

Towards Remote Memory Access Programming for Data Analytics

With R. Gerstenberger, M. Besta, R. Belli @ SPCL
presented at Lawrence Berkeley National Laboratory, Berkeley, CA, June 2015



2016
SC

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

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



MPI-1 MESSAGE PASSING – SIMPLE EAGER

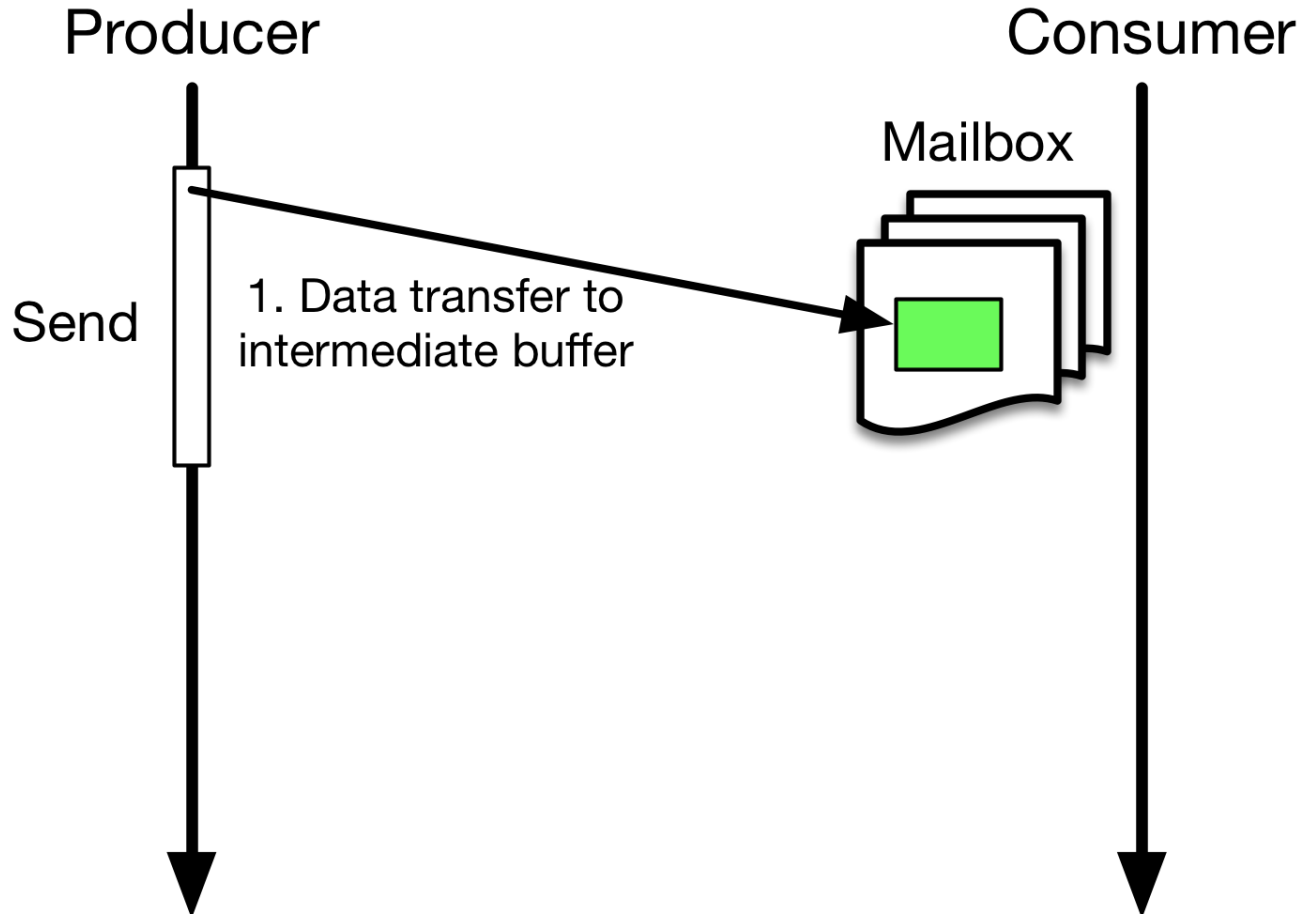
Producer



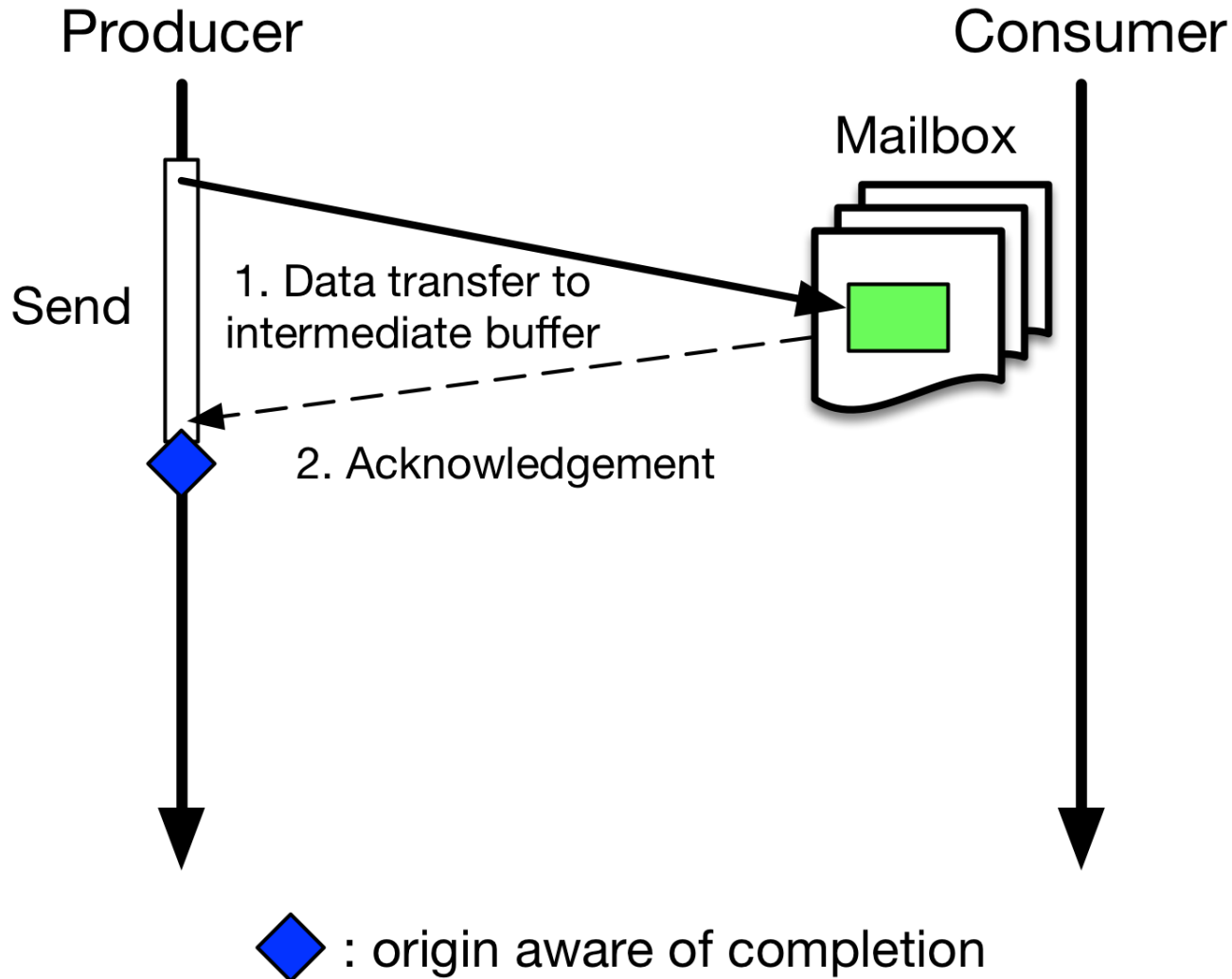
Consumer



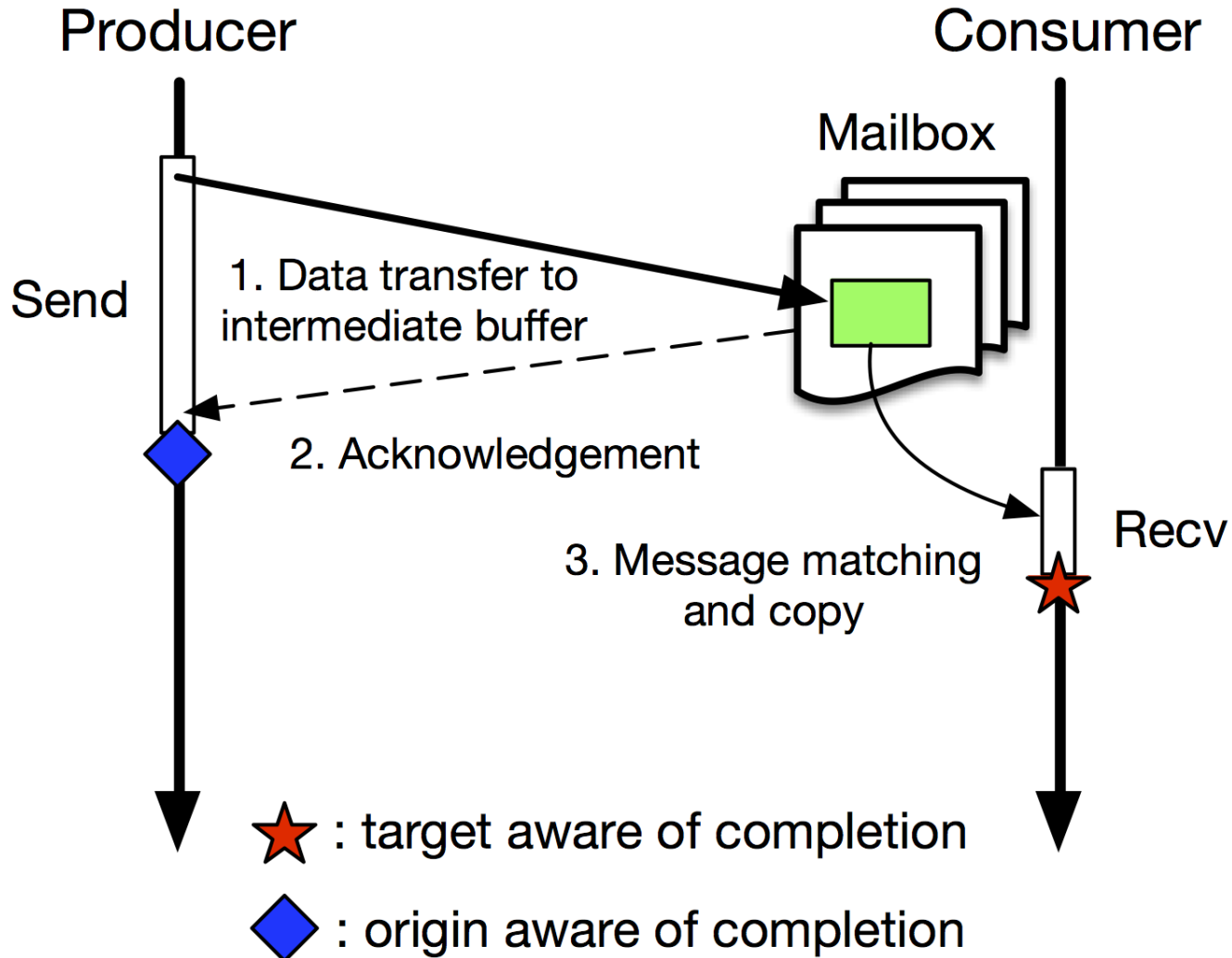
MPI-1 MESSAGE PASSING – SIMPLE EAGER



MPI-1 MESSAGE PASSING – SIMPLE EAGER



MPI-1 MESSAGE PASSING – SIMPLE EAGER



Critical path: 1 latency + 1 copy

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS

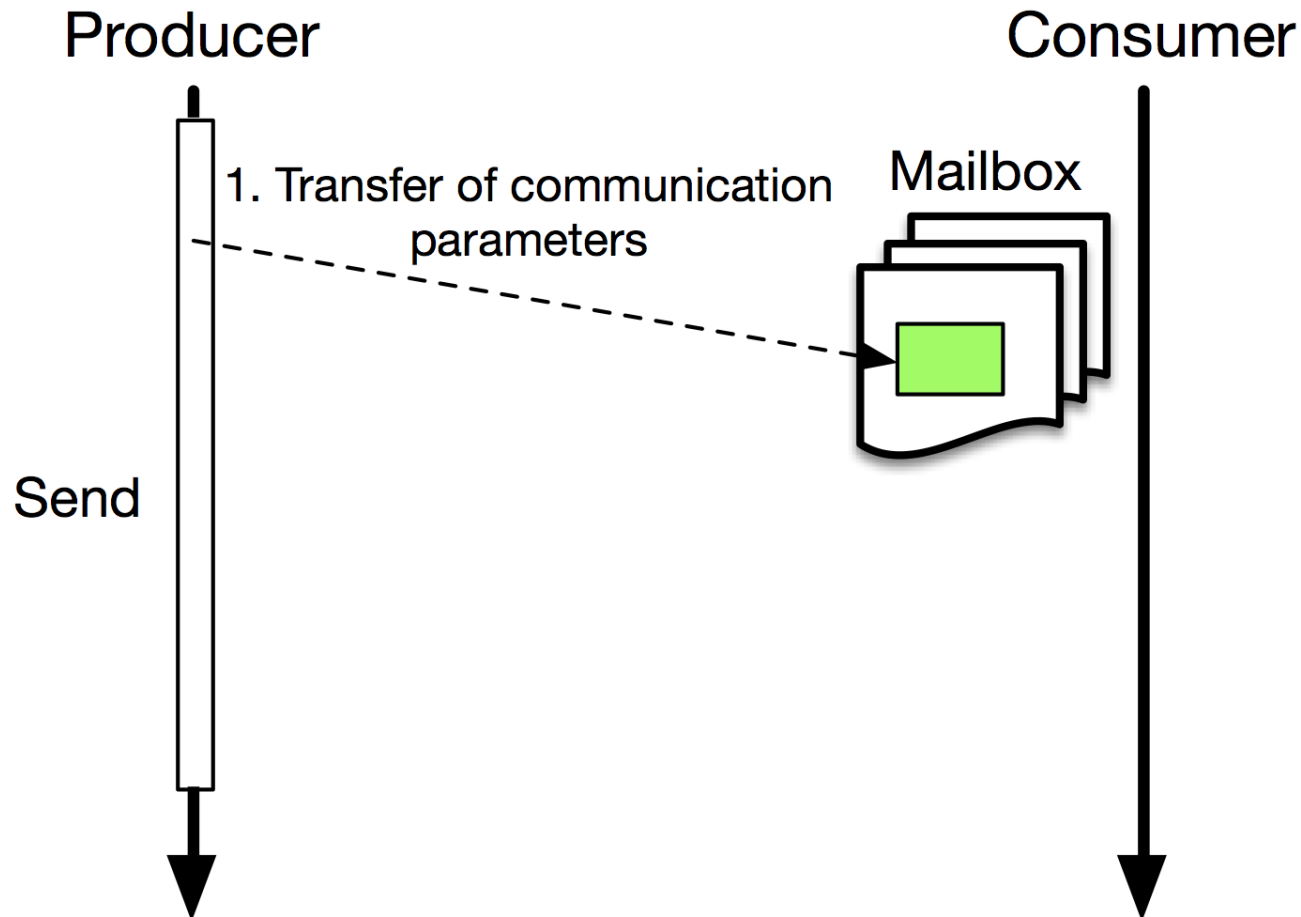
Producer



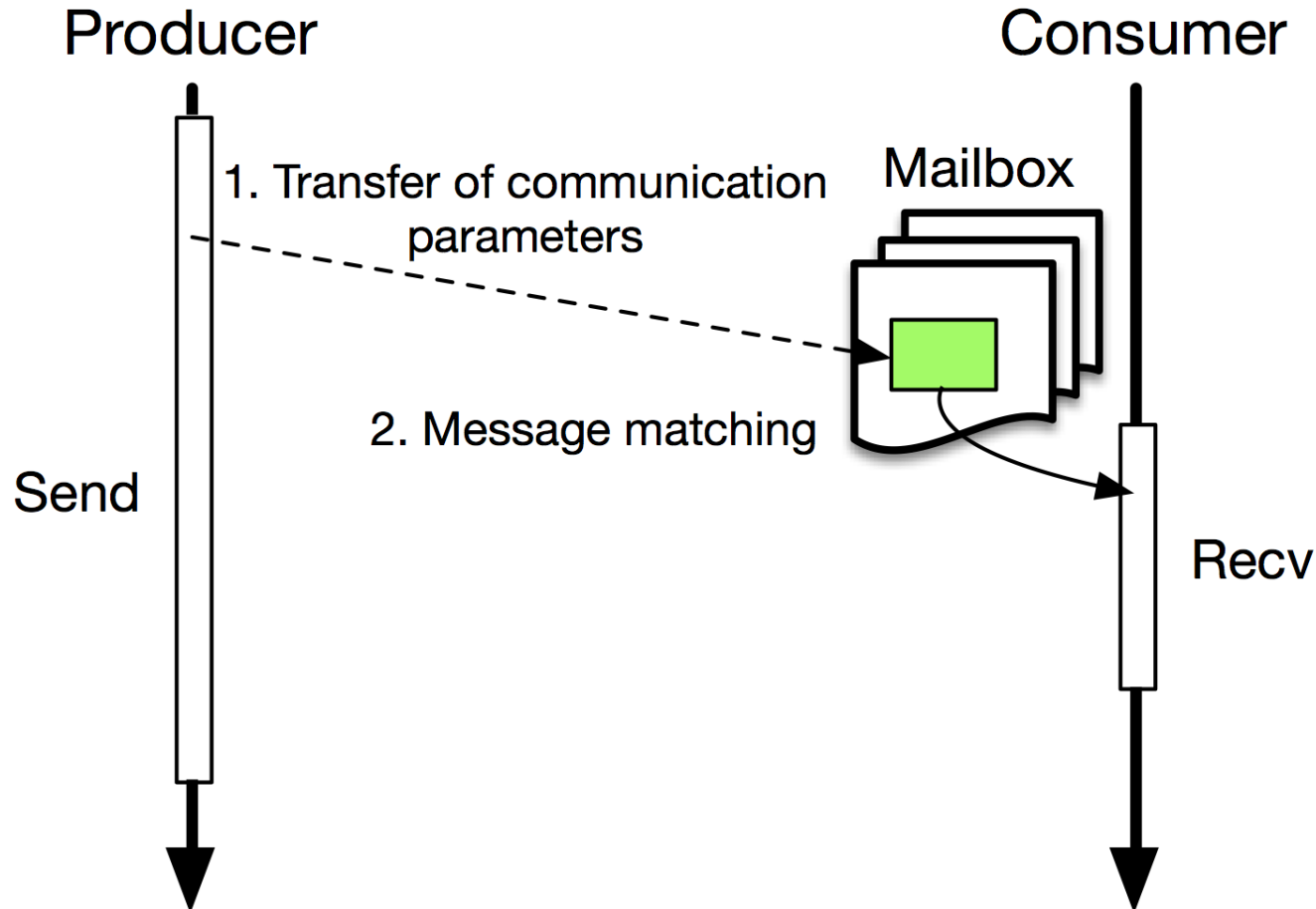
Consumer



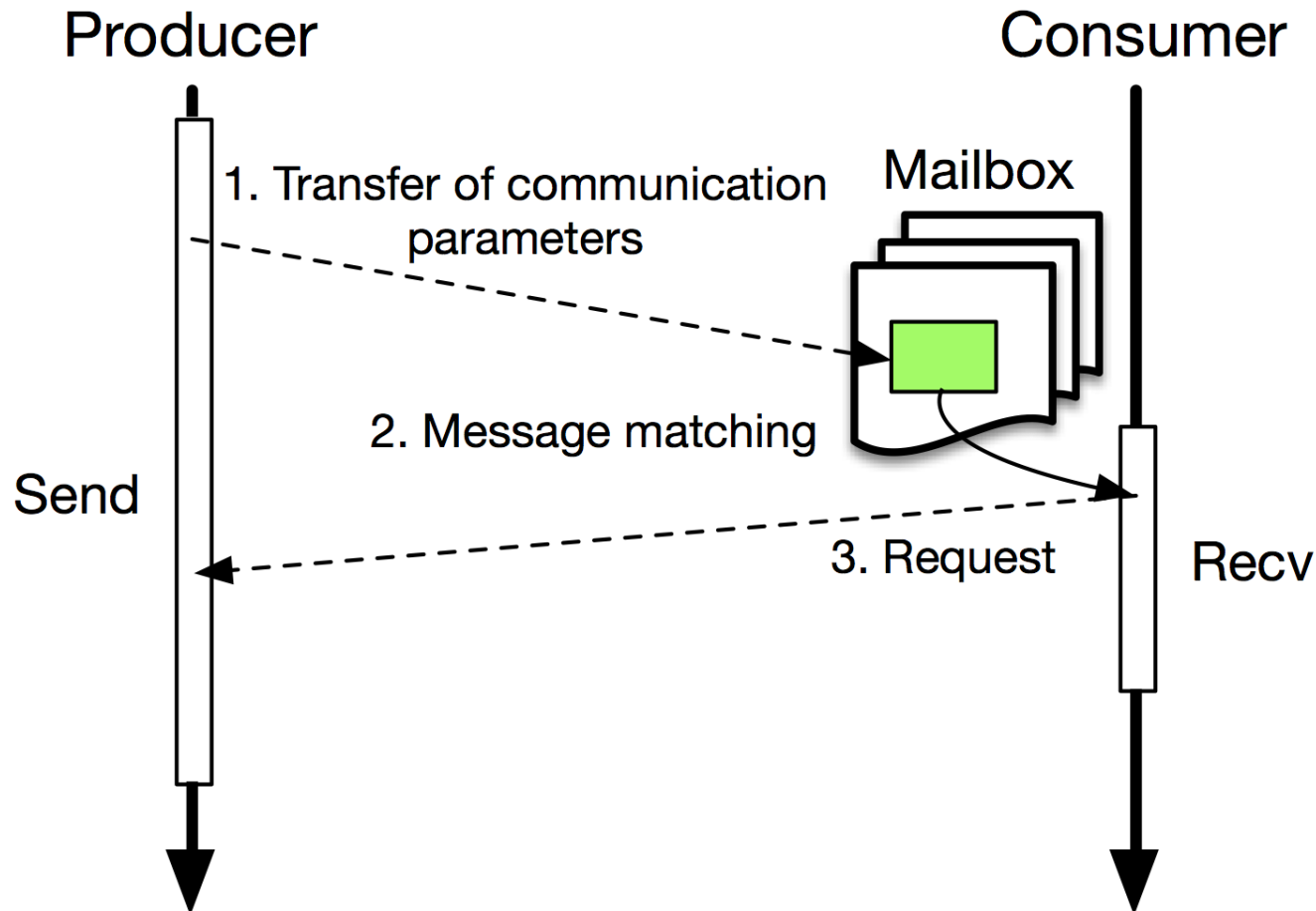
MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS



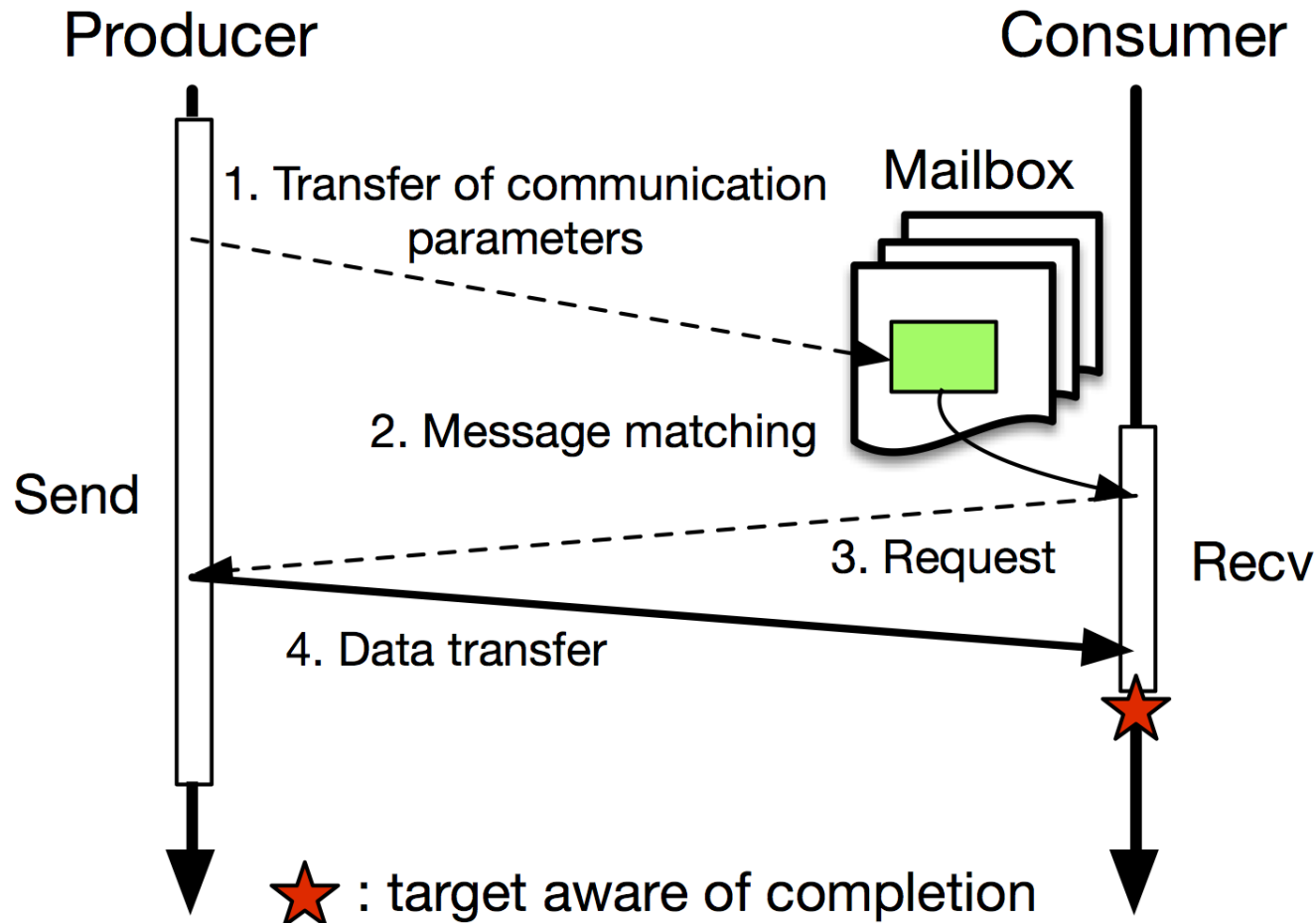
MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS



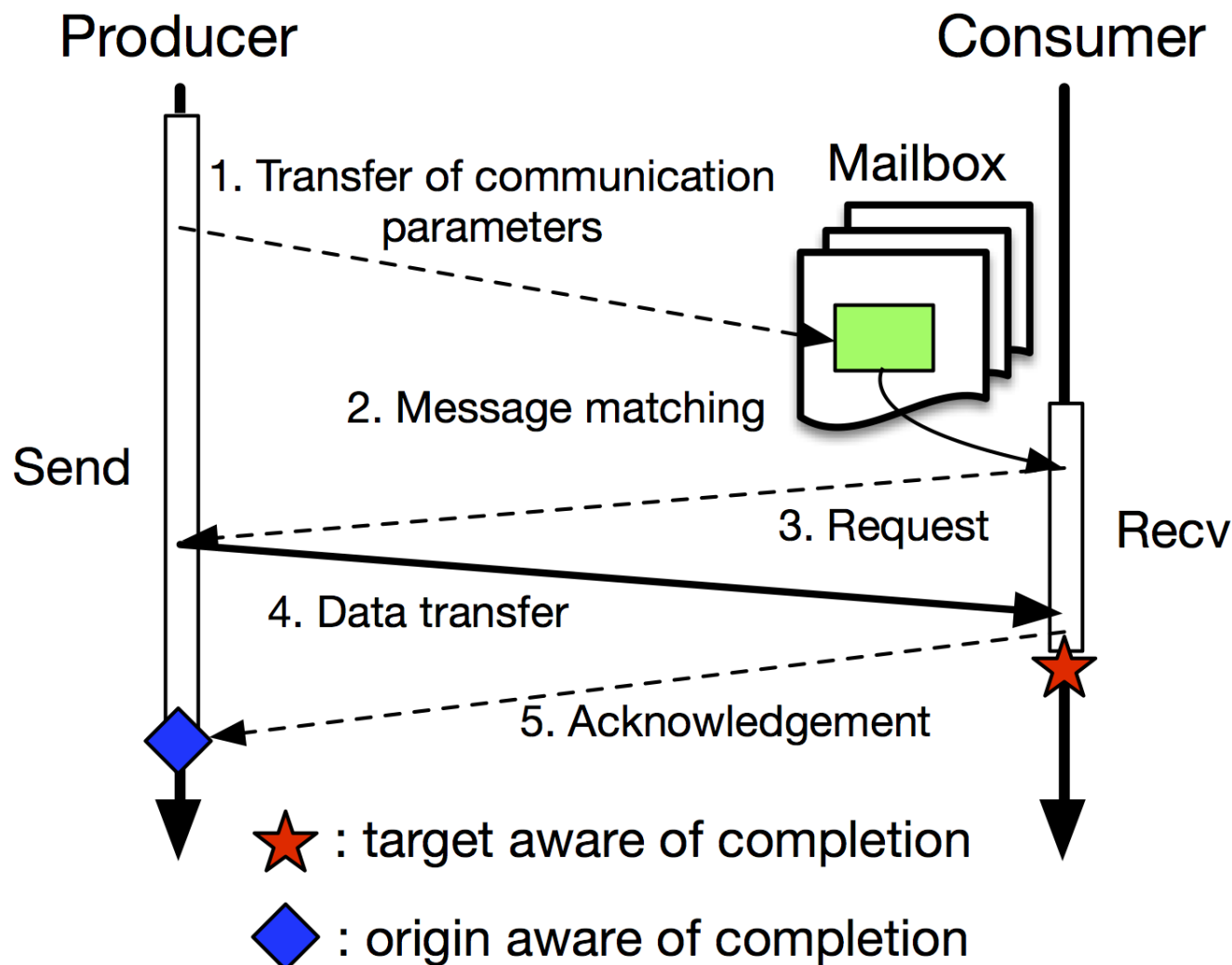
MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS



MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS



MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS



Critical path: 3 latencies

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



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”) [1] 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



