TORSTEN HOEFLER

Remote Memory Access Programming Models, Runtime Systems and Compilers

TORSTEN HOEFLER

Fast Runtime Systems, Remote Memory Access Programming Models, ~~and Compilers~~

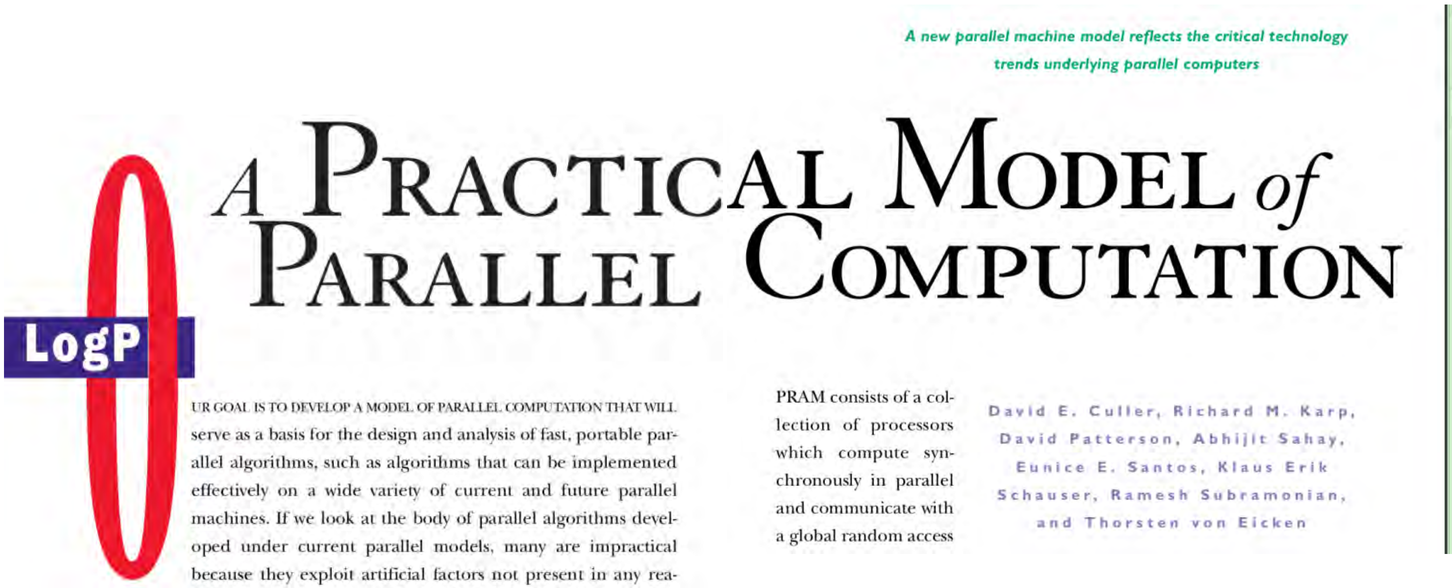
Motivation & Goals

- **My dream: provably optimal performance**
  - From problem to machine code
  - How to get there?

- **Model-based Performance Engineering!**
  1. Design a system model
  2. Define your problem
  3. Find (close-to) optimal solution in model
  4. Implement, test, refine if necessary

- **Will demonstrate techniques & insights**
  - And obstacles ☺

Example: Message Passing, Log(G)P


A new parallel machine model reflects the critical technology trends underlying parallel computers

LogP

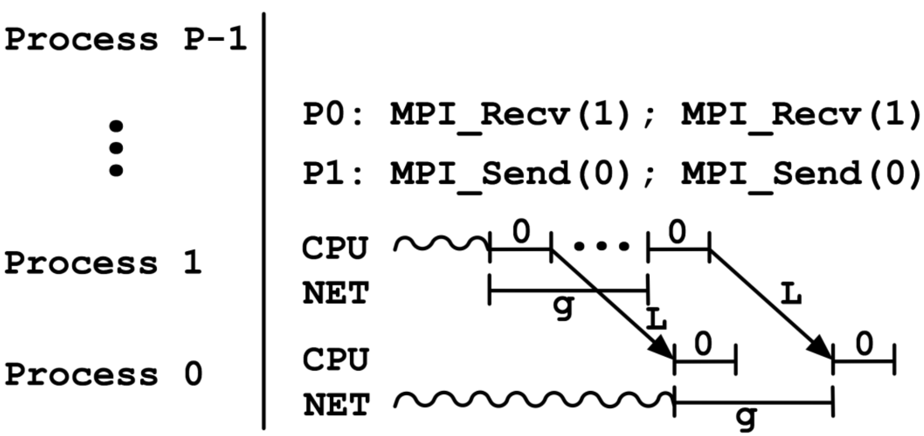
A PRACTICAL MODEL of PARALLEL COMPUTATION

UR GOAL IS TO DEVELOP A MODEL OF PARALLEL COMPUTATION THAT WILL serve as a basis for the design and analysis of fast, portable parallel algorithms, such as algorithms that can be implemented effectively on a wide variety of current and future parallel machines. If we look at the body of parallel algorithms developed under current parallel models, many are impractical because they exploit artificial factors not present in any rea-

PRAM consists of a collection of processors which compute synchronously in parallel and communicate with a global random access

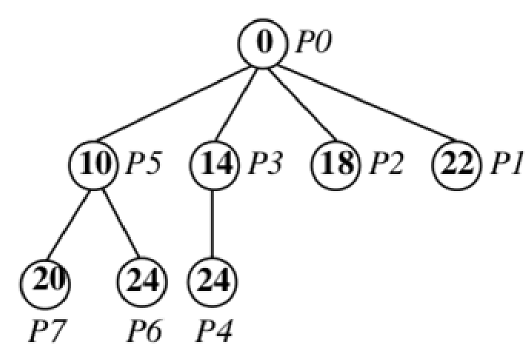
David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauser, Ramesh Subramonian, and Thorsten von Eicken

