Fast and Consistent Remote Direct Access to Non-volatile Memory

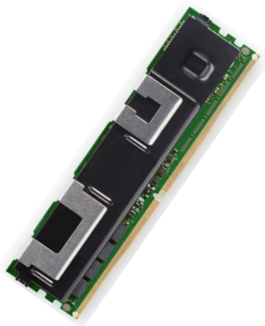
Jingwen Du, Fang Wang, Dan Feng, Weiguang Li, Fan Li
Huazhong University of Science and Technology, China
Corresponding author: wangfang@hust.edu.cn

Outline

- **Background and Motivation**
- **Our work: eFactory**
- **Performance Evaluation**
- **Conclusion**

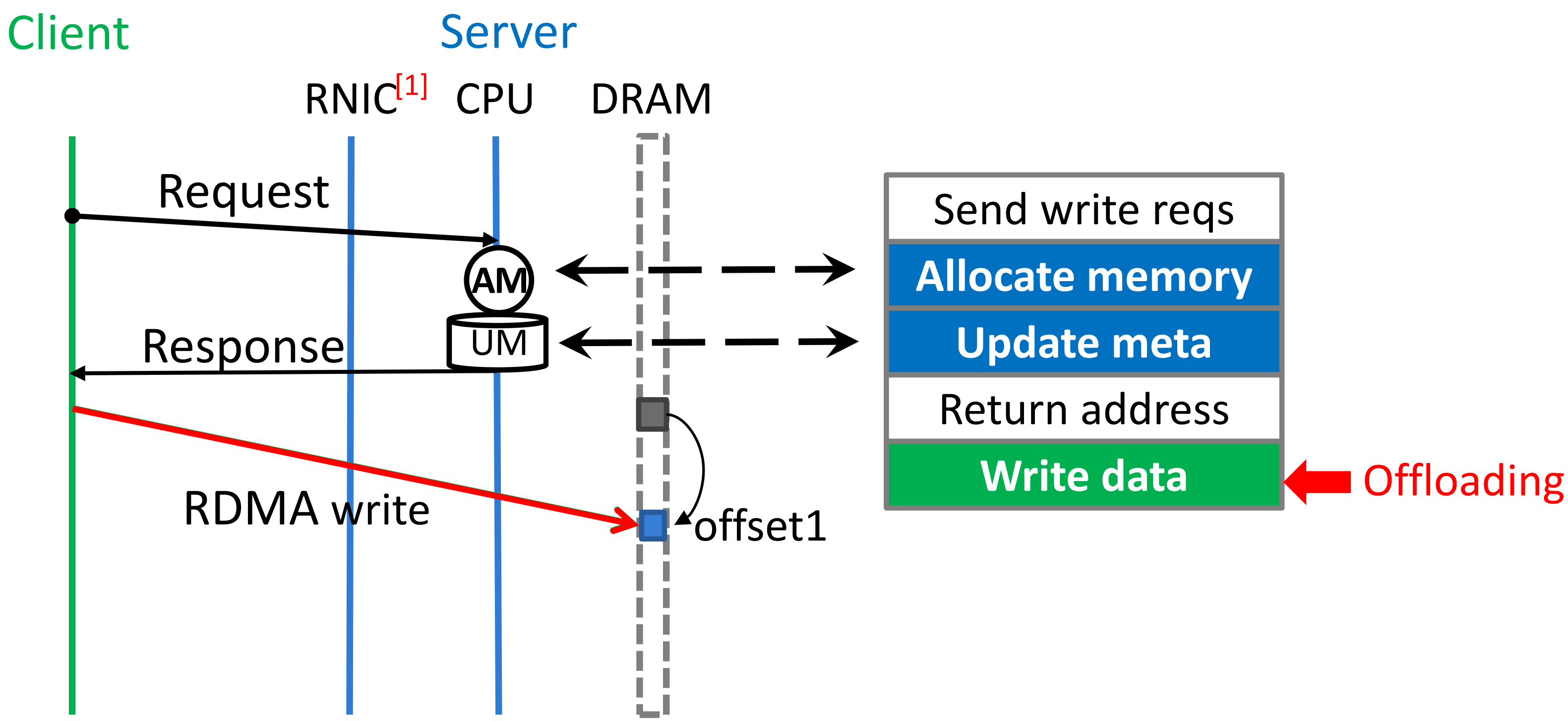
Fast Access to Remote Persistent Data

- **Modern data-intensive applications require fast access to massive amounts of persistent data**
- **Opportunities**
  - **Emerging hardware technologies: NVM & RDMA**

Non-volatile Main Memory (NVMM)	Remote Direct Memory Access (RDMA)
 • Non-volatility • Byte addressability • Large capacity • Low latency (100ns)	• Kernel-bypass network • high bandwidth (100Gbps) • Low latency ($2\mu s$) • One-sided: bypass remote CPUs • Two-sided: fast message passing

Client-active Scheme*

- **Data writing is offloaded to clients**
- **Process more requests and gain higher throughput**

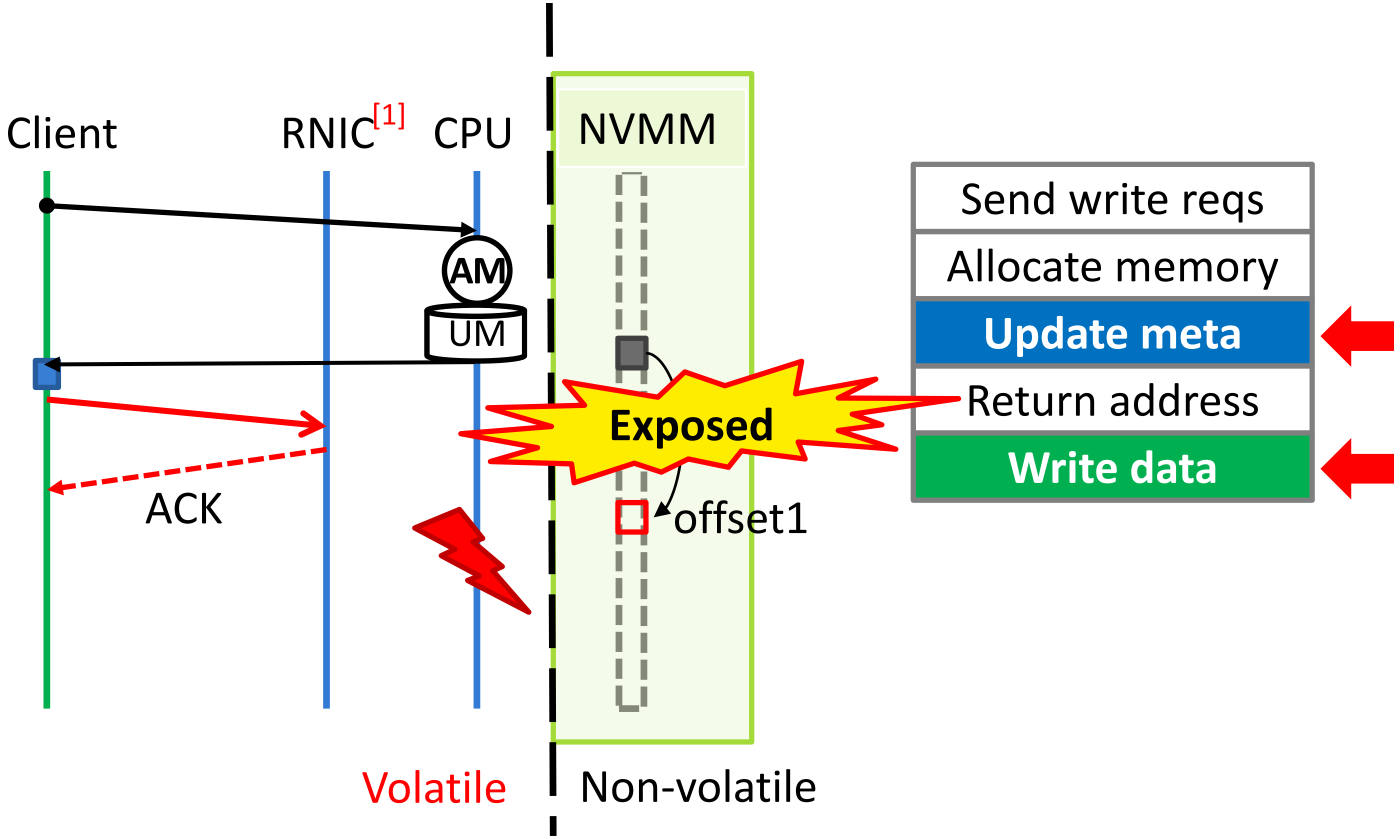


*proposed and used in previous work such as Octopus@ATC'17, Orion@FAST'19, etc.

[1] RNIC = RDMA-enabled Network Interface Card

Challenge: Remote Data Consistency

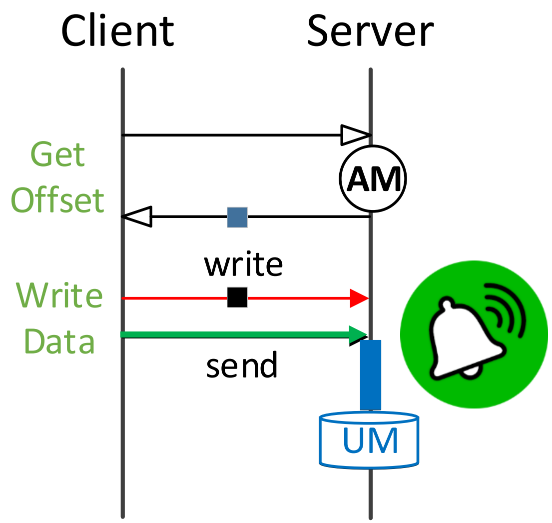
- **RDMA write doesn't have persistence semantics currently**
- **Metadata is updated before data writing**



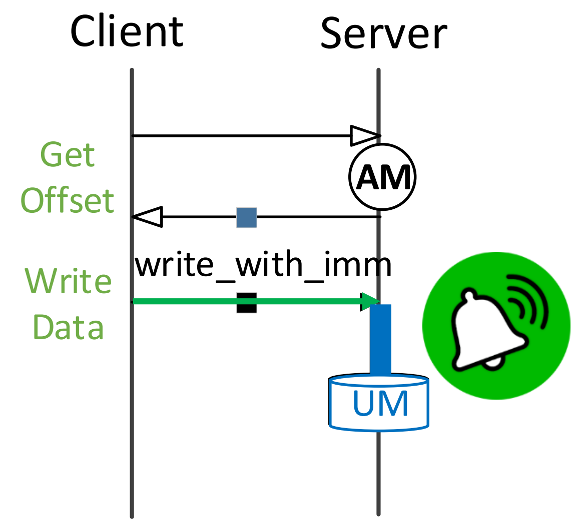
[1] RNIC = RDMA-enabled Network Interface Card

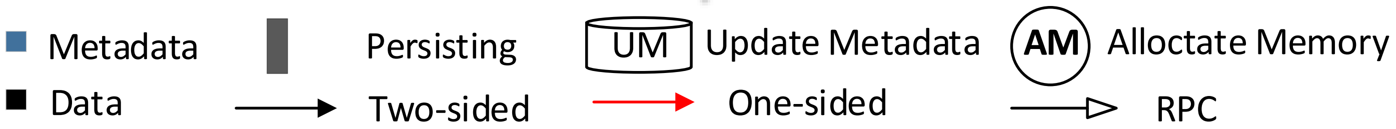
Current Solutions: SAW & IMM

- **Send after Write(SAW)[1]**
  - RDMA write followed by an extra RDMA send with DDIO enabled



- **IMM[2]**
  - replace RDMA write with write_with_imm primitive

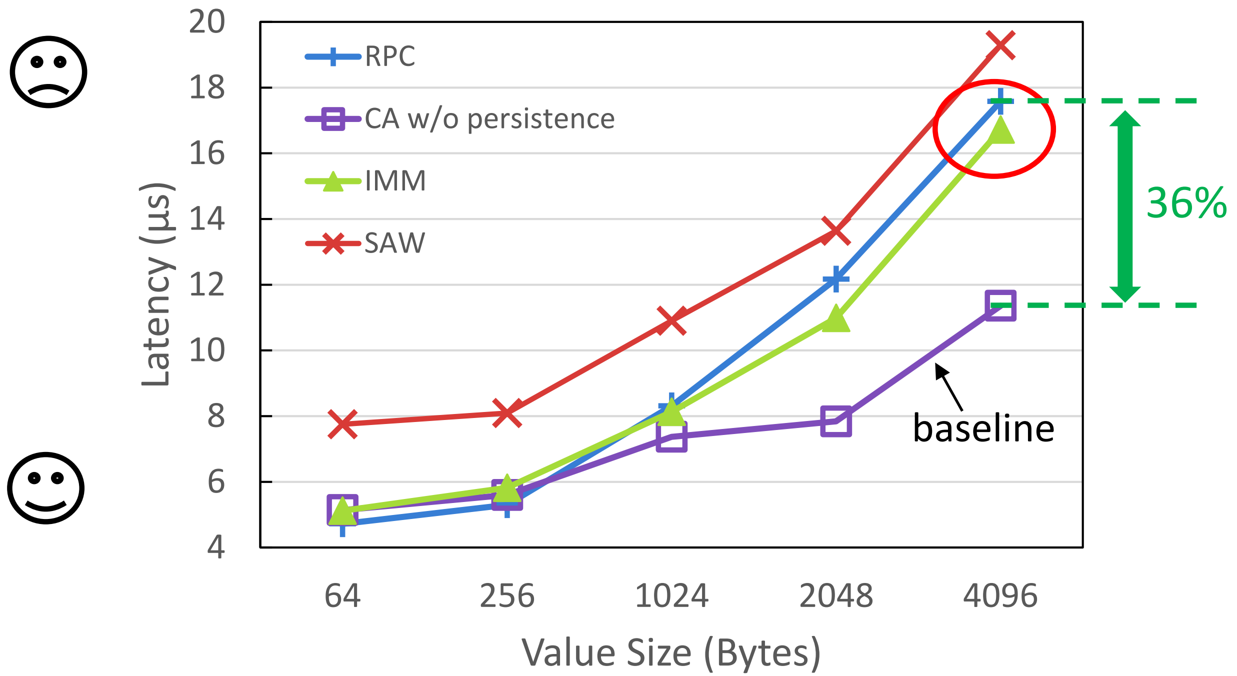




[1] Chet Douglas. RDMA with PMEM: Software mechanisms for enabling access to remote persistent memory. In *SDC* 2015.
[2] Yang J et al. Orion: A distributed file system for non-volatile main memory and RDMA-capable networks. FAST, 2019