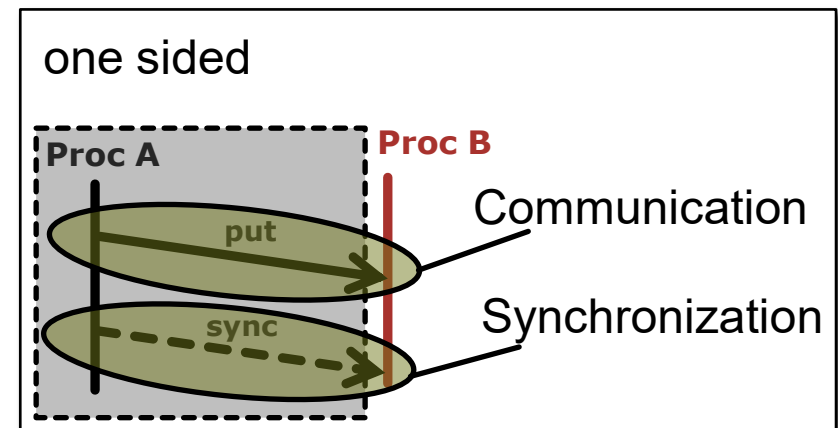
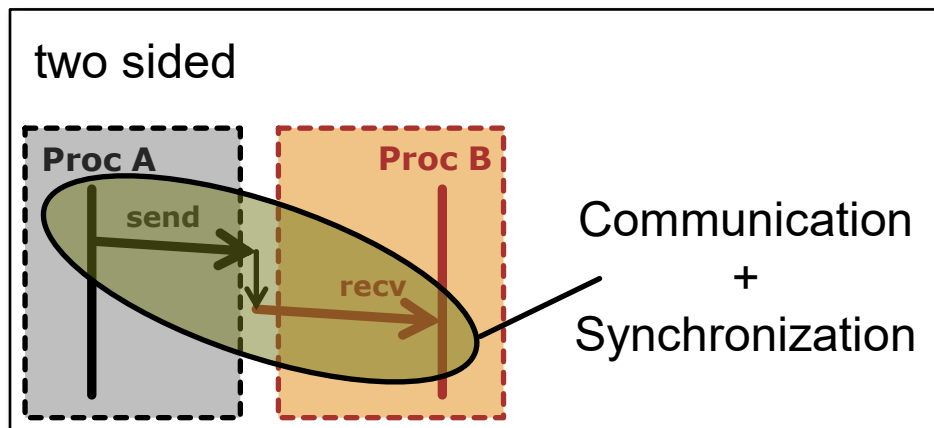
“Traditionally, HPC programming models are following hardware developments” (IPDPS’15)



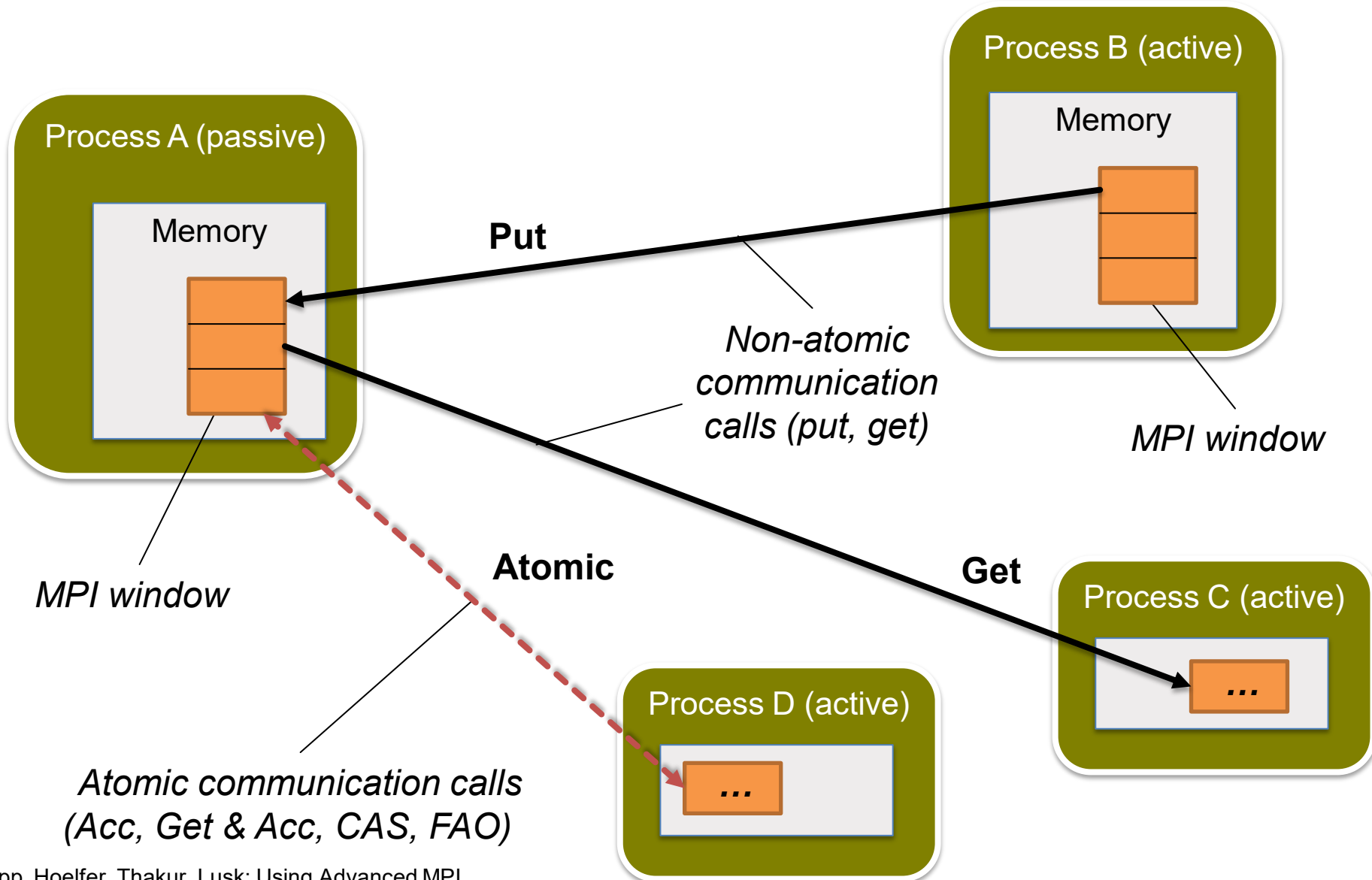
MPI-3 RMA SUMMARY

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

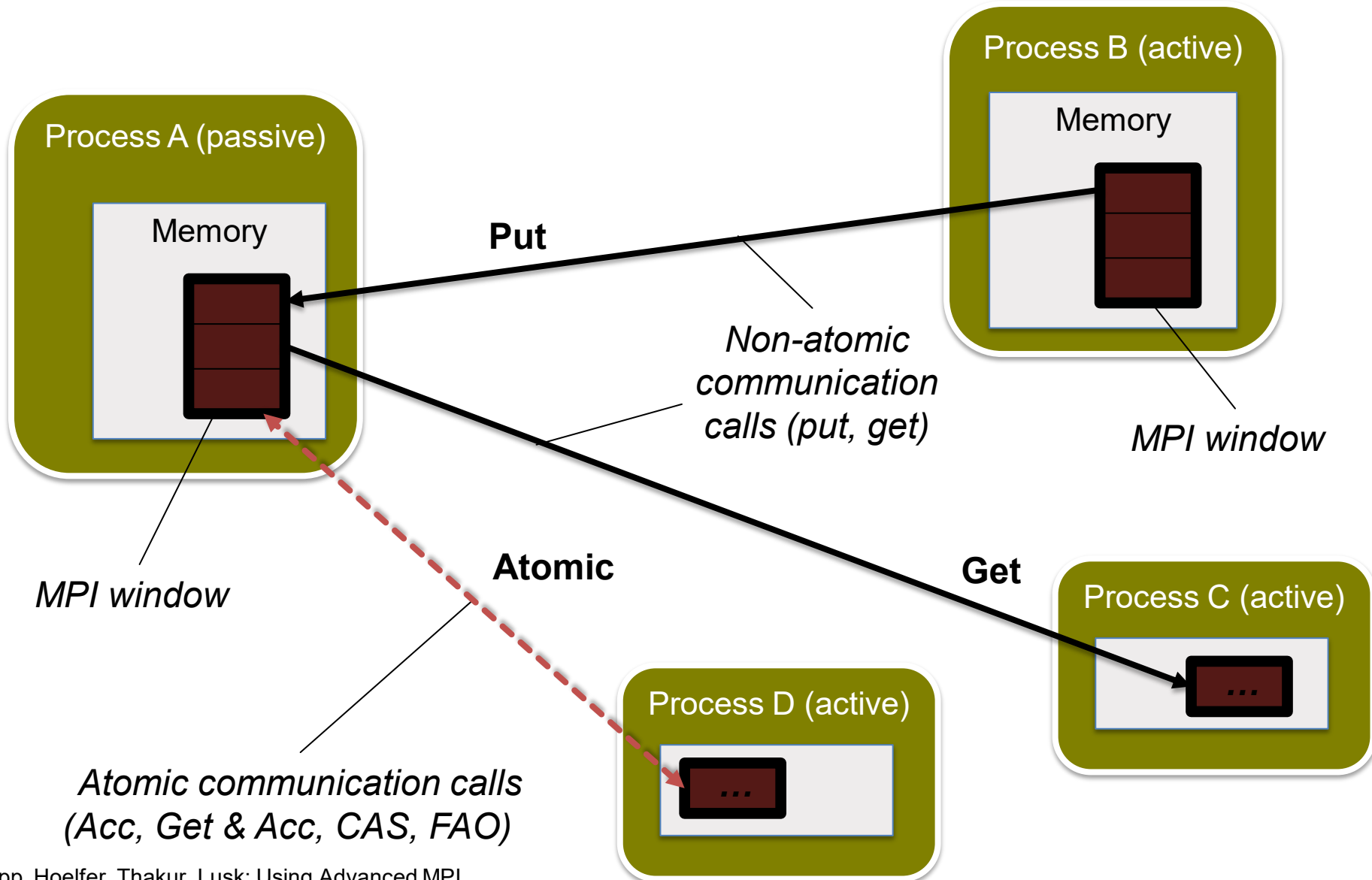
- **MPI-3 updates RMA (“MPI One Sided”)**
 - Significant change from MPI-2
- **Communication is „one sided” (no involvement of destination)**
 - Utilize direct memory access
- **RMA decouples communication & synchronization**
 - Fundamentally different from message passing



