One-sided vs. Two-sided Communication Paradigms on Relaxed Ordering Interconnects

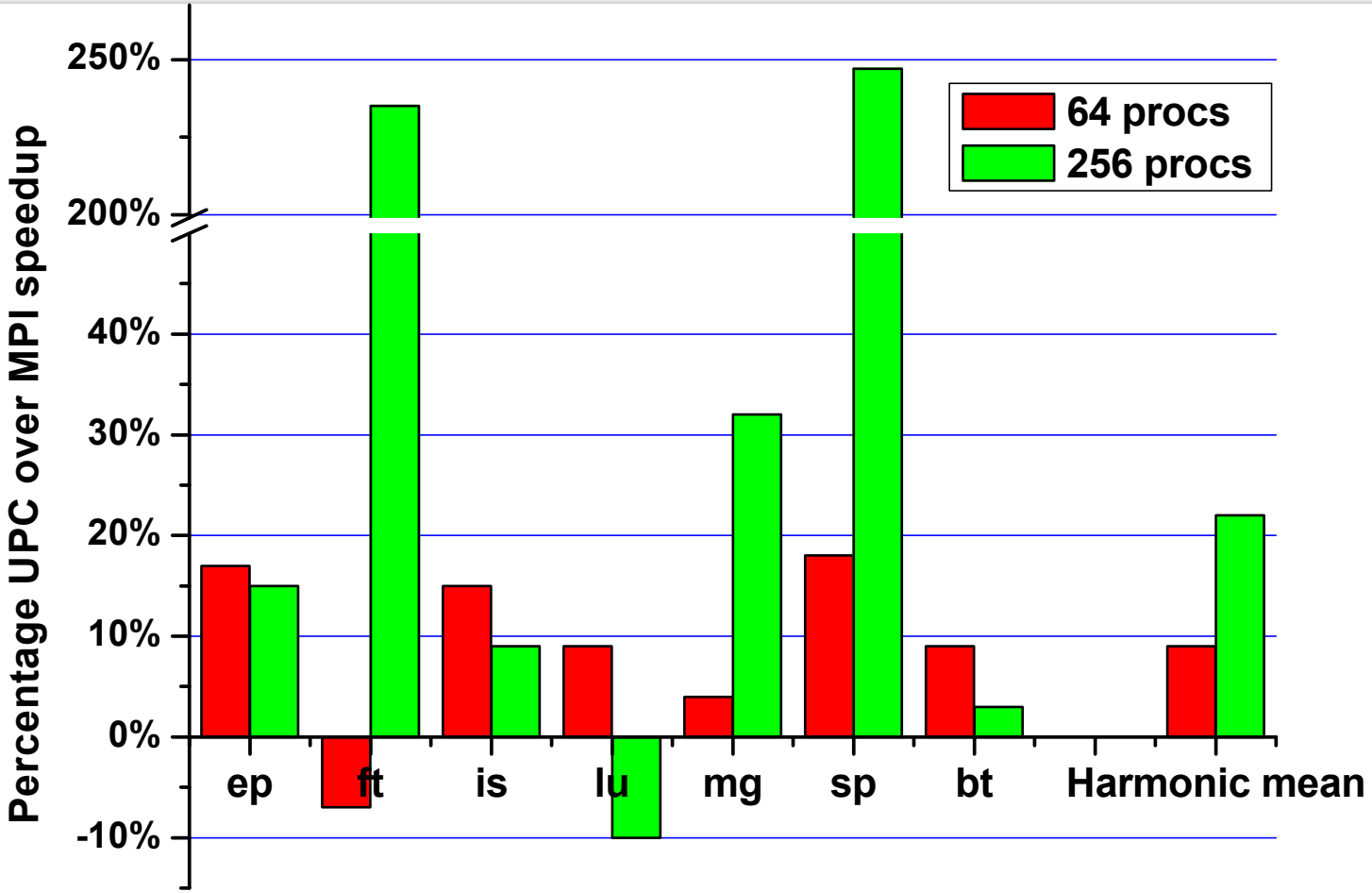
Khaled Ibrahim

Paul Hargrove, Costin Iancu, Kathy Yelick

Overview

- Application performance of one-sided vs. two-sided on Cray XE06 (Gemini Interconnect).
- Brief Overview of Cray software stack and hardware on Hopper.
  - Support of Relaxation
- Performance comparison of communication primitives using strict and relaxed ordering.
- Single-sided vs. two-sided communication paradigms interaction with relaxed ordering.

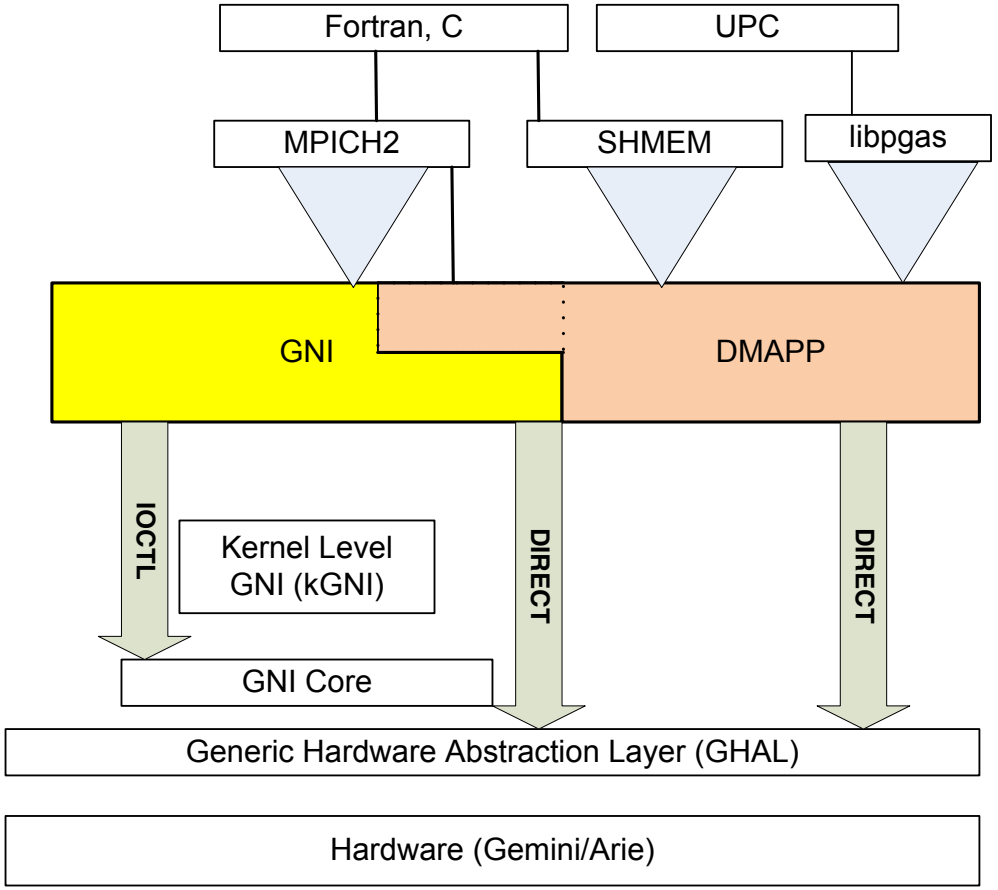
Motivation



Also, why Gasnet is not matching vendor performance?
Why single-sided performs better than two-sided MPI?

Cray Software Stack

- uGNI Multiple communication protocols
  - RDMA (*Block Transfer Engine* (BTE)).
    - Optimized for large messages
  - FMA
    - Optimized for small messages
- DMAPP communication protocol
  - High-level protocol for single-sided communication
  - support collectives



Hopper Hardware

Hopper Node
(Cray XE 6)

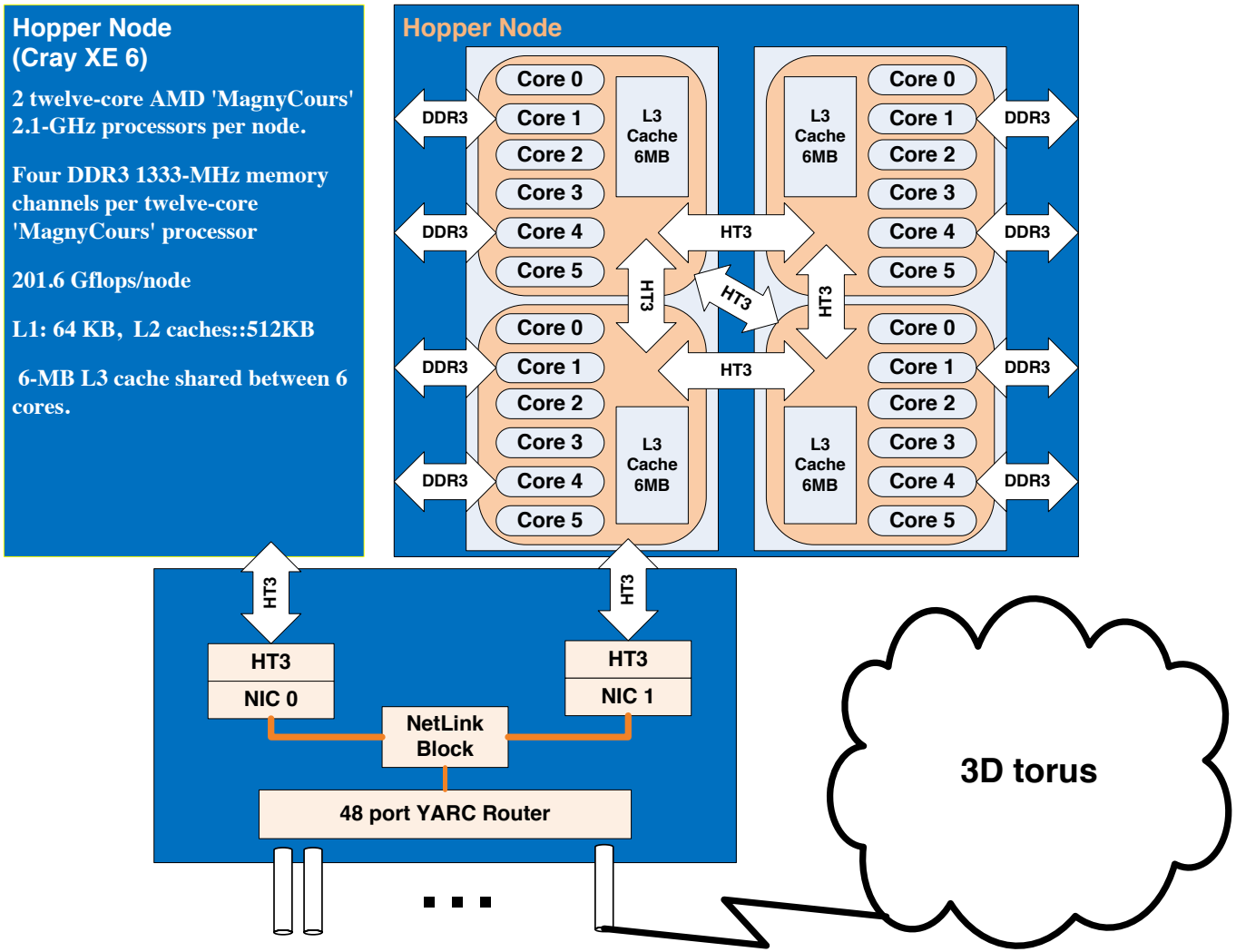
2 twelve-core AMD 'MagnyCours' 2.1-GHz processors per node.

