CUDA ON WINDOWS

Raphael Boissel, 3/20/2019

ARE YOU IN THE RIGHT ROOM ?

AKA WHAT IS THIS PRESENTATION ABOUT

Step into the details of CUDA on Windows

Explaining the odd behaviors and improving the performance of your application

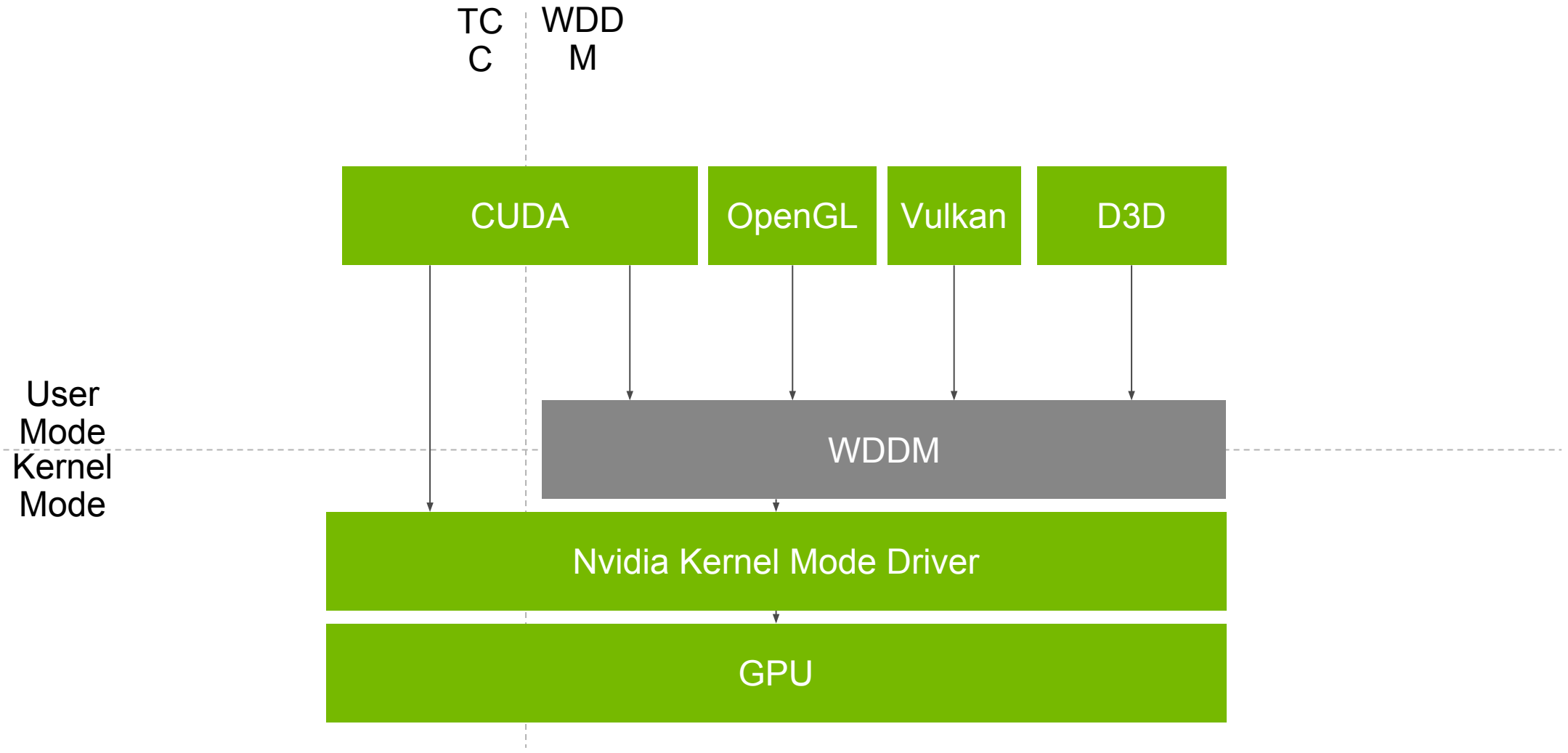
New features for CUDA that are now available on Windows too

From taking advantage of Nvlink on WDDM with P2P supports to Compute preemption taking a closer look at the new feature you can now use in your applications

