FUSEE: A Fully Memory-Disaggregated Key-Value Store

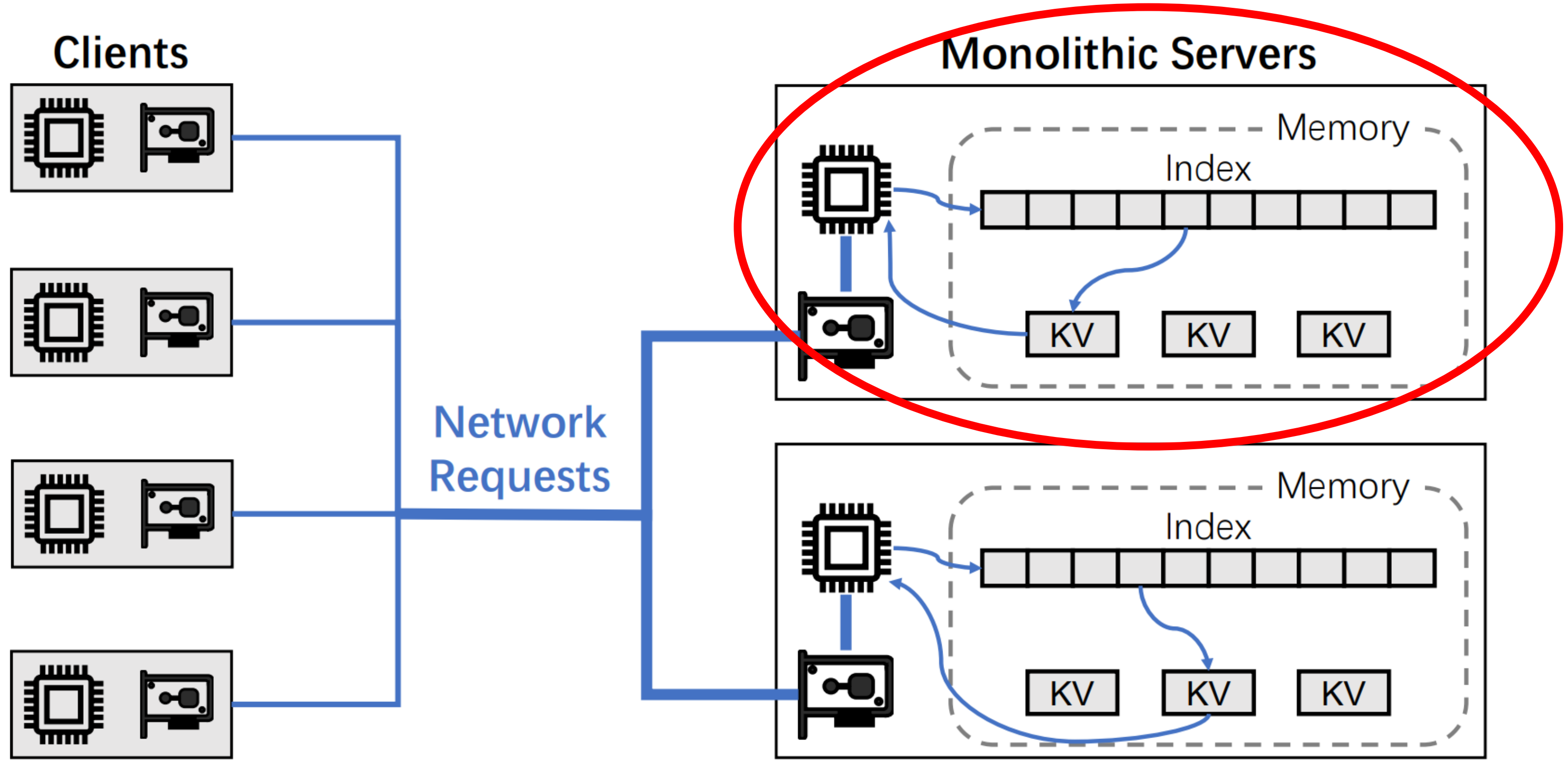
Jiacheng Shen, Pengfei Zuo, Xuchuan Luo, Tianyi Yang , Yuxin Su, Yangfan Zhou, ανδ Michael . Lyu

Speaker wrl

FAST 2023

Background

- **In-Memory Key-Value Stores**

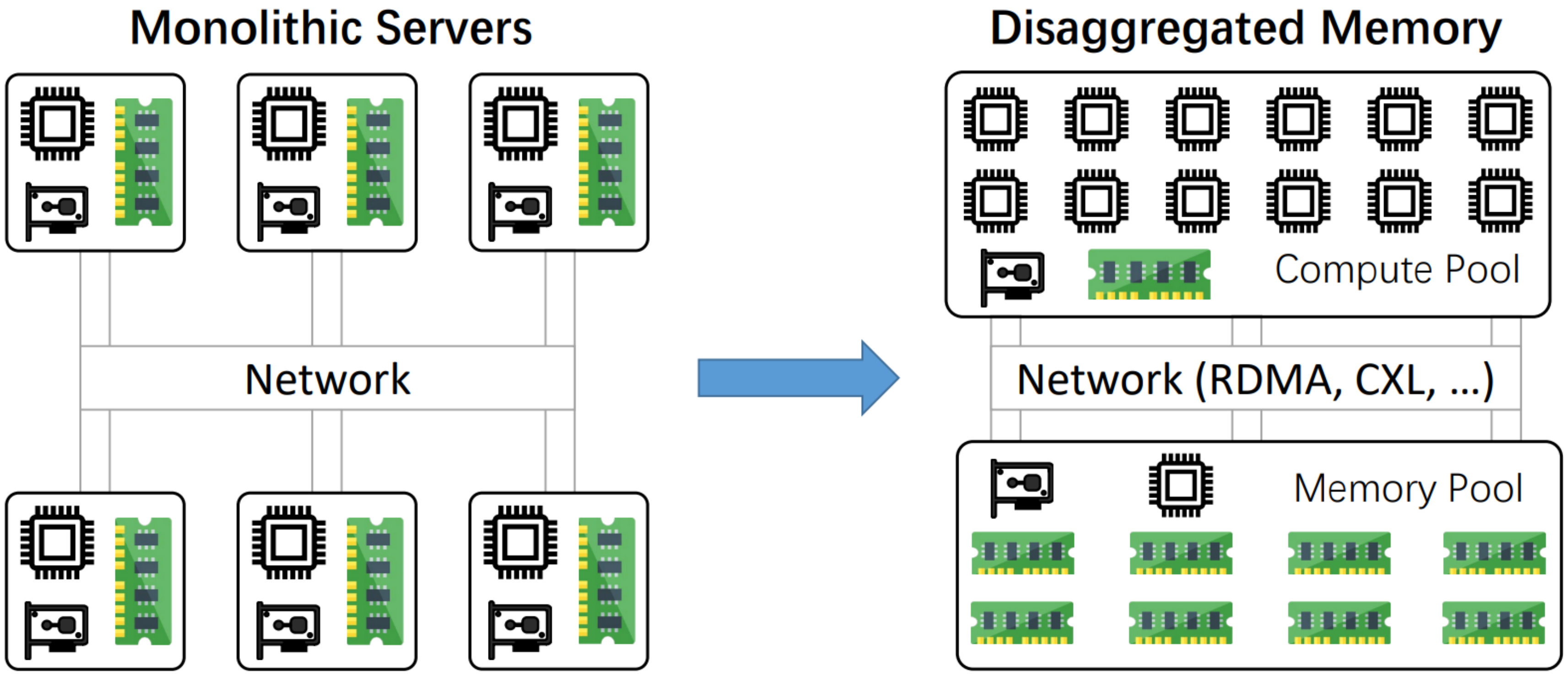


- **Existing Problem:**

Coupled CPU and Memory will be accompanied by low Resource efficiency and Elasticity.

