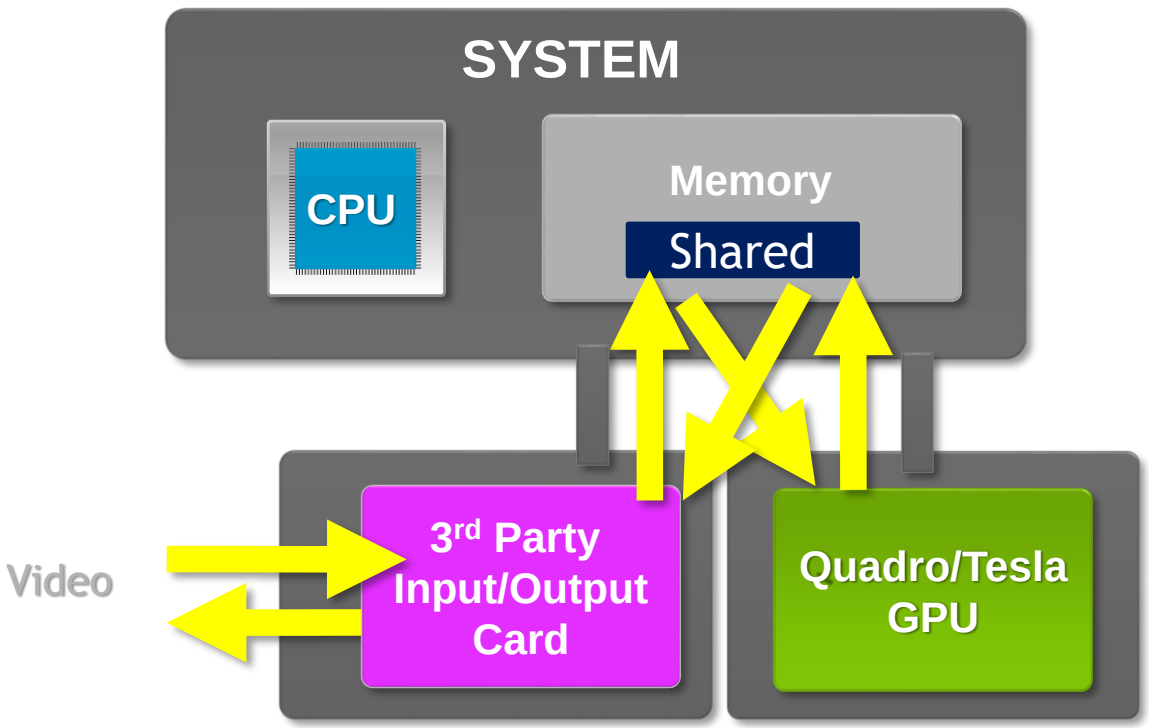
NVIDIA VIDEO CODEC SDK



S9331 – NVIDIA GPU Video Technologies: Overview, Applications and Optimization Techniques
Wednesday March 20, 2:00-2:50PM, Room 230C

GPU DIRECT FOR VIDEO

Video Transfers Through a Shareable System Memory Buffer

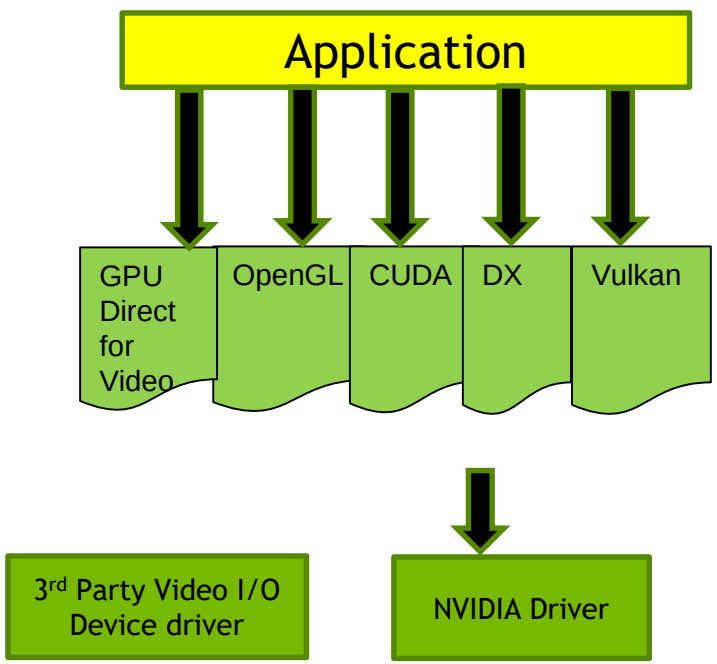


http://on-demand.gputechconf.com/siggraph/2016/video/sig1602-thomas-true-gpu-video-processing.mp4

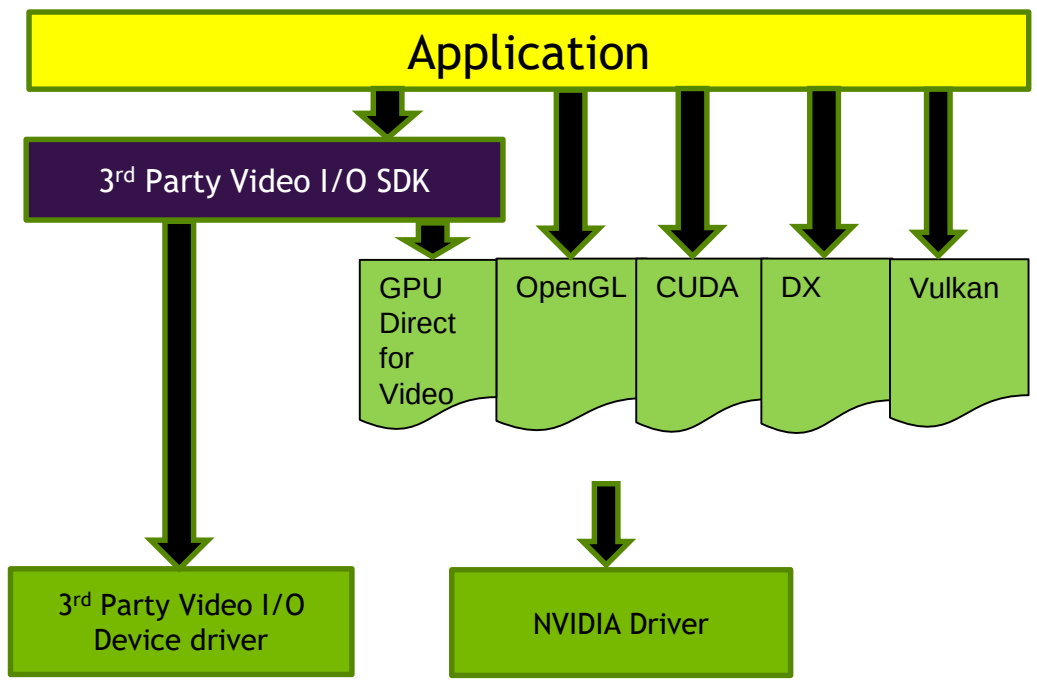
GPU DIRECT FOR VIDEO

Application Usage

Not This:



But This:



