
13th ANNUAL WORKSHOP 2017

# Asynchronous Peer-to-Peer Device Communication

Feras Daoud, Leon Romanovsky

PeerDirect ASYNC

[ 28 March, 2017 ]

# Agenda

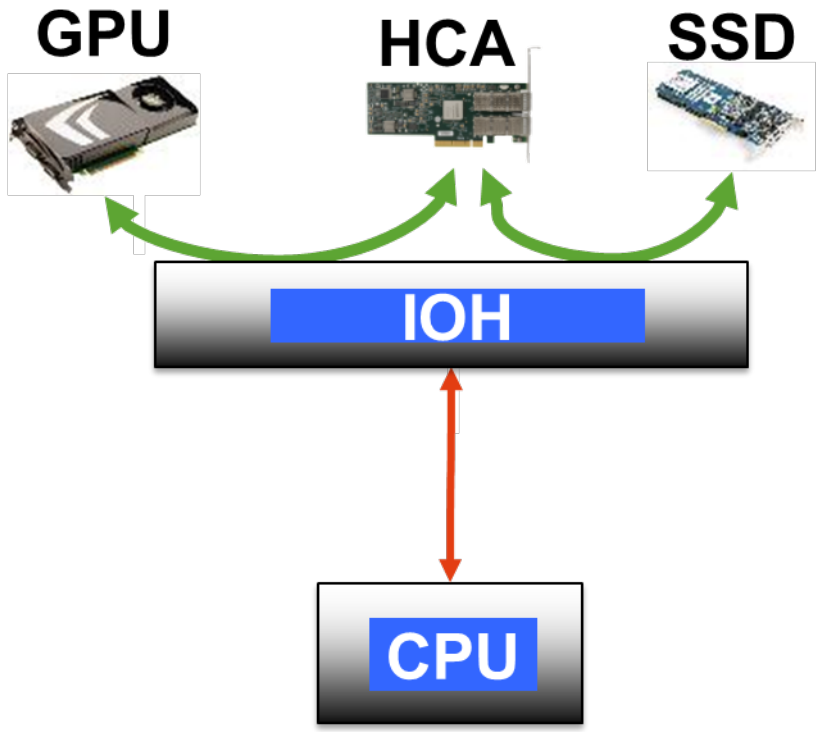
- Peer-to-Peer communication
- PeerDirect technology
- PeerDirect and PeerDirect Async
- Performance
- Upstream work


# Peer-to-Peer Communication

# Peer-to-Peer Communication

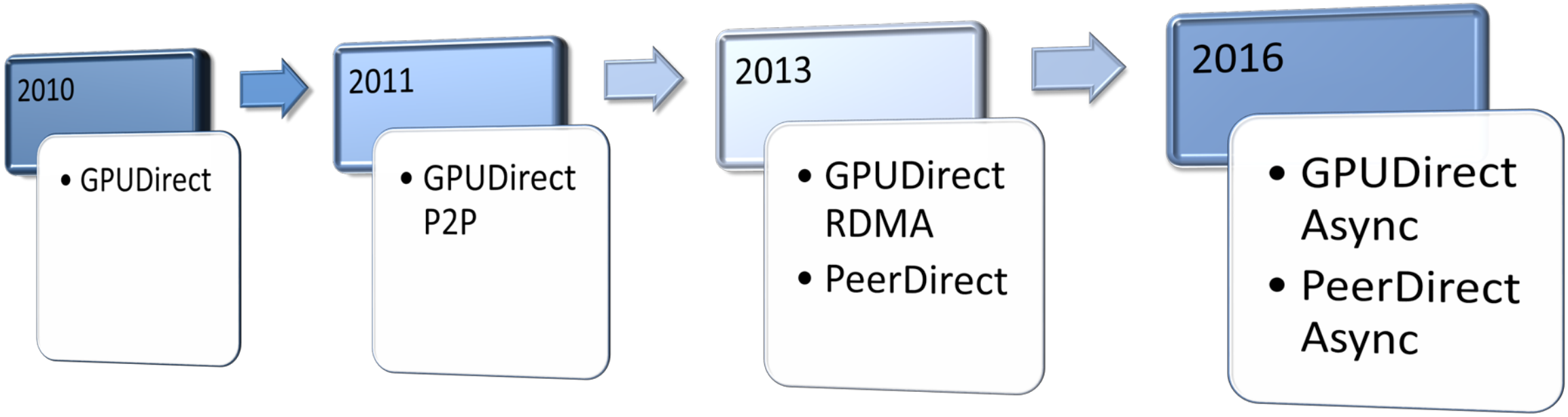
*"Direct data transfer between PCI-E devices without the need to use main memory as a temporary storage or use of the CPU for moving data."*

- **Main advantages:**
  - Allow direct data transfer between devices
  - Control the peers directly from other peer devices
  - Accelerate transfers between different PCI-E devices
  - Improve latency, system throughput, CPU utilization, energy usage
  - Cut out the middleman



