Scalable RDMA RPC on Reliable Connection with Efficient Resource Sharing

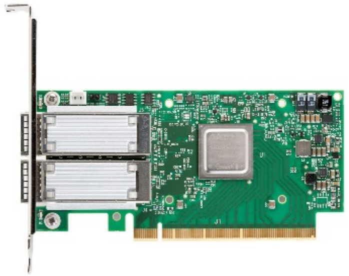
Youmin Chen, Youyou Lu, Jiwu Shu

Tsinghua University

http://storage.cs.tsinghua.edu.cn

Remote Direct Memory Access (RDMA)

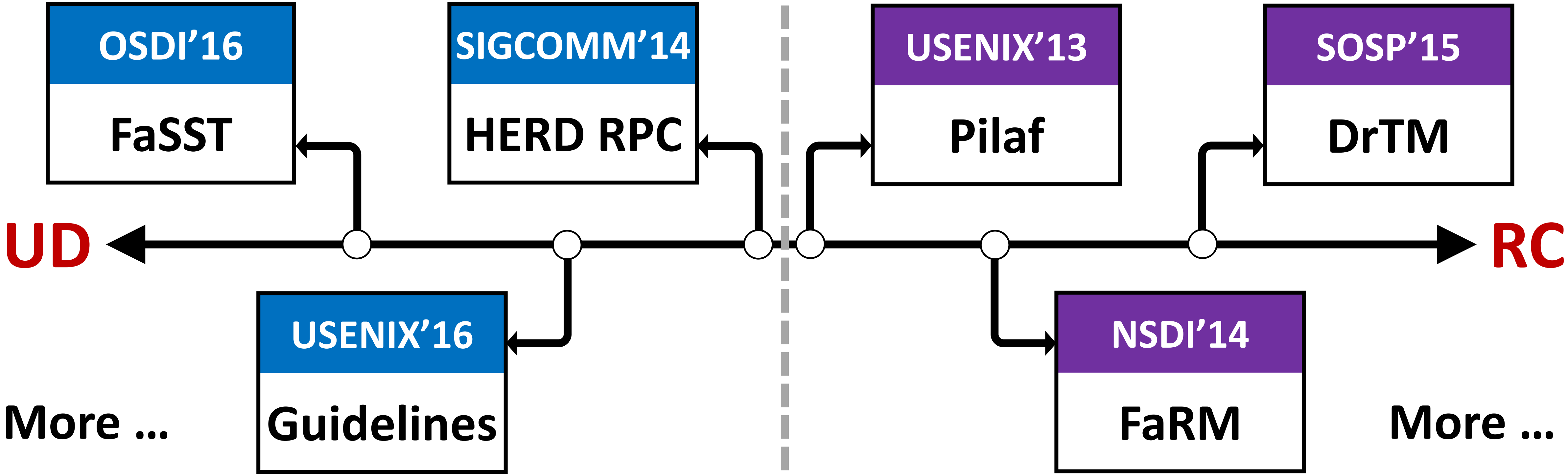
- **Device-Level Networking**
  - **Low** latency (< 1us)
  - **High** bandwidth
    CX-3: 56Gbps, CX-4/5: 100Gbps, CX-6:200Gbps



- **One-sided Verbs**
  - **Bypassing** remote CPUs
  - Directly **Read/Write/CAS** remote memories

Source Image from Mellanox&Chelsio

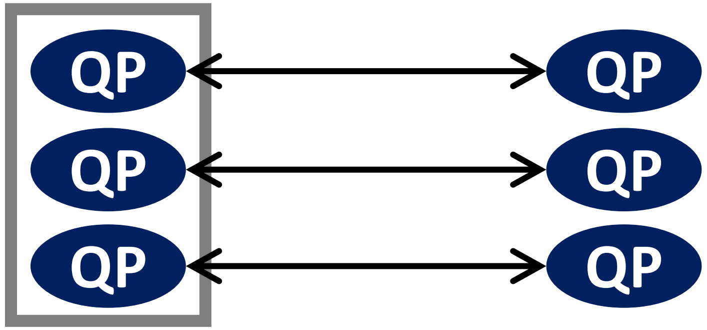
A long debate of whether using RC or UD



A long debate of whether using RC or UD

Reliable Connection (RC)

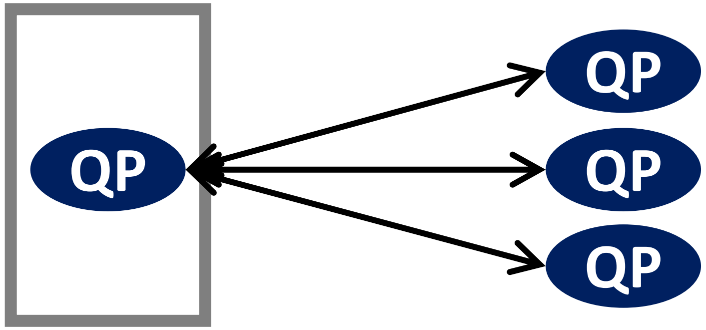
One-to-one paradigm



- Offloading with one-sided verbs
- Higher performance
- Reliable
- Flexible-sized transferring
- **Hard to scale (explain latter)**

Unreliable Datagram (UD)

