GPUrdma: GPU-side library for high performance networking from GPU kernels

Feras Daoud

Technion – Israel Institute of Technology

Mark Silberstein

Technion

Amir Watad

Technion

Agenda

1 Introduction

2 InfiniBand Background

3 GPUrdma

4 GPUrdma Evaluation

5 GPI2

What

- A GPU-side library for performing RDMA directly from GPU kernels

Why

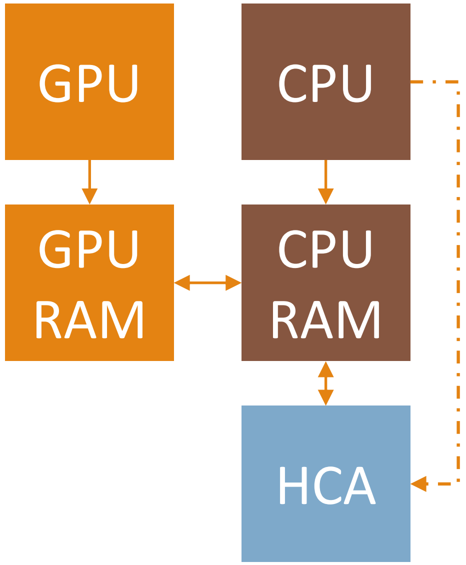
- To improve communication performance between distributed GPUs

Results

- 5 µsec GPU-to-GPU communication latency and up to 50 Gbps transfer bandwidth

Evolution of GPU-HCA interaction:

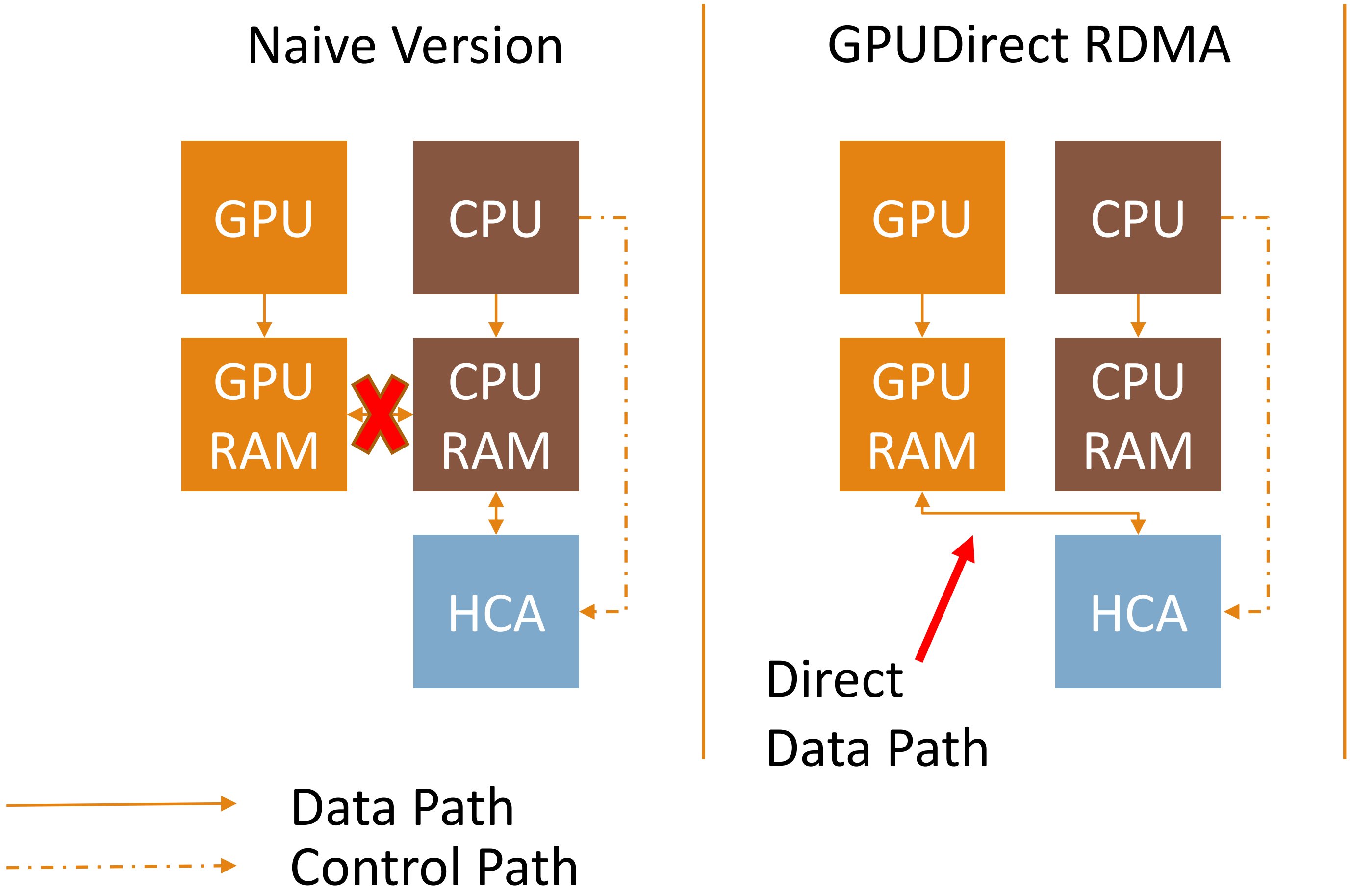
Naive Version



Data Path

Control Path

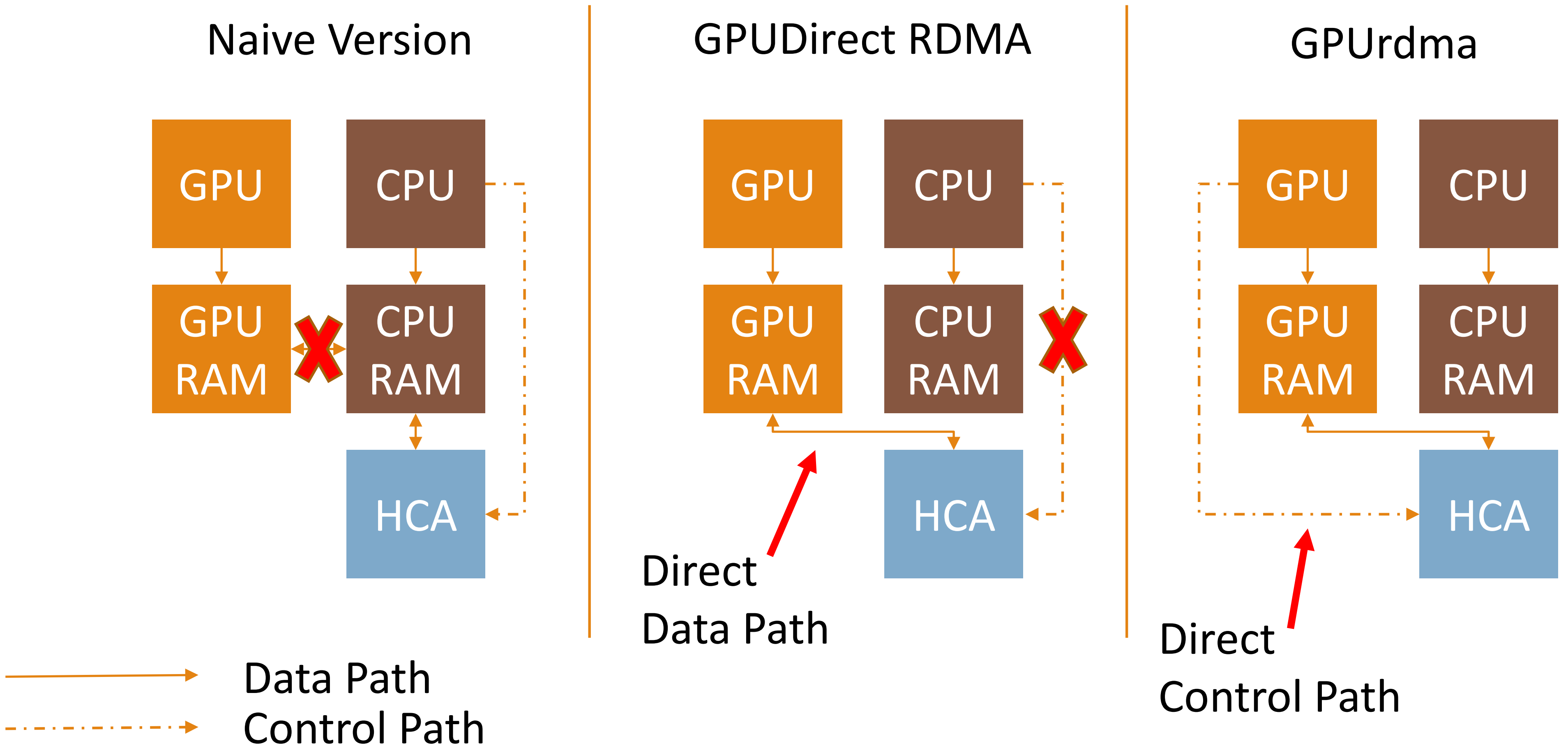
Evolution of GPU-HCA interaction:



Naive Version

GPUDirect RDMA

Evolution of GPU-HCA interaction:



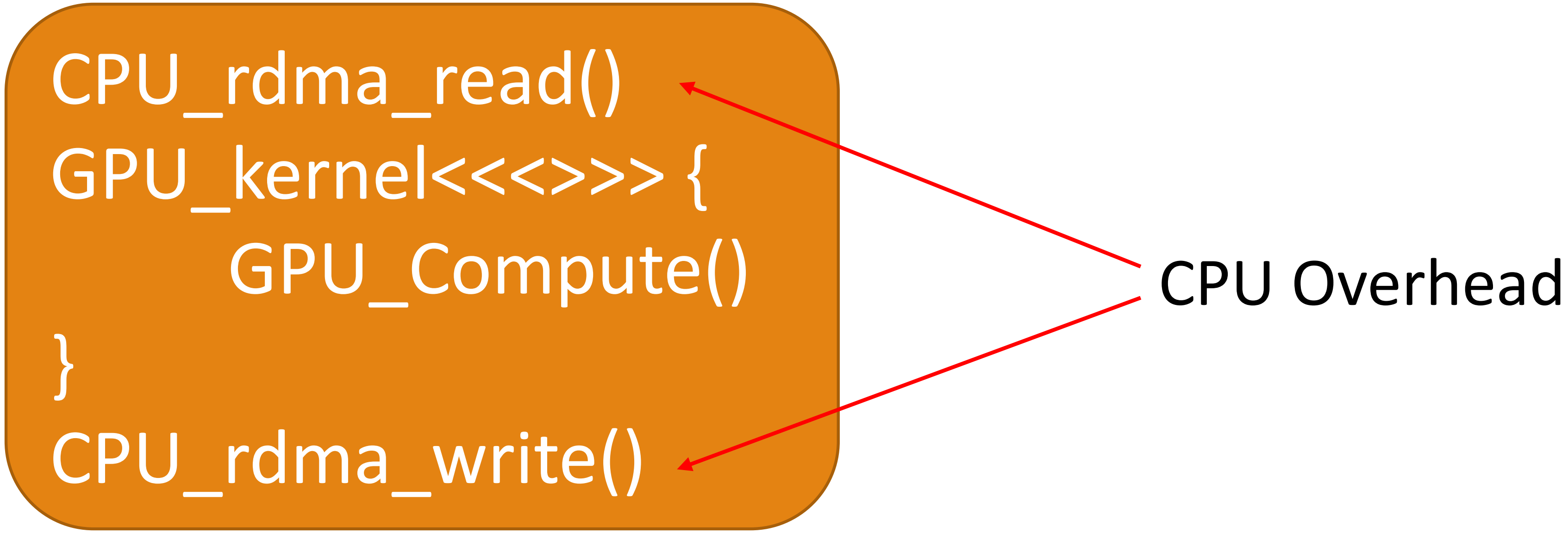
Motivations

GPUDirect RDMA Node

```
CPU_rdma_read()
GPU_kernel<<<>>> {
        GPU_Compute()
}
CPU_rdma_write()
```