Four DDR3 1333-MHz memory channels per twelve-core 'MagnyCours' processor

201.6 Gflops/node

L1: 64 KB, L2 caches::512KB

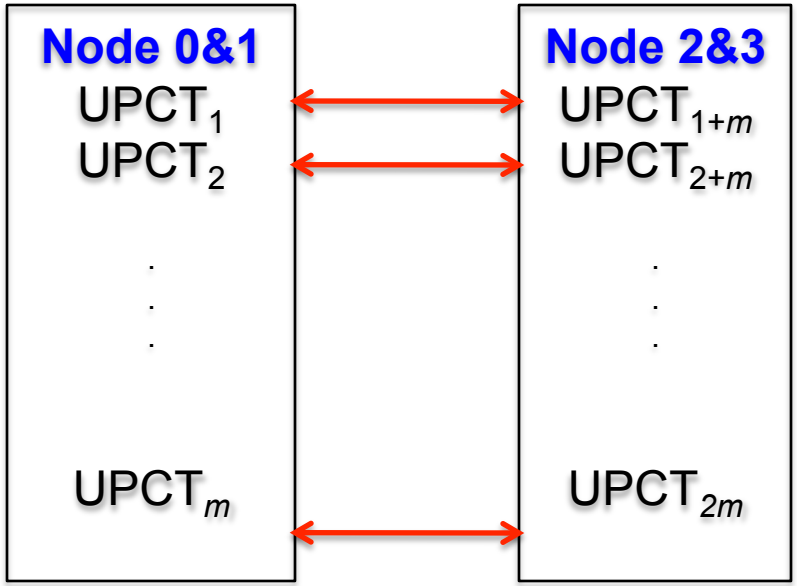
6-MB L3 cache shared between 6 cores.



Relaxation of Memory Transfers

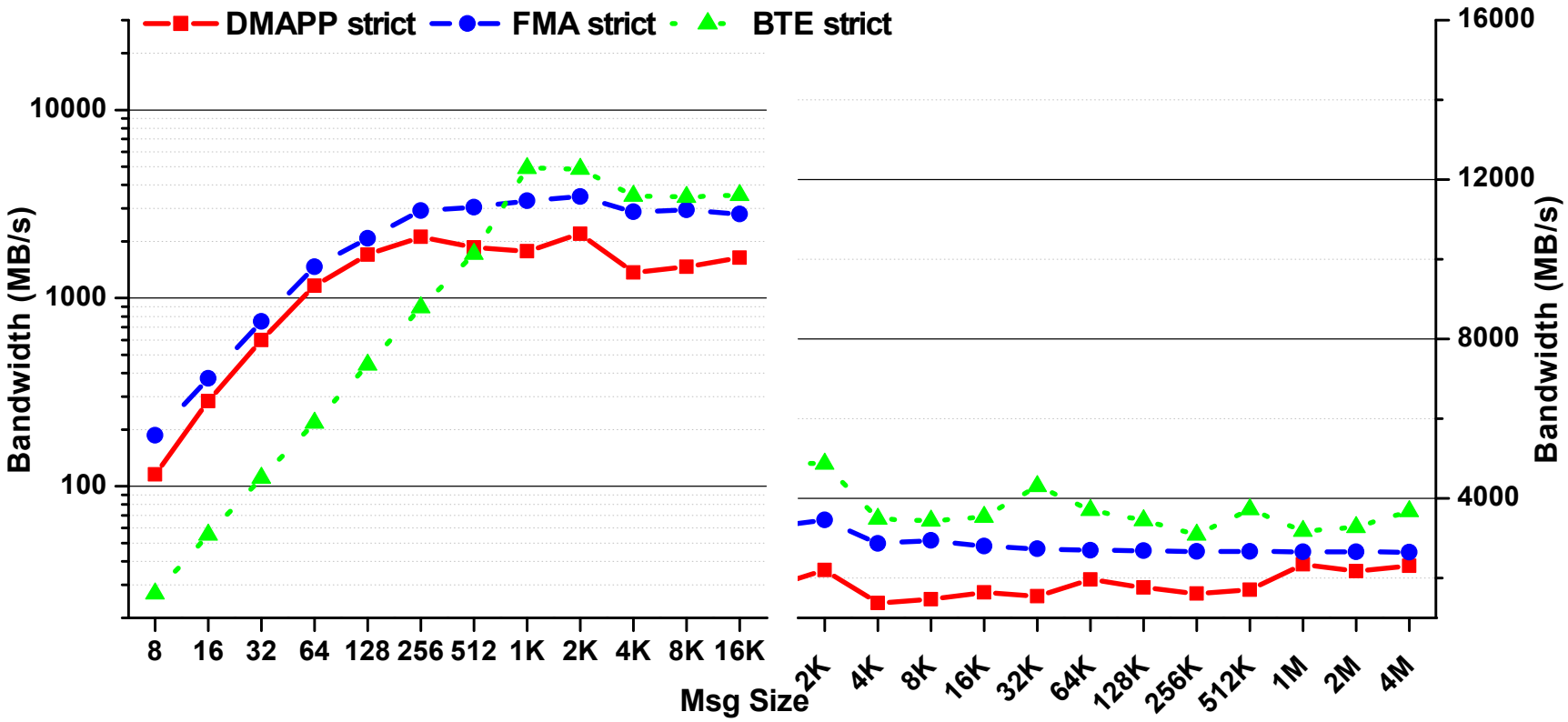
- Hypertransport transactions terms
  - Posted (writes without ack)
  - Non-Posted (reads or writes with Ack)
- Relaxation on Gemini
  - Strict (no reodering)
  - Default (non-posted get pass posted writes)
  - Relaxed (all non-posted pass posted writes)
- Relaxation affects both in-node memory transactions and remote memory transactions
- Interconnect Relaxation: How to?
  - Dynamic Routing
  - Multiple virtual channels
- Why?
  - Performance: easier way to improve throughput (BW), than to improve latency (wire delay)
  - Resilience and fault tolerance.

Microbenchmark

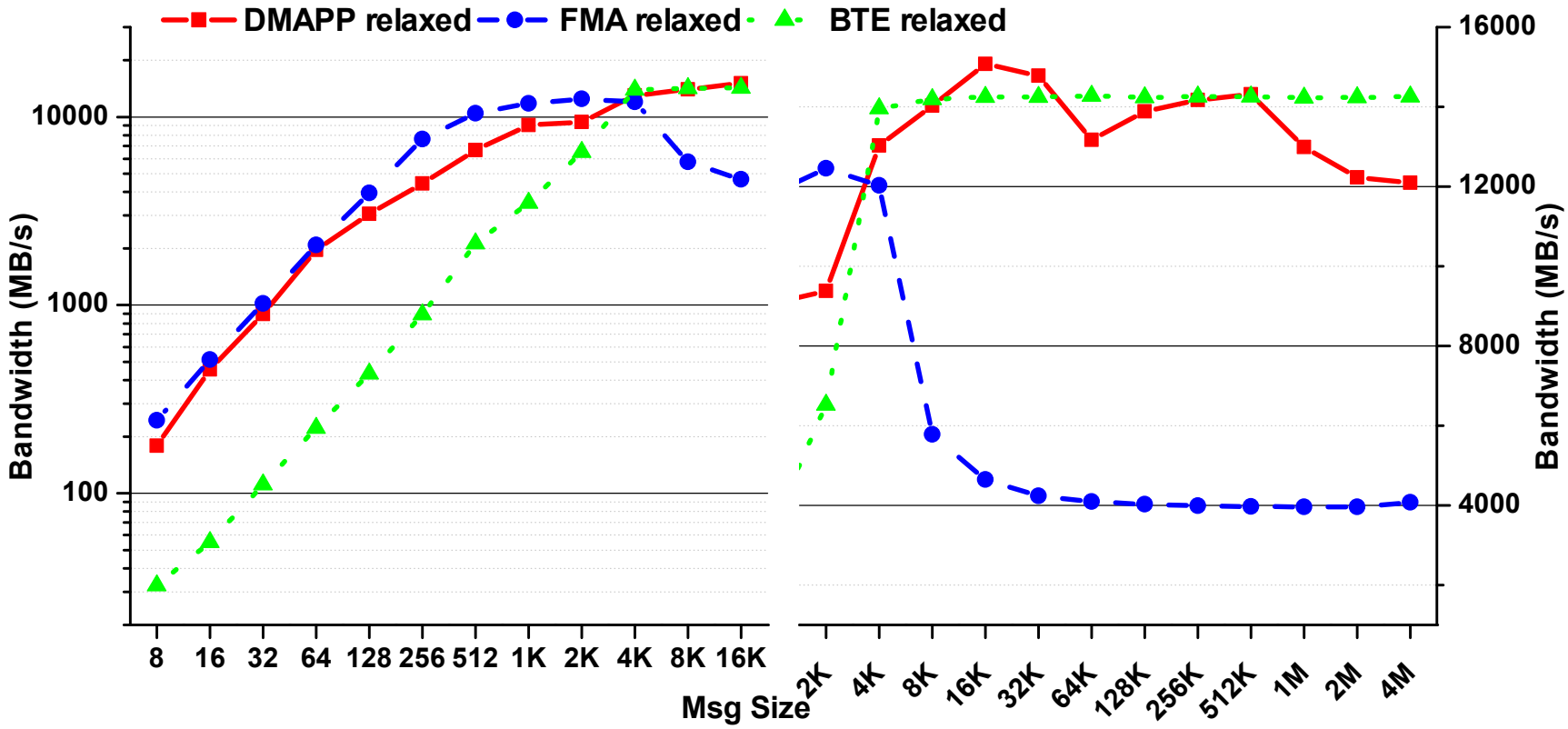


- Objective:
  - Compare the performance of low-level APIs vs. high level runtimes.
- Dialects
  - uGNI (FMA & BTE)
  - DMAPP
  - Berkeley UPC
  - Cray UPC
  - OSU MBW (MPI)
- Ranks: *m*
- Window: *w* number of outstanding non-blocking operations.
- Testbed: default
  - bidirectional
  - *48* communication pairs
  - Multiple outstanding messages (Window size)
    - Default 1

