ROHIT ZAMBRE

SCALABLE COMMUNICATION ENDPOINTS FOR MPI+THREADS APPLICATIONS

WHO WILL YOU BE LISTENING TO FOR THE NEXT 40 MINS?

- An intern

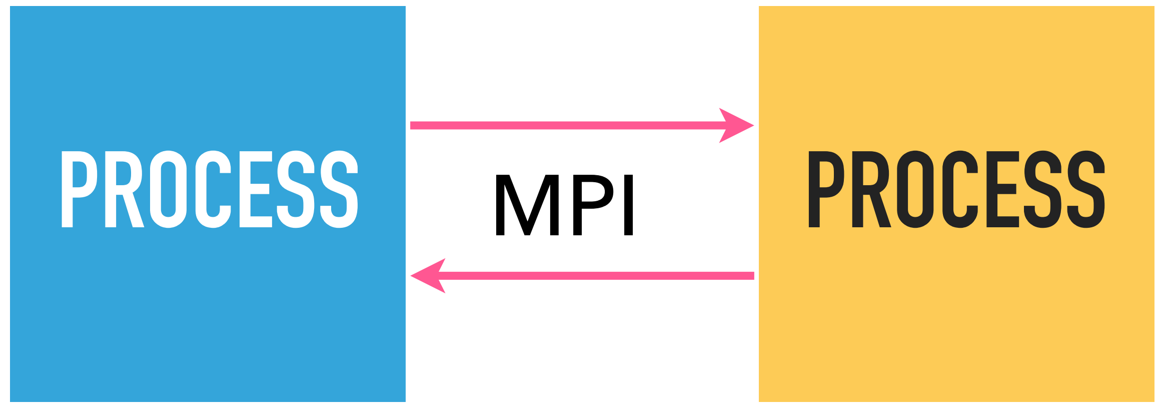
- Ph.D. student @ University of California, Irvine
  - HPC Forge research group
- Visiting student @ Argonne National Laboratory
  - Programming Models and Runtime Systems group
- This talk: research on multiple endpoints for MPI

A STANDARD FOR EXPLICIT COMMUNICATION BETWEEN PROCESSES RUNNING IN PARALLEL.

Message Passing Interface (MPI)

MPI AND PARALLELISM

- Parallelism from multiple processes. MPI for coordinating.



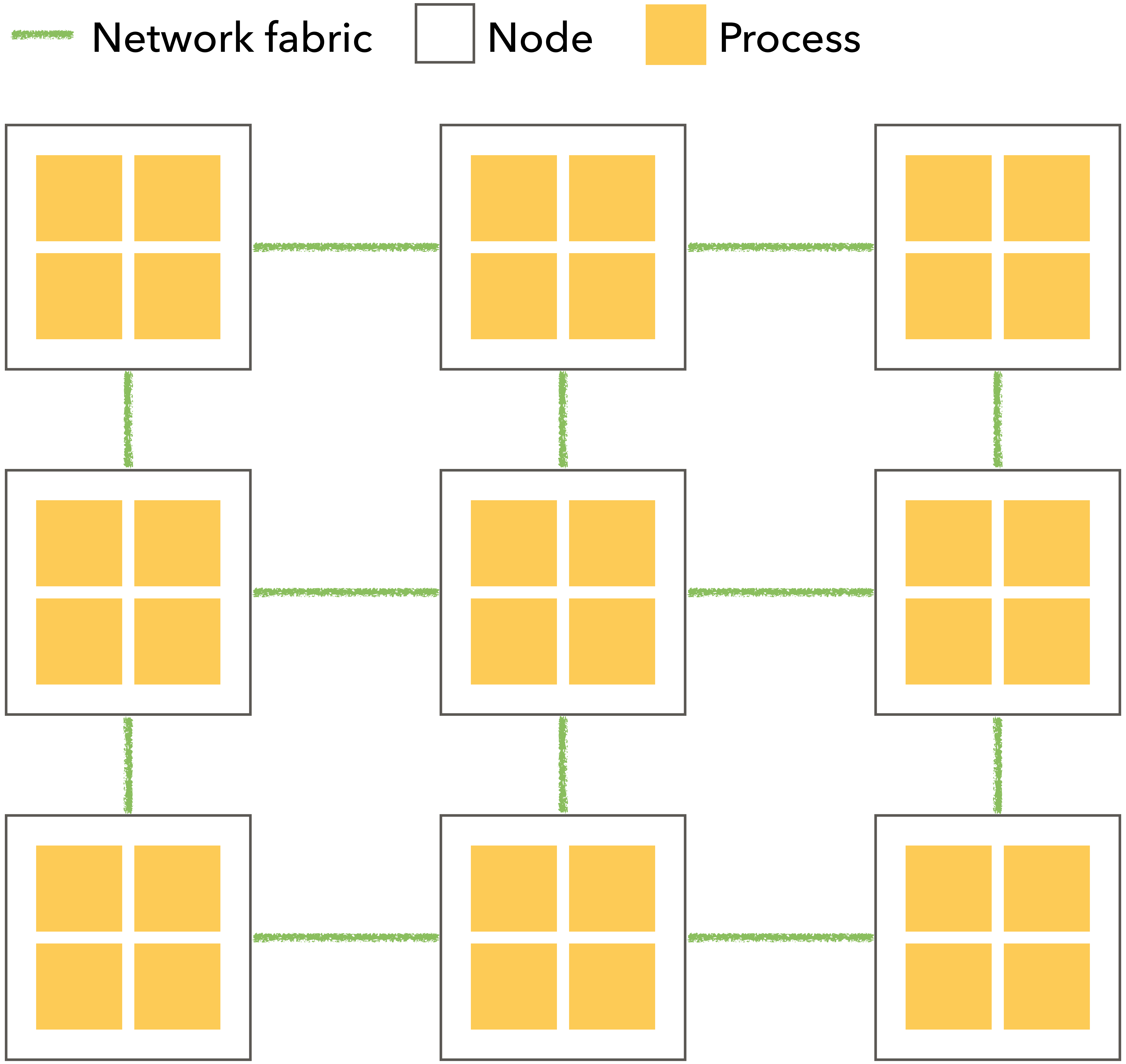
- MPI not an implementation. MPI forum (http://www.mpi-forum.org)
- MPICH, OpenMPI, MVAPICH, Intel MPI, Microsoft MPI, etc.
- Widely used: scientific simulations, modeling, USPS, etc.

TRADITIONAL MPI: MPI EVERYWHERE

App ignores location of processes

Leads to static split of on-node resources

Cores scaling a lot faster than TLB, network resources, etc.

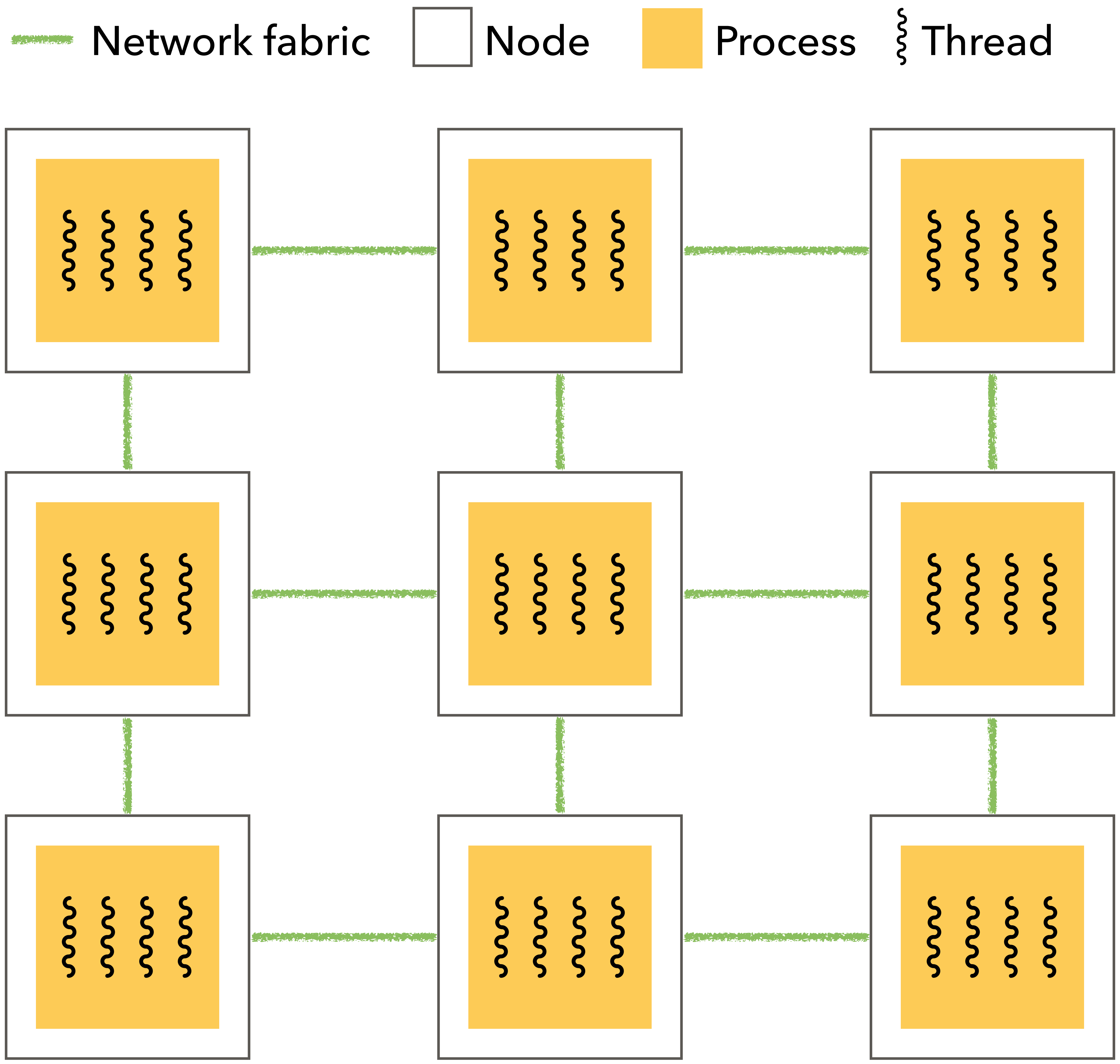


HYBRID MPI: MPI+THREADS

e.g. MPI+OpenMP

All threads access all on-node resources

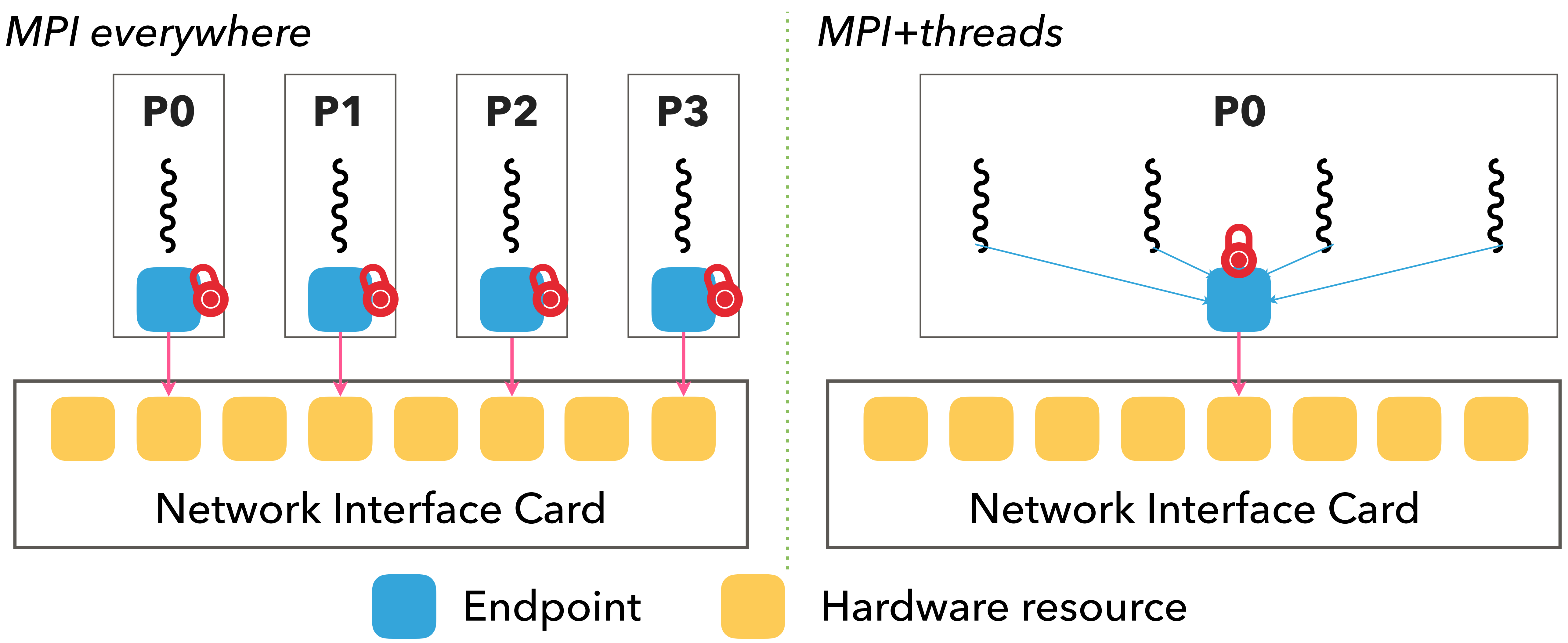
Handy compiler support for MPI user

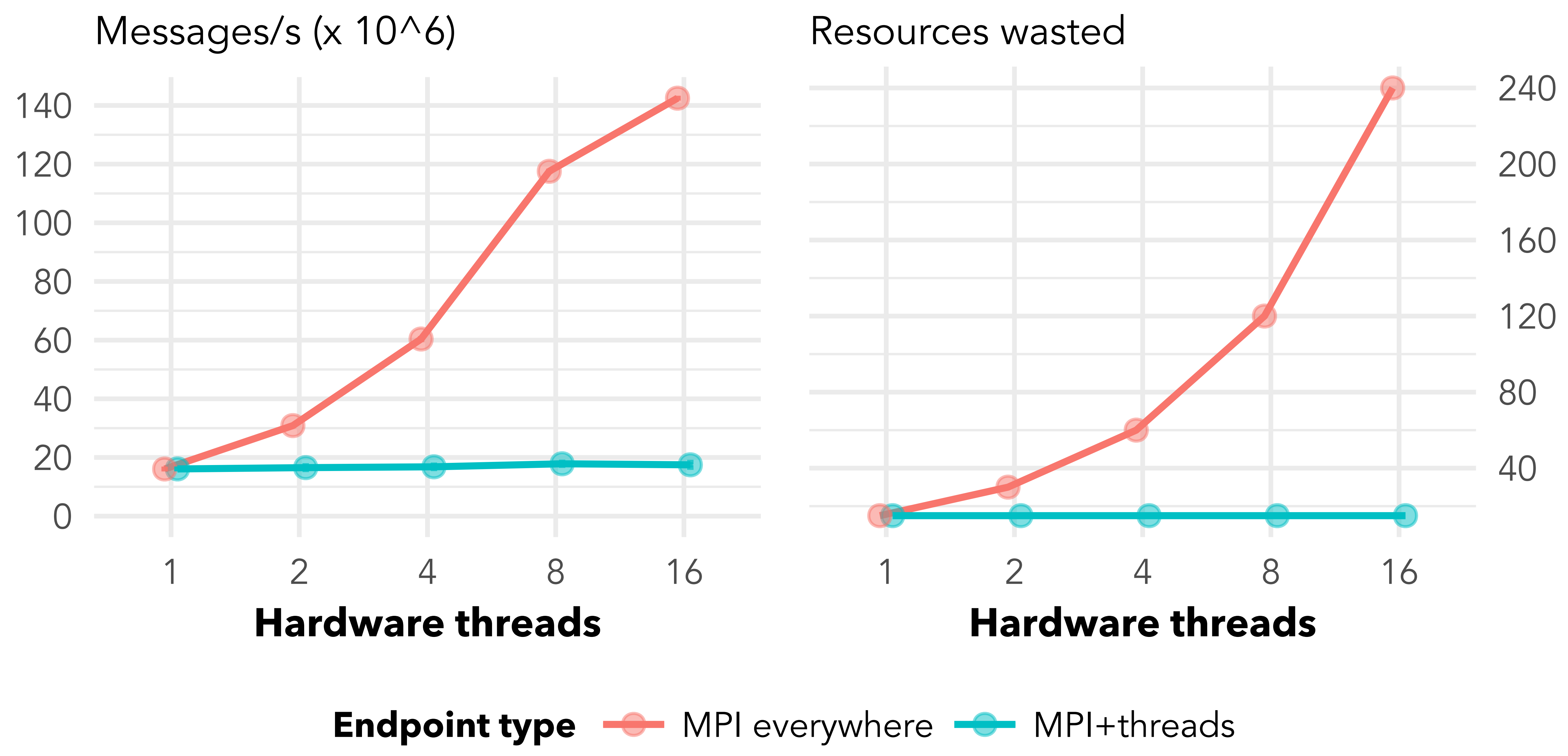


STATE OF THE ART

MPI everywhere achieves maximum possible throughput but wastes 93.75% of hardware resources

MPI+threads achieves maximum resource efficiency but performs up to 9x worse



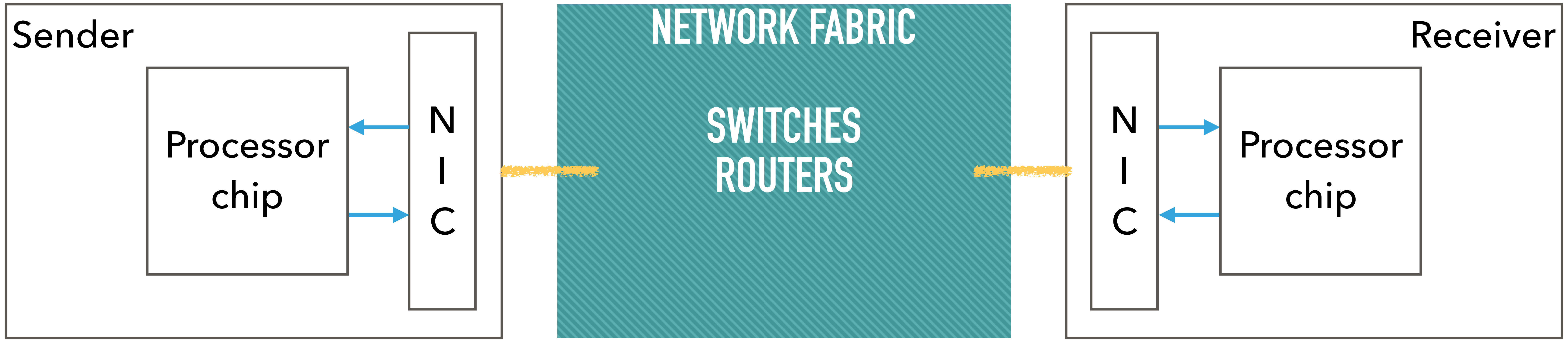


WHAT LEVEL OF RESOURCE SHARING IS IDEAL?

WHAT LEVEL OF RESOURCE SHARING IS IDEAL?

