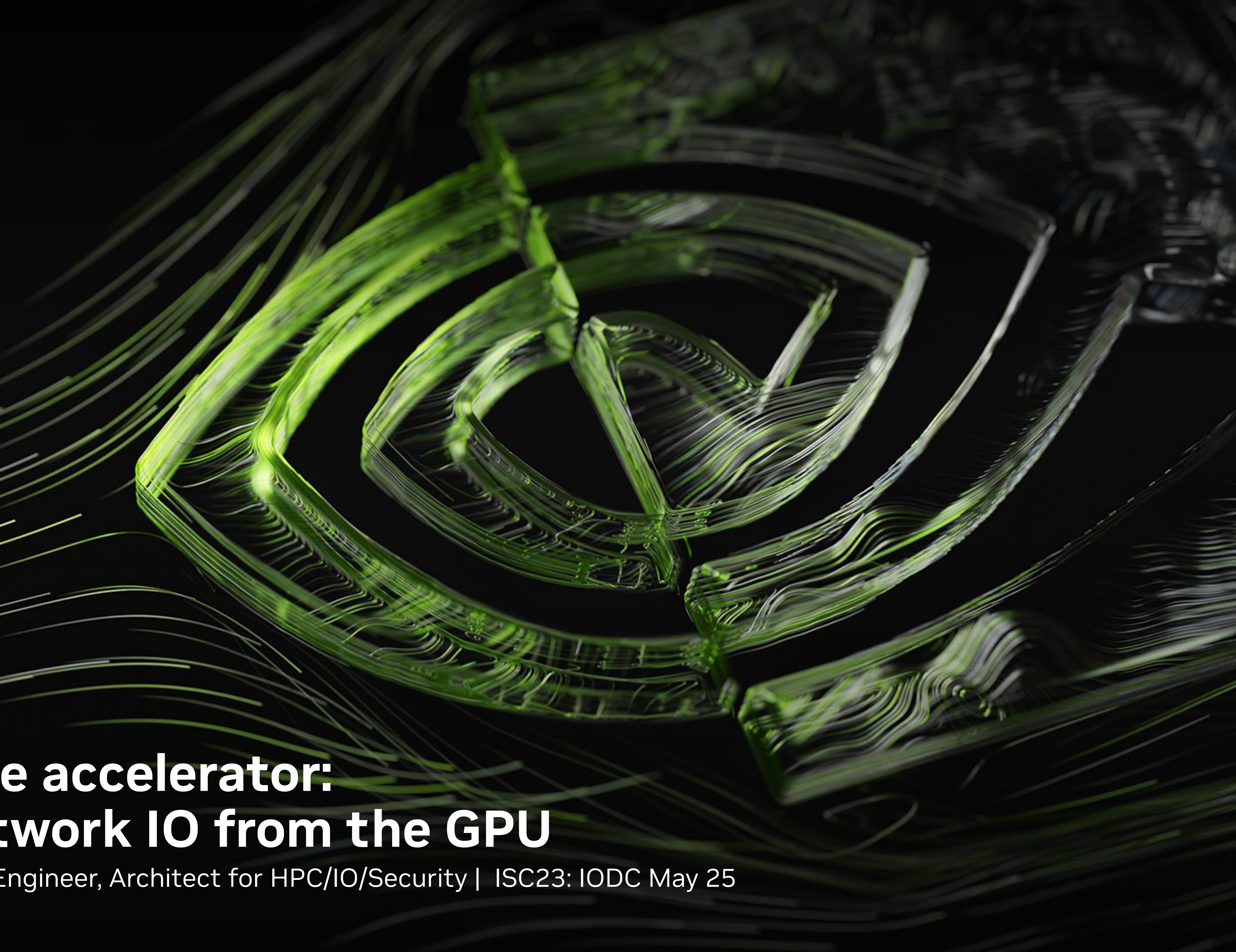


Initiate from the accelerator: storage and network IO from the GPU

Dr. CJ Newburn, Distinguished Engineer, Architect for HPC/IO/Security | ISC23: IODC May 25

Agenda

- Control and asynchrony
- GPU-initiated storage
  - Problem and usage models
  - AWS and NVGNN examples
- GPU-initiated storage POC
- GPU-initiated networking results

Control sources

Each of CPU, DPU, GPU can control NVMe storage

$$\begin{tikzcd} \text{CPU} \arrow[dr, dash] & \\ \text{GPU} & \text{NVMe} \arrow[l] \end{tikzcd}$$

$$\begin{tikzcd} \text{NW} \arrow[r] & \text{DPU} \arrow[d, dash] \\ & \text{NVMe} \end{tikzcd}$$

$$\begin{tikzcd} & \text{[CPU]} & \\ \text{GPU} & & \text{NVMe} \arrow[ll] \end{tikzcd}$$

	CPU	DPU	GPU
Usage:	read/compute/write	JBOF	RecSys, GML, GNN
Interface:	cuFile	NVMf	GPU-initiated
Coord:	orchestrate DMAs	stage data to/from media	spcl staging
Data:	bypass CPU	system cache or CMB	NVMe
Gain:	↑bw, ↓CPU utilization	max bandwidth, ↓cost	concurrency

