ROBERTO BELLI, TORSTEN HOEFLER

Notified Access: Extending Remote Memory Access Programming Models for Producer-Consumer Synchronization

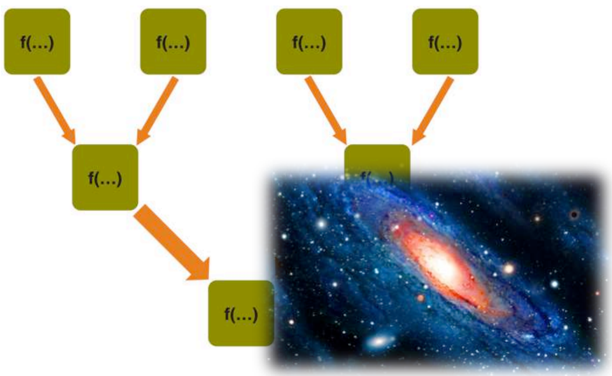
COMMUNICATION IN TODAY'S HPC SYSTEMS

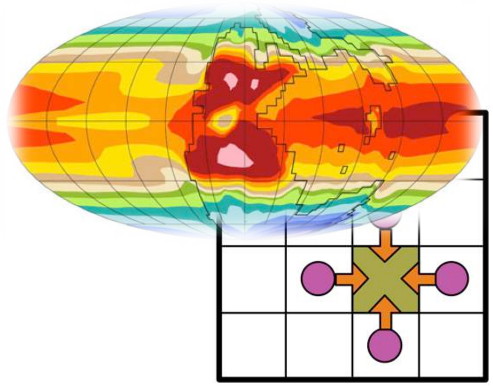
- **The de-facto programming model: MPI-1**
  - Using send/recv messages and collectives

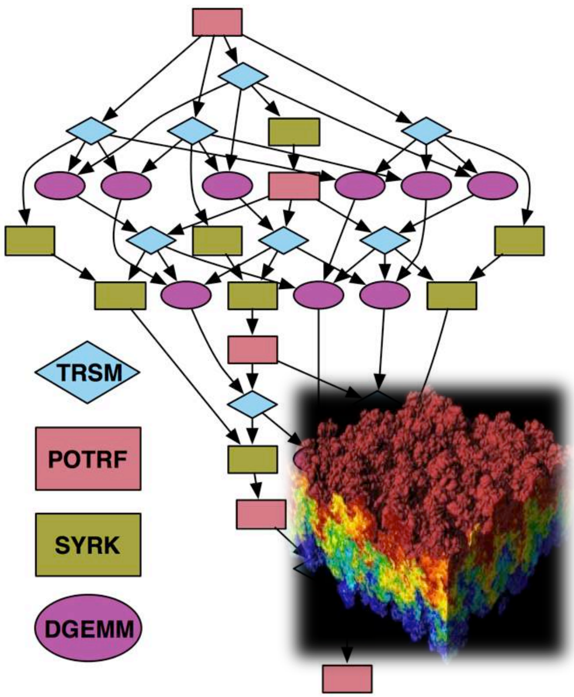
- **The de-facto network standard: RDMA**
  - Zero-copy, user-level, os-bypass, fuzz-bang

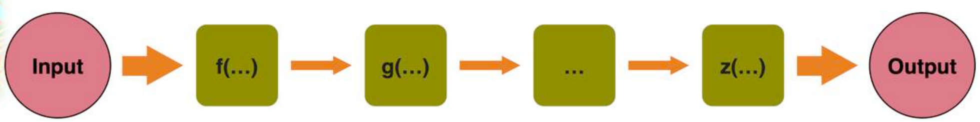
PRODUCER-CONSUMER RELATIONS

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



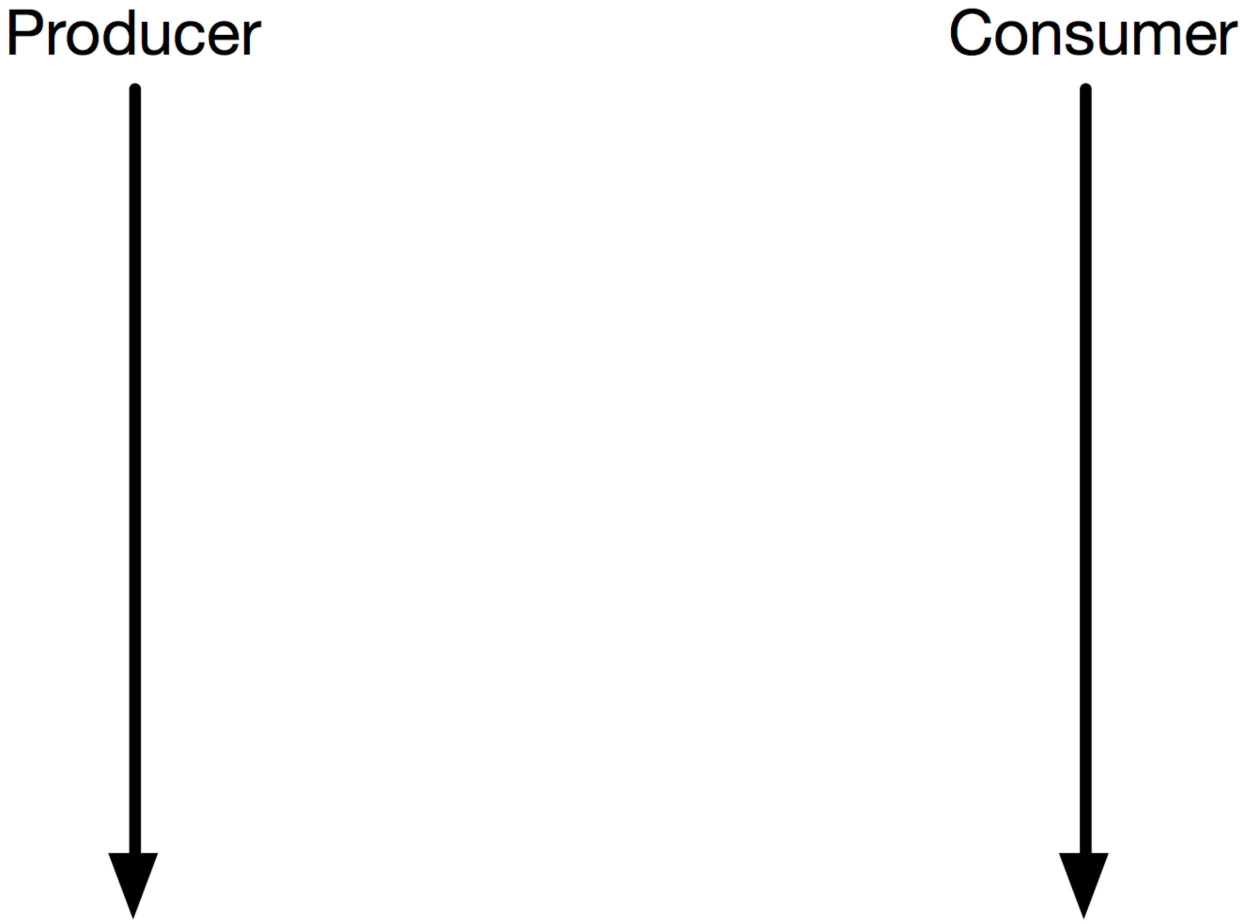






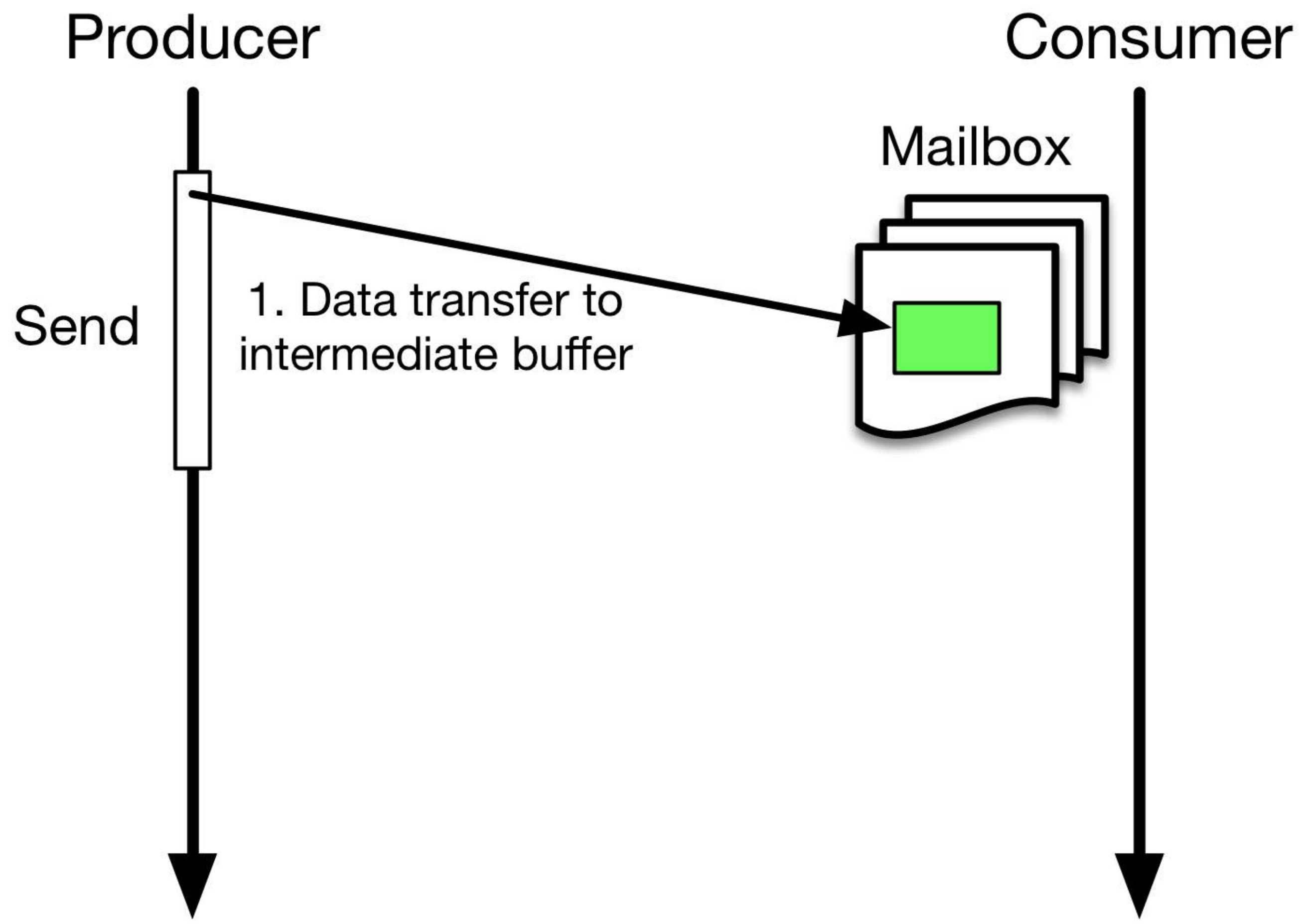
- **Perfectly supported by MPI-1 Message Passing**
  - But how does this actually work over RDMA?

MPI-1 MESSAGE PASSING – SIMPLE EAGER



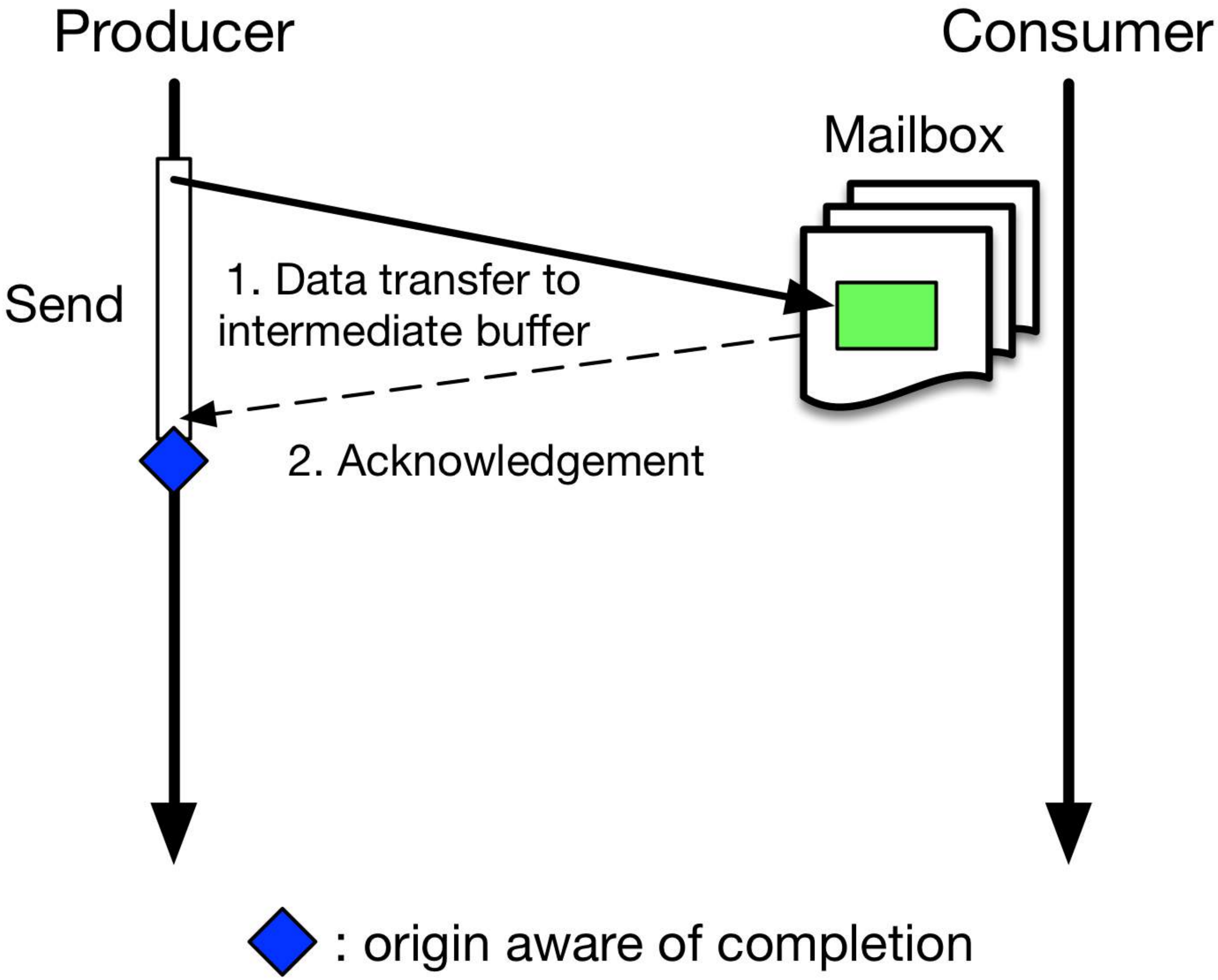
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, “High performance RDMA protocols in HPC.”, EuroMPI’06

MPI-1 Message Passing – Simple Eager



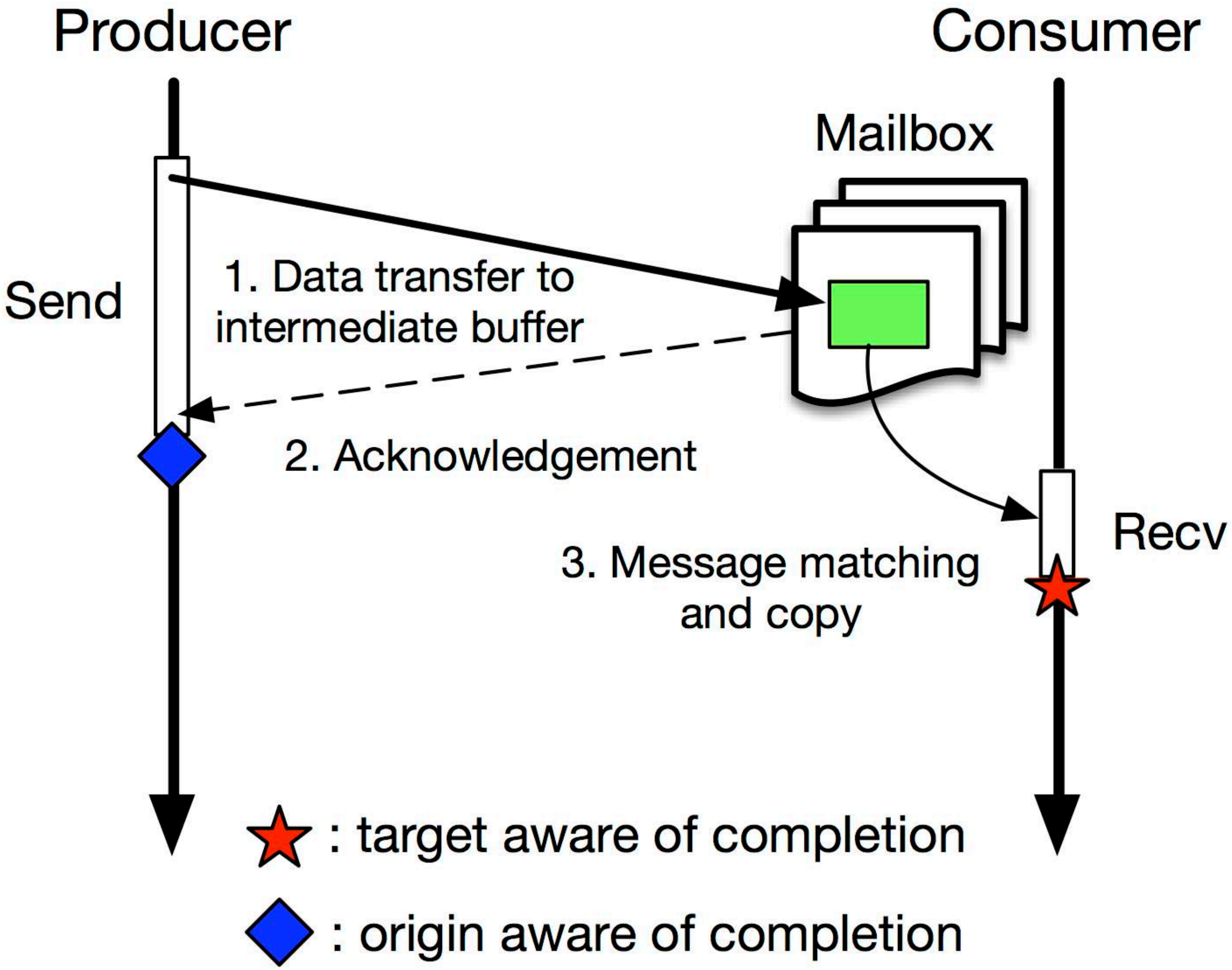
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High performance RDMA protocols in HPC.", EuroMPI'06

