Scaling OpenGL Applications Across Multiple GPUs

Shalini Venkataraman
Senior Applied Engineer
NVIDIA PSG

Outline

- Default behaviour with multiple gpus
- Programming for scaling
  - Pinning OpenGL context to GPU
  - Application structure
  - Optimized inter-GPU transfers
- Applications
  - Multi-display environments eg CAVE, Powerwall
  - Large data visualization, parallel rendering
  - Server-side rendering and remoting
- Middleware

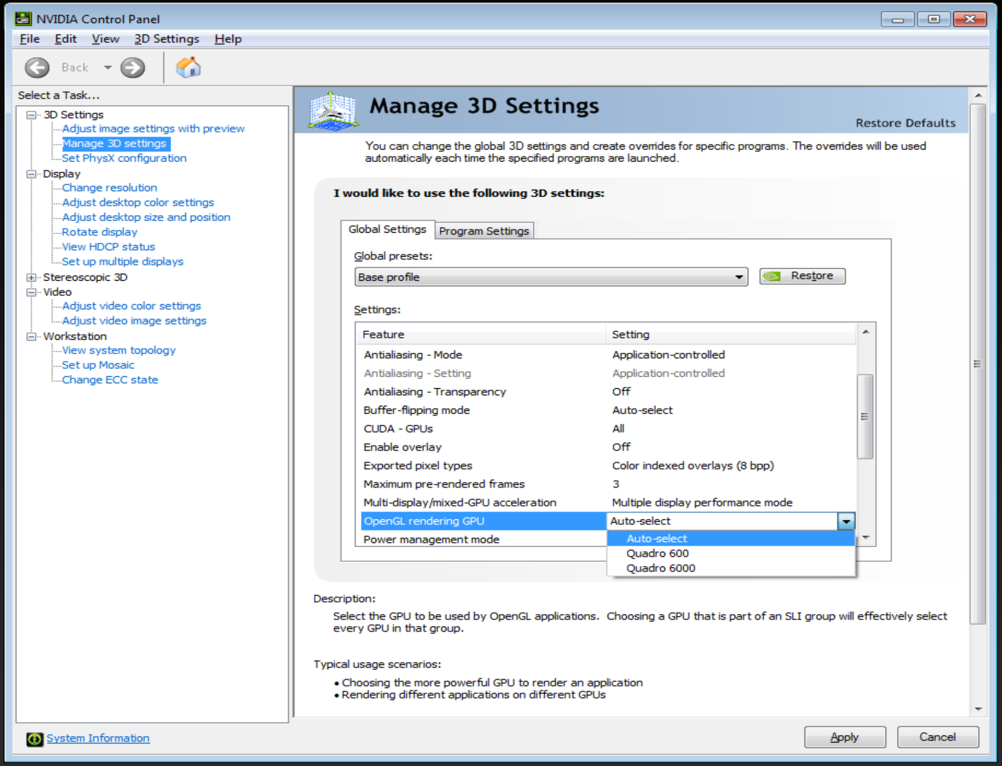
Multi-GPU - Transparent Behavior

- Default Behavior of OGL command dispatch
  - Win XP : Sent to all GPUs, slowest GPU gates performance
  - Linux : Only to the GPU attached to screen
  - Win 7: Sent to most powerful GPU and blitted across
- SLI AFR
  - Single threaded application
  - Data and commands are replicated across all GPUs

Specifying OpenGL GPU on NVIDIA Quadro

- Directed GPU Rendering
  - Quadro-only
  - Heuristics for automatic GPU selection
  - Allow app to pick the GPU for rendering, fast blit path to other displays
  - Programmatically using NVAPI or using CPL

    http://developer.nvidia.com/nvapi



Scaling Display - SLI Mosaic Mode

- Transparent
- Does frame synchronization
- Does fragment level clipping
- Disadvantages
  - Single view frustum
  - No geometry/vertex level clipping

CAVE system

Doug Trail, S0341-See the Big Picture Scalable Visualization Solutions for System Integrators, GTC 2012 Recordings

Programming for Scaling Rendering

- Focus on OpenGL graphics
- Onscreen Rendering
  - Display scaling for multi-projector, multi-tiled display environments
- Offscreen Parallel Rendering
  - Image Scaling - final image resolution
  - Data scaling - texture size, # triangles
  - Task/Process Scaling - eg render farm serving thin clients
- Amortize host resources across multiple GPUs

Programming for Multi-GPU

- Linux
  - Specify separate X screens using XOpenDisplay

```
Display* dpy = XOpenDisplay(":0."+gpu)
GLXContext = glxCreateContextAttribs(dpy,…);
```

  - Xinerama disabled
- Windows
  - Vendor specific extension
  - NVIDIA : NV_GPU_AFFINITY extension
  - AMD Cards : AMD_GPU_Association

GPU Affinity-
Enumerating and attaching to GPUs

- Enumerate GPUs

```
BOOL wglEnumGpusNV(UINT iGpuIndex, HGPUNV *phGPU)
```

- Enumerate Displays per GPU

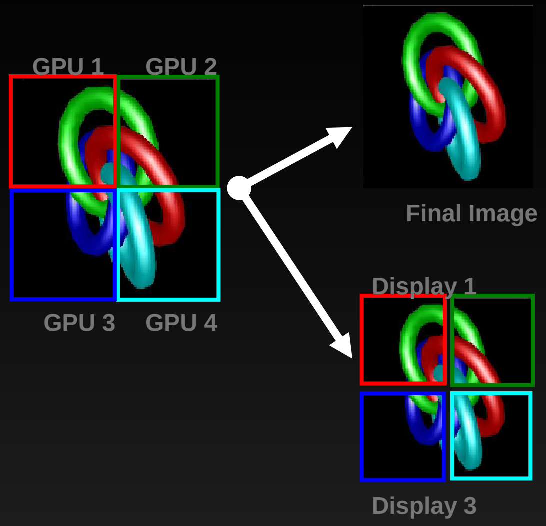
```
BOOL wglEnumGpusDevicesNV(HGPUNV hGPU, UINT iDeviceIndex,
                          PGPU_DEVICE lpGpuDevice);
```