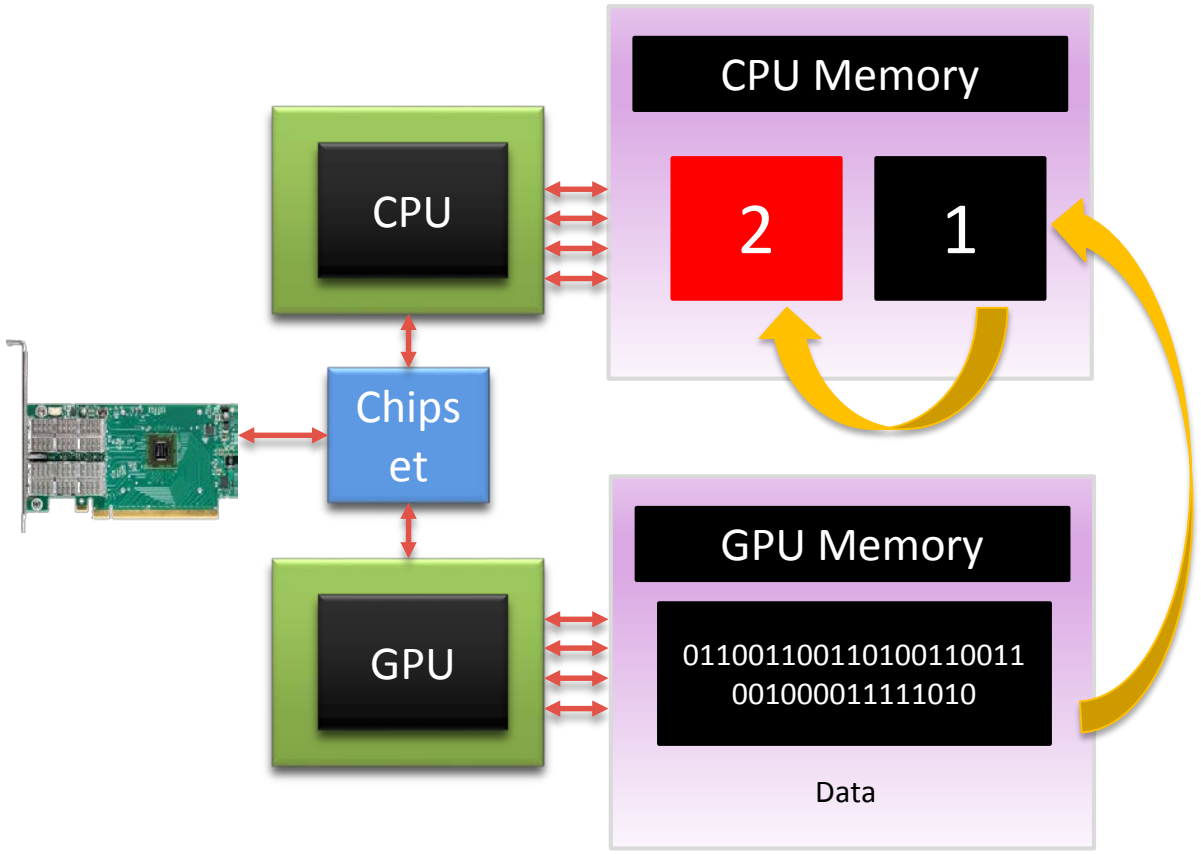

# PeerDirect Technology

# Timeline


2010
• GPUDirect
2011
• GPUDirect P2P
2013
• GPUDirect RDMA
• PeerDirect
2016
• GPUDirect Async
• PeerDirect Async

# Prior To GPUDirect

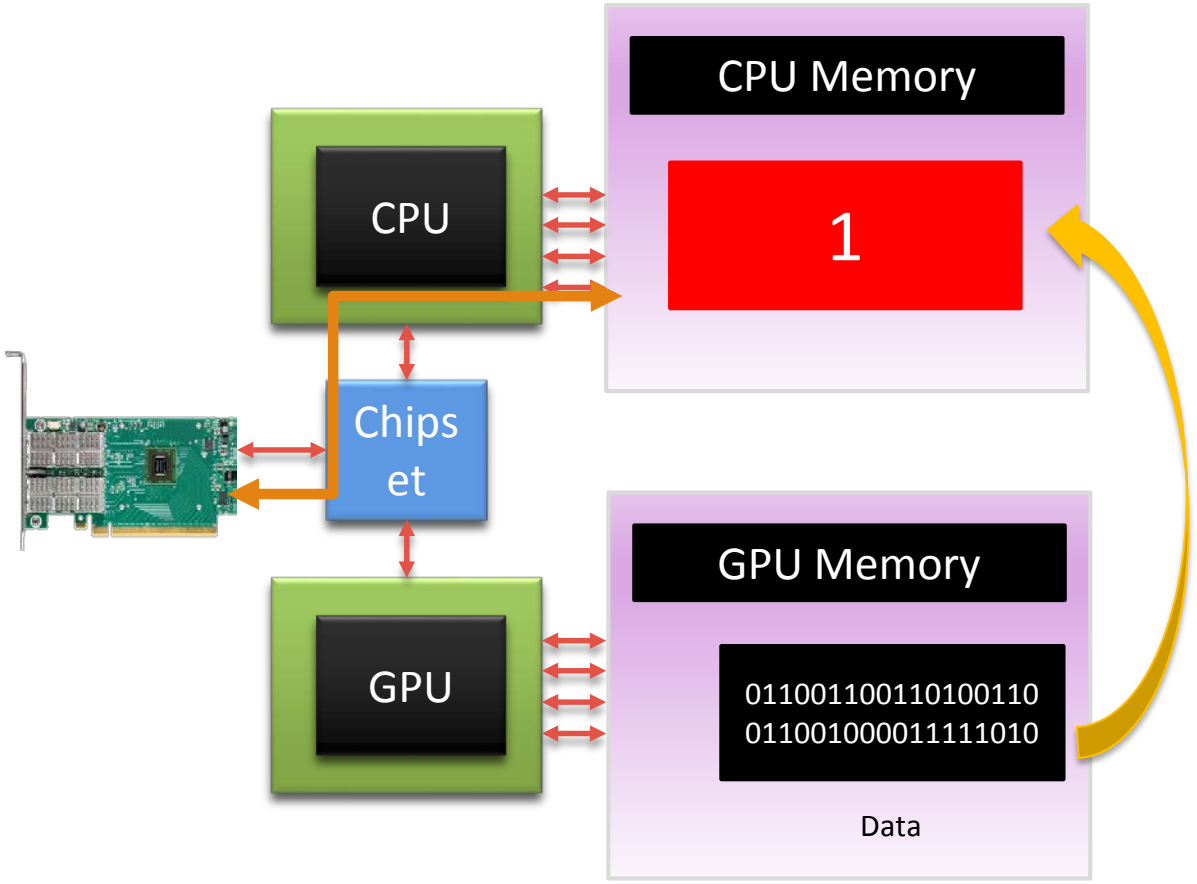
- GPUs use driver-allocated pinned memory buffers for transfers
- RDMA driver use pinned buffers for zero-copy kernel-bypass communication
- It was impossible for RDMA drivers to pin memory allocated by the GPU
- Userspace needed to copy data between the GPU driver's system memory region and the RDMA memory region



