```
// allocate device memory
int *gpu_A, *gpu_B, *gpu_C;
cudaMalloc((void**) &gpu_A, mem_size);
cudaMalloc((void**) &gpu_B, mem_size);
cudaMalloc((void**) &gpu_C, mem_size);

// copy host memory to device
cudaMemcpy(gpu_A, host_A, mem_size, cudaMemcpyHostToDevice);
cudaMemcpy(gpu_B, host_B, mem_size, cudaMemcpyHostToDevice);

// setup execution parameters
dim3 grid(1, 1, 1);
dim3 block(num_elements, 1, 1);

// execute the kernel
vec_add<<<grid, block>>>(gpu_A, gpu_B, gpu_C, num_elements);
cudaDeviceSynchronize();

// copy results back to host memory
cudaMemcpy(host_C, gpu_C, mem_size, cudaMemcpyDeviceToHost);
```

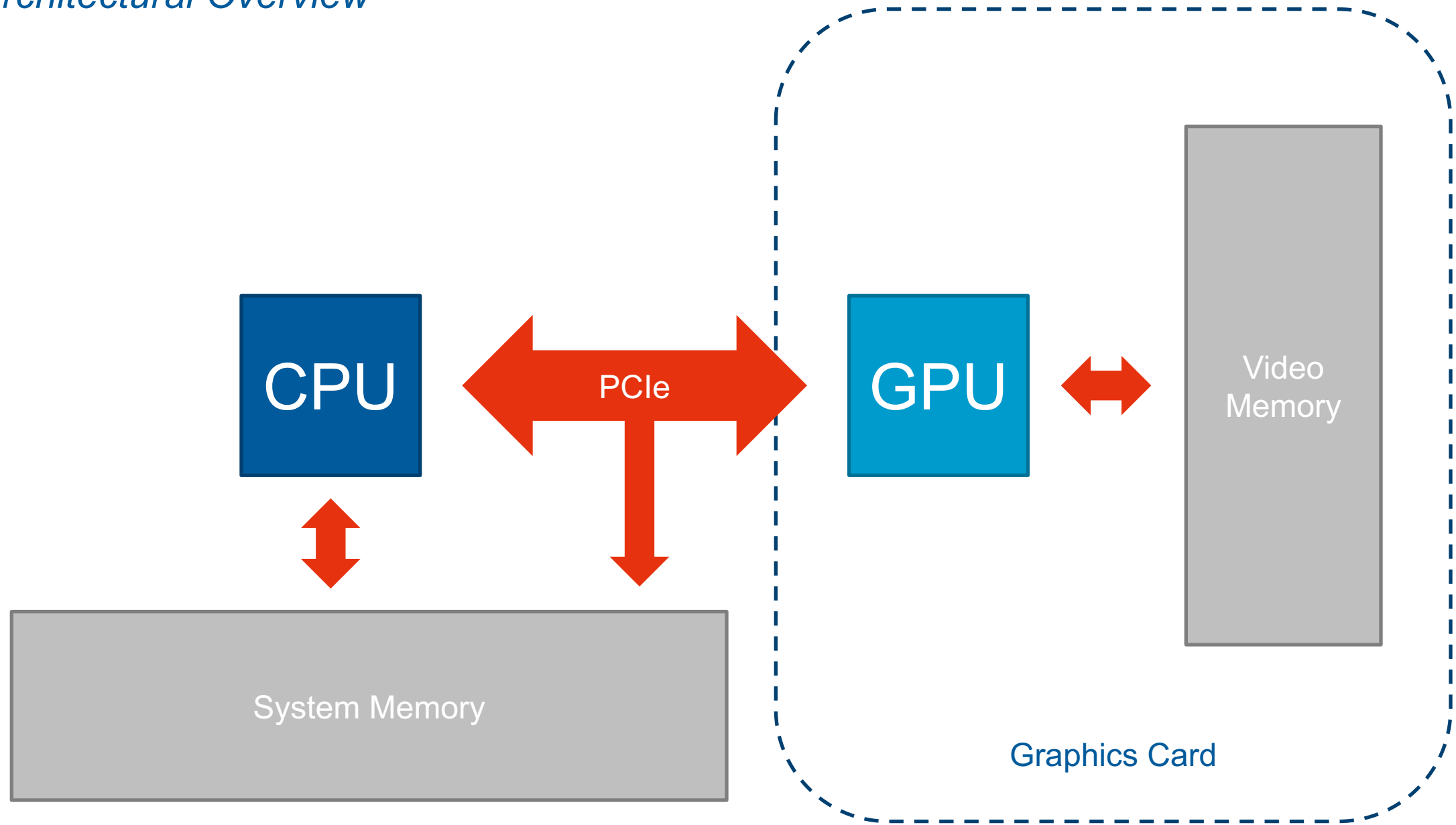
The CUDA API

29.03.23

The CUDA API

- Understand how the GPU fits into the rest of the system
- Overview of the CUDA software stack
- Basic understanding of the CUDA runtime API

Architectural Overview



Architectural Overview

- The GPU is a shared resource
- Used by many applications concurrently
  - Drawing application content
  - Drawing the desktop
  - Running physics simulations
  - Video decoding
  - …