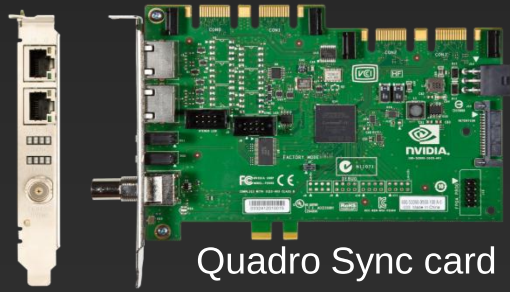
- Pinning OpenGL context to a specific GPU

```
For #GPUs enumerated {
        GpuMask[0]=hGPU[0];
        GpuMask[1]=NULL;
        //Get affinity DC based on GPU
        HDC affinityDC = wglCreateAffinityDCNV(GpuMask);
        setPixelFormat(affinityDC);
        HGLRC affinityGLRC = wglCreateContext(affinityDC);
}
```

Scaling - Onscreen Display

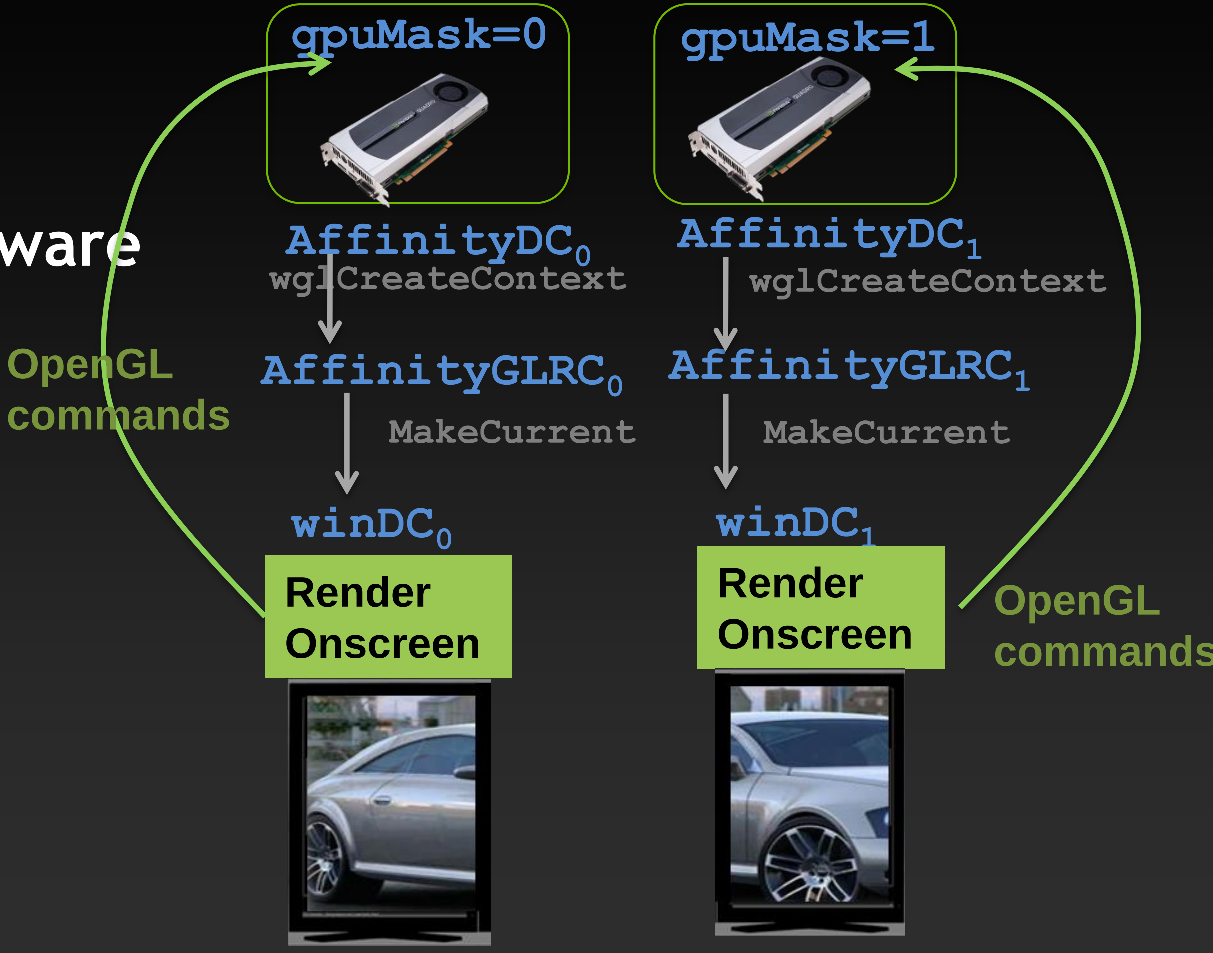
- Sort-First
  - Different GPUs render different portions on the screen
  - Data replicated across all GPUs
- Use cases
  - Fill rate bound apps like raytracing
  - 4K displays, Tiled walls
  - Stereo (needs Quadro Sync)




