EXTENDED TASK QUEUING:

ACTIVE MESSAGES FOR HETEROGENEOUS SYSTEMS

MICHAEL LEBEANE
POST GRADUATE RESEARCHER, AMD RESEARCH
GRADUATE STUDENT, THE UNIVERSITY OF TEXAS AT AUSTIN
MICHAEL.LEBEANE@AMD.COM

AUTHORS

Michael LeBeane, Brandon Potter, Abhisek Pan, Alexandru Dutu, Vinay Agarwala, Wonchan Lee, Deepak Majeti, Bibek Ghimire, Eric Van Tassell, Samuel Wasmundt, Brad Benton, Mauricio Breternitz, Michael L. Chu, Mithuna Thottethodi, Lizy K. John, Steven K. Reinhardt

DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

The work described in this presentation was made with Government support awarded by the DOE. The Government may have certain rights in this work.

ACCELERATORS EVERYWHERE!

INTRODUCTION

- Accelerators (especially GPUs) are everywhere in modern HPC
- Over 80 of the Top 500 supercomputers use accelerators[1]
- 100’s of applications designed to leverage GPU compute[2]
- Accelerator communication across nodes is cumbersome....

[1] TOP500.org, “Highlights – June 2016,” http://www.top500.org/lists/2016/06/highlights, 2016.

[2] Nvidia, “GPU-Accelerated Applications,” http://www.nvidia.com/content/gpu-applications/pdf/gpu-apps-catalog-mar2015.pdf, 2016.

CROSS-NODE HETEROGENEITY

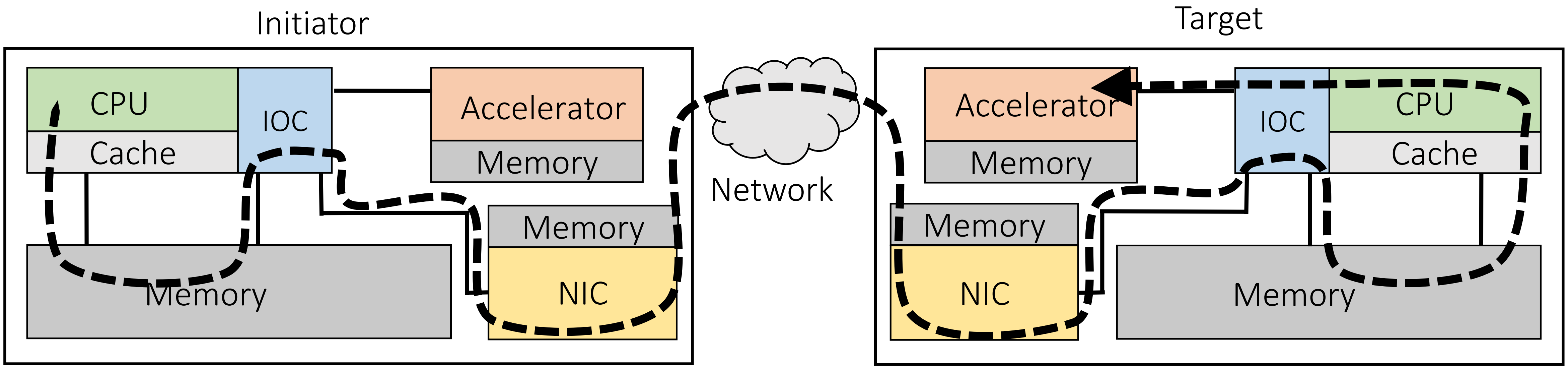
INTRODUCTION

Current HPC GPU Communication

- Send data through some local interconnect (e.g., PCIe) out of an HPC NIC (e.g., InfiniBand®)
- Target receives data and invokes GPU driver to enqueue task
- Optimized variants exist (e.g., GPUDirect RDMA[3]), but still must sync with CPU driver

Can we do better???

- Yes! But first we need to understand two important technologies…..

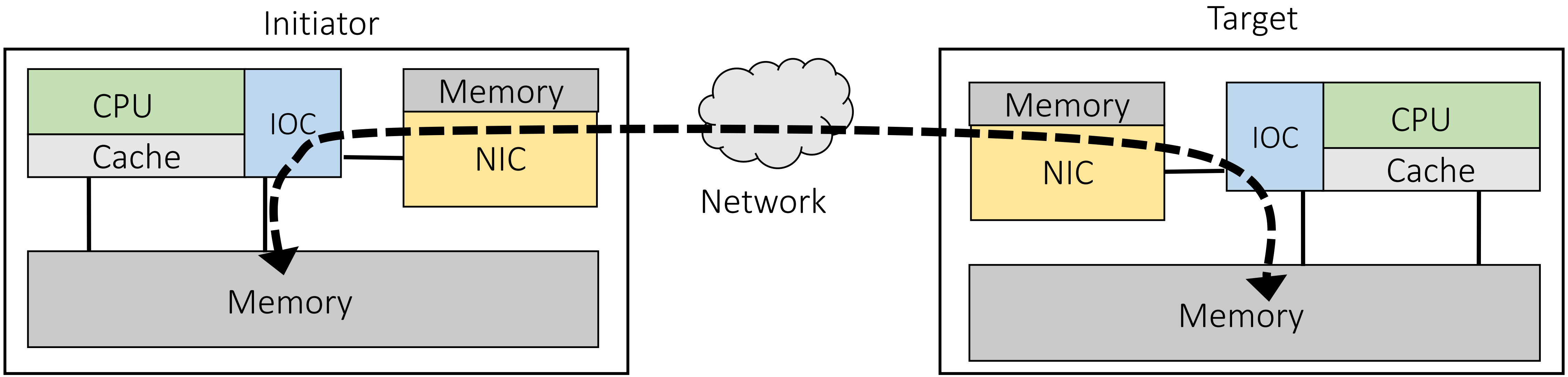


[3] Mellanox, "Mellanox GPUDirect RDMA user manual," http://www.mellanox.com/related-docs/prod_software/Mellanox_GPUDirect_User_Manual_v1.2.pdf, 2015.

REMOTE DIRECT MEMORY ACCESS (RDMA)

BACKGROUND

- 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
- Many common RDMA interfaces
  - RoCE, InfiniBand, iWARP, Portals 4, etc.



