Enabling Exascale Co-Design Architecture

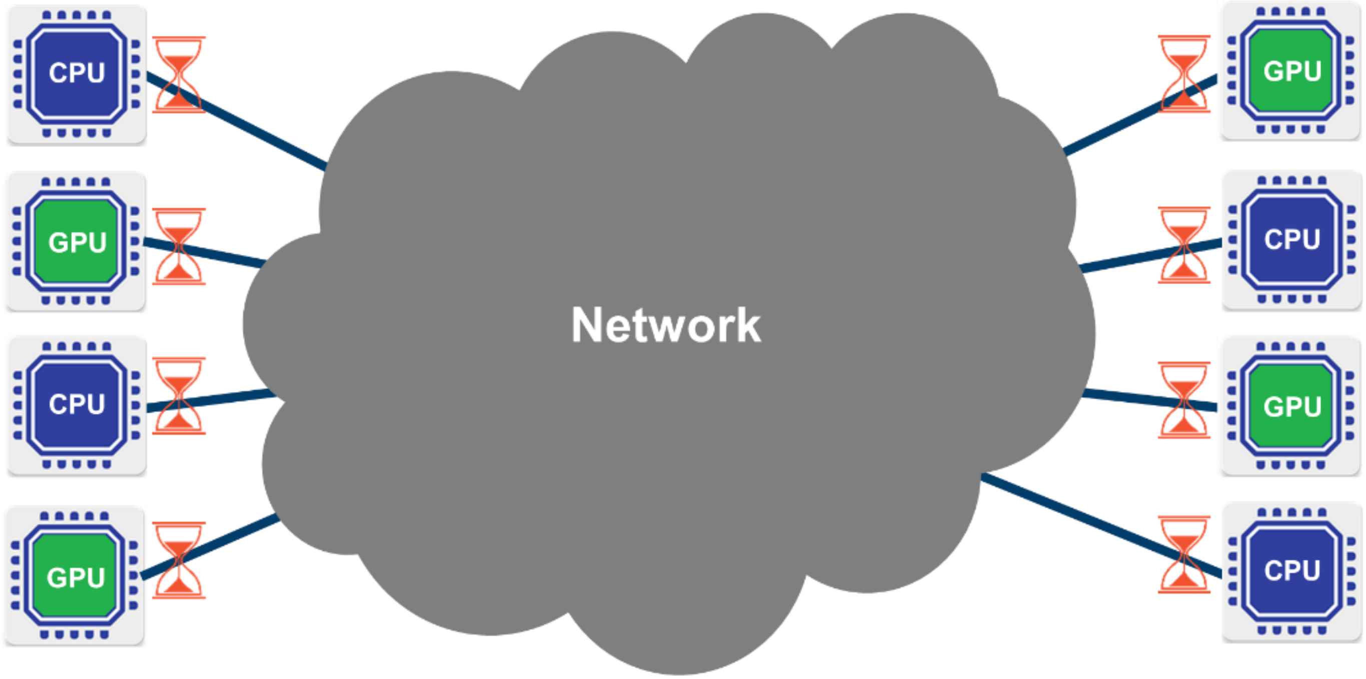
Devendar Bureddy

OSU MUG, AUG 17

Mellanox TECHNOLOGIES
Connect. Accelerate. Outperform.

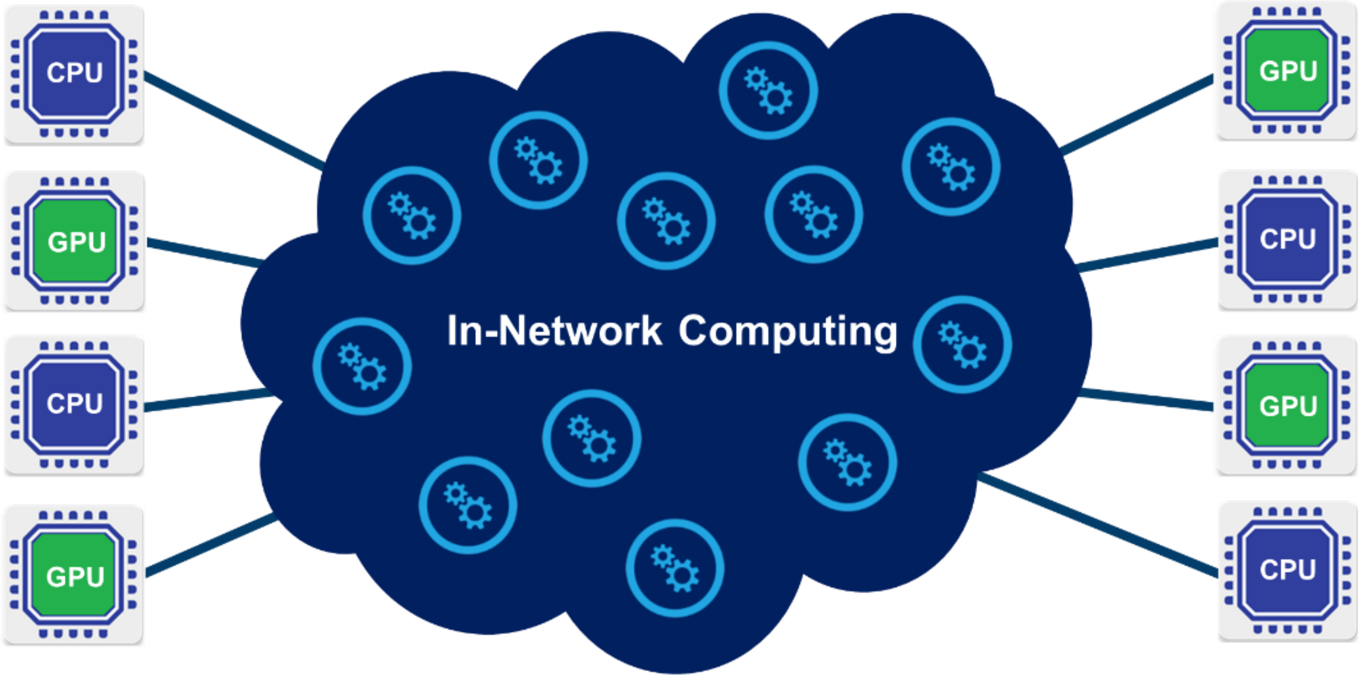
Exponential Data Growth – The Need for Intelligent and Faster Interconnect

CPU-Centric (Onload)



Must Wait for the Data
Creates Performance Bottlenecks

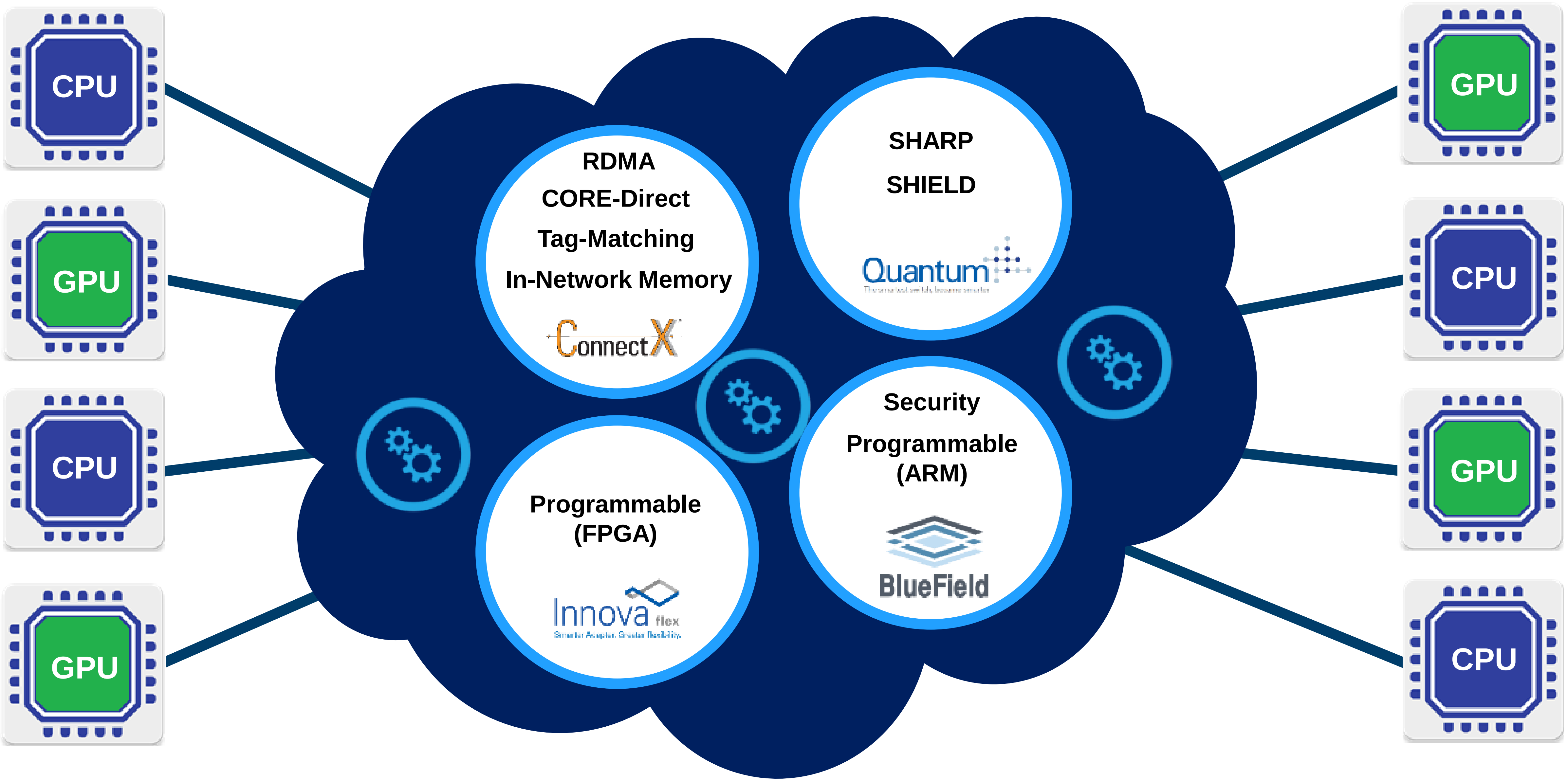
Data-Centric (Offload)



Process Data as it Moves!

Faster Data Speeds and In-Network Computing Enable Higher Performance and Scale

In-Network Computing to Enable Data-Centric Data Center



RDMA

CORE-Direct

Tag-Matching

In-Network Memory

SHARP

SHIELD

Programmable (FPGA)

Security

Programmable (ARM)

In-Network Computing Key for Highest Return on Investment

Highest-Performance 100Gb/s Interconnect Solutions

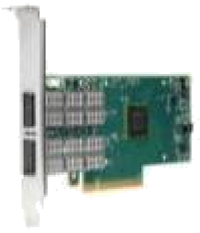
Adapters

ConnectX-5

100Gb/s Adapter, 0.6us latency

175-200 million messages per second

(10 / 25 / 40 / 50 / 56 / 100Gb/s)



Switch

Switch-IB 2™

36 EDR (100Gb/s) Ports, <90ns Latency

Throughput of 7.2Tb/s

7.02 Billion msg/sec (195M msg/sec/port)

Switch

Spectrum™

32 100GbE Ports, 64 25/50GbE Ports

(10 / 25 / 40 / 50 / 100GbE)

Throughput of 3.2Tb/s

Interconnect

LinkX™

Transceivers

Active Optical and Copper Cables

(10 / 25 / 40 / 50 / 56 / 100Gb/s)

VCSELs, Silicon Photonics and Copper

Software

HPC-X™

MPI, SHMEM/PGAS, UPC

For Commercial and Open Source Applications

Leverages Hardware Accelerations

Highest-Performance 200Gb/s Interconnect Solutions

Adapters

ConnectX-6

200Gb/s Adapter, 0.6us latency

200 million messages per second

(10 / 25 / 40 / 50 / 56 / 100 / 200Gb/s)

Switch

Quantum

The smartest switch, became smarter

40 HDR (200Gb/s) InfiniBand Ports

80 HDR100 InfiniBand Ports

Throughput of 16Tb/s, <90ns Latency

Switch

Spectrum-2

32 200GbE, 128 25/50GbE Ports

(10 / 25 / 40 / 50 / 100 / 200 GbE)

Throughput of 6.4Tb/s

Interconnect

LinkX

Transceivers

Active Optical and Copper Cables

(10 / 25 / 40 / 50 / 56 / 100 / 200Gb/s)

VCSELs, Silicon Photonics and Copper

Software

HPC-X

MPI, SHMEM/PGAS, UPC

For Commercial and Open Source Applications

Leverages Hardware Accelerations

ConnectX-6 200G HDR InfiniBand and Ethernet Smart Adapter

100/200Gb/s Throughput

0.6usec End-to-End Latency

175/200M Messages per Second

PCIe Gen3 and Gen4

Integrated PCIe Switch and Multi-Host Technology

Advanced Adaptive Routing

In-Network Computing (Collectives, Tag Matching)

In-Network Memory

Storage (NVMe), Security and Network Offloads

ConnectX®-6

Quantum 200G HDR InfiniBand Smart Switch

40 Ports of 200G HDR InfiniBand
80 Ports of 100G HDR100 InfiniBand
Switch System 800 Ports 200G, 1600 Ports 100G

16Tb/s Switch Capacity
Extremely Low Latency of 90ns
15.6 Billion Messages per Second

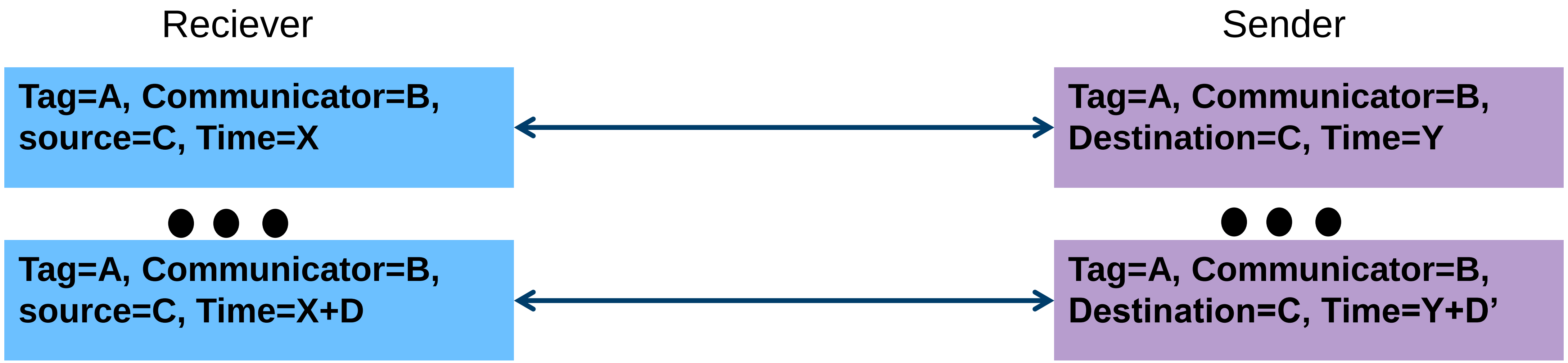
In-Network Computing (SHARPv2 Technology)
Flexible Topologies (Fat-Tree, Torus, Dragonfly, etc.)
Advanced Adaptive Routing

Quantum
The smartest switch, became smarter

MPI Tag-Matching Support

MPI Tag Matching

- Sender: tag, communicator, destination, source (implicit)
- Receiver: tag (may be wild carded), communicator, source (may be wild carded), destination (implicit)
- Matching: sender and receiver envelopes must match
- Matching Ordering:
  - Matching envelopes are required
  - Posted received must be matched in-order against the in-order posted sends



Reciever

Sender

Tag=A, Communicator=B, source=C, Time=X

Tag=A, Communicator=B, Destination=C, Time=Y

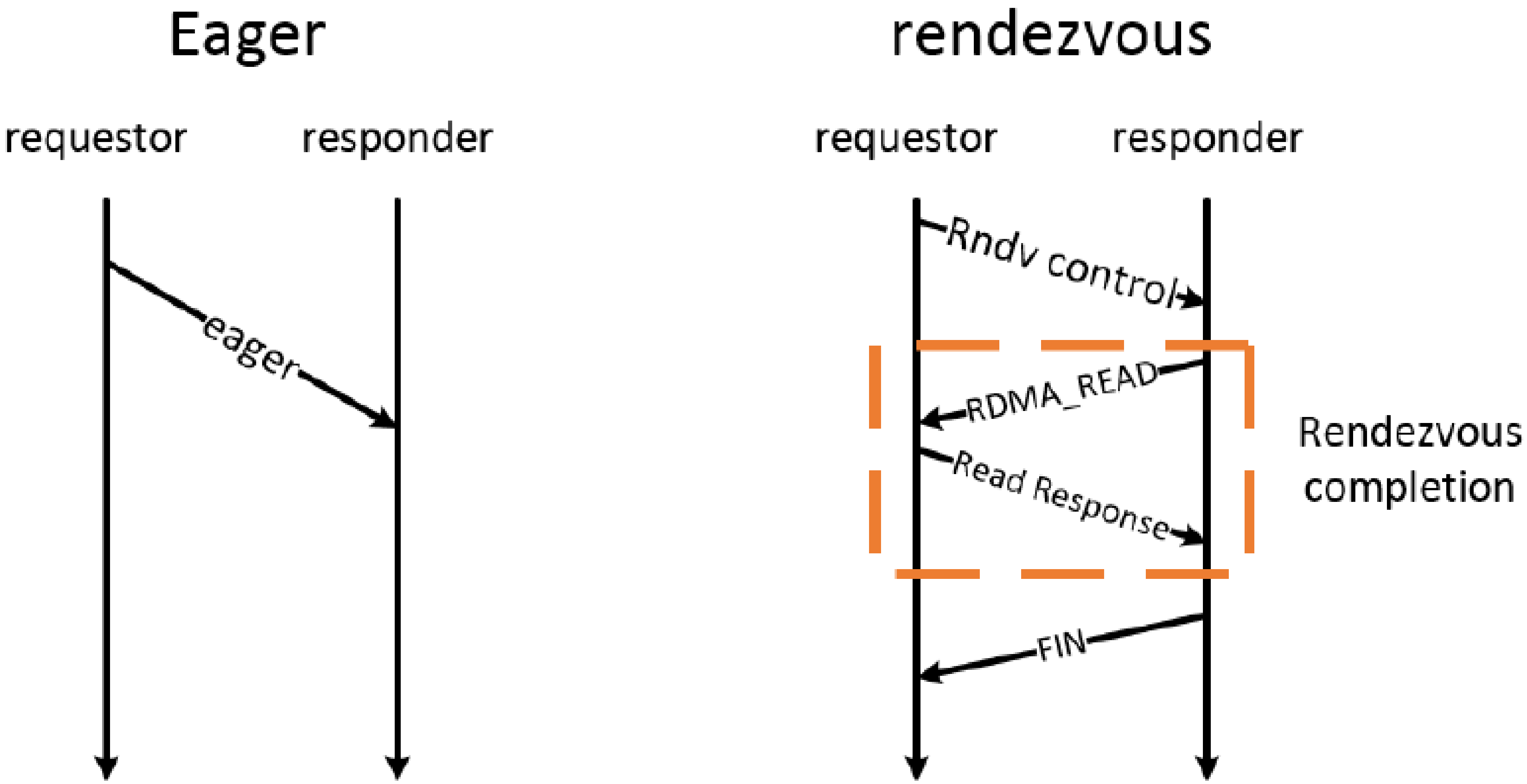
● ● ●

● ● ●

Tag=A, Communicator=B, source=C, Time=X+D

Tag=A, Communicator=B, Destination=C, Time=Y+D’