Trade Write Performance for Consistency

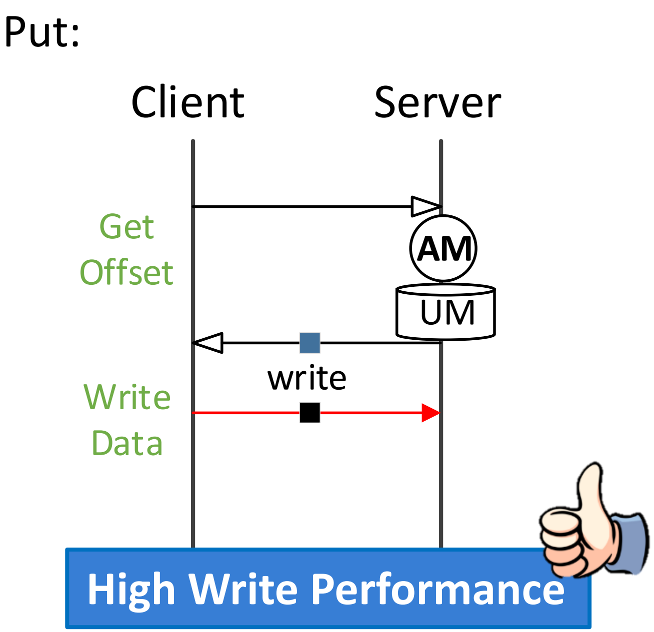
- **Client-active scheme w/o persistence (36% better than RPC)**
- **Writing durably with IMM & SAW loses latency advantage over RPCs (5%)**



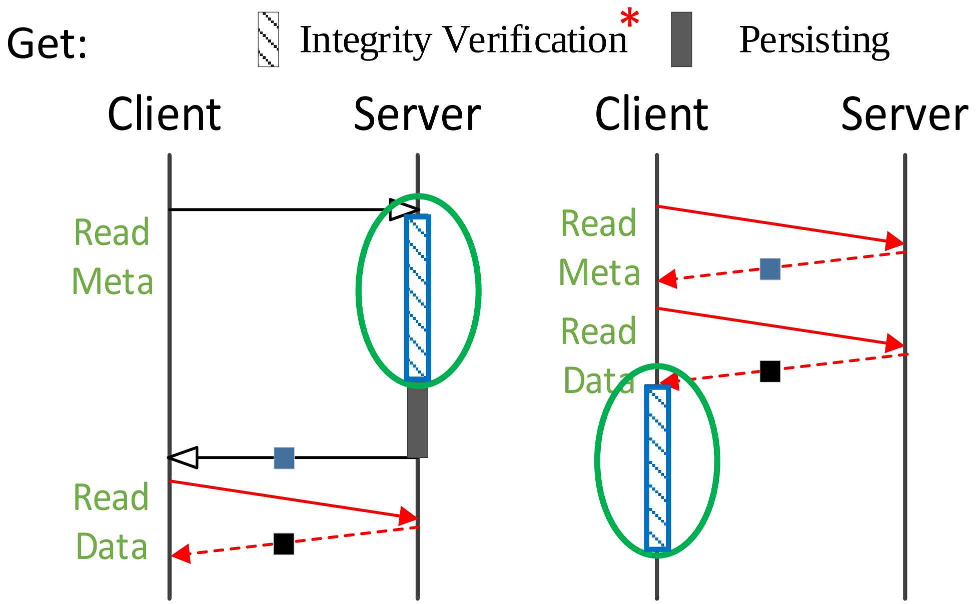
Latency of Writing to Remote NVMM

Current Solutions: Force & Erda

- **Use the client-active scheme w/o persisting immediately**
- **Verify data integrity with CRC* when reading**



Client-active w/o persistence



Forca[3]   **Erda**[4]

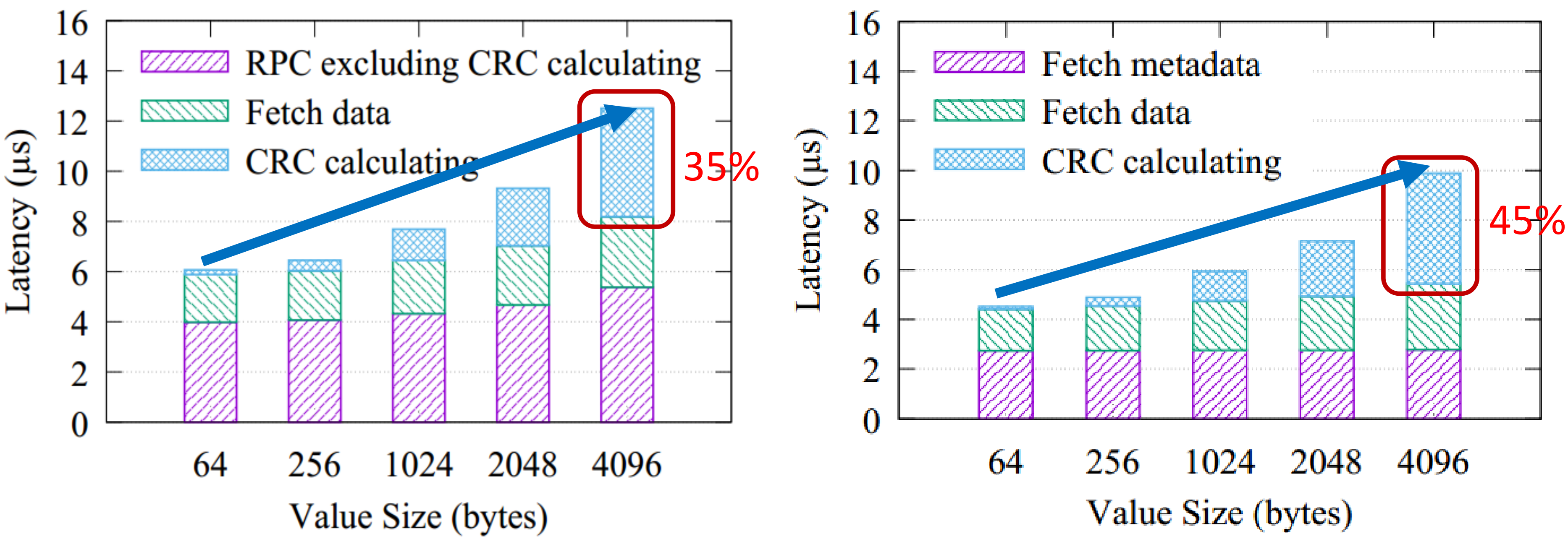
* Using CRC (Cyclic Redundancy Check)

[3] Huang H et al. Forca: Fast and atomic remote direct access to persistent memory, ICCD 2018

[4] X. Liu, Y. Hua, X. Li, and Q. Liu, "Write-optimized and consistent rdma-based nvm systems," arXiv preprint arXiv:1906.08173, 2019.

Trade Read Performance for Consistency

- **With value size increases, CRC overhead seriously degrades the read performance**



(a) Forca

(b) Erda

GET Latency Breakdown

Motivation

- **Existing solutions sacrifice either read or write performance in exchange for consistency guarantees**

(" X ": bad, "√": good, "--": moderate)

	Data Consistency	Read Performance	Write Performace
SAW[1]	√	√	X
IMM[2]	√	√	X
Forca[3]	√	X	√
Erda[4]	--	X	√
eFactory	√	√	√

[1] Chet Douglas. RDMA with PMEM: Software mechanisms for enabling access to remote persistent memory. In *Storage Developer Conference* 2015.
[2] Yang J et al. Orion: A distributed file system for non-volatile main memory and RDMA-capable networks. FAST, 2019
[3] Huang H et al. Forca: Fast and atomic remote direct access to persistent memory, ICCD 2018
[4] X. Liu, Y. Hua, X. Li, and Q. Liu, "Write-optimized and consistent rdma-based nvm systems," arXiv preprint arXiv:1906.08173, 2019.

Our Solution: eFactory

- **Multi-version Log-structuring Design**
- **Data Workflow**
  - **Client-active Write with Asynchronous Durability**
  - **Single Background Thread for Efficient Consistency**
  - **Hybrid Read Scheme**
- **Lock-free Log Cleaning Scheme** ➡ Please check it in our paper!

Multi-version Log-structuring Design

① **Out-of-place update with log structuring mechanism**
② **Multi-version linked list for each object**
③ **Embed durability flag in each object**
④ **2-offset region & backlink pointer for lock-free log cleaning**

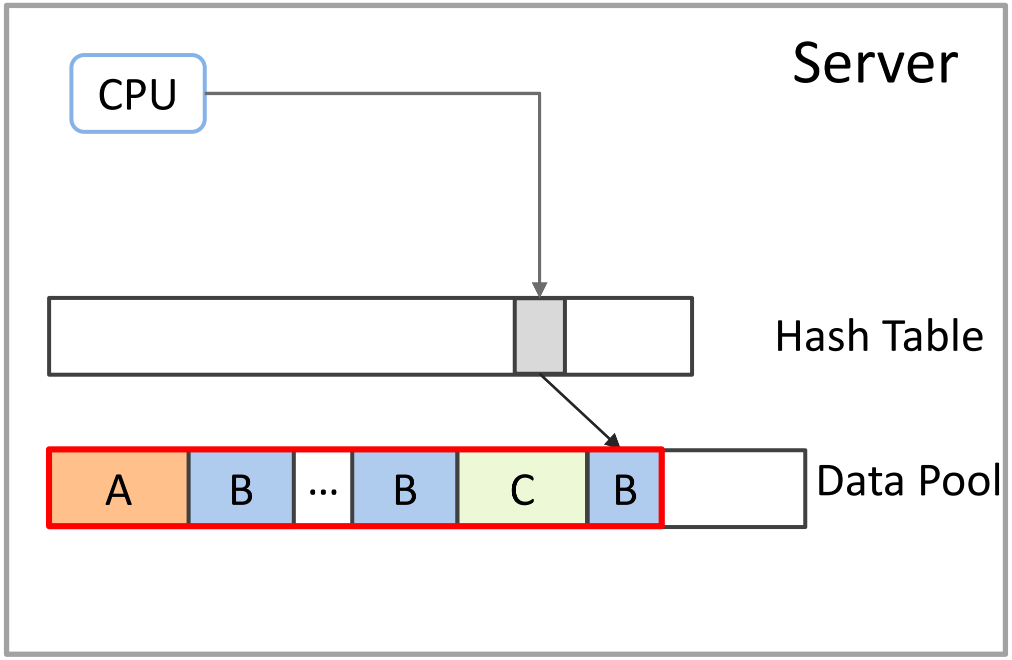
Multi-version Log-structuring Design

① **Out-of-place update with log structuring mechanism**

② **Multi-version linked list for each object**

③ **Embed durability flag in each object**

④ **2-offset region & backlink pointer for lock-free log cleaning**



✓ **Concurrent access**
✓ **Crash consistency**

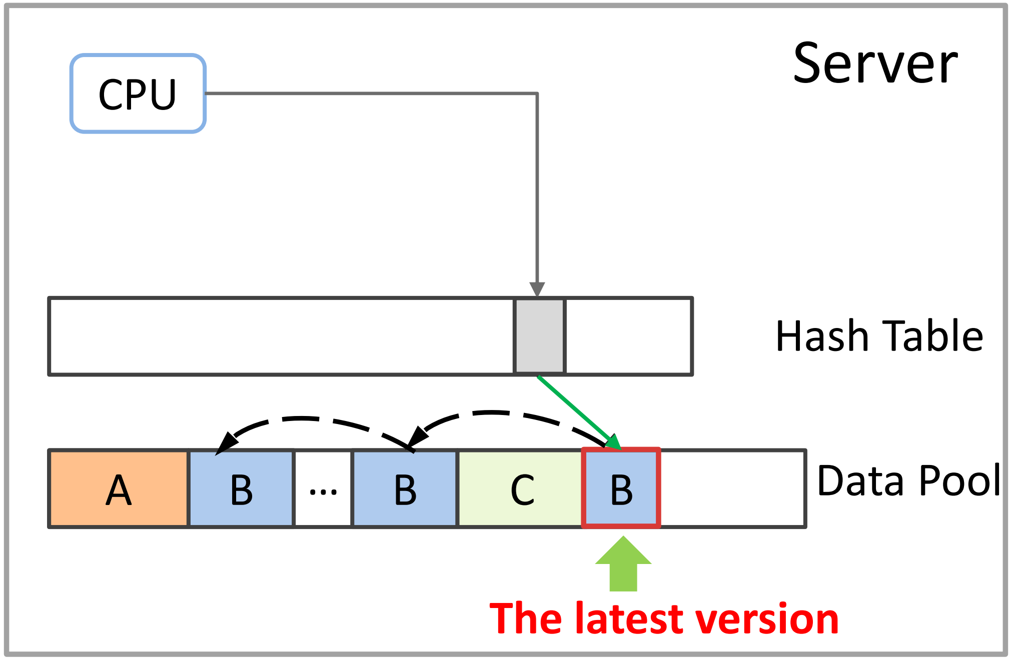
Multi-version Log-structuring Design

① **Out-of-place update with log structuring mechanism**

② **Multi-version linked list for each object**

③ **Embed durability flag in each object**

④ **2-offset region & backlink pointer for lock-free log cleaning**



Multi-version Log-structuring Design

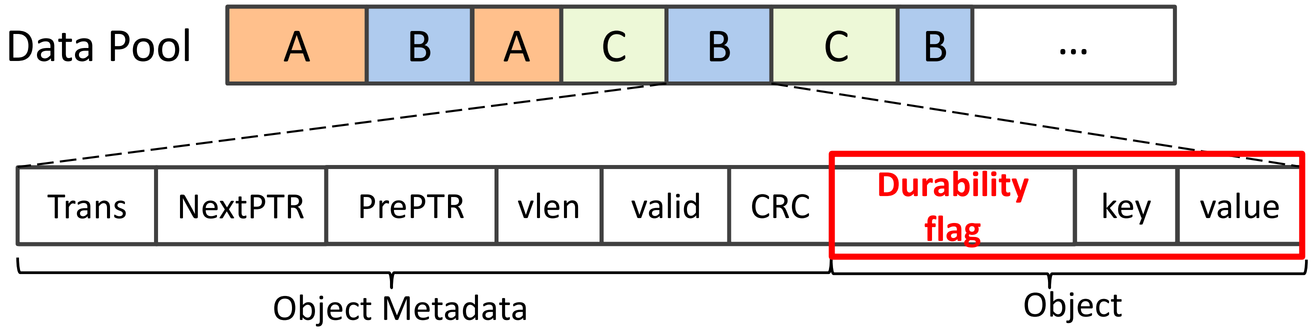
① Out-of-place update with log structuring mechanism

