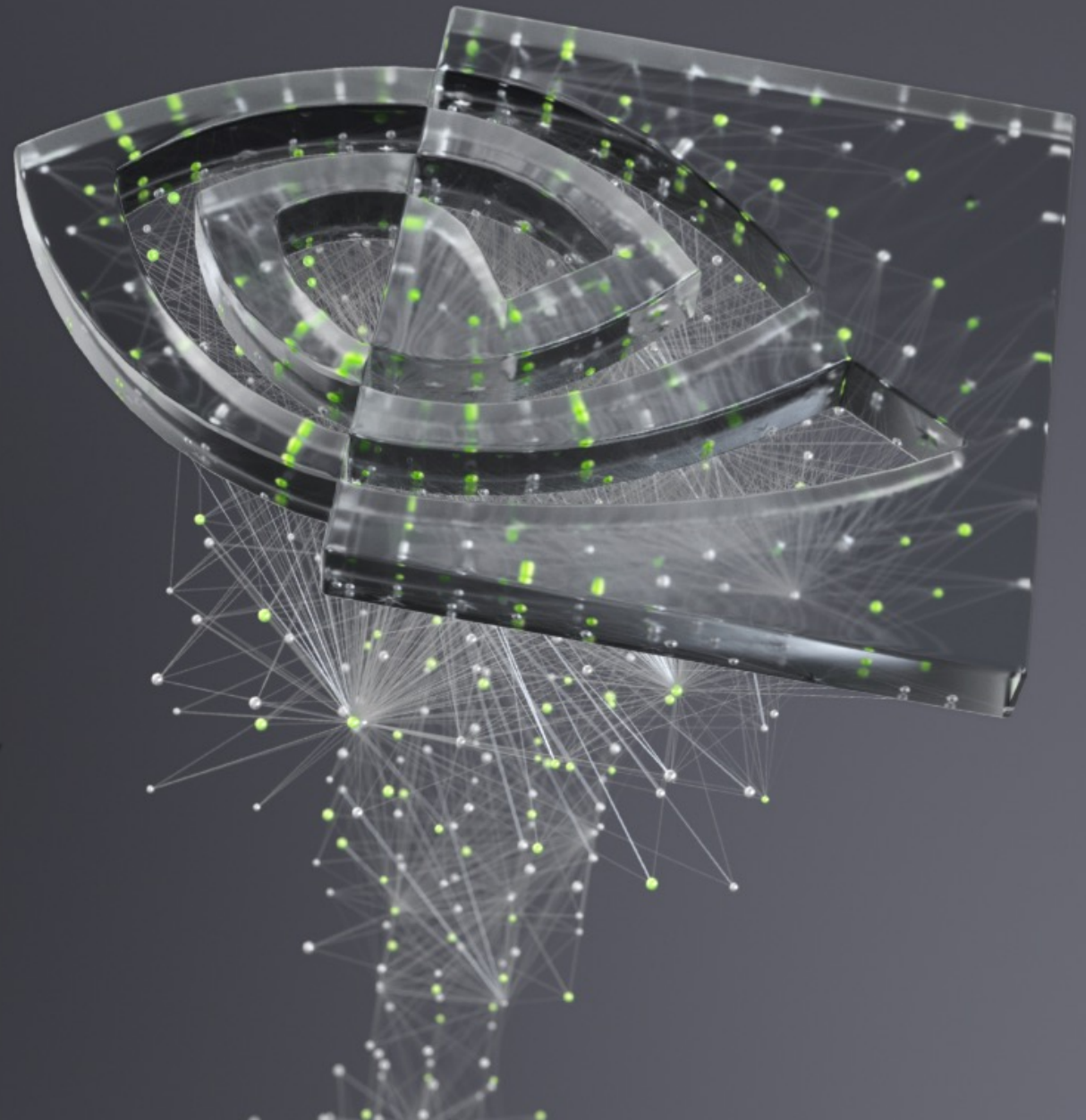


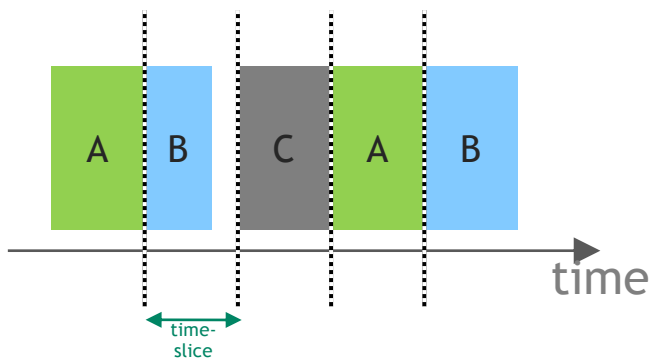
GPU CONCURRENCY

ROBERT SEARLES
5/26/2021

EXECUTION SCHEDULING & MANAGEMENT

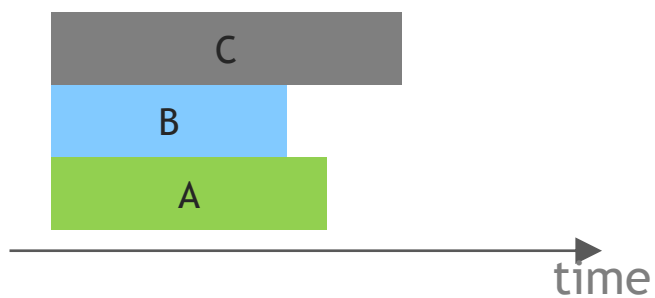
Pre-emptive scheduling

Processes share GPU through time-slicing
Scheduling managed by system



Concurrent scheduling

Processes run on GPU simultaneously
User creates & manages scheduling streams



CUDA CONCURRENCY MECHANISMS

	Streams	MPS	MIG
Partition Type	Single process	Logical	Physical
Max Partitions	Unlimited	48	7
Performance Isolation	No	By percentage	Yes
Memory Protection	No	Yes	Yes
Memory Bandwidth QoS	No	No	Yes
Error Isolation	No	No	Yes
Cross-Partition Interop	Always	IPC	Limited IPC
Reconfigure	Dynamic	Process launch	When idle

MPS: Multi-Process Service
MIG: Multi-Instance GPU

CUDA STREAMS

STREAM SEMANTICS

1. Two operations issued into the same stream will *execute in issue-order*. Operation B issued after Operation A will not begin to execute until Operation A has completed.
2. Two operations issued into separate streams have *no ordering prescribed by CUDA*. Operation A issued into stream 1 may execute before, during, or after Operation B issued into stream 2.