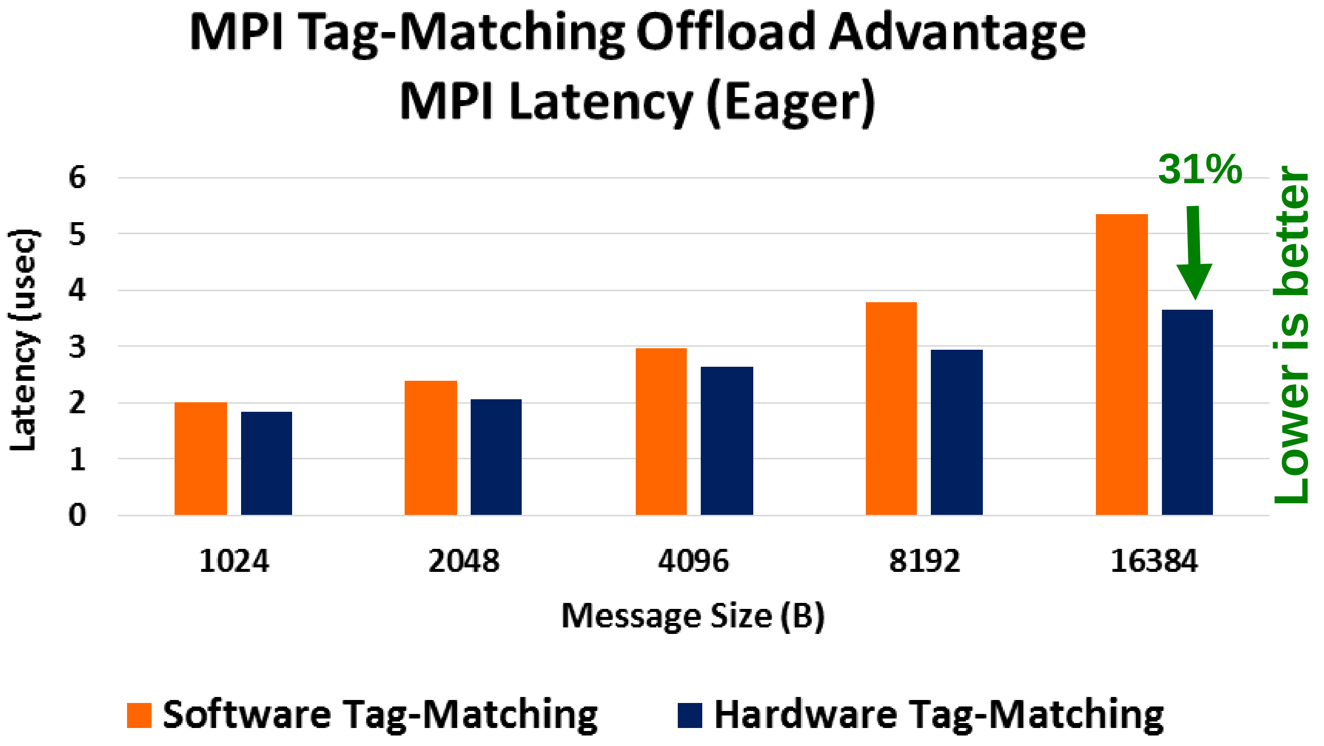
Tag Matching – ConnectX-5 Support

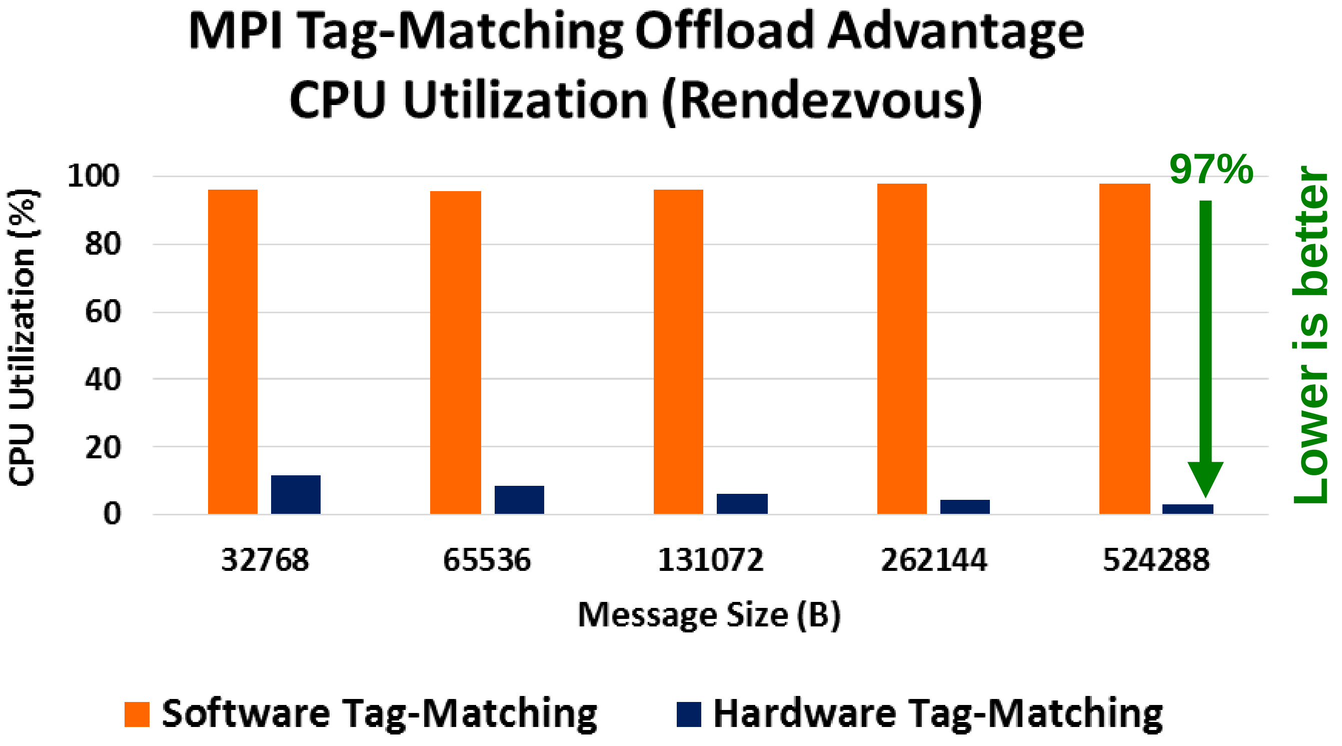


Mellanox's Tag-Matching Support

- Offloaded to the ConnectX-5 HCA
  - Full MPI tag-matching : tag matching as compute is progressing
  - Rendezvous offload : large data delivery as compute is progressing
- Control can be passed between Hardware and Software
- Verbs Tag-Matching support being up-streamed
- UCX support

MPI Tag-Matching Offload Advantages





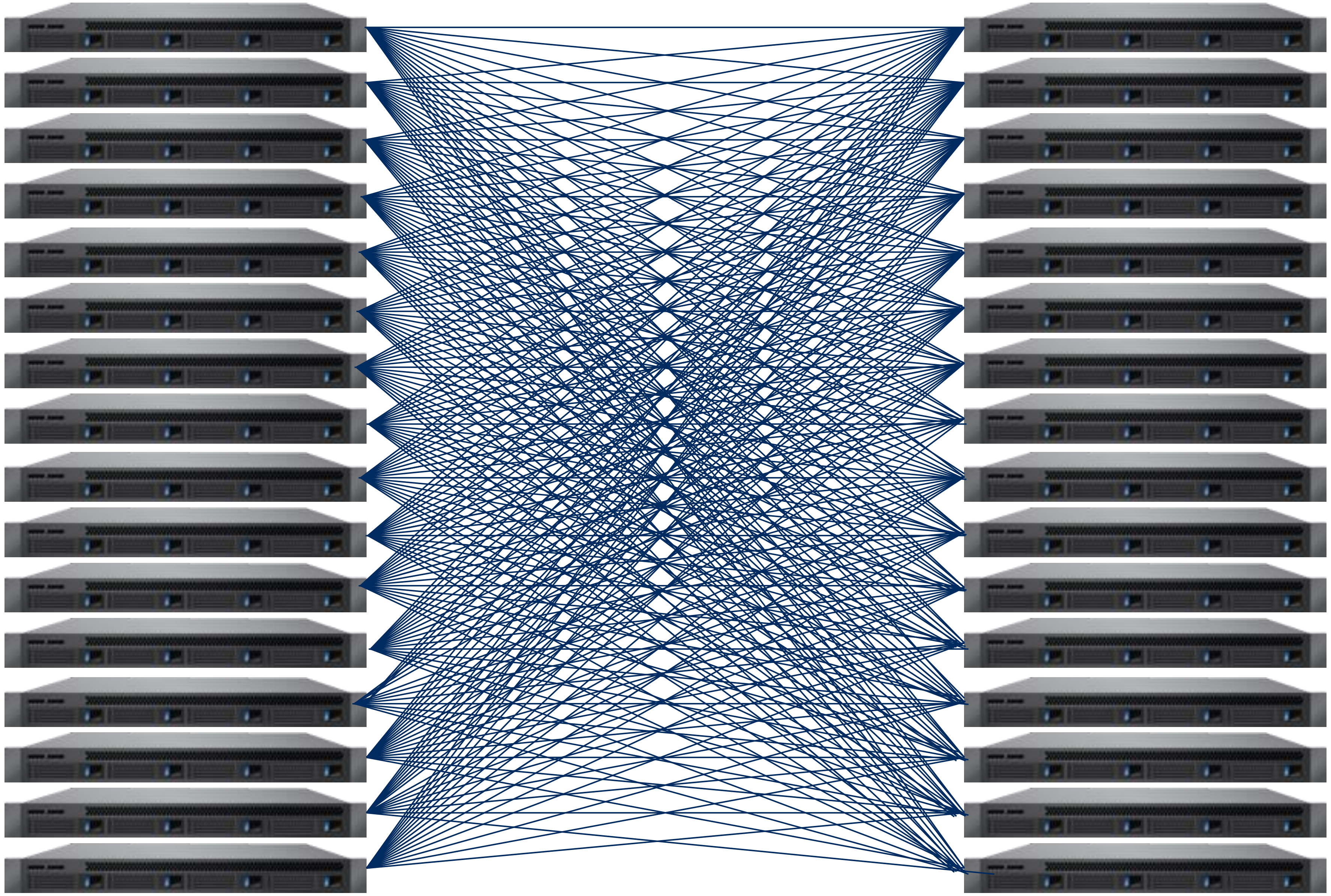
- 31% lower latency and 97% lower CPU utilization for MPI operations
- Performance comparisons based on ConnectX-5

Mellanox In-Network Computing Technology Deliver Highest Performance

Dynamically Connected Transport

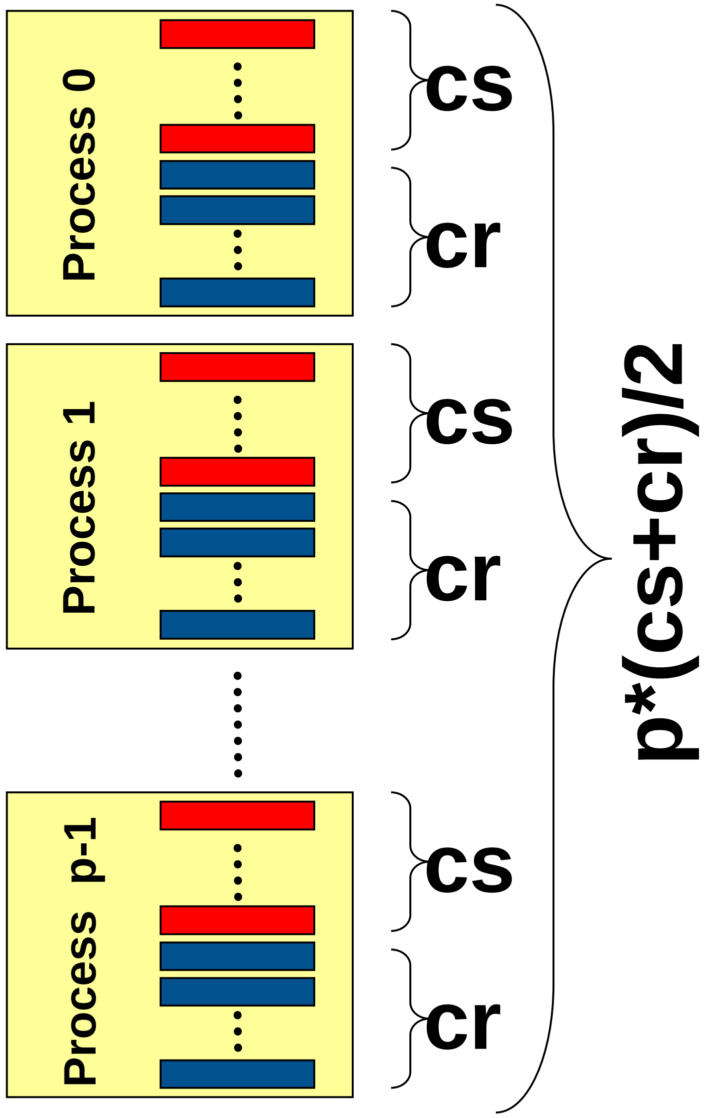
A Scalable Transport

Reliable Connection Transport Mode



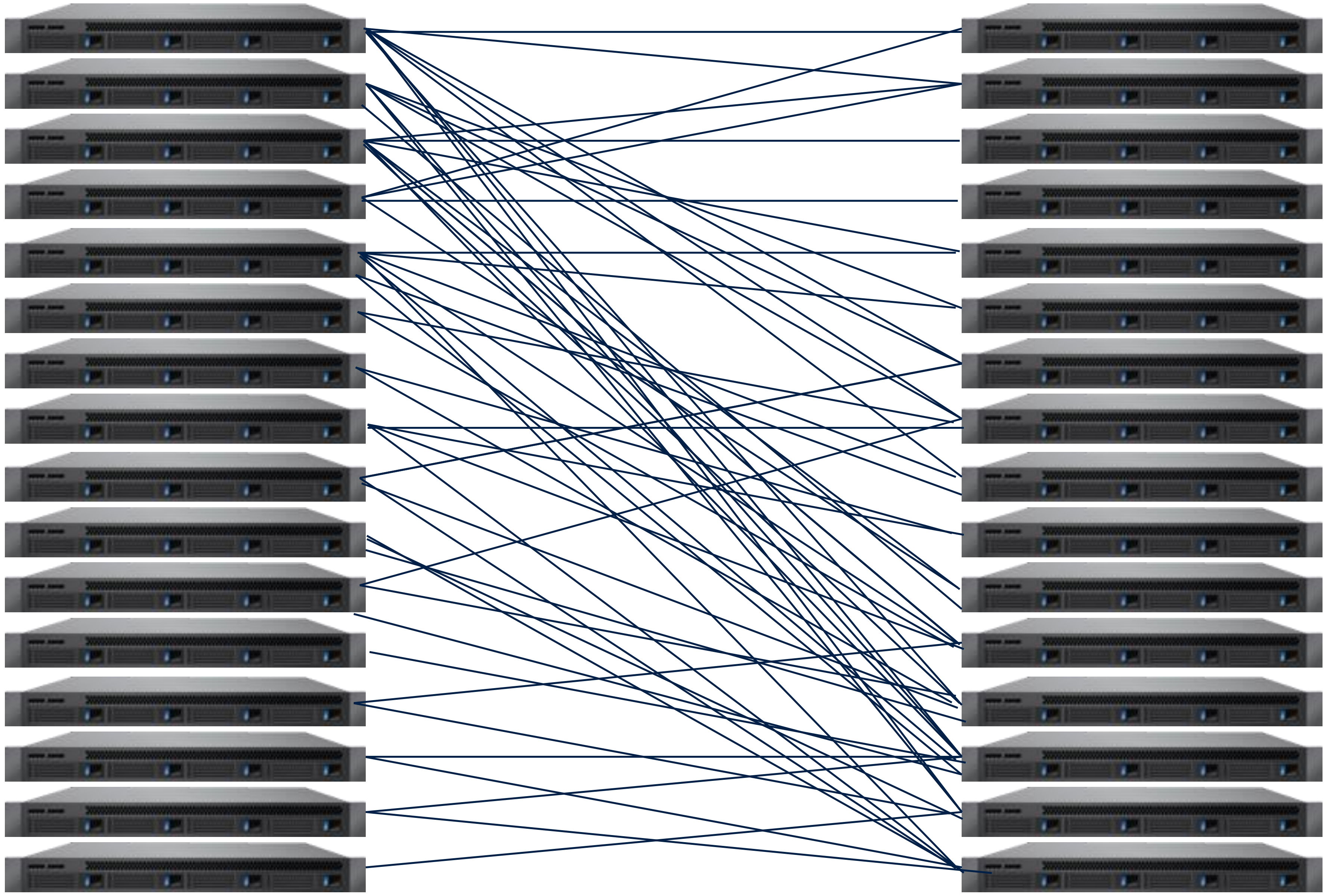
The DC Model

- 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

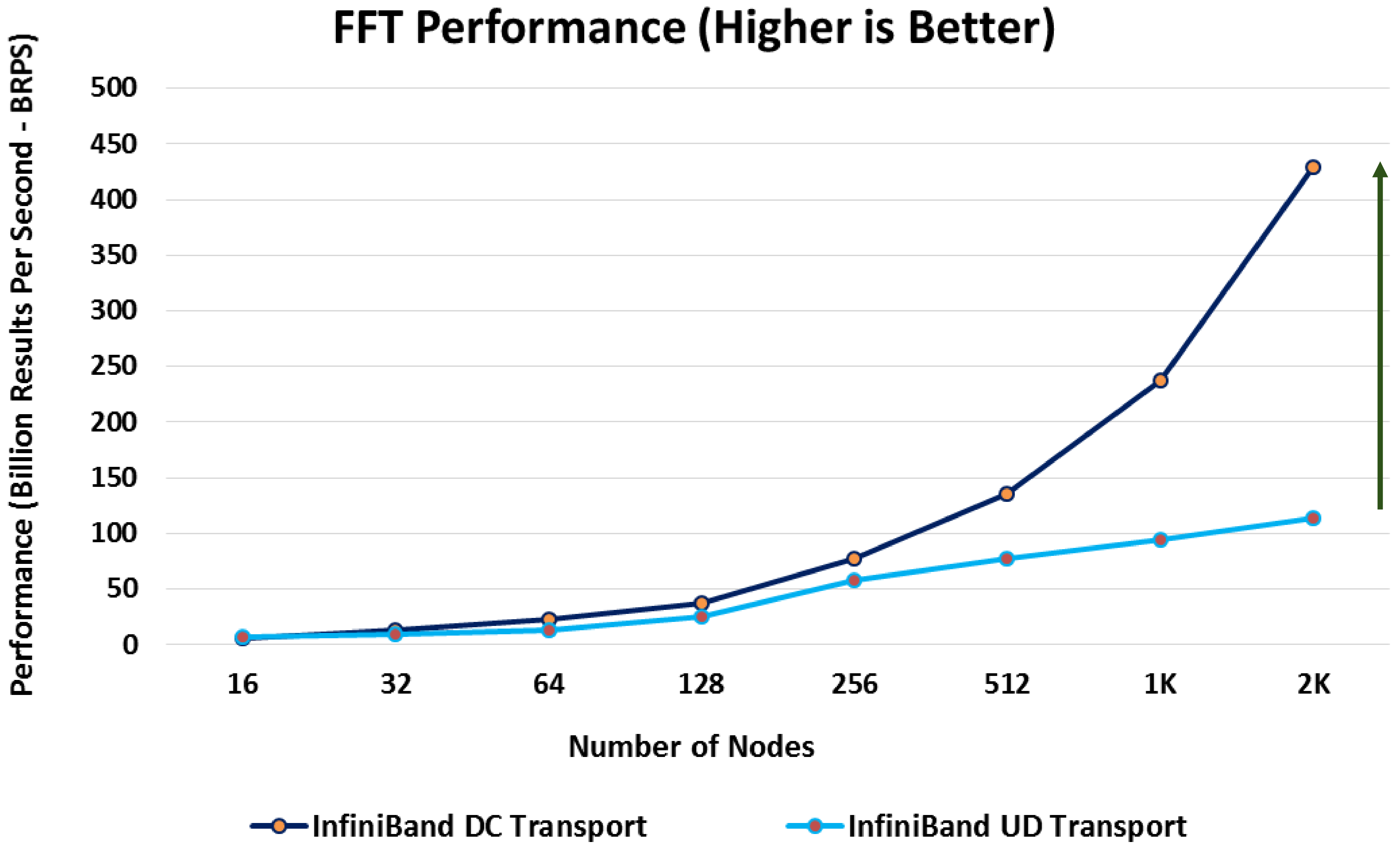


cs – concurrency of the sender
cr=concurrency of the responder

Dynamically Connected Transport Mode



Dynamically Connected Transport Performance Advantage



3.8X
Higher Performance

Non-Contiguous Data

UMR: Optimizing Non Contiguous Memory Transfers

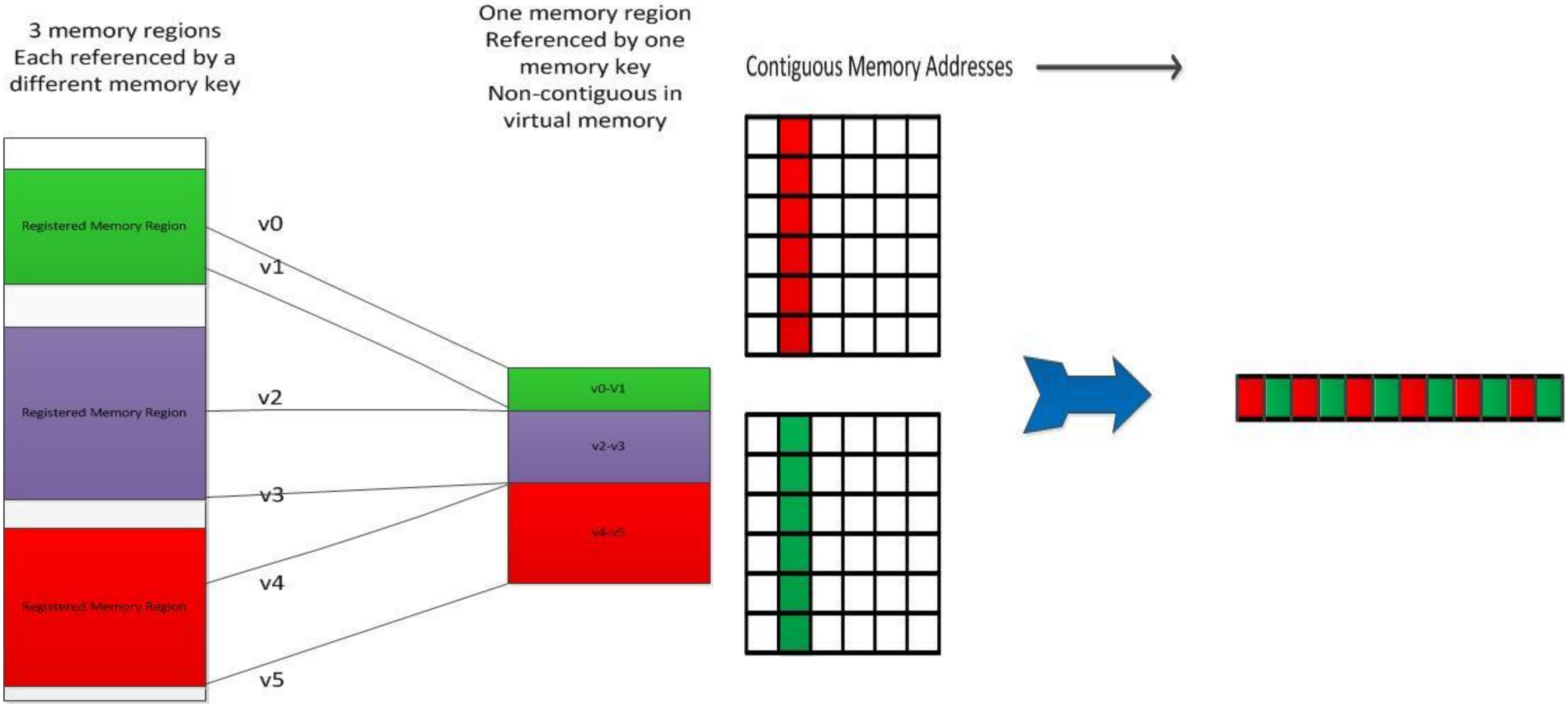
- Support combining contiguous registered memory regions into a single memory region. H/W treats them as a single contiguous region (and handles the non-contiguous regions)

