TORSTEN HOEFLER

Remote Memory Access Programming: Faster Parallel Computing Without Messages

with S. Ramos, R. Gerstenberger, M. Besta, R. Belli @ SPCL

presented at Tsinghua University, Beijing, China, November 2015

Founded 1855
- to support the industrialization of Switzerland

Top-20 general,
Top-10 Computer Science

18k students,
466 professors,
1.5 B budget

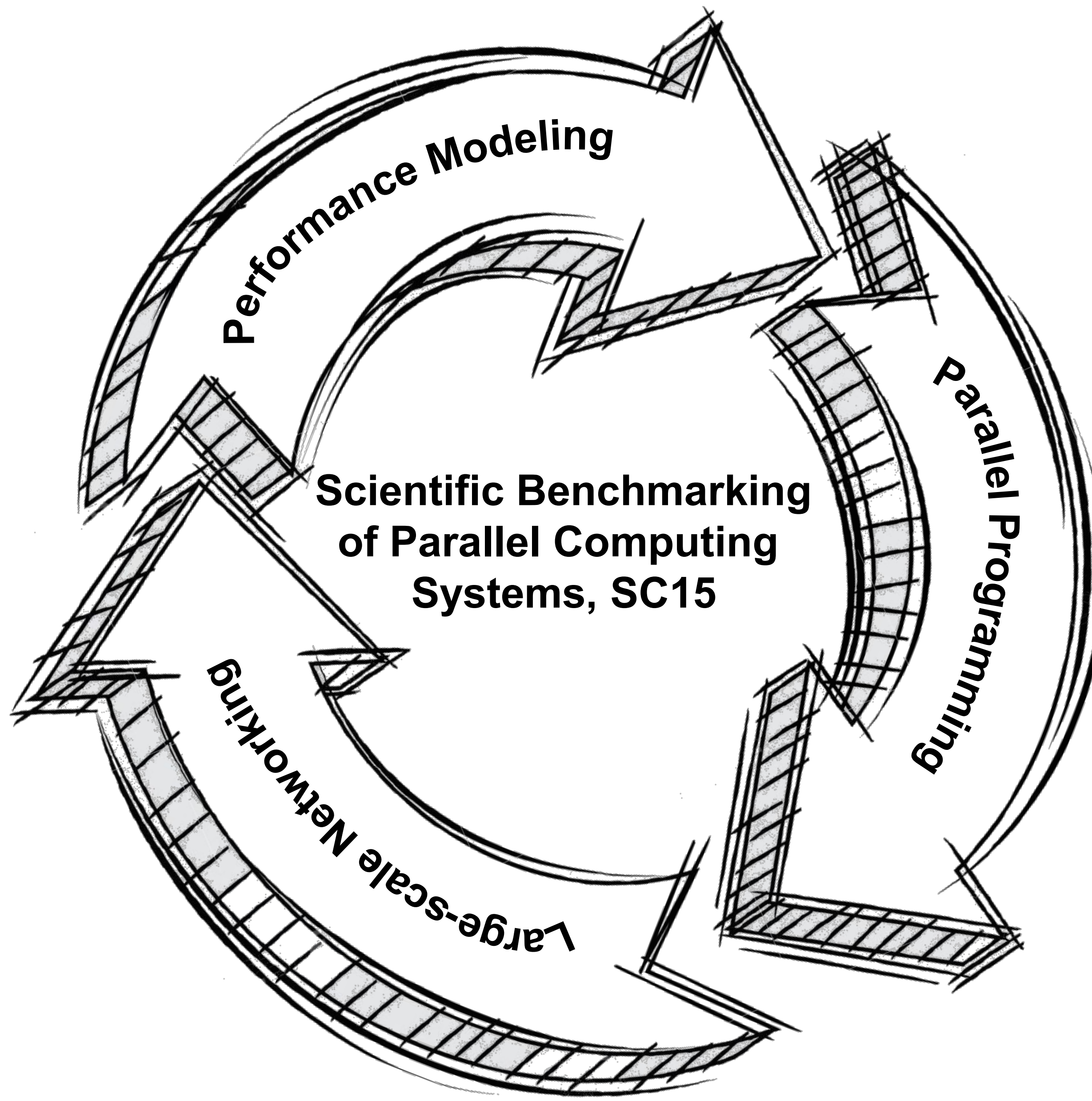
HPC at CSCS
- part of ETH

Piz Daint
5272 compute nodes
Xeon E5 + K20X
7.78 PF peak

Swiss HPC Community
- PASC

PASC16
Platform for Advanced Scientific Computing Conference
Lausanne Switzerland | 08-10 June 2016

CLIMATE & WEATHER
SOLID EARTH
LIFE SCIENCE
CHEMISTRY & MATERIALS
PHYSICS
COMPUTER SCIENCE & MATHEMATICS
ENGINEERING
EMERGING DOMAINS


Performance Modeling
Parallel Programming
Large-scale Networking
Scientific Benchmarking of Parallel Computing Systems, SC15

Motivation & Goals

- **My dream: provably optimal performance (time and energy)**
  - 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 → prove
  4. Implement, test, refine if necessary

- **Will demonstrate techniques & insights**
  - And obstacles ☺
  - RMA as a solution?

Example: Message Passing, Log(G)P

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

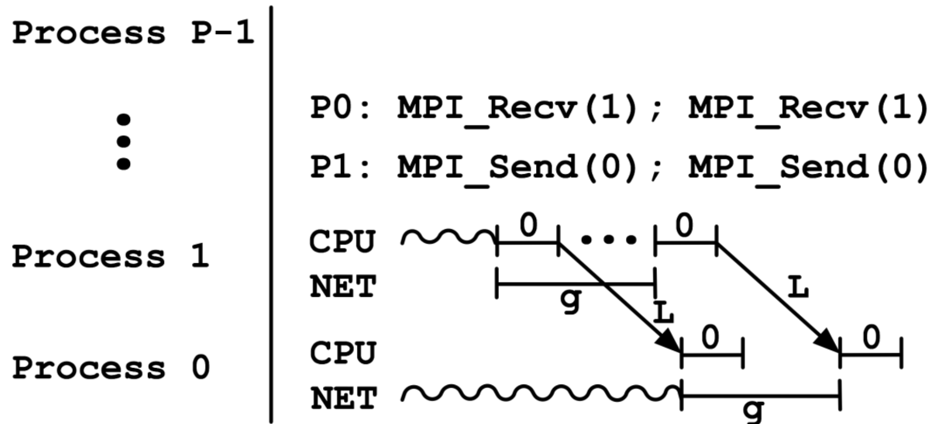
LogP

A Practical Model of Parallel Computation

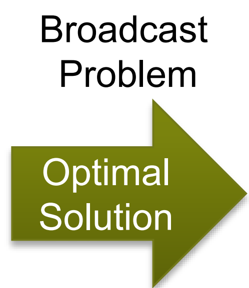
David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauser, Ramesh Subramonian, and Thorsten von Eicken

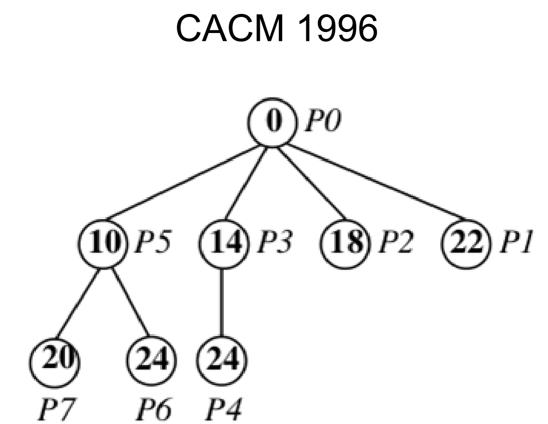
UR GOAL IS TO DEVELOP A MODEL OF PARALLEL COMPUTATION THAT WILL serve as a basis for the design and analysis of fast, portable parallel algorithms, such as algorithms that can be implemented effectively on a wide variety of current and future parallel machines. If we look at the body of parallel algorithms 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

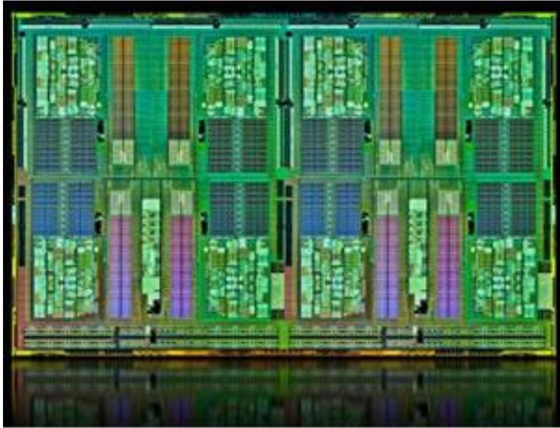


Broadcast Problem

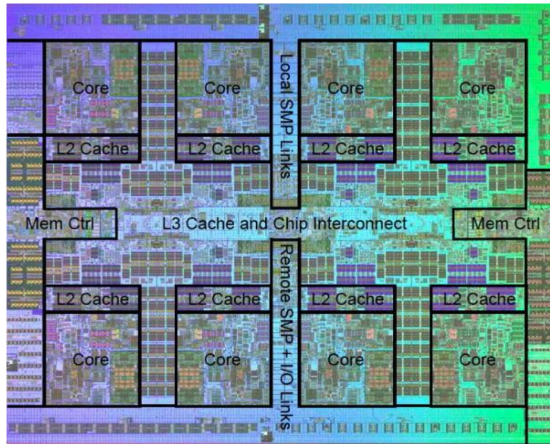




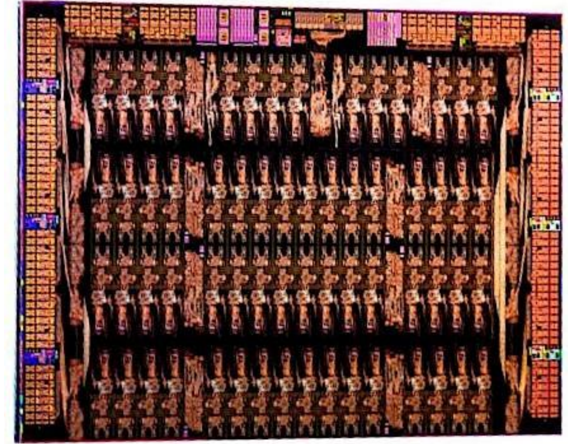
Hardware Reality



Interlagos, 8/16 cores, source: AMD

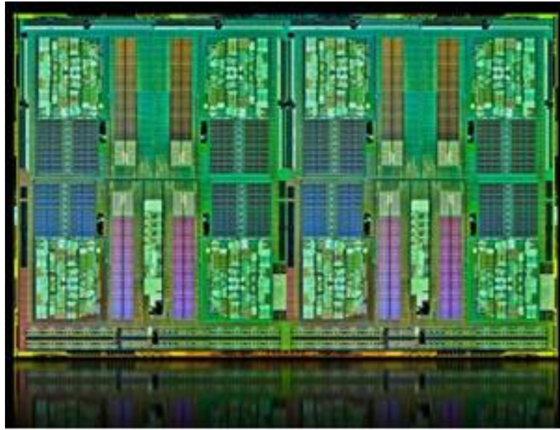


POWER 7, 8 cores, source: IBM

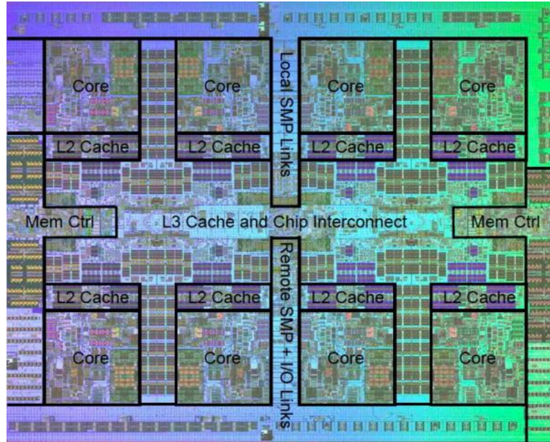


Xeon Phi, 64 cores, source: Intel

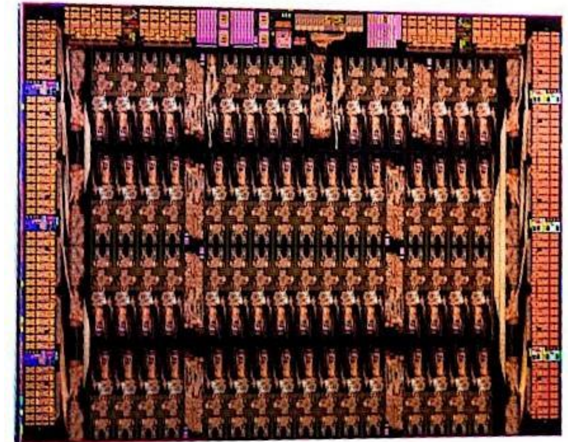
Hardware Reality



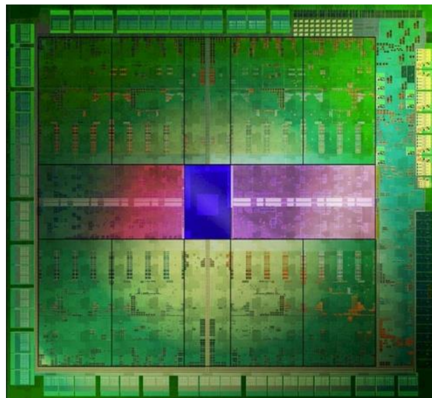
Interlagos, 8/16 cores, source: AMD



POWER 7, 8 cores, source: IBM

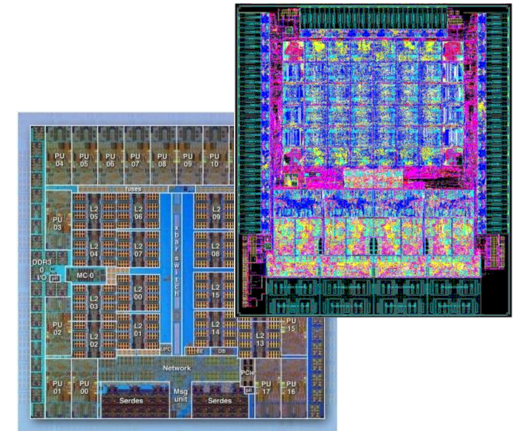


Xeon Phi, 64 cores, source: Intel



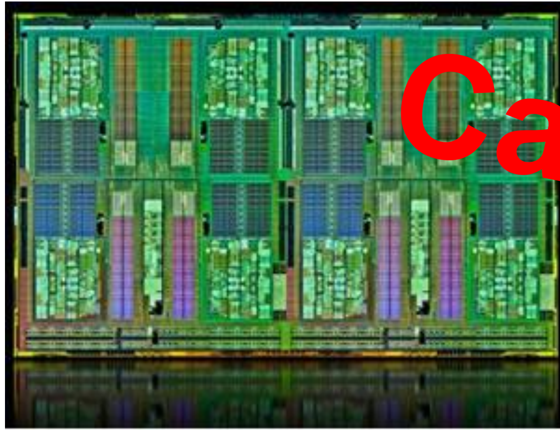
Kepler GPU, source: NVIDIA

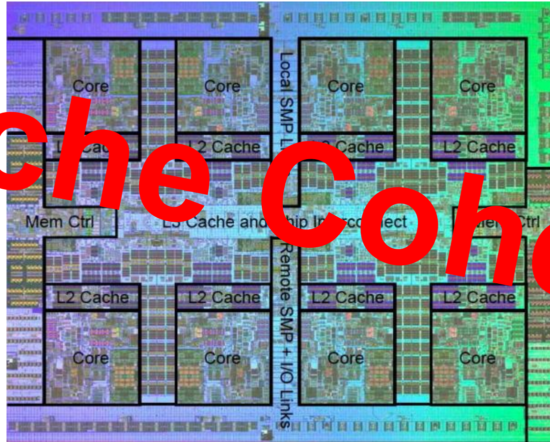
InfiniBand, sources: Intel, Mellanox

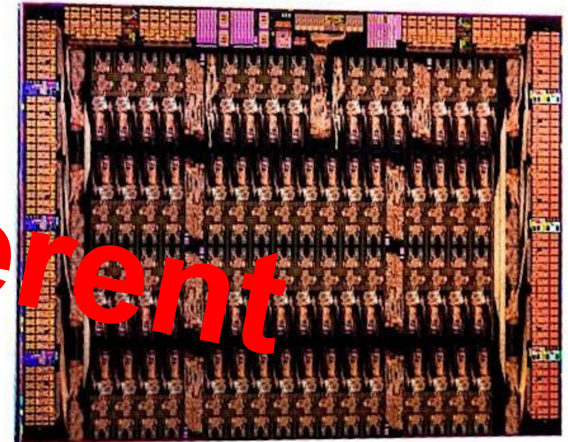


BG/Q, Cray Aries, sources: IBM, Cray

Hardware Reality





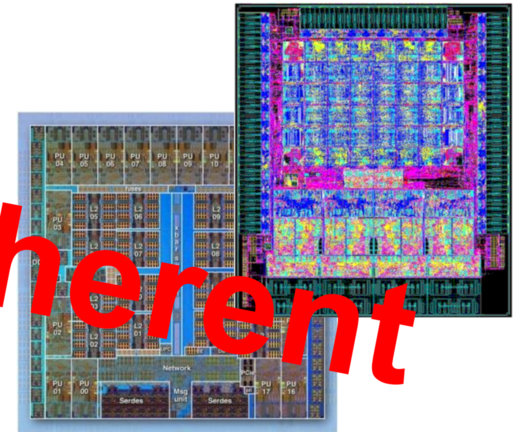


Cache Coherent

Interlagos, 8/16 cores, source: AMD

POWER 7, 8 cores, source: IBM

Xeon Phi, 64 cores, source: Intel



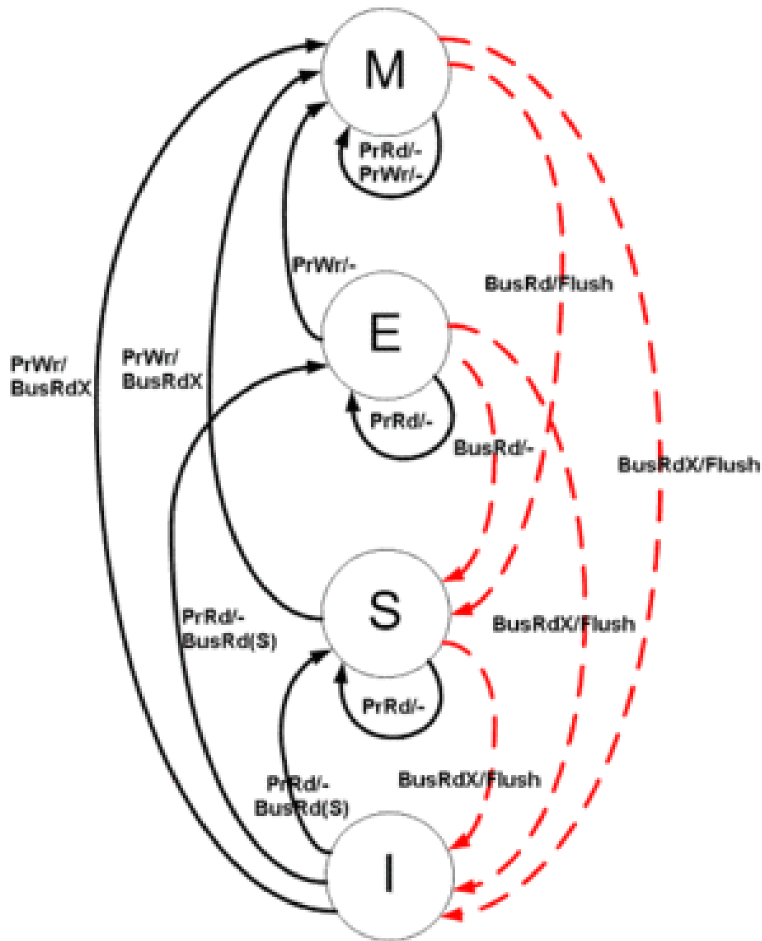
Not Cache Coherent

Kepler GPU, source: NVIDIA

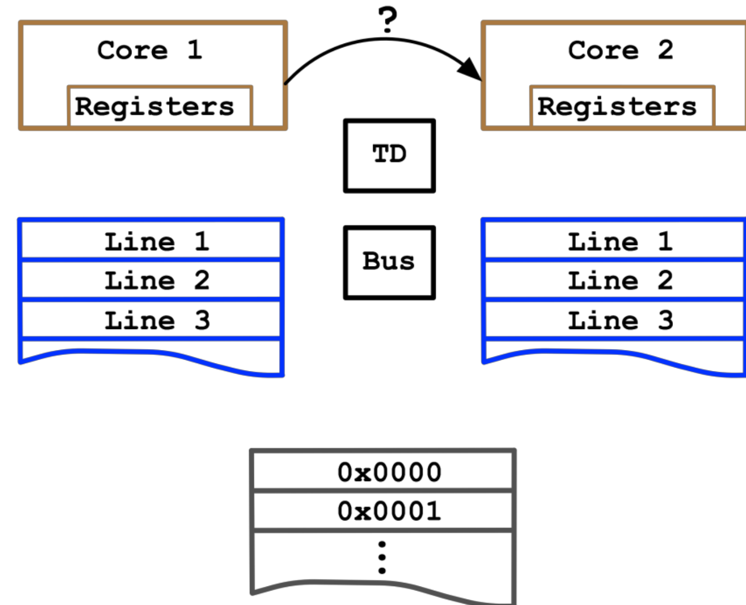
InfiniBand, sources: Intel, Mellanox

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

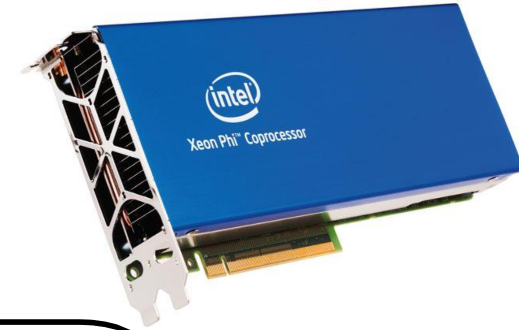
Example: Cache-Coherent Communication



Source: Wikipedia



Xeon Phi (Rough) Architecture

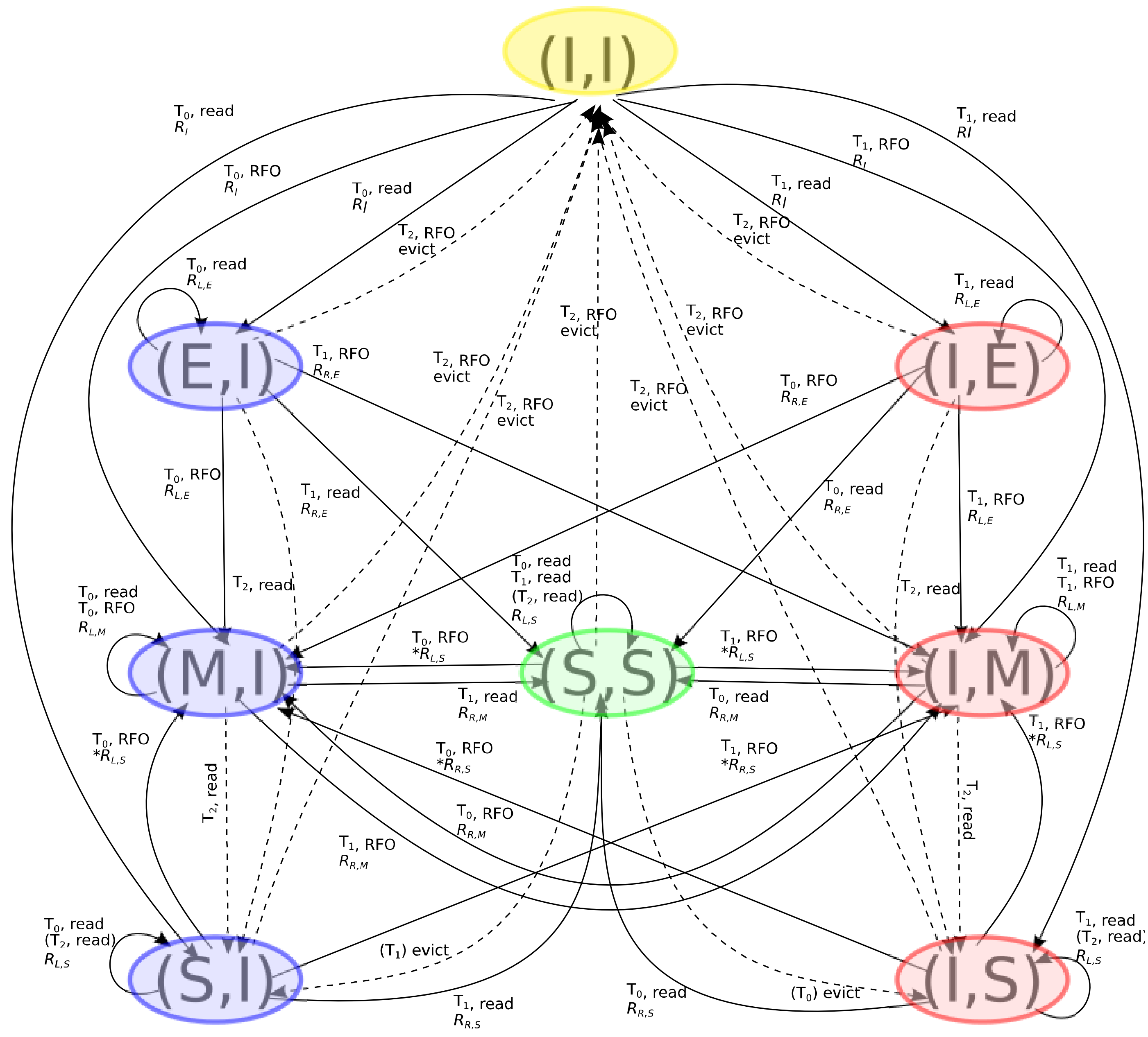


CORE CORE ••• CORE CORE

GDDR5 TD TD ••• TD TD GDDR5

GDDR5 TD TD ••• TD TD GDDR5

CORE CORE ••• CORE CORE



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

(I,I)

T_0, RFO
T_0, read
R_I

T_1, RFO
T_1, read
R_I

T_0, RFO
T_0, read
(T_2, read)
$R_{L,*}$

T_1, RFO
T_1, read
(T_2, read)
$R_{L,*}$

T_2, RFO
evict

T_2, RFO
evict

(*,I)

(I,*)

T_1, RFO
$R_{R,*}$

T_0, RFO
$R_{R,*}$

T_1, read
$R_{R,*}$

T_0, read
$R_{R,*}$

T_0, RFO
(T_1 evict)
$R_{L,*}$

T_2, RFO
evict

T_1, RFO
(T_0 evict)
$R_{L,*}$

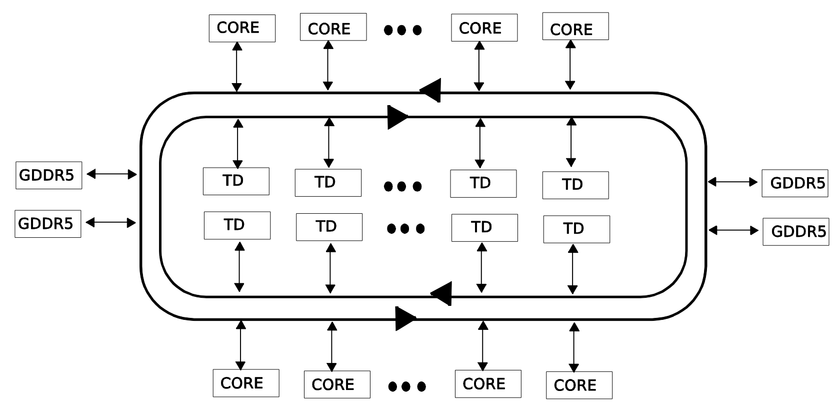
(*,*)

T_0, read
T_1, read
(T_2, read)
$R_{L,*}$

Invalid read R_I = 278 ns

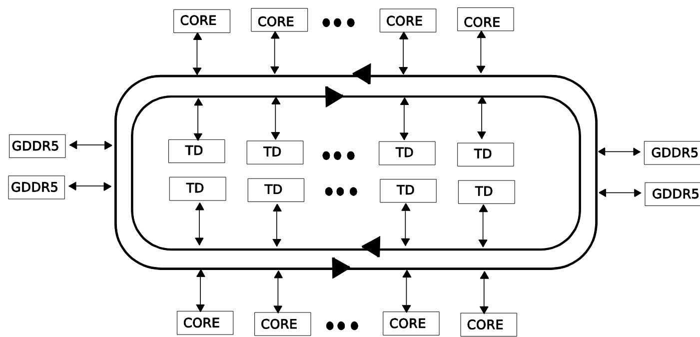
Local read: R_L = 8.6 ns

Remote read R_R = 235 ns



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

DTD Contention ☹

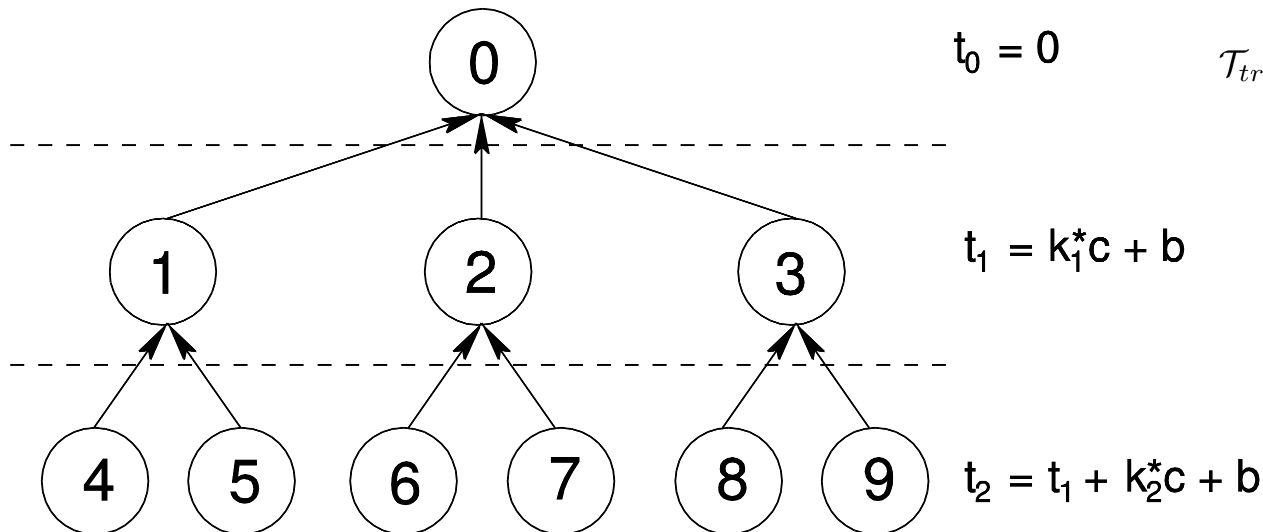


- **E state:**
  - a=0ns
  - b=320ns
  - c=56.2ns

$$\mathcal{T}_C(n_{th}) = c \cdot n_{th} + b - \frac{a}{n_{th}}$$

Designing Broadcast Algorithms

- **Assume single cache line → forms a Tree**
  - We choose d levels and k_j children in level j
  - Reachable threads: $n_{th} \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j$
  - Example: d=2, k_1=3, k_2=2:



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

c = DTD contention
b = transmit latency

Finding the Optimal Broadcast Algorithm

- **Broadcast example:**

Bcast cost

Number of levels

Reached threads

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

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

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

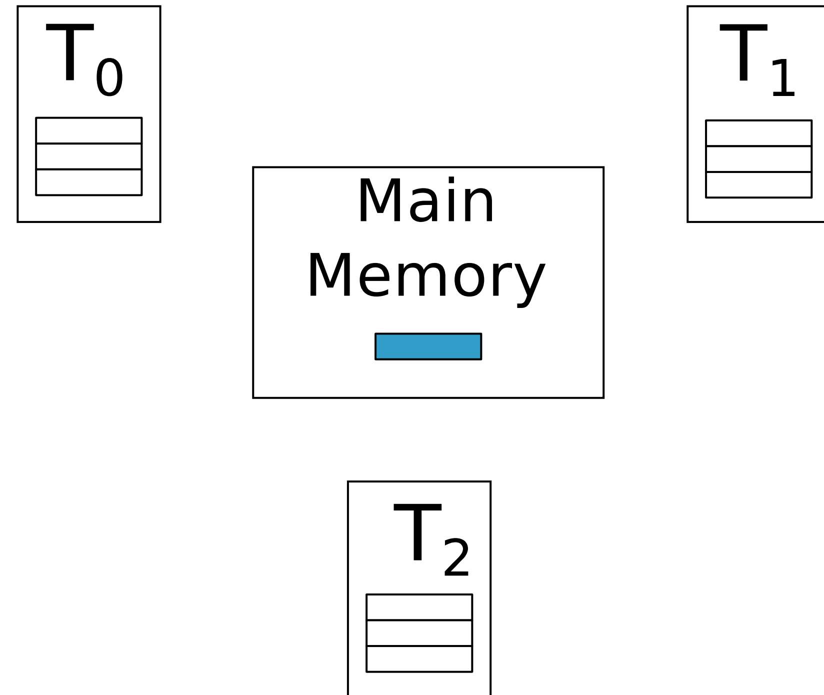
$$\mathcal{T}_{sbcast} = \min_{d,k_i} \left(\mathcal{T}_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} \mathcal{T}_{nb}(k_i + 1) \right)$$

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

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

Min-Max Modeling

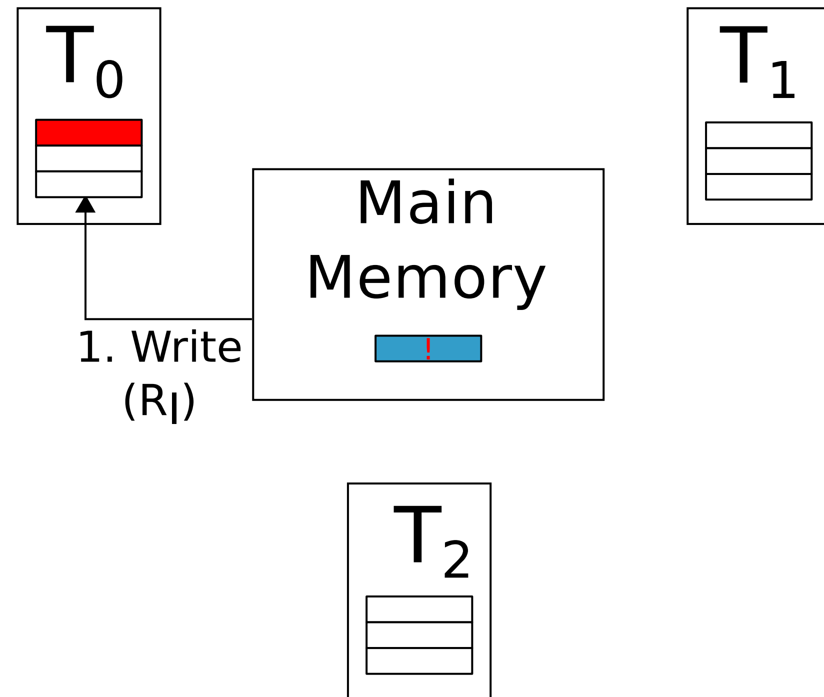
- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion



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

Min-Max Modeling

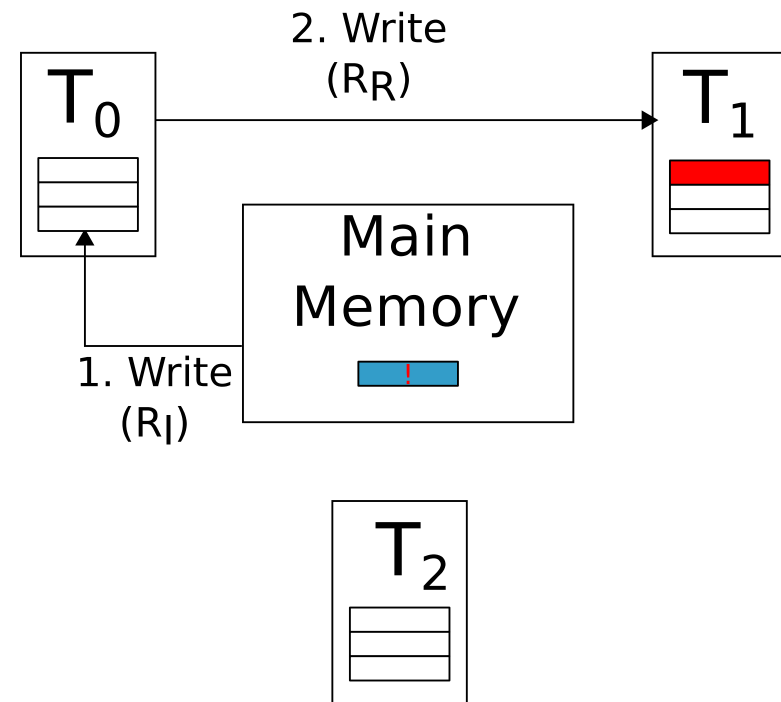
- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion



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

Min-Max Modeling

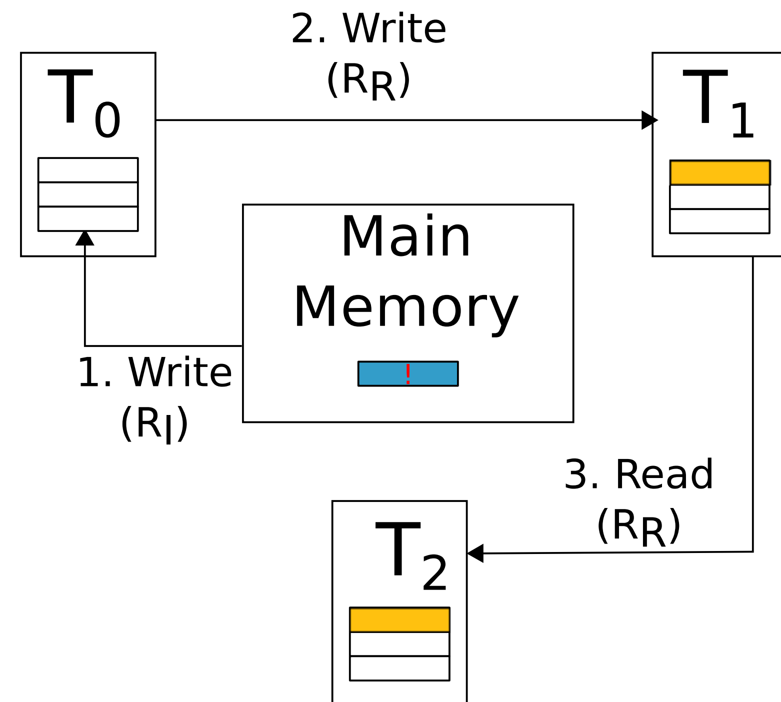
- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion



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

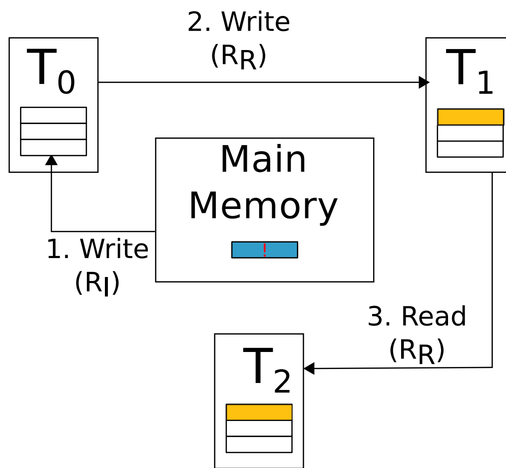
Min-Max Modeling

- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion



Min-Max Modeling

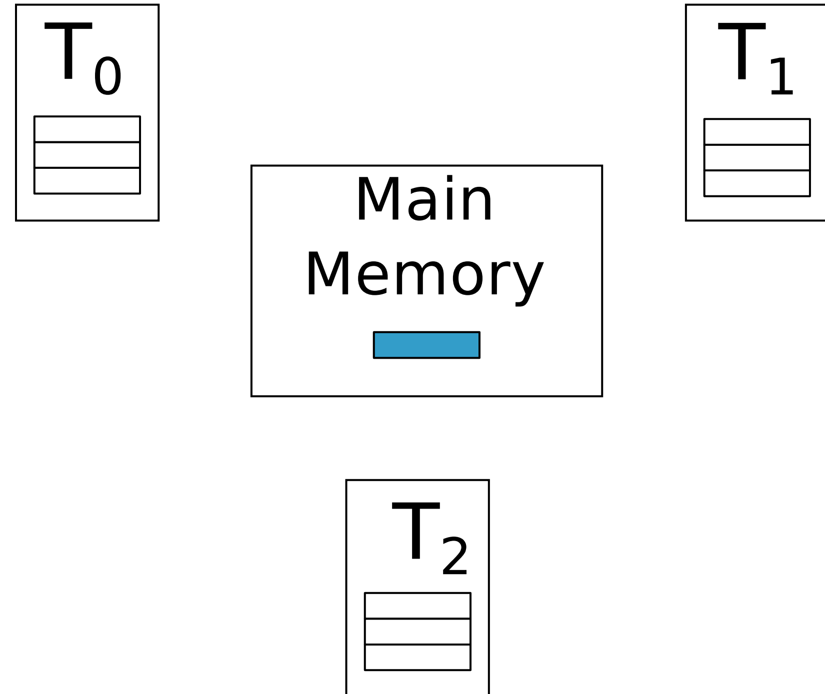
- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion

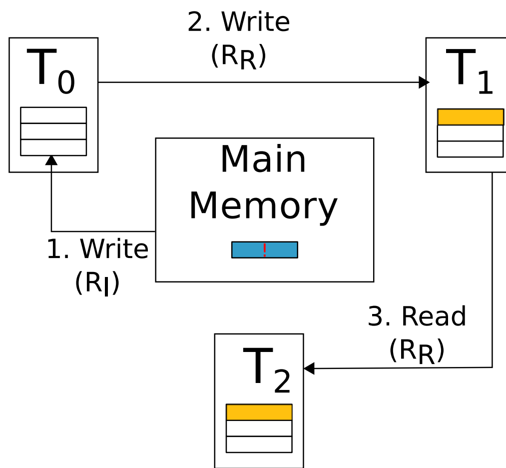


Ramos, Hoefler: “Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi “, HPDC’13

Min-Max Modeling

- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion

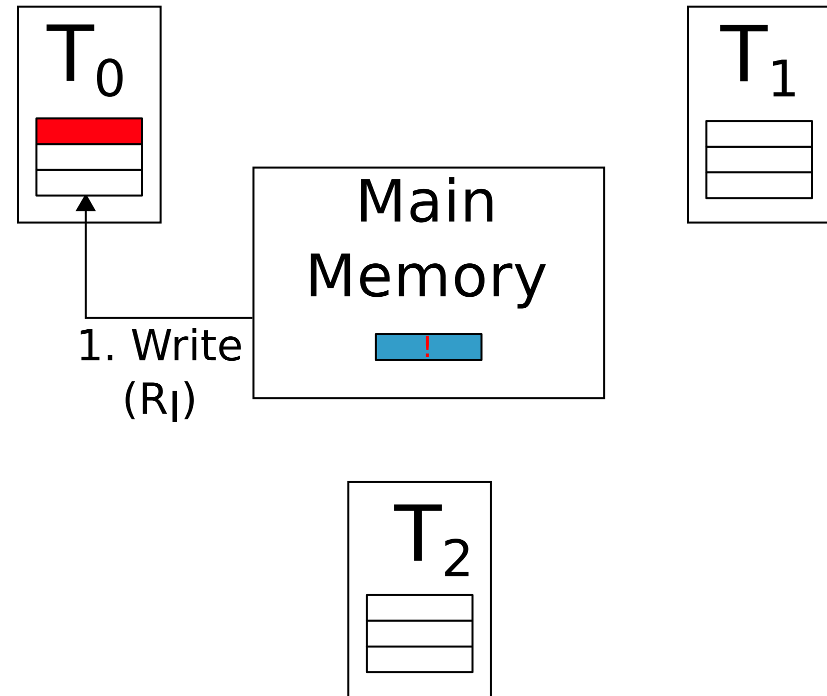


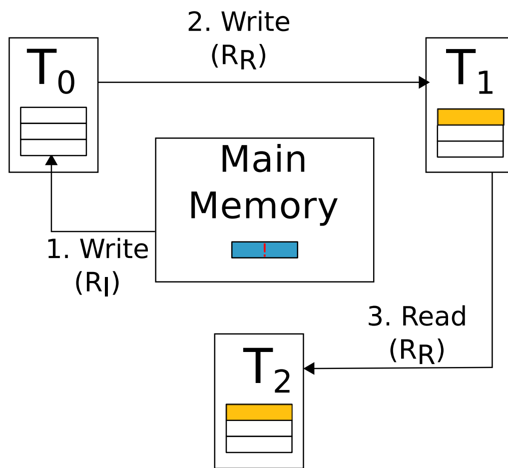


Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Min-Max Modeling

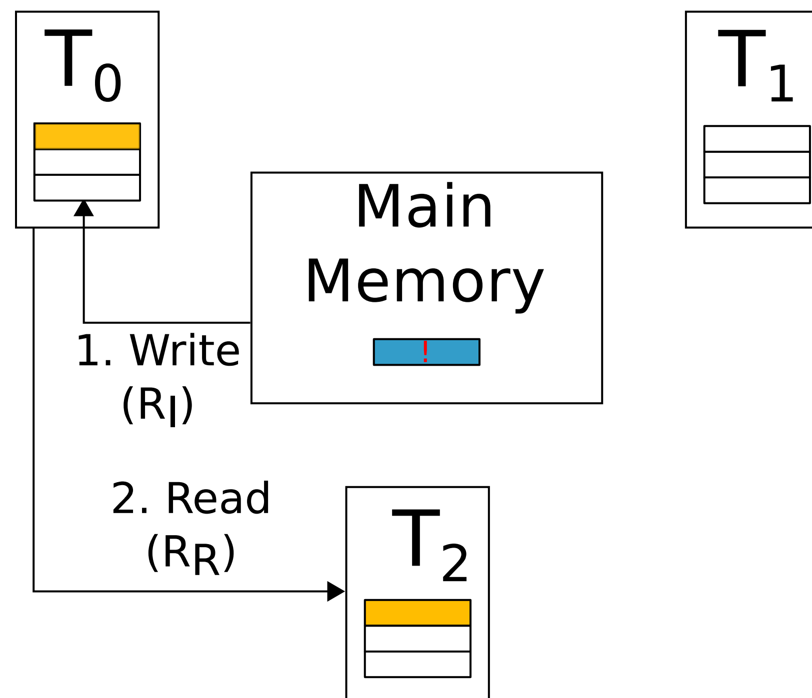
- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion

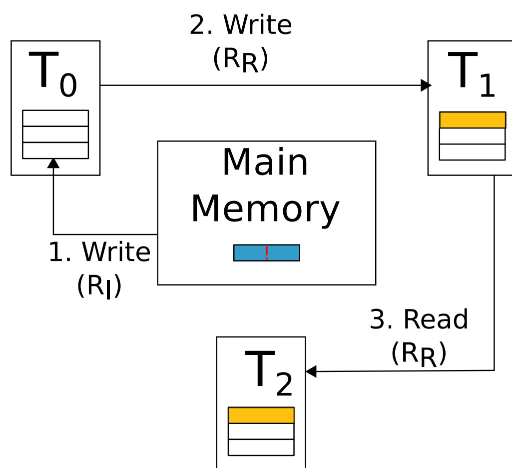




Min-Max Modeling

- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion

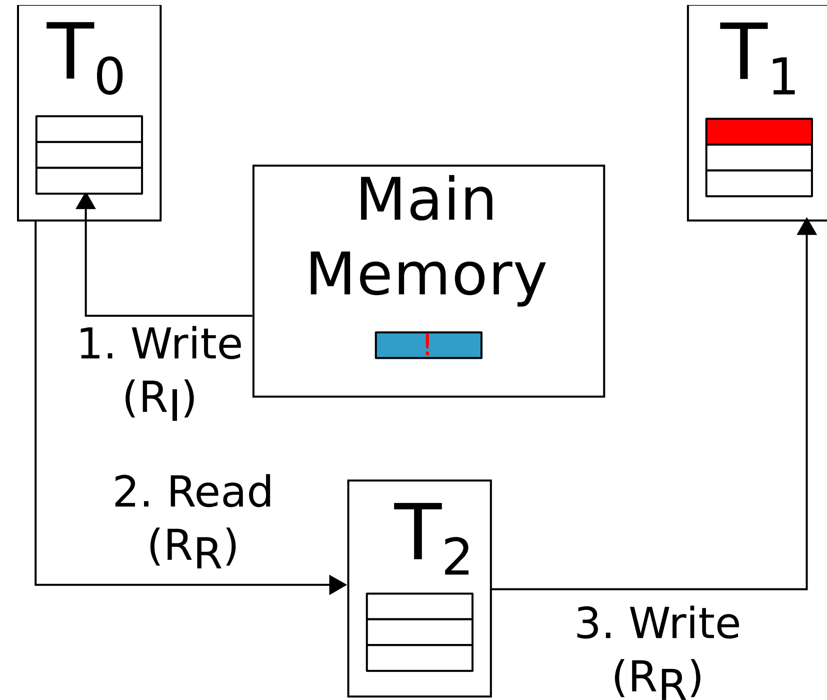


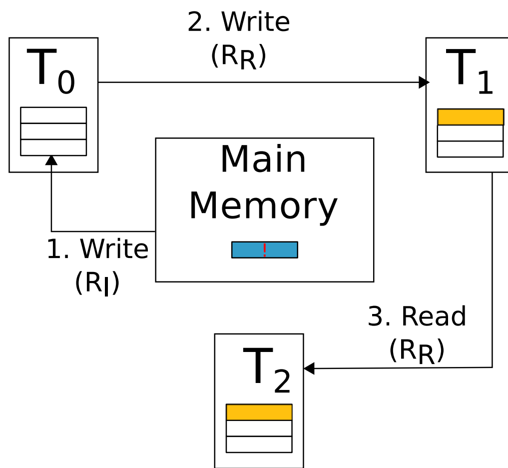


Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Min-Max Modeling

- **Example:**
  - $T_0 + T_1$ write CL
  - T_2 polls for completion

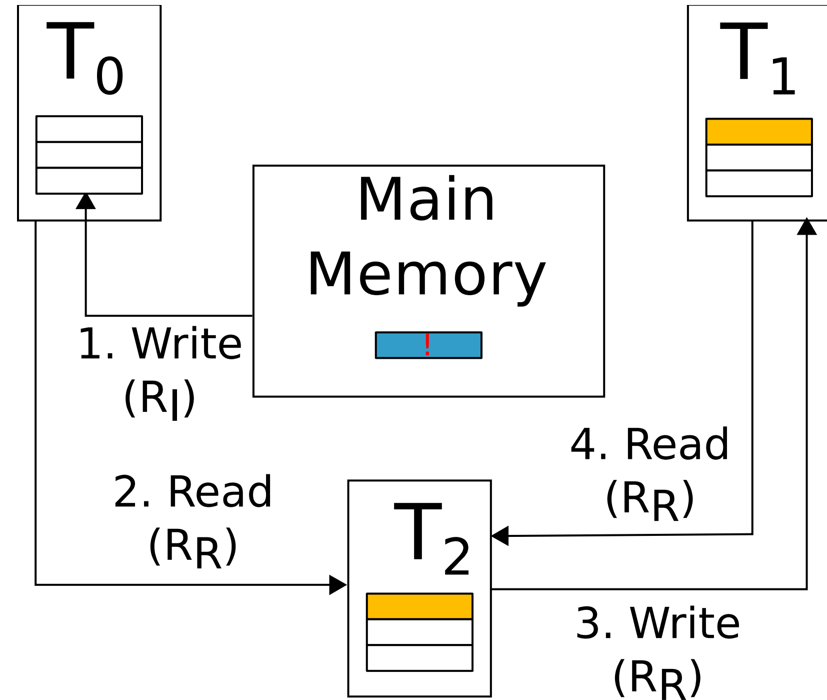


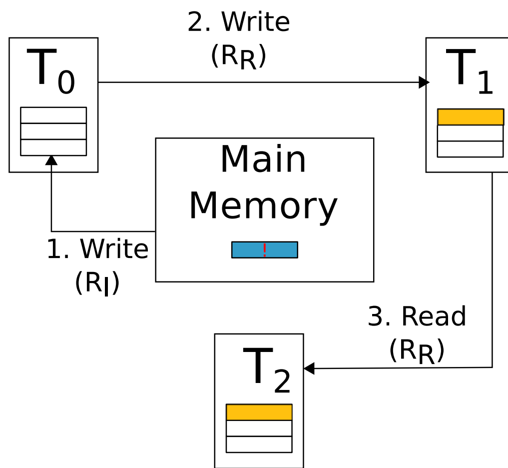


Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Min-Max Modeling

- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion

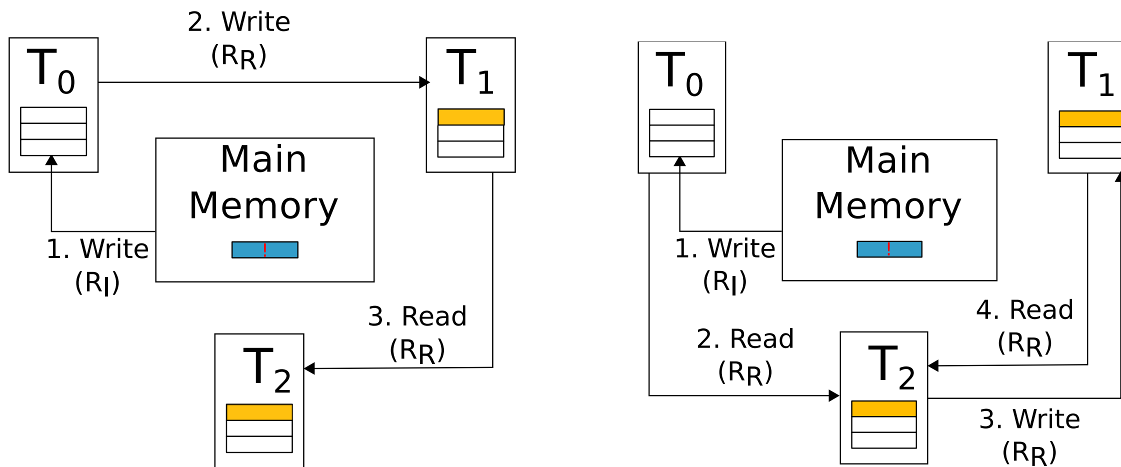




Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Min-Max Modeling

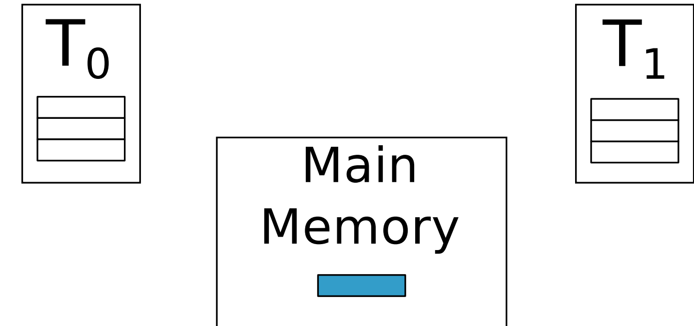
- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion

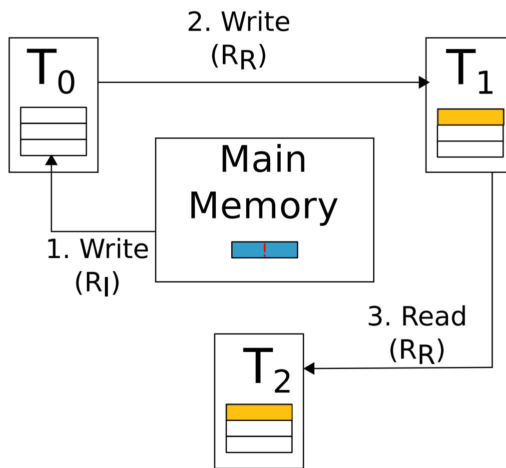


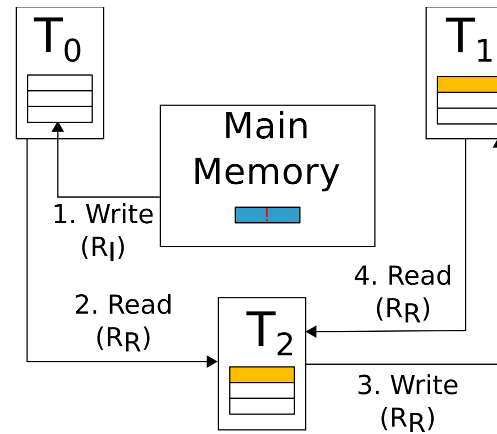
Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Min-Max Modeling

- **Example:**
  - $T_0 + T_1$ write CL
  - T_2 polls for completion

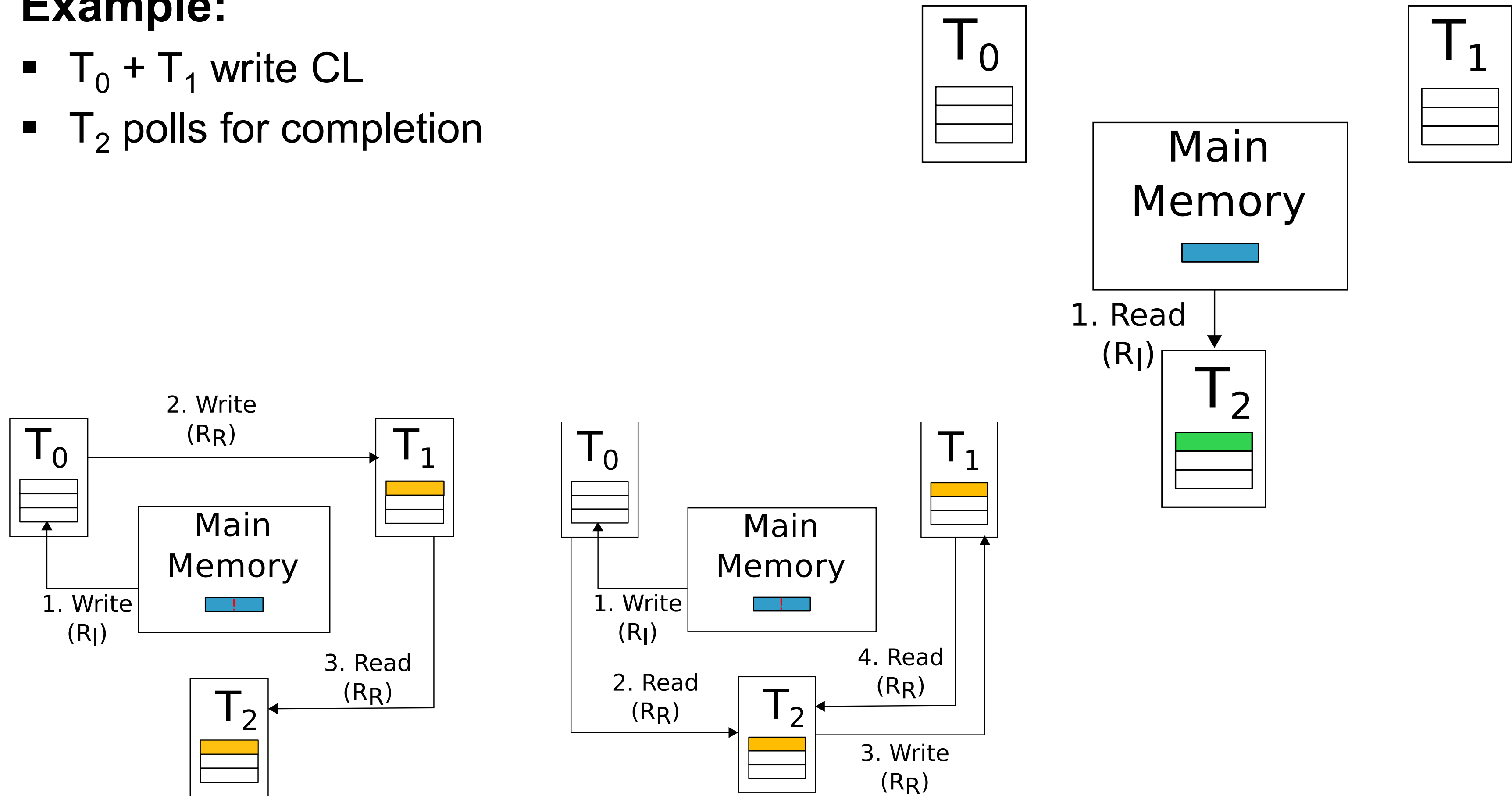






Min-Max Modeling

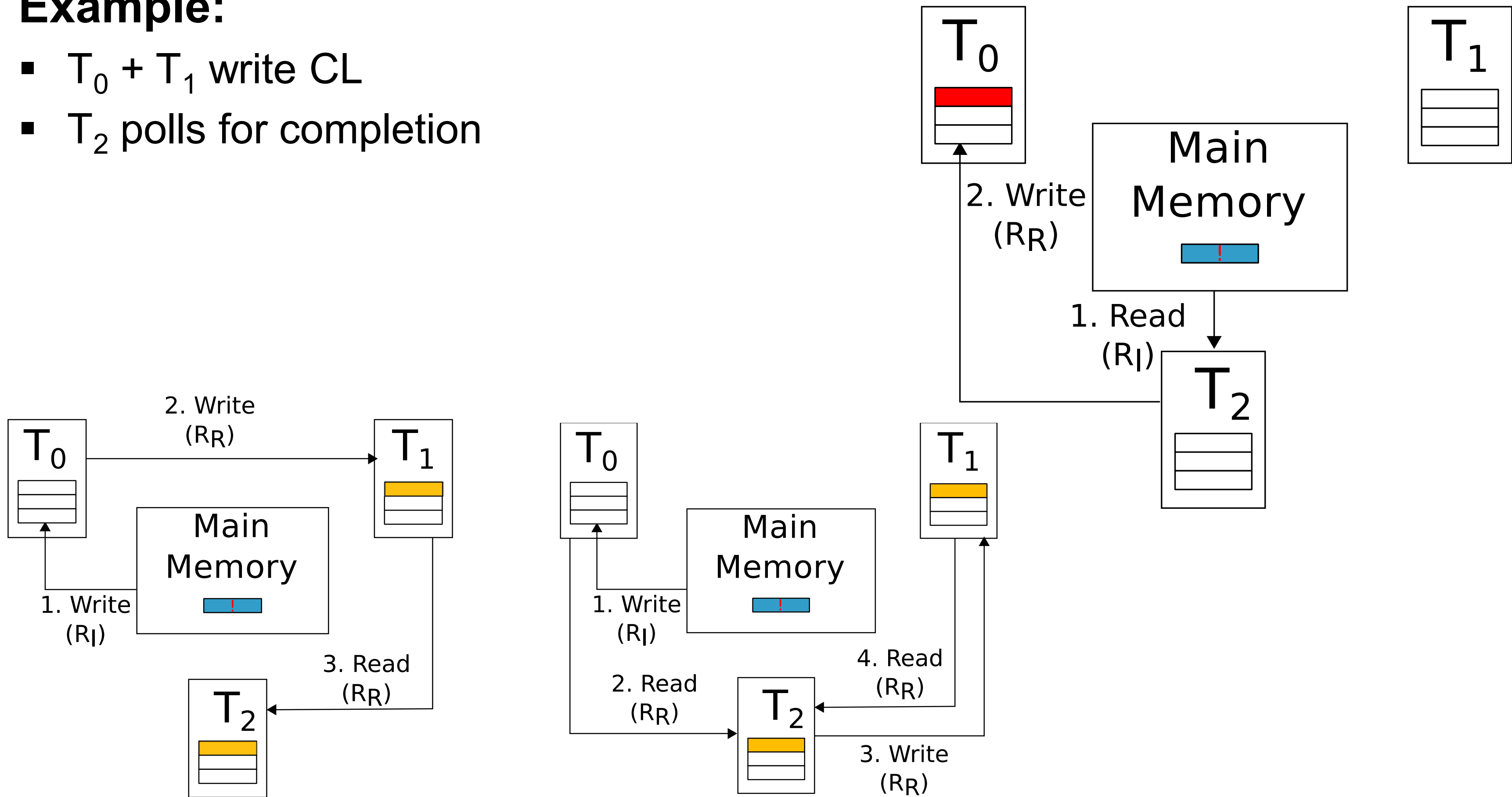
- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion



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

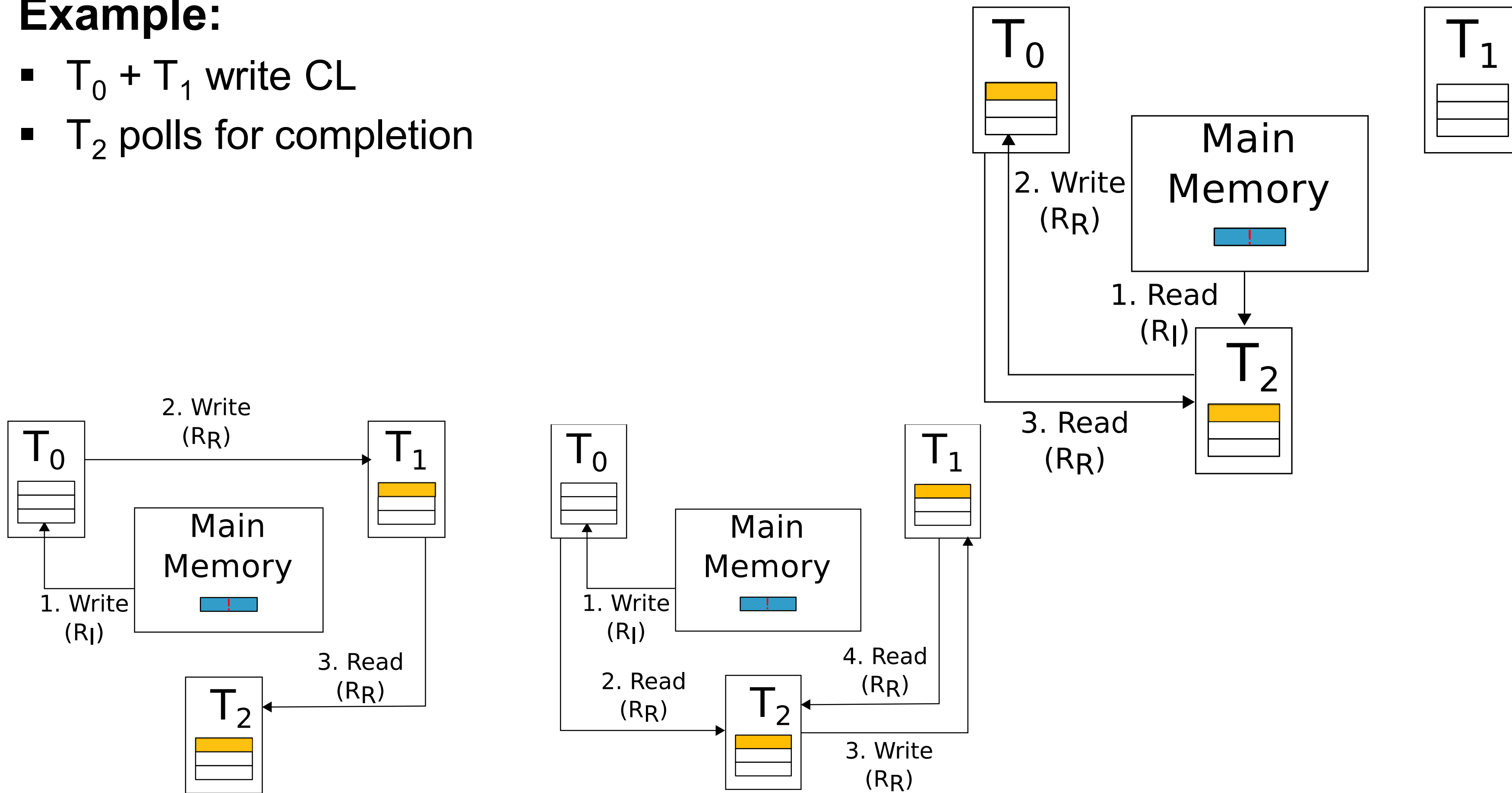
Min-Max Modeling

- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion



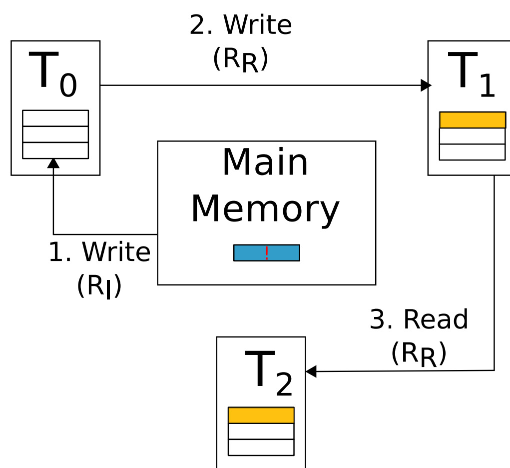
Min-Max Modeling

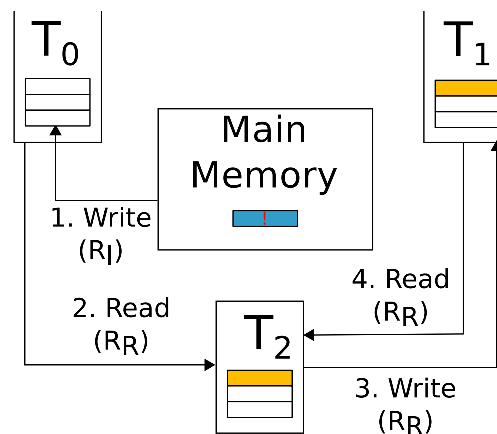
- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion

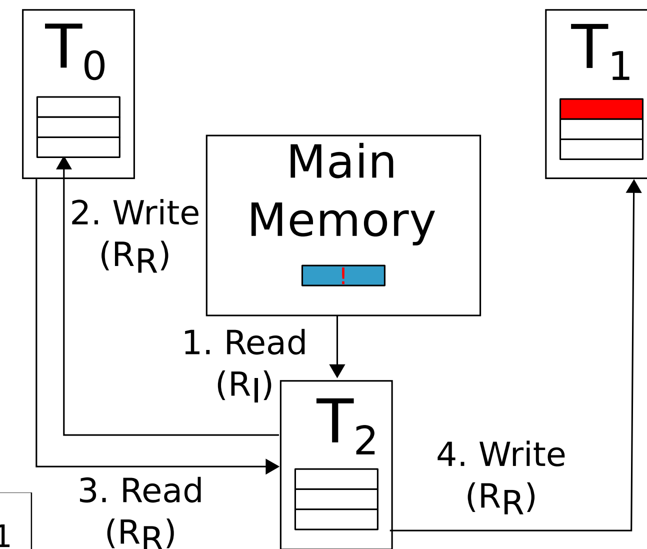


Min-Max Modeling

- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion



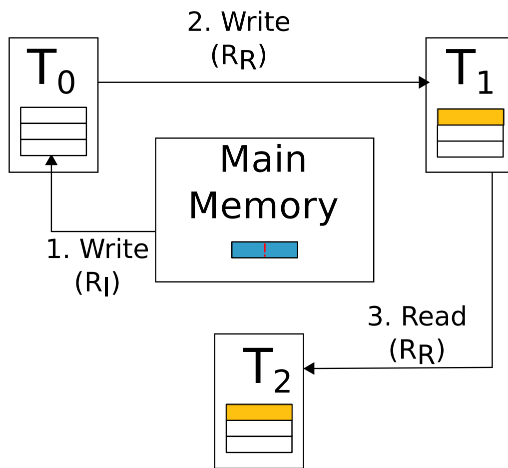


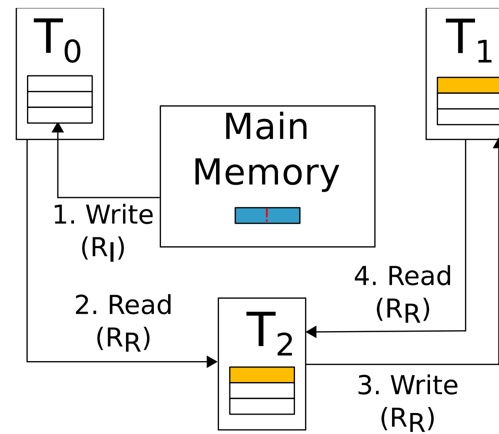


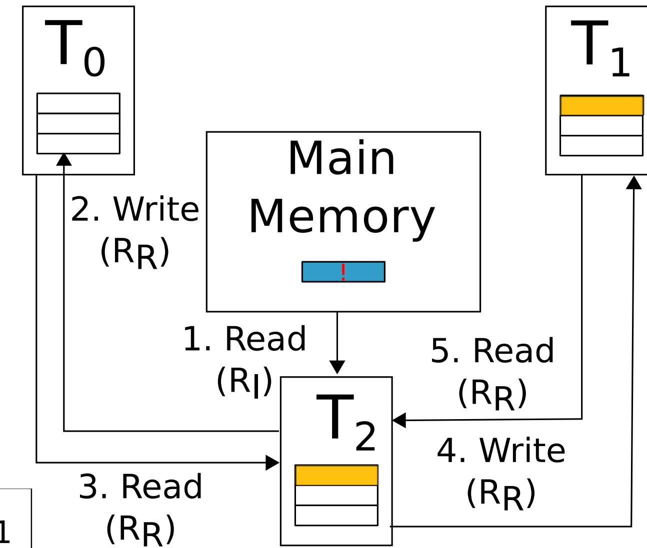
Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Min-Max Modeling

- **Example:**
  - $T_0 + T_1$ write CL
  - T_2 polls for completion



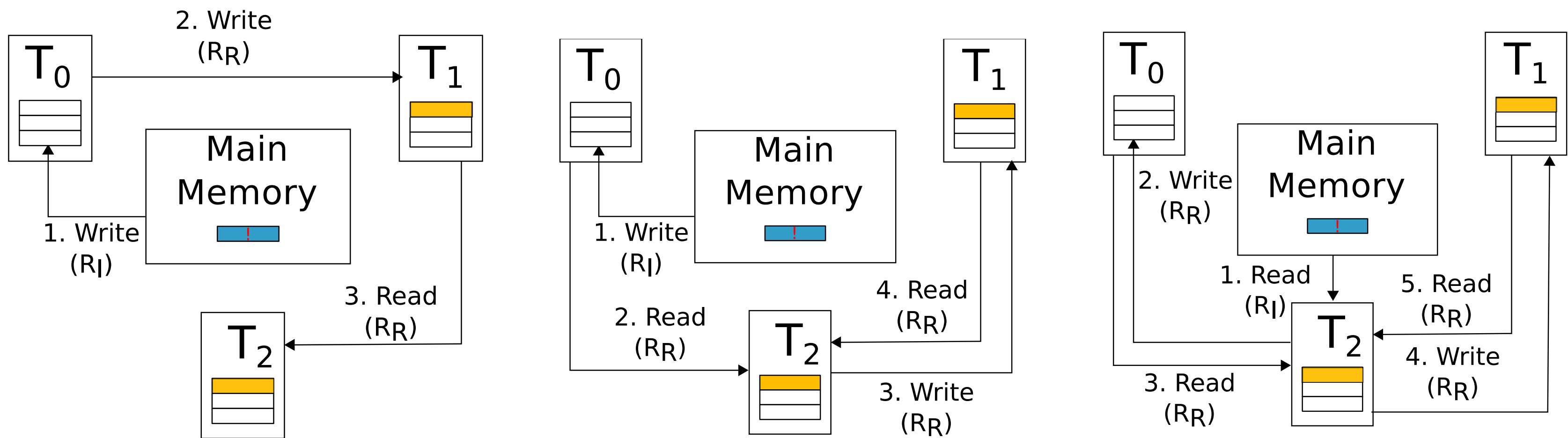




Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

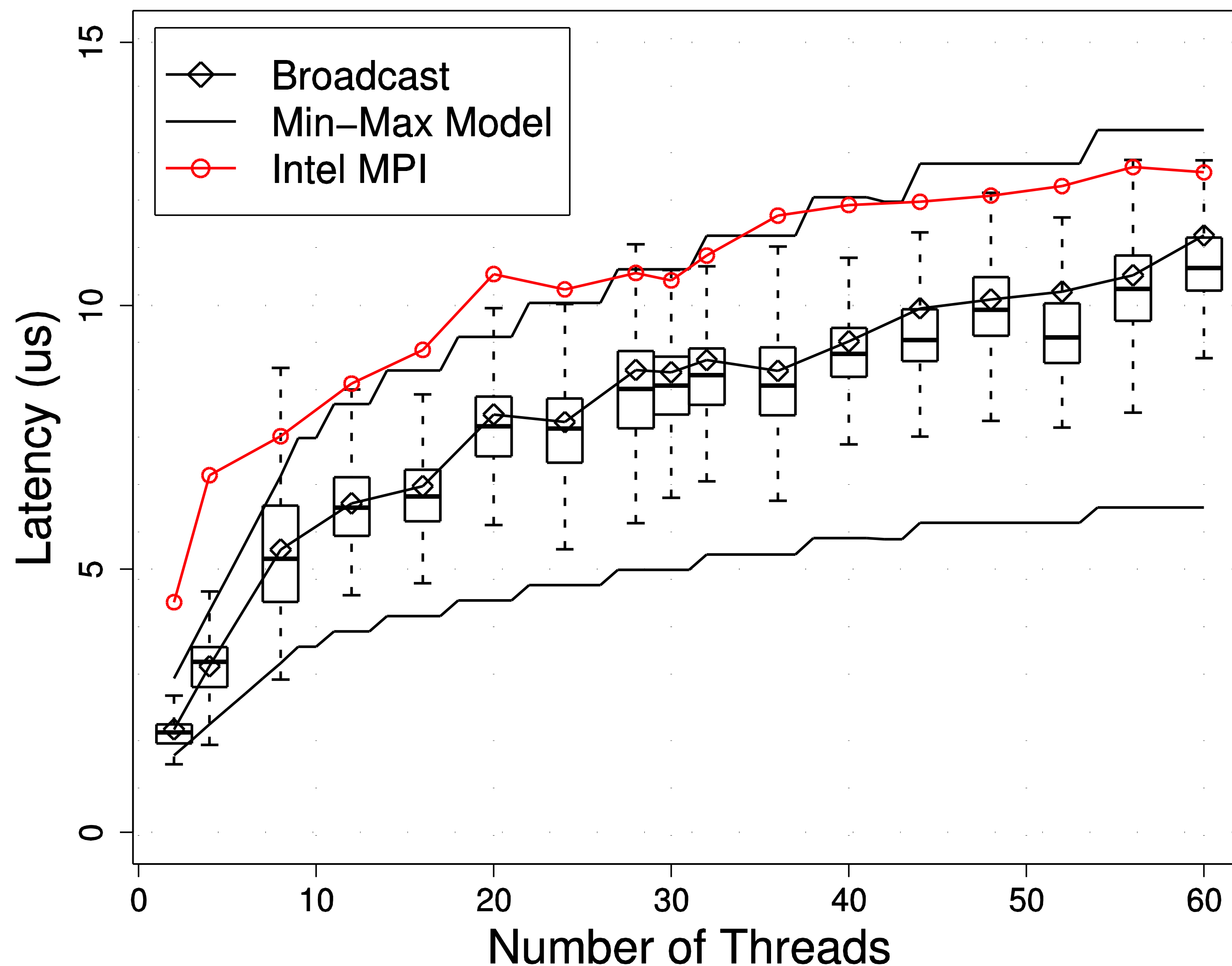
Min-Max Modeling

- **Example:**
  - T_0 + T_1 write CL
  - T_2 polls for completion



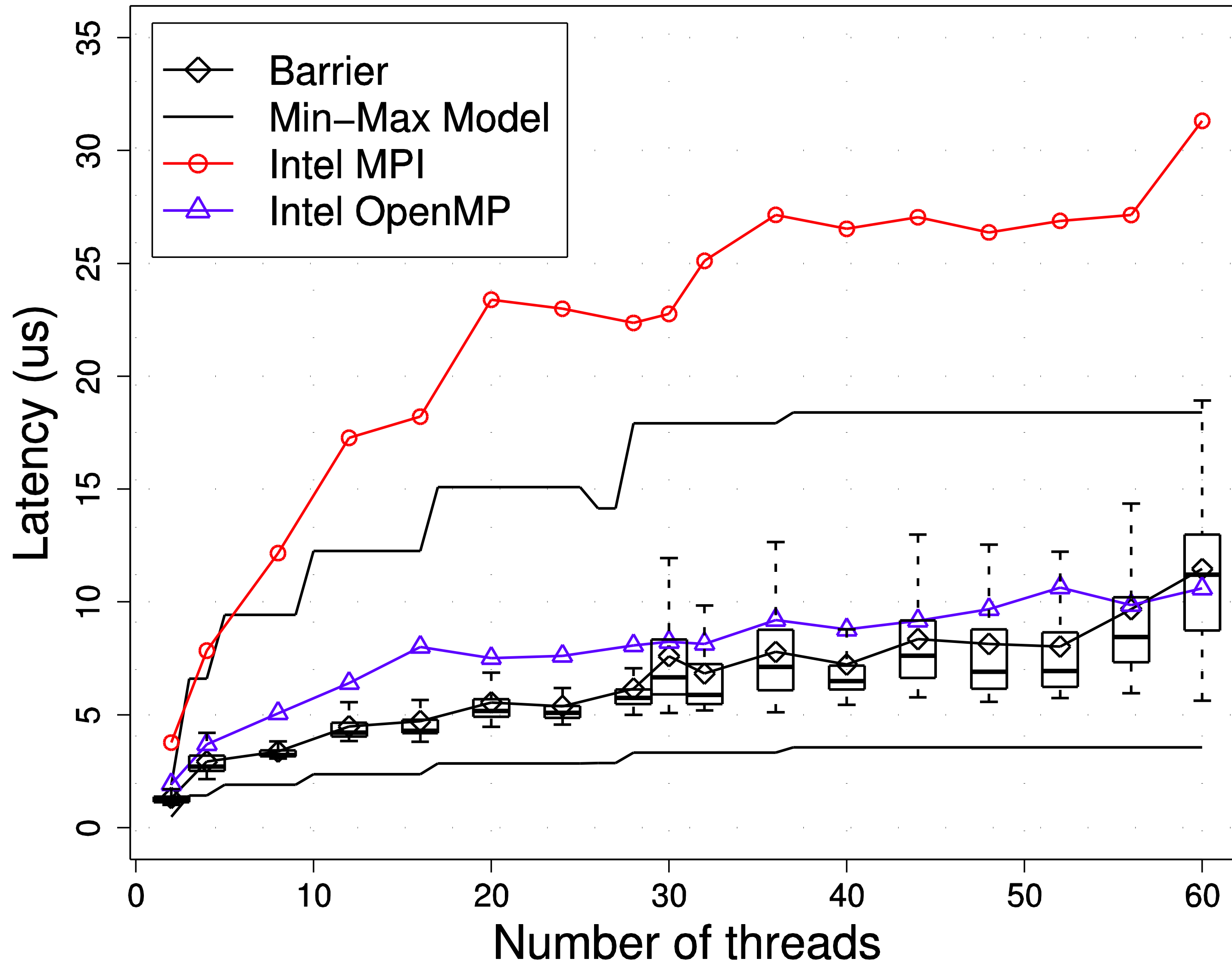
Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Small Broadcast (8 Bytes)



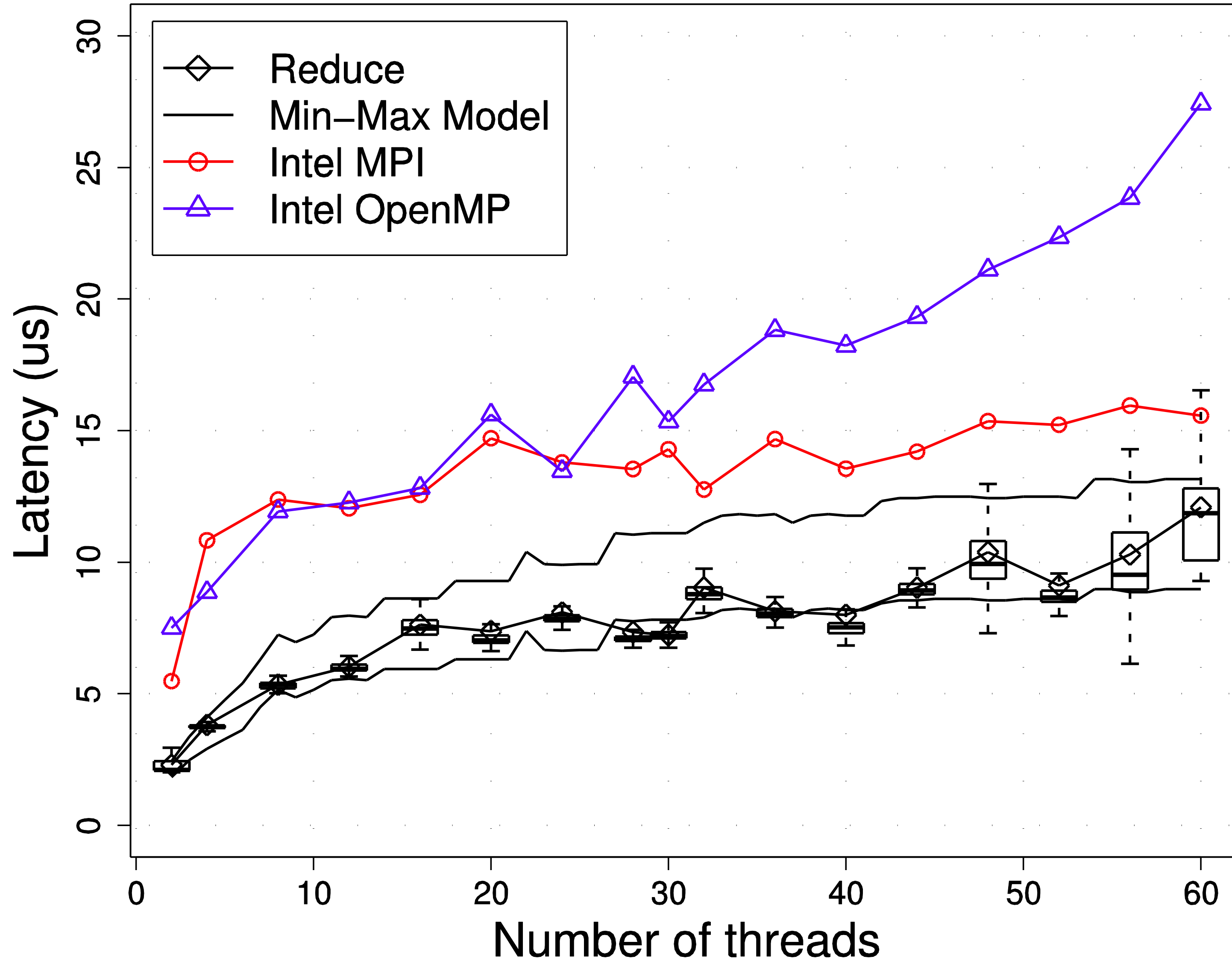
Ramos, Hoefler: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", HPDC'13

Barrier



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

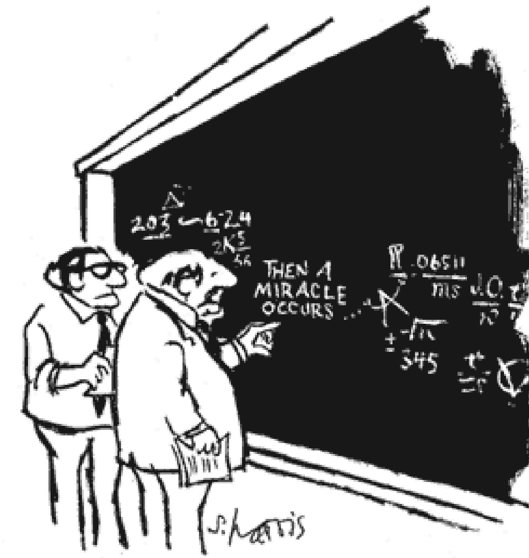
Small Reduction



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, 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**!



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

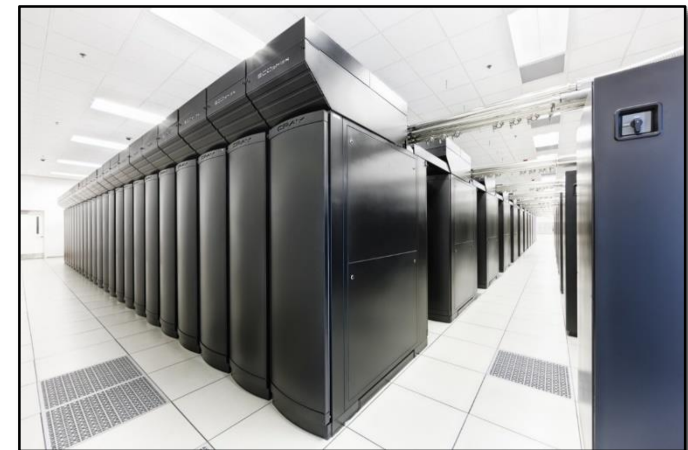
[1]: Calotoiu et al.: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes, SC13
[2]: Gerstenberger et al.: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13, Best Paper

Communication in Today's HPC Systems

- **The de-facto programming model: MPI-1**
  - Using send/recv messages and collectives

- **The de-facto network standard: RDMA, SHM**
  - Zero-copy, user-level, os-bypass, fuzz-bang

Random datacenter picture copyrighted by Reuters (yes, they go after academics with claims for 10 year old images)



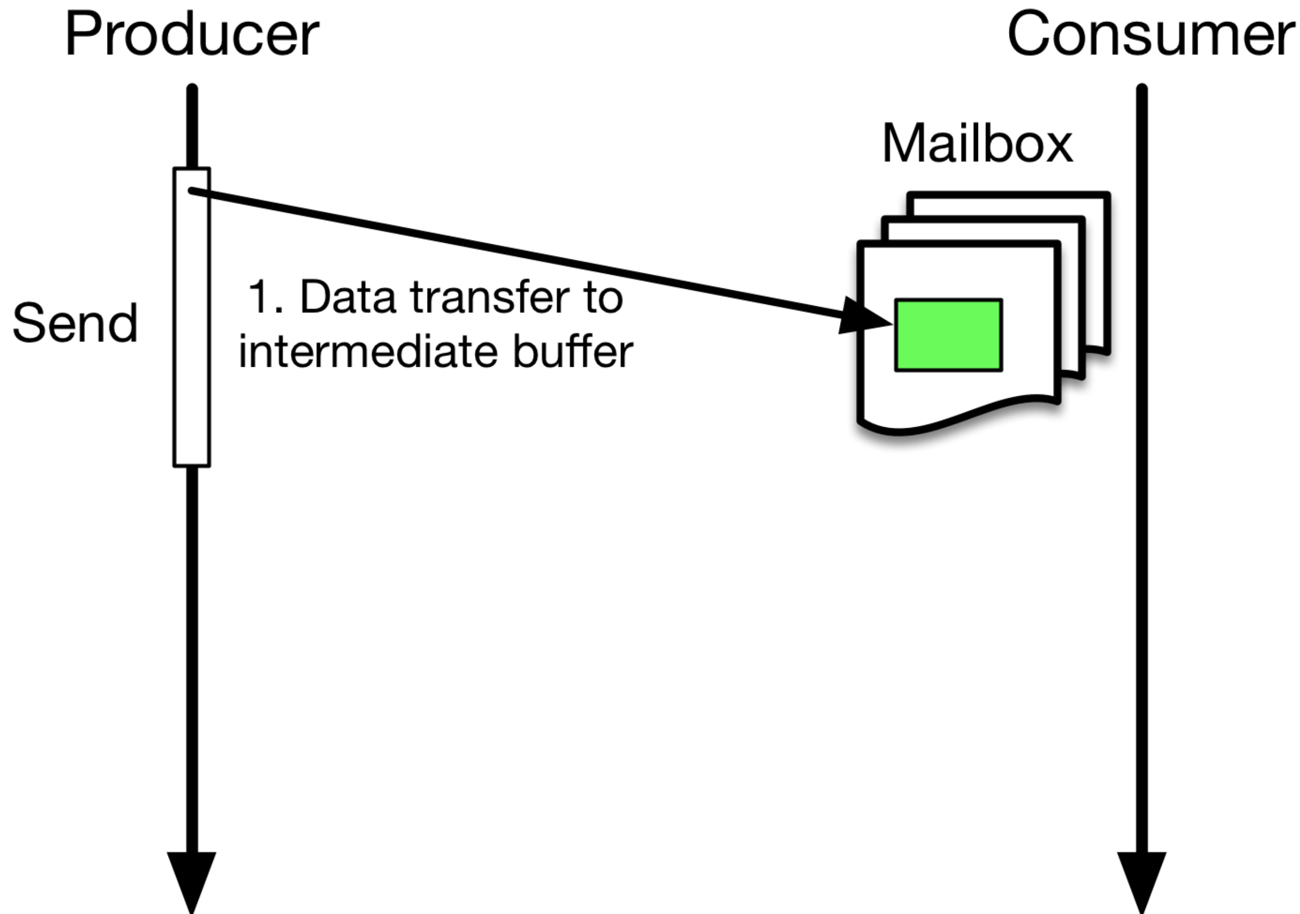
MPI-1 Message Passing – Simple Eager

Producer

Consumer

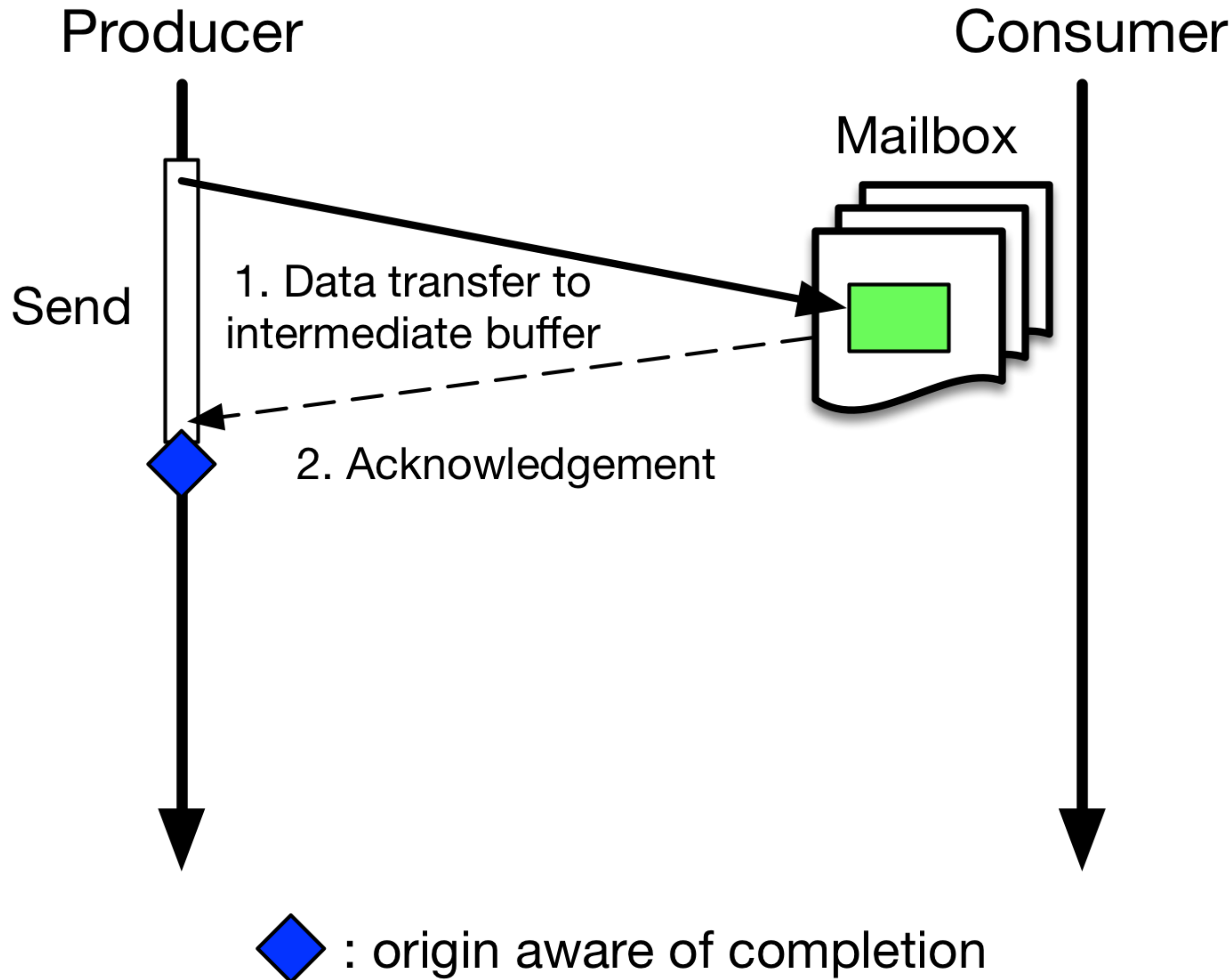
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High performance RDMA protocols in HPC.", EuroMPI'06