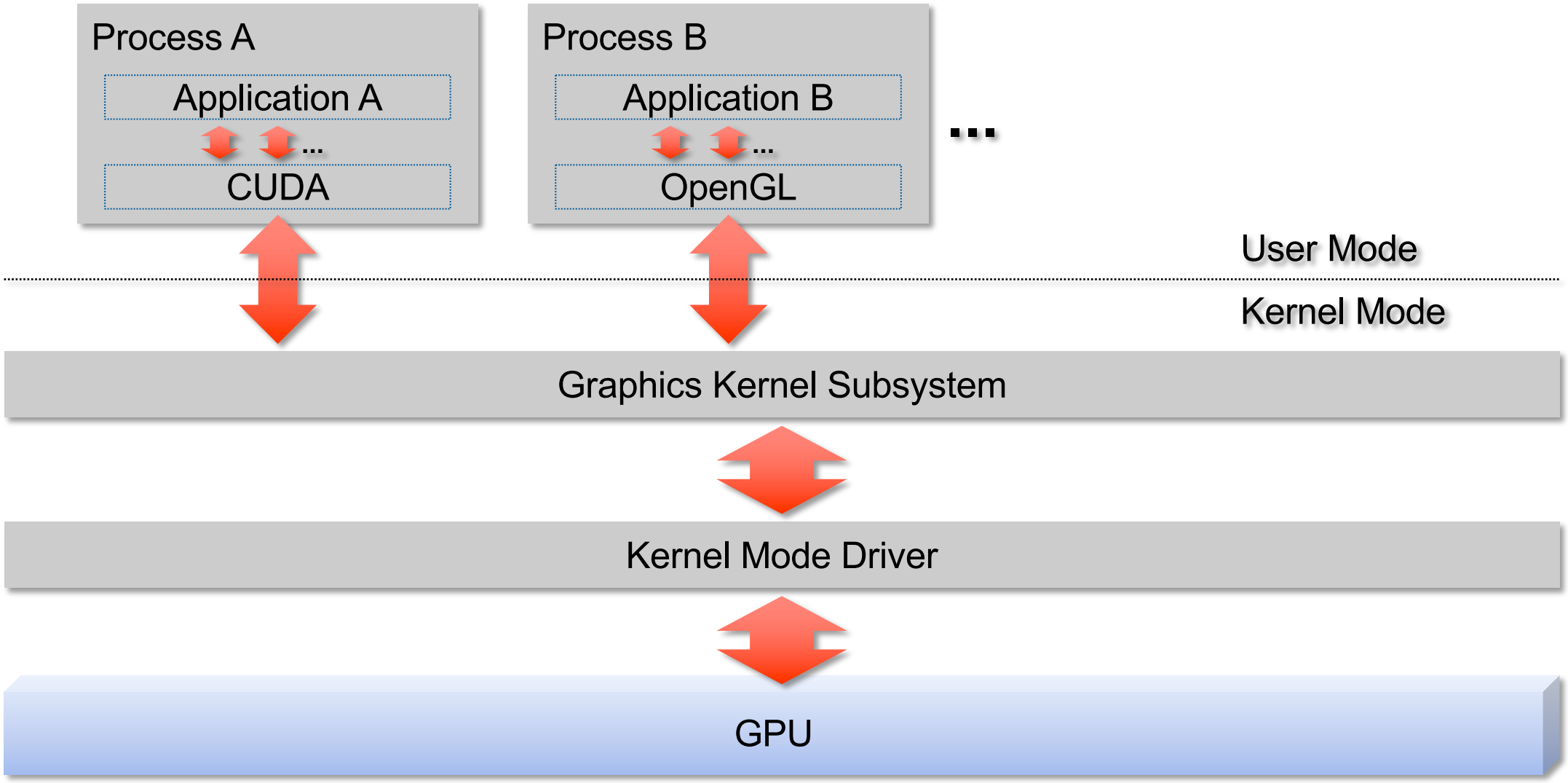
➔ need to schedule access

Architectural Overview

- GPU needs to be told what to do
  - Command buffer: list of operations for the GPU to perform
  - For example: draw triangles, run kernel, copy memory
- User-mode driver
  - Prepares command buffers per application
- Graphics kernel subsystem
  - Schedules access to the GPU
