InfiniBand Meets OpenSHMEM

March 2014

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™

Leading Supplier of End-to-End Interconnect Solutions

Comprehensive End-to-End InfiniBand and Ethernet Portfolio

ICs

Adapter Cards

Switches/Gateways

Host/Fabric Software

Metro / WAN

Cables/Modules

Virtual Protocol Interconnect (VPI) Technology

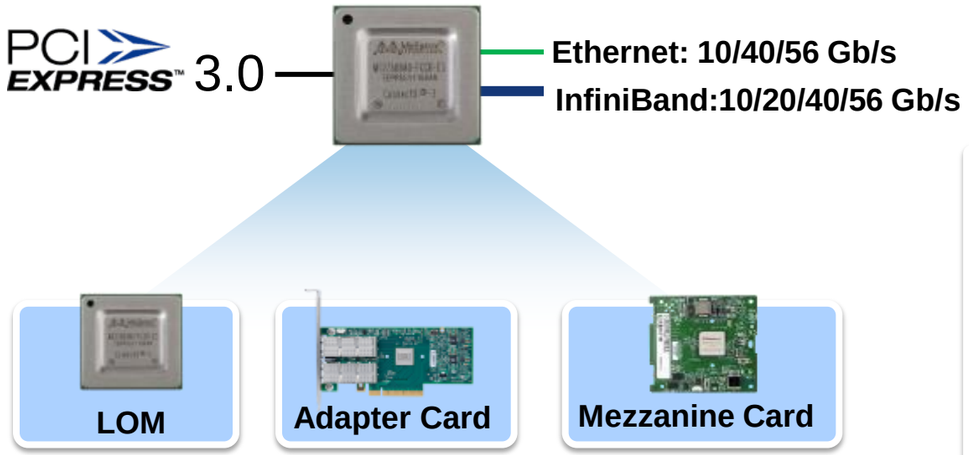
ConnectX®-3 VPI Adapter

redhat. Novell suse Microsoft vmware OPENFABRICS SOFTWARE

Applications

Networking | **Storage** | **Clustering** | **Management**

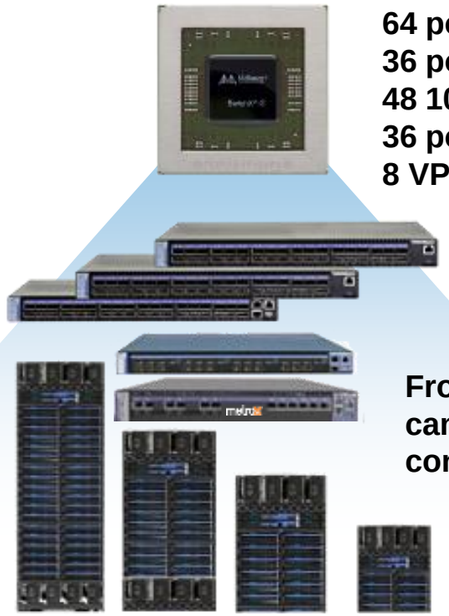
Acceleration Engines



SwitchX®-2 VPI Switch

Unified Fabric Manager

Switch OS Layer



64 ports 10GbE
36 ports 40/56GbE
48 10GbE + 12 40/56GbE
36 ports IB up to 56Gb/s
8 VPI subnets

From data center to campus and metro connectivity

Standard Protocols of InfiniBand and Ethernet on the Same Wire!

Mellanox ScalableHPC Communication Library to Accelerate Applications

MPI

OpenSHMEM / PGAS

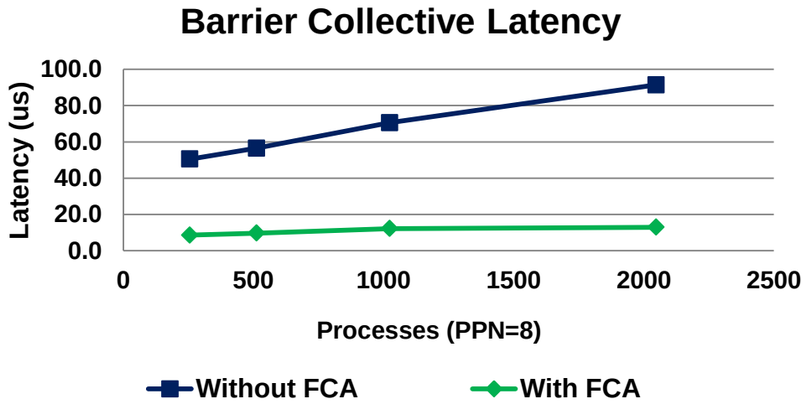
Berkeley UPC

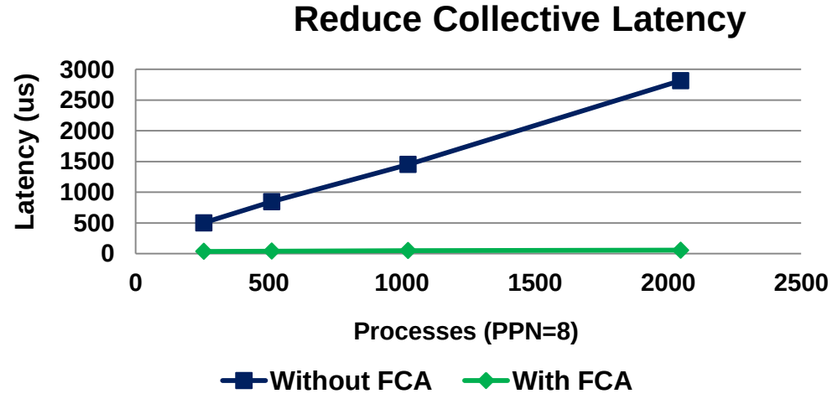
MXM

- Reliable Messaging
- Hybrid Transport Mechanism
- Efficient Memory Registration
- Receive Side Tag Matching

FCA

- Topology Aware Collective Optimization
- Hardware Multicast
- Separate Virtual Fabric for Collectives
- CORE-Direct Hardware Offload





Overview

- Extreme scale programming-model challenges
- Challenges for scaling OpenSHMEM to extreme scale
- InfiniBand enhancements
  - Dynamically Connected Transport (DC)
  - Cross-Channel synchronization
  - Non-contiguous data transfer
  - On Demand Paging
- Mellanox ScalableSHMEM

Exascale-Class Computer Platforms – Communication Challenges

- Very large functional unit count ~10,000,000
  - Implication to communication stack: Need scalable communication capabilities
    - Point-to-point
    - Collective
- Large on-”node” functional unit count ~500
  - Implication to communication stack: Scalable HCA architecture
- Deeper memory hierarchies
  - Implication to communication stack: Cache aware network access
- Smaller amounts of memory per functional unit
  - Implication to communication stack: Low latency, high b/w capabilities
- May have functional unit heterogeneity
  - Implication to communication stack: Support for data heterogeneity
- Component failures part of “normal” operation
  - Implication to communication stack: Resilient and redundant stack
- Data movement is expensive
  - Implication to communication stack: Optimize data movement

Challenges of a Scalable OpenSHMEM

- Scalable Communication interface
  - Interface objects scale in less than system dimension
  - Not a current issue, but keep in mind moving forward
- Support for asynchronous communication
  - Non-blocking communication interfaces
  - Sparse and topology aware interfaces ?
- System noise issues
  - Interfaces that support communication delegation (e.g., offloading)
- Minimize data motion
  - Semantics that support data “aggregation”

Scalable Performance Enhancements

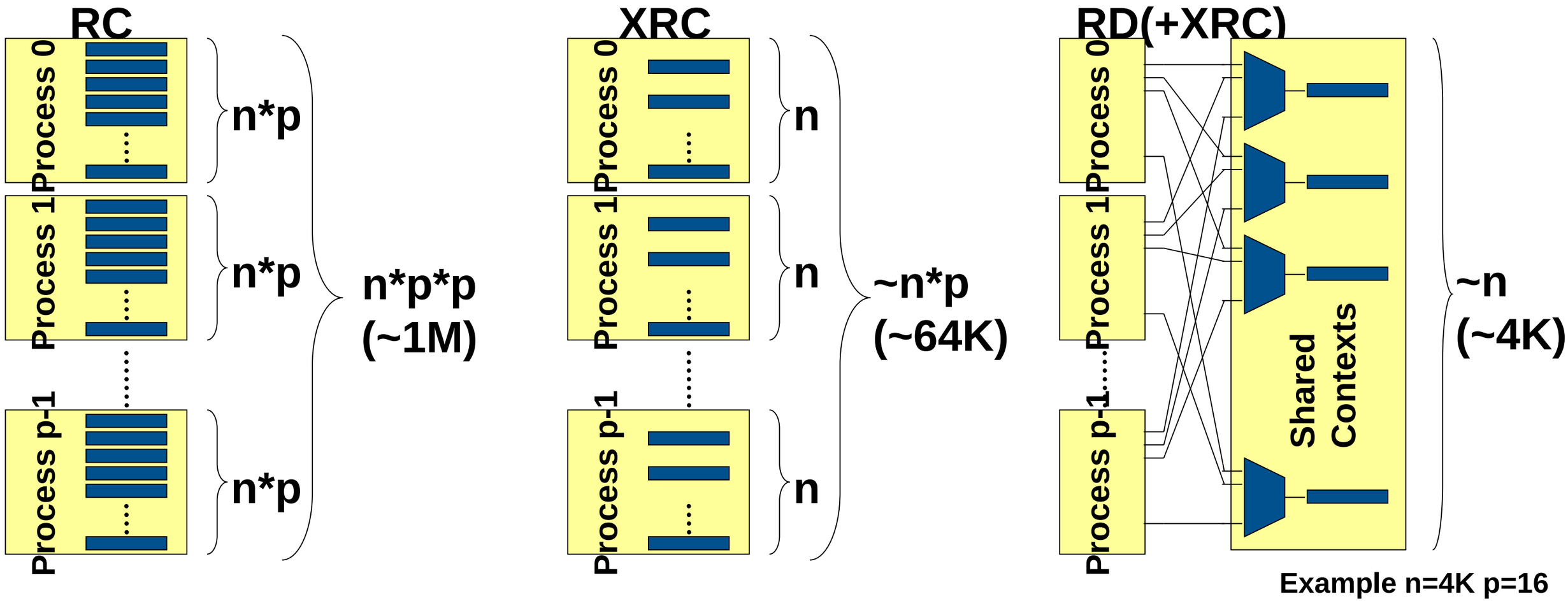
Dynamically Connected Transport

A Scalable Transport

New Transport

- **Challenges being addressed**:
  - Scalable communication protocol
  - High-performance communication
  - Asynchronous communication
- Current status: Transports in widest use
  - RC
    - High Performance: Supports RDMA and Atomic Operations
    - Scalability limitations: One connection per destination
  - UD
    - Scalable: One QP services multiple destinations
    - Limited communication support: No support for RDMA and Atomic Operations, unreliable
- Need scalable transport that also supports RDMA and Atomic operations ➔ DC – The best of both worlds
  - High Performance: Supports RDMA and Atomic Operations, Reliable
  - Scalable: One QP services multiple destinations

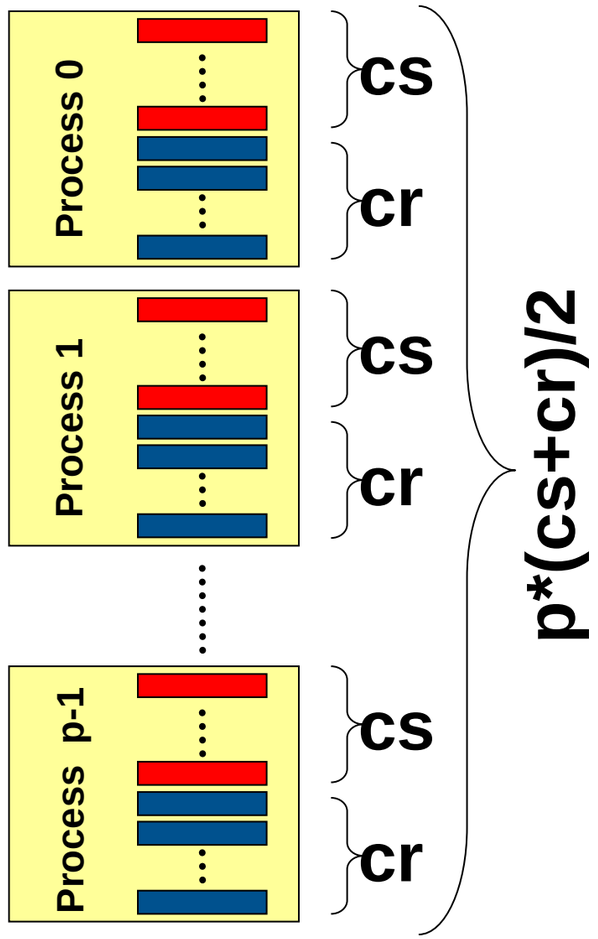
IB Reliable Transports Model



- QoS/Multipathing: 2 to 8 times the above
- Resource sharing (XRC/RD) causes processes to impact each-other

