Closing the Performance Gap Between Volatile and Persistent K-V Stores

Yihe Huang, Harvard University
Matej Pavlovic, EPFL
Virendra Marathe, Oracle Labs
Margo Seltzer, Oracle Labs
Tim Harris, Oracle Labs
Steve Byan, Oracle Labs

ATC 2018, 07/13/2018, Boston MA, USA

Key-Value Stores: Persistent or Fast?

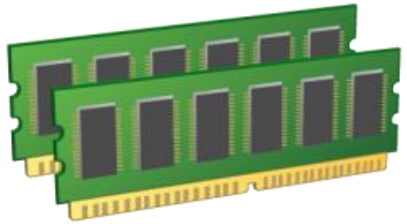
Persistent

Slow

or

Volatile

Fast

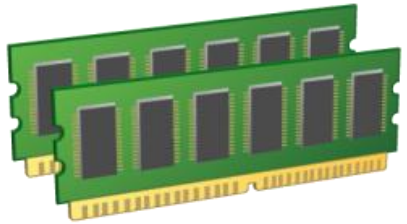


Key-Value Stores: Persistent or Fast?

✔ Persistent
✖ Slow

or

✖ Volatile
✔ Fast



What about both?

Non-Volatile Memory

- ✔ Faster than HDD/SSD
- ✔ Byte-addressable

Non-Volatile Memory

- ✔ Faster than HDD/SSD
- ✔ Byte-addressable
- ✖ Slower than DRAM
- ✖ Tricky to manage consistently

Non-Volatile Memory

✔ Faster than HDD/SSD

✔ Byte-addressable

✖ Slower than DRAM

✖ Tricky to manage consistently

DRAM K-V stores don’t easily translate to NVM

Outline

- Cross-Referencing Logs:
  Getting the best of DRAM and NVM

- Bullet:
  Fast and persistent key-value store

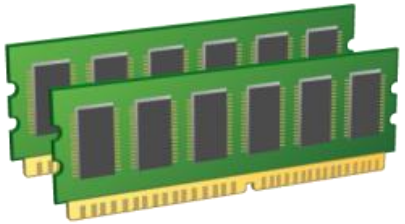
Outline

- **Cross-Referencing Logs:**
  Getting the best of DRAM and NVM

- Bullet:
  Fast and persistent key-value store

Caching

Volatile store (frontend)

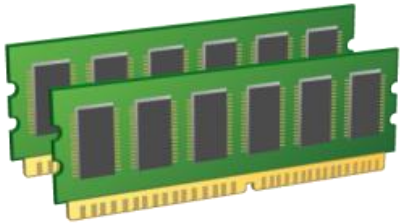


caches

Persistent store (backend)

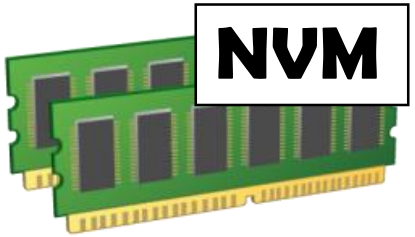
Caching

Volatile store
(frontend)

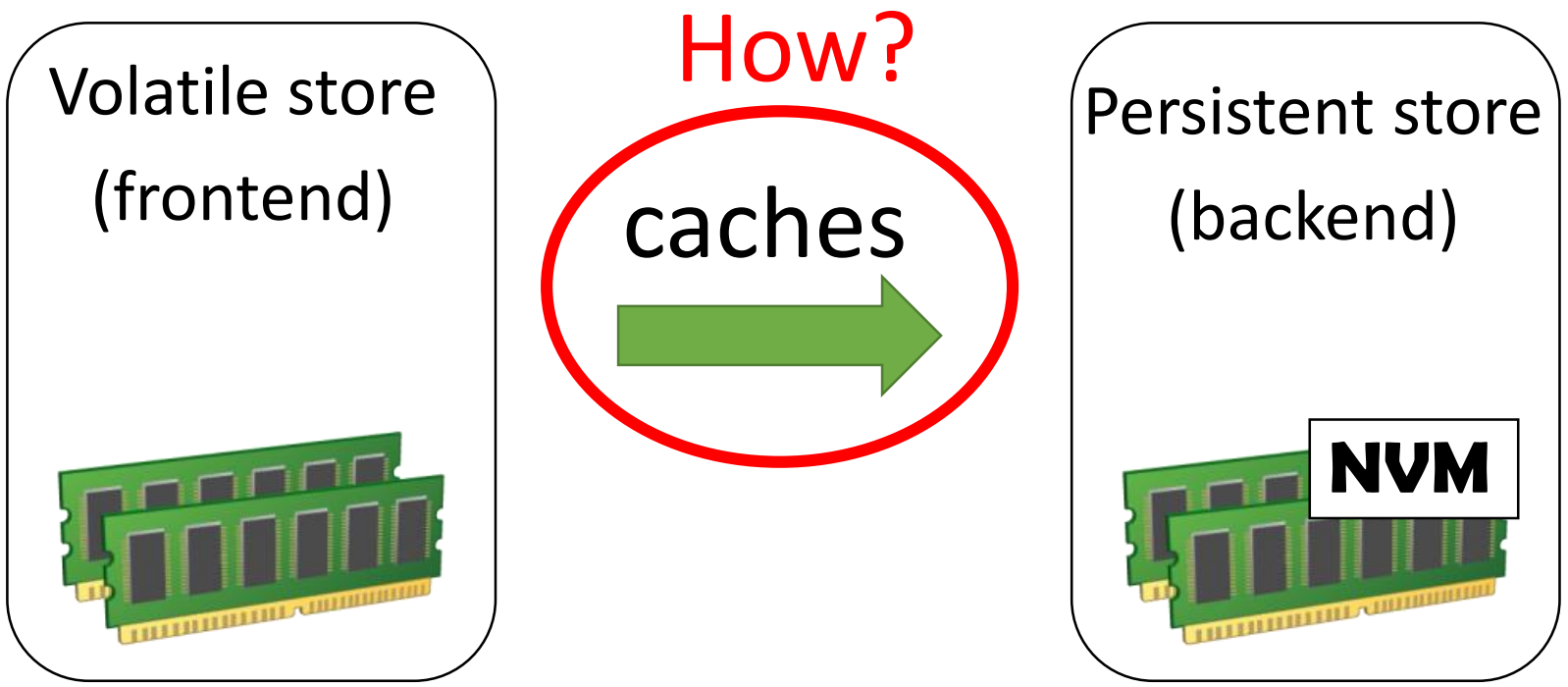


caches

Persistent store
(backend)



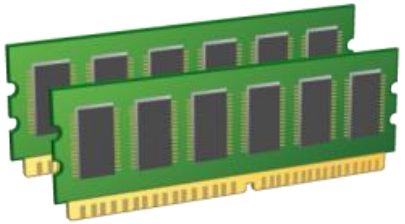
Caching


Volatile store
(frontend)
How?
caches
Persistent store
(backend)
NVM

Reading: Standard

Volatile store

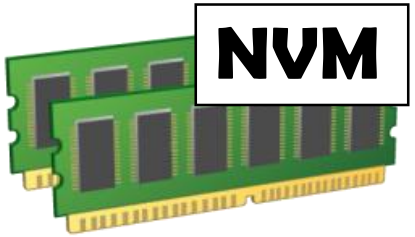
(frontend)



Lookup on cache miss

Persistent store

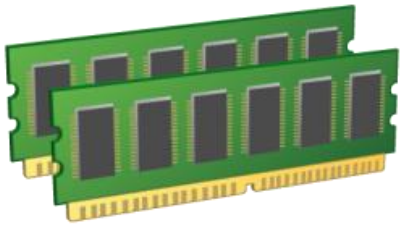
(backend)



Writing: Logs

Volatile store

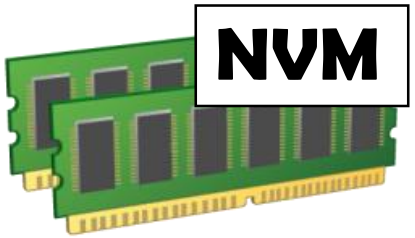
(frontend)



1. Persistently log op.
2. Update frontend
3. Return to client
4. Update backend in background

Persistent store

(backend)

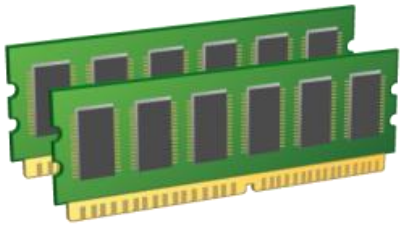


Wr How? ogs

1. Persistently log op.
2. Update frontend
3. Return to client
4. Update backend in background

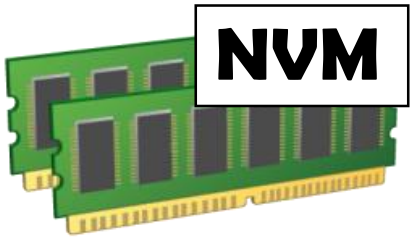
Volatile store

(frontend)



Persistent store

(backend)

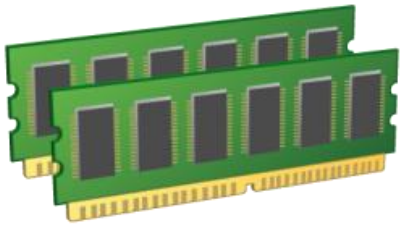


Wr How? ogs

1. Persistently log op.
2. Update frontend
3. Return to client
4. Update backend in background

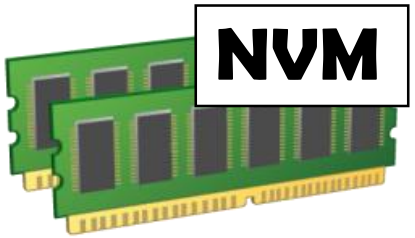
Volatile store

(frontend)



Persistent store

(backend)



Challenges: persistence, low latency, scalability, consistency

Cross-Referencing Logs

- Persistence:
  - Place logs in NVM
- Low latency
  - Few NVM accesses per log append
- Scalability
  - Multiple logs (one per thread)
- Consistency
  - Cross-log references

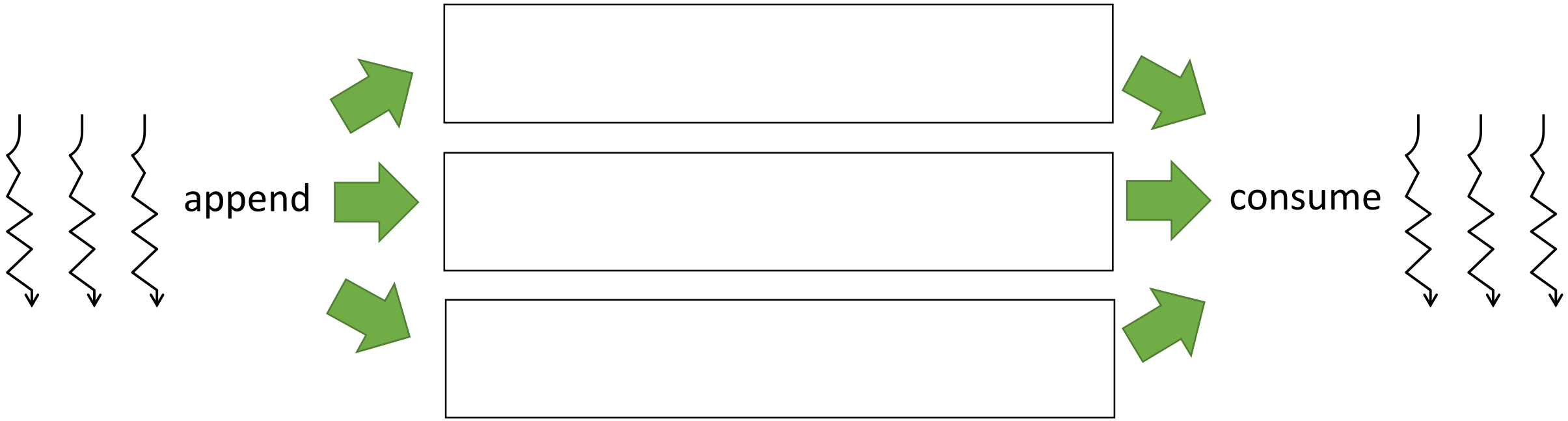
Cross-Referencing Logs

- ✔ Persistence:
  - Place logs in NVM
- ✔ Low latency
  - Few NVM accesses per log append
- ✔ Scalability
  - Multiple logs (one per thread)
- ✔ Consistency
  - Cross-log references

Cross-Referencing Logs

Log entry:

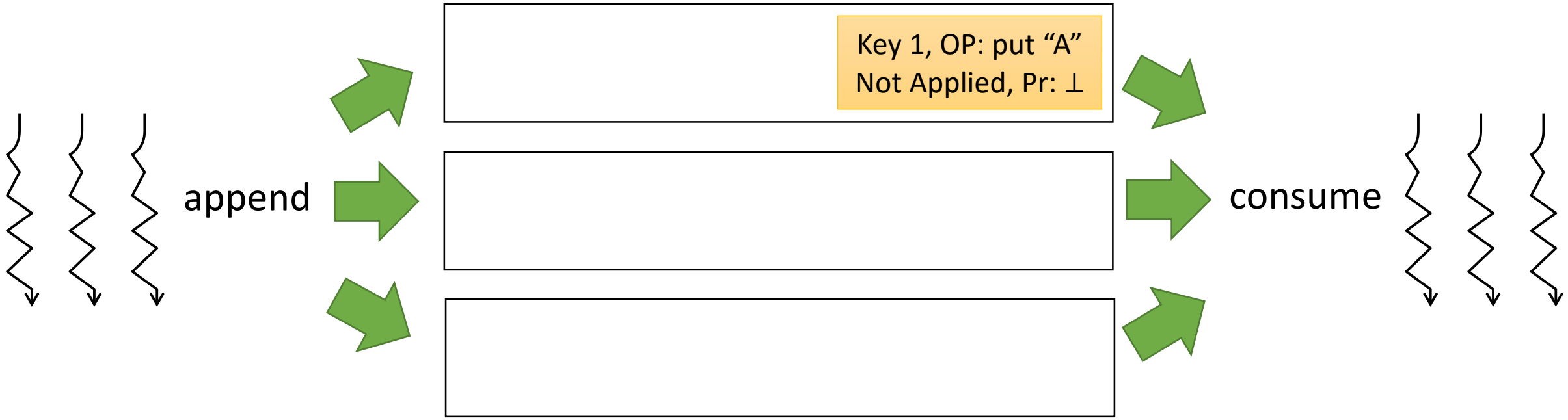
Key	OP	Applied	Prev



Cross-Referencing Logs

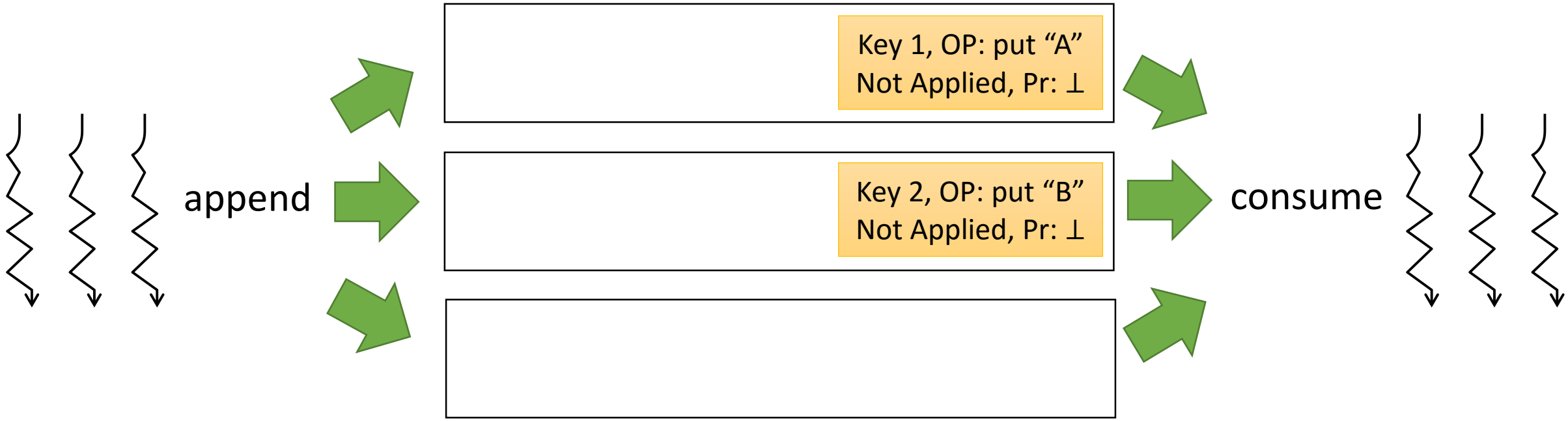
Log entry:

Key	OP	Applied	Prev



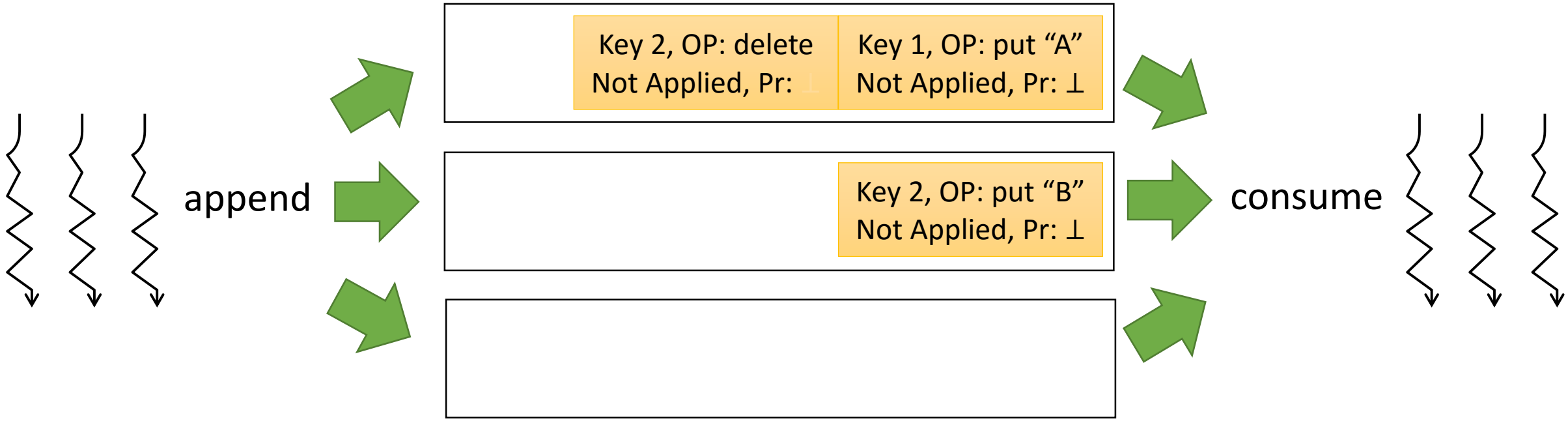
Cross-Referencing Logs

Log entry:



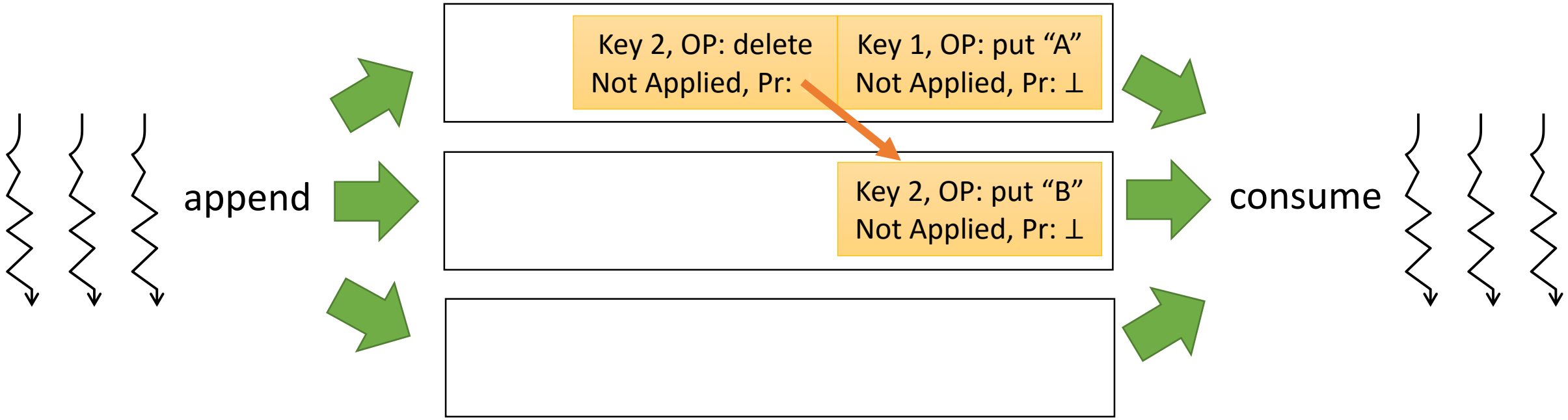
Cross-Referencing Logs

Log entry:



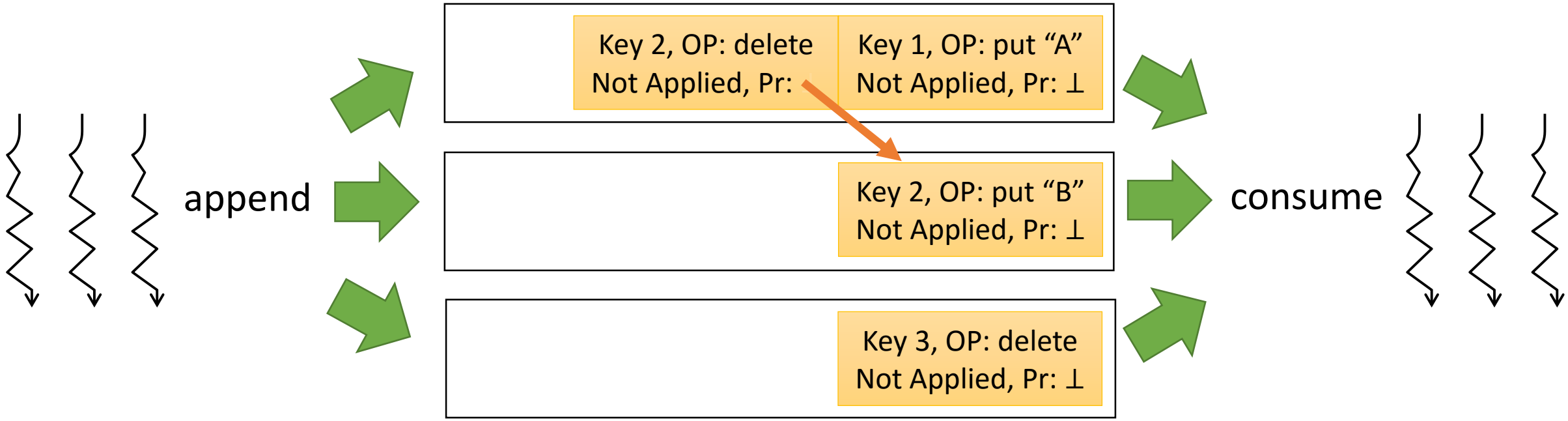
Cross-Referencing Logs

Log entry:



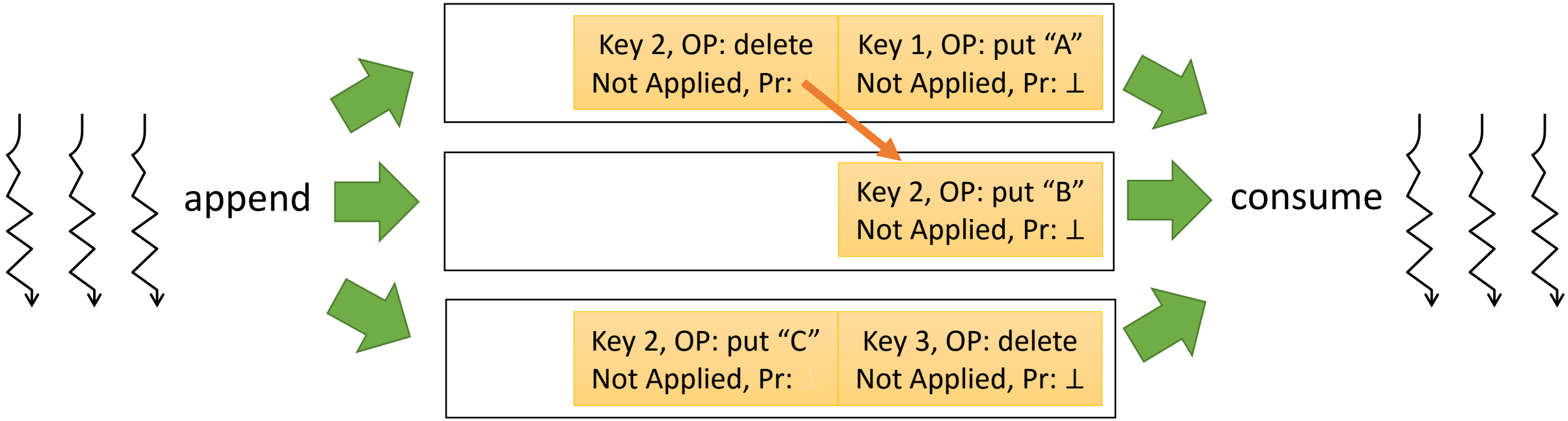
Cross-Referencing Logs

Log entry:



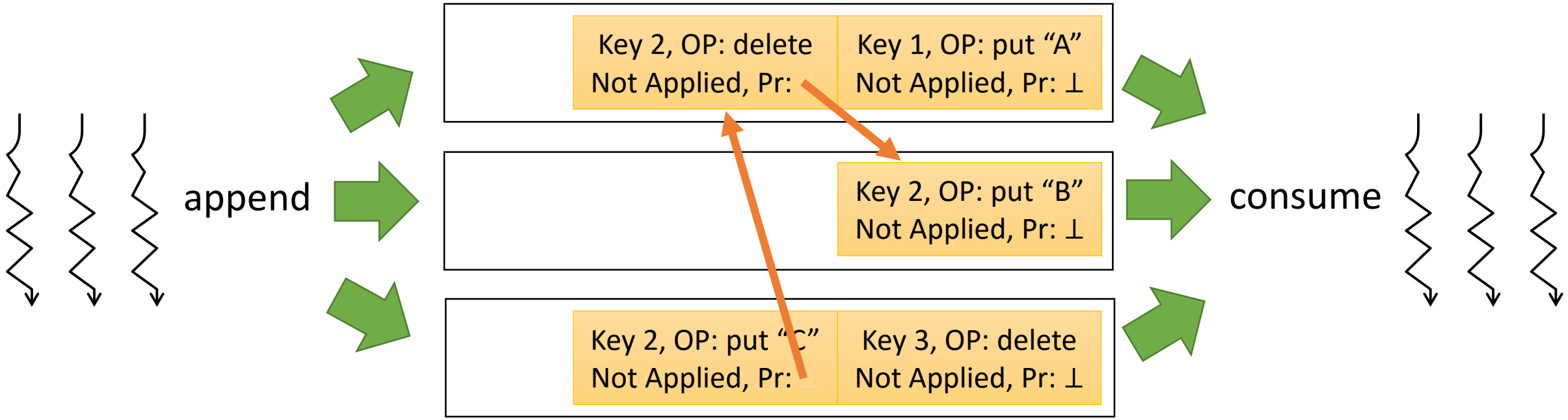
Cross-Referencing Logs

Log entry:



Cross-Referencing Logs

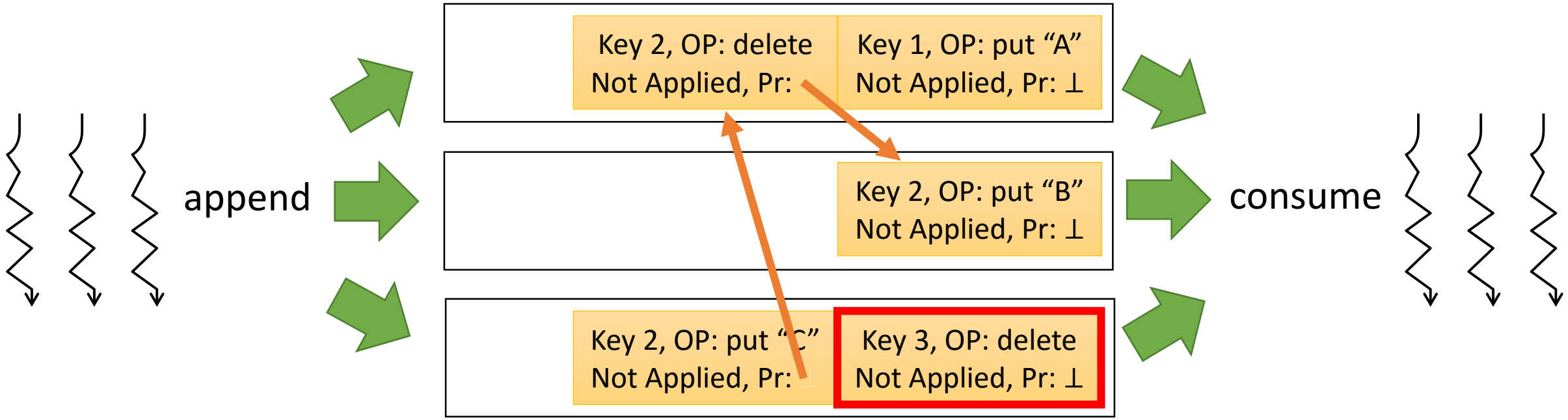
Log entry:



Cross-Referencing Logs

Log entry:

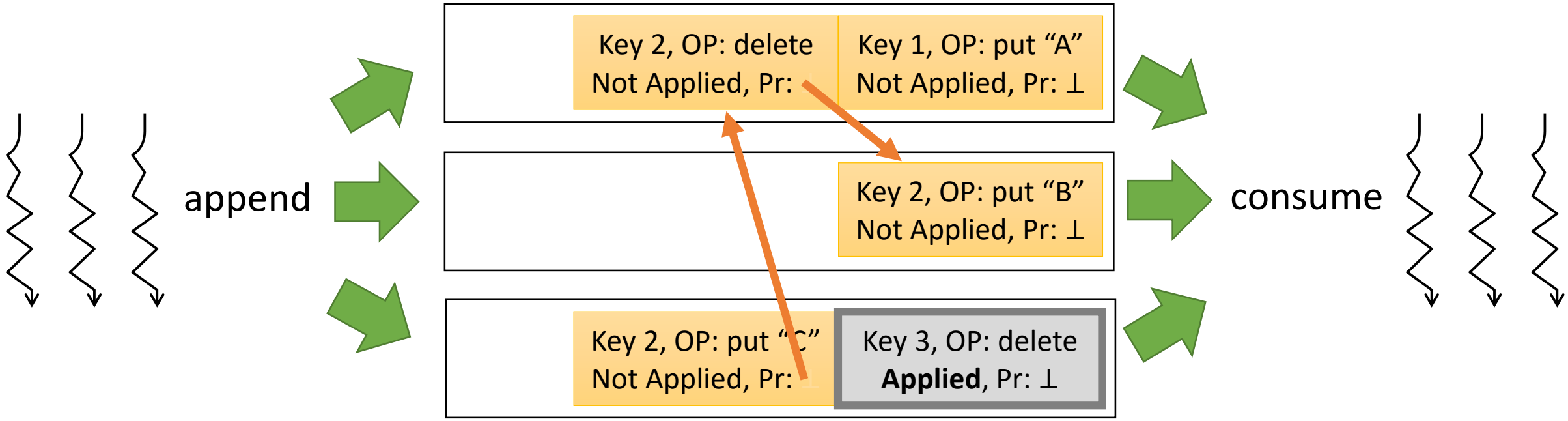
Key	OP	Applied	Prev



Cross-Referencing Logs

Log entry:

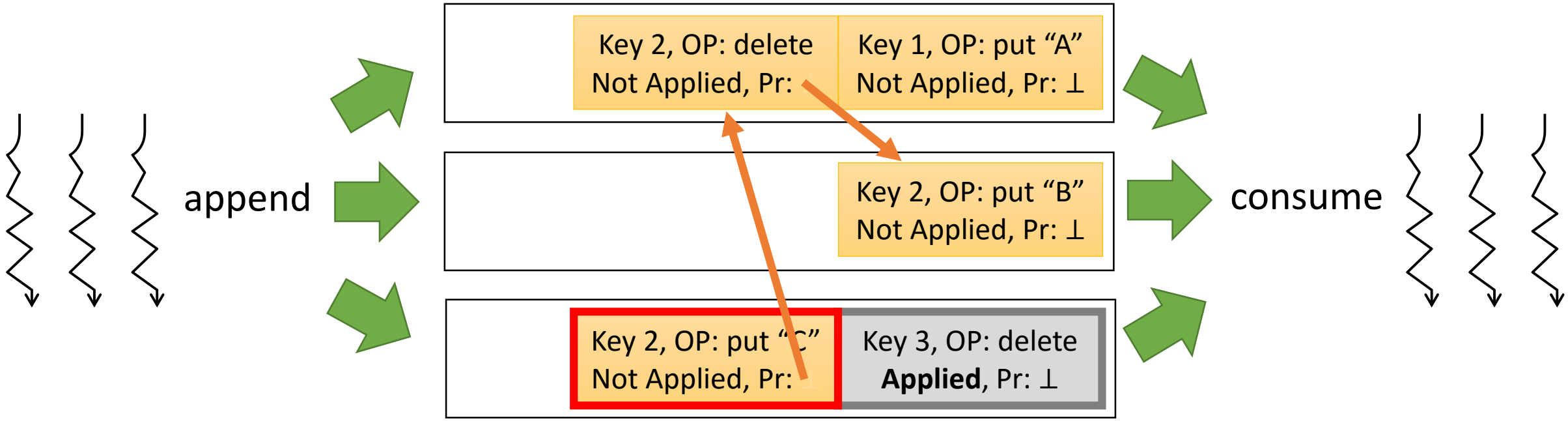
Key	OP	Applied	Prev



Cross-Referencing Logs

Log entry:

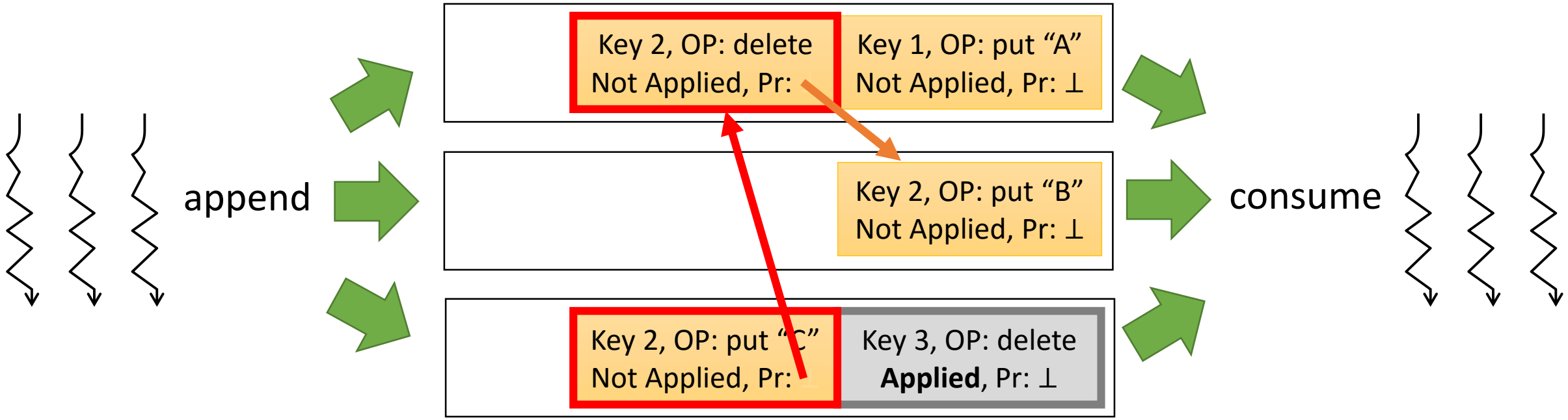
Key	OP	Applied	Prev



Cross-Referencing Logs

Log entry:

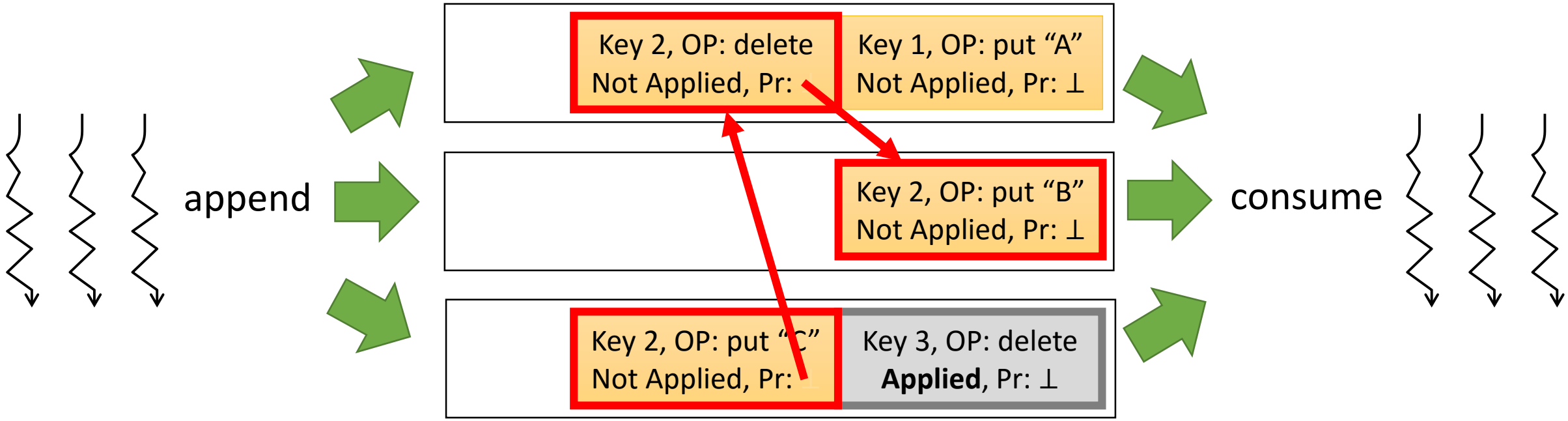
Key	OP	Applied	Prev



Cross-Referencing Logs

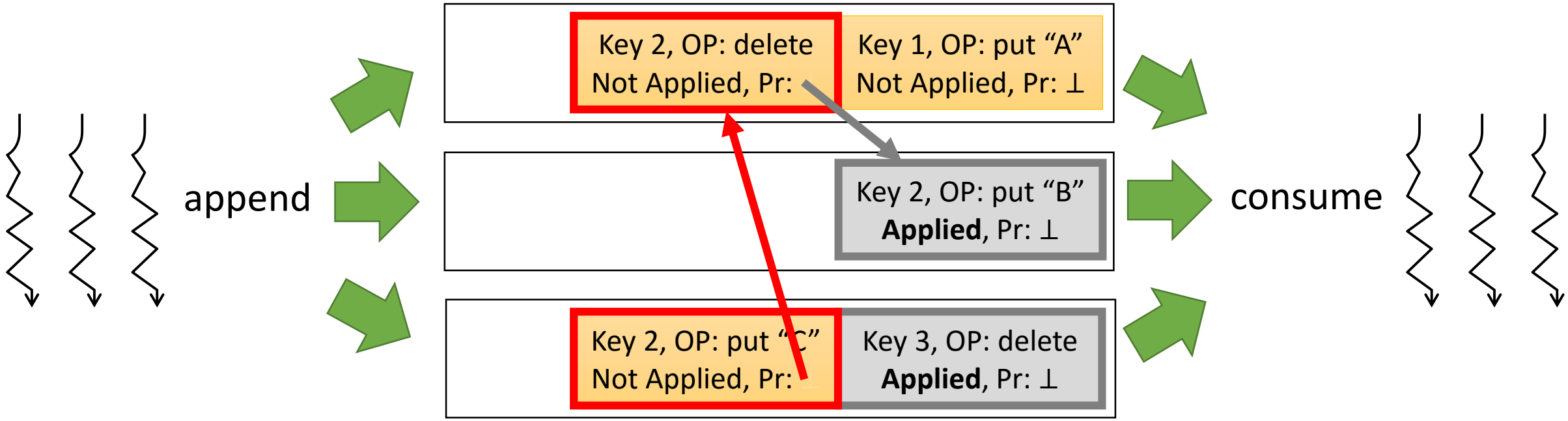
Log entry:

Key	OP	Applied	Prev



Cross-Referencing Logs

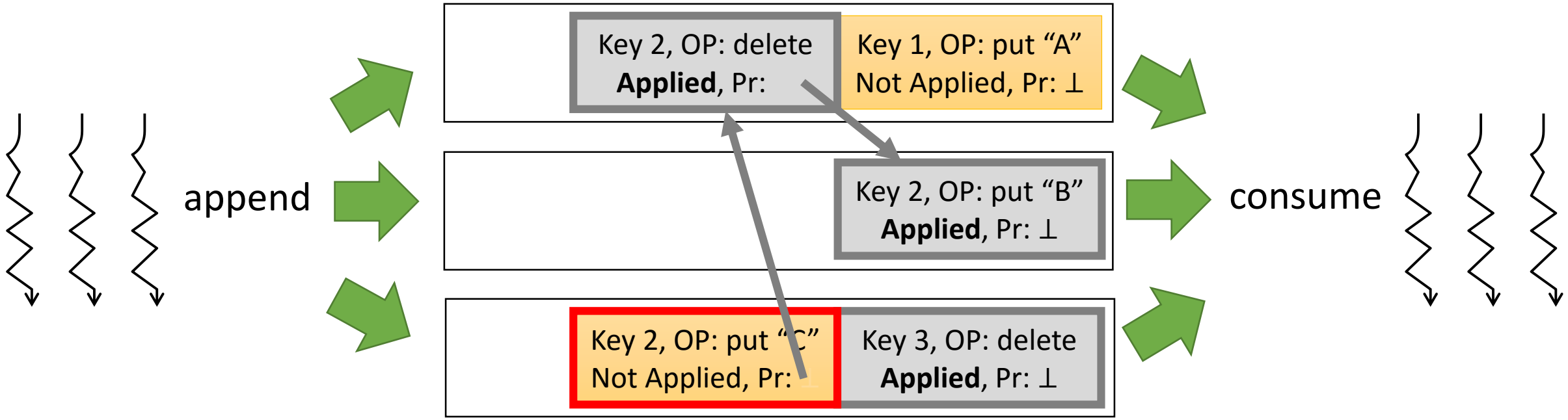
Log entry:



Cross-Referencing Logs

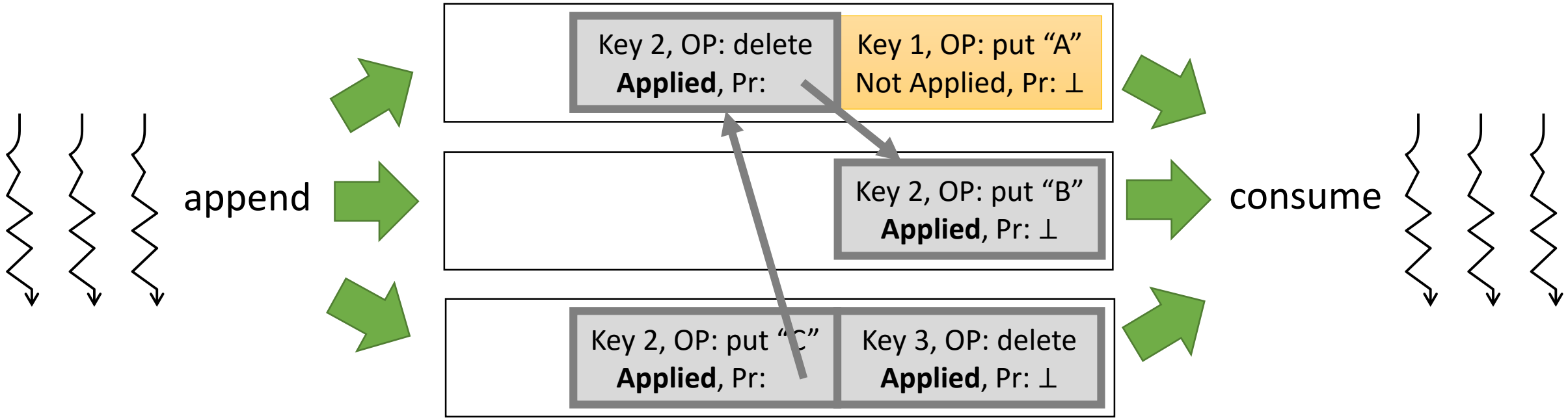
Log entry:

Key	OP	Applied	Prev



Cross-Referencing Logs

Log entry:



Outline

- Cross-Referencing Logs:
  Getting the best of DRAM and NVM

- **Bullet:**
  Fast and persistent key-value store

Outline

- Cross-Referencing Logs:

  Getting the best of DRAM and NVM

- **Bullet:**

  Fast and persistent key-value store

  - **Basic design**
  - **Optimizations**

Outline

- Cross-Referencing Logs:

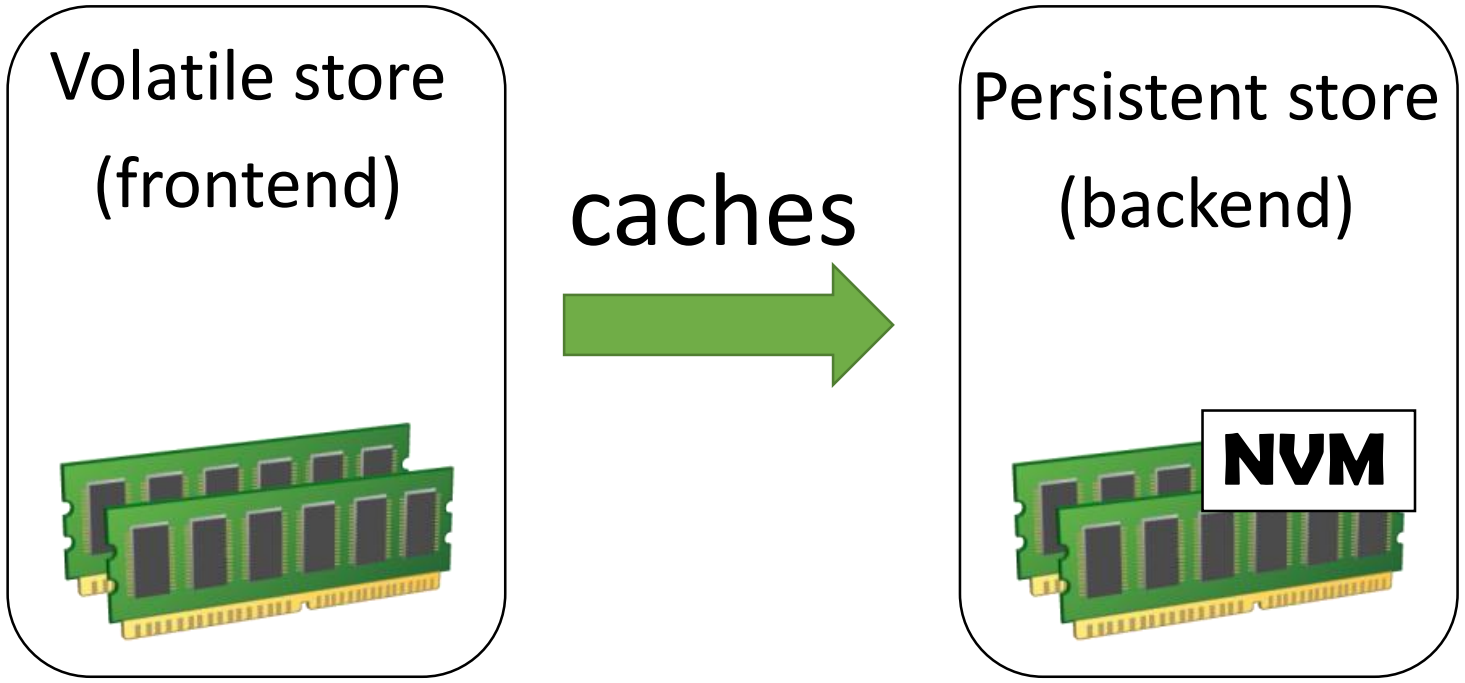
  Getting the best of DRAM and NVM

- Bullet:

  Fast and persistent key-value store

  - **Basic design**
  - Optimizations

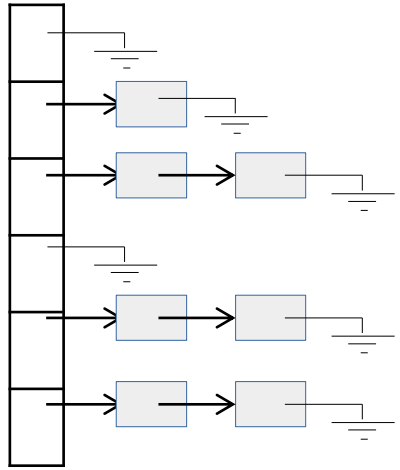
Bullet K-V Store



Bullet K-V Store

Volatile store

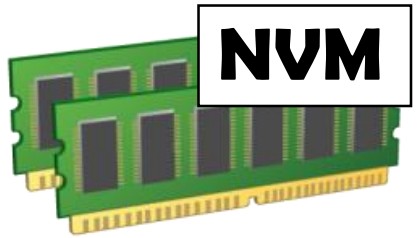
(frontend)



caches

Persistent store

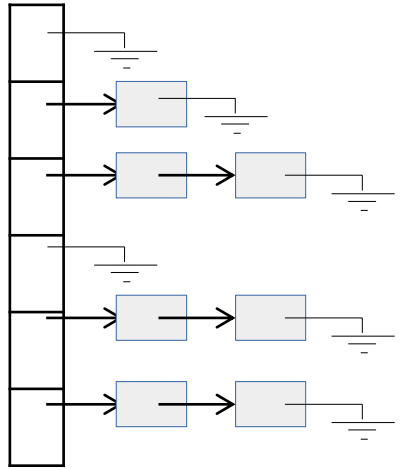
(backend)



Bullet K-V Store

Volatile store

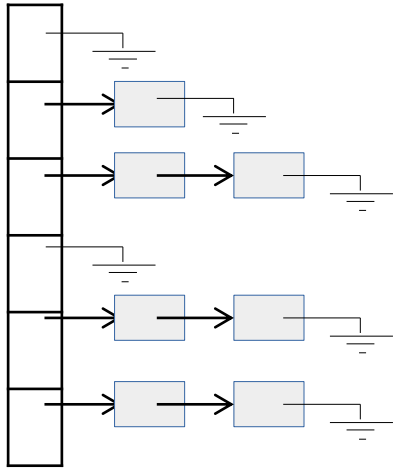
(frontend)



caches

Persistent store

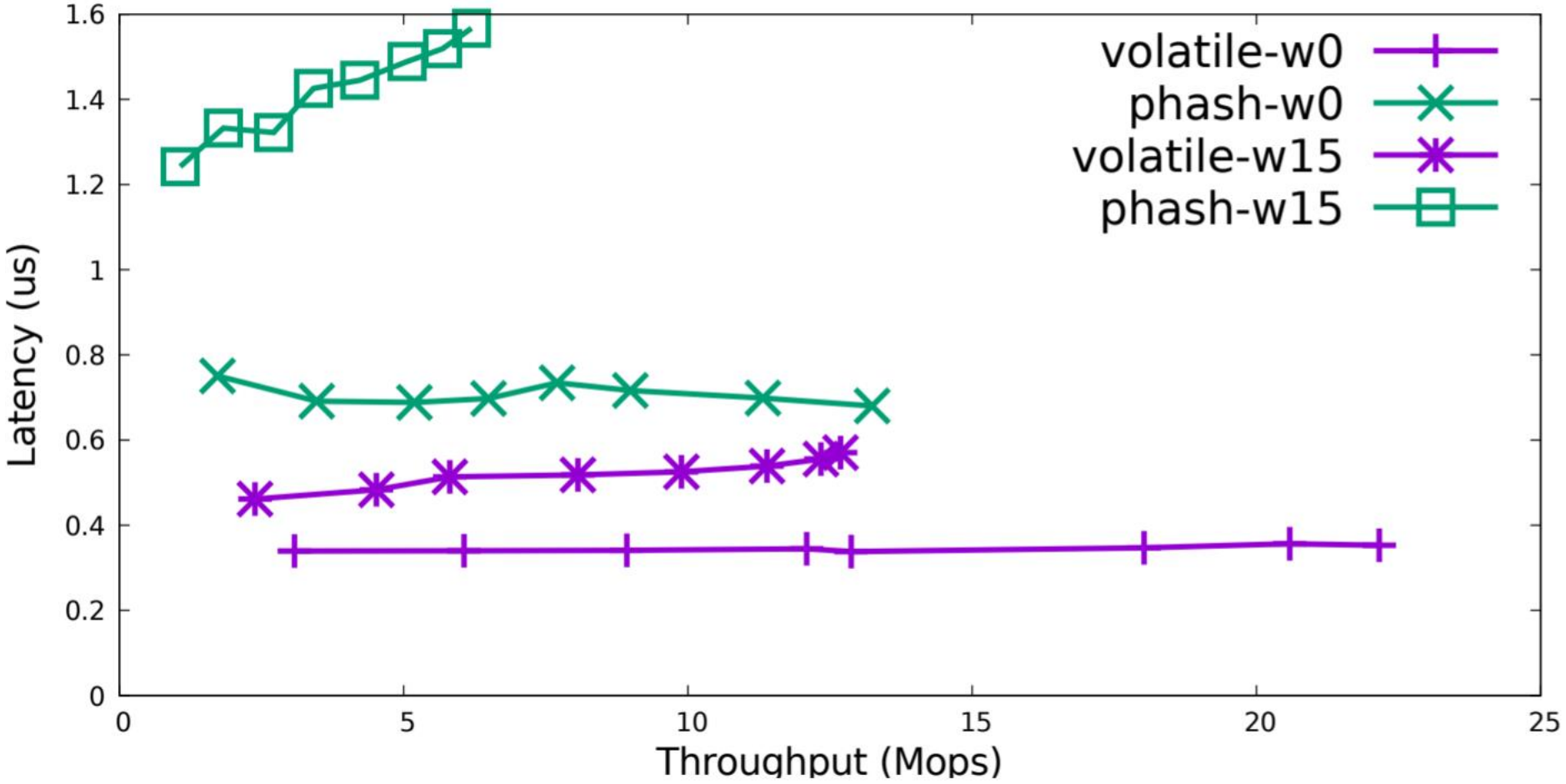
(backend)



Experimental Setup

- 16 CPU cores, 512 GB DRAM
- Intel’s NVM emulation platform
- Zipfian distribution of keys (YCSB)
- Measuring 99%ile latency
- Comparing against HiKV

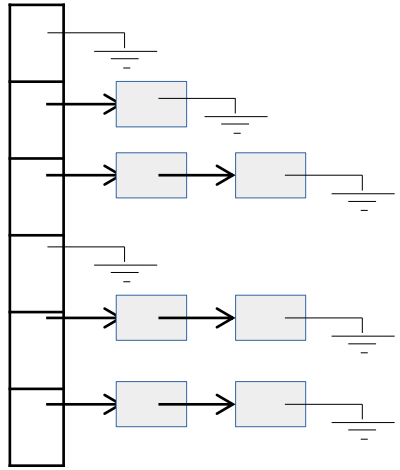
Raw Hash Table Performance



Bullet K-V Store

Volatile store

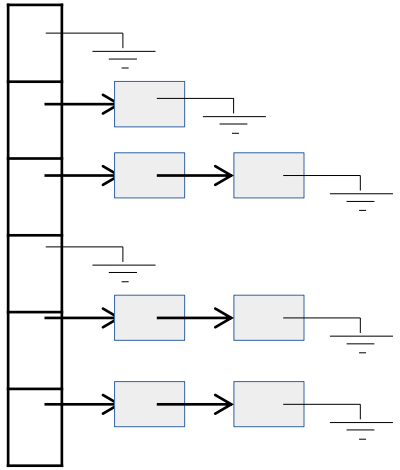
(frontend)



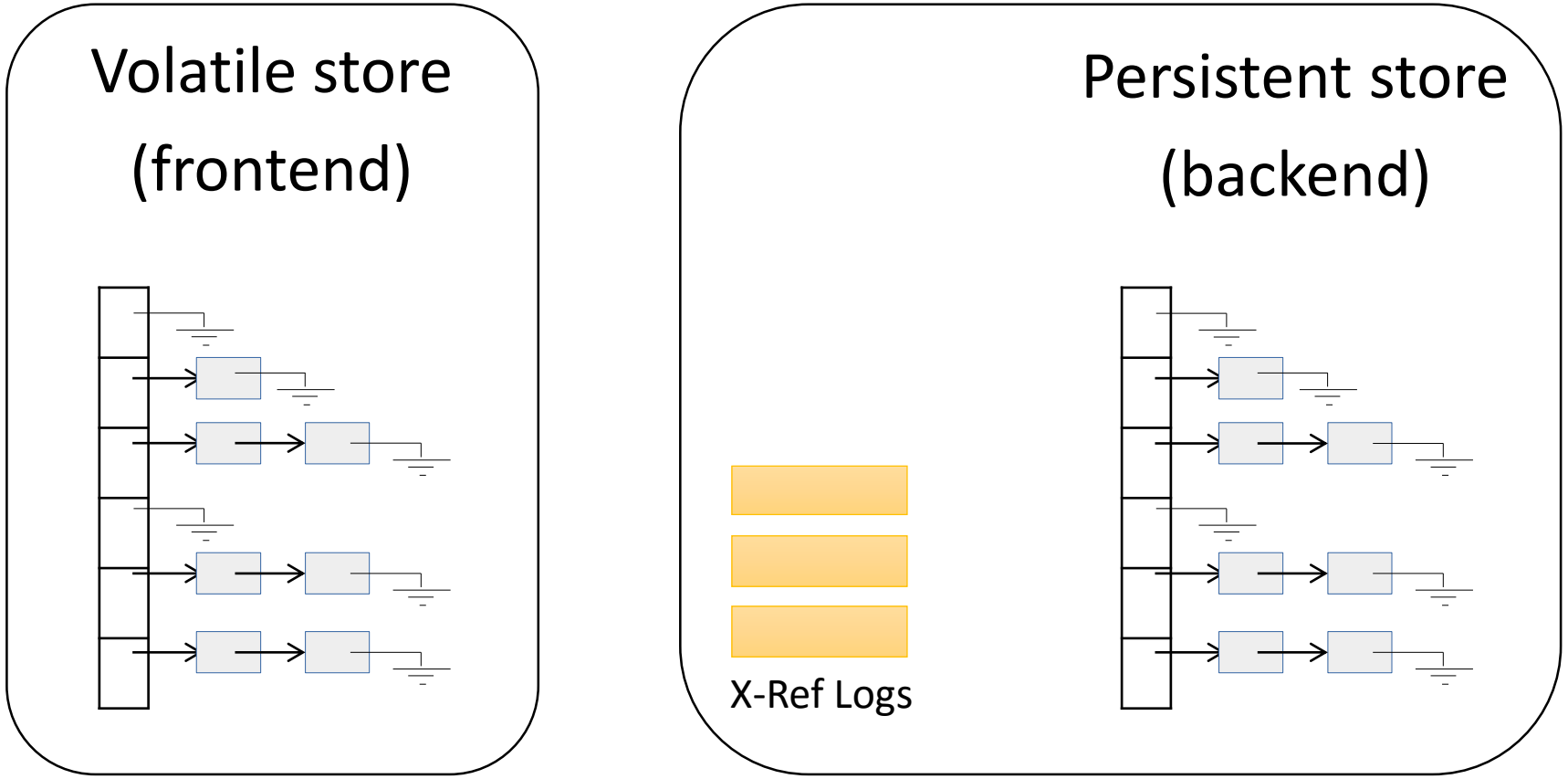
caches

Persistent store

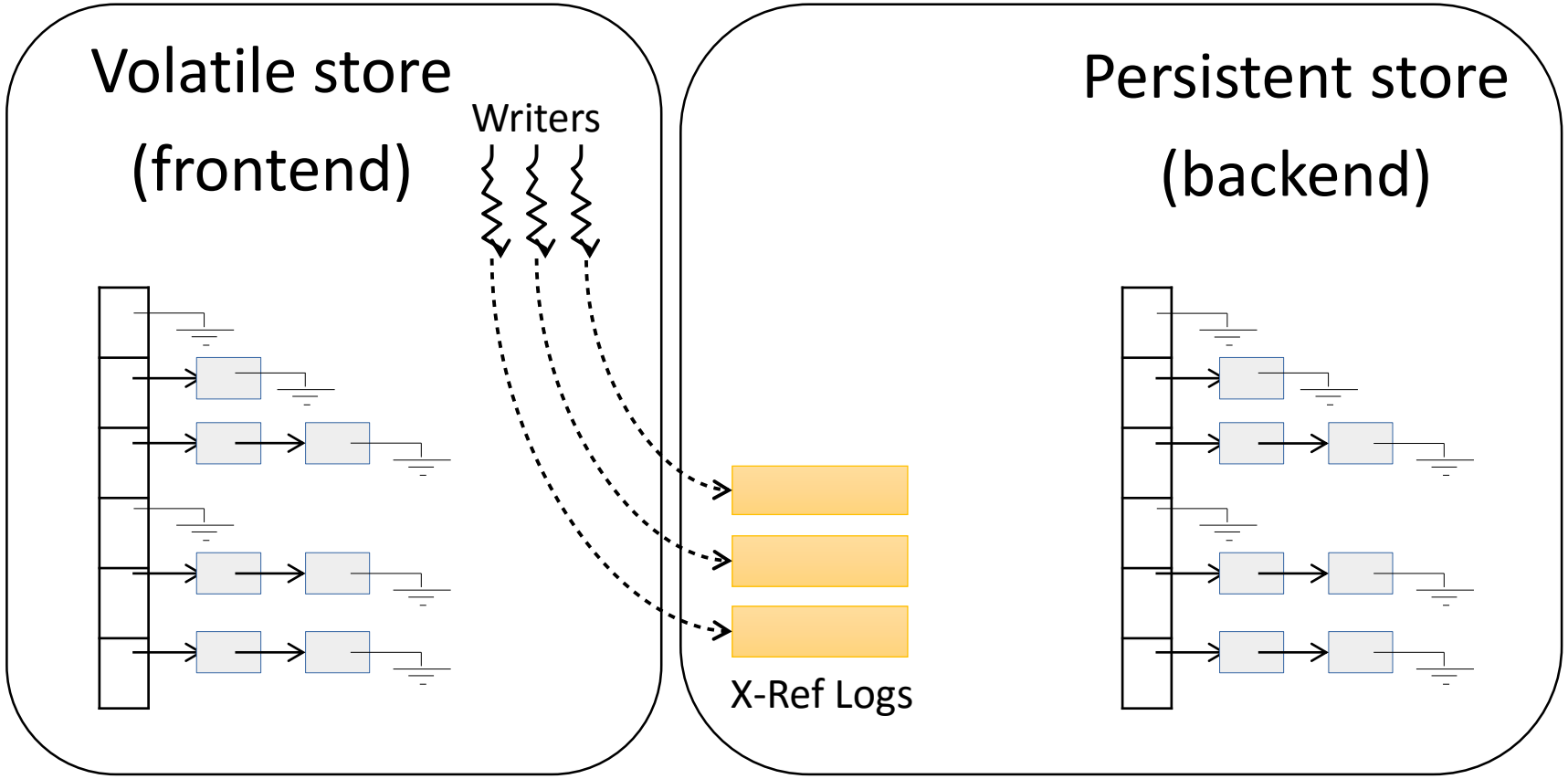
(backend)



