Open MPI State of the Union X
Community Meeting SC '16

Jeff Squyres
CISCO

George Bosilca
ICL UT

Perry Schmidt
IBM

Ralph Castain
intel

Yossi Itigin
Mellanox TECHNOLOGIES

Nathan Hjelm
Los Alamos NATIONAL LABORATORY EST. 1943

Open MPI State of the Union X
Community Meeting SC ‘16

10 years of SC Open MPI BOFs!

Public service announcement

www.mcs.anl.gov/eurompi2017/

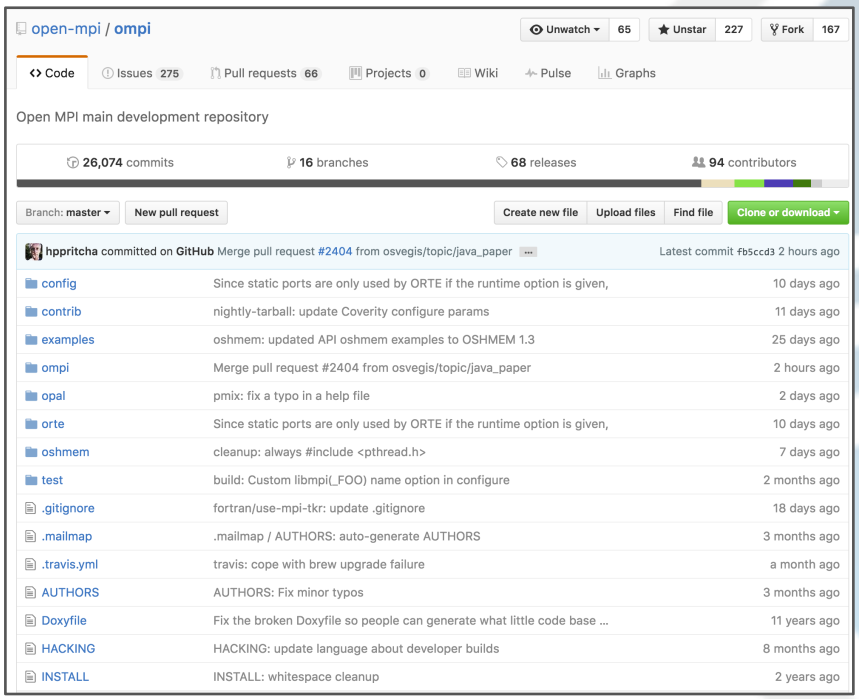
- CFP online at web site
- Location: Argonne National Laboratory (Chicago, Illinois, USA)

- Full paper submission deadline: 1st May 2016

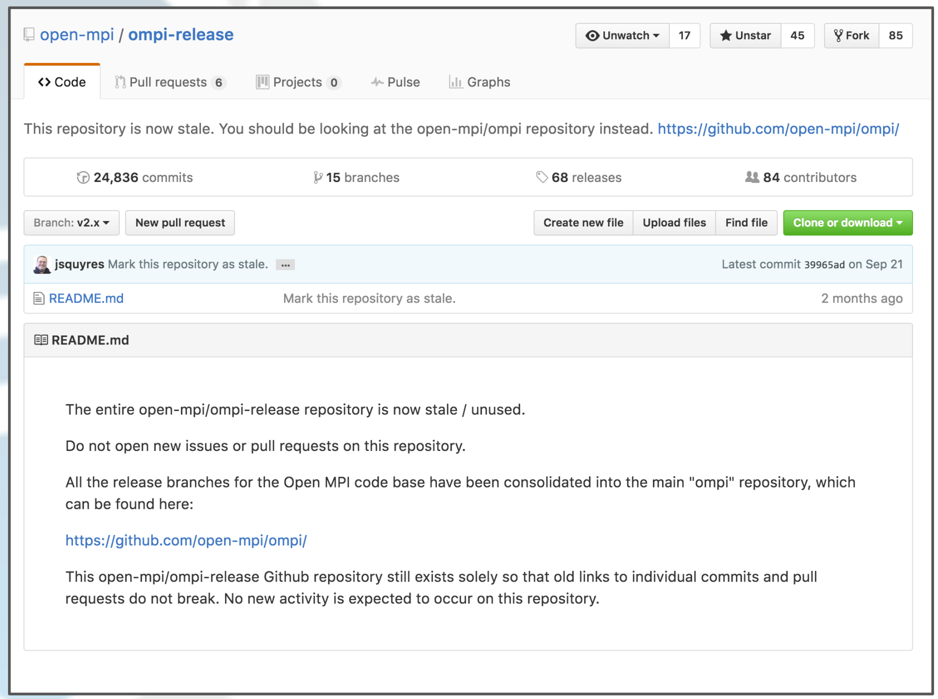
Github / Community Update

Github: we used to have 2 repos

ompi



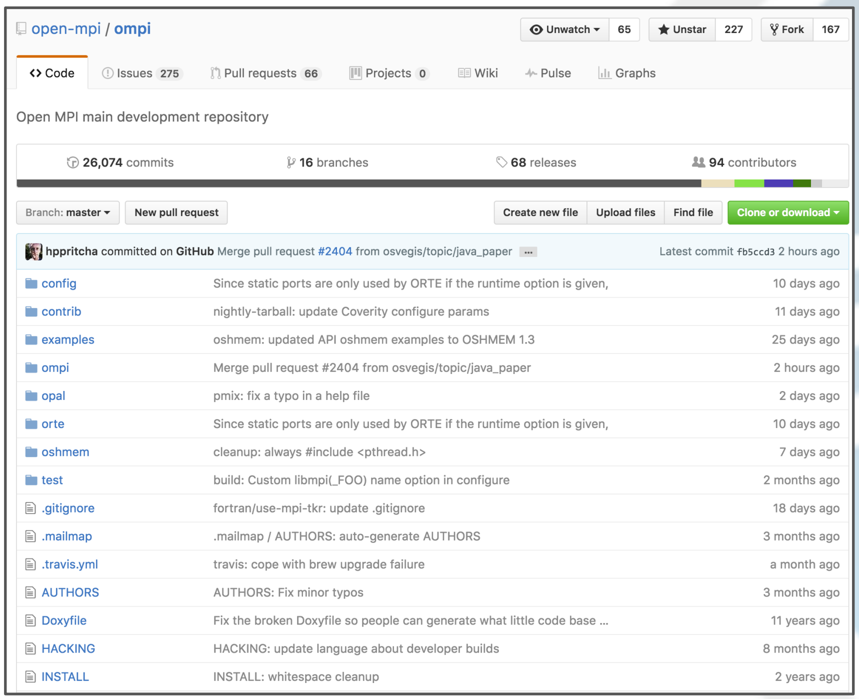
ompi-release



Github: we used to have 2 repos

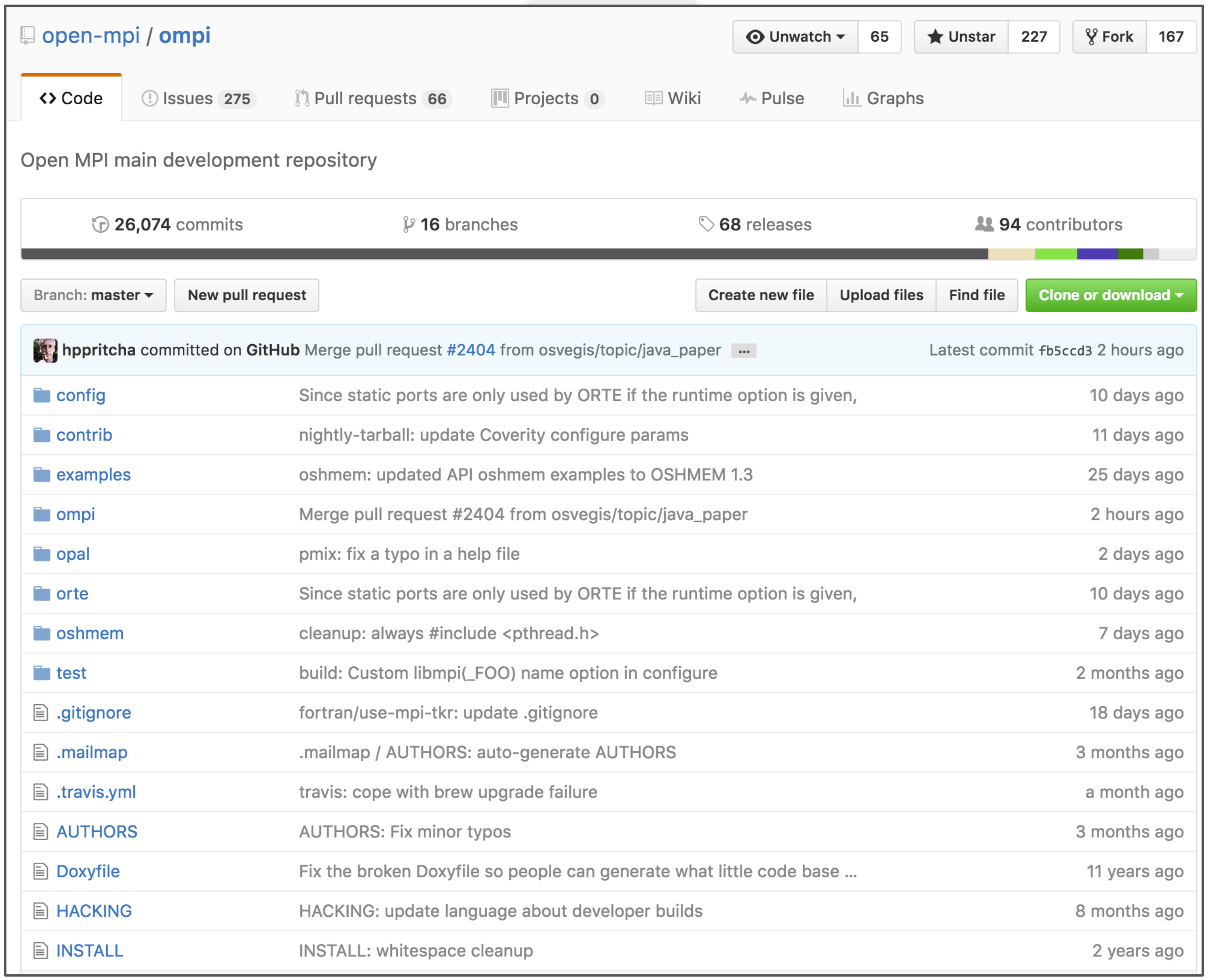
ompi

ompi-release



Github: now we have just one repo

ompi



Contribution policy

- For 10+ years, we have required a signed contribution agreement for “sizable” code contributions

The Open MPI Project
Software Grant and Corporate Contributor License Agreement (“Agreement”)
http://www.open-mpi.org/community/contribute/
(v1.6)

Thank you for your interest in The Open MPI Project (the “Project”). In order to clarify the intellectual property license granted with Contributions from any person or entity, the Open MPI Project copyright holders (the “Copyright Holders”) must have a Contributor License Agreement (CLA) on file that has been signed by each Contributor, indicating agreement to the license terms below. This license is for your protection as a Contributor as well as the protection of the Copyright Holders and their users; it does not change your rights to use your own Contributions for any other purpose.

This version of the Agreement allows an entity (the "Corporation") to submit Contributions to the Copyright Holders, to authorize Contributions submitted by its designated employees to the Copyright Holders, and to grant copyright and patent licenses thereto.

If you have not already done so, please complete and send a signed Agreement to ompi-contributors@lists.open-mpi.org. Please read this document carefully before signing and keep a copy for your records.

Corporation name: ________________________________

Corporation address: ________________________________

____

____

Point of Contact: ________________________________

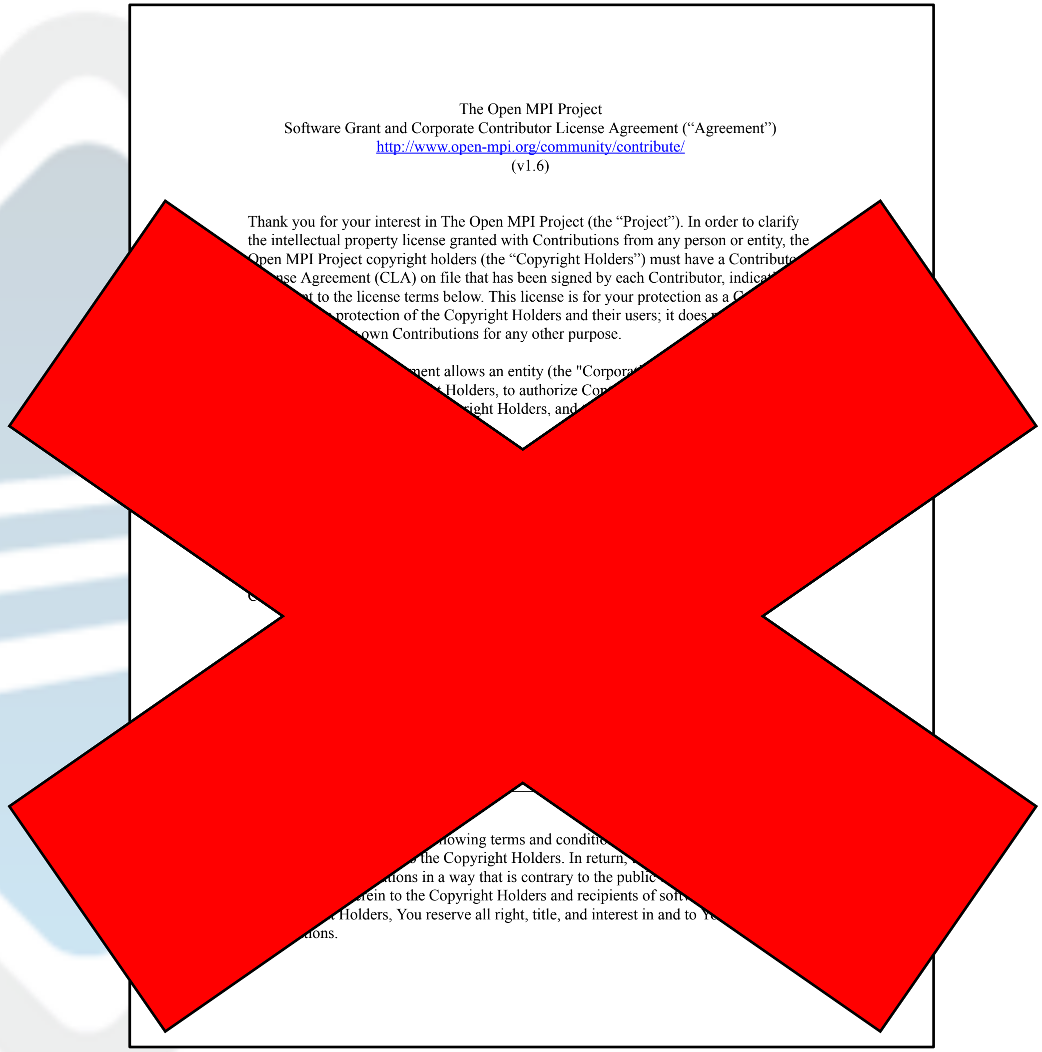
E-Mail: ________________________________

Telephone: ________________ Fax: ________________

You accept and agree to the following terms and conditions for Your present and future Contributions submitted to the Copyright Holders. In return, the Copyright Holders shall not use Your Contributions in a way that is contrary to the public benefit. Except for the license granted herein to the Copyright Holders and recipients of software distributed by the Copyright Holders, You reserve all right, title, and interest in and to Your Contributions.

Contribution policy

- This is no longer necessary


The Open MPI Project
Software Grant and Corporate Contributor License Agreement (“Agreement”)
http://www.open-mpi.org/community/contribute/
(v1.6)

Thank you for your interest in The Open MPI Project (the “Project”). In order to clarify the intellectual property license granted with Contributions from any person or entity, the

Contribution policy

- Instead, we now require a “Signed-off-by” token in commit messages

```
Some awesome new feature

Signed-off-by: Jeff Squyres <jsquyres@cisco.com>
```

- Can automatically be added by “`git commit –s`”

Signed-off-by

- Intent: make it easier for individuals and organizations to contribute to Open MPI

- “Signed-off-by” means agreement to the Open MPI Contributor’s Declaration
  - See the full definition here
  - This is common in many open source projects

Contributor’s Declaration

"By making a contribution to this project, I certify that:

1. The contribution was created in whole or in part by me and I have the right to submit it under the Open MPI open source license; or
2. The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the Open MPI open source license (unless I am permitted to submit under a different license); or
3. The contribution was provided directly to me by some other person who certified (1) or (2) and I have not modified it.
4. I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project and the open source license(s) involved."

Open MPI versioning

Open MPI versioning

- Open MPI will (continue to) use a “A.B.C” version number triple
- Each number now has a specific meaning:

A	This number changes when backwards compatibility breaks
B	This number changes when new features are added
C	This number changes for all other releases

Definition

- Open MPI vY is ***backwards compatible*** with Open MPI vX (where Y>X) if:
  - Users can compile a correct MPI / OSHMEM program with vX
  - Run it with the same CLI options and MCA parameters using vX or vY
  - The job executes correctly

What does that encompass?

- “Backwards compatibility” covers several areas:
  - Binary compatibility, specifically the MPI / OSHMEM API ABI
  - MPI / OSHMEM run time system
  - `mpirun` / `oshrun` CLI options
  - MCA parameter names / values / meanings

What does that *not* encompass?

