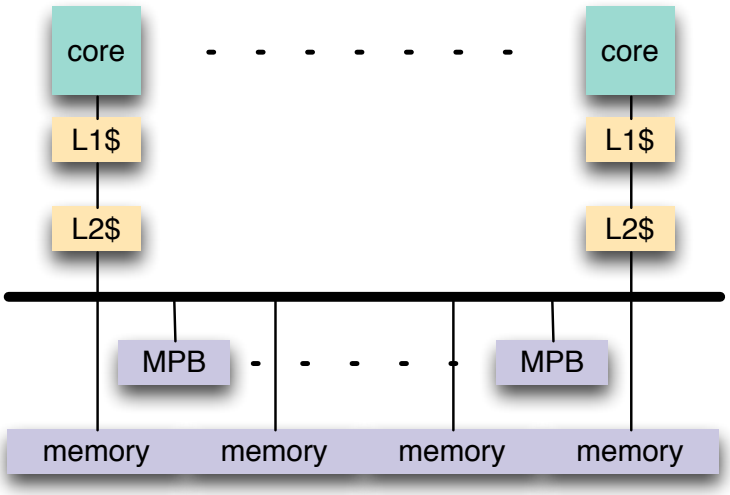
Brandenburg
University of Technology
Cottbus



10 April 2012

IN-MEMORY COMMUNICATION

EXPERIENCES WITH THE SCC

Randolf Rotta, Thomas Prescher, Jana Traue, Jörg Nolte

Brandenburg University of Technology Cottbus

OUTLINE

1. Why Messaging?
2. How we use it
3. Protocols on the Intel SCC
4. Benchmarks and Experiences

MESSAGES ARE FOR COORDINATION

Traditional usage

- transfer data
- synchronize running tasks
- initiate new tasks remotely

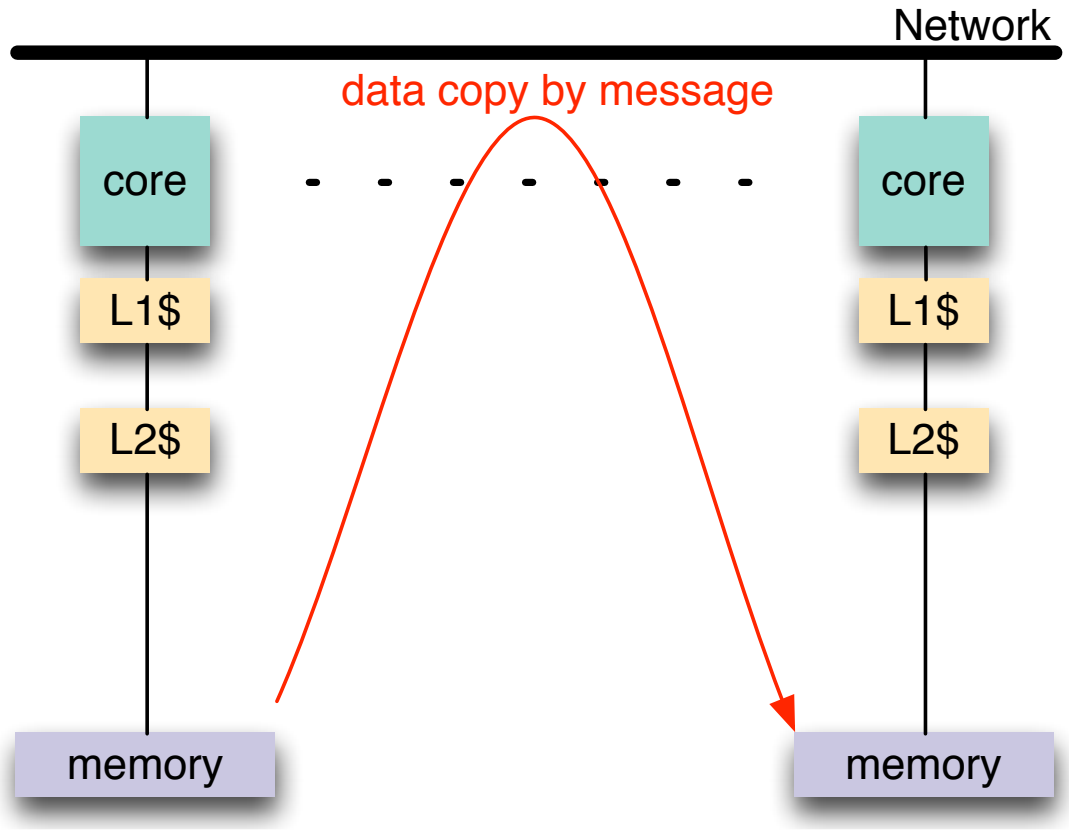
But: Many-core processors are hybrid systems

- distributed cores $\rightarrow$ message passing
- shared memory $\rightarrow$ data sharing

$\Rightarrow$ combine benefits of both
no need for data transfers, messages only for coordination

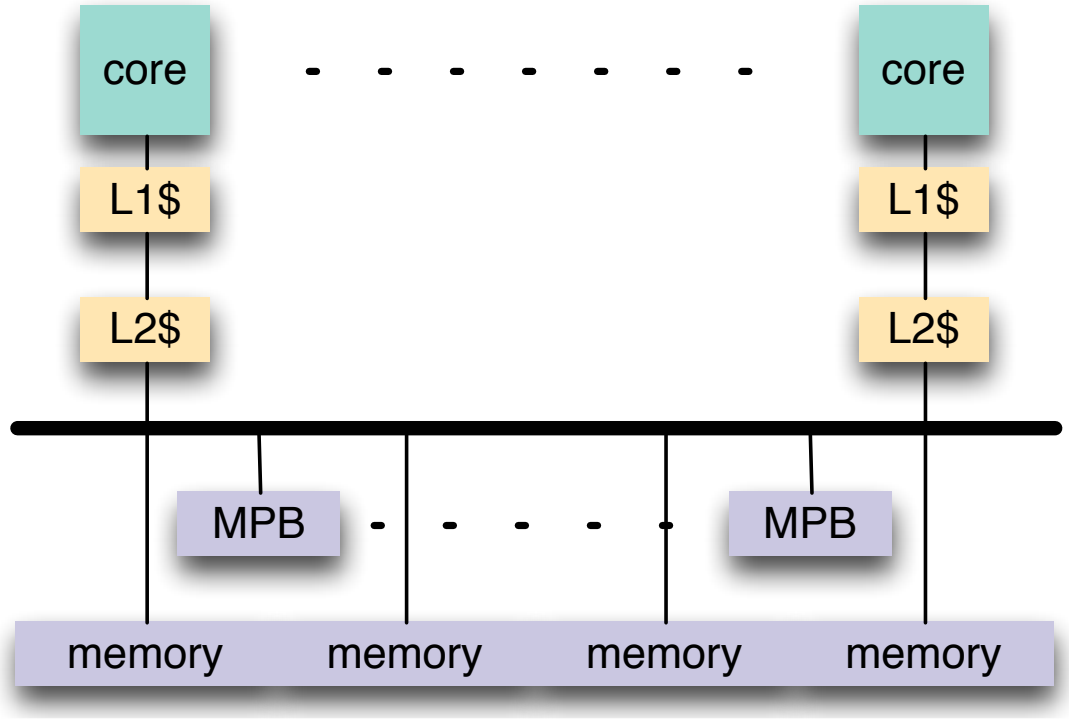
PAST: DISTRIBUTED MEMORY

Messages to copy data



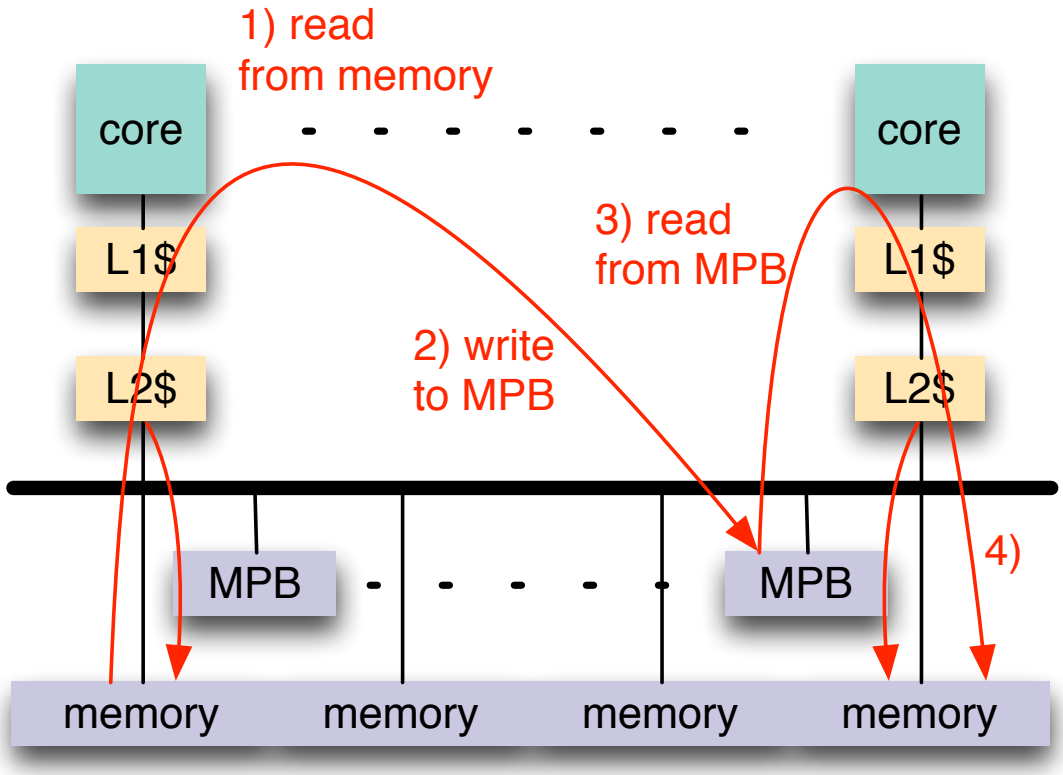
INTEL SCC: SHARED MEMORY, PRIVATE CACHES

On-Chip Network and Message Passing Buffers



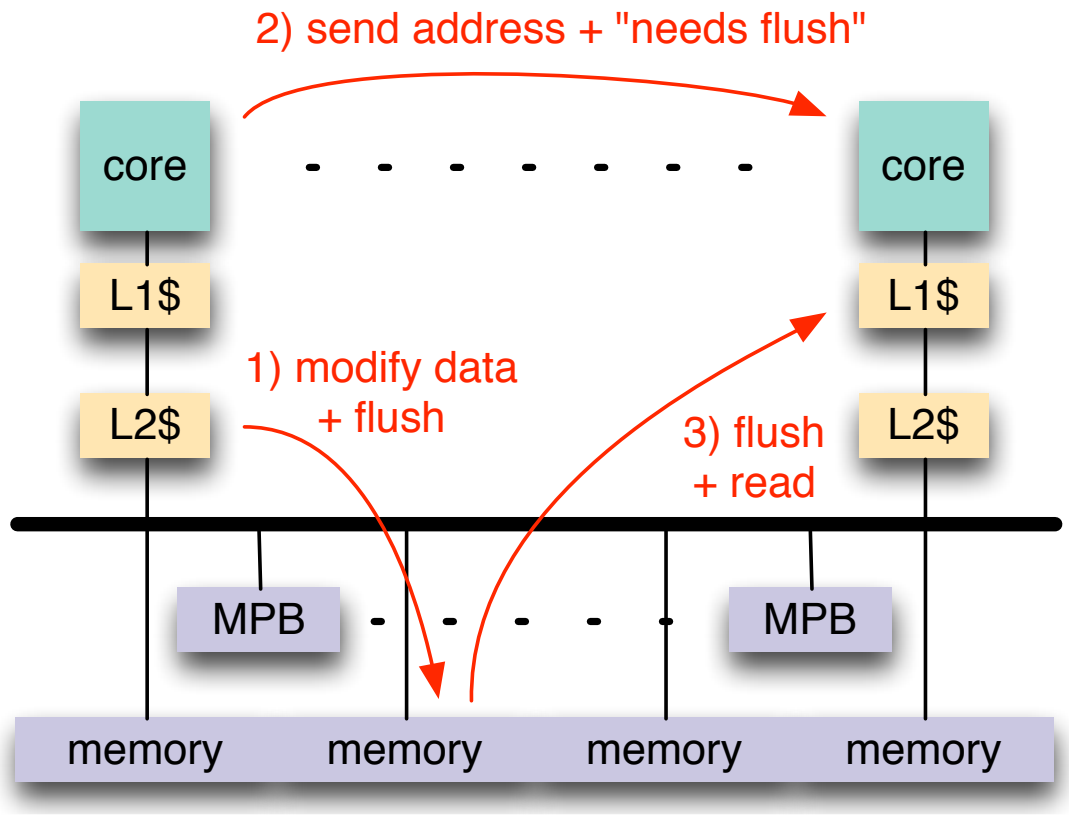
INTEL SCC: SHARED MEMORY, PRIVATE CACHES

