Replicating Persistent Memory Key-Value Stores with Efficient RDMA Abstraction

Qing Wang, Youyou Lu, Jing Wang, Jiwu Shu

Tsinghua University

Replicated Distributed Key-Value Stores

Replicated distributed key-value stores (KVSs) support many apps

- ❖ Durability ⇨ Storage devices (HDD、SSD)
- ❖ High availability ⇨ Data replication

Amazon DynamoDB

cassandra

ZippyDB

FOUNDATIONDB

Replicated Distributed Key-Value Stores

Replicated distributed key-value stores (KVSs) support many apps

- Durability ⇨ Storage devices (HDD、SSD)
- High availability ⇨ Data replication

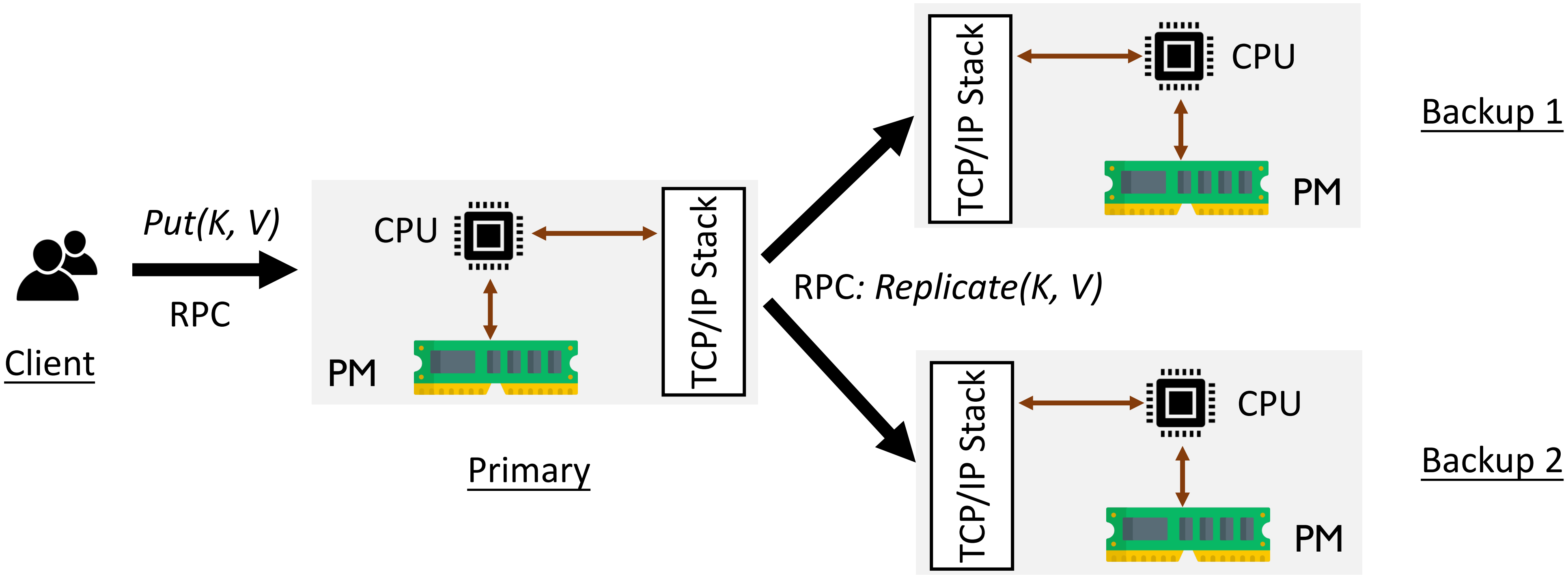
Amazon DynamoDB cassandra ZippyDB FOUNDATIONDB

How to optimize the latency of replicated KVSs by leveraging **modern hardware** ?

Step 1: Persistent Memory

Using persistent memory (PM) for storage

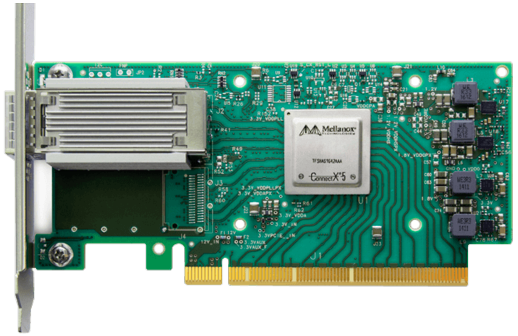
- Byte-addressable via *load/store* instructions
- Low latency (~100ns for small I/O)
- High-bandwidth (2GB/s write and 6GB/s read per DIMM)

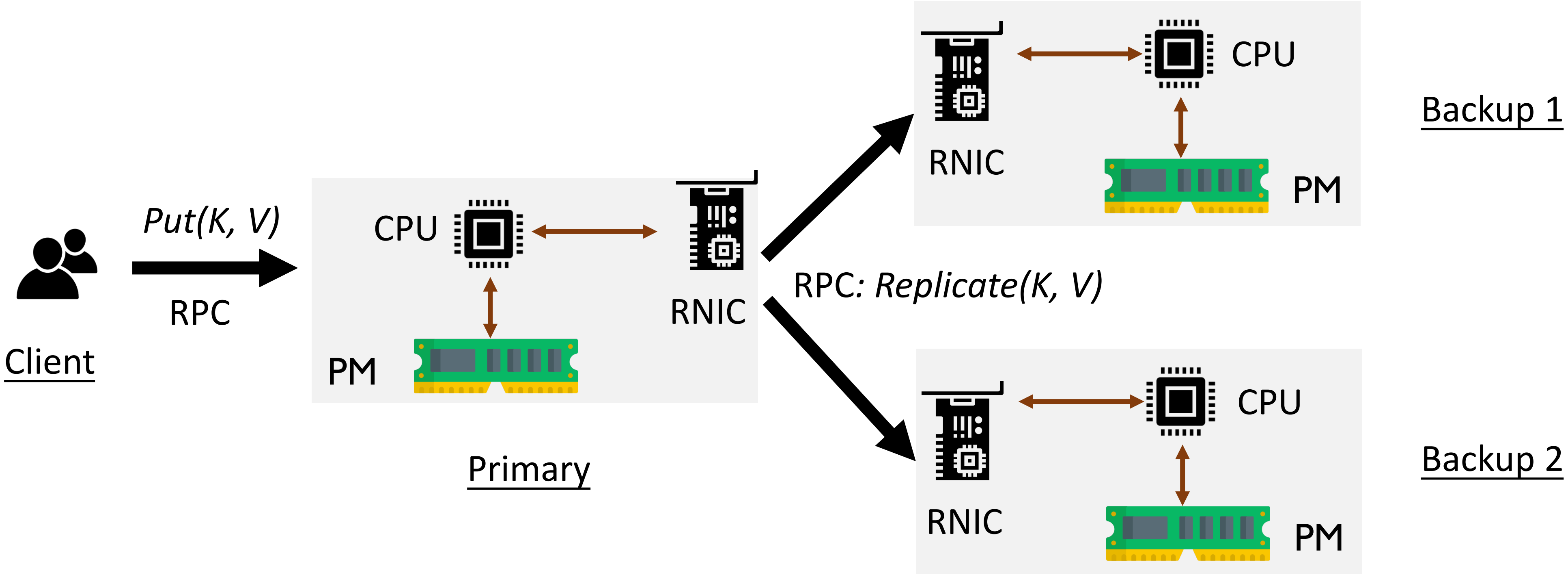


Step 2: RDMA Network

Using RDMA for network

- Bypass OS kernel: threads interact directly with NICs
- Hardware offloading: e.g., reliability (RC mode), packetization
- High performance: ~2μs RTT, 100-400Gbps

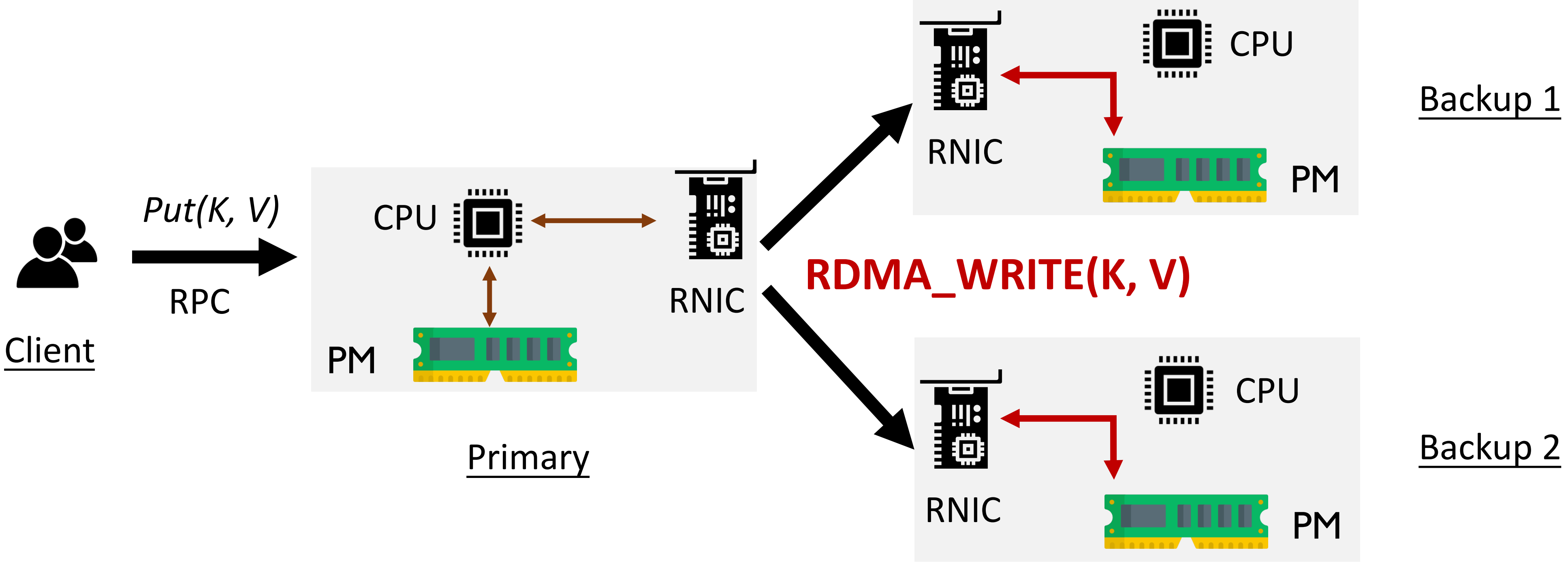




Step 3: One-sided Replication

Using one-sided WRITE for replication

- RDMA provides one-sided RDMA WRITE/READ, bypassing remote CPUs
- Primary pushes replicated objects to backups' PM via RDMA WRITE
- Eliminate *RPC queueing and CPU execution* of *backups* in the critical path
- E.g., Mu (OSDI'20, DRAM-based)



However, RDMA WRITE induces write amplification

Each server holds a number of backup logs and receives small RDMA WRITE

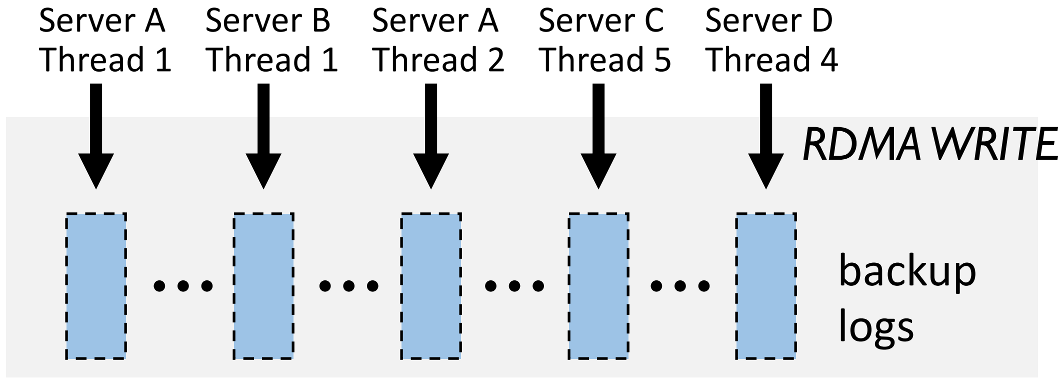
A number of backup logs caused by sharding:

Each server acts as backups for many shards

⇩

Allocates lots of backup logs, each accommodating RDMA WRITE from a remote thread (primaries)

- FaRM has thousands of backup logs per server
- #log = (#server – 1) * #(threads per server)



However, RDMA WRITE induces write amplification

Each server holds a number of backup logs and receives small RDMA WRITE

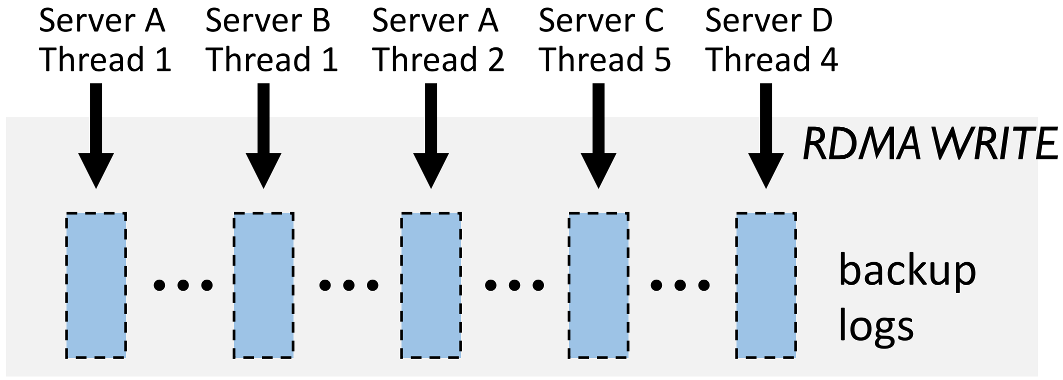
A number of backup logs caused by sharding:

Each server acts as backups for many shards

⇩

Allocates lots of backup logs, each accommodating RDMA WRITE from a remote thread (primaries)

- FaRM has thousands of backup logs per server
- #log = (#server – 1) * #(threads per server)



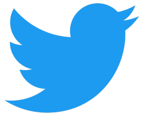
Small RDMA WRITE caused by small objects:

Small objects are prevalent

- In Meta's largest KVS ZippyDB, the average object size is ***90.8B*** (FAST'20)

Meta

- At Twitter, the average tweet is less than ***33 characters*** (Kangaroo, SOSP'21)



- ……

However, RDMA WRITE induces write amplification

PM devices have byte interface with a block-level internal access granularity

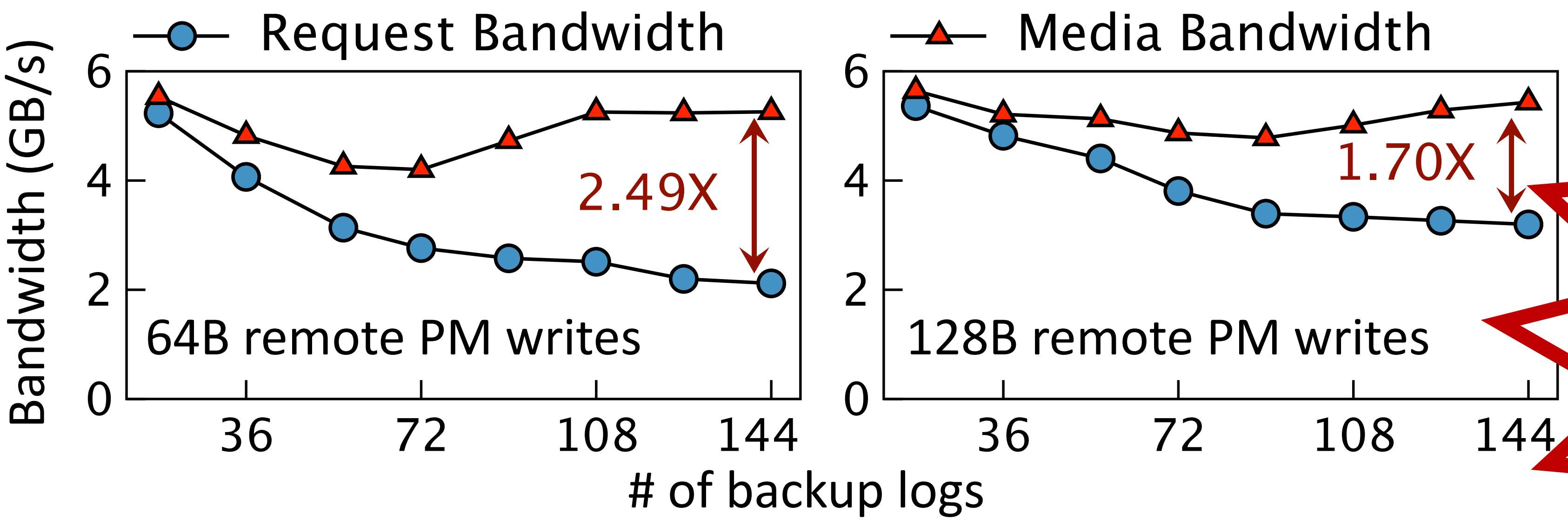
- Optane PM: 256B XPLine; CXL-SSD: Flash Page
- Devices combine *adjacent* small writes to control device-level write amplification (DLWA)
- Implication: PM devices prefer large writes or sequential small writes

However, RDMA WRITE induces write amplification

PM devices have byte interface with a block-level internal access granularity

- Optane PM: 256B XPLine; CXL-SSD: Flash Page
- Devices combine *adjacent* small writes to control device-level write amplification (DLWA)
- Implication: PM devices prefer large writes or sequential small writes

One-sided Replication in KVS: random small writes



(In the PM server, 18 cores perform local sequential PM writes, DDIO disabled)

Severe device-level write amplification

How to mitigate device-level write amplification ?

Using software batching ?

- ❖ Accumulate small writes within a timeout, then emit the batched writes to remote backup logs via one RDMA WRITE
- ❖ Problem:
  - Induce extra latency, remove benefits of extremely low-latency HW (PM、RDMA)
  - GET operations and sharding reduce the opportunity of batching

How to mitigate device-level write amplification ?

Using software batching ?

