The University of Texas at Austin
Electrical and Computer Engineering

PhD Defense

Optimizing Communication for Clusters of GPUs

Michael LeBeane
mlebeane@utexas.edu

Advisor: *Lizy K. John*

GPUs and Networks in the Wild

- **GPUs are everywhere in HPC, Big Data, Machine Learning, and beyond**
  - Excellent performance/watt for many classes of data-parallel computation
- **Many GPUs are required to solve the biggest computational problems**
  - Can only fit so many GPUs in a single node!
  - GPUs need to talk to each other through Network Interface Controllers (NICs)
  - Path between GPU and NIC needs to be ***efficient***
- **Vendor's are selling machines filled with many GPUs and NICs:**

Nvidia's DGX-2

16 Tesla V100
8 Mellanox 100G NICs
2 Ethernet NICs
2 Xeon Platinum
1.6:1 GPU/NIC Ratio

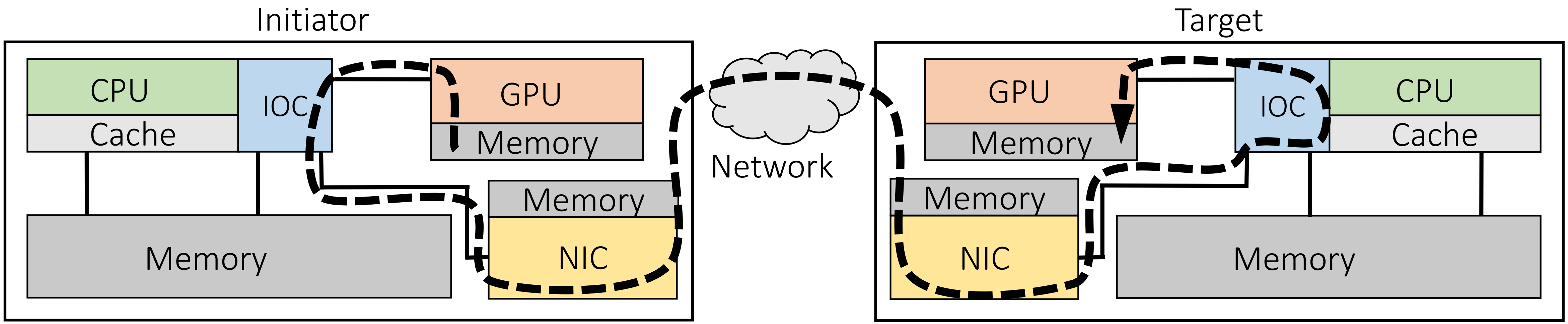
AMD's Project 47 Node

4 Radeon Instinct GPUs
2 Mellanox 100G NICs
1 EPYC 7601 32-Core CPU
2:1 GPU/NIC Ratio

IOC = IO Controller

Today’s GPU Networks

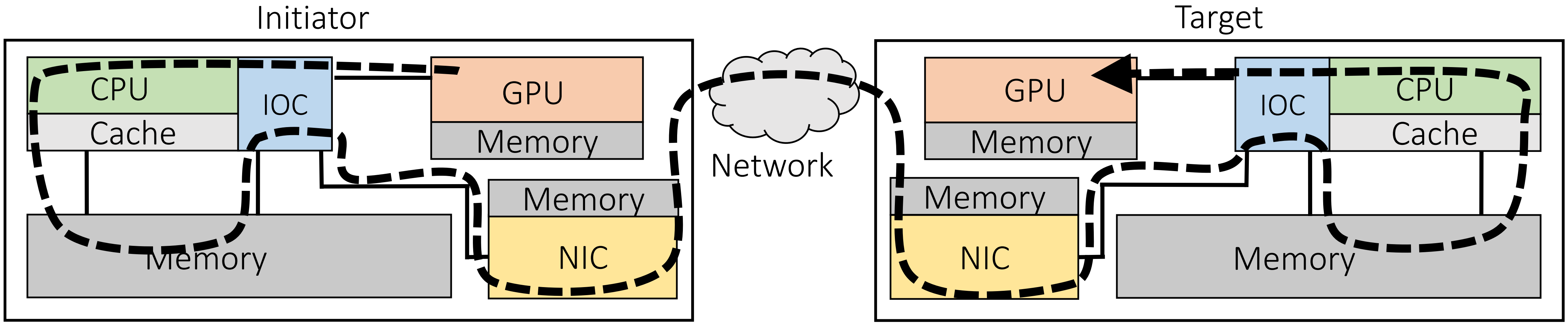
- **Largely focused on an optimized *data plane***
  - Path taken by the application data that needs to be transferred by the network
  - Industry technologies such as ROCn RDMA and GPUDirect RDMA allow peer-to-peer data transfers



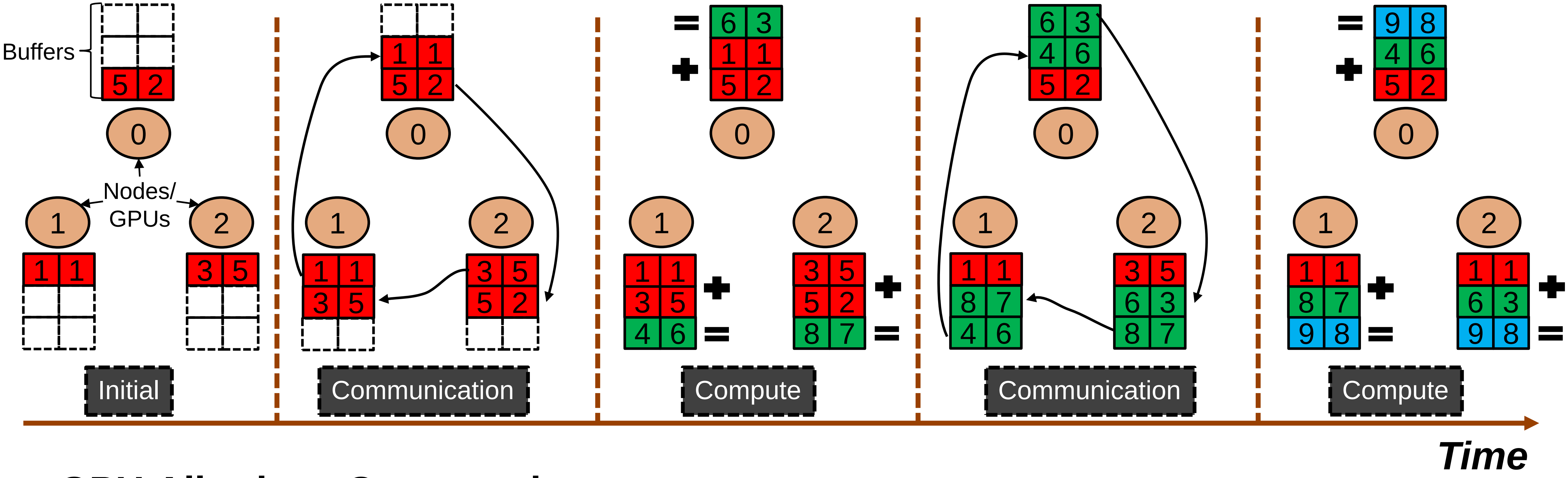
IOC = IO Controller

Challenges with Today’s GPU Networks

- ***Control plane*** **is unoptimized!**
  - Focused on a host-centric model where only the CPU can coordinate network transfers
  - Very high latencies to perform networking from the GPU



Motivating Example for Control Plane Optimizations



- **GPU Allreduce Computation**
  - Many communication/computation phases
  - Scaling out increases the number phases

Thesis Statement

GPU networking can be improved by both software and hardware enhancements that enable GPUs to more directly interface with the network control plane.

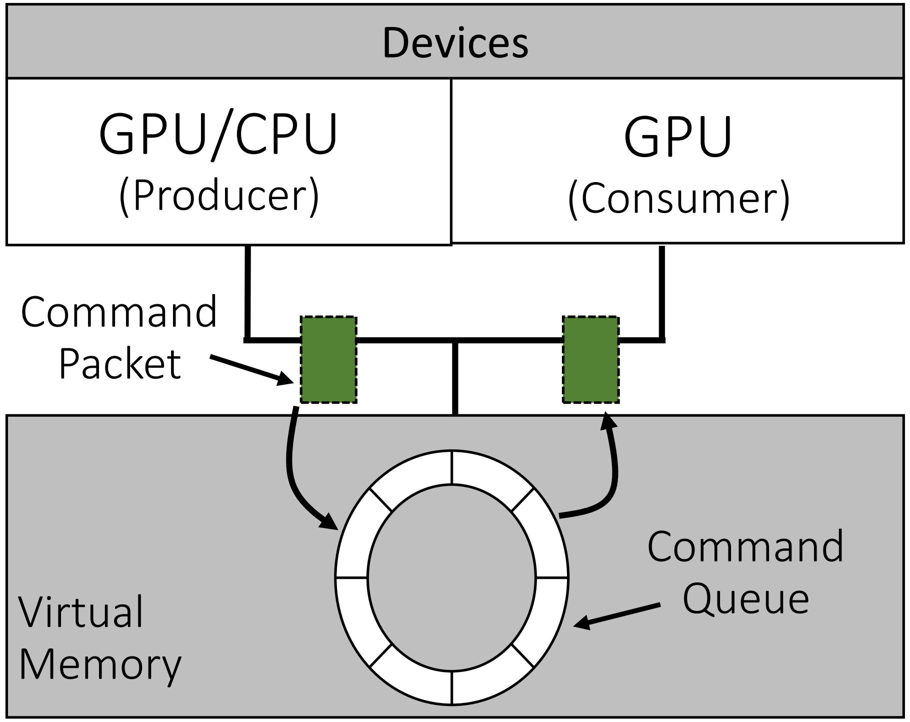
- **Proposed Solutions**
  - **Extended Task Queuing**
    - Direct NIC-to-GPU active messaging
  - **Command Processor Networking**
    - Dynamic communication using on-chip GPU Command Processor
  - **GPU Triggered Networking**
    - Initiate messages without critical path CPU

Outline

- Introduction
- **Contribution 1: Extended Task Queuing**
- Contribution 2: Command Processor Networking
- Contribution 3: GPU Triggered Networking
- Conclusion

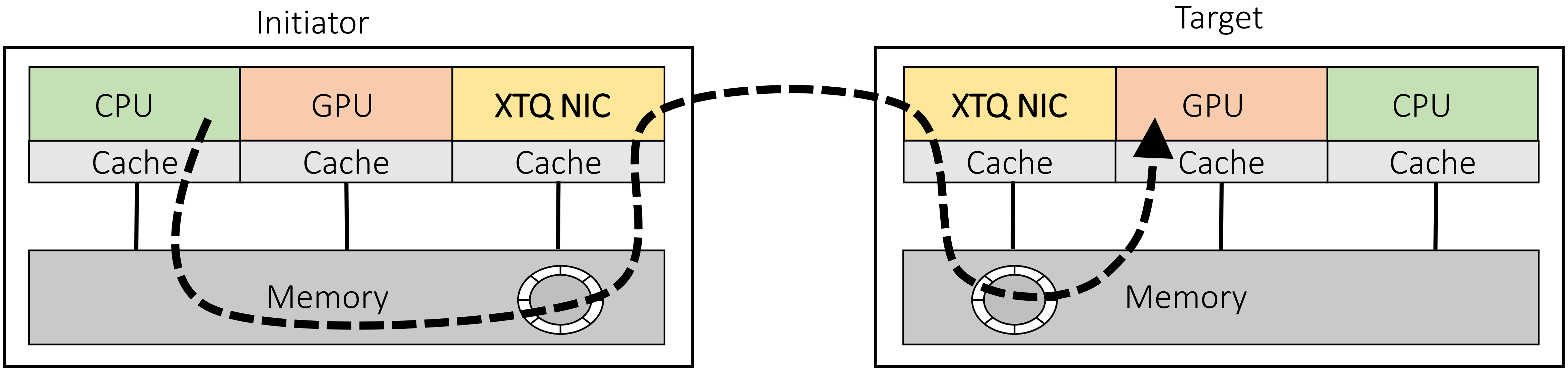
Local GPU Work Dispatch

- **GPUs consume work through in-memory command queues**
  - Queue format standardized through Heterogeneous System Architecture (HSA)
  - Any device can produce work for another device
  - Assumes unified virtual address space
- **Can we extend this across a node?**
  - NIC doesn't know how to talk to HSA queues
  - Initiator doesn't know the virtual addresses of resources at the target



Extended Task Queuing (XTQ) Overview

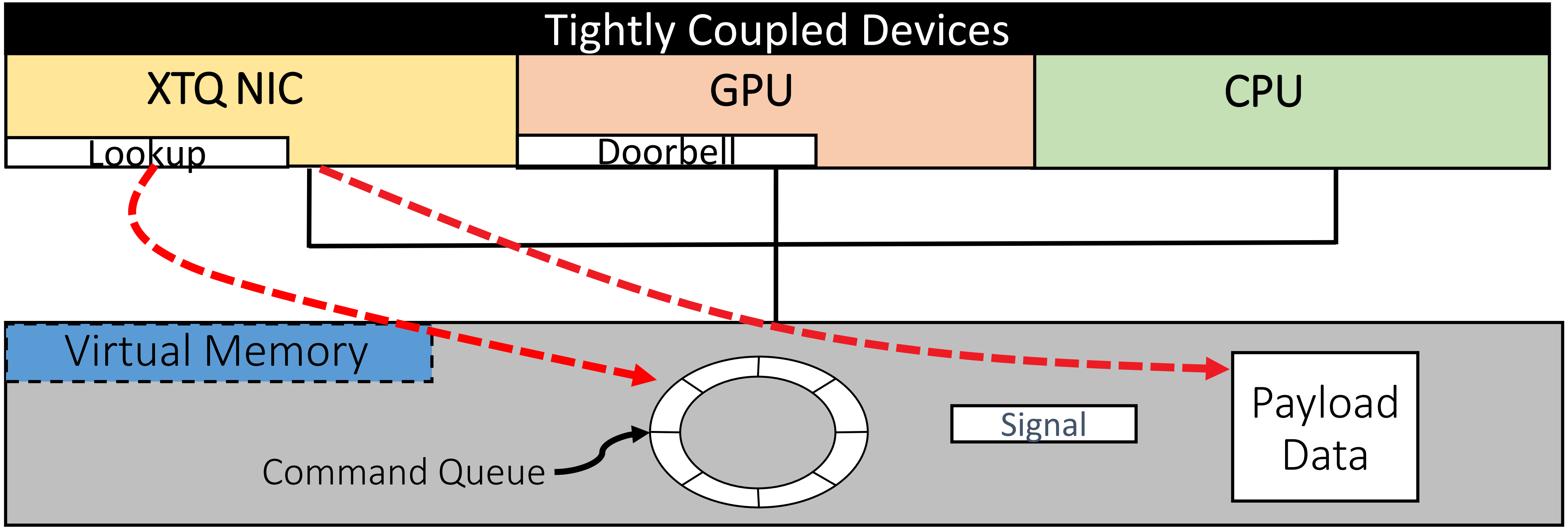
- **XTQ allows direct access to remote GPU queues**
  - Teach NICs how to speak with HSA queues
- **Enables *Active Messaging* without target CPU involvement**
  - Improves latency and frees CPU service thread(s)



M. LeBeane, B. Potter, A. Pan, A. Dutu, V. Agarwala, W. Lee, D. Majeti, B. Ghimire, E. Van Tassell, S. Wasmundt, B. Benton, M. Breternitz, M. L. Chu, M. Thottethodi, L. K. John, and S. K. Reinhardt, \Extended task queuing: active messages for heterogeneous systems," in Proc. of the Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC), 2016.

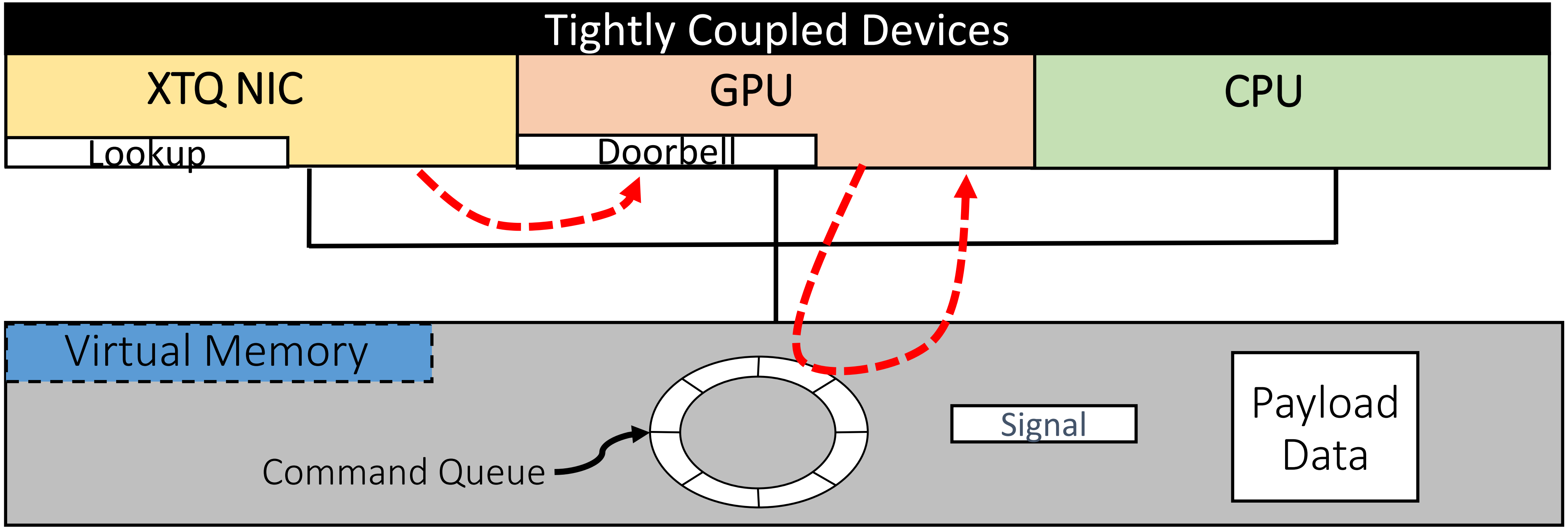
Target-side XTQ Operation

- **Payload data streams into target-side receive buffer**
- **Command descriptor is placed into command queue**



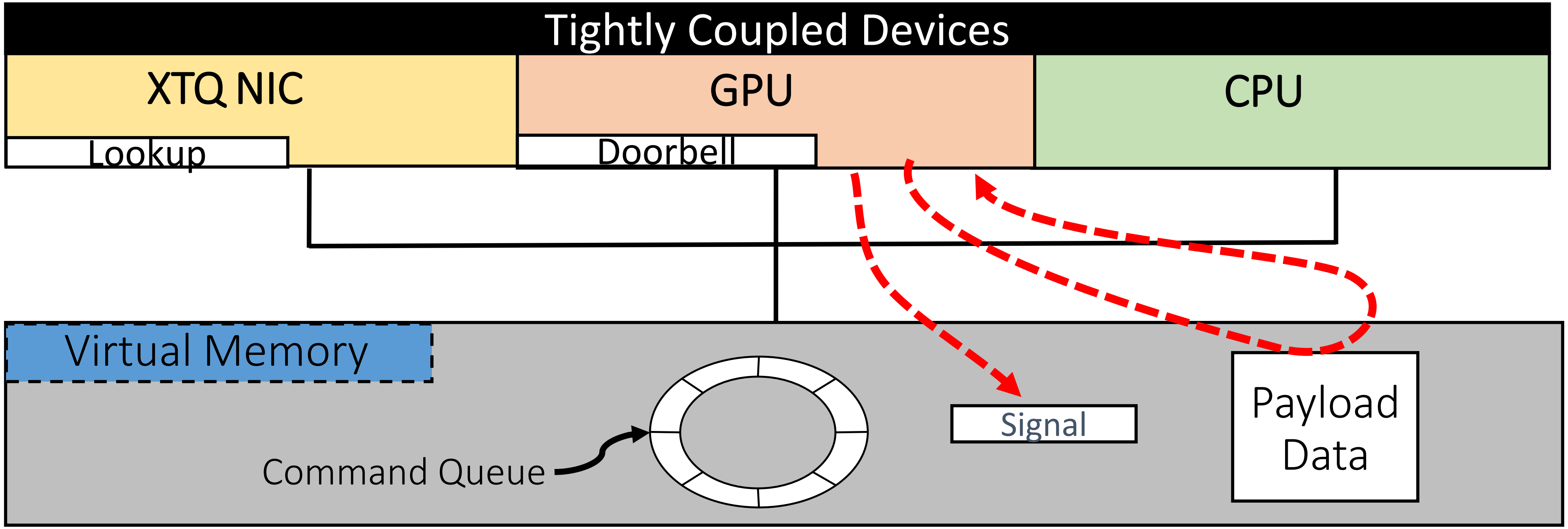
Target-side XTQ Operation

- **NIC notifies the GPU using memory-mapped doorbell**
- **GPU reads command packet**