MPI-1 Message Passing – Simple Eager



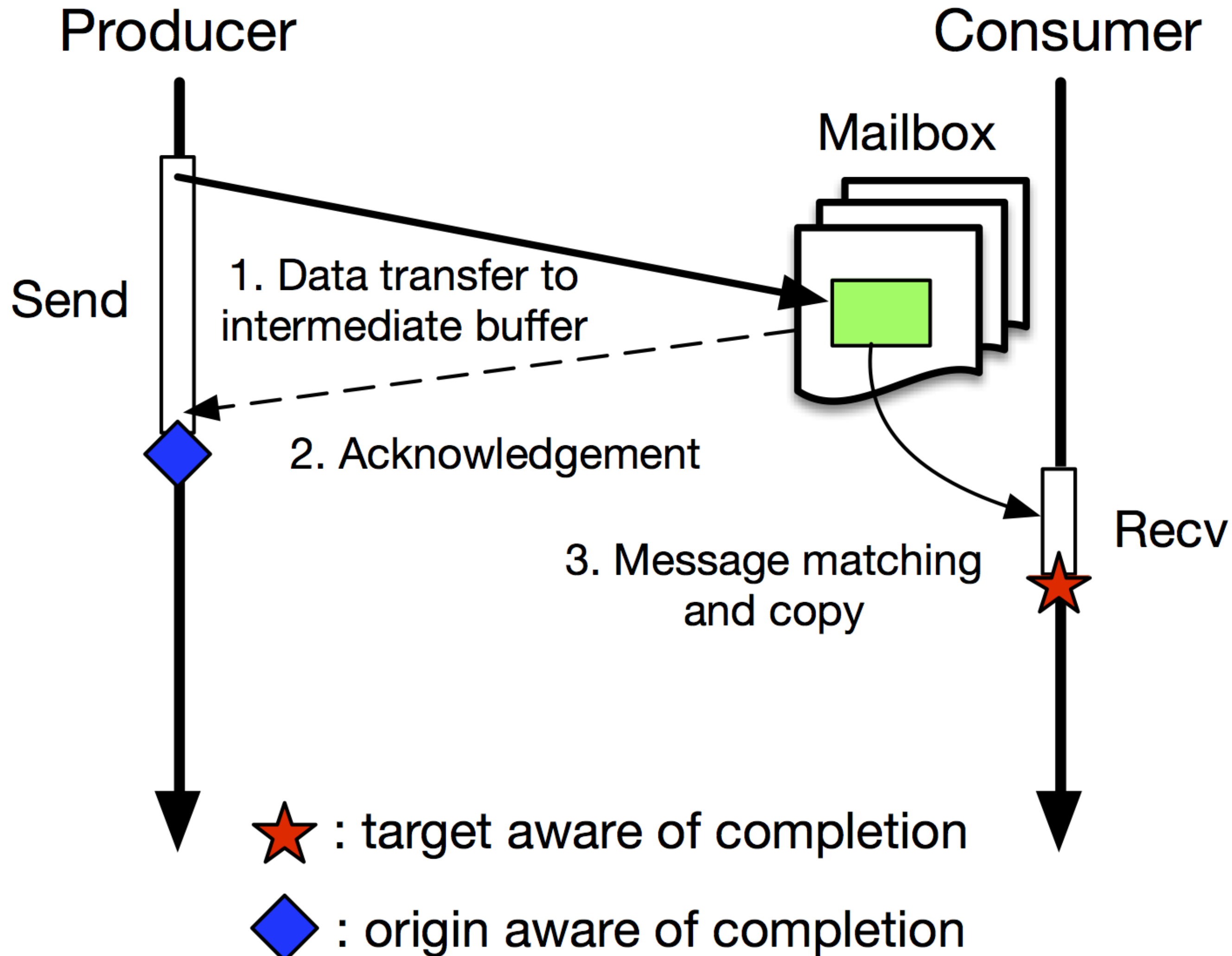
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, “High performance RDMA protocols in HPC.”, EuroMPI’06

MPI-1 Message Passing – Simple Eager



[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, “High performance RDMA protocols in HPC.”, EuroMPI’06

MPI-1 Message Passing – Simple Eager



Critical path: 1 latency + 1 copy

[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High performance RDMA protocols in HPC.", EuroMPI'06

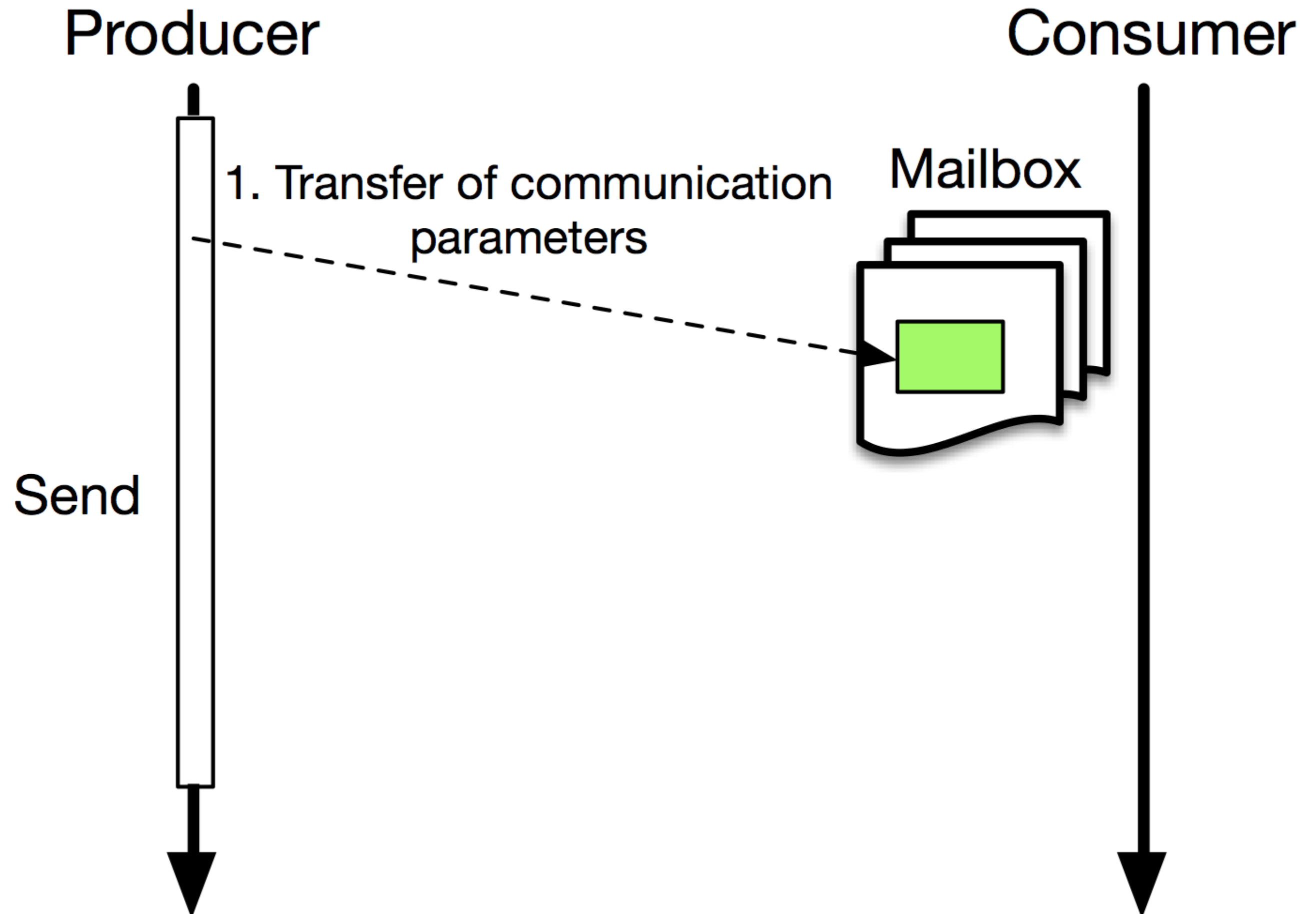
MPI-1 Message Passing – Simple Rendezvous

Producer

Consumer

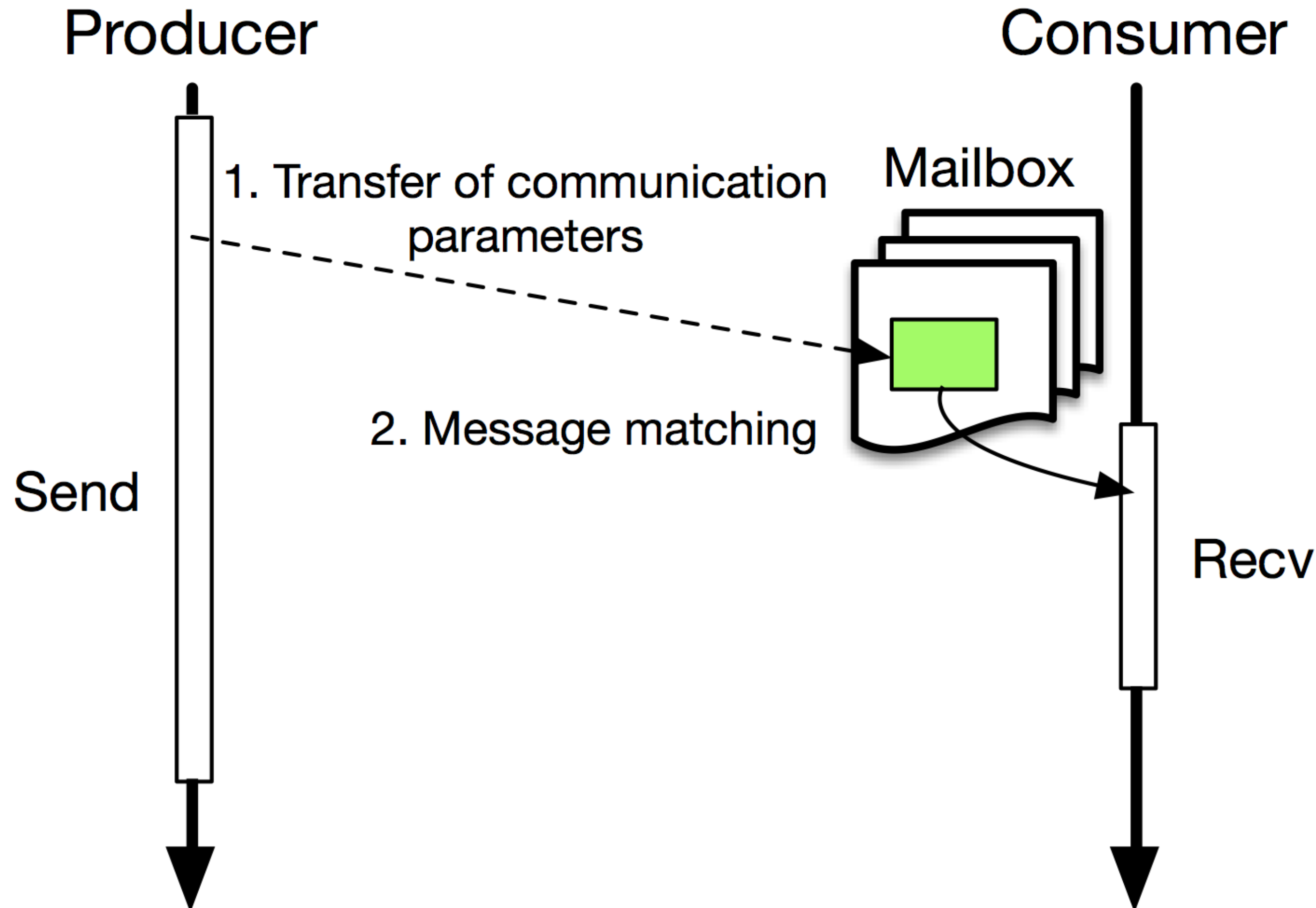
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, “High performance RDMA protocols in HPC.”, EuroMPI’06

MPI-1 Message Passing – Simple Rendezvous



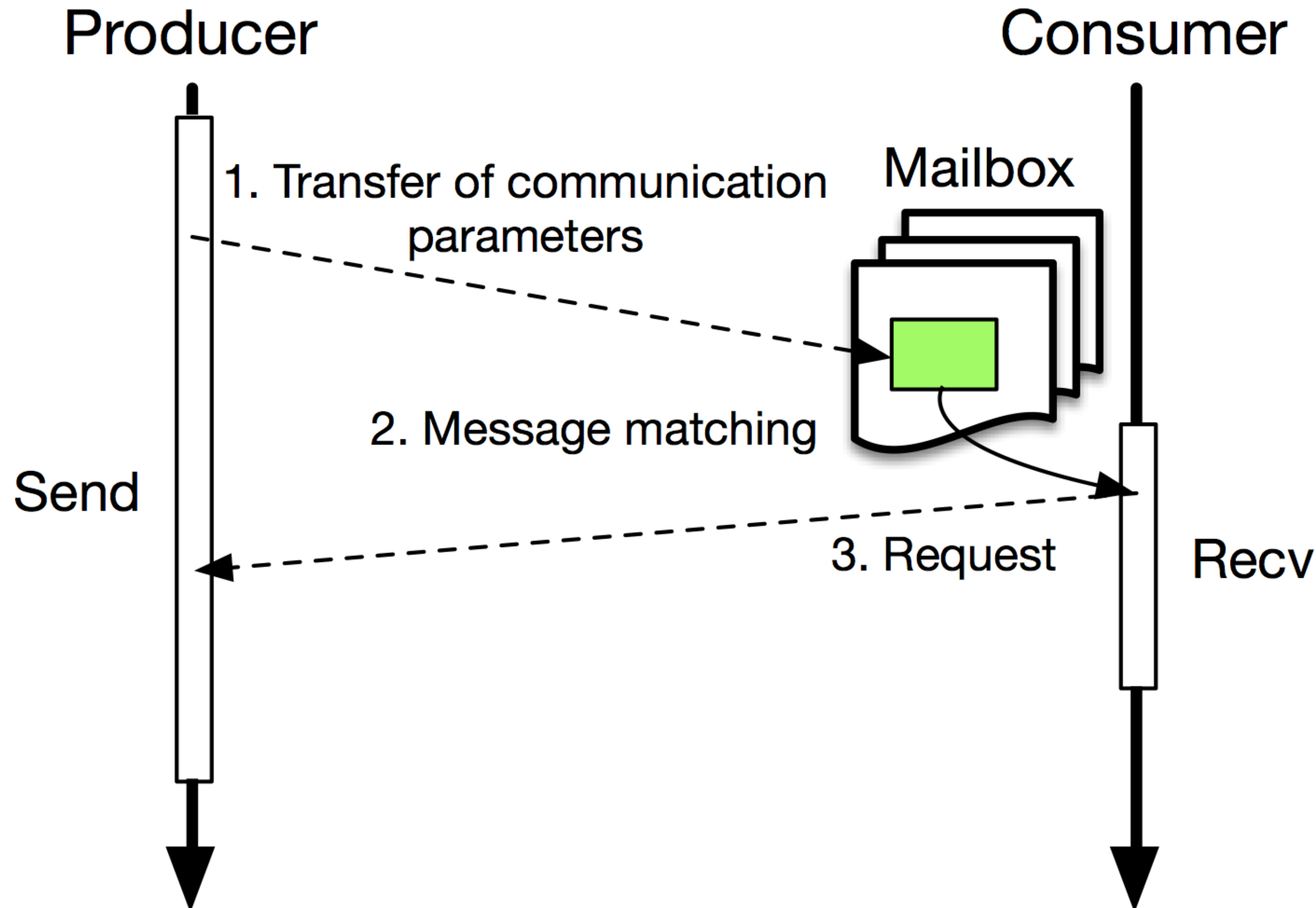
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, “High performance RDMA protocols in HPC.”, EuroMPI’06

MPI-1 Message Passing – Simple Rendezvous



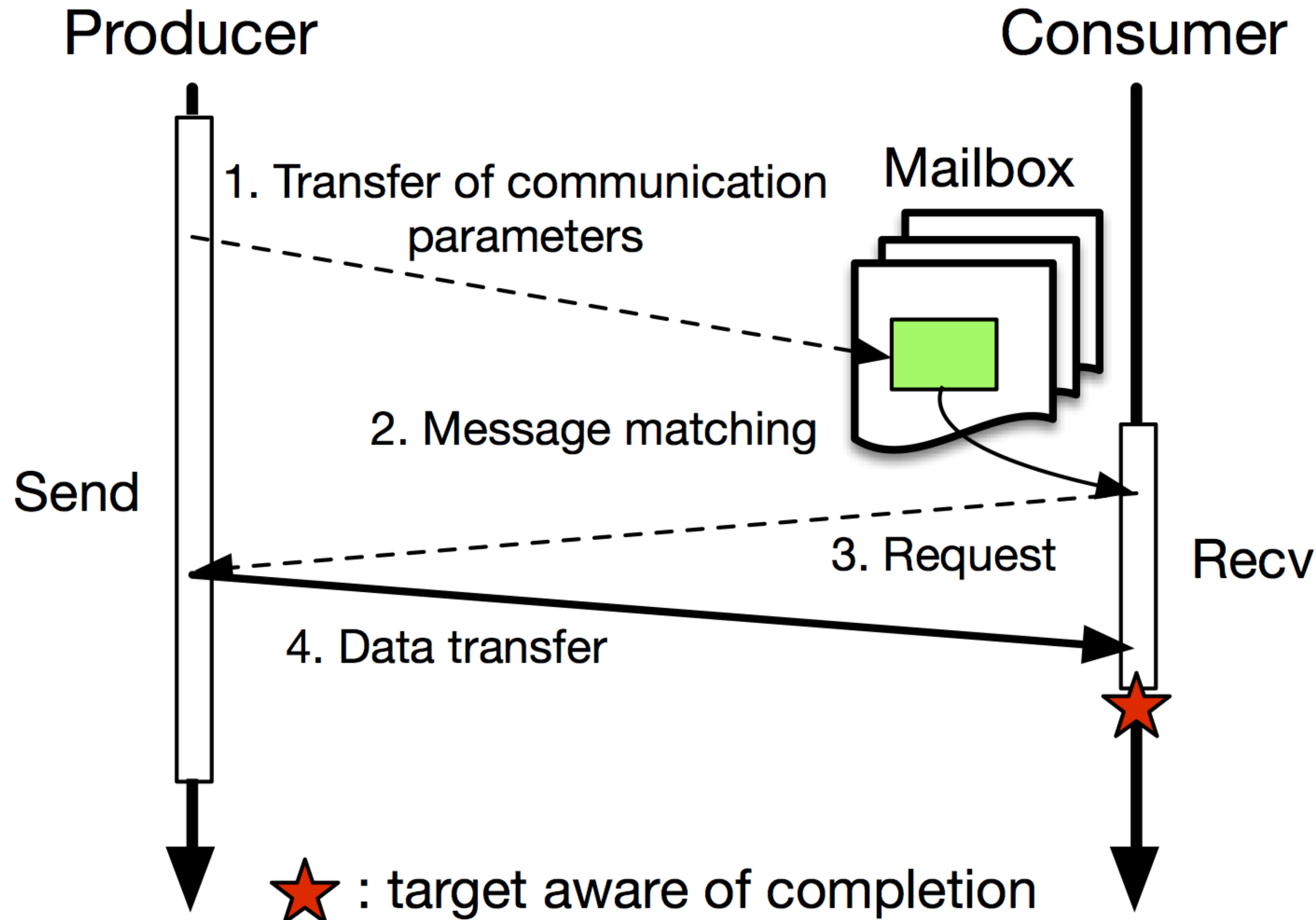
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, “High performance RDMA protocols in HPC.”, EuroMPI'06

MPI-1 Message Passing – Simple Rendezvous



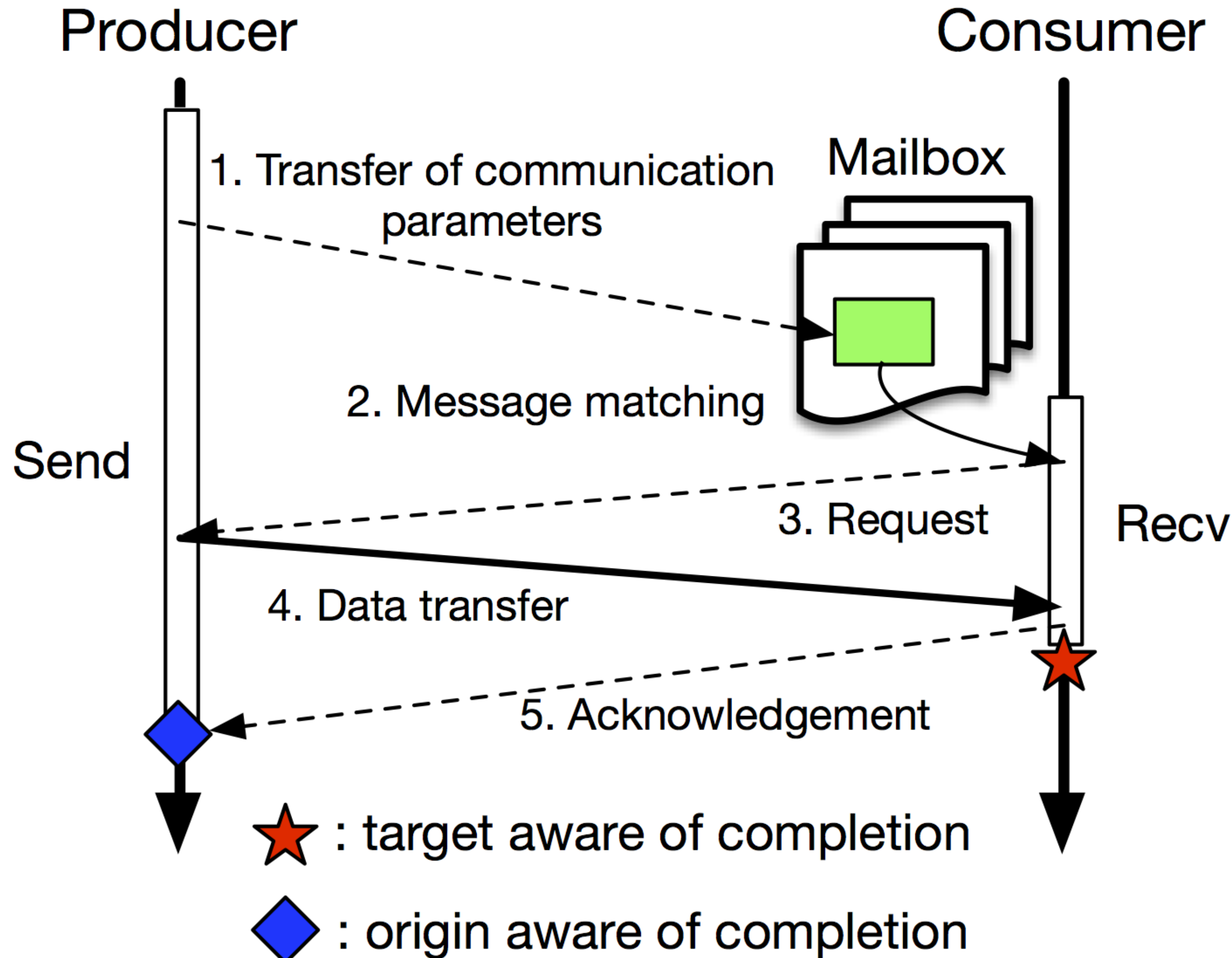
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, “High performance RDMA protocols in HPC.”, EuroMPI’06

MPI-1 Message Passing – Simple Rendezvous



[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, “High performance RDMA protocols in HPC.”, EuroMPI’06

MPI-1 Message Passing – Simple Rendezvous



Critical path: 3 latencies

[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High performance RDMA protocols in HPC.", EuroMPI'06

Communication in Today's

August 18, 2006

A Critique of RDMA

by Patrick Geoffray, Ph.D.

Do you remember VIA, the Virtual Interface Architecture? I do. In 1998, according to its promoters — Intel, Compaq, and Microsoft — VIA was supposed to change the face of high-performance networking. VIA was a buzzword at the time; Venture Capital was flowing, and startups multiplying. Many HPC pundits were rallying behind this low-level programming interface, which promised scalable, low-overhead, high-throughput communication, initially for HPC and eventually for the data center. The hype was on and doom was spelled for the non-believers.

It turned out that VIA, based on RDMA (Remote Direct Memory Access, or Remote DMA), was not an improvement on existing APIs to support widely used application-software interfaces such as MPI and Sockets. After a while, VIA faded away, overtaken by other developments.

VIA was eventually reborn into the RDMA programming model that is the basis of various InfiniBand Verbs implementations, as well as DAPL (Direct Access Provider Library) and iWARP (Internet Wide Area RDMA Protocol). The pundits have returned, VCs are spending their money, and RDMA is touted as an ideal solution for the efficiency of high-performance networks.

However, the evidence I'll present here shows that the revamped RDMA model is more a problem than a solution. What's more, the objective that RDMA pretends to address of efficient user-level communication between computing nodes is already solved by the two-sided Send/Recv model in products such as Quadrics QsNet, Cray SeaStar (implementing Sandia Portals), Qlogic InfiniPath, and Myricom's Myrinet Express (MX).

Send/Recv versus RDMA

The difference between these two paradigms, Send/Receive (Send/Recv) and RDMA, resides essentially in the

http://www.hpcwire.com/2006/08/18/a_critique_of_rdma-1/

Remote Memory Access Programming

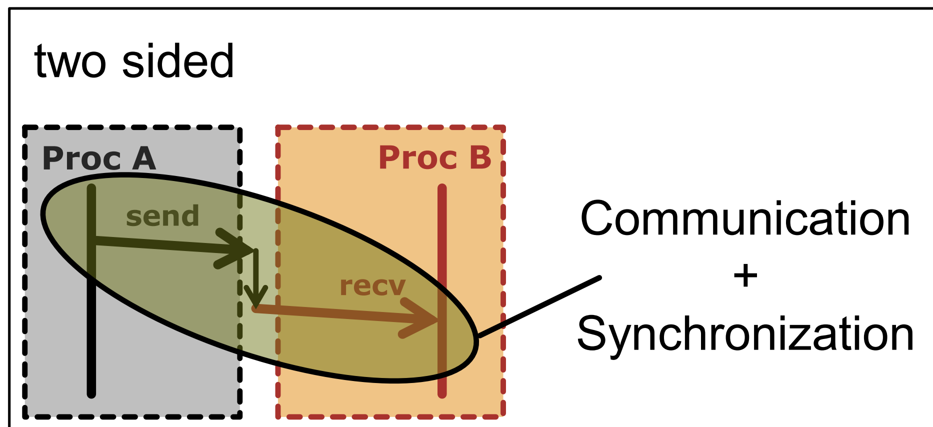
- **Why not use these RDMA features more directly?**
  - A global address space may simplify programming
  - … and accelerate communication
  - … and there could be a widely accepted standard

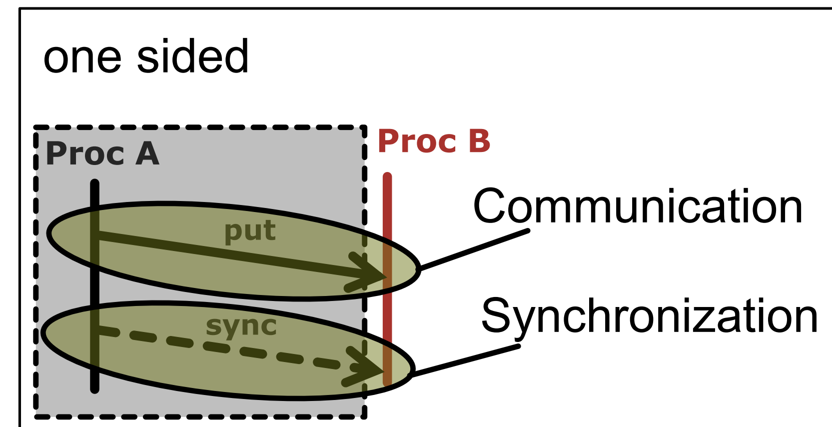
- **MPI-3 RMA ("MPI One Sided") was born**
  - Just one among many others (UPC, CAF, …)
  - Designed to react to hardware trends, learn from others
  - Direct (hardware-supported) remote access
  - New way of thinking for programmers

