



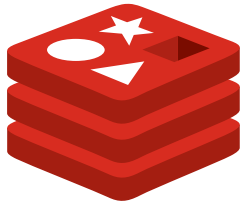
FORGE: Mitigating Synchronization Amplification for Memory-Disaggregated Caching Systems

Zhijun Yang, Yu Hua, Ming Zhang, Menglei Chen, Yixiao Wang
Huazhong University of Science and Technology, China

In-Memory Caching Systems

Fundamental infrastructure in cloud datacenters

- Cache hot objects in memory
- Serve low-latency Get/Set
- Offload backend storage



redis

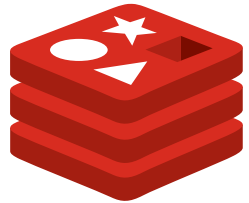


MEMCACHED

In-Memory Caching Systems

Fundamental infrastructure in cloud datacenters

- Cache hot objects in memory
- Serve low-latency Get/Set
- Offload backend storage

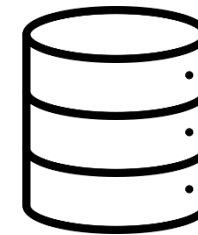


redis

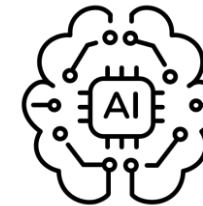


MEMCACHED

Widely used in:



Databases



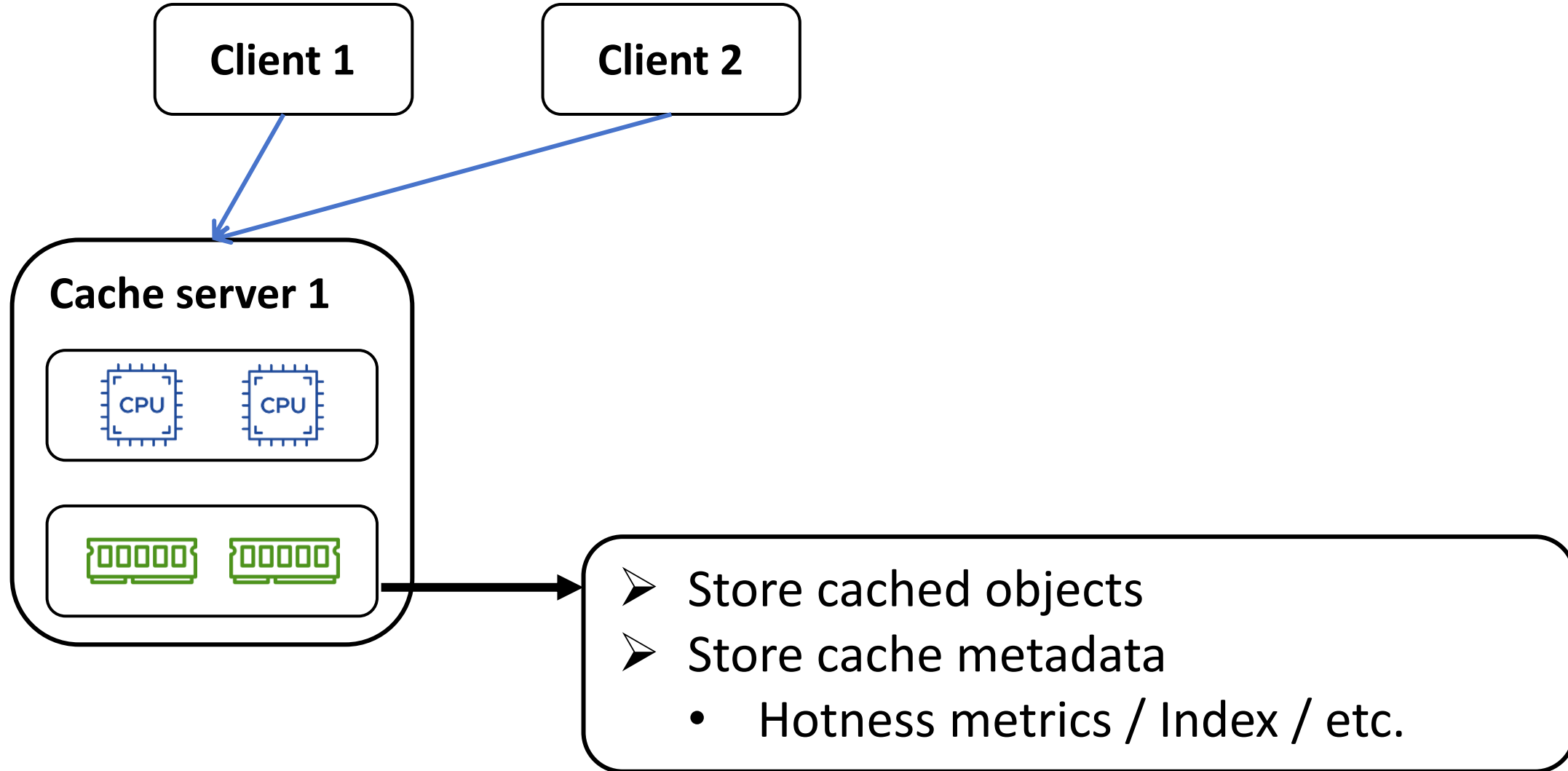
Machine
Learning



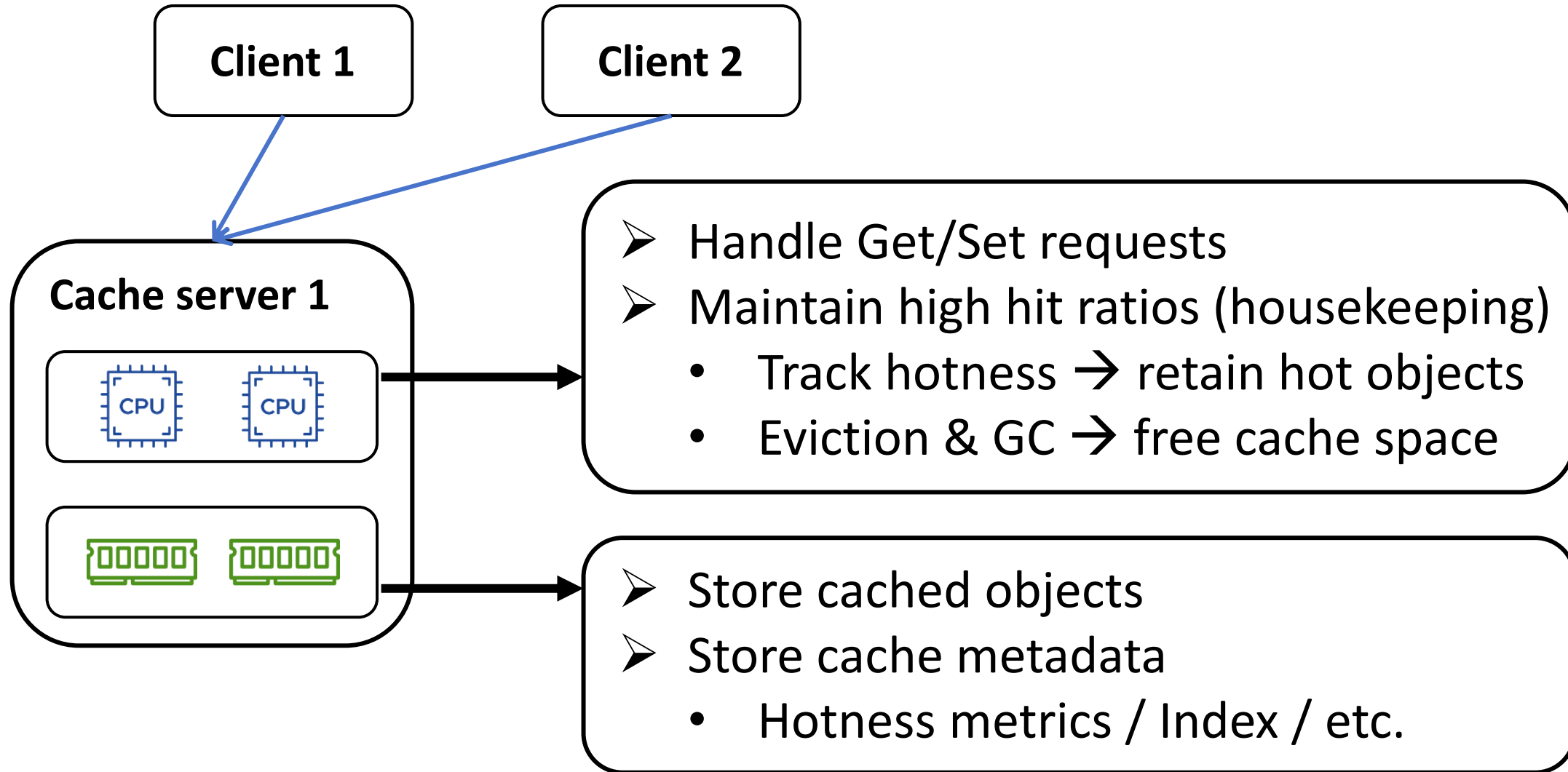
Serverless
Computing

...

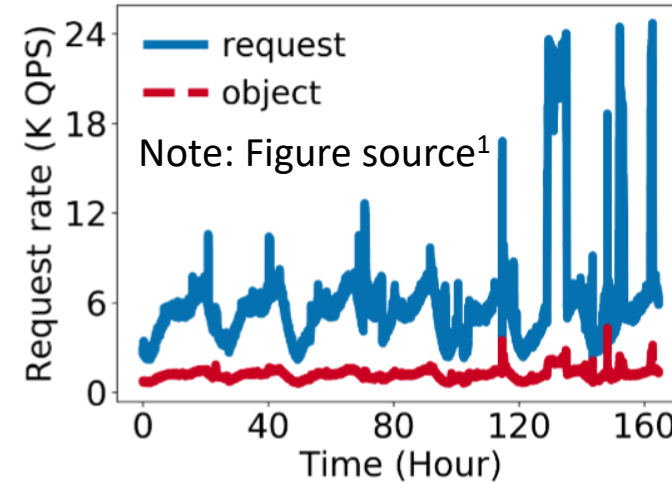
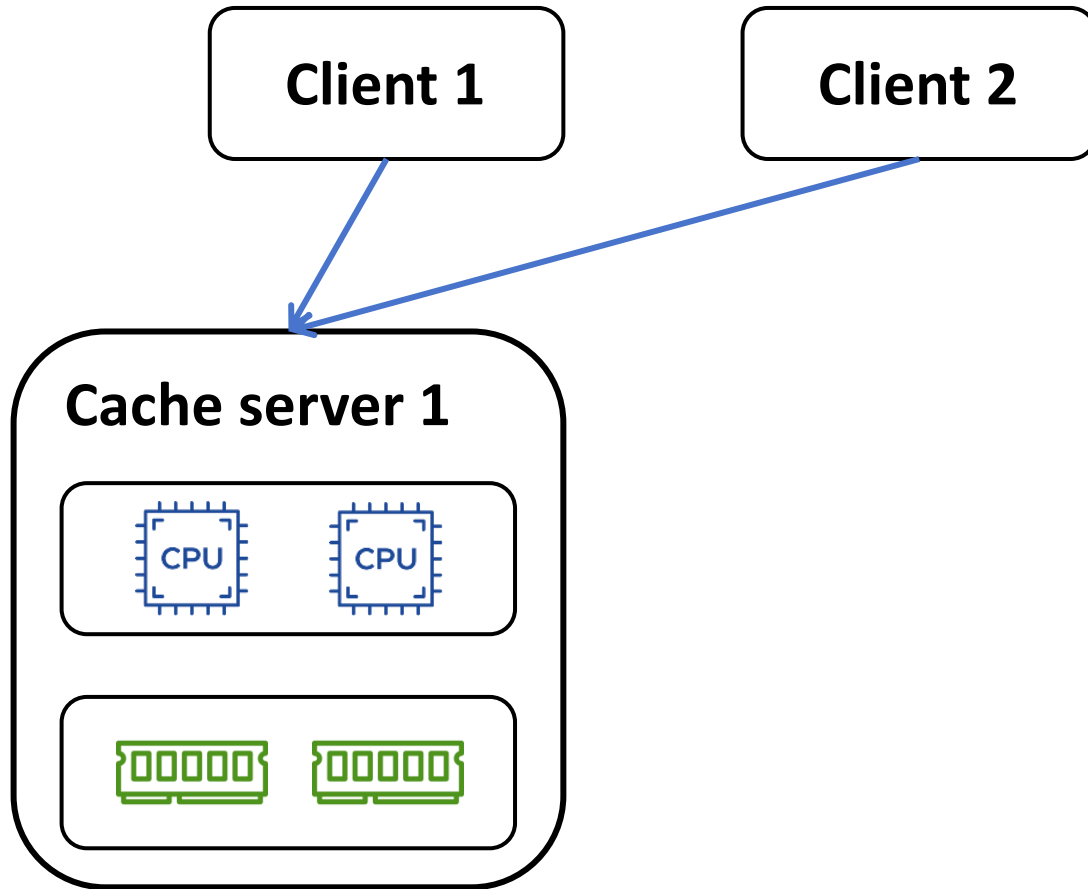
Caching Systems on Monolithic Servers



Caching Systems on Monolithic Servers

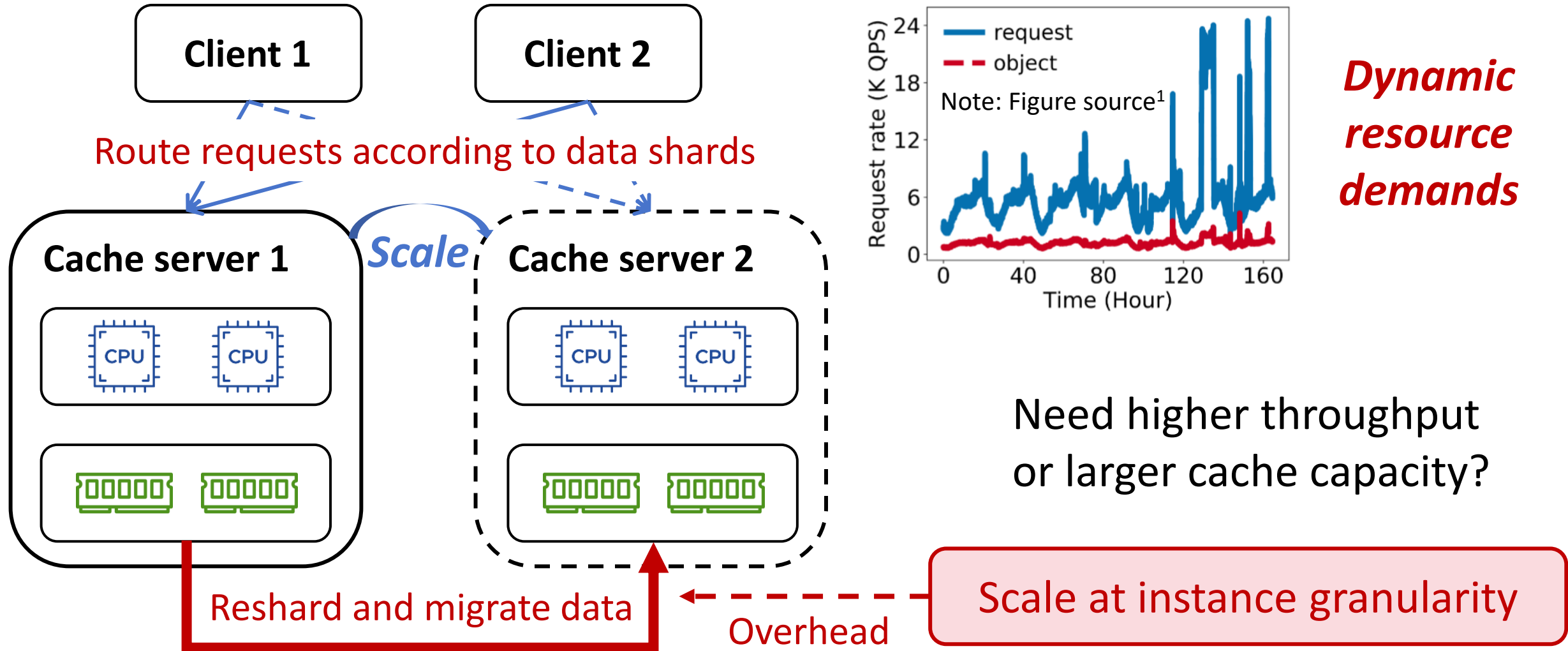


Scaling Monolithic Caching Systems



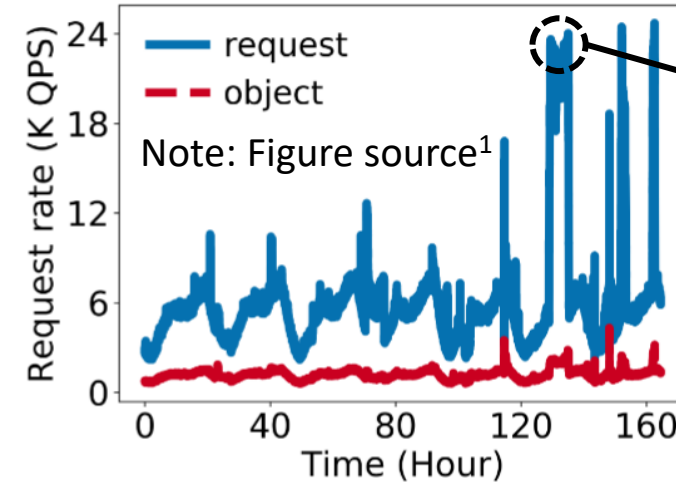
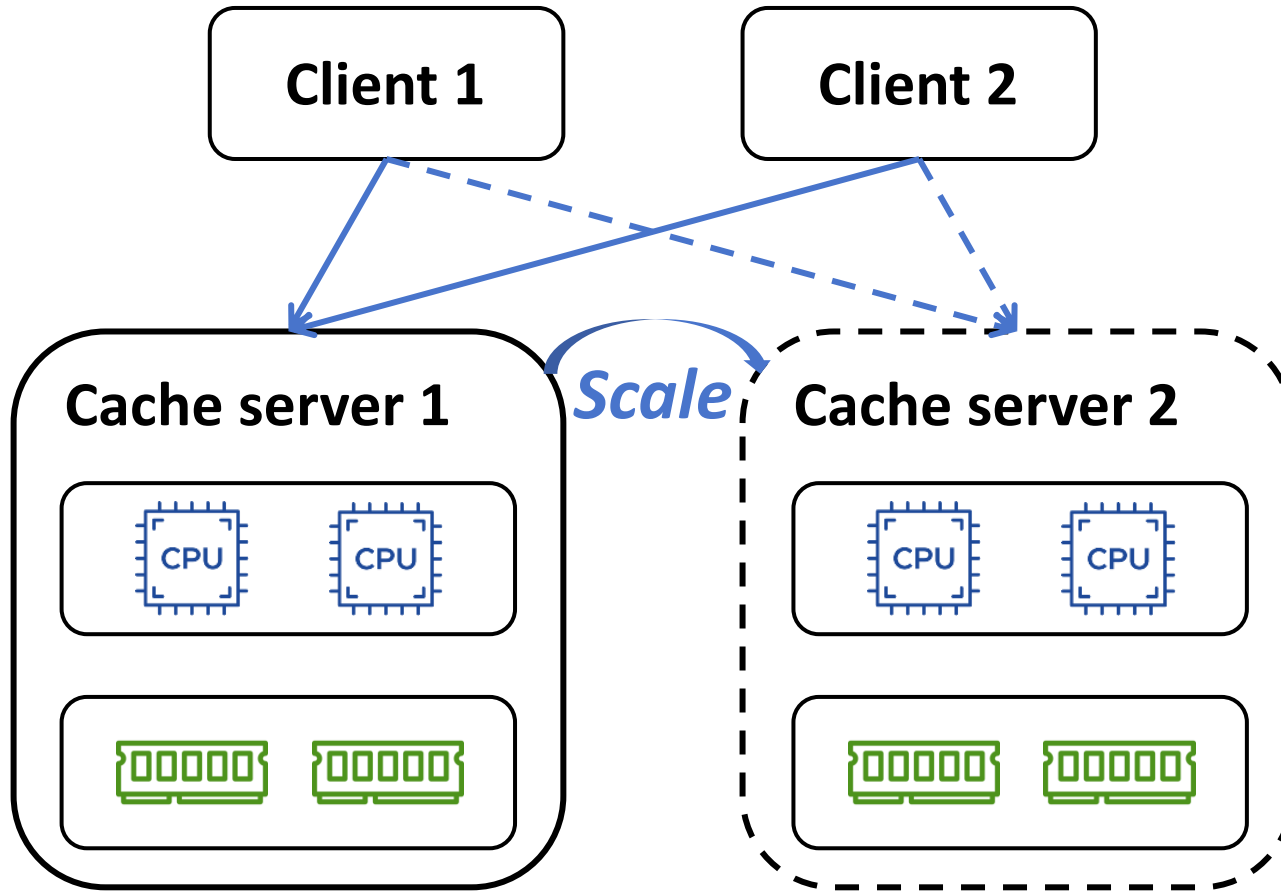
*Dynamic
resource
demands*

Scaling Monolithic Caching Systems



[1] Juncheng Yang et al. A large scale analysis of hundreds of in-memory cache clusters at Twitter. OSDI 2020, Fig. 4.

Scaling Monolithic Caching Systems



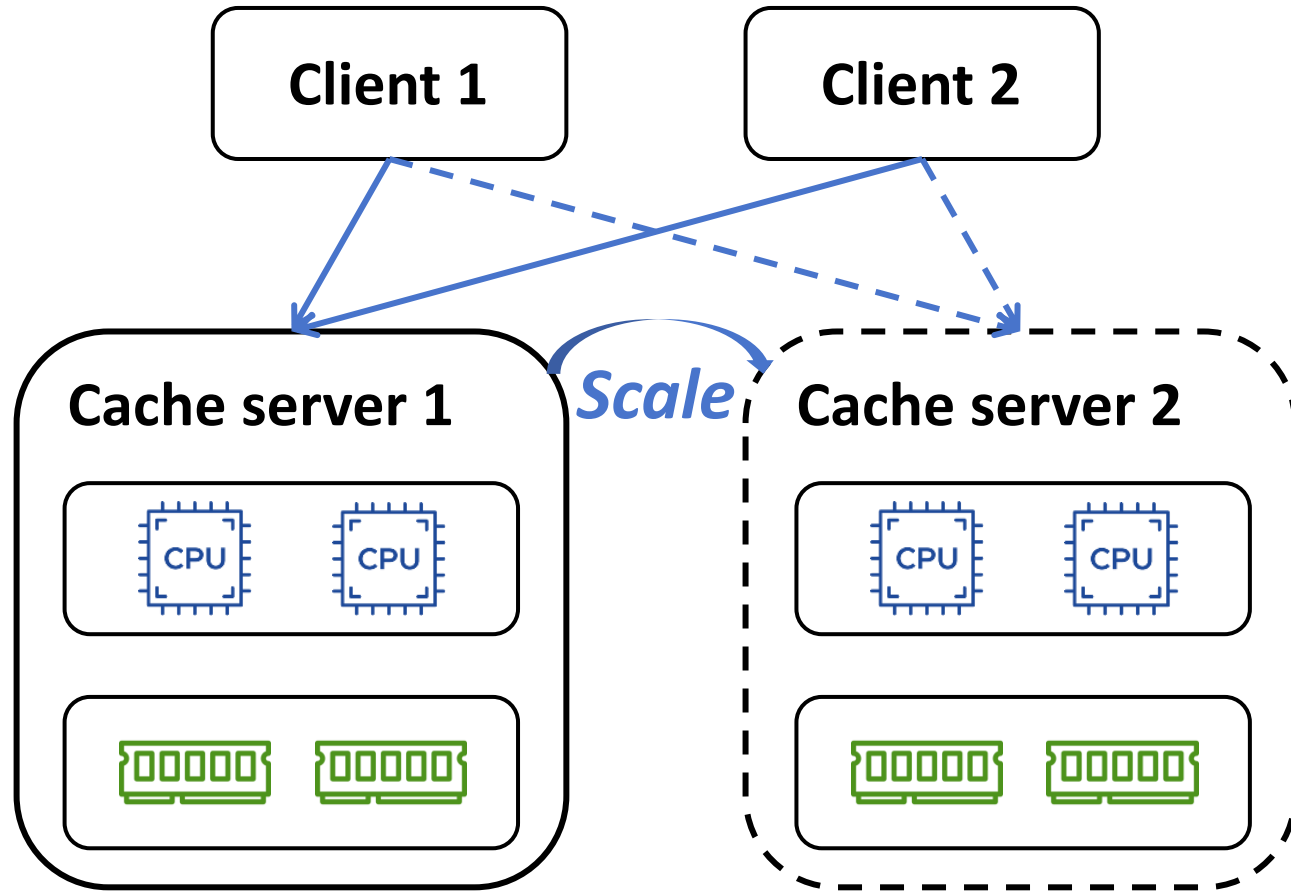
1 Need higher throughput

1 Need more CPUs

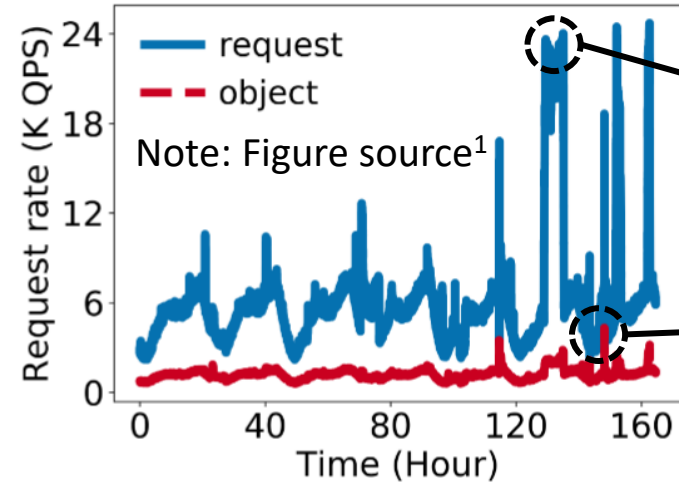
Monolithic scaling adds both CPUs & memory

Waste memory

Scaling Monolithic Caching Systems



Monolithic servers **prevent independent scaling** of CPUs and memory, causing **resource underutilization**