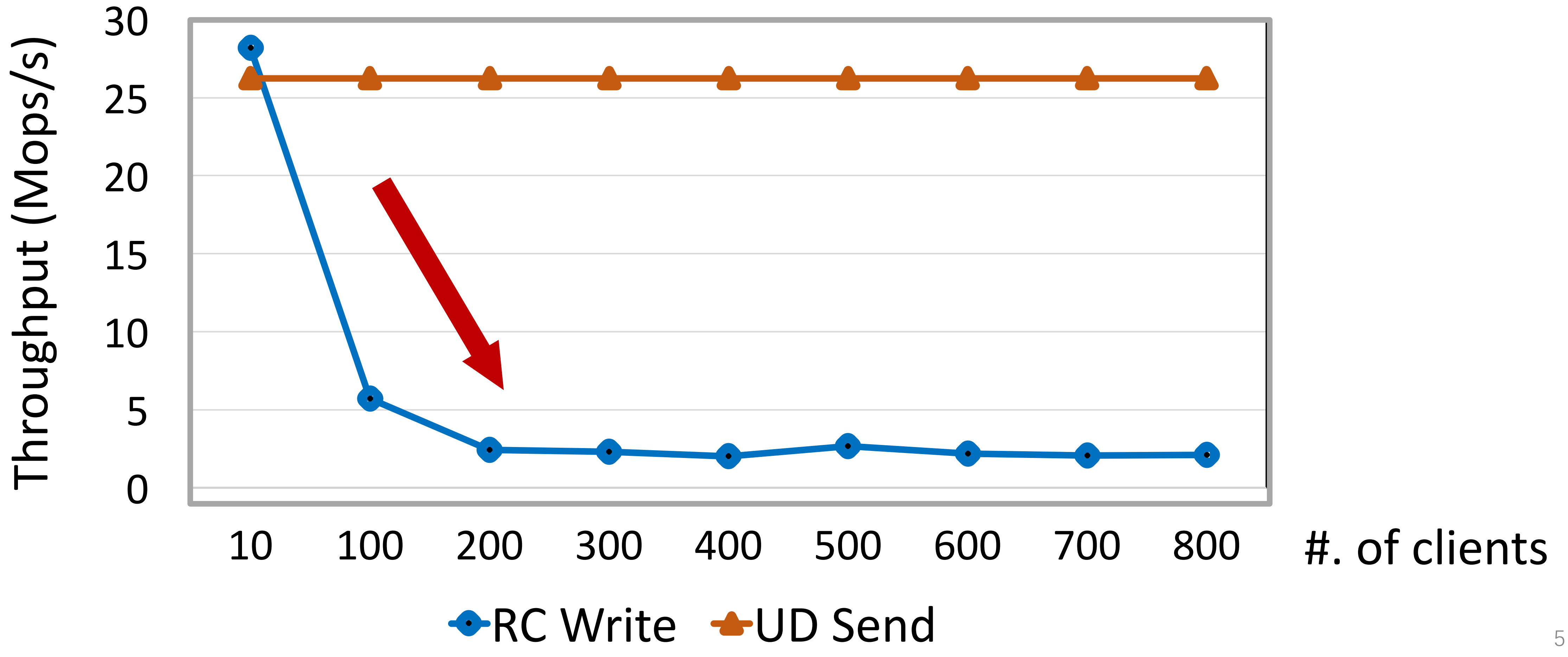
One-to-many paradigm



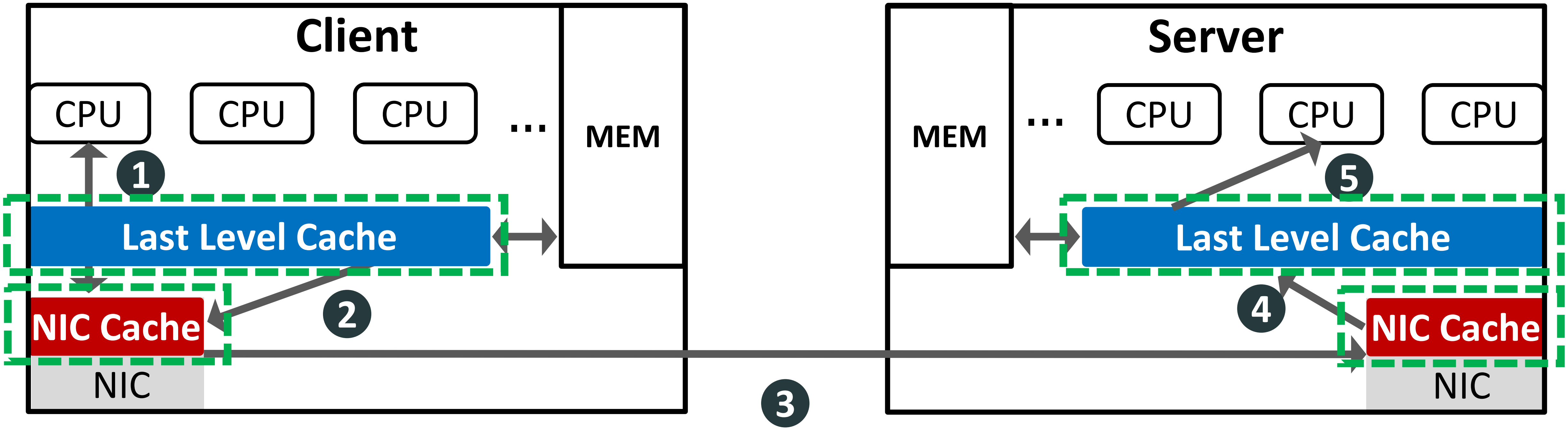
- Unreliable (risk of packet loss, out-of-order, etc.)
- Cannot support one-sided verbs
- MTU is only 4KB
- **Good scalability**

RC hard to scale!

- MCX353A **ConnectX-3** FDR HCA (single port)
- **1** server node send verbs to **11** client nodes



Why is RC hard to scale?



1 Memory-Mapped I/O

2 PCIe DMA Read

3 Packet Sending

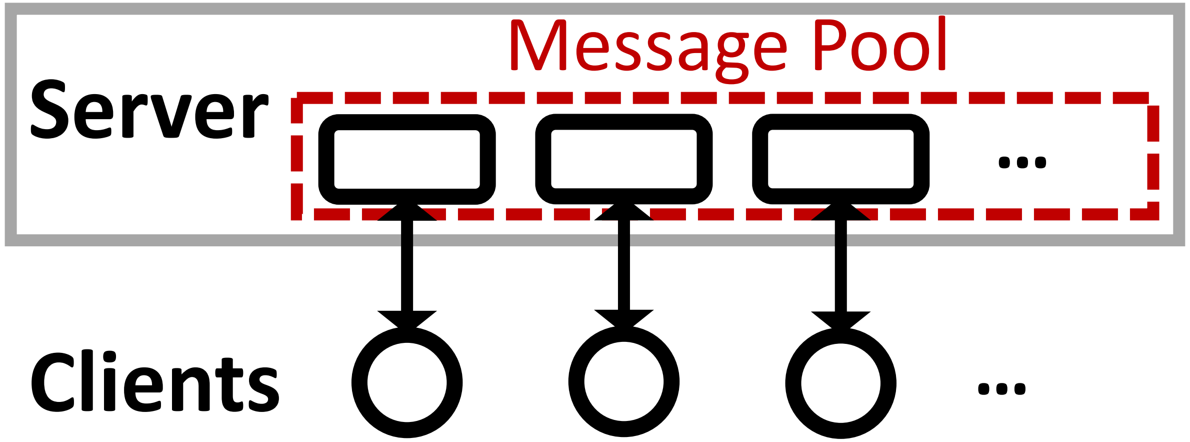
4 PCIe DMA Write (DDIO enabled)

5 CPU Polls Message

Why is RC hard to scale?

Two types of Resource Contention:

- **NIC Cache**[1]
  - Mapping table
  - QP states
  - Work queue elements
- **CPU Cache**
  - DDIO writes data to LLC
  - Only **10%** reserved for DDIO



With **RC**, the size of cached data is **proportional** to the number of clients!