- Open MPI only supports running exactly the same version of the runtime and MPI / OSHMEM libraries in a single job
  - If you mix-n-match vX and vY in a single job…

ERROR

Current version series

- v1.10.x series
  - Older, stable, rapidly hitting end of life
- v2.0.x series
  - Current stable series
- v2.x series
  - Upcoming series

v1.10.x Roadmap

v1.10.x release manager

- Ralph Castain, Intel

v1.10 series

- Soon to be end of life
- One more release expected: v1.10.5
  - Bug fixes only – no new features
  - Do not have a specific timeframe

If you are still running v1.10.x,
please start migrating to v2.0.x

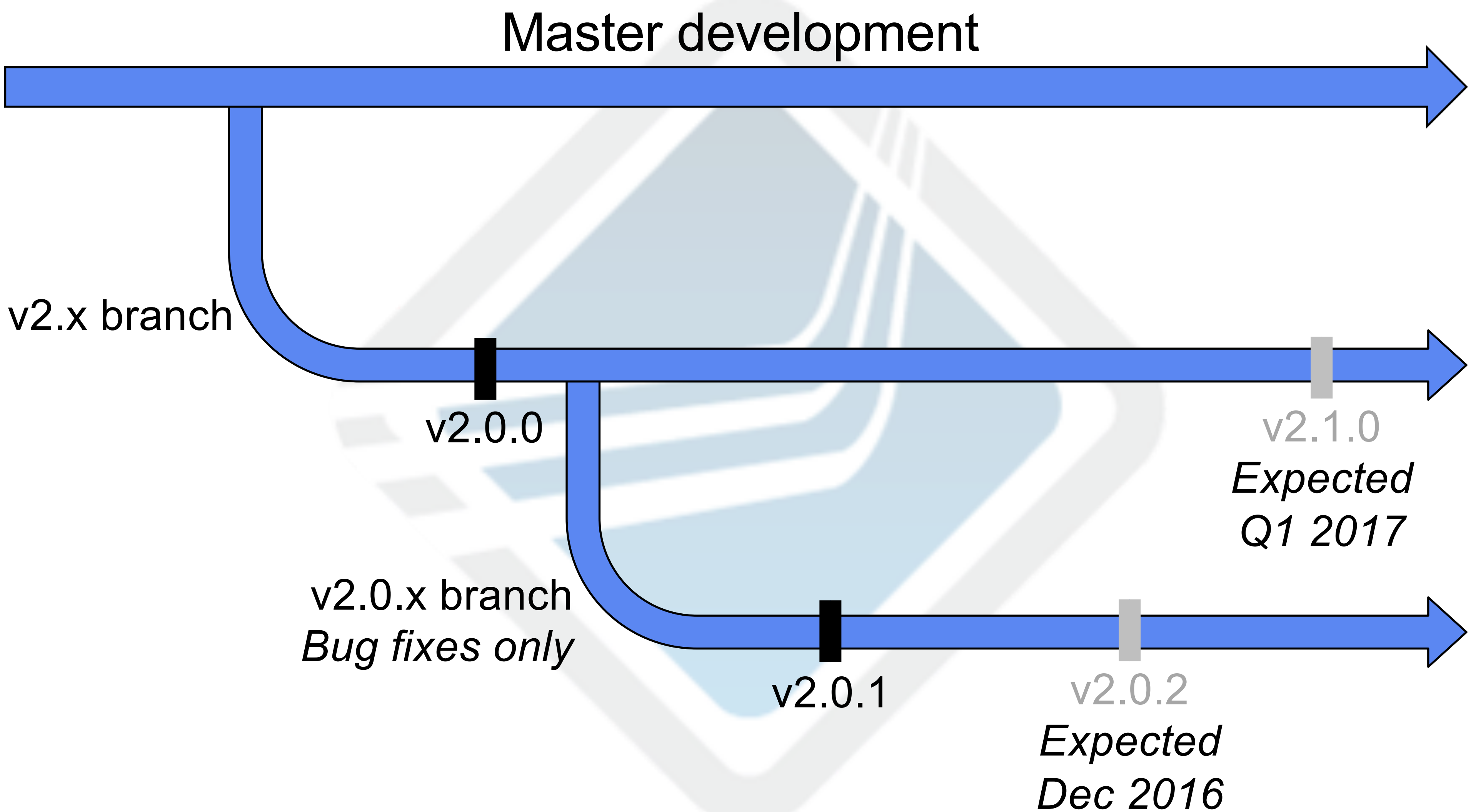
v2 Roadmap

v2.0.x and v2.x

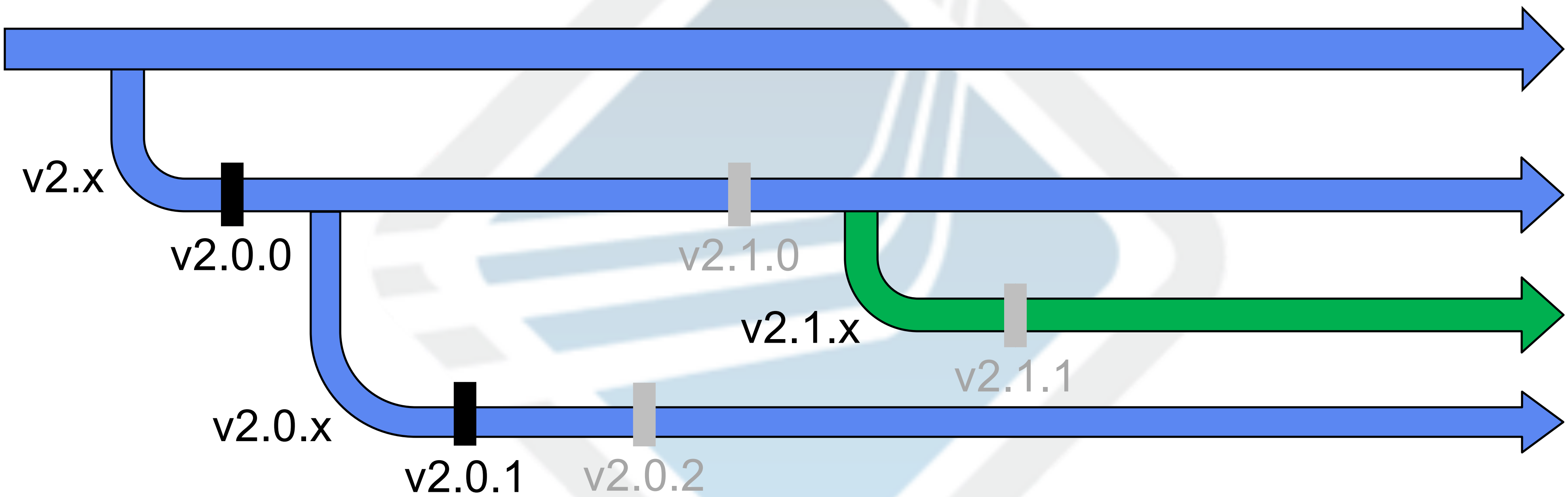
v2.x release managers

- Howard Pritchard, Los Alamos National Lab
- Jeff Squyres, Cisco Systems, Inc.

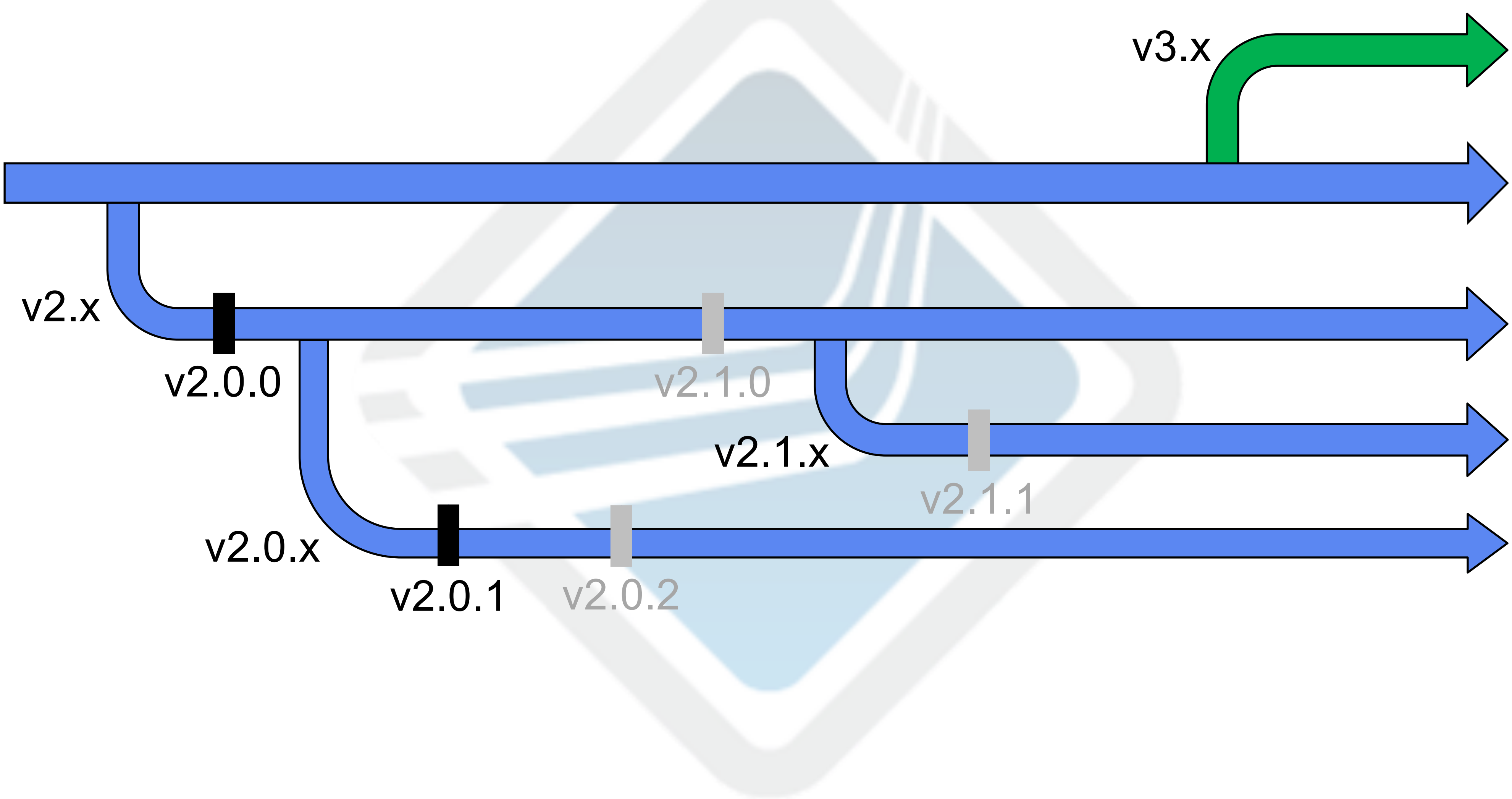
v2 versions


Master development
v2.x branch
v2.0.0
v2.1.0
Expected
Q1 2017
v2.0.x branch
Bug fixes only
v2.0.1
v2.0.2
Expected
Dec 2016

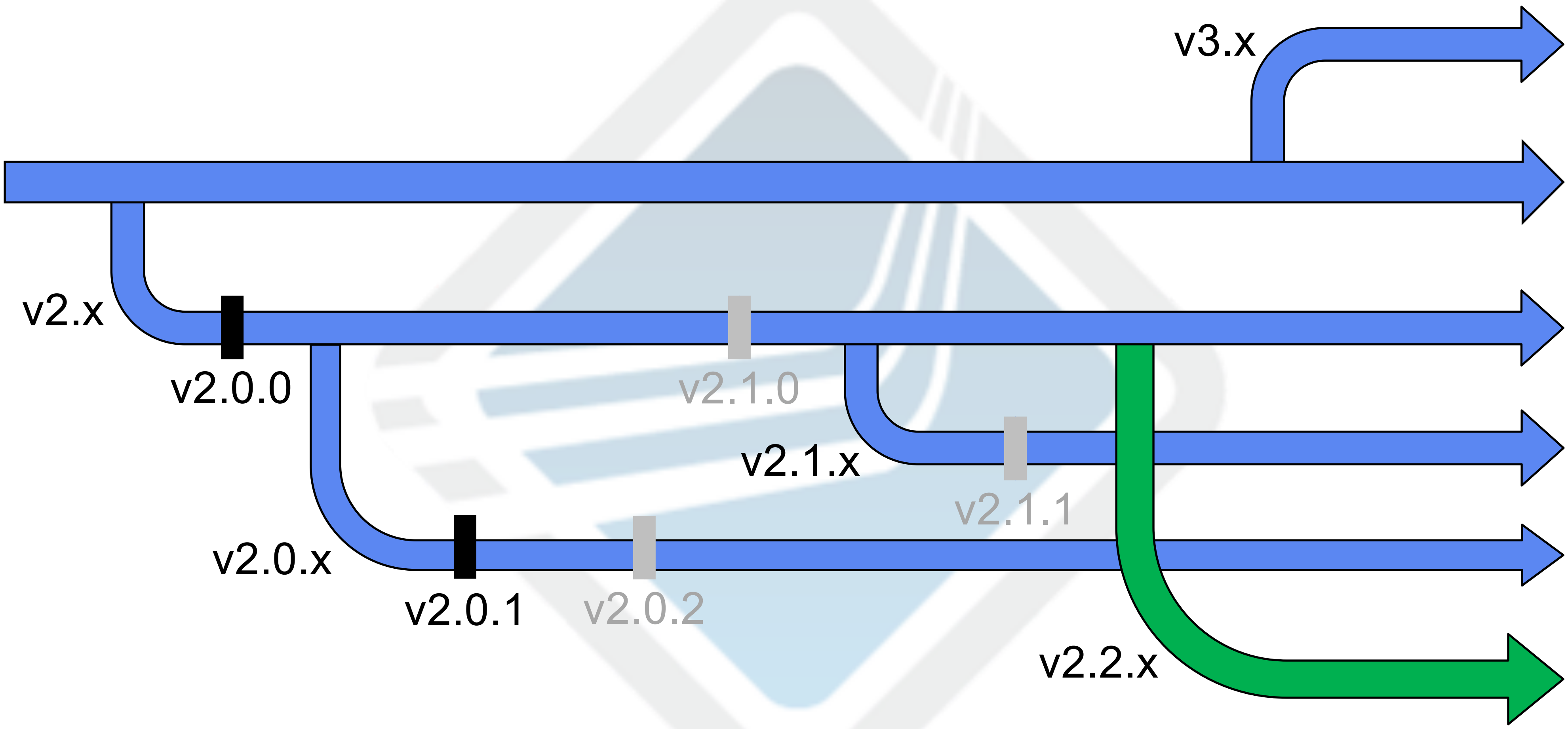
v2.1.x will get its own branch (bug fixes only)



v3.x will definitely (eventually) happen



Is it worthwhile to make an intermediate v2.2.x?



v2.2.x: pros and cons

PRO

- The v2.x branch is a good stable starting point
- Easy to maintain backwards compatibility with all v2.x series

CON

- Difficulty in porting new features from master branch
  - Due to drift from master
- Consumes developer resources and pushes back 3.x release
  - And therefore some ”bigger” features

This is an open question in the developer community

Should we do a v2.2.x series?

Please let us know your opinion!

www.open-mpi.org/sc16/

Random sample of v2.x features / work

Some lesser-known Open MPI features you may not be aware of

Singularity

- Containers are of growing interest
  - Packaged applications
  - Portability
- Some issues remain
  - Cross-container boundary interactions for MPI wireup, release manager interactions
  - Properly handling “containers” as “apps”

Leads: Ralph Castain (Intel), Greg Kurtzer (LBNL)

Open MPI Singularity Support

- PMIx support
  - Cross-version compatibility
  - Standardized protocol across environments
- Auto-detection of containers
  - Identify that app is a Singularity container
  - Do the Right Things to optimize behavior
- Auto-packaging of Open MPI apps
  - Singularity detects Open MPI app and automatically includes all required libs

ORTE Distributed Virtual Machine

- Original goal
  - Circumvent Cray's single job per node limit
  - Enable new programming model
- RADICAL-Pilot
  - Decompose large parallel problem in Bag of MPI Tasks
  - Decoupled, can be executed in parallel
  - Faster convergence to solution

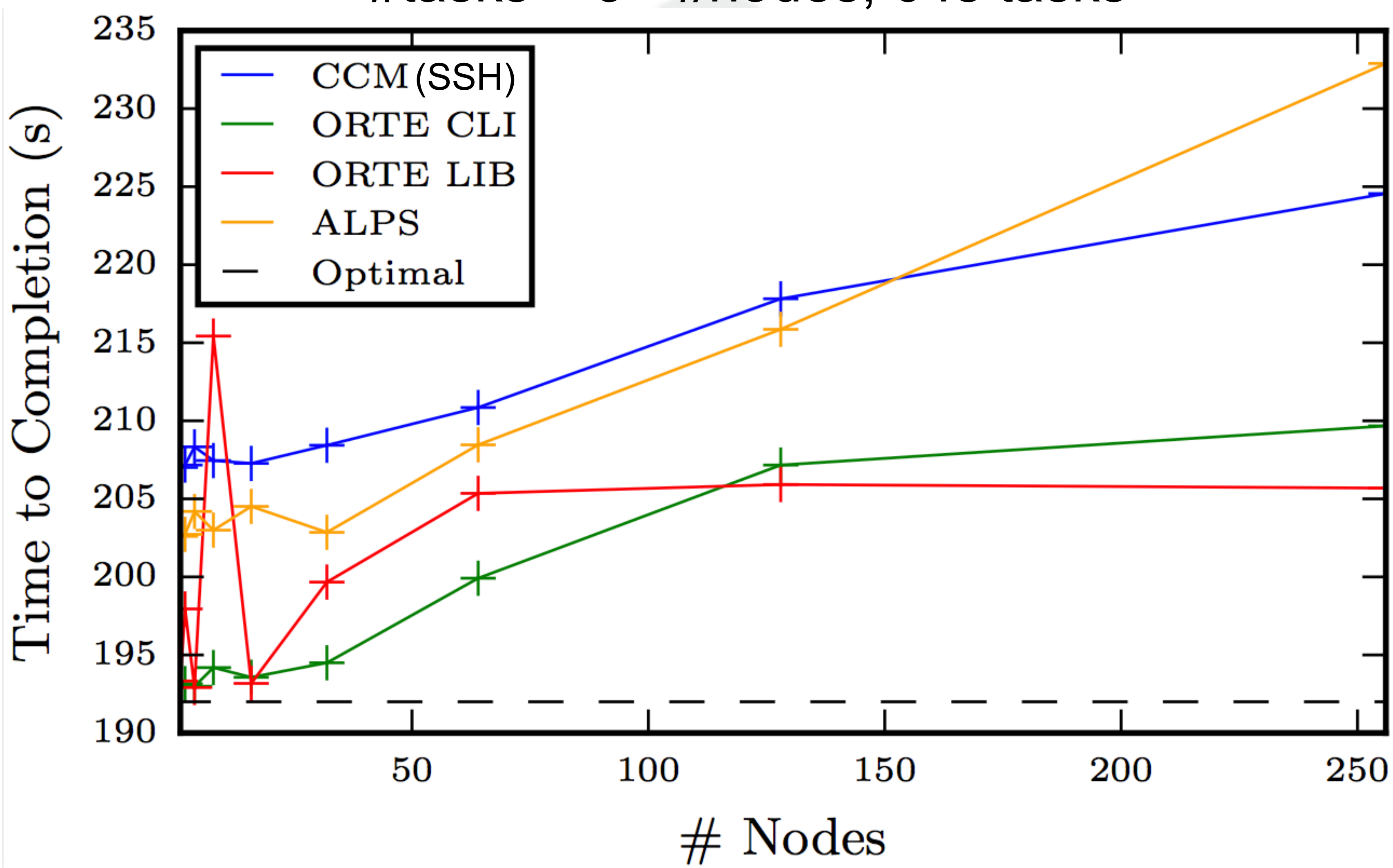
Leads: Mark Santcroos (Rutgers), Ralph Castain (Intel)

Role of DVM

- Launch/wireup time dominates execution
  - DVM instantiated once
  - Tasks highly asynchronous
  - Run tasks in parallel, share cpu cycles
- CLI interface
- Python language bindings through CFFI
- Result
  - Improved concurrency (~16k concurrent tasks)
  - Improved throughput (100 tasks/s)

ORTE-DVM + RADICAL-Pilot

#tasks = 3 * #nodes, 64s tasks



Future Work

- Bulk interface to `orte_submit()`
- OFI-based (libfabric) inter-ORTE daemon communication
- Optimize ORTE communication topology
- Topology aware task placement

Open MPI I/O ("OMPIO")

- Highly modular architecture for parallel I/O
  - Separate implementation than ROMIO
- Default parallel I/O library in Open MPI
  - For all file systems starting from the v2.0.-release with the exception of Lustre

Lead: Edgar Gabriel (U. Houston)