① Need higher throughput

② Need larger cache size

① Need more CPUs

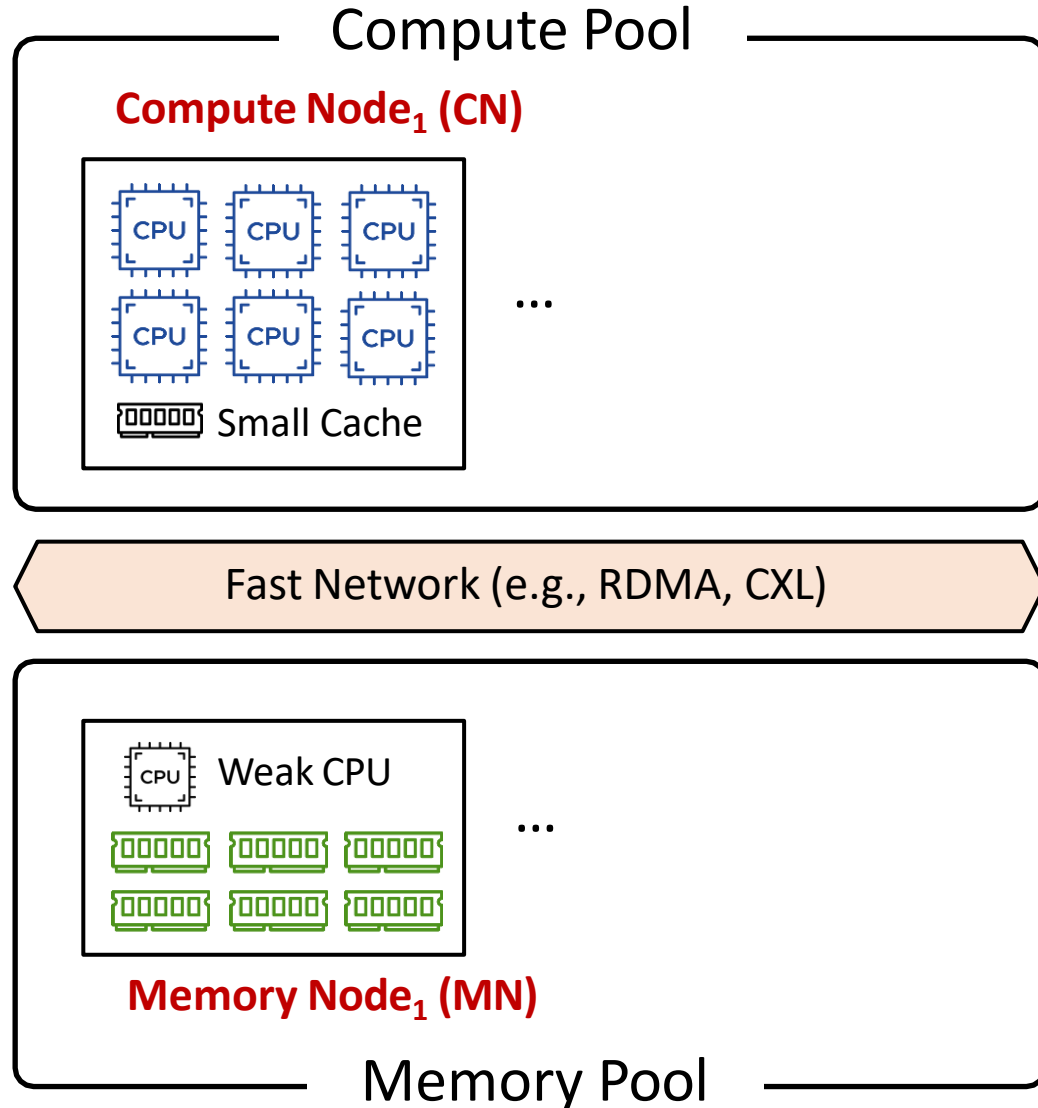
② Need more memory

Monolithic scaling adds both CPUs & memory

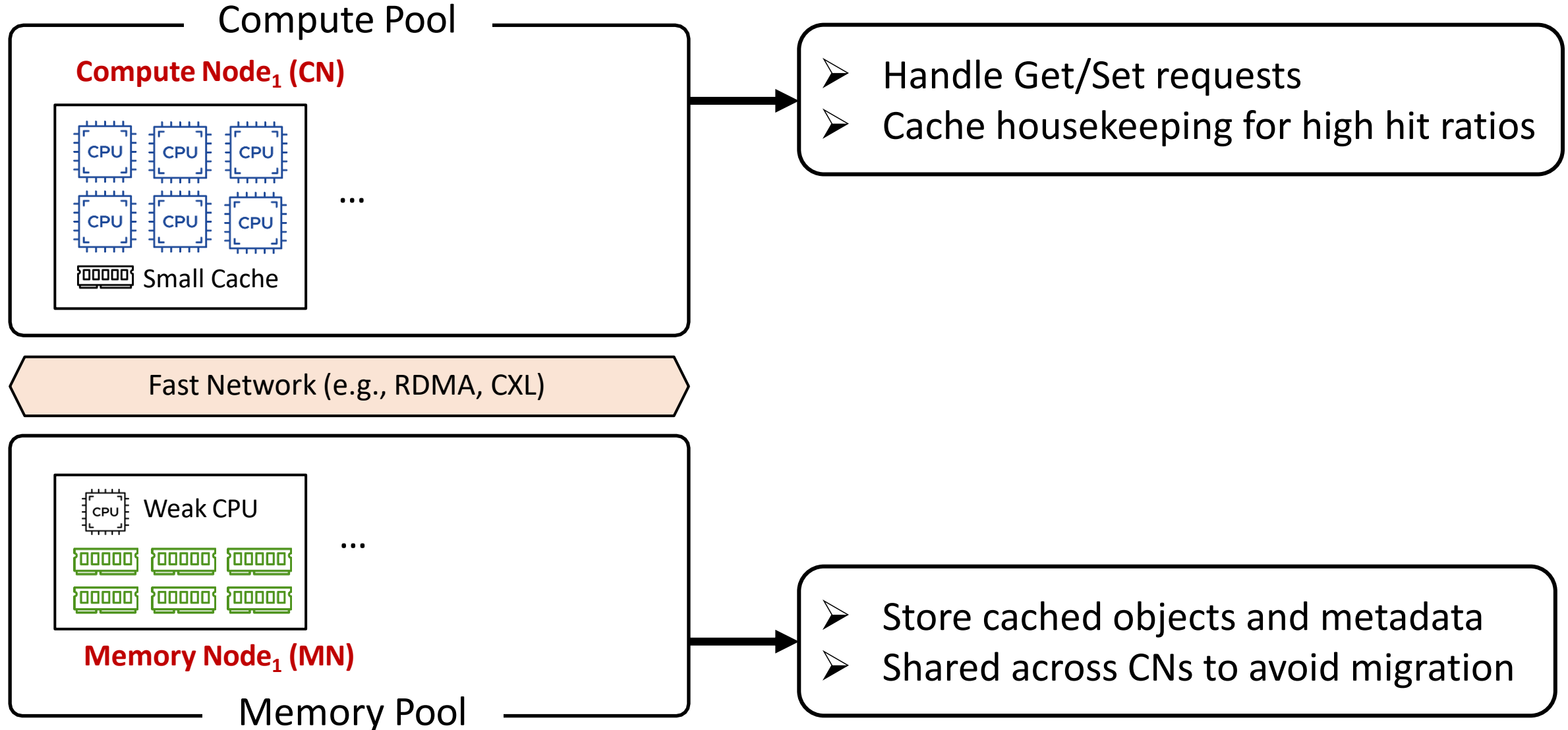
Waste memory

Waste CPUs

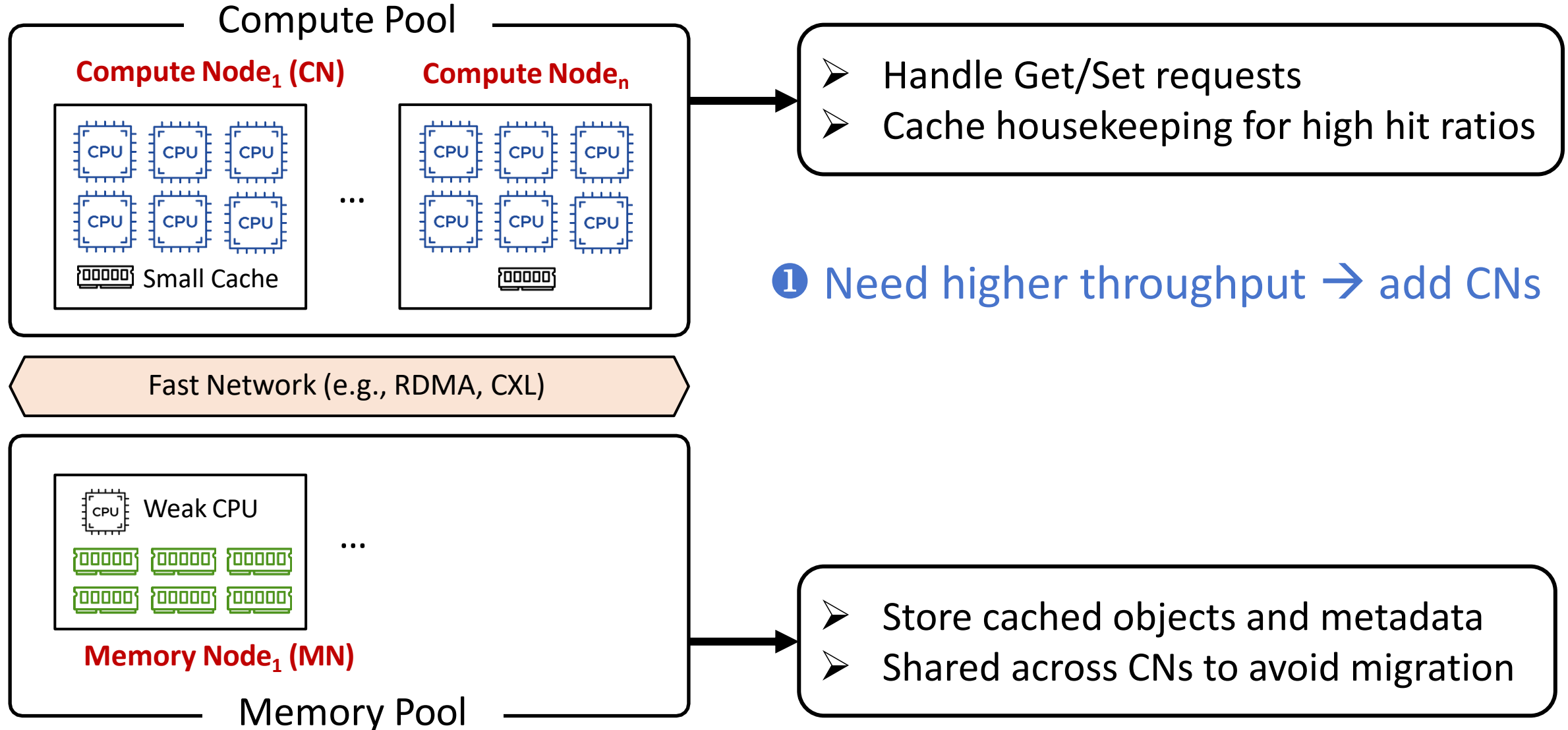
Disaggregated Memory (DM) for Caching



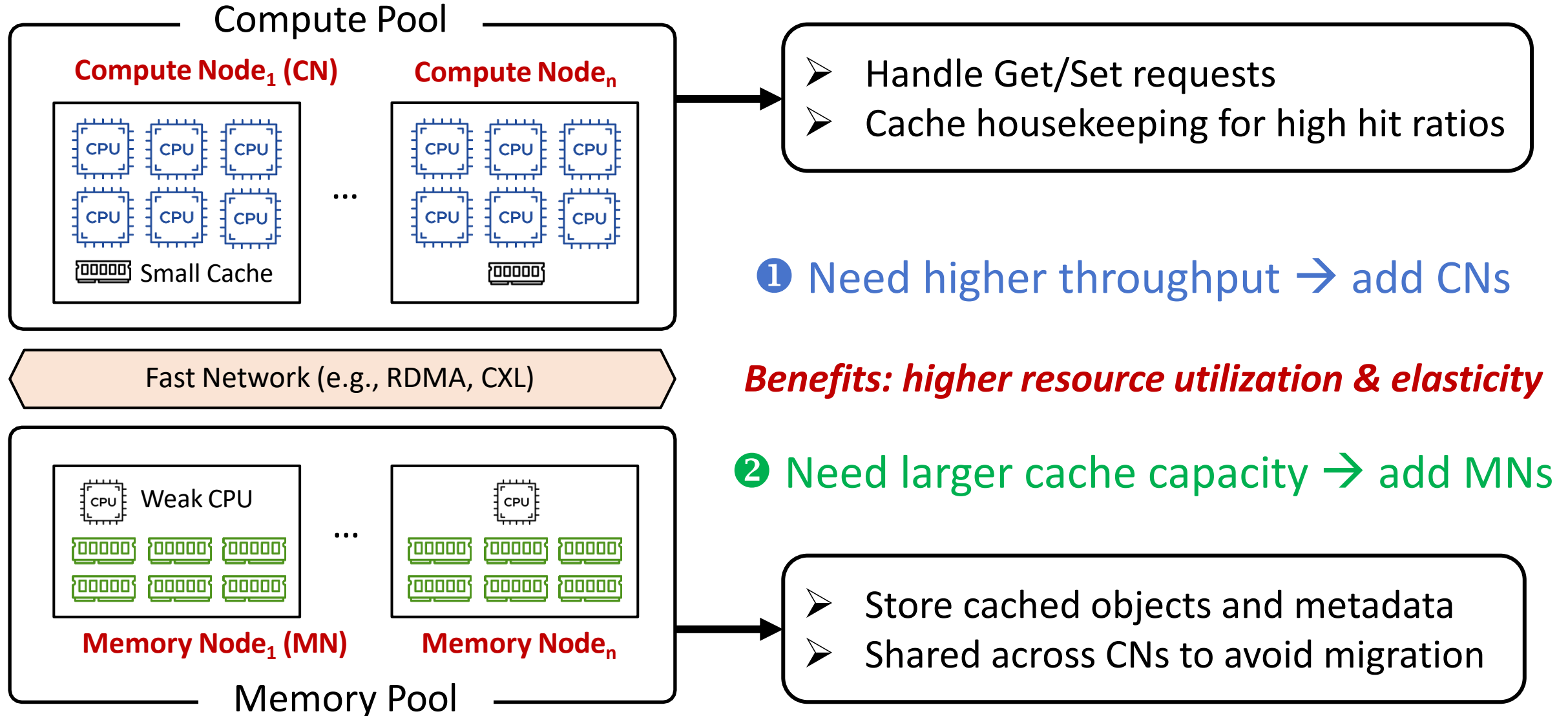
Disaggregated Memory (DM) for Caching



Disaggregated Memory (DM) for Caching



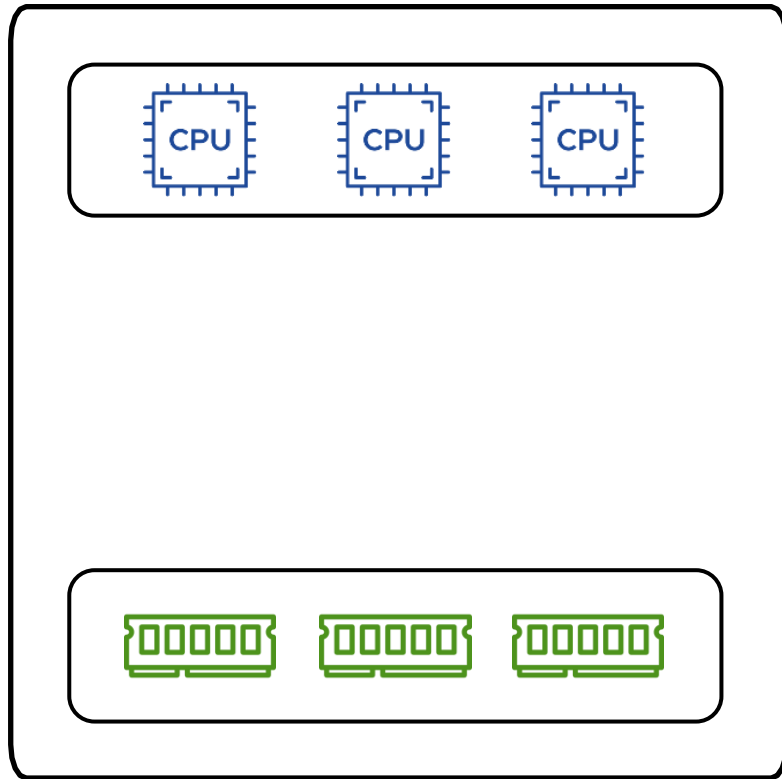
Disaggregated Memory (DM) for Caching



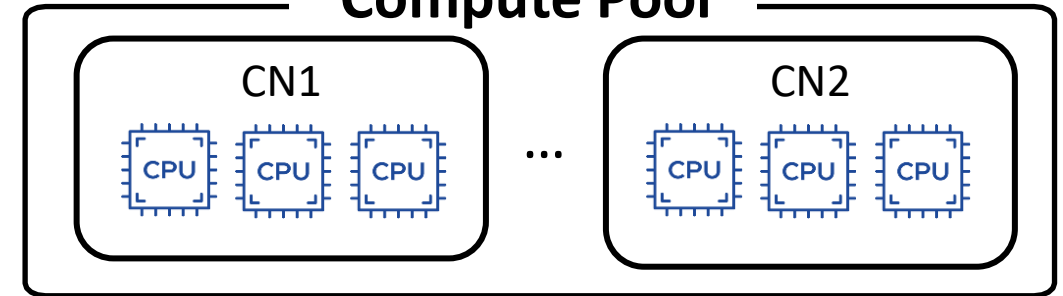
The Cost: Synchronization Amplification

Cache housekeeping: Hotness Tracking / Eviction Coordination / Garbage Collection

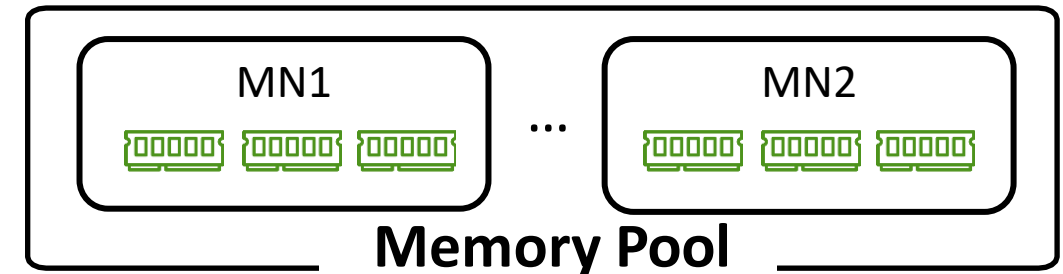
Monolithic cache server



Compute Pool



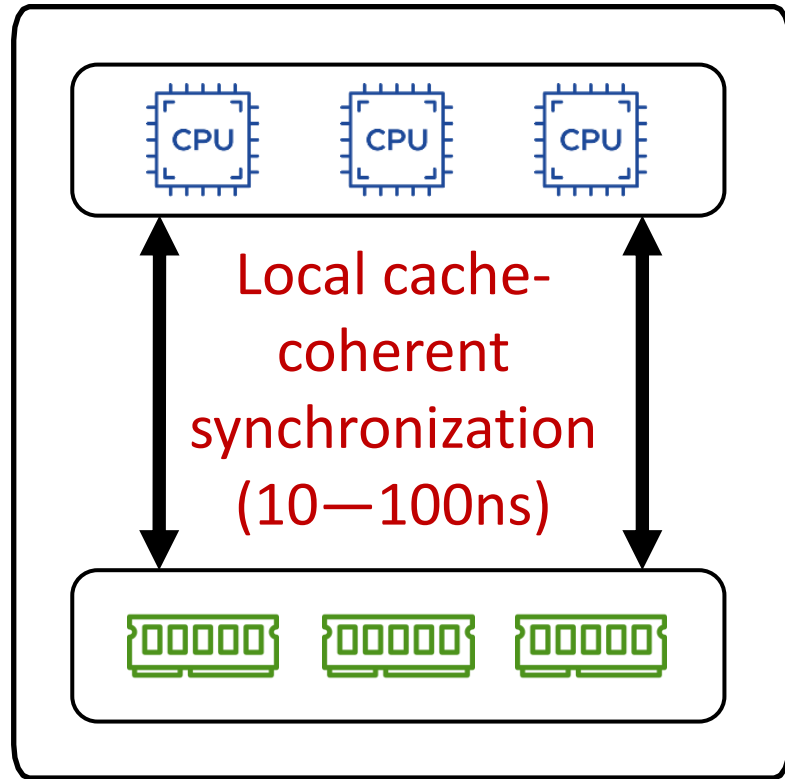
Memory Pool



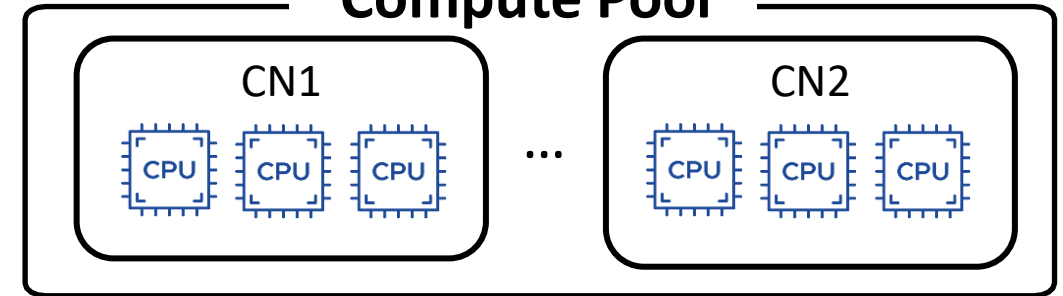
The Cost: Synchronization Amplification

Cache housekeeping: Hotness Tracking / Eviction Coordination / Garbage Collection

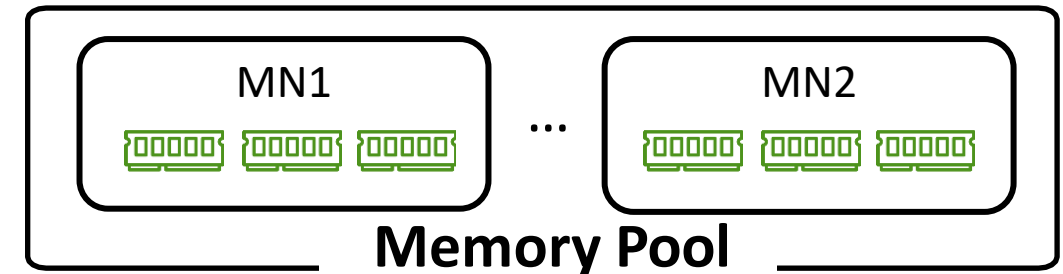
Monolithic cache server



Compute Pool

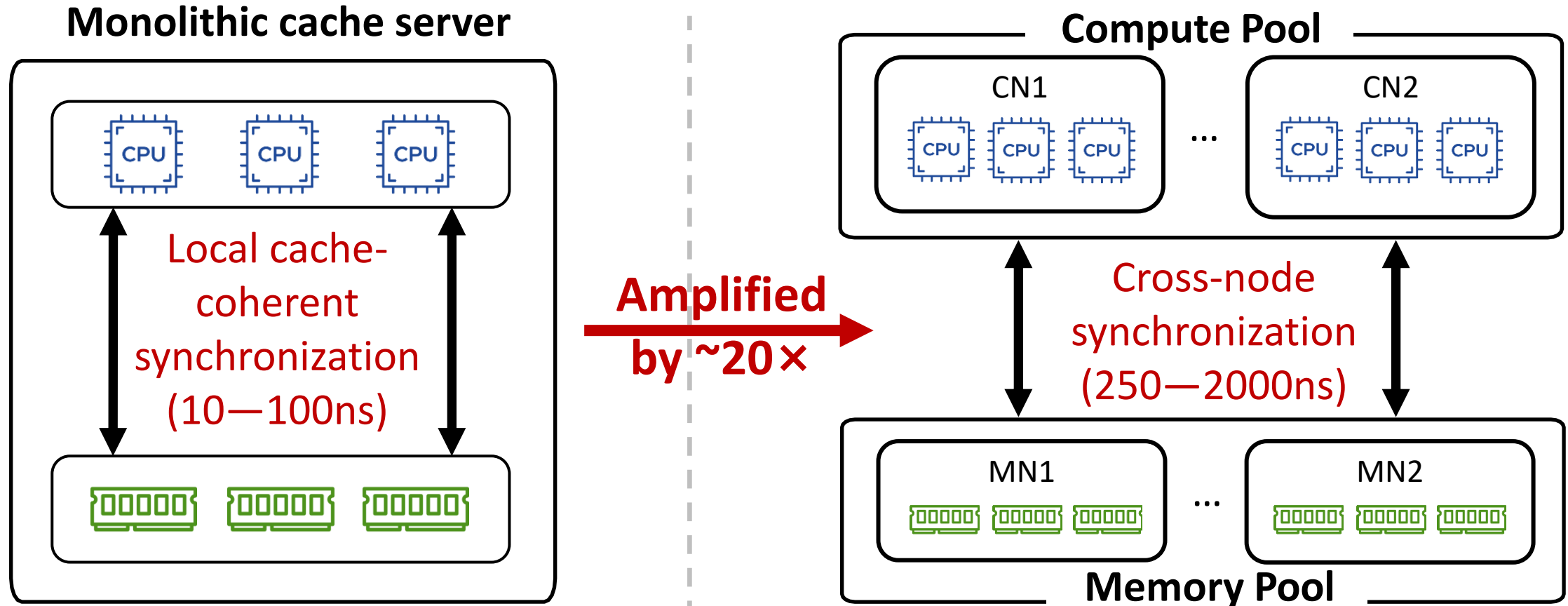


Memory Pool



The Cost: Synchronization Amplification

Cache housekeeping: Hotness Tracking / Eviction Coordination / Garbage Collection

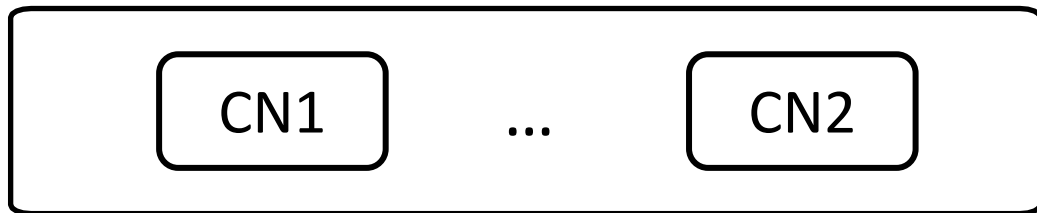


DM turns lightweight housekeeping into expensive cross-node synchronization.