MPI-1 MESSAGE PASSING – SIMPLE EAGER



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

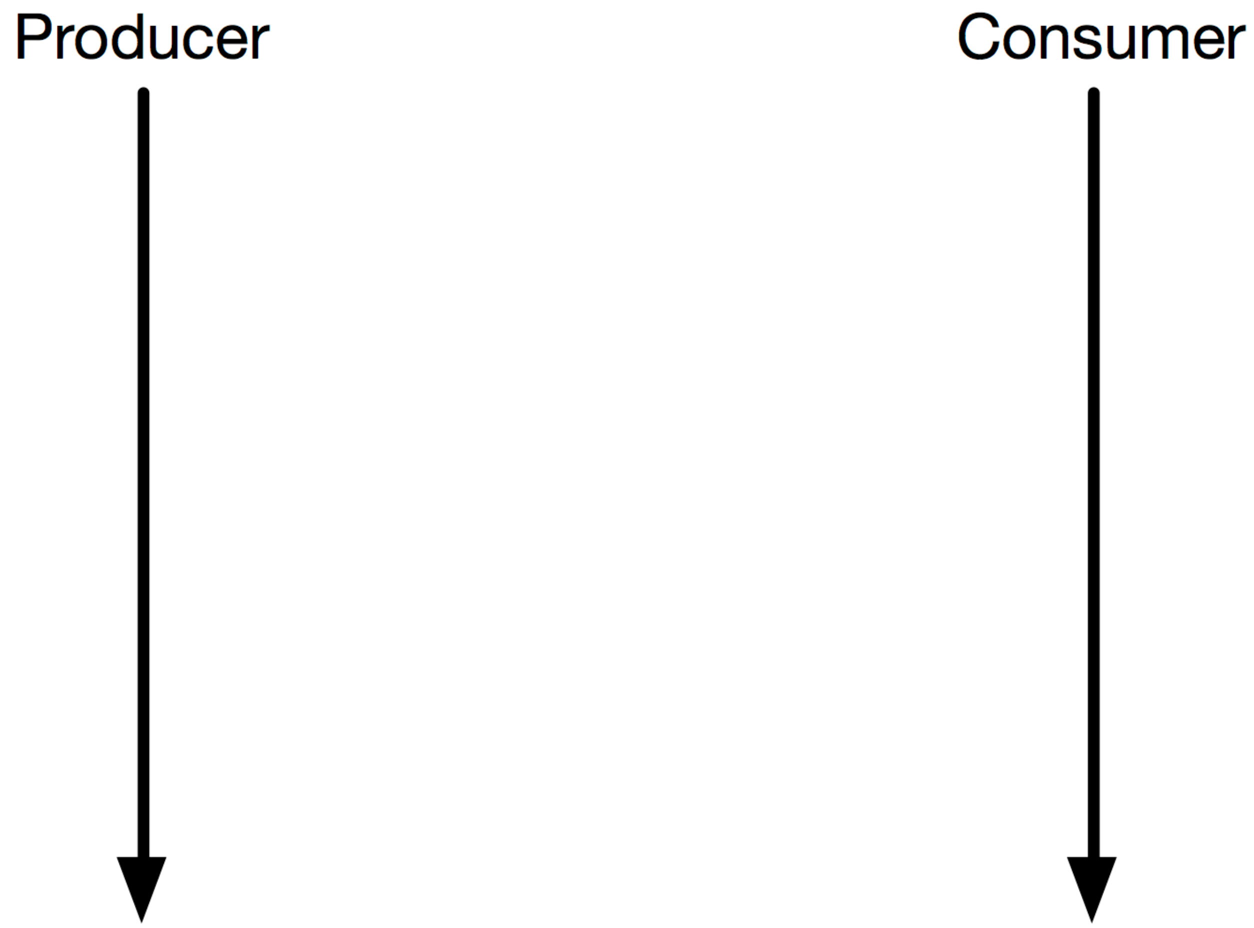
MPI-1 Message Passing – Simple Eager



Critical path: 1 latency + 1 copy

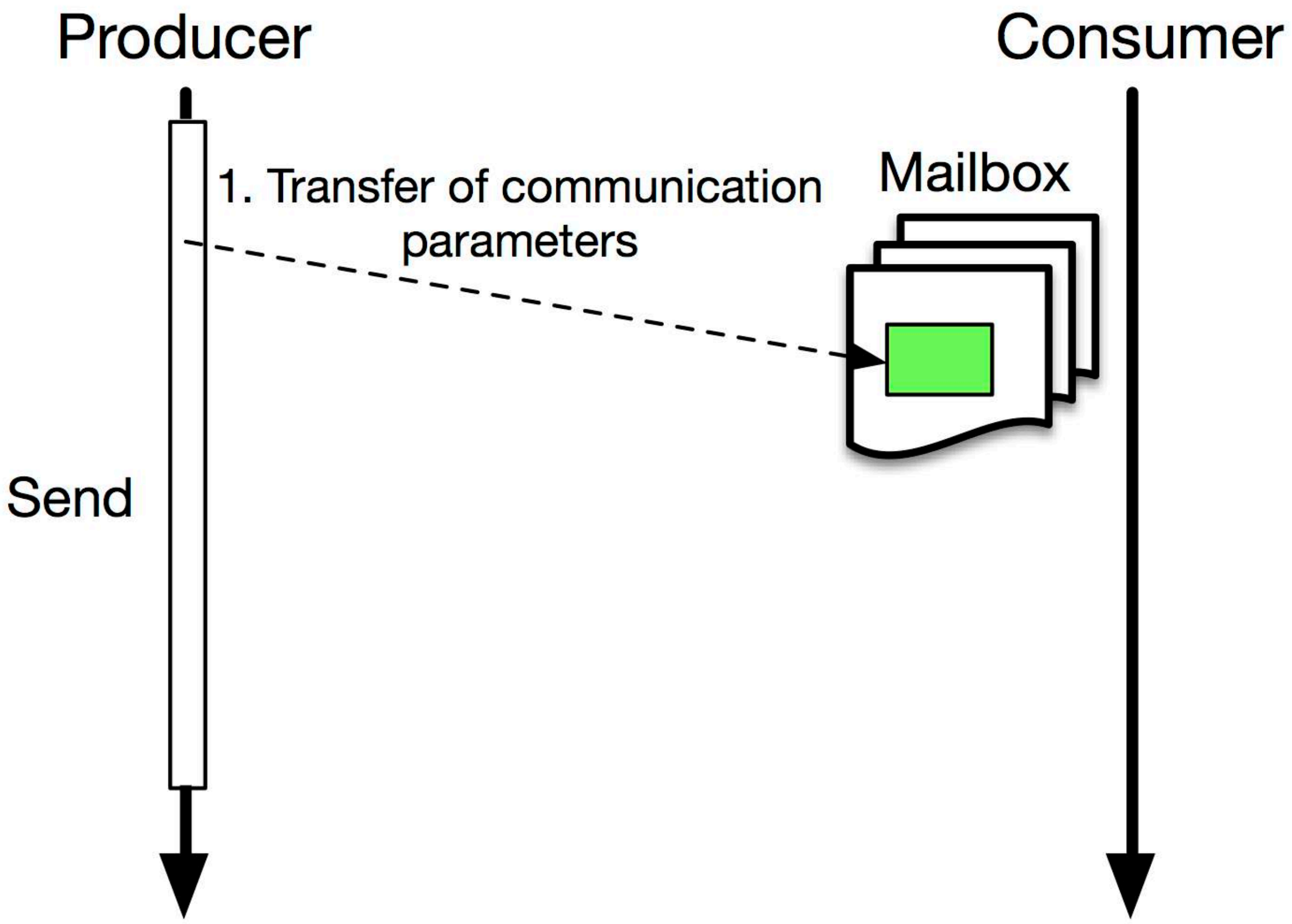
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, “High performance RDMA protocols in HPC.”, EuroMPI’06

MPI-1 Message Passing – Simple Rendezvous



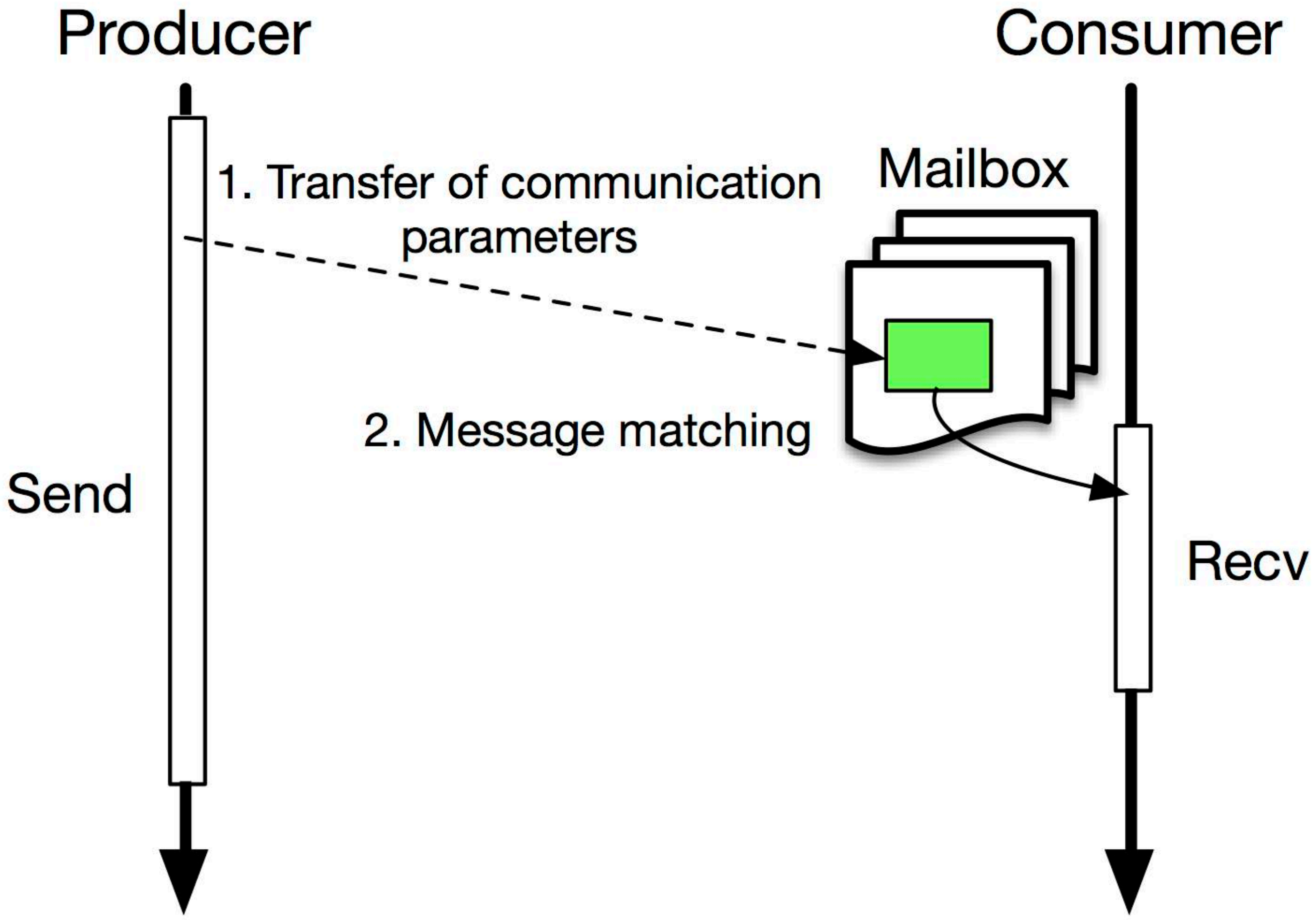
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, “High performance RDMA protocols in HPC.”, EuroMPI’06

MPI-1 MESSAGE PASSING – SIMPLE RENDEZVOUS



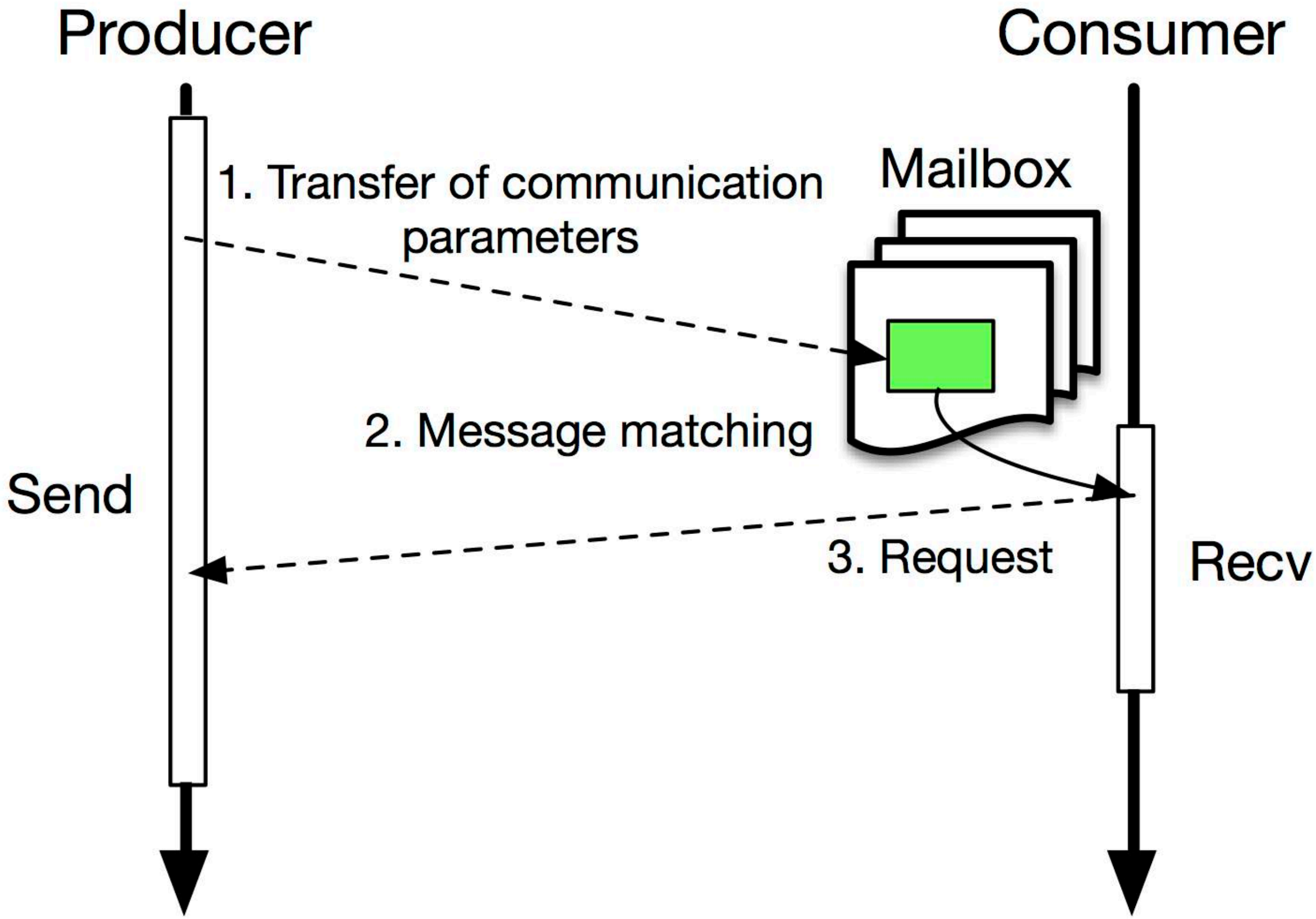
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High performance RDMA protocols in HPC.", EuroMPI'06

MPI-1 Message Passing – Simple Rendezvous



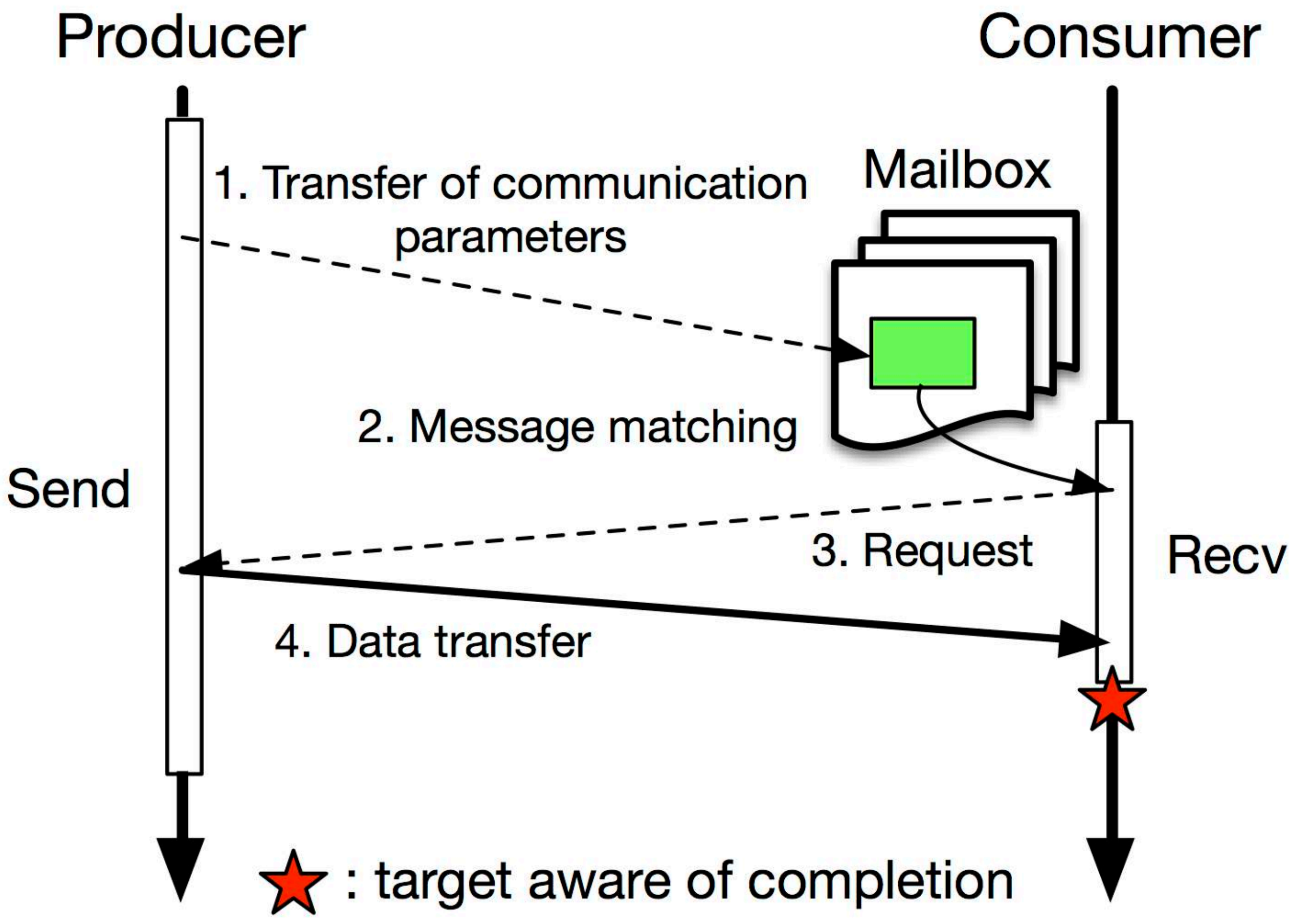
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High performance RDMA protocols in HPC.", EuroMPI'06

MPI-1 Message Passing – Simple Rendezvous



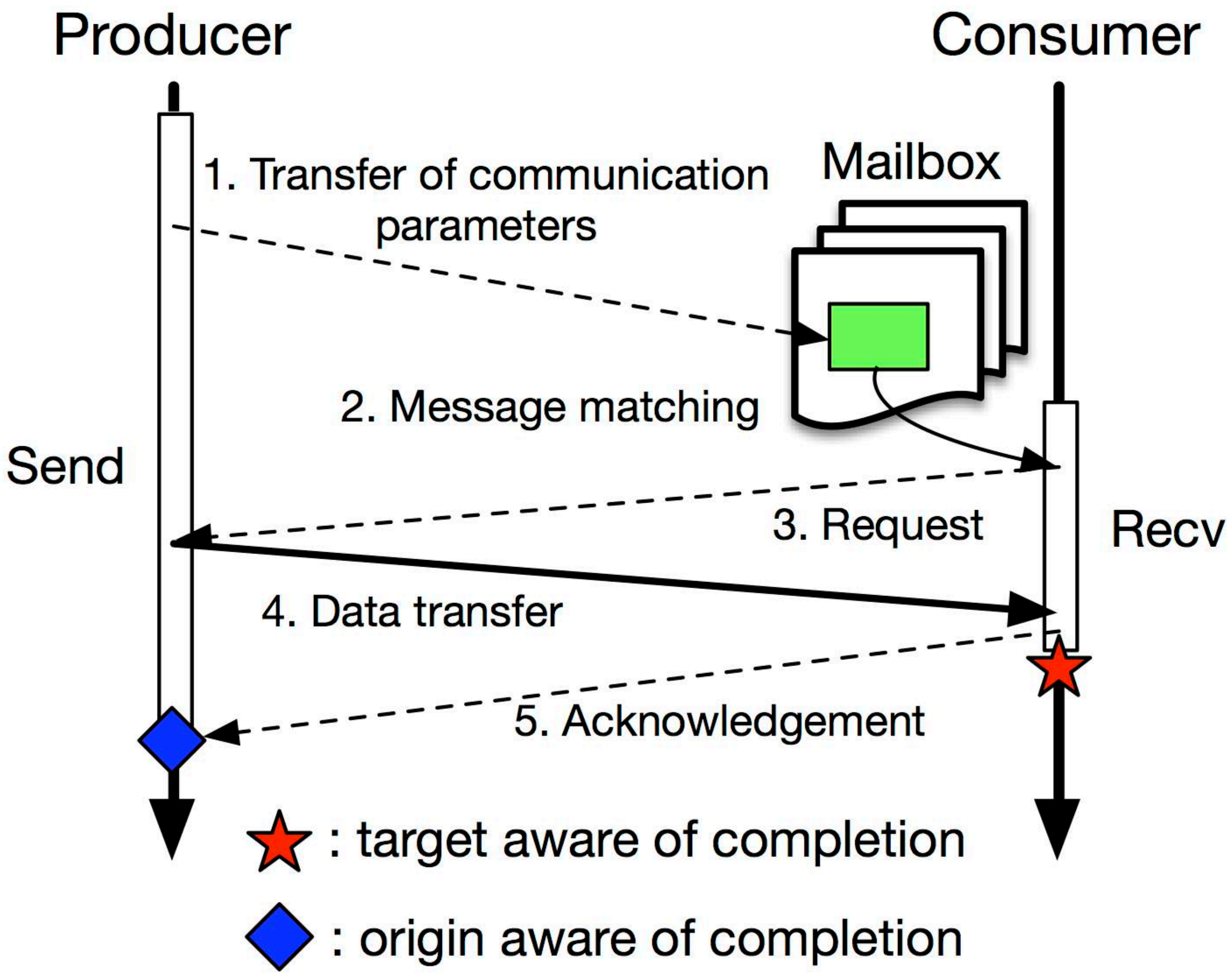
[1]: T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High performance RDMA protocols in HPC.", EuroMPI'06

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

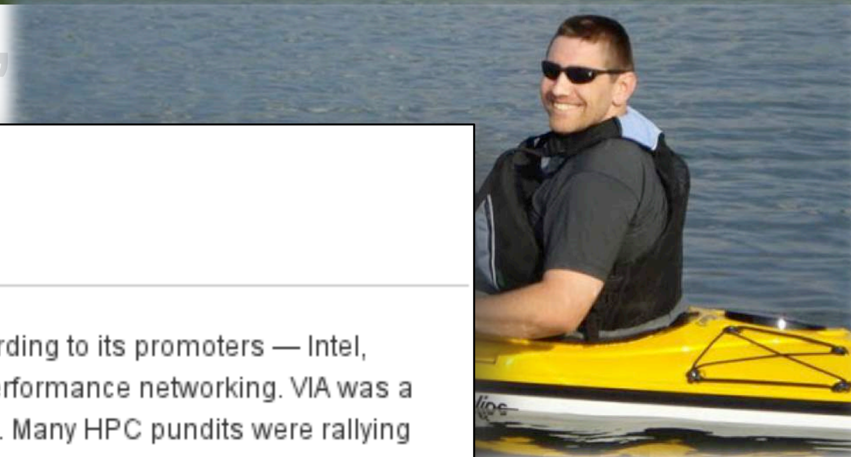
MPI-1 Message Passing – Simple Rendezvous



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'