- 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

Optimizing Non Contiguous Memory Transfers

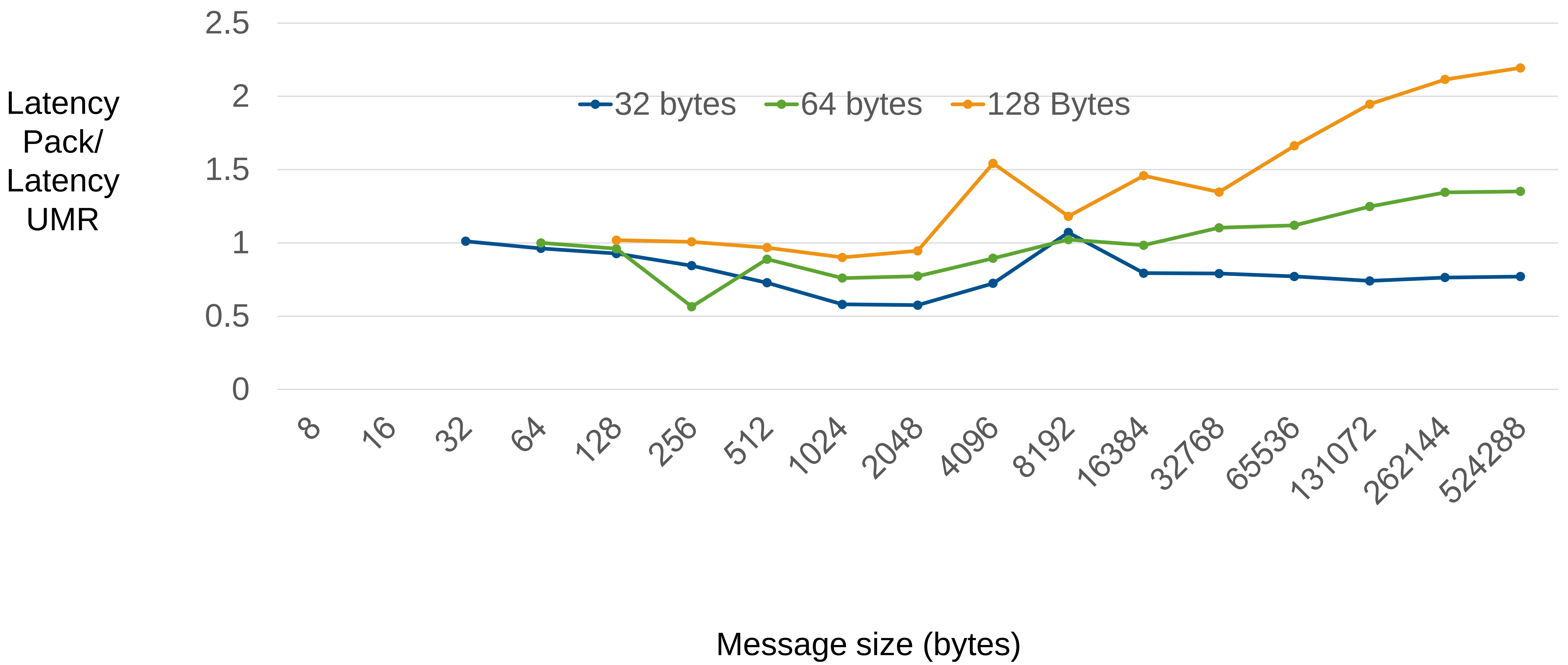


3 memory regions
Each referenced by a
different memory key

One memory region
Referenced by one
memory key
Non-contiguous in
virtual memory

Contiguous Memory Addresses

Hardware Gather/Scatter Capabilities – Regular Structure – Ping-Pong latency



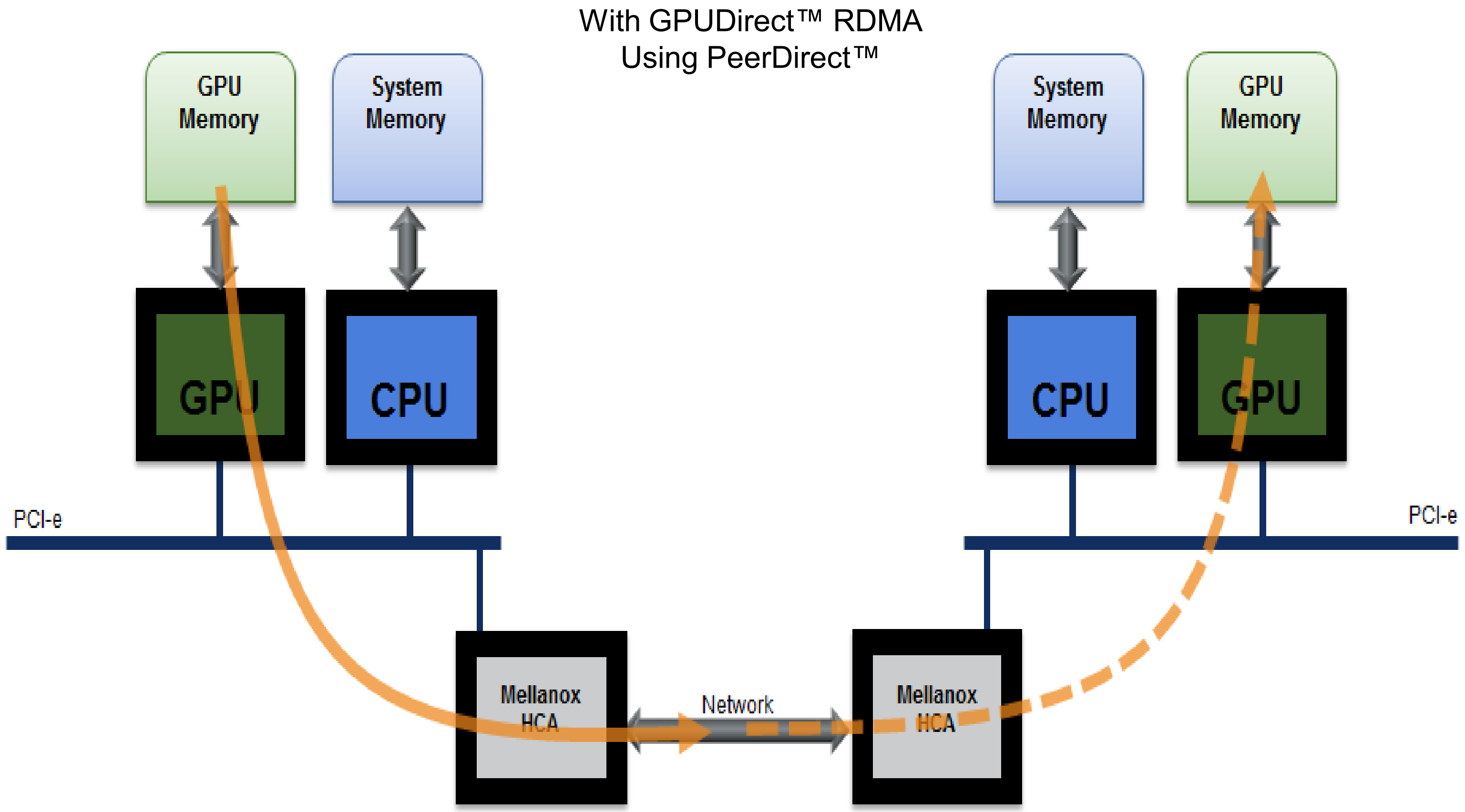
GPUDirect RDMA Technology

Maximize Performance via Accelerator and GPU Offloads

PeerDirect™ Features

- Accelerated Communication With Network And Storage Devices
  - Avoid unnecessary system memory copies and CPU overhead by copying data directly to/from pinned third-party device memory
  - Peer-To-Peer Transfers Between third-party device and Mellanox RDMA devices
  - Use high-speed DMA transfers to copy data between P2P devices
  - Eliminate CPU bandwidth and latency bottlenecks using direct memory access (DMA)
- RDMA support
  - With PeerDirect™ , memory of a third-party device can be used for Remote Direct Memory Access (RDMA) of the **data buffers** resulting in letting application run more efficiently.
  - Allow RDMA-based application to use a third-party device, such as a GPU for computing power, and RDMA interconnect at the same time w/o copying the data between the P2P devices.
  - Boost Message Passing Interface (MPI) Applications with zero-copy support
  - Support for RDMA transport over **InfiniBand and RoCE**

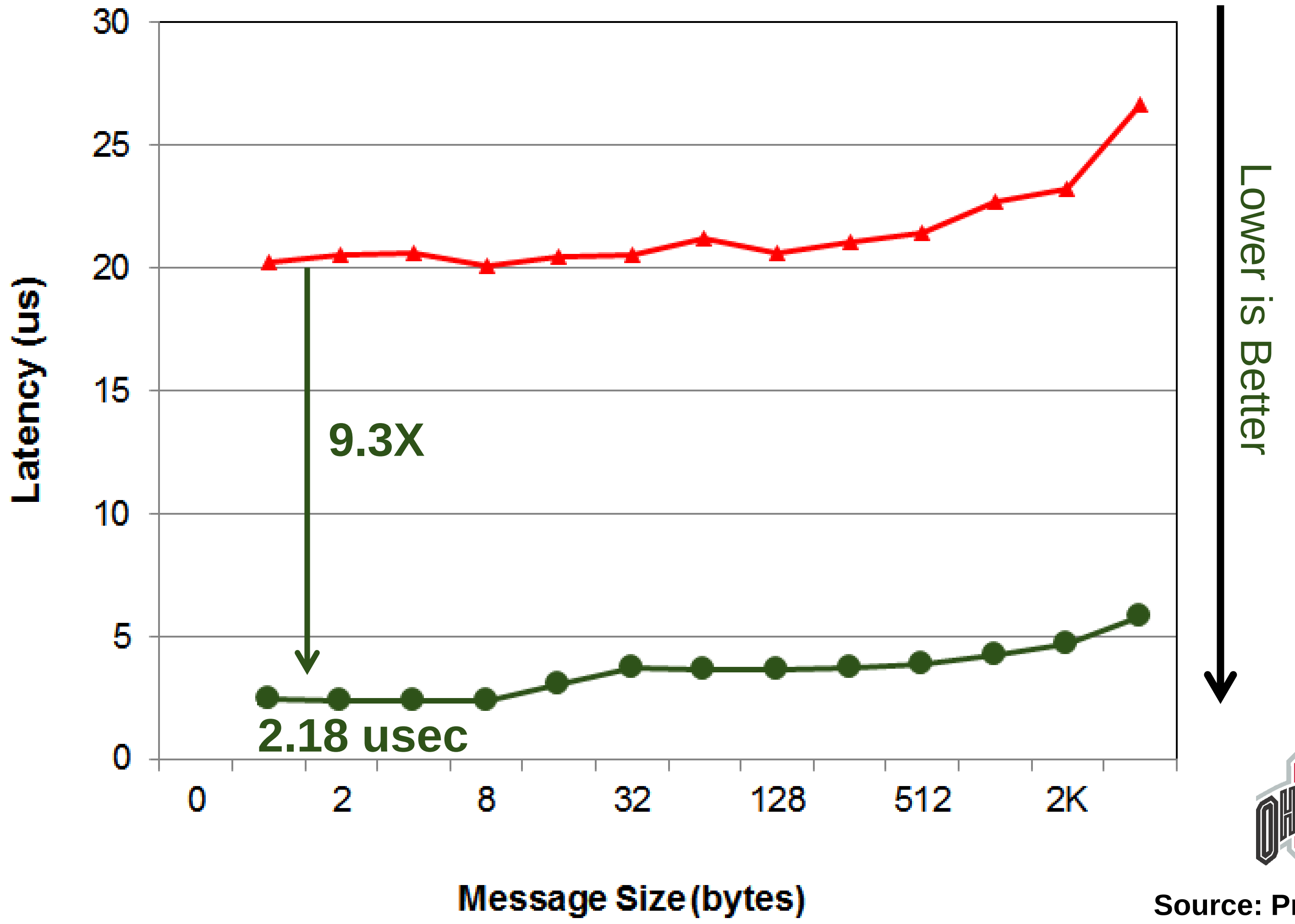
GPUDirect™ RDMA (GPUDirect 3.0)



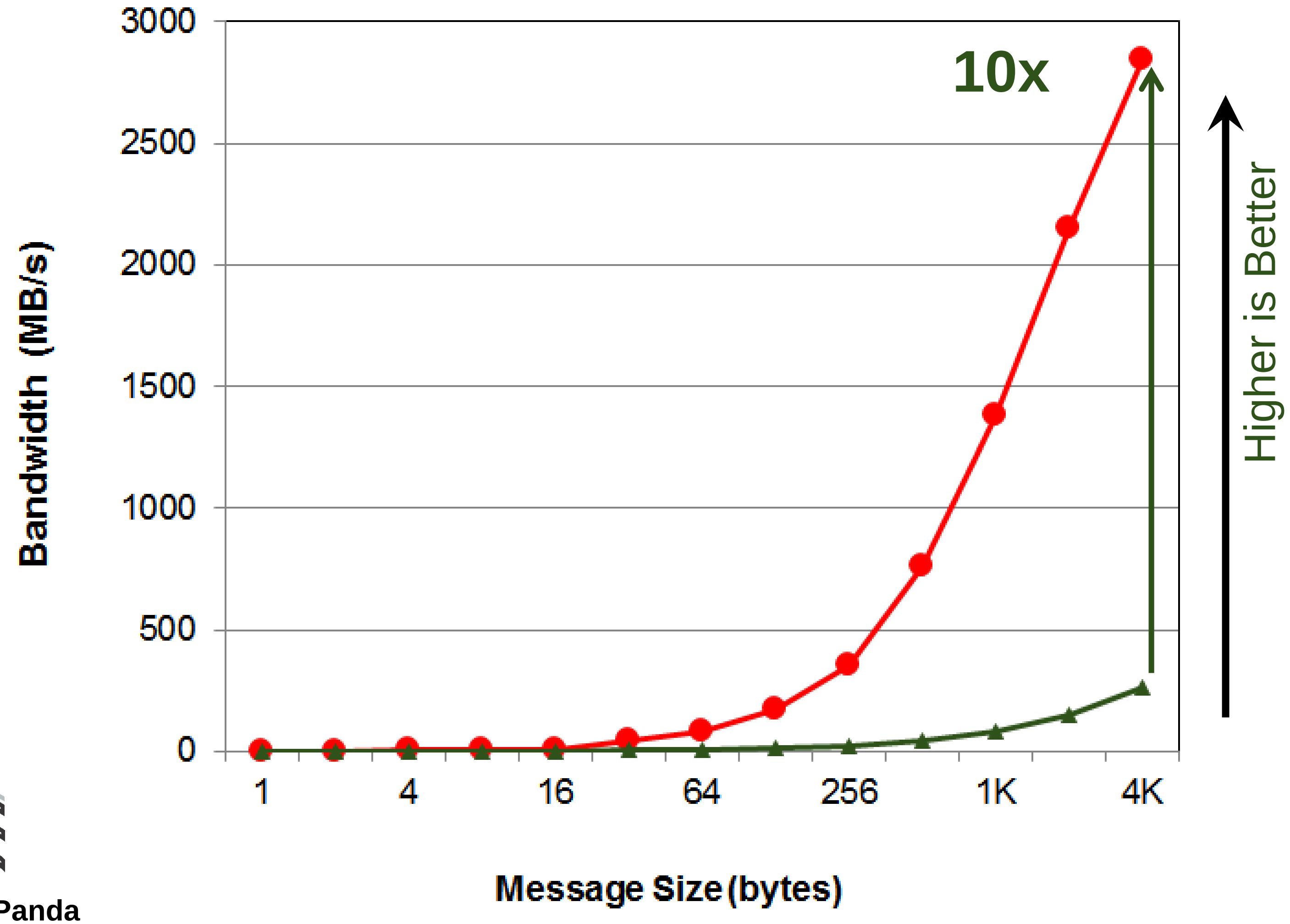
- Eliminates CPU bandwidth and latency bottlenecks
- Uses remote direct memory access (RDMA) transfers between GPUs
- Resulting in significantly improved MPI efficiency between GPUs in remote nodes
- Based on PCIe PeerDirect technology

Performance of MVAPICH2 with GPUDirect RDMA

GPU-GPU Internode MPI Latency



GPU-GPU Internode MPI Bandwidth



Source: Prof. DK Panda

88% Lower Latency

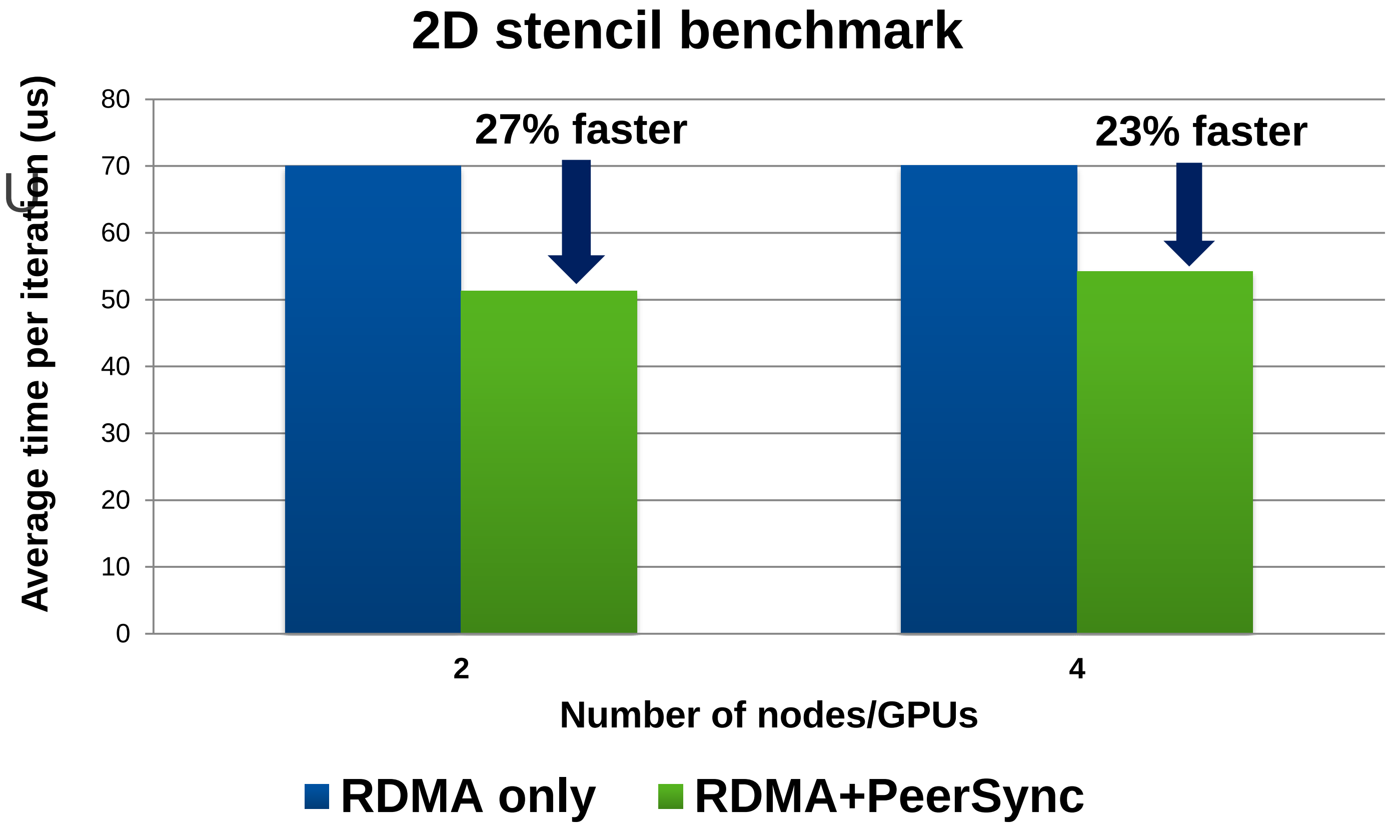
10X Increase in Throughput

GPUDirect Async

- GPUDirect RDMA (3.0)
  - direct data path between the GPU and Mellanox interconnect
  - Control path still uses the CPU

- GPUDirect ASync (GPUDirect 4.0)
  - Both data path and control path go directly between the GPU and the Mellanox interconnect
  - CPU prepares and queues communication tasks on GPU
  - GPU triggers communication on HCA
  - Mellanox HCA directly accesses GPU memory