Quadro Sync card

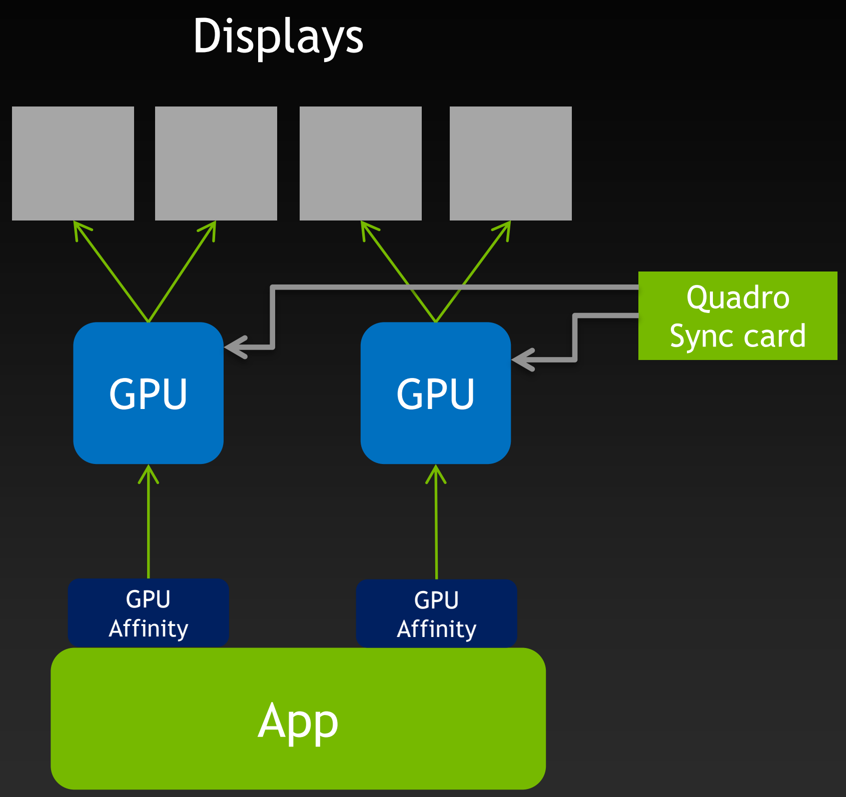
Onscreen Rendering - Overview

- Simple example of sort-first
- No Inter GPU communication
- Thread per GPU to keep hardware queue busy
- Totally programmable
  - Different view frustums
  - View specific optimizations



Adding Frame Synchronization

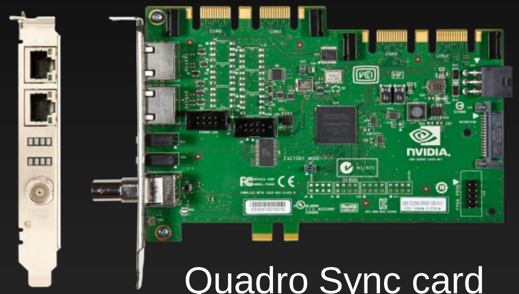
- Needs GSYNC for projection setups to avoid tearing
- *Framelock provides a* common sync signal between graphics cards to insure the vertical sync pulse starts at a common start.
- Between 2 GPUs framelock signal is provided between the CAT5 cable



Onscreen rendering + Framelock

- WGL/GLX extension : NV_Swap_Group syncs buffers between GPUs
  - Swap Groups : windows in a single GPU
  - Swap Barrier : Swap Groups across GPUs
- Init per window DC

```
for (i=0; i< numWindows ; i++) {
        GLuint swapGroup    = 1;
        wglJoinSwapGroupNV(winDC[i], swapGroup)
        wglBindSwapBarrierNV(swapGroup, 1);
}
```



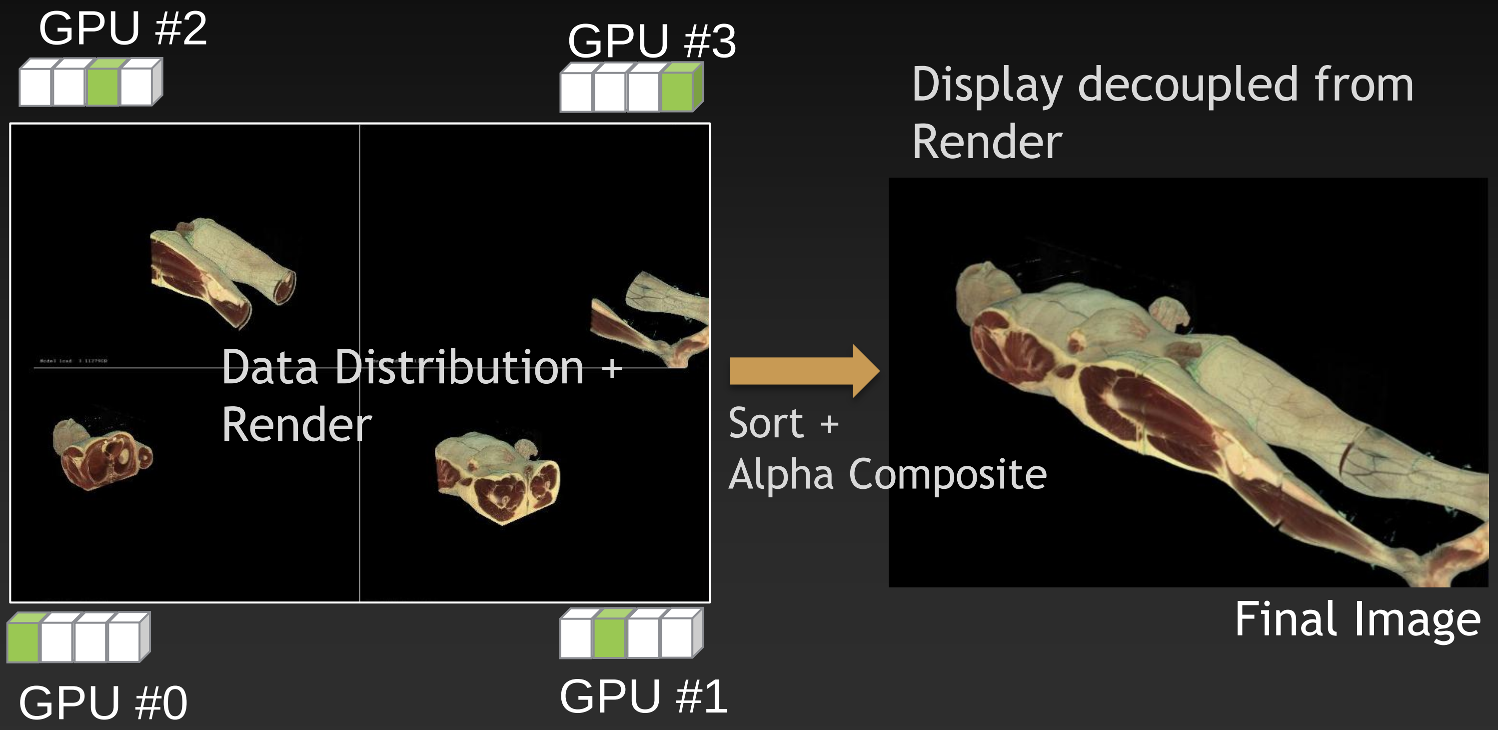
Quadro Sync card

- Display for each window in a separate thread

```
void renderThreadFunc(int idx) {
MakeCurrent(winDC[idx], affinityRC[idx])
//Do Drawing, only on GPU idx
SwapBuffers(winDC[idx]); //SYNC here for buffer swaps
}
```