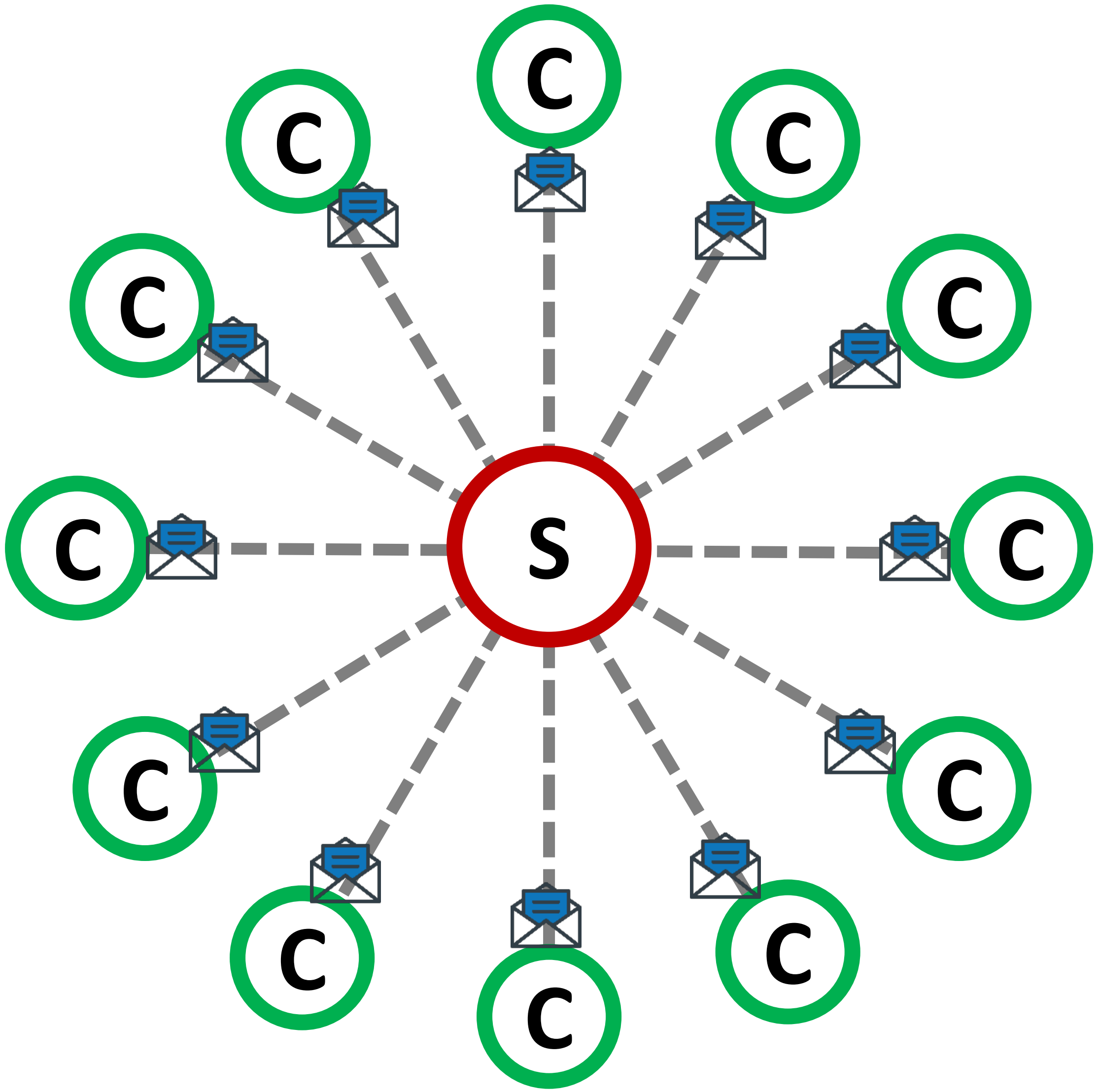
Our goal: how to make RC scalable

- **Focus on RPC primitive with RC write**
  - RPC is a good abstraction, widely used
  - RC write (one-sided) has higher throughput (FaRM)
- **Target at one-to-many data transferring paradigm**
  - e.g., MDS, KV store, parameter server, etc.
- **System-level solution**
  - Without any modifications to the hardware
- **Deployments**
  - Metadata server in Octopus
  - Distributed transactional system

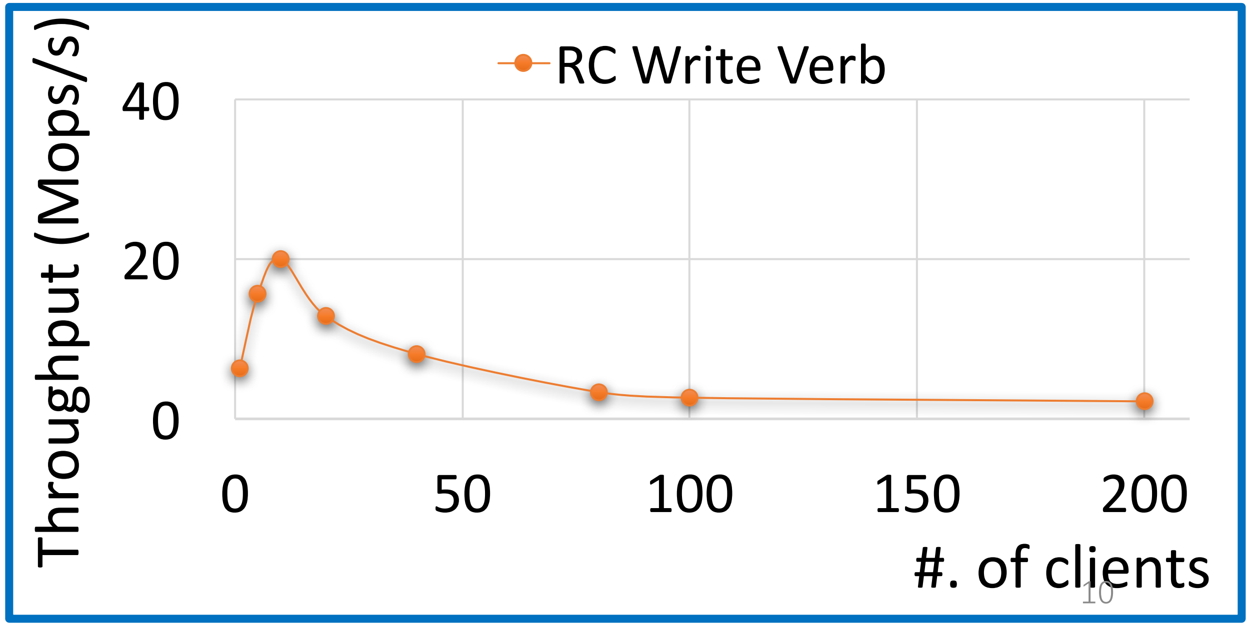
Outline

- **Grouping the connections**
- Multiplexing the message pool
- ScaleRPC: Putting it all together
- Evaluation
- Discussion and conclusion

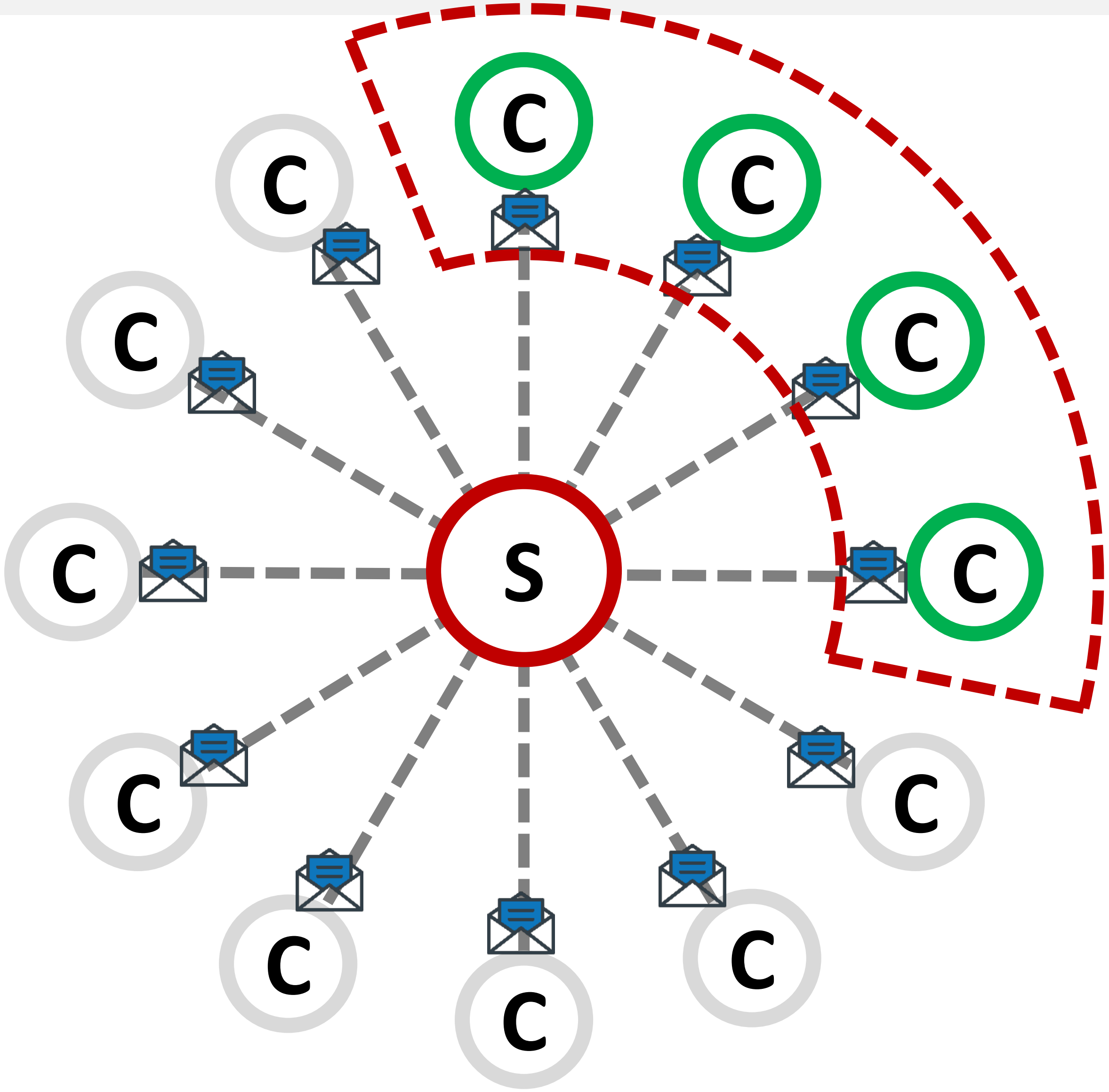
Grouping the connections



- **Naïve Approach**
  - NIC cache thrashing when the number of clients increases
  - Frequent swap in/out
  - Causing higher PCIe traffic