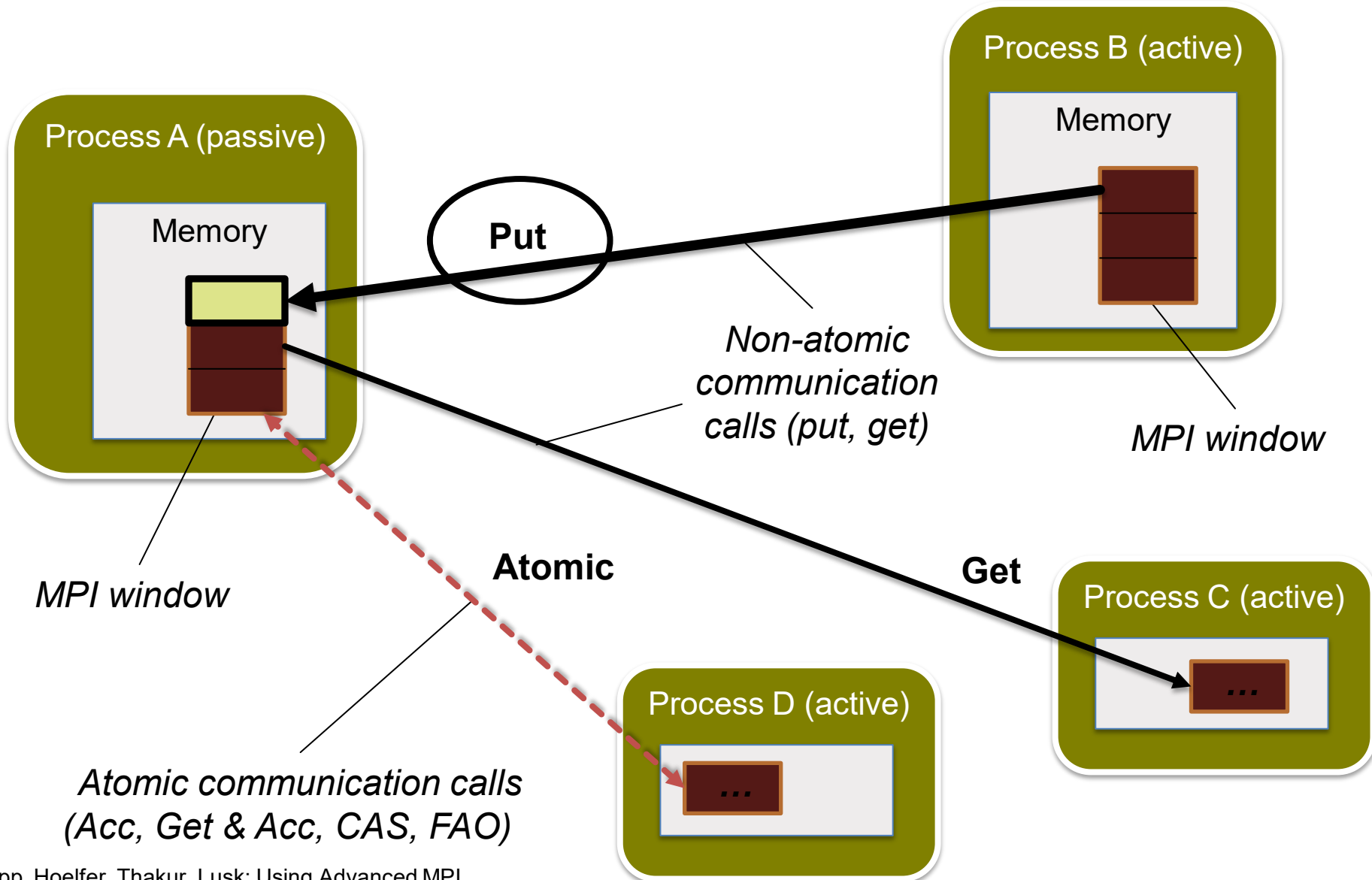
MPI-3 RMA COMMUNICATION OVERVIEW



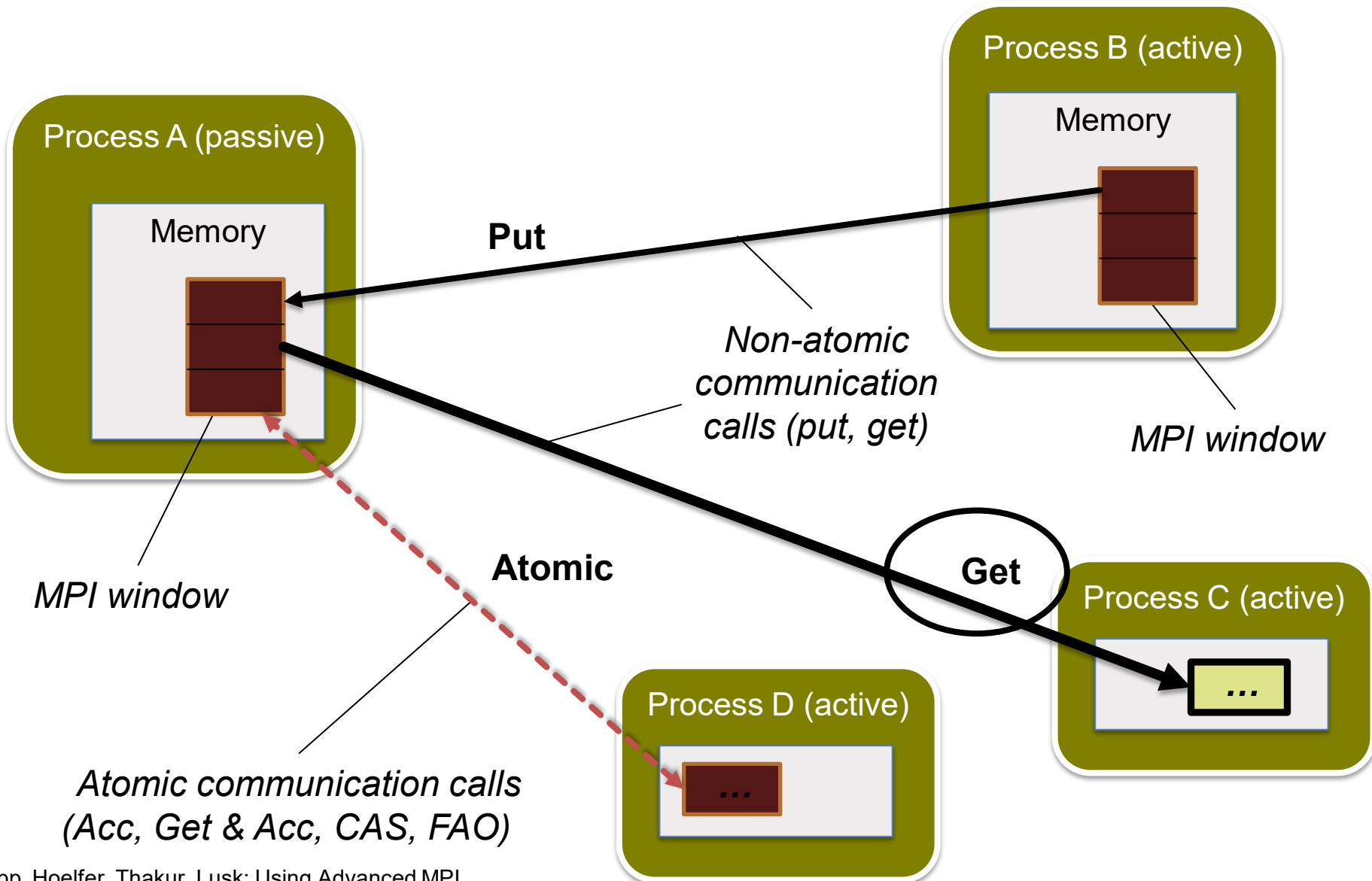
MPI-3 RMA COMMUNICATION OVERVIEW



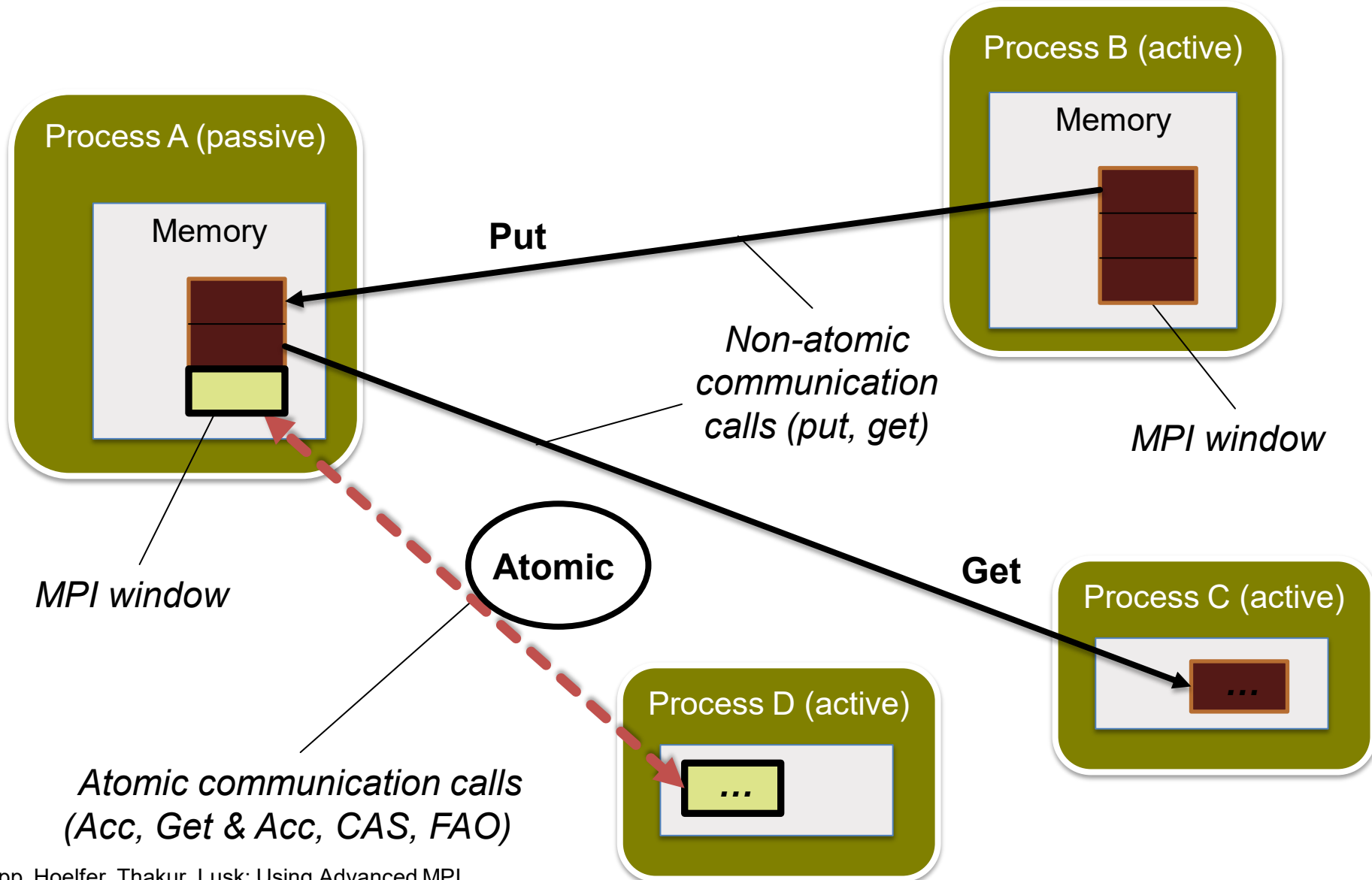
MPI-3 RMA COMMUNICATION OVERVIEW



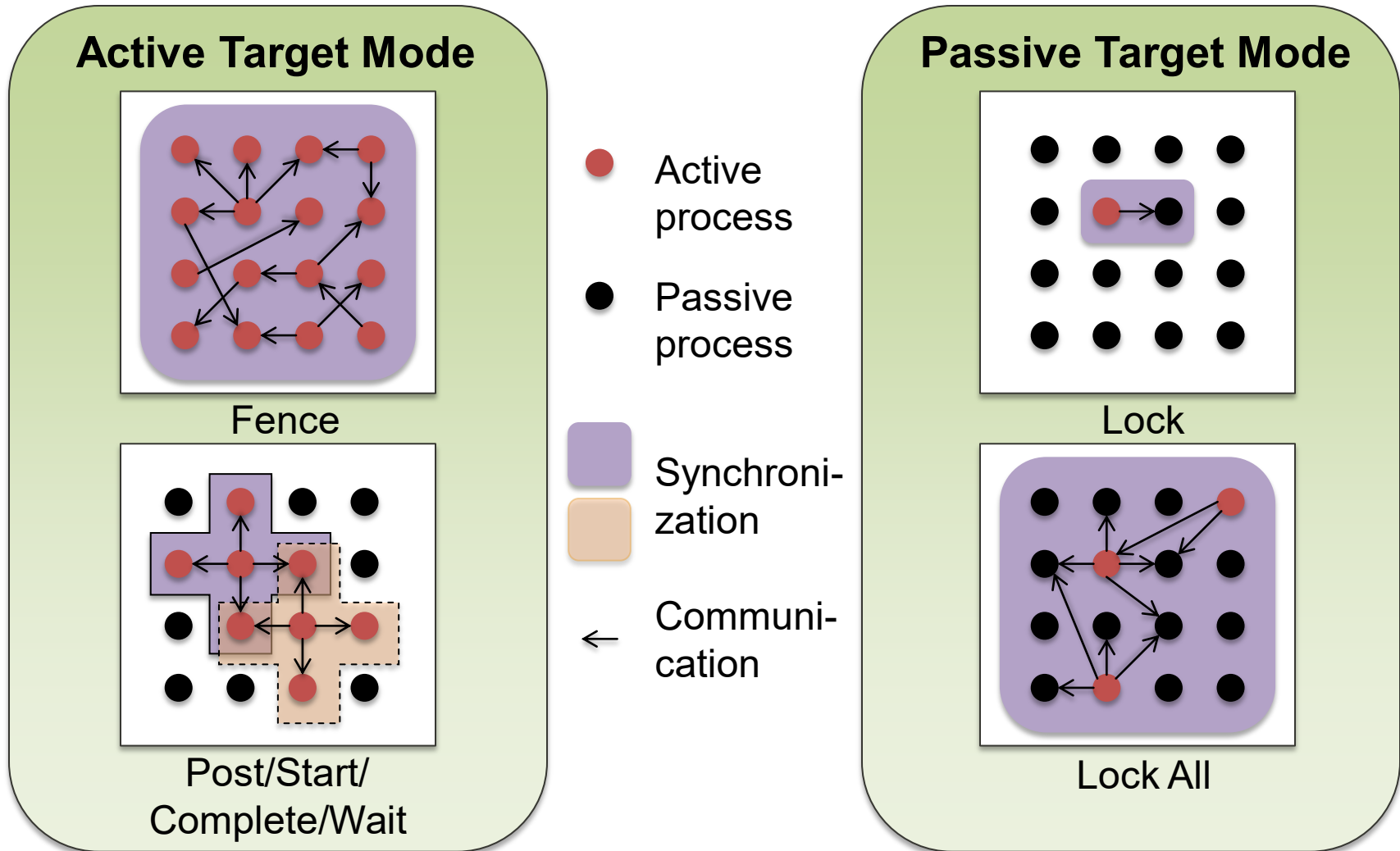
MPI-3 RMA COMMUNICATION OVERVIEW



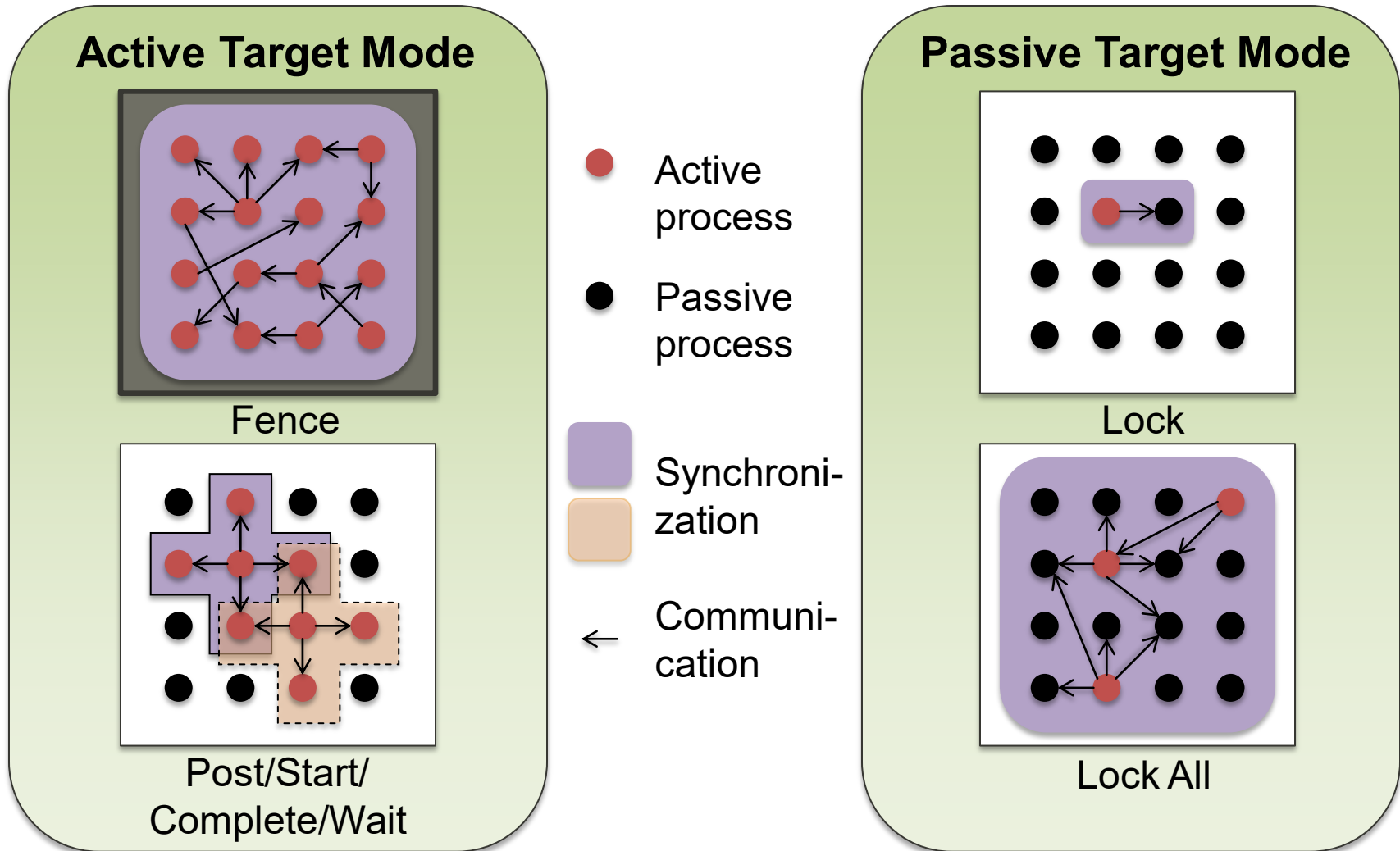
MPI-3 RMA COMMUNICATION OVERVIEW



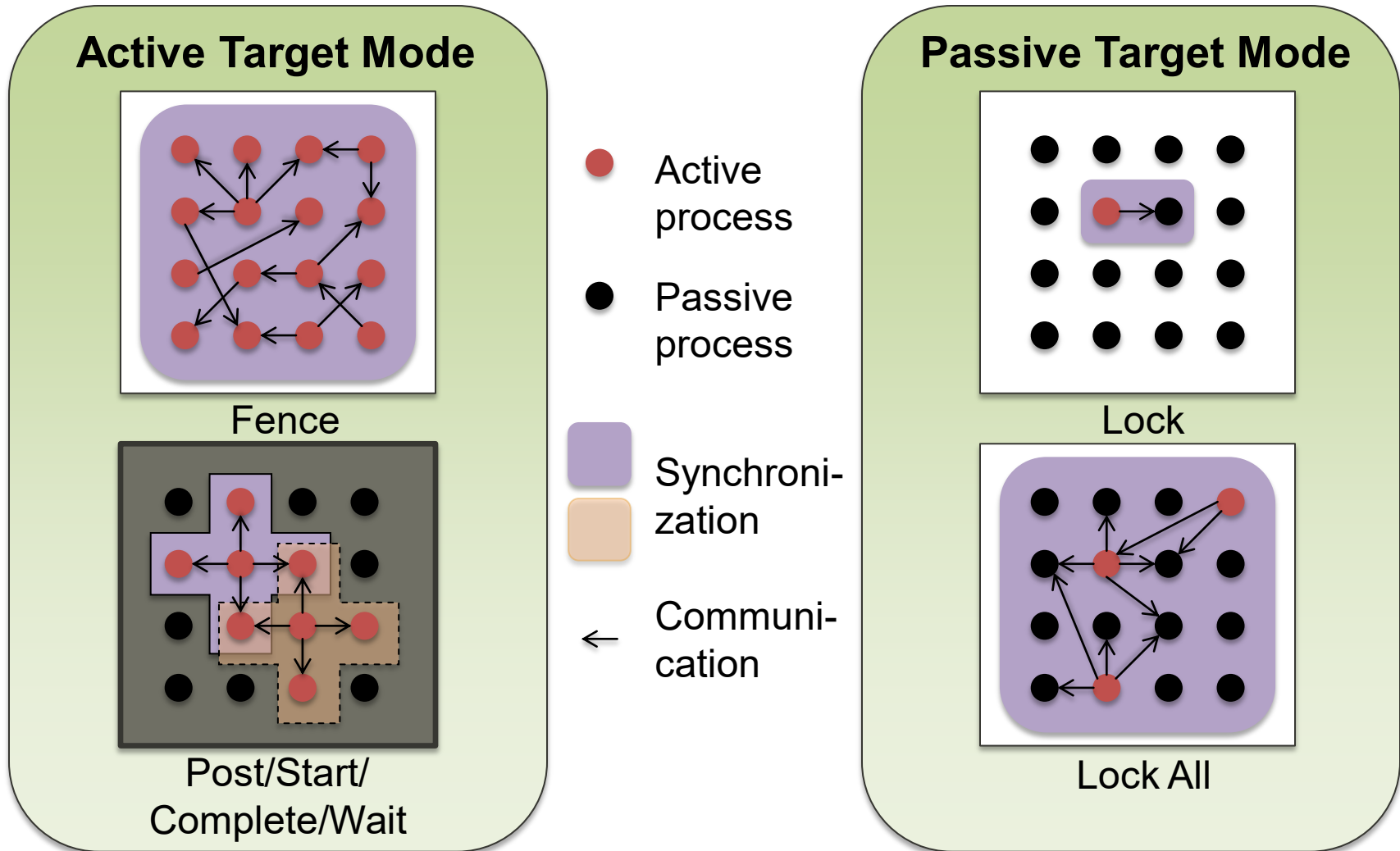
MPI-3 RMA SYNCHRONIZATION OVERVIEW



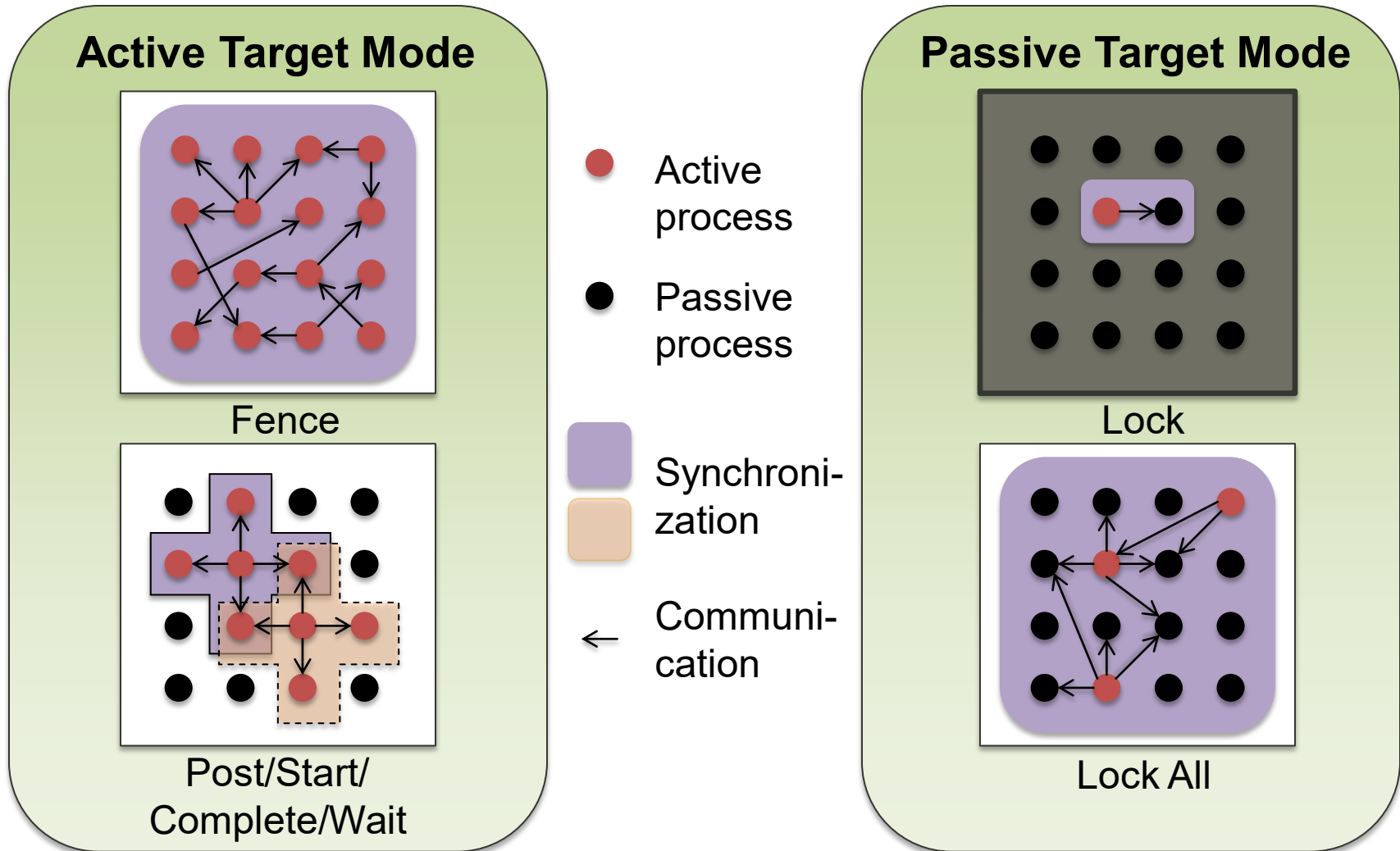
MPI-3 RMA Synchronization Overview



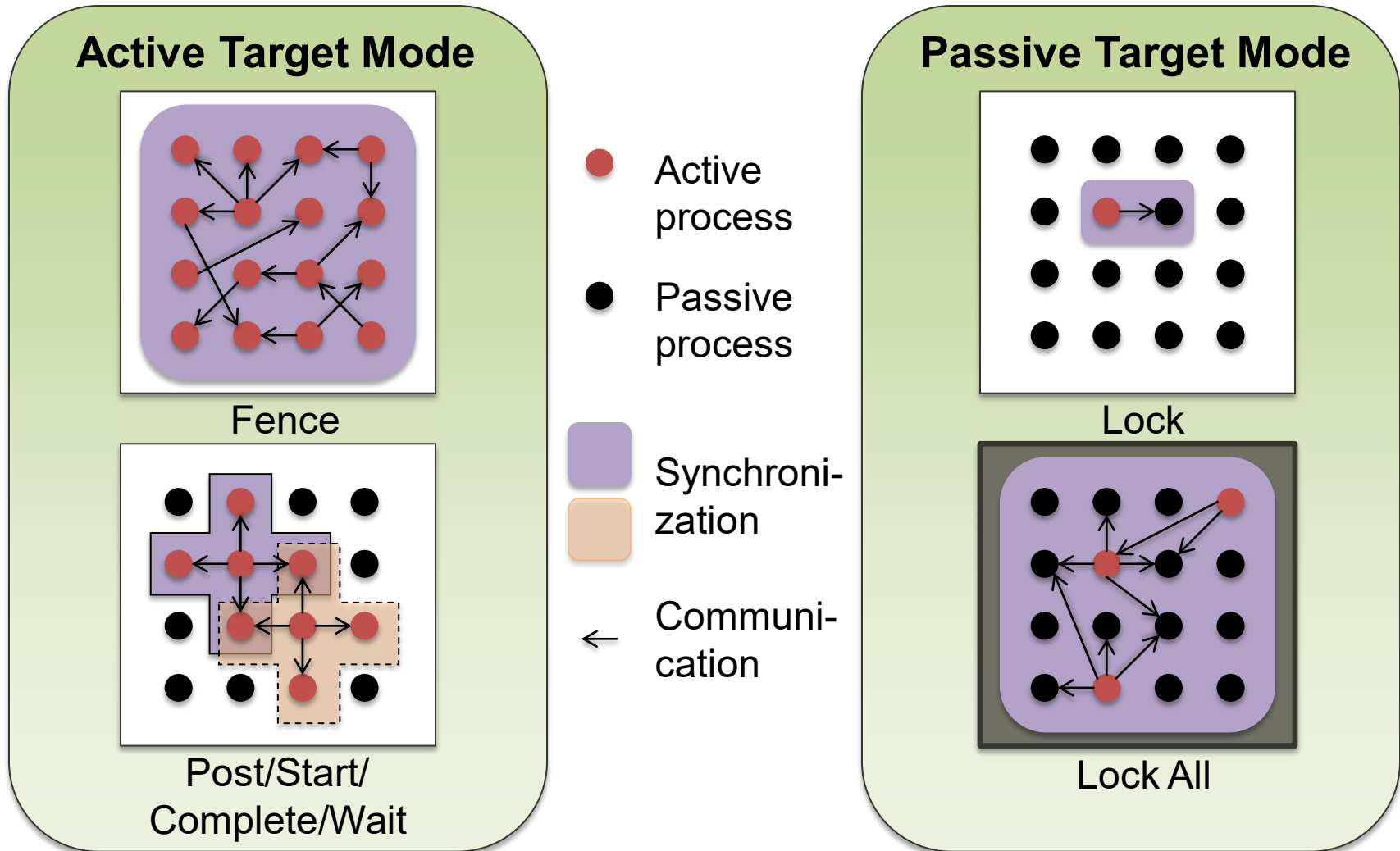
MPI-3 RMA SYNCHRONIZATION OVERVIEW



MPI-3 RMA Synchronization Overview

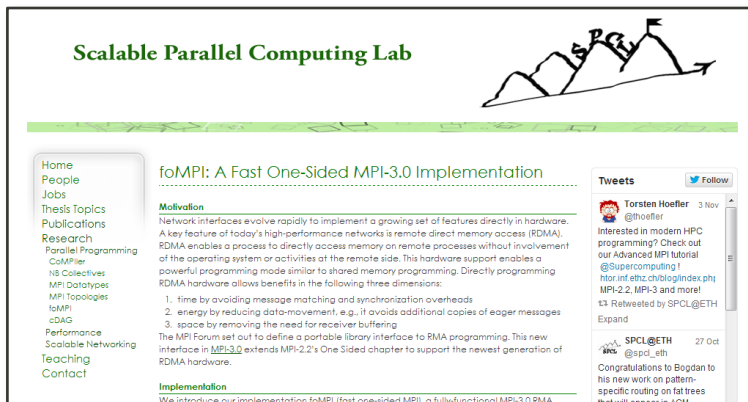


MPI-3 RMA Synchronization Overview



SCALABLE PROTOCOLS & REFERENCE IMPLEMENTATION

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

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., if 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.

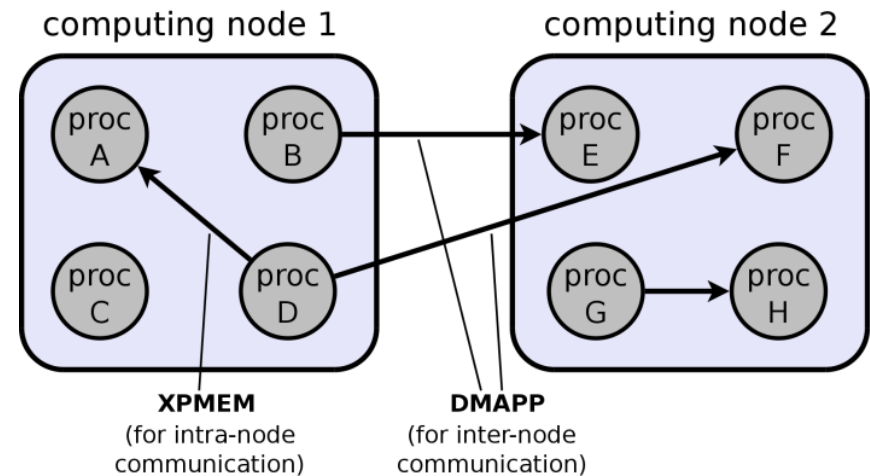
Implementation

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

Tweets

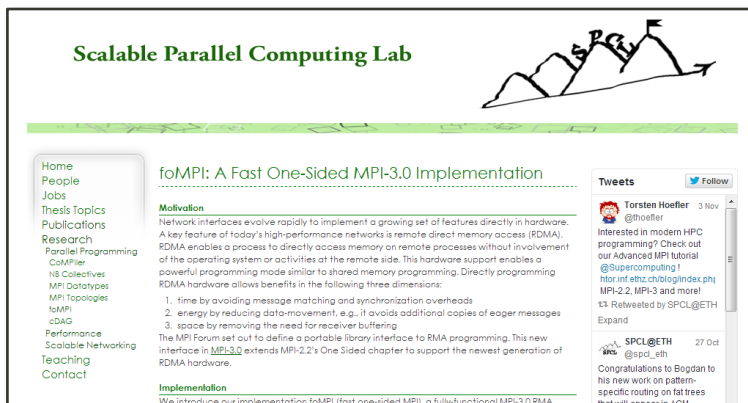
Torsten Hoefler @thoefler 3 Nov
Interested in modern HPC programming? Check out our Advanced MPI tutorial @Supercomputing1 http://inf.ethz.ch/blog/index.php/MPI-2.2, MPI-3 and more!
Retweeted by SPCL@ETH Expand

SPCL@ETH @spcl_eth 27 Oct
Congratulations to Bogdan to his new work on pattern-specific routing on fat trees that will speed up MPI



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

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., if 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.

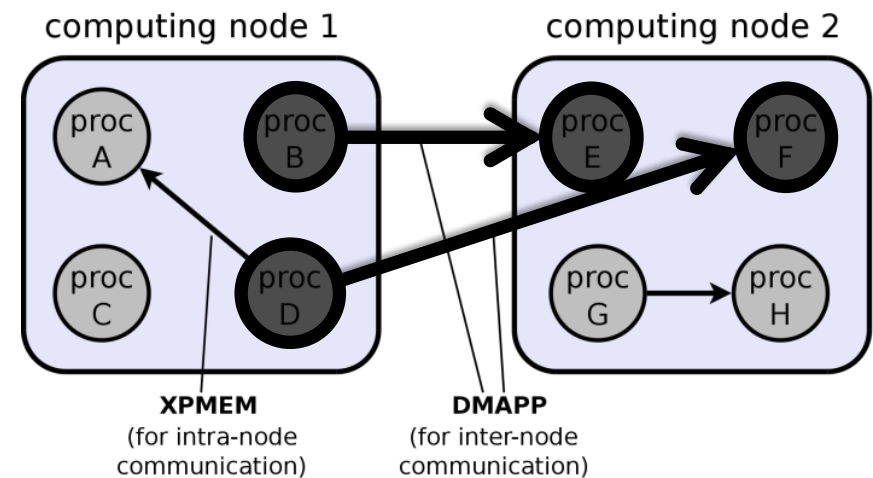
Implementation

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

Tweets

Torsten Hoefler @thoefler 3 Nov
Interested in modern HPC programming? Check out our Advanced MPI tutorial @Supercomputing1 for 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 speed up MPI



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

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., if avoids additional copies of eager messages
3. space by removing the need for receiver buffering

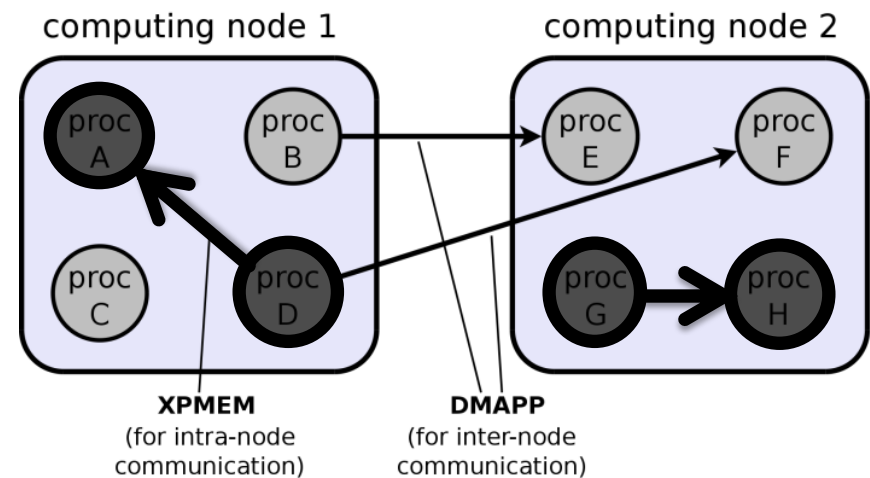
The MPI Forum set out to define a portable library interface to RMA programming. This new interface in [MPI-3.0](#) extends MPI-2.2's One Sided chapter to support the newest generation of RDMA hardware.

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

Tweets

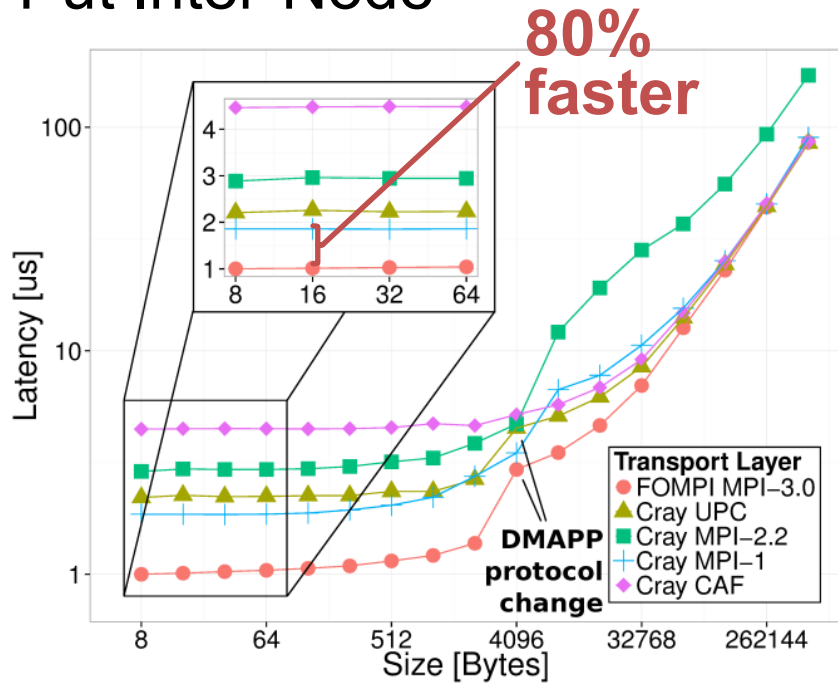
Torsten Hoefler @thoefler 3 Nov
 Interested in modern HPC programming? Check out our Advanced MPI tutorial @Supercomputing1 http://inf.ethz.ch/blog/index.php/MPI-2.2,MPI-3 and more!
 Retweeted by SPCL@ETH Expand

SPCL@ETH @spcl_eth 27 Oct
 Congratulations to Bogdan to his new work on pattern-specific routing on fat trees that will speed up MPI

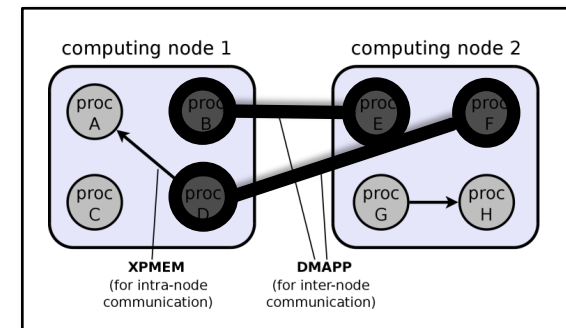
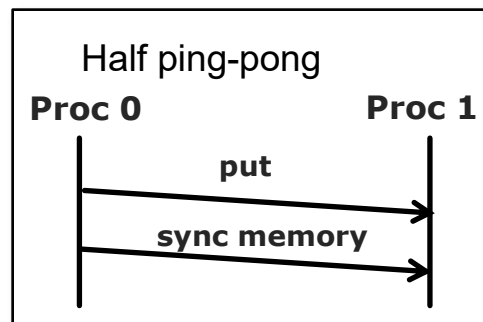
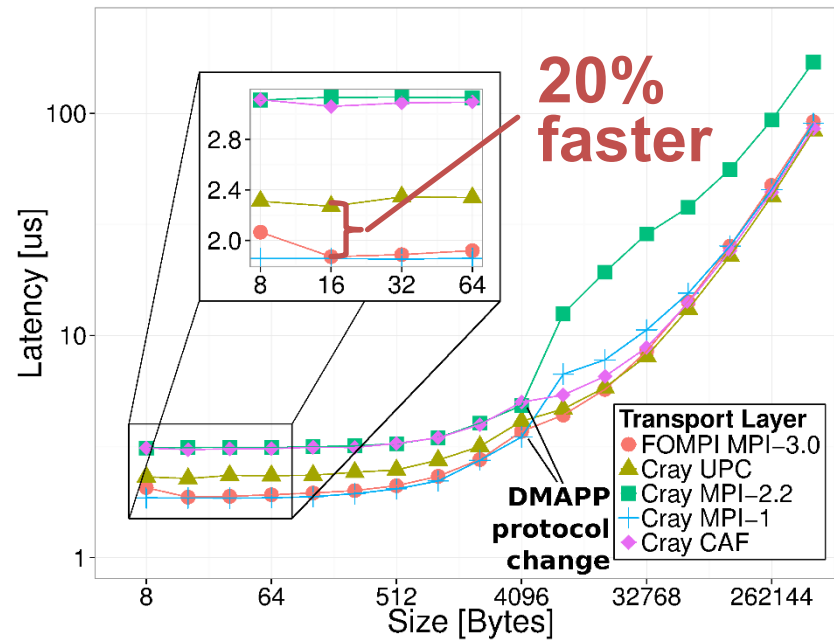


PERFORMANCE INTER-NODE: LATENCY

Put Inter-Node

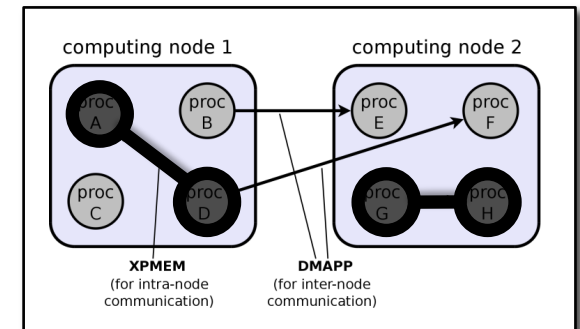
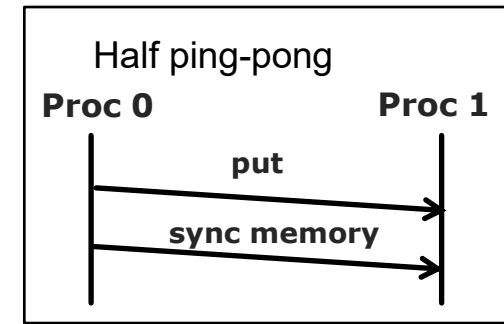
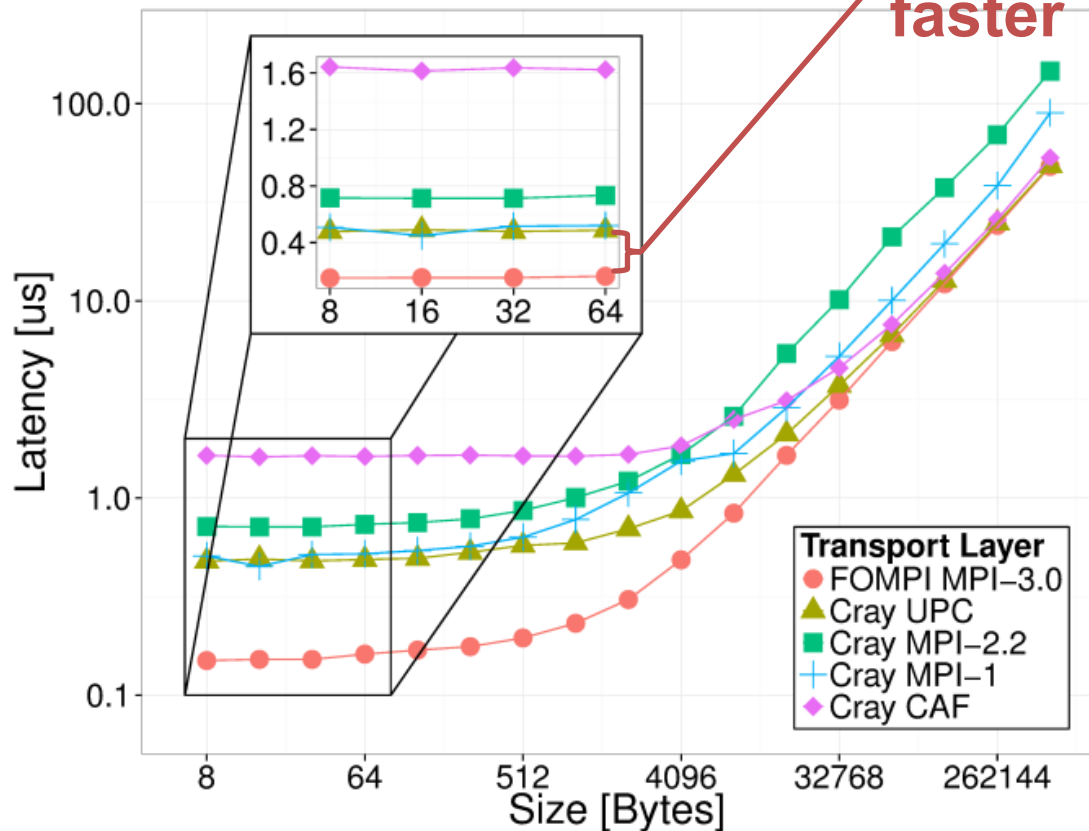


Get Inter-Node



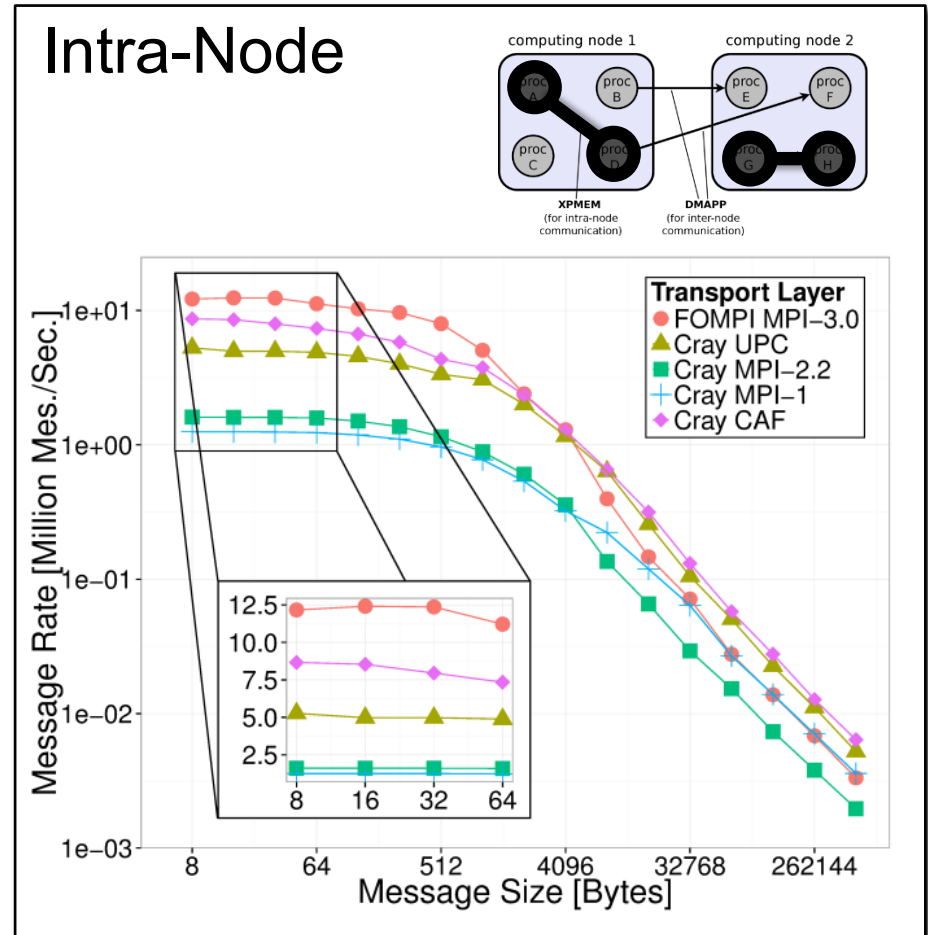
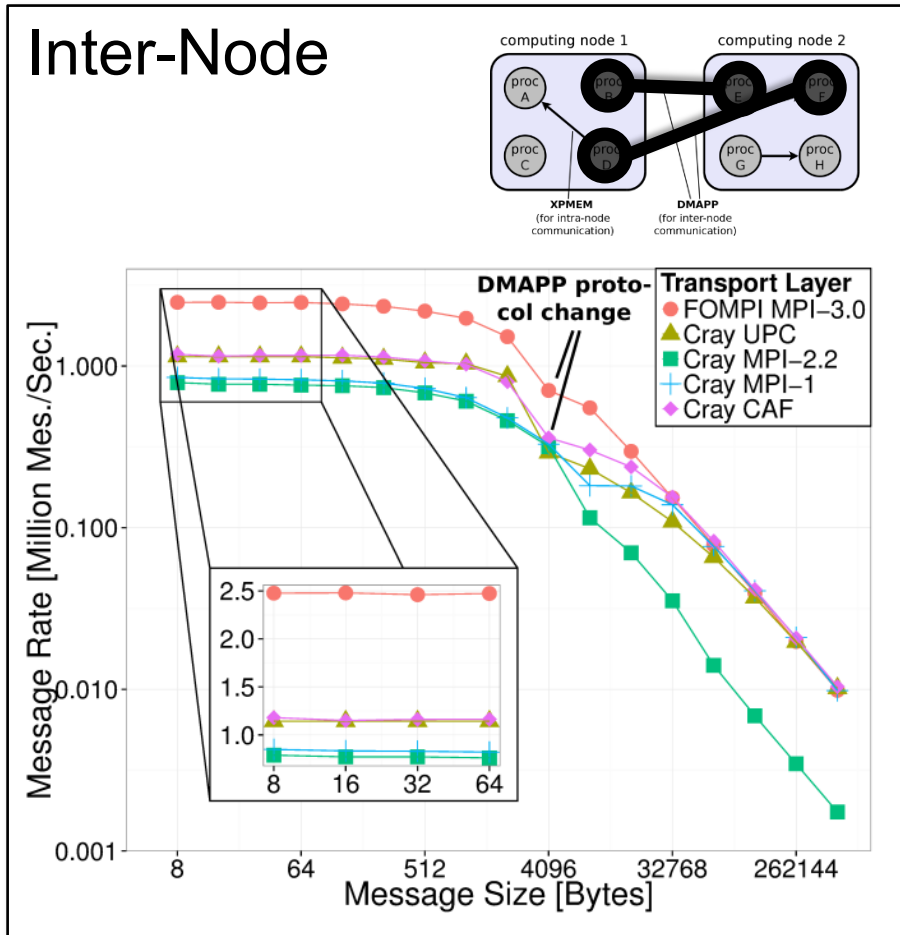
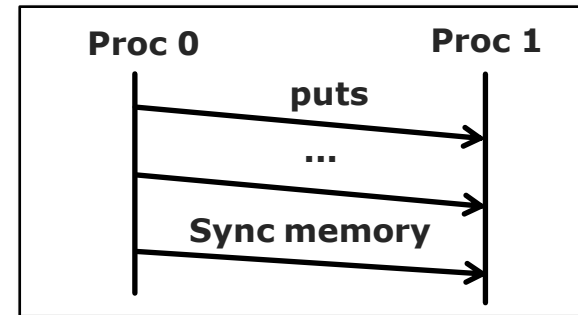
PERFORMANCE INTRA-NODE: LATENCY