CACM 1996

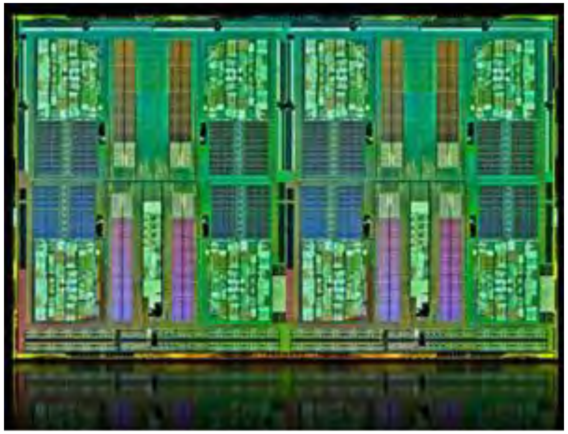


Broadcast Problem

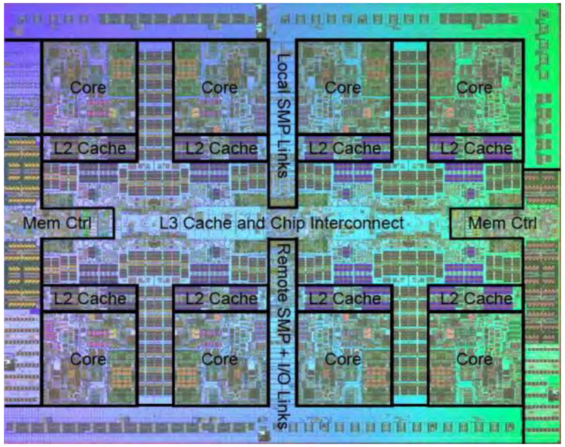
Optimal Solution



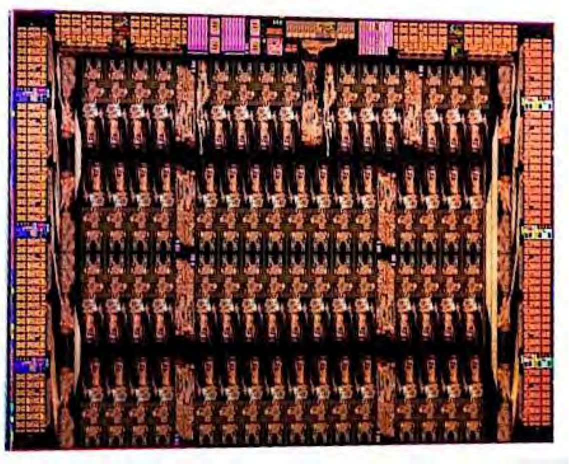
Hardware Reality



Interlagos, 8/16 cores, source: AMD

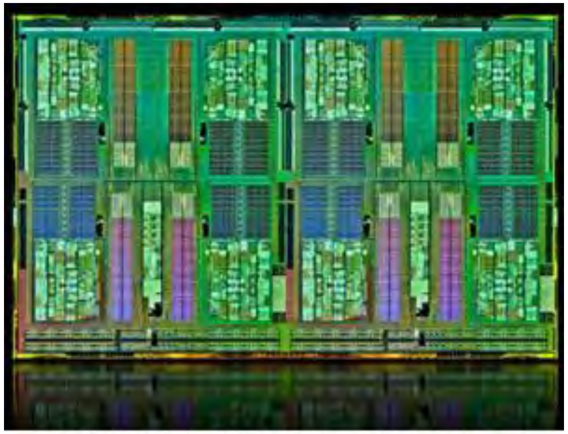


POWER 7, 8 cores, source: IBM

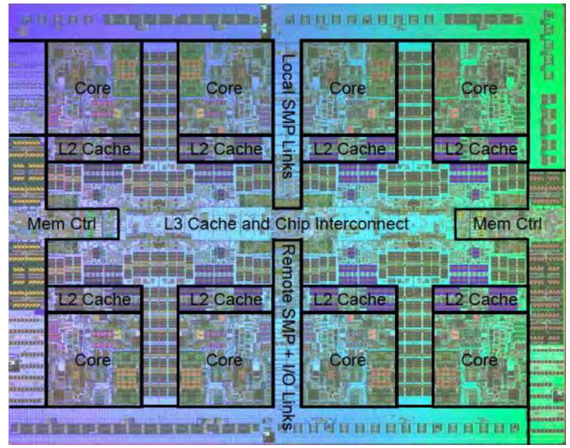


Xeon Phi, 64 cores, source: Intel

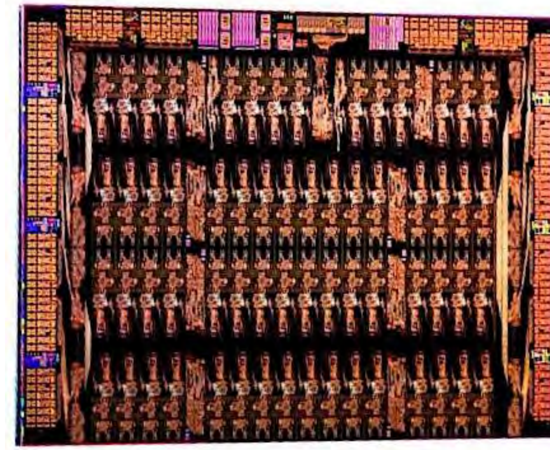
Hardware Reality



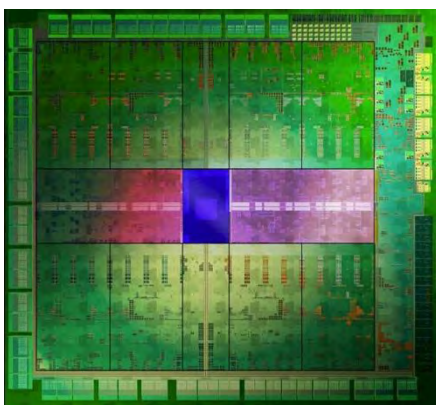
Interlagos, 8/16 cores, source: AMD



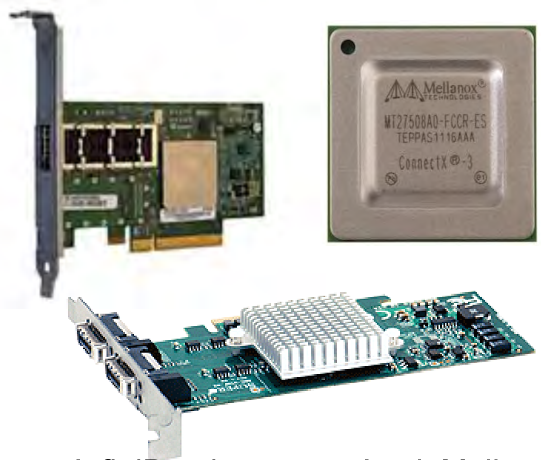
POWER 7, 8 cores, source: IBM



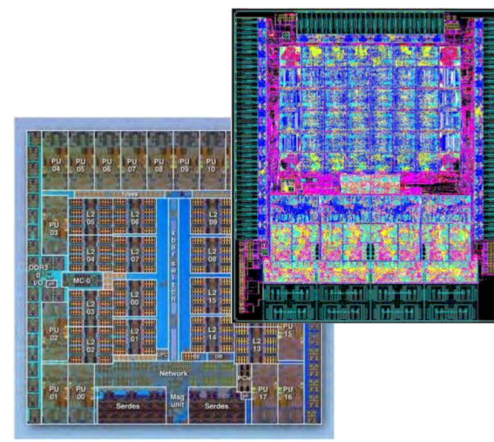
Xeon Phi, 64 cores, source: Intel



Kepler GPU, source: NVIDIA



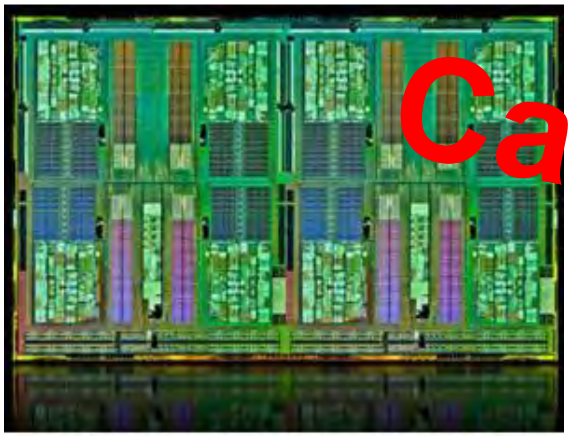
InfiniBand, sources: Intel, Mellanox



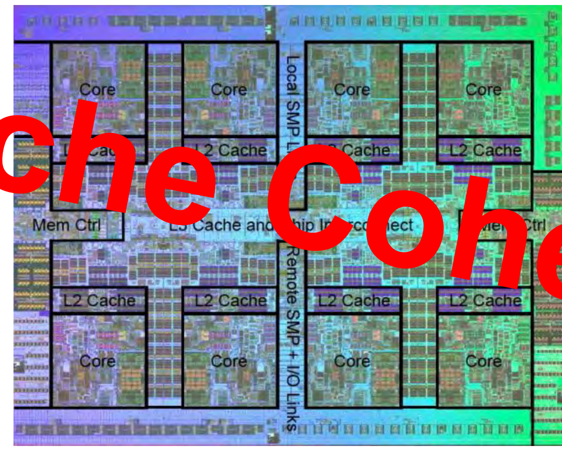
BG/Q, Cray Aries, sources: IBM, Cray

Hardware Reality

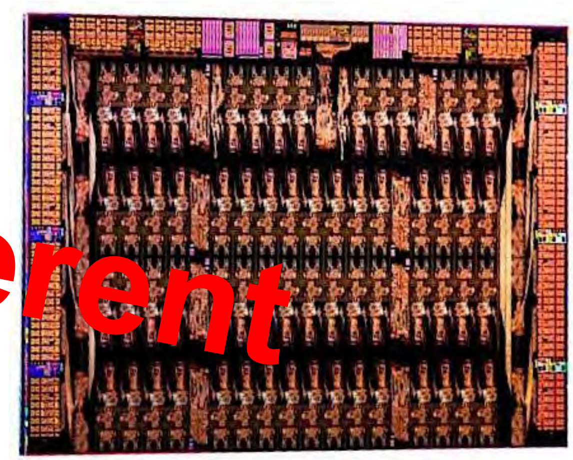
Cache Coherent



Interlagos, 8/16 cores, source: AMD

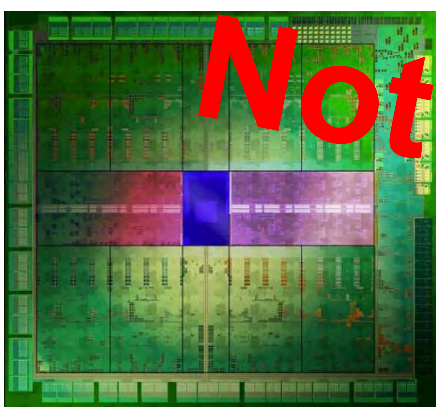


POWER 7, 8 cores, source: IBM



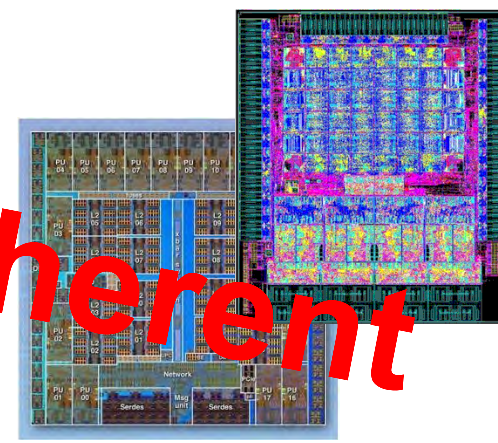
Xeon Phi, 64 cores, source: Intel

Not Cache Coherent



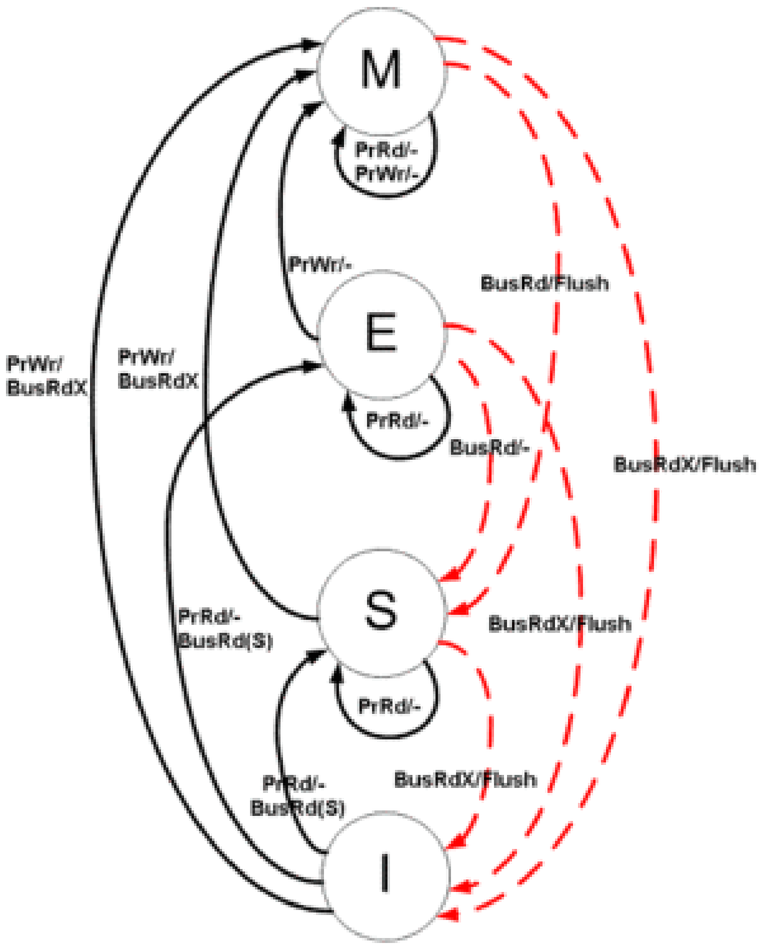
Kepler GPU, source: NVIDIA

InfiniBand, sources: Intel, Mellanox

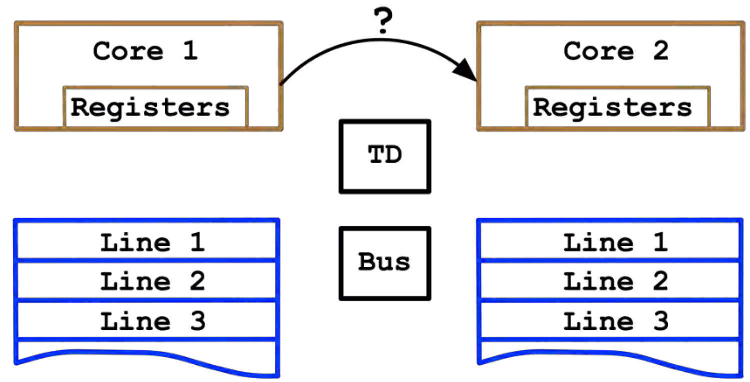


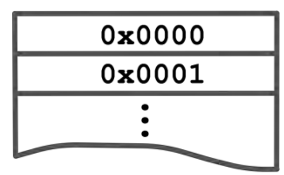
BG/Q, Cray Aries, sources: IBM, Cray

Topic: Cache-Coherent Communication

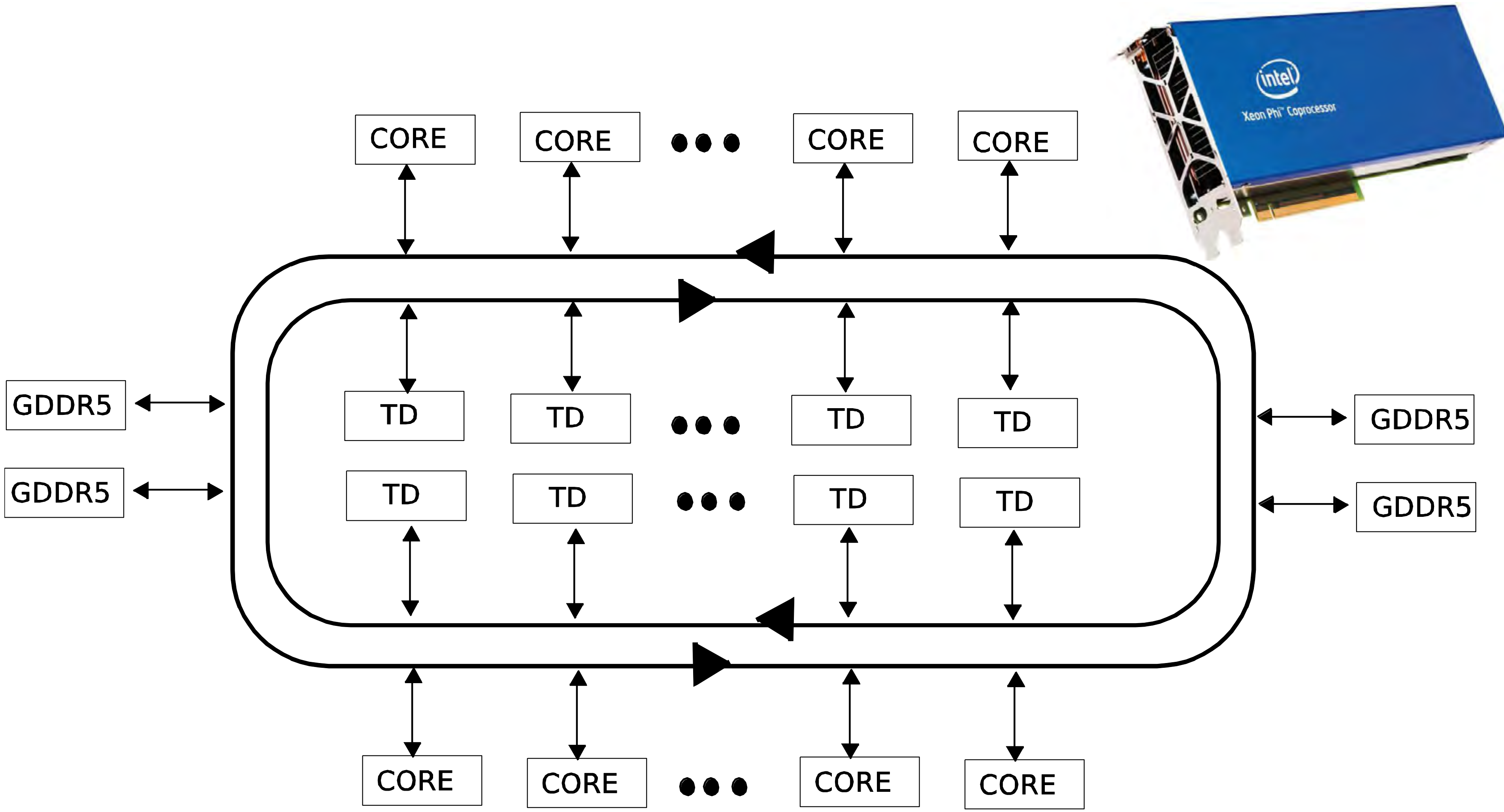


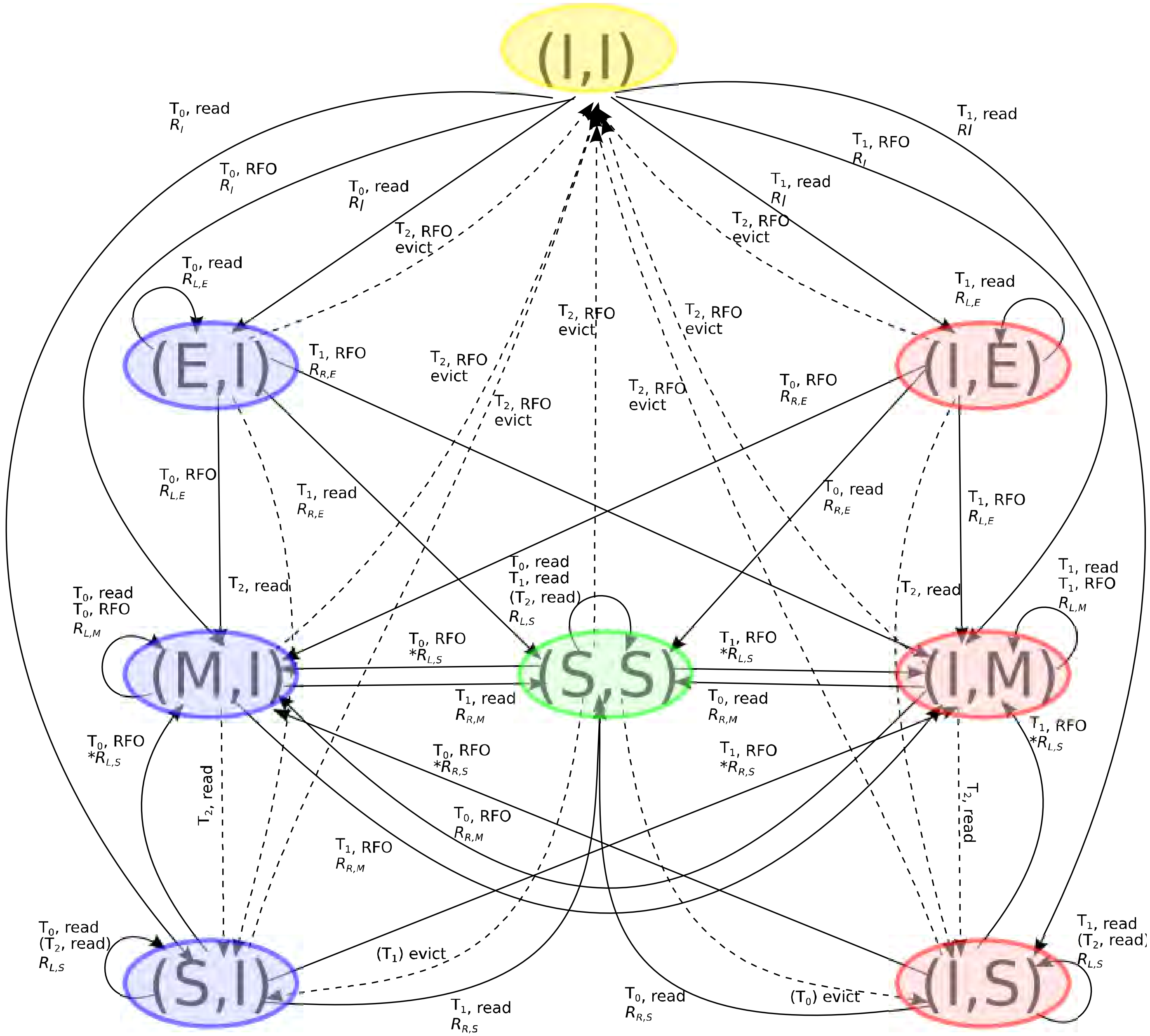
Source: Wikipedia

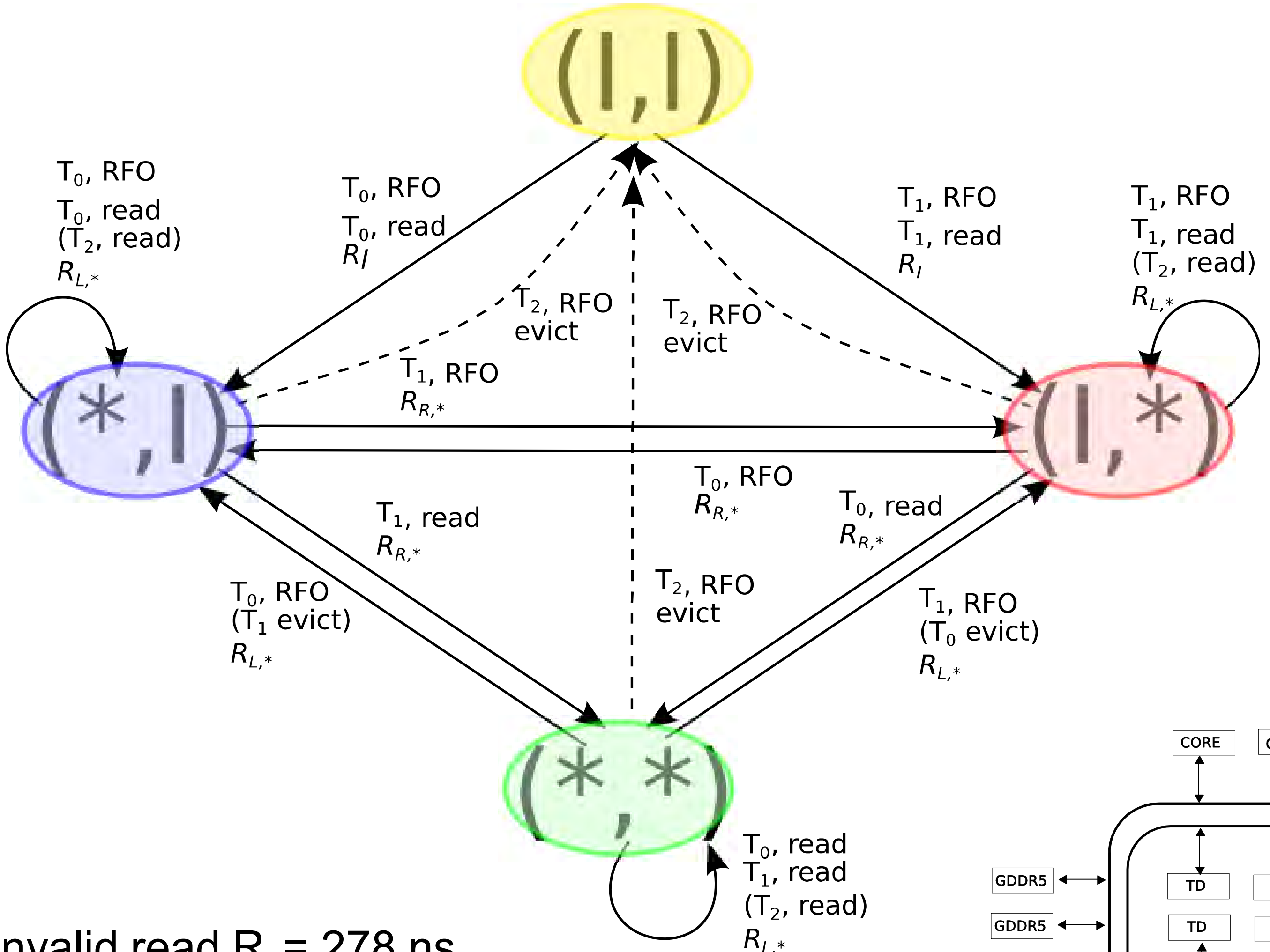




Xeon Phi (Rough) Architecture



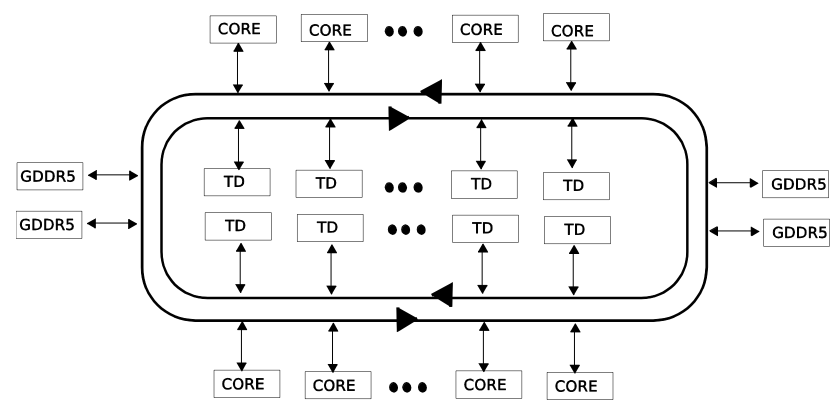

(I,I)
(E,I)
(I,E)
(M,I)
(S,S)
(I,M)
(S,I)
(I,S)
T_0, read
R_I
T_0, RFO
R_I
T_0, read
RI
T_2, RFO
evict
T_1, read
RI
T_1, RFO
R_I
T_1, read
RI
T_2, RFO
evict
T_0, read
$R_{L,E}$
T_1, read
$R_{L,E}$
T_1, RFO
$R_{R,E}$
T_2, RFO
evict
T_2, RFO
evict
T_2, RFO
evict
T_2, RFO
evict
T_2, RFO
evict
T_0, RFO
$R_{R,E}$
T_0, RFO
$R_{L,E}$
T_1, read
$R_{R,E}$
T_0, read
$R_{R,E}$
T_1, RFO
$R_{L,E}$
T_0, read
T_1, read
(T_2, read)
$R_{L,S}$
T_2, read
T_2, read
T_0, read
T_0, RFO
$R_{L,M}$
T_1, read
T_1, RFO
$R_{L,M}$
T_0, RFO
$*R_{L,S}$
T_1, RFO
$*R_{L,S}$
T_1, read
$R_{R,M}$
T_0, read
$R_{R,M}$
T_0, RFO
$*R_{L,S}$
T_0, RFO
$*R_{R,S}$
T_1, RFO
$*R_{R,S}$
T_1, RFO
$*R_{L,S}$
T_2, read
T_2, read
T_0, RFO
$R_{R,M}$
T_1, RFO
$R_{R,M}$
T_0, read
(T_2, read)
$R_{L,S}$
T_1, read
(T_2, read)
$R_{L,S}$
(T_1) evict
(T_0) evict
T_1, read
$R_{R,S}$
T_0, read
$R_{R,S}$



Invalid read R_I = 278 ns

Local read: R_L= 8.6 ns

Remote read R_R = 235 ns



Inspired by Molka et al.: "Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system"

Designing Optimal Algorithms

- **Broadcast example:**

Bcast cost

Number of levels

Reached threads

$$\mathcal{T}_{tree} = \sum_{i=1}^{d} \mathcal{T}_C(k_i) = \sum_{i=1}^{d} (c \cdot k_i + b)$$

$$= \sum_{i=1}^{d} (R_R + R_L + c \cdot (k_i - 1))$$

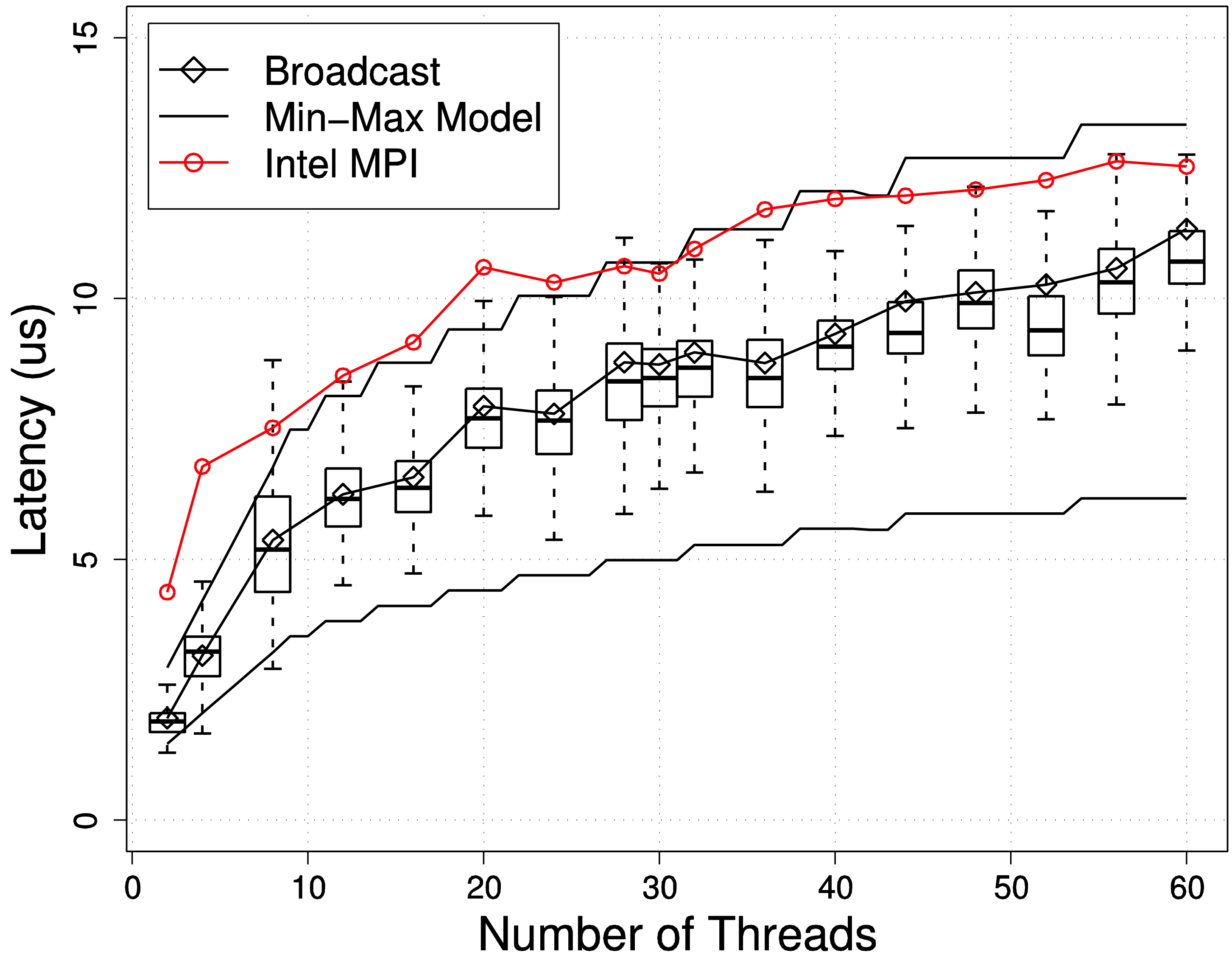
$$n_{th} \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j$$

$$\mathcal{T}_{sbcast} = \min_{d,k_i} \left(\mathcal{T}_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} \mathcal{T}_{nb}(k_i + 1) \right)$$

$$N \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j, \quad \forall i < j, k_i \leq k_j$$

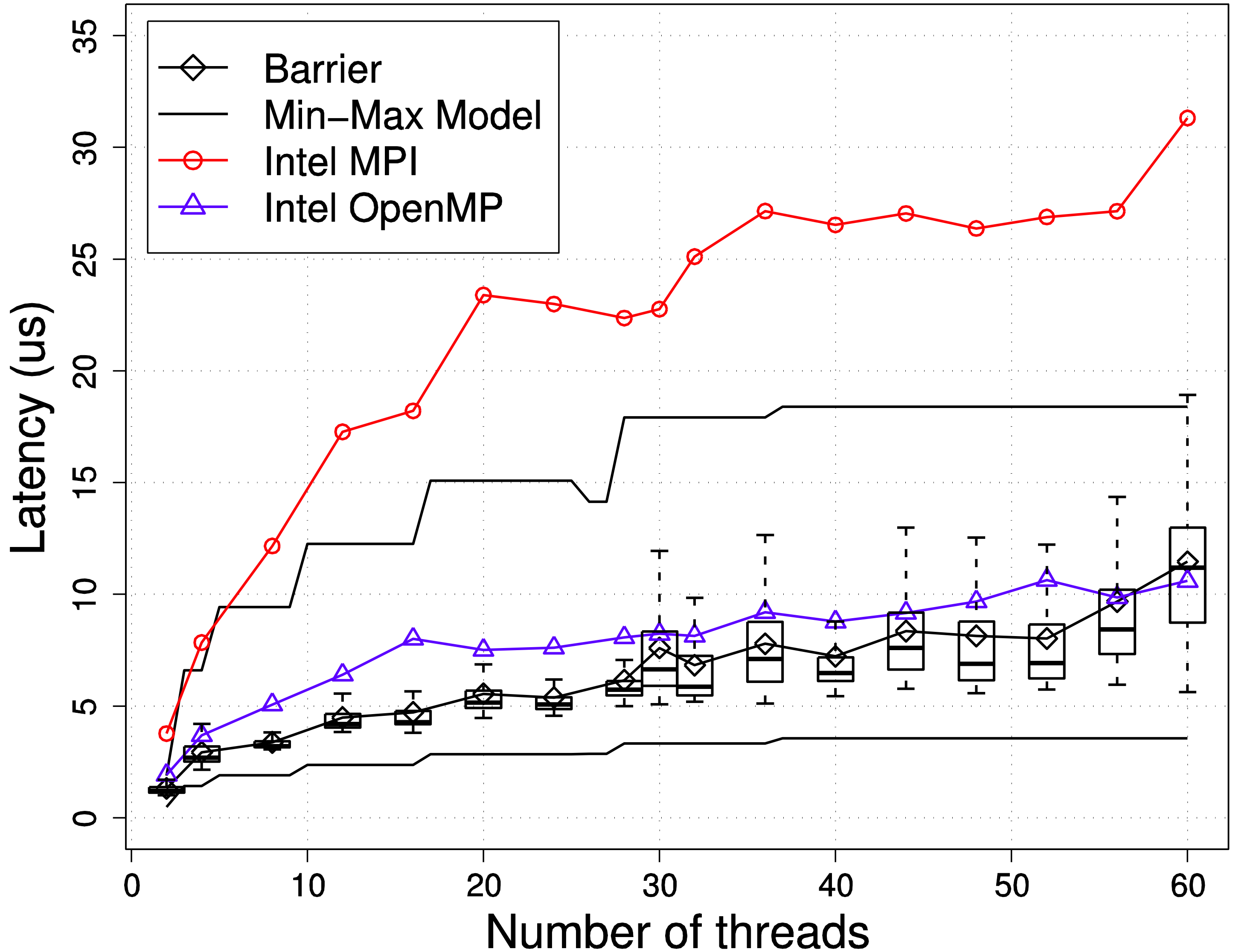
Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Small Broadcast (8 Bytes)



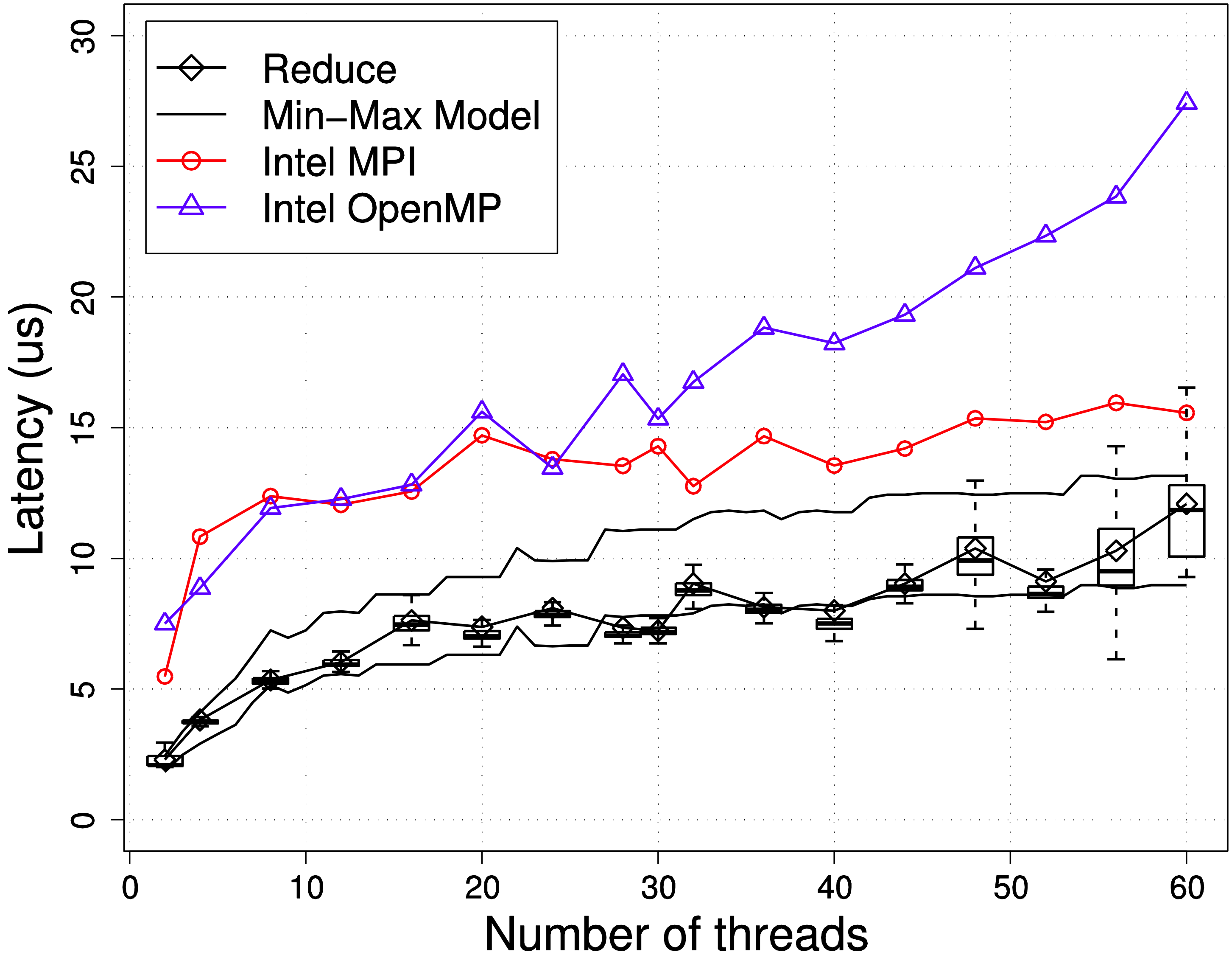
Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Barrier



Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

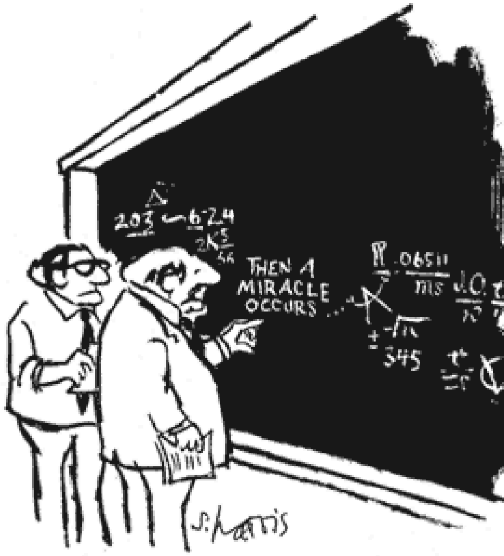
Small Reduction



Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Lessons learned

- **Rigorous modeling has large potential**
  - Coming with great cost (working on tool support [1])
- **Understanding cache-coherent communication performance is incredibly complex (but fun)!**
  - Many states, min-max modeling, NUMA, …
  - Have models for Sandy Bridge now (QPI, even worse!)
- **Cache coherence really gets in our way here ☹**
- **Obvious question: why do we need cache coherence?**
  - Answer: well, we don't! If we program right!
  - One option: Remote Memory Access (RMA) programming [2]



"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

[1]: Calotoiu et al.: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes, SC13
[2]: Gerstenberger et al.: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13, Best Paper

MPI-3.0 RMA

- MPI-3.0 supports RMA (“MPI One Sided”)
  - Designed to react to hardware trends
  - Majority of HPC networks support RDMA

[1] http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

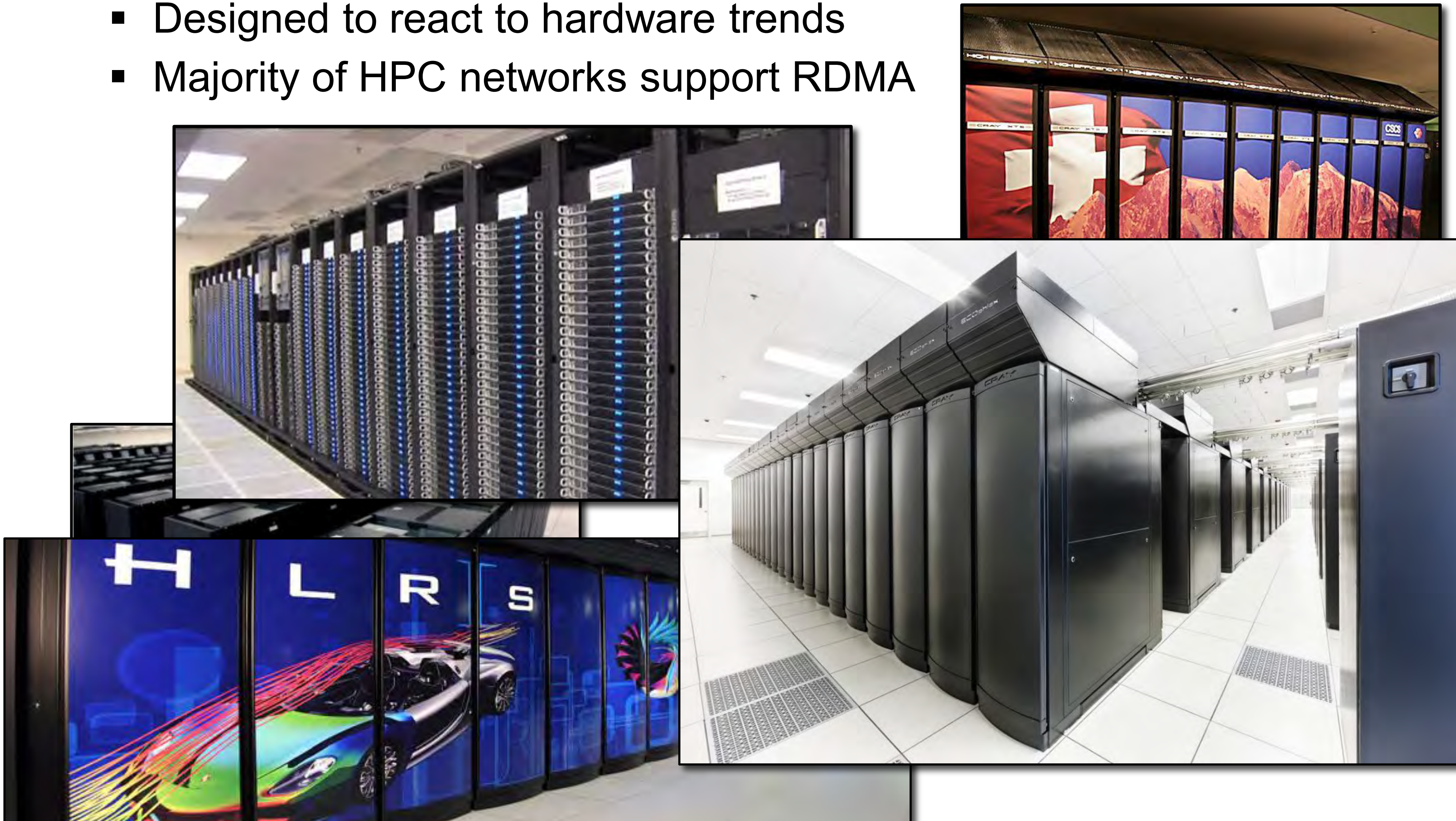
MPI-3.0 RMA

- MPI-3.0 supports RMA ("MPI One Sided")
  - Designed to react to hardware trends
  - Majority of HPC networks support RDMA

[1] http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

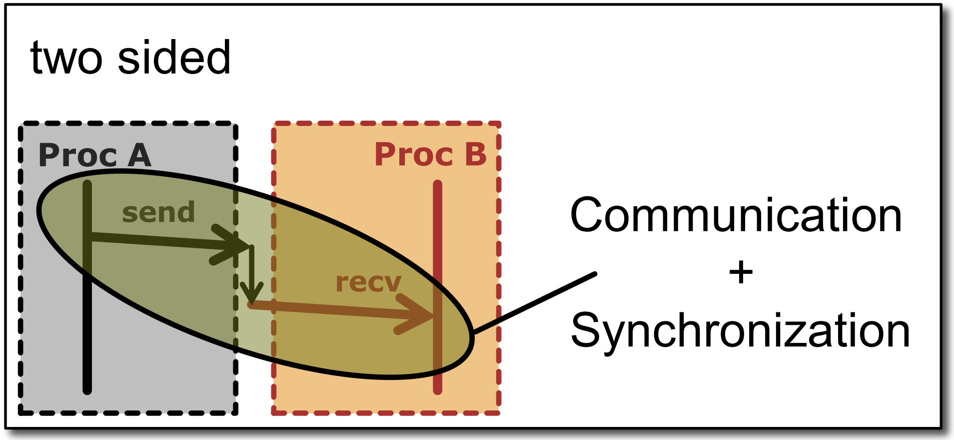
MPI-3.0 RMA

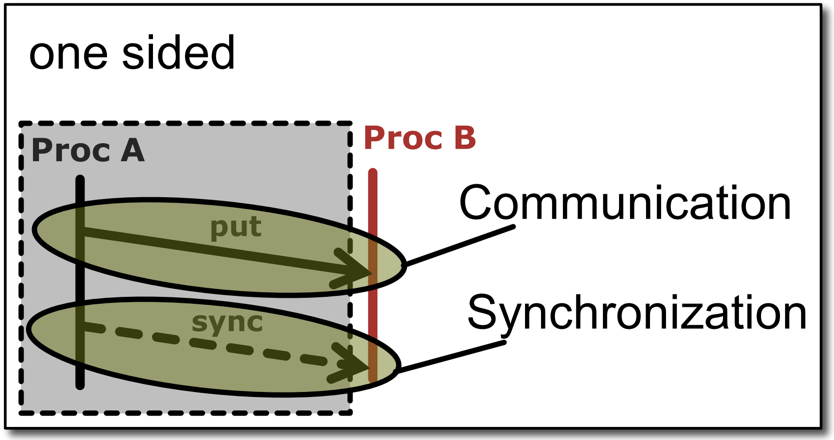
- MPI-3.0 supports RMA (“MPI One Sided”)
  - Designed to react to hardware trends
  - Majority of HPC networks support RDMA



MPI-3.0 RMA

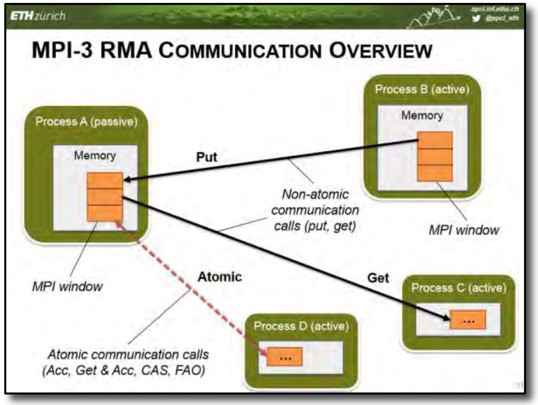
- MPI-3.0 supports RMA (“MPI One Sided”)
  - Designed to react to hardware trends
  - Majority of HPC networks support RDMA
- Communication is „one sided” (no involvement of destination)
- RMA decouples communication & synchronization
  - Different from message passing