# GPUDirect/GPUDirect P2P

- GPU and RDMA device share the same “pinned” buffers
- GPU copies the data to system memory
- RDMA device sends it from there

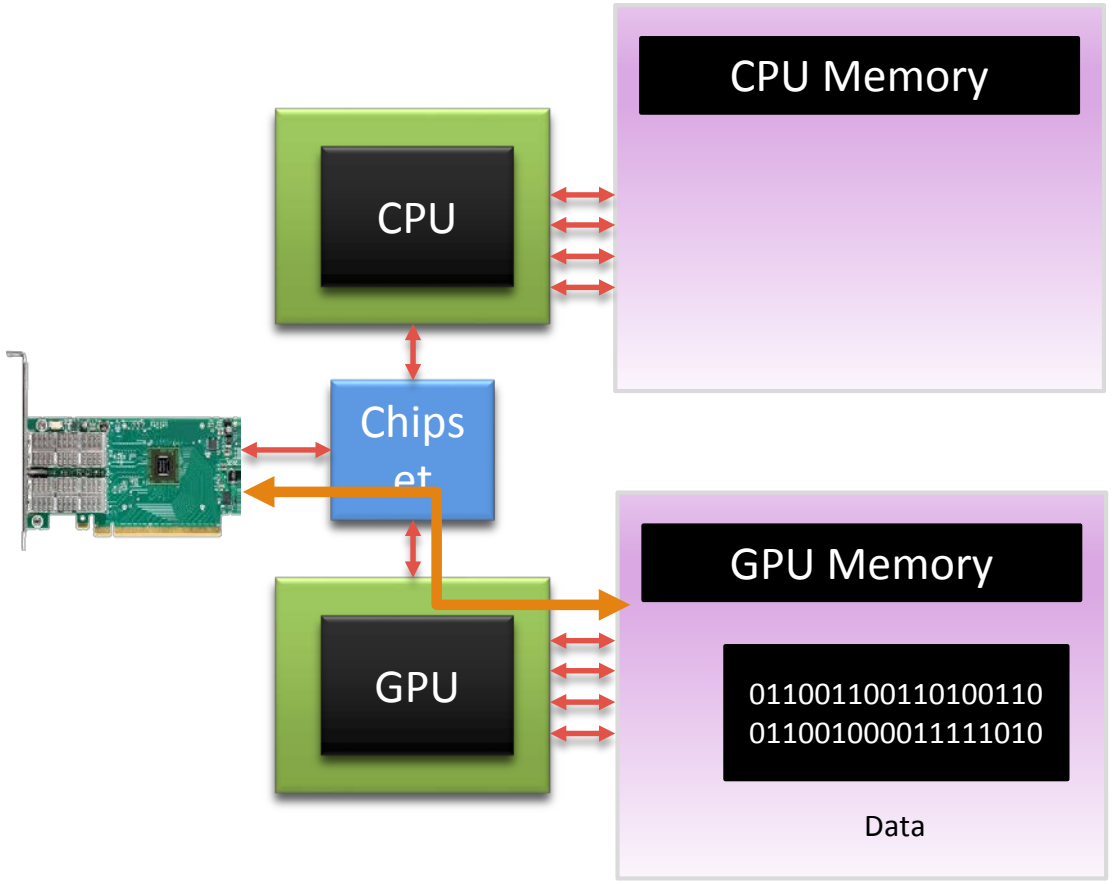
- Advantages
  - Eliminate the need to make a redundant copy in CUDA host memory
  - Eliminate CPU bandwidth and latency bottlenecks



# GPUDirect RDMA/PeerDirect

- CPU synchronizes between GPU tasks and data transfer
- HCA directly accesses GPU memory

- Advantages
  - Direct path for data exchange
  - Eliminate the need to make a redundant copy in host memory