② Multi-version linked list for each object

③ Embed durability flag in each object

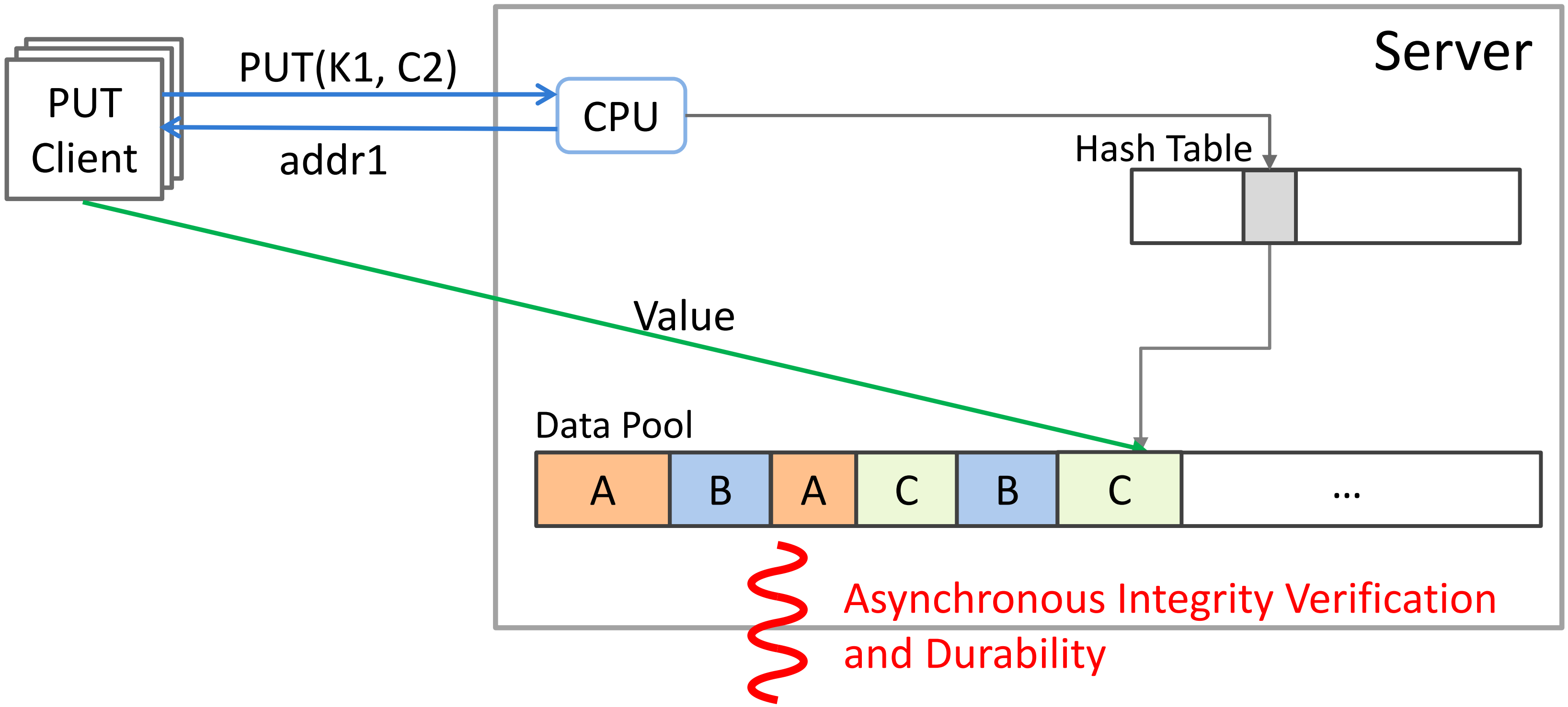
④ 2-offset region & backlink pointer for lock-free log cleaning

- ✓ Help cooperation between foreground thread and background thread
- ✓ Support the hybrid read scheme for high performance



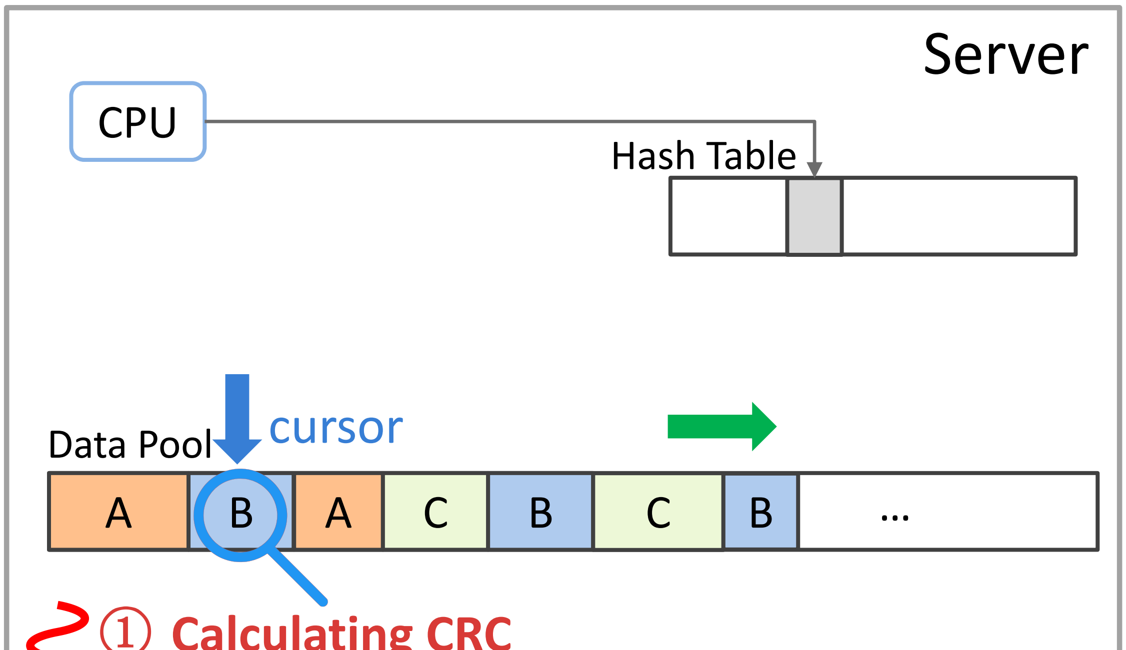
Client-active Write with Async Durability

✓ **High performance for write**



Background Thread for Efficient Consistency

✓ **Reduce persistence and CRC[1] overheads on the critical path**



① Calculating CRC

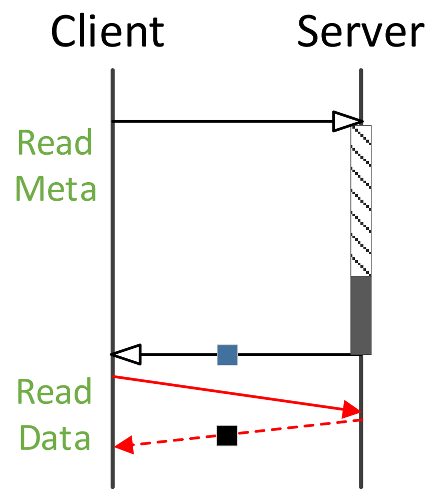
② Persisting value if the object is complete

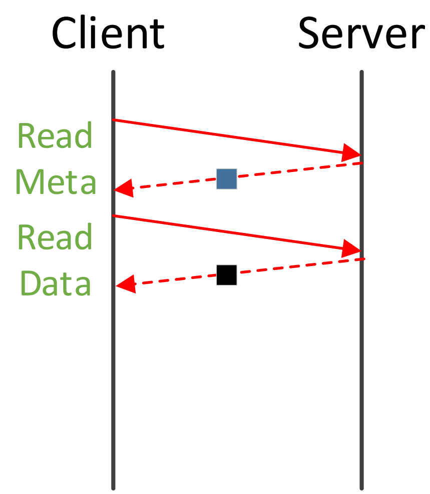
③ Set the durability flag

[1] Cyclic Redundancy Check

Hybrid Read: Basic Methods for GET

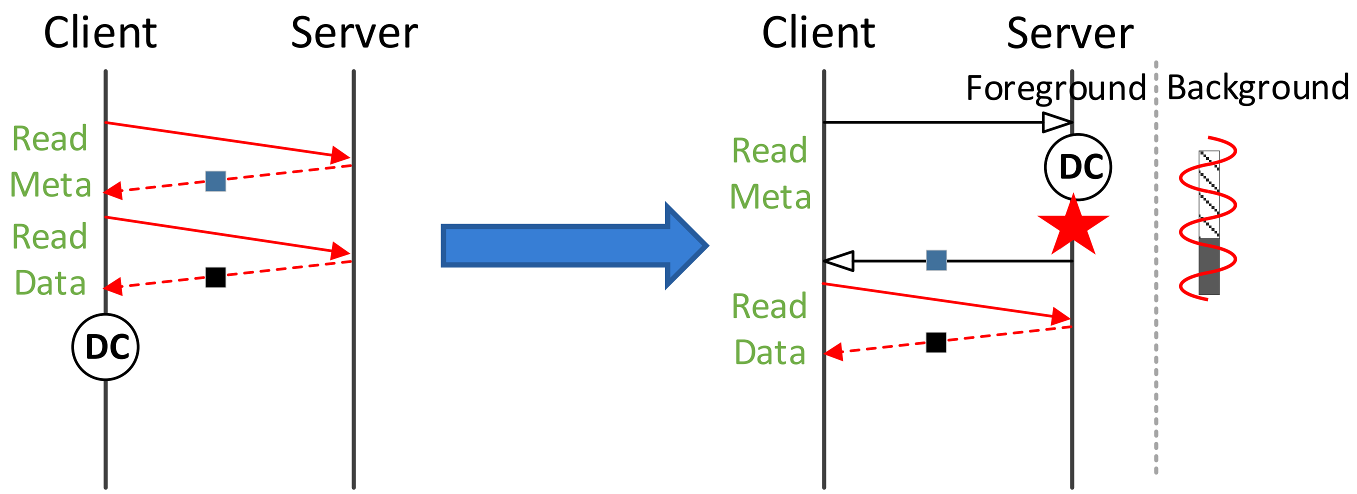
- **RPC+RDMA read**
  - **Easy to ensure data consistency**
  - **Relatively slow**
- **Pure RDMA read**
  - **High performance**
  - **Difficult to ensure data consistency**





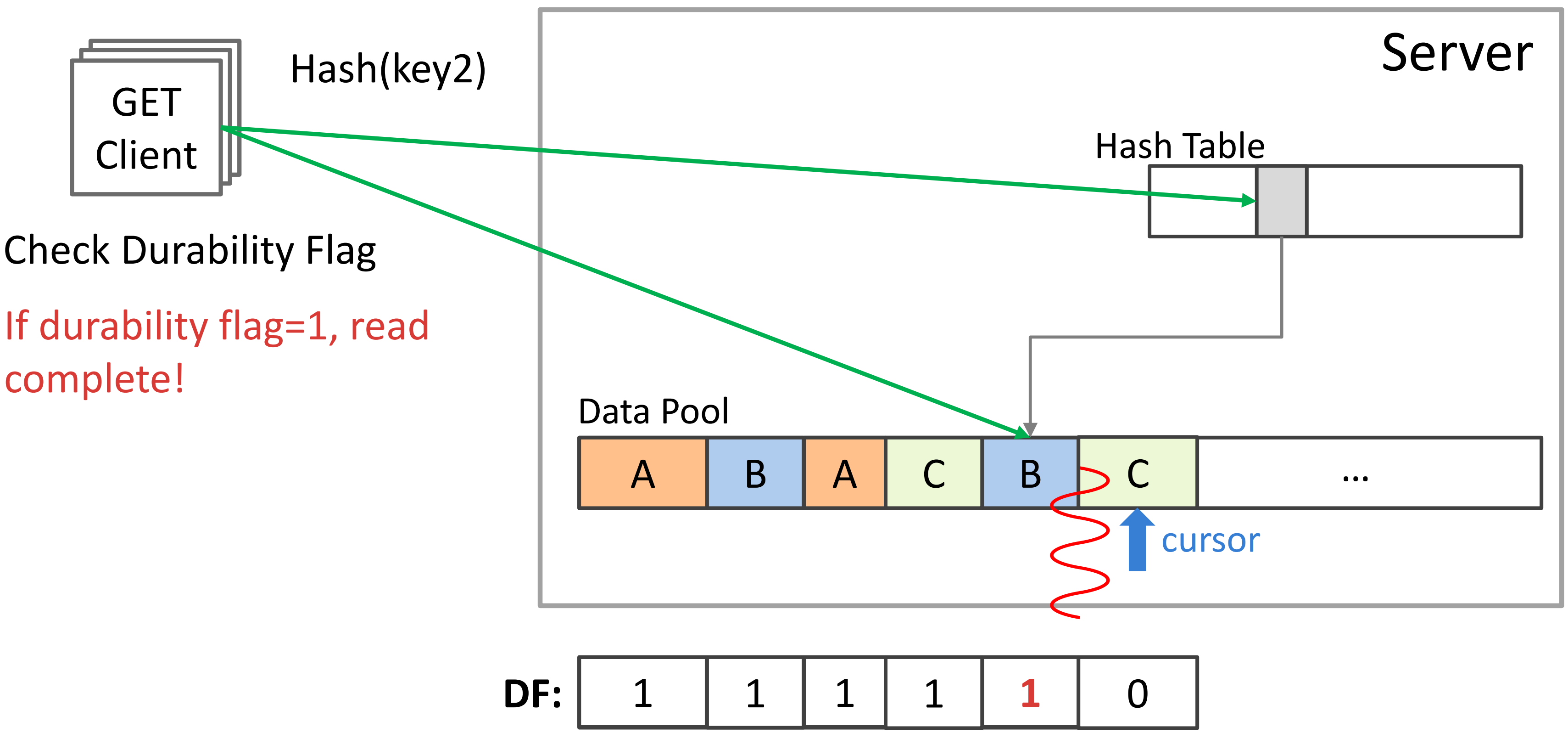
Hybrid Read: A+B

- **Challenge:** **Is it possible to achieve high performance while providing data consistency?**
- **Solution:** **Hybrid Read Scheme**
  - **Use pure RDMA Read method Optimistically**
  - **Fall back to RPC+RDMA read method**
    - ☐ **Selective durability guarantee**