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

MPI-3 RMA SUMMARY

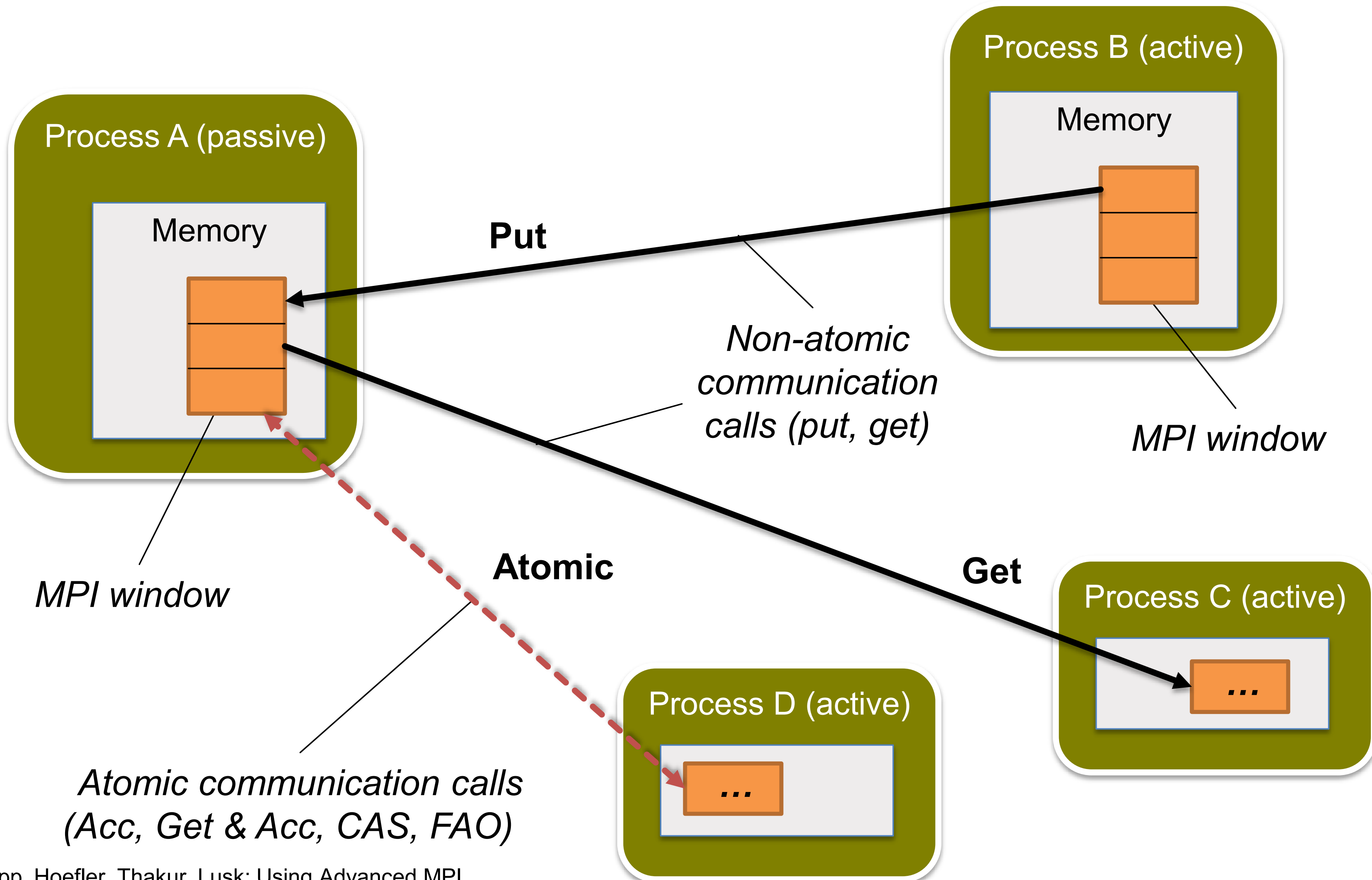
- **MPI-3 updates RMA ("MPI One Sided")**
  - Significant change from MPI-2
- **Communication is „one sided" (no involvement of destination)**
  - Utilize direct memory access
- **RMA decouples communication & synchronization**
  - Fundamentally different from message passing





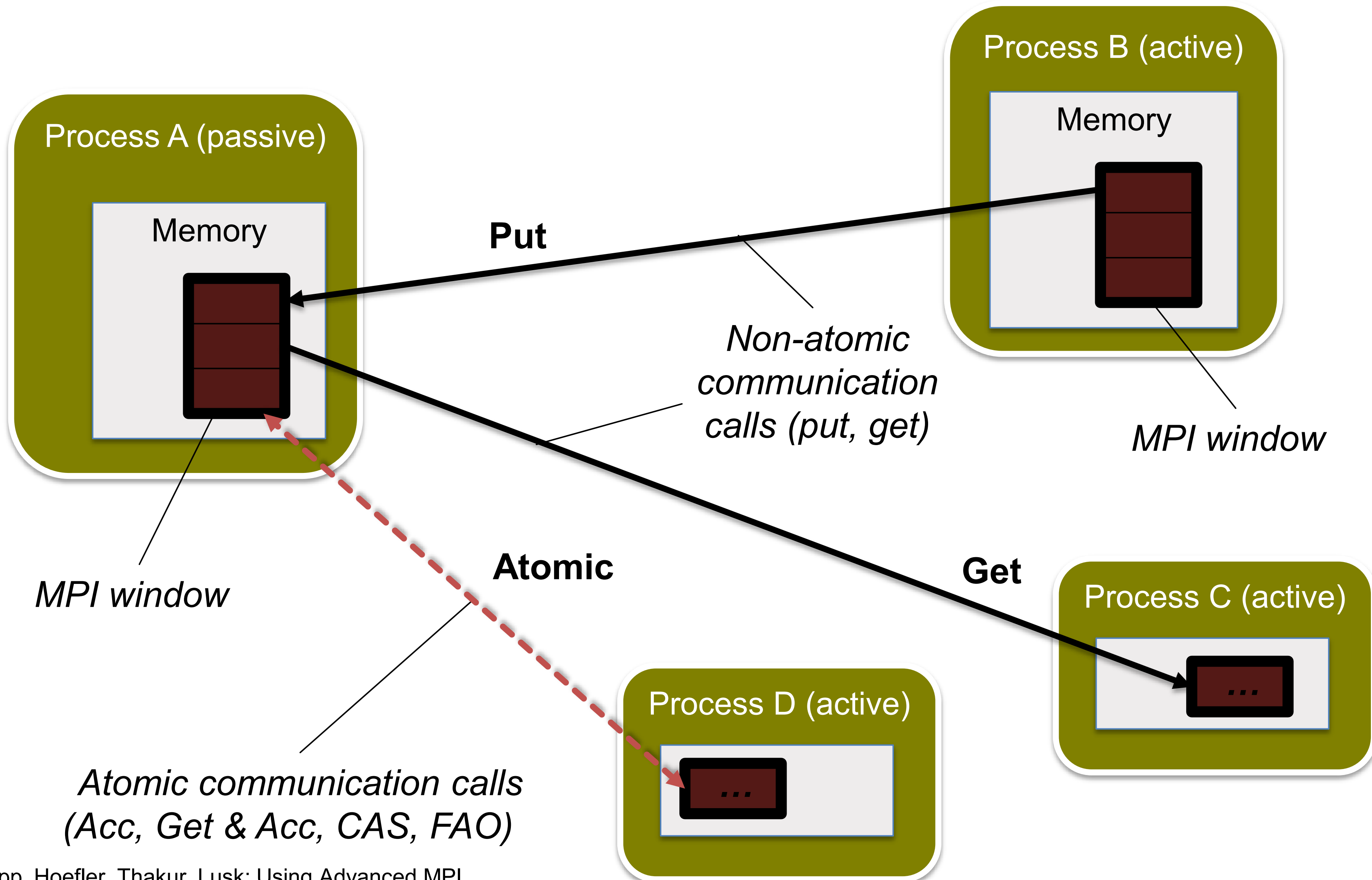
[1] http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

MPI-3 RMA Communication Overview



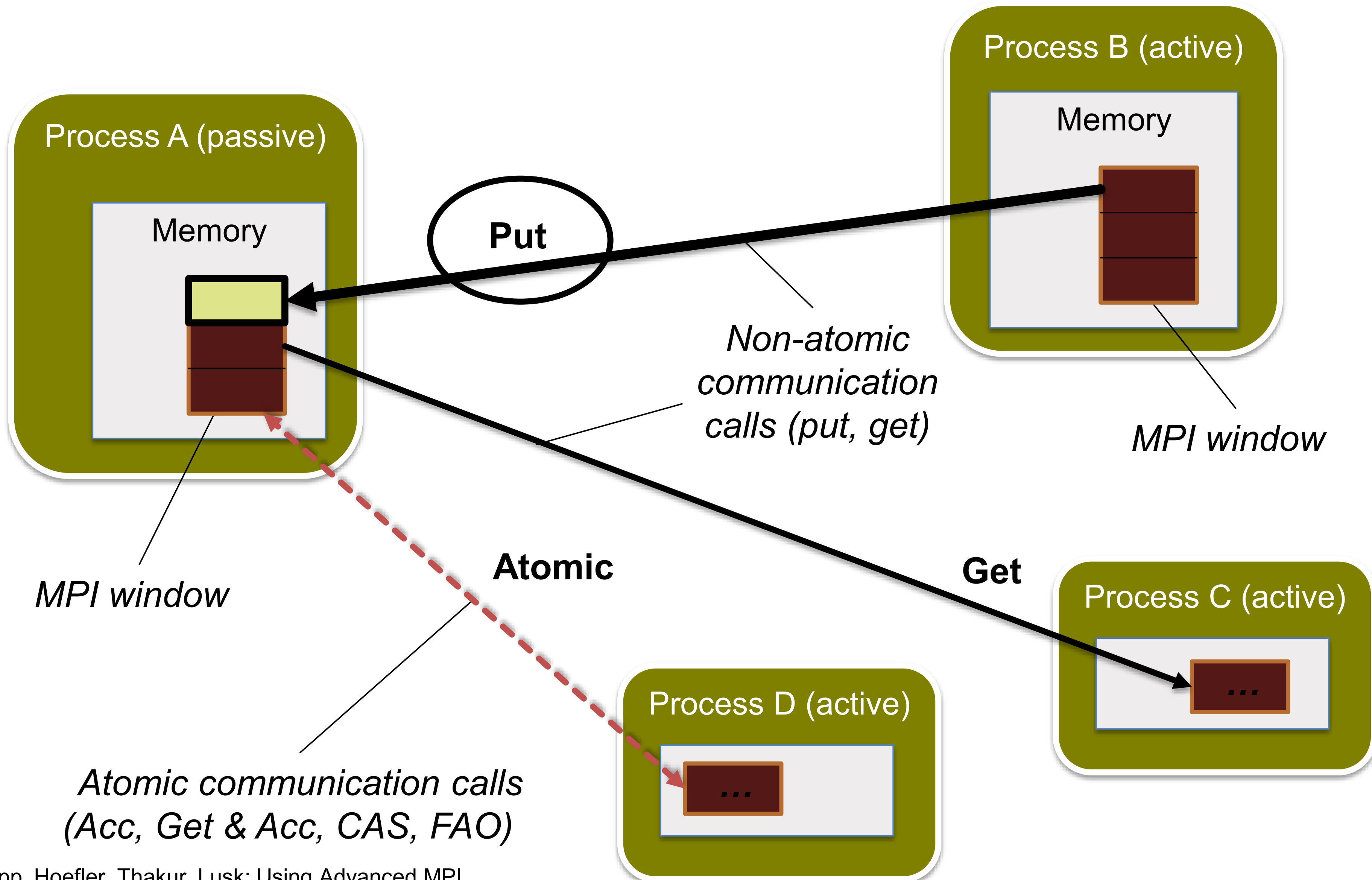
Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI

MPI-3 RMA Communication Overview



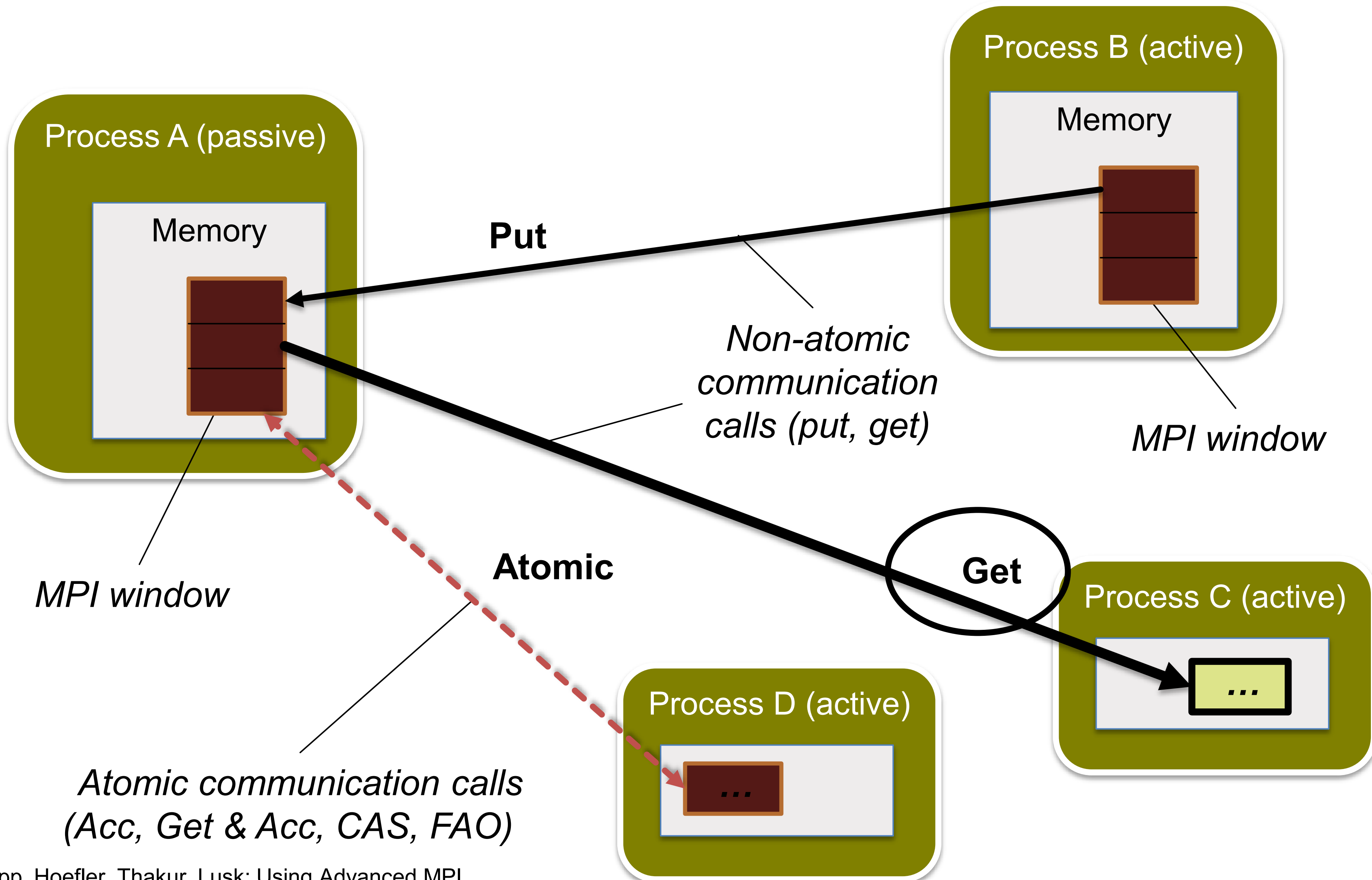
Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI

MPI-3 RMA Communication Overview



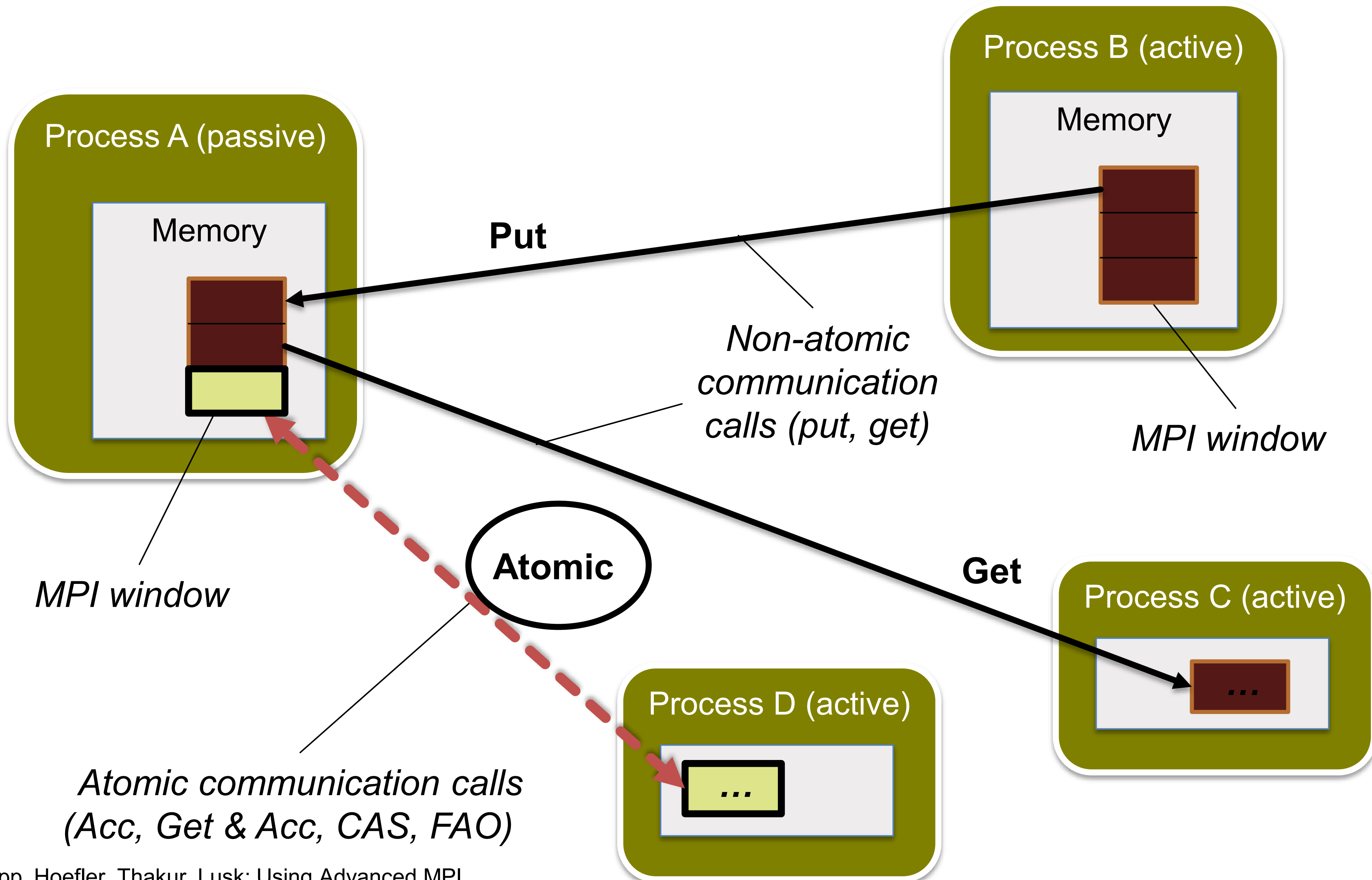
Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI

MPI-3 RMA Communication Overview



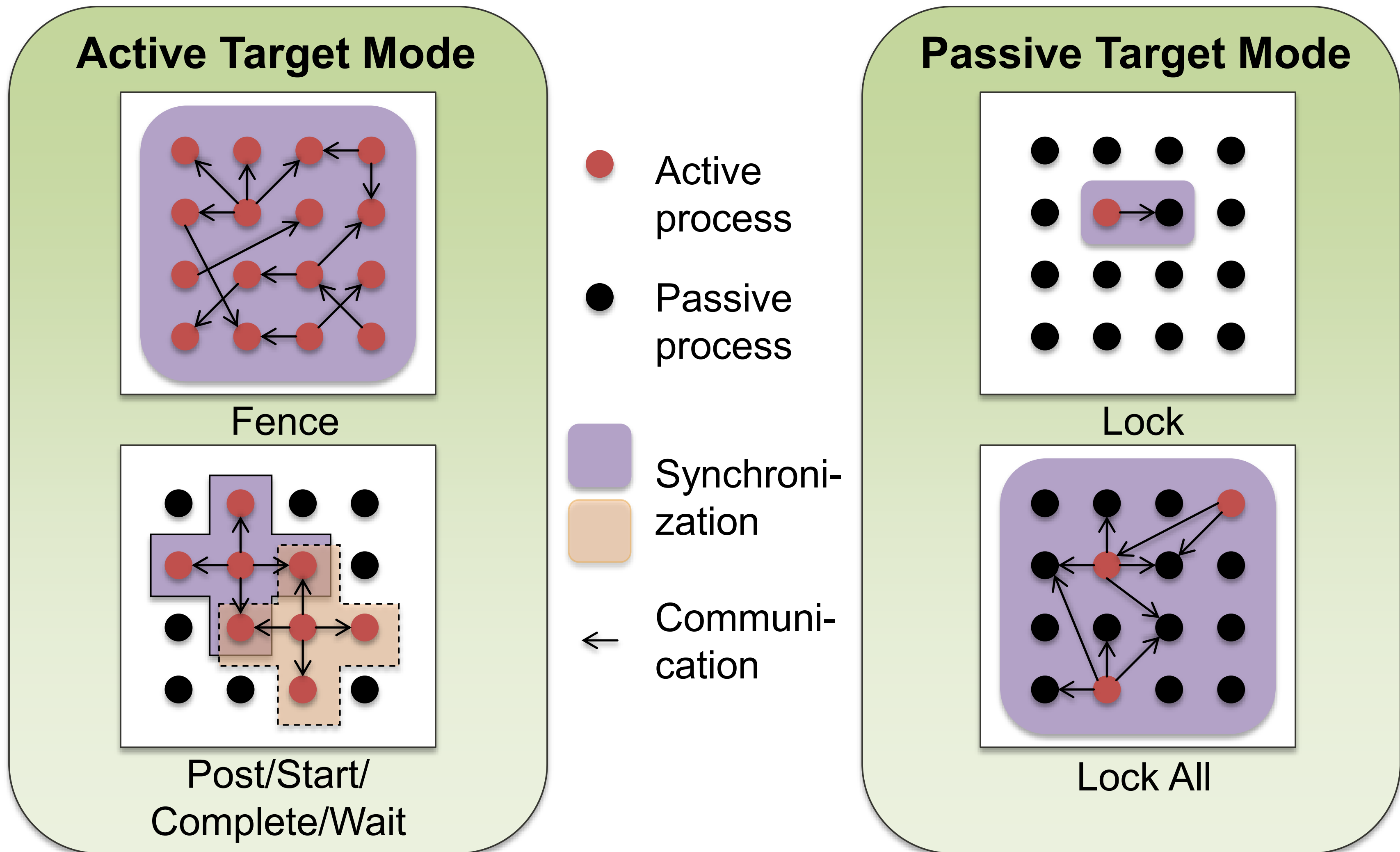
Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI

MPI-3 RMA Communication Overview



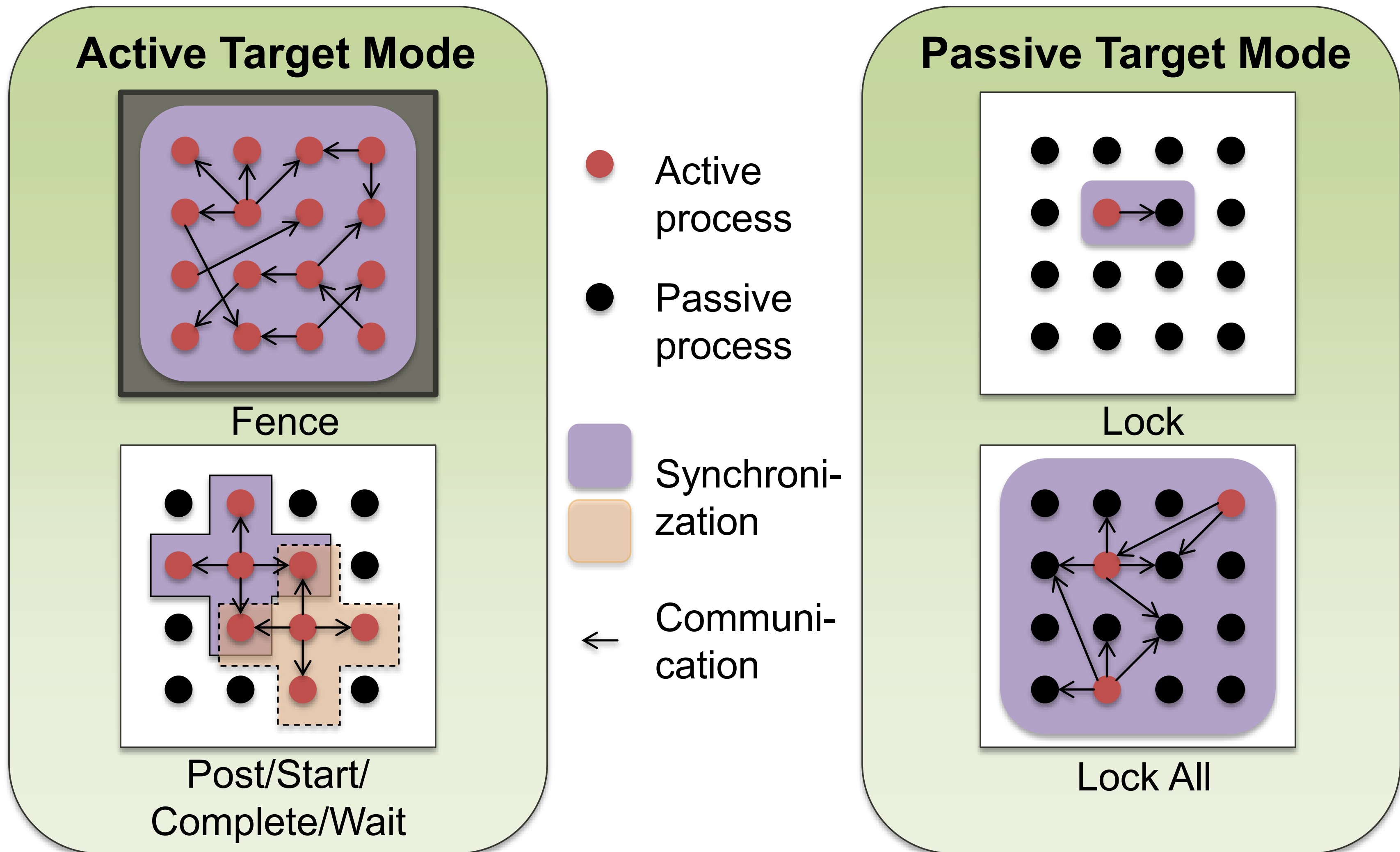
Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI

MPI-3 RMA Synchronization Overview



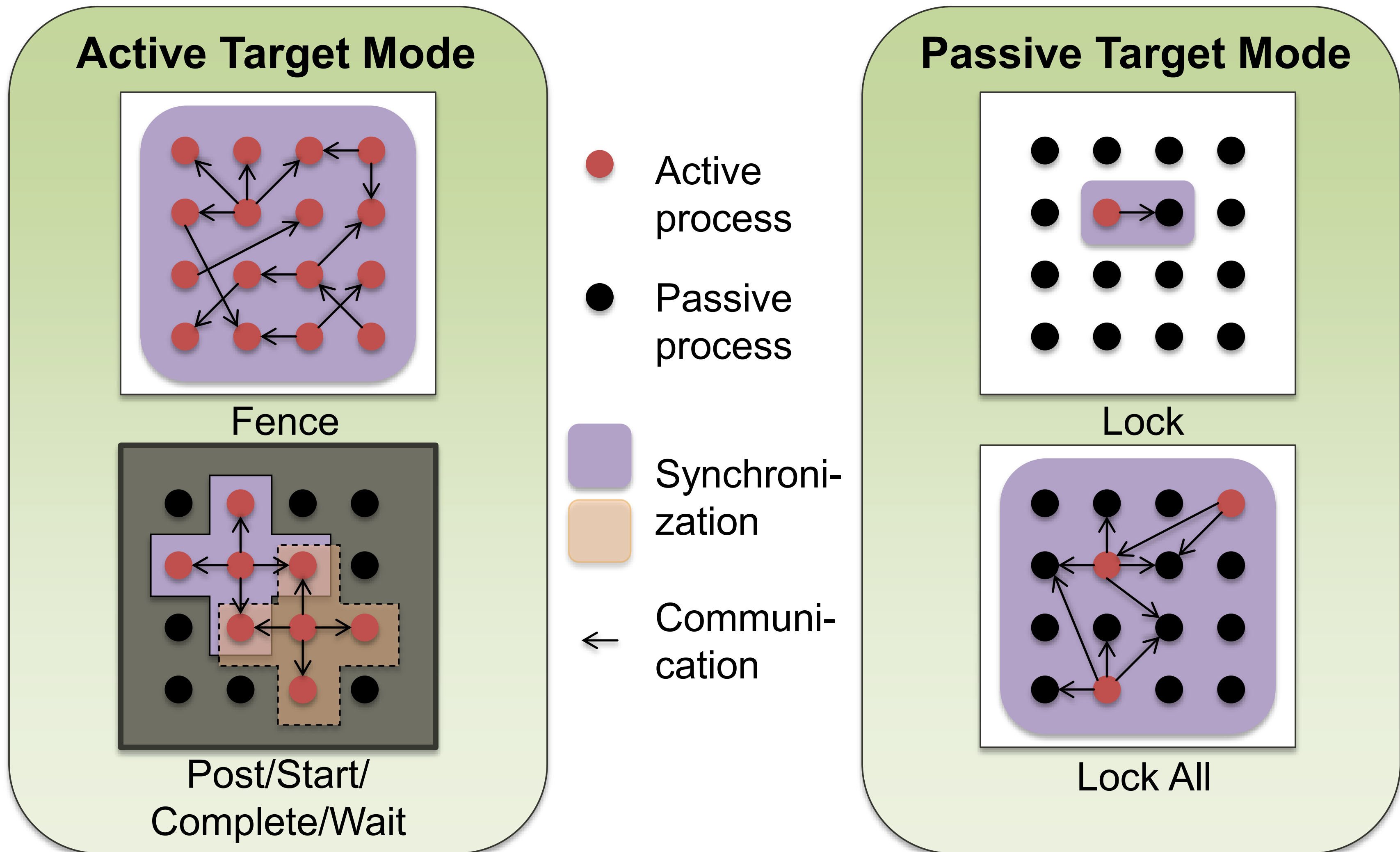
Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI

MPI-3 RMA Synchronization Overview



Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI

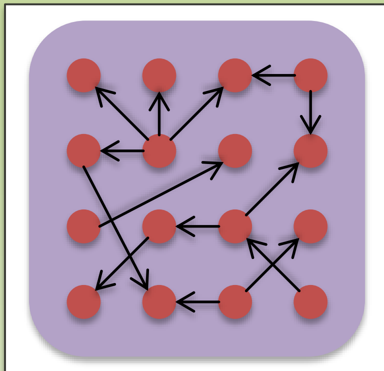
MPI-3 RMA Synchronization Overview



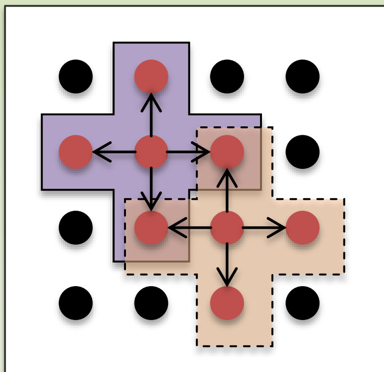
Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI

MPI-3 RMA Synchronization Overview

Active Target Mode



Fence



Post/Start/
Complete/Wait

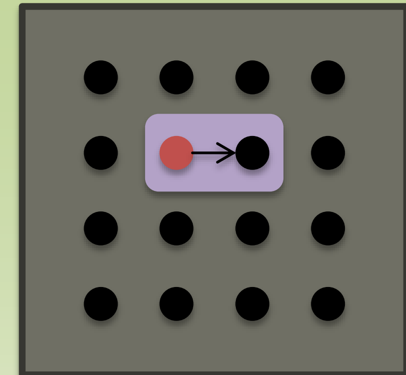
Active process

Passive process

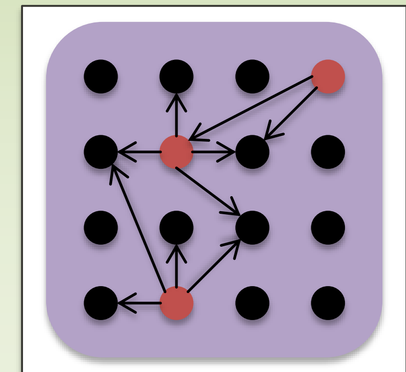
Synchronization

Communication

Passive Target Mode



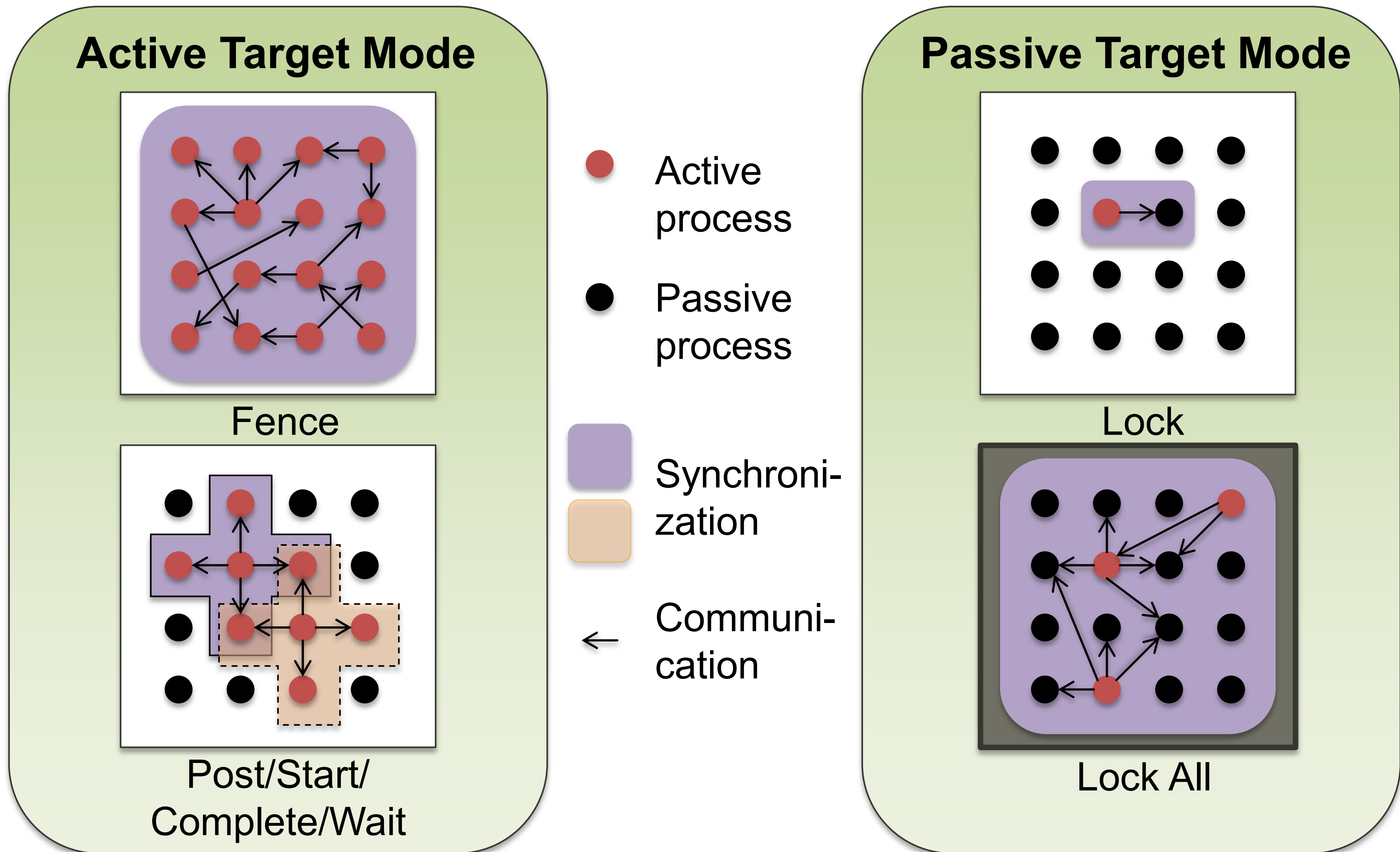
Lock



Lock All

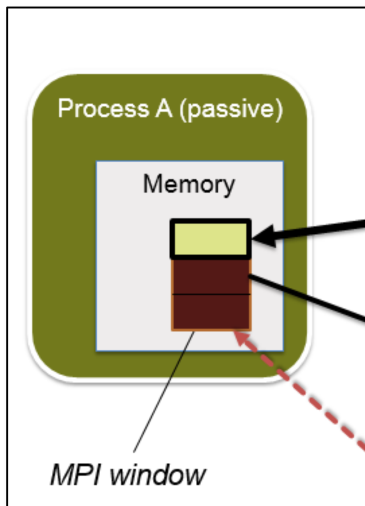
Gropp, Hoefler, Thakur, Lusk: Using Advanced MPI

MPI-3 RMA Synchronization Overview

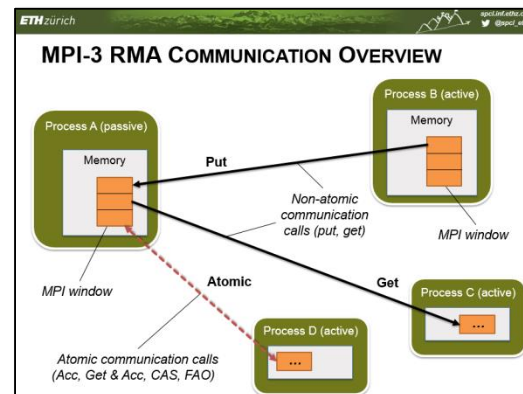


Scalable Protocols & Reference Implementation

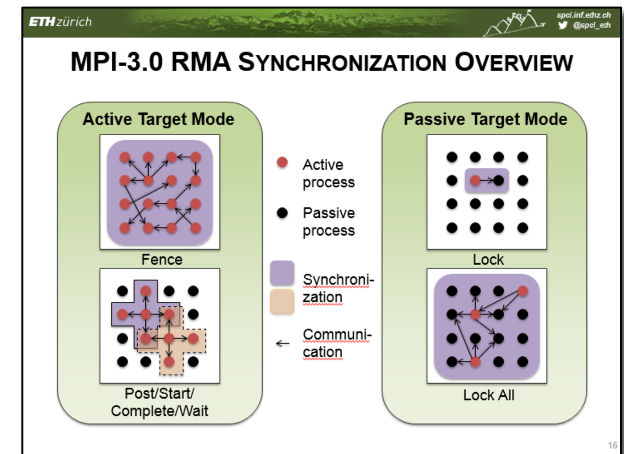
- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)
  - Window creation, communication and synchronization



Window creation



Communication

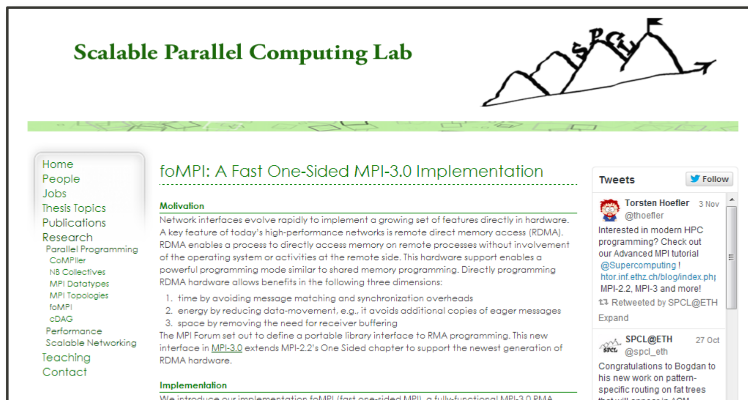


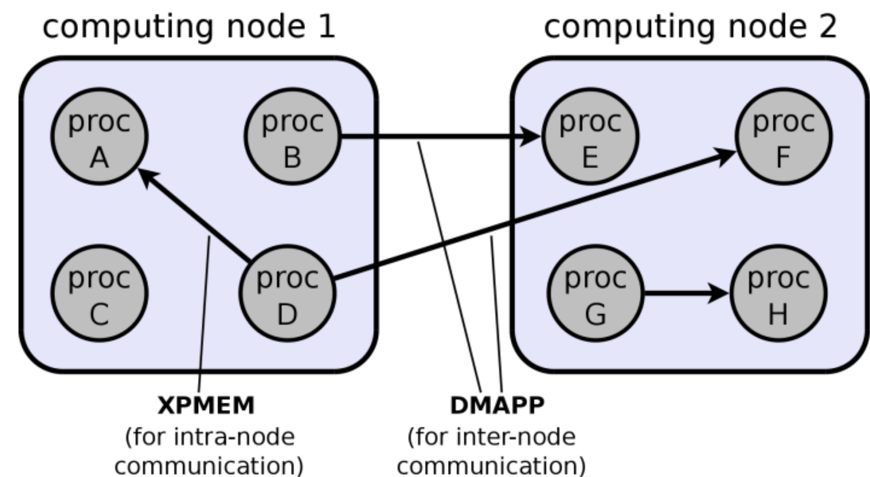
Synchronization

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

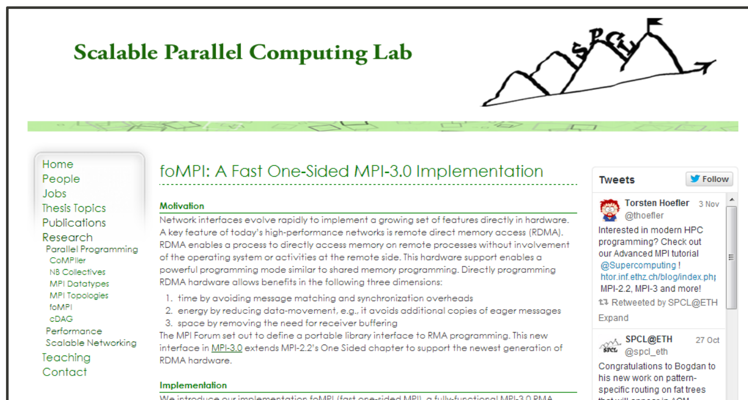


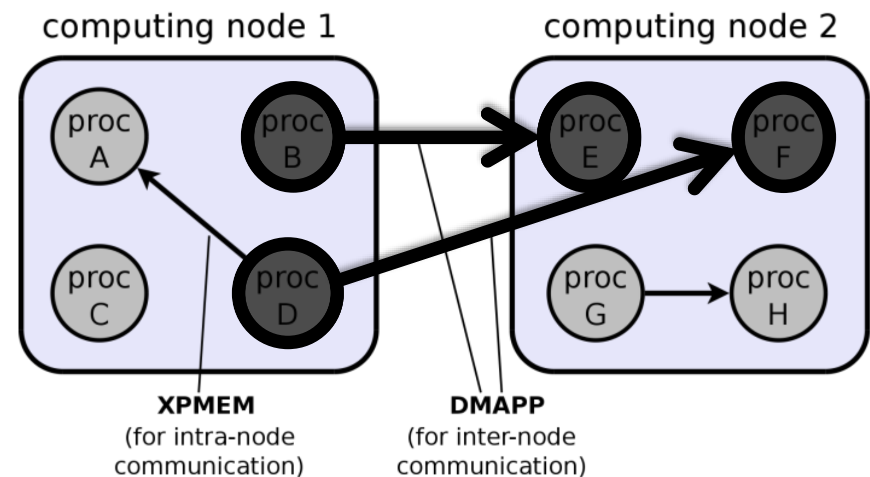


http://spcl.inf.ethz.ch/Research/Parallel_Programming/foMPI

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

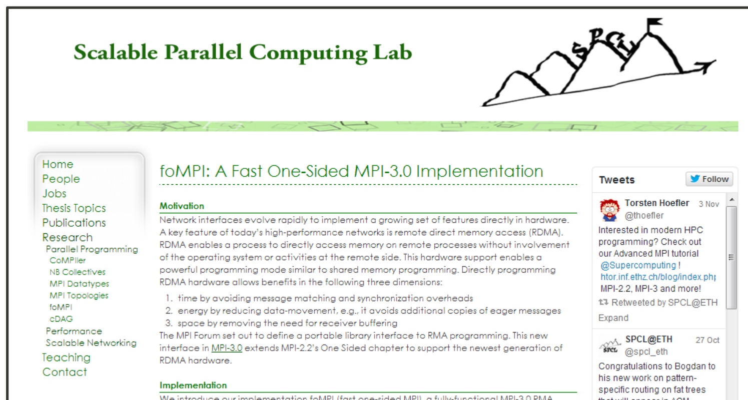


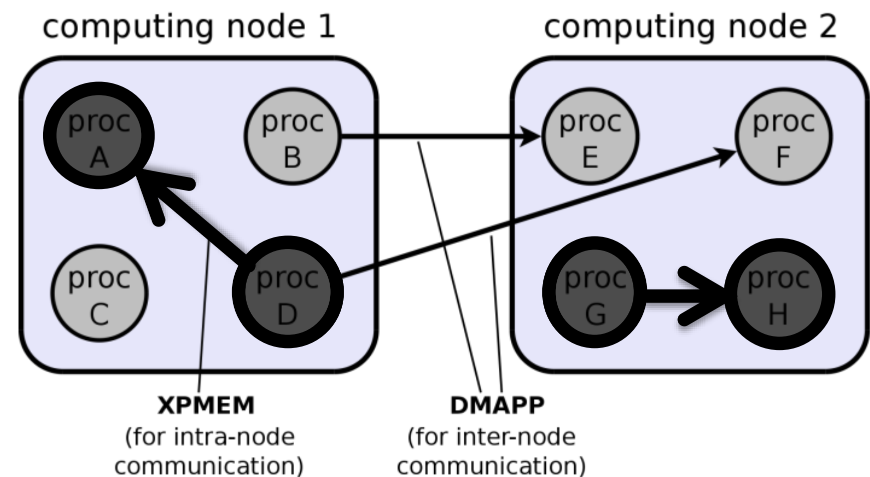


http://spcl.inf.ethz.ch/Research/Parallel_Programming/foMPI

Scalable Protocols & Reference Implementation

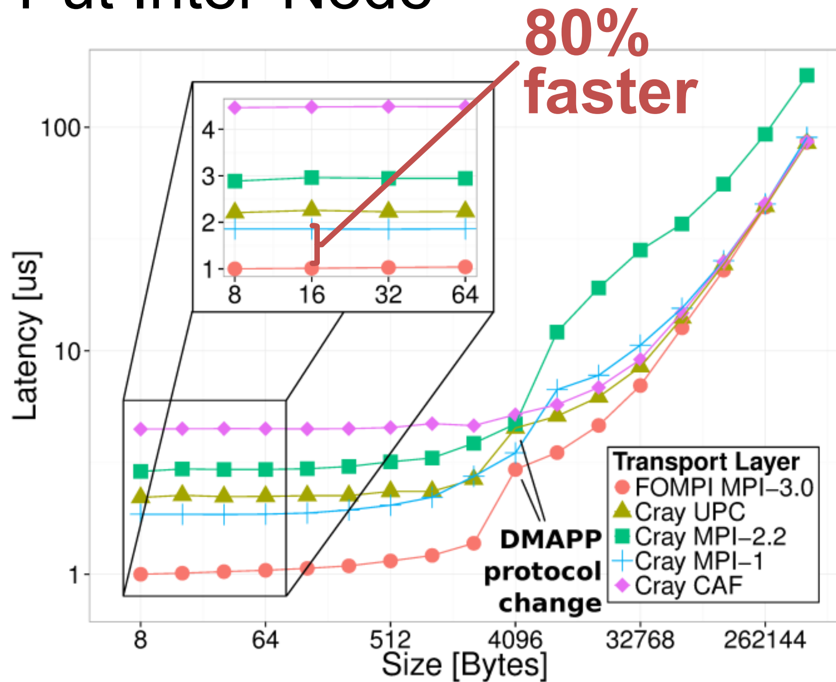
- Scalable & generic protocols
  - Can be used on any RDMA network (e.g., OFED/IB)
  - Window creation, communication and synchronization
- foMPI, a fully functional MPI-3 RMA implementation
  - DMAPP: lowest-level networking API for Cray Gemini/Aries systems
  - XPMEM: a portable Linux kernel module

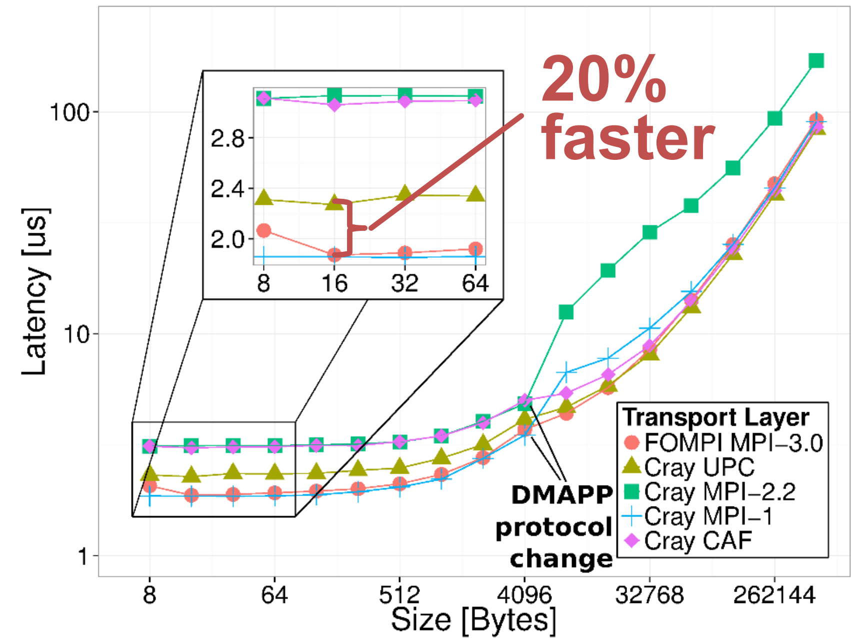


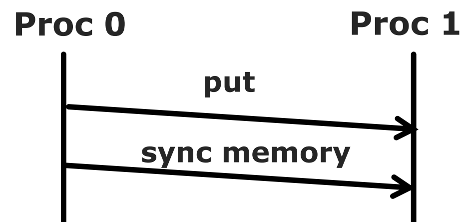


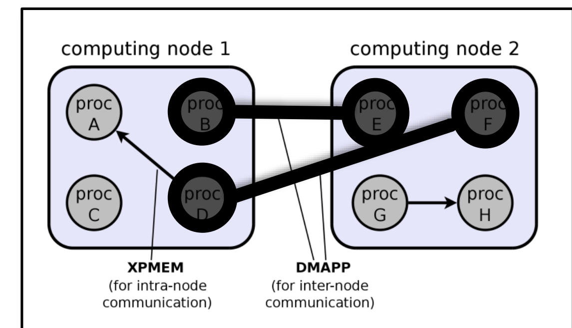
http://spcl.inf.ethz.ch/Research/Parallel_Programming/foMPI

Performance Inter-node: Latency









Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13

Performance Intra-node: Latency

Put/Get Intra-Node

3x faster

Latency [us]

100.0
10.0
1.0
0.1

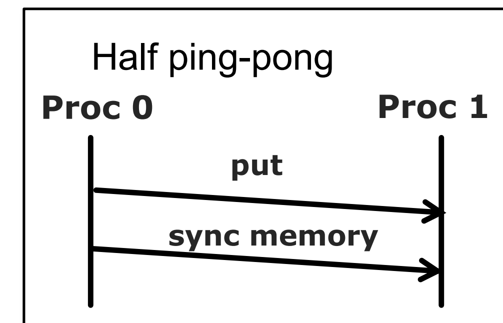
1.6
1.2
0.8
0.4

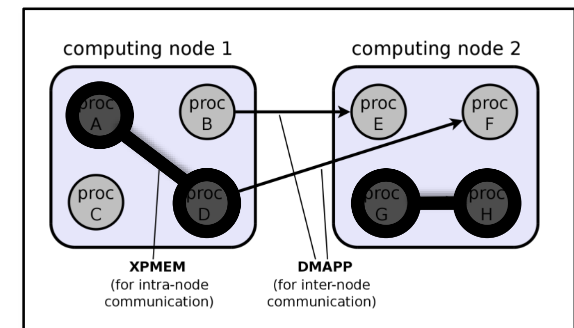
8 16 32 64

8 64 512 4096 32768 262144

Size [Bytes]

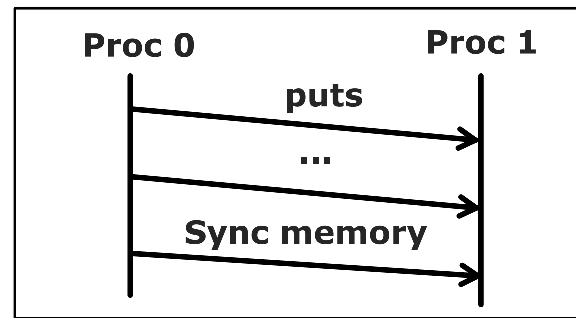
Transport Layer
- FOMPI MPI–3.0
- Cray UPC
- Cray MPI–2.2
- Cray MPI–1
- Cray CAF



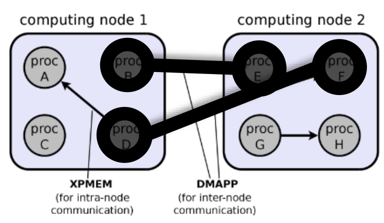


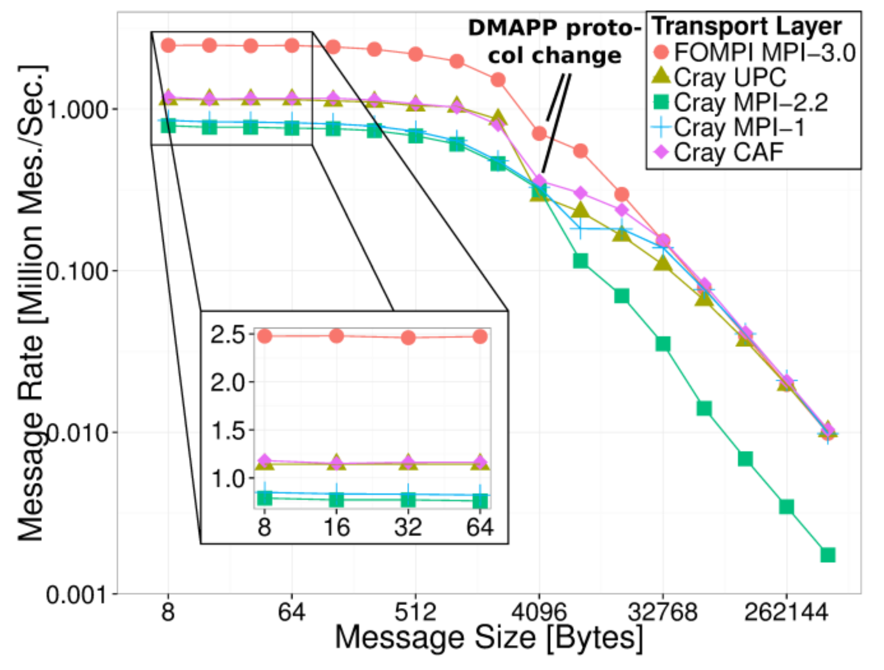
Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13

Performance: Message Rate

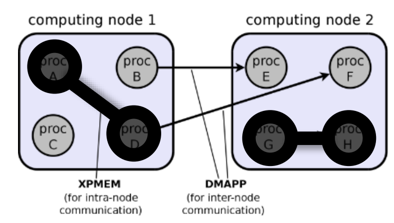


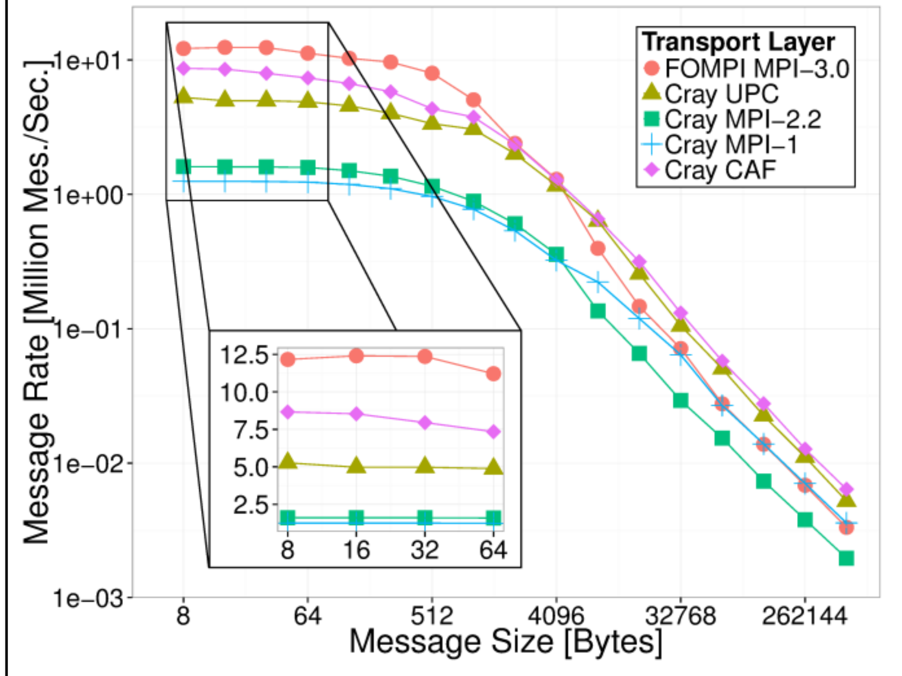
Inter-Node





Intra-Node

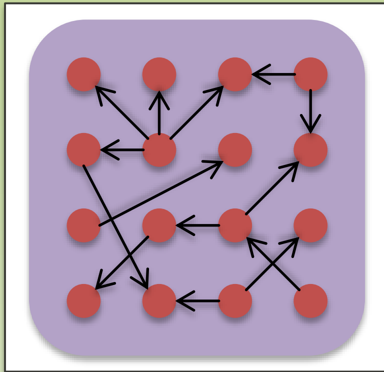


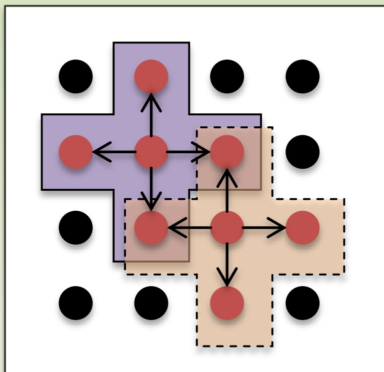


Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13

PART 3: SYNCHRONIZATION

Active Target Mode


Fence


Post/Start/
Complete/Wait

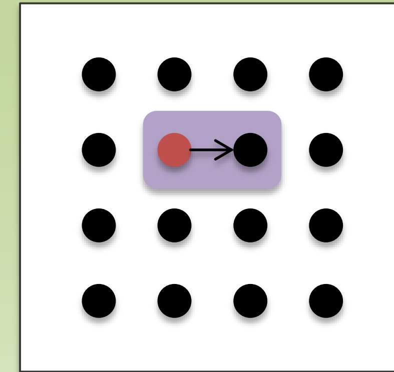
Active process

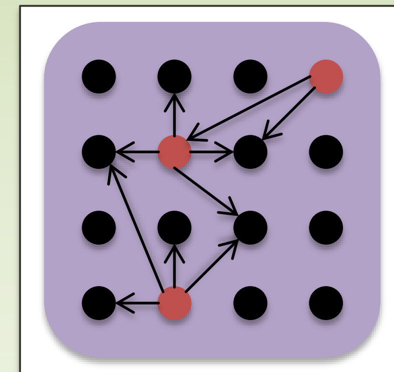
Passive process

Synchronization

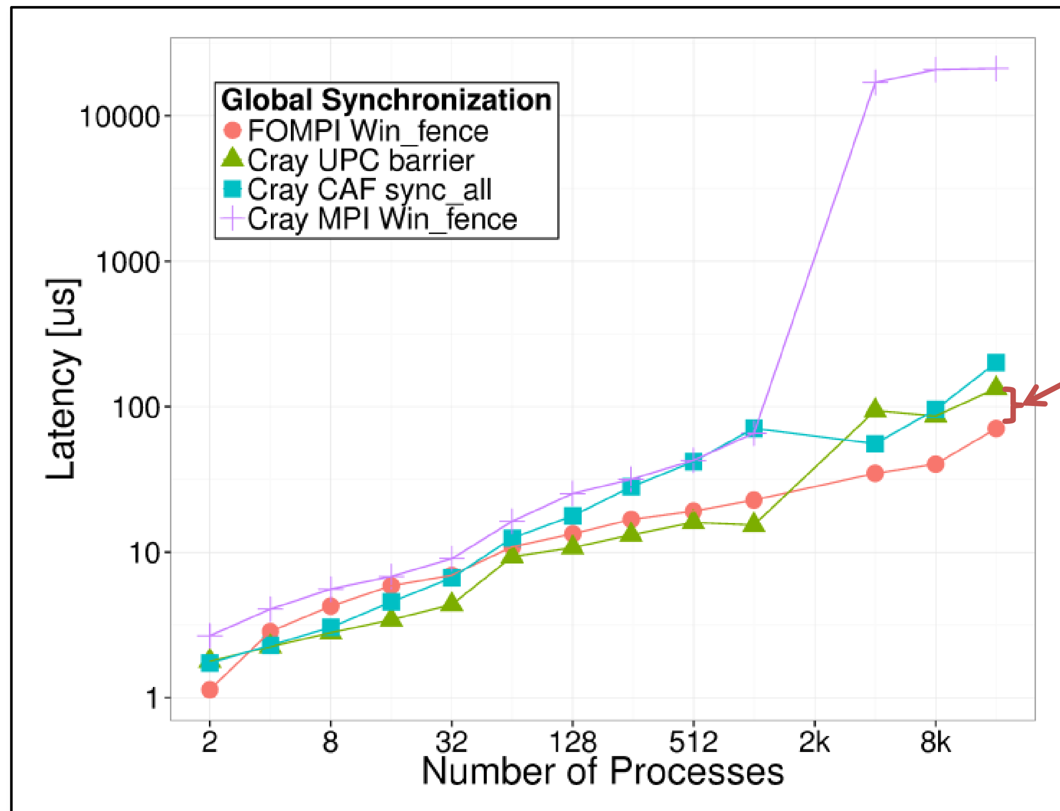
Communication

Passive Target Mode


Lock


Lock All

Scalable Fence Performance



90% faster

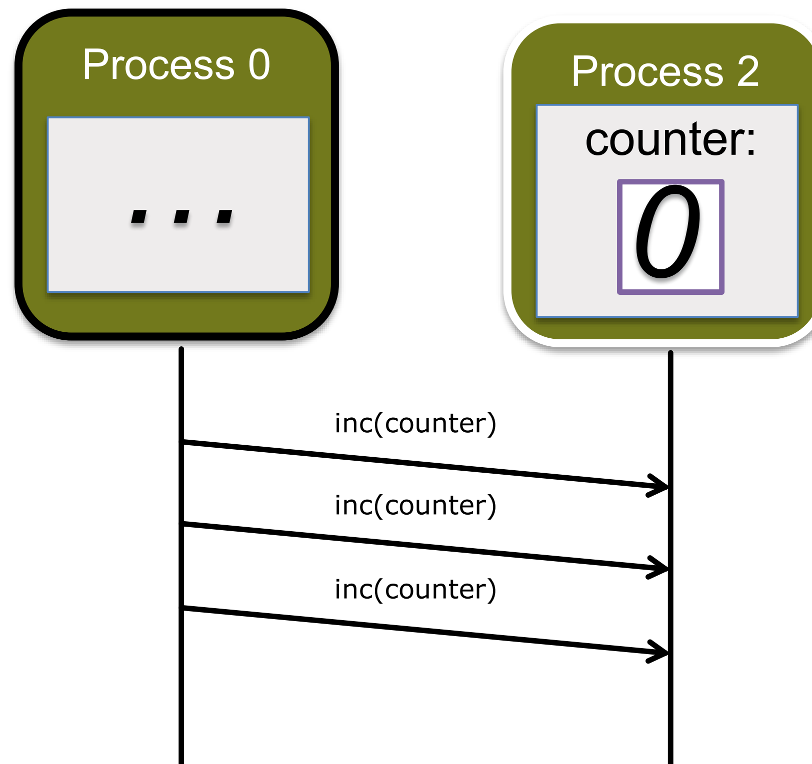
Time bound	$\mathcal{O}(\log p)$
Memory bound	$\mathcal{O}(1)$

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13

FLUSH SYNCHRONIZATION

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

- Guarantees remote completion
- Performs 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

