Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided

ROBERT GERSTENBERGER, MACIEJ BESTA, TORSTEN HOEFLER

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

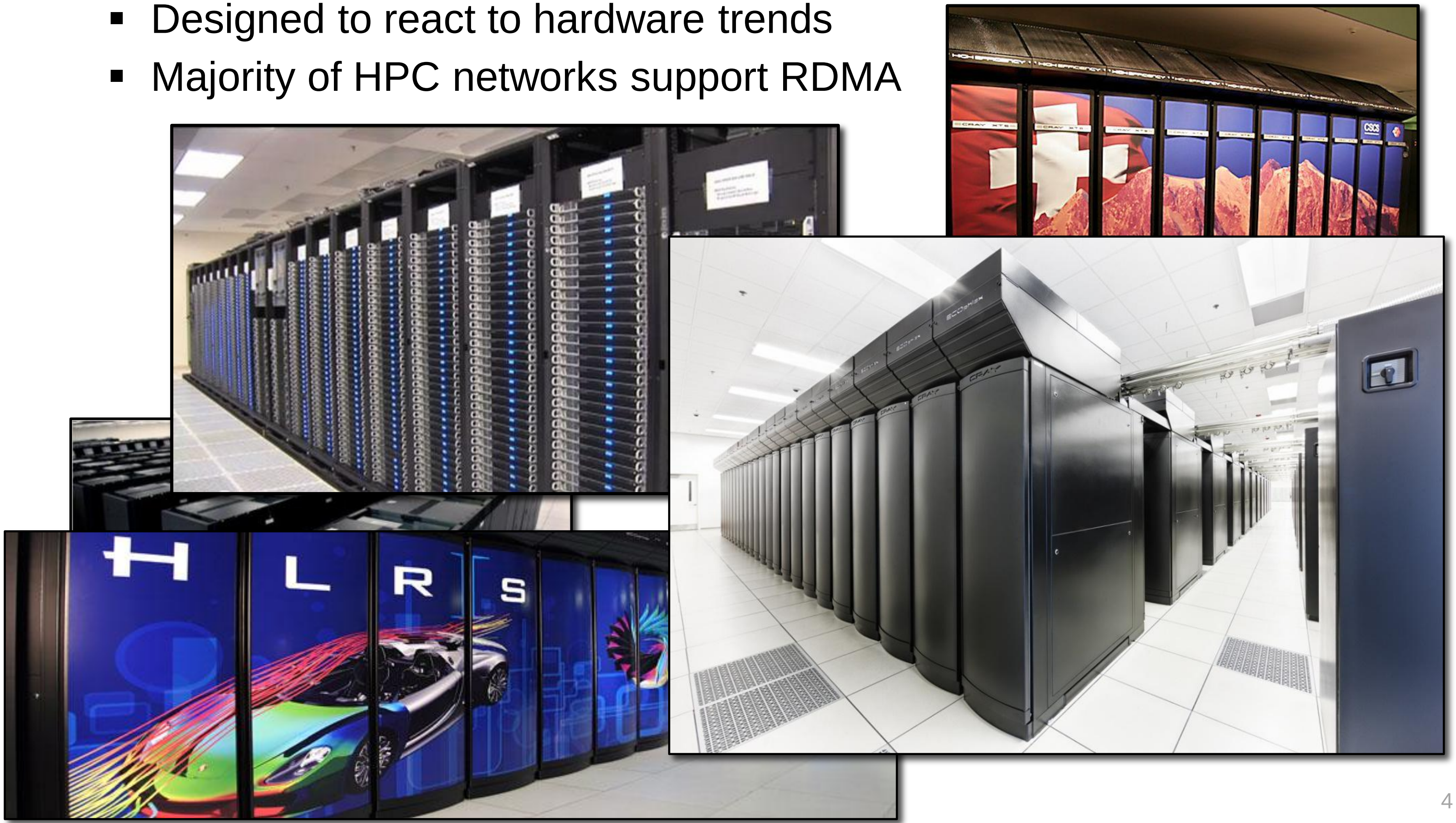
MPI-3.0 RMA

- 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

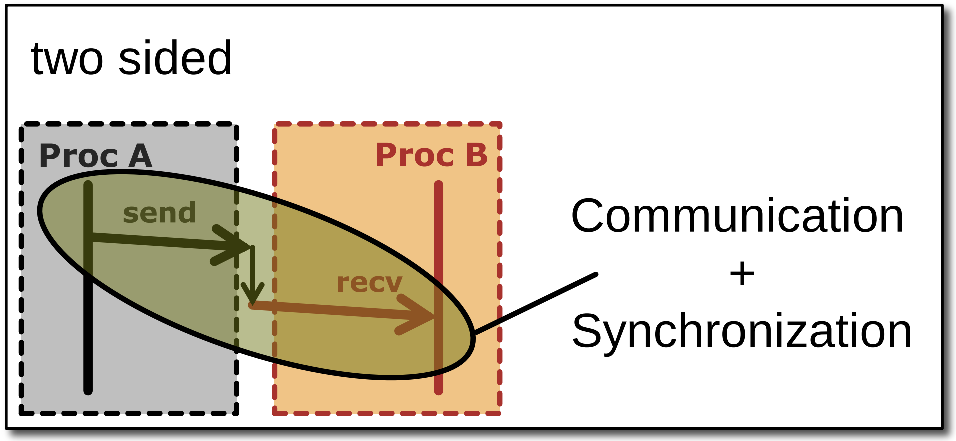
MPI-3.0 RMA

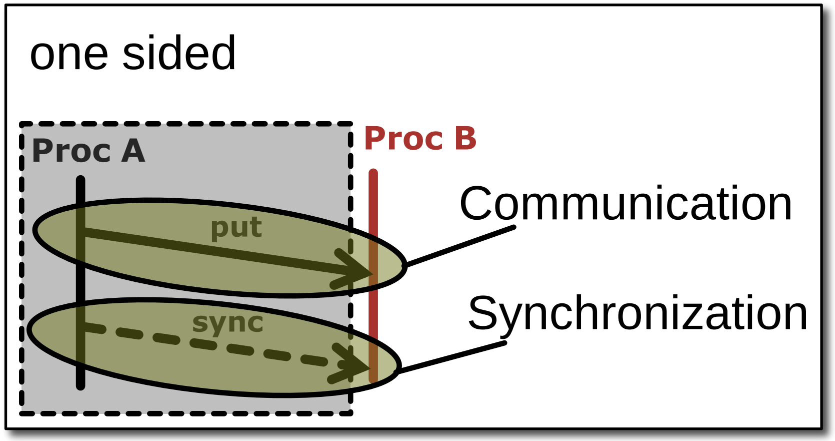
- MPI-3.0 supports RMA (“MPI One Sided”)
  - Designed to react to hardware trends
  - Majority of HPC networks support RDMA



MPI-3.0 RMA

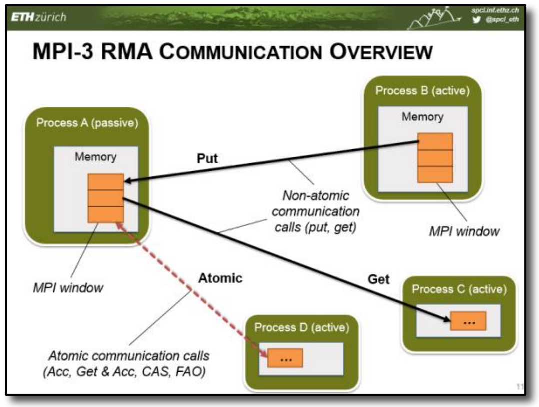
- MPI-3.0 supports RMA ("MPI One Sided")
  - Designed to react to hardware trends
  - Majority of HPC networks support RDMA
- Communication is „one sided" (no involvement of destination)
- RMA decouples communication & synchronization
  - Different from message passing



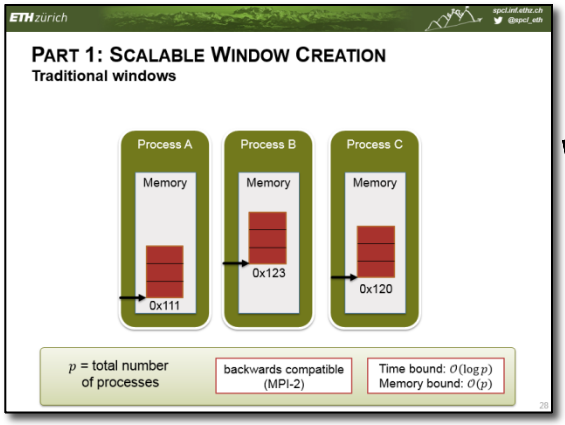


[1] http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

PRESENTATION OVERVIEW

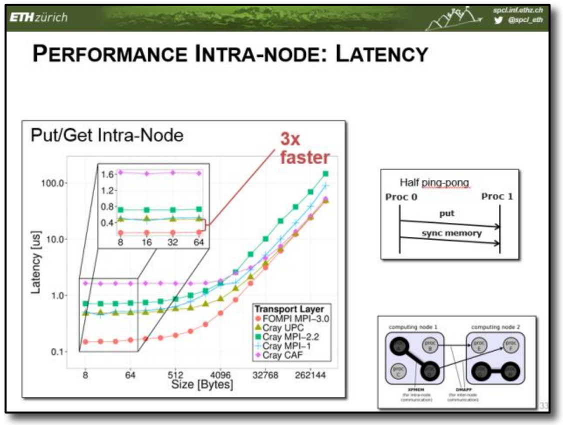

MPI-3 RMA COMMUNICATION OVERVIEW

1. Overview of three MPI-3 RMA concepts

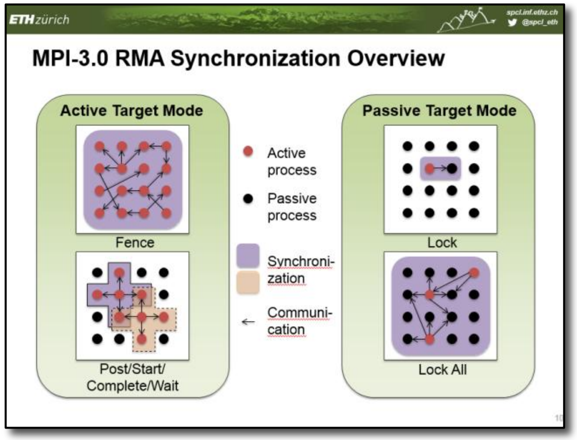

PART 1: SCALABLE WINDOW CREATION
Traditional windows

Process A
Memory
0x111
Process B
Memory
0x123
Process C
Memory
0x120
p = total number of processes
backwards compatible (MPI-2)
Time bound: $\mathcal{O}(\log p)$
Memory bound: $\mathcal{O}(p)$

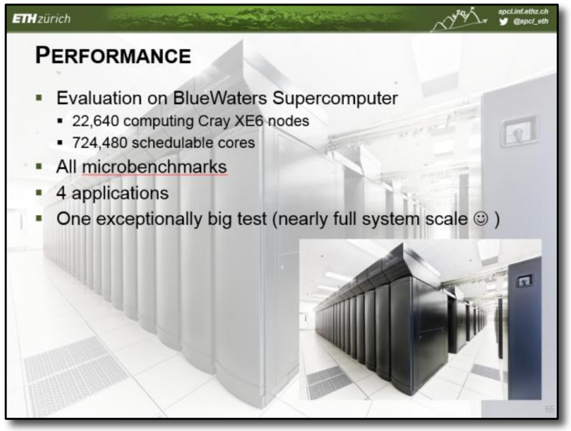
2. MPI window creation


PERFORMANCE INTRA-NODE: LATENCY

3. Communication


MPI-3.0 RMA Synchronization Overview

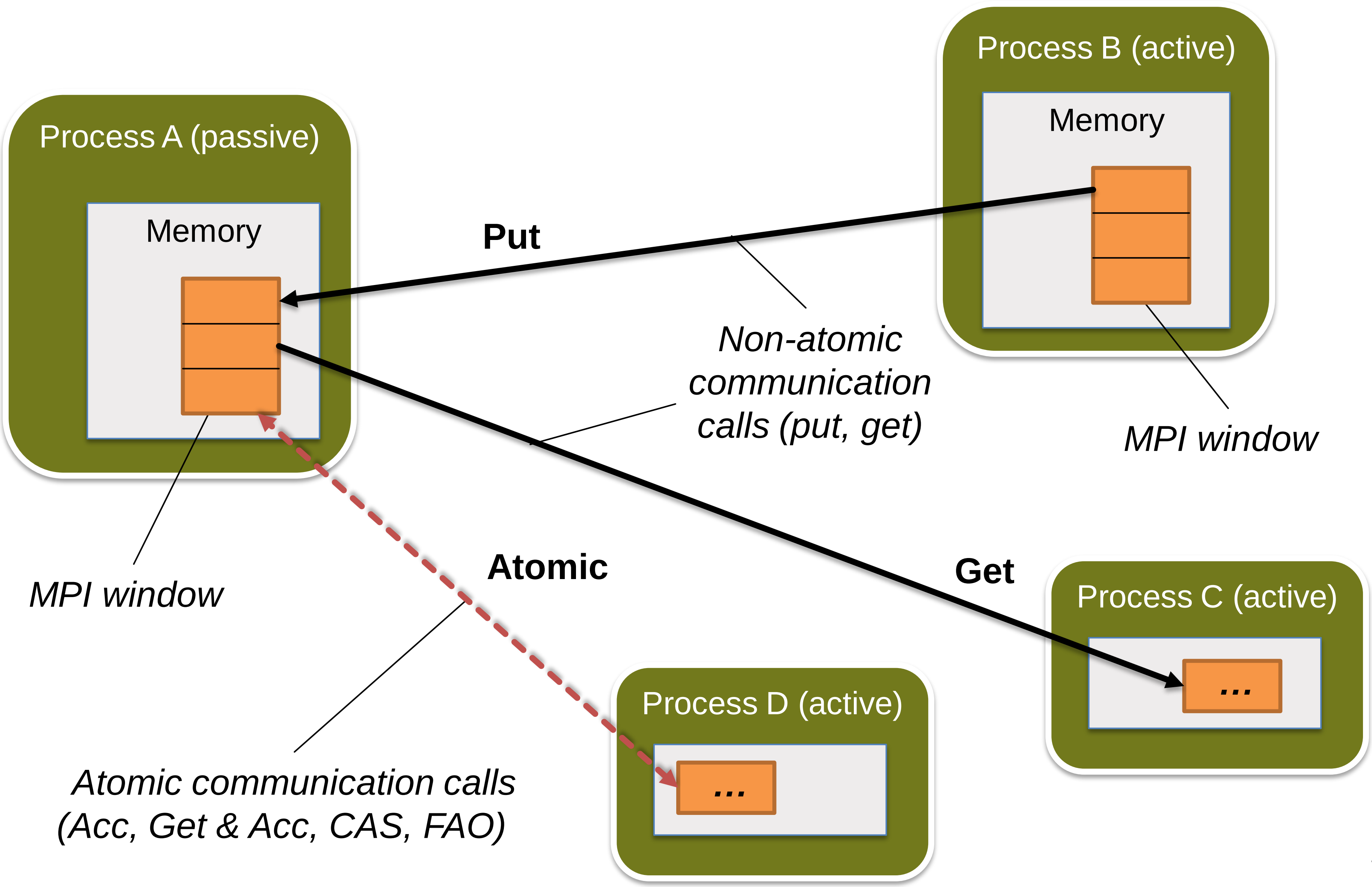
4. Synchronization


PERFORMANCE

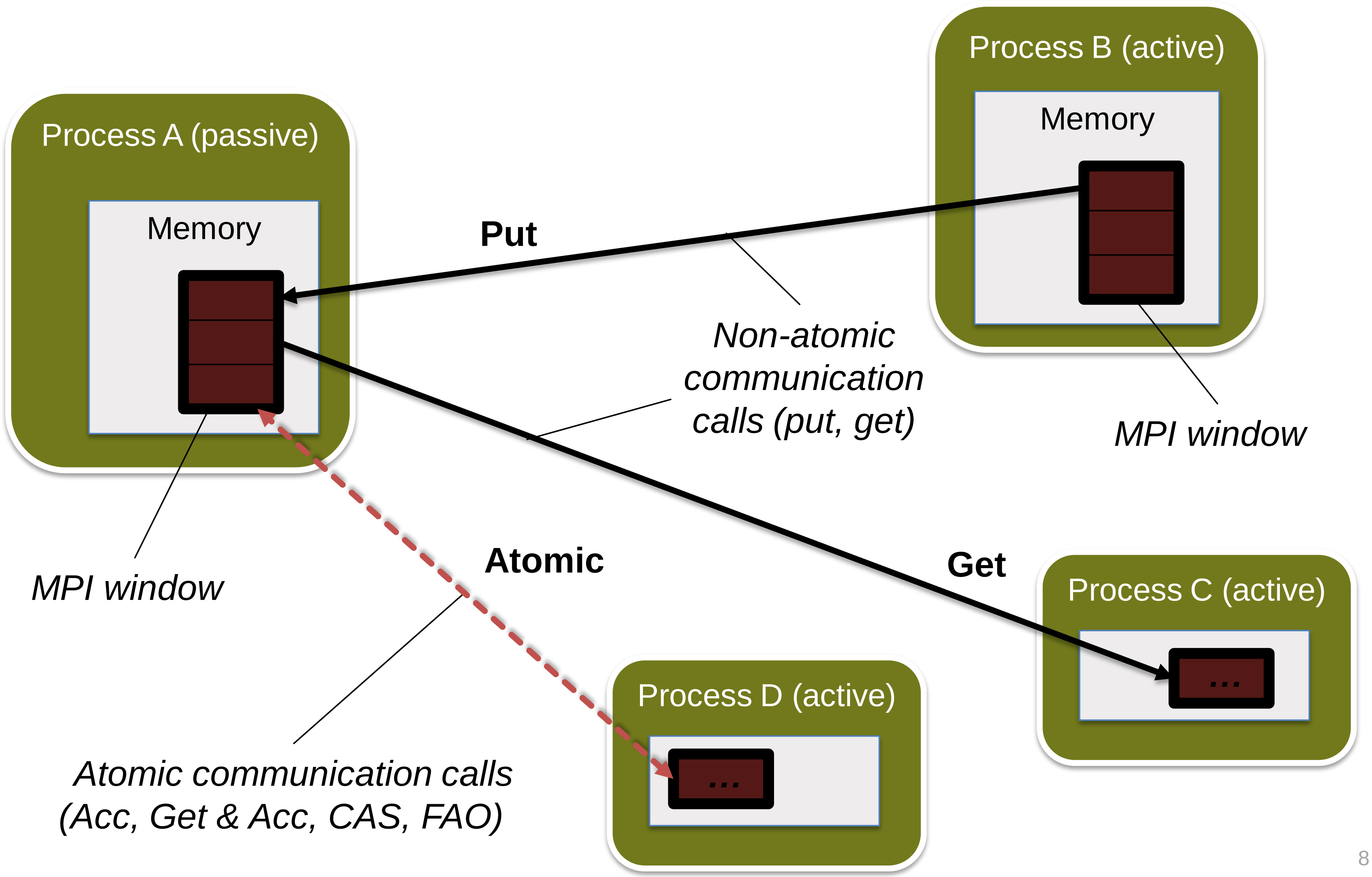
- Evaluation on BlueWaters Supercomputer
  - 22,640 computing Cray XE6 nodes
  - 724,480 schedulable cores
- All microbenchmarks
- 4 applications
- One exceptionally big test (nearly full system scale ☺)