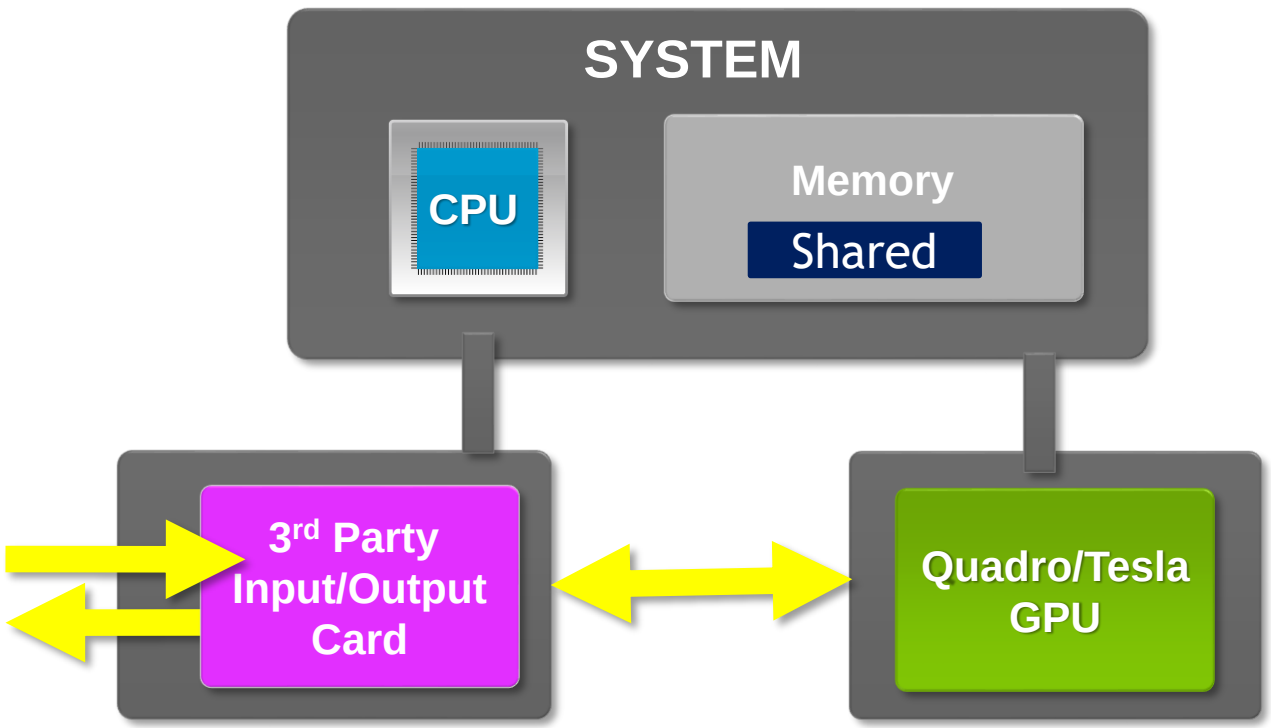
GPU DIRECT FOR VIDEO

Video Capture to OpenGL Texture

```
main()
{
    …..
    GLuint glTex;
    glGenTextures(1, &glTex);    \\ Create OpenGL texture obect
    glBindTexture(GL_TEXTURE_2D, glTex);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, bufferWidth, bufferHeight, 0, 0, 0, 0);
    glBindTexture(GL_TEXTURE_2D, 0);
    EXTRegisterGPUTextureGL(glTexIn);  \\ Register texture with 3rd party Video I/O SDK
    while(!quit)
    {
        EXTBegin(glTexIn);  \\ Release texture from Video I/O SDK
        Render(glTexIn);  \\ Use the texture
        EXTEnd(glTexIn);  \\ Release texture back to Video I/O SDK
    }
    EXTUnregisterGPUTextureGL(glTexIn);  \\ Unregister texture with 3rd party Video I/O SDK
}
```

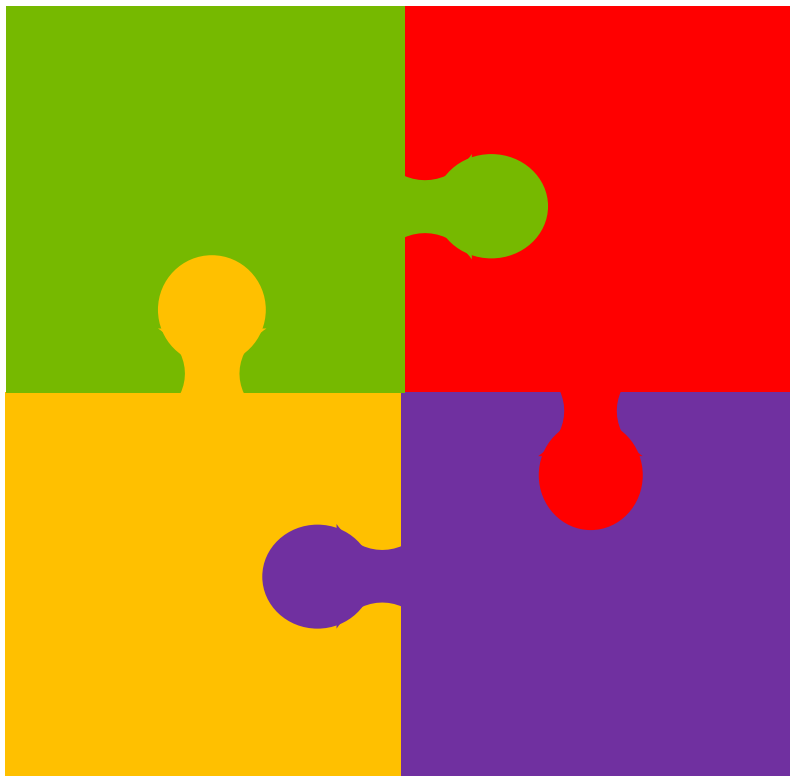
GPU DIRECT RDMA

Peer-to-Peer Video Transfers



https://docs.nvidia.com/cuda/gpudirect-rdma/index.html

PUTTING THE PIECES TOGETHER

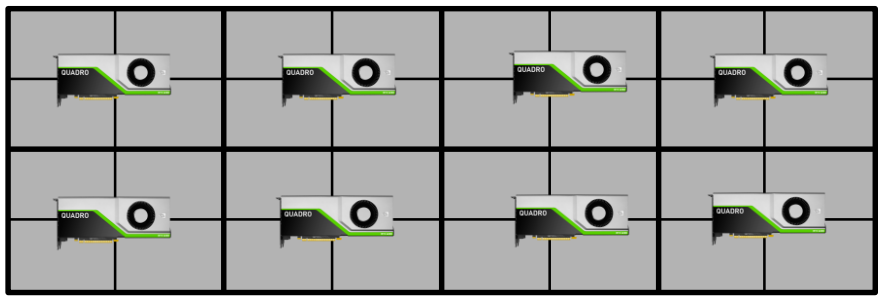


PUTTING THE PIECES TOGETHER

Application Steps to Success

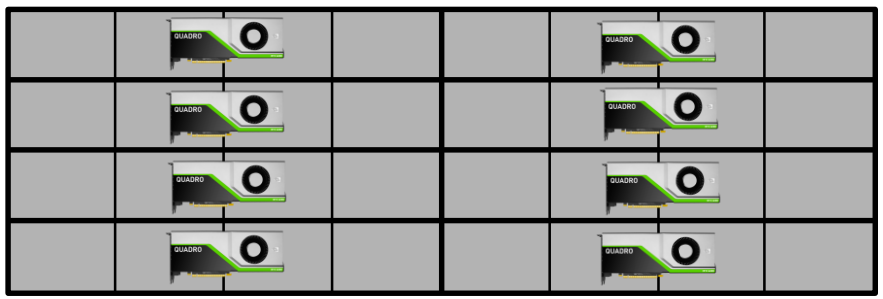
1. Design GPU-Display Topology to Optimize Locality
2. Single Full Screen Window with Multiple Viewports
3. Enumerate GPUs
4. Map GPUs to Displays
5. Perform Spatial Decomposition of Scene
6. Program Directed Compute
7. Program Directed Rendering
8. Swap / Present