Large messages push out cache contents ⇒ inefficient



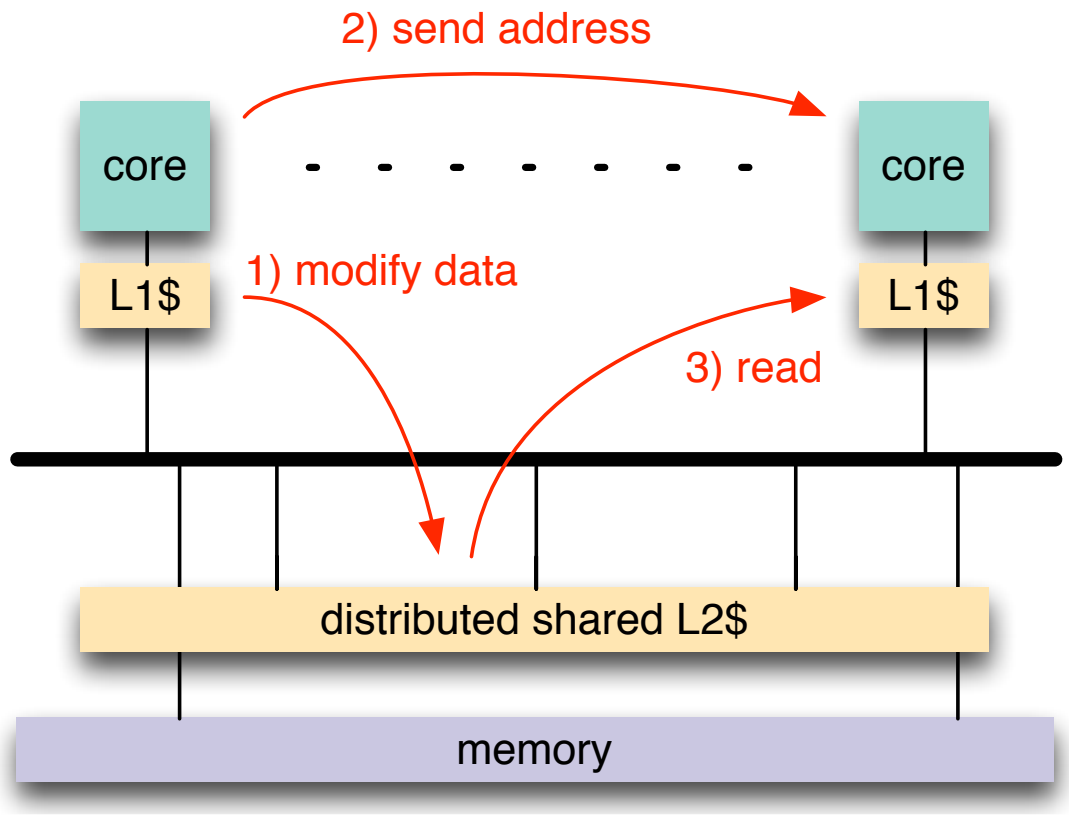
INTEL SCC: SHARED MEMORY, PRIVATE CACHES

Shared memory for data and messages



INTEL MIC: SHARED L2 CACHE

Shared L2$ for data; Messages stored in L2$



SUMMARY: MANY SMALL MESSAGES

- initiate tasks
- transfer addresses, parameters, results
- propagate consistency events
- synchronize running tasks

$\Rightarrow$ very small messages sufficient
but high number

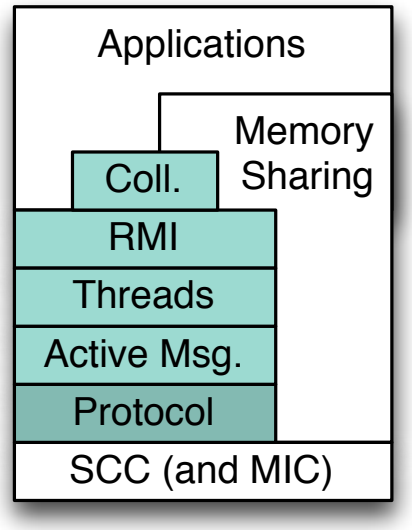
- examples: LRPC, Barrelfish, X10, ...

OUTLINE

1. Why Messaging?
2. **How we use it**
3. Protocols on the Intel SCC
4. Benchmarks and Experiences

SOFTWARE STACK: TACO

- aPGAS similar to X10, but pure C++
- global object pointers, remote method calls
- collective operations on groups
- based on cooperative user-level threads and active messages



Detail: Protocol Interface

- `send(dest, msg-ptr, length)`
  asynchronous
- `probe(receive-callback)`
  receive from any source
  callback is applied on each message

SOFTWARE STACK: MEMORY-EFFICIENT SHARING

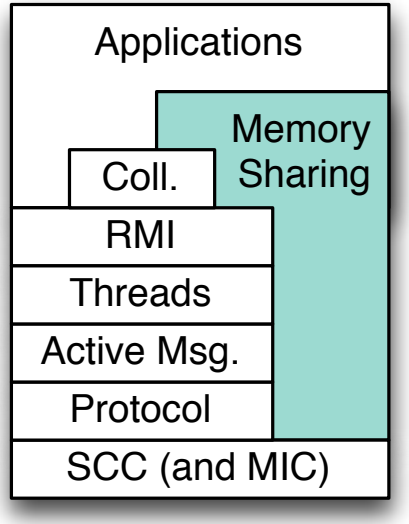
- SCC: LUT remapping, manual L2$ flushing

$\Rightarrow$ 2.5GB for shared objects

- hides coherence, replication, . . .
  using RMIs & collective operations internally

$\Rightarrow$ shared arrays, sets, graphs, . . .