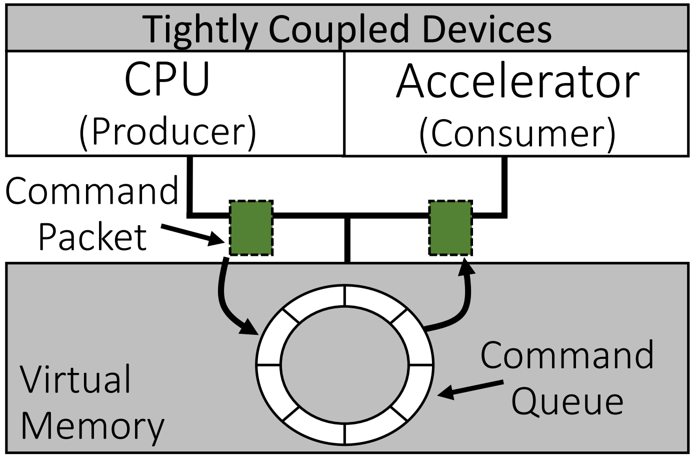
TIGHTLY COUPLED FRAMEWORKS

BACKGROUND

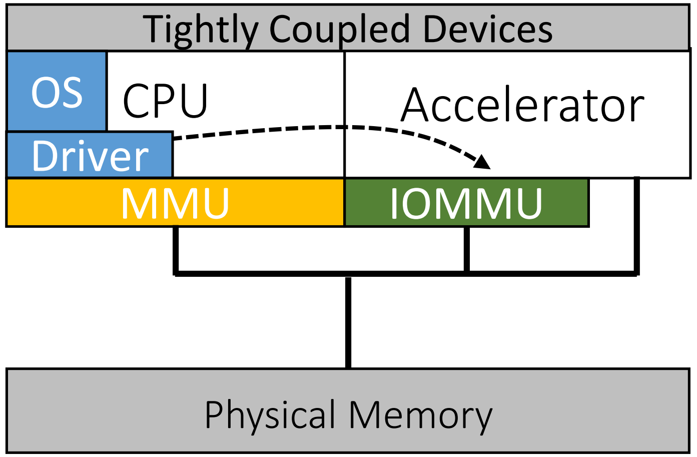
- ***Tightly Coupled Frameworks***
  - Complete system architectures and interconnects integrating CPUs, GPUs, and other accelerators
  - HSA™, OpenCAPI™, Gen-Z, CCIX, etc.

- HSA[4] will be our example tightly coupled framework for this work

- Relevant Features
  - User-level, architected command queuing
  - Globally coherent memory regions
  - Shared virtual address space



Architected Queuing



Shared Virtual Memory

[4] HSA Foundation, “HSA platform system architecture specification 1.0,” http://www.hsafoundation.com/standards, 2015.

CROSS-NODE HETEROGENEITY

INTRODUCTION

Tightly-coupled accelerator frameworks enable efficient, user-level task invocation between devices ***inside a node***

RDMA enables efficient data movement between devices ***across nodes***

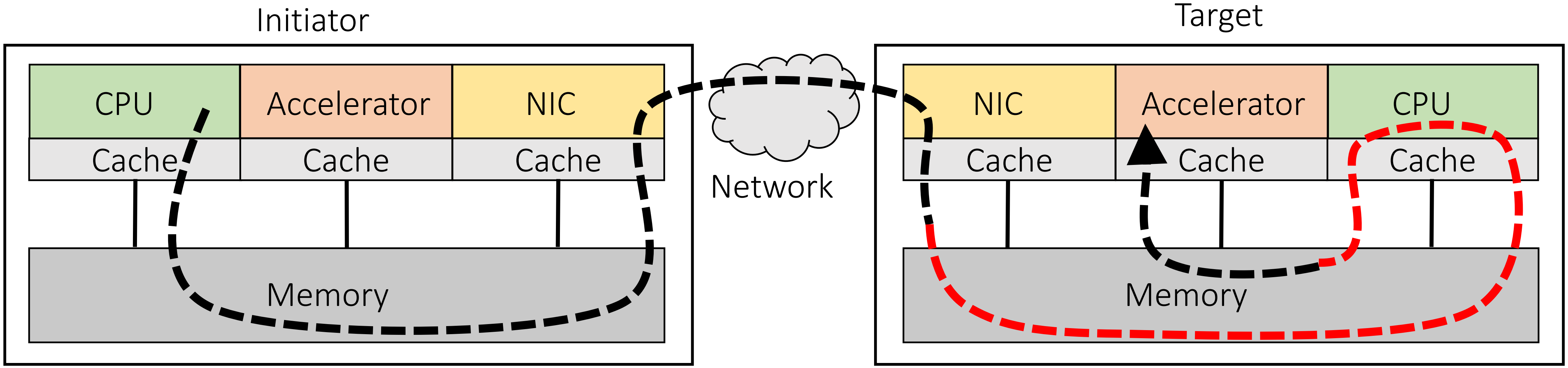
- By combining the ***intra-node*** tasking model of HSA with the ***inter-node*** data movement of RDMA, we can produce a ***generalized, user-level*** tasking framework for accelerators in distributed memory systems.

- What would such a system look like?

CROSS-NODE HETEROGENEITY

INTRODUCTION

- HSA-like Solution
  - Can communicate through shared virtual address space
  - **CPU must still launch tasks on target-side GPU**
  - Can we do even better?

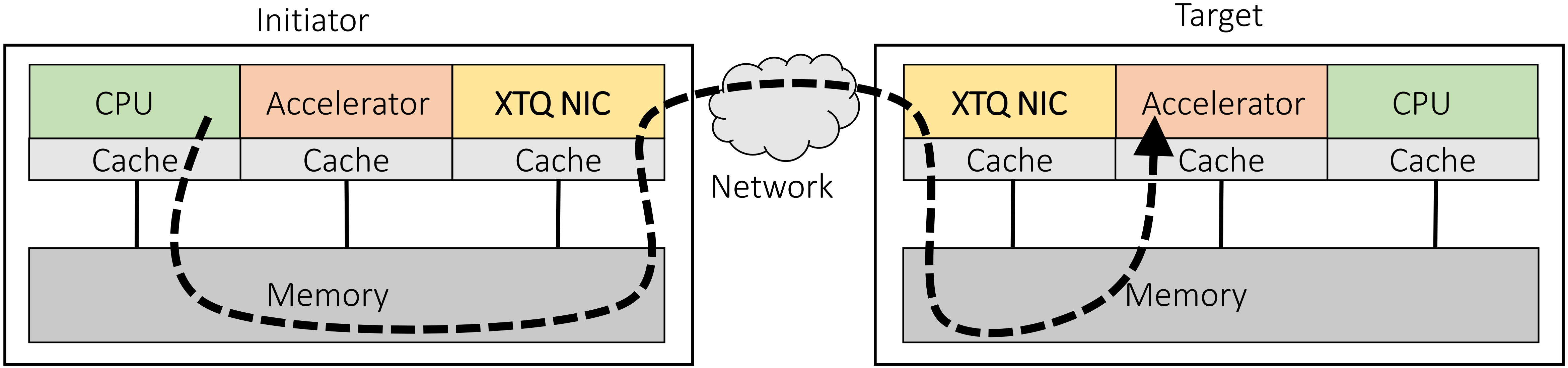


CROSS-NODE HETEROGENEITY

INTRODUCTION

- Our solution: ***Extended Task Queuing (XTQ)***
  - –Can communicate through shared virtual address space
  - –NIC is aware of all an chip compute devices
  - –**NIC is an HSA device**

XTQ



INTRODUCING XTQ

INTRODUCTION

AMD

- XTQ uses an HSA-compliant, RDMA-capable NIC to provide an **active messaging**[5] framework for all devices in distributed systems

- XTQ **reduces the launch latency** for remote GPU task invocation
  - Tasks are directly scheduled on the GPU by the NIC using shared memory queues

- XTQ **removes the message processing on the CPU** for GPU-destined tasks
  - The CPU is free to perform more useful computation