DESIGN TOPOLOGY TO OPTIMIZE LOCALITY



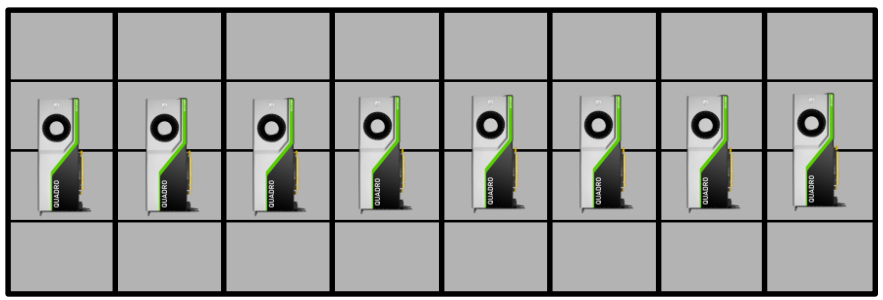
Quadrants

For Rectangular Content



Stripes

For Horizontal Content



Columns

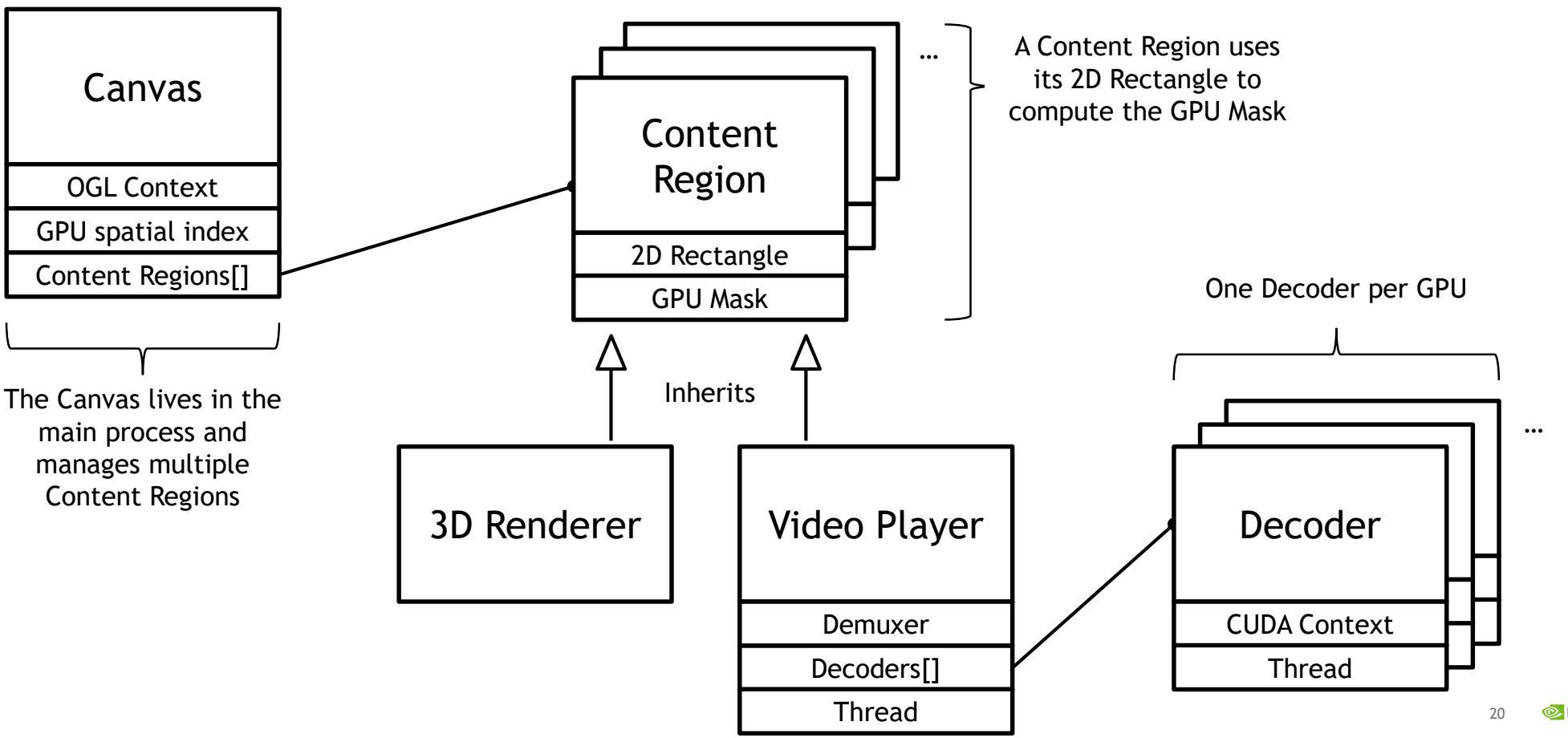
For Vertical Content

APPLICATION ARCHITECTURE

Full Screen Window with Content Regions

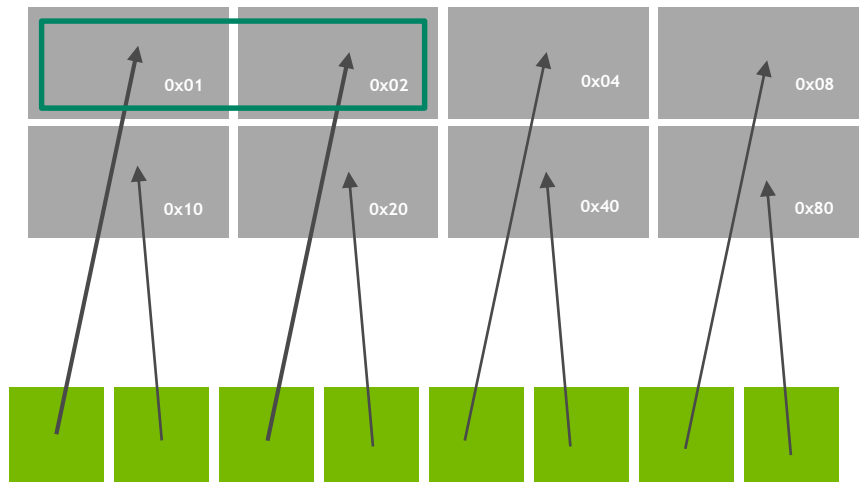
EXAMPLE SOFTWARE ARCHITECTURE

Mixed 3D and Video Content



MAPPING CONTENT REGIONS TO GPUS

Spatial Indexing



1. Query each GPU's pixel region
2. Store the regions in an index, e.g.:
   a) Flat list
   b) Quadtree
   c) R-Tree
3. For each content region
   a) Use the index to determine which GPUs are intersected
   b) Decode only on these GPUs
   c) Render only on these GPUs
   d) If the content region moves, re-query the index

0x01 | 0x02 = 0x03

GPU ENUMERATION

Windows NVAPI

```
// Enumerate Physical GPUs
NvU32 numPhysGpus = 0;
NvPhysicalGpuHandle nvGpuHandles[NVAPI_MAX_PHYSICAL_GPUS];
NvAPI_EnumPhysicalGPUs( numPhysGpus, &nvGpuHandles );
```

Mosaic will report all GPUs

```
// Enumerate Logical GPUs
NvU32 numLogiGpus = 0;
NvLogicalGpuHandle nvGpuHandles[NVAPI_MAX_LOGICAL_GPUS];
NvAPI_EnumLogicalGPUs( numLogiGpus, &nvGpuHandles );
```

Mosaic will report one GPU

https://developer.nvidia.com/nvapi

MAPPING LOGICAL GPUS TO PHYSICAL GPUS

Windows NVAPI

```
// Enumerate Logical GPUs
NvU32 numLogiGpus = 0;
NvLogicalGpuHandle nvGpuHandles[NVAPI_MAX_LOGICAL_GPUS];
NvAPI_EnumLogicalGPUs( numLogiGpus, &nvGpuHandles )

 // Map Logical GPUs to Physical GPUs
 for (NvU32 index = 0; index < numLogiGPUs; index++)
 {
     NV_LOGICAL_GPU_DATA logiGPUData = { 0 };
     logiGPUData.version = NV_LOGICAL_GPU_DATA_VER;
     logiGPUData.pOSAdapterId = malloc(sizeof(LUID));
     NvAPI_GPU_GetLogicalGpuInfo(nvGpuHandles[index], &logiGPUData);
 }
```

https://developer.nvidia.com/nvapi

MAPPING PHYSICAL GPUS TO DISPLAYS

Windows NVAPI

```
// Get connected display IDs for each GPU
NvU32 conDispIdCnt[NVAPI_MAX_PHYSICAL_GPUS] = { 0 };
NV_GPU_DISPLAYIDS *pConDispIds[NVAPI_MAX_PHYSICAL_GPUS];
NvU32 flags = NV_GPU_CONNECTED_IDS_FLAG_UNCACHED | NV_GPU_CONNECTED_IDS_FLAG_SLI |
              NV_GPU_CONNECTED_IDS_FLAG_FAKE;
for (NvU32 index = 0; index < numPhysGpus; index++)
{
    NvAPI_GPU_GetConnectedDisplayIds(nvGPUHandle[index], NULL, &conDispIdCnt[index], flags);
    if (conDispIdCnt[index])
    {
        pConDispIds[index] = (NV_GPU_DISPLAYIDS*)calloc(conDispIdCnt[index],
                                                        sizeof(NV_GPU_DISPLAYIDS));
        pConnectedDisplayIds[index]->version = NV_GPU_DISPLAYIDS_VER;
        NvAPI_GPU_GetConnectedDisplayIds(nvGPUHandle[index], pConDispIds[index],
                                         &conDispIdCnt[index], flags);
    }
}
```