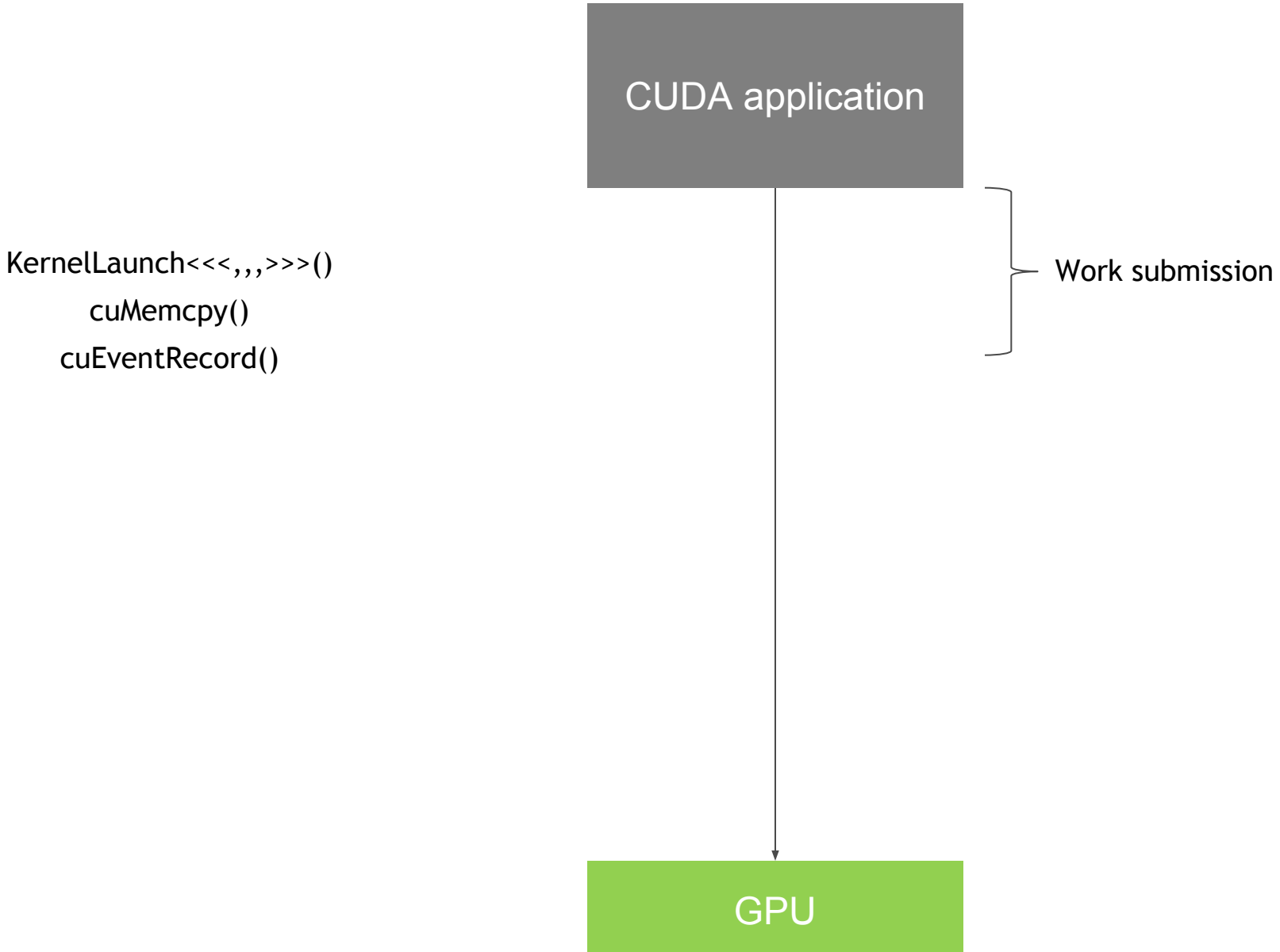

OVERVIEW

OVERVIEW

DRIVER STACK ON WINDOWS



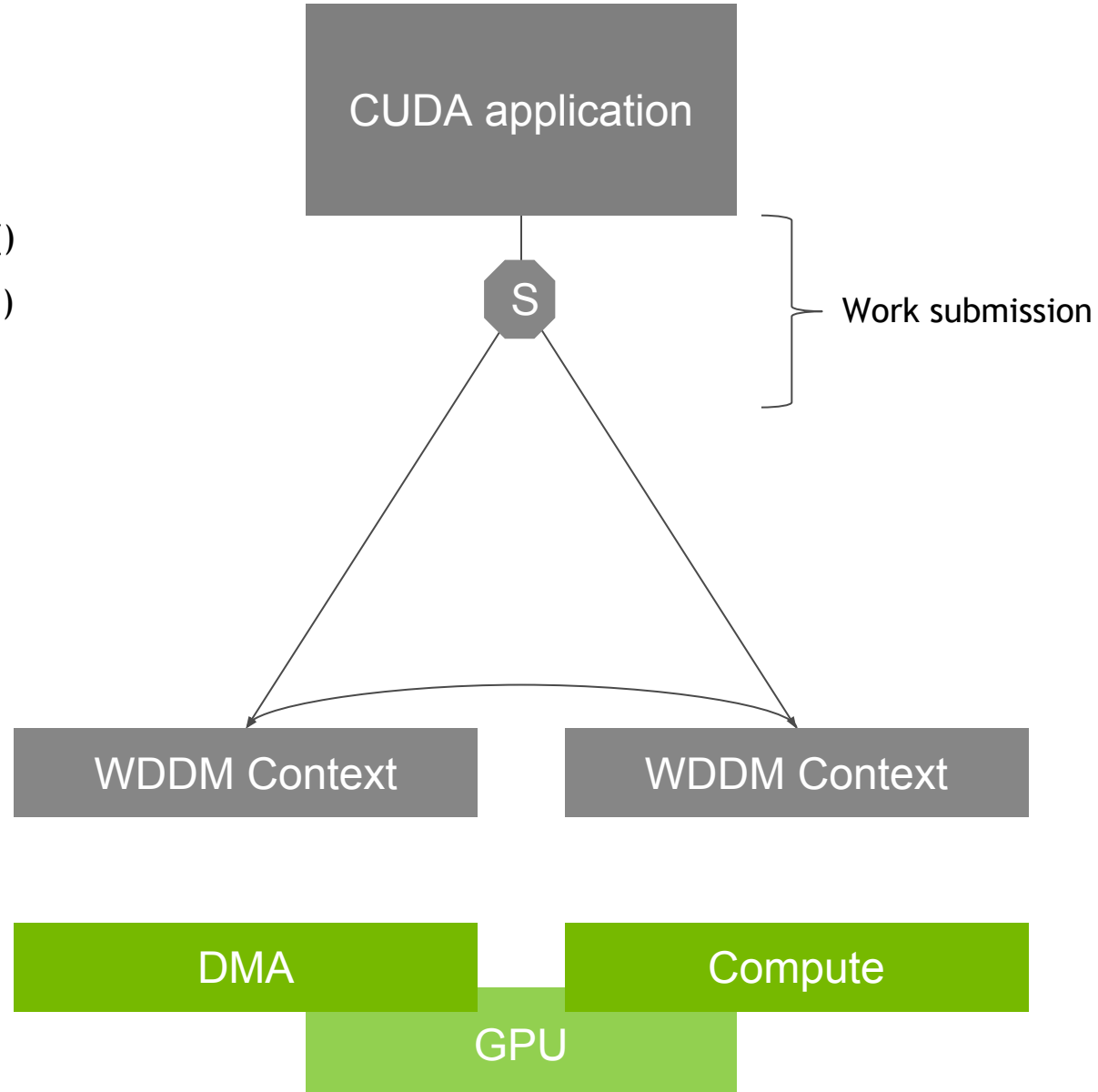

WORKLOAD SUBMISSION


CUDA application