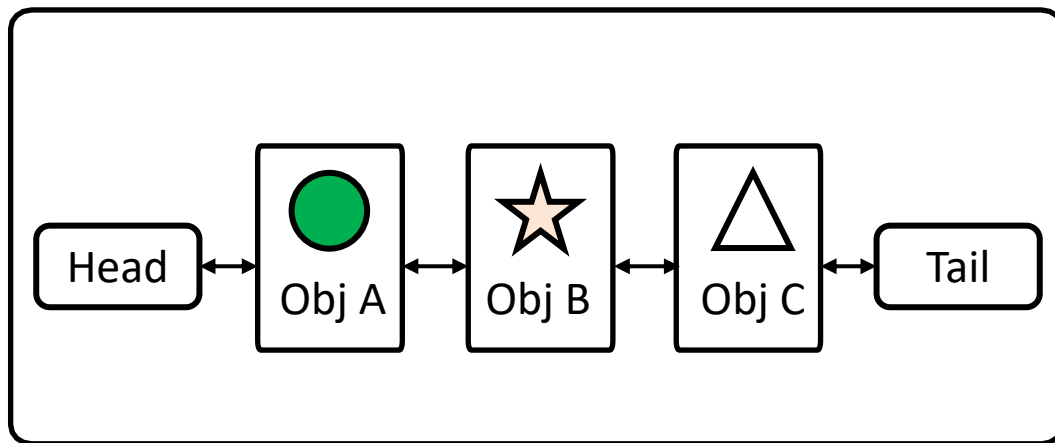
Why DM Cache Housekeeping Is Costly

Compute Pool

Queue-based scheme

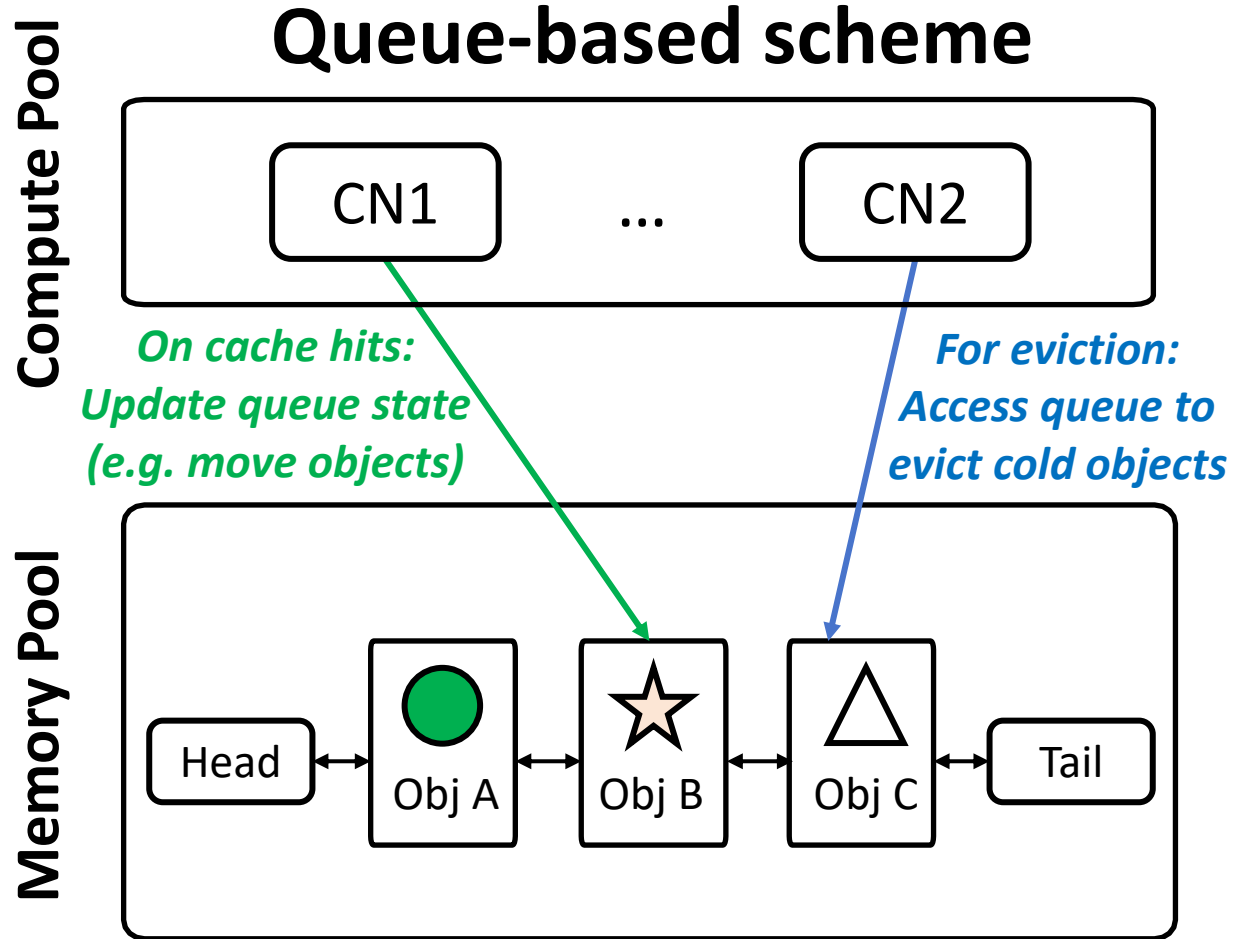


Memory Pool



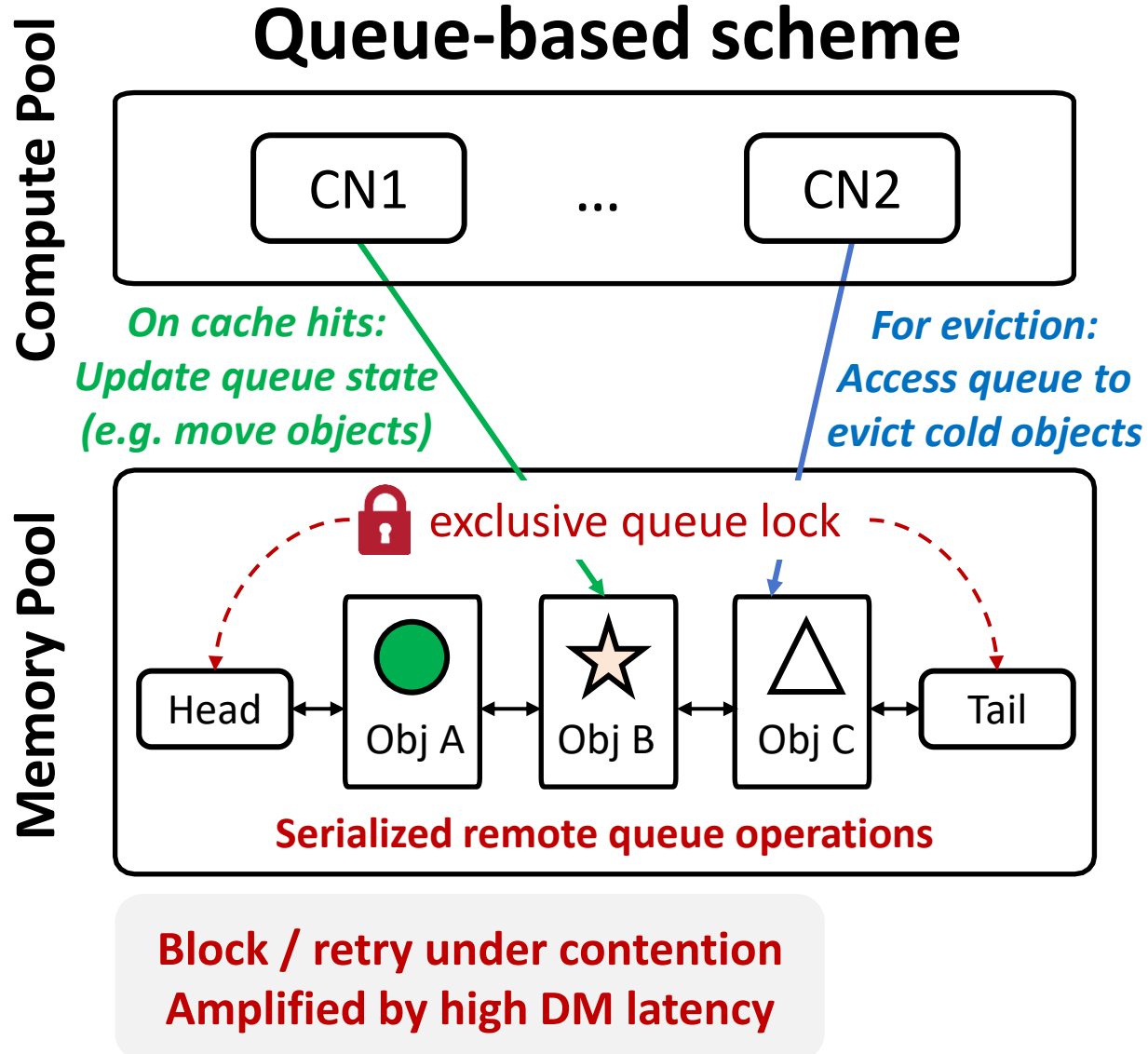
Sampling-based scheme

Why DM Cache Housekeeping Is Costly



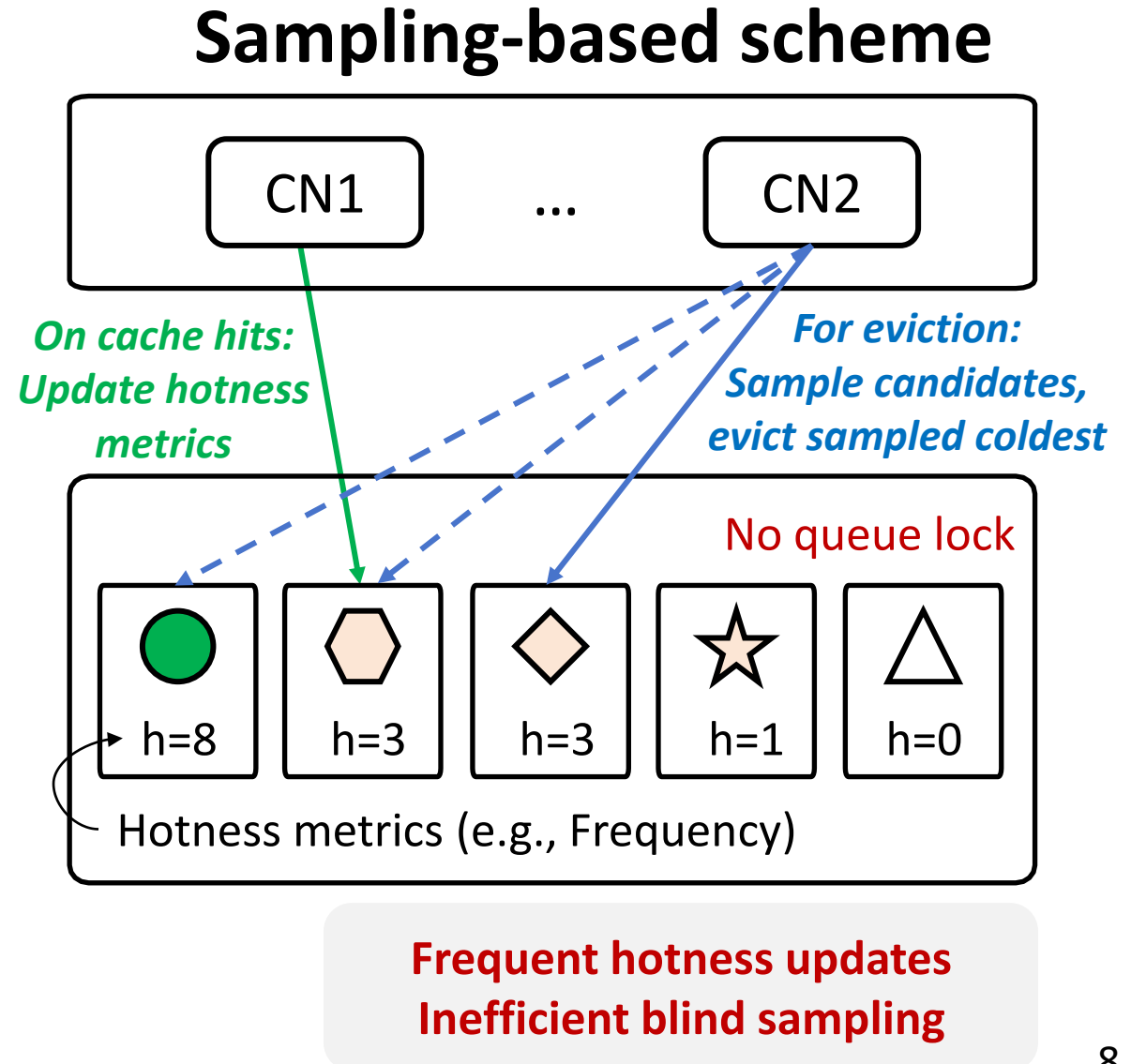
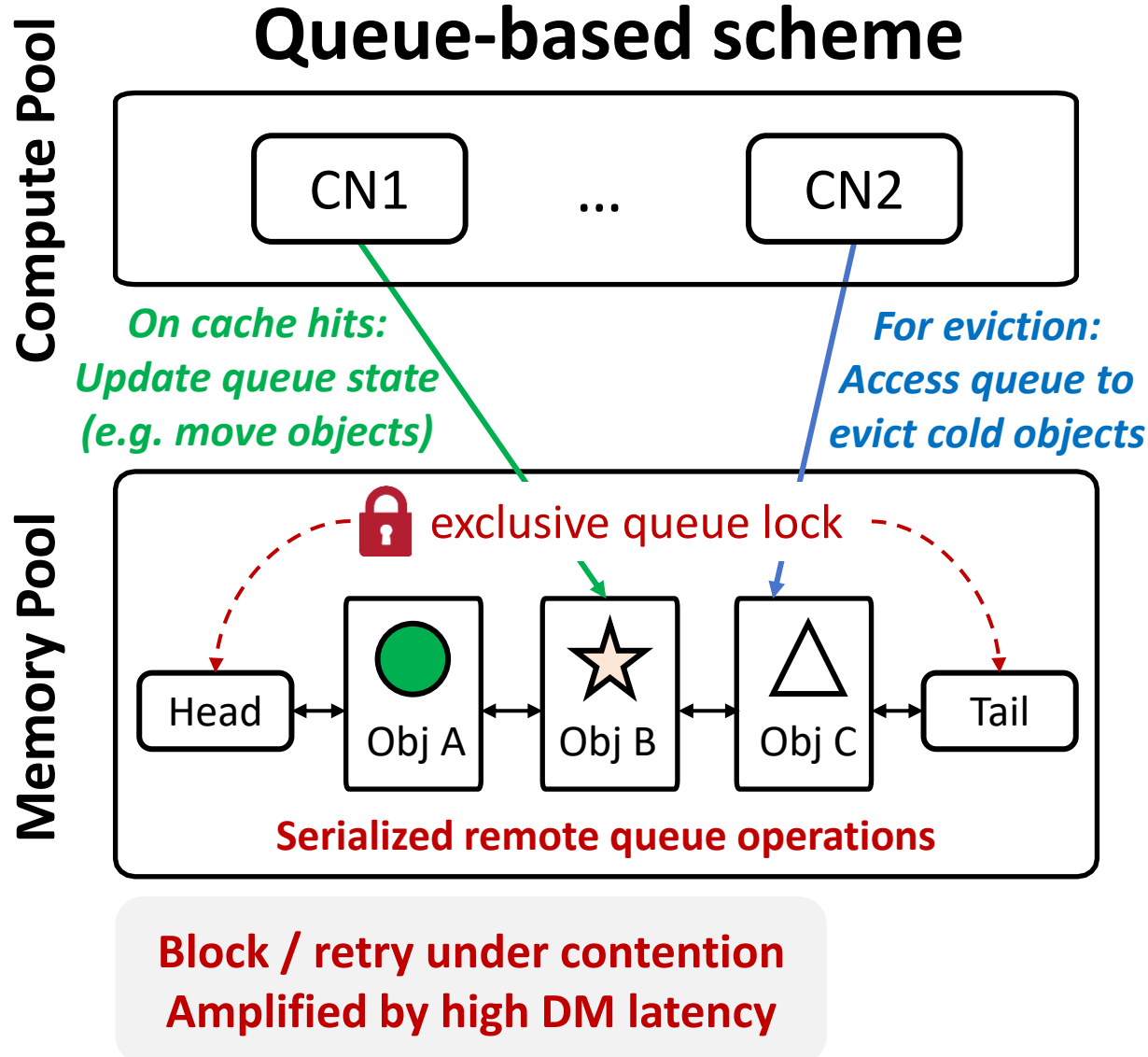
Sampling-based scheme

Why DM Cache Housekeeping Is Costly

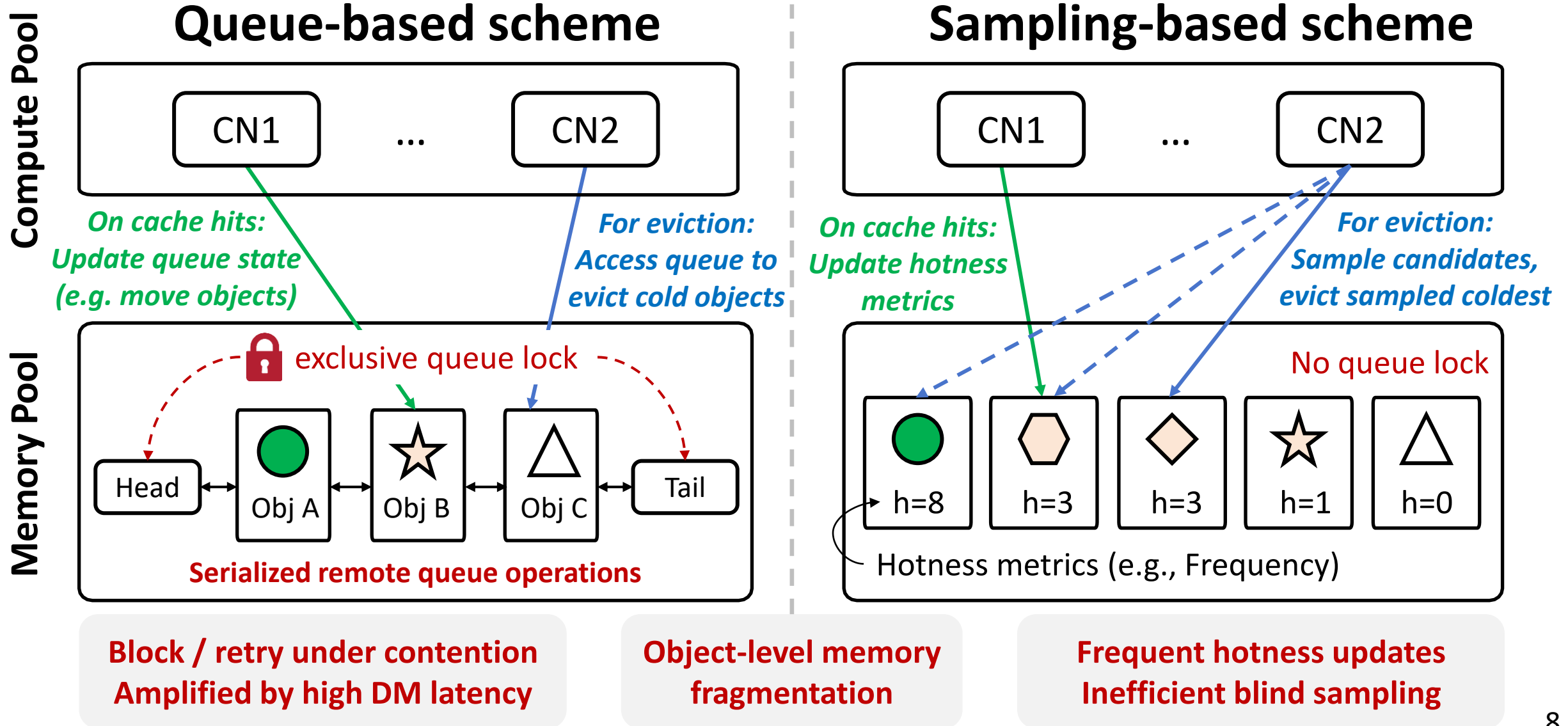


Sampling-based scheme

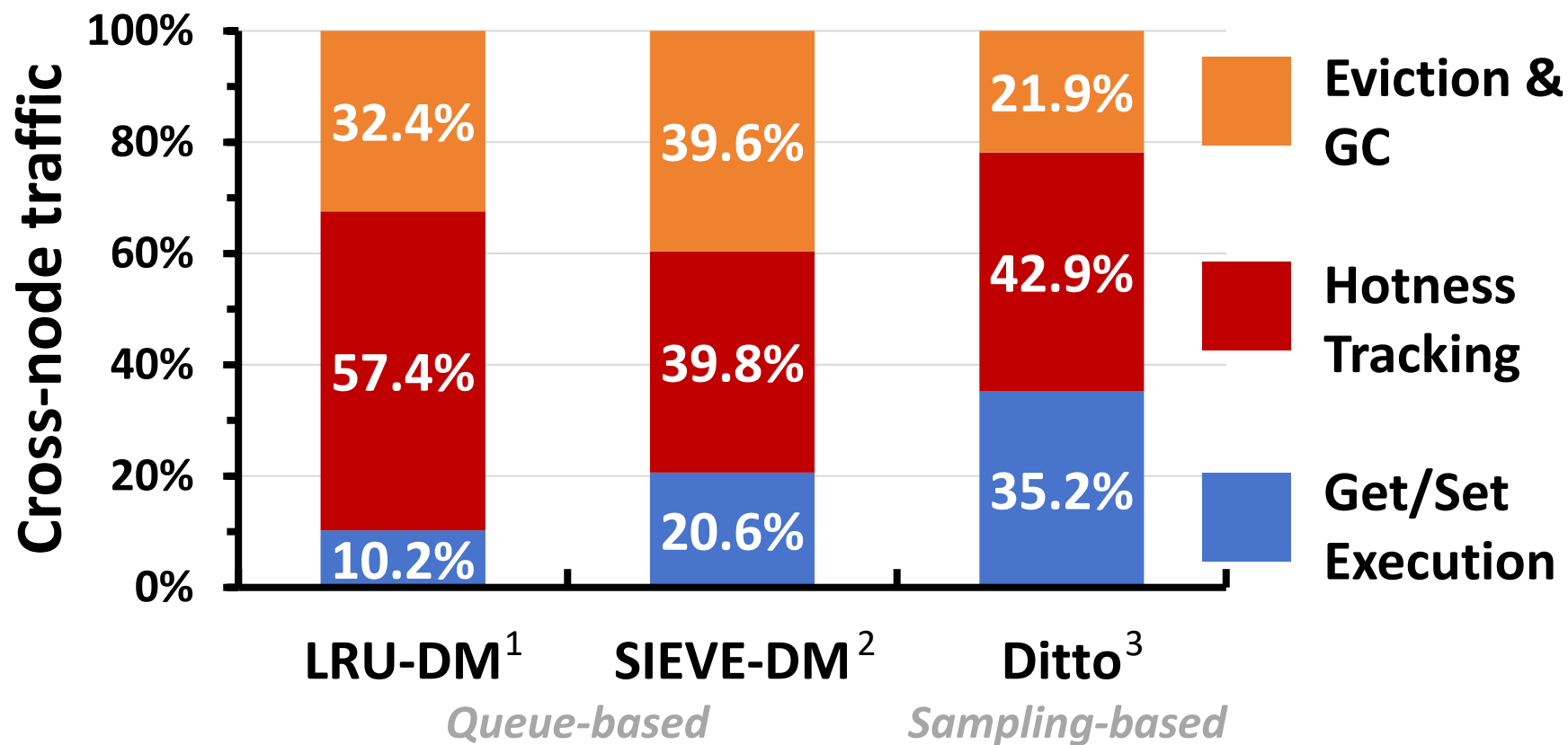
Why DM Cache Housekeeping Is Costly



Why DM Cache Housekeeping Is Costly



Synchronization Amplification is Severe

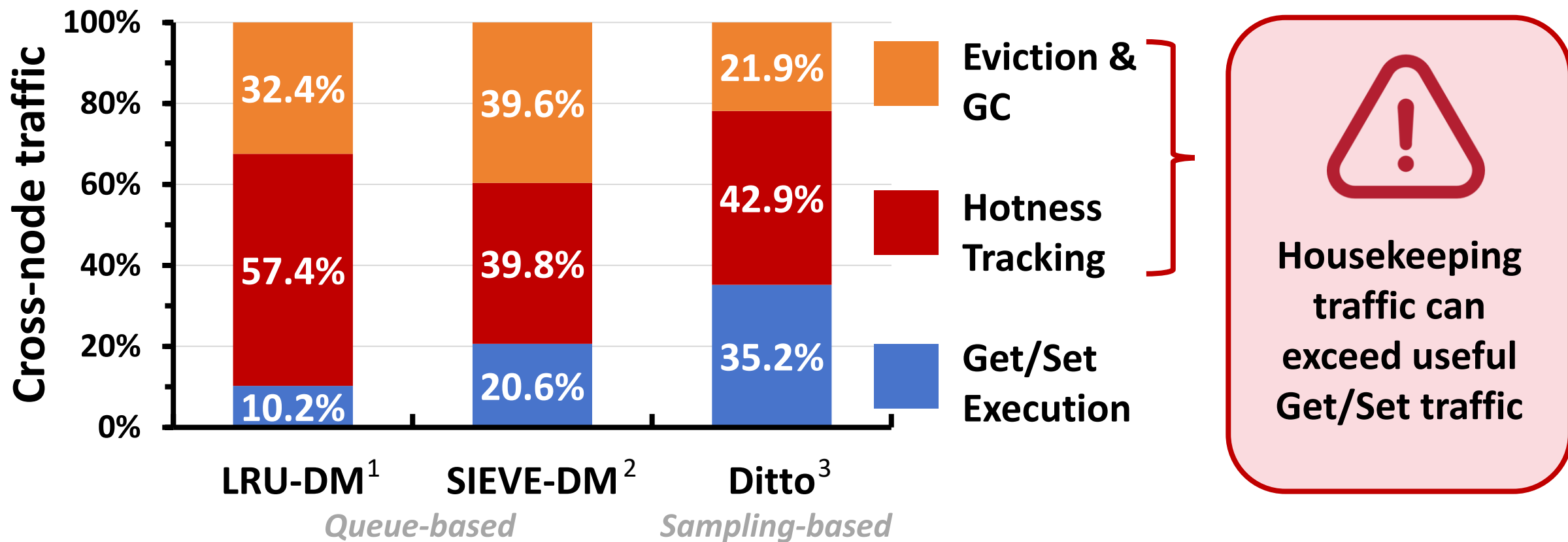


[1] LRU-DM and SIEVE-DM are RDMA-based DM adaptations of LRU and SIEVE. Setup: Twitter trace, 10% cache ratio, 16 CN threads.