Put/Get Intra-Node



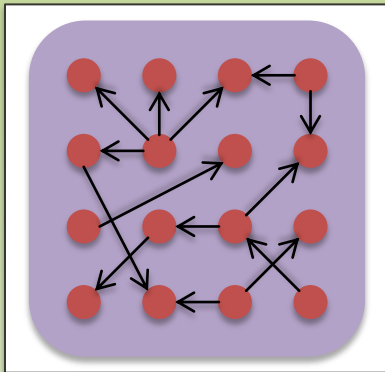


PERFORMANCE: MESSAGE RATE

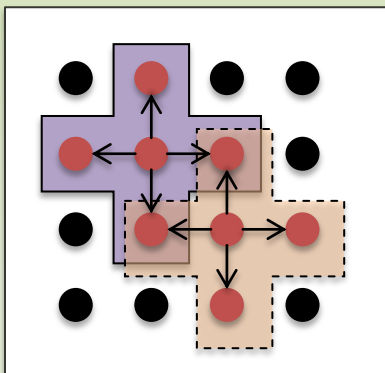


PART 3: SYNCHRONIZATION

Active Target Mode



Fence



Post/Start/
Complete/Wait

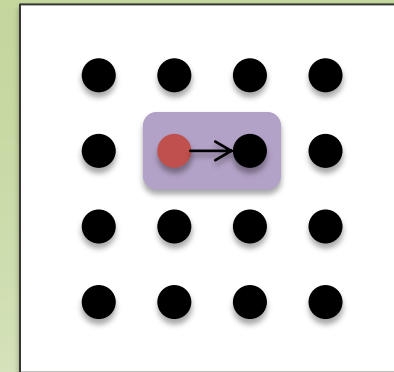
● Active process

● Passive process

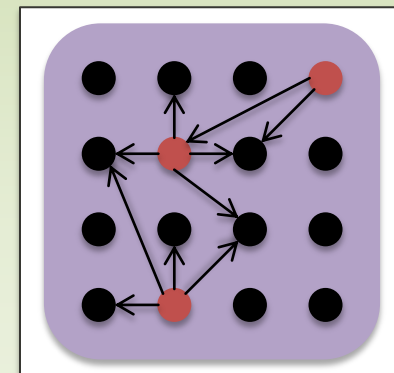
■ Synchroni-
zation

← Communi-
cation

Passive Target Mode

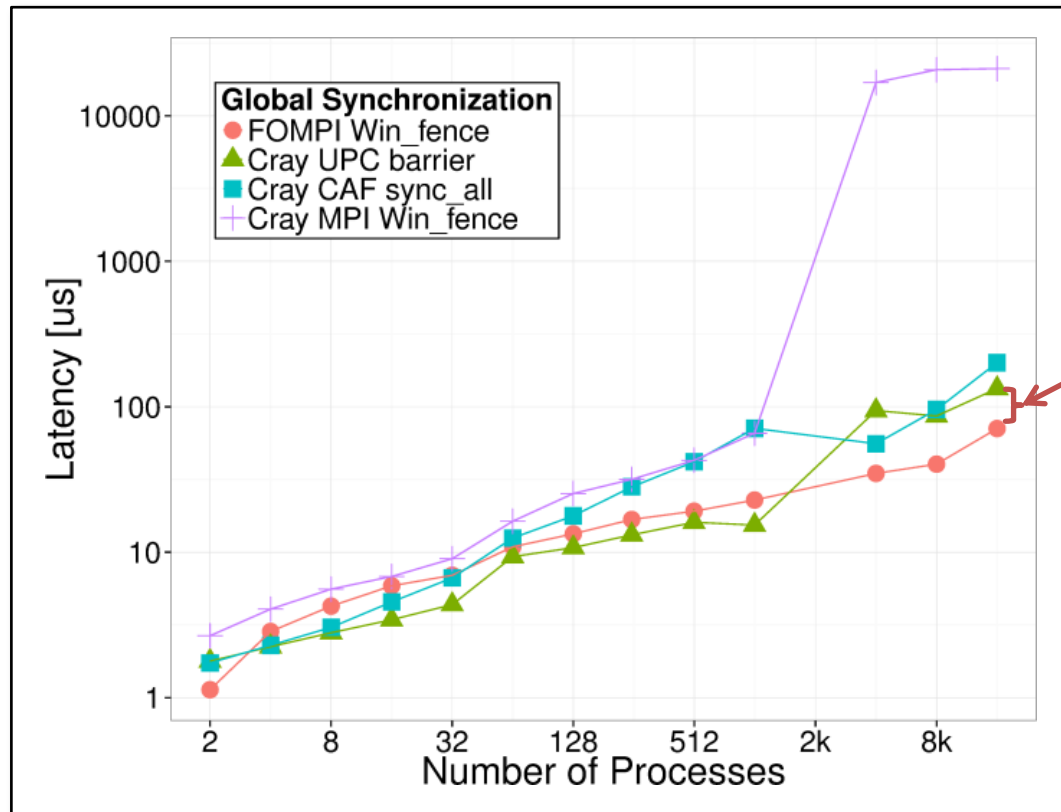


Lock



Lock All

SCALABLE FENCE PERFORMANCE



~1/2 latency

Time bound

$\mathcal{O}(\log p)$

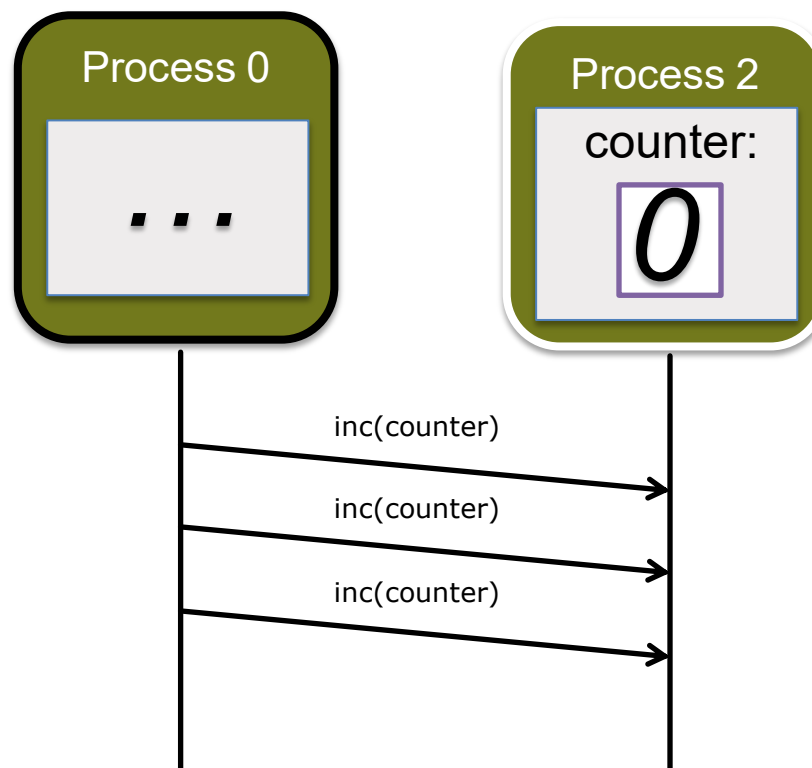
Memory bound

$\mathcal{O}(1)$

FLUSH SYNCHRONIZATION

Time bound	$O(1)$
Memory bound	$O(1)$

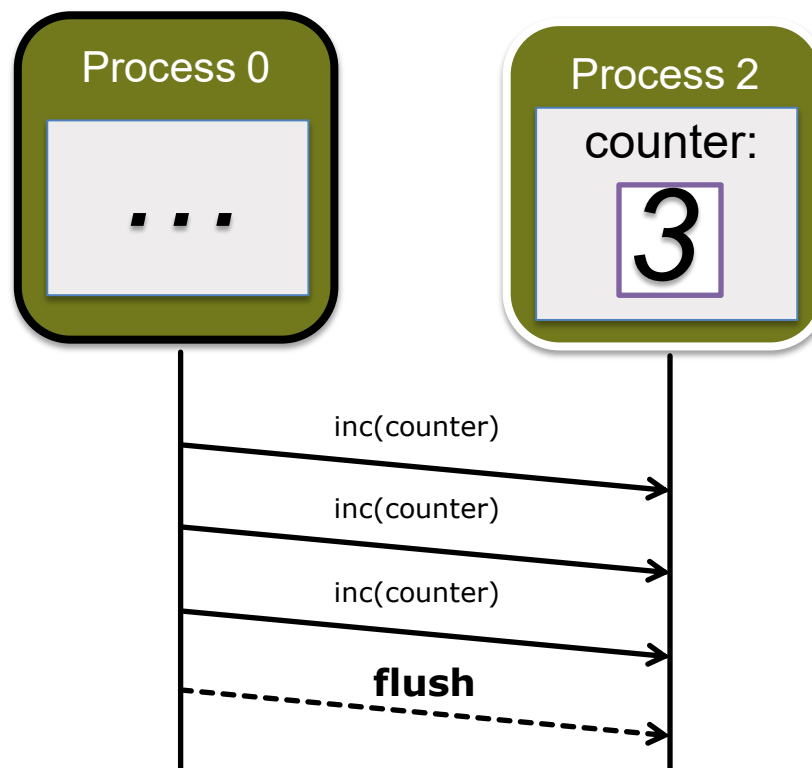
- Guarantees remote completion
- Performs a remote bulk synchronization and an x86 mfence
- One of the most performance critical functions, we add only **78 x86** CPU instructions to the critical path



FLUSH SYNCHRONIZATION

Time bound	$O(1)$
Memory bound	$O(1)$

- 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



APPLICATION PERFORMANCE

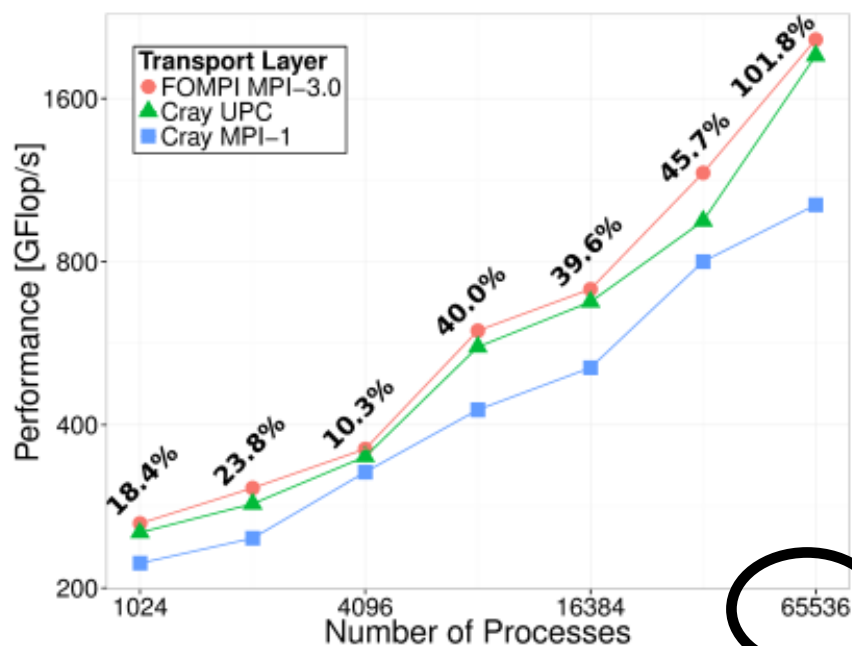
- Evaluation on Blue Waters System
 - 22,640 computing Cray XE6 nodes
 - 724,480 schedulable cores
- One nearly full-scale run 😊



PERFORMANCE: APPLICATIONS

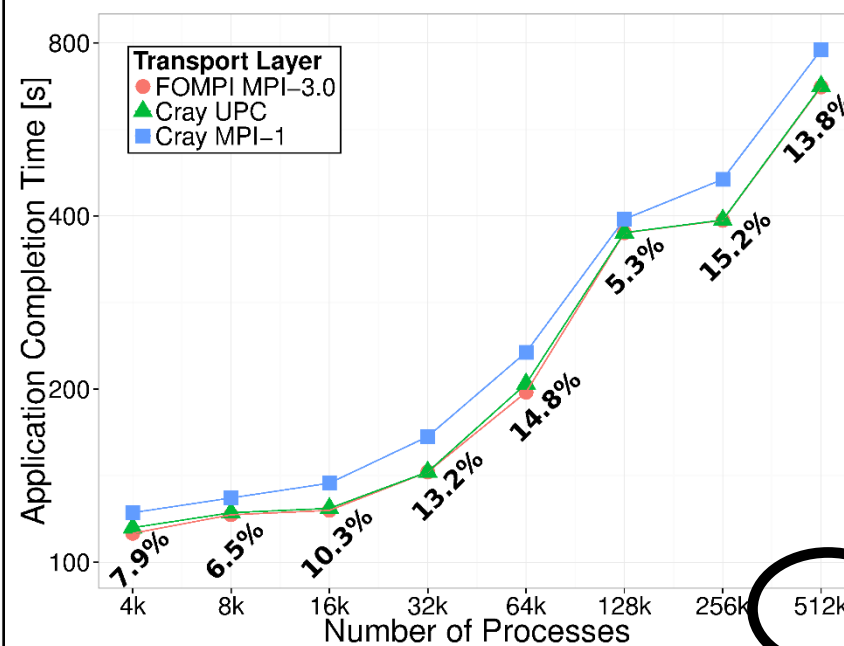
Annotations represent performance gain of foMPI [3] over Cray MPI-1.

NAS 3D FFT [1] Performance



scale
to 65k procs

MILC [2] Application 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

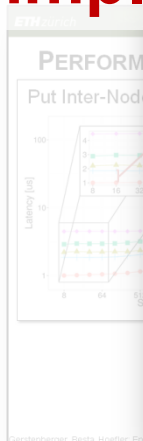
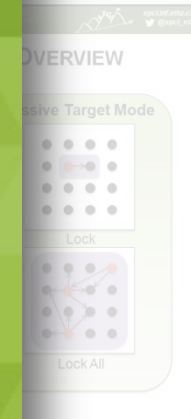
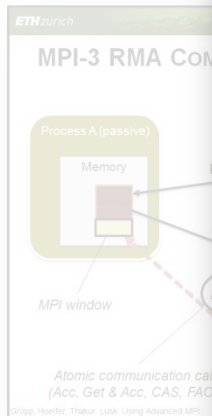
[3] Gerstenberger, Besta, Hoefer: Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided, SC13

IN CASE YOU WANT TO LEARN MORE

- Available
- Some are

SCIENTIFIC
AND
ENGINEERING
COMPUTATION
SERIES

*Using Advanced MPI
Modern Features of the
Message-Passing Interface*



William Gropp

Torsten Hoefler

Rajeev Thakur

Ewing Lusk



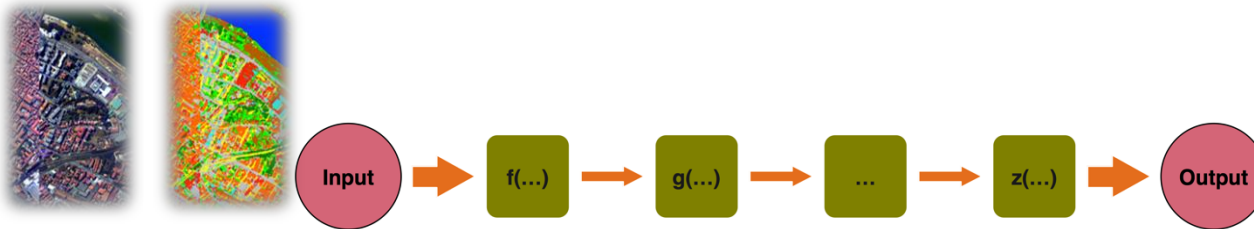
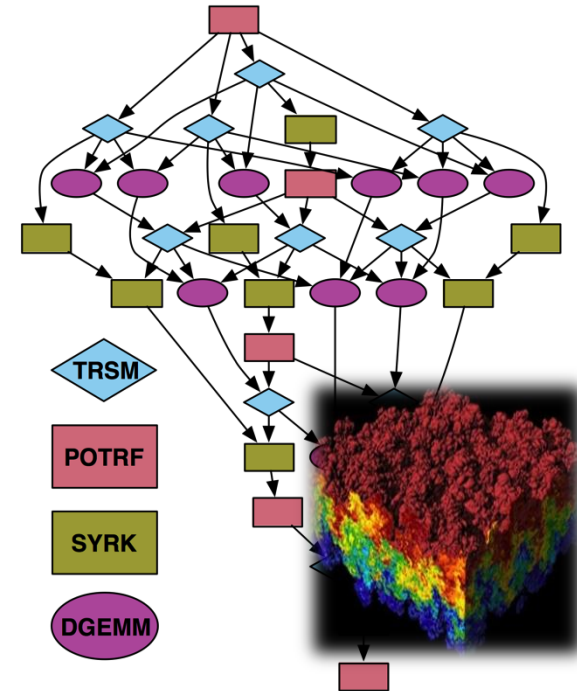
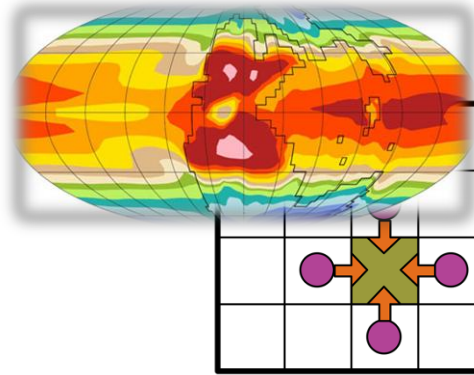
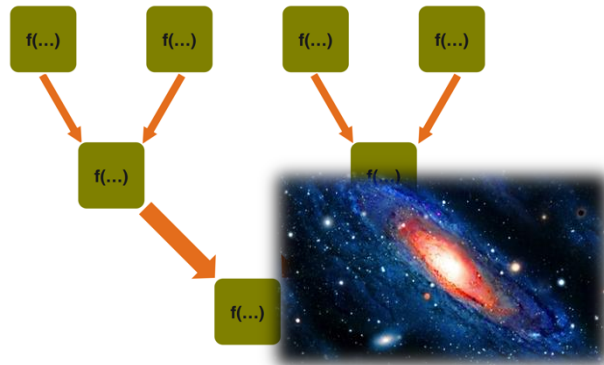
and communication and overlap, IPOPS'09
Using with MPI-3 One Sided, SC13

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?