KernelLaunch<<<,,,>>>()
cuMemcpy()
cuEventRecord()
Work submission
GPU

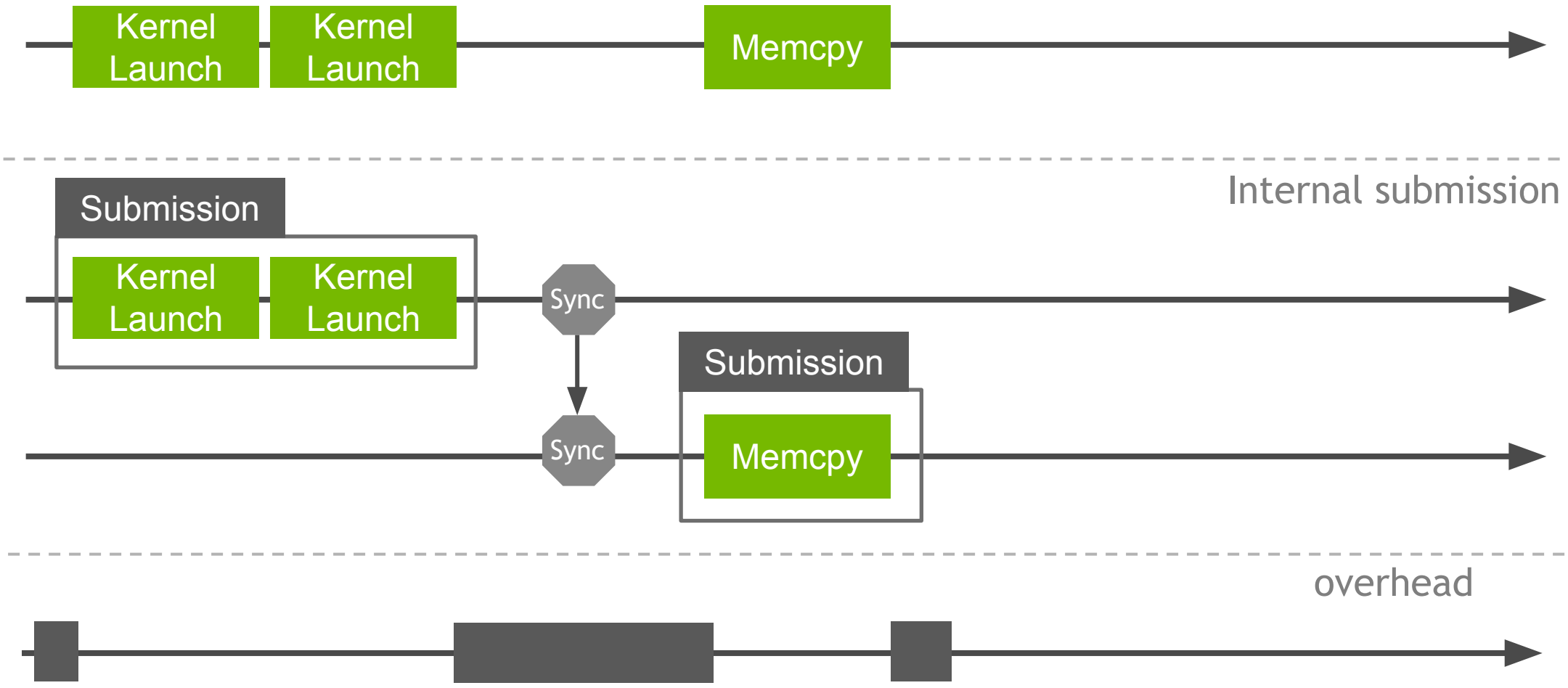
KernelLaunch<<<,,,>>>()

KernelLaunch<<<,,,>>>()

cuMemcpy()