DMAPP vs. uGNI (FMA and BTE) - strict

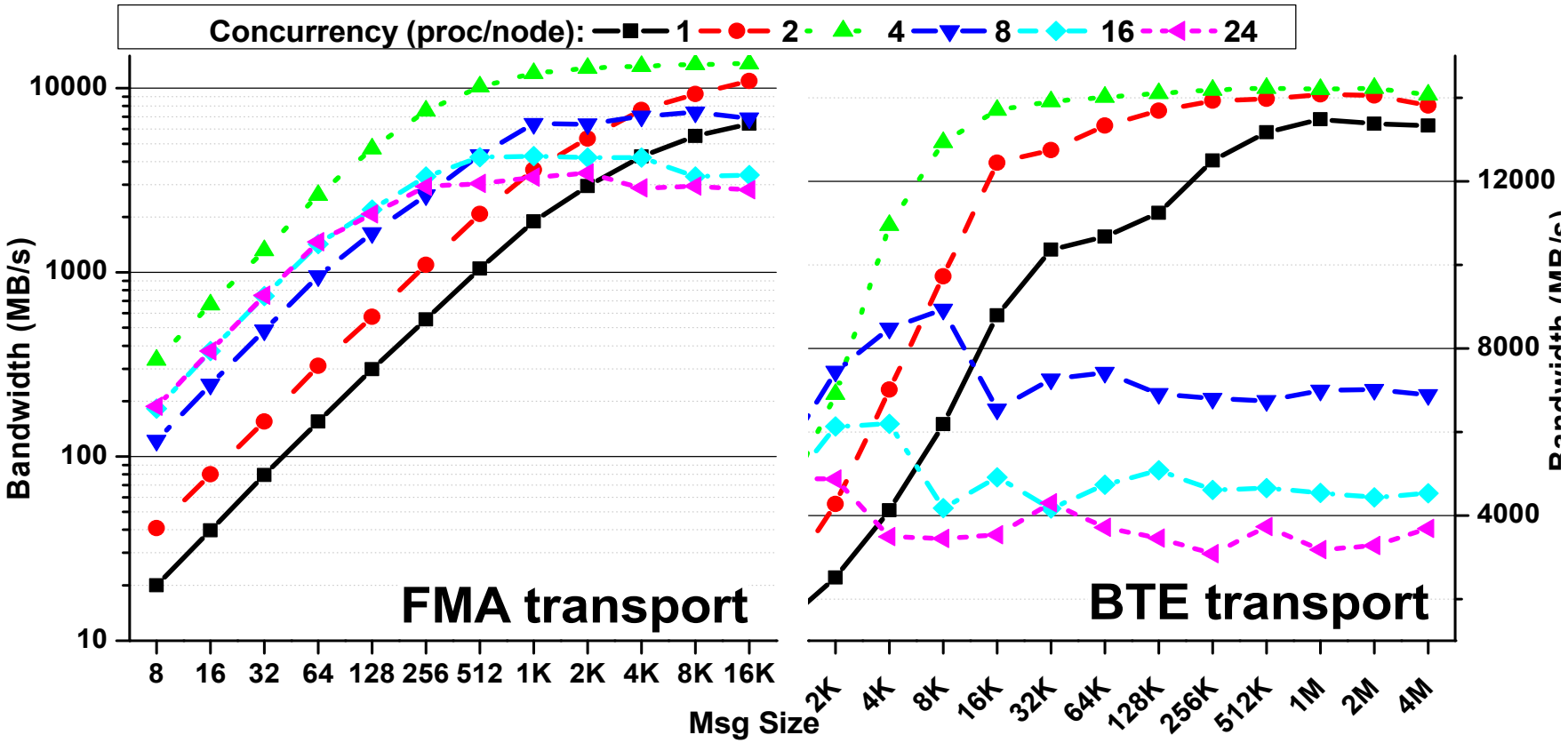


DMAPP vs. uGNI (FMA and BTE)- Relaxed

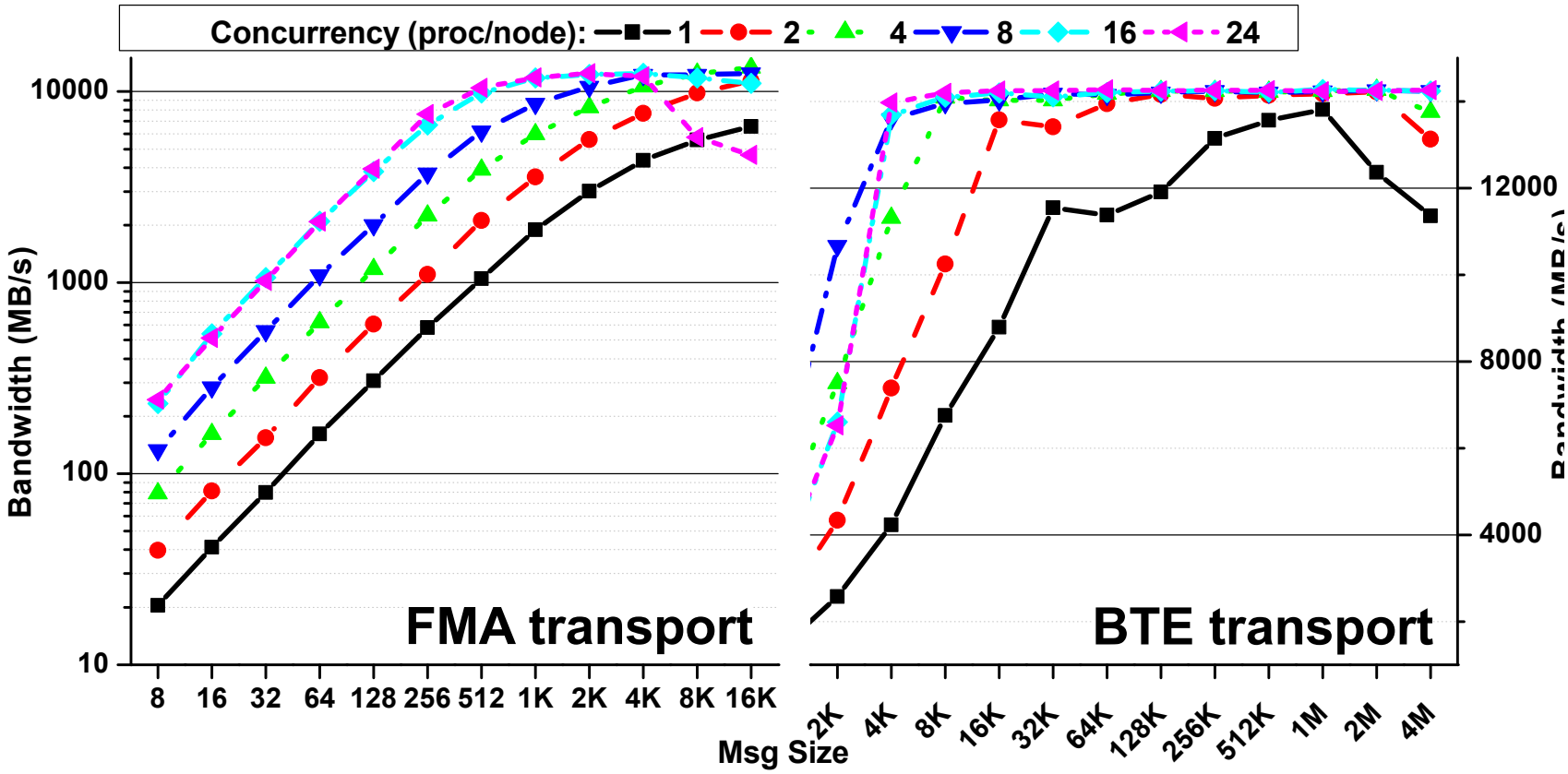


DMAPP can hide complexity of uGNI (FMA and BTE)