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

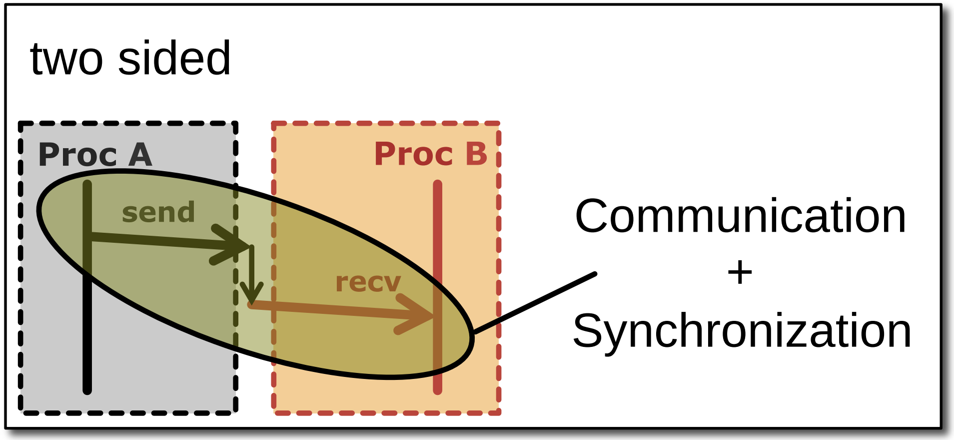
- **Why not use these RDMA features more directly?**
  - A global address space may simplify programming
  - … and accelerate communication
  - … and there could be a widely accepted standard

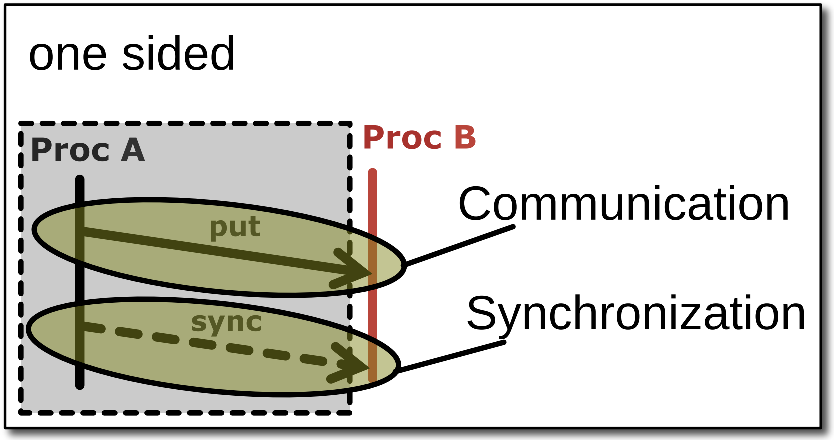
- **MPI-3 RMA ("MPI One Sided") was born**
  - Just one among many others (UPC, CAF, …)
  - Designed to react to hardware trends, learn from others
  - Direct (hardware-supported) remote access
  - New way of thinking for programmers

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

MPI-3 RMA SUMMARY

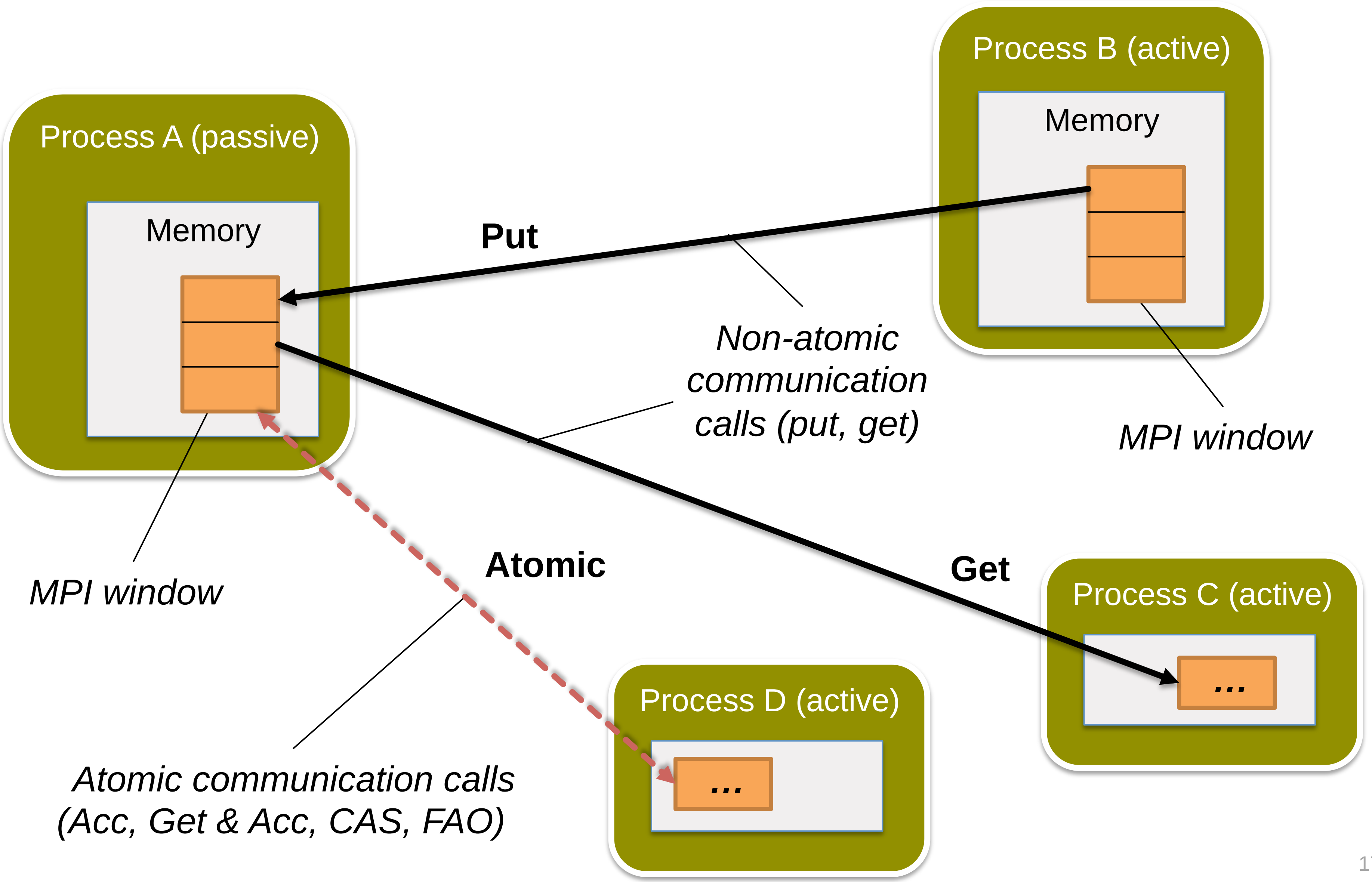
- **MPI-3 updates RMA ("MPI One Sided")**
  - Significant change from MPI-2
- **Communication is „one sided" (no involvement of destination)**
  - Utilize direct memory access
- **RMA decouples communication & synchronization**
  - Fundamentally different from message passing