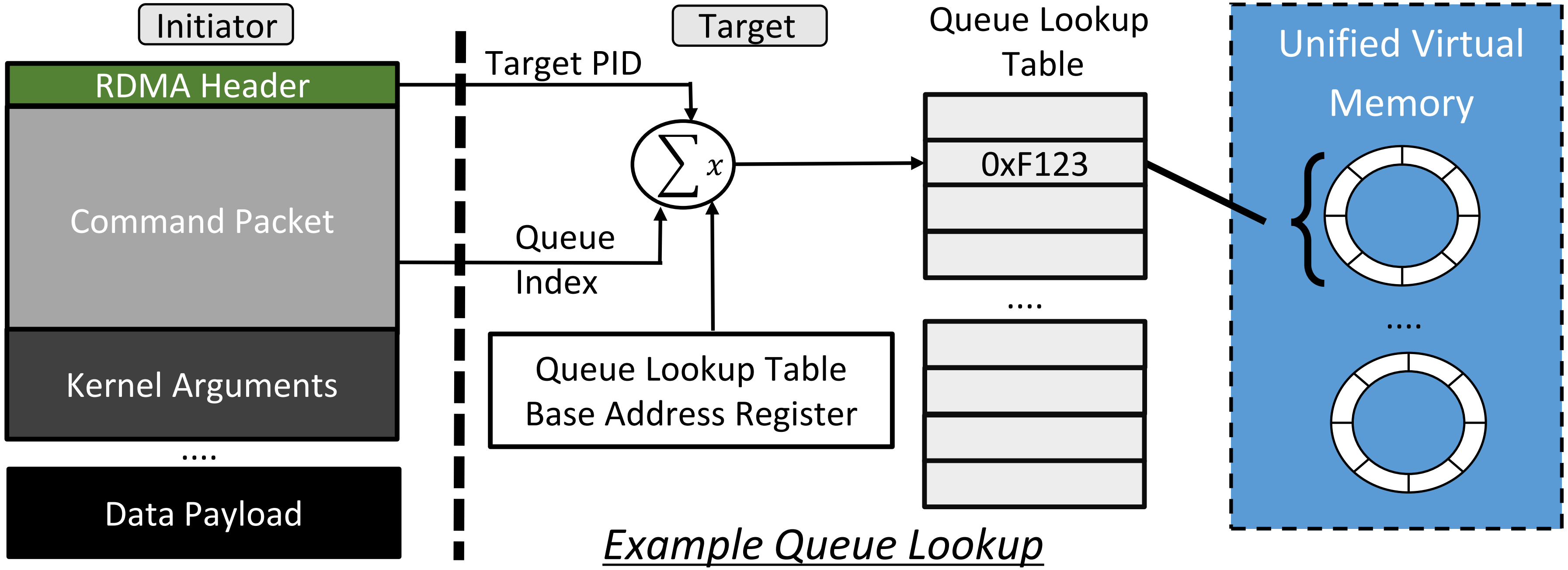
Target-side XTQ Operation

- **GPU reads transferred data**
- **GPU writes shared memory completion signal**



XTQ Coordinated Indices

- **How does initiator know about remote VAs at the target?**
- **Use *coordinated indices* specified by the initiator**
- **Lookup tables are populated by the target-side XTQ Library**

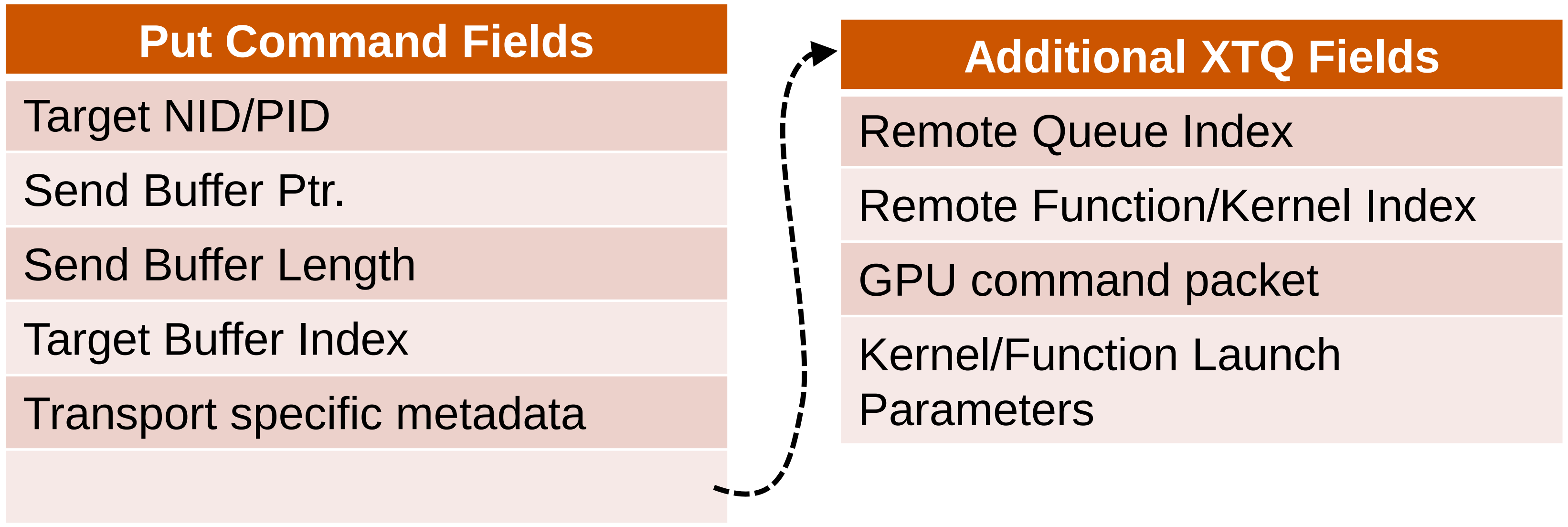


Example Queue Lookup

XTQ Runtime API

- **XTQ Put is implemented as a simple extension to standard RDMA put operation**
  - Compatible with many low-level RDMA transports (e.g. InfiniBand, RoCE, Portals 4, iWARP, etc.)
- **XTQ Registration API is used to provide address index-to-address translations**

Regular RDMA Put Operation | XTQ-Enhanced RDMA Put Operation | XTQ Rewrite Registration API



Put Command Fields
Target NID/PID
Send Buffer Ptr.
Send Buffer Length
Target Buffer Index
Transport specific metadata

Additional XTQ Fields
Remote Queue Index
Remote Function/Kernel Index
GPU command packet
Kernel/Function Launch Parameters

- Register Queue
  - Queue Desc. VA
- Register Function
  - Function Ptr. VA
  - Target Side Buffer VA
- Register Kernel
  - Kernel Ptr. VA
  - Target Side Buffer VA
  - Kernel Argument Size
  - Completion Signal VA

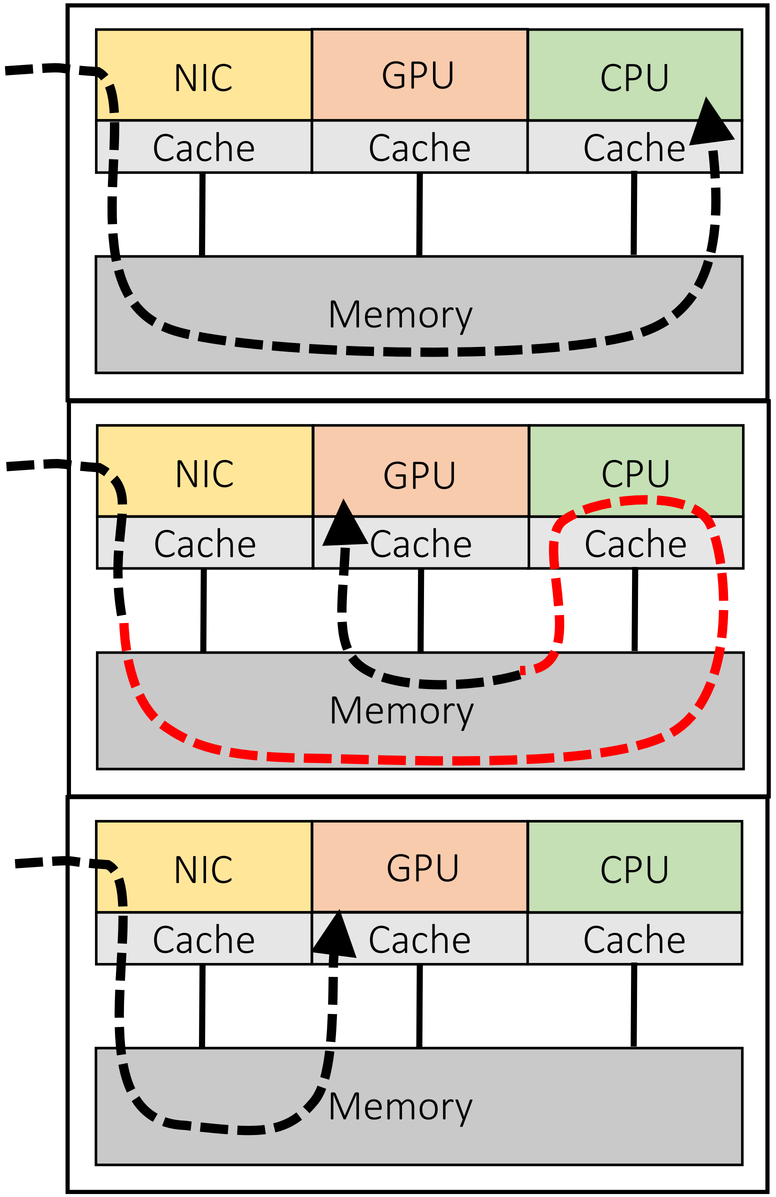
Experimental Setup

CPU and Memory Configuration	
Type	4-wide OOO, x86, 8 cores @ 4GHz
I,D-Cache	64KB, 2-way, 2 cycles
L2-Cache	2MB, 8-way, 8 cycles
L3-Cache	16MB, 16-way, 20 cycles
DRAM	DDR3, 8 Channels, 800MHz

GPU Configuration	
Type	AMD GCN3 @ 1GHz
CU Config	24 CUs with 4 SIMD-16 engines
Wavefronts	40 Waves per SIMD (64 lanes)
V-Cache	32KB, 16-way, 12 cycles, per CU
K-Cache	32KB, 8-way, 12 cycles, per 4 CU
I-Cache	64KB, 8-way, 12 cycles, per 4 CU
L2-Cache	1MB, 16-way, 8 banks, 100 cycles

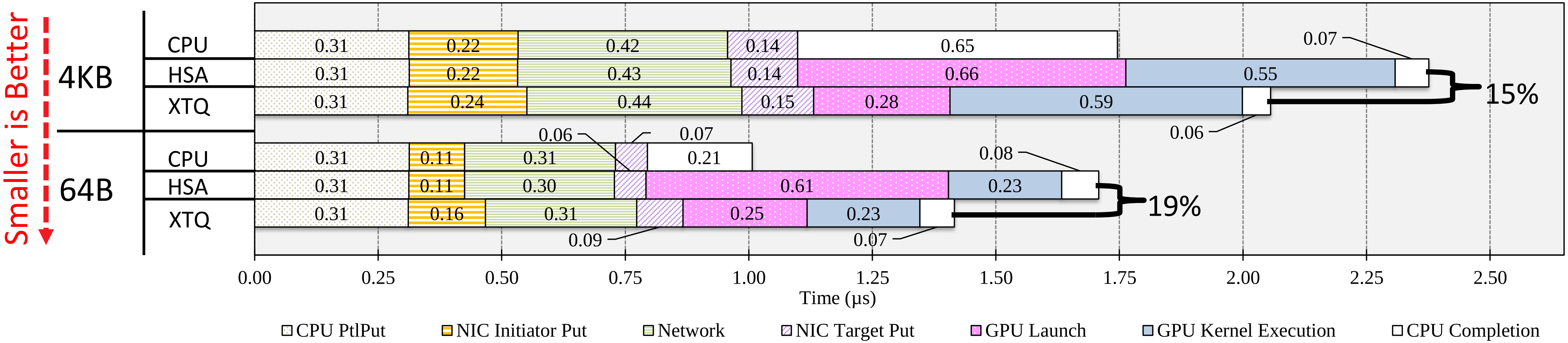
NIC Configuration	
Link Speed	100ns/ 100Gbps
Topology	Star

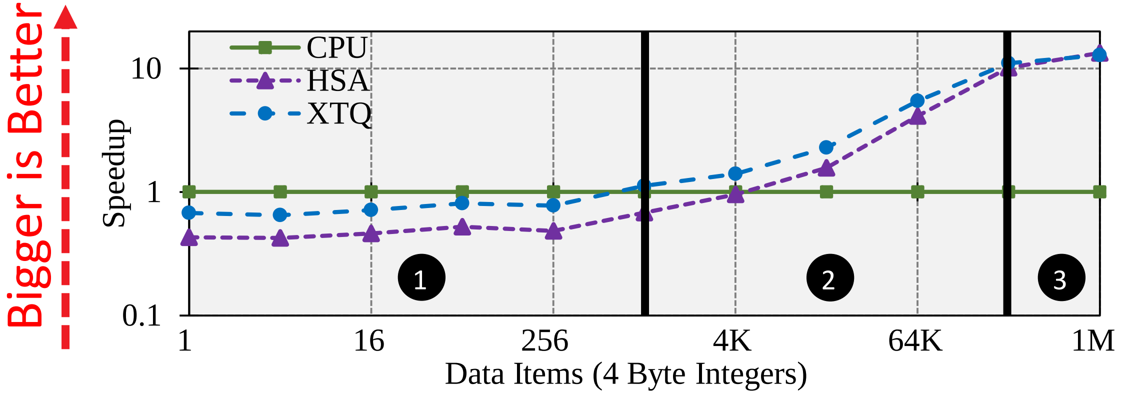
- **CPU: Standard CPU-only systems**
  - Baseline non-accelerated system
- **HSA: Currently available GPU systems**
  - Involves CPU runtime
- **XTQ: Extended Task Queuing**
  - Enables efficient active messaging style communication that bypasses the CPU on the target

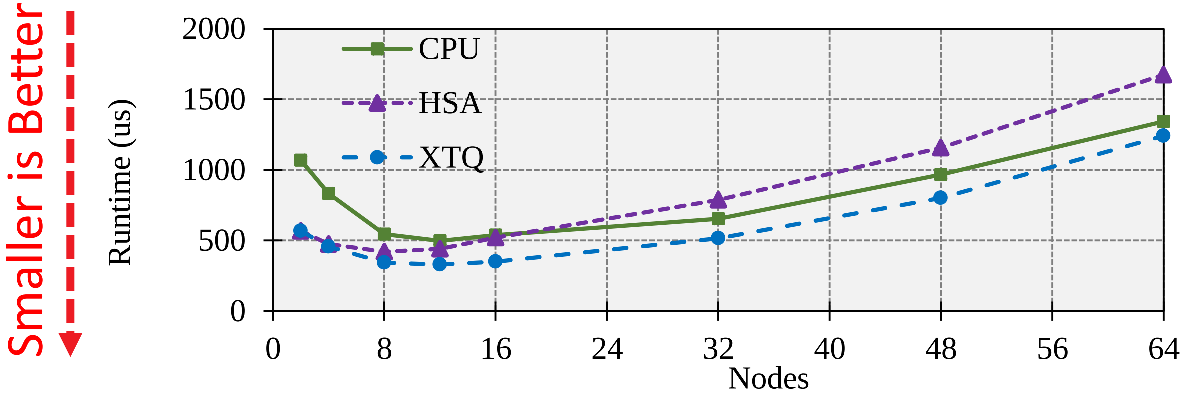


Results

Latency Decomposition



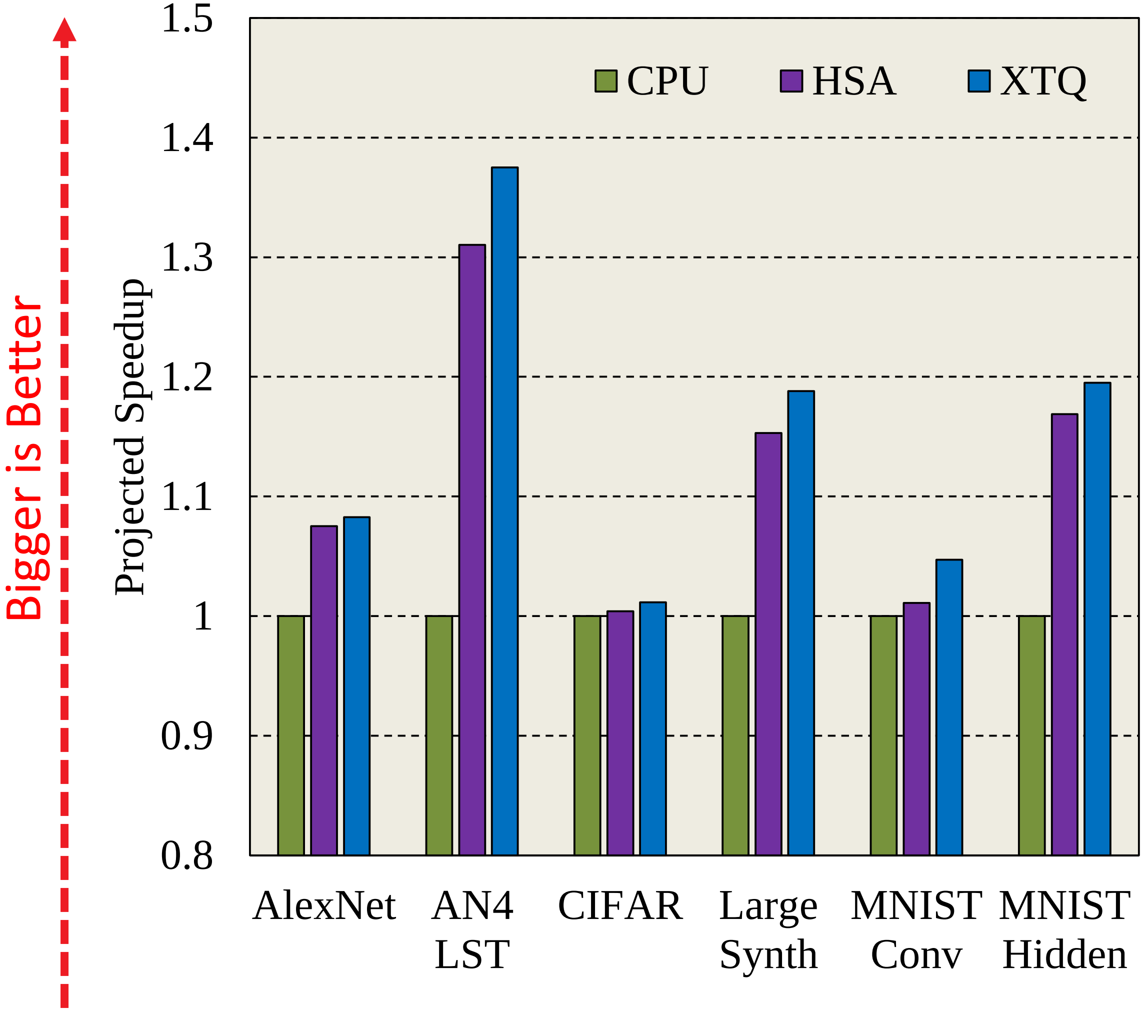




MPI Accumulate

MPI Allreduce

Results



Workload Name	Domain	%Blocked	Reductions
Alex Net	Classification	14%	4672
AN4 LSTM	Speech	50%	131192
CIFAR	Classification	4%	939820
Large Synth	Synthetic	28%	52800
MNIST Conv	Text Recognition	12%	900000
MNIST Hidden	Text Recognition	29%	900000

- **Use Microsoft’s Cognitive Toolkit and sample workloads**
- **Projected using simulation results + profiling data from TACC’s Stampede supercomputer**
- **Speedups bound by % time application blocked on network data**