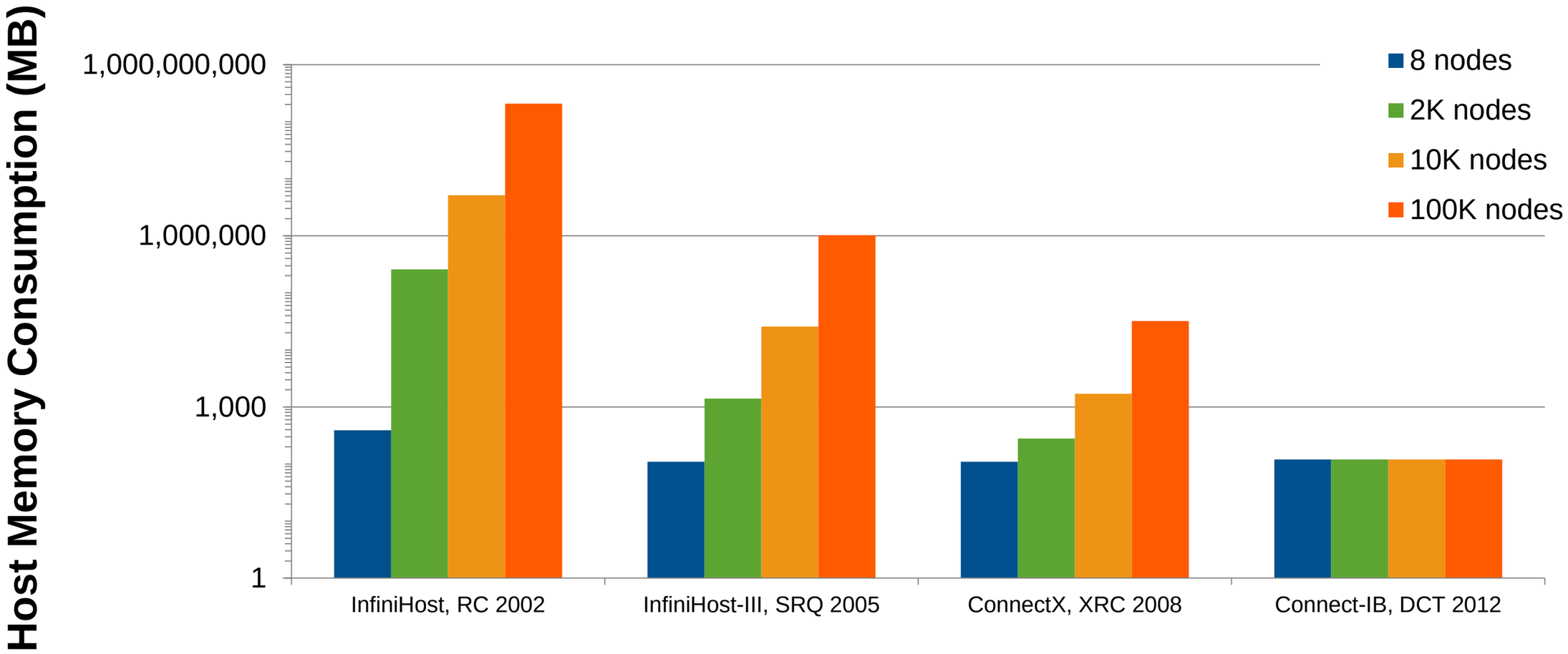
The DC Model

- Dynamic Connectivity
- Each DC Initiator can be used to reach any remote DC Target
- No resources' sharing between processes
  - process controls how many (and can adapt to load)
  - process controls usage model (e.g. SQ allocation policy)
  - no inter-process dependencies
- Resource footprint
  - Function of HCA capability
  - Independent of system size
- Fast Communication Setup Time



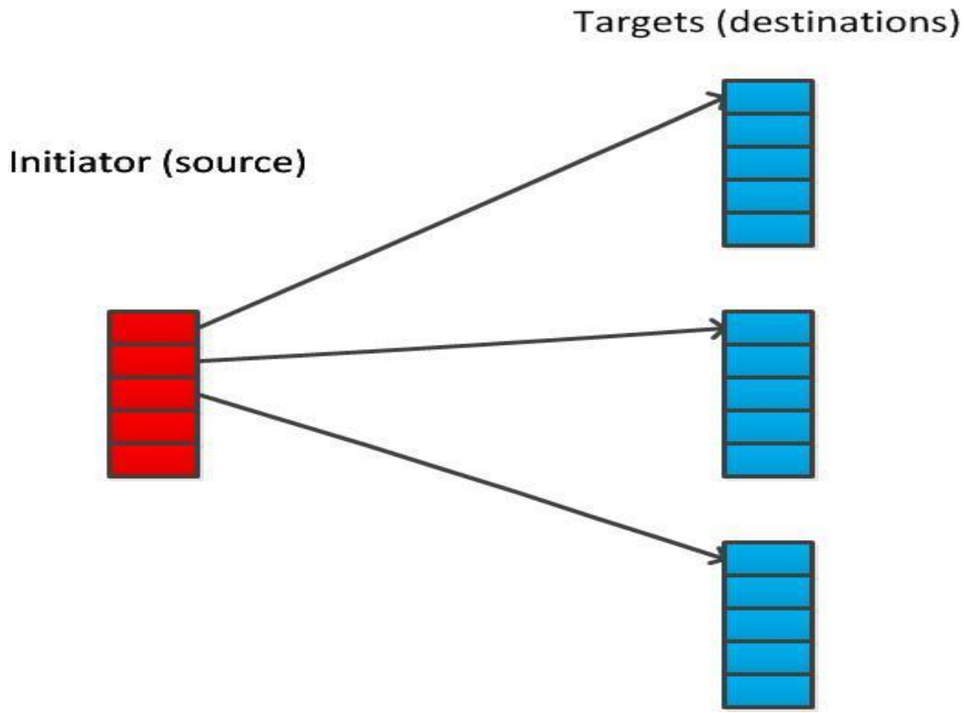
cs – concurrency of the sender
cr=concurrency of the responder

Connect-IB – Exascale Scalability

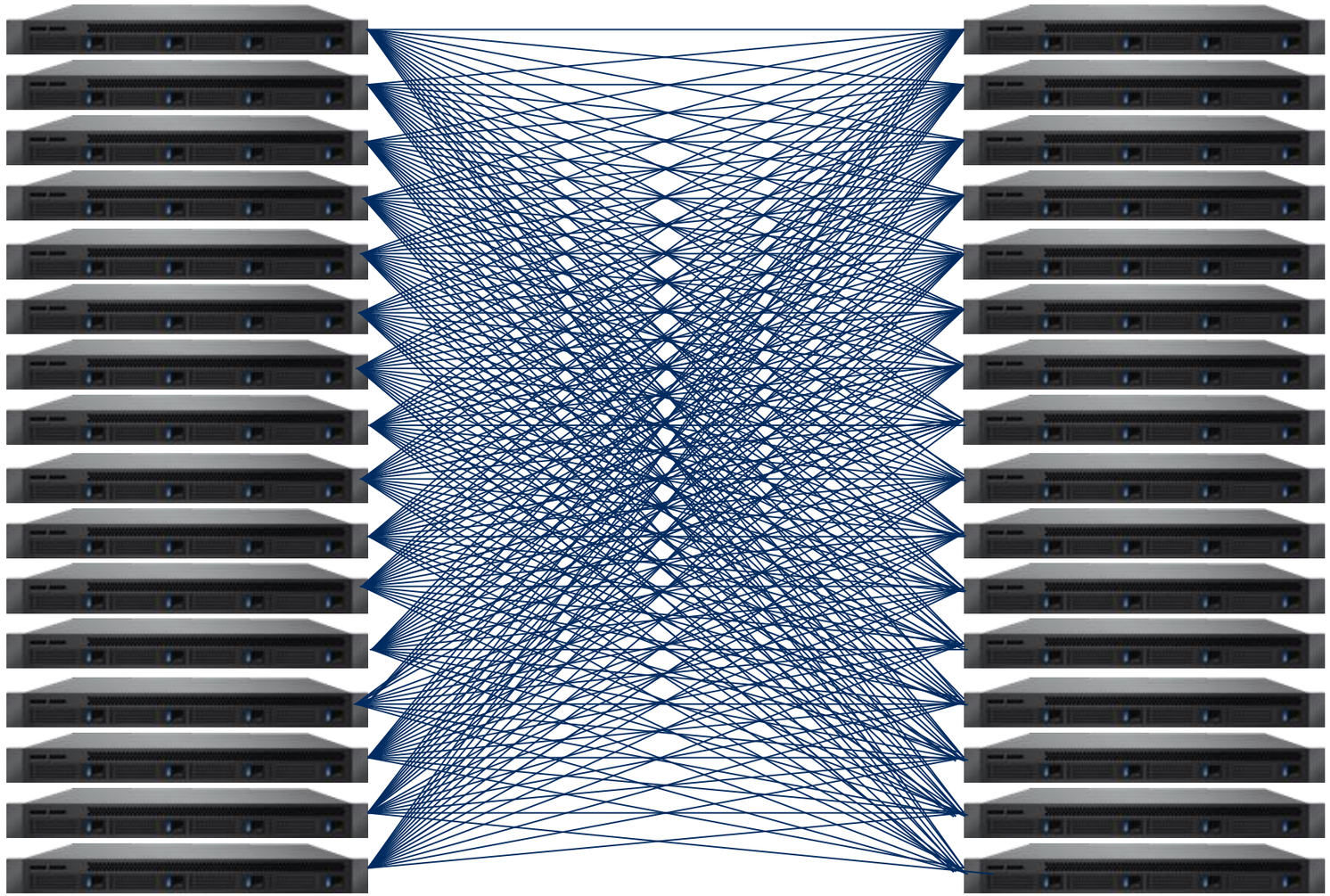


Dynamically Connected Transport

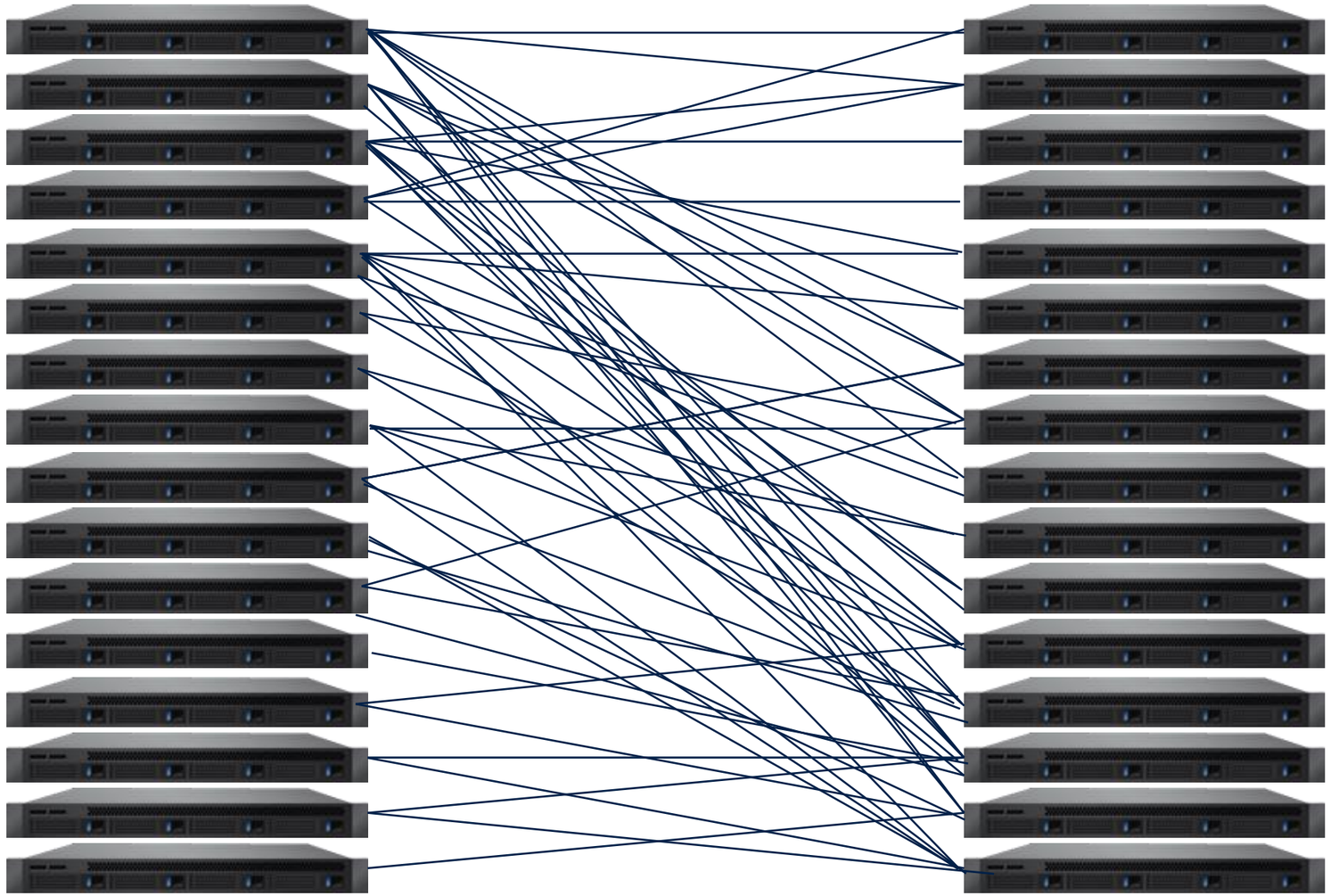
- Key objects
  - DC Initiator: Initiates data transfer
  - DC Target: Handles incoming data