It depends

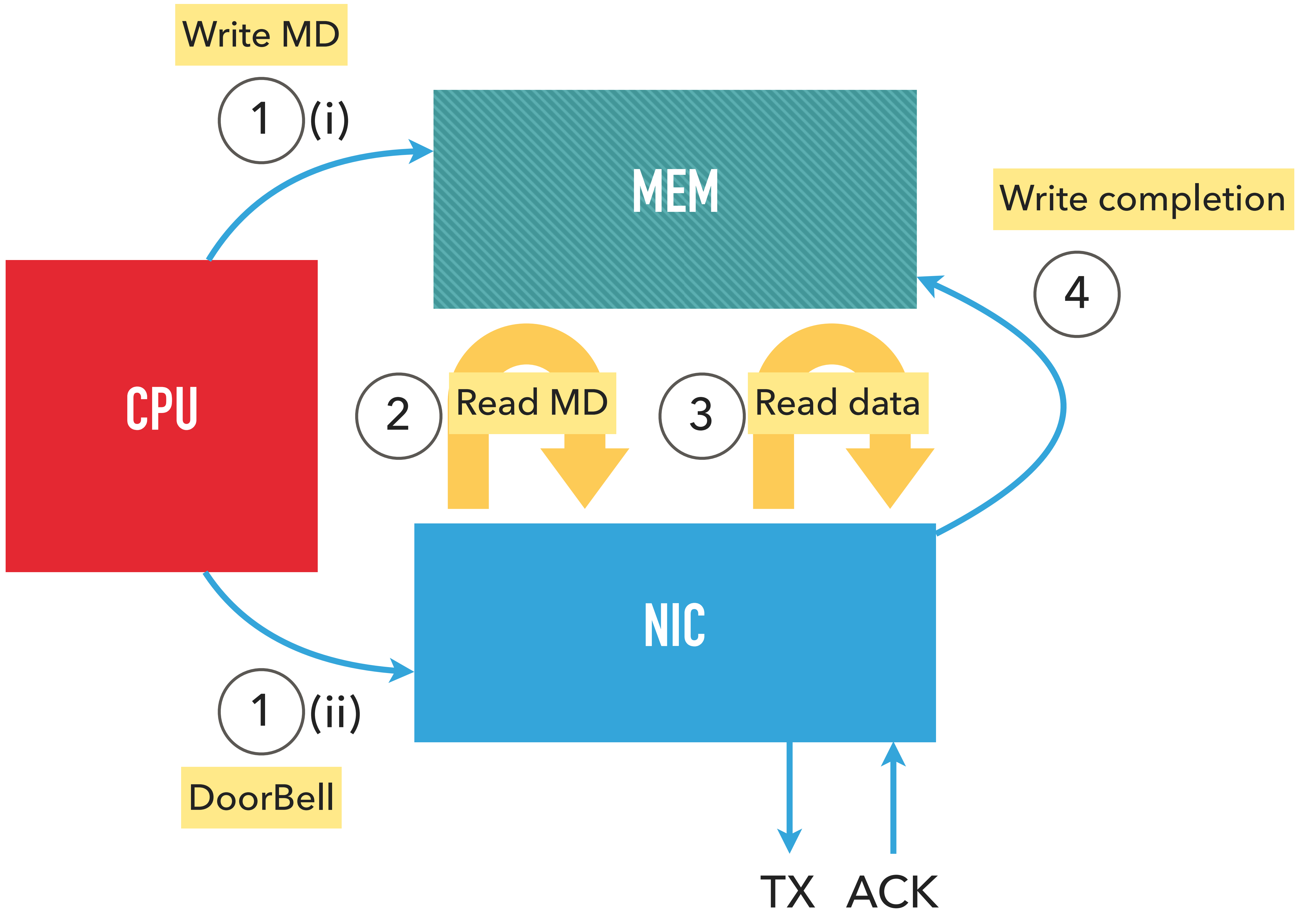
INTER-NODE COMMUNICATION



- The Network Interface Card (NIC) is the node's communication portal.
- The software (CPU) coordinates with the hardware (NIC) to *initiate* a transfer and confirm its *completion*.

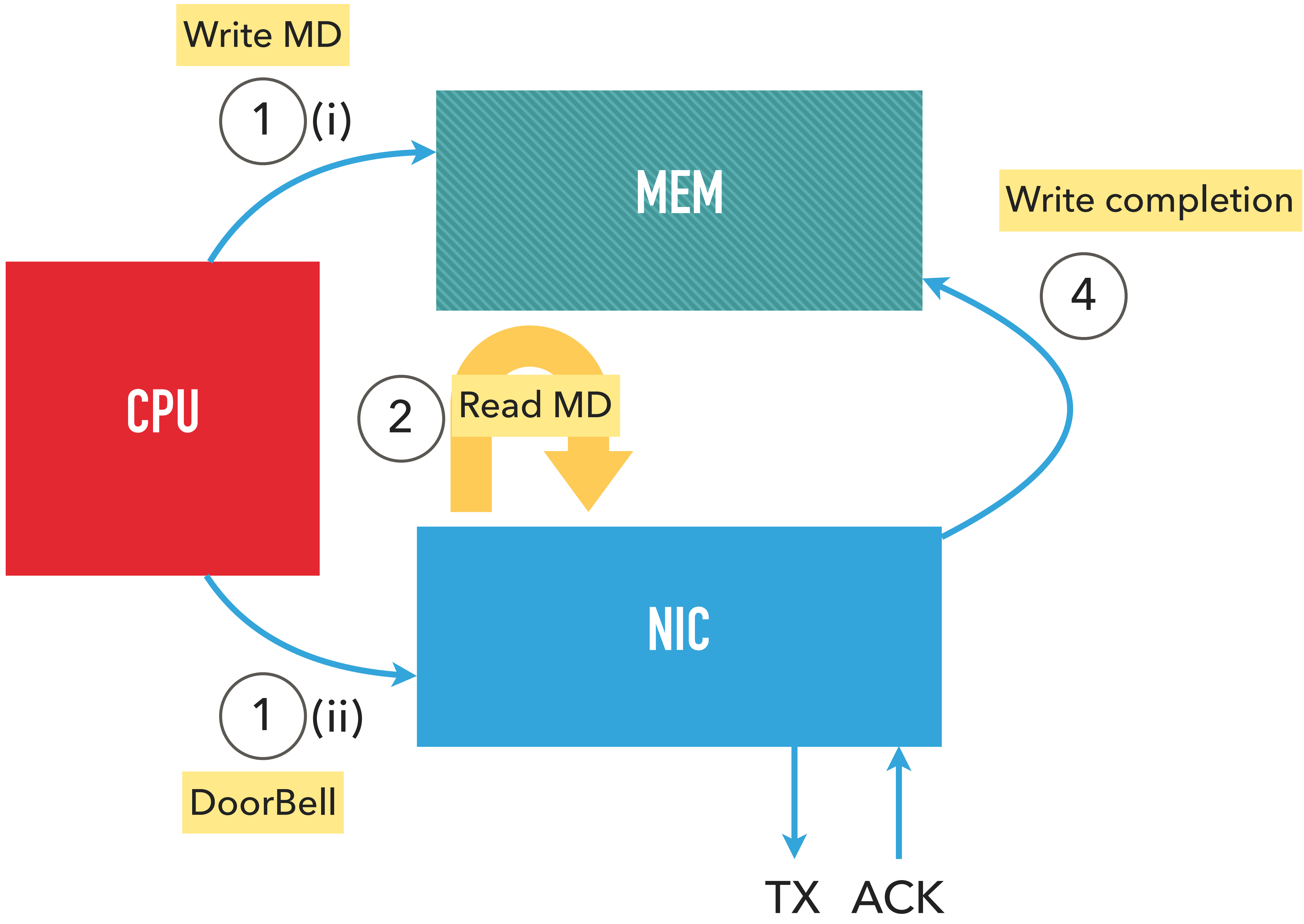
CPU–NIC INTERACTION: OVERVIEW

1. (i) Write message descriptor to MEM
   (ii) DoorBell (atomic 8-byte MMIO-write to NIC)
2. DMA-read to get message descriptor
3. DMA-read to get payload (data)
4. DMA-write completion



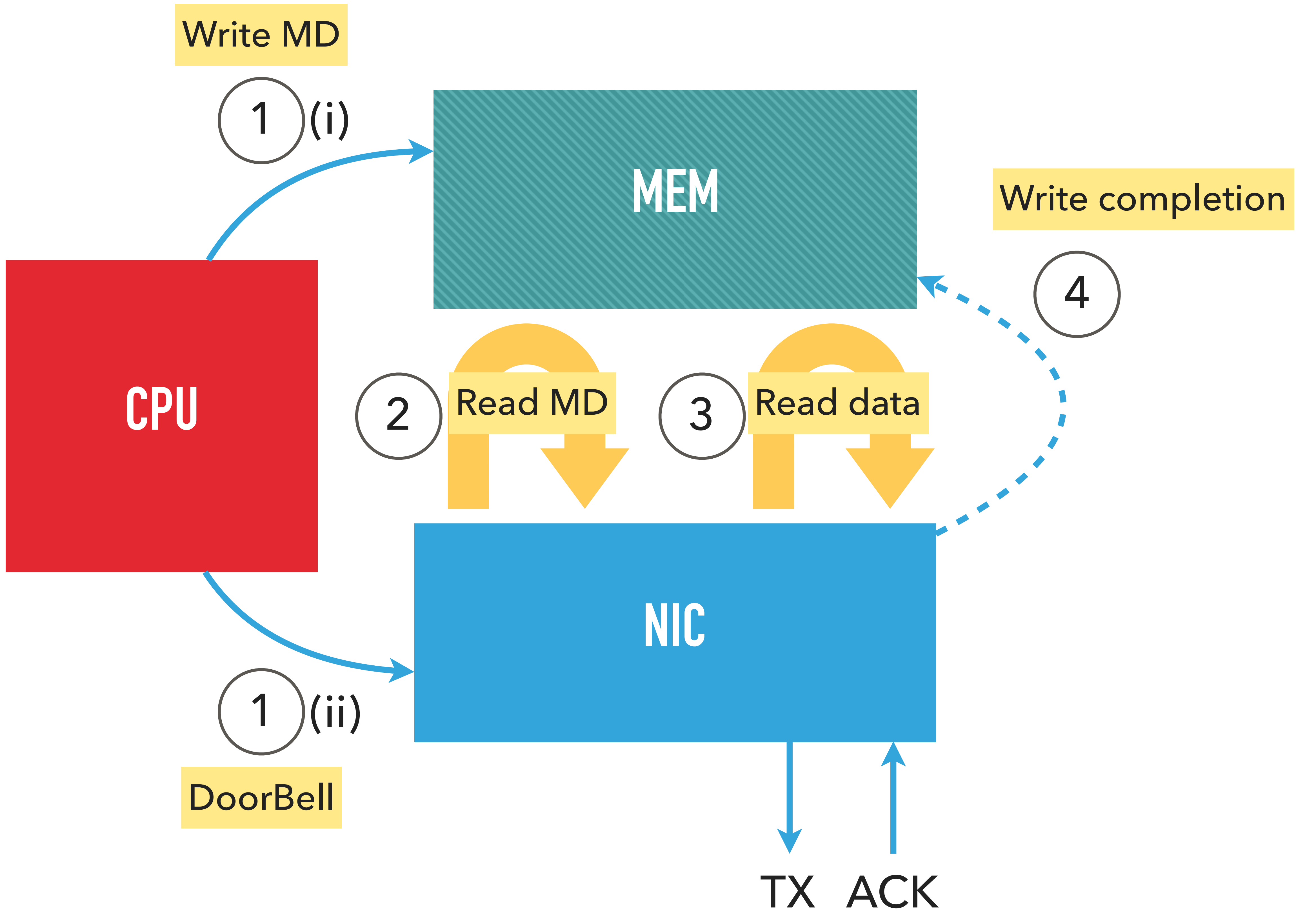
CPU-NIC INTERACTION FEATURE: INLINING

- Payload part of message descriptor.
  - CPU reads the data instead of NIC
- Removes (3)



CPU-NIC INTERACTION FEATURE: UNSIGNALED COMPLETIONS

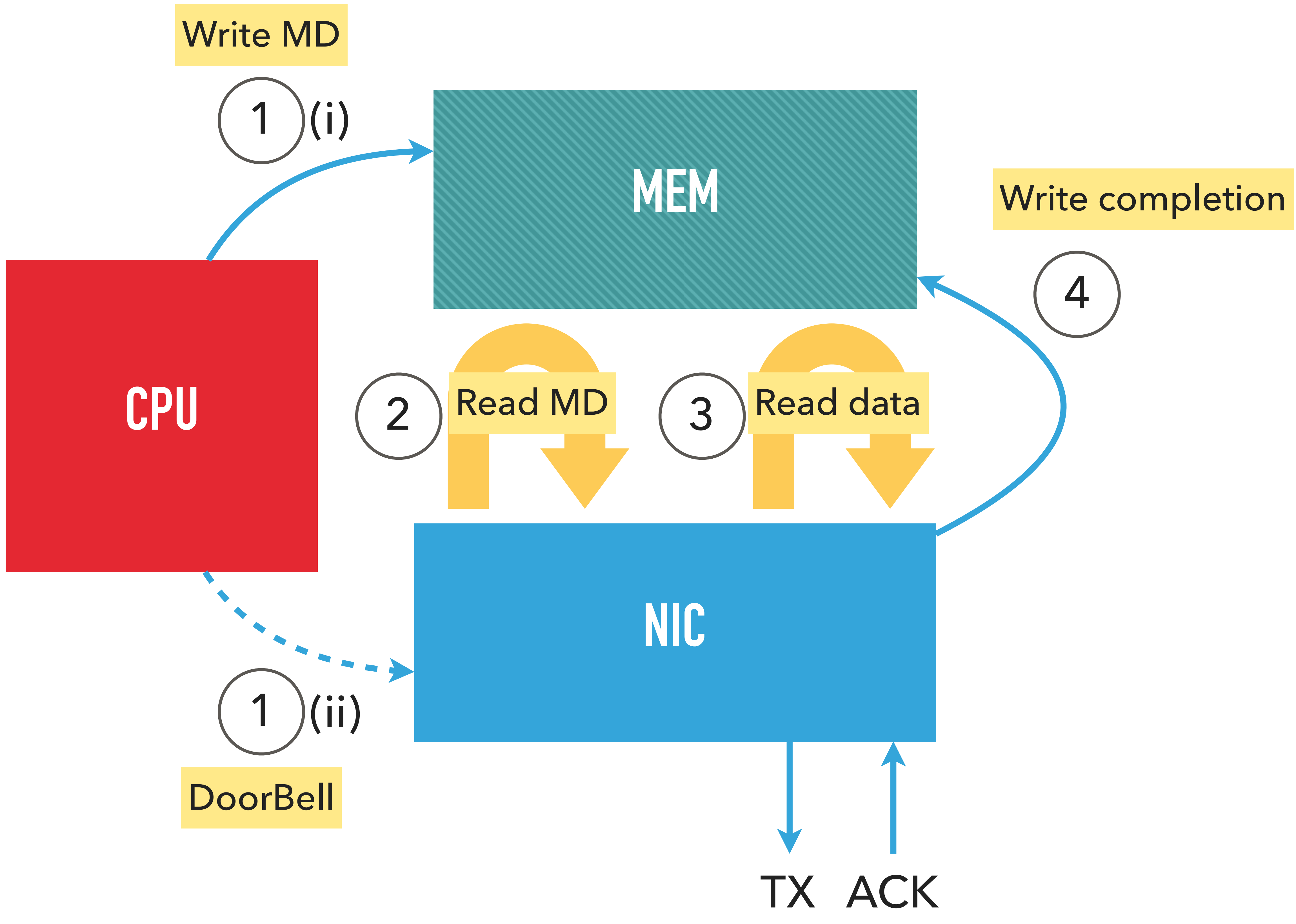
- We can choose whether or not to generate a completion for each message. If not, then the NIC will not DMA-write a completion for a successfully executed message request.
- Reduces the number of (4)



CPU-NIC INTERACTION FEATURE: POSTLIST

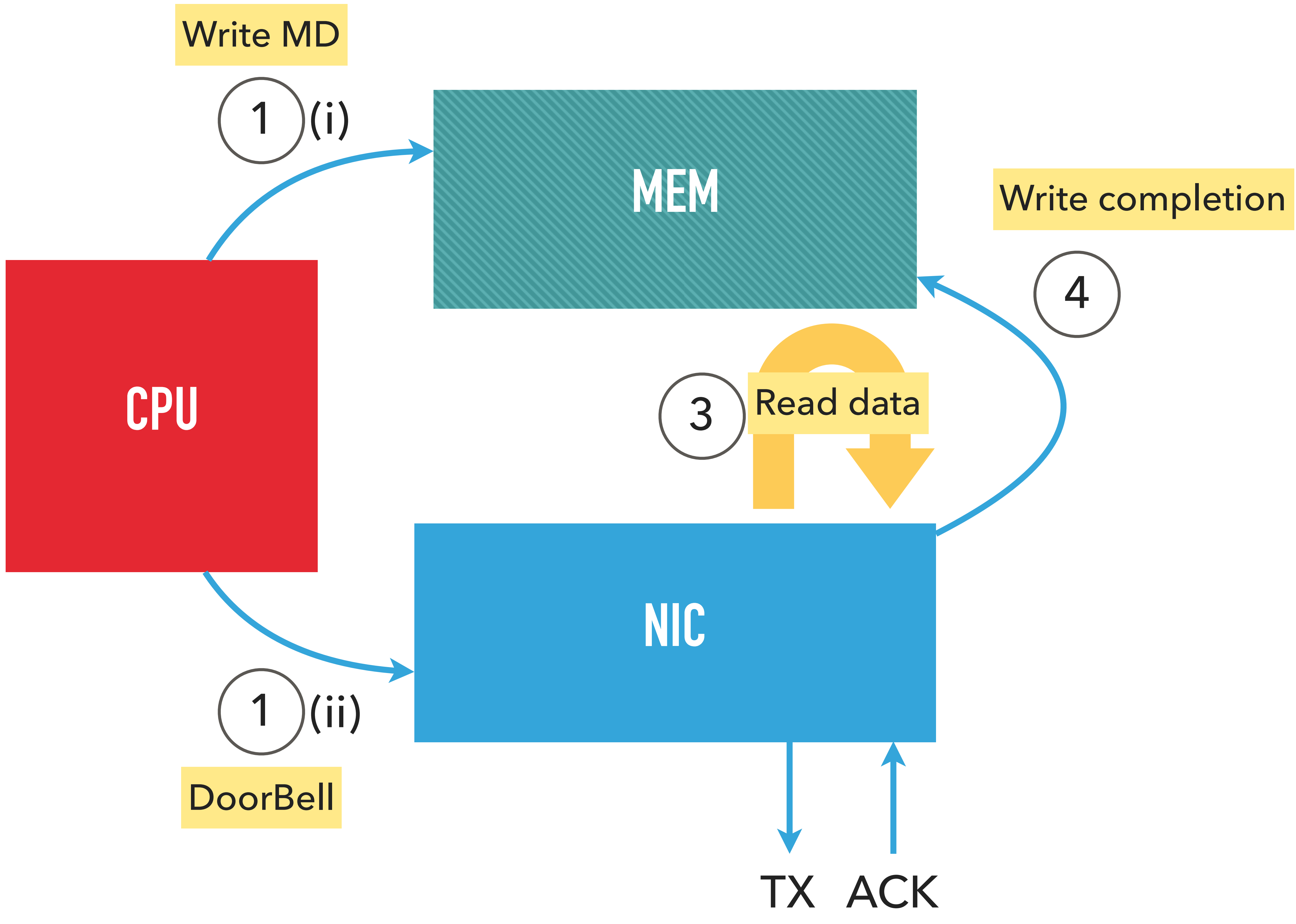
- Instead of sending one message every DoorBell, we can send a linked list of message descriptors with 1 DoorBell

- Reduces number of (1)(ii)



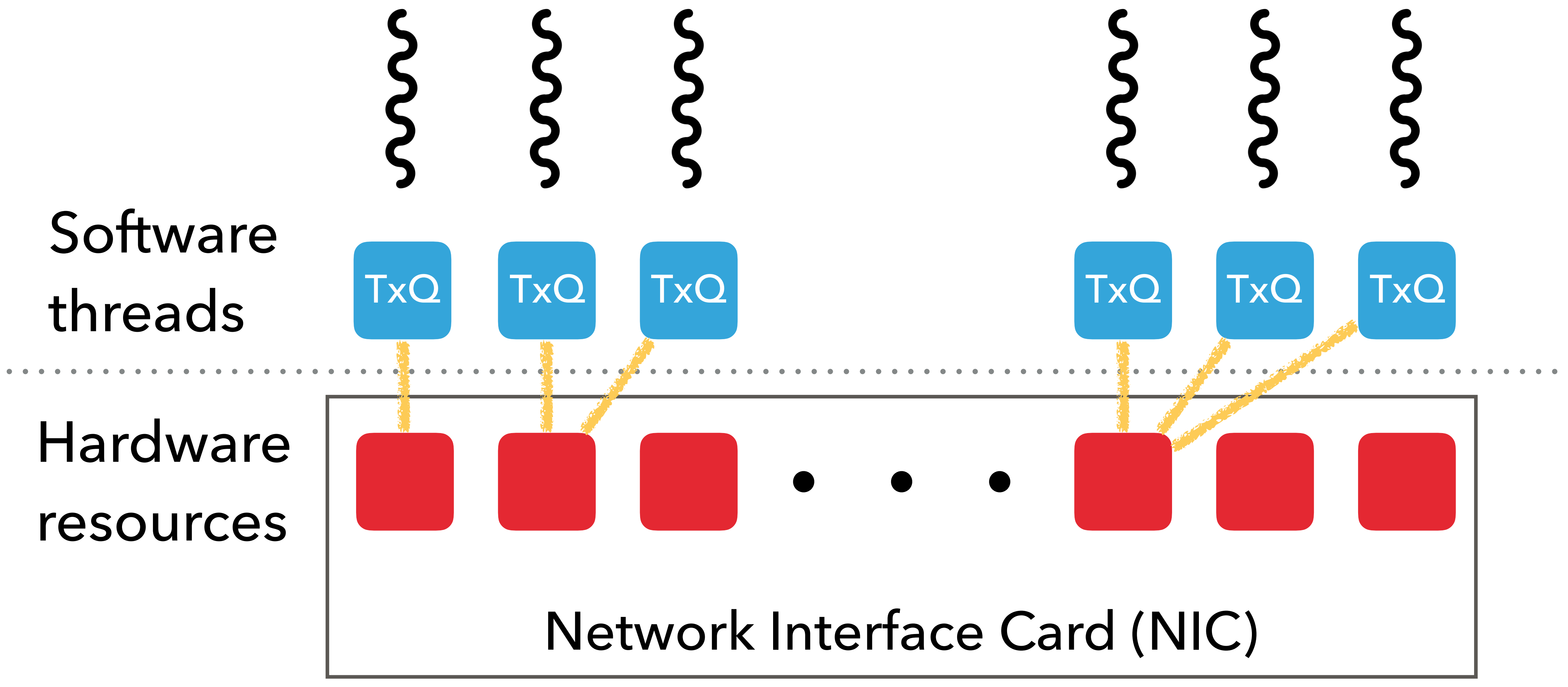
CPU-NIC INTERACTION FEATURE: BLUEFLAME

- Mellanox terminology for Programmed I/O (PIO)
- Send message descriptor as a part of DoorBell
- If used with inlining, will also cut off (3)
- Meaningless if used with Postlist