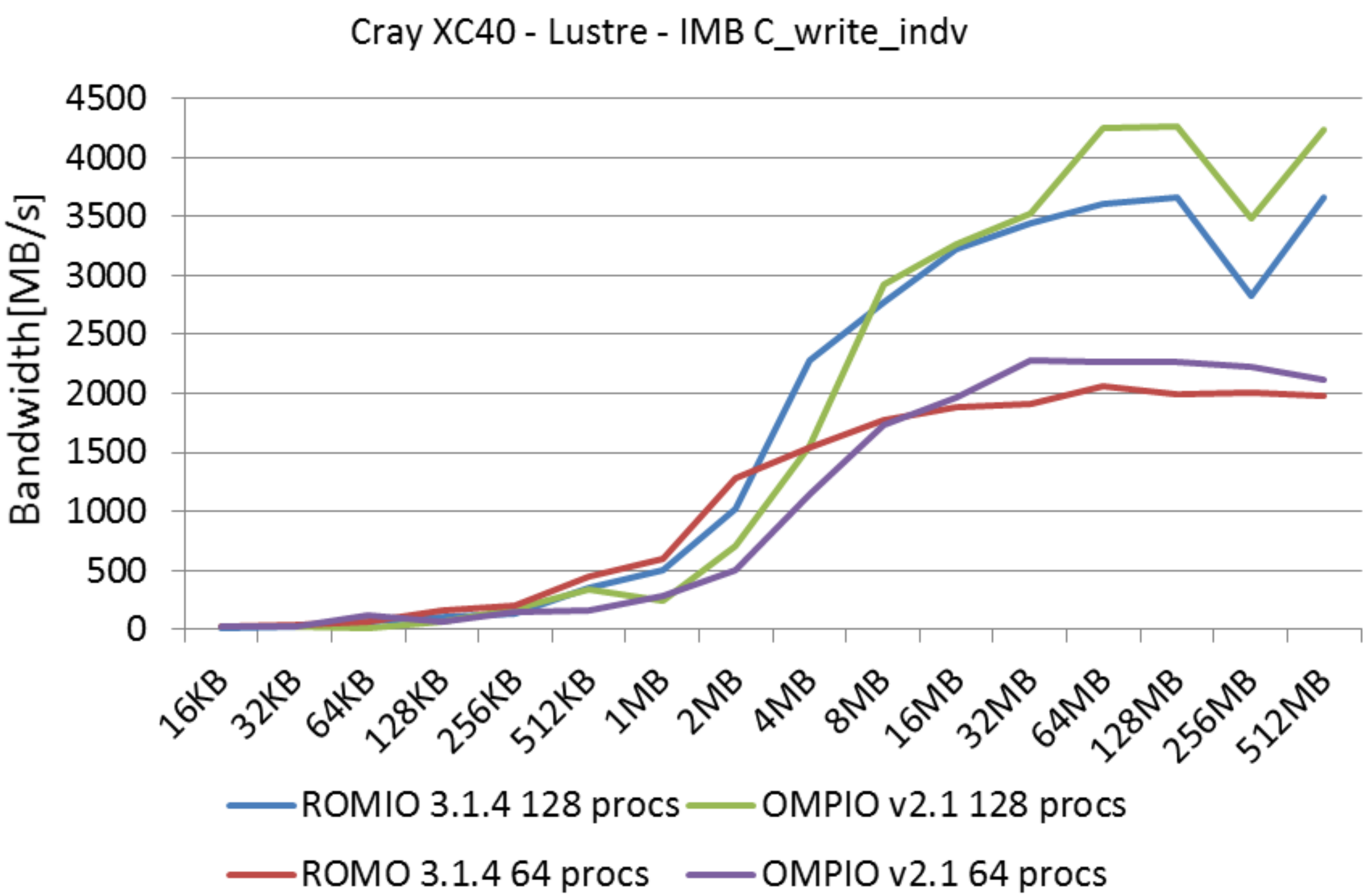
OMPIO key features

- Tightly integrated with the Open MPI architecture
  - Frameworks/modules, derived datatype handling, progress engine, etc.
- Support for multiple collective I/O algorithms
- Automatic adjustments of # of aggregators
- Multiple mechanisms available for shared file pointer operations

OMPIO ongoing work

- Enhance support for the Lustre file system in the v2.1.x release series
- Support for hybrid / multi-threaded applications
- Enhance support for GPFS
  - Collaboration with HLRS
- Enhance out-of-the-box performance of OMPIO

OMPIO


Cray XC40 - Lustre - IMB C_write_indv
Bandwidth[MB/s]
4500
4000
3500
3000
2500
2000
1500
1000
500
0
16KB
32KB
64KB
128KB
256KB
512KB
1MB
2MB
4MB
8MB
16MB
32MB
64MB
128MB
256MB
512MB
ROMIO 3.1.4 128 procs
OMPIO v2.1 128 procs
ROMO 3.1.4 64 procs
OMPIO v2.1 64 procs

AWS scale testing

- EC2 donation for Open MPI and PMIx scalability
  - Access to larger resources than individual organizations have
- Both science and engineering
  - Data-driven analysis
  - Used for regression testing
- Early days: no results to share yet

Leads: Jeff Squyres (Cisco), Peter Gottesman, Brian Barrett (AWS)

Exciting new capabilities in Open MPI

George Bosilca

ICL UT
INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

Exascale Computing Project and Open MPI

- DOE program to help develop software stack to enable application development for a wide range of exascale class systems
- Open MPI for Exascale (OMPI-X) was one of 35 proposals selected for funding:
  - Joint proposal involving ORNL, LANL, SNL, UTK, and LLNL
  - 3 year time frame

ECP
EXASCALE COMPUTING PROJECT

Exascale Computing Project and Open MPI Goals

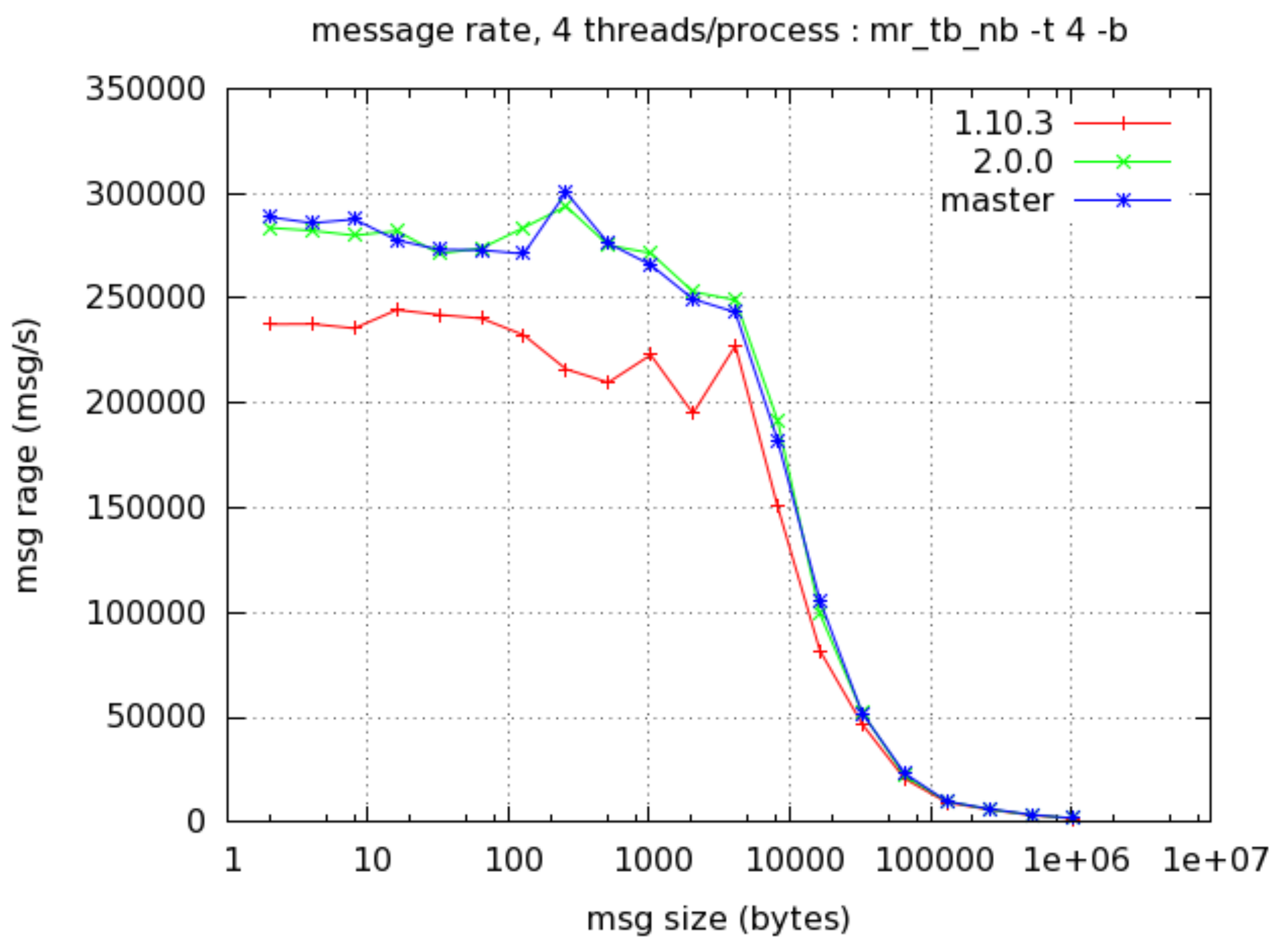
- Work with MPI Forum on extending the MPI standard to better support exascale applications
  - Endpoints, Finepoints, Sessions, Resilience
- Improved interoperability of OMPI with other programming models (MPI+X)
  - Process placement, thread marshaling, resource sharing
- Scalability and Performance
  - Memory footprint, Collective communications, Message Matching, PMIx
- Resilience/fault tolerance improvements
  - Integration with C/R libraries (FTI, SCR), in-place restart, ULFM
- Enhancing MPI Tools interface (network performance counters, better RMA support, etc.)

MPI_THREAD_MULTIPLE

- The transition to full threading support has been mostly completed
  - We also worked on the performance (fine grain locks)
  - Supports all threading models from a single library
    - All atomic accesses are protected
  - Allow asynchronous progress
- Complete redesign the request completion
  - Everything goes through requests (pt2pt, collectives, RMA*, I/O, *)
  - Threads are not competing for resources, instead they collaboratively progress
    - A thread will wait until all expected requests have been completed or an error has been raised
    - Less synchronizations, less overhead (better latency and injection rate).

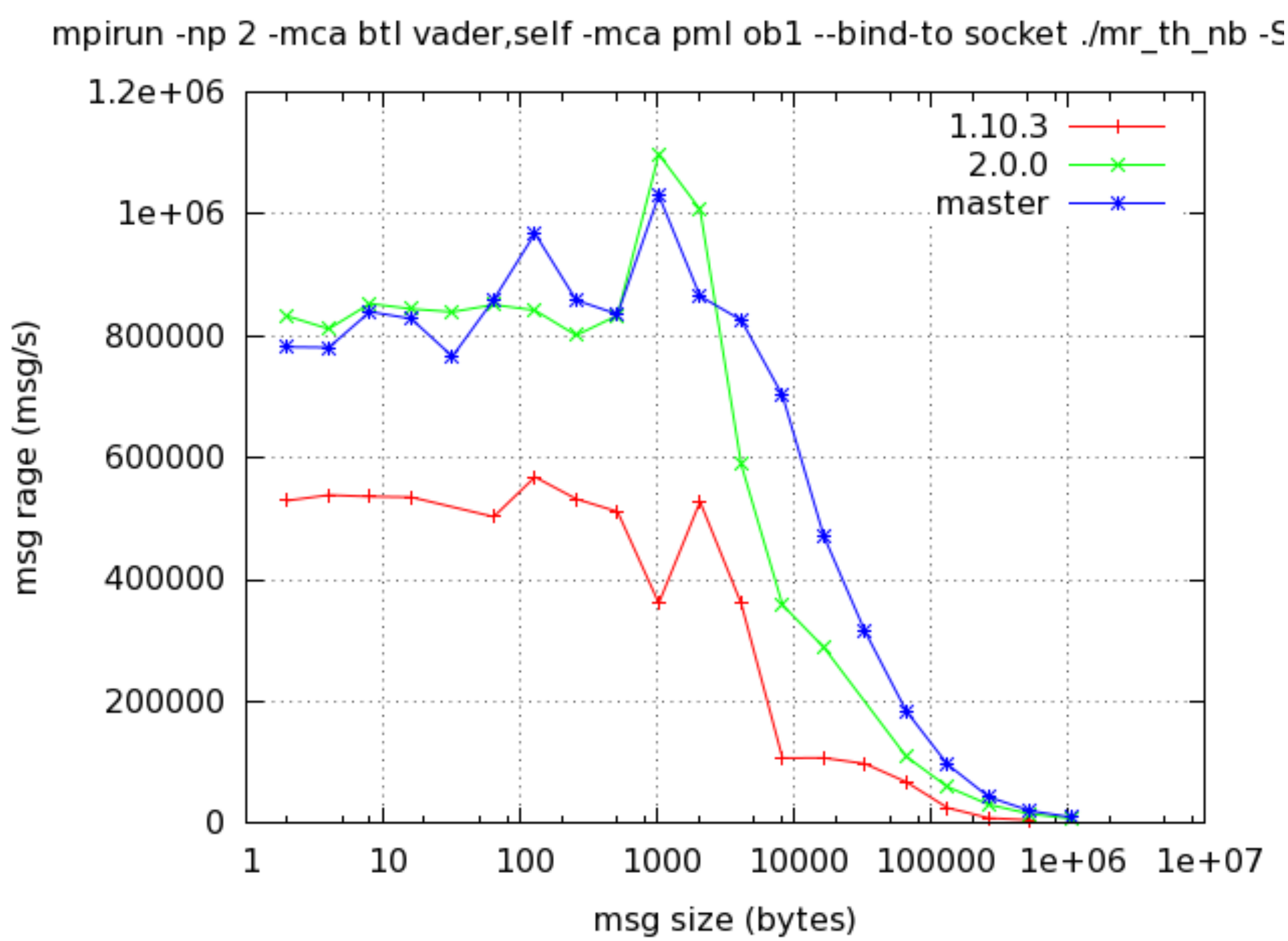
MPI_THREAD_MULTIPLE

- Messages per second injection rate (bigger is better)
- Each process bound to a NUMA node
  - 4 threads per process
  - Distributed environment TCP (ipoib)



MPI_THREAD_MULTIPLE

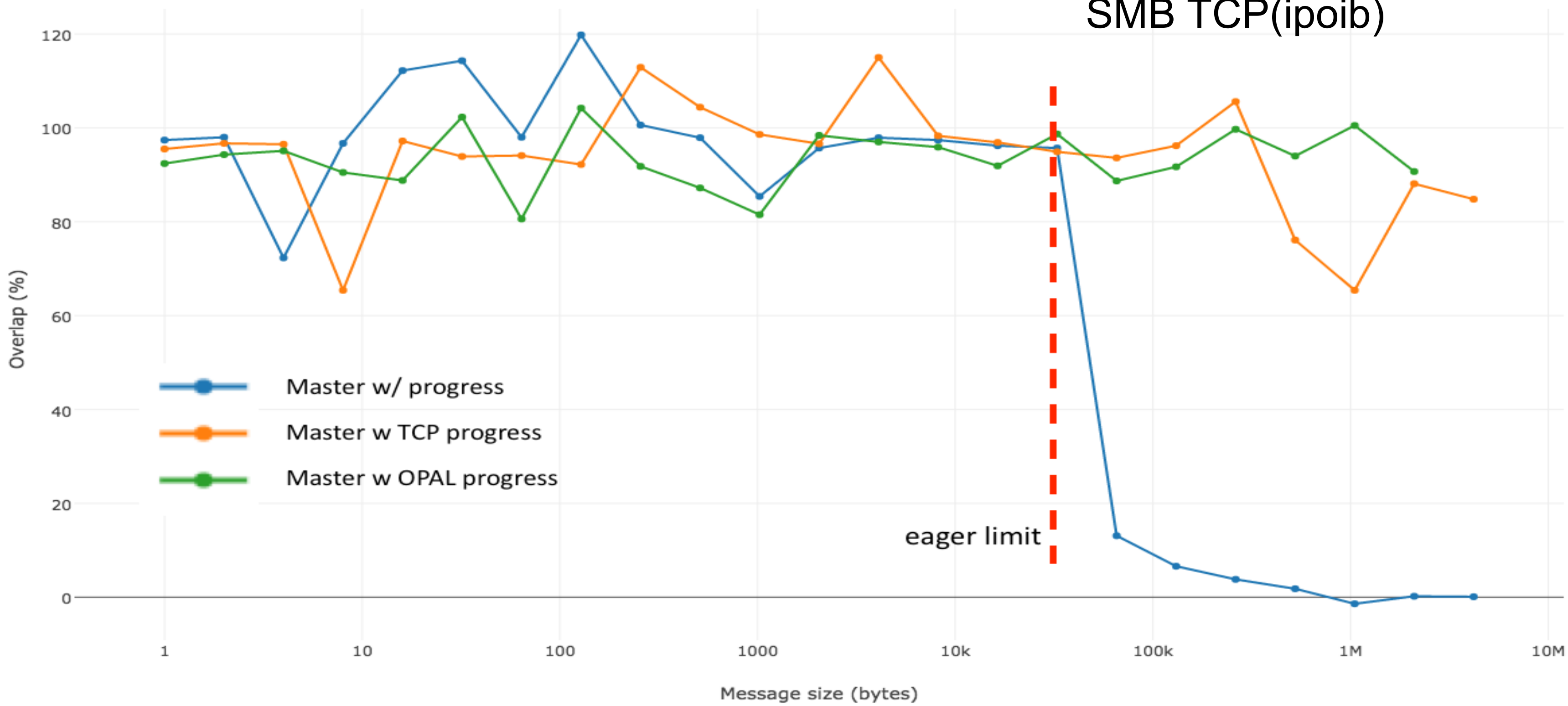
- Messages per second injection rate (bigger is better)
- Each process bound to a NUMA node
  - 4 threads per process
  - ~~Distributed environment TCP (ipoib)~~
  - Vader (shared memory)
- All BTLs show similar results



Asynchronous progress

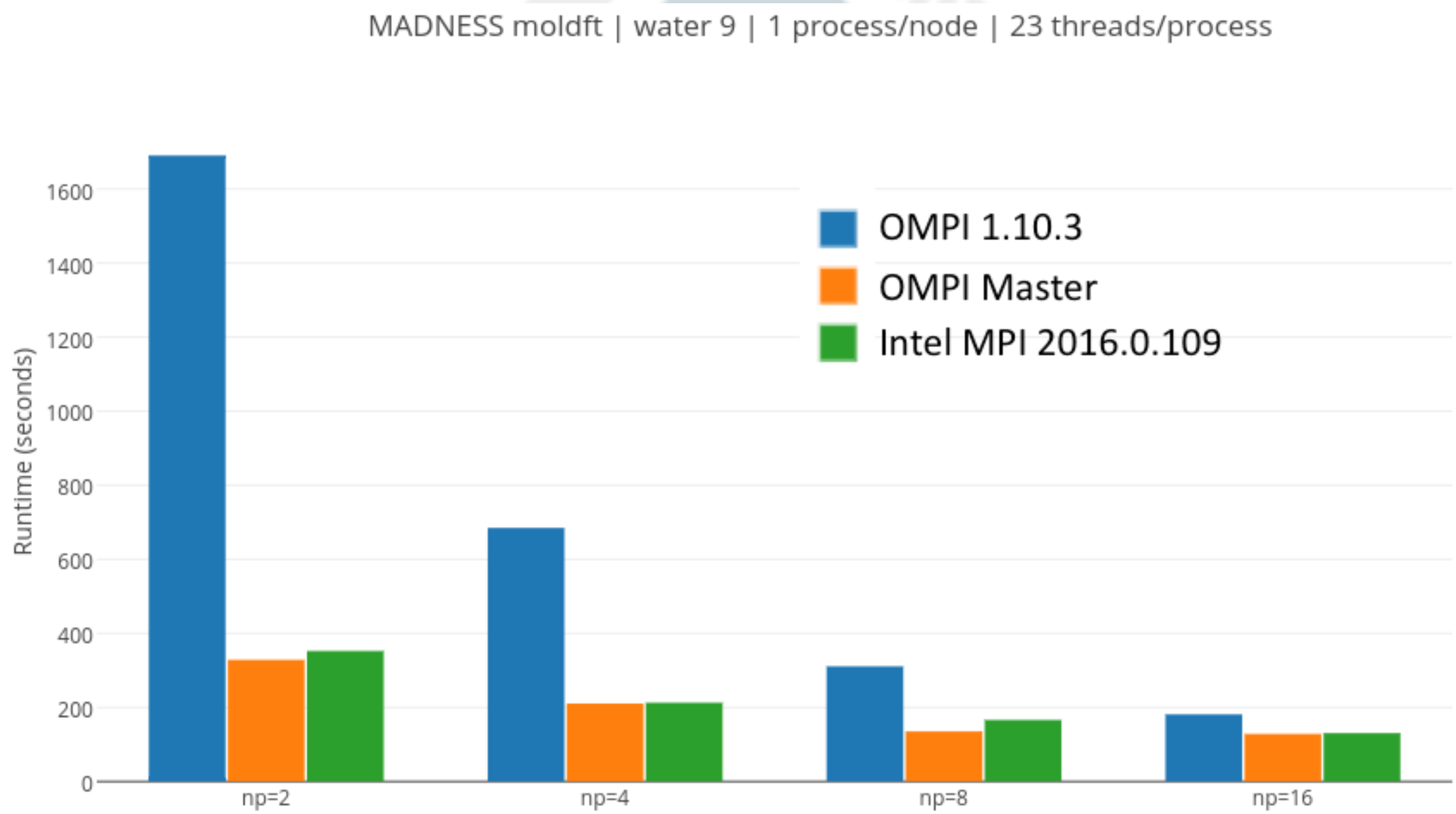
- The BTLs can either have their own progress thread (such as TCP and usnic) or take advantage of the async progress provided by OPAL
  - Adaptive Frequency: varies with the expected load (posted or ongoing data movements)

SMB TCP(ipoib)



Impact on applications

- MADNESS - Multiresolution Adaptive Numerical Environment for Scientific Simulation



Network support

It's complicated!