- coming soon: parallel graph partitioning



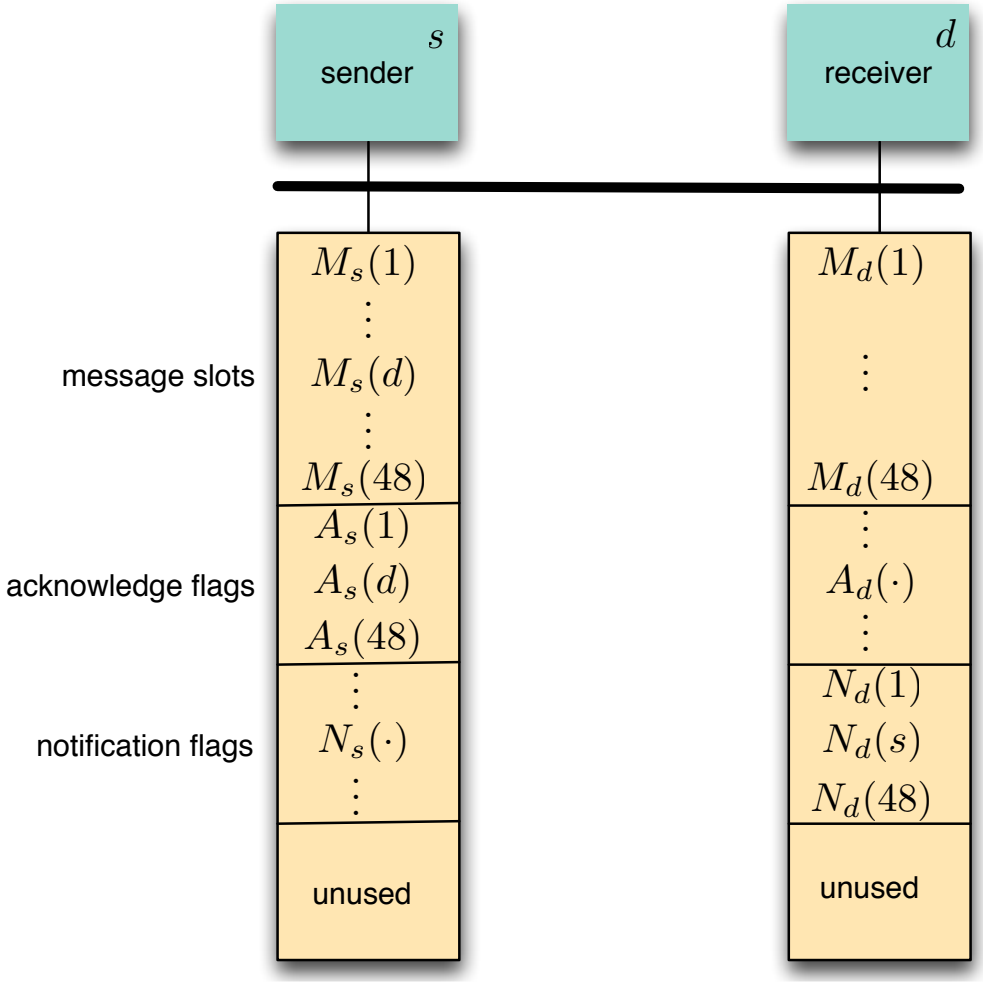
Observation

protocol performance limits level of parallelism

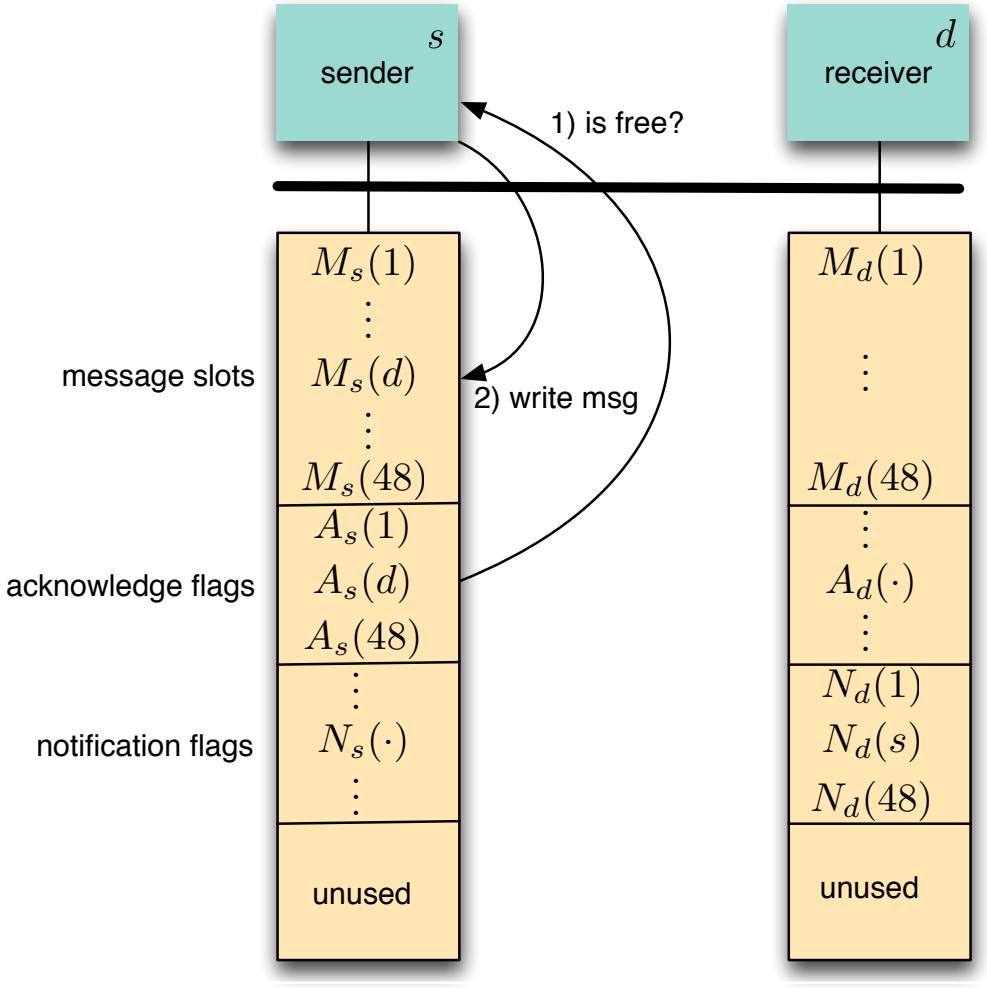
OUTLINE

1. Why Messaging?
2. How we use it
3. **Protocols on the Intel SCC**
4. Benchmarks and Experiences

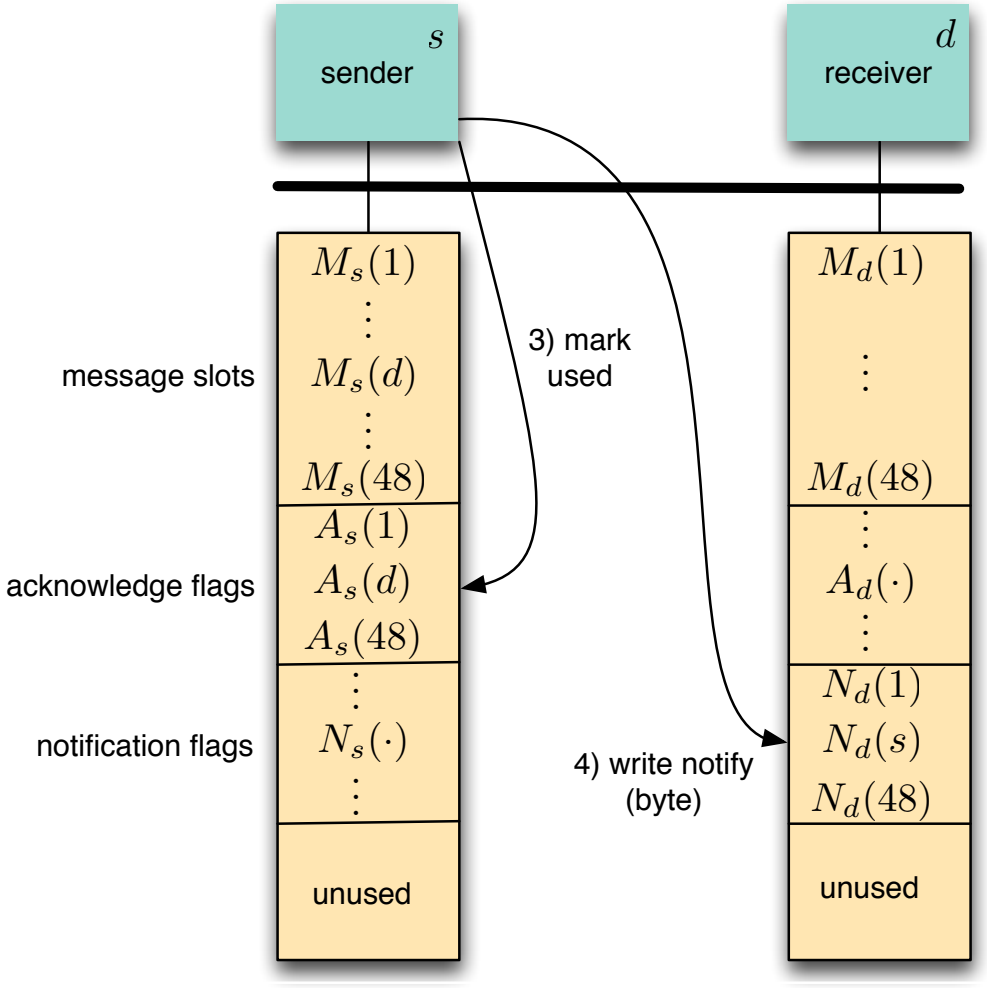
NVP: NOTIFICATION VECTOR PROTOCOL



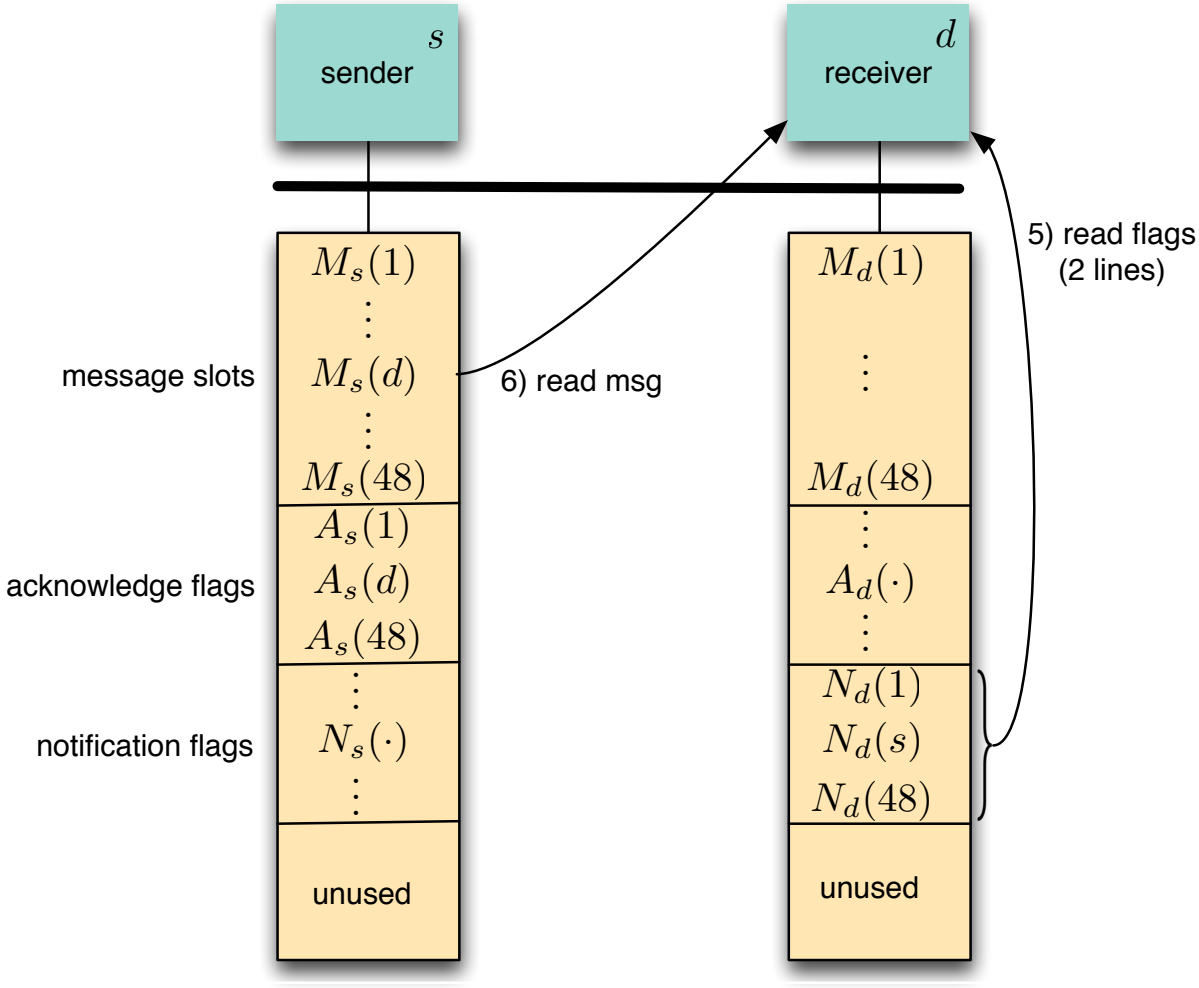