SUBMISSION OVERHEAD

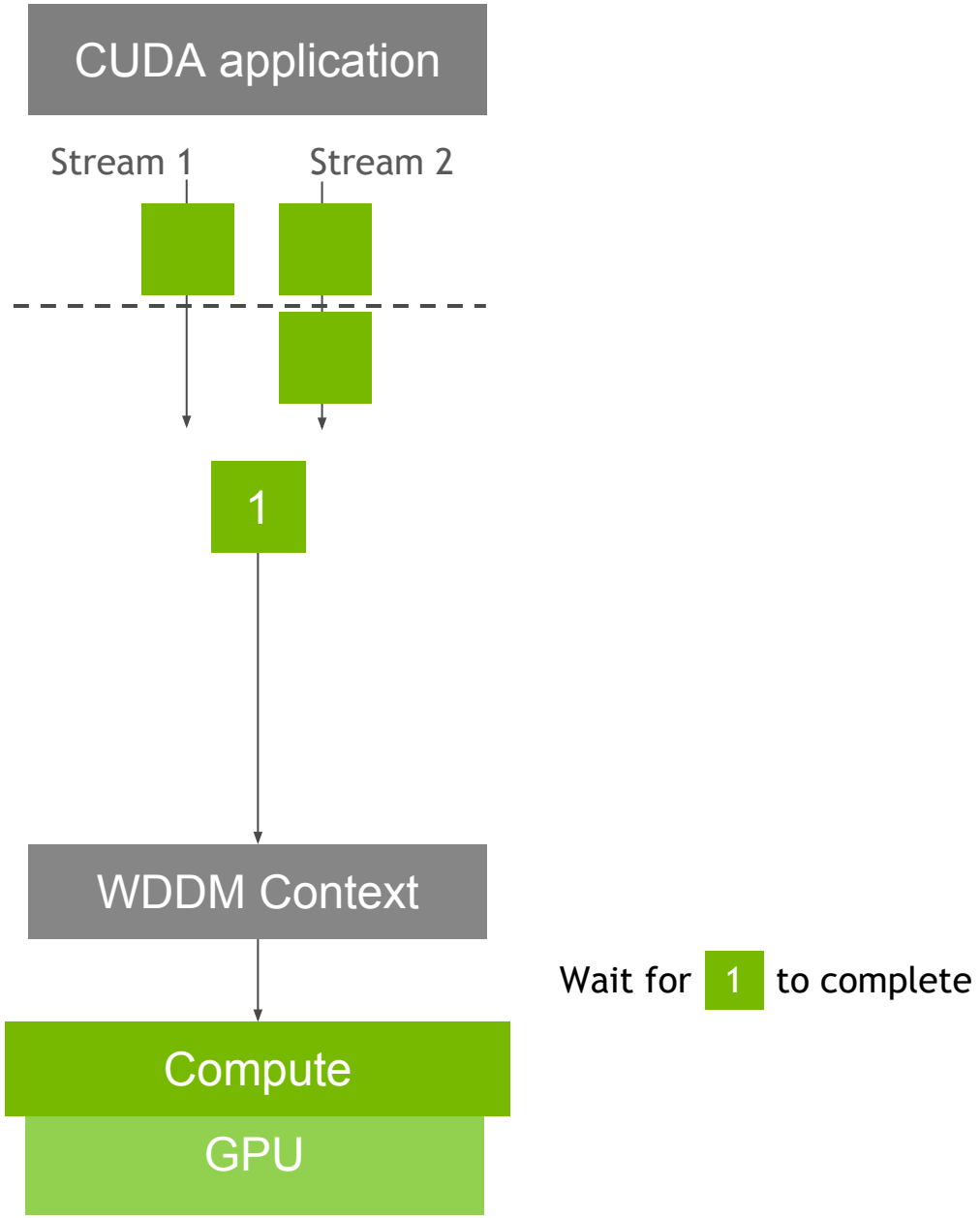


KernelLaunch<<<,,,>>>()

KernelLaunch<<<,,,>>>()

Stream Query

KernelLaunch<<<,,,>>>()

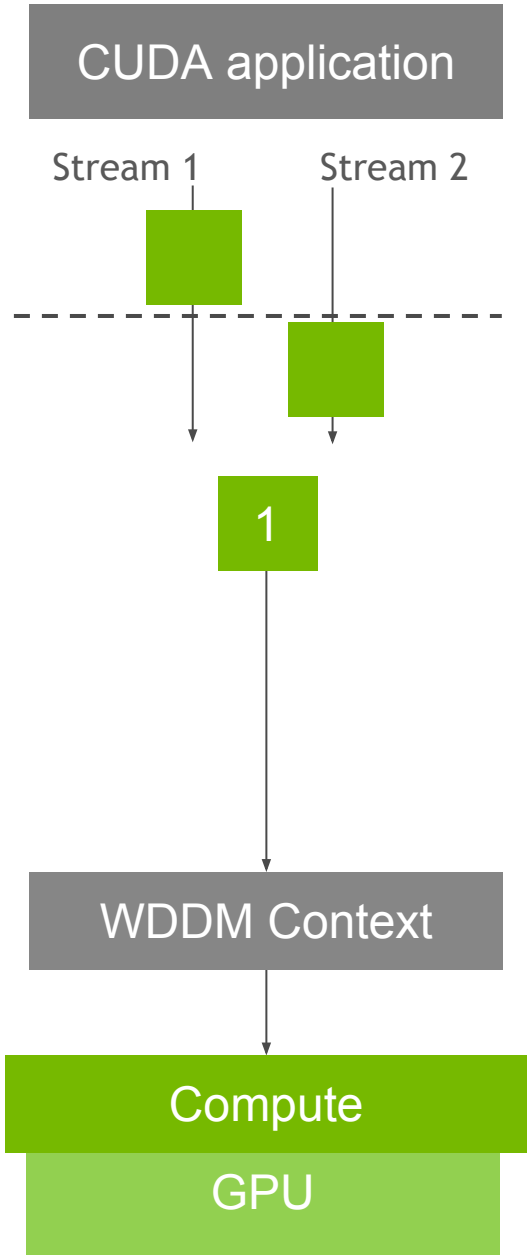


Wait for 1 to complete

KernelLaunch<<<,,,>>>()