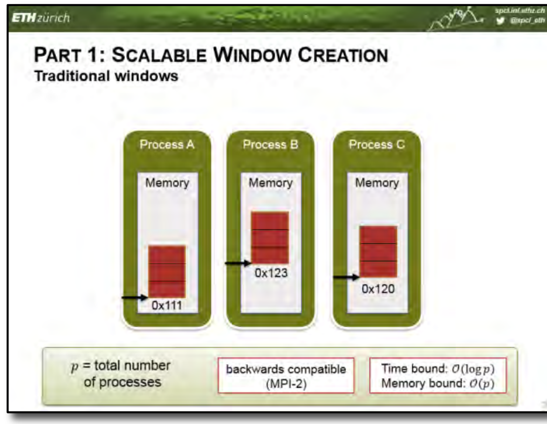


[1] http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

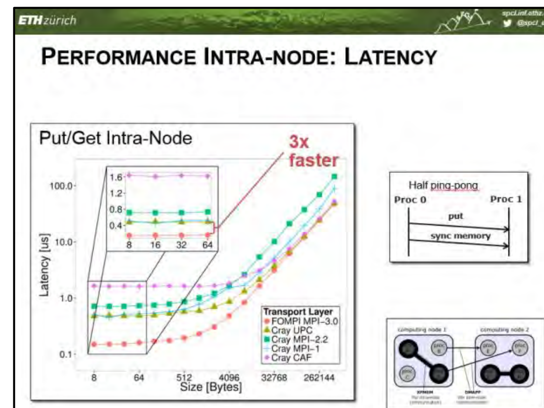
PRESENTATION OVERVIEW



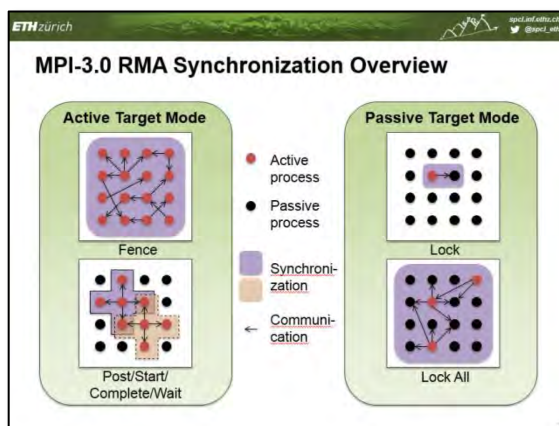
1. Overview of three MPI-3 RMA concepts



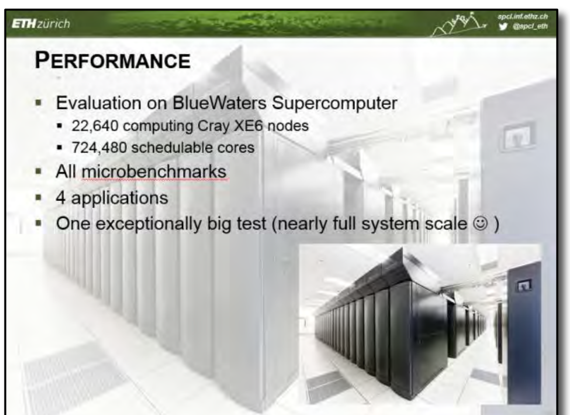
2. MPI window creation



3. Communication

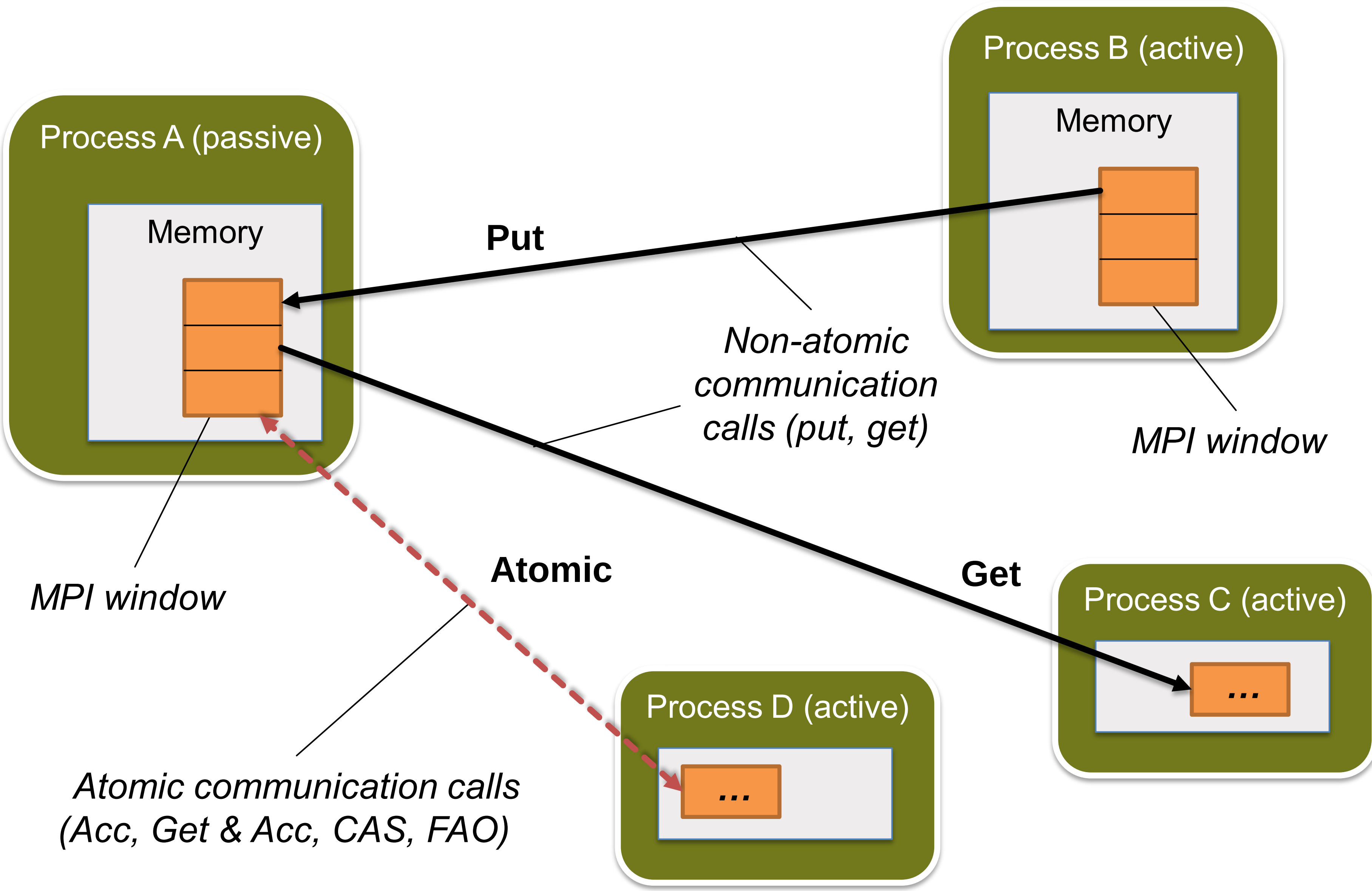


4. Synchronization

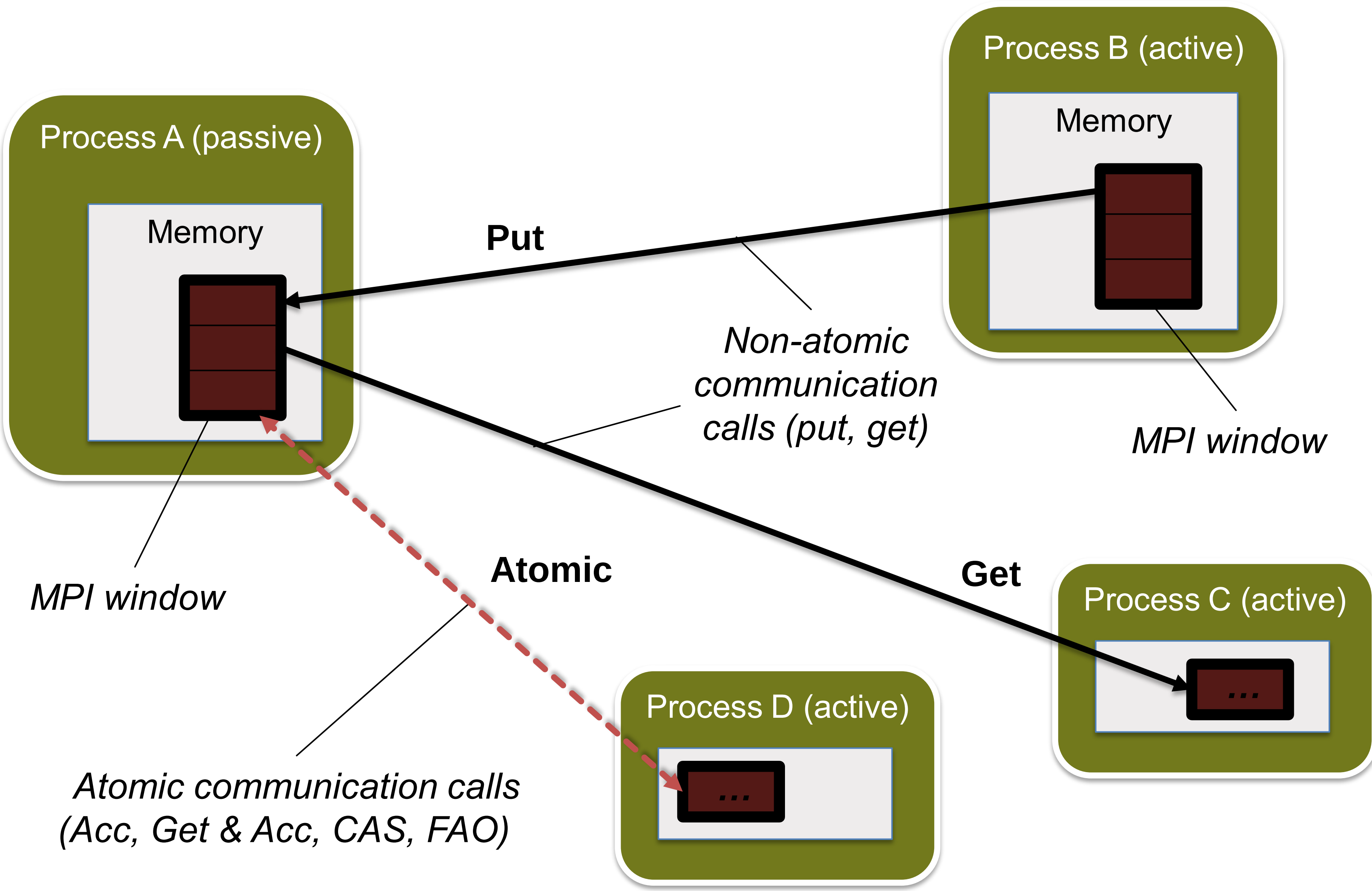


5. Application evaluation

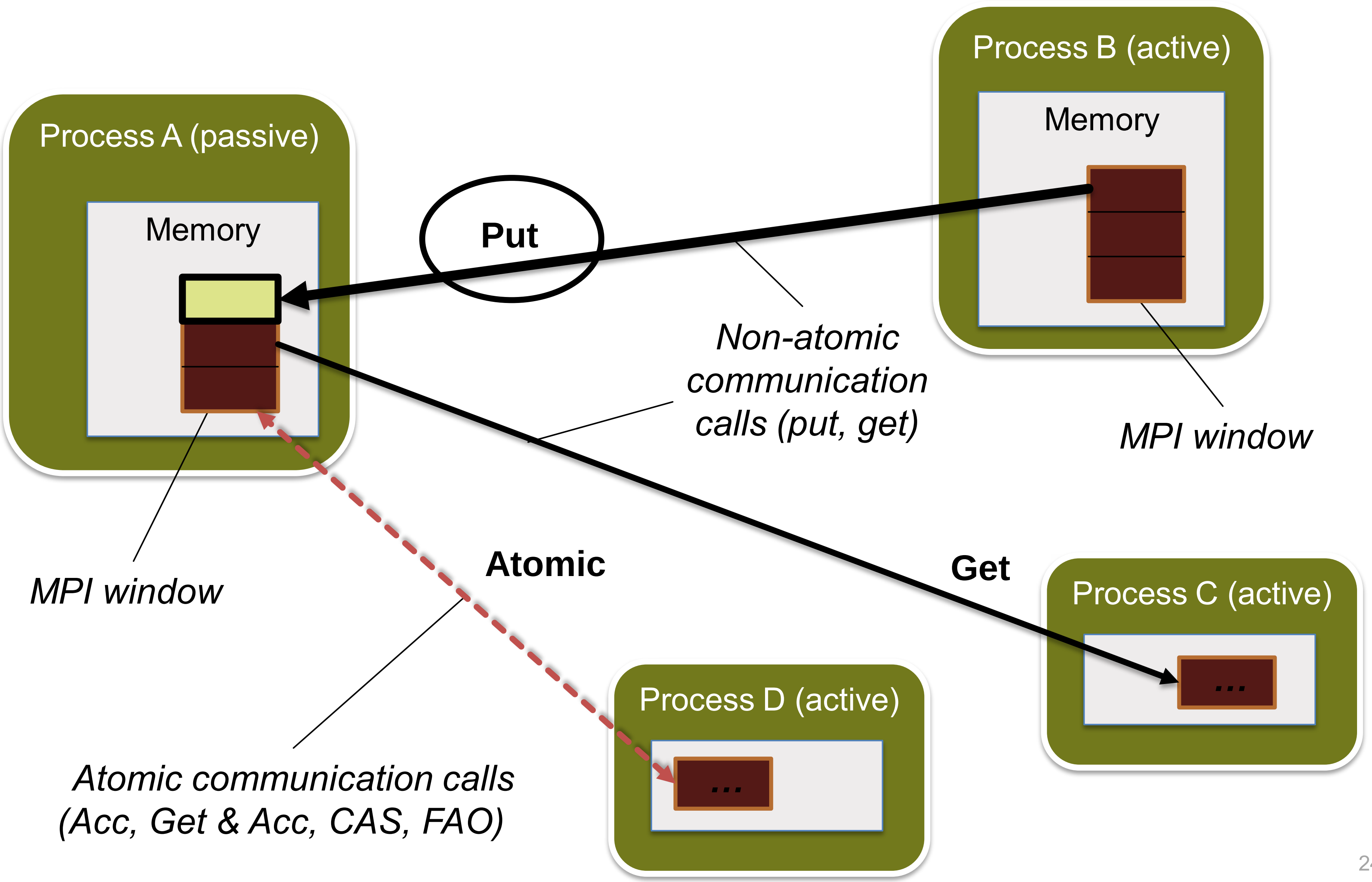
MPI-3 RMA Communication Overview



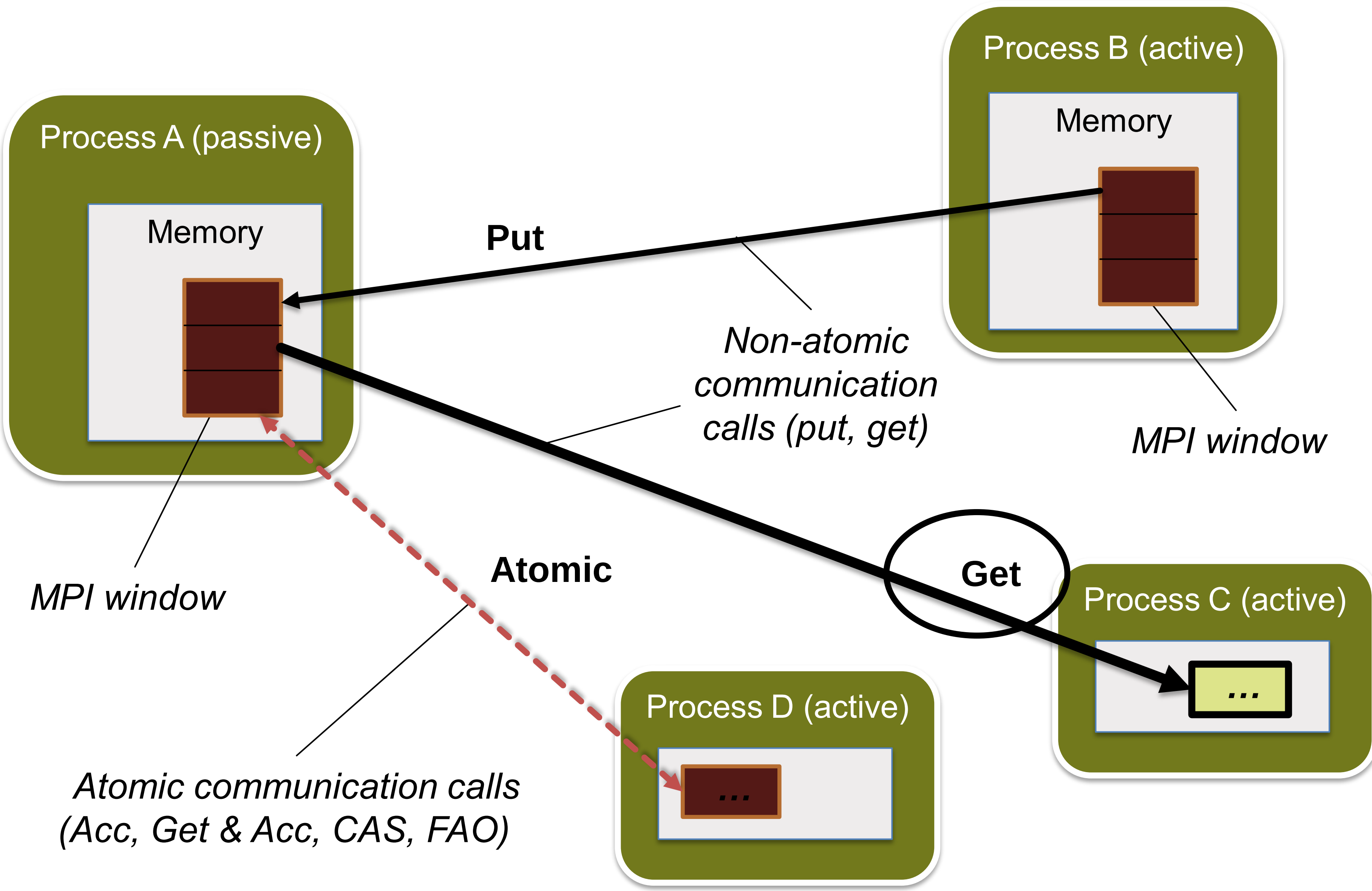
MPI-3 RMA Communication Overview



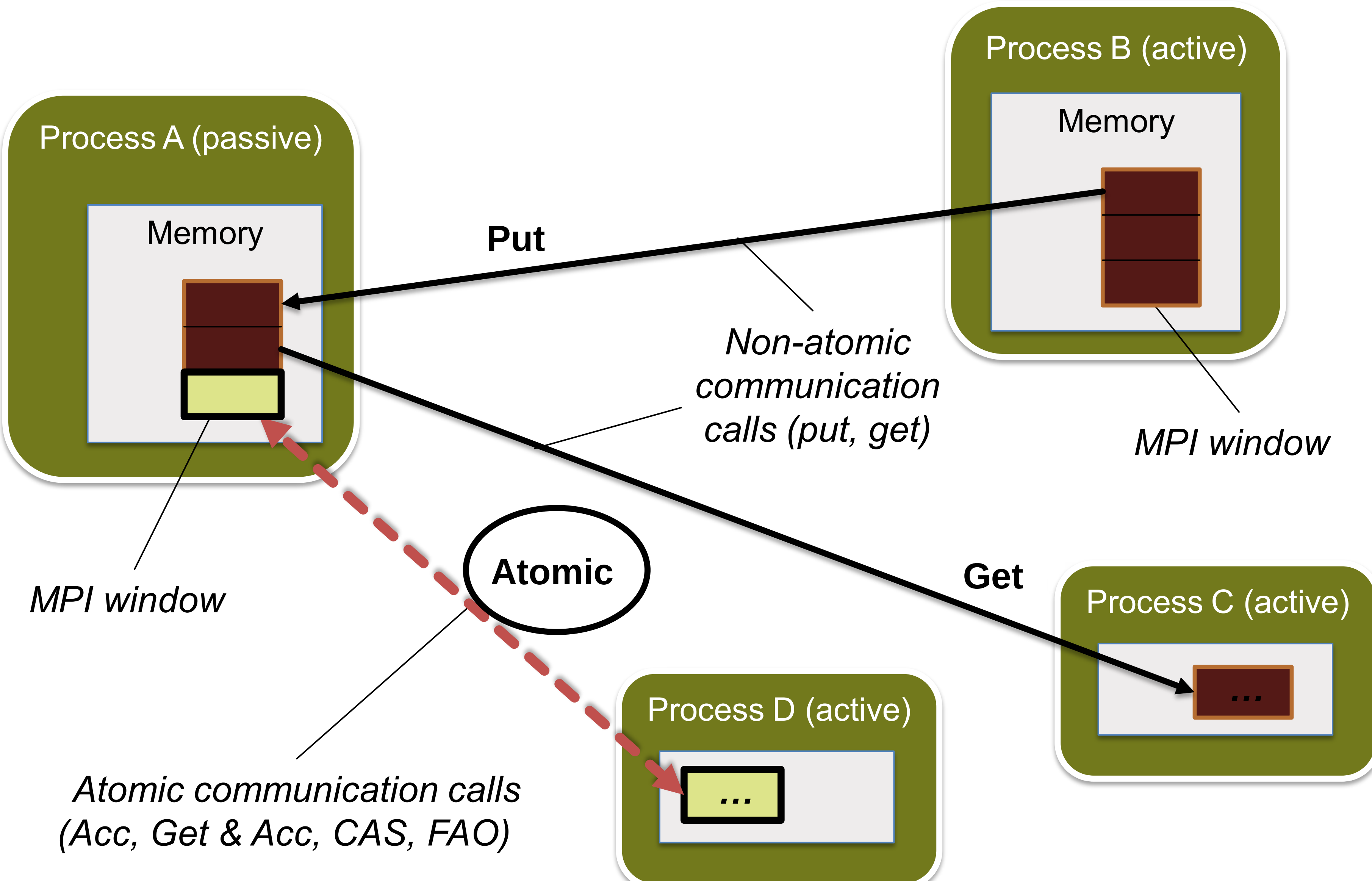
MPI-3 RMA Communication Overview



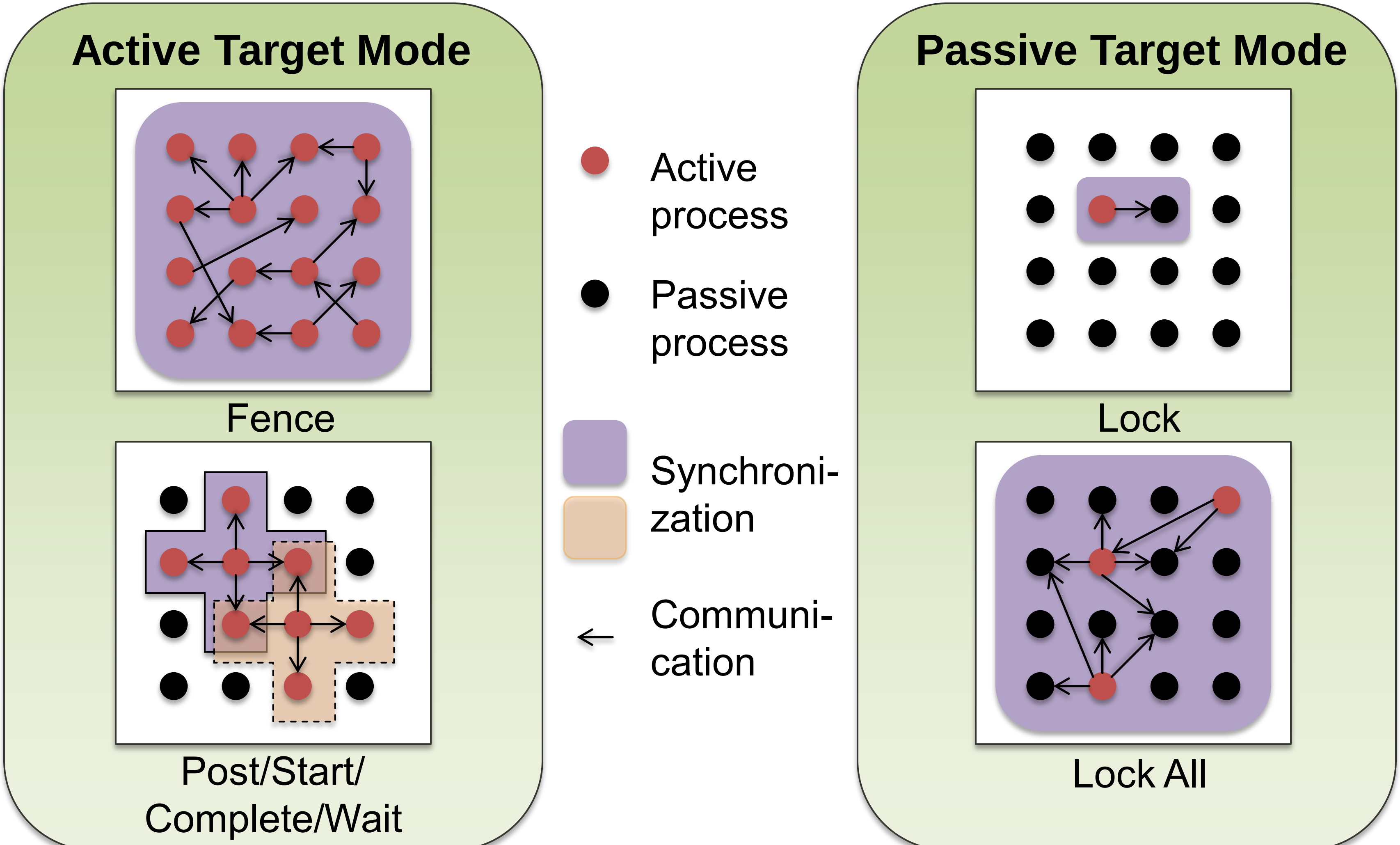
MPI-3 RMA Communication Overview



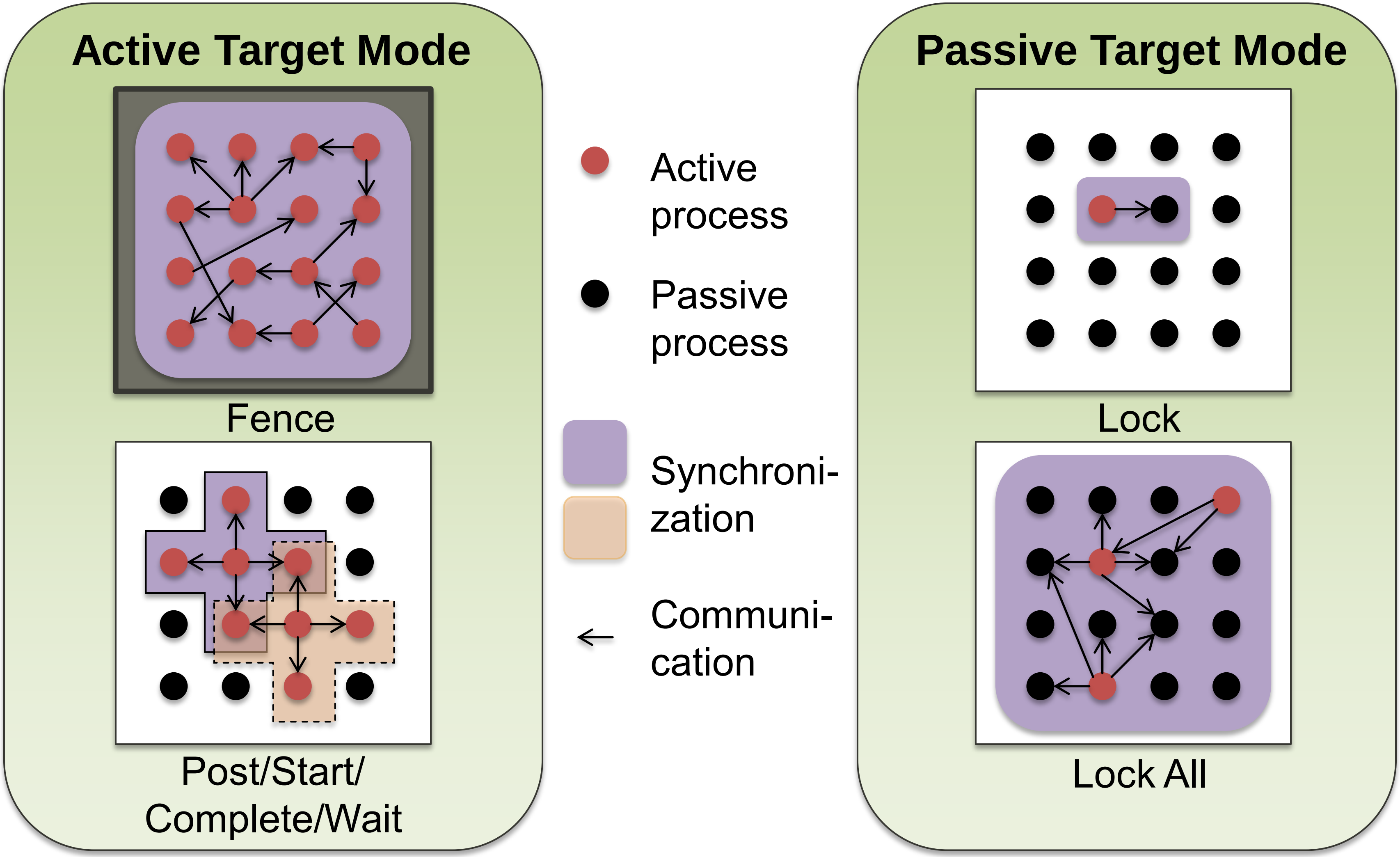
MPI-3 RMA Communication Overview



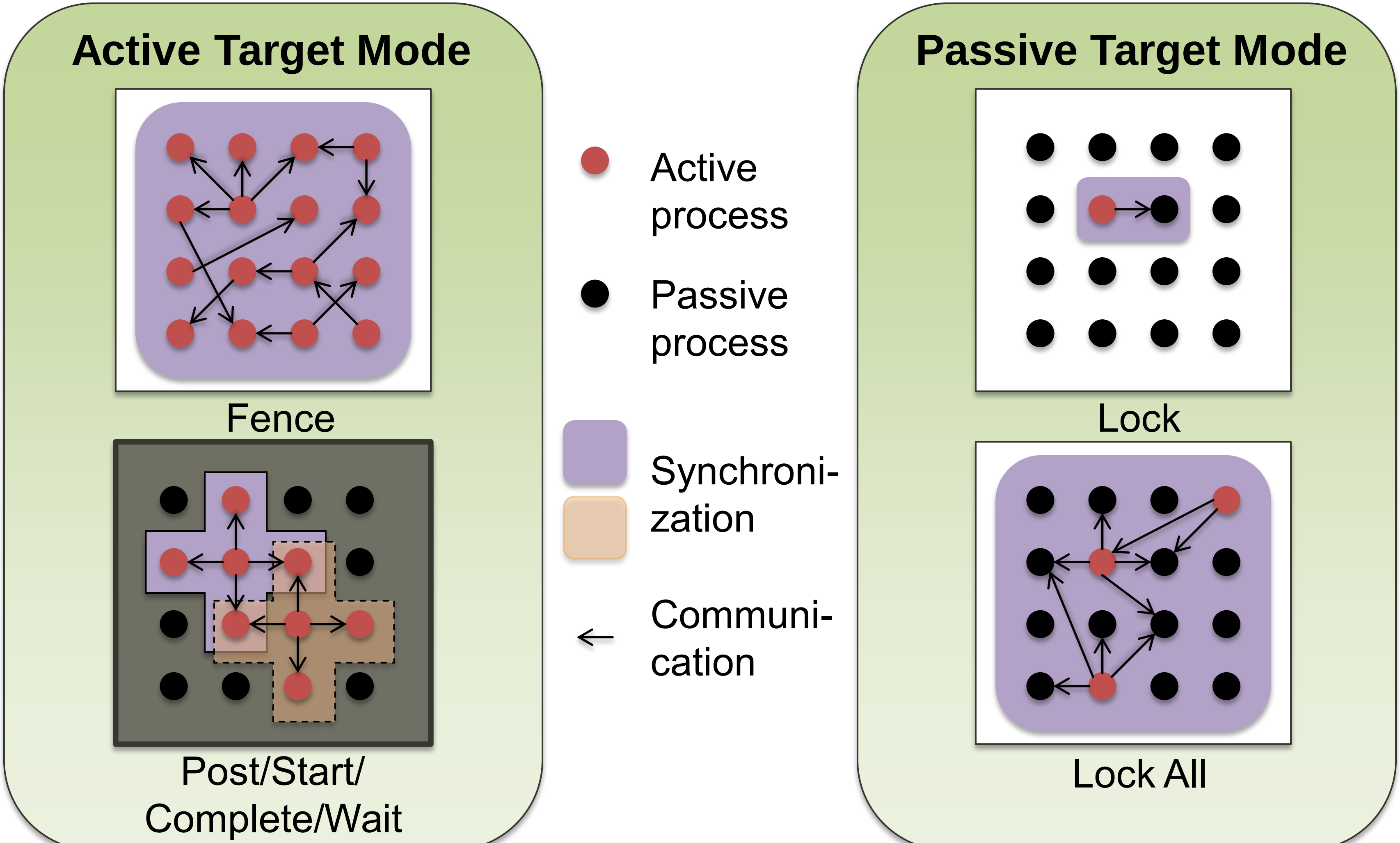
MPI-3.0 RMA Synchronization Overview



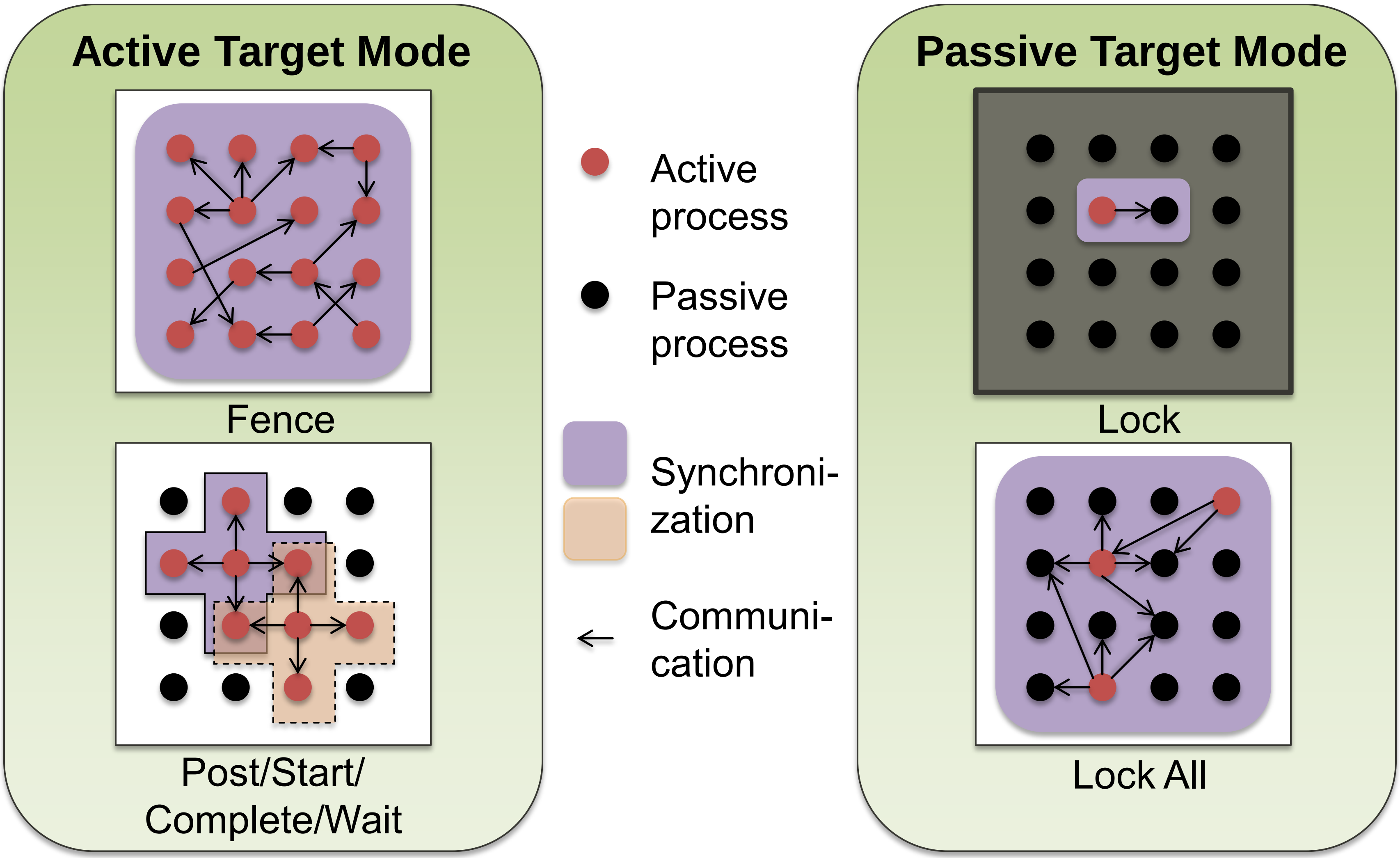
MPI-3.0 RMA Synchronization Overview



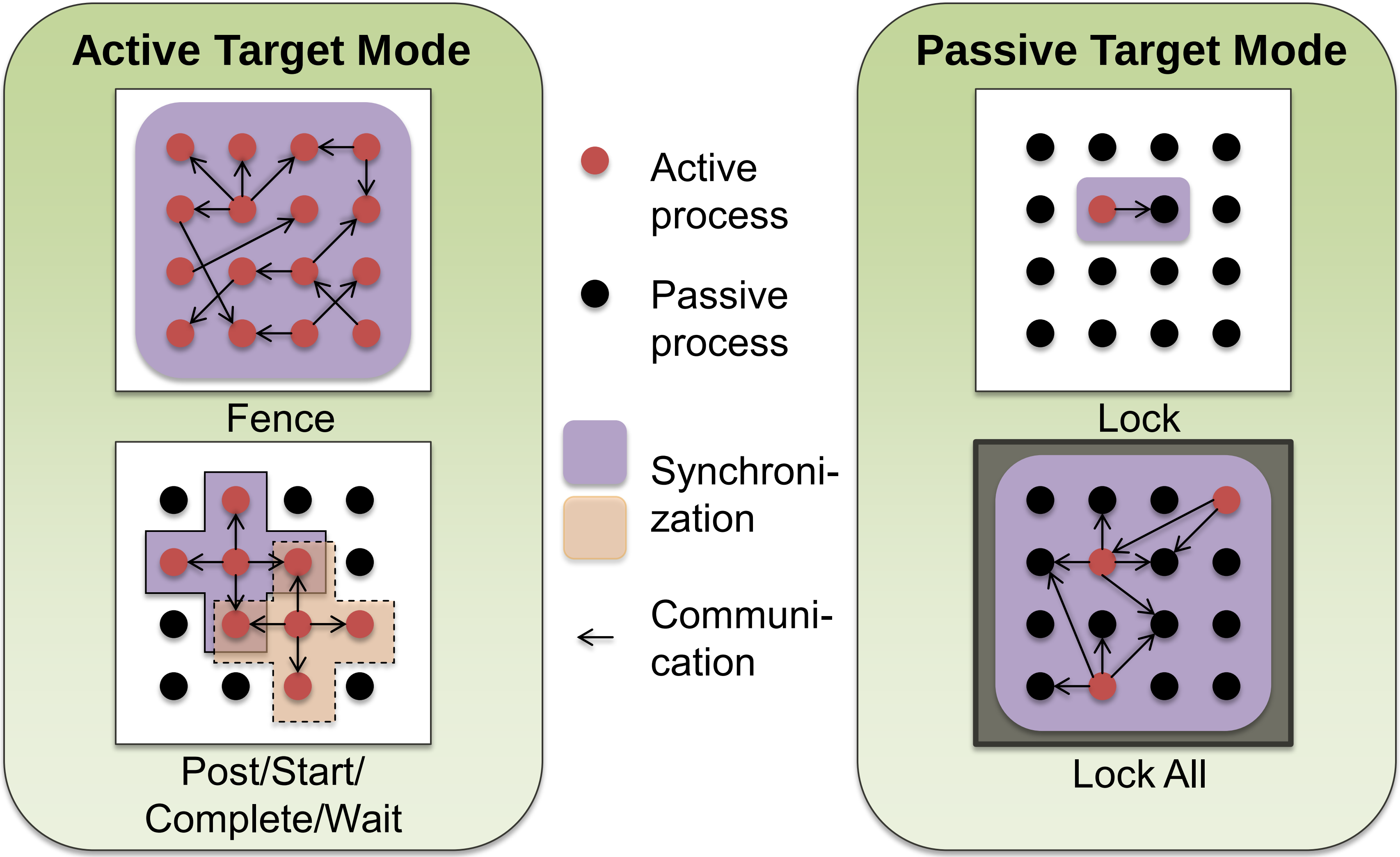
MPI-3.0 RMA Synchronization Overview



MPI-3.0 RMA SYNCHRONIZATION OVERVIEW

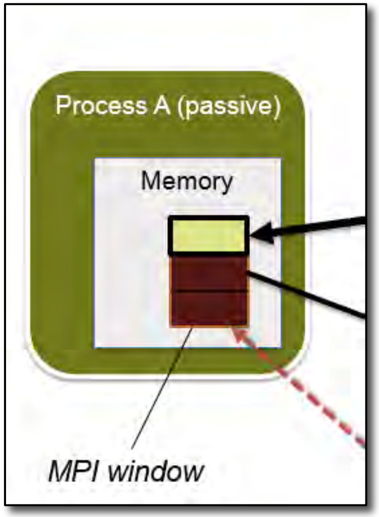


MPI-3.0 RMA SYNCHRONIZATION OVERVIEW

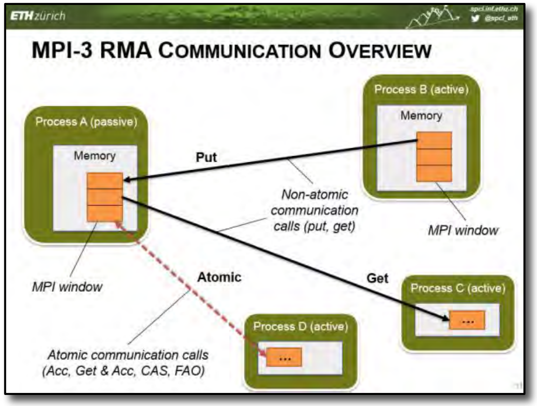


Scalable Protocols & Reference Implementation

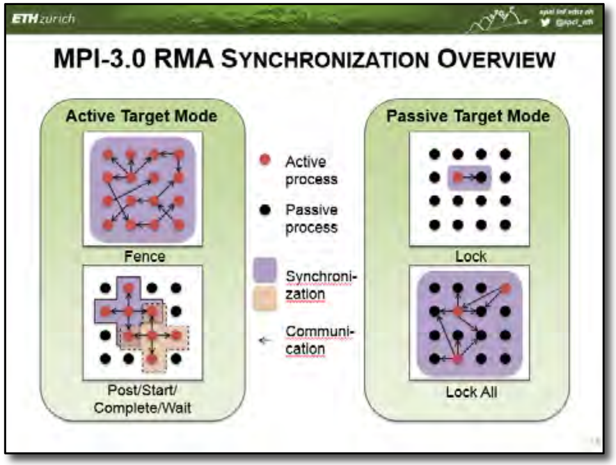
- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)
  - Window creation, communication and synchronization



Window creation



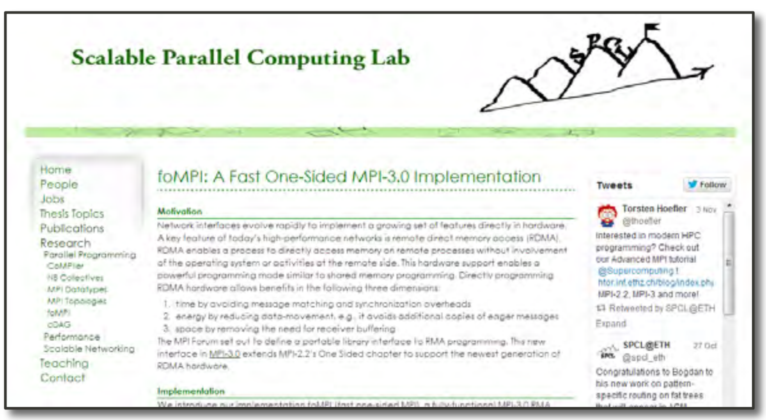
Communication

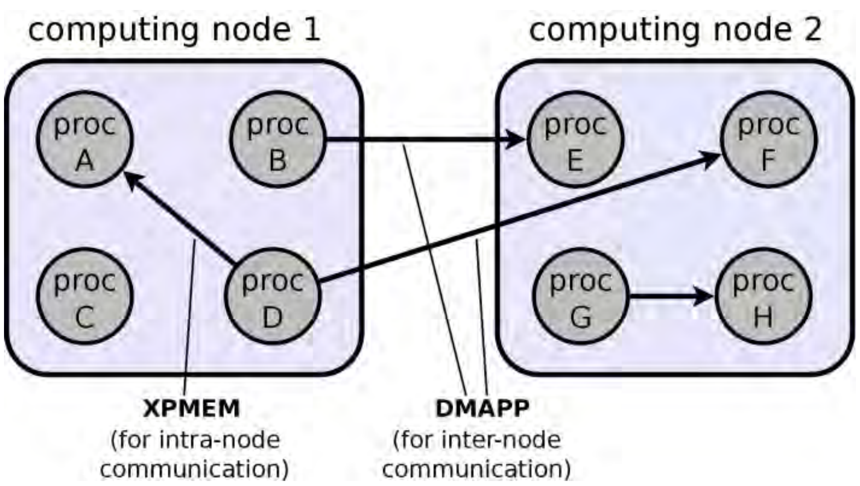


Synchronization

Scalable Protocols & Reference Implementation

- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)
  - Window creation, communication and synchronization