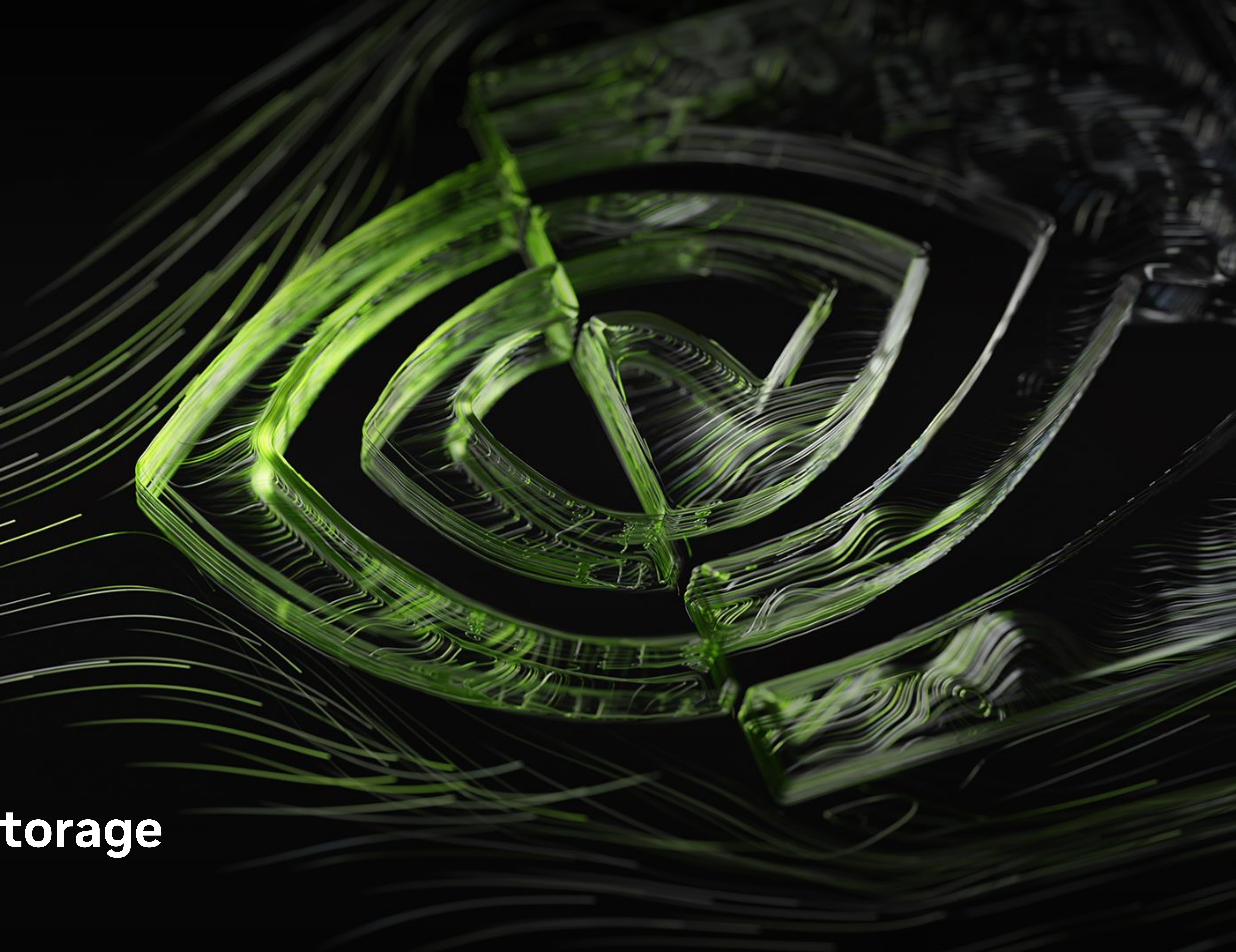
NVIDIA GPUDirect™

Variants on who decides what to do and when to do it

Increasing autonomy for GPU

GPUDirect (non-Async)	**GPUDirect Async**	
CPU initiated (prepared, triggered)	**CPU prepared, GPU triggered**	**GPU kernel initiated**
Video: GPUDirect Video Local GPU: GPUDirect Peer-Peer P2P Remote master: GPUDirect RDMA GDR Storage: GPUDirect Storage GDS	Stream triggered GDA-ST Graph triggered GDA-GT Kernel triggered GDA-KT	Network GDA-KI Network NVSHMEM blog GPUNetIO (blog) Storage GDA-KI Storage

- GPUDirect enables direct data movement to and from the GPU, without staging in CPU
  - As memory becomes migratable, the source/target may happen to be in GPU or CPU
- Network and storage IO involves
  - Preparation: create work request for an IO device, e.g. work queue entry on a cmd queue
  - Triggering: hand off work request to/sync with an IO device, e.g. ring a doorbell



GPU-initiated storage

Problem: Large volume of random fine-grained accesses

GPU concurrency is key to throughput

- Large volume of random fine-grained accesses → large concurrency to maximize tput
  - GPU $>>_{\text{concurrency}}$ CPU; control and data path would be bottlenecked on CPU
- Data consumed on GPU; requests may also be generated there
  - Feeding data through CPU becomes a bottleneck
- Criticality assumptions about rates
  - tput = min(GPU request generation, $ bw, NVMe access for misses, data consumption on GPU)
- GPU advantages over CPU

Generating requests	more threads generating requests
Requests to local cache latency	more threads accessing in parallel, tolerant of latency
Making requests to NVMes	more threads generating NVMe requests
Consuming data	more threads/other acceleration features like tensor cores

GPU-initiated storage usage models

Large batches of small IOs to GPU memory requiring efficient KeyValue APIs

- Graph neural network
  - Widely used in fraud detection, fake reviews, tracking bot assaults, recommendation systems
  - Nodes (millions to billions) and edges (billions to trillions) graphs
  - Each node and edge has embeddings of size upto 4KB (>10TB)
- Recommender systems
  - Training pipeline - 10-100 TB embedding models requiring fine-grain access
  - Data ingestion pipeline - requiring efficient preprocessing such as filtering and reconstruction
- Data analytics (cuDF, Spark from RAPIDs)
  - Spill management, shuffle management on billion rows
  - Cost reduction
- Omniverse
  - Low-latency persistent texture objects with multiple simultaneous clients
- Vector Search
  - Billions of documents represented as vectors (~20PB)
- Graph Analytics in ML - current cuGraph only supports if graph is in memory
  - Nodes (millions to billions) and edges (billions to trillions)
  - Require the graph in memory address space

Scale GNN training and memory requirements

	Graph Size	Total memory size	Node feature size	Reduce memory consumption with NVMe by (%)
OGBN-papers100M	#100M nodes #1.6B edges 128 node features	100 GB	52 GB	52%
MAG-LSC	#240M nodes #3.4B edges 768 node features	224 GB	174 GB	78%
Future target	O(10-100B) nodes O(100B-1T) edges	100 TB to >1 PB	100 TB to >1 PB > feature richness	improves with > nodes, compression

- Future: fits in O(1-10K) GPUs@O(100GB) HBM,O(100-1K) CPUs@1TB DRAM,O(10-100) NVMes@15.4TB
- Buying GPUs for their HBM isn't cost effective.
- NVMes are way less expensive than HBM or DDR, and enable scaling to much larger graphs
- Conclusion: if performance to NVMe can keep up, that's way more cost effective