NVP: NOTIFICATION VECTOR PROTOCOL



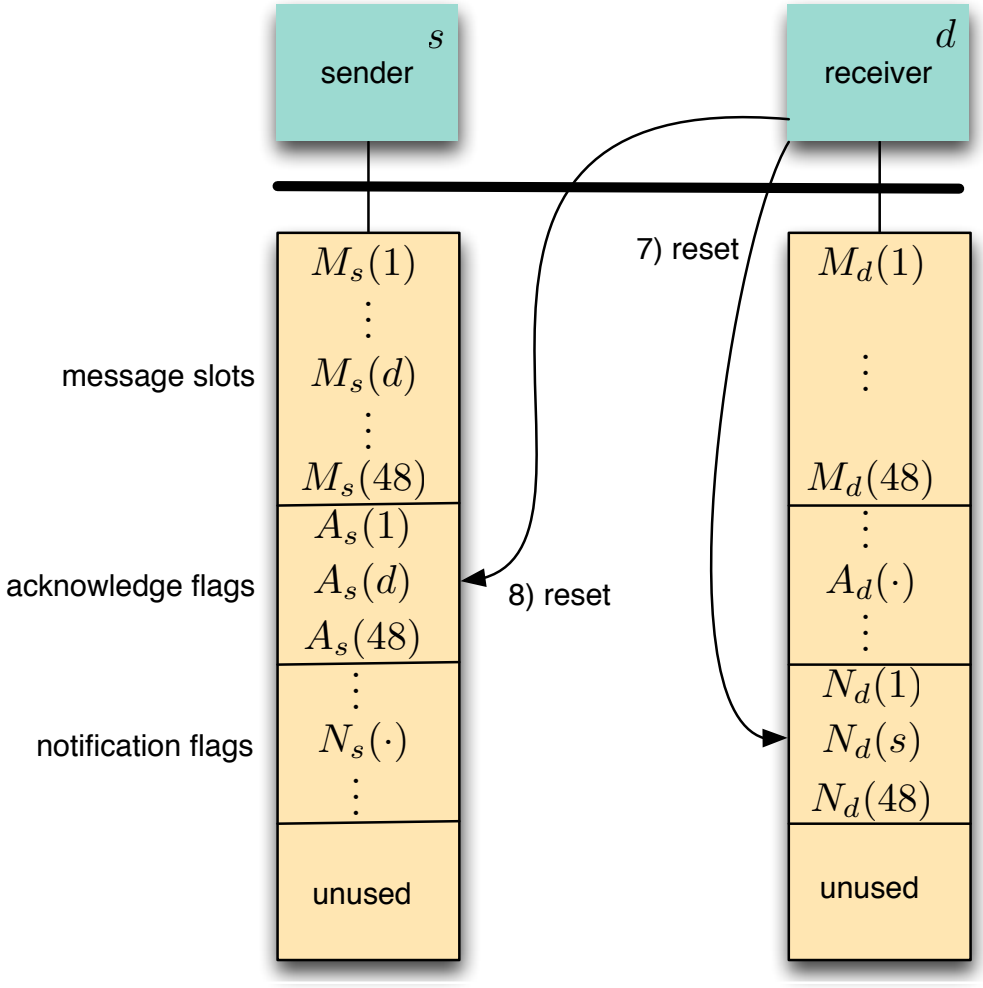
NVP: NOTIFICATION VECTOR PROTOCOL



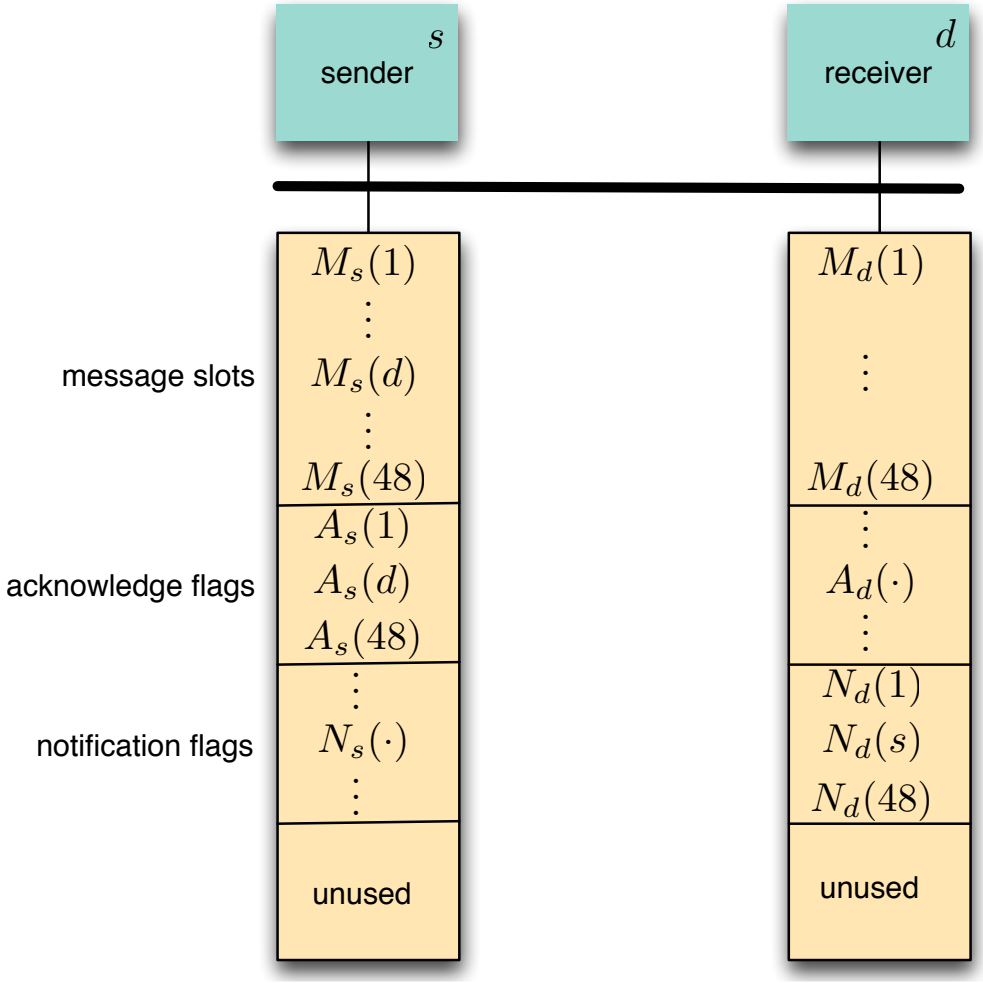
NVP: NOTIFICATION VECTOR PROTOCOL



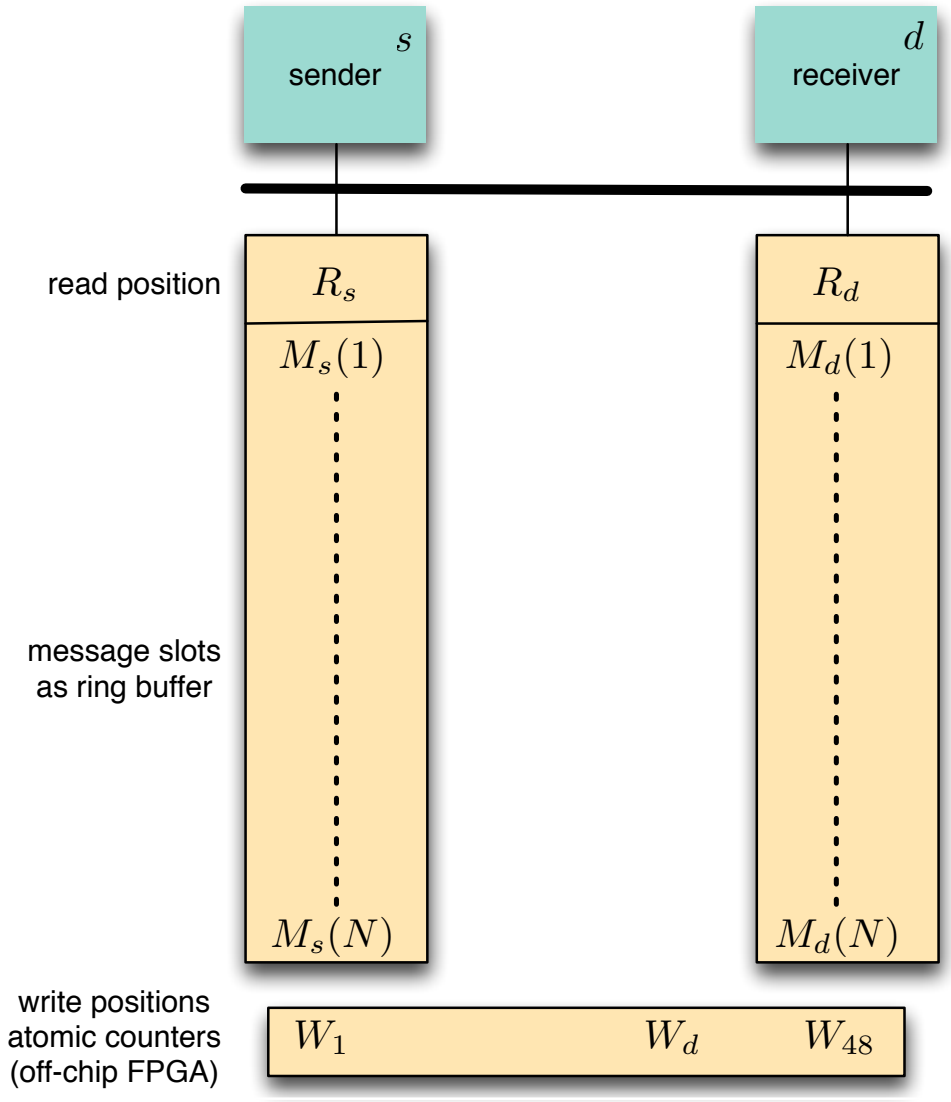
NVP: NOTIFICATION VECTOR PROTOCOL



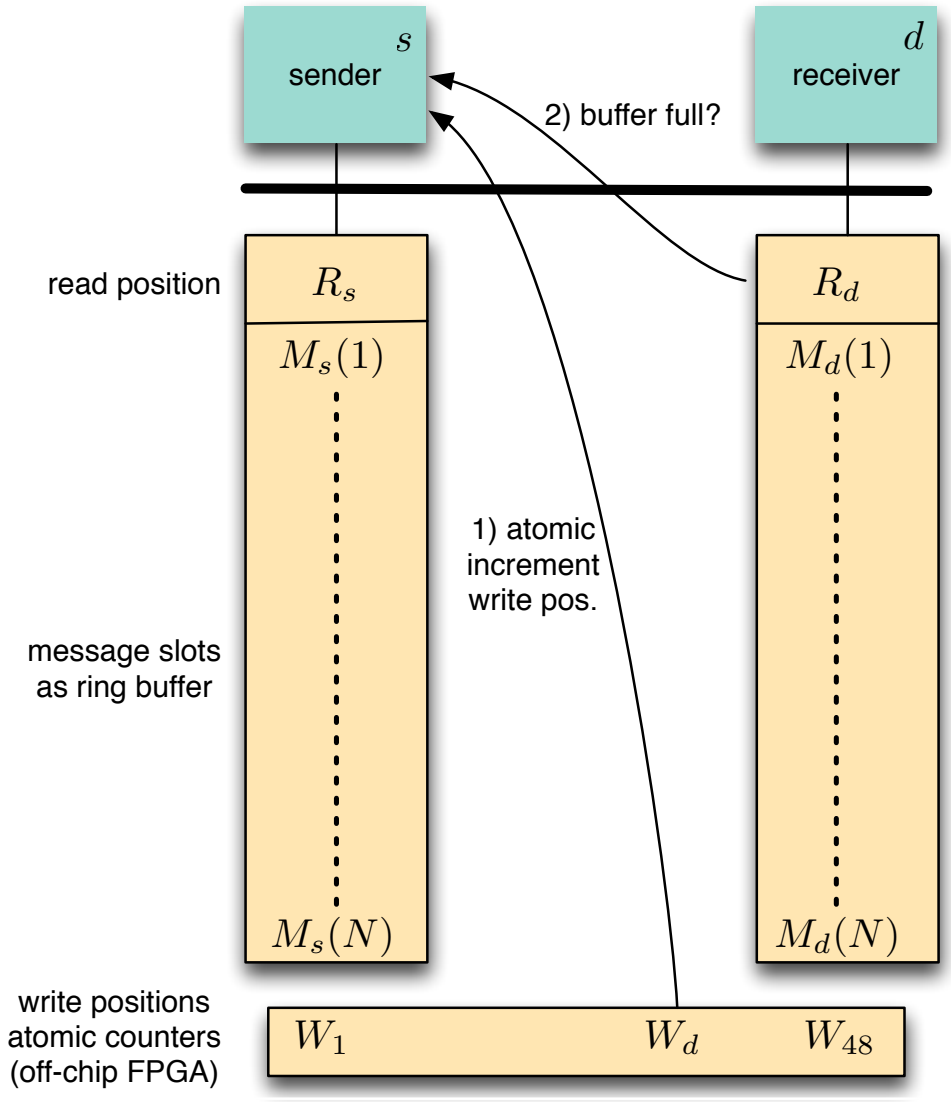
NVP: NOTIFICATION VECTOR PROTOCOL



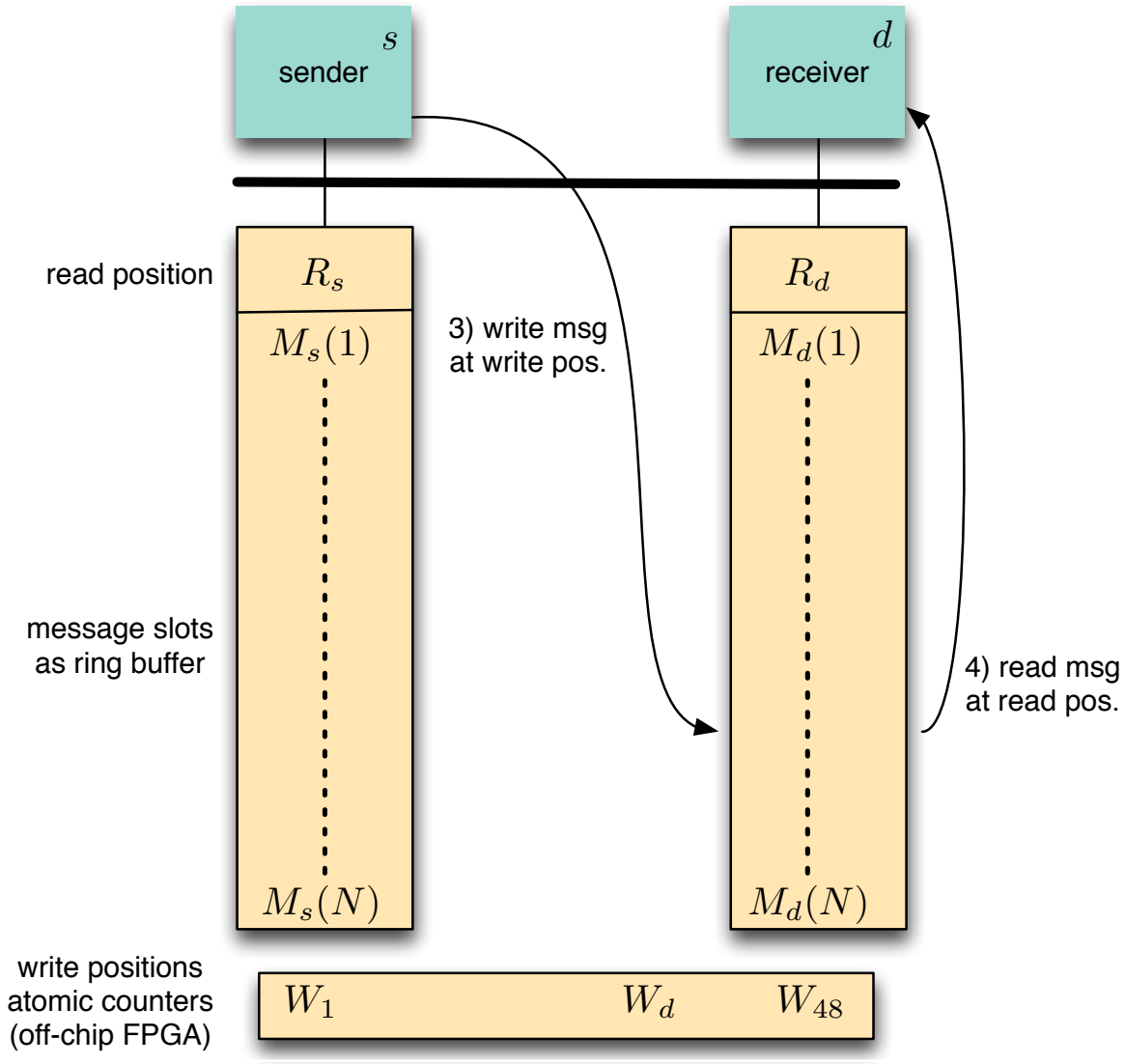
SRBP: SYNCHRONIZED RINGBUFFER PROTOCOL



