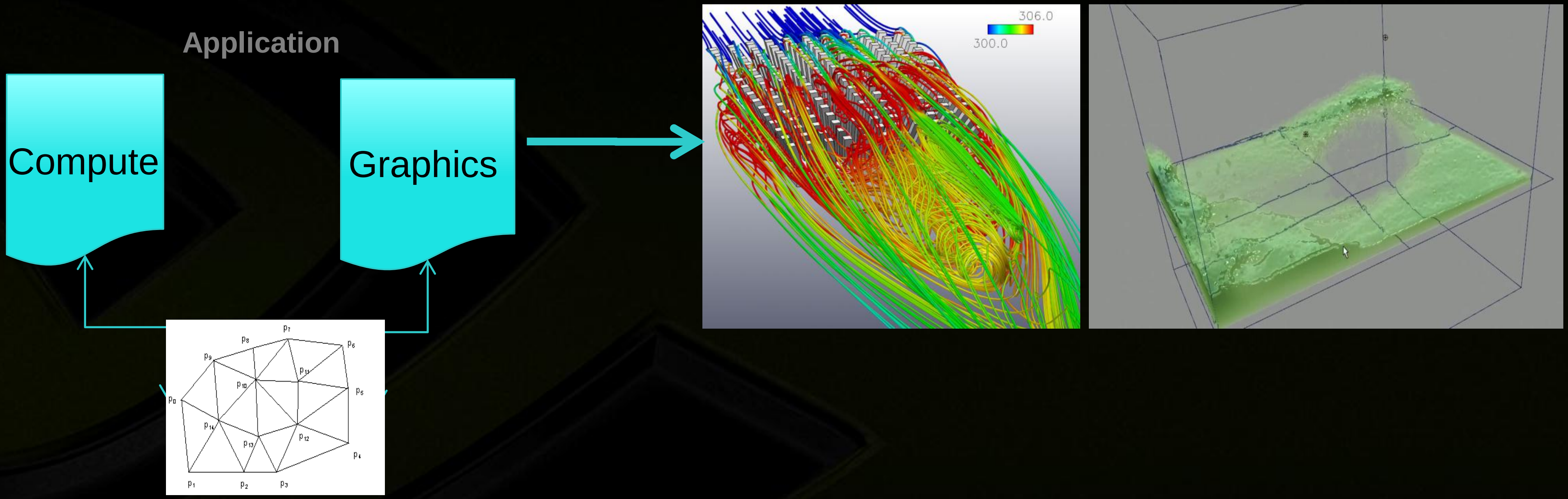
Mixing Graphics and Compute with multiple GPUs, Alina Alt

Agenda

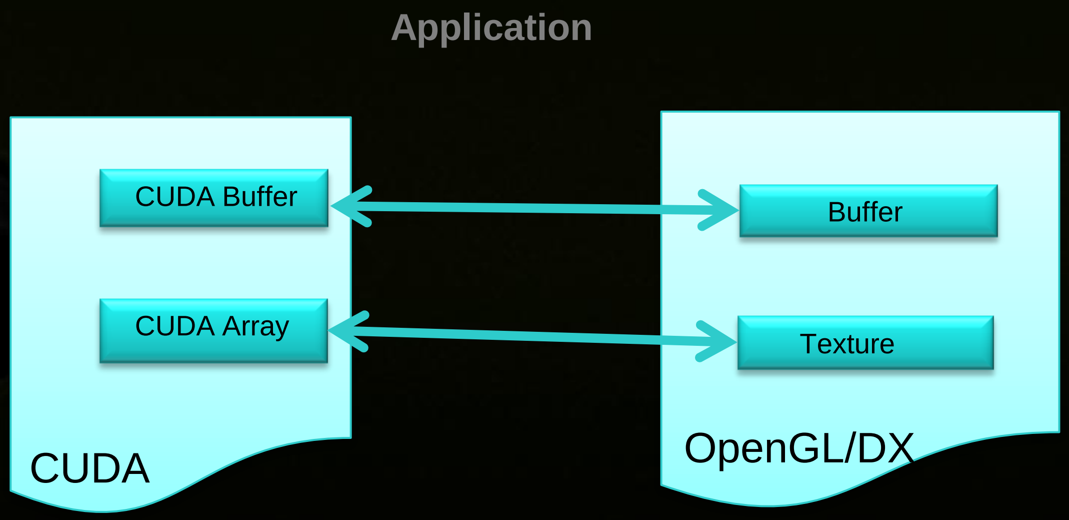
- Compute and Graphics API Interoperability
- Interoperability at a system level
- Application design considerations