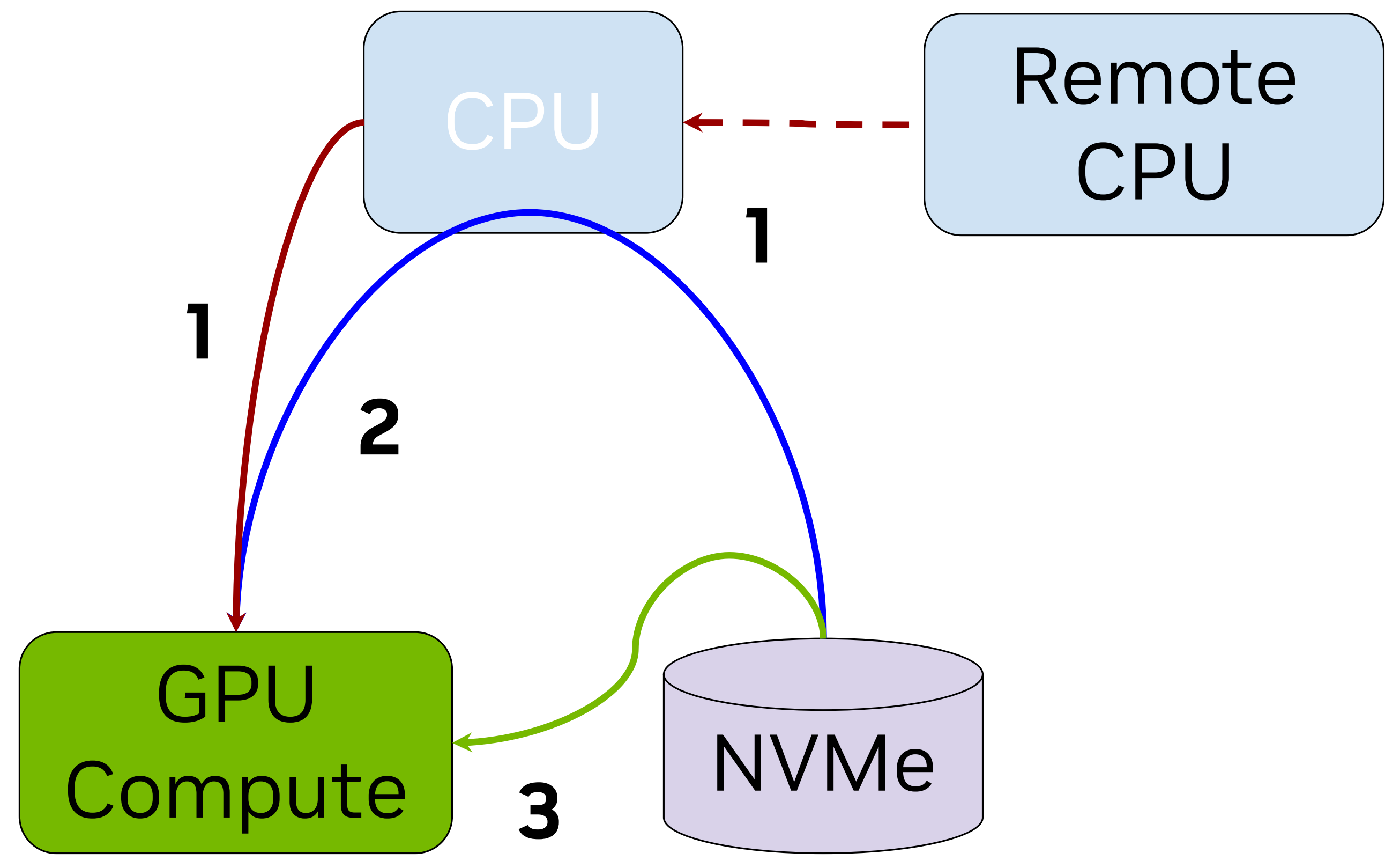
Distributed GNN training pipeline

Mini-batch training steps:

- Sample mini-batch
- Copy node/edge features
- Mini-batch computations

The data copy throughput required in each step.

	Mini-batch sampling (CPU)	Node feature copy (1: CPU-CPU-GPU)	Node feature copy (2: NVMe-mmap)	Mini-batch computation (GPU)
OGBN-papers100M	3407 MB/s	1020 MB/s	40.2 MB/s	6813 MB/s
MAG-LSC	4733 MB/s	1241 MB/s	41.2 MB/s	4730 MB/s



AWS system info:
g4dn.metal with T4 GPUs
(2560 CUDA cores @ 585MHz)
Data for 3 will be shown below

- CPU, NVMe can't keep up, need a better solution
- But fast-enough NVMe reduces memory consumption, enables lower cost

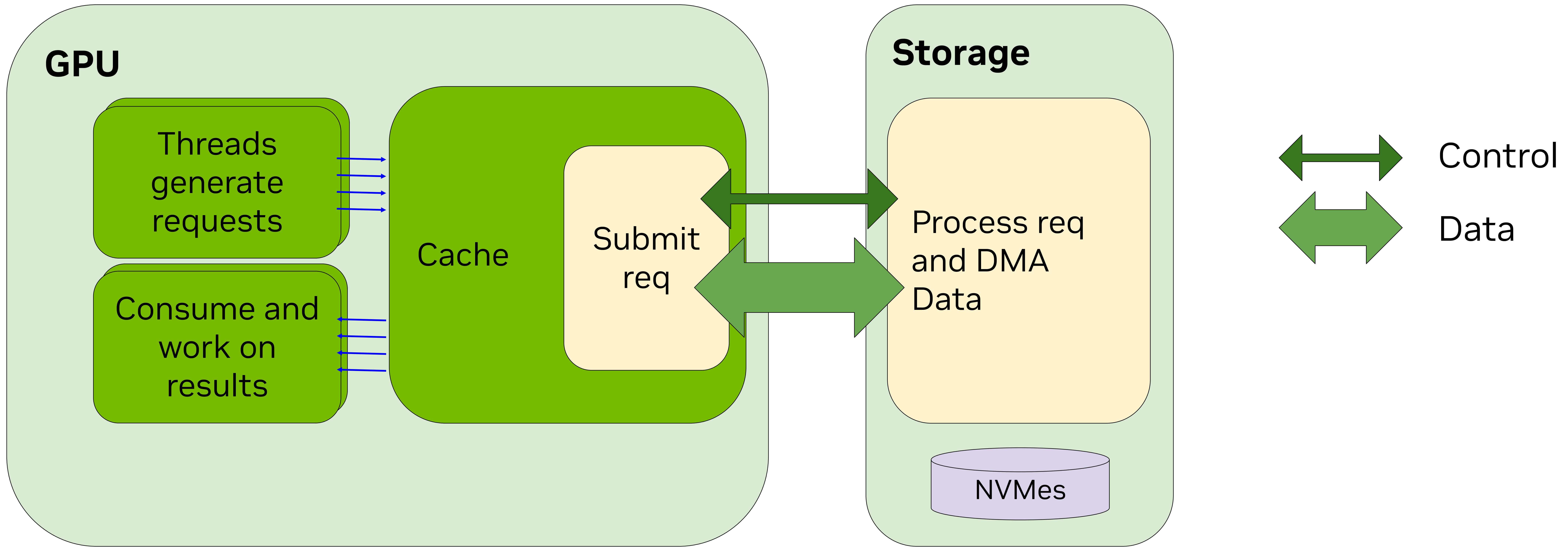
GPU-initiated storage design goals and requirements

POC in research, not a committed product yet

- Design goals
  - New APIs to access storage IO from GPU
  - Maximize throughput for large batches of small data accesses
  - Scale to problem sizes too big to fit into GPU HBM or CPU DDR with cheaper NVMe
  - Relieve NVMe IOPs bottleneck with GPUs vs. CPUs
- Design requirements
  - In case there’s any temporal or spatial locality to the data
    - Bandwidth out of the cache in the GPU >> PCIe bandwidth into the GPU
    - Make the cache line size match the block size to enable aggregation into block accesses
  - Storage capacity provided by NVMes
  - NVMe bandwidth is maximized by issuing concurrent accesses on abundant GPU threads