Maximum Performance For GPU Clusters



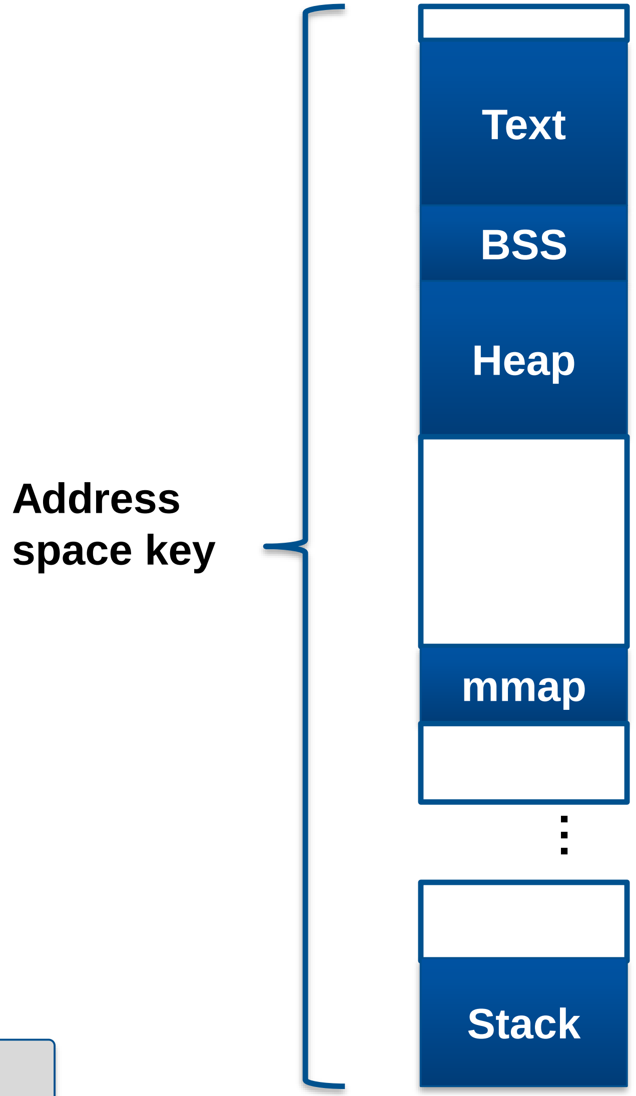
On Demand Paging

On-Demand Paging (ODP)

- 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
- MR pages are ***never*** pinned by the OS
  - Paged in when HCA needs them
  - Paged out when reclaimed by the OS
- Eliminates the price of pinning
  - Unlimited MR sizes
    - No need for special privileges
  - Physical memory optimized to hold current working set
    - For both CPU and IO access
  - Application pages may be migrated at will

ODP: The Global Address Space Key

- Register the whole process address space with a ***single*** key
  - MR covers existing and future memory mappings
- MR covers unmapped address ranges
  - Permissions checked at access (page fault) time
    - VMA permissions
    - MR access rights
  - RDMA access rights revoked upon invalidation or permission changes
- Granular remote permissions via Memory Windows
  - User-space equivalent for Fast Registration Work Requests…
- Eliminates the price of memory management
  - All data transfer done based on the address space key
  - No need to register and track any other MRs (!)



```
mr = ibv_reg_mr(pd, NULL, -1, IBV_ACCESS_LOCAL_WRITE | IBV_ACCESS_ON_DEMAND);
```

ODP: Memory Prefetching

- Pre-faults (populates) ODP MRs

- Best effort hint
  - Not necessarily all pages are pre-fetched
  - No guarantees that pages remain resident
  - Asynchronous
    - Can be invoked opportunistically in parallel to IO

- Use cases
  - Avoid multiple page faults by small transactions
  - Pre-fault a large region about to be accessed by IO

- EFAULT returned when
  - Range exceeds the MR
  - Requested range not mapped to address space

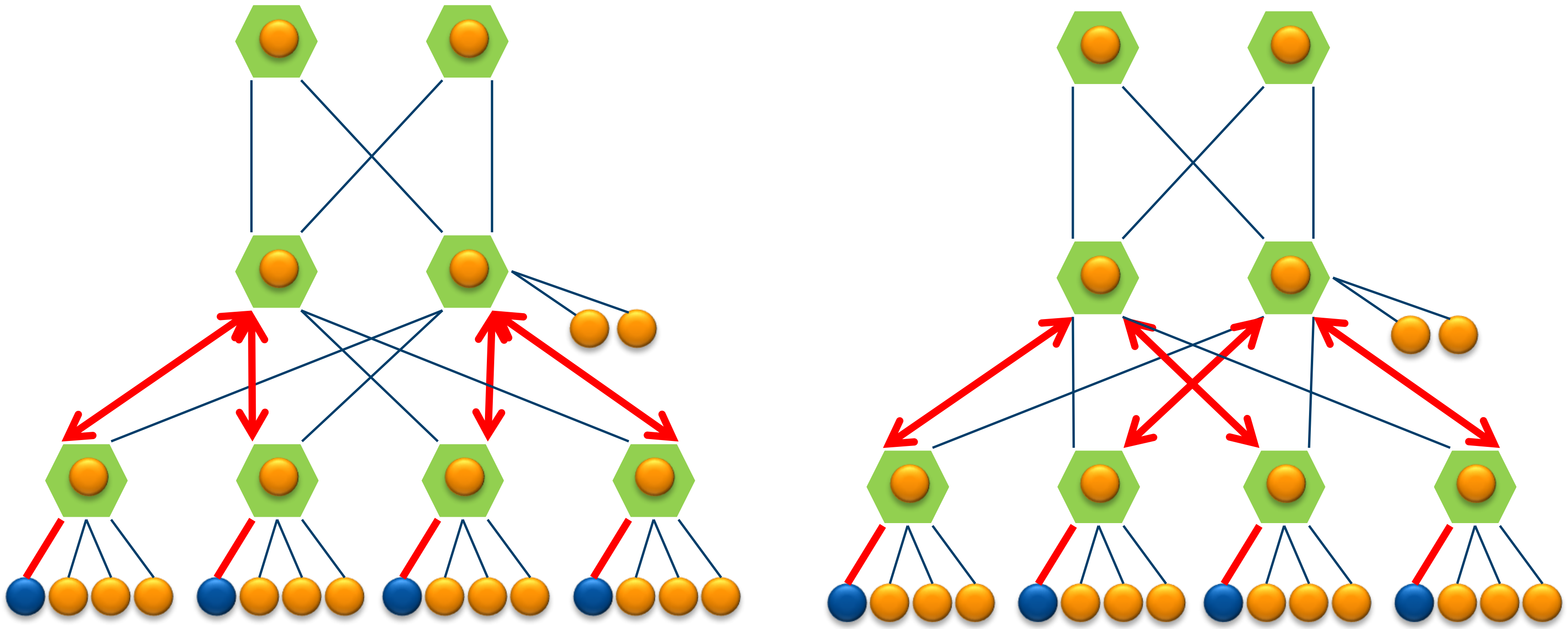
```
struct ibv_prefetch_attr {
        uint32_t comp_mask;
        int flags; /* IBV_ACCESS_LOCAL_WRITE */
        void *addr;
        size_t length;
};

int ibv_prefetch_mr(struct ibv_mr *mr,
                    struct ibv_prefetch_attr *attr,
                    size_t attr_size);
```

Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)

Compute in the Interior of the network

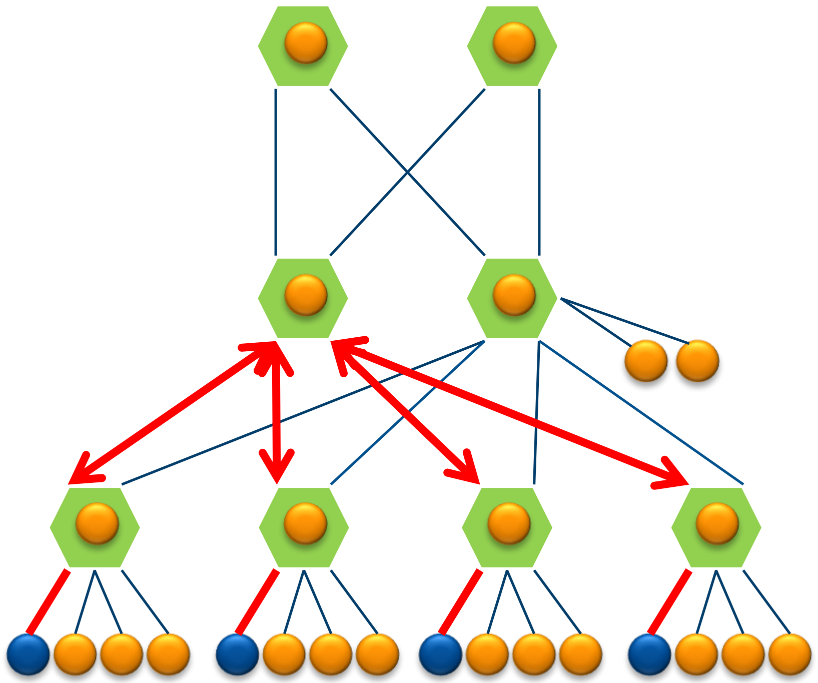
Recursive Doubling vs. SHARP Communication Patterns



Step 1 `

Step 2 `

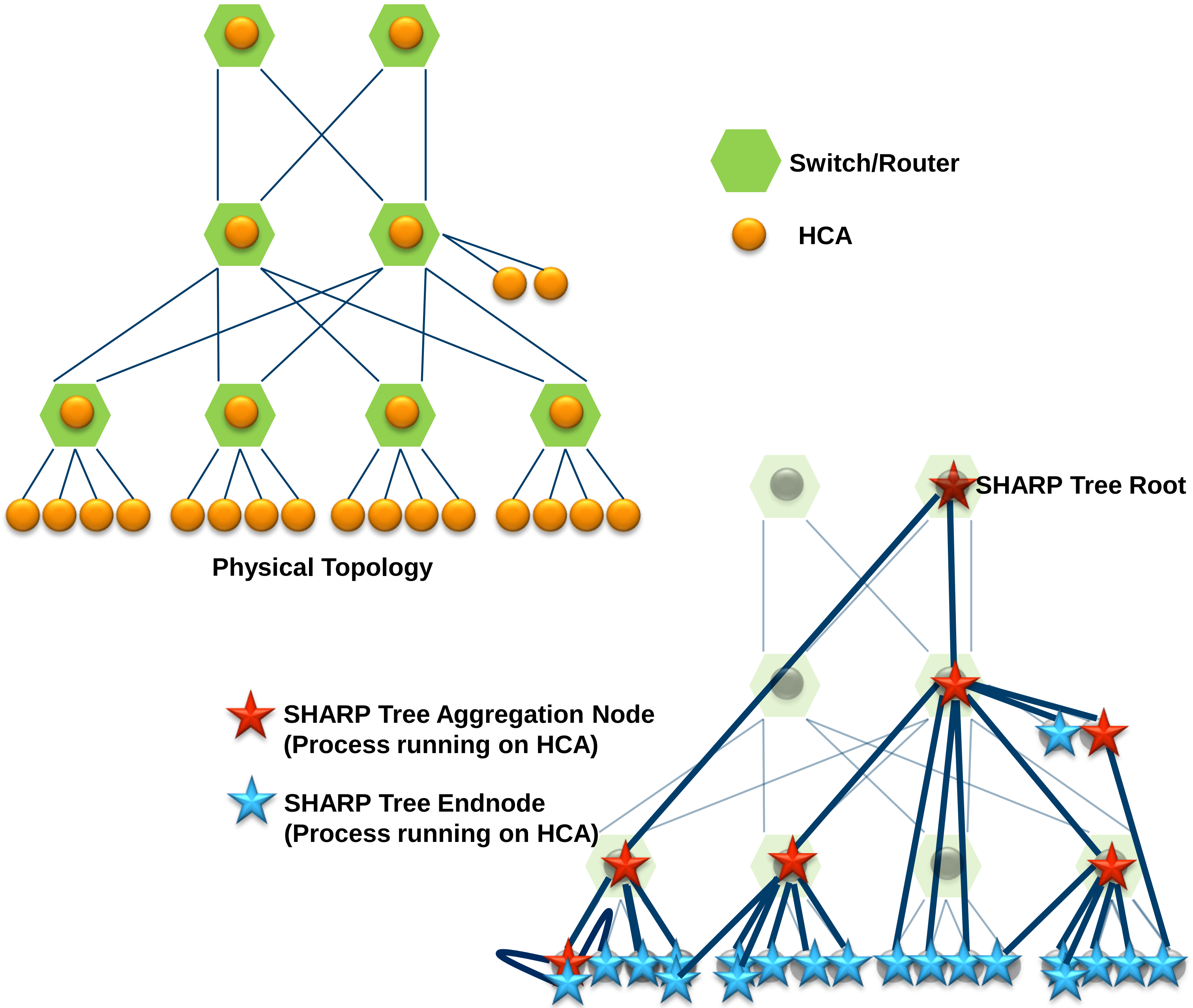
Recursive Doubling `



SHARP`

SHARP Trees

- SHAPR Operations are Executed by a SHARP Tree
  - Multiple SHARP Trees are Supported
  - Each SHARP Tree can handle Multiple Outstanding SHARP Operations
- SHARP Tree is a Logical Construct
  - Nodes in the SHARP Tree are IB end nodes
  - Logical tree defined on top of the physical underlying fabric
  - SHARP Tree Links are implemented on top of the IB transport (Reliable Connection)
  - Expected to follow the physical topology for performance but not required



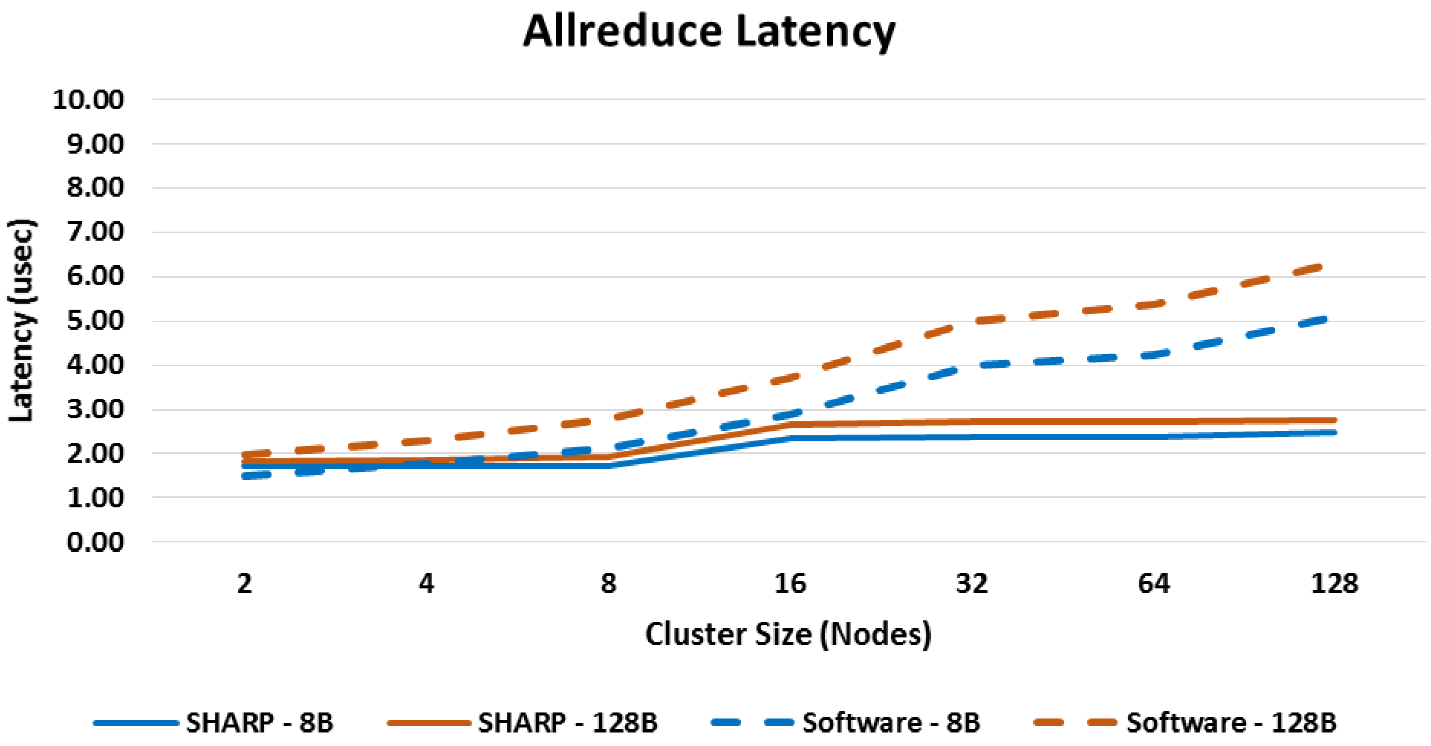
Feature Description

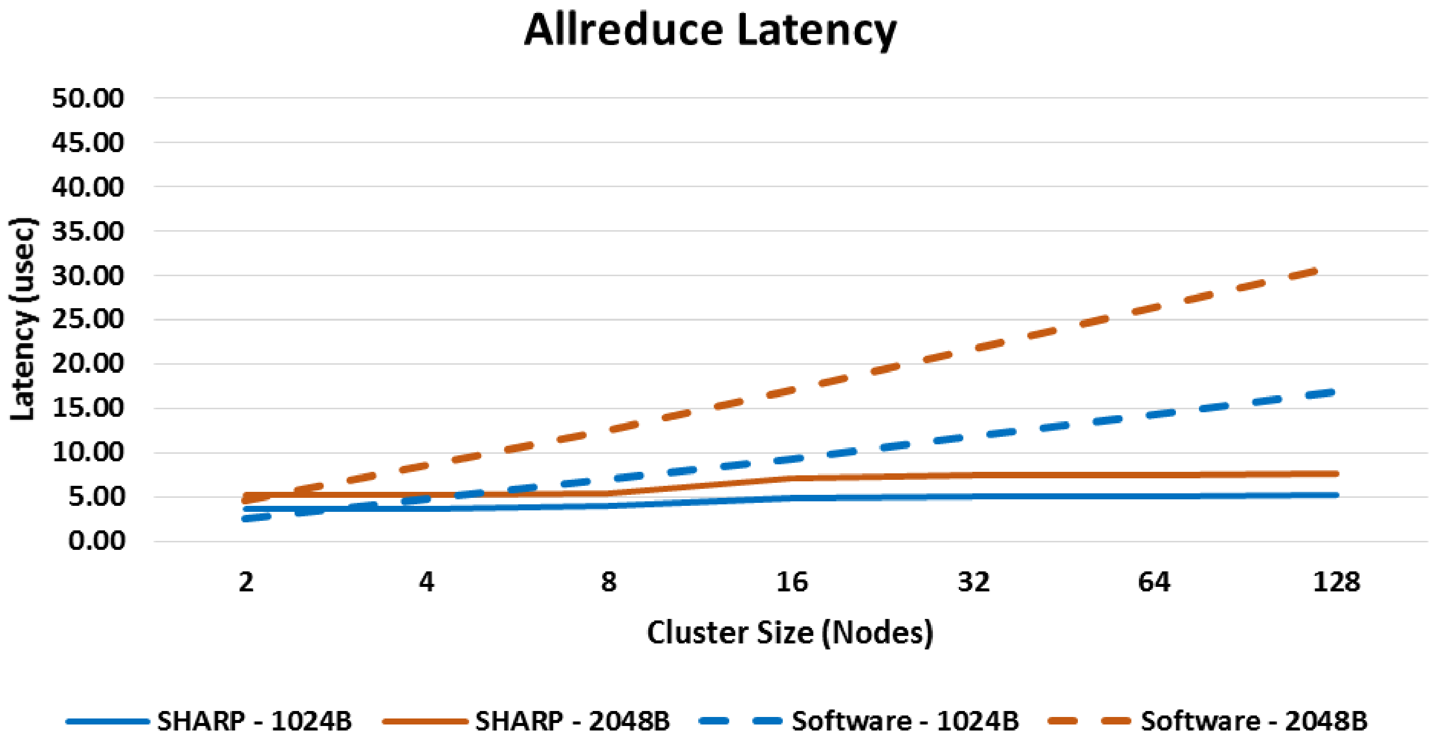
- Reliable Scalable General Purpose Primitive, Applicable to Multiple Use-cases
  - In-network Tree based aggregation mechanism
  - Large number of groups
  - Many simultaneous outstanding operations in flight

Accelerating HPC applications

- Scalable High Performance Collective Offload
  - Barrier, Reduce, All-Reduce
  - Sum, Min, Max, Min-loc, max-loc, OR, XOR, AND
  - Integer and Floating-Point
  - Repeatable results
- Significantly reduce MPI collective runtime
- Increase CPU availability and efficiency
- Enable communication and computation overlap

SHARP Allreduce Performance Advantages





**SHARP enables 75% Reduction in Latency
Providing Scalable Flat Latency**

HPC-X Software Toolkit

Mellanox HPC-X™ Scalable HPC Software Toolkit

HPC-X™

- Complete MPI, PGAS OpenSHMEM and UPC package
- Maximize application performance
- For commercial and open source applications
- Best out of the box experience

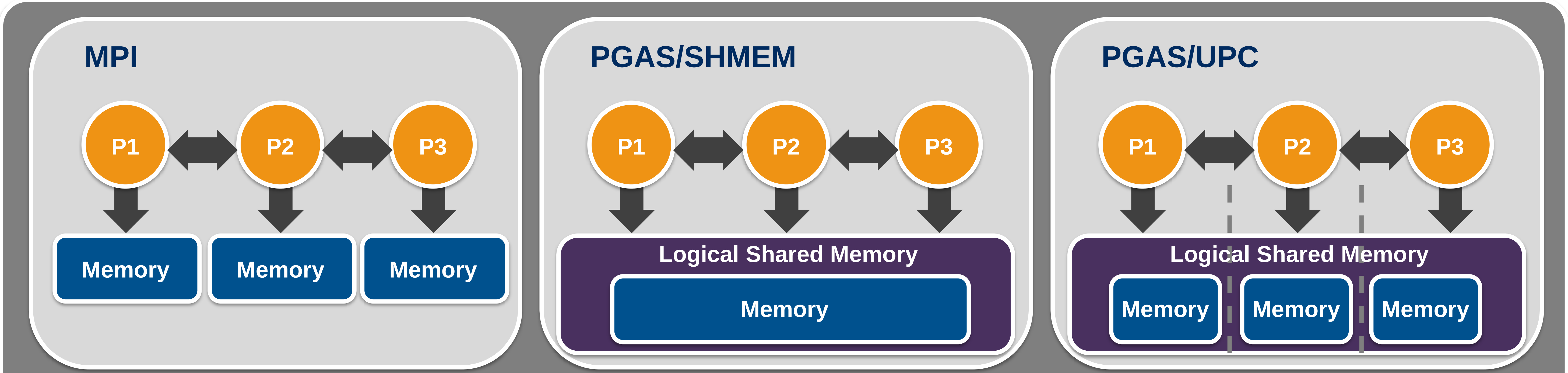
Mellanox HPC-X - Package Contents

❑ HPC-X – Mellanox Scalable HPC Toolkit

- Allow fast and simple deployment of HPC libraries
  - Both Stable & Latest Beta are bundled
  - All libraries are pre-compiled
  - Includes scripts/modulefiles to ease deployment