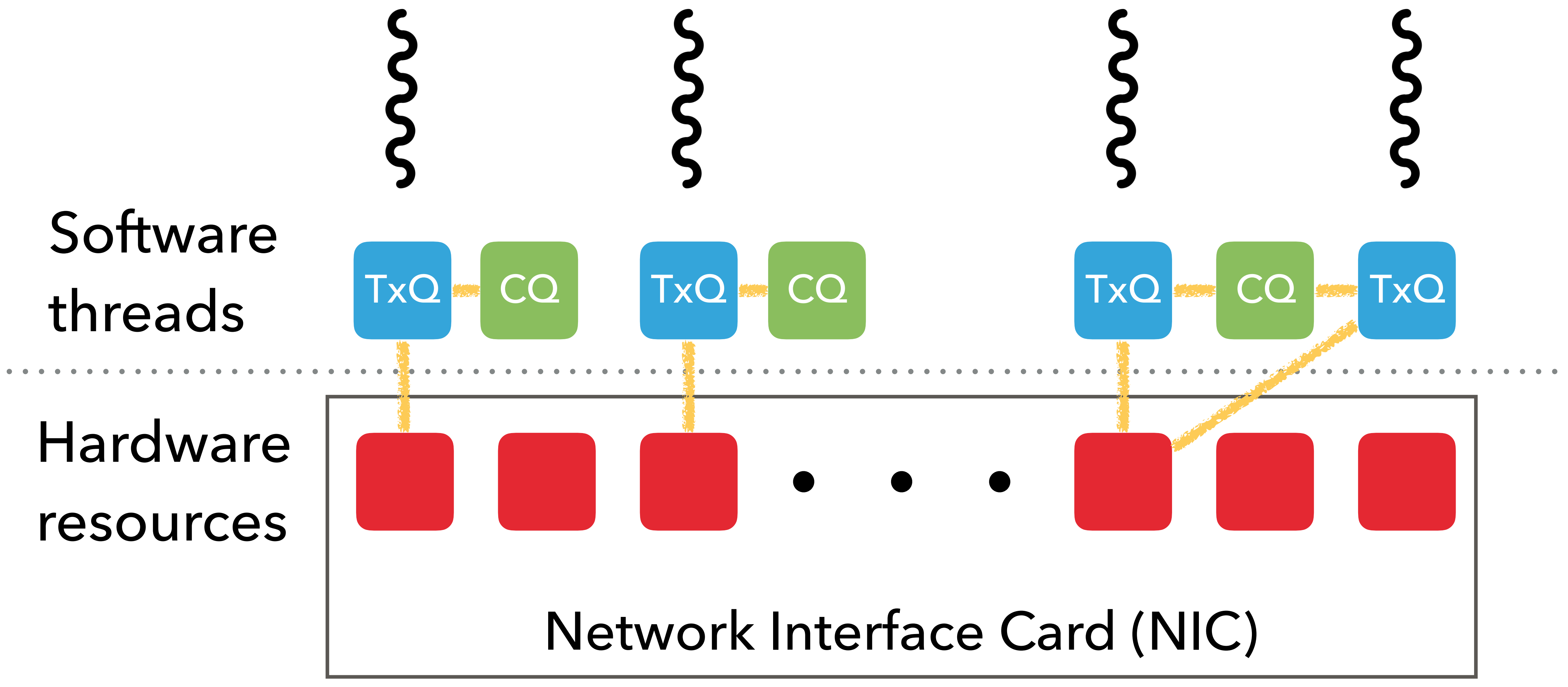
INITIATION INTERFACE

- Software transmit queue (TxQ) is a FIFO queue of messages.
- High-performance NICs feature multiple hardware resources (for scalability).
- The mapping between the TxQ and the hardware resource depends on the interconnect's driver. Hardware resource is `mmap()`ed.
- The thread posts messages to the endpoint, driver MMIO-writes to hardware resource to signal the NIC to send the message

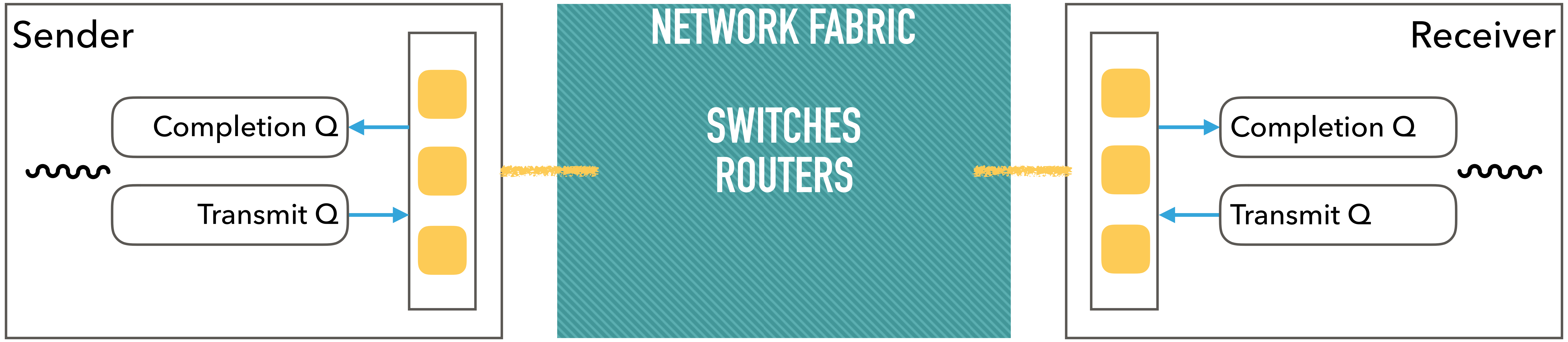


COMPLETION INTERFACE

- A FIFO completion queue (CQ) contains completions that indicate a successful, or unsuccessful completion of a message request.
- Mapping of CQs to the TxQ depends on software design, not the interconnect's driver.
- The NIC DMA-writes the completion, signaling the thread about the status of its message request.

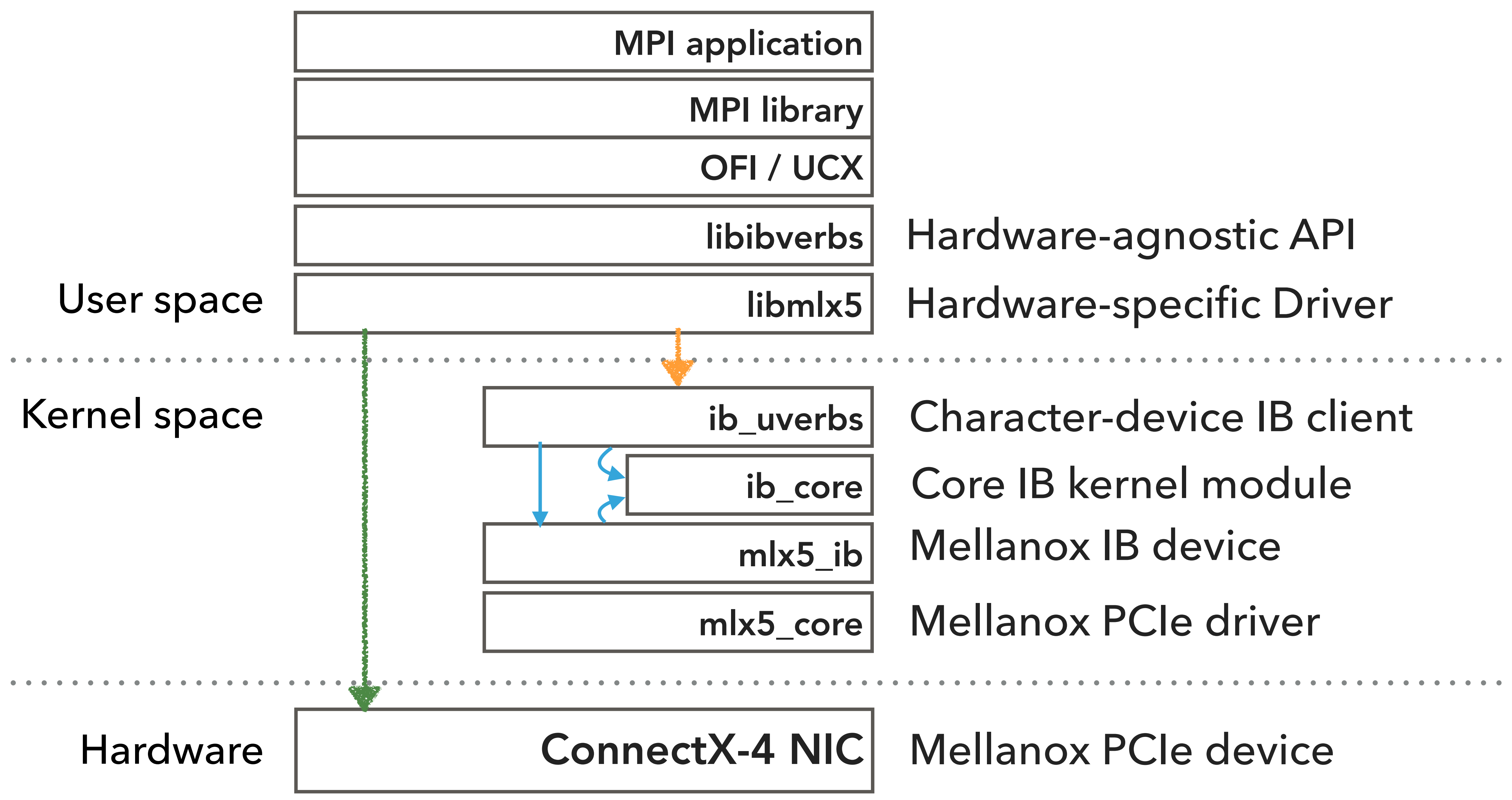


COMMUNICATION RESOURCES: TXQ + CQ + HARDWARE RESOURCES



- Traditionally, the TxQs and CQs are exposed only to the kernel. For example, send() or recv() in TCP/IP's `sockets` interface.
- High-performance interconnects expose TxQs and CQs to the user-space to prevent system call overheads (kernel by-pass).
- Impact **memory usage** and **hardware resource** consumption

HIGH-PERFORMANCE INTERCONNECT OF CHOICE: MELLANOX INFINIBAND



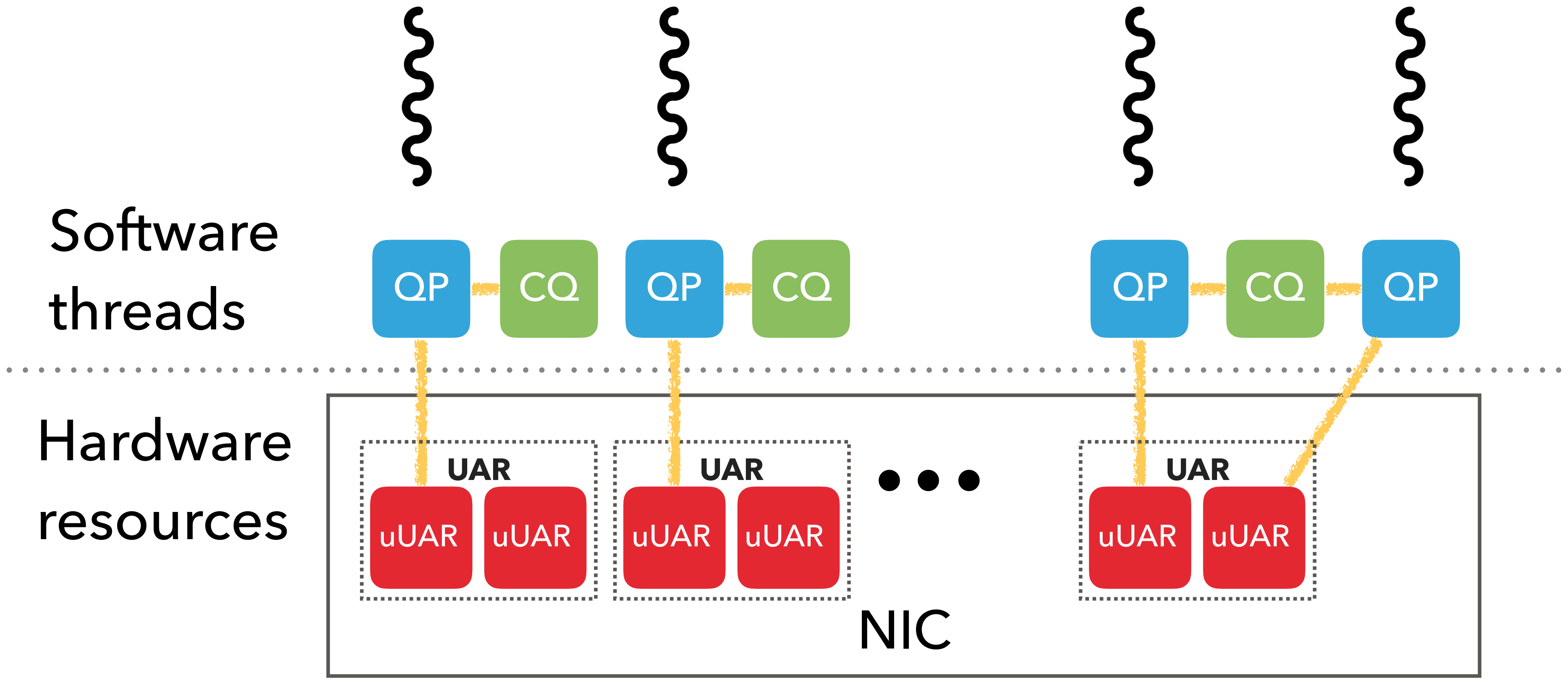
- Slow-path: user code interacts with kernel-code by `write()` on `/dev/infiniband/uverbs0`. Slow-path contains resource setup operations.
- Fast-path: user code writes to NIC's hardware resources

INFINIBAND COMMUNICATION RESOURCES

Transmit queue: Queue Pair (QP)

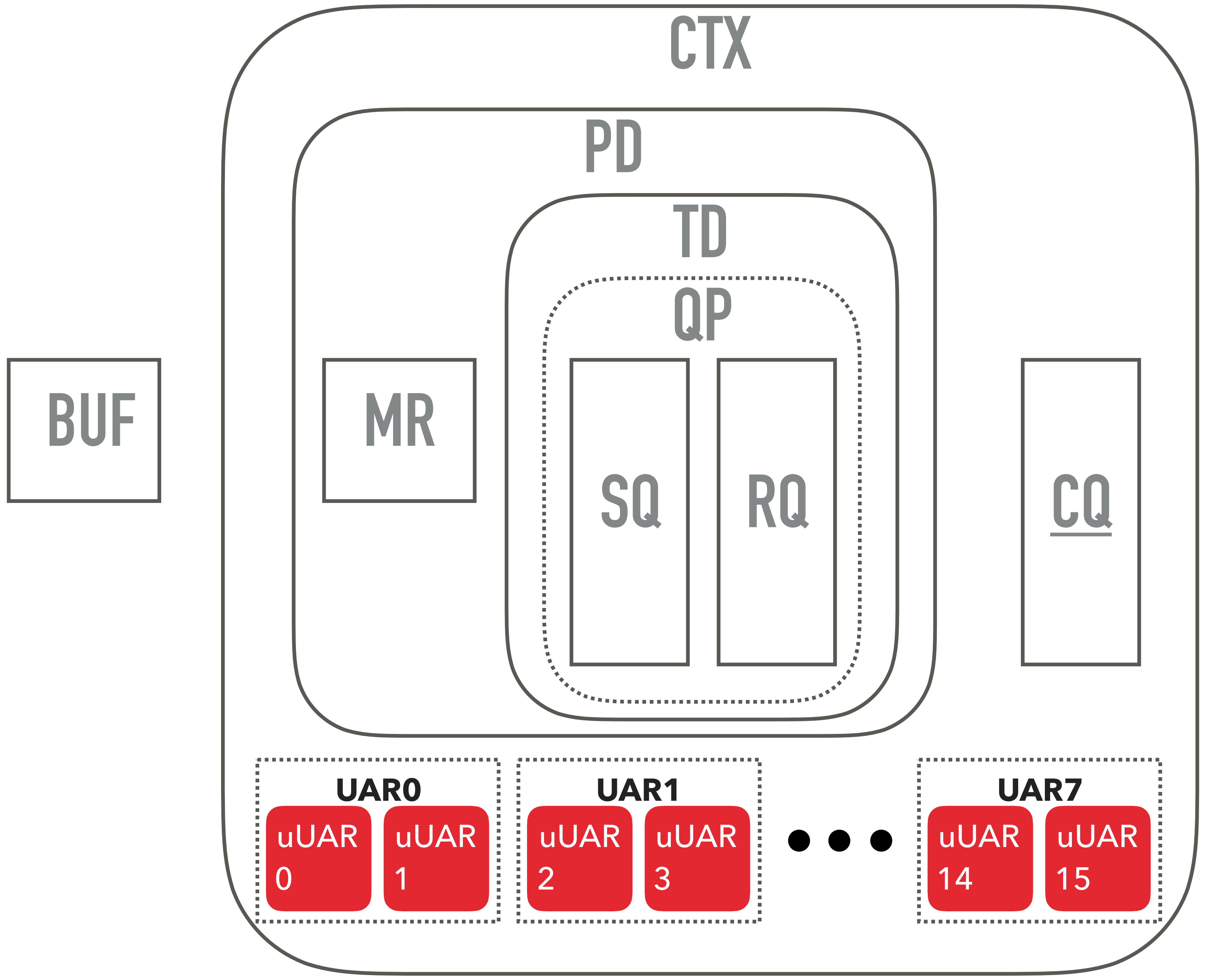
Completion queue: Completion Queue (CQ)

Hardware resources: User Access Region (UAR) + micro UARs (uUARs)



INFINIBAND API

- 1 **Device Context** (`ibv_open_device`): 8 UARs; 16 uUARs
- 1 Protection Domain (`ibv_alloc_pd`)
- 1 Memory Region (`ibv_create_mr`)
- 1 **Completion Queue** (`ibv_create_cq`)
- Memory buffer (`malloc`)
- 1 **Queue Pair** (`ibv_create_qp`)
- 1 **Thread Domain** (`ibv_alloc_td`): 1 UAR; 2 uUARs



NAIVE ENDPOINTS

- To achieve maximum possible throughput, create *n* contexts with TD-assigned-QPs for *n* threads
  - 18 uUARs per thread: 8 static UARs + 1 dynamic UAR
- This naive approach impacts
  - Memory usage
  - Hardware resource usage

MEMORY USAGE

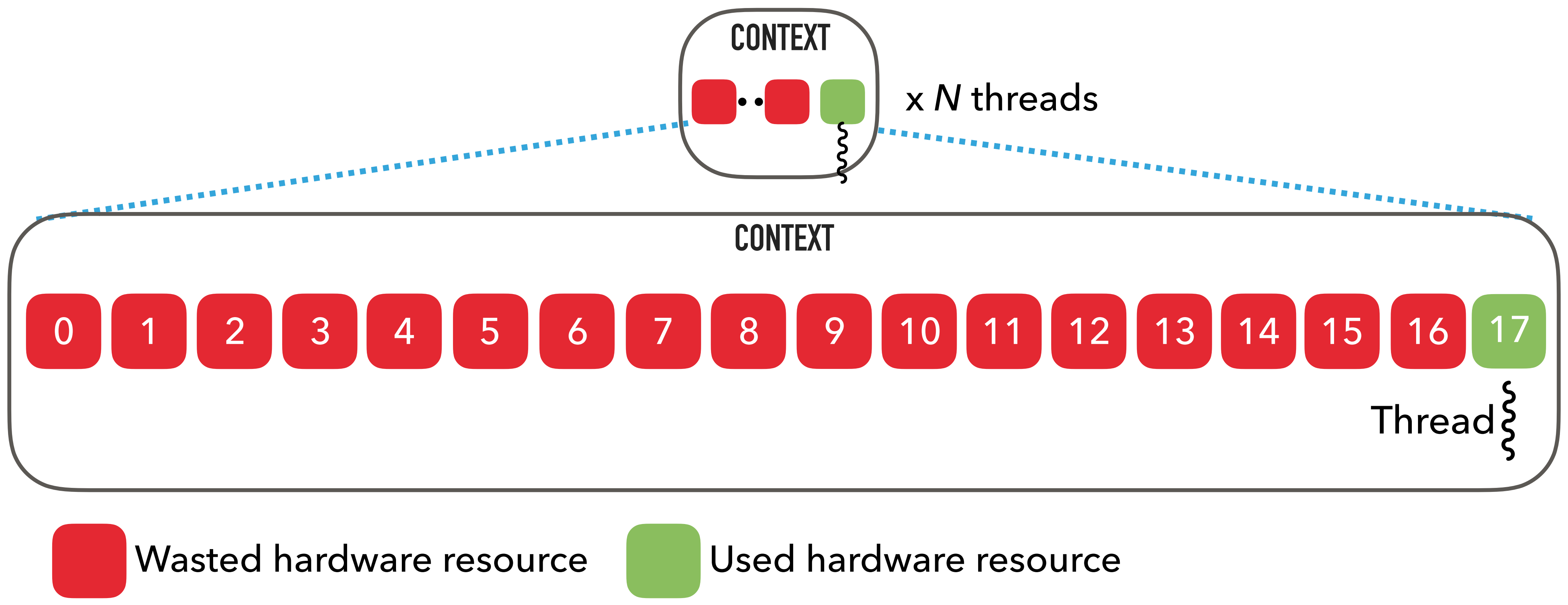
- Each QP and CQ occupy memory with their circular buffers (FIFO queue)
- CTX uses 74% of the memory required for 1 endpoint
- Creating 16 endpoints, for example, with the naive approach will use 5.15 MB.
- Not of immediate concern since the memory on nodes of clusters and supercomputers is in the order of GB.