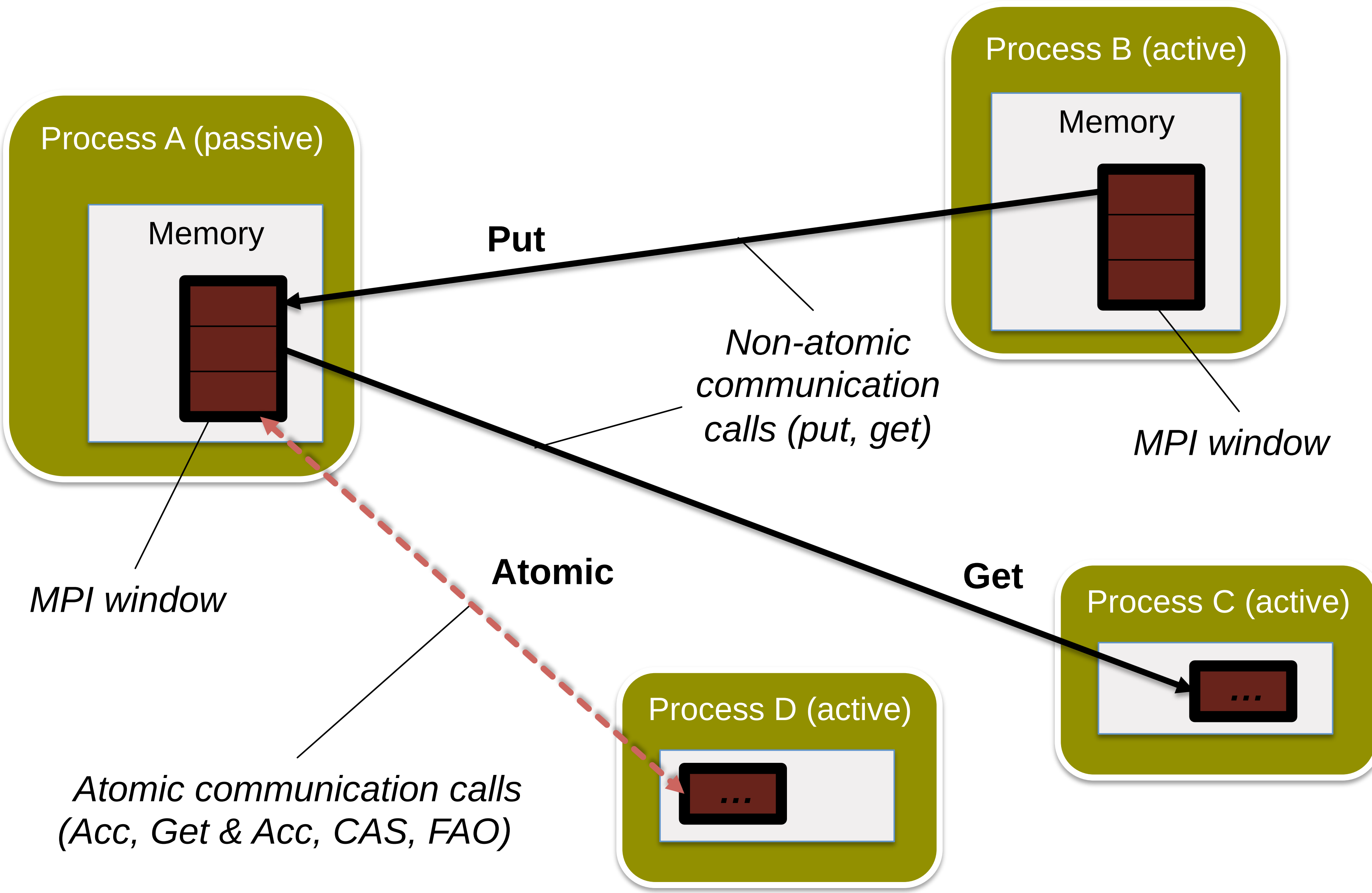


[1] http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

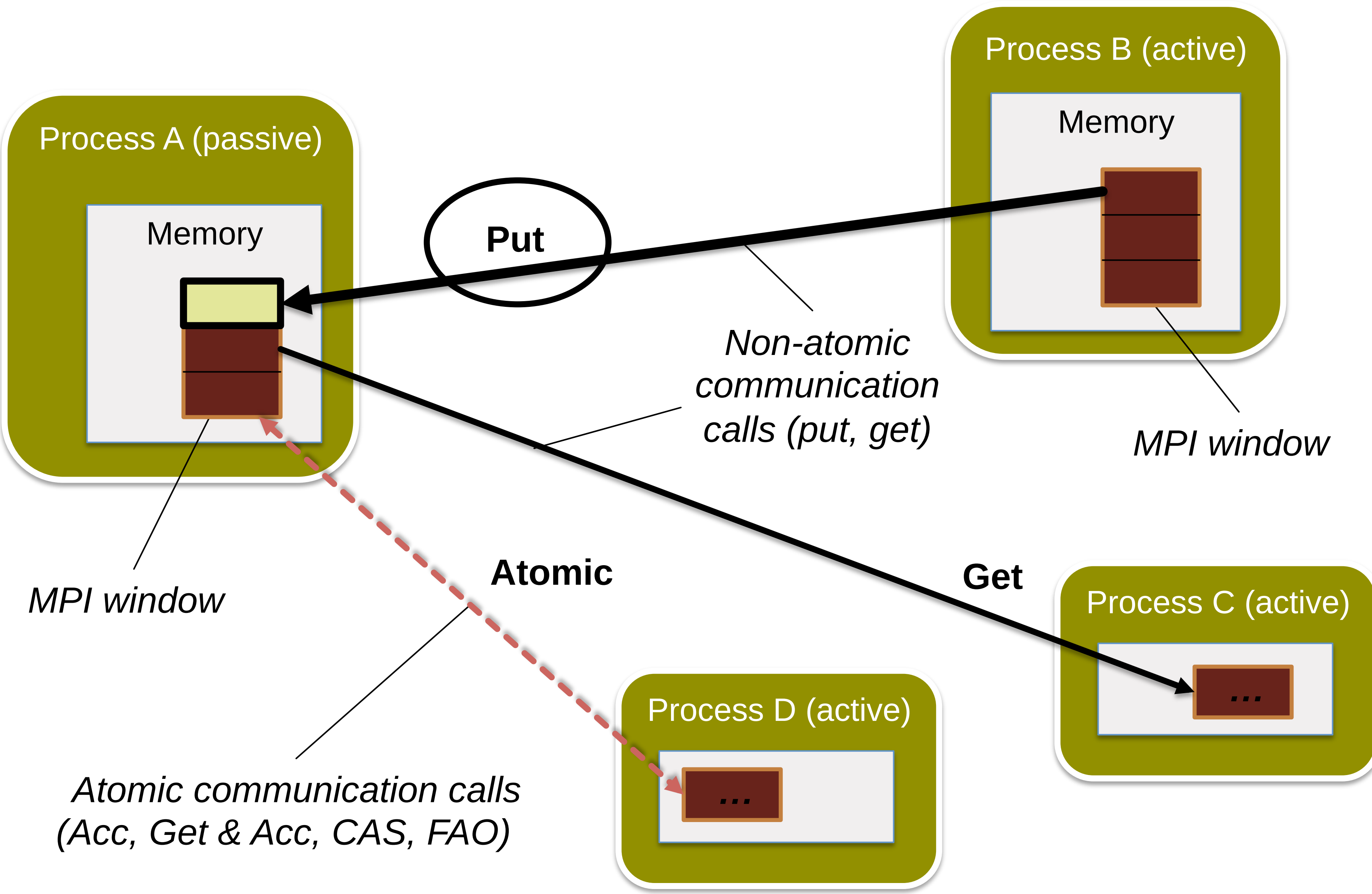
MPI-3 RMA Communication Overview



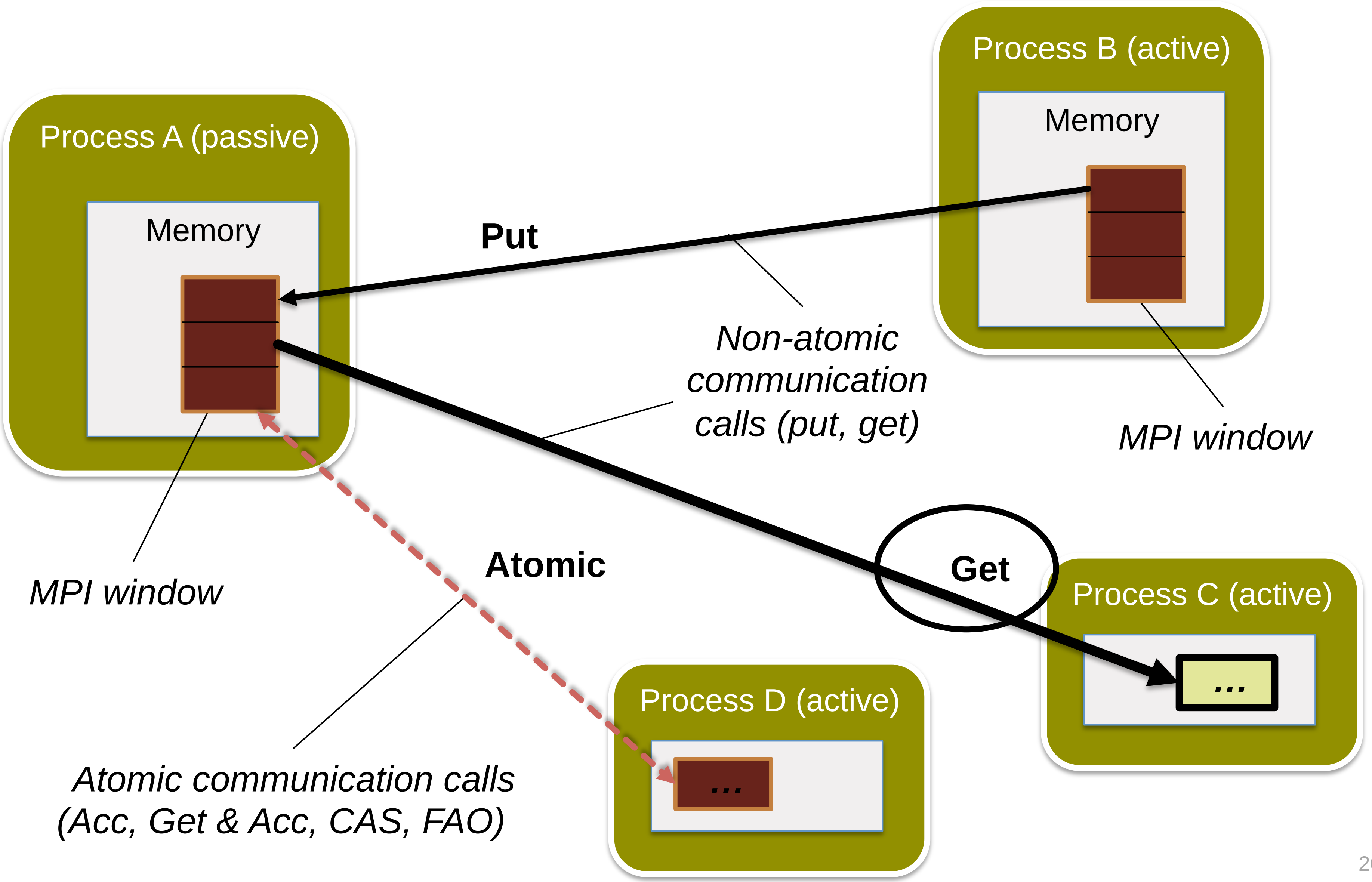
MPI-3 RMA Communication Overview



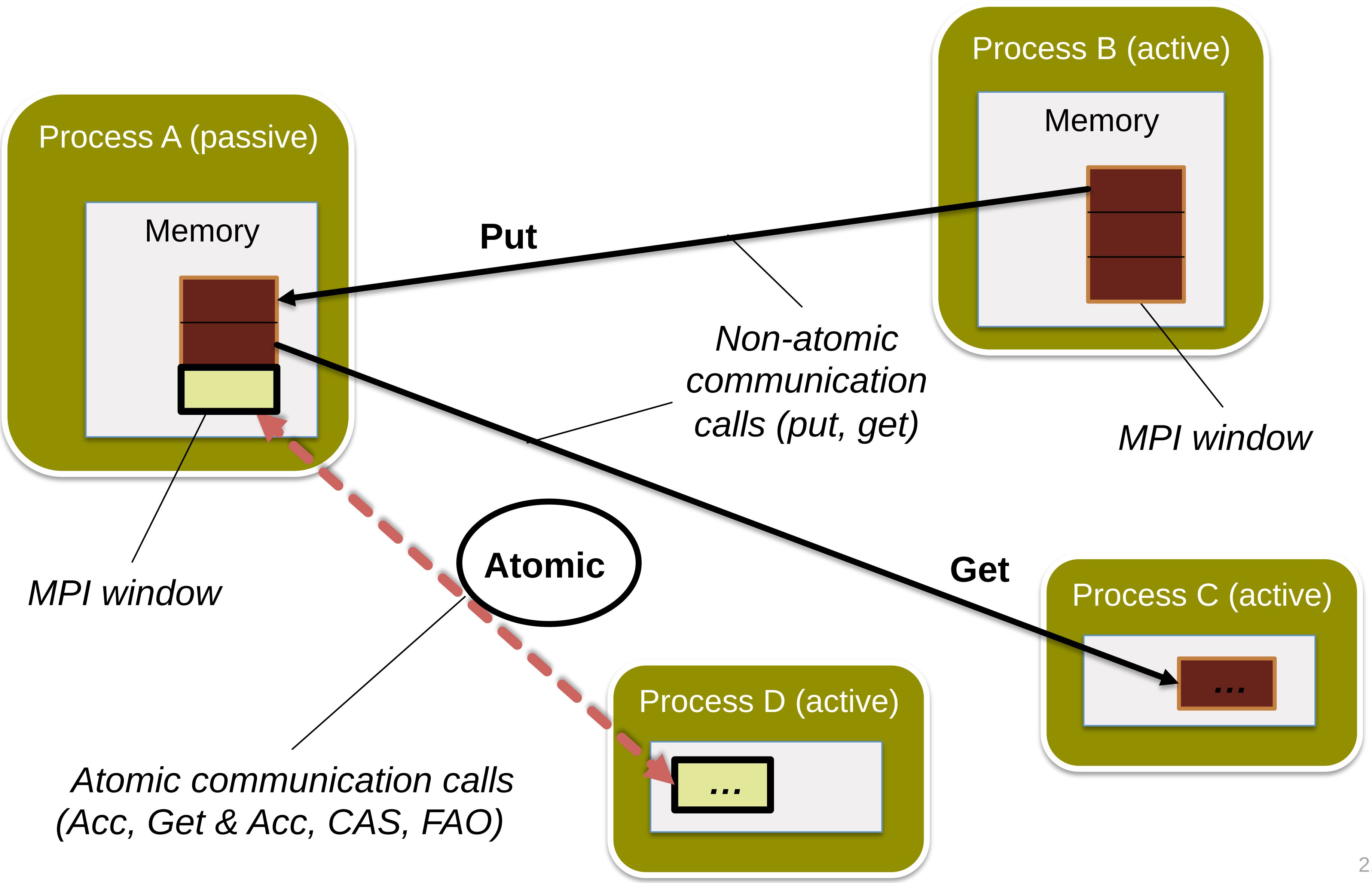
MPI-3 RMA COMMUNICATION OVERVIEW



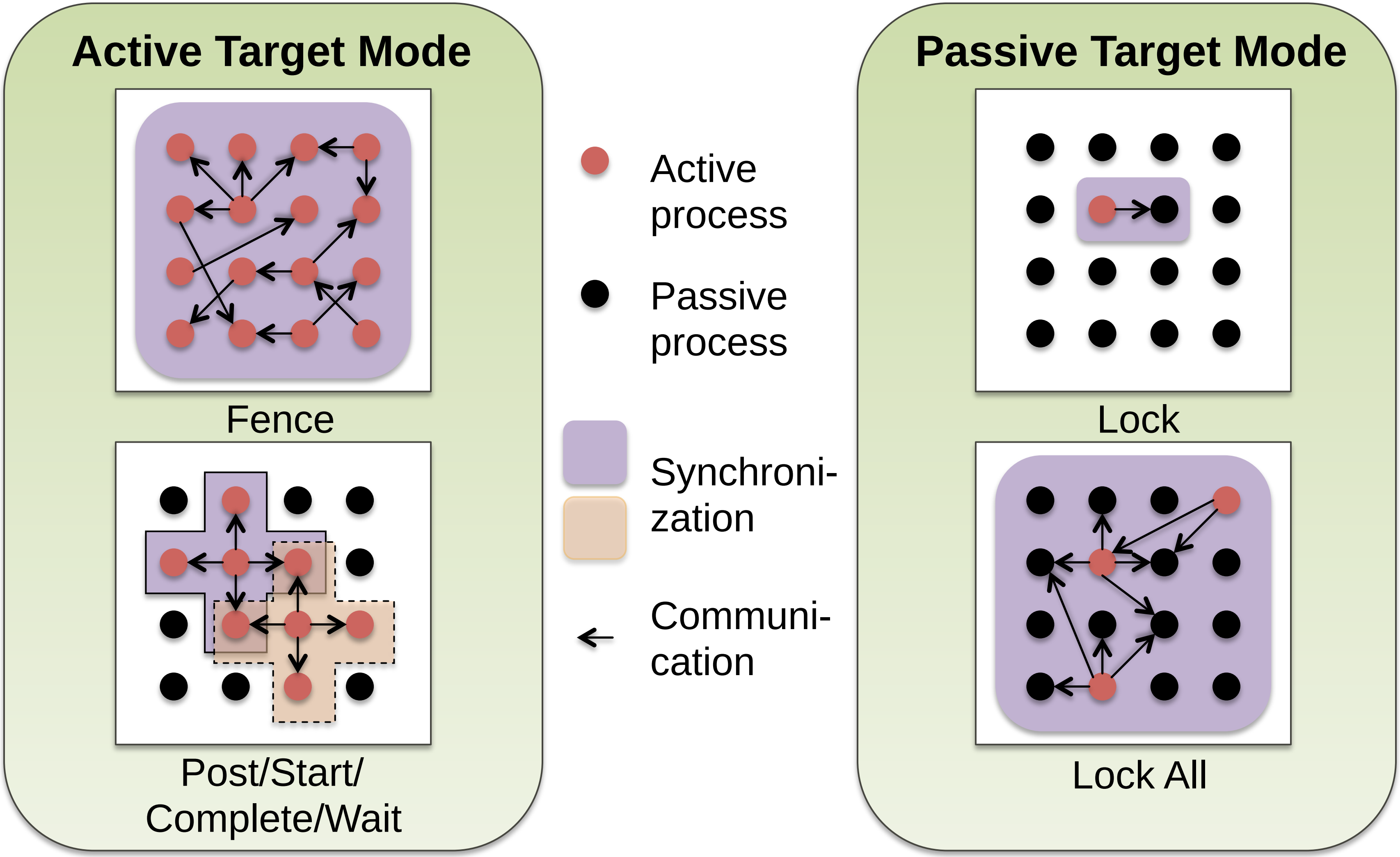
MPI-3 RMA Communication Overview



MPI-3 RMA Communication Overview

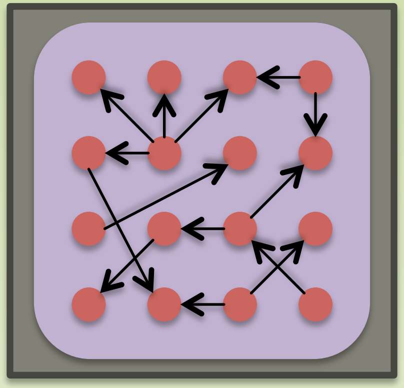


MPI-3 RMA Synchronization Overview

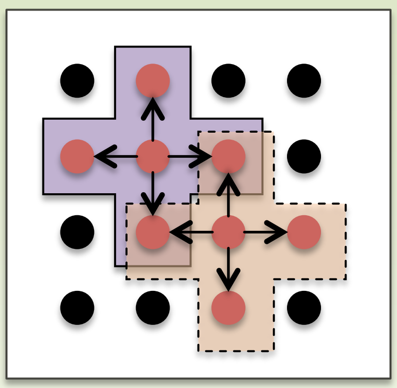


MPI-3 RMA SYNCHRONIZATION OVERVIEW

Active Target Mode



Fence



Post/Start/
Complete/Wait

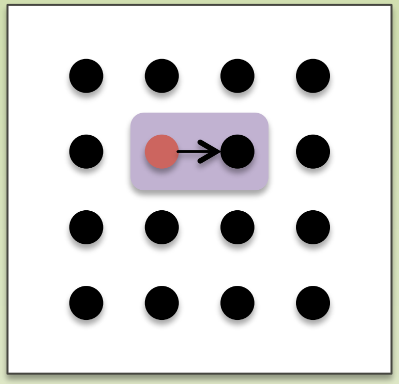
Active process

Passive process

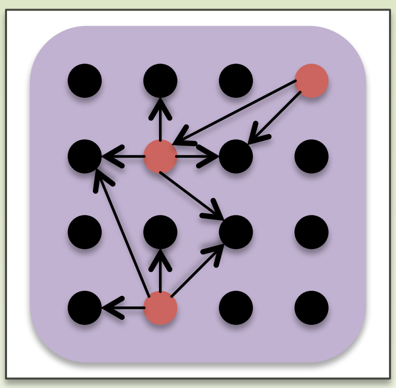
Synchronization

Communication

Passive Target Mode

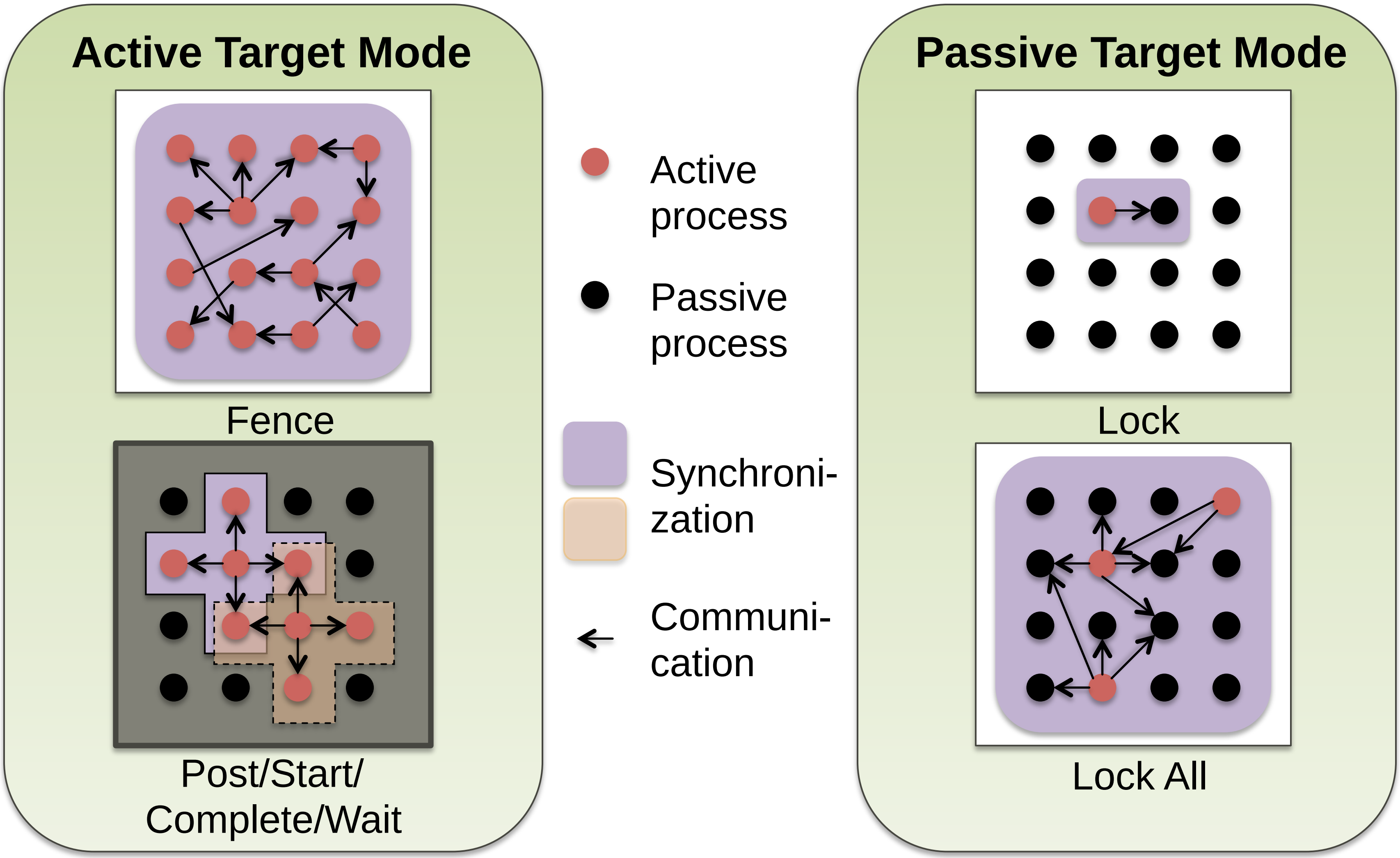


Lock

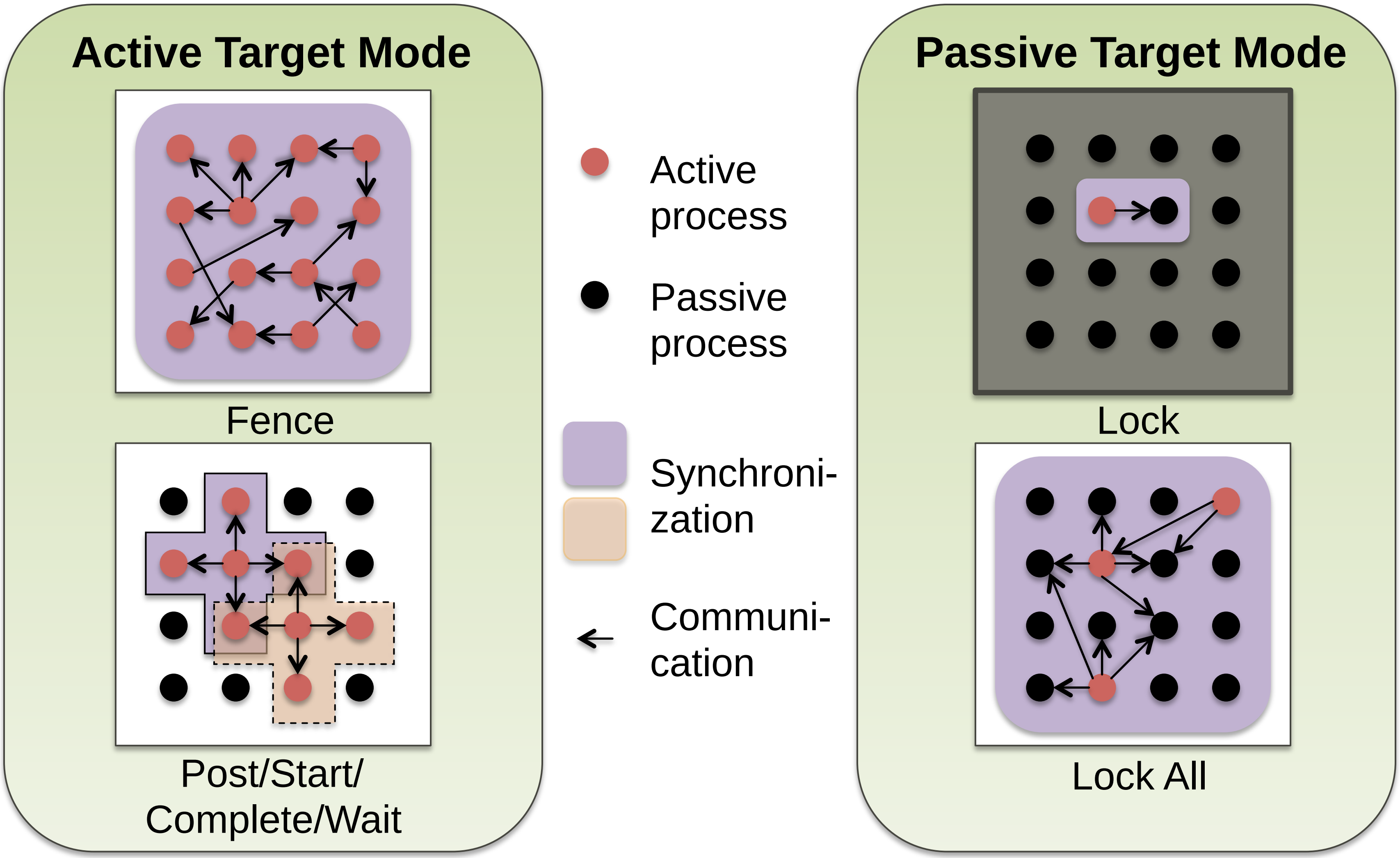


Lock All

MPI-3 RMA Synchronization Overview

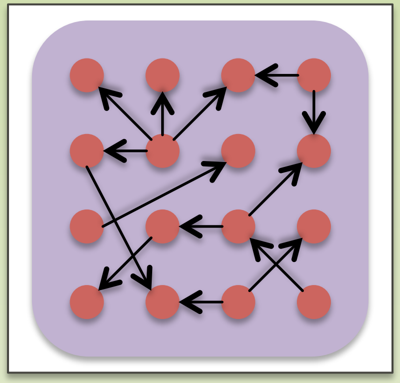


MPI-3 RMA Synchronization Overview

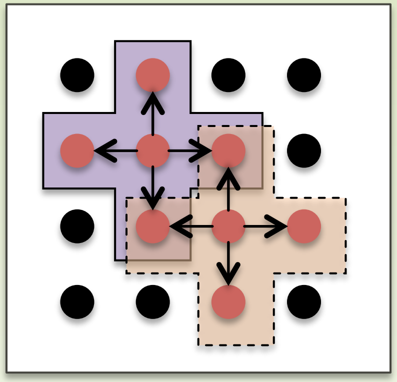


MPI-3 RMA Synchronization Overview

Active Target Mode



Fence



Post/Start/
Complete/Wait

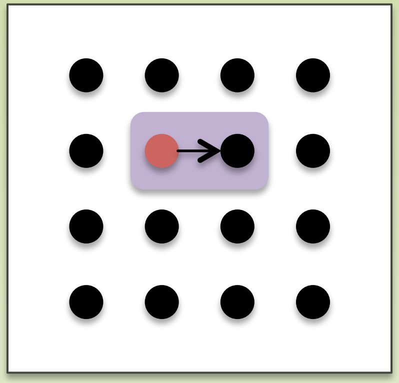
Active process

Passive process

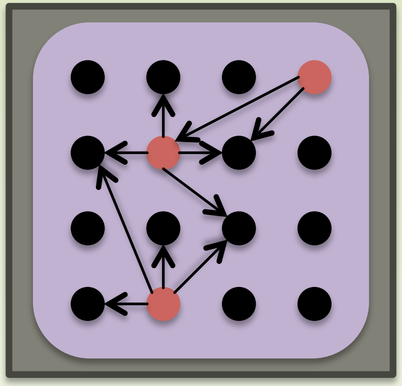
Synchroni-zation

Communi-cation

Passive Target Mode



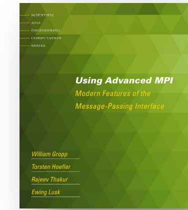
Lock



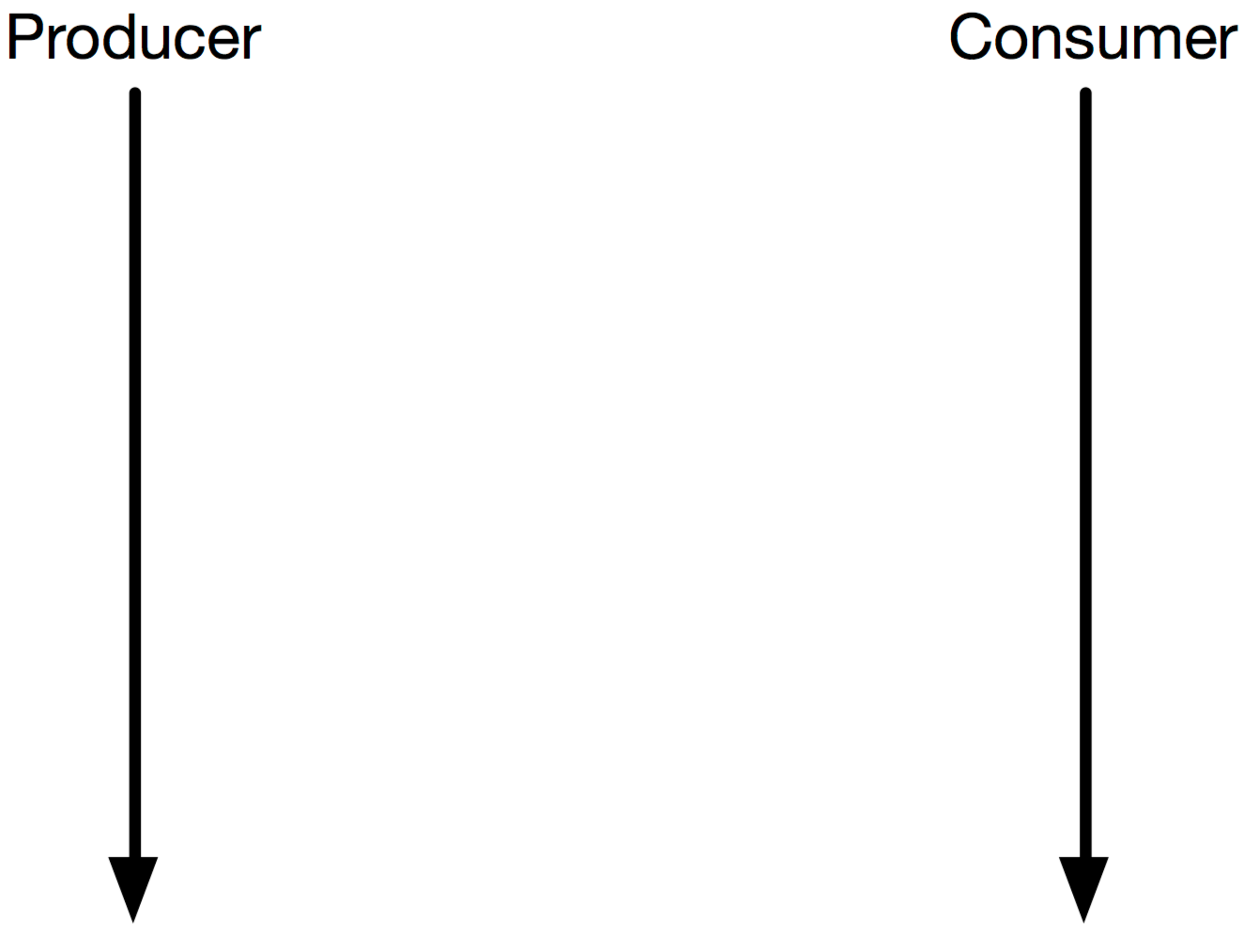
Lock All

IN CASE YOU WANT TO LEARN MORE

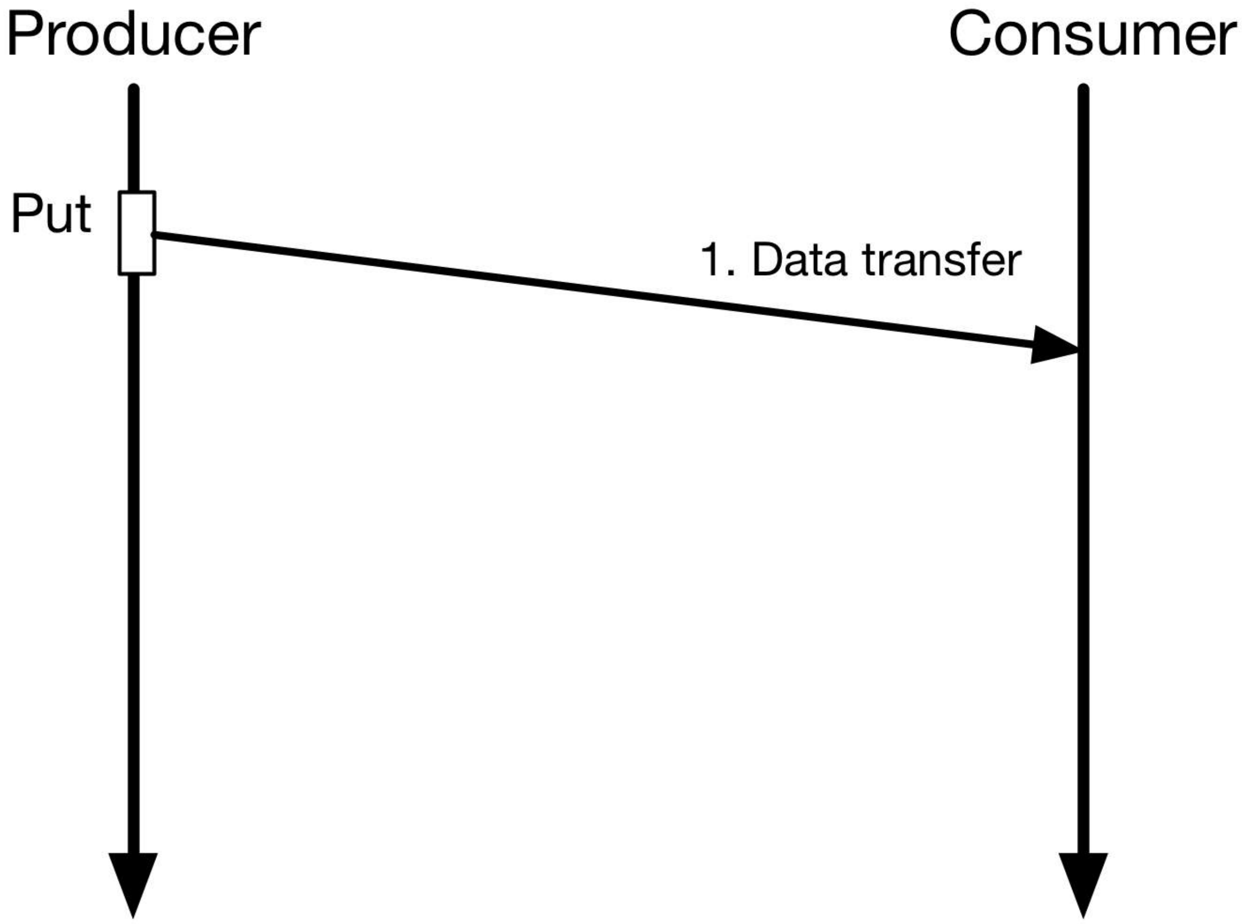
How to implement producer/consumer in passive mode?

