Hewlett Packard Enterprise

MPI Topics on the HPE Cray EX supercomputer

Comprehensive General LUMI Course

April 23–26, 2024

Agenda

- Message passing and Cray MPICH in general
- Overlapping communication
- Environment variables for MPI
- Cray MPICH on Slingshot
- GPU Support in Cray MPICH
- Rank Reordering
- MPMD application launch

Basics about communication

The basics

With very few exceptions parallel applications will communicate data

This communication can be characterized by

- Latency
  - The time it takes for a message to get to a destination
  - Composed of constant hardware and software overheads
  - Dominates the performance of small messages
- Bandwidth
  - The maximum rate at which data can flow over the network.
  - Dominates the performance of larger messages
  - Bandwidth between nodes generally depends upon the number of possible paths between nodes on the network (topology)
  - Can usually be tuned with a large enough budget

How message size affects performance

- The decisions made by application developer can affect the overall performance of the application.
- The size of messages sent between processes affects how important latency and bandwidth costs become.
- When a message is small the network latency is dominant.
- Therefore it is advisable to try and bundle multiple small messages into fewer larger messages to reduce the number of latency penalties.
- This is true for all closely coupled communication over any protocols,
  eg MPI, SHMEM, UPC, TCP/IP

On- and off-node performance

- The rise of multi-core has led to fat nodes being common
  - 18 years ago we had one or two CPUs per node…
  - Now we routinely see 128 CPUs per node

- Codes usually have multiple MPI ranks per node
  - Many (even most) codes are flat MPI
    – rather than hybrid with, for instance, OpenMP threads
  - Even hybrid codes usually have more than one rank per node
    – as threading does not usually scale well across NUMA regions (e.g. sockets)

- Latency and bandwidth are different for on- and off-node messages
  - messages between PEs on the same node (intra-node) will be faster
  - messages between PEs on different nodes (inter-node) will be slower

- We can optimise application performance by maximising communication between process on the same node/socket

Overlapping Communication

Overlapping Communication with Computation

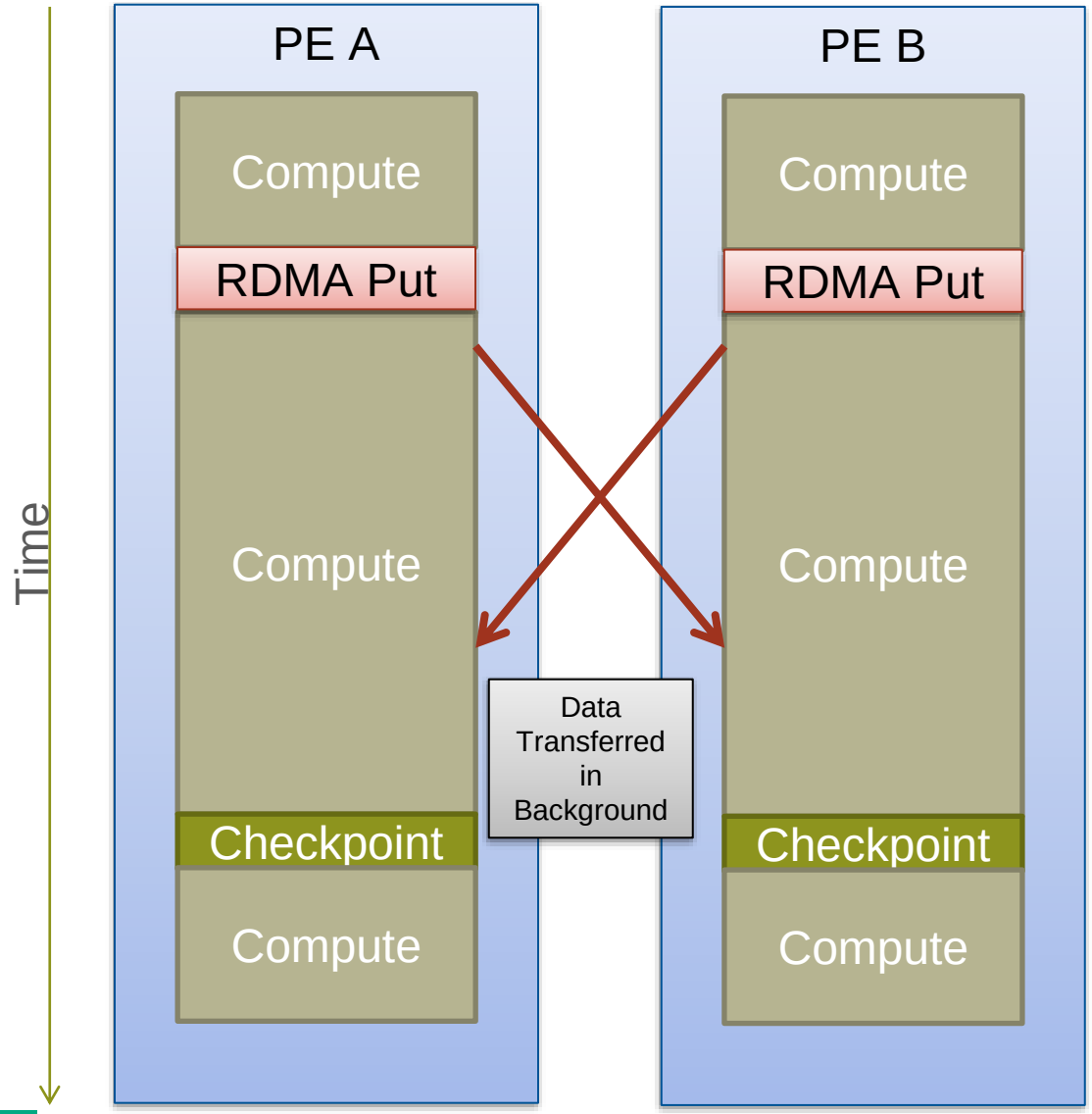
- The Holy Grail

- Do communication "in the background"
  - While each PE does (separate) computation

- The cost of communication is then almost nothing
  - Save the overhead of initiating transfers and synchronisation

- Relies on
  - Having enough (independent) computation to hide the comms time
  - Having the correct code structure to make this possible
  - Using non-blocking communication calls, e,g. via RDMA

Using RDMA to overlap communication and computation



So, rather than sitting waiting for a communication operation to complete, applications can use asynchronous RDMA operations instead. e.g. putting some data into a remote PEs memory.

The application could then continue with other useful computation until the checkpoint where the data is required.

Programming models to achieve overlap

- PGAS programming models (Fortran coarrays, SHMEM, UPC, Chapel?, GPI, MPI3 RMA
  - These tend to work on some form of ‘symmetric’ allocation
  - Put and Get semantics
  - Often complicated synchronization semantics

- However, the 2-sided communication API in MPI is by far the most popular

Two-sided protocols

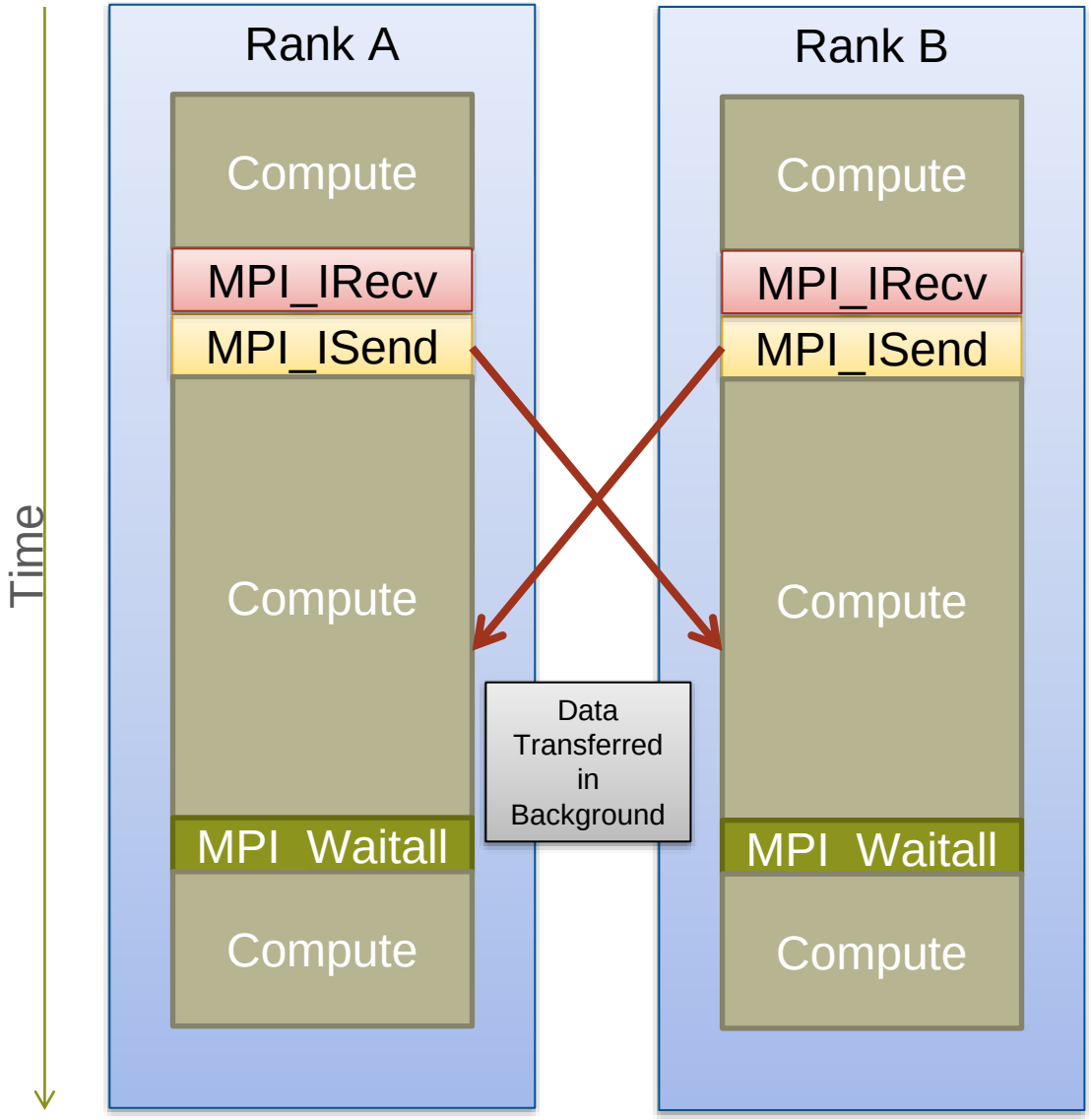
- Typically two-sided protocols like MPI are easier to use.
  - The implicit synchronisation between PEs makes it easier to write programs that are not as vulnerable to race conditions.
  - They allow for data to be sent or received into or from any part of the PEs address space
  - Messages can be matched (or not) via tags or by the PE source (`MPI_ANY_SOURCE`, `MPI_ANY_TAG`)

- However this additional flexibility often requires the CPU to perform many of these tasks

- This means communication may wait until the CPU enters an MPI call.

- Overheads caused by the MPI standard may increase latency and reduce effective bandwidth!

Overlapping communication and computation with MPI



The MPI API provides many functions that allow point-to-point messages to be performed asynchronously.

Also collectives with MPI-3

Ideally applications would be able to overlap communication and computation, hiding all data transfer behind useful computation.

Unfortunately this is not always possible in the application and not always possible at the implementation level.

What prevents overlap

- Overlapping computation/comms not always possible
  - even if library has asynchronous API calls
  - and the application has enough computation to allow overlap

- Usual reason:
  - sending PE does not know where to put messages on the destination
    - this is part of the `MPI_Recv`, not `MPI_Send`.