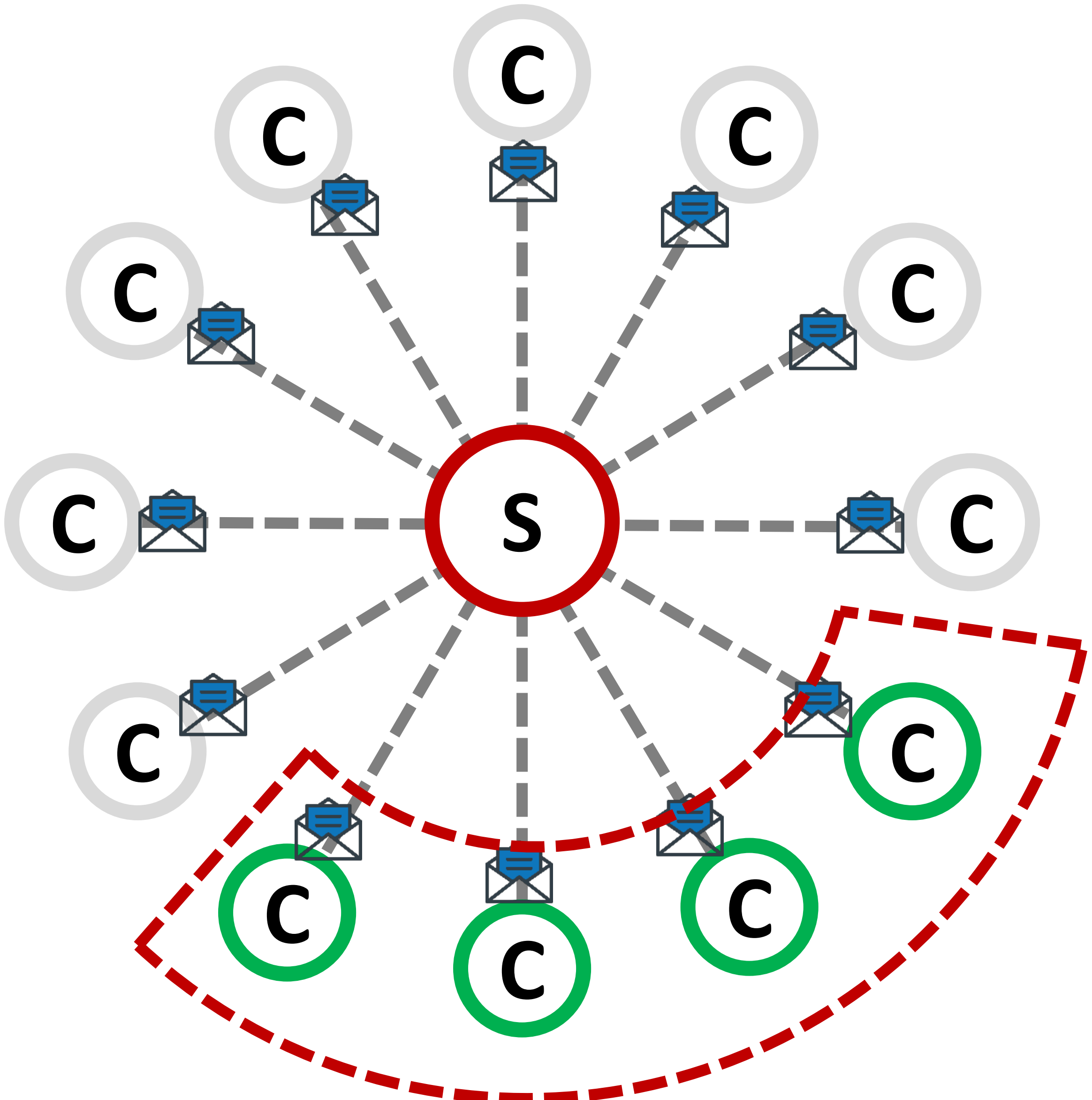


Grouping the connections



- **Connection Grouping**
  - Serve **one group at a time slice**

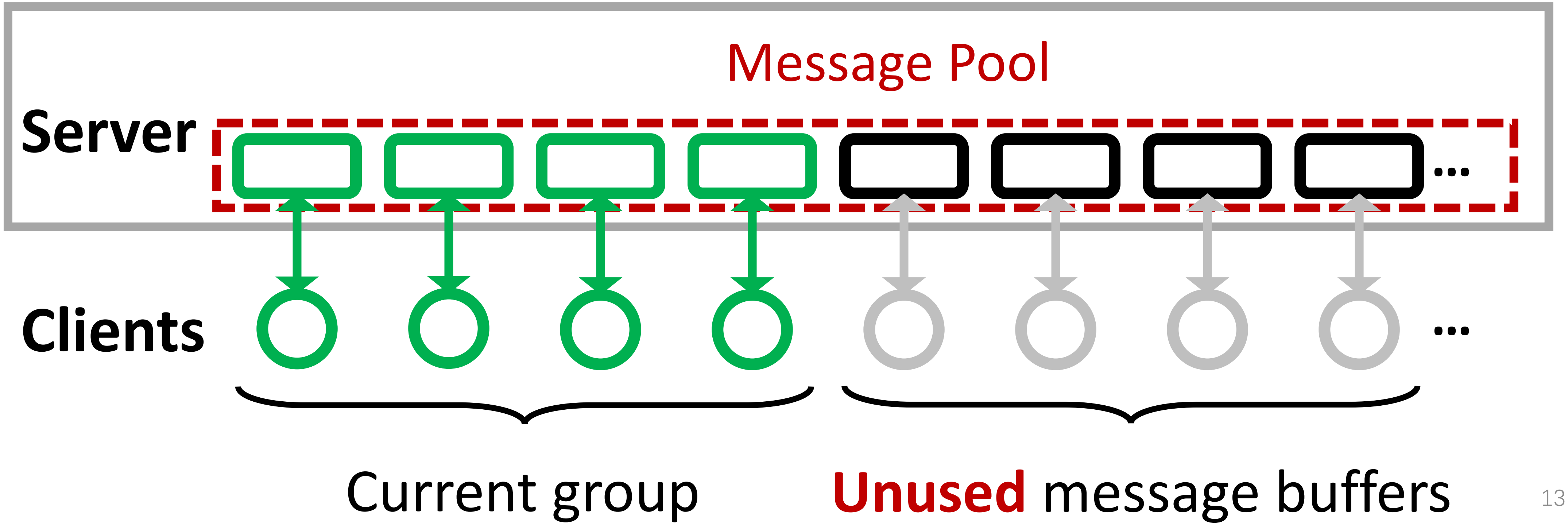
Grouping the connections



- Connection Grouping
  - Serve **one group at a time slice**
  - **Better cache locality:** recently accessed metadata is more likely be used again