5. Application evaluation

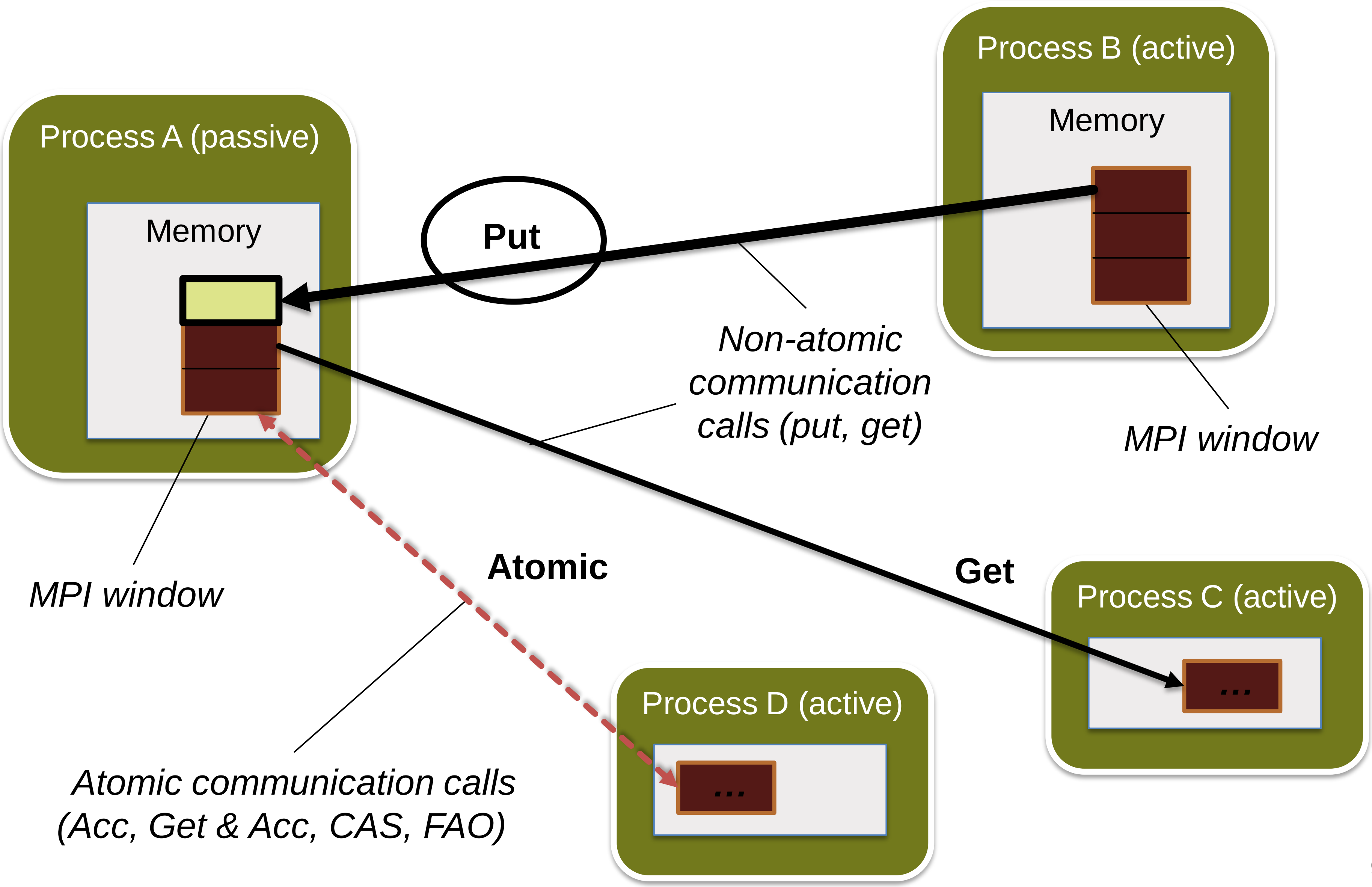
MPI-3 RMA Communication Overview



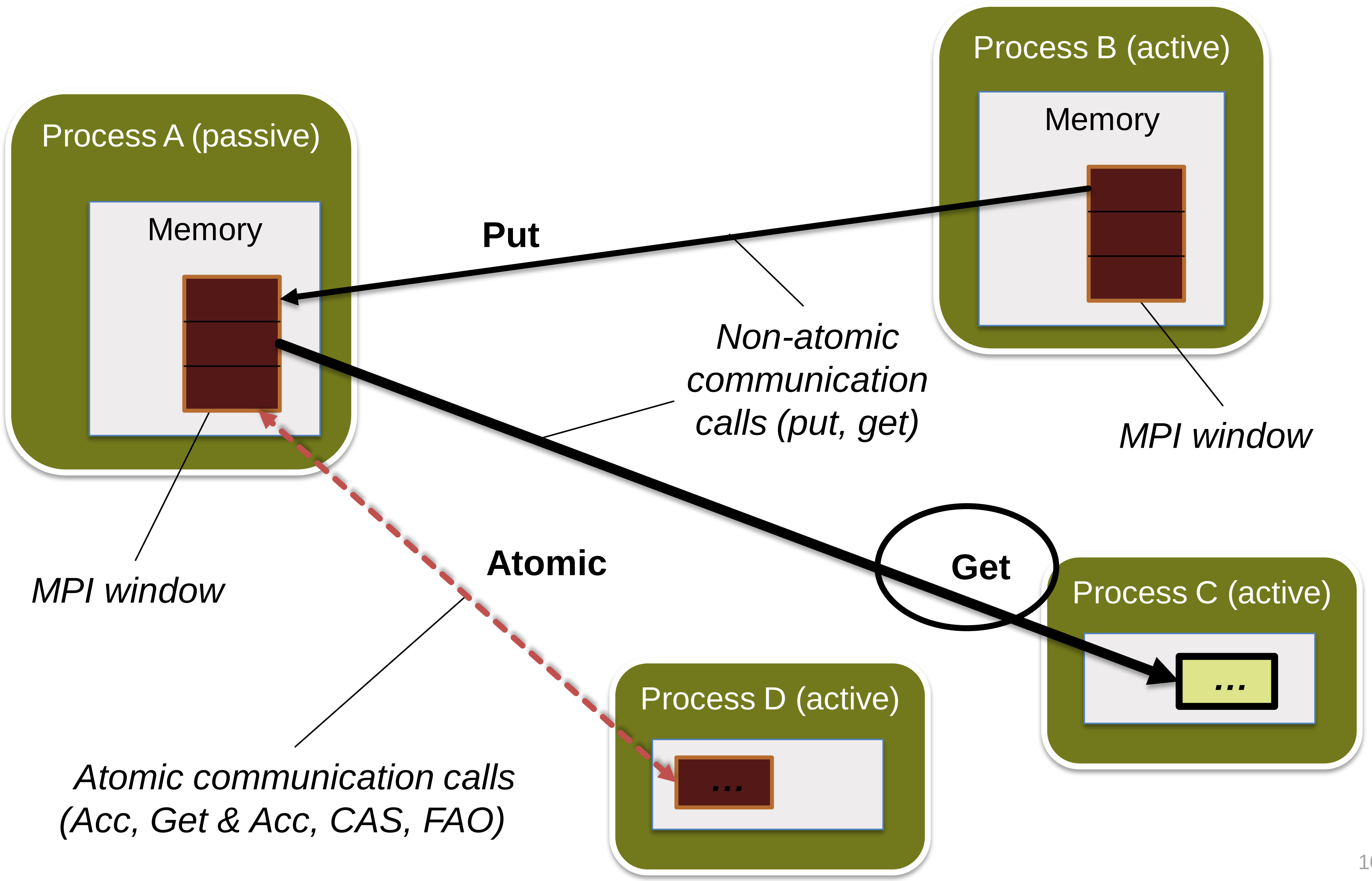
MPI-3 RMA Communication Overview



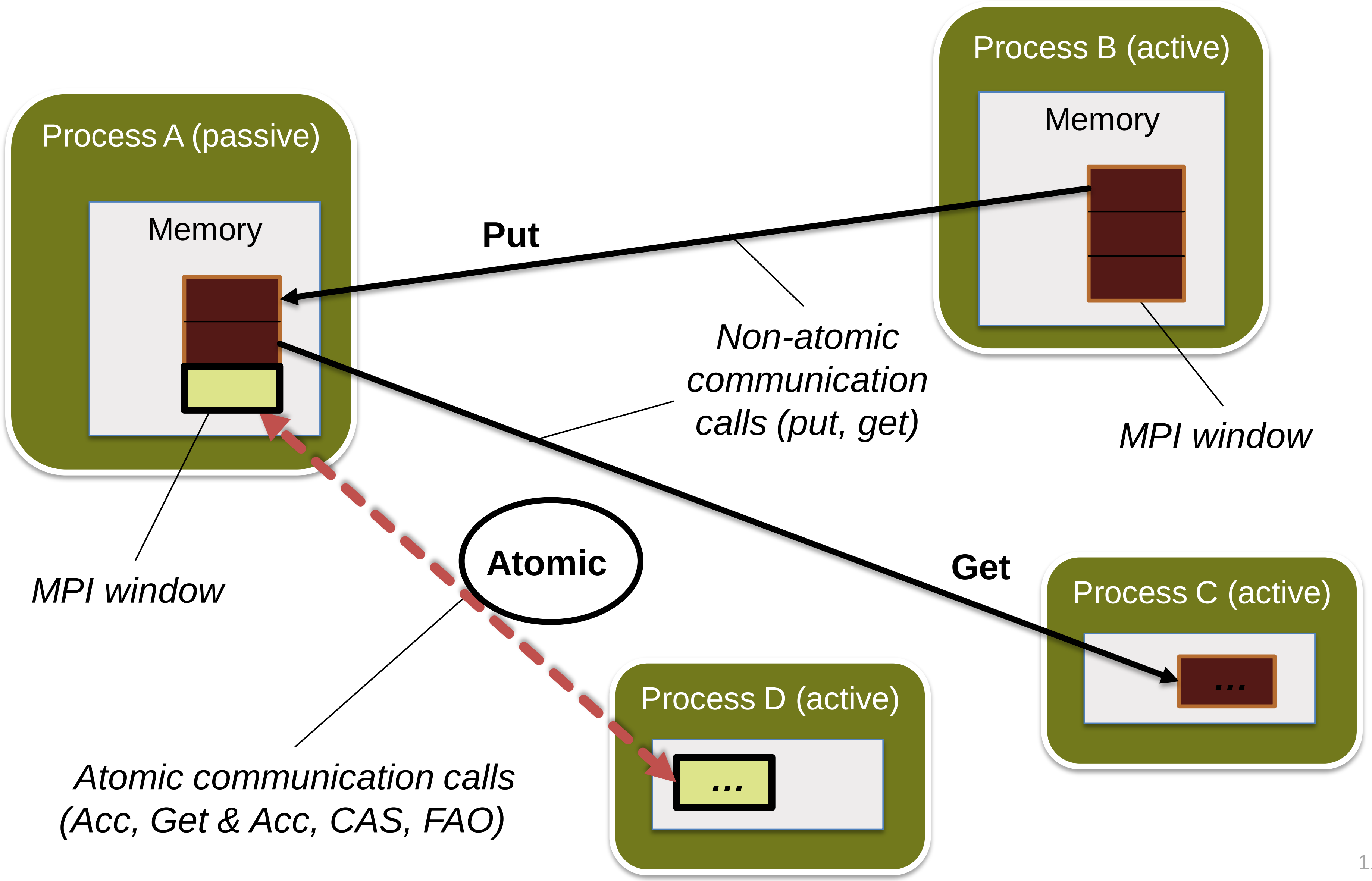
MPI-3 RMA Communication Overview



MPI-3 RMA Communication Overview

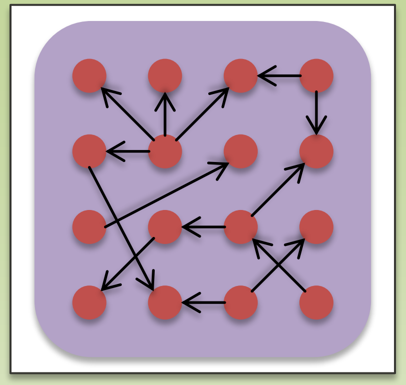


MPI-3 RMA Communication Overview

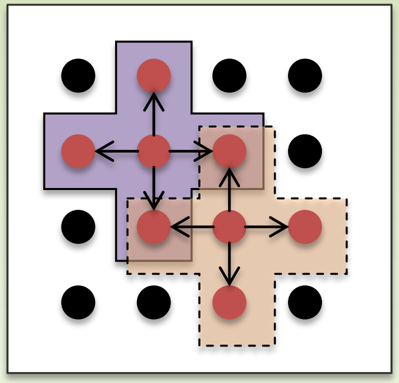


MPI-3.0 RMA Synchronization Overview

Active Target Mode



Fence



Post/Start/
Complete/Wait

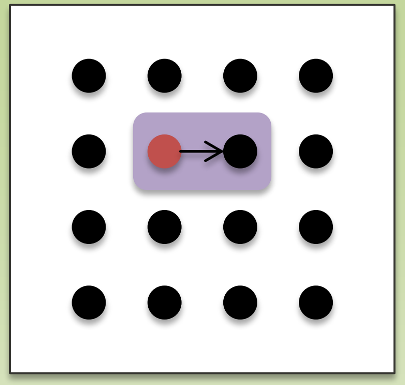
● Active process

● Passive process

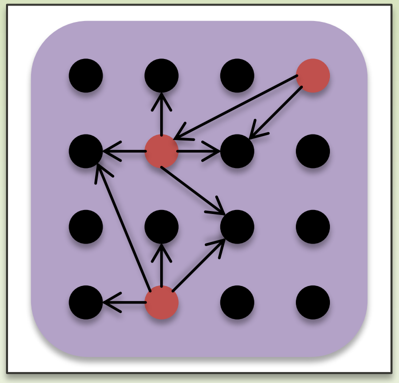
▨ Synchronization

← Communication

Passive Target Mode



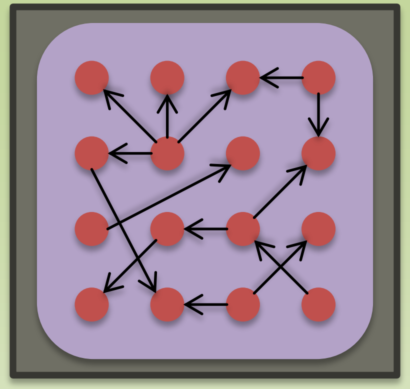
Lock

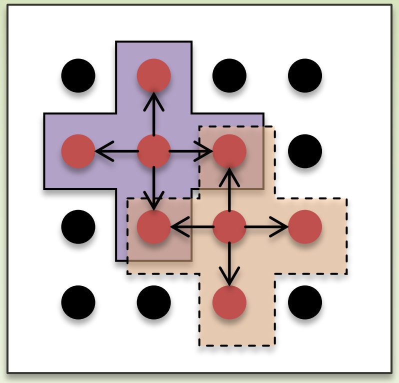


Lock All

MPI-3.0 RMA Synchronization Overview

Active Target Mode


Fence


Post/Start/
Complete/Wait

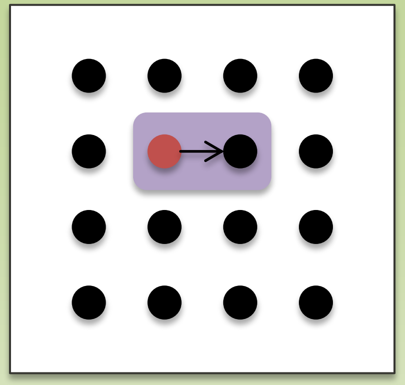
● Active process

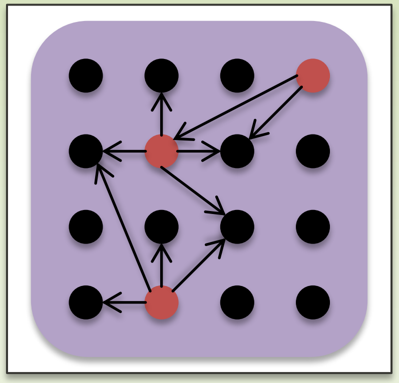
● Passive process

▨ Synchroni-
zation

← Communi-
cation

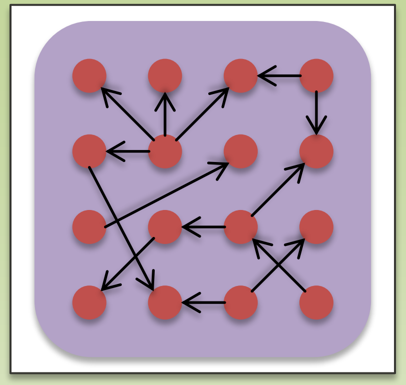
Passive Target Mode

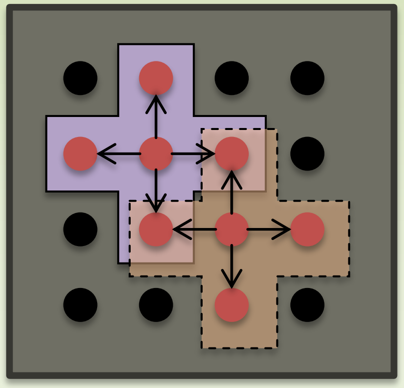

Lock


Lock All

MPI-3.0 RMA Synchronization Overview

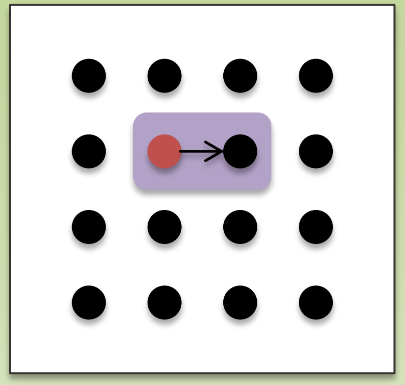
Active Target Mode

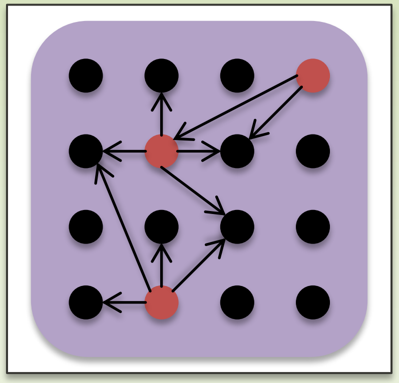

Fence


Post/Start/
Complete/Wait

●	Active process
●	Passive process
▨	Synchroni-zation
←	Communi-cation

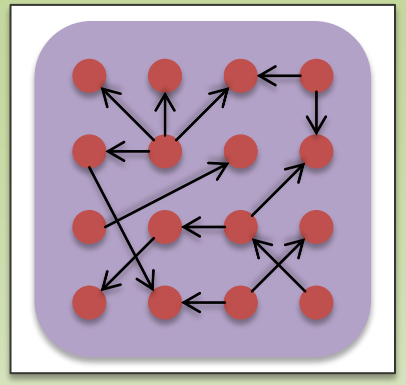
Passive Target Mode


Lock

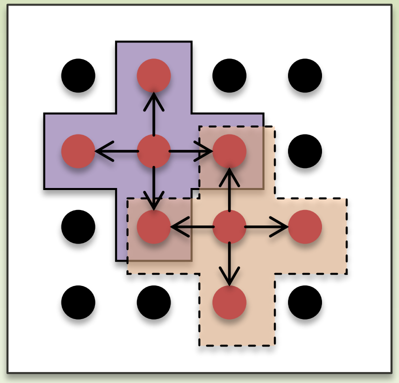

Lock All

MPI-3.0 RMA Synchronization Overview

Active Target Mode



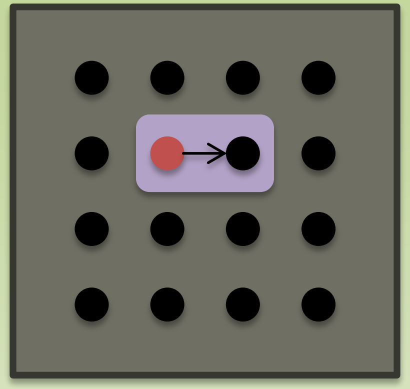
Fence



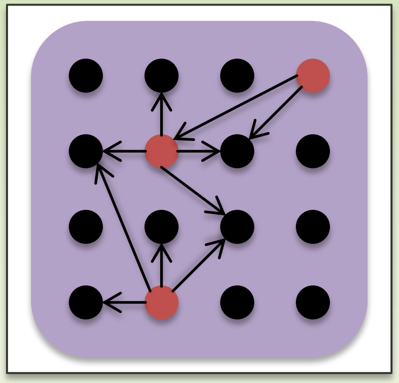
Post/Start/
Complete/Wait

●	Active process
●	Passive process
🟪 🟧	Synchronization
←	Communication

Passive Target Mode



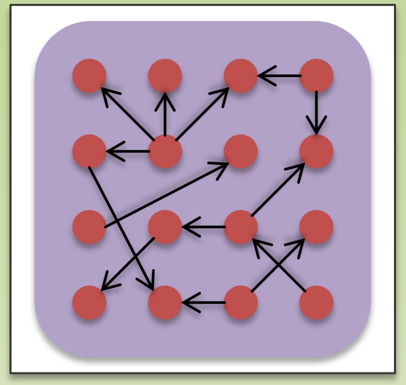
Lock

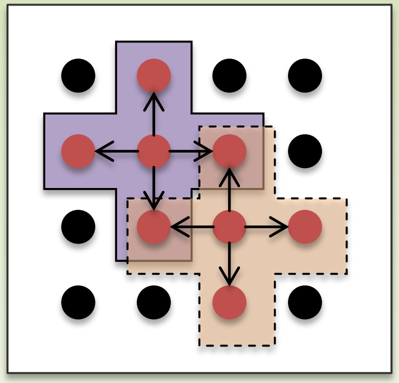


Lock All

MPI-3.0 RMA SYNCHRONIZATION OVERVIEW

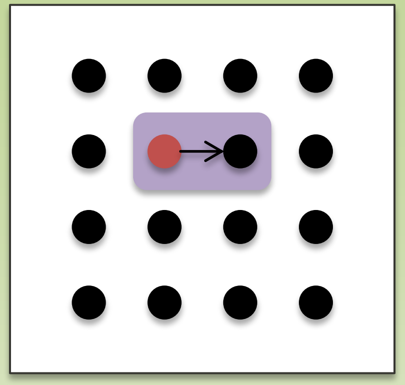
Active Target Mode

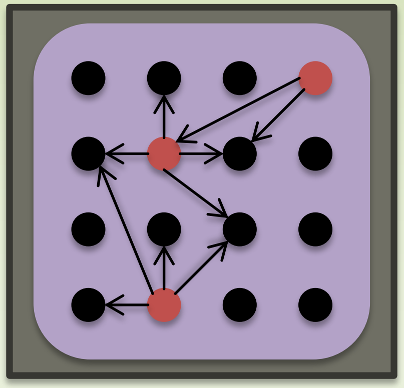

Fence


Post/Start/
Complete/Wait

●	Active process
●	Passive process
▨	Synchroni-zation
←	Communi-cation

Passive Target Mode


Lock


Lock All

Scalable Protocols & Reference Implementation

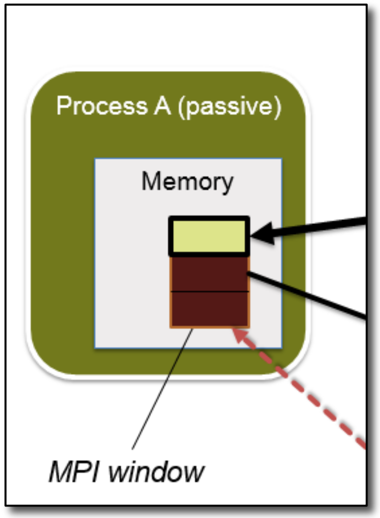
- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)

Scalable Protocols & Reference Implementation

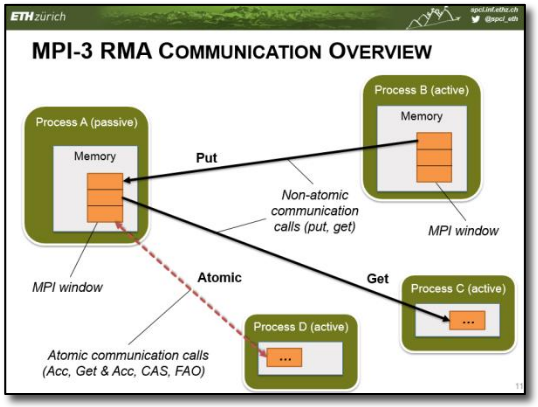
- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)

SCALABLE PROTOCOLS & REFERENCE IMPLEMENTATION

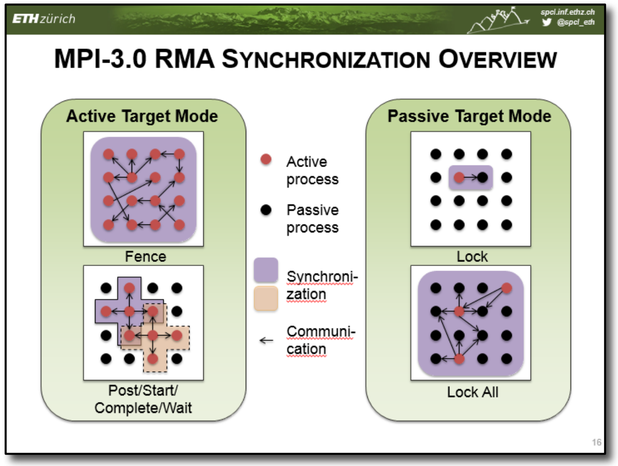
- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)
  - Window creation, communication and synchronization



Window creation


MPI-3 RMA COMMUNICATION OVERVIEW

Communication

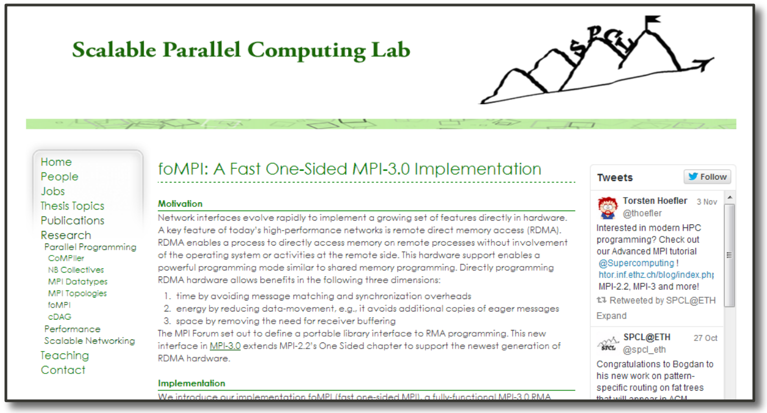

MPI-3.0 RMA SYNCHRONIZATION OVERVIEW

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 Parallel Computing Lab

Home
People
Jobs
Thesis Topics
Publications
Research
Parallel Programming
CoMPIler
NB Collectives
MPI Datatypes
MPI Topologies
foMPI
cDAG
Performance
Scalable Networking
Teaching
Contact

foMPI: A Fast One-Sided MPI-3.0 Implementation

Motivation

Network interfaces evolve rapidly to implement a growing set of features directly in hardware. A key feature of today's high-performance networks is remote direct memory access (RDMA). RDMA enables a process to directly access memory on remote processes without involvement of the operating system or activities at the remote side. This hardware support enables a powerful programming mode similar to shared memory programming. Directly programming RDMA hardware allows benefits in the following three dimensions:

1. time by avoiding message matching and synchronization overheads
2. energy by reducing data-movement, e.g., it avoids additional copies of eager messages
3. space by removing the need for receiver buffering

The MPI Forum set out to define a portable library interface to RMA programming. This new interface in MPI-3.0 extends MPI-2.2's One Sided chapter to support the newest generation of RDMA hardware.

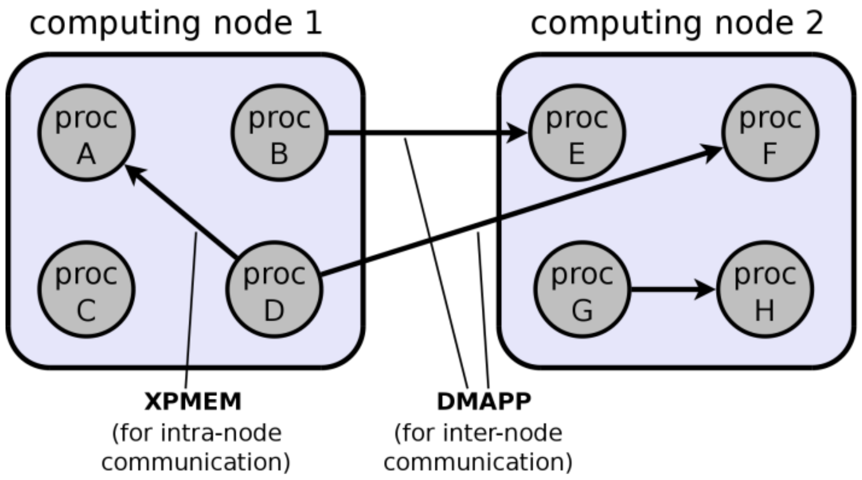
Implementation

We introduce our implementation foMPI (fast one-sided MPI), a fully-functional MPI-3.0 RMA

Tweets Follow

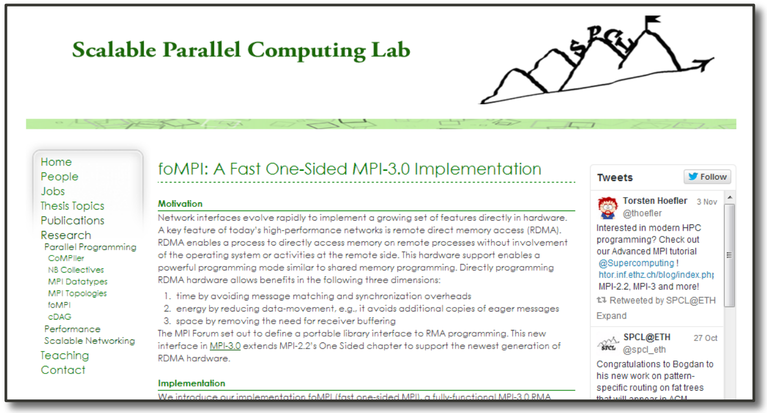
Torsten Hoefler 3 Nov
@thoefler
Interested in modern HPC programming? Check out our Advanced MPI tutorial @Supercomputing ! htor.inf.ethz.ch/blog/index.ph MPI-2.2, MPI-3 and more!
Retweeted by SPCL@ETH
Expand

SPCL@ETH 27 Oct
@spcl_eth
Congratulations to Bogdan to his new work on pattern-specific routing on fat trees that will appear in ACM



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 Parallel Computing Lab

Home
People
Jobs
Thesis Topics
Publications
Research
Parallel Programming
CoMPIler
NB Collectives
MPI Datatypes
MPI Topologies
foMPI
cDAG
Performance
Scalable Networking
Teaching
Contact

foMPI: A Fast One-Sided MPI-3.0 Implementation

Motivation

Network interfaces evolve rapidly to implement a growing set of features directly in hardware. A key feature of today's high-performance networks is remote direct memory access (RDMA). RDMA enables a process to directly access memory on remote processes without involvement of the operating system or activities at the remote side. This hardware support enables a powerful programming mode similar to shared memory programming. Directly programming RDMA hardware allows benefits in the following three dimensions:

1. time by avoiding message matching and synchronization overheads
2. energy by reducing data-movement, e.g., it avoids additional copies of eager messages
3. space by removing the need for receiver buffering

The MPI Forum set out to define a portable library interface to RMA programming. This new interface in MPI-3.0 extends MPI-2.2's One Sided chapter to support the newest generation of RDMA hardware.

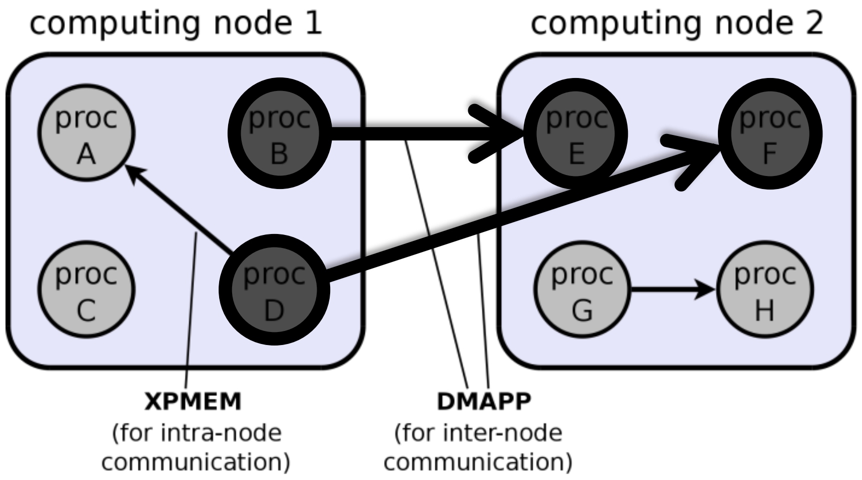
Implementation

We introduce our implementation foMPI (fast one-sided MPI), a fully-functional MPI-3.0 RMA

Tweets | Follow

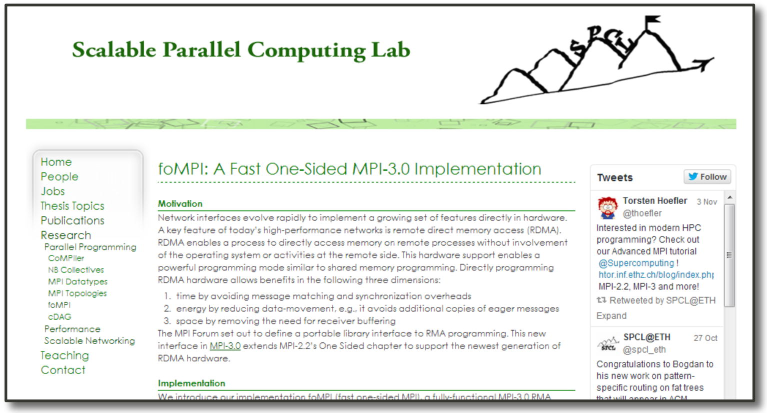
Torsten Hoefler @thoefler 3 Nov
Interested in modern HPC programming? Check out our Advanced MPI tutorial @Supercomputing ! htor.inf.ethz.ch/blog/index.php MPI-2.2, MPI-3 and more!
Retweeted by SPCL@ETH
Expand

SPCL@ETH @spcl_eth 27 Oct
Congratulations to Bogdan to his new work on pattern-specific routing on fat trees that will appear in ACM



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 Parallel Computing Lab

Home
People
Jobs
Thesis Topics
Publications
Research
Parallel Programming
CoMPIler
NB Collectives
MPI Datatypes
MPI Topologies
foMPI
cDAG
Performance
Scalable Networking
Teaching
Contact

foMPI: A Fast One-Sided MPI-3.0 Implementation

Motivation

Network interfaces evolve rapidly to implement a growing set of features directly in hardware. A key feature of today's high-performance networks is remote direct memory access (RDMA). RDMA enables a process to directly access memory on remote processes without involvement of the operating system or activities at the remote side. This hardware support enables a powerful programming mode similar to shared memory programming. Directly programming RDMA hardware allows benefits in the following three dimensions:

1. time by avoiding message matching and synchronization overheads
2. energy by reducing data-movement, e.g., it avoids additional copies of eager messages
3. space by removing the need for receiver buffering

The MPI Forum set out to define a portable library interface to RMA programming. This new interface in MPI-3.0 extends MPI-2.2's One Sided chapter to support the newest generation of RDMA hardware.

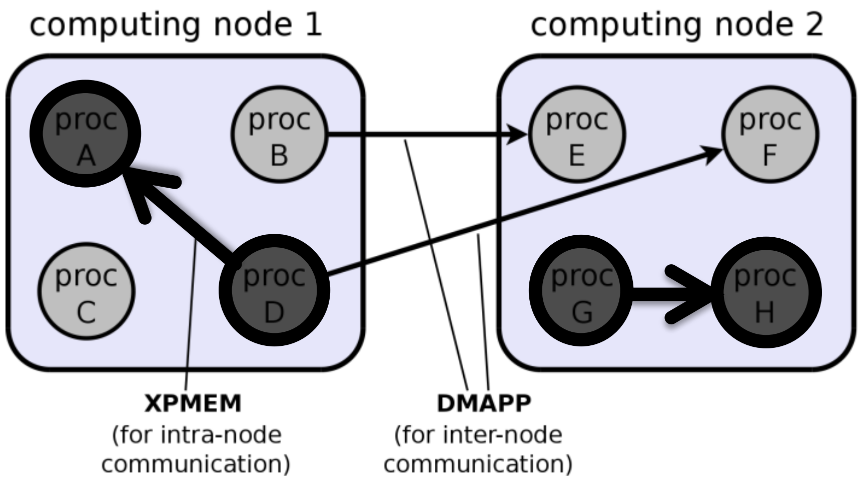
Implementation

We introduce our implementation foMPI (fast one-sided MPI), a fully-functional MPI-3.0 RMA

Tweets Follow

Torsten Hoefler 3 Nov
@thoefler
Interested in modern HPC programming? Check out our Advanced MPI tutorial @Supercomputing ! htor.inf.ethz.ch/blog/index.php MPI-2.2, MPI-3 and more!
Retweeted by SPCL@ETH
Expand

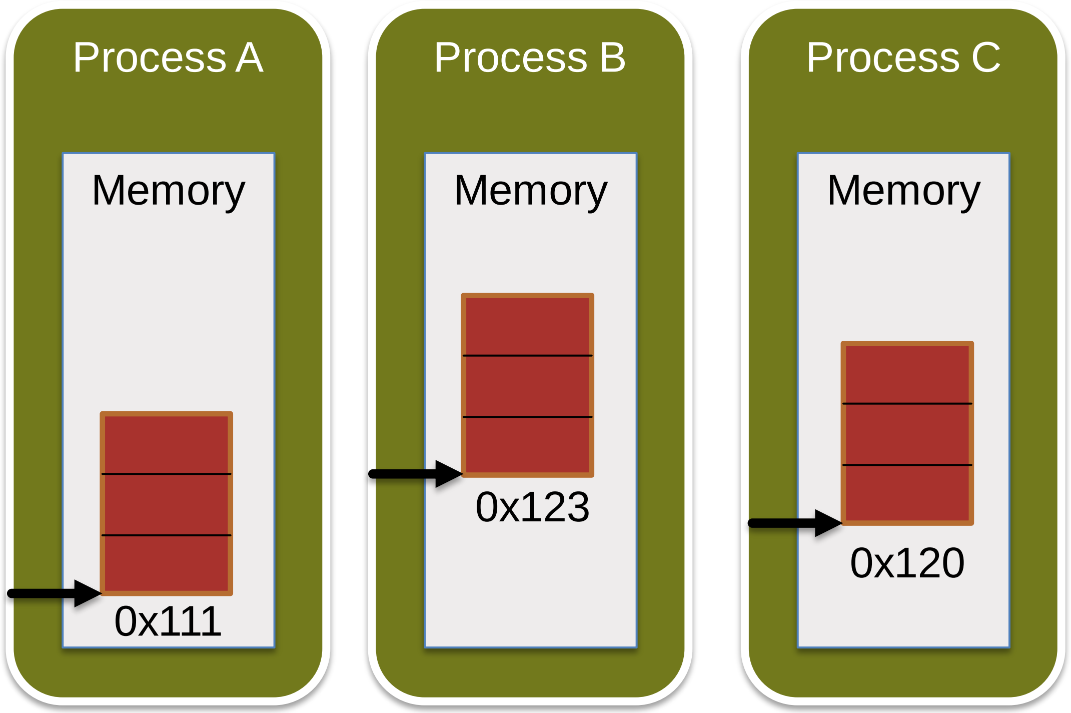
SPCL@ETH 27 Oct
@spcl_eth
Congratulations to Bogdan to his new work on pattern-specific routing on fat trees that will appear in ACM