Strict Ordering and Concurrency

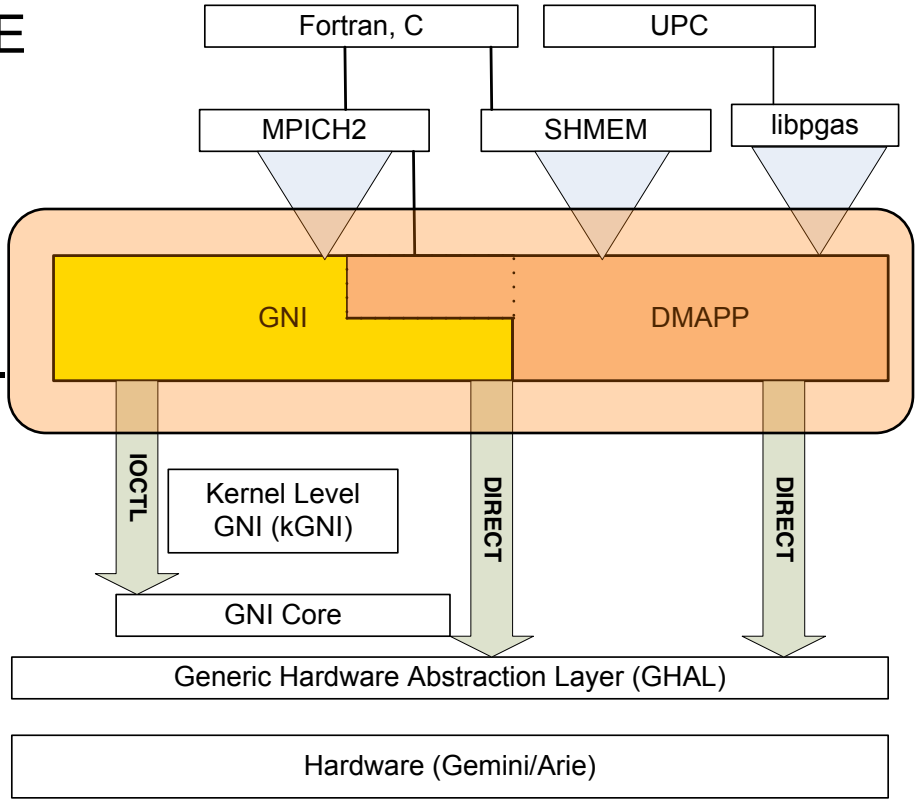


Relaxed Ordering and Concurrency



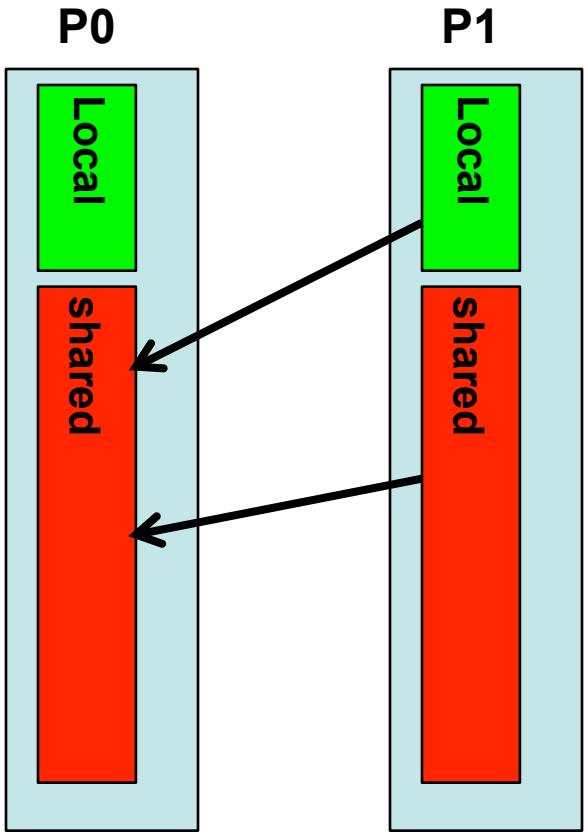
Low-Level API Summary

- DMAPP hides switching complexity between FMA and BTE
  - It also provides collectives
- Relaxed ordering improves performance (expected).
- Strict ordering hurts performance MOSTLY under high concurrency.
- Registration is an expensive memory operations that better be removed from critical path of execution.
- Performance Difference between One-sided and Two sided is NOT due to different Performance of low-level APIs.



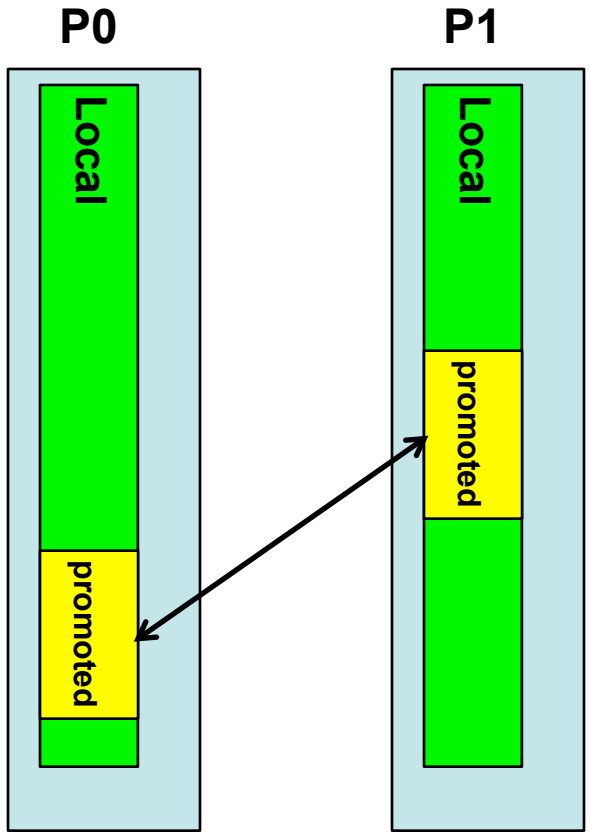
Communication Paradigms

One-sided UPC



put/get operations

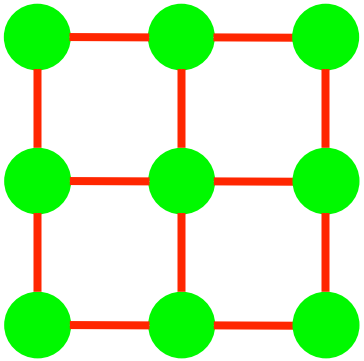
Two-sided MPI



send/recv operations

Traditional

- Shared Memory - Coherent Caches
  - One-sided load/store
  - Default to relaxed ordering
    - Strict for synchronization

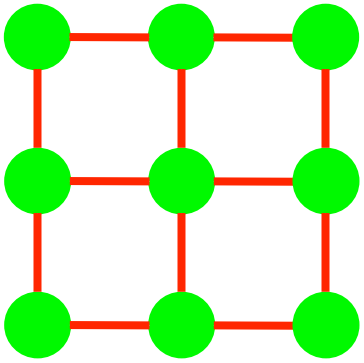


- Message Passing - no coherence
  - Send/recv matching
  - Strict matching

Communication Paradigm Territory

Traditional

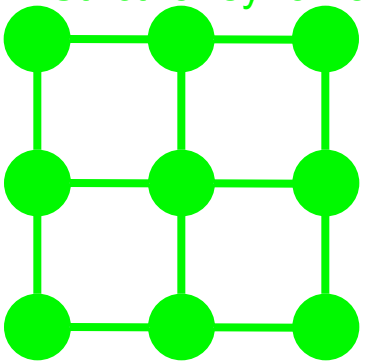
- Shared Memory - Coherent Caches
  - One-sided load/store
  - Default to relaxed ordering
    - Strict for synchronization



- Message Passing - no coherence between nodes
  - Load/store matching
  - Strict matching

PGAS

- Shared Memory - Coherent Caches
  - One-sided load/store
  - Default to relaxed ordering
    - Strict for synchronization

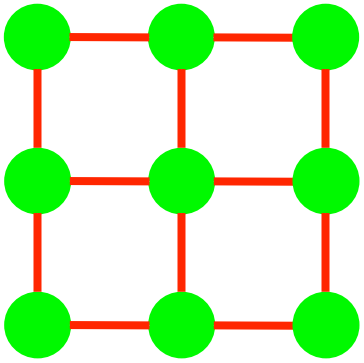


- Partition Global Address space
  - One-sided put/get
    - strict ordering if blocking
    - Relaxed if non-blocking.

Communication Paradigm Territory

Traditional

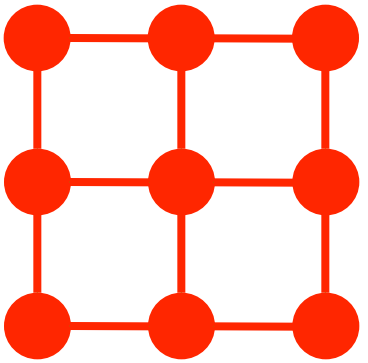
- Shared Memory - Coherent Caches
  - One-sided load/store
  - Default to relaxed ordering
    - Strict for synchronization



- Message Passing - no coherence between nodes
  - Send/recv matching
  - Strict matching

MPI+MPI

- Shared Memory - Coherent Caches
  - Send/recv matching
  - Shared memory bypass



- Message passing
  - Send/recv Matching
  - Matching strict, progress is not

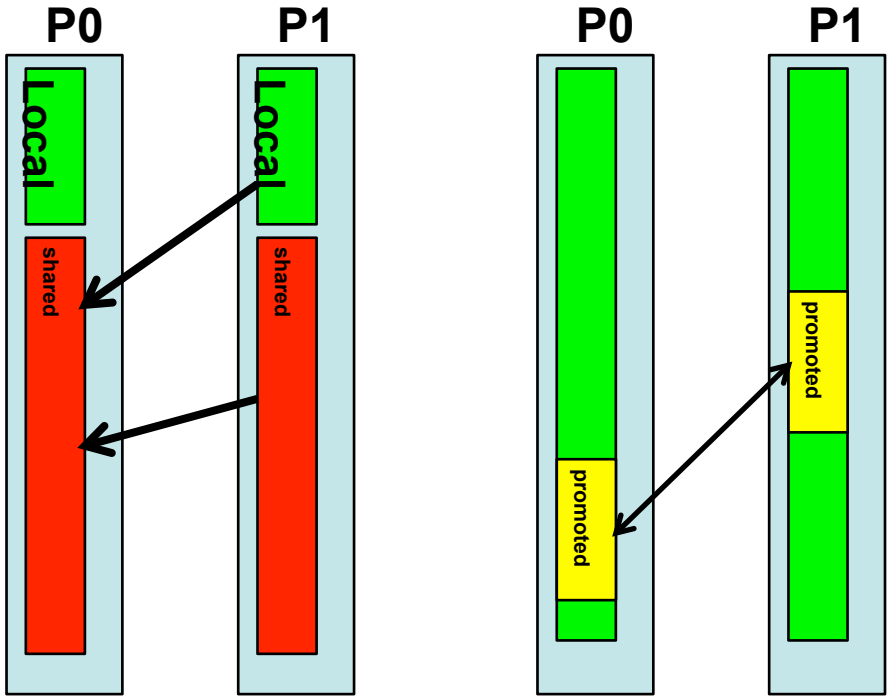
Relaxed Ordering and Communication Paradigm