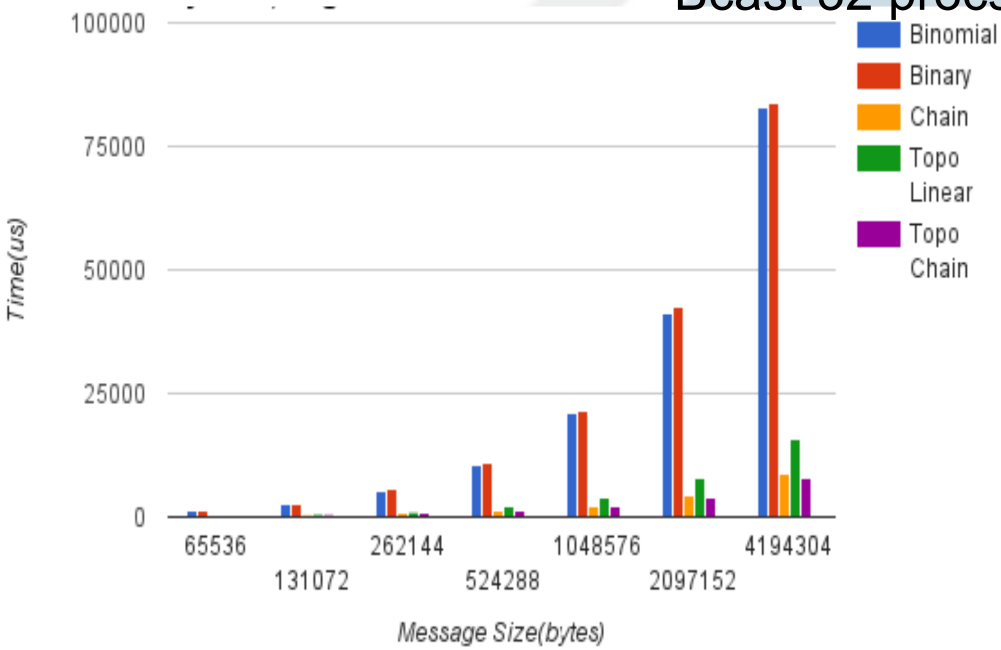
(only listing those that are in release branches)

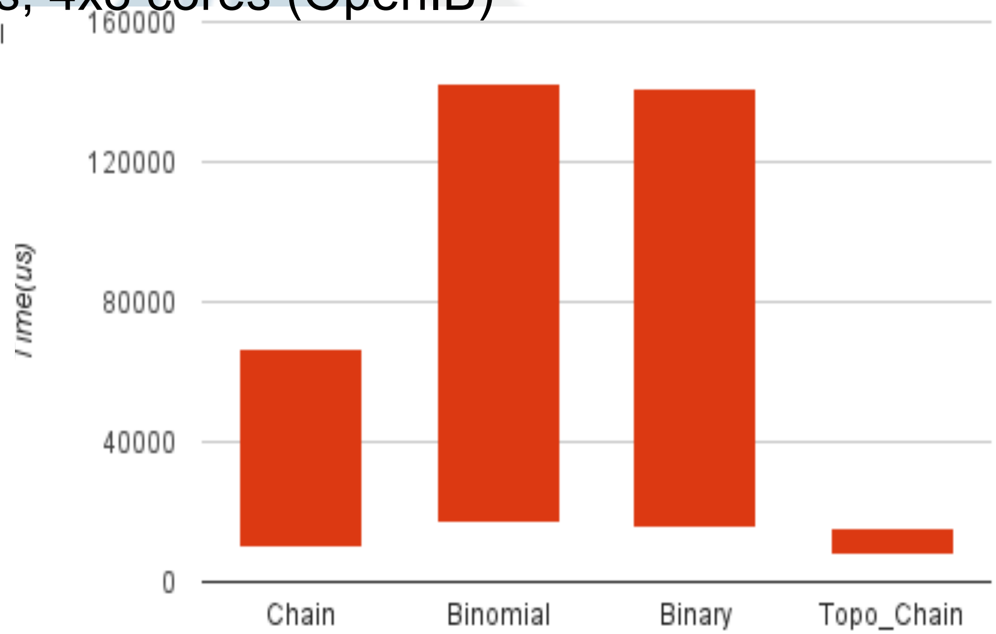
	Name	Owner	Status	Thread
BTL	OpenIB	Chelsio	maintenance	Done
	Portals4	SNL	maintenance	Not done
	Scif	LANL	maintenance	Not done
	self	UTK	active	Done
	sm	UTK	active	Done
	smcuda	NVIDIA/UTK	active	Done
	tcp	UTK	active	Done
	uGNI	LANL	active	Done
	usnic	CISCO	active	Done
	vader	LANL	active	Done
PML	Yalla	Mellanox	active	In progress
	UCX	Mellanox/UTK	active	Done
MTL	MXM	Mellanox	active	In progress
	OFI	Intel	active	In progress
	Portals4	SNL	active	In progress
	PSM	Intel	active	In progress
	PSM2	Intel	active	In progress

Collective Communications

- Architecture aware: Reshape tuned to account for process placement and node architecture
- Classical 2 levels decision (inter and intra-node) composition of collective algorithms
  - Pipeline possible but no other algorithmic composition possible

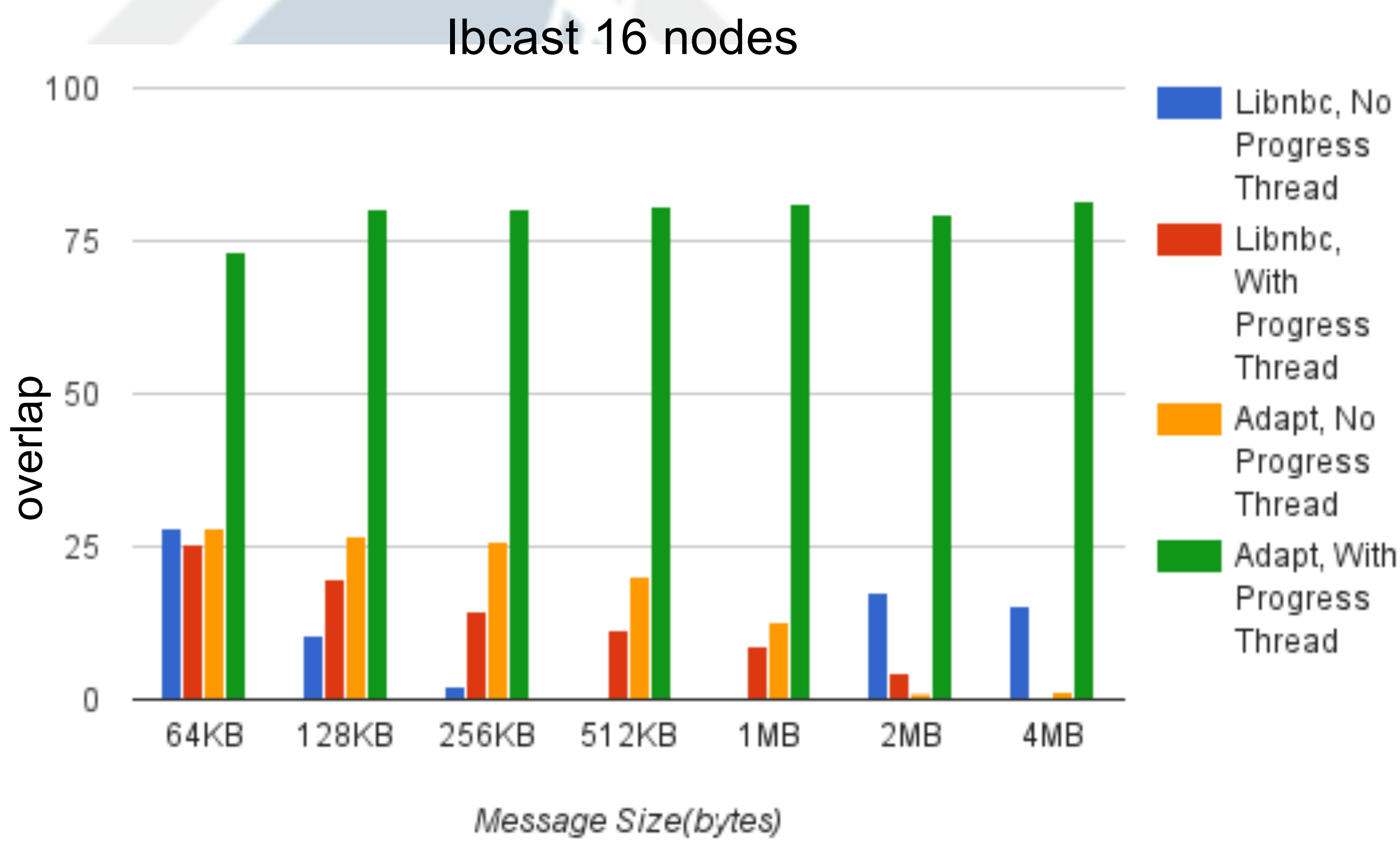
Bcast 32 procs, 4x8 cores (OpenIB)





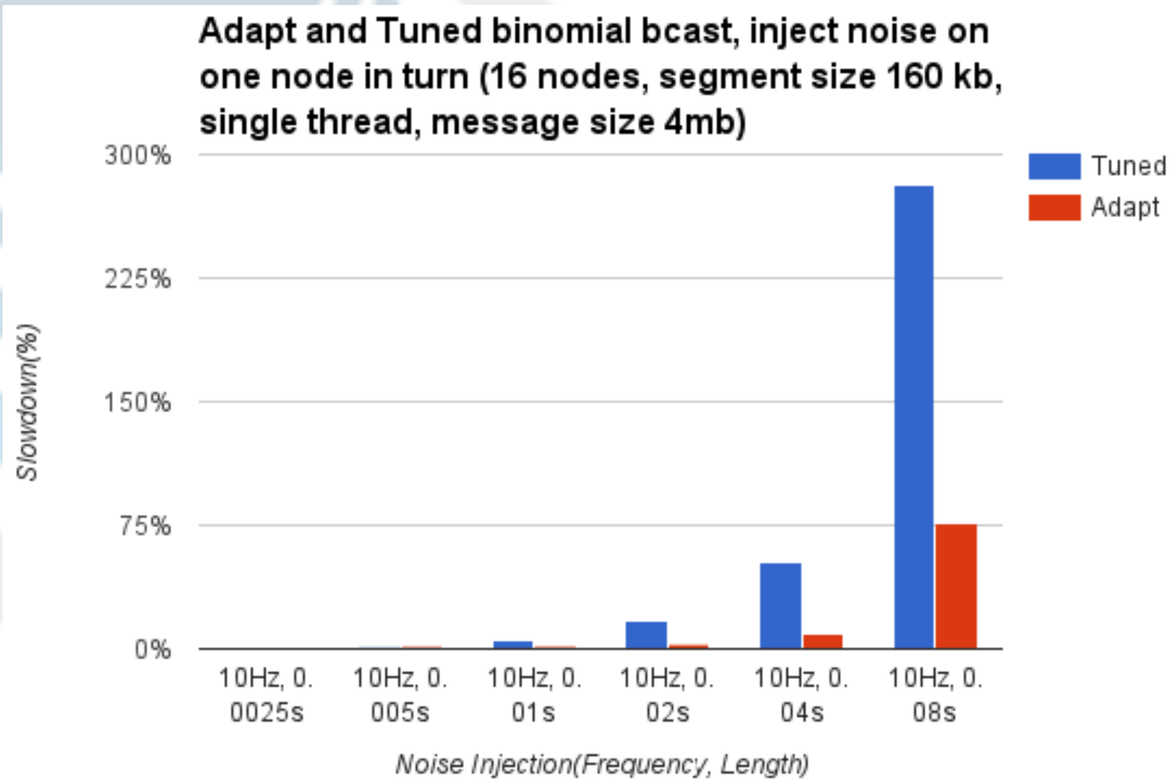
Collective Communications

- Dataflow Collectives: Different algorithms compose naturally (using a dynamic granularity for the pipelining fragments)
  - Async Collectives starts as soon as the process learns that a collective is progressing on a communicator (somewhat similar to unexpected collectives)
  - The algorithm automatically adapts to network conditions
  - Resistant to system noise

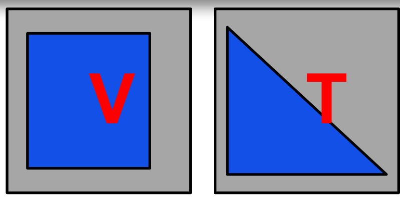


Collective Communications

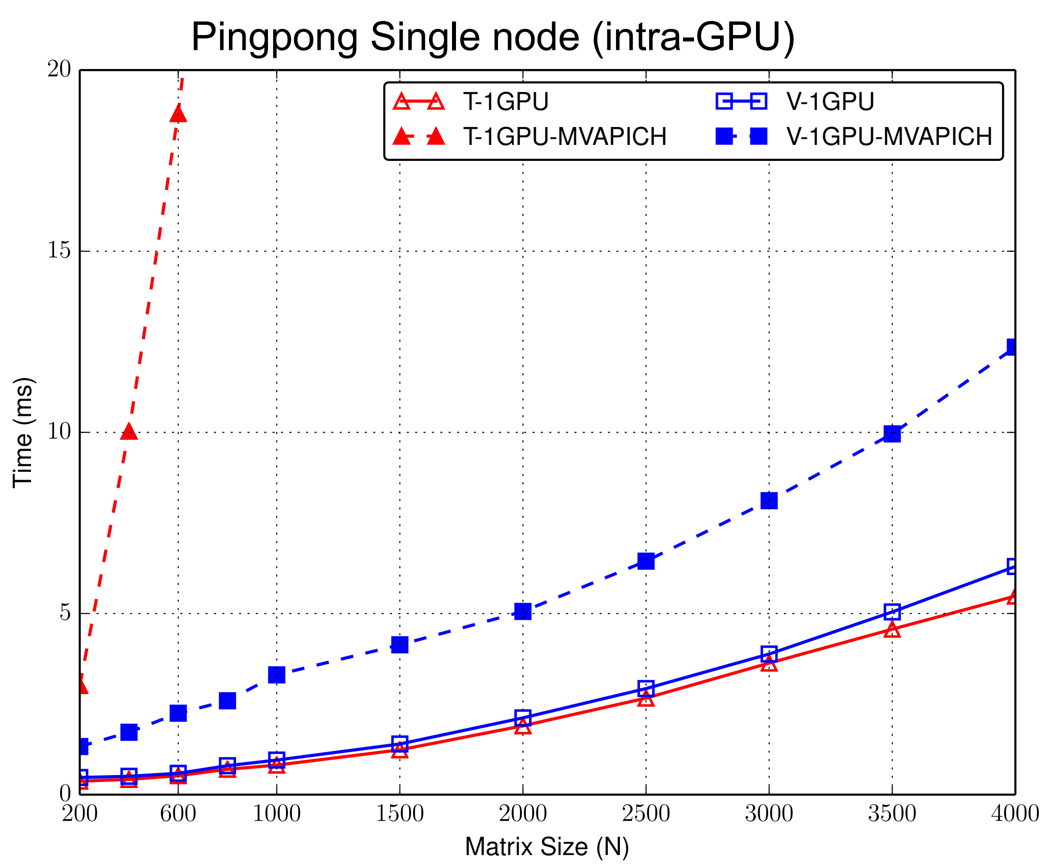
- Dataflow Collectives: Different algorithms compose naturally (using a dynamic granularity for the pipelining fragments)
  - The algorithm automatically adapts to network conditions
  - Resistant to system noise



CUDA Support

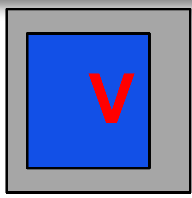


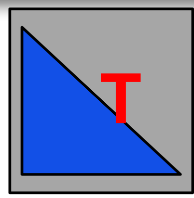
Ivy Bridge E5-2690 v2 @ 3.00GHz, 2 sockets 10-core, 4 K40/node
MVAPICH 2.2-GDR



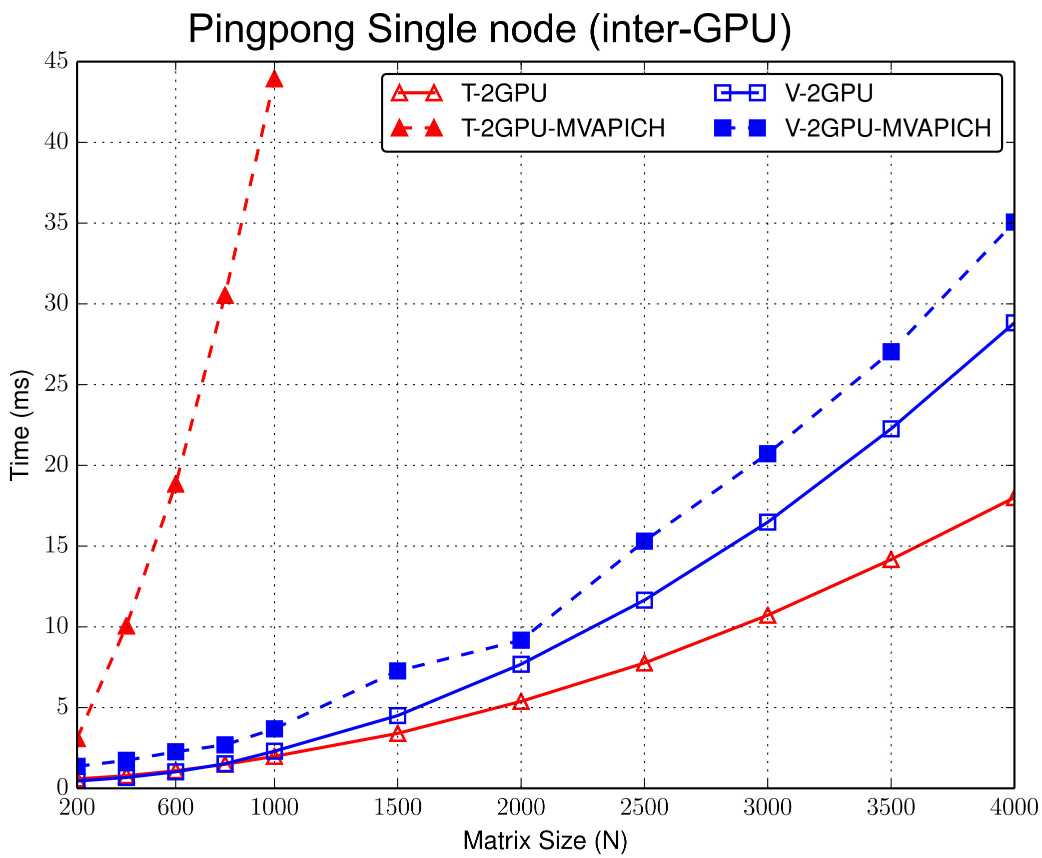
- Multi-level coordination protocol based on the location of the source and destination memory
  - Support for GPUDirect
- Delocalize part of the datatype engine into the GPU
  - Driven by the CPU
  - Provide a different datatype representation (avoid branching in the code)
- Deeply integrated support for OpenIB and shared memory
  - BTL independent support available for everything else