Compute and Visualize the same data



Compute/Graphics interoperability

- Set up the objects in the graphics context
- Register objects with the compute context
- Map/Unmap the objects from the compute context



Simple OpenGL-CUDA interop sample: Setup and Register of Buffer Objects

```
GLuint imagePBO;
cudaGraphicsResource_t    cudaResourceBuf;
//OpenGL buffer creation
glGenBuffers(1, &imagePBO);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB, imagePBO);
glBufferData(GL_PIXEL_UNPACK_BUFFER_ARB, size, NULL,
    GL_DYNAMIC_DRAW);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB,0);
//Registration with CUDA
cudaGraphicsGLRegisterBuffer(&cudaResourceBuf, imagePBO,
    cudaGraphicsRegisterFlagsNone);
```

Simple OpenGL-CUDA interop sample: Setup and Register of Texture Objects

```
GLuint imageTex;
cudaGraphicsResource_t    cudaResourceTex;
//OpenGL texture creation
glGenTextures(1, &imageTex);
glBindTexture(GL_TEXTURE_2D, imageTex);
//set texture parameters here
glTexImage2D(GL_TEXTURE_2D,0, GL_RGBA8UI_EXT, width, height, 0,
    GL_RGBA_INTEGER_EXT, GL_UNSIGNED_BYTE, NULL);
glBindTexture(GL_TEXTURE_2D, 0);
//Registration with CUDA
cudaGraphicsGLRegisterImage (&cudaResourceTex, imageTex,
    GL_TEXTURE_2D, cudaGraphicsRegisterFlagsNone);
```

Simple OpenGL-CUDA interop sample

```
unsigned char *memPtr;
cudaArray *arrayPtr;
while (!done) {
    cudaGraphicsMapResources(1, &cudaResourceBuf, cudaStream);
    cudaGraphicsResourceGetMappedPointer((void **)&memPtr, &size,
                                          cudaResourceBuf);
    doWorkInCUDA(cudaArray, memPtr, cudaStream);
    cudaGraphicsUnmapResources(1, & cudaResourceBuf, cudaStream);
    doWorkInGL(imagePBO, imageTex);
}
```