- The host CPU is often required
  - complex tasks performed on the CPU
    - e.g. matching message tags with the sender and receiver
  - But messages can only "progress" when program is in MPI
    - i.e. within MPI library function or subroutine
    - even if this is just a call to MPI_Probe

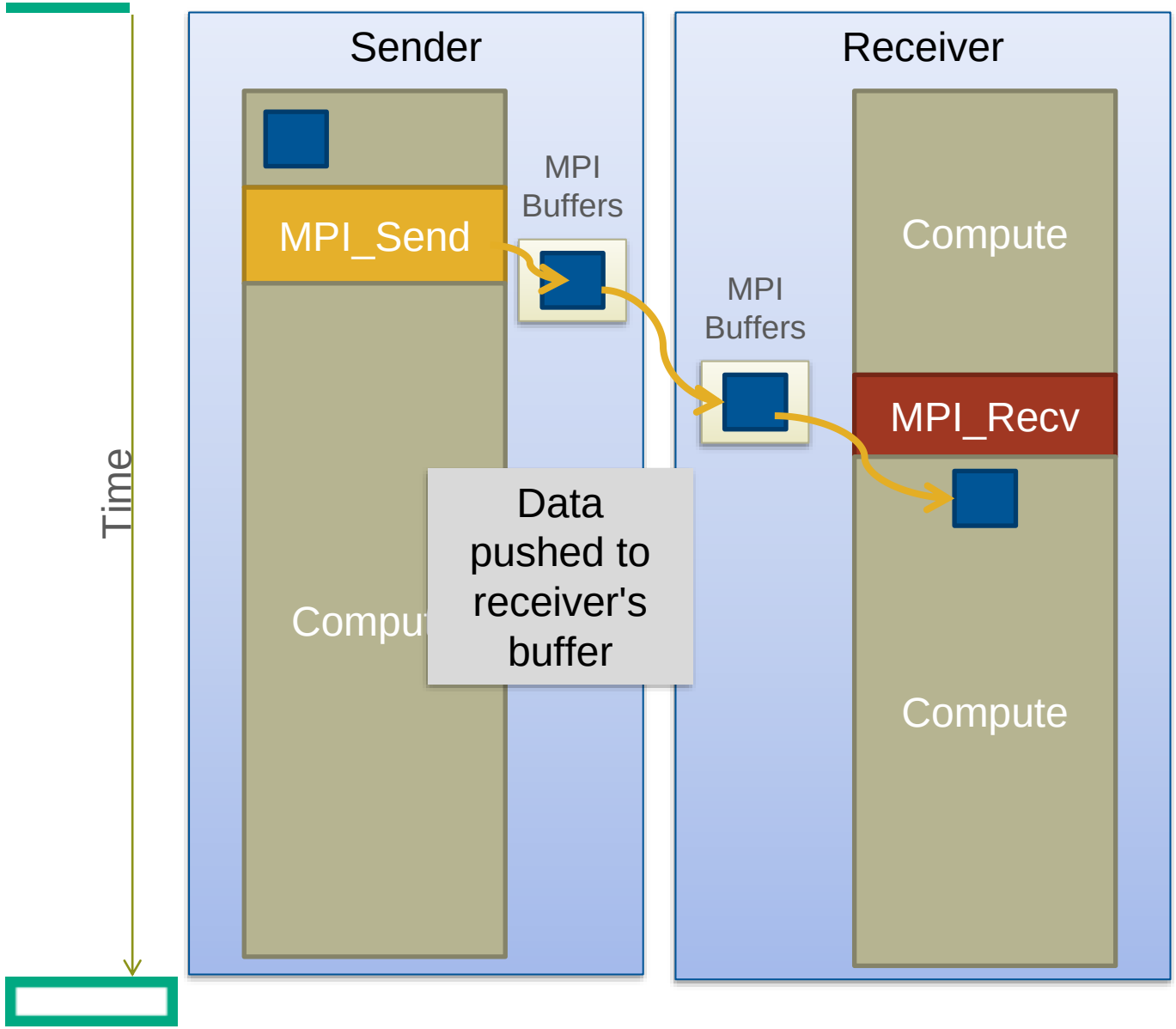
MPI Messaging Protocols

- To understand when overlap is or isn't possible
  - need to understand how MPI actually sends messages

- Multiple different protocols
  - choice depends on message size

1. Eager messaging
   - Used for small messages
   - Offers good potential for overlap
2. Segmentation And Reassembly (SAR) [only used in SS10 systems]
   - Similar to eager with multiple messages (new for the Slingshot implementation)
3. Rendezvous messaging
   - Used for large messages
   - Does not usually overlap (without progress engine)

EAGER buffering small messages



Smaller messages can avoid this problem using the eager protocol.

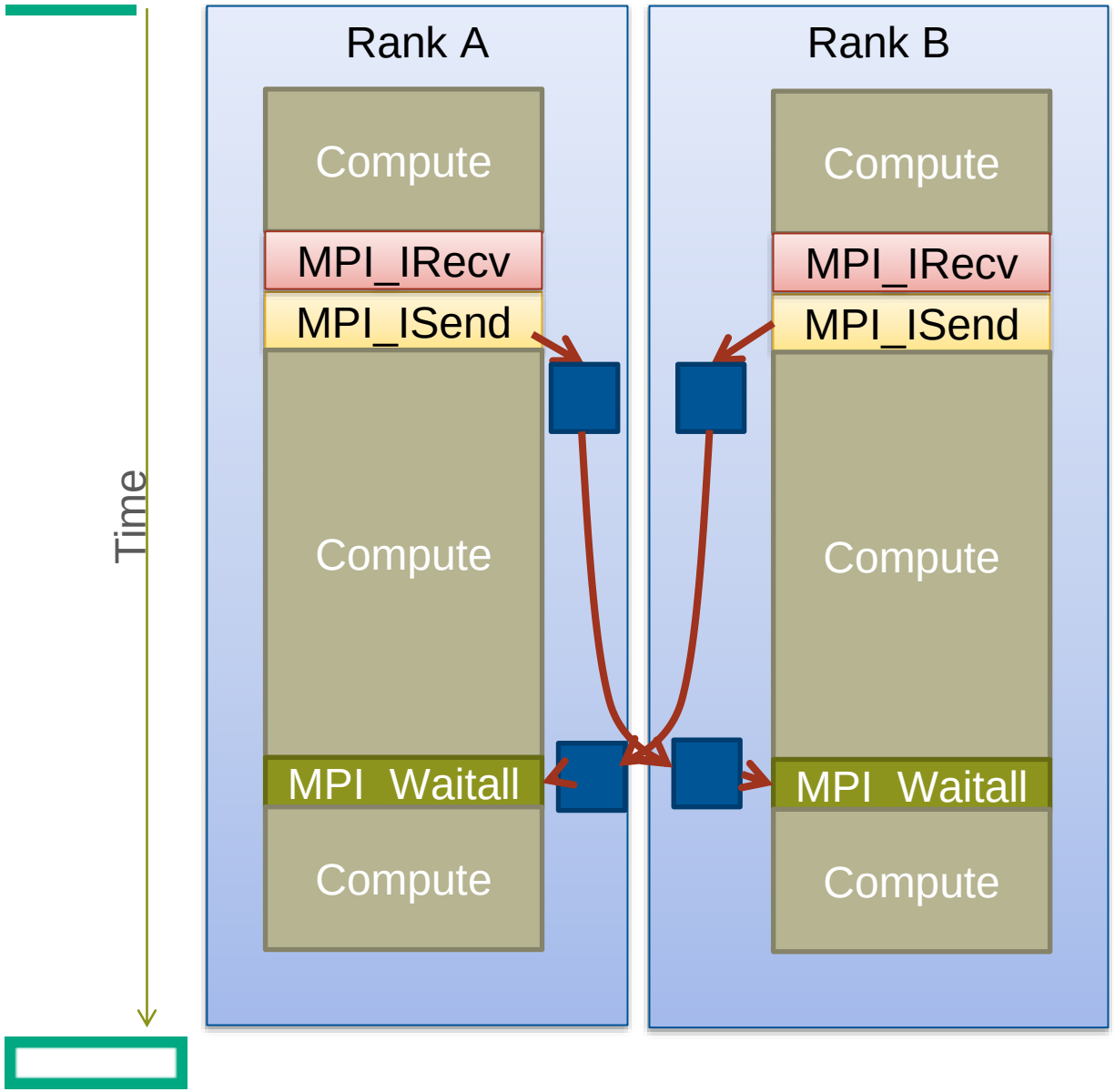
If the sender does not know where to put a message it can be buffered until the sender is ready to take it.

When MPI_Recv is called the library fetches the message data from the remote buffer and into the appropriate location (or potentially local buffer)

Sender can proceed as soon as data has been copied to the buffer.

Sender will block if there are no free buffers

EAGER potentially allows overlapping

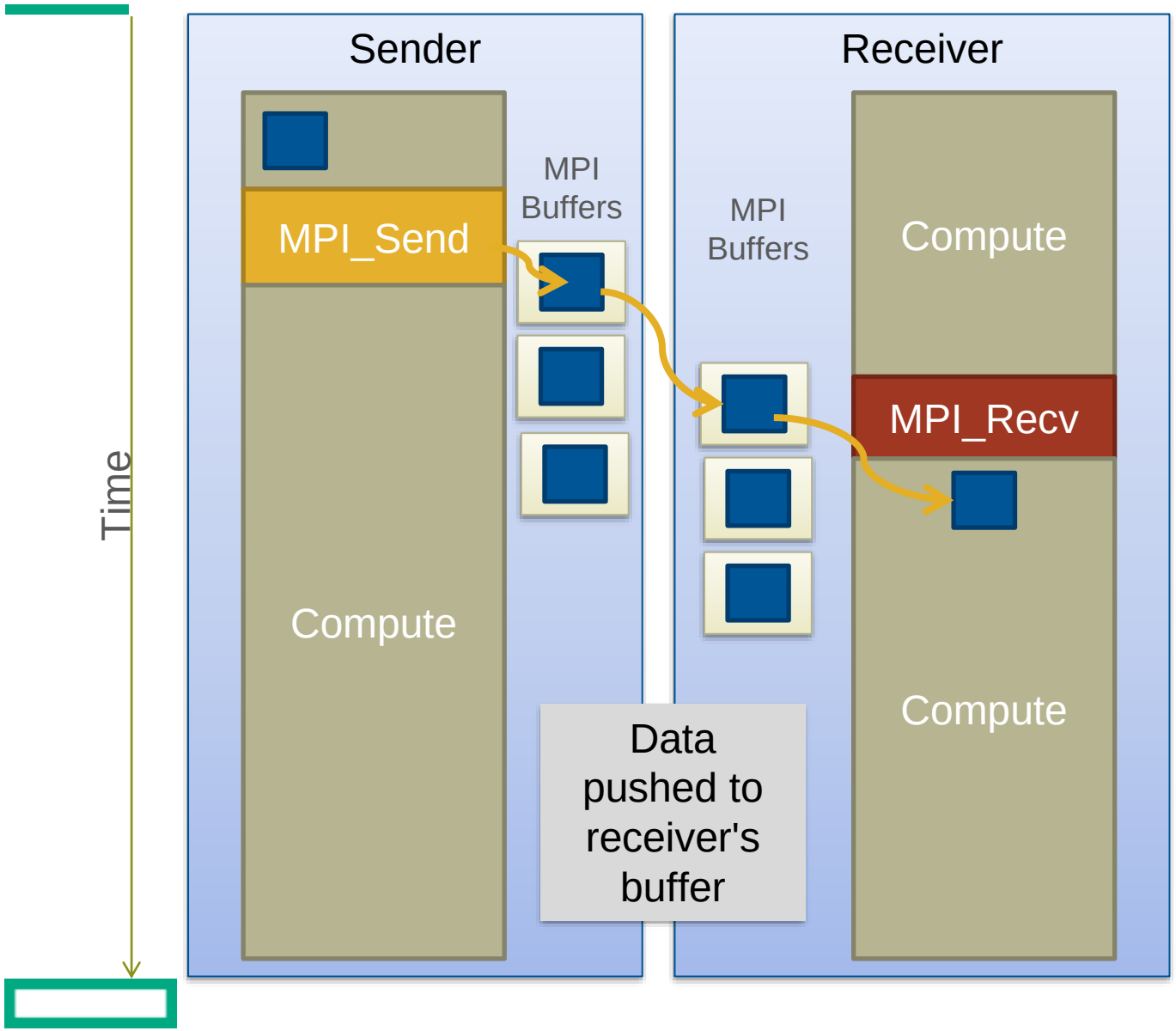


Data is pushed into an empty buffer(s) on the remote processor.

Data is copied from the buffer into the real receive destination when the MPI_Wait or MPI_Waitall is called.

Involves an extra memory copy, but much greater opportunity for overlap of computation and communication.