Background

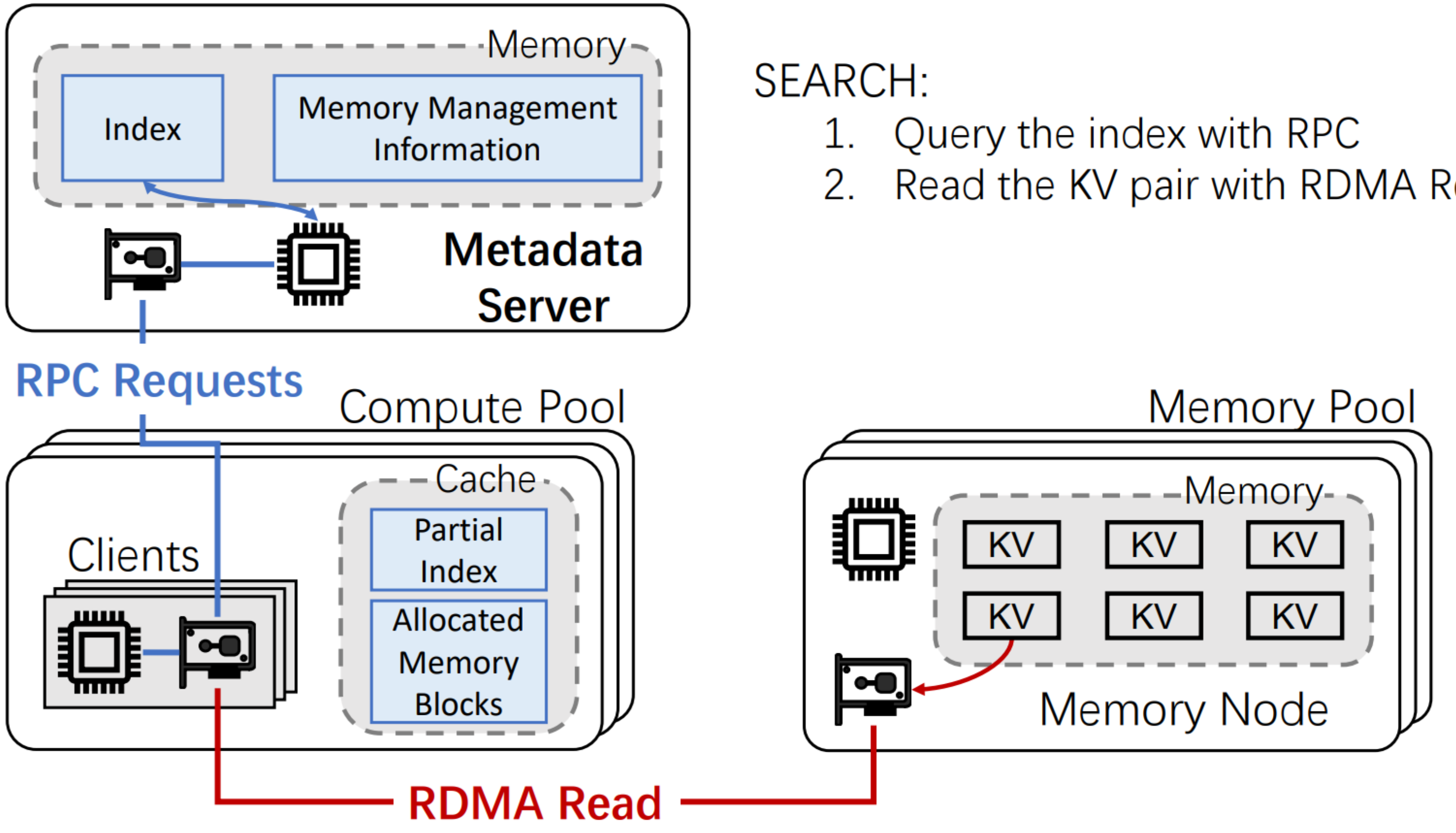
- **Disaggregated Memory**



- **Compute Pool: Strong computing power and weak storage capacity**
- **Memory Pool: Weak computing power and Strong storage capacity**

Background

- **Existing Semi-Disaggregated Key-Value Stores**



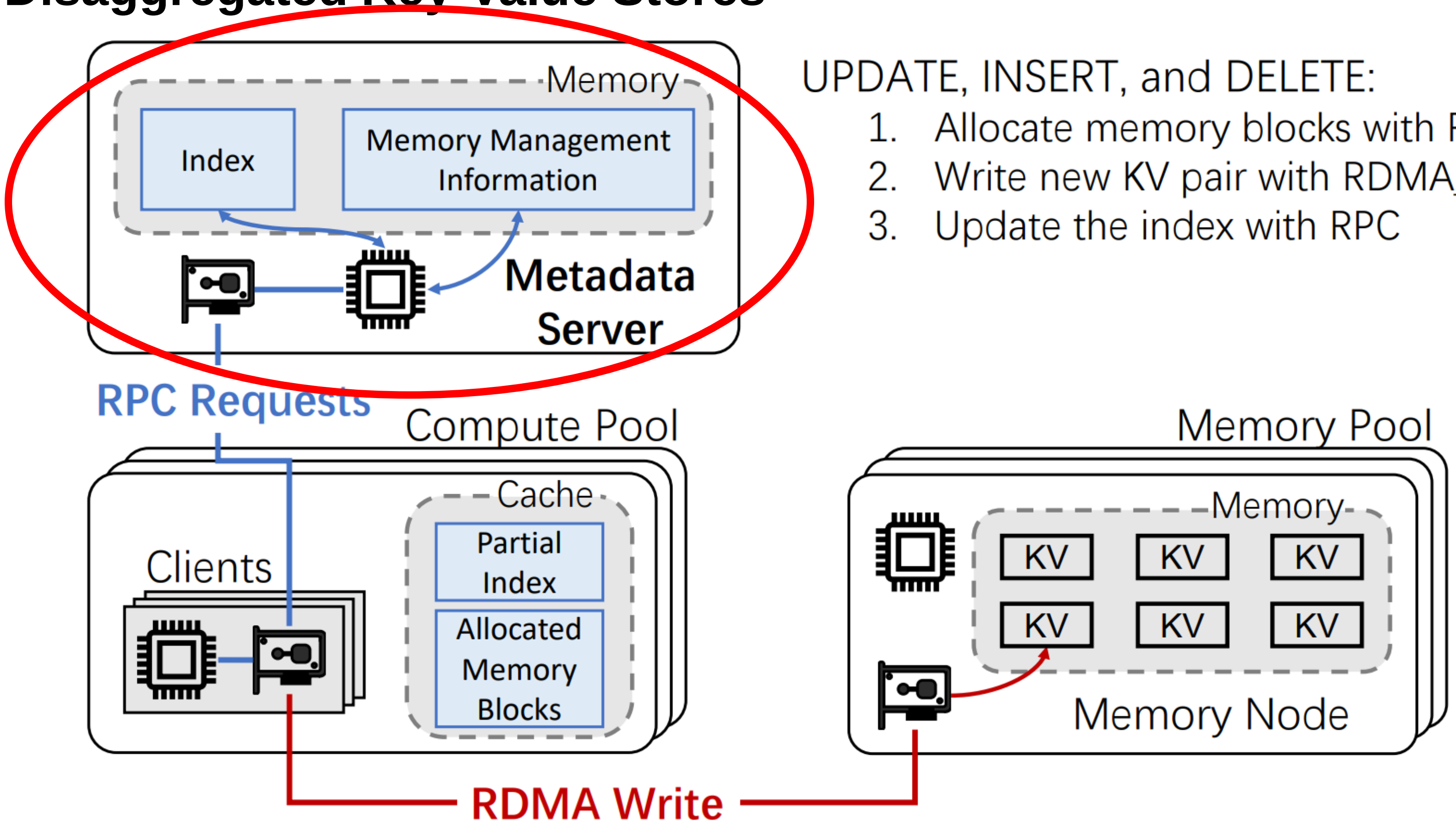
SEARCH:

1. Query the index with RPC
2. Read the KV pair with RDMA Read

- **MetadataServer: Used for managing metadata (address indexing)**

Background

➢ **Existing Semi-Disaggregated Key-Value Stores**



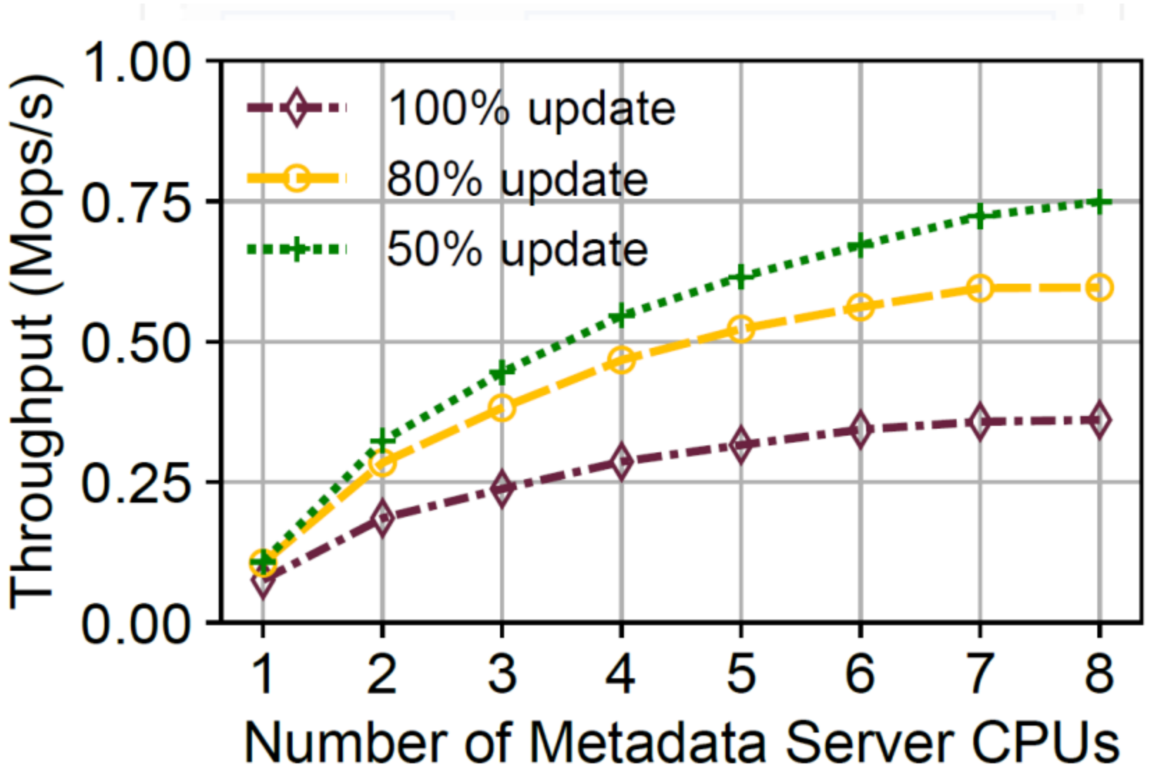
UPDATE, INSERT, and DELETE:

1. Allocate memory blocks with RPC
2. Write new KV pair with RDMA_WRITE
3. Update the index with RPC

➢ **MetadataServer: Used for managing metadata (address indexing)**

Problem

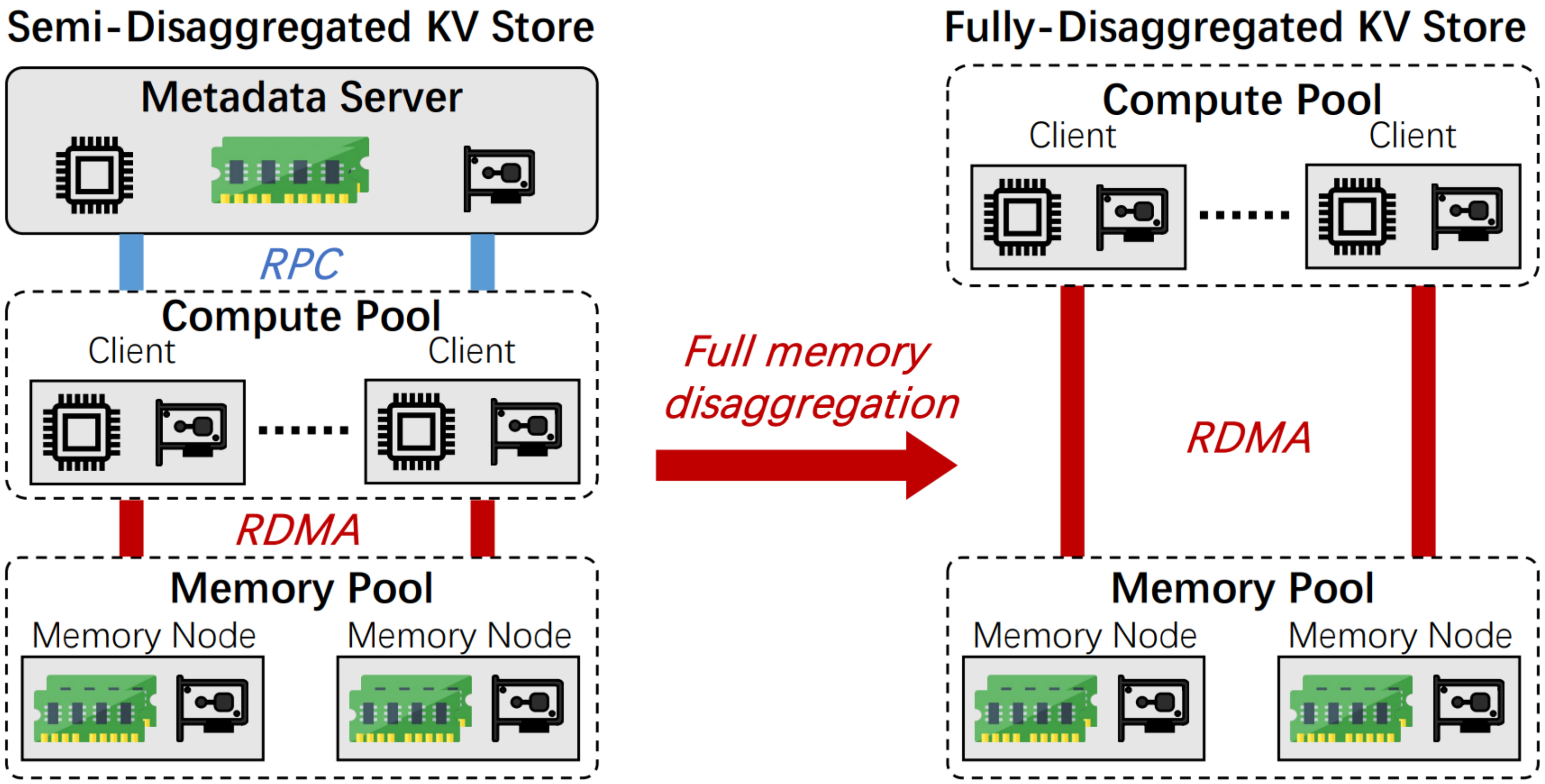
➢ **The problem: The metadata management still relies on monolithic servers, cannot fully exploit the resource efficiency and elasticity benefits of DM**



Additional resource required:
At least 6 CPU cores and 1 RNIC

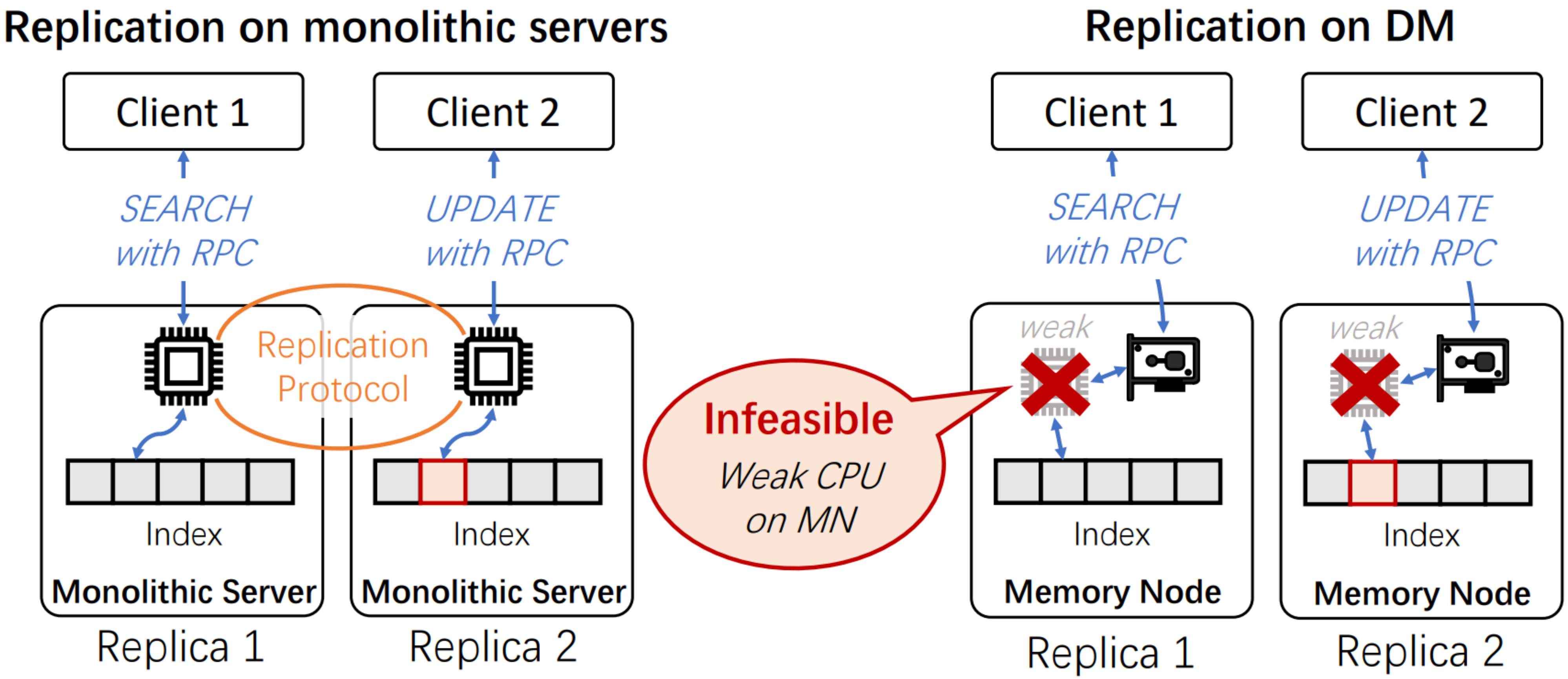
Motivation&Key Idea

- The motivation: Can the metadata server be removed to achieve complete decoupling?
- The key-idea: Is it possible to manage metadata directly with clients?



Challenge 1: Client-Centric Index Replication

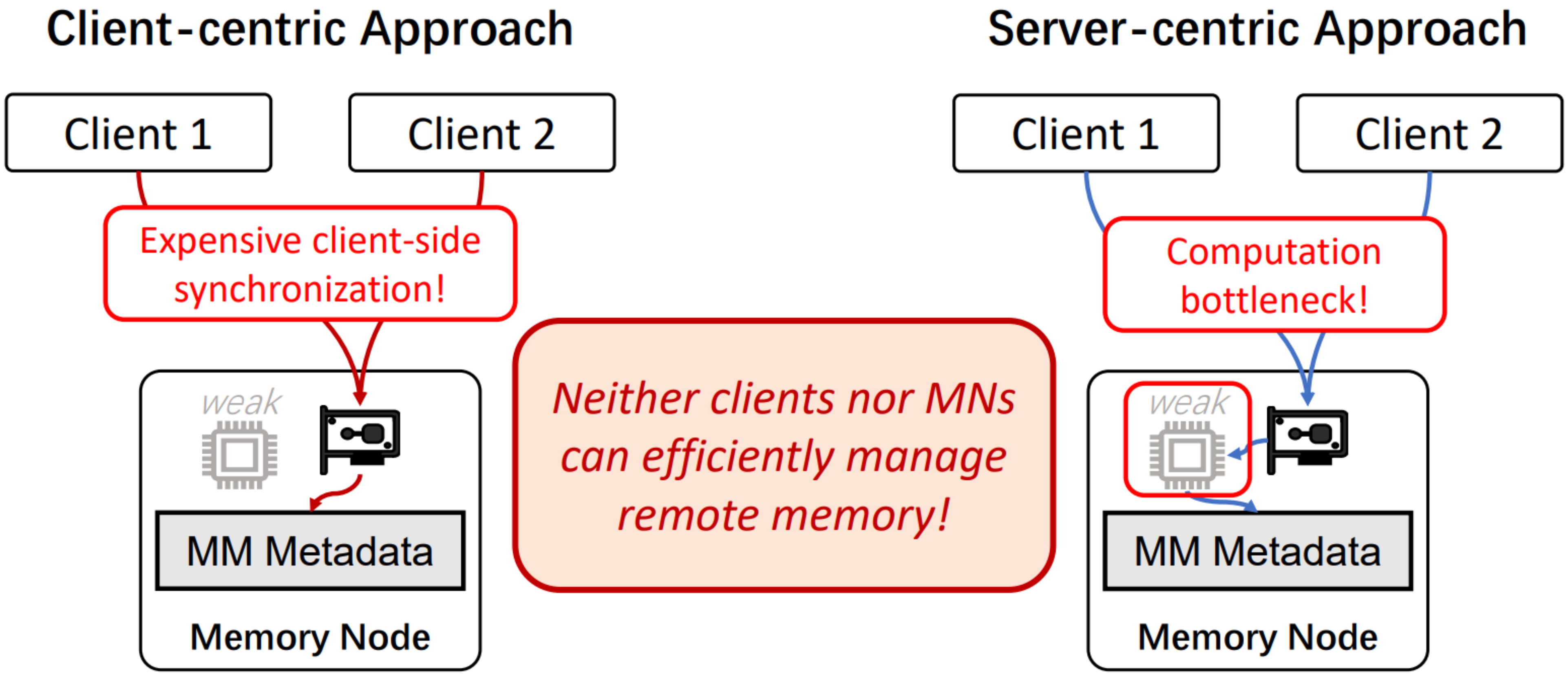
- Replication on monolithic servers: Replication protocols
- Replication on DM: Weak compute power