Reliable Connection Transport Mode



Dynamically Connected Transport Mode



Cross-Channel Synchronization

Challenges Being Addressed

- Scalable Collective communication
- Asynchronous communication
- Communication resources (not computational resources) are used to manage communication
- Avoids some of the effects of system noise

High Level Objectives

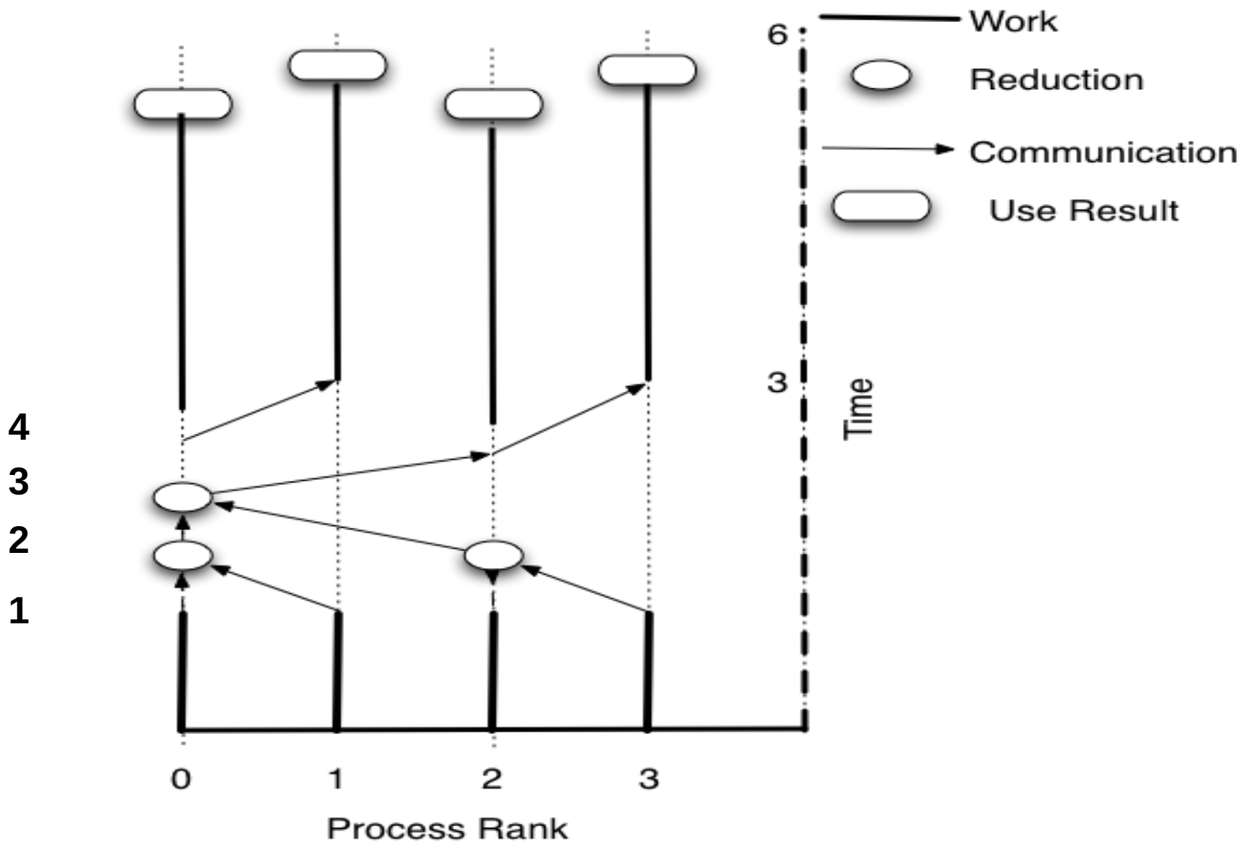
- Provide synchronization mechanism between QP's
- Provide mechanisms for managing communication dependencies to be managed without additional host intervention
- Support asynchronous progress of multi-staged communication protocols

Motivating Example - HPC

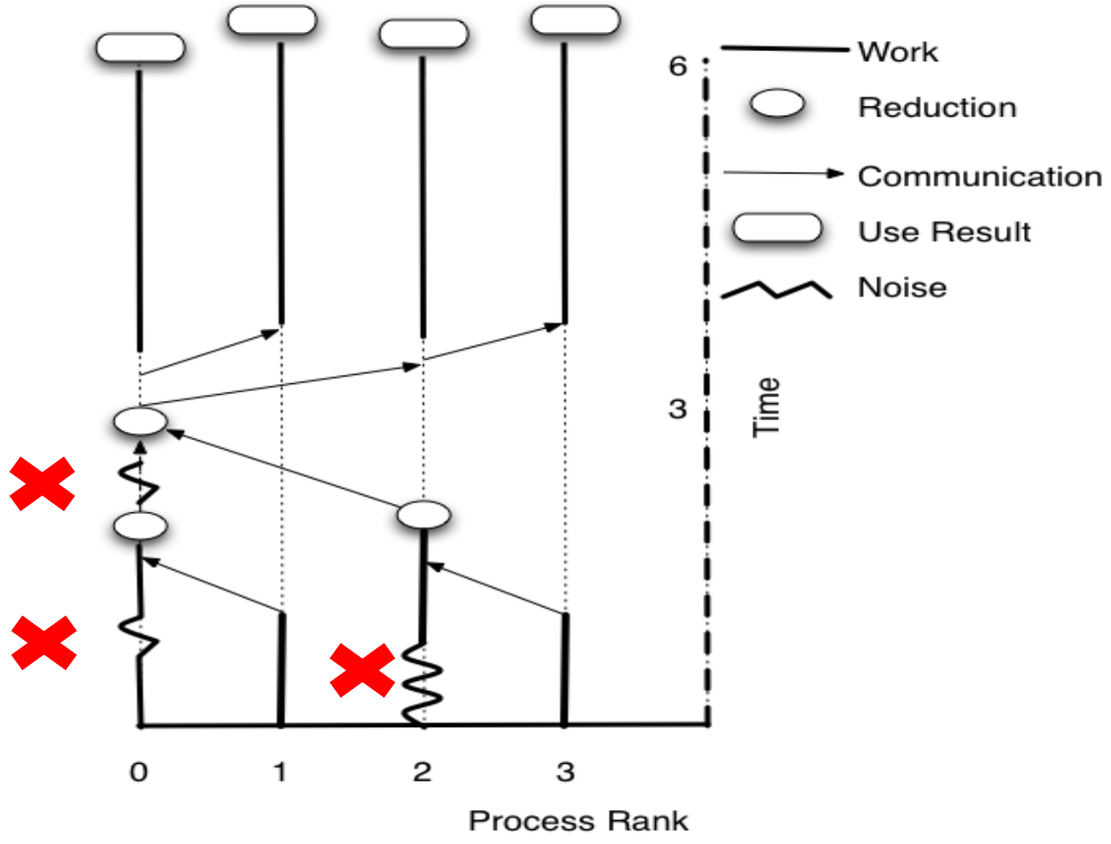
- Collective communications optimization
- Communication pattern involving multiple processes
- Optimized collectives involve a communicator-wide data-dependent communication pattern, e.g., communication initiation is dependent on prior completion of other communication operations
- Data needs to be manipulated at intermediate stages of a collective operation (reduction operations)
- Collective operations limit application scalability
  - Performance, scalability, and system noise

Scalability of Collective Operations

Ideal Algorithm

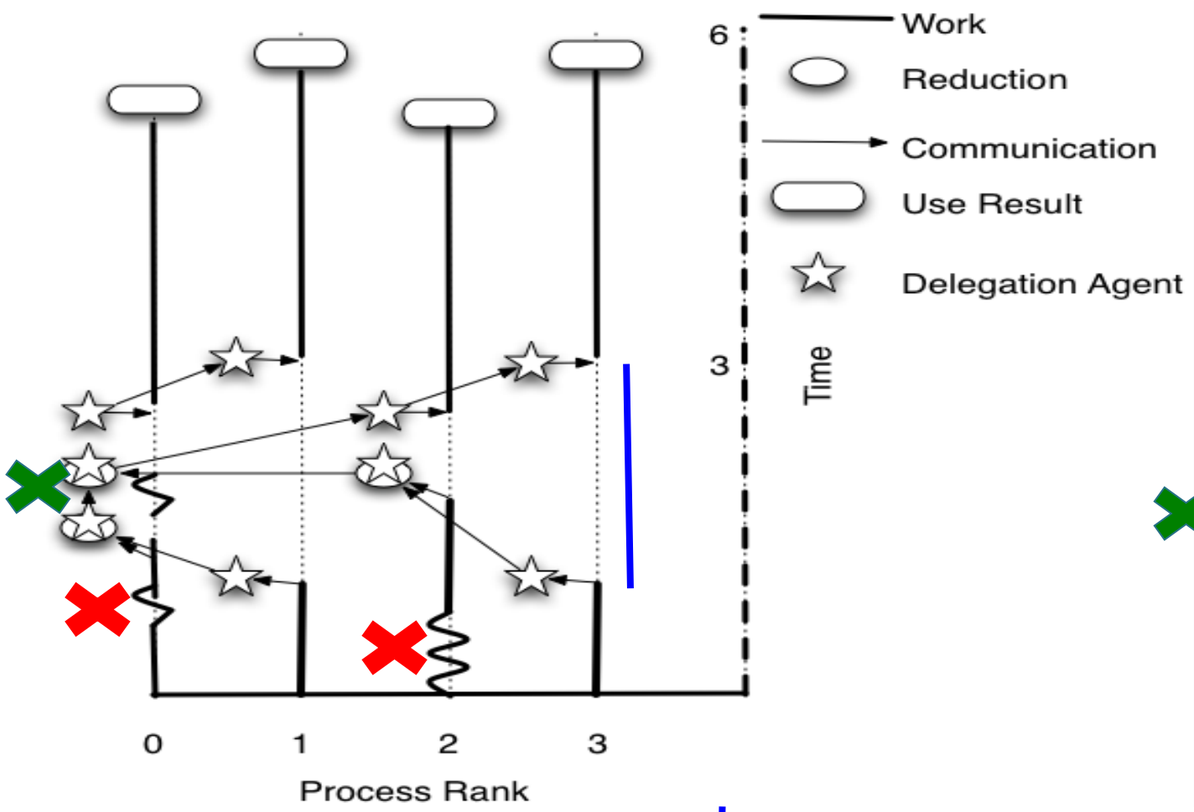


Impact of System Noise

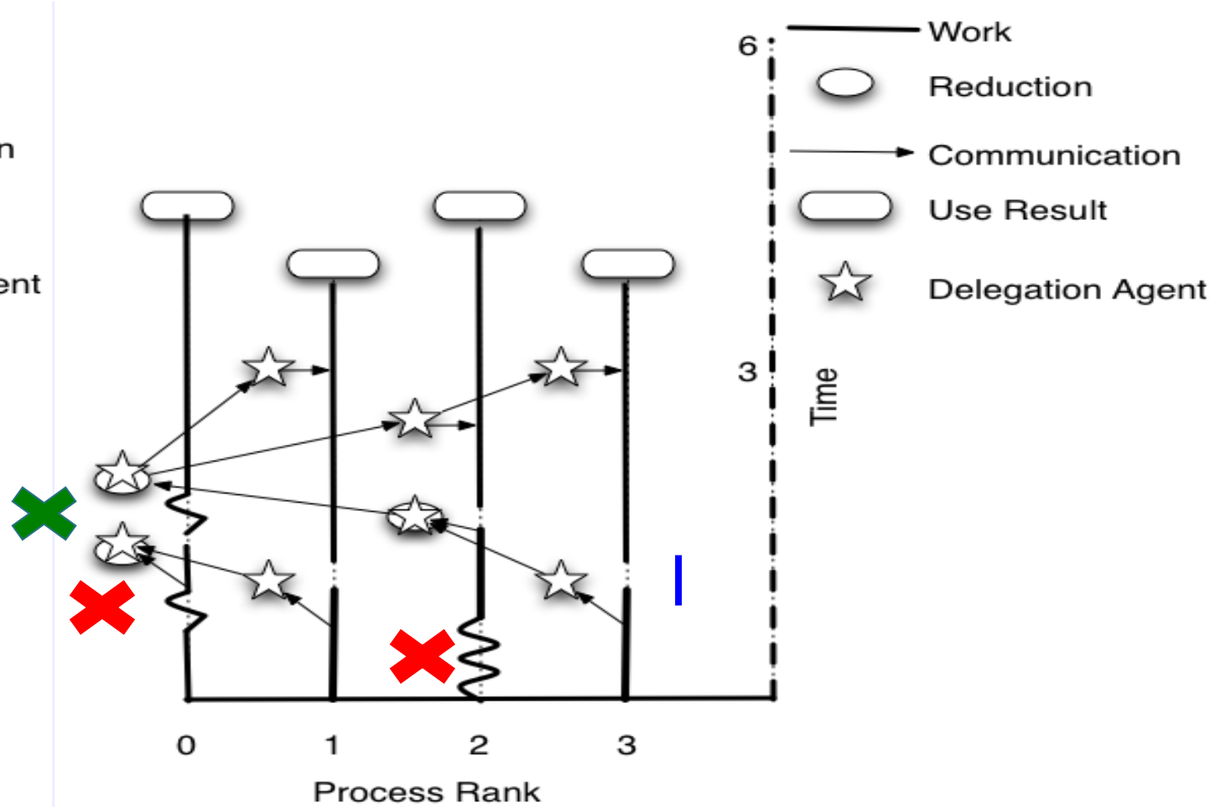


Scalability of Collective Operations - II

Offloaded Algorithm



Nonblocking Algorithm



- Communication processing

Network Managed Multi-stage Communication

- Key Ideas
  - Create a local description of the communication pattern
  - Pass the description to the communication subsystem
  - Manage the communication operations on the network, freeing the CPU to do meaningful computation
  - Check for full-operation completion