Motivations

GPUDirect RDMA Node

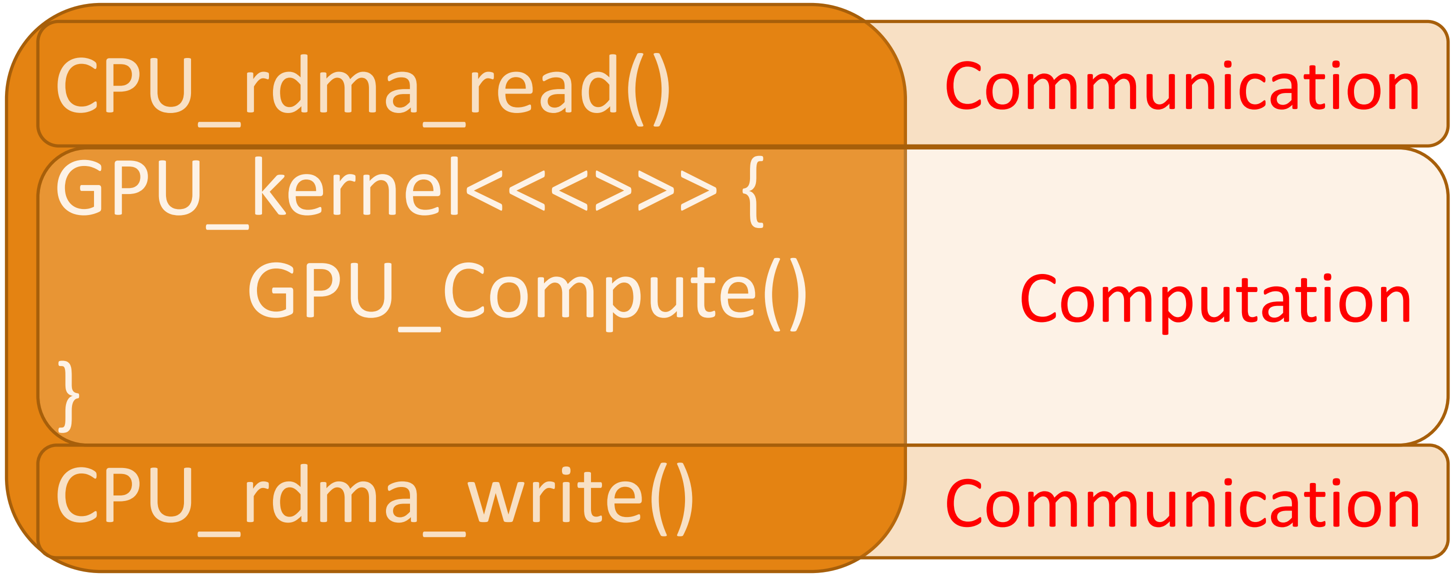


```
CPU_rdma_read()
GPU_kernel<<<>>> {
        GPU_Compute()
}
CPU_rdma_write()
```

CPU Overhead

Motivations

GPUDirect RDMA Node



```
CPU_rdma_read()
GPU_kernel<<<>>> {
        GPU_Compute()
}
CPU_rdma_write()
```

Communication

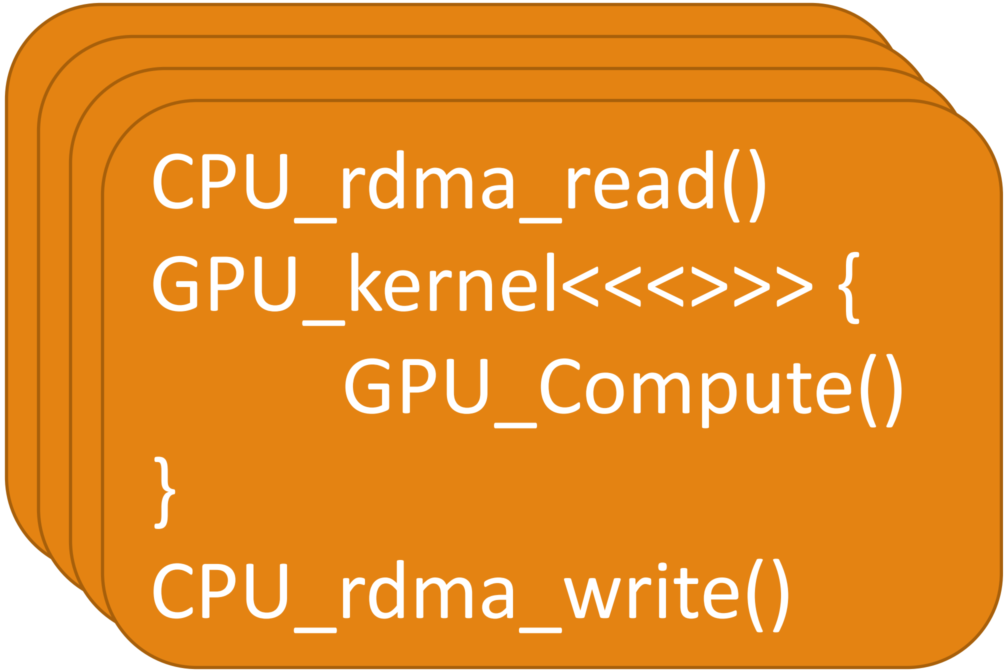
Computation

Communication

Bulk-synchronous design and explicit pipelining

Motivations

GPUDirect RDMA Node



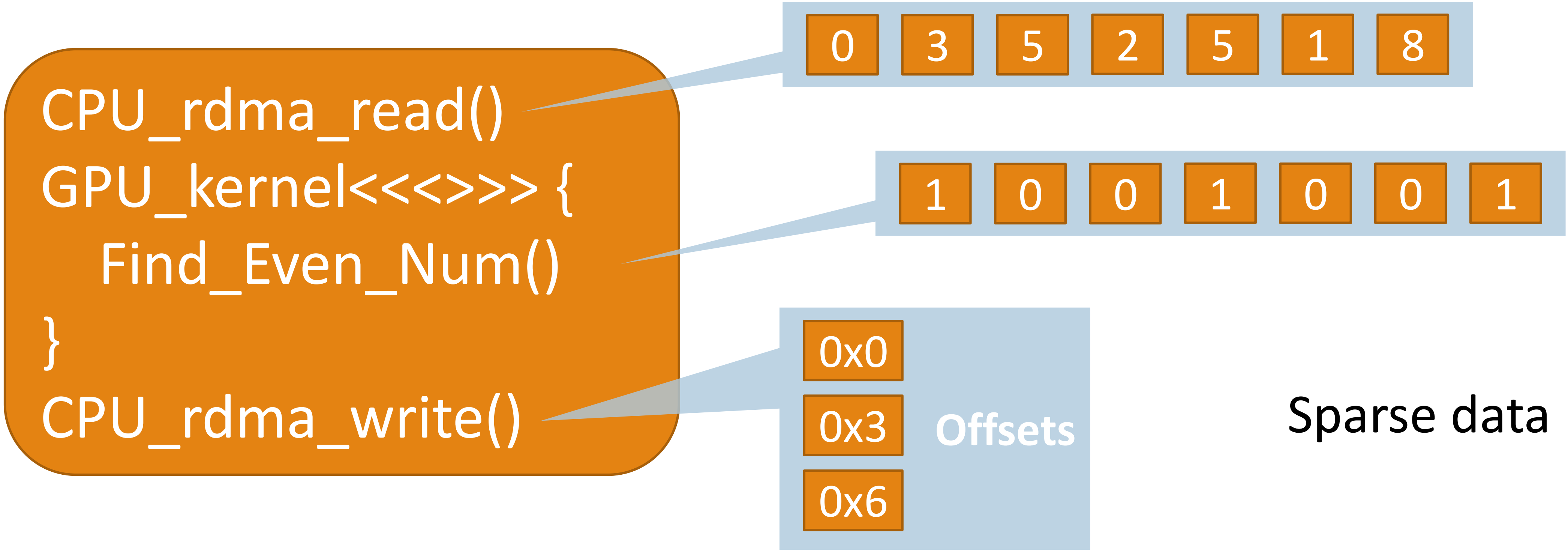
```
CPU_rdma_read()
GPU_kernel<<<>>> {
        GPU_Compute()
}
CPU_rdma_write()
```

Multiple GPU kernel invocations

1. kernel calls overhead
2. Inefficient shared memory usage

Motivations

GPUDirect RDMA Node



```
CPU_rdma_read()
GPU_kernel<<<>>> {
   Find_Even_Num()
}
CPU_rdma_write()
```

0	3	5	2	5	1	8

1	0	0	1	0	0	1

Offsets
0x0
0x3
0x6

Sparse data

GPUrdma library

GPUrdma Node

```
GPU_kernel<<<>>> {
   GPU_rdma_read()
   GPU_Compute()
   GPU_rdma_write()
}
```

- No CPU intervention
- Overlapping communication and computation
- One kernel call
- Efficient shared memory usage
- Send spare data directly from the kernel