Stream Query

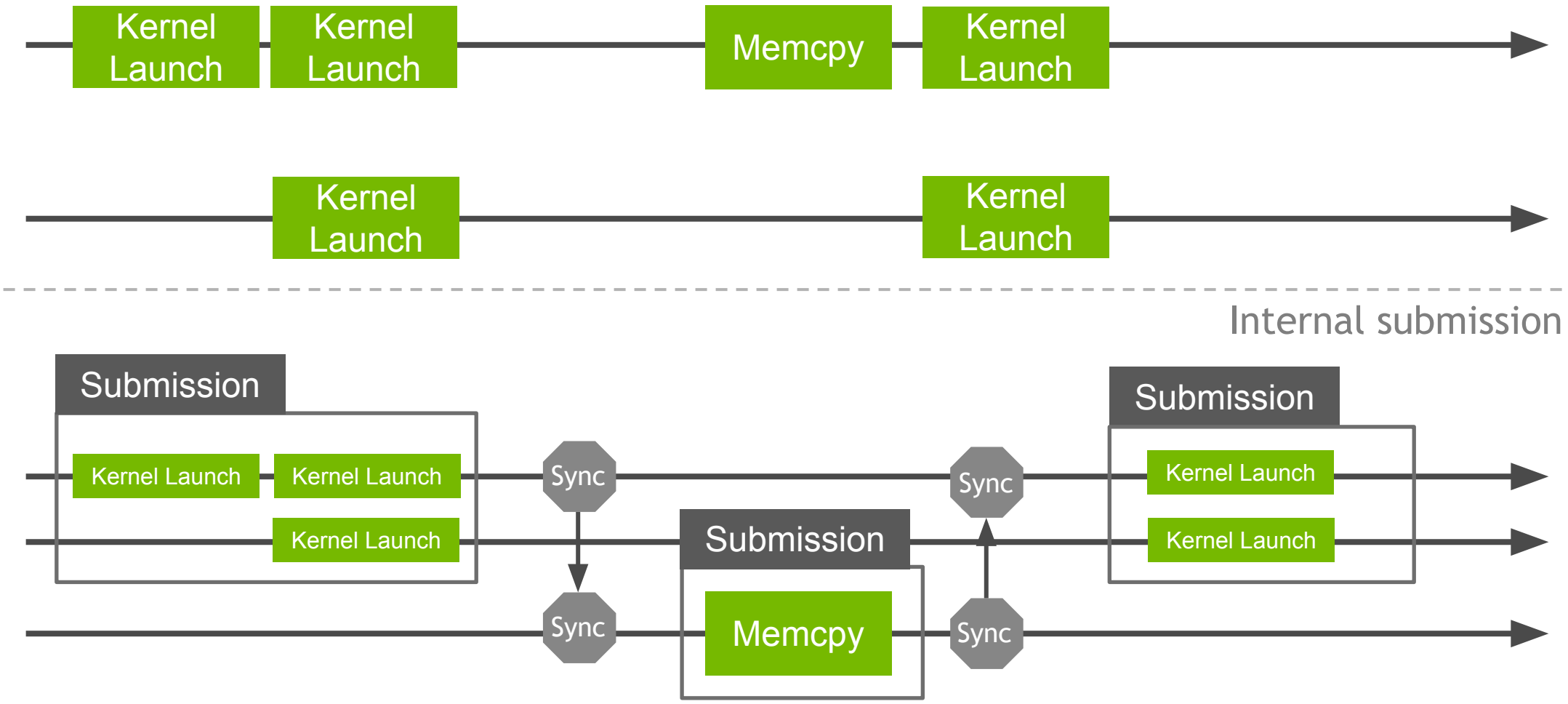
KernelLaunch<<<,,,>>>()



