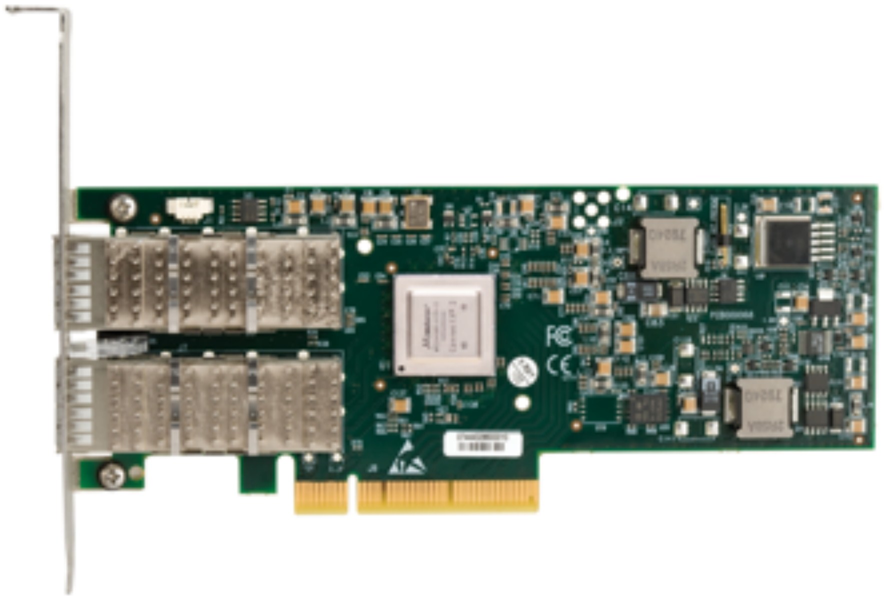
Using RDMA Efficiently for Key-Value Services

Anuj Kalia (CMU)
Michael Kaminsky (Intel Labs), David Andersen (CMU)

RDMA



Remote Direct Memory Access: A network feature that allows direct access to the memory of a remote computer.

HERD

1. Improved understanding of RDMA through micro-benchmarking

2. High-performance key-value system:

    - Throughput: 26 Mops *(2X higher than others)*

    - Latency: 5 μs *(2X lower than others)*

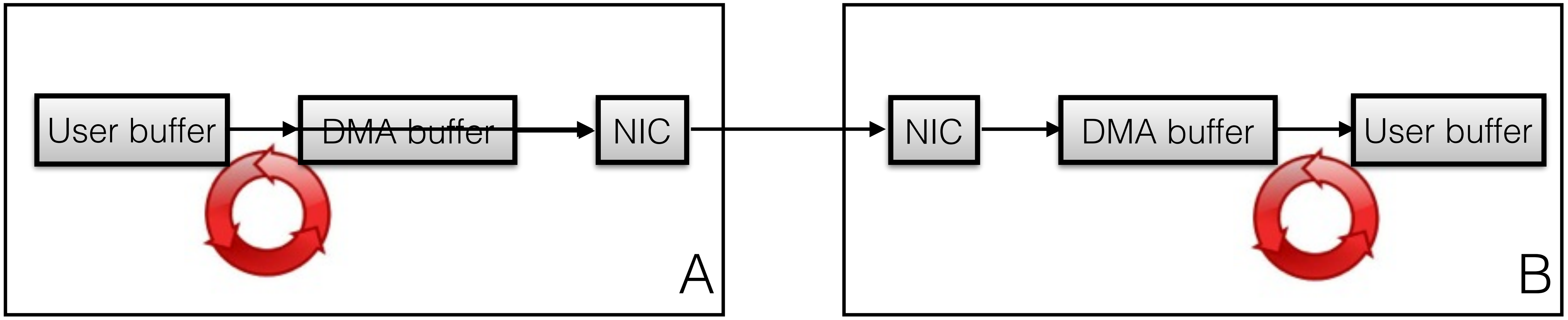
RDMA intro

Features:

- Ultra-low latency: 1 μs RTT
- Zero copy + CPU bypass

Providers:

InfiniBand, RoCE,…

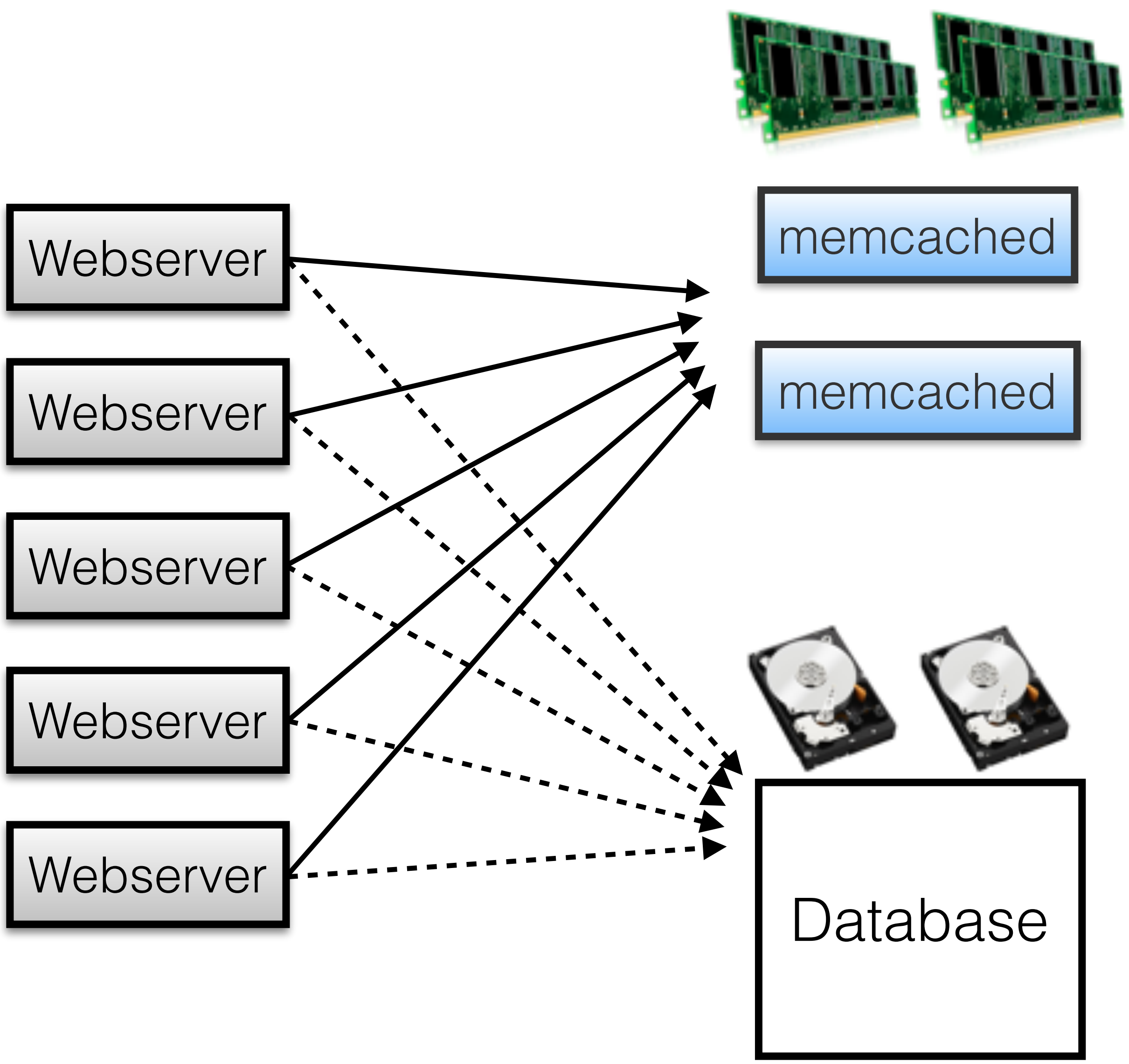


RDMA in the datacenter

48 port 10 GbE switches

Switch	RDMA	Cost
Mellanox SX1012	**YES**	$5,900
Cisco 5548UP	**NO**	$8,180
Juniper EX5440	**NO**	$7,480

In-memory KV stores

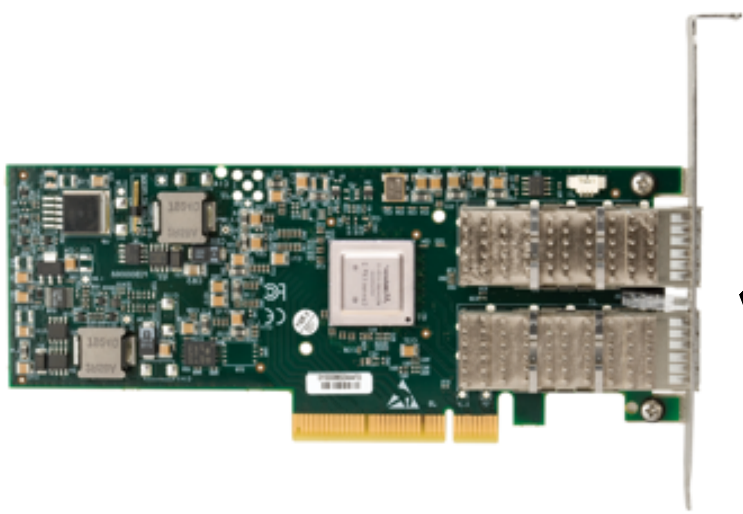


Interface: GET, PUT

Requirements:

- Low latency
- High request rate

RDMA basics



RNIC

Verbs

RDMA read:

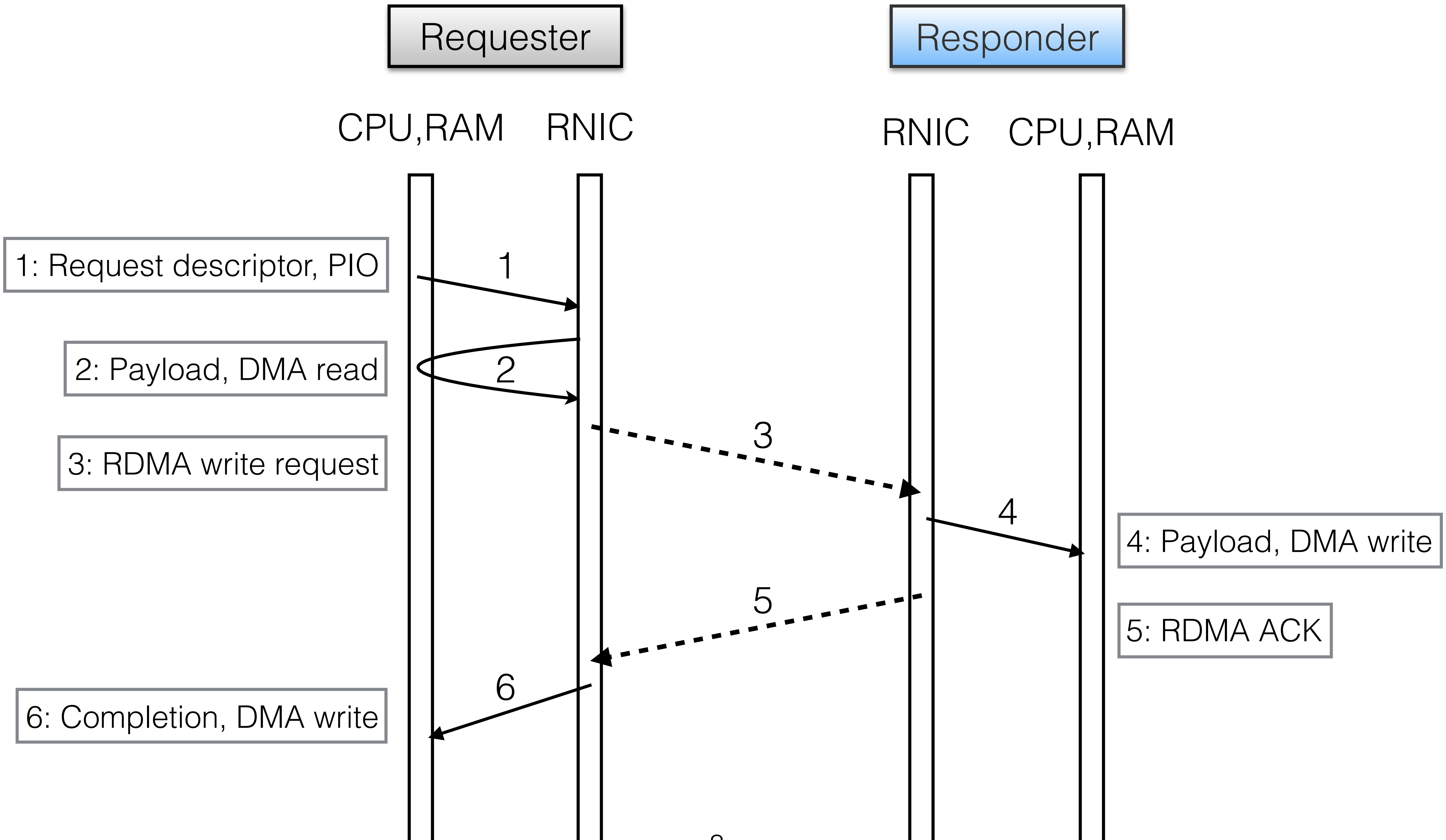
```
READ(local_buf, size, remote_addr)
```