Agenda

1 Introduction

2 InfiniBand Background

3 GPUrdma

4 GPUrdma Evaluation

5 GPI2

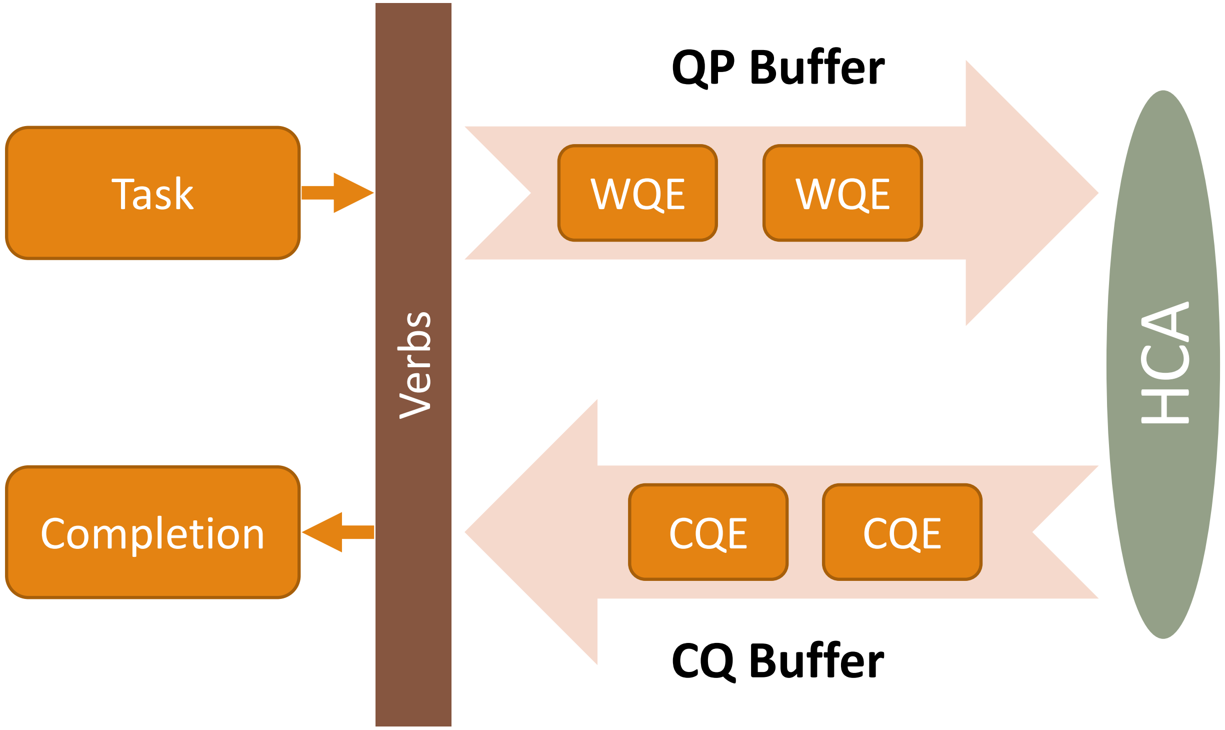
InfiniBand Background

1. Queue pair buffer (QP)
   - Send queue
   - Receive queue

2. Work Queue Element
   - Contains communication instructions

3. Completion queue buffer (CQ)
   - Contains completion elements

4. Completion queue element
   - Contains information about completed jobs

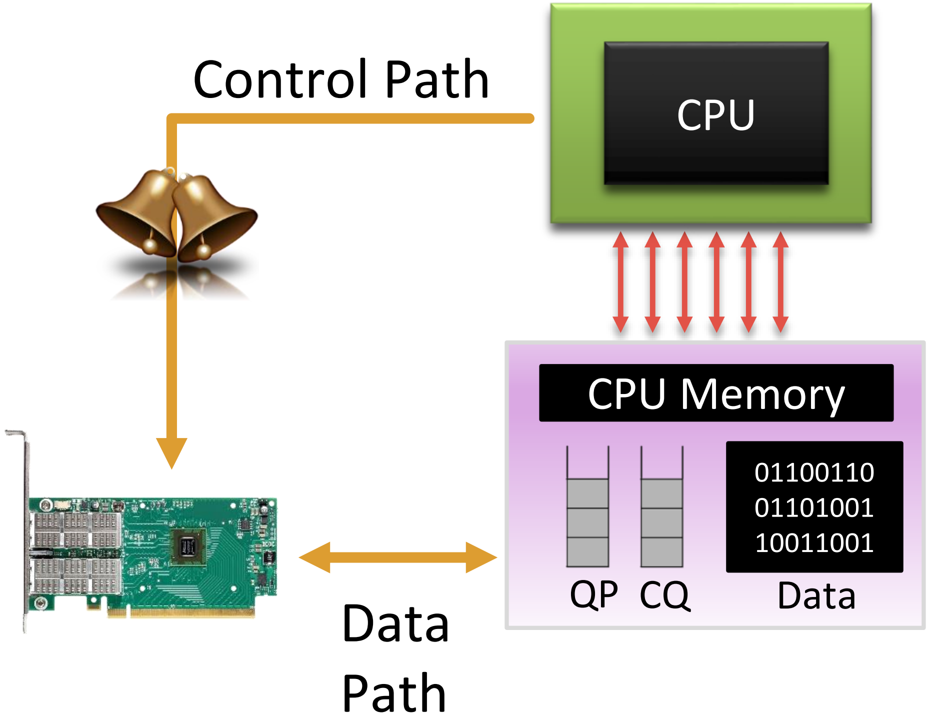


InfiniBand Background

Ring the Door-Bell to execute jobs

- MMIO address
- Informs the HCA about new jobs

1. Write work queue element to QP buffer
2. Ring the Door-Bell
3. Check completion queue element status



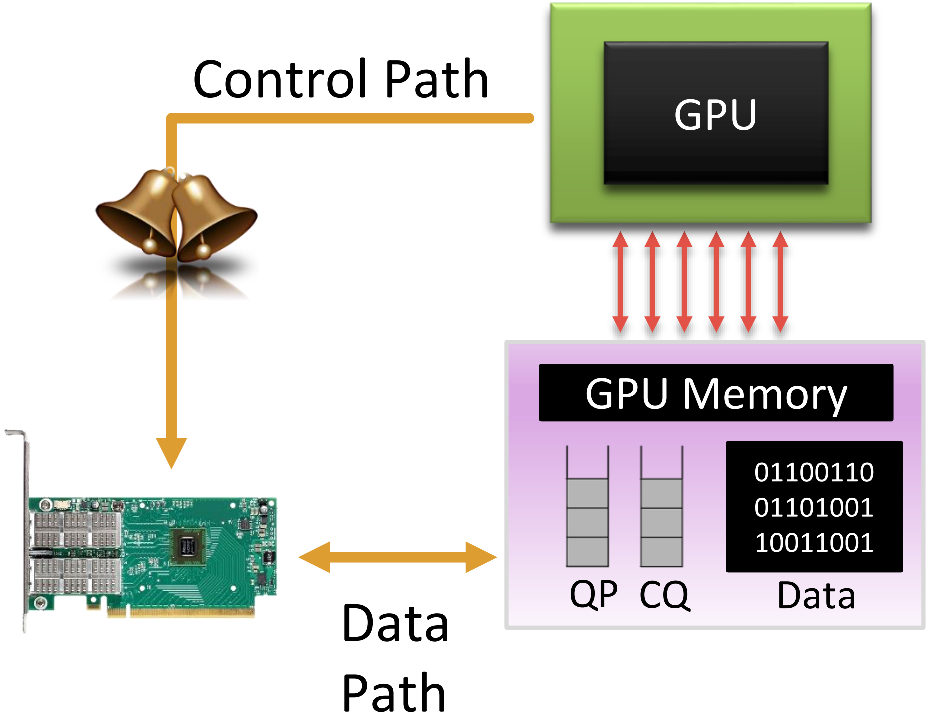
Agenda

1. Introduction
2. InfiniBand Background
3. GPUrdma
4. GPUrdma Evaluation
5. GPI2

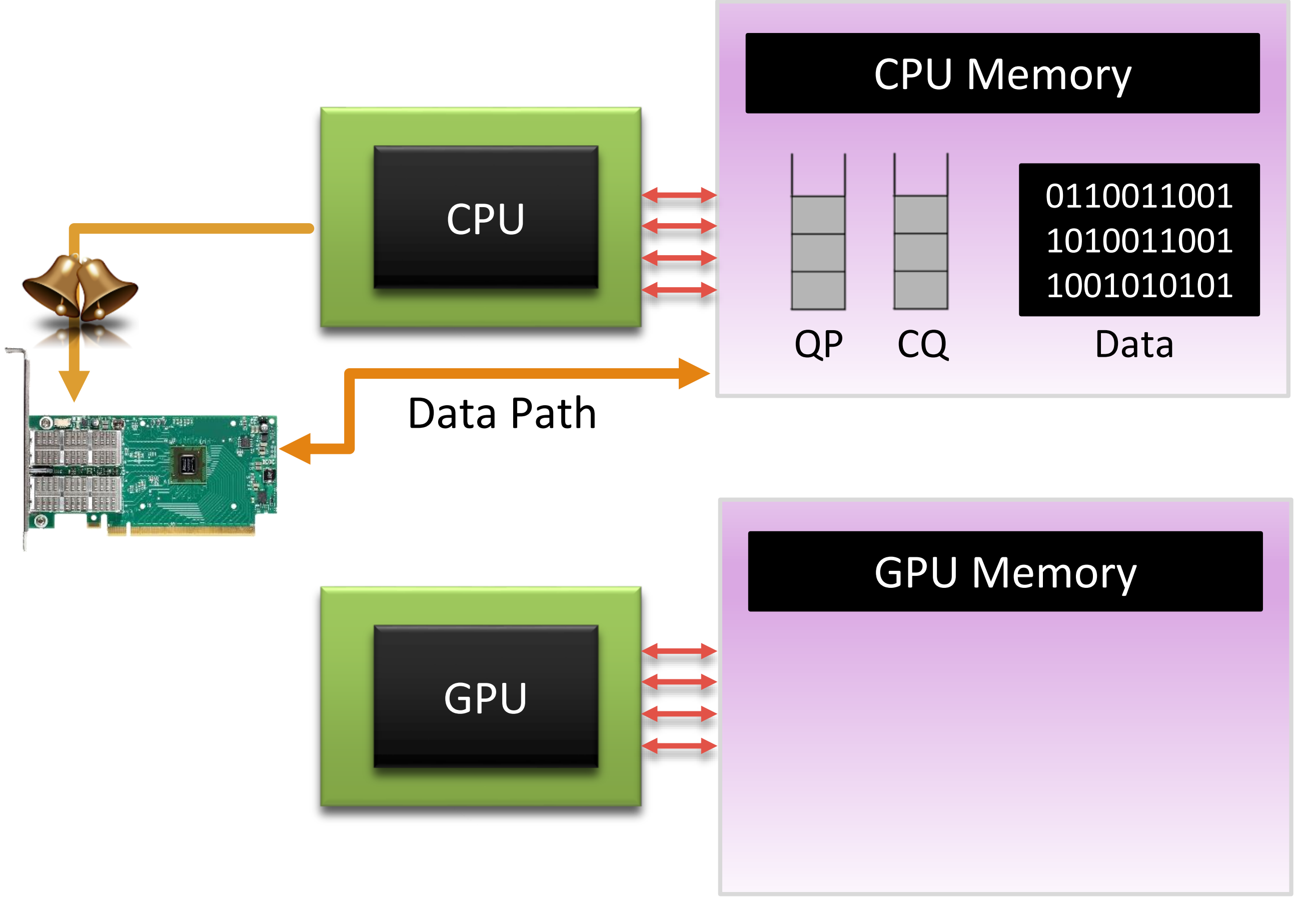
GPUrdma Node

- Direct path for data exchange
- Direct HCA control from GPU kernels
- No CPU intervention