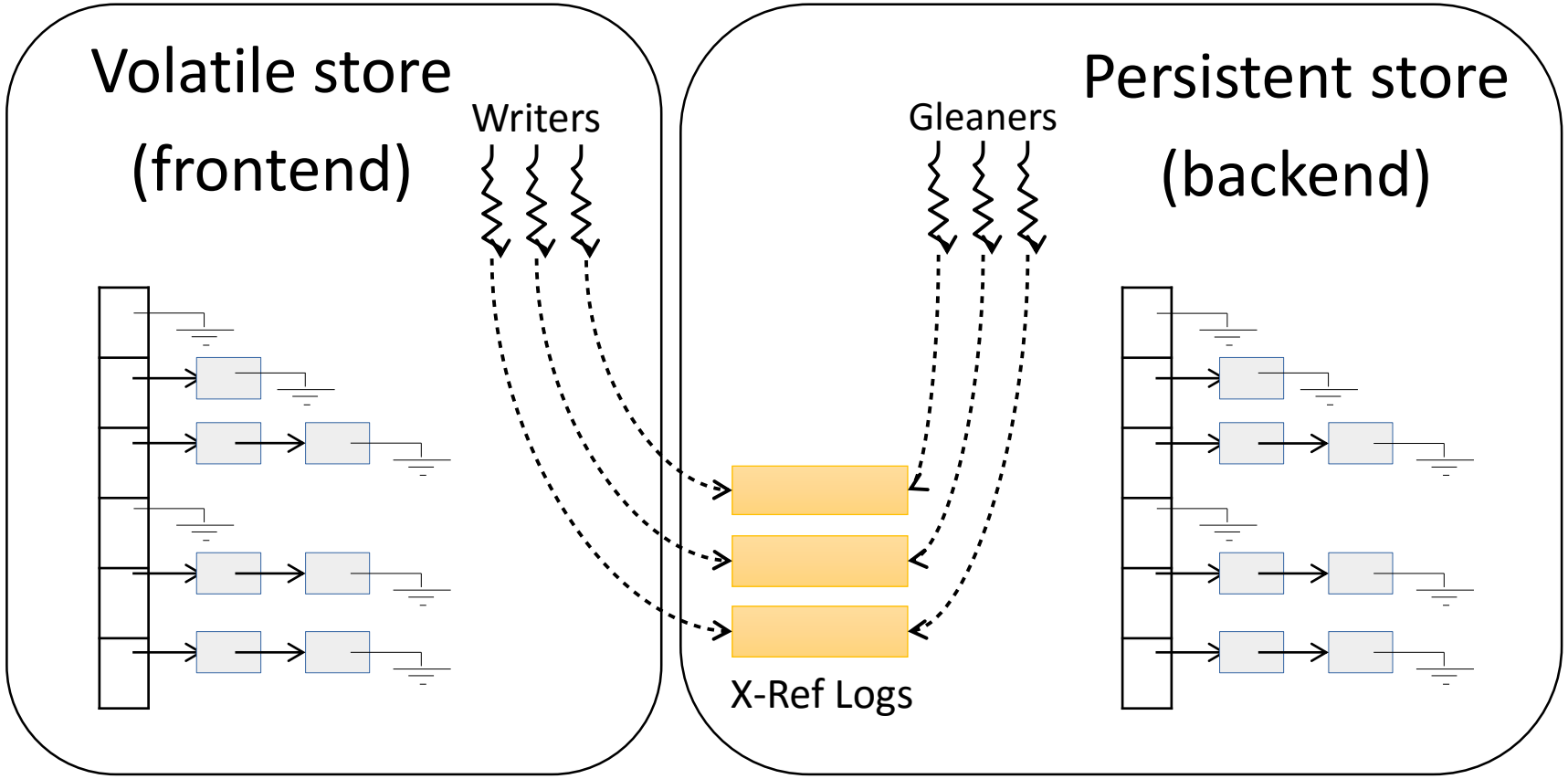
Bullet K-V Store



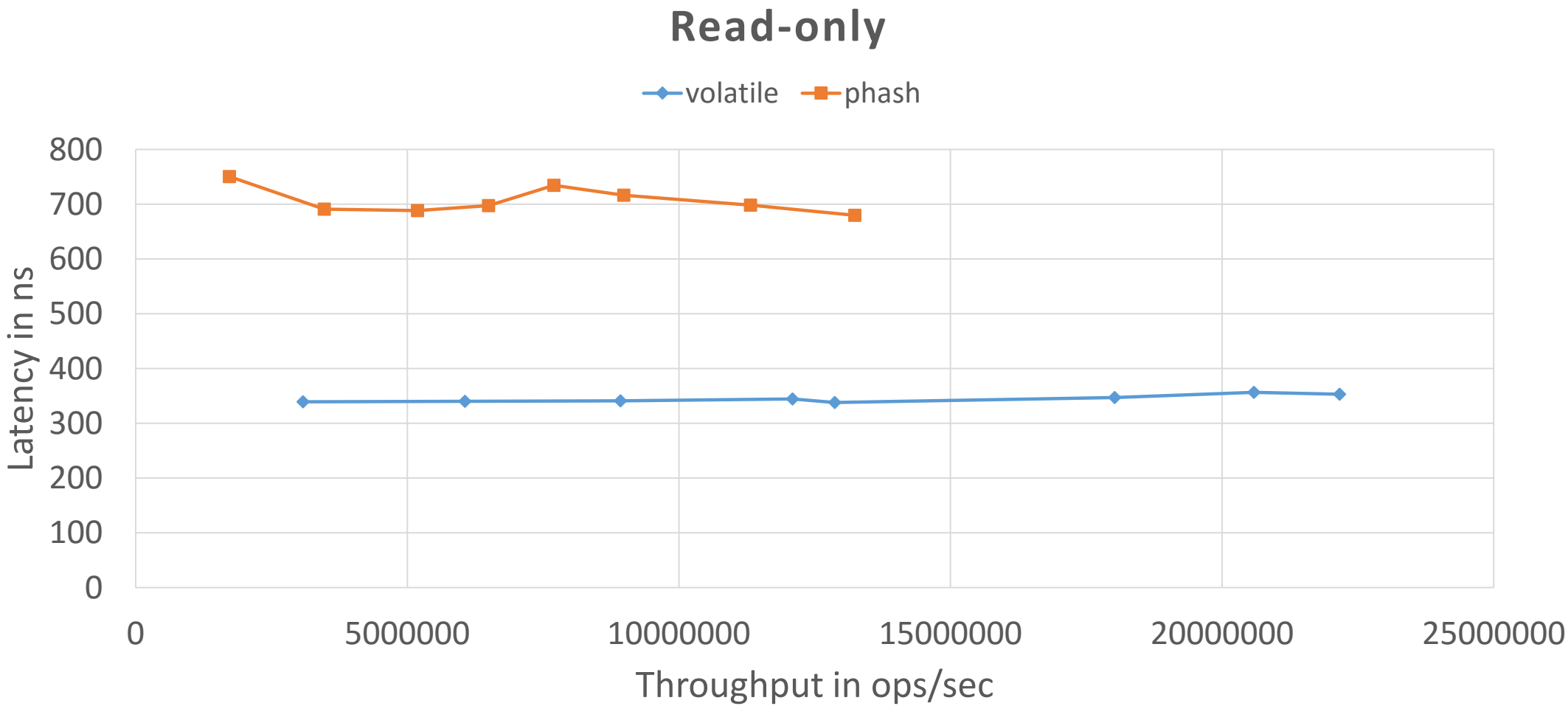
Bullet K-V Store



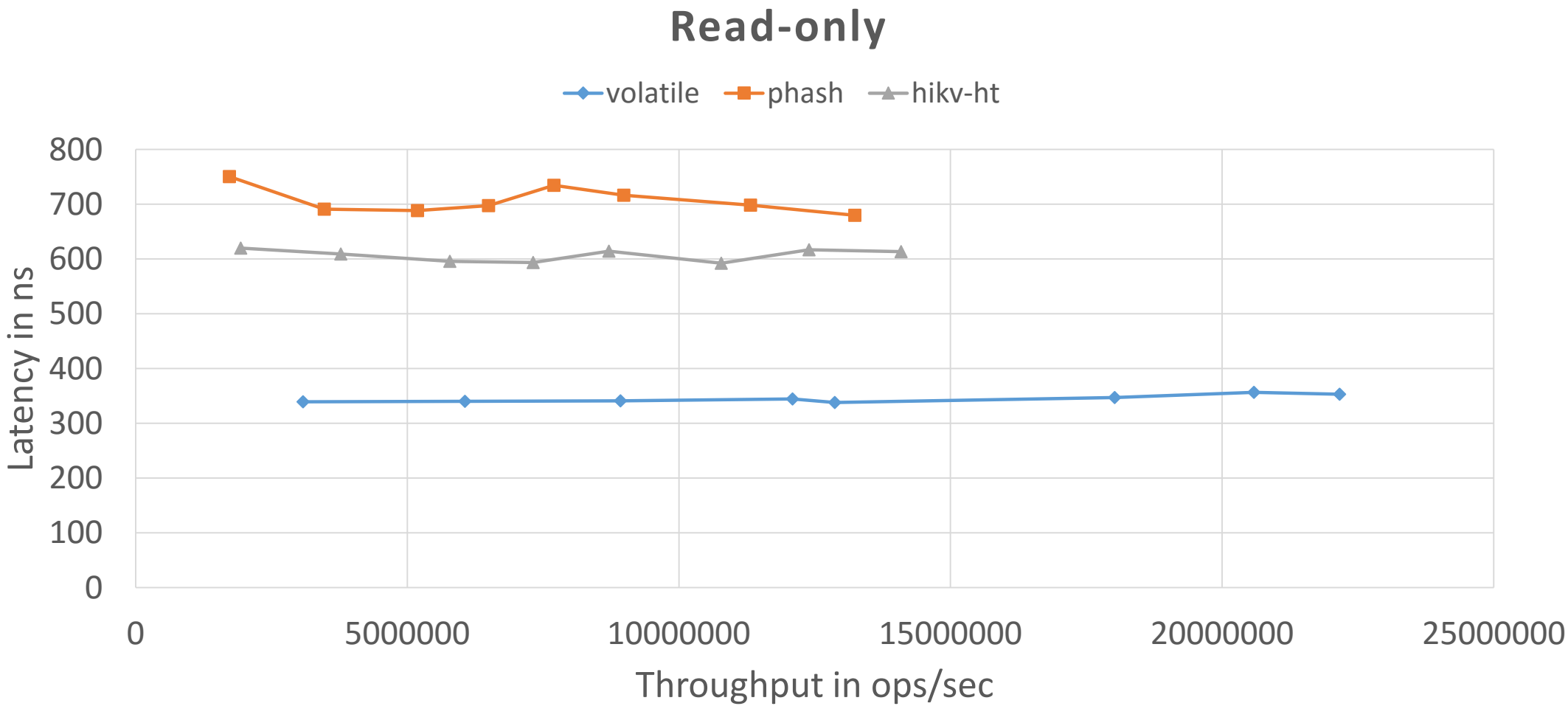
Bullet K-V Store



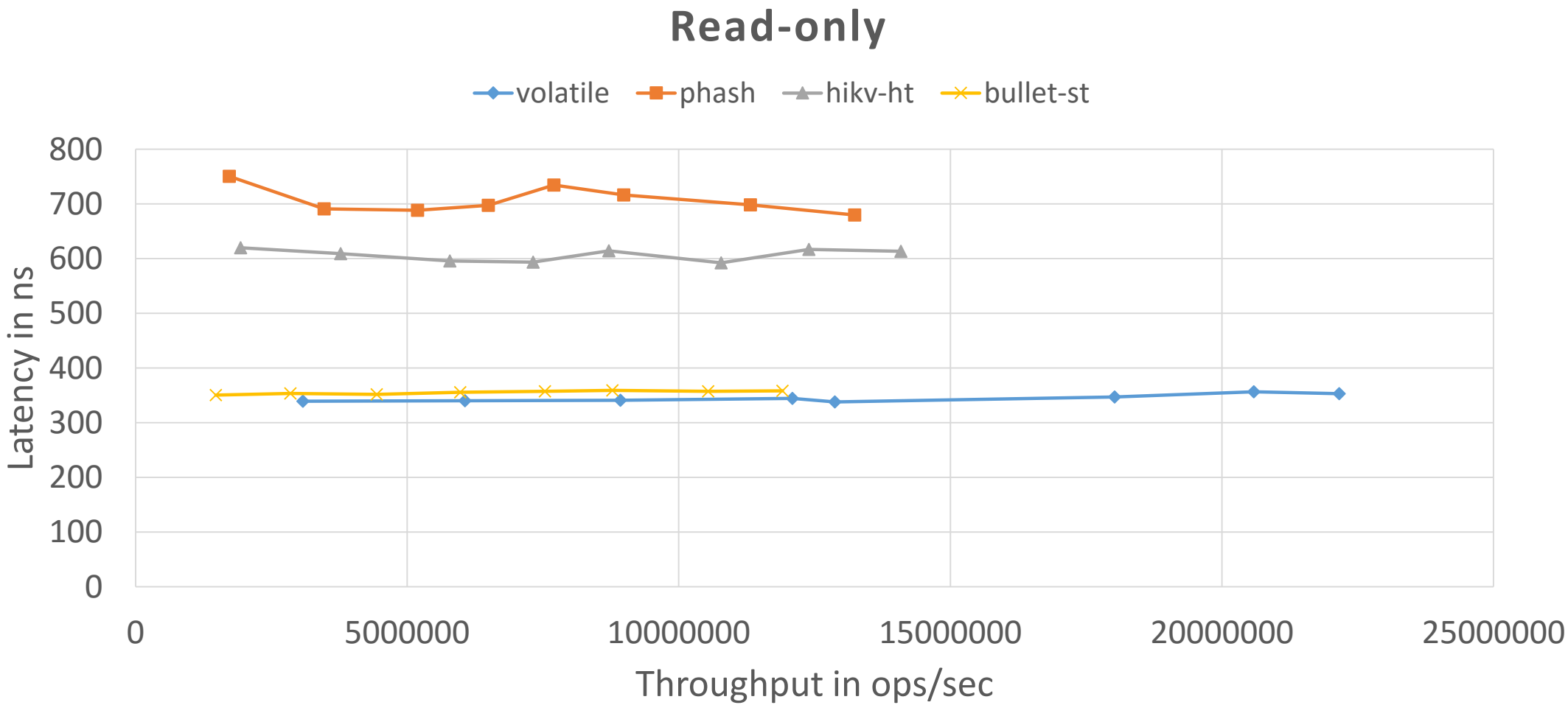
Bullet Performance



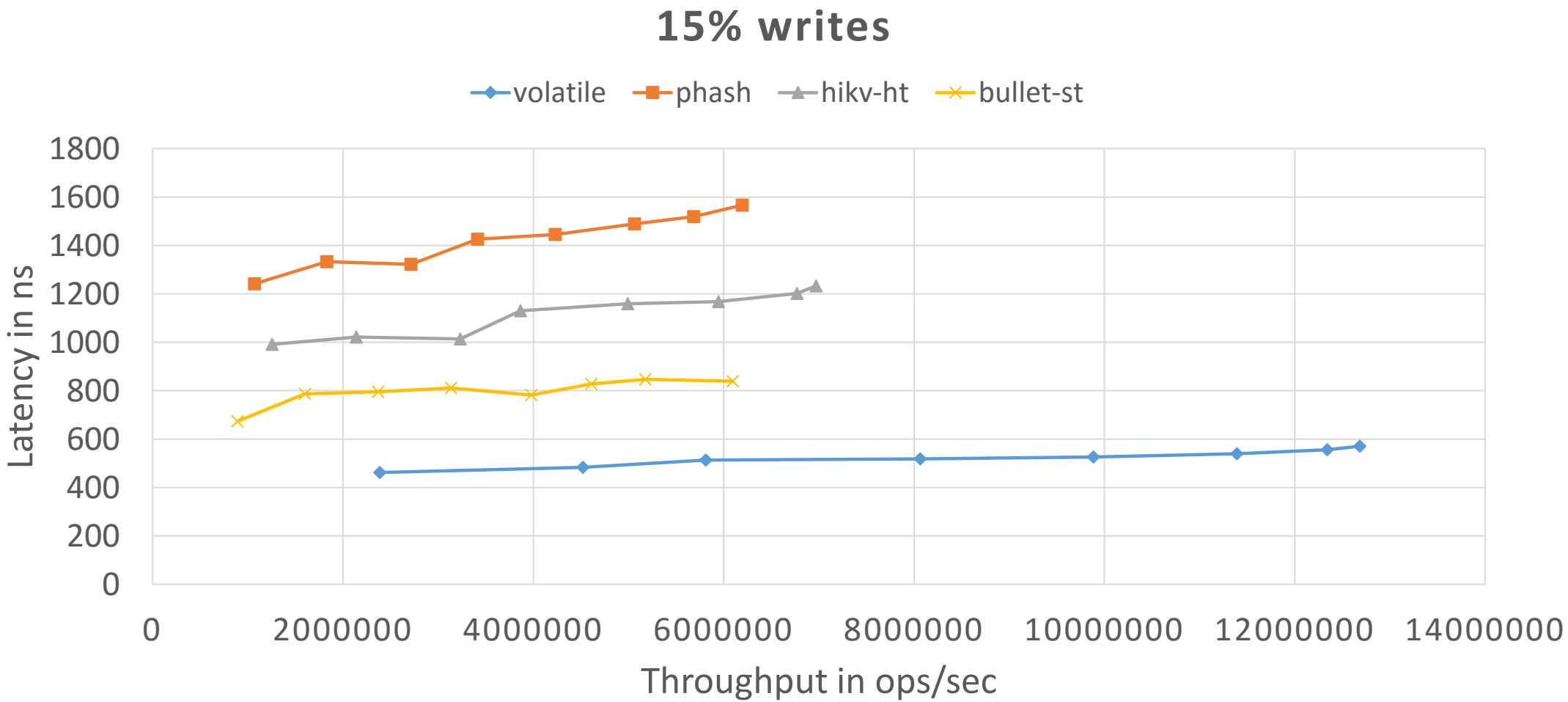
Bullet Performance



Bullet Performance



Bullet Performance



Outline

- Cross-Referencing Logs:

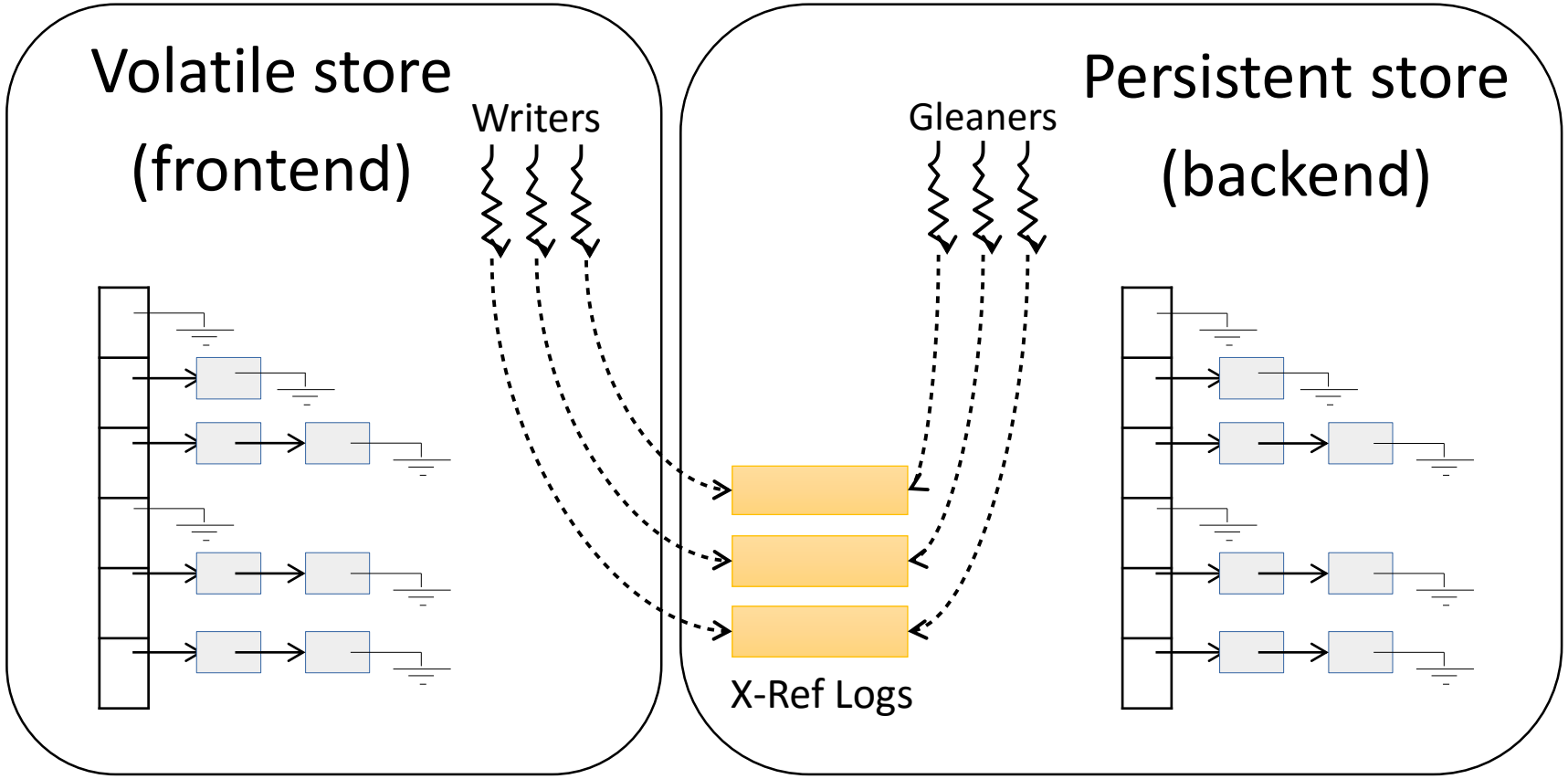
  Getting the best of DRAM and NVM

- Bullet:

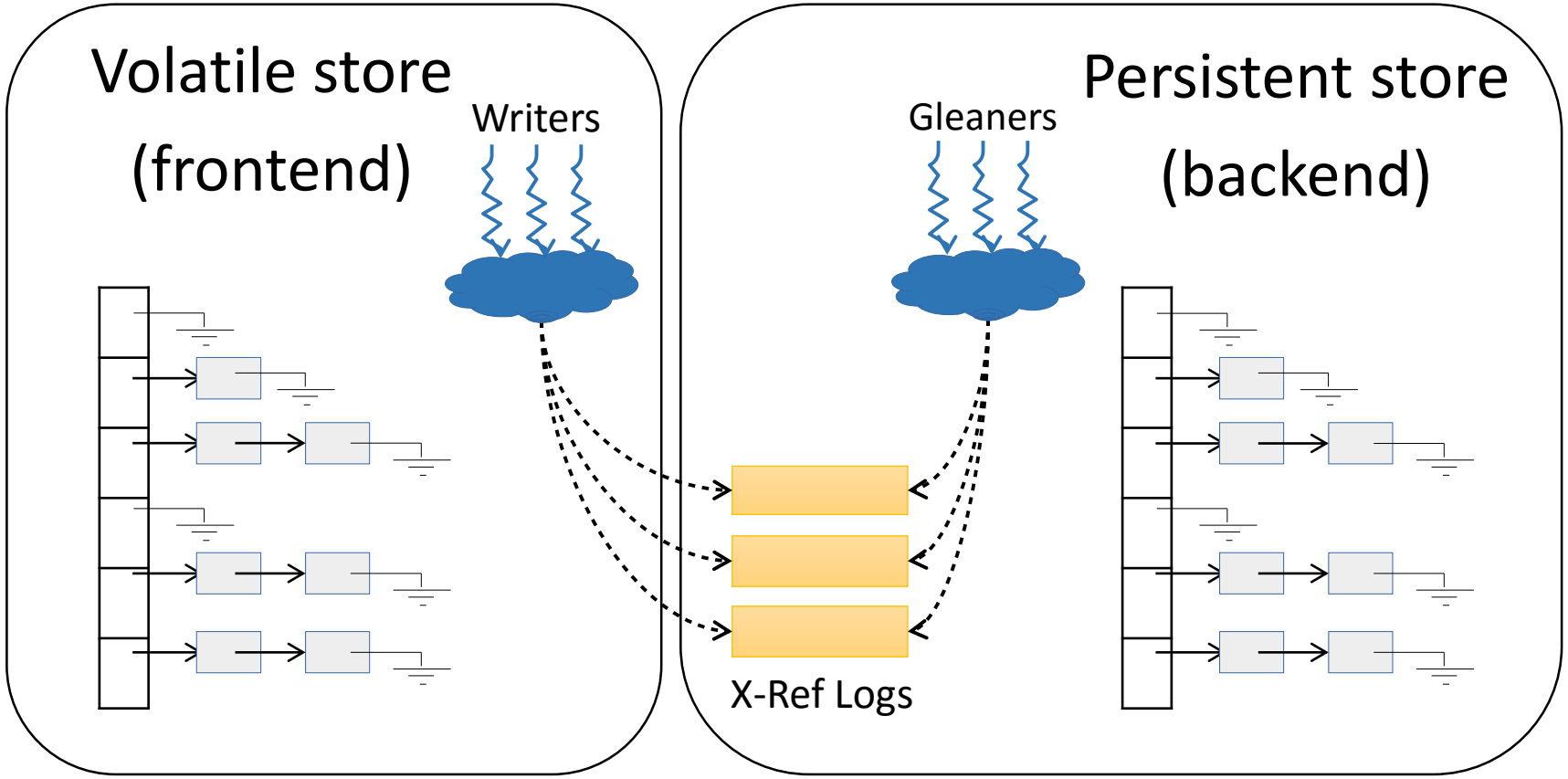
  Fast and persistent key-value store

  - Basic design
  - **Optimizations**

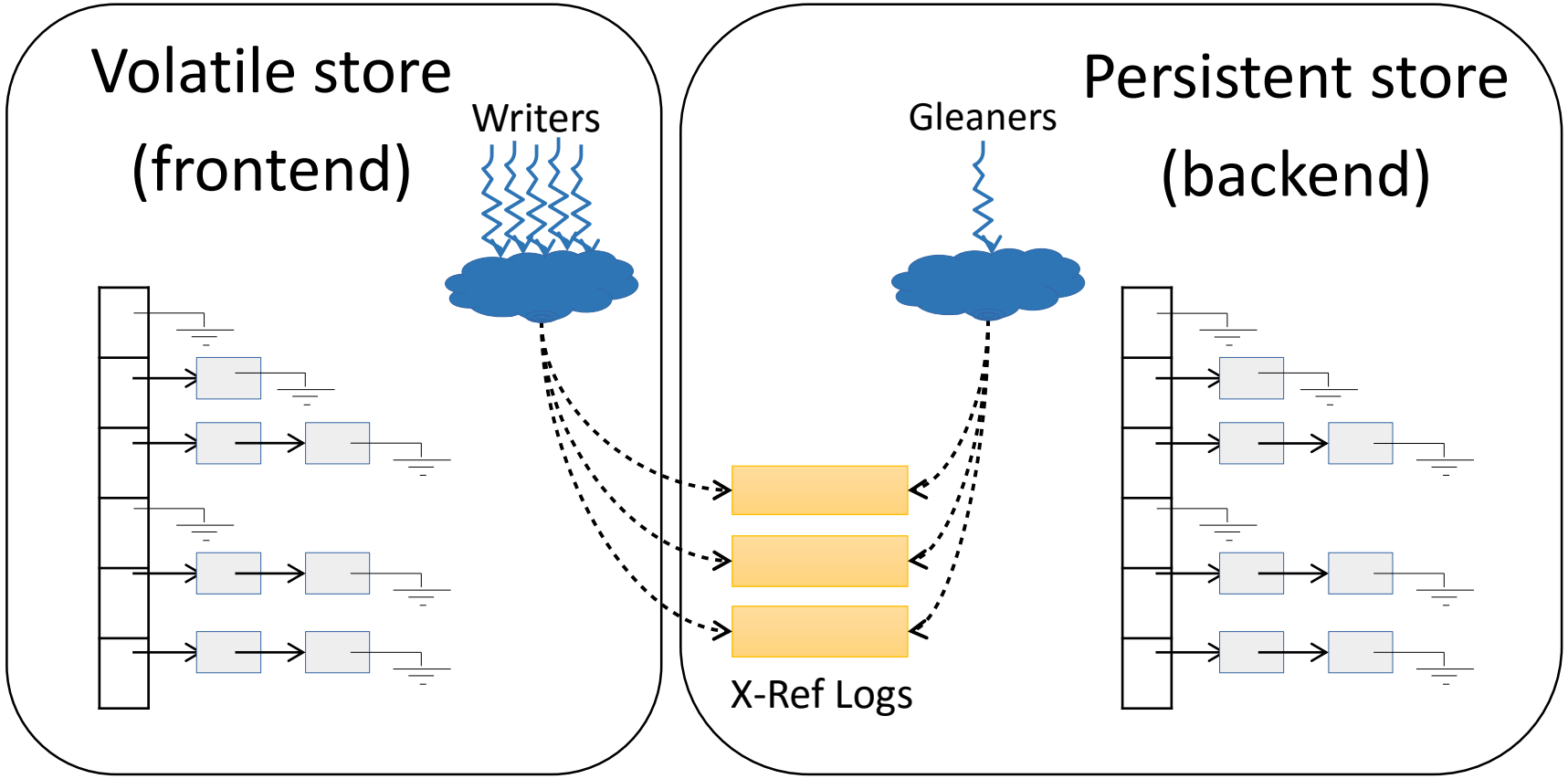
Optimization: Dynamic Worker Threads



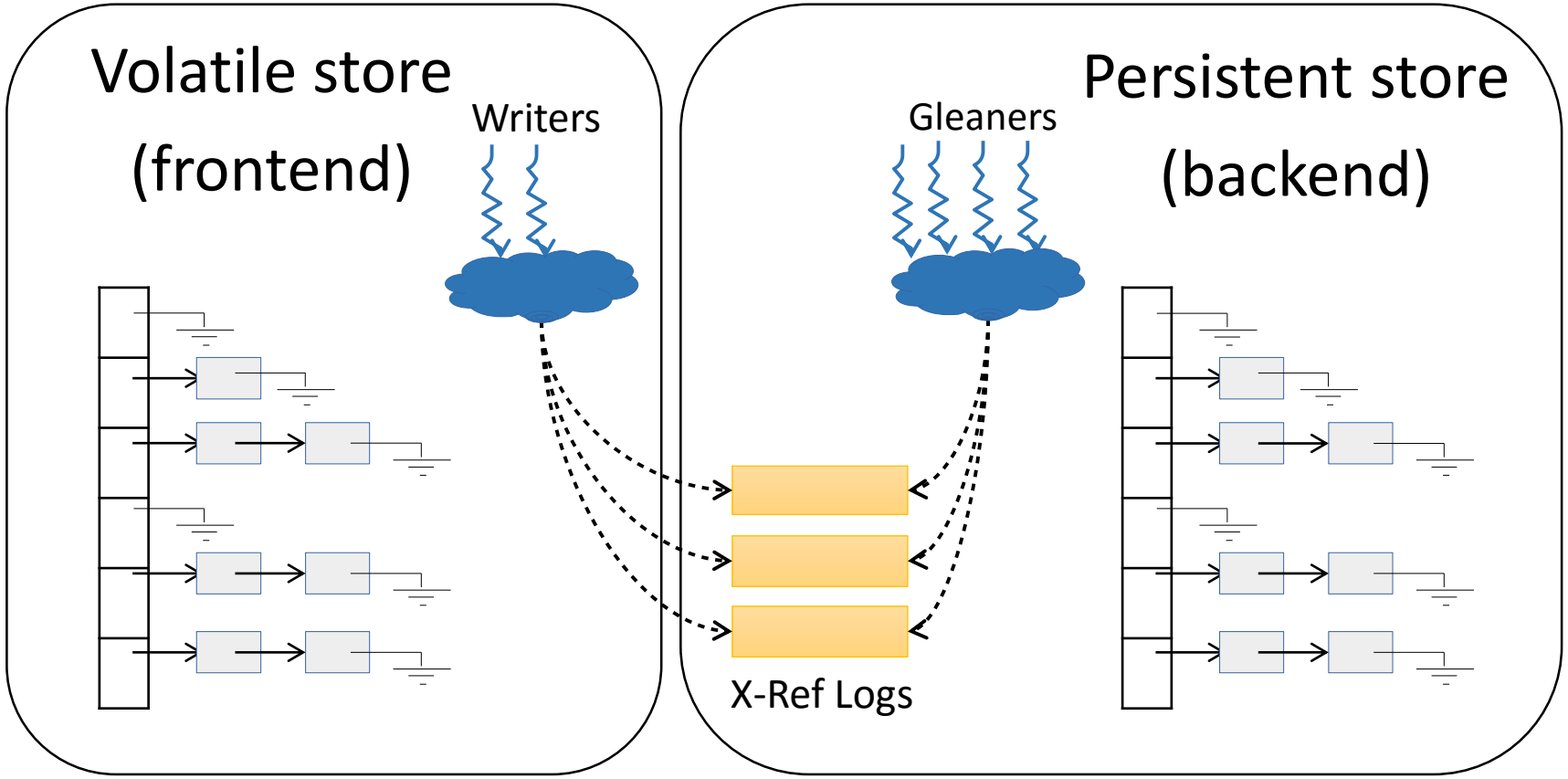
Optimization: Dynamic Worker Threads



Read-Heavy Workload

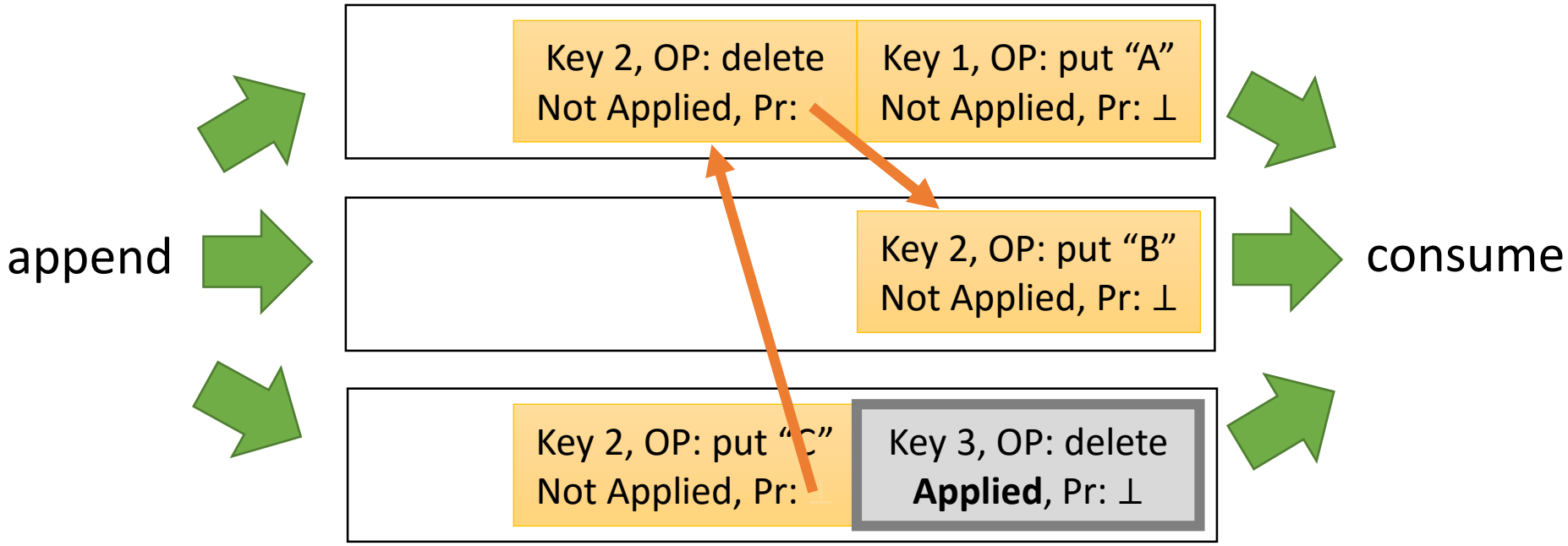


Write-Heavy Workload



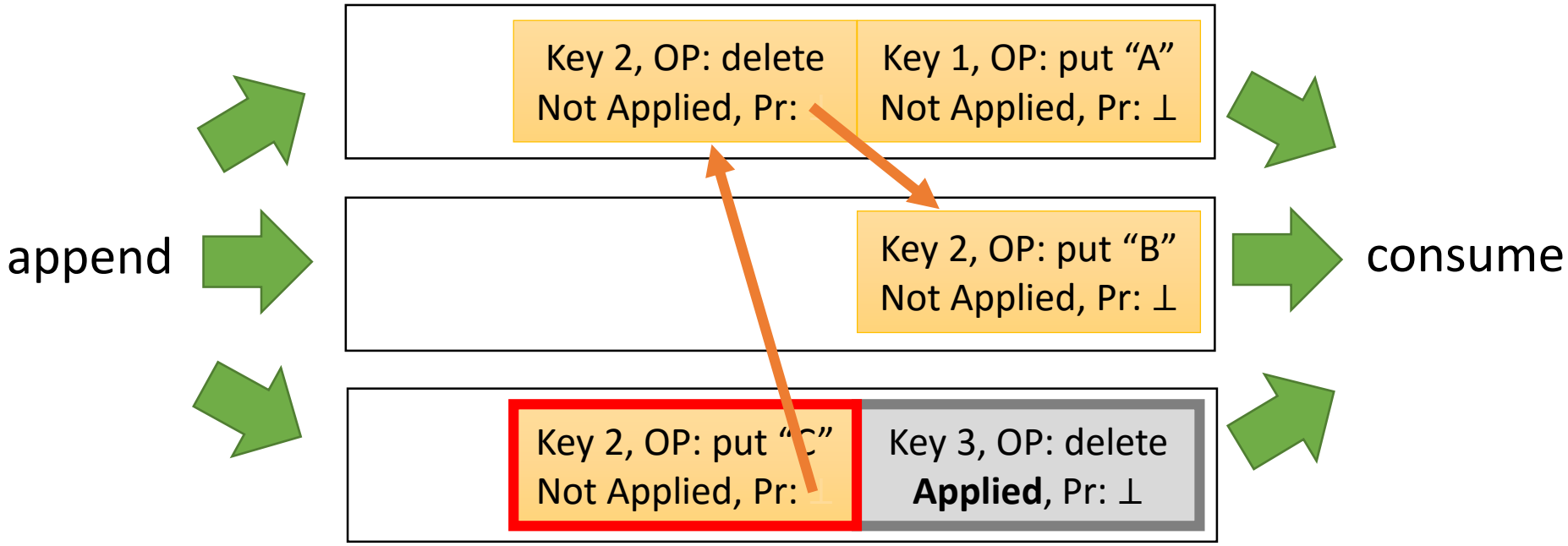
Optimization: Overwriting

Log entry:



Optimization: Overwriting

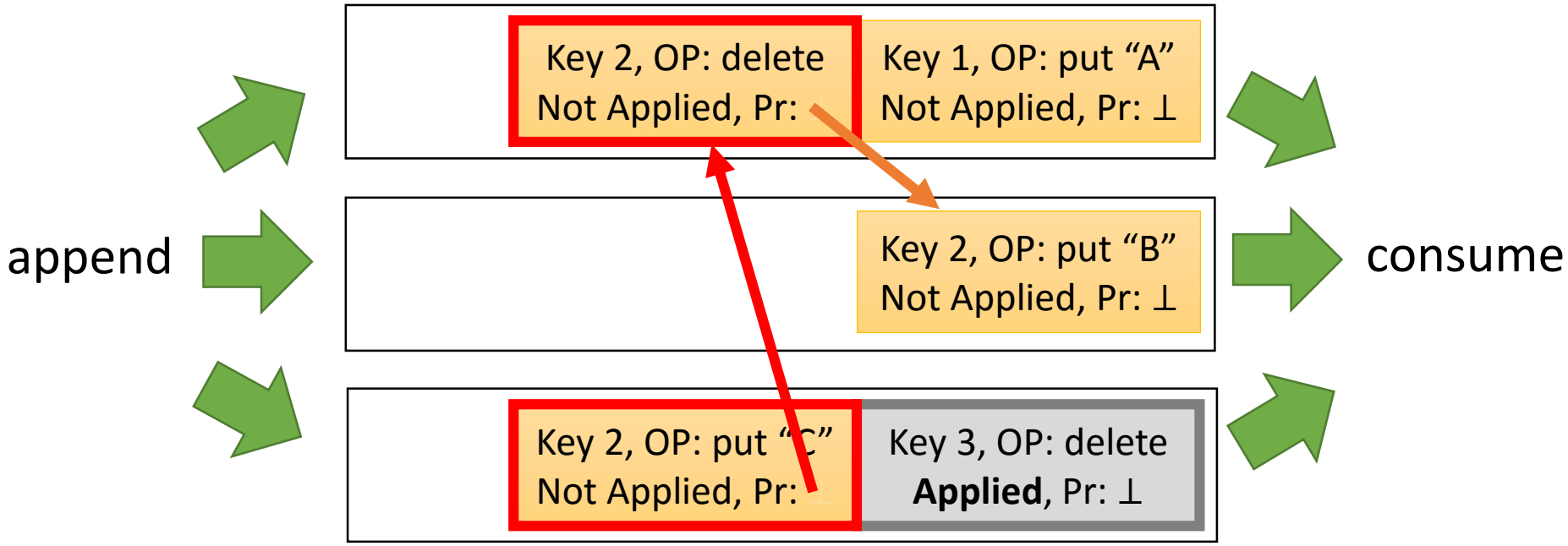
Log entry:



Optimization: Overwriting

Log entry:

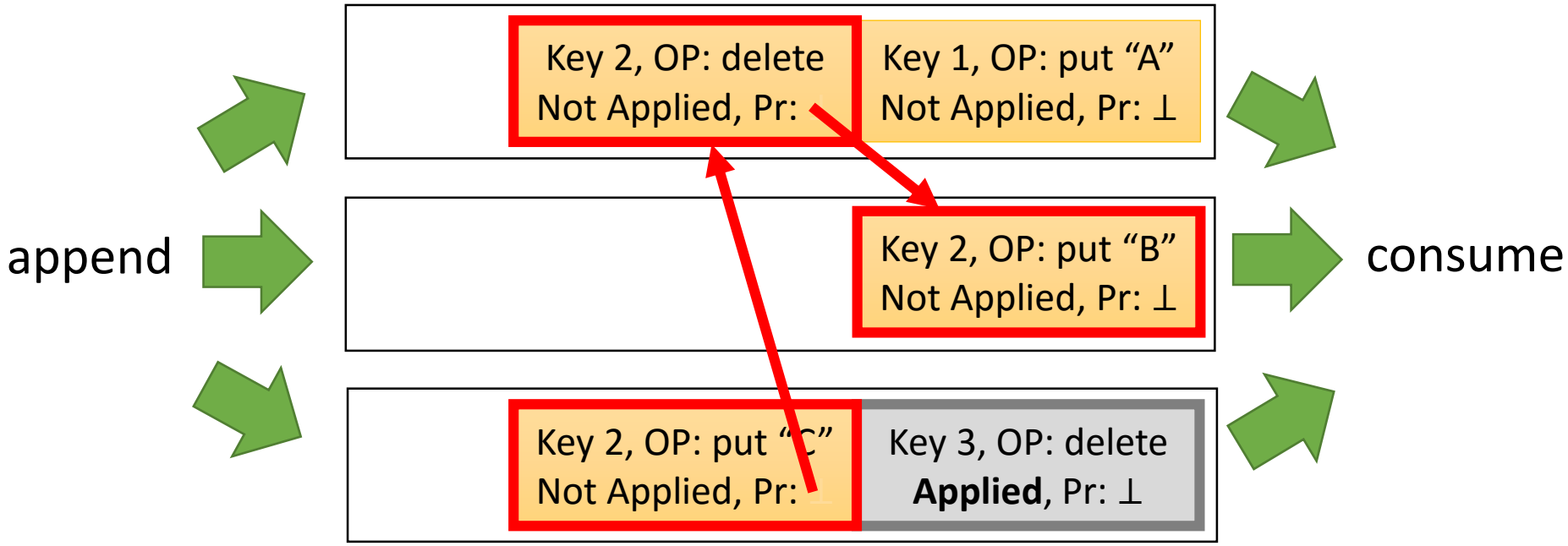
Key	OP	Applied	Prev



Optimization: Overwriting

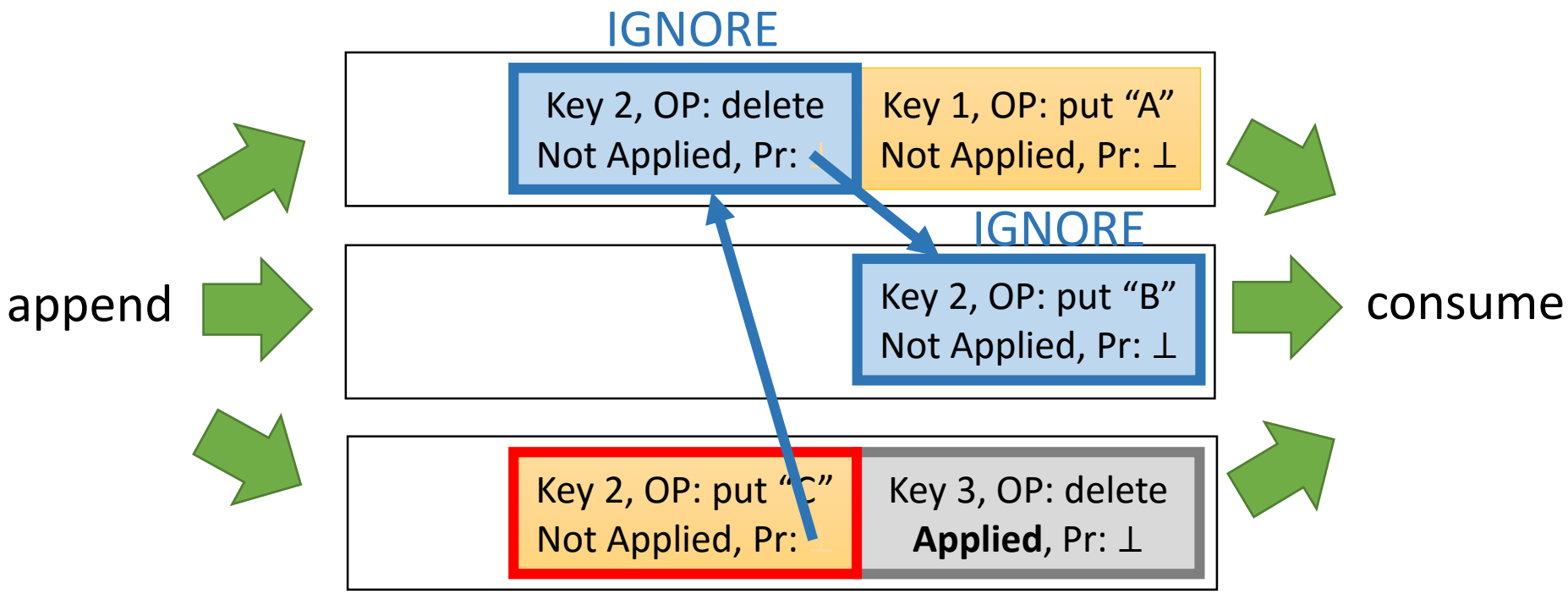
Log entry:

Key	OP	Applied	Prev

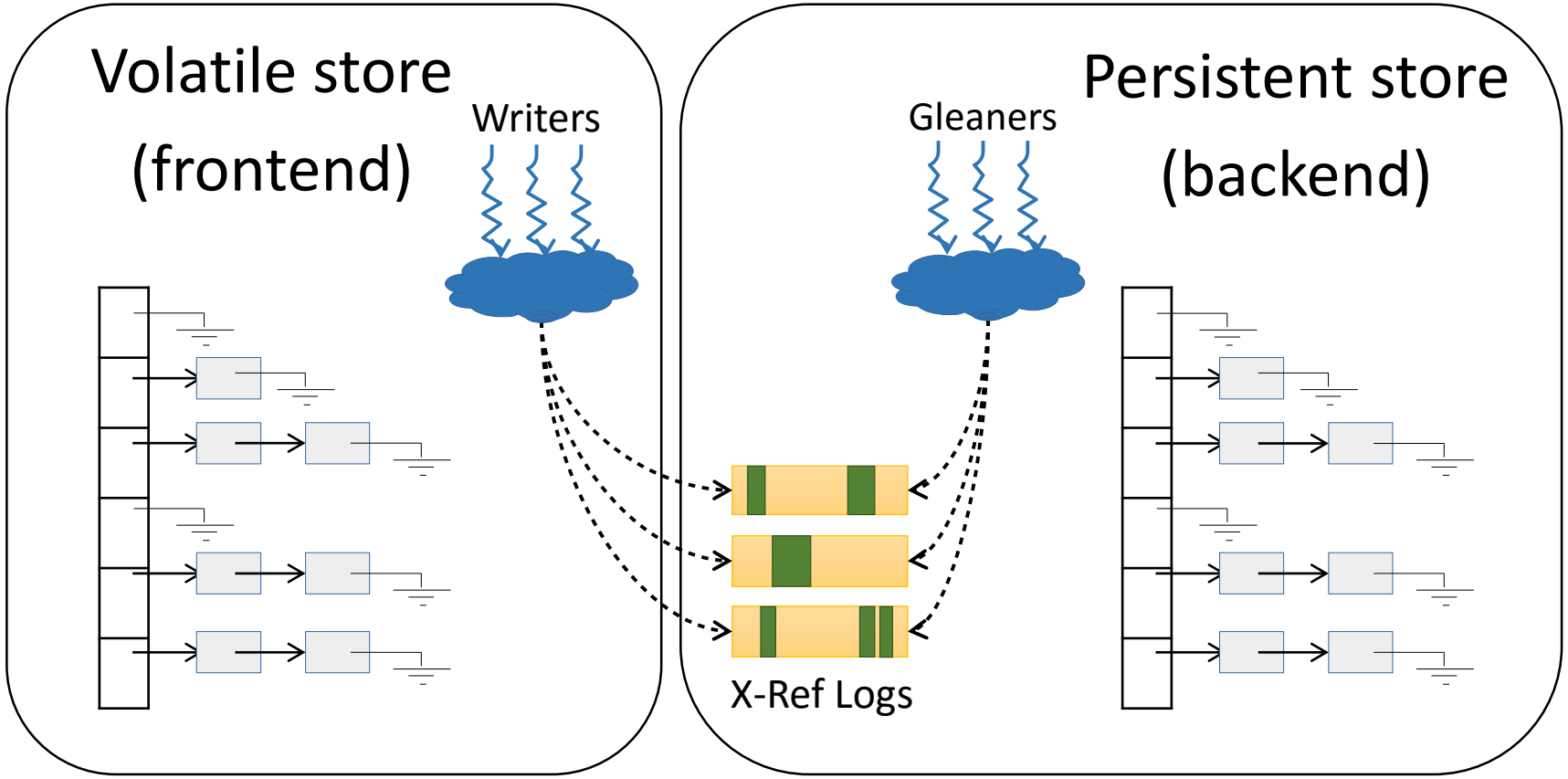


Optimization: Overwriting

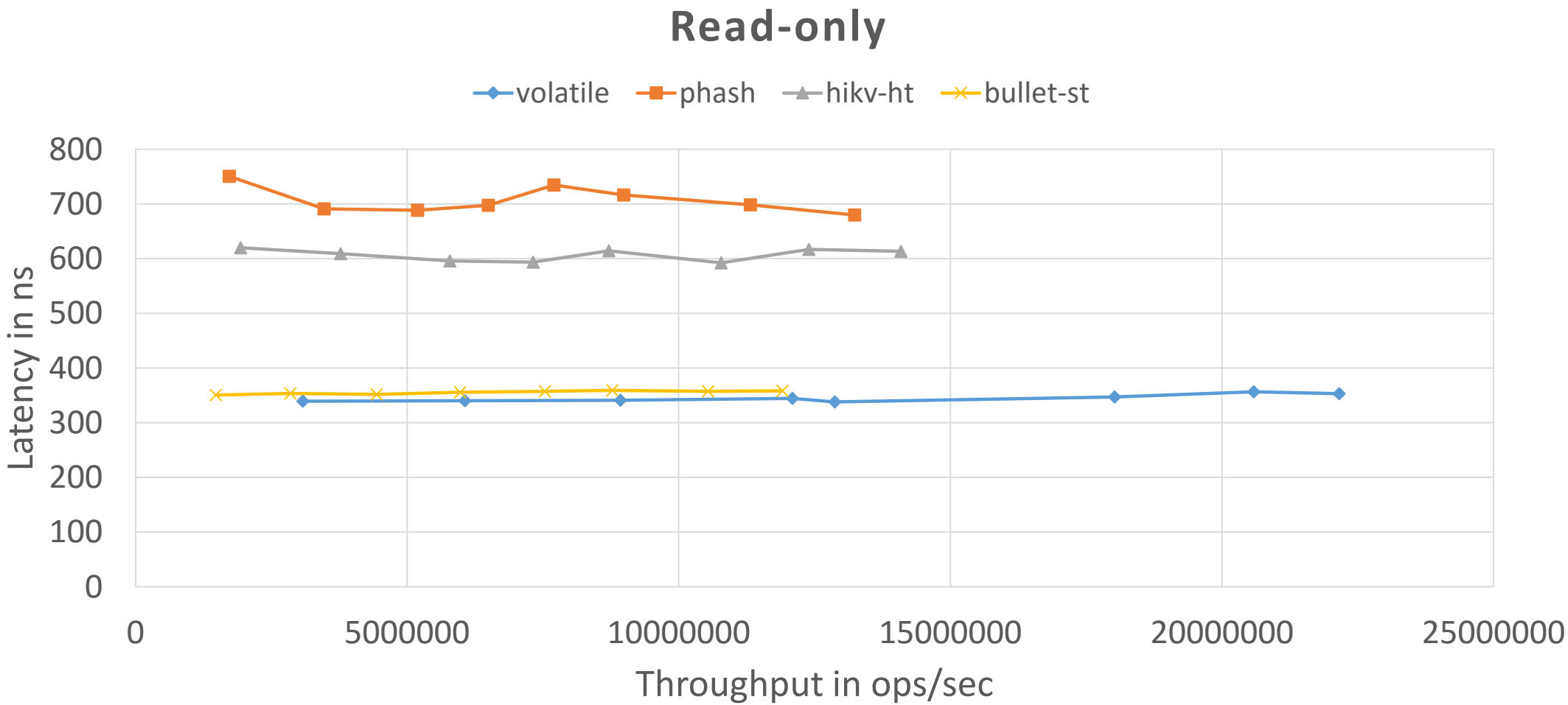
Log entry:



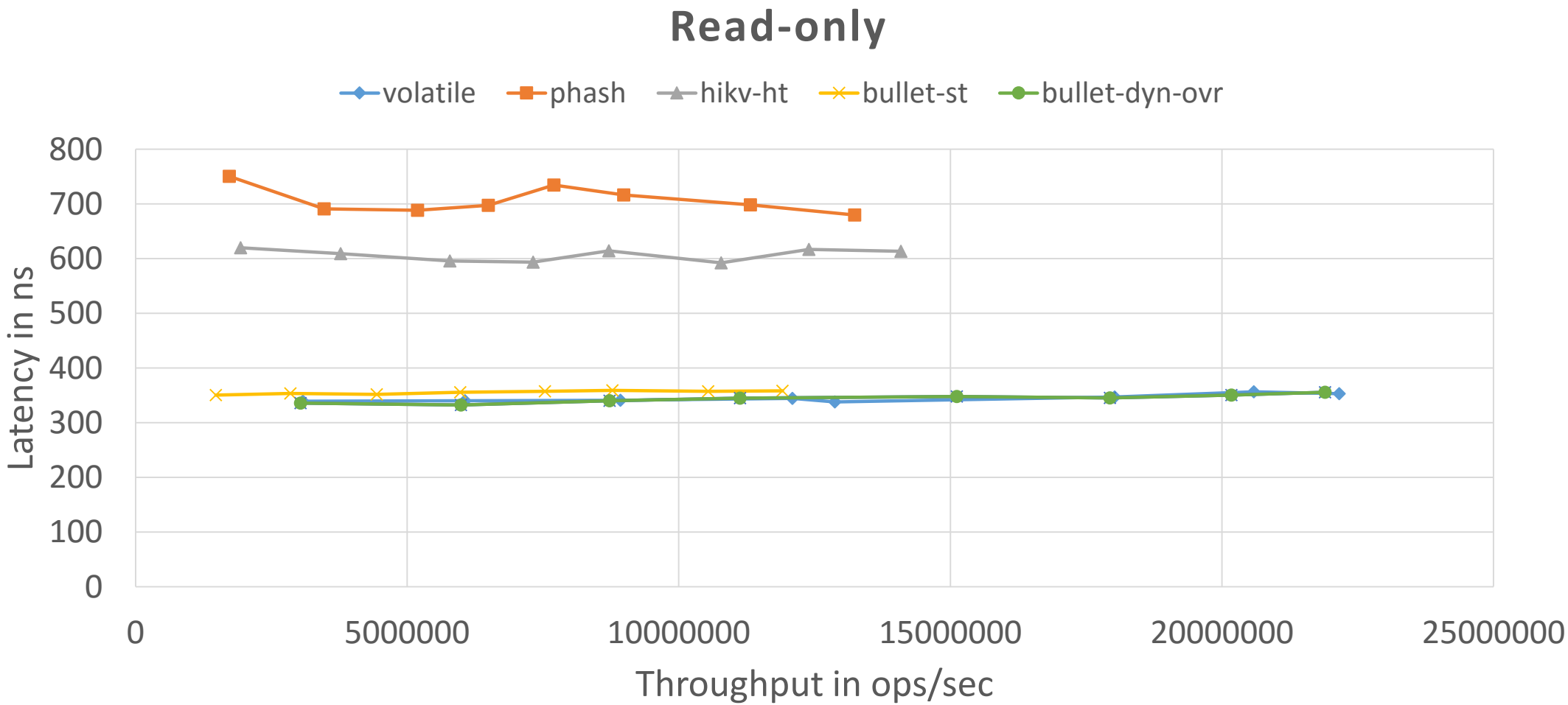
Optimization: Overwriting



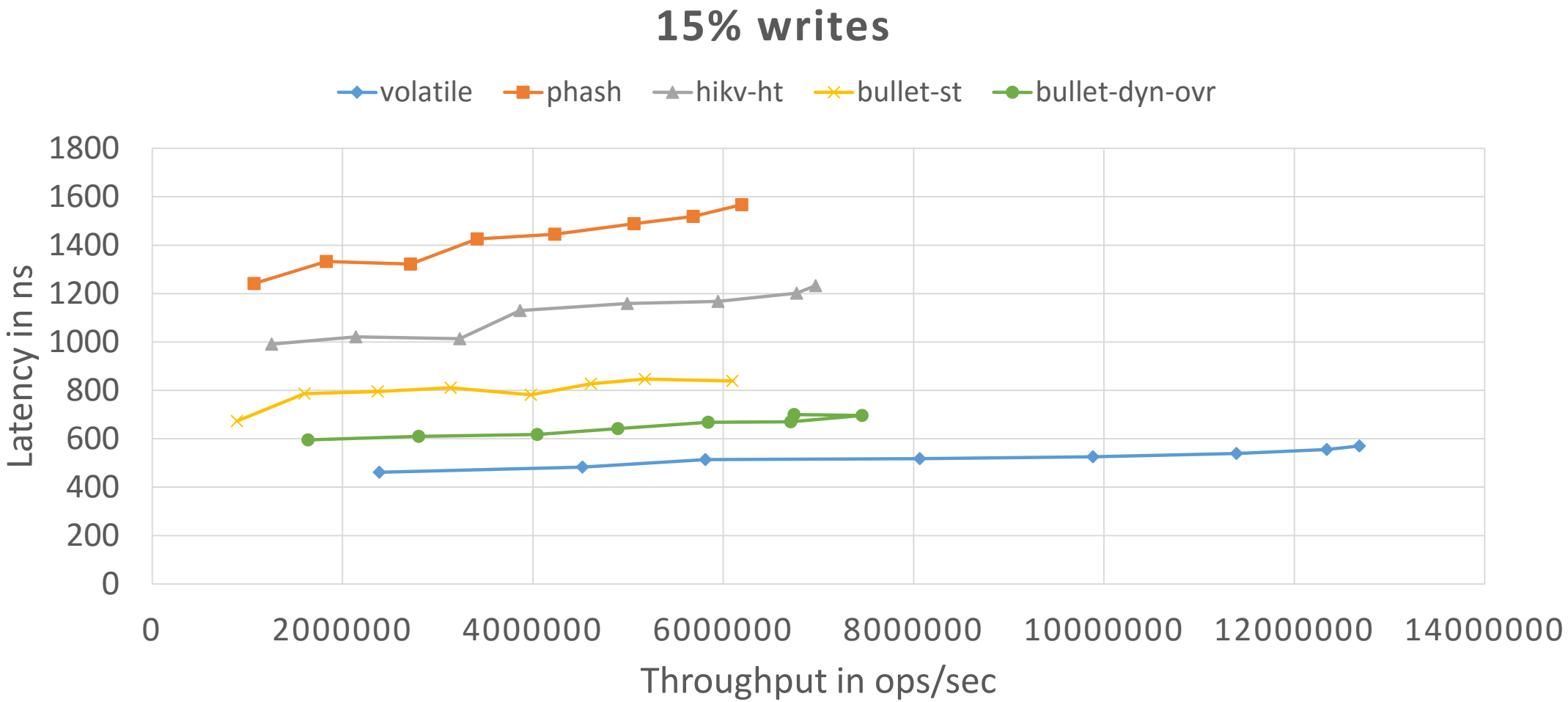
Bullet Performance



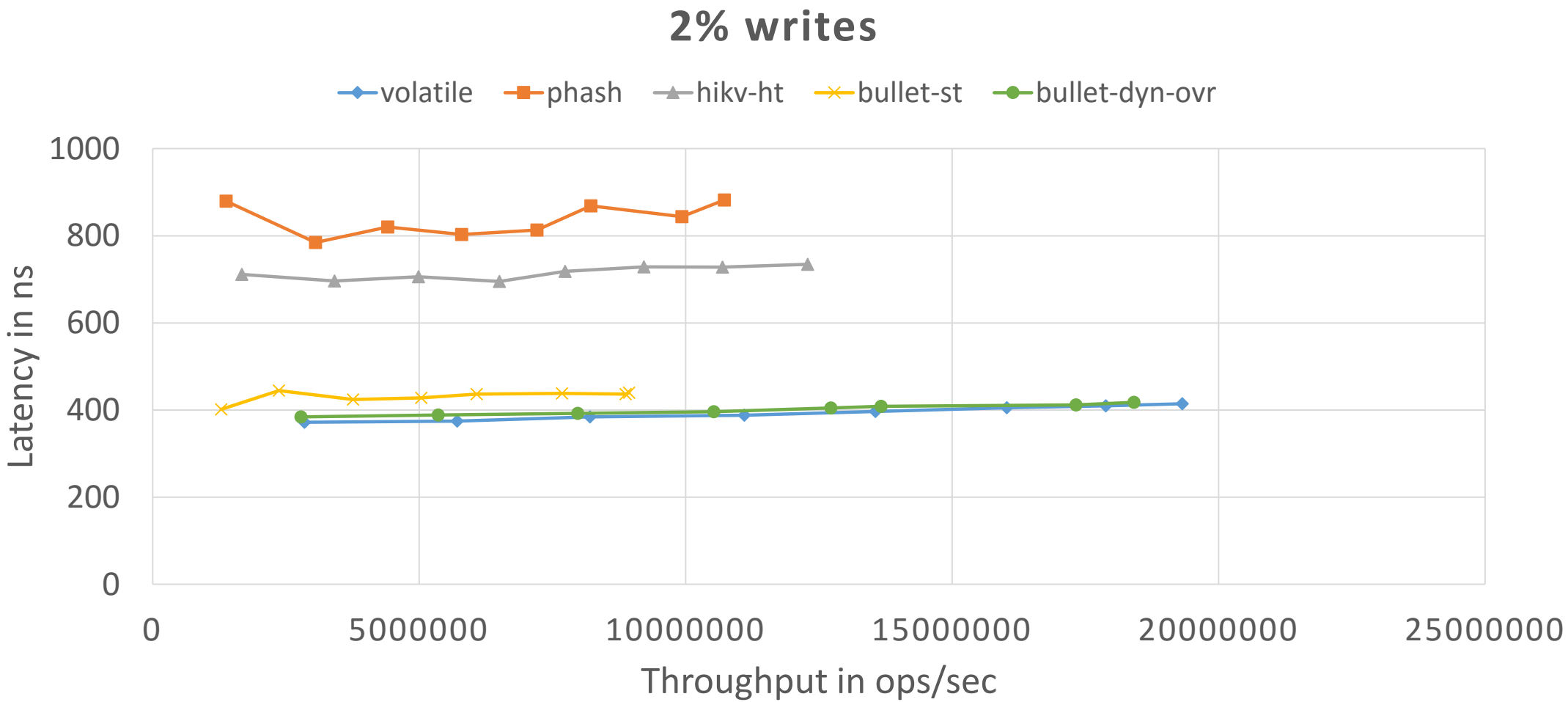
Bullet Performance



Bullet Performance

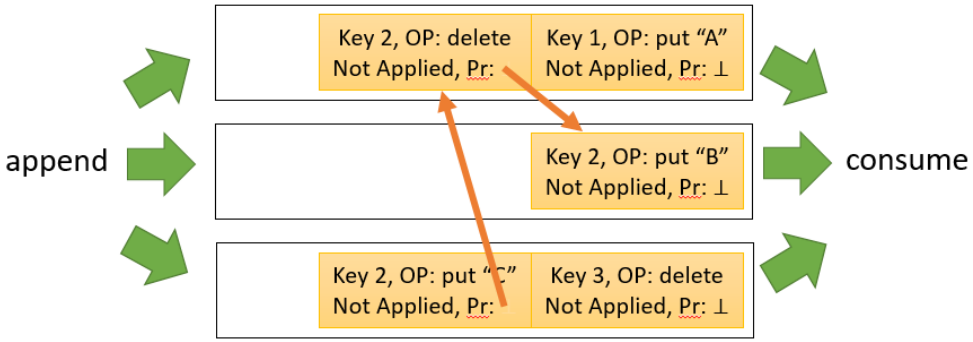


Bullet Performance



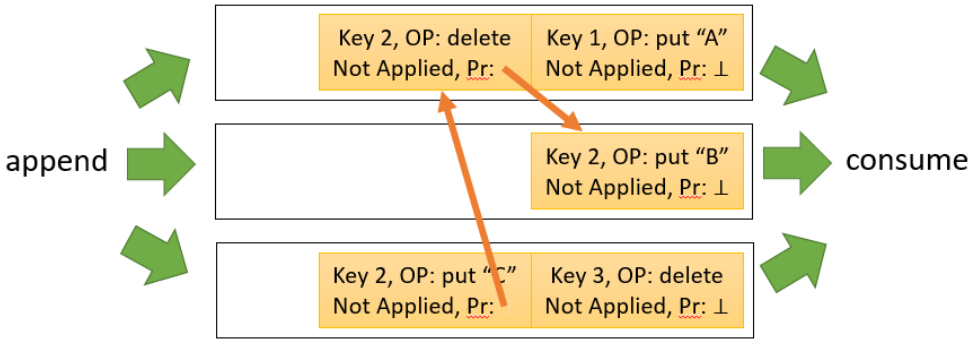
Summary

- DRAM cache for NVM
  - DRAM frontend, NVM backend
  - Novel caching scheme: Cross-referencing logs

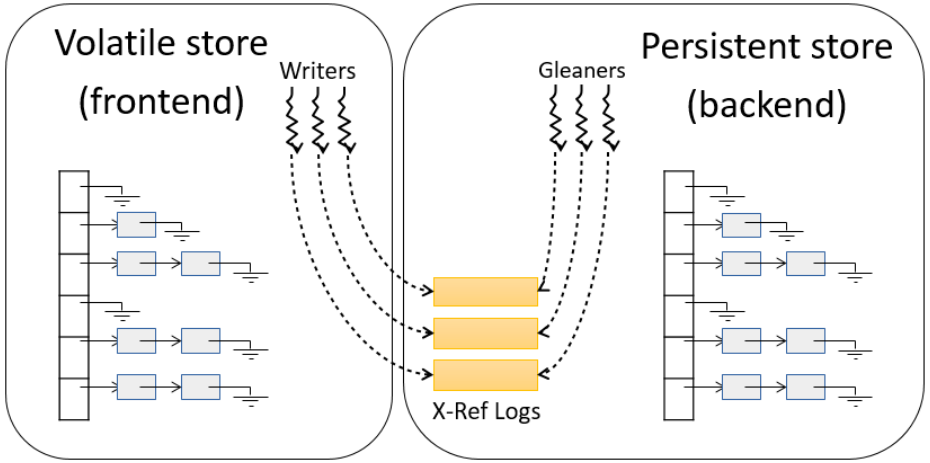


Summary

- DRAM cache for NVM
  - DRAM frontend, NVM backend
  - Novel caching scheme: Cross-referencing logs

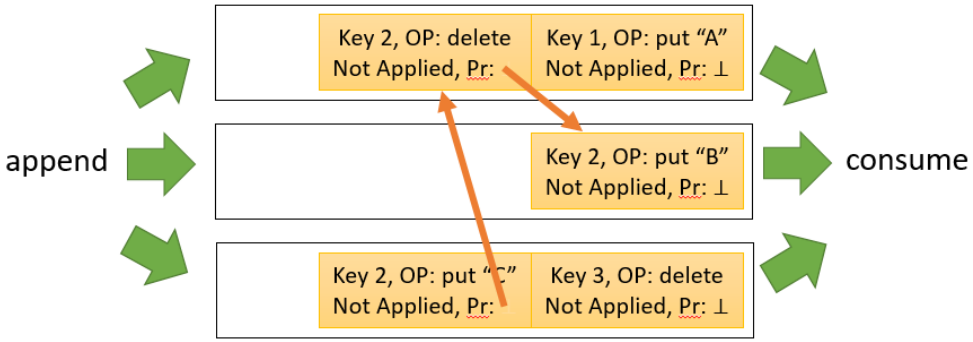


- Bullet K-V store
  - Almost identical hash tables in DRAM and NVM
  - DRAM performance for reads
  - Substantial latency improvements for writes

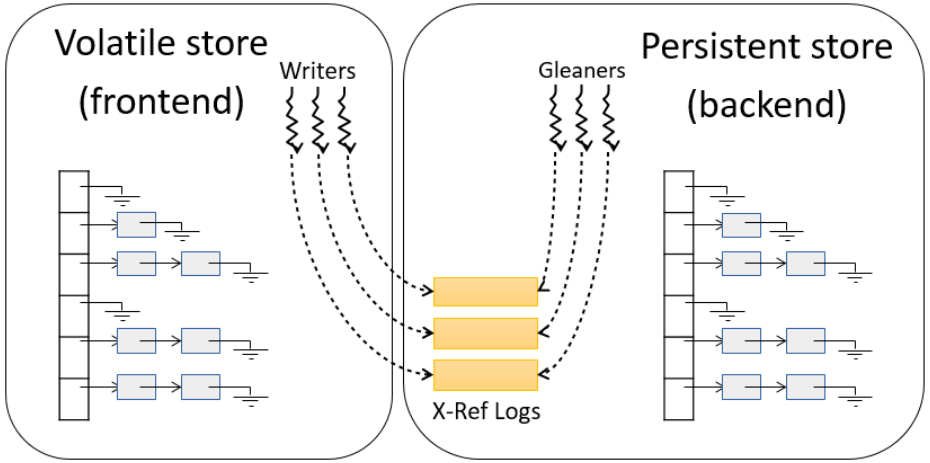


Summary

- DRAM cache for NVM
  - DRAM frontend, NVM backend
  - Novel caching scheme: Cross-referencing logs



- Bullet K-V store
  - Almost identical hash tables in DRAM and NVM
  - DRAM performance for reads
  - Substantial latency improvements for writes



- More details in the paper!