ONE SIDED – PUT + SYNCHRONIZATION

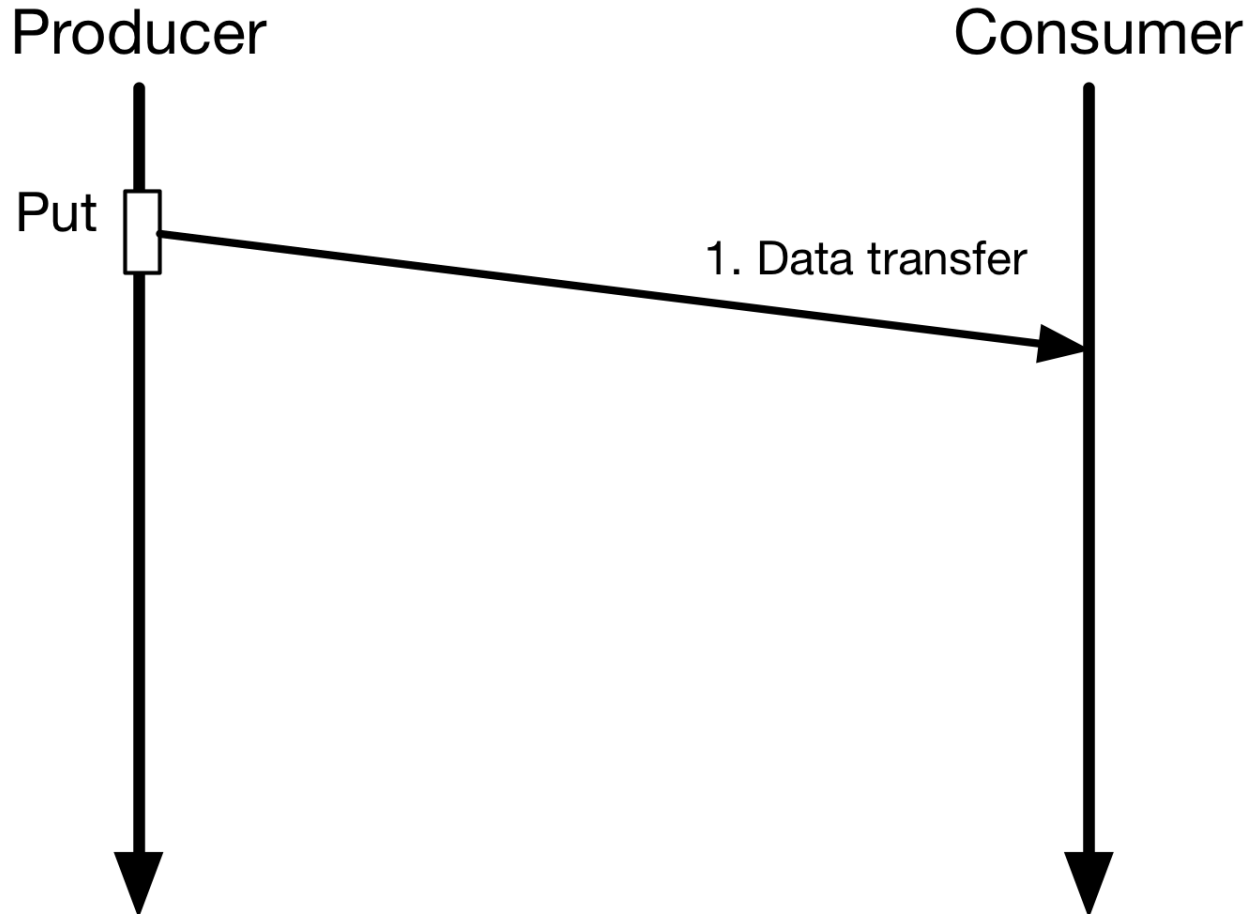
Producer



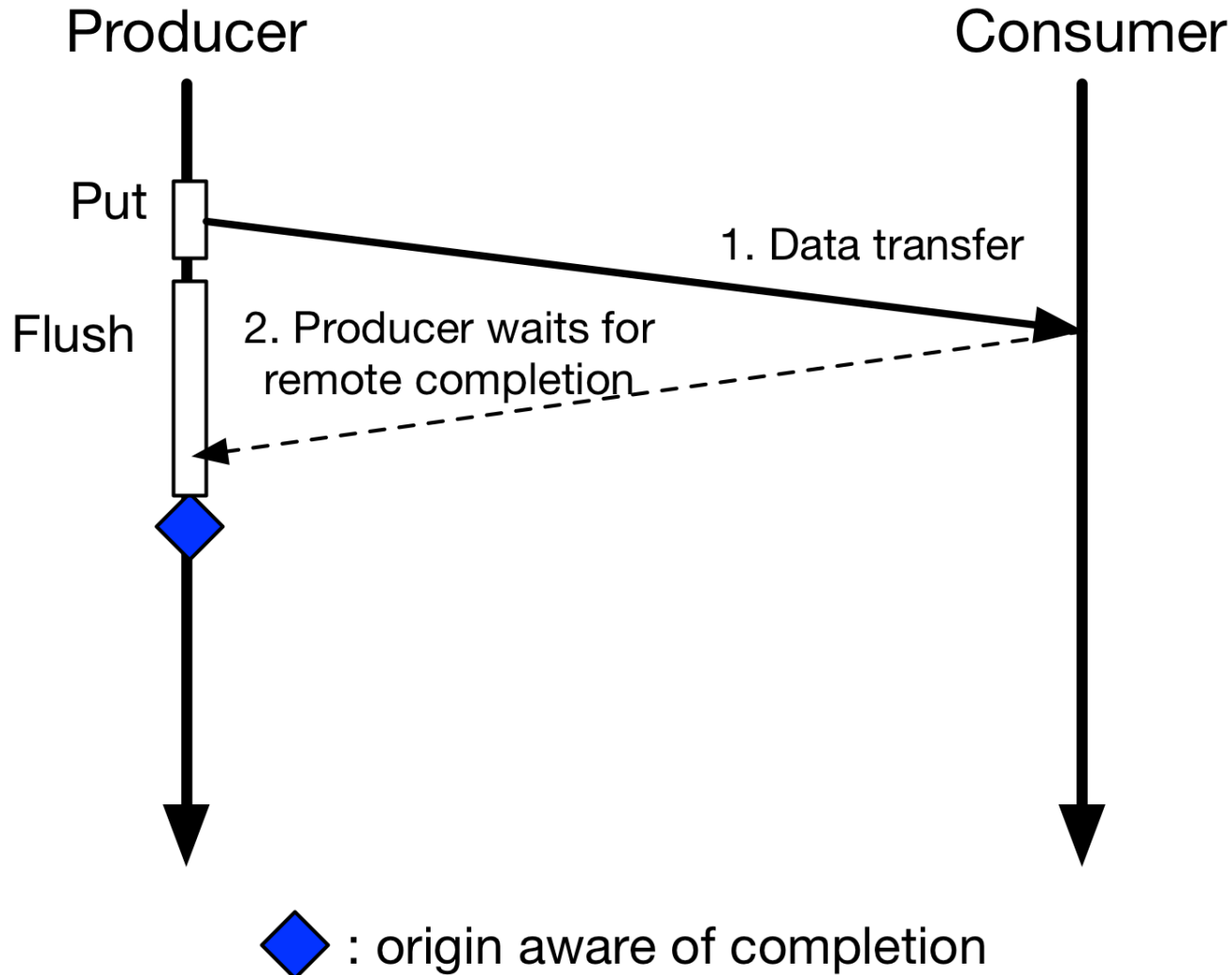
Consumer



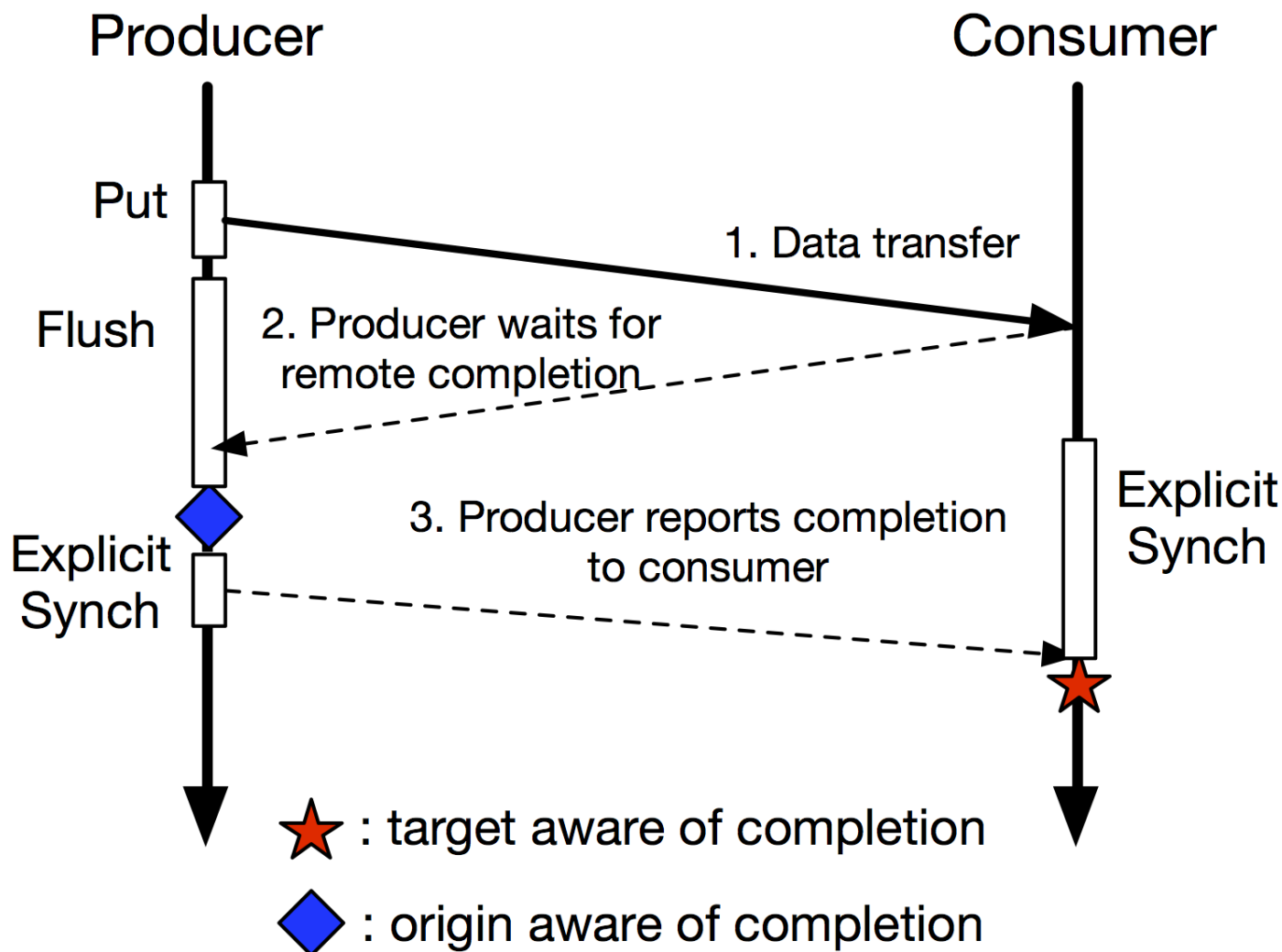
ONE SIDED – PUT + SYNCHRONIZATION



ONE SIDED – PUT + SYNCHRONIZATION



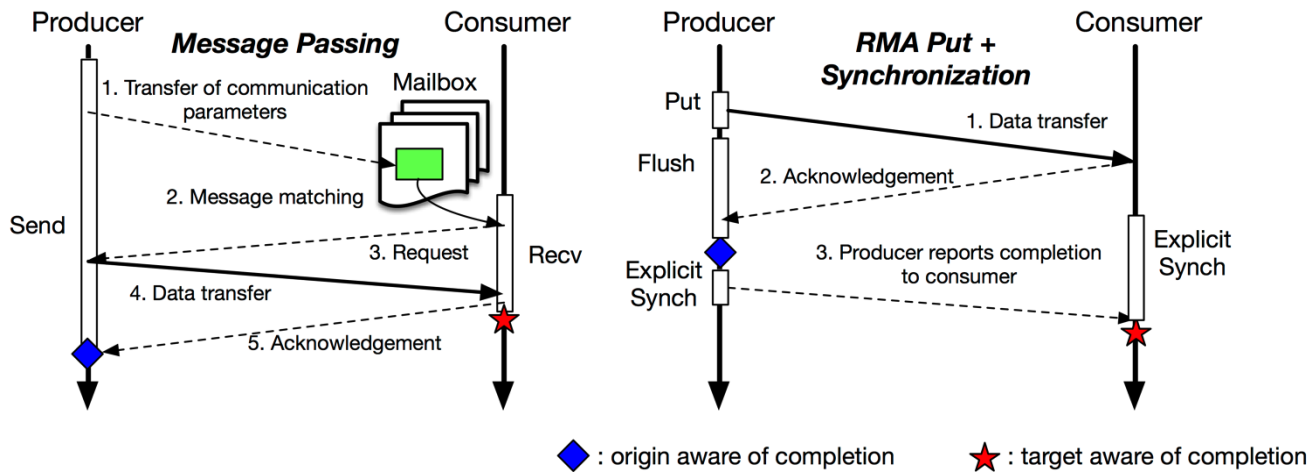
ONE SIDED – PUT + SYNCHRONIZATION



Critical path: 3 latencies



COMPARING APPROACHES

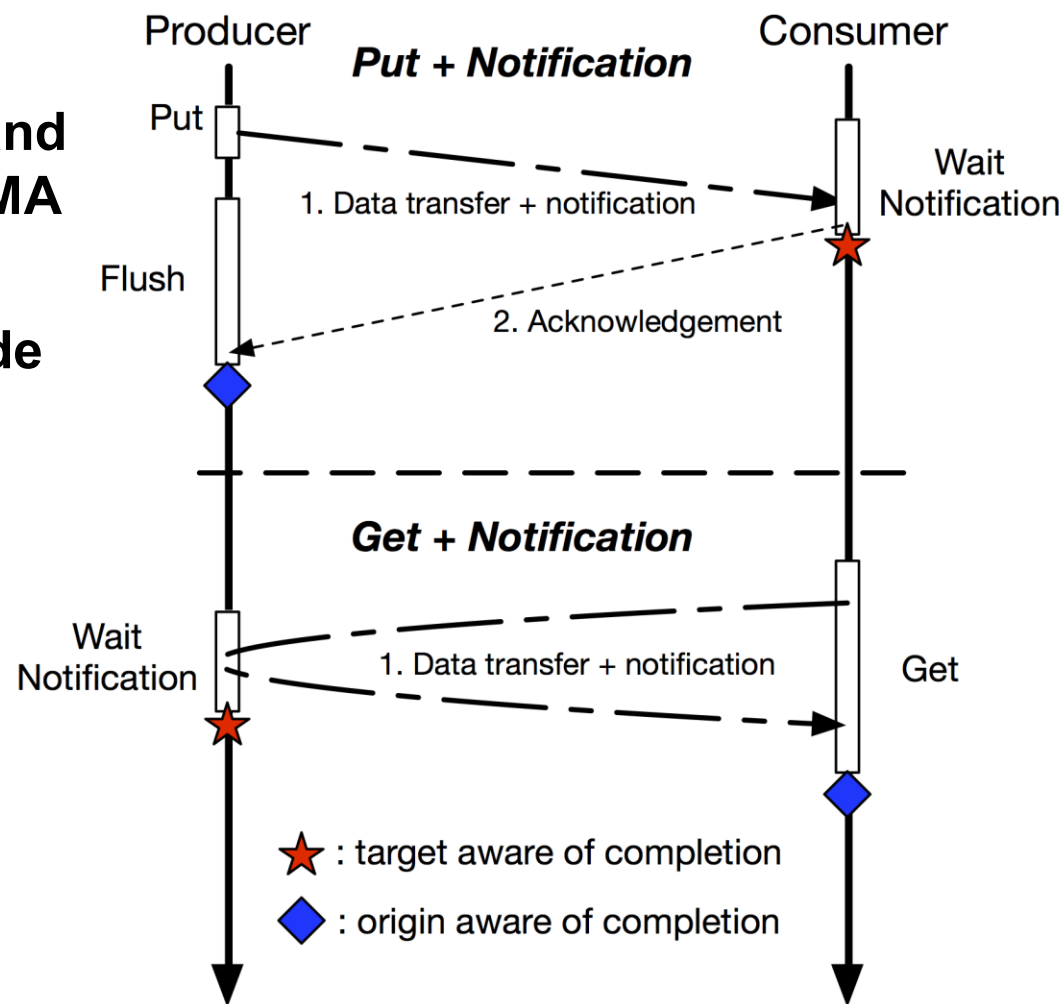


Message Passing
 1 latency + copy /
 3 latencies

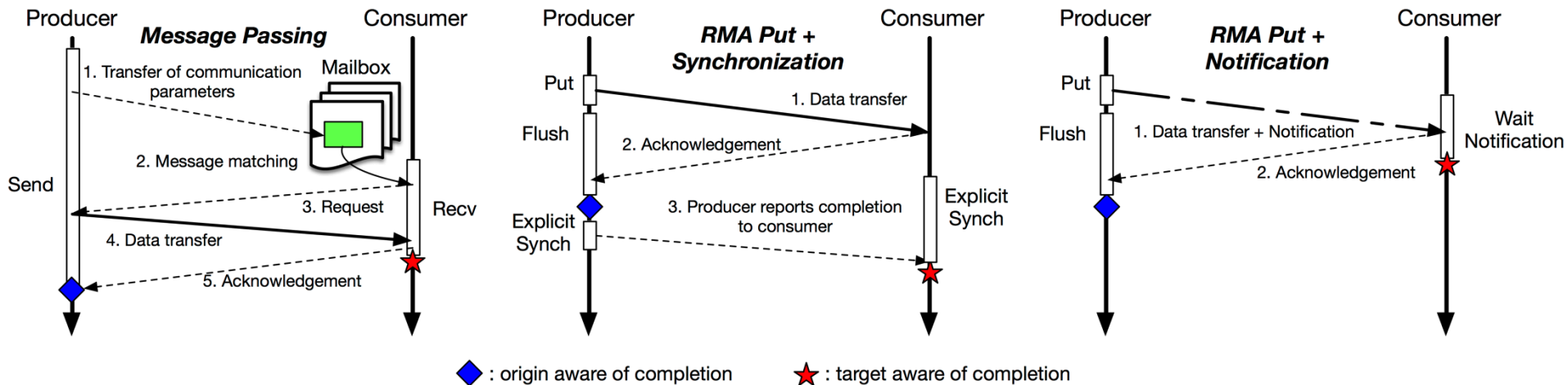
One Sided
 3 latencies

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



COMPARING APPROACHES

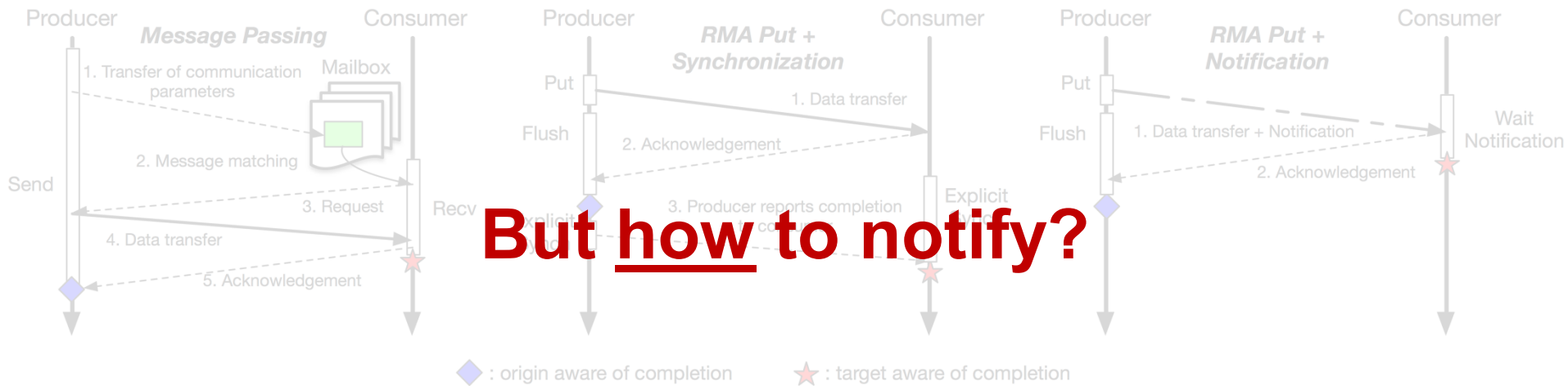


Message Passing
 1 latency + copy /
 3 latencies

One Sided
 3 latencies

Notified Access
 1 latency

COMPARING APPROACHES



But how to notify?

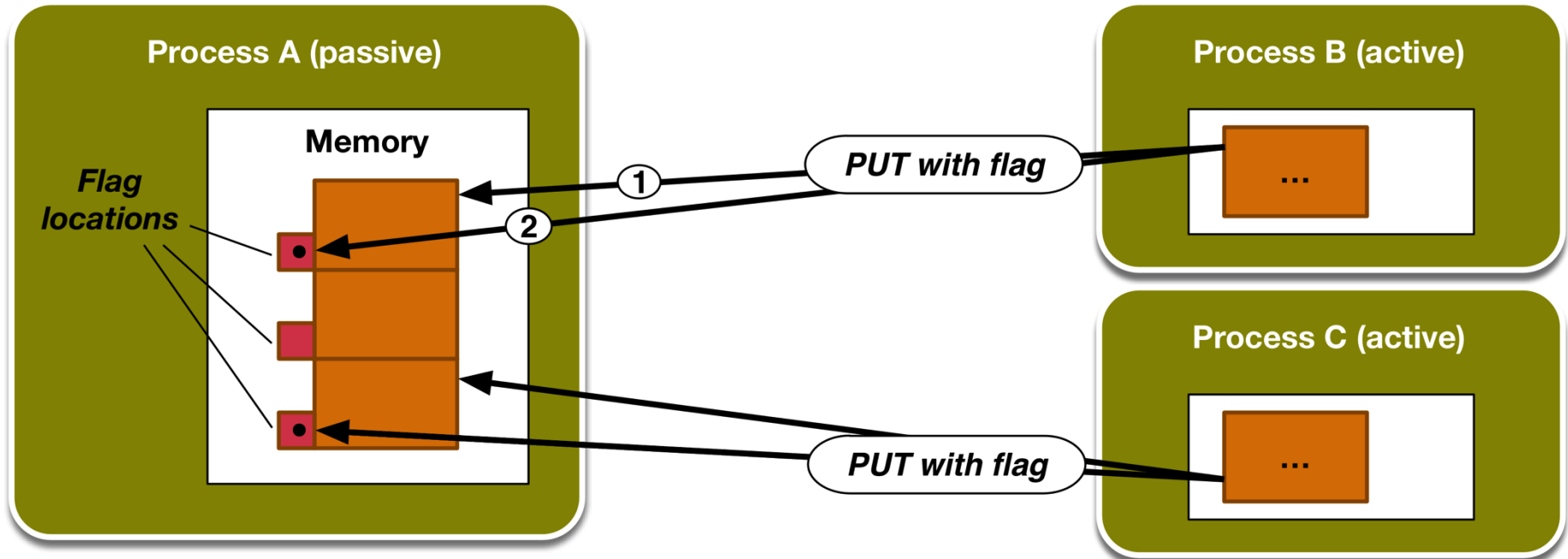
Message Passing
1 latency + copy /
3 latencies

One Sided
3 latencies

Notified Access
1 latency

PREVIOUS WORK: OVERWRITING INTERFACE

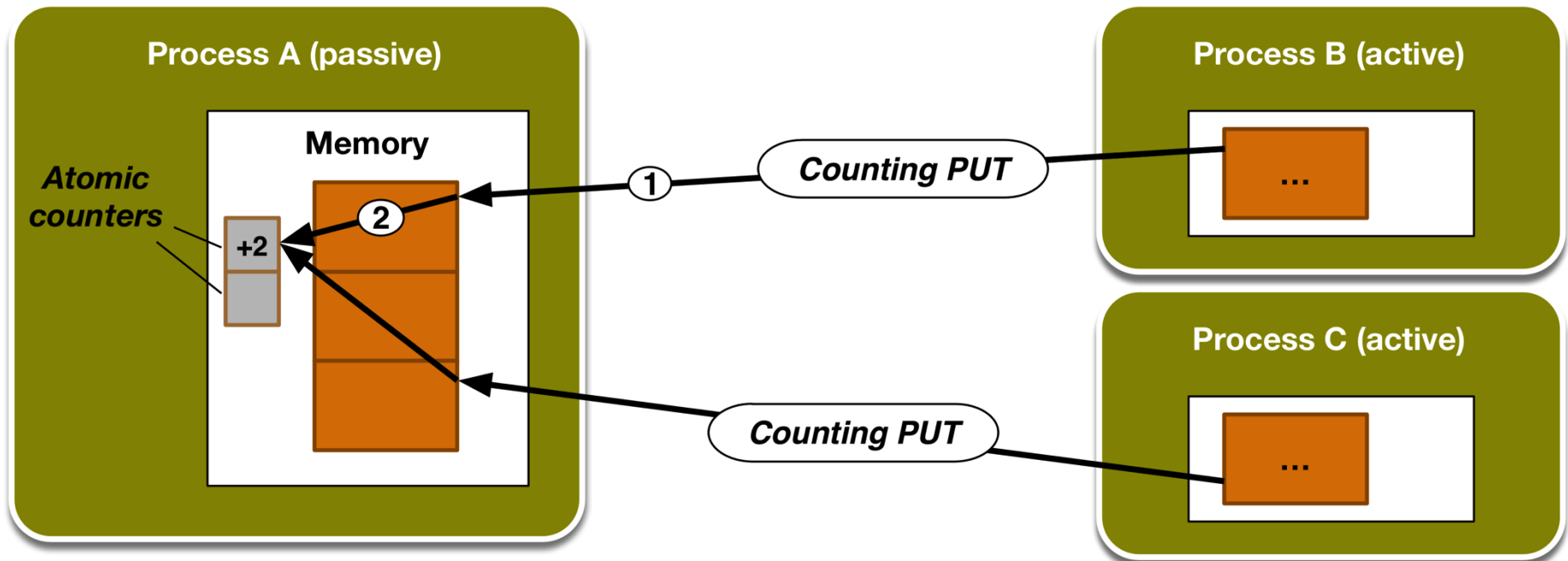
- **Flags (polling at the remote side)**
 - Used in *GASPI*, *DMAPP*, *NEON*



- **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

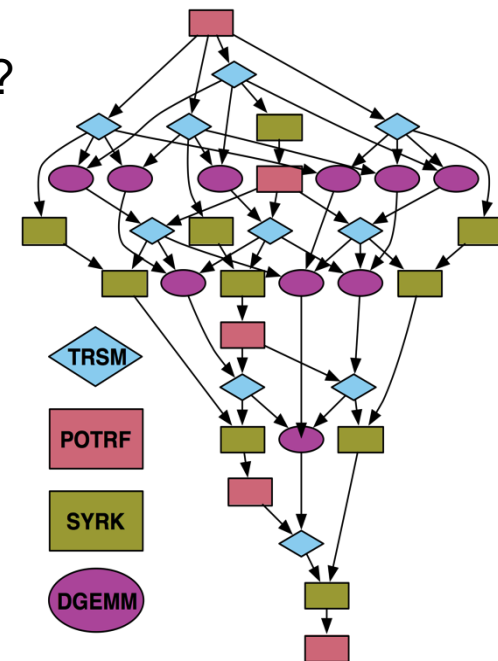
- Atomic counters (accumulate notifications → scalable)
 - Used in *Split-C*, *LAPI*, *SHMEM - Counting Puts*, ...



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

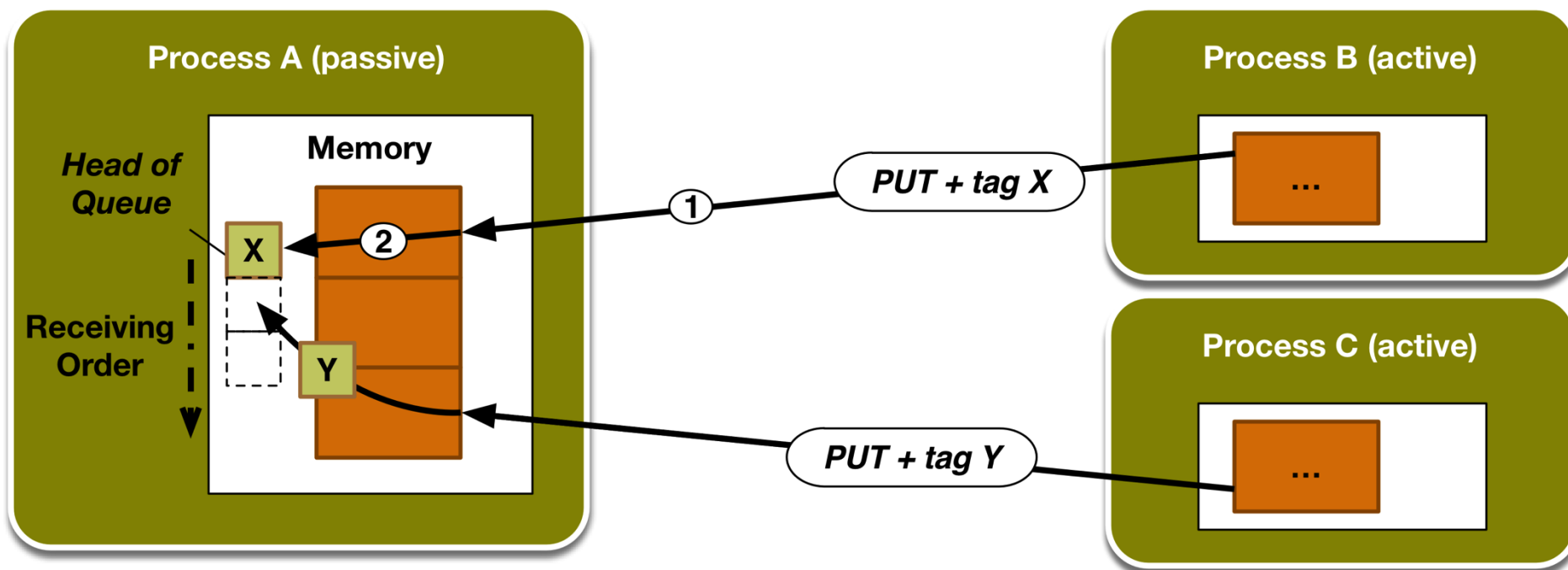
WHAT IS A GOOD NOTIFICATION INTERFACE?