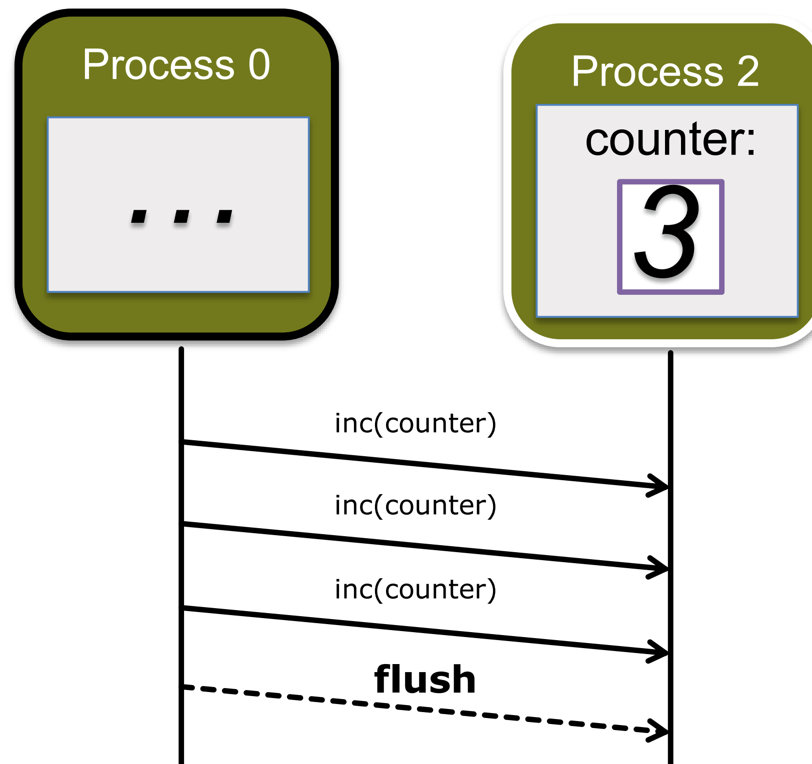


Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13

Flush Synchronization

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

- Guarantees remote completion
- Performs 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



Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13

Performance Modeling

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

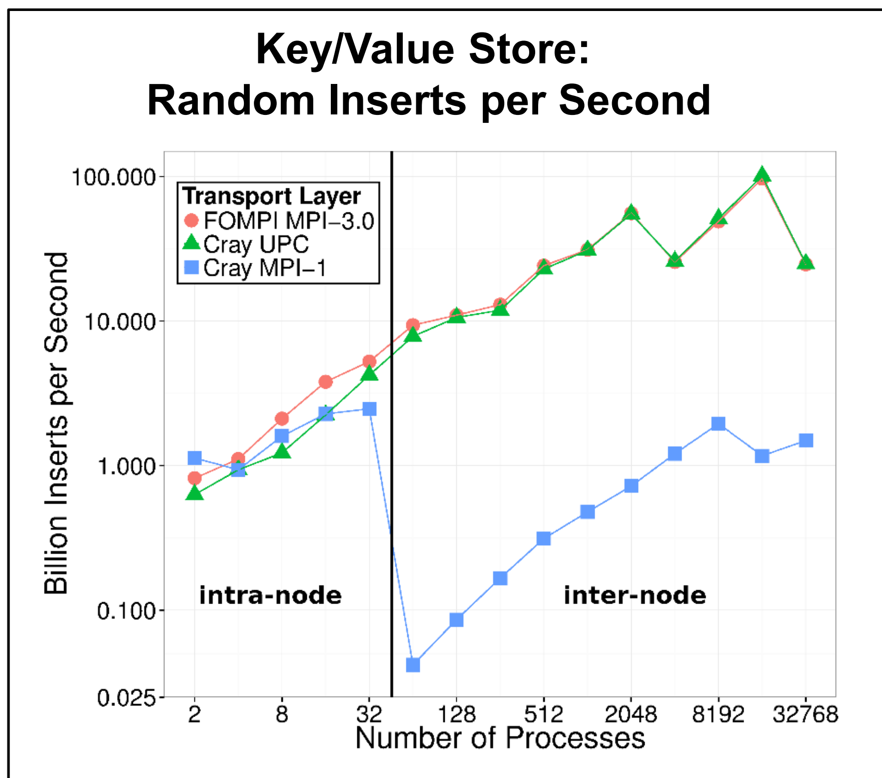
Put/get	$\mathcal{P}_{put} = 0.16ns \cdot s + 1\mu s$ $\mathcal{P}_{get} = 0.17ns \cdot s + 1.9\mu s$
Atomics	$\mathcal{P}_{acc,sum} = 28ns \cdot s + 2.4\mu s$ $\mathcal{P}_{acc,min} = 0.8ns \cdot s + 7.3\mu s$

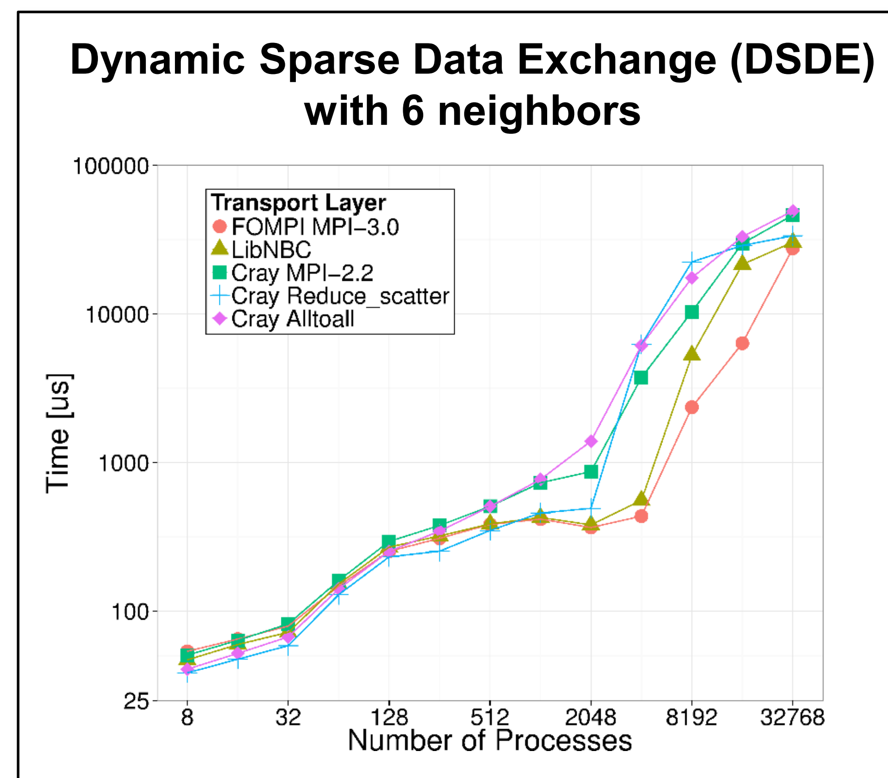
Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13

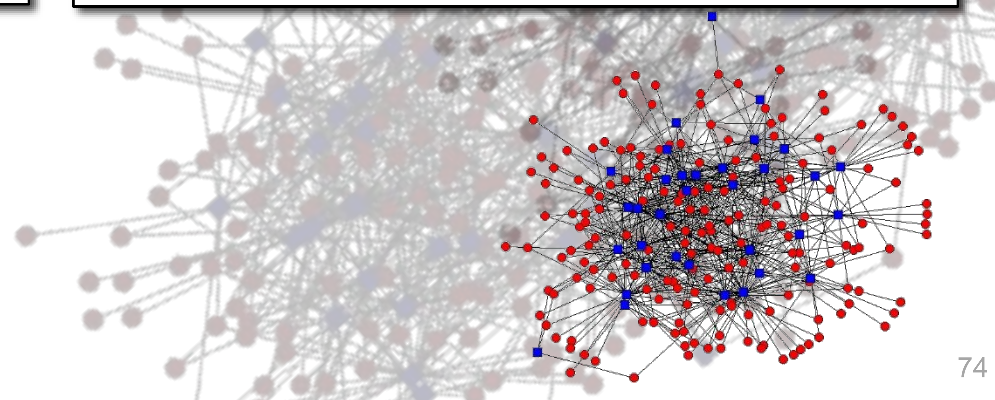
Application Performance

- 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

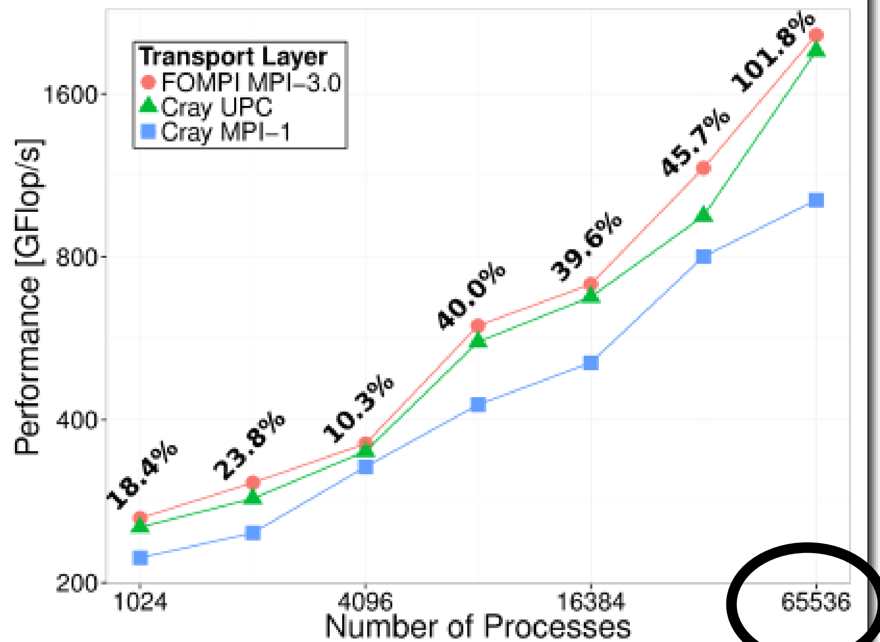


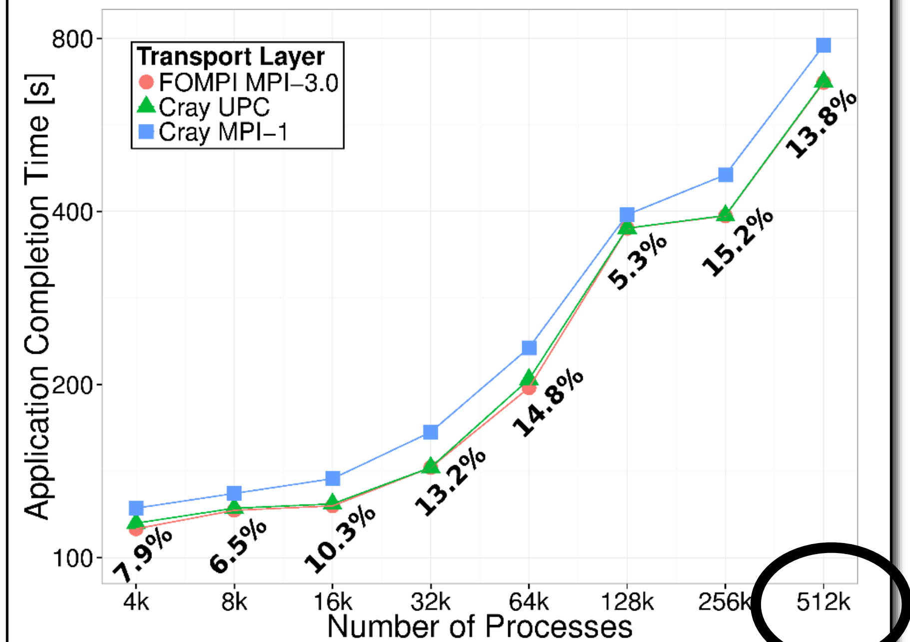




Performance: Applications

Annotations represent performance gain of foMPI [3] over Cray MPI-1.





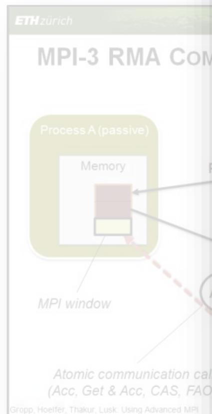
scale to 65k procs

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
[3] Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13

IN CASE YOU WANT TO LEARN MORE

- Available
  - Some ar

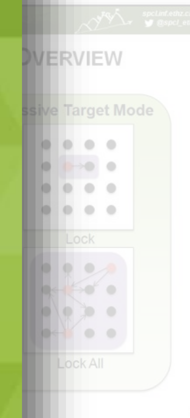


SCIENTIFIC AND ENGINEERING COMPUTATION SERIES

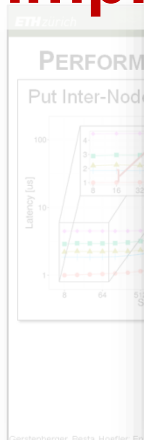
Using Advanced MPI

Modern Features of the Message-Passing Interface

William Gropp
Torsten Hoefler
Rajeev Thakur
Ewing Lusk

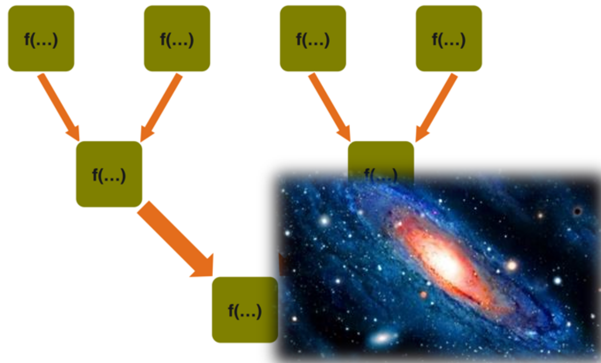


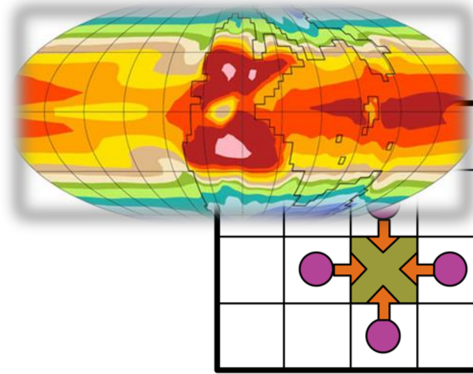
How to implement producer/consumer in passive mode?

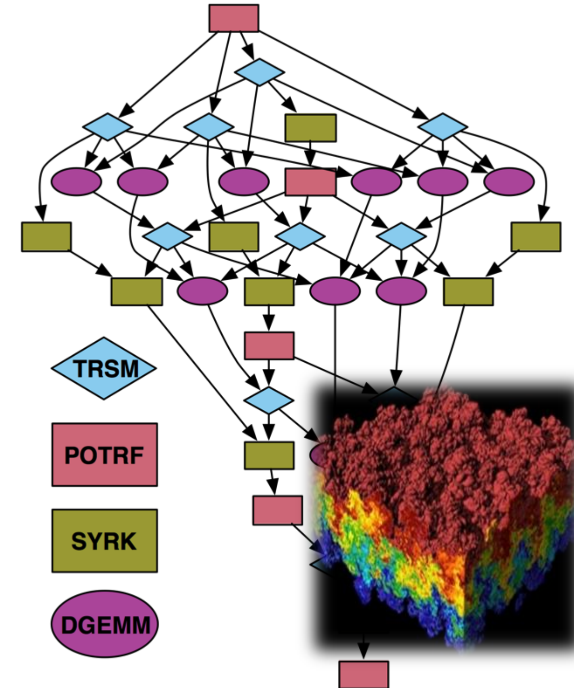


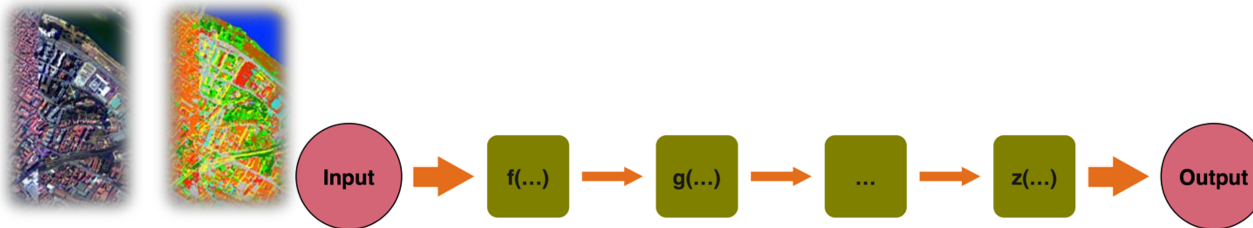
PRODUCER-CONSUMER RELATIONS

- **Most important communication idiom**
  - Some examples:





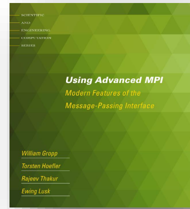




- **Perfectly supported by MPI-1 Message Passing**
  - But how does this actually work over RDMA?

Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

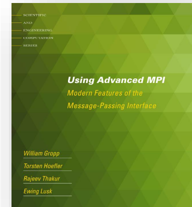
ONE SIDED – PUT + SYNCHRONIZATION



Producer

Consumer

Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15



One Sided – Put + Synchronization

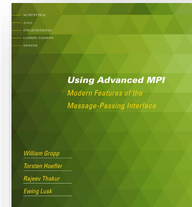
Producer

Consumer

Put

1. Data transfer

Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15



One Sided – Put + Synchronization

Producer

Consumer

Put

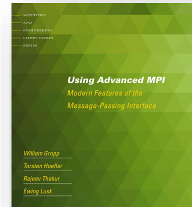
1. Data transfer

Flush

2. Producer waits for
remote completion

: origin aware of completion

Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15



One Sided – Put + Synchronization

Producer

Consumer

Put

Flush

1. Data transfer

2. Producer waits for remote completion

Explicit Synch

3. Producer reports completion to consumer

Explicit Synch

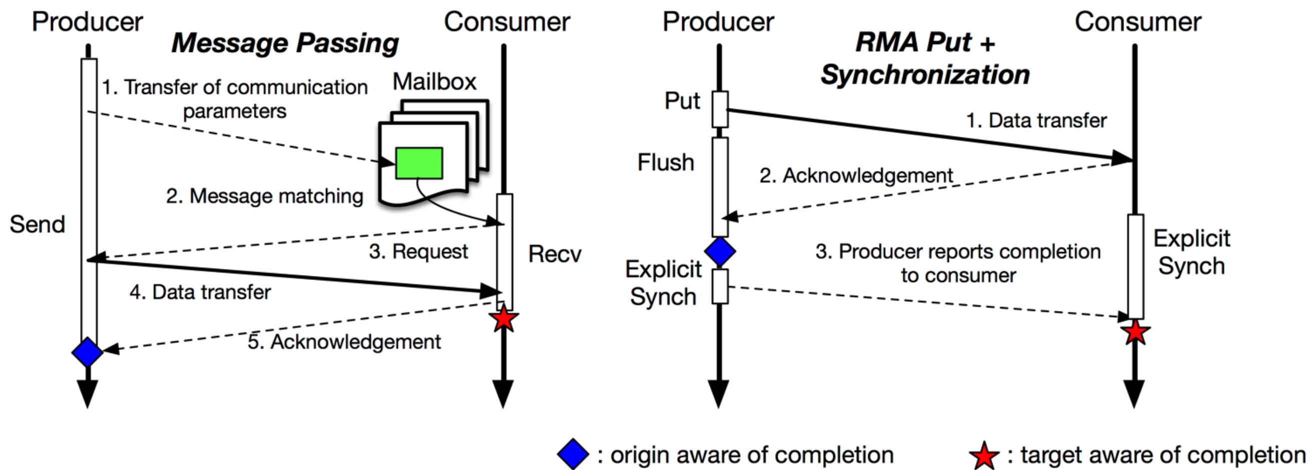
★ : target aware of completion

◆ : origin aware of completion

Critical path: 3 latencies

Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

COMPARING APPROACHES



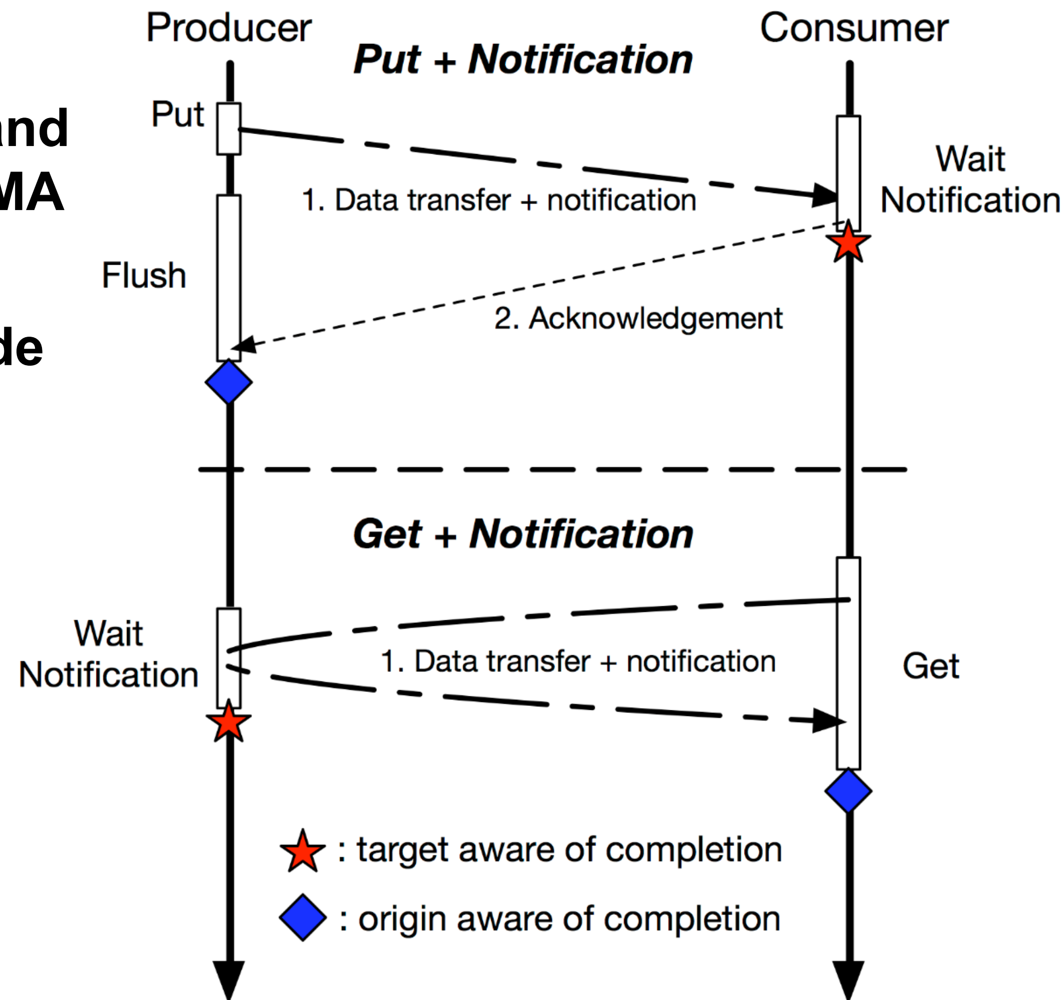
Message Passing
1 latency + copy / 3 latencies

One Sided
3 latencies

Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

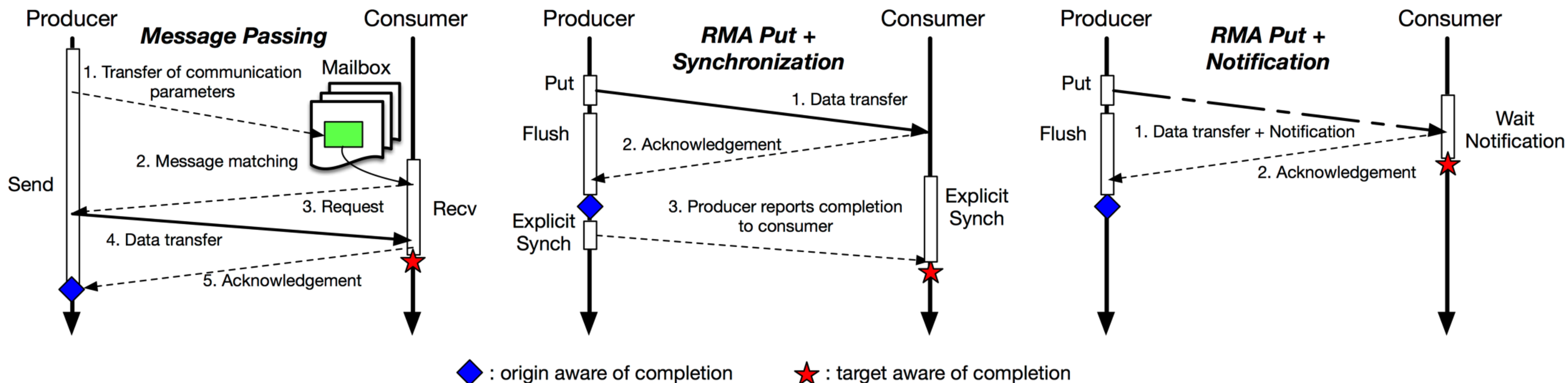
IDEA: RMA NOTIFICATIONS

- **First seen in Split-C (1992)**
- **Combine communication and synchronization using RDMA**
- **RDMA networks can provide various notifications**
  - Flags
  - Counters
  - Event Queues



Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

COMPARING APPROACHES



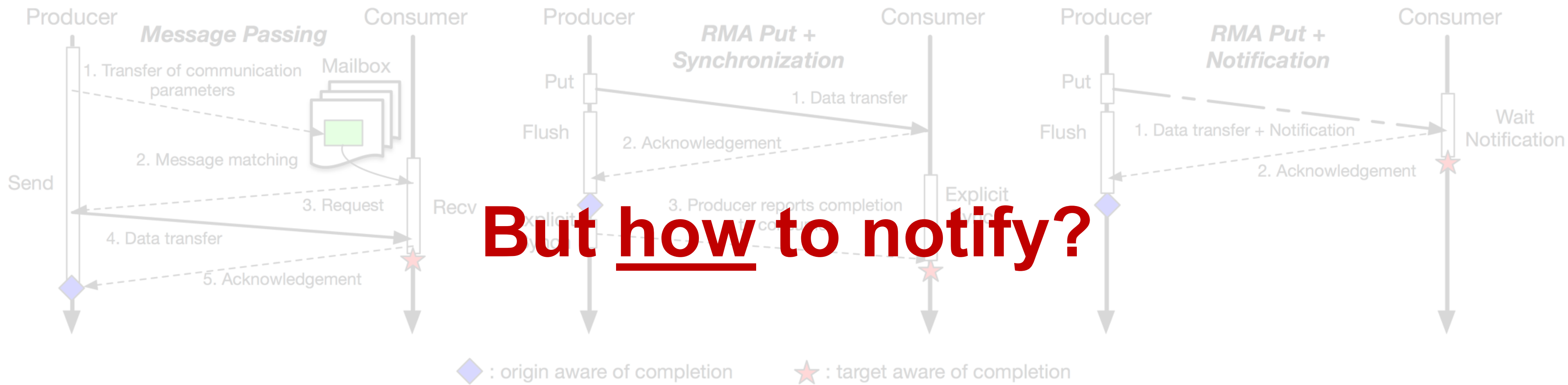
Message Passing
1 latency + copy / 3 latencies

One Sided
3 latencies

Notified Access
1 latency

Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

Comparing Approaches



But how to notify?

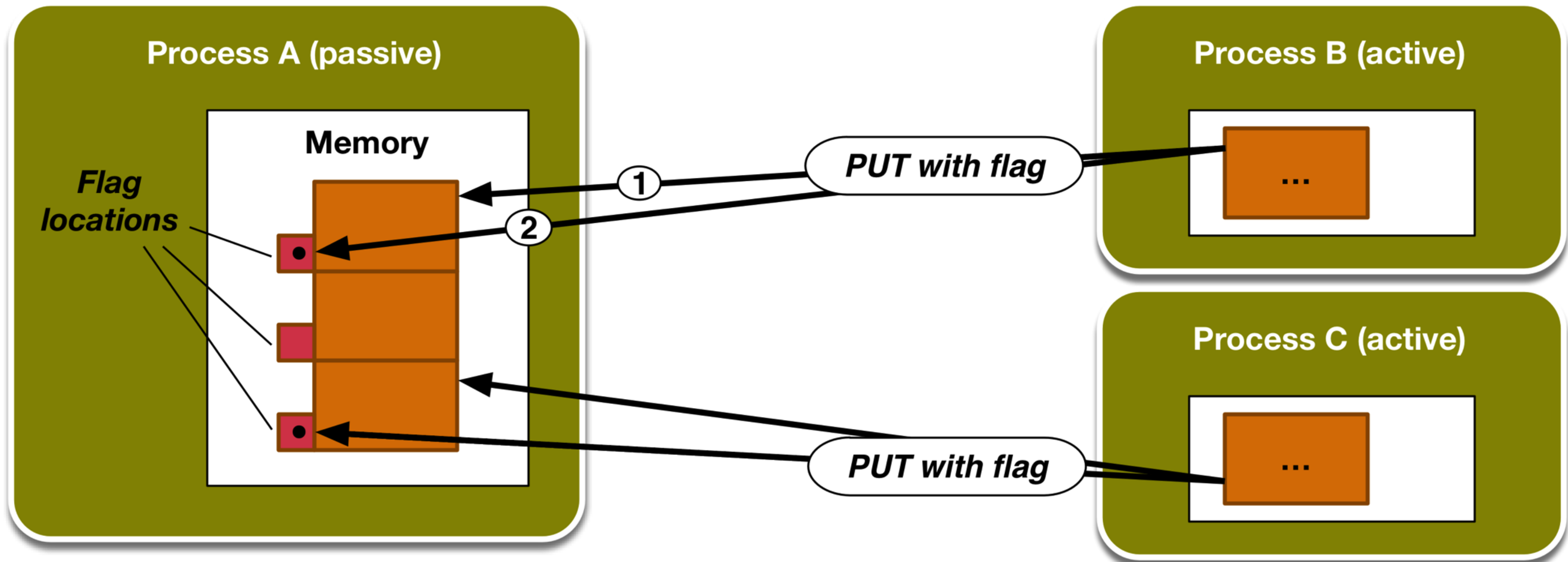
Message Passing
1 latency + copy / 3 latencies

One Sided
3 latencies

Notified Access
1 latency

Previous work: Overwriting Interface

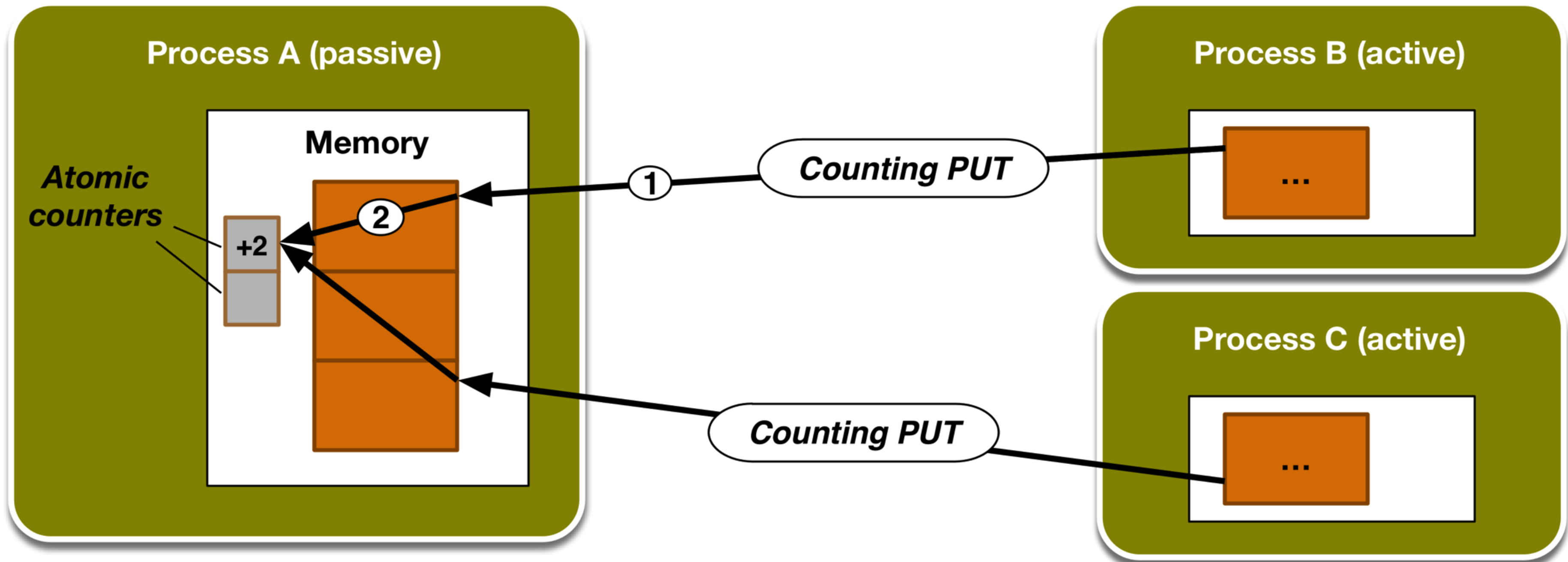
- **Flags (polling at the remote side)**
  - Used in *GASPI, DMAPP, NEON*