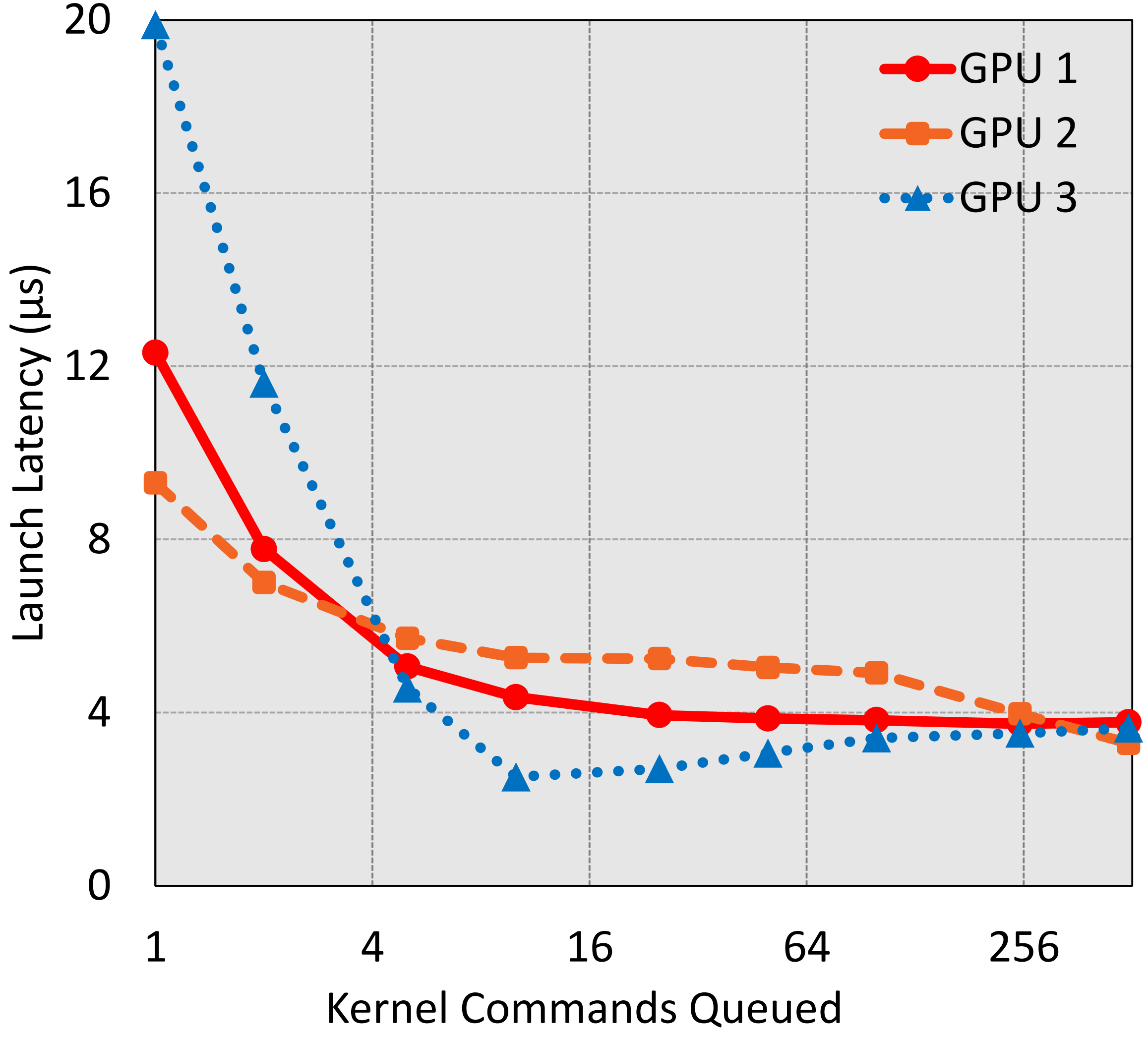
Outline

- Introduction
- Contribution 1: Extended Task Queuing
- **Contribution 2: Command Processor Networking**
- Contribution 3: GPU Triggered Networking
- Conclusion

Motivating Intra-kernel Networking

- **XTQ provides optimized remote kernel invocation**
  - But still at kernel boundaries
  - Kernel launches are expensive!
  - Best case ~3µs
    - Network latency is < 0.7µs……
- **Can we do better?**
  - Networking from within a kernel?
  - What have other researchers tried?

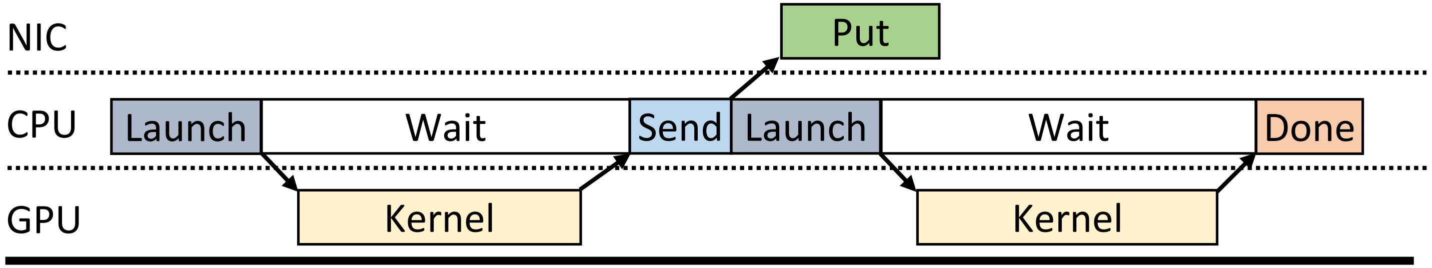
Smaller is Better



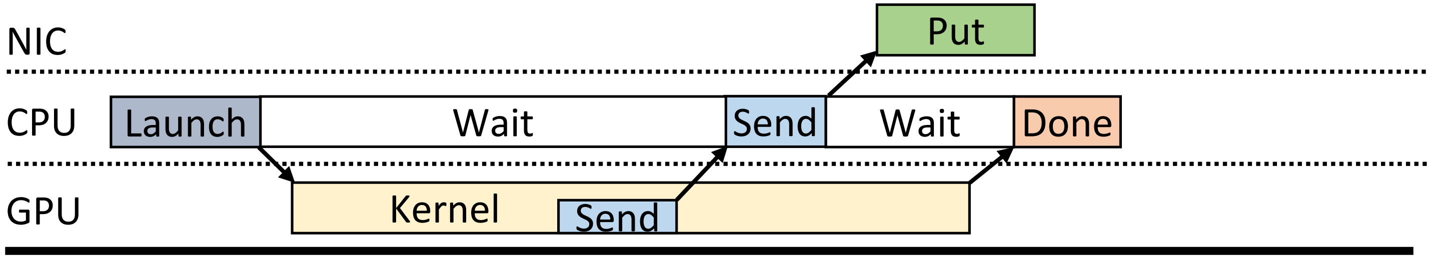
Prior-art in Intra-kernel Networking

- **GPU can send messages inside a kernel**
- **CPU thread is responsible for taking packets from GPU and poking NIC**
- **Will refer to this style of intra-kernel networking as *GPU Host Networking***

- **Host Driven Networking (e.g., MPI + CUDA)**



- **GPU Host Networking**



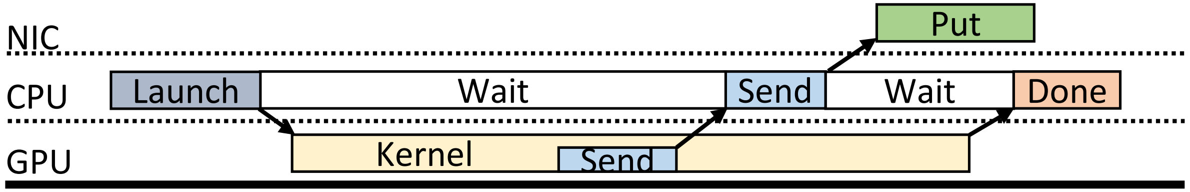
S. Kim, S. Huh, Y. Hu, X. Zhang, E. Witchel, A. Wated, and M. Silberstein, “GPUnet: Networking Abstractions for GPU Programs,” *In USENIX Conf. on Operating Systems Design and Implementation* (OSDI). 2014.

J. A. Stuart and J. D. Owens, “Message passing on data-parallel architectures,” *In Intl. Symp. on Parallel Distributed Processing* (IPDPS). 2009.

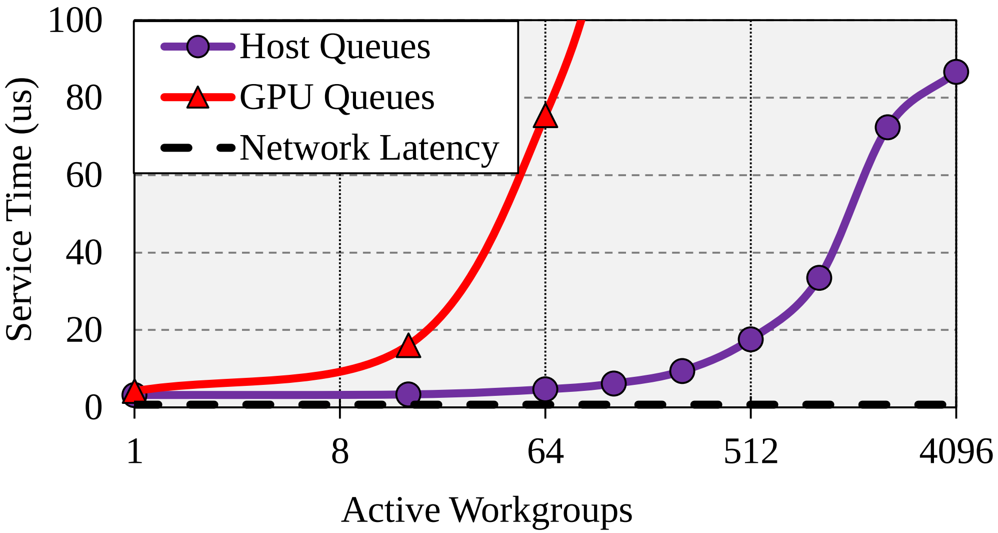
T. Gysi, J. Bär, and T. Hoefler., “dCUDA: hardware supported overlap of computation and communication,” In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (SC). 2016.

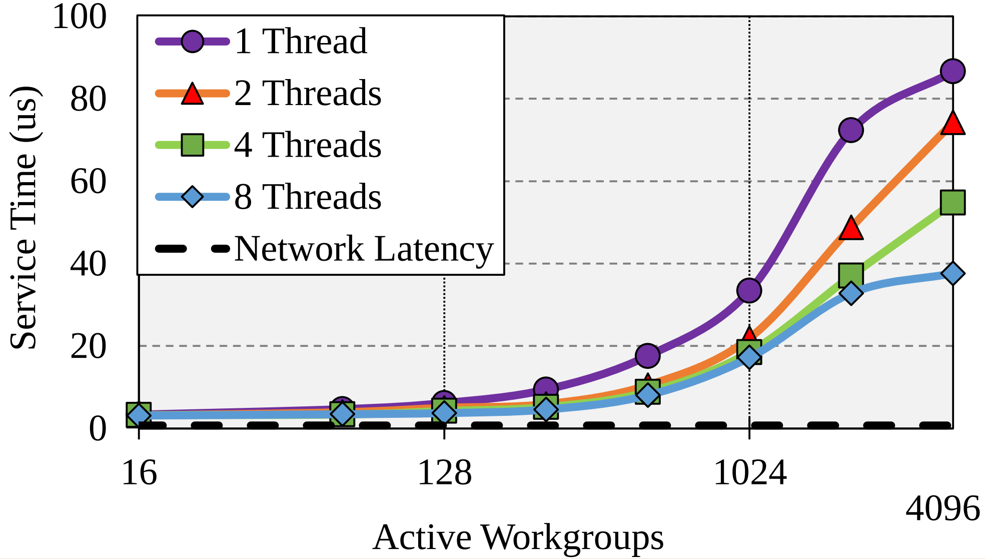
Performance Problems with GPU Host Networking

- **Need multiple trips over IO bus**
- **Where to place queues?**
  - GPU memory vs. host memory
  - High latency in both cases
- **Not scalable**
  - 4096 Work-groups fills the GPU
  - Still 40µs latency with 8 threads



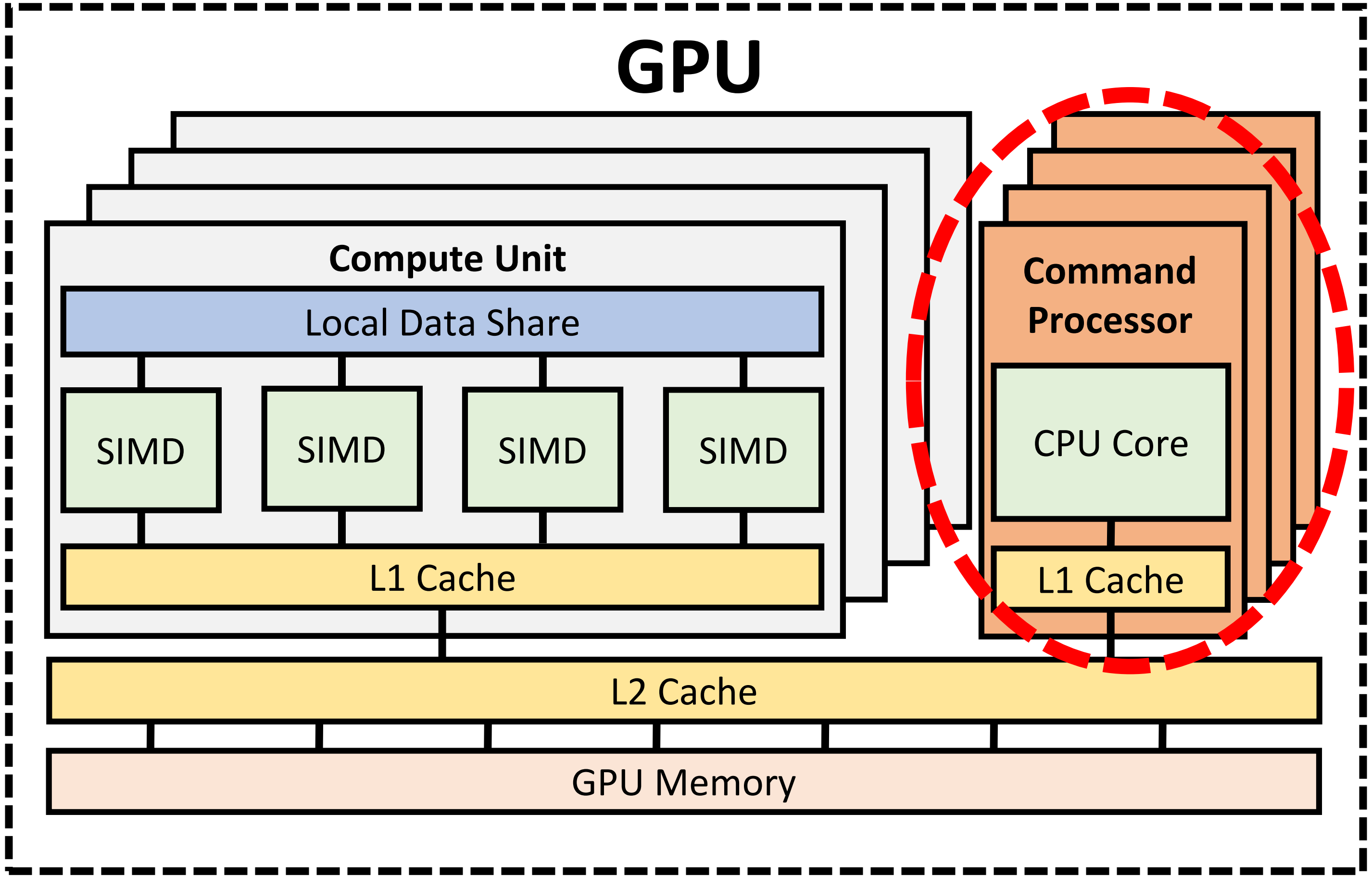
Smaller is Better





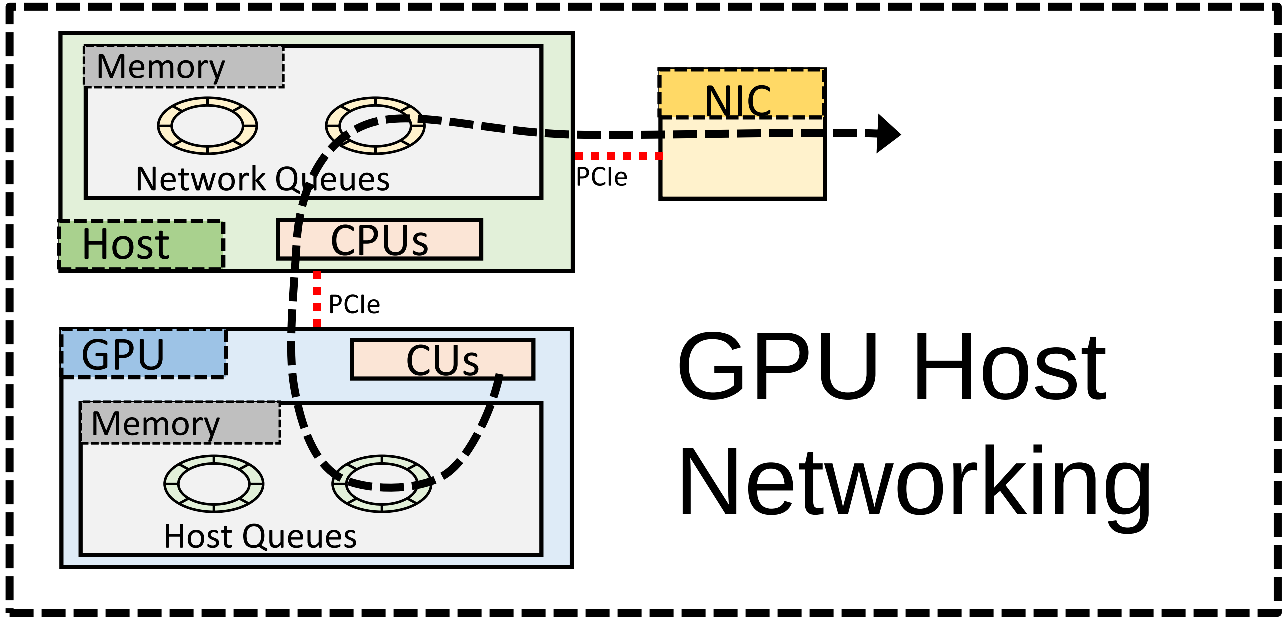
Command Processor Overview

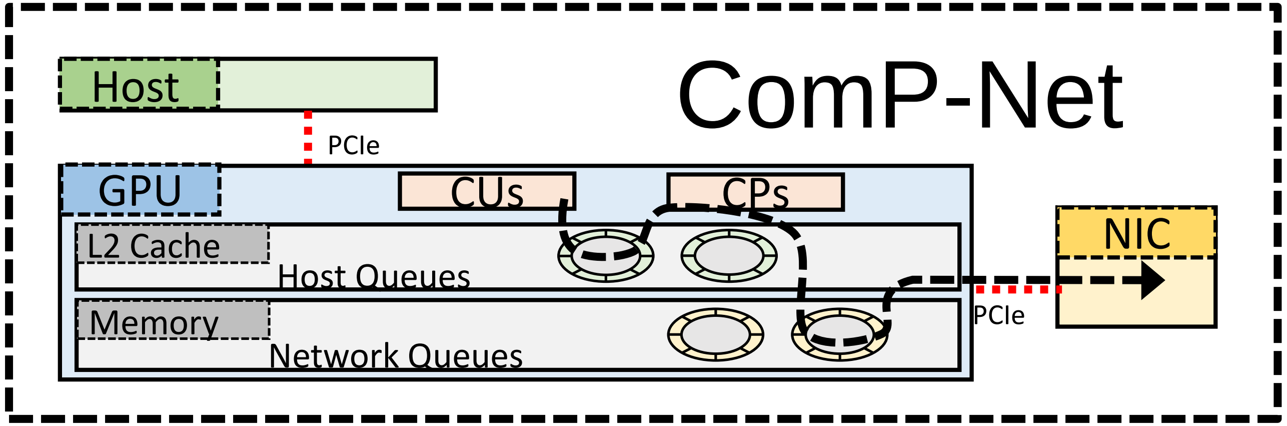
- **GPUs have built in CPUs called Command Processors (CPs)**
  - Scalar cores == good at running network runtime code
  - Connect to GPU CUs through a shared LLC
- **Traditionally used to launch kernels**
  - But intra-kernel networking encourages less kernels…..