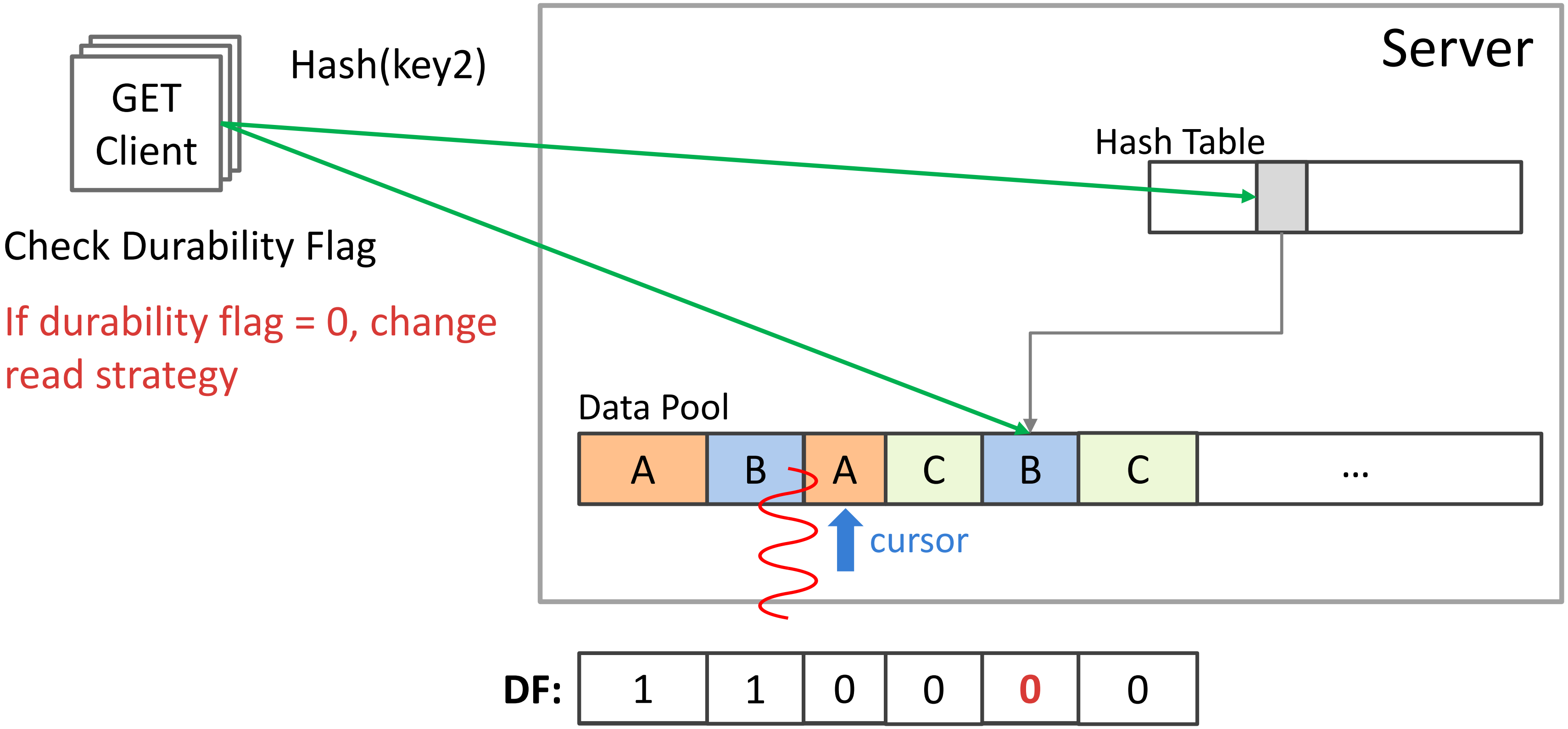
Hybrid Read: Use Pure Read Optimistically