- foMPI, a fully functional MPI-3 RMA implementation
  - DMAPP: lowest-level networking API for Cray Gemini/Aries systems
  - XPMEM, a portable Linux kernel module

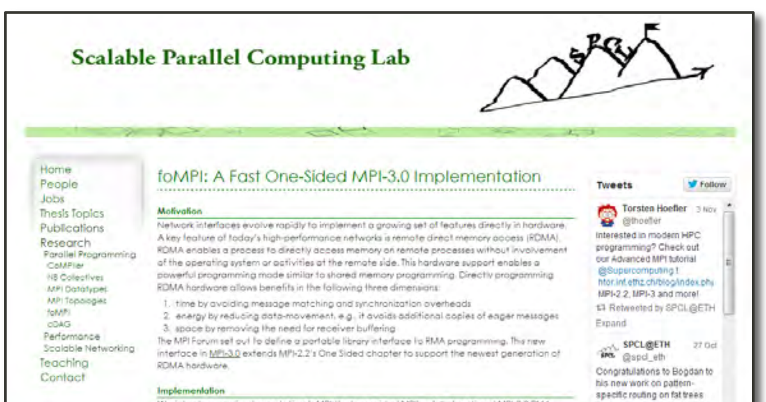


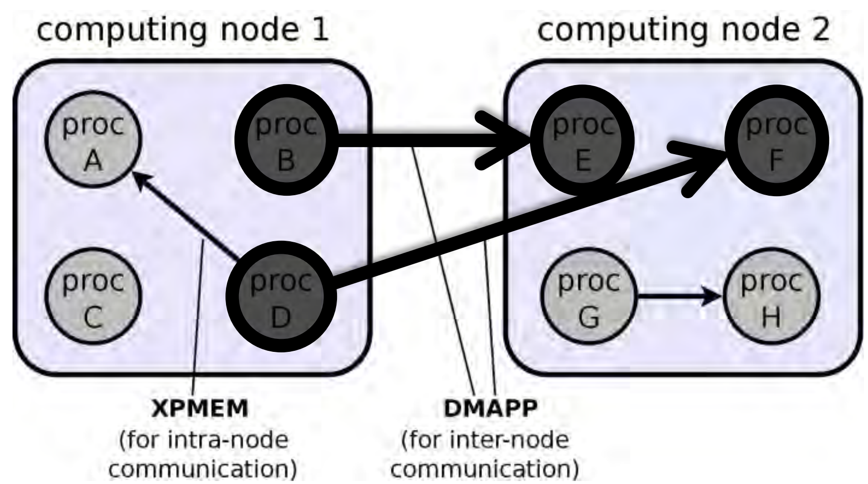


Scalable Protocols & Reference Implementation

- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)
  - Window creation, communication and synchronization

- foMPI, a fully functional MPI-3 RMA implementation
  - DMAPP: lowest-level networking API for Cray Gemini/Aries systems
  - XPMEM, a portable Linux kernel module



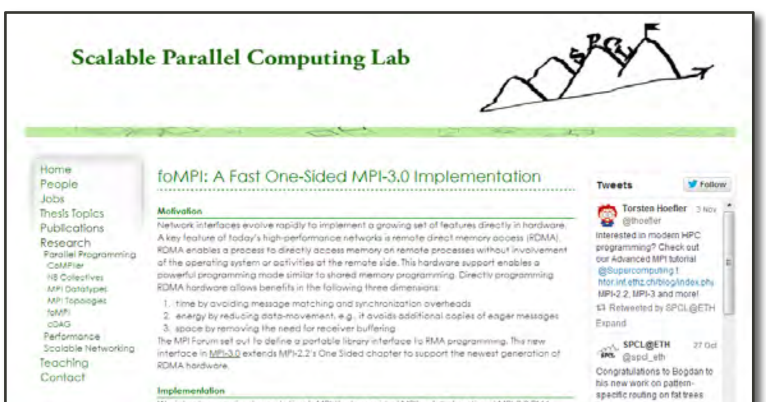


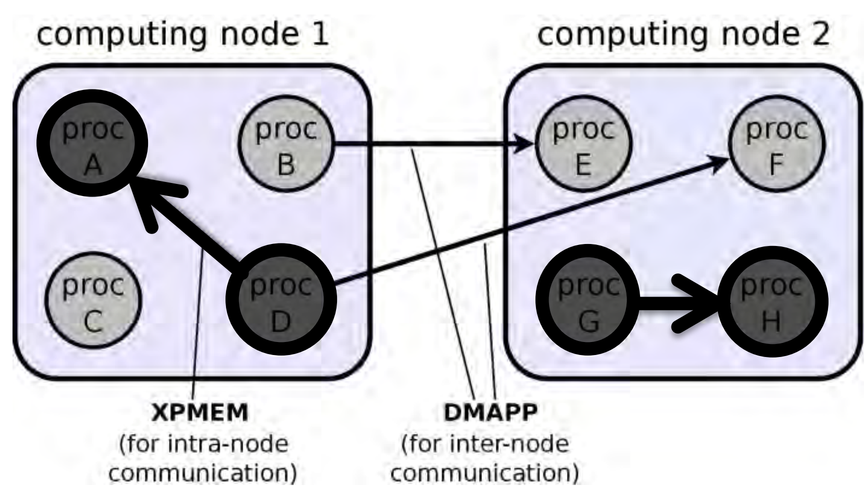
http://spcl.inf.ethz.ch/Research/Parallel_Programming/foMPI

Scalable Protocols & Reference Implementation

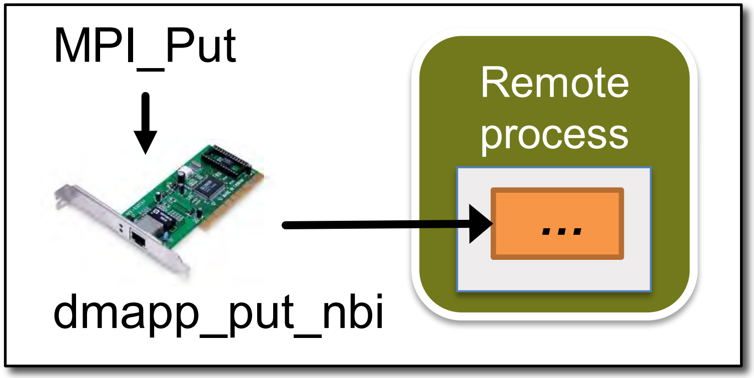
- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)
  - Window creation, communication and synchronization

- foMPI, a fully functional MPI-3 RMA implementation
  - DMAPP: lowest-level networking API for Cray Gemini/Aries systems
  - XPMEM: a portable Linux kernel module

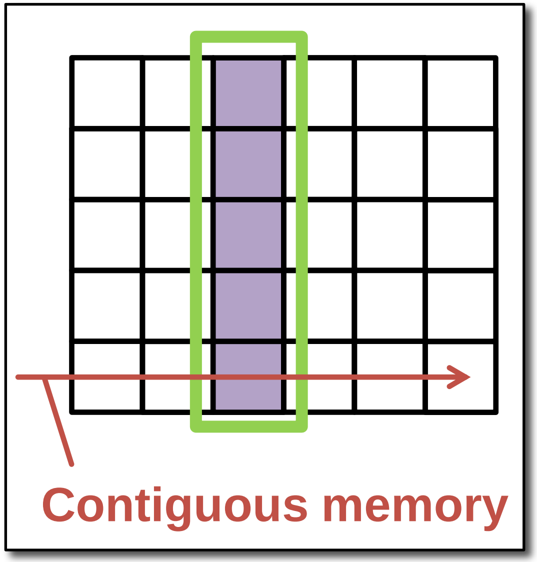


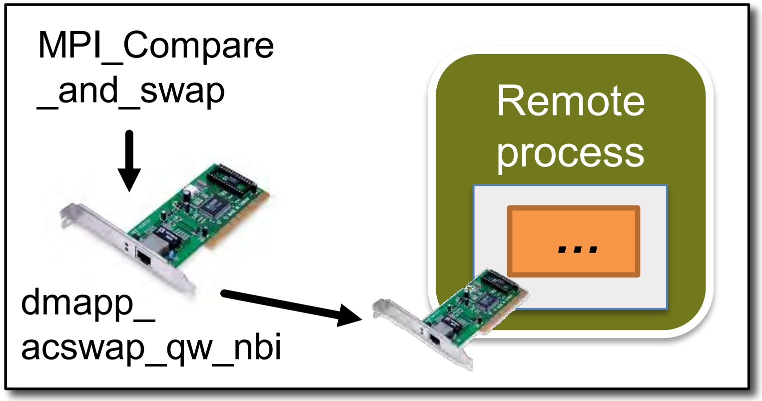


COMMUNICATION



- Put and Get:
  - Direct DMAPP put and get operations or local (blocking) memcpy (XPMEM)
- Accumulate:
  - DMAPP atomic operations for 64Bit types
  - ...or fall back to remote locking protocol
- MPI datatype handling with MPITypes library [1]
  - Fast path for contiguous data transfers of common intrinisic datatypes (e.g., MPI_DOUBLE)

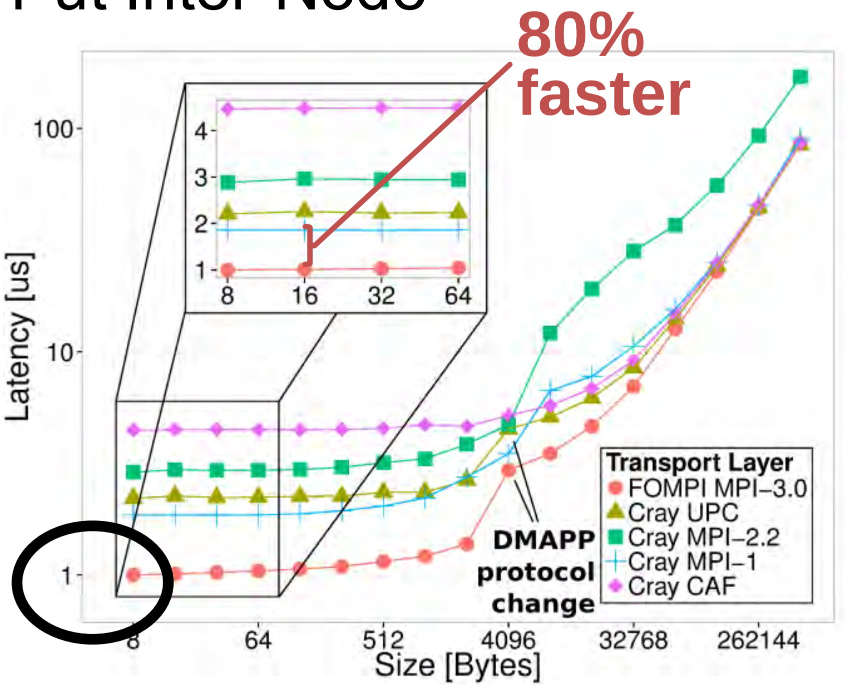




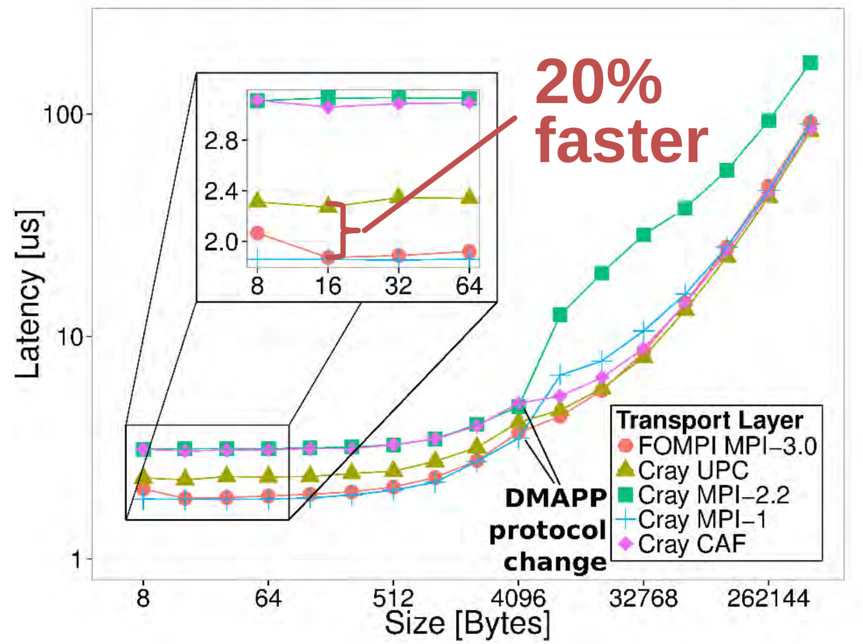
[1] Ross, Latham, Gropp, Lusk, Thakur. Processing MPI datatypes outside MPI. EuroMPI/PVM'09