Command Processor Networking (ComP-Net) Overview

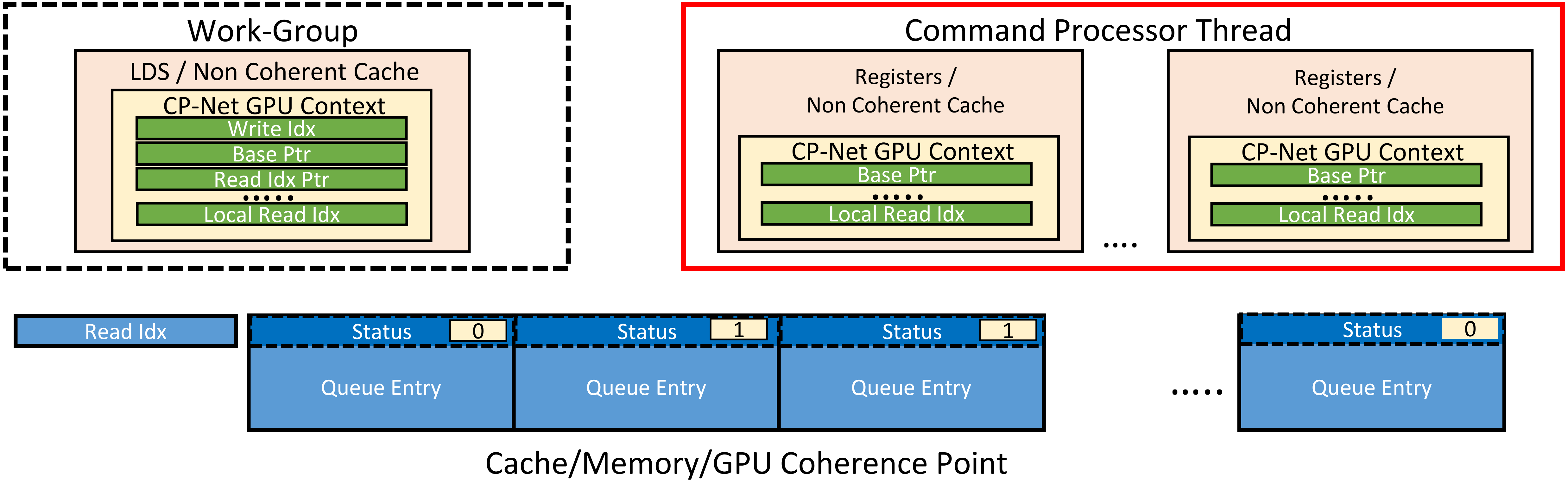
- **Uses built in CP to support network operations**
- **CP/GPU communicate over shared L2 cache instead of PCIe**
- **Potentially much faster (lower latency) than other GHN designs**
- **Scales naturally**
  - Every GPU has multiple CP threads





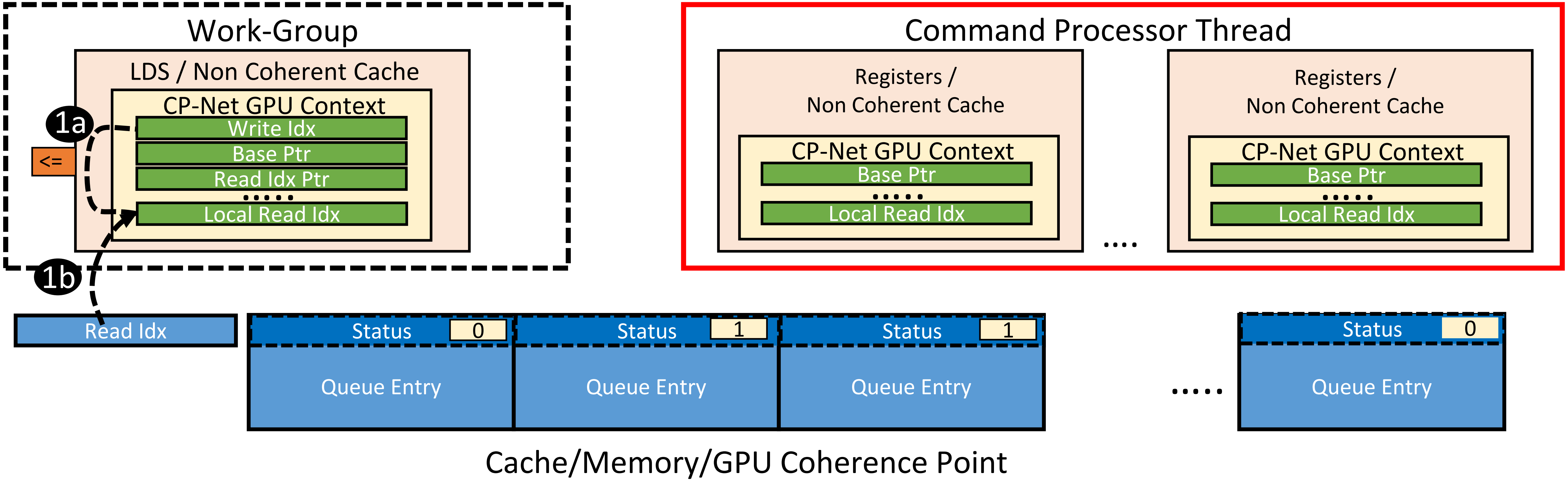
M. LeBeane, K. Hamidouche, B. Benton, M. Breternitz, S. K. Reinhardt, and L. K. John, “ComP-Net: Command Processor Networking for Efficient Intra-kernel Communications on GPUs," in Proc. of the Intl. Conf Parallel Architectures and Compilation Techniques (PACT), 2018.

ComP-Net Producer/Consumer Queue



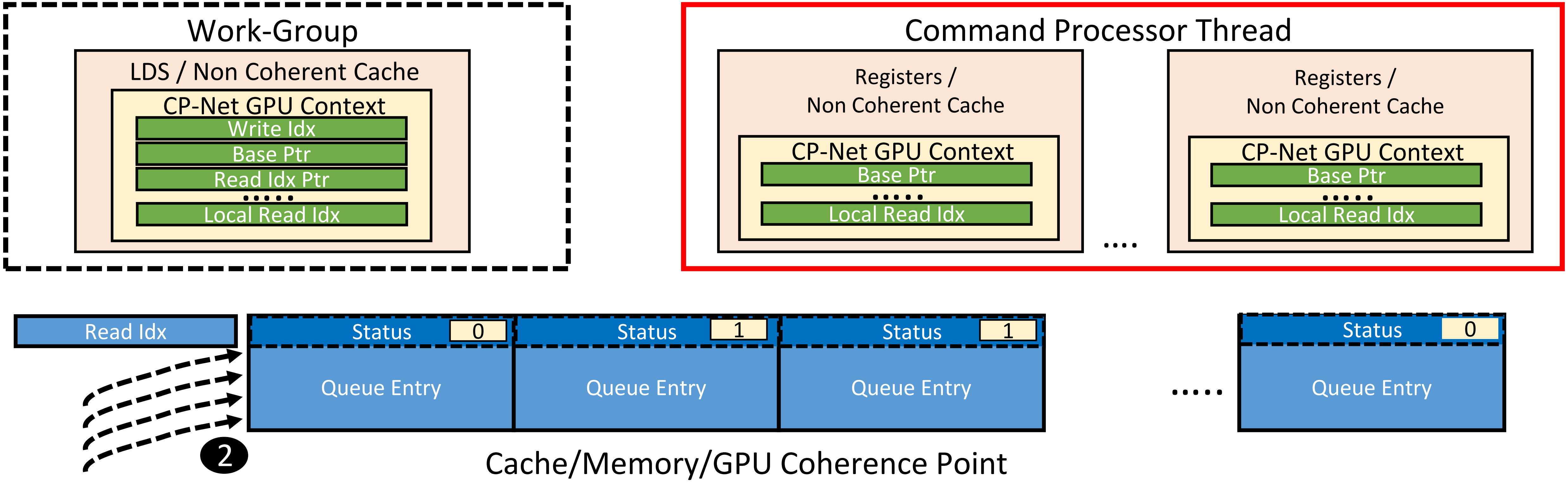
- **Main component of ComP-Net Runtime is CP/GPU producer/consumer queue**
- **Most steps are straightforward**

ComP-Net Producer/Consumer Queue



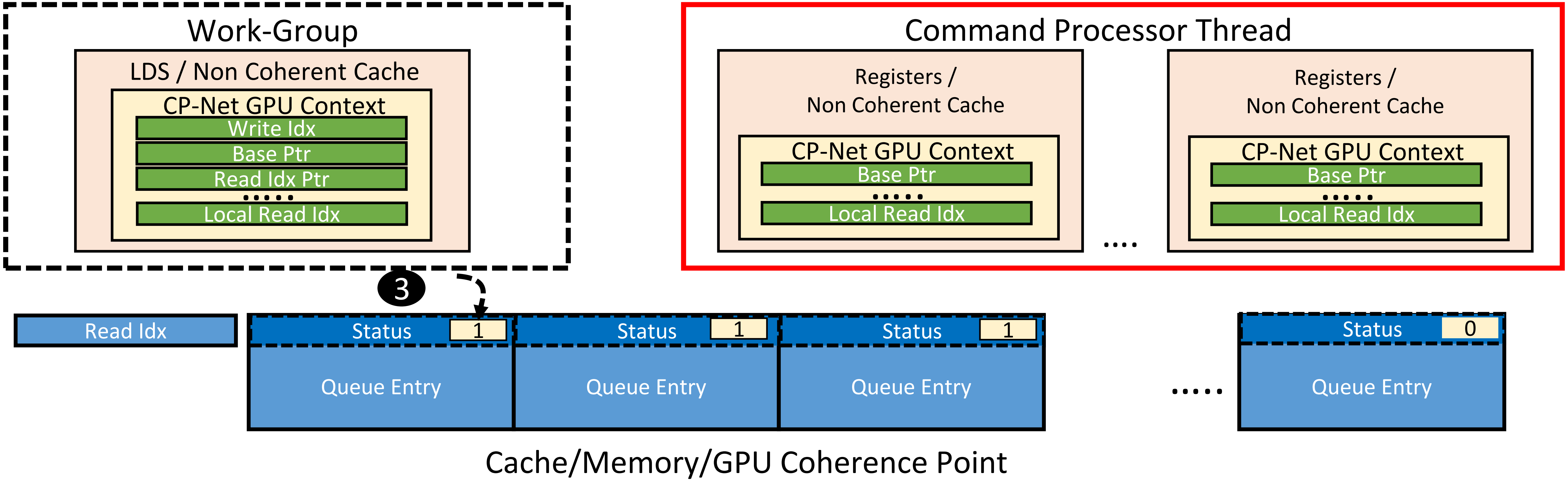
- **1a) Check if queue is full (using local Read Idx)**
- **1b) If full, update Read Idx and loop till not full**

ComP-Net Producer/Consumer Queue



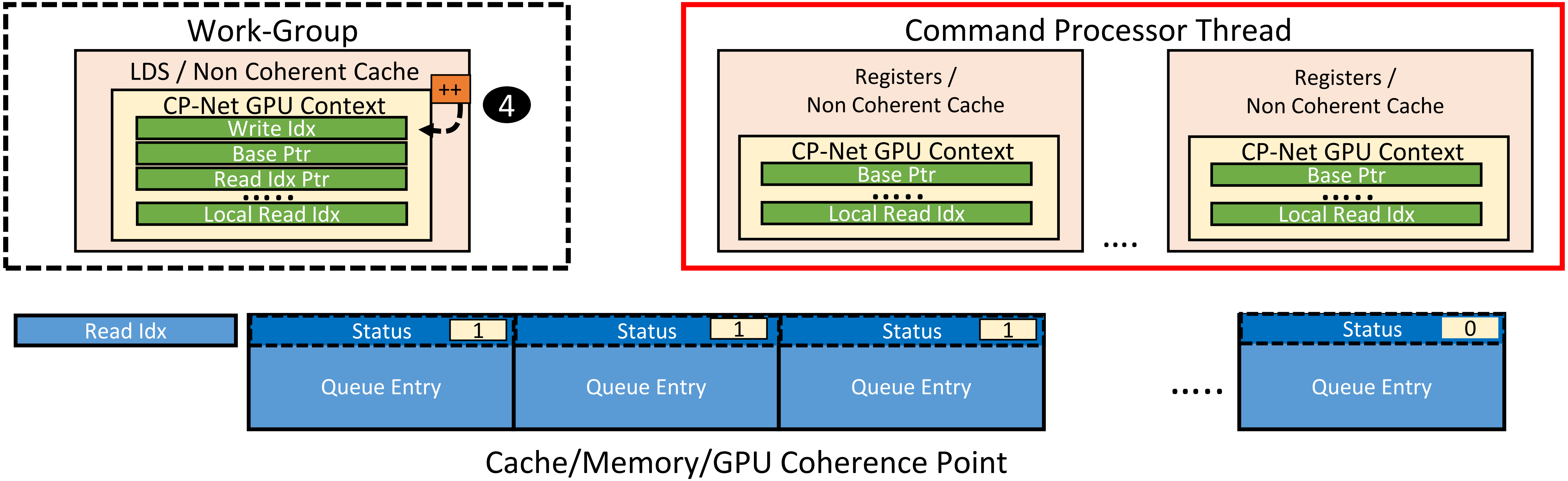
- **2) Fill Queue Entry with networking metadata**
  – Or Inline small payloads in the Queue Entry itself

ComP-Net Producer/Consumer Queue



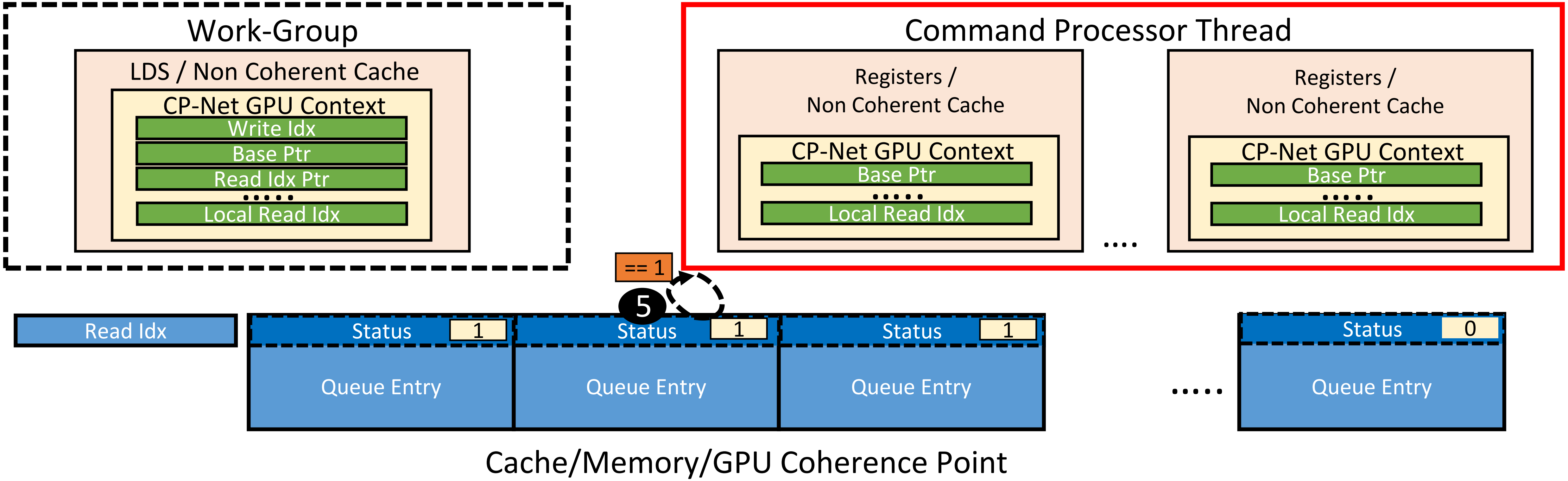
- **3) Set status flag with release marker to notify CP**

ComP-Net Producer/Consumer Queue



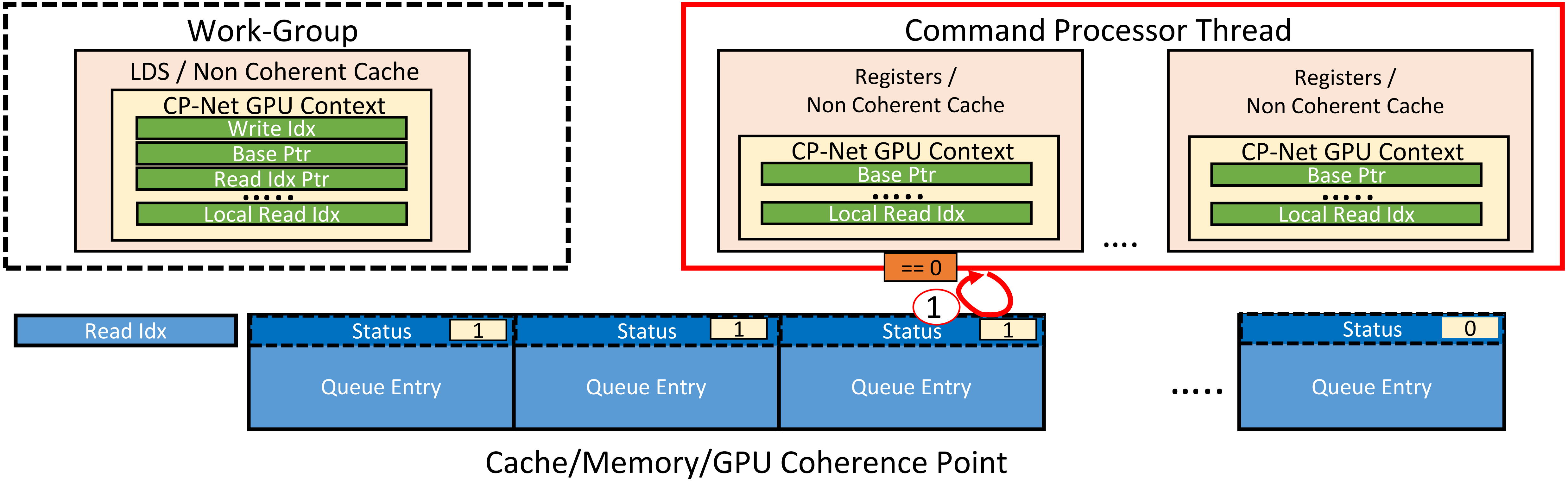
- **4) Increment local Write Idx**

ComP-Net Producer/Consumer Queue



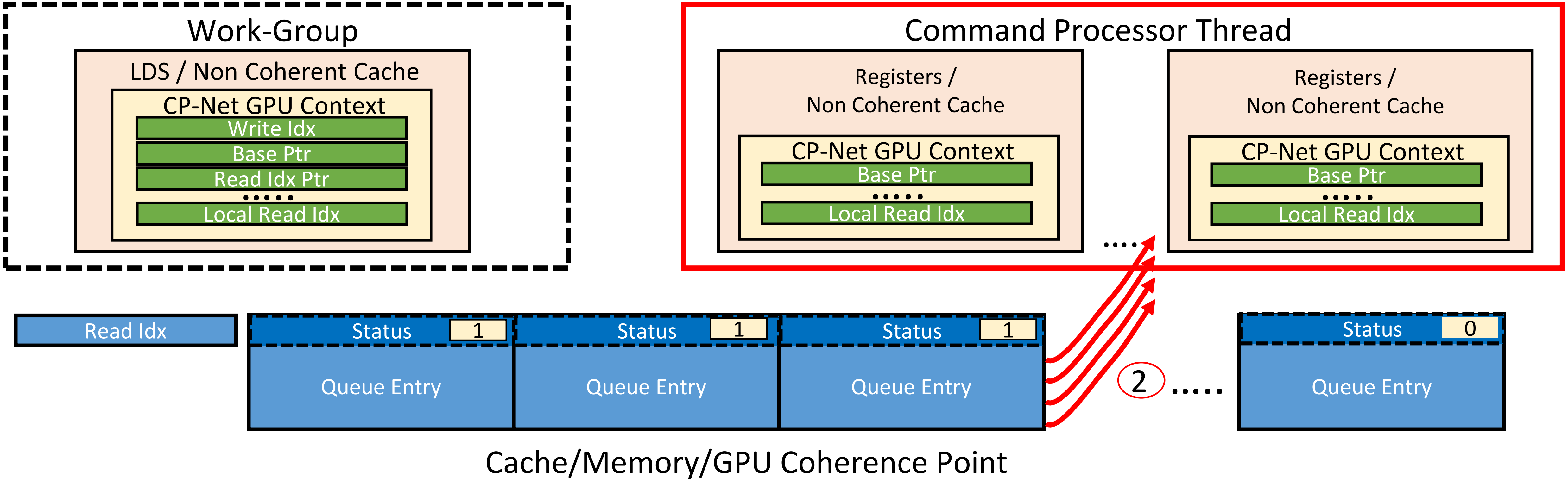
- **5) Check status bit to determine when CP completes operation**