- Operation: Usually, cudaMemcpyAsync or a kernel call. More generally, most CUDA API calls that take a stream parameter, as well as stream callbacks.

STREAM EXAMPLES

- Host/Device execution concurrency:

```
Kernel<<<b, t>>>(…);    // this kernel execution can overlap with
cpuFunction(…);         // this host code
```

- Concurrent kernels:

```
Kernel<<<b, t, 0, streamA>>>(…);    // these kernels have the possibility
Kernel<<<b, t, 0, streamB>>>(…);    // to execute concurrently
```

  - In practice, concurrent kernel execution on the same device is hard to witness
  - Requires kernels with relatively low resource utilization and relatively long execution time
  - There are hardware limits to the number of concurrent kernels per device
  - Less efficient than saturating the device with a single kernel

INTRODUCTION TO CUDA's MULTI-PROCESS SERVICE (MPS)

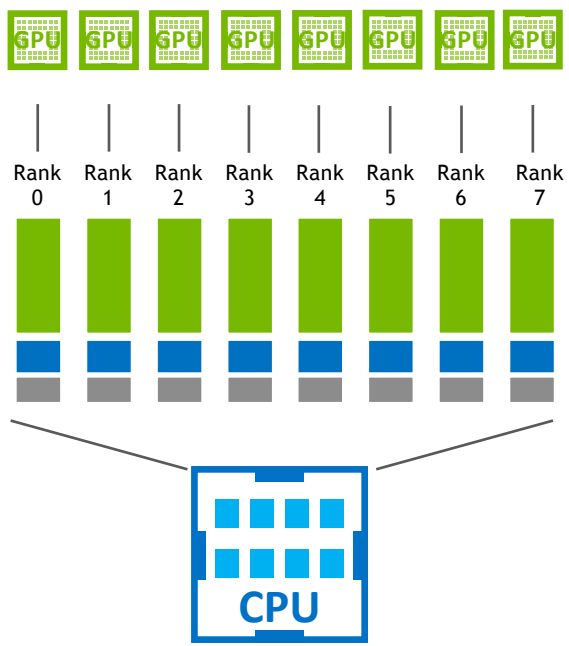
MPI DECOMPOSITION

Spread work across GPUs

Very common in HPC

Designed to **concurrently** execute multiple MPI ranks on different GPUs

Used when there is **too much work** for a single GPU.



MULTI-PROCESS SERVICE (MPS) OVERVIEW

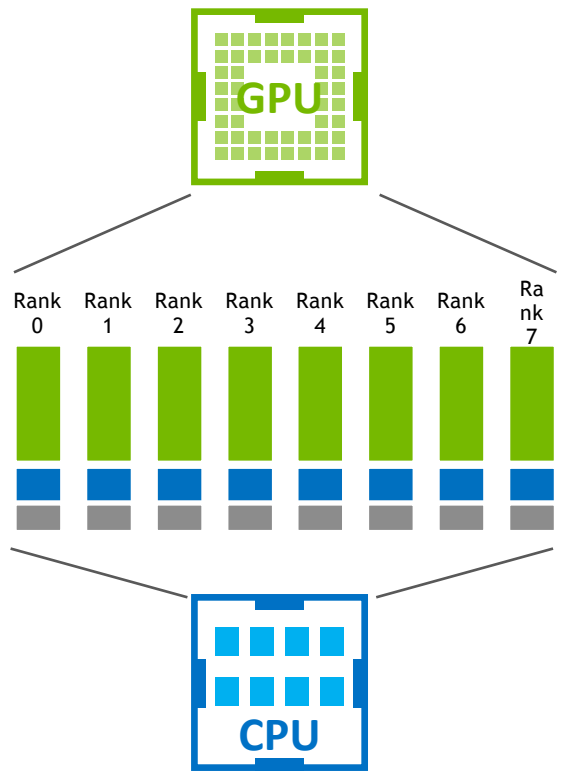
First available on Kepler architecture
(SM 3.5 - K20/K40/K80)

Significantly enhanced on Volta (SM 7.0)

Designed to **concurrently** map multiple MPI ranks onto a single GPU

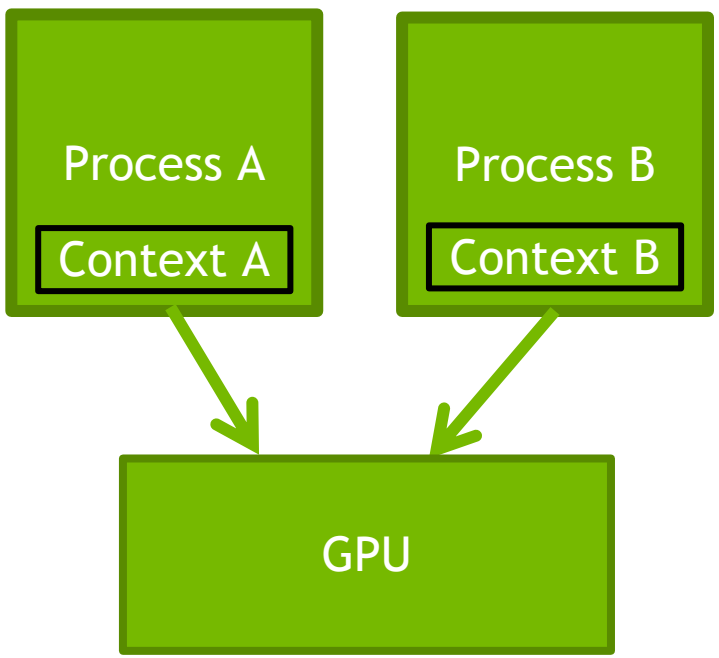
Used when each rank is **too small** to fill the GPU on its own