GPU-initiated storage architecture

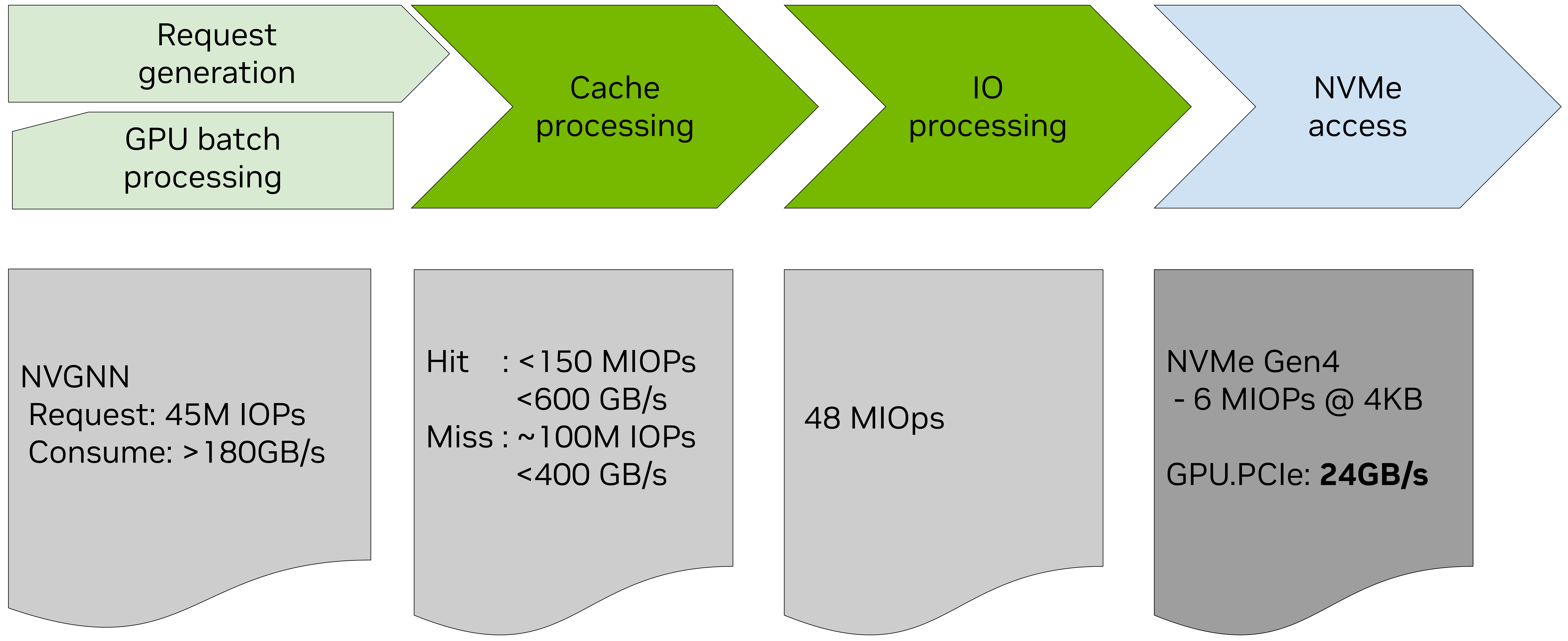
Ultimate removal of the CPU as a bottleneck for storage



- Request, initiation, service, consumption all happen on the GPU
- GDA KI Storage enables storage IO accesses that are both initiated and triggered by GPU
- Features a key pillar of Magnum IO: flexible abstraction

Performance results

Balanced end to end system matches best-available throughput



Per GPU tput on A100
6910 CUDA cores @1.41GHz
Transfer size = 4KB

Request generation / GPU batch processing	Cache processing	IO processing	NVMe access
NVGNN Request: 45M IOPs Consume: >180GB/s	Hit : <150 MIOPs <600 GB/s Miss : ~100M IOPs <400 GB/s	48 MIOps	NVMe Gen4 - 6 MIOPs @ 4KB GPU.PCIe: **24GB/s**

- Data lookup acceleration enables higher throughput by reducing the IO bottleneck to (feature) data
  - transparent data reuse benefit: cache bw (400-600 GB/s) >> PCIe into the GPU (24 GB/s)
  - IO processing (48 MIOPs) keeps up with PCIe-saturating NVMe IOPs rates (6-48 MIOPs)
- GPUs are latency tolerant - HW context switching covers miss latency

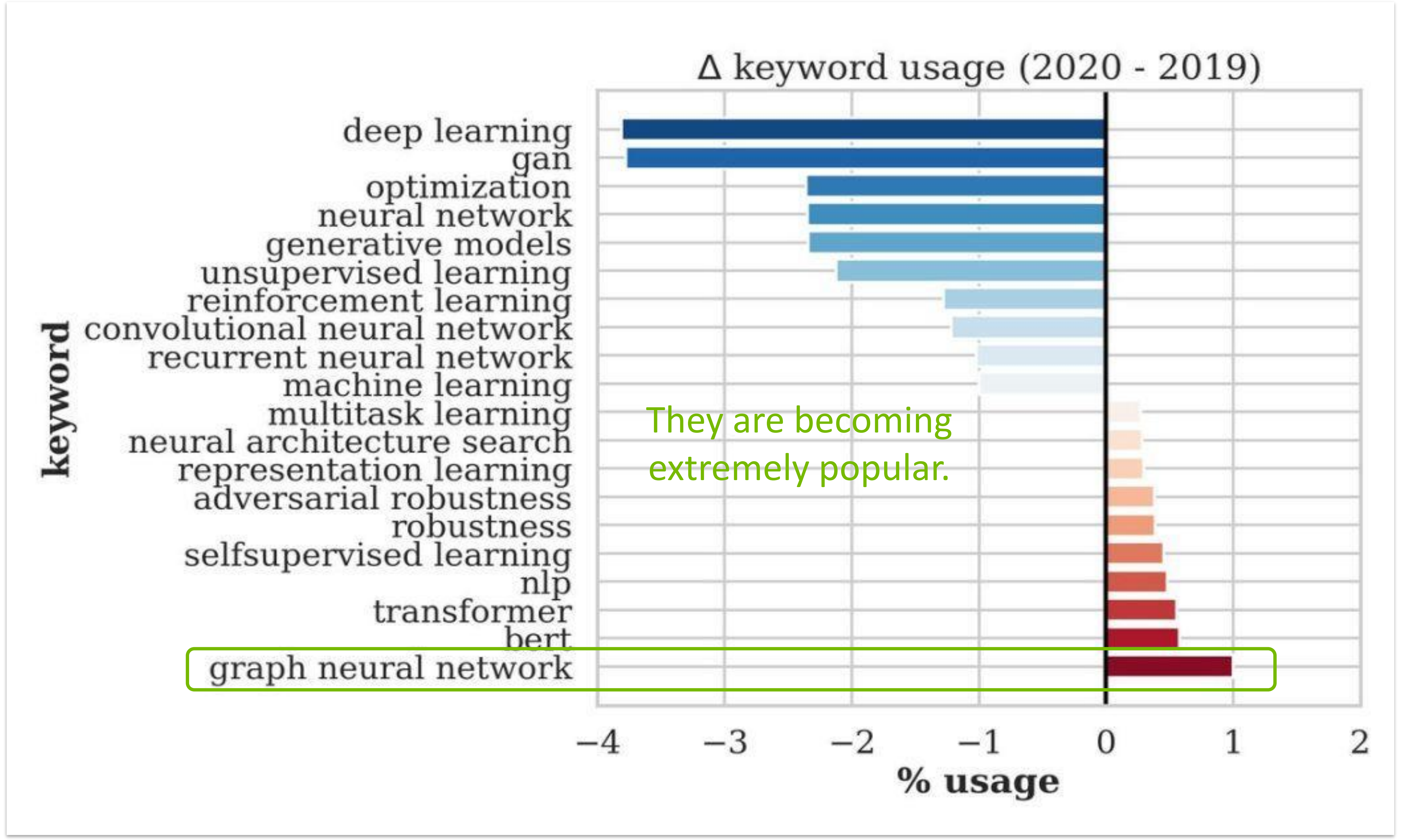
Performance

Measured bandwidth saturating Gen4
on MAG240M dataset for GraphSAGE model using Samsung 173X SSDs