ComP-Net Producer/Consumer Queue



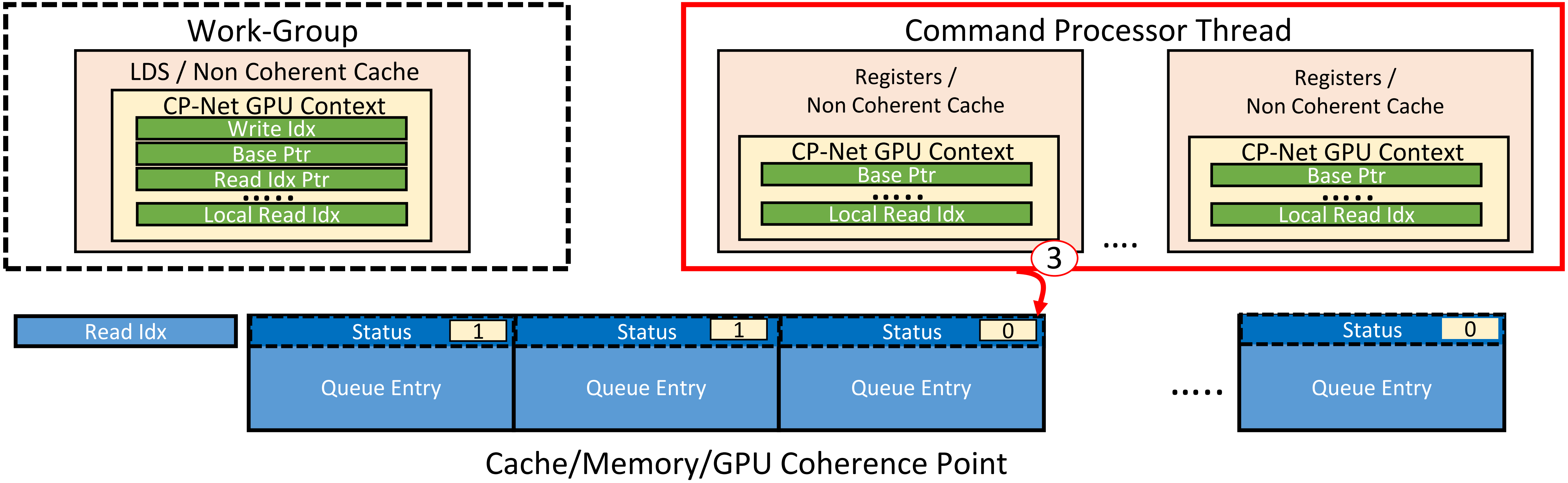
- **1) Poll on next Queue Entry based on local Read Idx with acquire marker**

ComP-Net Producer/Consumer Queue



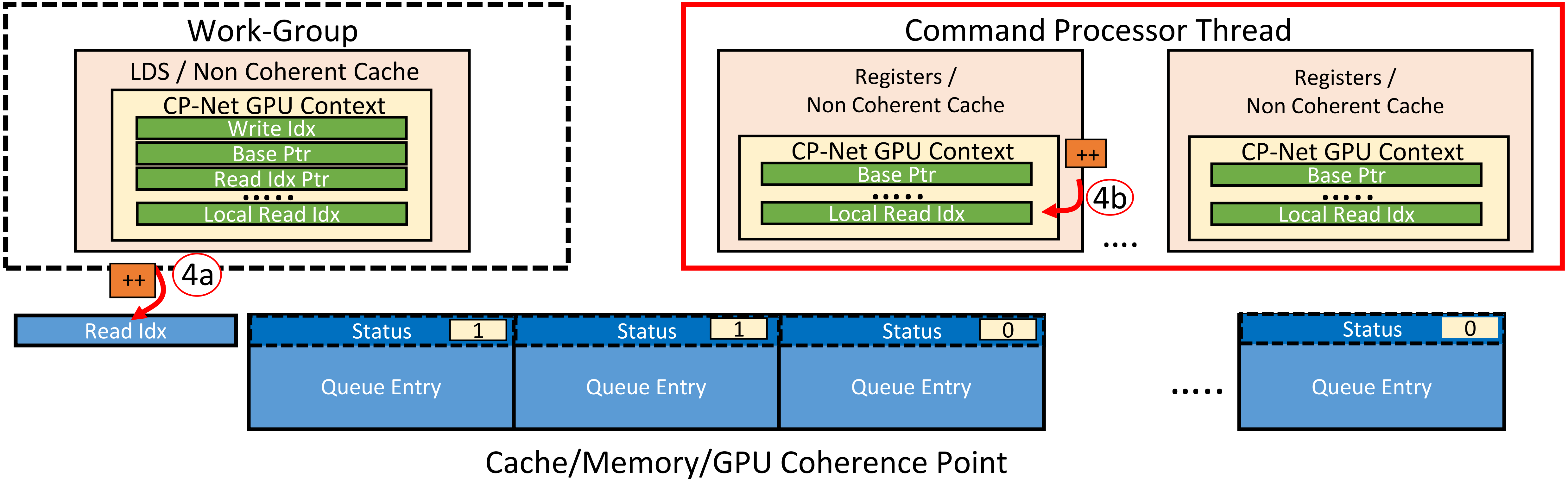
- **2) Read data from Queue Entry**

ComP-Net Producer/Consumer Queue



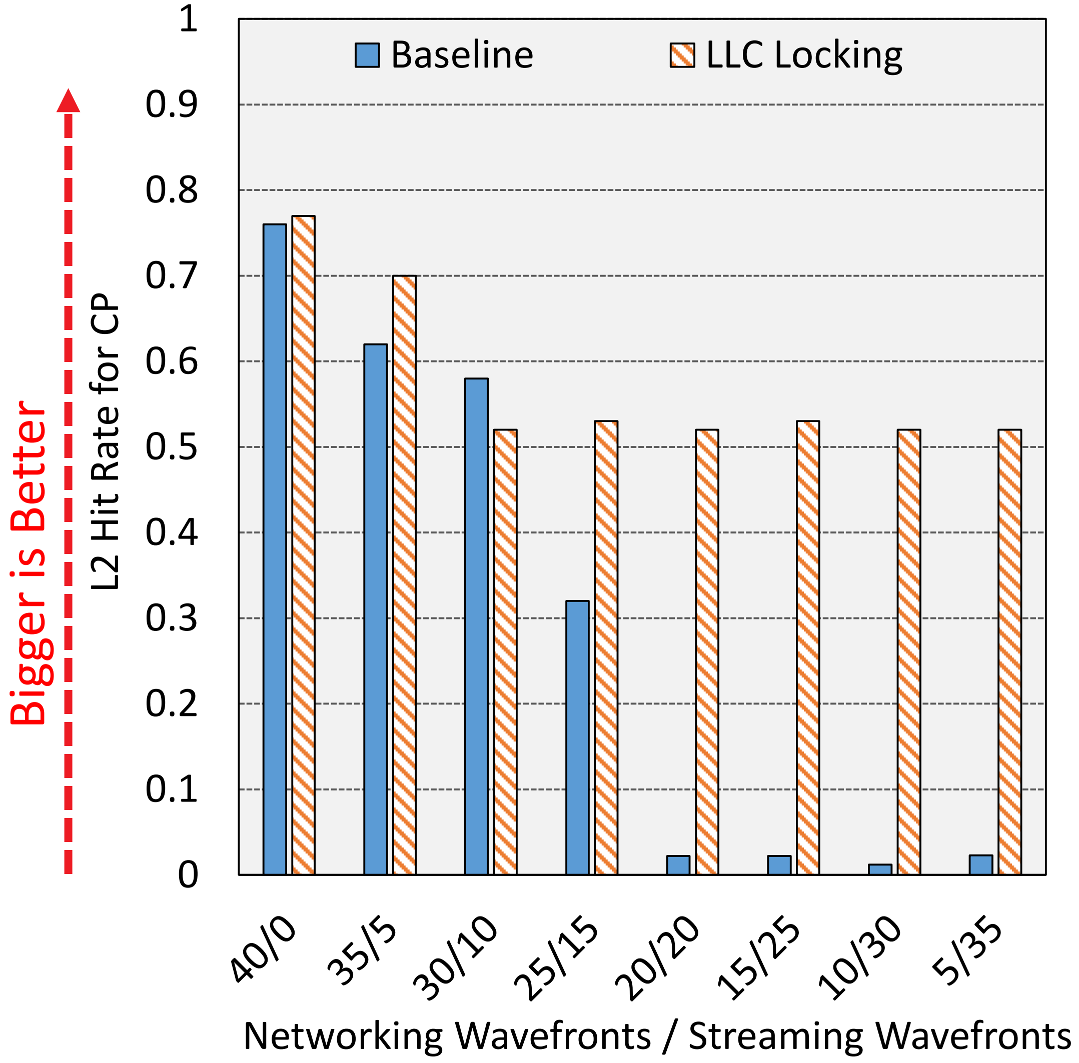
- **3) Perform Network operation and set Status flag to 0 when complete with release marker**

ComP-Net Producer/Consumer Queue



- **4a) Update global read Idx**
- **4b) Update local read Idx with release marker**

Tackling GPU Cache Thrashing



- **Residency of data in GPU L2 is very small**
- **Work-group data produced for CP is evicted when other work-groups are performing streaming memory accesses**
- **Can be solved through cache line locking**
  - Preliminary results are promising
  - Still much to explore here

Experimental Setup

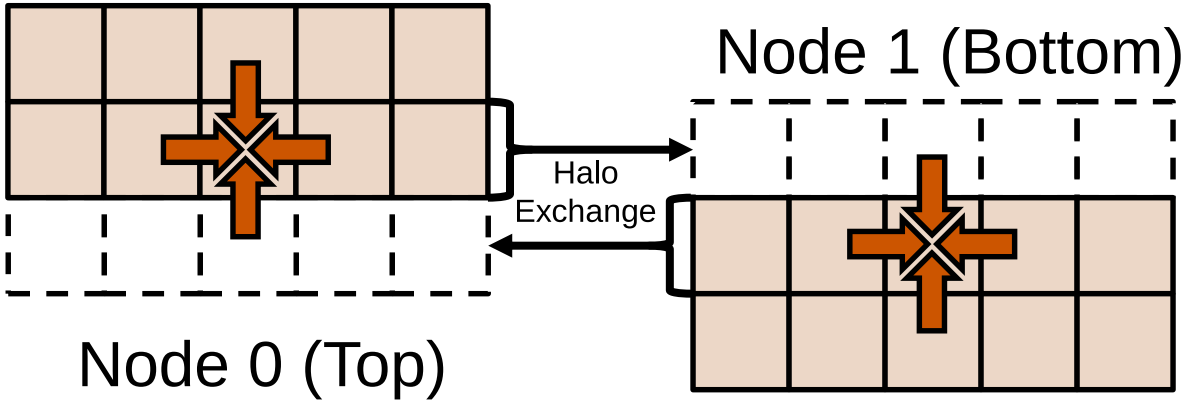
- **CPU: Standard CPU-only systems**
  - Baseline non-accelerated system
- **HDN: Host Driven Networking**
  - Kernel boundary networking (host MPI + CUDA)

Intra-kernel Networking Schemes:

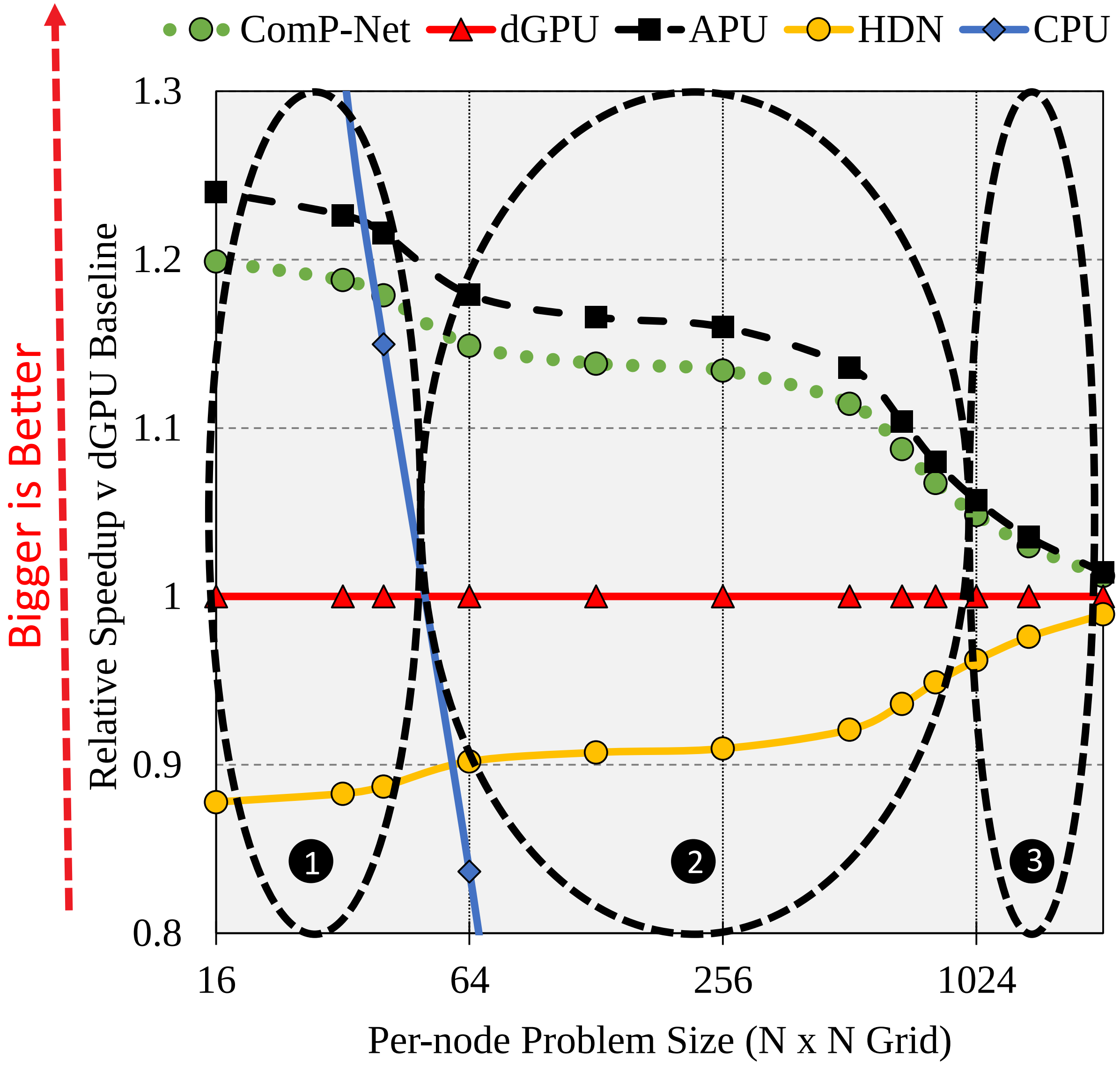
- **APU: CPU/GPU on the Same Die**
  - Intra-kernel networking through host threads on an APU
- **dGPU: GPU Host Networking**
  - Intra-kernel networking through host threads on a dGPU
- **ComP-Net: Command Processor Networking**
  - Intra-kernel networking through command processor

CPU and Memory Configuration	
Type	8-wide OOO, x86, 8 cores @ 4GHz
I,D-Cache	64KB, 2-way, 2 cycles
L2-Cache	2MB, 8-way, 8 cycles
L3-Cache	16MB, 16-way, 20 cycles
DRAM	DDR4, 8 Channels, 2133MHz
GPU Configuration	
Type	AMD GCN3 @ 1.5GHz
CU Config	12 CUs with 4 SIMD-16 engines
Wavefronts	40 Waves per SIMD (64 lanes)
V-Cache	32KB, 16-way, 12 cycles, per CU
K-Cache	32KB, 8-way, 12 cycles, per 4 CU
I-Cache	64KB, 8-way, 12 cycles, per 4 CU
L2-Cache	1MB, 16-way, 8 banks, 100 cycles
CP Configuration	
Type	2-wide OOO, x86, 2 cores @ 2GHz
D-Cache	32KB, 8-way, 4 cycles
I-Cache	16KB, 8-way, 4 cycles

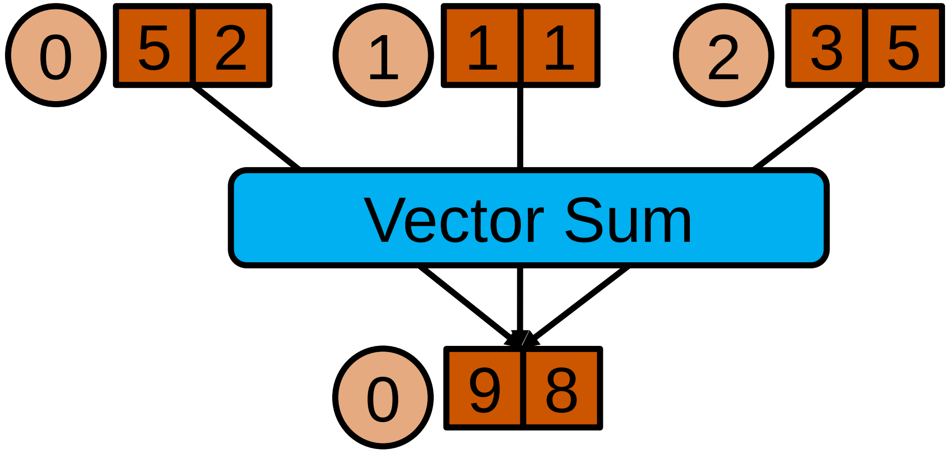
Results



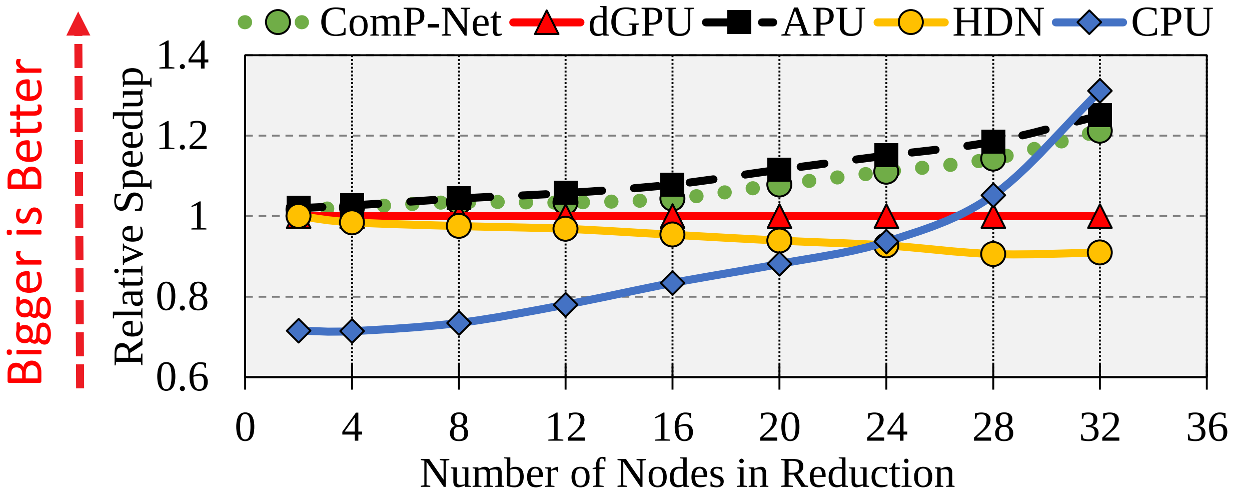
- **2D Jacobi Stencil**
  - 1D data decomposition
  - Iterative compute and halo exchange
  - Three regions of interest