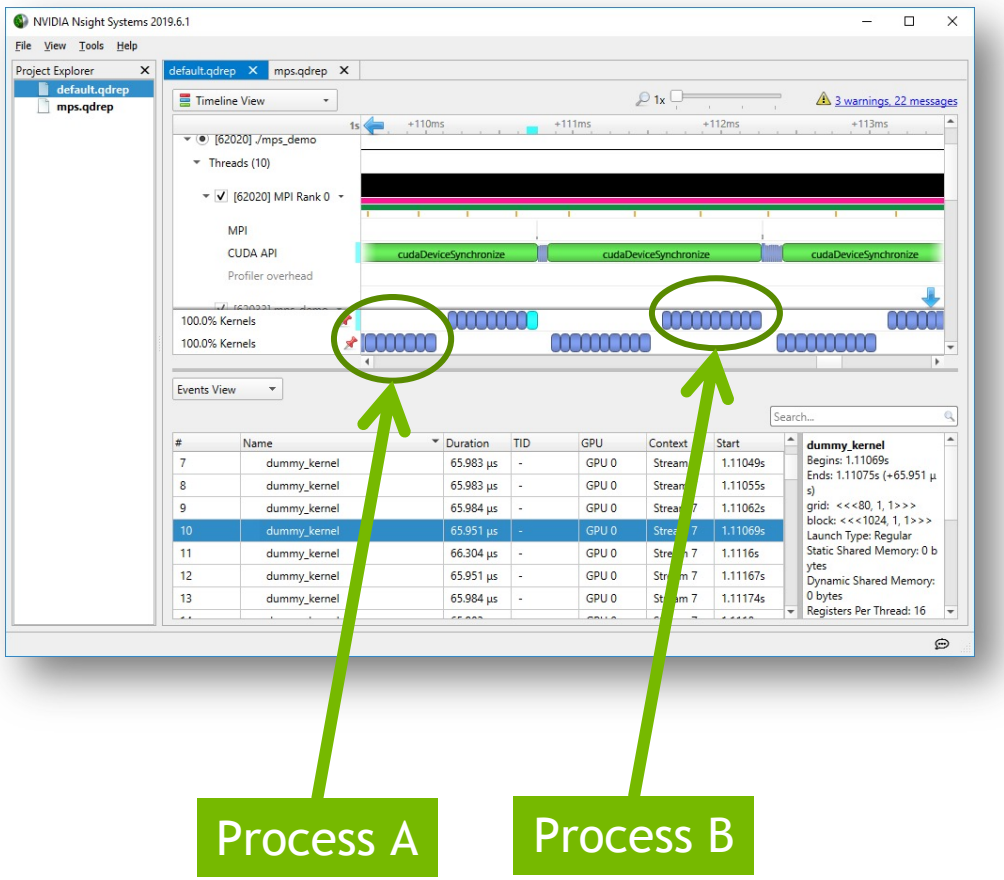
On Summit, use *-alloc_flags=gpumps* when submitting a job with *bsub*