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

PART 1: SCALABLE WINDOW CREATION

Traditional windows



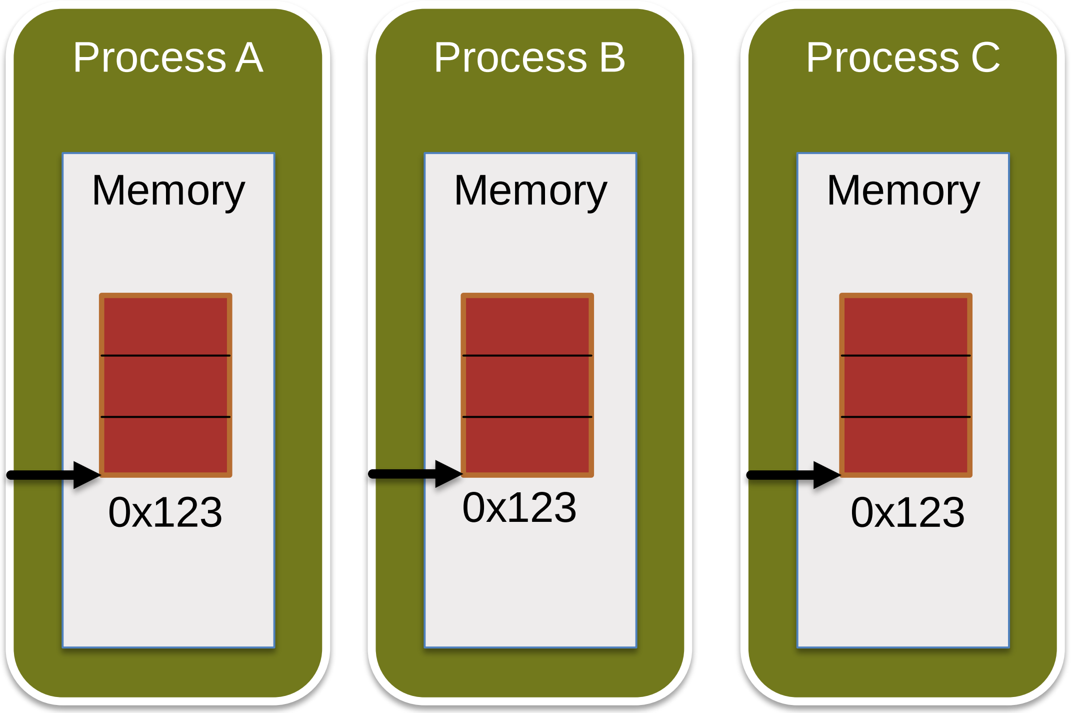
p = total number of processes

backwards compatible (MPI-2)

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

Part 1: Scalable Window Creation

Allocated windows



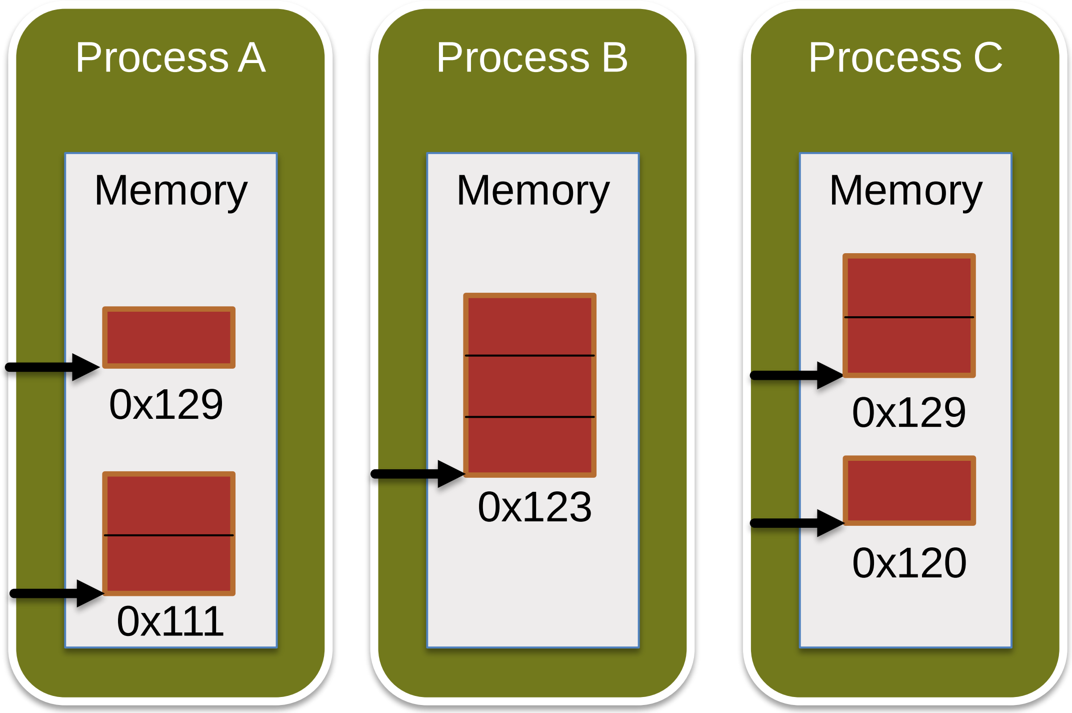
p = total number of processes

Allows MPI to allocate memory

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

Part 1: Scalable Window Creation

Dynamic windows



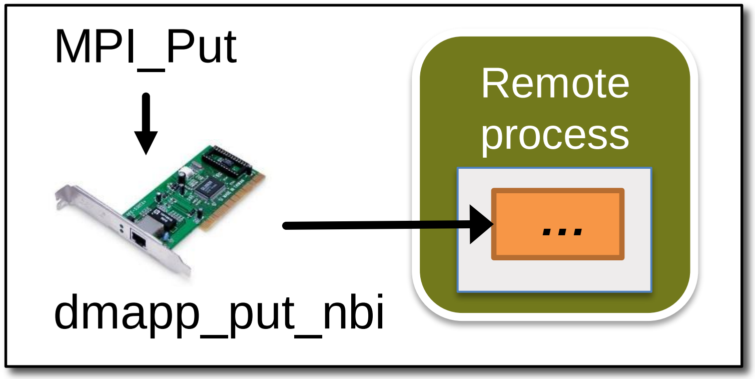
p = total number of processes

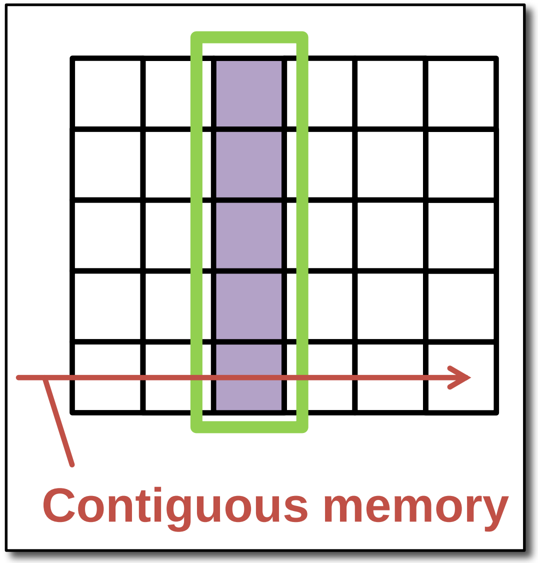
Local attach/detach
Most flexible

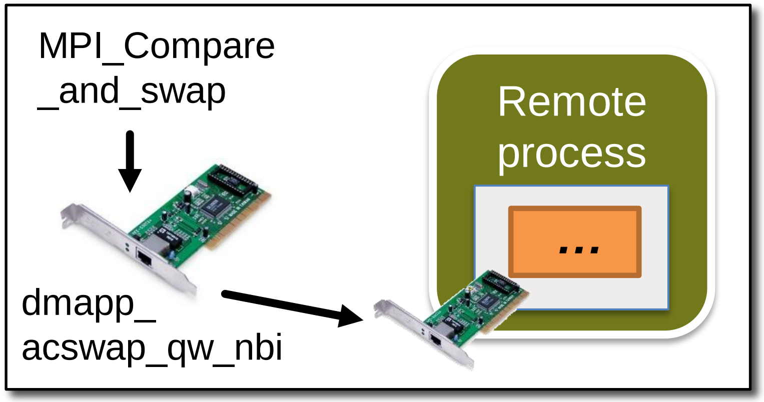
Time bound: $\mathcal{O}(\log p)$
Memory bound: $\mathcal{O}(p)$

Part 2: 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)

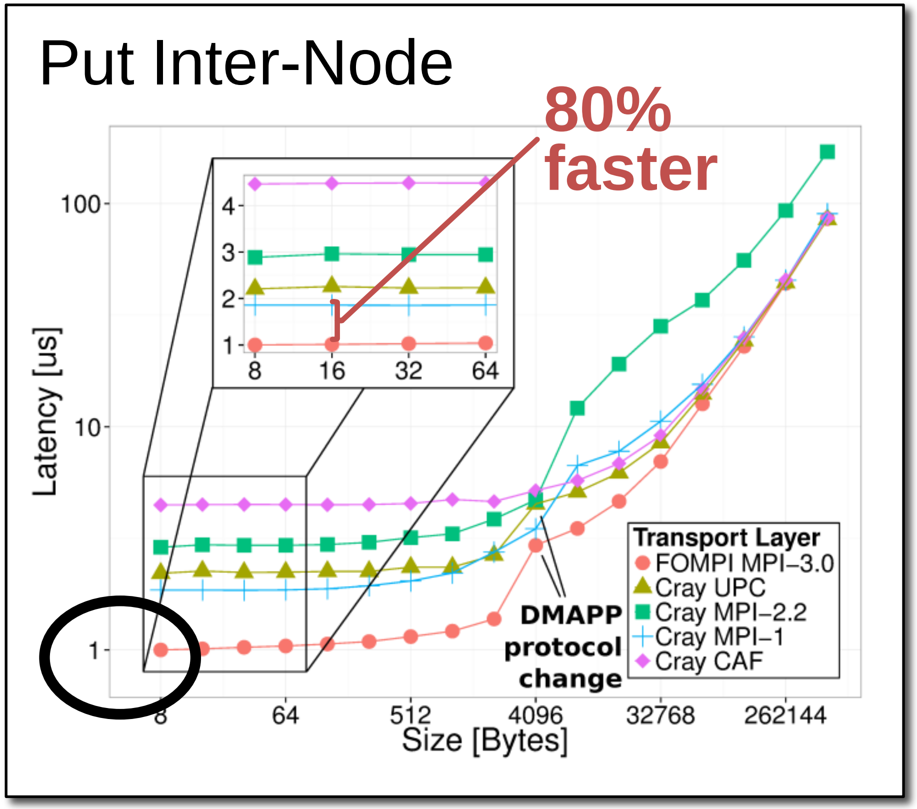


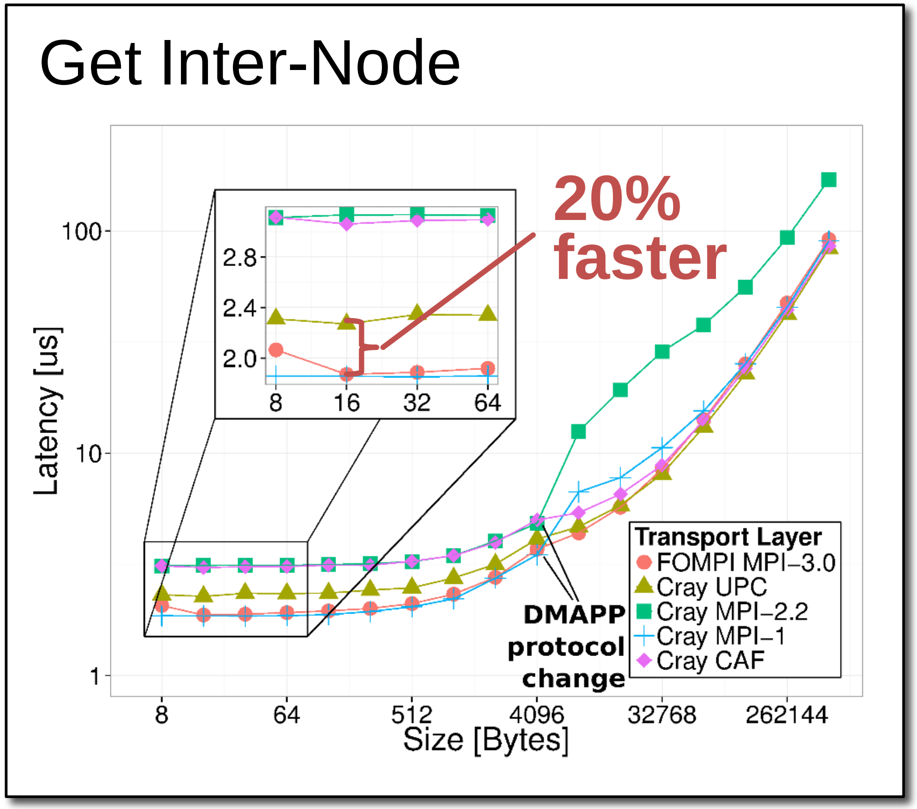


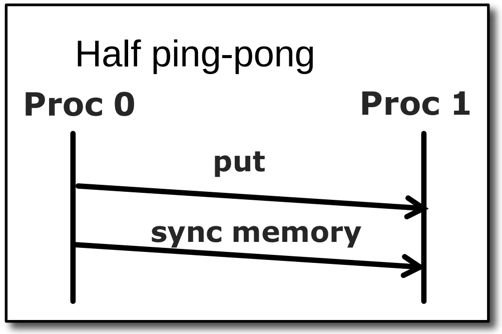


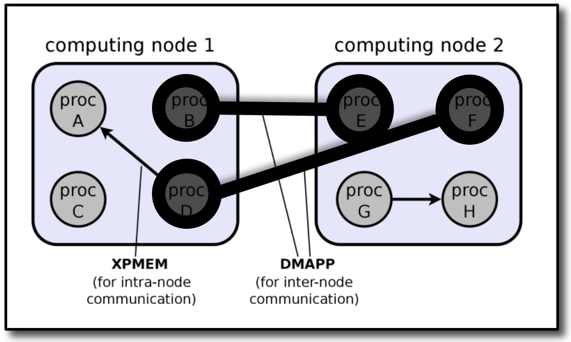
[1] Ross, Latham, Gropp, Lusk, Thakur. Processing MPI datatypes outside MPI. EuroMPI/PVM’09

Performance Inter-node: Latency

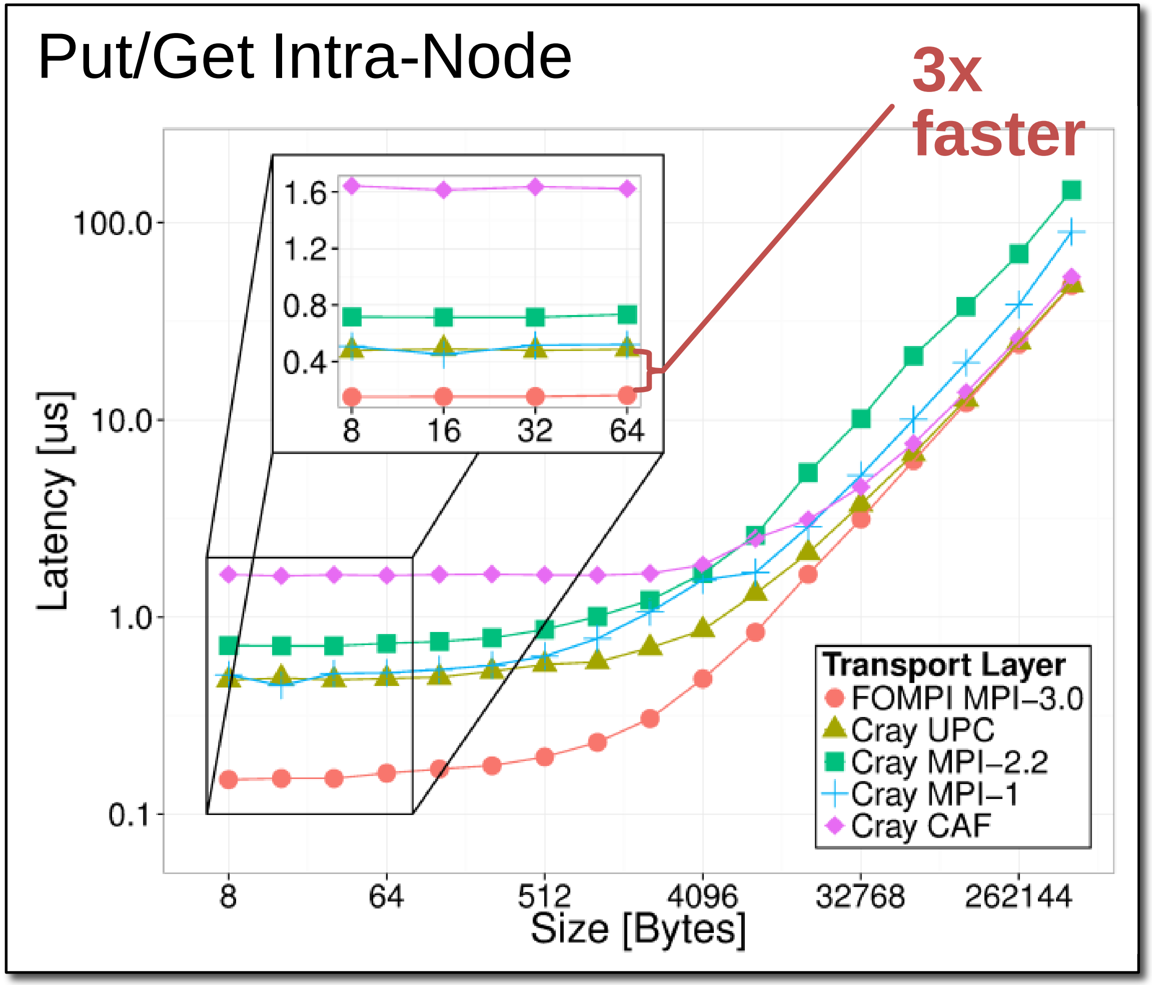


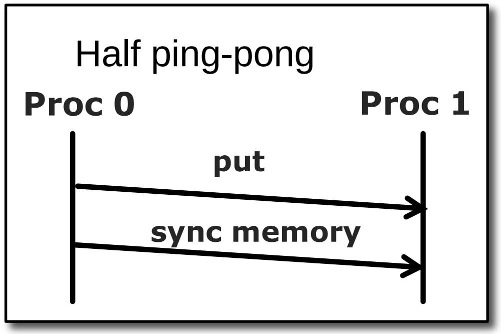


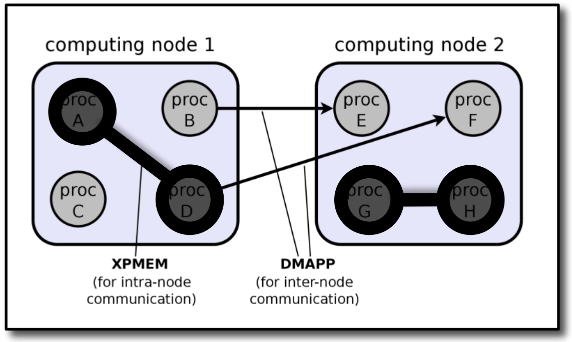




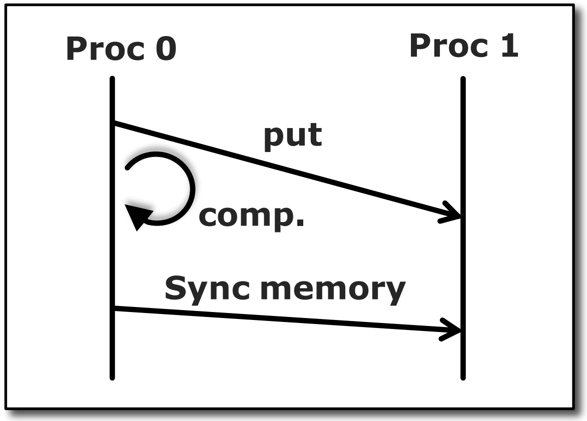
PERFORMANCE INTRA-NODE: LATENCY



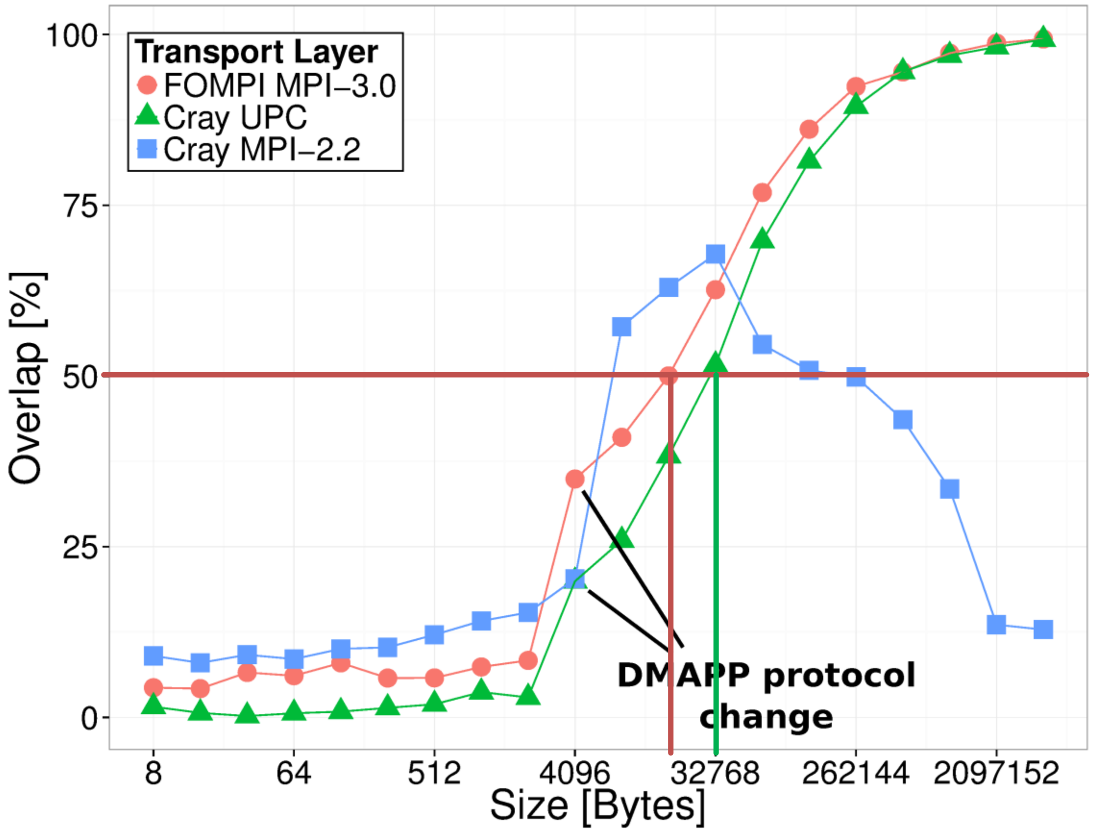




PERFORMANCE: OVERLAP

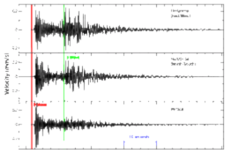


Inter-Node Overlap in %

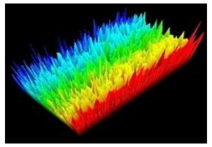


Useful for, e.g., scientific codes:

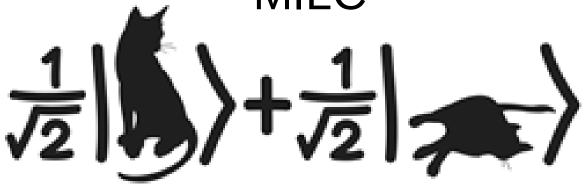
AWM-Olsen seismic



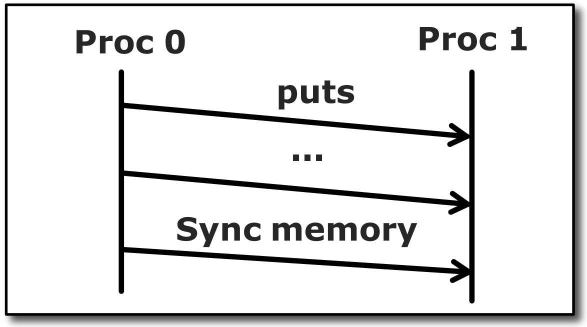
3D FFT



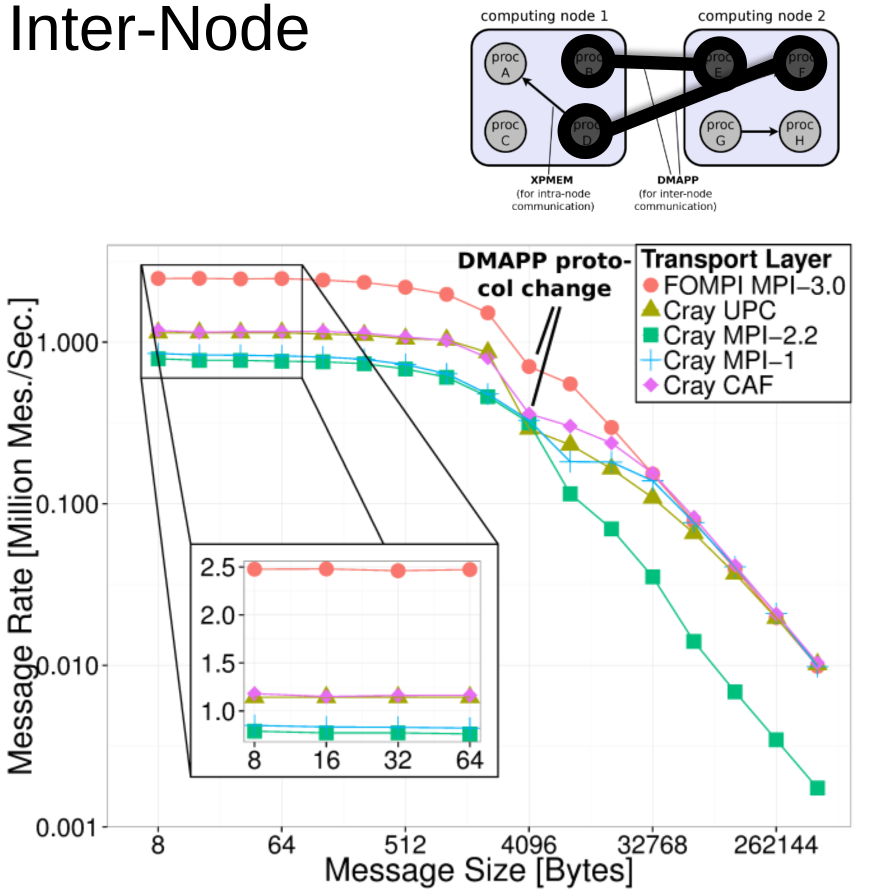
MILC



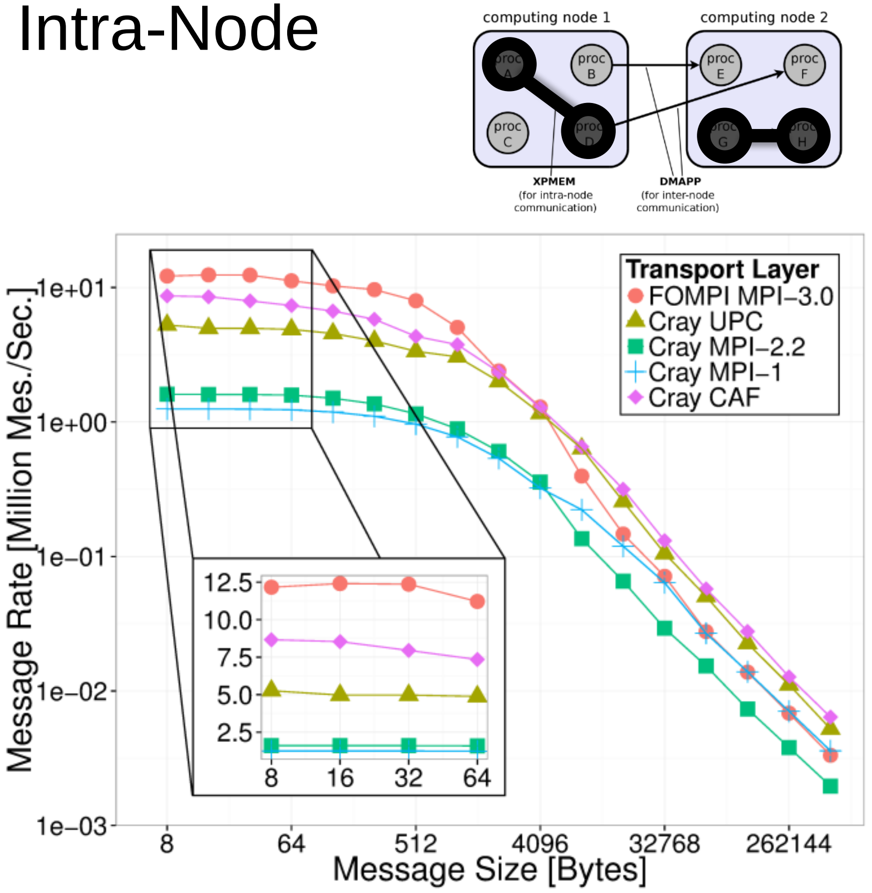
Performance: Message Rate



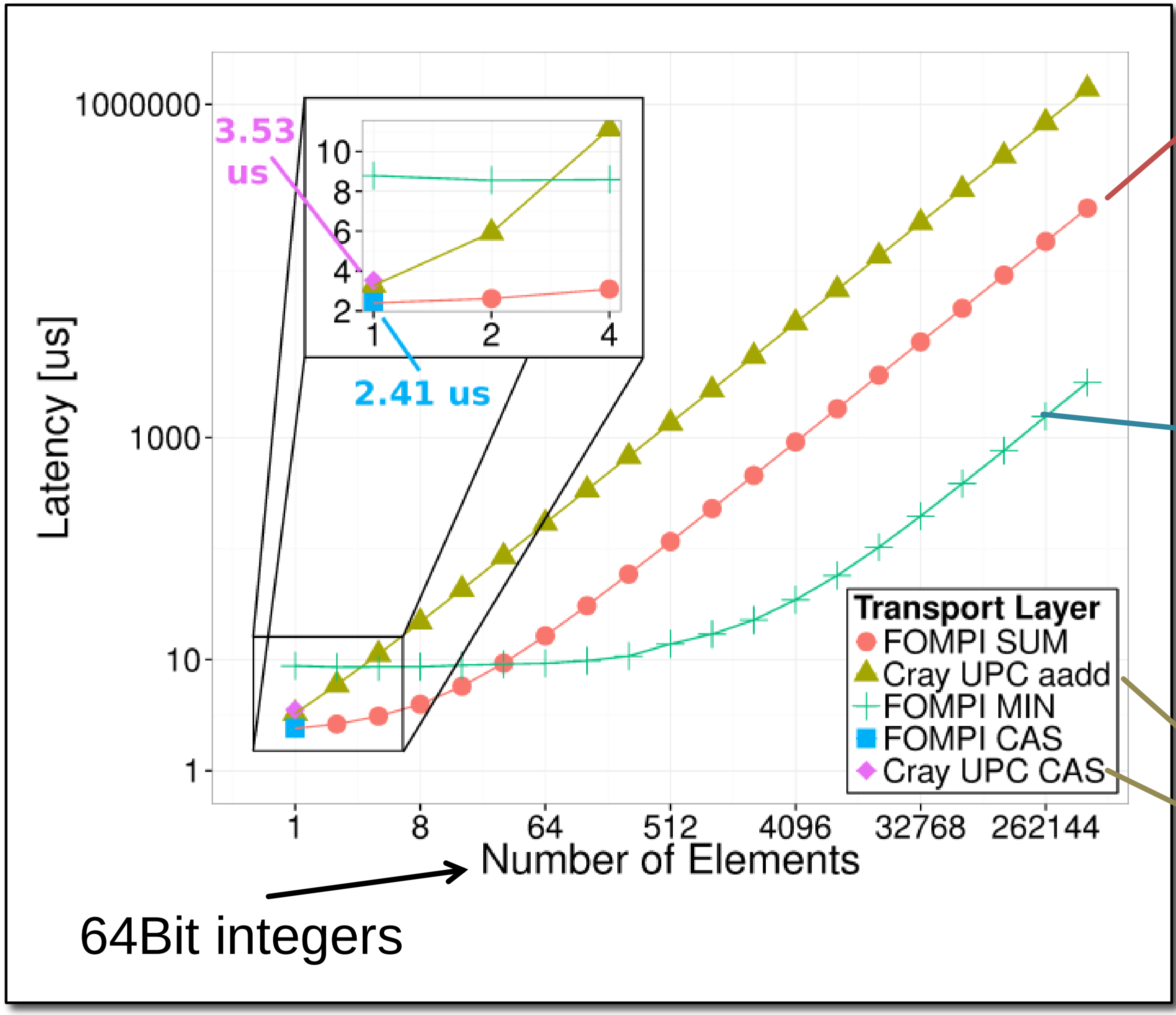
Inter-Node



Intra-Node



PERFORMANCE: ATOMICS



64Bit integers

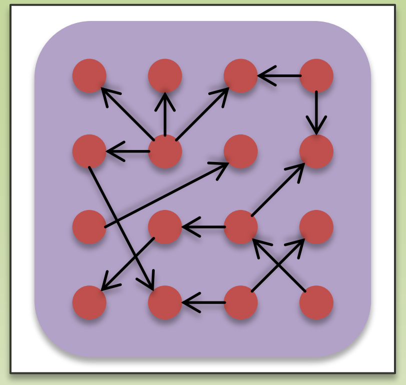
hardware-accelerated protocol:
lower latency