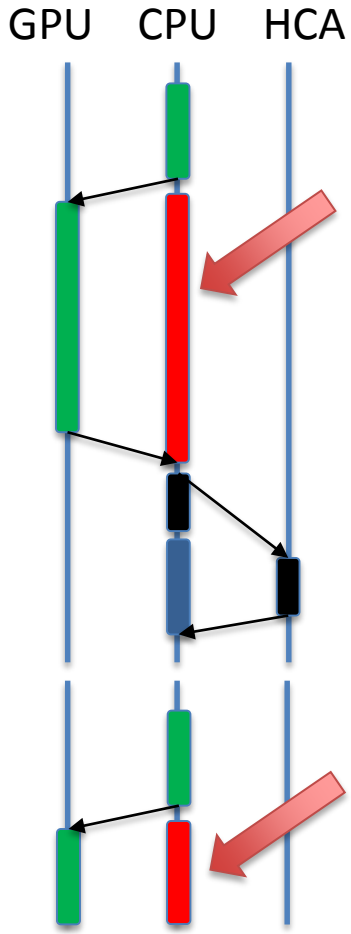


# GPUDirect RDMA/PeerDirect

## CPU Utilization

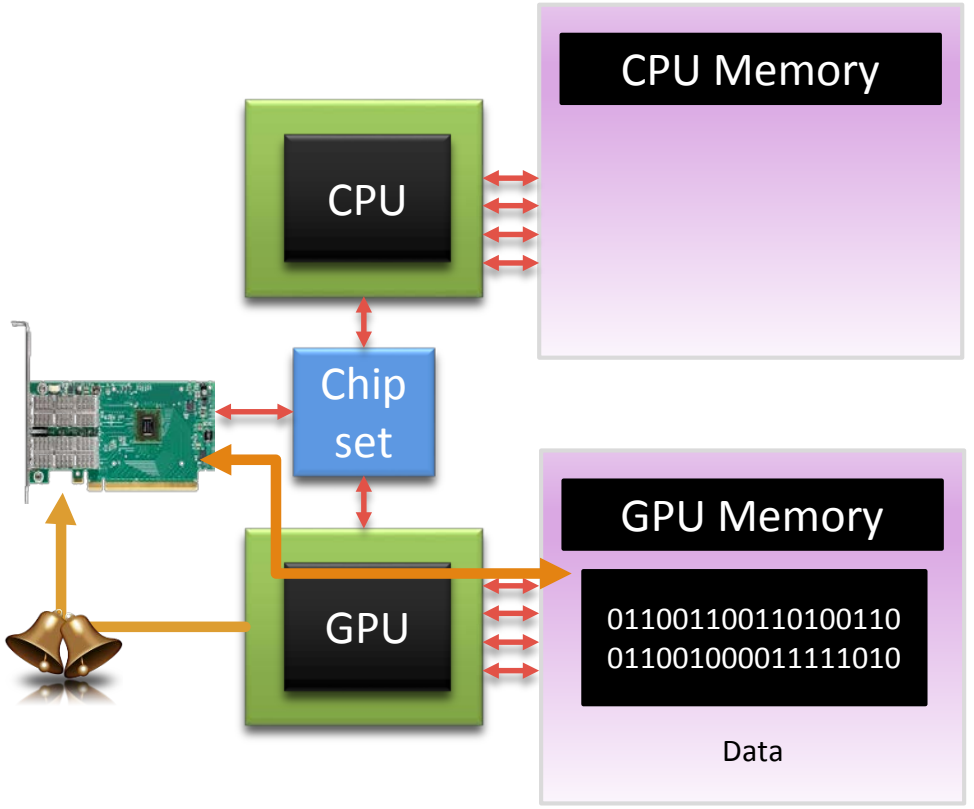
```
while(fin) {
    gpu_kernel <<<… , stream>>>(buf);
    cudaStreamSynchronize(stream);
    ibv_post_send(buf);
    ibv_poll_cq(cqe);
}
```

100% CPU Utilization



# GPUDirect Async/PeerDirect Async

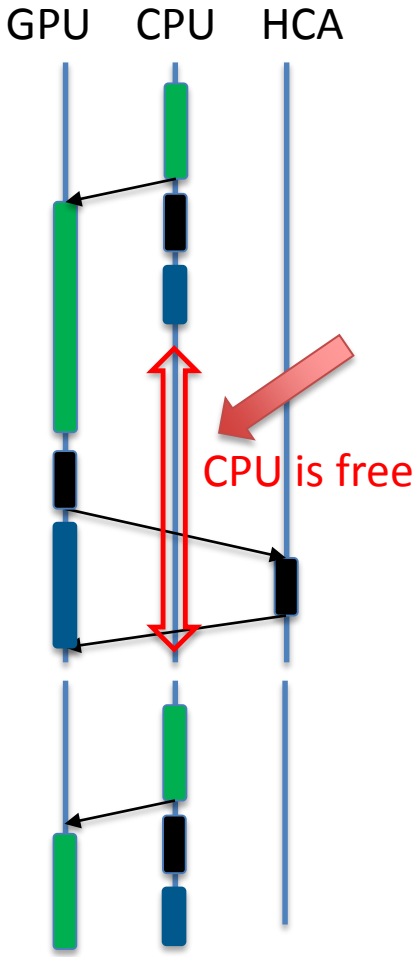
- Control the HCA from the GPU
  - Performance
    - Enable batching of multiple GPU and communication tasks
    - Reduce latency
  - Reduce CPU utilization
    - Light weight CPU
    - Less power
- CPU prepares and queues compute and communication tasks on GPU
- GPU triggers communication on HCA
- HCA directly accesses GPU memory



# GPUDirect Async/PeerDirect Async

```
while(fin) {
    gpu_kernel <<<… , stream>>>(buf);
    gds_stream_queue_send(stream, qp,
buf);
    gds_stream_wait_cq(stream, cqe);
}
```

No CPU in critical path



# Peer-to-Peer Evolution

| | |
|---|---|
| GPUDirect | • Eliminate the need to make a redundant copy in CUDA host memory<br>• Eliminate CPU bandwidth and latency bottlenecks |
| PeerDirect | • Eliminate the need to make a redundant copy in host memory<br>• Direct path for data exchange |
| PeerDirect Sync | • Control RDMA device from the GPU<br>• Reduce CPU utilization |