PROCESSES SHARING GPU WITHOUT MPS

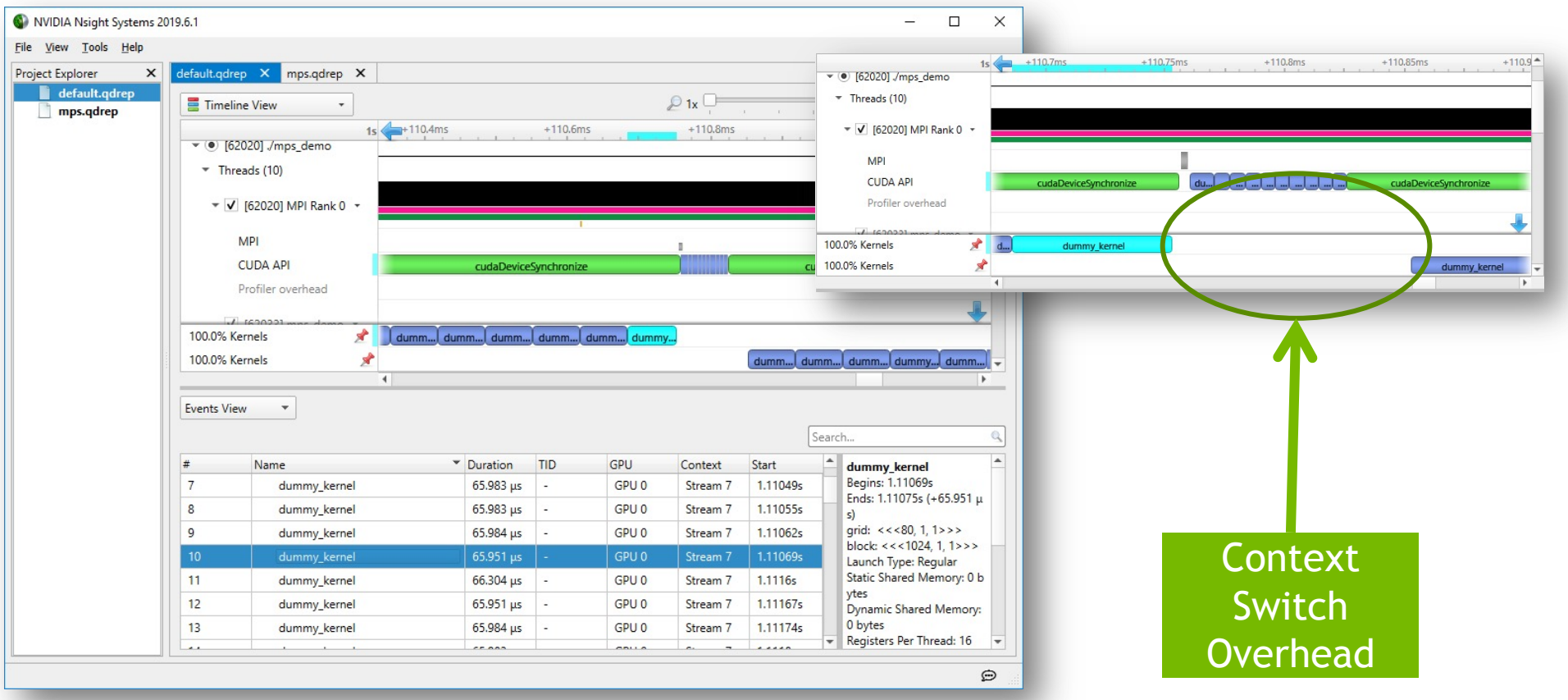
No Overlap





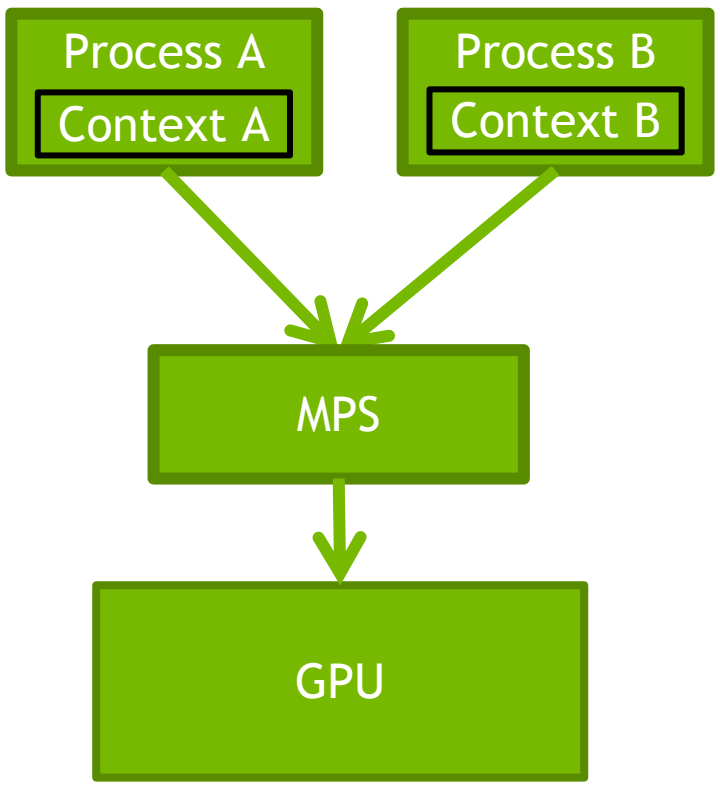
PROCESSES SHARING GPU WITHOUT MPS

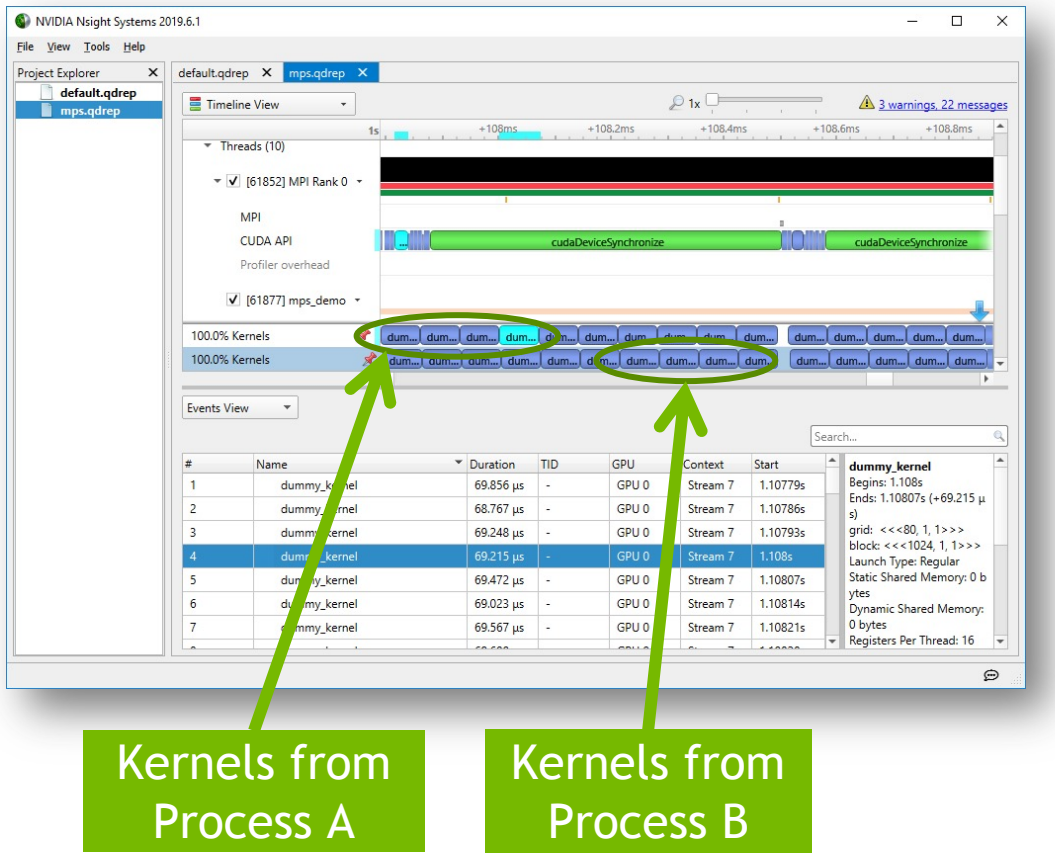
Additional small overhead arising from pre-emptive context switch