fall-back protocol:
higher bandwidth

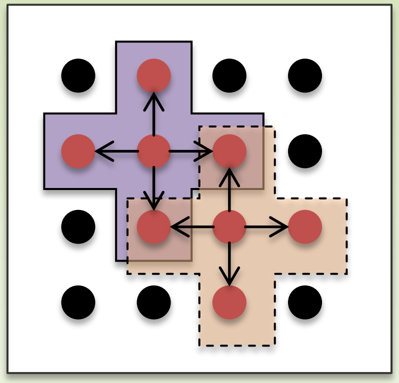
proprietary

PART 3: SYNCHRONIZATION

Active Target Mode



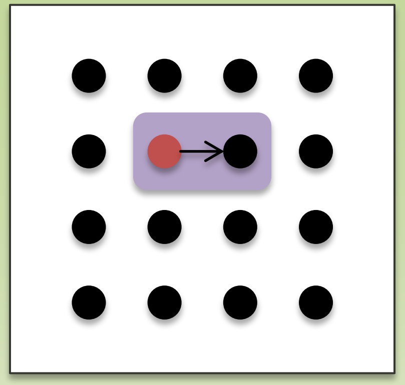
Fence



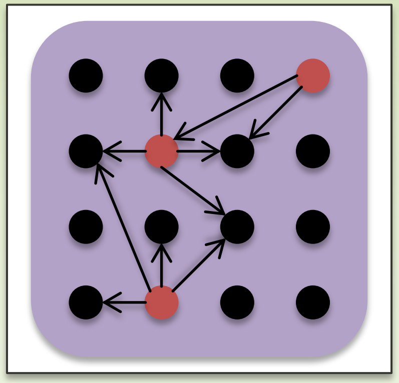
Post/Start/
Complete/Wait

- ● Active process
- ● Passive process
- ▨ Synchroni-zation
- ← Communi-cation

Passive Target Mode



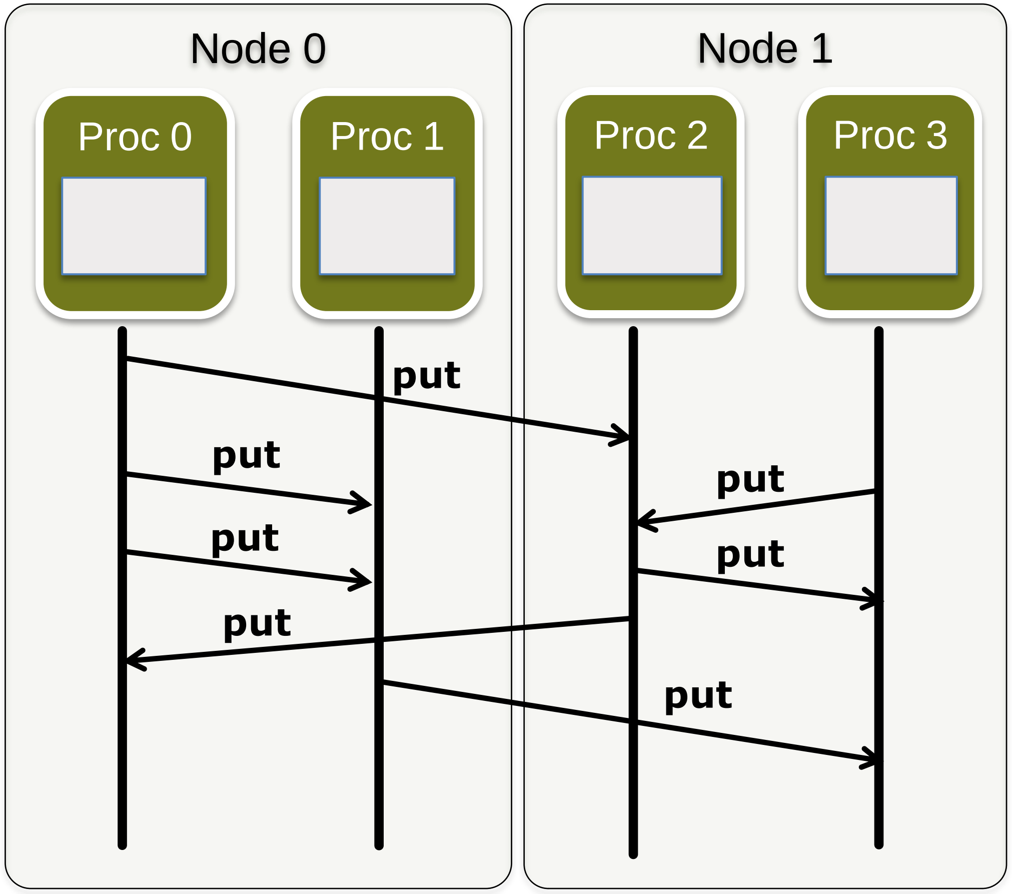
Lock



Lock All

Scalable Fence Implementation

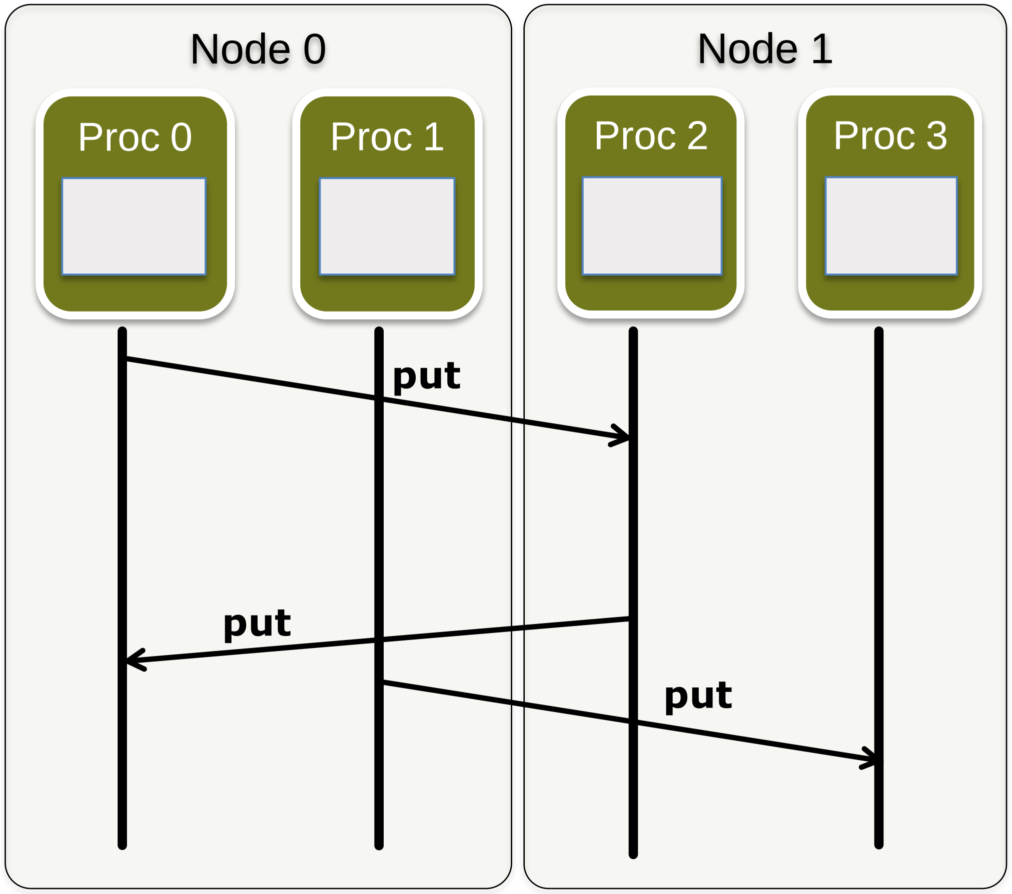
- Collective call
- Completes all outstanding memory operations



```
int MPI_Win_fence(…) {
   asm( mfence );
   dmapp_gsync_wait();
   MPI_Barrier(...);
   return MPI_SUCCESS;
}
```

Scalable Fence Implementation

- Collective call
- Completes all outstanding memory operations

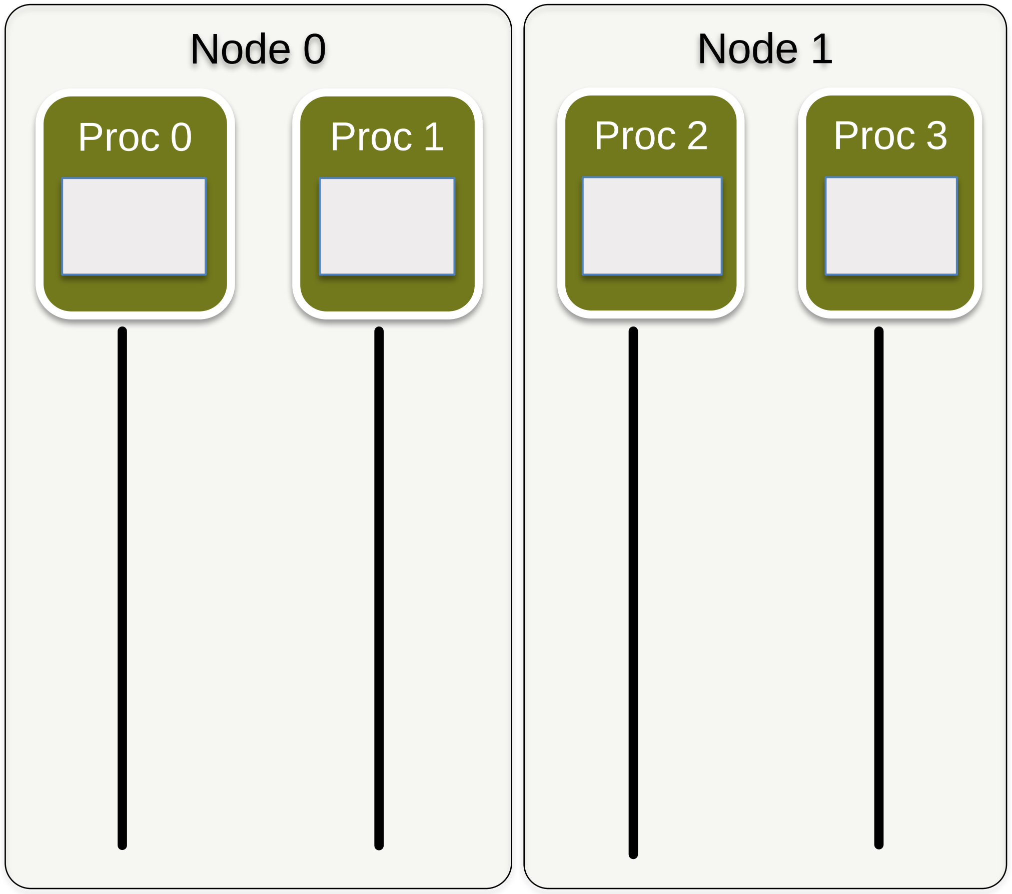


```
int MPI_Win_fence(…) {
   asm( mfence );
   dmapp_gsync_wait();
   MPI_Barrier(...);
   return MPI_SUCCESS;
}
```

Local completion (XPMEM)

Scalable Fence Implementation

- Collective call
- Completes all outstanding memory operations

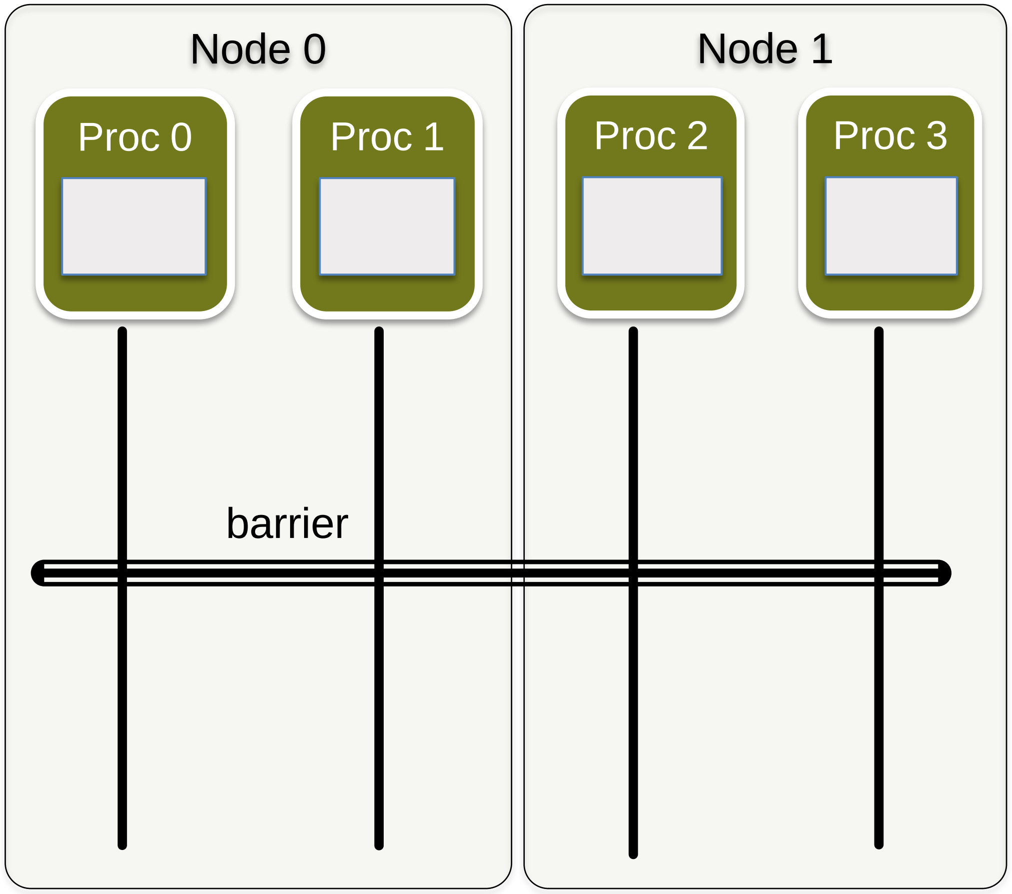


```
int MPI_Win_fence(…) {
   asm( mfence );
   dmapp_gsync_wait();
   MPI_Barrier(...);
   return MPI_SUCCESS;
}
```

Remote completion (DMAPP)

Scalable Fence Implementation

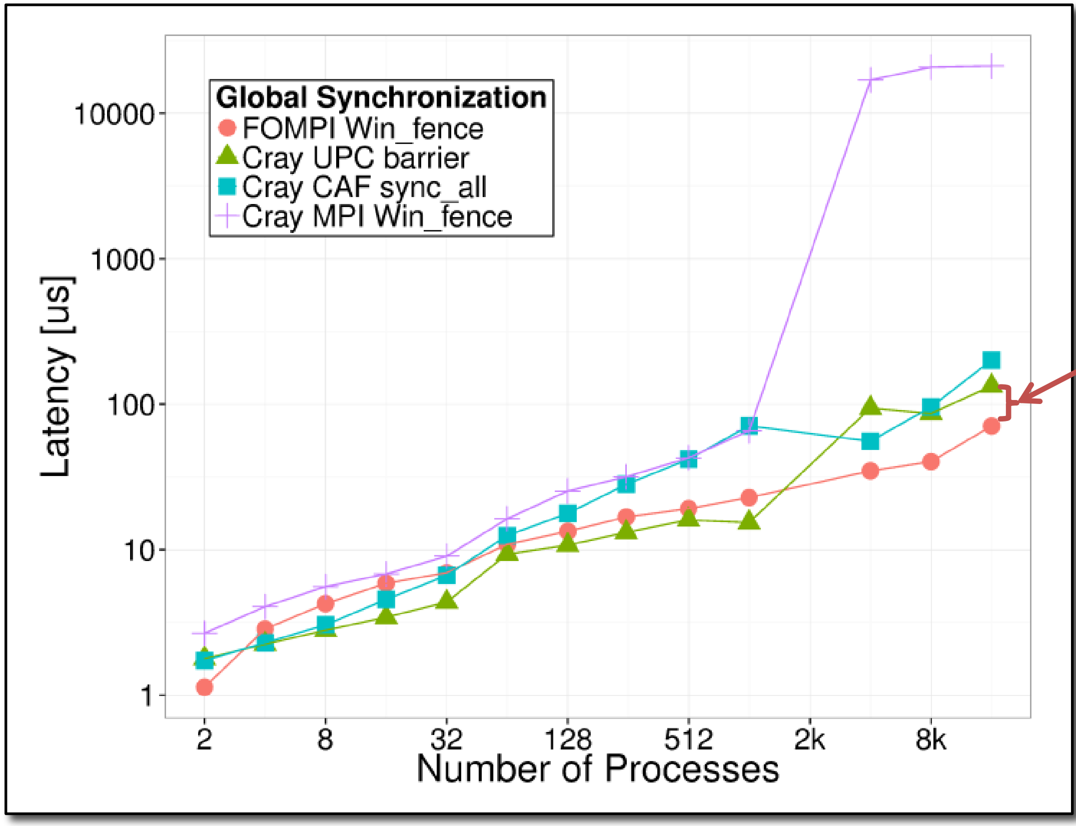
- Collective call
- Completes all outstanding memory operations



```
int MPI_Win_fence(…) {
  asm( mfence );
  dmapp_gsync_wait();
  MPI_Barrier(...);
  return MPI_SUCCESS;
}
```

Global completion

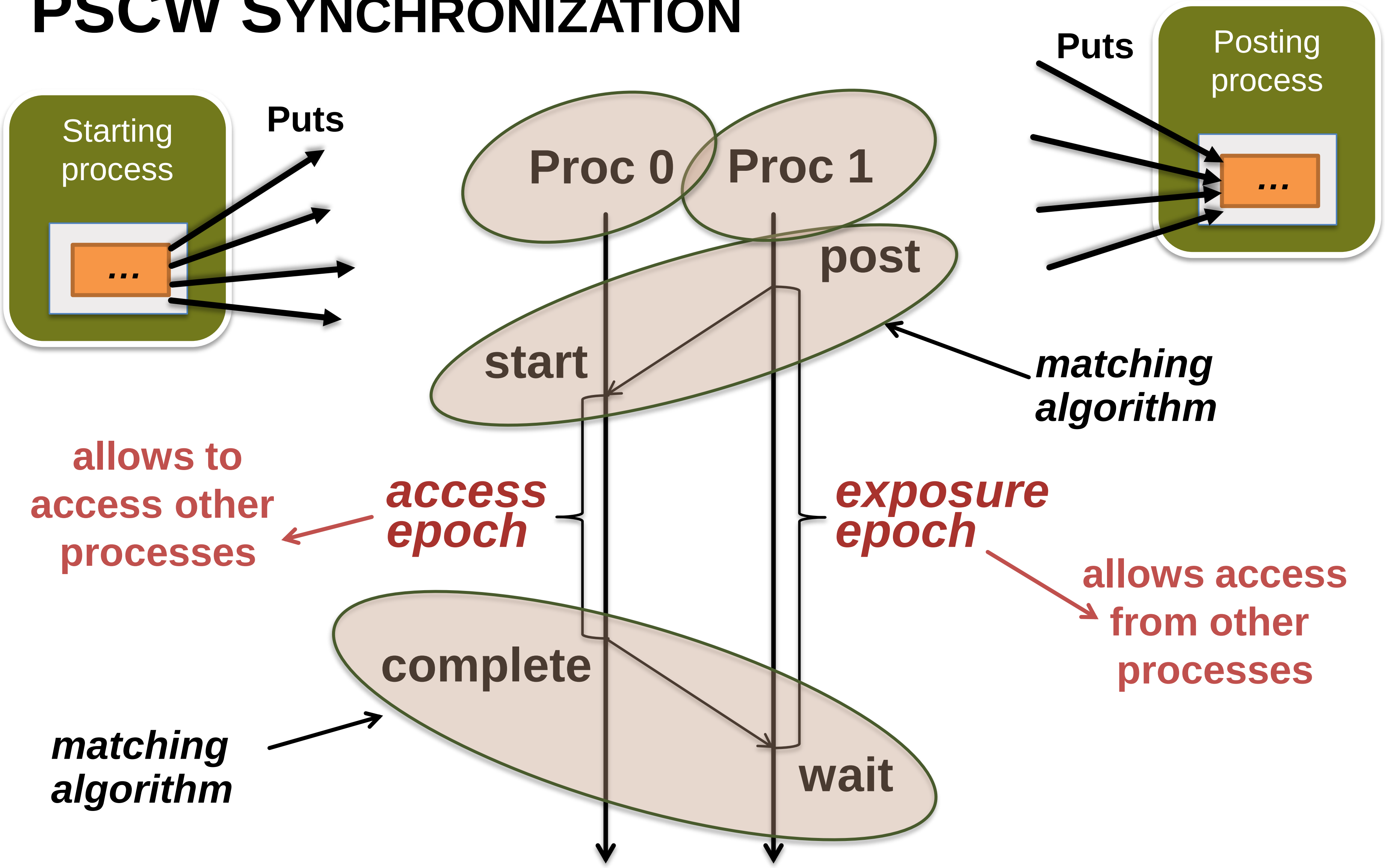
Scalable Fence Performance



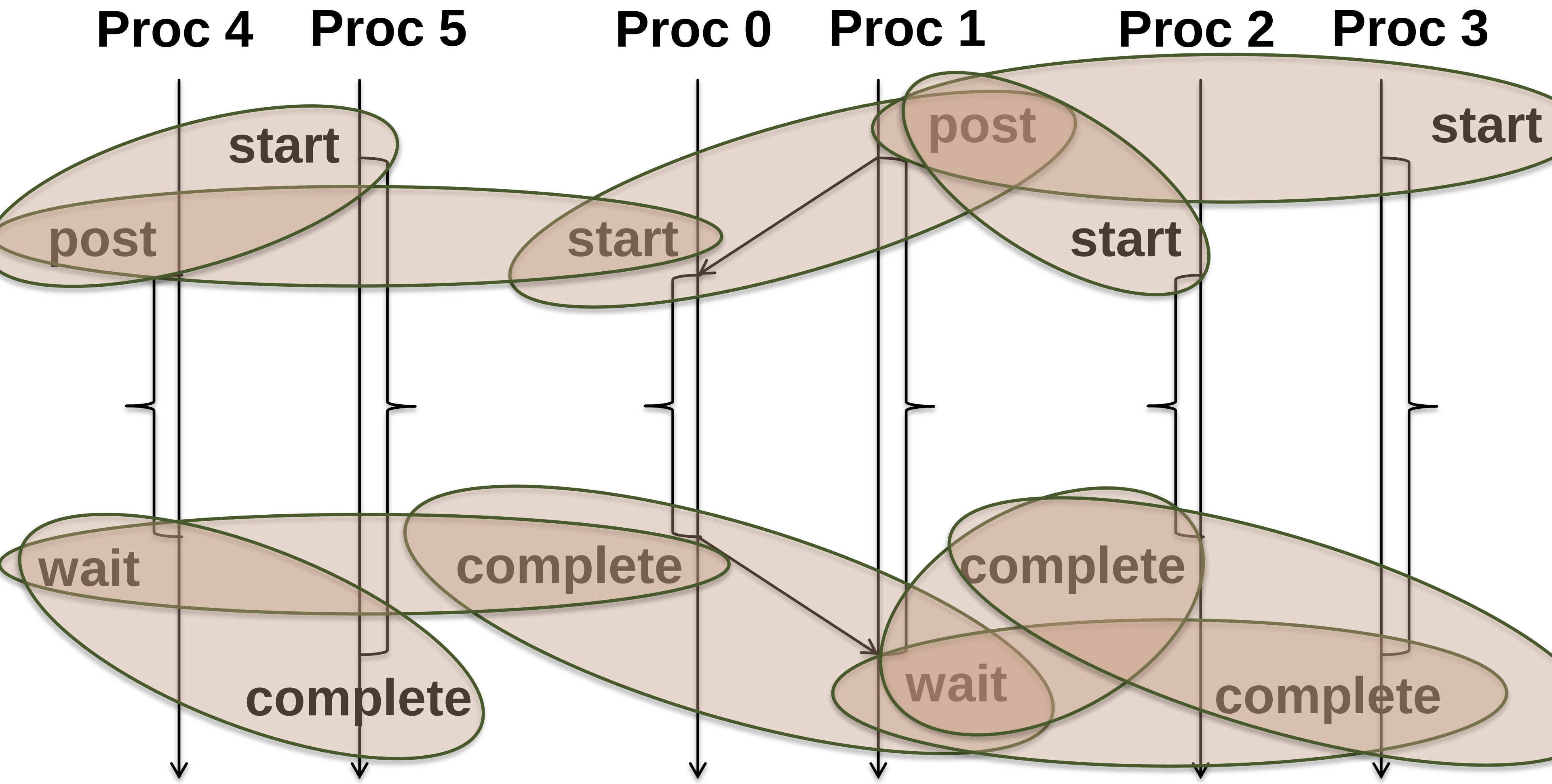
90% faster

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

PSCW SYNCHRONIZATION

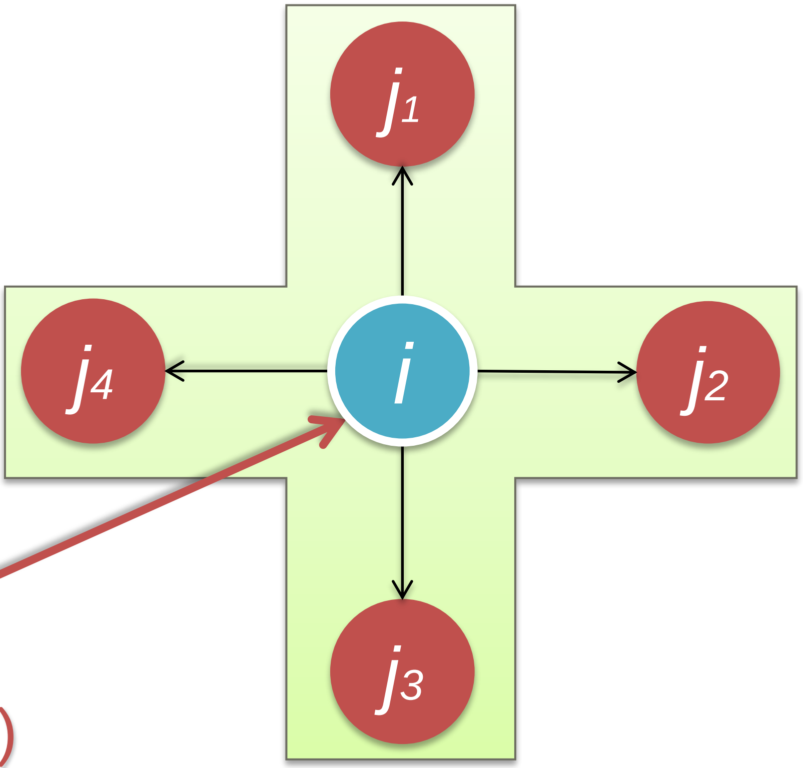


PSCW SYNCHRONIZATION



PSCW Scalable Post/Start Matching

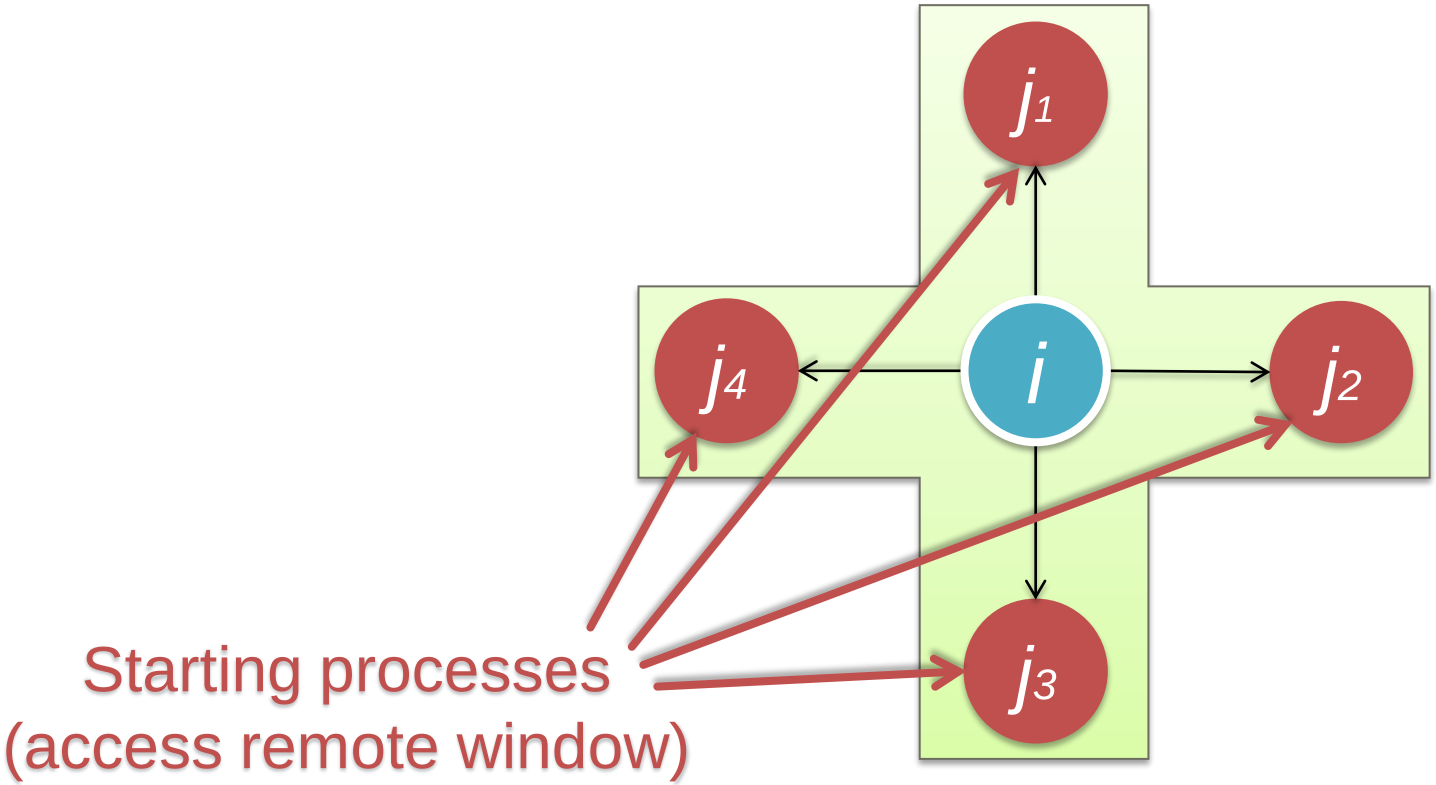
- In general, there can be n *posting* and m *starting* processes
- In this example there is one *posting* and 4 *starting* processes