- **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



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);
```

EXPERIMENTAL SETTING



CSCS

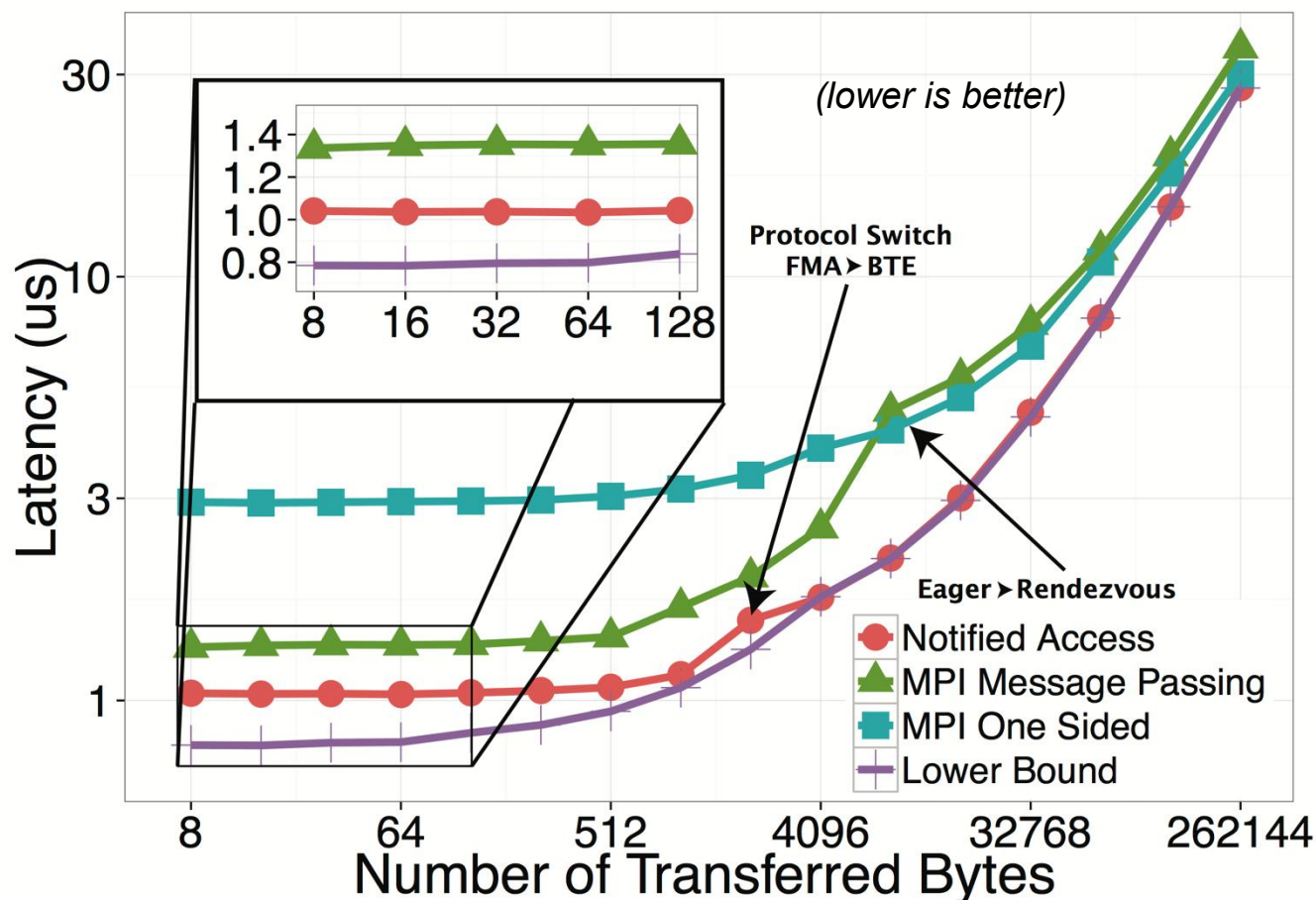
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

- **Piz Daint [1]**
 - 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



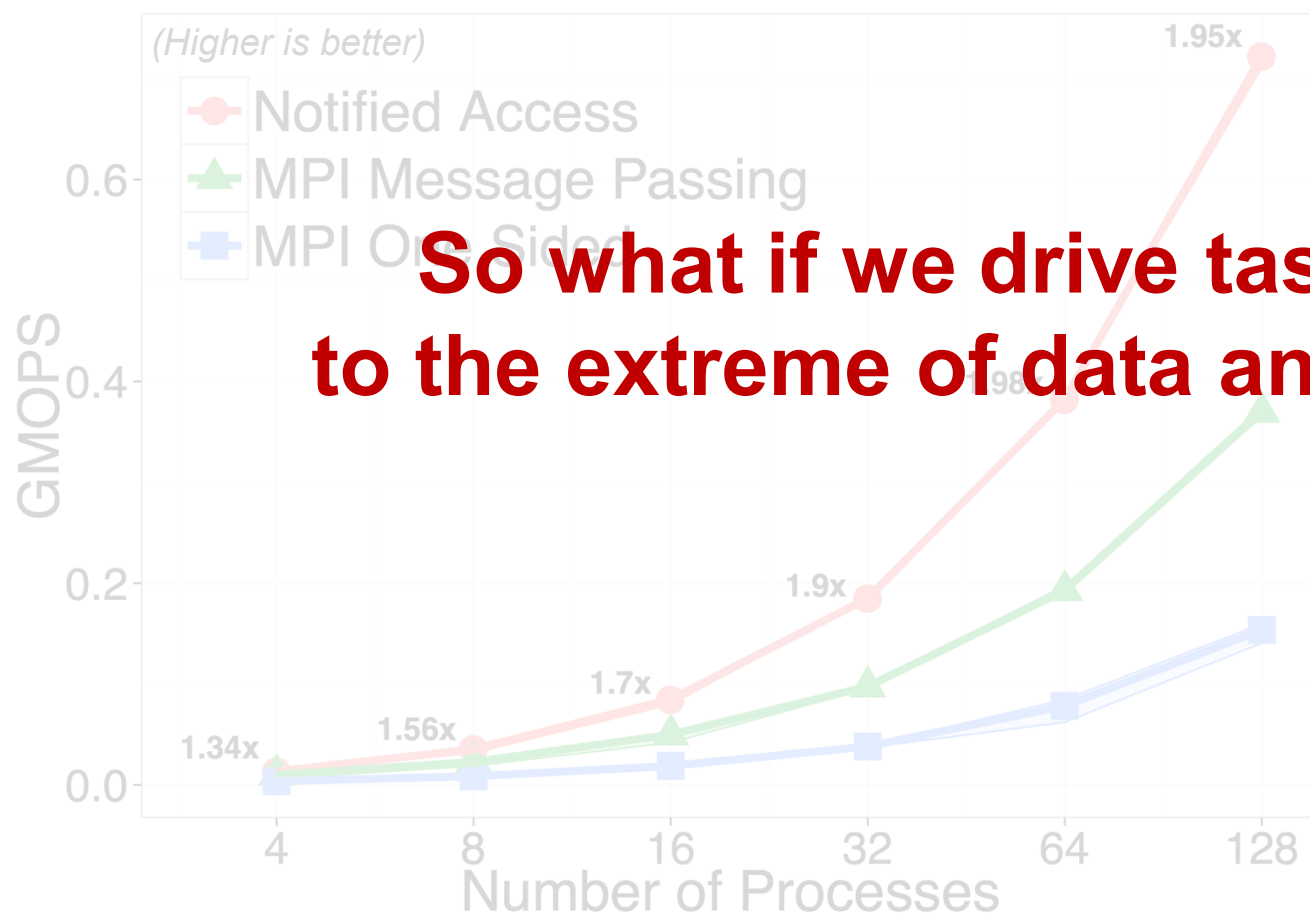
PING PONG PERFORMANCE (INTER-NODE)

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

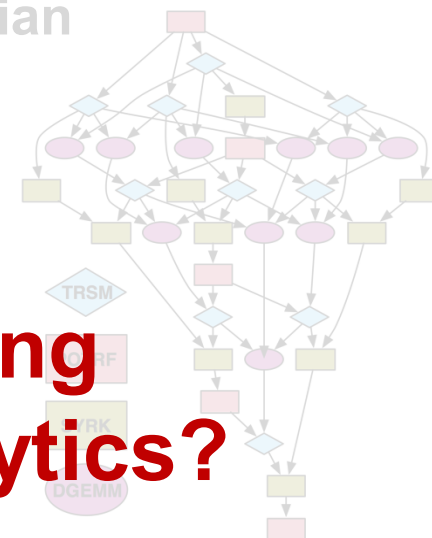


CHOLESKY – MANY-TO-MANY SYNCHRONIZATION

- 1,000 repetitions, each timed separately, RDTSC timer
- 95% confidence interval always within 10% of median

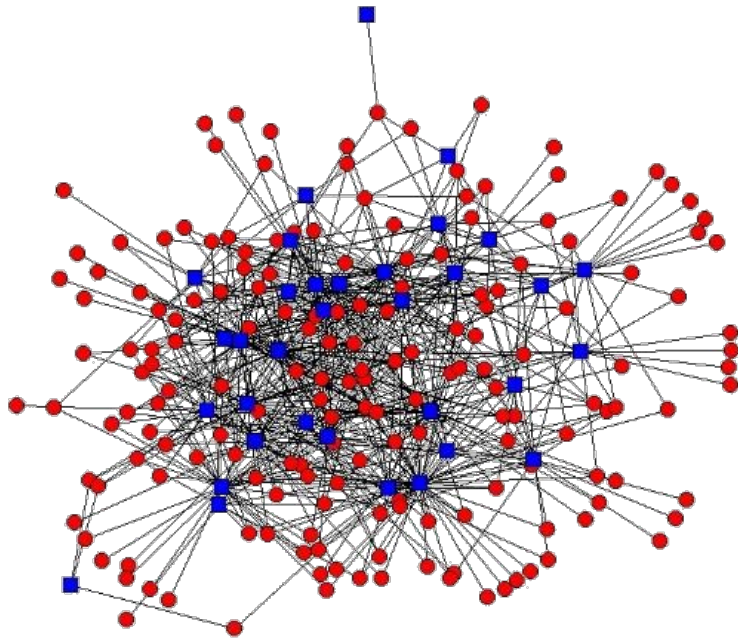


So what if we drive tasking to the extreme of data analytics?



LARGE-SCALE IRREGULAR GRAPH PROCESSING

- Becoming more important [1]
 - Machine learning
 - Computational science
 - Social network analysis



$$\frac{1}{\sqrt{2}} |\text{cat}\rangle + \frac{1}{\sqrt{2}} |\text{dog}\rangle$$

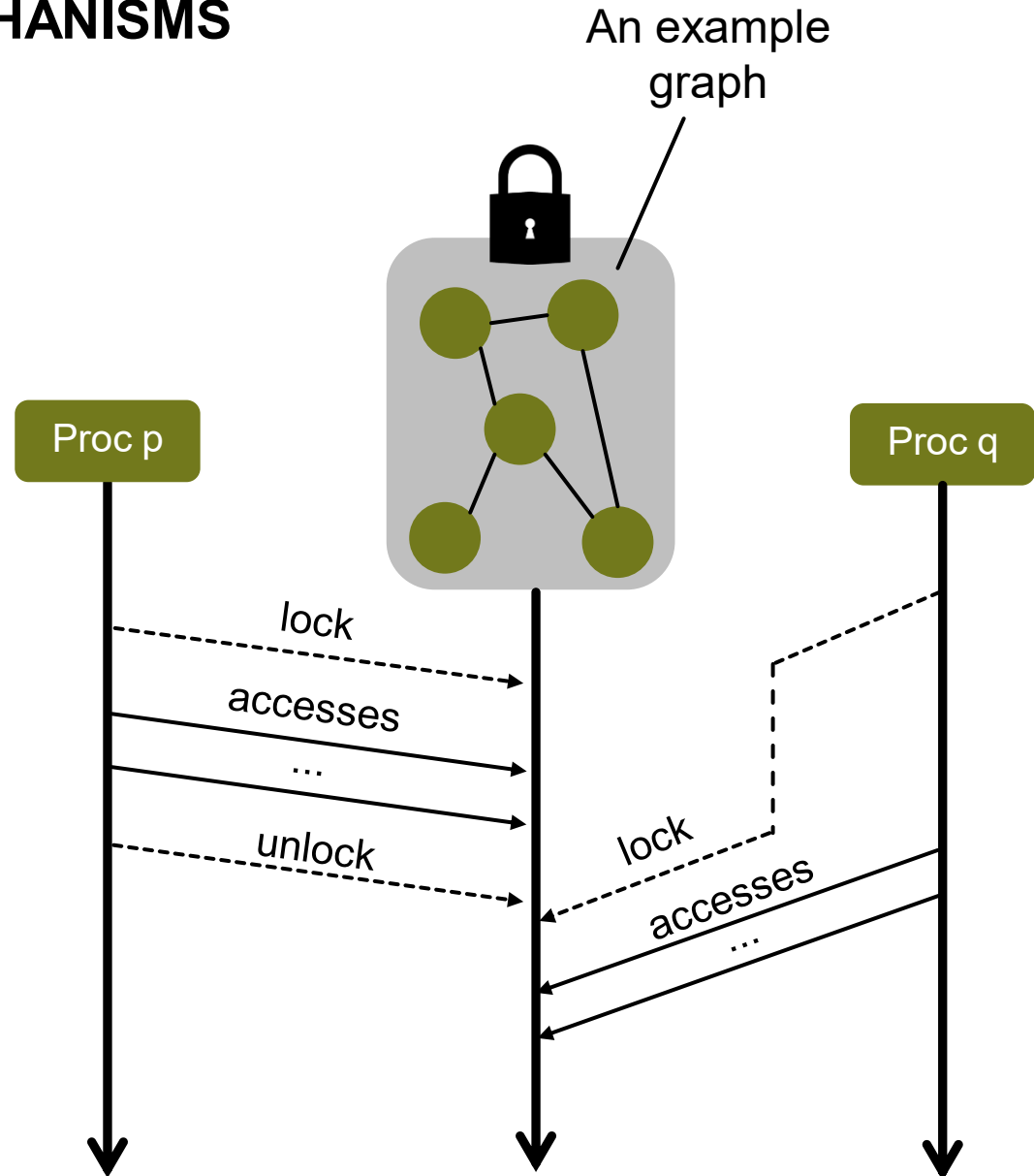
SYNCHRONIZATION MECHANISMS

COARSE LOCKS

✓ Simple protocols

✗ Serialization

✗ Detrimental performance



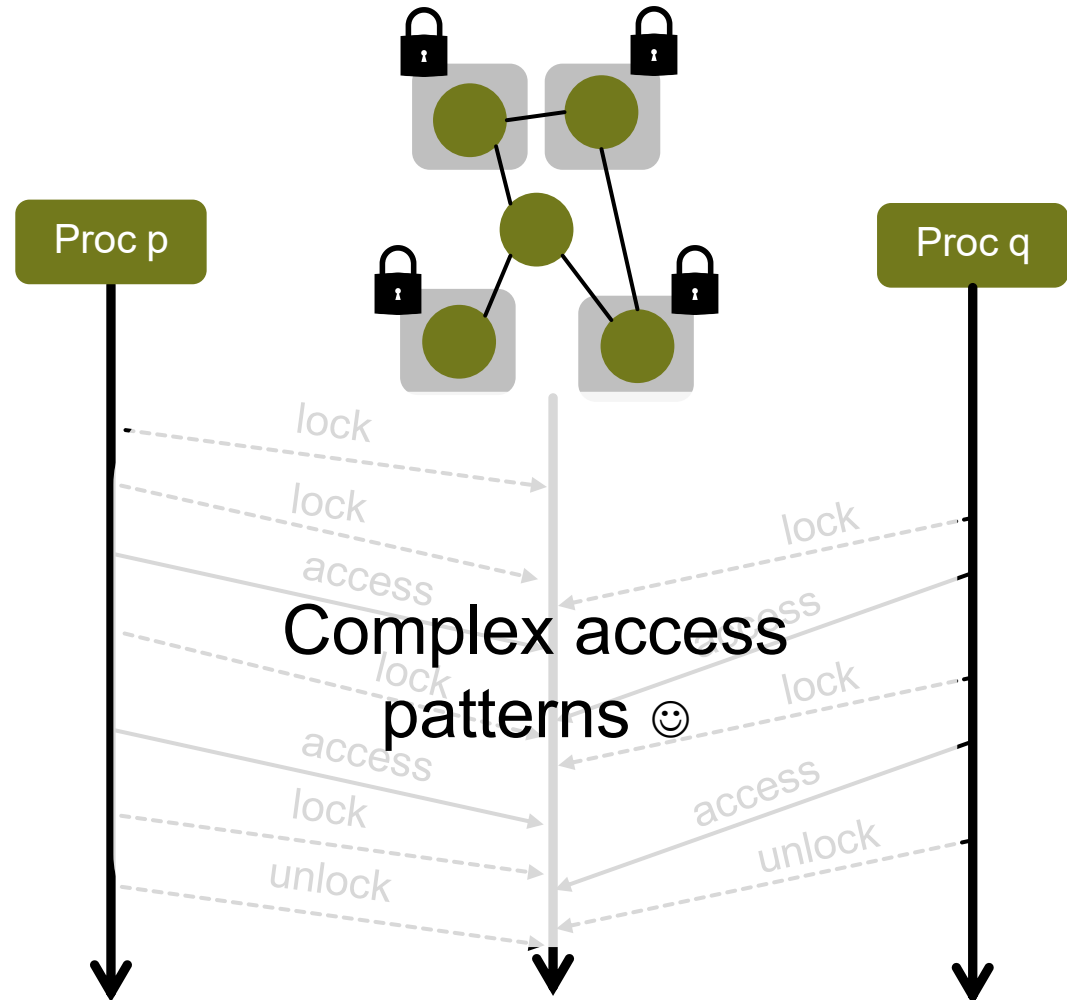
SYNCHRONIZATION MECHANISMS

FINE LOCKS

✓ Higher performance possible

✗ Complex protocols

✗ Risk of deadlocks



SYNCHRONIZATION MECHANISMS

ATOMIC OPERATIONS



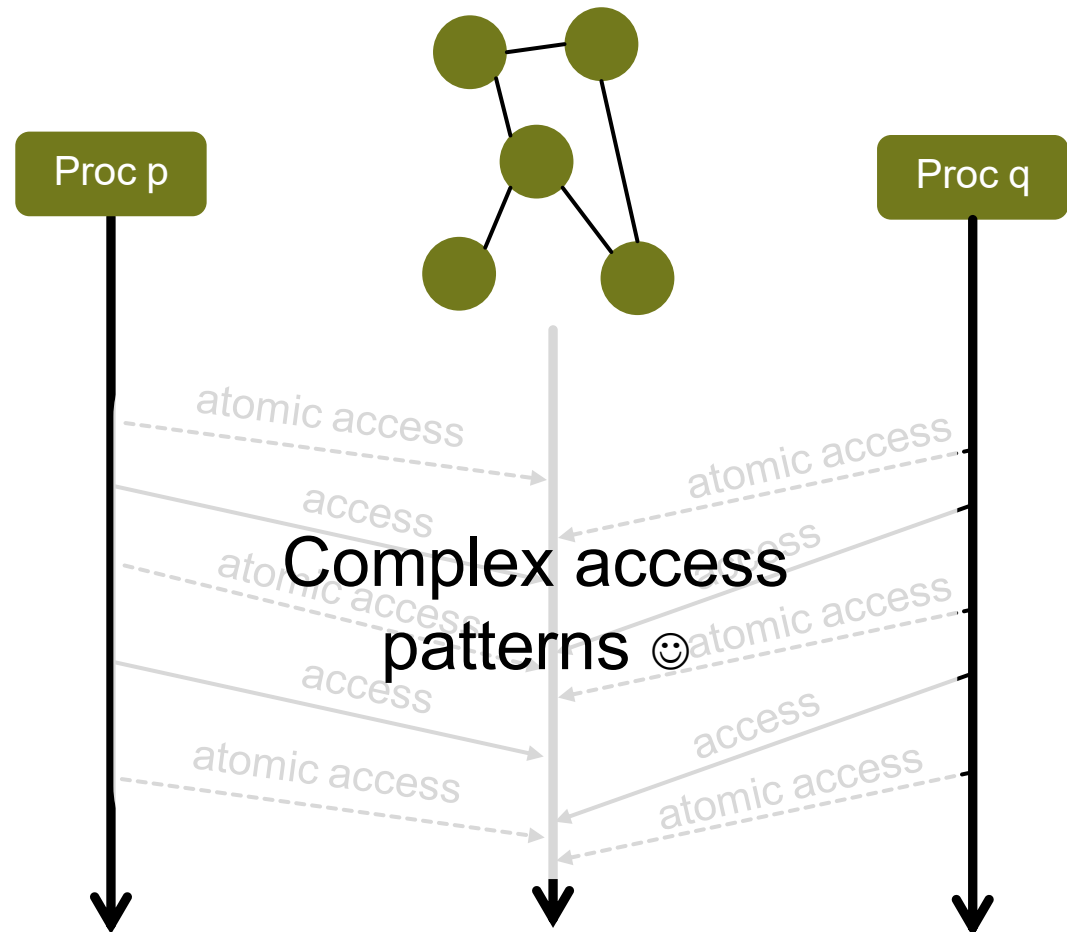
High performance (may be challenging to get)



Complex protocols



Subtle issues (ABA, ...)



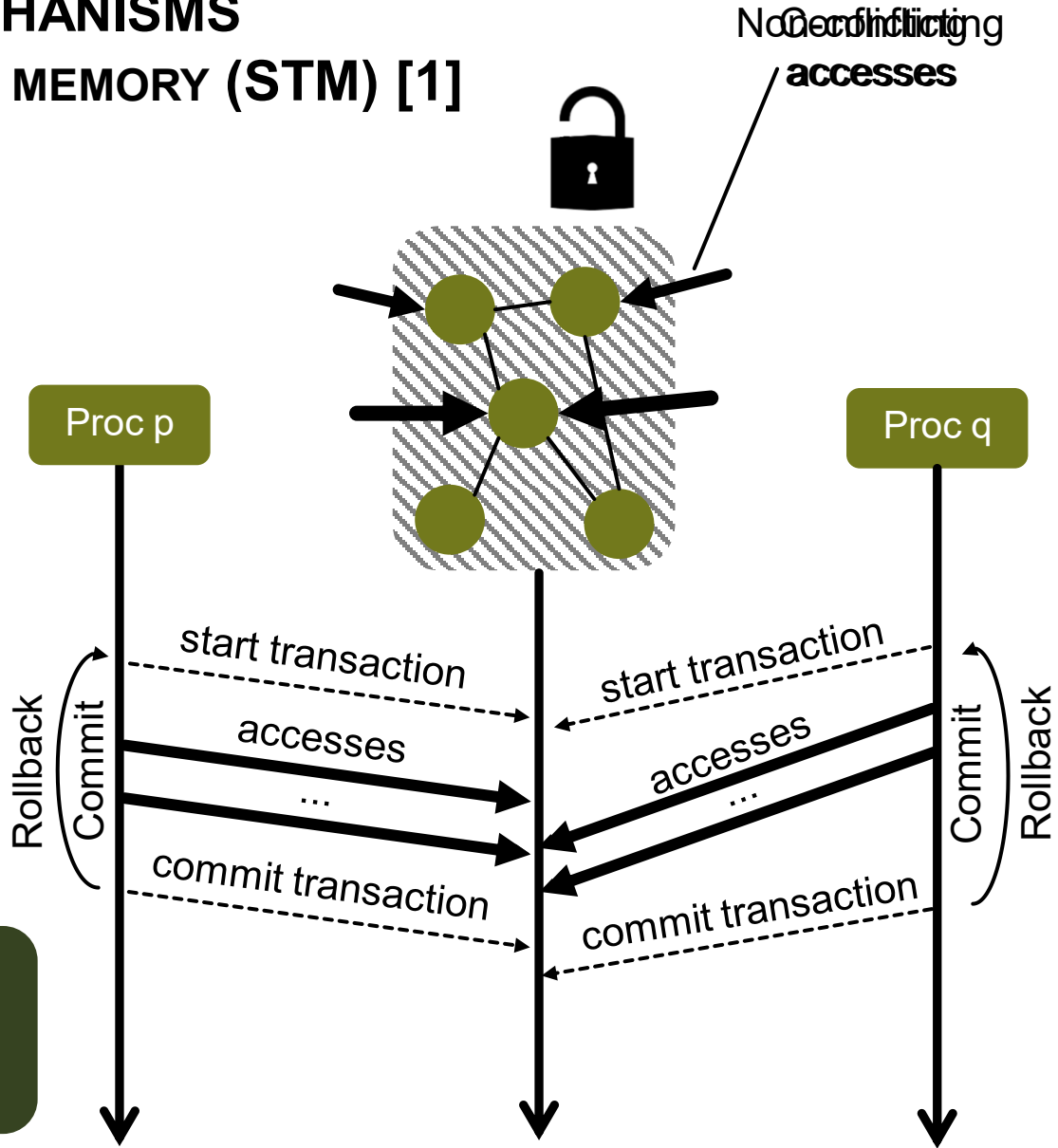
SYNCHRONIZATION MECHANISMS

SOFTWARE TRANSACTIONAL MEMORY (STM) [1]

Conflicts solved with rollbacks and/or serialization.

✗ Software overheads

✓ Simple protocols



[1] N. Shavit and D. Touitou. Software transactional memory. PODC'95.

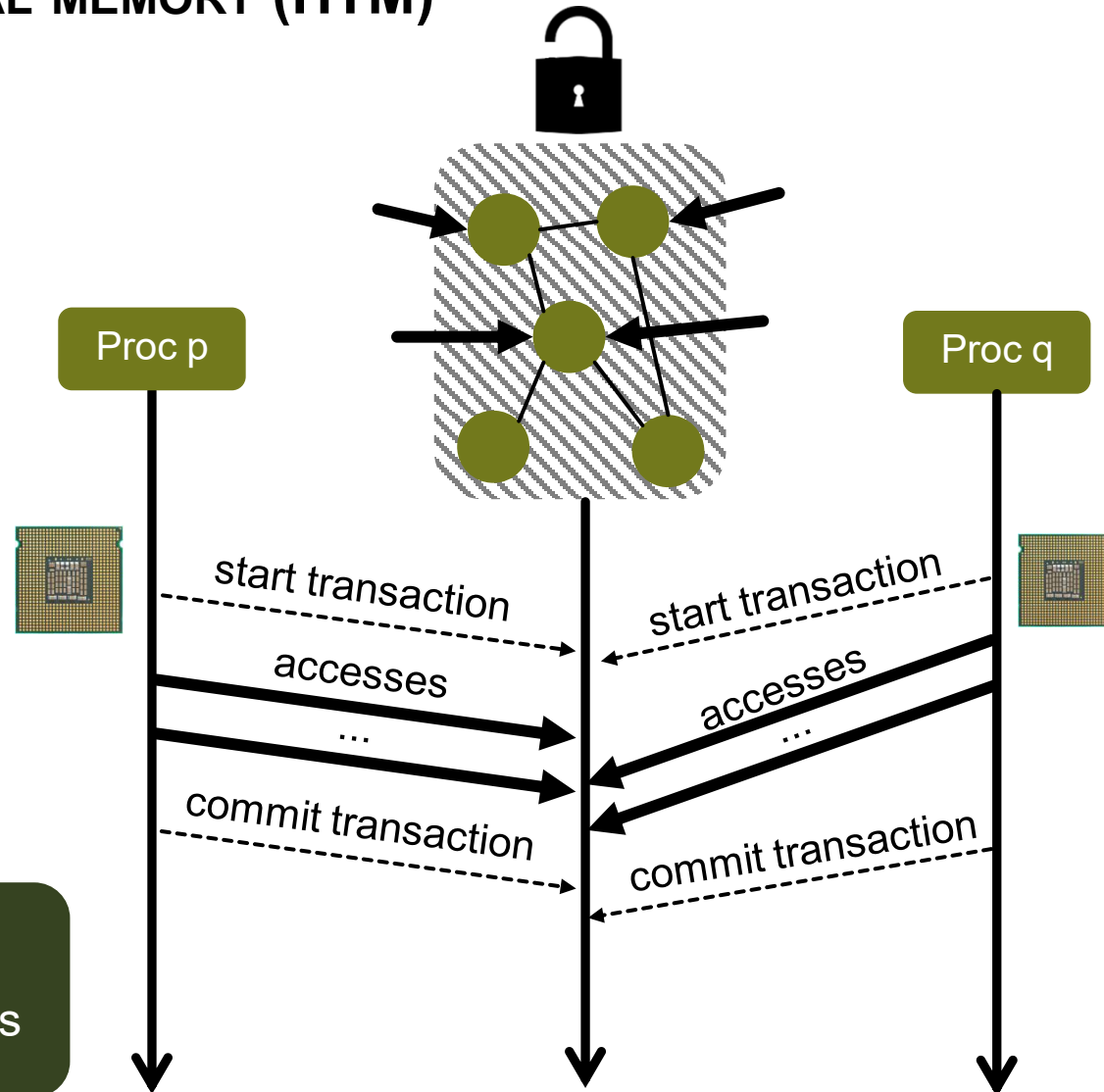
SYNCHRONIZATION MECHANISMS

HARDWARE TRANSACTIONAL MEMORY (HTM)

Conflicts solved with rollbacks and/or HW serialization.

?
High performance?
For graphs?

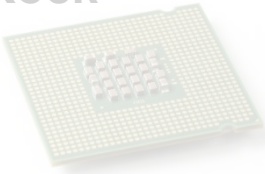
✓
Simple protocols



HARDWARE TRANSACTIONAL MEMORY



Rock



Vega



They offer
programmability, how
about performance?

Haswell



BlueGene/Q



POWER8

ACTIVE MESSAGES (AM)





AM++ [1]

GASNet [2]

Process p

Active message
Z's addr | Payload

Process q

Memory

A's addr: Handler A

...

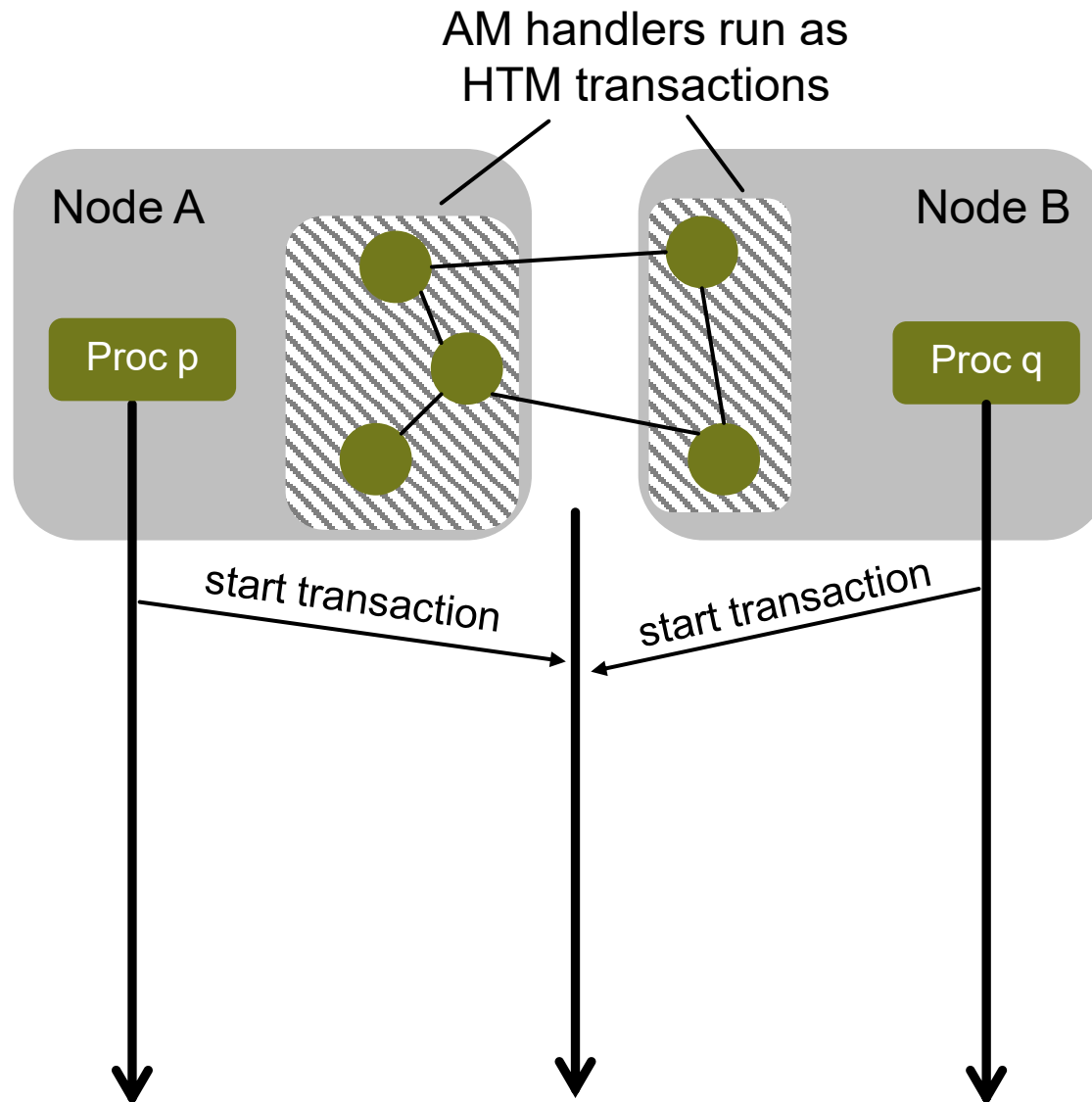
Z's addr: Handler Z



[1] J. J. Willcock et al. AM++: A generalized active message framework. PACT'10.

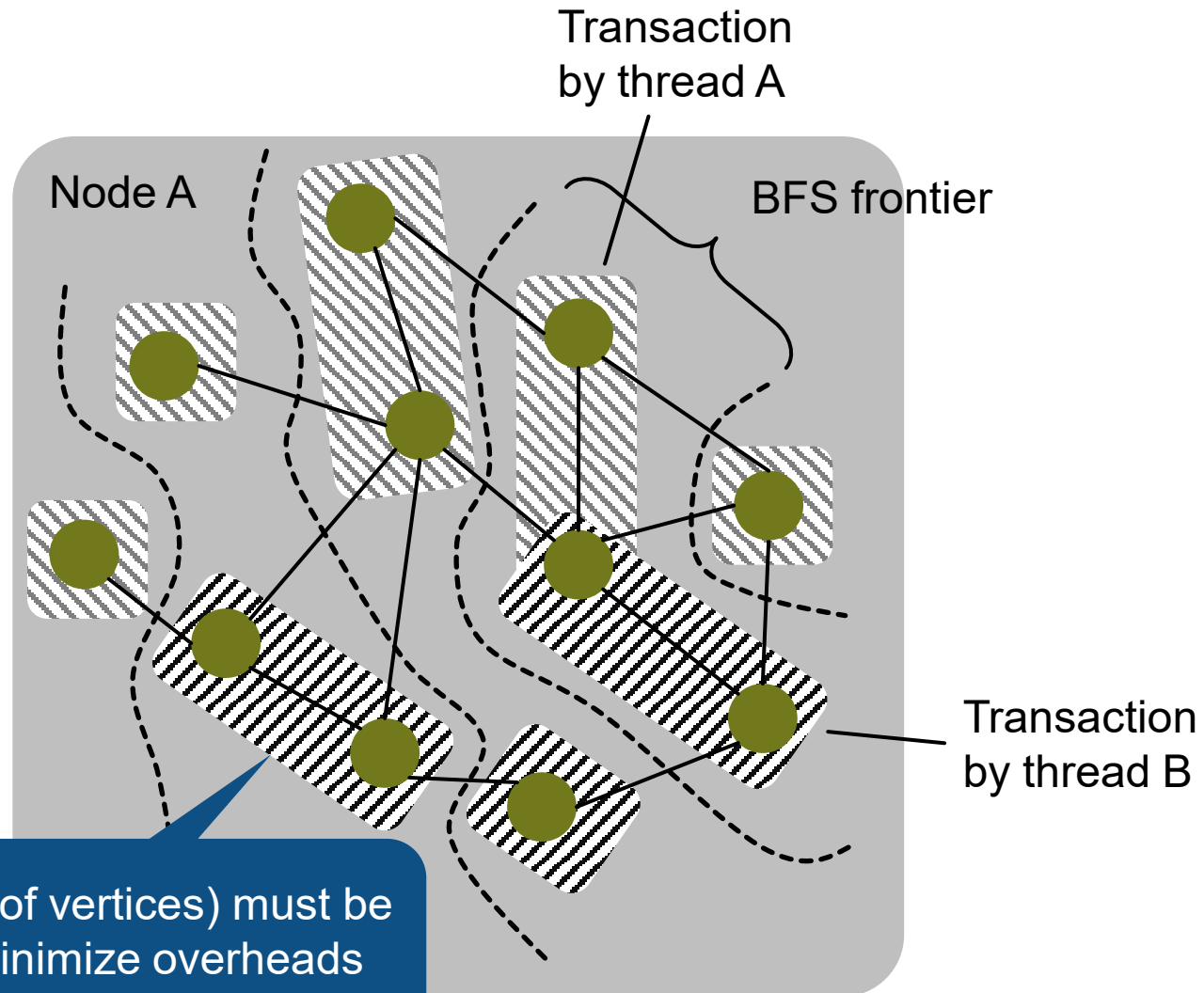
[2] D. Bonachea, GASNet Specification, v1.1. Berkeley Technical Report. 2002.

AM + HTM = ATOMIC ACTIVE MESSAGES

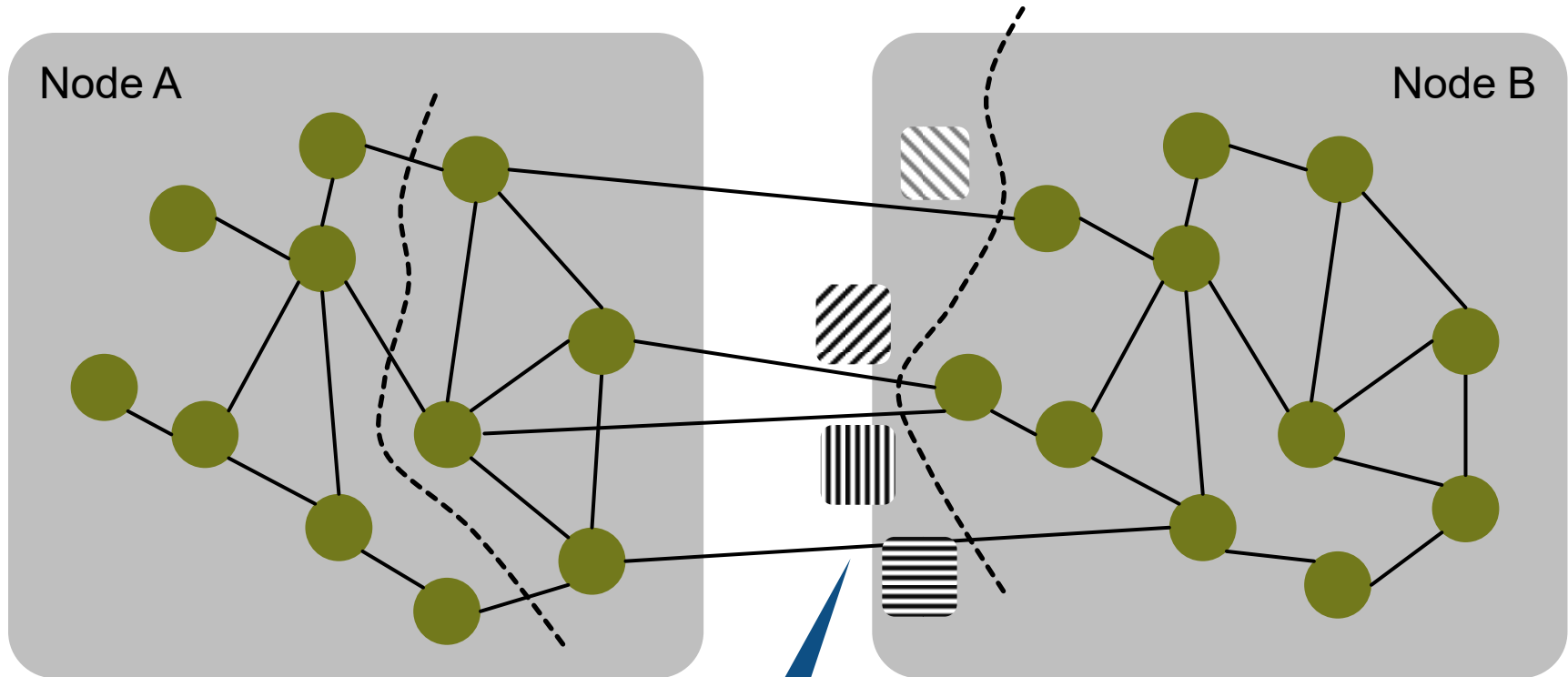


ACCESSING MULTIPLE VERTICES ATOMICALLY

Example: BFS

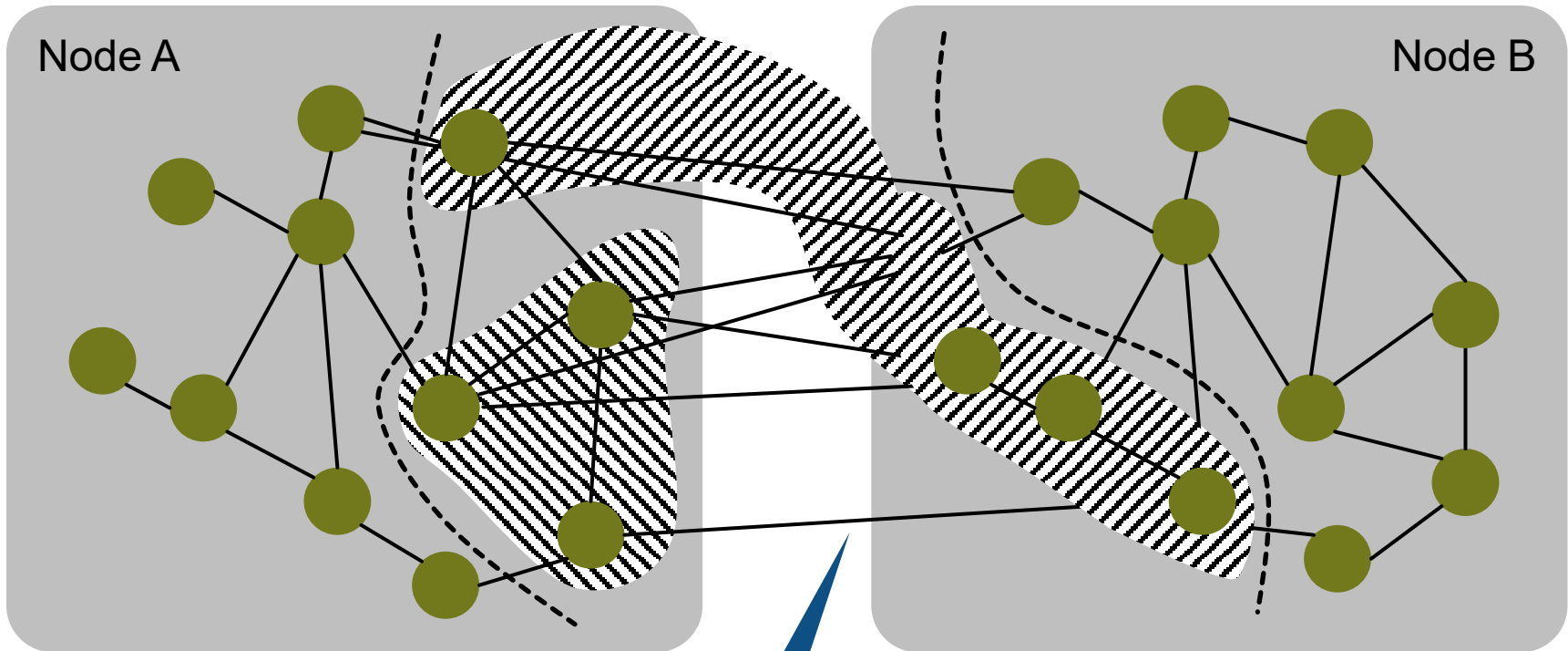


TRANSFERRING TRANSACTIONS ACROSS NODES



Transactions must be appropriately coalesced to minimize communication overheads

EXECUTING TRANSACTIONS ON MULTIPLE NODES



Vertices must be appropriately relocated to enable execution of a hardware transaction

PERFORMANCE ANALYSIS

RESEARCH QUESTIONS



How can we implement AAM handlers to run most efficiently?



What are performance tradeoffs related to HTM?



What are advantages of HTM over atomics for AAM?



What are the optimal transaction sizes?

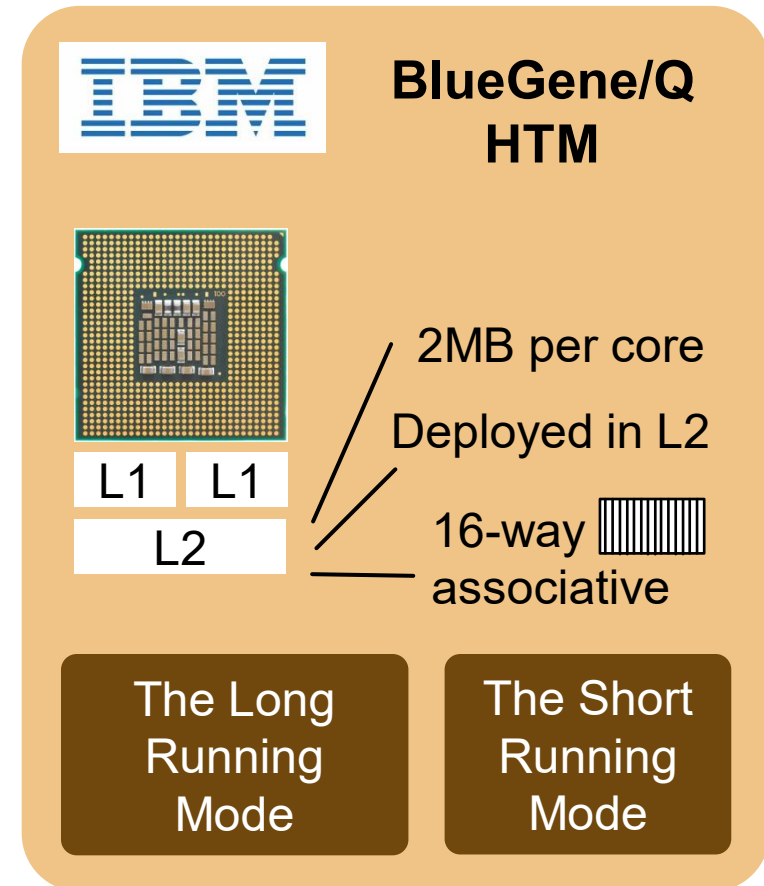
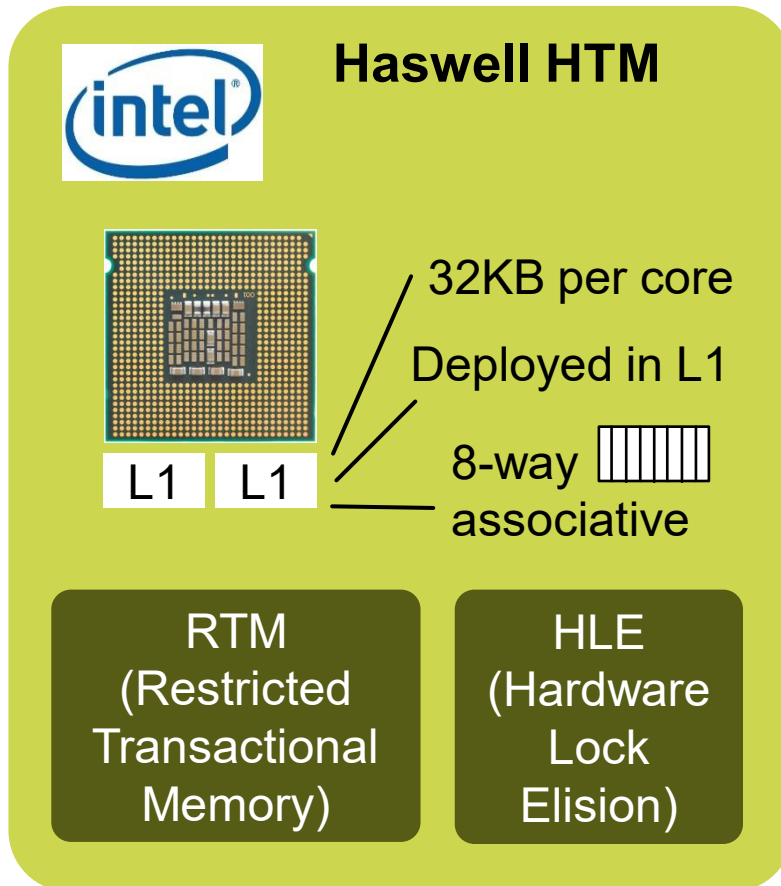
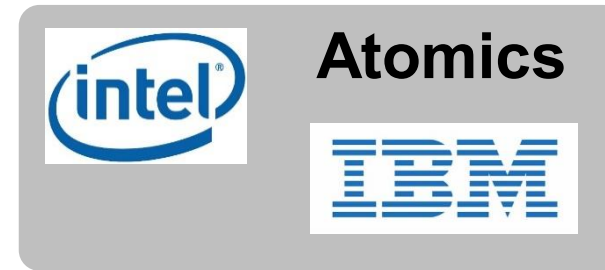
PERFORMANCE ANALYSIS

TYPES OF MACHINES

- Evaluation on 3 machines
 - Intel Haswell server
 - InfiniBand cluster
 - IBM BlueGene/Q



PERFORMANCE ANALYSIS CONSIDERED MECHANISMS



SINGLE-VERTEX TRANSACTIONS

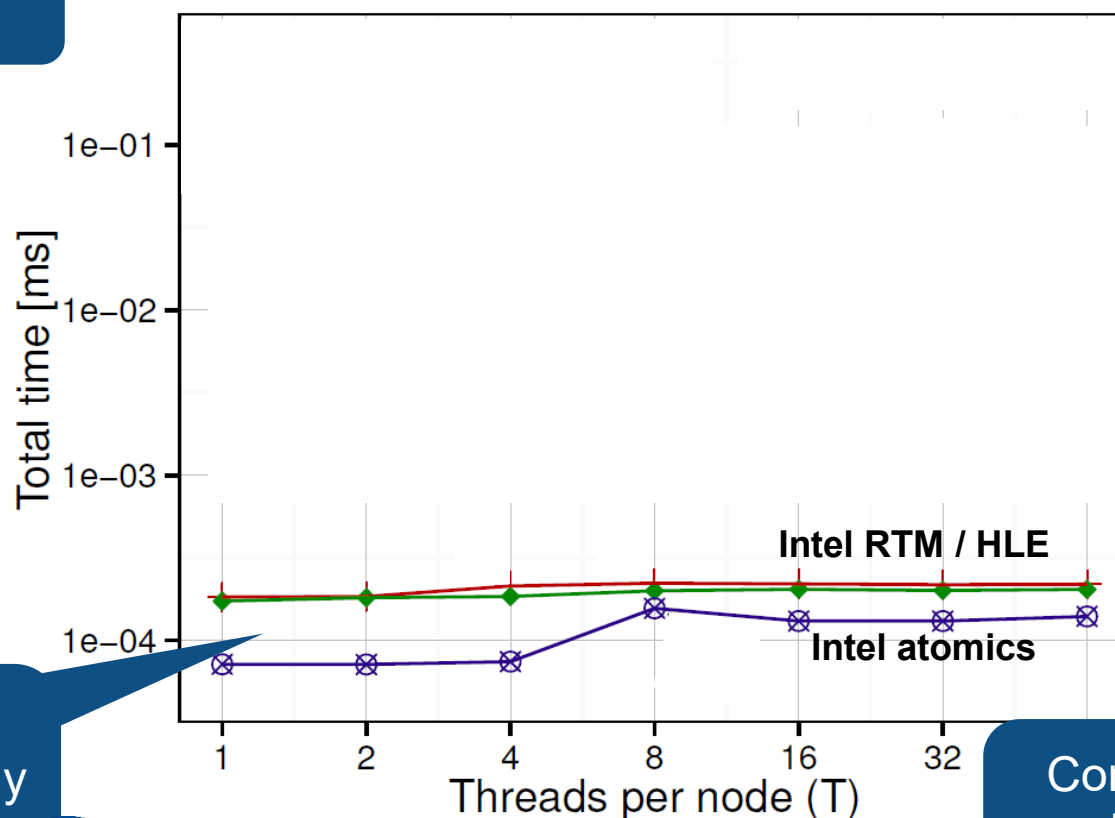
MARKING A VERTEX AS VISITED

Used in BFS,
SSSP, ...

Very few
aborts

Lower contention
(10 racing accesses/vertex)