- Kernel-mode driver
  - Submits command buffers to the GPU

Architectural Overview

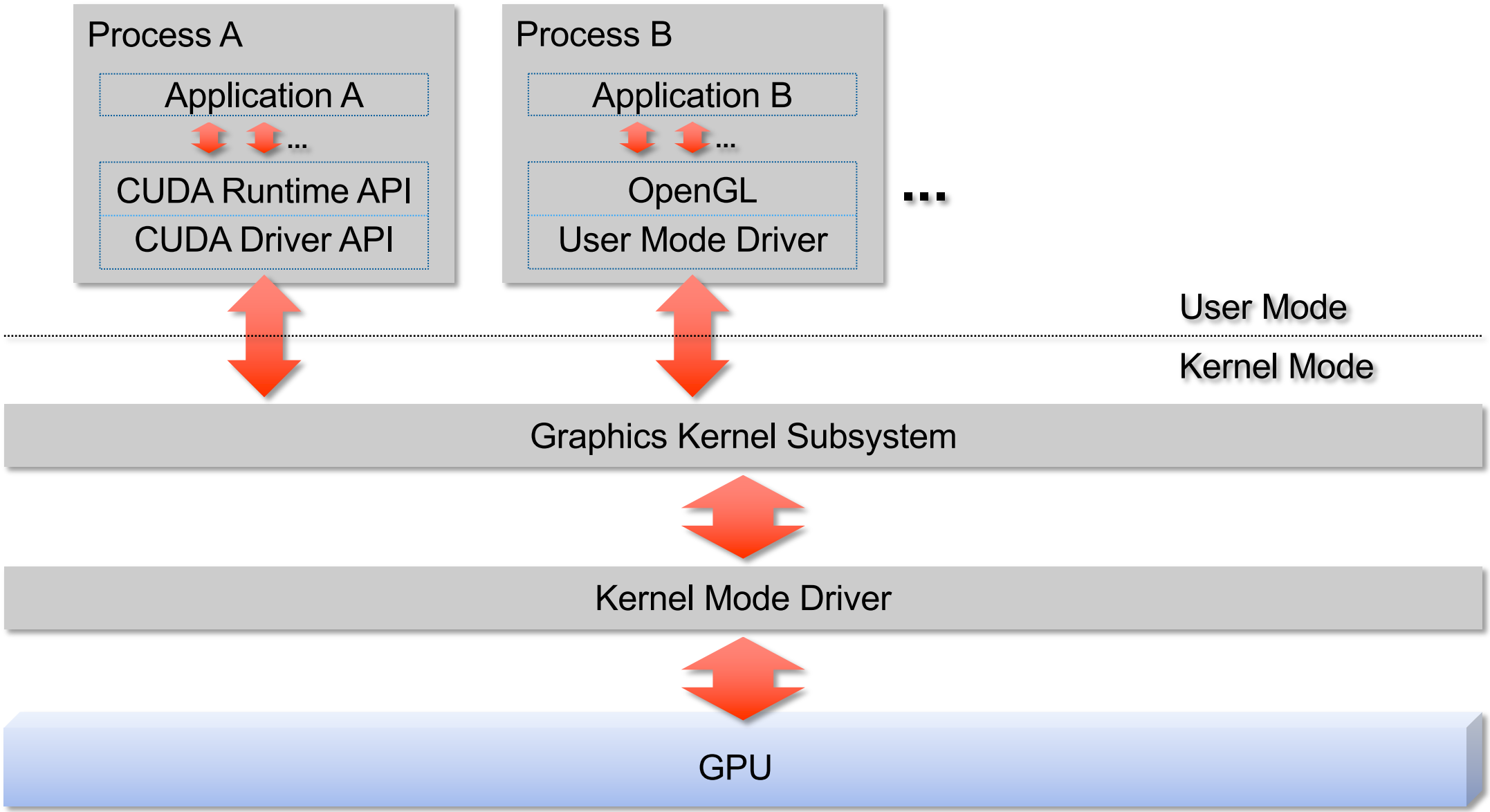
- Application
  - Uses Application Programming Interface (API)
  - To talk to the user-mode driver
  - Which talks to the operating system kernel
  - Which talks to the kernel-mode driver
  - Which talks to the GPU