- Current Assumptions
  - Data delivery is detected by new Completion Queue Events
  - Use Completion Queue to identify the data source
  - Completion order is used to associate data with a specific operation
  - Use RDMA with the immediate to generate Completion Queue events

Key New Features

- New QP trait - Managed QP: WQE on such a QP must be enabled by WQEs from other QP's
- Synchronization primitives:
  - Wait work queue entry: waits until specified completion queue (QP) reaches specified producer index value
  - Enable tasks: WQE on one QP can "enable" a WQE on a second QP
- Submit lists of task to multiple QP's in single post - sufficient to describe chained operations (such as collective communication)
- Can setup a special completion queue to monitor list completion (request CQE from the relevant task)

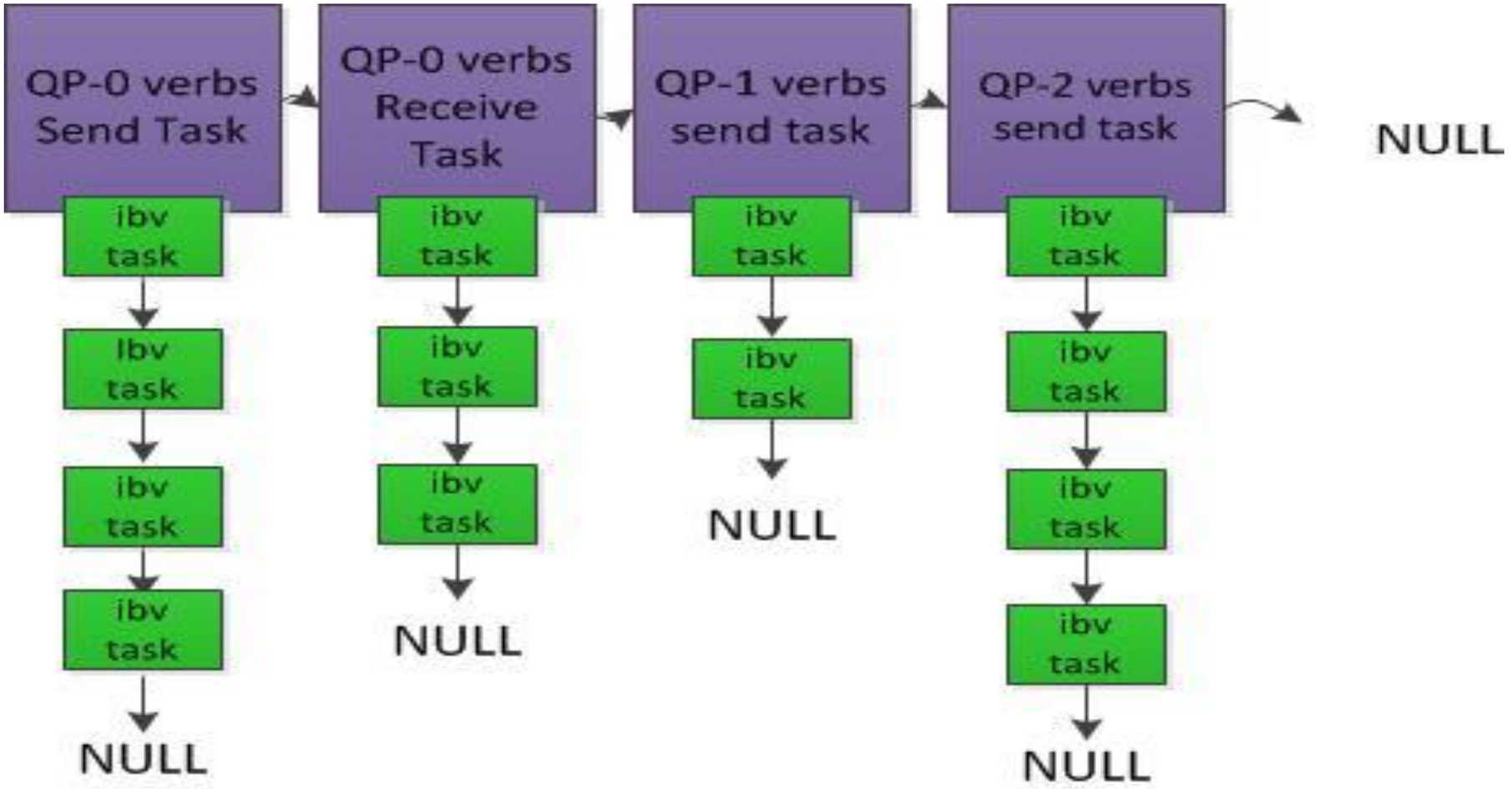
Setting up CORE-*Direct* QP's

- Create QP with the ability to use the CORE-Direct primitives
- Decide if managed QP's will be used, if so, need to create QP that will take the enable tasks. Most likely, this is a centralized resource handling both the enable and the wait tasks
- Decide on a Completion Queue strategy
- Setup all needed QP's

Initiating CORE-*Direct* Communication

- Task list is created
- Target QP for task
- Operation send/wait/enable
- For wait, the number of completions to wait for
  - Number of completions is specified relative to the beginning of the task list
  - Number of completions can be positive, zero, or negative (wait on previously posted tasks)
- For enable, the number of send tasks on the target QP is specified
  - Number of tasks to enable

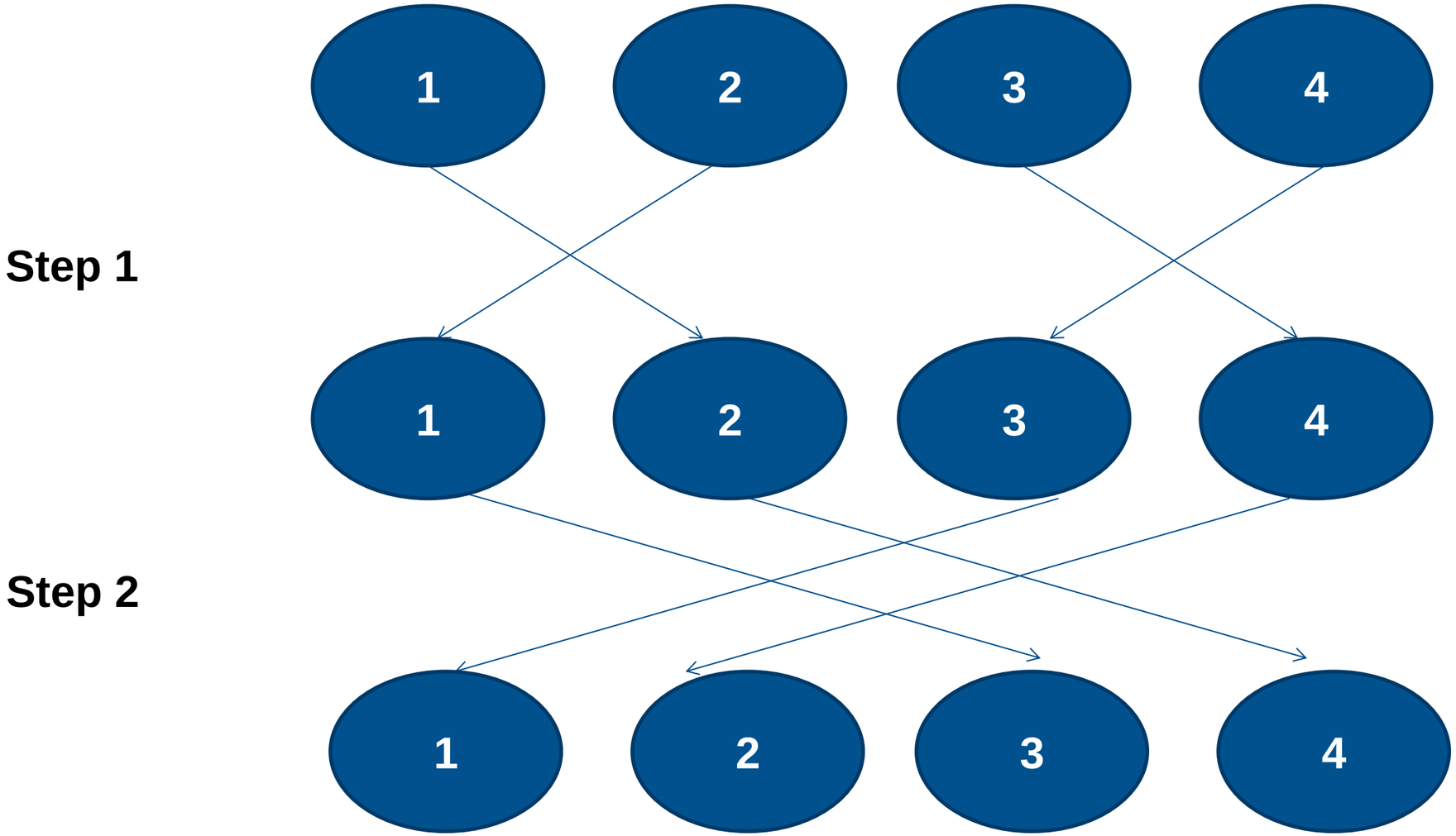
Posting of Tasks



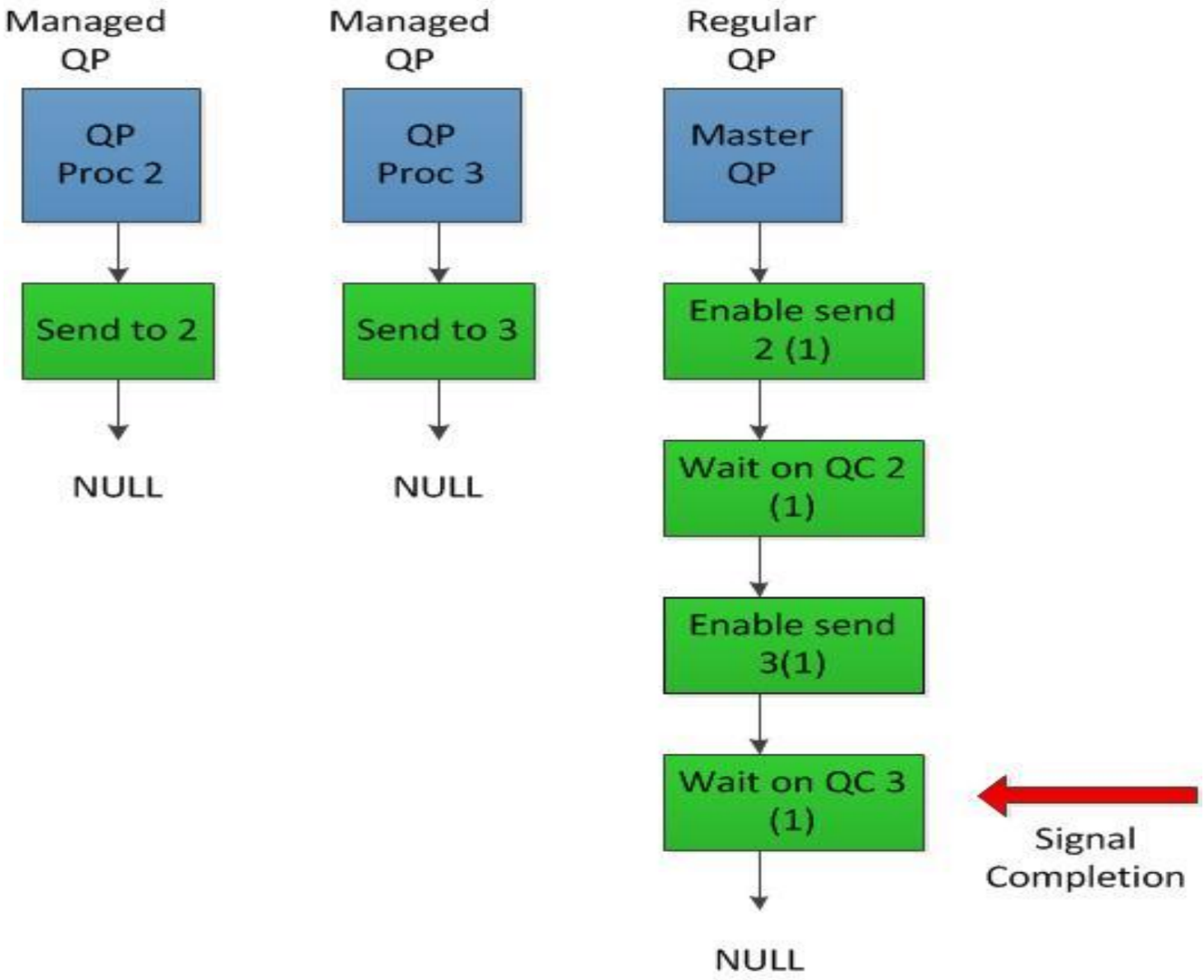
CORE-Direct Task-List Completion

- Can specify which task will generate completion, in “Collective” completion queue
- Single CQ signals full list (collective) completion
- CPU is not needed for progress