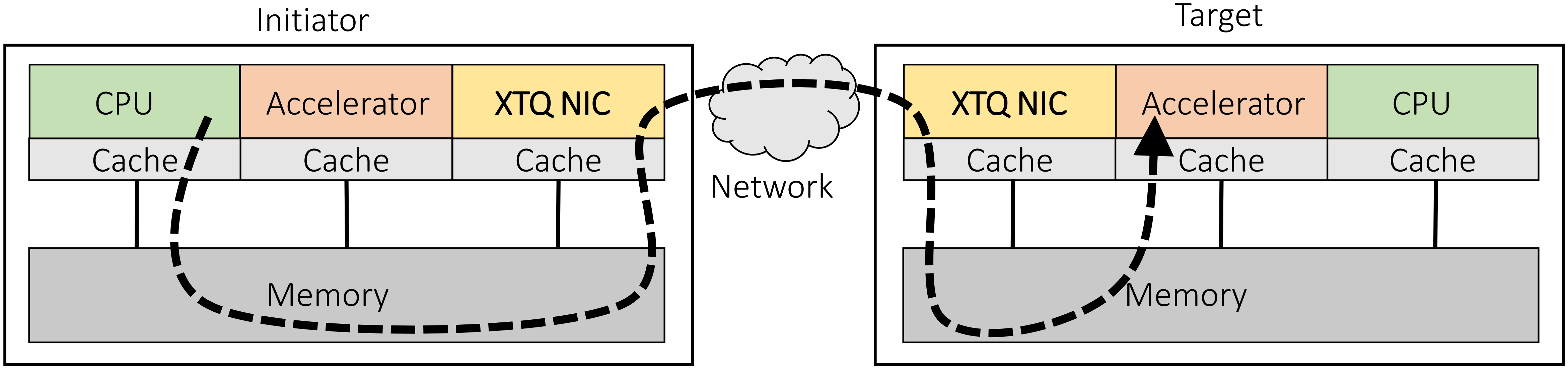
XTQ

[5] T. Eicken, D. Culler, S. Goldstein, and K. Schauser, "Active messages: A mechanism for integrated communication and computation," in Int. Symp. on Computer Architecture (ISCA), 1992, pp. 256–266.

OVERVIEW

XTQ ARCHITECTURE

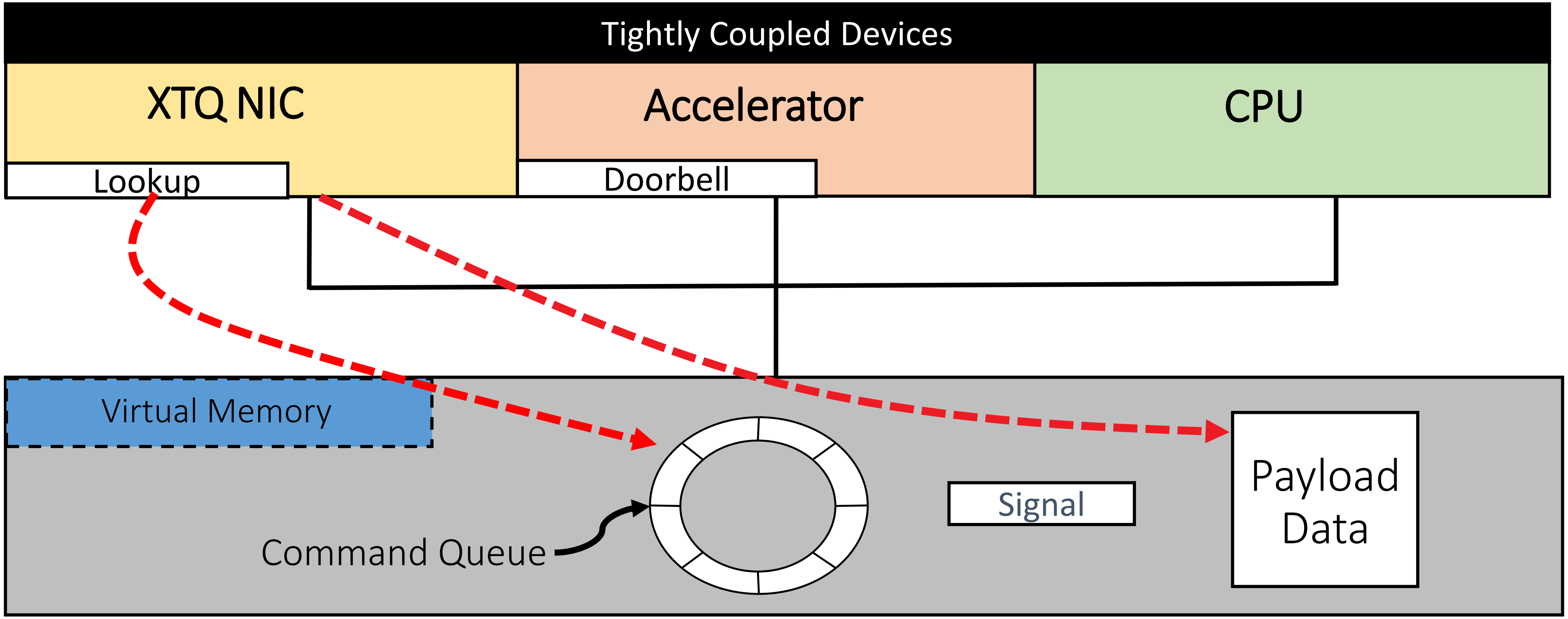
- XTQ NIC extends RDMA operations to access HSA task queues
- On initiator, put operation is very standard
  - NIC performs local DMA read of send buffer and transfers over the network
- **The magic happens on the target side**



TARGET-SIDE XTQ PUT

XTQ ARCHITECTURE

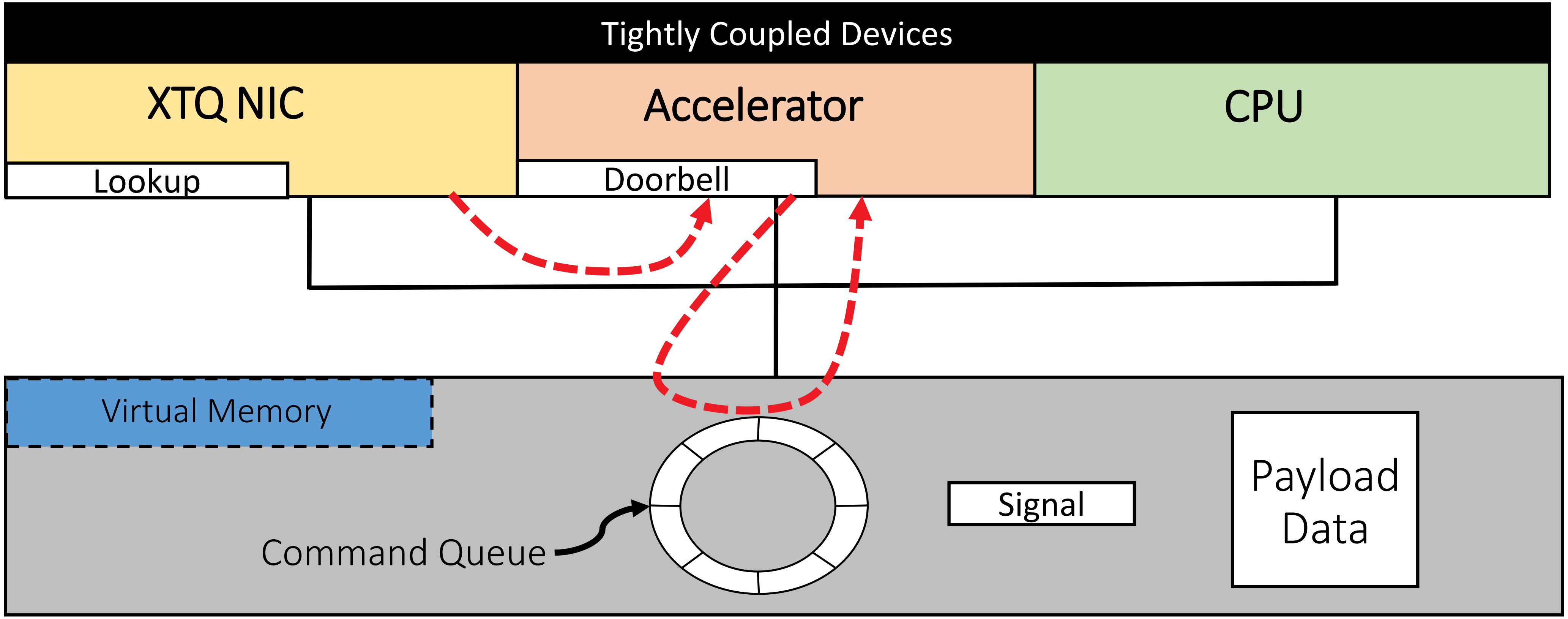
- Payload data streams into target-side receive buffer
- Command descriptor is placed into HSA queue