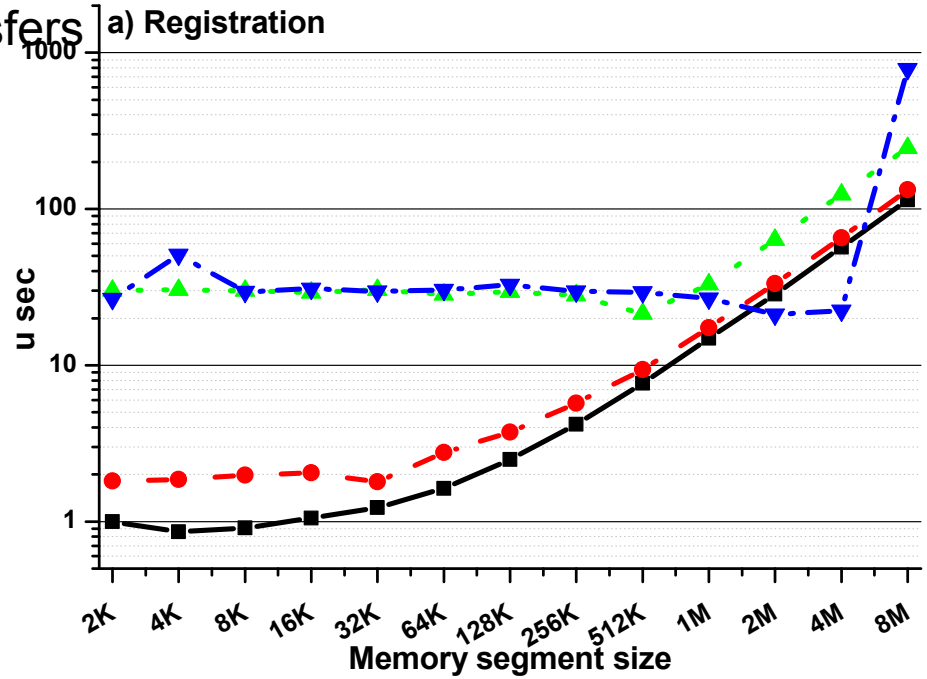
- Requirement to service multiple outstanding requests
  - Minimal remote-end involvement
  - Unambiguous destination
- HPC runtime expectation from Device Driver APIs
  - point-to-point ordering
    - Node-to-node ordering
    - Rank-to-rank ordering (multi-core makes it different from node-to-node)
  - Or, relaxed ordering, with multiple completion semantic (local and global)

Unambiguous Destination (Registration)

Registration (Resolving remote ambiguity):

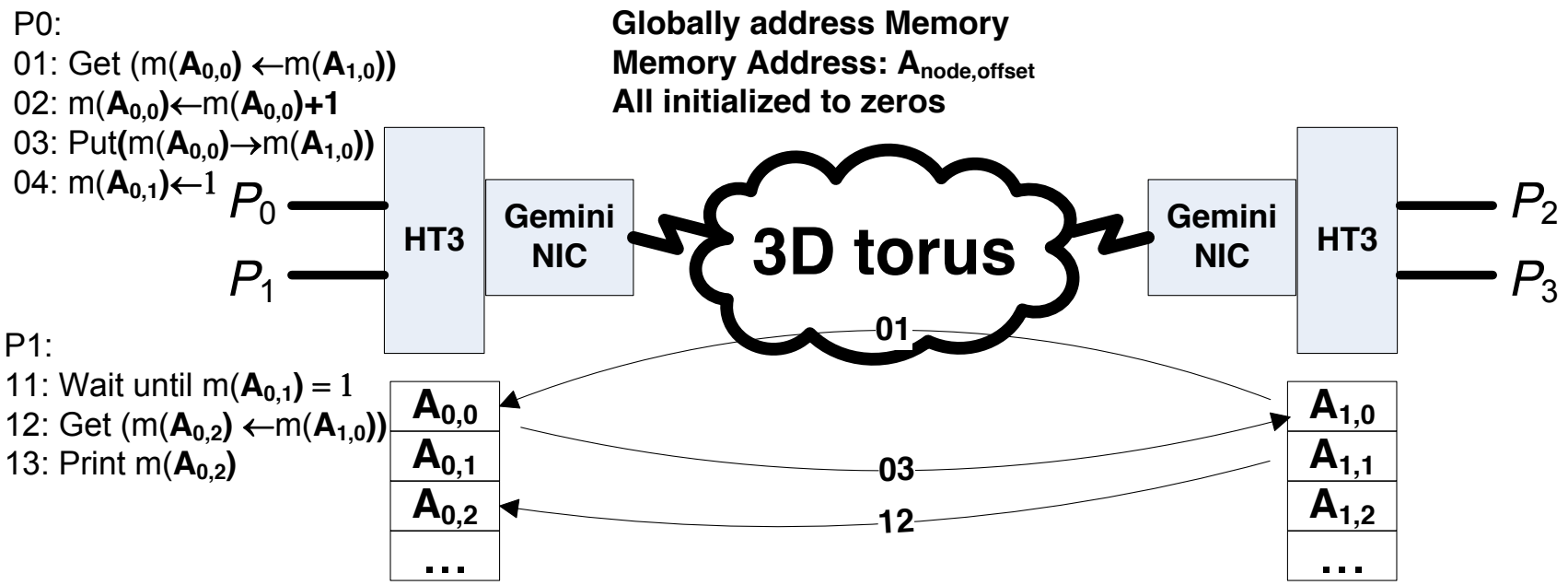
- Prevents memory swapping
- Allows offloading communication to NIC
- Allows out-of-order progress of many transfers
- Expensive (hopefully not frequent)
- Limited and shared resources





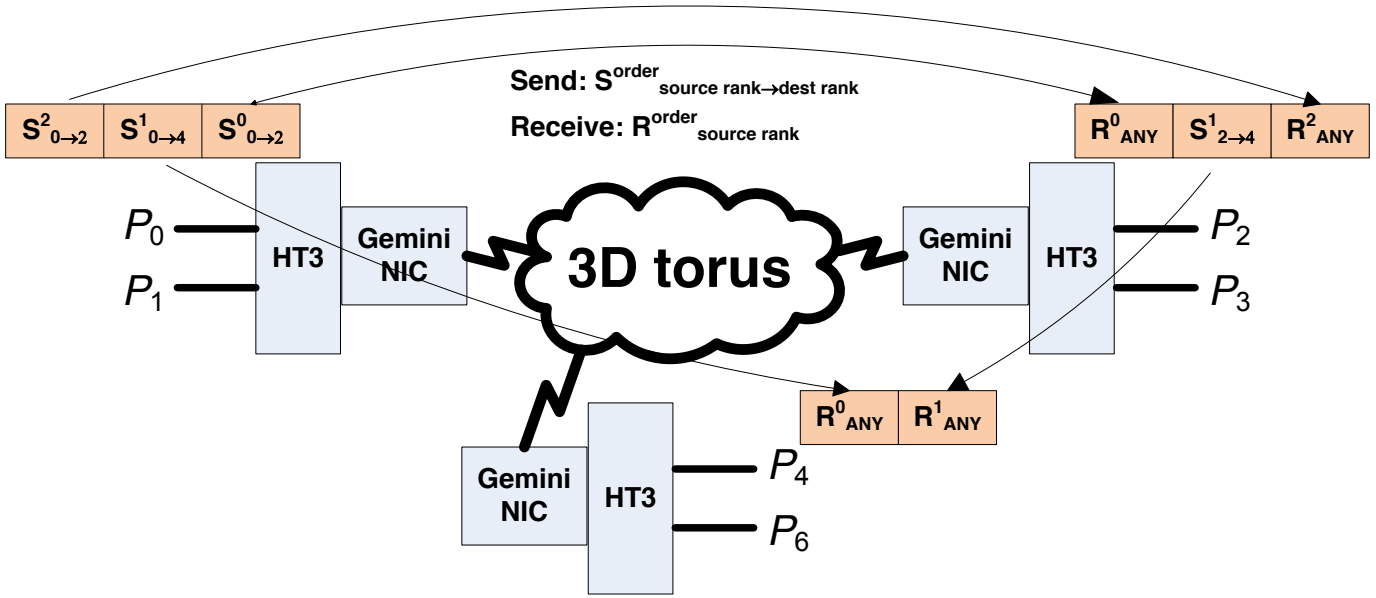
UPC Shared Memory Ordering Challenge-Example

- Cray definition of "Global Completion"
  - GASnet **Heisenbug**!



MPI Ordering

- Non-overtake rule (determinism)
  - If a sender sends two messages (Message 1 and Message 2) in succession to the same destination, and both match the same receive, the receive operation will receive Message 1 before Message 2.
  - If a receiver posts two receives (Receive 1 and Receive 2), in succession, and both are looking for the same message, Receive 1 will receive the message before Receive 2.



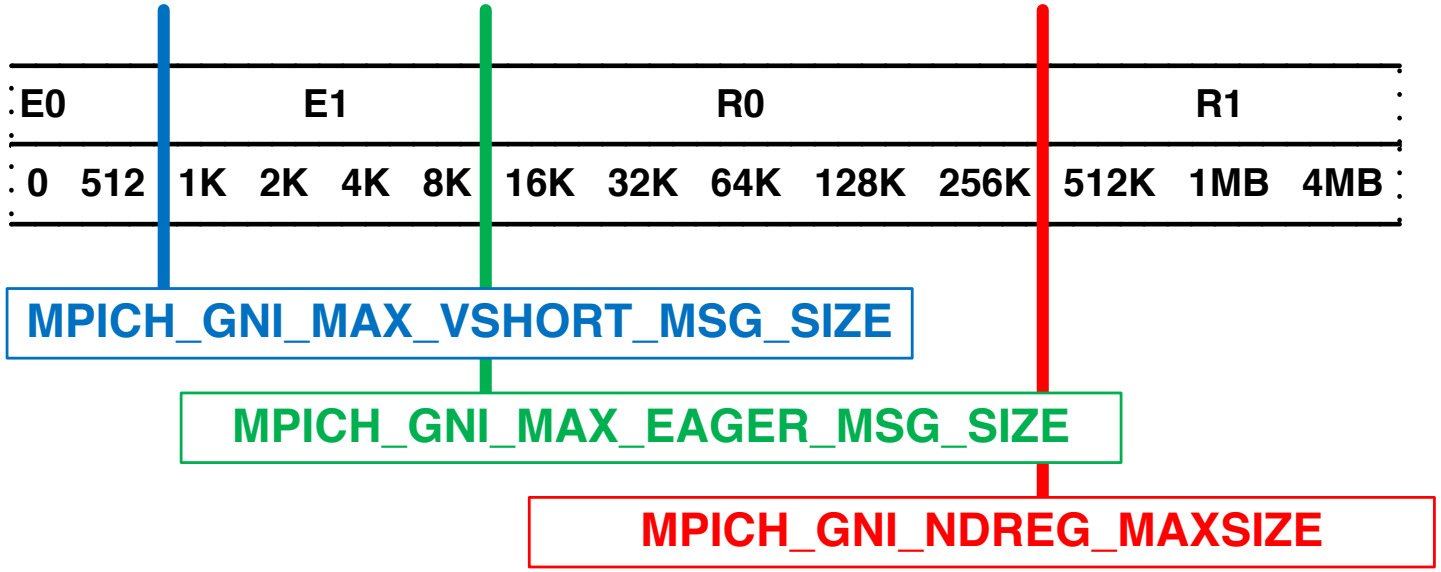
Cray MPI Method to Exploit Gemini Relaxed Ordering