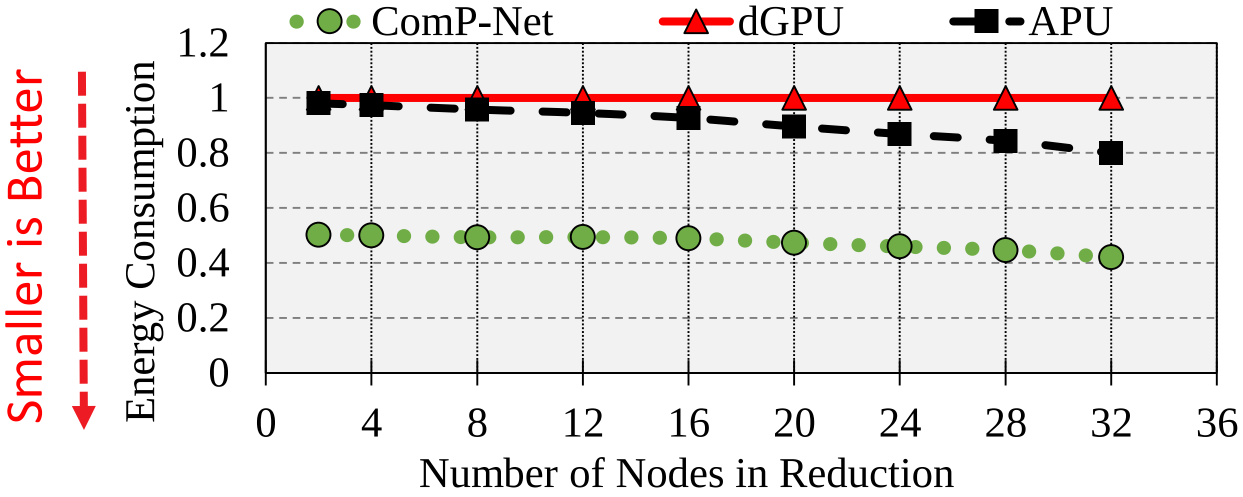


Results

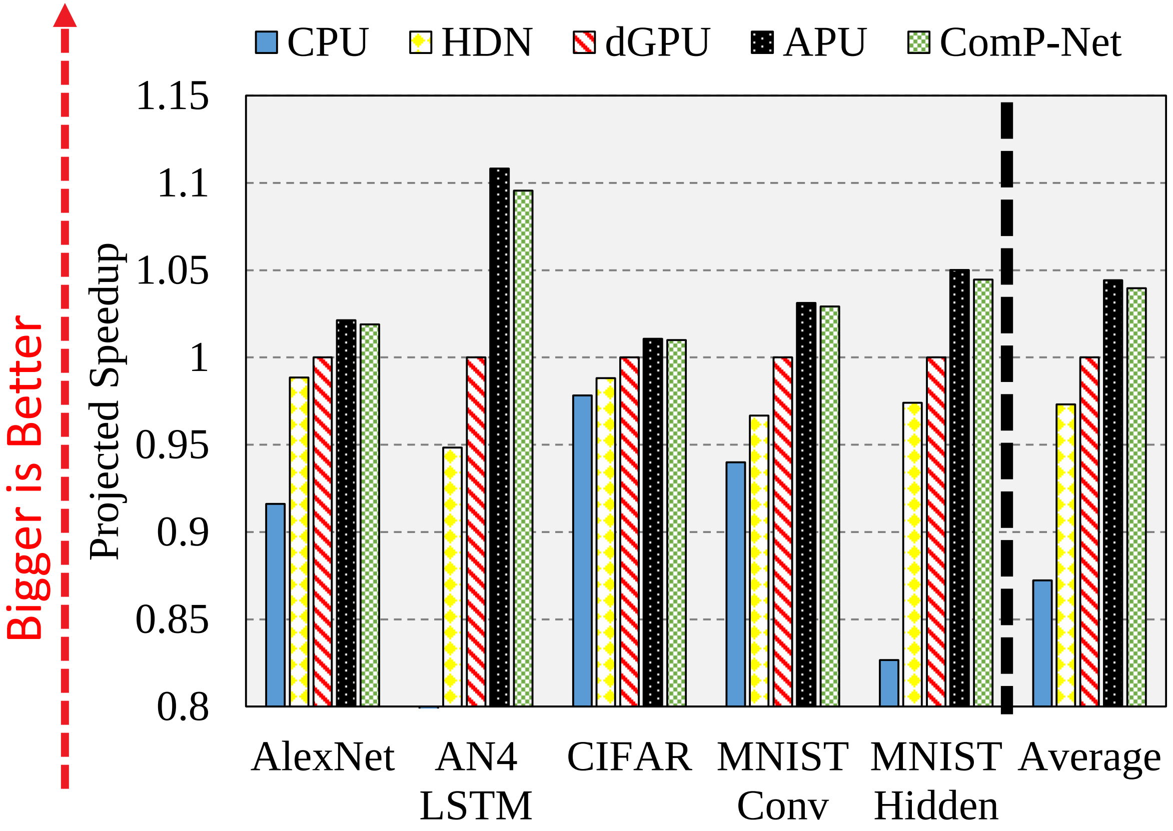


- **64MB Reduction (strong scaling)**
  - APU performs better than ComP-Net
  - ComP-Net is much more energy efficient





Results



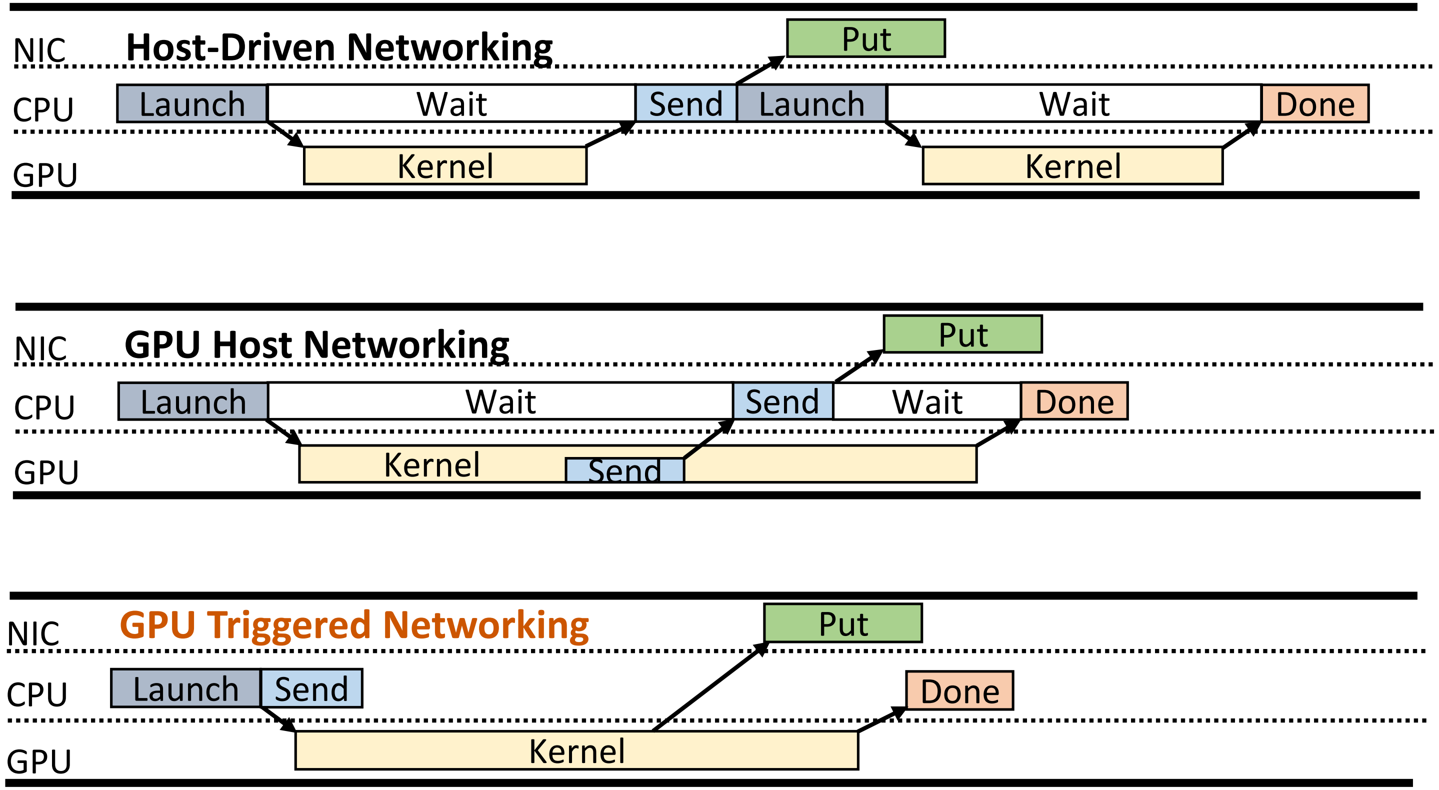
Workload Name	Domain	%Blocked	Reductions
Alex Net	Classification	14%	4672
AN4 LSTM	Speech	50%	131192
CIFAR	Classification	4%	939820
Large Synth	Synthetic	28%	52800
MNIST Conv	Text Recognition	12%	900000
MNIST Hidden	Text Recognition	29%	900000

Outline

- Introduction
- Contribution 1: Extended Task Queuing
- Contribution 2: Command Processor Networking
- **Contribution 3: GPU Triggered Networking**
- Conclusion

GPU Triggered Networking (GPU-TN) Overview

- **CPU creates network operation off the critical path**
  - Registers with the NIC
- **GPU simply ‘triggers’ operation when the data is ready**
- **Provides intra-kernel GPU networking without requiring a CPU thread**



M. LeBeane, K Hamidouche, B. Benton, M. Breternitz, S. K. Reinhardt, and L. K. John, “GPU Triggered Networking for Intra-Kernel Communications,“ in Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC), 2017.

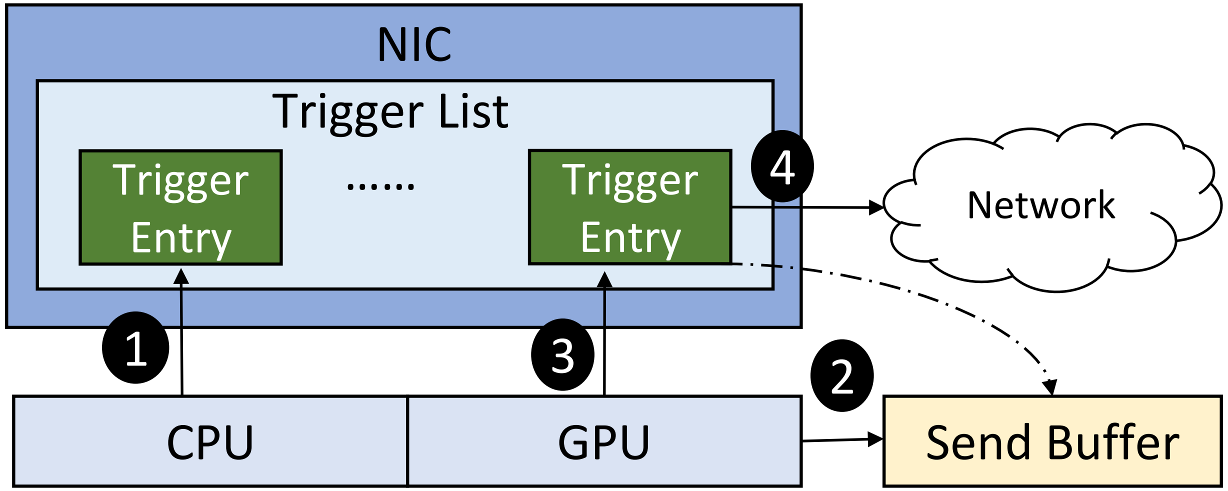
GPU-TN Architecture

❶ CPU Creates Triggered Entry

- Trigger Entry consists of:
  - Network Operation
  - Tag
  - Counter
  - Threshold
- Appends entry to Trigger List

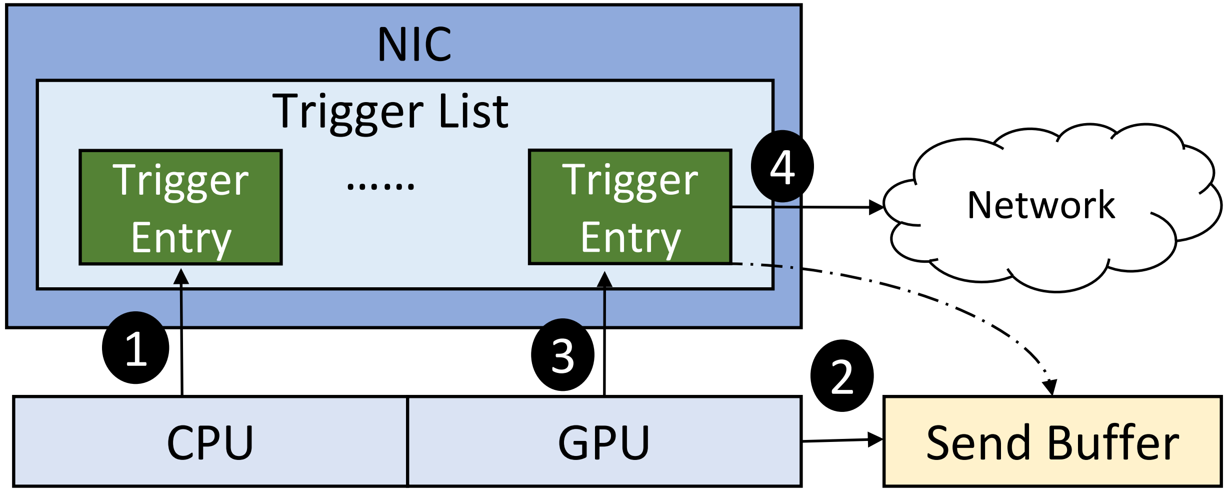
❷ GPU Fills Send Buffer

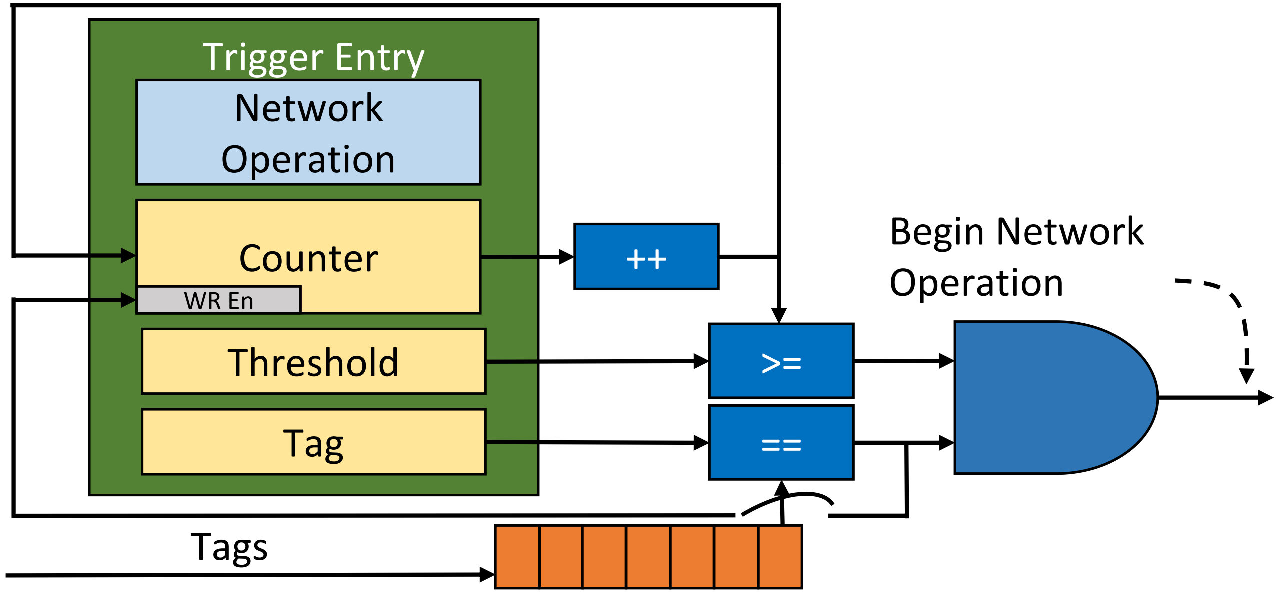
- During kernel execution



GPU-TN Architecture

- ❸ **GPU initiates Put operation**
  - GPU Provides Tag
- ❹ **NIC sends message**
  - Message triggered when counter >= CPU provided threshold
- **HW complexity?**
  - ‘Trigger list’ might not be a list
- **CPU/GPU race conditions?**
  - Allocate null entry for unexpected triggers





Experimental Setup

- **CPU: Standard CPU-only systems**
  - Baseline non-accelerated system
- **HDN: Host Driven Networking**
  - No driver interactions on the critical path, but may involve CPU runtime
- **GDS-Sim: GPUDirect Async**
  - Preregistration of communication but at kernel boundaries
- **GHN: GPU Host Networking**
  - Intra-kernel networking through host threads
- **GPU-TN: GPU Triggered Networking**
  - Preregistration of network operations and intra-kernel networking

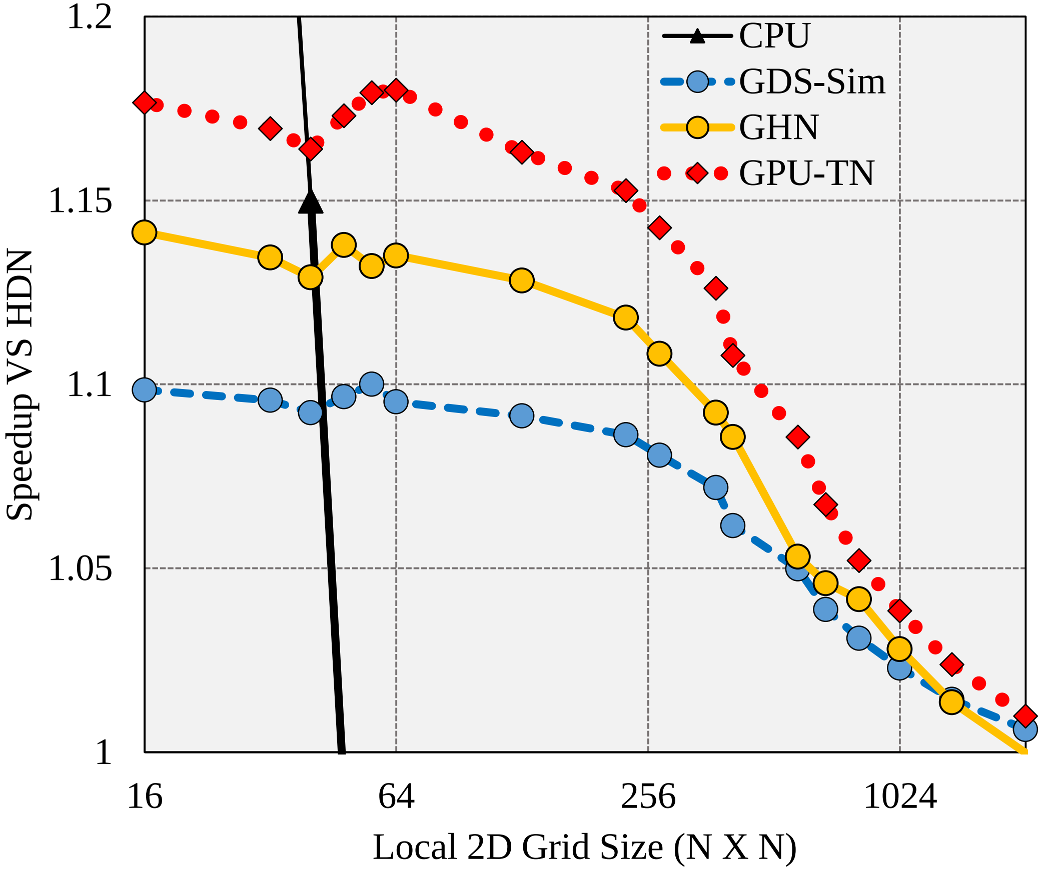
CPU and Memory Configuration	
Type	8-wide OOO, x86, 8 cores @ 4GHz
I,D-Cache	64KB, 2-way, 2 cycles
L2-Cache	2MB, 8-way, 8 cycles
L3-Cache	16MB, 16-way, 20 cycles
DRAM	DDR4, 8 Channels, 2133MHz

GPU Configuration	
Type	AMD GCN3 @ 1.5GHz
CU Config	24 CUs with 4 SIMD-16 engines
Wavefronts	40 Waves per SIMD (64 lanes)
V-Cache	32KB, 16-way, 12 cycles, per CU
K-Cache	32KB, 8-way, 12 cycles, per 4 CU
I-Cache	64KB, 8-way, 12 cycles, per 4 CU
L2-Cache	1MB, 16-way, 8 banks, 100 cycles