TARGET-SIDE XTQ PUT

XTQ ARCHITECTURE

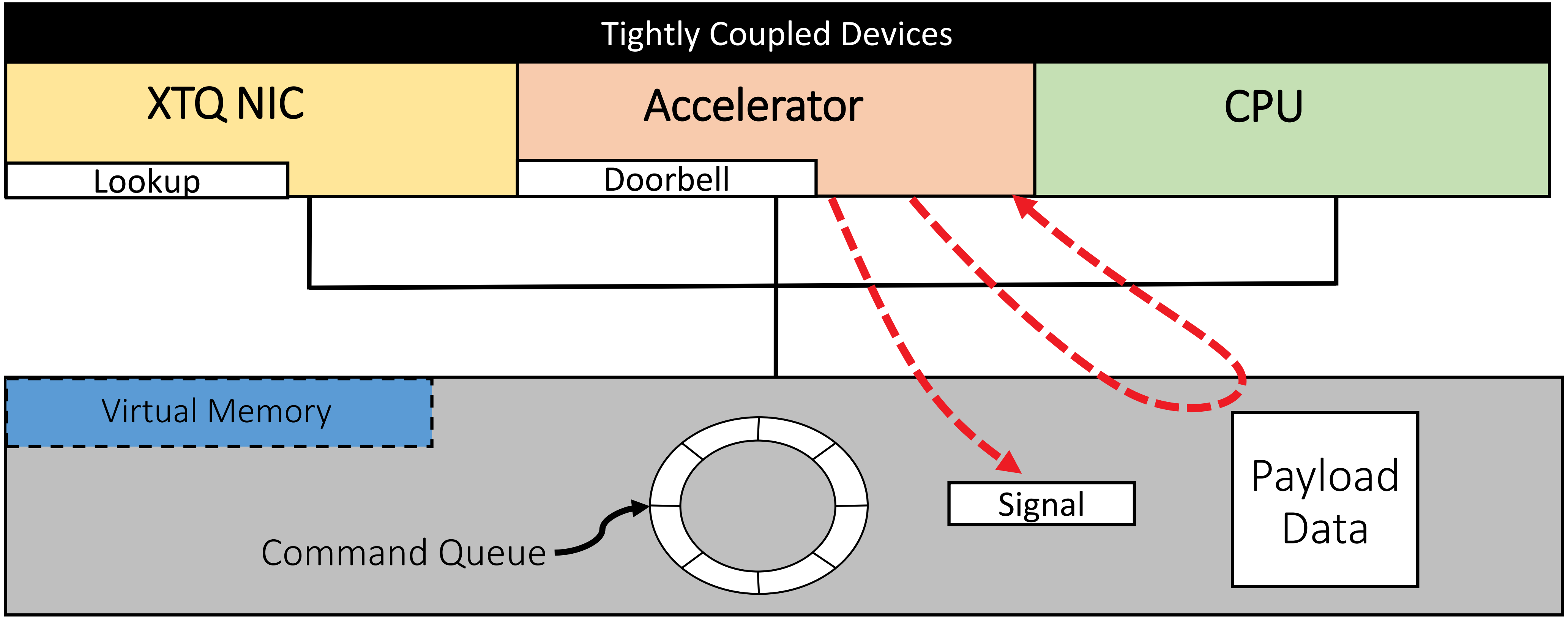
- NIC notifies the accelerator using memory-mapped doorbell
- Accelerator reads command packet



TARGET-SIDE XTQ PUT

XTQ ARCHITECTURE

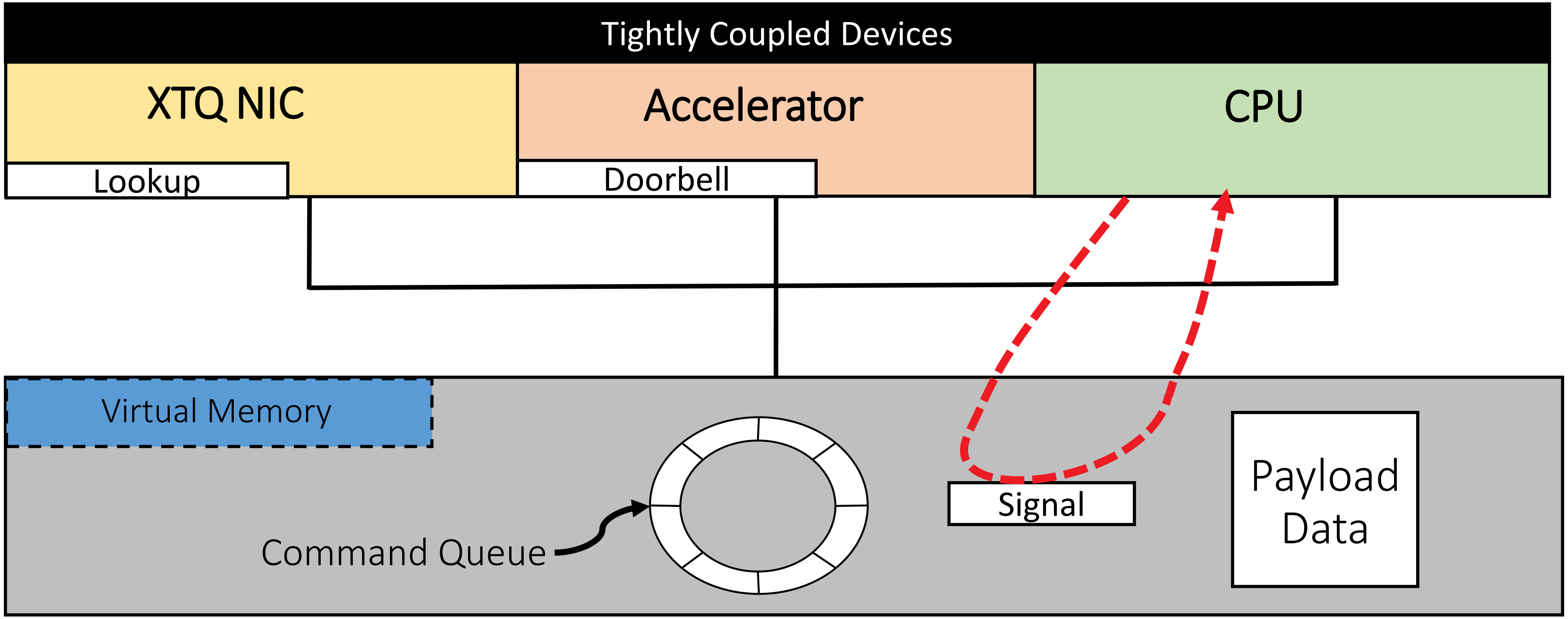
- Accelerator reads transferred data
- Accelerator writes shared memory completion signal