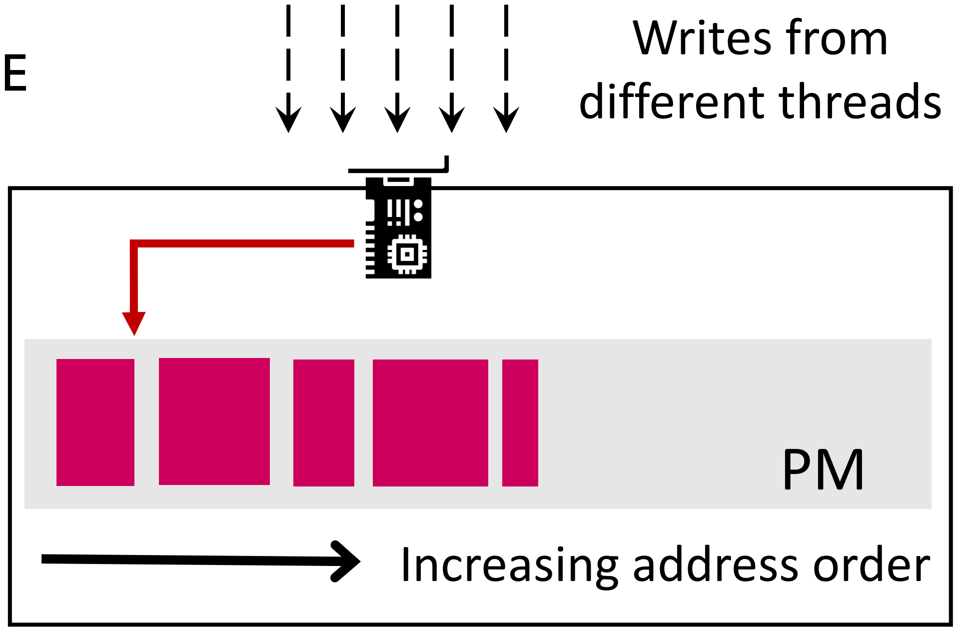
- ❖ Accumulate small writes within a timeout, then emit the batched writes to remote backup logs via one RDMA WRITE
- ❖ Problem:
  - Induce extra latency, remove benefits of extremely low-latency HW (PM、RDMA)
  - GET operations and sharding reduce the opportunity of batching

Can we mitigate DLWA without inducing any software delay ?

Our Idea – New RDMA abstraction: Rowan

Rowan (remote write aggregation):

- Receiver-side NICs land remote writes to PM sequentially, and return ACKs
- Receiver-side NICs decide destination addresses
  - Do not need per-remote-thread log area for RDMA WRITE

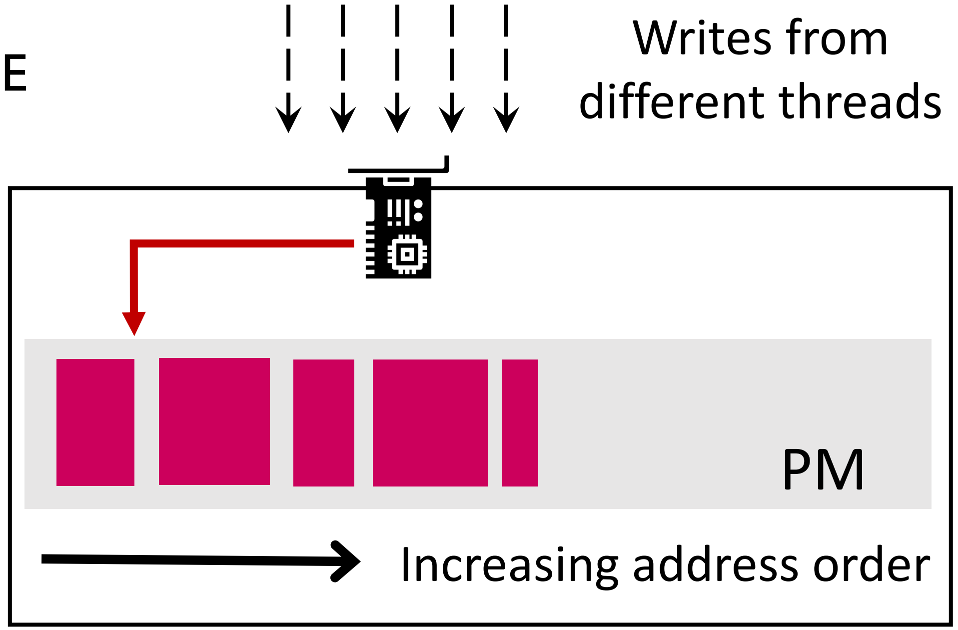


Rowan Abstraction (Receiver-side)

Our Idea – New RDMA abstraction: Rowan

Rowan (remote write aggregation):

- ❖ Receiver-side NICs land remote writes to PM sequentially, and return ACKs
- ❖ Receiver-side NICs decide destination addresses
  - Do not need per-remote-thread log area for RDMA WRITE
- ❖ Benefits
  - Low latency: one-sided, no delay at sender/receiver
  - Low DLWA: sequential small writes
  - High throughput: NIC ASIC executes data path

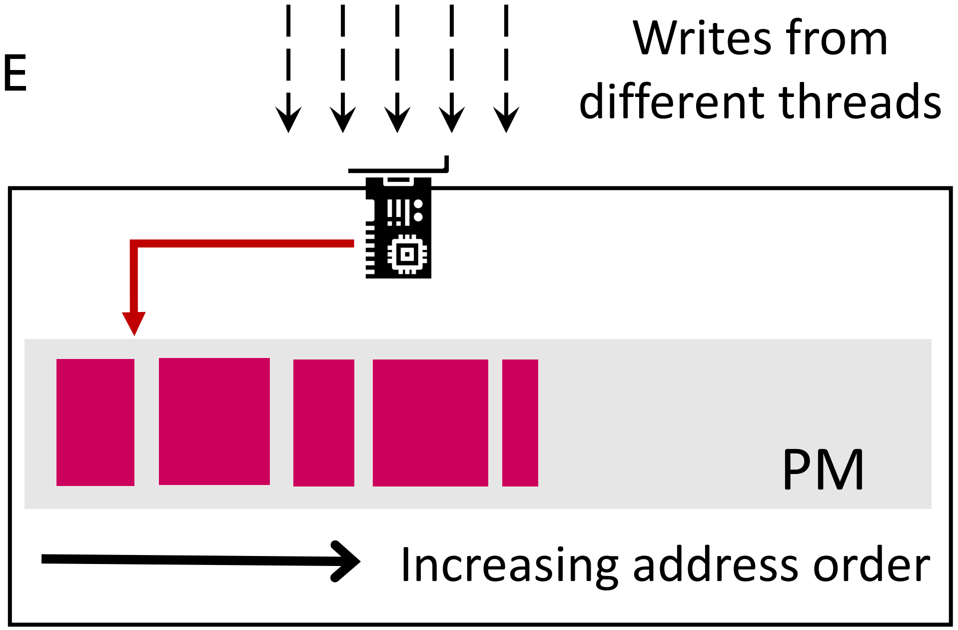


Rowan Abstraction (Receiver-side)

Our Idea – New RDMA abstraction: Rowan

Rowan (remote write aggregation):

- ❖ Receiver-side NICs land remote writes to PM sequentially, and return ACKs
- ❖ Receiver-side NICs decide destination addresses
  - Do not need per-remote-thread log area for RDMA WRITE
- ❖ Benefits
  - Low latency: one-sided, no delay at sender/receiver
  - Low DLWA: sequential small writes
  - High throughput: NIC ASIC executes data path



Rowan Abstraction (Receiver-side)

Simple RDMA abstraction, but how to implement it using commodity RDMA NICs ?

Observations

Observation 1:

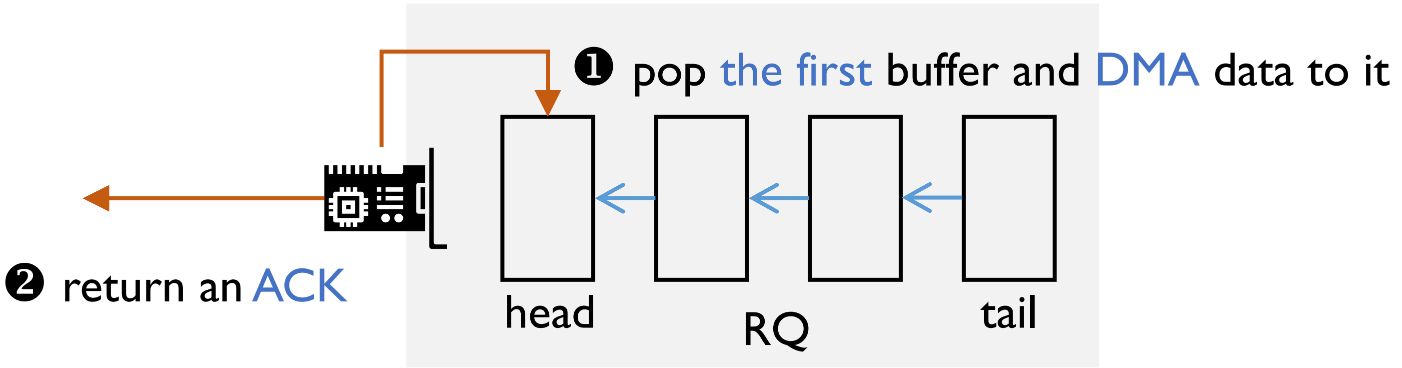
❖ RDMA SEND in RC mode is one-sided on the data path

- Control path: receiver's CPU prepares receive buffers via RDMA RECV
- Data path: receiver's NIC performs all tasks: DMA data, and return hardware ACKs

Observation 2:

❖ In a receive queue (RQ), receive buffers are consumed in order

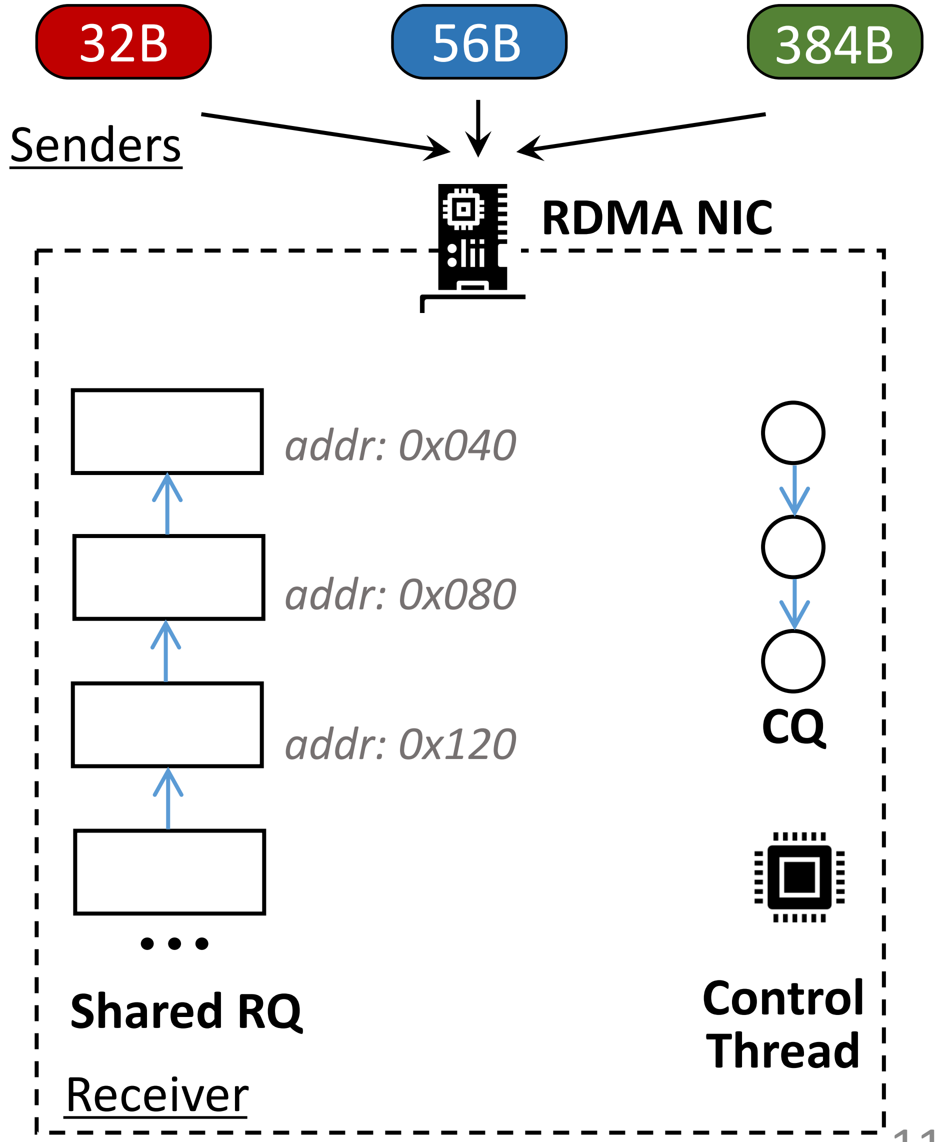
- the receiver-side NIC pops the first buffer in the associated RQ and lands data to it



Rowan – Basic Architecture

Rowan Basic Architecture

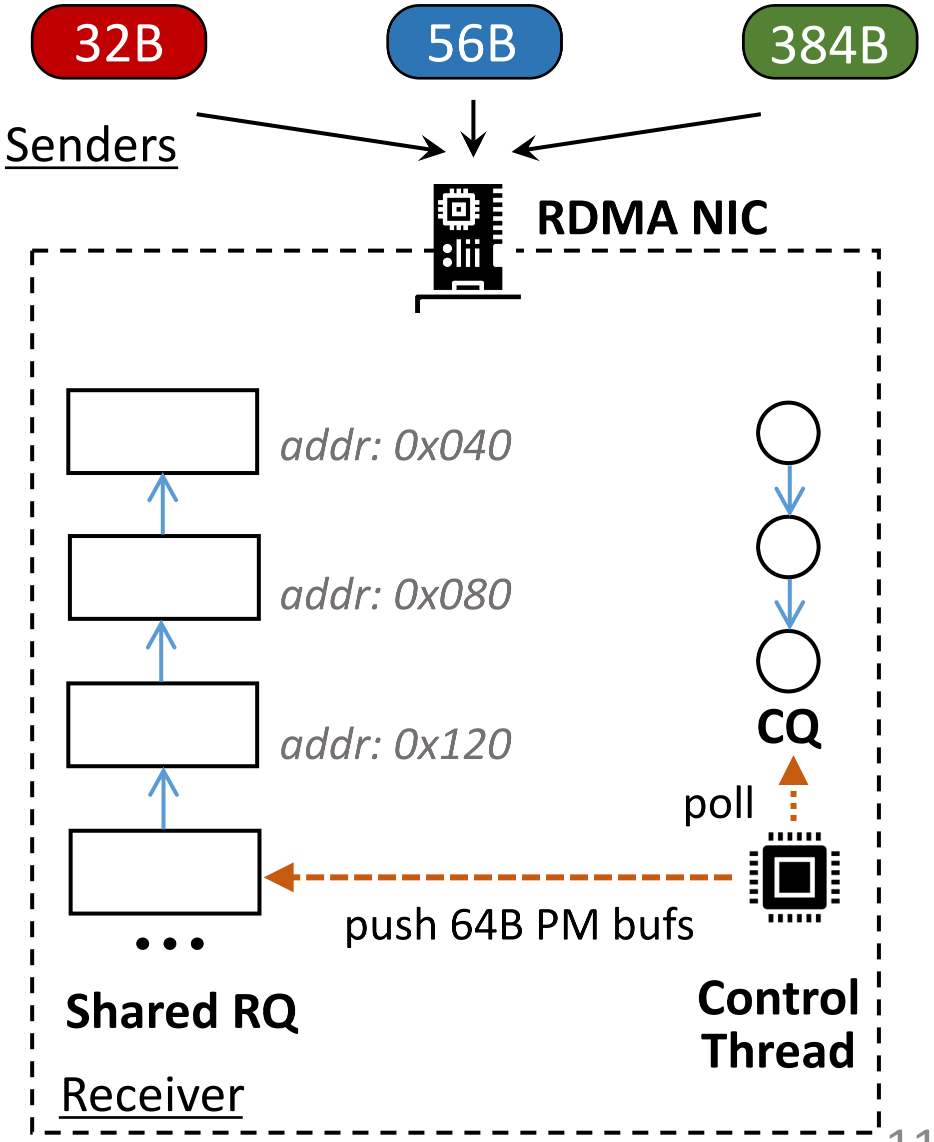
- RC Queue Pair (QP), enabling hardware ACKs
- A Shared Receive Queue (SRQ)
  - SEND requests from different remote QPs use the same RQ