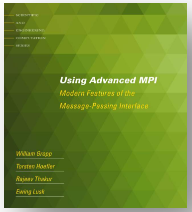


One Sided – Put + Synchronization

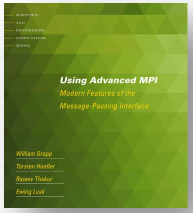


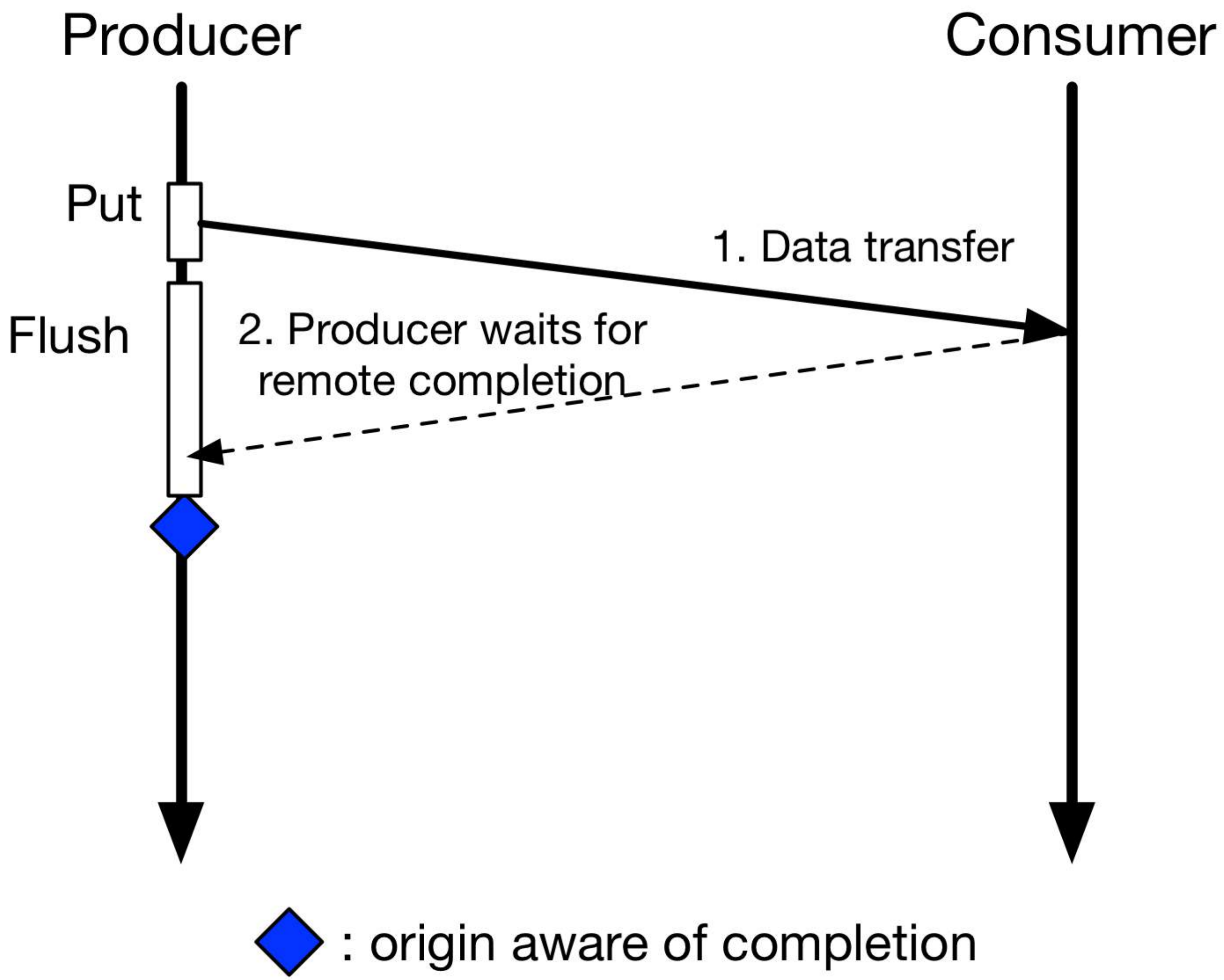
ONE SIDED – PUT + SYNCHRONIZATION

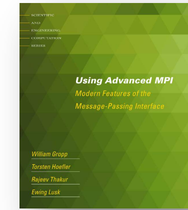




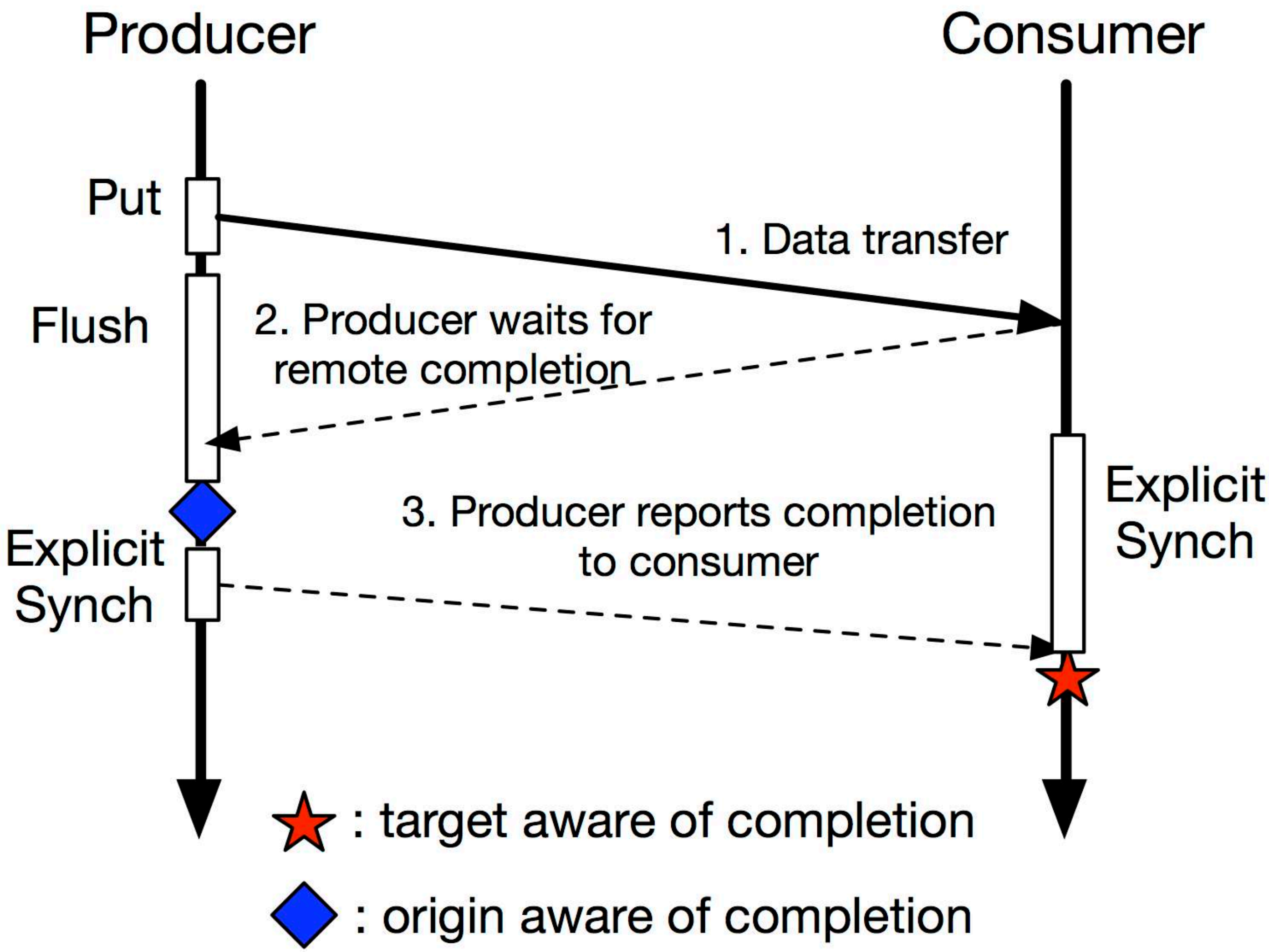
One Sided – Put + Synchronization





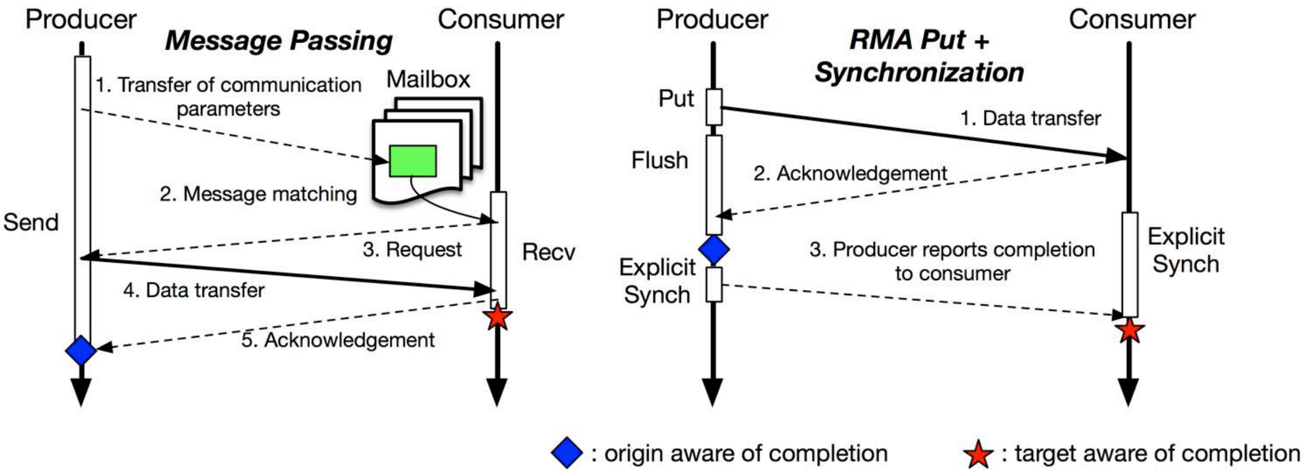


One Sided – Put + Synchronization



Critical path: 3 latencies

COMPARING APPROACHES

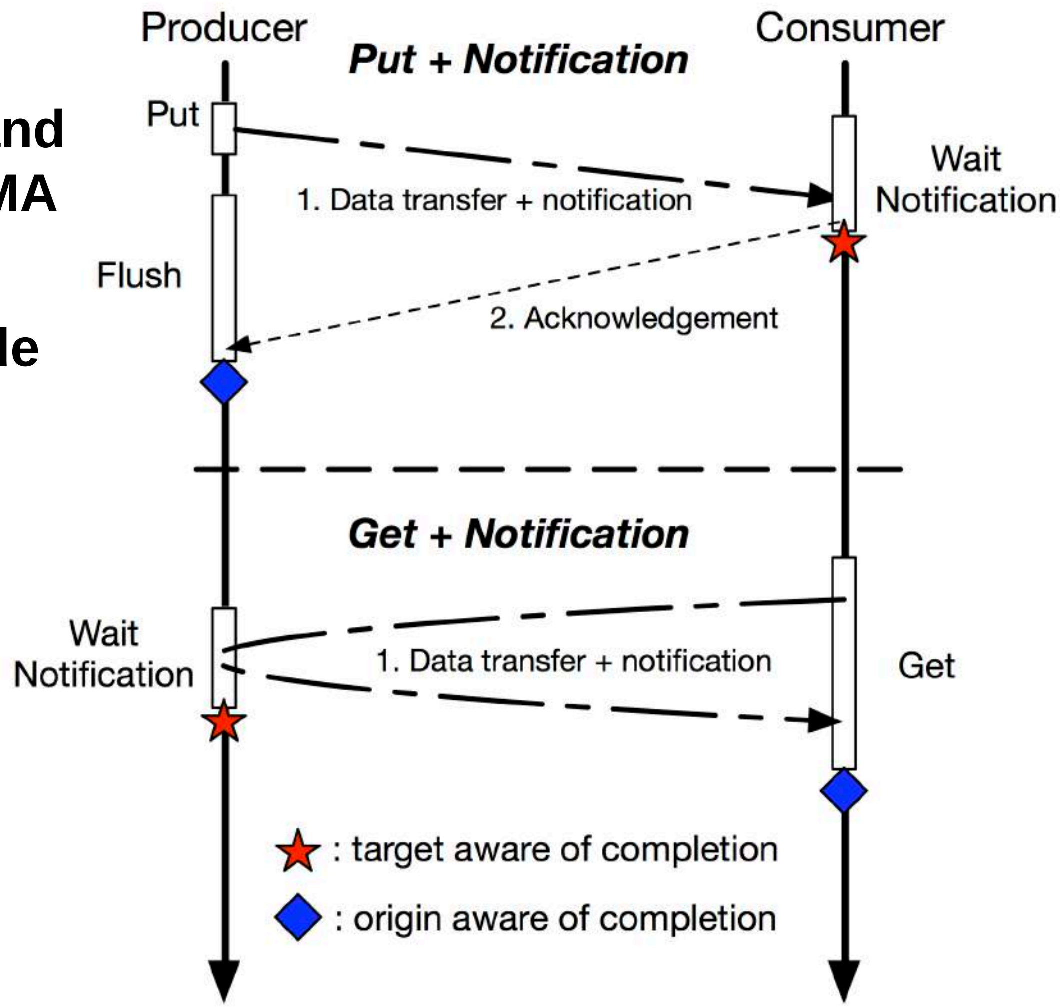


Message Passing
1 latency + copy / 3 latencies

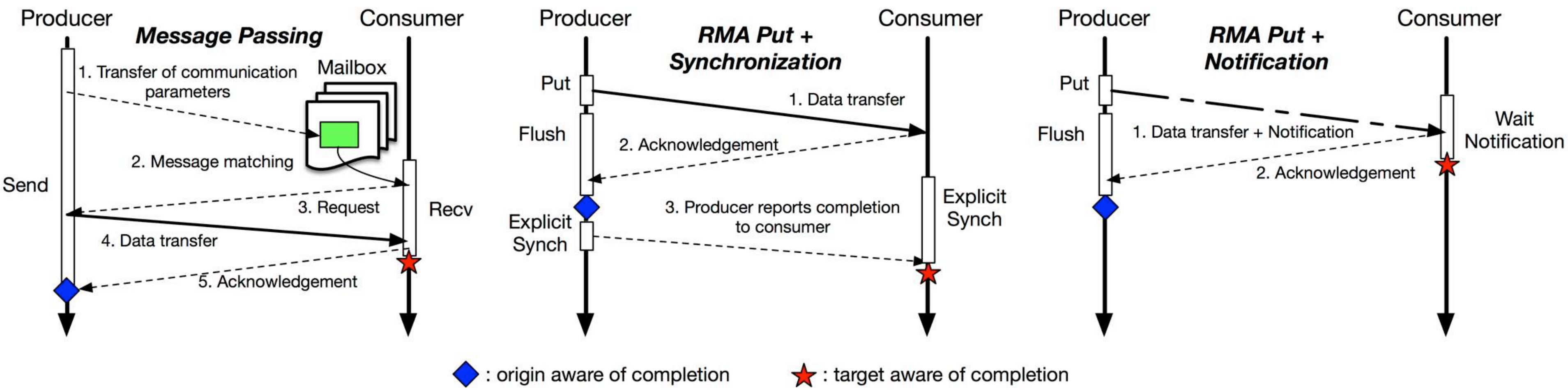
One Sided
3 latencies

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

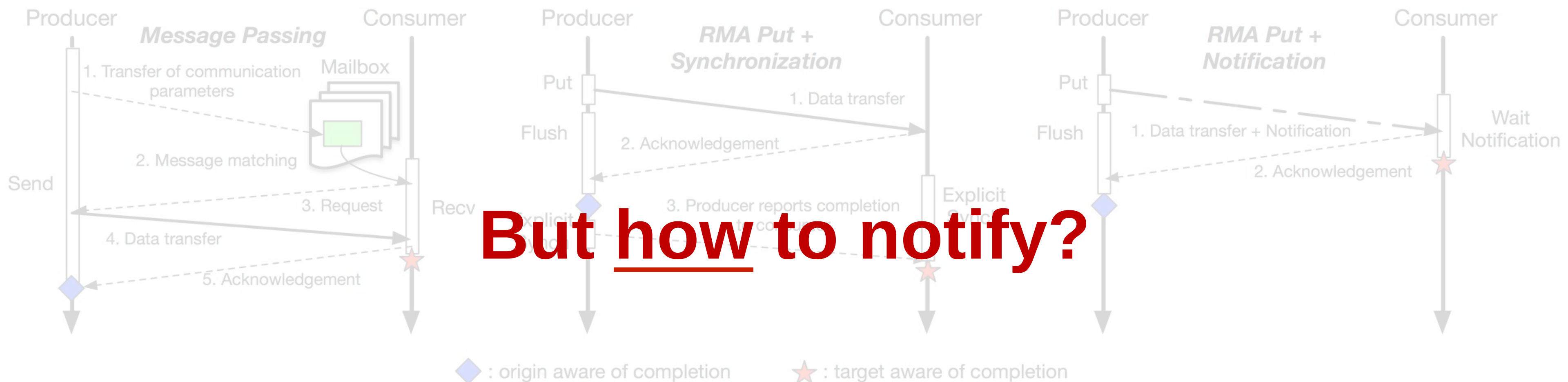


Message Passing
1 latency + copy / 3 latencies

One Sided
3 latencies

Notified Access
1 latency

COMPARING APPROACHES



But how to notify?