TABLE I
BYTES USED BY VERBS RESOURCES

CTXs	PDs	MRs	QPs	CQs	Total
256K	144	144	80K	9K	345K

HARDWARE RESOURCE USAGE

- Total number of uUARs on the ConnectX-4 is 16,336.
    - Translates to a max of 907 CTXs since each CTX allocates 18 uUARs.
- Naive approach translates to only 1 uUAR being utilized every 18 uUARs.
    - Translates to 94% resource wastage.
    - Need a 2nd NIC only after using 6% on the 1st.
- Could've reduced cost and power usage of the device.

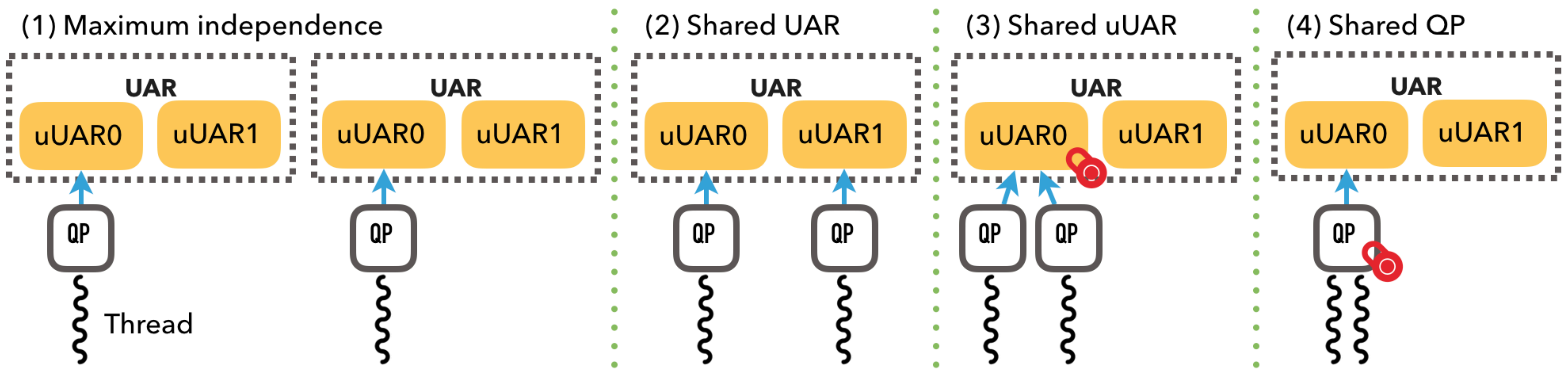


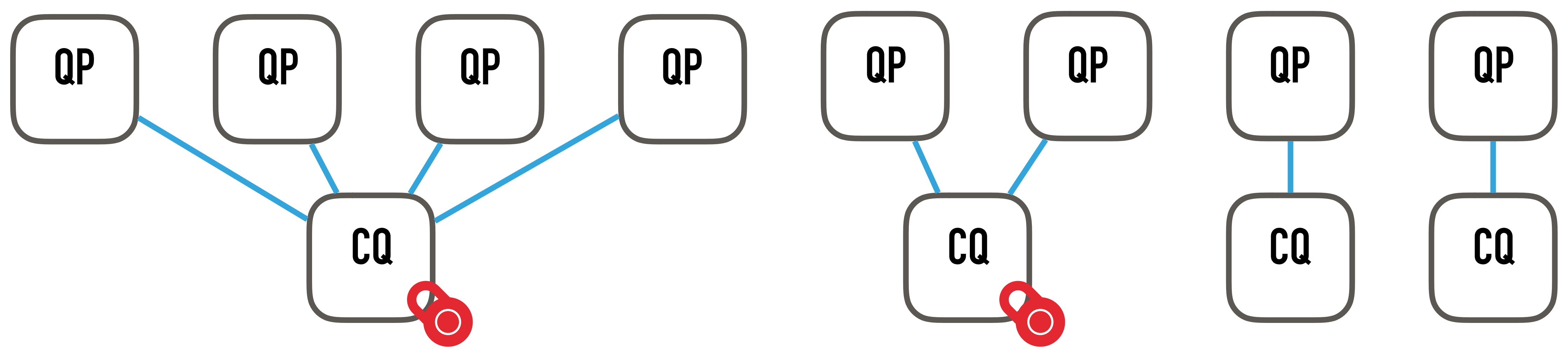
COMMUNICATION RESOURCE SHARING

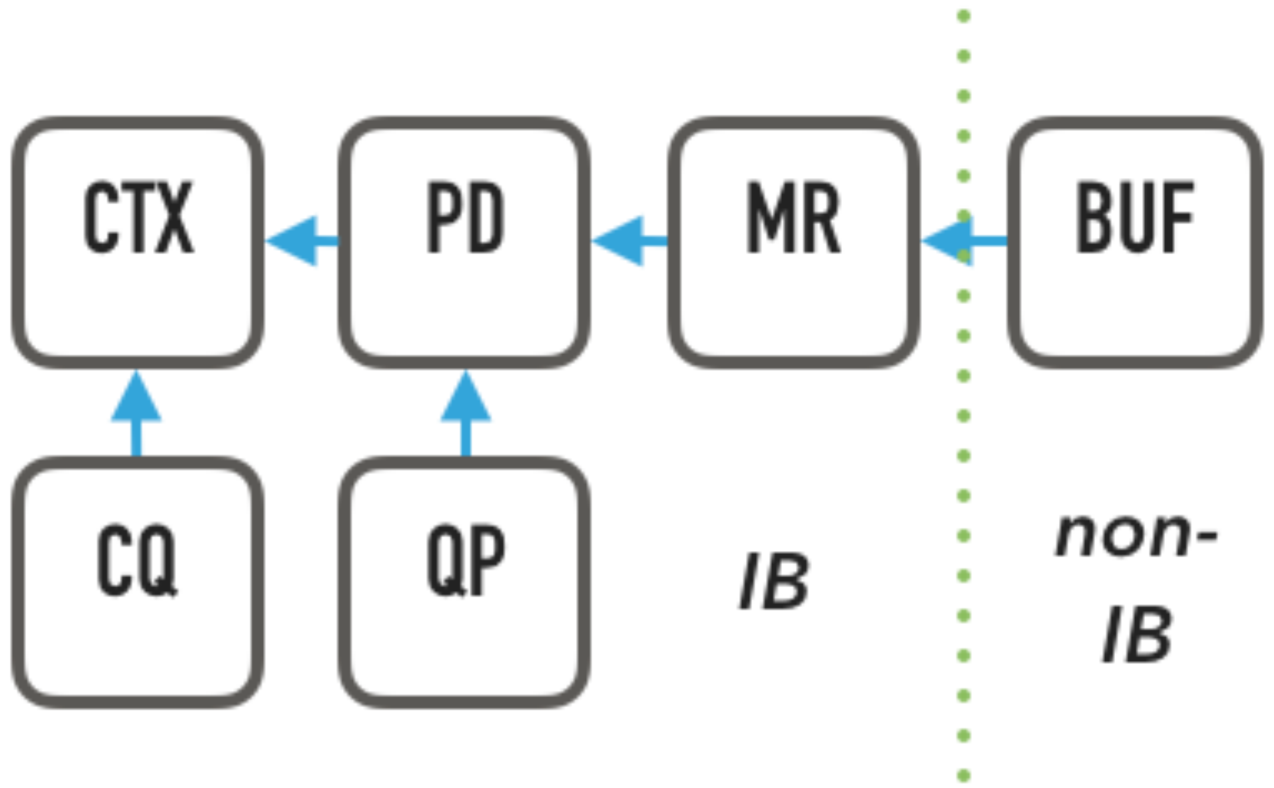
Analytically, 4 ways for initiation interface

Arbitrary number of ways for completion interface

Sharing analysis based on IB's resource hierarchy

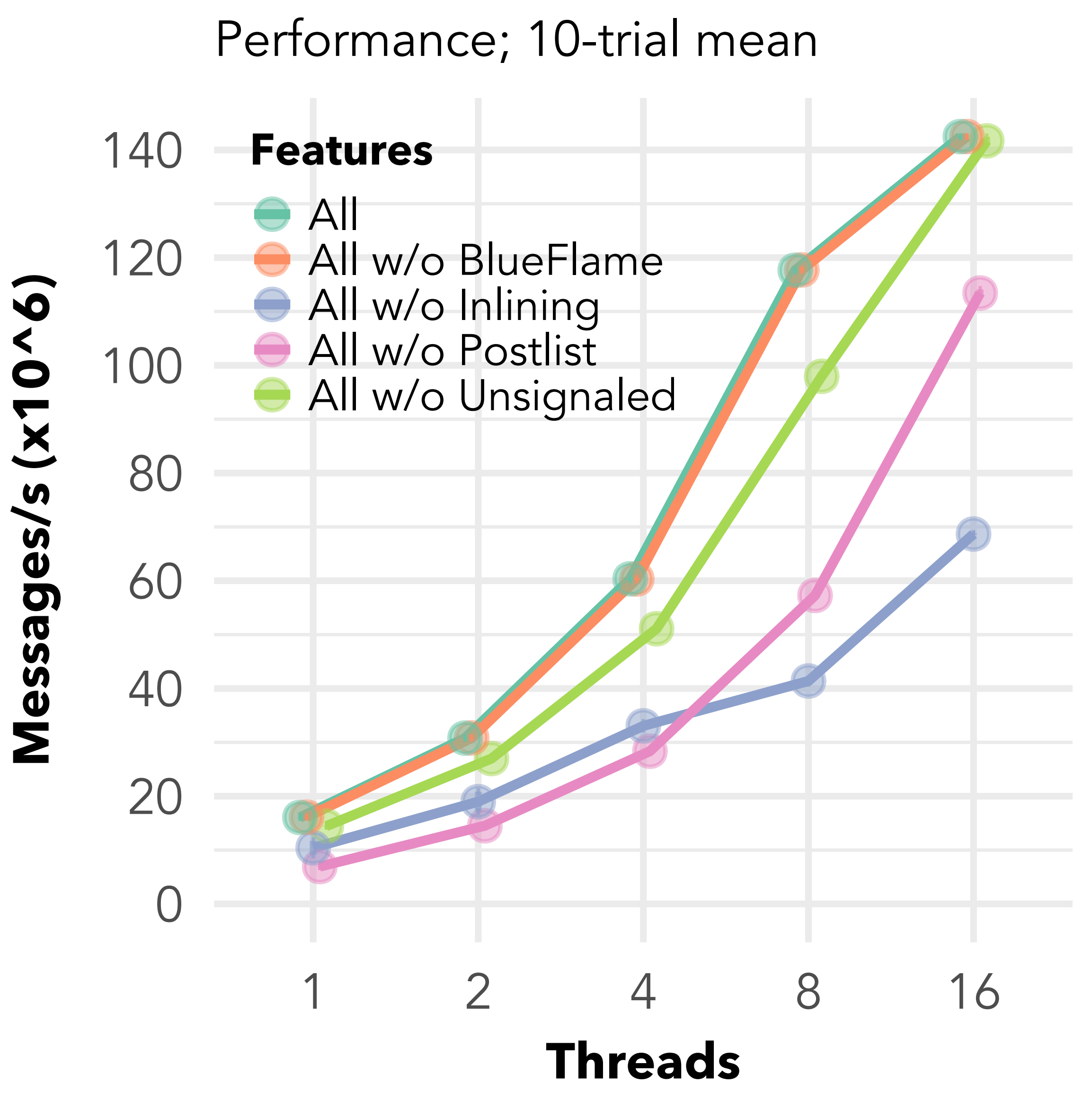


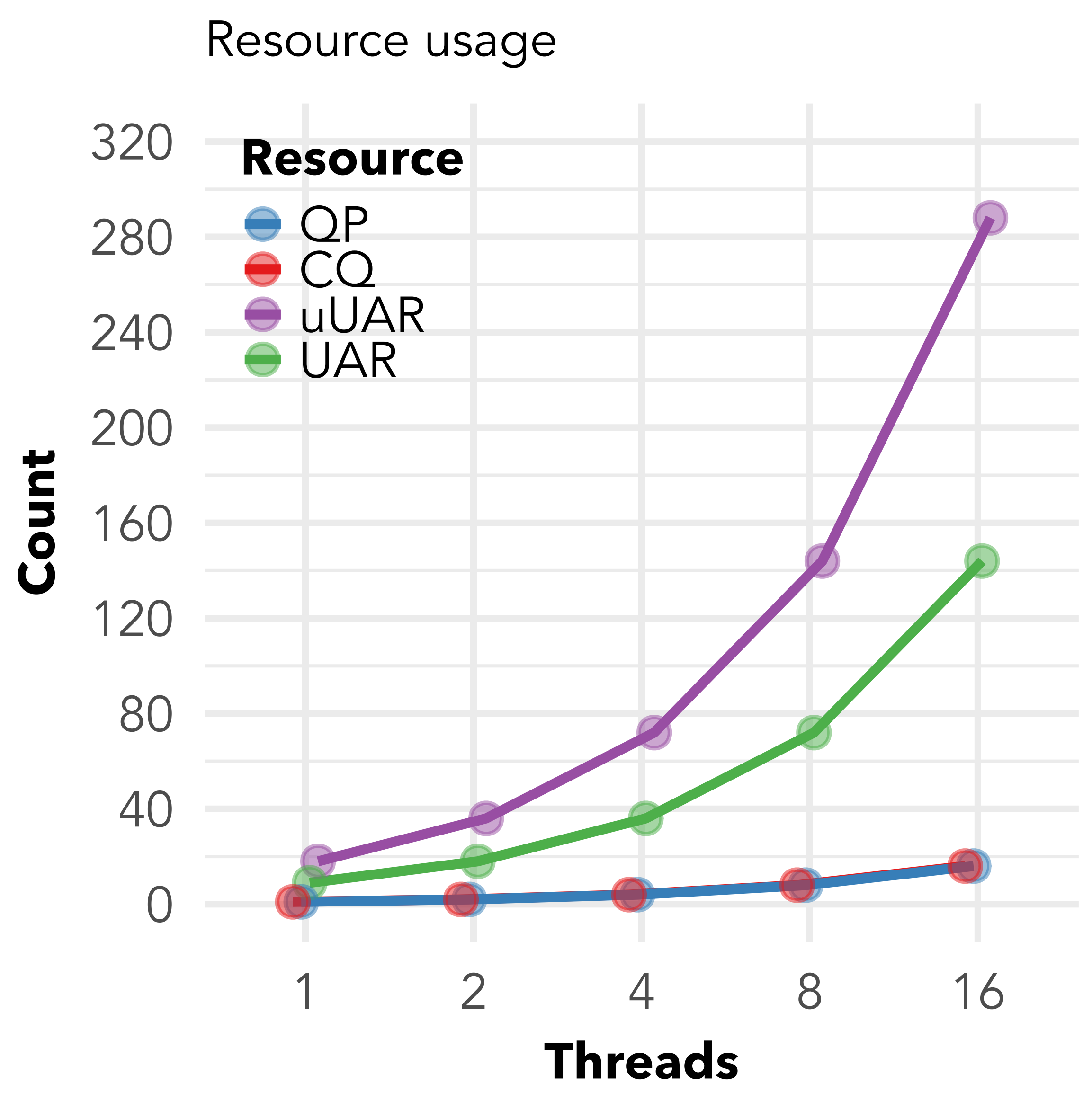




EVALUATION SETUP

- Multi-threaded 2-byte RDMA-write sender-receiver message rate benchmark
- Intel Haswell @ 2.5 GHz + ConnectX-4
- 16 threads
- QP depth: 64
- Postlist: 32; Unsignaled: 64
- To study effect of feature *f*: "All w/o *f*"



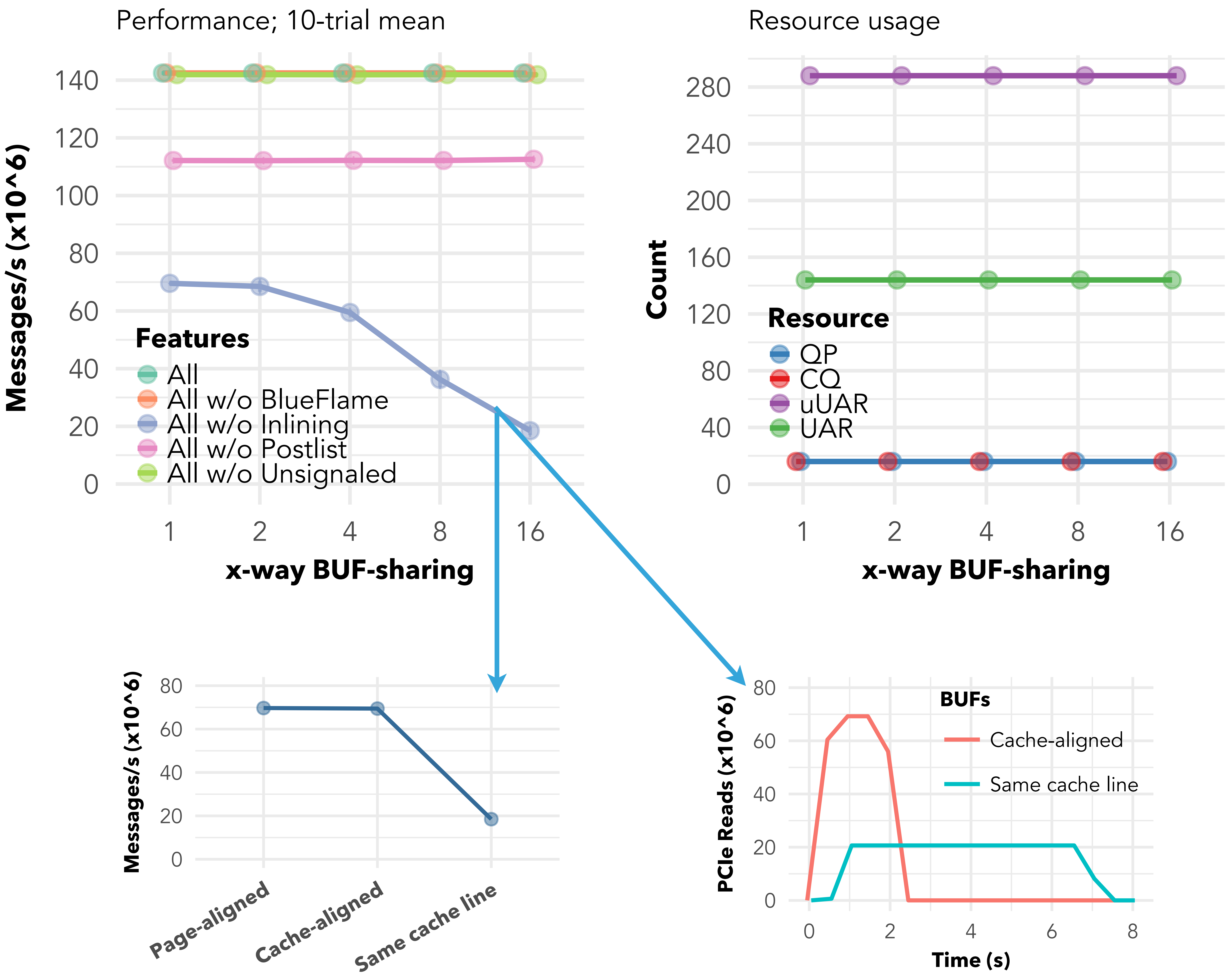


BUFFER SHARING

Hash function of NIC's parallel TLB based on cache line.