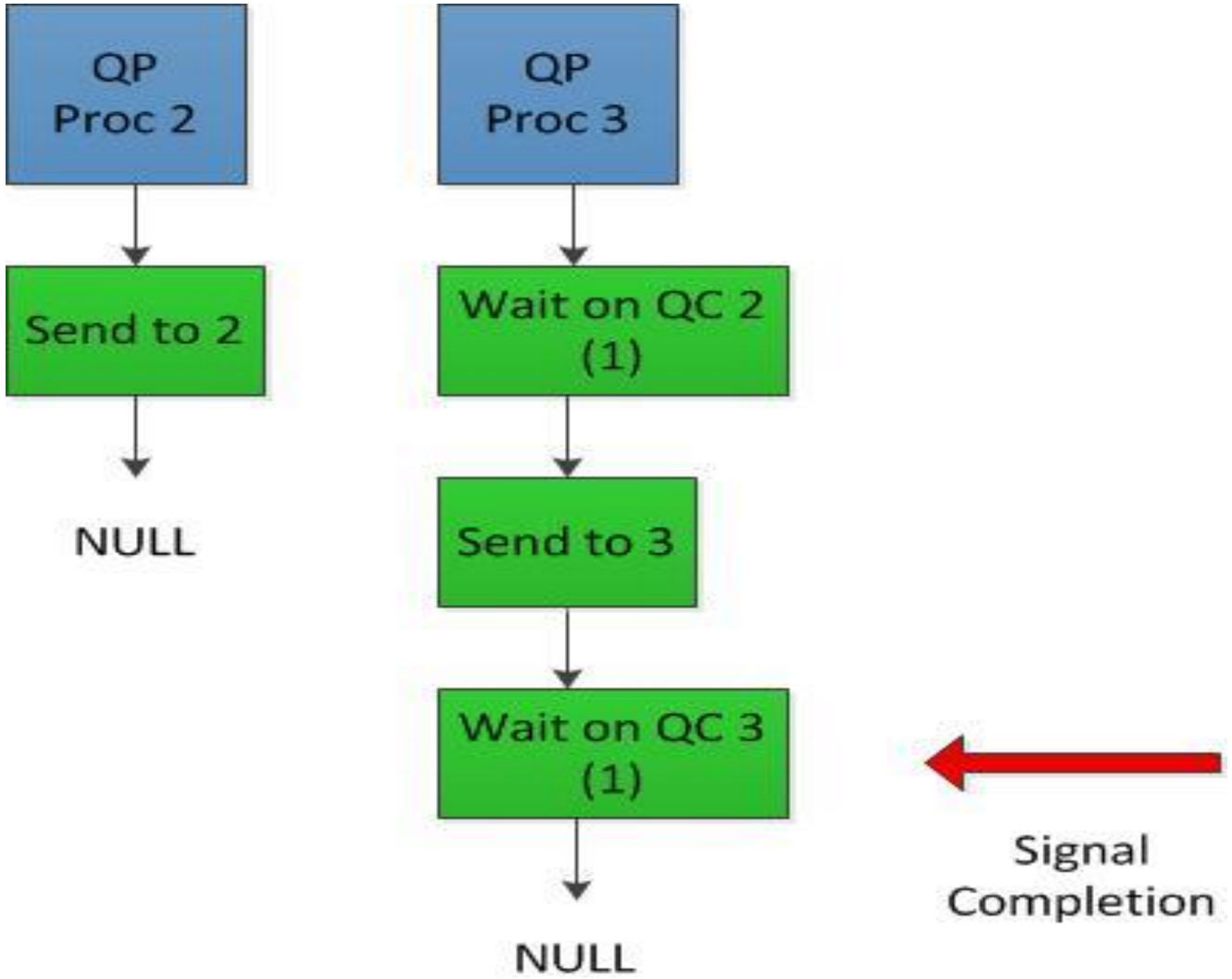
Example – Four Process Recursive Doubling



Four Process Barrier Example – Using Managed Queues – Rank 0



Four Process Barrier Example – No Managed Queues – Rank 0



User-Mode Memory Registration

Key Features

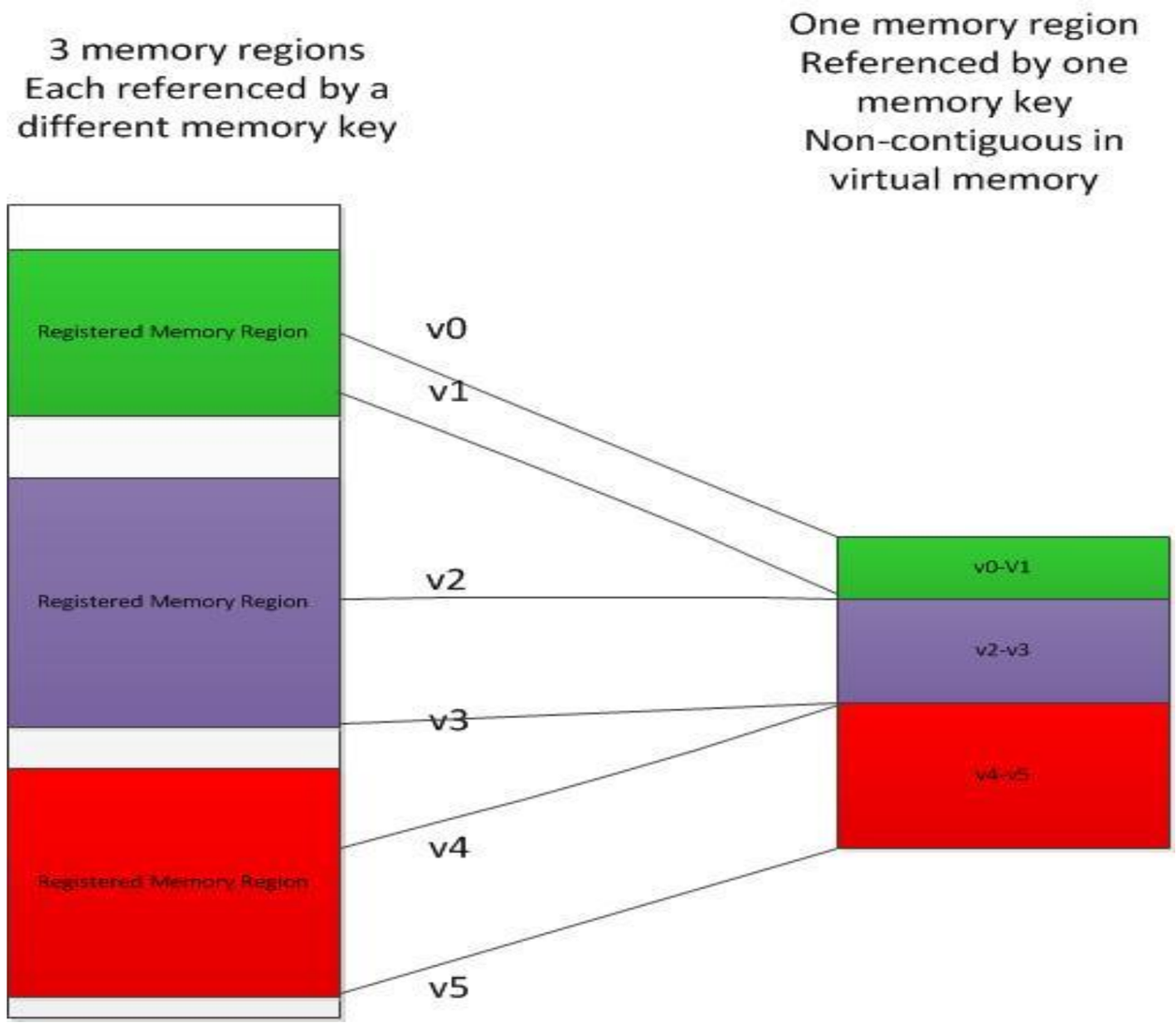
- Support combining contiguous registered memory regions into a single memory region. H/W treats them as a single contiguous region (and handles the non-contiguous regions)

- For a given memory region, supports non-contiguous access to memory, using a regular structure representation – base pointer, element length, stride, repeat count.
  - Can combine these from multiple different memory keys

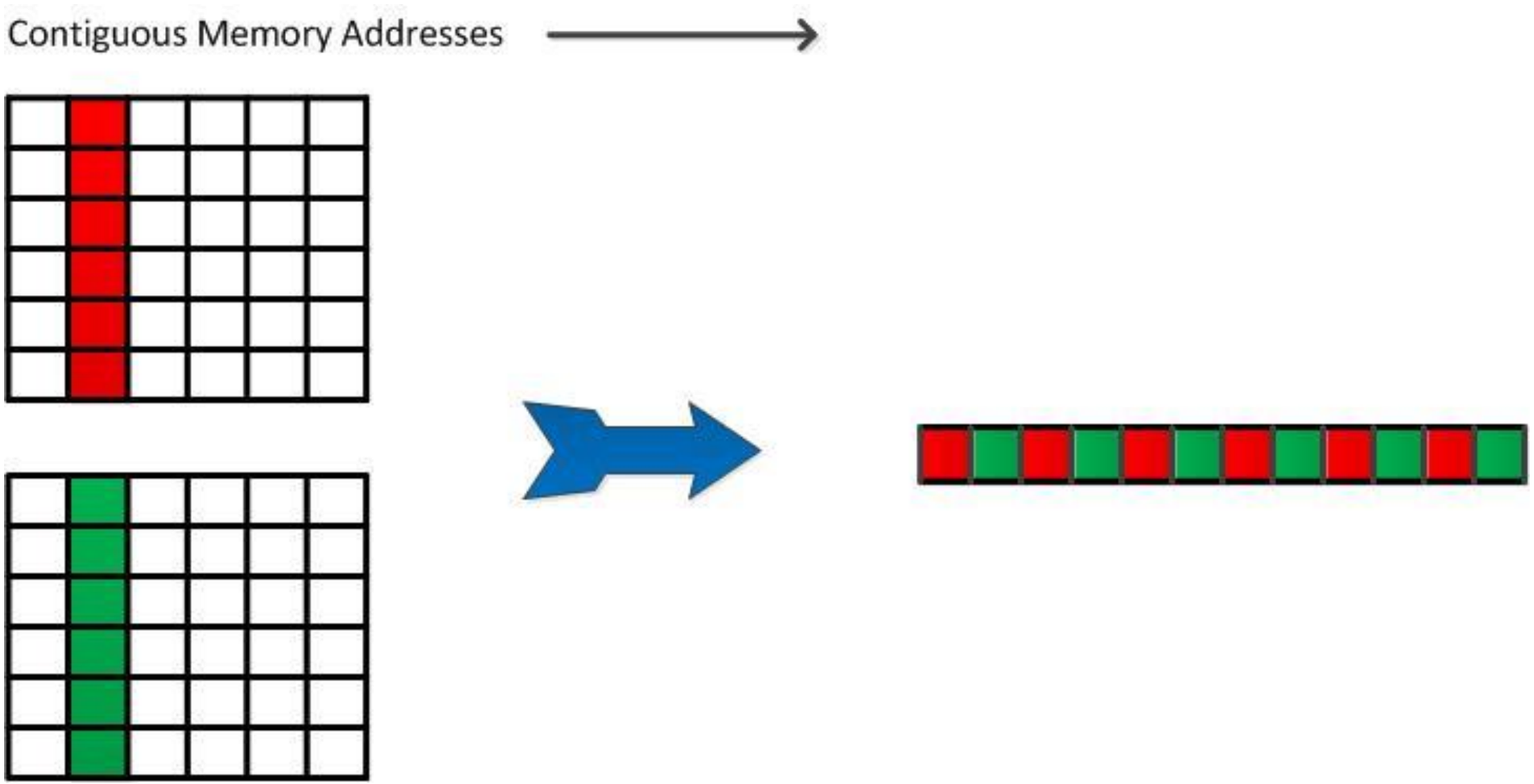
- Memory descriptors are created by posting WQE's to fill in the memory key

- Supports local and remote non-contiguous memory access
  - Eliminates the need for some memory copies