[2] Yazhuo Zhang et al. SIEVE is simpler than LRU: An efficient turn-key eviction algorithm for web caches. NSDI 2024.

[3] Jiacheng Shen et al. Ditto: An elastic and adaptive memory-disaggregated caching system. SOSP 2023.

Synchronization Amplification is Severe



How can we redesign DM cache housekeeping for efficient synchronization?

[1] LRU-DM and SIEVE-DM are RDMA-based DM adaptations of LRU and SIEVE. Setup: Twitter trace, 10% cache ratio, 16 CN threads.

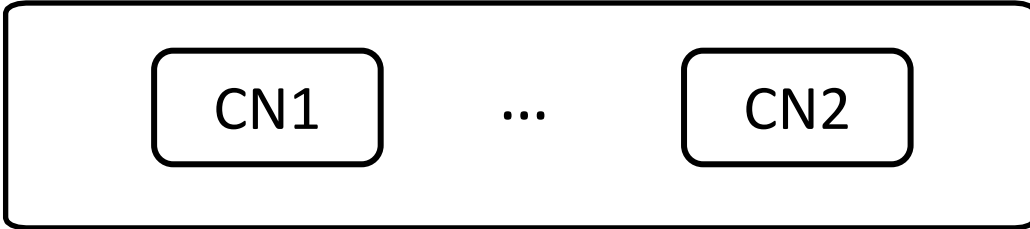
[2] Yazhuo Zhang et al. SIEVE is simpler than LRU: An efficient turn-key eviction algorithm for web caches. NSDI 2024.

[3] Jiacheng Shen et al. Ditto: An elastic and adaptive memory-disaggregated caching system. SOSP 2023.

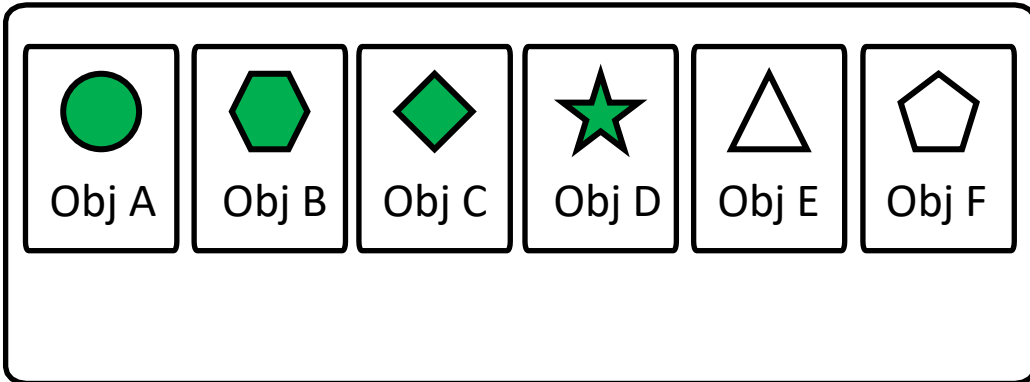
C1: Efficiency-Effectiveness Trade-off in Eviction

Compute Pool

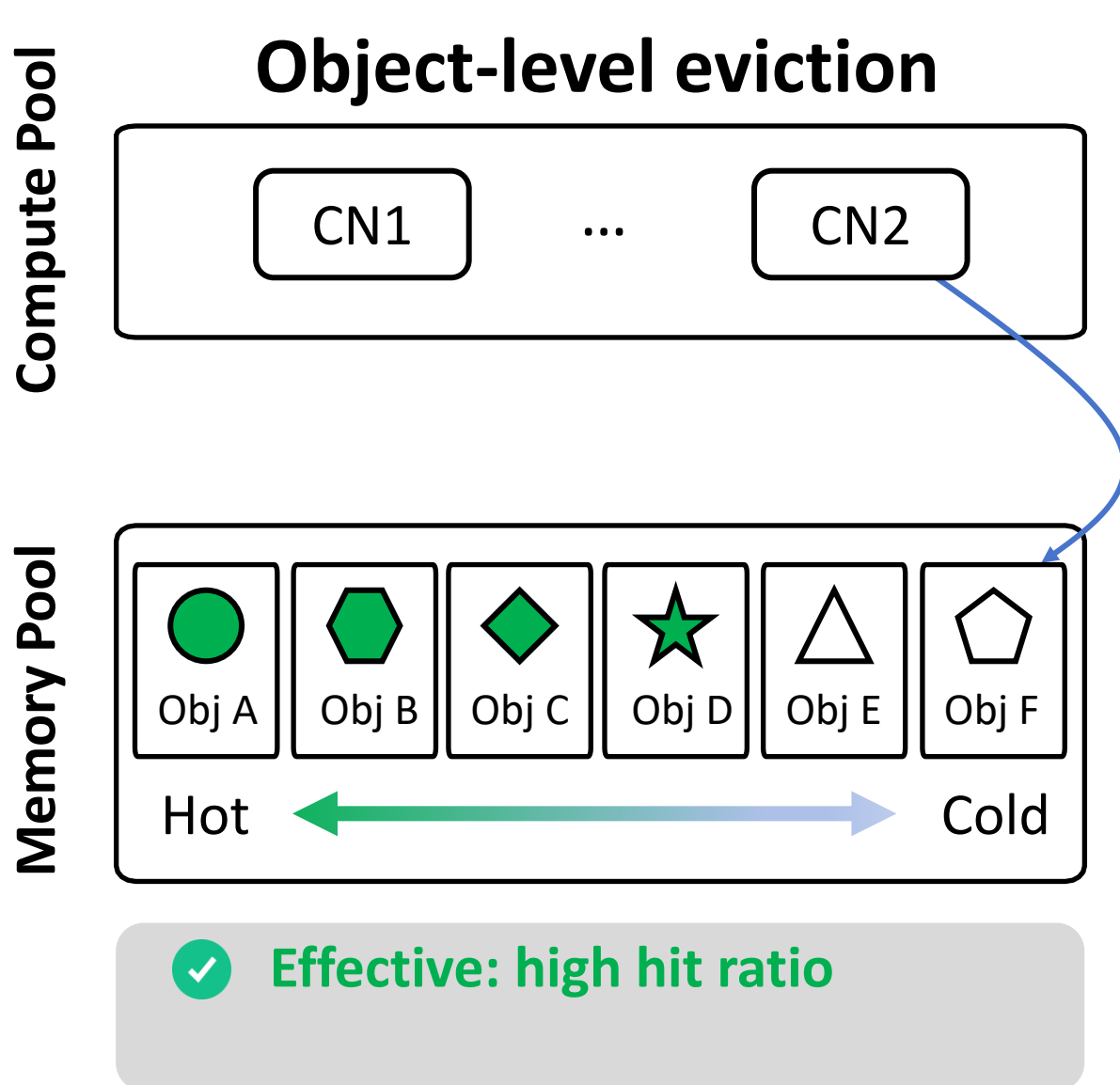
Object-level eviction



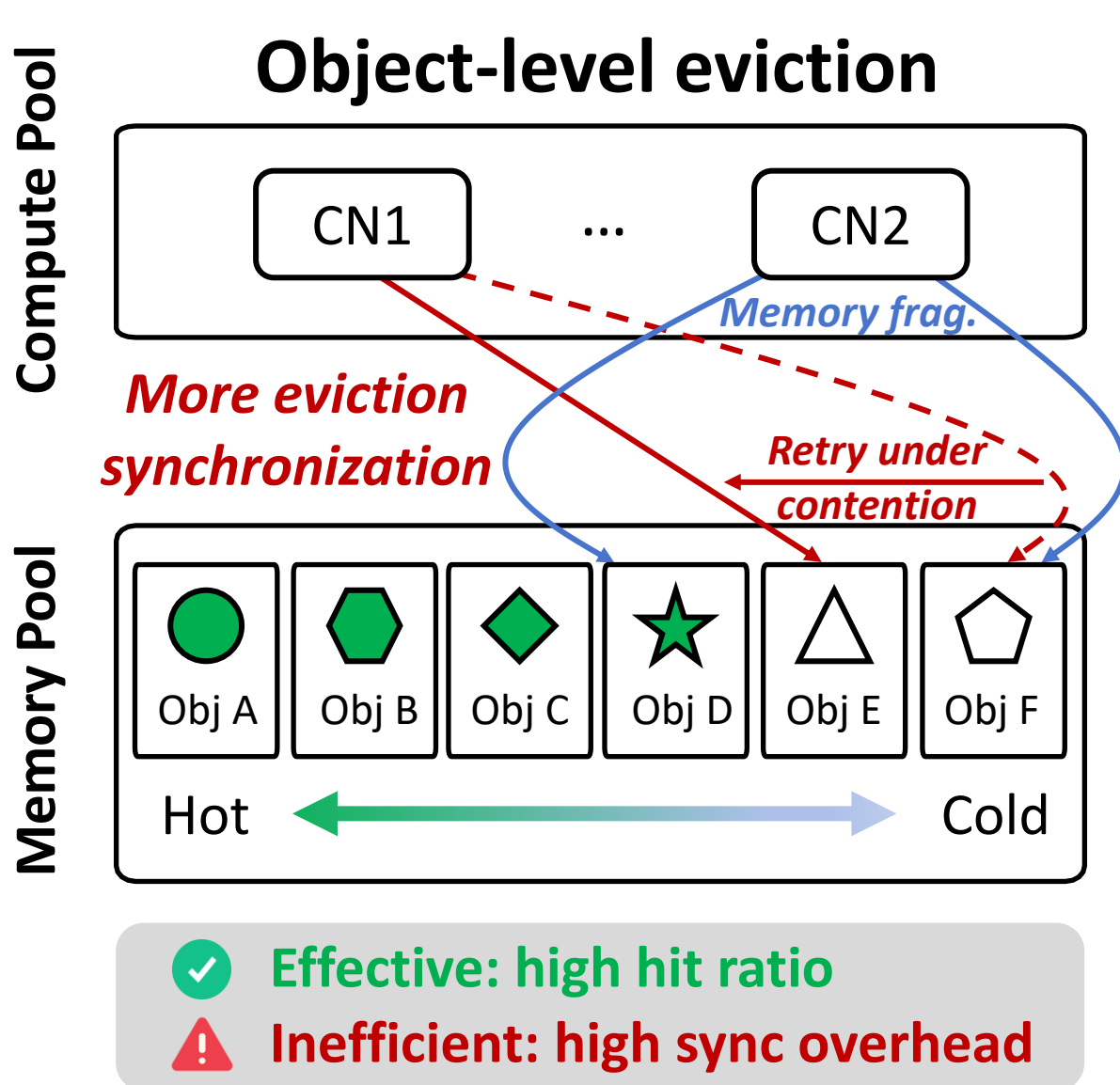
Memory Pool



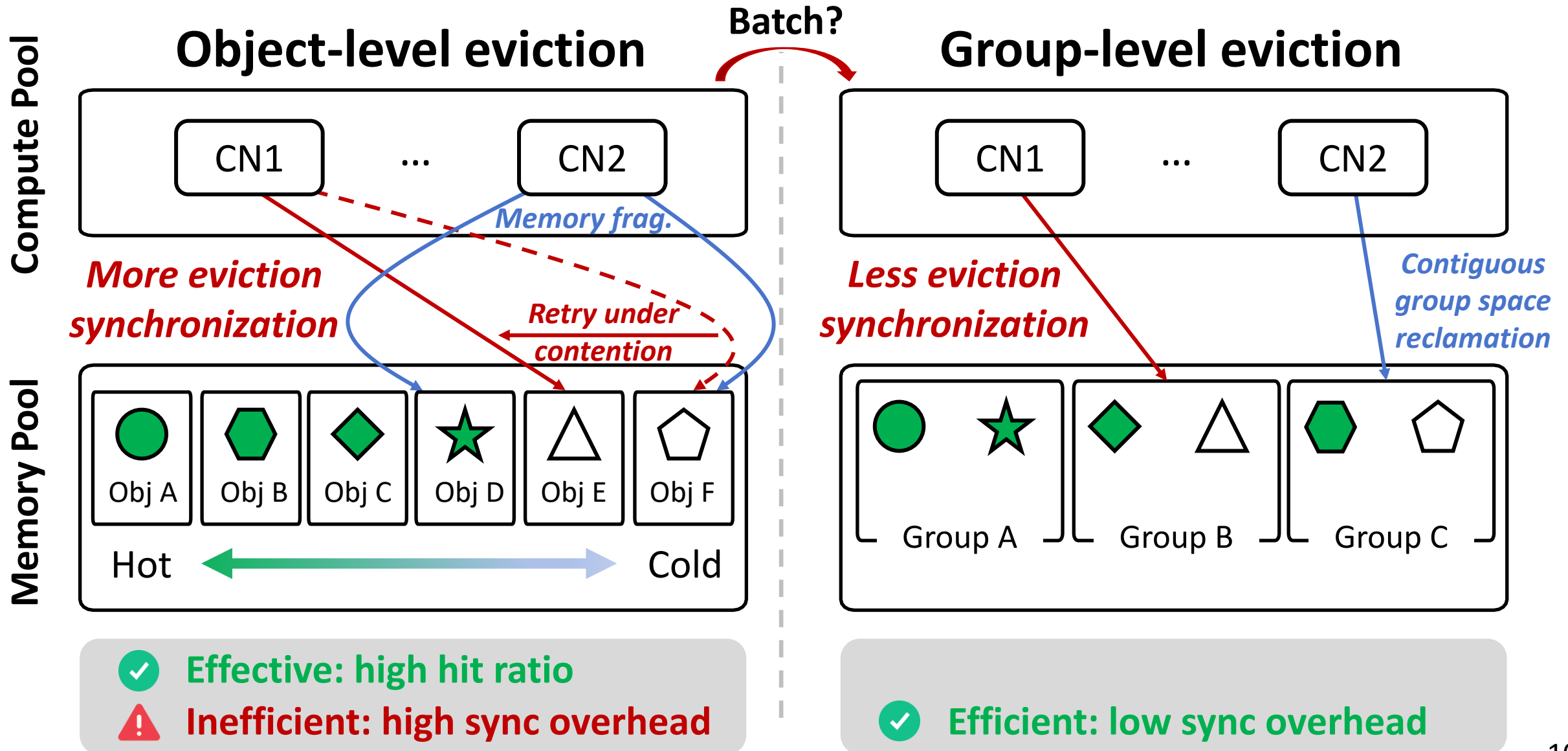
C1: Efficiency-Effectiveness Trade-off in Eviction



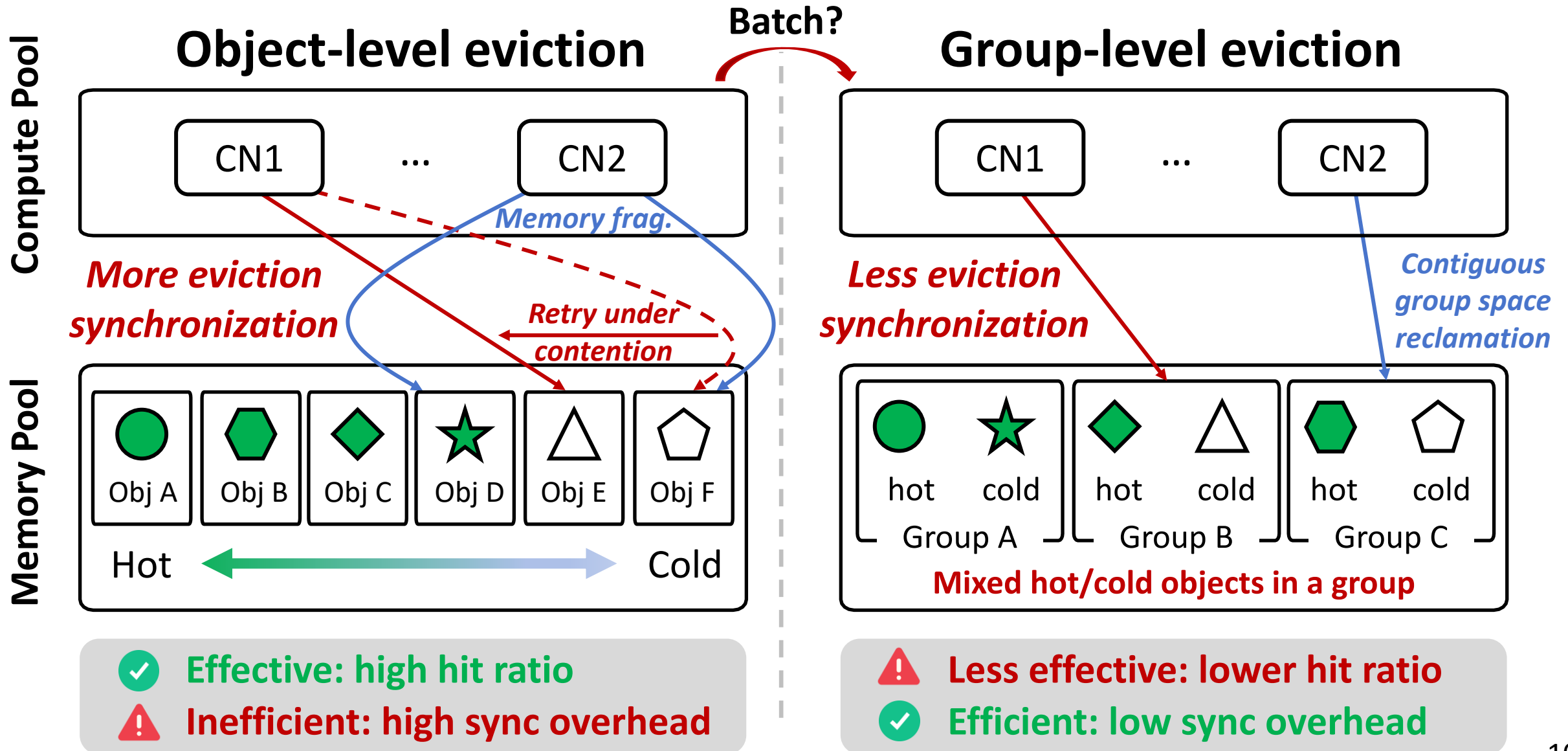
C1: Efficiency-Effectiveness Trade-off in Eviction



C1: Efficiency-Effectiveness Trade-off in Eviction



C1: Efficiency-Effectiveness Trade-off in Eviction



C2: Accuracy–Overhead Dilemma in Hotness Tracking

Compute Pool

Object-level tracking

CN1

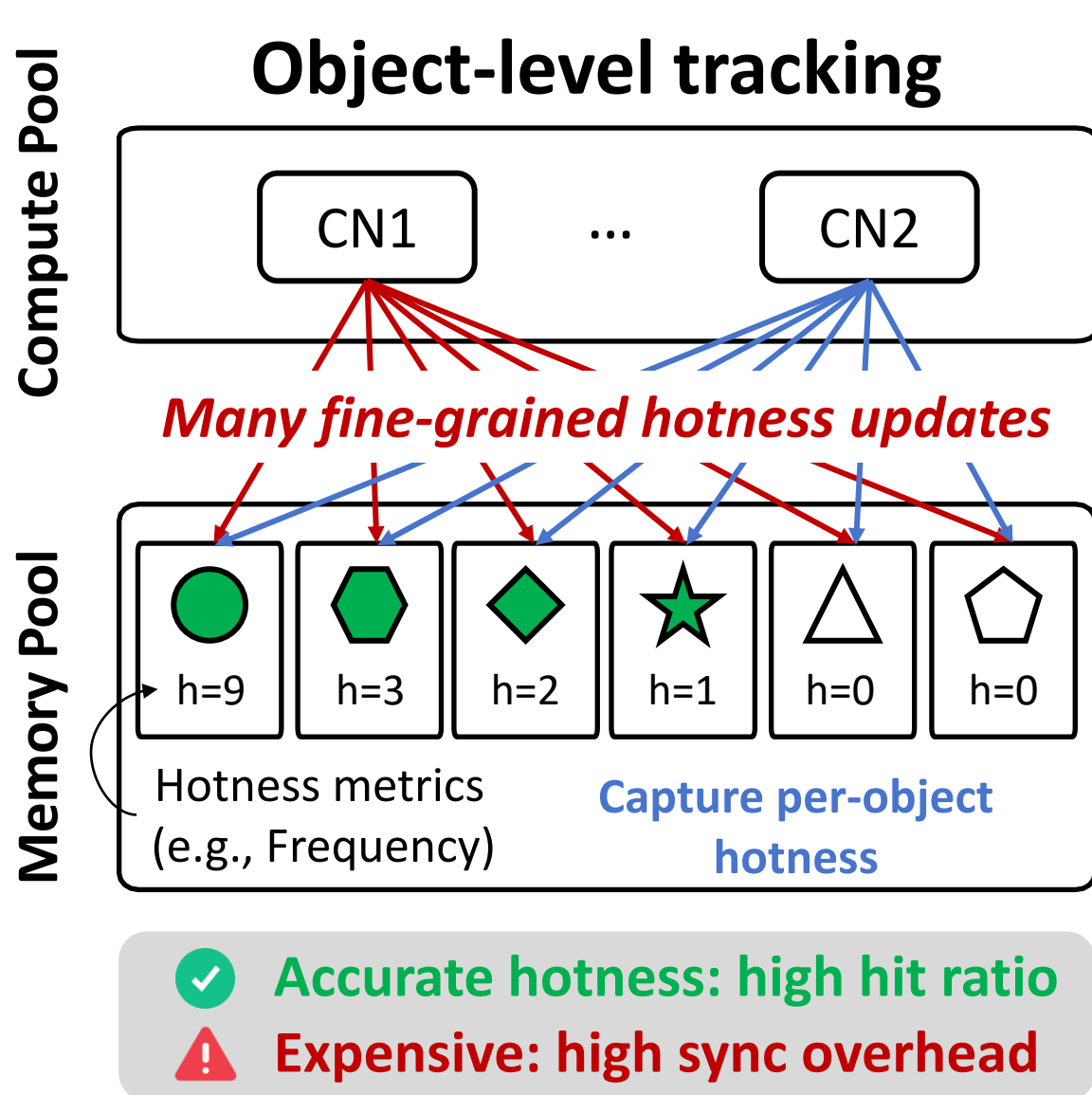
...