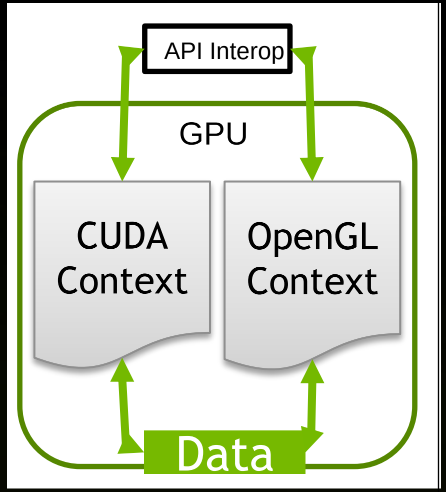
Simple OpenGL-CUDA interop sample

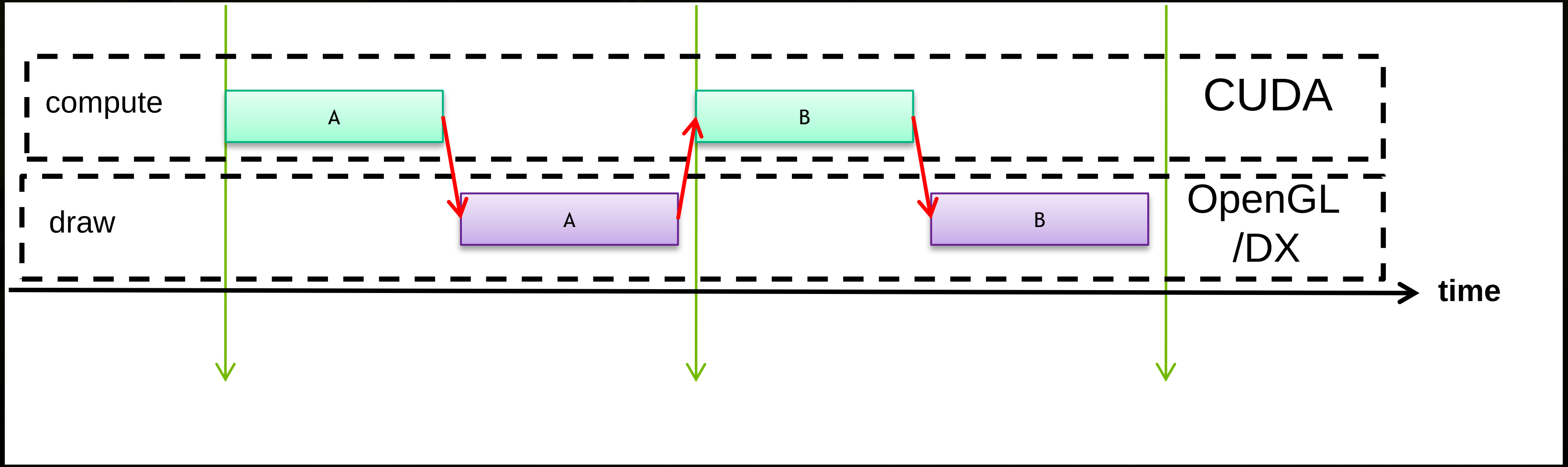
```
unsigned char *memPtr;
cudaArray *arrayPtr;
while (!done) {
    cudaGraphicsMapResources(1, &cudaResourceBuf, cudaStream);
    cudaGraphicsResourceGetMappedPointer((void **)&memPtr, &size,
                                                                  cudaResourceBuf);
    doWorkInCUDA(cudaArray, memPtr, cudaStream);
    cudaGraphicsUnmapResources(1, & cudaResourceBuf, cudaStream);
    doWorkInGL(imagePBO, imageTex);
}
```

Context switching

Interoperability behavior:single GPU

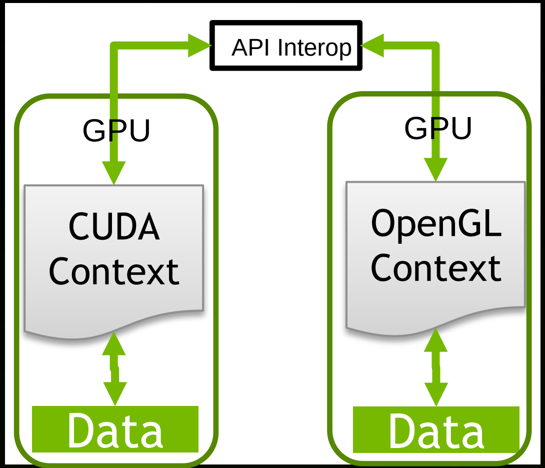
- The resource is shared
- Context switch is fast and independent on data size

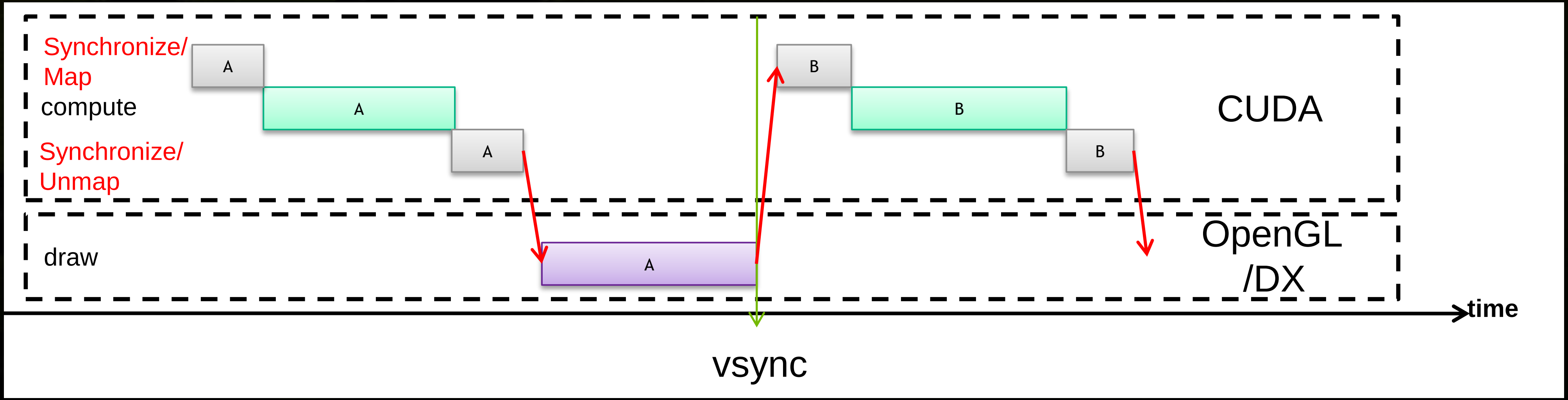




Interoperability behavior: multiple GPUs

- Each context owns a copy of the resource
- Context switch can be slow and is dependent on data size