Combining Contiguous Memory Regions



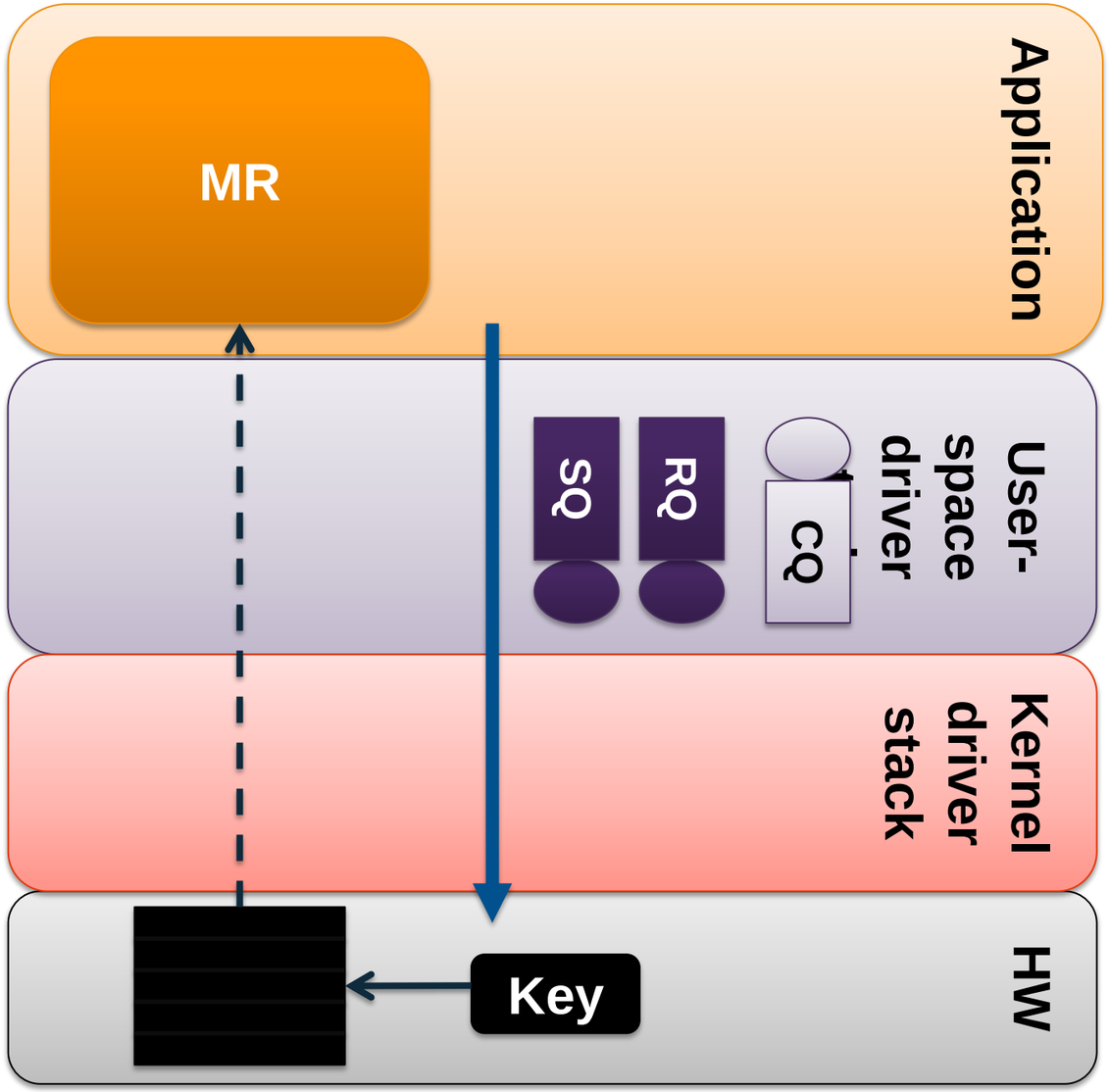
Non-Contiguous Memory Access – Regular Access



On-Demand Paging

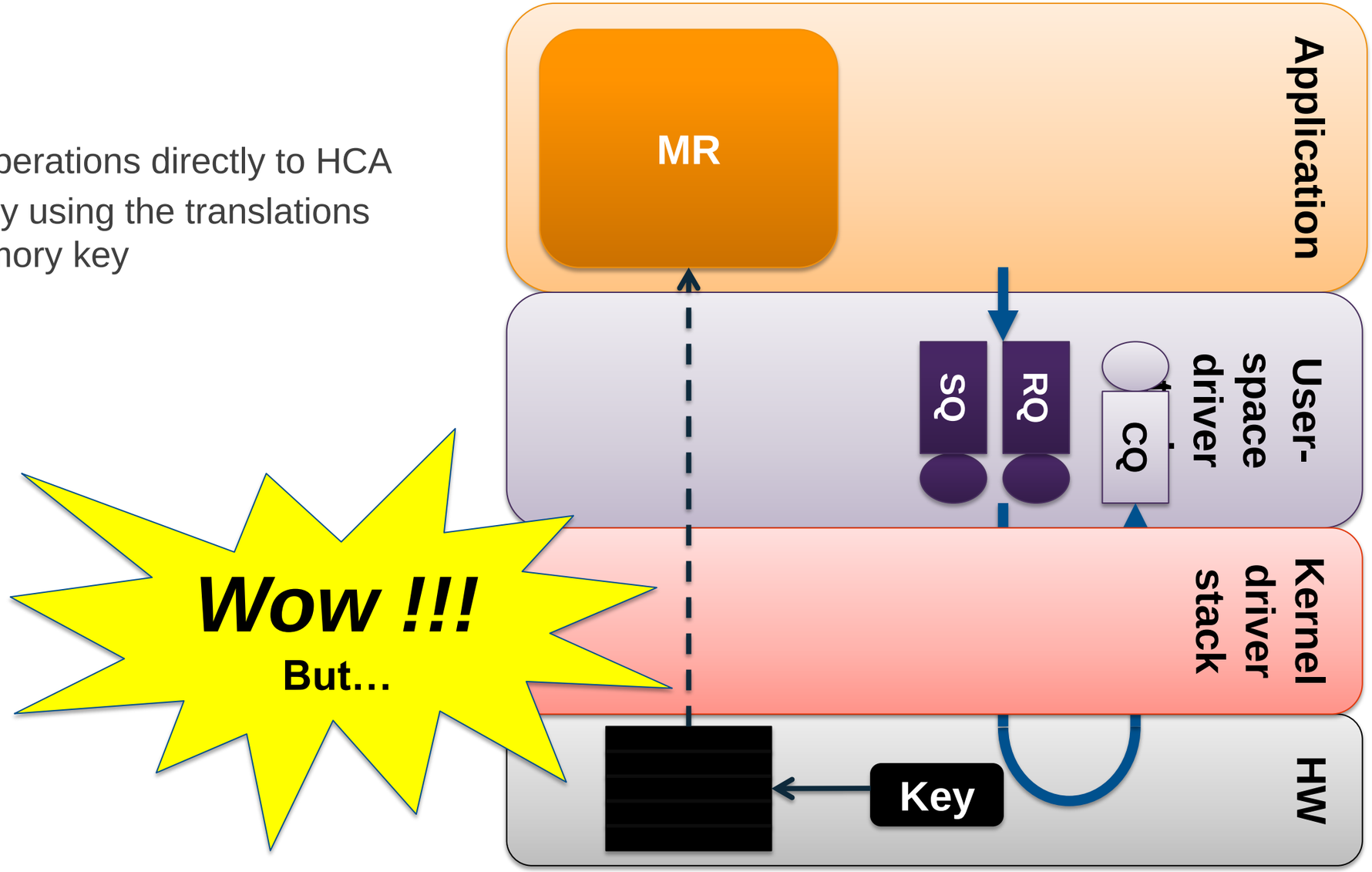
Memory Registration

- Apps register Memory Regions (MRs) for IO
  - Referenced memory must be part of process address space at registration time
  - Memory key returned to identify the MR
- Registration operation
  - Pins down the MR
  - Hands off the virtual to physical mapping to HW



Memory Registration – continued

- Fast path
  - Applications post IO operations directly to HCA
  - HCA accesses memory using the translations referenced by the memory key



Challenges

- Size of registered memory must fit physical memory
- Applications must have memory locking privileges
- Continuously synchronizing the translation tables between the address space and the HCA is hard
  - Address space changes (malloc, mmap, stack)
  - NUMA migration
  - fork()
- Registration is a costly operation
  - No locality of reference

Achieving High Performance

- Requires careful design
- Dynamic registration
  - Naïve approach induces significant overheads
  - Pin-down cache logic is complex and not complete
- Pinned bounce buffers
  - Application level memory management
  - Copying adds overhead

On Demand Paging

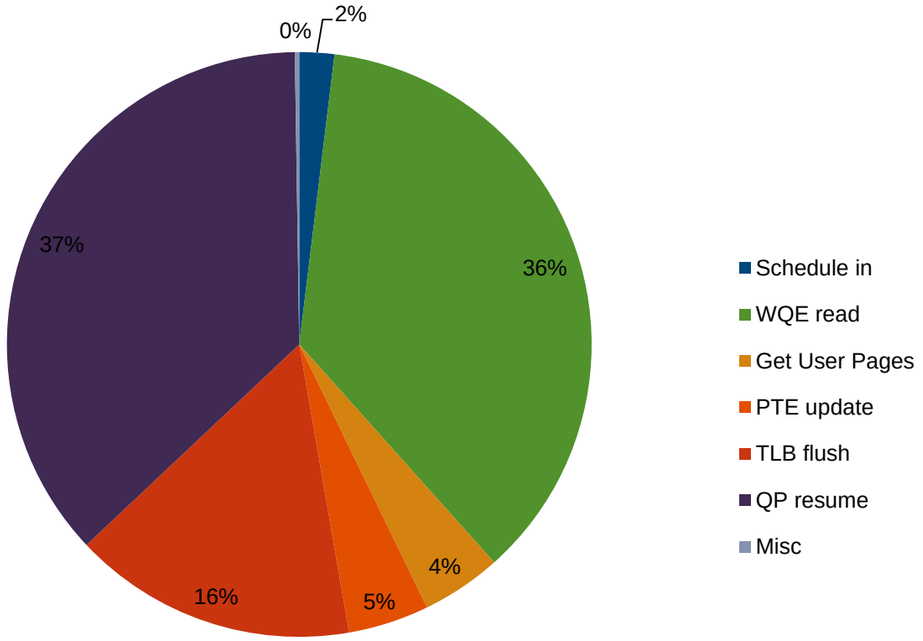
- MR pages are never pinned by the OS
  - Paged in when HCA needs them
  - Paged out when reclaimed by the OS
- HCA translation tables may contain non-present pages
  - Initially, a new MR is created with non-present pages
  - Virtual memory mappings don’t necessarily exist

Semantics

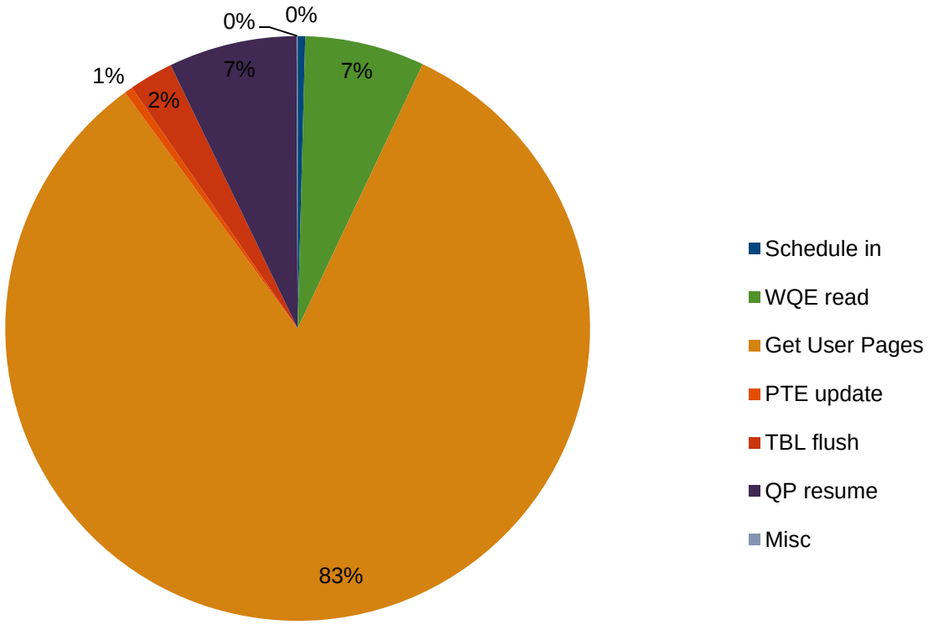
- ODP memory registration
  - Specify IBV_ACCESS_ON_DEMAND access flag
- Work request processing
  - WQEs in HW ownership must reference mapped memory
    - From Post_Send()/Recv() until PollCQ()
  - RDMA operations must target mapped memory
  - Access attempts to unmapped memory trigger an error
- Transport
  - RC semantics unchanged
  - UD responder drops packets while page faults are resolved
    - Standard semantics cannot be achieved unless wire is back-pressured

Execution Time Breakdown (Send Requestor)

4K Page fault (135us total time)



4M Page fault (1ms total time)



Mellanox ScalableSHMEM

Mellanox's OpenSHMEM Implementation

- Implemented within the context of the Open MPI project
- Exploit's Open MPI's component architecture, re-using components used for the MPI implementation
- Adds OpenSHMEM specific components
- Uses InifiniBand optimized point-to-point and Collective communication modules
  - Hardware Multicast
  - CORE-Direct
  - Dynamically Connected Transport
  - Enhanced Hardware scatter/gather (UMR) – coming soon
  - On Demand Paging – coming soon