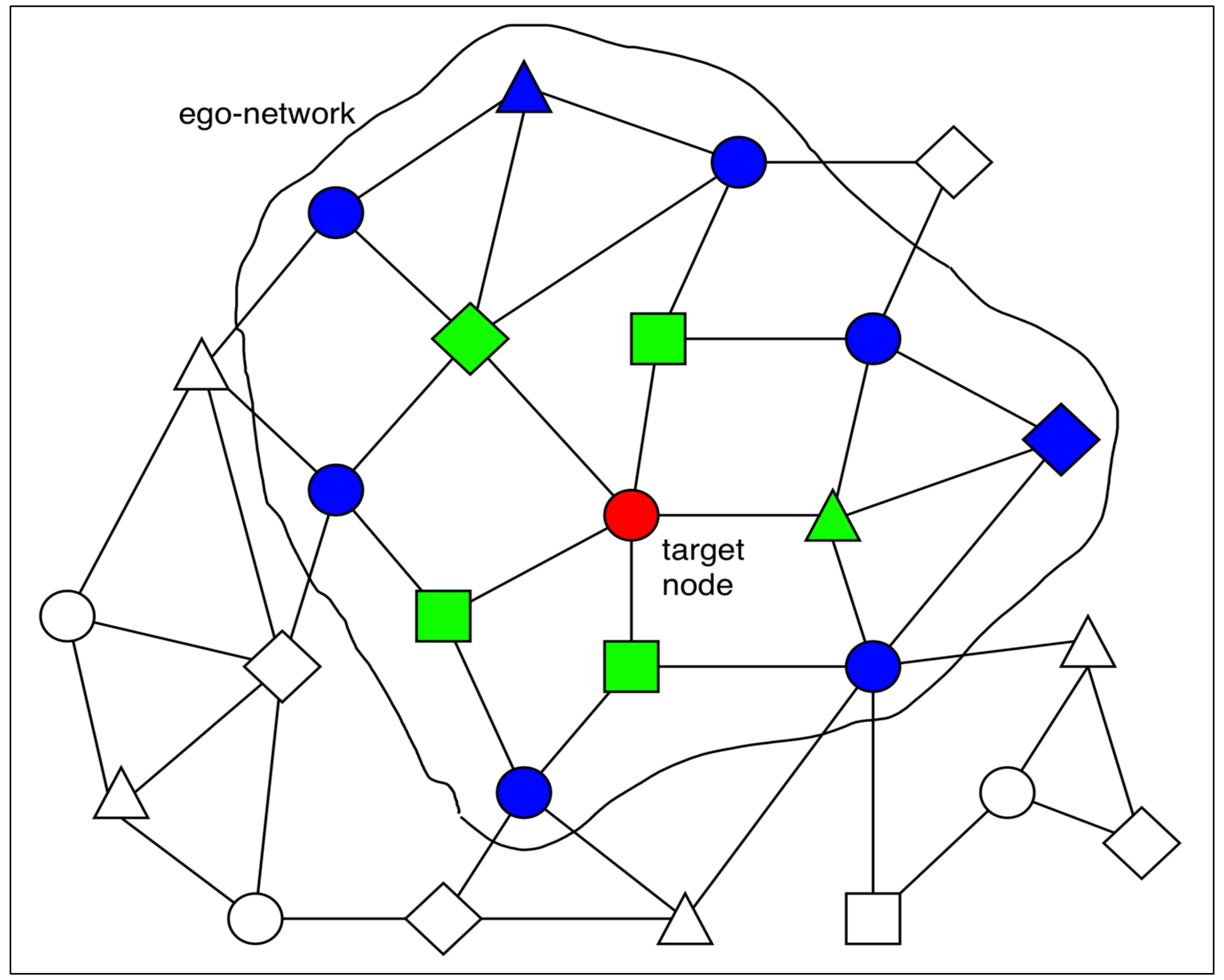
Bandwidth
IOPs
IO Bandwidth Achieved (GBps)
Million IOPs
Batches
0
5
10
15
20
25
30
0
1
2
3
4
5
6
7
1 … 100

Graph neural network (GNN)

A family of (deep) neural networks that learn node, edge, and graph embeddings



How do GNNs work?



Ego-network around each node is used to learn an embedding that captures task-specific info

The embeddings use both the structure of the graph and the features of the nodes and edges

Embeddings are learned in E2E fashion; predictions are a function of target nodes’ ego-network

AWS applicability

GPU-based data delivery rate relieves bottleneck, saves $, motivates a GPU-based sampler

	Mini-batch sampling @AWS (CPU)	Node feature copy			Mini-batch computation @AWS (GPU)
		1: CPU-CPU-GPU @AWS	2: NVMe-mmap @AWS	3: GPU-initiated storage @NV subsystem	
OGBN-papers100M	3407MB/s	1020MB/s	40.2MB/s	up to 24 GB/s	6813MB/s
MAG-LSC	4733MB/s	1241MB/s	41.2MB/s	up to 24 GB/s	4730MB/s

   - Bandwidth significantly exceeds current alternatives of CPU and NVMe via CPU
   - Much lower TCO while mitigating data access bottlenecks
   - Provides massive memory address space and scales with the graph size
   - Simplifies the programming to the storage
   - Minimizes the cost of the graph partitioning
- Priorities among GPU-initiated storage future directions
   - Distributed
   - Unified data access API to hide all complexity
   - Prefetching

*

GPU-initiated storage: Current POC limitations

Effective proof of concept with extensible architecture

- Scale
  - Single node with CPU/GPU/NVMe
- Access APIs
  - Memory array abstraction
- Loading NVMe
  - NVMe is preloaded
  - Requests always hit in NVMe tier
- Not integrated into AWS's E2E system

Credits

AWS

- Da Zheng, Sr. Applied Scientist, AWS, ML framework/algo lead

GPUDirect Storage team

- Kiran Kumar Modukuri, Zhen Zeng, Sourab Gupta, Rebanta Mitra, Prashant Prabhu, Aniket Borkar, Vahid Noormofidi, Sandeep Joshi, Salah Chaou