CUDA Support



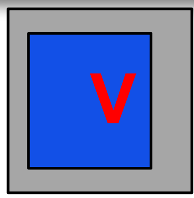


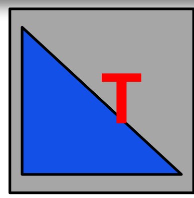
Ivy Bridge E5-2690 v2 @ 3.00GHz, 2 sockets 10-core, 4 K40/node
MVAPICH 2.2-GDR



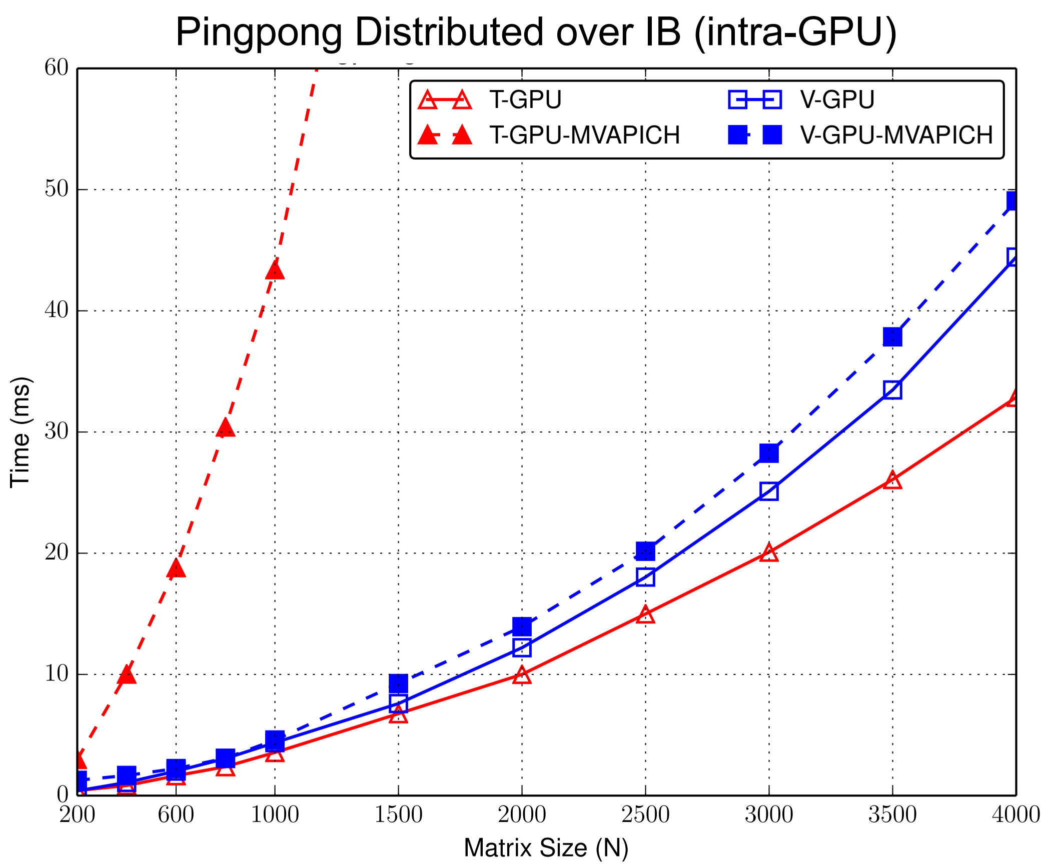
- Multi-level coordination protocol based on the location of the source and destination memory
  - Support for GPUDirect
- Delocalize part of the datatype engine into the GPU
  - Driven by the CPU
  - Provide a different datatype representation (avoid branching in the code)
- Deeply integrated support for OpenIB and shared memory
  - BTL independent support available for everything else

CUDA Support





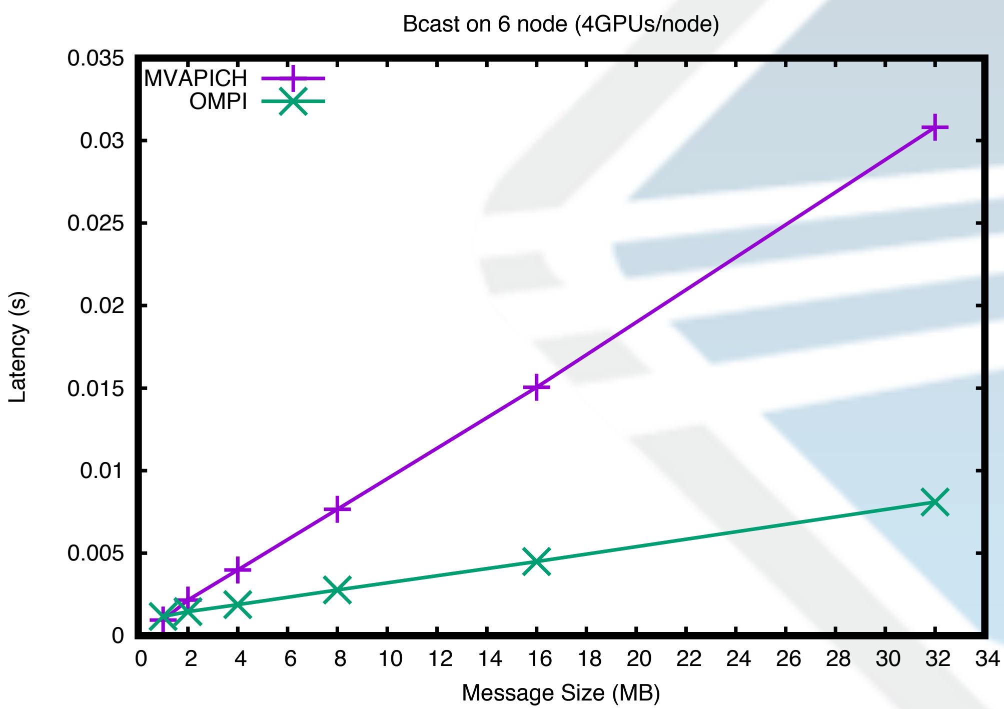
Ivy Bridge E5-2690 v2 @ 3.00GHz, 2 sockets 10-core, 4 K40/node
MVAPICH 2.2-GDR

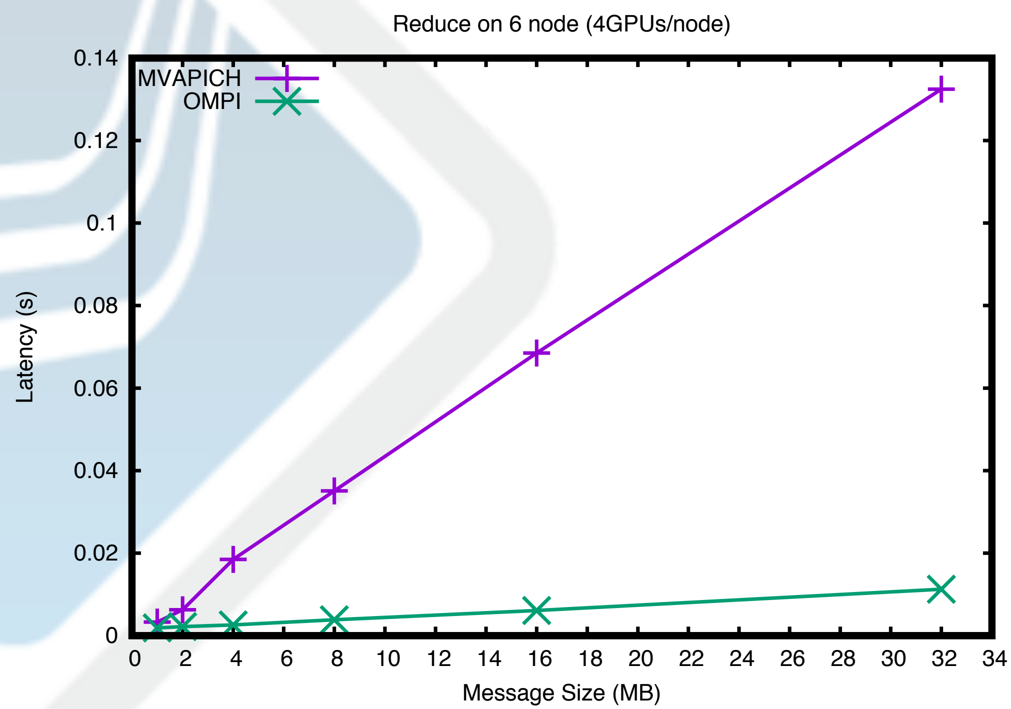


- Multi-level coordination protocol based on the location of the source and destination memory
  - Support for GPUDirect
- Delocalize part of the datatype engine into the GPU
  - Driven by the CPU
  - Provide a different datatype representation (avoid branching in the code)
- Deeply integrated support for OpenIB and shared memory
  - BTL independent support available for everything else

CUDA Support

- Architecture-aware collective support
  - Dataflow algorithm
  - Node-level algorithm take advantage of the bidirectional bandwidth of the PCI

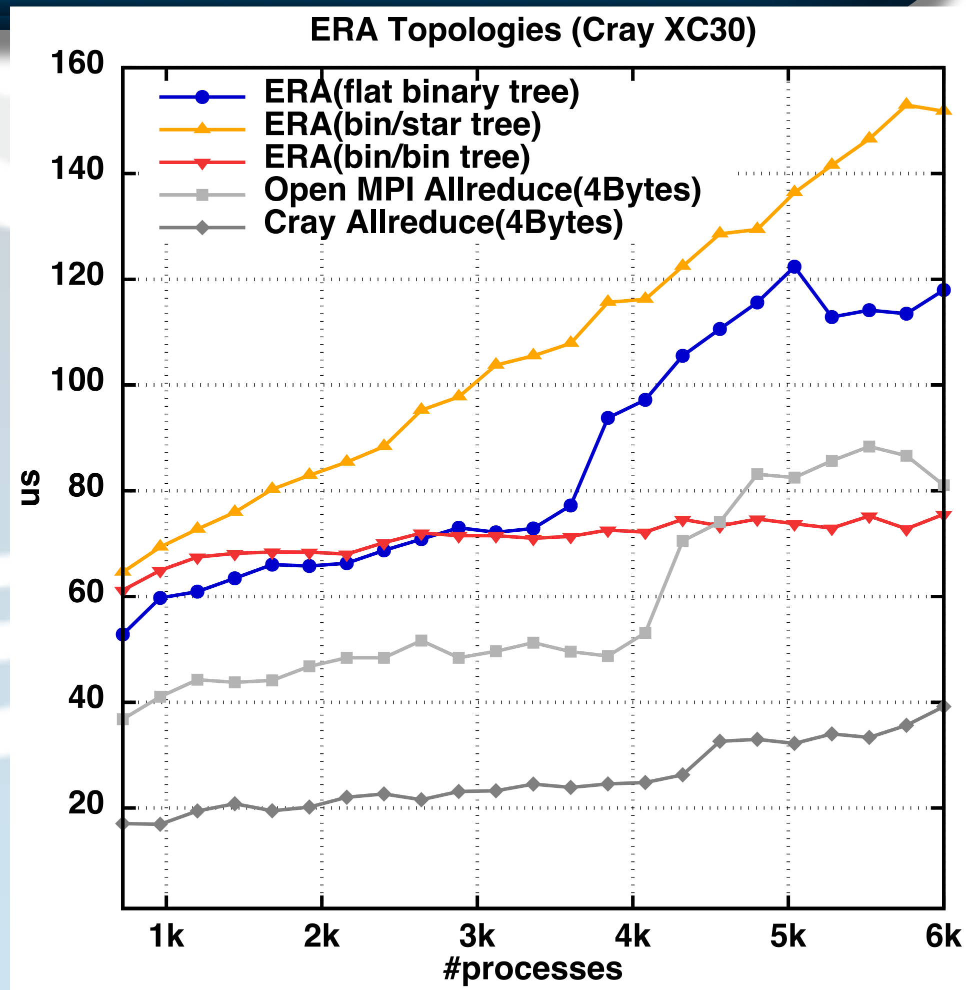


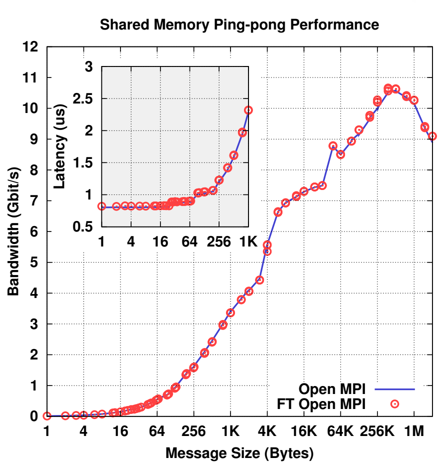


Ivy Bridge E5-2690 v2 @ 3.00GHz, 2 sockets 10-core, 4 K40/node
MVAPICH 2.2-GDR

User Level Failure Mitigation

- Open MPI implementation updated in-sync with Open MPI 2.x
- Scalable fault tolerant algorithms demonstrated in practice for revoke, agreement, and failure detection (SC'14, EuroMPI'15, SC'15, SC'16)





Point to point performance unchanged With FT enabled

Fault Tolerant Agreement costs approximately 2x Allreduce

Open MPI and Fujitsu

Fujitsu Limited

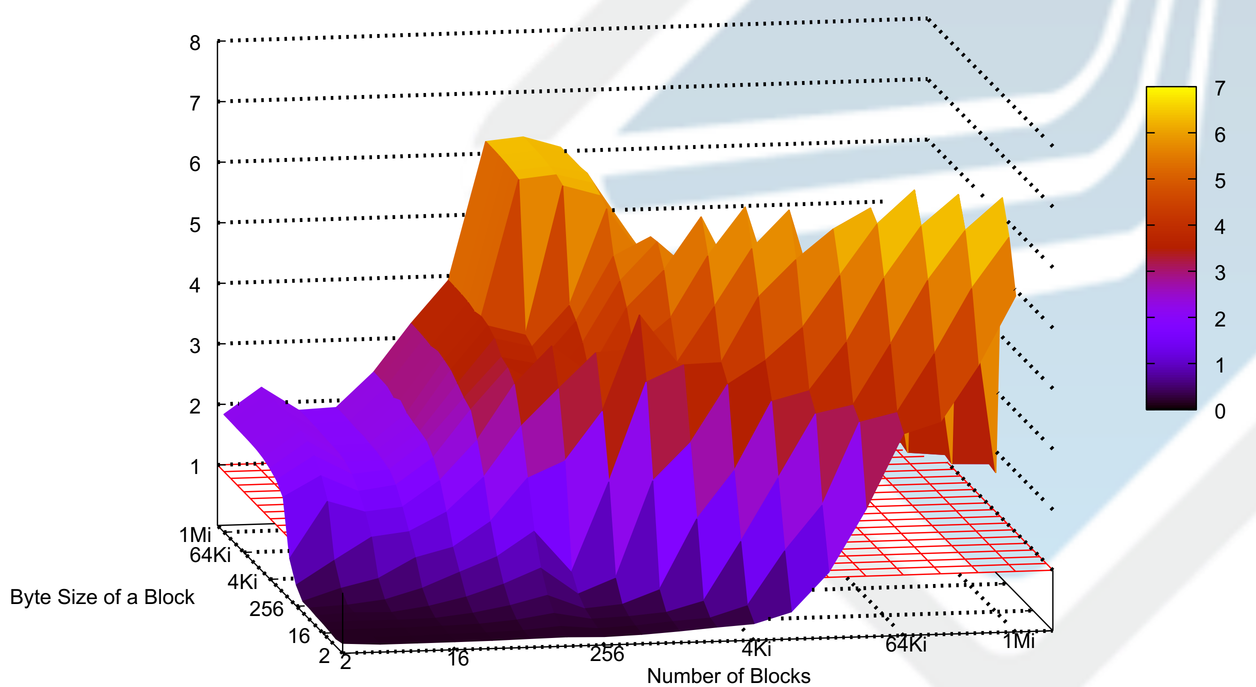
Fujitsu MPI with Open MPI Community

- Fujitsu MPI is based on Open MPI
  - Running on K computer and its commercial/ successor machines for over 5 years.
  - For Post-K, also Open MPI based.
- Collaboration plan with OMPI community
  - ARM support
  - PMIx integration
  - Reduced memory footprint
  - New MPI version support
  - Thread parallelization extension etc.