Typical eager vs. rendezvous

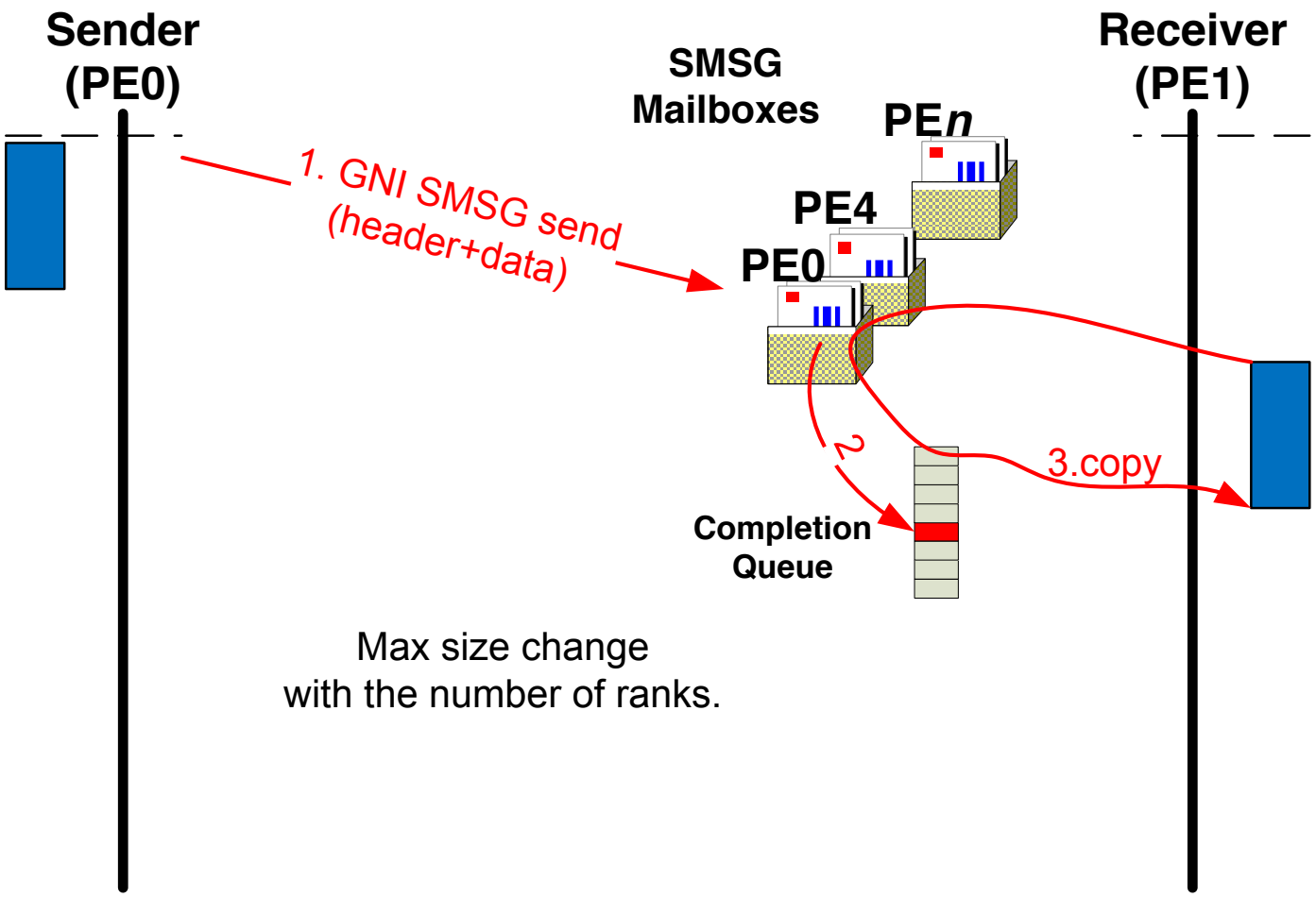
Cray's

- Two eager modes
- Two rendezvous mode
- Switching is controlled based on Message size
- Registration cache is used

(may not be able to tell which memory will be used for communication)

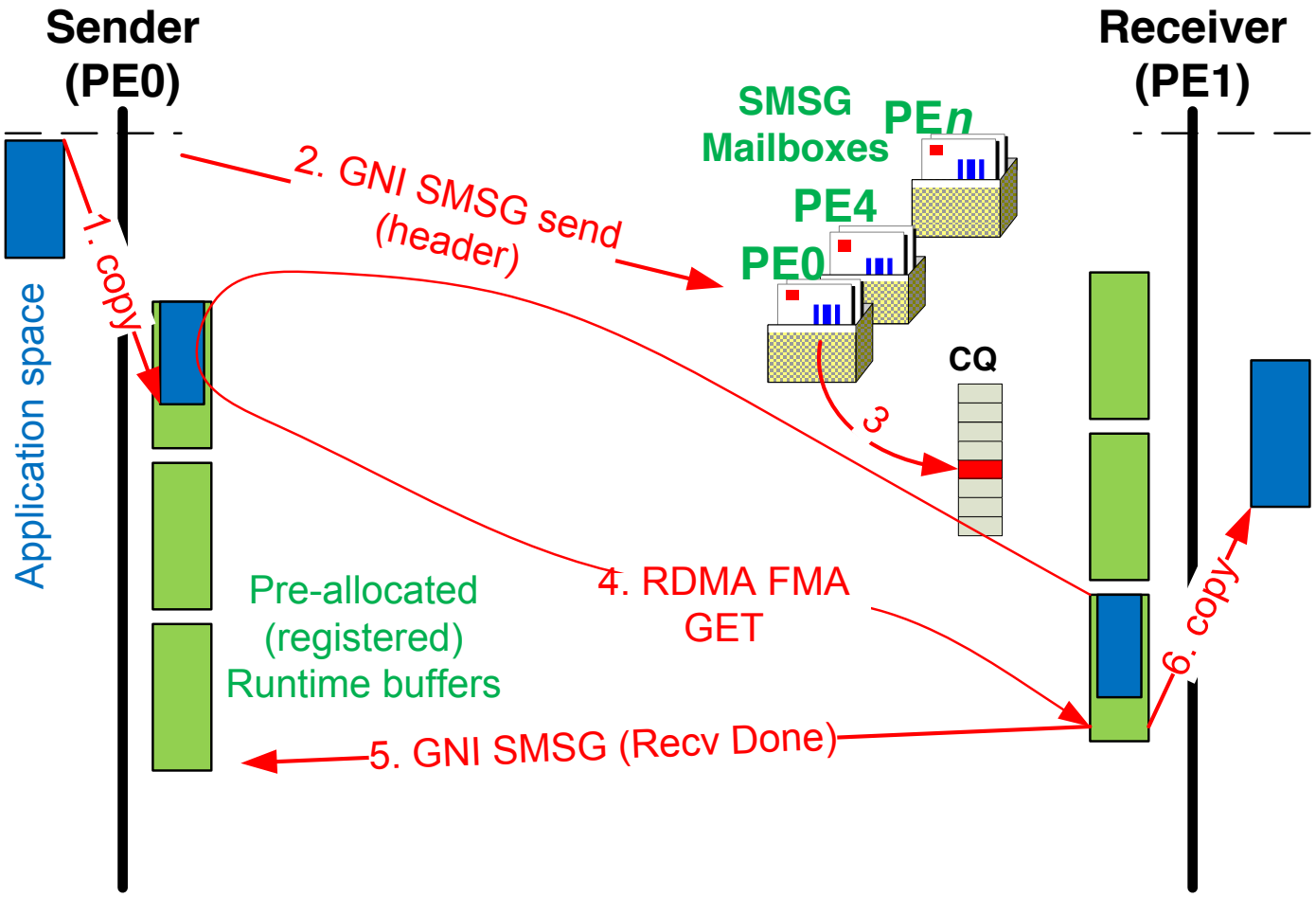


MPI Eager 0

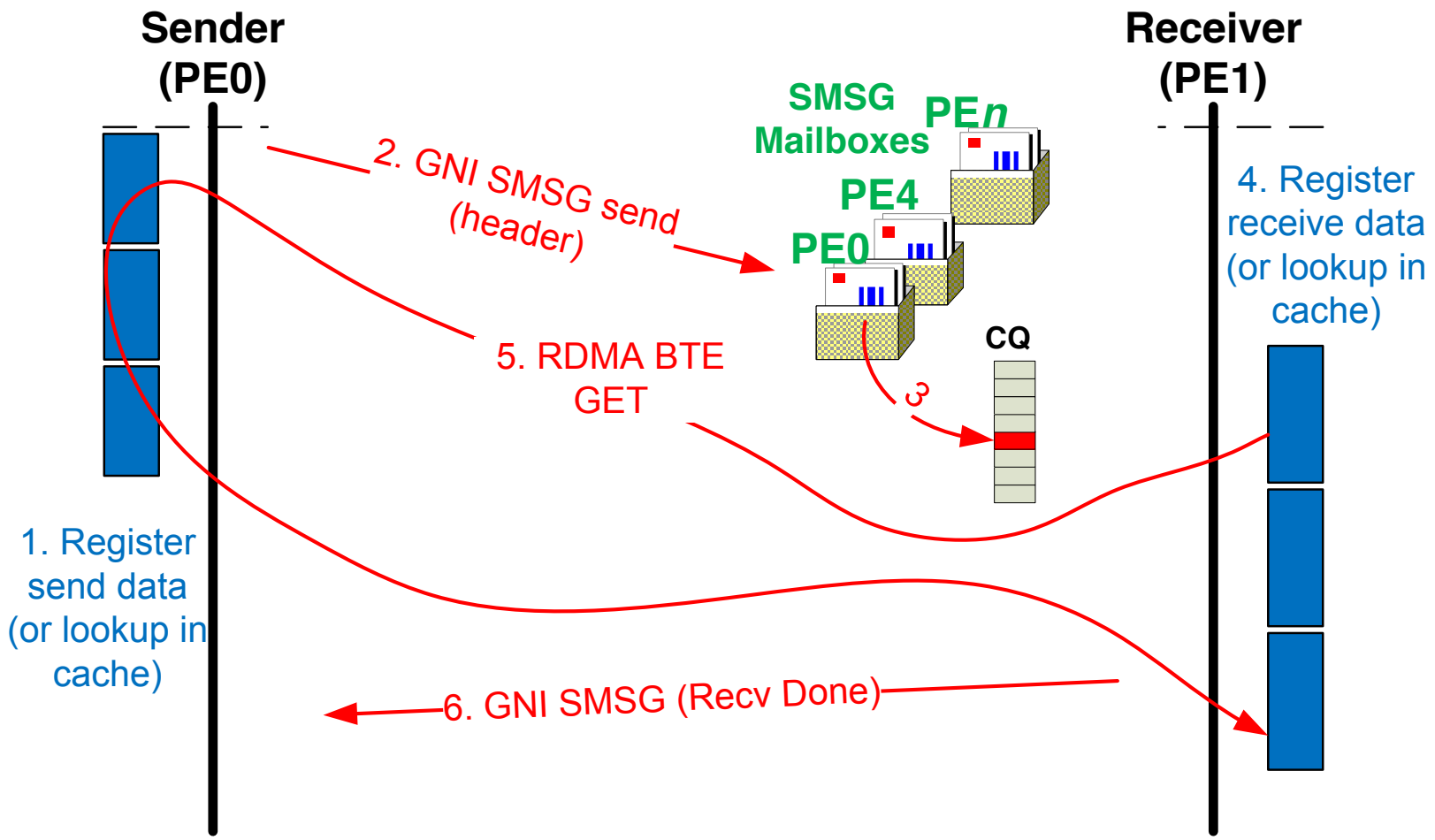


Max size change
with the number of ranks.