NVIDIA Research team

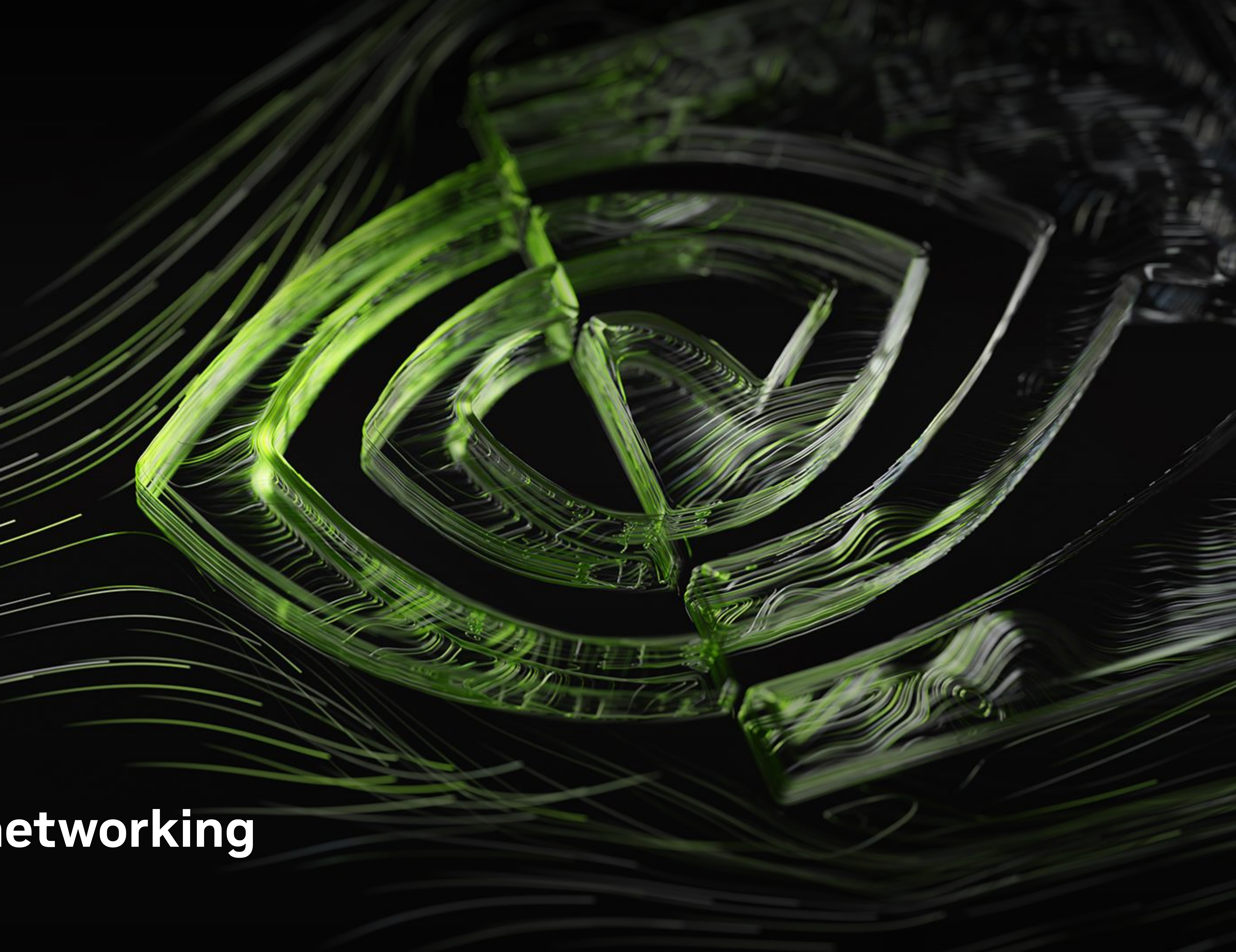
- Wen-mei Hwu, Isaac Gelado, Vikram Sharma Mailthody, Zaid Qureshi

NVIDIA GNN team

- Kyle Kranen, Nicolas Castet, Onur Yılmaz, Joe Eaton

UIUC

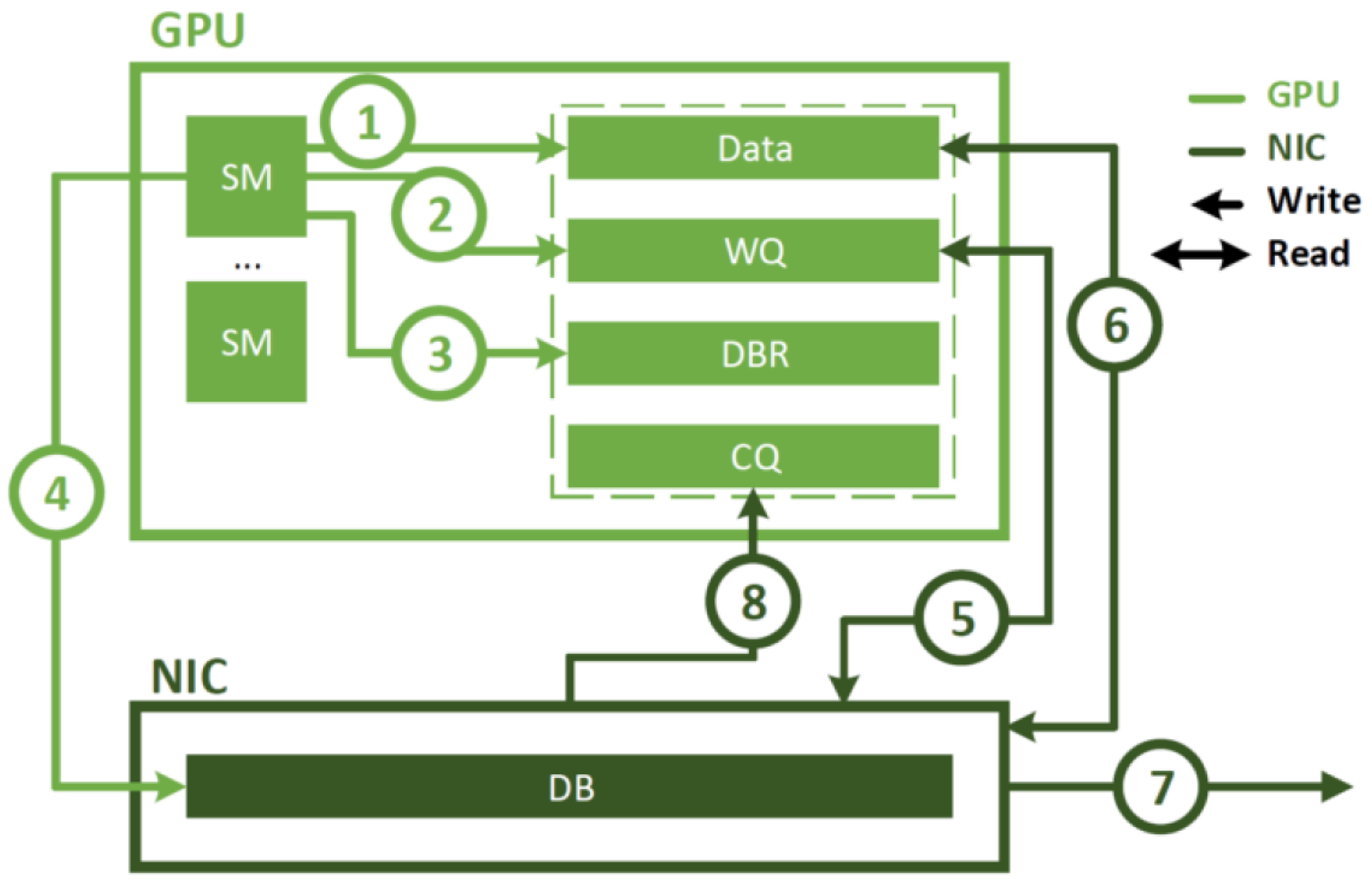
- Arpandeep Khatua, Jeongmin Park



GPU-initiated networking

GPU-initiated networking: NVSHMEM

A GPU-initiated version of OpenSHMEM/PGAS for fine-grained interleaving of compute and communication