# PeerDirect

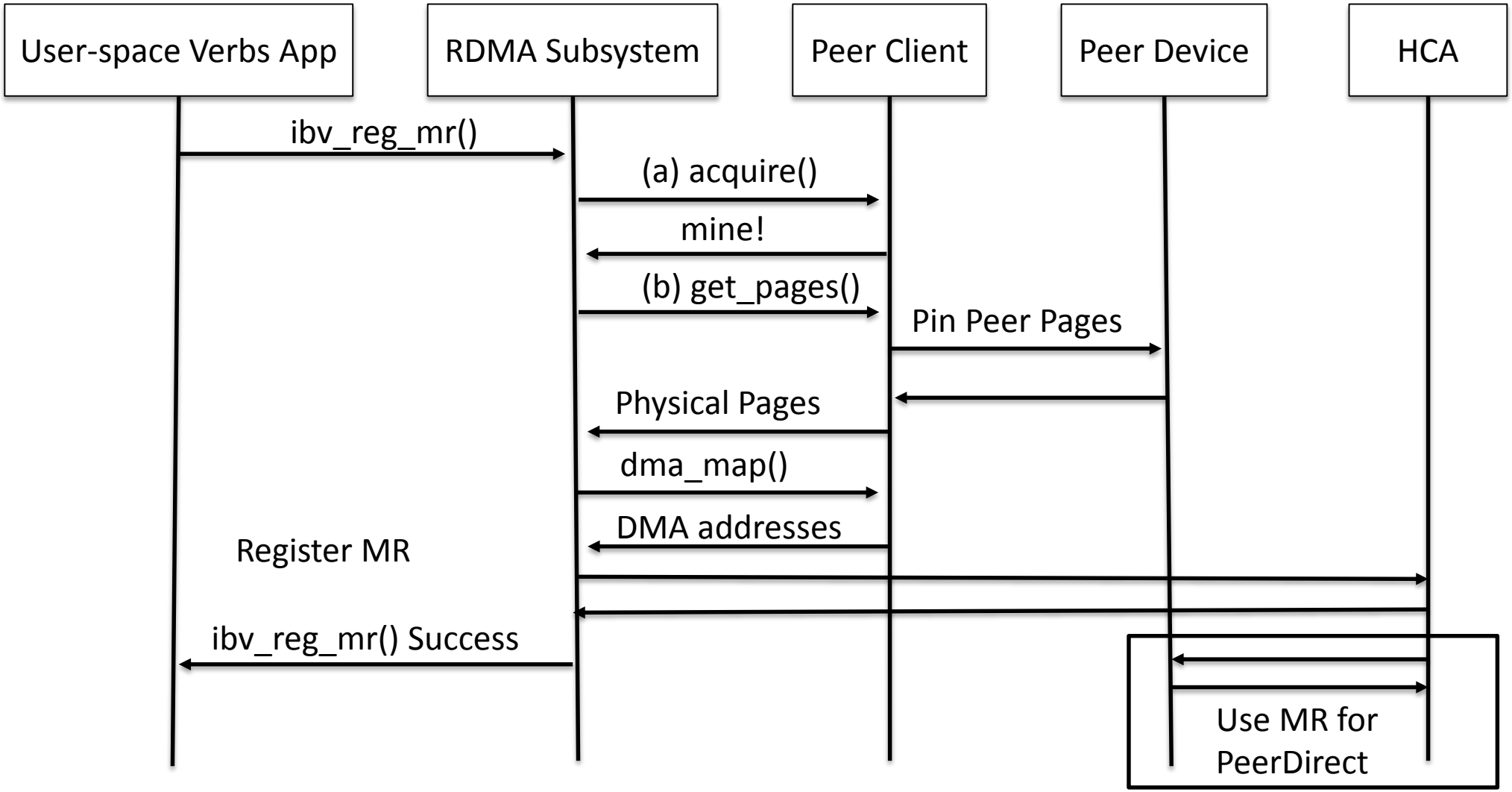
# PeerDirect

## How Does It Work?

- Allow ibv_reg_mr() to register peer memory
- Peer devices implement new kernel module – io_peer_mem
- Register with RDMA subsystem - ib_register_peer_memory_client()
- io_peer_mem implements the following callbacks :
  - **acquire()** – detects whether a virtual memory range belongs to the peer
  - **get_pages()** – asks the peer for the physical memory addresses matching the memory region
  - **dma_map()** – requests the bus addresses for the memory region
  - Matching callbacks for release: **dma_unmap()**, **put_pages()** and **release()**

# PeerDirect

## Memory Region Registration




PeerDirect ASYNC

# PeerDirect Async

# PeerDirect Async

## How Does It Work?

- Allow peer devices to control the network card
  - latency reduction, batching of management operations
- Two new supported operations
  - Queue a set of send operations to be triggered by the GPU - ibv_exp_peer_commit_qp()
  - Test for a “successful completion” - ibv_exp_peer_peek_cq()
- Dedicated QPs and CQs for PeerDirect Sync
  - Avoid to interlock PeerDirect Sync and normal post_send/poll_cq
- Device agnostic
  - Currently, built to support NVIDIA’s GPUs
  - Support other HW as well – FPGAs; storage controllers