- Weak MN-side compute power cannot execute replication protocols

Challenge 2: Remote Memory Allocation

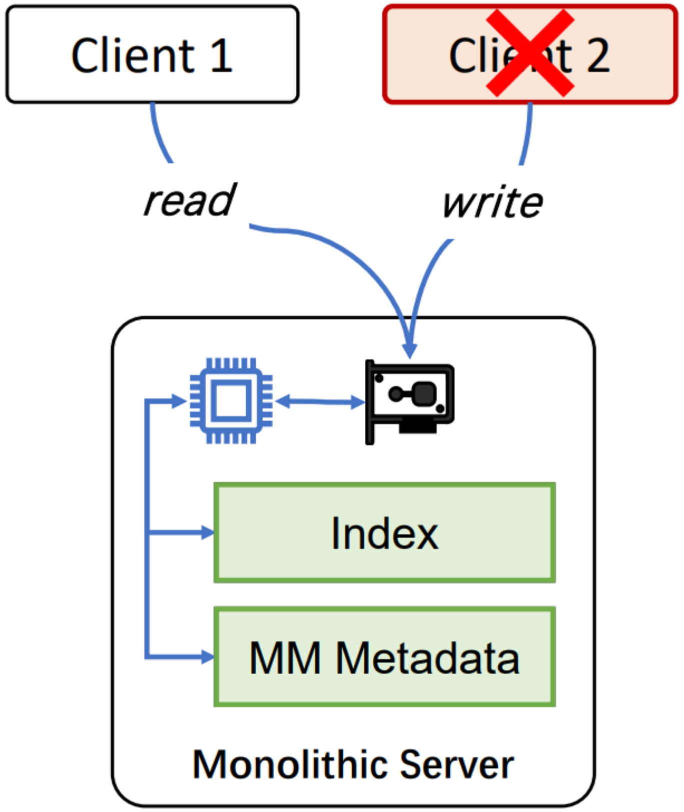
- ➢ **Client-centric Approach: compute on Client**
- ➢ **Server-centric Approach: compute on Server**



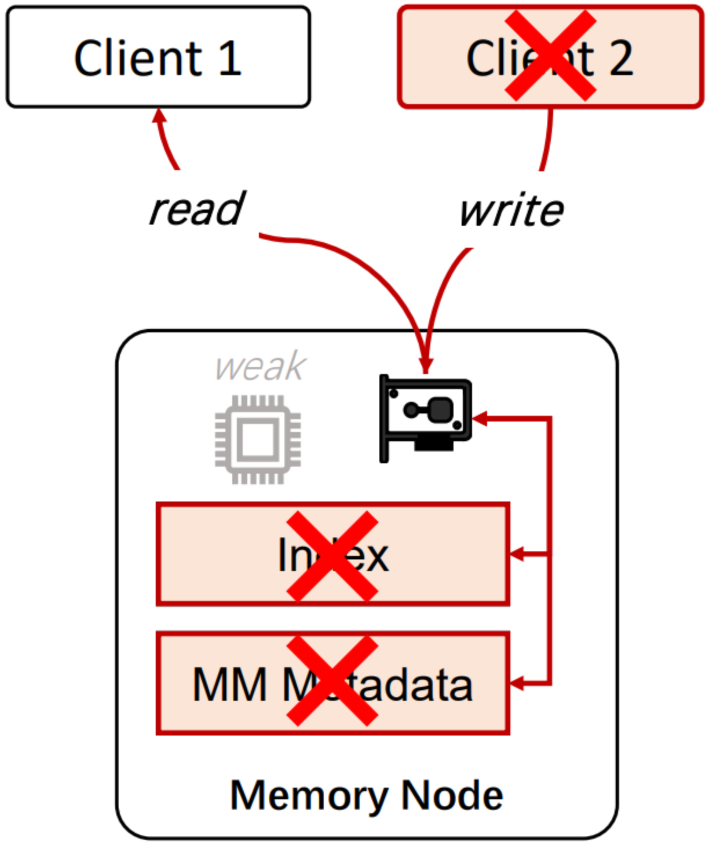
- ➢ **Weak MN-side compute power cannot efficiently allocate memory spaces**

Challenge 3: Metadata Corruption

- ➢ **monolithic-server-based KV stores:**
  - **CPUs of the KV store server exclusively access and modify metadata**

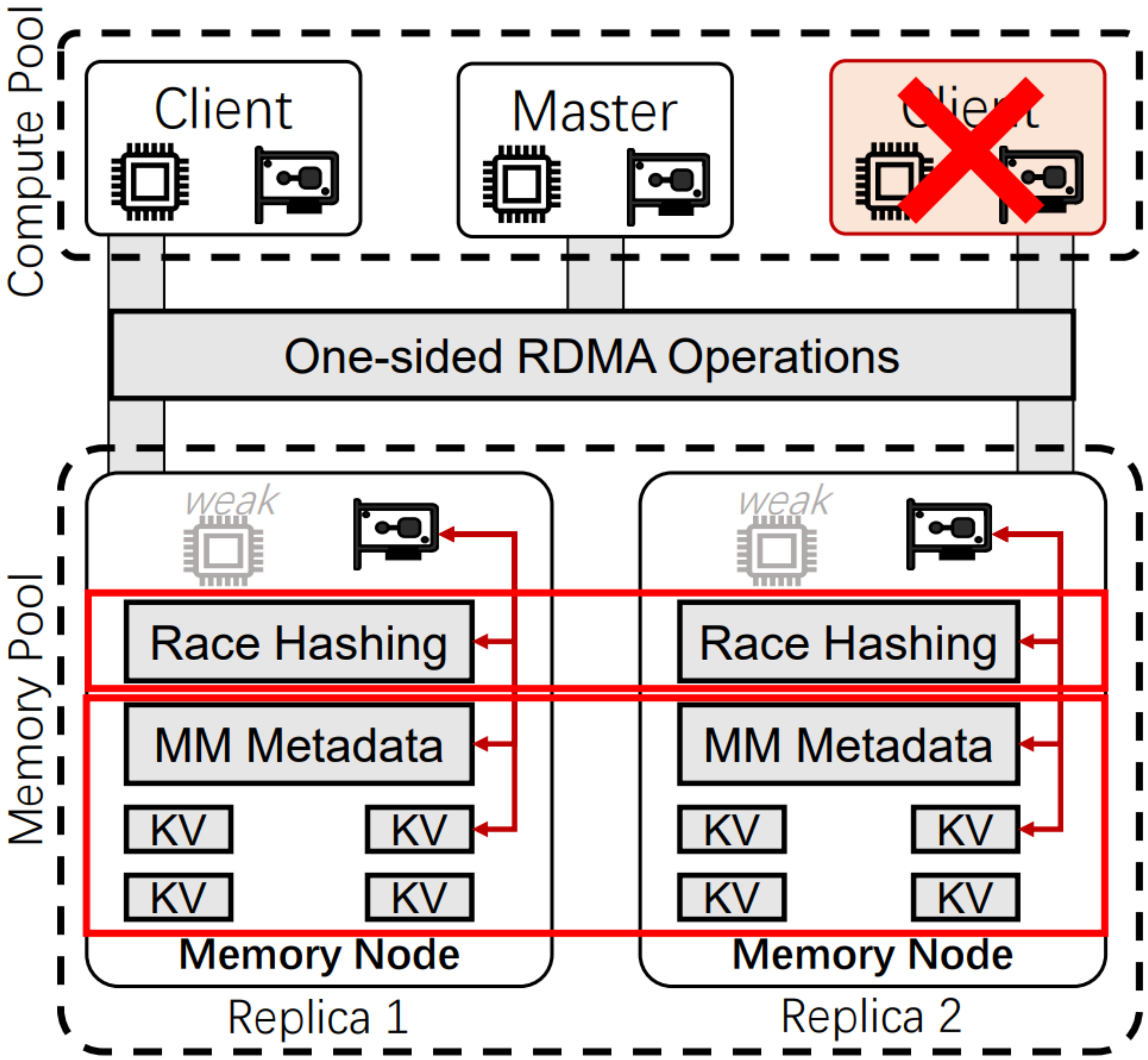


- ➢ **fully memory-disaggregated KV stores:**
  - **Clients directly access and modify metadata with one-sided RDMA verbs**