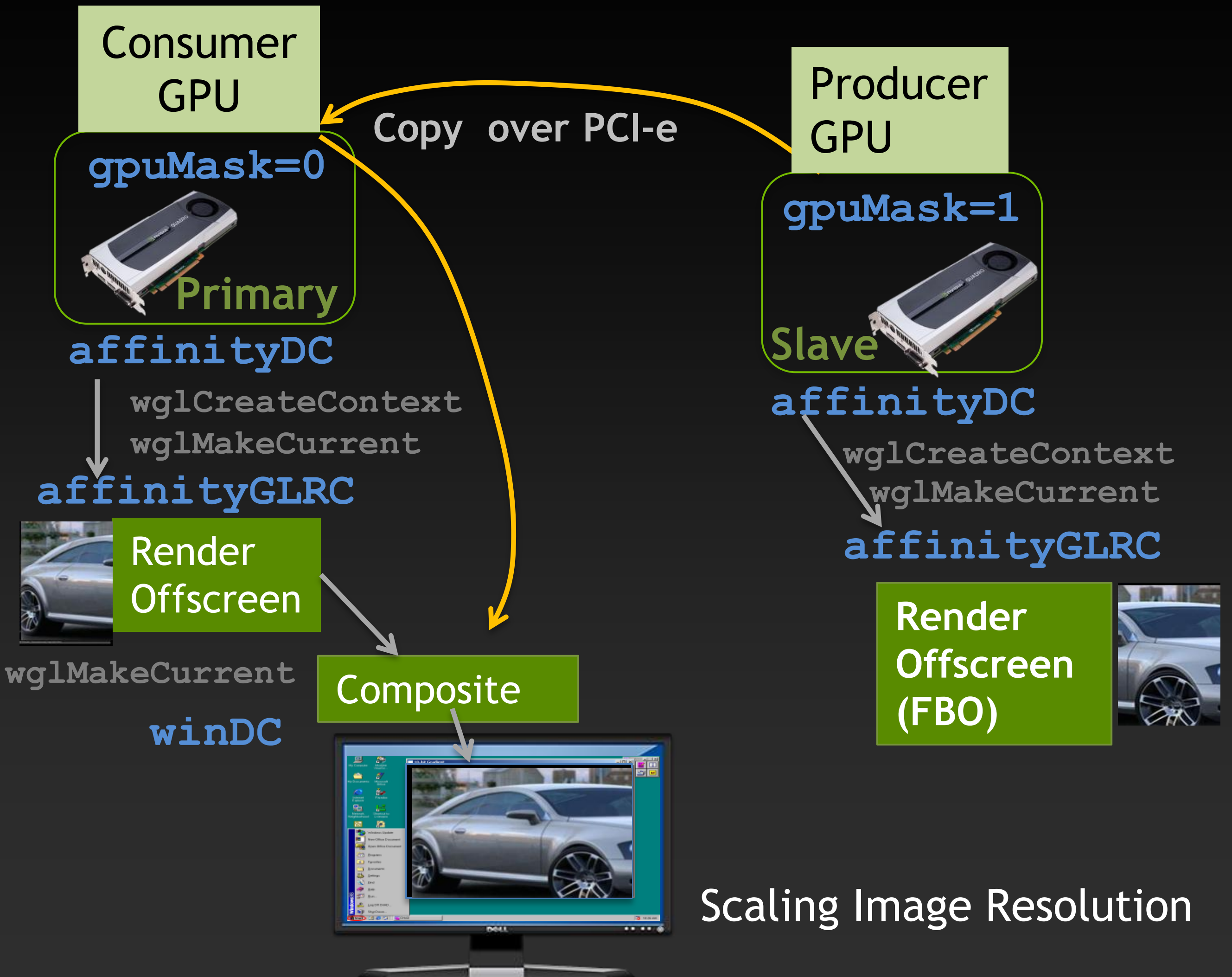
Offscreen Rendering - Scaling Data size

- Scaling data size using Sort-Last approach
  - Eg Visible Human Dataset : 14GB 3D Texture rendered across 4GPUs



Using GPU Affinity

- App manages
  - Distributing render workload
  - implementing various composition methods for final image assembly
- InterGPU communication
- Data, image & task scaling