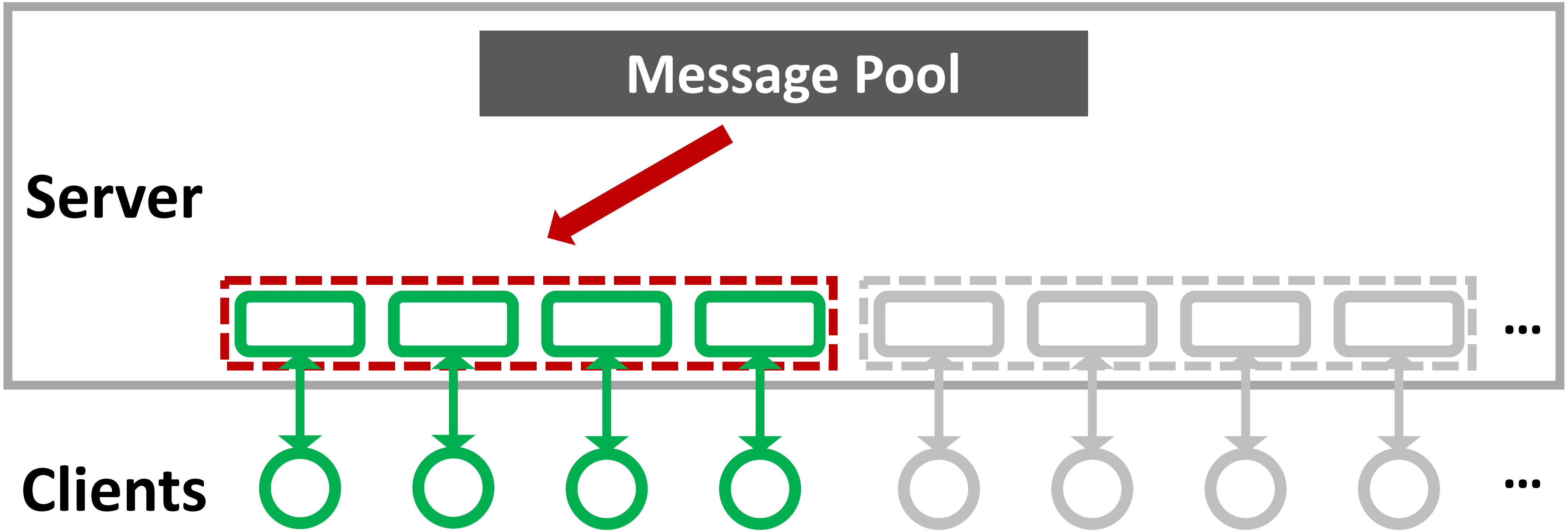
Virtualized Mapping

- **Alleviate the contention in the CPU cache**
  - Reduce memory footprint in the message pool
- **Observations:**
  - When grouping the clients, only **part** of the message pool is used



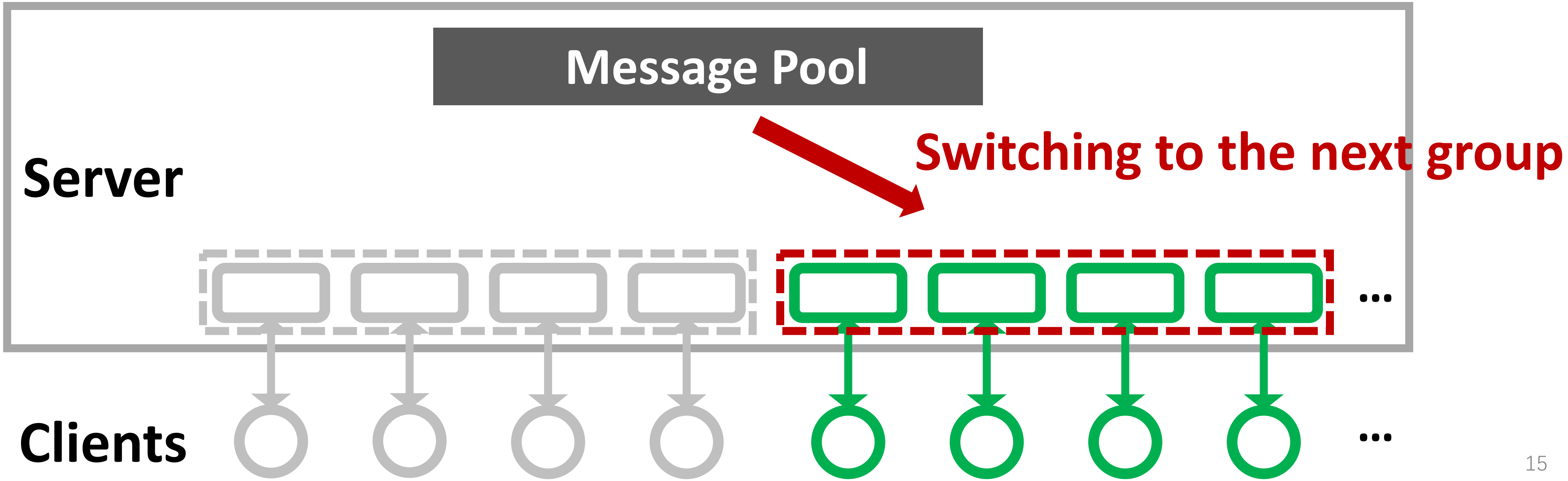
Virtualized Mapping

- We don't need to assign a message buffer for each client
  - **Virtualize** a single physical message pool to be **shared** among multiple groups
  - Without extra overhead for loading/saving the context



Virtualized Mapping

- We don’t need to assign a message buffer for each client
  - **Virtualize** a single physical message pool to be **shared** among multiple groups
  - Without extra overhead for loading/saving the context



Challenges & solutions

- **Static grouping** is **suboptimal** when clients have
  - Varying requirements for the tail latency
  - Varying frequencies of the posted RPCs
  - Varying payload sizes
  - Varying execution times for different handlers

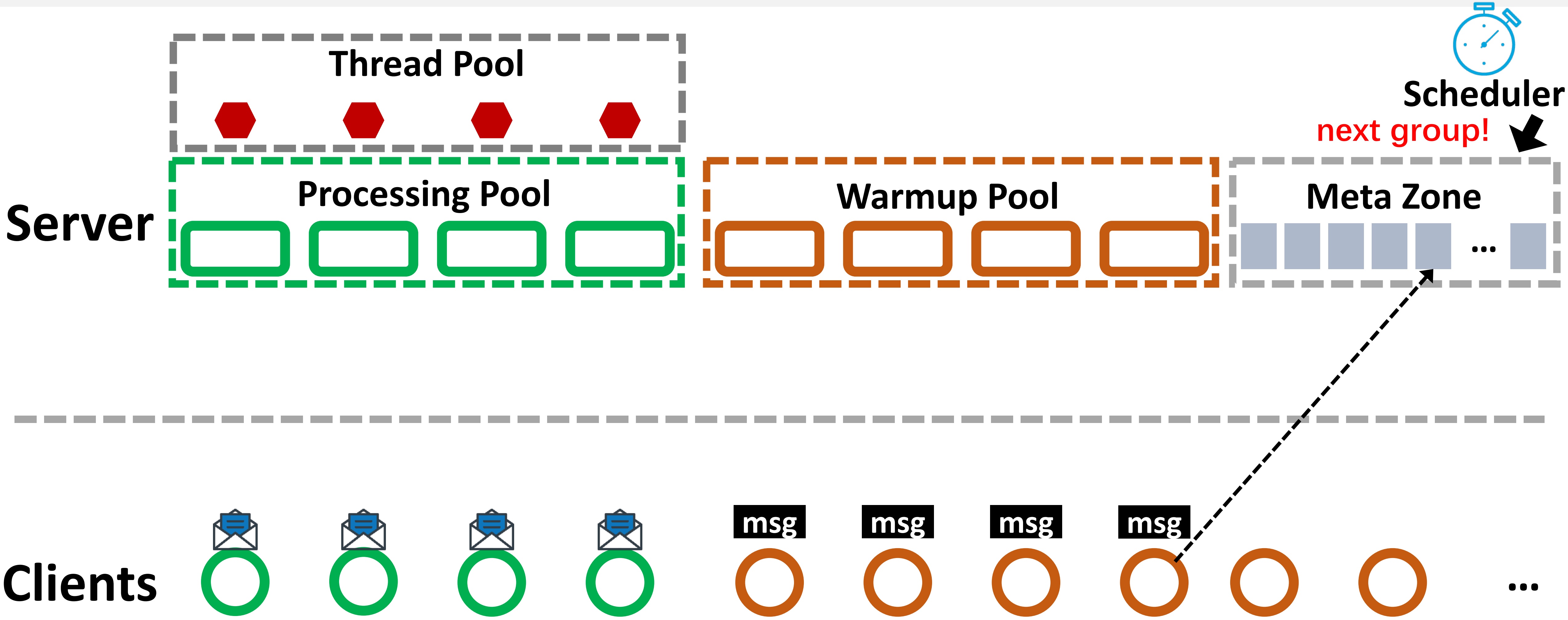
➡ **Priority-based scheduler:** monitors the performance of each clients and dynamically adjust the group size and time slice length.