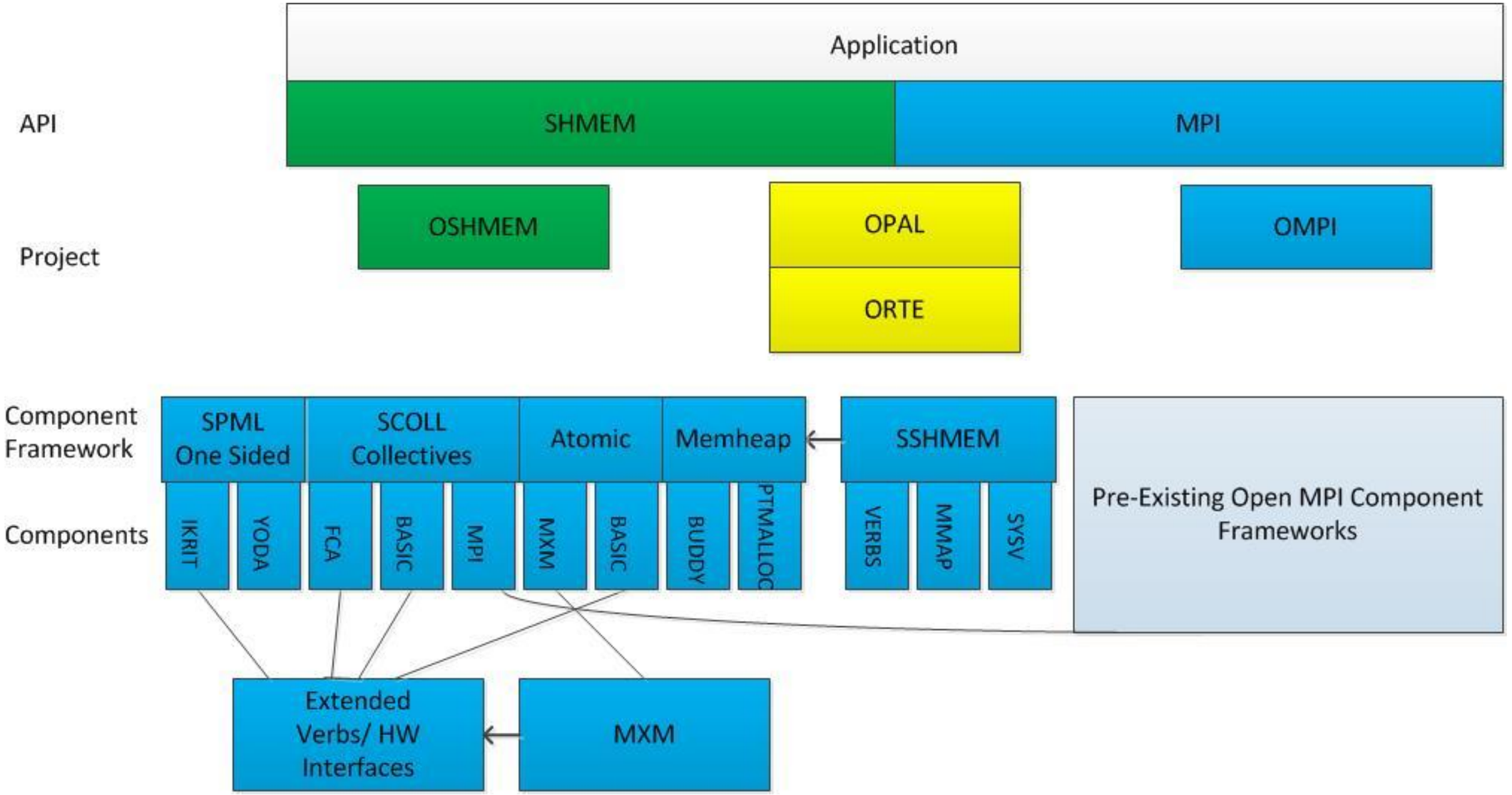
Leveraging the work already done for MPI implementation

- Similarities
  - Atomics, collectives operations
  - one-sided operations (put/get)
  - Job start and runtime support (mapping/binding/…)
- Differences
  - SPMD
  - No communicators (yet)
  - No user-defined data types
  - Limited set of collectives, and slightly different semantics (Barrier)
  - Application can put/get data from pre-allocated heap or static variables
  - No file I/O support

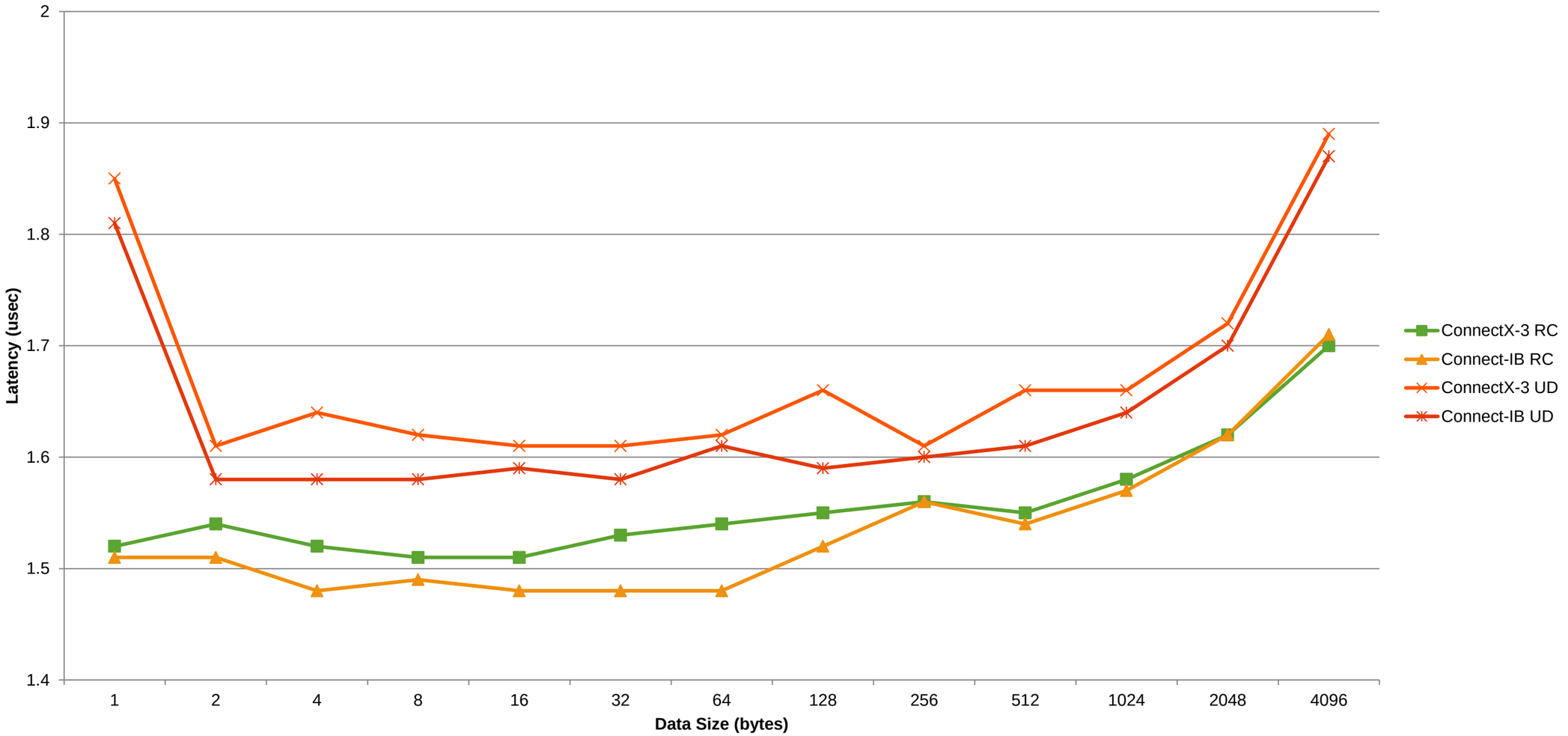
OMPI + OSHMEM

- Many OMPI frameworks reused (runtime, platform support, job start, BTL, BML, MTL, profiling, autotools)
- OSHMEM specific frameworks added, keeping MCA plugin architecture (SCOLL, SPML, atomics, synchronization and ordering enforcement)
- OSHMEM supports Mellanox p2p and collectives accelerators (MXM, FCA) as long as OMPI provided transports (TCP, OpenIB, Portals, …)