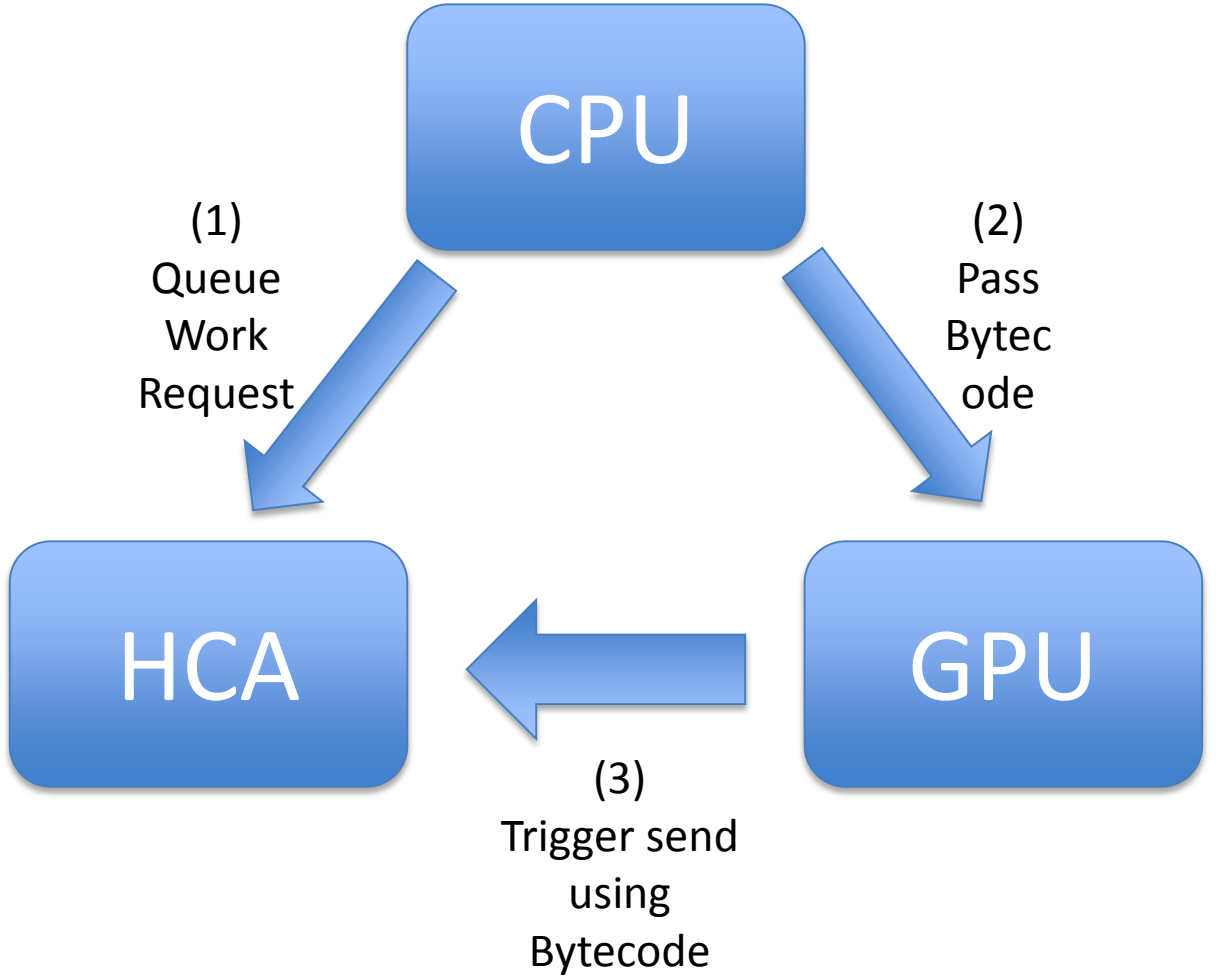
# Transmit Operation

*Create a QP ->*

*Mark it for PeerDirect Sync ->*

*Associate it with the peer*

1. Post work requests using ibv_post_send()
   - Doorbell record is not updated
   - Doorbell is not ringed
2. Use ibv_exp_peer_commit_qp() to get bytecode for committing all WQEs currently posted to the send work queue
3. Queue the translated bytecode operations on the peer after the operations that generate the data that will be sent