NIC Configuration	
Link Speed	100ns/ 100Gbps
Topology	Star

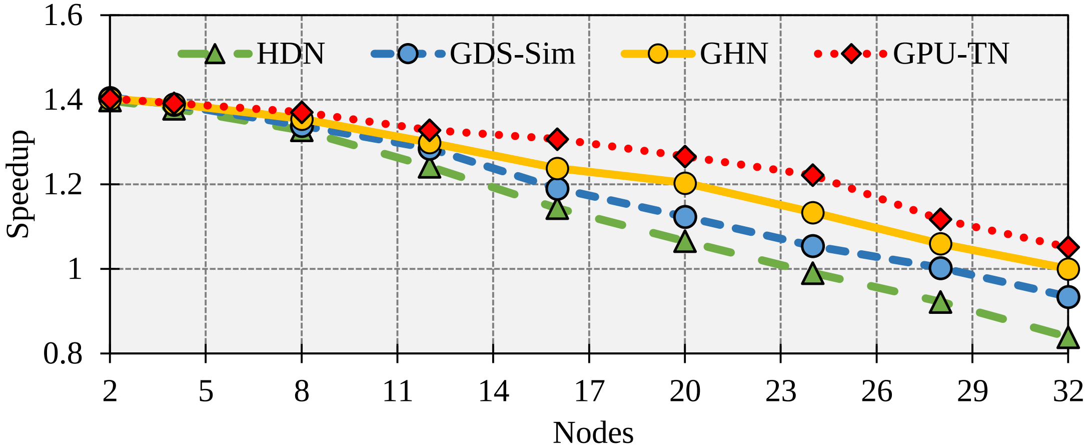
Results

Bigger is Better

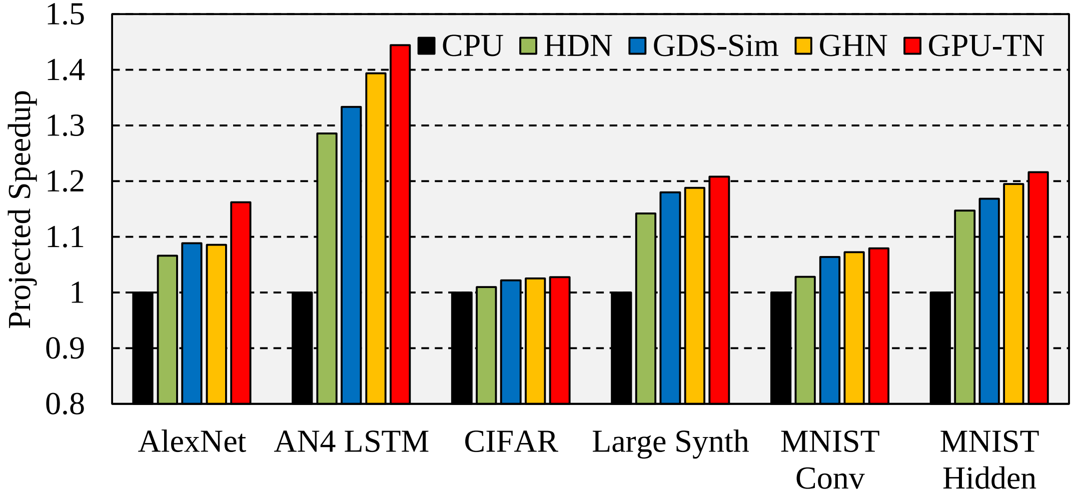


- 2D Jacobi Stencil

Bigger is Better



- **64MB Reduction (strong scaling)**



- **Machine Learning Training Phase**

Outline

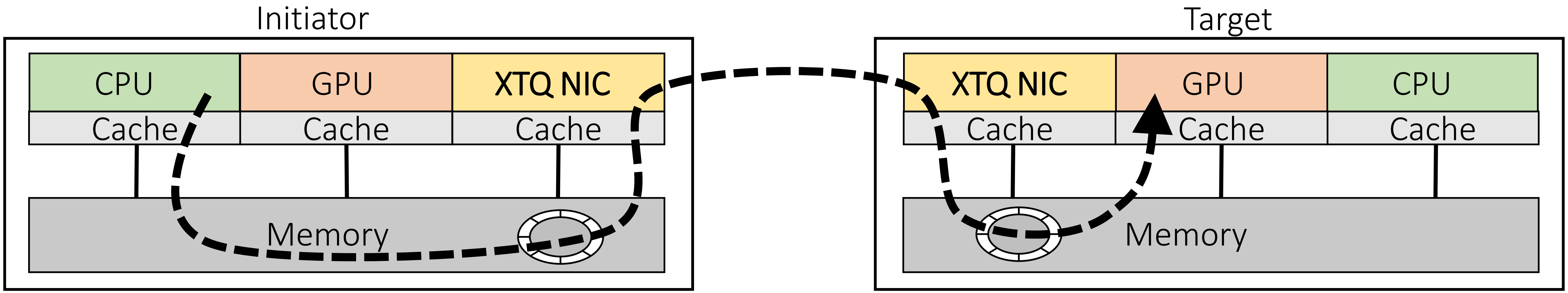
- Introduction
- Contribution 1: Extended Task Queuing
- Contribution 2: Command Processor Networking
- Contribution 3: GPU Triggered Networking
- **Conclusion**

Summary

- **Presented 3 enhancements to improve GPU networking**
  - **Extended Task Queuing**
    - Direct NIC-to-GPU active messaging
  - **Command Processor Networking**
    - Dynamic communication using on-chip GPU Command Processor
  - **GPU Triggered Networking**
    - Initiate messages without critical path CPU

Extended Task Queuing (XTQ) Summary

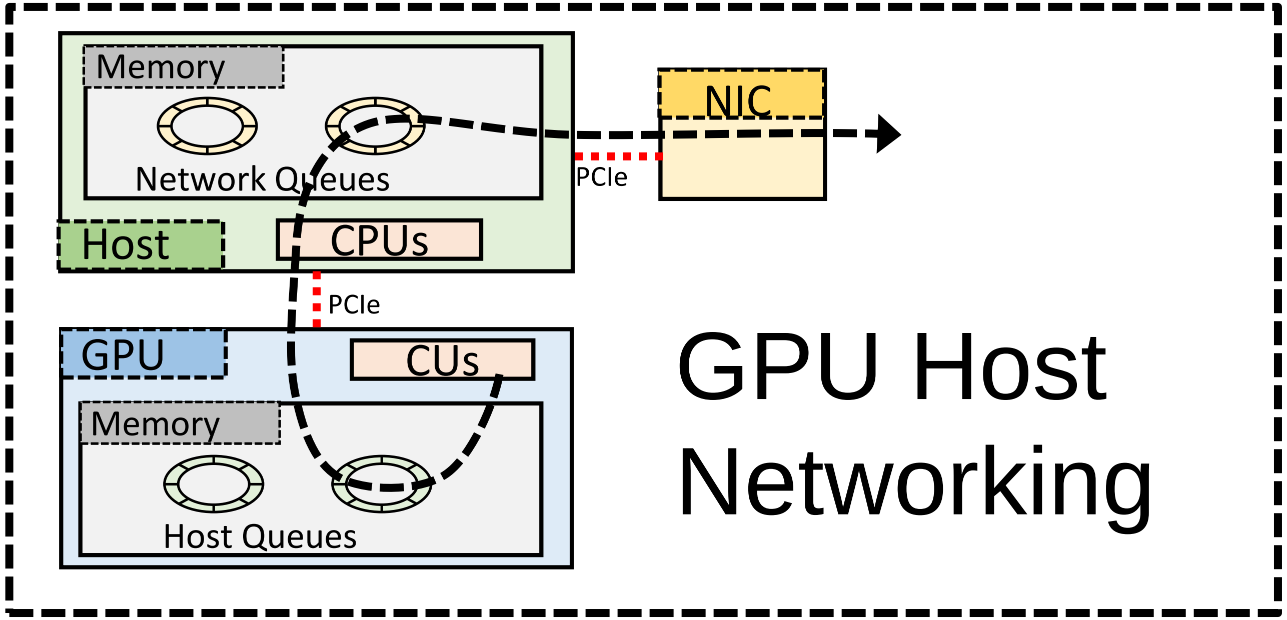
- **XTQ allows direct access to remote GPU queues**
  – Teach NICs how to speak with HSA queues
- **Enables *Active Messaging* without target CPU involvement**
  – Improves latency and frees CPU service thread(s)
- **Improves application performance by ~15%**

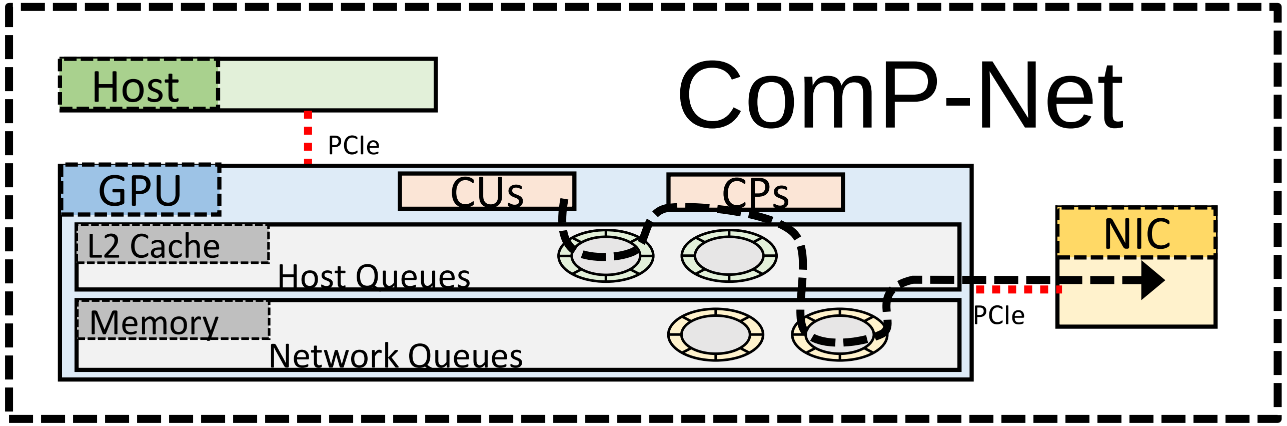


M. LeBeane, B. Potter, A. Pan, A. Dutu, V. Agarwala, W. Lee, D. Majeti, B. Ghimire, E. Van Tassell, S. Wasmundt, B. Benton, M. Breternitz, M. L. Chu, M. Thottethodi, L. K. John, and S. K. Reinhardt, \Extended task queuing: active messages for heterogeneous systems," in Proc. of the Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC), 2016.

Command Processor Networking (ComP-Net) Summary

- **Uses built in CP to support network operations**
- **CP/GPU communicate over shared L2 cache instead of PCIe**
- **Potentially much faster (lower latency) than other GHN designs**
- **Scales naturally**
  - Every GPU has multiple CP threads
- **Improves application performance ~20% vs other GHN approaches**

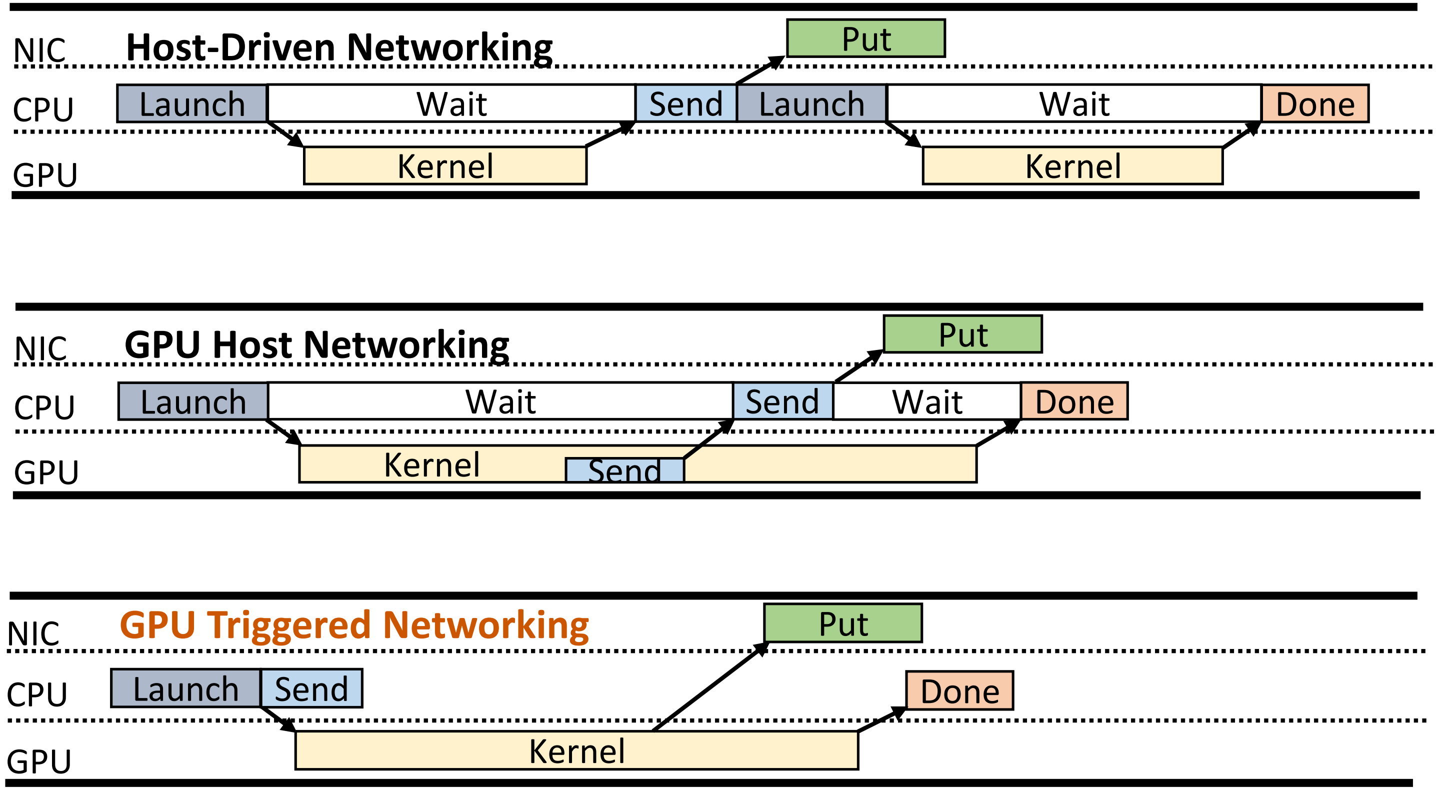




M. LeBeane, K. Hamidouche, B. Benton, M. Breternitz, S. K. Reinhardt, and L. K. John, “ComP-Net: Command Processor Networking for Efficient Intra-kernel Communications on GPUs," in Proc. of the Intl. Conf Parallel Architectures and Compilation Techniques (PACT), 2018.

GPU Triggered Networking (GPU-TN) Summary

- **CPU creates network operation off the critical path**
  - Registers with the NIC
- **GPU simply 'triggers' operation when the data is ready**
- **Provides intra-kernel GPU networking without requiring a CPU thread**
- **Improves application performance ~20% vs GPUDirect Async**



M. LeBeane, K Hamidouche, B. Benton, M. Breternitz, S. K. Reinhardt, and L. K. John, "GPU Triggered Networking for Intra-Kernel Communications," in Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC), 2017.

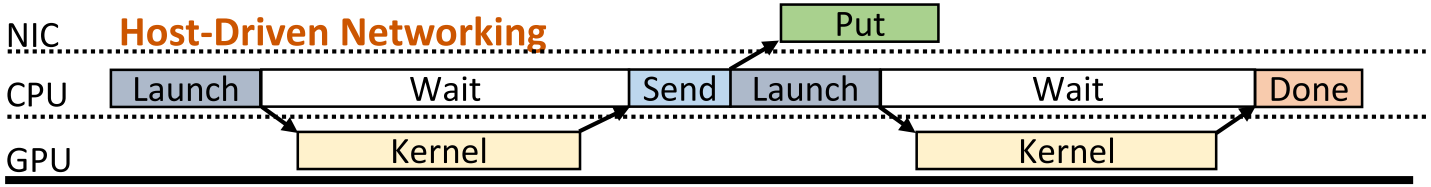
Towards the Future…..

- **This dissertation motivates the need for more independent accelerators**
  - Cannot funnel everything through a central CPU!
  - Concepts are applicable to many types of accelerators and networks
- **Still much to do!**
  - Application Redesign Opportunities
    - Applications presented in this dissertation are scratching the surface
    - Algorithms with dynamic communication could significantly benefit from these techniques
  - Leveraging Emerging NIC Technologies for GPUs
    - Mellanox BlueField, collective offload, programmable message handlers
    - How could more intelligent NICs assist with GPU networking?

Thank You!

Host-Driven Networking (HDN)

- **CPU controls networking through driver/runtime**
- **Messages sent at kernel boundaries**
- **Research implementations include:**
  - CUDA-Aware MPI [Kraus ‘14]
  - CUDA-Aware OpenSHMEM [Hamidouche ’16]
    - GPUDirect RDMA [Mellanox ‘13]



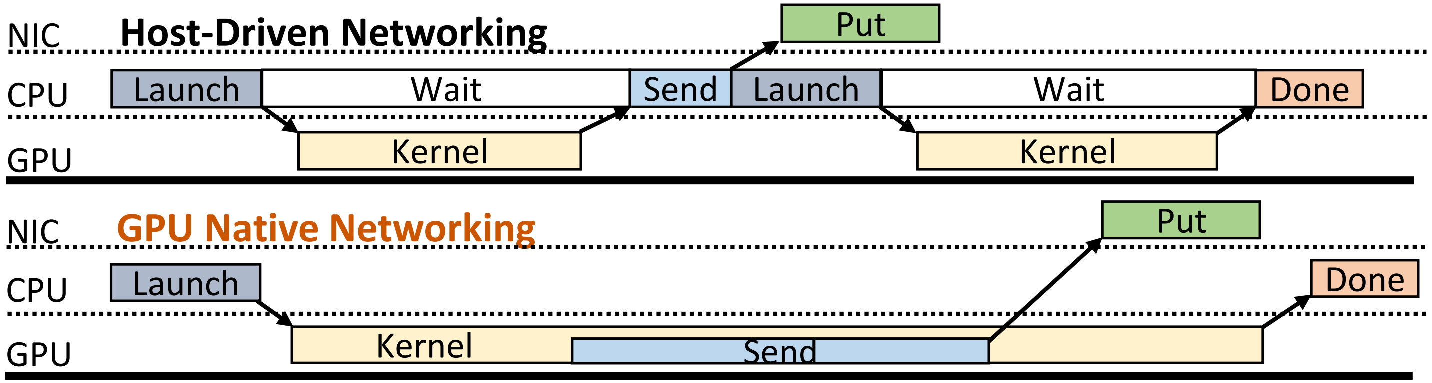
J. Kraus. “Introduction to CUDA-aware MPI and Nvidia GPUDirect,” *GPU Tech. Conference*. 2014.

K. Hamidouche, A. Venkatesh, A. A. Awan, H. Subramoni, C.H. Chu, and D. K. Panda, “CUDA-Aware OpenSHMEM,” *Journal on Parallel Computing.* 2016.