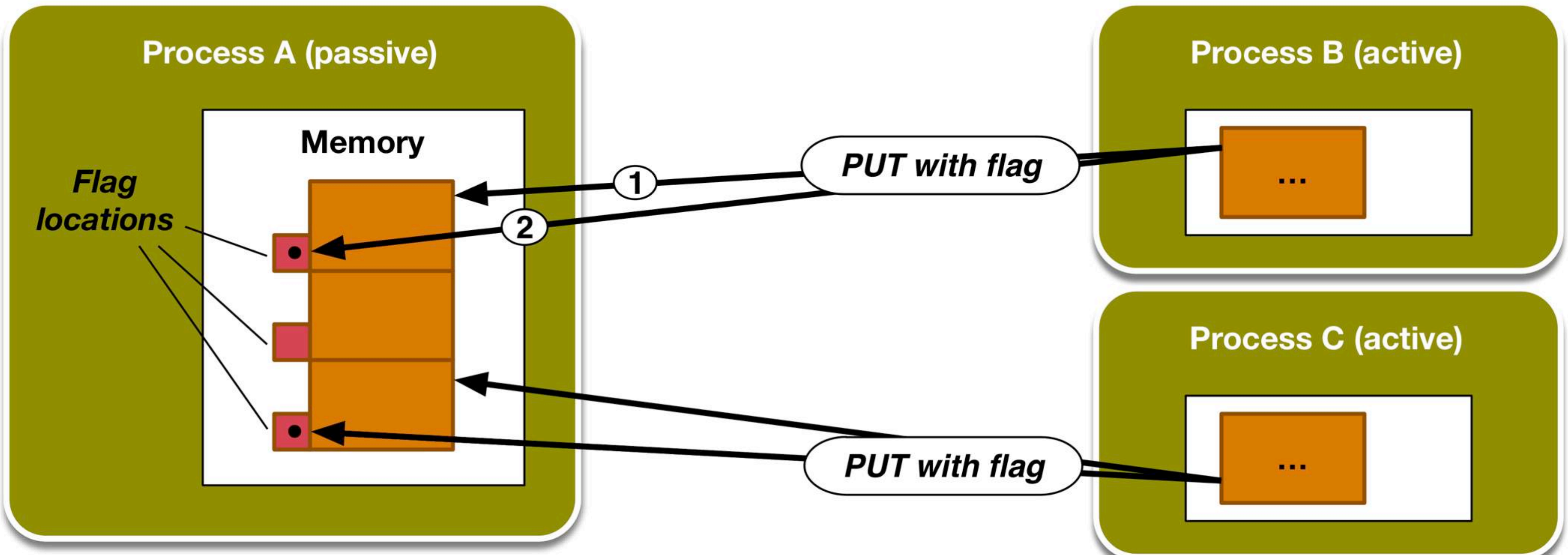
Message Passing
1 latency + copy / 3 latencies

One Sided
3 latencies

Notified Access
1 latency

Previous work: Overwriting Interface

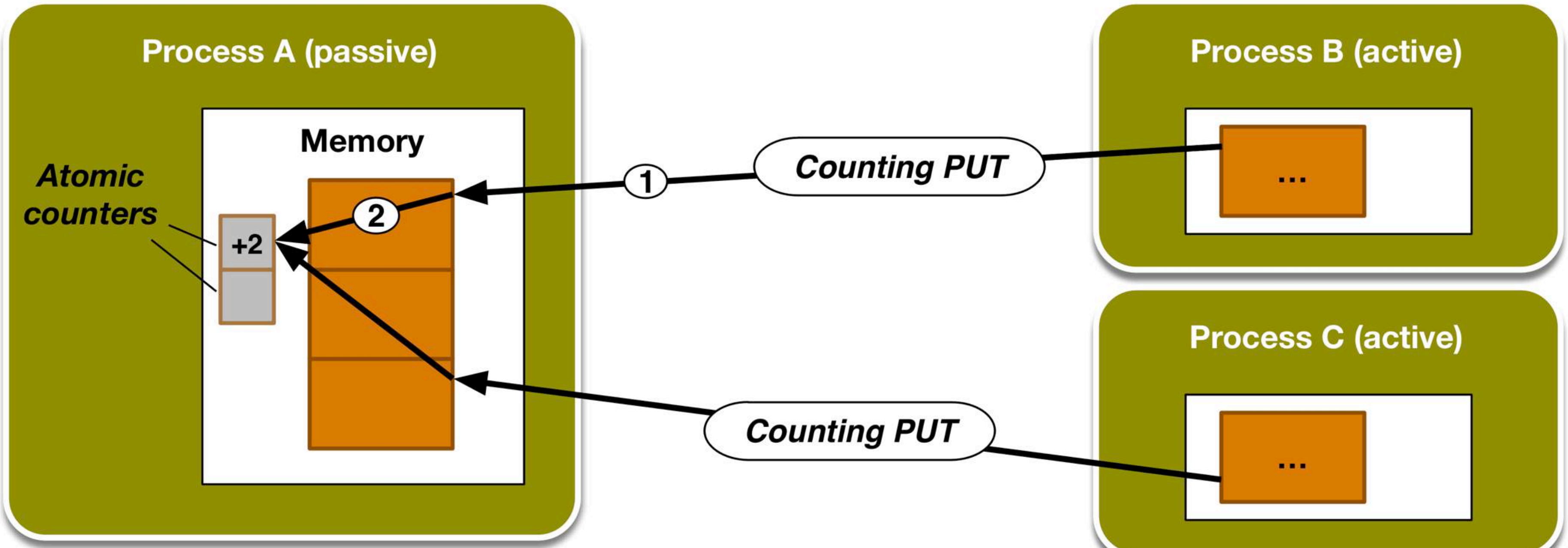
- **Flags (polling at the remote side)**
  - Used in *GASPI, DMAPP, NEON*



- **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

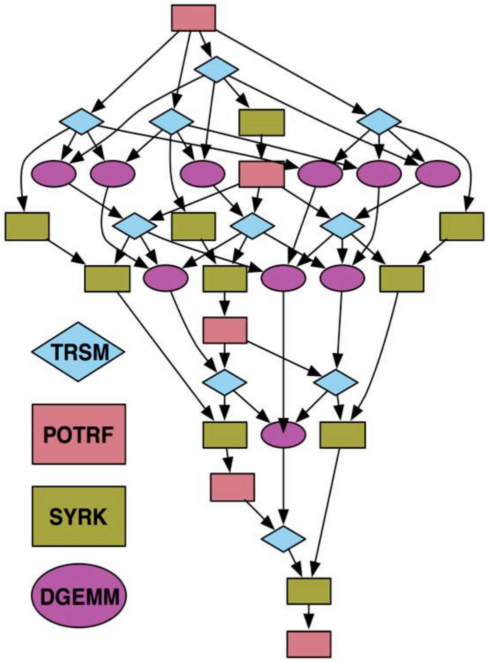
- **Atomic counters (accumulate notifications → scalable)**
  - Used in *Split-C, LAPI, SHMEM - Counting Puts, …*



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

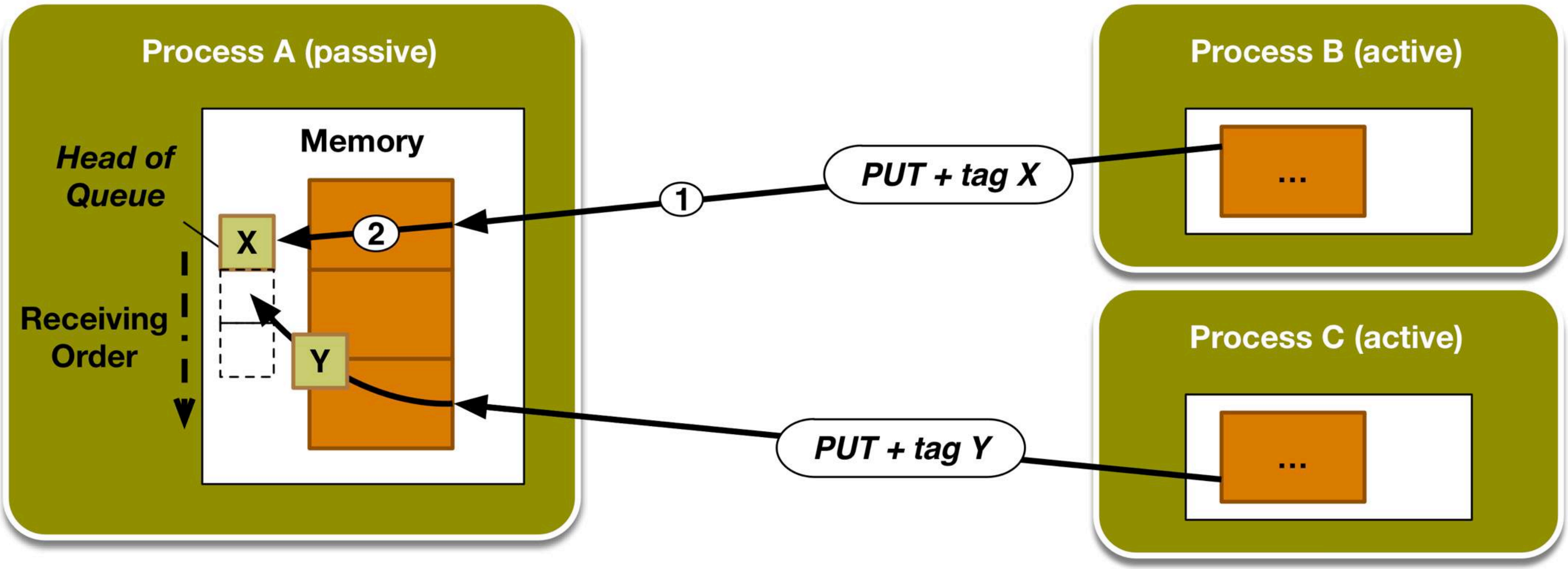
What is a Good Notification Interface?

- **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        (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);
```

NOTIFIED ACCESS – AN MPI INTERFACE

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

Example Communication Primitives

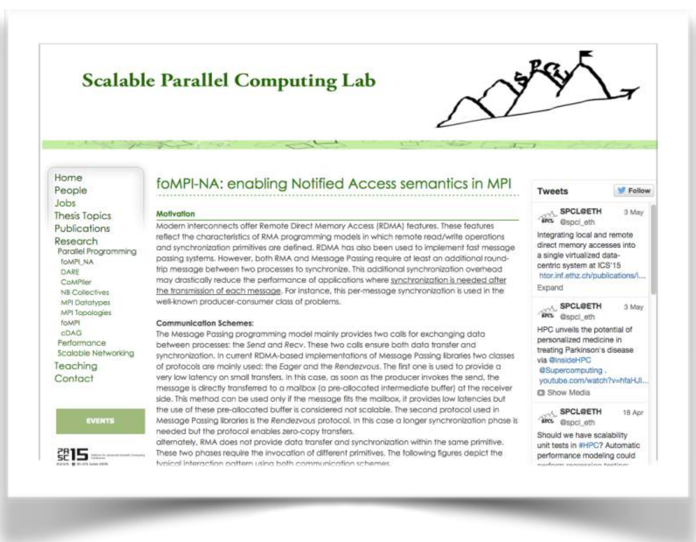
```
int MPI_Put_notify(void *origin_addr, int origin_count, MPI_Datatype origin_type, int target_rank,
                   MPI_Aint target_disp, int target_count, MPI_Datatype target_type, MPI_Win win,
                   int tag);
int MPI_Get_notify(void *origin_addr, int origin_count, MPI_Datatype origin_type,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

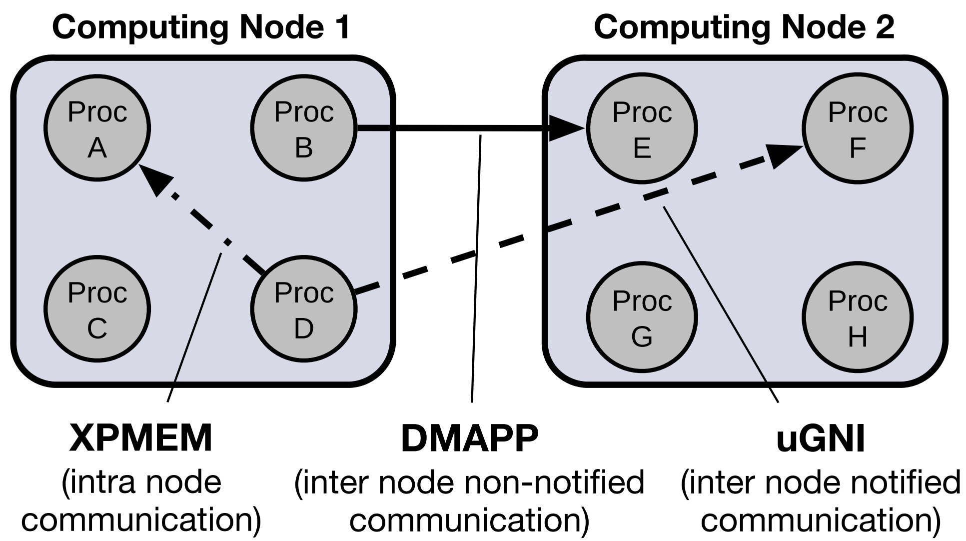
Example Synchronization Primitives

```
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);
```

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

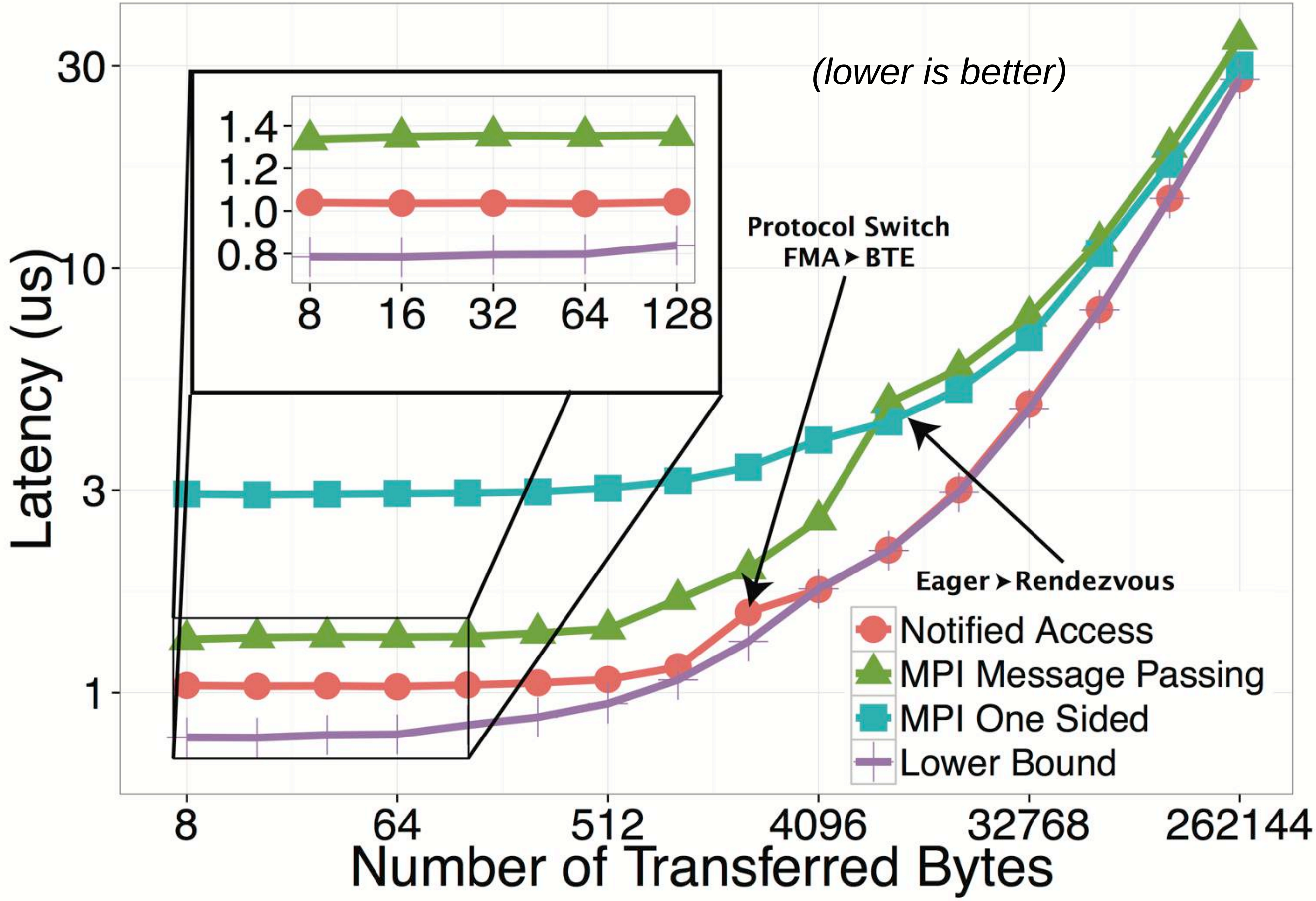
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

- **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

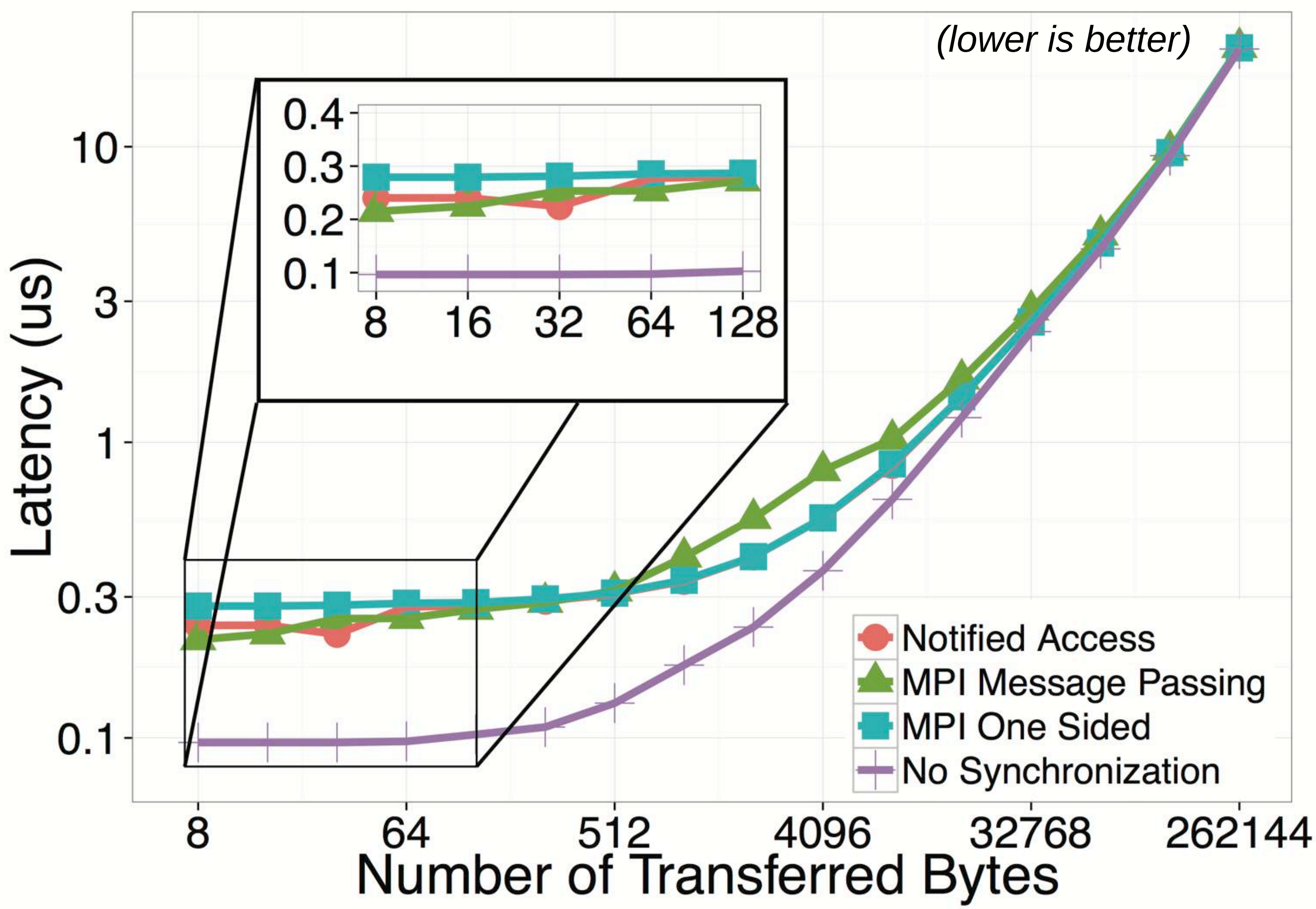
Ping Pong Performance (Inter-Node)

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



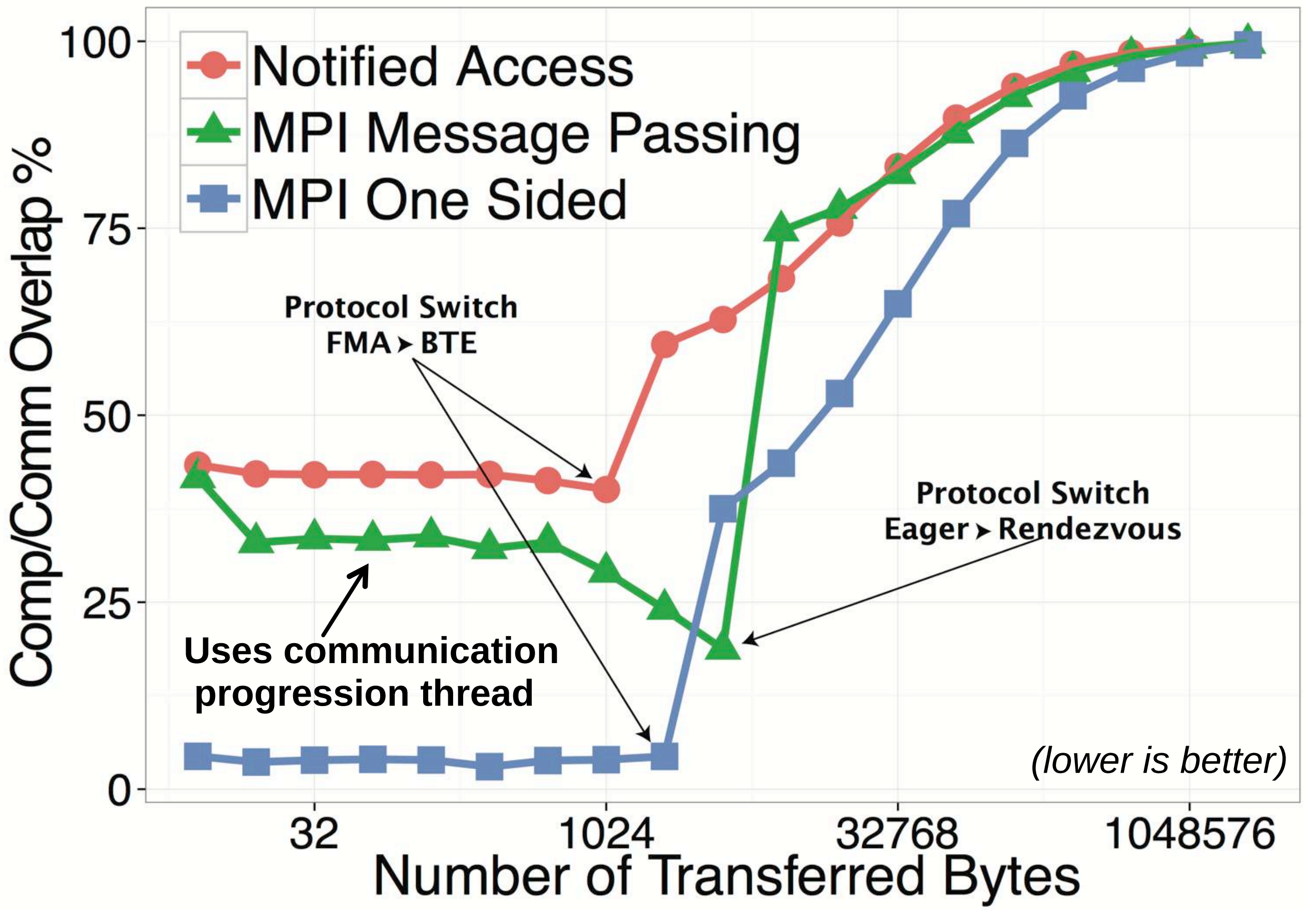
Ping Pong Performance (Intra-Node)