Rowan – Basic Architecture

Rowan Basic Architecture

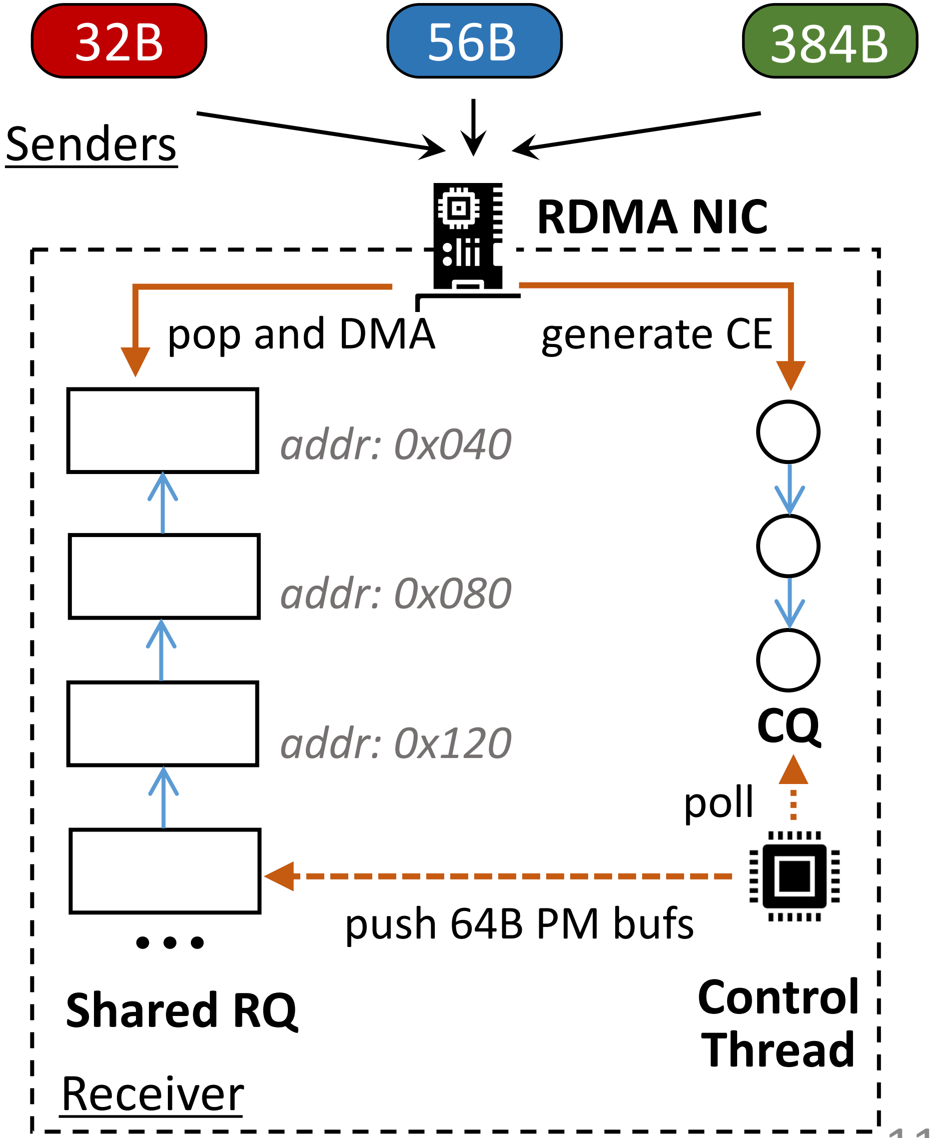
- ❖ RC Queue Pair (QP), enabling hardware ACKs
- ❖ A Shared Receive Queue (SRQ)
  - SEND requests from different remote QPs use the same RQ
- ❖ Control path: a control thread
  - Pushes 64B PM buffers to SRQ in increasing address order
  - Polls Completion Queue (CQ) of the SRQ



Rowan – Basic Architecture

Rowan Basic Architecture

- ❖ RC Queue Pair (QP), enabling hardware ACKs
- ❖ A Shared Receive Queue (SRQ)
  - SEND requests from different remote QPs use the same RQ
- ❖ Control path: a control thread
  - Pushes 64B PM buffers to SRQ in increasing address order
  - Polls Completion Queue (CQ) of the SRQ
- ❖ Data path: NIC
  - 1) Pops the first buffer in SRQ and DMAs data to it
  - 2) Returns an ACK and generates a CQ entry

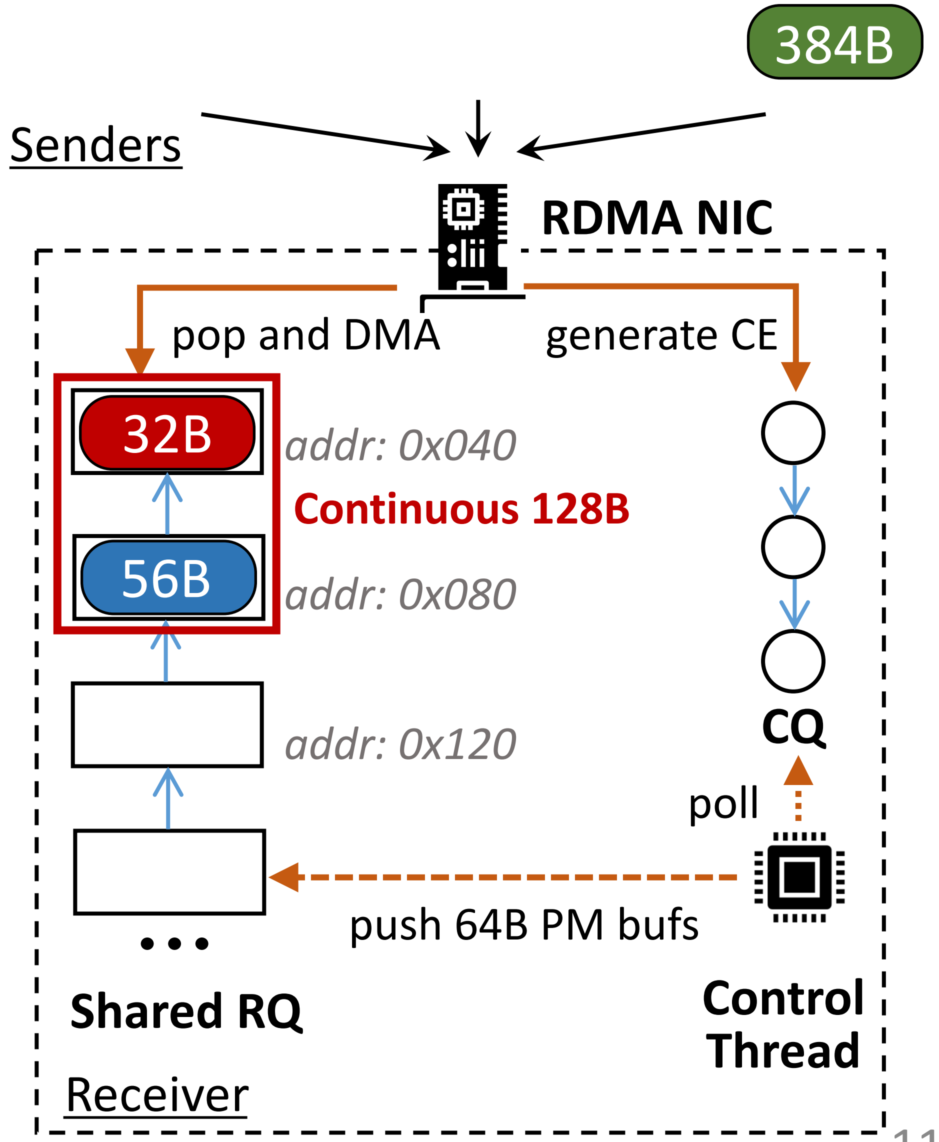


Rowan – Basic Architecture

Rowan Basic Architecture

- RC Queue Pair (QP), enabling hardware ACKs
- A Shared Receive Queue (SRQ)
  - SEND requests from different remote QPs use the same RQ
- Control path: a control thread
  - Pushes 64B PM buffers to SRQ in increasing address order
  - Polls Completion Queue (CQ) of the SRQ
- Data path: NIC
  - 1) Pops the first buffer in SRQ and DMAs data to it
  - 2) Returns an ACK and generates a CQ entry

Writes from different senders can be combined into *the same PM internal block*

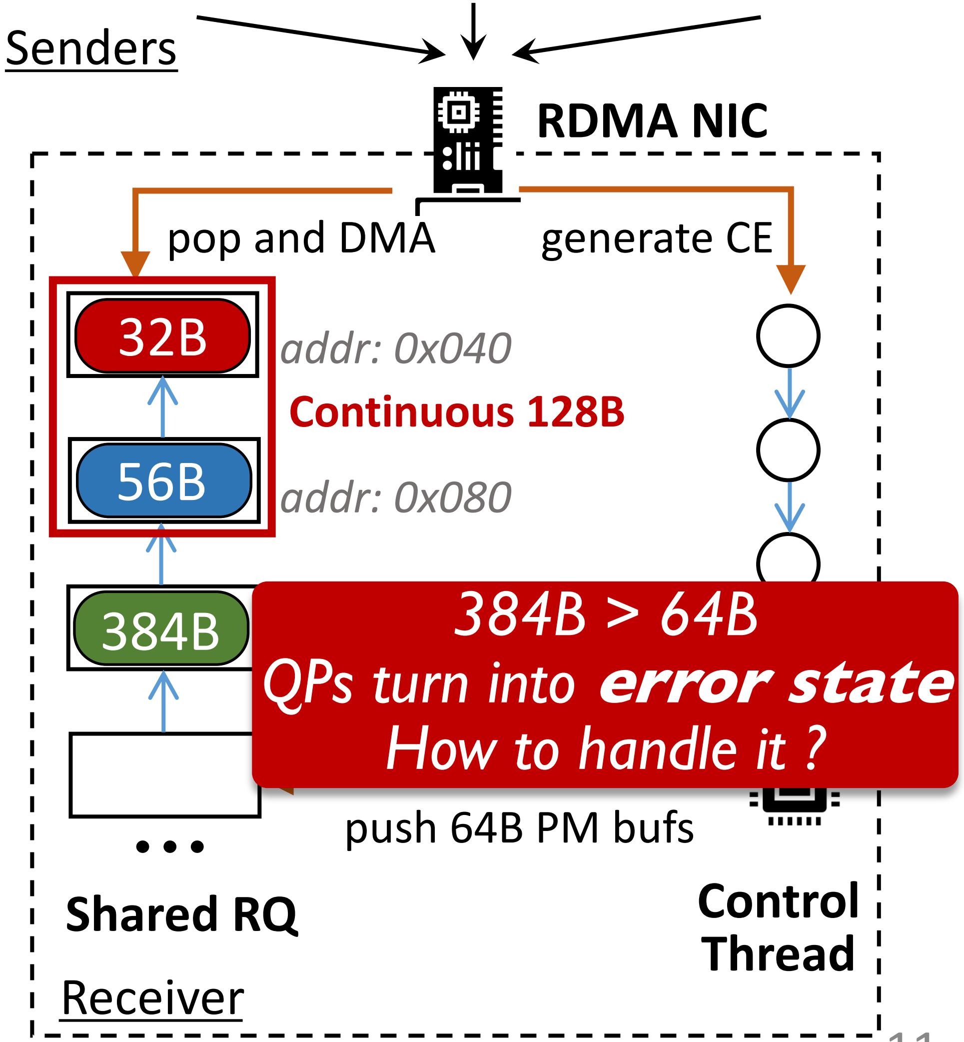


Rowan – Basic Architecture

Rowan Basic Architecture

- RC Queue Pair (QP), enabling hardware ACKs
- A Shared Receive Queue (SRQ)
  - SEND requests from different remote QPs use the same RQ
- Control path: a control thread
  - Pushes 64B PM buffers to SRQ in increasing address order
  - Polls Completion Queue (CQ) of the SRQ
- Data path: NIC
  - 1) Pops the first buffer in SRQ and DMAs data to it
  - 2) Returns an ACK and generates a CQ entry

Writes from different senders can be combined into *the same PM internal block*

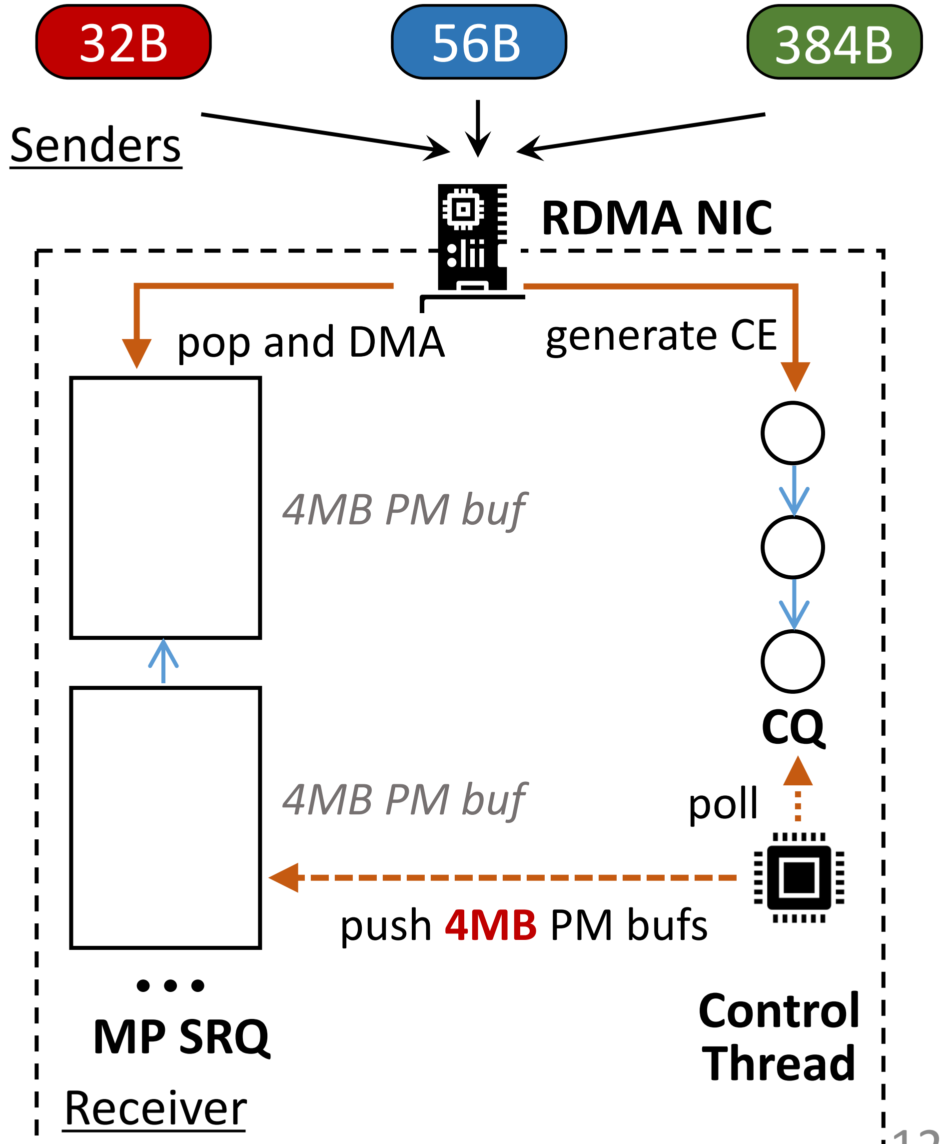


384B > 64B
*QPs turn into **error state***
How to handle it ?

Rowan – Handling Variable-sized Writes

Leveraging Multi-Packet (MP) RQ

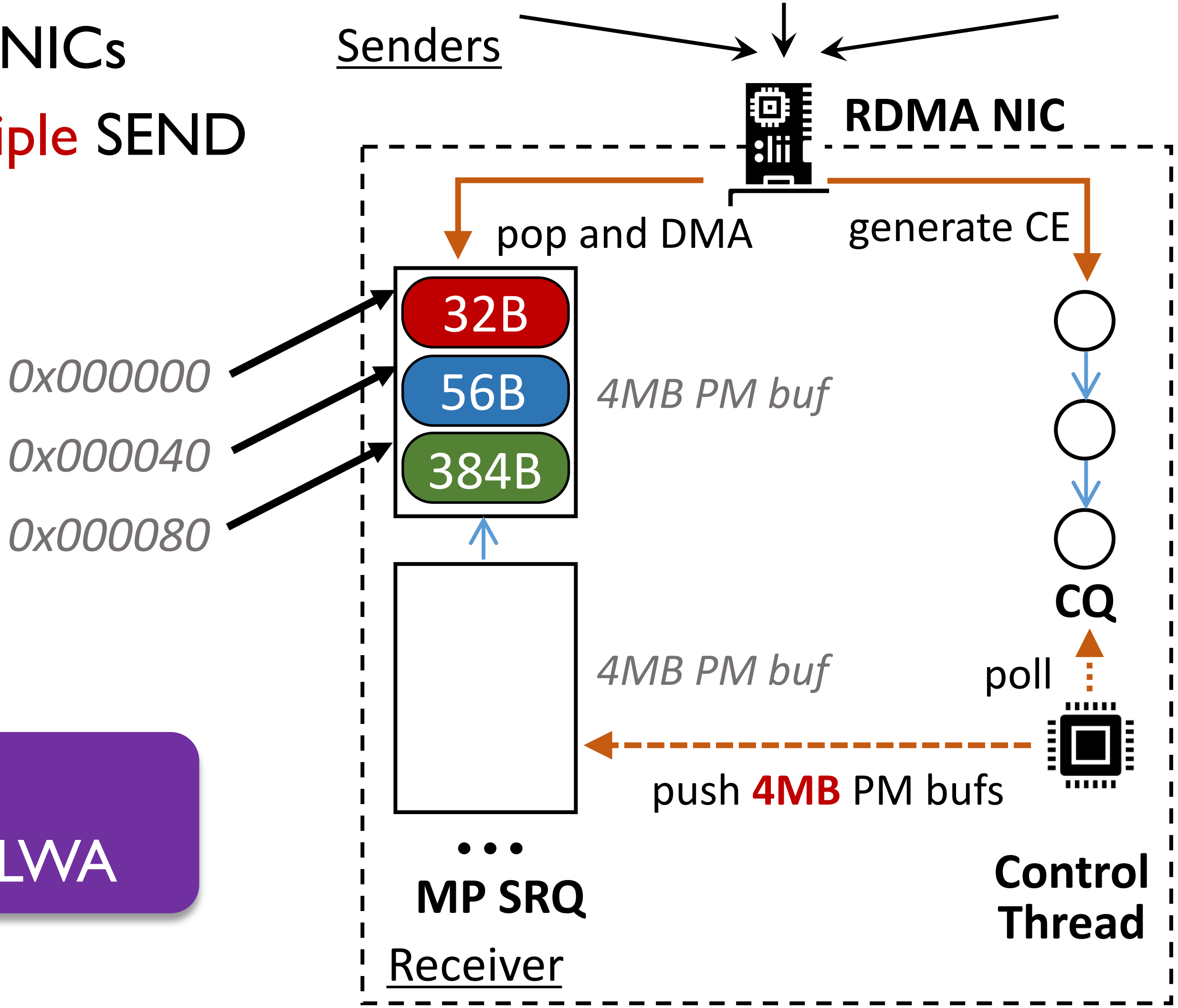
- A new type of RQ, supported by CX-4/5/6 NICs
- Each receive buffer can accommodate multiple SEND
- Define a stride (e.g., 64B in the right figure)
  - Each message has a stride-aligned start address