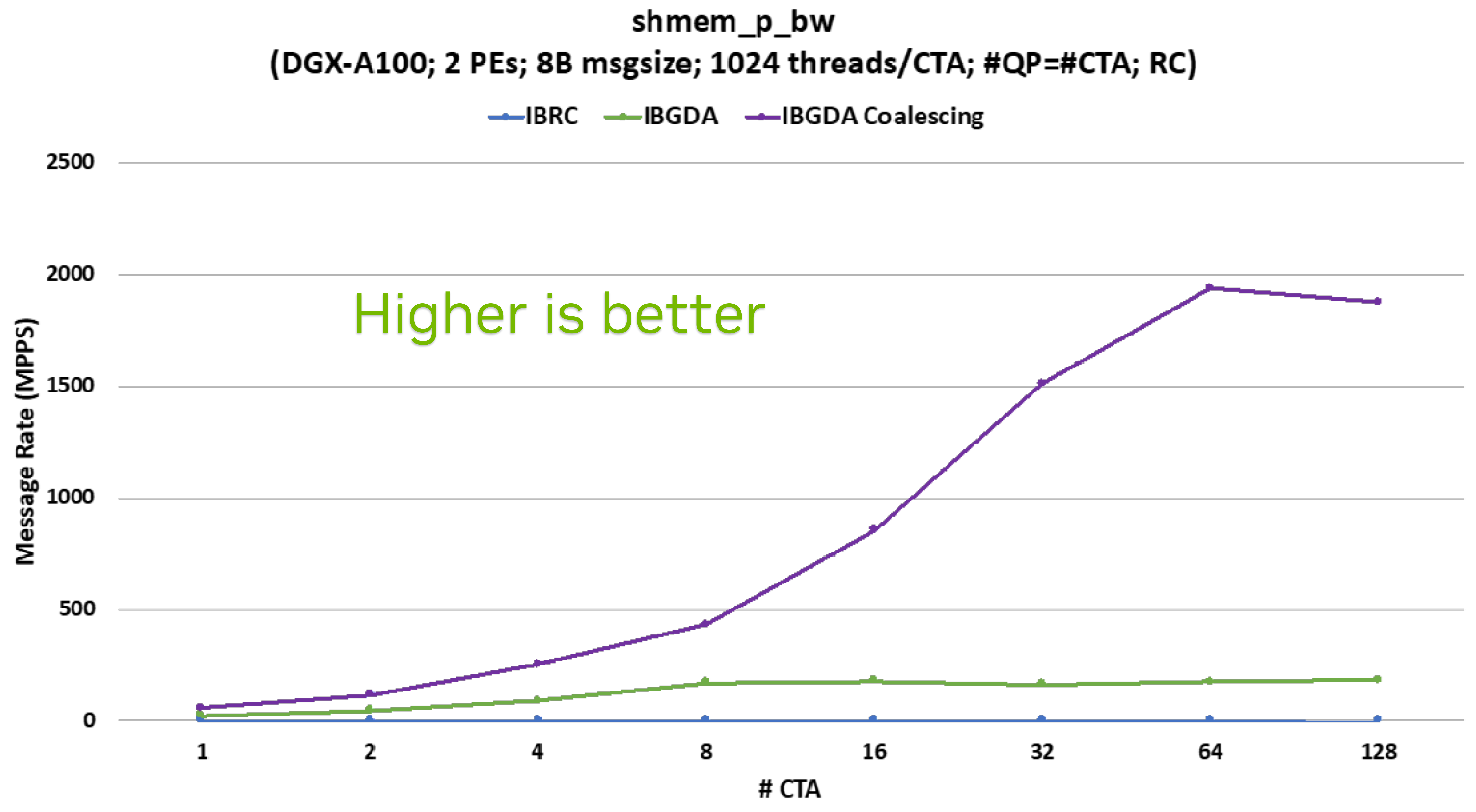
User kernel running on the SM:

1. Writes data into a buffer in memory. The user then calls an NVSHMEM routine like nvshmem putmem
2. Within this routine, NVSHMEM code in kernel writes a work request to the NIC's work queue
3. Writes a doorbell record (if lossy)
4. Rings doorbell on the NIC.
5. NIC processes the work request
6. Grabs the data
7. Performs the requested communication
8. Writes a completion event to the completion queue (CQ) that can later be processed by NVSHMEM

GPUDirect Async Kernel-Initiated Networking bandwidth gains

100x higher put bandwidth from GPU-initiated, then 10x more from GPU coalescing

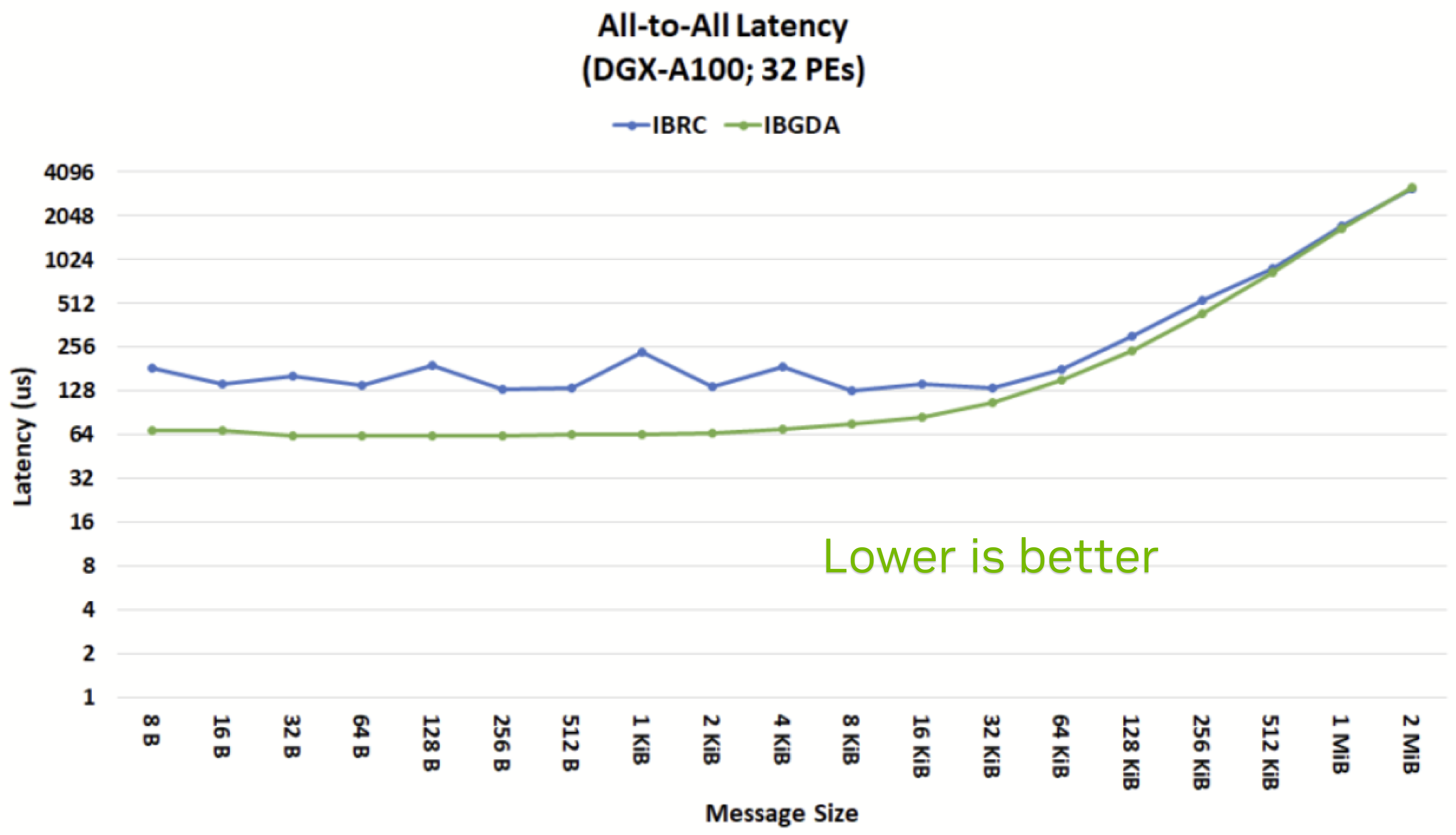
- IBRC = GPU calls back to CPU to initiate on its behalf
  - Use of GPU reduces sync
- IBGDA = GPU-initiated
  - Saturates NIC @ 180MOps/s
  - 100x speedup vs. IBRC's 1.7 MIOPs
- Coalescing of scalar put operations targeting adjacent memory locations provides a 10x boost in perf
  - GPU naturally coalesces
  - Messages are spent more effectively
  - More bandwidth