https://developer.nvidia.com/nvapi

MAPPING DISPLAYS TO SCREEN AREA

Windows NVAPI

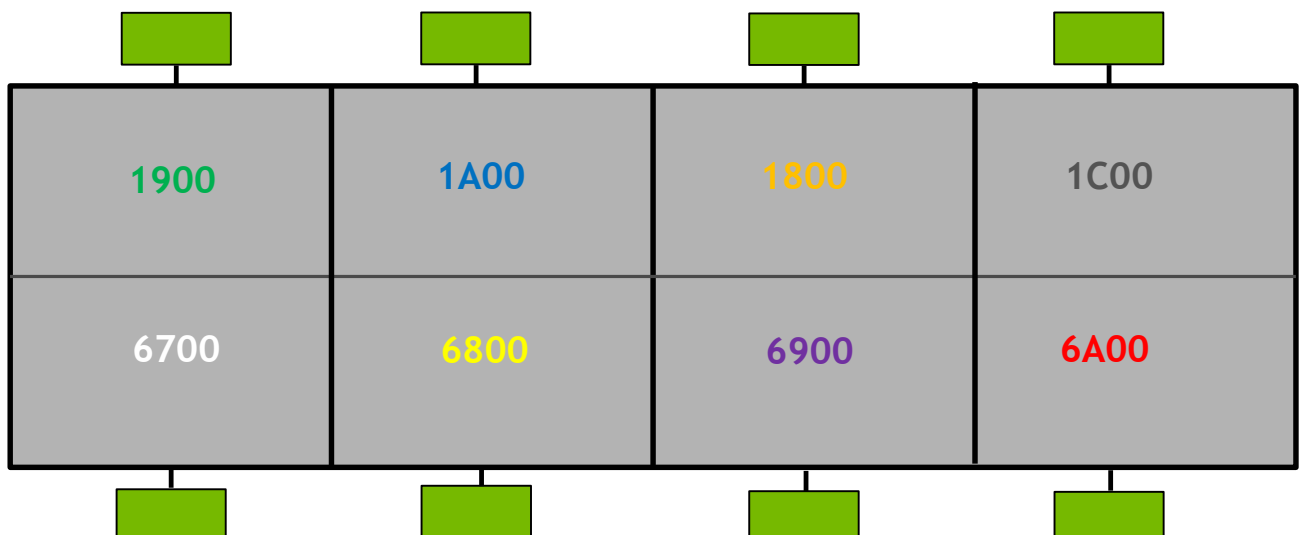
```
// Get screen coordinates for each connected display for each GPU
for (NvU32 index = 0; index < numPhysGpus; index++)
{
    for (NvU32 display = 0; display < nvConnectedDisplayIdCount[index]; display++)
    {
        NvSBox dRect = { 0 }; // Desktop rect
        NvSBox sRect = { 0 }; // Scanout rect
        NvAPI_GPU_GetScanoutConfiguration(pConnectedDisplayIds[index][display].displayID,
                                          &dRect, &sRect);
    }
}
```

https://developer.nvidia.com/nvapi

MAPPING PHYSICAL GPUS TO DISPLAYS

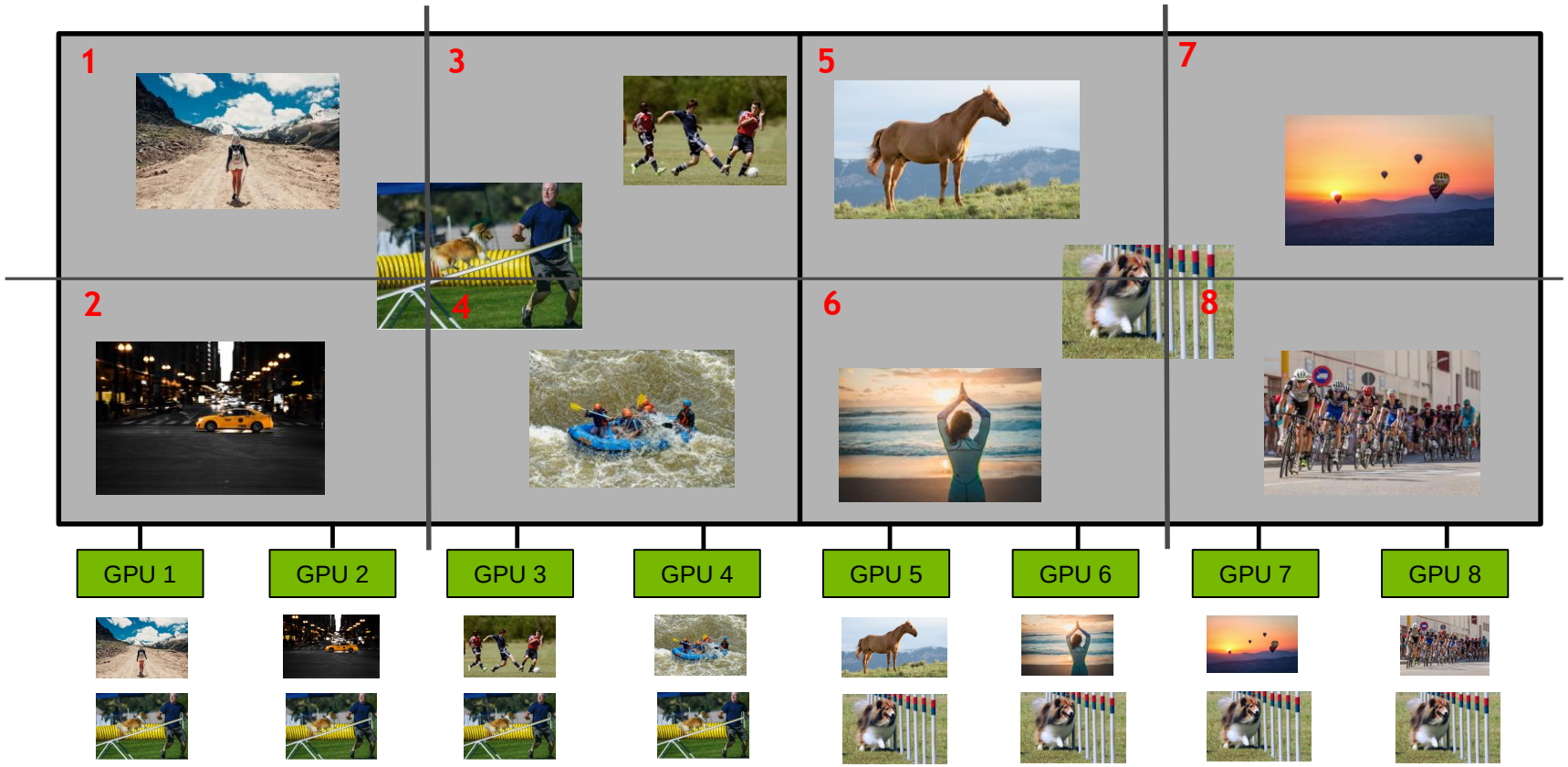
Windows NVAPI

```
NVAPI Physical GPU to Display Mapping:
--------------------------------------
Number of Physical GPUs: 8
GPU: 0000000000006A00 Bus: 106 Slot: 0 Name: Quadro RTX 6000 Desktop Rect: (5760,1200,1920,1200)
GPU: 0000000000001900 Bus: 25 Slot: 0 Name: Quadro RTX 6000 Desktop Rect: (0,0,1920,1200)
GPU: 0000000000001A00 Bus: 26 Slot: 0 Name: Quadro RTX 6000 Desktop Rect: (1920,0,1920,1200)
GPU: 0000000000001C00 Bus: 28 Slot: 0 Name: Quadro RTX 6000 Desktop Rect: (5760,0,1920,1200)
GPU: 0000000000006700 Bus: 103 Slot: 0 Name: Quadro RTX 6000 Desktop Rect: (0,1200,1920,1200)
GPU: 0000000000006900 Bus: 105 Slot: 0 Name: Quadro RTX 6000 Desktop Rect: (3840,1200,1920,1200)
GPU: 0000000000001B00 Bus: 27 Slot: 0 Name: Quadro RTX 6000 Desktop Rect: (3840,0,1920,1200)
GPU: 0000000000006800 Bus: 104 Slot: 0 Name: Quadro RTX 6000 Desktop Rect: (1920,1200,1920,1200)

NVAPI Logical GPU to Physical GPU Mapping:
------------------------------------------
index: 0 OS Adapter ID: 00000209F7E74370 Physical GPU Count: 8 Physical GPU Handles: 0000000000006800 0000000000006A00 0000000000001900 0000000000001A00 0000000000001C00 0000000000006700 0000000000006900 0000000000001B00

Press any key to continue . . .
```



SPATIAL MAPPING

Dividing the Workload Among the Physical GPUs



DIRECTED COMPUTE

Explicit GPU Programming

```
// Enumerate CUDA GPUs
int numGPUs;
CK_CUDA(cudaGetDeviceCount(&numGPUs));

// Get PCI bus ID and device ID for each GPU
std::vector<int> busIDList(numGPUs);  // Bus IDs
std::vector<int> devIDList(numGPUs);  // Device IDs
for (int i = 0; i < numGPUs; i++)
{
    CK_CUDA(cudaDeviceGetAttribute(&busIDList[i], cudaDevAttrPciBusId, i));
    CK_CUDA(cudaDeviceGetAttribute(&devIDList[i], cudaDevAttrPciDevId, i));
}

// Match PCI bus ID and device ID to those returned from NVAPI

// Set CUDA device to matched GPU
CK_CUDA(cudaSetDevice(matchedGPU));
```

DIRECTED RENDERING

OpenGL: Don't Use GPU Affinity

Application must:
1. Manage multiple GPU Context
2. Multi-pump the API

Enumerate GPUs:

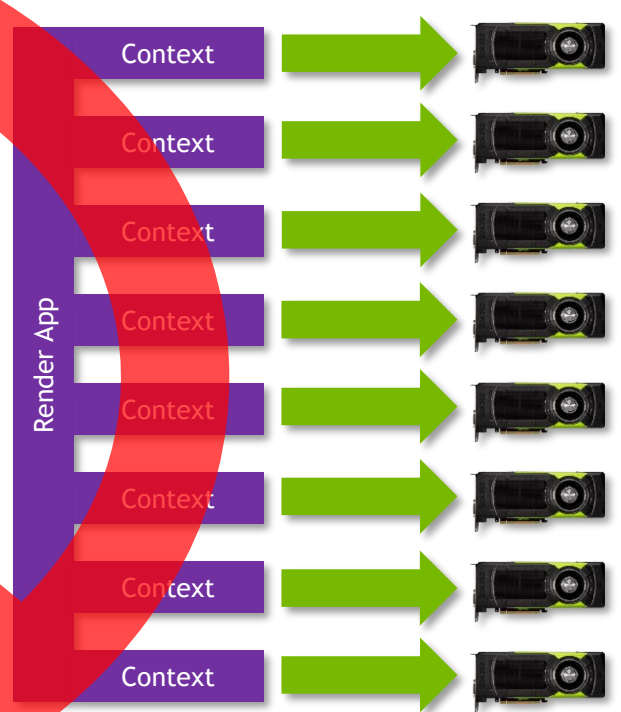
```
wglEnumGpusNV( UINT iGPUIndex, HGPUNV* phGPU );
```

Enumerate displays per GPU:

```
wglEnumGpuDevicesNV( HGPUNV hGPU, UINT iDeviceIndex,
                     PGPU_DEVICE lpGpuDevice );
```

Create an OpenGL context for a specific GPU:

```
HGPUNV gpuMask[2] = {phGPU, nullptr};
HDC affinityDc = wglCreateAffinityDCNV( gpuMask );
SetPixelFormat( affinityDc, ... );
HGLRC affinityGlrc = wglCreateContext( affinityDc );
```

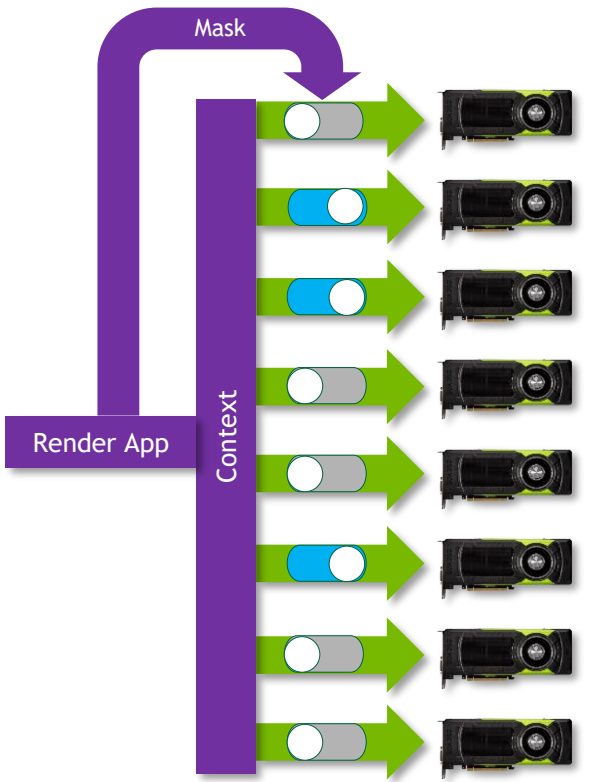


https://www.khronos.org/registry/OpenGL/extensions/NV/WGL_NV_gpu_affinity.txt

DIRECTED RENDERING

OpenGL: Use NV_gpu_multicast

```
// Enable OpenGL Multicast Extension
SetEnvironmentVariable(L"GL_NV_GPU_MULTICAST", L"1");

// Enumerate Multicast GPUs
GLint numMulticastGPUs;
glGetIntegerv(GL_MULTICAST_GPUS_NV, &numMulticastGPUs);
maskAllGPUs = 0;
for (int i = 0; i < numMulticastGPUs; ++i)
    m_maskAllGPUs |= 1 << i;

if (numMulticastGPUs > 1)
    LOG(LogLevel::INFO) << "System is multicast-enabled.";


// Render on Specific GPU
glRenderGpuMaskNv(GPUmask);
```



https://www.khronos.org/registry/OpenGL/extensions/NV/NV_gpu_multicast.txt

DIRECTED RENDERING

More OpenGL Multicast Functionality

Modify Buffer Object Data on One or More GPUs:

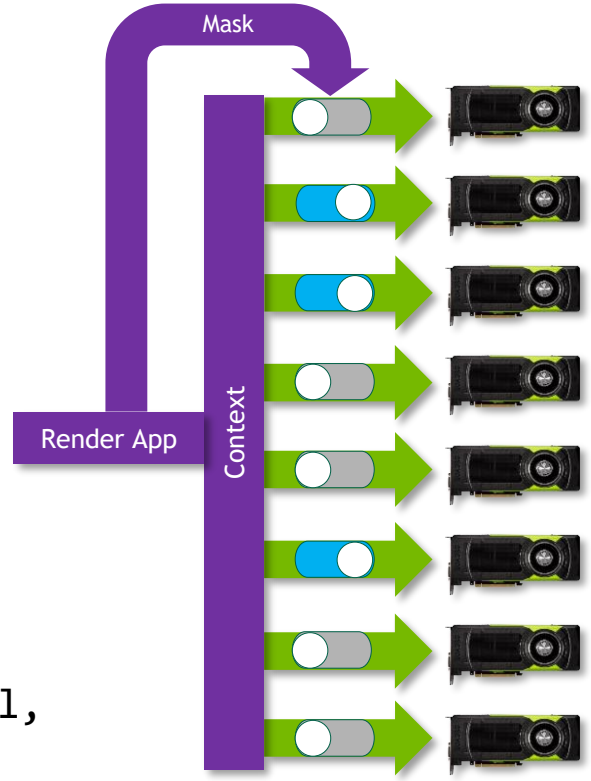
```
glMulticastBufferSubDataNV(GPUmask, buffer, offset, size, data);
```

Copy Between Buffers:

```
glMulticastCopyBufferSubDataNV(readGPUmask, writeGPUmask,
                               readBuffer, writeBuffer,
                               readOffset, writeOffset,
                               size);
```

Copy Image Data Between GPUs:

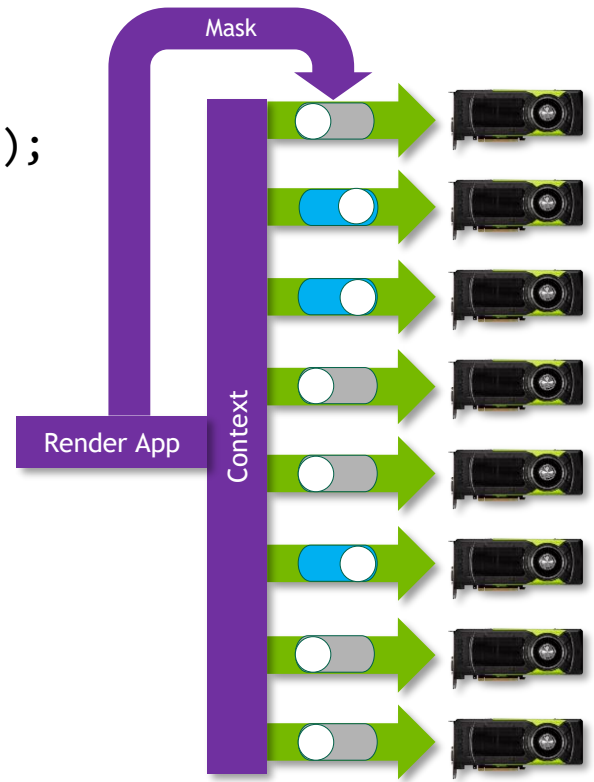
```
glMulticastCopyImageSutDataNV(srcGPUmask, writeGPUmask,
                              srcName, srcTarget, srcLevel,
                              srcX, srcY, srcZ,
                              dstName, dstTarget, dstLevel,
                              dstX, dstY, dstZ,
                              srcWidth, srcHeight, srcDepth);
```



DX12

Explicit GPU Programming

```
// Create D3D12 Device from DXGI Adapter
UINT dxgiFactoryFlags = 0;
ComPtr<IDXGIFactory4> factory;
CreateDXGIFactory2(dxgiFactoryFlags, IID_PPV_ARGS(&factory));

ComPtr<IDXGIAdapter1> adapter;
GetHardwareAdapter(factory.Get(), &adapter);

ComPtr<ID3D12Device> device;
D3D12CreateDevice(adapter.Get(), D3D_FEATURE_LEVEL_11_0,
                  IID_PPV_ARGS(&device));

// Enumerate Linked Adapter GPUs
UINT numMulticastGPUs = pDevice->GetNodeCount();
UINT maskAllGPUs = 0;
for (int i = 0; i < numMulticastGPUs; ++i)
    m_maskAllGPUs |= 1 << i;

if (numMulticastGPUs > 1)
    LOG(LogLevel::INFO) << "System is multicast-enabled.";
```