- **GET(Key2): Case 1**

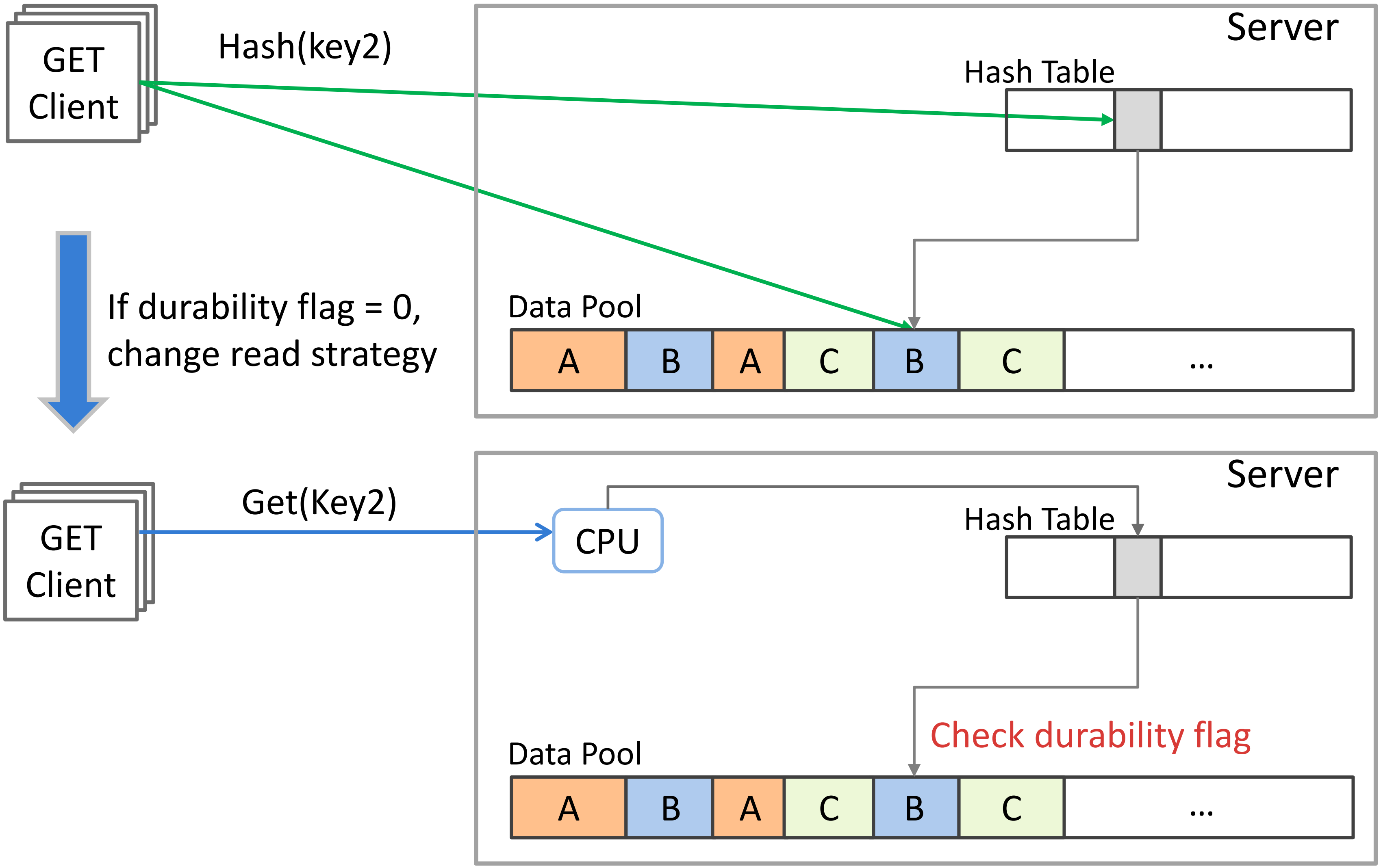


Hybrid Read: Fall back to RPC+RDMA Read

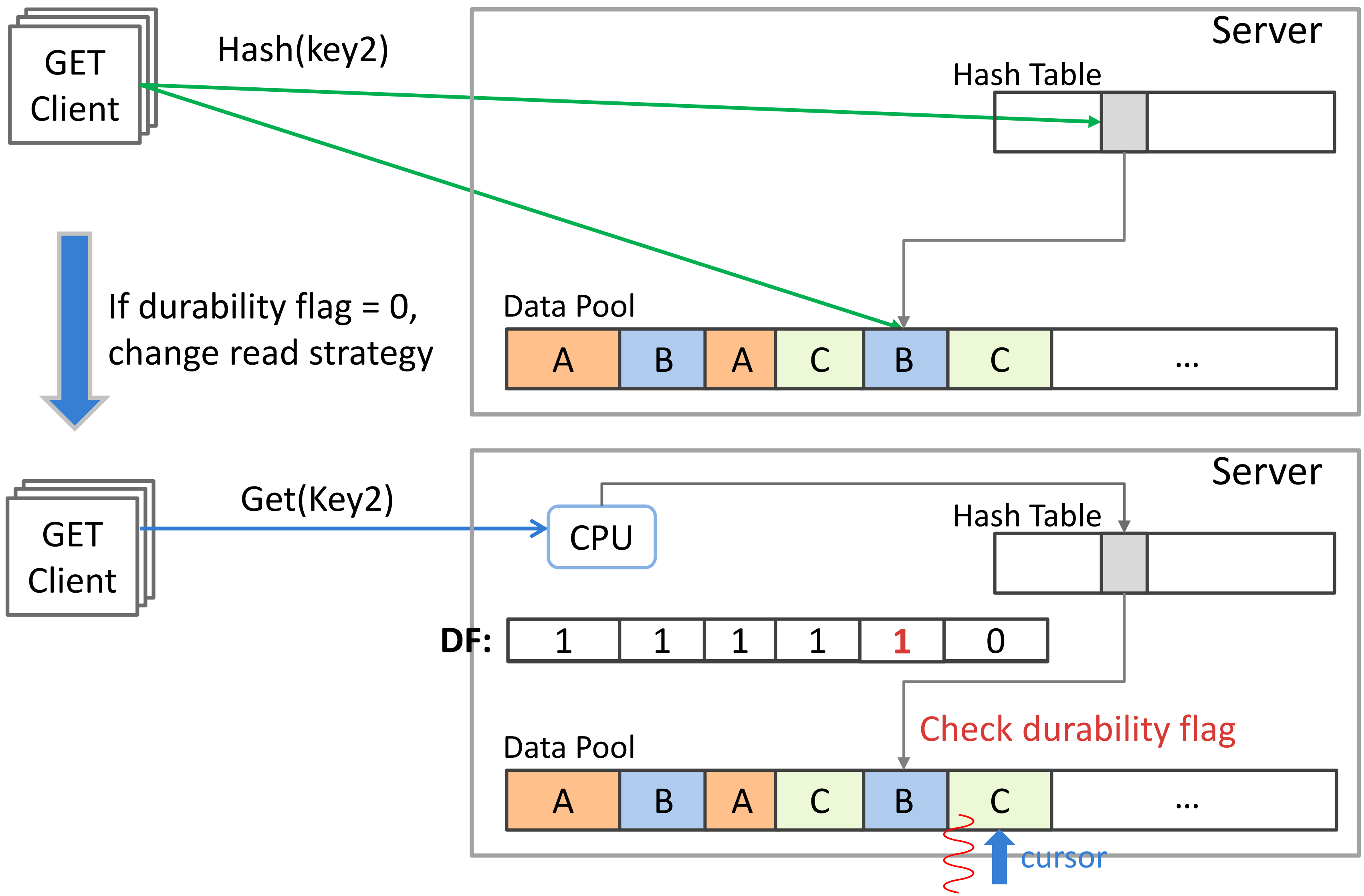
- **GET(Key2): Case 2**



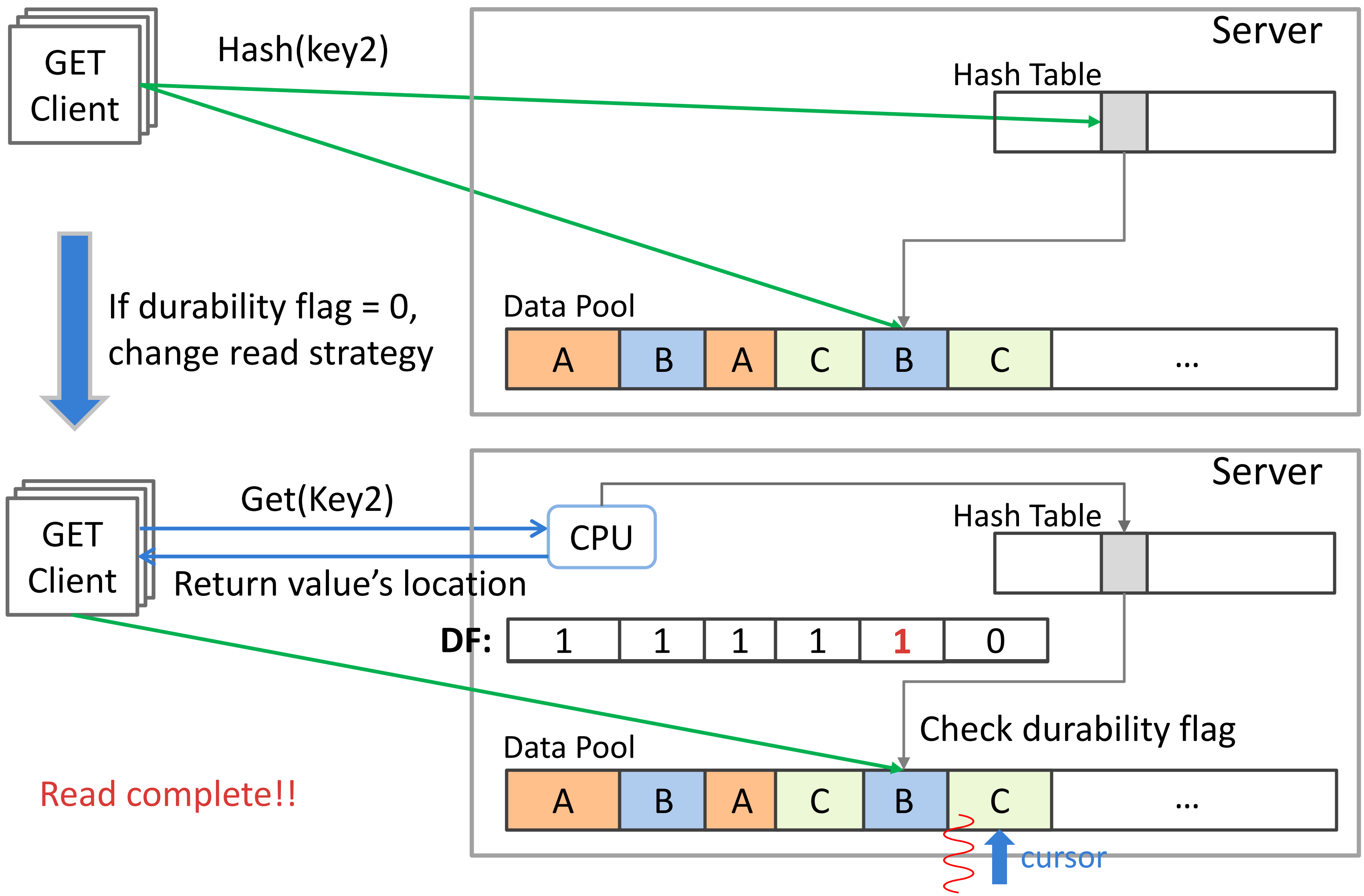
Hybrid Read: Fall back to RPC+RDMA Read



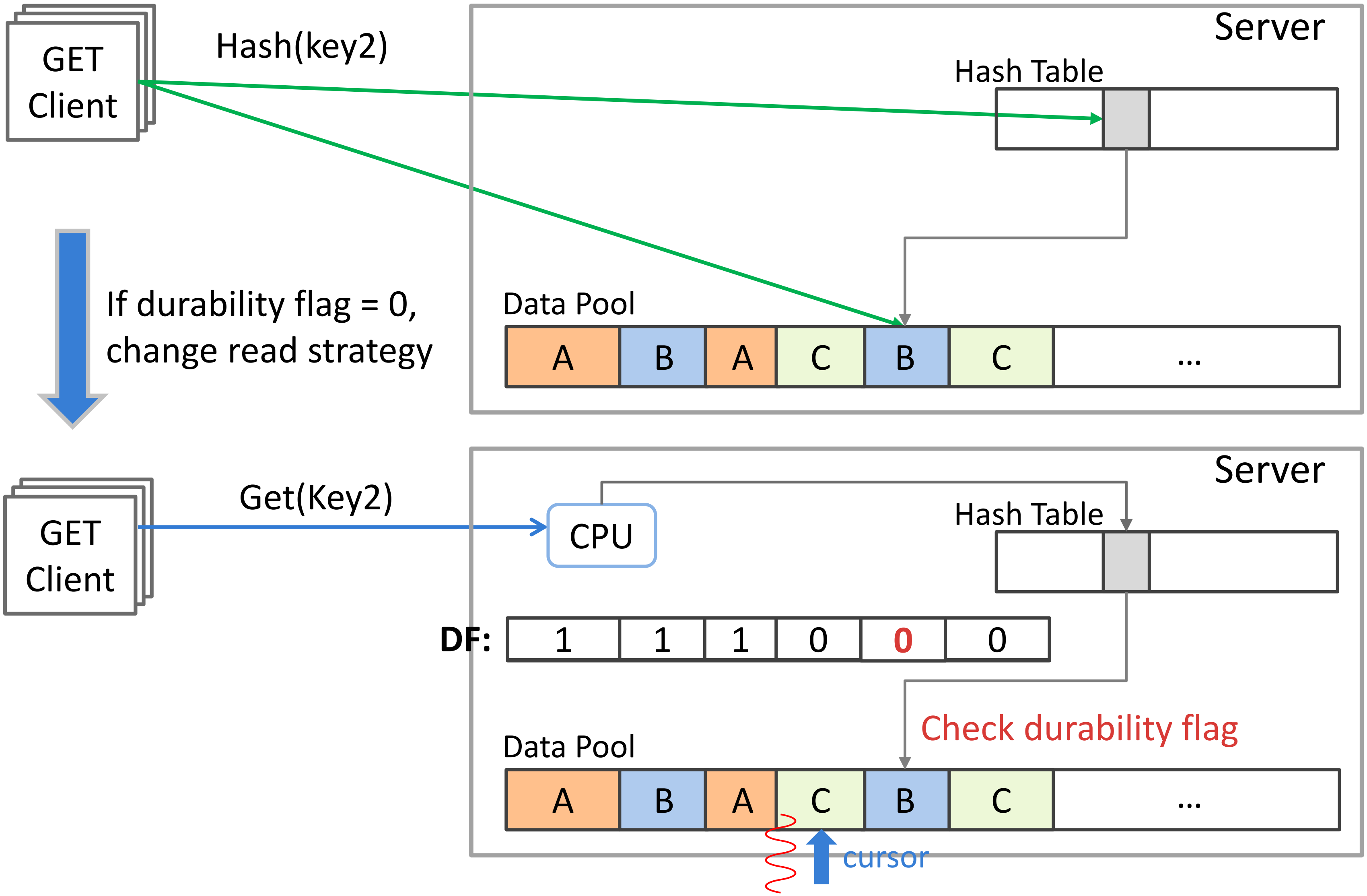
Hybrid Read: Fall back to RPC+RDMA Read



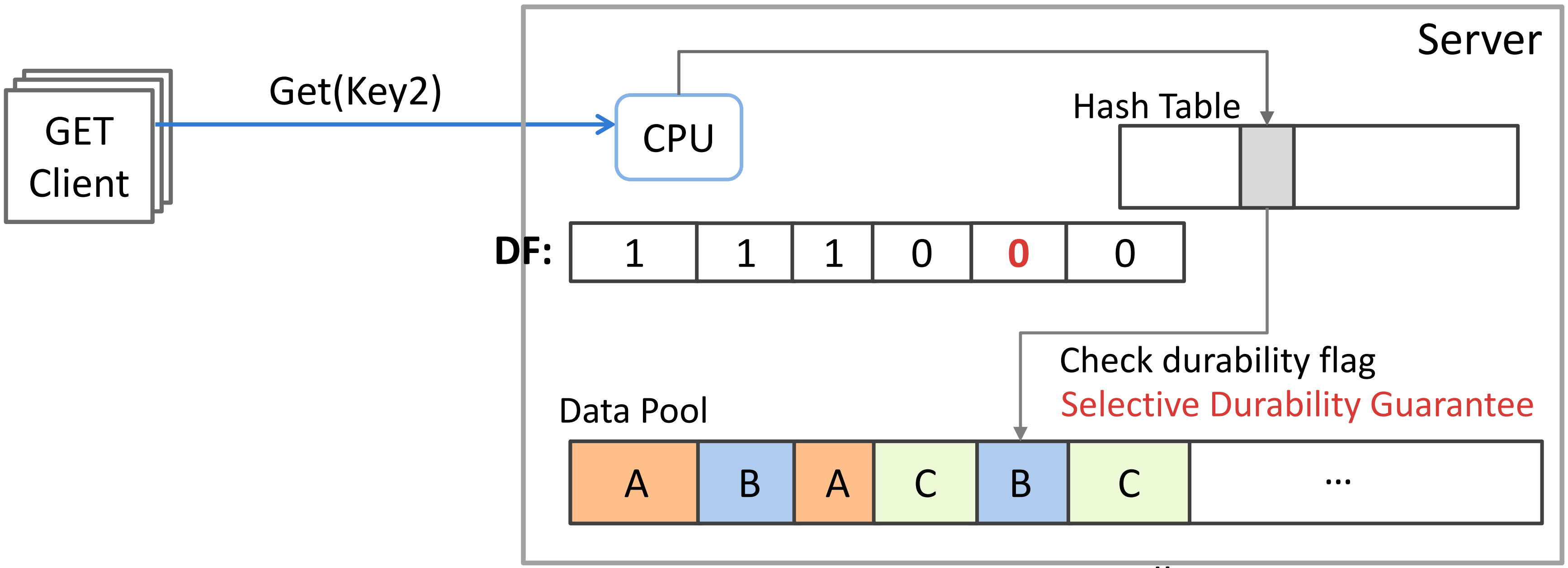
Hybrid Read: Fall back to RPC+RDMA Read



Hybrid Read: Selective Durability Guarantee



Hybrid Read: Selective Durability Guarantee

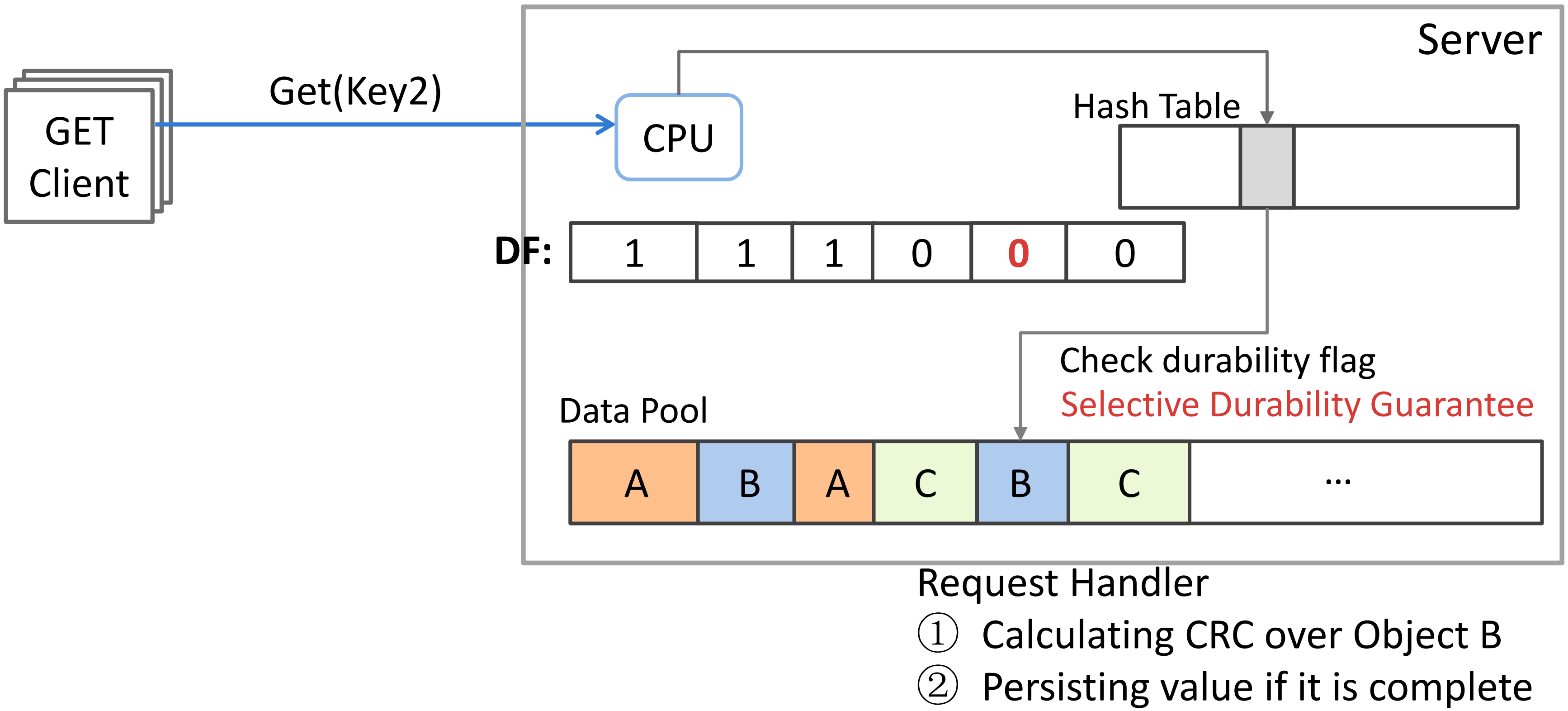


Request Handler

① Calculating CRC over Object B

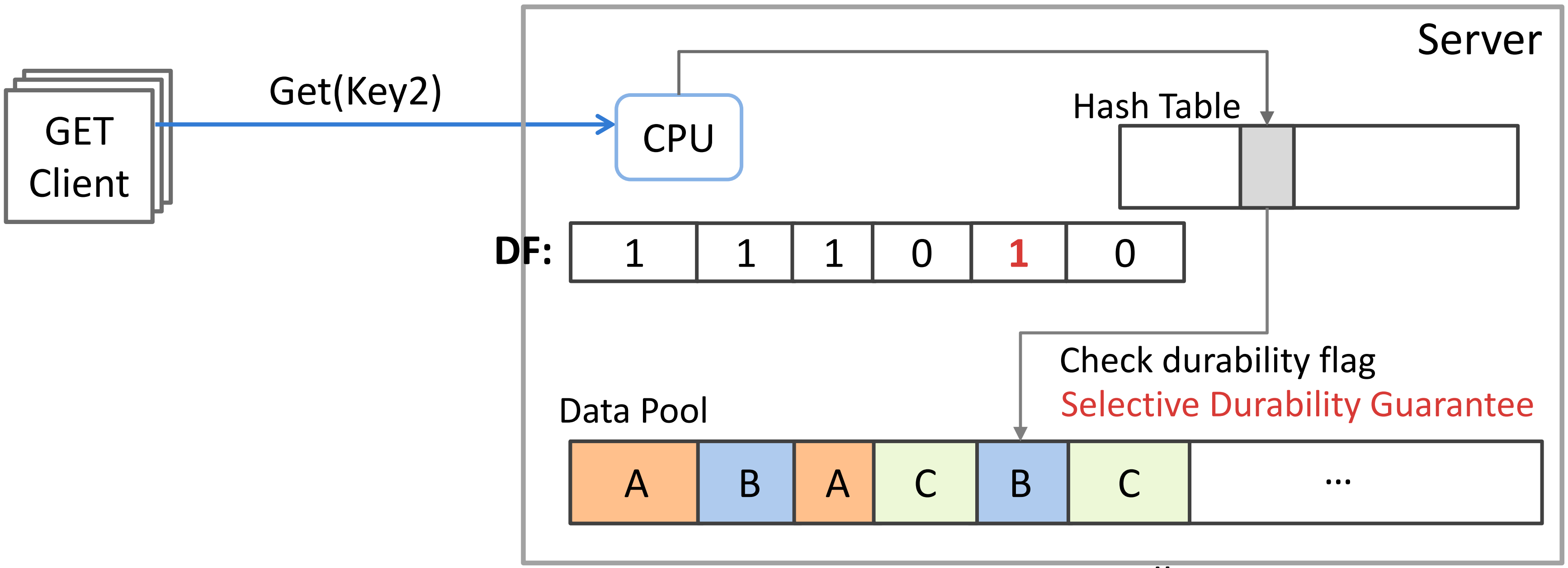
Hybrid Read: Selective Durability Guarantee