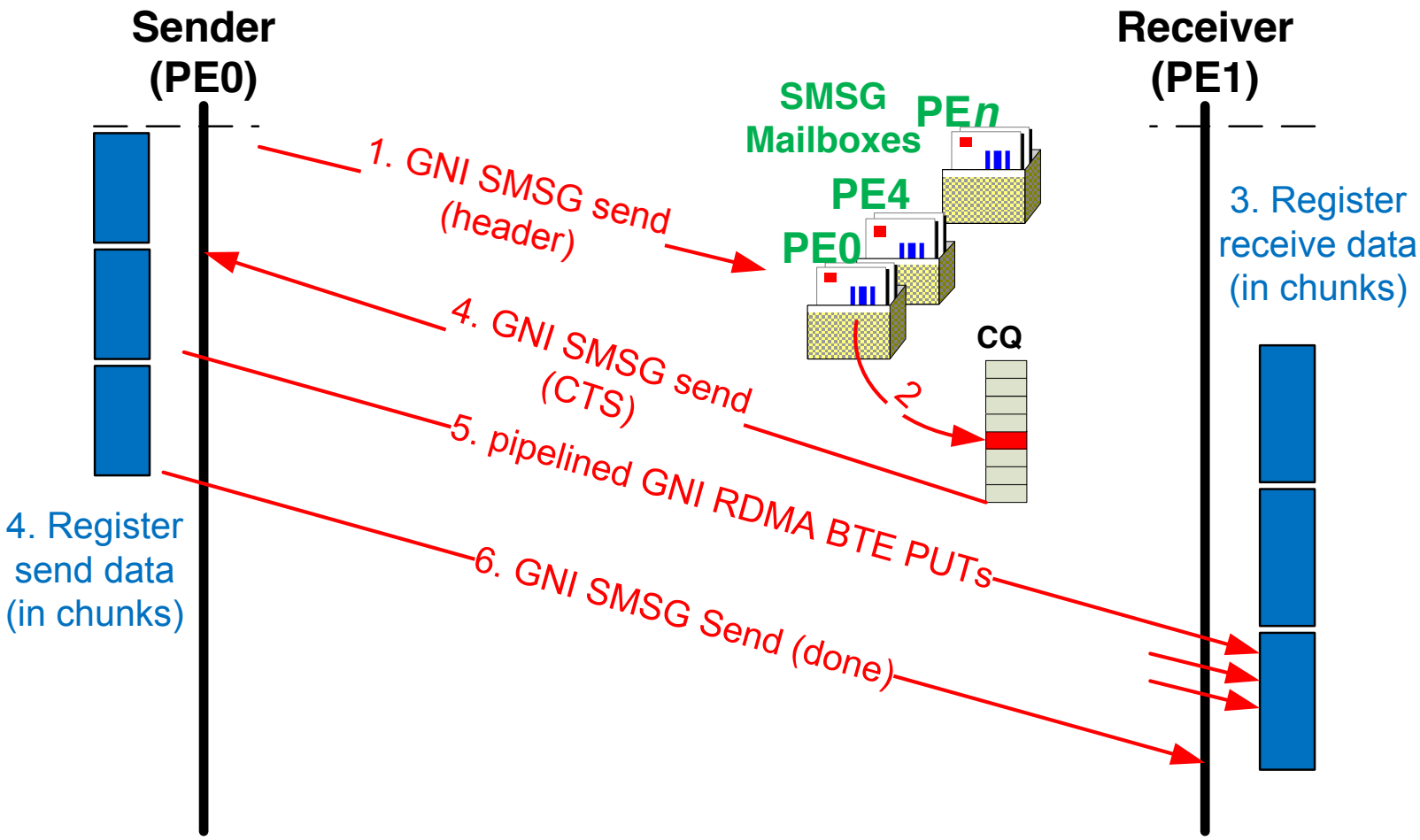
MPI Eager 1


Sender
(PE0)
Receiver
(PE1)
SMSG
Mailboxes
PEn
PE4
PE0
CQ
Application space
1. copy
2. GNI SMSG send
(header)
3
4. RDMA FMA
GET
5. GNI SMSG (Recv Done)
6. copy
Pre-allocated
(registered)
Runtime buffers