➢ Client-failures compromises the correctness of the entire KV store

FUSEE Overview

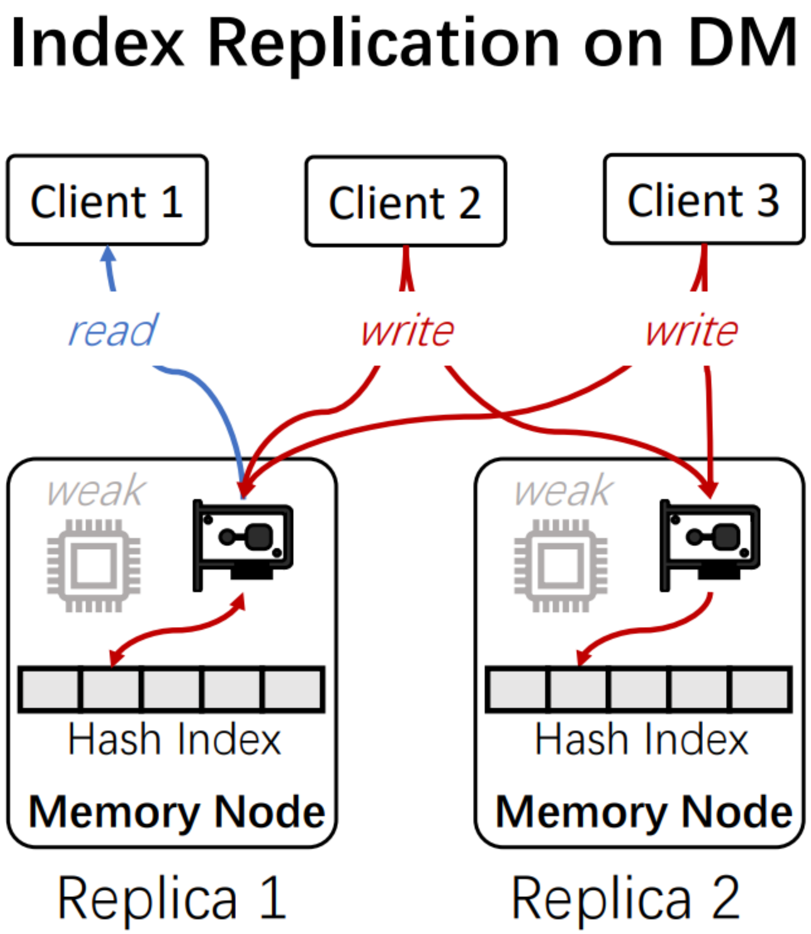


- **Metadata corruption:**
  Embedded operation log

- **Client-centric index replication:**
  The SNAPSHOT protocol

- **Remote memory allocation:**
  Two-level memory allocation

Design —Client-Centric Index Replication

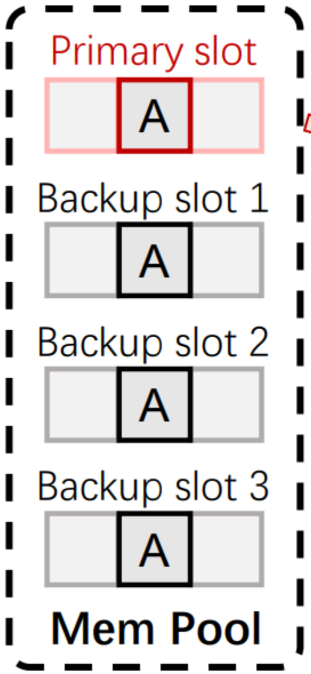


Main problem

- **How to protect readers from reading incomplete writes?**
- **How to efficiently resolve write-write conflicts among clients?**

Design —Client-Centric Index Replication

- **How to protect readers from reading incomplete values?**
  - Separate index replicas into primary and backups
  - Resolve write-write conflicts on backup replicas

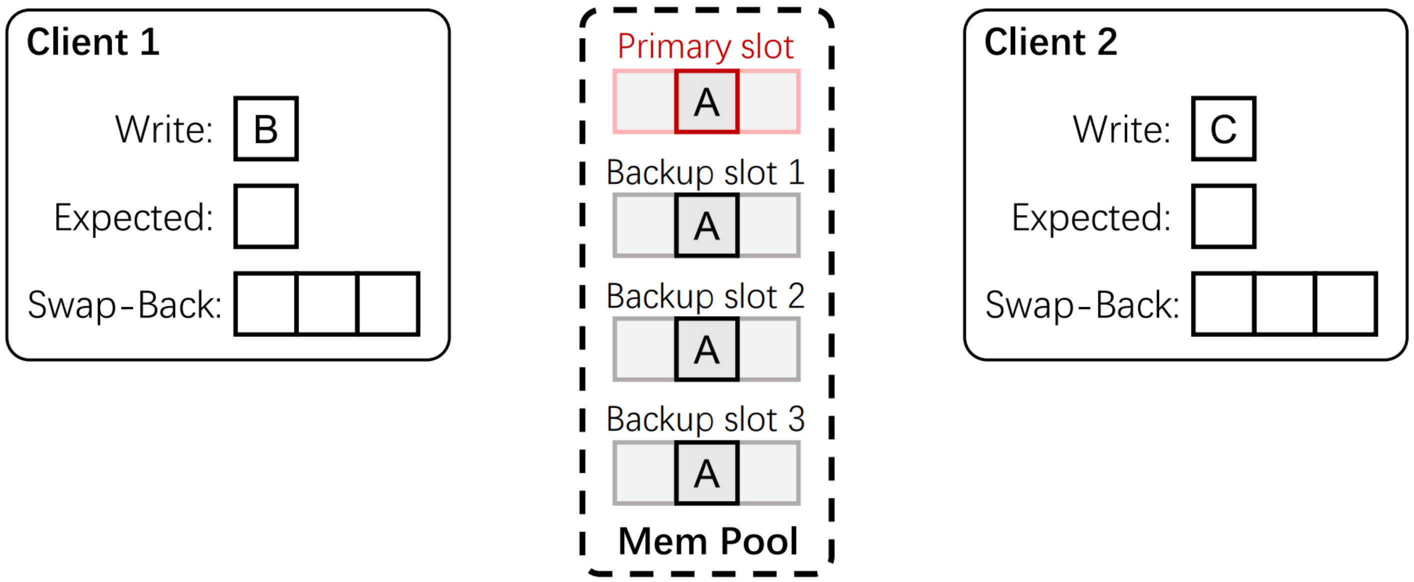


Always contain the correct value!

Directly read with RDMA_READ.

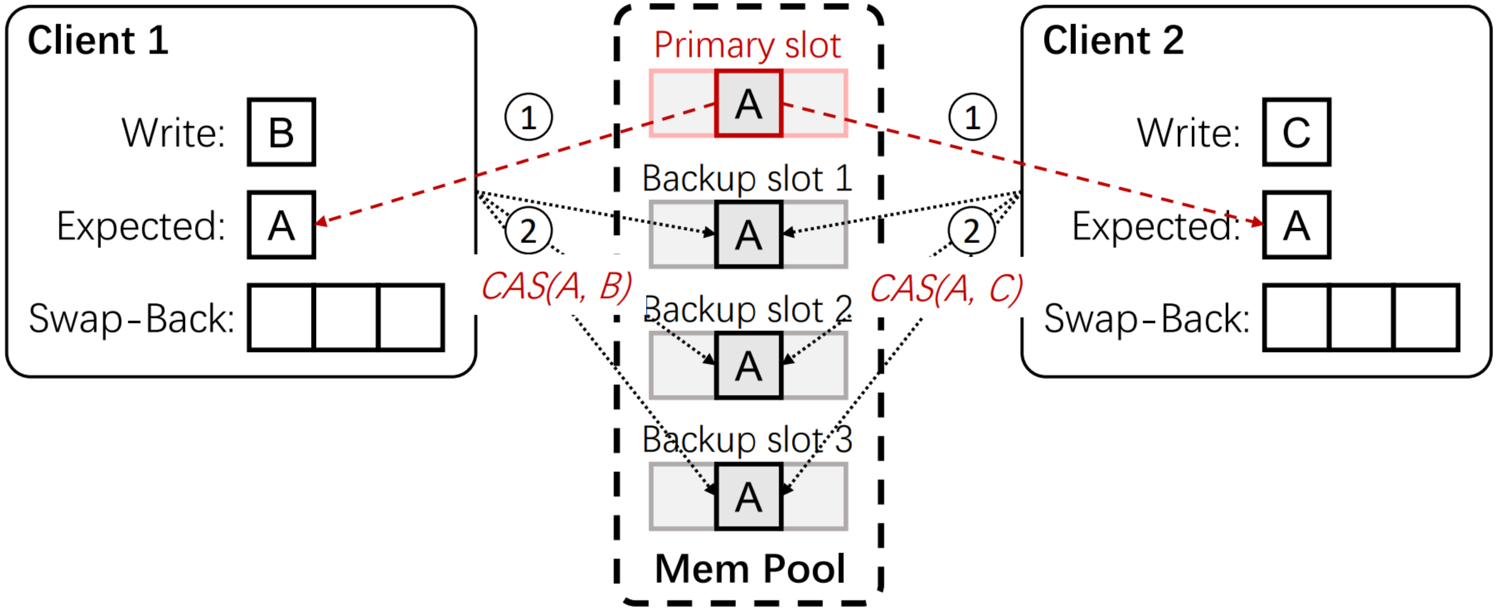
Design —Client-Centric Index Replication

- **How to efficiently resolve write-write conflicts on clients?**
  - Out-of-place KV modification -> Conflict clients write different values
  - Last-writer-wins conflict resolution -> Simple rules on client sides



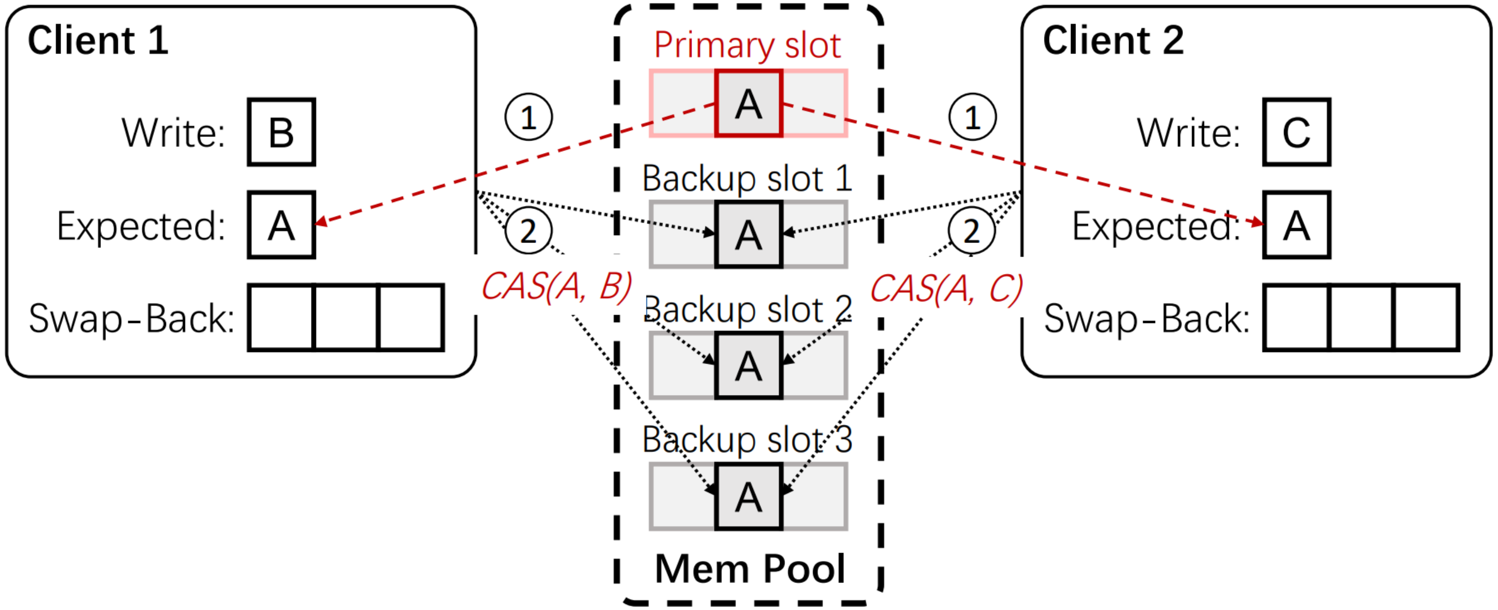
Design —Client-Centric Index Replication