Posting process
(opens its window)

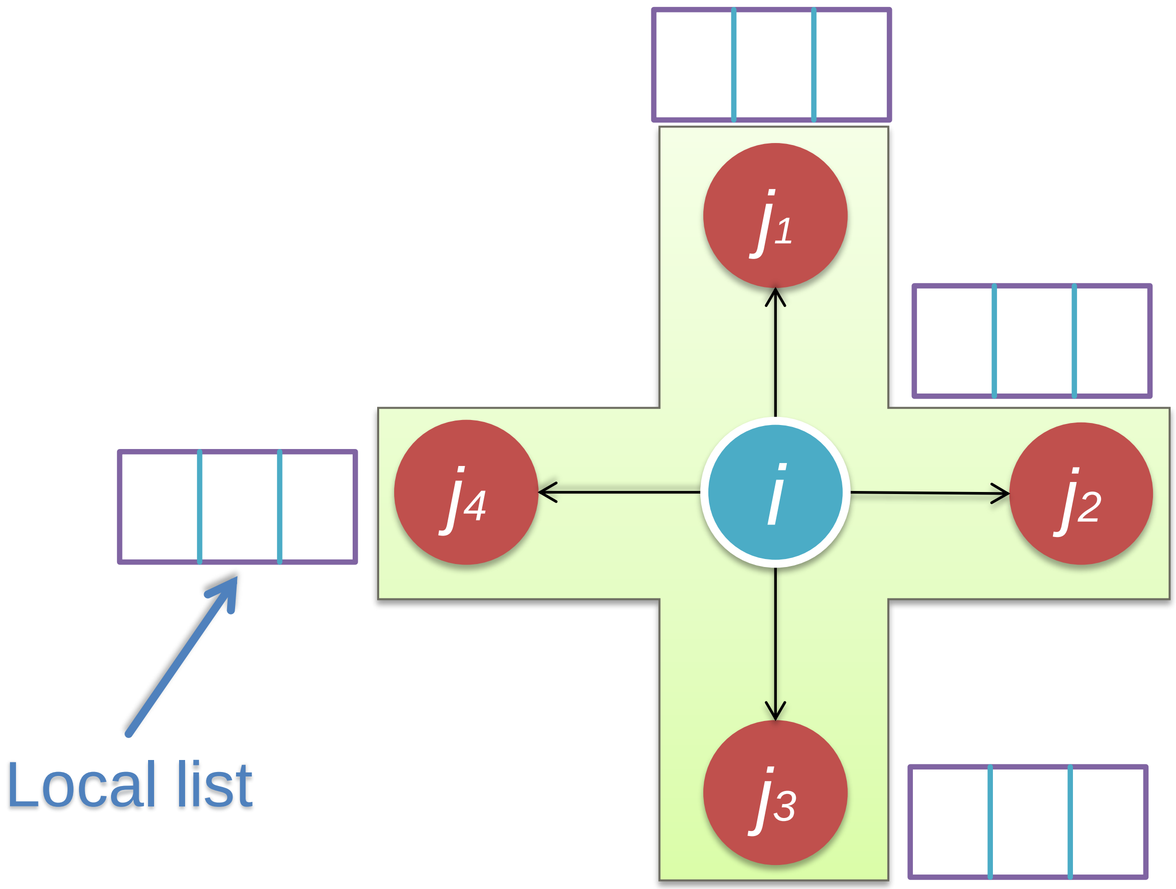
PSCW Scalable Post/Start Matching

- In general, there can be n *posting* and m *starting* processes
- In this example there is one *posting* and 4 *starting* processes



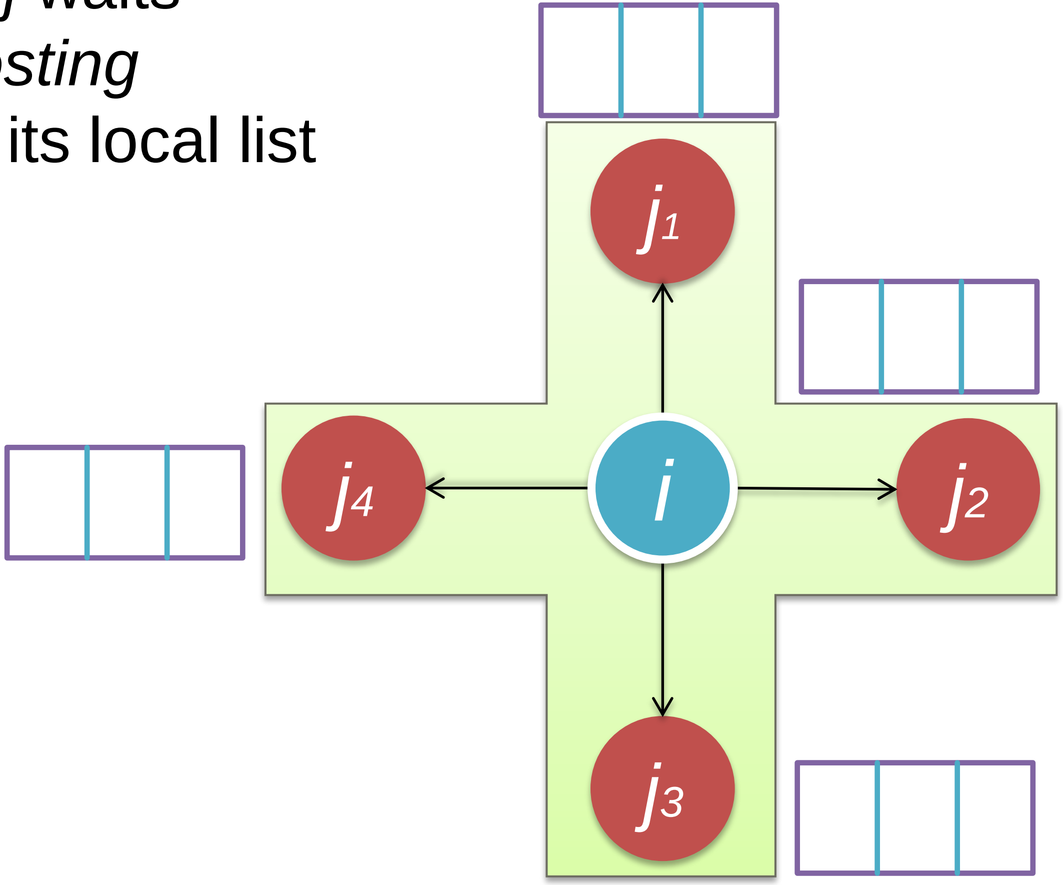
PSCW Scalable Post/Start Matching

- Each starting process has a local list



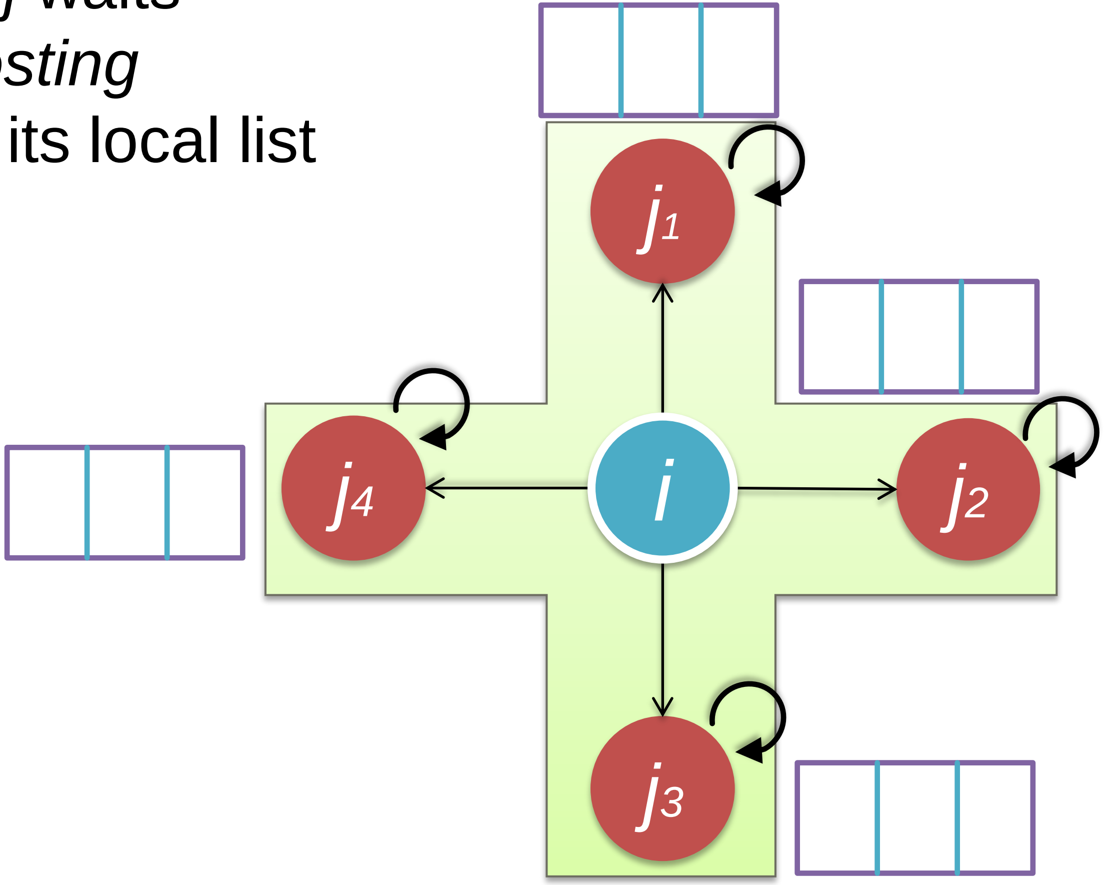
PSCW Scalable Post/Start Matching

- *Posting* process *i* adds its rank *i* to a list at each *starting* process $j_1, \ldots, j_4$
- Each *starting* process *j* waits until the rank of the *posting* process *i* is present in its local list



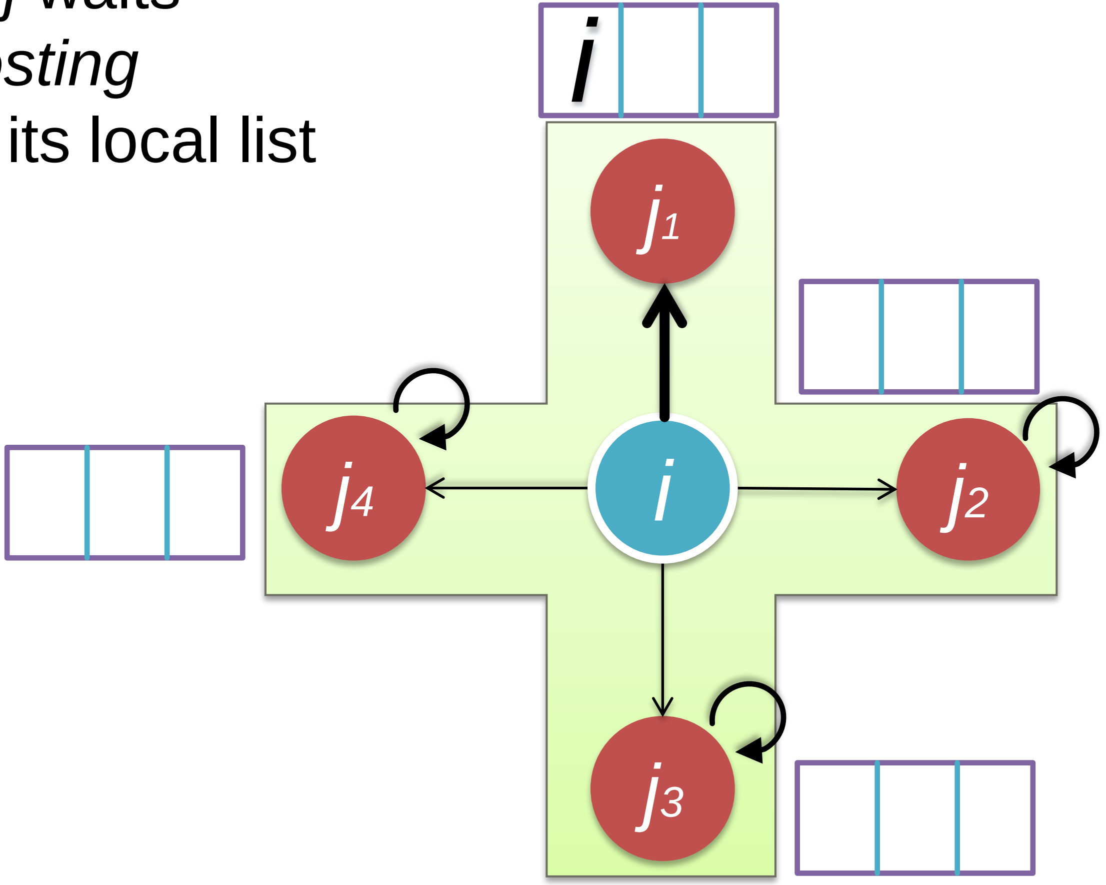
PSCW Scalable Post/Start Matching

- *Posting* process *i* adds its rank *i* to a list at each *starting* process $j_1, \ldots, j_4$
- Each *starting* process *j* waits until the rank of the *posting* process *i* is present in its local list



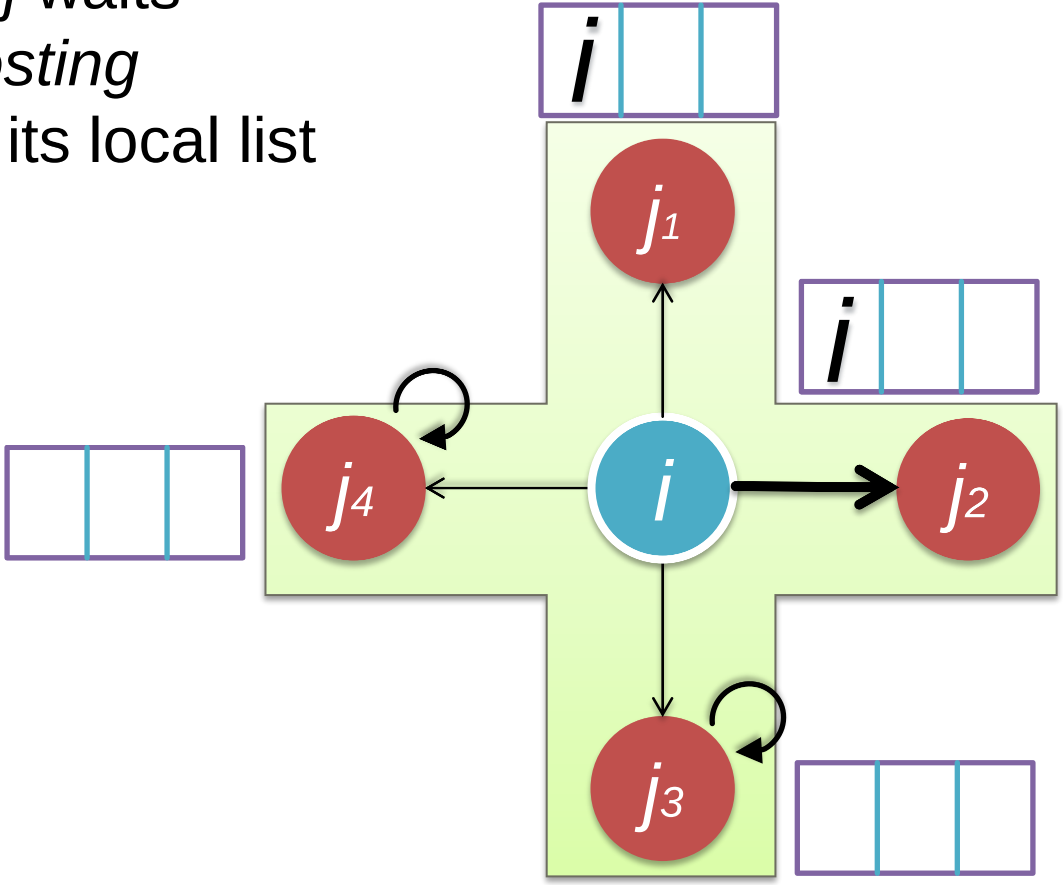
PSCW Scalable Post/Start Matching

- *Posting* process *i* adds its rank *i* to a list at each *starting* process $j_1, \ldots, j_4$
- Each *starting* process *j* waits until the rank of the *posting* process *i* is present in its local list



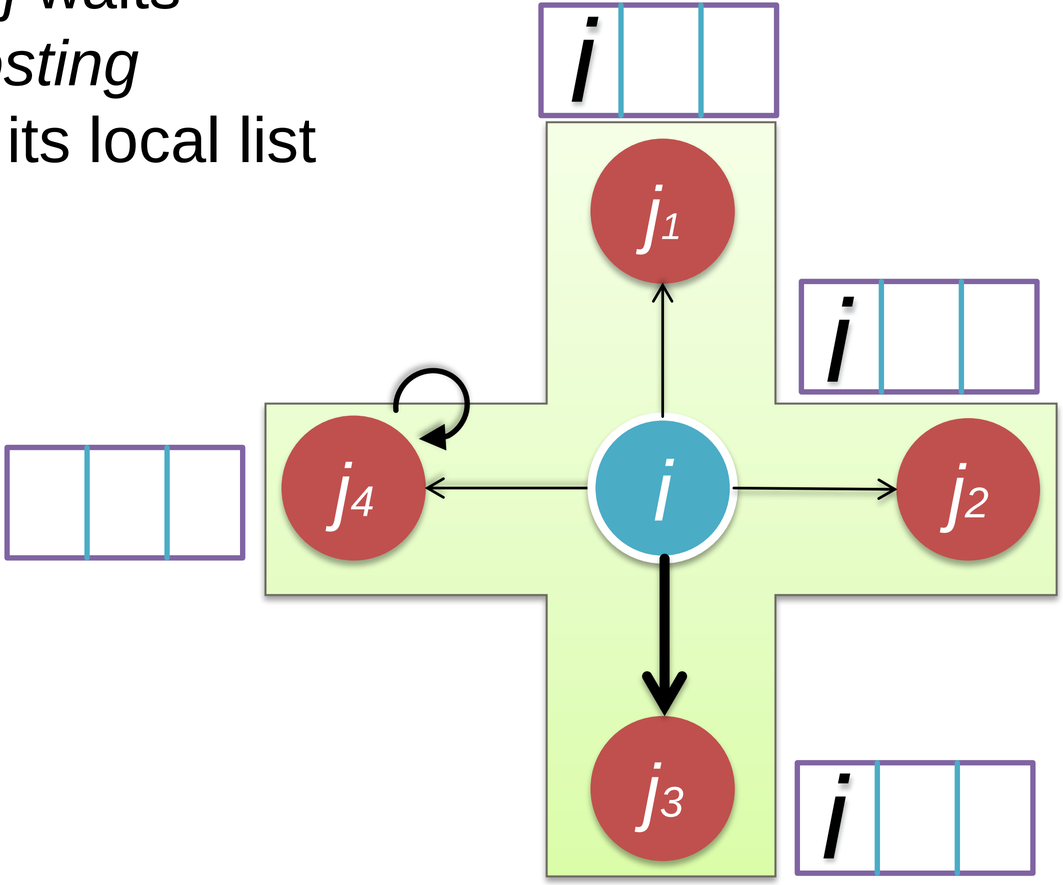
PSCW Scalable Post/Start Matching

- *Posting* process *i* adds its rank *i* to a list at each *starting* process $j_1, \ldots, j_4$
- Each *starting* process *j* waits until the rank of the *posting* process *i* is present in its local list



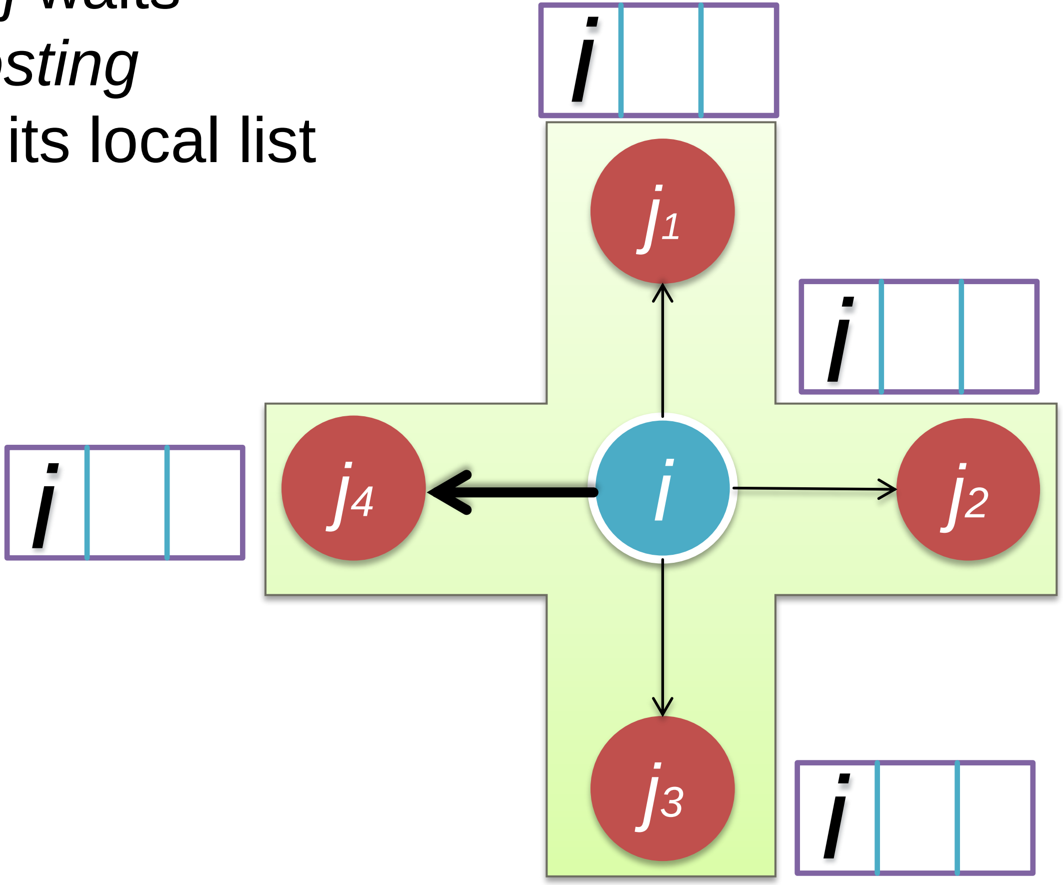
PSCW Scalable Post/Start Matching

- *Posting* process *i* adds its rank *i* to a list at each *starting* process $j_1, \ldots, j_4$
- Each *starting* process *j* waits until the rank of the *posting* process *i* is present in its local list



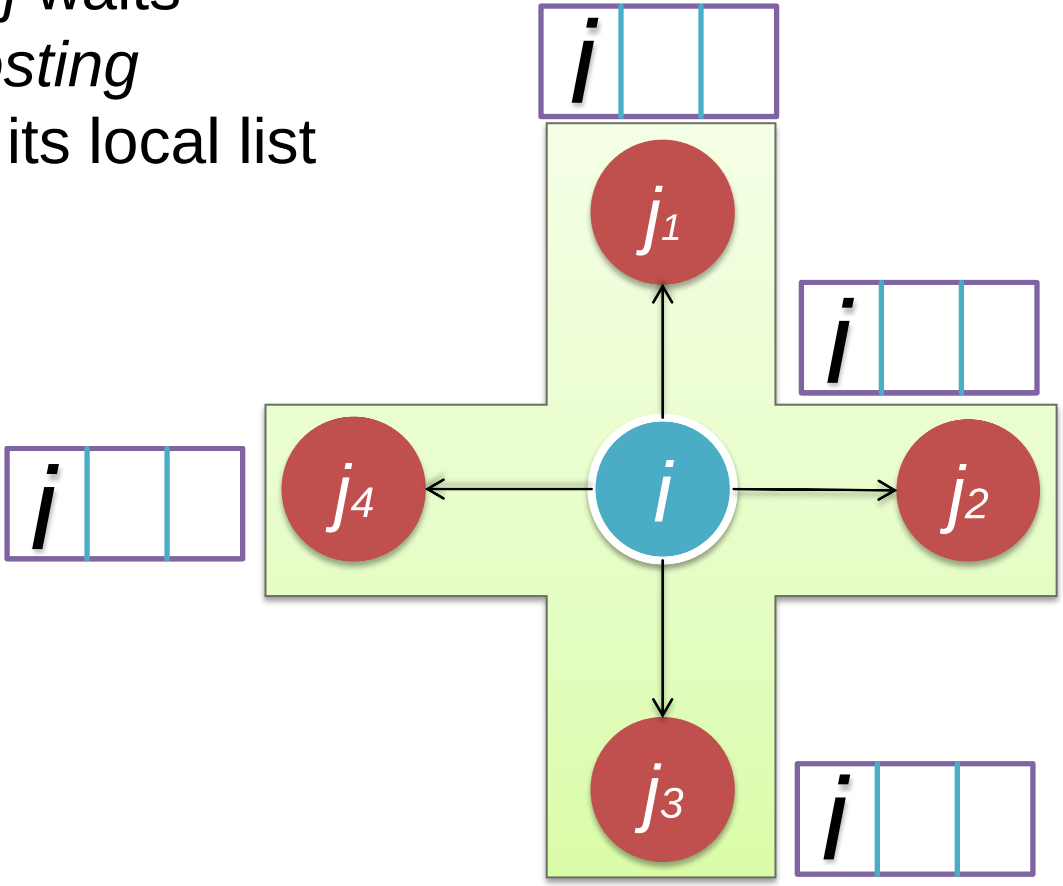
PSCW Scalable Post/Start Matching

- *Posting* process *i* adds its rank *i* to a list at each *starting* process $j_1, \dots, j_4$
- Each *starting* process *j* waits until the rank of the *posting* process *i* is present in its local list