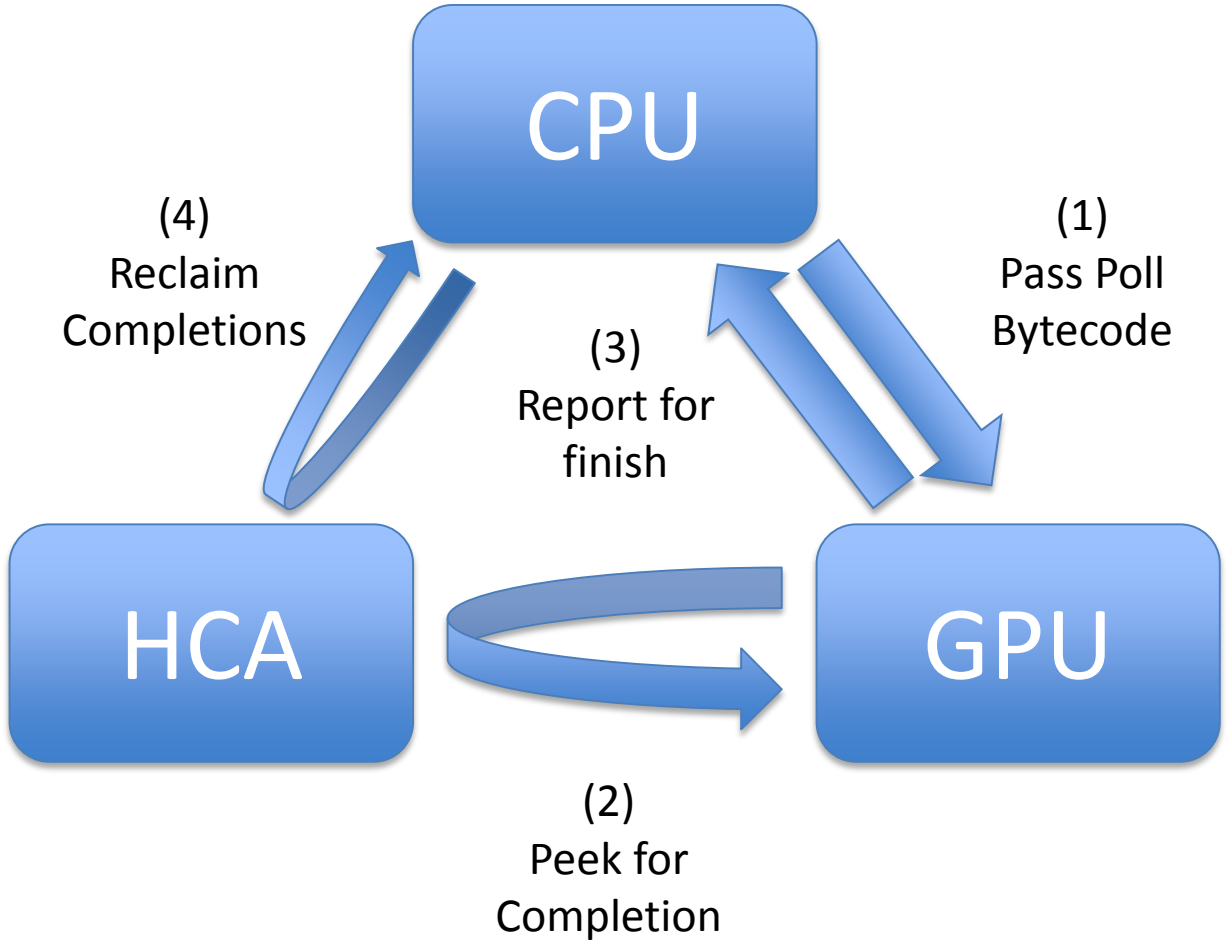
# Completion Handling

*Create a CQ ->*

*Mark it for PeerDirect Sync ->*

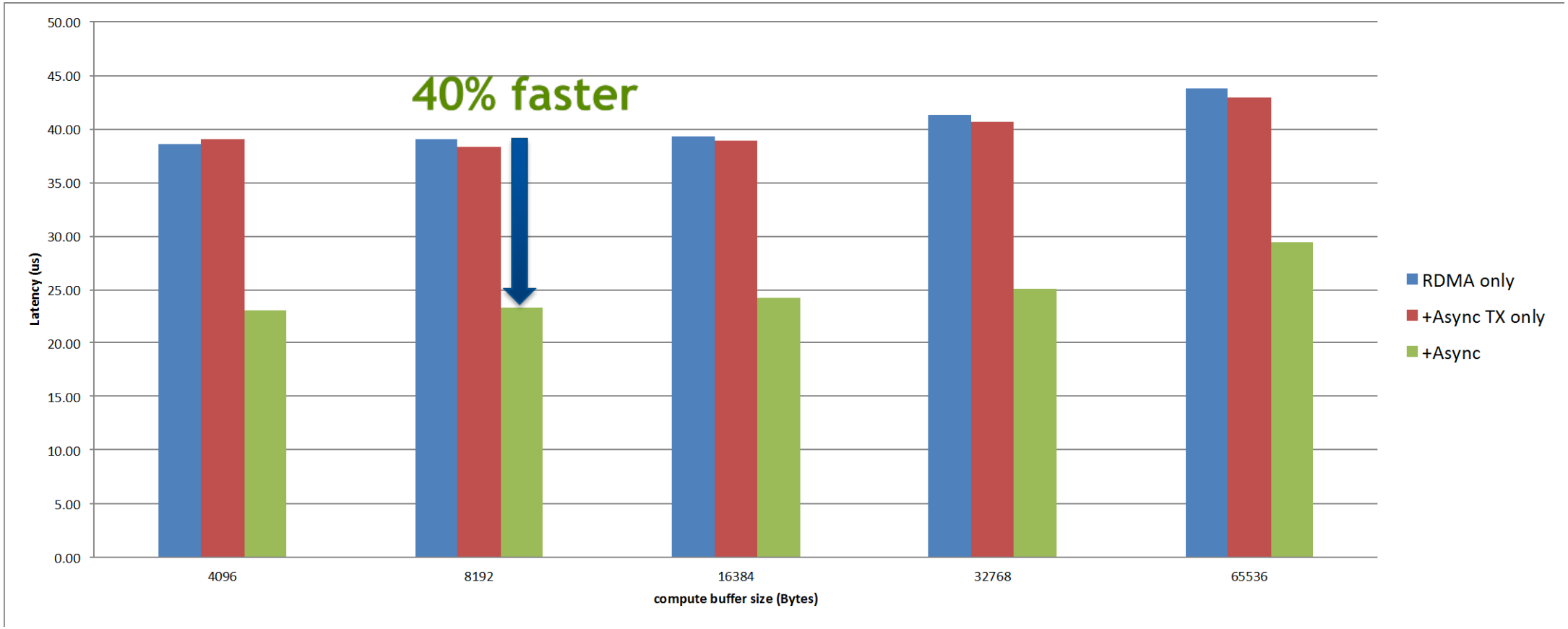
*Associate it with the peer*

1. Use ibv_exp_peer_peek_cq() to get bytecode for peeking a CQ in a specific offset from the currently expected CQ entry
2. Queue the translated operations on the peer before the operations that use the received data
3. Synchronize the CPU with the peer to insure that all the operations has ended
4. Use ibv_poll_cq() to consume the completion entries