Native GPU Node

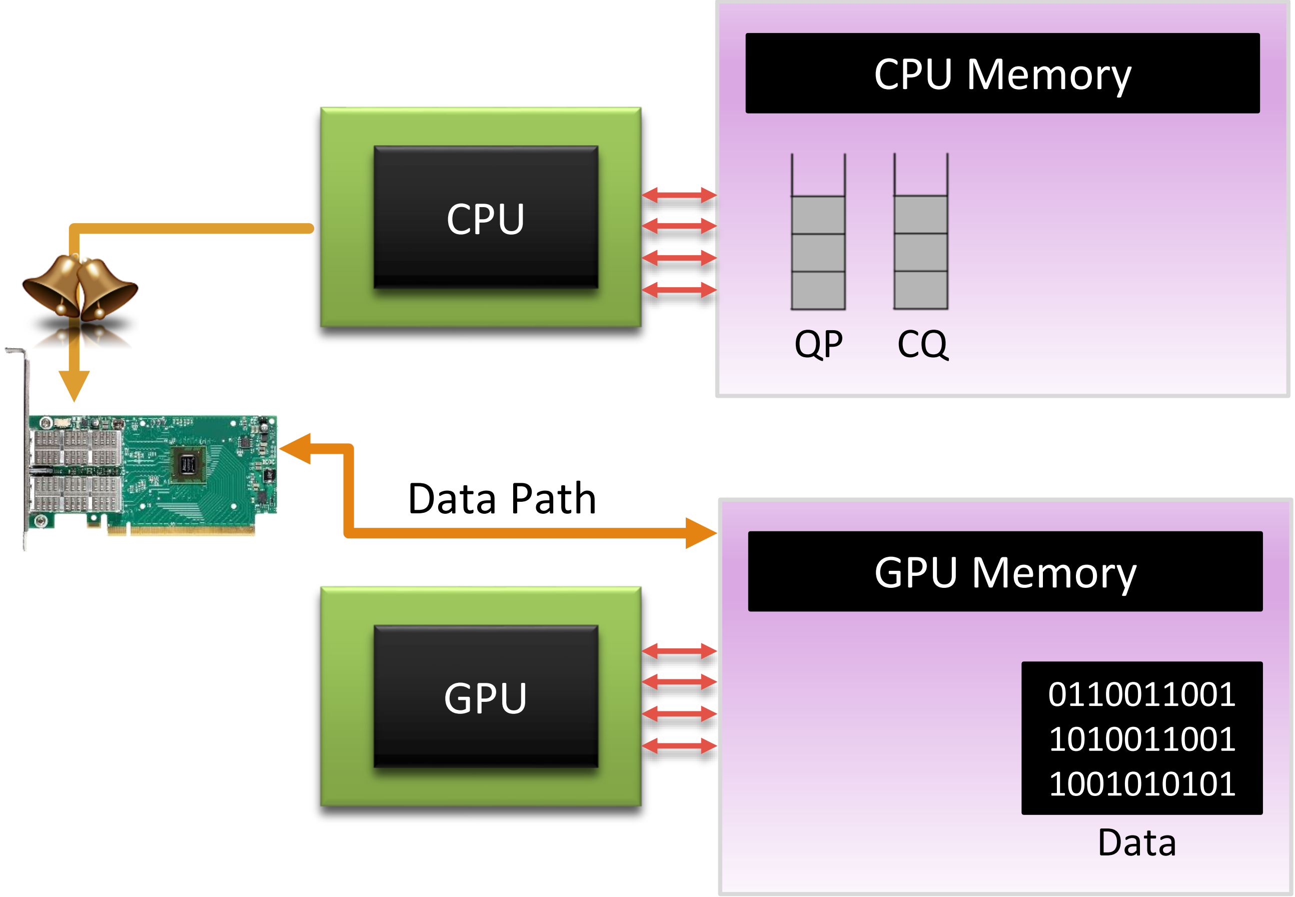


GPUrdma Implementation



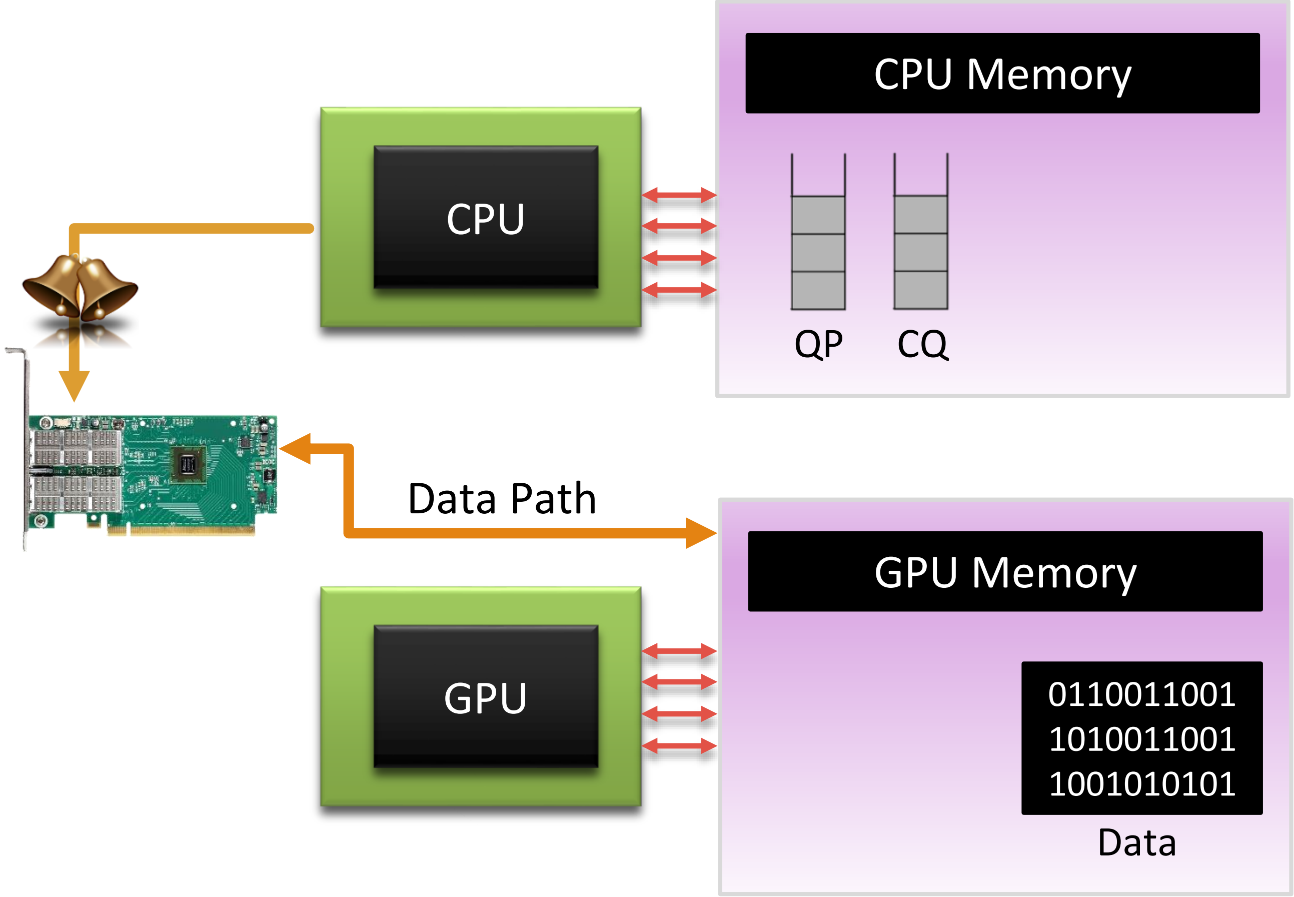
GPUrdma Implementation

Data Path - GPUDirect RDMA



GPUrdma Implementation

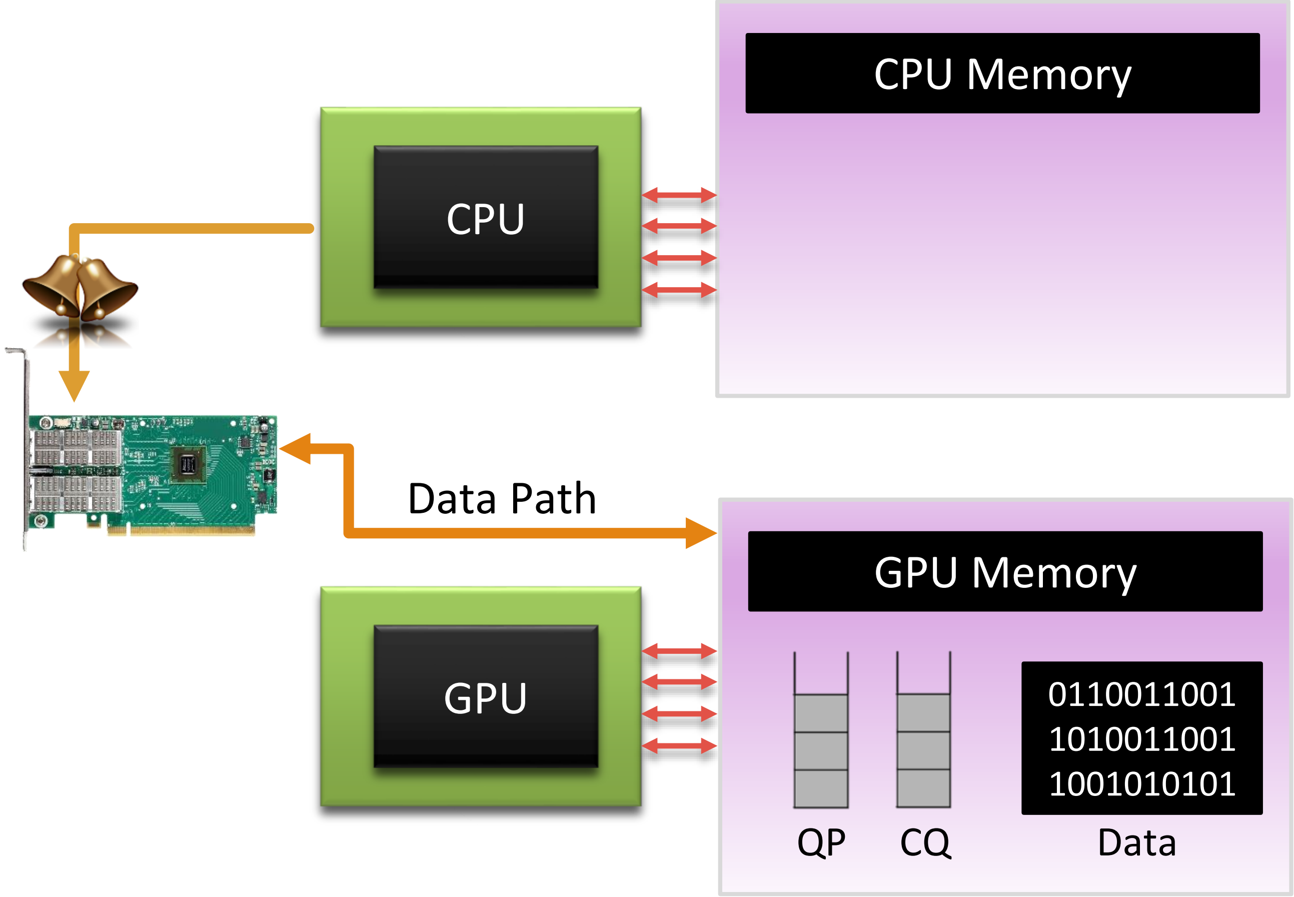
1. Move QP, CQ to GPU memory



GPUrdma Implementation

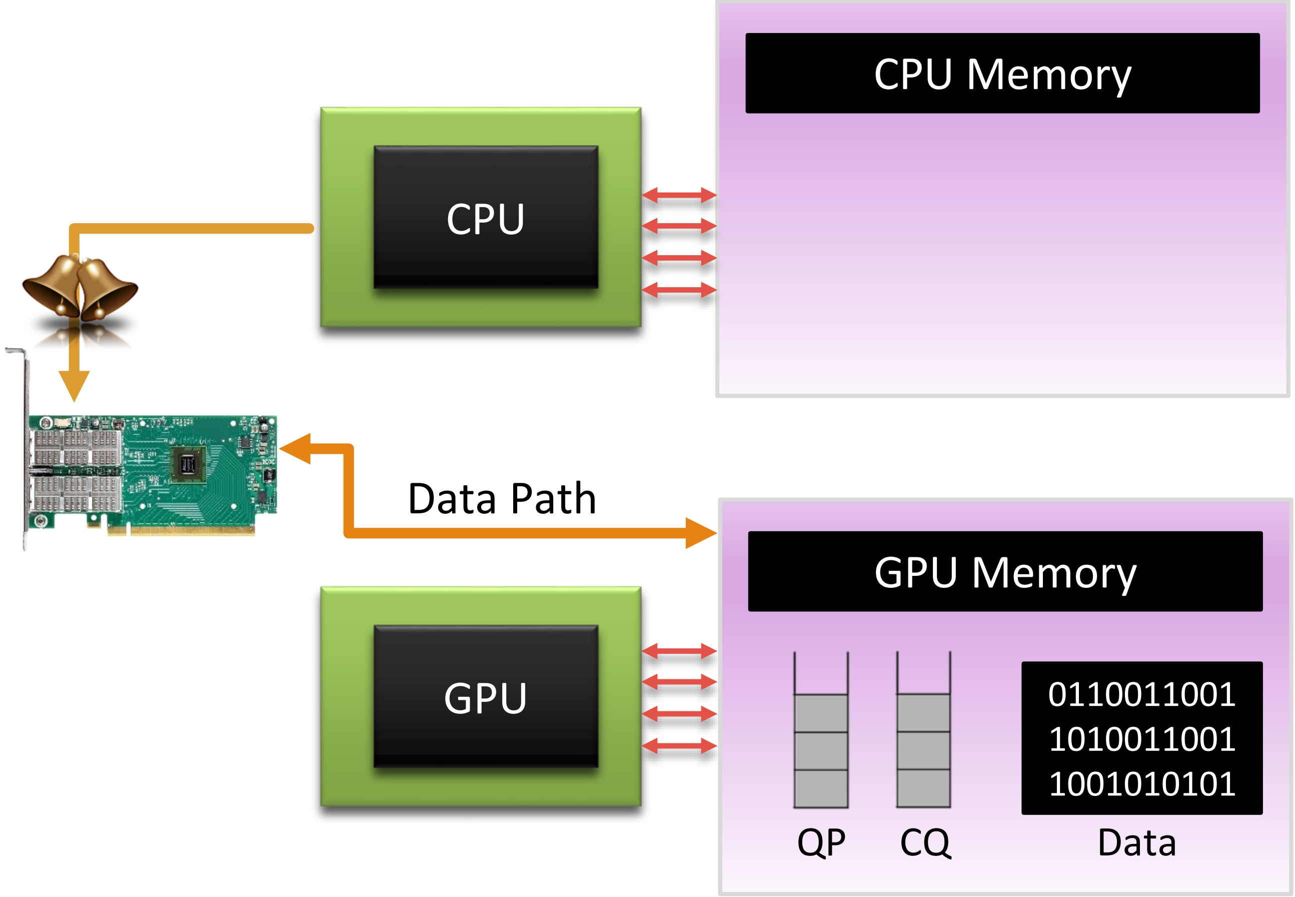
1. Move QP, CQ to GPU memory

Modify InfiniBand Verbs
- ibv_create_qp()
- ibv_create_cq()



GPUrdma Implementation

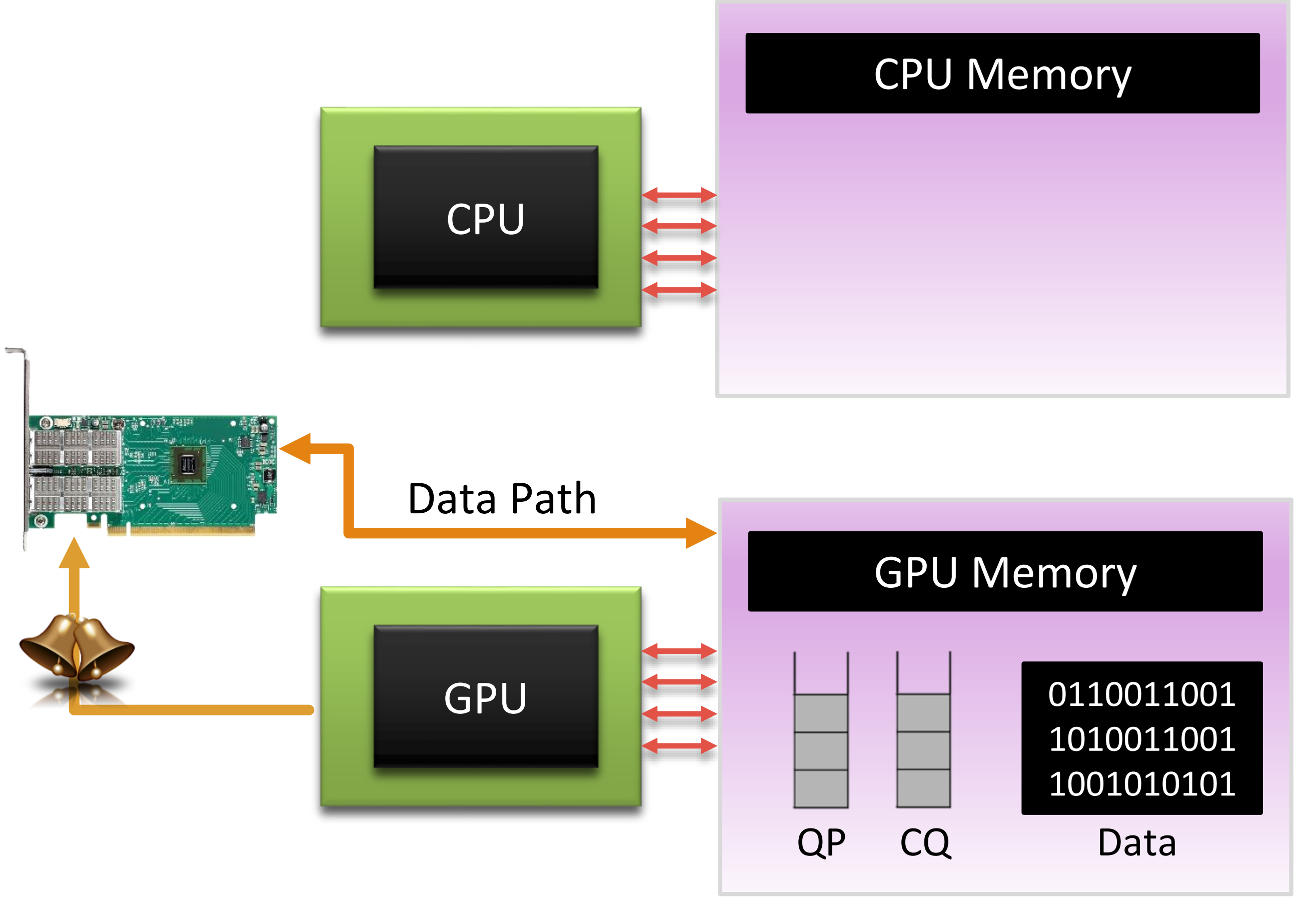
2. Map the HCA doorbell address into GPU address space