RDMA write:

```
WRITE(local_buf, size, remote_addr)
```

⋮

Life of a WRITE



Recent systems

Pilaf [ATC 2013]

FaRM-KV [NSDI 2014]: an example usage of FaRM

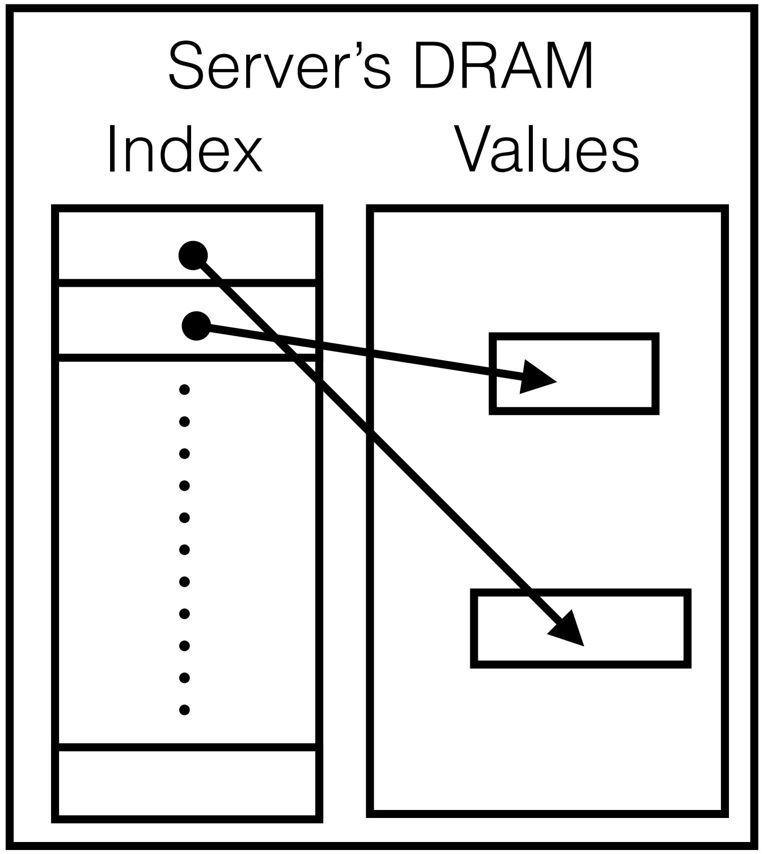
Approach: RDMA reads to access remote data structures

Reason: the allure of CPU bypass

The price of CPU bypass

Key-Value stores have an inherent level of indirection.

An index maps a keys to address. Values are stored separately.



At least 2 RDMA reads required:

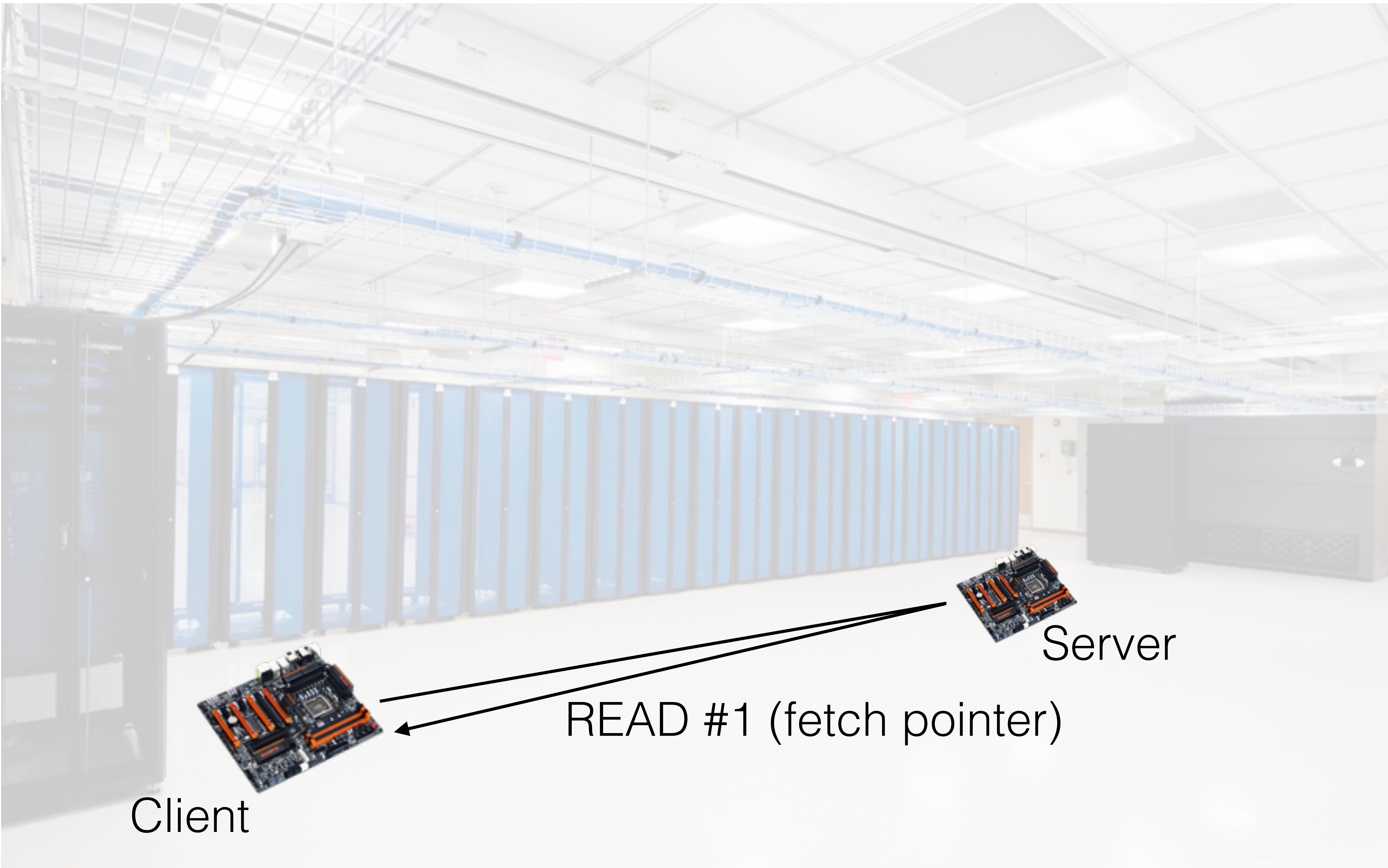
≧ 1 to fetch address

1 to fetch value

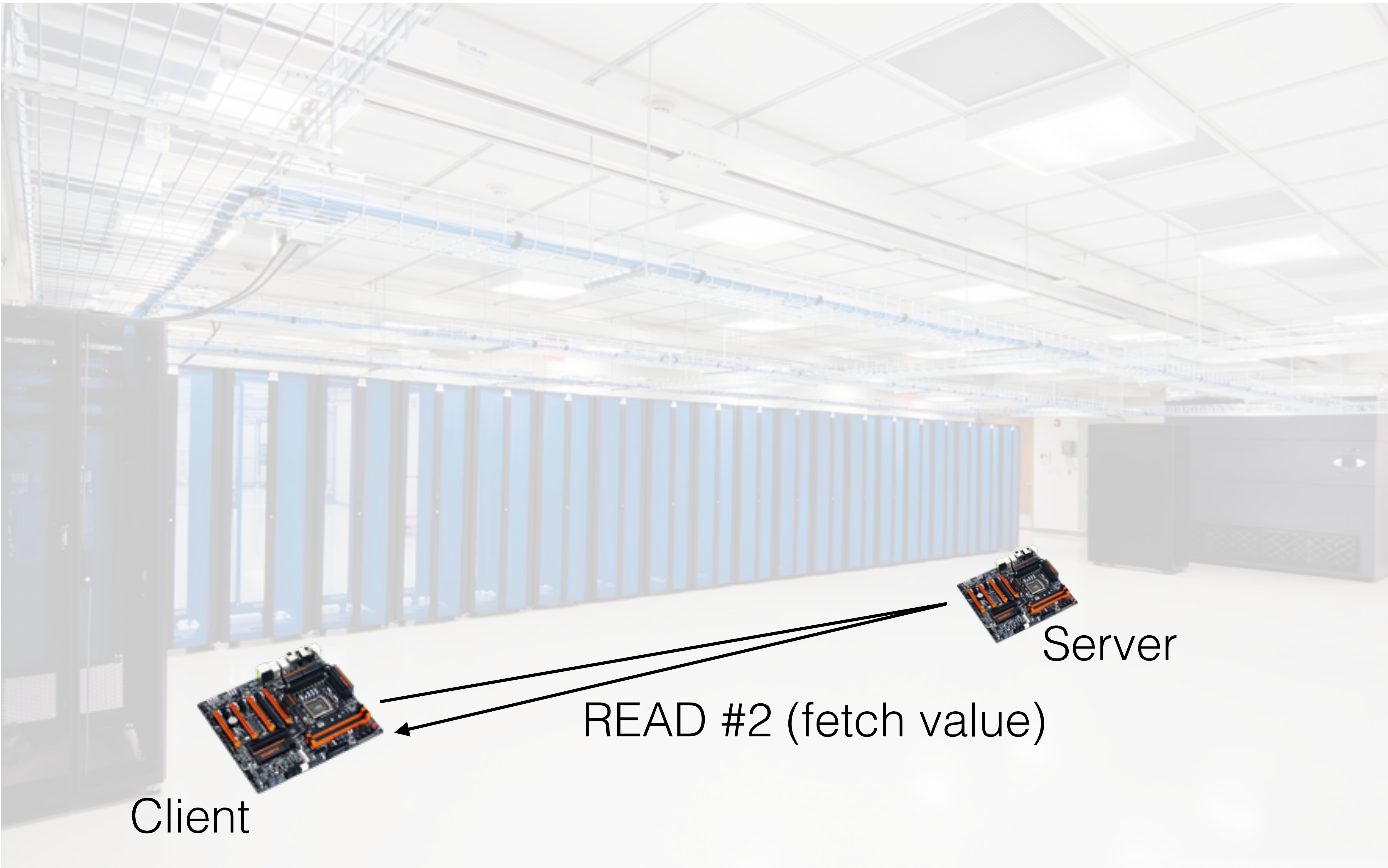
Not true if value is in index

The price of CPU bypass

The price of CPU bypass



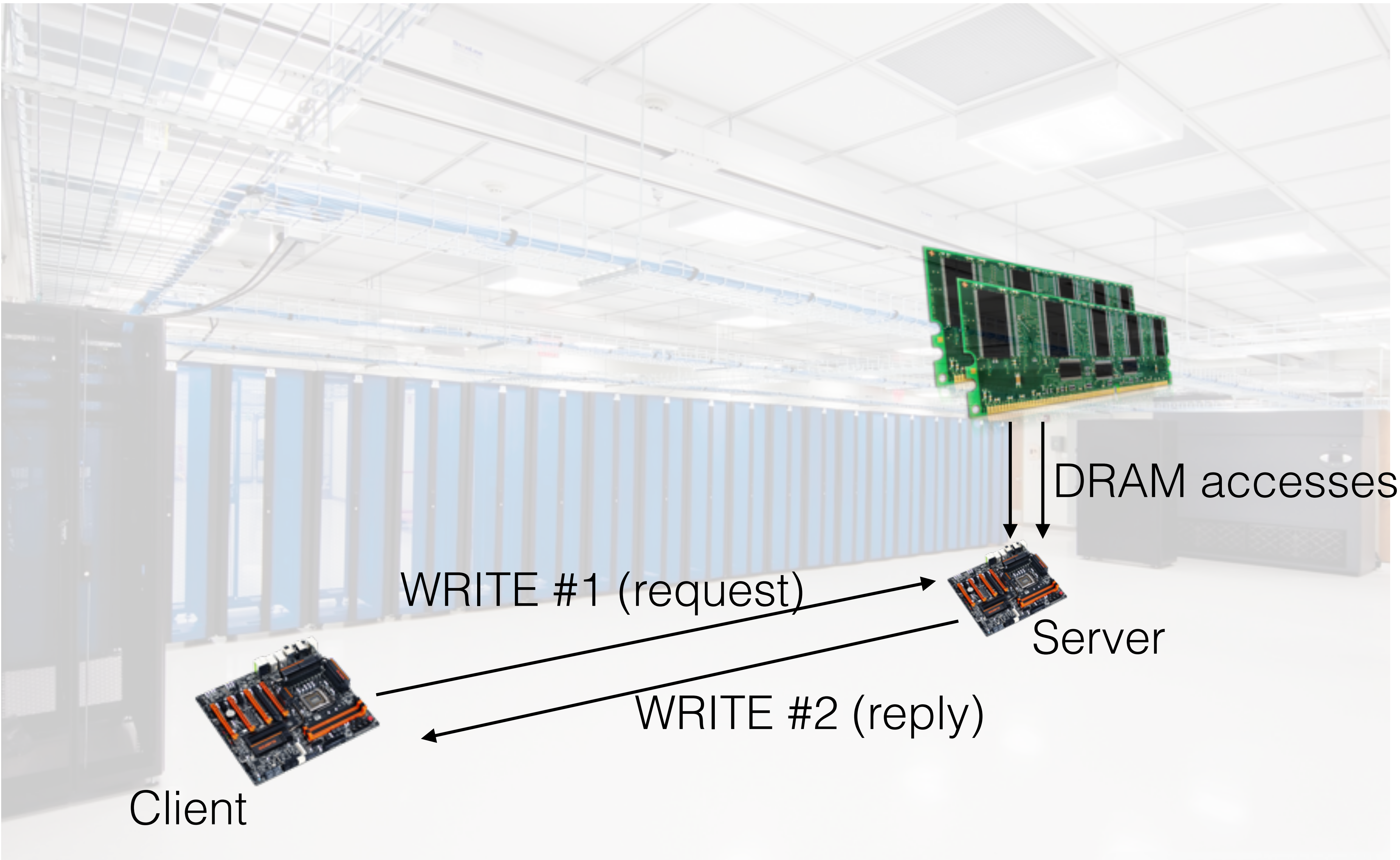
The price of CPU bypass



Our approach

Goal	Main ideas
#1: Use a single round trip	Request-reply with server CPU involvement + WRITEs faster than READs
#2. Increase throughput	Low level verbs optimizations
#3. Improve scalability	Use datagram transport

#1: Use a single round trip



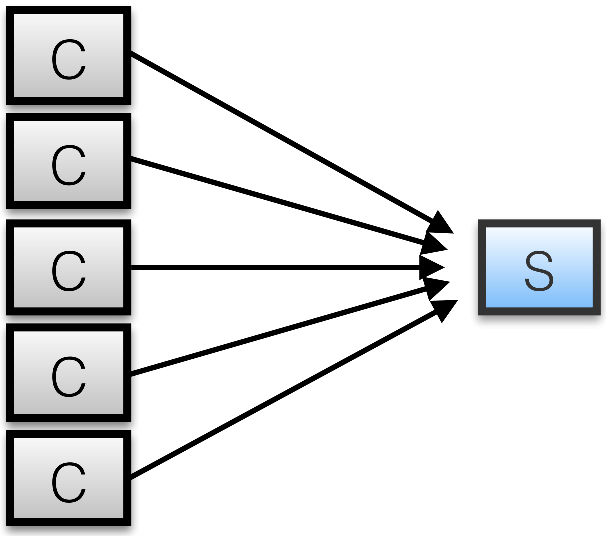
#1: Use a single round trip

Operation	Round Trips	Operations at server's RNIC
READ-based GET	**2+**	2+ RDMA reads
HERD GET	**1**	2 RDMA writes