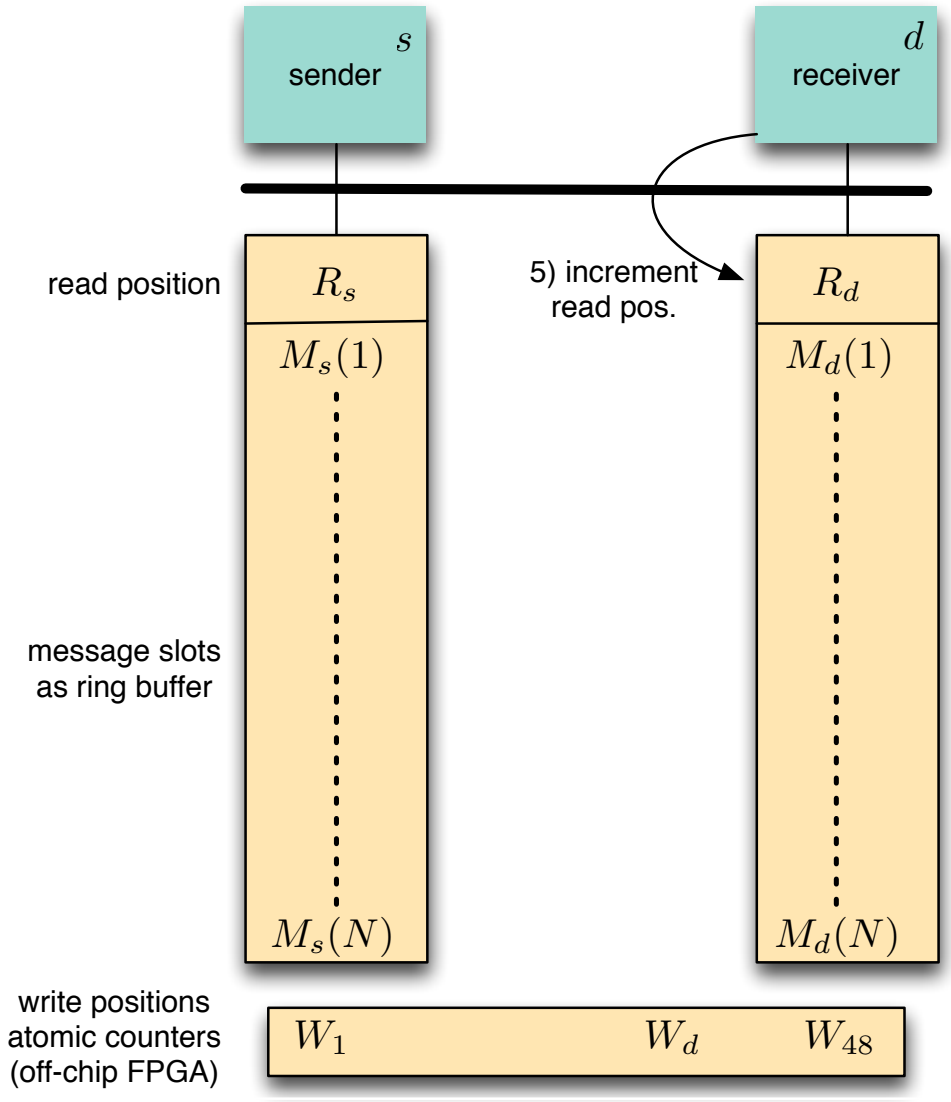
SRBP: SYNCHRONIZED RINGBUFFER PROTOCOL



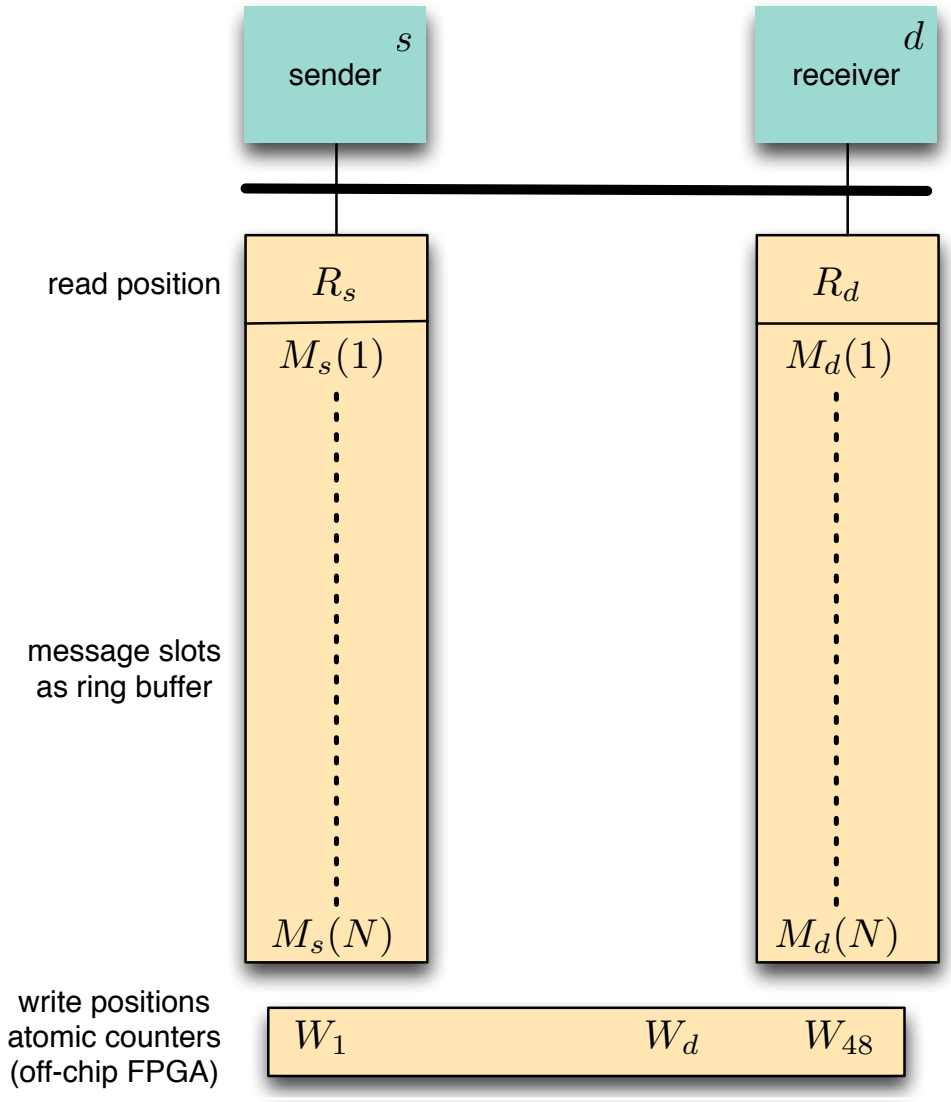
SRBP: SYNCHRONIZED RINGBUFFER PROTOCOL



SRBP: SYNCHRONIZED RINGBUFFER PROTOCOL



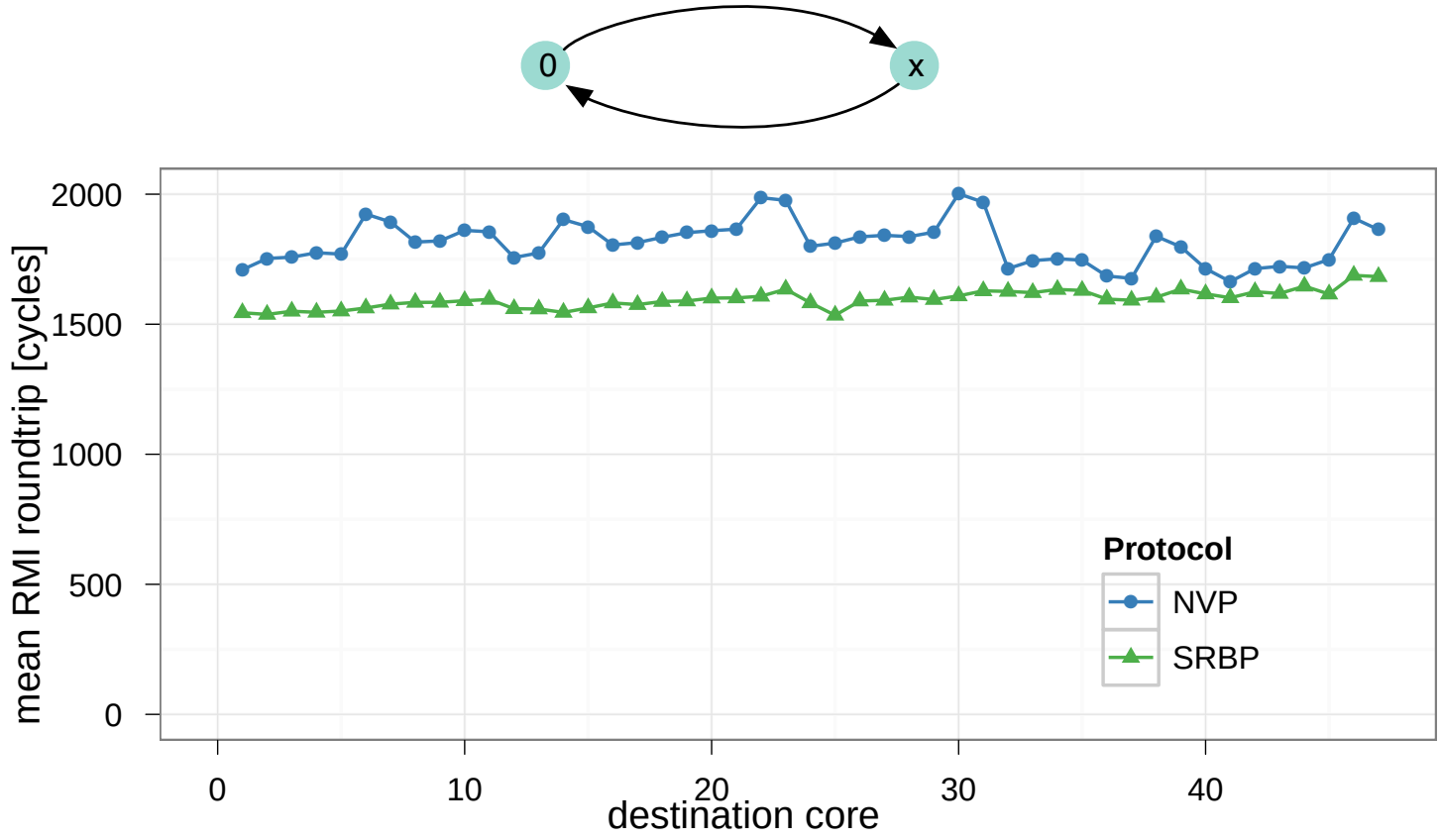
SRBP: SYNCHRONIZED RINGBUFFER PROTOCOL



OUTLINE

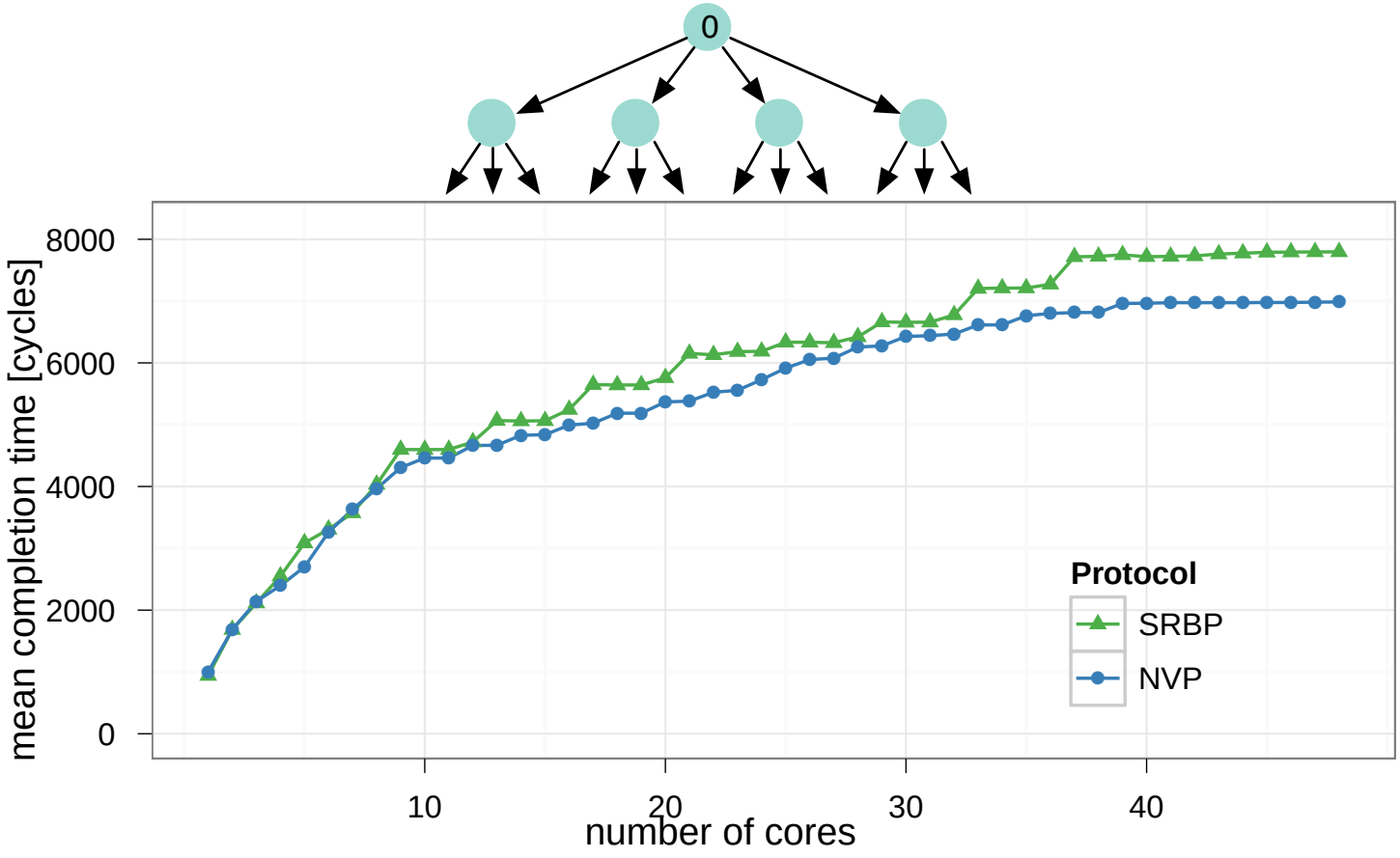
1. Why Messaging?
2. How we use it
3. Protocols on the Intel SCC
4. **Benchmarks and Experiences**