- **Disadvantages**
  - Location of the flag chosen at the sender side
  - Consumer needs at least one flag for every process
  - Polling a high number of flags is inefficient

Previous work: Counting Interface

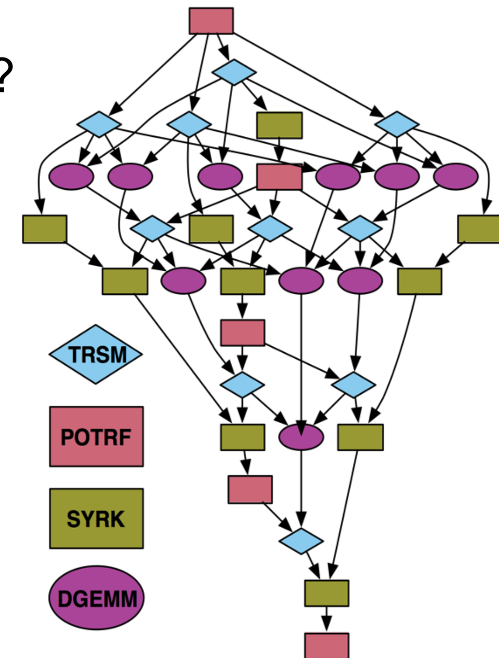
- **Atomic counters (accumulate notifications → scalable)**
  - Used in *Split-C, LAPI, SHMEM - Counting Puts, …*



- **Disadvantages**
  - Dataflow applications may require many counters
  - High polling overhead to identify accesses
  - Does not preserve order (may not be linearizable)

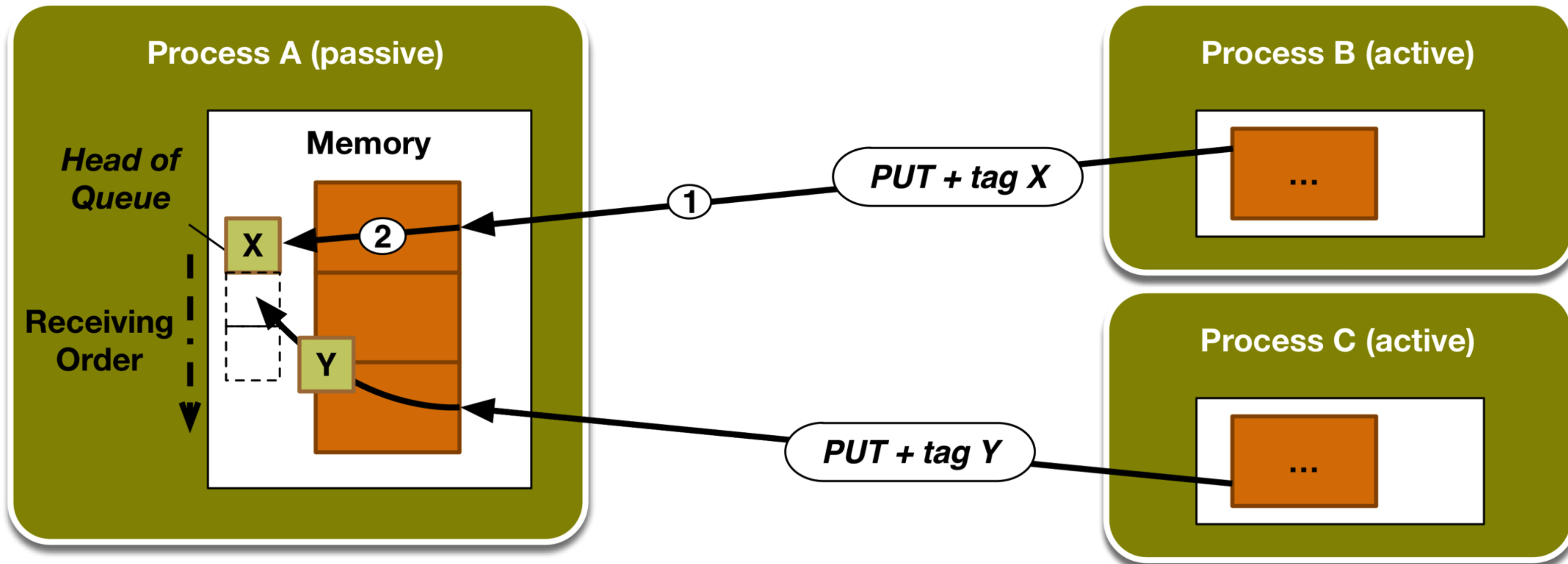
What is a Good Notification Interface?

- **Scalable to yotta-scale**
  - Does memory or polling overhead grow with # of processes?
- **Computation/communication overlap**
  - Do we support maximum asynchrony? (better than MPI-1)
- **Complex data flow graphs**
  - Can we distinguish between different accesses locally?
  - Can we avoid starvation?
  - What about load balancing?
- **Ease-of-use**
  - Does it use standard mechanisms?



Our Approach: Notified Access

- **Notifications with MPI-1 (queue-based) matching**
  - Retains benefits of previous notification schemes
  - Poll only head of queue
  - Provides linearizable semantics



Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

NOTIFIED ACCESS – AN MPI INTERFACE

- **Minor interface evolution**
  - Leverages MPI two sided <source, tag> matching
  - Wildcards matching with FIFO semantics

Example Communication Primitives

```
int MPI_Put       (void *origin_addr, int origin_count, MPI_Datatype origin_type, int target_rank,
                   MPI_Aint target_disp, int target_count, MPI_Datatype target_type, MPI_Win win);

int MPI_Get       (void *origin_addr, int origin_count, MPI_Datatype origin_type, int target_rank,
                   MPI_Aint target_disp, int target_count, MPI_Datatype target_type, MPI_Win win);
```

Example Synchronization Primitives

```
/*Functions already available in MPI*/
int MPI_Start(MPI_Request *request);
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status);
int MPI_Wait(MPI_Request *request, MPI_Status *status);
```

Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

Notified Access – An MPI Interface

- **Minor interface evolution**
  - Leverages MPI two sided <source, tag> matching
  - Wildcards matching with FIFO semantics

Example Communication Primitives

```
int MPI_Put_notify(void *origin_addr, int origin_count, MPI_Datatype origin_type, int target_rank,
                   MPI_Aint target_disp, int target_count, MPI_Datatype target_type, MPI_Win win,
                   int tag);
int MPI_Get_notify(void *origin_addr, int origin_count, MPI_Datatype origin_type, int target_rank,
                   MPI_Aint target_disp, int target_count, MPI_Datatype target_type, MPI_Win win,
                   int tag);
```

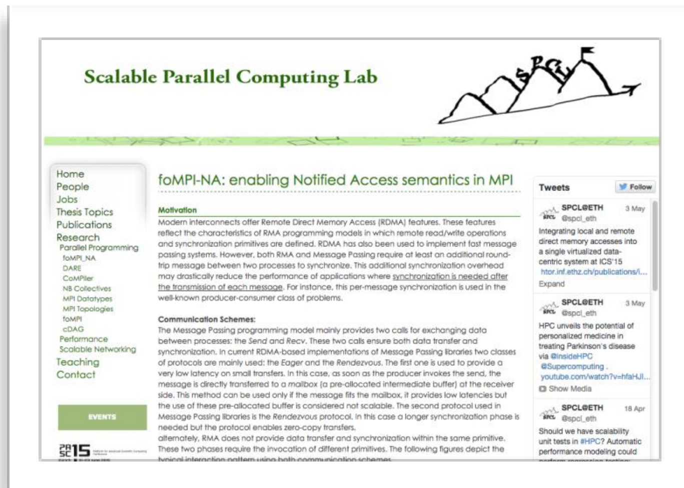
Example Synchronization Primitives

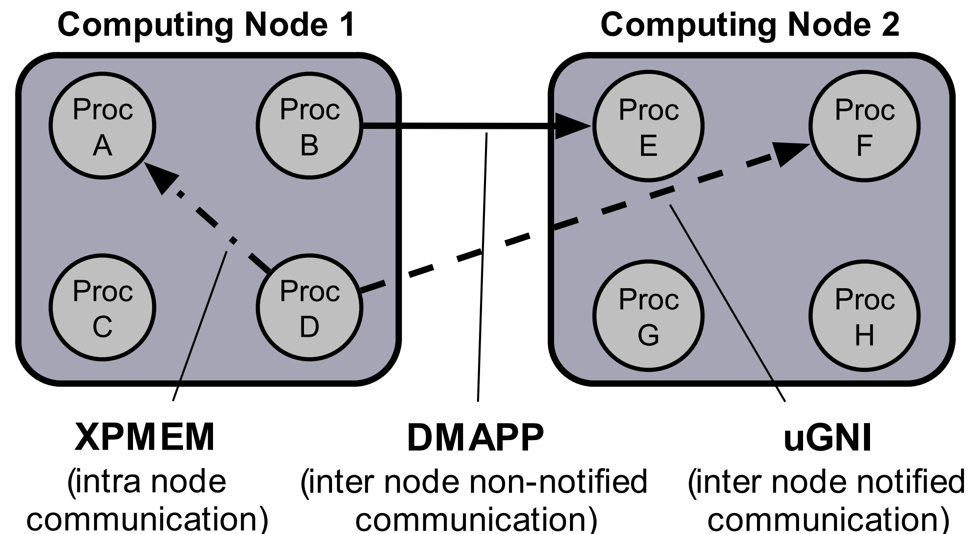
```
int MPI_Notify_init(MPI_Win win, int src_rank, int tag, int expected_count, MPI_Request *request);
/*Functions already available in MPI*/
int MPI_Start(MPI_Request *request);
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status);
int MPI_Wait(MPI_Request *request, MPI_Status *status);
```

Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

Notified Access - Implementation

- **foMPI – a fully functional MPI-3 RMA implementation**
  - Runs on newer Cray machines (Aries, Gemini)
  - DMAPP: low-level networking API for Cray systems
  - XPMEM: a portable Linux kernel module
- **Implementation of Notified Access via uGNI [1]**
  - Leverages uGNI queue semantics
  - Adds unexpected queue
  - Uses 32-bit immediate value to encode source and tag





[1] http://spcl.inf.ethz.ch/Research/Parallel_Programming/foMPI_NA/

EXPERIMENTAL SETTING

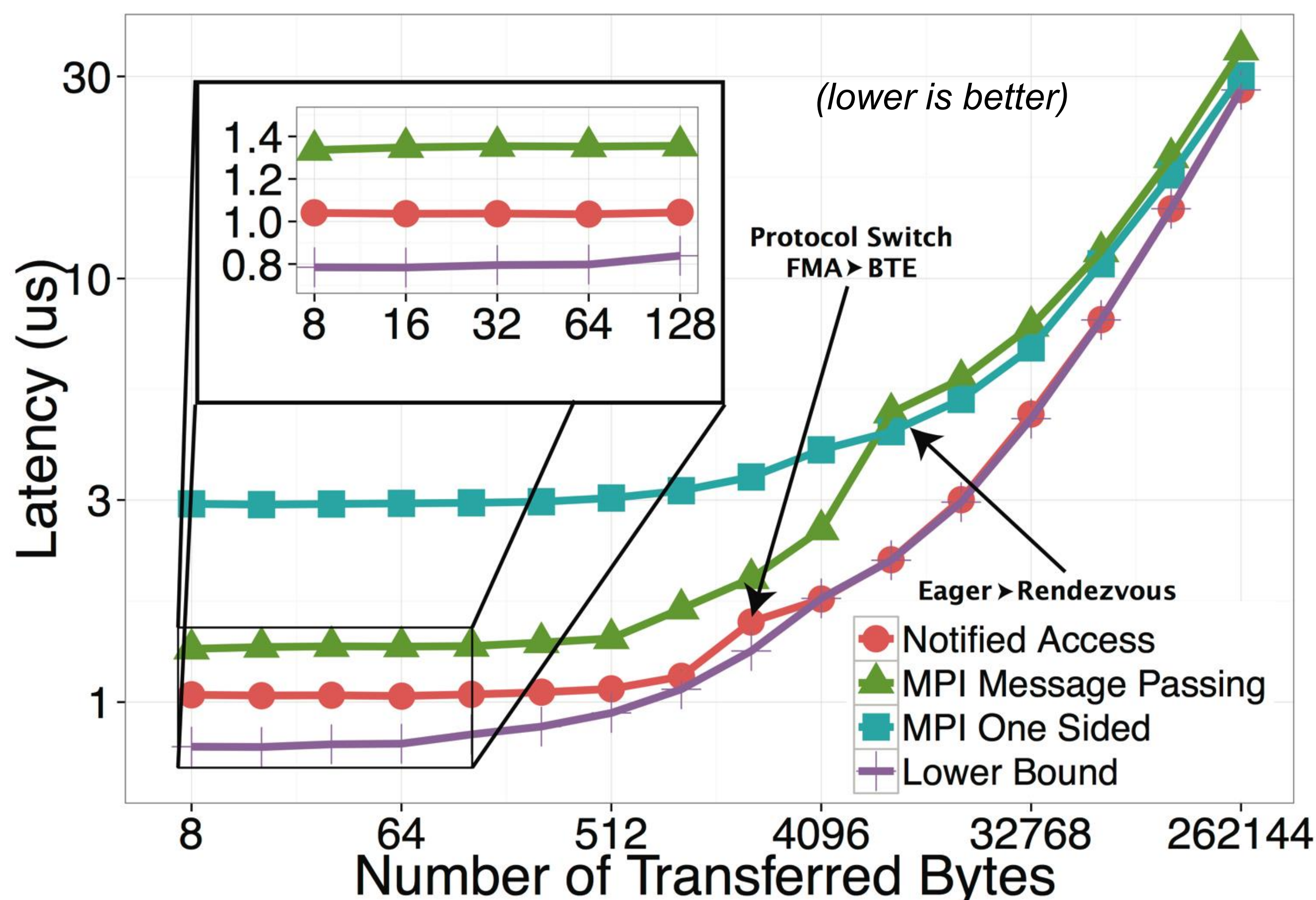
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

- **Piz Daint**
  - Cray XC30, Aries interconnect
  - 5'272 computing nodes (Intel Xeon E5-2670 + NVIDIA Tesla K20X)
  - Theoretical Peak Performance 7.787 Petaflops
  - Peak Network Bisection Bandwidth 33 TB/s

[1] http://www.cscs.ch

Ping Pong Performance (Inter-Node)

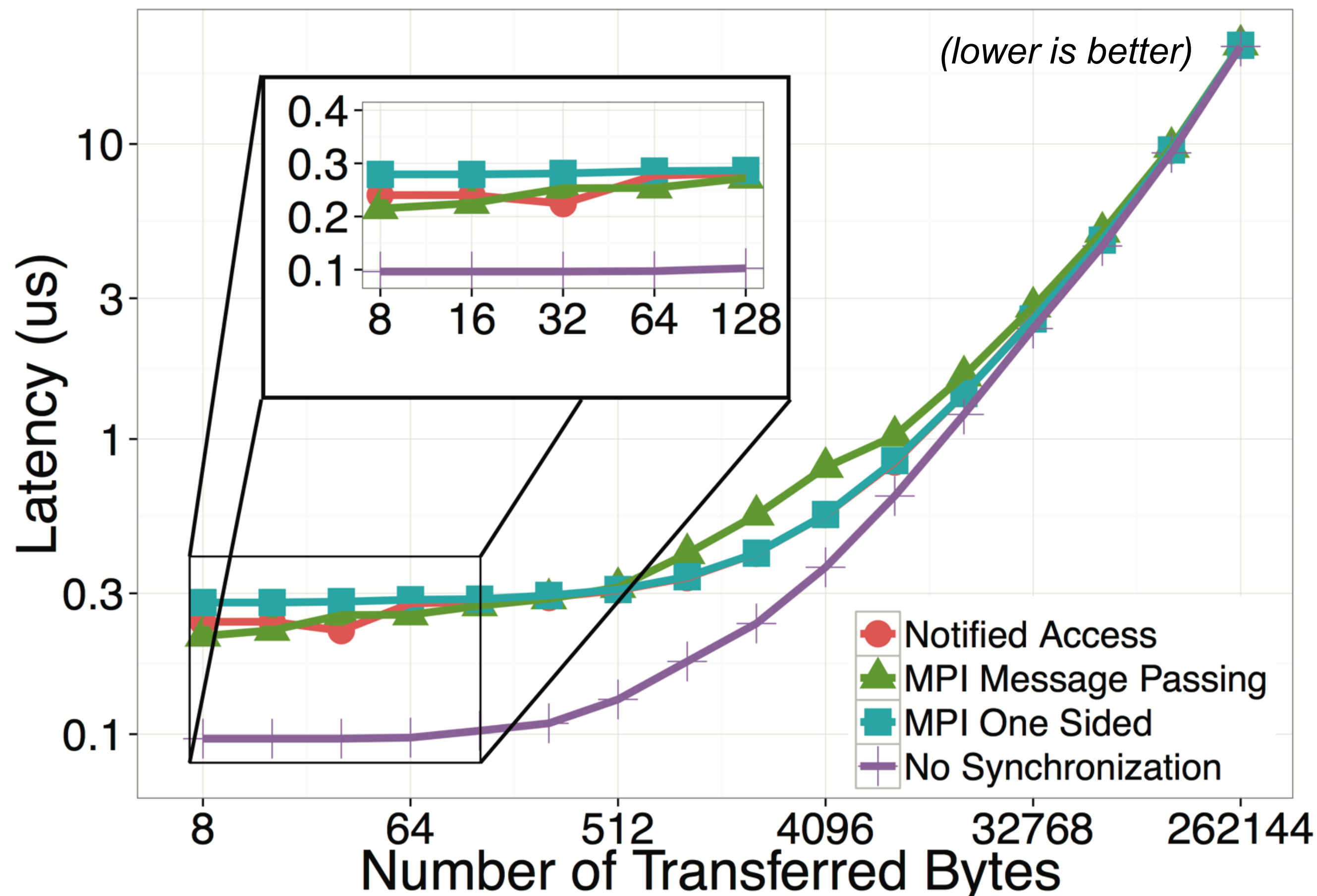
- **1000 repetitions, each timed separately, RDTSC timer**
- **95% confidence interval always within 1% of median**



Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

PING PONG PERFORMANCE (INTRA-NODE)

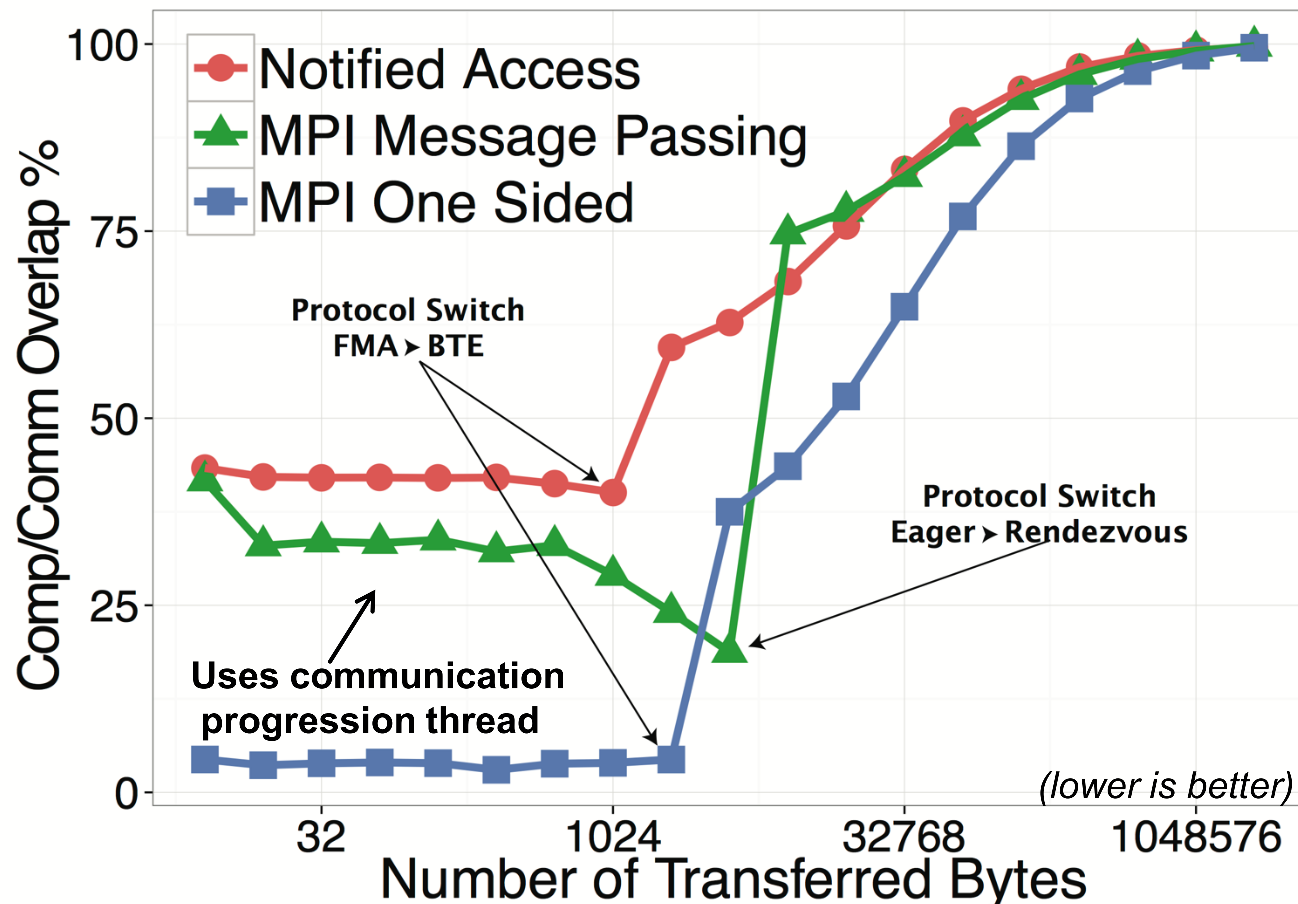
- **1000 repetitions, each timed separately, RDTSC timer**
- **95% confidence interval always within 1% of median**



Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

Computation/Communication Overlap

- **1000 repetitions, each timed separately, RDTSC timer**
- **95% confidence interval always within 1% of median**



Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15

PIPELINE – ONE-TO-ONE SYNCHRONIZATION

- 1000 repetitions, each timed separately, RDTSC timer
- 95% confidence interval always within 1% of median

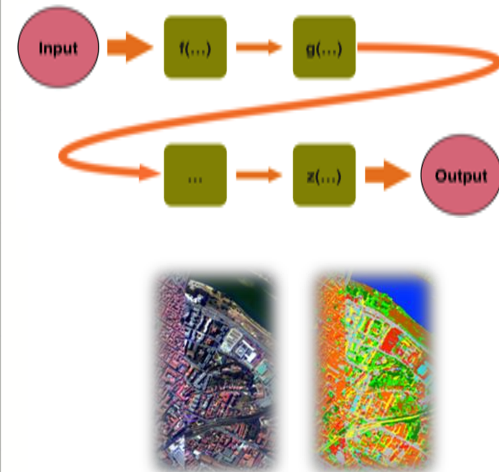
MPI Message Passing
MPI One Sided
Notified Access

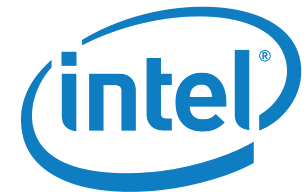
Normalized Completion Time
3
2
1

(lower is better)

1.1x
1.32x
1.43x
1.52x
1.58x
2.17x

4
8
16
32
64
128
Number of Processes

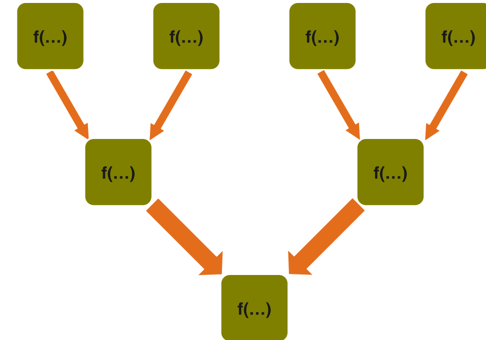




[1] https://github.com/intelesg/PRK2

Reduce – One-to-Many Synchronization

- **Reduce as an example (same for FMM, BH, etc.)**
  - Small data (8 Bytes), 16-ary tree
  - 1000 repetitions, each timed separately with RDTSC



Notified Access
MPI Message Passing
MPI One Sided PSCW
MPI Reduce

Completion Time (us)

Number of Processes

(lower is better)

CRAY
Supercomputer

Cholesky – Many-to-Many Synchronization

- 1000 repetitions, each timed separately, RDTSC timer
- 95% confidence interval always within 10% of median

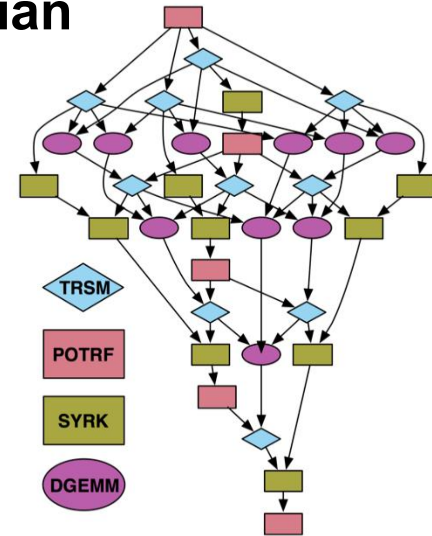
(Higher is better)

Notified Access
MPI Message Passing
MPI One Sided

GMOPS: 0.0, 0.2, 0.4, 0.6

Number of Processes: 4, 8, 16, 32, 64, 128

1.34x, 1.56x, 1.7x, 1.9x, 1.98x, 1.95x



[1]: J. Kurzak, H. Ltaief, J. Dongarra, R. Badia: "Scheduling dense linear algebra operations on multicore processors“, CCPE 2010

Discussion and Conclusions

- **Performance of cache-coherency is hard to model**
  - Min/max models
- **RDMA+SHM are de-facto hardware mechanisms**
  - Gives rise to RMA programming
- **MPI-3 RMA standardizes clear semantics**
  - Builds on existing practice (UPC, CAF, ARMCI etc.)
  - Rich set of synchronization mechanisms
- **Notified Access can support producer/consumer**
  - Maintains benefits of RDMA
- **Fully parameterized LogGP-like performance model**
  - Aids algorithm development and reasoning

applicable at least to:

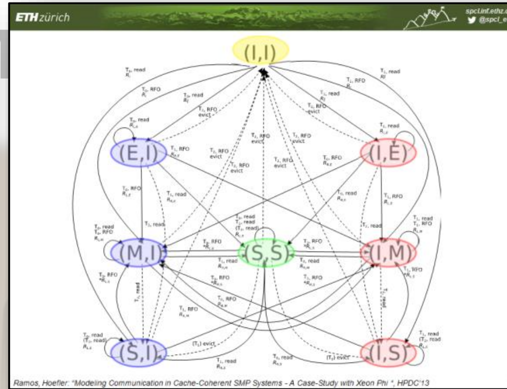
	Shared Memory	uGNI FMA	uGNI BTE
L	$0.25\mu s$	$1.02\mu s$	$1.32\mu s$
G	$0.08ns$	$0.105ns$	$0.101ns$

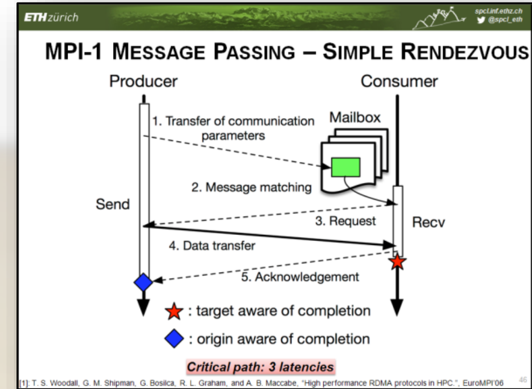
Function	Time
MPI_Notify_init	$t_{init} = 0.07\mu s$
MPI_Request_free	$t_{free} = 0.04\mu s$
MPI_Start	$t_{start} = 0.008\mu s$
MPI_{Put|Get}_notify	$t_{na} = 0.29\mu s$

ACKNOWLEDGMENTS

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





Performance Modeling

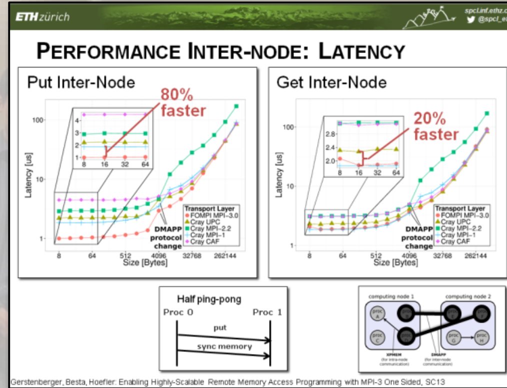
Performance functions for synchronization protocols

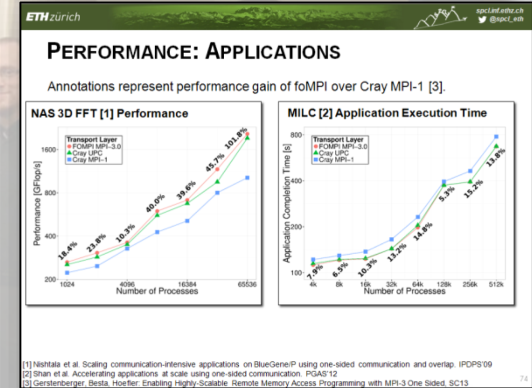
Fence	$\mathcal{P}_{fence} = 2.9\mu s \cdot \log_2(p)$
PSCW	$\mathcal{P}_{start} = 0.7\mu s, \mathcal{P}_{wait} = 1.8\mu s$ $\mathcal{P}_{post} = \mathcal{P}_{complete} = 350ns \cdot k$
Locks	$\mathcal{P}_{lock,excl} = 5.4\mu s$ $\mathcal{P}_{lock,shrd} = \mathcal{P}_{lock_all} = 2.7\mu s$ $\mathcal{P}_{unlock}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

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

Gerstenberger, Besta, Hoefler: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13



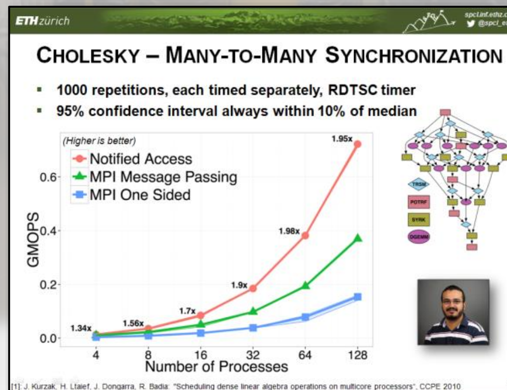


Idea: RMA Notifications

- First seen in Split-C (1992)
- Combine communication and synchronization using RDMA
- RDMA networks can provide various notifications
  - Flags
  - Counters
  - Event Queues

Producer — Consumer
Put + Notification
Put
1. Data transfer + notification
Wait Notification
Flush
2. Acknowledgement
Get + Notification
Wait Notification
1. Data transfer + notification
Get
: target aware of completion
: origin aware of completion

Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15



Scientific and Engineering Computation Series

Using Advanced MPI

Modern Features of the Message-Passing Interface

William Gropp
Torsten Hoefler
Rajeev Thakur
Ewing Lusk