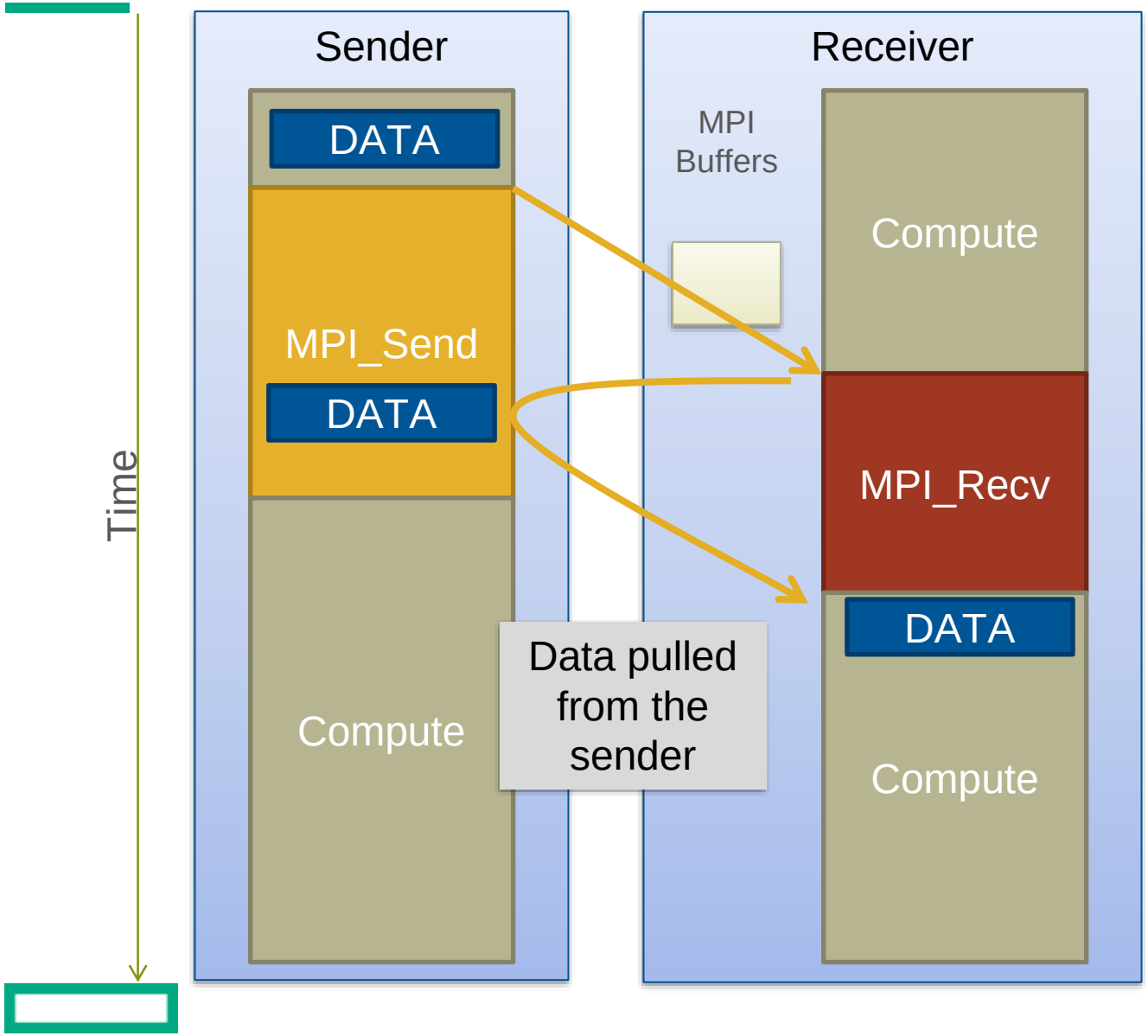
Segmentation and Reassembly (SAR)



Like Eager but data sent in chunks and reassembled on receive side

Reassembly is done in the MPI_Recv

Rendezvous Messaging – larger messages



Larger messages (that are too big to fit in the buffers) are sent via the rendezvous protocol

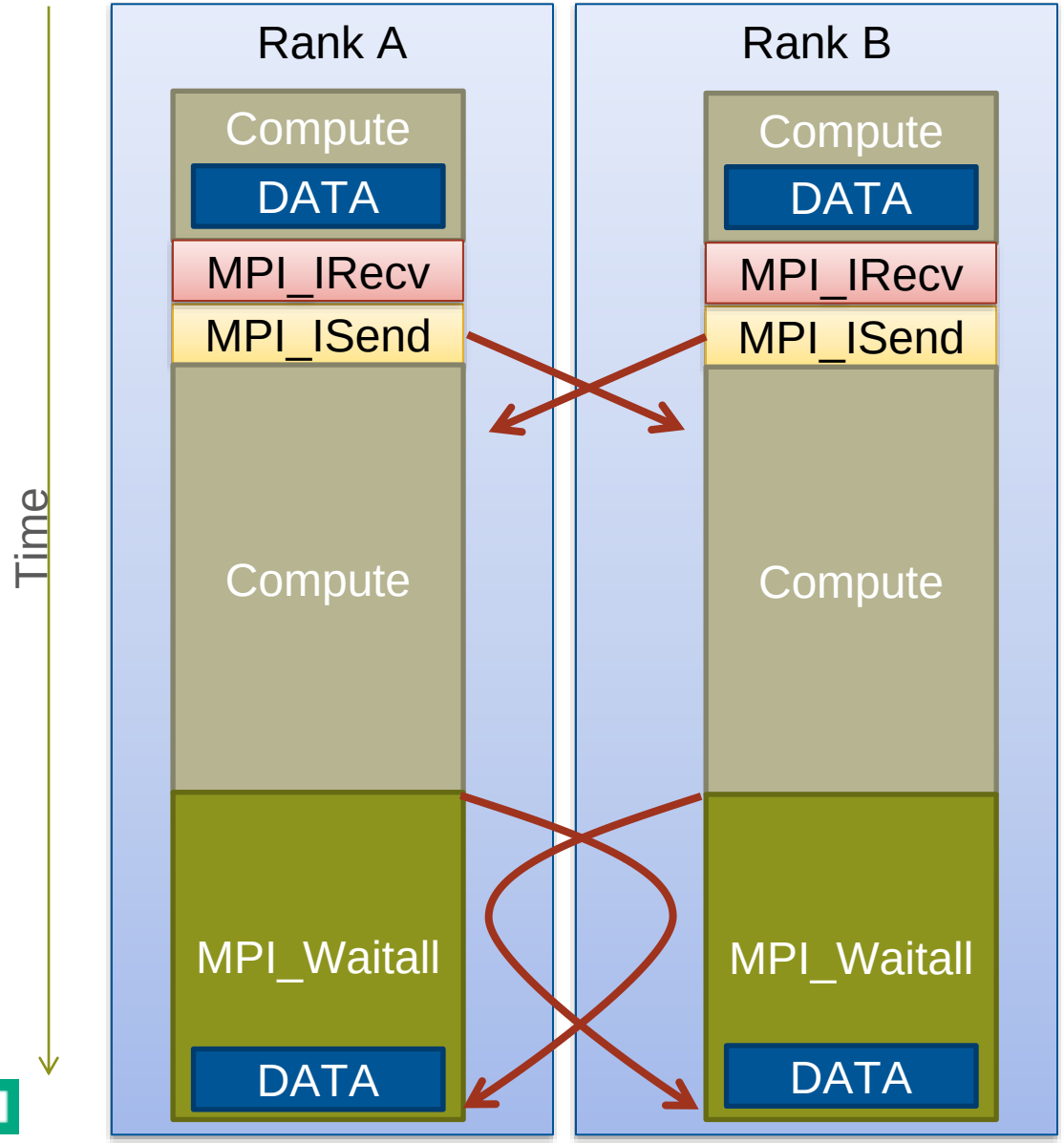
Messages cannot begin transfer until MPI_Recv called by the receiver.

Data is pulled from the sender by the receiver.

Sender must wait for data to be copied to receiver before continuing.

Sender and Receiver block until communication is finished

Rendezvous does not usually overlap



With rendezvous data transfer is often only occurs during the Wait or Waitall statement.

When the message arrives at the destination, the host CPU is busy doing computation, so is unable to do any message matching.

Control only returns to the library when MPI_Waitall occurs and does not return until all data is transferred.

There has been no overlap of computation and communication.

Making messages more eager

- One way to improve performance
  - send more messages on the eager protocol; potentially more overlap

- Do this by raising the value of the eager/Rendezvous threshold
  - set environment variable in jobscript
  - `export FI_CXI_RDZV_THRESHOLD=<value>`
  - `value` is in bytes:
    - default is 16364 bytes. (~16 kB)
  - Messages sized above this value will use Rendezvous

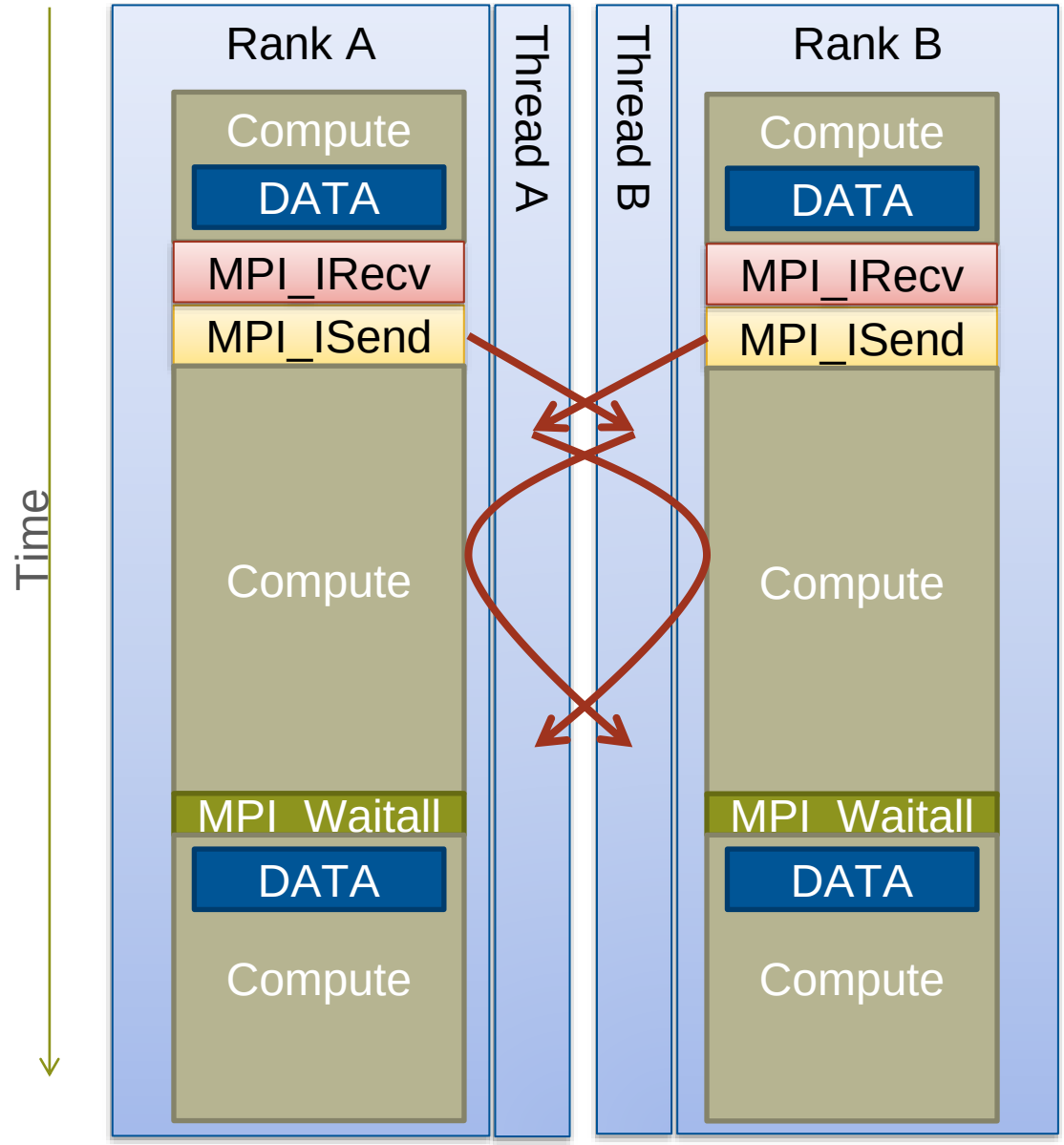
- When might this help
  - If MPI takes a significant time in the profile
  - If you have a lot of messages between 16kB and, say, 256 kB

    - CrayPAT MPI tracing can tell you this

- Also try to post `MPI_IRecv` call before the `MPI_ISend` call
  - can avoid unnecessary buffer copies

The MPI Progress Engine

Progress helps deliver overlap



Cray's MPT library can spawn additional threads that allow progress of messages while computation occurs in the background.

Thread performs message matching and initiates the transfer.

Data has already arrived by the time Waitall is called, so overlap between compute and communication.

Async Progress Engine support

- Used to improve communication/computation overlap
  - Each MPI rank starts a “helper thread” during MPI_Init
- Helper threads progress MPI engine in background
  - while the application computes
- These threads need some resource to run on !