TARGET-SIDE XTQ PUT

XTQ ARCHITECTURE

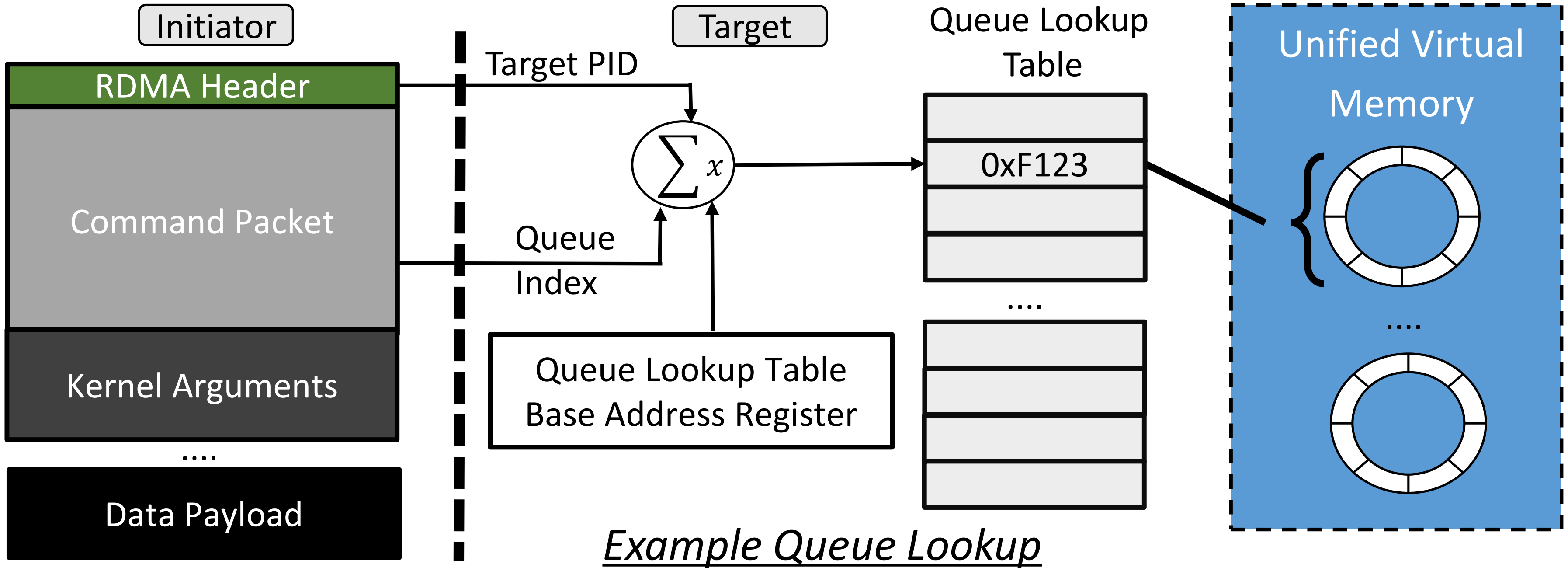
- CPU reads shared memory completion signal



CHALLENGES

XTQ ARCHITECTURE

- Address Translation?
  - 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



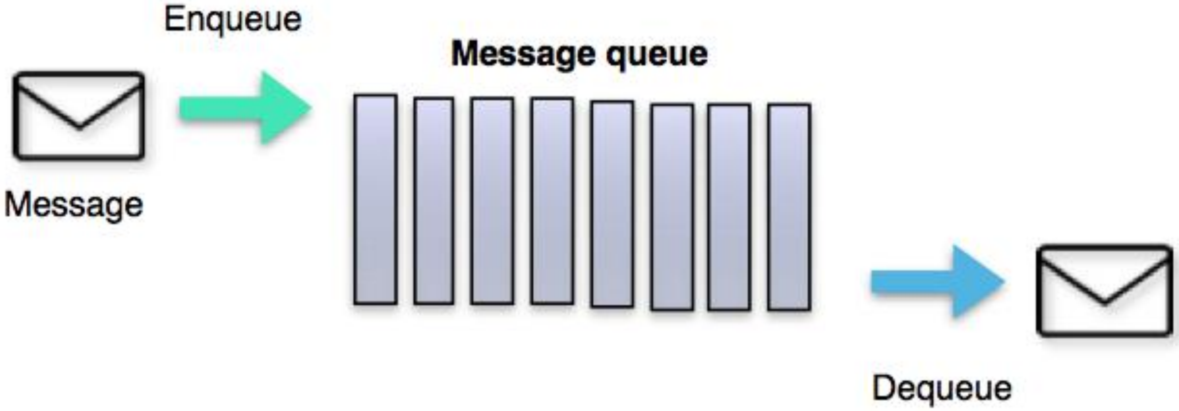
Example Queue Lookup

CHALLENGES

XTQ ARCHITECTURE

◢Flow Control? Security?

- XTQ data structures need flow control and security
- Low-level networking APIs provide mechanisms to support these features
- XTQ can adopt the policies of the transport it extends



XTQ RDMA EXTENSIONS

XTQ 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

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

XTQ-Enhanced RDMA Put Operation

Additional XTQ Fields
Remote Queue Index
Remote Function/Kernel Index
HSA-style command packet
Kernel/Function Launch Parameters

XTQ Rewrite Registration API

- 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
  - HSA-style completion signal VA

EXPERIMENTAL FRAMEWORK

RESULTS

- All data collected in gem5[6]
  - System call emulation mode (no OS)
  - AMD GPU model[7]
  - Full Support for HSA
  - Tightly coupled system

- Portals 4-based NIC model[8]
  - Low-level RDMA network programming API currently supported by:
    - MPICH, Open MPI, GASNet, Berkeley UPC, GNU UPC, and others
  - XTQ implemented as an extension of the Portals 4 remote Put operation

CPU and Memory Configuration	
CPU Type	8-wide OOO, 4Ghz, 8 cores
I,D-Cache	64K, 2-way, 1 cycle
L2-Cache	2MB, 8-way, 4 cycles
L3-Cache	16MB, 16-way, 20 cycles
DRAM	DDR3, 4 Channels, 800MHz

GPU Configuration	
GPU Type	1 Ghz, 24 Compute Units
D-Cache	16kB, 64B line, 16-way, 4 cycles
I-Cache	32kB, 64B line, 8-way, 4 cycles
L2-Cache	768kB, 64B line, 16-way, 24 cycles

NIC Configuration	
Link Speed	100ns/ 100Gbps
Network API	Portals 4
Topology	Star

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," SIGARCH Comput. Archit. News, pp. 1–7, 2011.