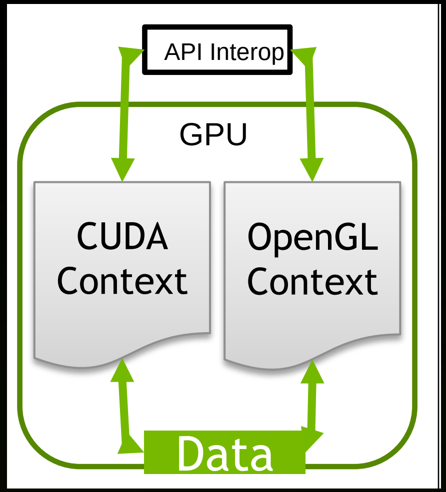
Simple OpenGL-CUDA interop sample

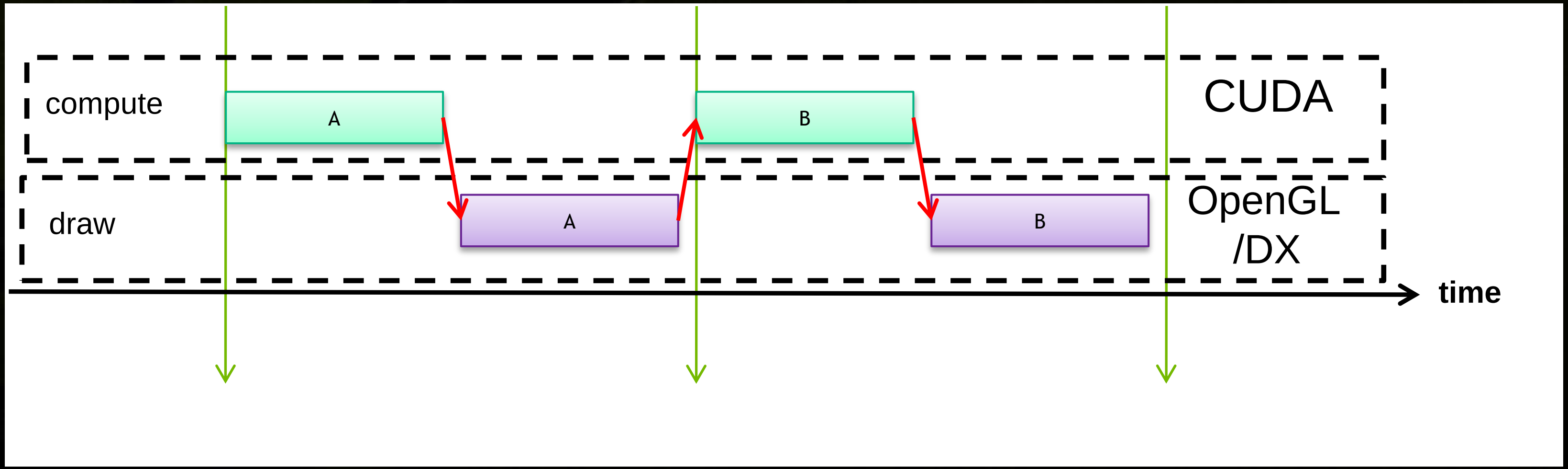
- If CUDA is the producer….
- Use mapping hint cudaGraphicsMapFlagsWriteDiscard with cudaGraphicsResourceSetMapFlags()

```
unsigned char *memPtr;
cudaGraphicsResourceSetMapFlags(cudaResourceBuf,
                                cudaGraphicsMapFlagsWriteDiscard);
while (!done) {
    cudaGraphicsMapResources(1, &cudaResourceBuf, cudaStream);
    cudaGraphicsResourceGetMappedPointer((void **)&memPtr, &size, cudaResourceBuf);
    doWorkInCUDA(memPtr, cudaStream);
    cudaGraphicsUnmapResources(1, &cudaResourceBuf, cudaStream);
    doWorkInGL(imagePBO);
}
```