ROUNDTRIPS: <2000 CYCLES



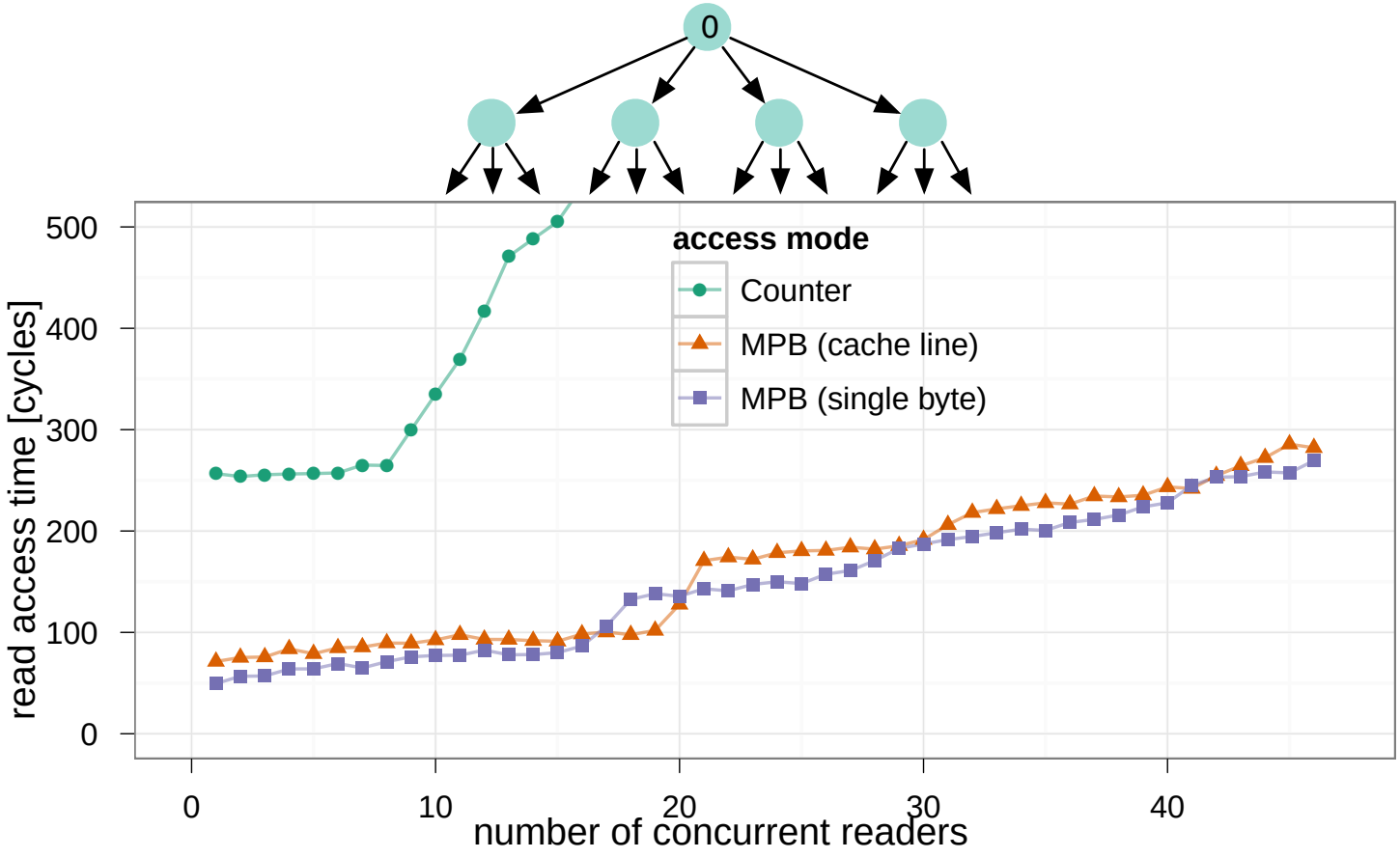
- SRBP >200 cycles faster than NVP
- distance has only little impact (10%)

COLLECTIVE OPERATIONS: LOGARITHMIC GROWTH



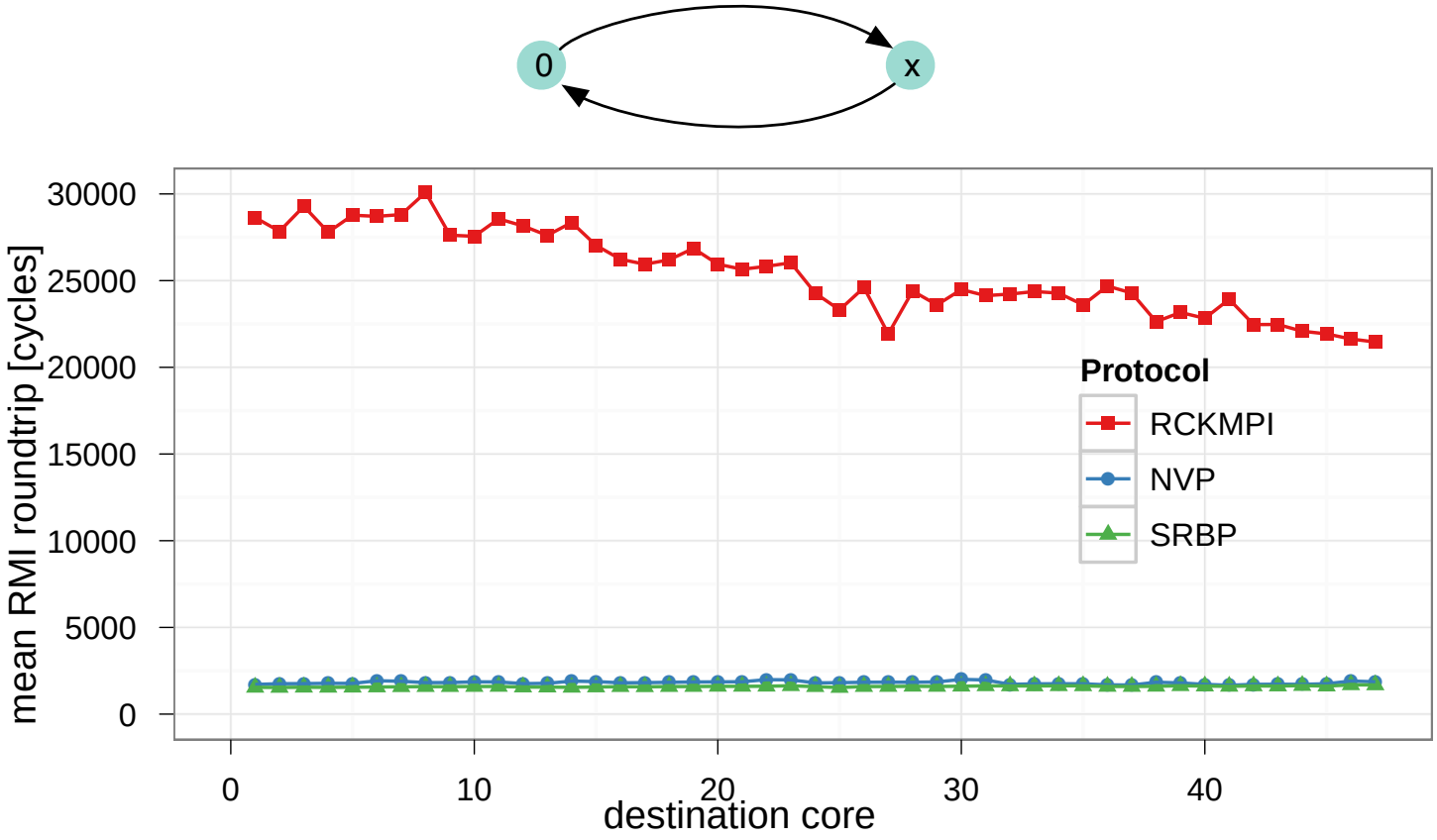
- constant overhead per core, interleaves with other tasks
- SRBP slower than NVP (counter congestion)

COLLECTIVE OPERATIONS: LOGARITHMIC GROWTH



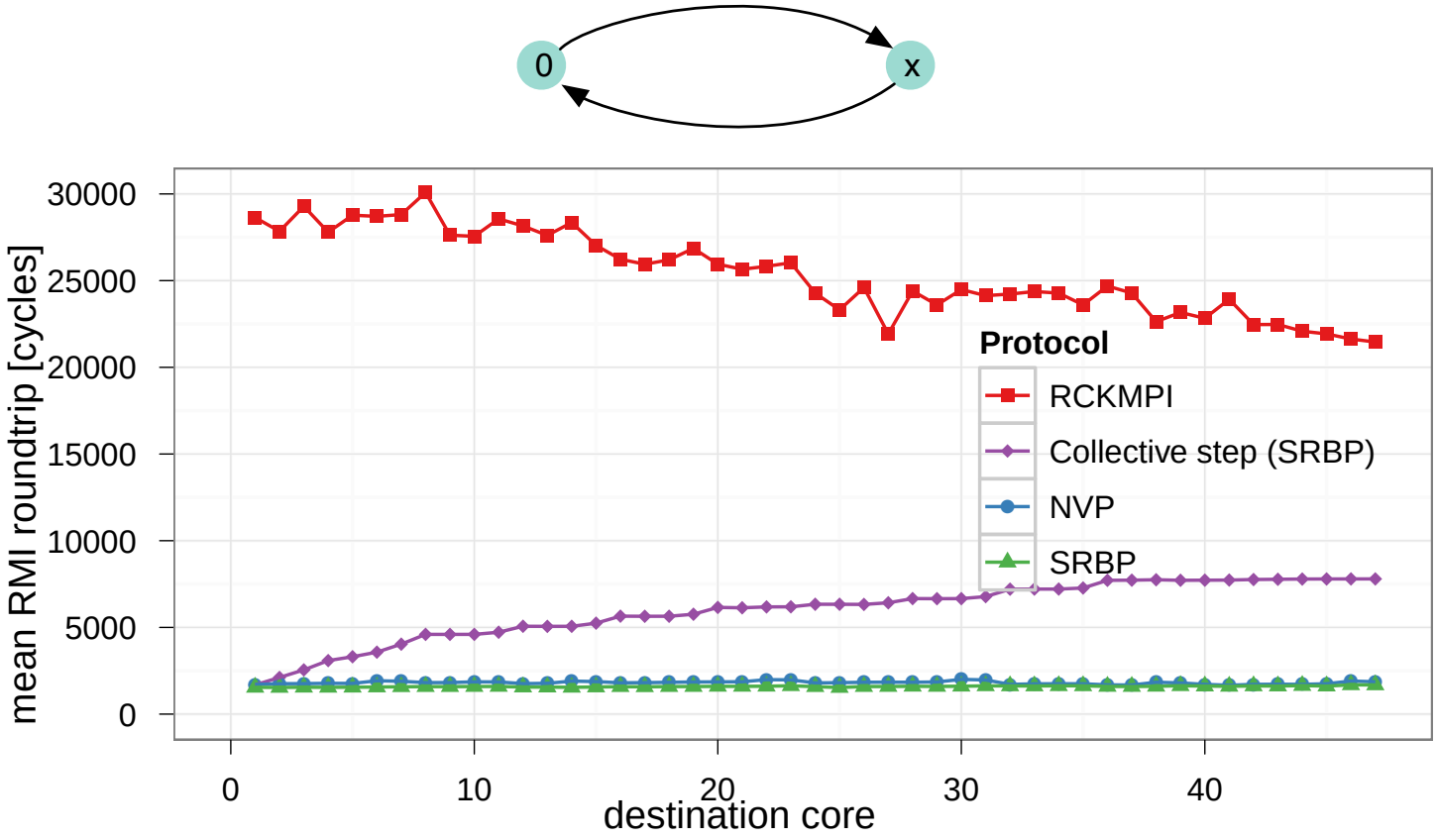
- constant overhead per core, interleaves with other tasks
- SRBP slower than NVP (counter congestion)