PROCESSES SHARING GPU WITH MPS

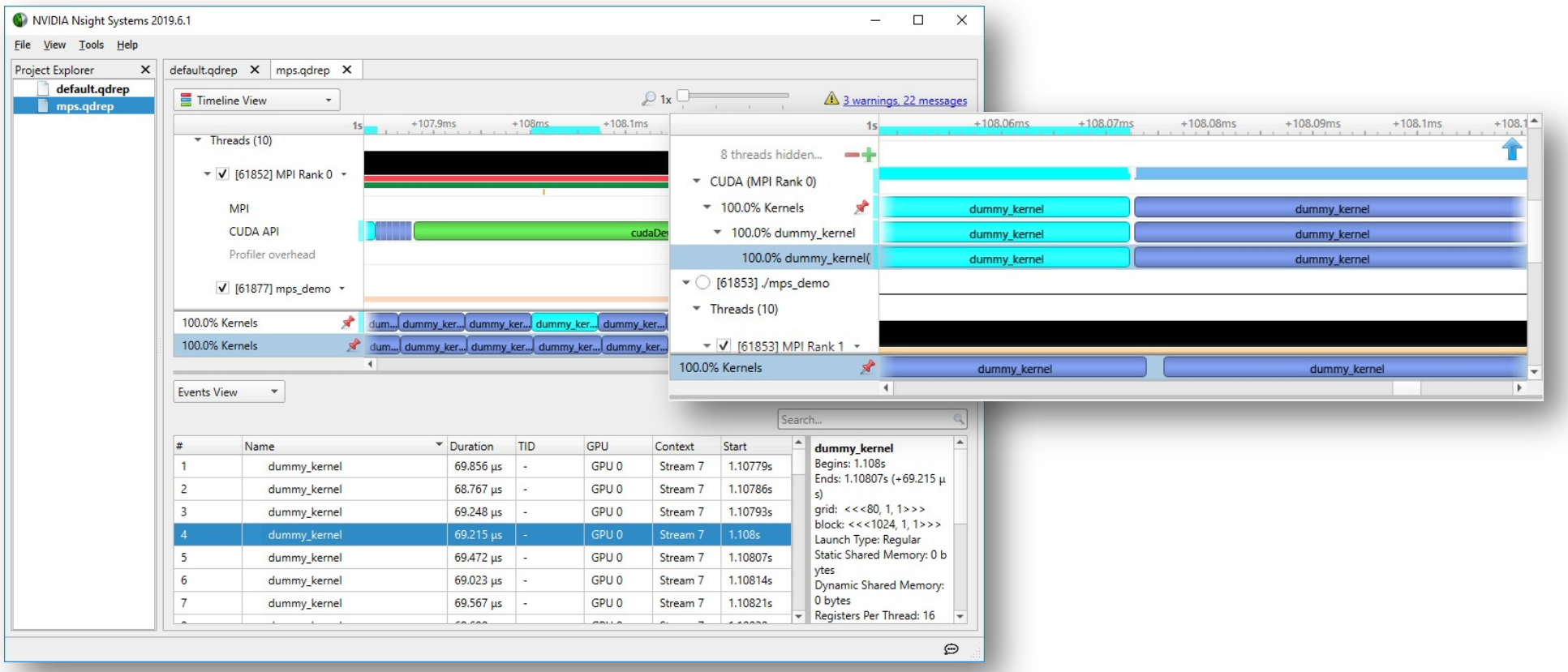
Maximum Overlap





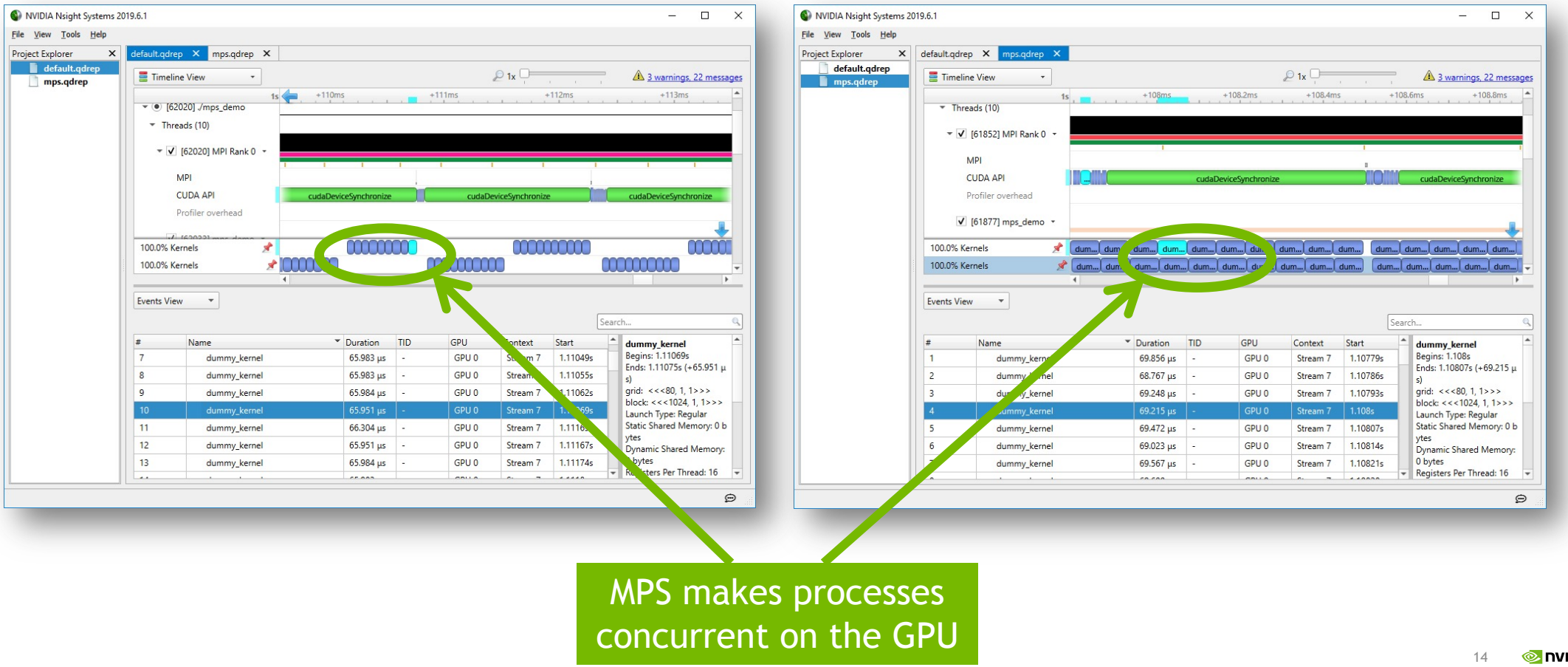
PROCESSES SHARING GPU WITH MPS

No Context Switch Overhead



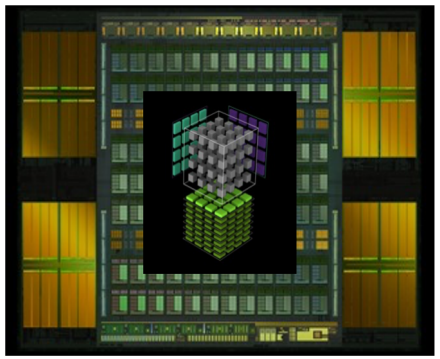
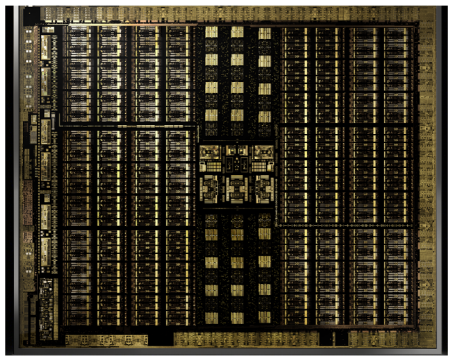
WITHOUT vs. WITH MPS, SIDE-BY-SIDE