Mellanox, “Mellanox GPUDirect RDMA User Manual,” http://www.mellanox.com/related-docs/prod_software/Mellanox GPUDirect User Manual v1.2.pdf. 2015

GPU Native Networking (GNN)

- **GPU runs networking stack**
- **Persistent kernels and LDS memory used for network data structures**
- **Research implementations include:**
  - GPUrdma [Daoud ’16]
  - IBV on GPUs [Oden ‘14]



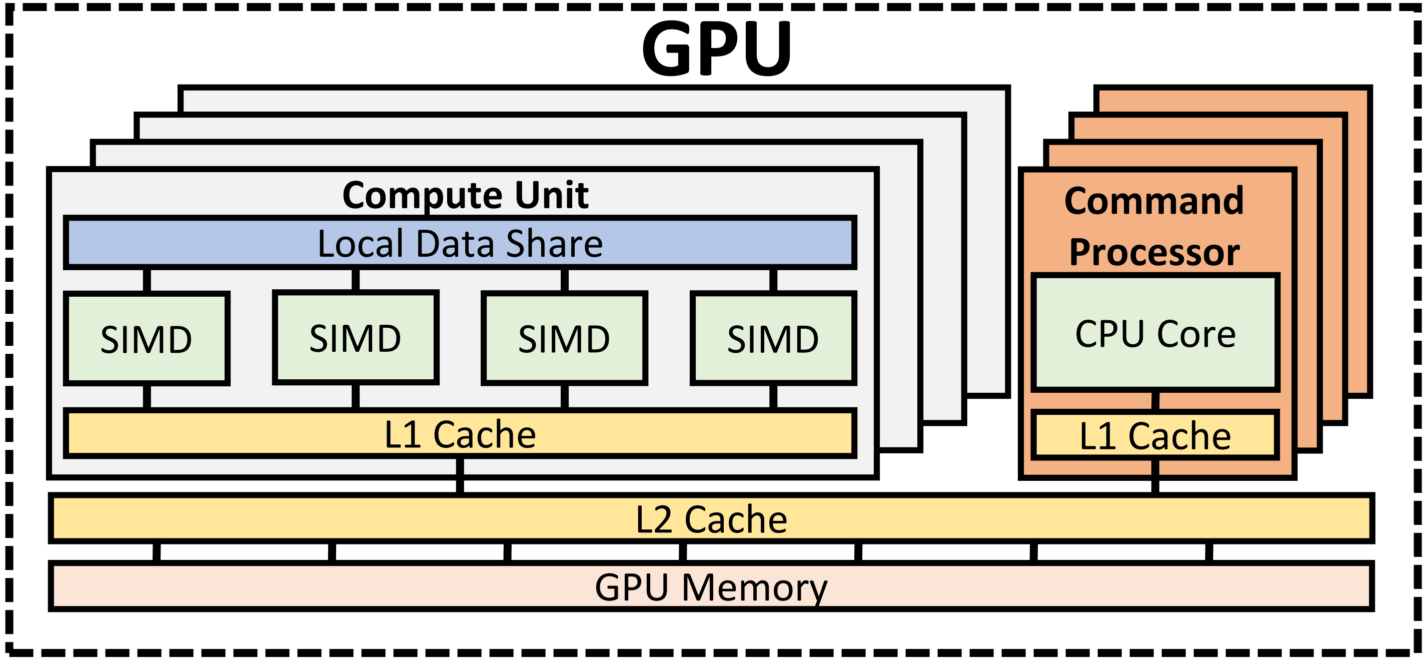
F. Daoud, A. Watad, and M. Silberstein, “GPUrdma: GPU-side Library for High Performance Networking from GPU Kernels,” *In Intl. Workshop on Runtime and Operating Systems for Supercomputers* (ROSS). 2016.

L. Oden, H. Froning, and F. J. Pfreundt, “Infiniband-Verbs on GPU: A Case Study of Controlling an Infiniband Network Device from the GPU,” *In Intl. Conf. on Parallel Distributed Processing Symposium Workshops* (IPDPSW). 2014.

GPU Architecture and Terminology

AMD <=> Nvidia Translator

- Work-item = Thread
- Wavefront (64 Threads) = Warp (32 Threads)
  - Unit of thread dispatch
- Work-group = Thread Block
  - Unit of Synchronization
- Local Data Share (LDS) = Shared Memory
  - Work-group scratchpad
- Compute Unit (CU) = Streaming Multi-Processor (SM)
  - Collection of SIMD engines sharing LDS and L1 cache



- Kernel
  - GPU SIMT Function
- Command Processor (CP)
  - Dispatch engine and scheduler

GPU-TN Kernel Programming Interface

Work-item Level

```
__kernel void
kern1(__global char *trigAddr,
      const int tagBase,
      __global void *buffer)
{
// do work
buffer = ...;
int id = get_global_id();
*trigAddr = tagBase + id;
// do additional work
...
}
```

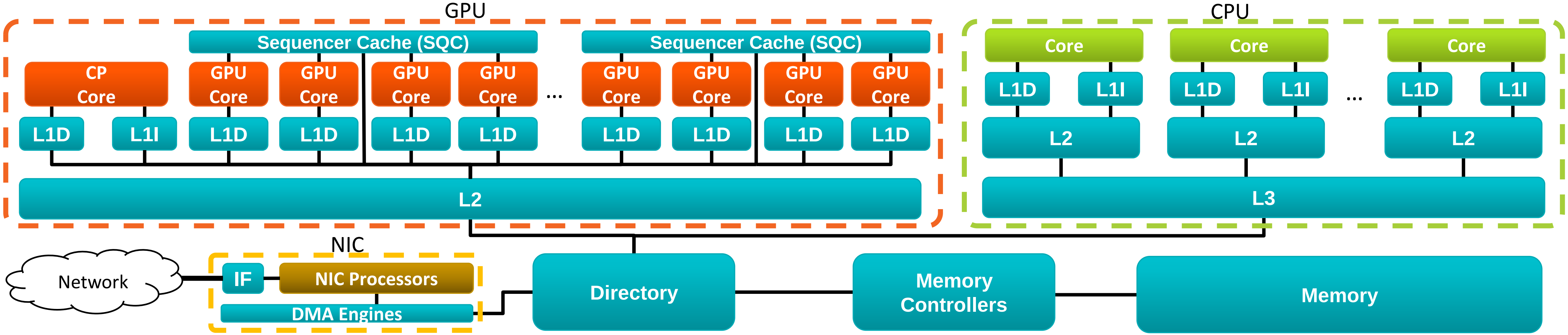
Work-group Level

```
__kernel void
kern2(__global char *trigAddr,
      const int tagBase,
      __global void *buffer)
{
// do work
buffer = ...;
wg_barrier();
if (!get_local_id()) {
  int id = get_group_id();
  *trigAddr = tagBase + id;
}
// do additional work
...
}
```

Kernel Level

```
__kernel void
kern3(__global char *trigAddr,
      const int tag,
      __global void *buffer)
{
// do work
buffer = ...;
wg_barrier();
if (!get_local_id())
  *trigAddr = tag;
// do additional work
...
}
```

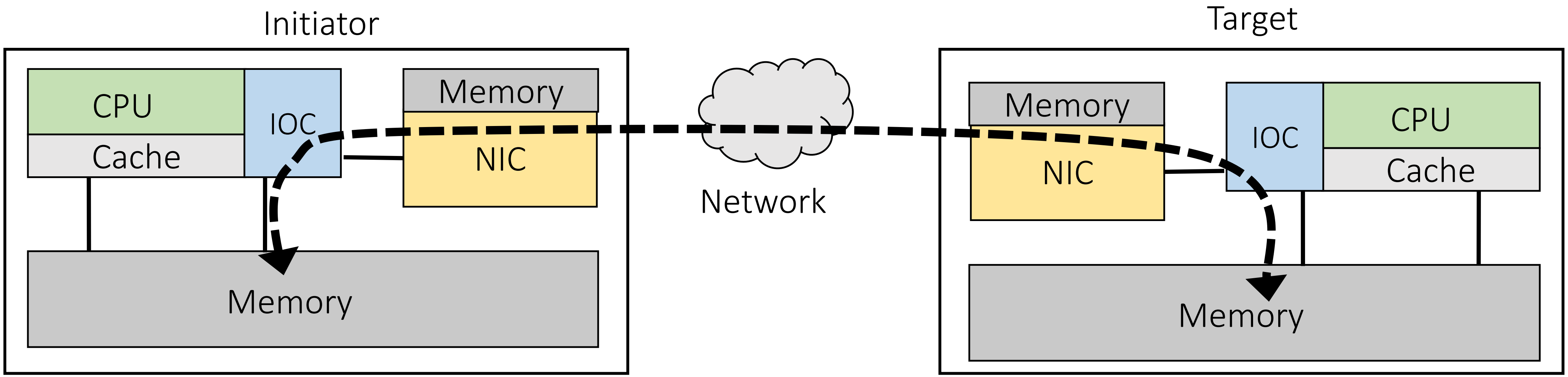
Simulation Infrastructure



- **gem5 + AMD GCN3 GPU model + Custom Portals4 NIC Model**
  - CPU power model with McPAT
  - Baseline model is coherent APU
    - dGPU modeled with extra delay for IO bus, different memory controllers, and by disabling coherence probes
- **Each section has slightly different parameters**
  - Will be discussed before results presented

Remote Direct Memory Access (RDMA)

- **RDMA allows for direct access of remote memory without involving CPU**
  - Heavy lifting is performed on the NIC (off-load networking model)
  - Generally expressed in terms of remote Put/Get operations
- **Maps naturally to “one-sided” communication semantics**
  - Puts/Gets vs. Send/Receive



ComP-Net Host and GPU API

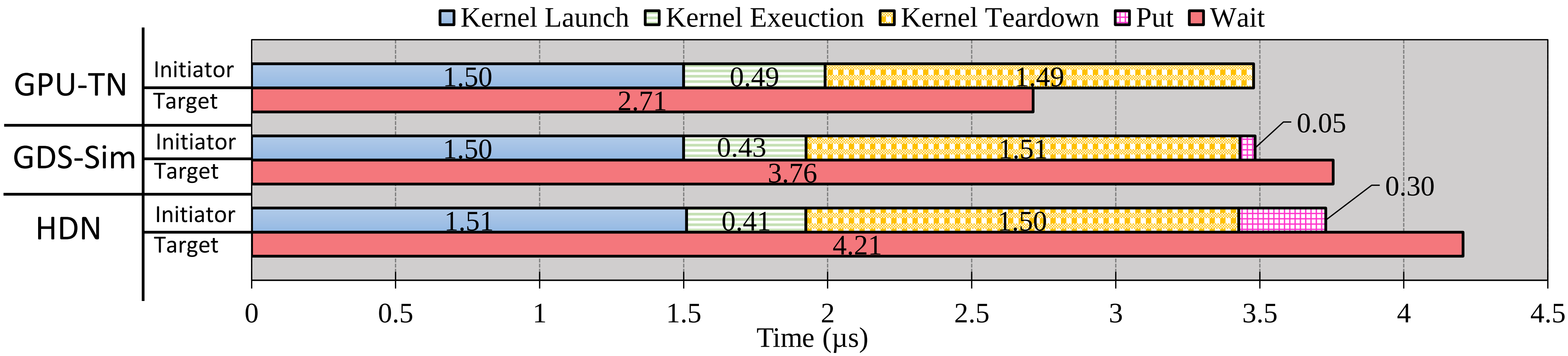
Host Code

```
__host__ void
hostInit() {
  //Initialize ComP-Net
  cpnet_handle_t* cpnet_handle;
  cpnet_init(&cpnet_handle, GRID_SZ / WG_SZ);
  // Allocate symmetric heap memory
  char* buf = cpnet_shmalloc(sizeof(char) *
                             GRID_SZ / WG_SZ);
  //Initiator/target launches kernel
  if (cpnet_handle->pe == INITIATOR) {
    hipLaunchKernel(Ping, GRID_SZ, GRID_SZ /
                    WG_SZ, 0, 0, cpnet_handle,
                    buf);
  } else { /* Launch target kernel. */ }
}
```

GPU Code

```
__device__ void
Ping(cpnet_handle_t *cpnet_handle,
     char* wg_buffer) {
  // Extract context from global handle
  __shared__ cpnet_ctx_t cpnet_ctx;
  cpnet_ctx_create(cpnet_handle,
                   cpnet_ctx);
  // Each WG pings target
  cpnet_shmem_char_p(cpnet_ctx,
                     wg_buffer[hipBlockIdx_x],
                     1, TARGET);
  // Each WG waits for pong target
  cpnet_shmem_char_wait_until(
    wg_buffer[hipBlockIdx_x, 1);
  cpnet_ctx_destroy(cpnet_ctx);
}
```

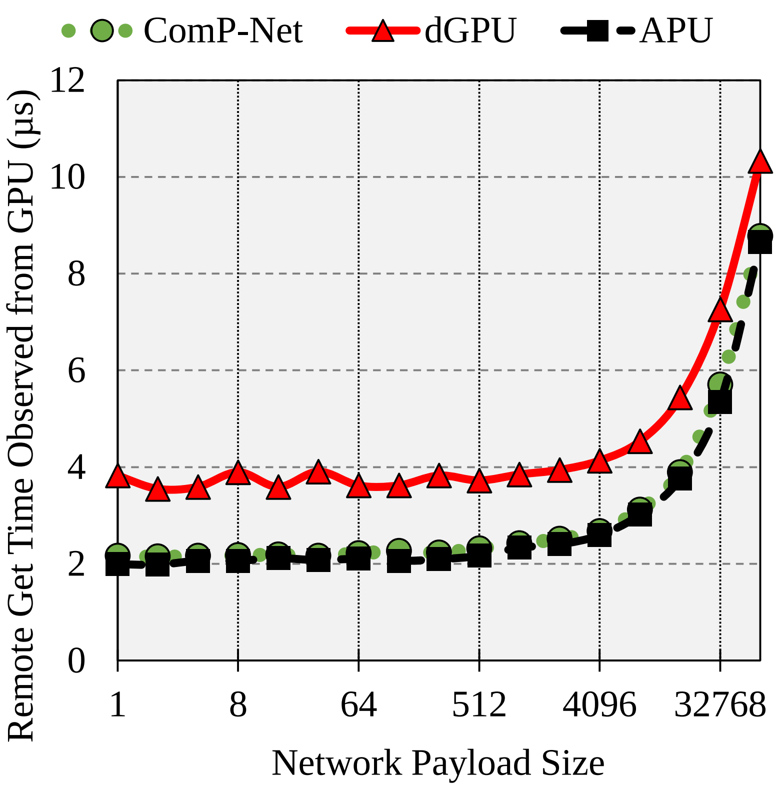
Latency Microbenchmark



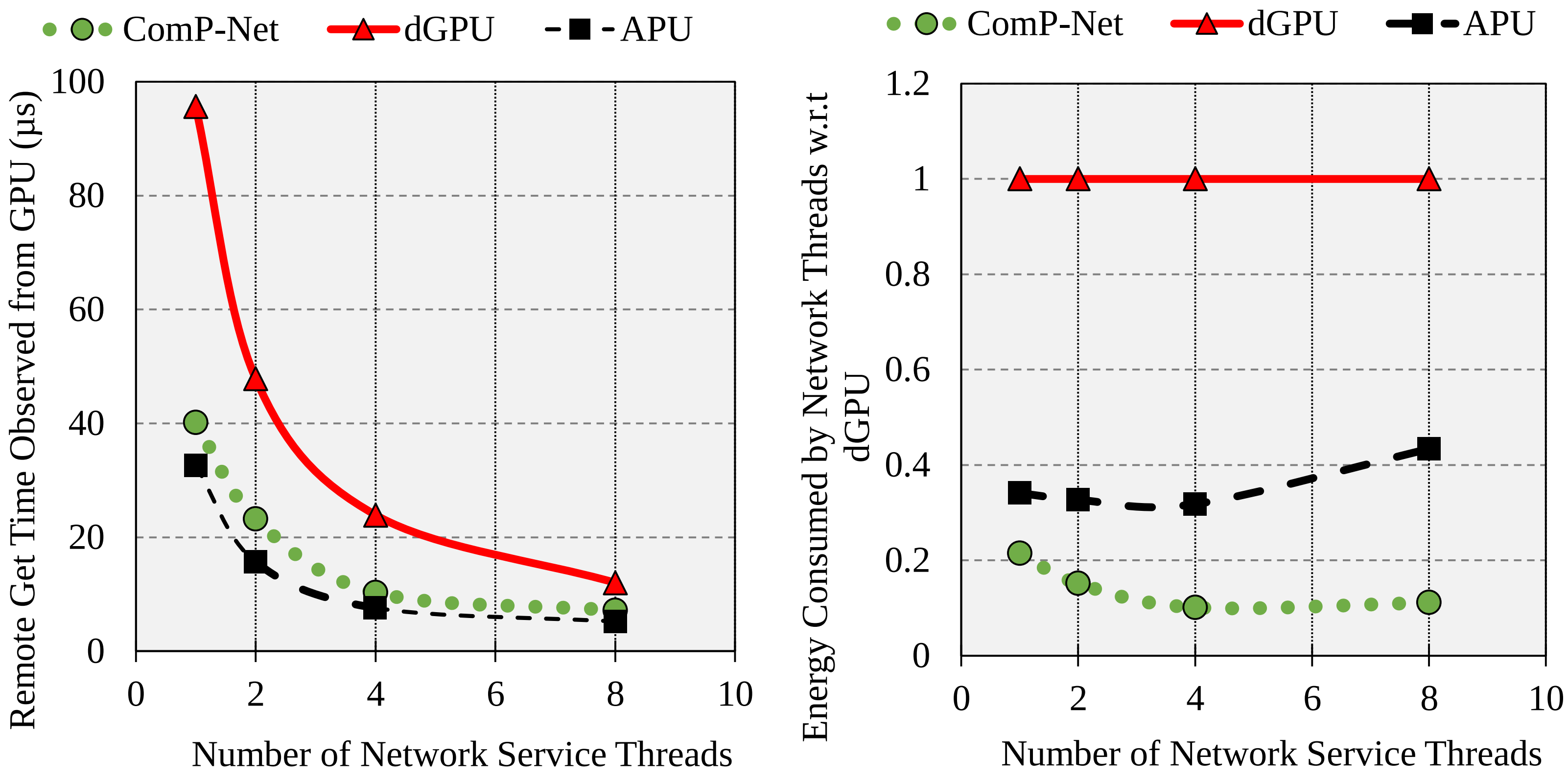
Smaller is Better

- **One-sided put latency benchmark**
  - Initiator launches dummy kernel, executes network command, and terminates
  - Target polls on put location
- **Take-away messages**
  - HDN < GDS-Sim < GPU-TN
  - GPU-TN actually overlaps kernel teardown with network transfer!

Microbenchmarks



Sweep of payload size for 1 WG and 1 Thread

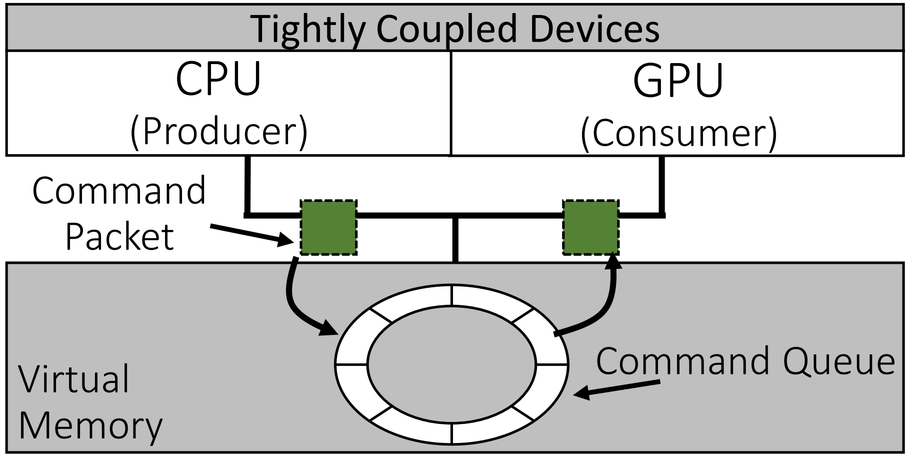


Sweep of threads for 1 byte transfers and 480 WGs

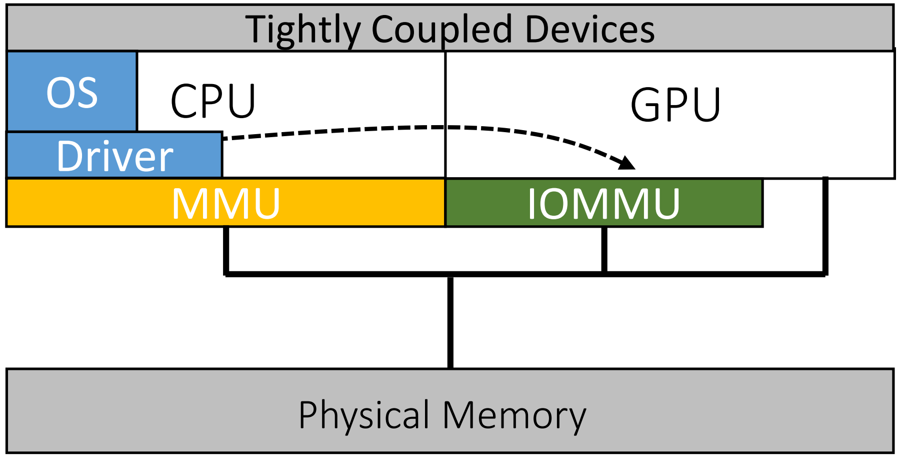
Where are GPUs heading?

- **Friendlier programming abstractions**
  - Nicer abstractions in CUDA and OpenCL
    - Dynamic Parallelism, Unified Memory, etc.
  - Single-source, kernel-less programming support
    - C++ AMP, OpenMP, AMD HC Language, etc.
- **Architectural Support**
  - User-level kernel-launch
  - Shared virtual address space
  - Virtualization
  - Multiprocessing
  - (Sometimes) Coherent caches

What about networking support?



Architected Queuing



Shared Virtual Memory