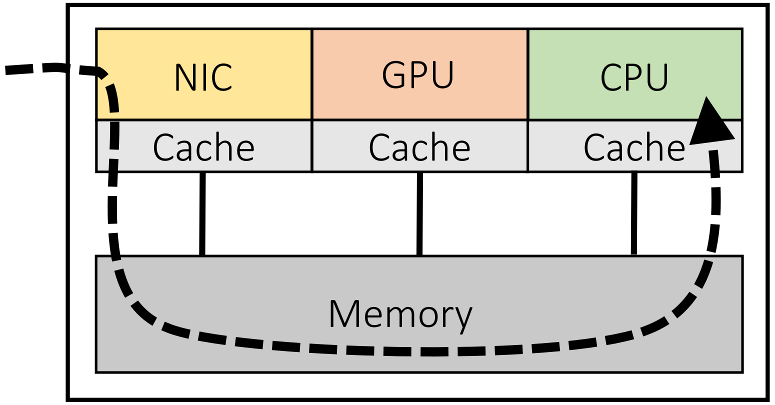
[7] AMD. (2015) The AMD gem5 APU simulator: Modeling heterogeneous systems in gem5. http://gem5.org/GPU_Models.

[8] Sandia National Laboratories, "The Portals 4.0.2 network programming interface," http://www.cs.sandia.gov/Portals/portals402.pdf, 2014.

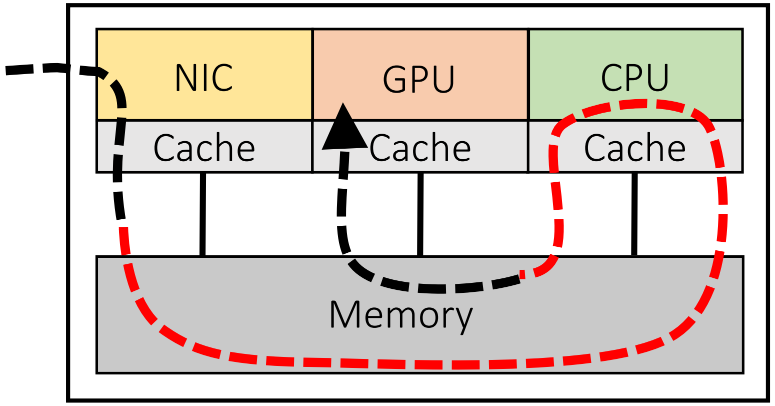
SYSTEM MODELS

RESULTS

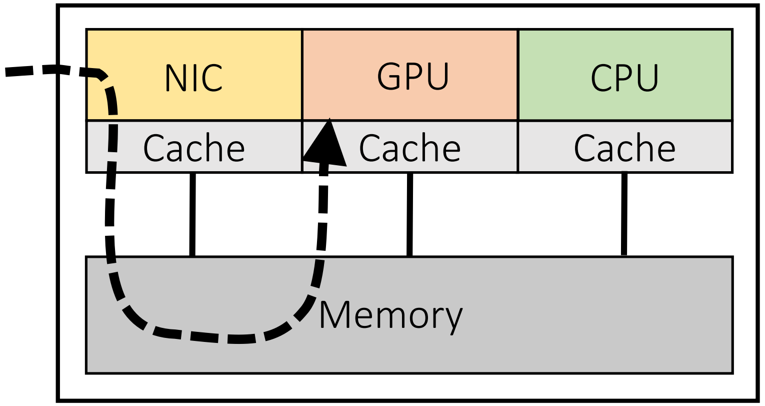
- Target-side tasking control path:



CPU



HSA



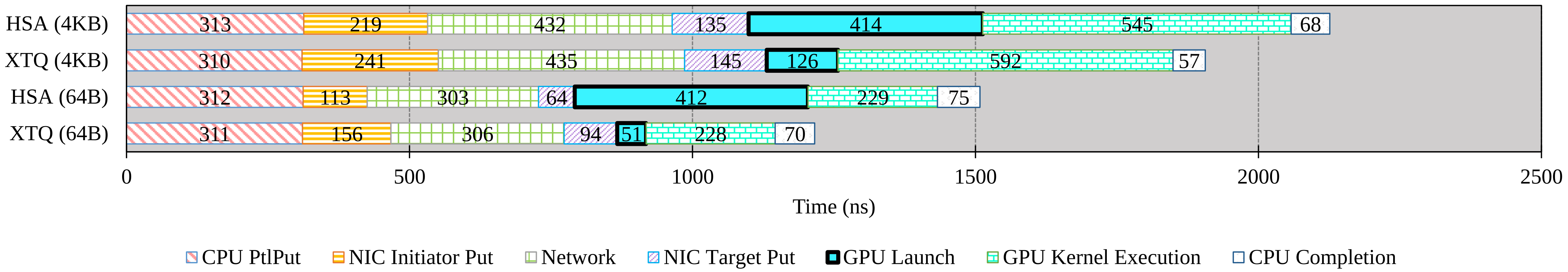
XTQ

- CPU: CPU performs computation
- HSA: GPU performs computation through CPU-side HSA Runtime
- XTQ: GPU performs computation using XTQ NIC-to-accelerator tasking