✓ Lower latency

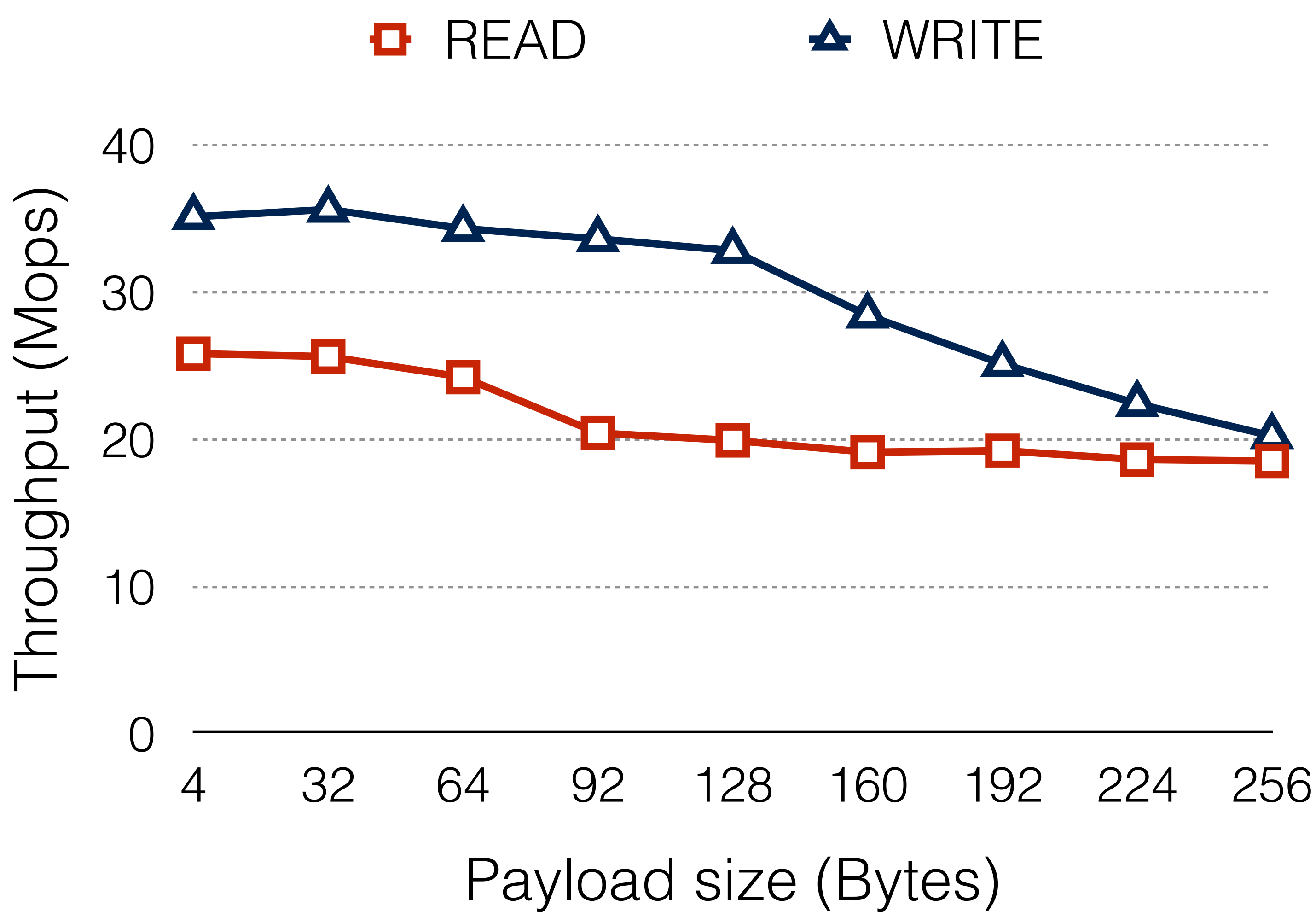
? High throughput

RDMA WRITEs faster than READs



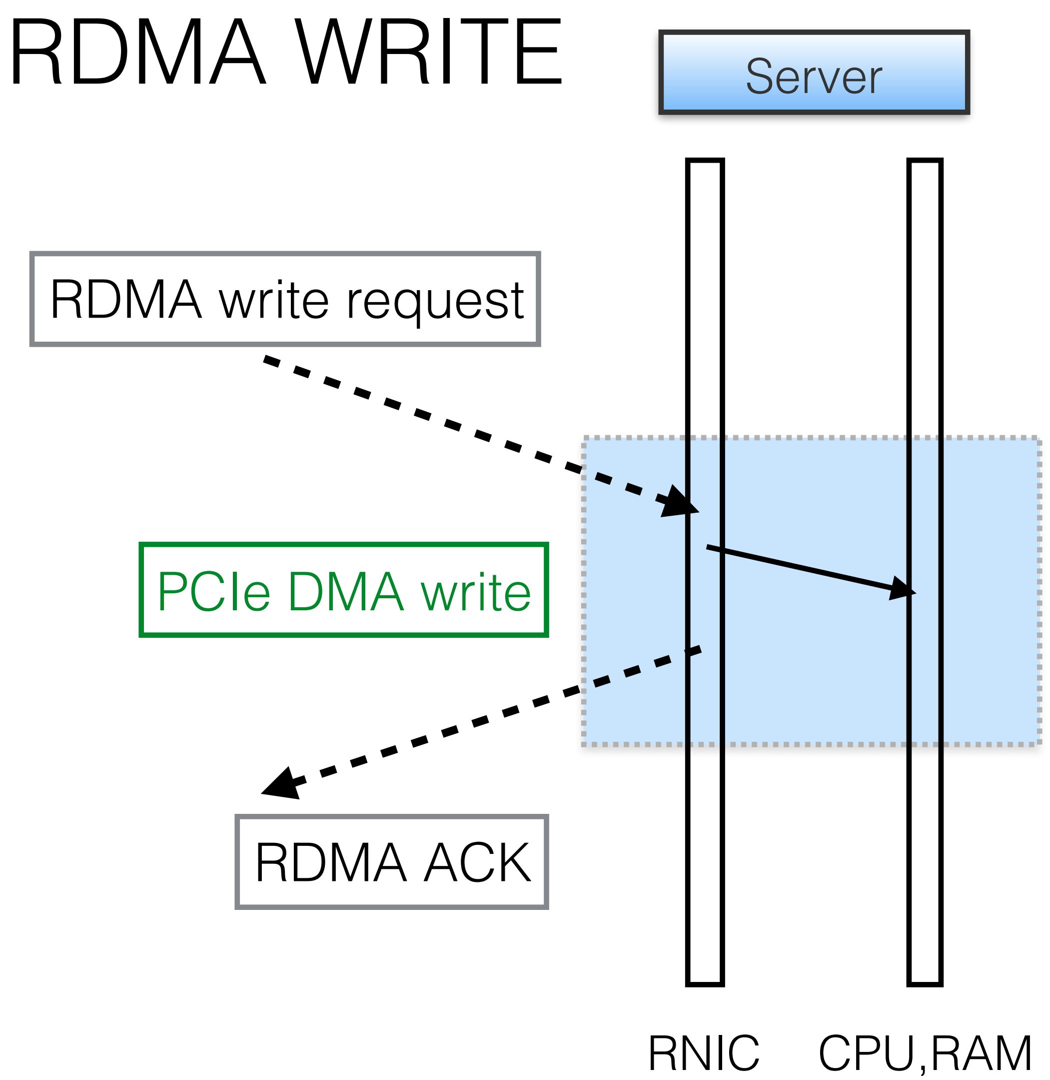
Setup: Apt Cluster

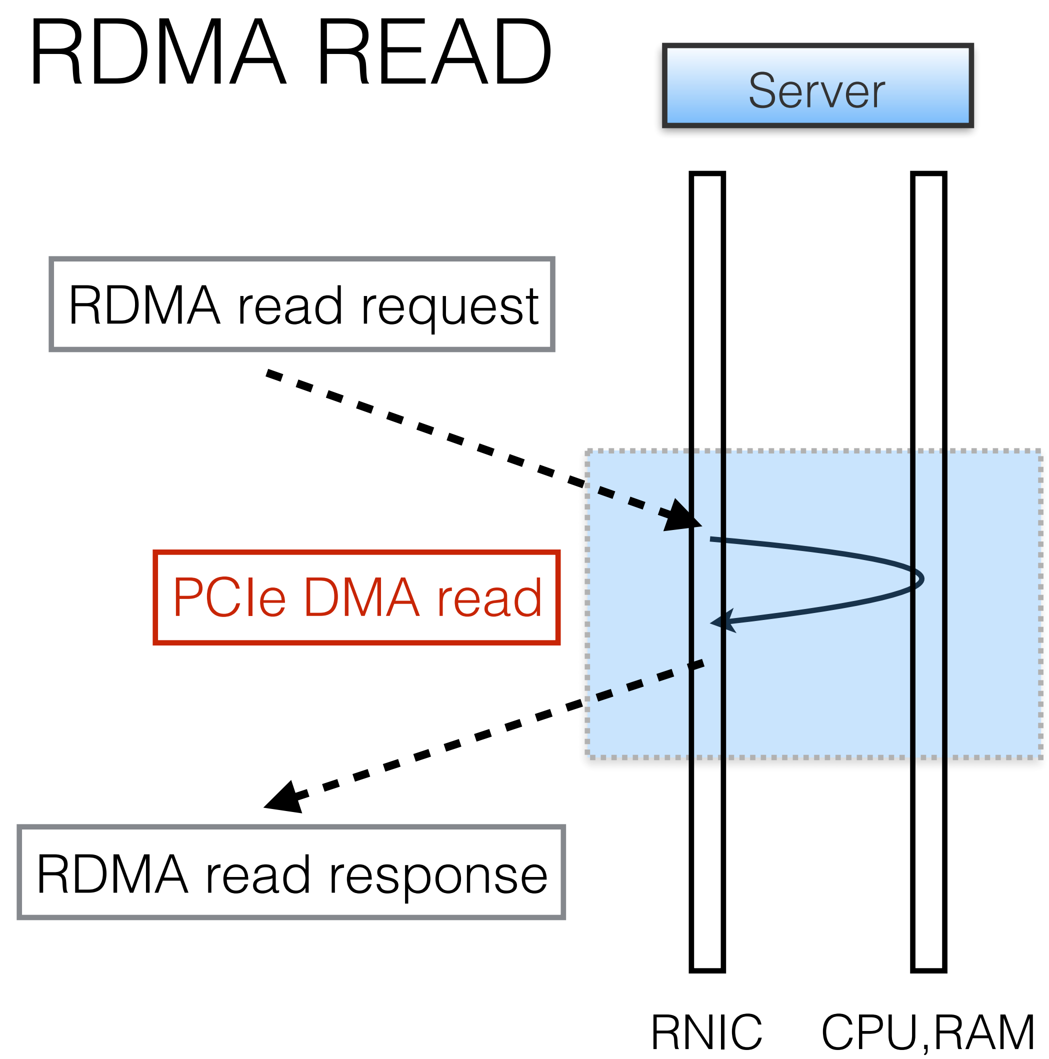
192 nodes, 56 Gbps IB



RDMA WRITEs faster than READs

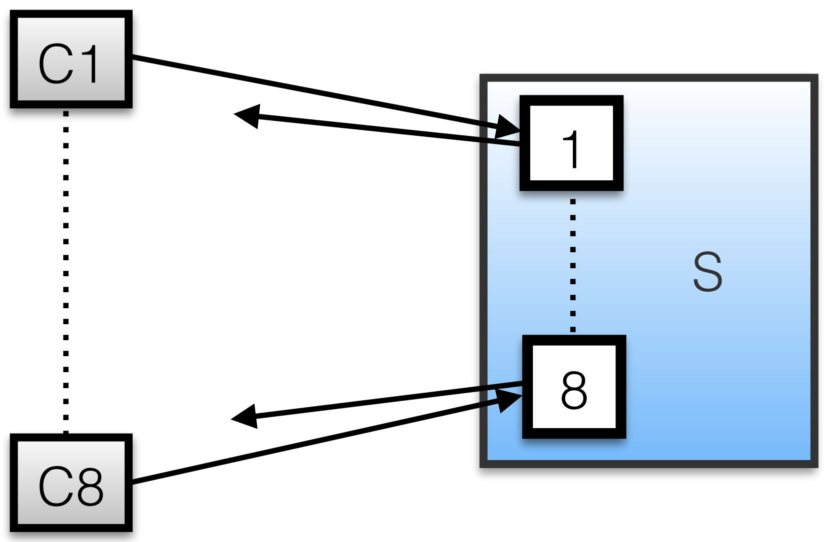
Reason: PCIe writes faster than PCIe reads





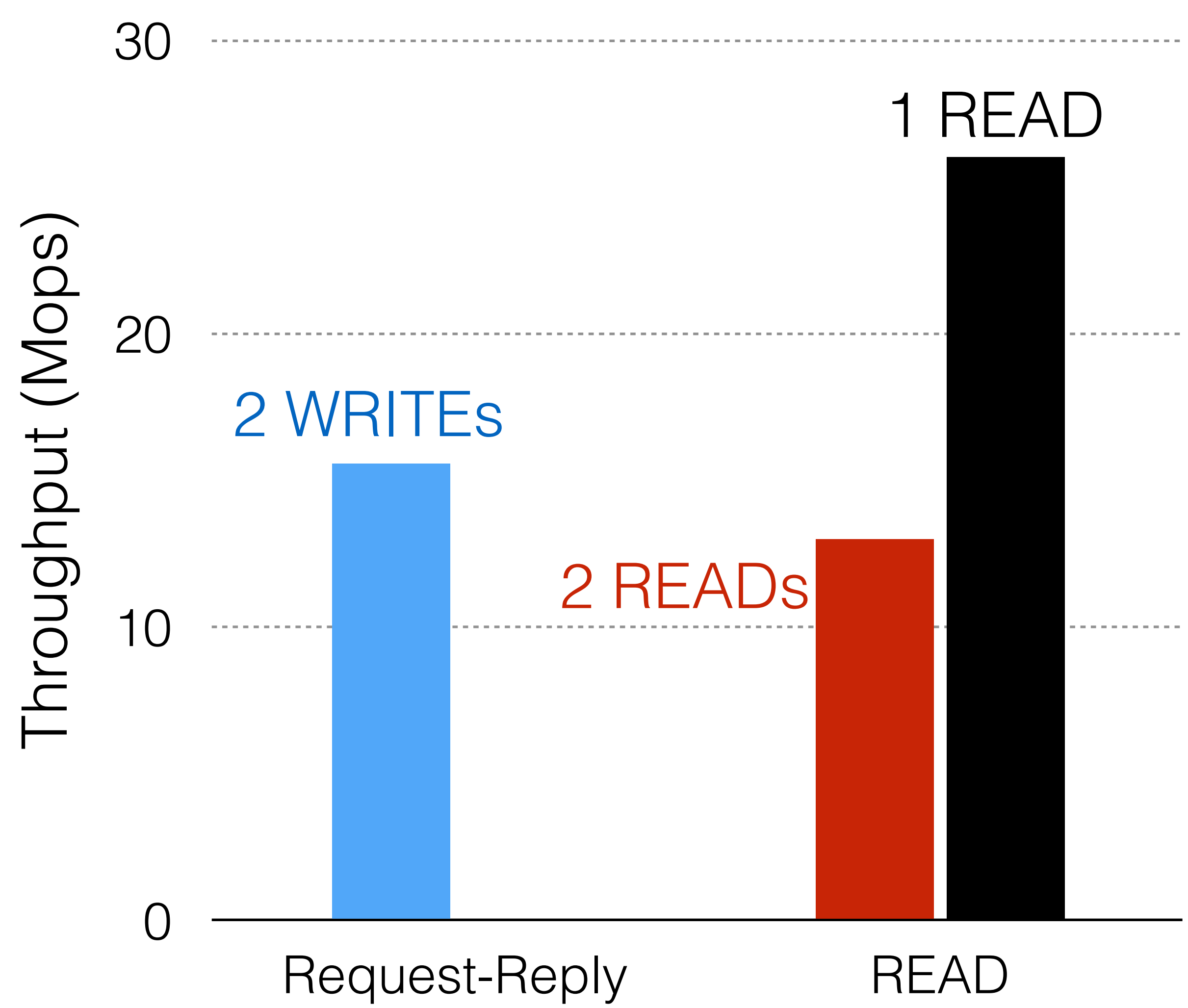
High-speed request-reply

Request-reply throughput:



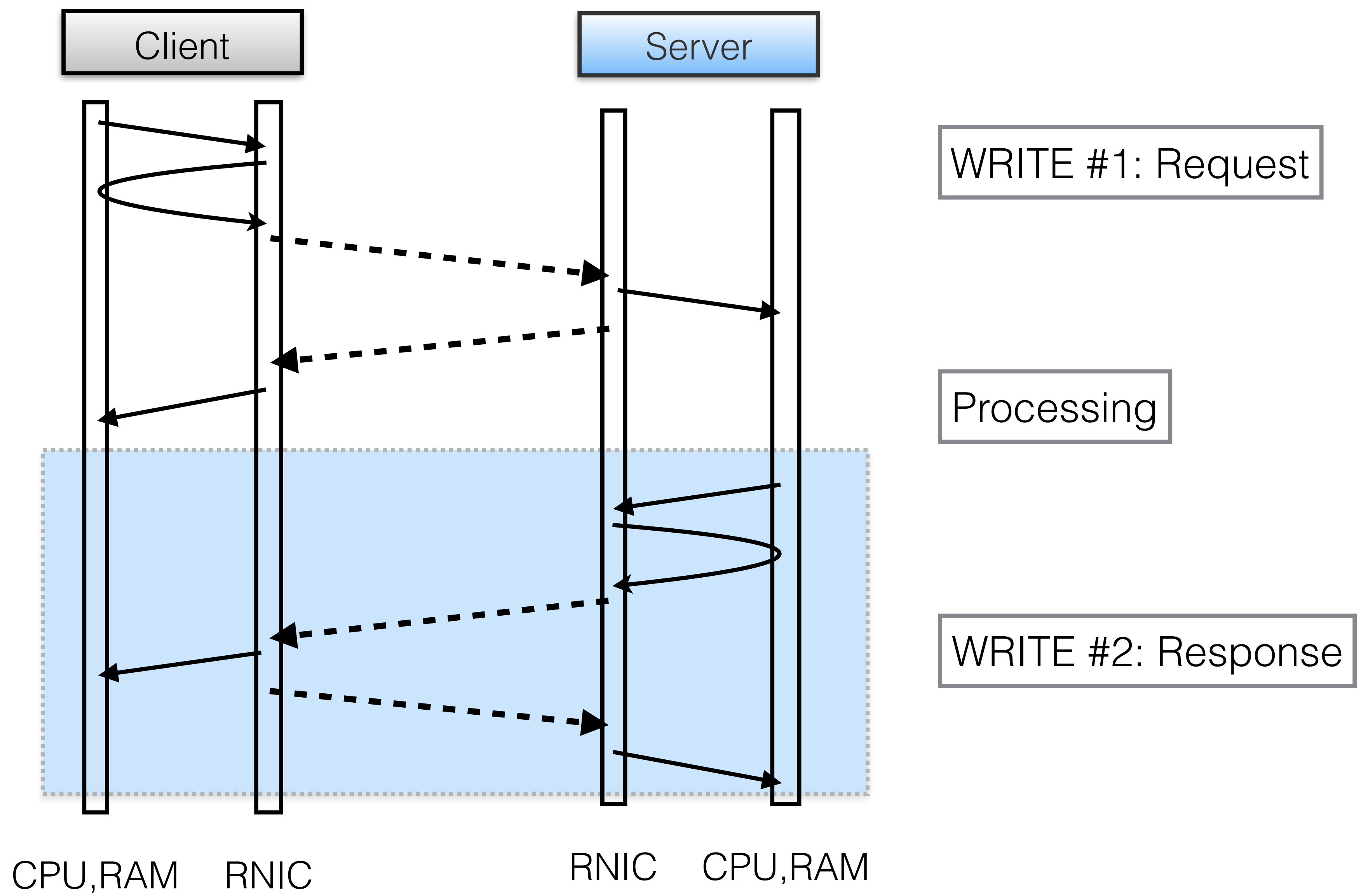
Setup: one-to-one client-server communication