- **How to efficiently resolve write-write conflicts on clients?**
  - Out-of-place KV modification -> Conflict clients write different values
  - Last-writer-wins conflict resolution -> Simple rules on client sides



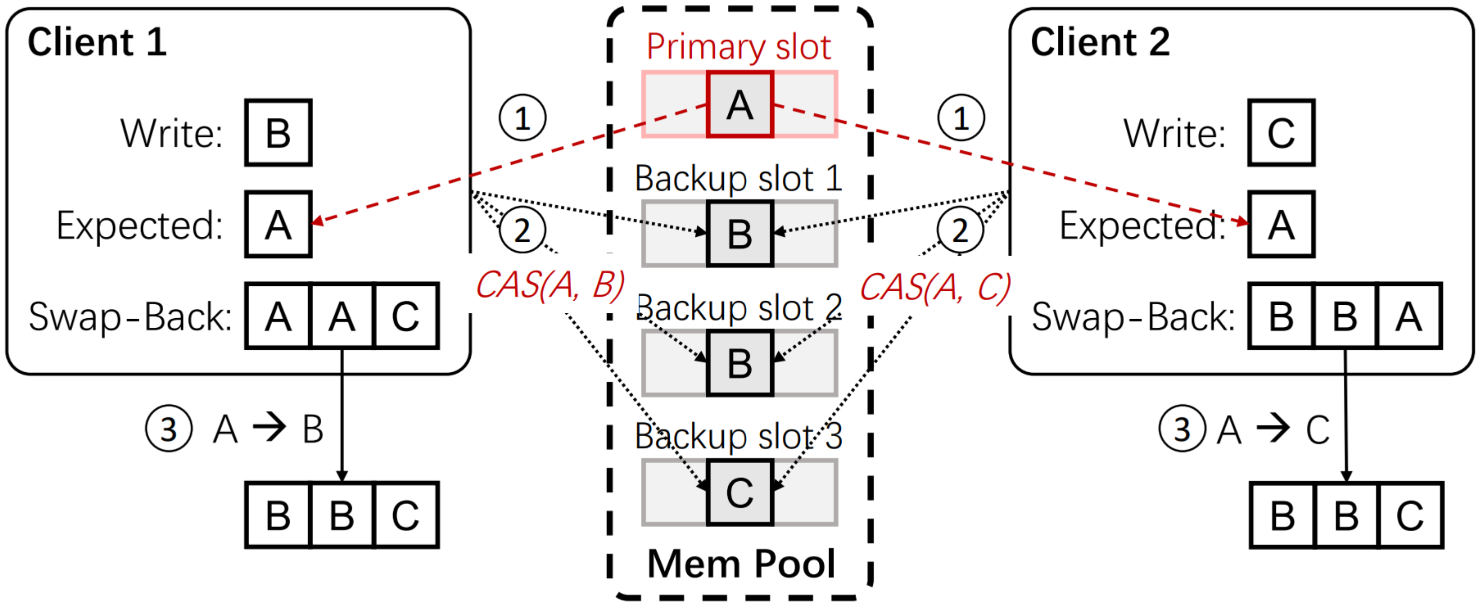
Design —Client-Centric Index Replication

- **How to efficiently resolve write-write conflicts on clients?**
  - Out-of-place KV modification -> Conflict clients write different values
  - Last-writer-wins conflict resolution -> Simple rules on client sides



Design —Client-Centric Index Replication

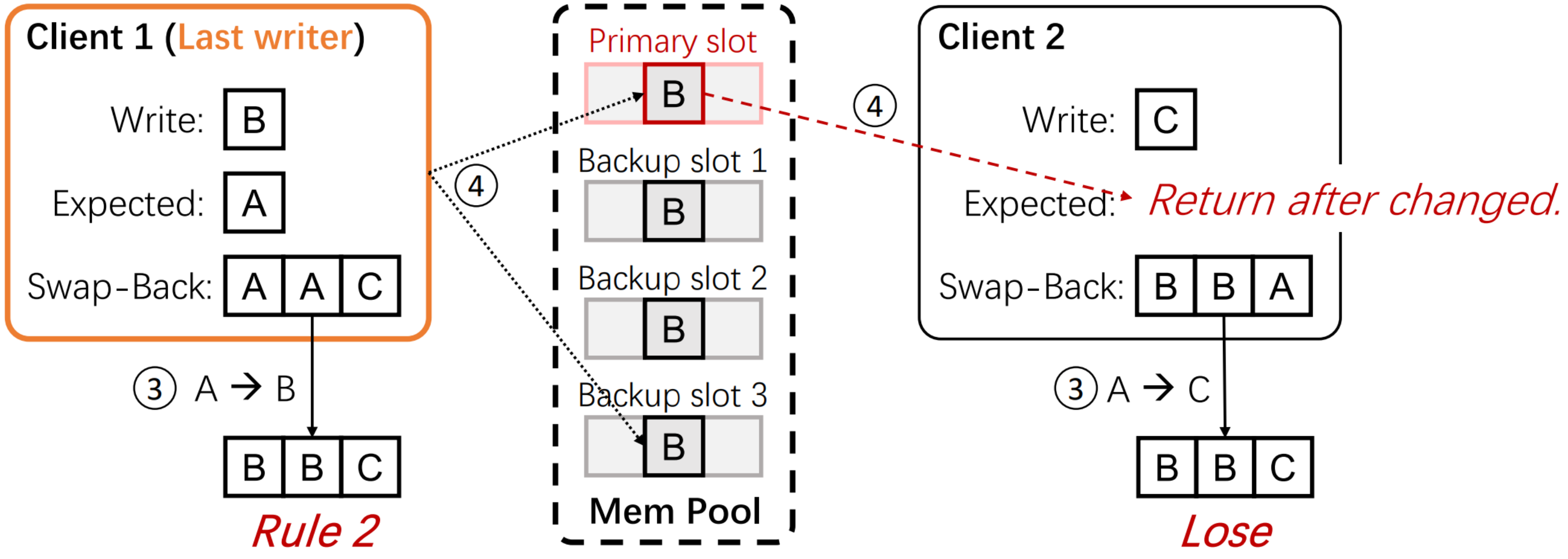
- **How to efficiently resolve write-write conflicts on clients?**
  - Out-of-place KV modification -> Conflict clients write different values
  - Last-writer-wins conflict resolution -> Simple rules on client sides



Design —Client-Centric Index Replication

➢ **A client is a last writer if:**

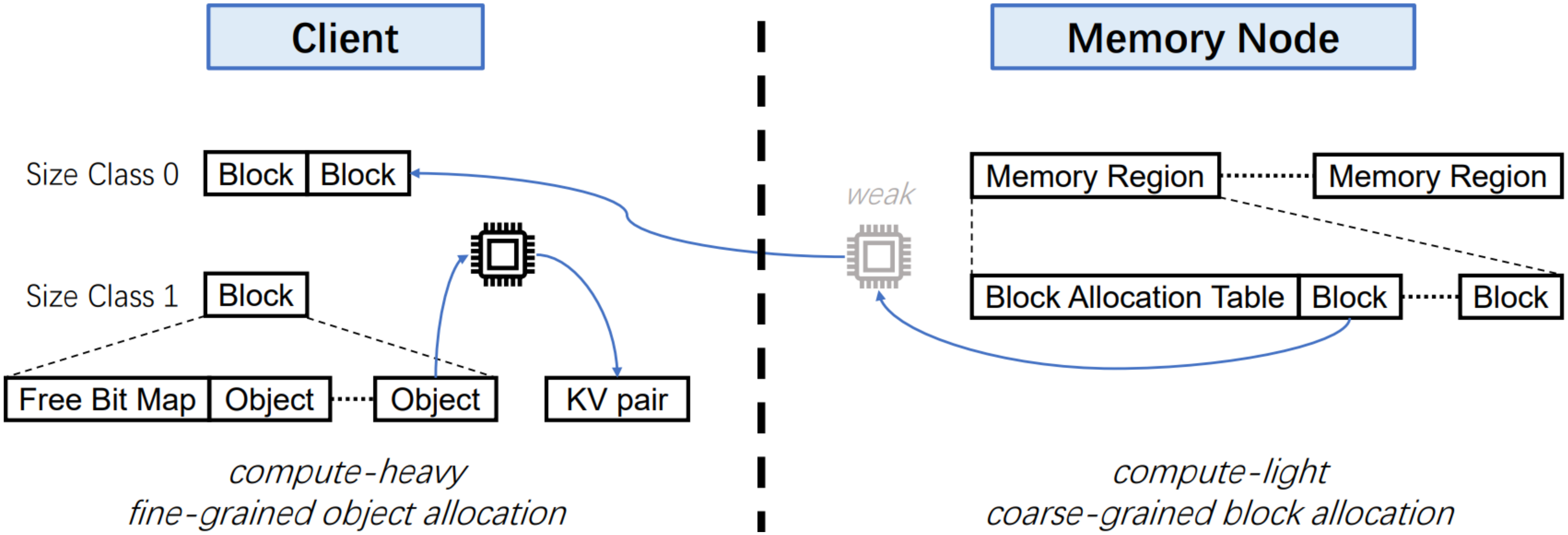
- Successfully modified all backup slots
- Or, modified a majority of backup slots
- Or, writes the smallest value when there is no majority