Using the Progress Engine

To enable on HPE Cray EX,
export MPICH_ASYNC_PROGRESS=1
(note that you don't need to increase MPICH_MAX_THREAD_SAFETY)

Also need somewhere for progress engine threads to run:

- Each MPI rank will have one extra thread (so OMP_NUM_THREADS+1 threads in total)
- Appropriate number of CPUs (or hyperthreads) need to be assigned using srun
  - If you are running without hyperthreads: srun --hint=nomultithread …
    - the (second) hyperthreads are spare, and can be used for the progress engine threads
    - the exact flag combination for this is case-dependent
  - If you are running with hyperthreads: srun --hint=multithread …
    - assign an extra resource to each rank by increasing srun --cpus-per-task by 1
- Tip: use placement reporting test applications (e.g. acheck, xthi) to check this

Will the Progress Engine help?

- For codes that spend a lot of time on large-message transfers
  - and using non-blocking MPI calls

- Yes, it can help
  - 10% or more performance improvements seen with some apps

- Why might it not help
  - (even if we have slow, large message transfers with non-blocking MPI)
  - MPICH_ASYNC_PROGRESS has performance implications
  - Leaving cores free means fewer processes per node
    - Less computational power per node
    - Reduced amount of intra-node MPI messages

Optimizing MPI Collectives

- Cray MPICH uses various algorithms for several collectives
  - `Alltoall, Allreduce, Allgather, Allgatherv, Gatherv, Scatterv`
- Library-internal decisions on which to use are based on
  - number of ranks on the calling communicator
  - message sizes
- Can change this decision with environment variables
  - see the MPI man page ‘man mpi’ for full environment variables listing
  - each collective will have a different ENV variable
  - eg export `MPICH_ALLGATHER_VSHORT_MSG=128`
- When might you try this
  - eg If `Allgather` suddenly becomes very important for a small change in problem size

Miscellaneous Useful Flags

- Performance enhancements
  - `export MPICH_COLL_SYNC=1`
    - Adds a barrier before collectives, try this if perftools makes your code run faster.
- Reporting
  - `export MPICH_CPUMASK_DISPLAY=1`
    - Shows the binding of each MPI rank by core and hostname
  - `export MPICH_ENV_DISPLAY=1`
    - Print the value of all MPI environment variables at runtime (STDERR)
  - `export MPICH_MPIIO_STATS=[1,2]`
    - Prints some MPI-IO stats useful for optimisation (STDERR) or outputs comprehensive data to filesystem
  - export MPICH_MEMORY_REPORT=[1,2,3]
  - `export MPICH_RANK_REORDER_DISPLAY=1`
    - Prints the node that each rank is residing on, useful for checking `MPICH_RANK_REORDER_METHOD` results.
  - `export MPICH_VERSION_DISPLAY=1`
    - Display library version and build information.
- For more information: `man mpi`

How Can I make MPI Faster?

- Runtime options
  - Try to maximise on-node transfers (rank reordering)
  - If you need to communicate a lot off node then under populating nodes may help
- Help the MPI library get better overlap
  - Try to call MPI often (`MPI_Test`, `MPI_Wait`, `MPI_Request_get_status` will help progress)
  - `MPI_Testsome` is better than `MPI_Testany`
  - use non-blocking MPI calls
    - `MPI_Isend`, `MPI_Irecv`, `MPI_Iallgather`…
  - small messages use the EAGER protocol with good overlap potential
    - try to send more data using the small-message EAGER method
    - consider raising the EAGER threshold
  - larger messages use the RENDEZVOUS protocol
    - post non-blocking receives as early as possible (with work between irecv and the sends)
    - consider using the asynchronous progress thread

How Can I make MPI Faster...?

- Try to reorder code to give more potential for overlap
  - local computation (or I/O) that can be done while messages transfer
- Perhaps consider adding some PGAS

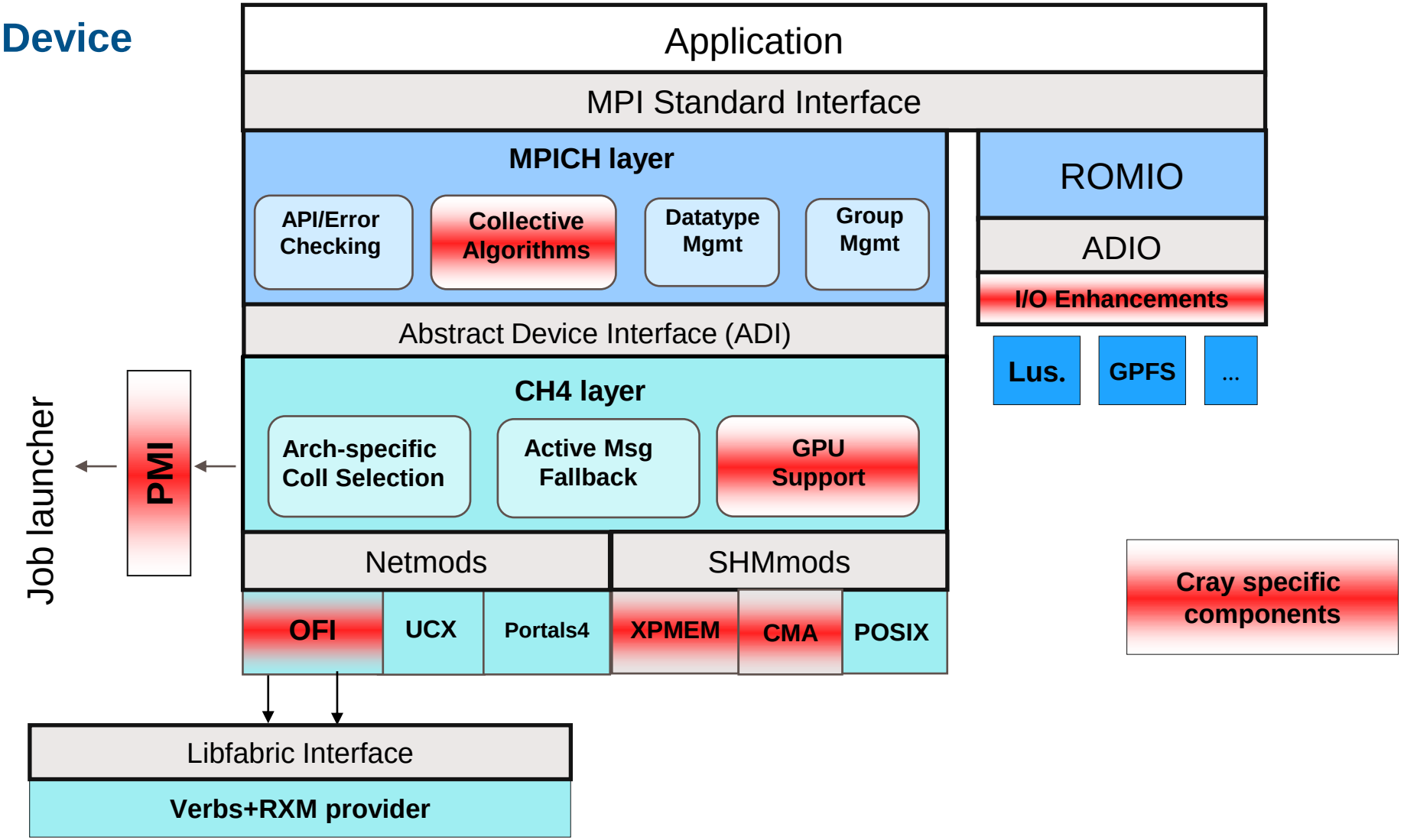
New features in Cray MPICH on Slingshot

Agenda

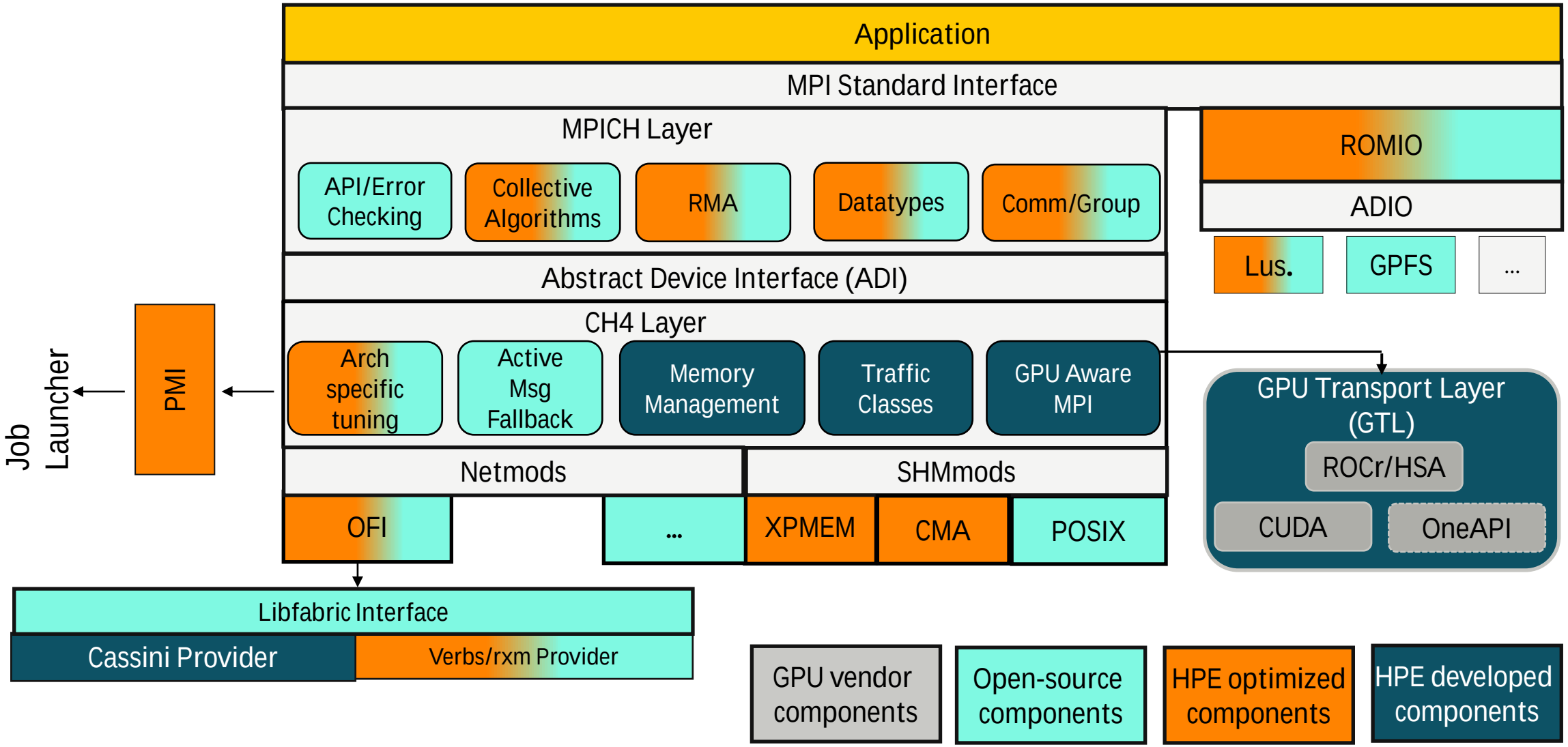
- HPE Cray MPI overview
- Multi-NIC support
- MPI Collectives
- MPI-3 RMA
- Recommendations

Cray MPICH Software Stack on Slingshot (Before LUMI-C Upgrade)

CH4 Device



HPE Cray MPI Software Architecture



Key MPI Changes for the HPE Cray EX

- Launcher Flexibility
  - Cray PALs launcher (mpiexec)
  - standard Slurm (srun)
- Kernel Feature Set Flexibility
  - Support multiple OS distros (CentOS/SLES/etc)
  - Run-time query for features, use most optimal
  - XPMEM, CMA, hugepages, GPUs
- Interconnect Flexibility
  - Supports the libfabric API
    - Verbs+rxm provider for Mellanox NICs; Cassini provider for Slingshot NIC
  - Supports the UCX API
    - UCX driver for Mellanox NICs
  - Cray MPI runs on Slingshot and Infiniband clusters

HPE Cray MPI for Slingshot Features and Optimizations

- Based on ANL MPICH CH4 device
- Enhanced libfabric verbs+rxm provider w/ XRC over RoCE for Slingshot 10
- Cassini provider for Slingshot 11
- XPMEM and CMA for on-node single-copy transfers
- Small-msg on-node collective optimizations
- Optimizations for select collective operations
- Multiple NICs per node
- Multithreading performance enhancements
- Support for hugepage memory allocations
- Scalable Cray PMI implementation interfaces with:
  - Slurm
  - Parallel Application Launch Service (PALS)
- Flexible, intuitive rank re-ordering feature
- MPI I/O performance enhancements and stats
- MPICH ABI-compatible
  - Compatible with Intel MPI, MPICH and MVAPICH libraries
  - Allows ISV apps to “transparently” use Cray MPI on HPE systems