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



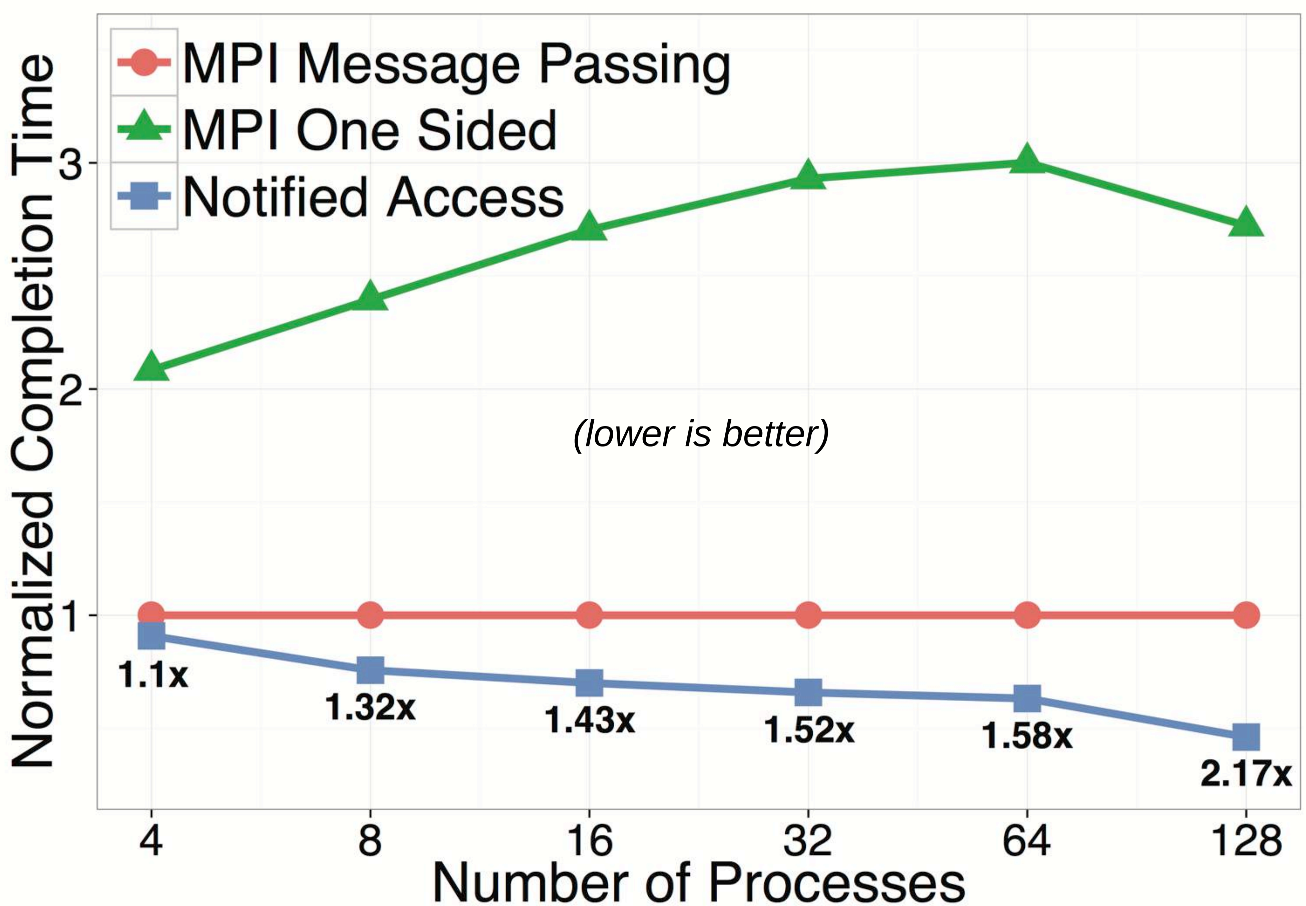
COMPUTATION/COMMUNICATION OVERLAP

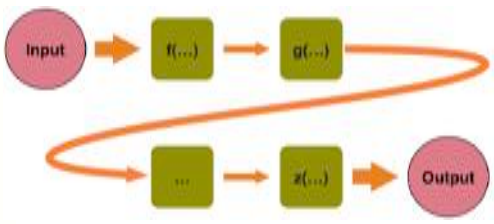
- **1000 repetitions, each timed separately, RDTSC timer**
- **95% confidence interval always within 1% of median**

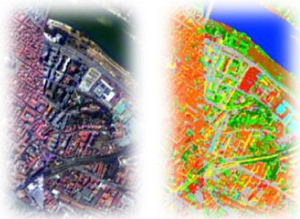


PIPELINE – ONE-TO-ONE SYNCHRONIZATION

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



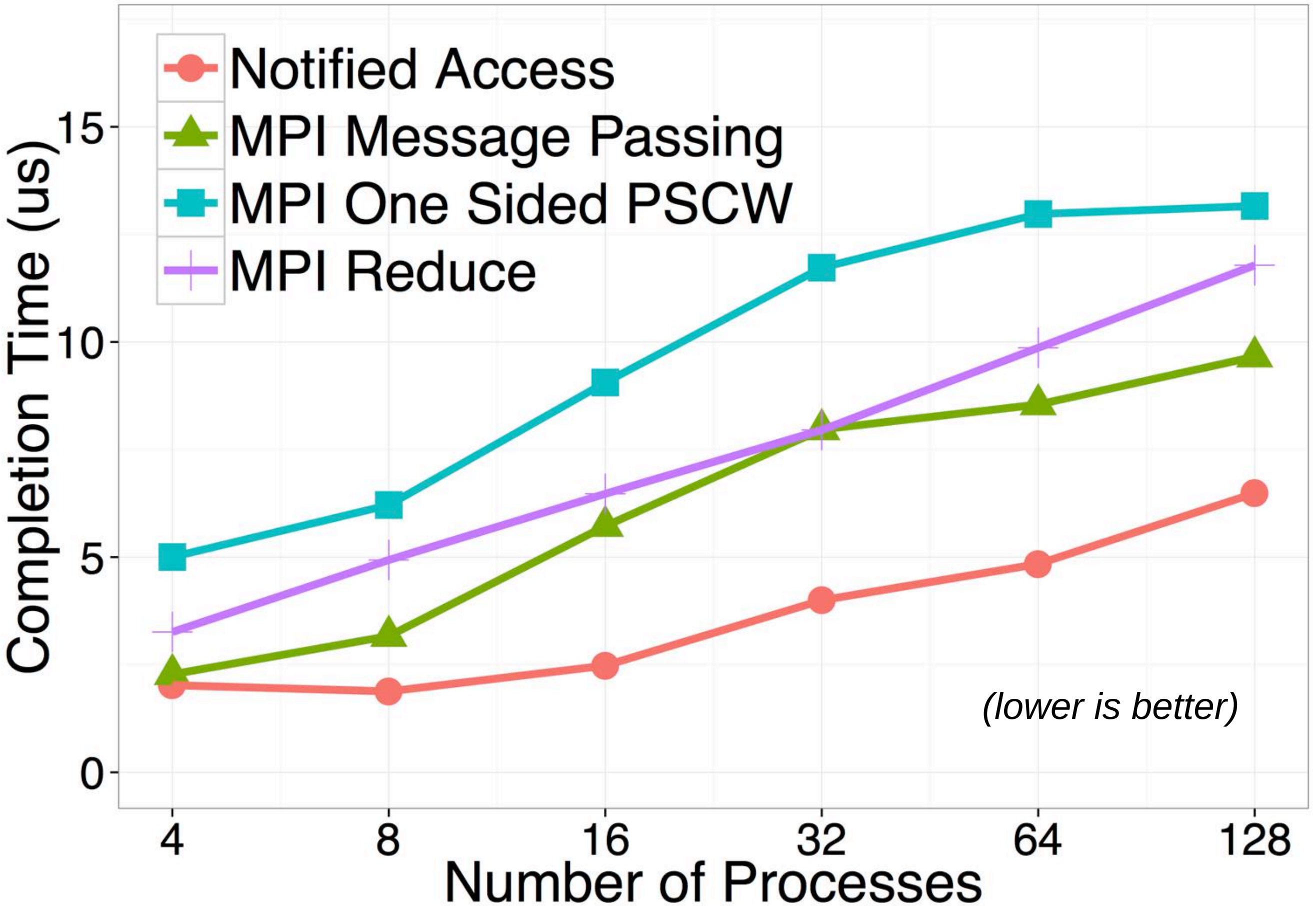


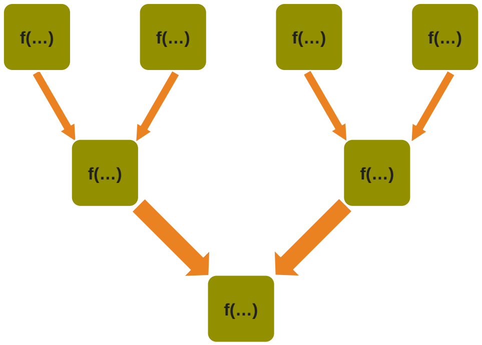


[1] https://github.com/intelesg/PRK2

Reduce – One-to-Many Synchronization

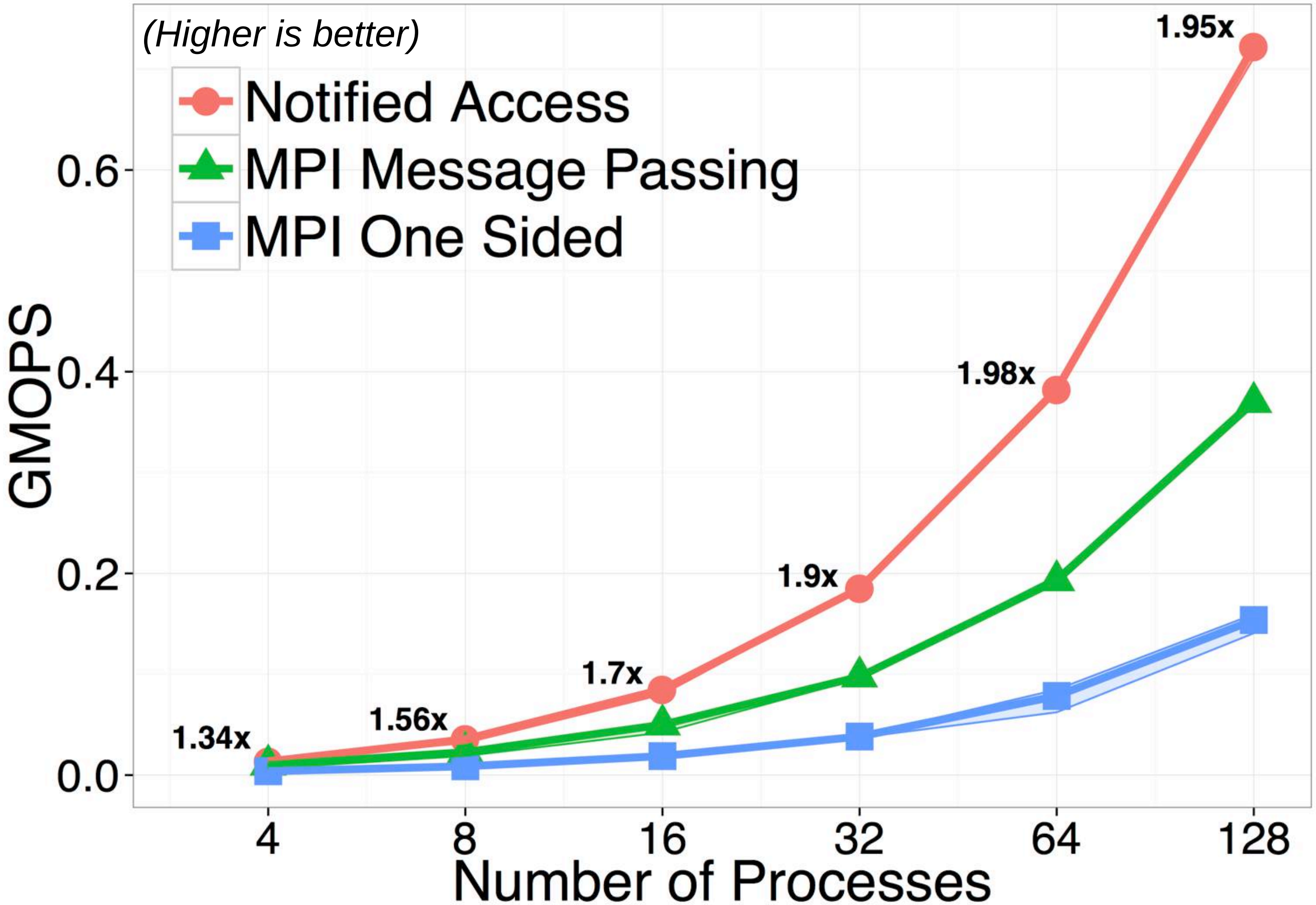
- **Reduce as an example (same for FMM, BH, etc.)**
  - Small data (8 Bytes), 16-ary tree
  - 1000 repetitions, each timed separately with RDTSC

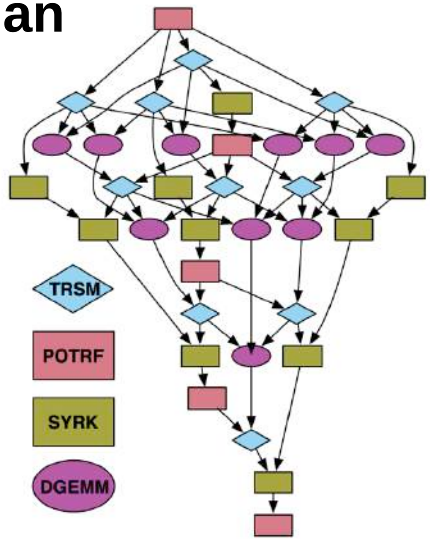




Cholesky – Many-to-Many Synchronization

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





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

Discussion and Conclusions

- **Simple and fast solution**
  - The interface lies between RMA and Message Passing
  - Similarity to MPI-1 eases adoption of NA
  - Richer semantics then current notification systems
  - Maintains benefits of RDMA for producer/consumer
- **Effect on other RMA operations needs to be defined**
  - Either synchronizing [1] or no effect
  - Currently discussed in the MPI Forum
- **Fully parameterized LogGP-like performance model**

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

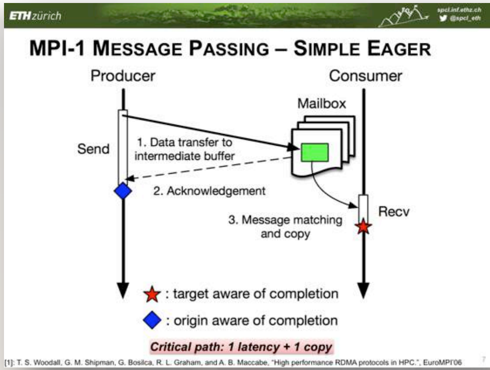
Function	Time
MPI_Notify_init	$t_{init} = 0.07\mu s$
MPI_Request_free	$t_{free} = 0.04\mu s$
MPI_Start	$t_{start} = 0.008\mu s$
MPI_{Put|Get}_notify	$t_{na} = 0.29\mu s$

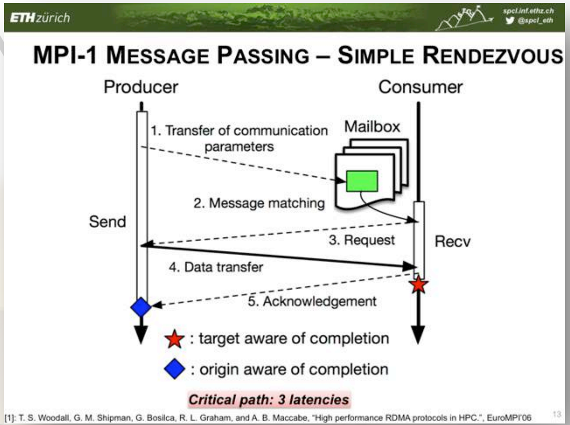
[1]: Kourosh Gharachorloo, et al.. "Memory consistency and event ordering in scalable shared-memory multiprocessors"., ISCA'90

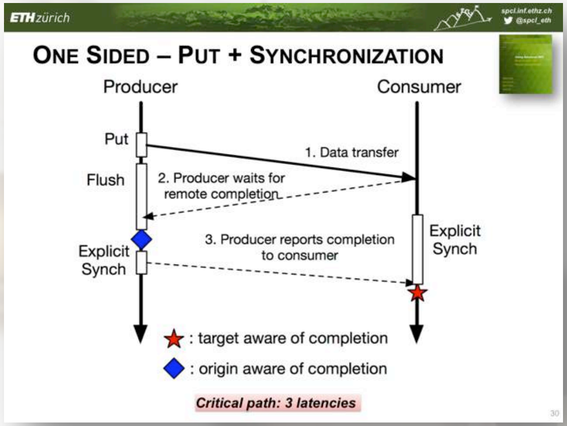
ACKNOWLEDGMENTS

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

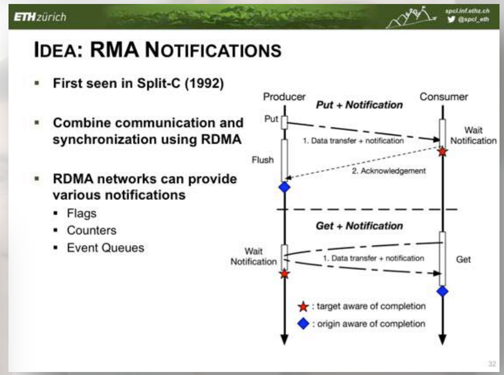
ACKNOWLEDGMEN

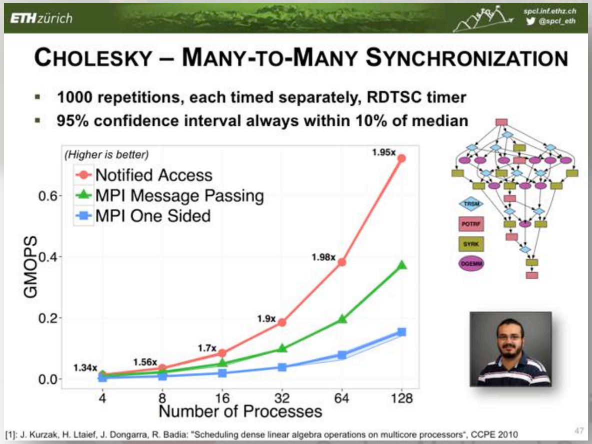


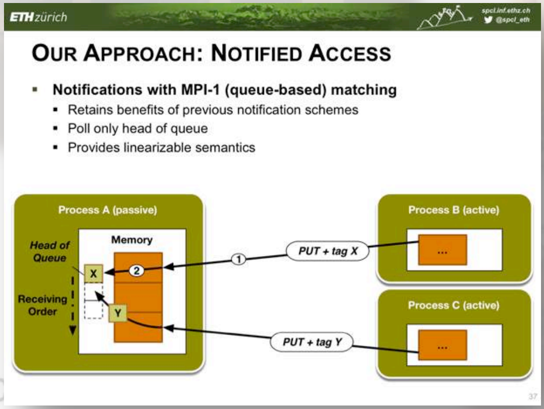




Thank you
for your attention







Swiss National Supercomputing Centre

BACKUP SLIDES

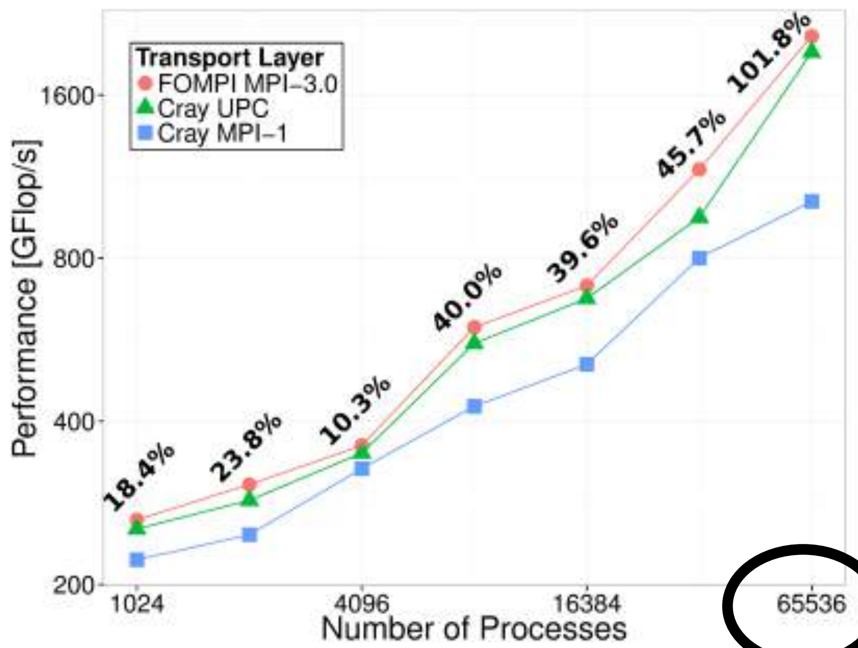
Notified Access - Example

```
MPI_Win win;
MPI_Request notification_request;
MPI_Status notification_status;
int win_size = 2 * MAX_SIZE * sizeof(double);
double *buf; int my_rank;
MPI_Win_allocate(win_size, sizeof(double), MPI_INFO_NULL, MPI_COMM_WORLD, &buf, &win);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
/* initialize notification request */
int customTag = 99; int expected_count = 1; int assert = 0;
MPI_Notify_init(win, partner_rank, customTag, expected_count, &notification_request);
MPI_Win_lock_all(assert, win);
for(size=8; size<MAX_SIZE; size++) {
  if (my_rank==client_rank) {
    /* send ping */
    MPI_Put_notify(buf, size, MPI_DOUBLE, partner_rank, 0, size, MPI_DOUBLE, win, customTag);
    MPI_Win_flush(partner_rank,win);
    /* wait for pong */
    MPI_Start(&notification_request);
    MPI_Wait(&notification_request, &notification_status);
  } else { /* server */
    /* wait for ping */
    MPI_Start(&notification_request);
    MPI_Wait(&notification_request, &notification_status);
    /* send pong */
    MPI_Put_notify(buf, size,MPI_DOUBLE, partner_rank, MAX_SIZE, size,MPI_DOUBLE, win, customTag);
    MPI_Win_flush(partner_rank, win);
  }
} /* end of iterations */
MPI_Win_unlock_all(win);
MPI_Request_free(&notification_request);
MPI_Win_free(&win);
```