Without inlining, parallel DMA reads to same address are serialized.

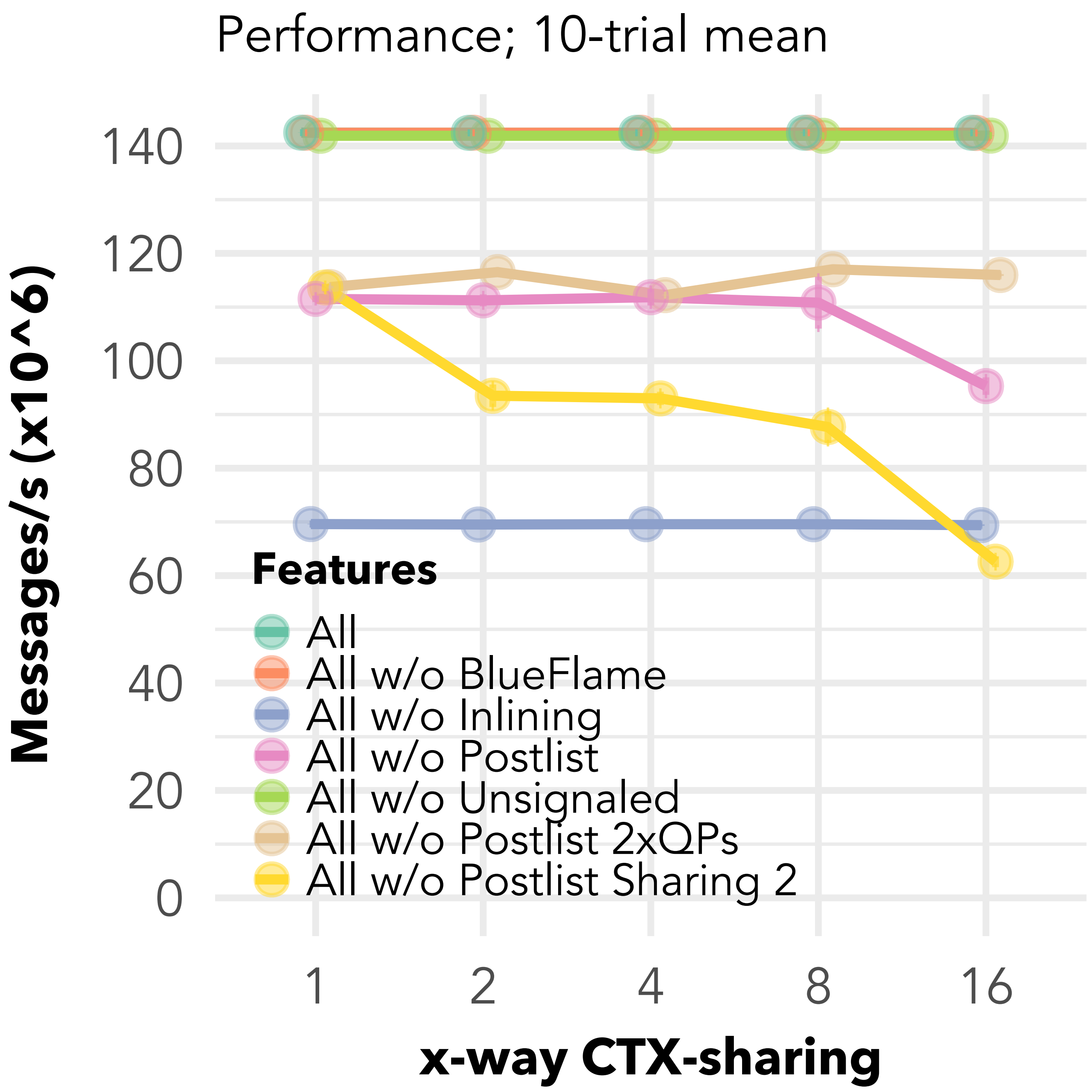
No impact on communication resources

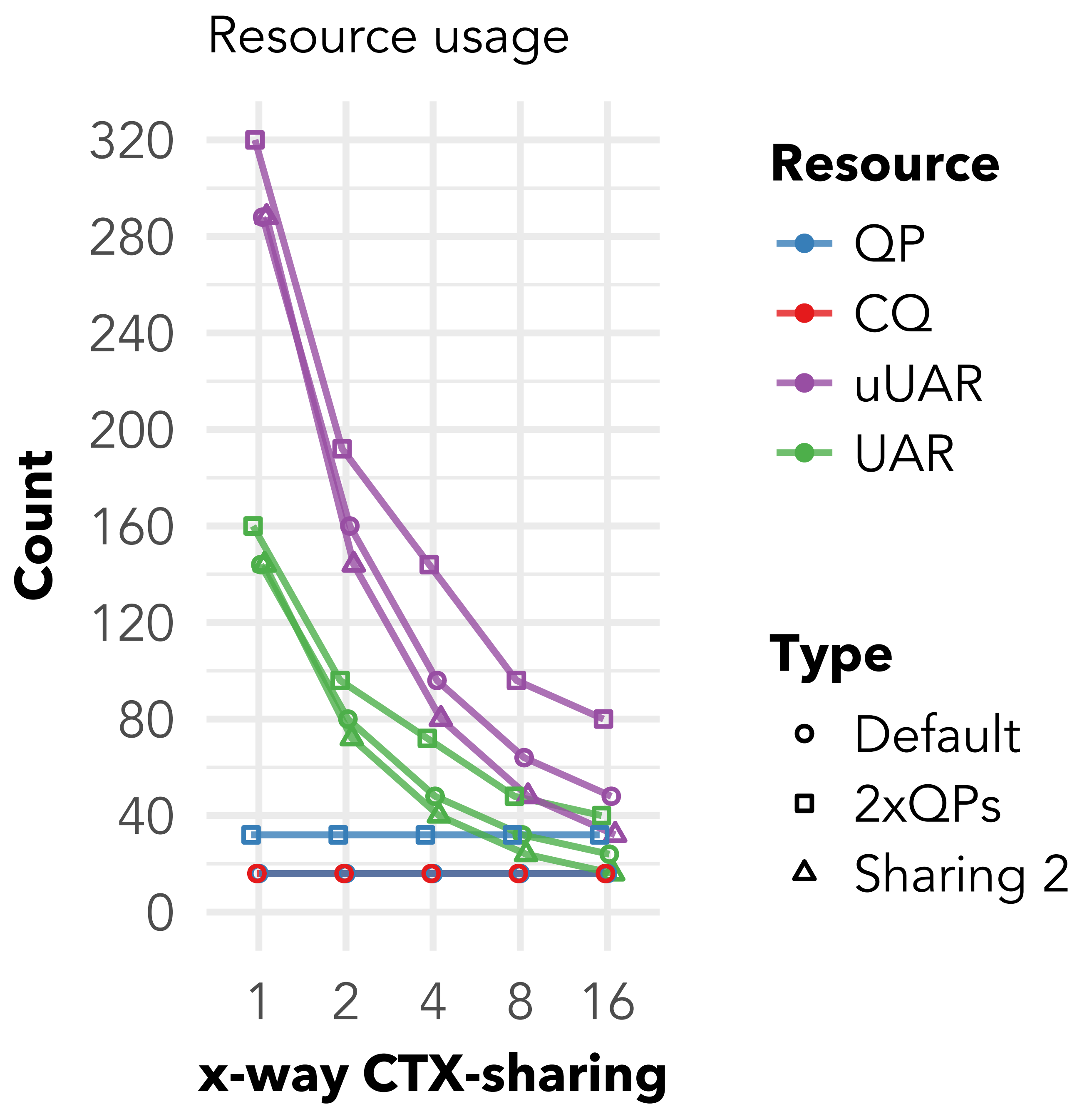


CONTEXT SHARING

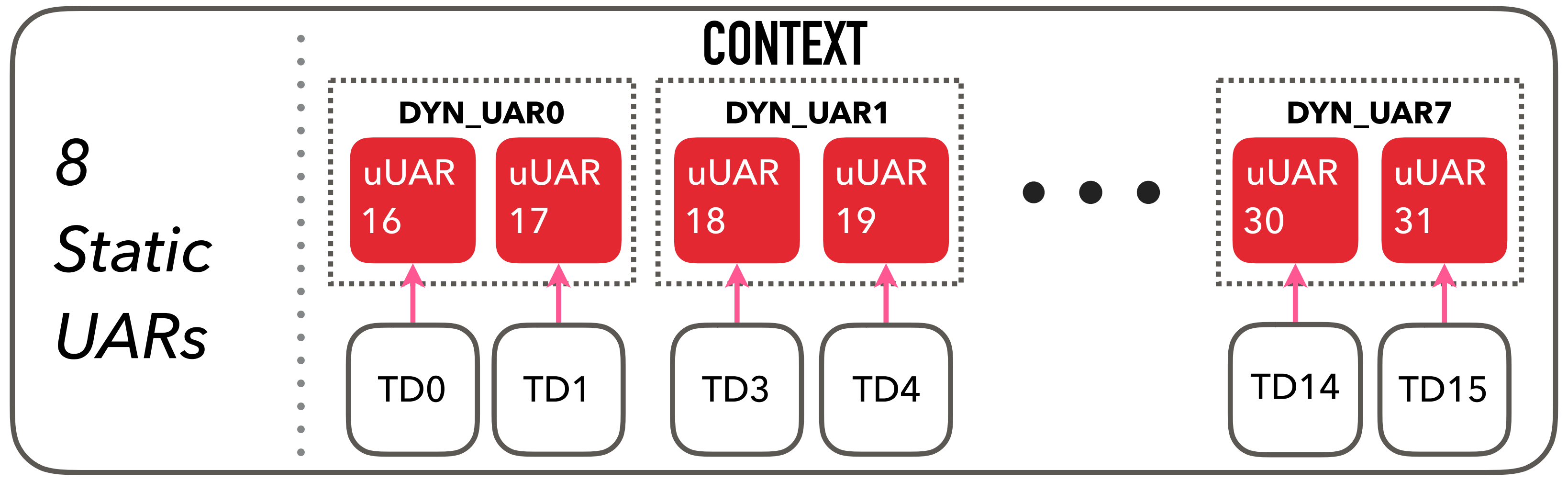
Sharing the UAR hurts performance with BlueFlame

No impact on QPs and CQs. Very impactful on UAR and uUAR usage.

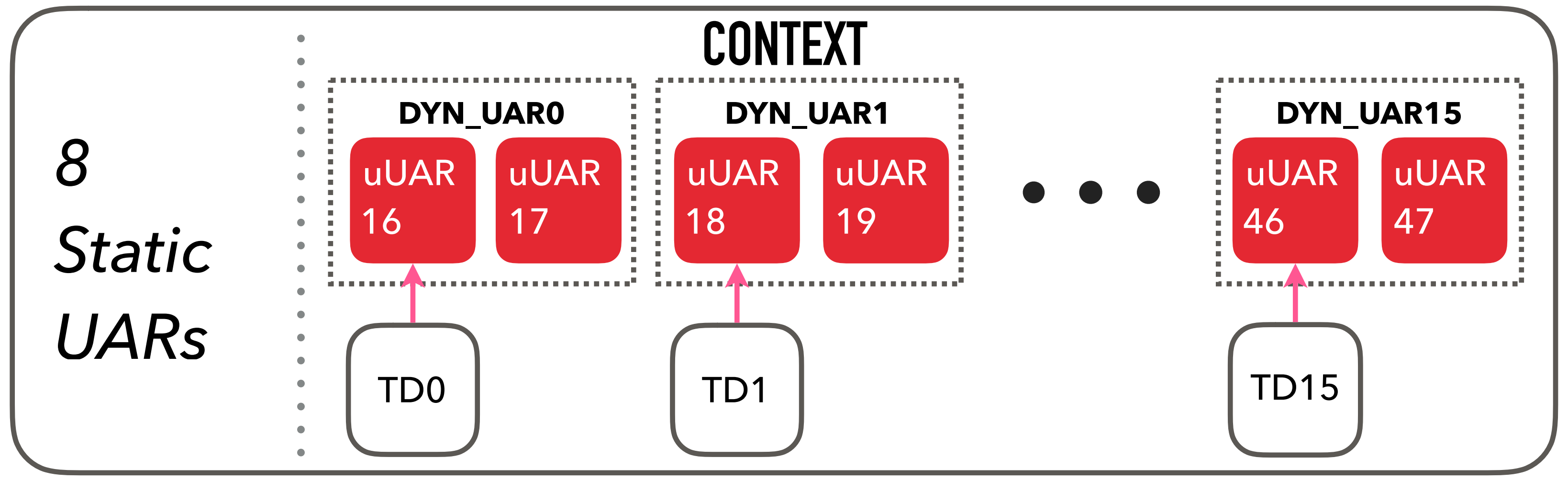




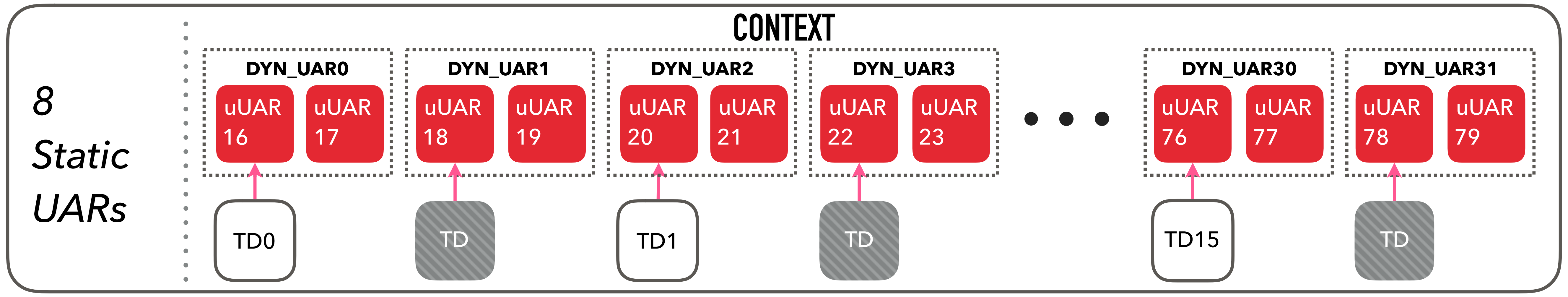
CONTEXT SHARING



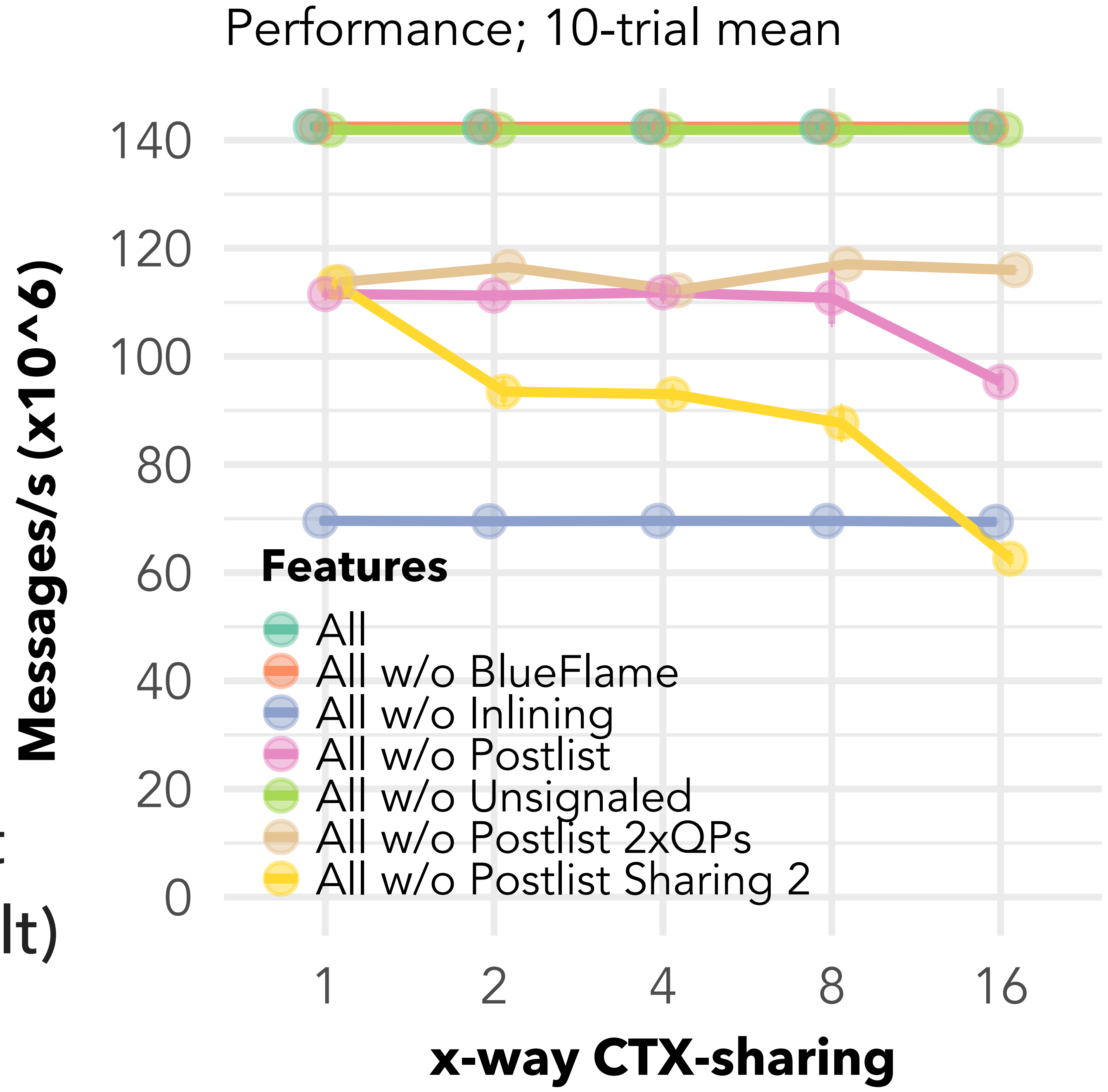
Sharing 2

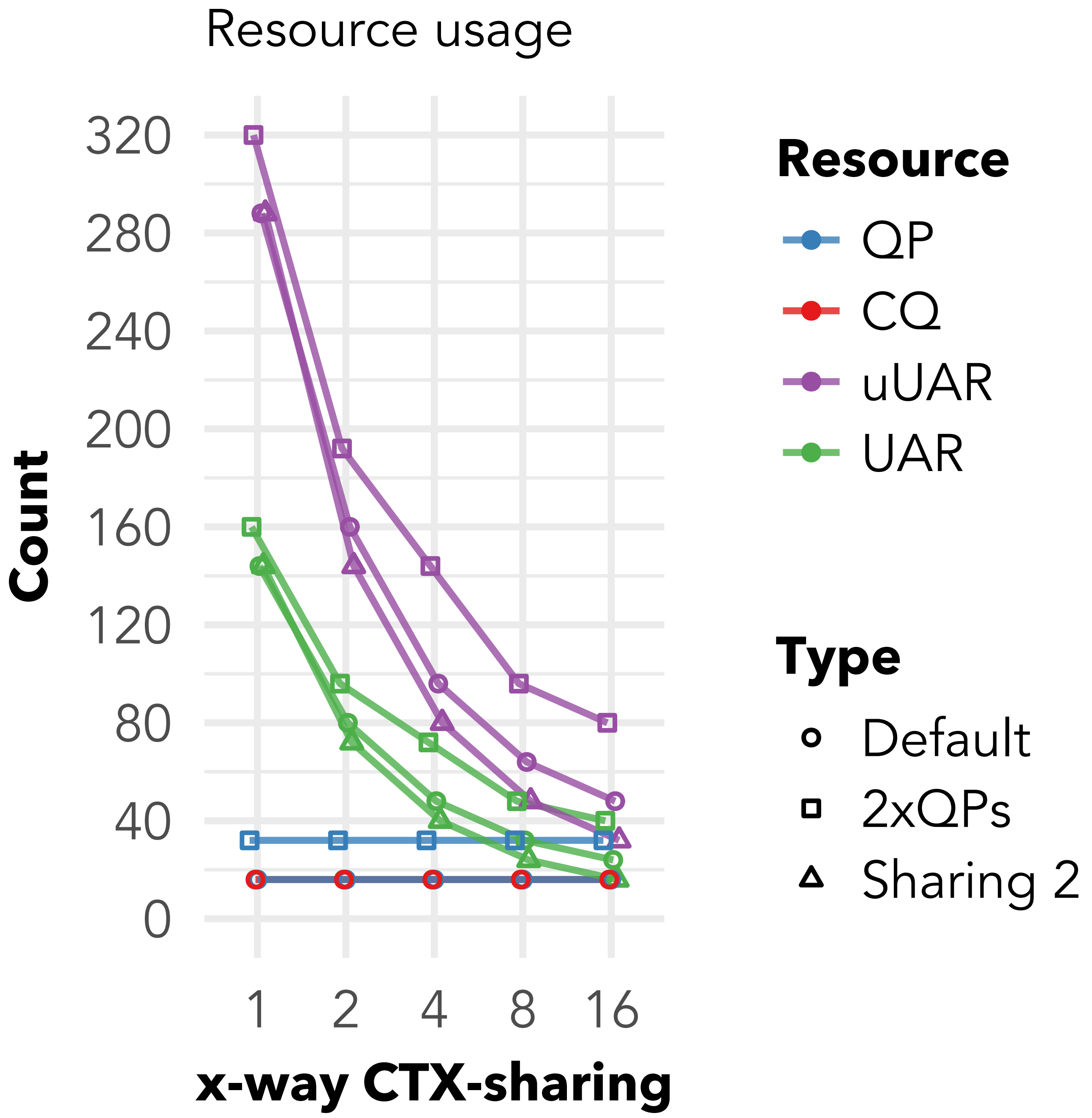


Maximally independent paths (Default)