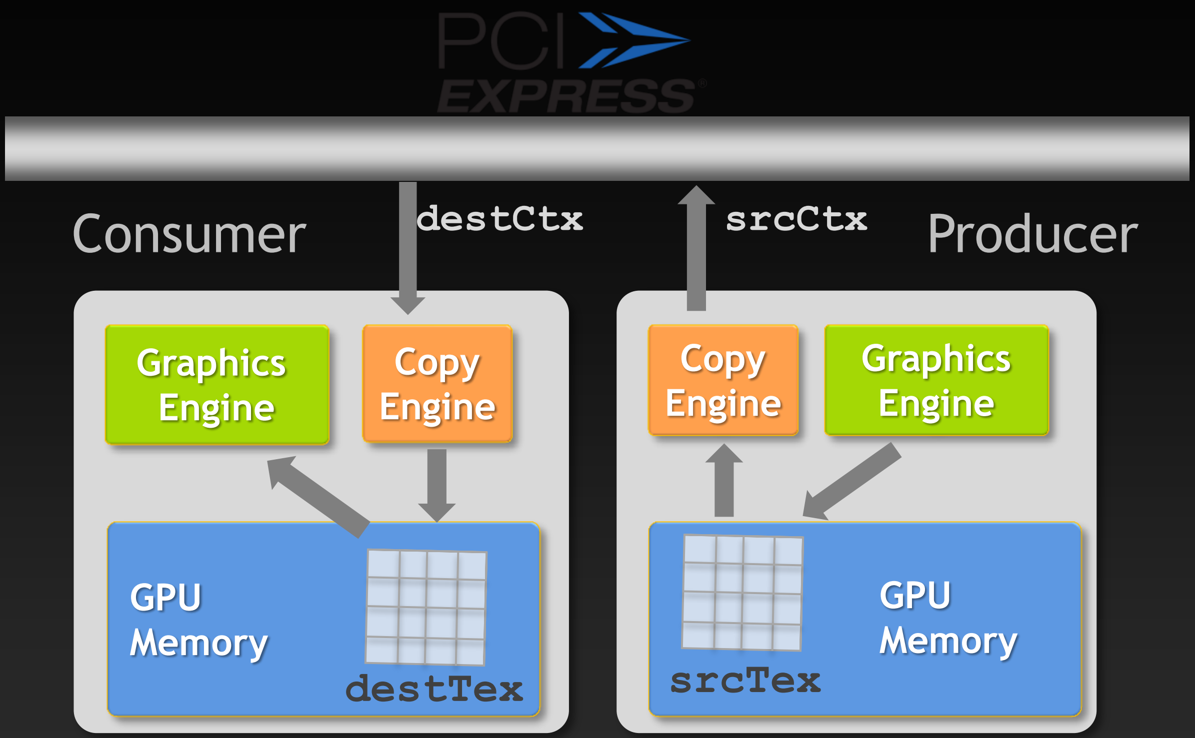
Scaling Image Resolution

Sharing data between GPUs

- For multiple contexts on same GPU
  - `ShareLists & GL_ARB_Create_Context`
- For multiple contexts across multiple GPU
  - Readback (GPU_1-Host) → Copies on host → Upload (Host-GPU_0)
- `NV_copy_image` extension for OGL 3.x
  - Windows - `wglCopyImageSubData`
  - Linux - `glXCopyImageSubDataNV`
  - Avoids extra copies, same pinned host memory is accessed by both GPUs

NV_Copy_Image Extension

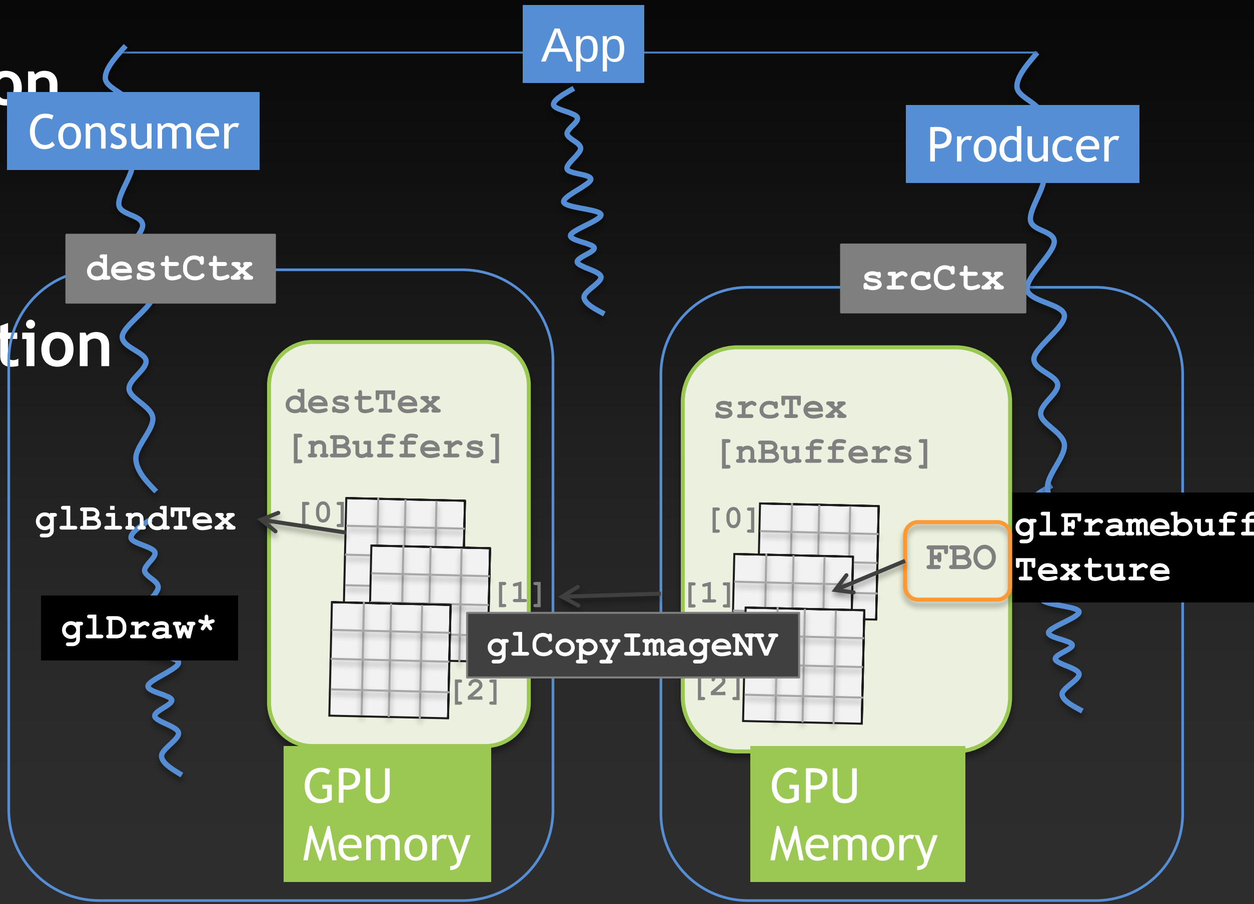
- Transfer in single call
  - No binding of objects
  - No state changes
  - Supports 2D, 3D textures & cube maps
- Async for Fermi & above
  - Requires programming




```
wglCopyImageSubDataNV(srcCtx, srcTex, GL_TEXTURE_2D,0, 0, 0, 0,
        destCtx, destTex, GL_TEXTURE_2D, 0, 0, 0, 0,
        width, height, 1);
```