```
// start handler
if(!v.visited) {
  v.visited = 1;
}
// finish handler
```



Atomics
(CAS) slightly
faster than
HTM

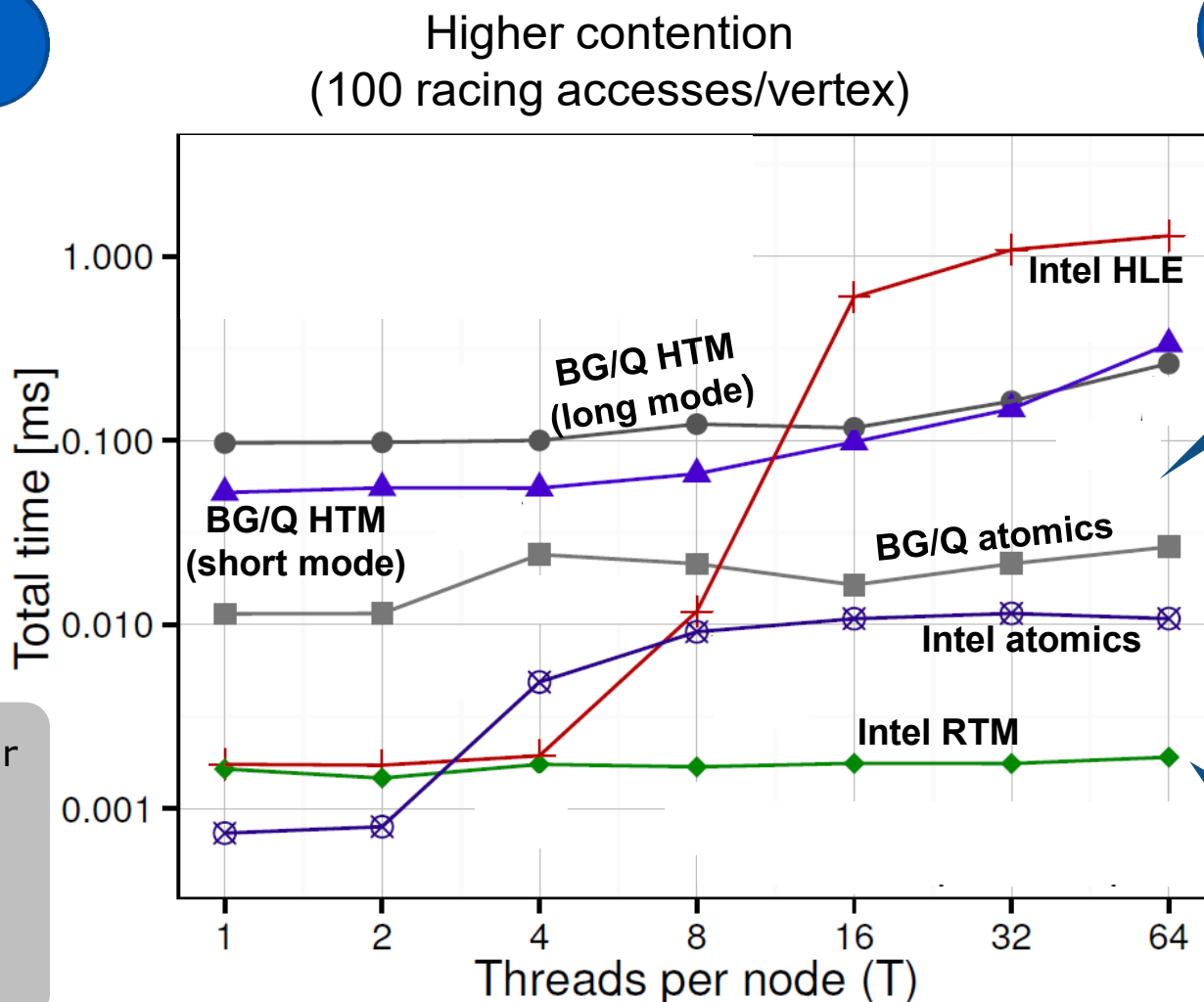
Commit
overheads
dominate

SINGLE-VERTEX TRANSACTIONS

MARKING A VERTEX AS VISITED

Used in BFS,
SSSP, ...

Still very
few
aborts



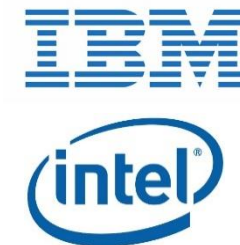
BG/Q
HTM still
worse (L1
vs L2
matters!)

RTM
better
than
atomics

```
// start handler
if(!v.visited) {
  v.visited = 1;
}
// finish
handler
```


SINGLE-VERTEX TRANSACTIONS INCREMENTING VERTEX RANK

Used in
PageRank