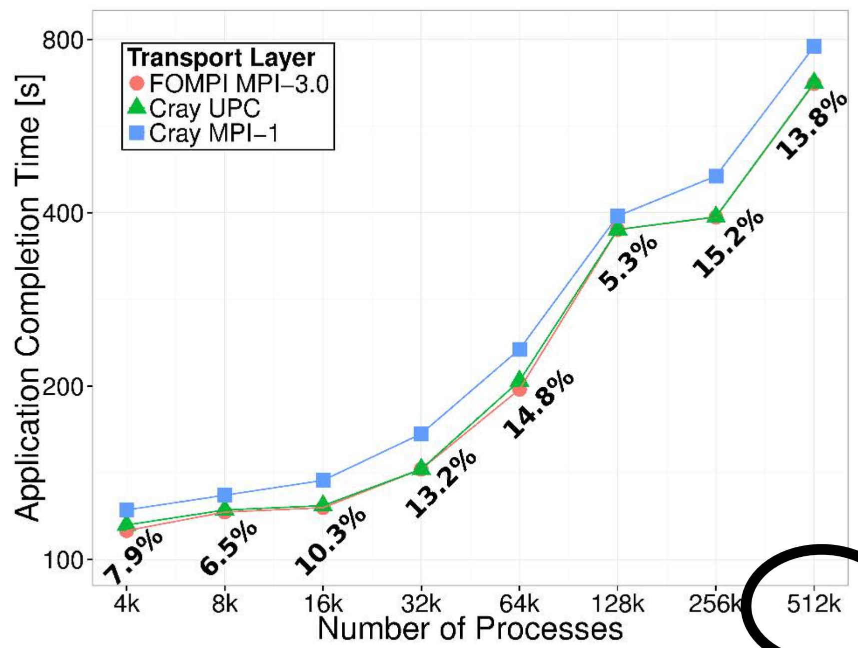
PERFORMANCE: APPLICATIONS

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

NAS 3D FFT [1] Performance



MILC [2] Application Execution Time



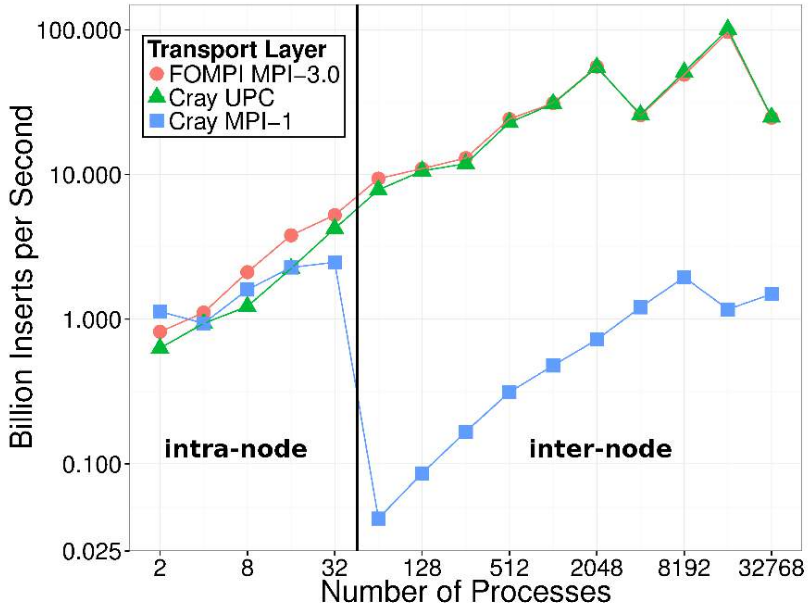
scale to 65k procs

scale to 512k procs

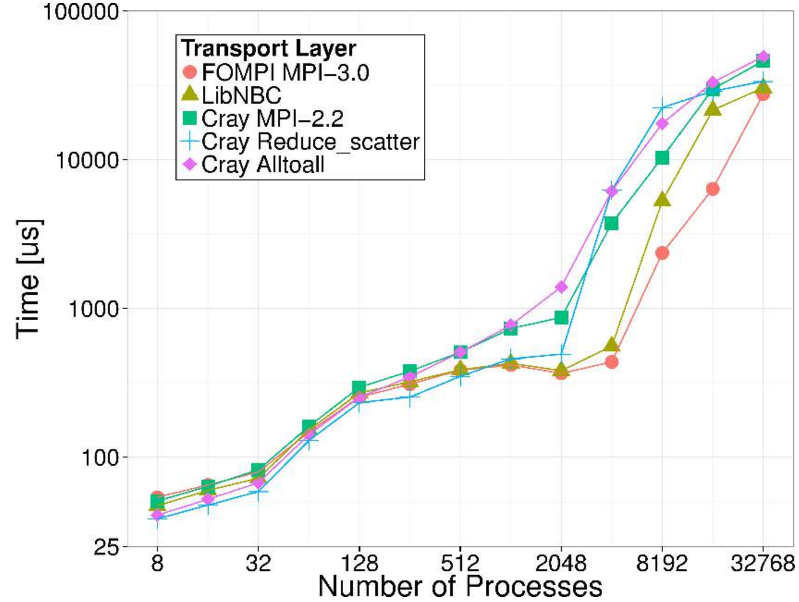
[1] Nishtala et al. Scaling communication-intensive applications on BlueGene/P using one-sided communication and overlap. IPDPS'09
[2] Shan et al. Accelerating applications at scale using one-sided communication. PGAS'12

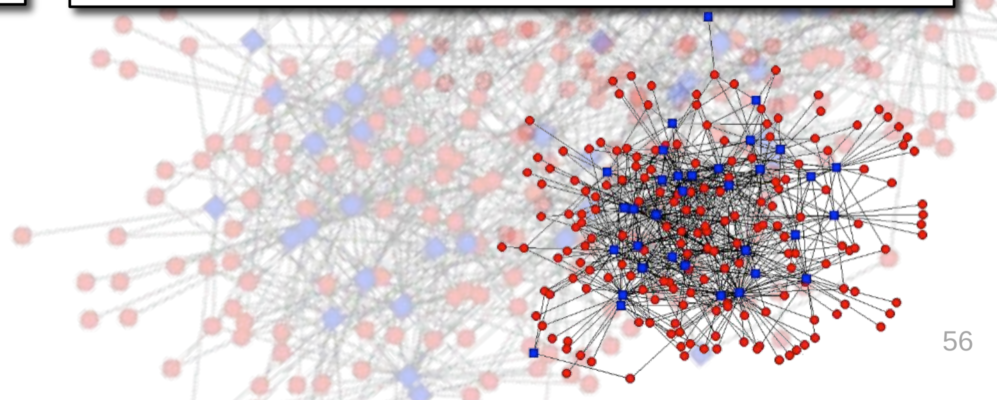
Performance: Motif Applications

Key/Value Store: Random Inserts per Second

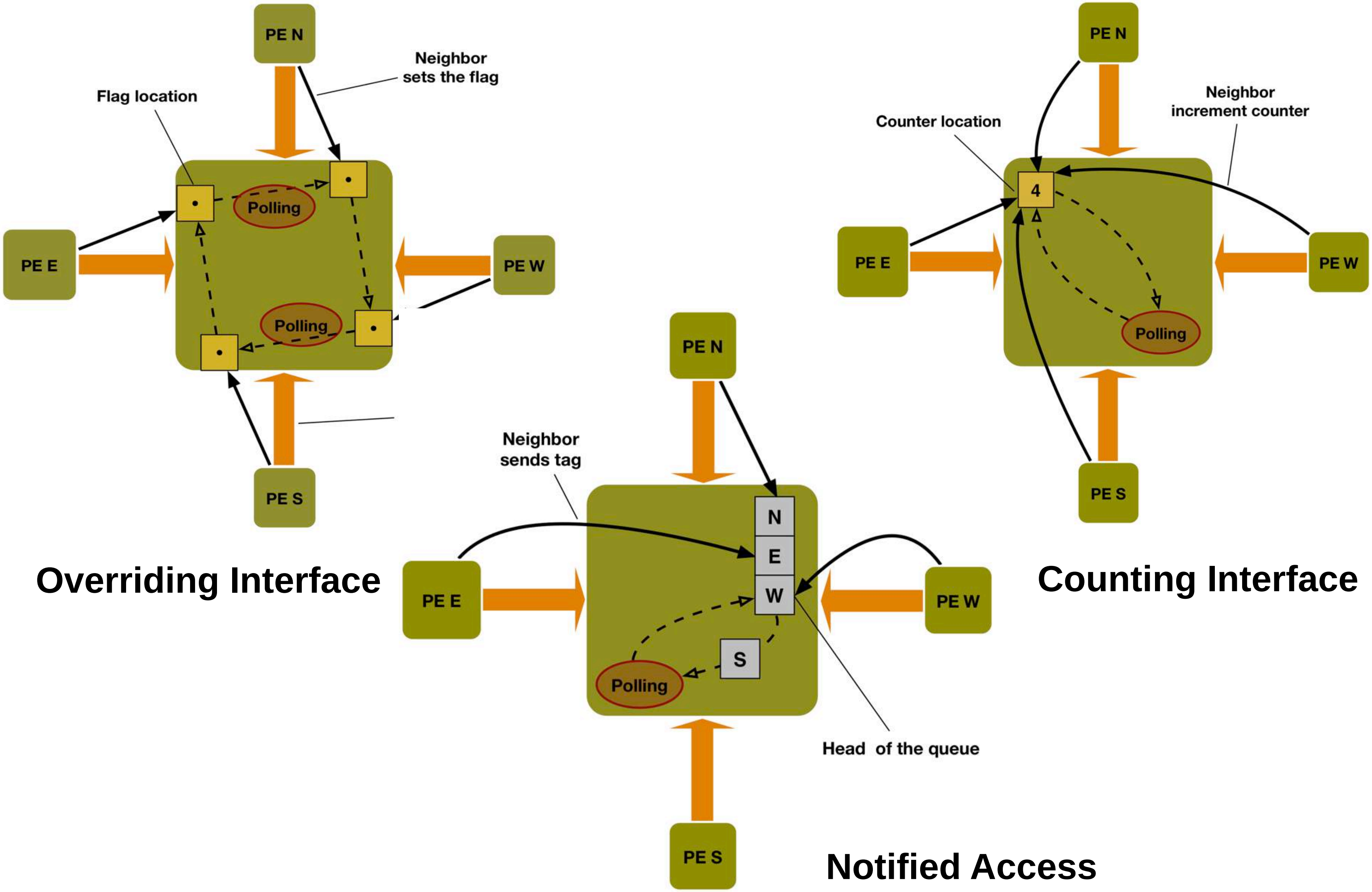


Dynamic Sparse Data Exchange (DSDE) with 6 neighbors

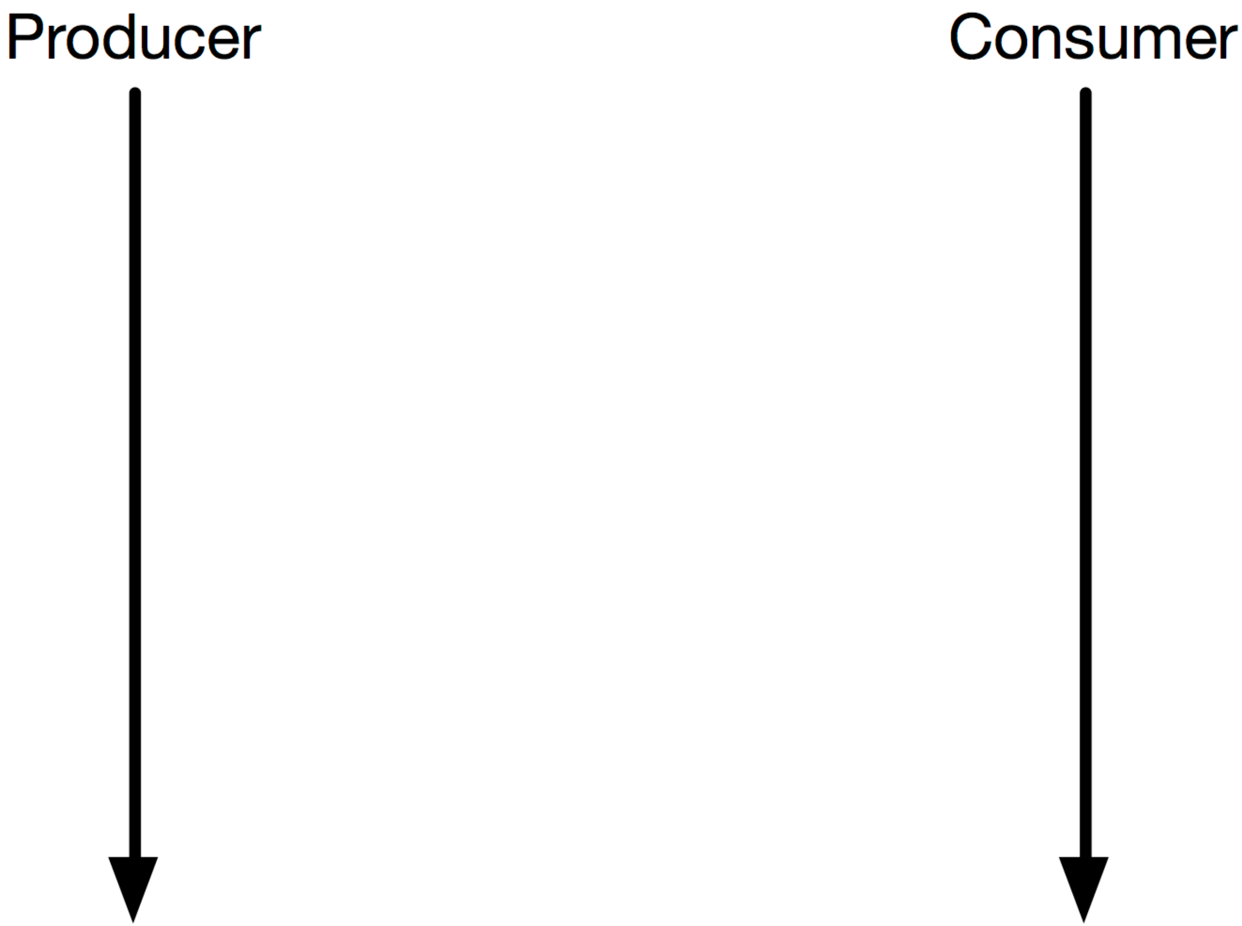




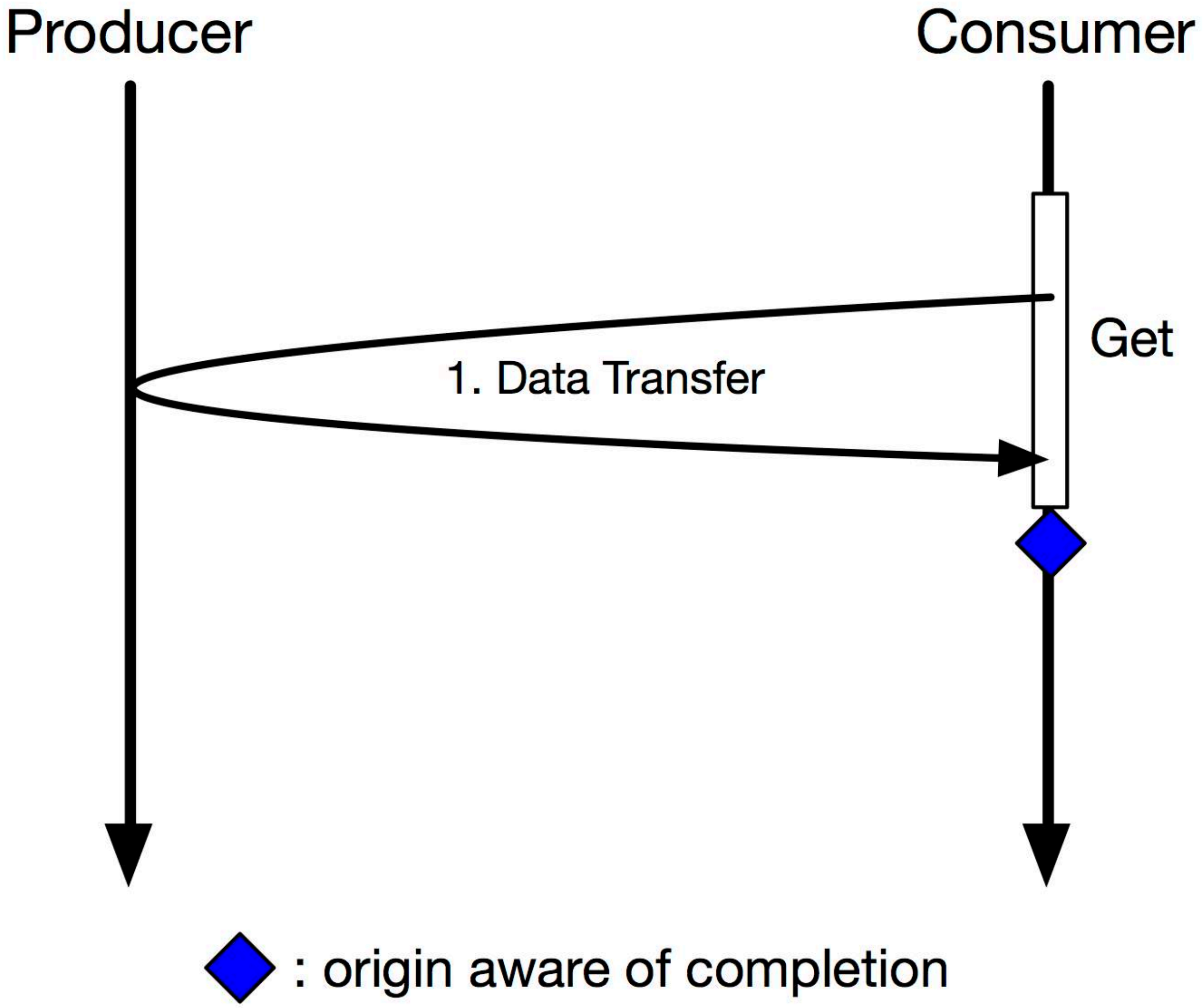
COMPARING APPROACHES – EXAMPLE



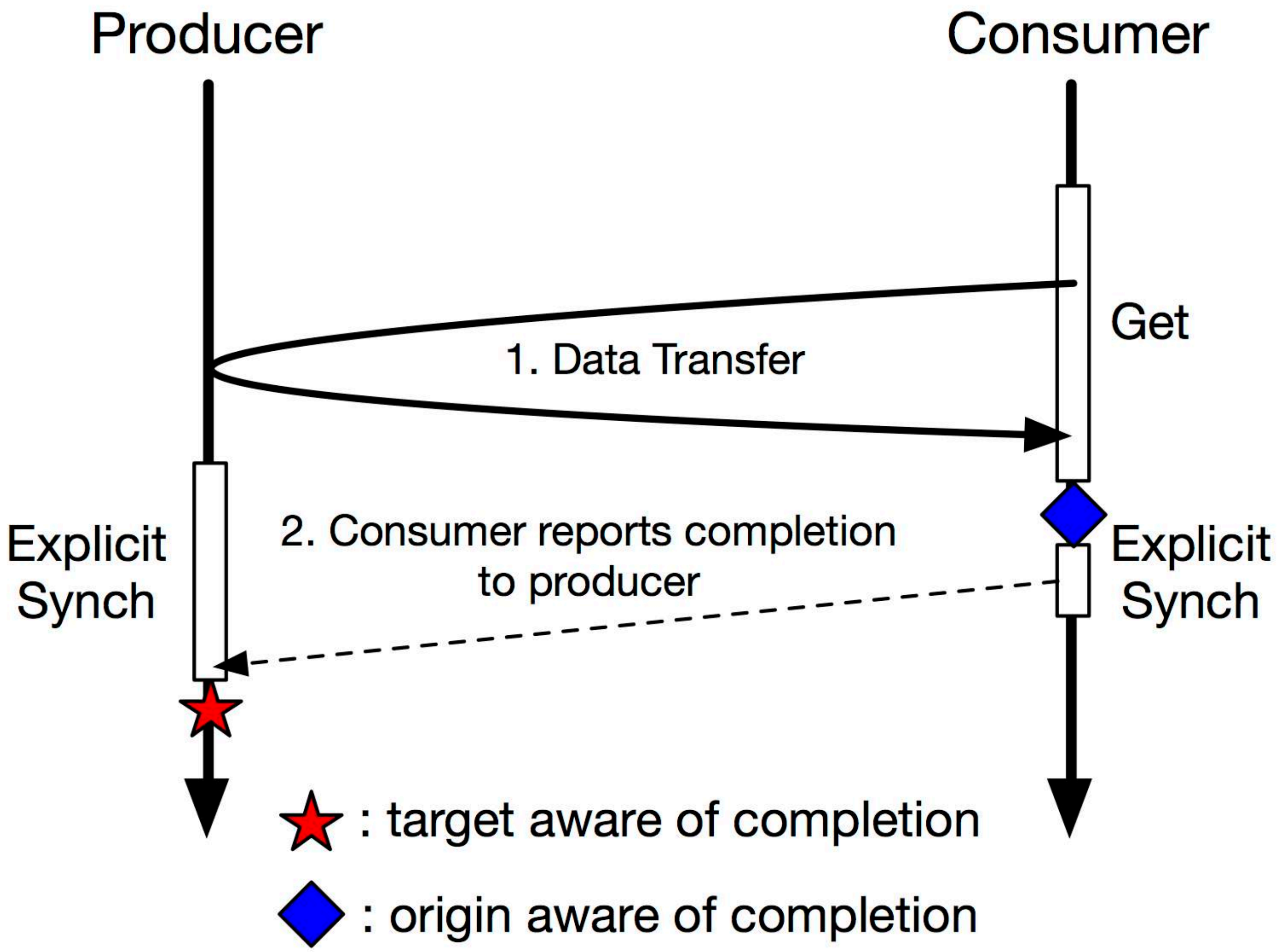
ONE SIDED – GET + SYNCHRONIZATION



One Sided – Get + Synchronization



One Sided – Get + Synchronization



Critical Path: 3 Messages

COMPARING APPROACHES