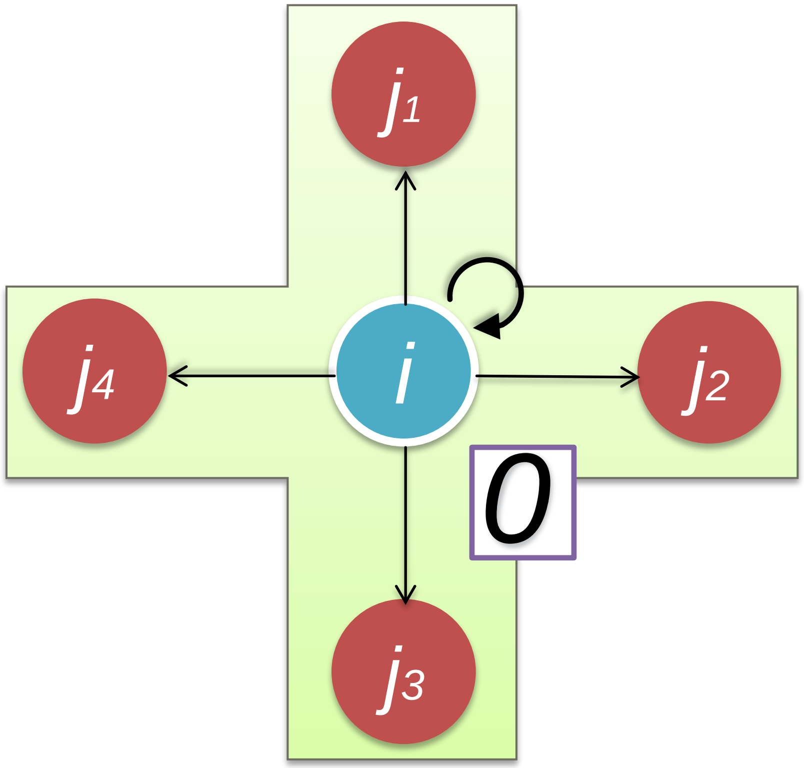
PSCW Scalable Post/Start Matching

- *Posting* process *i* adds its rank *i* to a list at each *starting* process $j_1, \ldots, j_4$
- Each *starting* process *j* waits until the rank of the *posting* process *i* is present in its local list



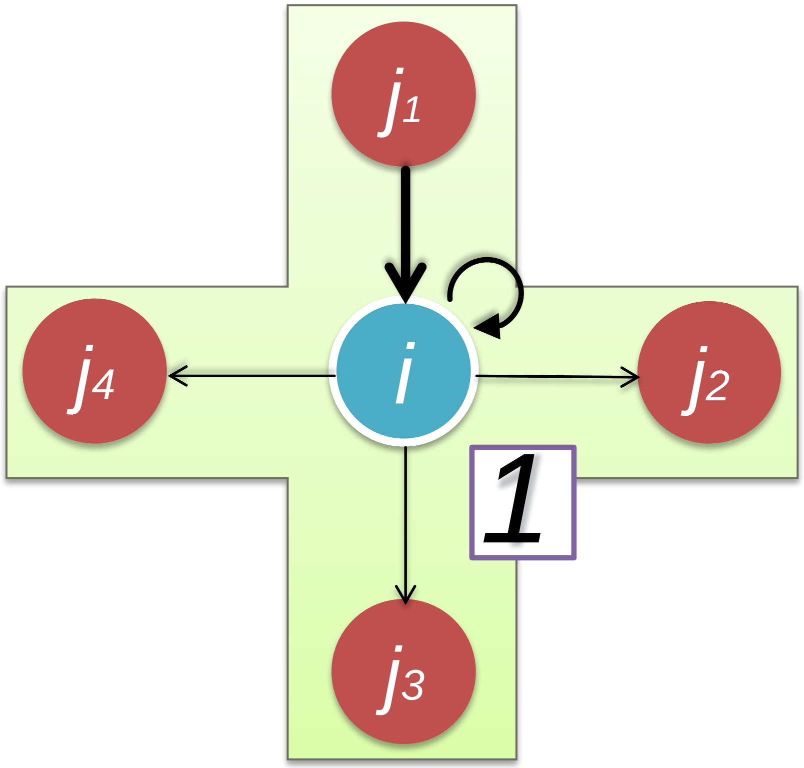
PSCW Scalable Complete/Wait Matching

- Each *starting* process increments a counter stored at the *posting* process



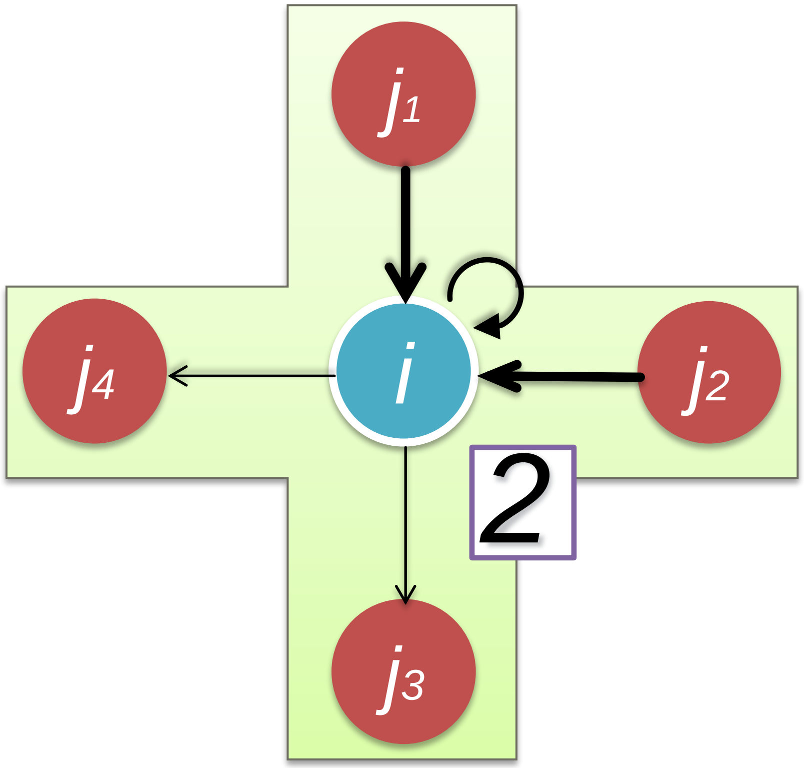
PSCW Scalable Complete/Wait Matching

- Each *starting* process increments a counter stored at the *posting* process



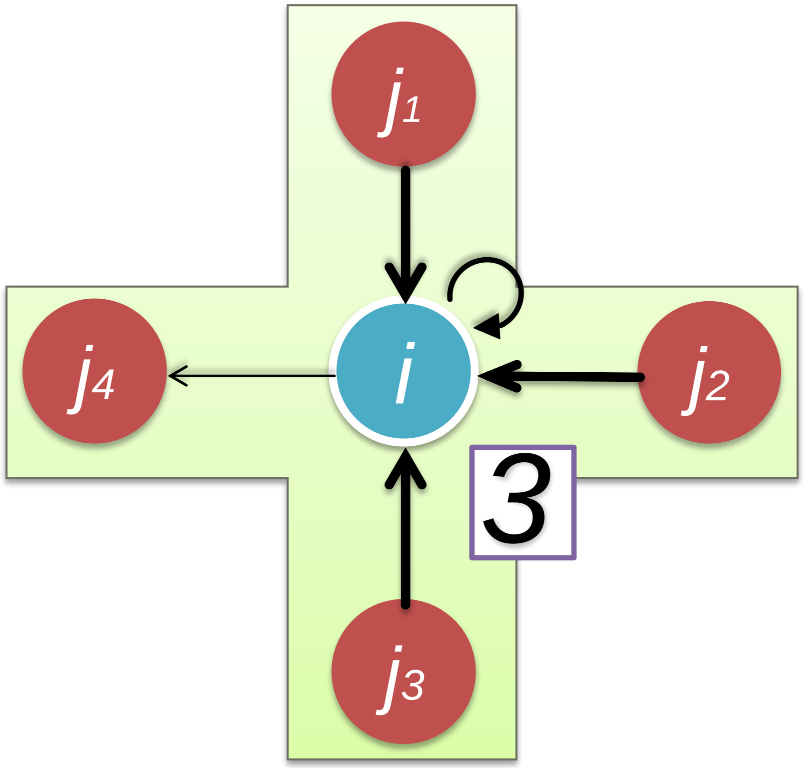
PSCW Scalable Complete/Wait Matching

- Each *starting* process increments a counter stored at the *posting* process



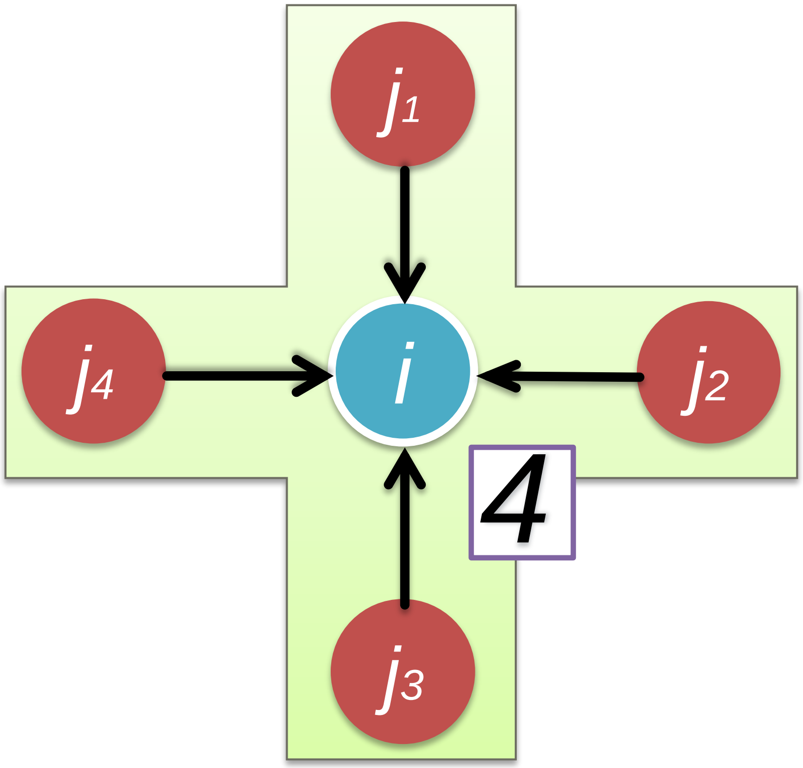
PSCW Scalable Complete/Wait Matching

- Each *starting* process increments a counter stored at the *posting* process



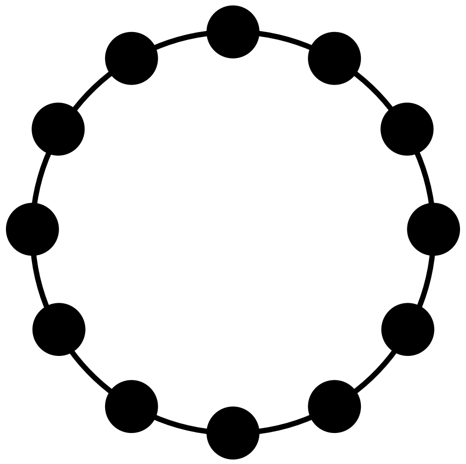
PSCW Scalable Complete/Wait Matching

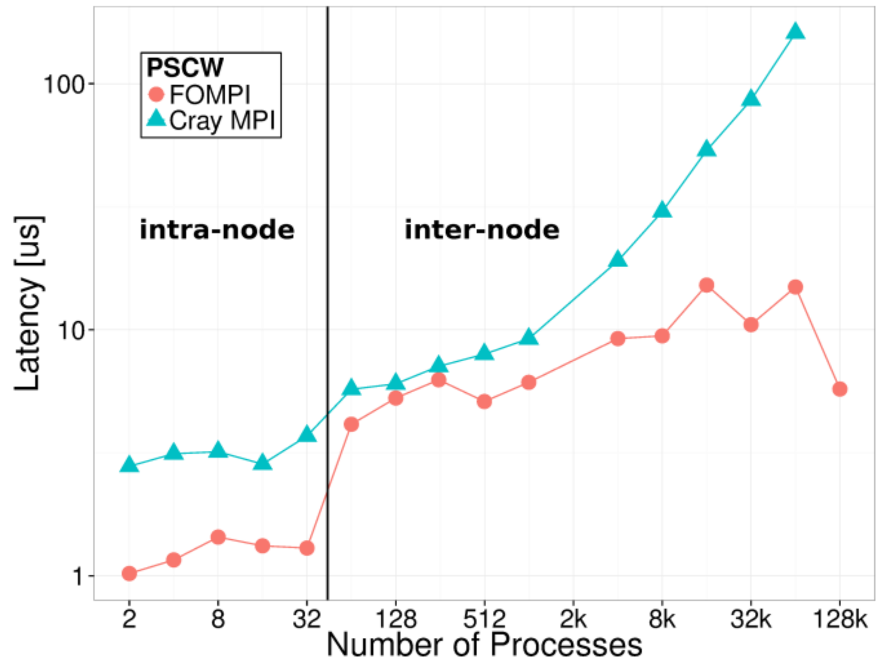
- Each *starting* process increments a counter stored at the *posting* process
- When the counter is equal to the number of *starting* processes, the *posting* process returns from wait



PSCW Performance

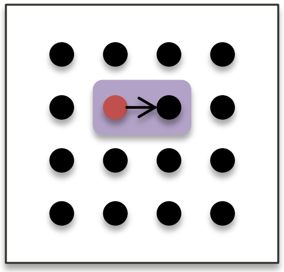
Time bound
$\mathcal{P}_{start} = \mathcal{P}_{wait} = \mathcal{O}(1)$ $\mathcal{P}_{post} = \mathcal{P}_{complete} = \mathcal{O}(\log p)$
Memory bound
$\mathcal{O}(\log p)$ (for scalable programs)



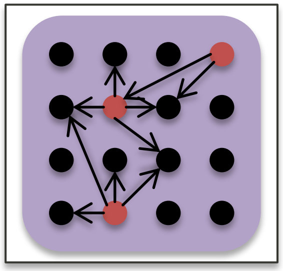


Ring Topology

Scalable Lock Synchronization



Lock/Unlock (shared/exclusive)

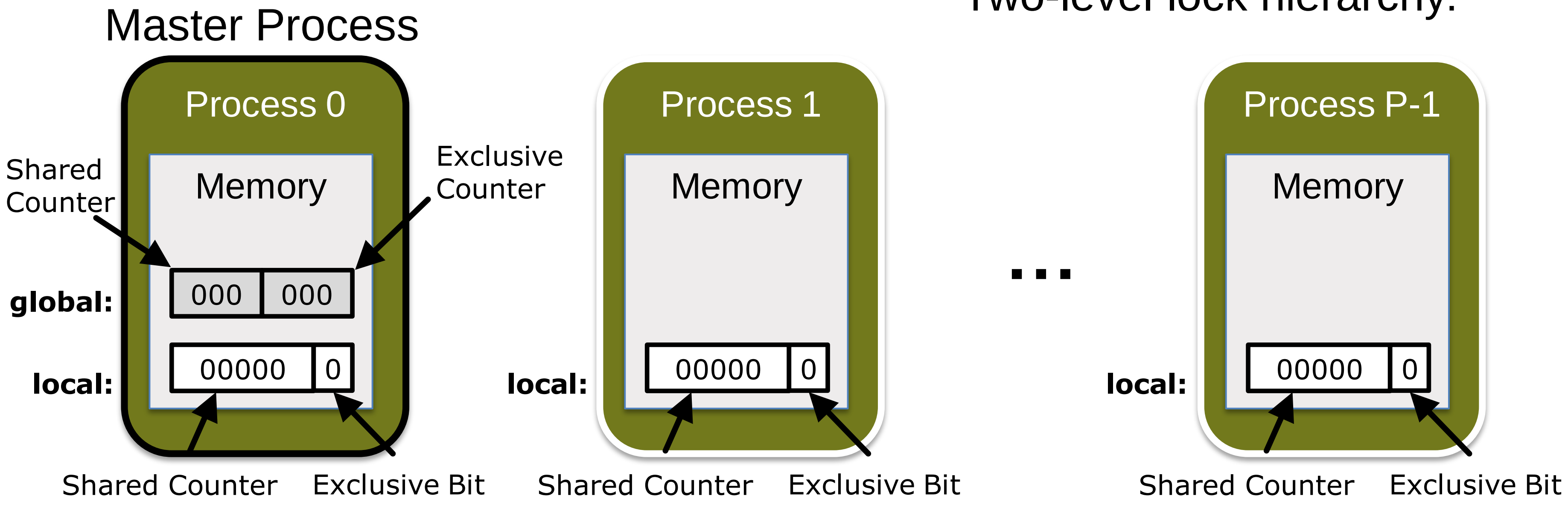


Lock All (always shared)

● Active process

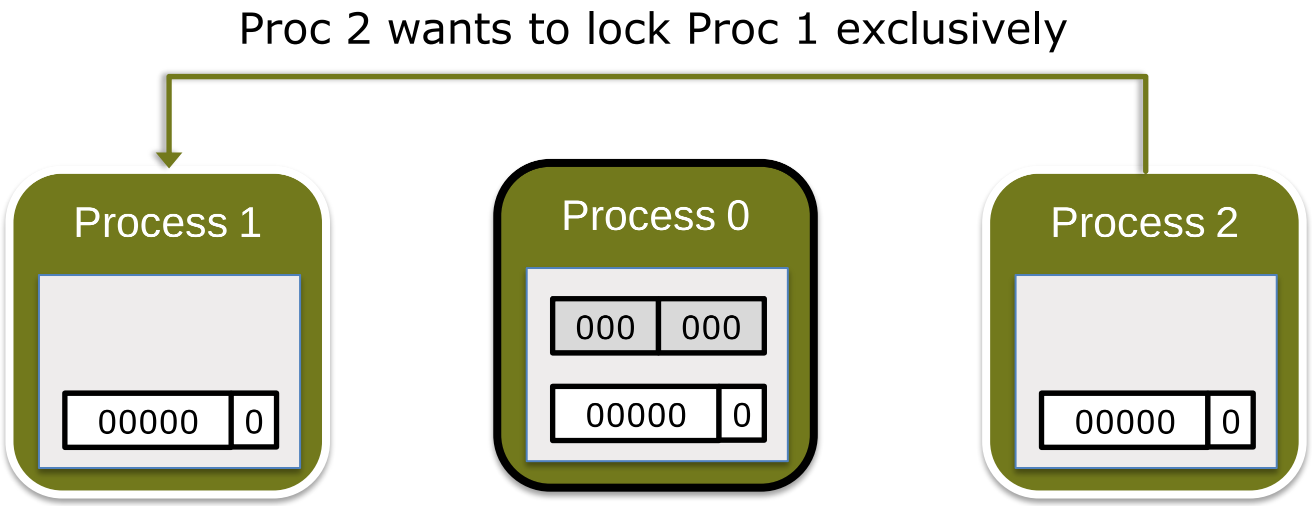
● Passive process

Two-level lock hierarchy:



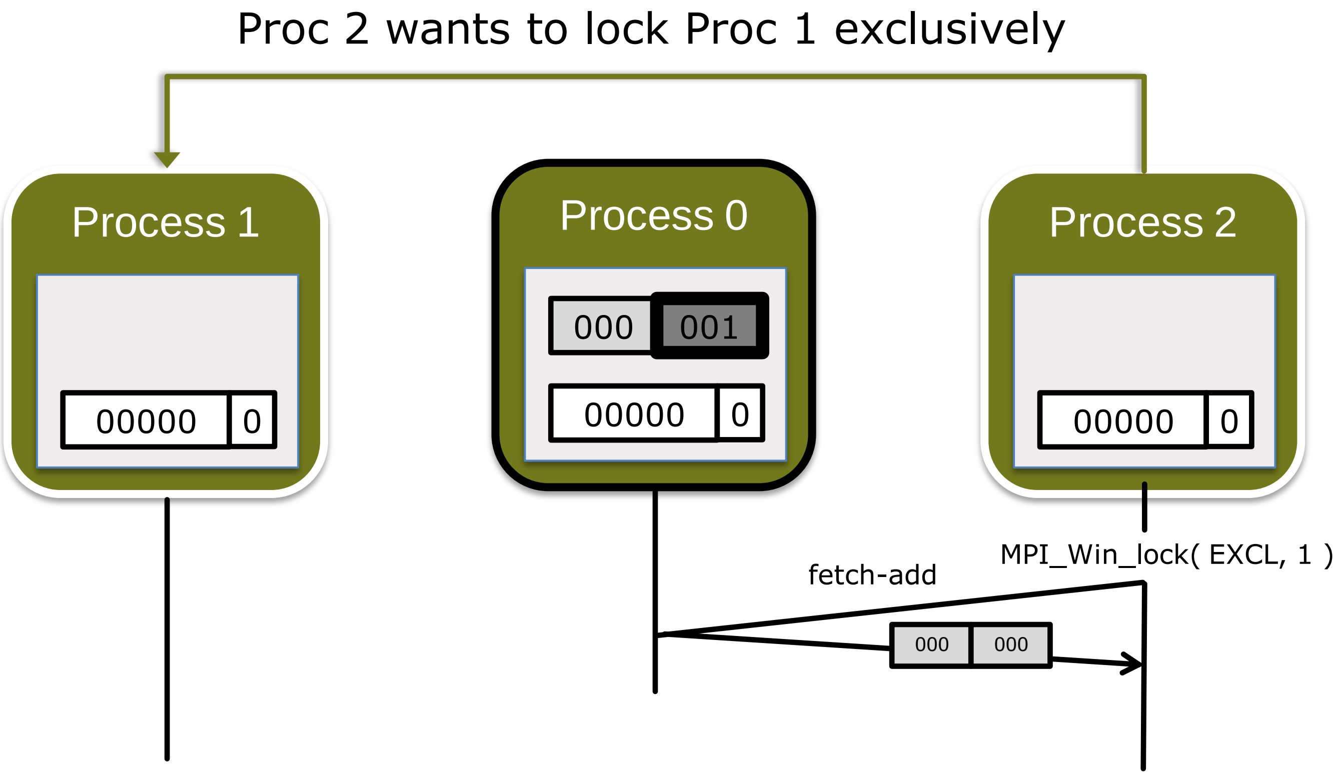
Exclusive Local Lock: Two Phases

- PHASE 1: increment the global exclusive counter
  (Invariant 1: no global shared lock held concurrently)



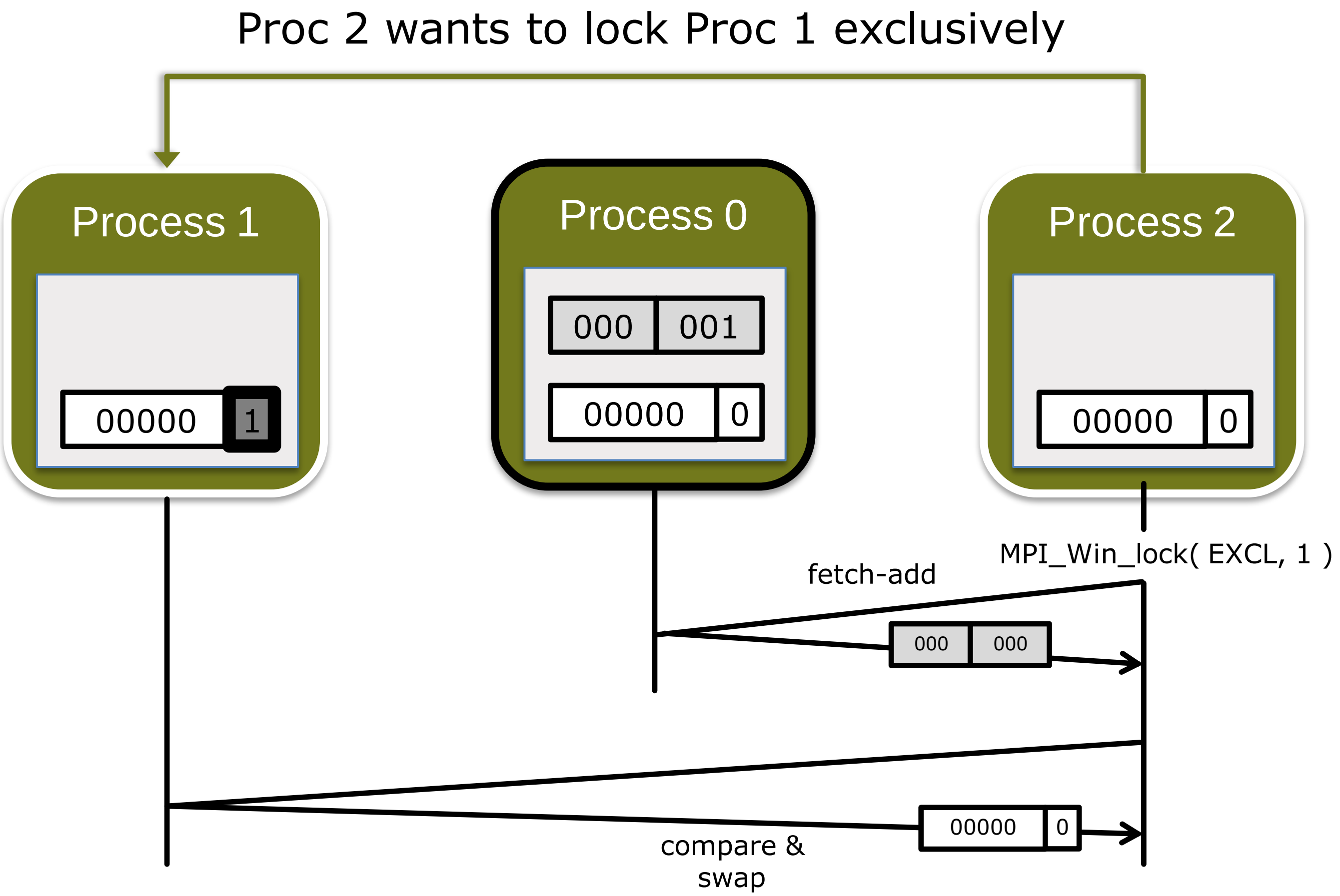
Exclusive Local Lock: Two Phases

- PHASE 1: increment the global exclusive counter
  (Invariant 1: no global shared lock held concurrently)



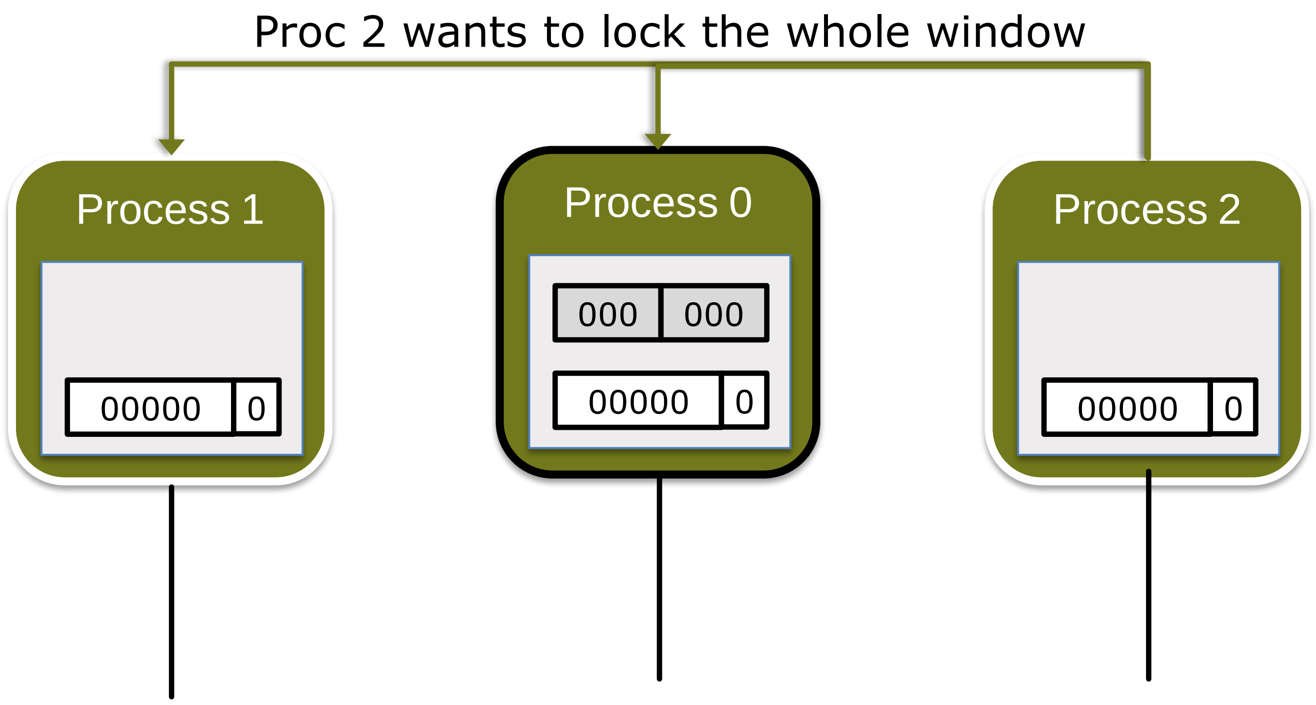
Exclusive Local Lock: Two Phases

- PHASE 1: increment the global exclusive counter
  (Invariant 2: no local shared/exclusive lock held concurrently)