Design —Two-Level Memory Management

➢ **Challenge: Where to execute memory management logic?**

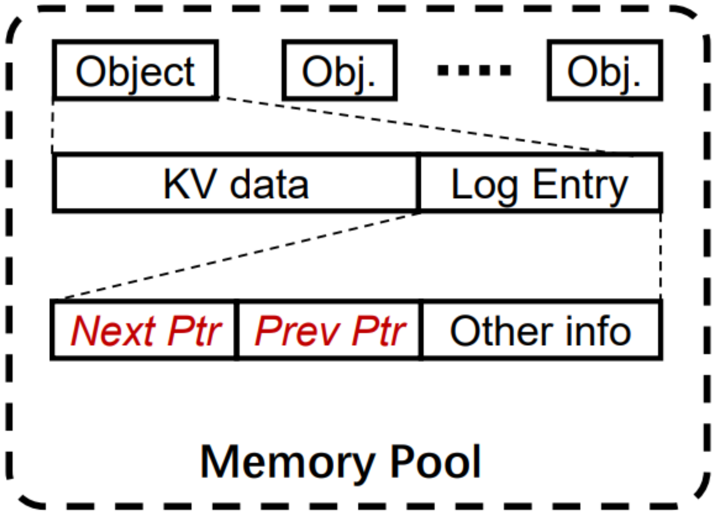
Key idea: Separate into compute-light and compute-heavy tasks



Design —Embedded Operation Log

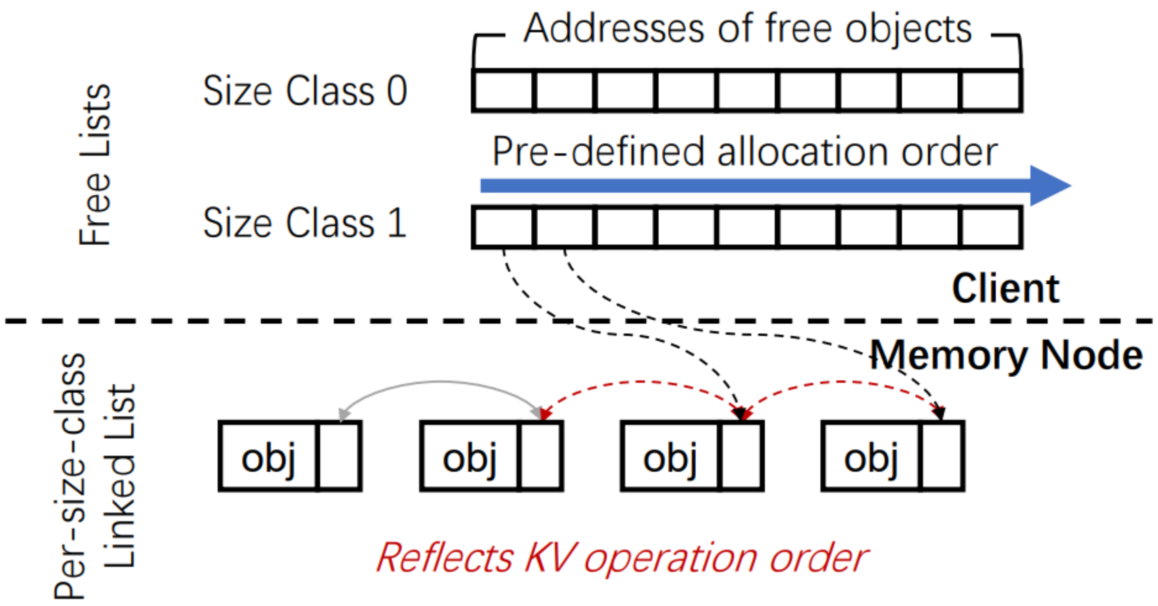
- **Challenge: An additional RTT to create a log entry before operation starts**

  Key idea: Embed operation logs into KV pairs to reduce the additional RTT



How to construct operation sequence?

Use memory allocation order!



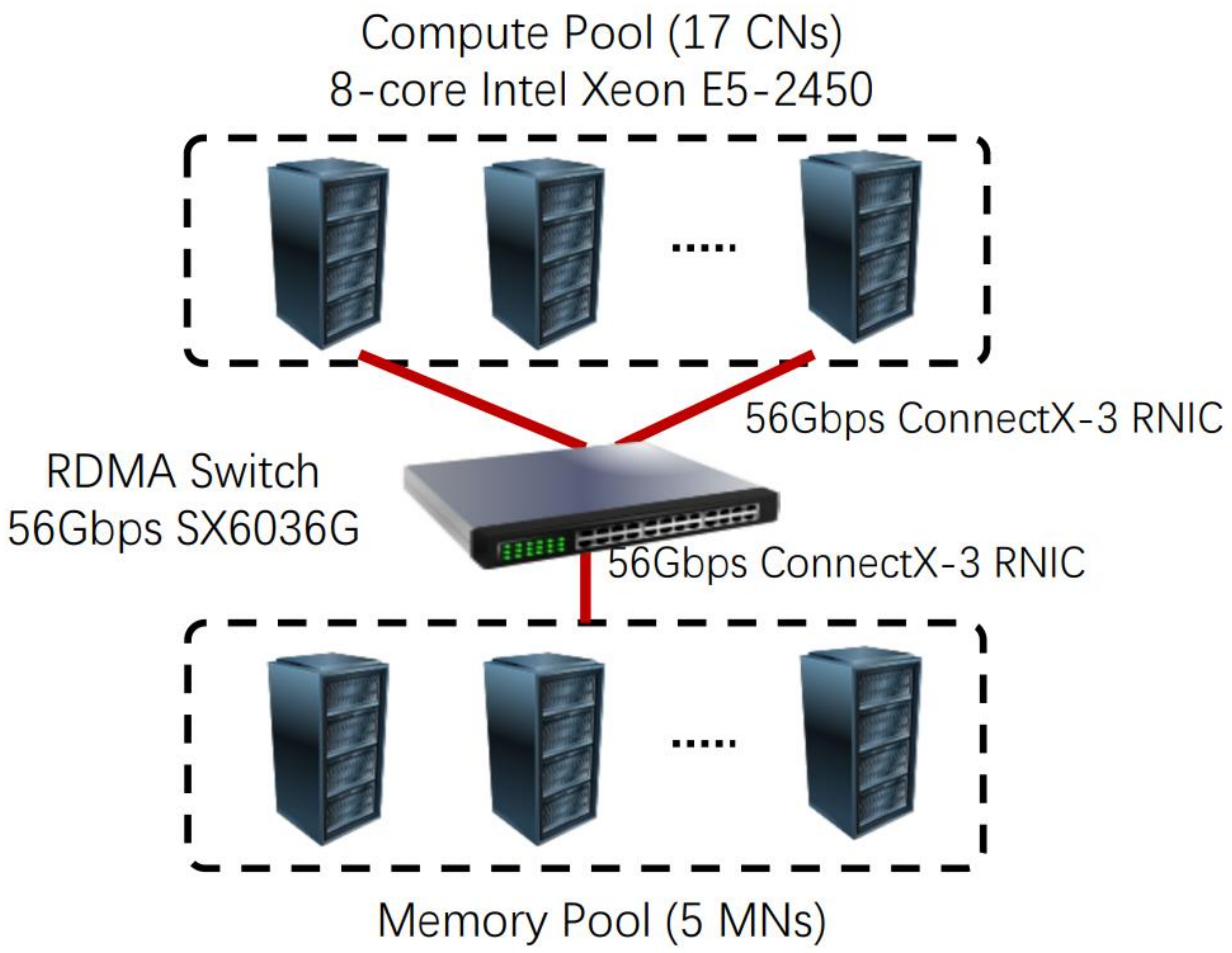
Evaluation

➢ **Benchmarks:**

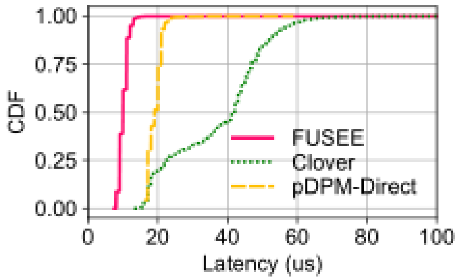
- YCSB hybrid benchmarks
- Microbenchmark

➢ **Comparisons:**

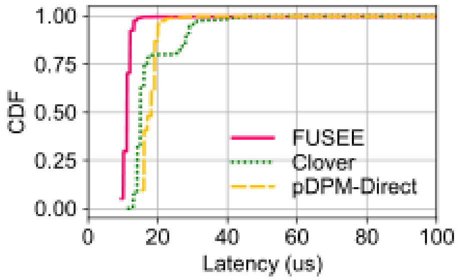
- Clover [ATC 20]
- pDPM-Direct [ATC 20]
- FUSEE-NC (no cache)



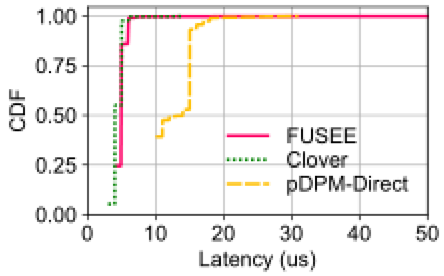
Evaluation —Latency&Throughput



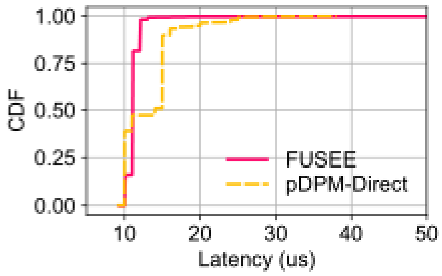
(a) INSERT latency CDF.