Wait for 1 to complete

SUBMISSION

OVERHEAD

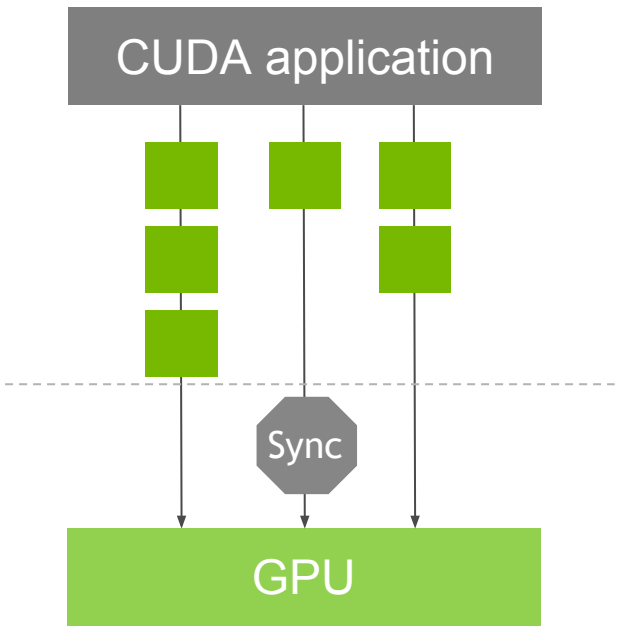


PERFORMANCE ON WDDM

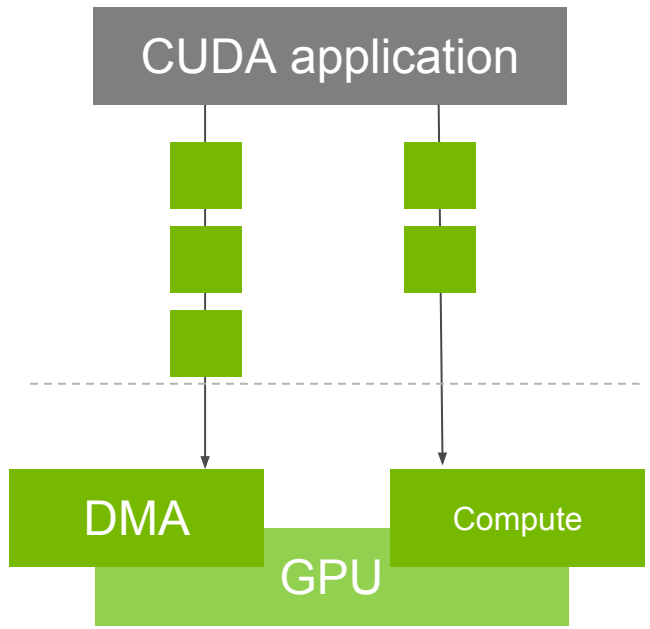
Key points to remember

Batch your submissions

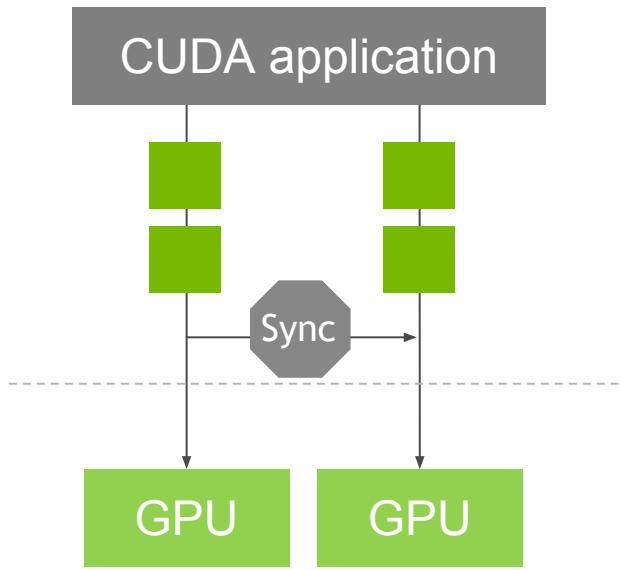
Even between streams



Keep the same type of submission together



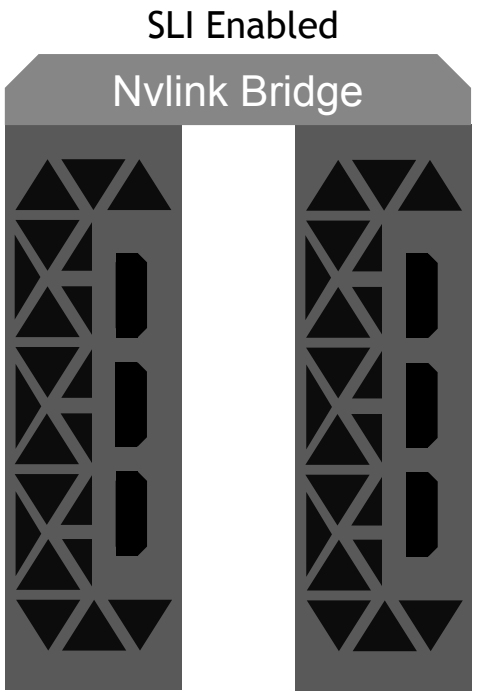
Minimize the use of events between GPUs and Contexts




NEW FEATURES

PEER 2 PEER ON WDDM2

OVERVIEW



Works on windows 10 (WDDM2)

Needs SLI enabled and a system capable of doing P2P

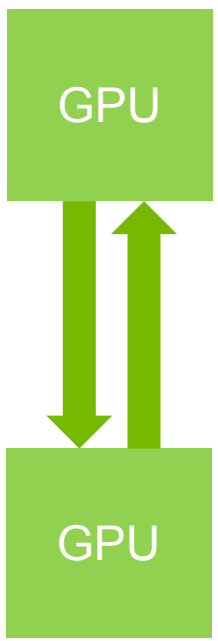
Once the system is setup the P2P APIs will be available *(use the P2P query APIs to check specific capabilities of your system before enabling P2P or using a specific feature)*