MPS enables inter-process concurrency on the GPU

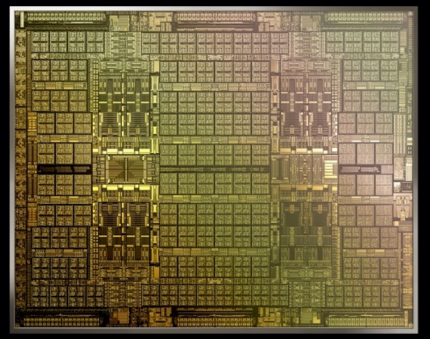


CUDA 11
Multi-Instance GPU (MIG)

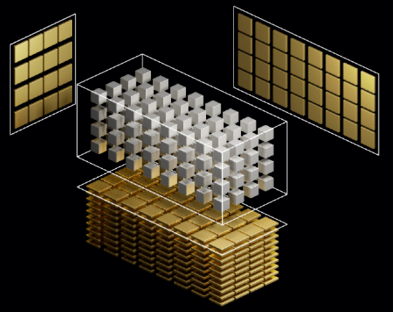
GPU ARCHITECTURE AND CUDA

CUDA 8.0	CUDA 9.0	CUDA 10.0	CUDA 11.0
			
2016	2017	2018	2020
PASCAL	VOLTA	TURING	AMPERE
HBM, NVLINK, FP16	HBM, NVLINK, TENSOR CORES, MPS	TENSOR CORES, RT CORES	HBM, NVLINK, TENSOR CORES, PARTITIONING

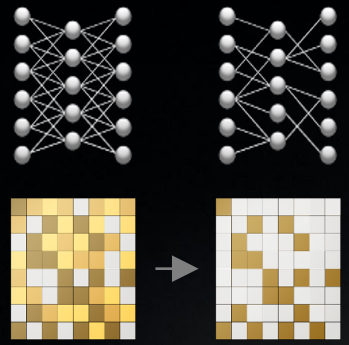
INTRODUCING NVIDIA A100



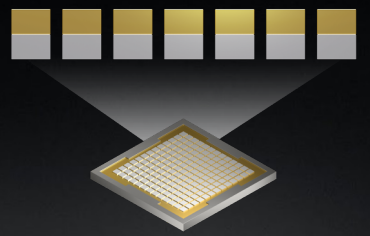
Ampere
World's Largest 7nm chip
54B XTORS, HBM2



3rd Gen Tensor Cores
Faster, Flexible, Easier to use
20x AI Perf with TF32



New Sparsity Acceleration
Harness Sparsity in AI Models
2x AI Performance

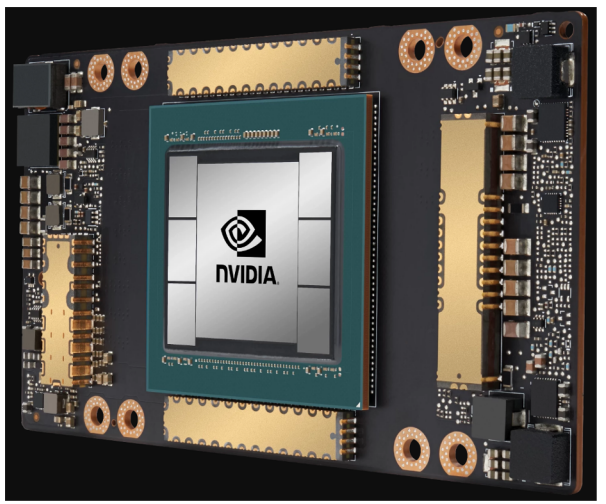


New Multi-Instance GPU
Optimal utilization with right sized GPU
7x Simultaneous Instances per GPU

3rd Gen NVLINK and NVSWITCH
Efficient Scaling to Enable Super GPU
2X More Bandwidth

NVIDIA A100: SPECS

	V100	A100
SMs	80	108
Tensor Core Precision	FP16	FP64, TF32, BF16, FP16, I8, I4, B1
Shared Memory per Block	96 kB	160 kB
L2 Cache Size	6144 kB	40960 kB
Memory Bandwidth	900 GB/sec	1555 GB/sec
NVLink Interconnect	300 GB/sec	600 GB/sec
FP64 Throughput		9.7 | 19.5 TFLOPS
TF32 Tensor Core	N/A	156 | 312 TFLOPS



MULTI-INSTANCE GPU (MIG)

Optimize GPU Utilization, Expand Access to More Users with Guaranteed Quality of Service

Up To 7 GPU Instances In a Single A100:
Dedicated SM, Memory, L2 cache, Bandwidth for hardware QoS & isolation

Simultaneous Workload Execution With Guaranteed Quality Of Service:
All MIG instances run in parallel with predictable throughput & latency