CN2

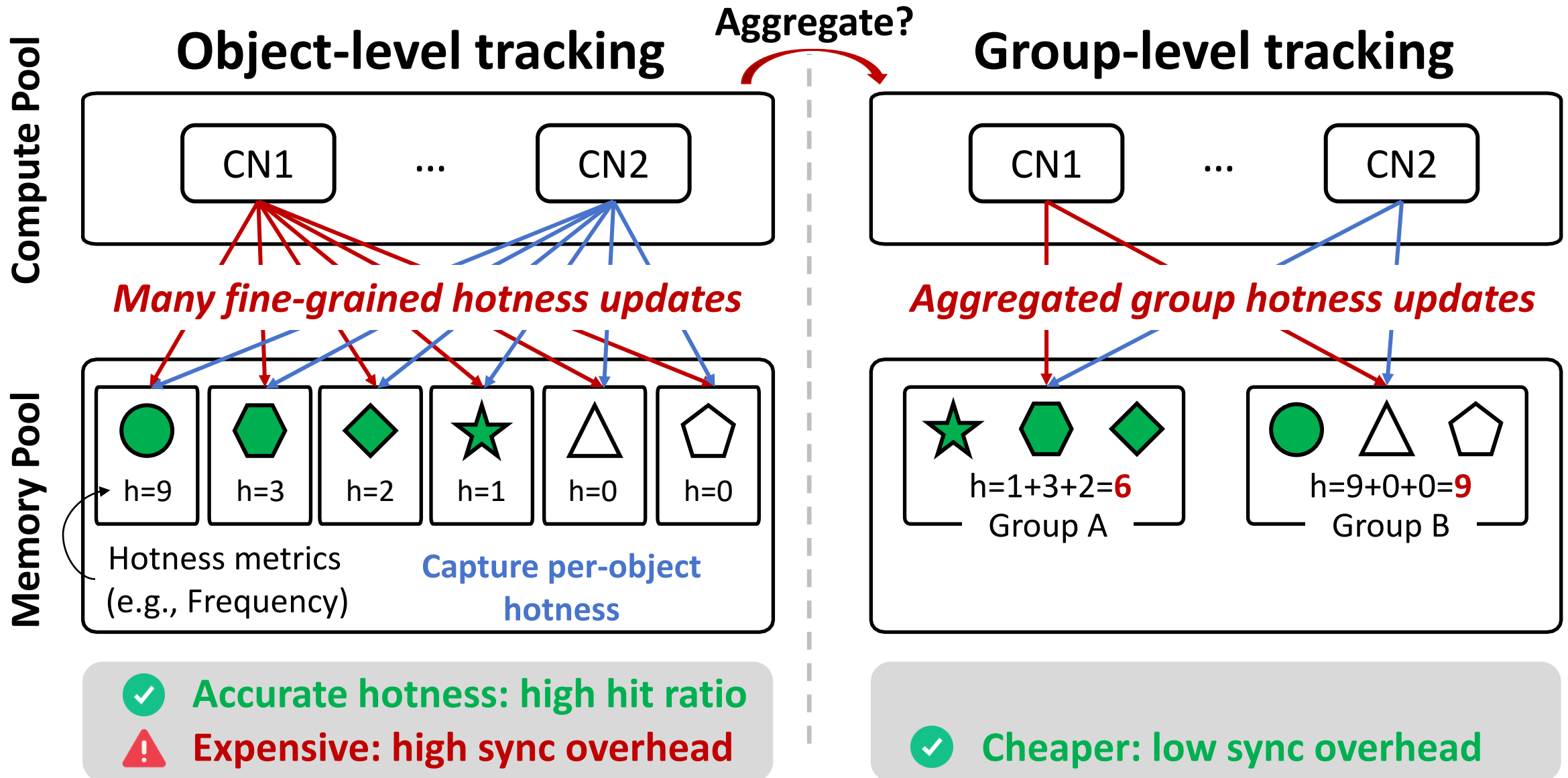
Memory Pool



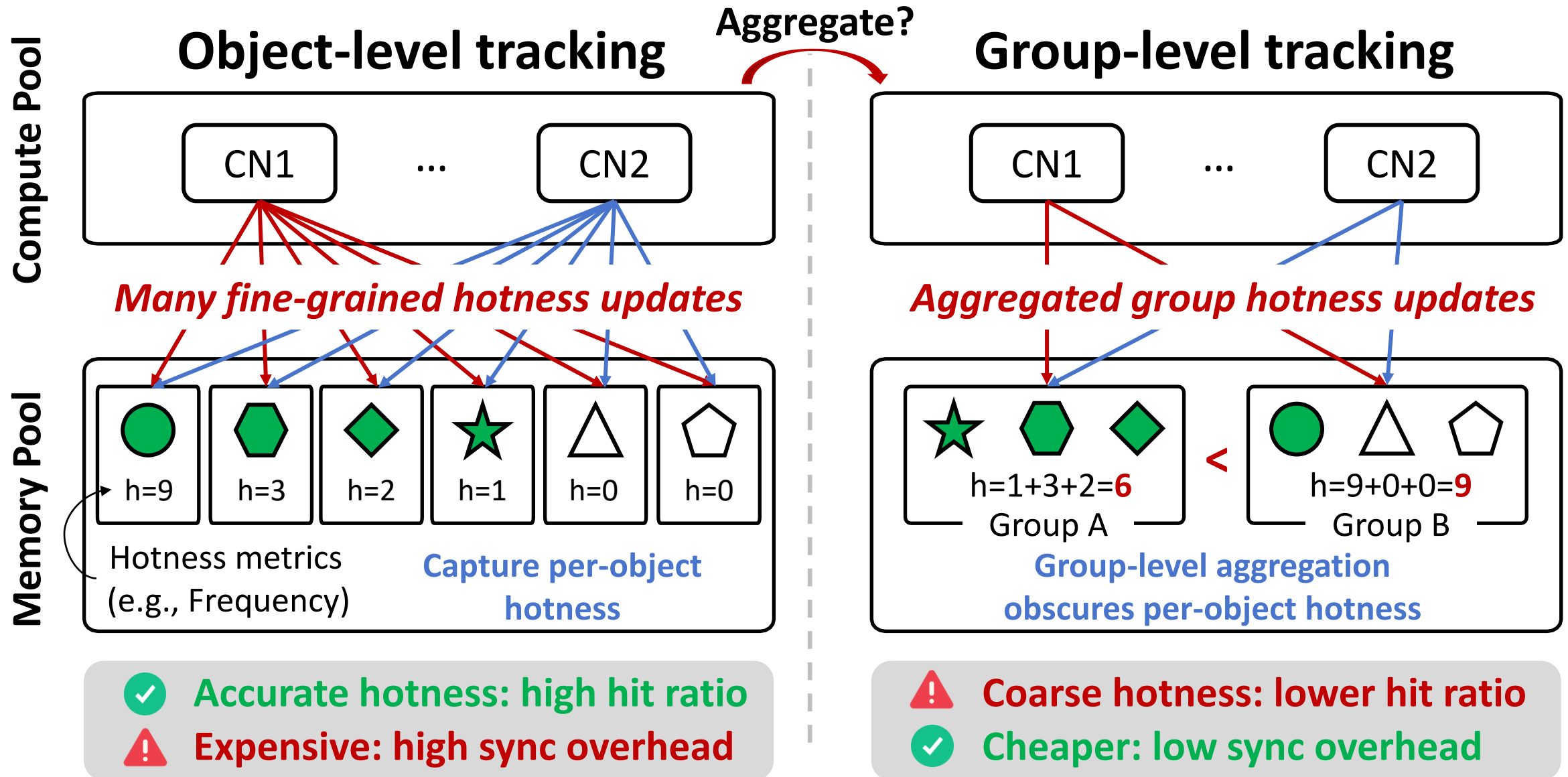
C2: Accuracy–Overhead Dilemma in Hotness Tracking



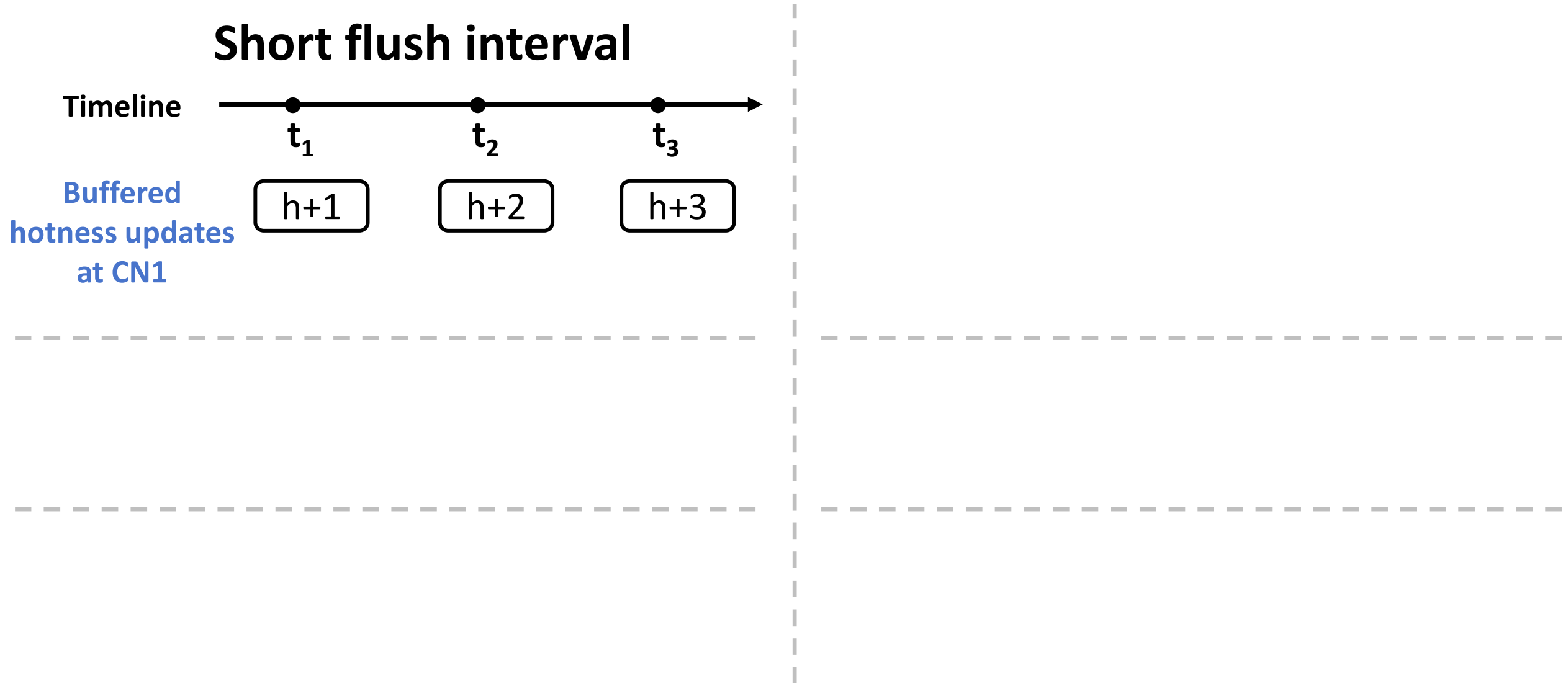
C2: Accuracy–Overhead Dilemma in Hotness Tracking



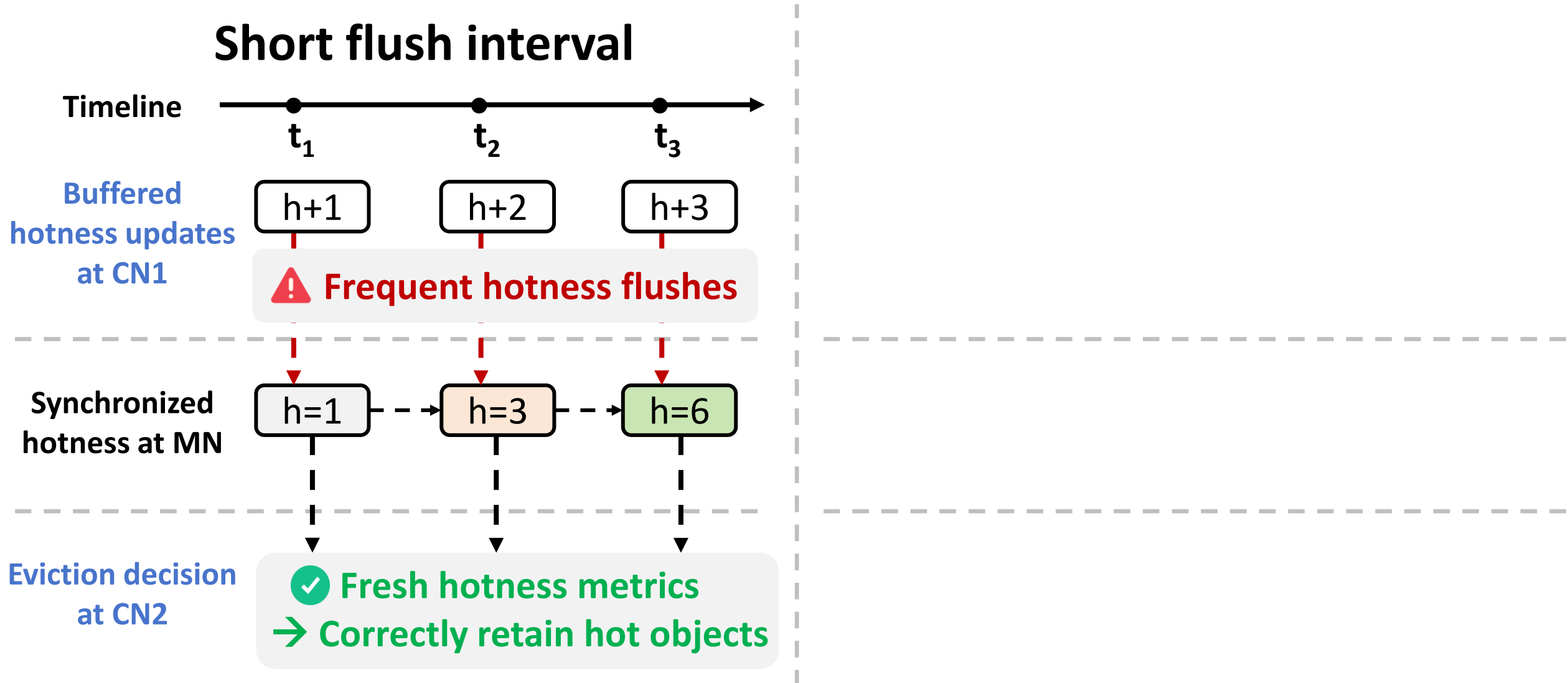
C2: Accuracy–Overhead Dilemma in Hotness Tracking



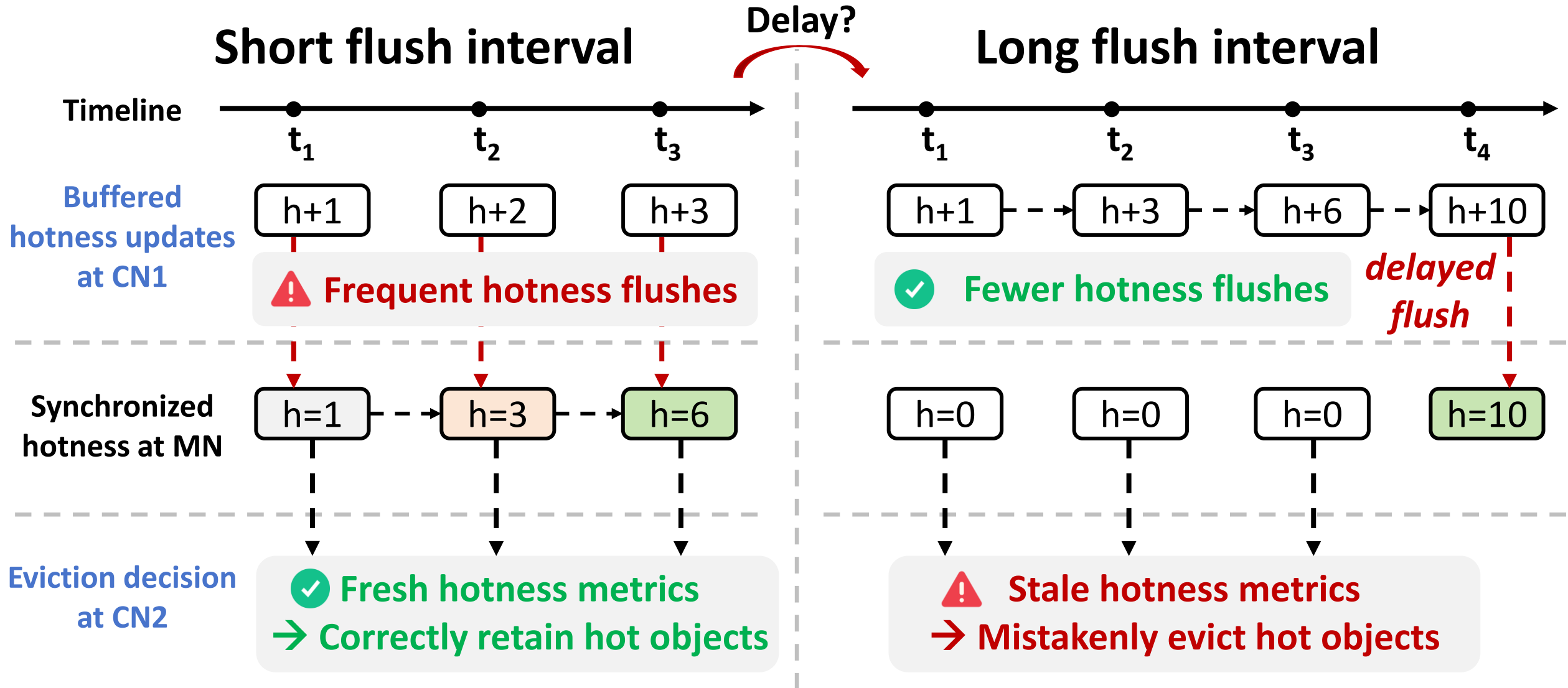
C3: Timeliness–Cost Conflict in Synchronization



C3: Timeliness–Cost Conflict in Synchronization



C3: Timeliness–Cost Conflict in Synchronization



FORGE Overview

Principle: prioritize synchronization efficiency

Compute Pool

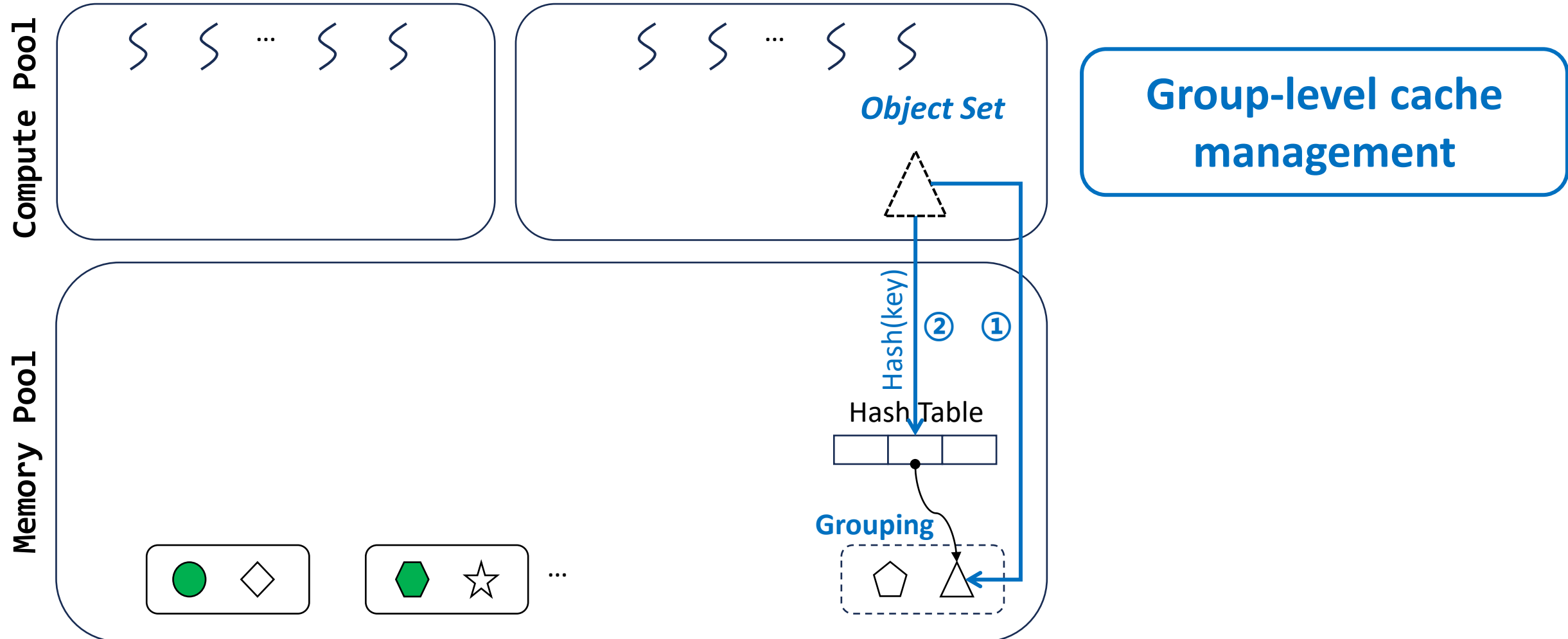
§ § ... § §

§ § ... § §

Memory Pool

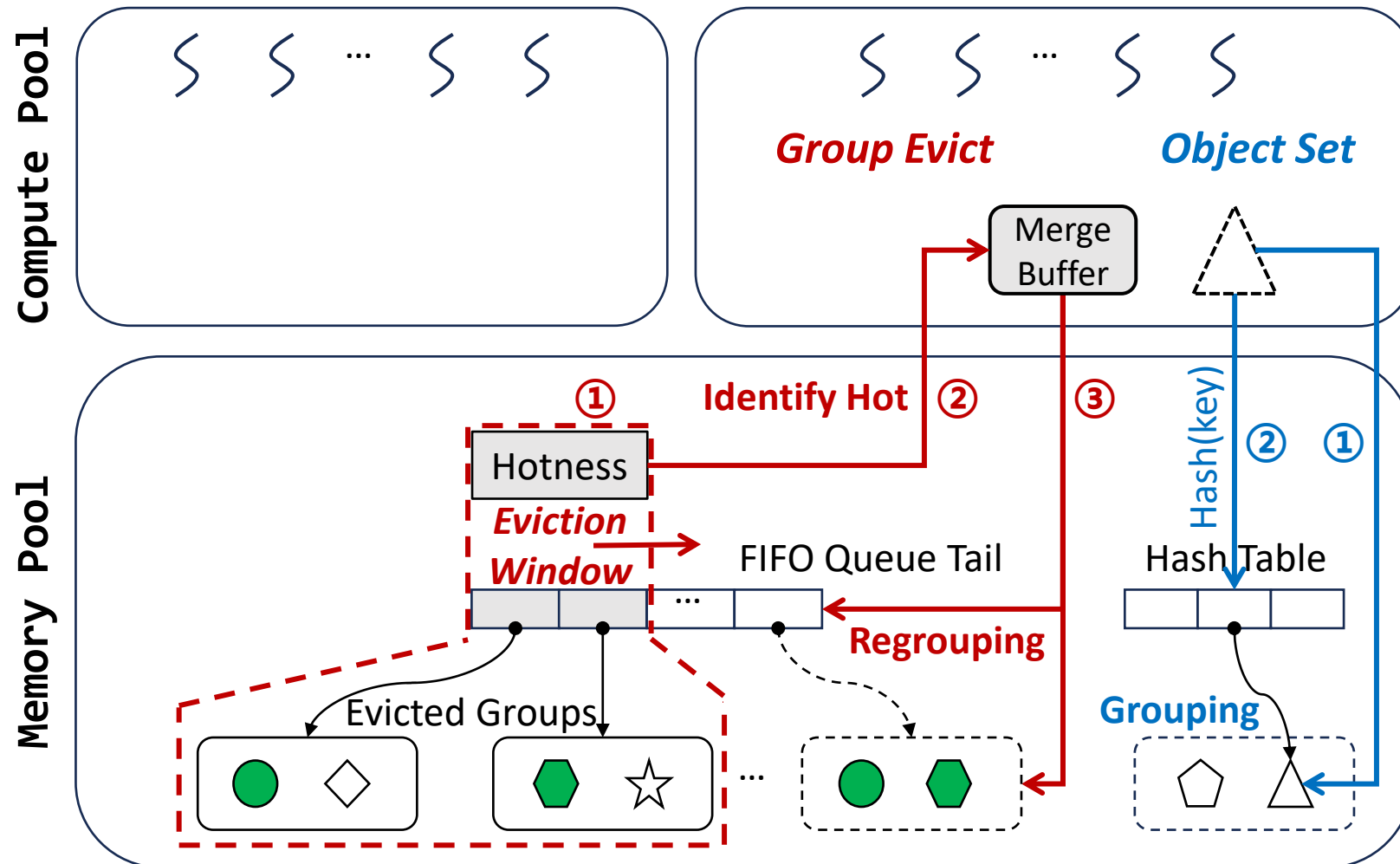
FORGE Overview

Principle: prioritize synchronization efficiency



FORGE Overview

Principle: prioritize synchronization efficiency

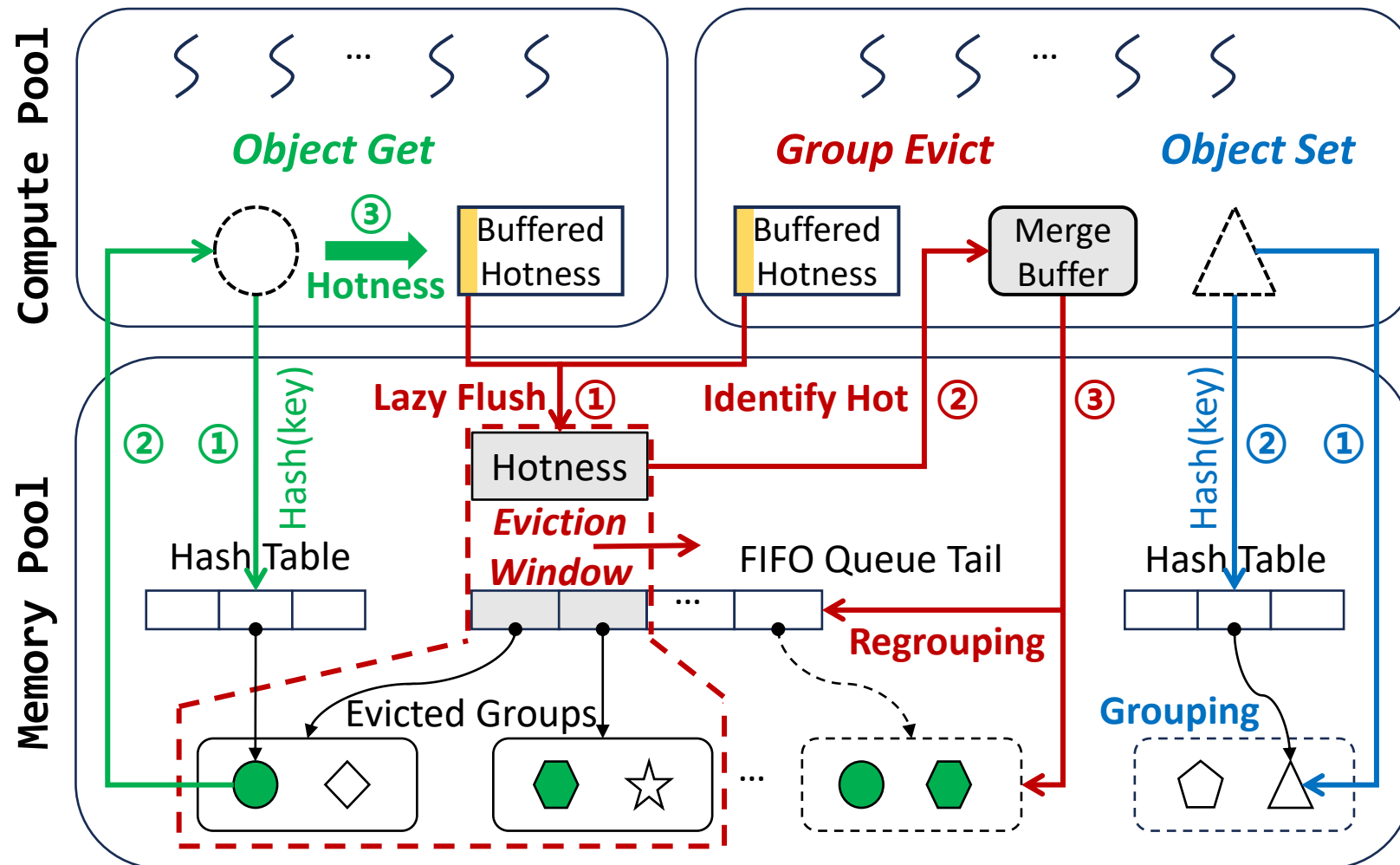


Group-level cache management

Contention-free, hotness-aware FIFO

FORGE Overview

Principle: prioritize synchronization efficiency



Group-level cache management

Contention-free, hotness-aware FIFO

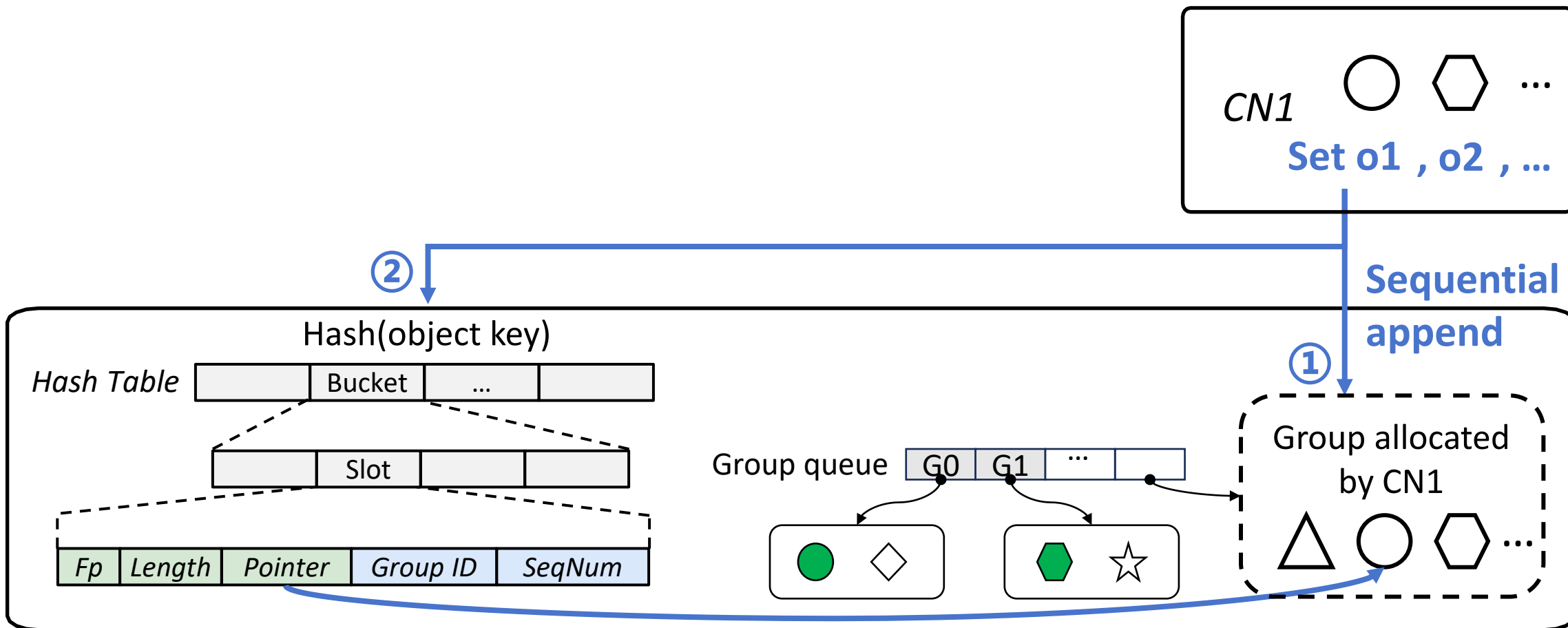
Object-level tracking, lazy hotness sync

Grouping-aware Get/Set

Set: write-time grouping

Compute Pool

Memory Pool



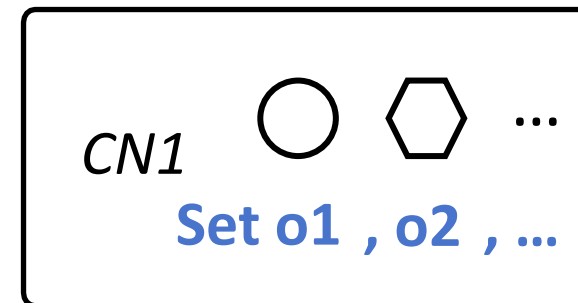
Temporal locality: objects written close in time often exhibit correlated access patterns.