- **Switching** between the groups should be **efficient**

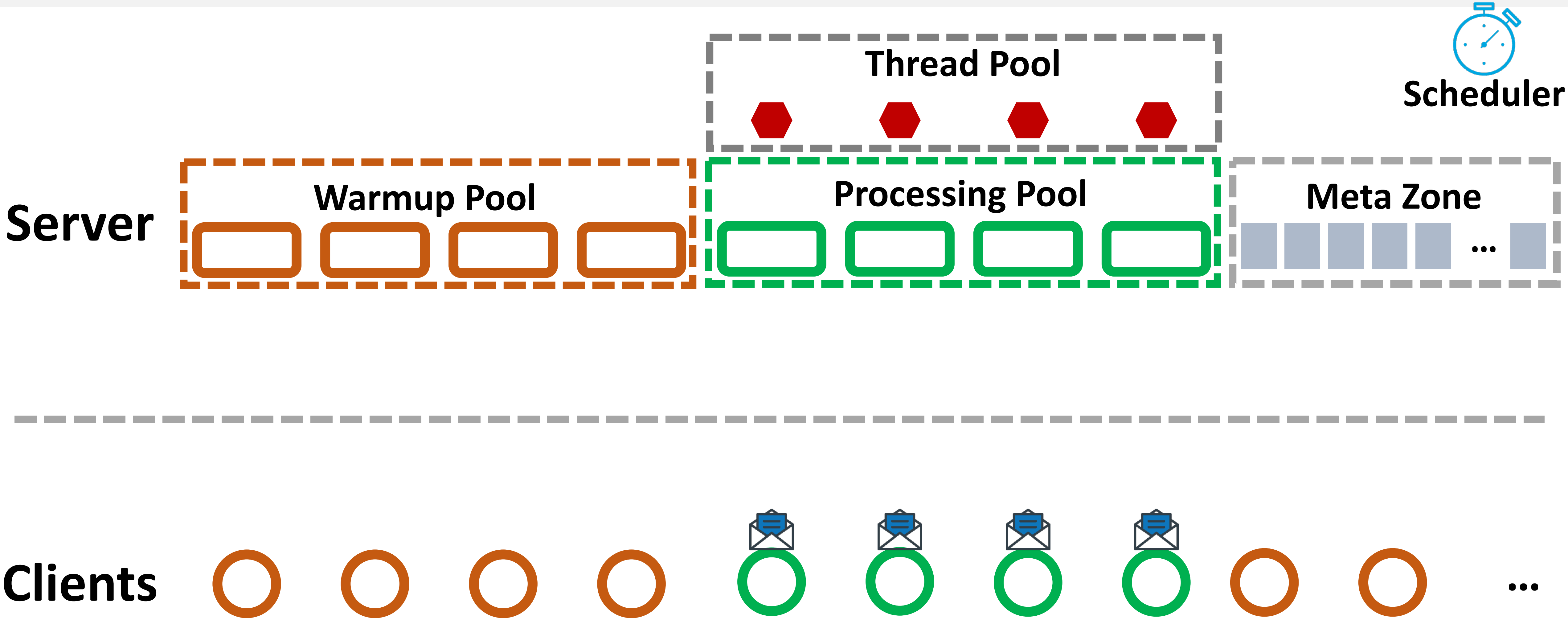
➡ **Warmup pool:** before being served, clients from the next group put their new requests in the warmup pool first

More: check our paper!

ScaleRPC: Putting it all together



ScaleRPC: Putting it all together



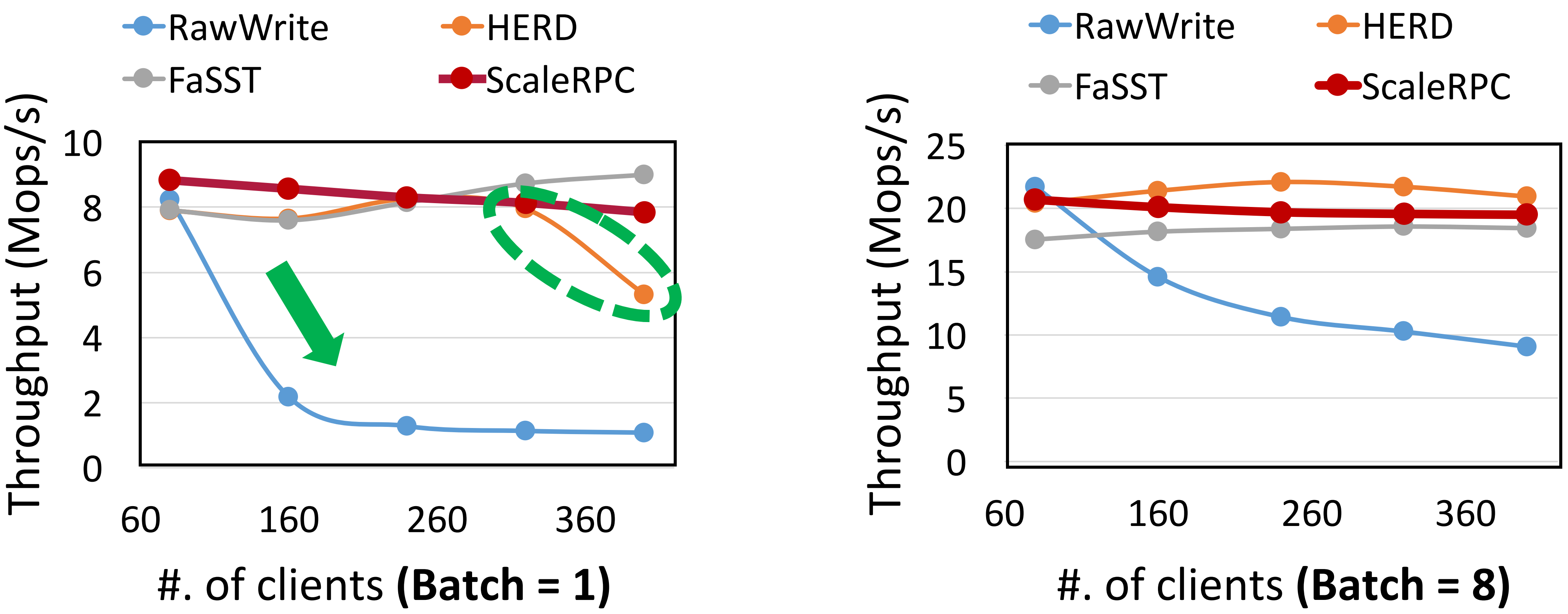
Evaluation

- **Platform**
  - 2× 2.2GHz Intel Xeon E5-2650 v4 CPUs (24 cores in total)
  - 128 GB DRAM
  - MCX353A CX-3 FDR HCAs (56 Gbps IB and 40 GbE)
  - 12-node cluster connected with Mellanox SX-1012 switch
- **Compared Systems**