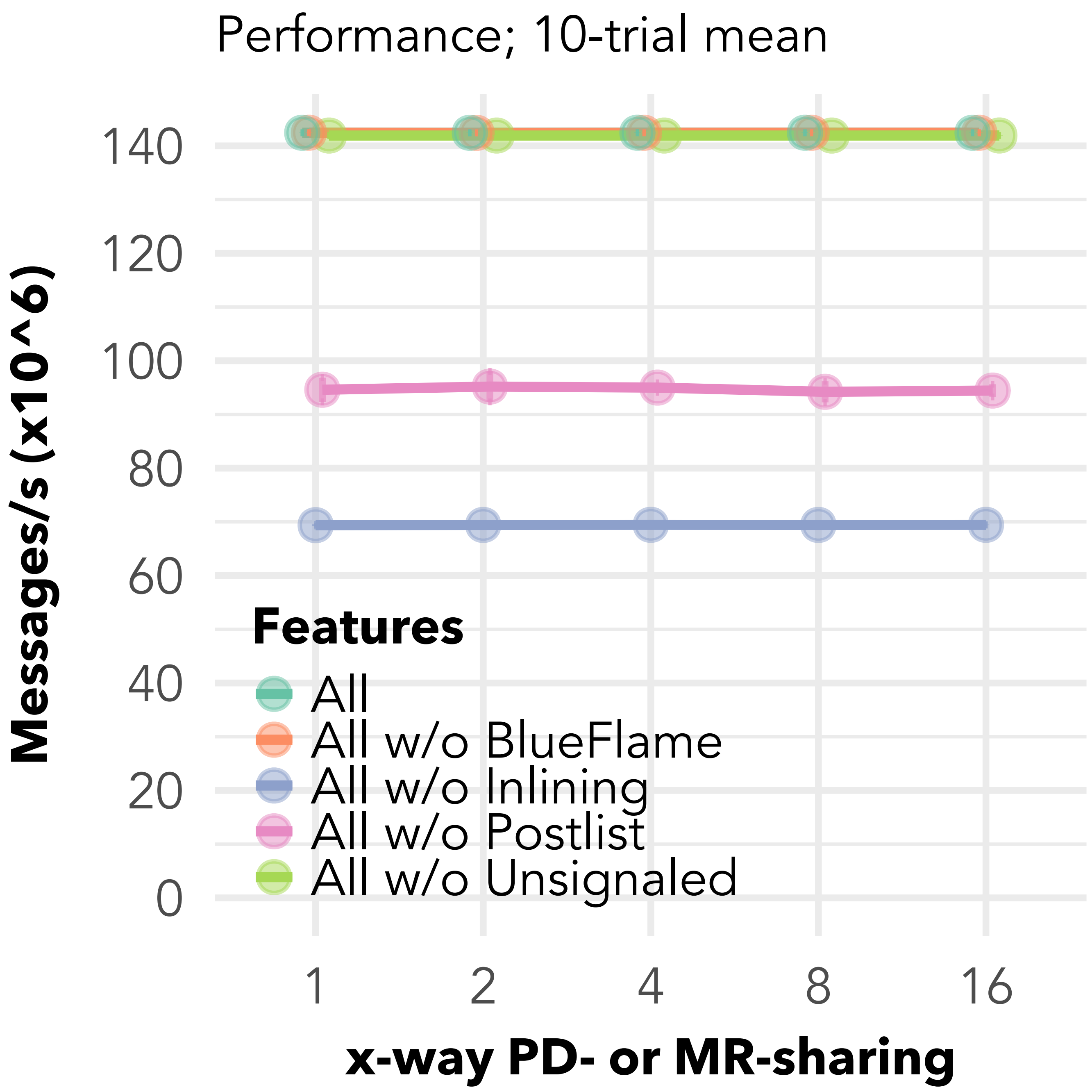
2xQPs

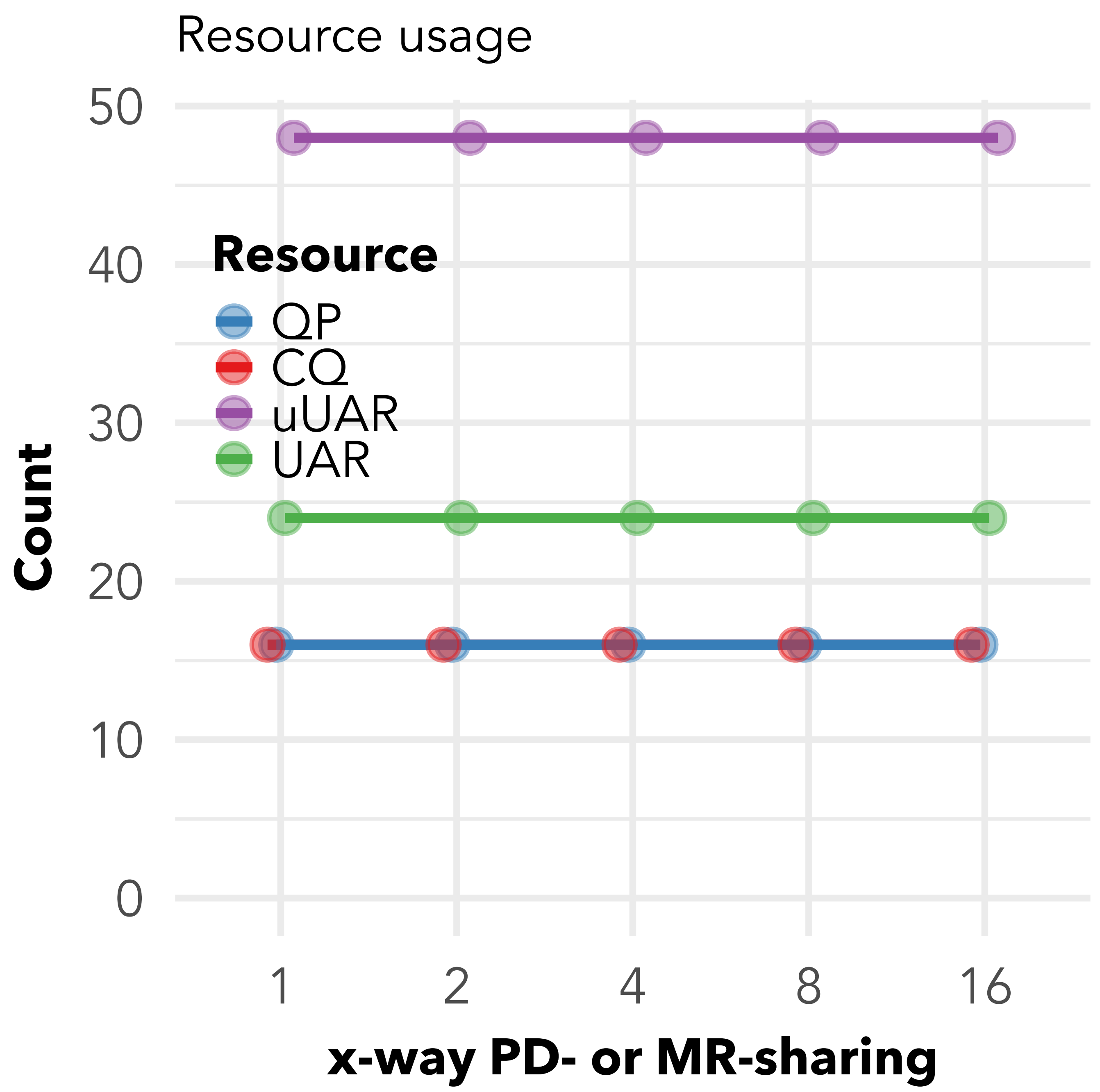




PROTECTION DOMAIN OR MEMORY REGION SHARING

No impact on performance or communication resource usage

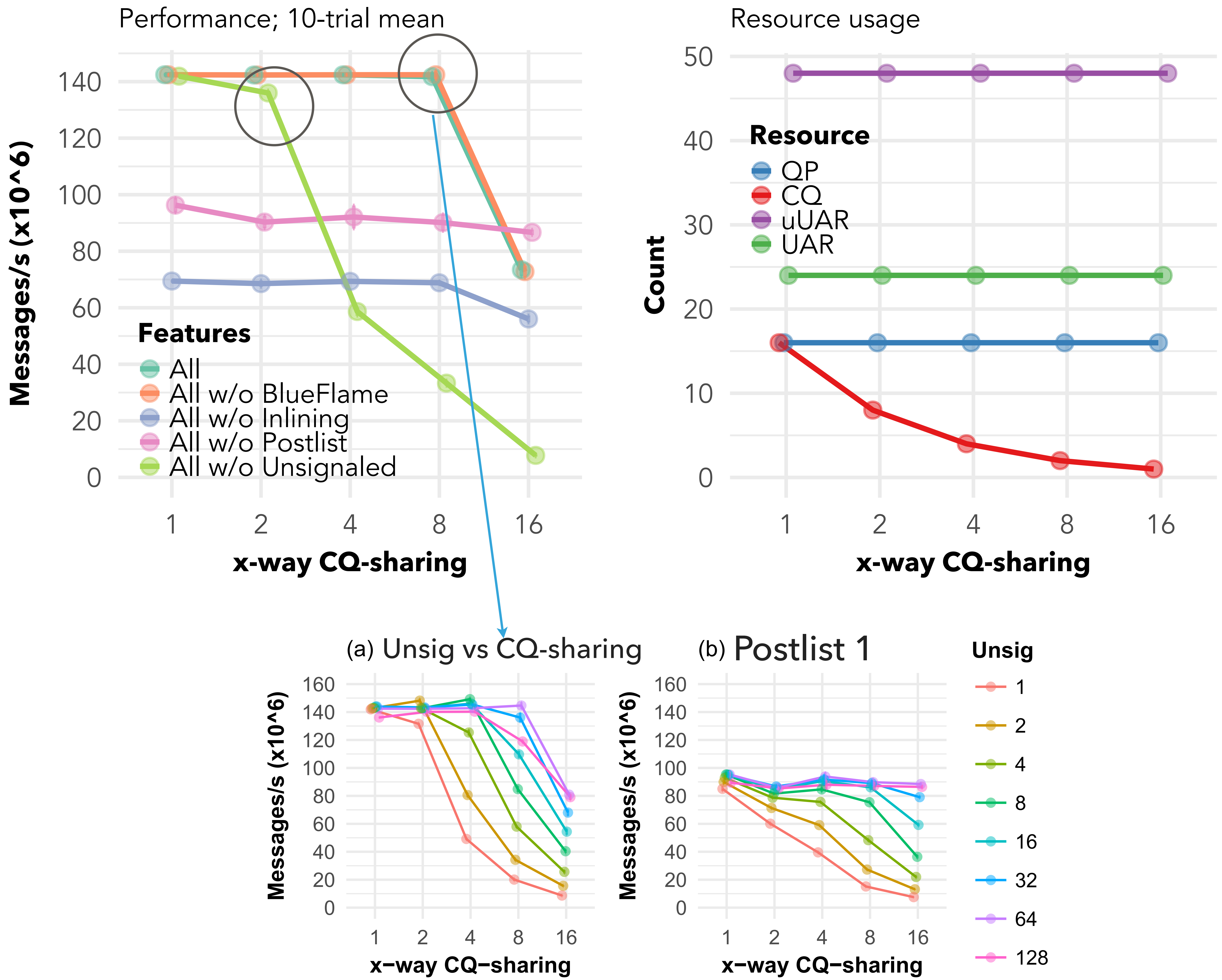




COMPLETION QUEUE SHARING

Without unsignaled, lock-contention, and atomics on the CQ degrades performance.

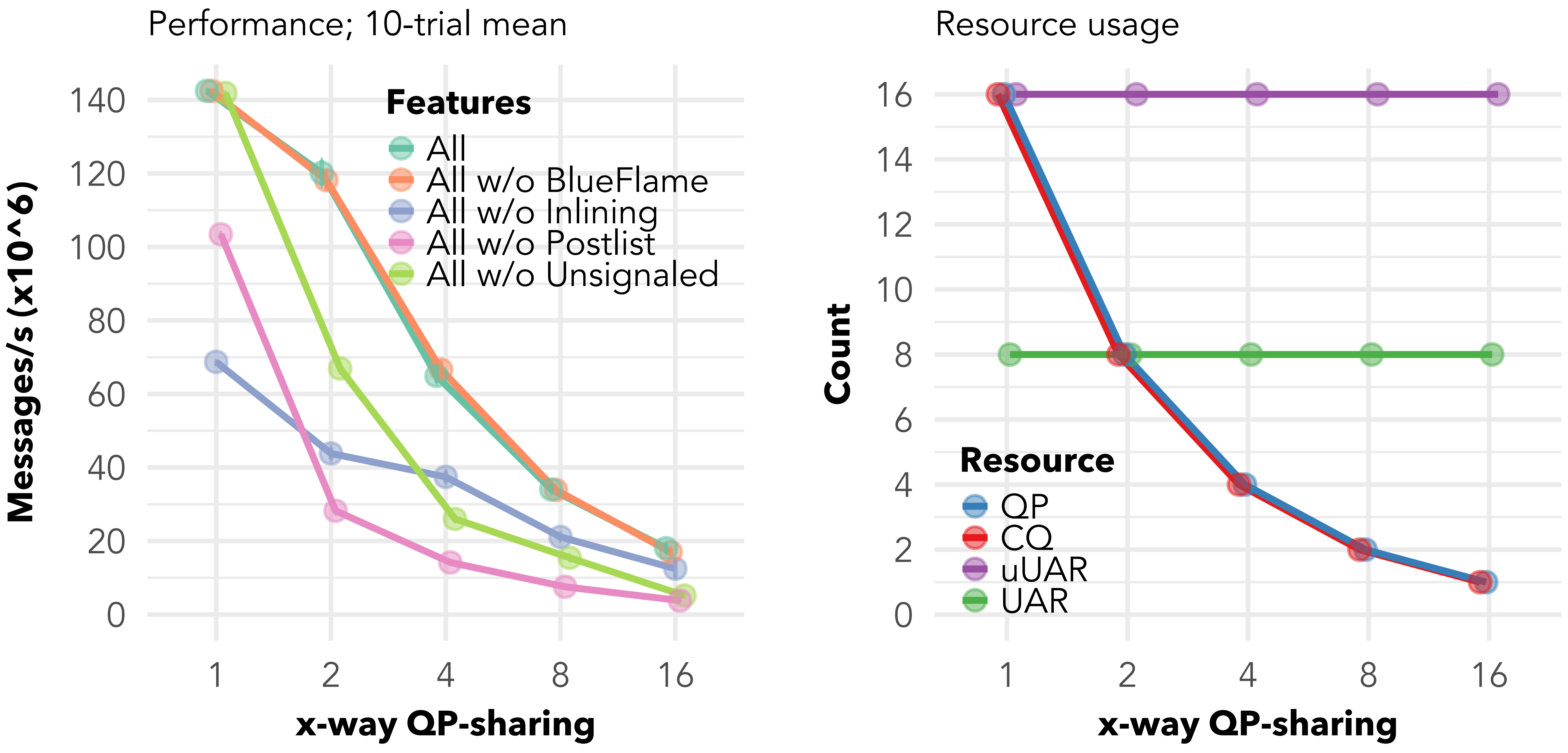
Impacts only CQ usage.



QUEUE PAIR SHARING

Contention on QP's lock, atomics, and decrease in utilization of underlying network parallelism.

Impacts only QP and CQ usage.



LESSONS LEARNED FROM ANALYSIS

- Each thread must have its own cache-aligned buffer
- Can use Protection Domain and Memory Region at will.
- CTX-sharing the most critical for hardware resource usage.

16 threads	% Performance	Lesser uUARs
No CTX sharing	100%	288
2xDynamic	100%	3.6x (80)
Default TDs	80%	6x (48)
Sharing 2	50%	9x (32)

LESSONS LEARNED FROM ANALYSIS

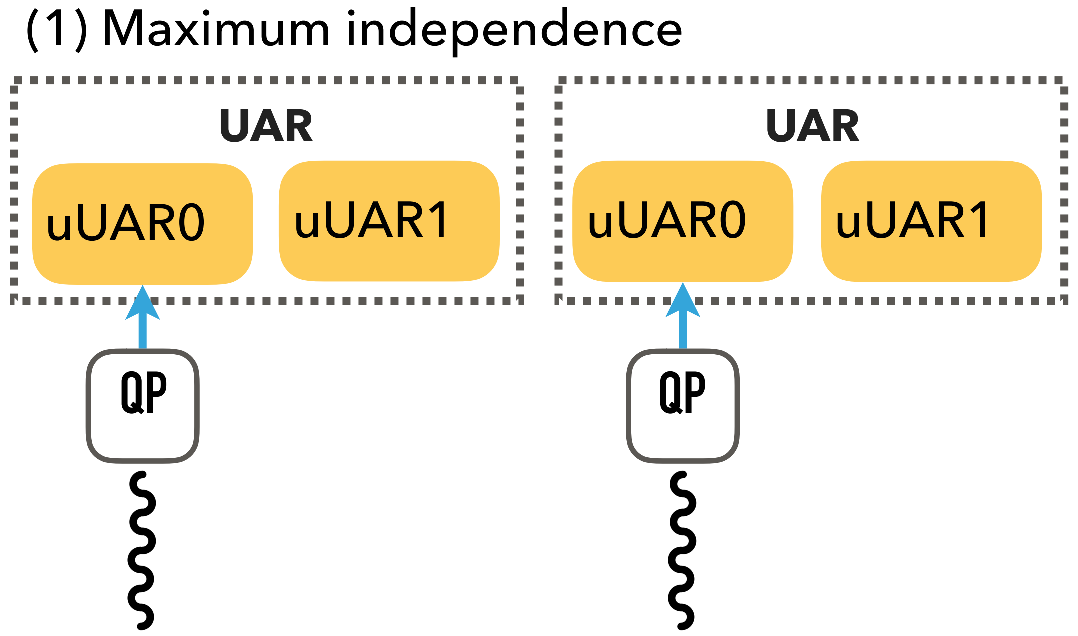
- Each thread must have its own cache-aligned buffer
- Can use Protection Domain and Memory Region at will.
- CTX-sharing the most critical for hardware resource usage.
- Only QP- and CQ-sharing impact memory usage
  - However reduction in memory usage not as meaningful (1.1x) when compared to the consequent drop in performance (18x with 16-way CQ-sharing)