Rowan – Handling Variable-sized Writes

Leveraging Multi-Packet (MP) RQ

- ❖ A new type of RQ, supported by CX-4/5/6 NICs
- ❖ Each receive buffer can accommodate multiple SEND
- ❖ Define a stride (e.g., 64B in the right figure)
  - Each message has a stride-aligned start address

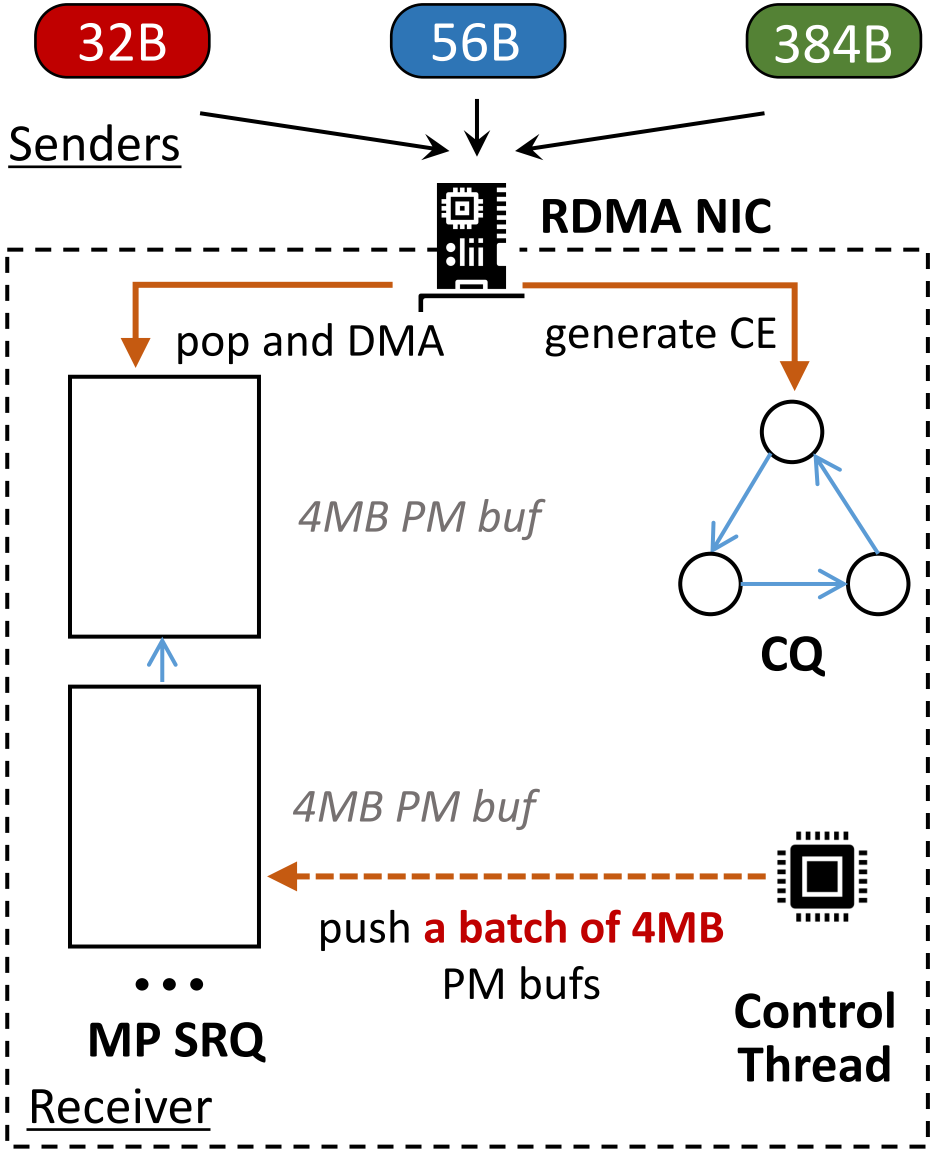


Rowan supports variable-sized writes,
while combining small writes to mitigate DLWA

Rowan – Control Path Optimization

Avoid control thread become bottleneck

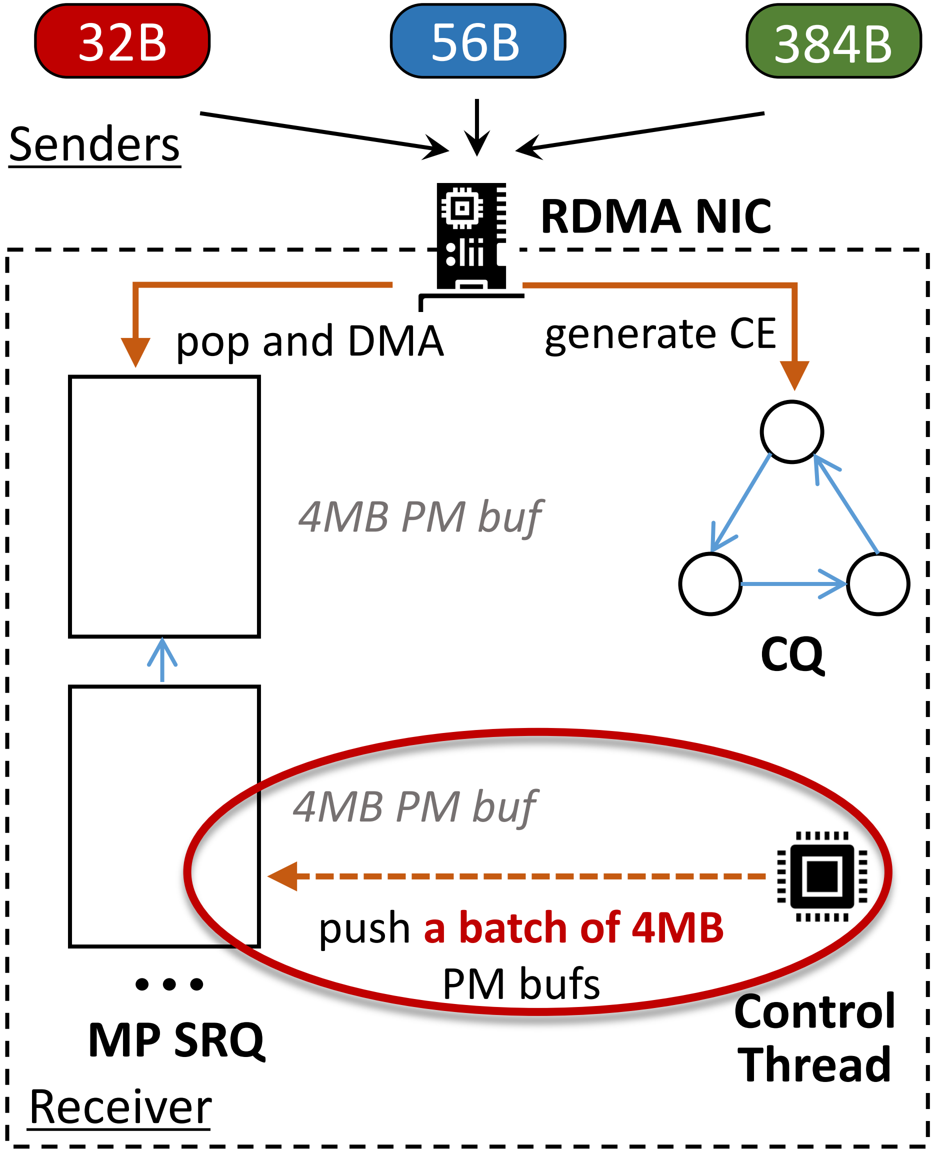
- ❖ Data path: > 50Mops/s
- ❖ Two tasks of control thread：
  - ❶ Push PM buffers to MP SRQ
  - ❷ Poll CQ (RDMA RECV cannot be unsignaled)



Rowan – Control Path Optimization

Avoid control thread become bottleneck

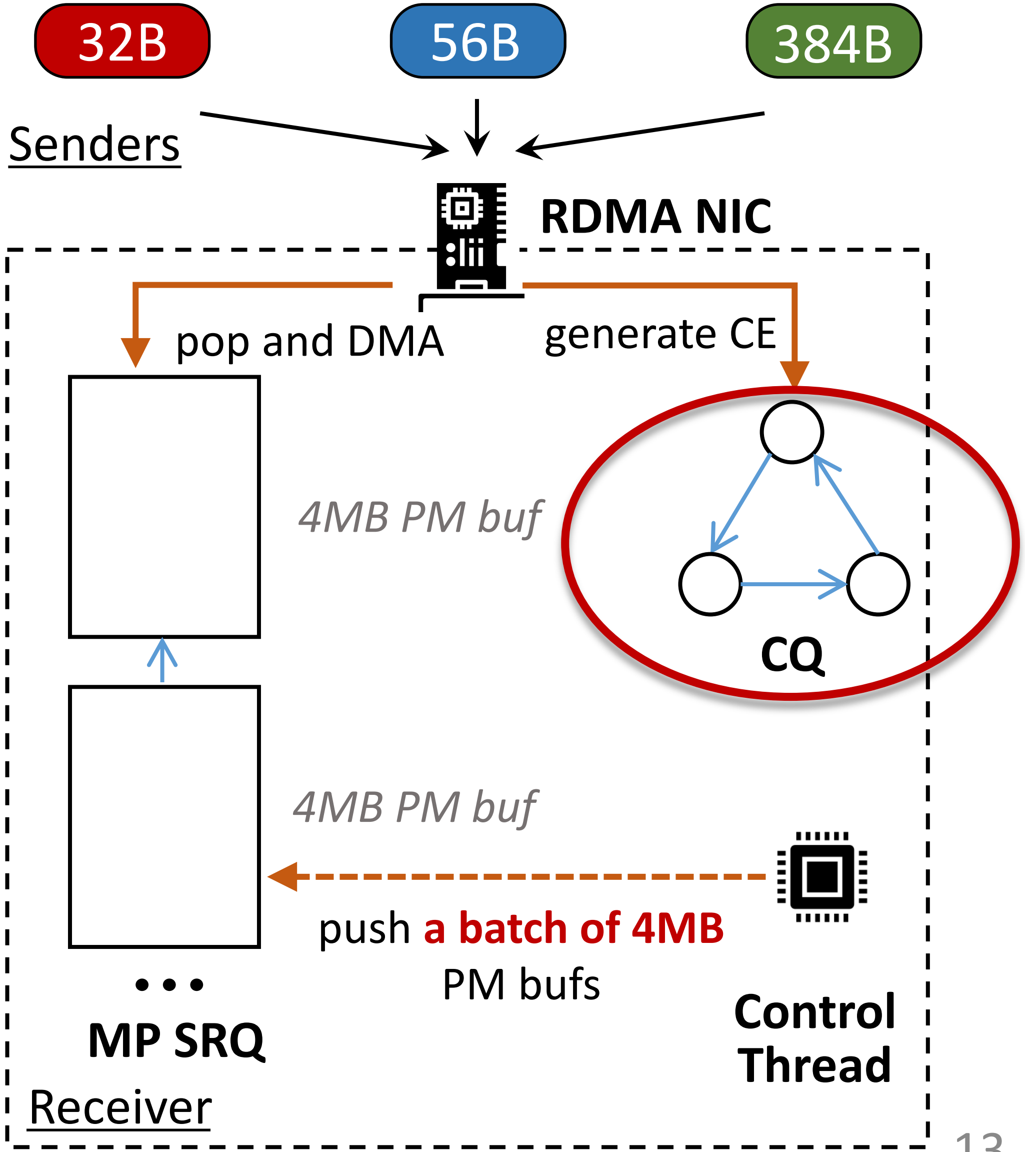
- Data path: > 50Mops/s
- Two tasks of control thread：
  - ❶ Push PM buffers to MP SRQ
  - ❷ Poll CQ (RDMA RECV cannot be unsignaled)
- Low overhead RDMA RECV
  - Large recv buffer (e.g., 4MB) using MP features
  - Post a batch of RDMA RECV at a time



Rowan – Control Path Optimization

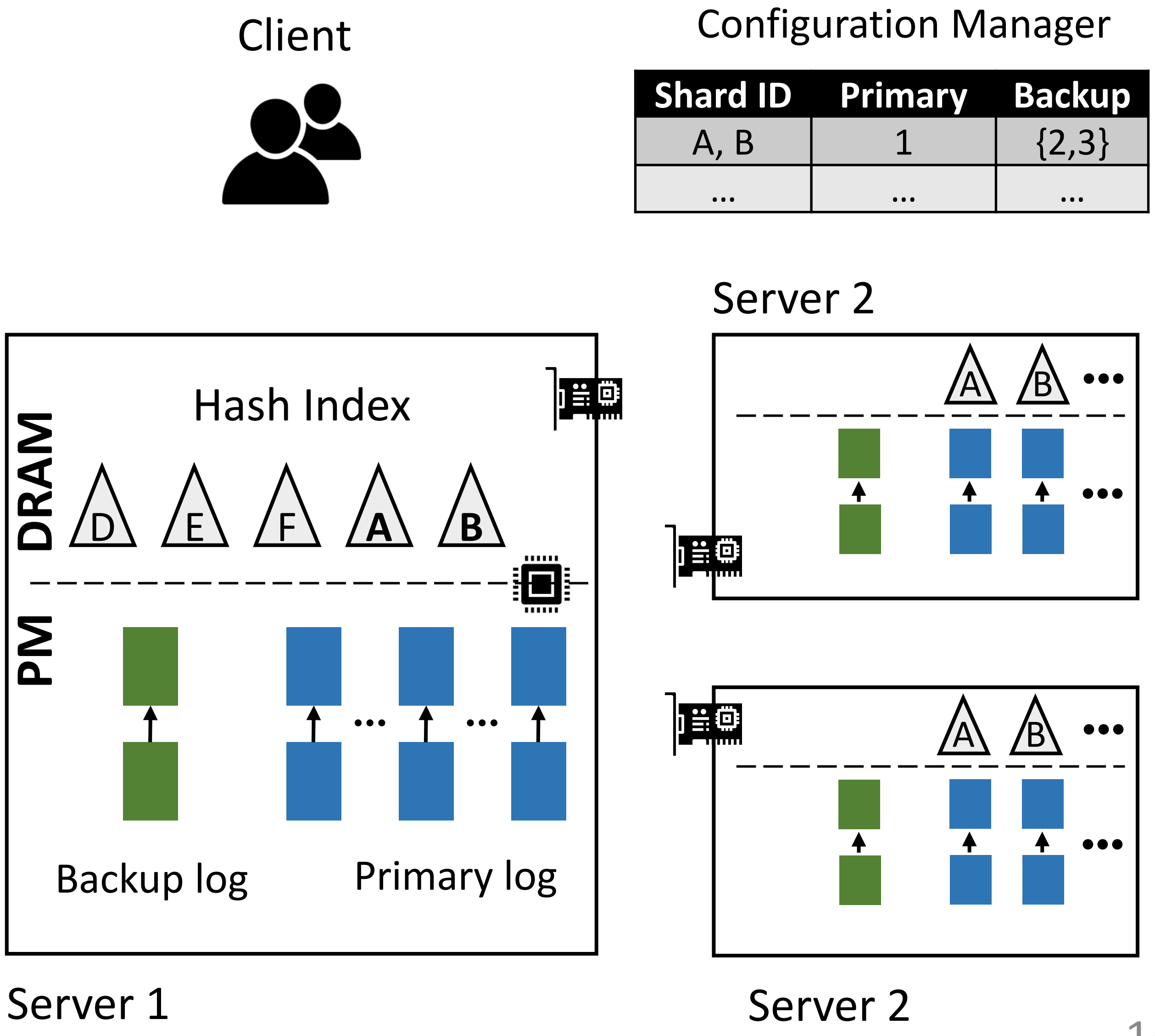
Avoid control thread become bottleneck

- Data path: > 50Mops/s
- Two tasks of control thread：
  - ❶ Push PM buffers to MP SRQ
  - ❷ Poll CQ (RDMA RECV cannot be unsignaled)
- Low overhead RDMA RECV
  - Large recv buffer (e.g., 4MB) using MP features
  - Post a batch of RDMA RECV at a time
- Eliminate CQ polling
  - Like eRPC@NSDI'19
  - Ring-structure CQ and NIC can overwrite CQ entries
  - Flag: *IBV_EXP_CQ_IGNORE_OVERRUN*



Rowan-KV

- Log-structured data layout
- Primary-backup replication

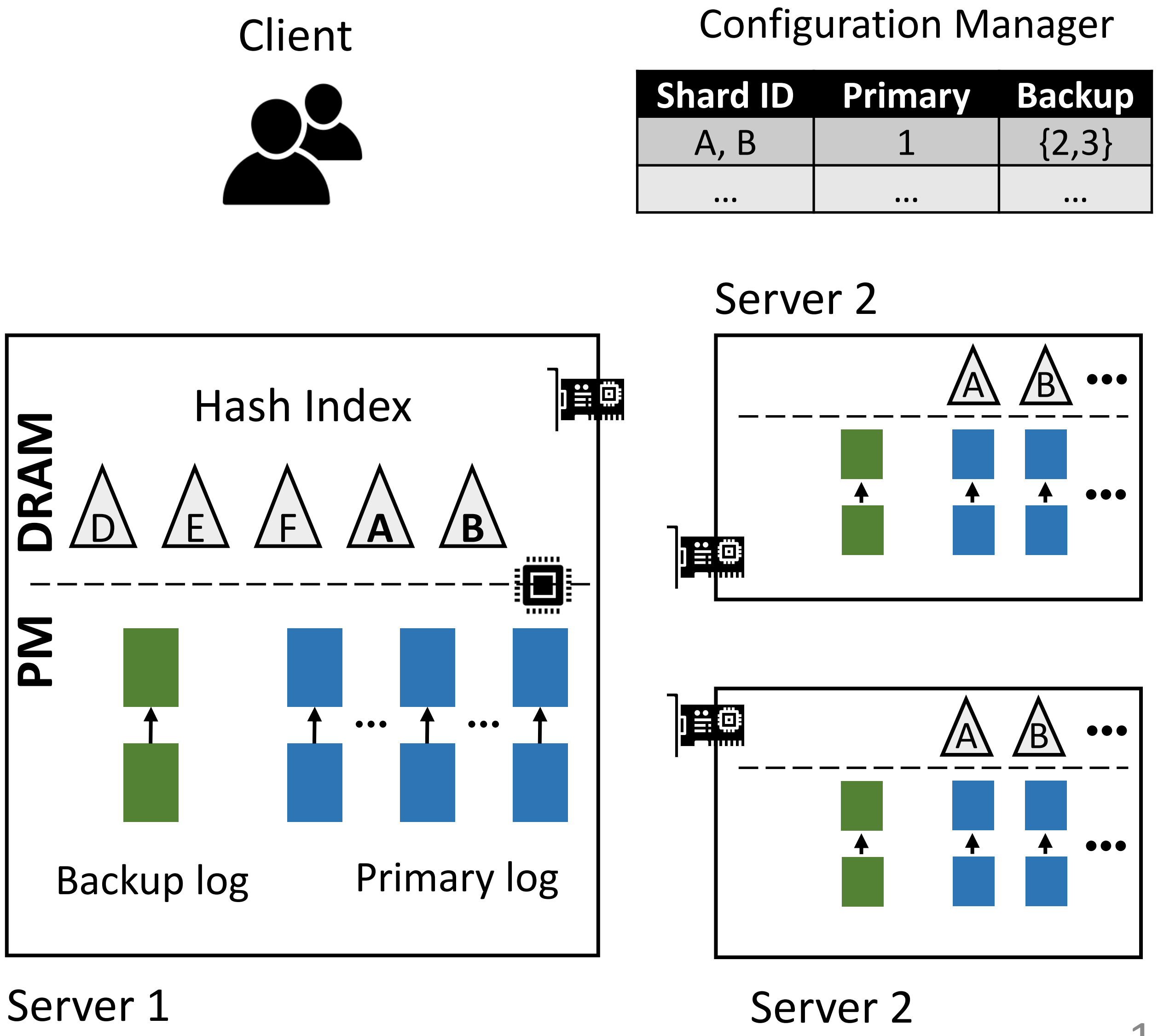


Configuration Manager

Shard ID	Primary	Backup
A, B	1	{2,3}
…	…	…

Rowan-KV

- ❖ Log-structured data layout
- ❖ Primary-backup replication
- ❖ Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes



Configuration Manager

Shard ID	Primary	Backup
A, B	1	{2,3}
…	…	…

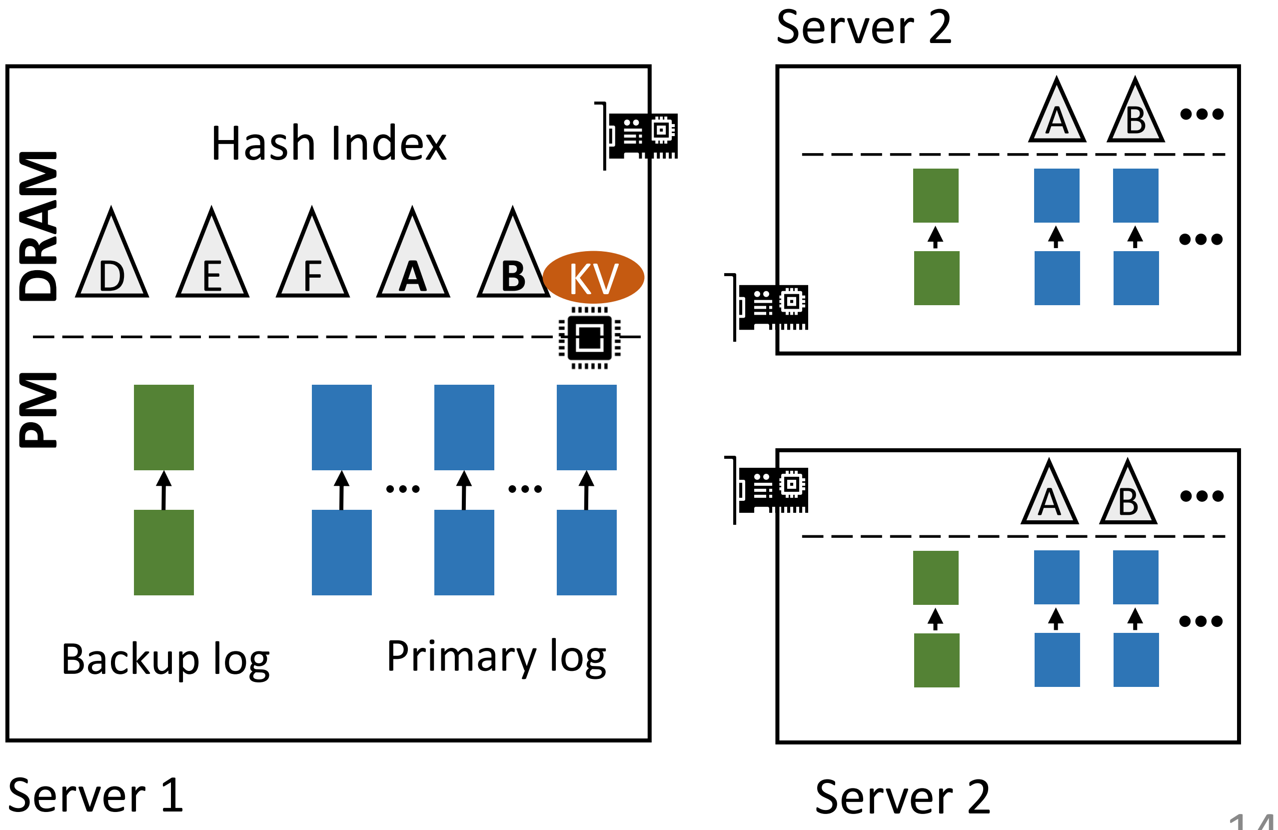
Rowan-KV

- ❖ Log-structured data layout
- ❖ Primary-backup replication
- ❖ Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- ❖ Workflow of a PUT operation
  - ❶ Client sends an RPC to the primary (**P**)

Client

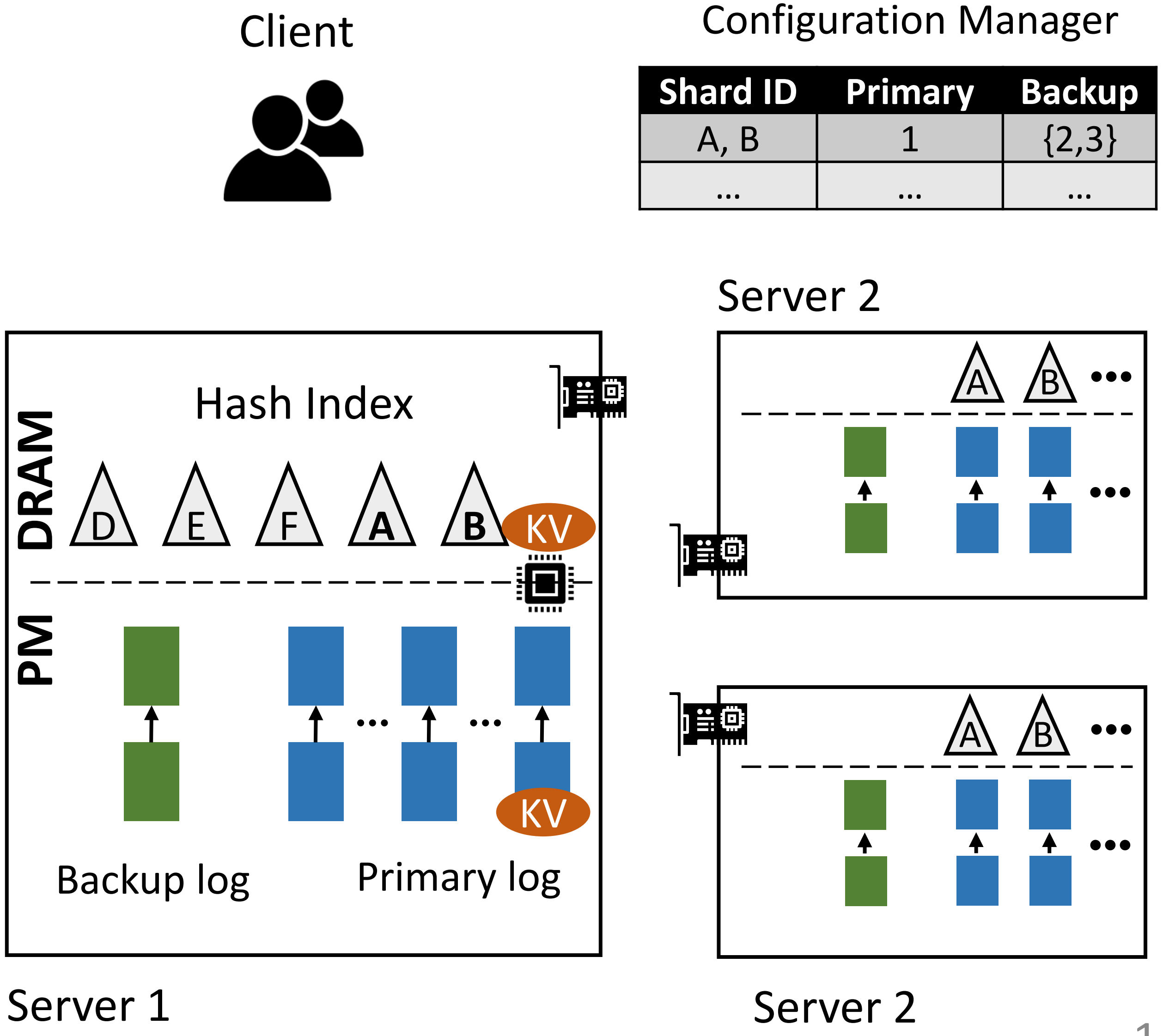
Configuration Manager

Shard ID	Primary	Backup
A, B	1	{2,3}
…	…	…



Rowan-KV

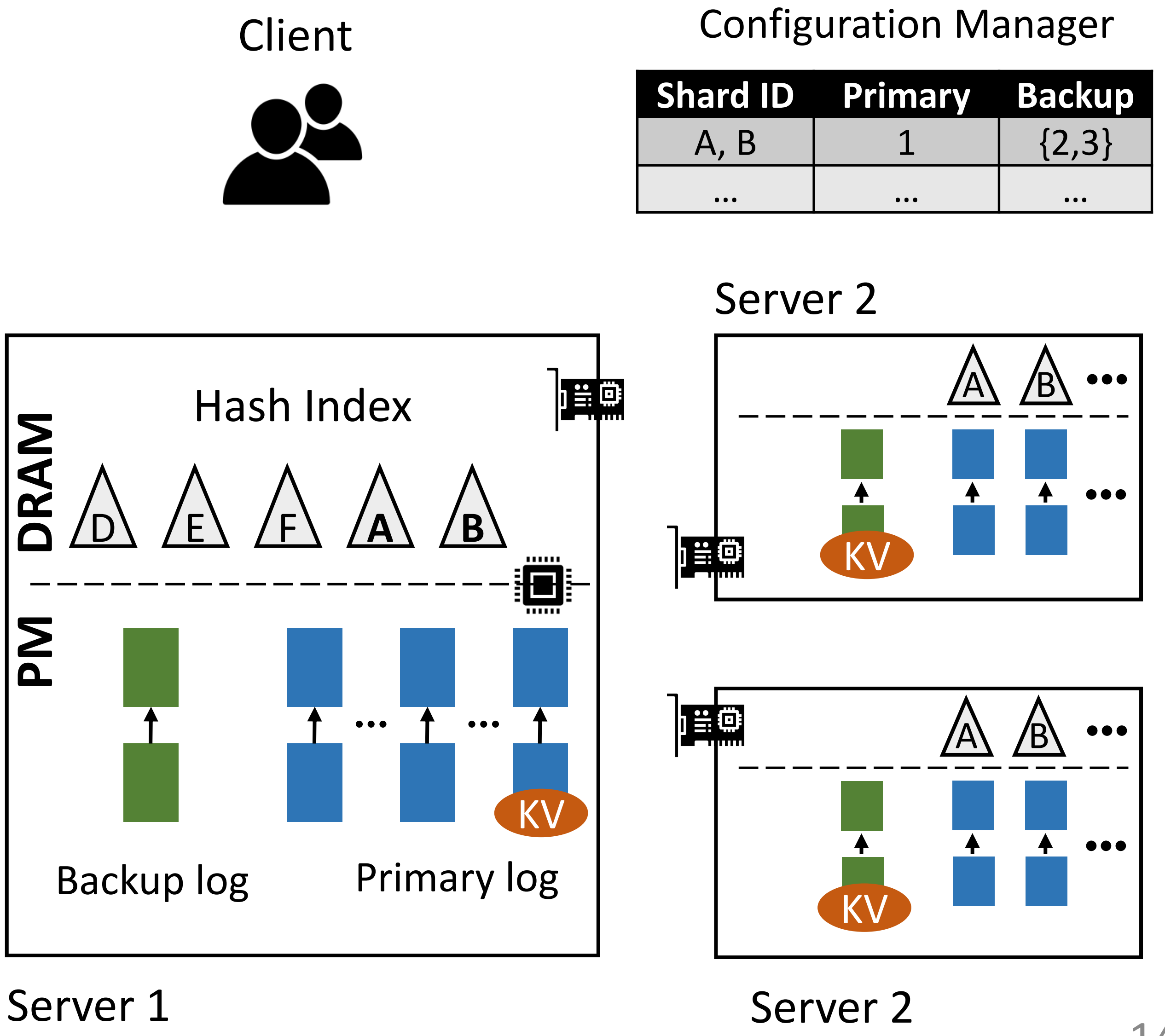
- Log-structured data layout
- Primary-backup replication
- Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- Workflow of a PUT operation
  - ❶ Client sends an RPC to the primary (**P**)
  - ❷ **P** appends an entry **E** to the local primary log



Shard ID	Primary	Backup
A, B	1	{2,3}
…	…	…

Rowan-KV

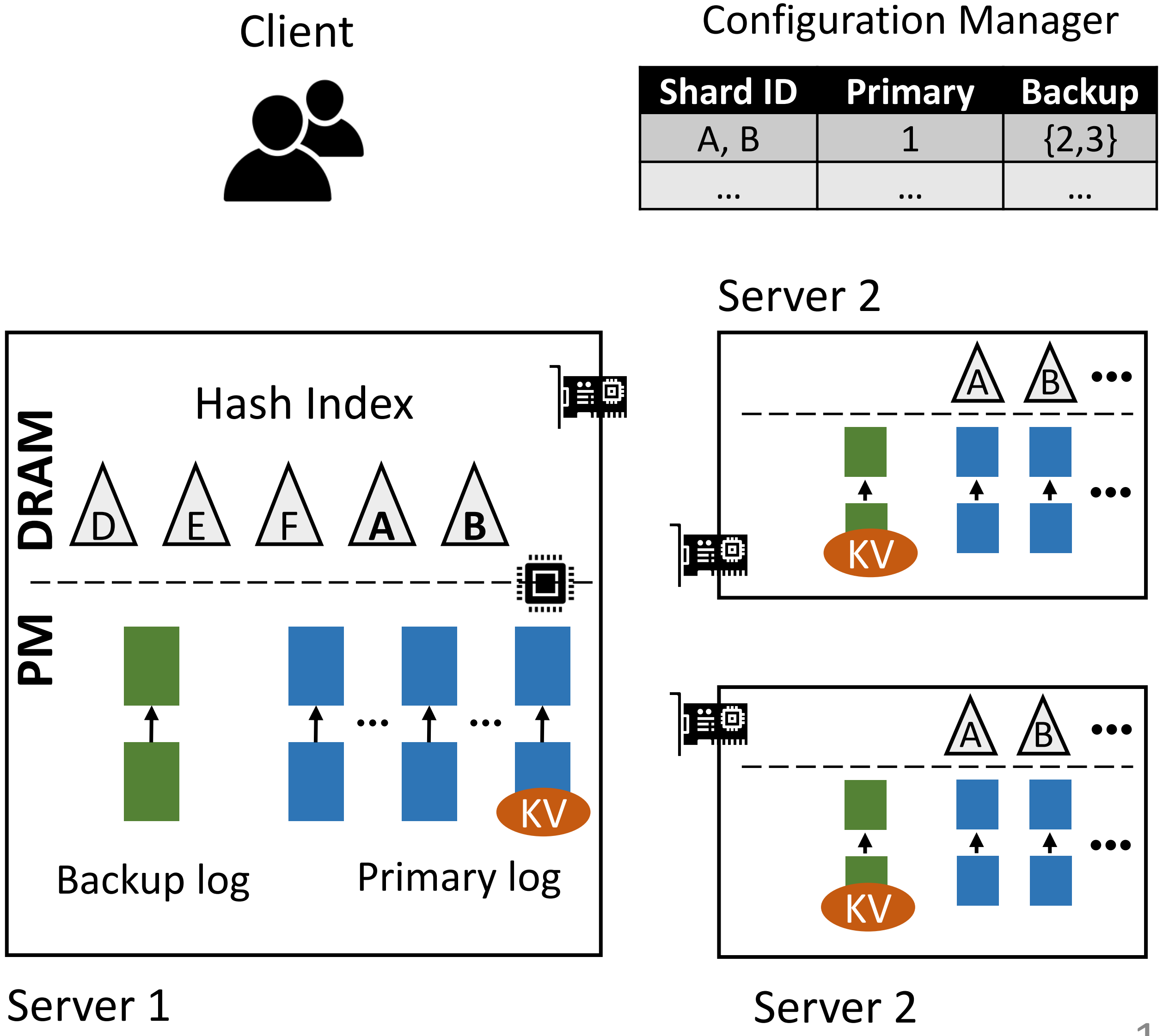
- Log-structured data layout
- Primary-backup replication
- Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- Workflow of a PUT operation
  - ❶ Client sends an RPC to the primary (**P**)
  - ❷ **P** appends an entry **E** to the local primary log
  - ❸ **P** writes **E** to backup logs of all backups via Rowan



Shard ID	Primary	Backup
A, B	1	{2,3}
...	...	...