- Package Includes
  - OpenMPI/OpenSHMEM
  - BUPC (Berkeley UPC)
  - UCX
  - MXM
  - FCA-2.5
  - FCA-3.x (HCOLL)
  - KNEM
    - Allows fast intra-node MPI communication for large messages
  - Profiling Tools
    - Libibprof
    - IPM
  - Standard Benchmarks
    - OSU
    - IMB

Mellanox HPC-X Software Ecosystem



Point-to-Point: MXM -> UCX

- Reliable Messaging Optimized for Mellanox HCA
- Hybrid Transport Mechanism
- Efficient Memory Registration
- Receive Side Tag Matching

Collective: FCA

- Hardware Acceleration: SHArP, Multicast, CORE-Direct
- Topology Aware
- Separate Virtual Fabric for Collectives

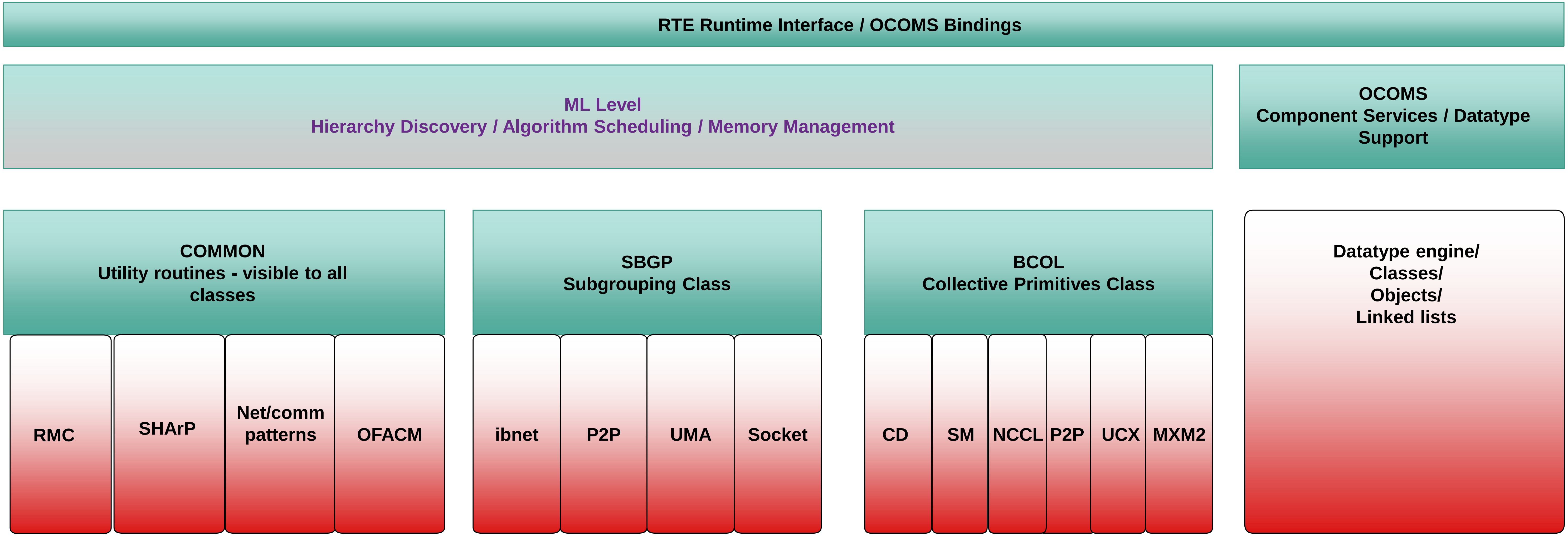
InfiniBand Verbs API

HPC-X: Collective Communication Library - HCOLL

HCOLL

- Scalable infrastructure: Designed and implemented with current and emerging “*extreme-scale”* systems in mind
  - Scalable communicator creation
  - Scalable memory consumption
  - Scalable runtime interface
  - Asynchronous execution
- Flexible and Extensible: Plug-and-play component architecture
  - Leverage object-oriented design patterns
- Adaptive: Designed specifically for current and emerging heterogeneous memory subsystems
  - Can evolve gracefully in lockstep with new architectures
- Optimized collectives: Collective primitives are tuned to a particular communication substrate
- Expose CORE-Direct capabilities
  - Fully asynchronous, non-blocking collectives: Maximize the opportunity for the application developer to overlap computation with communication
  - Increase resilience to the effects of system noise on collective operations at extreme-scale
- Rapidly expose emerging Mellanox hardware features to ULPs with minimal effort e.g. multicast, DC, SHArP, UMR
- Easily integrated into other packages

HCOLL Software Architecture



RTE Runtime Interface / OCOMS Bindings

ML Level
Hierarchy Discovery / Algorithm Scheduling / Memory Management

OCOMS
Component Services / Datatype Support

COMMON
Utility routines - visible to all classes

RMC

SHArP

Net/comm patterns

OFACM

SBGP
Subgrouping Class

ibnet

P2P

UMA

Socket

BCOL
Collective Primitives Class

CD

SM

NCCL

P2P

UCX

MXM2

Datatype engine/
Classes/
Objects/
Linked lists

HCOLL Features

- Blocking and non-blocking collective routines
- Modular component architecture
- Scalable runtime interface (RTE)
  - Successfully integrated into OMPI – “hcoll” component in “coll” framework
  - Successfully integrated in Mellanox OSHMEM
  - Experimental integration in MPICH
  - Working prototype SLURM/PMI2 plug-in
- Host level hierarchy awareness
  - Core groups*
  - Socket groups
  - UMA groups
- Support for network level topology awareness
- Exposes Mellanox and InfiniBand specific capabilities
  - CORE-Direct
  - MXM 2(3).x
    - UD, RC, DC
  - UCX
  - Hardware multicast
  - SHARP
- GPUs support with NCCL

Runtime parameters for FCA-3.x in HPCx

- FCA 2.5
  - To enable FCA support
    - -mca coll_fca_enable 1
    - -mca coll_fca_np 0
    - -x fca_ib_dev_name=mlx5_0 (if multiple interfaces existed)
- FCA 3.1+ (HCOLL)
  - To enable HCOLL support explicitly:
    - -mca coll_hcoll_enable 1
    - -mca coll_hcoll_np 0
    - -x HCOLL_MAIN_IB=mlx5_0:1 (if multiple interfaces existed)

  - To enable HCOLL multicast support explicitly:
    - -x HCOLL_ENABLE_MCAST_ALL=1
    - -x HCOLL_MCAST_NP=0
  - Comm context cache
    - -x HCOLL_CONTEXT_CACHE_ENABLE=1
  - SHArP
    - -x HCOLL_ENABLE_SHARP=1

Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)

Compute in the Interior of the network

SHARP SW Overview

- Release

SHARP version	MOFED version	SwitchIB-2 FW	HPCX version	UFM version
v1.0	MLNX OFED 3.3-x.x.x	15.1100.0072	1.6.392	-
v1.1	MLNX OFED 3.4-0.1.2	15.1200.0102	1.7.405	-
v1.2	MLNX OFED 4.0-x.x.x	15.1200.0102	1.8.xxx	5.8-5.9
v1.3	MLNX OFED 4.1-x.x.x	15.1300.0126	1.9.5	

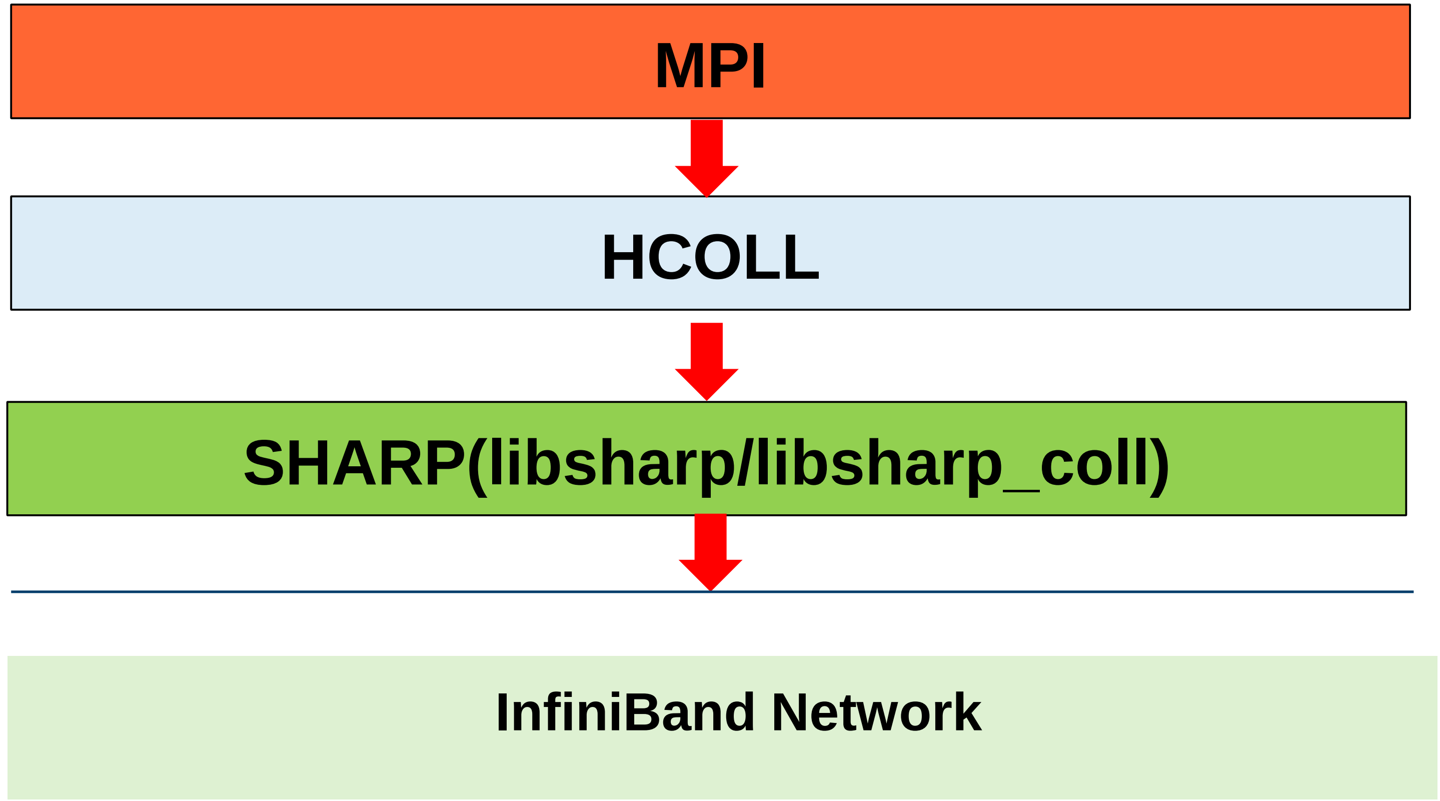
- Packages
  - MLNX OFED – 4.1.x.x
  - HPC-X – 1.9.x
  - UFM ((Aggregation Manager only) - 5.8-5.9
- Prerequisites
  - Switch-IB 2 firmware - 15.1100.0072 or later
  - MLXN OS - 3.6.1002 or later
  - MLNX OpenSM 4.7.0 or later (available with MLNX OFED 3.3-x.x.x or UFM 5.6)

SHARP SW components

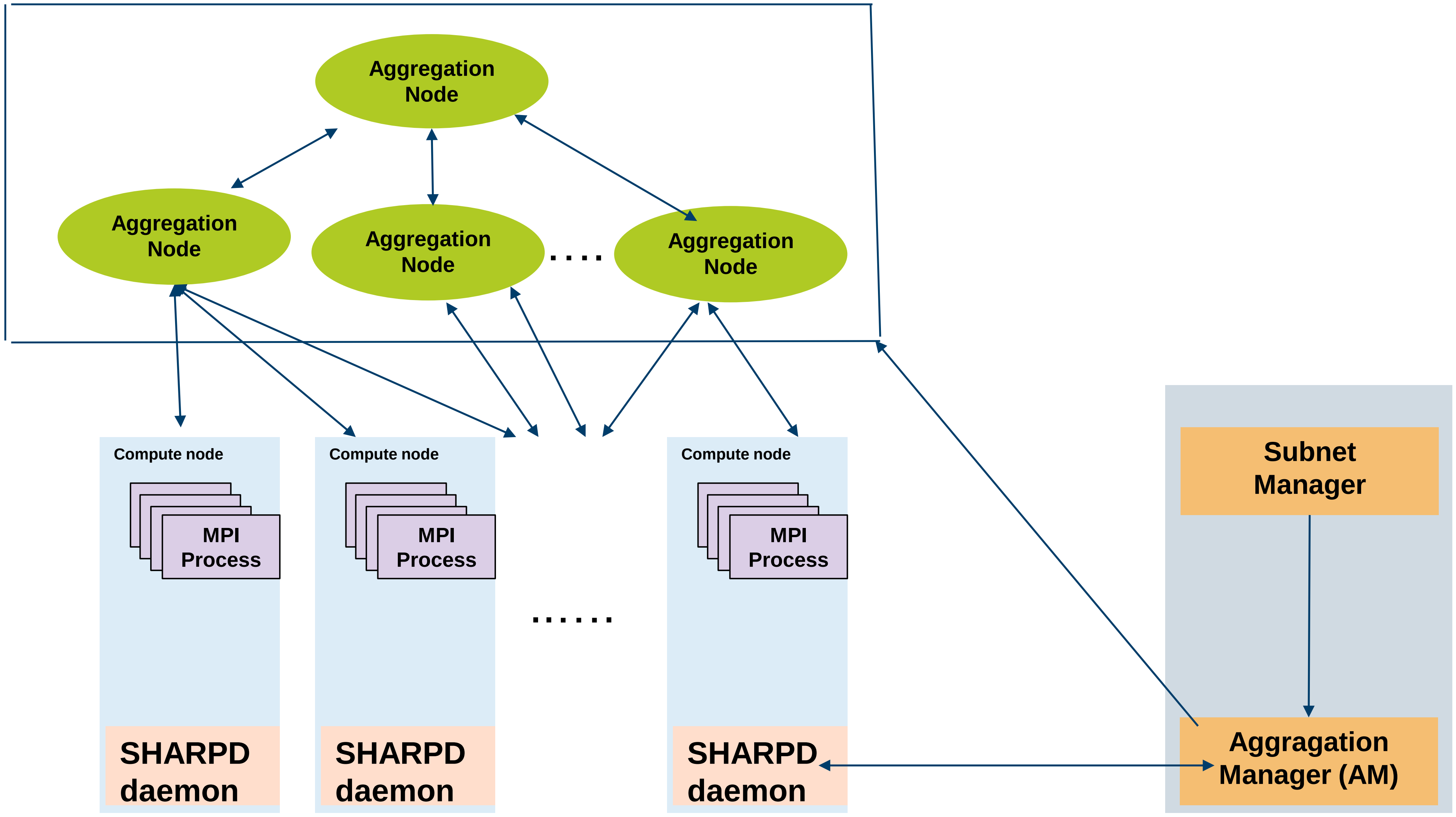
- SHArP SW components:
  - Libs
    - libsharp.so (low level api)
    - libsharp_coll.so (high level api)
  - Daemons
    - sharpd, sharp_am
  - Scripts
    - sharp_benchmark.sh
    - sharp_daemons_setup.sh
  - Utilities
    - sharp_coll_dump_config
    - sharp_hello
    - sharp_mpi_test
  - public API
    - sharp.h

HPCX/SHARP SW architecture

- HCOLL
  - optimized collective library
- Libsharp_coll.so
  - Implementation of high level sharp API for enabling sharp collectives for MPI
  - uses low level libsharp.so API
- Libsharp.so
  - Implementation of low level sharp API
- High level API
  - Easy to use
  - Easy to integrate with multiple MPIs(OpenMPI, MPICH, MVAPICH)



SHARP SW Architecture



SHArP: SHArP Daemons

- sharpd: SHArP daemon
  - compute nodes
  - Light wait process
  - Almost 0% cpu usage
  - Only control path

- sharp_am: Aggregation Manager daemon
  - same node as Subnet Manager
  - Resource manager

SHArP: Configuring Subnet Manager

- Edit the opensm.conf file.
- Configure the "routing_engine" parameter.
  - ftree fabric : routing_engine ftree,updn
  - hypercube fabric : routing_engine dor
- Set the parameter “sharp_enabled” to “2”.
- Run OpenSM with the configuration file.
  - % opensm -F <opensm configuration file> -B
- Verify that the Aggregation Nodes were activated by the OpenSM, run "ibnetdiscover".

For example:

```
vendid=0x0
devid=0xcf09
sysimgguid=0x7cfe900300a5a2a0
caguid=0x7cfe900300a5a2a8
Ca      1 "H-7cfe900300a5a2a8"          # "Mellanox Technologies Aggregation Node"
[1](7cfe900300a5a2a8)   "S-7cfe900300a5a2a0"[37]      # lid 256 lmc 0 "MF0;sharp2:MSB7800/U1" lid 512 4xFDR
```

•Create the sharp_am.conf file:

SHARP: Configuring Aggregation Manager

- Using OpensSM 4.9 or later does not require any special configuration in the AM.
- Configure AM with OpenSM v4.7-4.8:
  - Create a configuration directory for the future SHArP configuration file.

    -
      ```
      % mkdir $HPCX_SHARP_DIR/conf
      ```
  - Create root GUIDs file.
    - Copy the root_guids.conf file if used for configuration of Subnet Manager to $HPCX_SHARP_DIR/conf/root_guid.cfg (or)
    - Identify the root switches of the fabric and create a file with the node GUIDs of the root switches of the fabric.
    - For example : if there are two root switches files contains

      ```
      0x0002c90000000001
      0x0002c90000000008
      ```
  - Create sharp_am.conf file

    ```
    % cat > $HPCX_SHARP_DIR/conf/sharp_am.conf << EOF
    root_guids_file $HPCX_SHARP_DIR/conf/root_guid.cfg
    ib_port_guid <PortGUID of the relevant HCA port or 0x0>
    EOF
    ```

SHARP: Running SHARP Daemons

- Setup the daemons
  - $HPCX_SHARP_DIR/sbin/sharp_daemons_setup.sh

- Usage
  - Usage: sharp_daemons_setup.sh <-s> <-r> [-p SHArP location dir] <-d daemon> <-m>

```
-s - Setup SHArP daemon
-r - Remove SHArP daemon
-p - Path to alternative SHArP location dir
-d - Daemon name (sharpd or sharp_am)
-m - Add monit capability for daemon control
```

  - $HPCX_SHARP_DIR/sbin/sharp_daemons_setup.sh -s $HPCX_SHARP_DIR -d sharp_am
  - $service sharp_am start

SHARP: Running SHARP Daemons

sharp_am

- %$HPCX_SHARP_DIR/sbin/sharp_daemons_setup.sh -s $HPCX_SHARP_DIR -d sharp_am
- %service sharp_am start
- Log : /var/log/sharp_am.log

Sharpd

- conf file: $HPCX_SHARP_DIR/conf/sharpd.conf
  - ib_dev <relevant_hca:port>
  - sharpd_log_level 2
- %pdsh -w <hostlist> $HPCX_SHARP_DIR/sbin/sharp_daemons_setup.sh -s $HPCX_SHARP_DIR -d sharpd
- %pdsh -w jupiter[001-032] service sharpd start
- Log : /var/log/sharpd.log

MPI Collective offloads using SHARP

- Enabled through FCA-3.x (HCOLL)
- Flags
  - HCOLL_ENABLE_SHARP (default : 0)

    0 - Don't use SHArP

    1 - probe SHArP availability and use it

    2 - Force to use SHArP

    3 - Force to use SHArP for all MPI communicators

    4 - Force to use SHArP for all MPI communicators and for all supported collectives

  - HCOLL_SHARP_NP (default: 2)

    Number of nodes(node leaders) threshold in communicator to create SHArP group and use SHArP collectives

  - SHARP_COLL_LOG_LEVEL

    0 – fatal , 1 – error, 2 – warn, 3 – info, 4 – debug, 5 – trace

  - HCOLL_BCOL_P2P_ALLREDUCE_SHARP_MAX
    - Maximum allreduce size run through SHArP

MPI Collectives offloads using SHARP

- Resources (quota)
  - SHARP_COLL_JOB_QUOTA_MAX_GROUPS
    - #communicators
  - SHARP_COLL_JOB_QUOTA_OSTS
    - Parallelism on communicator
  - SHARP_COLL_JOB_QUOTA_PAYLOAD_PER_OST
    - Payload/OST
- For complete list of SHARP COLL tuning options
    - $HPCX_SHARP_DIR/bin/sharp_coll_dump_config -f

HPC-X:Point-to-Point Support

MXM and migrating over to UCX

MXM

Mellanox HPC-X - MXM

- Point to Point acceleration
- InfiniBand and RoCE
- Ease of use - simple API
- Uses multiple transports
  - RC, UD, DC, SHM, loopback
- Hybrid mode – mix transports
  - switching between them as needed.
- Increases scalability by using DC and/or UD
- Efficient memory registration
- Improves shared memory communication using process-to-process memcpy (KNEM)
- Support for hardware atomics