ROUNDTRIPS: 15X FASTER THAN RCKMPI



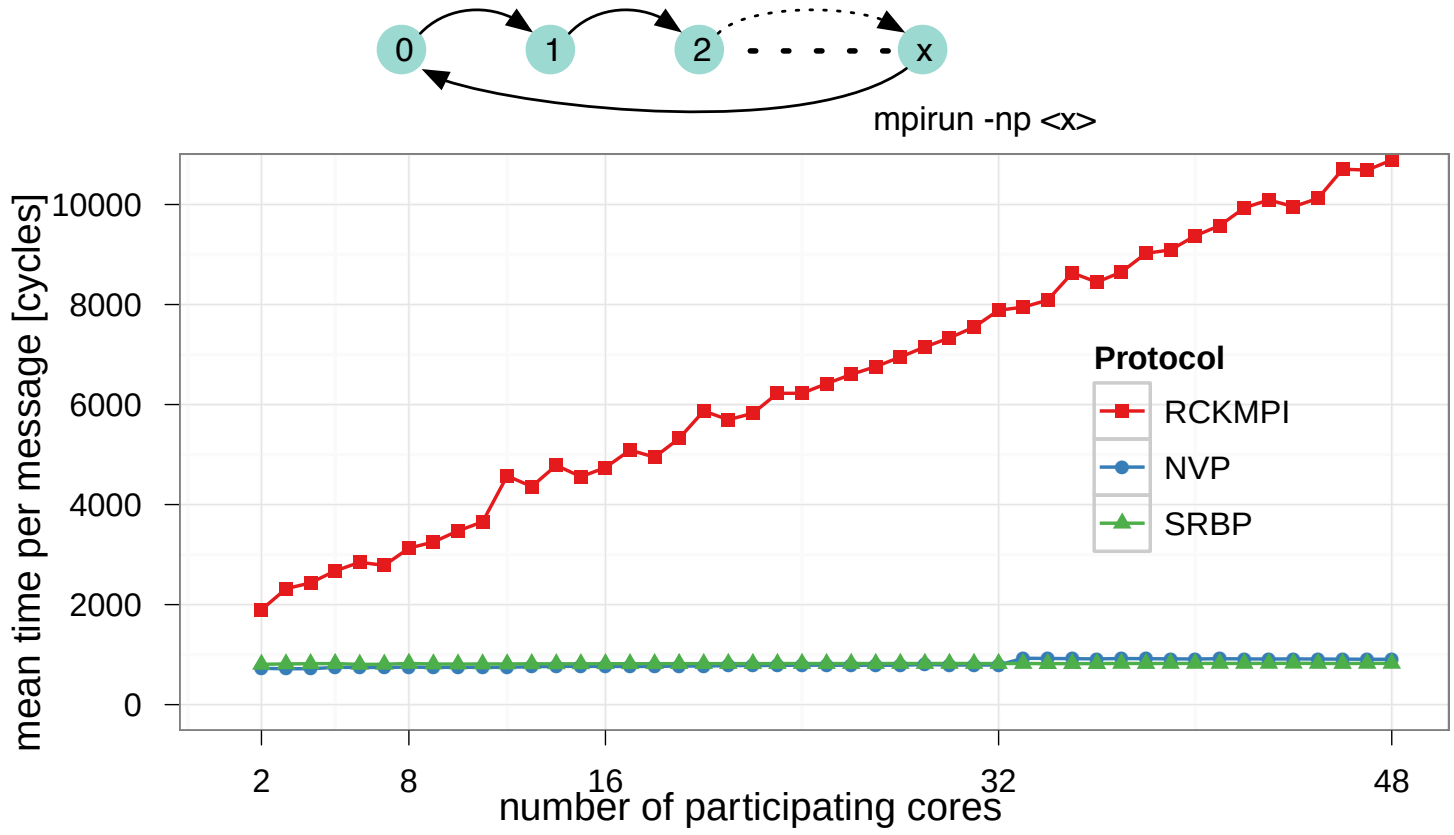
- MPI: supports large messages, tags, source-dest matching, . . .
- NVP/SRBP: offloading 15x smaller workloads still efficient

ROUNDTRIPS: 15X FASTER THAN RCKMPI



- MPI: supports large messages, tags, source-dest matching, . . .
- NVP/SRBP: offloading 15x smaller workloads still efficient

MAIN COST FACTOR: POLLING OVERHEAD

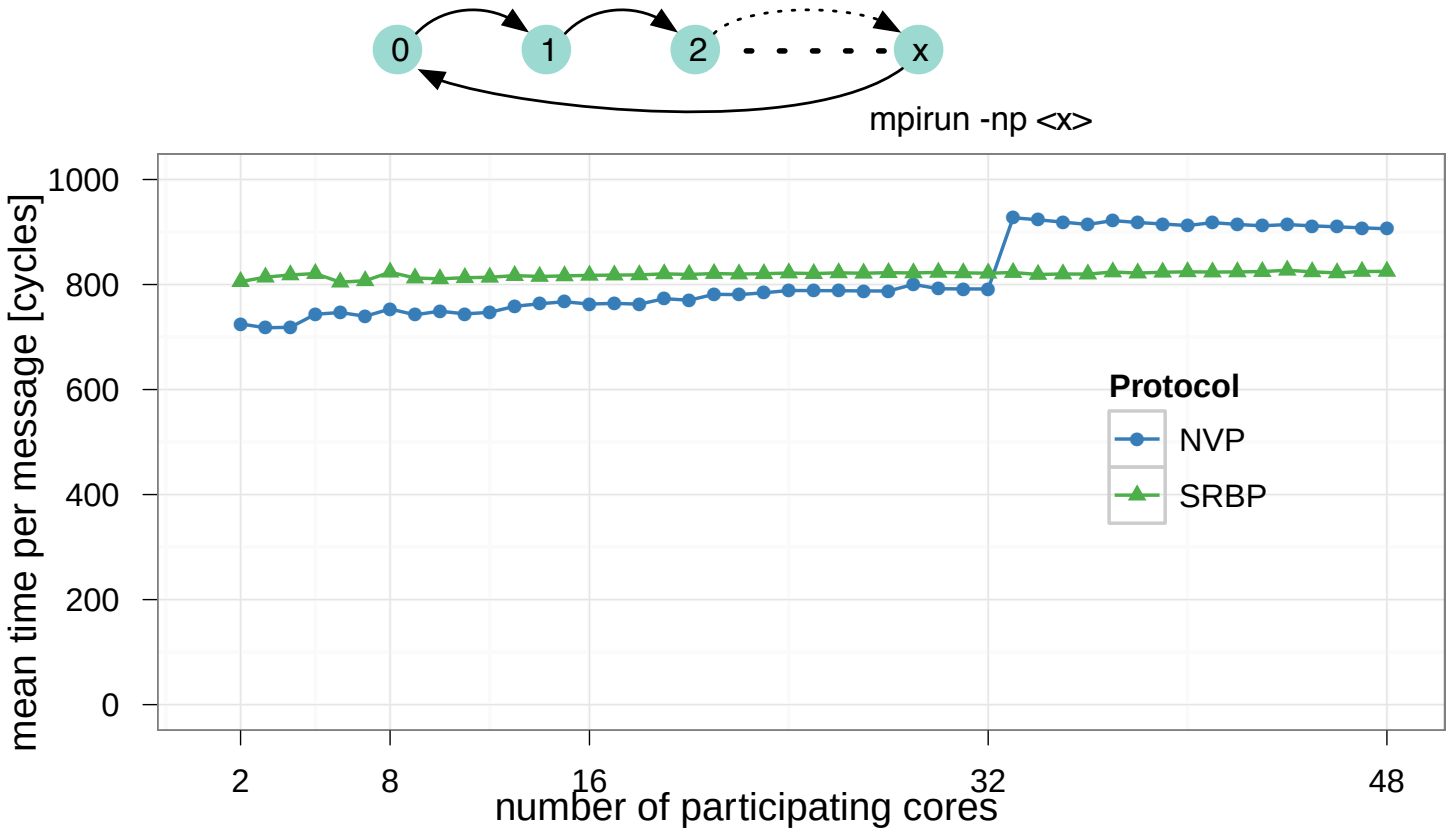


RCKMPI: polling one line per core, takes 200 cycles

NVP: polling one line per 32 cores

SRBP: polling on single word

MAIN COST FACTOR: POLLING OVERHEAD



RCKMPI: polling one line per core, takes 200 cycles

NVP: polling one line per 32 cores

SRBP: polling on single word

CONCLUSION: EFFICIENCY THROUGH SPECIALIZATION

- many-cores are hybrid systems

$\Rightarrow$ combine shared memory and message passing

- just small & simple messages

$\Rightarrow$ simpler & faster protocols, more parallelism

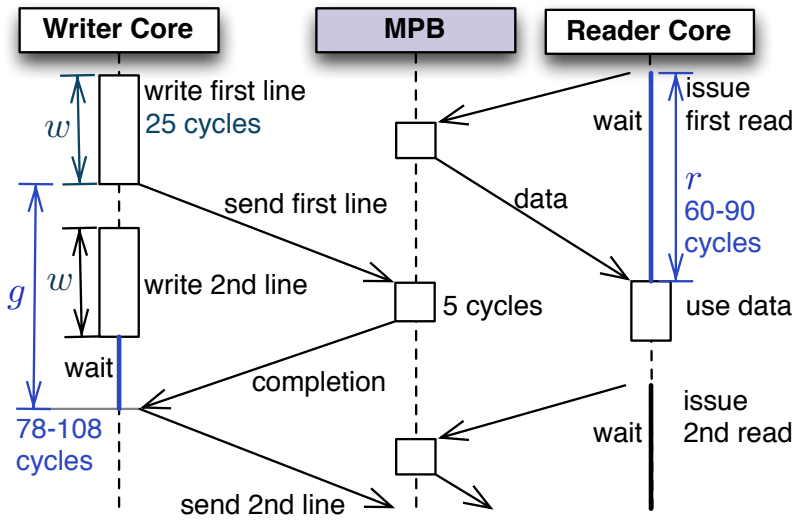
- analysis $\rightsquigarrow$ 70% overhead by notification and probe