Producer-Consumer Application Structure

- One thread per GPU to maximize CPU core utilization
- OpenGL commands are asynchronous
- Need GPU level synchronization
  - Use GL_ARB_SYNC
- Can scale to multiple producers/consumers
- Pool of textures to maintain overlap



OpenGL Synchronization

- OpenGL commands are asynchronous
  - When glDrawXXX returns, does not mean command is completed
- Sync object glSync (ARB_SYNC) is used for multi-threaded apps that need sync, Since OpenGL 3.2
  - Eg compositiing texture on gpu1 waits for rendering completion on gpu0
- Fence is inserted in a nonsignaled state but when completed changed to signalled.

```
//Producer Context
glDrawXX
GLSync fence = glFenceSync(..)
```

unsignalled

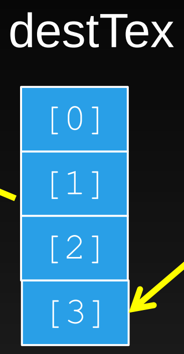
signalled

```
//Consumer Context
glWaitSync(fence)
glBindComposite & draw
cpu work eg memcpy
```

Producer-Consumer Pipeline

Consumer Thread

```
// Wait for signal to start consuming
CPUWait(producedFenceValid);
glWaitSync(producedFence[1]);

// Bind texture object
glBindTexture(destTex[1]);

// Composite as needed

// Signal that consumer has finished
using this texture
consumedFence[1] = glFenceSync(…);
CPUSignal(consumedFenceValid);
```



Producer Thread

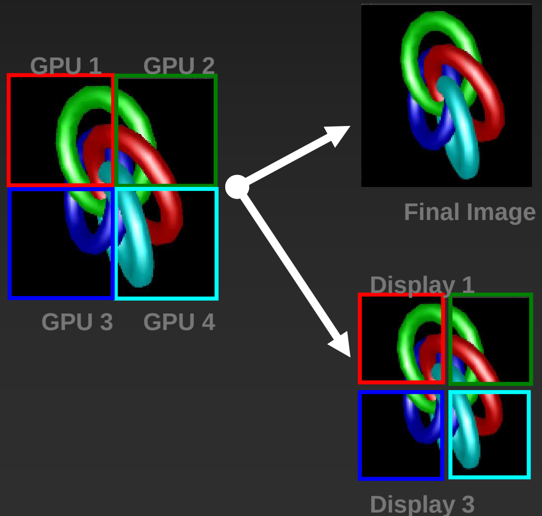
```
// Wait for
CPUWait(consumedFenceValid);
glWaitSync(consumedFence[3]);

// Bind render target
glFramebufferTexture2D(srcTex[3]);

// Draw here…

// Unbind
glFramebufferTexture2D(0);

// Copy over to consumer GPU
wglCopyImageSubDataNV(srcCtx,srcTex[3],
        ..destCtx,destTex[3]);

// Signal that producer has completed
producedFence[3] = glFenceSync(…);
CPUSignal(producedFenceValid);
```

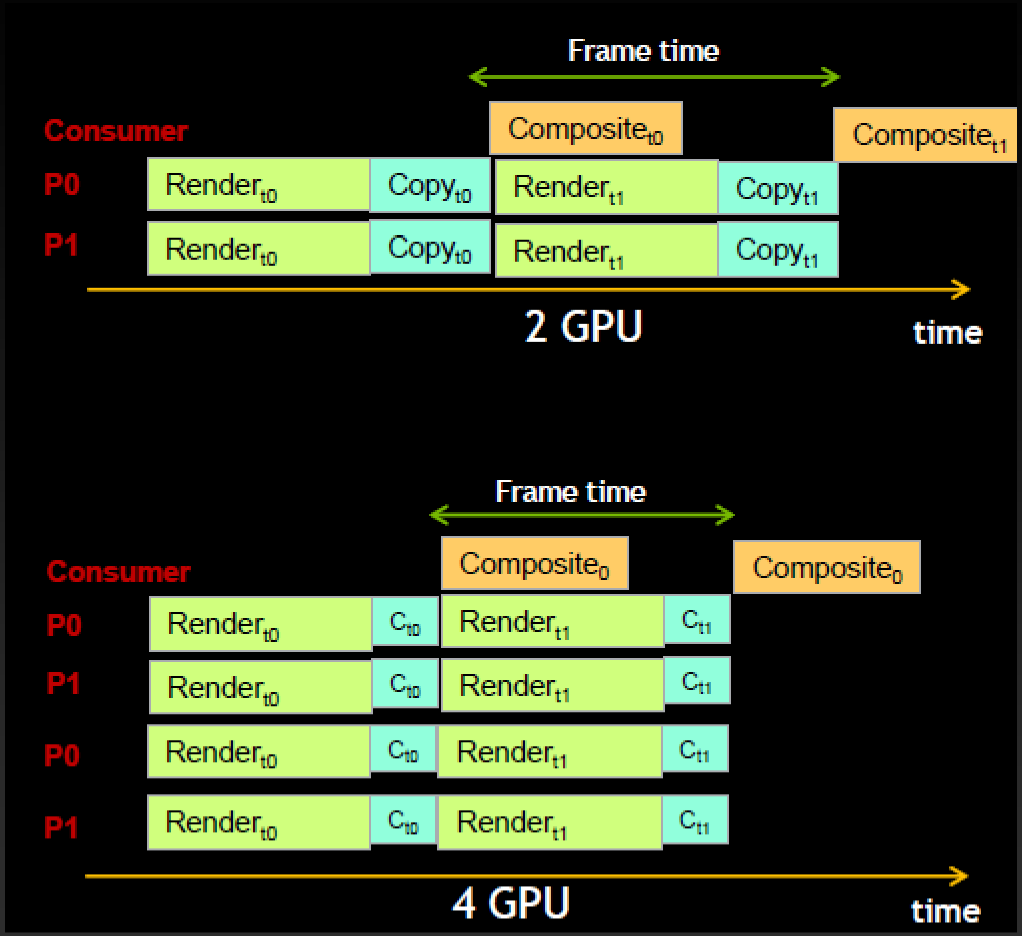
```
GLsync consumedFence[MAX_BUFFERS];
GLsync producedFence[MAX_BUFFERS];
HANDLE consumedFenceValid, producedFenceValid;
```