MPI Rendezvous 0



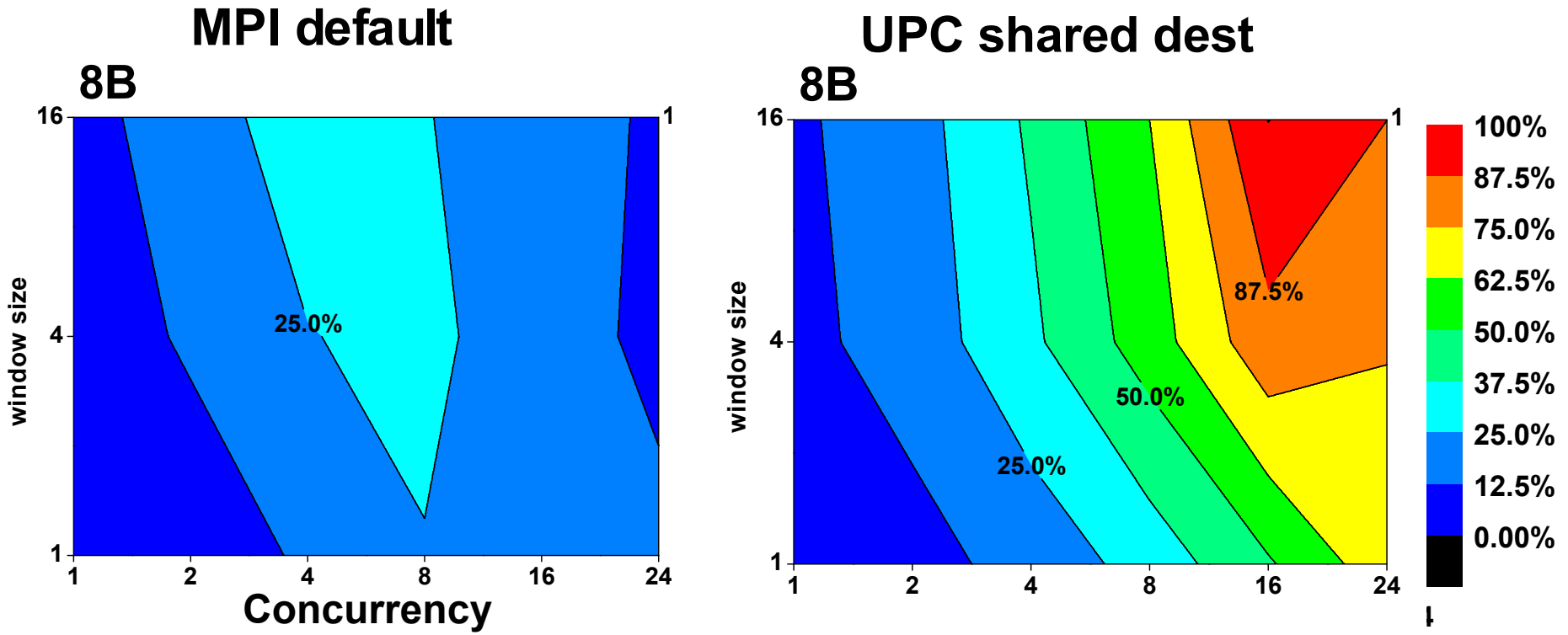
MPI Rendezvous 1



Two-sided Summary of Challenges

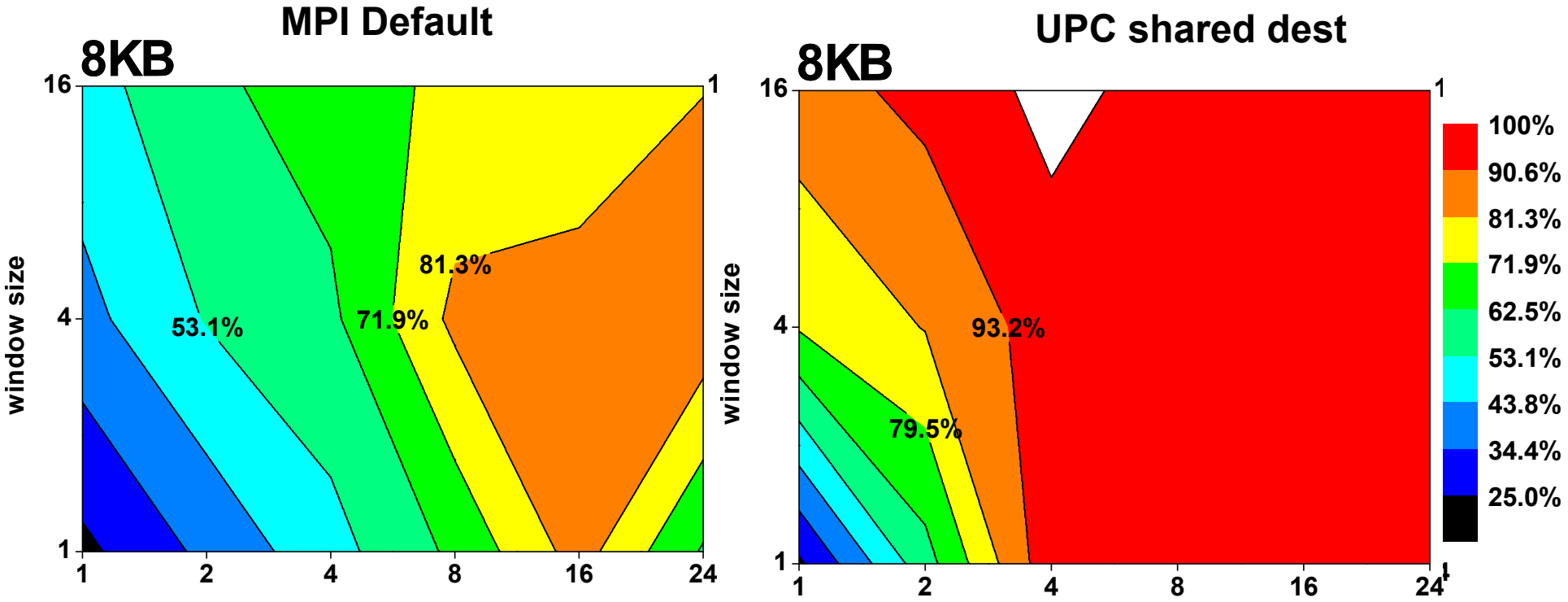
- Remote involvement
  - Rank-to-rank strict ordering can cause
    - Node-to-node strict ordering by hardware
    - How to handle message cancellation?!
  - Message size ambiguity
    - int MPI_Irecv(buf, **count**,datatype, source, tag, …)
      - count reflects buffer size **NOT** msg size.
      - Receiver cannot tell what protocol will be used before matching
    - Probing obstruct relaxed ordering progress.
- Remote readiness for transfer
  - All memory space default to local
  - Registration is on-demand
    - Variability of performance based on hit/miss in registration cache.

MPI vs. UPC 8B Transactions



- Experiment:
  - Vary number of messages per rank.
  - Vary the number of ranks per node.

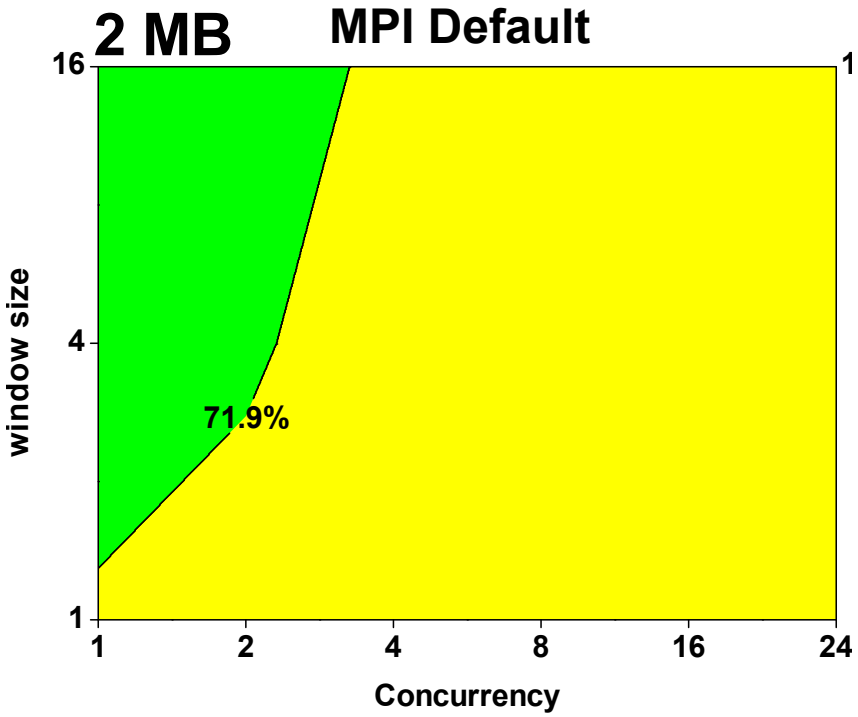
MPI vs. UPC 8KB Transactions

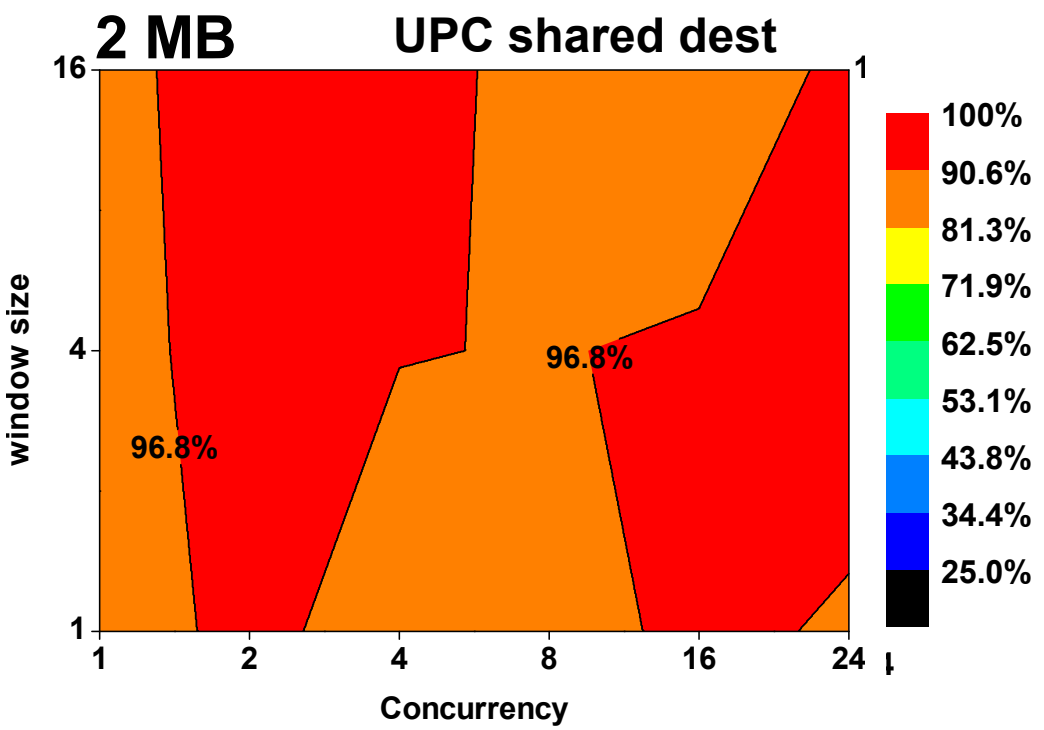


Application developer perspective:

What level of concurrency should I use?
What is the implication for intra-node communications?

MPI vs. UPC 2MB Transactions





Two-sided MPI vs. One-sided UPC

Difference is NOT due:

Runtime optimization

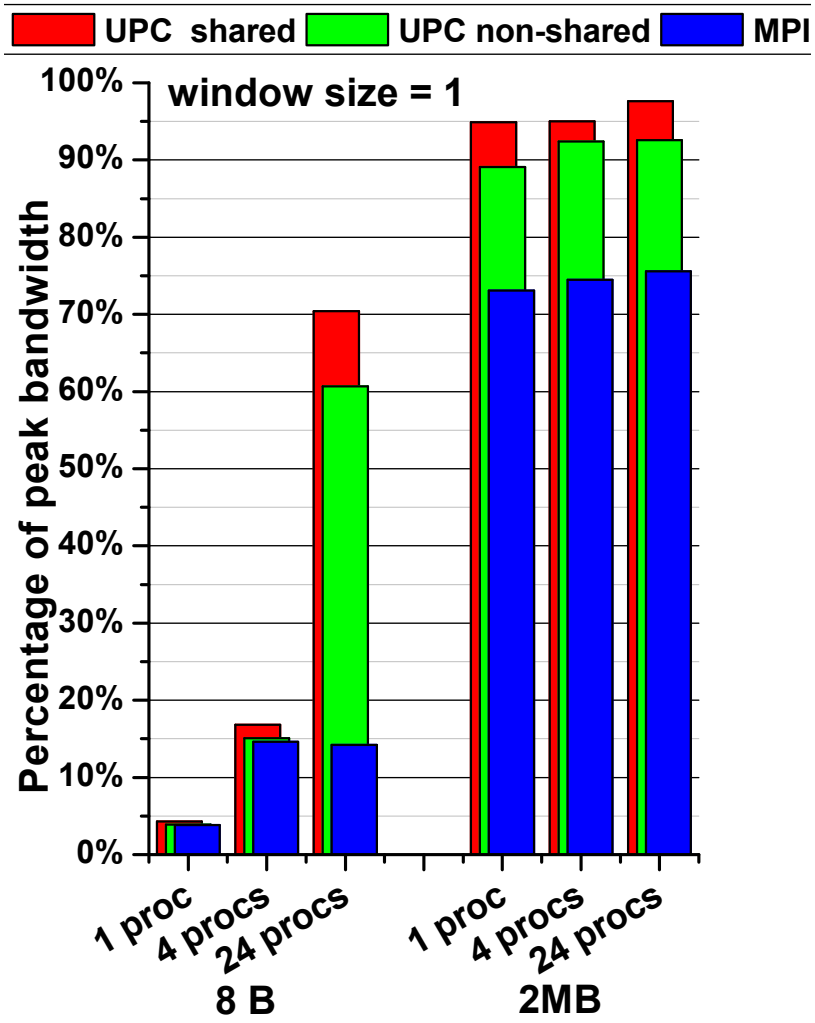
Registration overhead

or, Copying

Difference is due to:

language semantic

and ability to exploit relaxed ordering.



Conclusion

- One-sided communication exploits relaxed ordering with ease
  - Communicate-able memory is easier to identify
    - annotated shared - can be registered upfront.
  - Relaxed vs. strict ordering is explicitly specified
    - by programmer (or programming language). (Default to relaxed)
  - Large percentage of the peak performance for different communication patterns
- Two-sided faces the following challenges
  - Strict matching between send and receives (large startup overhead)
  - Locality notion (everything default to local)
    - Expensive to prepare buffer for communication (registration).
  - Receiver ambiguity about transactions (probing or over allocation)
  - Performance may need high node concurrency (problematic to in-node communication).