- **Case 1: Calculated CRC == Stored CRC**



Hybrid Read: Selective Durability Guarantee

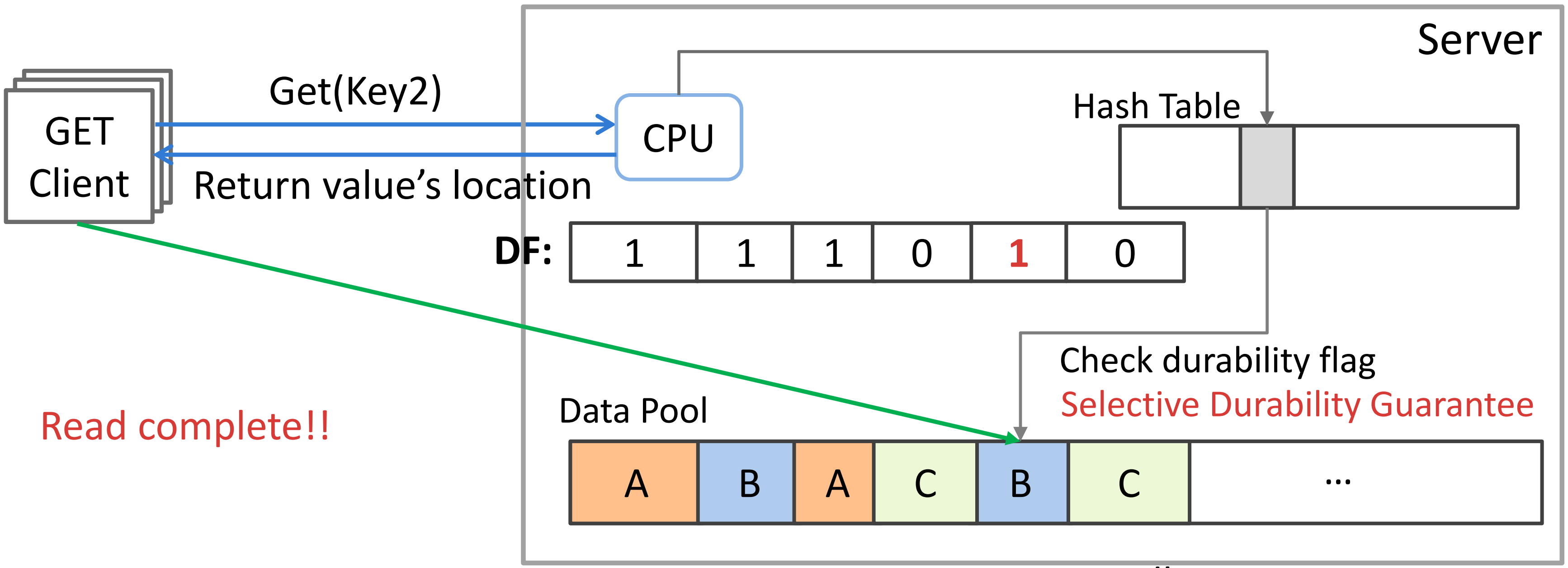
- **Case 1: Calculated CRC == Stored CRC**



Request Handler
① Calculating CRC over Object B
② Persisting value if it is complete
③ Set durability flag

Hybrid Read: Selective Durability Guarantee

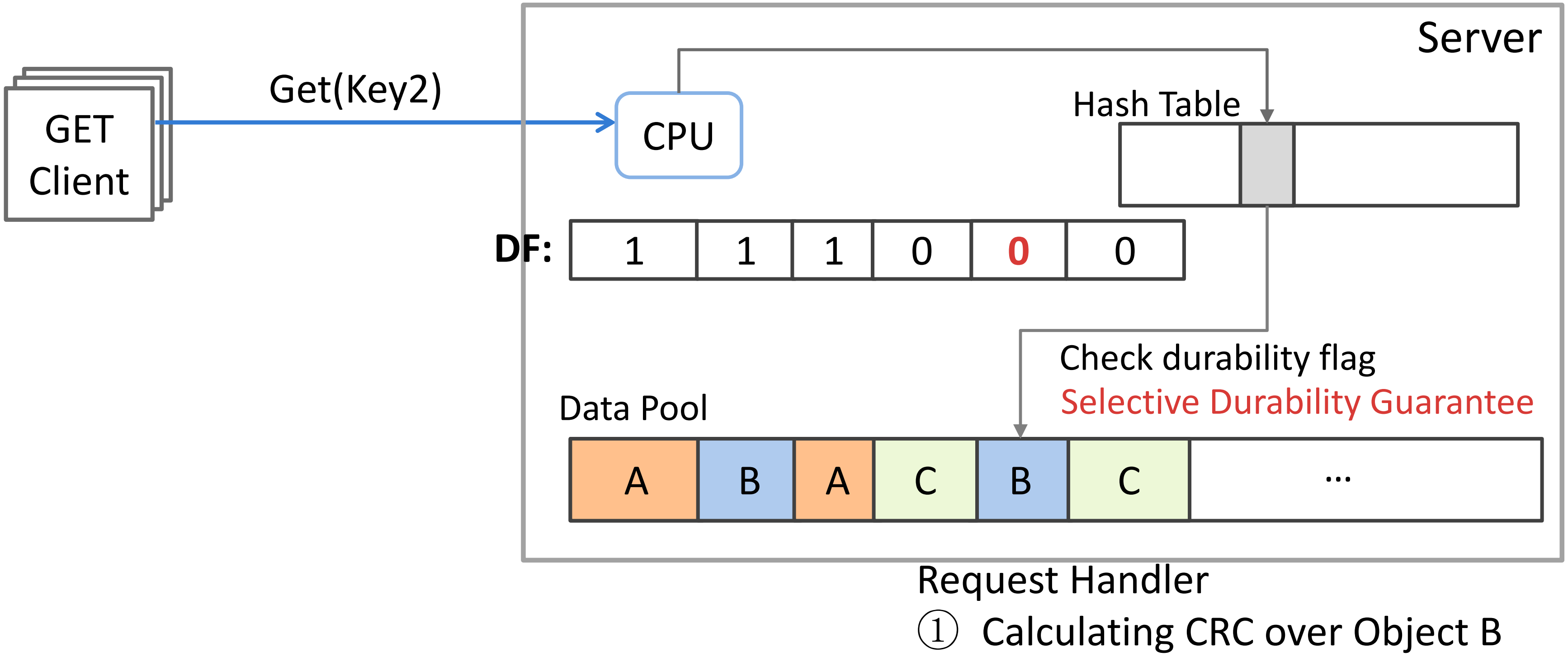
- **Case 1: Calculated CRC == Stored CRC**



Request Handler
① Calculating CRC over Object B
② Persisting value if it is complete
③ Set durability flag

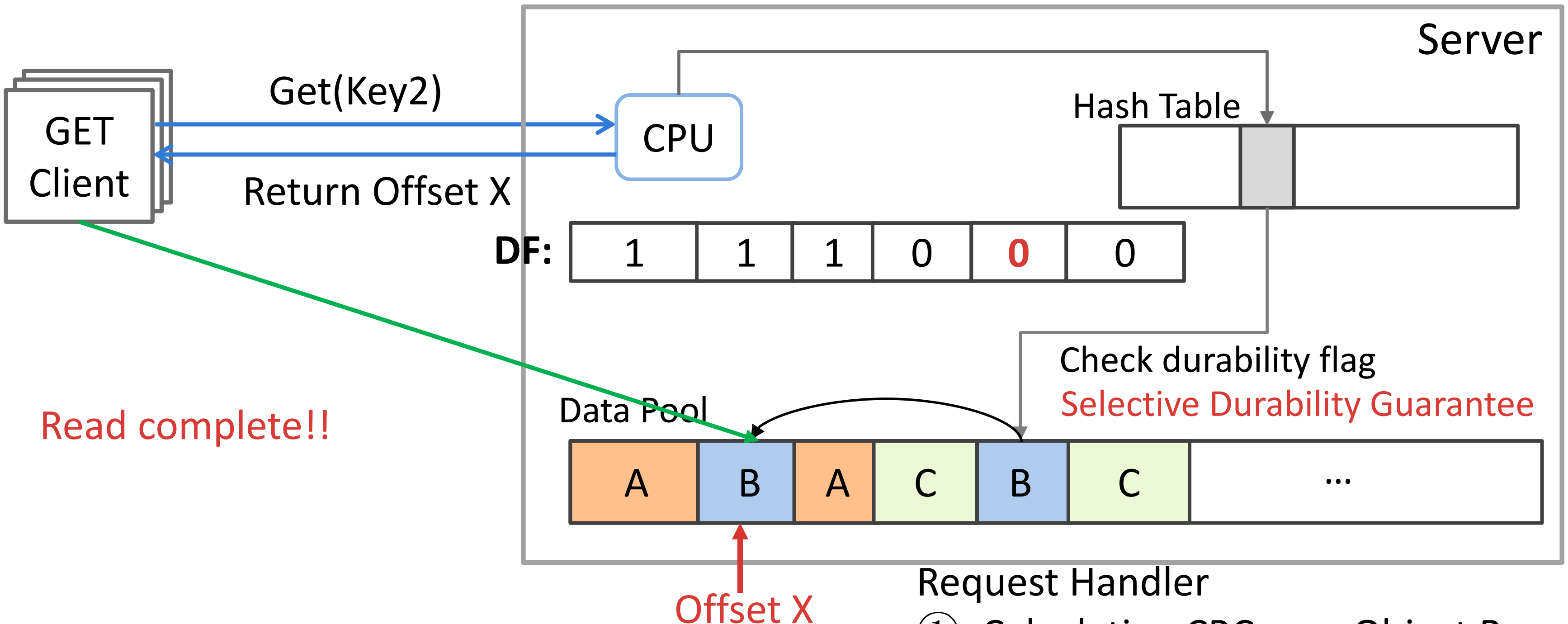
Hybrid Read: Selective Durability Guarantee

- **Case 2: Calculated CRC != Stored CRC**



Hybrid Read: Selective Durability Guarantee

- **Case 2: Calculated CRC != Stored CRC**



Request Handler

① Calculating CRC over Object B

② Traverse the version list until find the completely durable version

Hybrid Read Scheme

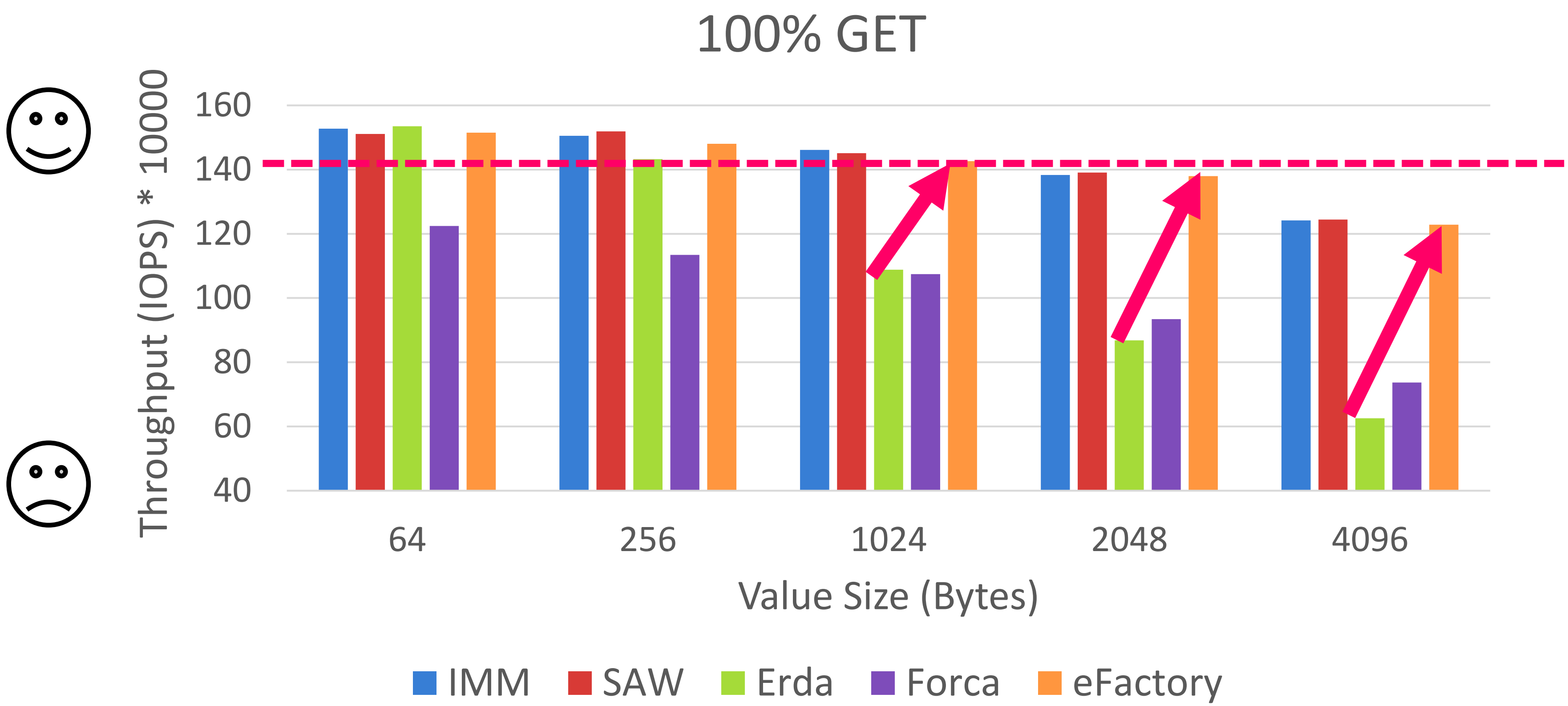
- ✓ **High Performance**
  - Use pure RDMA read method optimistically
  - Durability flag: verify data consistency efficiently
  - Selective durability guarantee: no need to wait bg thread
- ✓ **Consistency Guarantee**
  - Background thread: async verification and durability
  - Request Handler: selective durability guarantee

Performance Evaluation

- Environment
  - Use PMDK[1] to emulate persistent memory
  - YCSB[2] benchmark (Zipfian distribution)
  - Mellanox ConnectX-5 InfiniBand NIC
  - Two 10-Core CPUs with 25MB L3 cache
- Apple-to-apple Comparison on the same codebase
  - SAW: Send after Write @SDC'15
  - IMM: Solutions using write_with_imm @FAST'19
  - Forca @ICCD'18
  - Erda @arvix'19