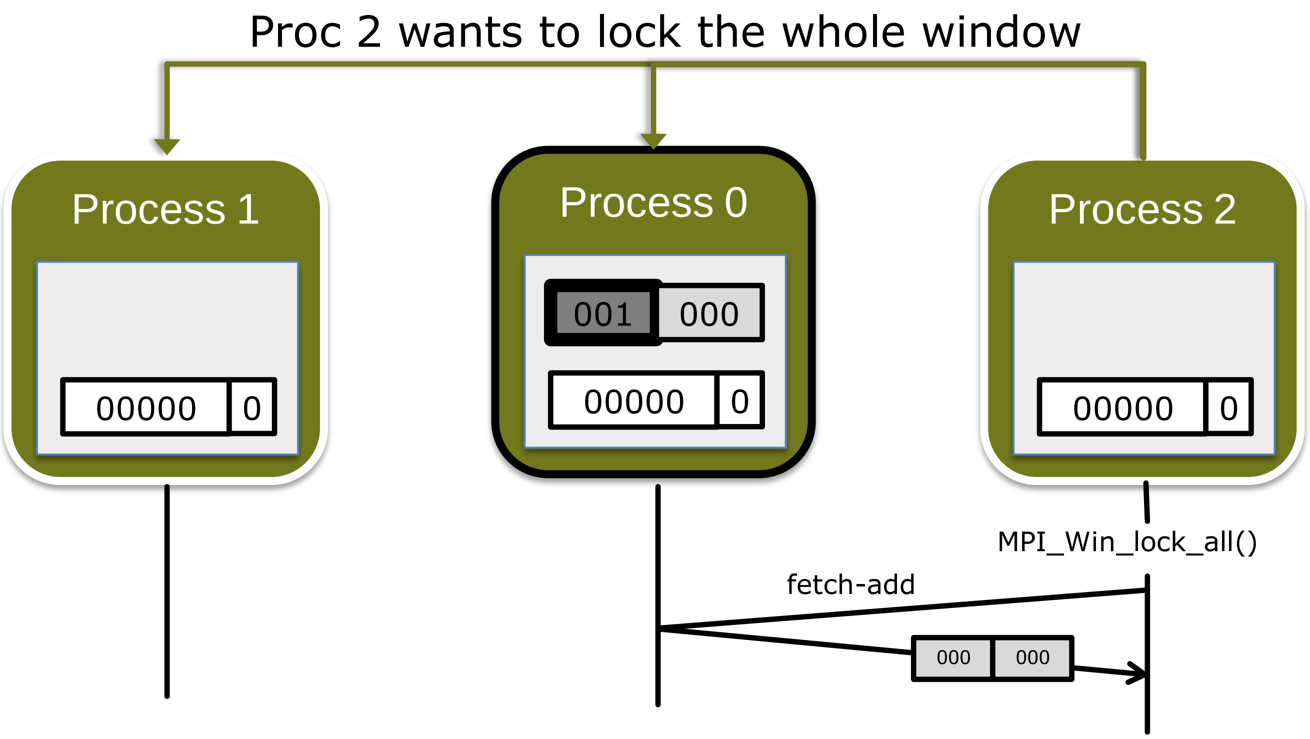
Shared Global Lock: One Phase

- Increment global shared counter
  (Invariant: no local exclusive lock is held concurrently)



Shared Global Lock: One Phase

- Increment global shared counter
  (Invariant: no local exclusive lock is held concurrently)

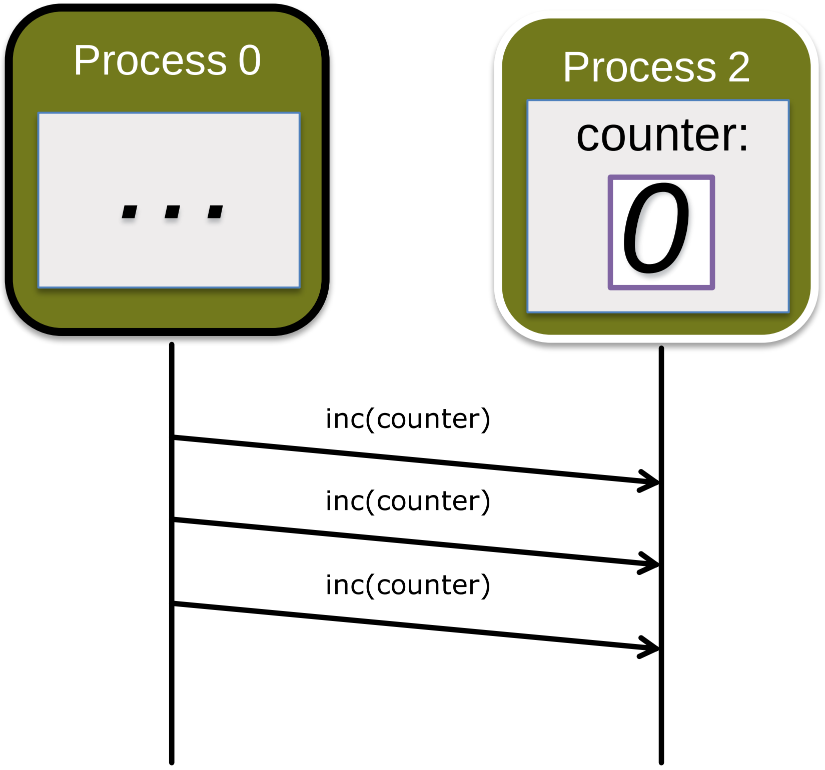


- Constant number of operations for p processes ☺

FLUSH SYNCHRONIZATION

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

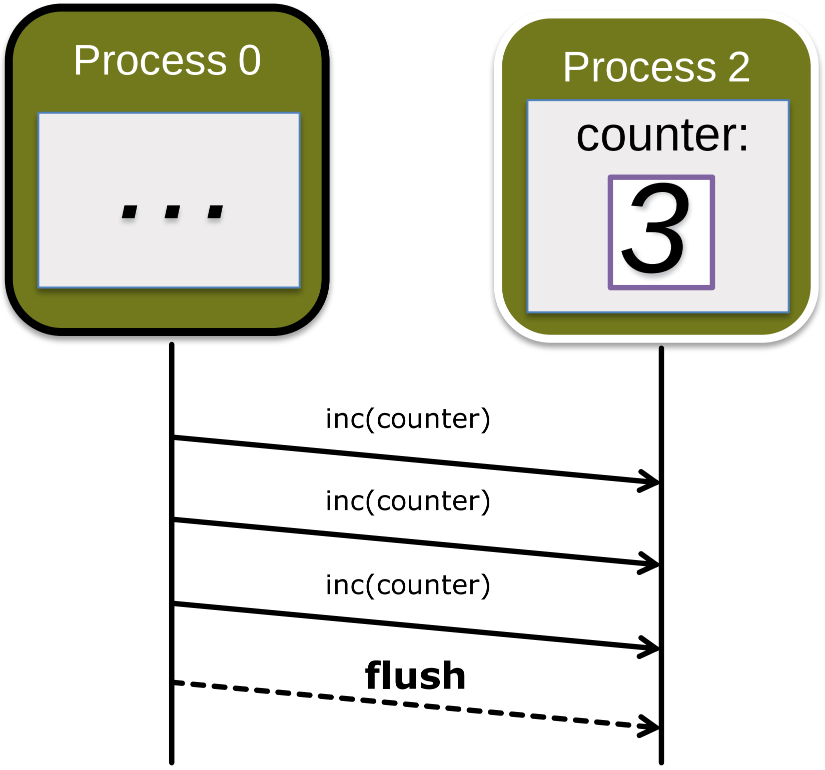
- Guarantees remote completion
- Issues a remote bulk synchronization and an x86 `mfence`
- One of the most performance critical functions, we add only 78 x86 CPU instructions to the critical path

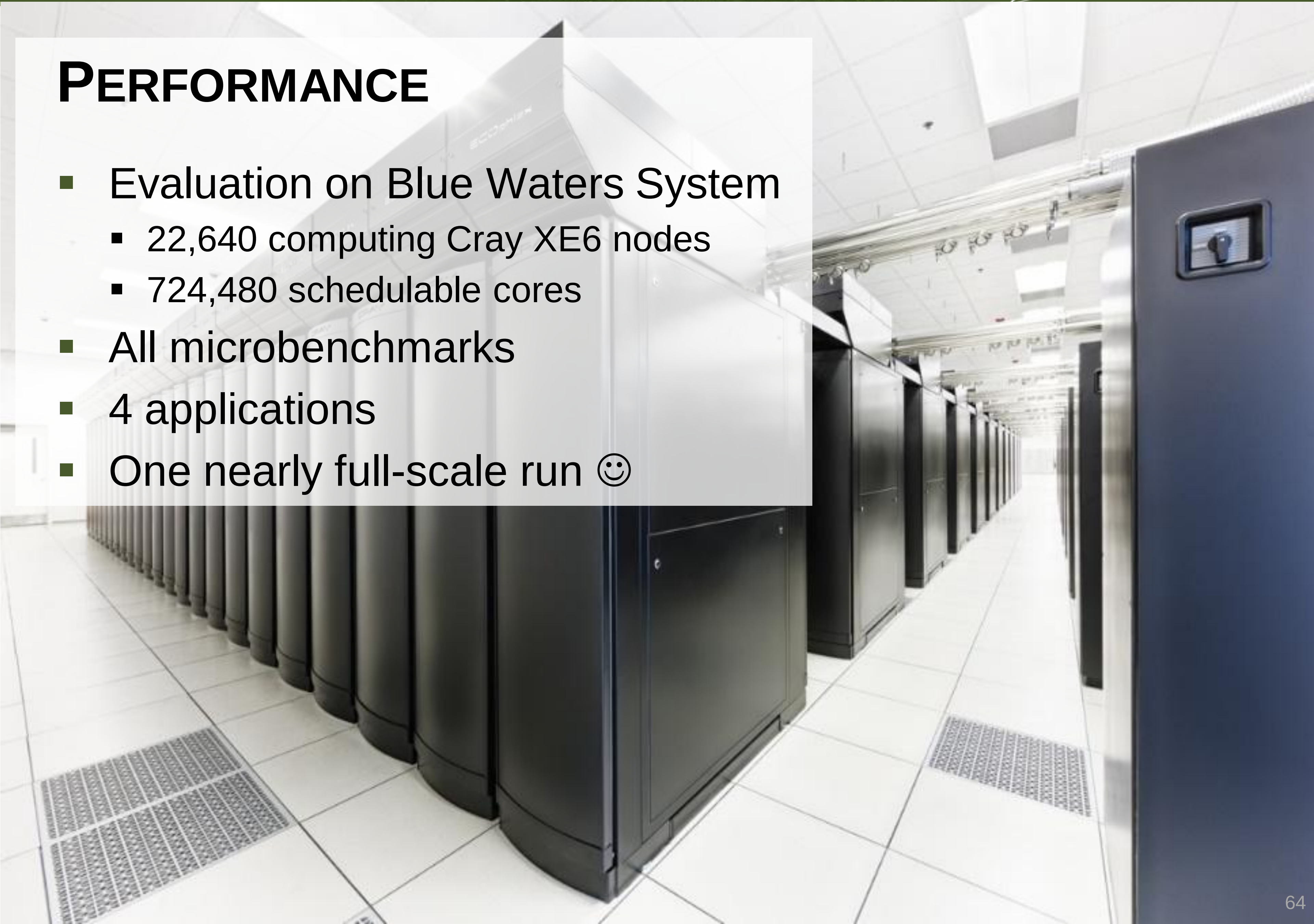


Flush Synchronization

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

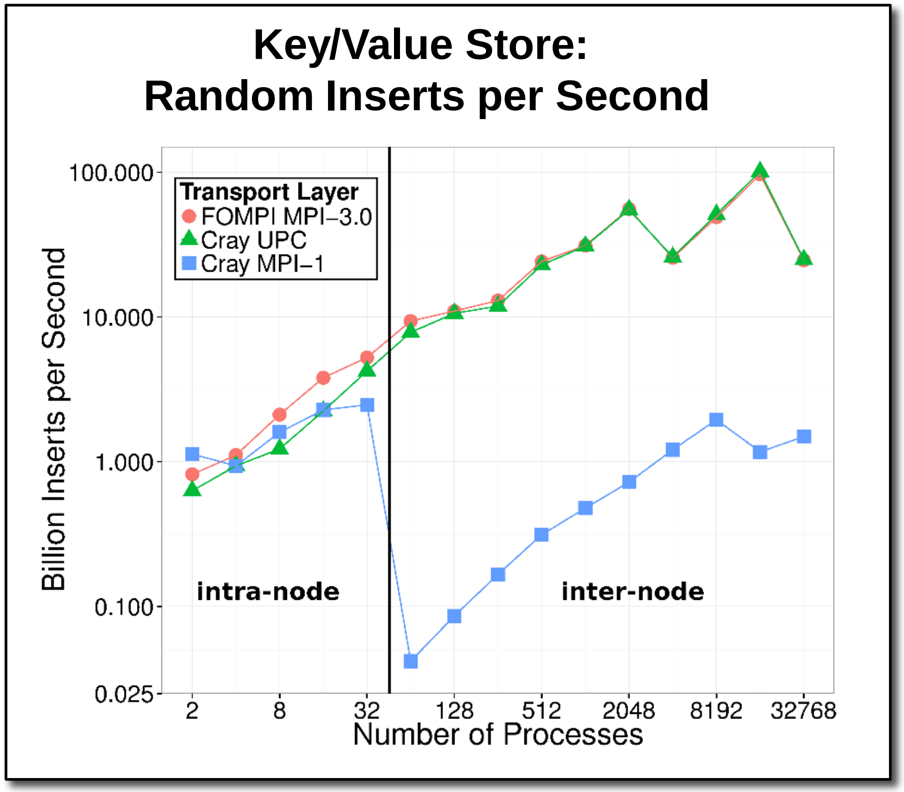


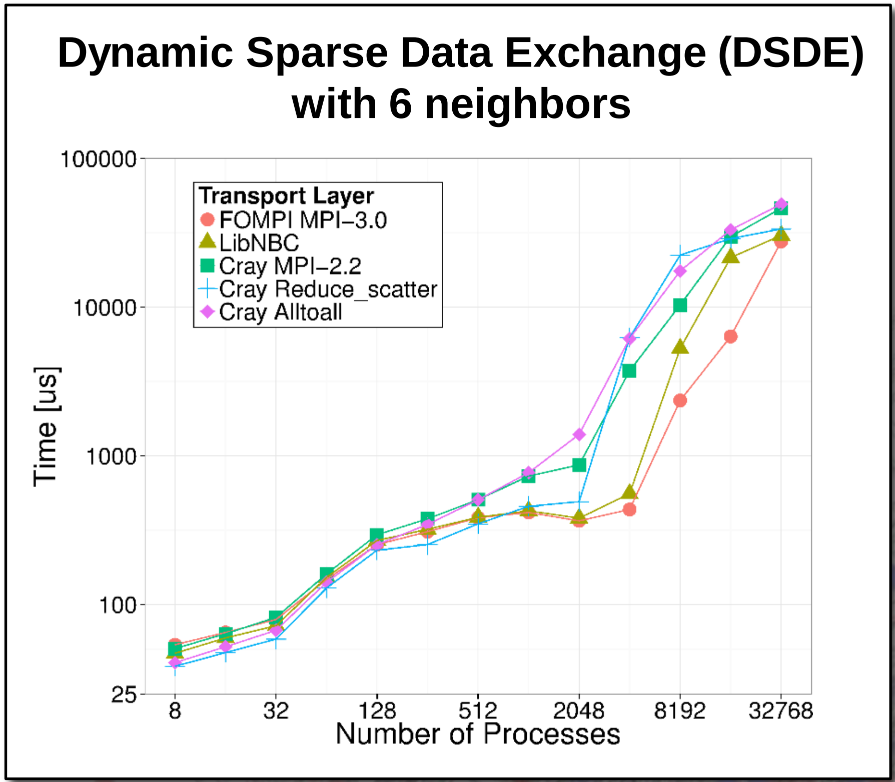


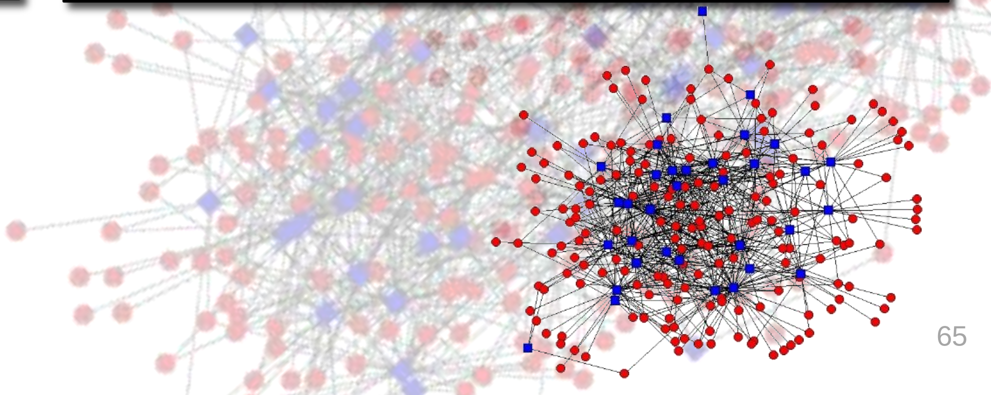
PERFORMANCE

- 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



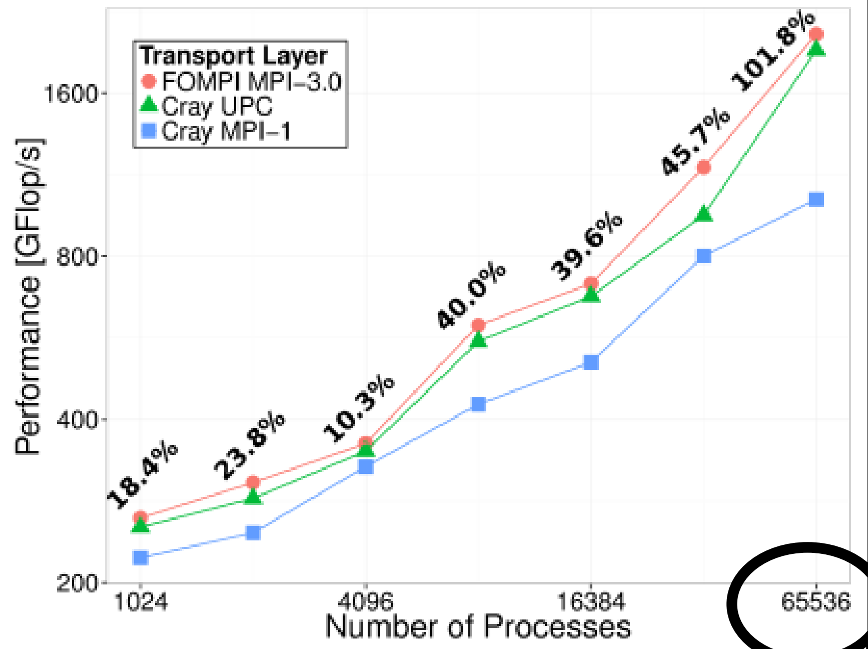




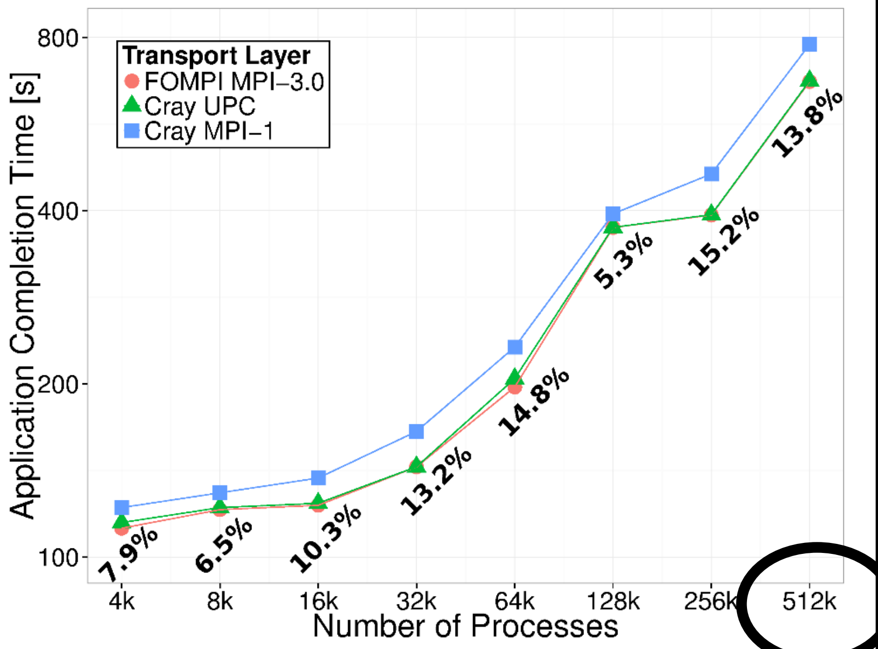
Performance: Applications

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

NAS 3D FFT [1] Performance



MILC [2] Application Execution Time



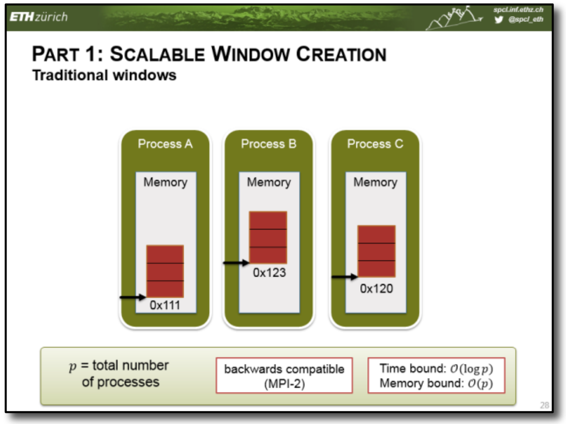
scale to 65k procs

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


PART 1: SCALABLE WINDOW CREATION
Traditional windows

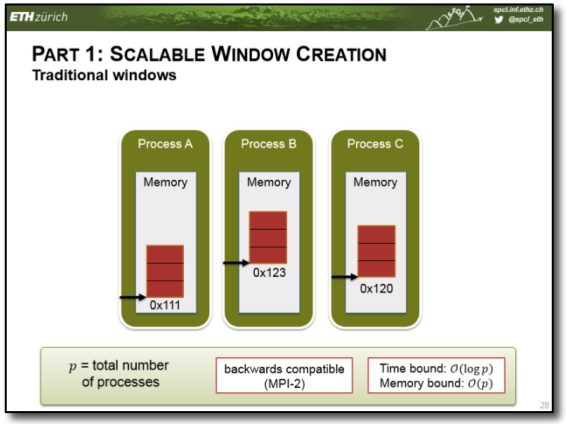
p = total number of processes

backwards compatible (MPI-2)

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

1. MPI window creation routines

CONCLUSIONS & SUMMARY



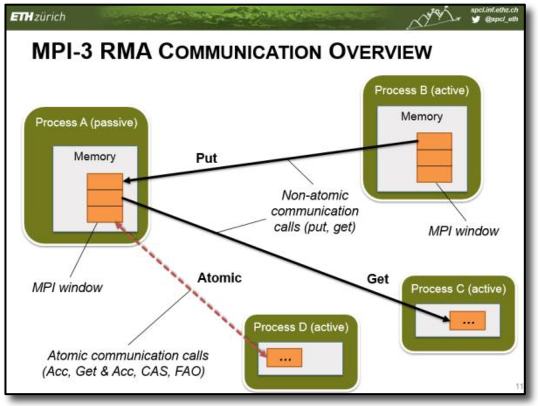
PART 1: SCALABLE WINDOW CREATION
Traditional windows

p = total number of processes

backwards compatible (MPI-2)

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

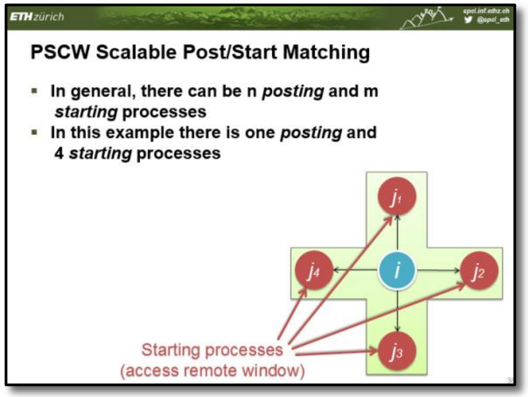
1. MPI window creation routines



MPI-3 RMA COMMUNICATION OVERVIEW

2. Non-atomic & atomic communication

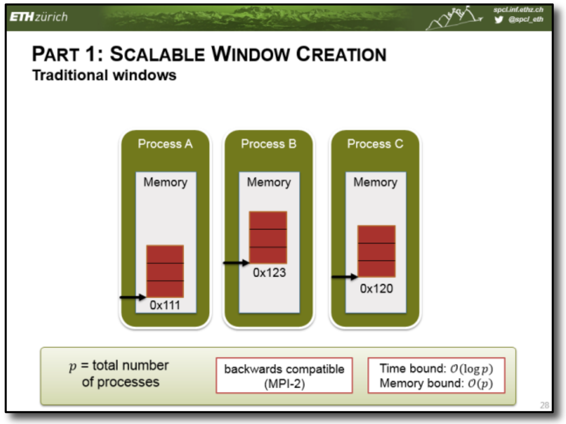
CONCLUSIONS & SUMMARY



PSCW Scalable Post/Start Matching

- In general, there can be n *posting* and m *starting* processes
- In this example there is one *posting* and 4 *starting* processes

3. Fence / PSCW



PART 1: SCALABLE WINDOW CREATION

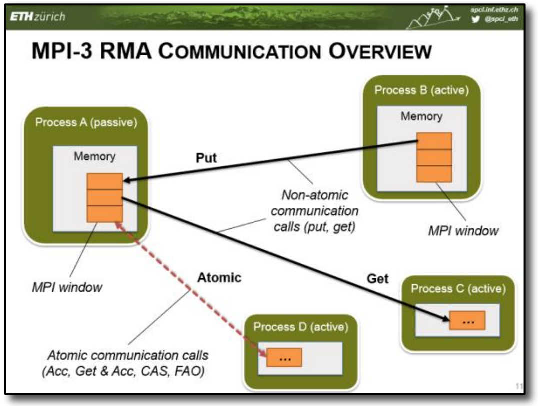
Traditional windows

p = total number of processes

backwards compatible (MPI-2)

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

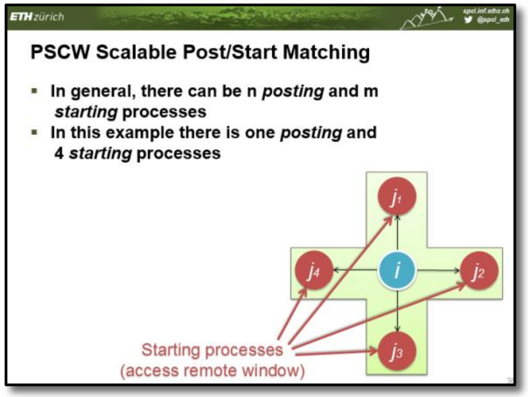
1. MPI window creation routines



MPI-3 RMA COMMUNICATION OVERVIEW

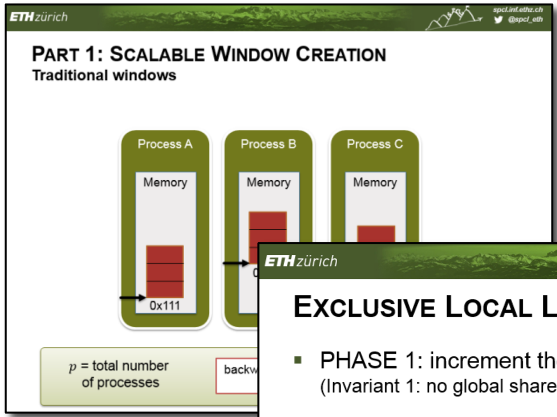
2. Non-atomic & atomic communication

Conclusions & Summary


PSCW Scalable Post/Start Matching

- In general, there can be n *posting* and m *starting* processes
- In this example there is one *posting* and 4 *starting* processes

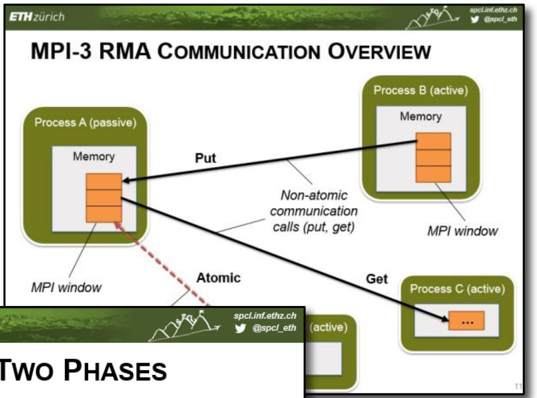
3. Fence / PSCW


Part 1: Scalable Window Creation

Traditional windows

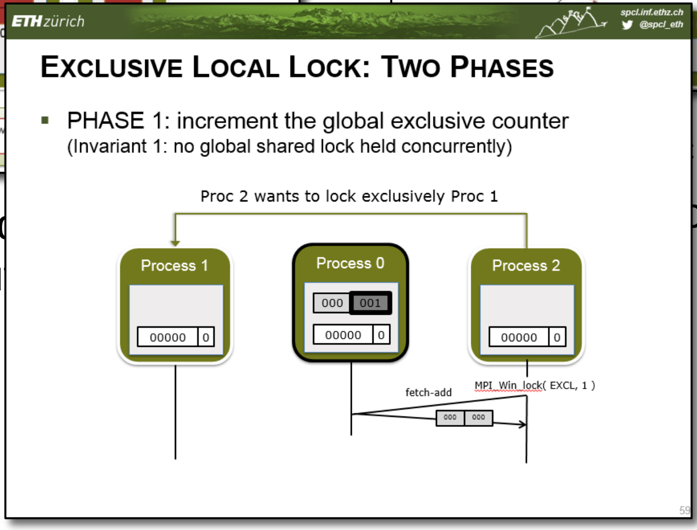
1. MPI win

rou


MPI-3 RMA Communication Overview

& atomic

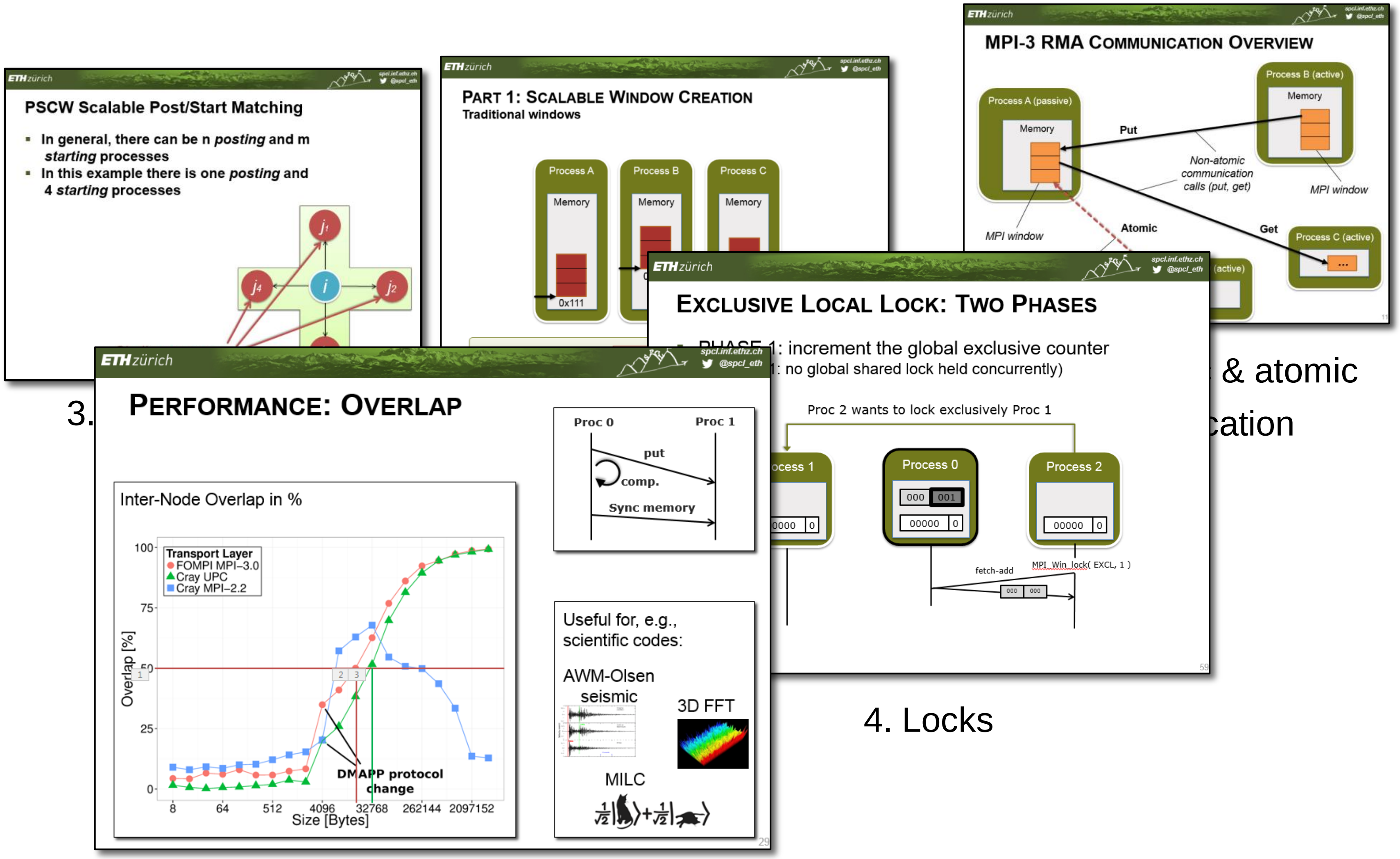
cation


Exclusive Local Lock: Two Phases

- PHASE 1: increment the global exclusive counter (Invariant 1: no global shared lock held concurrently)

4. Locks

Conclusions & Summary



ETH zürich

PSCW Scalable Post/Start Matching

- In general, there can be n *posting* and m *starting* processes
- In this example there is one *posting* and 4 *starting* processes

Part 1: Scalable Window Creation

Traditional windows

Process A, Process B, Process C, Memory, 0x111

MPI-3 RMA Communication Overview

Process A (passive), Memory, Put, Process B (active), Memory, Non-atomic communication calls (put, get), MPI window, Atomic, Get, Process C (active), MPI window

Exclusive Local Lock: Two Phases

PHASE 1: increment the global exclusive counter
1: no global shared lock held concurrently)

Proc 2 wants to lock exclusively Proc 1

Process 1, Process 0, Process 2, 000, 001, 00000, 0, fetch-add, MPI_Win_lock(EXCL, 1)

Performance: Overlap

Proc 0, Proc 1, put, comp., Sync memory

Inter-Node Overlap in %

Transport Layer: ● FOMPI MPI-3.0, ▲ Cray UPC, ■ Cray MPI-2.2

Overlap [%], Size [Bytes], DMAPP protocol change

Useful for, e.g., scientific codes:

AWM-Olsen seismic, 3D FFT, MILC

3.