Interoperability behavior:single GPU

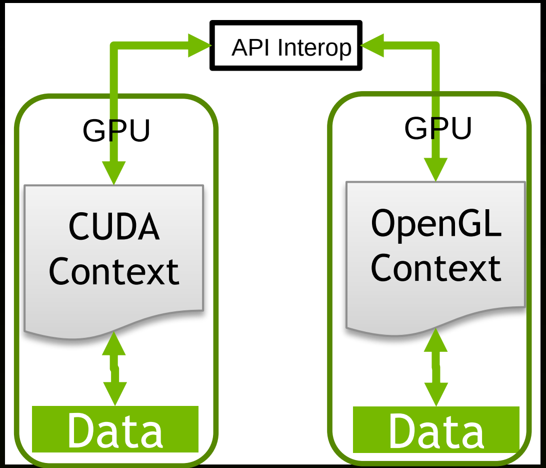
- **Tasks are still serialized….**

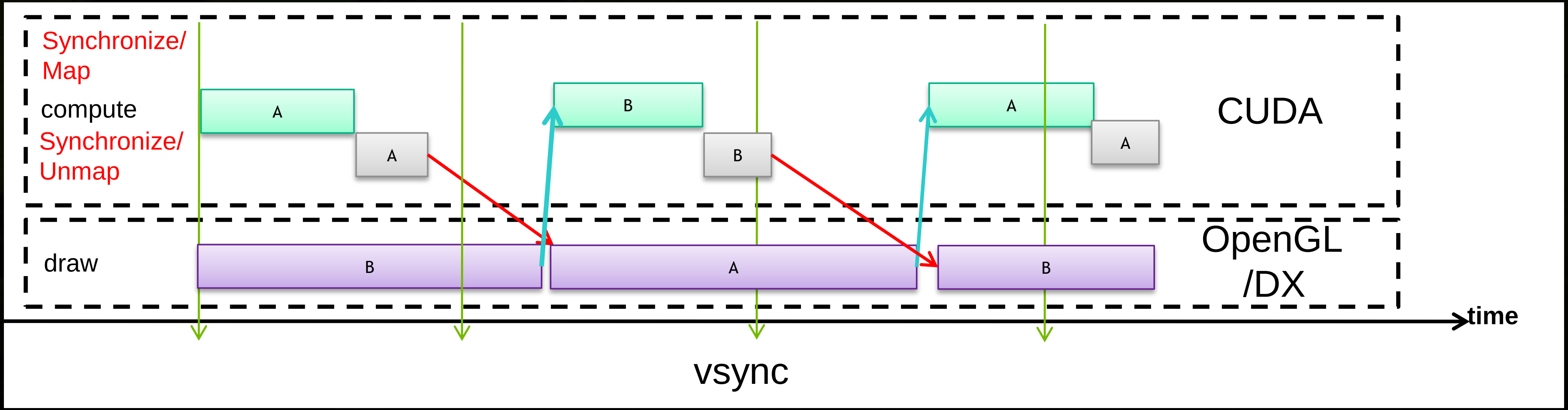




Interoperability behavior: multiple GPUs

- **Tasks are overlapped**





Simple OpenGL-CUDA interop sample

- Similar considerations are applicable when OpenGL is the producer and CUDA is consumer:
  - cudaGraphicsMapFlagsReadOnly

Application Example:Adobe Premiere Pro

- OpenGL plug-in within CUDA application.

Main CUDA thread

```
int count = 1;
while (!done) {
    SignalWait(oglDone[count]);
    doWorkInCUDA(memPtr, NULL);
    EventSignal(cudaDone[count]);
    count  = (count+1)%2;
}
```

OpenGL Worker thread

```
mainCtx = wglCreateContext(hDC);
wglMakeCurrent(hDC, mainCtx);
//Register OpenGL objects with CUDA
int count = 0;
while (!done) {
    SignalWait(cudaDone[count]);
    cudaGraphicsUnmapResources(1, &cudaResourceBuf[count], NULL);
    doWorkInGL(imagePBO[count]);
    cudaGraphicsMapResources(1, &cudaResourceBuf[count], NULL);
    cudaGraphicsResourceGetMappedPointer((void **)&memPtr, &size, cudaResourceBuf[count]);
    EventSignal(oglDone[count]);
    count  = (count+1)%2;
}
```