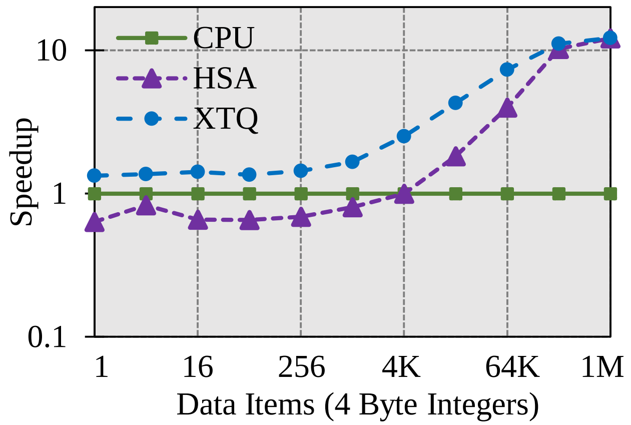
MICROBENCHMARKS

RESULTS

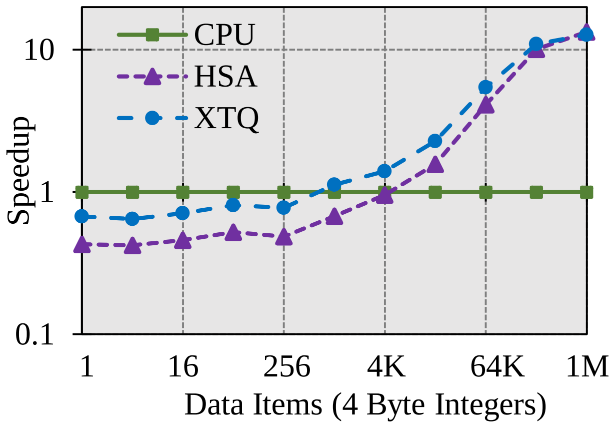
Latency Decomposition



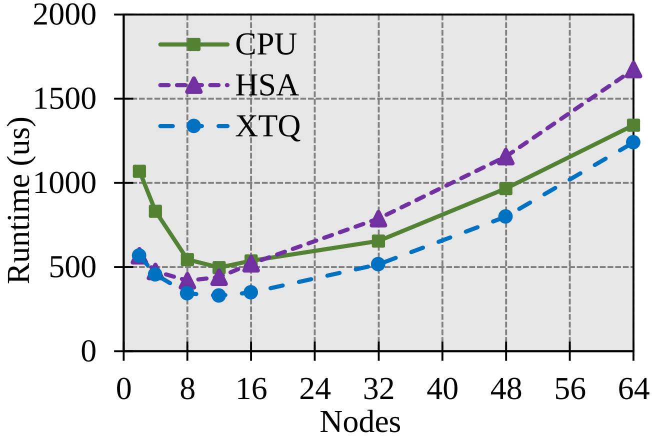
MPI Accumulate



MPI Reduce

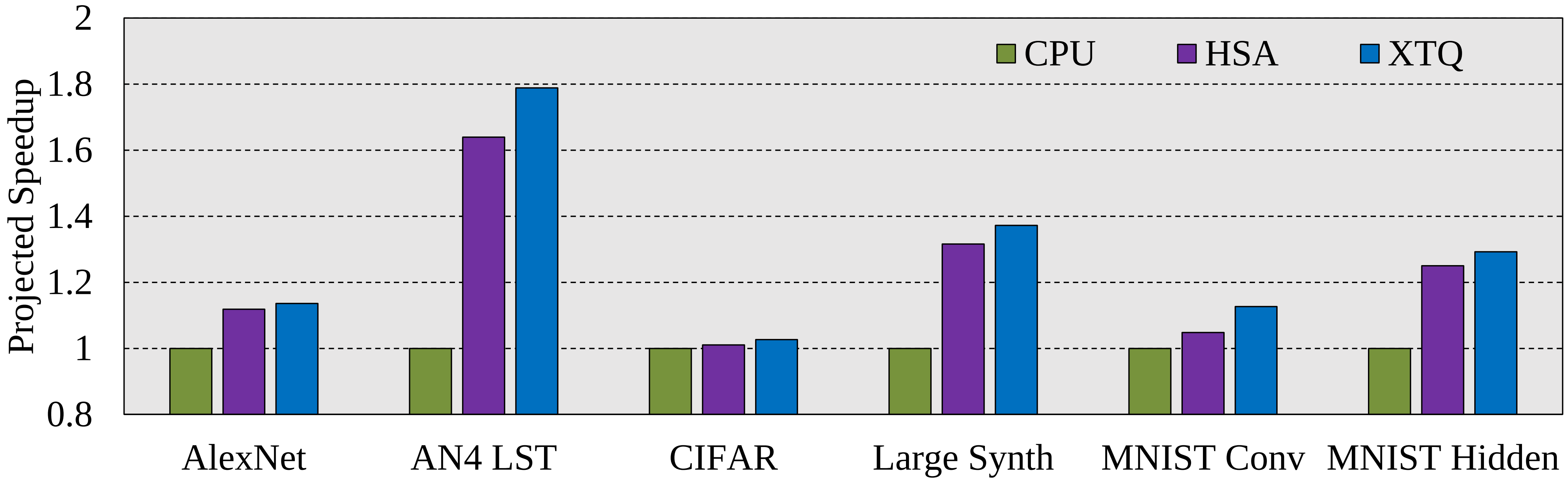


MPI Allreduce



MACHINE LEARNING BENCHMARKS
RESULTS

- Benchmarks from the Microsoft Cognitive Toolkit (CNTK)[9]
- Results are projected using real runs augmented with Allreduce() speed-up numbers from the simulator



[9] Agarwal, et.al., An introduction to computational networks and the Computational Network Toolkit," Microsoft, Technical Report, 2014.

SUMMARY

CONCLUSION

- XTQ uses an HSA-compliant, RDMA-capable NIC to provide an **active messaging** framework for all devices in distributed systems

- XTQ **reduces the launch latency** for remote GPU task invocation
  - Tasks are directly scheduled on the GPU by the NIC using shared memory queues

- XTQ **removes the message processing on the CPU** for GPU-destined tasks
  - The CPU is free to perform more useful computation

THANK YOU!

Michael.Lebeane@amd.com

mlebeane@utexas.edu

QUESTIONS?

REFERENCES

[1] TOP500.org, “Highlights – June 2016,” http://www.top500.org/lists/2016/06/highlights, 2016.

[2] Nvidia, “GPU-Accelerated Applications,” http://www.nvidia.com/content/gpu-applications/pdf/gpu-apps-catalog-mar2015.pdf, 2016.

[3] Mellanox, “Mellanox GPUDirect RDMA user manual,” http://www.mellanox.com/related-docs/prod_software/Mellanox_GPUDirect_User_Manual_v1.2.pdf, 2015.

[4] HSA Foundation, “HSA platform system architecture specification 1.0,” http://www.hsafoundation.com/standards, 2015.

[5] T. Eicken, D. Culler, S. Goldstein, and K. Schauser, “Active messages: A mechanism for integrated communication and computation,” in Int. Symp. on Computer Architecture (ISCA), 1992, pp. 256–266.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, “The gem5 simulator,” SIGARCH Comput. Archit. News, pp. 1–7, 2011.

[7] AMD. (2015) The AMD gem5 APU simulator: Modeling heterogeneous systems in gem5. http://gem5.org/GPU_Models.

[8] Sandia National Laboratories, “The Portals 4.0.2 network programming interface,” http://www.cs.sandia.gov/Portals/portals402.pdf, 2014.

[9] A. Agarwal, E. Akchurin, C. Basoglu, G. Chen, S. Cyphers, J. Droppo, A. Eversole, B. Guenter, M. Hillebrand, T. R. Hoens, X. Huang, Z. Huang, V. Ivanov, A. Kamenev, P. Kranen, O. Kuchaiev, W. Manousek, A. May, B. Mitra, O. Nano, G. Navarro, A. Orlov, H. Parthasarathi, B. Peng, M. Radmilac, A. Reznichenko, F. Seide, M. L. Seltzer, M. Slaney, A. Stolcke, H. Wang, Y. Wang, K. Yao, D. Yu, Y. Zhang, and G. Zweig, An introduction to computational networks and the Computational Network Toolkit,” Microsoft, Technical Report, 2014.