Fujitsu Contribution Examples

- Thread parallelization in the MPI library
- Statistical information for application tuning

Speed-up Ratio of MPI_PACK Compared to Single Thread




```
==================================================================
/***************** MPI Statistical Information *****************/
==================================================================

------------------------- MPI Information ------------------------
Dimension                      3
Shape                      2x3x4

------------------ Per-peer Communication Count ------------------
                             MAX              MIN              AVE
In_Node                  1024 [    0]        0 [    1]      512.0
Neighbor                 3072 [    1]        0 [    8]     1621.3
Not_Neighbor             3072 [   11]        0 [    0]      938.7
Total_Count              3072 [    0]     3072 [    0]     3072.0
Connection                 46 [    0]        9 [    4]       11.8
Max_Hop                     4 [    0]        2 [    4]        3.1
Average_Hop              2.27 [   35]     1.60 [    6]       1.84

---------------- Per-peer Transmission Size (MiB) ----------------
                             MAX              MIN              AVE
In_Node                256.00 [    0]     0.00 [    1]     128.00
Neighbor               768.00 [    1]     0.00 [    8]     405.33
Not_Neighbor           768.00 [   11]     0.00 [    0]     234.67
Total_Size             768.00 [    0]   768.00 [    0]     768.00

---------------- Per-protocol Communication Count ----------------
                             MAX              MIN              AVE
Eager                       0 [    0]        0 [    0]        0.0
Rendezvous               3072 [    0]     3072 [    0]     3072.0
Hasty_Rendezvous            0 [    0]        0 [    0]        0.0
Persistent_Extended_IF      0 [    0]        0 [    0]        0.0
Unexpected_Message          1 [    0]        1 [    0]        1.0

------------------ Barrier Communication Count -------------------
                             MAX              MIN              AVE
Tofu                     8217 [    0]     8217 [    0]     8217.0
Soft                        1 [    0]        1 [    0]        1.0
```

See https://github.com/open-mpi/ompi/wiki/Meeting-2016-02-attachments/Fujitsu-OMPI-Dev-Meeting-2016-Feb.pdf

RMA Update

Nathan Hjelm
Los Alamos National Laboratory

v2.x osc/pt2pt

- Fully supports MPI-3.1 RMA
- Full support for MPI datatypes
- Emulates one-sided operation using point-to-point components (PML) for communication
- Improved lock-all scaling
- Improved support for MPI_THREAD_MULTIPLE
- Caveat: asynchronous progress support lacking
  - Targets must enter MPI to progress any one-sided data movement!
  - Doesn’t really support passive-target

v2.x osc/rdma

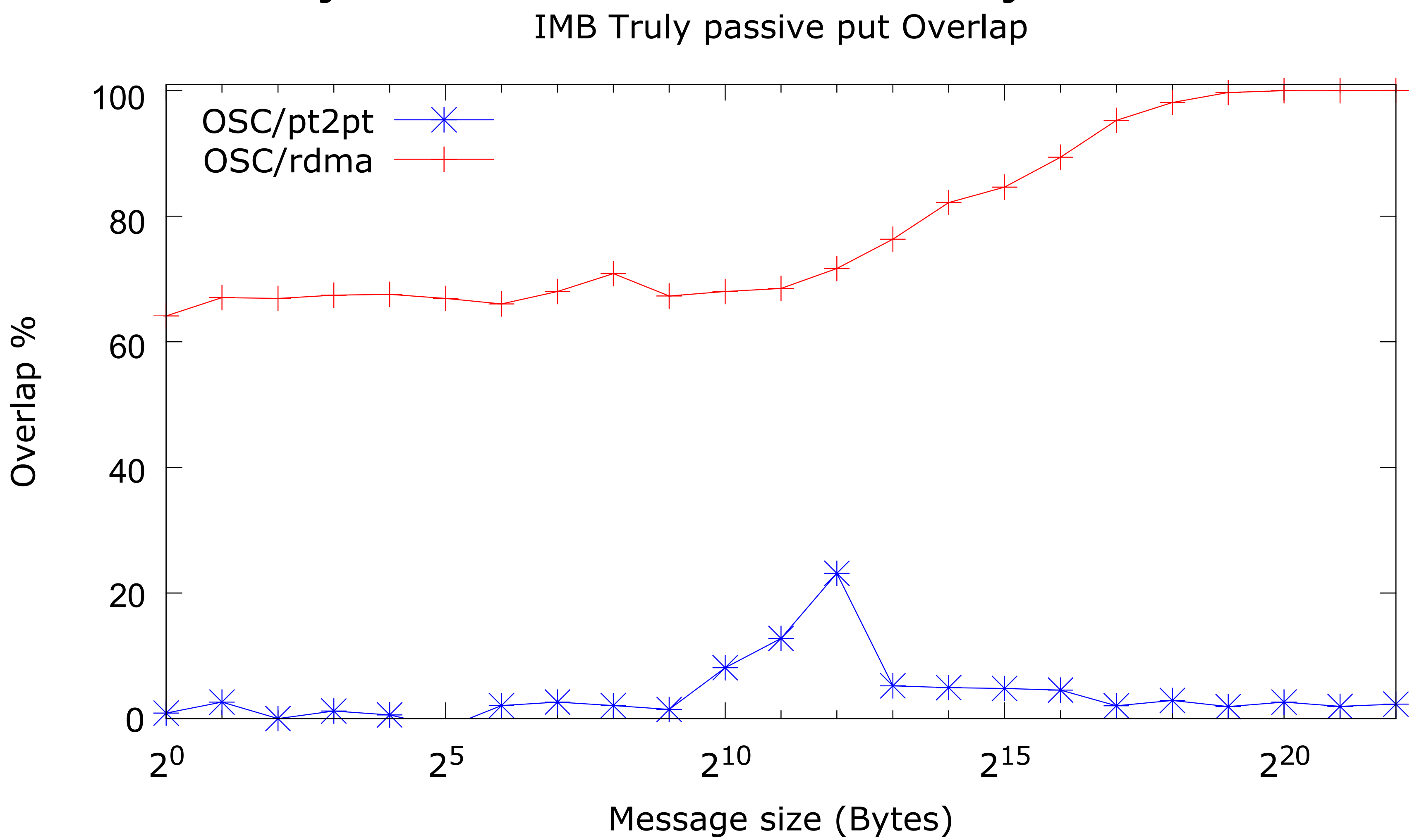
- Fully supports MPI-3.1 RMA
- Full support for MPI datatypes
- Fully supports passive target RMA operations
- Uses network RMA and atomic operation support through Byte Transport Layer (BTL)
- Supports Infiniband, Infinipath/Omnipath**, and Cray Gemini/Aries
- Additional networks can be supported
  - Requirements: put, get, fetch-and-add, and compare-and-swap

v2.x osc/rdma

- Improved support for MPI_THREAD_MULTIPLE
- Improved memory scaling
- Support for hardware atomics
  - Supports MPI_Fetch_and_add and MPI_Compare_and_swap
  - Supports 32 and 64 bit integer and floating point values
  - Accelerated MPI_Ops varies by hardware
  - Set osc_rdma_acc_single_intrinsic MCA variable to true to enable

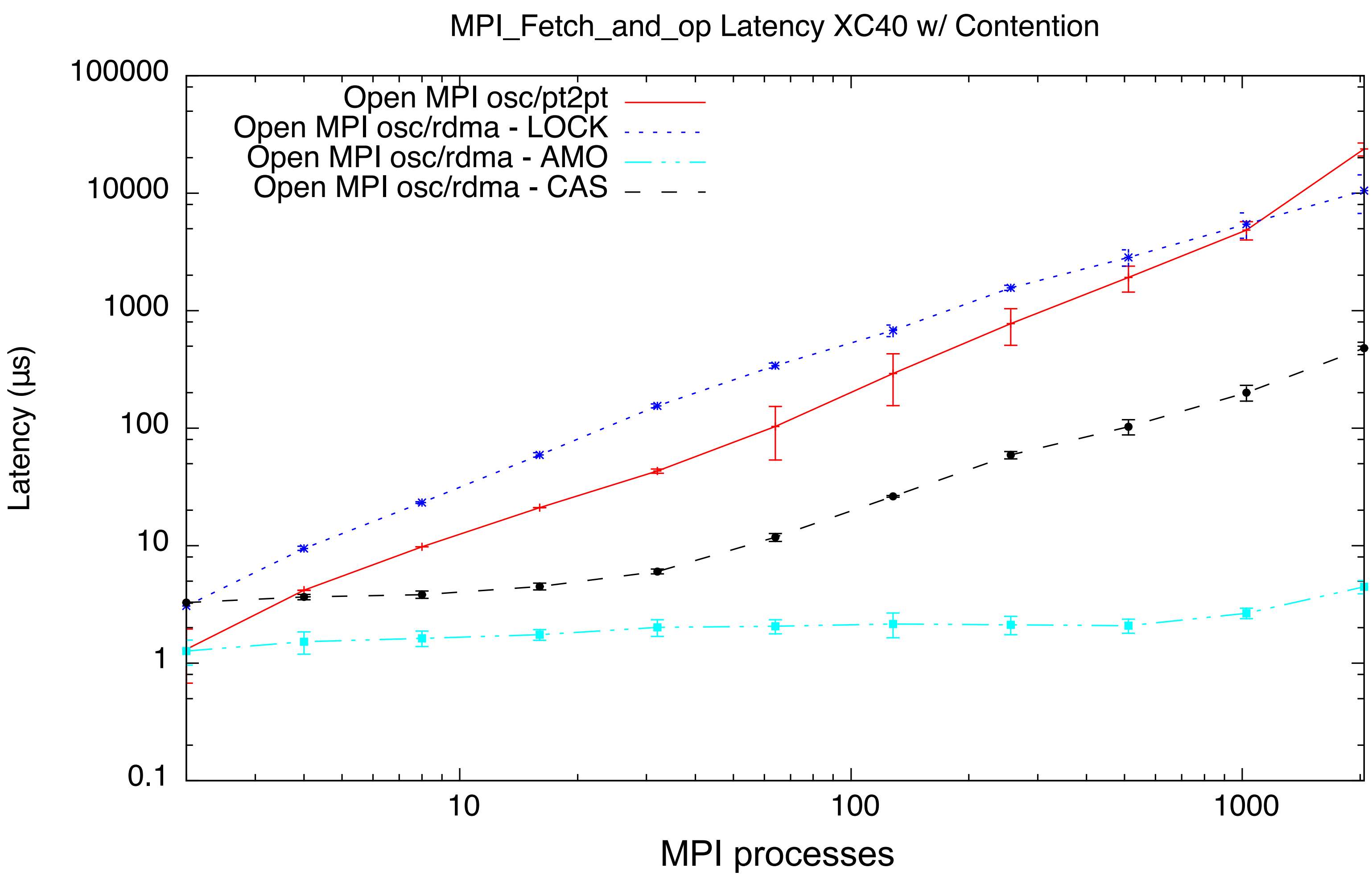
v2.x RMA Performance

- IMB Truly Passive Put on Cray XC-40



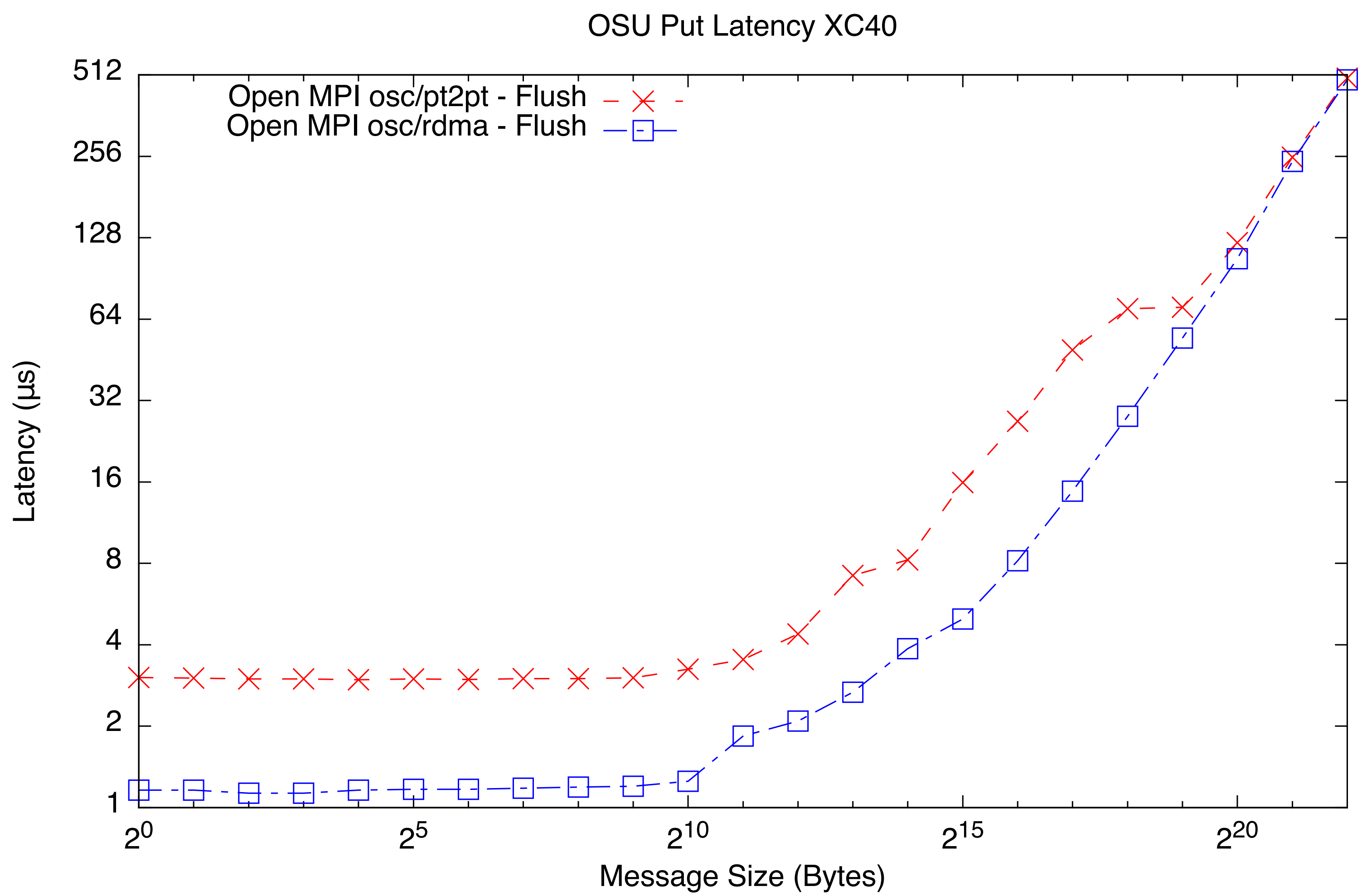
v2.x RMA Performance

- Contended MPI_Fetch_and_op performance



v2.x RMA Performance

- osc_put_latency with MPI_Win_flush on XC-40



IBM Spectrum MPI

Perry Schmidt

IBM Spectrum MPI

- IBM Spectrum MPI is a pre-built, pre-packaged version of Open MPI plus IBM value add components.
- Supports both PowerPC and x86
- Includes many of the popular Open MPI components selectable at runtime
  - E.g. MXM, usNIC, Omnipath
- Spectrum MPI 10.1.0.2 to release in December of 2016.
  - First release (Spectrum MPI 10.1) was July of 2016.
- Part of IBM Spectrum HPC Software Stack

Evaluation Downloads for Spectrum MPI:
https://www-01.ibm.com/marketing/iwm/iwm/web/reg/pick.do?source=swerpsysz-lsf-3&S_PKG=mpi101&S_TACT=000002PW&lang=en_US

IBM Value Add Components

- PAMI PML and OSC
  - Improved point-to-point and one-sided performance for Mellanox Infiniand
  - Includes support for Nvidia GPU buffers
  - Additional performance optimizations
- IBM Collective Library
  - Significant performance improvements for blocking and non-blocking collectives over OMPI collectives.
  - Dynamic collective algorithm selection.
- GPU support
  - CUDA-Aware support for Nvidia GPU cards.
  - Adding GPU RDMA Direct / Async in future release.

IBM Testing and Support

- Extensive level of testing for IBM releases
  - Standard Open MPI release testing…
  - …Plus Platform MPI test suites
  - …Plus HPC stack integration testing
- IBM Customer Support
  - For customers running a licensed copy of IBM Spectrum MPI
  - IBM will work with customers and partners to resolve issues in non IBM-owned components

Community support

- Activity participating with Open MPI Community
- MTT and Jenkins testing on IBM PowerPC servers
- New features that will be contributed back
  - Improved LSF support
  - -aff: easy to use affinity controls
  - -prot: protocol connectivity report
  - -entry: dynamic layering of multiple PMPI libraries
  - …and more…
- PMIx improvements (go to their BOF…)
  - Focus on CORAL-sized clusters
- Bug fixes, bug fixes, bug fixes…

Mellanox Community Efforts

Yossi Itigin

Mellanox
TECHNOLOGIES

The Power of Community Compels Us

- Engaged in multiple open source efforts enabling exascale MPI and PGAS applications
  - UCX
    - Open source
    - Strong vendor ecosystem
    - Near bare metal performance across a rage of fabrics
    - InfiniBand, uGNI, RoCE, shared memory
  - PMIx (PMI eXascale)
    - Open source
    - Exascale job launch
    - Supported by SLURM, LSF, PBS

Exascale Enabled Out-of-the-Box

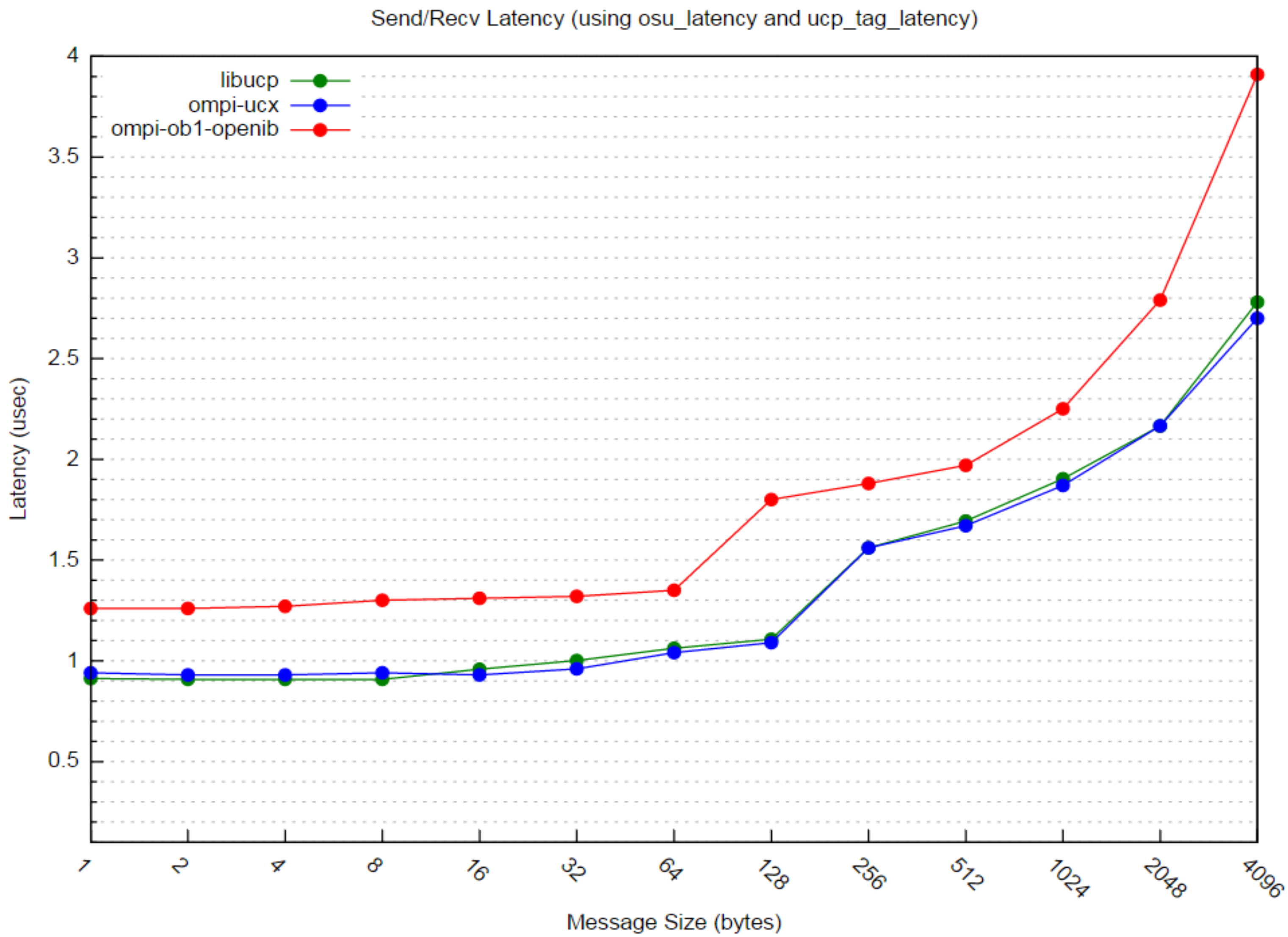
- UCX
  - UCX PML starting from v1.10 (MPI)
  - UCX SPML starting from v1.10 (OSHMEM)
  - Support for advanced PMIx features
    - Direct modex
    - Non-blocking fence
    - Eliminate the barrier in initialization
- OSHMEM
  - Open MPI v2.1.0 is (will be) OSHMEM v1.3 compliant!