Application Example:Autodesk Maya

- **CUDA plug-in within an OpenGL application**

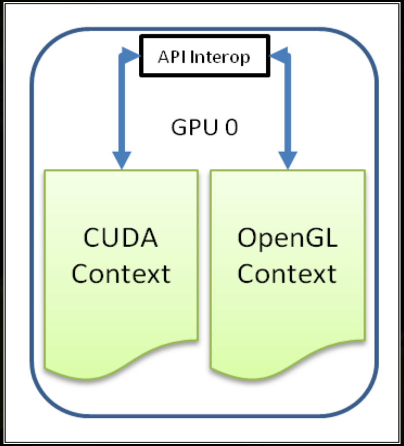
CUDA worker thread N

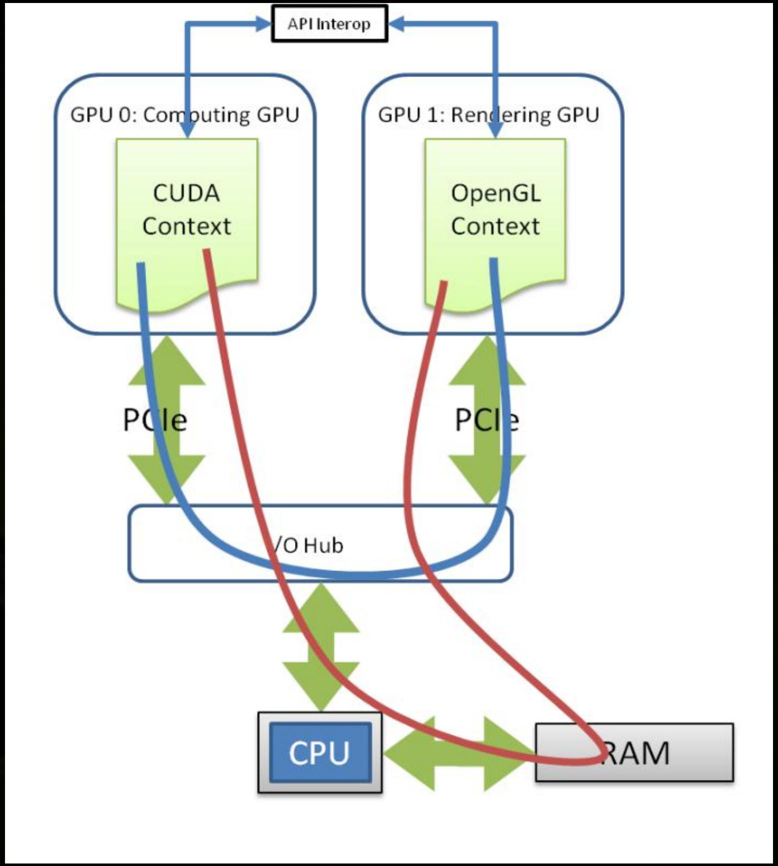
```
SignalWait(setupCompleted);
wglMakeCurrent(hDC,workerCtx);
//Register OpenGL objects with CUDA
int count = 1;
while (!done) {
    SignalWait(oglDone[count]);
    glWaitSync(endGLSync[count]);
    cudaGraphicsMapResources(1, &cudaResourceBuf[count], cudaStream[N]);
    cudaGraphicsResourceGetMappedPointer((void **)&memPtr, &size, cudaResourceBuf[count]);
    doWorkInCUDA(memPtr, cudaStream[N]);
    cudaGraphicsUnmapResources(1, &cudaResourceBuf[count], cudaStream[N]);
    cudaStreamSynchronize(cudaStreamN);
    EventSignal(cudaDone[count]);
    count = (count+1)%2;
}
```

Main OpenGL thread

```
mainCtx = wglCreateContext(hDC);
workerCtx = wglCreateContextAttrib
                    (hDC,m
wglMakeCurrent(hDC, mainCtx);
//Create OpenGL objects
EventSignal(setupCompleted);
int count = 0;
while (!done) {
    SignalWait(cudaDone[count]);
    doWorkInGL(imagePBO[count]);
    endGLSync[count] = glFenceSync(…);
    EventSignal(oglDone[count]);
    count = (count+1)%2;
}
```