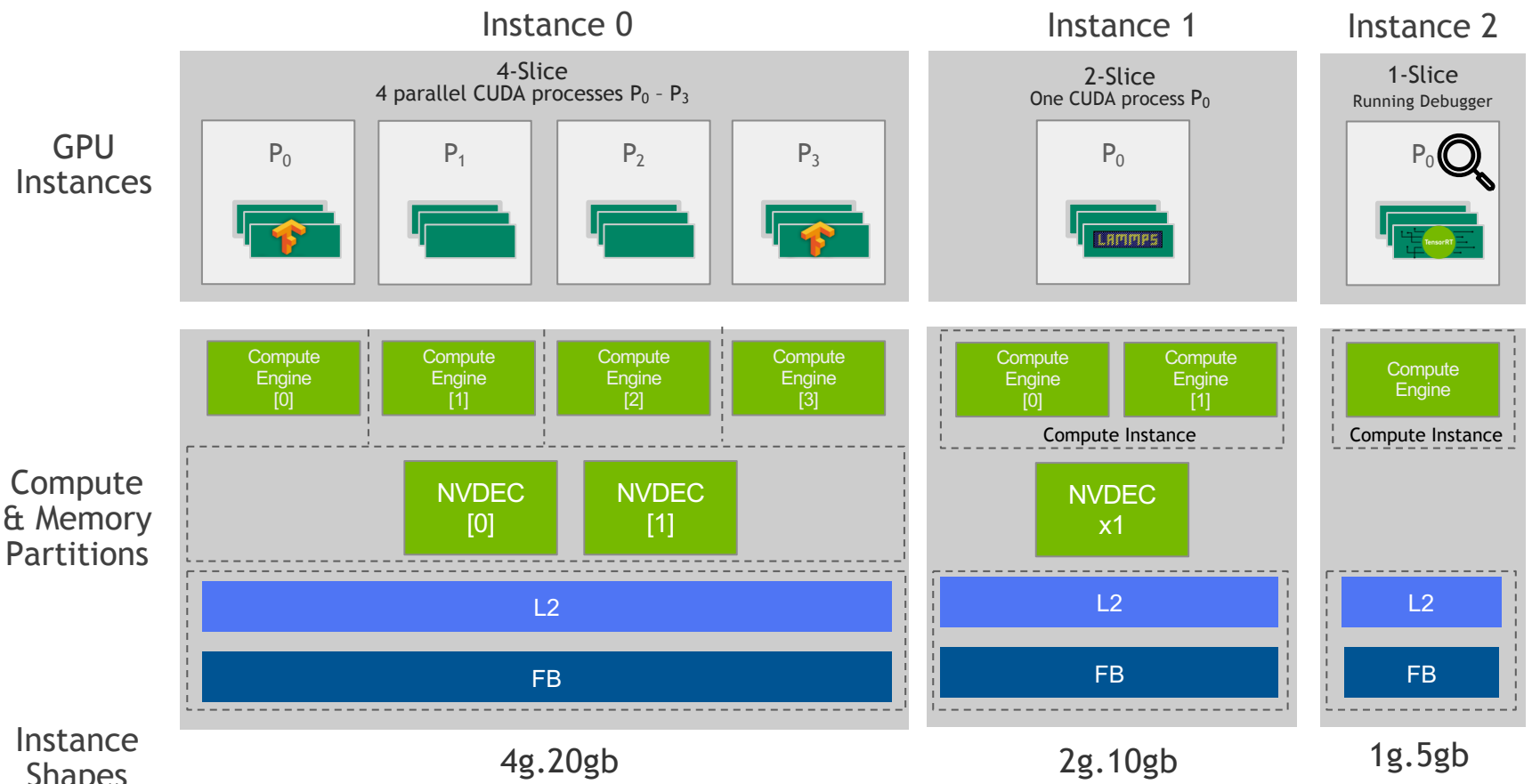
Right Sized GPU Allocation:
Different sized MIG instances based on target workloads

Diverse Deployment Environments:
Supported with Bare metal, Docker, Kubernetes, Virtualized Env.

MIG User Guide: https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html

EXAMPLE: TWO LEVEL PARTITIONING

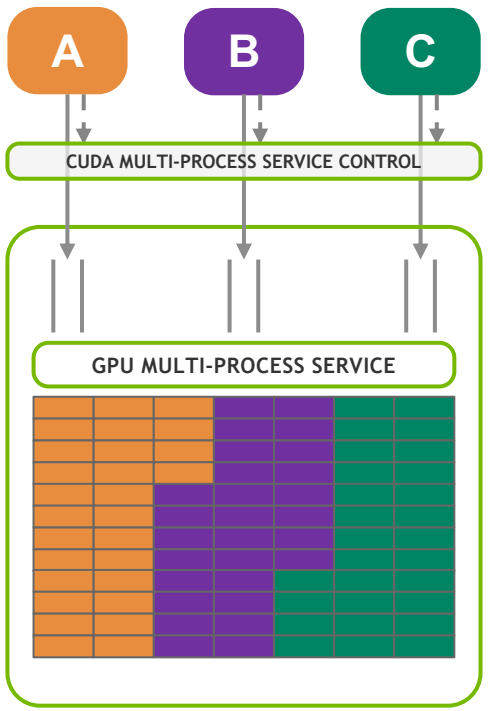
A100-40GB – Up to 7 GPUs



MIG ISOLATION

- Computational Isolation
  - Streaming Multiprocessors (SMs) are *not* shared between GPU Instances
  - This provides high Quality of Service (QoS) for each GPU Instance
- DRAM Bandwidth Isolation
  - Slices of the L2 Cache are physically associated with particular DRAM channels and memory
  - Isolating MIGs to non-overlapping sets of L2 Cache slices does two things:
    - Isolates BW
    - Allocates DRAM memory between the MIGs
- Configuration Isolation
  - Creating GPU Instances or Compute Instances do not disturb work running on existing instances
- Error Isolation
  - Resources within the chip are separately resettable