```
// start handler  
v.rank++;  
// finish handler
```



Atomics always
outperform HTM



The reason: each transaction always modifies some
memory cell, increasing the number of conflicts

PERFORMANCE MODEL

ATOMICS VS TRANSACTIONS

Time to modify N vertices with atomics:

$$T_{AT}(N) = A_{AT}N + B_{AT}$$

Overhead per vertex

Startup overheads

Time to modify N vertices with a transaction

$$T_{HTM}(N) = A_{HTM}N + B_{HTM}$$

Overhead per vertex

Startup overheads

We predict that:

$$B_{AT} < B_{HTM}$$

$$A_{AT} > A_{HTM}$$

Transactions' cost grows slower

Transaction startup overheads dominate

PERFORMANCE MODEL

ATOMICS VS TRANSACTIONS

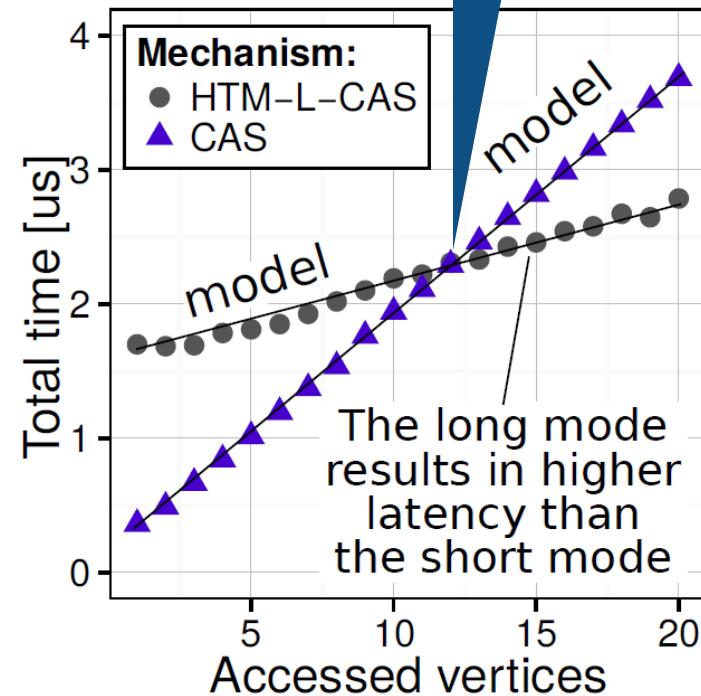
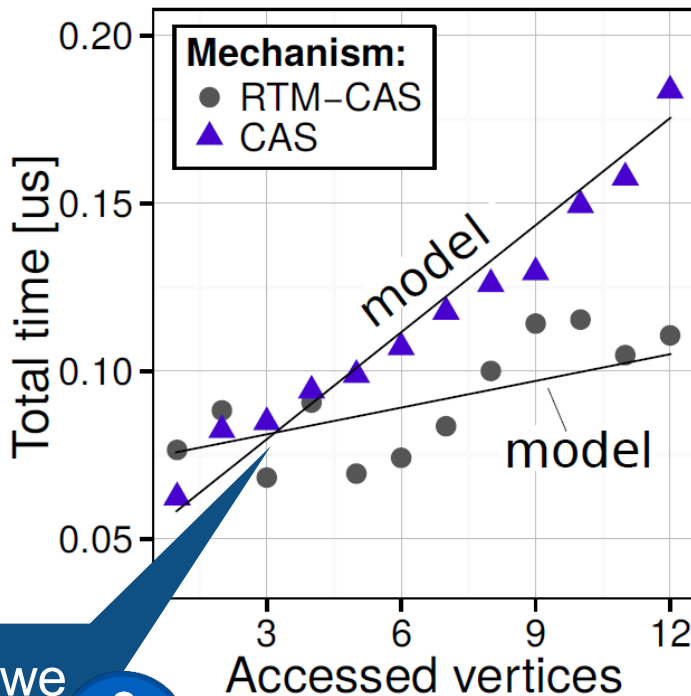
- Can we amortize HTM startup/commit overheads with larger transaction sizes?

Indeed:

$$B_{AT} < B_{HTM}$$

$$A_{AT} > A_{HTM}$$

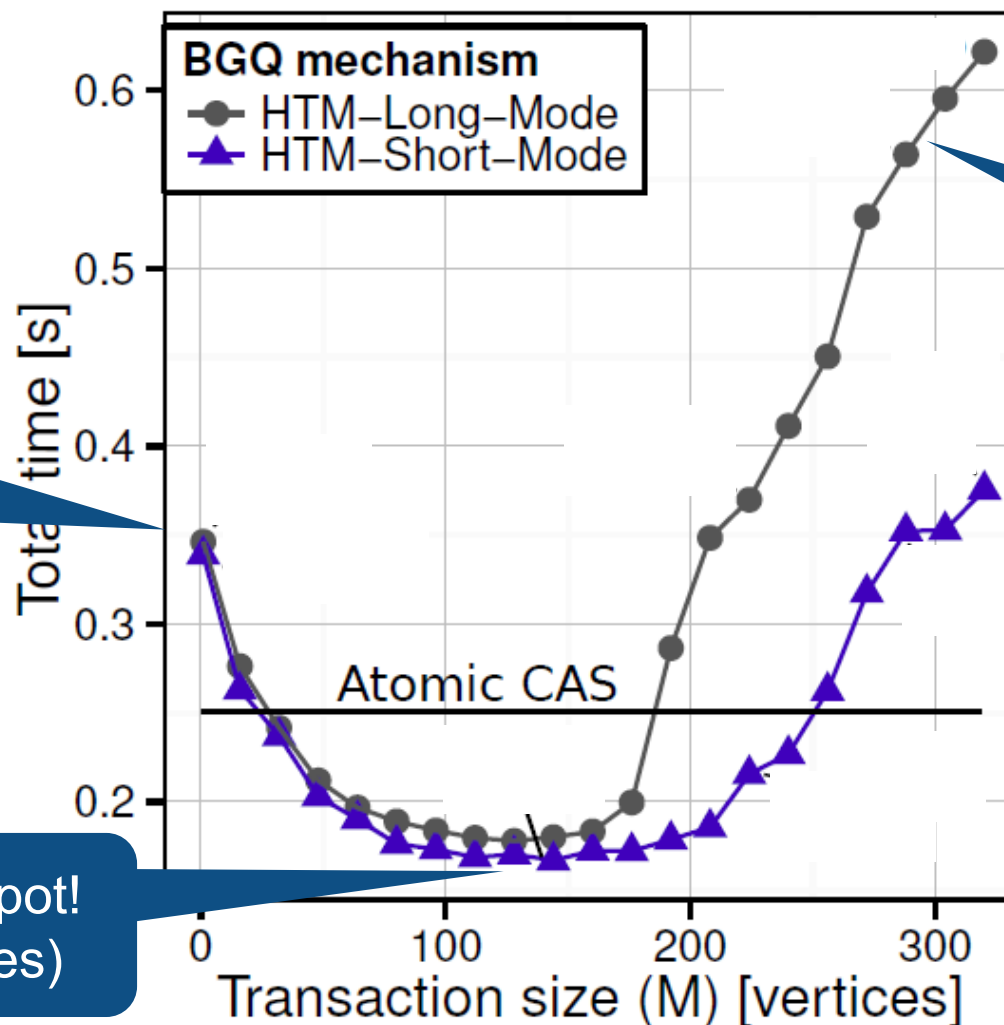
! Yes, we can!



! Yes, we can!

MULTI-VERTEX TRANSACTIONS

MARKING VERTICES AS VISITED

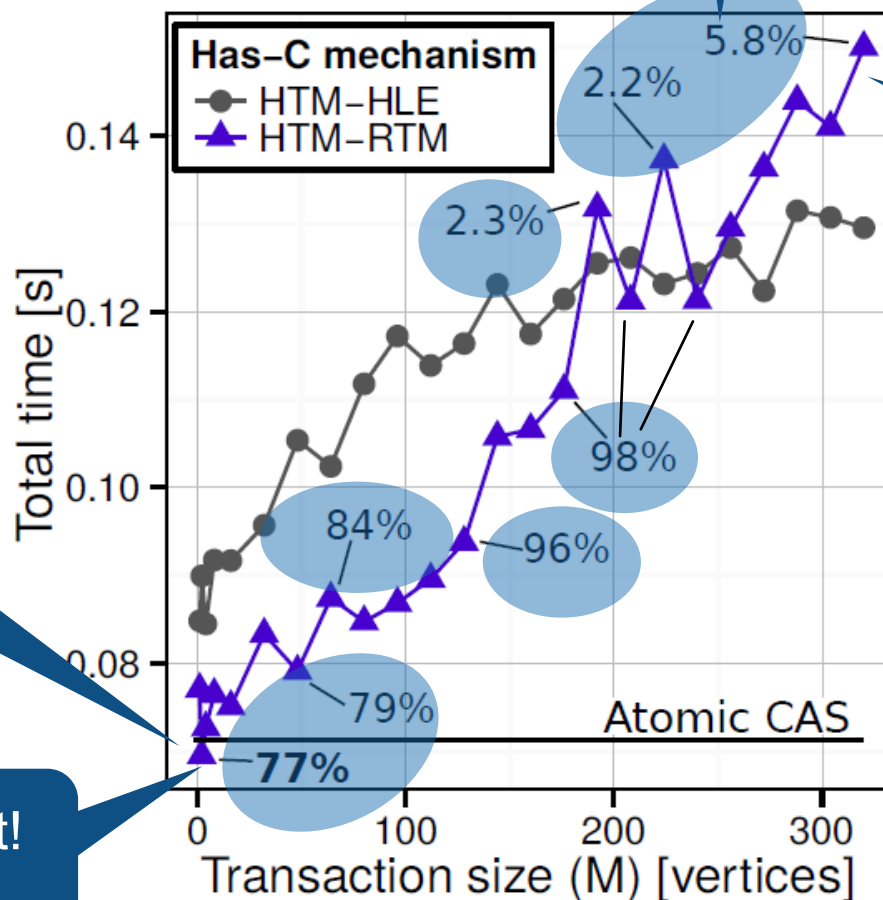


! Startup and commit overheads

! The sweetspot!
(144 vertices)

! Abort and rollback overheads

MULTI-VERTEX TRANSACTIONS MARKING VERTICES AS VISITED



Numbers: % of aborts due to HTM capacity overflows

Startup and commit overheads

The sweetspot! (2 vertices)

Abort and rollback overheads

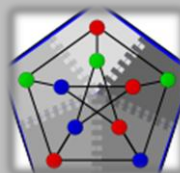
Majority of aborts are due to HTM capacity overflows (small cache size & associativity)

EVALUATION CONSIDERED ENGINES



[1] Hand-tuned
algorithm-specific
codes

PBGL [4]



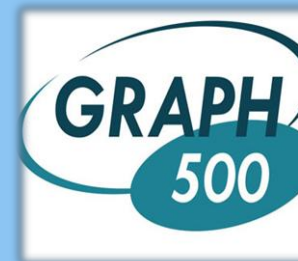
Distributed
HPC libraries

Galois

[2] Runtimes that exploit
amorphous data-parallelism

AAM +

Improving
Graph500
design



HAMA [3]



Hadoop-based
BSP engines

[1] R. Murphy et al. Introducing the Graph 500. CUG'10.

[2] M. Kulkarni et al. Optimistic Parallelism Requires Abstractions. PLDI'07.

[3] S. Seo et al. HAMA: An Efficient Matrix Computation with the MapReduce Framework. CLOUDCOM'10.

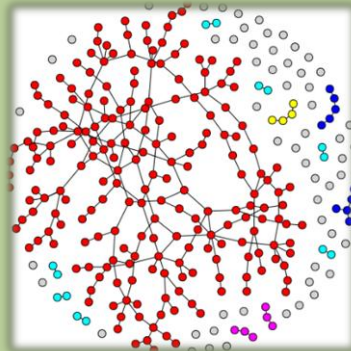
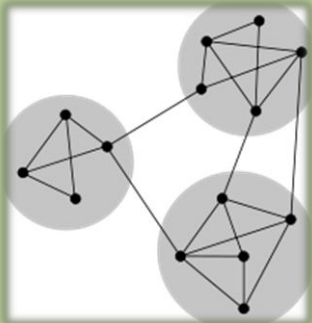
[4] D. Gregor and A. Lumsdaine. The parallel BGL: A generic library for distributed graph computations. POOSC'05.

EVALUATION

CONSIDERED TYPES OF GRAPHS

Synthetic graphs

Kronecker [1]



Erdős-Rényi [2]

Real-world SNAP graphs [3]

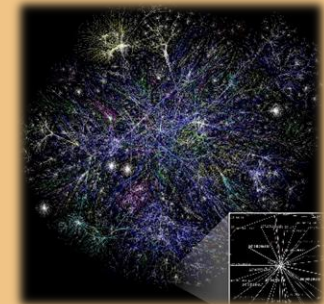


Road networks

Social networks

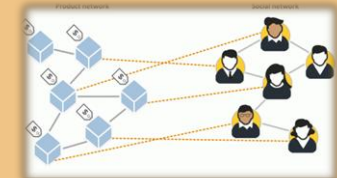
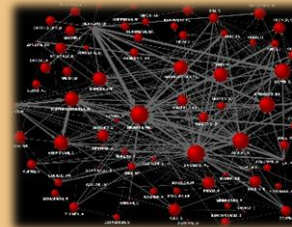


Comm. graphs



Web graphs

Citation graphs



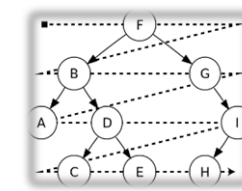
Purchase networks

[1] J. Leskovec et al. Kronecker Graphs: An Approach to Modeling Networks. J. Mach. Learn. Research. 2010.

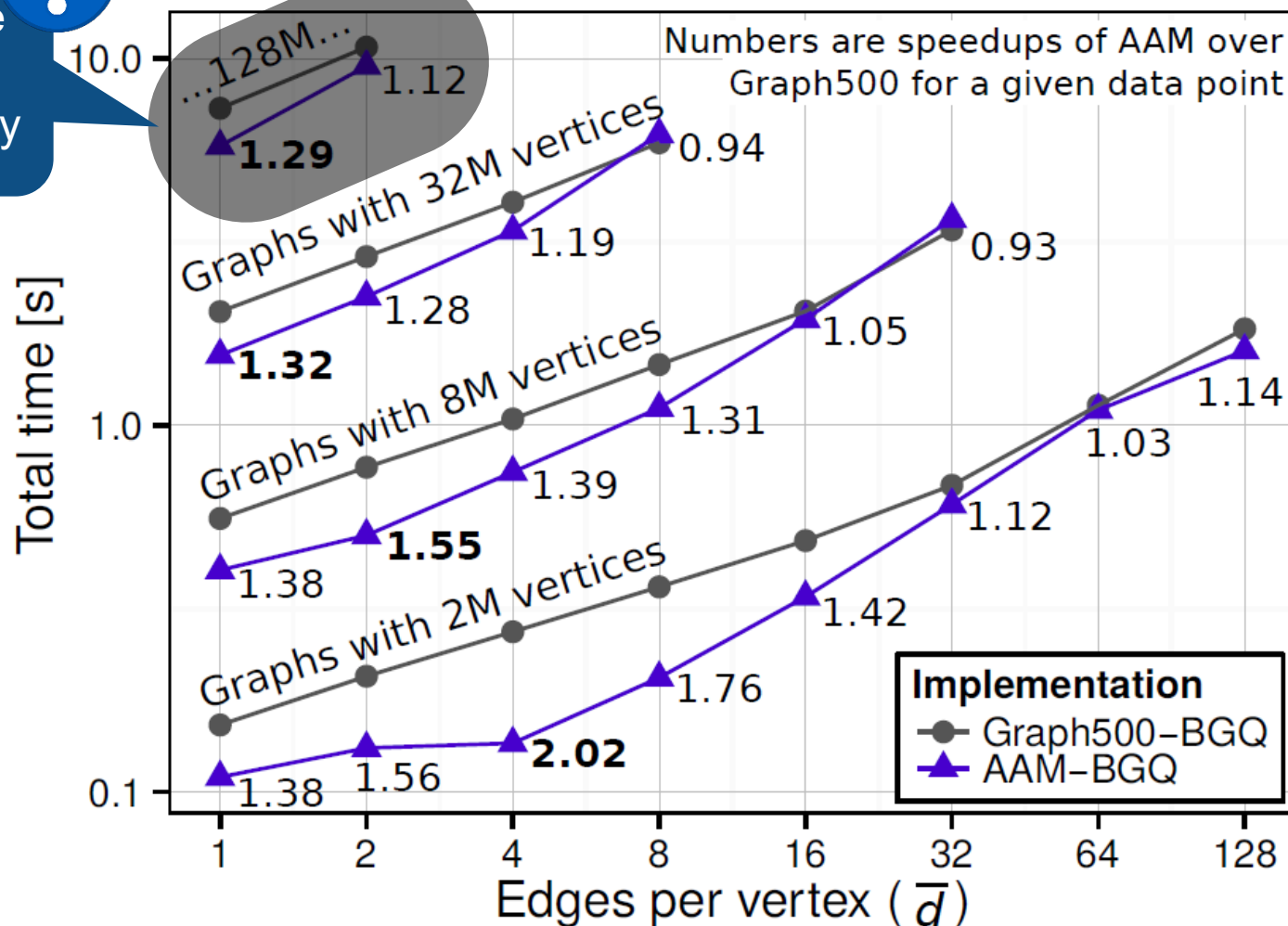
[2] P. Erdos and A. Renyi. On the evolution of random graphs. Pub. Math. Inst. Hun. A. Science. 1960.

[3] <https://snap.stanford.edu>

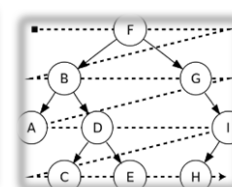
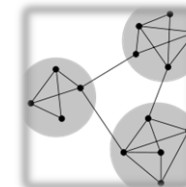
ACCELERATING STATE-OF-THE-ART GRAPH500 + AAM (BLUEGENE/Q)



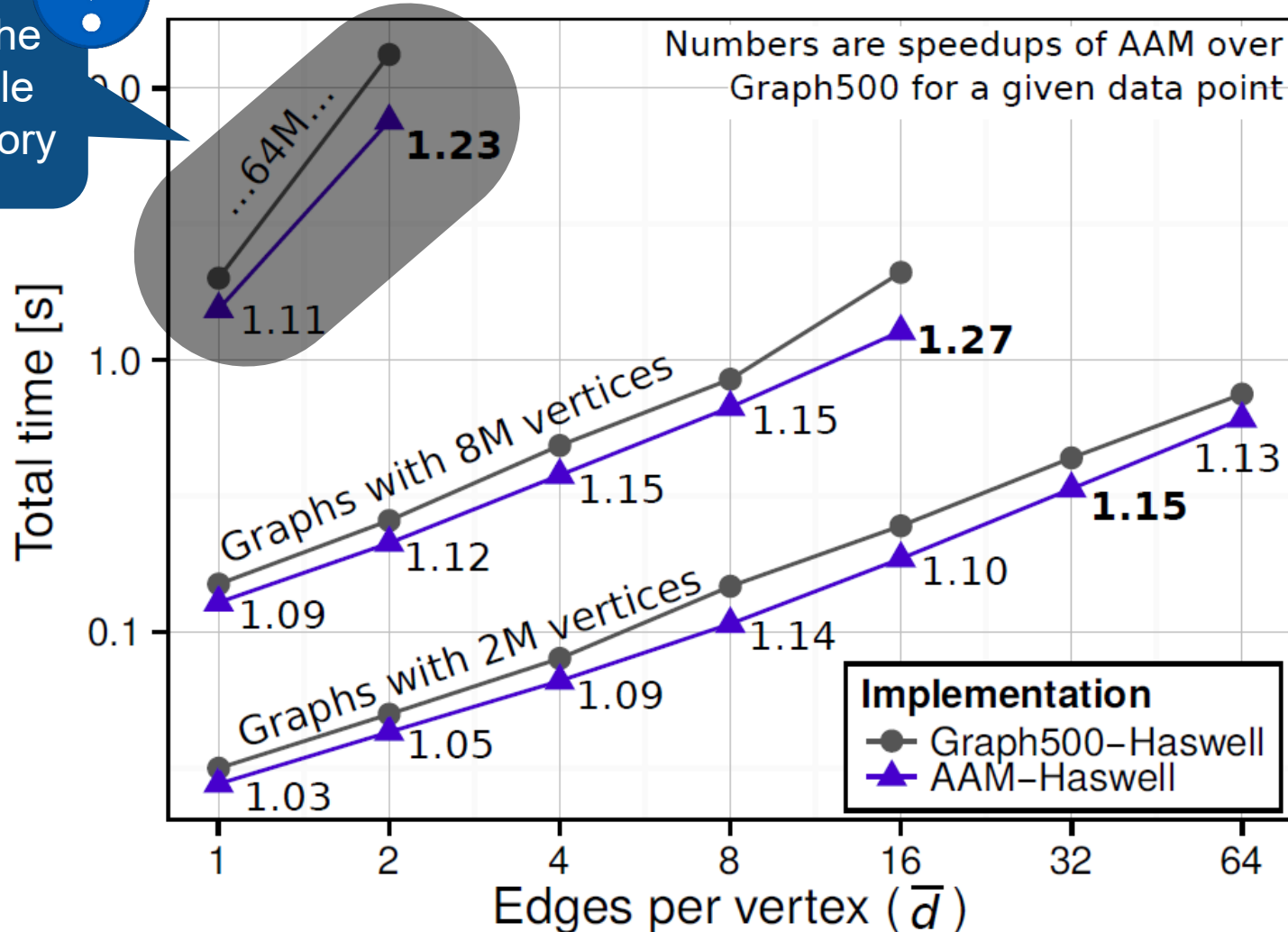
Fill the
whole
memory



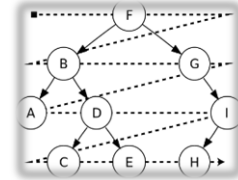
ACCELERATING STATE-OF-THE-ART GRAPH500 + AAM (HASWELL)



Fill the
whole
memory



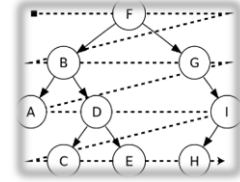
OUTPERFORMING STATE-OF-THE-ART



Input graph properties				BG/Q analysis				Haswell analysis					
Type	ID	Name	N	E	5 cores gCPU ($M = 24$)	M	5 cores gCPU	5 cores gCPU ($M = 2$)	5 cores Galois ($M = 2$)	M	5 cores gCPU	5 cores Galois	5 cores FLAMA
E-commerce networks (10k)	wWT	wiki-Talk	2.4M	5M	2.82	45	1.35	0.91	1.22	8	1.06	1.25	344
	zED	zoo-EDuAll	265k	120k	3.07	32	1.36	0.93	0.93	1	0.97	1.15	1445
	zLV	zoo-LVcs	1.6M	984	2.16	13	1.35	1.05	1.16	8	1.05	1.13	307
Road networks (9k)	rCA	roadNet-CA	1.9M	5.5M	<1	2	1.59	1.33	1.74	8	1.36	1.80	>10 ⁵
	rTX	roadNet-TX	1.3M	3.8M	<1	2	1.53	1.29	1.89	6	1.42	2.08	>10 ⁵
	rPA	roadNet-PA	1M	3M	<1	2	1.59	<1	2.00	9	1.07	2.16	>10 ⁵
Citation graphs (65k)	ciP	cit-Patents	3.7M	16.5M	1.16	8	1.67	1.01	1.26	2	1.01	1.26	1875
Web graphs (70k)	wGL	web-Google	875k	5.1M	1.78	12	2.08	0.98	1.26	6	1.06	1.35	365
	wBS	web-BerkStan	685k	7.6M	1.91	24	1.91	0.93	1.31	5	1.07	1.49	759
	wSP	web-Stanford	281k	2.3M	1.89	24	1.89	0.98	1.54	5	1.07	1.58	1077

😊 No, you don't have to read it. All details are in the paper. Here: just a summary.

OUTPERFORMING STATE-OF-THE-ART



Average overall speedup (geometric mean) over Graph 500: 1.07,
Galois: 1.40, HAMA ~1000

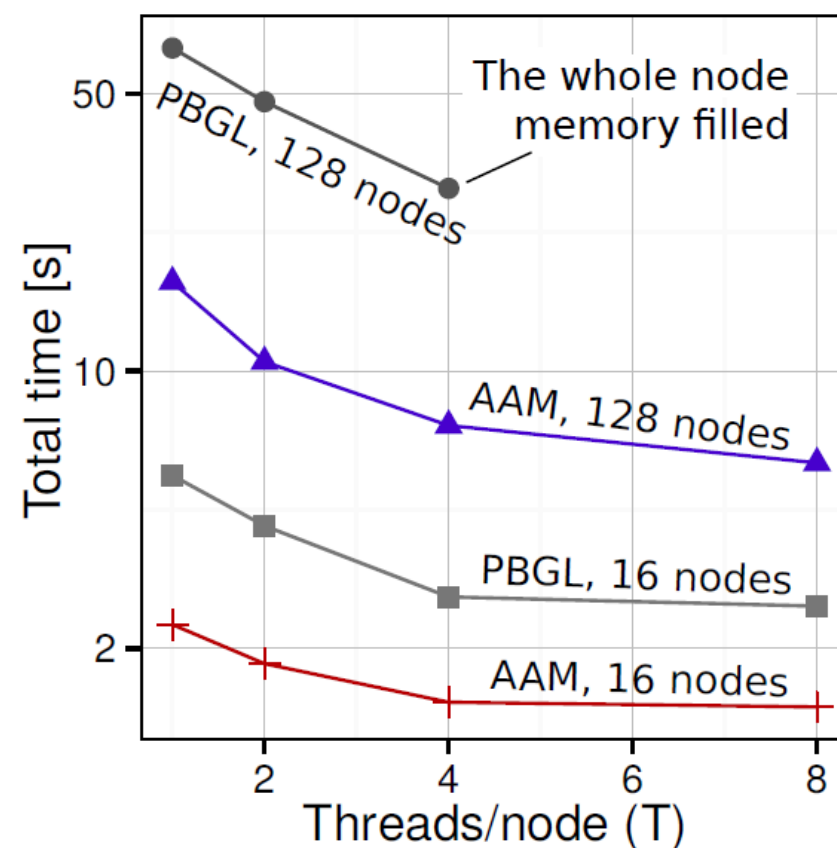
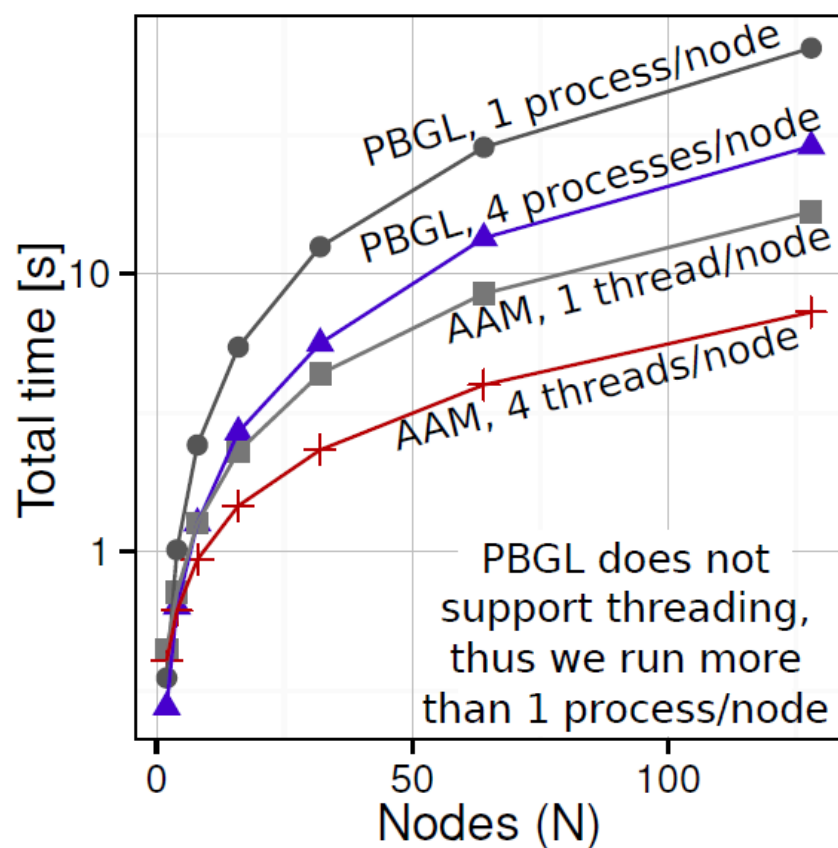
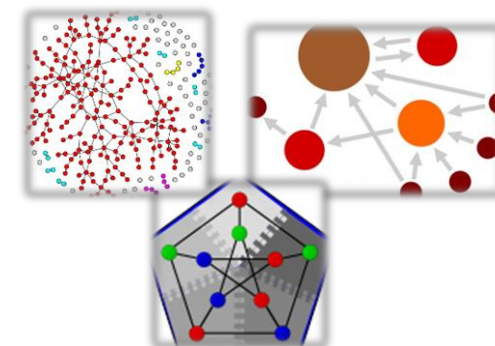


1.85x on average, up to 4.3x

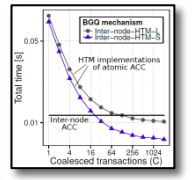
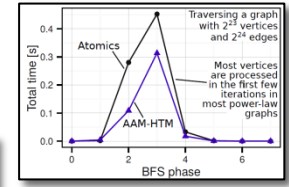
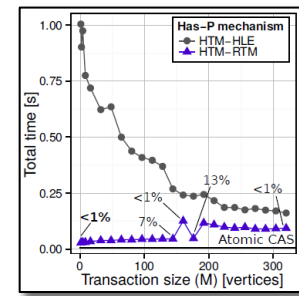
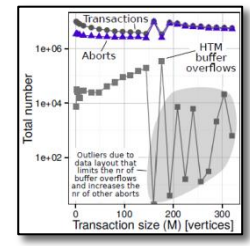
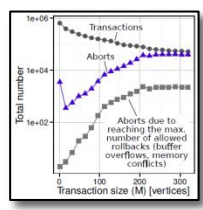
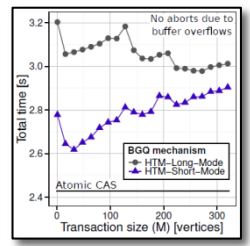
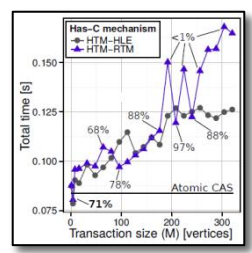
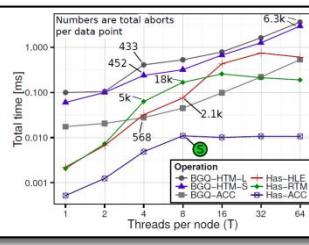
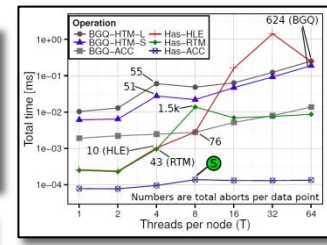
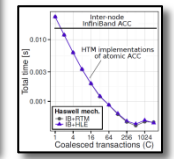
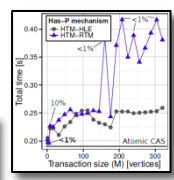
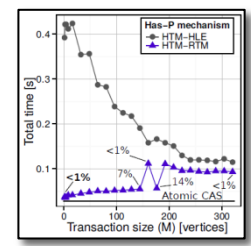
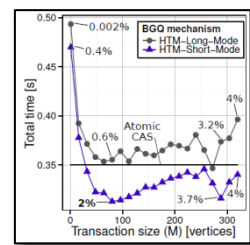
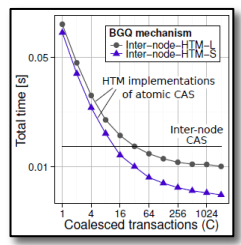
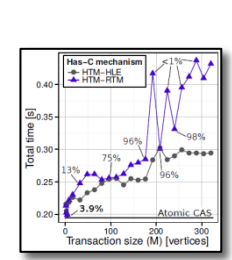



OUTPERFORMING STATE-OF-THE-ART

SCALABILITY ANALYSIS: DISTRIBUTED-MEMORY

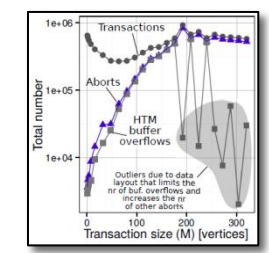
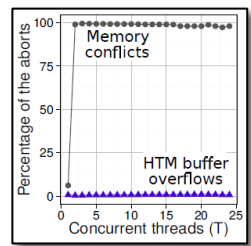
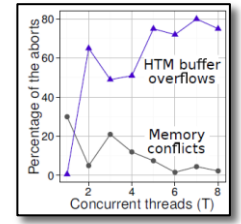
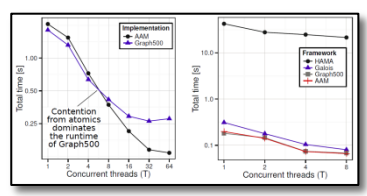


OTHER ANALYSES



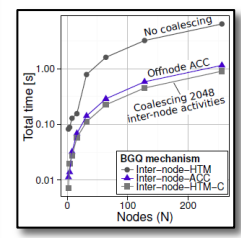
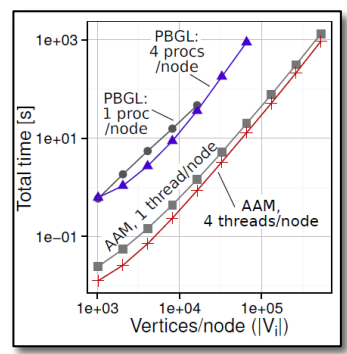
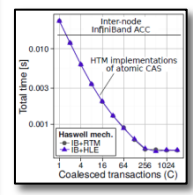
Aborts due to:

	Memory conflicts	Buffer overflows	Other reasons
10 ops	Has-RTM: 1,520	BGQ-STM-L: 624	BGQ-STM-S: 623
100 ops	Has-RTM: 18,952	BGQ-STM-L: 6,374	BGQ-STM-S: 6,392

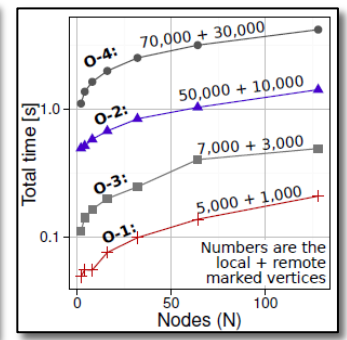


Aborts due to:

	Memory conflicts	Buffer overflows	Other reasons
10 ops	Has-RTM: 2	BGQ-STM-L: 802	BGQ-STM-S: 1,118
100 ops	Has-RTM: 2	BGQ-STM-L: 1,539	BGQ-STM-S: 2,242



Input graph properties				BG/Q analysis			Haswell analysis						
Type	ID	Name	V	E	S over g500 (M = 24)	M	S over g500 (M = 2)	S over Galois (M = 2)	M	S over g500	S over Galois	S over HAMA	
Comm. networks (CNs)	cWT	wiki-Talk	2.4M	5M	2.82	48	3.35	0.91	1.22	6	0.96	1.28	344
	cEU	email-EuAll	265k	420k	3.67	32	4.36	0.76	0.88	4	0.97	1.12	1448
	sLV	soc-Lived.	4.8M	69M	1.44	12	1.56	1.05	1.1	3	1.07	1.12	> 10 ⁴
Social networks (SNs)	sOR	com-orkut	3M	117M	1.22	20	1.27	1.06	0.69	4	1.13	0.74	> 10 ⁴
	sLJ	com-lj	4M	34M	1.44	12	1.54	1.03	1.03	4	1.04	1.04	603
	sYT	com-youtube	1.1M	2.9M	1.67	8	1.84	0.96	1.1	5	0.98	1.11	670
	sDB	com-dblp	317k	1M	1.33	8	1.80	=1	2.5	2	=1	2.53	2160
	sAM	com-amazon	334k	925k	1.14	8	1.62	1.04	1.64	2	1.04	1.64	1426
Purchase network (PNs)	pAM	amazon0601	403k	3.3M	1.45	8	1.91	=1	1.25	3	1.03	1.30	618
	rCA	roadNet-CA	1.9M	5.5M	=1	2	1.59	1.33	1.74	8	1.38	1.80	> 10 ⁴
	rTX	roadNet-TX	1.3M	3.8M	=1	2	1.53	1.29	1.89	6	1.42	2.08	> 10 ⁴
Road networks (RNs)	rPA	roadNet-PA	1M	3M	=1	2	1.52	=1	2.00	9	1.07	2.16	> 10 ⁴
	cIP	cit-Patents	3.7M	16.5M	1.16	8	1.57	1.01	1.26	2	1.01	1.26	1875
Web graphs (WGs)	wGL	web-Google	875k	5.1M	1.78	12	2.08	0.98	1.26	6	1.06	1.35	365
	wBS	web-BerkStan	685k	7.6M	1.91	24	1.91	0.93	1.31	5	1.07	1.40	755
	wSF	web-Stanford	281k	2.3M	1.89	24	1.89	0.98	1.54	5	1.07	1.58	1077



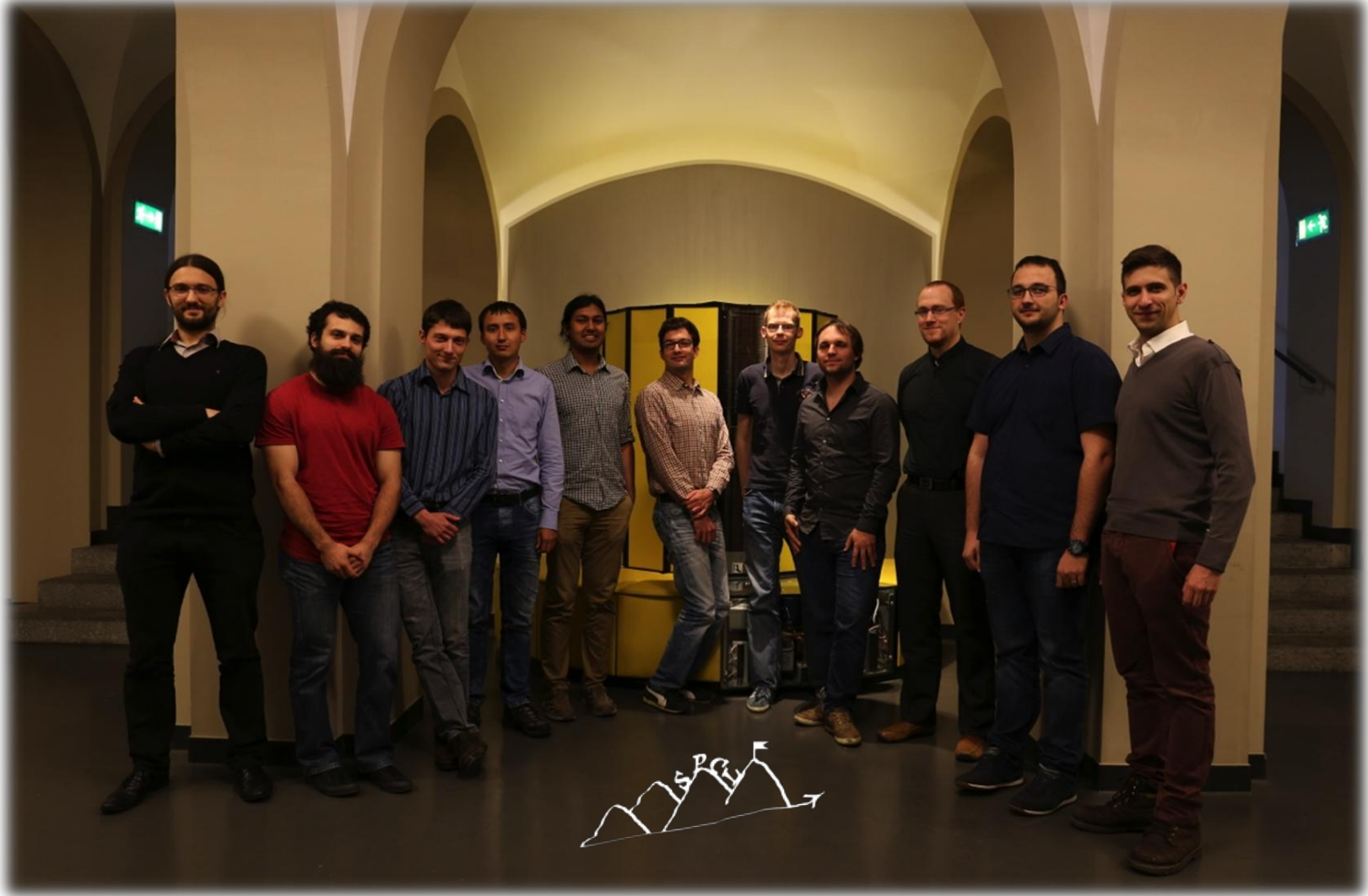
DISCUSSION AND CONCLUSIONS

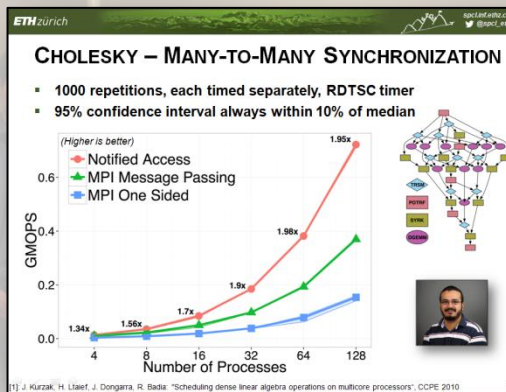
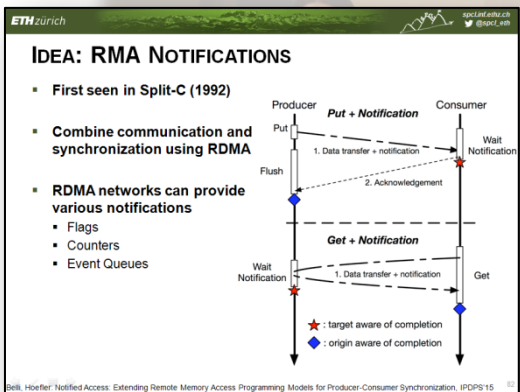
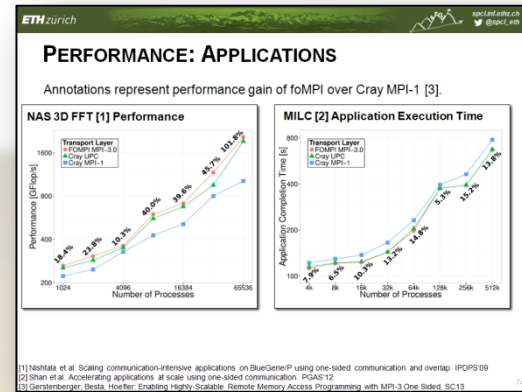
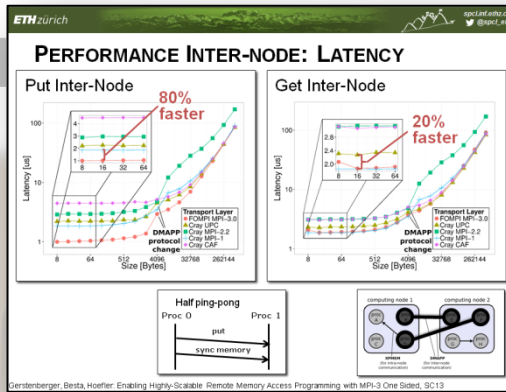
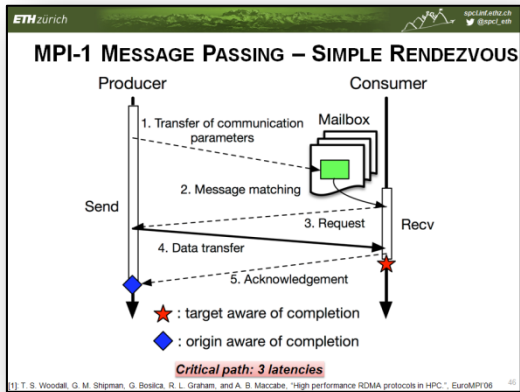
- **MPI-3 RMA [1] standardizes weak remote memory**
 - Builds on existing practice (UPC, CAF, ARMCI etc.)
 - Rich set of synchronization mechanisms
- **Notified Access [2] can support producer/consumer**
 - Maintains benefits of RDMA
- **HTM can accelerate parallel applications [3]**
 - Uses optimistic coarsened irregular parallelism
- **Atomic Active Messages use HTM on DM systems**
 - First steps towards software-emulated TM over RDMA
 - Thinking about hardware support
- **Significant speedups over highly-tuned graph frameworks**
 - Haswell: 7% over Graph 500, 40% over Galois, 1000x over Hama
- **What next? Discussions?**
 - GraphBlas using RMA+HTM?



[1] T. Hoefler, J. Dinan, R. Thakur, B. Barrett, P. Balaji, W. Gropp, K. Underwood: Remote Memory Access Programming in MPI-3, TOPC
[2] Belli, Hoefler: Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization, IPDPS'15
[3] Besta, Hoefler: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, HPDC'15

ACKNOWLEDGMENTS





SCIENTIFIC AND ENGINEERING COMPUTATION SERIES

Using Advanced MPI
Modern Features of the Message-Passing Interface

William Gropp
Torsten Hoefler
Rajeev Thakur
Ewing Lusk