RPC	Description
RawWrite RPC	A baseline RPC with all the optimizations in ScaleRPC disabled
HERD RPC	A scalable RPC with a hybrid of UC write and UD send verbs
FaSST RPC	A scalable RPC based on UD send verbs

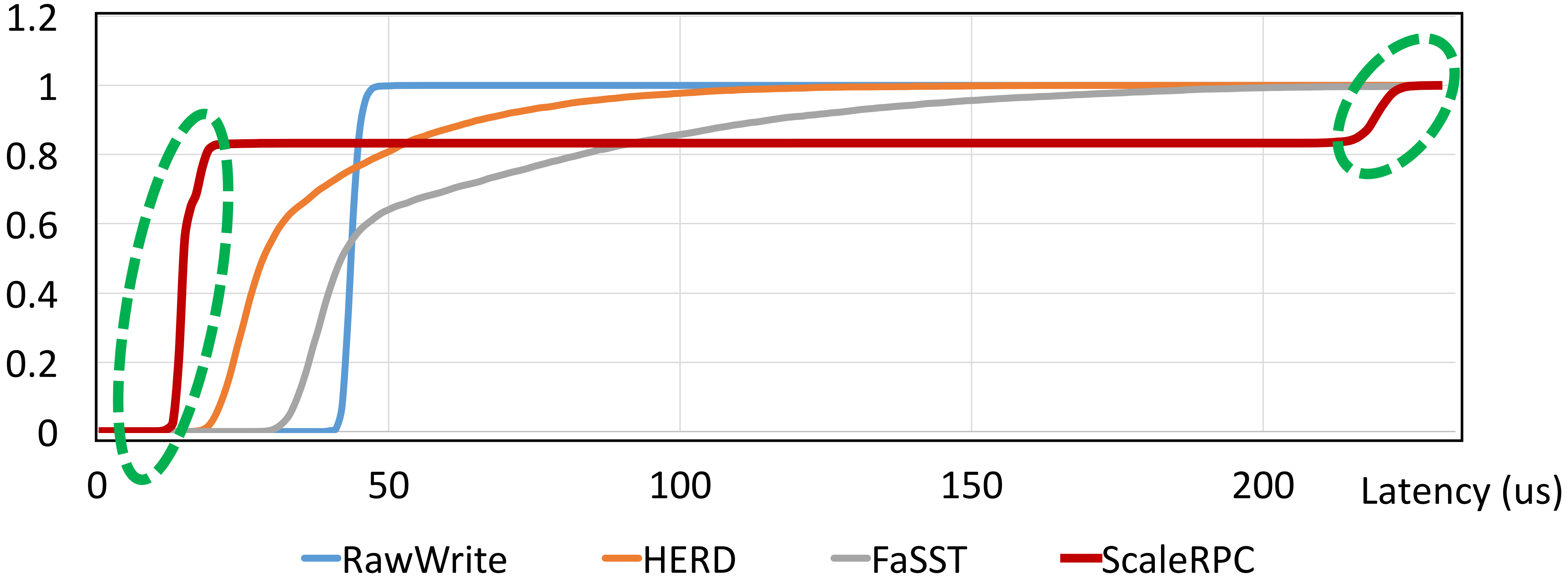
Evaluation

■ **Throughput**



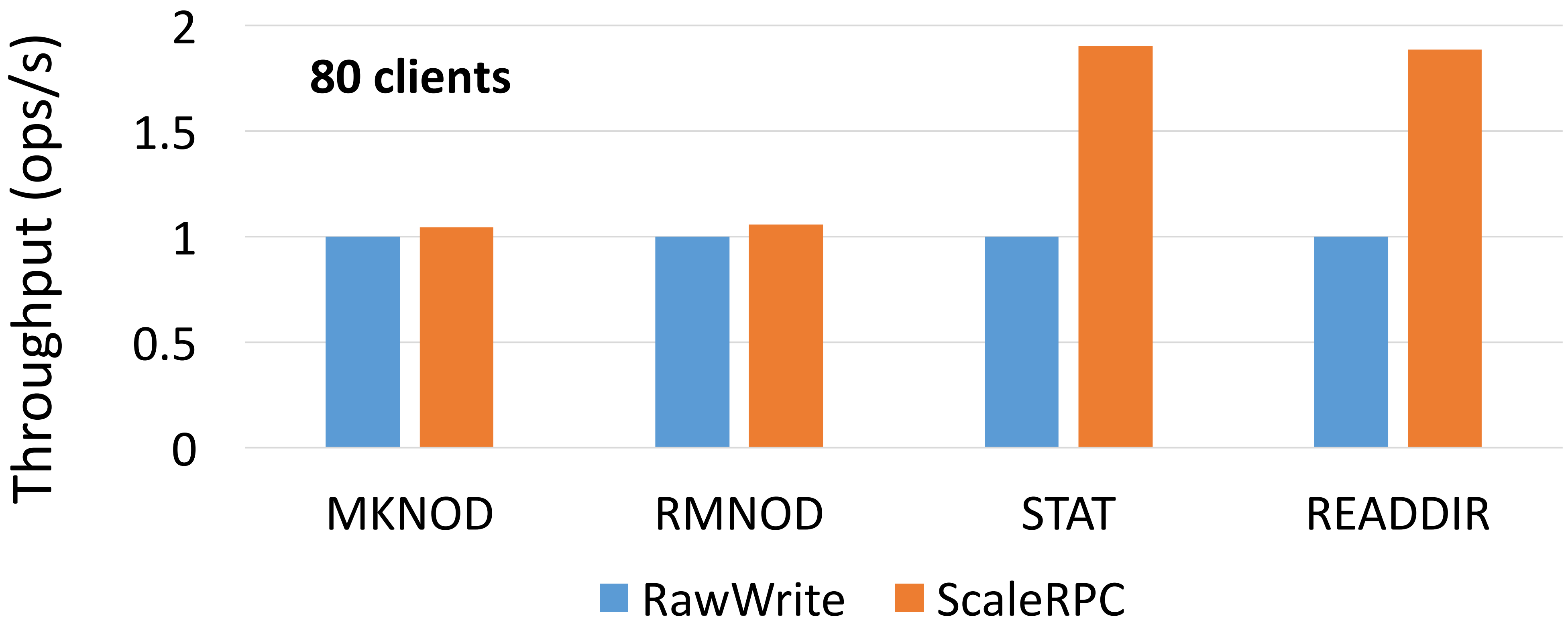
Evaluation

■ **Latency distribution**



Evaluation

- **Metadata Server in Octopus (Distributed File System)**



Conclusions

- A **system-level approach** to improve the scalability of RC RDMA
- **Connection grouping** and **virtualized mapping** to efficiently share the hardware resources