Multi-level CPU and GPU sync primitives

Applications : Image Scaling

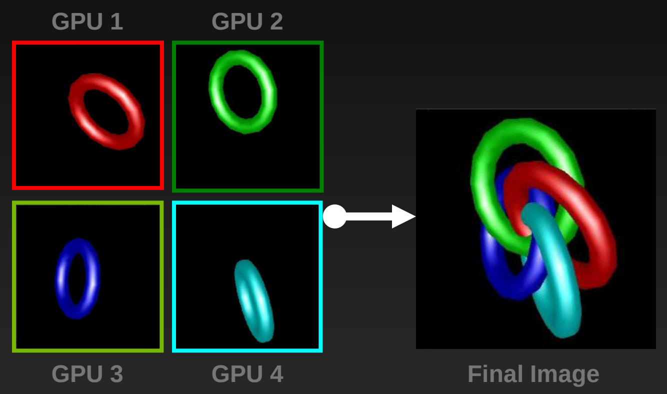
- Sort-first
  - Each GPU works on a smaller subregion of final image
  - Adding more GPUs reduces transfer time per GPU
  - Total data transferred remains constant





Applications : Texture/Geometry Scaling

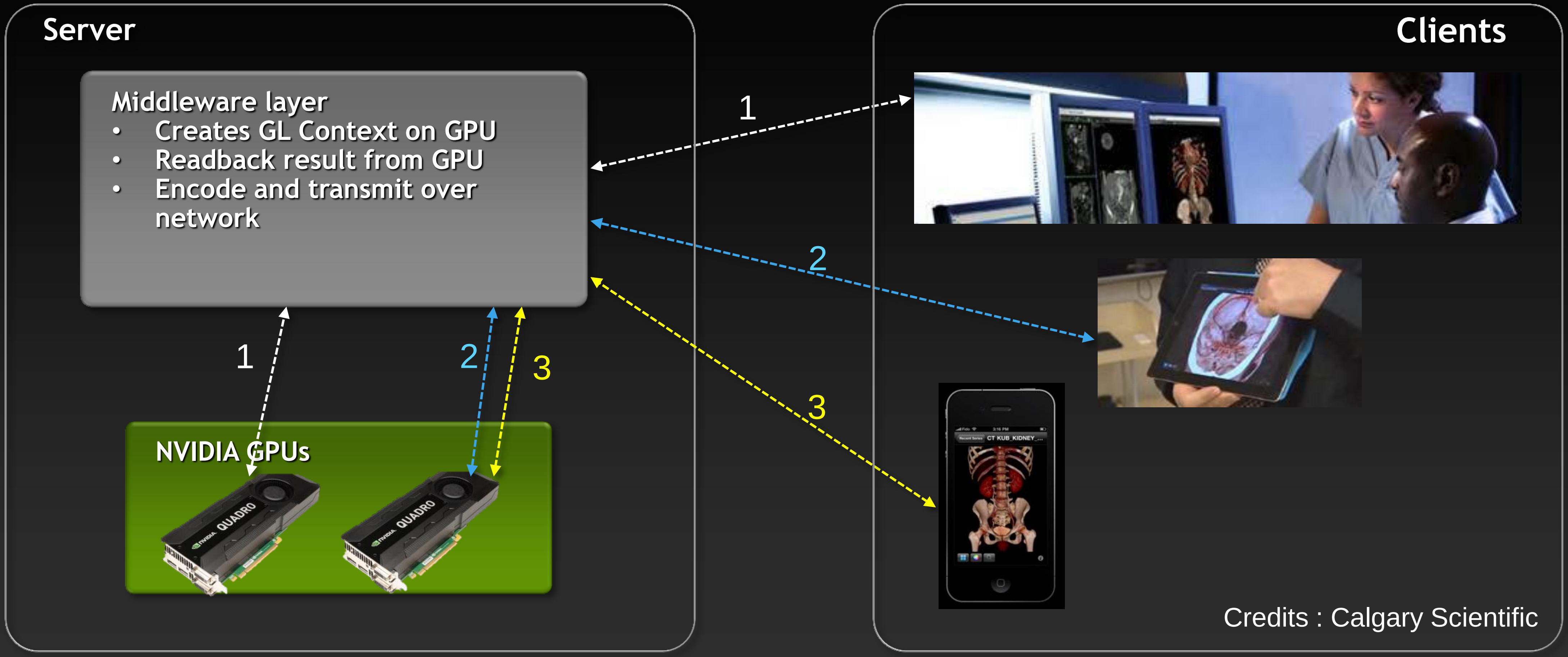
- Adding more GPUs increases transfer time
  - But scales data size
- Full-res images transferred between GPUs
- Volumetric Data
  - Transfer RGBA images
- Polygonal Data (2X transfer overhead)
  - Transfer RGBA and Depth (32bit) images



Applications : Task Scaling

- Render scaling
  - Flight simulation, raytracing
- Server-side rendering
  - Assign GPU for a user depending on heuristics
  - Eg using `GL_NVX_MEMORY_INFO` to assign GPU