Rowan-KV

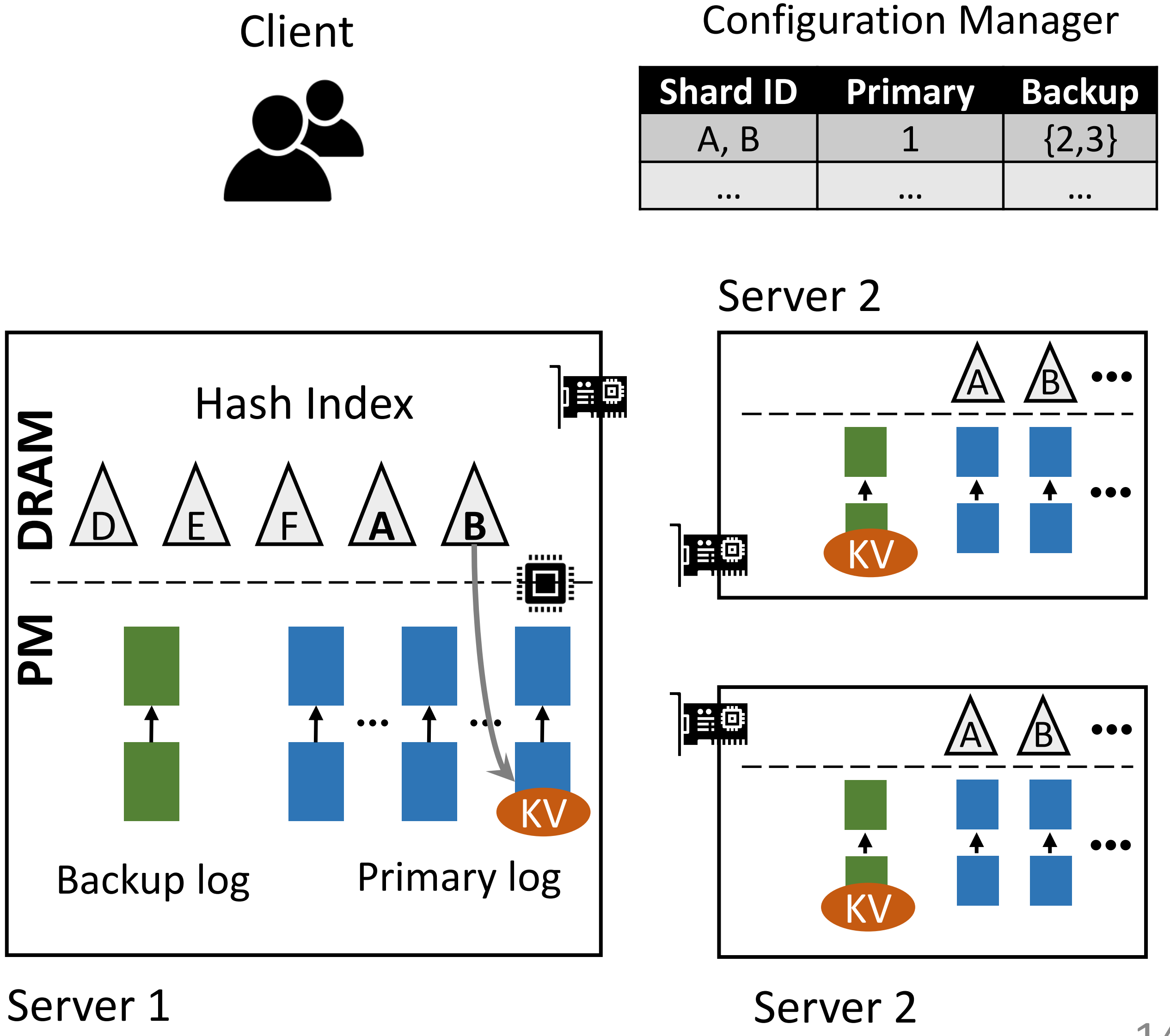
- ❖ Log-structured data layout
- ❖ Primary-backup replication
- ❖ Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- ❖ Workflow of a PUT operation
  - ❶ Client sends an RPC to the primary (**P**)
  - ❷ **P** appends an entry **E** to the local primary log
  - ❸ **P** writes **E** to backup logs of all backups via Rowan
  - ❹ **P** waits for hardware ACKs from backups' NICs



Shard ID	Primary	Backup
A, B	1	{2,3}
…	…	…

Rowan-KV

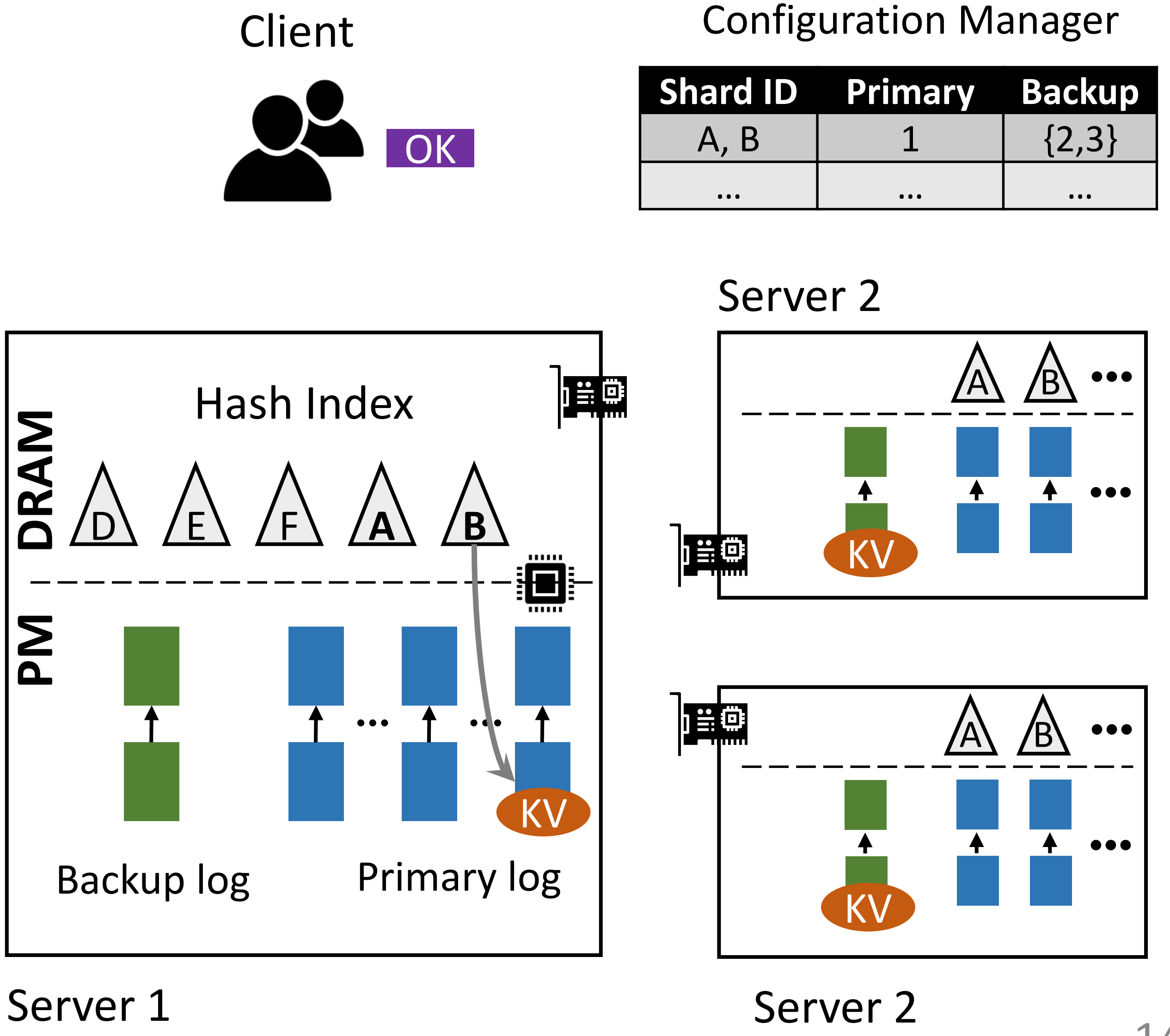
- ❖ Log-structured data layout
- ❖ Primary-backup replication
- ❖ Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- ❖ Workflow of a PUT operation
  - ❶ Client sends an RPC to the primary (**P**)
  - ❷ **P** appends an entry **E** to the local primary log
  - ❸ **P** writes **E** to backup logs of all backups via Rowan
  - ❹ **P** waits for hardware ACKs from backups' NICs
  - ❺ **P** updates index, pointing to **E** in primary log



Shard ID	Primary	Backup
A, B	1	{2,3}
…	…	…

Rowan-KV

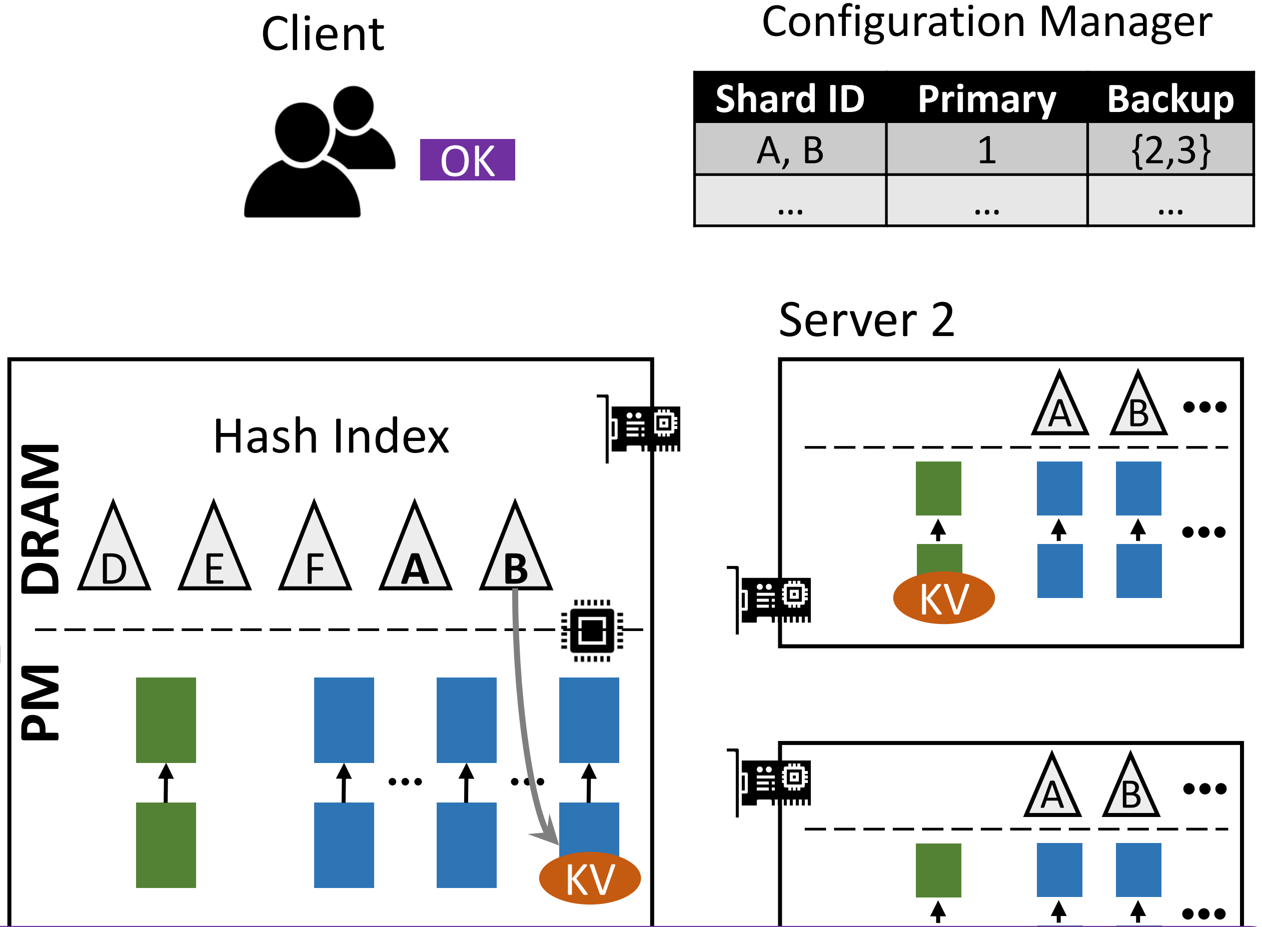
- ❖ Log-structured data layout
- ❖ Primary-backup replication
- ❖ Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- ❖ Workflow of a PUT operation
  - ❶ Client sends an RPC to the primary (**P**)
  - ❷ **P** appends an entry **E** to the local primary log
  - ❸ **P** writes **E** to backup logs of all backups via Rowan
  - ❹ **P** waits for hardware ACKs from backups' NICs
  - ❺ **P** updates index, pointing to **E** in primary log
  - ❻ **P** returns a response



Shard ID	Primary	Backup
A, B	1	{2,3}
…	…	…

Rowan-KV

- ❖ Log-structured data layout
- ❖ Primary-backup replication
- ❖ Three components per server
  - A single backup log managed by one Rowan instance
  - Per-thread primary logs
  - Per-shard DRAM hash indexes
- ❖ Workflow of a PUT operation
  - ❶ Client sends an RPC to the primary (**P**)
  - ❷ **P** appends an entry **E** to the local primary log
  - ❸ **P** writes **E** to backup logs of all backups via Rowan
  - ❹ **P** waits for hardware ACKs from backups' NICs
  - ❺ **P** updates index, pointing to **E** in primary log
  - ❻ **P** returns a response



Configuration Manager

Shard ID	Primary	Backup
A, B	1	{2,3}
…	…	…

1) Low latency : One-sided replication

2) Low DLWA : Log-structured & Rowan merges replication writes into a single backup log

More Design Details : Check Our Paper

Digest and Garbage Collection

- Reserve dedicated threads, RAMCloud-style GC

Failover

- FaRM's reconfiguration-style approach

Dynamic Resharding

- Shard-level migration

Fast Remote Persistency with disabled DDIO

- Prefetching、Reducing PCIe Txns

Replicating Persistent Memory Key-Value Stores with Efficient RDMA Abstraction

Qing Wang, Youyou Lu, Jing Wang, and Jiwu Shu*

Tsinghua University

Abstract

Combining persistent memory (PM) with RDMA is a promising approach to performant replicated distributed key-value stores (KVSs). However, existing replication approaches do not work well when applied to PM KVSs: 1) Using RPC induces software queueing and execution at backups, increasing request latency; 2) Using one-sided RDMA `WRITE` causes many streams of small PM writes, leading to severe device-level write amplification (DLWA) on PM.

In this paper, we propose Rowan, an efficient RDMA abstraction to handle replication writes in PM KVSs; it aggregates concurrent remote writes from different servers, and lands these writes to PM in a sequential (thus low DLWA) and one-sided (thus low latency) manner. We realize Rowan with off-the-shelf RDMA NICs. Further, we build Rowan-KV, a log-structured PM KVS using Rowan for replication. Evaluation shows that under write-intensive workloads, compared with PM KVSs using RPC and RDMA `WRITE` for replication, Rowan-KV boosts throughput by 1.22× and 1.39× as well as lowers median `PUT` latency by 1.77× and 2.11×, respectively, while largely eliminating DLWA.

1 Introduction

Replicated distributed key-value stores (KVSs) support many applications by providing durability and high availability [28, 56, 76]. The recent commercialization of persistent memory (PM), e.g., Intel's Optane DIMMs, enables local storage with extremely low latency (e.g., ~100ns when persisting small data [73]). When building replicated distributed KVSs with such fast storage media, network and CPU will become determinants of request latency, since replicating an object (i.e., key-value pair) involves multiple times of network communication and request queueing/execution.

RDMA, a widely-deployed network technology [34, 37, 53], is promising to mitigate the network and CPU overhead. First, RDMA delivers low latency (~2μs) due to protocol-offload RDMA NICs (RNICs) and kernel-bypass software. Second, RDMA provides one-sided `WRITE` and `READ`, allowing remote memory accesses without involvement of remote CPUs. Recent work have leveraged `WRITE` to replicate data in DRAM (i.e., `WRITE`-enabled replication) [17, 30, 31, 69]. This eliminates software queueing/execution of backups in the critical path, thus significantly cutting the replication latency compared with RPC-enabled replication.

Yet, in the context of PM KVSs, `WRITE`-enabled replication approach does not work well: it induces severe device-level write amplification (DLWA) on PM. Specifically, a KVS is typically finely sharded for load balancing and fast recovery, so every server acts as backups for many shards, receiving numerous concurrent replication writes from many remote threads; besides, these replication writes are typically small (~100B) due to prevalent tiny objects in real-world workloads [24, 52]. In `WRITE`-enabled replication approaches (e.g., FaRM [31]), each server allocates an exclusive backup log for *every remote thread*, to accommodate remote `WRITE` from primaries. When adopting `WRITE`-enabled replication to PM KVSs, these backup logs generate a huge number of PM write streams[1], which contain lots of small-sized writes. These numerous write streams lead to severe DLWA, since PM has block access granularity at media level (e.g., 256B in Optane DIMMs) and its hardware combining capacity is bounded. In our experiments, with 128B RDMA `WRITE`, 144 remote PM write streams cause 1.58× DLWA (§2.4). DLWA wastes limited PM write bandwidth, shortens PM lifetime, and harms PM's persistence efficiency.

In this paper, we propose Rowan, an efficient RDMA abstraction to handle replication writes on PM KVSs. Rowan can aggregate numerous concurrent remote writes from different servers, and land these writes to PM *sequentially*, so as to largely eliminate DLWA. Besides, it is one-sided as RDMA `WRITE`, enabling backup-passive replication with low latency and high CPU efficiency. We realize Rowan with off-the-shelf RNICs based on two observations: 1) RDMA `SEND` is two-sided on the control path but *one-sided on the data path*; 2) RNICs consume receive buffers *in order*. Thus, we let a control thread at the receiver side push PM-resident buffers into receive queues *in increasing address order*. Senders only need to issue `SEND` for remote PM writes and wait for ACKs generated by receiver-side RNICs. We leverage two RNIC hardware features, shared receive queue (SRQ) [11] and multi-packet receive queue (MP RQ) [7, 9], to merge writes from different connections and support variable-sized writes, respectively. We also streamline Rowan's control path by minimizing the control thread's tasks. A Rowan instance can

---

*Jiwu Shu is the corresponding author (shujw@tsinghua.edu.cn)

[1] A *write stream* is a group of writes targeting contiguous addresses, e.g., writes that perform log appending.