Grouping-aware Get/Set

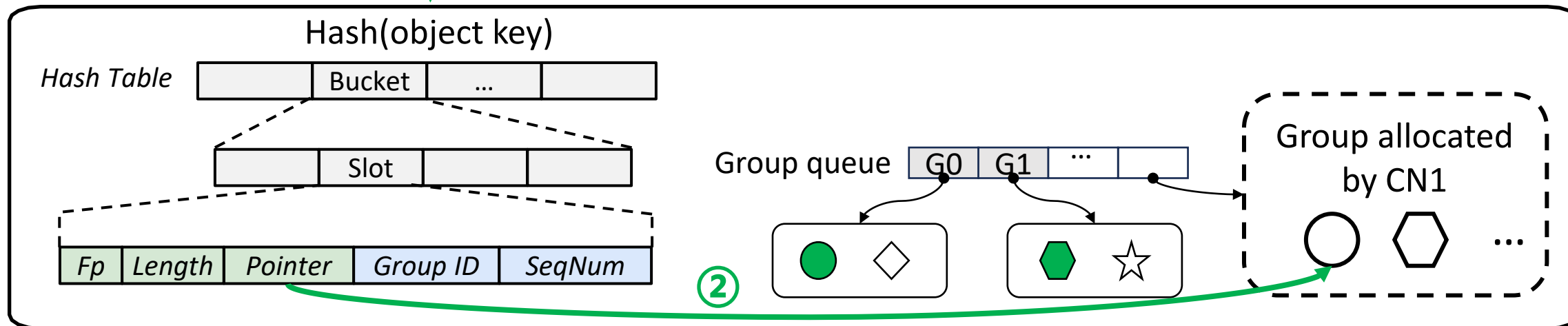
Get: object hotness tracking

Set: write-time grouping

Compute Pool



Memory Pool

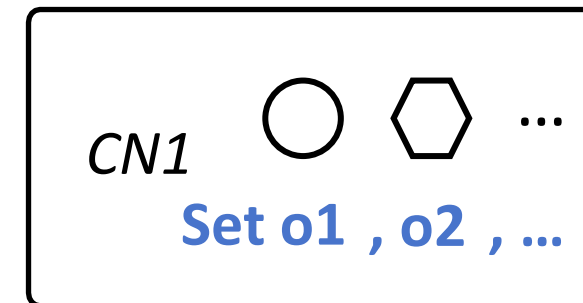
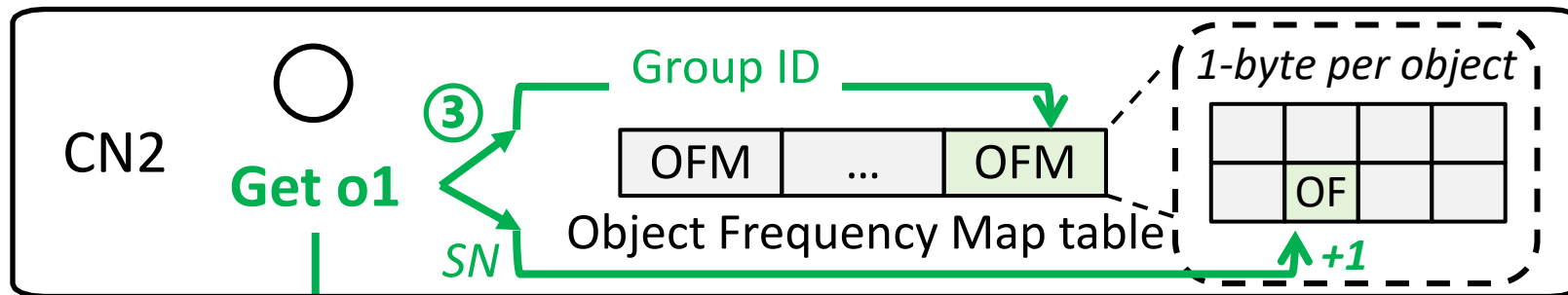


Grouping-aware Get/Set

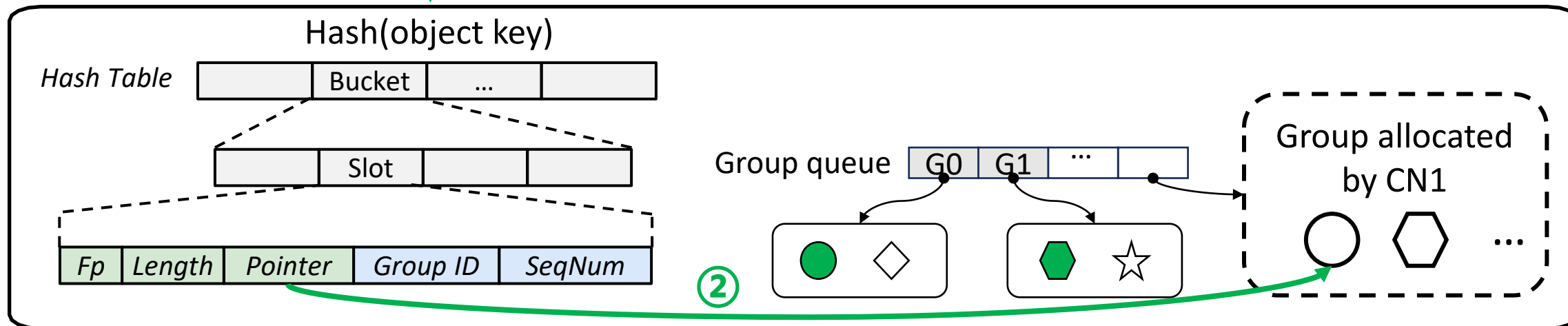
Get: object hotness tracking

Set: write-time grouping

Compute Pool



Memory Pool



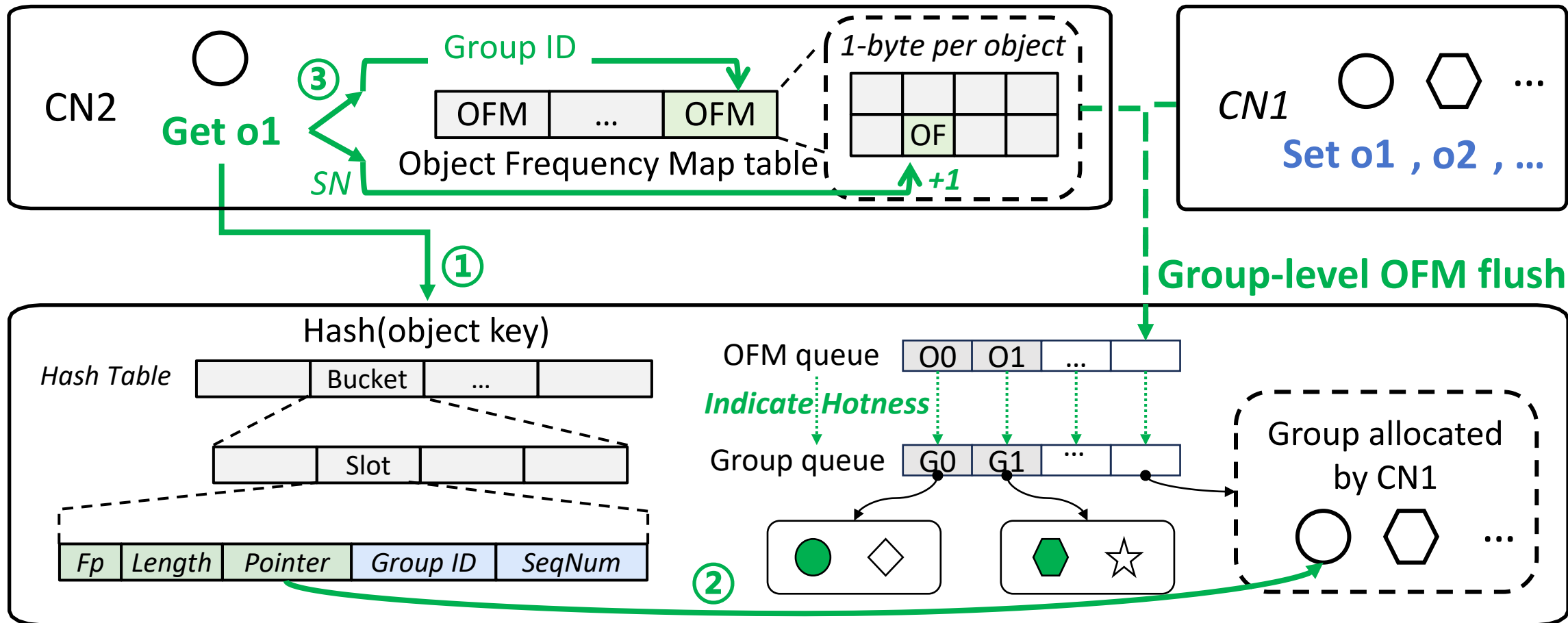
Grouping-aware Get/Set

Get: object hotness tracking

Set: write-time grouping

Compute Pool

Memory Pool



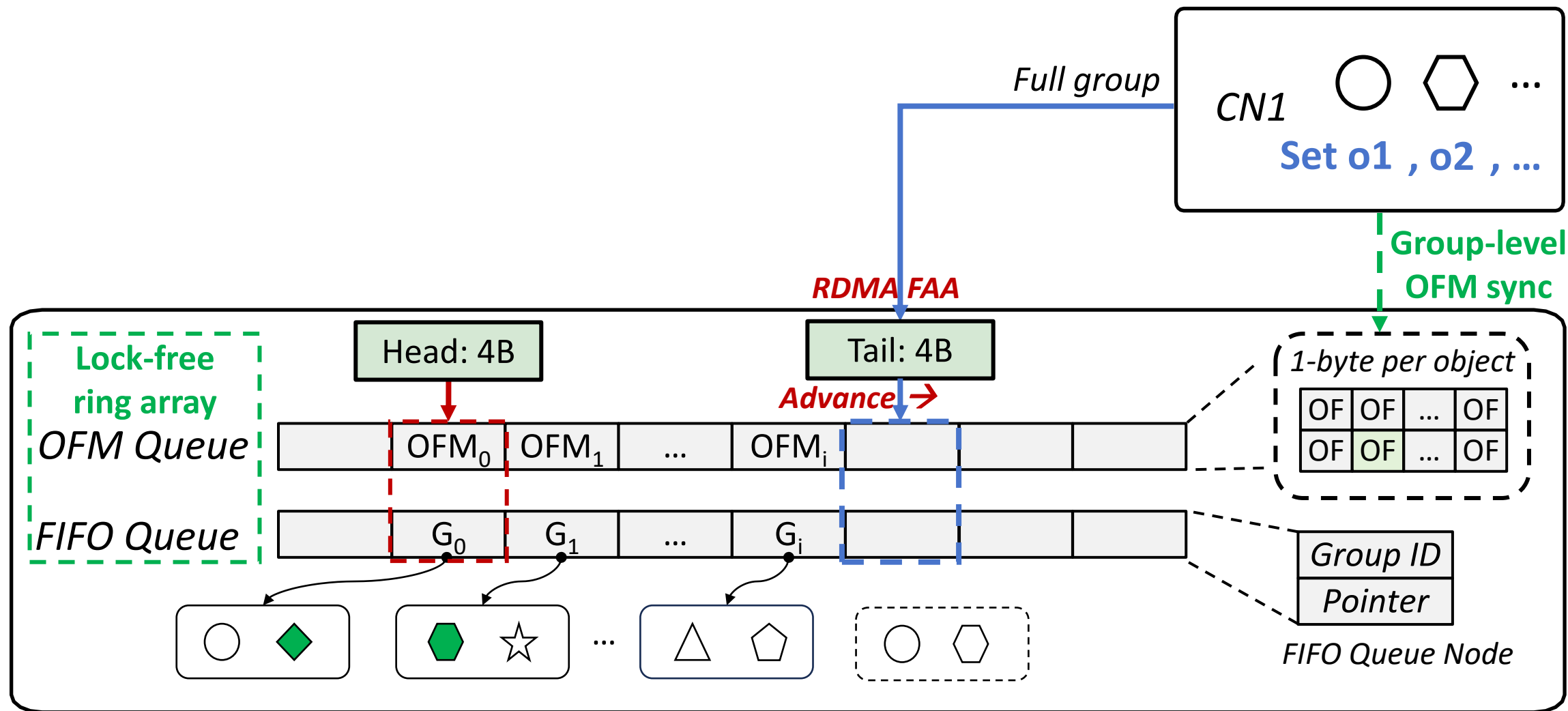
Hybrid granularity: accurate object-level hotness tracking, batched group-level sync.