XTQ PORTALS EXTENSIONS

XTQ API

```
XtqPut(ptl_handle_md_t md_handle,
       ptl_size_t local_offset,
       ptl_size_t length,
       ptl_handle_md_t md_handle2,
       ptl_size_t local_offset2,
       ptl_size_t length2,
       ptl_ack_req_t ack_req,
       ptl_process_t target_id,
       ptl_pt_index_t pt_index,
       ptl_match_bits_t match_bits,
       ptl_size_t remote_offset,
       void *user_ptr,
       ptl_hdr_data_t hdr_data);
);
```

XTQ Command Packet

Generic Data Buffer

- Primary operation is XtqPut
- Same as regular PtlPut except:
  - Two send buffers
  - One for generic data
  - One for XTQ command packet

XTQ PORTALS EXTENSIONS

XTQ API

- Three functions populate the target-side Rewrite Tables

- One of each for queues, kernels descriptors, and function descriptors

```
int XtqRegisterQueue(int id,
                     hsa_queue_t* q);

int XtqRegisterFunction(int id,
                        void *buffer_pointer,
                        void (*function) (uint64_t,uint64_t,uint64_t,uint64_t);

int XtqRegisterKernel(int id,
                      void* buffer_pointer,
                      int kernarg_size,
                      int rewrite_offset,
                      hsa_signal_t completion_signal,
                      uint64_t kernel);
```

SAMPLE CODE

XTQ API

Initiator

```
// Initialize RDMA comm layer
int rank = RdmaInit();
int index = 42;
// Construct XTQ command
void *cmd = ConstructCmd(CMD_SIZE, index);
// Post initialization sync with target
ExecutionBarrier();
// Launch on remote GPU using XTQ
XtqPut(TARGET, cmd, CMD_SIZE,
       payload, BUFFER_SIZE);
```

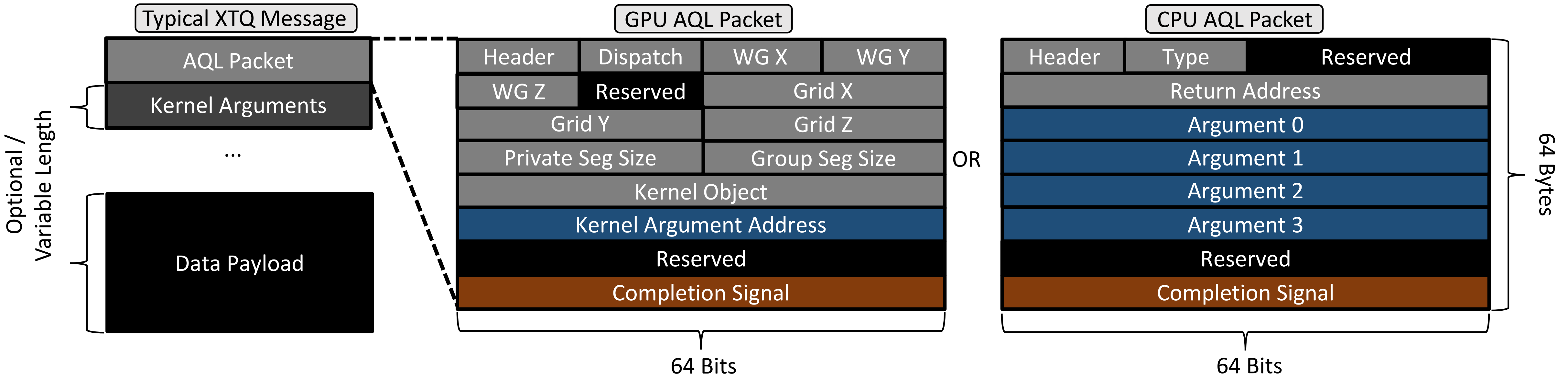
Target

```
// Initialize RDMA comm layer
int rank = RdmaInit();
int index = 42;
// Post receive buffer
RdmaPostBuffer(recv_buf);
// Initialize HSA CPU Runtime
TaskingInit(&signal, &kernel, &queue);
// Register Kernel/Queues
XtqRegisterKernel(signal, kernel, index);
XtqRegisterQueue(queue, index);
// Post initialization sync with initiator
ExecutionBarrier();
// Wait for GPU to complete task
SignalWait(signal);
```

XTQ PACKET FORMAT

XTQ ARCHITECTURE

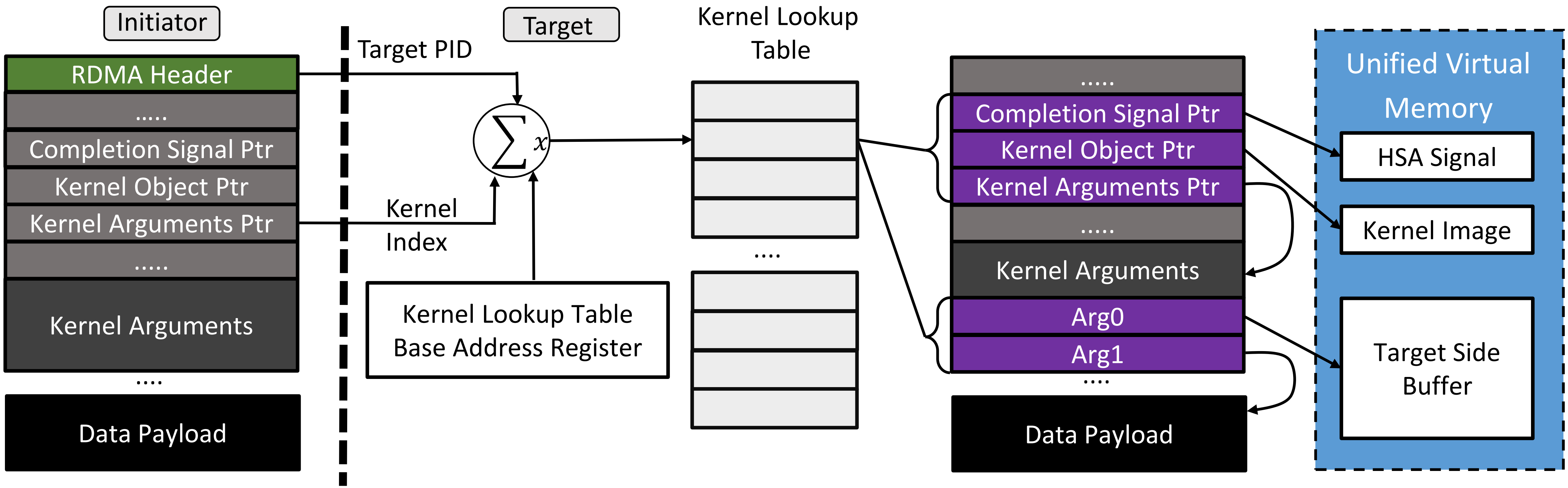
- XTQ command packets are HSA AQL packets
  - Currently CPU and GPU format is supported
  - Optional Fields:
    - Kernel Arguments
    - Data Payload



XTQ REWRITE FUNCTIONALITY

XTQ ARCHITECTURE

- Initiator does not know about target side resources needed for tasking
- Several fields populated by the target-side NIC using ***coordinated indices*** specified by the initiator
- Rewrite tables are populated by the target-side XTQ Library



PORTALS4

BACKGROUND

- **Low-level network interface** designed to provide RDMA primitives for higher level network protocols

- Software reference implementation of Portals 4 using **InfiniBand** is publicly available

- Currently supported by:
  - MPICH, Open MPI, GASNet, Berkeley UPC, GNU UPC, and others

- XTQ leverages Portals for its basic RDMA features

ACTIVE MESSAGES

BACKGROUND

- Messages that specify **computation**
- Each message contains enough info to perform some action
- Message contains pointer to code
- Input data is optionally transmitted