Server-side Rendering



Using GL_NVX_gpu_memory_info

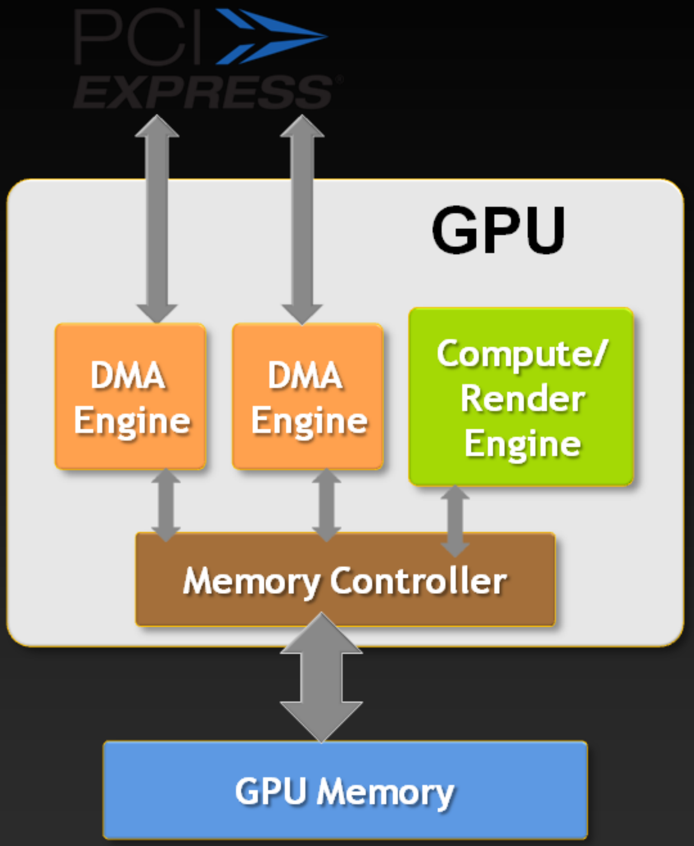
- Extension provides a snapshot view of memory usage
- OS dependent - creation vs first use
- Buffers can migrate between system and video memory depending on usage

```
#define GPU_MEMORY_INFO_DEDICATED_VIDMEM_NVX 0x9047
#define GPU_MEMORY_INFO_TOTAL_AVAILABLE_MEMORY_NVX 0x9048
#define GPU_MEMORY_INFO_CURRENT_AVAILABLE_VIDMEM_NVX 0x9049
glGetIntegerv(GPU_MEMORY_INFO_TOTAL_AVAILABLE_MEMORY_NVX, &total_available_memory);
glGetIntegerv(GPU_MEMORY_INFO_DEDICATED_VIDMEM_NVX, &dedicated_vidmem);
glGetIntegerv(GPU_MEMORY_INFO_CURRENT_AVAILABLE_VIDMEM_NVX,
&current_available_vidmem);
```

http://developer.download.nvidia.com/opengl/specs/GL_NVX_gpu_memory_info.txt

Fast Readbacks with Copy Engines

- Fermi+ have copy engines
  - GeForce, low-end Quadro- 1 CE
  - Quadro 4000+ - 2 CEs
- Allows copy-to-host + compute + copy-to-device to overlap simultaneously
- Graphics/OpenGL
  - Using PBO's in multiple threads
  - Handle synchronization



Multi-threaded Readbacks

Render Thread

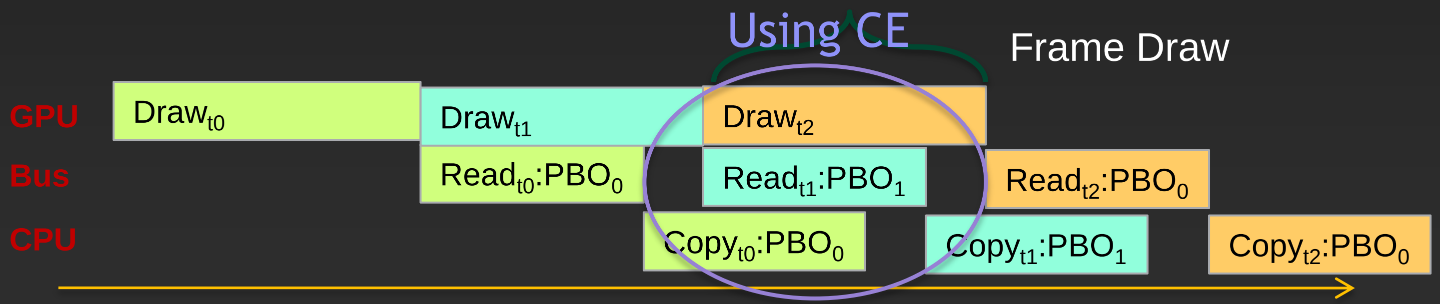
```
// Wait for readback to complete
CPUWait(endReadbackValid);
glWaitSync(endReadback[3]);

// Bind render target
glFramebufferTexture(Tex[3]);

// Draw

// Signal next readback
startReadback[3] = glFenceSync(…);
CPUSignal(startReadbackValid);
```

Tex

[0]
[1]
[2]
[3]

Readback Thread

```
// Readback thread
CPUWait(startReadbackValid);
glWaitSync(startReadback[2]);

// Readback to PBO
glBindBuffer(GL_PIXEL_PACK_BUFFER, pbo)
glBindTexture(Tex[2]);
glGetTexImage(..,0);

// map and memcpy to cpu memory

// Signal download complete
endReadback[2] = glFenceSync(…);
CPUSignal(endReadbackValid);
```



Middleware

- Equalizer
  - Scales from single-node multi-gpu to a multi-node cluster
  - Implements various load-balancing, image reassembly and composition optimization
  - Open Source - www.equalizergraphics.com
- CompleX
  - NVIDIA’s implementation
  - Single system multi-GPU only
  - http://developer.nvidia.com/compleX

References

- **SIGGRAPH ASIA 2012**
  - Mixing Graphics and Compute, Thursday 29 Nov, 16.00-16.45 Room K
  - Current Trends in Advanced GPU Rendering, Friday 30 Nov, 16.00-16.45, Room K
- **OpenGL Insights chapters**
  - Chapter 29 Fermi Asynchronous Texture Transfers
  - Chapter 27 - Multi-GPU Rendering on NVIDIA Quadro
  - Source Code - https://github.com/OpenGLInsights/OpenGLInsightsCode

- **GTC 2012 On-demand talks** http://www.gputechconf.com/gtcnew/on-demand-gtc.php
  - S0353 - Programming Multi-GPUs for Scalable Rendering
  - S0356 - Optimized Texture Transfers