SCALABLE ENDPOINTS

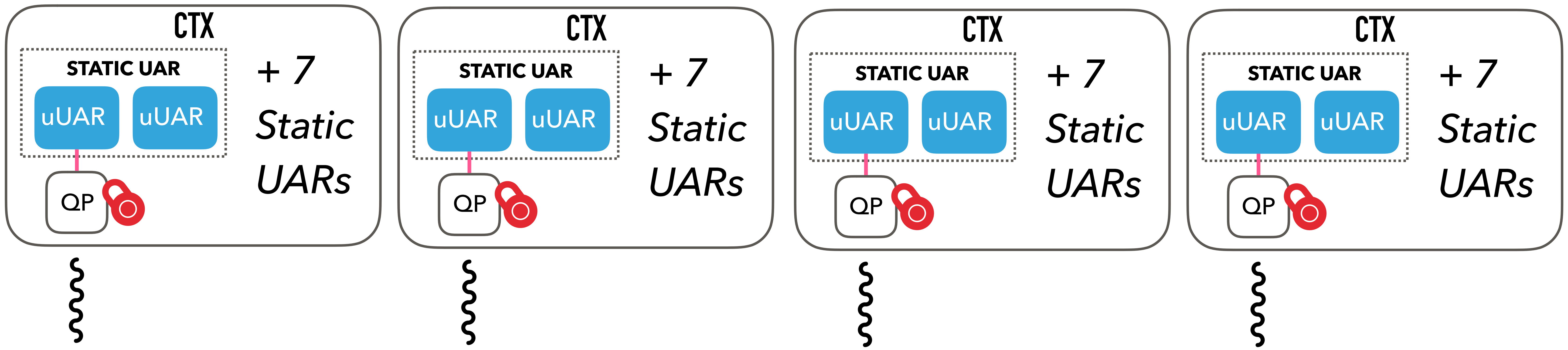
Max performance but not best

15 uUARs wasted per thread

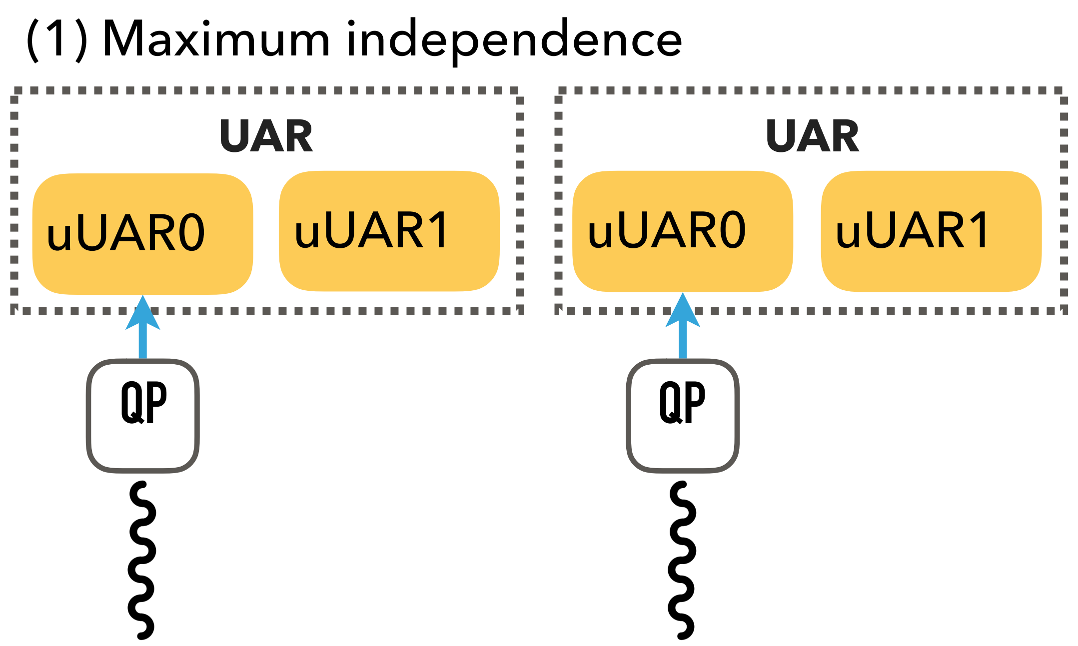
Memory usage identity function of threads



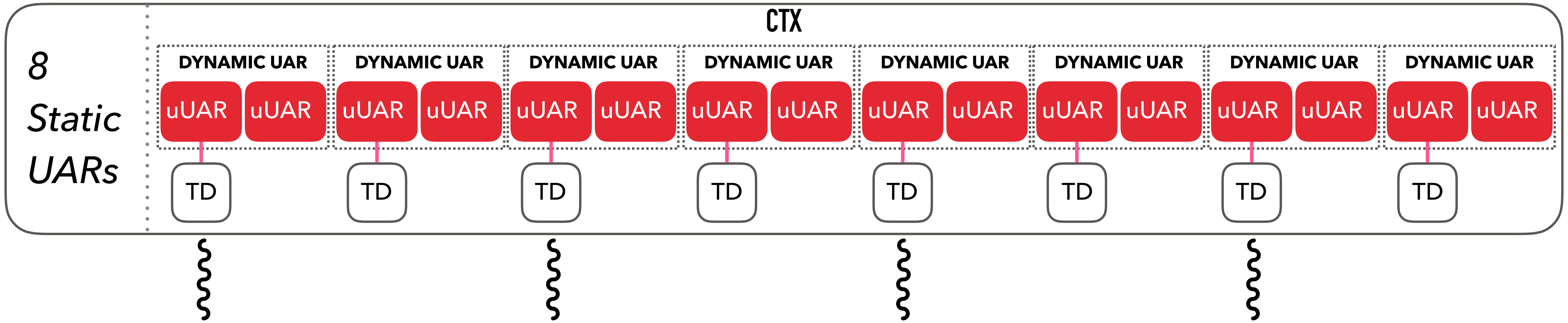
MPI-everywhere



SCALABLE ENDPOINTS



2xDynamic



Best performance

3 uUARs per thread + 16 uUARs wasted

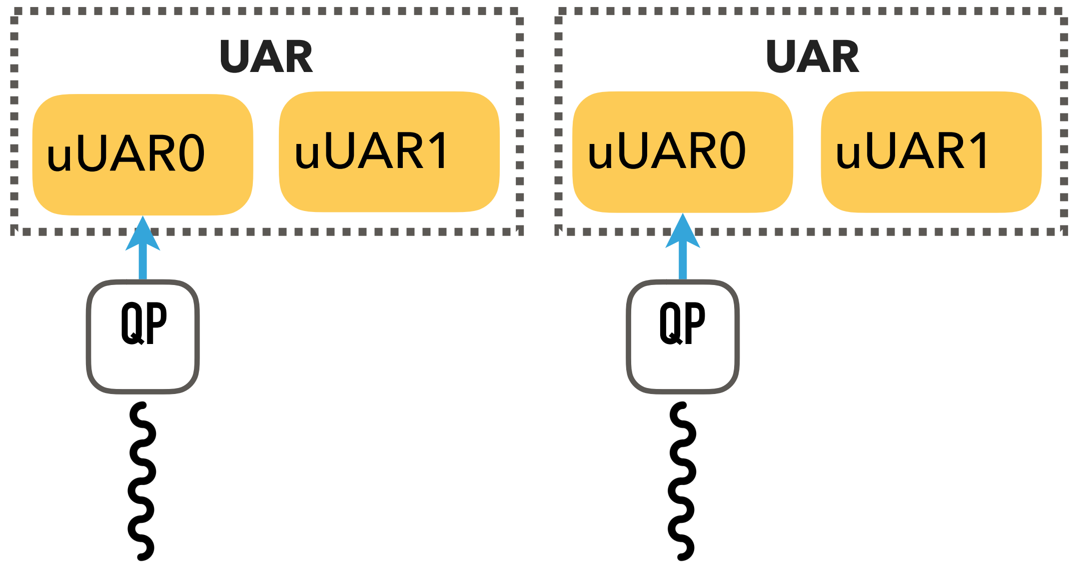
Memory usage twice as that of MPI-everywhere