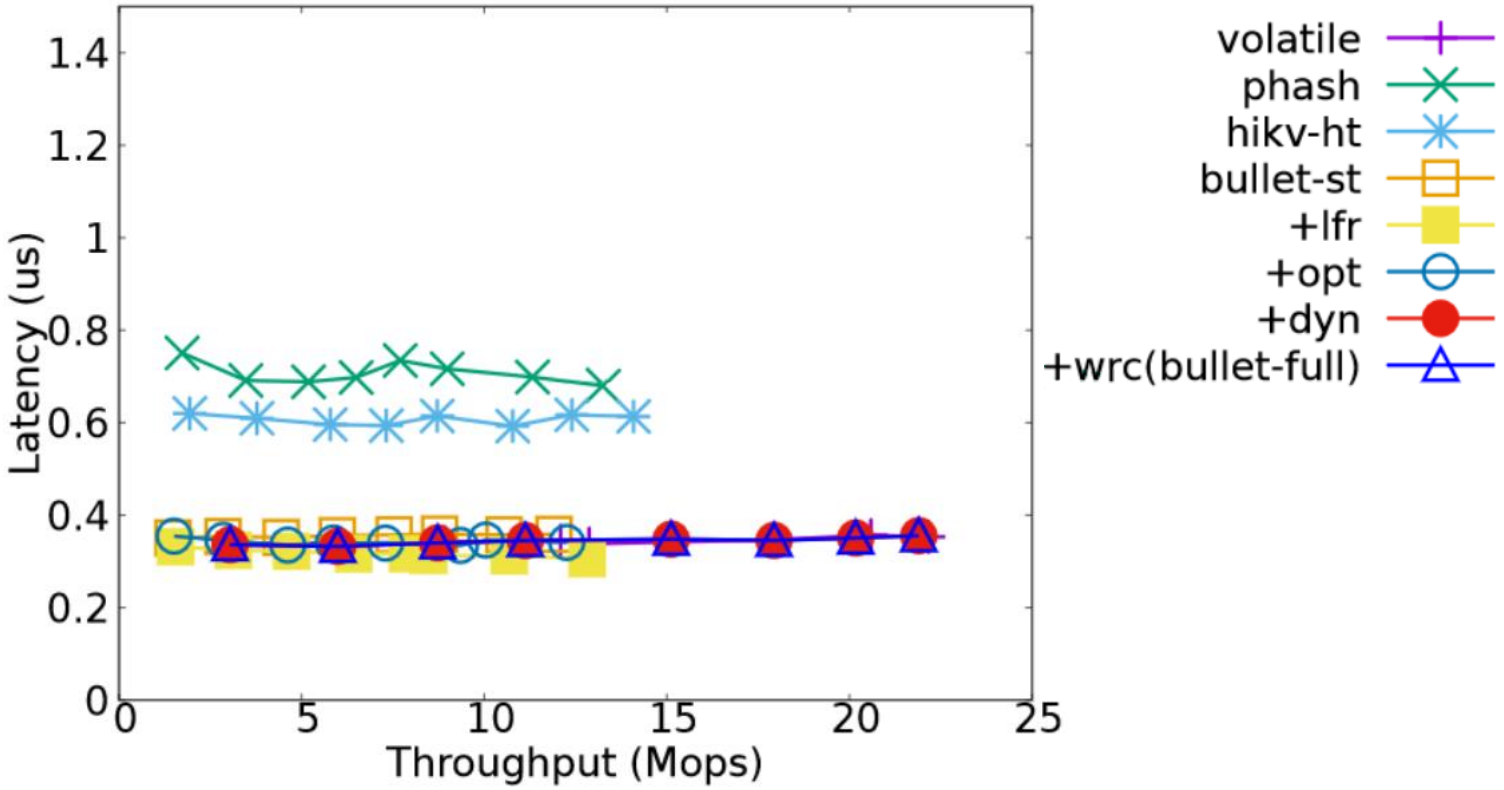
Backup Slides

Optimization Summary

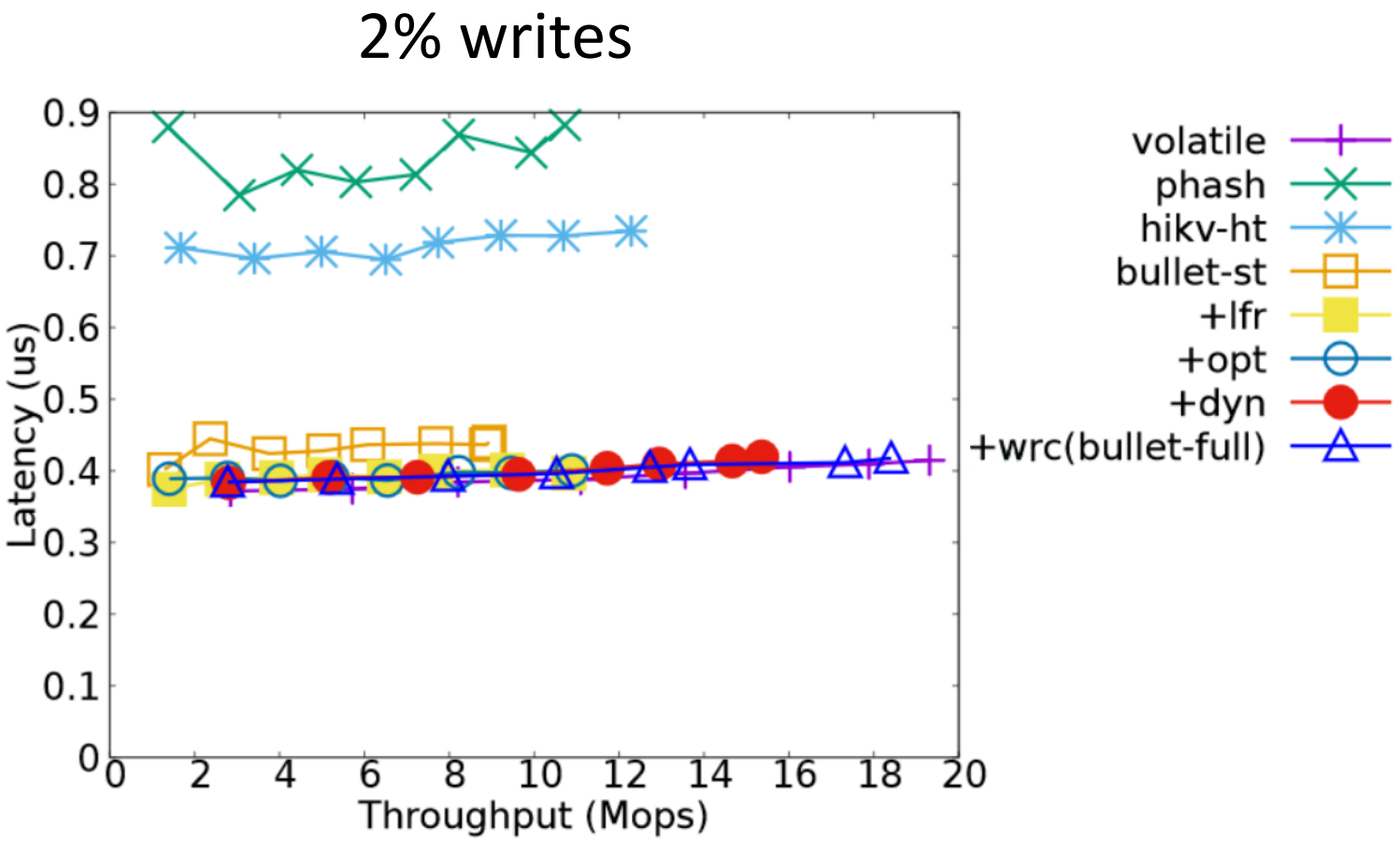
- **Dynamic worker threads**
- **Overwrites**
- Non-blocking reads (see paper)
- Backend access off critical path (see paper)

Bullet Performance

Read-only

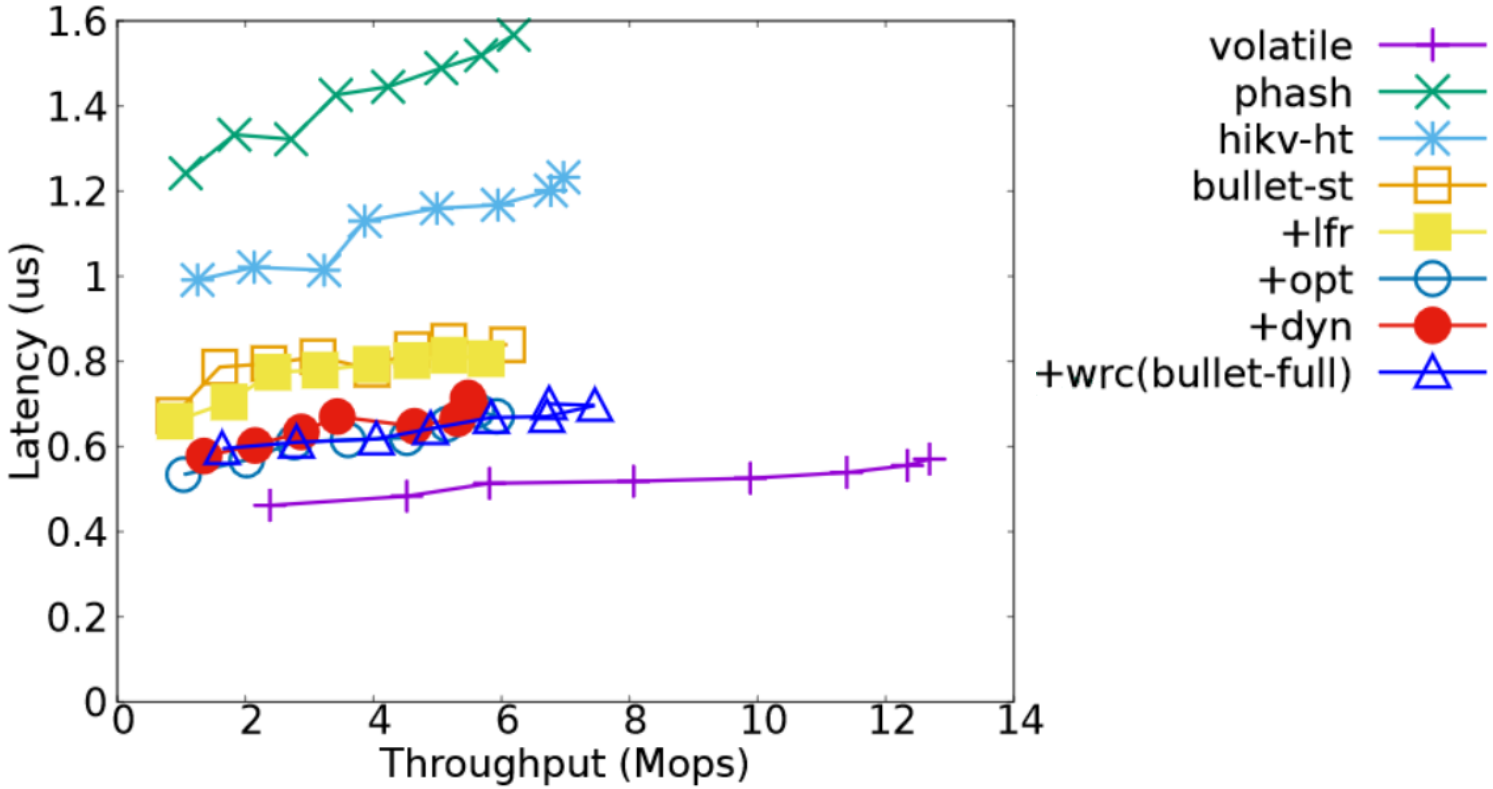


Bullet Performance



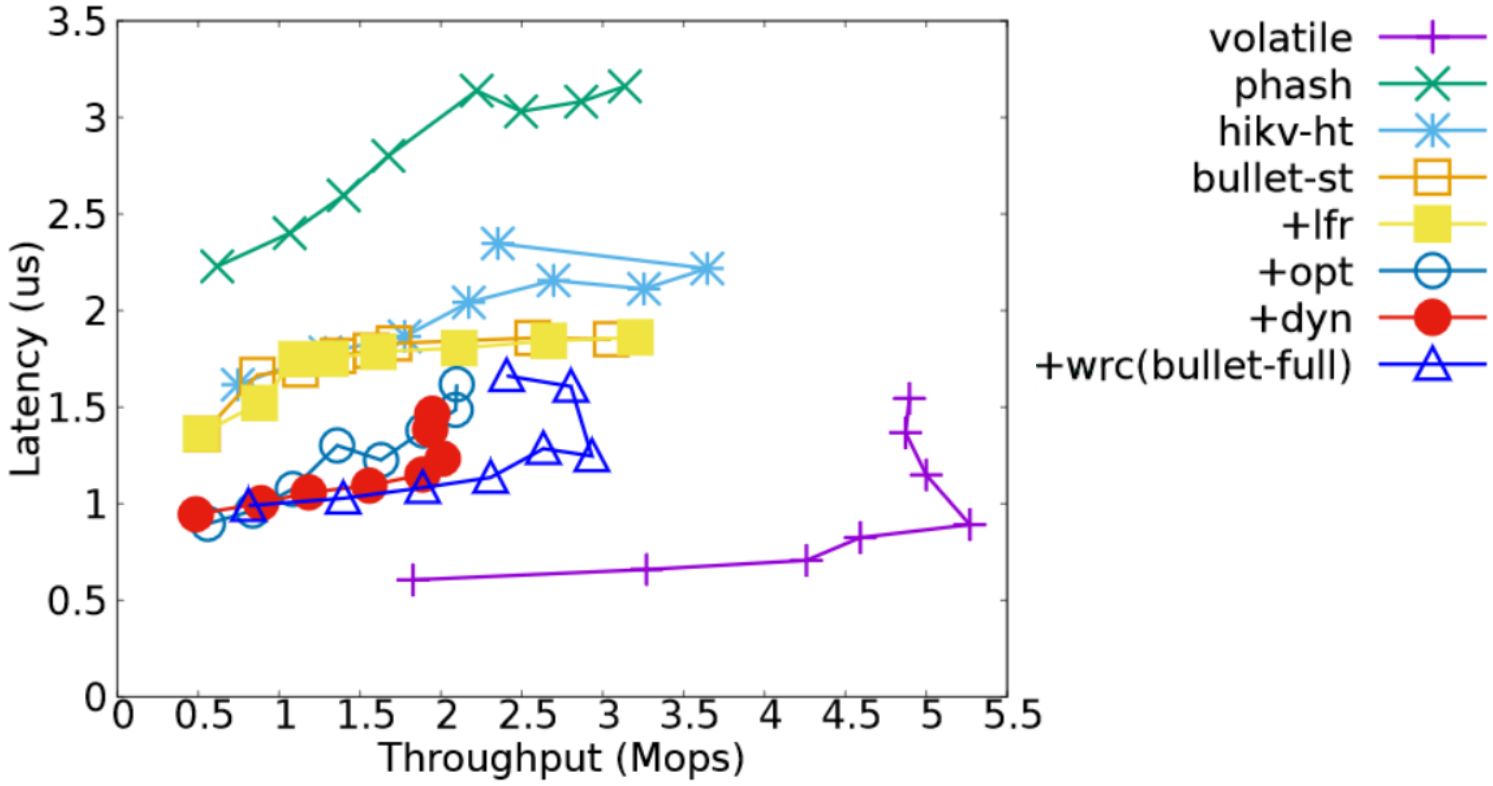
Bullet Performance

15% writes



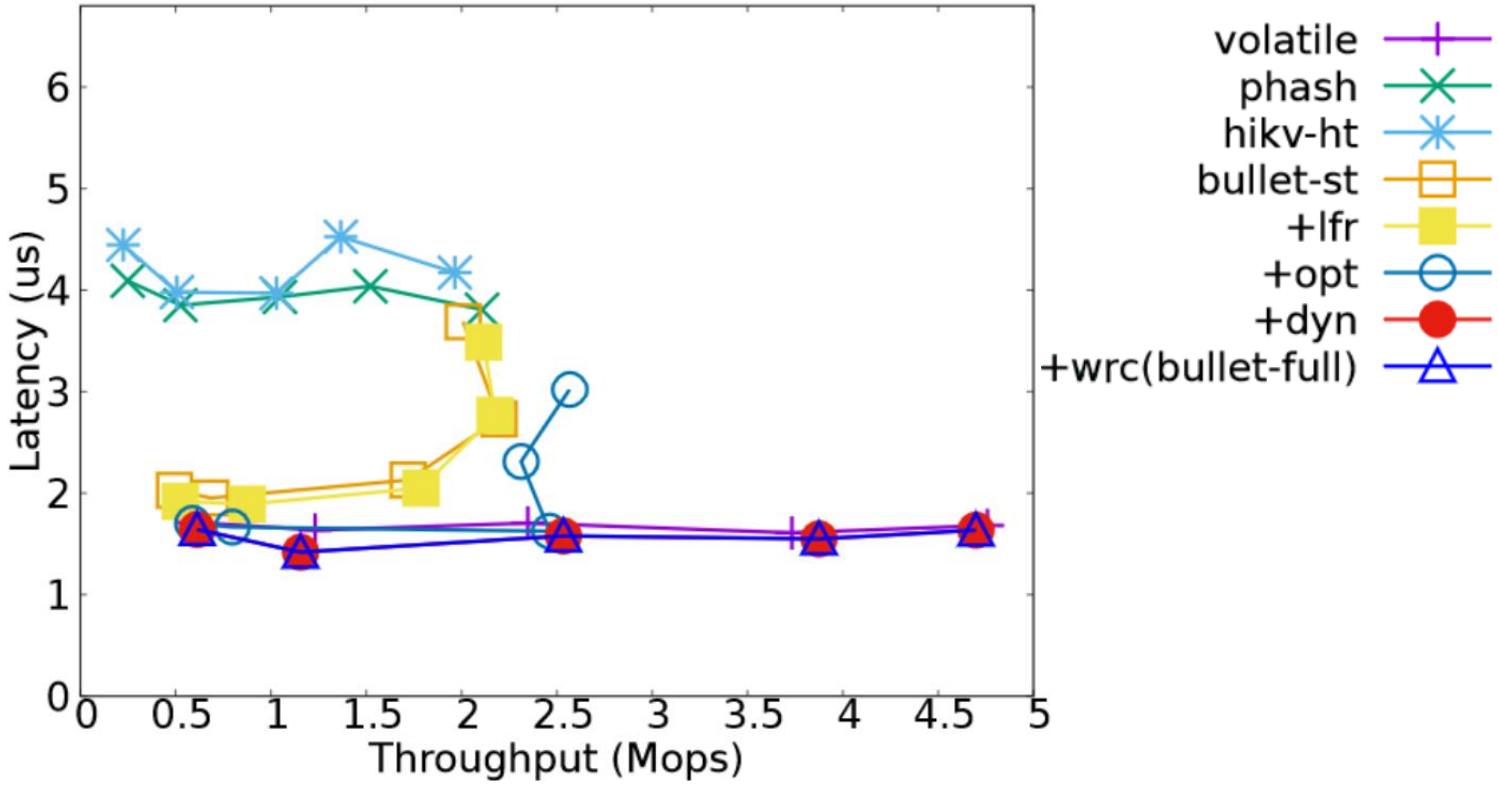
Bullet Performance

50% writes

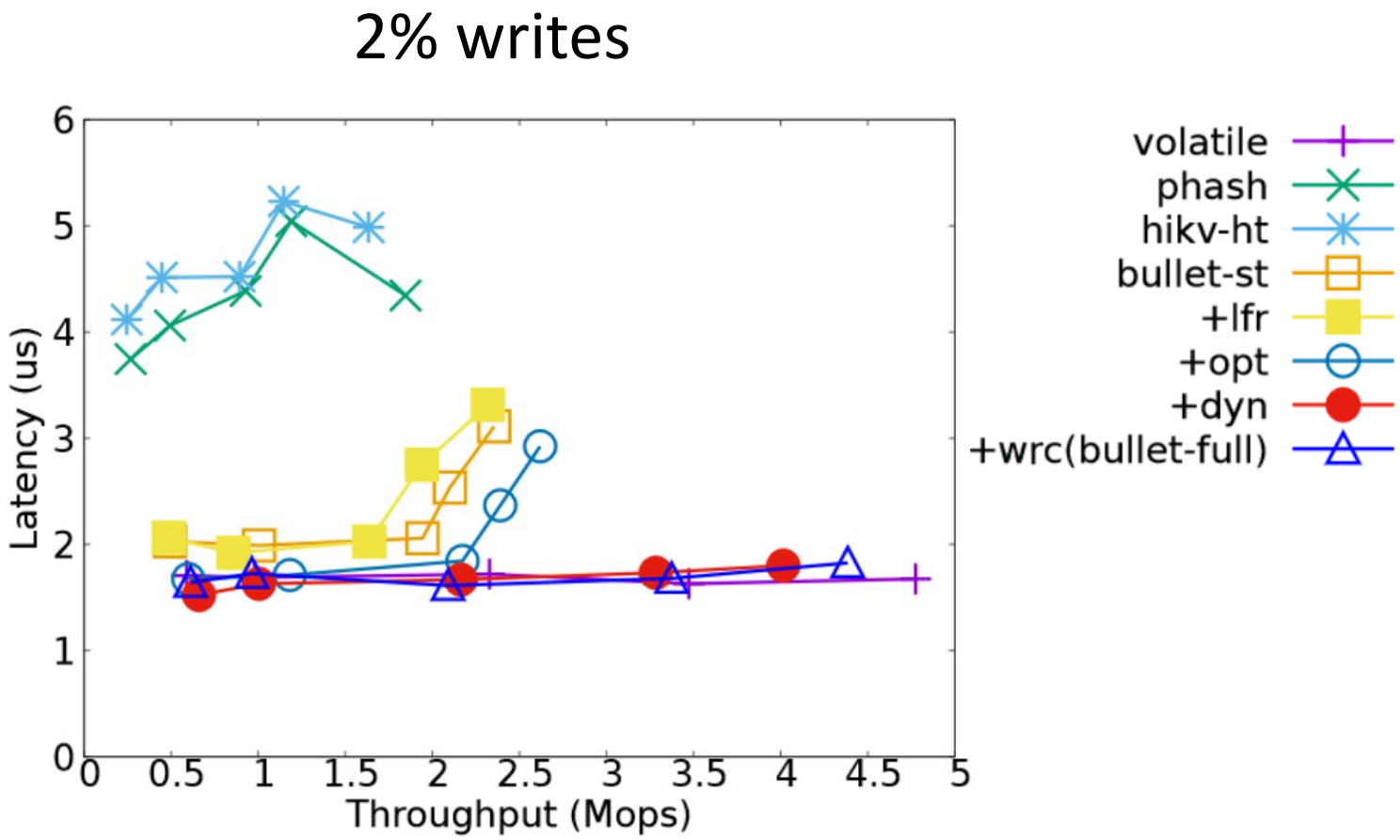


Bullet Performance (end-to-end)

Read-only

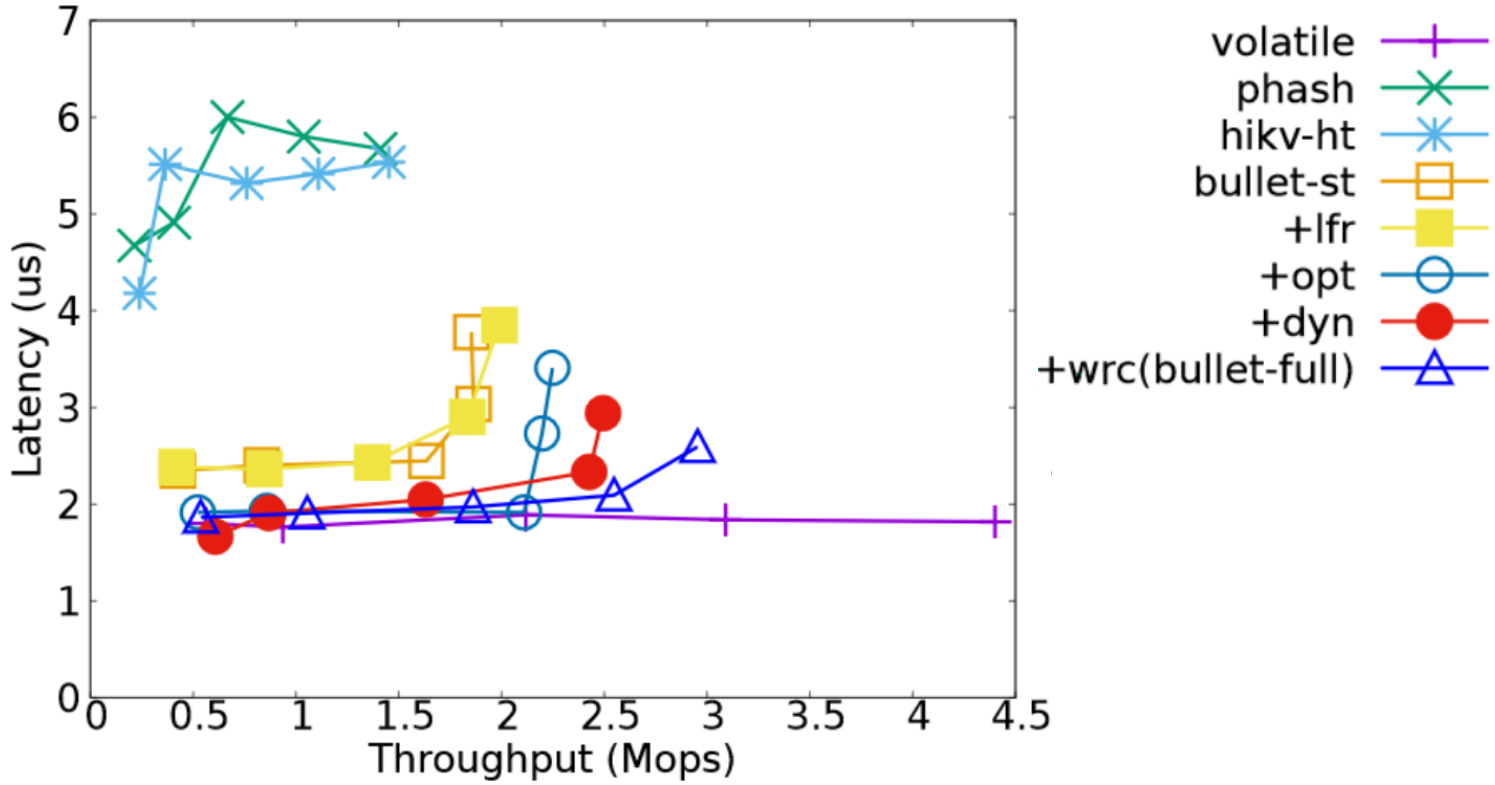


Bullet Performance (end-to-end)