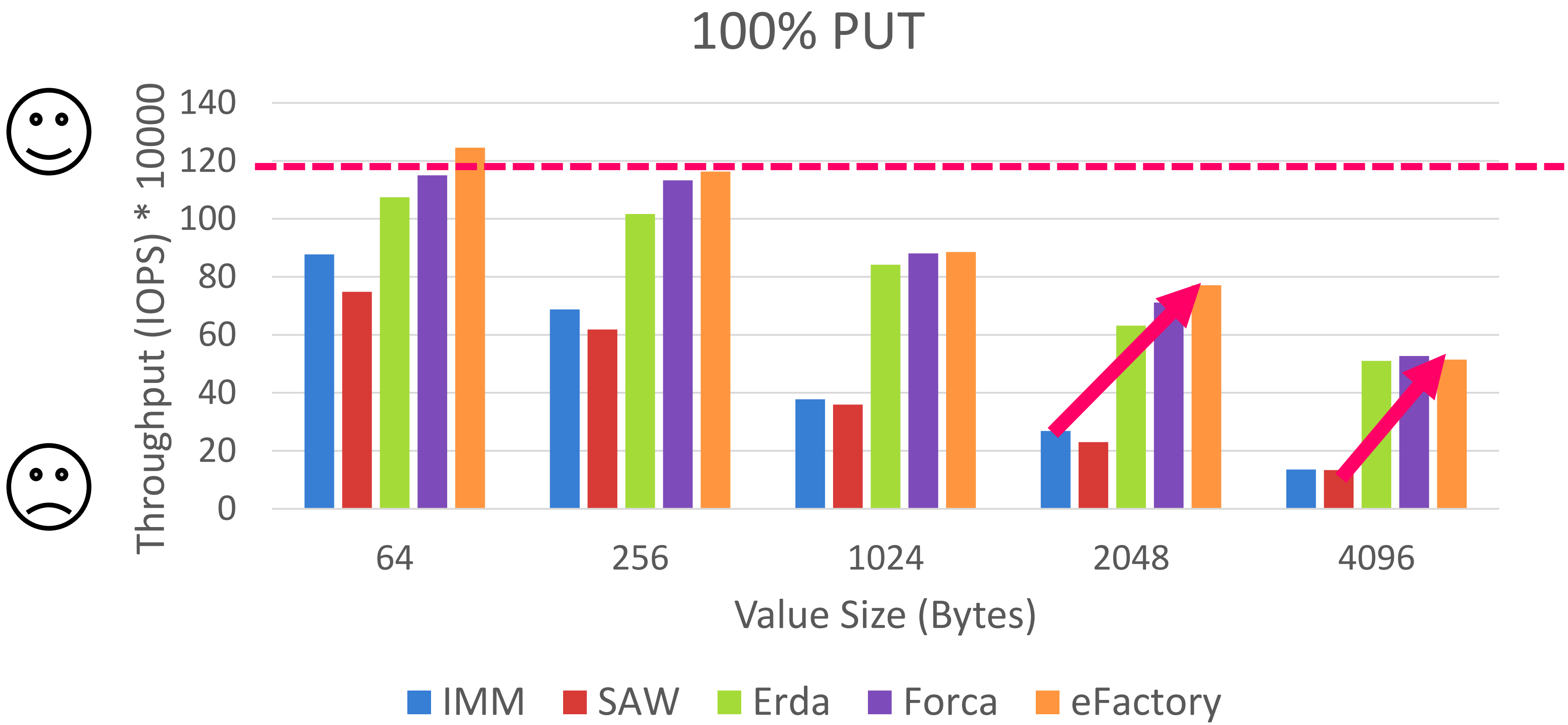
[1] 2016. How to emulate Persistent Memory. http://pmem.io/2016/02/22/pm-emulation.html
[2] YCSB: https://github.com/brianfrankcooper/YCSB @ SOCC'10

Read Throughput



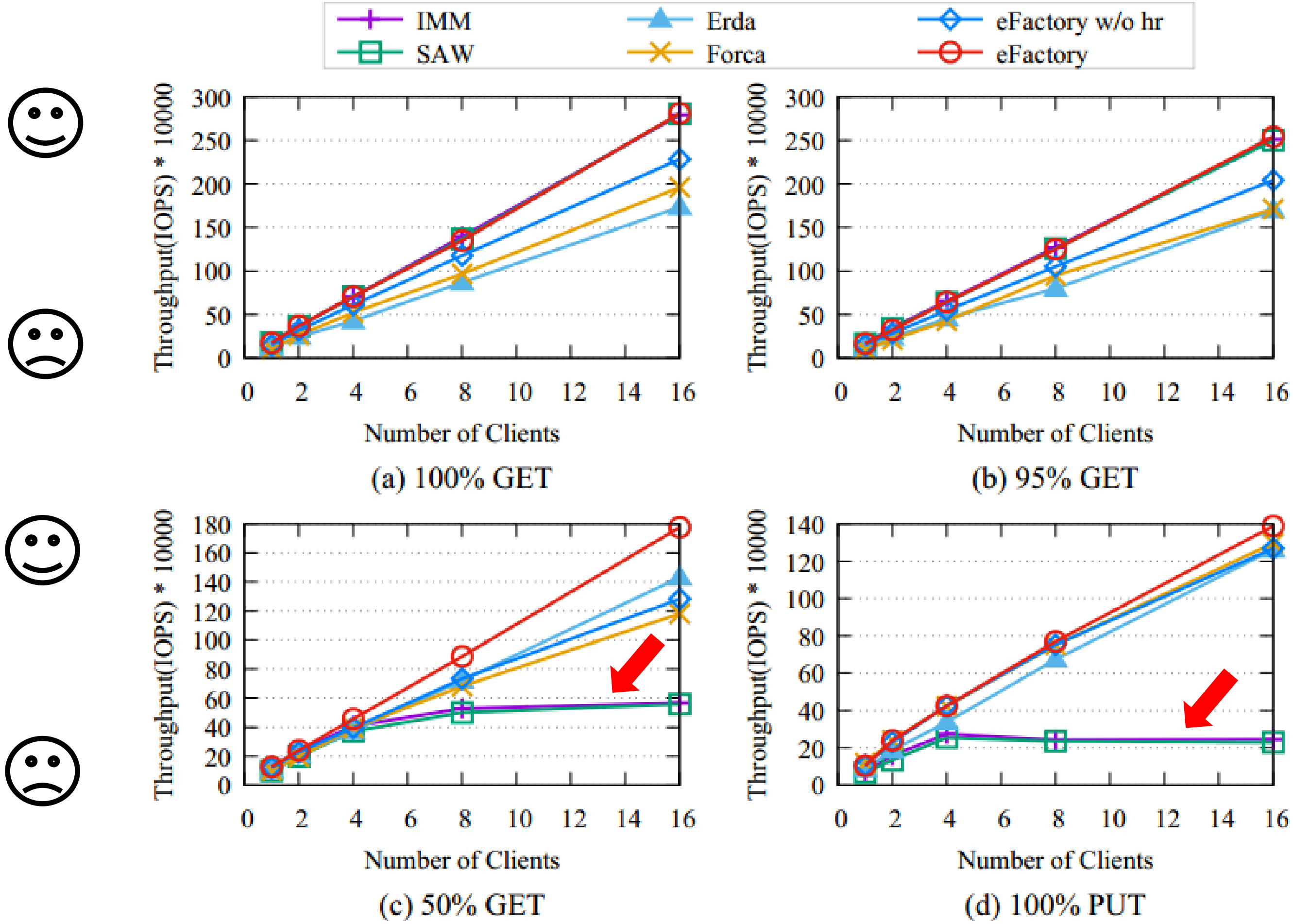
- ***Comparable read throughput as IMM and SAW (Gap: 2%)***
- ***Better than Erda (1.3x-1.96x) and Forca (1.24x-1.67x)***

Write Throughput



- ***Better than IMM (0.42x-2.79x) and SAW (0.66x-2.85x)***
- ***Slightly higher than Erda and Forca***

Scalability



When write dominates, IMM and SAW fail to scale well

Conclusion

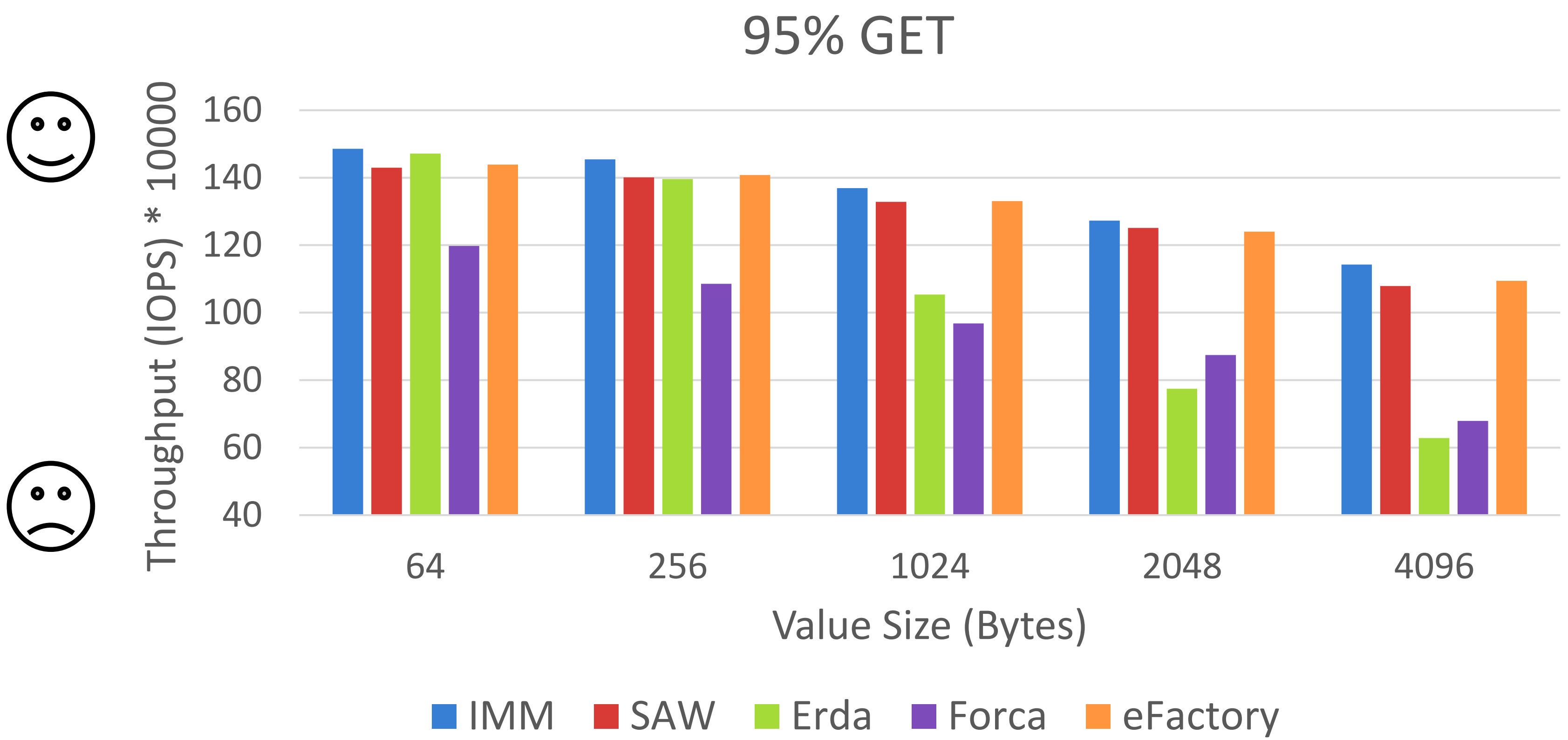
- Existing solutions sacrifice either read or write performance to guarantee data consistency of RDMA-based NVM systems
- We propose eFactory, a multi-version log-structuring design
  - Client-active Write with Asynchronous Durability
  - Single Background Thread for Efficient Consistency
  - Hybrid Read Scheme
  - Lock-free Log Cleaning Scheme
- Results: high performance for both read and write while providing data consistency
  - Outperforms IMM and SAW by 0.42x-2.79x and 0.66x-2.85x for write
  - 1.3x-1.96x and 1.24x-1.67x of Erda and Forca for read

Thanks for listening

Q&A

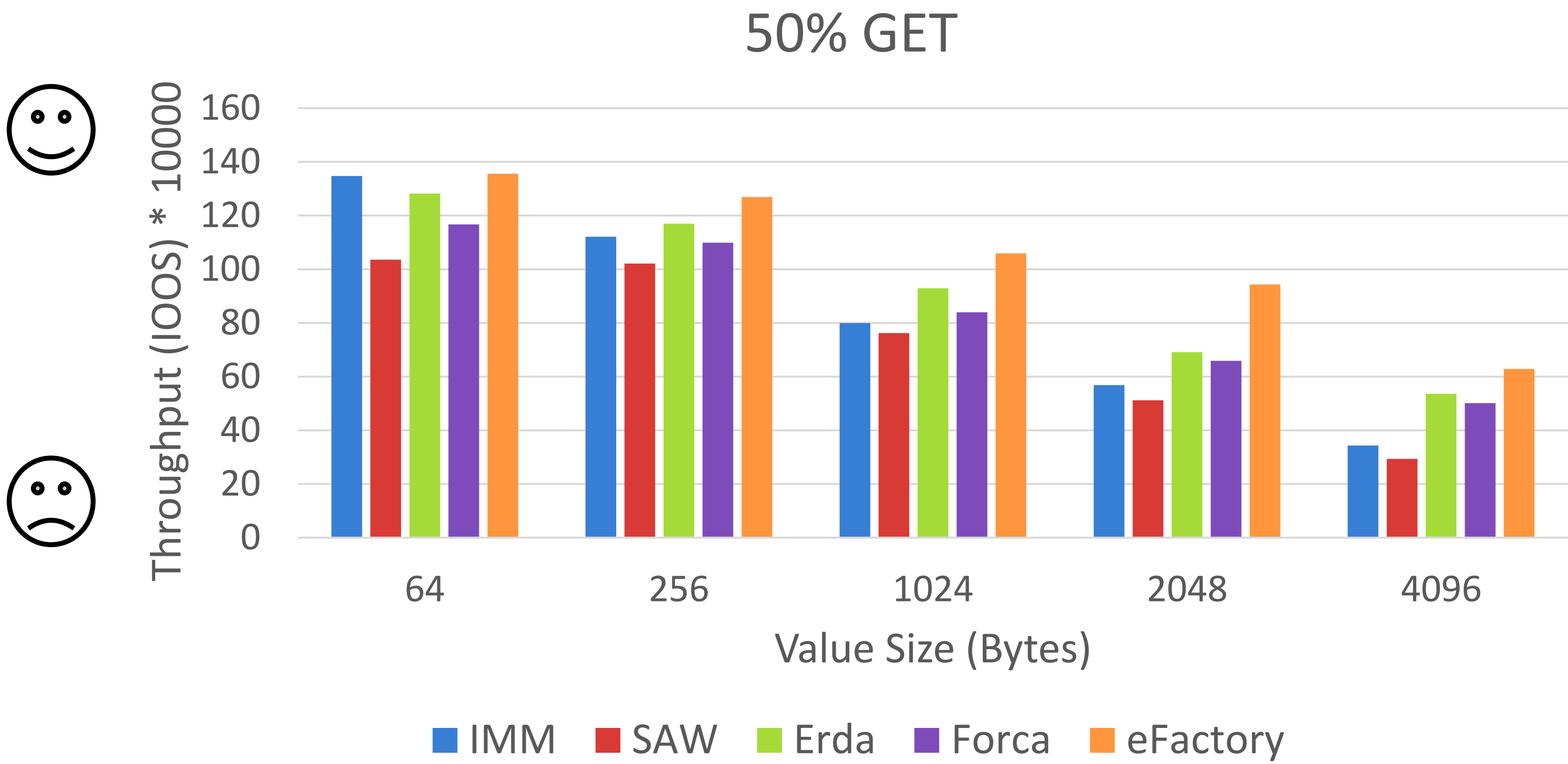
Backup

Throughput with read-intensive workload



- ***Accounts for 95% of IMM's throughput***
- ***Outperforms Erda and Forca by 0.26x-0.74x and 0.2x-0.61x***