Process A
Application A
...
CUDA
Process B
Application B
...
OpenGL
...
User Mode
Kernel Mode
Graphics Kernel Subsystem
Kernel Mode Driver
GPU

CUDA APIs

- Driver API
  - Low-level interface to the user-mode driver
- Runtime API
  - Higher-level interface built on top of driver API
  - Relies on compiler magic (`nvcc`)
  - More commonly used
    - Most documentation assumes you're using the runtime API

what we will be using


Process A
Application A
...
CUDA Runtime API
CUDA Driver API
Process B
Application B
...
OpenGL
User Mode Driver
...
User Mode
Kernel Mode
Graphics Kernel Subsystem
Kernel Mode Driver
GPU

CUDA API Concepts: Context

- Represents a session using a GPU
- Context for CUDA API calls to operate in
  - Tracks state associated with current operation

- A process can have multiple contexts
- A context can be *current* on a given thread
  - Context for API calls made by that thread
  - Only one current context per thread
  - Context only current in one thread at the same time

CUDA API Concepts: Streams

- Represents a queue of commands
  - For example, memory copies, kernel launches
  - Executed in order on the GPU but
  - Asynchronously with the CPU
- Can have multiple streams per context
  - No implicit ordering among commands from separate streams
  - Allows for asynchronous operation
    - For example, copy results from previous kernel launch back while next kernel is running
- Default stream (stream 0)
  - By default (no pun intended): all other streams implicitly synchronized with default stream
  - Can be turned into normal stream via compiler flag/API option

Memory Allocation

- GPU has its own memory
- To do any kind of processing: need to
  - Allocate GPU memory
  - Copy data into GPU memory
  - Do the actual processing
  - Copy results back from GPU memory
  - If not needed anymore: free GPU memory

- `cudaMalloc()`/`cudaFree()`
- `cudaMemcpy()`

Error Handling

- CUDA API calls return `cudaError_t`
- Represents the last error that happened
- Attention: This means that API calls may return errors that happened during processing of an operation kicked off by a previous call!

Demo Time

Recap

- Basic API operation
  - `cudaMalloc()`
    - To allocate memory the GPU can operate on
    - Don’t forget to `cudaFree()`
  - `cudaMemcpy()`
    - To copy data into GPU memory and
    - Back to system memory
  - Error handling

Visual Studio Code

Visual Studio Code

- Modern, popular, cross-platform IDE
  - Support for many languages
    - Python
    - Jupyter
    - C/C++
    - …
    - via extensions
  - CUDA support
    - Nsight Visual Studio Code Edition extension
    - C/C++ extension (automatically installed)