32 byte payloads



#2: Increase throughput

Simple request-reply:

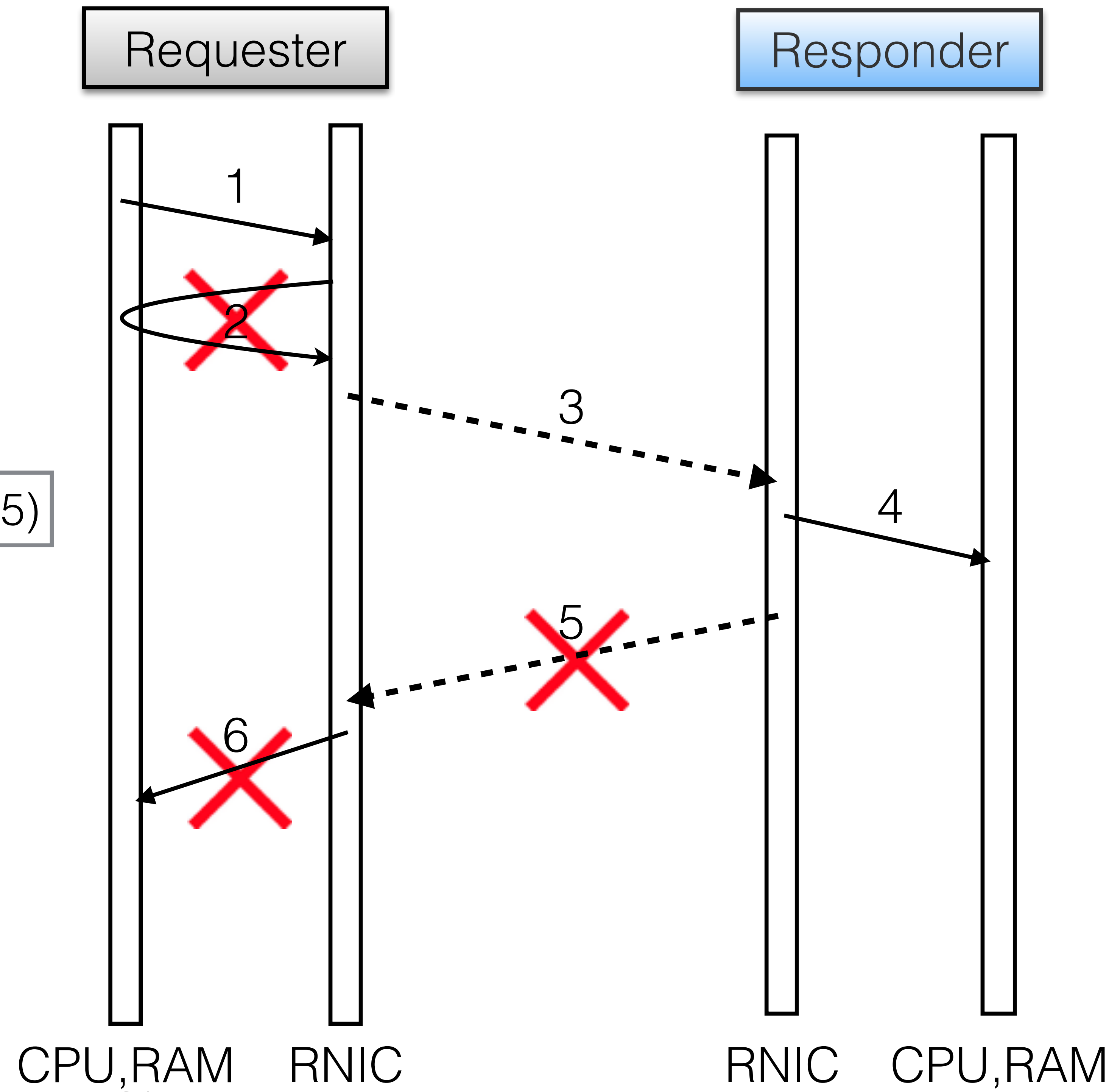


Optimize WRITEs

+inlining: encapsulate payload in request descriptor (2→1)

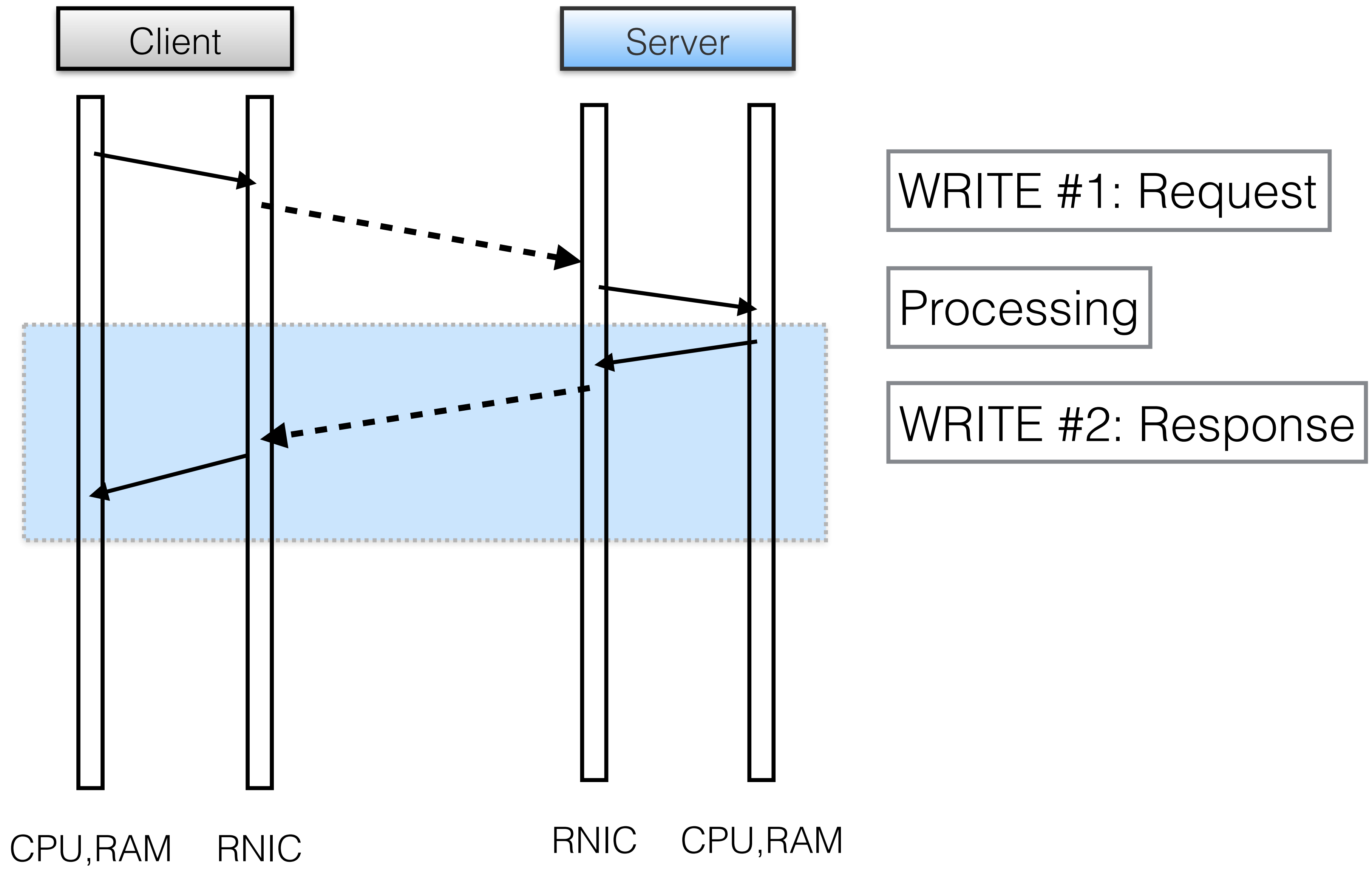
+unreliable: use unreliable transport (- 5)

+unsignaled: don't ask for request completions (- 6)

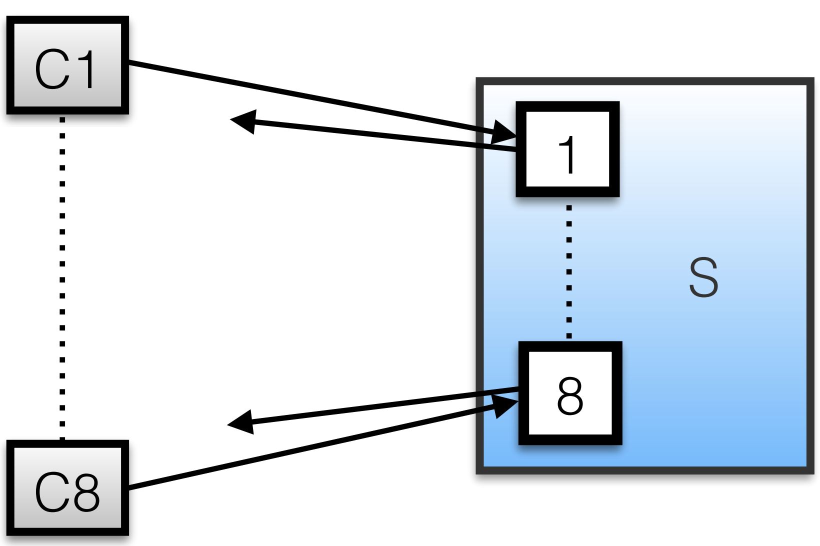


#2: Increase throughput

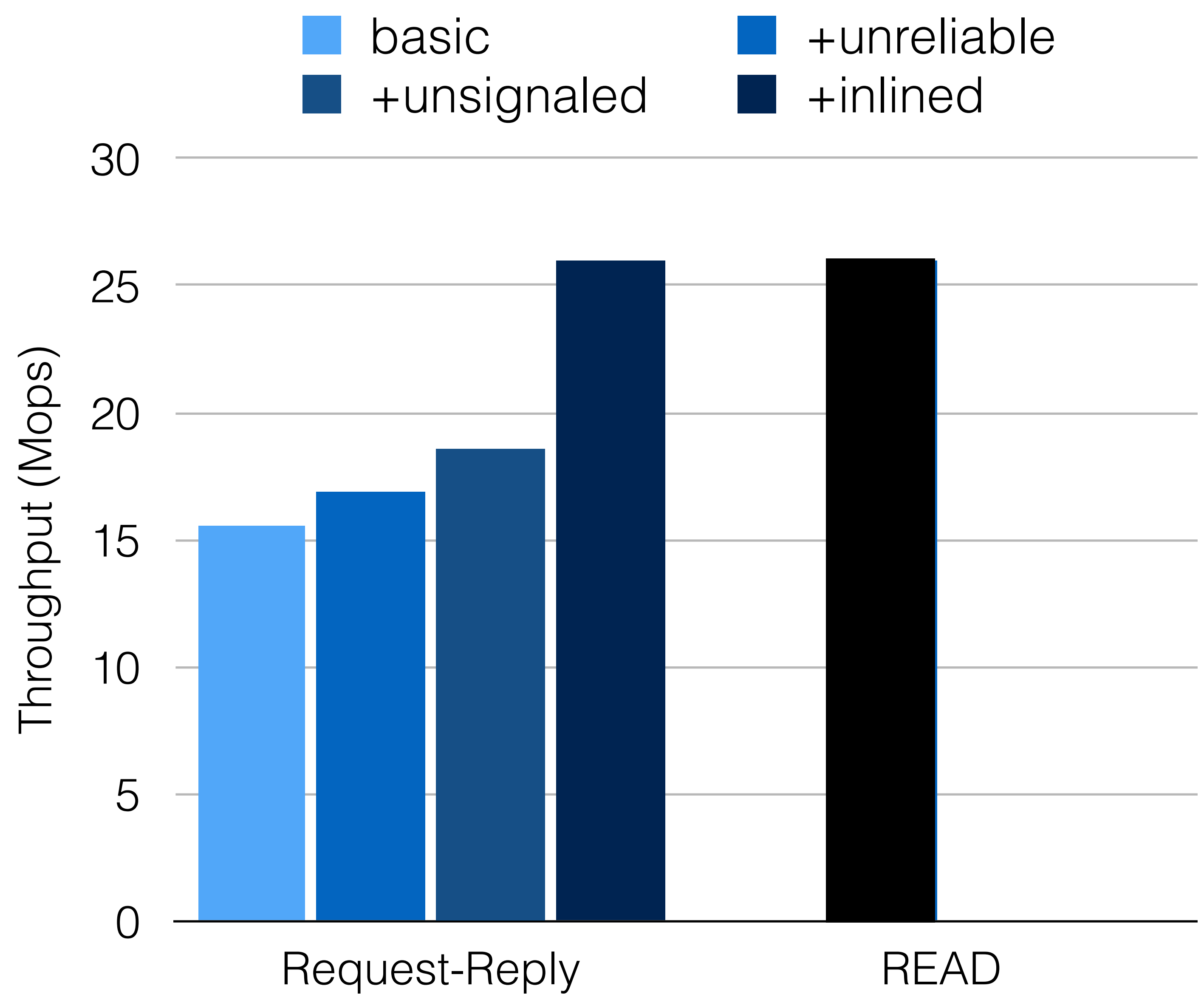
Optimized request-reply:



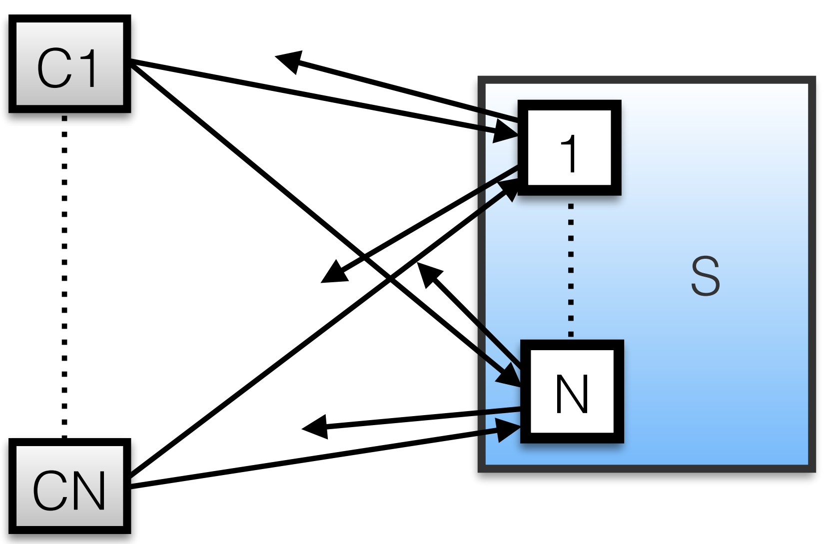
#2: Increase throughput



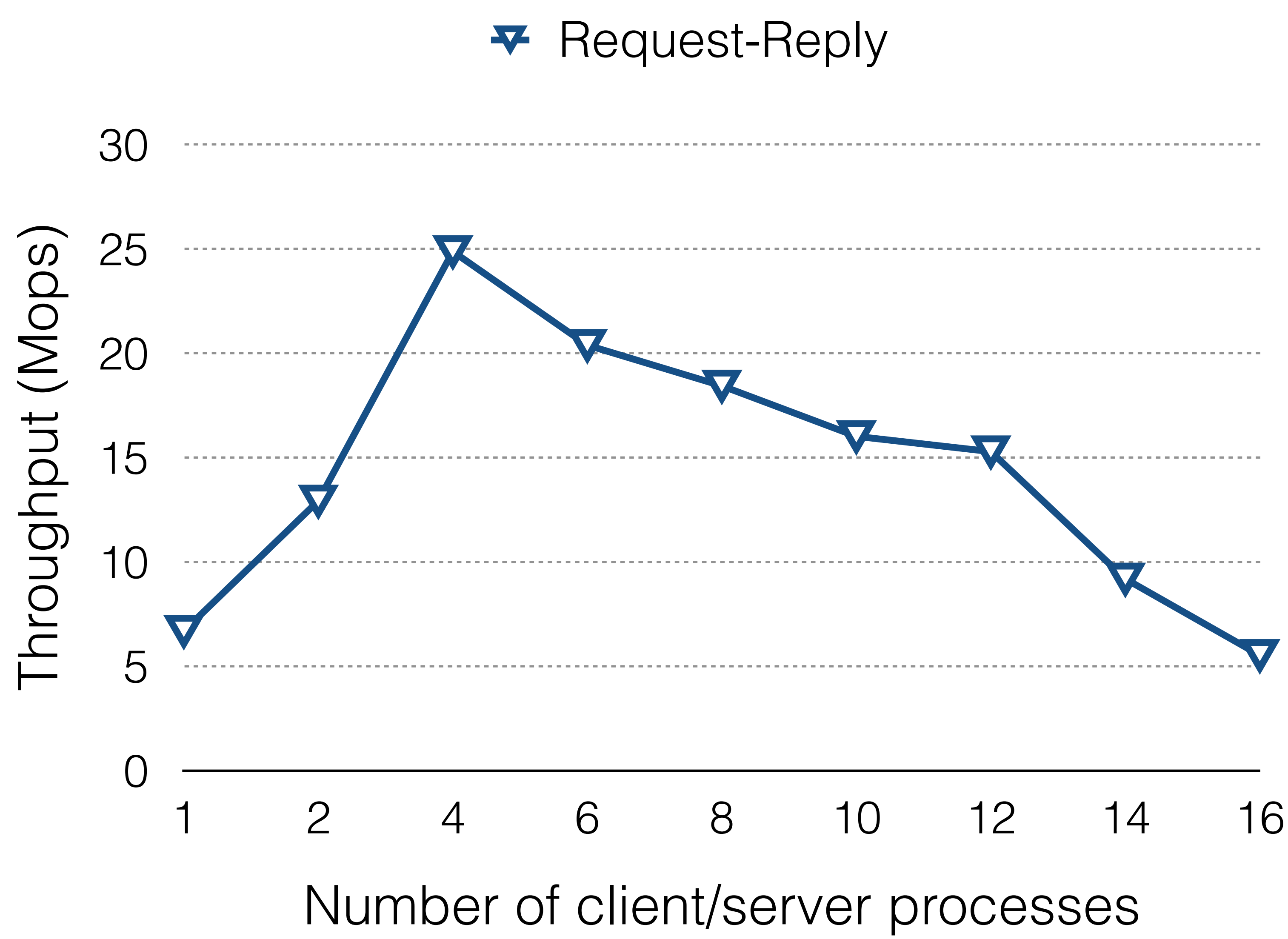
Setup: one-to-one client-server communication