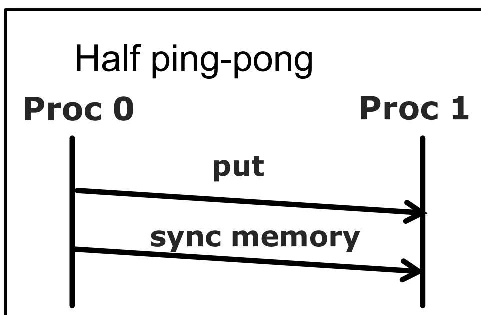
Performance Inter-node: Latency

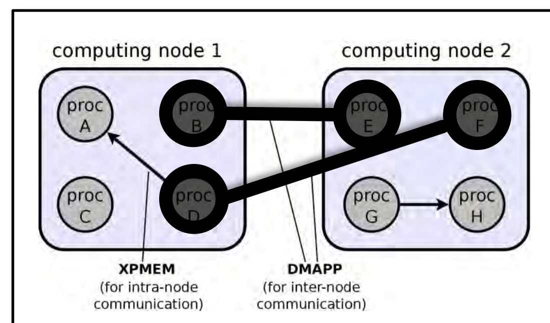
Put Inter-Node



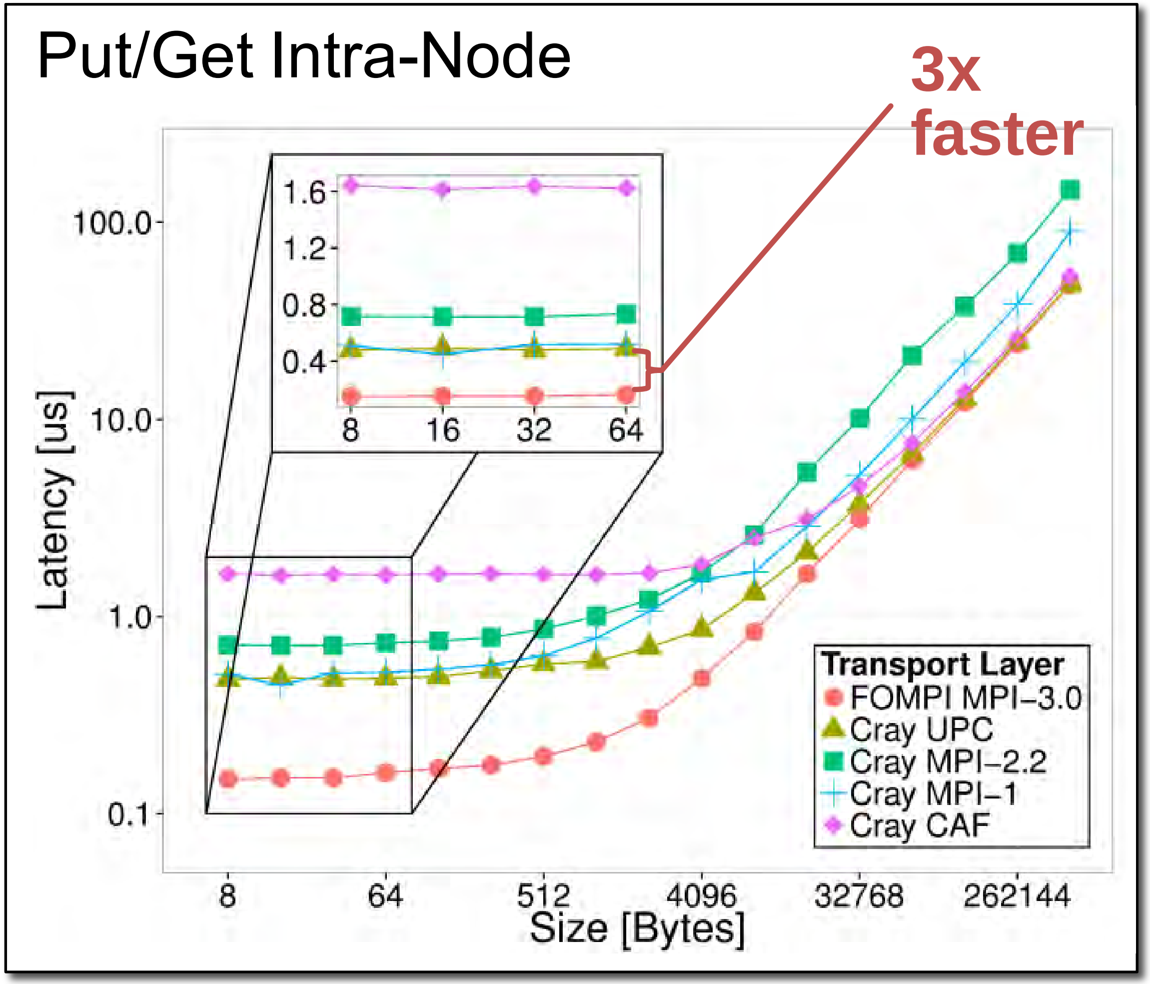
Get Inter-Node

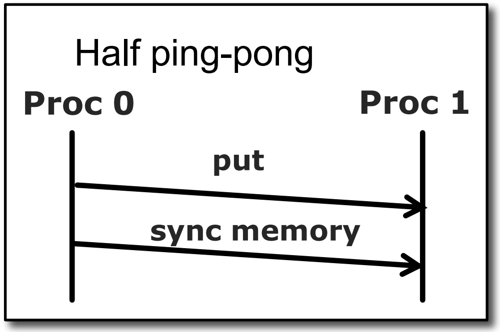


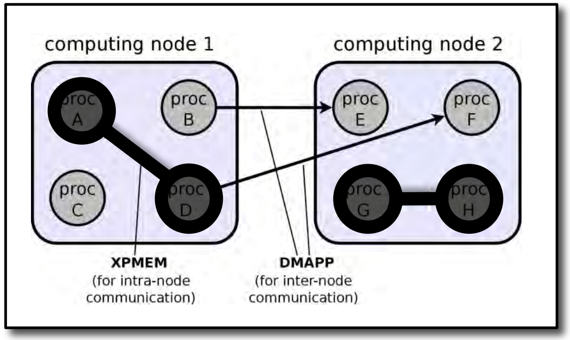




Performance Intra-node: Latency

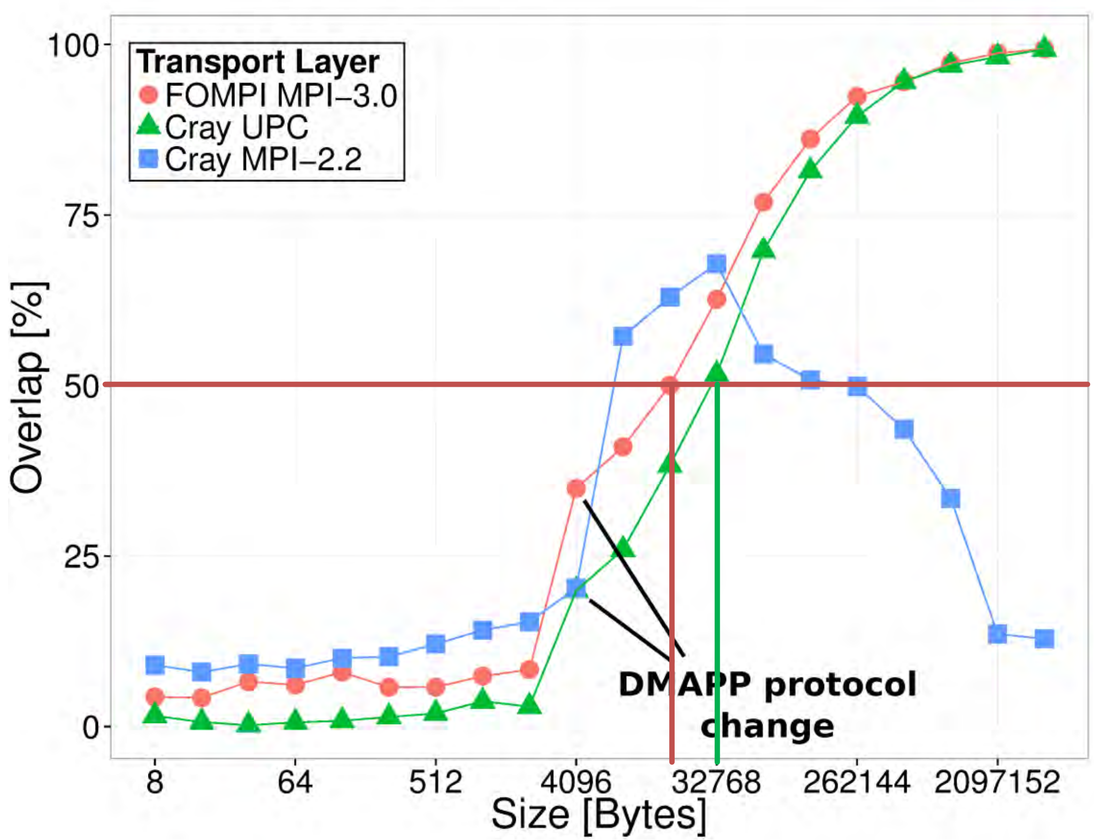


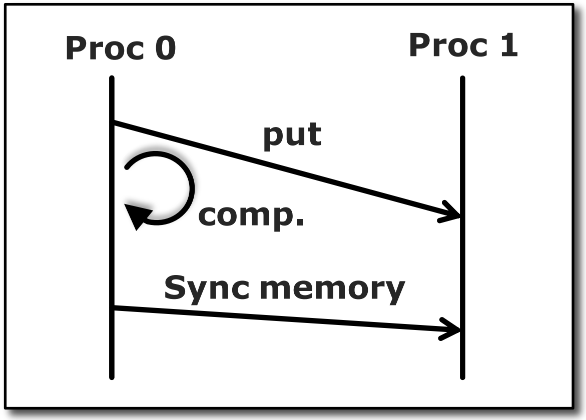




PERFORMANCE: OVERLAP

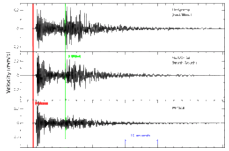
Inter-Node Overlap in %



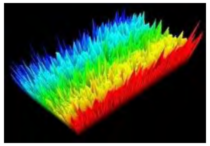


Useful for, e.g., scientific codes:

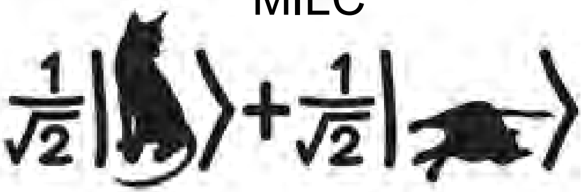
AWM-Olsen seismic



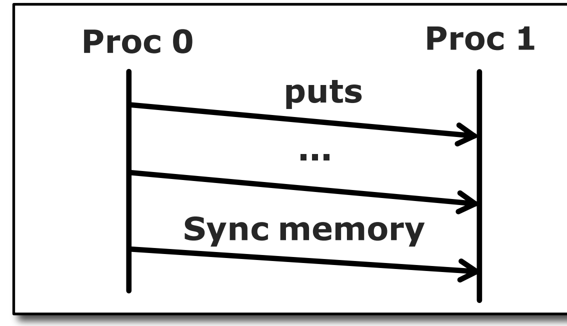
3D FFT



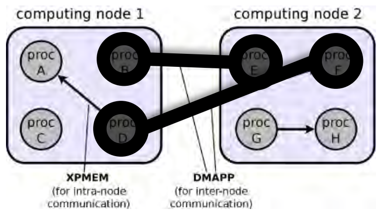
MILC

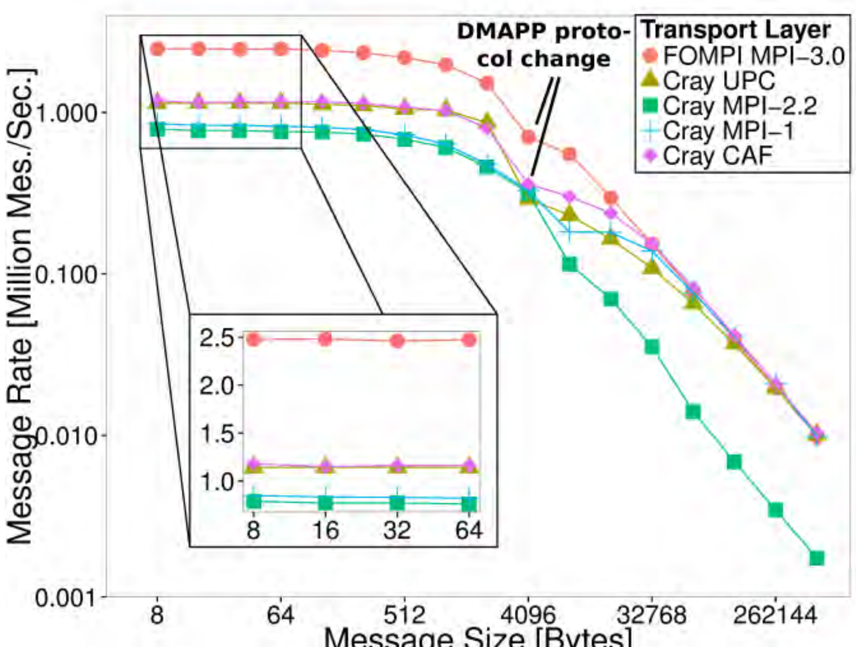


Performance: Message Rate

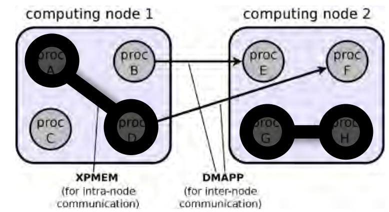


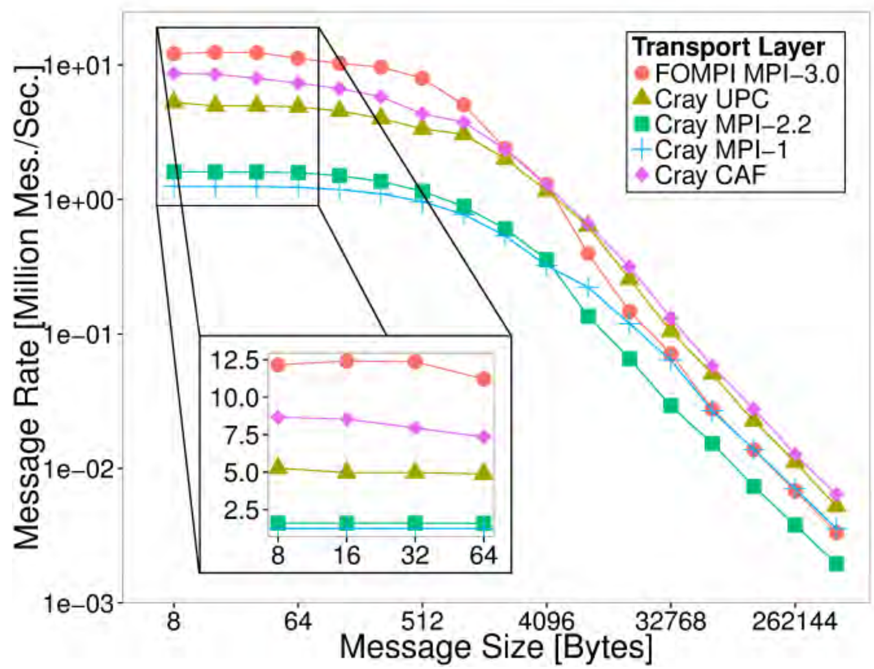
Inter-Node



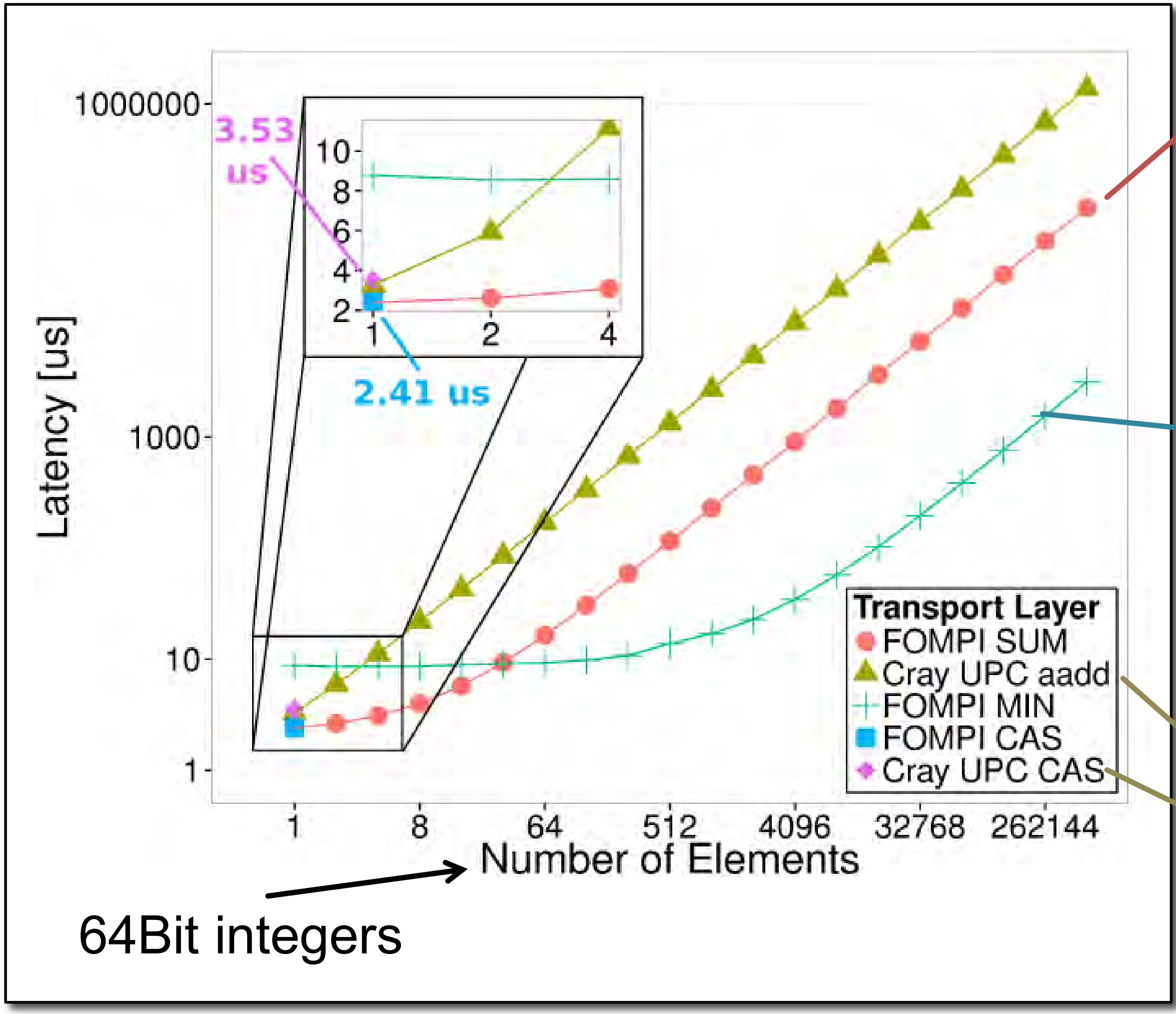


Intra-Node





PERFORMANCE: ATOMICS

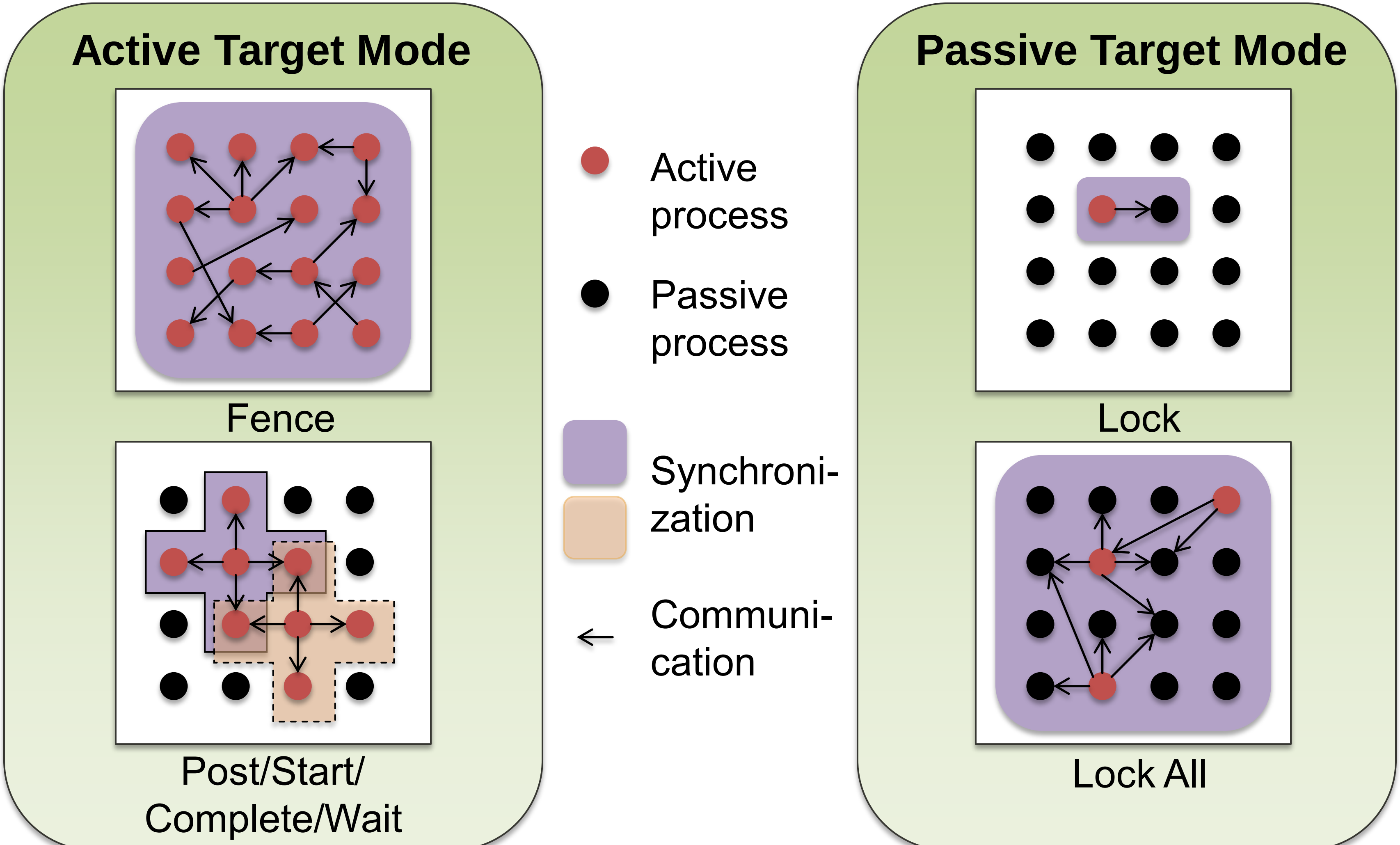


hardware-accelerated protocol:
lower latency

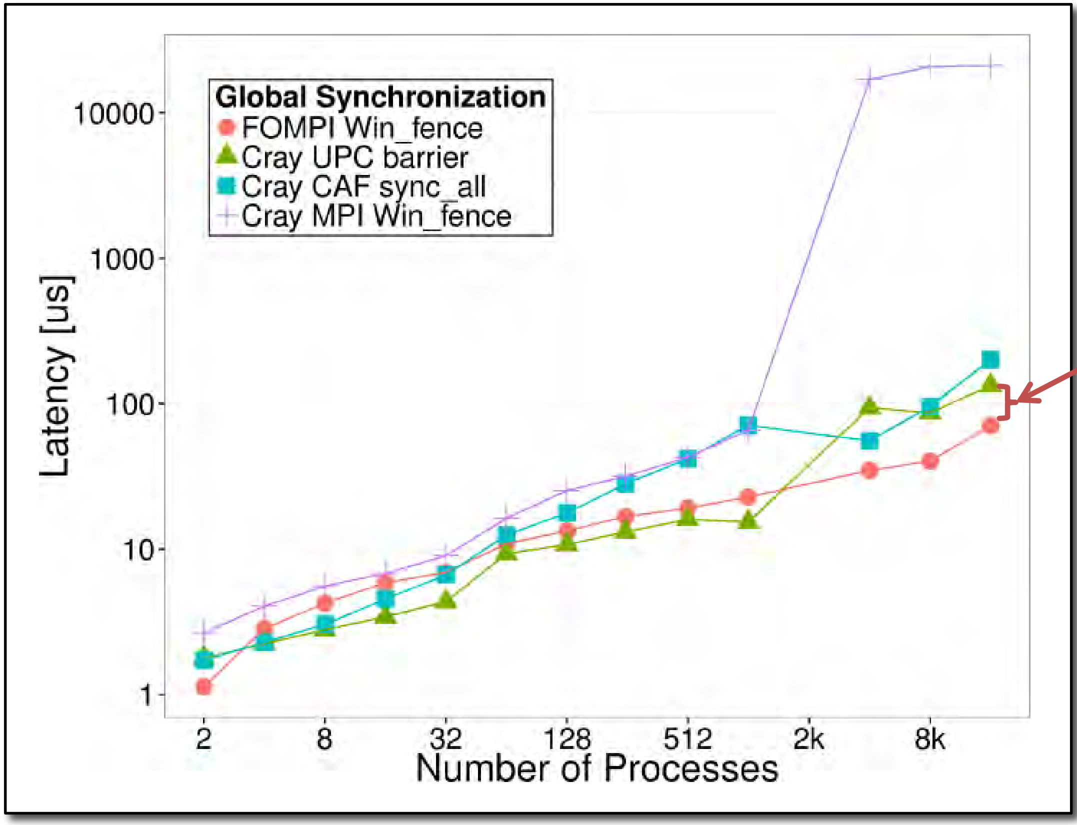
fall-back protocol:
higher bandwidth

proprietary

PART 3: SYNCHRONIZATION



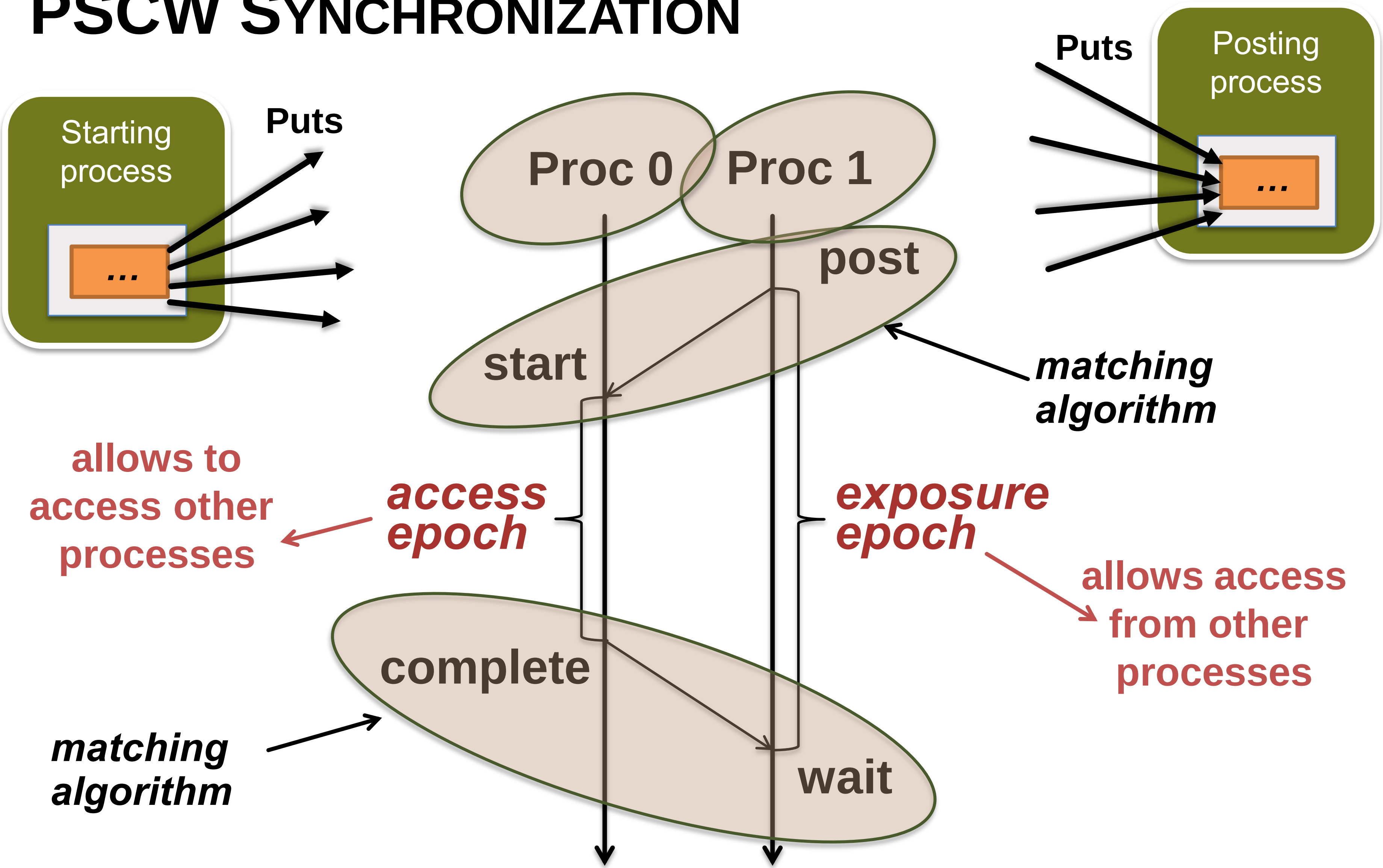
Scalable Fence Performance



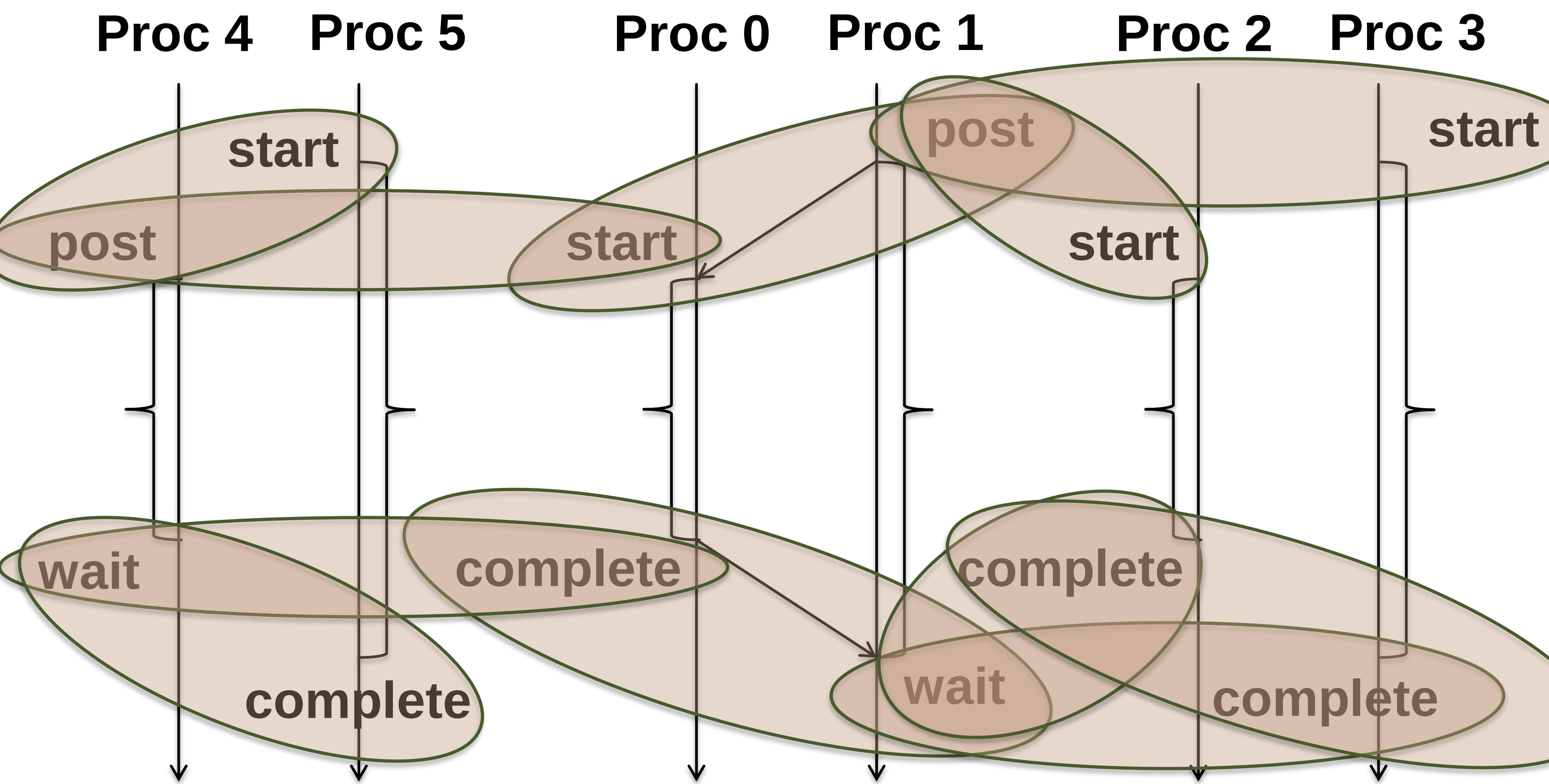
90% faster

Time bound	$\mathcal{O}(\log p)$
Memory bound	$\mathcal{O}(1)$

PSCW SYNCHRONIZATION

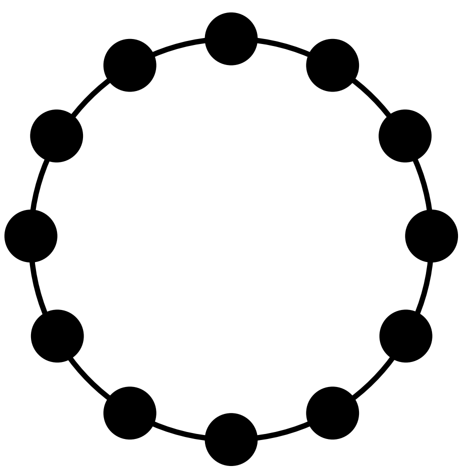


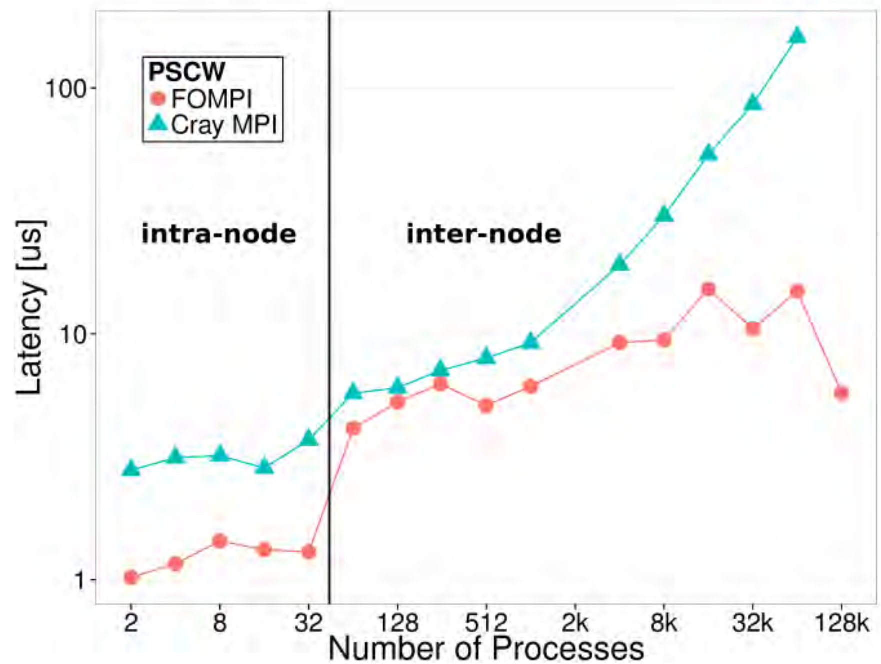
PSCW SYNCHRONIZATION



PSCW PERFORMANCE

Time bound
$\mathcal{P}_{start} = \mathcal{P}_{wait} = \mathcal{O}(1)$ $\mathcal{P}_{post} = \mathcal{P}_{complete} = \mathcal{O}(\log p)$
Memory bound
$\mathcal{O}(\log p)$ (for scalable programs)



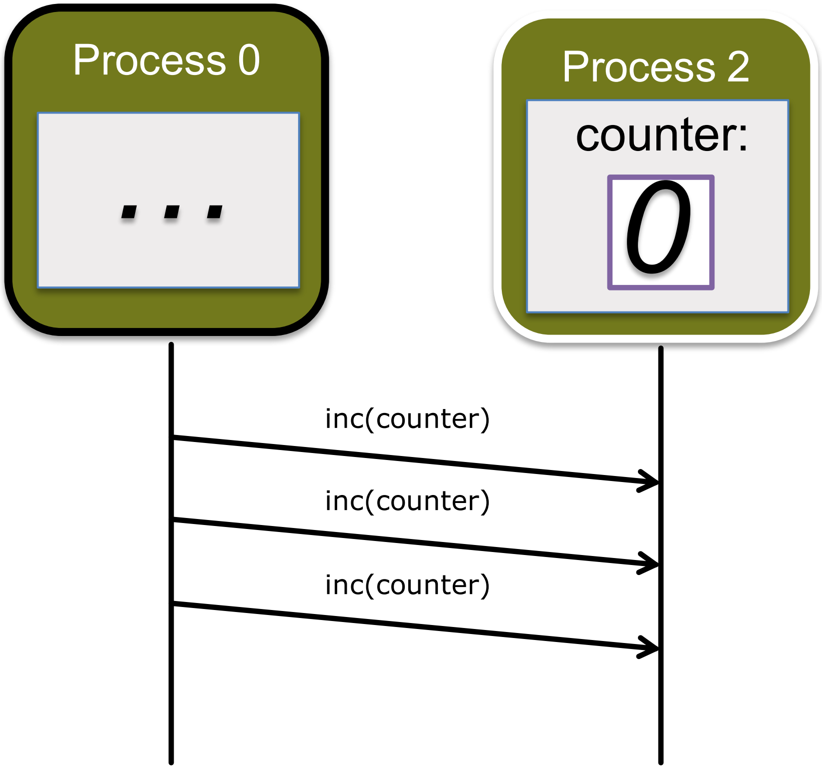


Ring Topology

Flush Synchronization

Time bound	$\mathcal{O}(1)$
Memory bound	$\mathcal{O}(1)$

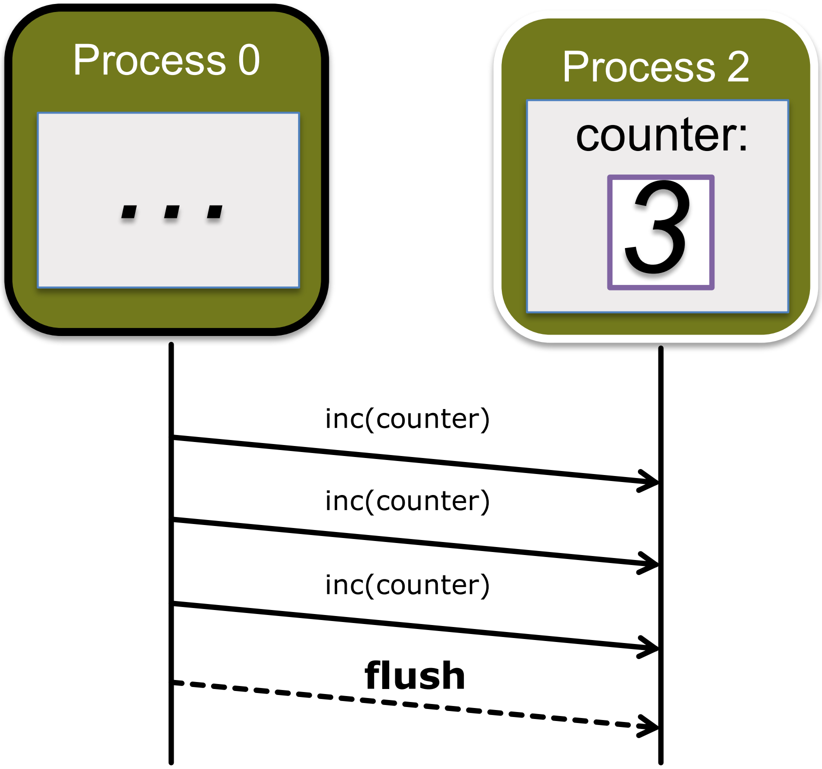
- Guarantees remote completion
- Issues a remote bulk synchronization and an x86 `mfence`
- One of the most performance critical functions, we add only 78 x86 CPU instructions to the critical path



FLUSH SYNCHRONIZATION

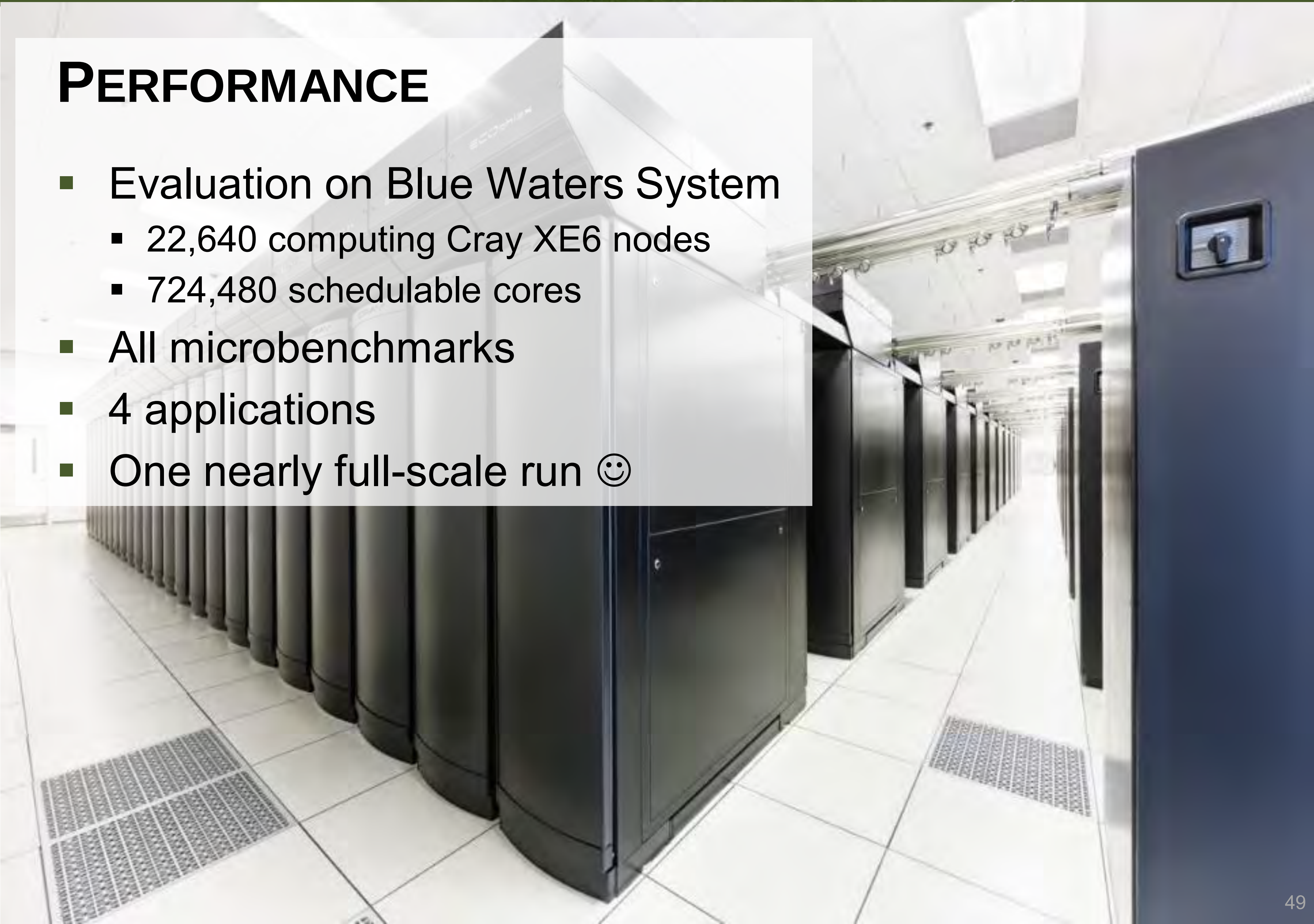
Time bound	$\mathcal{O}(1)$
Memory bound	$\mathcal{O}(1)$

- Guarantees remote completion
- Issues a remote bulk synchronization and an x86 `mfence`
- One of the most performance critical functions, we add only 78 x86 CPU instructions to the critical path



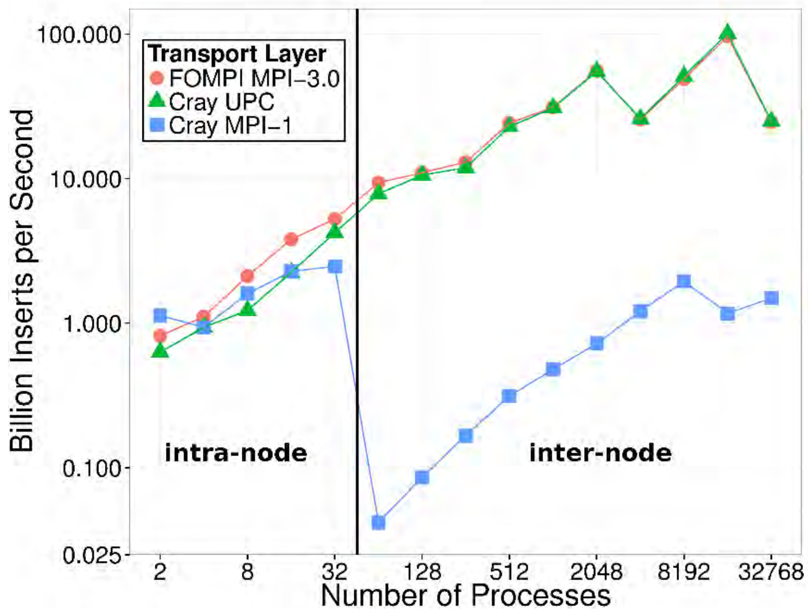
PERFORMANCE

- Evaluation on Blue Waters System
  - 22,640 computing Cray XE6 nodes
  - 724,480 schedulable cores
- All microbenchmarks
- 4 applications
- One nearly full-scale run ☺

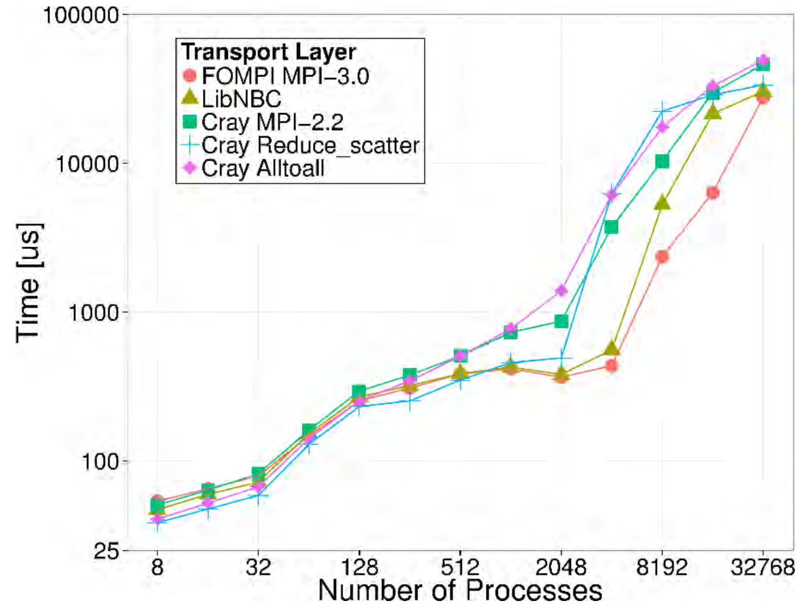


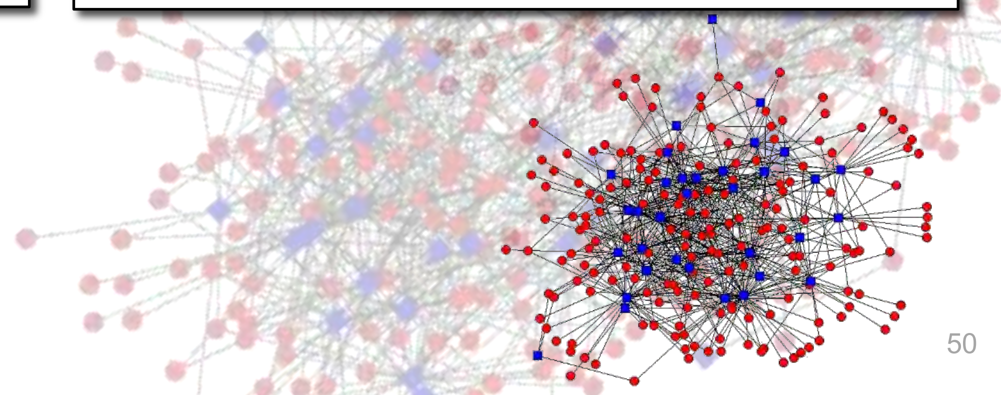
PERFORMANCE: MOTIF APPLICATIONS

Key/Value Store: Random Inserts per Second



Dynamic Sparse Data Exchange (DSDE) with 6 neighbors

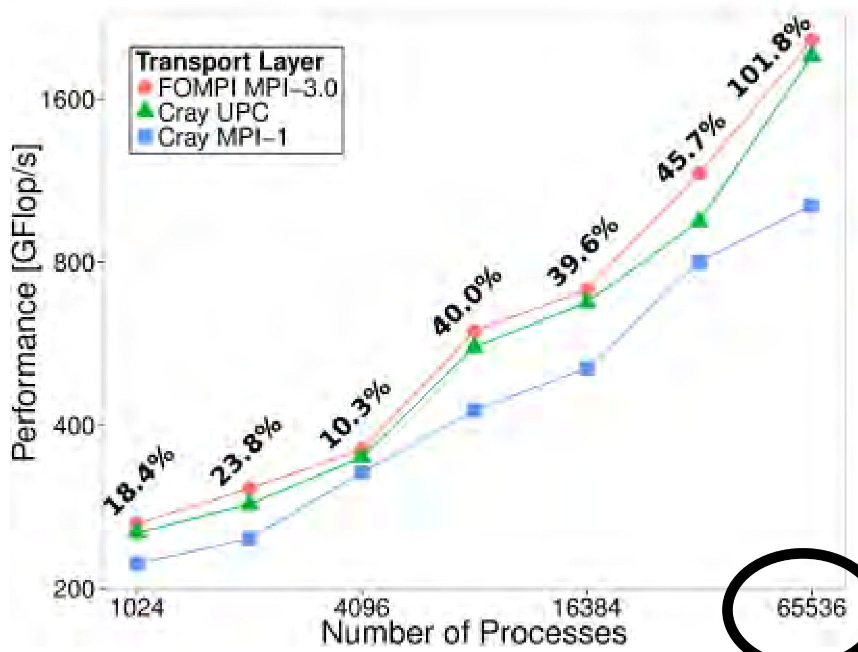




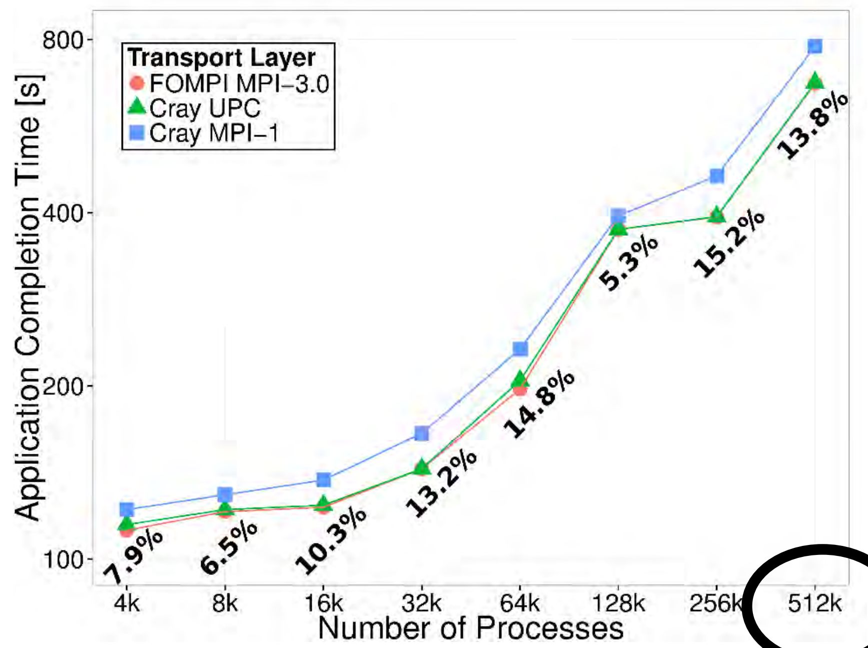
PERFORMANCE: APPLICATIONS

Annotations represent performance gain of foMPI over Cray MPI-1.