Multiple NIC Support (1)

- Each MPI rank is assigned to use a single NIC
  - Cray MPI does not support striping data across multiple NICs

- MPICH_OFI_NIC_POLICY - Selects the rank-to-NIC assignment policy used by Cray MPI
  - BLOCK (default)
    - Use a block distribution. Consecutive local ranks on a node are equally distributed among the available NICs on the node
  - NUMA
    - Local ranks are assigned to the NIC that is closest to the rank's numa node affinity.
  - ROUND-ROBIN
    - The first local rank on a node is assigned to NIC 0, the second rank is assigned NIC 1, etc.
  - GPU
    - Local ranks are assigned to the NIC closest to the GPU selected by user, if multiple NICs are assigned to a NUMA node then round-robin
  - USER (custom assignment)
    - Use a custom NIC assignment as specified by MPICH_OFI_NIC_MAPPING
- MPICH_OFI_NIC_VERBOSE – Displays pertinent information related to NIC selection
  - Set to 1 for concise information
  - Set to 2 for more verbose rank-to-NIC assignment information

Multiple NIC Support (2)

- MPICH_OFI_NIC_MAPPING:
  - Relevant if policy is set to USER
  - Each rank must have a NIC mapping
    - –Assign ranks 0, 16, 32, 48 to NIC 0, remaining ranks to NIC1
    - –MPICH_OFI_NIC_MAPPING="0:0,16,32,48; 1:1-15,17-31,33-47,49-63"

- MPICH_OFI_NUM_NICS:
  - Specifies number of NICs that can be used on each node
    - –Default: Use all available NICs
    - –To limit use of "n" NICs/node, MPICH_OFI_NUM_NICS="n"
    - –To specify a specific set of NICS, MPICH_OFI_NUM_NICS="2:0,3"
      - – Assuming node has 4 NICs (0-3), this setting will use only 2 NICs, indexes 0 and 3

MPI Multi-NIC environment variables

Environment Variable	Default	Purpose
MPICH_OFI_NIC_VERBOSE	0	If set to 1, displays output during MPI_Init to identify what NICs are available to the job. NIC names, addresses, index values and numa affinity is displayed. Setting to 2 displays the specific NIC each rank has been assigned.

```
PE 0: ======================= Display NIC Addrs  =======================
PE 0: Hostname: nid000004
PE 0:   MPICH_OFI_NIC_POLICY: BLOCK
PE 0:   Number of NICs: 2
PE 0:     nic_index 0: domain_name=cxi0, numa_domain=2, addr=0x0000e5ff
PE 0:     nic_index 1: domain_name=cxi1, numa_domain=6, addr=0x000165ff
PE 0:   Number of NUMA domains: 8
PE 0:     numa_domain 0: cpu_list=[0-15,128-143]
PE 0:     numa_domain 1: cpu_list=[16-31,144-159]
PE 0:     numa_domain 2: cpu_list=[32-47,160-175]
PE 0:     numa_domain 3: cpu_list=[48-63,176-191]
PE 0:     numa_domain 4: cpu_list=[64-79,192-207]
PE 0:     numa_domain 5: cpu_list=[80-95,208-223]
PE 0:     numa_domain 6: cpu_list=[96-111,224-239]
PE 0:     numa_domain 7: cpu_list=[112-127,240-255]
PE 0: ================================================================
```

MPI Collectives

- HPE Cray MPI offers software optimizations for various collectives:
  - MPI_Allreduce, MPI_Bcast, MPI_Alltoall(v), MPI_Allgather(v), MPI_Barrier, MPI_Gatherv, MPI_Igatherv, and MPI_Scatterv
- Non-blocking collectives for communication/computation overlap (async progress thread)
- Optimizations are be enabled by default
  - Library takes account of number of ranks in calling communicator and message sizes
  - Env. variables to tune performance are documented
  - eg `MPICH_ALLGATHER_VSHORT_MSG=128`
  - When to try these: if you see sudden performance jumps with problem size
- For other collectives, Cray MPI will utilize implementations of collective operations inherited from ANL MPICH:
  - Default tuning configurations (from ANL MPICH) will be used for initial release versions
  - Customized tuning configurations to be evaluated in the future

RMA

- MPI-3 RMA operations are fully supported

- Intra-node RMA operations rely on XPMEM, CMA or POSIX shared memory
  - XPMEM optimizations are enabled by default, if available in the kernel
  - If XPMEM is not available (non-COS kernels), CMA is used automatically

- Inter-node RMA operations rely on the OFI layer and the verbs;rxm or Cassini provider for Slingshot systems
  - Efficient multi-NIC support is enabled by default on Slingshot systems with multiple NICs

- HPE Cray MPI handles different OFI completion semantics for RMA operations

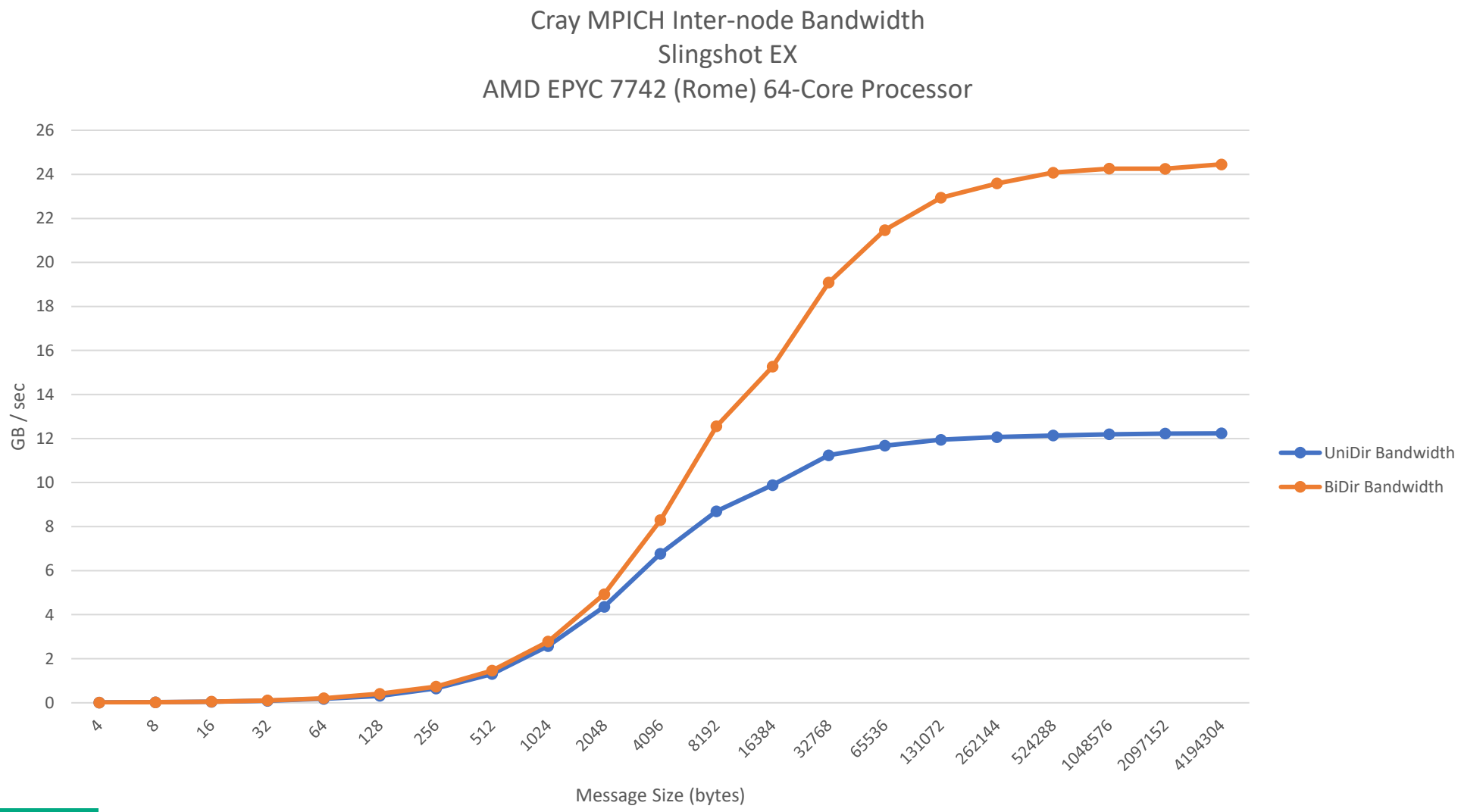
Summary of New Libfabric OFI Environment Variables

Environment Variable	Purpose
`FI_CXI_RDZV_THRESHOLD`	Specifies the transmit buffer size/inject size in bytes. Messages of size less than this will be transmitted via an eager protocol and those above will be transmitted via a rendezvous protocol. Default is 16,364
`MPICH_OFI_USE_PROVIDER`	Specifies the libfabric provider to use. By default, the "verbs;ofi_rxm" provider or “cxi” provider is selected for Slingshot systems.
`MPICH_OFI_VERBOSE`	If set, displays verbose output during MPI_Init to verify which libfabric provider was selected, along with the name and address of the NIC(s) being used. Not set by default

Summary of Libfabric OFI Environment Variables for Multi-NIC

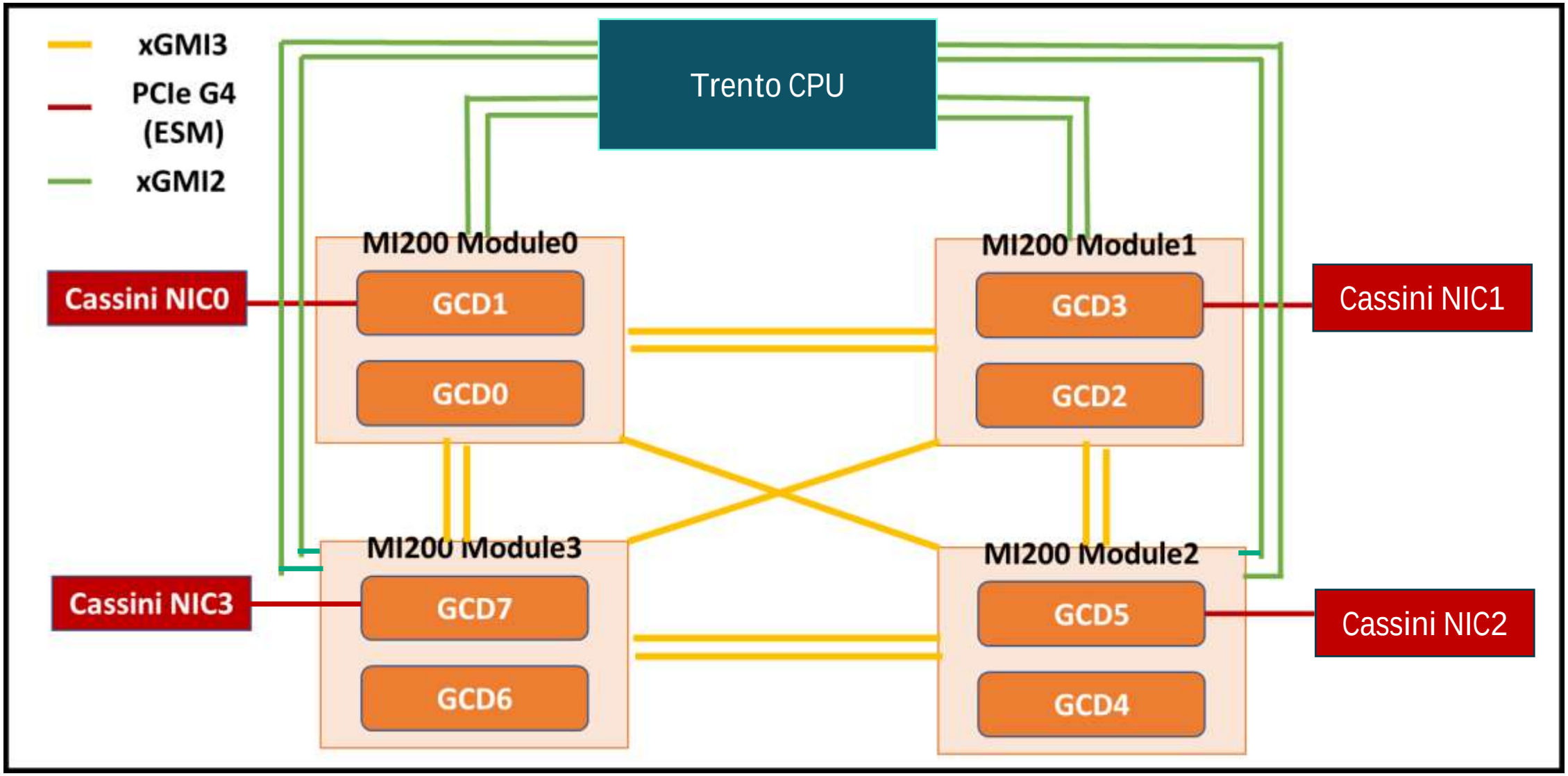
Environment Variable	Default Value	Purpose
MPICH_OFI_NIC_POLICY	Block	Selects the rank-to-NIC assignment policy used by Cray MPI. Options: BLOCK, ROUND-ROBIN, NUMA, GPU, and USER
MPICH_OFI_NIC_MAPPING	Unset	Specifies the precise rank-to-NIC mapping to use on each node.
	Block	Selects the rank-to-NIC assignment policy used by Cray MPI.
MPICH_OFI_NIC_VERBOSE	0	If set to 1, verbose information pertaining to NIC selection is printed at the start of the job.
MPICH_OFI_NUM_NICs	Unset	Specifies the number of NICs the job can use on a per-node basis. By default, when multiple NICs per node are available, MPI attempts to use them all.
MPICH_OFI_SKIP_NIC_SYMMETRY_TEST	0	If set to 1, the check for NIC symmetry (i.e. make sure all nodes in the job have the same number of Nics available) is performed during MPI_Init will be bypassed.