Contention-free FIFO Management

Set: write-time grouping

Compute Pool

Memory Pool

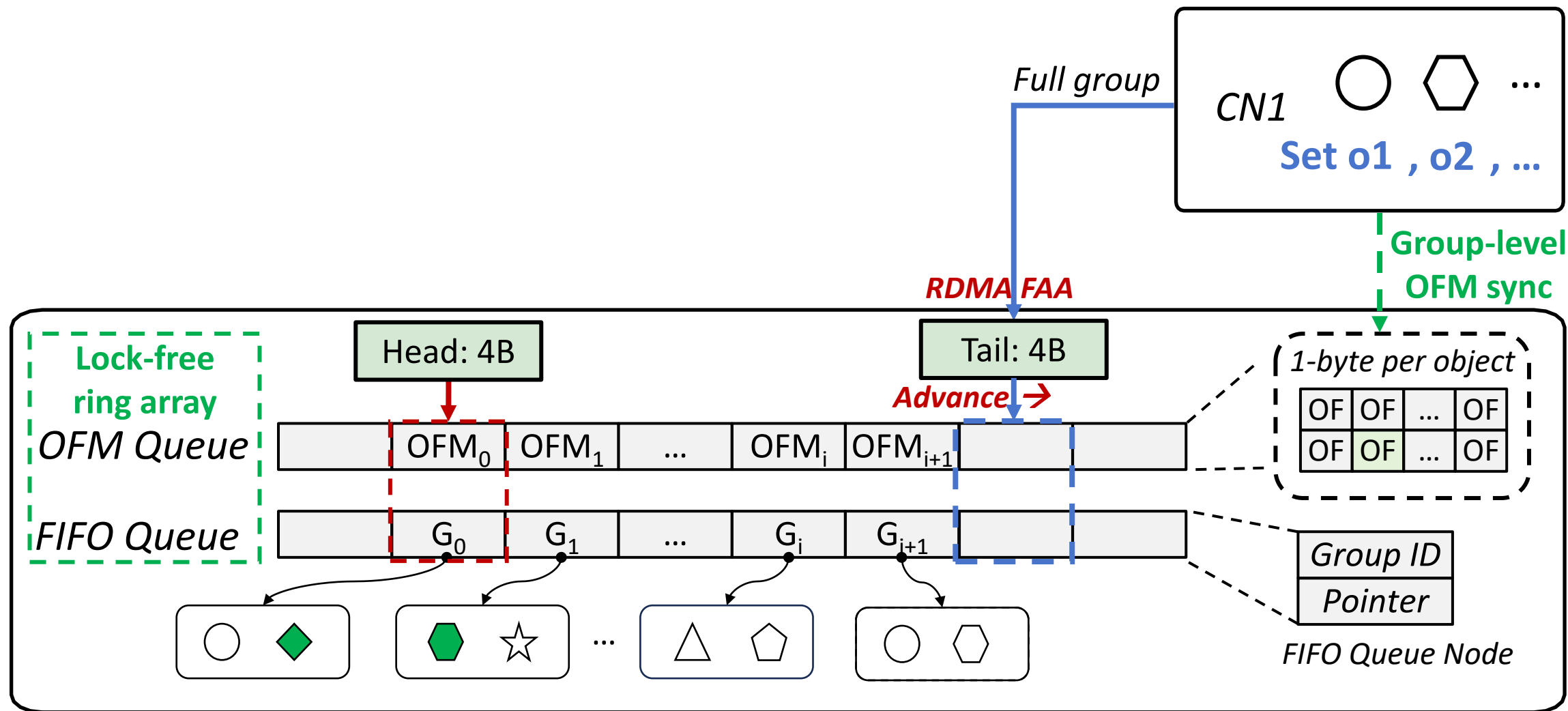


Contention-free FIFO Management

Compute Pool

Memory Pool

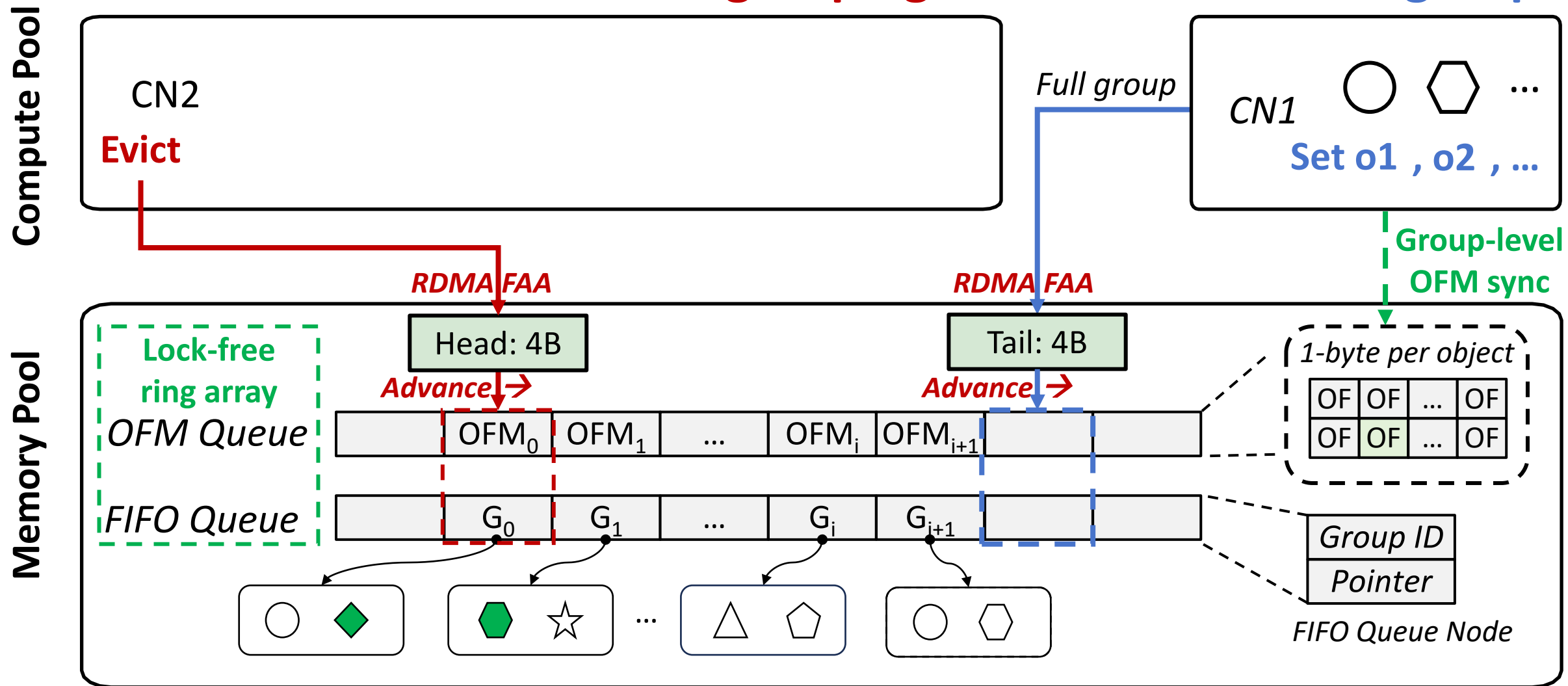
Set: write-time grouping



Contention-free FIFO Management

Evict: hotness-based regrouping

Set: write-time grouping



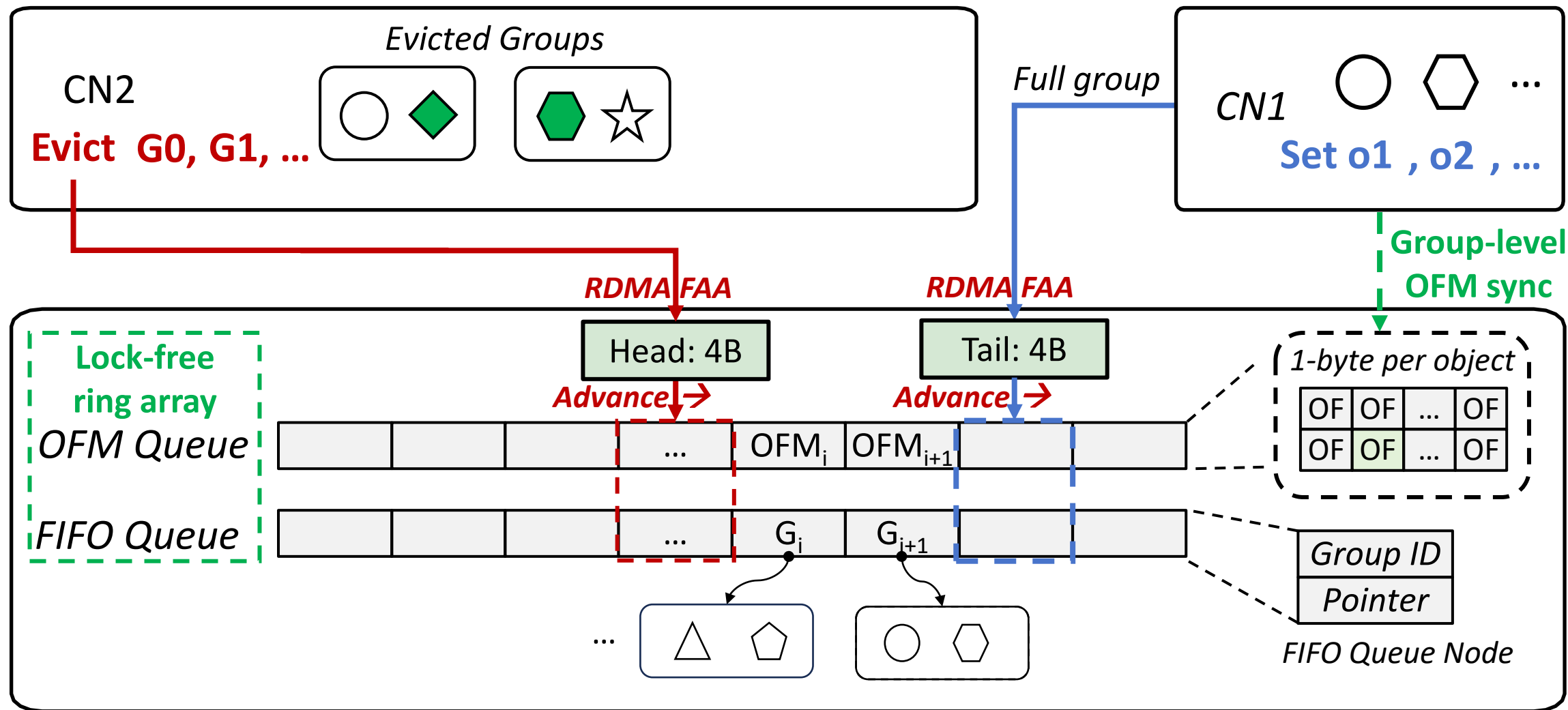
Contention-free FIFO Management

Evict: hotness-based regrouping

Set: write-time grouping

Compute Pool

Memory Pool



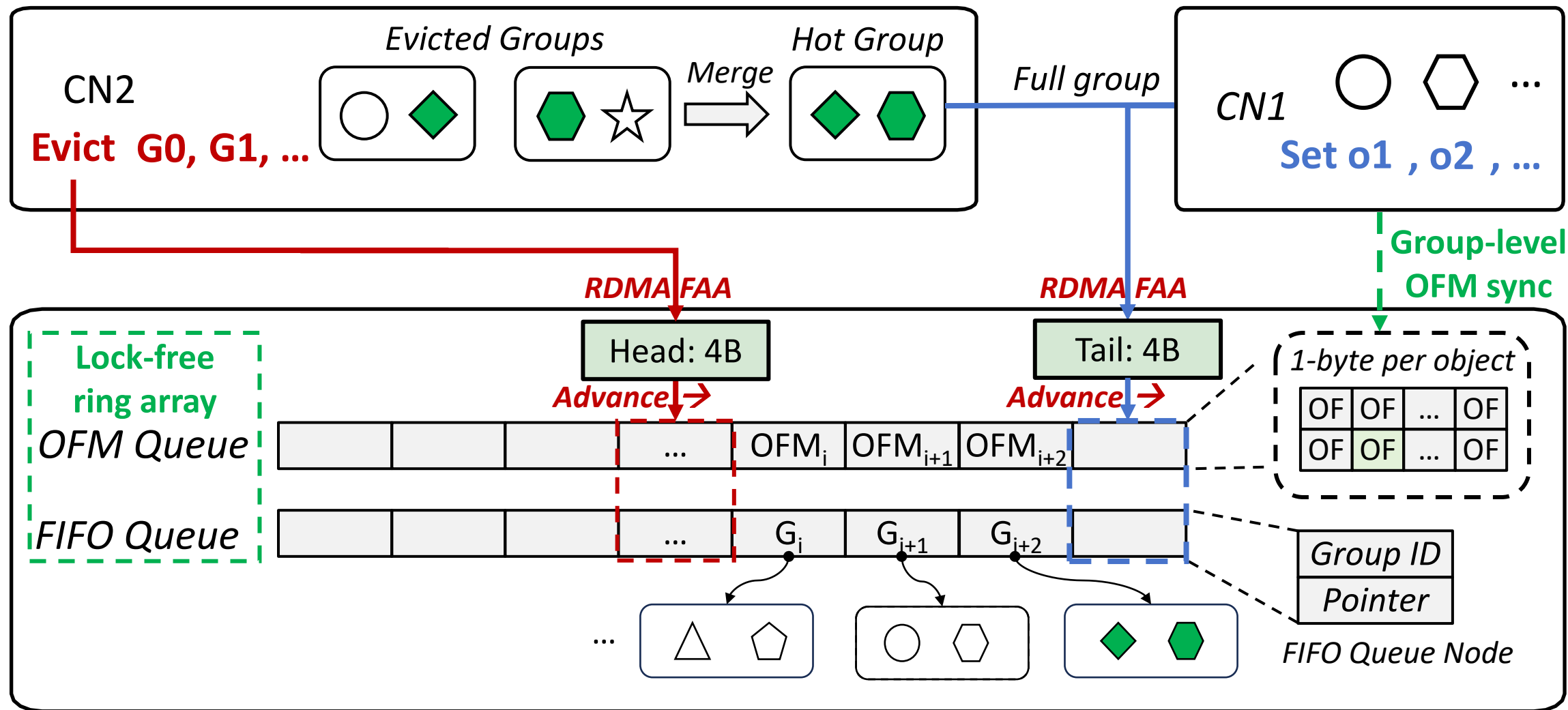
Contention-free FIFO Management

Evict: hotness-based regrouping

Set: write-time grouping

Compute Pool

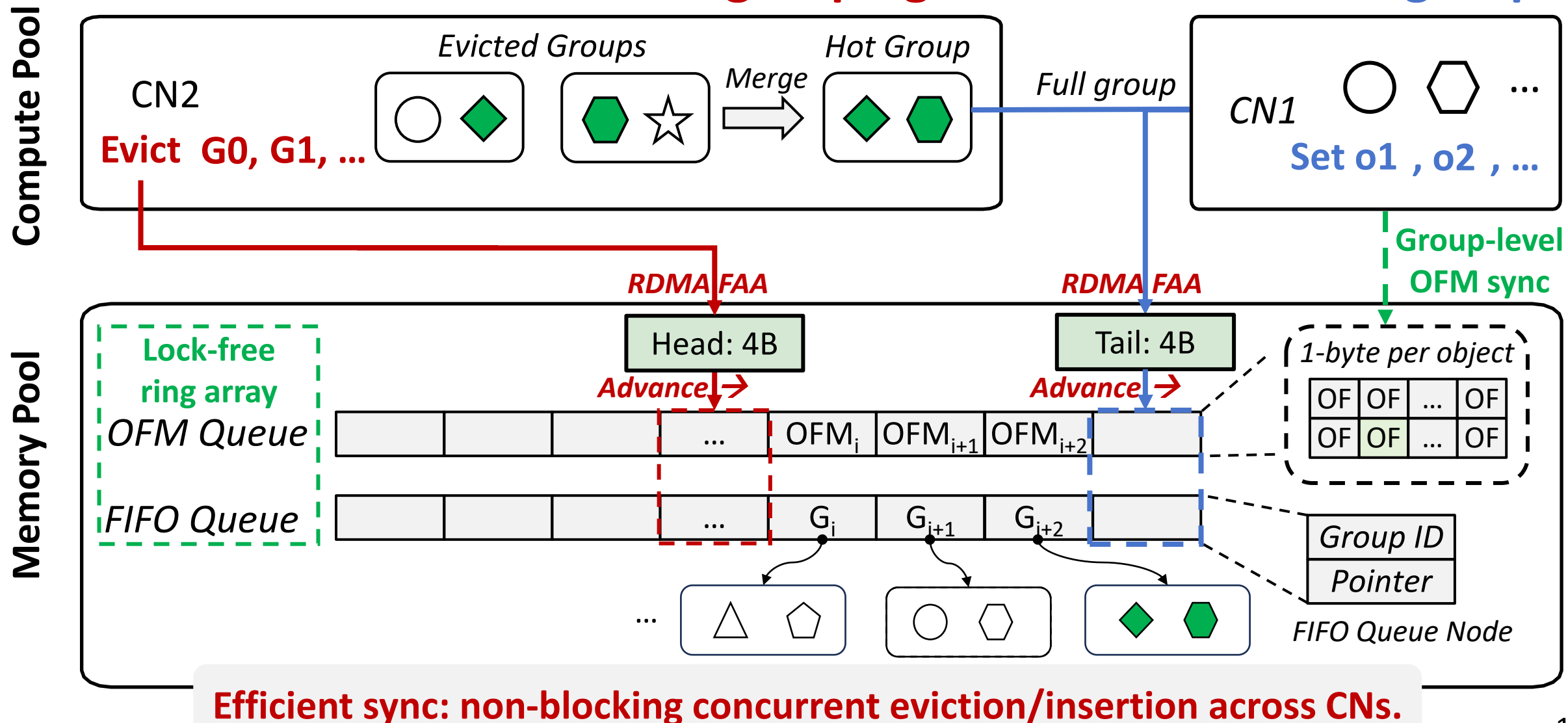
Memory Pool



Contention-free FIFO Management

Evict: hotness-based regrouping

Set: write-time grouping



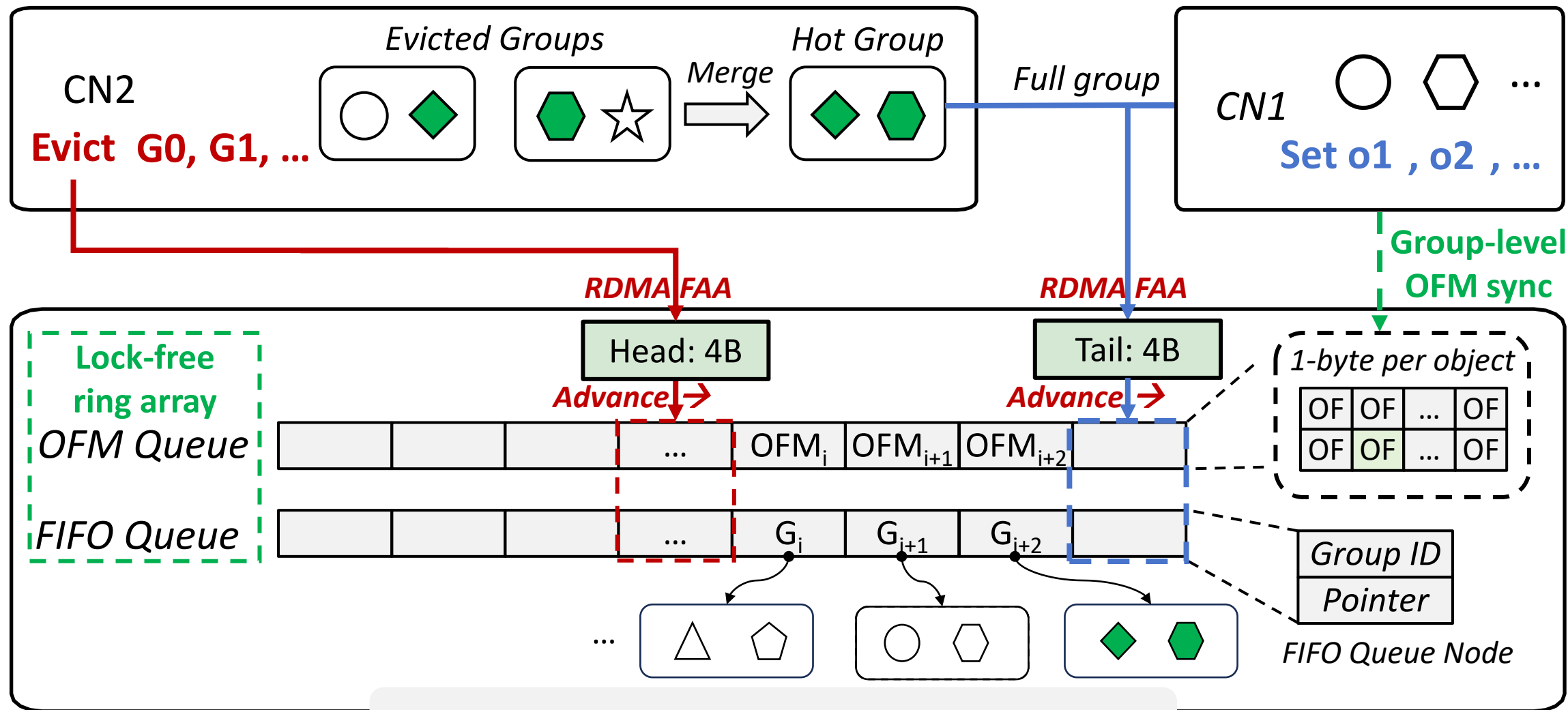
Contention-free FIFO Management

Evict: hotness-based regrouping

Set: write-time grouping

Compute Pool

Memory Pool



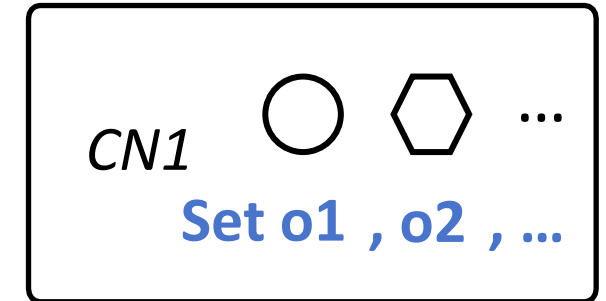
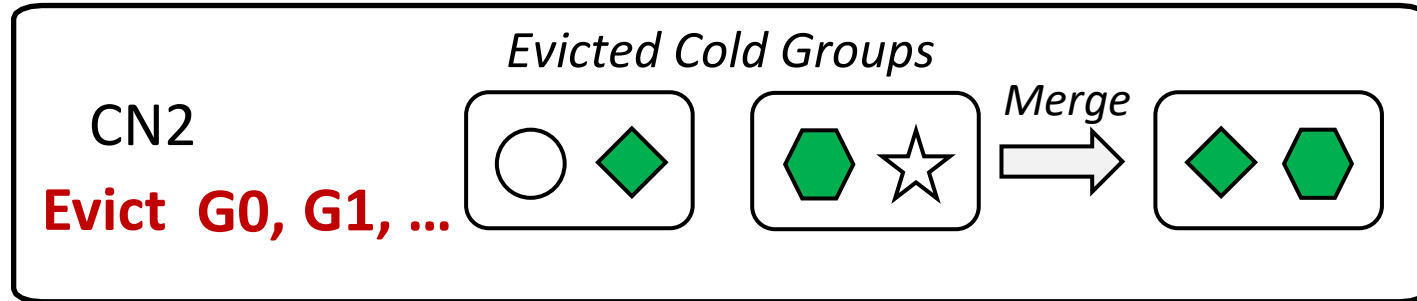
However, plain FIFO is hotness-agnostic.

Hotness-aware FIFO Segmentation

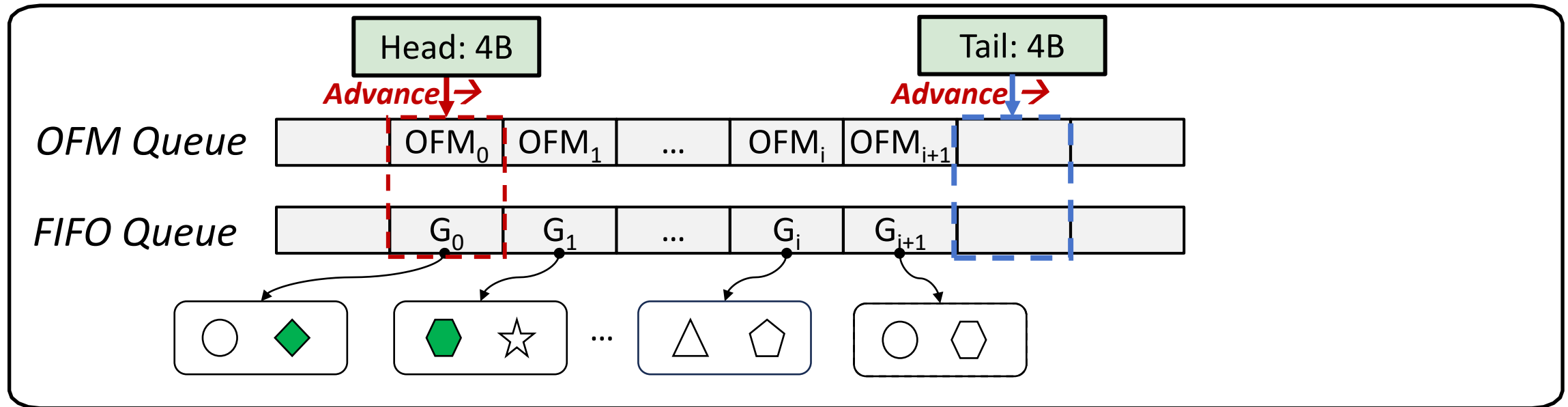
Evict: hotness-based regrouping

Set: write-time grouping

Compute Pool



Memory Pool

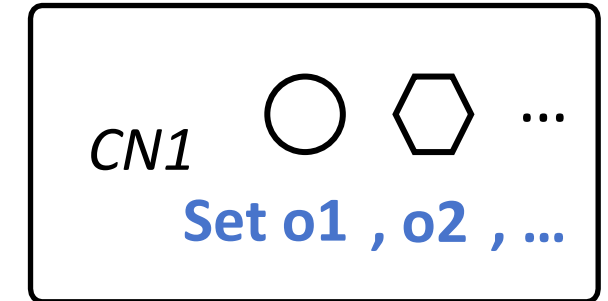
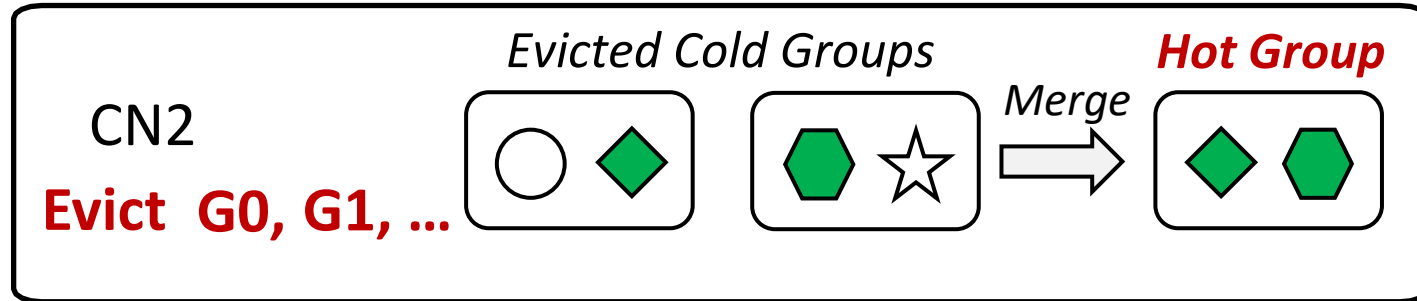


Hotness-aware FIFO Segmentation

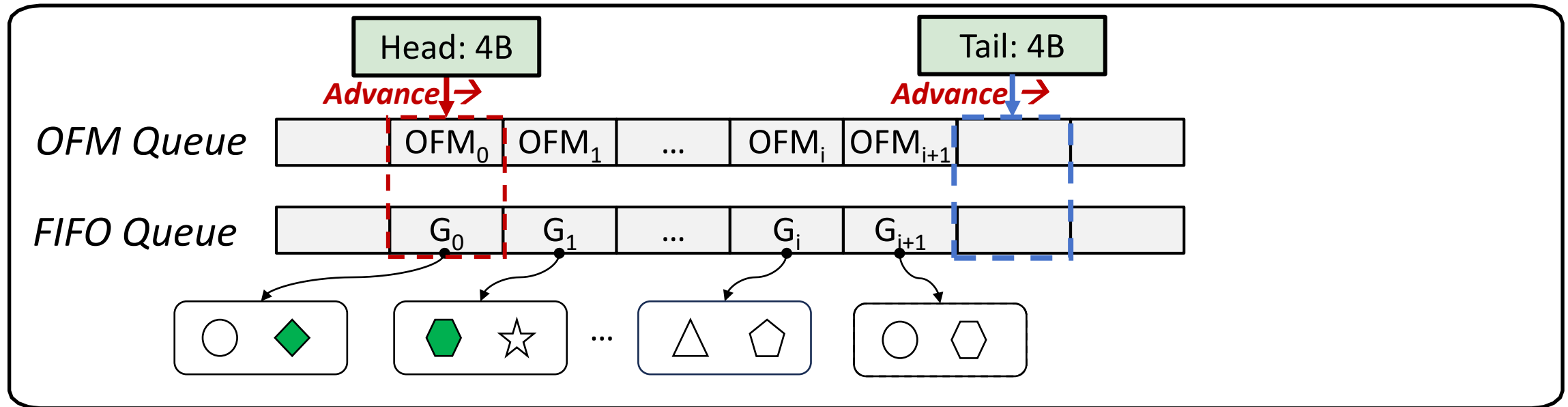
Evict: hotness-based regrouping

Set: write-time grouping

Compute Pool



Memory Pool

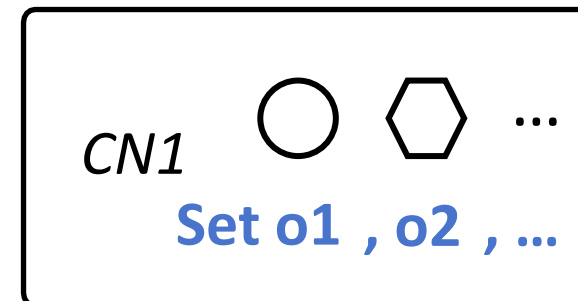
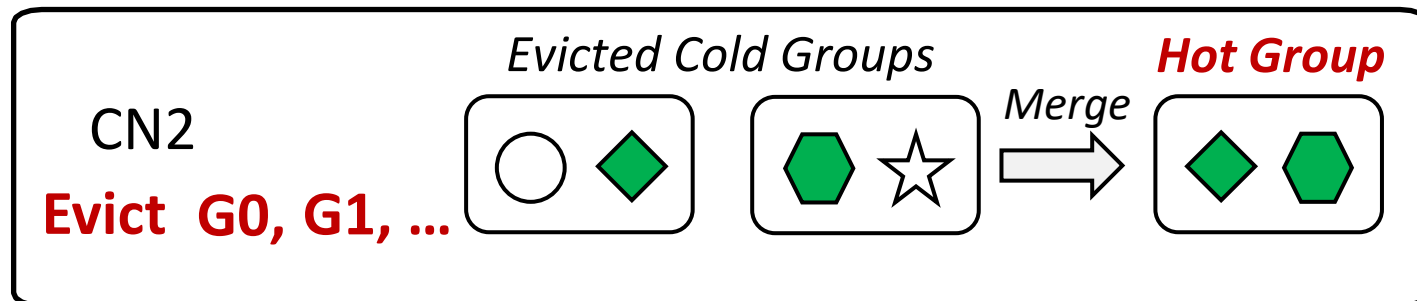


Hotness-aware FIFO Segmentation

Evict: hotness-based regrouping

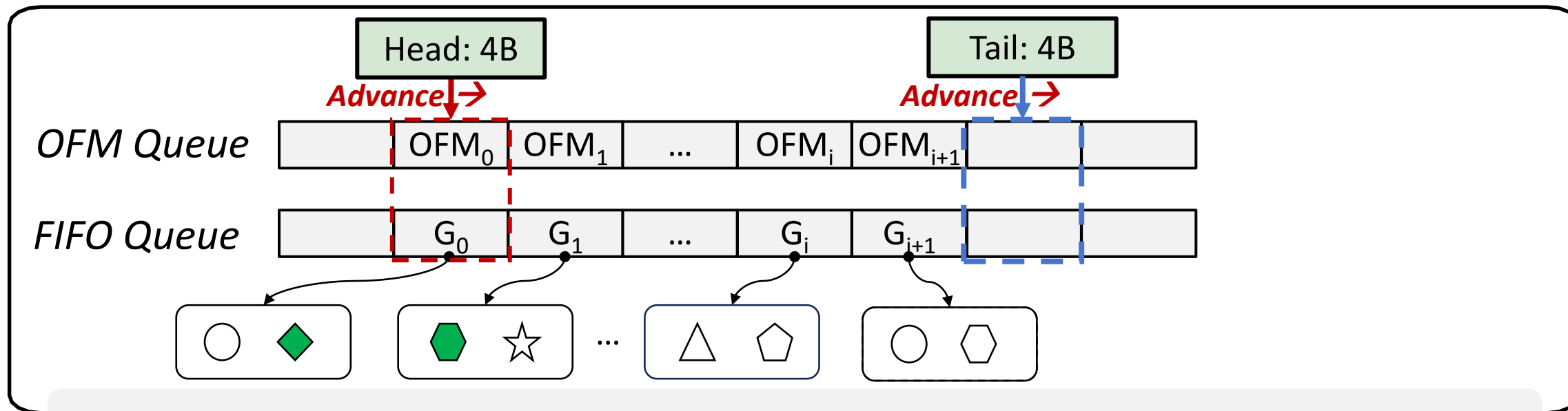
Set: write-time grouping

Compute Pool



New groups (likely cold)

Memory Pool



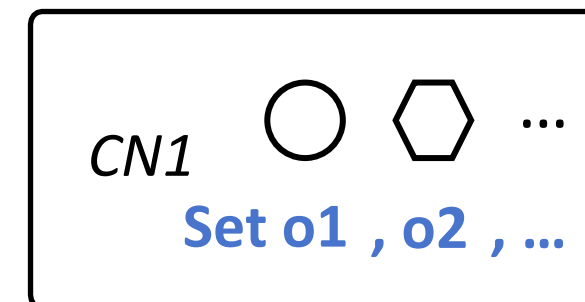
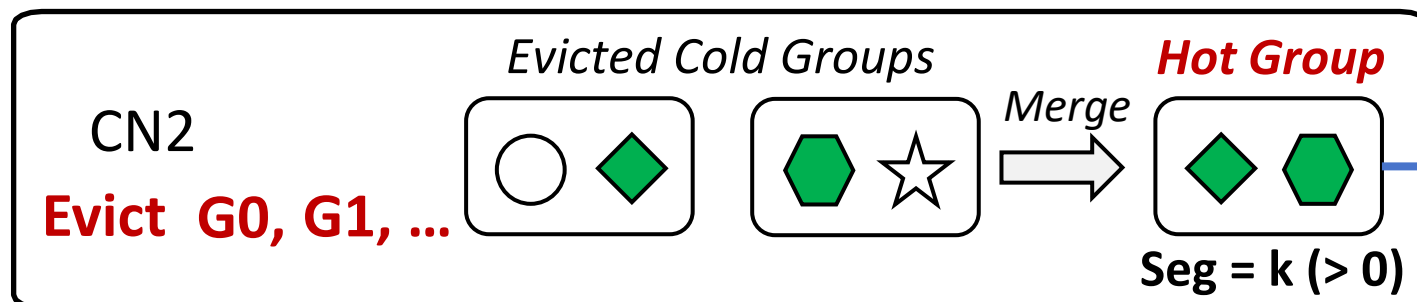
Evict colder groups sooner; defer hotter-group eviction without blocking.