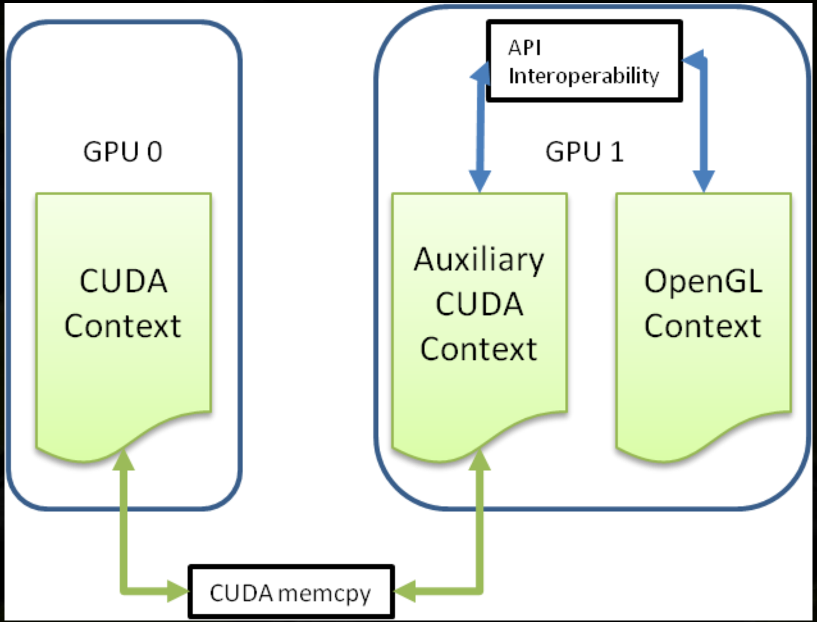
API Interoperability hides all the complexity



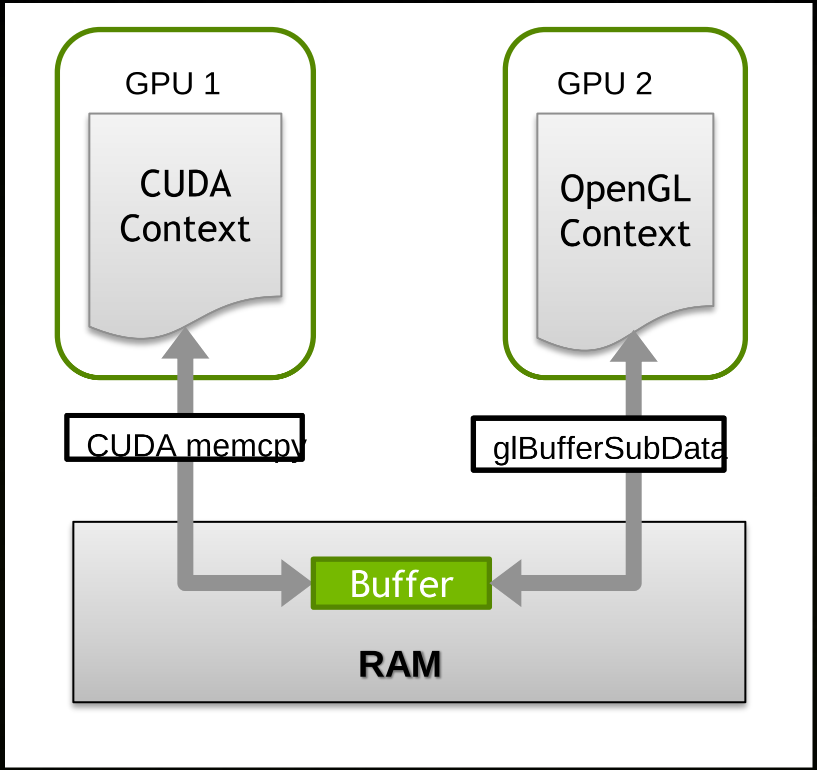


Manual Inetroperability

A)



B)



Application Design Considerations

- Context switching performance varies with system configuration and OS.
- Provision for multi-GPU environments:
  - No heuristics. Let the user chose the GPUs.
  - Use cudaD3D[9|10|11]GetDevices/cudaGLGetDevices to match CUDA and OpenGL device enumerations
- Avoid synchronized GPUs for CUDA
- CUDA-OpenGL interoperability can perform slower if OpenGL context spans multiple GPU
- Consider manual interoperability for fine-grained control

Resources

- CUDA samples/documentation:

http://developer.nvidia.com/cuda-downloads

- OpenGL Insights, Patric Cozzi, Christophe Riccio, 2012. ISBN 1439893764. *www.openglinsights.com*