HPE Cray EX Slingshot Performance (1 process to 1 process)



GPU Support in HPE Cray MPI

LUMI-G GPU BLADE ARCHITECTURE (LUMI-G)



GPU support in Cray MPICH

- Mi250X GPU offers a significant amount of high-performance memory (128GB)
  - Furthermore each GPU is ”attached” to a corresponding NIC
- Applications can boost up their performance by deploying "GPU Aware" MPI communications
  - Applications that perform MPI operations with communication buffers that are on GPU-attached memory regions
  - In other words: use GPU pointers in the MPI calls

  ➔ Strongly suggested!
- Cray MPI offers "GPU Aware" MPI support with the following technologies:
  - GPU-NIC RDMA (for inter-node MPI transfers)
  - GPU Peer2Peer IPC (for intra-node MPI transfers)
- Available in `PrgEnv-amd`, `PrgEnv-cray`, and `PrgEnv-gnu`
- Set `MPICH_GPU_SUPPORT_ENABLED=1` to enable GPU support
  - It will crash/hang if the variable is not set (or set to 0) and the parallel application uses communication buffers that are on GPU-attached memory regions for MPI communications
- Check `mpi` man page, search for the environment variables with suffix `MPICH_GPU_` for more info

New MPI Environment Variables for GPU Support

Environment Variable	Default Value	Purpose
MPICH_GPU_SUPPORT_ENABLED	0	Enables a parallel application to performs MPI operations with communication buffers that are on GPU-attached memory regions.
MPICH_GPU_IPC_ENABLED	1*	Enables GPU IPC support for intra-node GPU-GPU communication operations.
MPICH_GPU_EAGER_REGISTER_HOST_MEM	1*	Registers the CPU-attached shared memory regions with the GPU runtime layers.
MPICH_GPU_IPC_THRESHOLD	8192	Intra-node GPU-GPU transfers with payloads of size greater than or equal to this value will use the IPC capability. Transfers with smaller payloads will use CPU-attached shared memory regions.
MPICH_GPU_NO_ASYNC_MEMCPY	1	Enables optimization for intra-node MPI transfers involving CPU and GPU buffers. If set to 0, it reverts to using blocking memcpy operations for intra-node MPI transfers involving CPU and GPU buffers.
MPICH_GPU_COLL_STAGING_AREA_OPT	0	Enables experimental optimization for collective operations (e.g. MPI_ALLreduce) involving GPU-GPU transfers with large payloads

Example: OSU benchmark

- Benchmark for several MPI operations, including GPU-aware operations
  - https://mvapich.cse.ohio-state.edu/benchmarks/
  - Enable rocm tests (`--enable-rocm` flag in configure)
- Example of the p2p/osu_bw, 2 MPI tasks on 2 node each

`./p2p_osu_bw H H`

```
# OSU MPI Bandwidth Test v5.9
# Size      Bandwidth (MB/s)
1                       2.04
2                       4.08
4                       8.15
8                      16.13
16                     31.27
32                     65.56
64                    132.73
128                   268.91
256                   497.55
512                   993.99
1024                 1984.37
2048                 3784.15
4096                 7874.46
8192                12849.28
16384               17380.11
32768               17833.26
65536               20208.35
131072              21203.71
262144              21694.38
524288              21806.31
1048576             22058.25
2097152             22083.57
4194304             22153.19
```

`./p2p_osu_bw D D`

```
# OSU MPI-ROCM Bandwidth Test v5.9
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
1                       2.07
2                       4.16
4                       8.32
8                      16.57
16                     32.59
32                     68.18
64                    136.40
128                   271.23
256                   501.66
512                   999.26
1024                 1893.28
2048                 3753.52
4096                 7929.02
8192                15869.25
16384               20331.70
32768               21164.55
65536               22623.41
131072              23210.06
262144              23577.66
524288              23794.47
1048576             23886.72
2097152             23932.34
4194304             23954.98
```

GPU-aware MPI summary

Steps for enabling GPU-aware MPI:

- Load the module `craype-accel-amd-gfx90a`
- Make sure you do the linking via the compiler wrappers (ftn, cc, CC)
  - If you don't do that you will get a runtime error message:
    ```
    MPIDI_CRAY_init: GPU_SUPPORT_ENABLED is requested, but GTL library is not linked
    ```
  - Can still force the linking of the GTL library without the compiler wrappers:
    ```
    ${PE_MPICH_GTL_DIR_amd_gfx90a} ${PE_MPICH_GTL_LIBS_amd_gfx90a}
    ```
  - Use the `ldd` command to check the linked libraries of you application executable
- Set the env variable MPICH_GPU_SUPPORT_ENABLED=1

Tuning/Workarounds We Have Found Useful so far

- For codes with MPI_Alltoallv:
  - For sparsely-populated data, try: `export MPICH_ALLTOALLV_THROTTLE=<value>`
    `-<value>` can (should) be larger than the number of MPI ranks
  - For densely-populated data, you may want to reduce below the default value of 8

MPI Errors

If a code fails in MPI

- The MPI error handler will abort the program by default and print a stacktrace with an error message
- You will probably need to submit a support query/ticket to have these investigated:
  - Need all error messages, not just a snippet, jobid, module list etc.
  - Can help if you use srun –label if it is not obvious which rank is reporting an error.

Some common points to be aware of

- An error with PMI symbols in the stack is most likely a startup failure and a system problem.
- An error during message progress can be a consequence of another rank failing (segv, coredump etc.) so it is important to find the first error

Rank Reordering in MPI

Rank Placement

- Rank placement can have a big effect on load balance
  - dictates to which core a given PE image (MPI rank) is mapped
  - at application launch

- There is a default "SMP-style" mapping pattern

- The ordering can be changed at runtime
  - using an env. var.
  - `export MPICH_RANK_REORDER_METHOD=<value>`

- There are four values
  - 0: Round-robin placement
  - 1: SMP-style placement (default)
  - 2: Folded rank placement
  - 3: Custom ordering

Rank Placement

Start with a list of nodes to run on

0: Round-robin placement
- Sequential ranks are allocated one per node in sequence
- Placement starts again on first node if we reach the last node

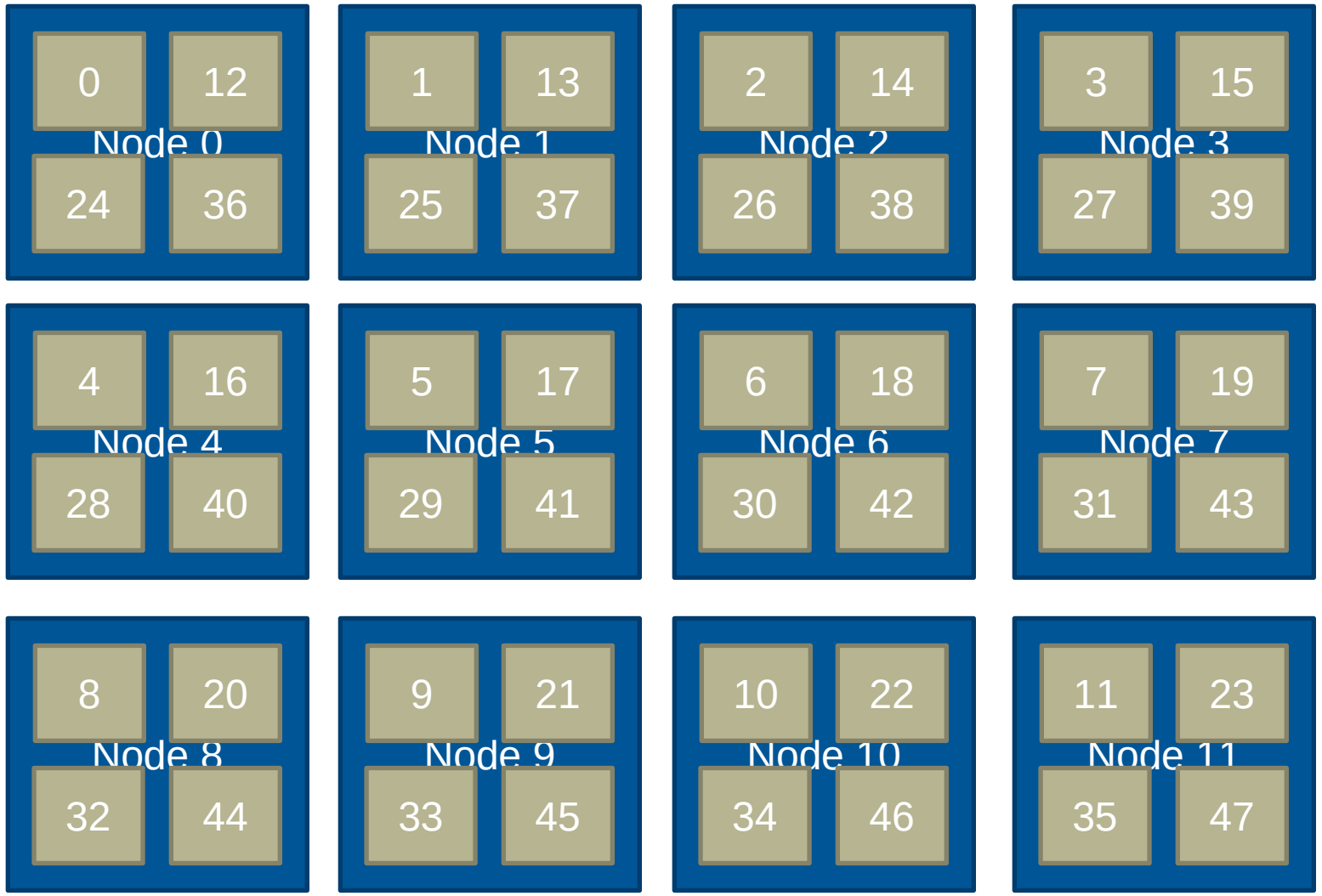
1: SMP-style placement (default)
- Sequential ranks fill up each node in turn
- Only then move on to the next node

2: Folded rank placement
- Similar to round-robin placement
- except each pass over node list is in the opposite direction