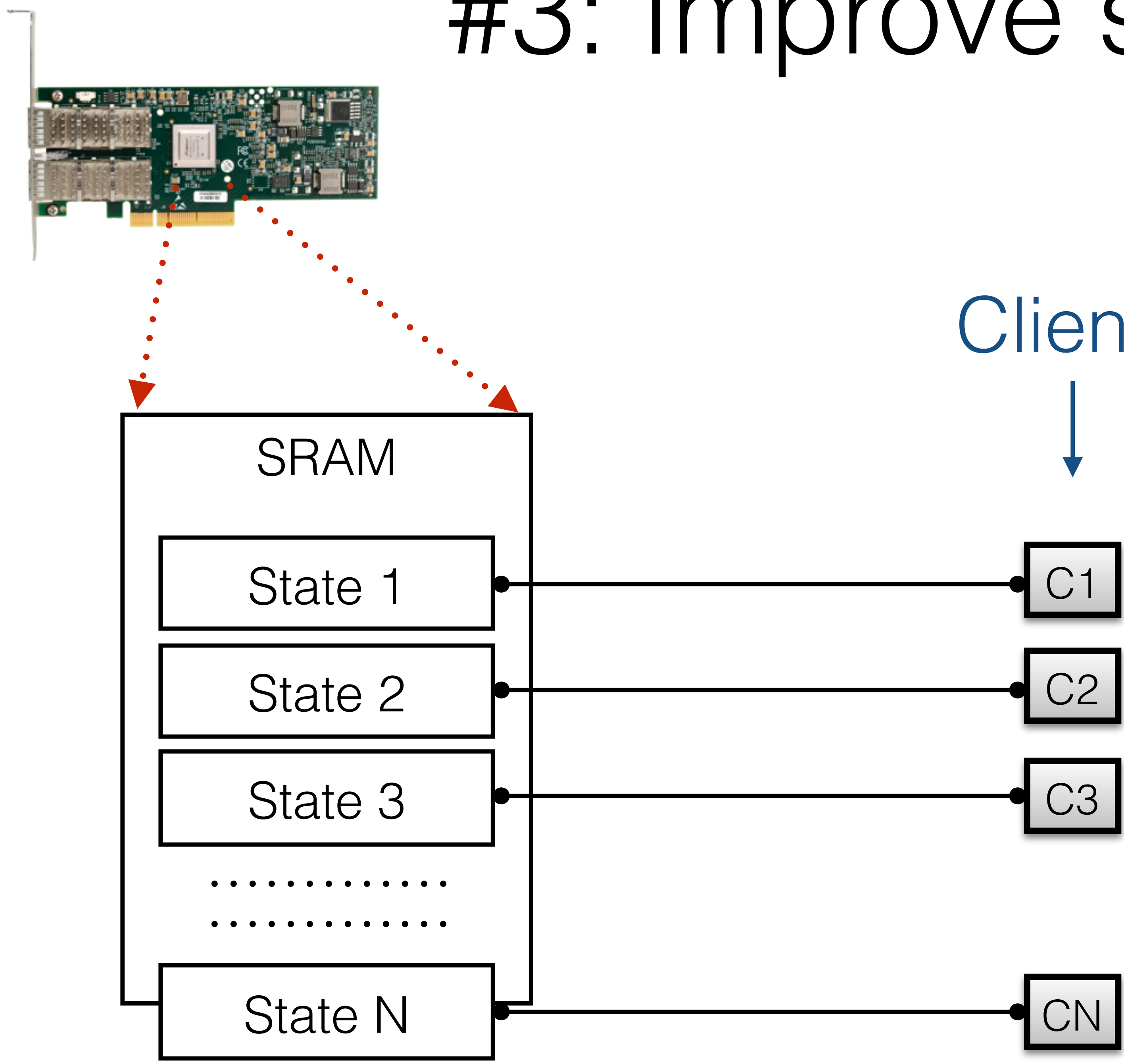
#3: Improve scalability



Setup

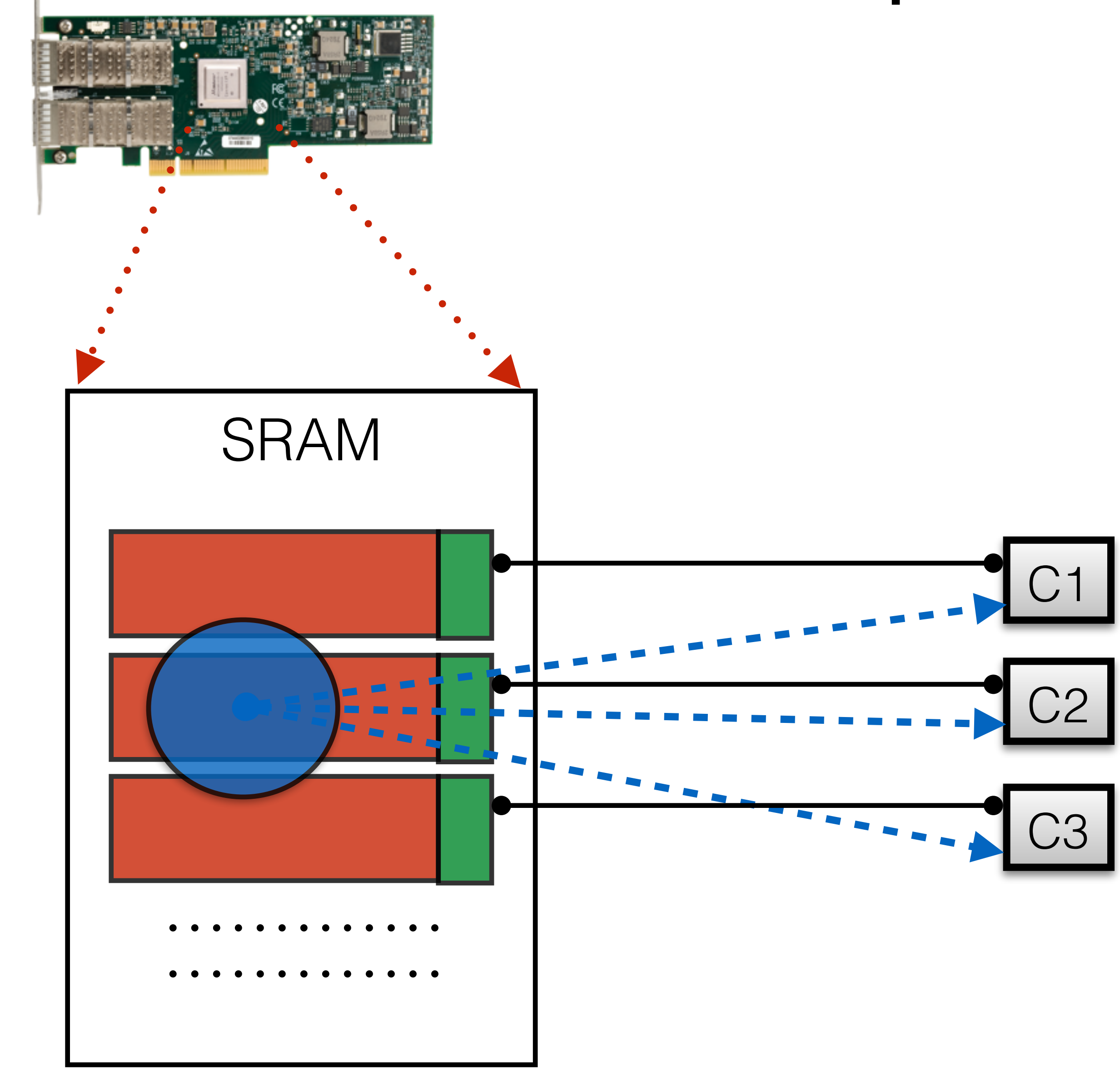


#3: Improve scalability



||state|| > SRAM

#3: Improve scalability



Inbound scalability ≫ outbound *because* inbound state ( ) ≪ outbound ( )

↓

Use datagram for outbound replies

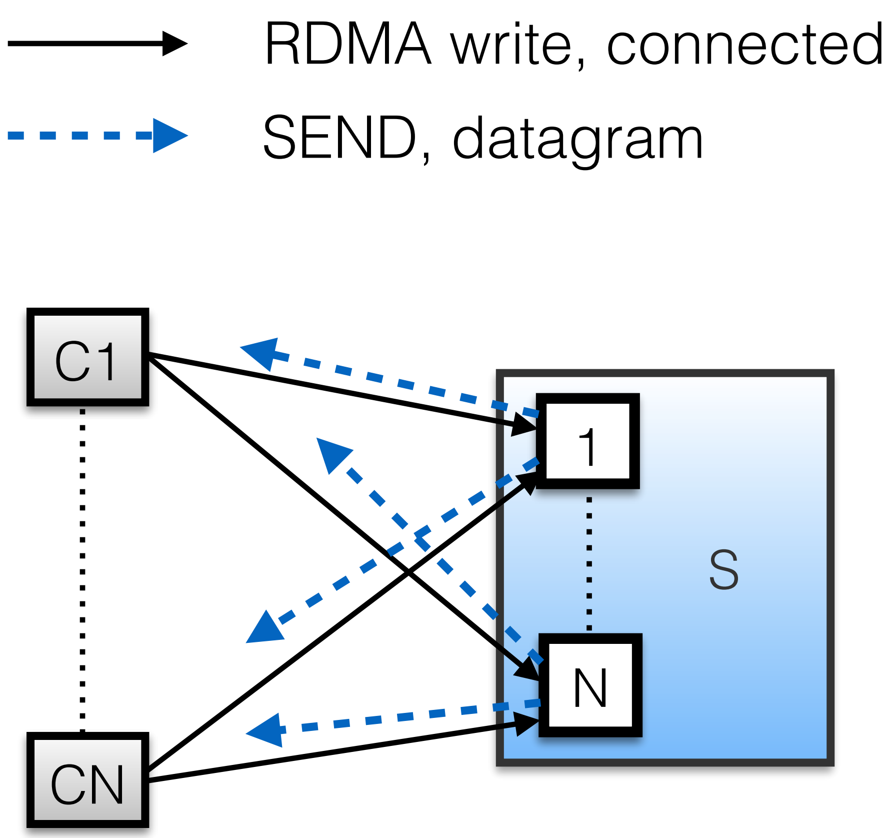
↓

Datagram only supports SEND/RECV.
SEND/RECV is slow.

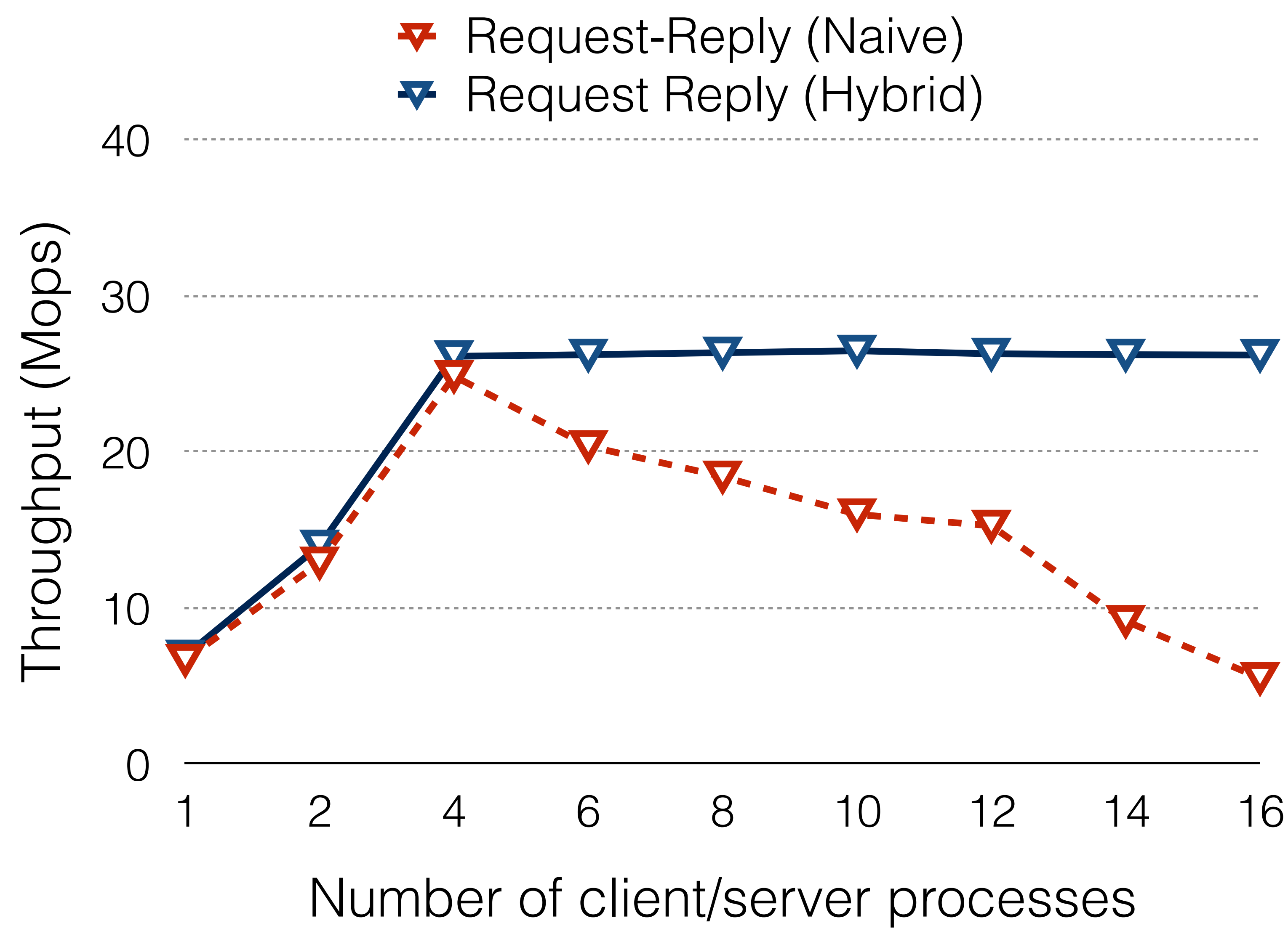
↓

SEND/RECV is slow only at the receiver

Scalable request-reply



Setup



Evaluation

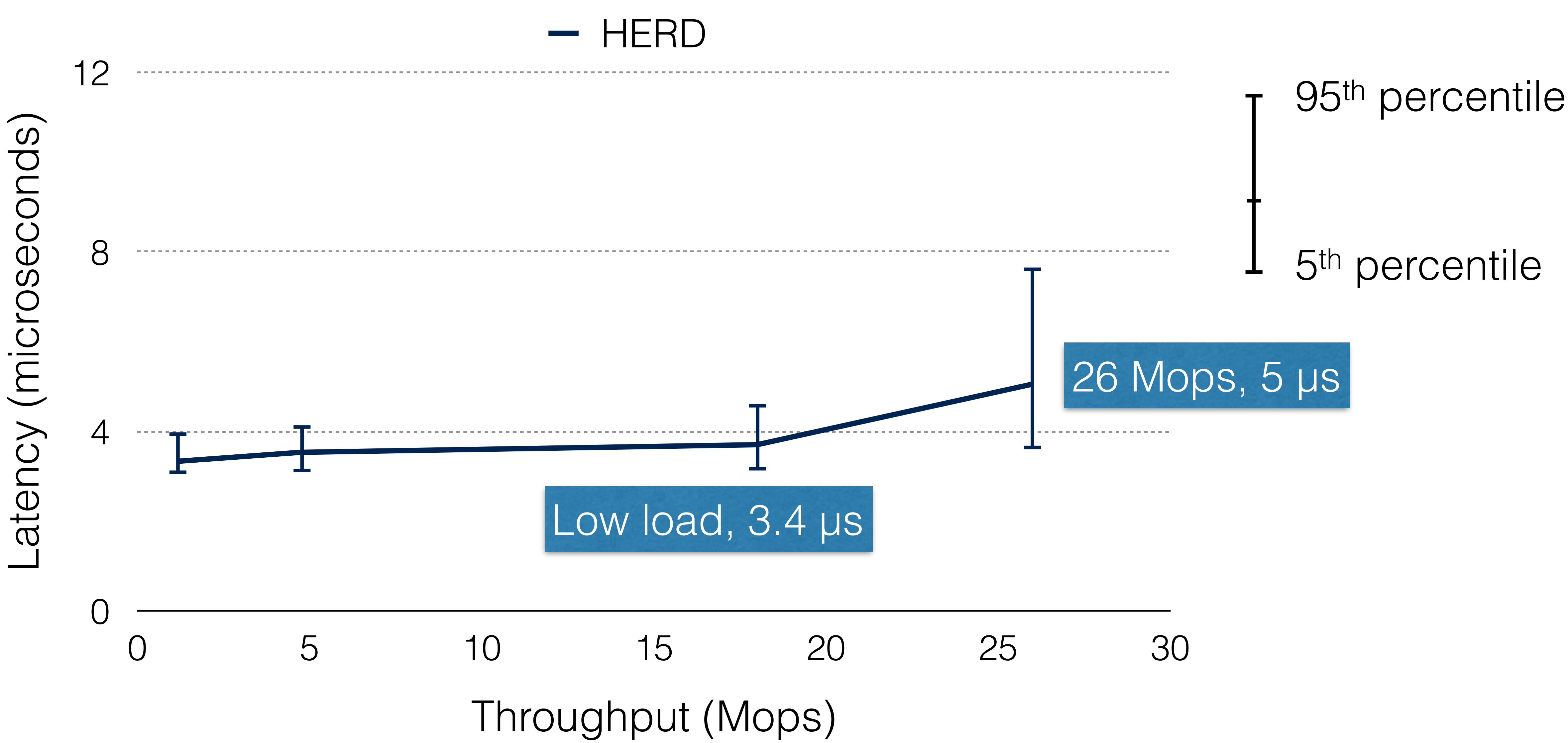
HERD = Request-Reply + MICA [NSDI 2014]