Experimental Setup

Hardware Platform

- 6 machines as servers
- Intel Xeon Gold 6240M CPU (18 physical/36 logical cores)
- 3 × 256GB Optane DIMMs (6GB/s writes, 18 GB/s reads)
- 100Gbps Mellanox ConnectX-5 NIC

Software Setting

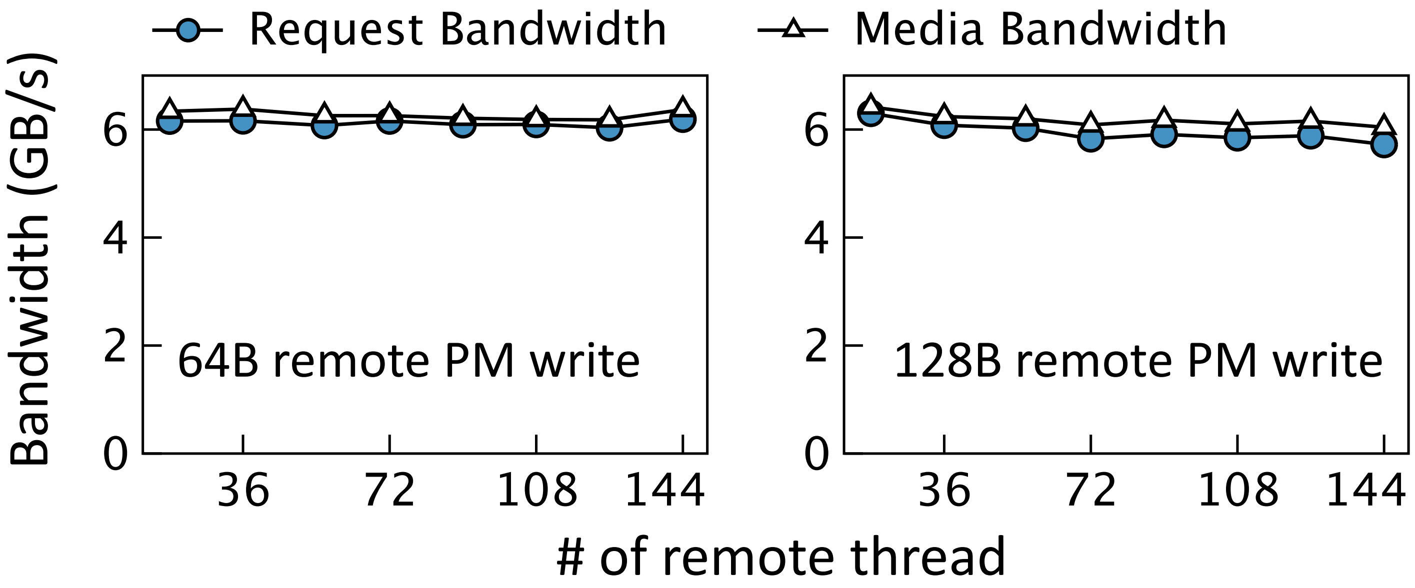
- 24 cores for worker threads; 5/6/1 cores for digest/GC/control
- Replication factor: 3
- Each server holds 48 shards
- Disable DDIO and send 1B RDMA READ for persistency of RDMA WRITE or Rowan

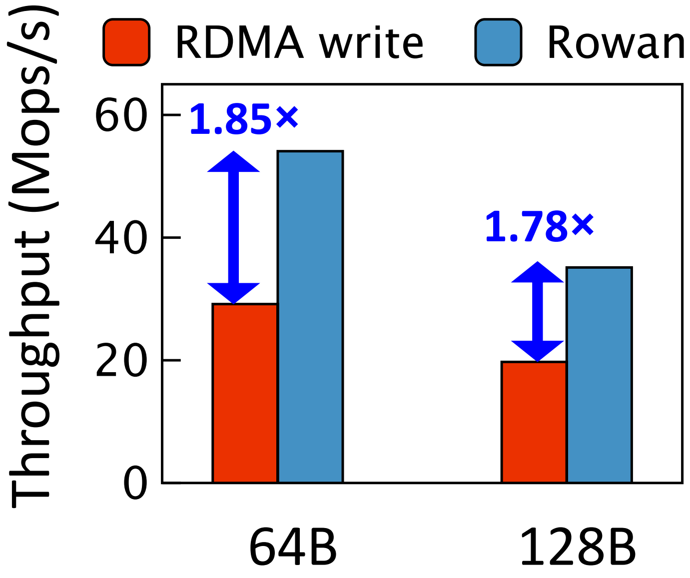
Performance of Rowan

- Remote threads concurrently perform PM writes to a PM server via one Rowan instance
- In the PM server, 18 cores perform local sequential PM writes

Performance of Rowan

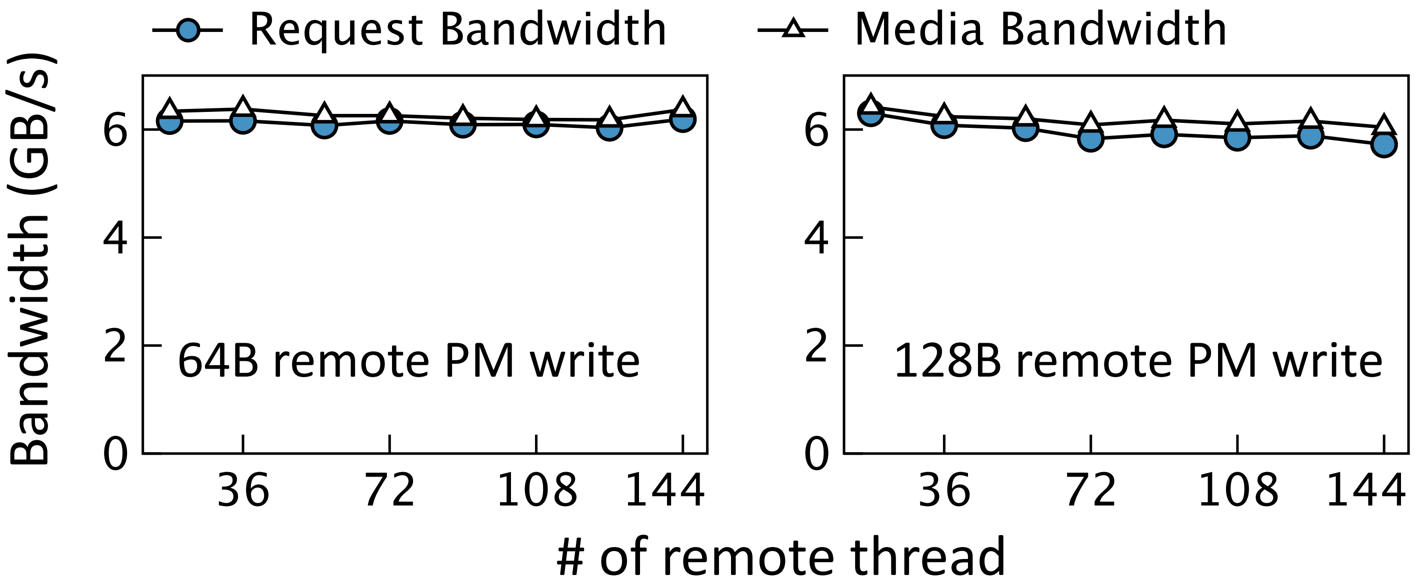
- Remote threads concurrently perform PM writes to a PM server via one Rowan instance
- In the PM server, 18 cores perform local sequential PM writes

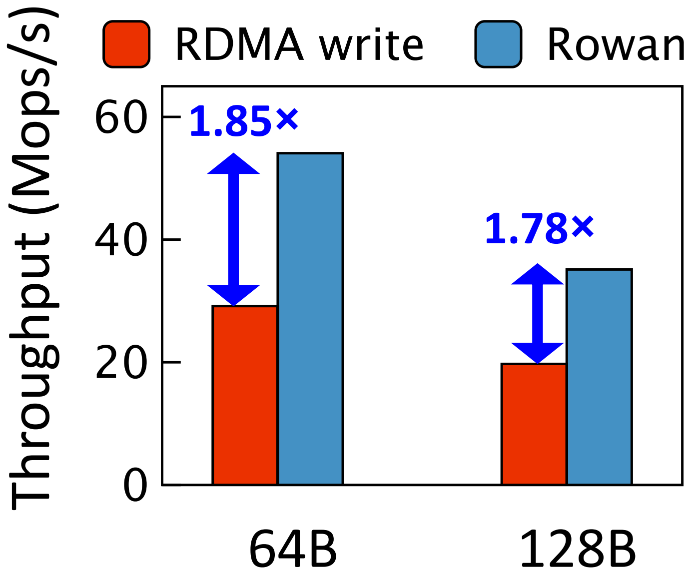




Performance of Rowan

- Remote threads concurrently perform PM writes to a PM server via one Rowan instance
- In the PM server, 18 cores perform local sequential PM writes

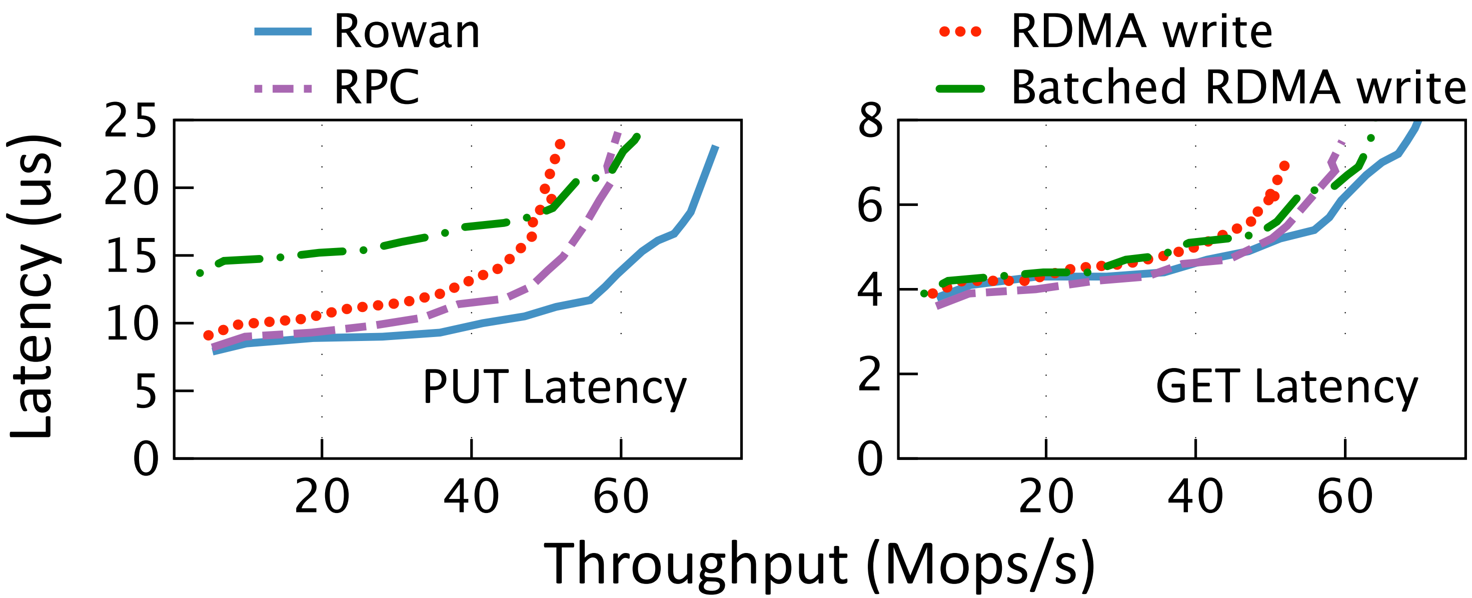




Rowan can largely eliminate device-level write amplification (DLWA), and thus has higher (1.85X) throughput than RDMA WRITE

Performance of Rowan-KV

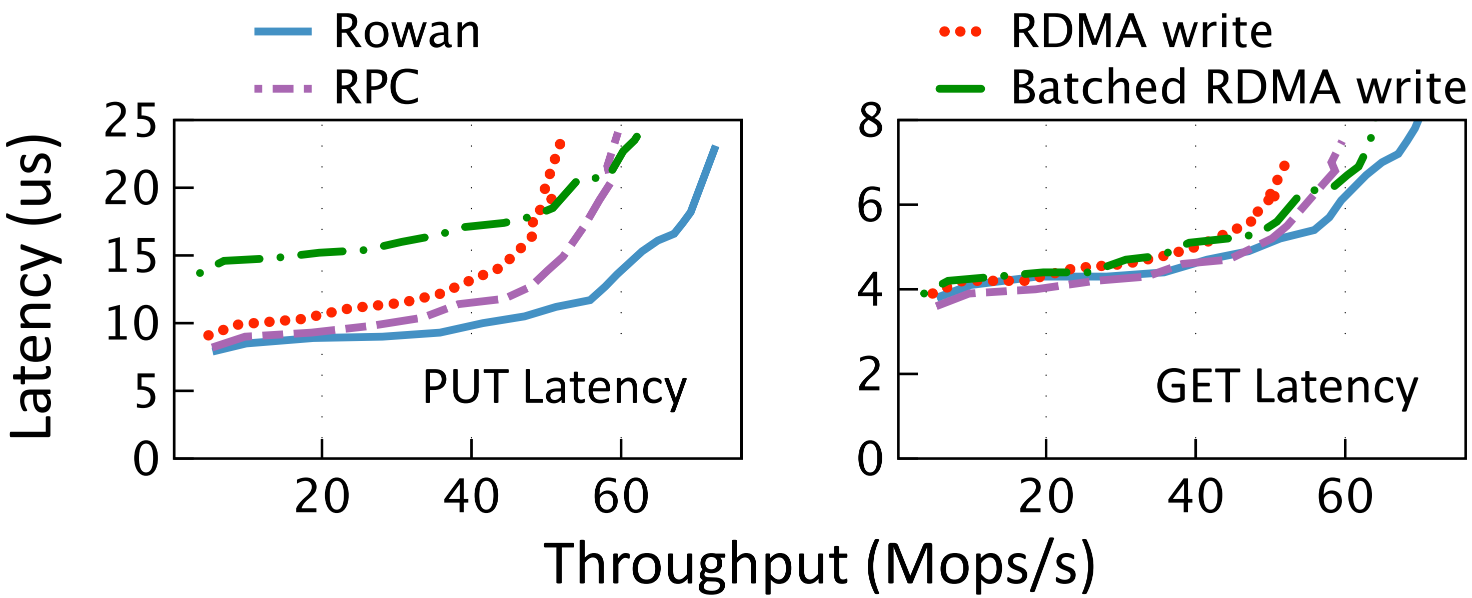
- Compare it with KVSs using different replication approaches (6 severs, 8 clients)
- PUT/GET: 50%/50%; Object size: Facebook ZippyDB (avg. 90.8B)
- Batched RDMA write: 5us timeout or 256B batched writes



(a) Throughout vs. Latency

Performance of Rowan-KV

- Compare it with KVSs using different replication approaches (6 severs, 8 clients)
- PUT/GET: 50%/50%; Object size: Facebook ZippyDB (avg. 90.8B)
- Batched RDMA write: 5us timeout or 256B batched writes

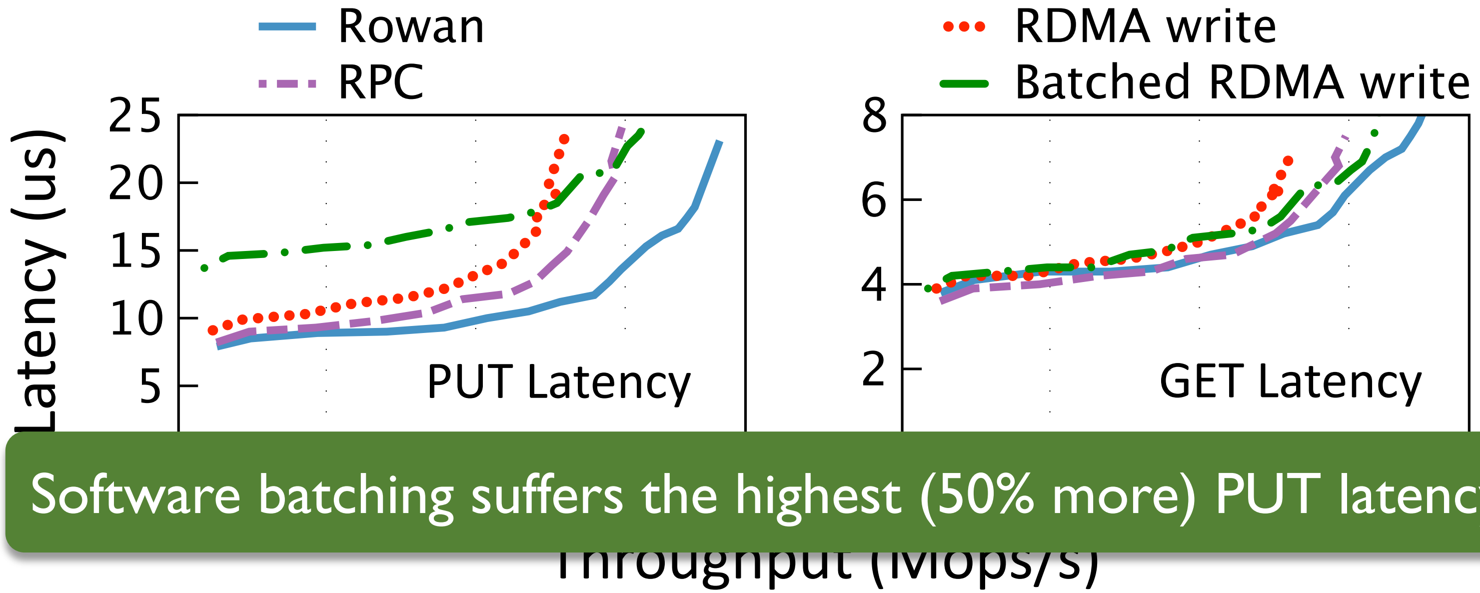


(a) Throughout vs. Latency

Under write-intensive workloads, compared with RPC and RDMA WRITE, Rowan boosts KVS's throughput (by 1.2X and 1.4X) & reduces PUT latency (by 1.8X and 2.1X)