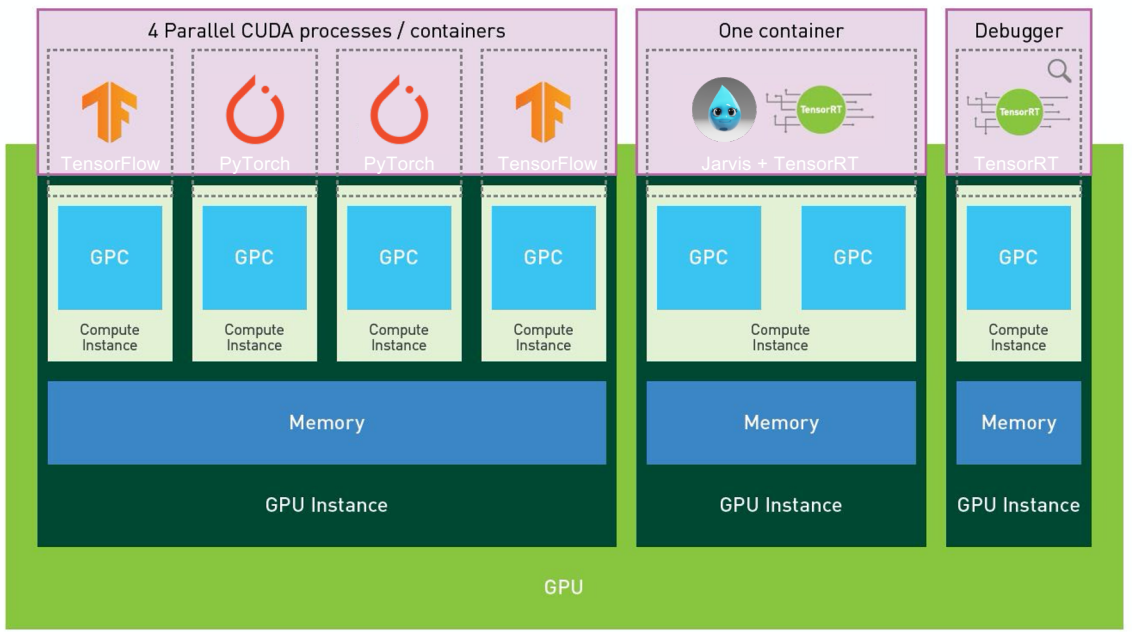
LOGICAL VS. PHYSICAL PARTITIONING



Multi-Process Service

Dynamic contention for GPU resources

Single tenant



Multi-Instance GPU

Hierarchy of instances with guaranteed resource allocation

Multiple tenants

CUDA BEHAVIOR AND DEVELOPER TOOLS

Can CUDA code continue to run on a separate part of the chip when we re-configure the chip?

Yes, as long as you don't reconfigure the partition the code is running on

All CUDA code will have to stop to enable/disable MIG mode

What is the difference between "compute instance" and "GPU instance"?

GPU instances can be further sub-divided into compute instances (one per slice or GPC) that share memory. It is a more advanced use-case.

What is difference between CUDA streams, MPS, and MIG?

MPS does not HW partition the processes

MIG partitions jobs each with dedicated compute, memory, and memory bandwidth

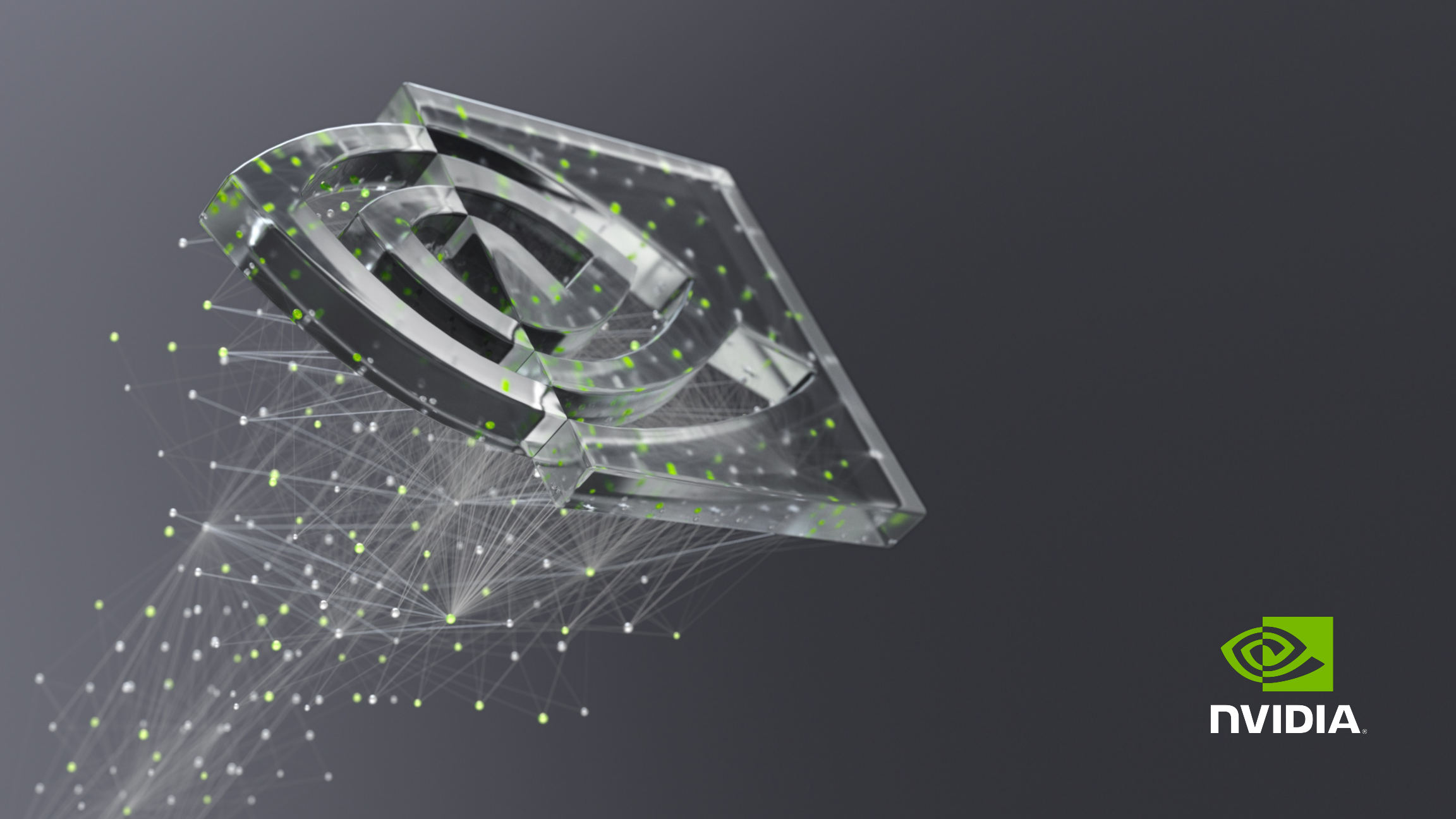
CUDA stream: Per process, no hardware isolation

Will MIG replace MPS

MPS is a feature of CUDA and will continue as a feature with roadmap, it is not meant to be a "replacement for" MPS or fixing any MPS bug.

FUTURE MODULES

- Multi-Threading + CUDA Streams
- MPI/MPS + CUDA
- Debugging Tools
  - Compute sanitizer & cuda-gdb


NVIDIA