Mellanox HPC-X - MXM

- Thread-safe
  - Supports all MPI threading models
  - allow use of Hybrid model, i.e. MPI + OpenMP
- Re-use same protocols on different transports
  - Inline for small data
  - Fragmentation for medium data
  - Rendezvous for large data
- Scalability:
  - Fixed amount of buffers
  - Create connections on-demand
  - Reduce memory consumption per-connection
  - Scalable tag matching

Mellanox HPC-X – MXM Yalla

- Communications Library Support – MXM
  - In OpenMPI v1.8, a new pml layer(yalla) was added that reduces overhead by bypassing layers and using the MXM library directly.
    - for messages < 4K in size
      - ✓ Improves latency by 5%
      - ✓ Improves message rate by 50%
      - ✓ Improve bandwidth by up to 45%

    HPC-X will choose this pml automatically
  - To use this, pass the following command line in mpirun:
    - --mca pml yalla (instead of --mca pml cm –mca mtl mxm)

MXM

- MXM:
  - To enable MXM support explicitly:
    - -mca pml yalla
  - Transport selection
    - -x MXM_TLS=self,shm,**ud**
  - Device selection
    - -x MXM_RDMA_PORTS=mlx5_0:1 (if multiple interfaces existed)
  - For running low-level micro benchmarks, use RC transport
    - -x MXM_TLS=self,shm,**rc**
  - To disable MXM and enable openib BTL
    - --mca pml ob1

UCX

UCX

MXM

- Developed by Mellanox Technologies
- HPC communication library for InfiniBand devices and shared memory
- Primary focus: MPI, PGAS

UCCS

- Developed by ORNL, UH, UTK
- Originally based on Open MPI BTL and OPAL layers
- HPC communication library for InfiniBand, Cray Gemini/Aries, and shared memory
- Primary focus: OpenSHMEM, PGAS
- Also supports: MPI

PAMI

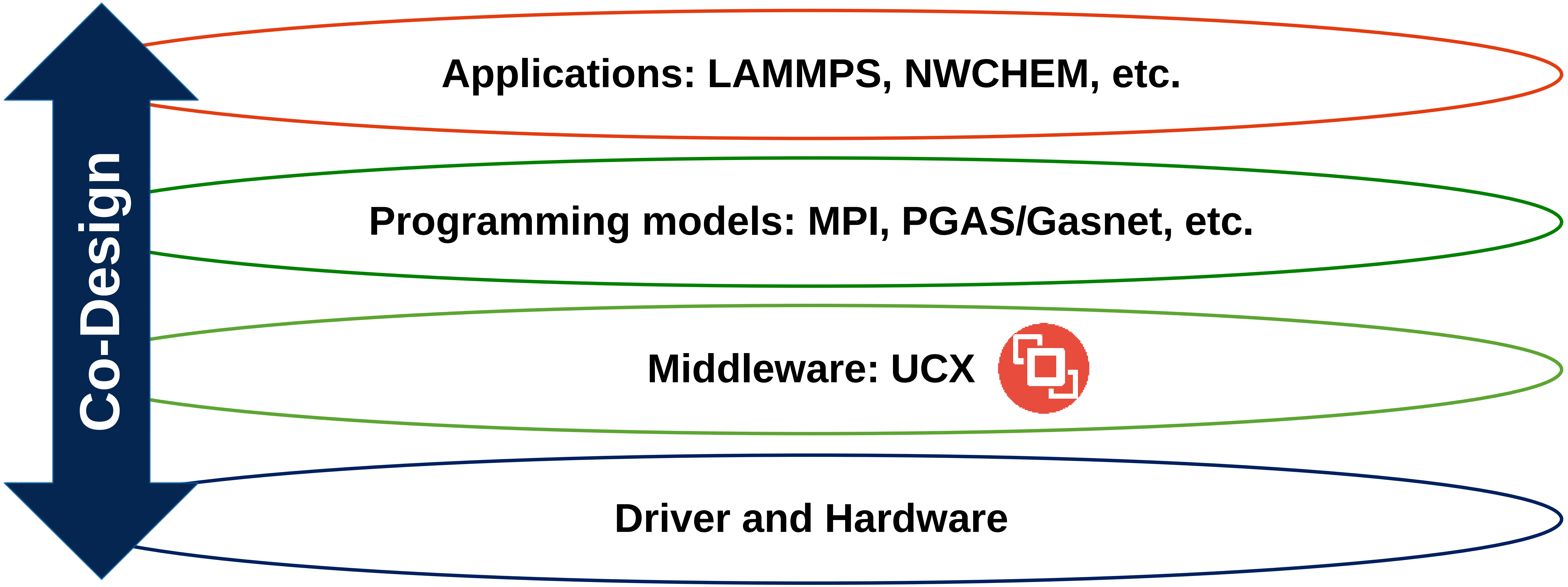
- Developed by IBM on BG/Q, PERCS, IB VERBS
- Network devices and shared memory
- MPI, OpenSHMEM, PGAS, CHARM++, X10
- C++ components
- Aggressive multi-threading with contexts
- Active Messages
- Non-blocking collectives with hw accleration support

UCX is an Integration of Industry and Users Design Efforts

UCX

- Collaboration between industry, laboratories, and academia
- To create open-source production grade communication framework for data centric and HPC applications
- To enable the highest performance through co-design of software-hardware interfaces

A Different Model - Co-Design Effort



Co-Design

Applications: LAMMPS, NWCHEM, etc.

Programming models: MPI, PGAS/Gasnet, etc.

Middleware: UCX

Driver and Hardware

Co-Design Effort Between National Laboratories, Academia, and Industry

UCX Framework Mission

- Collaboration between industry, laboratories, and academia
- Create open-source production grade communication framework for HPC applications
- Enable the highest performance through co-design of software-hardware interfaces
- Unify industry - national laboratories - academia efforts

API

Exposes broad semantics that target data centric and HPC programming models and applications

Performance oriented

Optimization for low-software overheads in communication path allows near native-level performance

Production quality

Developed, maintained, tested, and used by industry and researcher community

Community driven

Collaboration between industry, laboratories, and academia

Research

The framework concepts and ideas are driven by research in academia, laboratories, and industry

Cross platform

Support for Infiniband, Cray, various shared memory (x86-64 and Power), GPUs

Co-design of Exascale Network APIs

The UCX Framework

UC-S for Services

This framework provides basic infrastructure for component based programming, memory management, and useful system utilities

Functionality:
Platform abstractions, data structures, debug facilities.

UC-T for Transport

Low-level API that expose basic network operations supported by underlying hardware. Reliable, out-of-order delivery.

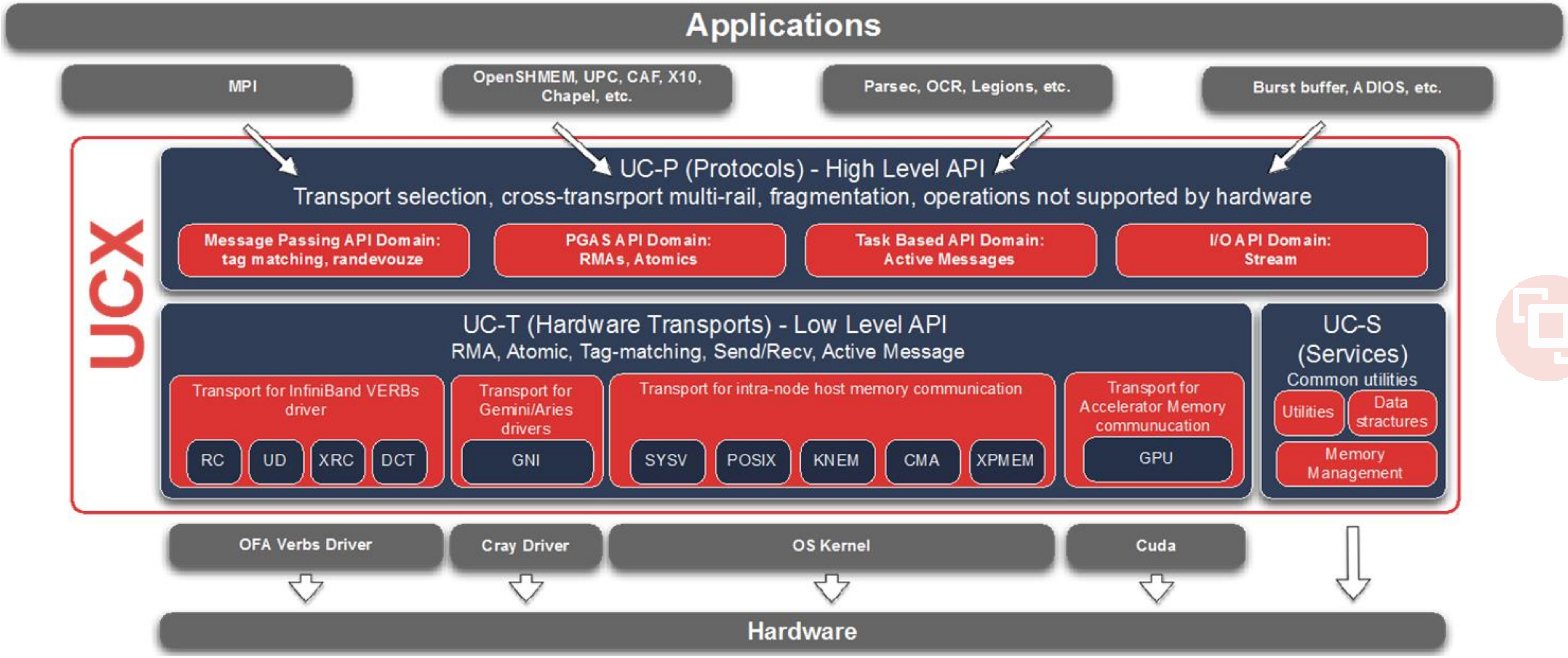
Functionality:
Setup and instantiation of communication operations.

UC-P for Protocols

High-level API uses UCT framework to construct protocols commonly found in applications

Functionality:
Multi-rail, device selection, pending queue, rendezvous, tag-matching, software-atomics, etc.

UCX High-level Overview


Applications
MPI
OpenSHMEM, UPC, CAF, X10, Chapel, etc.
Parsec, OCR, Legions, etc.
Burst buffer, ADIOS, etc.
UCX
UC-P (Protocols) - High Level API
Transport selection, cross-transrport multi-rail, fragmentation, operations not supported by hardware
Message Passing API Domain: tag matching, randevouze
PGAS API Domain: RMAs, Atomics
Task Based API Domain: Active Messages
I/O API Domain: Stream
UC-T (Hardware Transports) - Low Level API
RMA, Atomic, Tag-matching, Send/Recv, Active Message
Transport for InfiniBand VERBs driver
RC
UD
XRC
DCT
Transport for Gemini/Aries drivers
GNI
Transport for intra-node host memory communication
SYSV
POSIX
KNEM
CMA
XPMEM
Transport for Accelerator Memory communucation
GPU
UC-S (Services)
Common utilities
Utilities
Data structures
Memory Management
OFA Verbs Driver
Cray Driver
OS Kernel
Cuda
Hardware

Unified, Light-Weight, High-Performance Communication Framework

UCX

- OpenMPI
  - PML
  - OSC component
- OpenMPI/OpenSHMEM
  - SPML
- MPICH
- ORNL SHMEM
- SLURM PMIx
- Distributed TensorFlow – gRPC
- Caffe2*

Differences between UCX and MXM

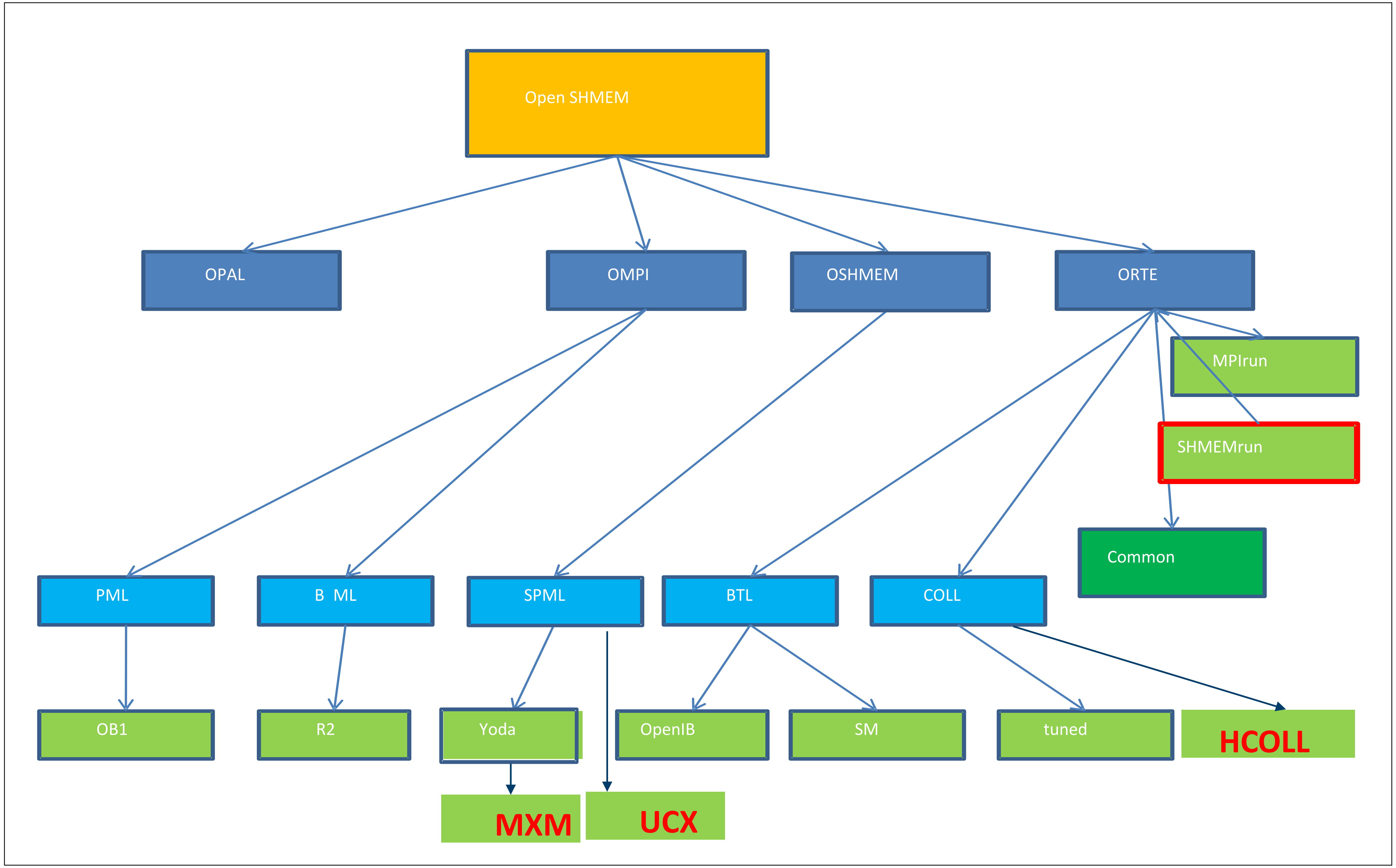
- Simple and consistent API
- Choosing between low-level and high-level API allows easy integration with a wide range of applications and middleware.
- Protocols and transports are selected by capabilities and performance estimations, rather than hard-coded definitions.
- Support thread contexts and dedicated resources, as well as fine-grained and coarse-grained locking.
- Accelerators are represented as a transport, driven by a generic “glue” layer, which will work with all communication networks.

HPC-X: OpenSHMEM Support

HPC-X: OpenSHMEM Support

- Major focus: Utilize Mellanox hardware capabilities to accelerate the OSHMEM implementation
  - Offload Network management
  - RDMA
  - Hardware atomic support
  - Dynamically Connected Transport
  - SHARP
  - DC/RC adaptive routing support
  - On Demand Paging
  - SHIELD
- V1.3 OpenSHMEM SPEC fully support
- SHMEM big job start improvements with PMIx
- OpenSHMEM collectives
  - SHARP supported (HCOLL)
  - OpenSHMEM shared memory collectives
- UCX support in spml UCX
- Data path performance improvements with direct verbs in UCX
  - Includes atomic support
- Added support for transparent huge page allocation
- Active participants in the OpenSHMEM community
  - Working groups of greatest interest
    - Collectives
    - Threading support

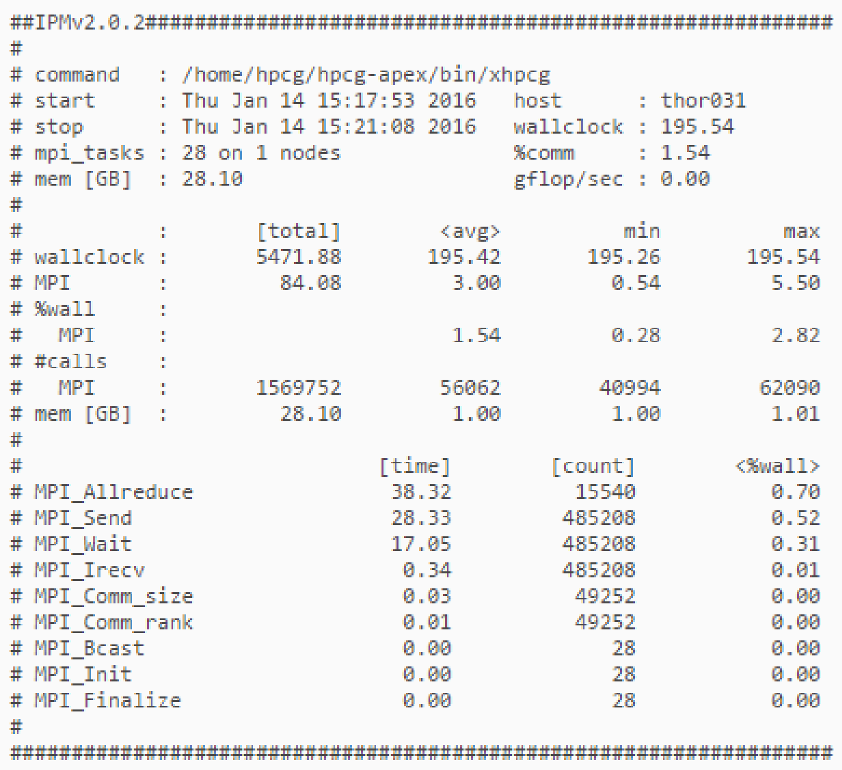
SHMEM Integration with Open MPI



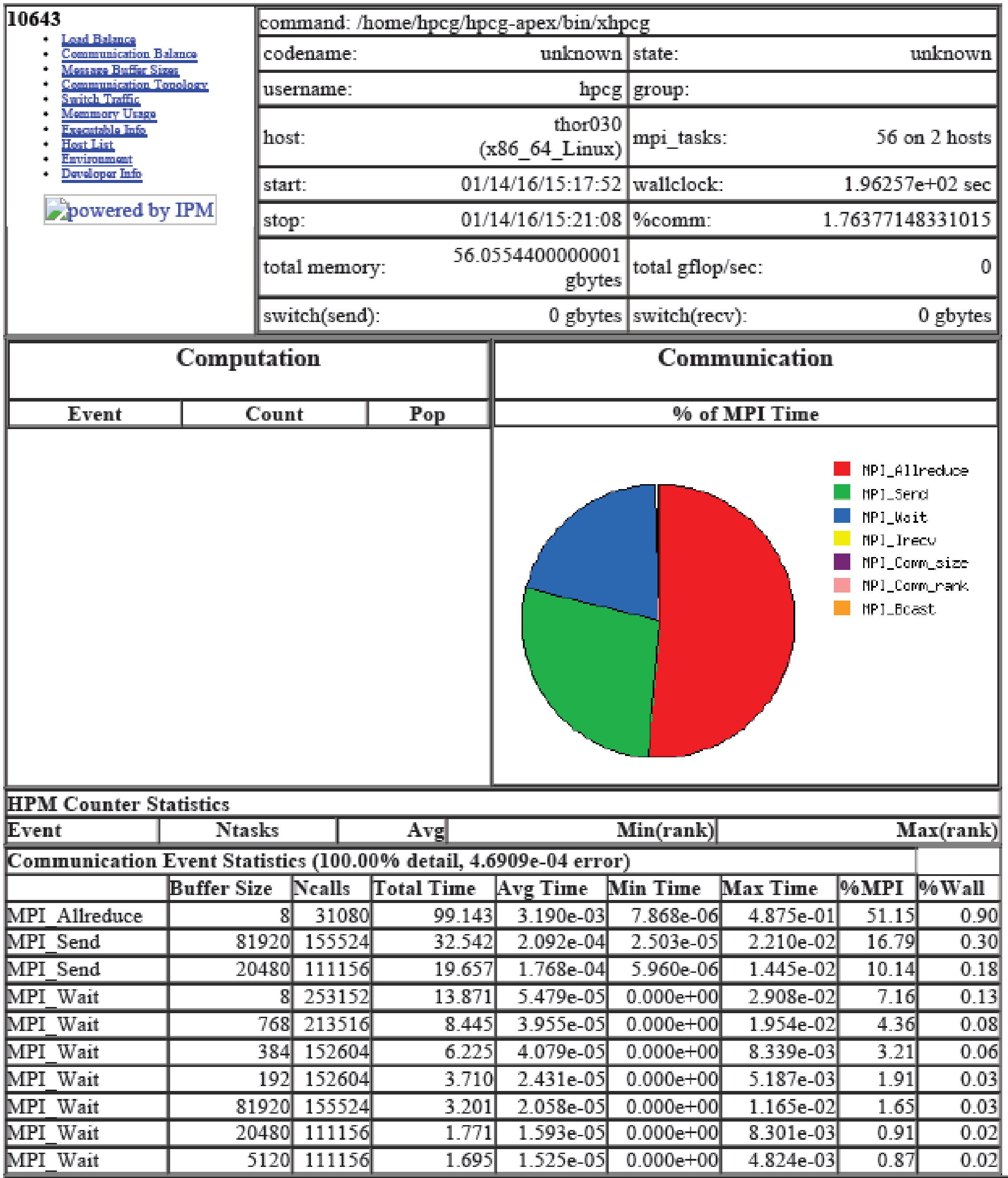
HPC-X: IPM Profiler

Profiling MPI

- To profile MPI API:
  - $ export IPM_KEYFILE=$HPCX_HOME_IPM_DIR/etc/ipm_key_mpi
  - $ export IPM_LOG=FULL
  - $ export LD_PRELOAD=$HPCX_HOME_IPM_DIR/lib/libipm.so
  - $ mpirun -x LD_PRELOAD
  - $ $HPCX_HOME_IPM_DIR/bin/ipm_parse **-full** outfile.xml
  - $ $HPCX_HOME_IPM_DIR/bin/ipm_parse -**htm**l outfile.xml



```
##IPMv2.0.2########################################################
#
# command   : /home/hpcg/hpcg-apex/bin/xhpcg
# start     : Thu Jan 14 15:17:53 2016   host      : thor031
# stop      : Thu Jan 14 15:21:08 2016   wallclock : 195.54
# mpi_tasks : 28 on 1 nodes              %comm     : 1.54
# mem [GB]  : 28.10                      gflop/sec : 0.00
#
#           :       [total]        <avg>         min         max
# wallclock :       5471.88       195.42      195.26      195.54
# MPI       :         84.08         3.00        0.54        5.50
# %wall     :
#   MPI     :                       1.54        0.28        2.82
# #calls    :
#   MPI     :       1569752        56062       40994       62090
# mem [GB]  :         28.10         1.00        1.00        1.01
#
#                             [time]        [count]        <%wall>
# MPI_Allreduce                38.32          15540           0.70
# MPI_Send                     28.33         485208           0.52
# MPI_Wait                     17.05         485208           0.31
# MPI_Irecv                     0.34         485208           0.01
# MPI_Comm_size                 0.03          49252           0.00
# MPI_Comm_rank                 0.01          49252           0.00
# MPI_Bcast                     0.00             28           0.00
# MPI_Init                      0.00             28           0.00
# MPI_Finalize                  0.00             28           0.00
#
###################################################################
```