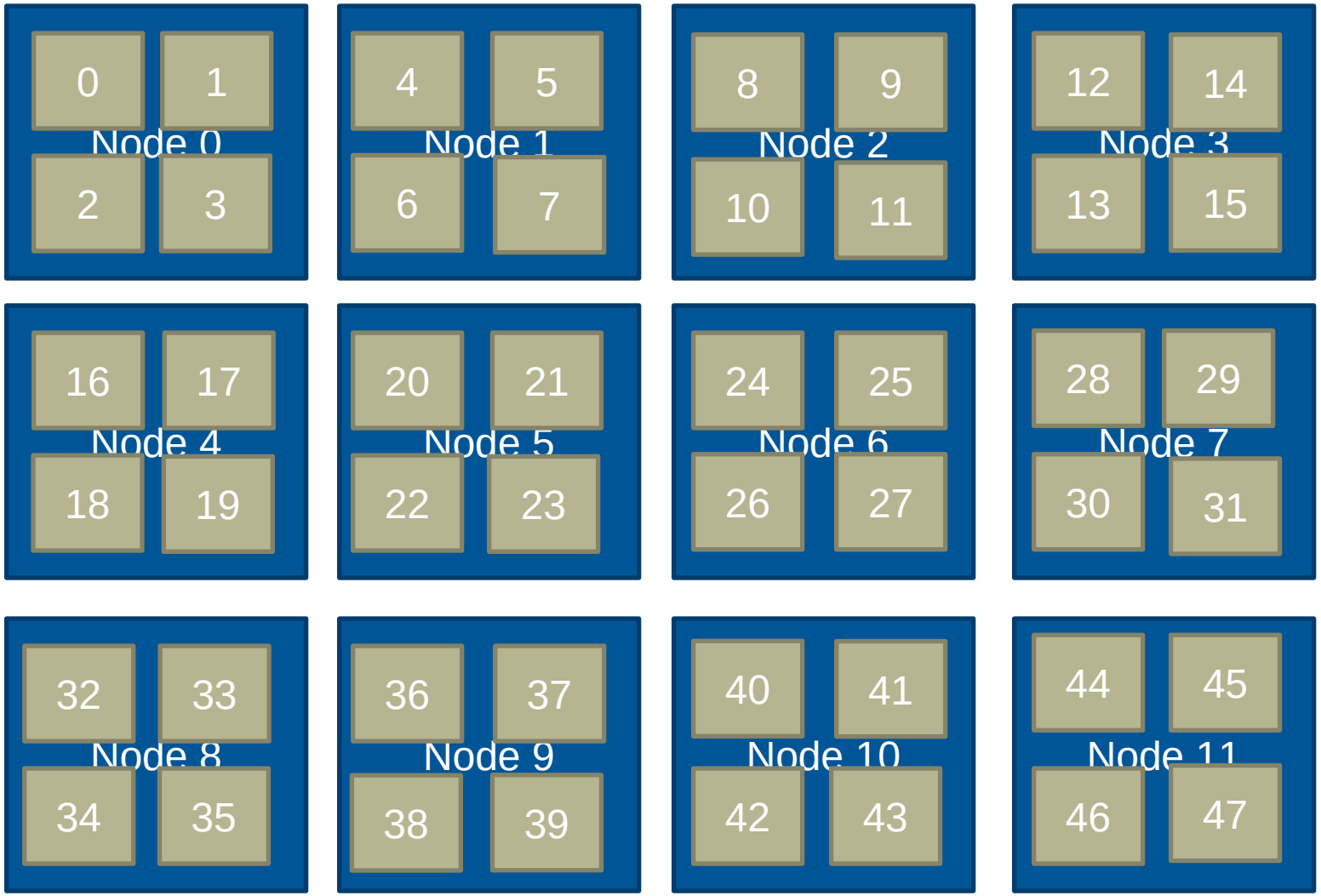
3: Custom ordering
- The location of each rank in turn is specified in a list

- Examples of these are shown on the next slide
  - For a simplified example of four cores per node

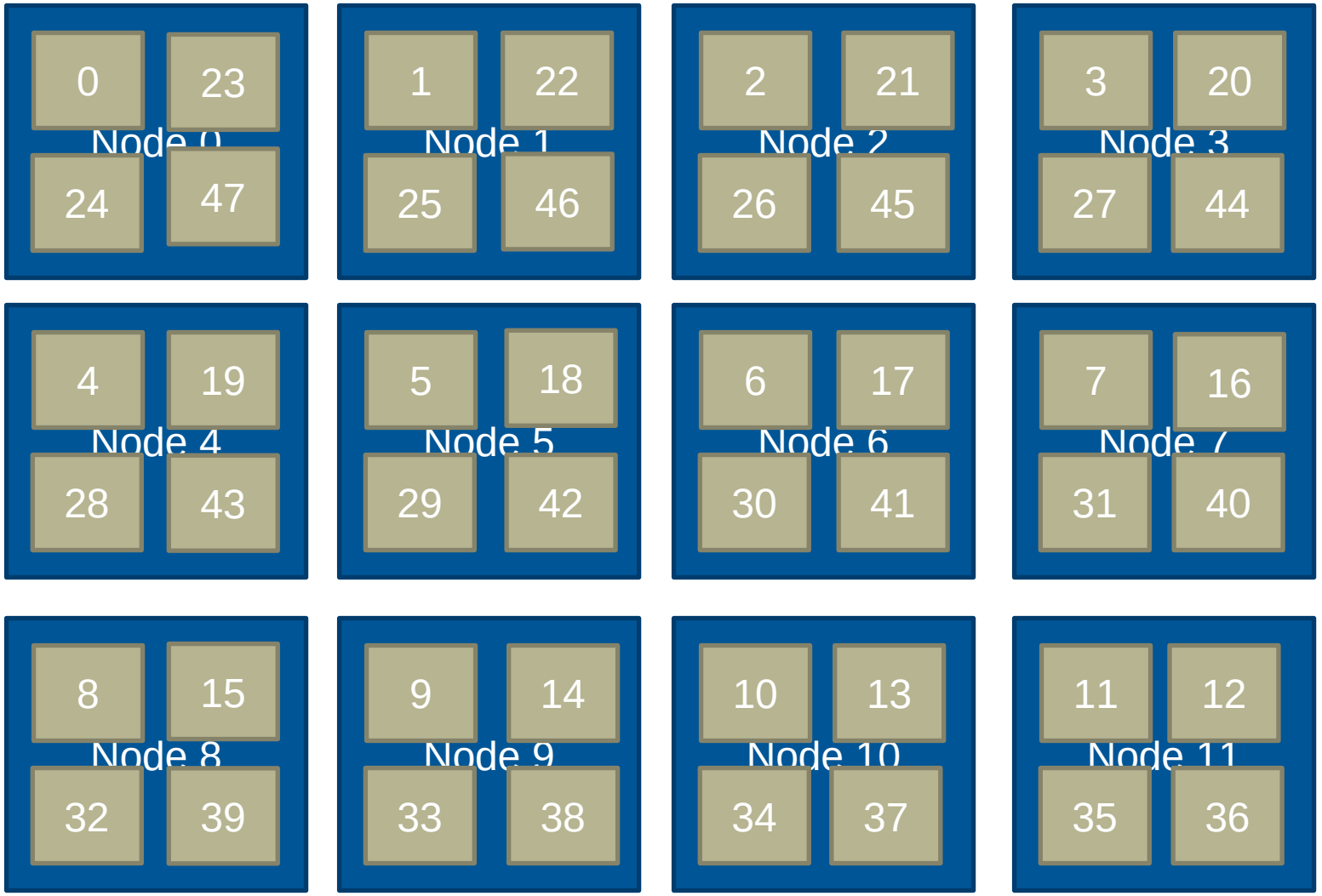
0: Round Robin Placement



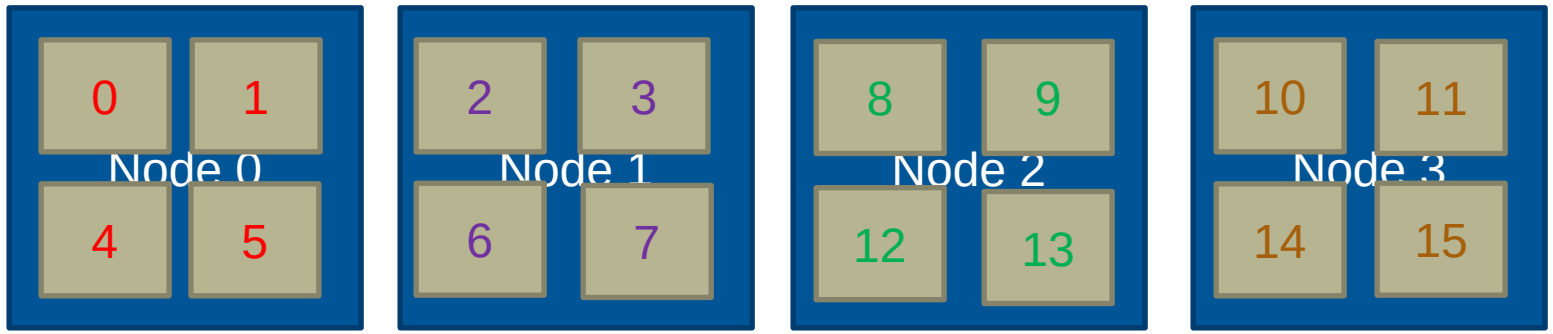
1: SMP Placement



2: Folded Placement



3: Custom Example

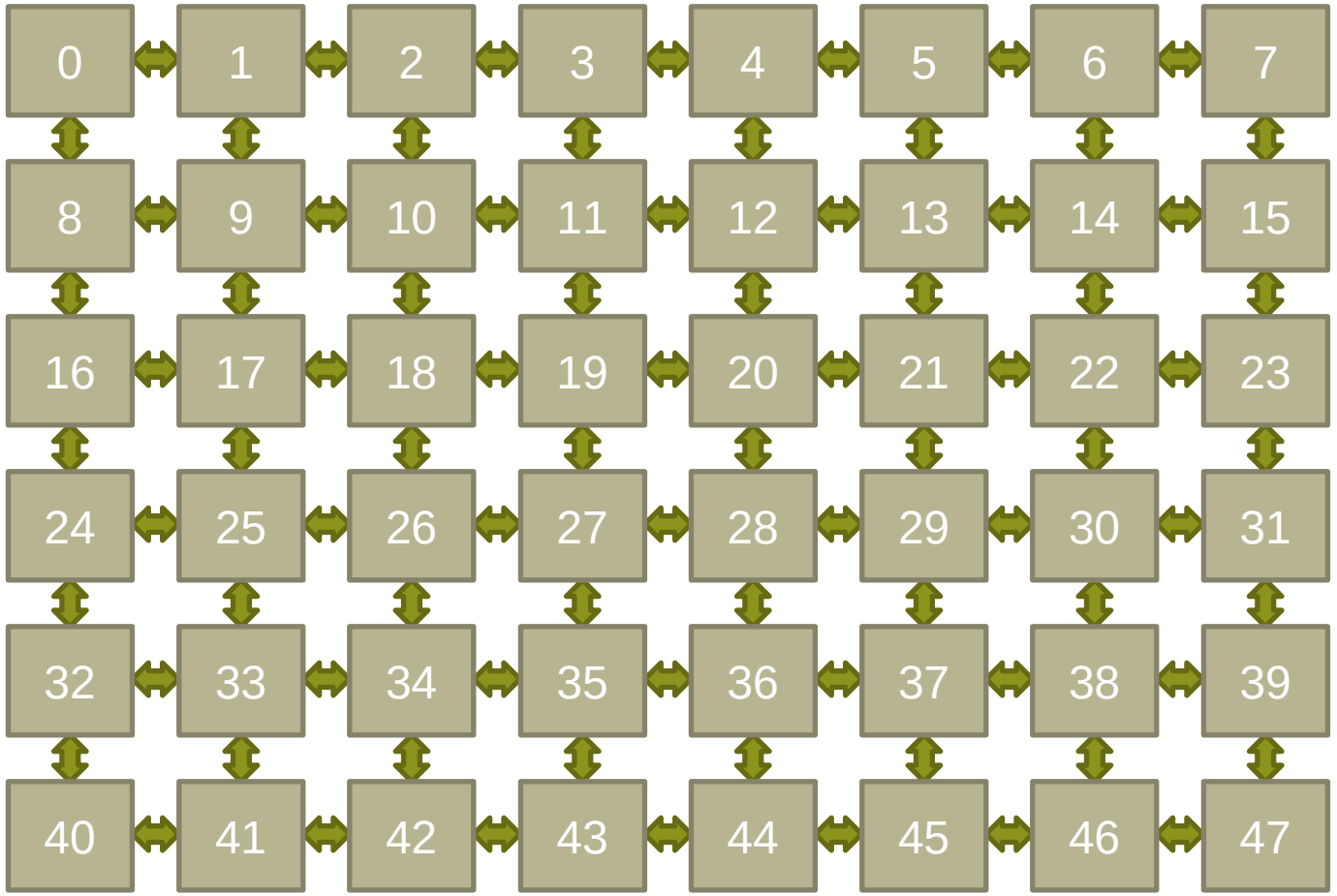



```
MPICH_RANK_ORDER: 0,1,4,5,2,3,6,7,8,9,12,13,10,11,14,15
```

- MPICH_RANK_REORDER=3 enables this
- Ordering comes from file MPICH_RANK_ORDER
  - comma separated ordered list
    - can optionally be condensed into hyphenated ranges
  - all ranks should be included in the list once and only once
- Nodes are filled up SMP-style
  - but not with sequential rank numbers
  - instead, take ranks sequentially from the MPICH_RANK_ORDER list