GPUrdma Implementation

2. Map the HCA doorbell address into GPU address space

Modify NVIDIA driver



Agenda

1 Introduction

2 InfiniBand Background

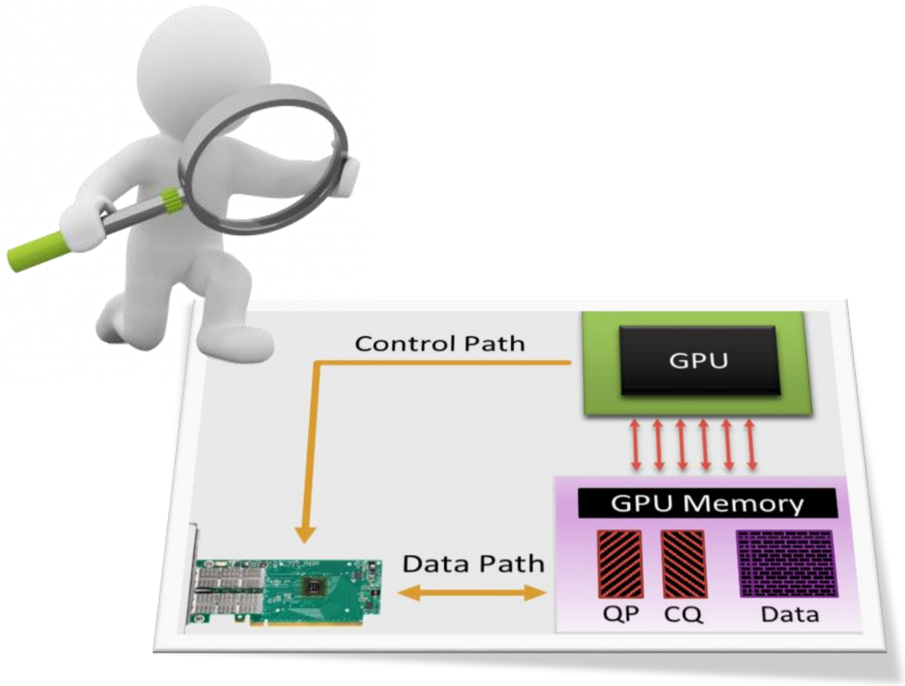
3 GPUrdma

4 GPUrdma Evaluation

5 GPI2

GPUrdma Evaluation

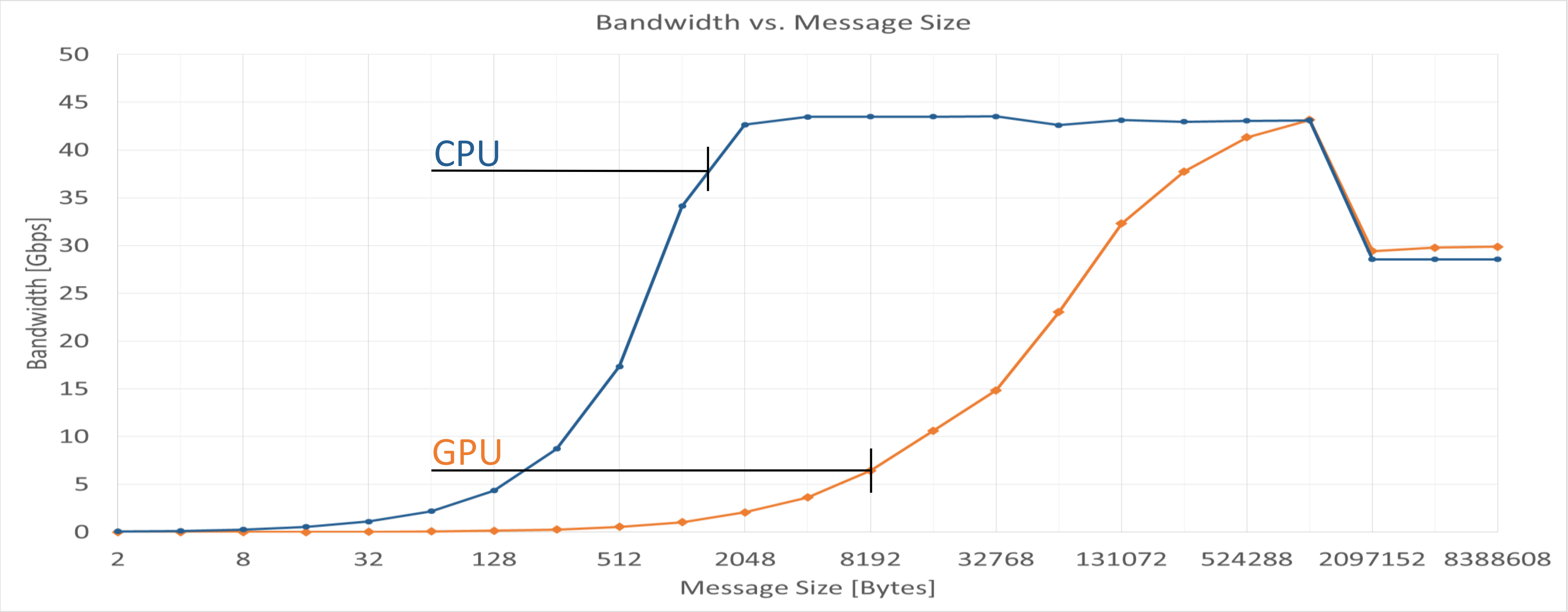
- Single QP
- Multiple QP
- Scalability - Optimal QP/CQ location