10643

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memmory Usage
- Executable Info
- Host List
- Environment
- Developer Info

powered by IPM

command: /home/hpcg/hpcg-apex/bin/xhpcg			
codename:	unknown	state:	unknown
username:	hpcg	group:	
host:	thor030 (x86_64_Linux)	mpi_tasks:	56 on 2 hosts
start:	01/14/16/15:17:52	wallclock:	1.96257e+02 sec
stop:	01/14/16/15:21:08	%comm:	1.76377148331015
total memory:	56.0554400000001 gbytes	total gflop/sec:	0
switch(send):	0 gbytes	switch(recv):	0 gbytes

Computation

Event	Count	Pop

Communication

% of MPI Time

HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)

Communication Event Statistics (100.00% detail, 4.6909e-04 error)

	Buffer Size	Ncalls	Total Time	Avg Time	Min Time	Max Time	%MPI	%Wall
MPI_Allreduce	8	31080	99.143	3.190e-03	7.868e-06	4.875e-01	51.15	0.90
MPI_Send	81920	155524	32.542	2.092e-04	2.503e-05	2.210e-02	16.79	0.30
MPI_Send	20480	111156	19.657	1.768e-04	5.960e-06	1.445e-02	10.14	0.18
MPI_Wait	8	253152	13.871	5.479e-05	0.000e+00	2.908e-02	7.16	0.13
MPI_Wait	768	213516	8.445	3.955e-05	0.000e+00	1.954e-02	4.36	0.08
MPI_Wait	384	152604	6.225	4.079e-05	0.000e+00	8.339e-03	3.21	0.06
MPI_Wait	192	152604	3.710	2.431e-05	0.000e+00	5.187e-03	1.91	0.03
MPI_Wait	81920	155524	3.201	2.058e-05	0.000e+00	1.165e-02	1.65	0.03
MPI_Wait	20480	111156	1.771	1.593e-05	0.000e+00	8.301e-03	0.91	0.02
MPI_Wait	5120	111156	1.695	1.525e-05	0.000e+00	4.824e-03	0.87	0.02

Profiling MPI

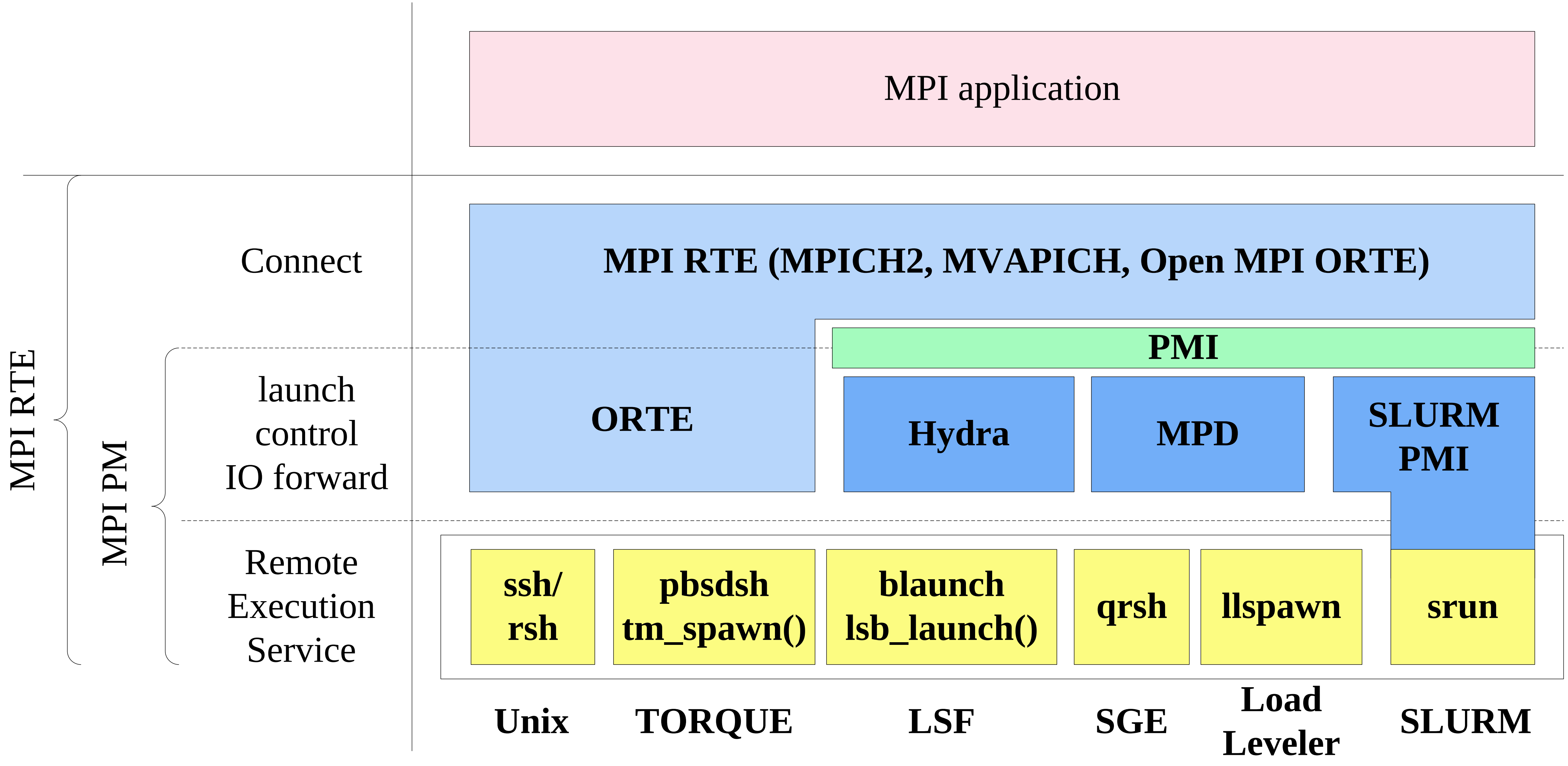
- To troubleshoot for MPI load imbalance
  - Apply blocking before MPI collective operations
  - Show the effect when processes not synchronized before entering into MPI collective ops
- Instrumentation can be applied on per-collective basis
  - $ export IPM_ADD_BARRIER_TO_REDUCE=1
  - $ export IPM_ADD_BARRIER_TO_ALLREDUCE=1
  - $ export IPM_ADD_BARRIER_TO_GATHER=1
  - $ export IPM_ADD_BARRIER_TO_ALL_GATHER=1
  - $ export IPM_ADD_BARRIER_TO_ALLTOALL=1
  - $ export IPM_ADD_BARRIER_TO_ALLTOALLV=1
  - $ export IPM_ADD_BARRIER_TO_BROADCAST=1
  - $ export IPM_ADD_BARRIER_TO_SCATTER=1
  - $ export IPM_ADD_BARRIER_TO_SCATTERV=1
  - $ export IPM_ADD_BARRIER_TO_GATHERV=1
  - $ export IPM_ADD_BARRIER_TO_ALLGATHERV=1
  - $ export IPM_ADD_BARRIER_TO_REDUCE_SCATTER=1

PMIx

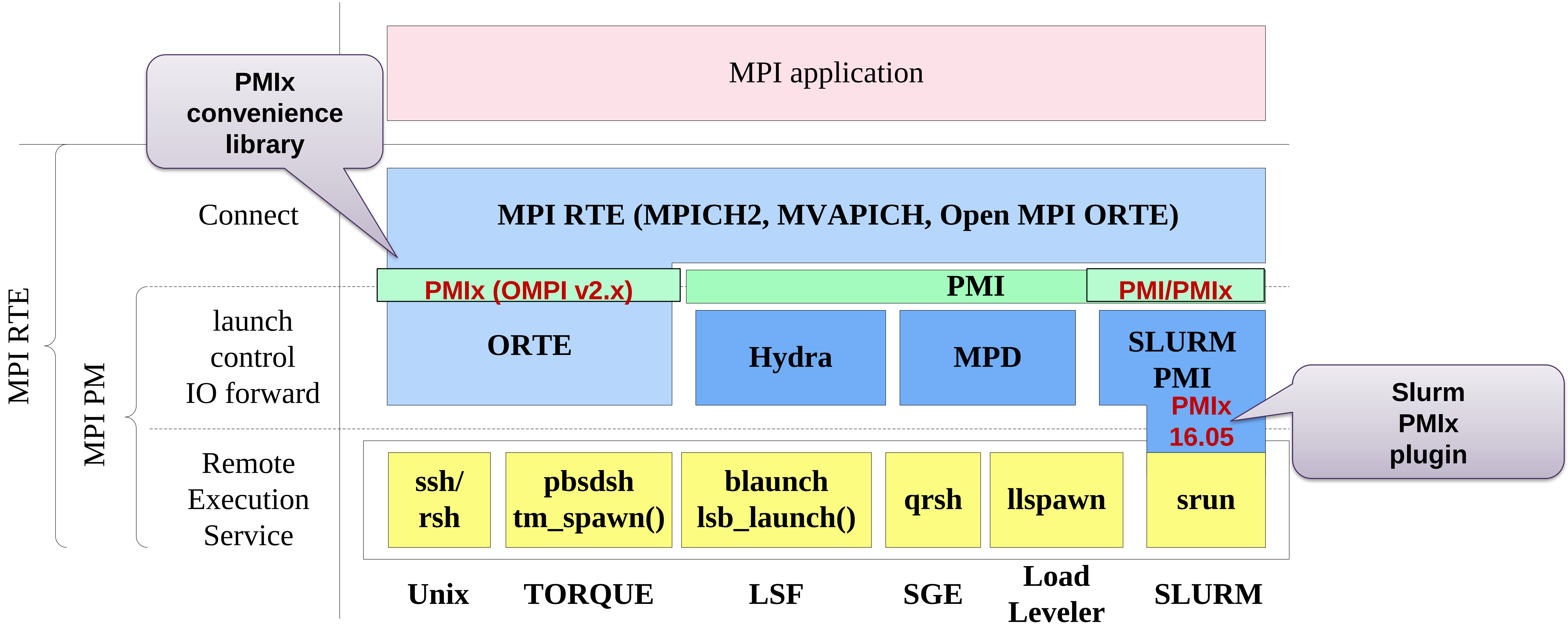
PMIx – PMI exascale

- Collaborative open source effort led by Intel, Mellanox Technologies, IBM, Adaptive Computing, and SchedMD…

Process Management Interface: RTE – application



Process Management Interface: RTE – application (2)



PMIx motivation

- Exascale launch times are a hot topic
  - Desire: reduce from many minutes to few seconds
  - Target: O(10^6) MPI processes on O(10^5) nodes thru MPI_Init in < 30 seconds
- New programming models are exploding
  - Driven by need to efficiently exploit scale vs. resource constraints
  - Characterized by increased application-Resource Manager integration

PMIx creation and goals

- Result of a close work of Vendors, OEMs, customers and open source community.
- Goal: to design a scalable API that addresses measured limitations of PMI2.
- Data driven design.
- Two generations of API:
  - PMIx API v1 basic functionality targeting scalable job launch
    - Concept of “direct modex”:
      - allows to avoid global synchronization overhead for loosely connected applications.
      - process contact data requested directly from the PMIx server that manages this process.
      - PMIx_Get has the argument that allows to specify the rank that submitted requested key.
    - Concept of a data scope:
      - allows to reduce memory footprint and communication overhead
      - “local” (intra-node only), “remote” (for processes from other nodes only), “global” for everybody.
    - Rich and extendable job-level information:
      - allows Resource Manager (RM) to provide the information that is already known to it to reduce initialization overhead.
      - Example: local ranks, hwloc topology, binding information, etc.
  - PMIx API v2 introduces wider functionality for:
      - communication with RM
      - fault tolerance features
      - Debuggers support
      - etc.

PMIx implementations

PMIx community develops and maintains PMIx convenience library.

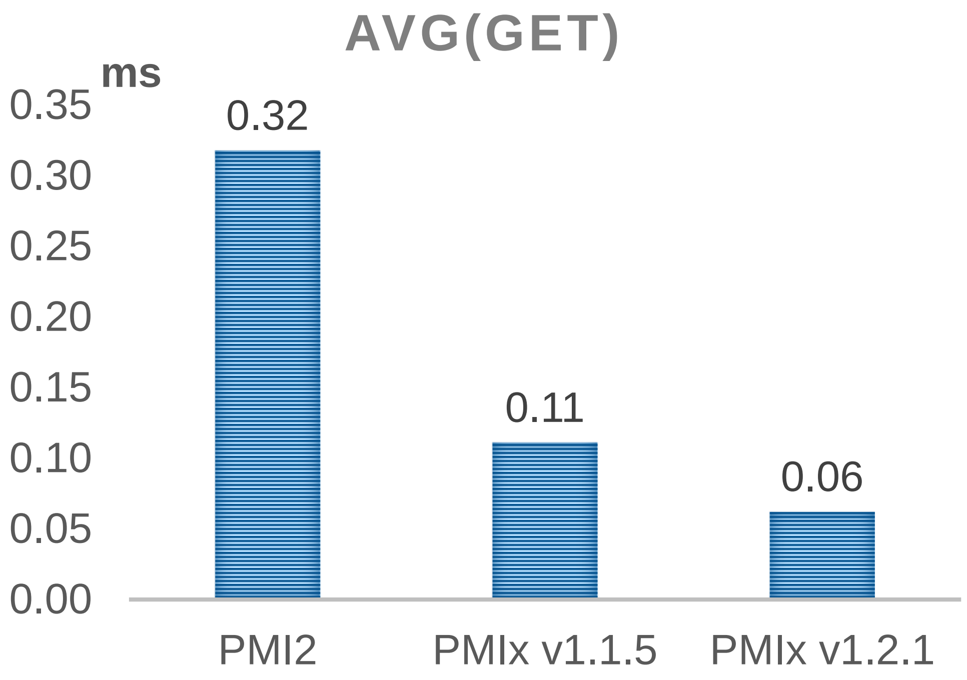
- Convenience library:
  - Available on GitHub: https://github.com/pmix/pmix/releases.
  - Provides reference implementation of PMIx API.
  - Implements the client-side API interface to the application.
  - Implements the server-side API to Resource Managers to encapsulate client-server communication. This simplifies RM integration process.
- All generations of API (major version number) are implemented and available:
  - PMIx API v1: most recent release is PMIx v1.2.2
  - PMIx API v2: most recent release is PMIx v2.0.0
- Basic implementation decisions was data driven and based on the analysis of existing PMI codes:
  - Data blobs are used to encapsulate all of the keys submitted by a rank. All keys are requested at once reducing server-server communication overhead in direct modex mode.
  - Shared memory is used for intra-node communication (client-server) to reduce memory footprint and data access latency (starting from PMIx v1.2.0).

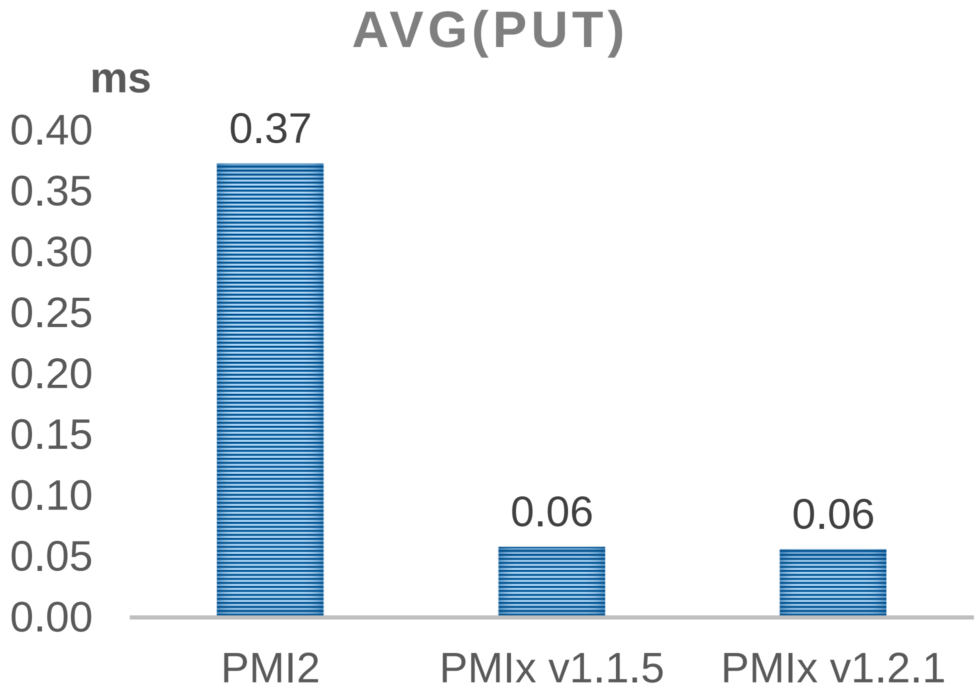
PMIx key access latency

PMIx convenience library has a `pmix_perf` tool to measure PMIx performance characteristics (see `contrib/perf_tools` directory). This tool allows to measure PMI operations performance and memory footprint for PMIx and PMI2 implementations.

Below the comparison of Get and Put operations conducted with Slurm implementations of PMI2 and PMIx is provided:

- System: 64 32-core Intel Xeon nodes.
- Both versions of PMIx demonstrate same Put latency as all keys are cached locally on the client side until PMIx_Commit is called. The latency of PMI2 implementation is much higher because each Put operation results in client-server communications: https://github.com/SchedMD/slurm/blob/master/contribs/pmi2/pmi2_api.c#L708:L730
- For each Get operation: a) PMI2 implementation performs client-server communication; b) PMIx v1.1.5 does one such communication per rank; while c) v1.2.1 uses shared memory to independently access the data.