PEER 2 PEER ON WDDM2

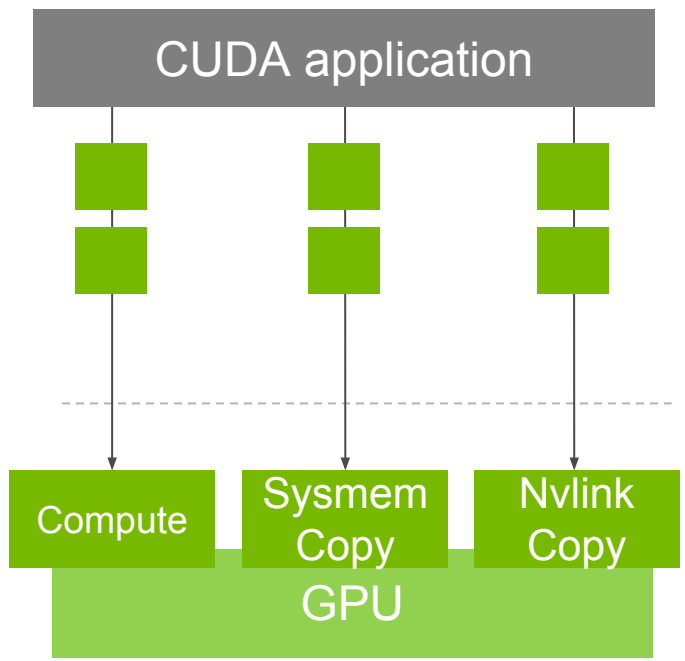
MAXIMIZING BANDWITH

Use both GPUs to do copy

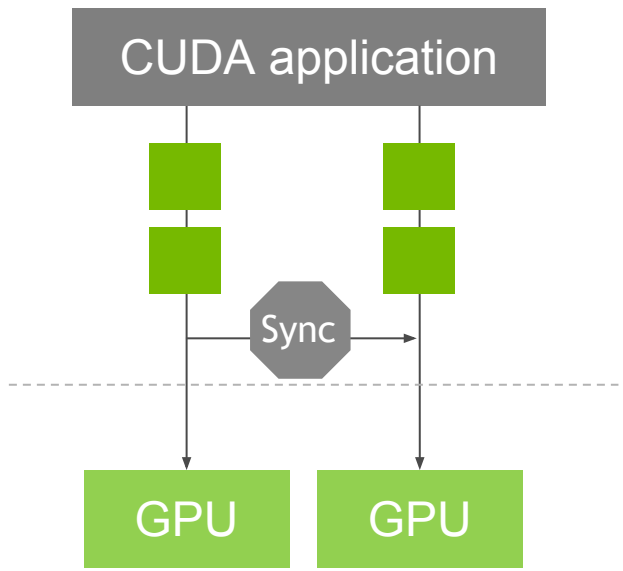
Utilize each GPU engines to saturate the bidirectional bandwidth



Parallelize copy and compute workloads



Minimize the use of events between GPUs and Contexts



PEER 2 PEER ON WDDM2

AVOIDING SUBMISSION LATENCY ISSUES

Group your asynchronous copies to avoid submission overhead, and maximize copy size

> *On very high bandwidth link like nvlink2 the overhead of a submission can quickly become visible. Avoid small independent copies is key to achieving peak bandwidth*

Only use events to synchronize between the two GPUs when necessary

> *Depending on where the event is pushed in the sequence, it might be translated into some primitives that need some extra work on the host. Also minimal taken individually, they can add up quickly if the app is extensively relying on events.*

Be careful when mixing P2P and graphics interop

> *Graphics has its own set of challenges when it comes to SLI, it is easy to see noticeable performance degradation when combining P2P and graphics interop if GPU usage and resource location is not carefully considered*

COMPUTE PREEMPTION

OVERVIEW

A kernel can now run for more than 2s on WDDM2 without hitting a TDR

This is limited to Windows 10 RS4 and above and requires a Pascal Card. Programs should always check if compute preemption is on before trying to use it.

Enabled by default when the configuration supports it

There is no registry key or specific procedure for enablement if the configuration supports it, the feature will be enabled

Works between processes (Graphics / Compute)

Long running compute kernels that will usually prevents graphics rendering to complete degrading user experience are now preemptible so the graphics apps will stay responsive

COMPUTE PREEMPTION

OVERVIEW

Just because you can doesn't mean you should run kernels for an extended period

Preemption on WDDM comes with some internal scheduling policies that makes it hard to purposely take advantage of compute preemption. The easiest way is to simply design your application without worrying about TDR.