NAS 3D FFT [1] Performance



MILC [2] Application Execution Time



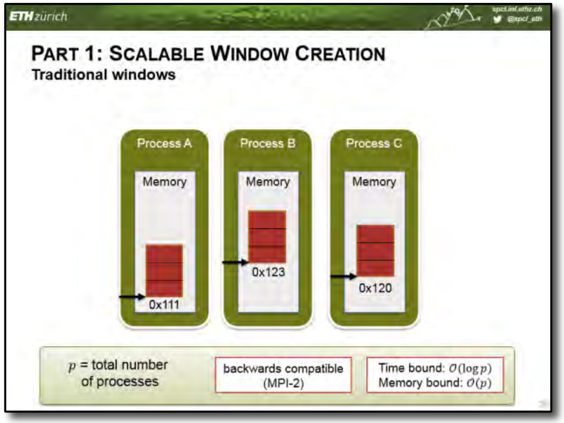
scale to 65k procs

scale to 512k procs

[1] Nishtala et al. Scaling communication-intensive applications on BlueGene/P using one-sided communication and overlap. IPDPS'09
[2] Shan et al. Accelerating applications at scale using one-sided communication. PGAS'12

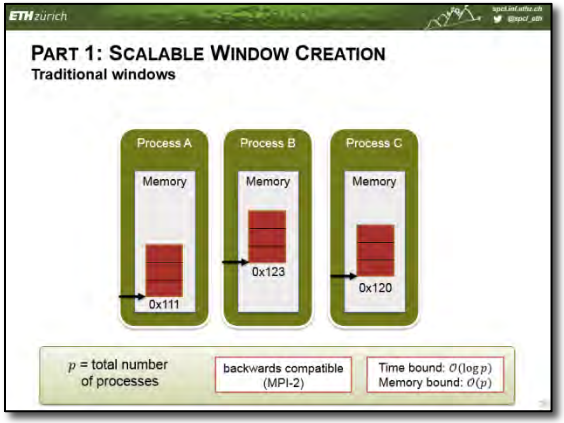
CONCLUSIONS & SUMMARY

CONCLUSIONS & SUMMARY

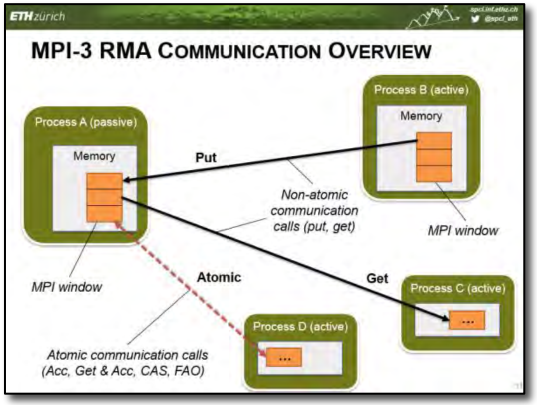


1. MPI window creation routines

CONCLUSIONS & SUMMARY

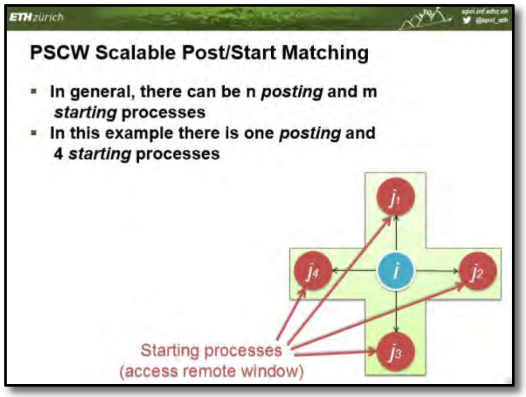


1. MPI window creation routines

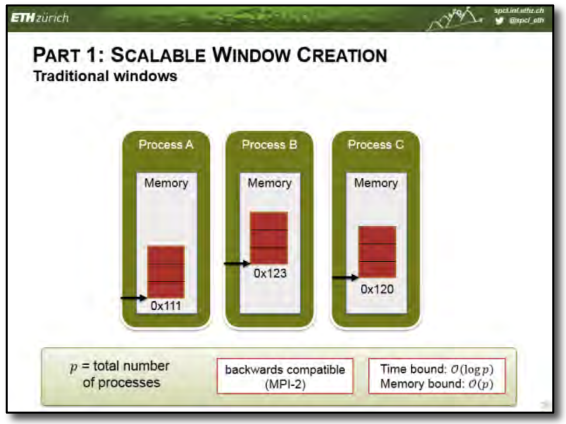


2. Non-atomic & atomic communication

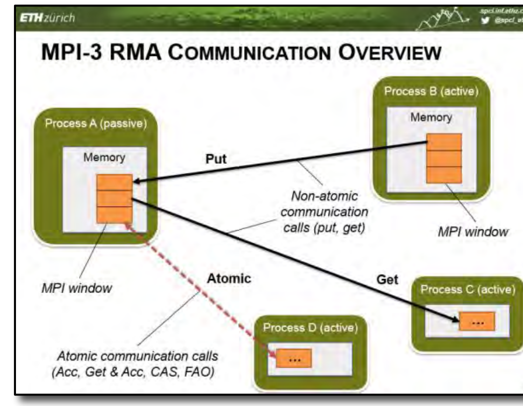
Conclusions & Summary



3. Fence / PSCW

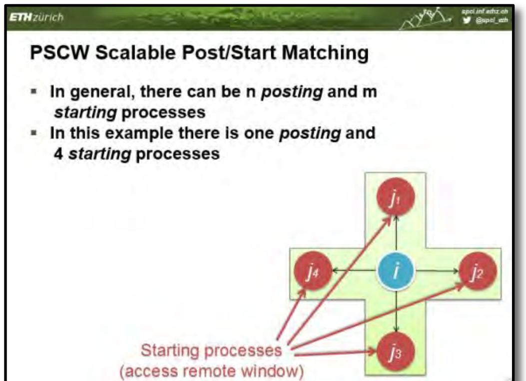


1. MPI window creation routines

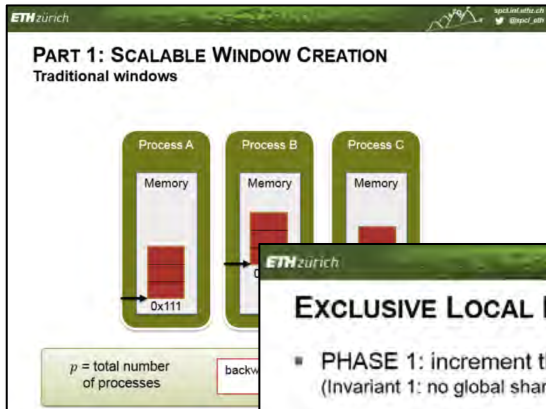


2. Non-atomic & atomic communication

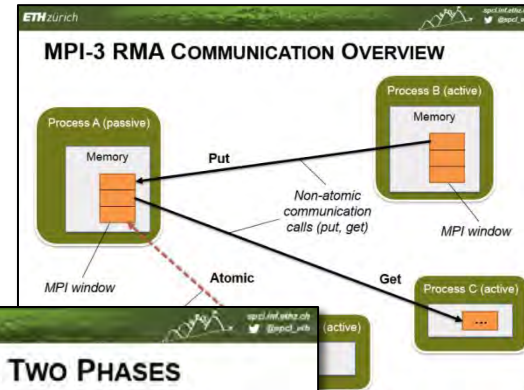
CONCLUSIONS & SUMMARY



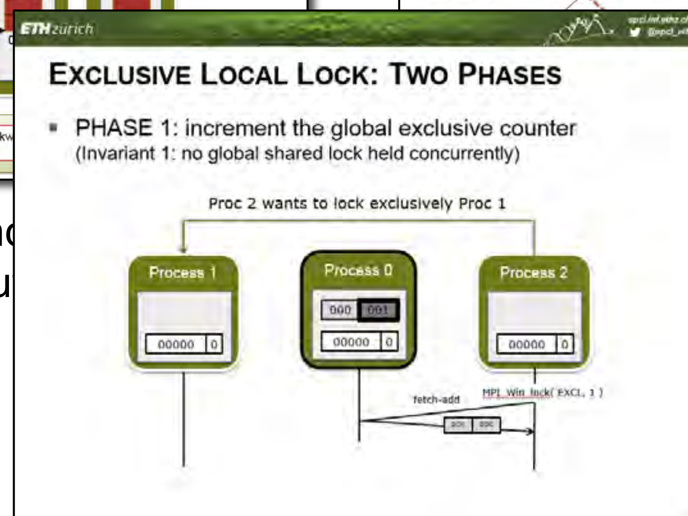
3. Fence / PSCW



1. MPI win
rou

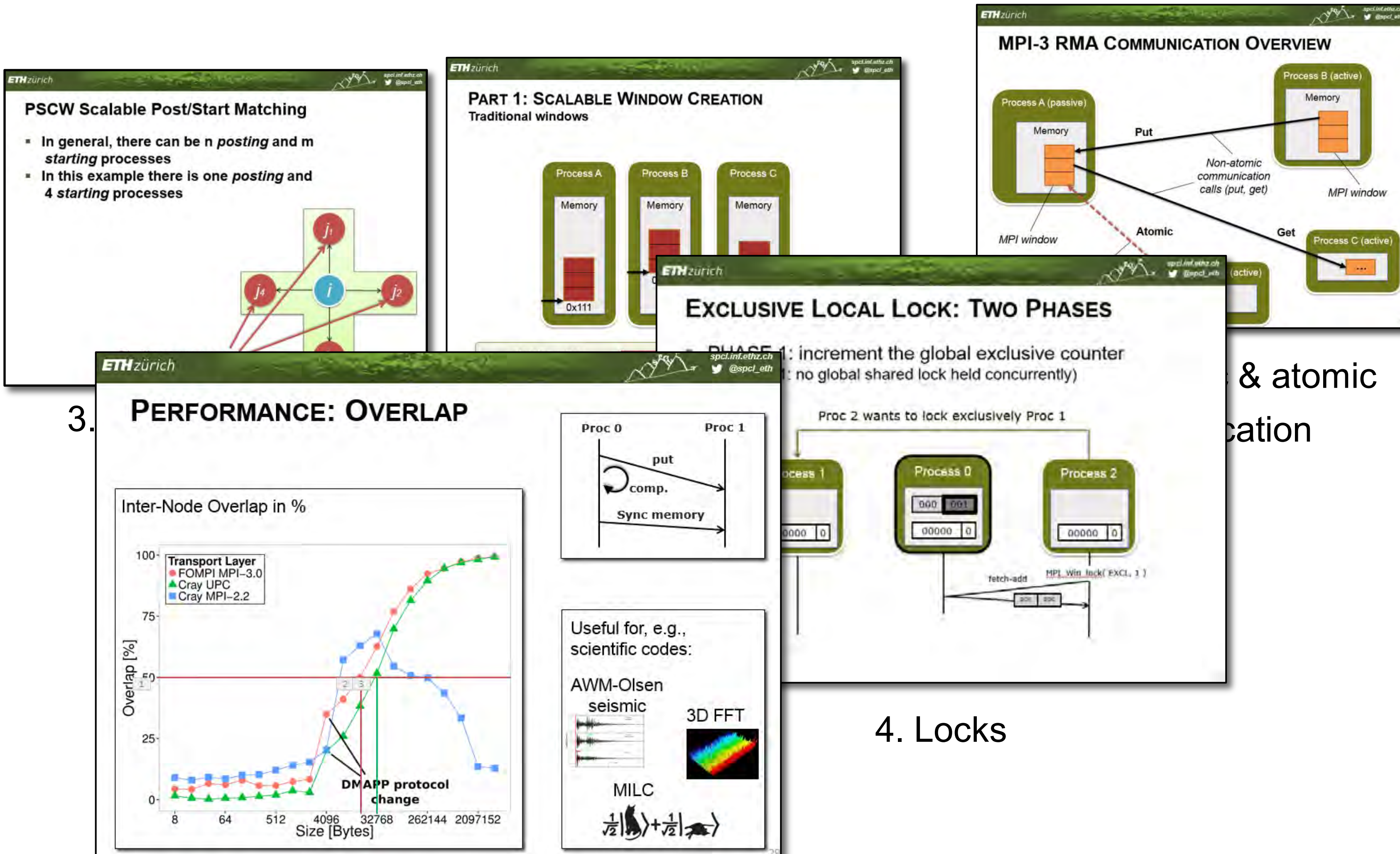


& atomic
cation



4. Locks

Conclusions & Summary



3.

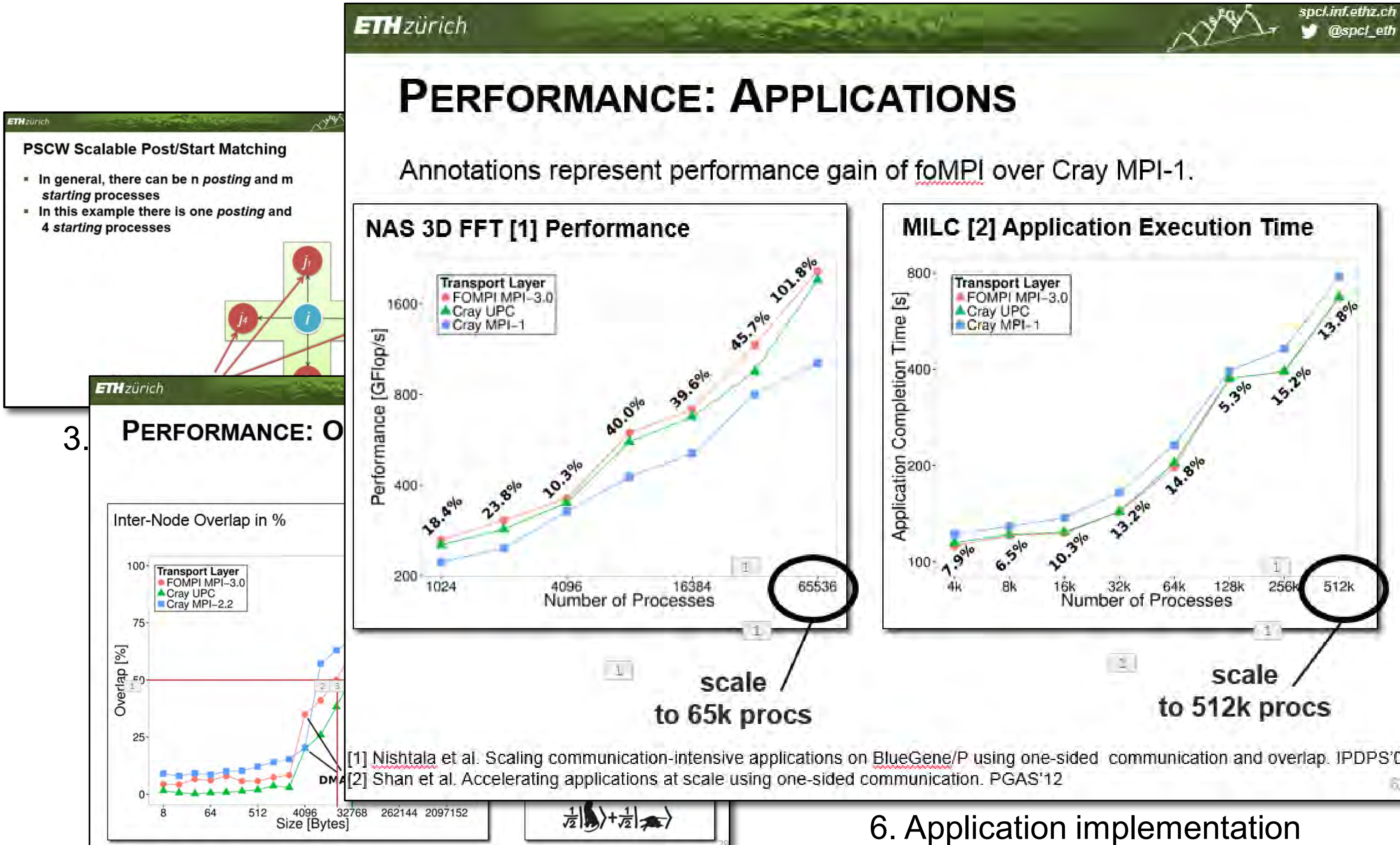
& atomic

cation

4. Locks

5. foMPI reference implementation

Conclusions & Summary



3.

5. foMPI reference implementation

6. Application implementation & evaluation

Conclusions & Summary

PSCW Scalable Po

- In general, there ca *starting* processes
- In this example the 4 *starting* process

3. PE

Performance Modelling

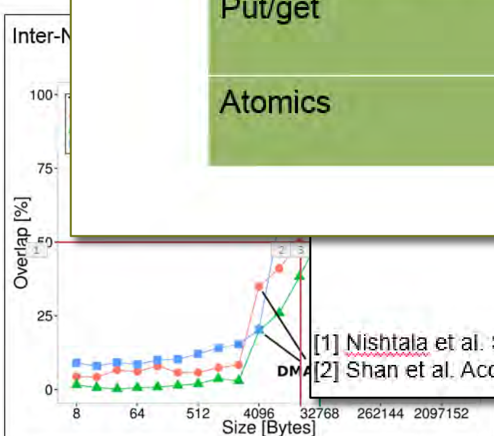
Performance functions for synchronization protocols

Fence	$\mathcal{P}_{fence} = 2.9\mu s \cdot \log_2(p)$
PSCW	$\mathcal{P}_{start} = 0.7\mu s, \mathcal{P}_{wait} = 1.8\mu s$ $\mathcal{P}_{post} = \mathcal{P}_{complete} = 350ns \cdot k$
Locks	$\mathcal{P}_{lock,excl} = 5.4\mu s$ $\mathcal{P}_{lock,shrd} = \mathcal{P}_{lock\_all} = 2.7\mu s$ $\mathcal{P}_{unlock} = \mathcal{P}_{unlock\_all} = 0.4\mu s$ $\mathcal{P}_{flush} = 76ns$ $\mathcal{P}_{sync} = 17ns$

Performance functions for communication protocols

Put/get	$\mathcal{P}_{put} = 0.16ns \cdot s + 1\mu s$ $\mathcal{P}_{get} = 0.17ns \cdot s + 1.9\mu s$
Atomics	$\mathcal{P}_{acc,sum} = 28ns \cdot s + 2.4\mu s$ $\mathcal{P}_{acc,min} = 0.8ns \cdot s + 7.3\mu s$

MPI-1.



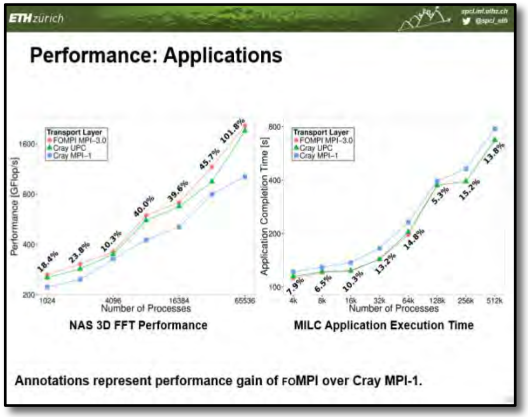
scale to 65k procs

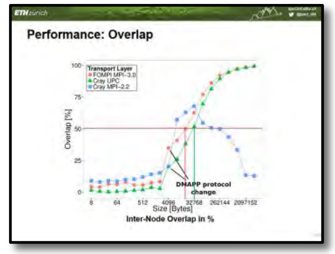
scale to 512k procs

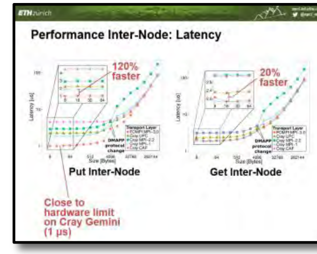
[1] Nishtala et al. Scaling communication-intensive applications on BlueGene/P using one-sided communication and overlap. IPDPS'09
[2] Shan et al. Accelerating applications at scale using one-sided communication. PGAS'12

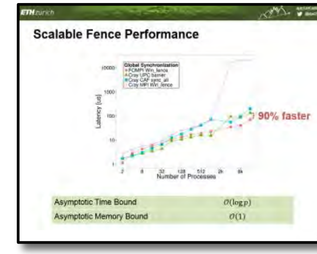
5. foMPI reference implementation

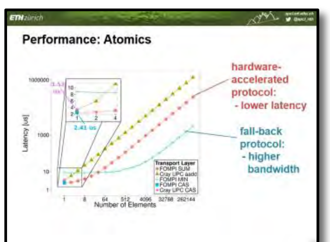
6. Application implementation & evaluation











Thank you
for your attention