Performance of Rowan-KV

- Compare it with KVSs using different replication approaches (6 severs, 8 clients)
- PUT/GET: 50%/50%; Object size: Facebook ZippyDB (avg. 90.8B)
- Batched RDMA write: 5us timeout or 256B batched writes

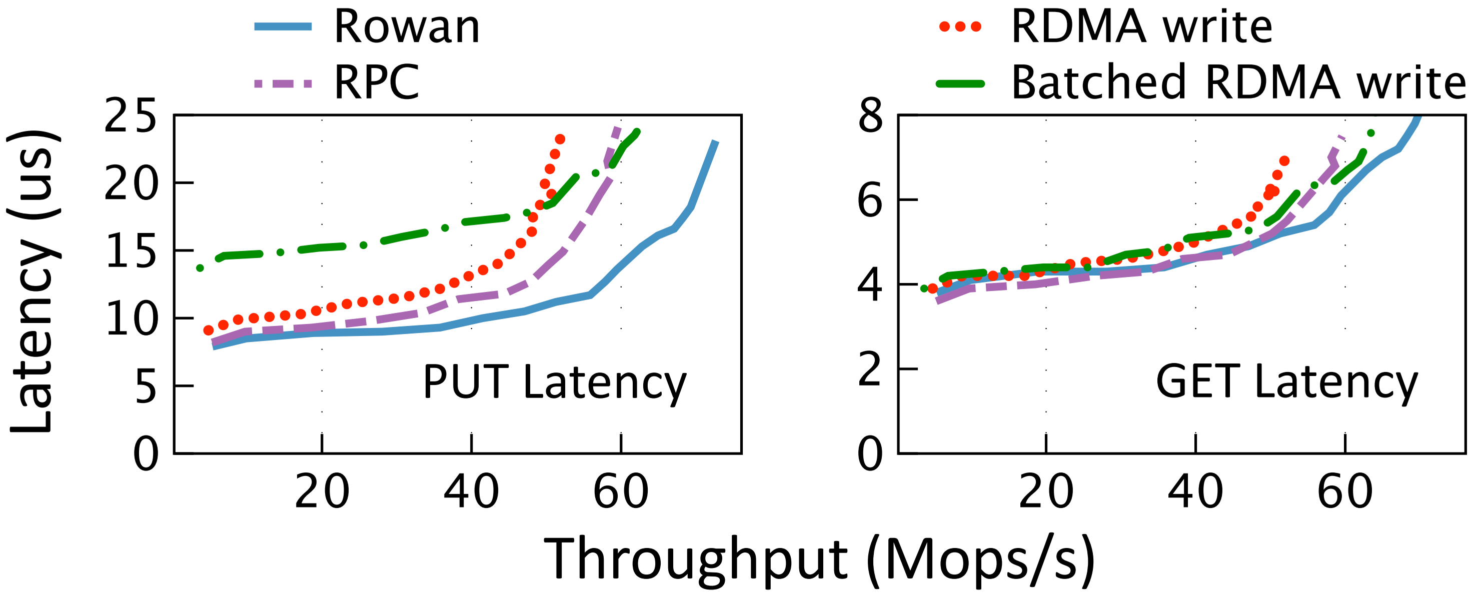


Software batching suffers the highest (50% more) PUT latency

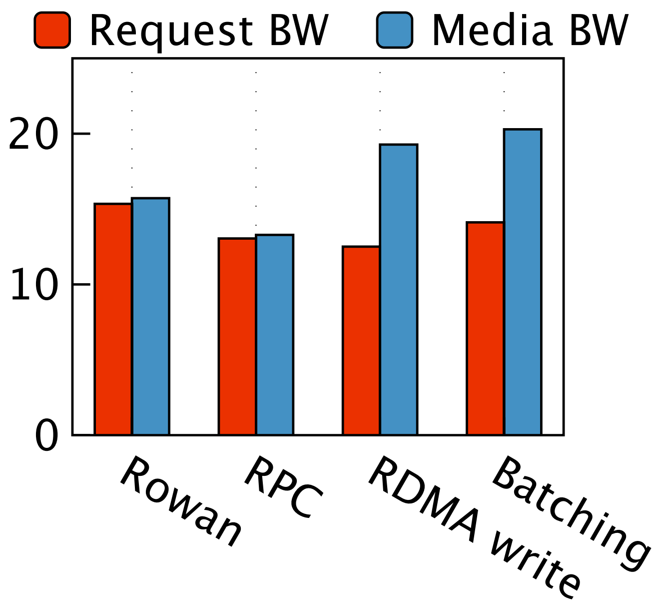
(a) Throughout vs. Latency

Performance of Rowan-KV

- ❖ Compare it with KVSs using different replication approaches (6 severs, 8 clients)
- ❖ PUT/GET: 50%/50%; Object size: Facebook ZippyDB (avg. 90.8B)
- ❖ Batched RDMA write: 5us timeout or 256B batched writes



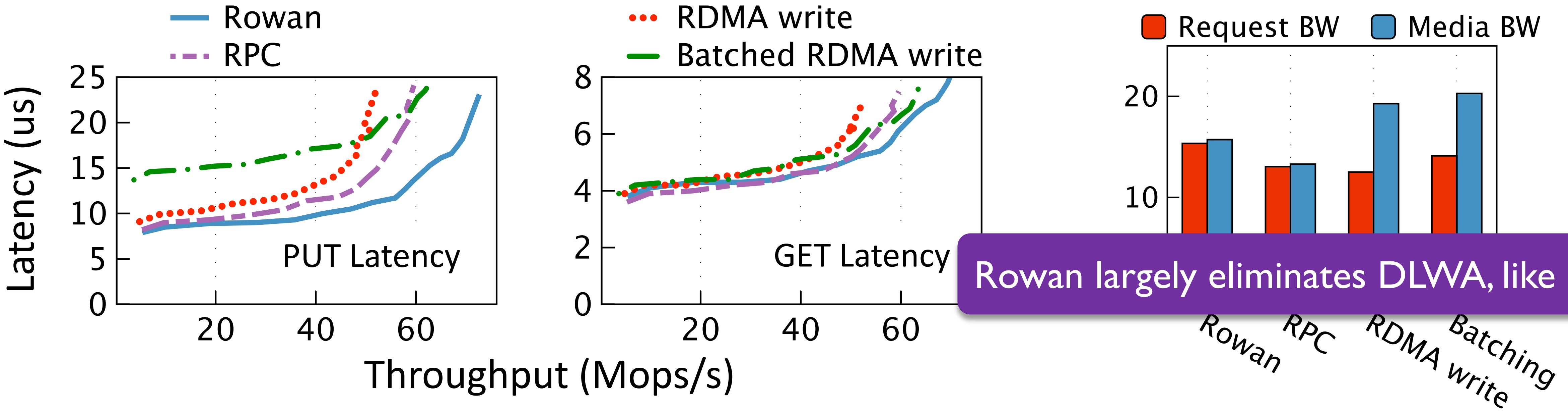
(a) Throughout vs. Latency



(b) DLWA

Performance of Rowan-KV

- Compare it with KVSs using different replication approaches (6 severs, 8 clients)
- PUT/GET: 50%/50%; Object size: Facebook ZippyDB (avg. 90.8B)
- Batched RDMA write: 5us timeout or 256B batched writes



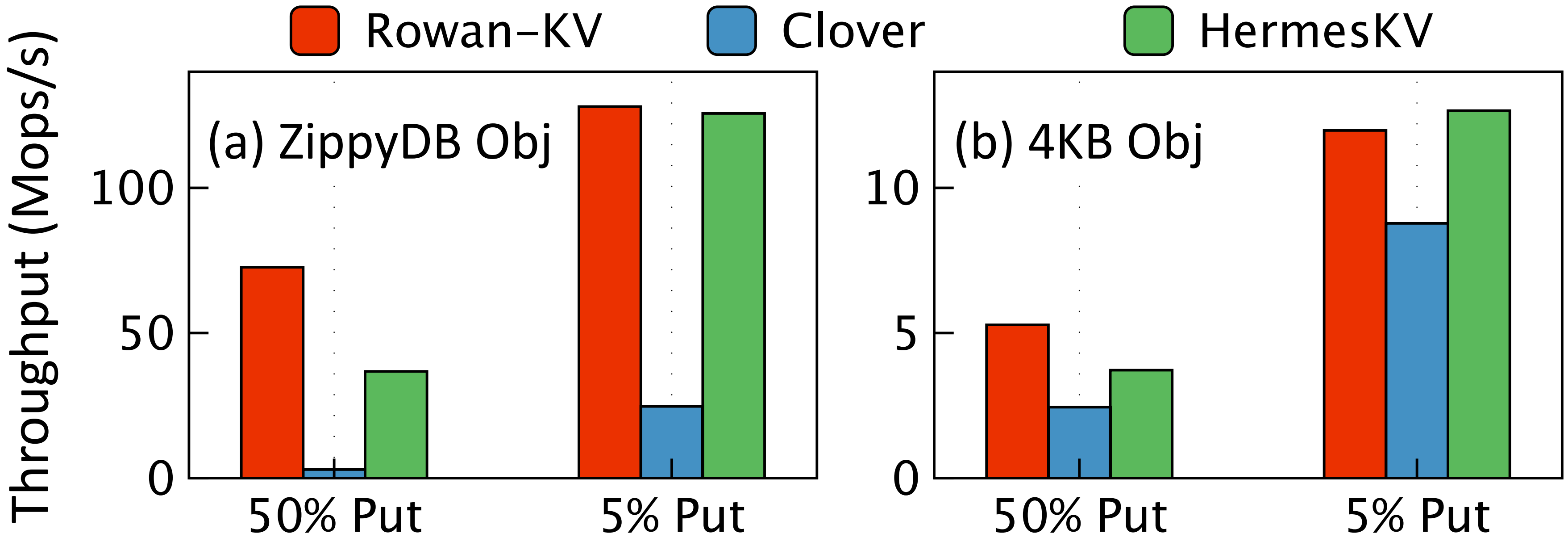
(a) Throughout vs. Latency

(b) DLWA

Rowan largely eliminates DLWA, like RPC

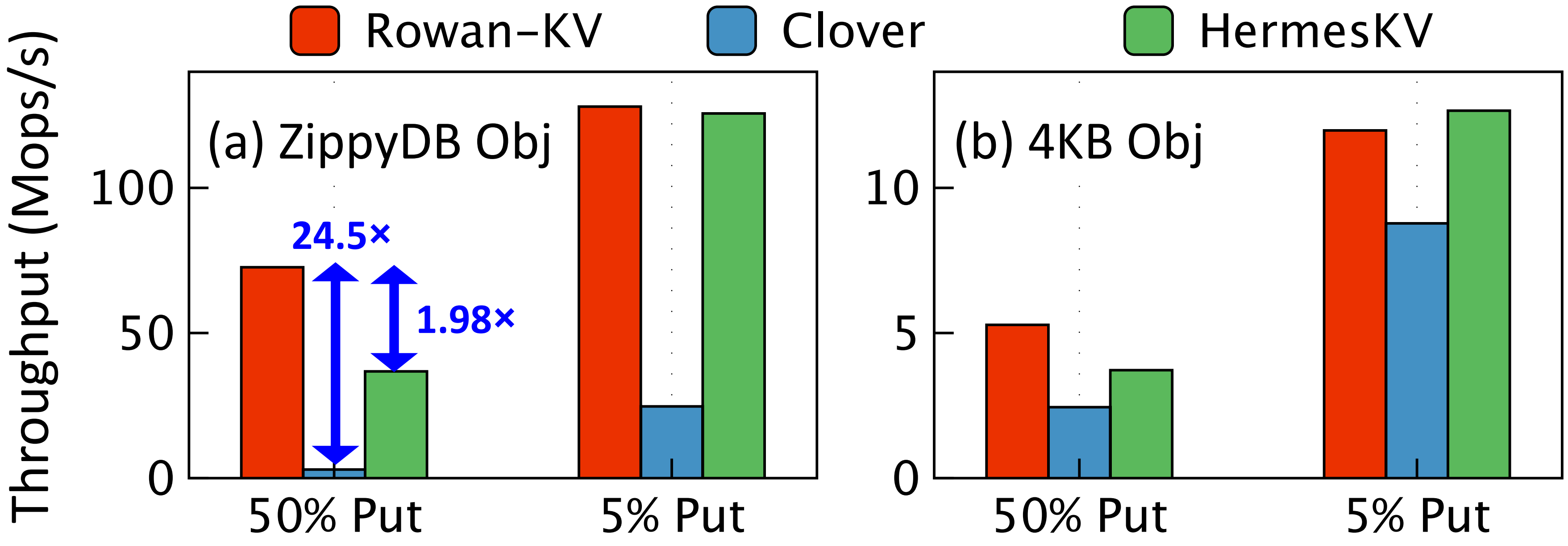
Performance Comparison with Other KVSs

- Clover [ATC'20]: one-sided READ/WRITE for replication
- HermesKV [ASPLOS'20]: broadcast replication protocol via RPC
- 6 Servers



Performance Comparison with Other KVSs

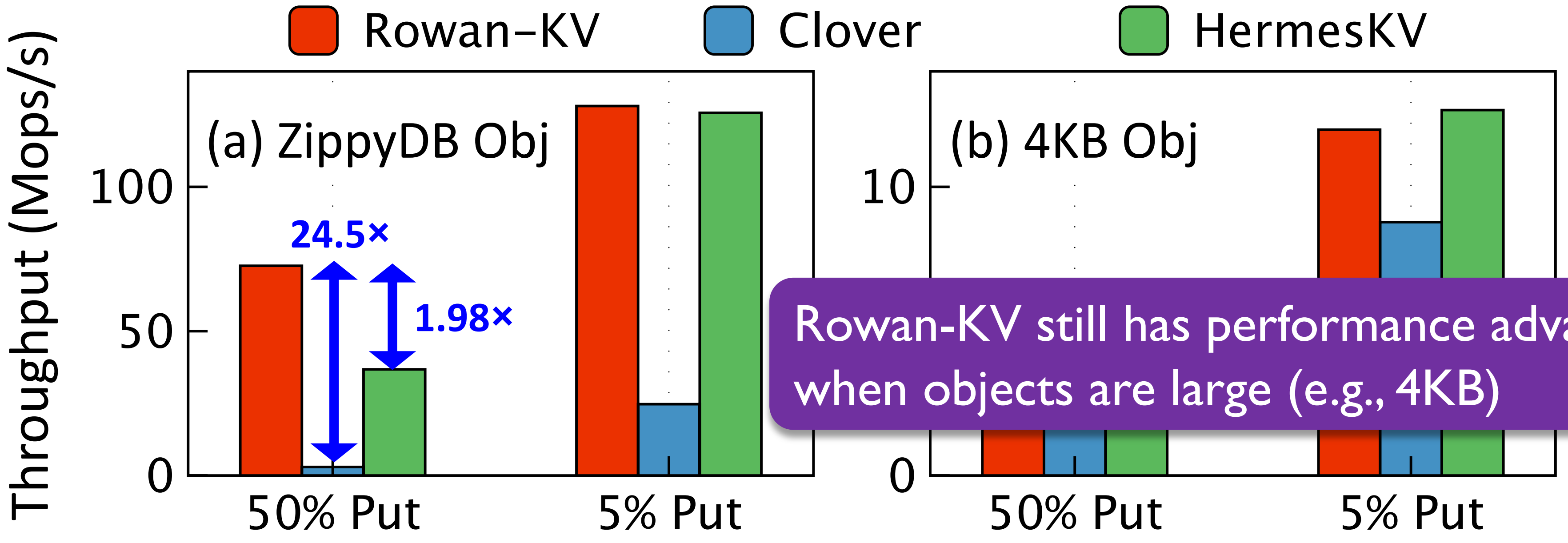
- Clover [ATC'20]: one-sided READ/WRITE for replication
- HermesKV [ASPLOS'20]: broadcast replication protocol via RPC
- 6 Servers



Under write-intensive workloads (i.e., 50% PUT), Rowan-KV outperforms Clover and HermesKV significantly (24.5X and 1.98X) when objects are small

Performance Comparison with Other KVSs

- Clover [ATC'20]: one-sided READ/WRITE for replication
- HermesKV [ASPLOS'20]: broadcast replication protocol via RPC
- 6 Servers



Rowan-KV still has performance advantages when objects are large (e.g., 4KB)

Under write-intensive workloads (i.e., 50% PUT), Rowan-KV outperforms Clover and HermesKV significantly (24.5X and 1.98X) when objects are small

Conclusion

- One-sided replication can achieve extreme low latency
    - Remove software latency of backups (RPC queueing, CPU execution) from the critical path

Conclusion

- One-sided replication can achieve extreme low latency
    - Remove software latency of backups (RPC queueing, CPU execution) from the critical path
- RDMA WRITE for replication induces severe device-level write amplification on PM
    - Pre-allocate many logs for remote threads
    - Small objects in workloads vs. block-level internal access granularity in PM devices

Conclusion

- One-sided replication can achieve extreme low latency
  - Remove software latency of backups (RPC queueing, CPU execution) from the critical path
- RDMA WRITE for replication induces severe device-level write amplification on PM
  - Pre-allocate many logs for remote threads
  - Small objects in workloads vs. block-level internal access granularity in PM devices
- We propose Rowan, a one-sided RDMA abstraction
  - Translating concurrent remote small writes into a single write stream
  - Rowan-based KVS achieves high performance, while largely eliminating DLWA

Conclusion

- One-sided replication can achieve extreme low latency
  - Remove software latency of backups (RPC queueing, CPU execution) from the critical path

- RDMA WRITE for replication induces severe device-level write amplification on PM
  - Pre-allocate many logs for remote threads
  - Small objects in workloads vs. block-level internal access granularity in PM devices

- We propose Rowan, a one-sided RDMA abstraction
  - Translating concurrent remote small writes into a single write stream
  - Rowan-based KVS achieves high performance, while largely eliminating DLWA

- Takeaway
  - For one-sided writes, receiver-side NIC is good at managing storage/memory devices

    1) It can coordinate requests from different senders

    2) It can allocate addresses according to features of storage/memory devices

Thanks & QA

Replicating Persistent Memory Key-Value Stores with Efficient RDMA Abstraction

Contact Information: wq1997@tsinghua.edu.cn