NVIDIA Tesla K40c GPU Mellanox Connect-IB HCA

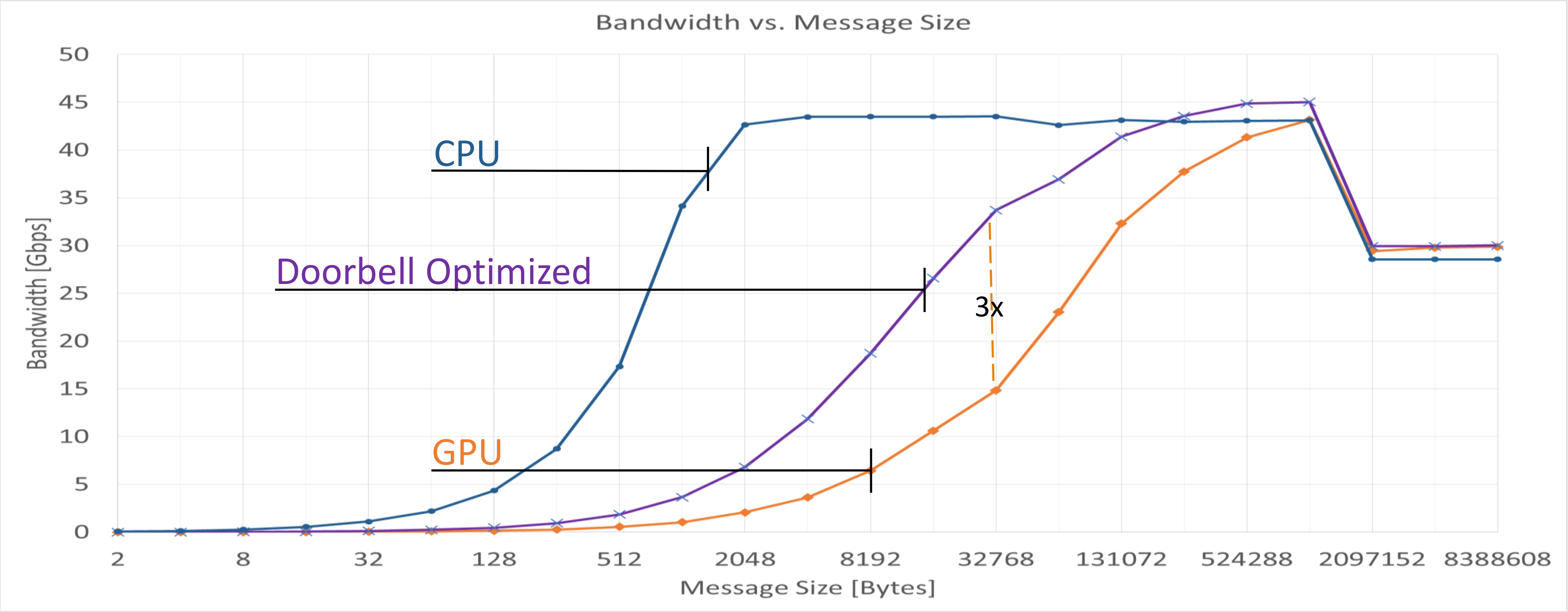
GPUrdma – 1 thread , 1 QP

- Best Performance CPU controller VS GPU controller



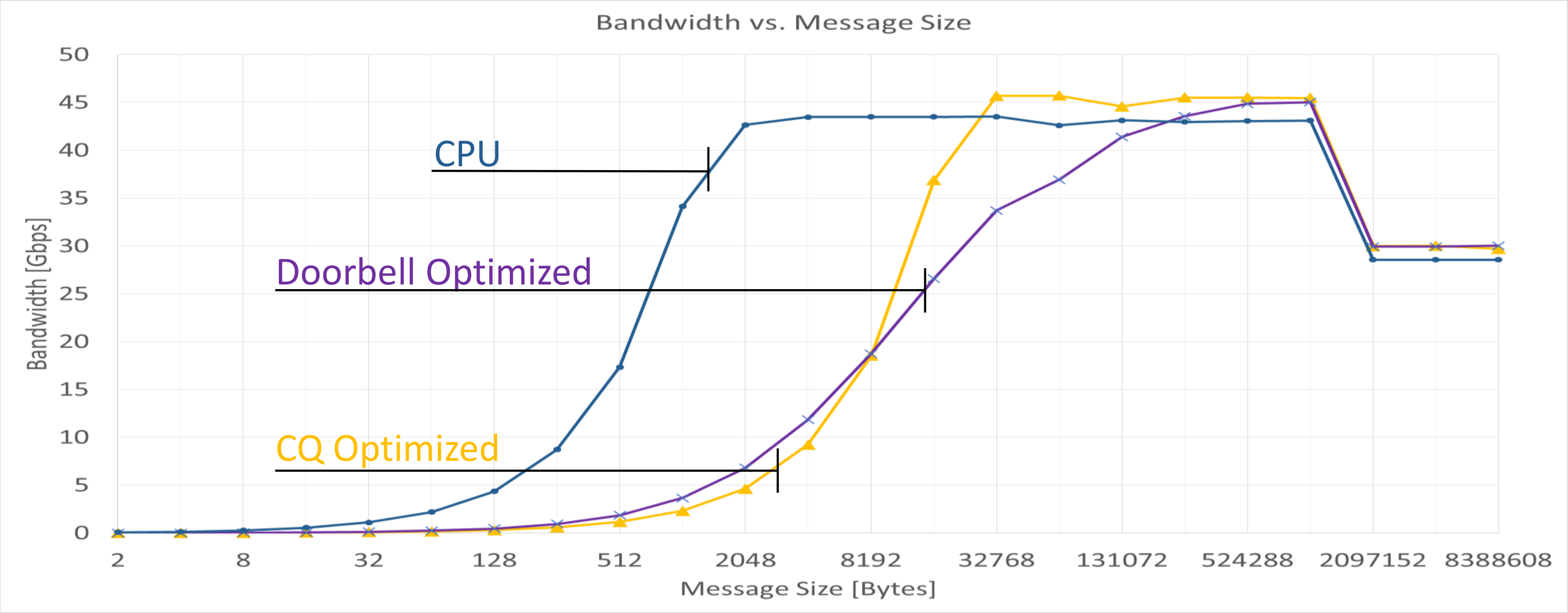
GPUrdma – 1 thread , 1 QP

- GPU controller – Optimize doorbell rings



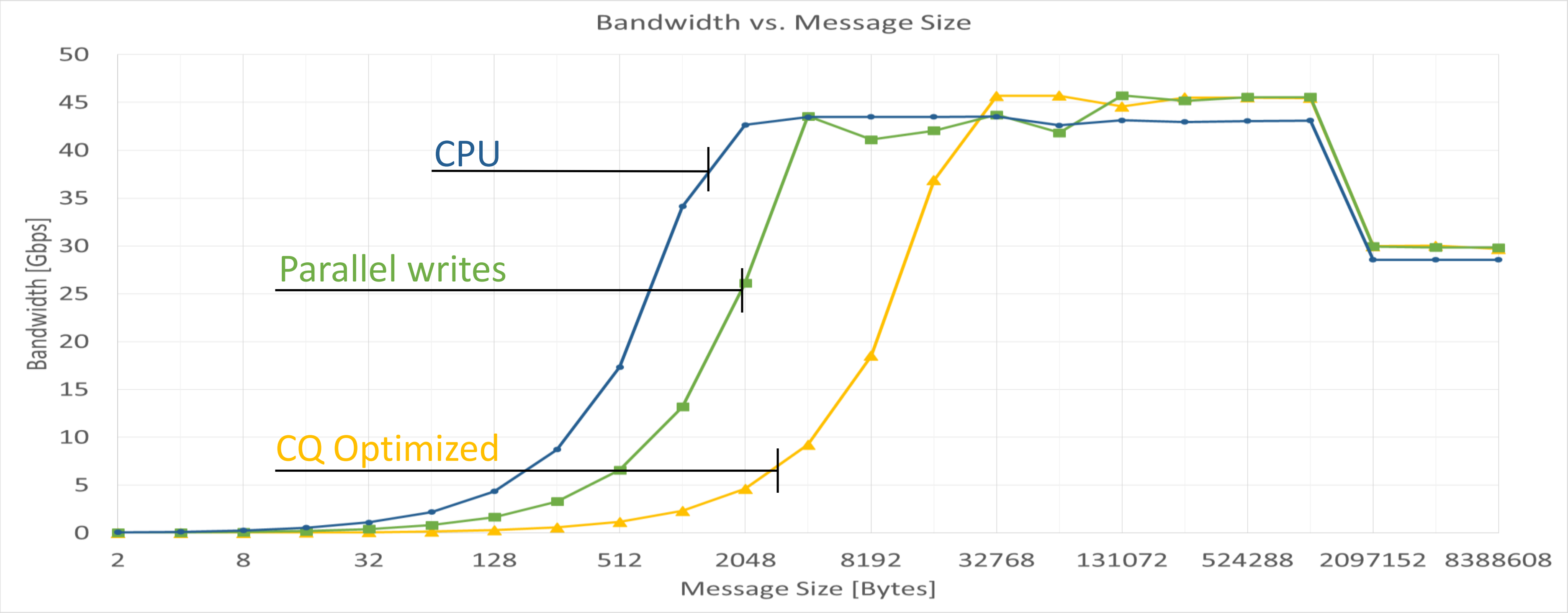
GPUrdma – 1 thread , 1 QP

- GPU controller – Optimize CQ poll



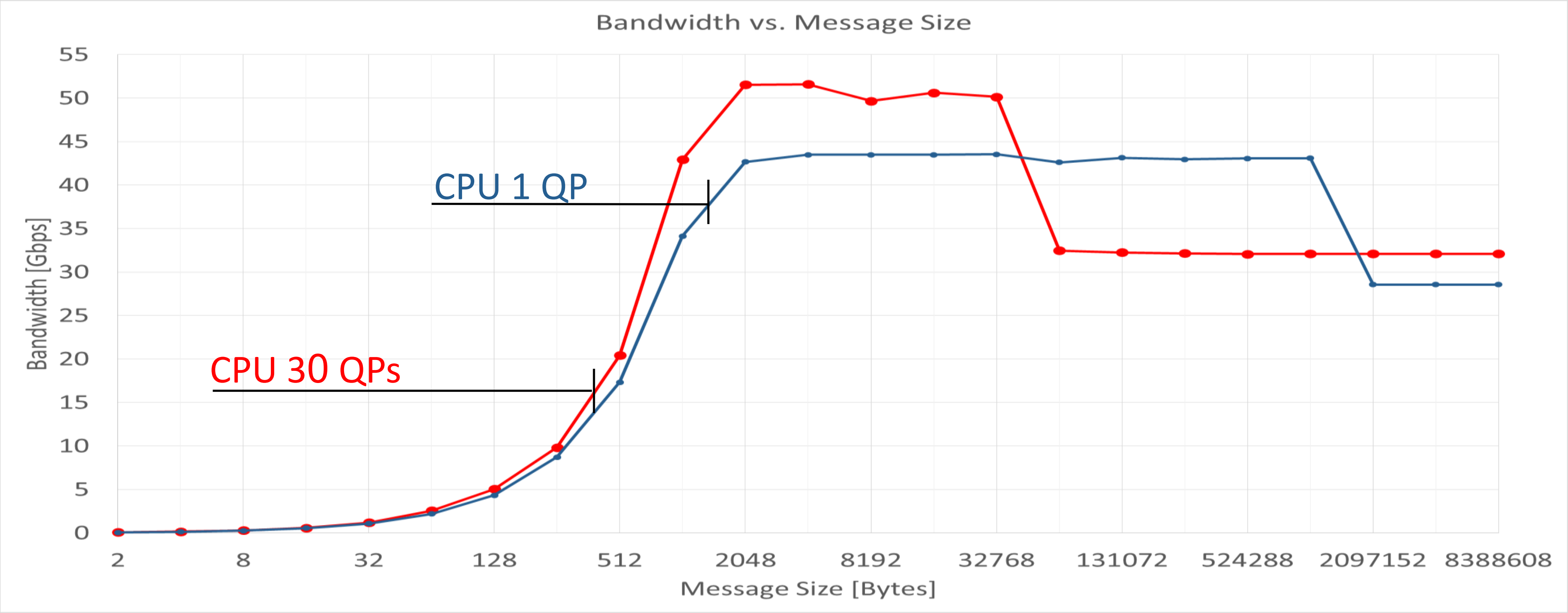
GPUrdma – 32 threads , 1 QP

- GPU controller – Write parallel jobs



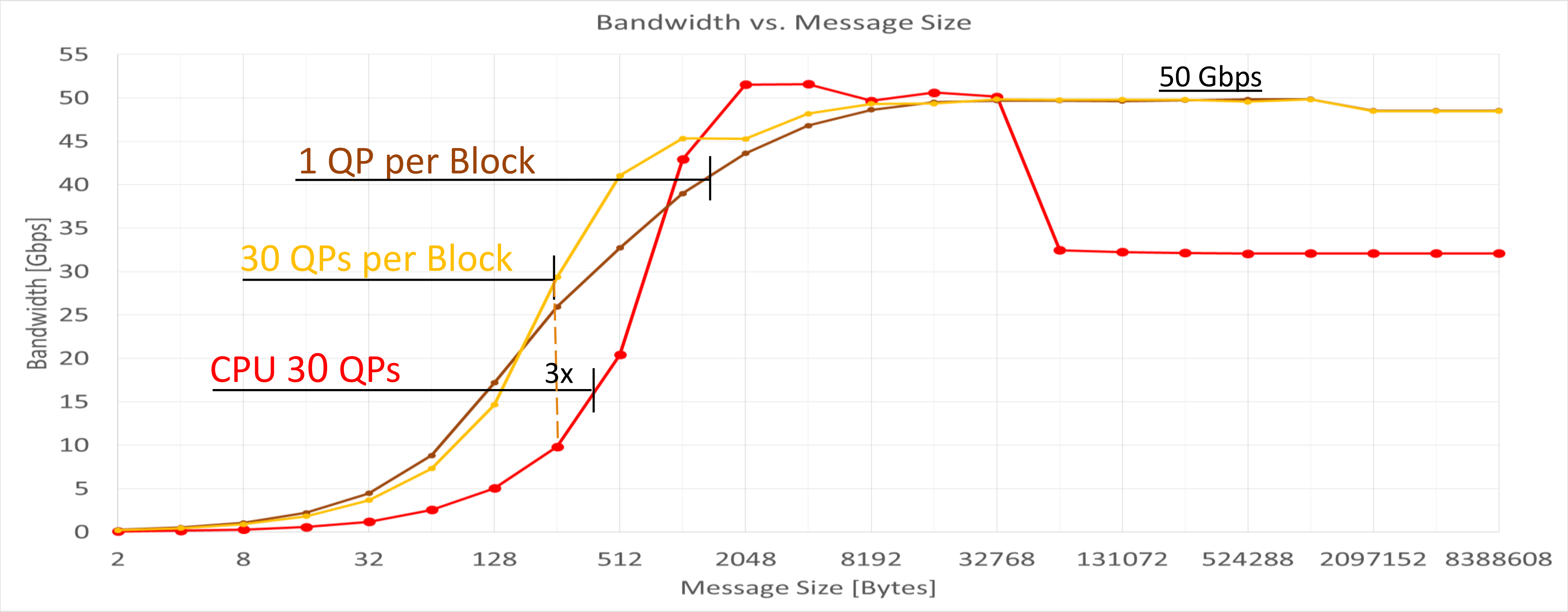
GPUDirect RDMA

- CPU controller



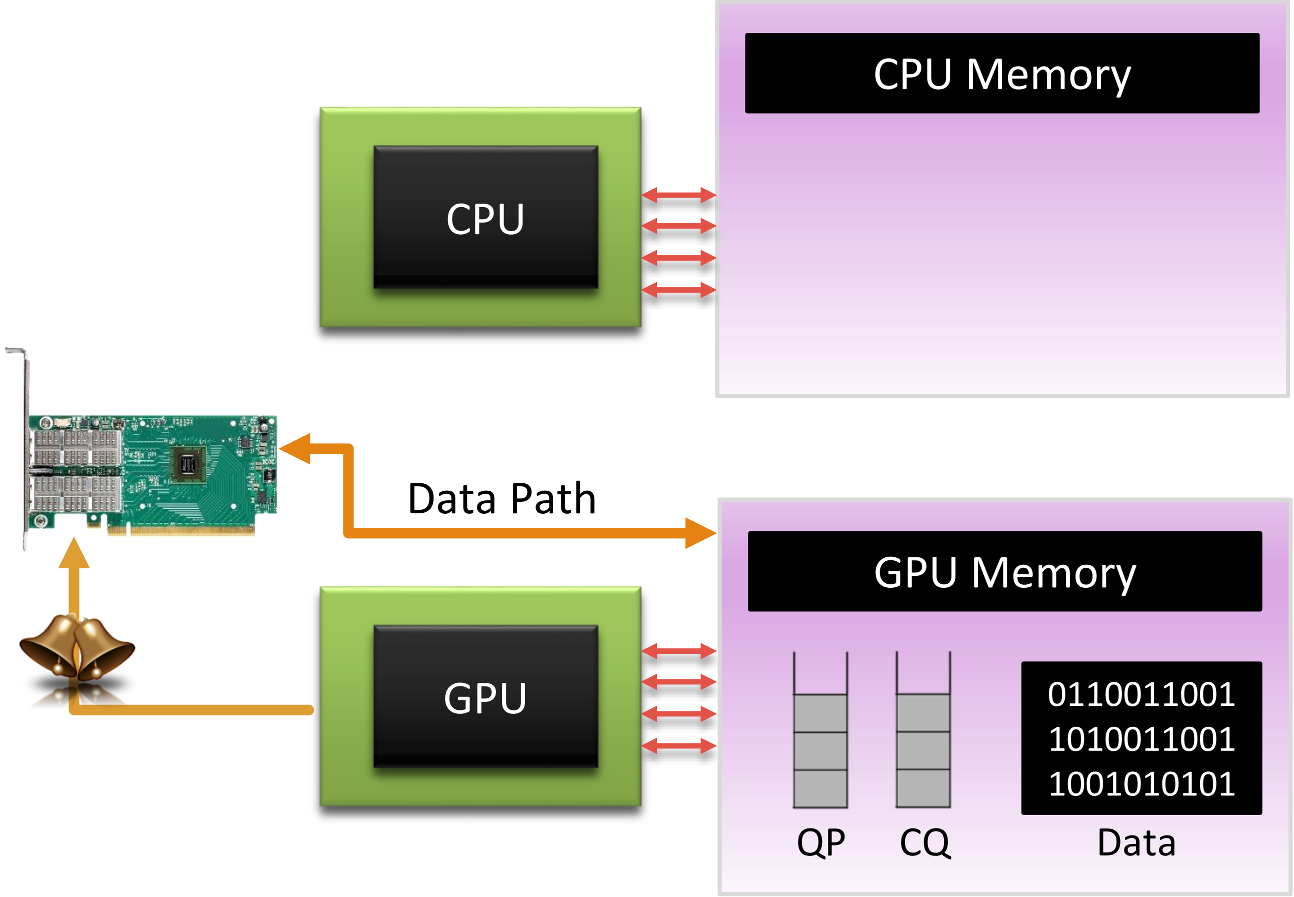
GPUrdma – 30 QPs

- 1 QP per Block   vs 30 QPs per Block



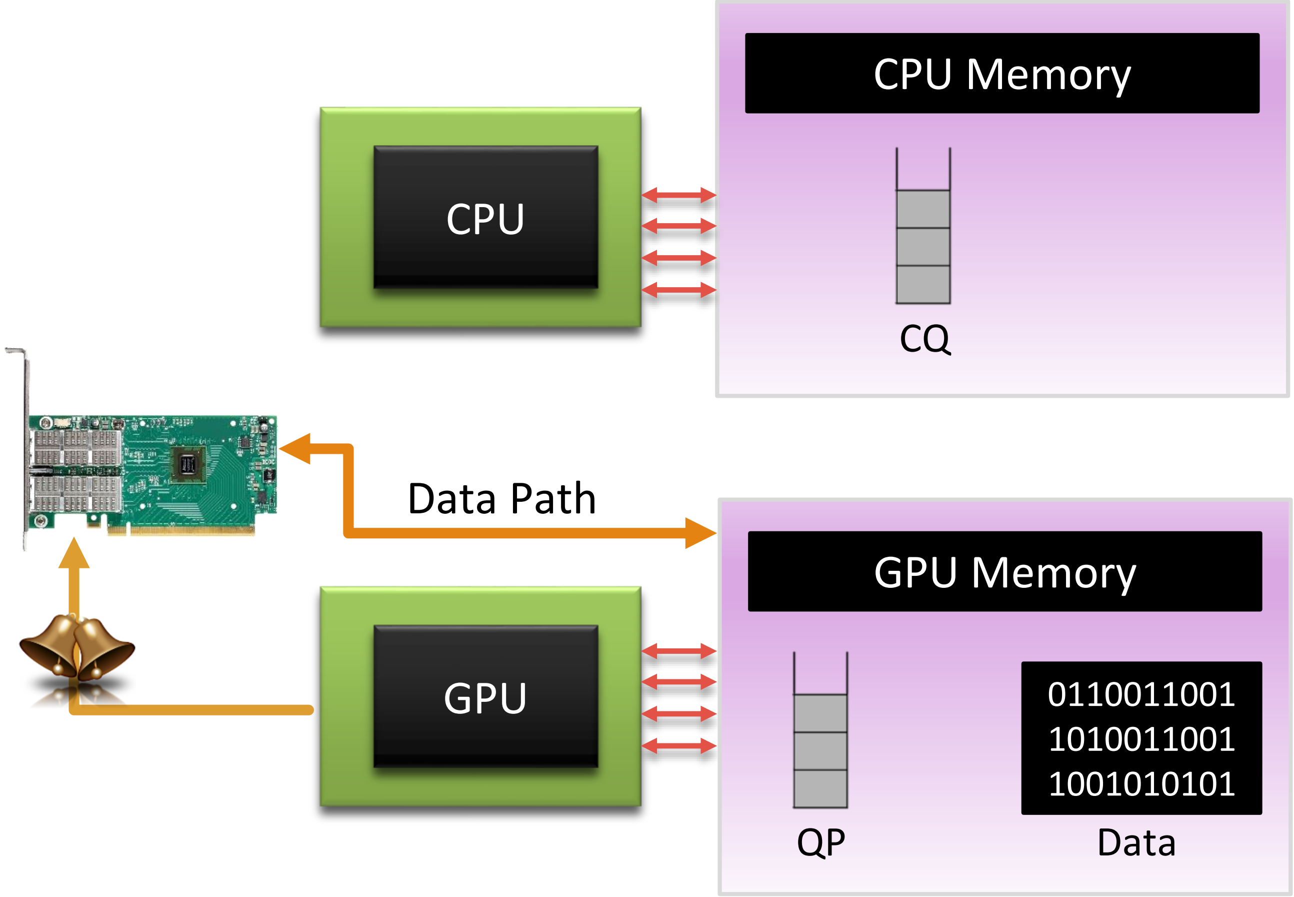
Scalability – Optimal QP/CQ location:

1. QP and CQ in GPU memory
2. QP in GPU and CQ in system memory
3. CQ in GPU and QP in system memory
4. QP and CQ in GPU memory



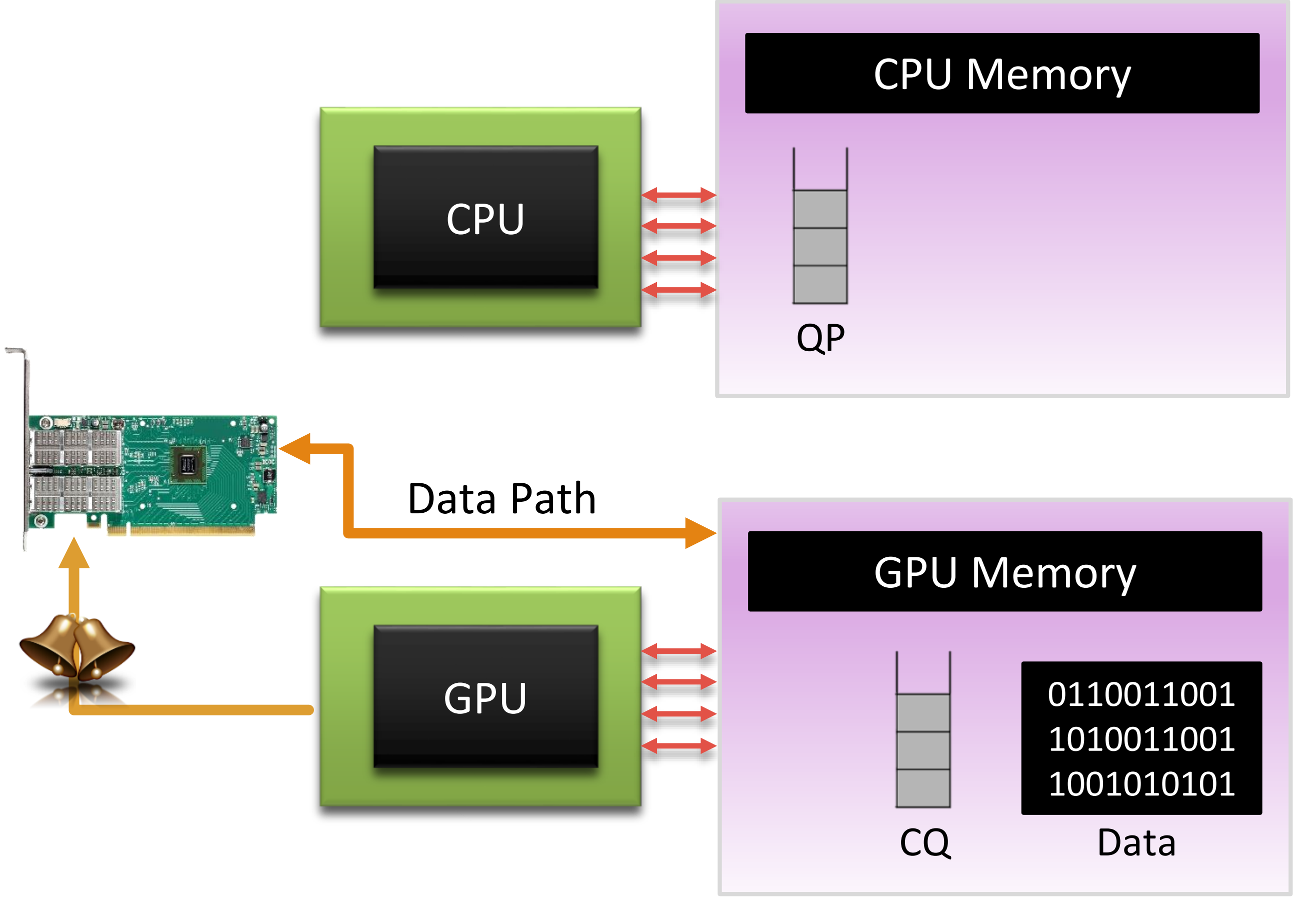