GPUDirect Async Kernel-Initiated Networking latency benefits

2+x lower All2All latency from GPU-initiated

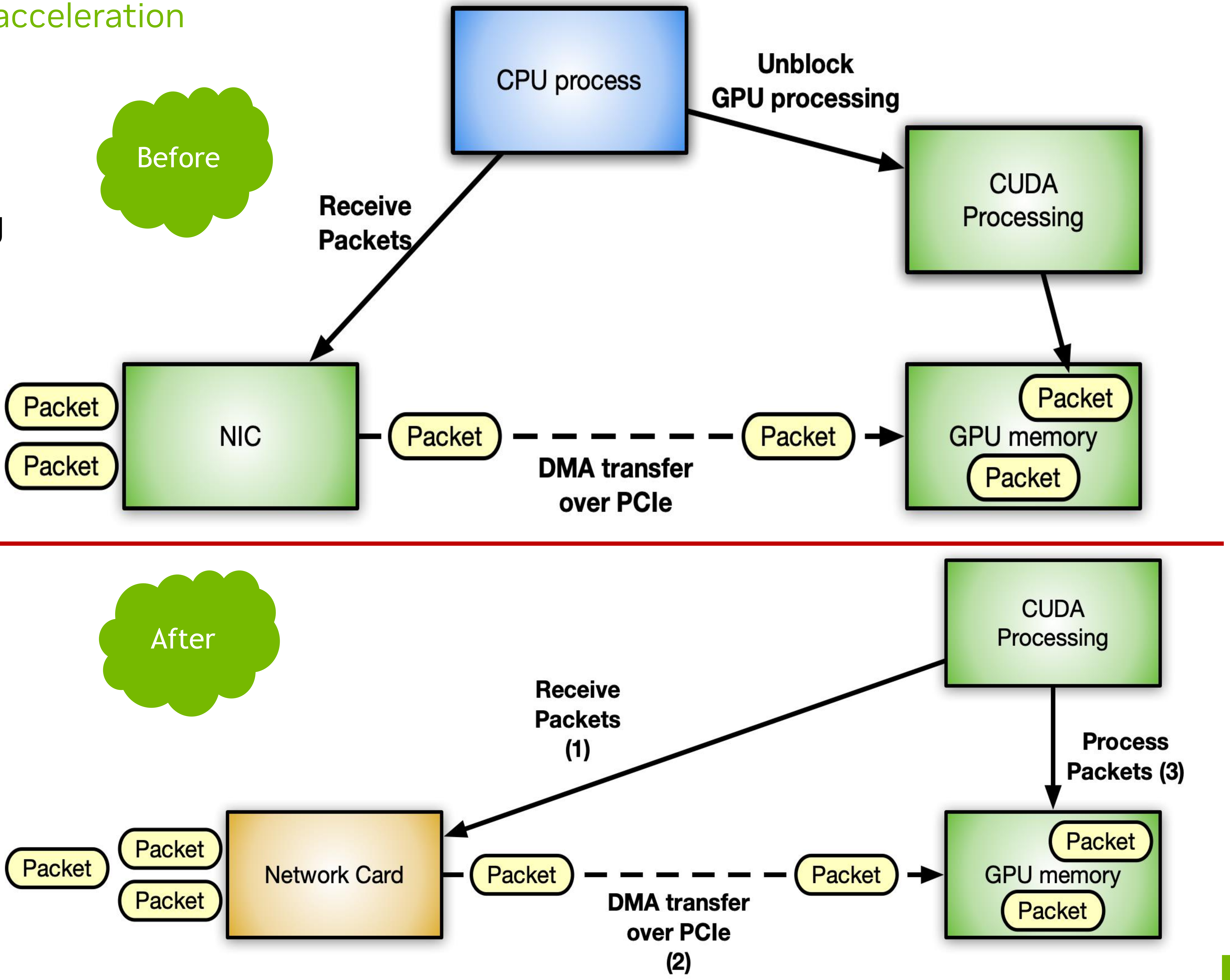
- IBRC = GPU calls back to CPU to initiate on its behalf
  - CPU thread is a serialization point
  - Proxy thread processes in batches; missing a batch leads to variation in delays
- IBGDA = GPU-initiated
  - 2^{+}x lower latency
  - Smooth and stable performance



DOCA GPUNetIO

GPU-initiated, NIC-assisted packet processing acceleration

- Real-time GPU packet processing, e.g. for sensor-based systems, 5G components
- New IO protocol between GPU and NIC/DPU
  - Expose NIC registers to the GPU
- Removes the CPU from the data path
  - Sends and receives from the GPU
  - Packets land directly into the GPU
- Minimizes latency



Call to action

- Remove CPU bottlenecks to boost bandwidth and reduce latency
- GPU-initiated storage: make the most of GPU pins bandwidth
    - Engage with us to share your usage models and feedback
- Try out GPU-initiated networking
    - NVSHMEM
    - GPUNetIO

Safe travels home! Until next year...

Comparison with UVM

More capacity, better perf on misses, better for fine-grained sparse access

SCADA vs. UVM:

- Higher-level abstraction vs. load/store model
    - No application change for UVM
    - Application change required for SCADA, but that's needed anyway to extend to remote memory or storage
- Capacity: Enables accessible memory >> CPU memory capacity
- Miss handling: rate of accesses >> rate of repeated page faults
    - Example: dependent accesses
    - High arithmetic intensity will reduce reference rate → miss rate
    - Prefetching is more beneficial for UVM than SCADA
    - UVM page fault can halt the whole or large subset of GPU
- Granularity of locality: can be fine-grained
    - Request packing for fine-grained accesses can lead to better efficiency than page granularity
    - Software-defined cache can be tailored to each application if there's locality