Compare against emulated versions of Pilaf and FaRM-KV

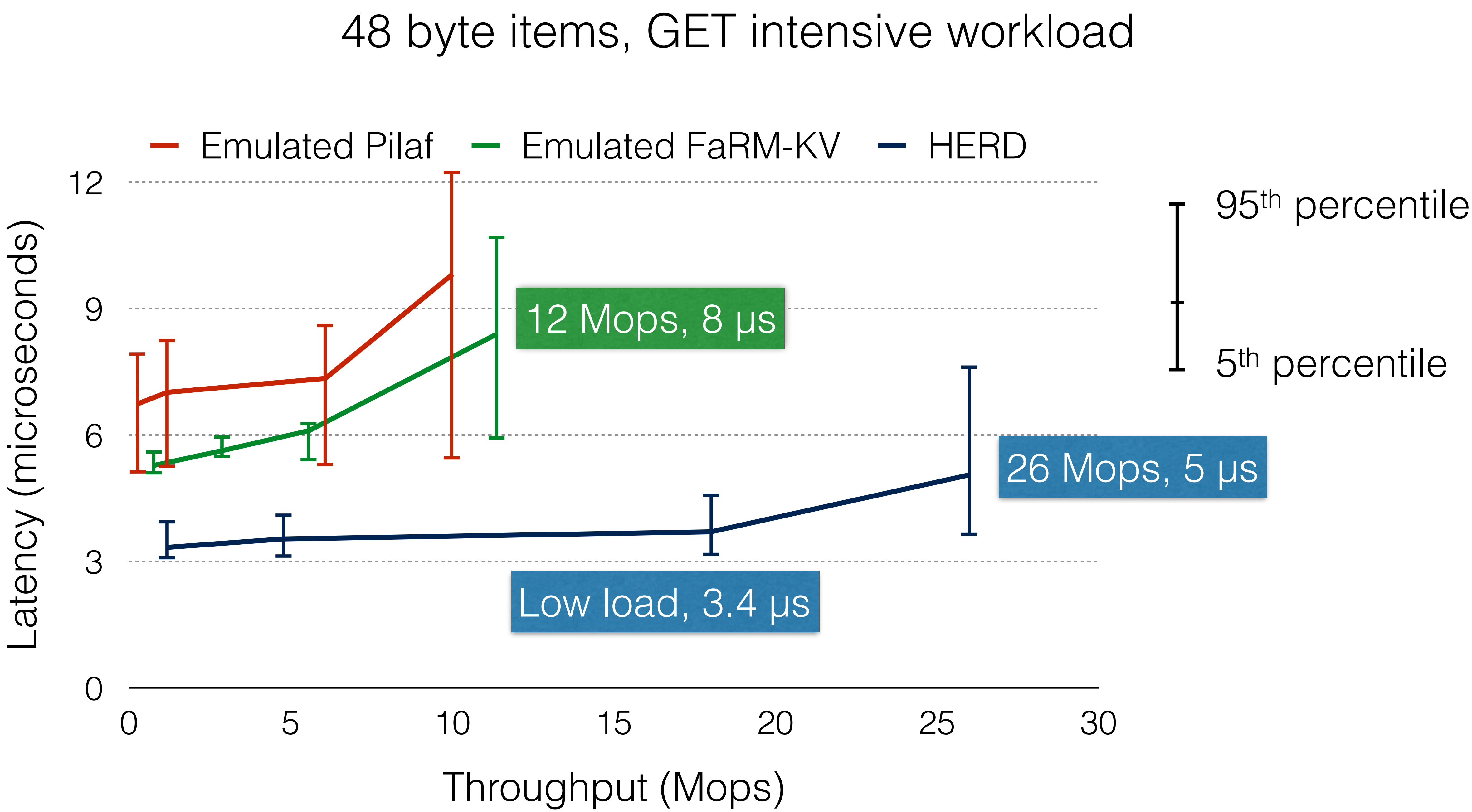
- No datastore
- Focus on maximum performance achievable

Latency vs throughput

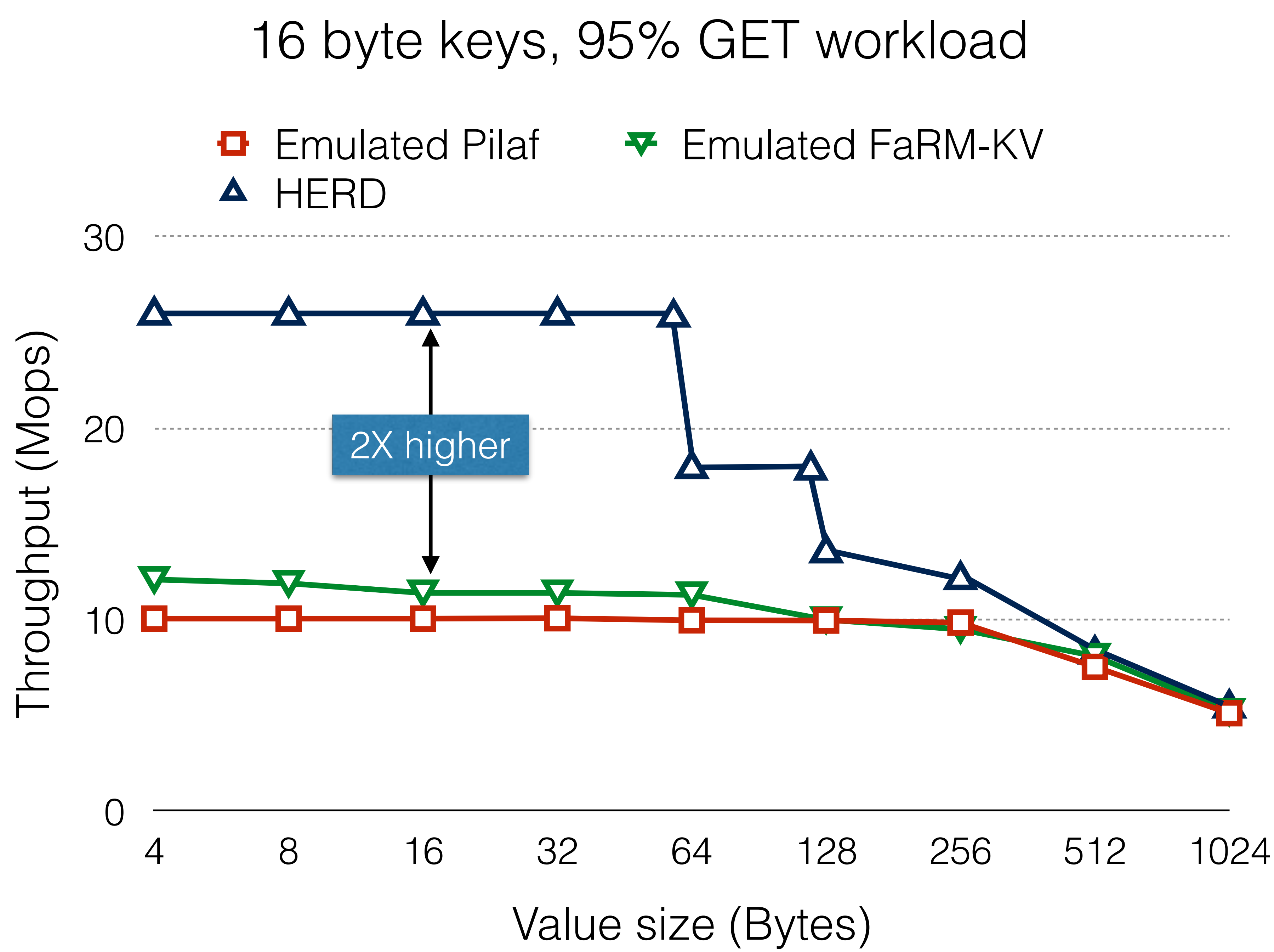
48 byte items, GET intensive workload



Latency vs throughput



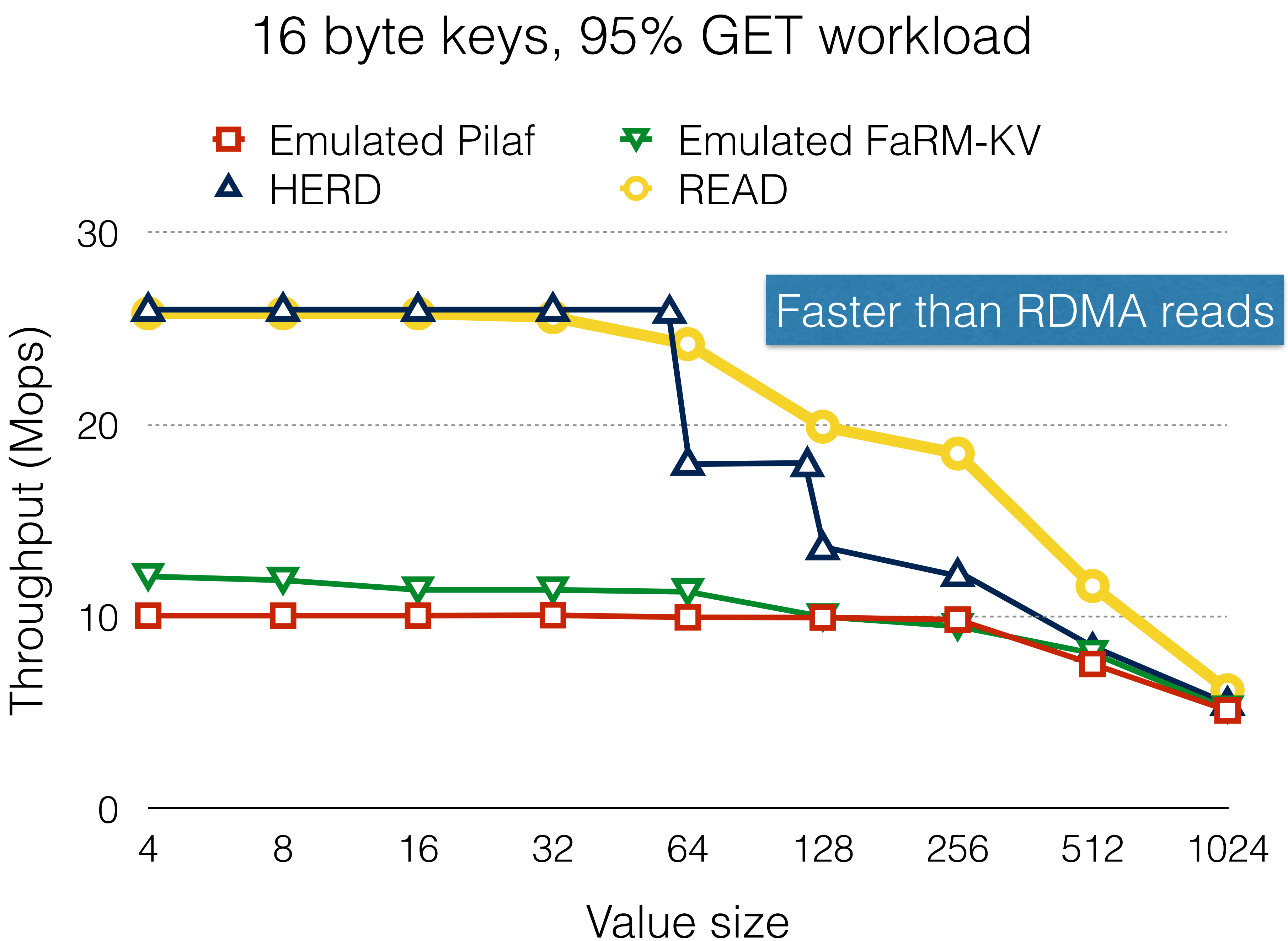
Throughput comparison



HERD

- Re-designing RDMA-based KV stores to use a single round trip
    - WRITEs outperform READs
    - Reduce PCIe and InfiniBand transactions
    - Embrace SEND/RECV
- Code is online: https://github.com/efficient/HERD

Throughput comparison



Throughput comparison