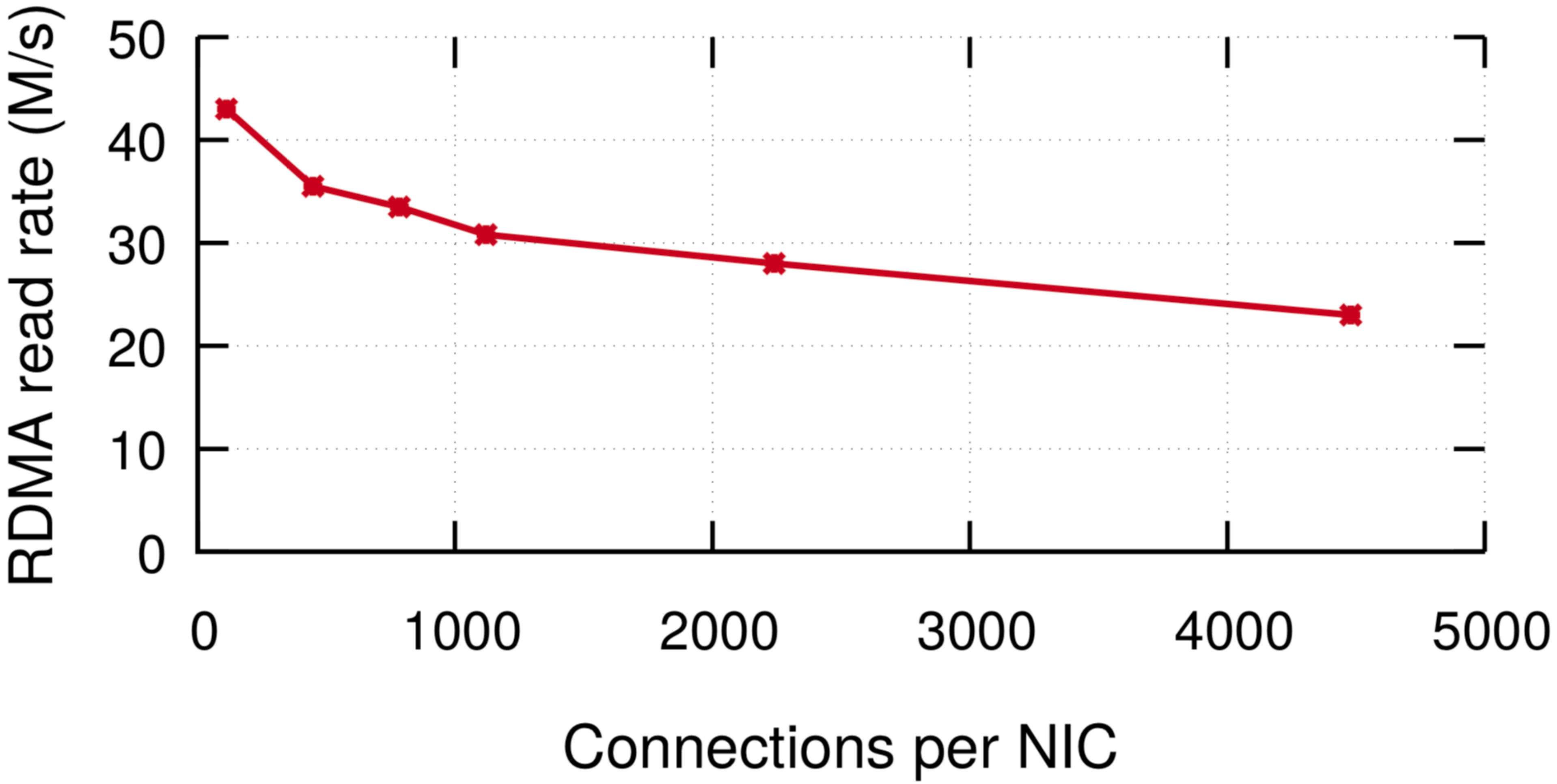


Thanks & QA

Tsinghua University

http://storage.cs.tsinghua.edu.cn

Backups



Source Image from eRPC[NSDI'19]

Discussions

Other potential approaches

- **Dynamically Connected Transport (DCT)**
  - Sharing the context between all the connections
  - DCT almost doubles the number of packet
  - Increases latency by 100ns to 3us on RC mode[1]
- **Newer generation of HCAs (CX-4/5)**
  - eRPC reveals that with CX-5, the throughput drops almost by half as the number of connections increases to 5K[2]

[1] Hari Subramoni, Khaled Hamidouche, Akshey Venkatesh, Sourav Chakraborty, and Dhabaleswar K Panda. 2014. Designing MPI library with dynamic connected transport (DCT) of InfiniBand: early experiences. In International Supercomputing Conference. Springer, 278–295.

[2] Kalia, Anuj, Michael Kaminsky, and David G. Andersen. "Datacenter RPCs can be General and Fast." *NSDI'19* (2019).

Discussions

■ Deploymment Considerations

- ScaleRPC assumes the clients execute independently and there is no synchronization among them

➡ Less common case

- The RPC server and clients are assumed to cooperate together to make the aforementioned optimizations work properly

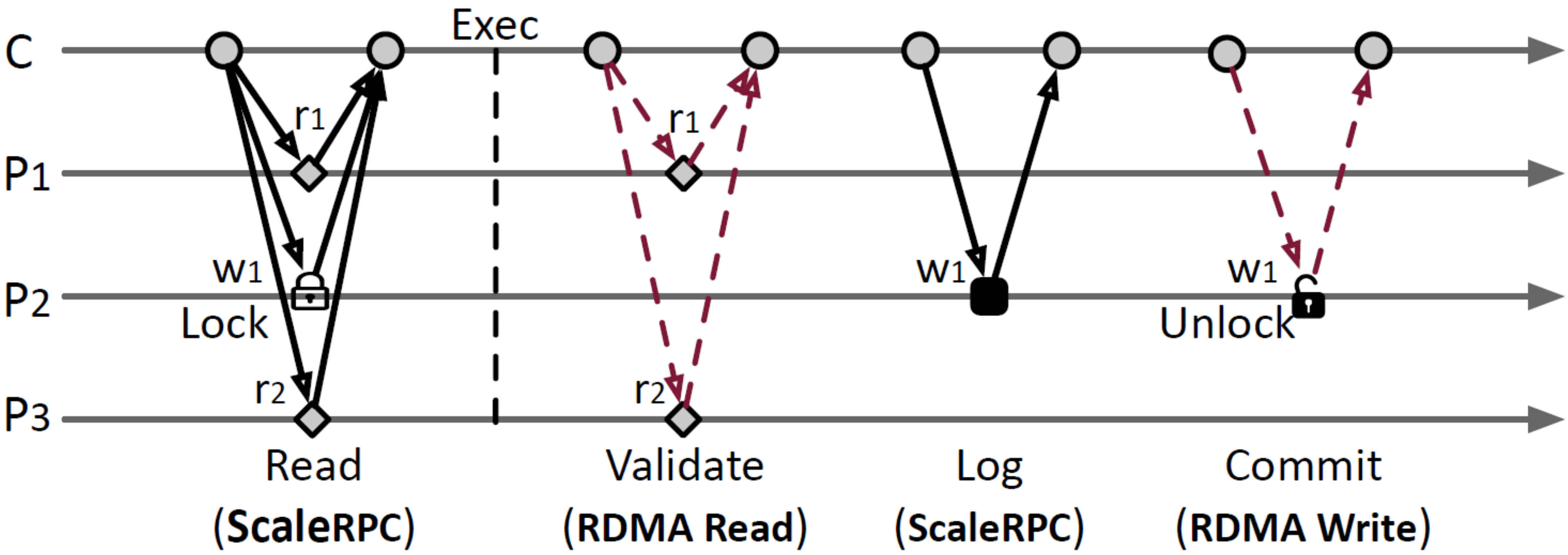
➡ A bunch of easy-to-use APIs (SyncCall, AsyncCall, PollCompletion)

- ScaleRPC improves the overall throughput and shortens the average latency, but magnifies the tail latency

➡ Rely on the priority-based scheduler to share the resources

Evaluation

- **with TX**



Evaluation

- with TX