Throughput with write-intensive workload



➢ ***Achieves the highest throughput for all the value sizes***

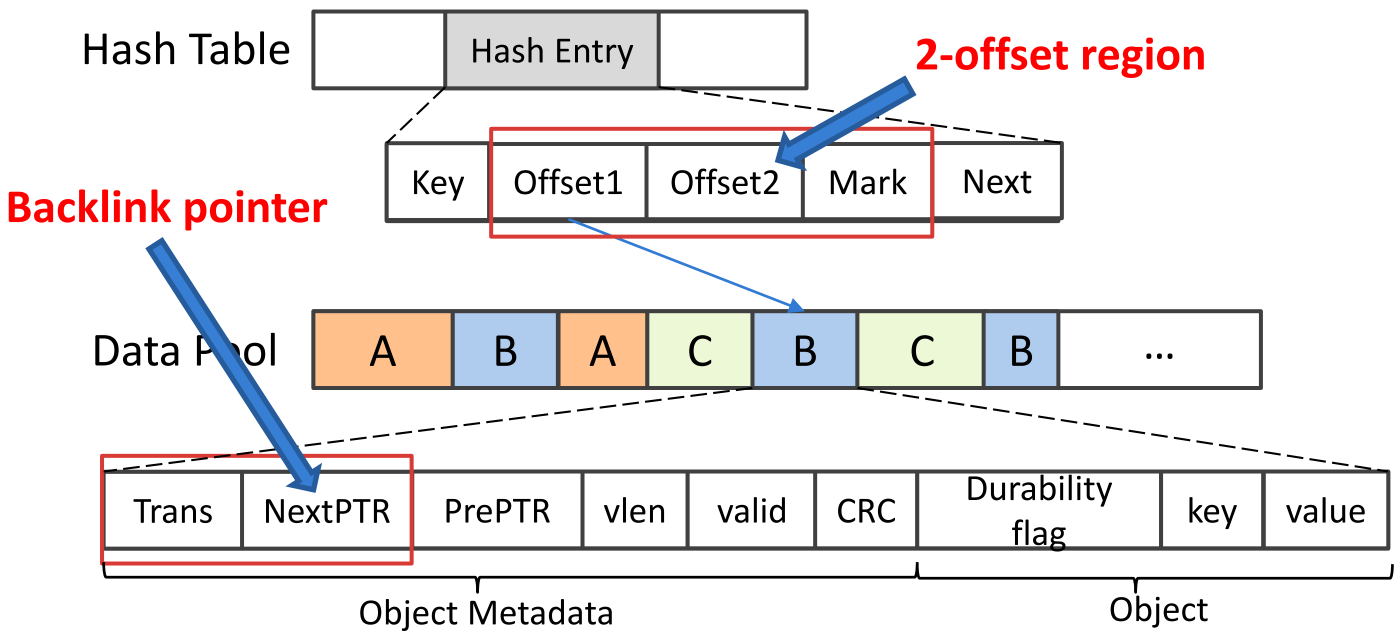
Multi-version Log-structuring Design

① **Out-of-place update with log structuring mechanism**

② **Multi-version linked list for each object**

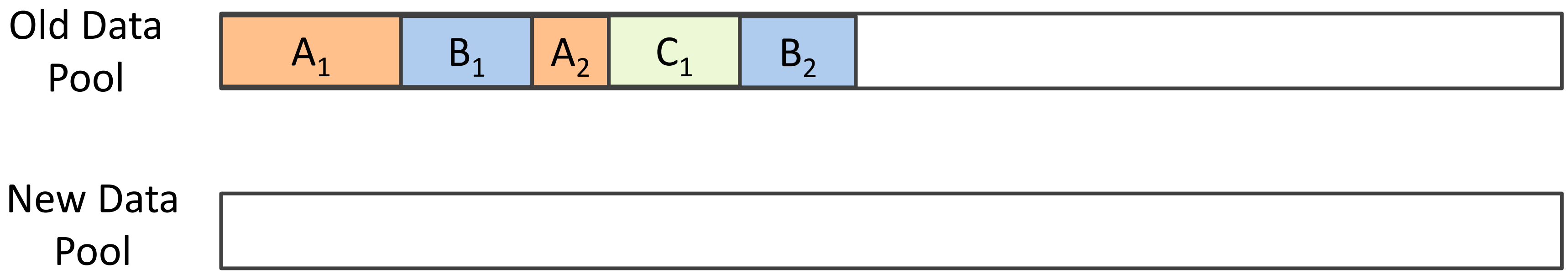
③ **Embed durability flag in each object**

④ **2-offset region & backlink pointer for lock-free log cleaning**



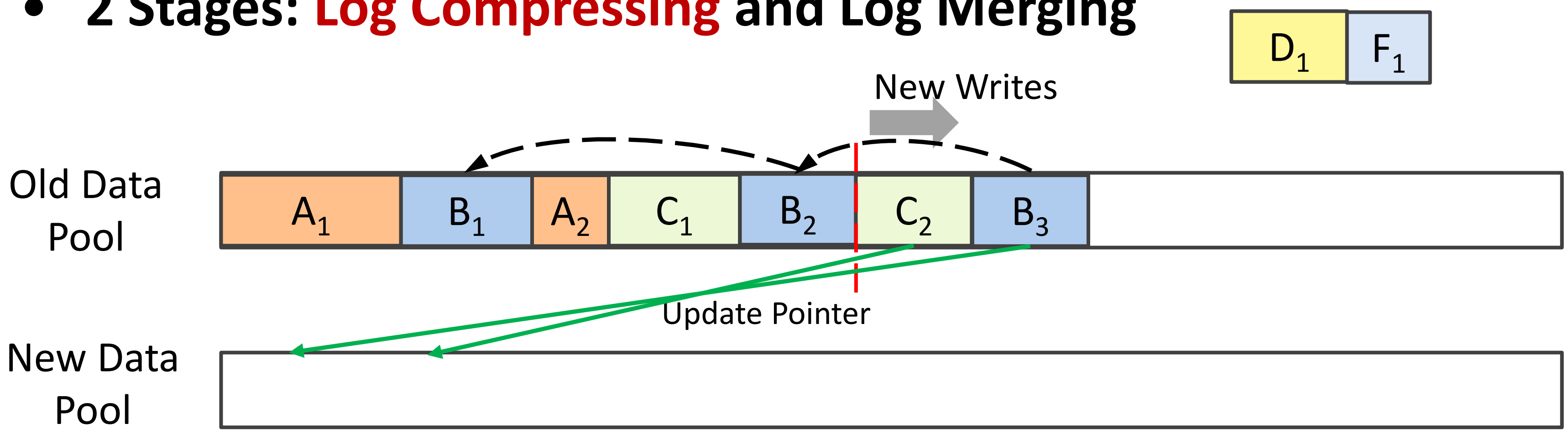
Lock-free Log Cleaning Scheme

- **2 Stages: Log Compressing and Log Merging**



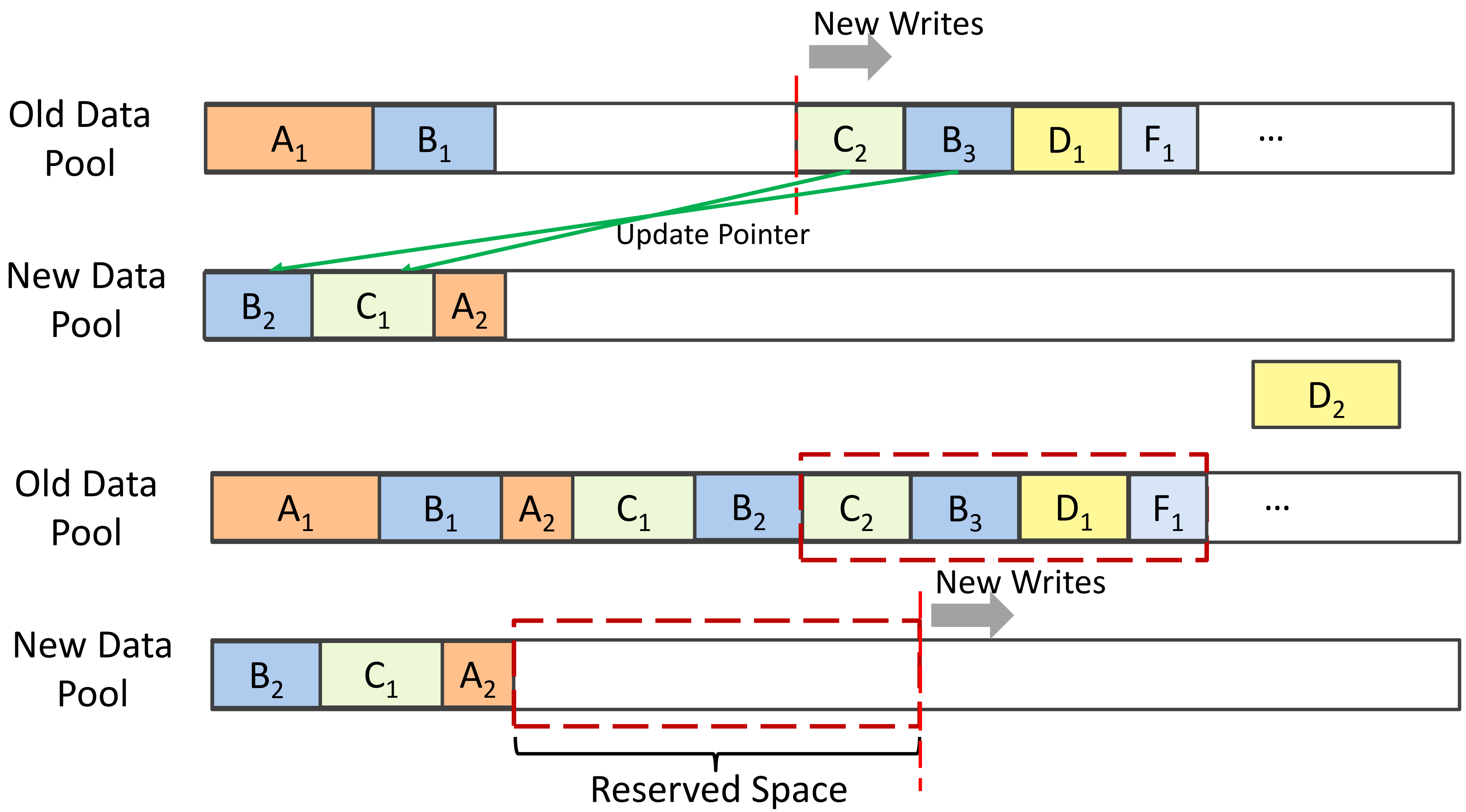
Lock-free Log Cleaning Scheme

- **2 Stages: Log Compressing and Log Merging**



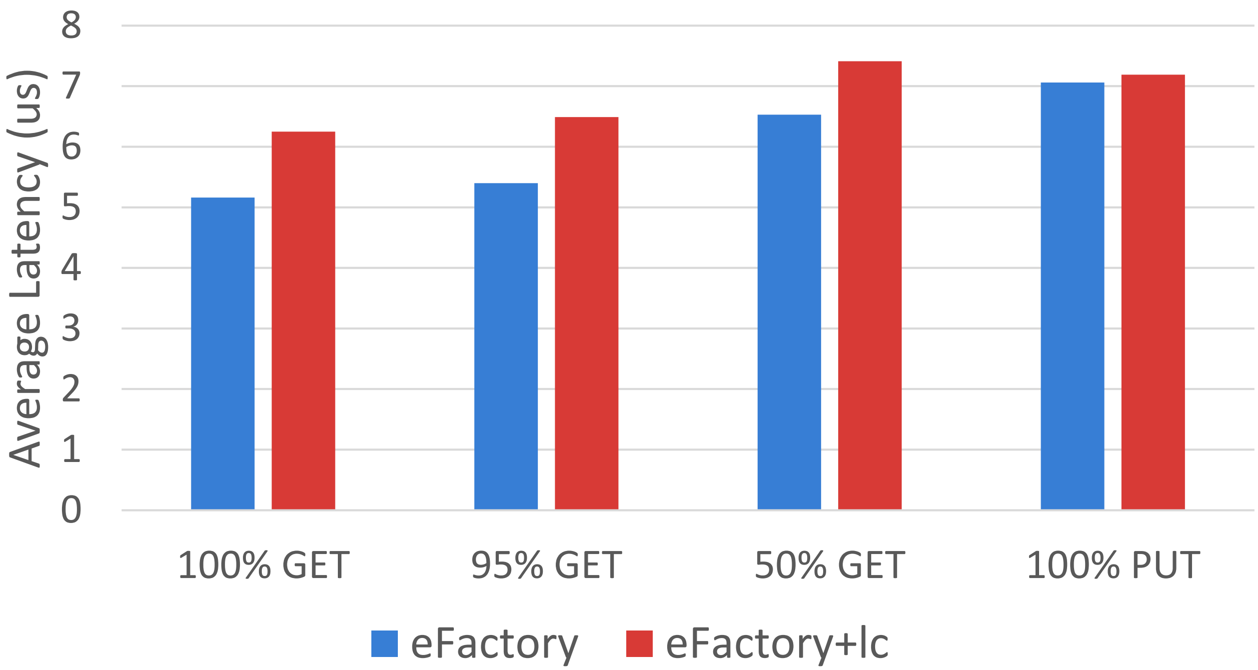
Lock-free Log Cleaning Scheme

- **2 Stages: Log Compressing and Log Merging**



Performance Impact by Log Cleaning

- **Performance decreases since the reading process falls back to RPC+RDMA read scheme**



Overall, log cleaning incurs 1%-21% performance overhead