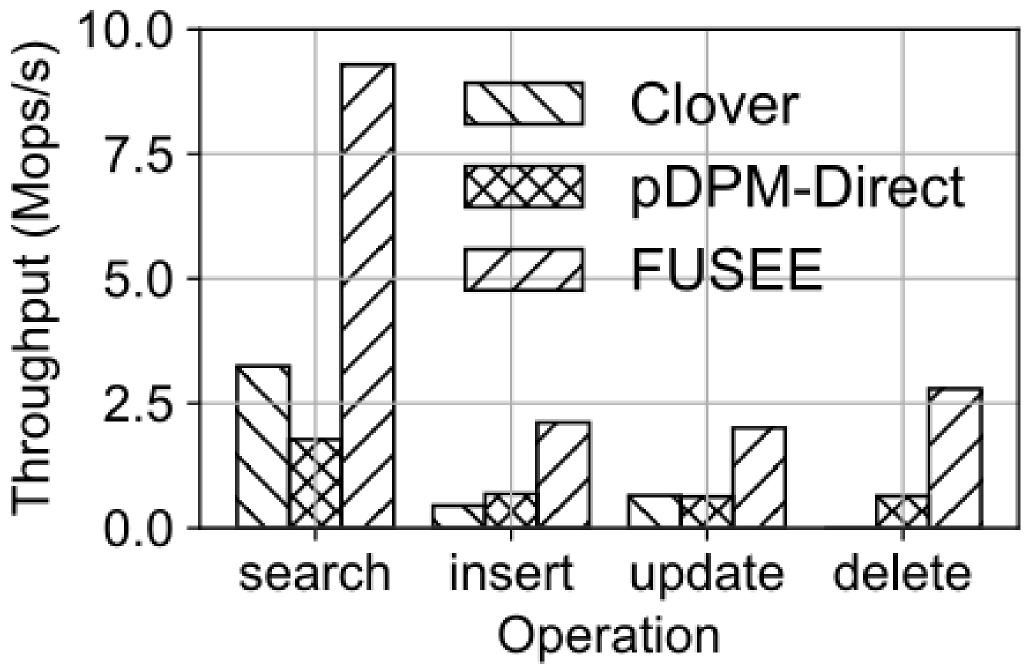
(b) UPDATE latency CDF.



(c) SEARCH latency CDF.

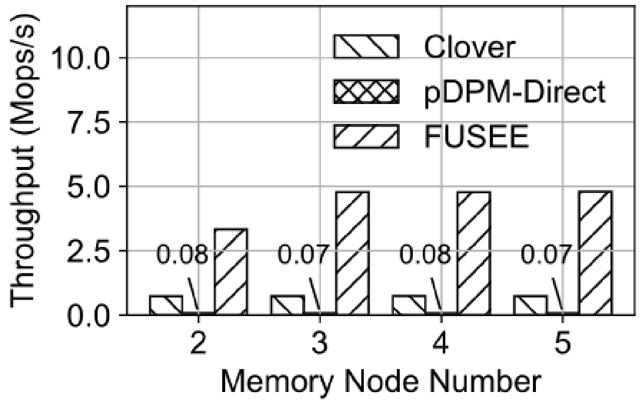


(d) DELETE latency CDF.

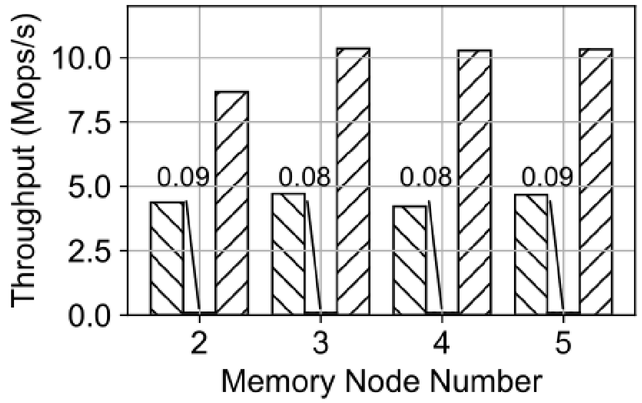


- **FUSEE performs the best on INSERT and UPDATE, since the SNAPSHOT replication protocol has bounded RTTs.**
- **FUSEE has a little higher SEARCH latency than Clover since FUSEE reads the hash index and the KV pair in a single RTT, which is slower than only reading the KV pair in Clover.**
- **FUSEE has the Highest Throughput.**

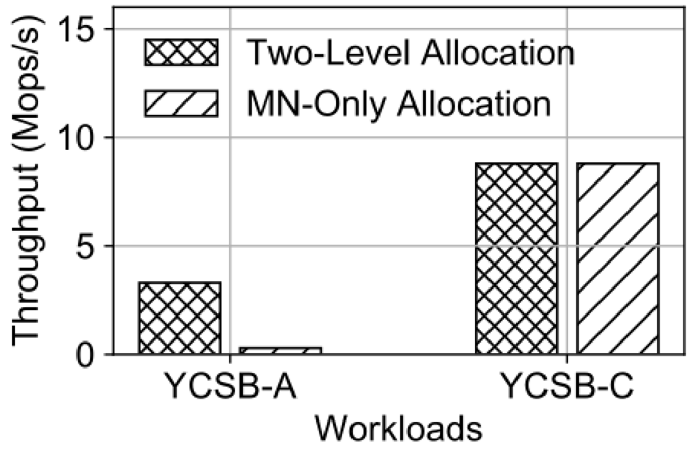
Evaluation —YCSB-Throughput & Two-level memory allocation performance.



(a) YCSB-A throughput.



(b) YCSB-C throughput.



- **As for FUSEE, the throughput improves as the number of memory nodes increases from 2 to 3. There is no further throughput improvement because the total throughput is limited by the number of compute nodes.**
- **The YCSB-A throughput drops 90.9% due to the limited compute power on MNs, while the YCSB-C throughput remains the same since no memory allocation is involved in the read-only setting.**

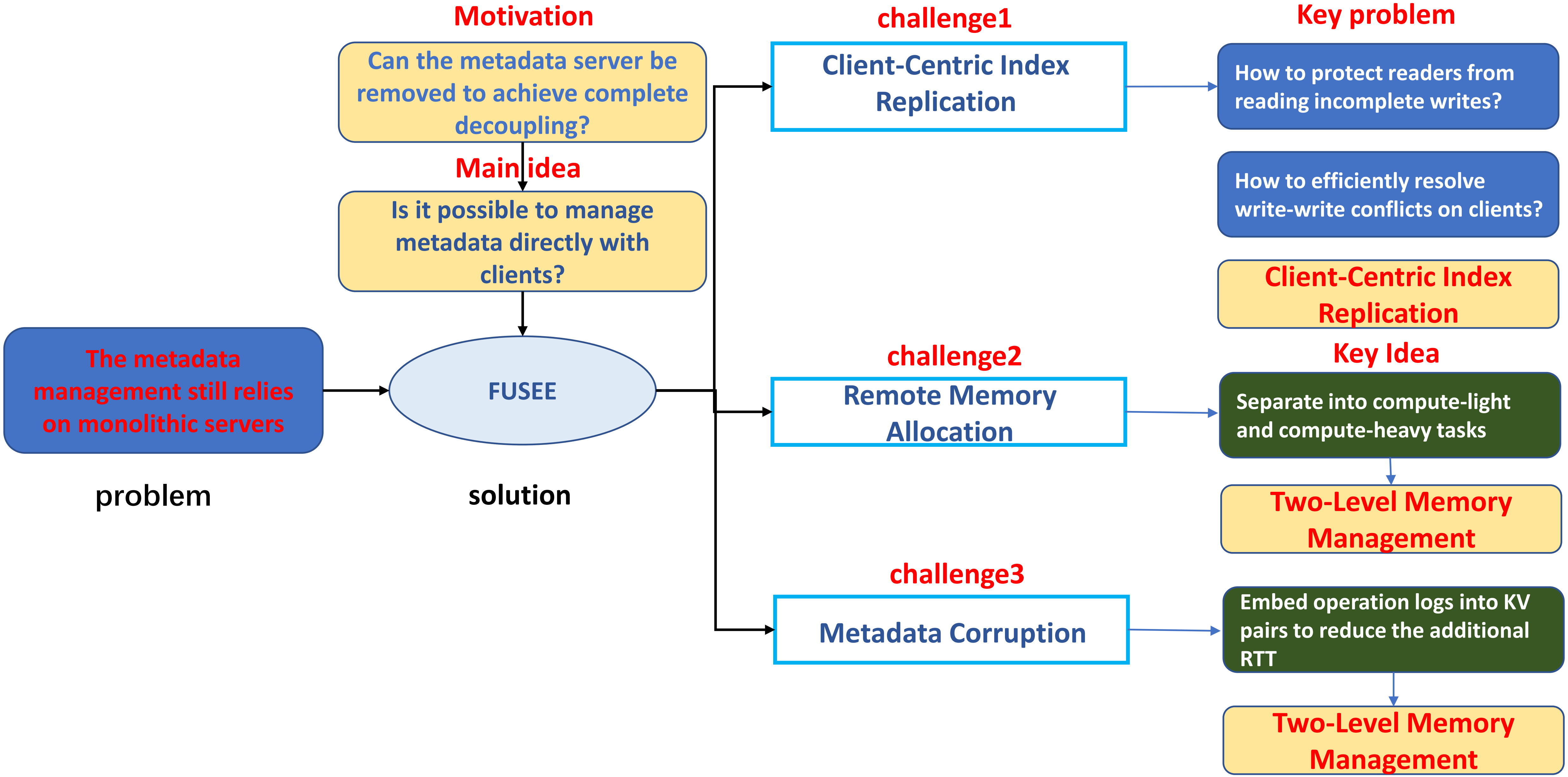
Evaluation —Recover from Crashed Clients.

Table 1: Client recovery time breakdown.

Step	Time (ms)	Percentage
Recover connection & MR	163.1	92.1%
Get Metadata	0.3	0.2%
Traverse Log	3.5	2.0%
Recover KV Requests	3.5	2.0%
Construct Free List	6.6	3.7%
Total	177.0	100%

- **The log traversal and KV request recovery only account for 4% of the recovery time, which implies the affordable overhead of log traversal.**

Conclusion



Motivation
Can the metadata server be removed to achieve complete decoupling?
Main idea
Is it possible to manage metadata directly with clients?
The metadata management still relies on monolithic servers
FUSEE
problem
solution
challenge1
Client-Centric Index Replication
challenge2
Remote Memory Allocation
challenge3
Metadata Corruption
Key problem
How to protect readers from reading incomplete writes?
How to efficiently resolve write-write conflicts on clients?
Client-Centric Index Replication
Key Idea
Separate into compute-light and compute-heavy tasks
Two-Level Memory Management
Embed operation logs into KV pairs to reduce the additional RTT
Two-Level Memory Management