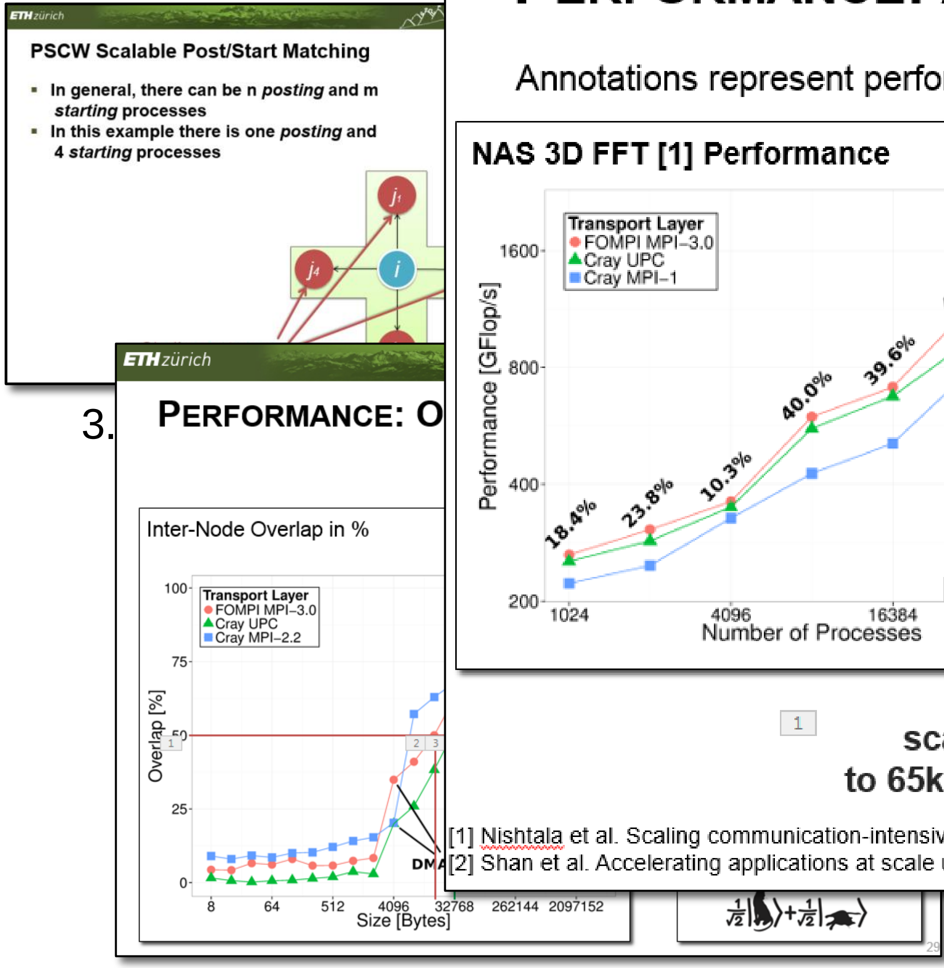
& atomic

cation

4. Locks

5. foMPI reference implementation

Conclusions & Summary


ETH zürich

PSCW Scalable Post/Start Matching

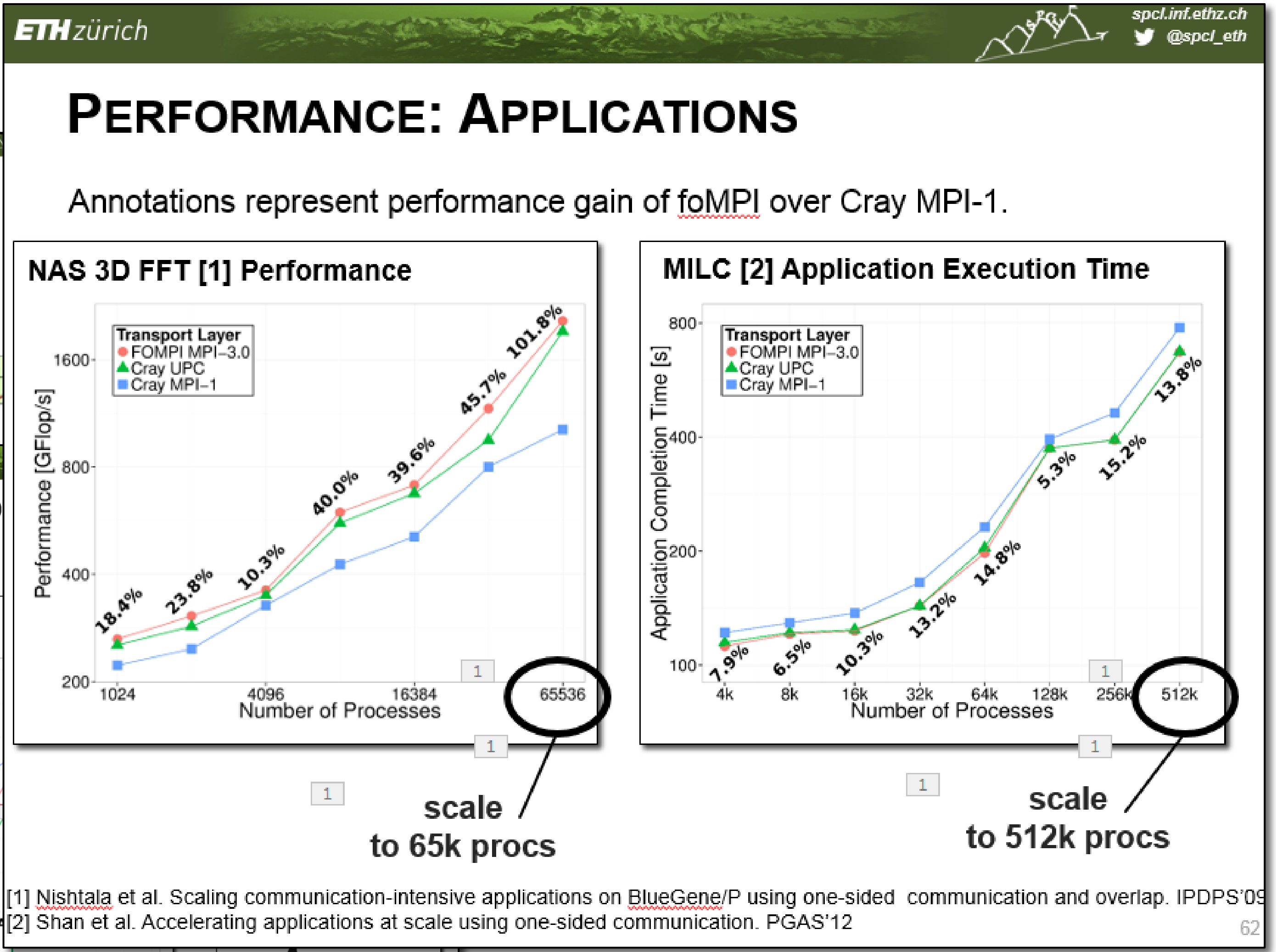
- In general, there can be n *posting* and m *starting* processes
- In this example there is one *posting* and 4 *starting* processes

3.

ETH zürich

Performance: O

Inter-Node Overlap in %


ETH zürich spcl.inf.ethz.ch @spcl_eth

Performance: Applications

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

scale to 65k procs

scale to 512k procs

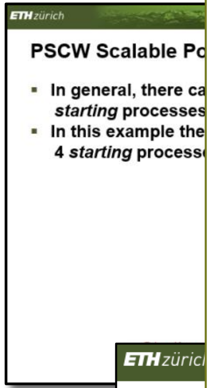
[1] Nishtala et al. Scaling communication-intensive applications on BlueGene/P using one-sided communication and overlap. IPDPS'09
[2] Shan et al. Accelerating applications at scale using one-sided communication. PGAS'12

62

5. foMPI reference implementation

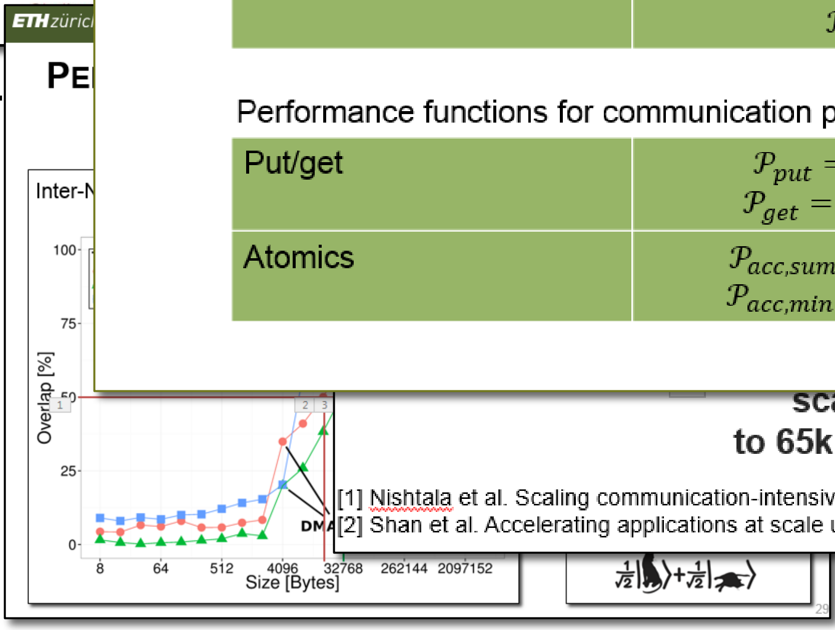
6. Application implementation & evaluation

Conclusions & Summary


PSCW Scalable Po

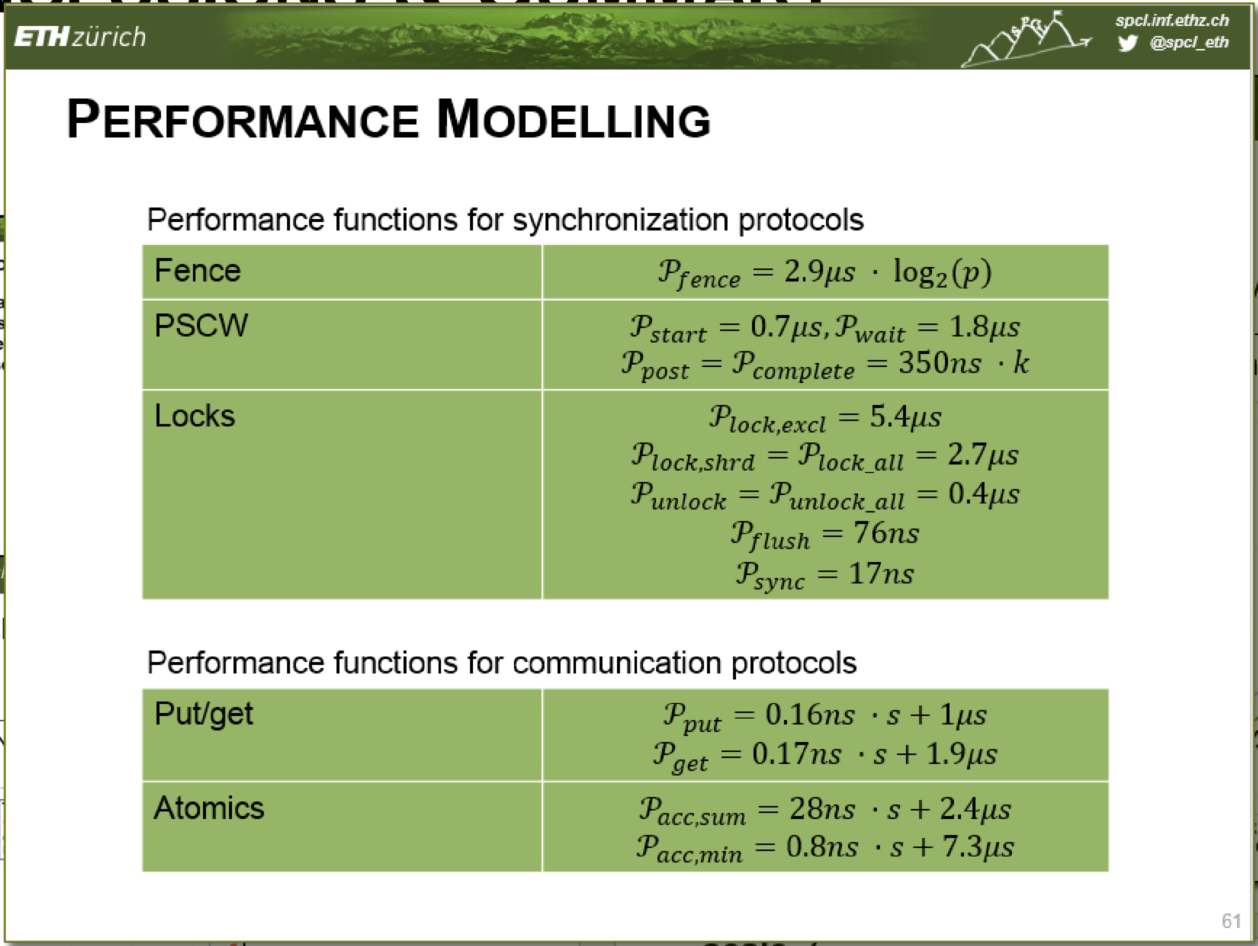
- In general, there ca *starting* processes
- In this example the 4 *starting* process

3.


PE

scale to 65k procs

5. foMPI reference implementation


Performance Modelling

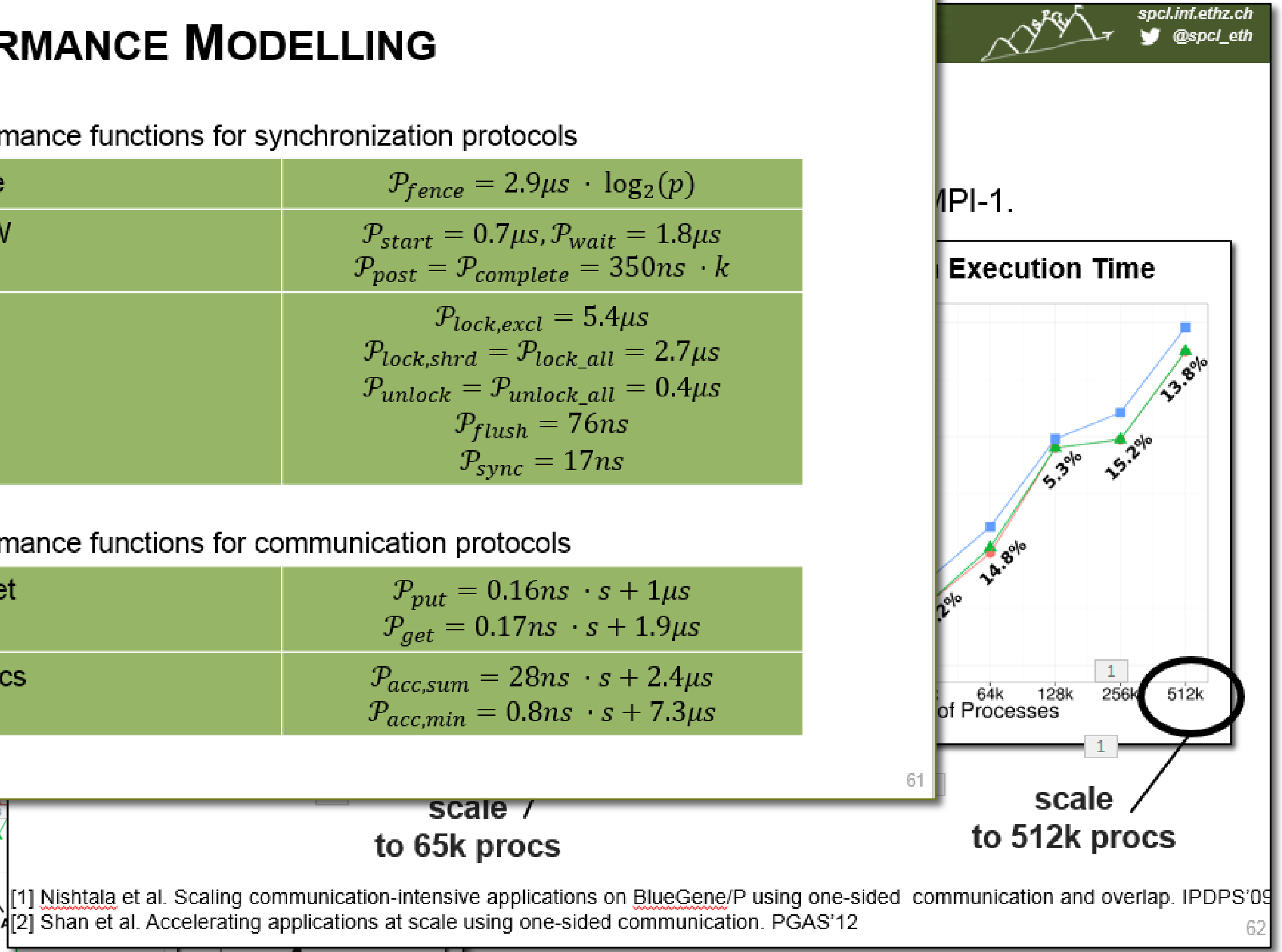
Performance functions for synchronization protocols

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	$\mathcal{P}_{acc,sum} = 28ns \cdot s + 2.4\mu s$ $\mathcal{P}_{acc,min} = 0.8ns \cdot s + 7.3\mu s$

61


MPI-1.

Execution Time

scale to 512k procs

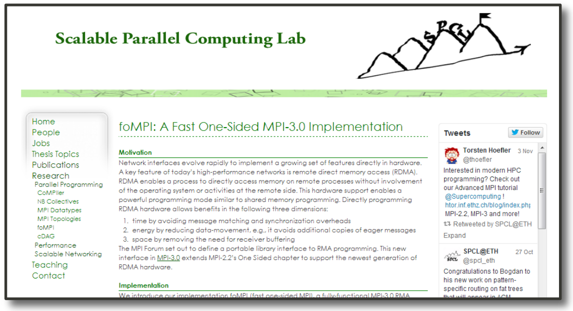
[1] Nishtala et al. Scaling communication-intensive applications on BlueGene/P using one-sided communication and overlap. IPDPS'09
[2] Shan et al. Accelerating applications at scale using one-sided communication. PGAS'12

62

6. Application implementation & evaluation

ACKNOWLEDGMENTS

- Try foMPI yourself:
  http://spcl.inf.ethz.ch/Research/Parallel_Programming/foMPI
- Ongoing work on DDT (see ACM Students Research Competition Poster)
- Thanks to: Timo Schneider, Greg Bauer, Bill Kramer, Duncan Roweth, Nick Wright, Paul Hargrove (and the whole UPC team) and the MPI Forum RMA WG …


Scalable Parallel Computing Lab

Home
People
Jobs
Thesis Topics
Publications
Research
Parallel Programming
CoMPIler
NB Collectives
MPI Datatypes
MPI Topologies
foMPI
cDAG
Performance
Scalable Networking
Teaching
Contact

foMPI: A Fast One-Sided MPI-3.0 Implementation

Motivation

Network interfaces evolve rapidly to implement a growing set of features directly in hardware. A key feature of today's high-performance networks is remote direct memory access (RDMA). RDMA enables a process to directly access memory on remote processes without involvement of the operating system or activities at the remote side. This hardware support enables a powerful programming mode similar to shared memory programming. Directly programming RDMA hardware allows benefits in the following three dimensions:

1. time by avoiding message matching and synchronization overheads
2. energy by reducing data-movement, e.g., it avoids additional copies of eager messages
3. space by removing the need for receiver buffering

The MPI Forum set out to define a portable library interface to RMA programming. This new interface in MPI-3.0 extends MPI-2.2's One Sided chapter to support the newest generation of RDMA hardware.

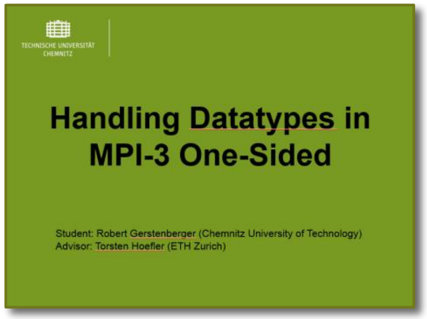
Implementation

We introduce our implementation foMPI (fast one-sided MPI), a fully-functional MPI-3.0 RMA

Tweets Follow

Torsten Hoefler 3 Nov
@thoefler
Interested in modern HPC programming? Check out our Advanced MPI tutorial @Supercomputing ! htor.inf.ethz.ch/blog/index.ph MPI-2.2, MPI-3 and more!
Retweeted by SPCL@ETH
Expand

SPCL@ETH 27 Oct
@spcl_eth
Congratulations to Bogdan to his new work on pattern-specific routing on fat trees

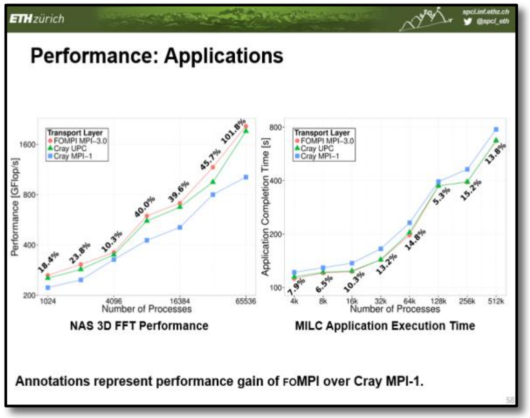

TECHNISCHE UNIVERSITÄT CHEMNITZ

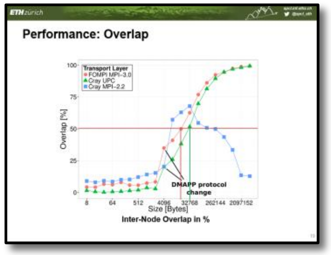
Handling Datatypes in MPI-3 One-Sided

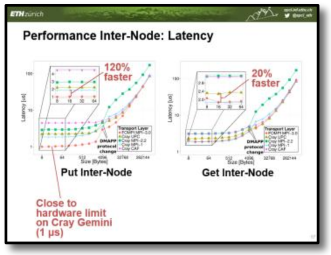
Student: Robert Gerstenberger (Chemnitz University of Technology)
Advisor: Torsten Hoefler (ETH Zurich)

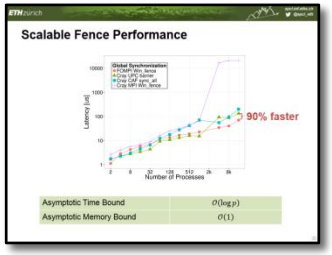
… and the institutions:

NSF · DEPARTMENT OF ENERGY · UNITED STATES OF AMERICA · BLUE WATERS SUSTAINED PETASCALE COMPUTING · Sandia National Laboratories

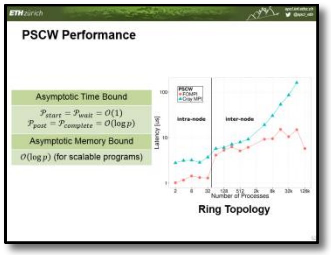

Performance: Applications
NAS 3D FFT Performance
MILC Application Execution Time
Annotations represent performance gain of foMPI over Cray MPI-1.

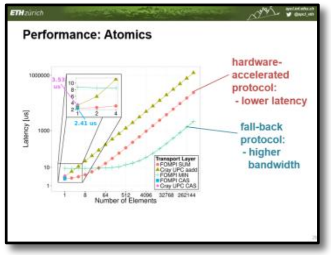


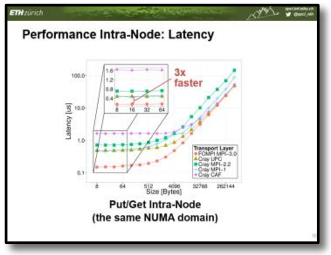

Performance Inter-Node: Latency
120% faster
20% faster
Put Inter-Node
Get Inter-Node
Close to hardware limit on Cray Gemini (1 µs)

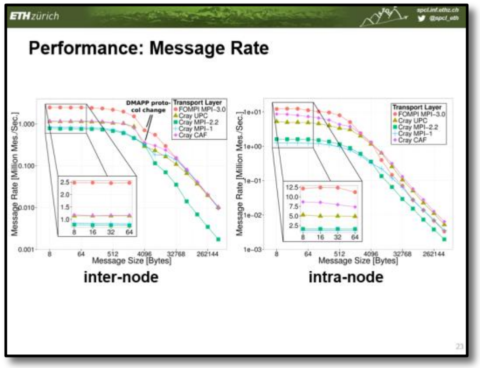

Scalable Fence Performance
90% faster
Asymptotic Time Bound $\mathcal{O}(\log p)$
Asymptotic Memory Bound $\mathcal{O}(1)$

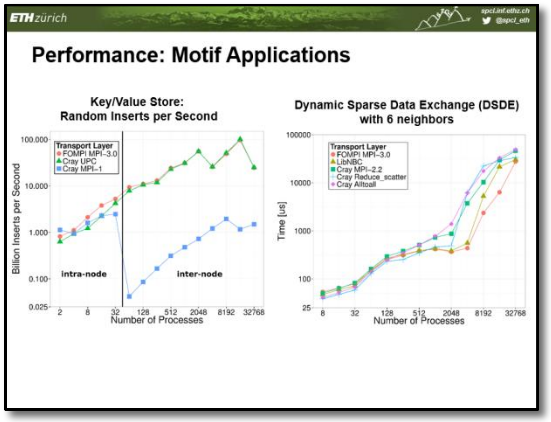
Thank you
for your attention


PSCW Performance
Asymptotic Time Bound
$\mathcal{P}_{start} = \mathcal{P}_{wait} = \mathcal{O}(1)$
$\mathcal{P}_{post} = \mathcal{P}_{complete} = \mathcal{O}(\log p)$
Asymptotic Memory Bound
$\mathcal{O}(\log p)$ (for scalable programs)
Ring Topology








Performance: Motif Applications
Key/Value Store: Random Inserts per Second
Dynamic Sparse Data Exchange (DSDE) with 6 neighbors
intra-node
inter-node