SCALABLE ENDPOINTS

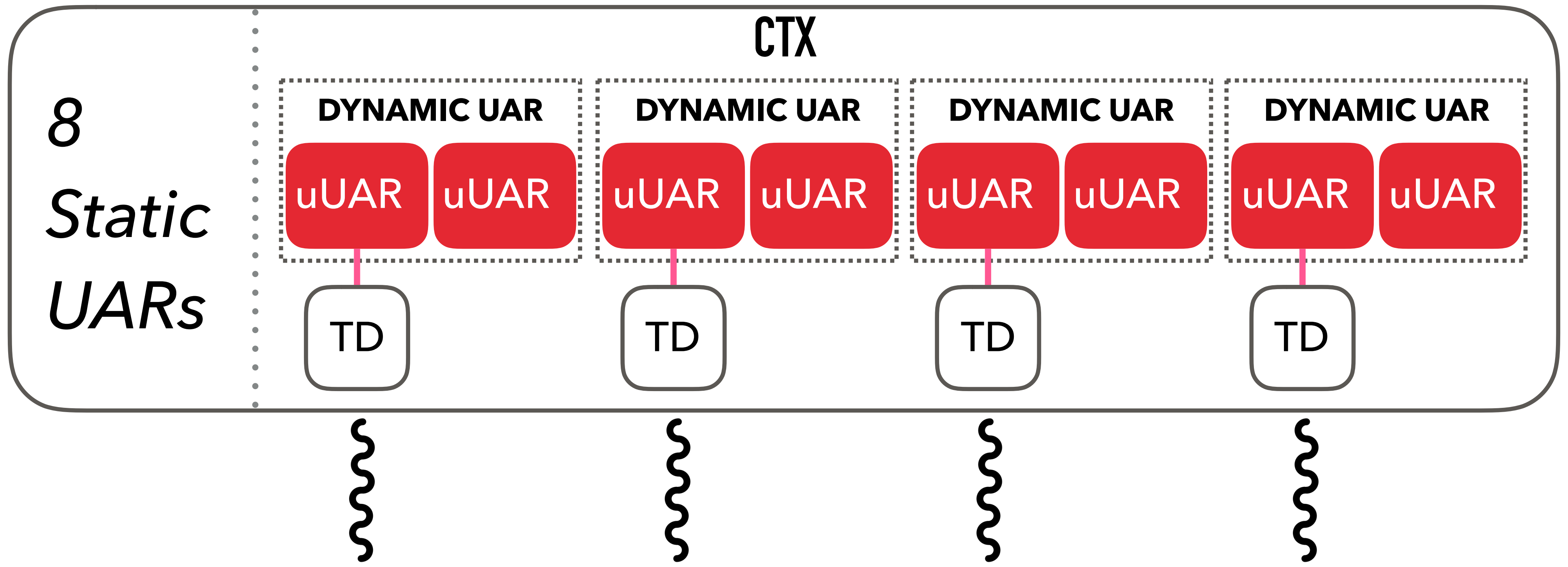
Will hurt performance

1 uUAR per thread + 16 uUARs wasted

Memory usage same as MPI-everywhere



Dynamic

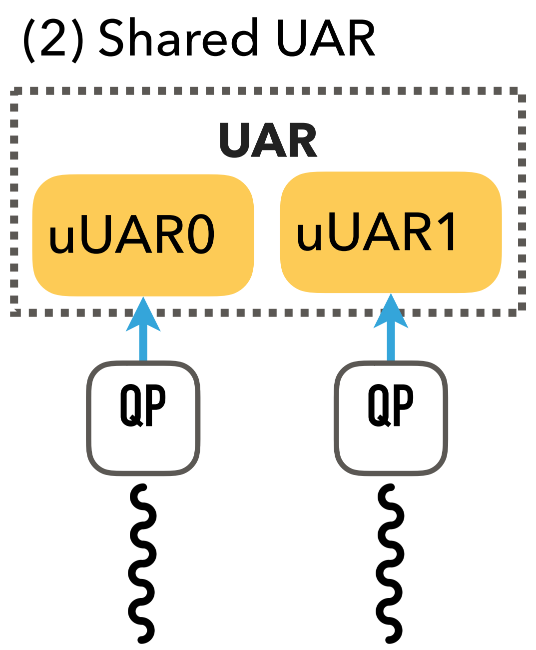


SCALABLE ENDPOINTS

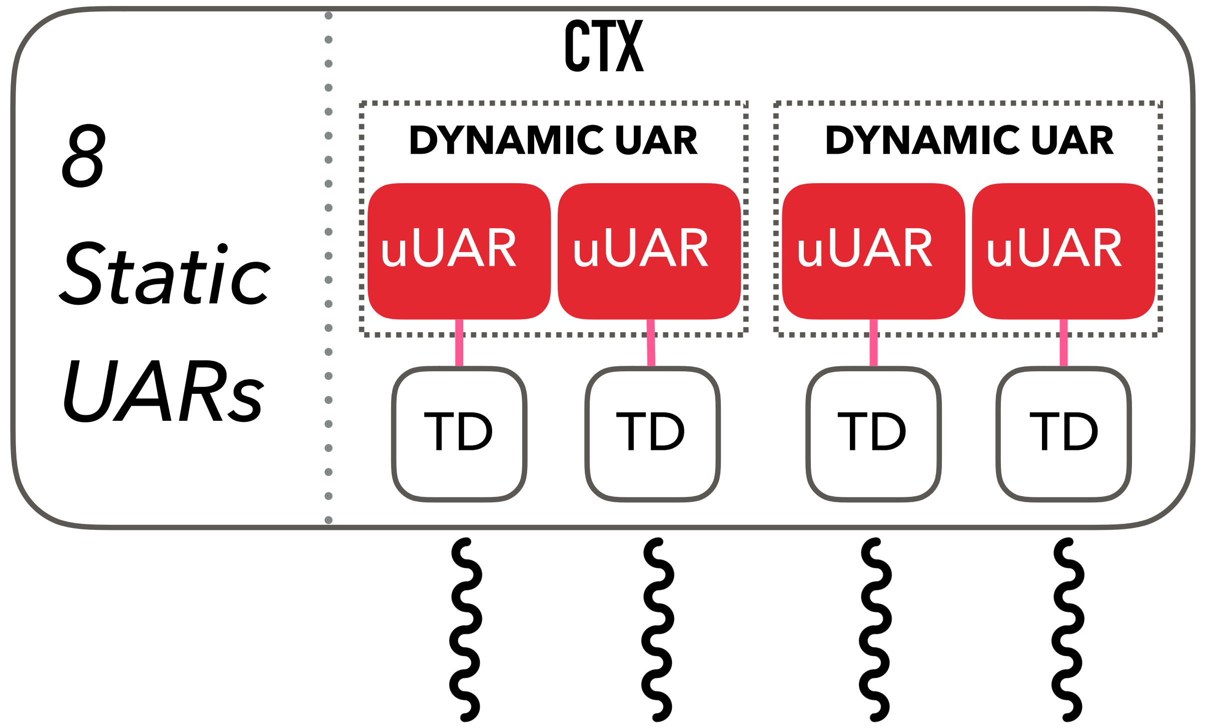
Will hurt performance

16 uUARs wasted

Memory usage same as MPI-everywhere



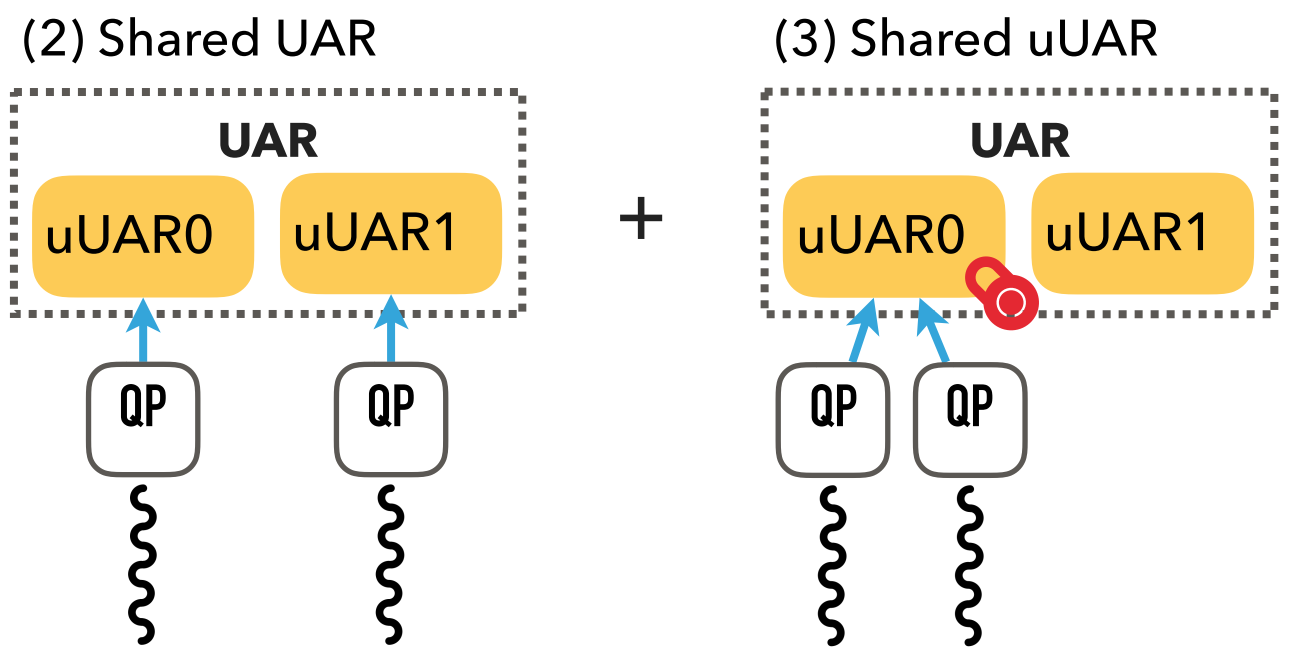
Shared Dynamic



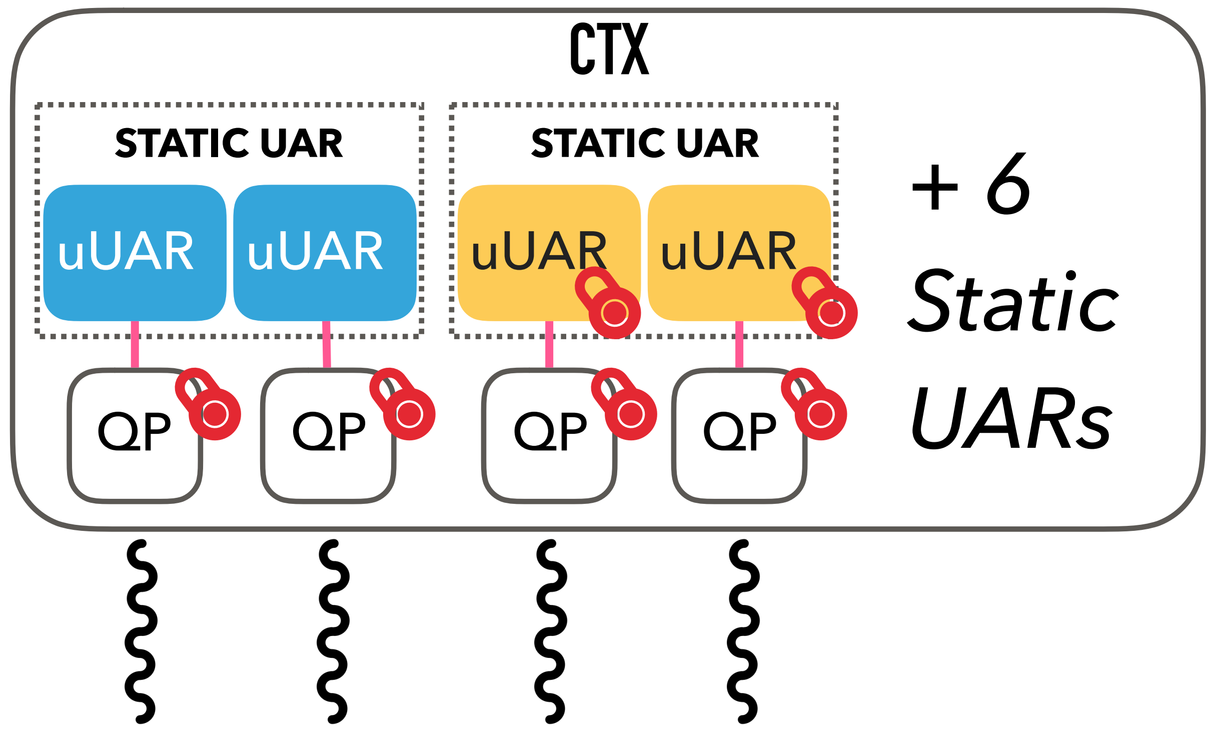
SCALABLE ENDPOINTS

Locks + UAR sharing will hurt performance

Memory usage same as MPI-everywhere



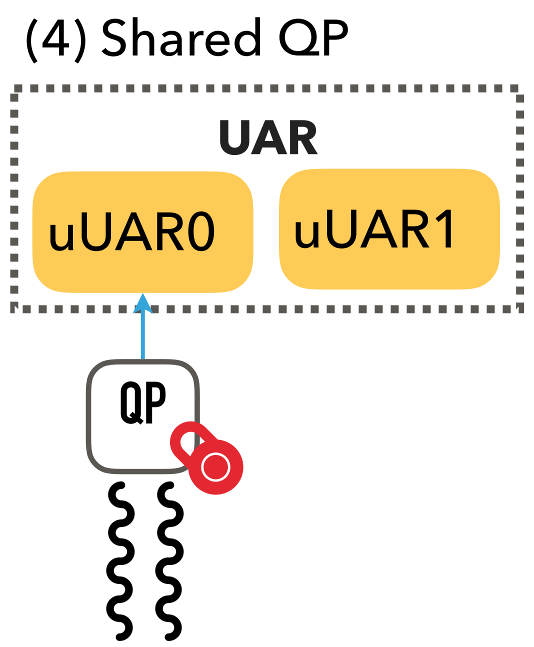
Static

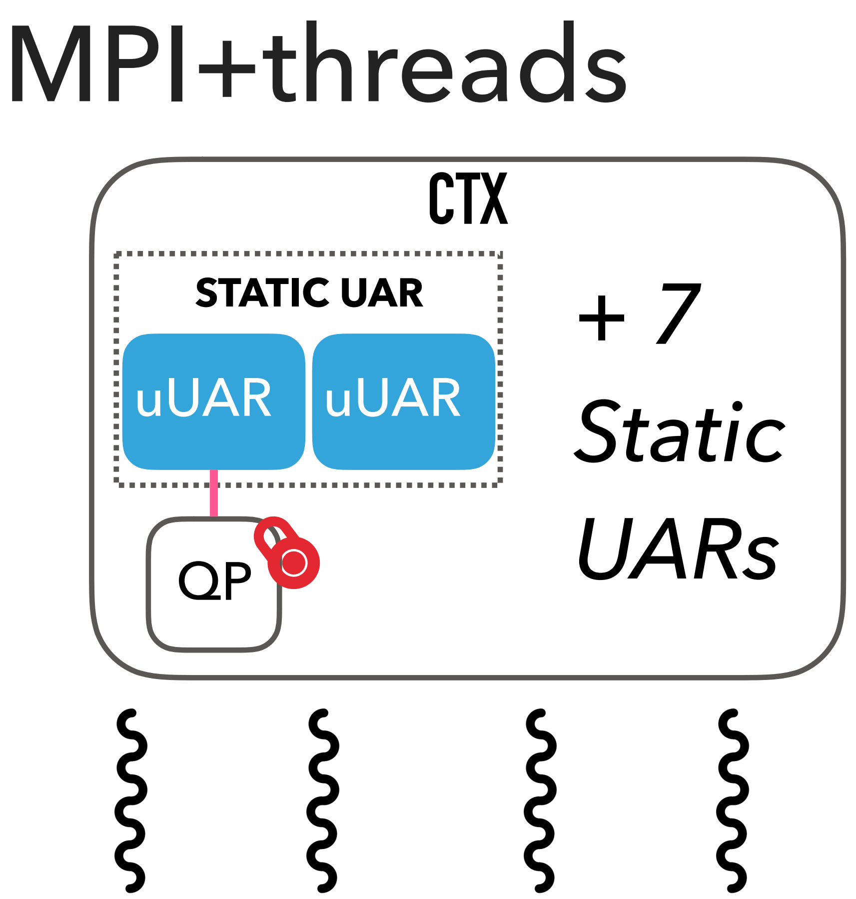


SCALABLE ENDPOINTS

Worst performance

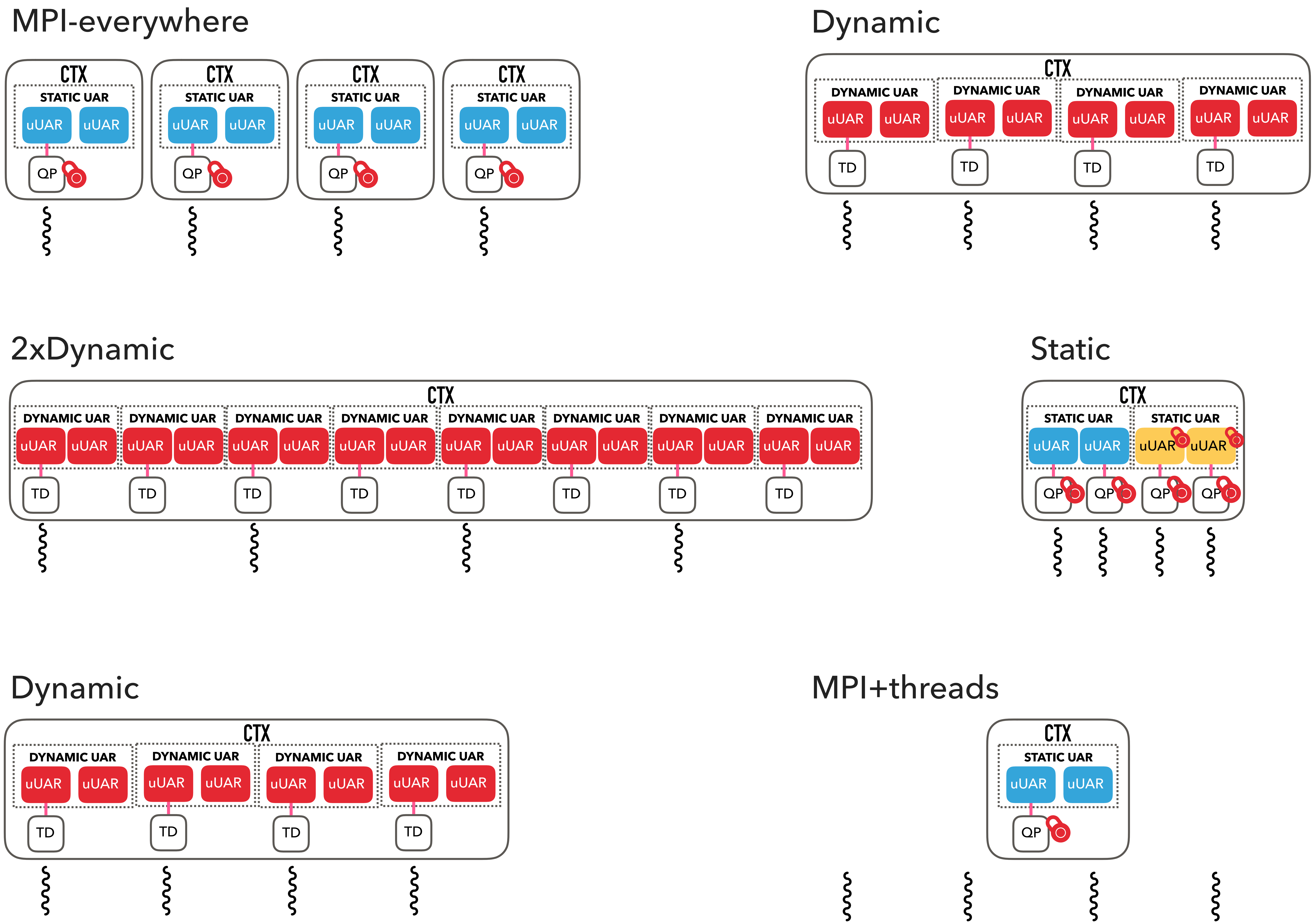
Least resource usage





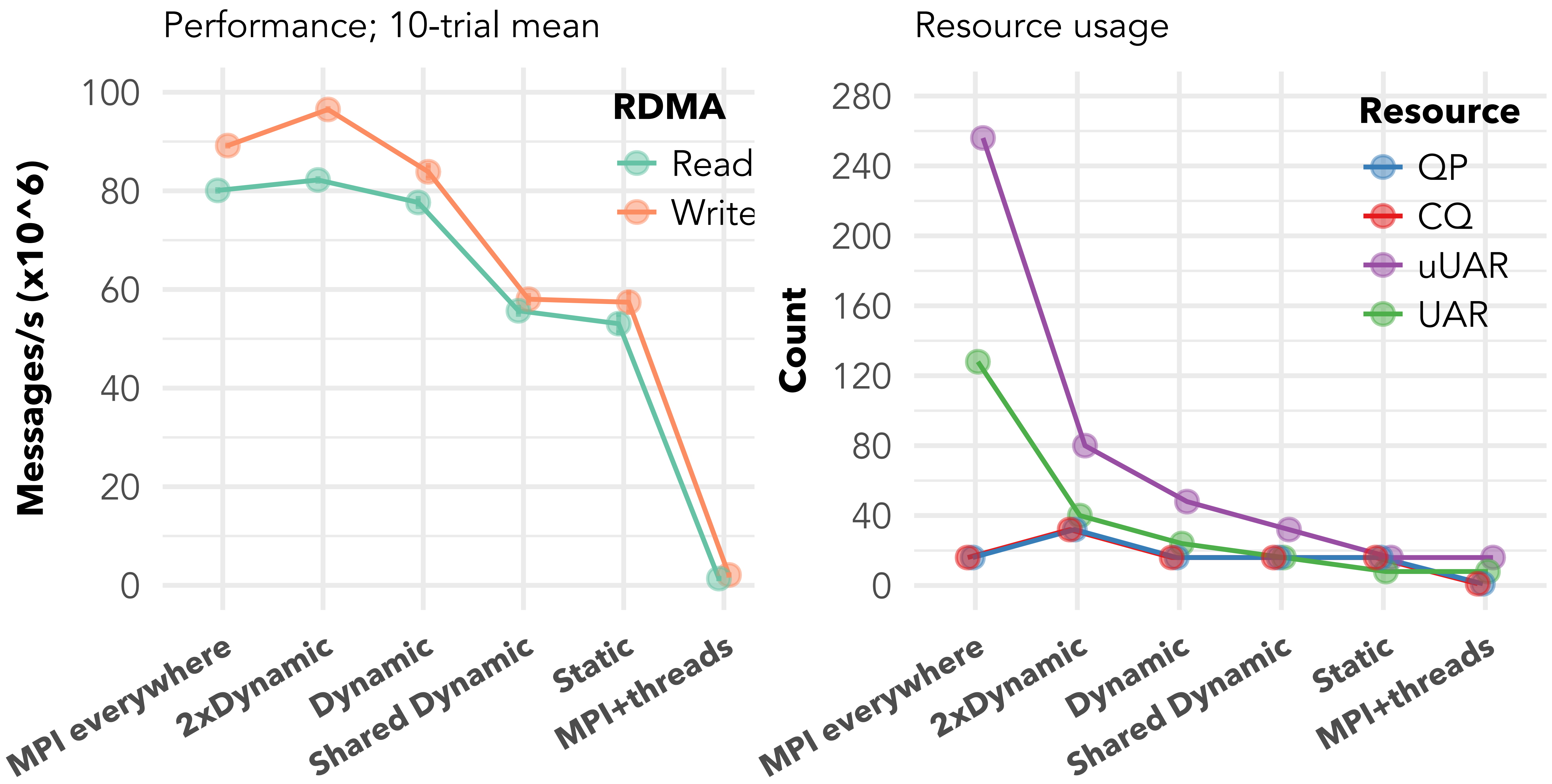
SCALABLE ENDPOINTS

CQ can be shared in any manner for any of these categories.

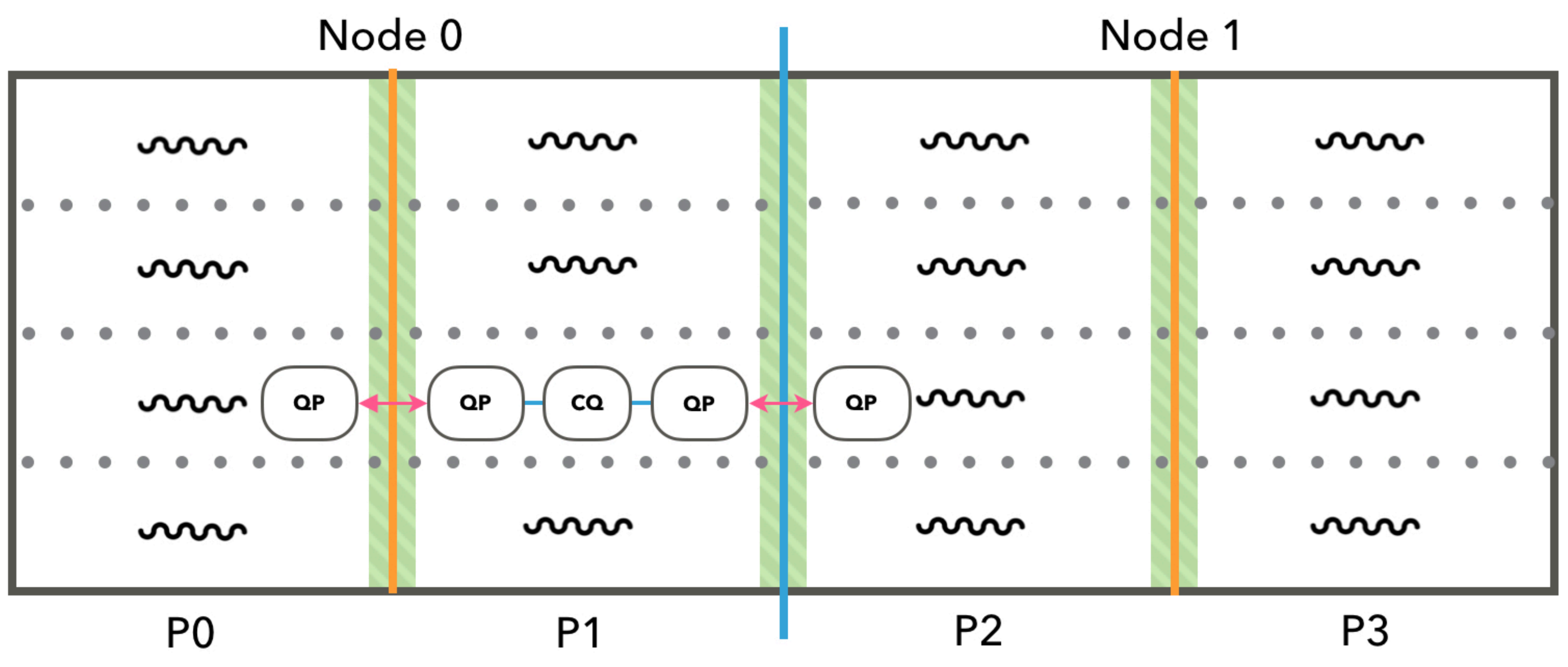


GLOBAL ARRAY EVALUATION

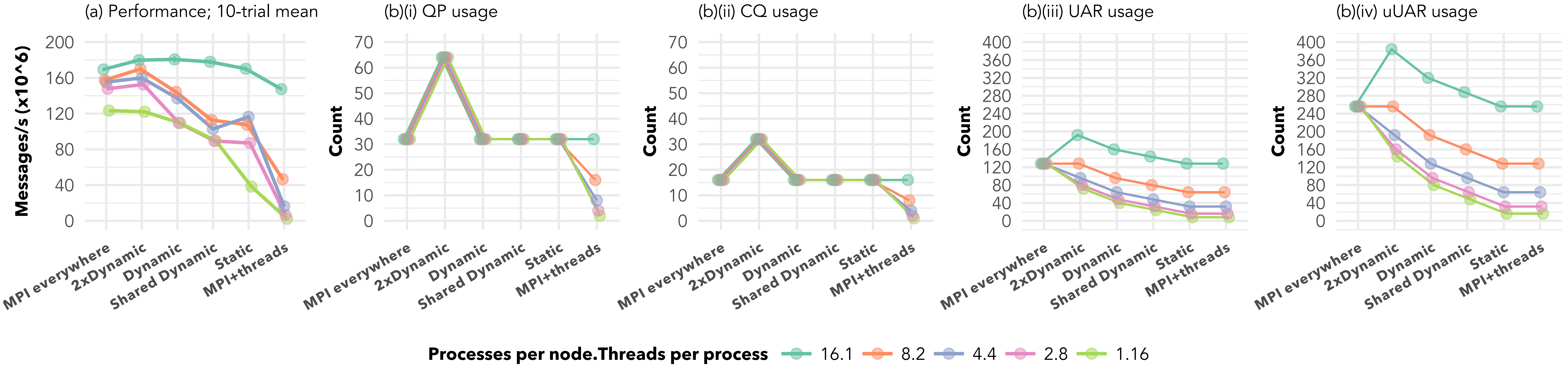
- Fetching and writing tiles from a global array
- Double precision global matrix multiply (DGEMM) A x B = C
  - All three sit on a server node
  - Client node RDMA-gets tiles, multiplies them, RDMA-puts them back
- Trend: Performance decreases with increasing resource sharing



STENCIL EVALUATION



- 5-pt stencil, 1D partitioning with varying hybrid environments for 16 threads.
- Similar trend of decreasing performance with increasing resource sharing



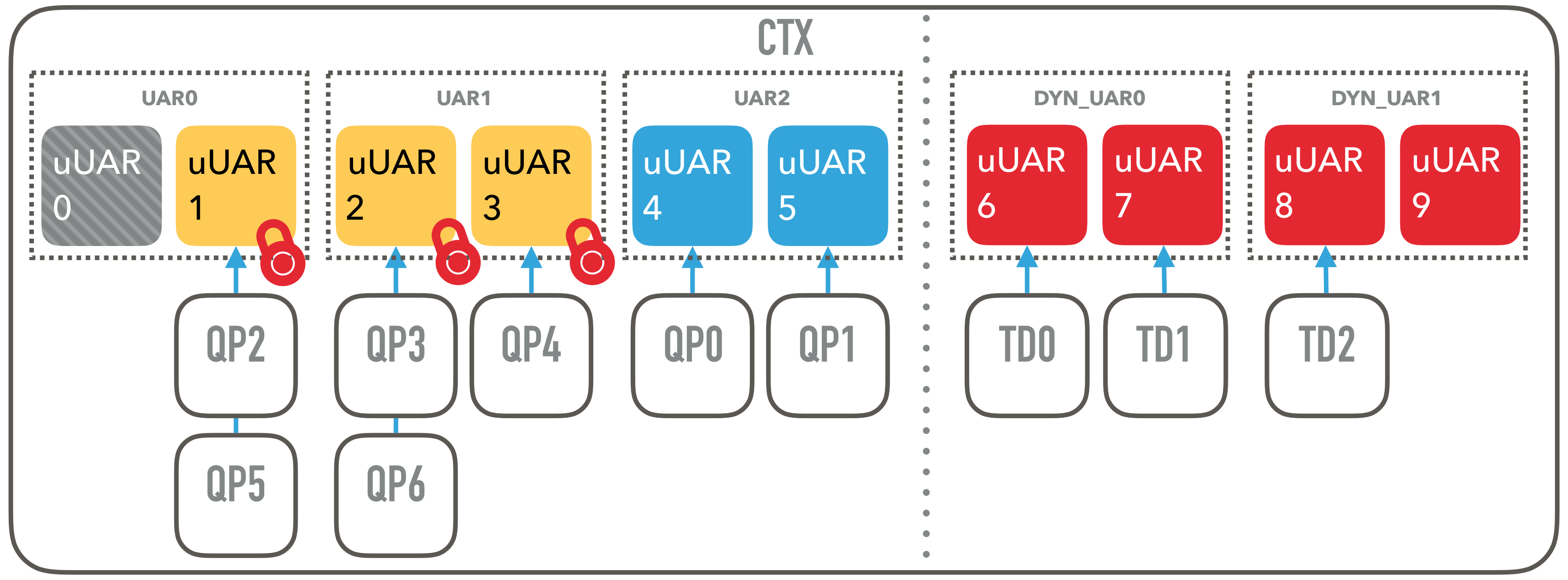
RELATED WORK AND FUTURE DIRECTION

- MPI Endpoints proposal (https://github.com/mpi-forum/mpi-issues/issues/56)
  - Top-down approach: create extra MPI processes to serve as "endpoints" for threads.
  - James Dinan et al.
- Ours is a bottom-up approach.
- Figure out how to expose this to MPI through OFI/UCX middleware.
- MPI library can handle endpoints OR expose to MPI user?

BACKUP SLIDES

MAPPING UAR TO QP

- During QP creation, mlx5 maps the QP to a uUAR
- The 16 uUARs considered to be “statically” allocated. mlx5 categorizes 4 of these into low latency (blue) and 11 into regular (yellow) and the zeroth into high latency (grey).
    - BlueFlame-writes possible only on low latency and regular uUARs. High latency uUARs allow only Doorbells.
    - Only 1 QP mapped to low latency uUAR. No lock needed for BlueFlame-writes. Hence, “low latency”.
    - Regular uUARs may be mapped to multiple QPs. Lock necessary for concurrent BlueFlame-writes.
    - High latency uUAR needs no locks for concurrent Doorbells since Doorbells are atomic. Used only when no regular uUARs are present.
- TDs assigned to new, “dynamically” allocated uUARs. Even TDs allocate new UAR page. Odd TDs use uUAR of already allocated UAR.



USER ACCESS REGION

- A micro User Access Region (uUAR) is the hardware resource for a mlx5 device.
- uUARs are part of a UAR page. A UAR is part of the NIC's address space.
- During CTX creation, mlx5 allocates 8 UAR pages by default
- Each UAR Consists of 4 uUARs (also know as blue flame registers)
  - Only the first 2 can be used by user. Latter 2 used by hardware for priority tasks
  - So we have 16 uUARs in each CTX by default.
- First 8 bytes of the uUAR's buffer is the Doorbell register. Even Doorbells on buf_0; odd Doorbells on buf_1.