DIRECTED RENDERING

DX12 Linked Adapter Functionality

Create Command Queue on Single GPU:

```
CreateCommandQueue(desc, riid, &cmdQueue);
```

Create a Command List on a Single GPU:

```
CreateCommandList(nodeMask, type, cmdAllocater,
                  initialState, riid, &cmdList);
```

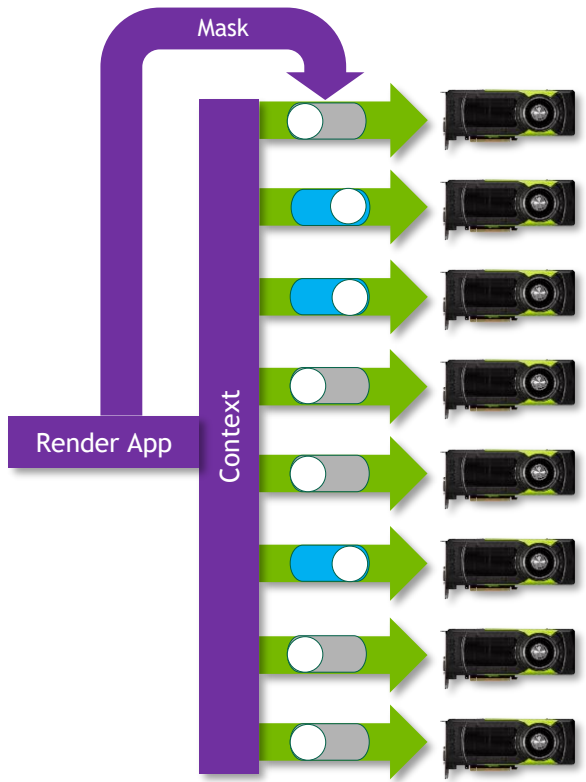
Create Graphics Pipeline State on Multiple GPUs:

```
CreateGraphicsPipelineState(desc, riid, &pipelineState);
```

Create Compute Pipeline State on Multiple GPUs:

```
CreateComputePipelineState(desc, riid, &pipelineState);
```

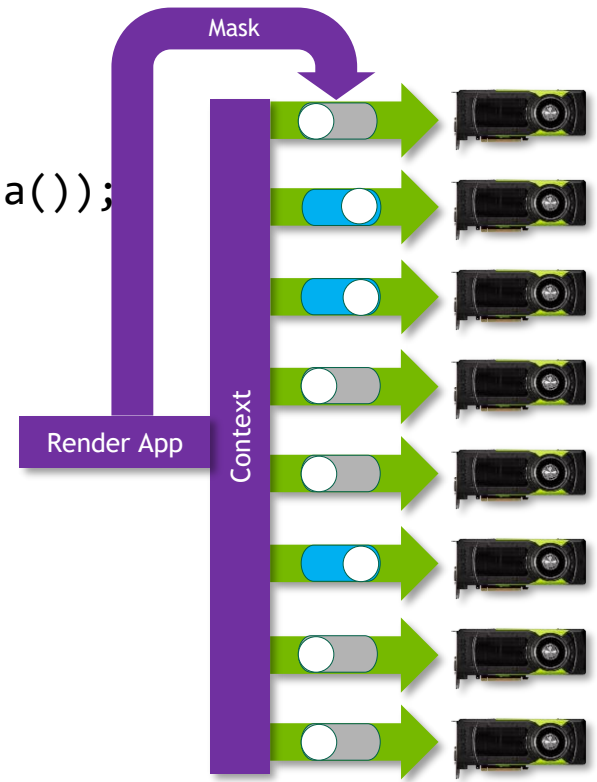
https://docs.Microsoft.com/en-us/windows/desktop/direct3d12/multi-engine



VULKAN

Explicit GPU Programming

```
// Enumerate Physical Device Groups
uint32_t count = 0;
vkEnumeratePhysicalDeviceGroups(instance, &count, nullptr);
std::vector<VkPhysicalDeviceGroupProperties> props(count);
vkEnumeratePhysicalDeviceGroups(instance, &count, props.data());

// Build Device Mask
uint32_t maskAllGPUs = 0;
for (uint32_t i = 0; i < count; i++)
{
    maskAllGPUs |= 1 << i;
}

if (maskAllGPUs > 1)
    LOG(LogLevel::INFO) << "System is multicast-enabled.";
```

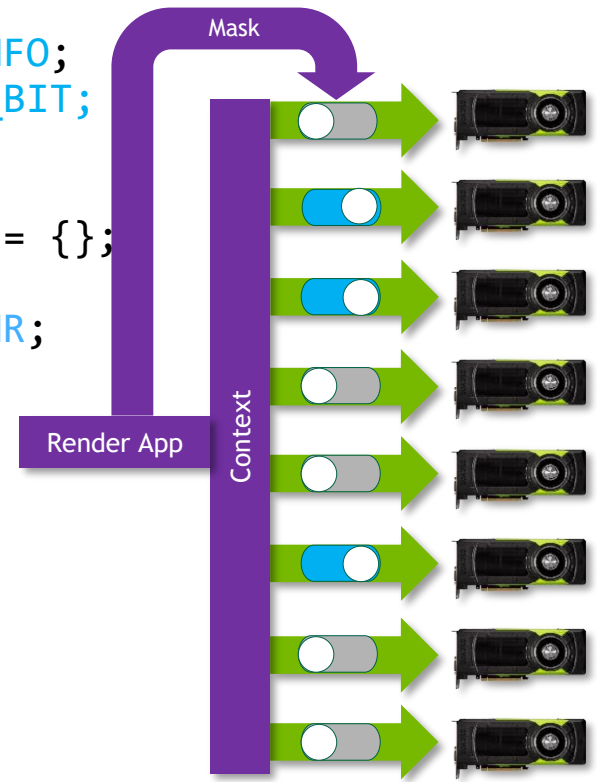


DIRECTED RENDERING

Specify Device Mask to Command Buffer

```
vkCommandBufferBeginInfo beginInfo = {};
beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
beginInfo.flags = VK_COMMAND_BUFFER_USAGE_SIMULTANEOUS_USE_BIT;

// VK_KHR_device_group
VkDeviceGroupCommandBufferBeginInfoKHR devicGroupBeginInfo = {};
devicGroupBeginInfo.sType =
VK_STRUCTURE_TYPE_DEVICE_GROUP_COMMAND_BUFFER_BEGIN_INFO_KHR;

// Limit this command buffer to GPU 0
devicGroupBeginInfo.deviceMask = 0b0000'0001;
beginInfo.pNext = &devicGroupBeginInfo;

vkBeginCommandBuffer(cmdBuffer, &beginInfo);
```

```
// Update the device mask of a command buffer
vkCmdSetDeviceMask(cmdBuffer, deviceMask);
```



PER-GPU RESOURCE ALLOCATION AND UPDATES

OpenGL

- GPU-shared storage unless PER_GPU_STORAGE_BIT_NV flag specified to glBufferStorage()
- Use **glMulticastSubBufferDataNV()** to update on specific GPU according to device mask

DX12 / Vulkan

- Memory allotted on each GPU
- Buffer created / updated according to device mask