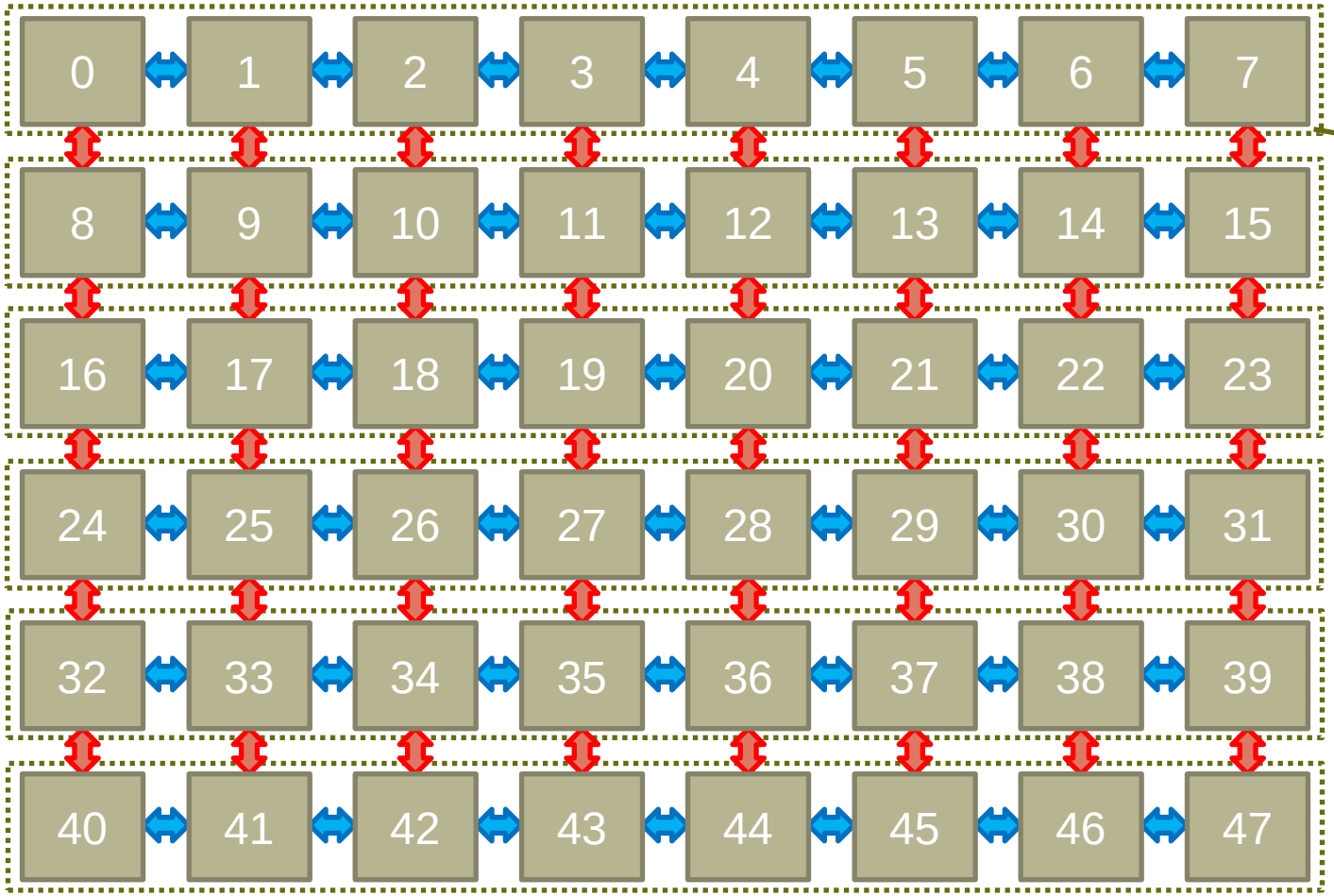
```
MPICH_RANK_ORDER: 0,1,4,5,2,3,6-9,12,13,10,11,14,15
```

Optimising 2D Boundary Swap with Custom Rank Reordering



- Each rank communicates with its N-S and E-W neighbours.

Default Rank Order: Suboptimal

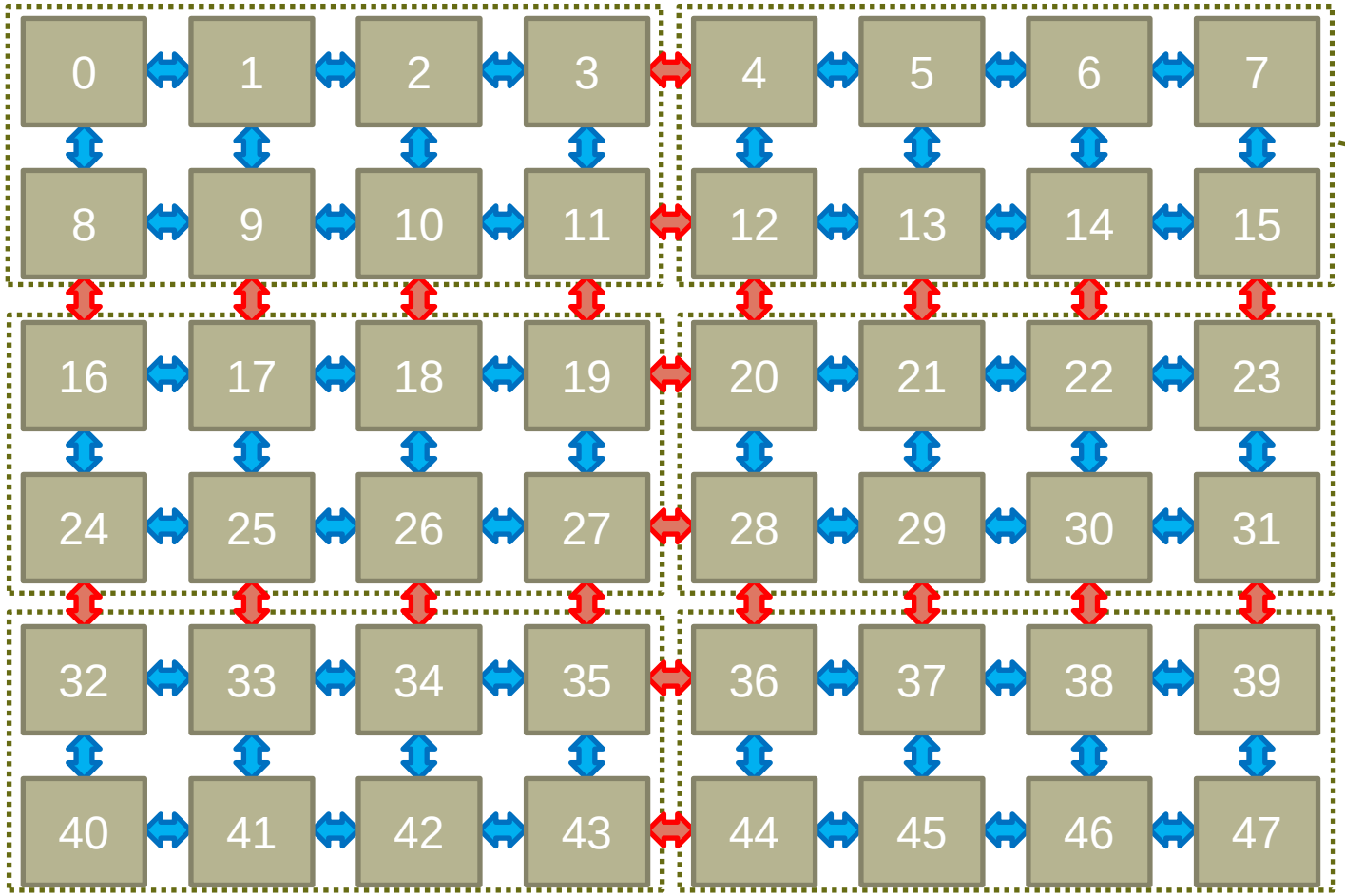


Node Boundaries with default SMP layout

Internode comms slower than Intranode comms, so reducing total number will improve communication performance

Default SMP layout: Inter:Intra ratio = 40:42

Improved Customized Order Using Sub-cells



Node Boundaries with customised layout

Internode comms reduced by reorganising into 4x2 cells.

Patterns can often be recognised by CrayPAT.

Customised ordering: Inter:Intra ratio = 22:60 (was 40:42)
Even more effective with 3D and fatter nodes.

Rank Reordering

- Easy to experiment with
  - defaults at least should be tested with every application…
  - Perftools can help generate the reorder file
  - Perftools also provides a grid_order utility to optimize nearest-neighbour communication for an MPI program operating on a distributed grid.
    - generates a rank reordering file that embeds part of a cartesian grid (within a rank) which is the local part of a global grid.

- When might rank reordering be useful?
  - If point-to-point communication consumes a significant fraction of program time and a load imbalance detected
    - e.g. for nearest-neighbour exchanges (see next slide)
  - Also shown to help for collectives (alltoall) on subcommunicators
  - Spread out I/O servers across nodes
  - If there is a good use case for exploiting the hyperthreads / SMT threads

Launching Applications in MPMD Fashion

Why MPMD?

- MPMD is an approach where we want to run multiple parallel applications (executables) at the same time
- This is different from programming logic that has different ranks accomplish different tasks
- In order to facilitate communication between these applications we require a mechanism to launch them so they
  - Form a single MPI application
  - Sharing a common MPI_COMM_WORLD

- Example1: Coupled NWP Simulation
  - Atmosphere model
  - Ocean model
  - I/O servers

MPMD in Slurm

- There are various mechanisms and approaches that can be used to launch MPMD applications in a SLURM environment

- Slurm Multiple Program launch
  `srun –multiprog`

- SLURM Heterogeneous Job Support

- Custom launch script

Slurm Multiple Program Launch

- Provide a configuration file to srun
  srun –multi-prog <conf_file>
- The configuration file is of this form:
  - Comments starting with #
  - Lines corresponding to each application that provide:
    - Task rank: A number, integer range (eg 0-3), or * for last entry
    - Executable name / pathname
    - Arguments (optional): May include %t for task number and %o for offset within task ranks (0-based)

Slurm Multi-prog Example

```
###################################################################
# multi-prog configuration file
###################################################################
       800        um
       200        nemo  -v
        20        xios
```

Slurm Heterogeneous Job Support

- This can be specified in various ways
  - As a colon-separated set of flags to sbatch
  - As a combination of batch script comments and optional colon-separated srun command.

In a batch job the SBATCH comments should be separated by

#SBATCH hetjob

Let's consider a complicated example:

- Place application A on 1 node placing 32 tasks allocating 4 cpus for OpenMP threads
- Place application B on 2 nodes placing 4 tasks and allocating 16 cpu to each OpenMP task

Hetjob Run Script

```
# <<<  normal generic SBATCH options>>>
#
# First component

#SBATCH --partition=standard
##SBATCH --qos=standard
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --cpus-per-task=4
#SBATCH --hint=nomultithread

# Second component

#SBATCH hetjob

#SBATCH --partition=standard
##SBATCH --qos=standard
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=16
```

Hetjob **Run Script… Continued**

```
#
# job environment setup etc.

# Don’t want this passed to srun
unset OMP_NUM_THREADS

# Launch the two groups
# subsequent components can inherit earlier options…
srun --het-group=0 --distribution=block:block \
  --export=all,OMP_NUM_THREADS=4 ./app_A : \
  --het-group=1 --export=all,OMP_NUM_THREADS=16  ./app_B
```

Custom Launch Script

- For some situations, an alternative is to run a script (wrapper) that then runs the application
- Launch the wrapper instead of the application
  - `srun … <wrapper_script>`
- The wrapper script contains logic to launch the correct application with relevant arguments if any
- The wrapper can use Slurm environment variables such as `SLURM_PROCID` to determine which task/rank it was launched for

- An example script could be along the following lines…

Custom Launch Script

```
#!/bin/ksh
# Slurm MPMD launch wrapper - HR 20201028
#
# define list of applications, PE counts and arguments
apps=(mpmd_server mpmd_client)
counts=(2 6)
args=(‘-v','')

index=0
nchunks=${#counts[*]}
count=${counts[0]}

while [ $index -lt $nchunks ]
 do
  if [ $SLURM_PROCID -lt $count ]
   then
     exec ./${apps[$index]} ${args[$index]}
   fi
  let index+=1
  let count+=${counts[$index]}
 done
```

Information on Cray MPI

- `man intro_mpi`
  - Includes documentation of environment variables for control and display
- Fabric info (libfabric and provider)
  - man fabric
  - man fi_cxi

Recap

- Cray MPICH in general
- Overlapping communication
- Environment variables for MPI
- Cray MPICH on Slingshot
- Rank Reordering
- MPMD application launch