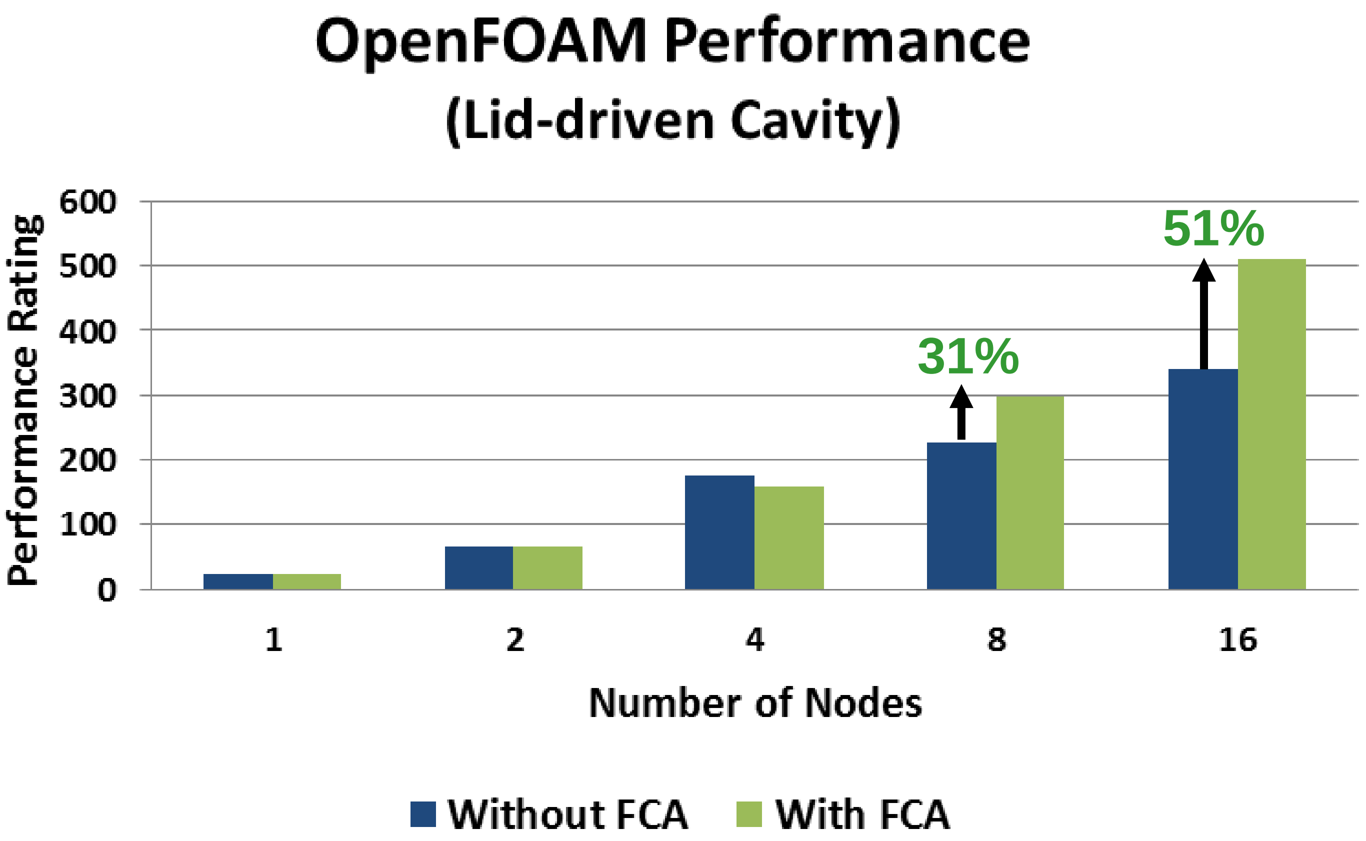
PMIx integration

- PMIx integration
  - Open MPI (starting from v2.0.0)
    - Significant architectural changes support “direct modex” :
      “Add procs” in bulk MPI_Init → “Add proc” on-demand on first use outside MPI_init.
    - Both native launching and direct launching under supported RMs is available.
    - Client side framework added to OPAL with components for: Cray PMI, PMI1, PMI2, PMIx.
  - Slurm (starting from v16.05)
    - Slurm v16.05:
      - basic support of PMIx v1.x.
      - Slurm RPC based inter-server communication.
    - Slurm v17.11:
      - basic support for PMIx v2.x is added (PMIx v1.x is still supported)
        - `<slurm sources> $./configure … --with-pmix=“<pmix-v1.x-path>:<pmix-v2.x-path>”`
        - Command above will result in 2 plugin binaries to be generated: pmix_v1 and pmix_v2.
        - One can select between installed libraries by selecting the plugin at runtime:
        - `~ $ srun --mpi=pmix_v2 ./mpi_hello_world`
      - Improvements in inter-node communications
        - According to conducted measurements Slurm RPC is convenient but has high latency
        - The work to enable UCX and TCP based inter-server communication was performed in 2017.
          - » TCP and UCX support is already accepted into Slurms main repository
          - » Collective operations refactoring is under review and scheduled for Slurm 17.11.

Application Case Studies with HPCX

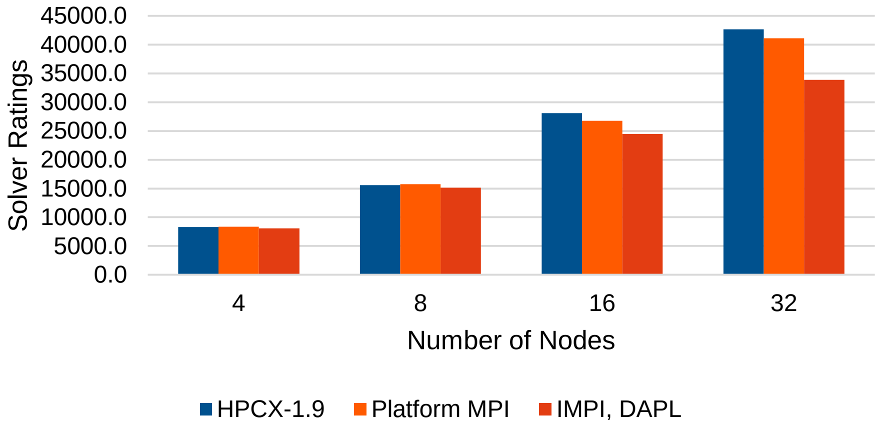
OpenFOAM Performance – FCA

- FCA enables nearly 51% performance gain at 16 nodes / 256 cores
  - Bigger advantage expected at higher node count / core count

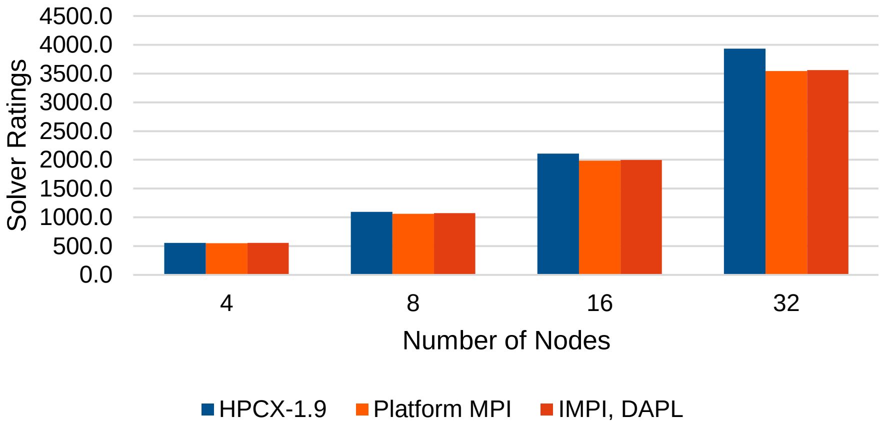


ANSYS Fluent

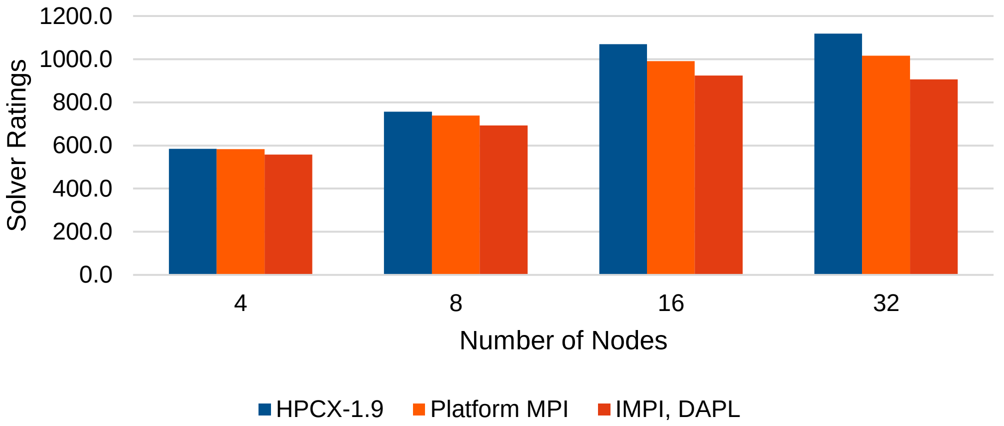
ANSYS Fluent Performance (fluidized_bed_2m)



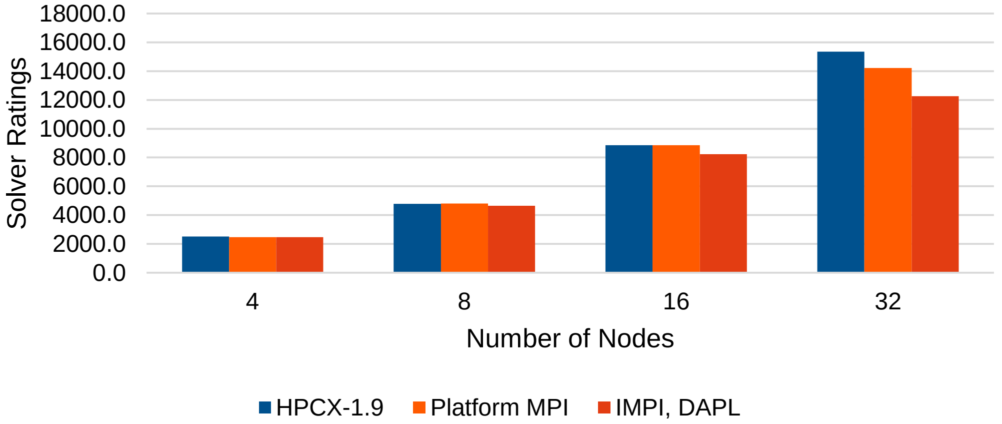
ANSYS Fluent Performance (combustor_12m)



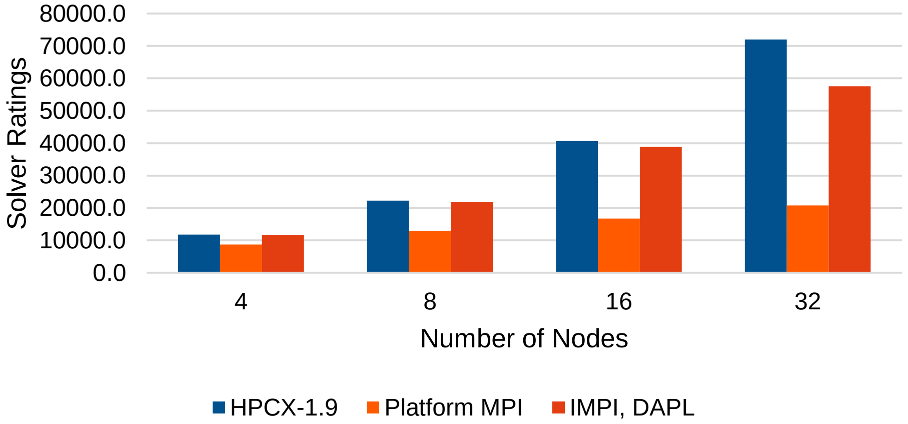
ANSYS Fluent Performance (ice_2m)



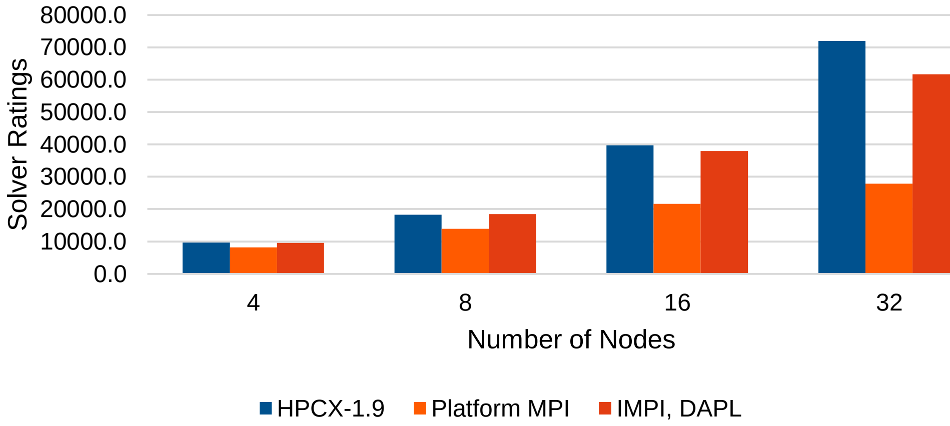
ANSYS Fluent Performance (oil_rig_7m)



ANSYS Fluent Performance (rotor_3m)

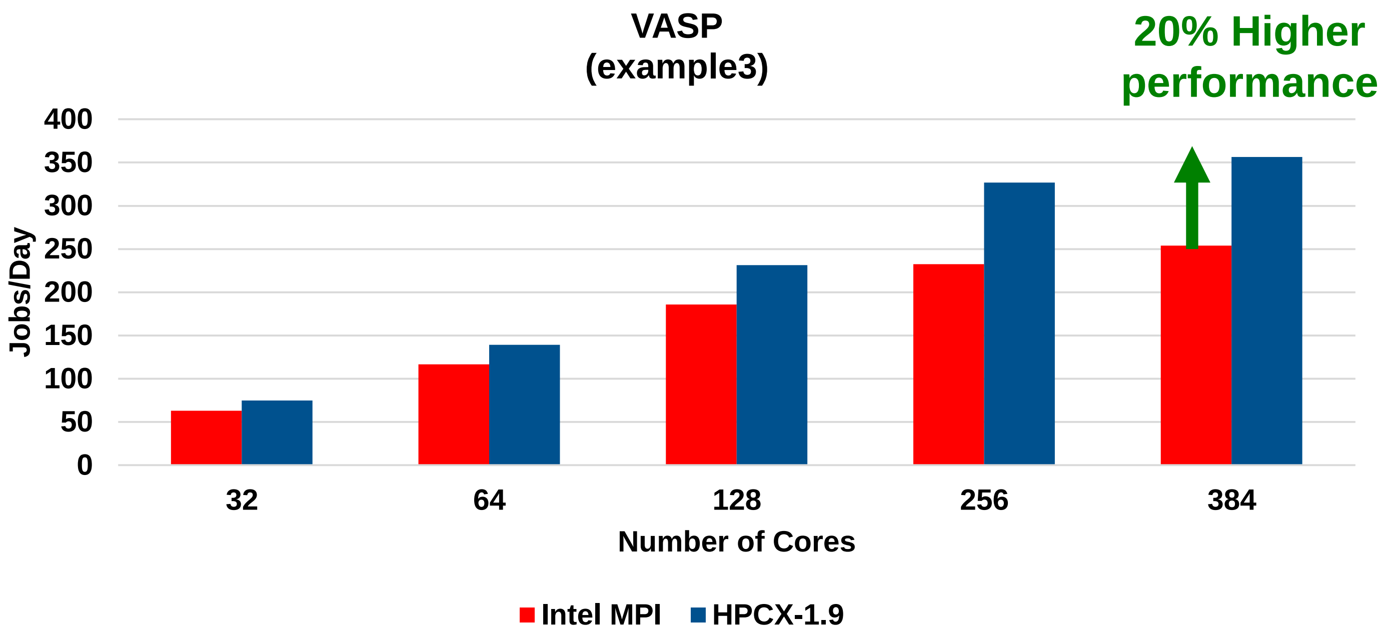


ANSYS Fluent Performance (pump_2m)



Application Performance – VASP

Stands for "Vienna Ab-initio Simulation Package". Quantum-mechanical molecular dynamics simulation software



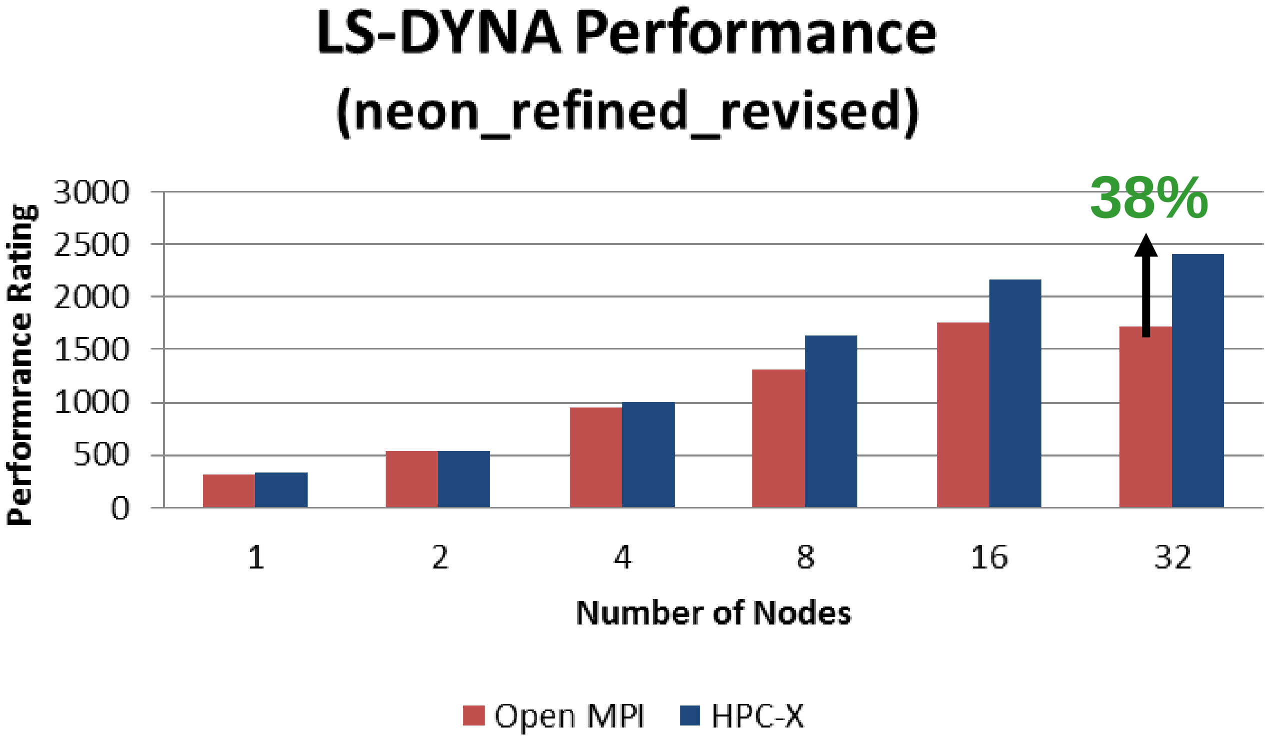
20% Higher performance

HPC-X Delivers 20% Higher Performance and Superior Scaling

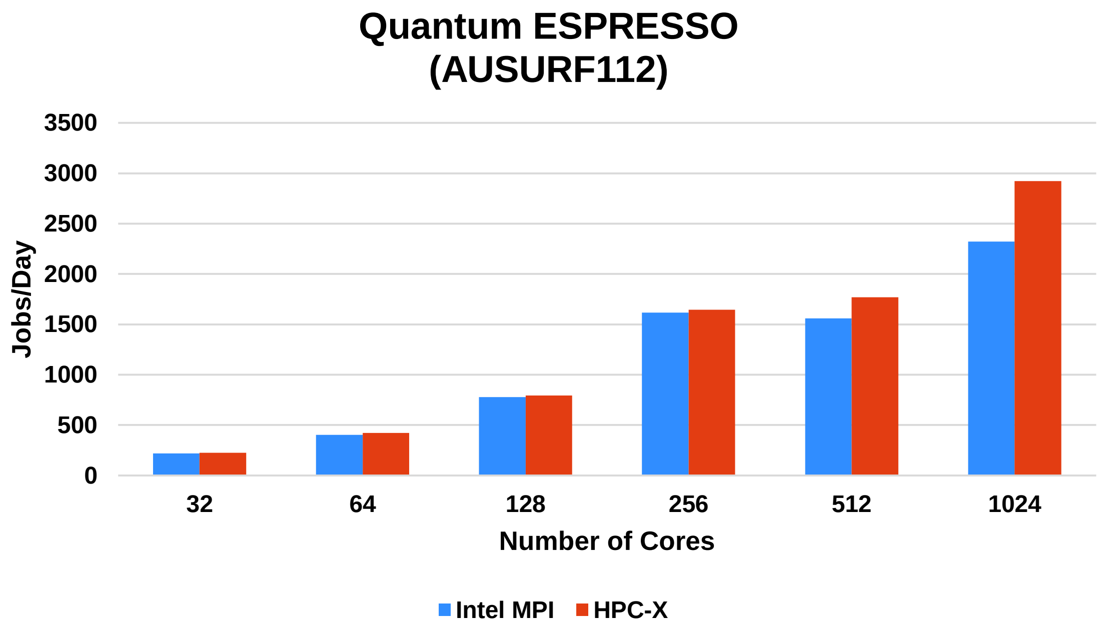
LS-DYNA Performance – MPI Optimization

- FCA and MXM enhance LS-DYNA performance at scale for HPC-X
  - Open MPI and HPC-X are based on the Open MPI distribution
  - The “yalla” PML, UD transport and memory optimization in HPC-X reduce overhead
  - MXM provides a speedup of 38% over un-tuned baseline run at 32 nodes (768 cores)
- MCA parameters for MXM:
  - For enabling MXM:

```
-mca btl_sm_use_knem 1 -mca pml yalla -x MXM_TLS=ud,shm,self -x MXM_SHM_RNDV_THRESH=32768 -x
MXM_RDMA_PORTS=mlx5_0:1
```



Quantum ESPRESSO



Best Practices (HPC-X vs Other MPIs)

- ANSYS Fluent
  - HPC-X provides 19% higher performance than Intel MPI on 32 nodes
  - http://www.hpcadvisorycouncil.com/pdf/Fluent_Analysis_and_Profiling_Intel_E5_2680v2.pdf
- CD-adapco STAR-CCM+
  - Up to 21% higher than Intel MPI and Platform MPI, at 32 nodes
  - http://www.hpcadvisorycouncil.com/pdf/STAR-CCM_Analysis_Intel_E5_2680_V2.pdf
- OpenFOAM
  - HPC-X delivers 13% gain over Intel MPI on 32 nodes
  - HPC-X demonstrates 24% improvement over Open MPI
  - http://www.hpcadvisorycouncil.com/pdf/OpenFOAMConf2015_PakLui.pdf
- QuantumESPRESSO
  - HPC-X delivers 103% gain over Intel MPI on 32 nodes
  - http://www.hpcadvisorycouncil.com/pdf/QuantumEspresso_Performance_Analysis_Intel_Haswell.pdf
- WRF
  - HPC-X delivers 13% performance improvement over Platfrom MPI on 32 nodes
  - http://www.hpcadvisorycouncil.com/pdf/WRF_Analysis_and_Profiling_Intel_E5-2680v2.pdf

Summary

- Mellanox solutions provide a proven, scalable and high performance end-to-end connectivity
- Flexible, support all compute architectures: x86, ARM, GPU, FPGA etc.
- Standards-based (InfiniBand, Ethernet), supported by large eco-system
- Higher performance: 200Gb/s, 0.6usec latency, 200 million messages/sec
- In-network comuputing
- HPC-X software provides leading performance for MPI, OpenSHMEM/PGAS and UPC
- Superiors applications offloads: RDMA, Collectives, scalable transport
- Backward and future compatible

Thank You

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.