CASE STUDY

Multi-GPU Video Compositor

MULTI-GPU VIDEO COMPOSITOR

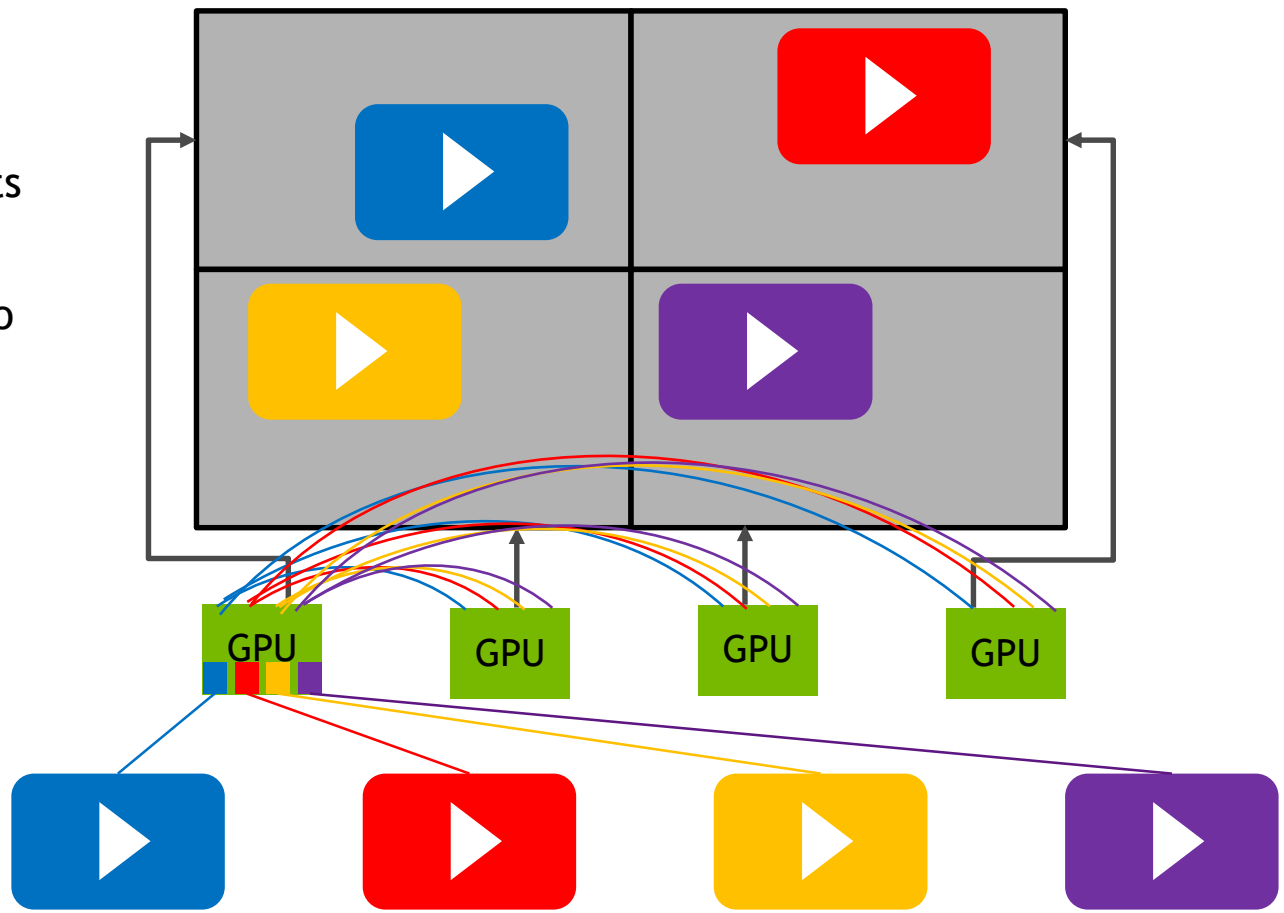
Naïve Approach: Single GPU Decode = PCIE Transfers to All GPUs

No Mosaic

- Video display cannot cross display boundaries.
- Requires multiple rendering contexts

Single GPU Decode

- PCIe transfer of uncompressed video frames to each GPU.
- Decoder can become a bottleneck.



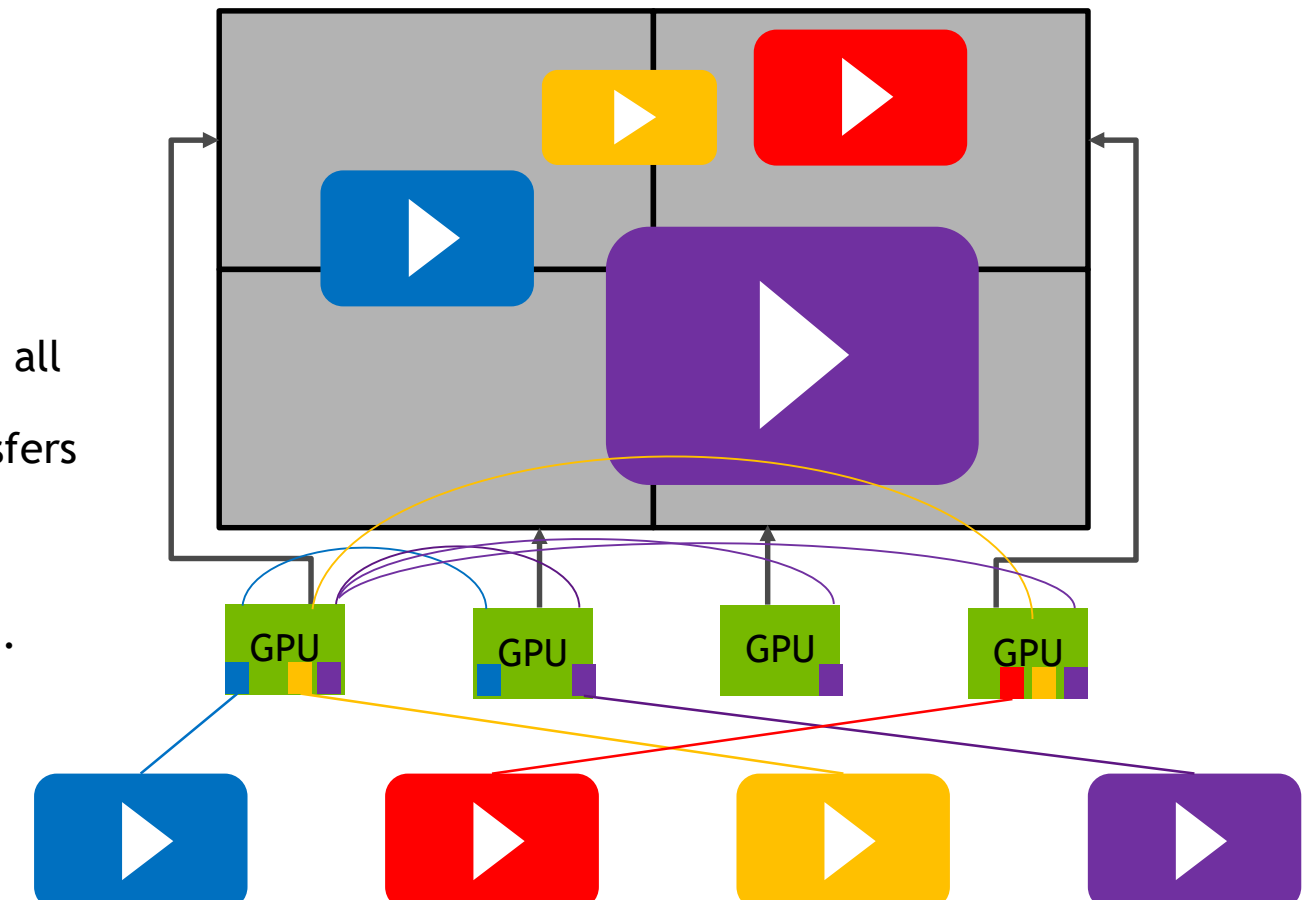
MULTI-GPU VIDEO COMPOSITOR

Optimized Approach: Application-Managed Peer-to-Peer Data Movement

Mosaic
- Single display. Easier application management.
- Video display can cross display boundaries.

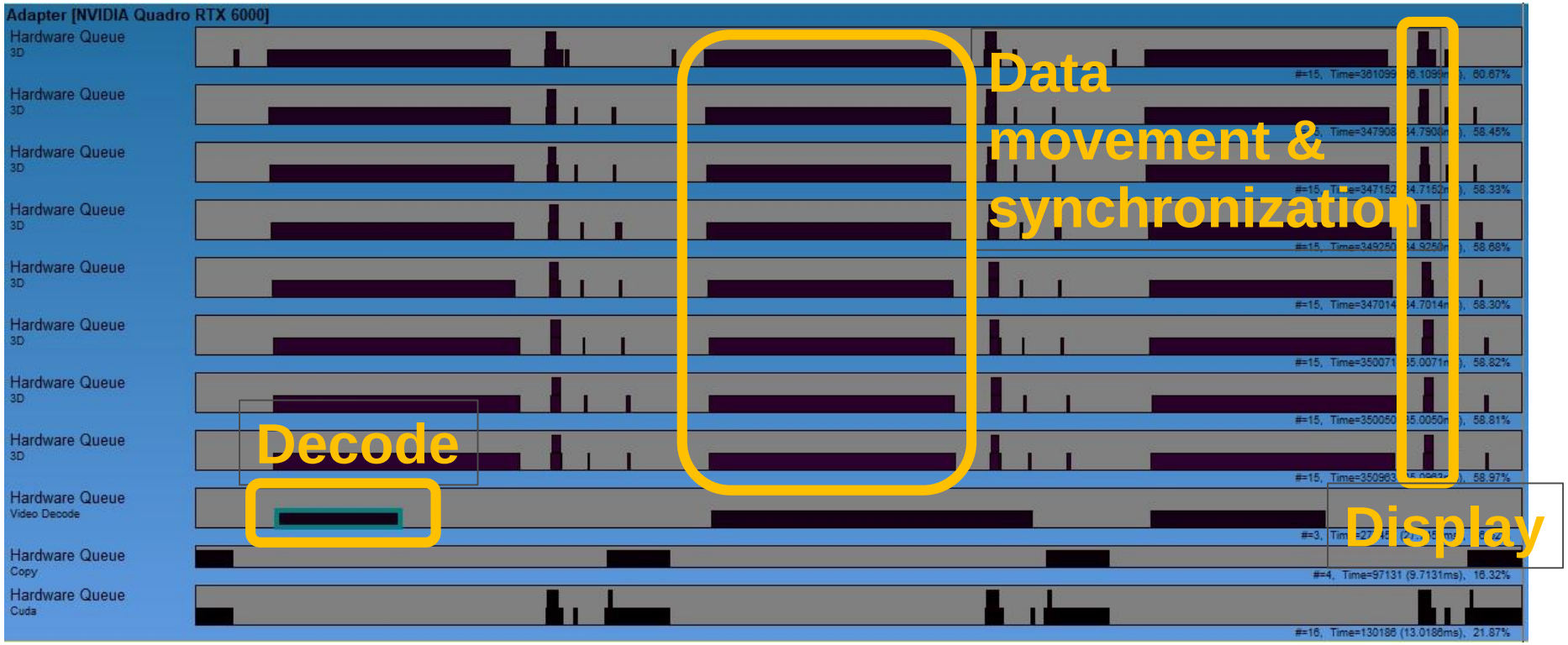
Multicast
- Single rendering context can span all GPUs / displays.
- Eliminates unnecessary data transfers and duplication to all GPUs.