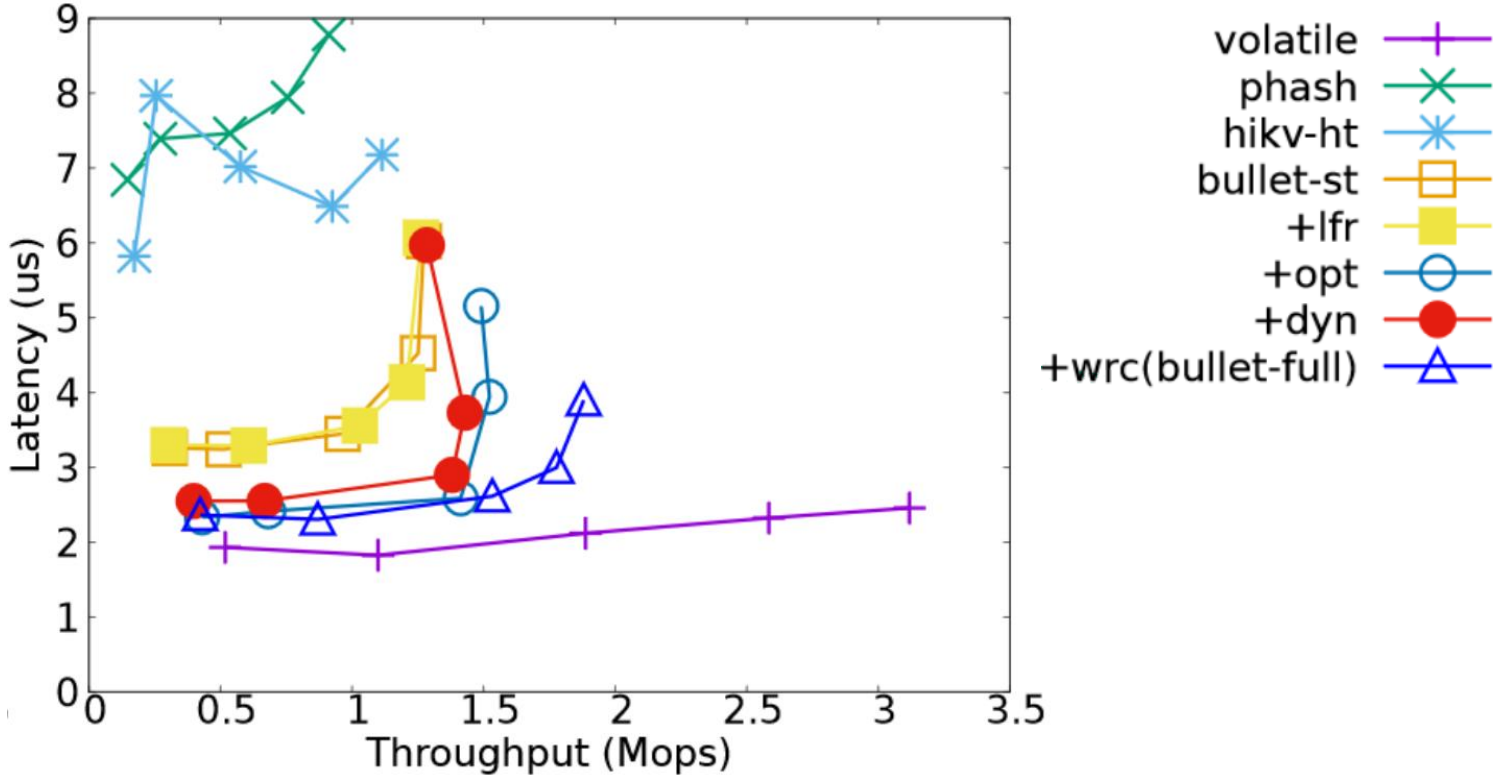
Bullet Performance (end-to-end)

15% writes



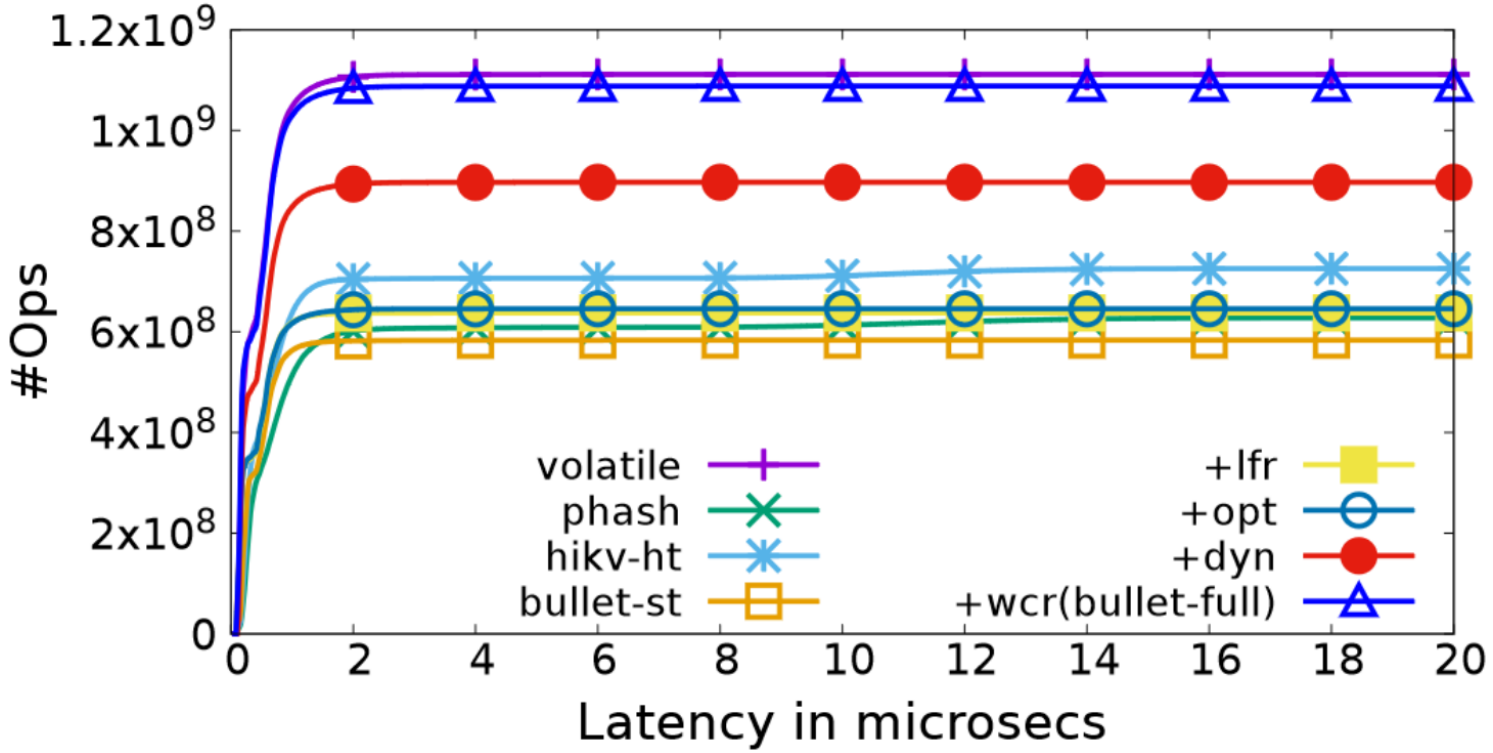
Bullet Performance (end-to-end)

50% writes



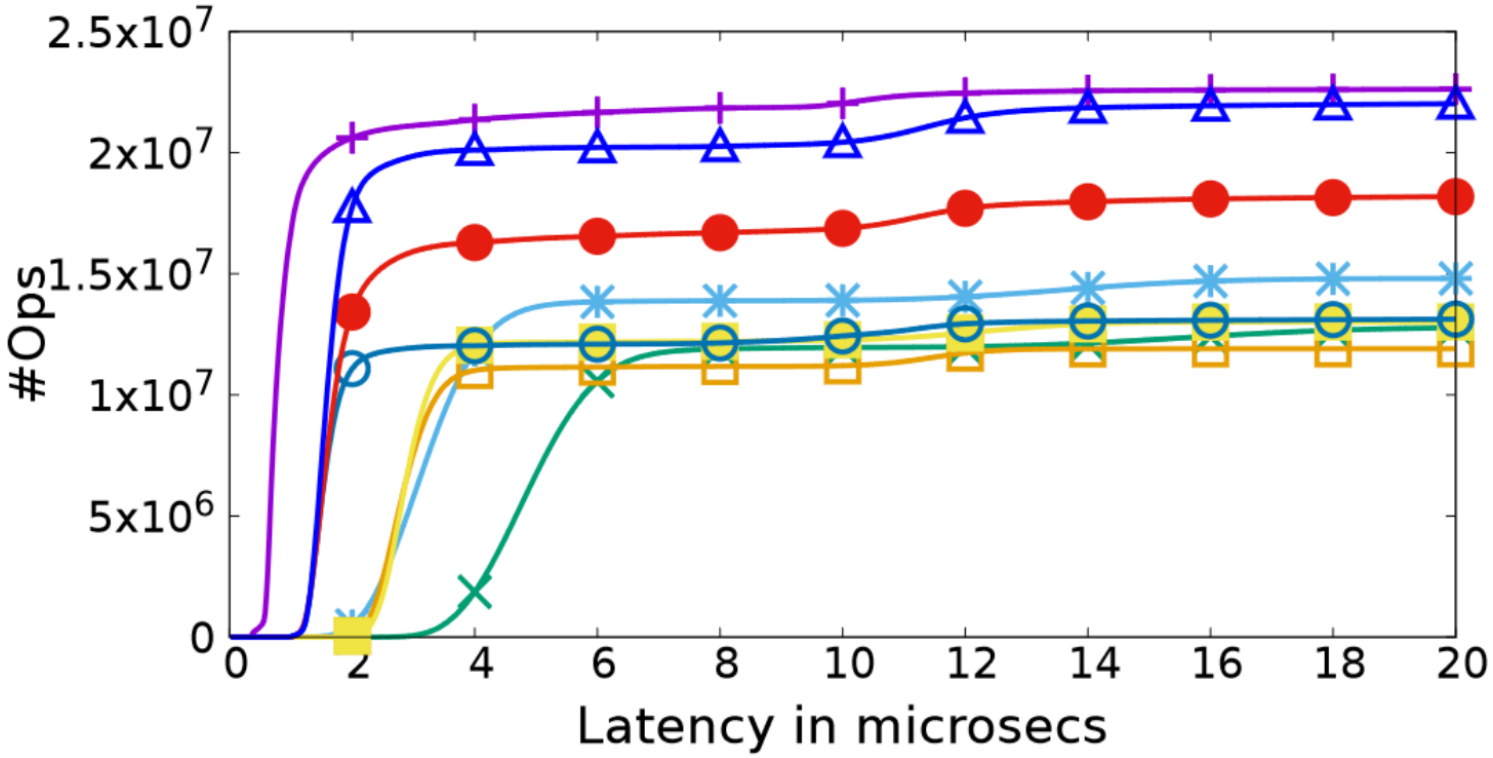
Latency Distribution

Reads

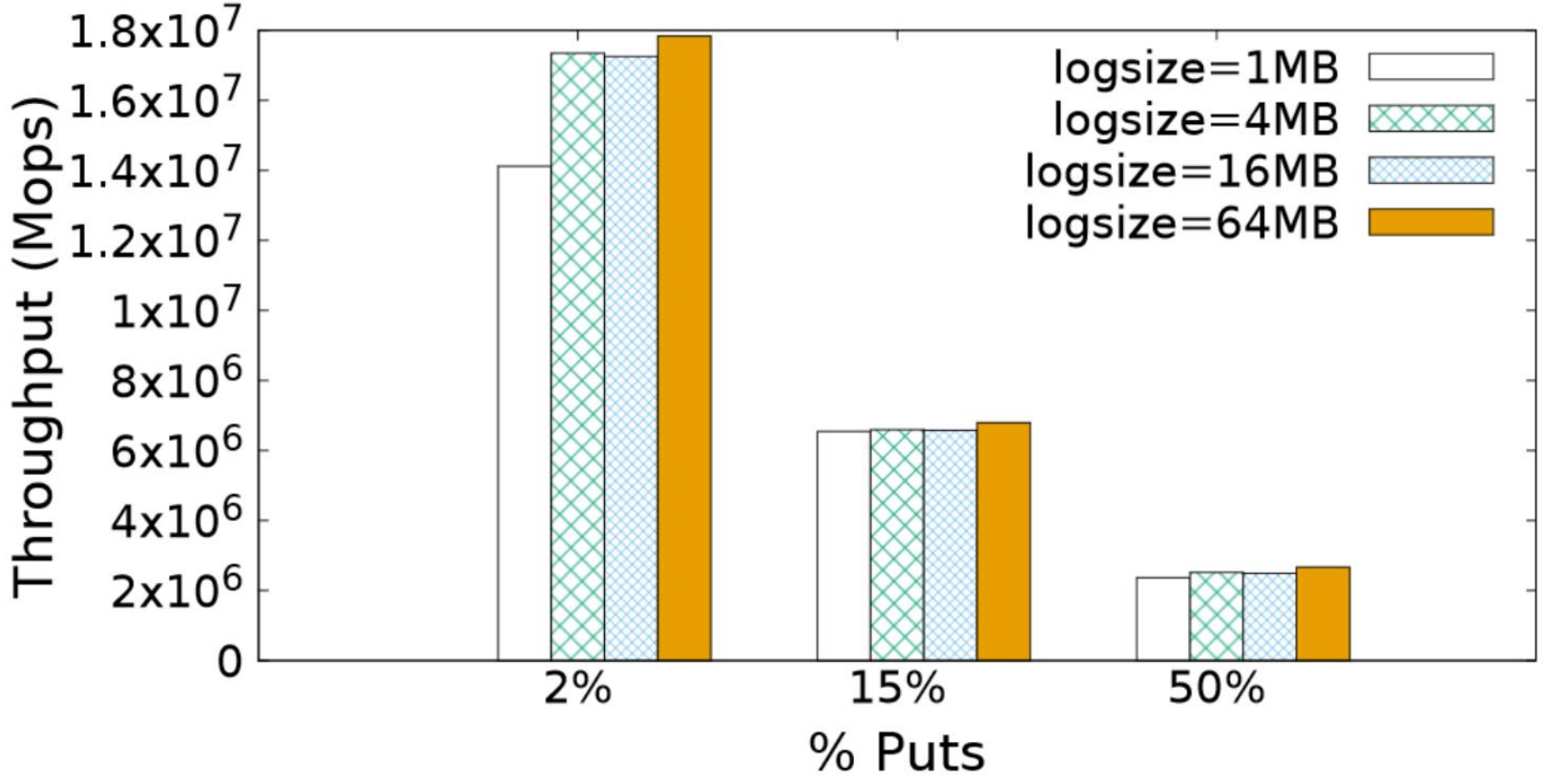


Latency Distribution

Writes



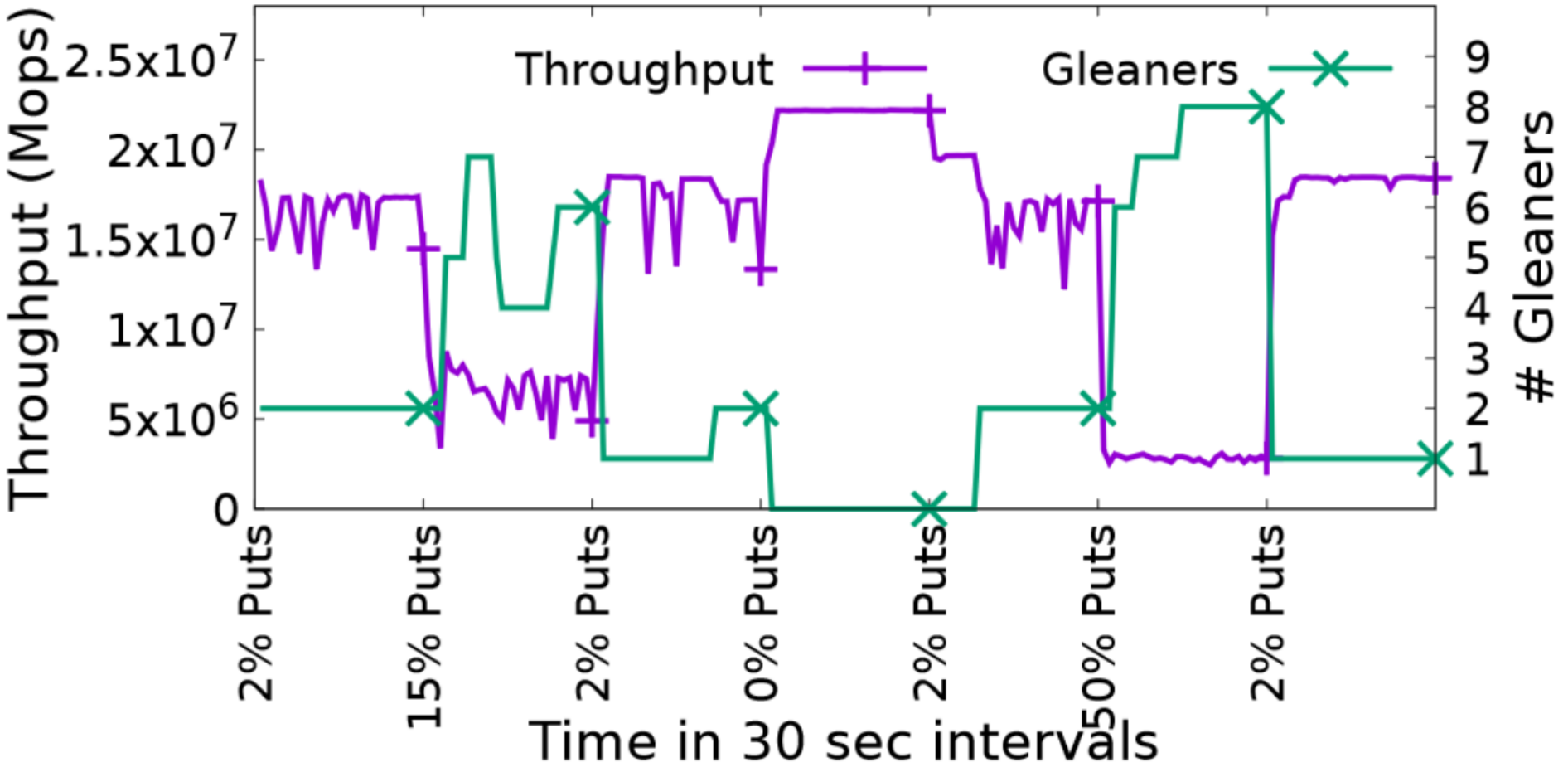
Log Size Senzitivity



Experimental Setup

- 16 CPU cores
- 512 GB DRAM
- NVM simulated using DRAM
  - 384 GB (out of 512GB)
  - 300ns load (2x DRAM)
  - 100ns persist barrier
  - Same throughput
- DRAM cache fits all data
- Zipfian distribution of keys (YCSB)
- 50M K/V pairs, 16 B keys, 100 B values
- Measuring 99%ile latency

Dynamic Worker Thread Switching

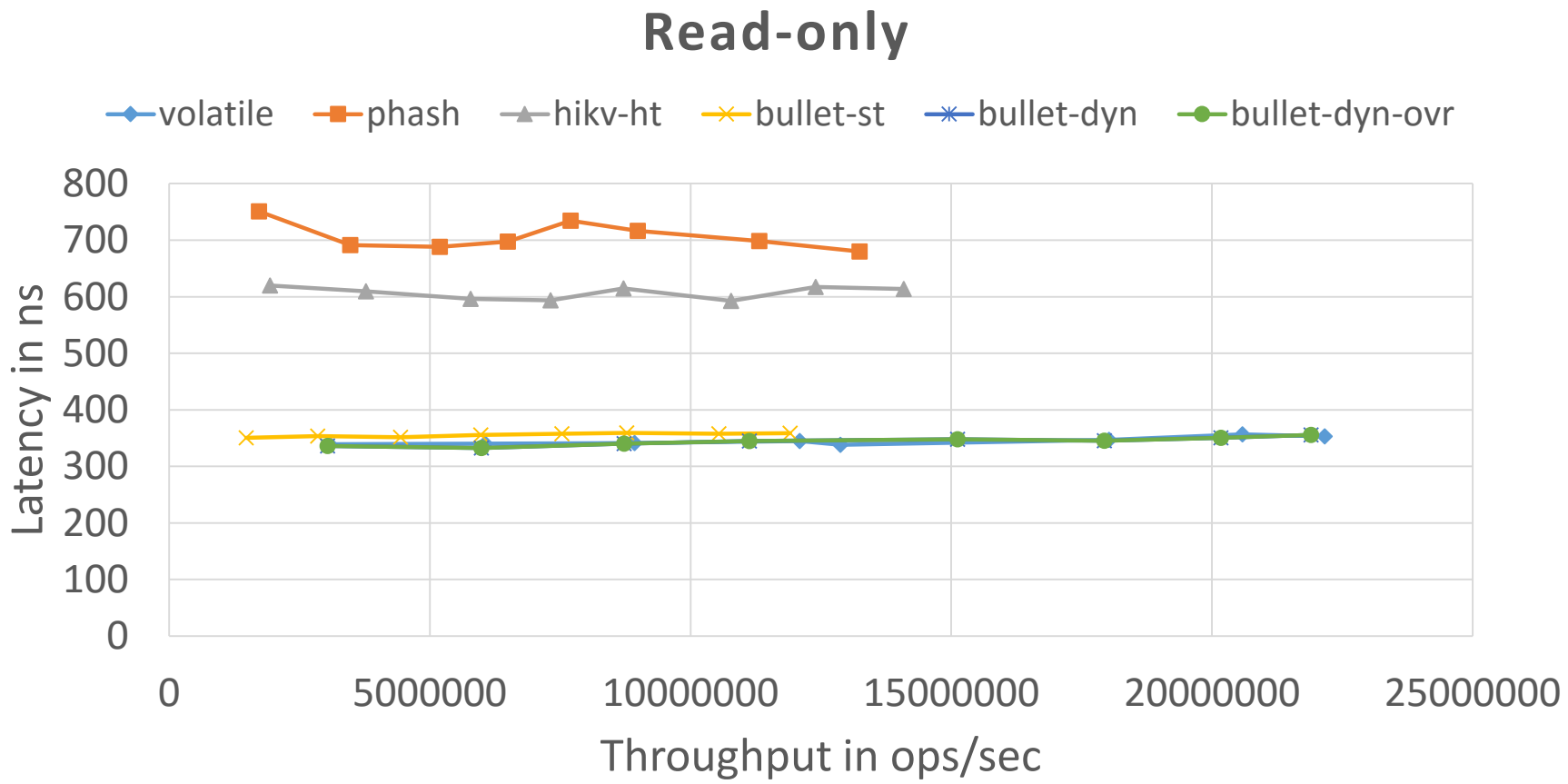


Cross-Referencing Logs

- Persistence:
  - Circular buffers in NVM
- Low latency
  - Fast appends (3 persist barriers)
- Scalability
  - Multiple logs, stall only when all full
  - Coarse-grained synchronization
  - Epoch-based space reclamation
- Consistency
  - Cross-log references

Cross-Referencing Logs Summary

- Persistent circular buffers (in NVM)
- Cross-log pointers for consistency
- Coarse-grained synchronization
- Fast appends
- Epoch-based space reclamation


Read-only
volatile
phash
hikv-ht
bullet-st
bullet-dyn
bullet-dyn-ovr
800
700
600
500
400
300
200
100
0
Latency in ns
0
5000000
10000000
15000000
20000000
25000000
Throughput in ops/sec