Scalability – Optimal QP/CQ location:

1. QP and CQ in GPU memory
2. QP in GPU and CQ in system memory
3. CQ in GPU and QP in system memory
4. QP and CQ in GPU memory



Scalability – Optimal QP/CQ location:

1. QP and CQ in GPU memory
2. QP in GPU and CQ in system memory
3. CQ in GPU and QP in system memory
4. QP and CQ in GPU memory



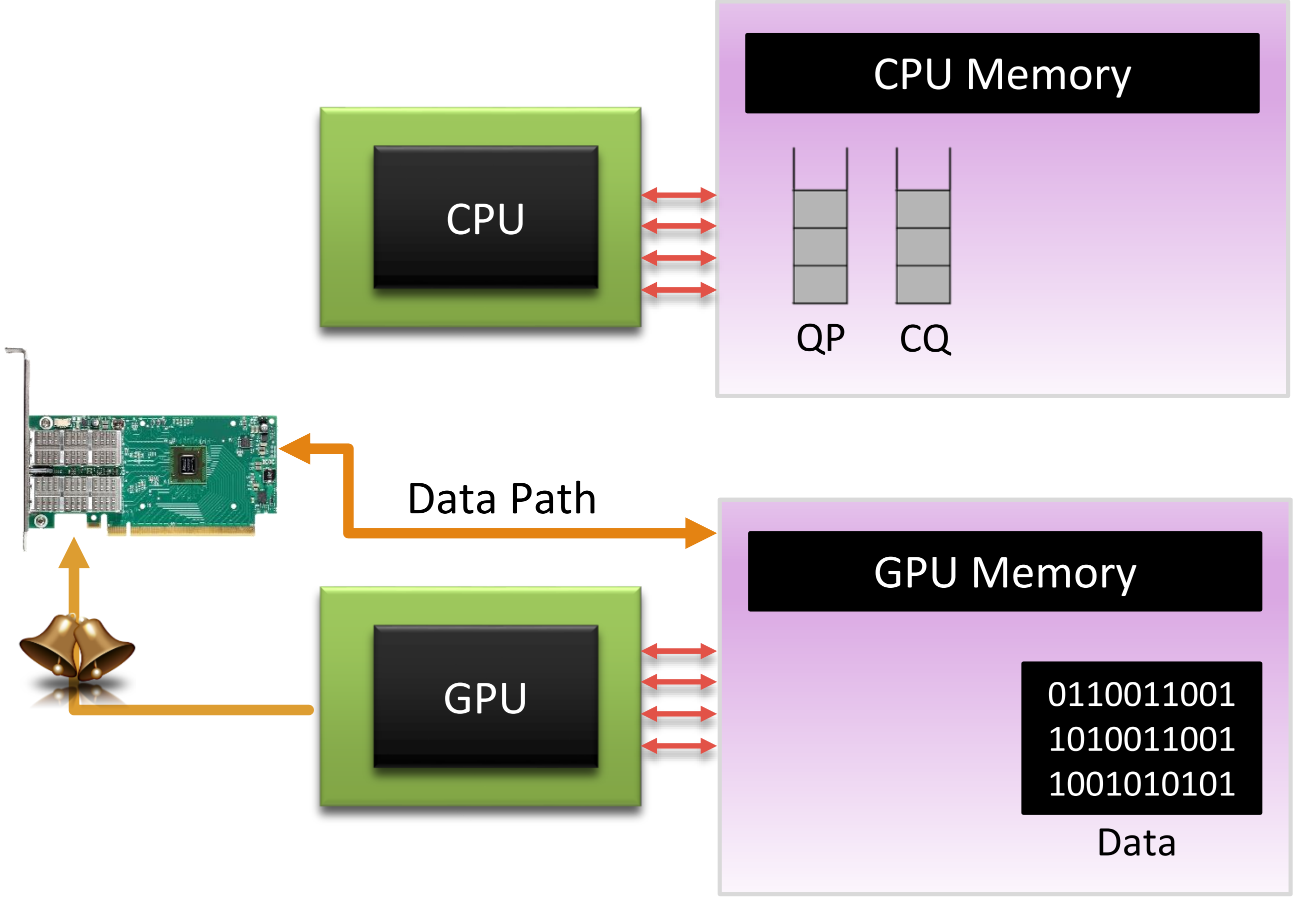
Scalability – Optimal QP/CQ location:

1. QP and CQ in GPU memory

2. QP in GPU and CQ in system memory

3. CQ in GPU and QP in system memory

4. QP and CQ in system memory



Optimal QP/CQ location:

- Throughput: No difference
- Latency:

	QP in CPU	QP in GPU
CQ in CPU	8.6	6.2
CQ in GPU	6.8	4.8

Transfer latency [µsec]

Limitations

GPUDirect RDMA - CUDA v7.5:

Running kernel may observe STALE DATA or data that arrives OUT-OF-ORDER

Scenario:

Intensive RDMA writes to GPU memory

Good news:

NVIDIA announced a CUDA 8 feature that enables consistent update

Suggested fix:

CRC32 integrity check API for error detection

Agenda

1 Introduction

2 InfiniBand Background

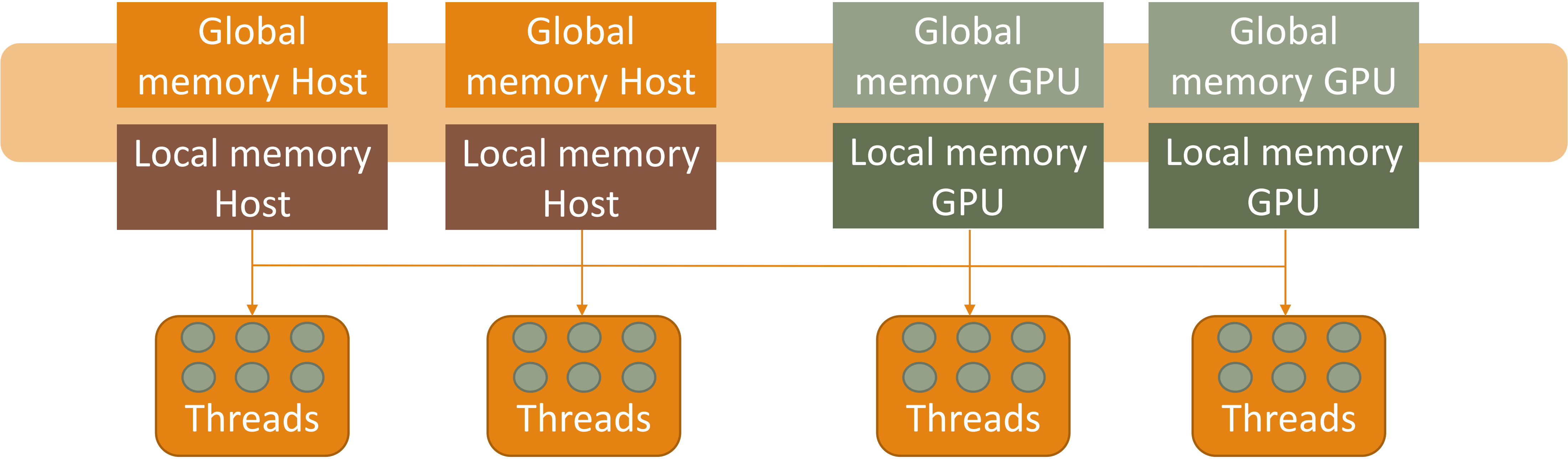
3 GPUrdma

4 GPUrdma Evaluation

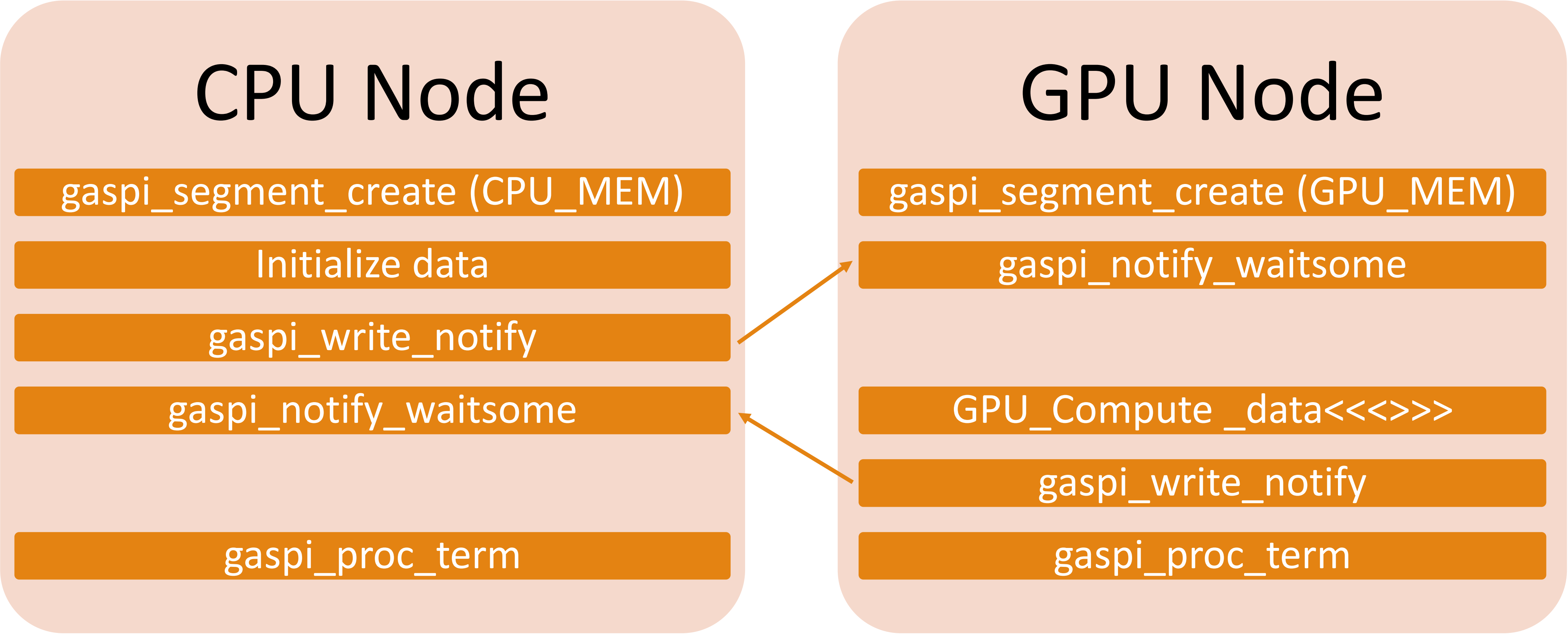
5 GPI2

GPI2 for GPUs:

GPI - A framework to implement **P**artitioned **G**lobal **A**ddress **S**pace (PGAS)
GPI2 - Extends this global address space to GPU memory



GPI2 code example



CPU Node

gaspi_segment_create (CPU_MEM)

Initialize data

gaspi_write_notify

gaspi_notify_waitsome

gaspi_proc_term

GPU Node

gaspi_segment_create (GPU_MEM)

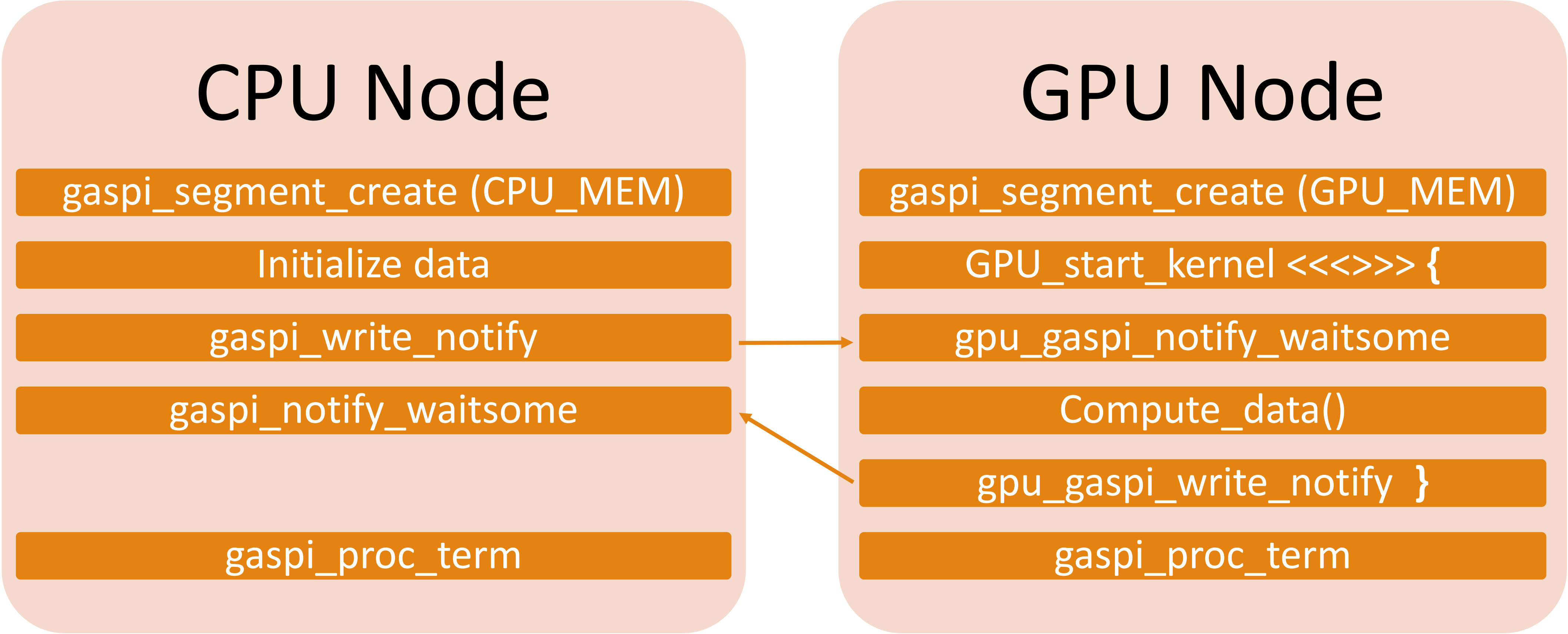
gaspi_notify_waitsome

GPU_Compute _data<<<>>>

gaspi_write_notify

gaspi_proc_term

GPI2 using GPUrdma



CPU Node

gaspi_segment_create (CPU_MEM)

Initialize data

gaspi_write_notify

gaspi_notify_waitsome

gaspi_proc_term

GPU Node

gaspi_segment_create (GPU_MEM)

GPU_start_kernel <<<>>> {

gpu_gaspi_notify_waitsome

Compute_data()

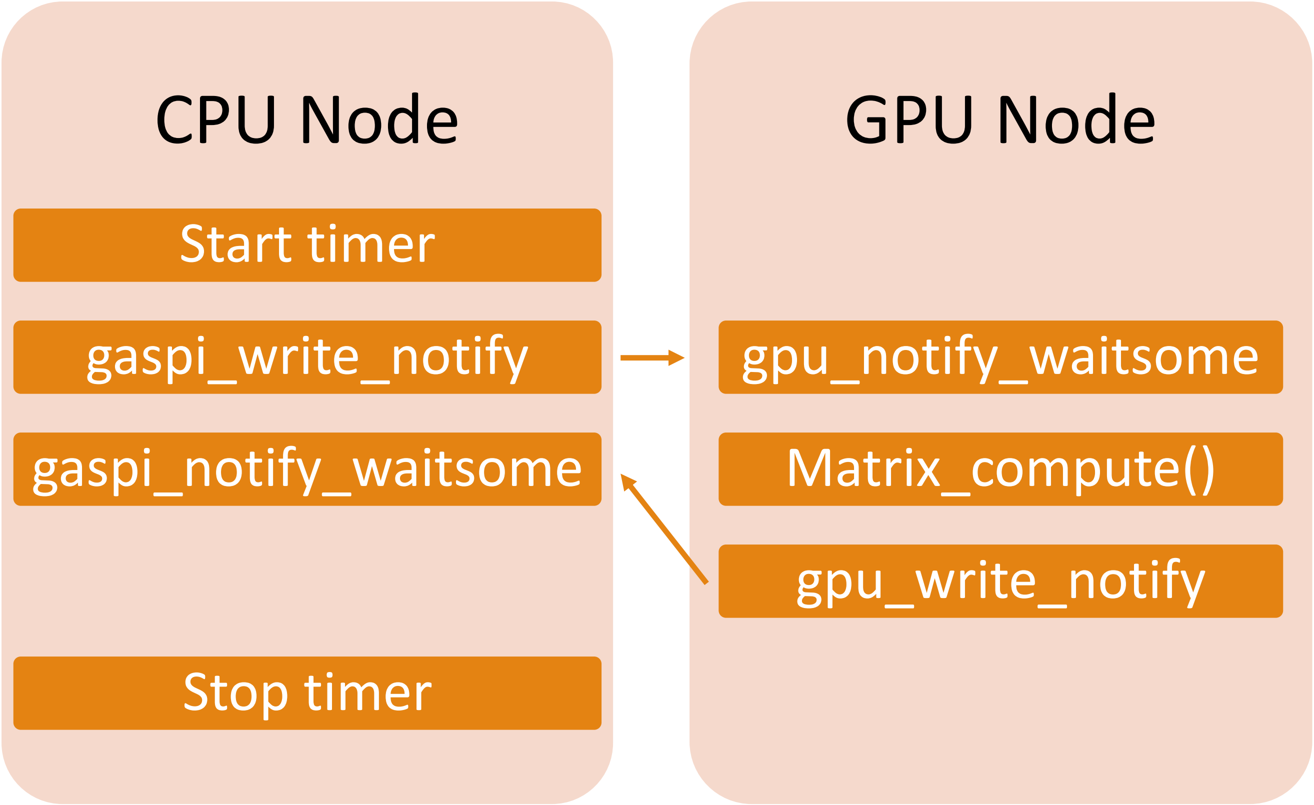
gpu_gaspi_write_notify }

gaspi_proc_term

GPUrdma Multi-Matrix vector product

Batch size [Vectors]	GPI2	GPUrdma
480	2.6	11.7
960	4.8	18.8
1920	8.4	25.2
3840	13.9	29.1
7680	19.9	30.3
15360	24.3	31.5

- System throughput in millions of 32x1 vector multiplications per second as a function of the batch size



Related works

Lena Oden, Fraunhofer Institute for Industrial Mathematics:

- Infiniband-Verbs on GPU: A case study of controlling an Infiniband network device from the GPU
- Analyzing Put/Get APIs for Thread-collaborative Processors

Mark Silberstein, Technion – Israel Institute of Technology:

- GPUnet: networking abstractions for GPU programs
- GPUfs: Integrating a file system with GPUs

Thanks