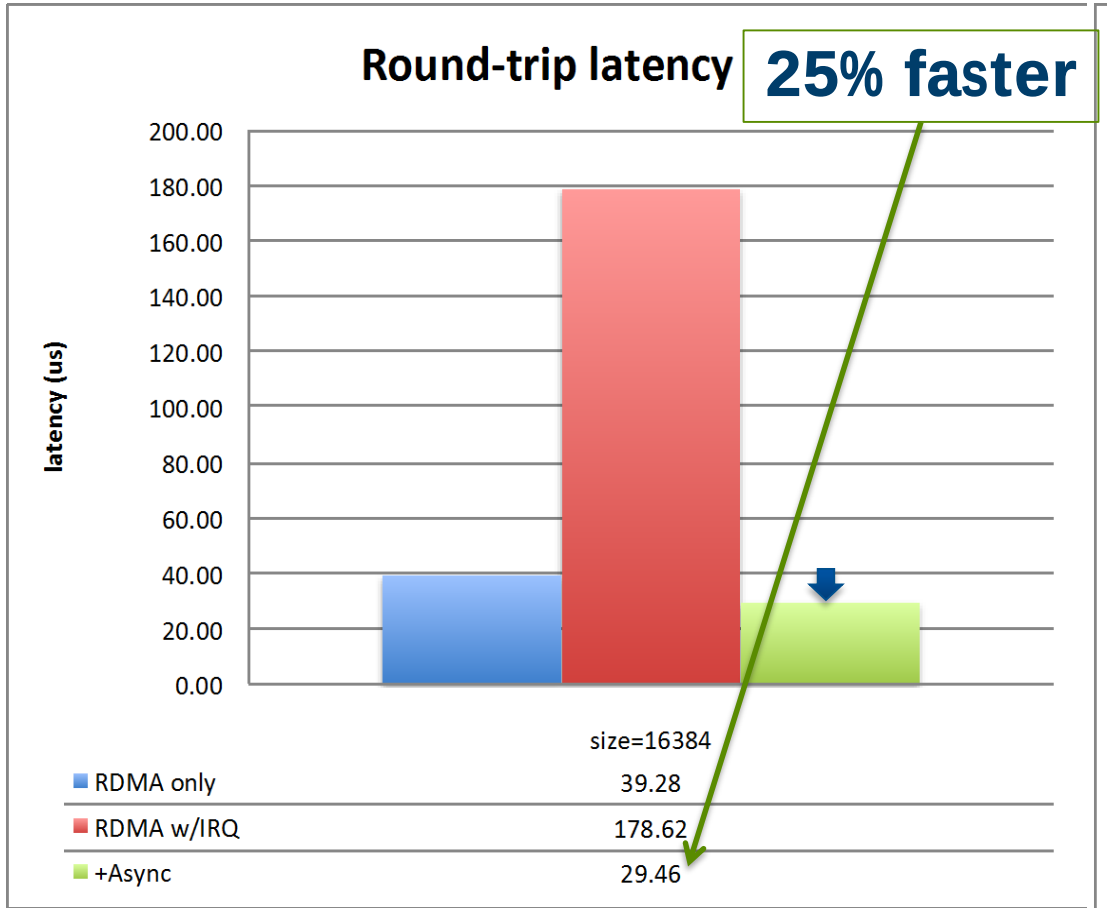

# Performance

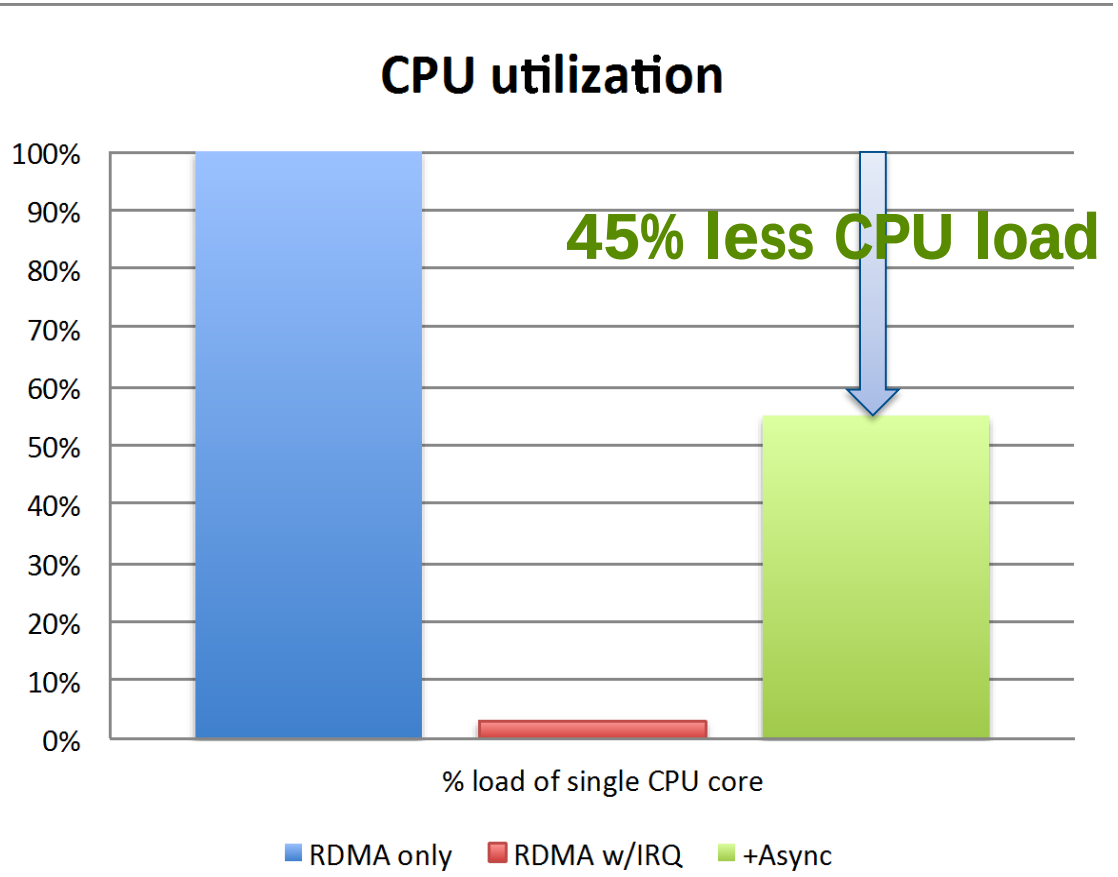
# Performance mode




[*] modified ud_pingpong test: recv+GPU kernel+send on each side.
2 nodes: Ivy Bridge Xeon + K40 + Connect-IB + MLNX switch, 10000 iterations, message size: 128B, batch size: 20

# Economy Mode




[*] modified ud_pingpong test, HW same as in previous slide


# Upstream Work

# Peer-to-Peer – Upstream Proposals

- **Peer-to-Peer DMA**
  - Mapping DMA addresses of PCI device to IOVA of other device
- **ZONE_DEVICE**
  - Extend ZONE_DEVICE functionality to memory not cached by CPU
- **RDMA extension to DMA-BUF**
  - Allow memory region create from DMA-BUF file handle
- **IOPMEM**
  - A block device for PCI-E memory
- **Heterogeneous Memory Management (HMM)**
  - Common address space will allow migration of memory between devices

OPENFABRICS ALLIANCE

13th ANNUAL WORKSHOP 2017

# THANK YOU

Feras Daoud, Leon Romanovsky


# BACKUP

# Bytecode

```
struct peer_op_wr {
    struct peer_op_wr *next; enum ibv_exp_peer_op type;
    union {
        struct { uint64_t fence_flags; } fence;
        struct {
            uint32_t data;
            uint64_t target_id;
            size_t offset;
        } dword_va;
        struct {
            uint64_t data;
            uint64_t target_id;
            size_t offset;
        } qword_va;
        struct {
            void *src; uint64_t target_id;
            size_t offset; size_t len;
        } copy_op;
    } wr;
    uint32_t comp_mask;
};
```

```
enum ibv_exp_peer_op {
        IBV_EXP_PEER_OP_FENCE           = 0,

        IBV_EXP_PEER_OP_STORE_DWORD     = 1,
        IBV_EXP_PEER_OP_STORE_QWORD     = 2,
        IBV_EXP_PEER_OP_COPY_BLOCK      = 3,

        IBV_EXP_PEER_OP_POLL_AND_DWORD  = 12,
        IBV_EXP_PEER_OP_POLL_NOR_DWORD  = 13,
};

enum ibv_exp_peer_fence {
        IBV_EXP_PEER_FENCE_OP_READ   = (1 << 0),
        IBV_EXP_PEER_FENCE_OP_WRITE  = (1 << 1),
        IBV_EXP_PEER_FENCE_SCOPE_CPU  = (1 << 2),
        IBV_EXP_PEER_FENCE_SCOPE_HCA  = (1 << 3),
        IBV_EXP_PEER_FENCE_MEM_SYS   = (1 << 4),
        IBV_EXP_PEER_FENCE_MEM_PEER  = (1 << 5),
};
```