$\Rightarrow$ great potential for power savings by better HW support

OUTLINE

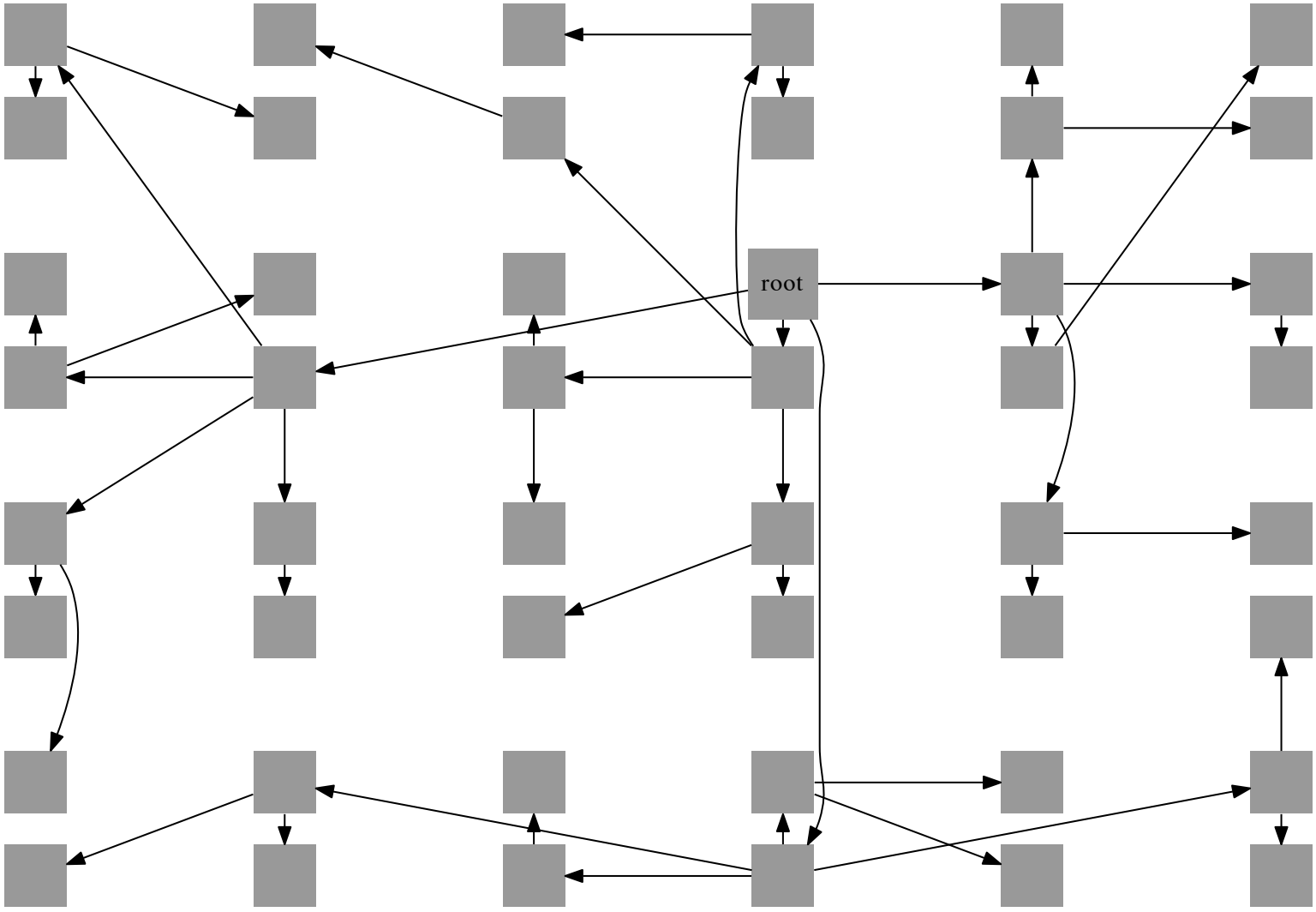
5. Appendix

WGR COST MODEL

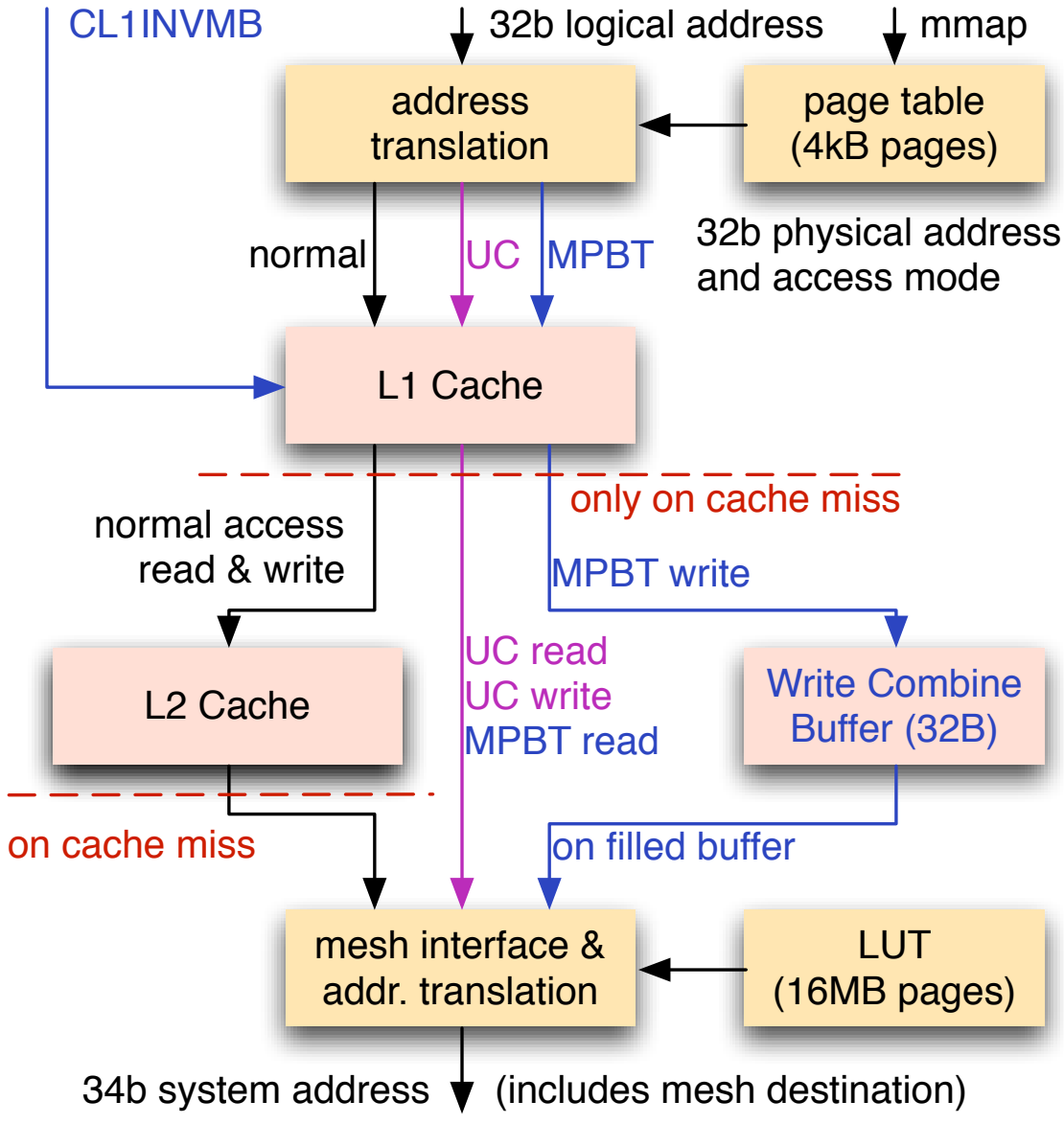


	UC	bypass	MPBT	bypass
w	5	5	25	25
g	51–84	17	78–108	39
r	54–87	19	60–90	21

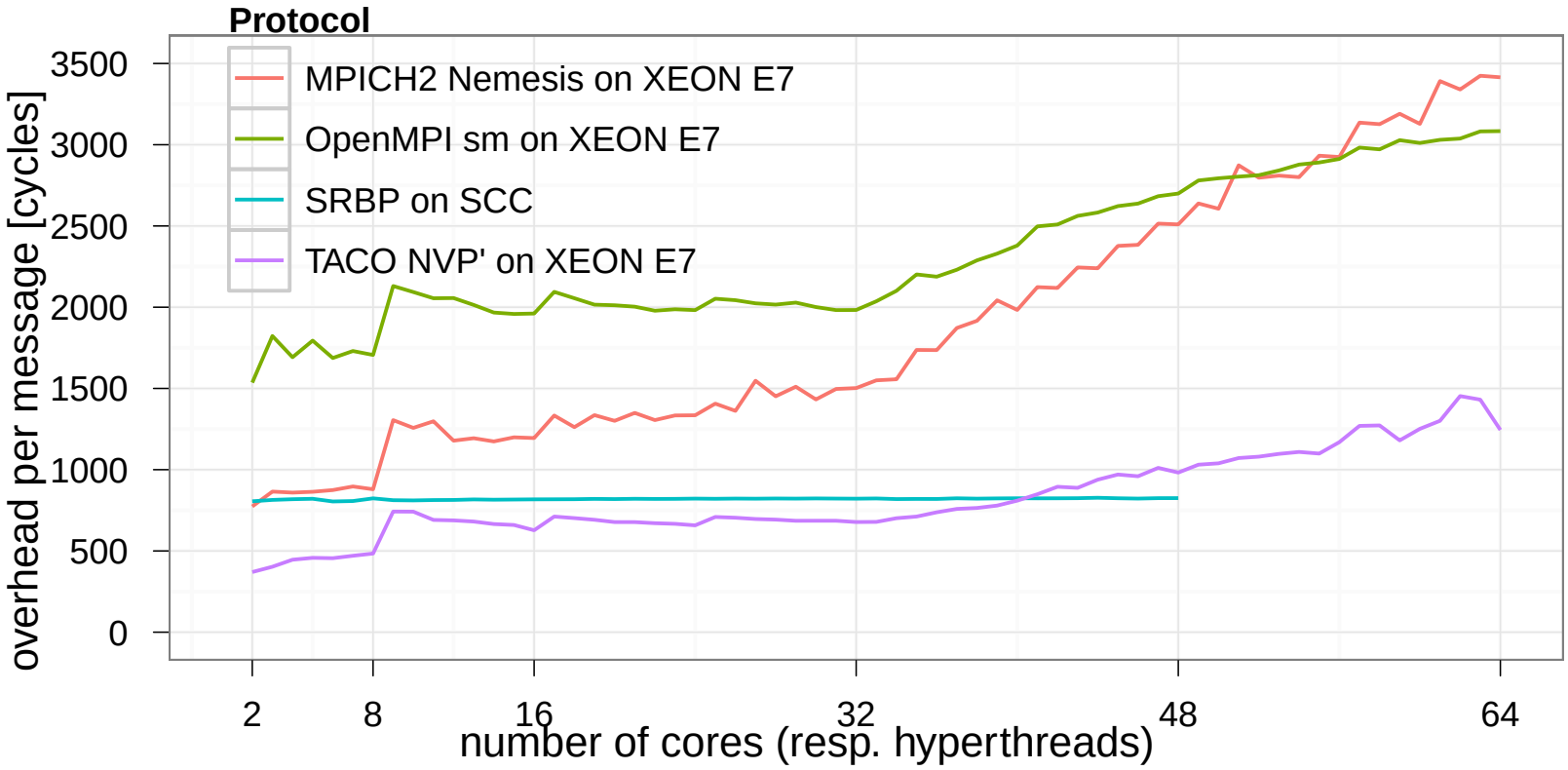
COLLECTIVE OPERATIONS



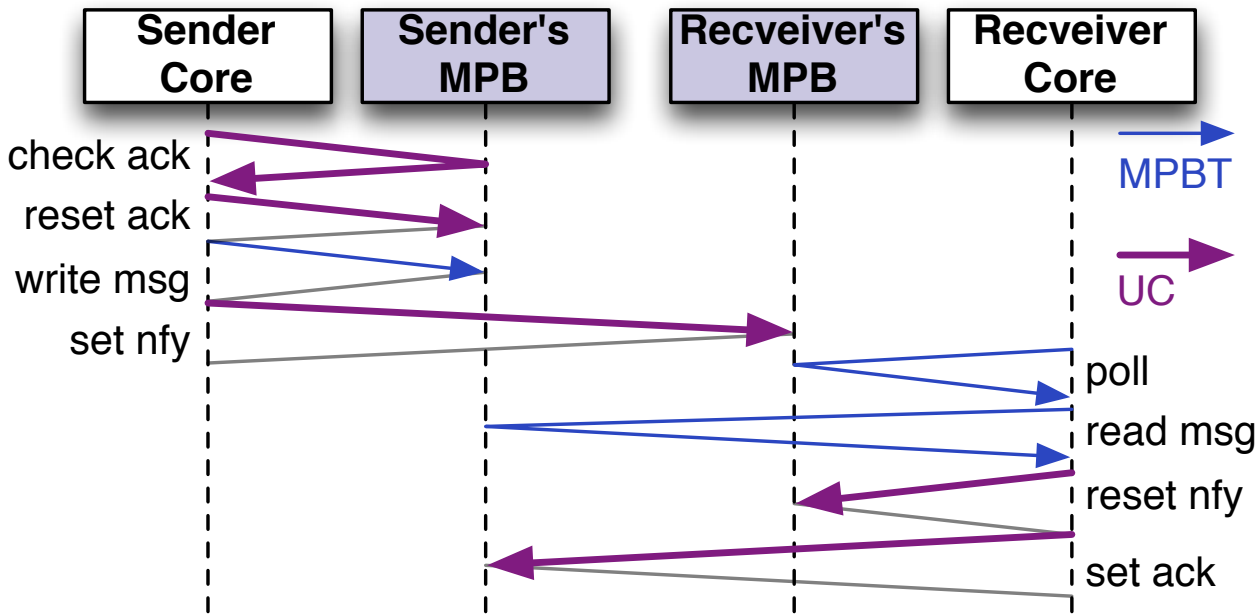
SCC ADDRESS TRANSLATION



PROTOCOL SCALABILITY ON XEON E7



PROTOCOLS: NVP



PROTOCOLS: SRBP