OMPI-UCX Performance Data

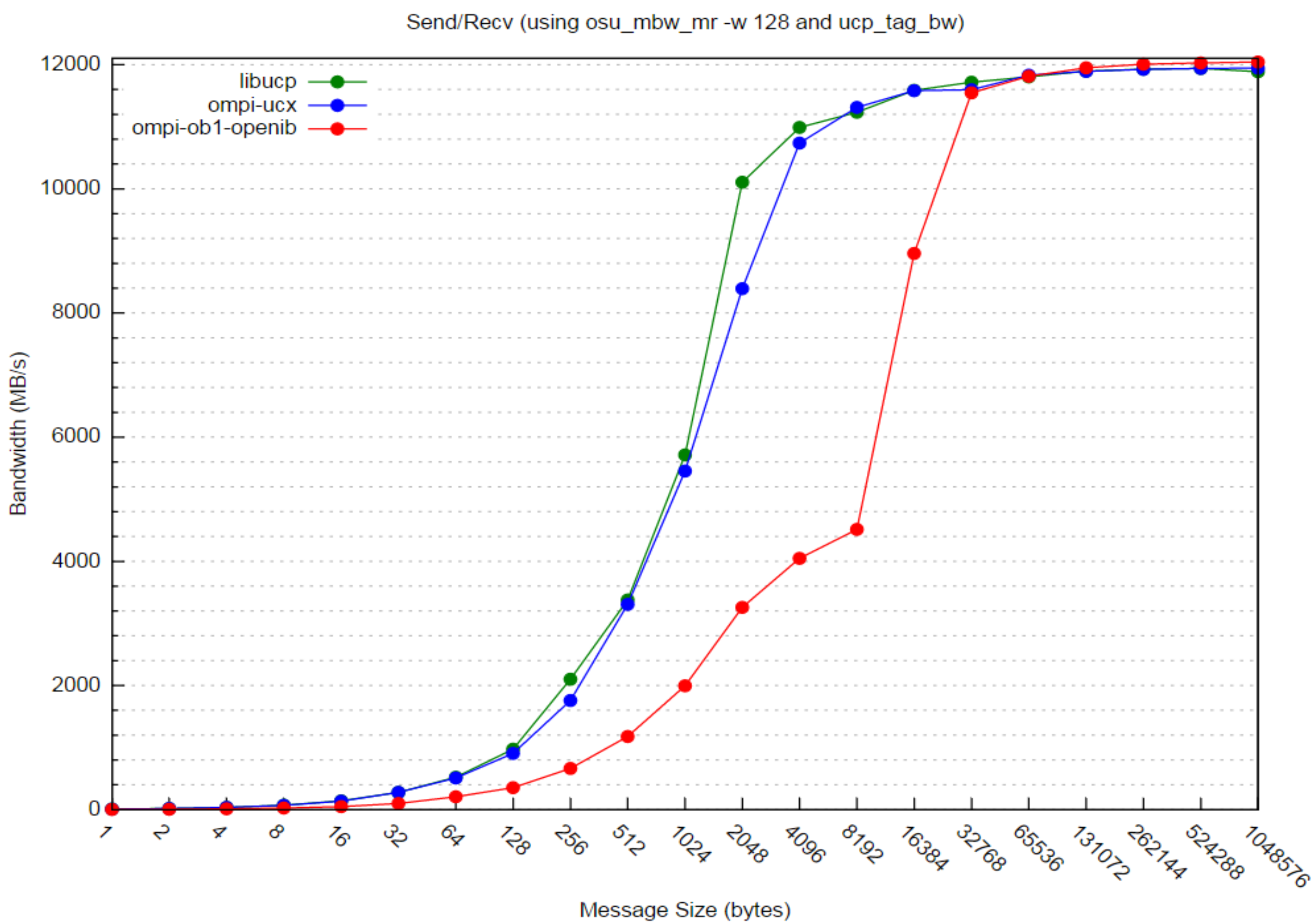
- Mellanox system
  - Switch: SX6518
  - InfiniBand ConnectX-4
  - CPU: Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
  - Red Hat Enterprise Linux Server release 7.2
  - Open MPI/SHMEM v2.0.2a1
  - UCX version:

    # UCT version=1.0.2129
    # configured with: --disable-logging --disable-debug --disable-assertions --disable-params-check --prefix=/hpc/local/benchmarks/hpcx_install_Wednesday/hpcx-icc-redhat7.2/ucx --with-knem=/hpc/local/benchmarks/hpcx_install_Wednesday/hpcx-icc-redhat7.2/knem
  - Benchmark: OSU v5.1

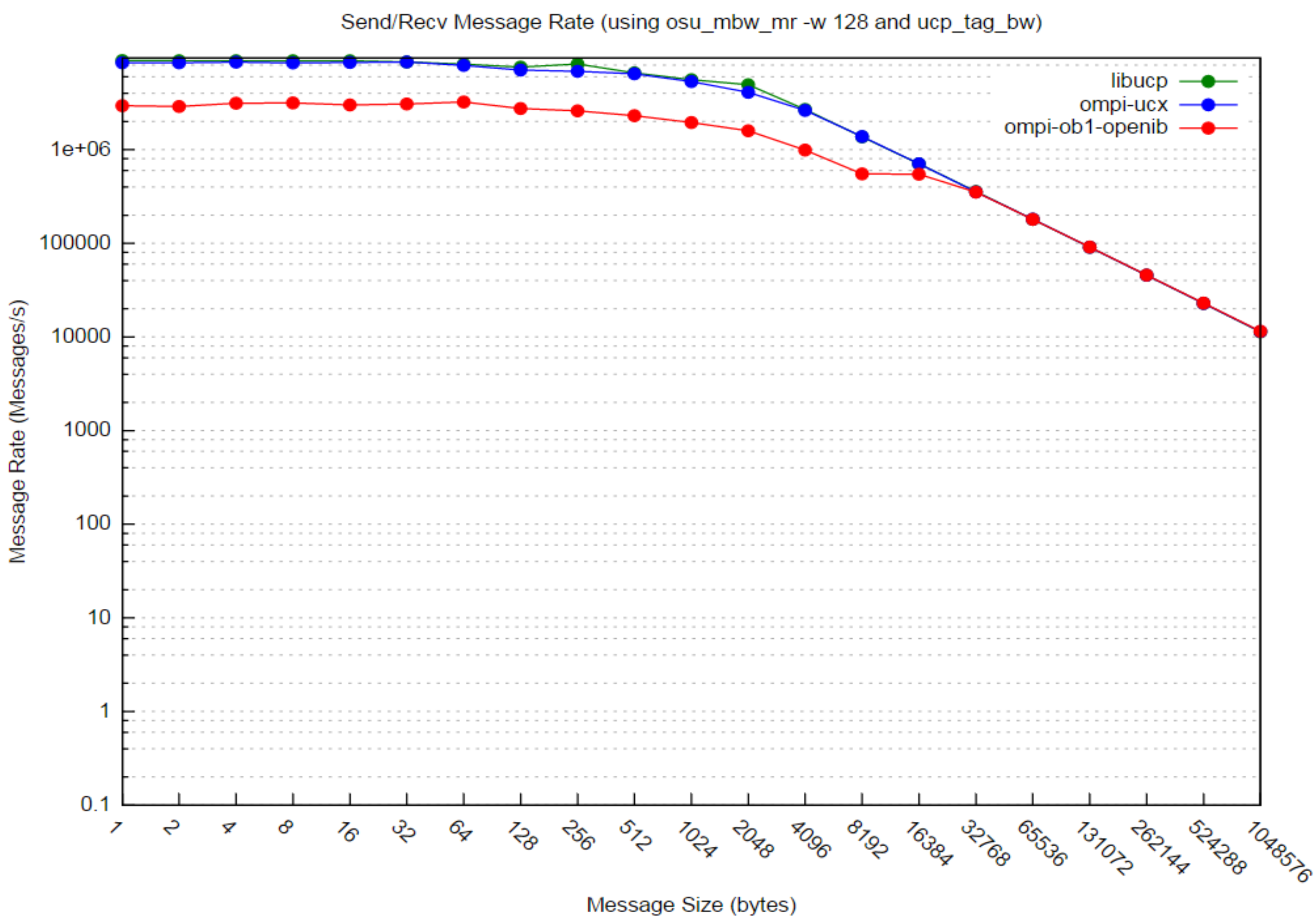
MPI UCX PML Point-to-Point Latency



MPI UCX PML Point-to-Point Bandwidth



MPI UCX PML Point-to-Point Message Rate

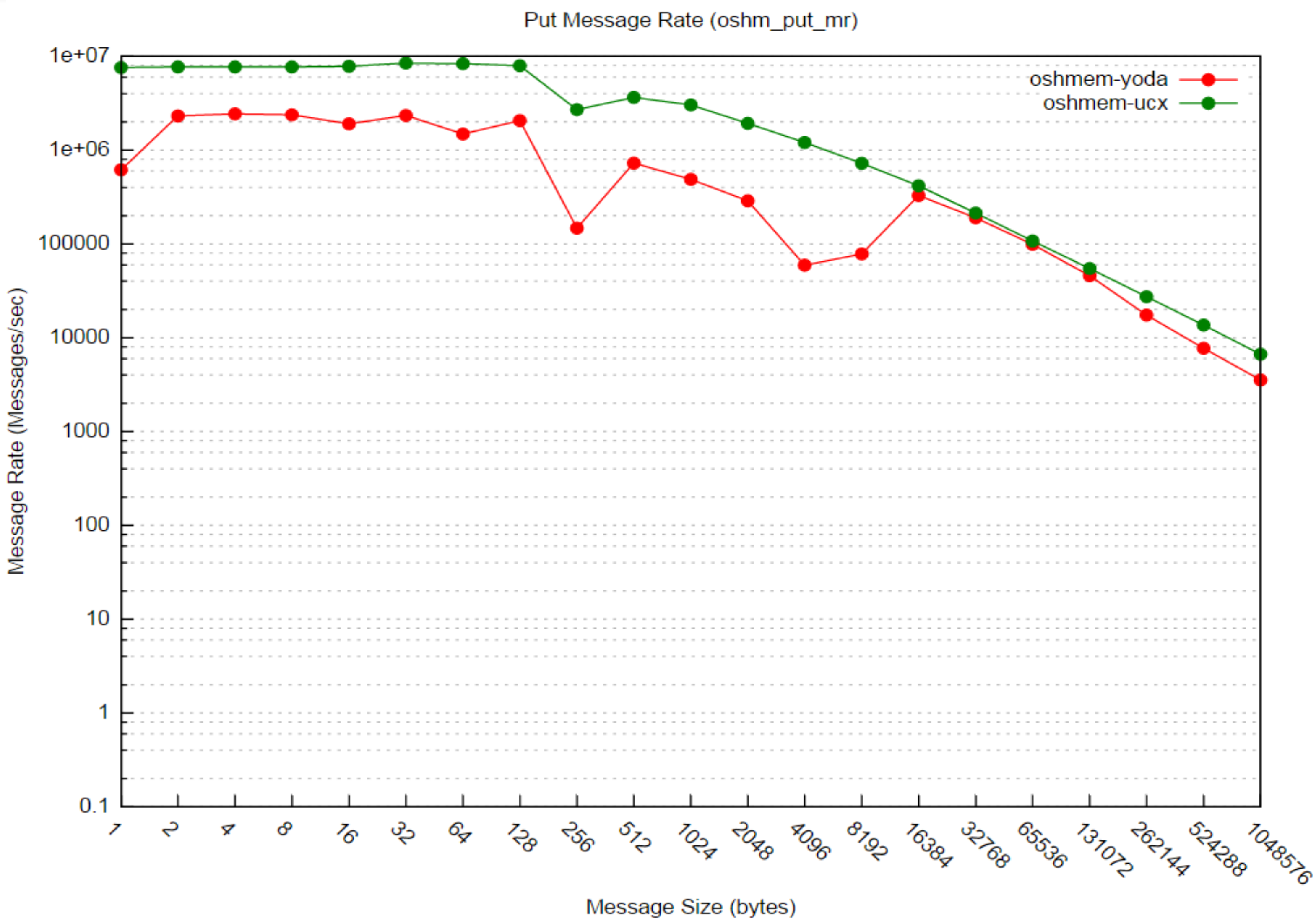


OSHMEM + UCX SPML Atomic Rates

OSU OpenSHMEM Atomic Operation Rate
ConnectX-4 HCA
Millions of atomic operations per second

Operation	RC Transport	DC Transport
shmem_int_fadd	14.03	15.6
shmem_int_finc	23.03	22.55
shmem_int_add	87.73	115.75
shmem_int_inc	81.13	122.92
shmem_int_cswap	23.14	22.74
shmem_int_swap	23.17	22.26
shmem_longlong_fadd	23.24	22.87
shmem_longlong_finc	23.15	22.83
shmem_longlong_add	80.08	91.22
shmem_longlong_inc	76.13	95.61
shmem_longlong_cswap	15.18	22.7
shmem_longlong_swap	22.79	22.84

OSHMEM + UCX SPML One Sided Message Rate (osu_oshm_put_mr)



Mapping, Ranking, Binding: Oh My!

Ralph H Castain
Intel, Inc.

Why bind?

- In Open MPI early days
  - No default binding
  - Simple bind-to-core, bind-to-socket options
- Motivators
  - Competitor “out-of-the-box” comparisons
    - Bound by default
  - Researchers
    - Fine-grained positioning, binding
  - More complex chips

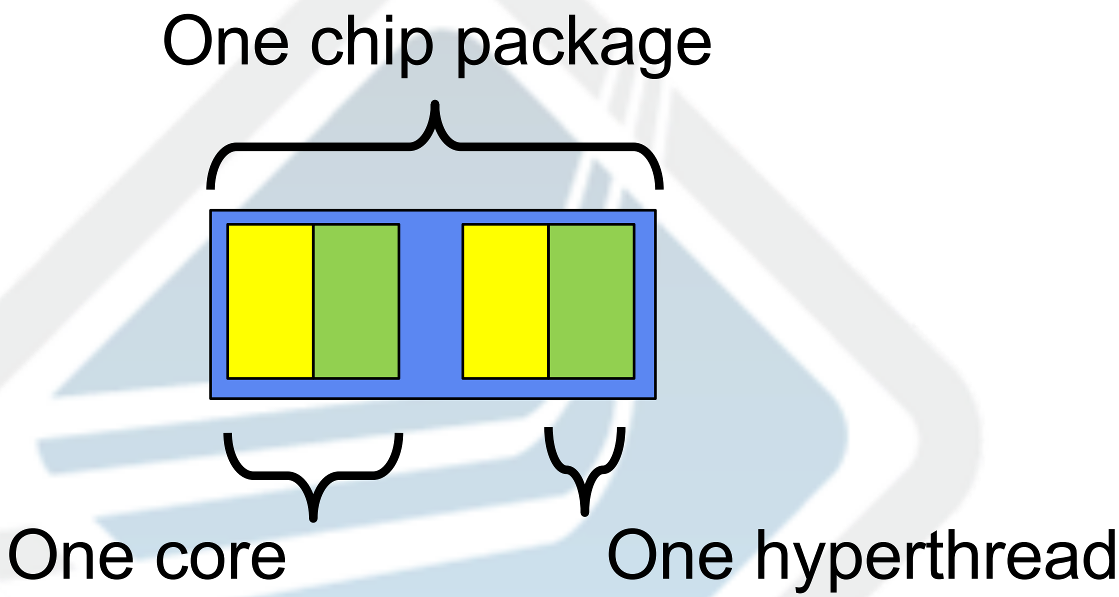
Terminology

- Slots
  - How many processes allowed on a node
    - Oversubscribe: #processes > #slots
  - Nothing to do with hardware!
    - System admins often configure a linkage
- Processing element (“PE”)
  - Smallest atomistic processor
  - Frequently core or HT (tile?)
  - Overload: more than one process bound to PE

Three Phases

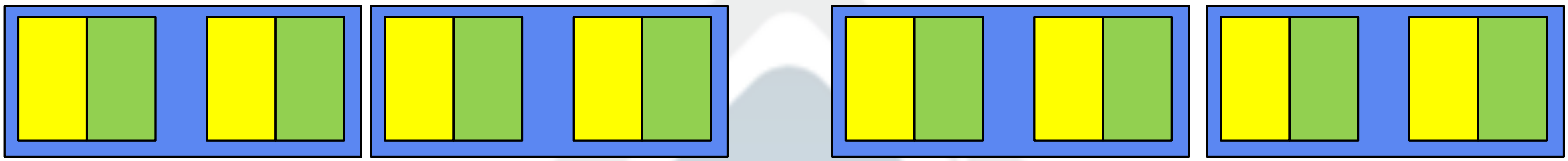
- Mapping
  - Assign each process to a location
  - Determines which processes run on what nodes
  - Detects oversubscription
- Ranking
  - Assigns MPI_COMM_WORLD rank to each process
- Binding
  - Binds processes to specific processing elements
  - Detects overload

Examples: notation



Mapping

Node 1 Slots = 3/node Node 2



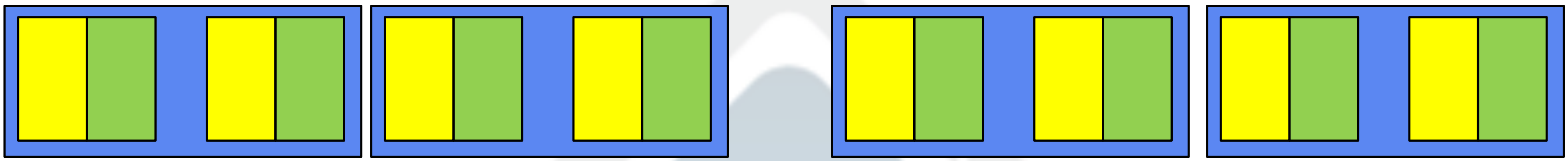
Slot
Node
NUMA
Socket
Cache (L1,2,3)
Core
HWThread

Oversubscribe
Nooversubscribe
Span
PE

#procs = 5

Mapping

Node 1 Slots = 3/node Node 2



ABC DE

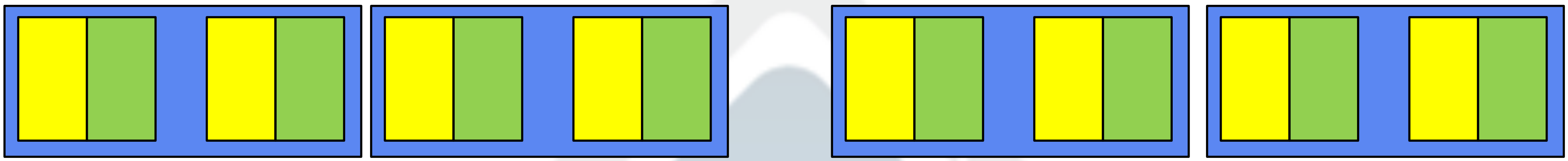
Slot
Node
NUMA
Socket
Cache (L1,2,3)
Core
HWThread