Preemption doesn't give extra parallelism between streams within a process

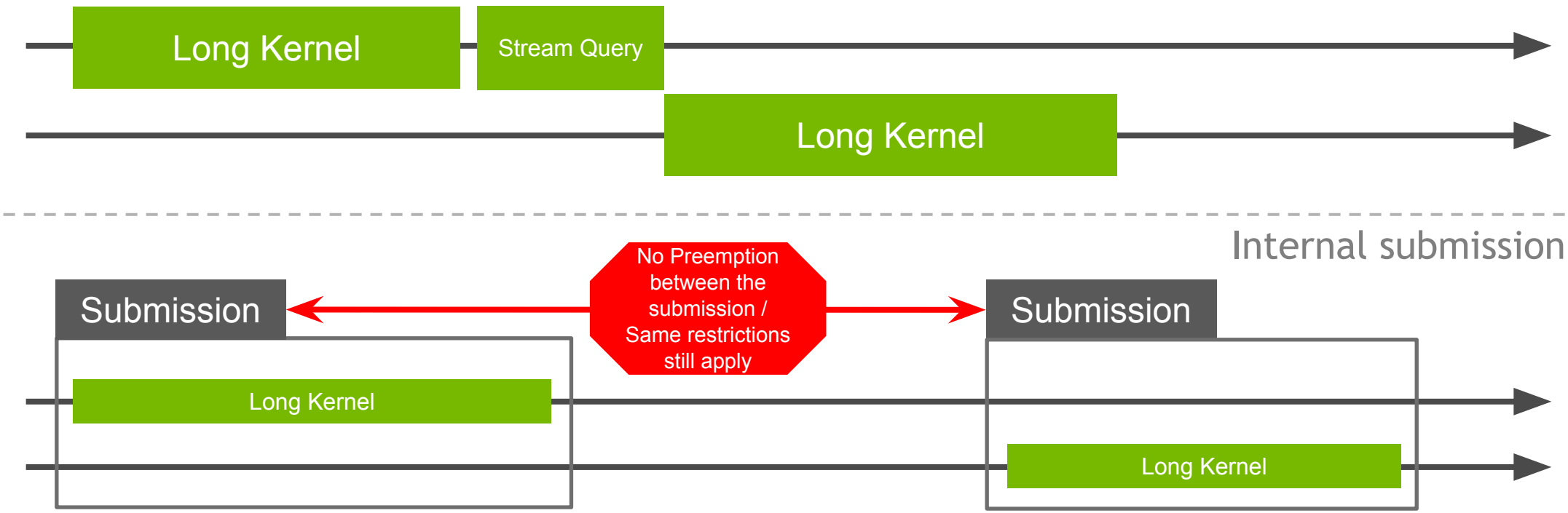
Preemption occurs at internal WDDM submission boundaries. So the previous restrictions on how some kernels might not run concurrently still applies.

Existing programs that were relying on disabling TDR should now work out of the box

This is typically where this features becomes useful: programs that contained kernels running for seconds at a time will no longer impact the user experience on desktop

COMPUTE PREEMPTION

UNDERSTANDING THE INTERNALS



MODERN GRAPHICS INTEROP

OVERVIEW

Legacy API has issues

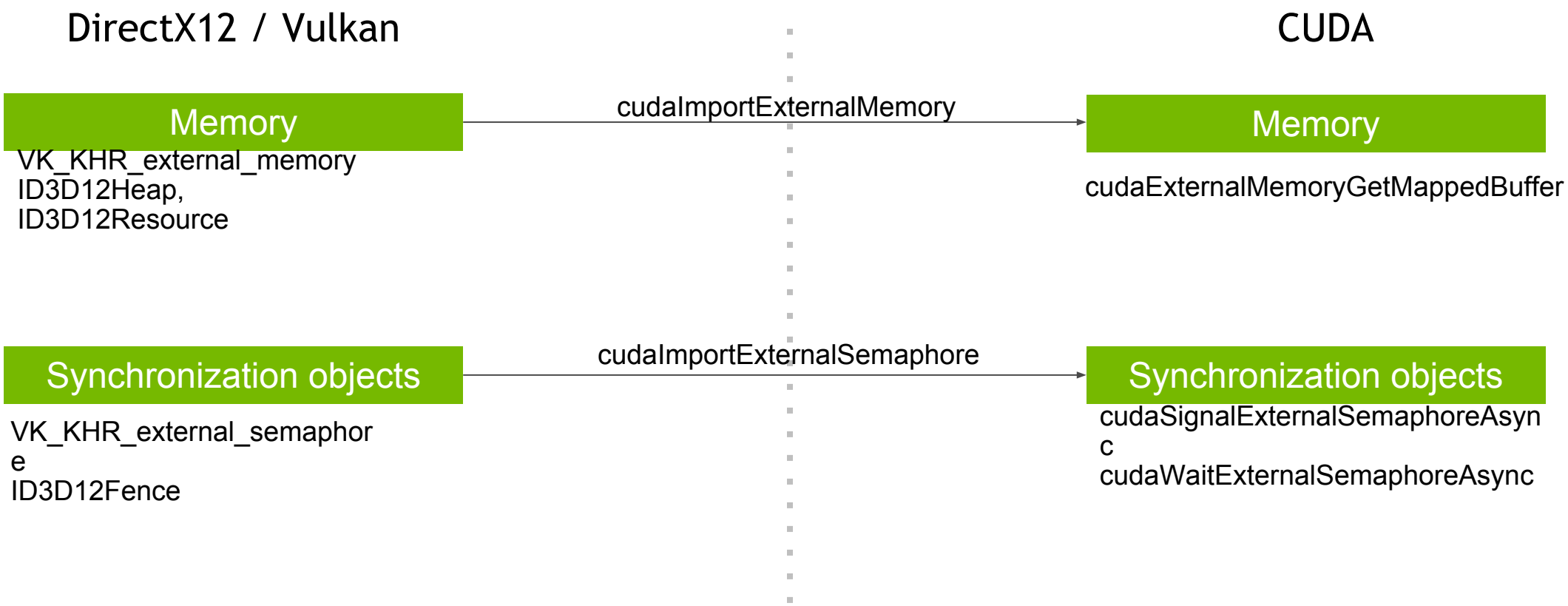
The old API (Register Resource, Map, Unmap, Unregister resource) may introduced a lot of hidden operations that are hard to control (Reallocation, creation of local copy, extra heavy synchronization …)

New API for Vulkan and DirectX12 follow an explicit model

Memory allocations (buffers or images) are imported in CUDA and the synchronization objects from graphics APIs are imported as well. Instead of an implicit synchronization and allocation model, the users is now responsible for explicit synchronization and memory management.

MODERN GRAPHICS INTEROP

OVERVIEW



CONCLUSION


QUESTIONS


NVIDIA