Multi-GPU Decode
- Distributes decode to display GPU.
- Eliminates PCIe data transfers.
- Eliminates potential decoder bottleneck.
- Parallel decoding.



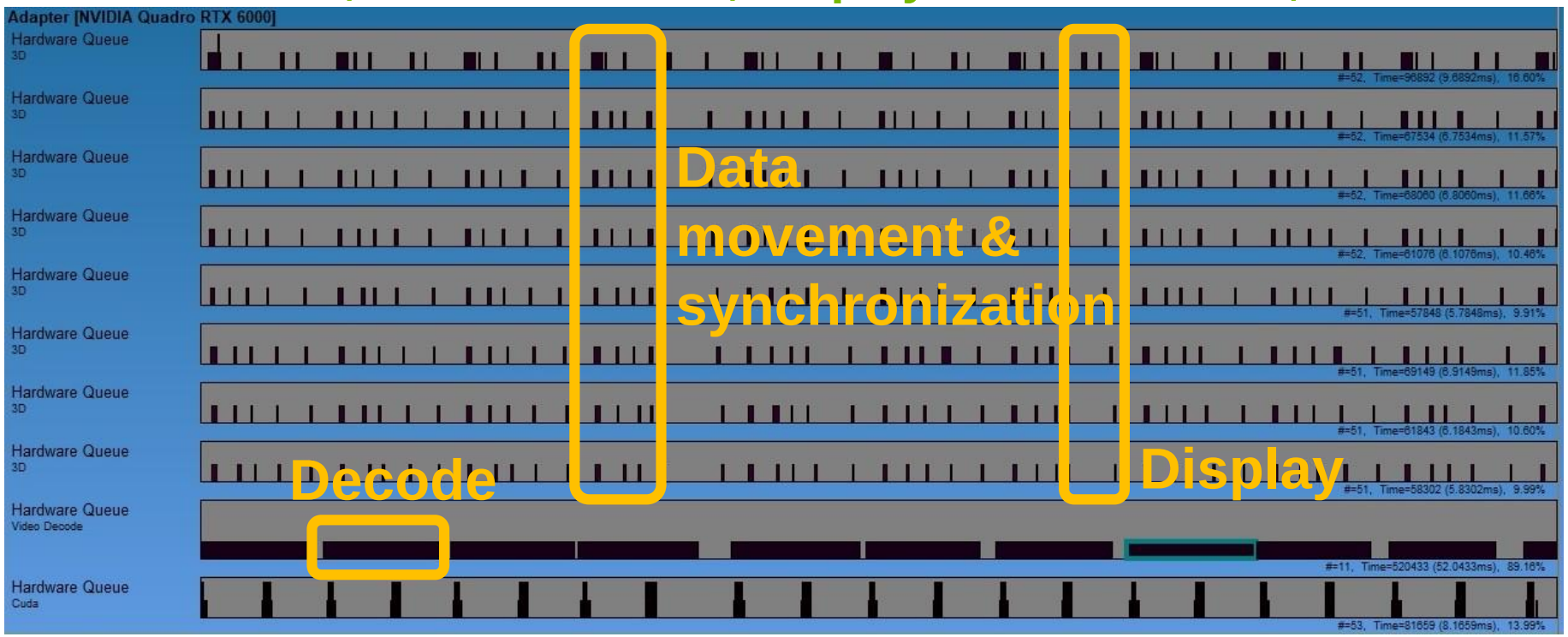
DIRECTED TEST RESULT

No. Streams=1, Decode=1 GPU, Display=8 GPU Mosaic, Multicast=Off



DIRECTED TEST RESULT

No. Streams=1, Decode=1 GPU, Display=8 GPU Mosaic, Multicast=On

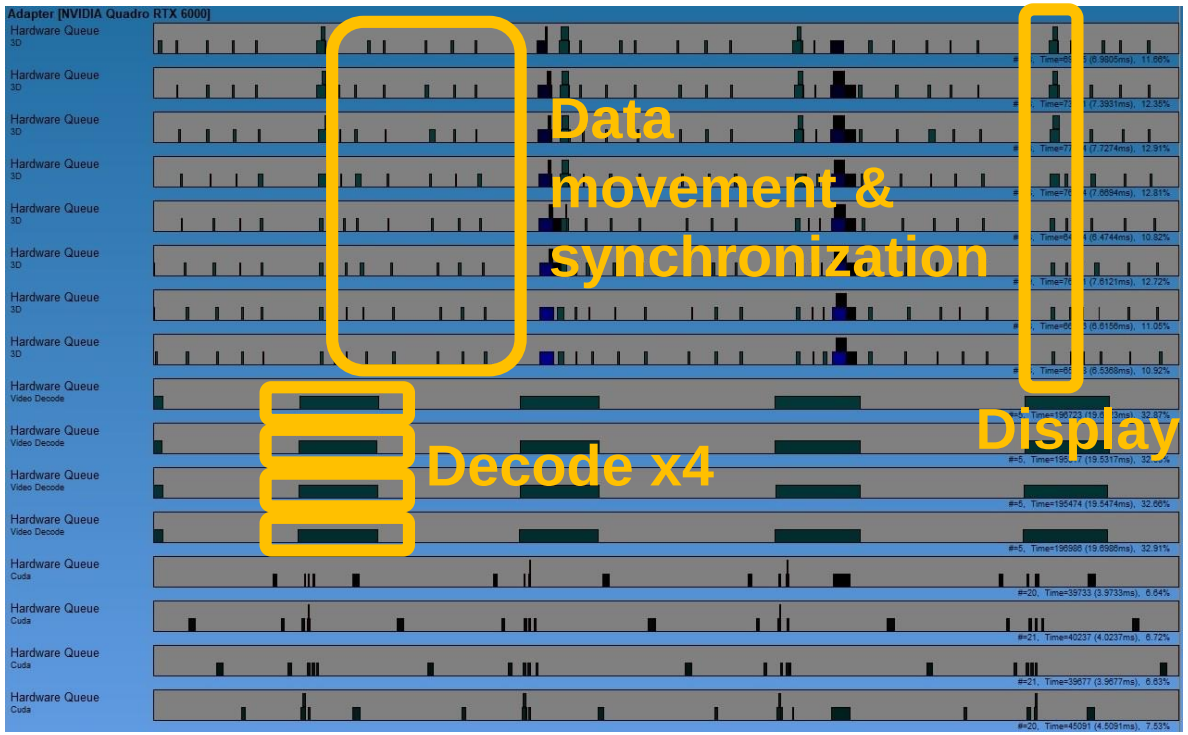


Frame Draw Time ~5ms

Trace Window ~60 ms

DIRECTED TEST RESULT

No. Streams=4, Decode=4 GPU, Display=8 GPU Mosaic, Multicast=On



Frame Draw Time ~12ms

Trace Window ~60 ms

IMPLEMENTATION DETAILS

Tying Up Some Loose Ends

R421 GA3 Driver Required – for NVAPI

Windows 10 RS5 – Unlimited Engines in Linked Adapter Mode (LDA)

Contact Quadro SVS alias to enable Multicast on Mosaic

Quadro SVS Email Alias: QuadroSVS@nvidia.com

MORE INFORMATION

Learn More / Connect With An Expert

S9331 – NVIDIA GPU Video Technologies: Overview, Applications and Optimization Techniques
Wednesday March 20, 2:00-2:50PM, Room 230C

CE9103 – Connect with the Experts: NVIDIA GPU Video Technologies: Video, Capture and Optical Flow SDK
Wednesday March 20, 3:00-4:00PM, Hall 3 Pod A

CE9128 – Connect with the Experts: NVIDIA Quadro Advanced Display Features
Thursday March 21, 11:00AM-12:00PM, Hall 3 Pod B


Q & A
NVIDIA