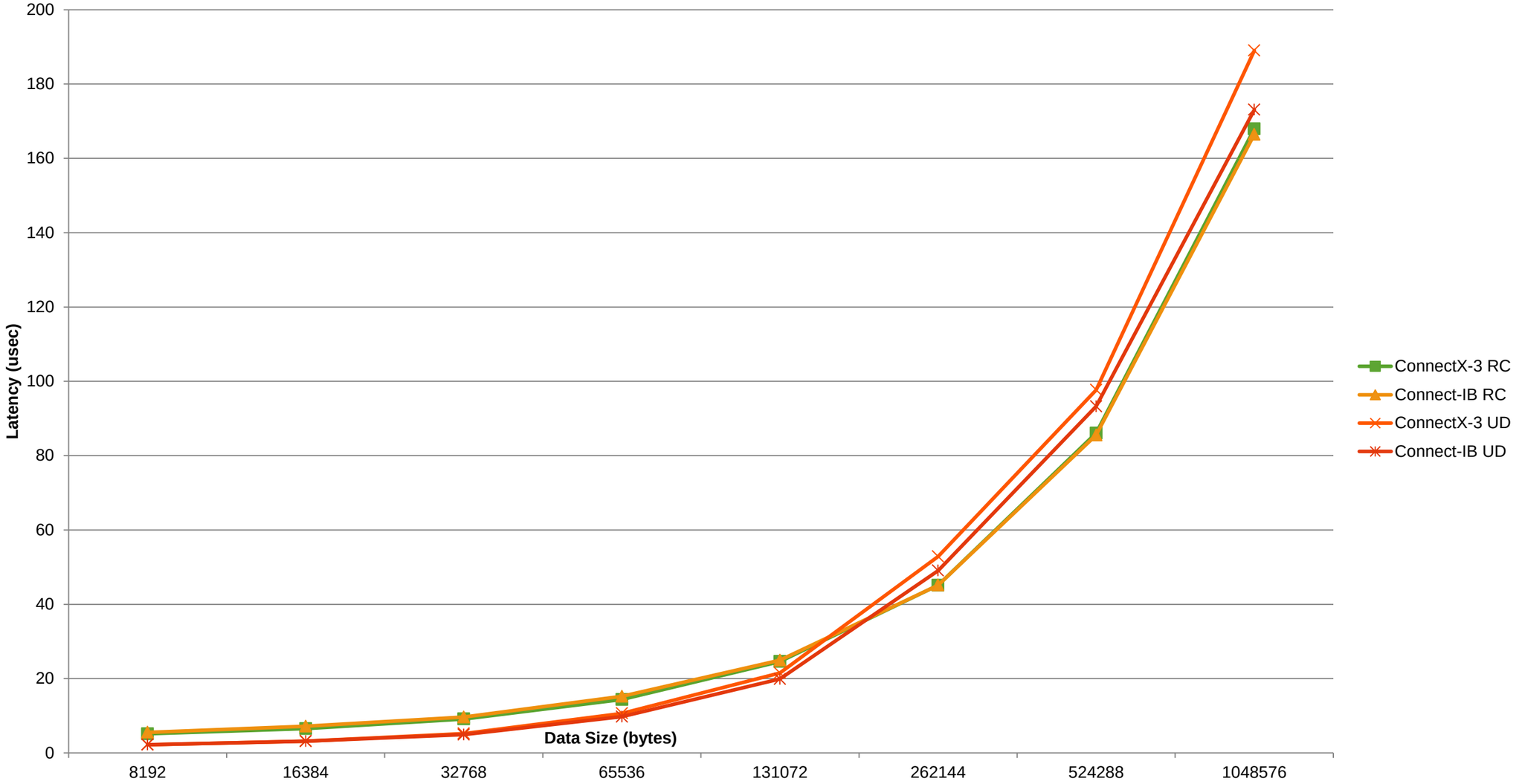
SHMEM Implementation Architecture



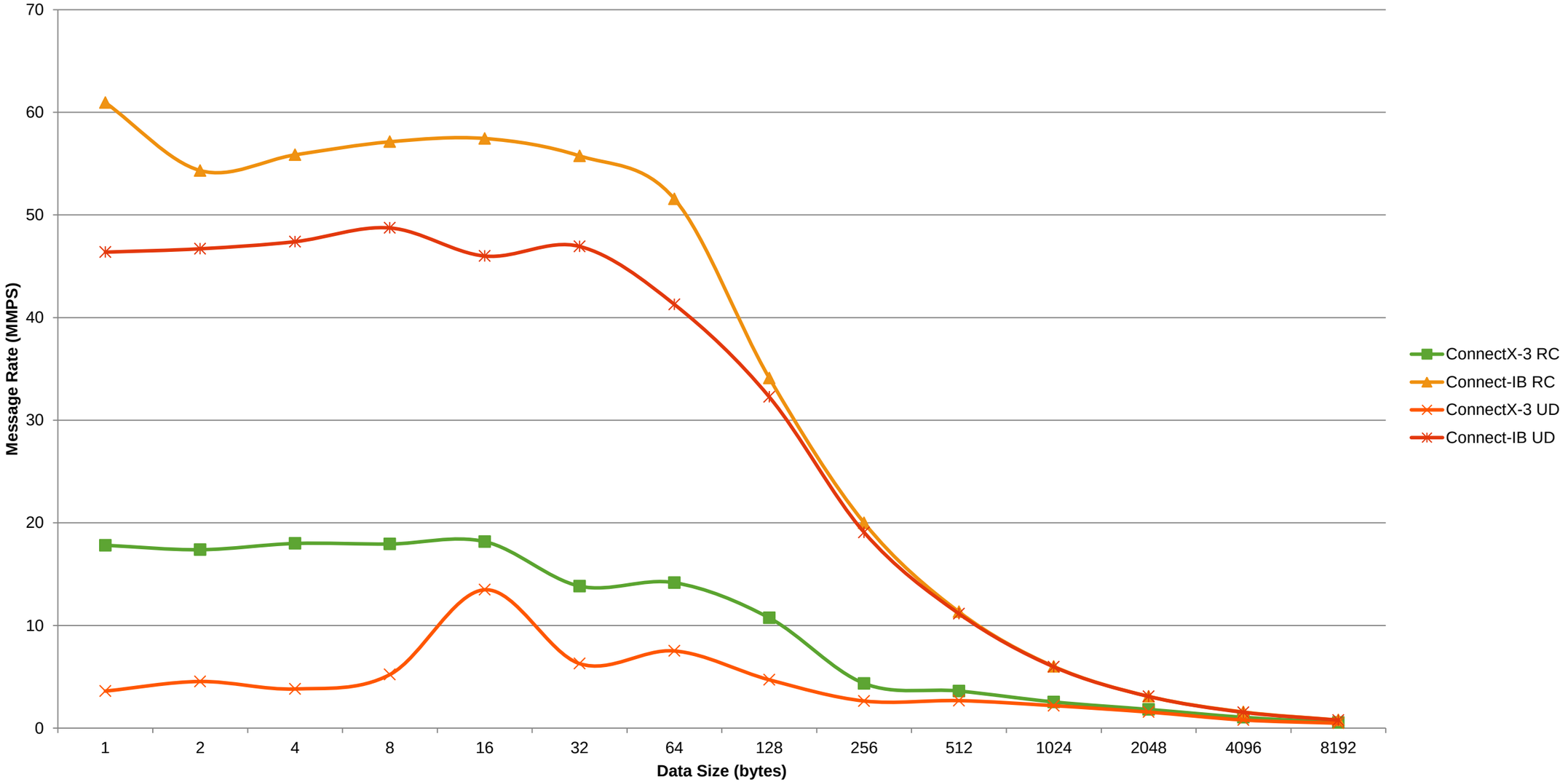
SHMEM Put Latency



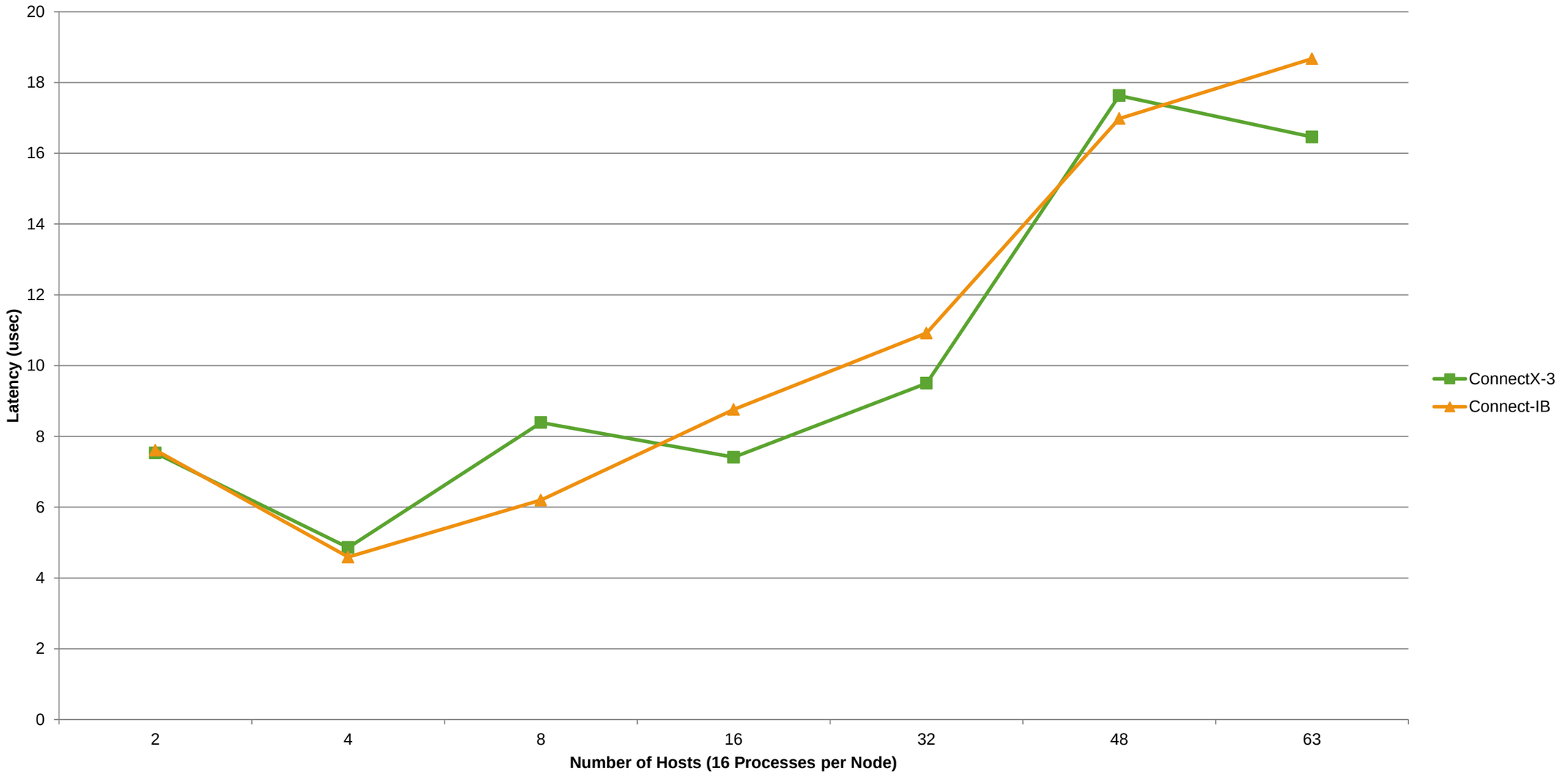
SHMEM Put Latency



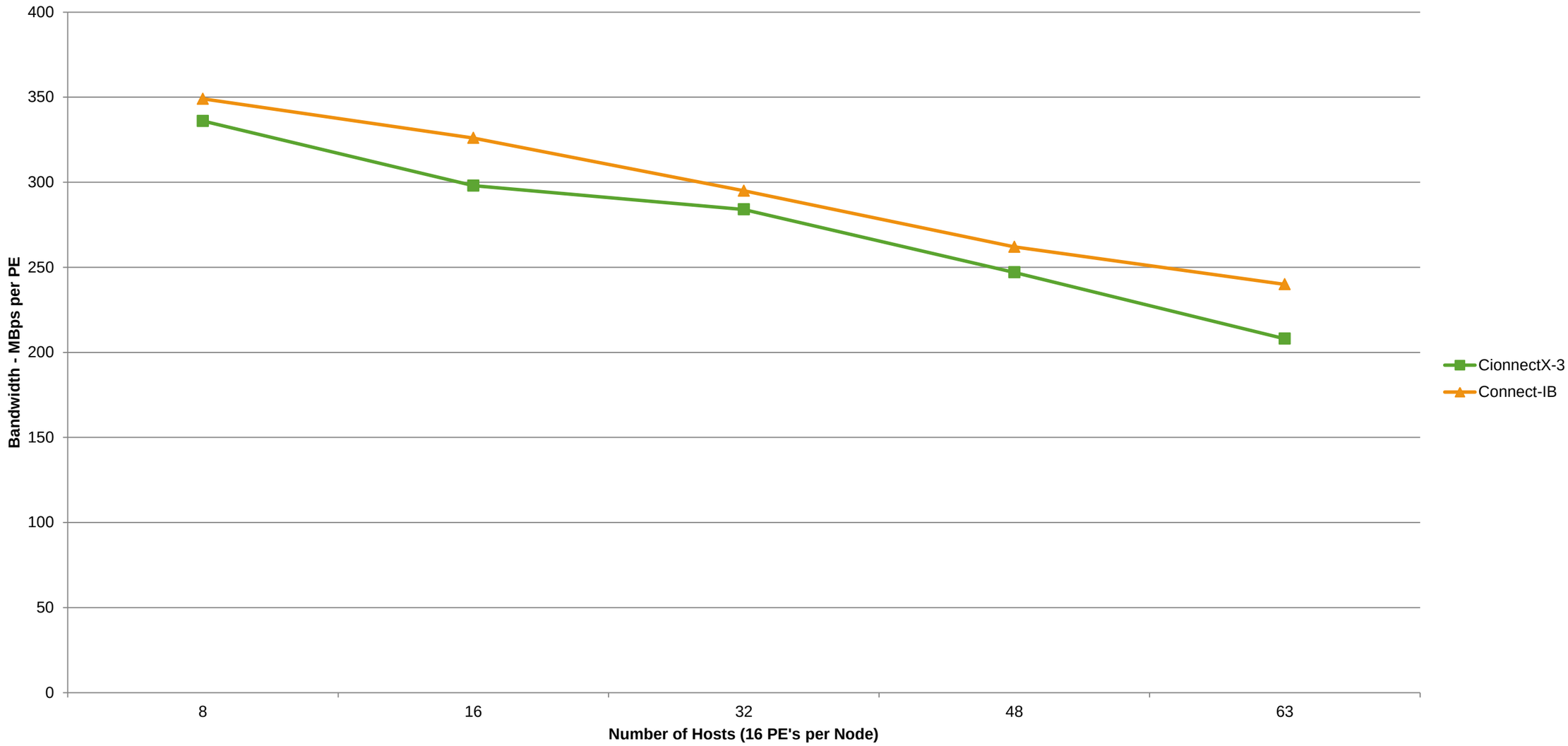
‘SHMEM 8 Byte Message Rate



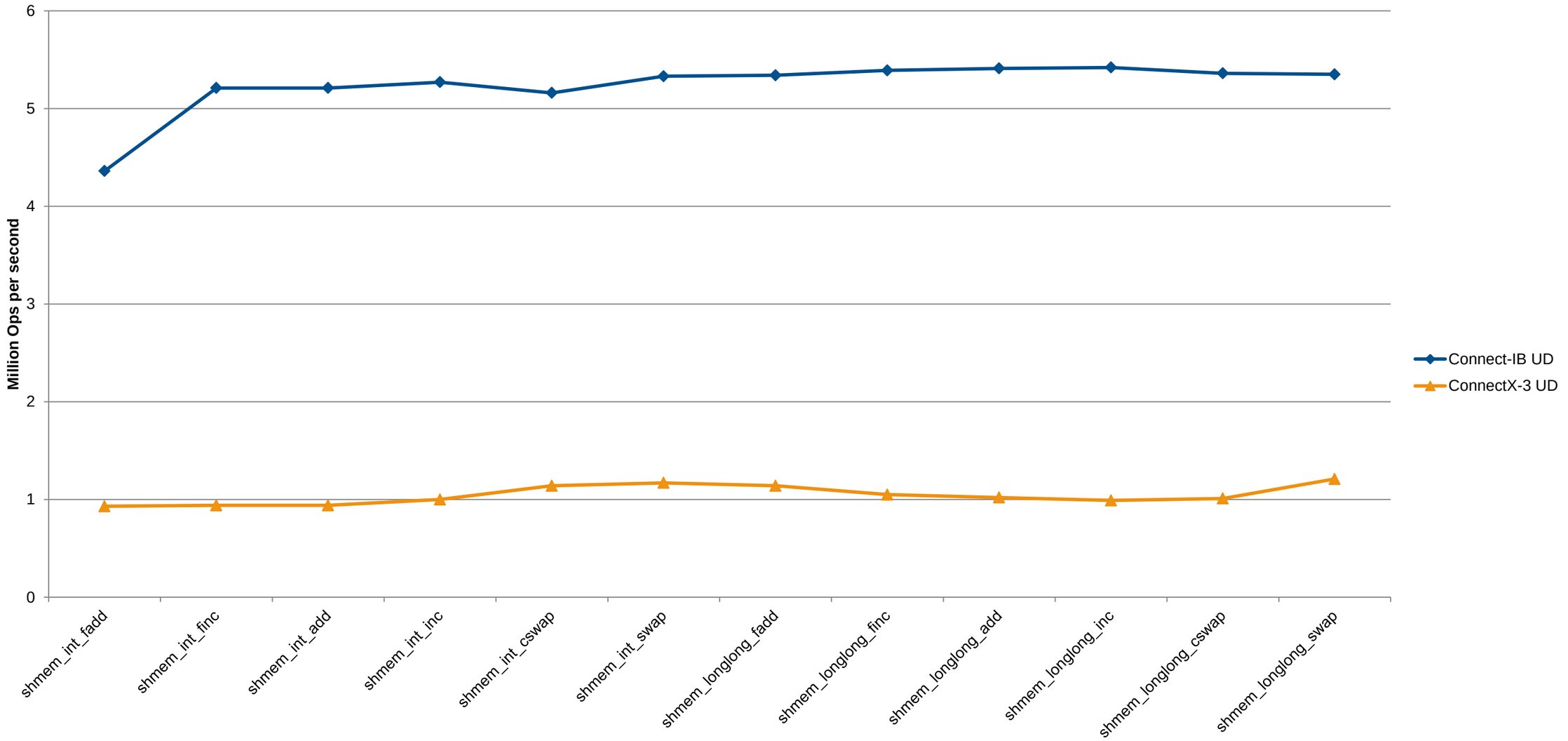
‘SHMEM Barrier - Latency



SHMEM All-to-All Benchmark (Message Size – 4096 Bytes)



SHMEM Atomic Updates



For more information: HPC@mellanox.com

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™