Oversubscribe
Nooversubscribe
Span
PE

#procs = 5

Mapping

Node 1 Slots = 3/node Node 2



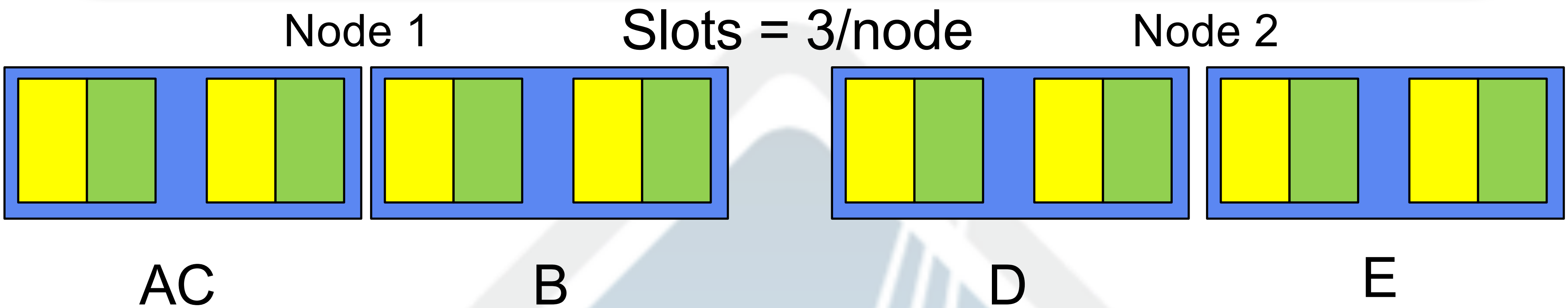
ACE BD

Slot
Node
NUMA
Socket
Cache (L1,2,3)
Core
HWThread

Oversubscribe
Nooversubscribe
Span
PE

#procs = 5

Mapping

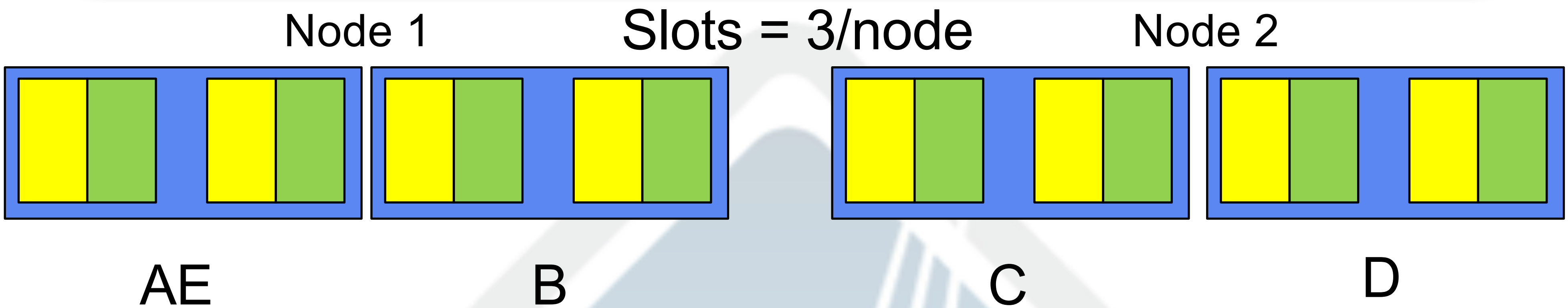


Slot
Node
NUMA
Socket
Cache (L1,2,3)
Core
HWThread

Oversubscribe
Nooversubscribe
Span
PE

#procs = 5

Mapping



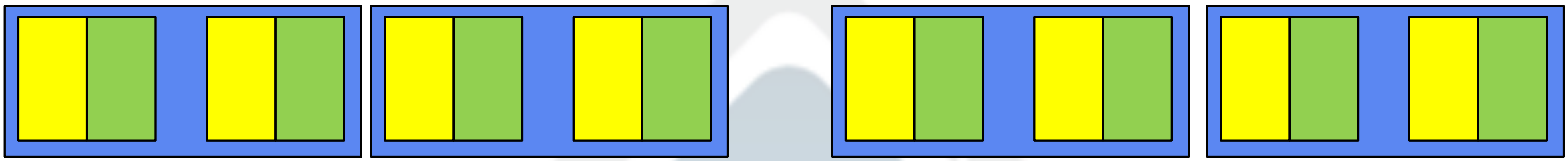
Slot
Node
NUMA
Socket ✓
Cache (L1,2,3)
Core
HWThread

Oversubscribe
Nooversubscribe
Span ✓
PE

#procs = 5

Ranking

Node 1 Slots = 3/node Node 2



AE 0,1 | B 2 | C 3 | D 4

Slot ✔
Node
NUMA
Socket
Cache (L1,2,3)
Core
HWThread

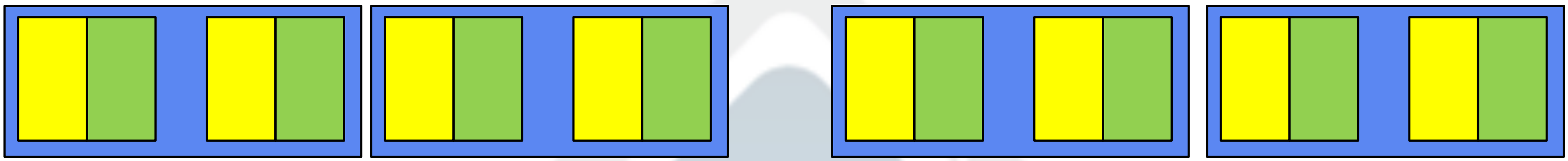
Span
Fill ✔

Default

Map-by socket:span
#procs = 5

Ranking

Node 1 Slots = 3/node Node 2



AE 0,2

B 4

C 1

D 3

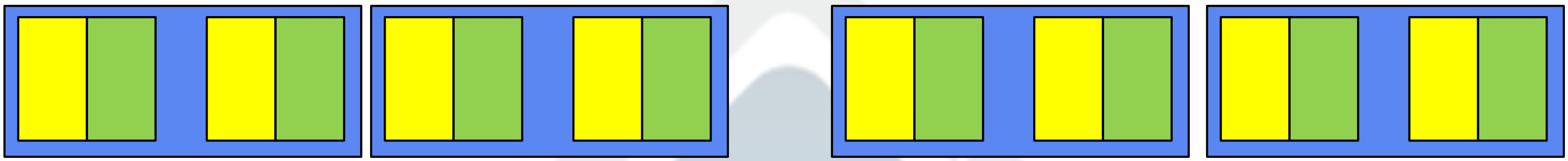
Slot
Node ✔
NUMA
Socket
Cache (L1,2,3)
Core
HWThread

Span
Fill

Map-by socket:span
#procs = 5

Ranking

Node 1 Slots = 3/node Node 2



AE	B	C	D
0,2	1	3	4

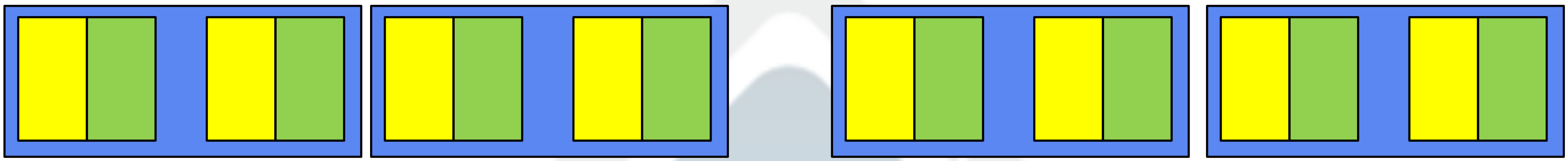
Slot
Node
NUMA
Socket ✔
Cache (L1,2,3)
Core
HWThread

Span
Fill

Map-by socket:span
#procs = 5

Ranking

Node 1 Slots = 3/node Node 2



AE 0,4 | B 1 | C 2 | D 3

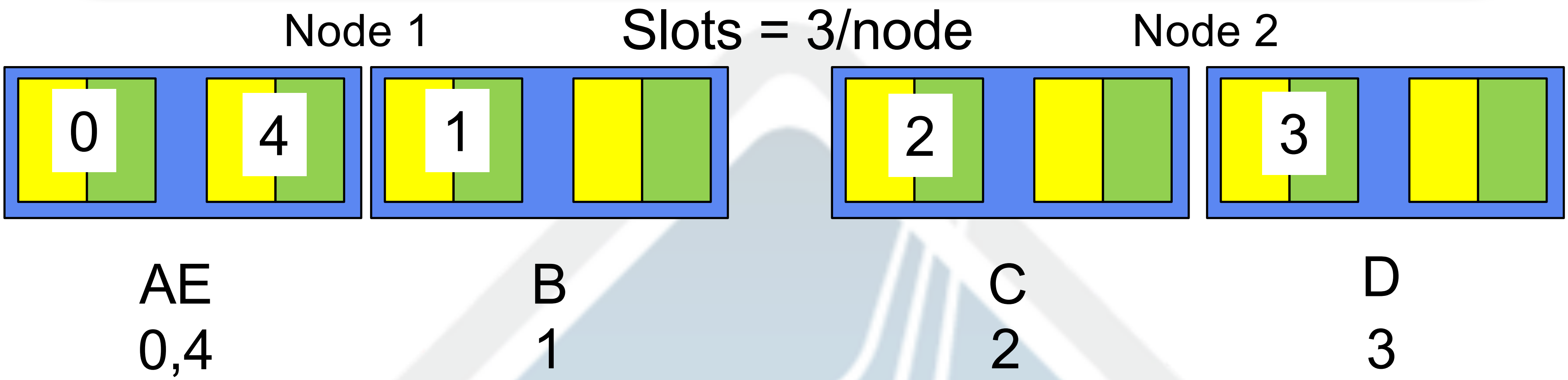
Slot
Node
NUMA
Socket ✔
Cache (L1,2,3)
Core
HWThread

Span ✔
Fill

Map-by socket:span
#procs = 5

Binding

Slots = 3/node

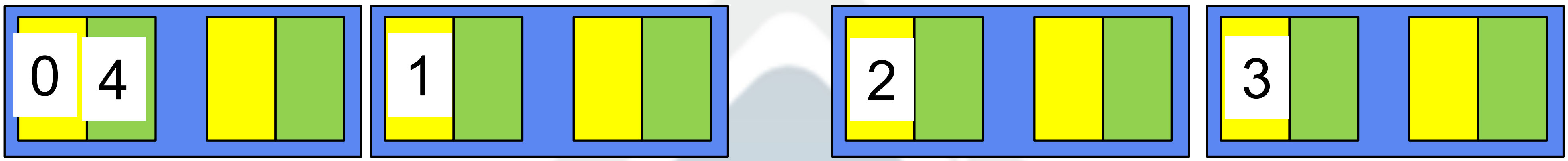


NUMA
Socket
Cache (L1,2,3)
Core
HWThread

Rank-by socket:span
Map-by socket:span
#procs = 5

Binding

Node 1 Slots = 3/node Node 2



AE
0,4

B
1

C
2

D
3

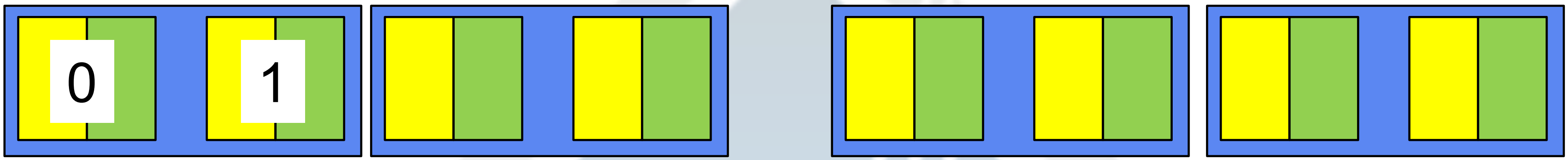
NUMA
Socket
Cache (L1,2,3)
Core
HWThread ✔

Rank-by socket:span
Map-by socket:span
#procs = 5

Defaults

#procs <= 2

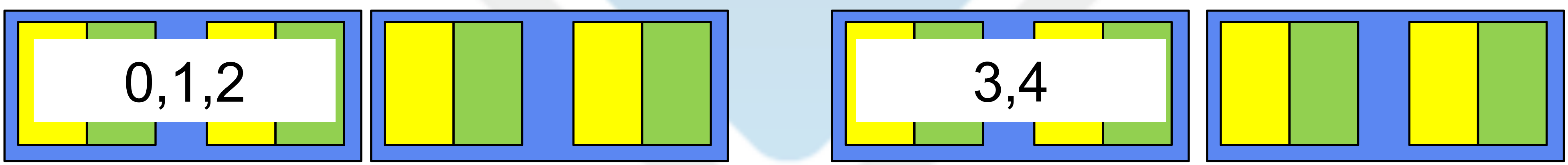
Node 1 Slots = 3/node Node 2



Map-by core Rank-by slot Bind-to core

#procs > 2

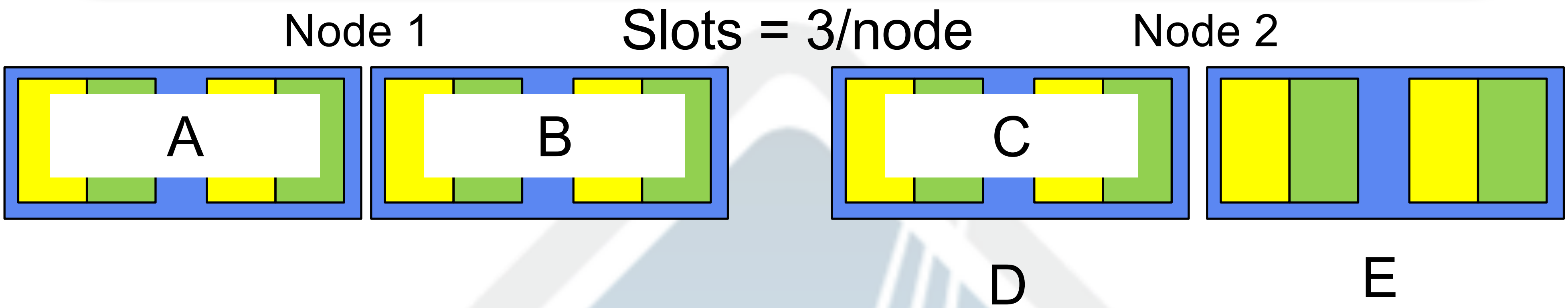
Node 1 Slots = 3/node Node 2



Map-by slot Rank-by slot Bind-to NUMA

Mapping: PE Option

Slots = 3/node

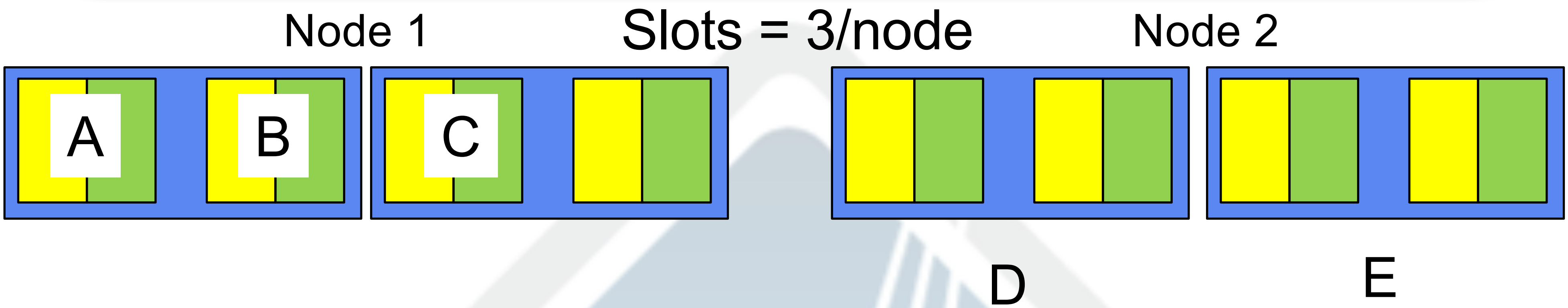


Slot
Node
NUMA
Socket
Cache (L1,2,3)
Core ✔
HWThread

Oversubscribe
Nooversubscribe
Span
PE=2 ✔ => bind-to core

Rank-by slot
#procs = 3

Mapping: PE Option



Slot
Node
NUMA
Socket
Cache (L1,2,3)
Core
HWThread

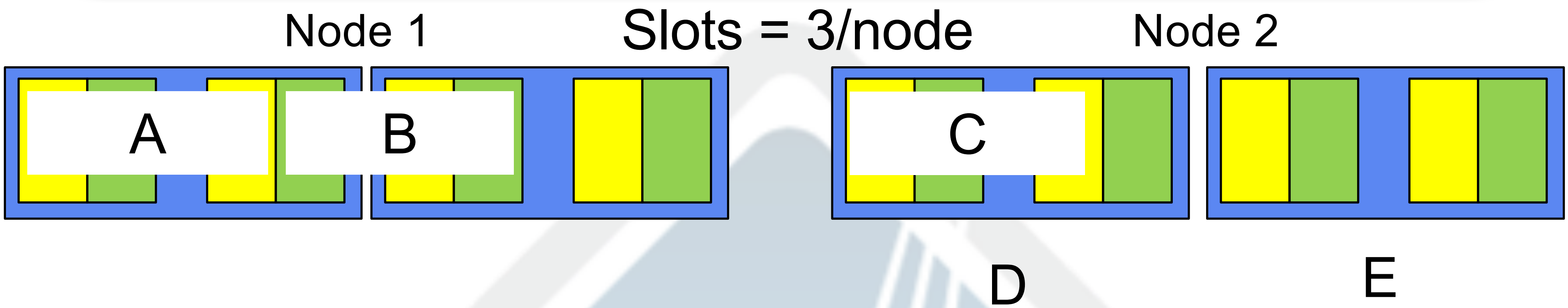
Oversubscribe
Nooversubscribe
Span
PE=2 => bind-to hwt

hwthreads-as-cpus

Rank-by slot
#procs = 3

Mapping: PE Option

Slots = 3/node



Slot
Node
NUMA
Socket
Cache (L1,2,3)
Core ✔
HWThread

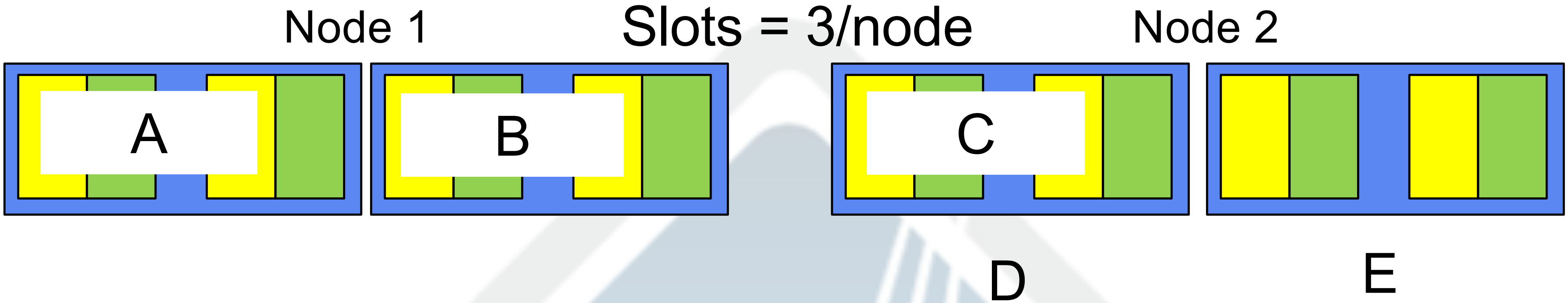
hwthreads-as-cpus

Oversubscribe
Nooversubscribe
Span
PE=3 ✔ => bind-to hwt

Rank-by slot
#procs = 3

Mapping: PE Option

Slots = 3/node



Slot
Node
NUMA
Socket ✔
Cache (L1,2,3)
Core
HWThread

Oversubscribe
Nooversubscribe
Span
PE=3 ✔ => bind-to hwt

hwthreads-as-cpus

Rank-by slot
#procs = 3

Conclusion

- bind-to defaults
  - map-by specified → bind to that level
  - map-by not specified
    - np <= 2: bind-to core
    - np > 2: bind-to NUMA
  - PE > 1 → bind to PE
- Map, rank, bind
  - Separate dimensions
  - Considerable flexibility

Right combination is highly application specific!

Wrap up

Where do we need help?

- Code
  - Any bug that bothers you
  - Any feature that you can add
- ***User documentation***
- Testing
- Usability
- Release engineering

Come join us!

Jeff Squyres

George Bosilca

Perry Schmidt

Ralph Castain

Yossi Itigin

Nathan Hjelm