Hotness-aware FIFO Segmentation

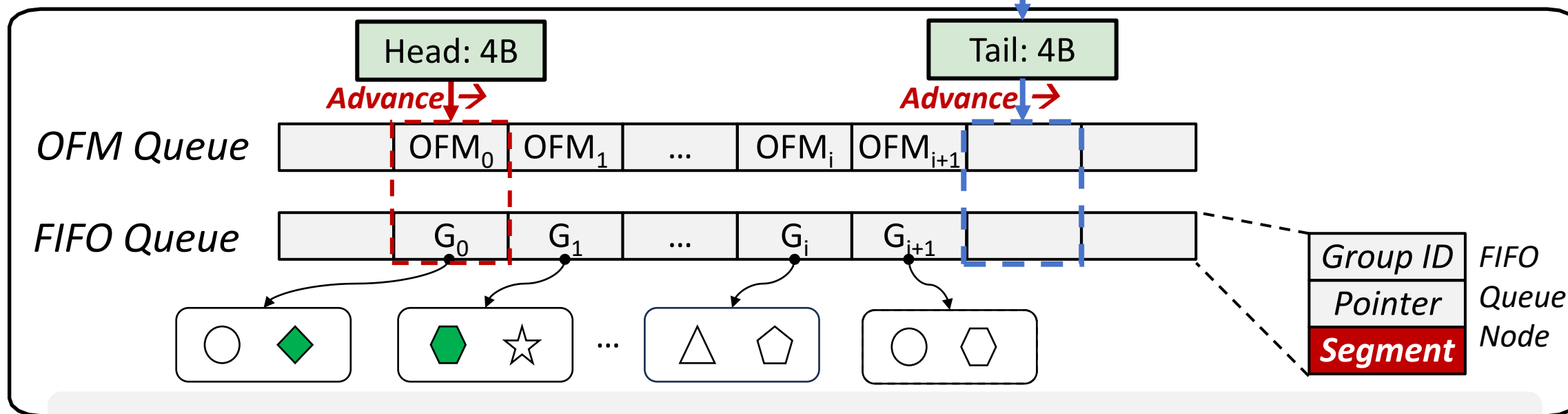
Evict: hotness-based regrouping

Set: write-time grouping

Compute Pool



Memory Pool

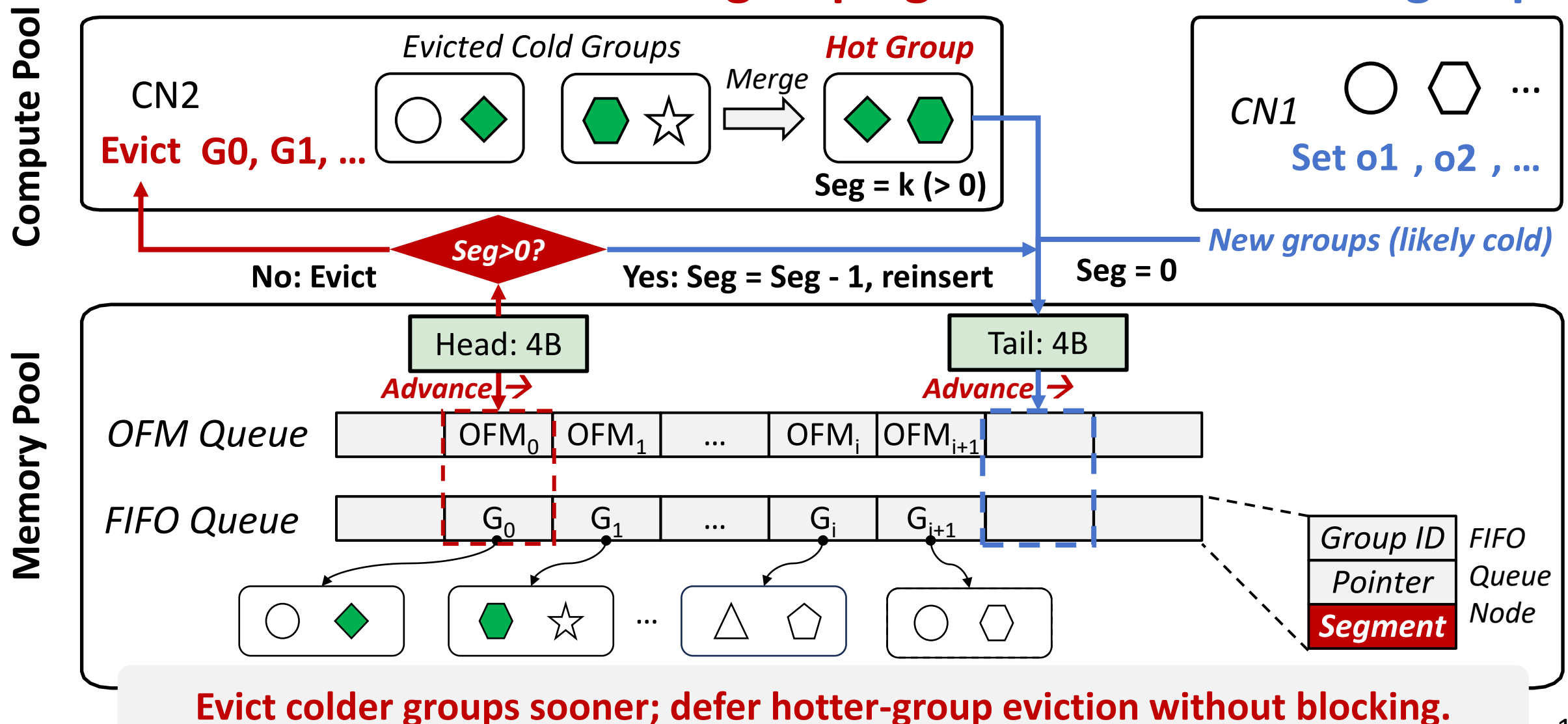


Evict colder groups sooner; defer hotter-group eviction without blocking.

Hotness-aware FIFO Segmentation

Evict: hotness-based regrouping

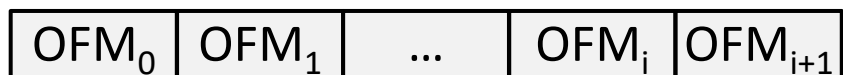
Set: write-time grouping



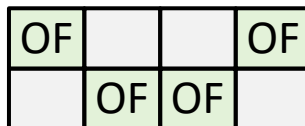
Lazy Hotness Synchronization

Compute Pool

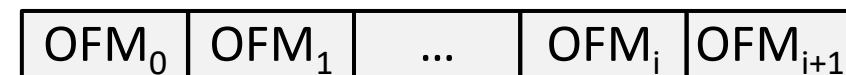
CN1 Buffered hotness metrics
(object frequency)



1-byte per object



CN2 Buffered hotness metrics
(object frequency)



When to sync?

Group-level
OFM sync

Group-level
OFM sync

Memory Pool

Head: 4B

Tail: 4B

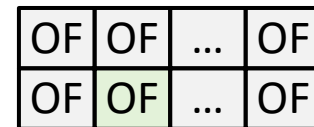
Advance →

Advance →

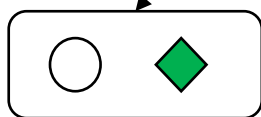
OFM Queue



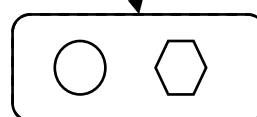
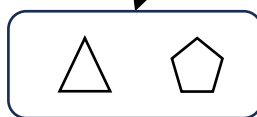
1-byte per object



FIFO Queue



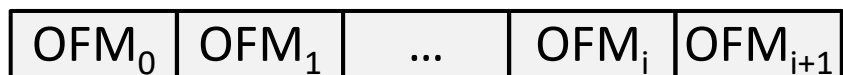
...



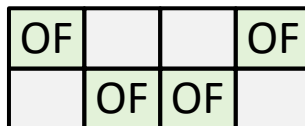
Lazy Hotness Synchronization

Compute Pool

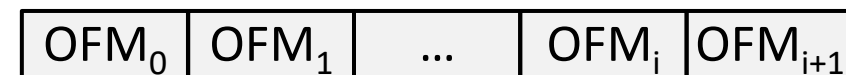
CN1 Buffered hotness metrics
(object frequency)



1-byte per object



CN2 Buffered hotness metrics
(object frequency)



When to sync?

Insight: synchronized metrics are only used during eviction.

Memory Pool

Head: 4B

Tail: 4B

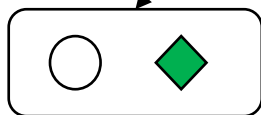
Advance →

Advance →

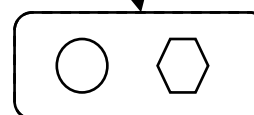
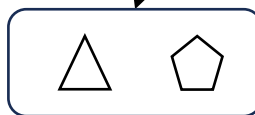
OFM Queue



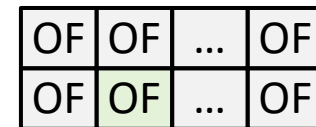
FIFO Queue



...



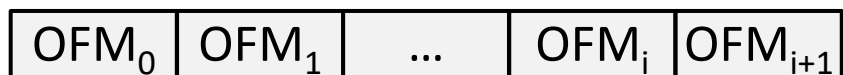
1-byte per object



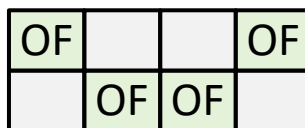
Lazy Hotness Synchronization

Compute Pool

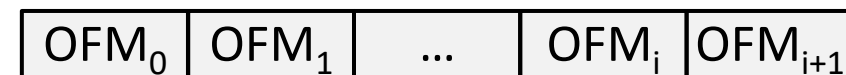
CN1 Buffered hotness metrics
(object frequency)



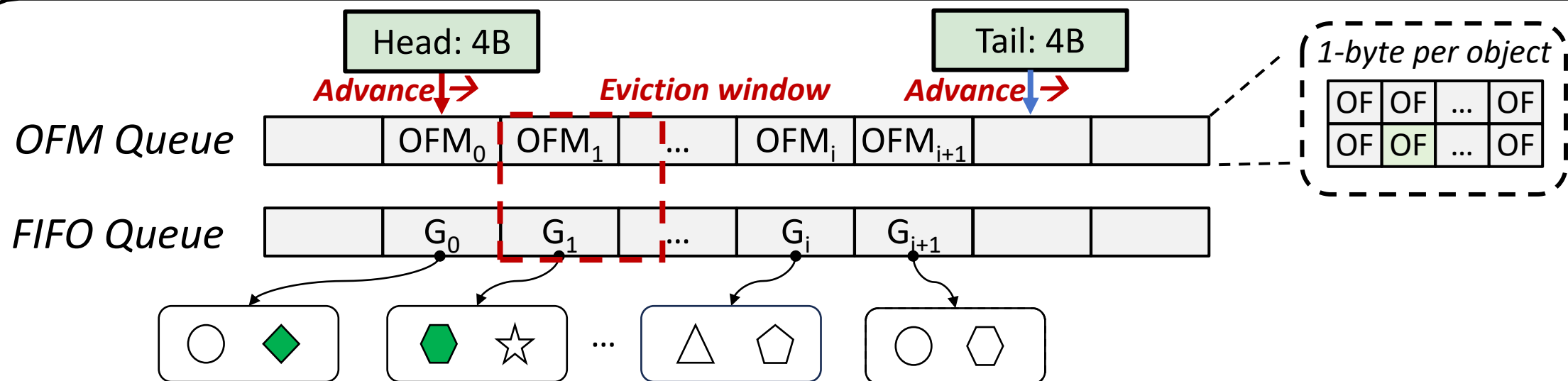
1-byte per object



CN2 Buffered hotness metrics
(object frequency)



Memory Pool



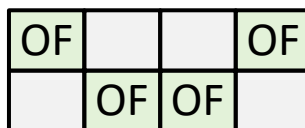
Lazy Hotness Synchronization

Compute Pool

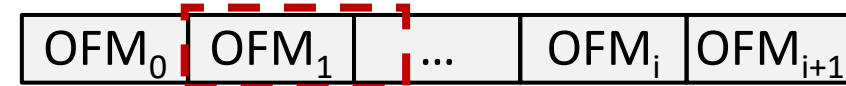
CN1 Buffered hotness metrics
(object frequency)



1-byte per object



CN2 Buffered hotness metrics
(object frequency)



Lazily defer flush until groups near the FIFO head

Memory Pool

Head: 4B

Tail: 4B

Advance

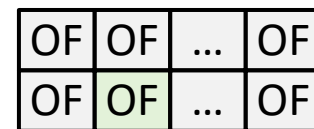
Eviction window

Advance

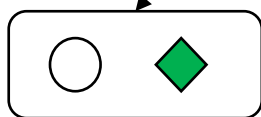
OFM Queue



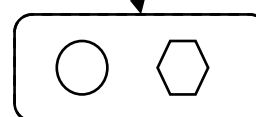
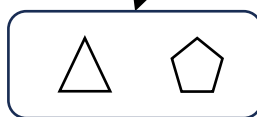
1-byte per object



FIFO Queue



...



Lazy-yet-timely hotness flushing eliminates unnecessary sync.

More Details

- Lock-free Get/Set consistency
- Lightweight regrouping
- Lazy sync implementation
- Support for multi-queue designs
- ...



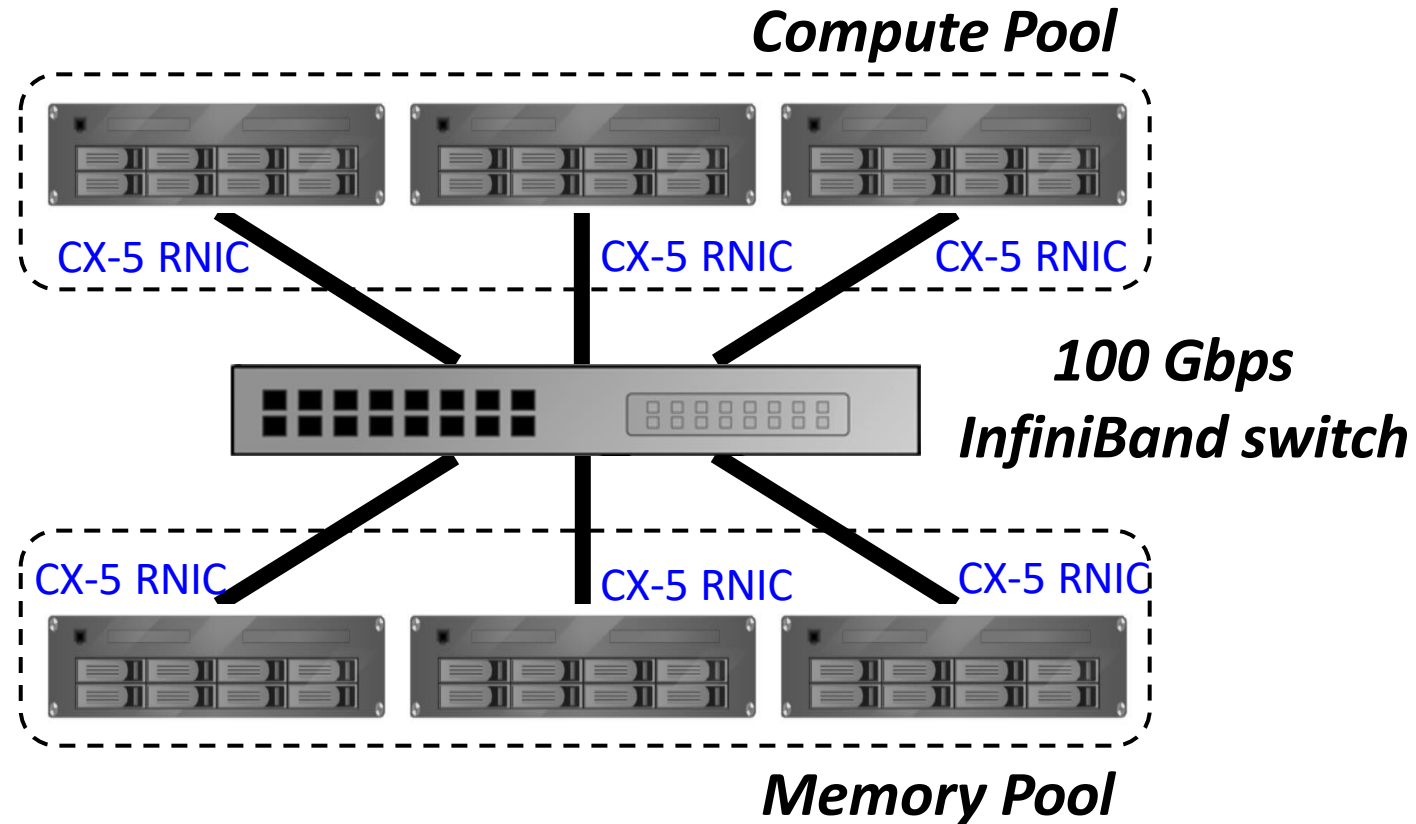
Evaluation

➤ Workloads

- YCSB A/B/C/D/E
- CloudPhysics
- Wikimedia
- Microsoft Research Cambridge
- IBM
- Meta
- Twitter

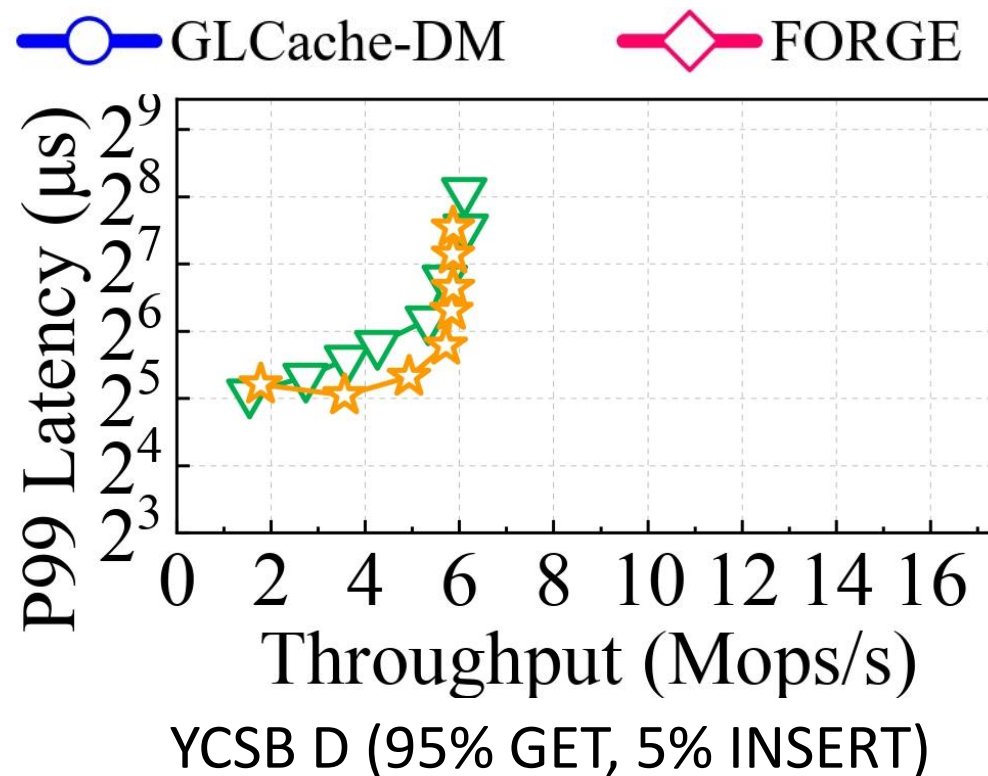
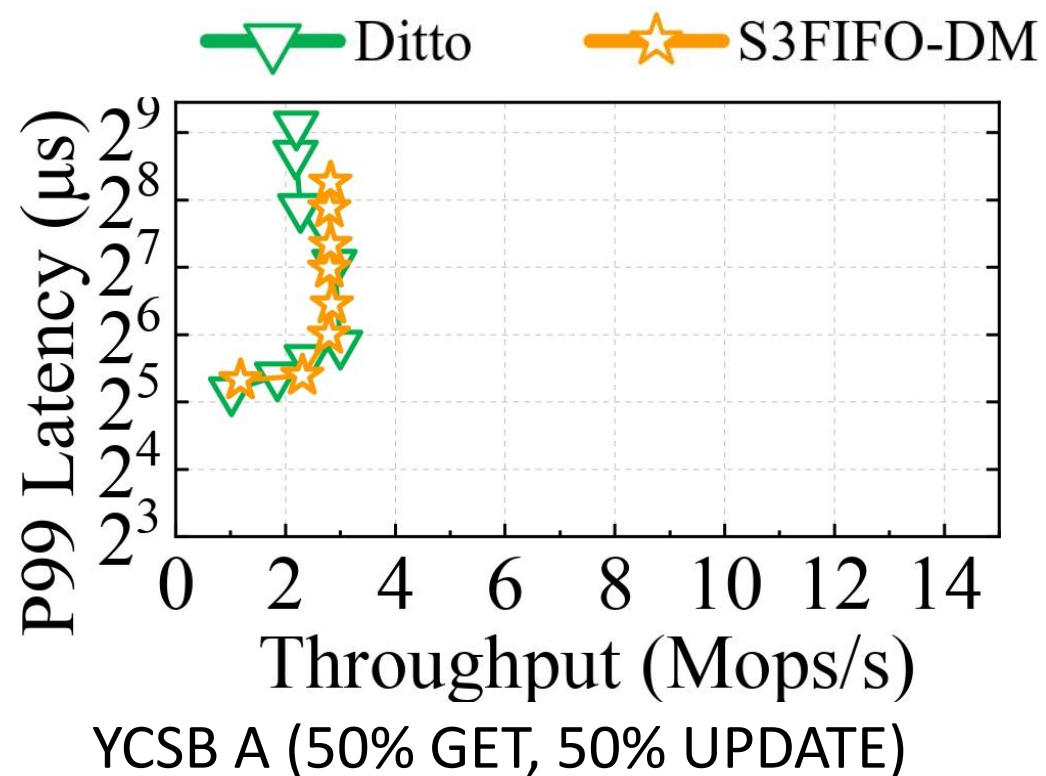
➤ Comparisons

- Ditto [SOSP '23] (state-of-the-art in RDMA-based DM caching systems)
- S3FIFO-DM (adapted from S3-FIFO [SOSP '23], a recent object-level FIFO scheme)
- GLCache-DM (adapted from GL-Cache [FAST '24], a recent group-level scheme)



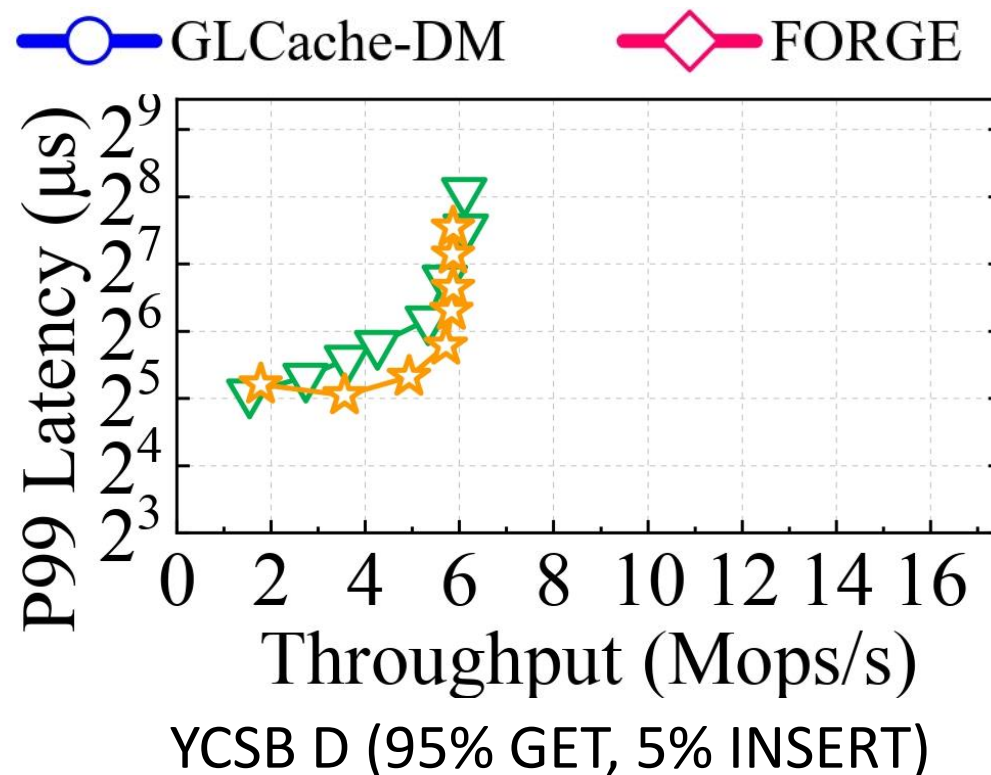
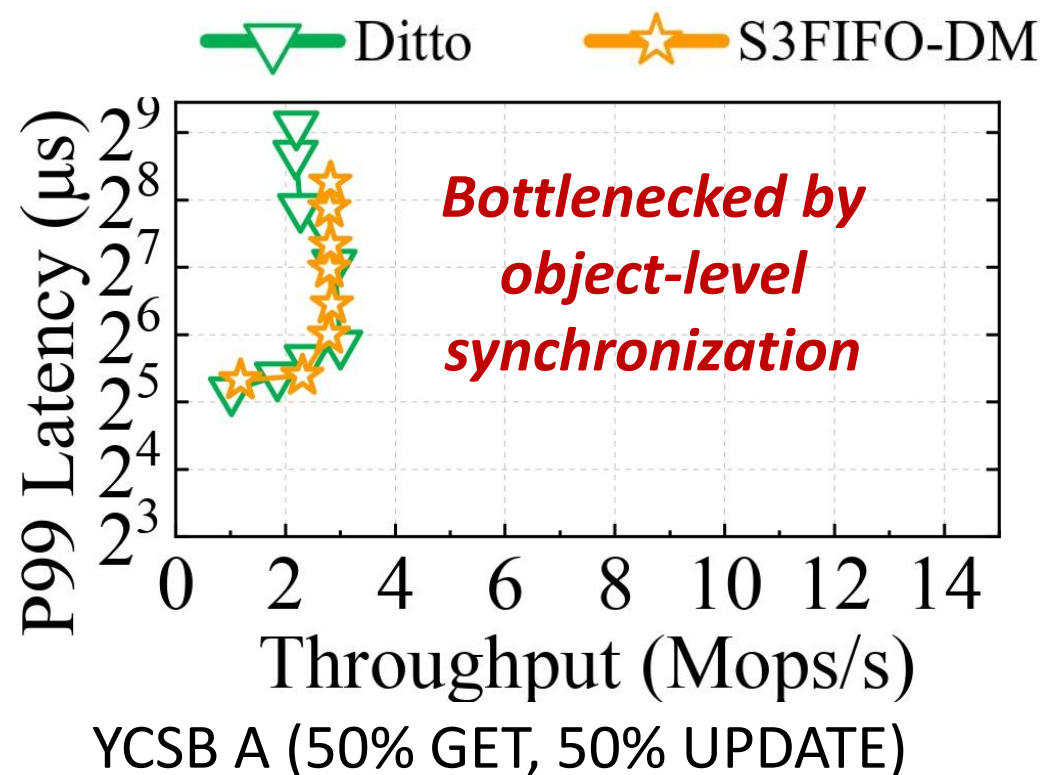
Evaluation: YCSB workloads

- Zipfian distribution, $\theta = 0.99$, object sizes = 256 bytes, cache size = 20%



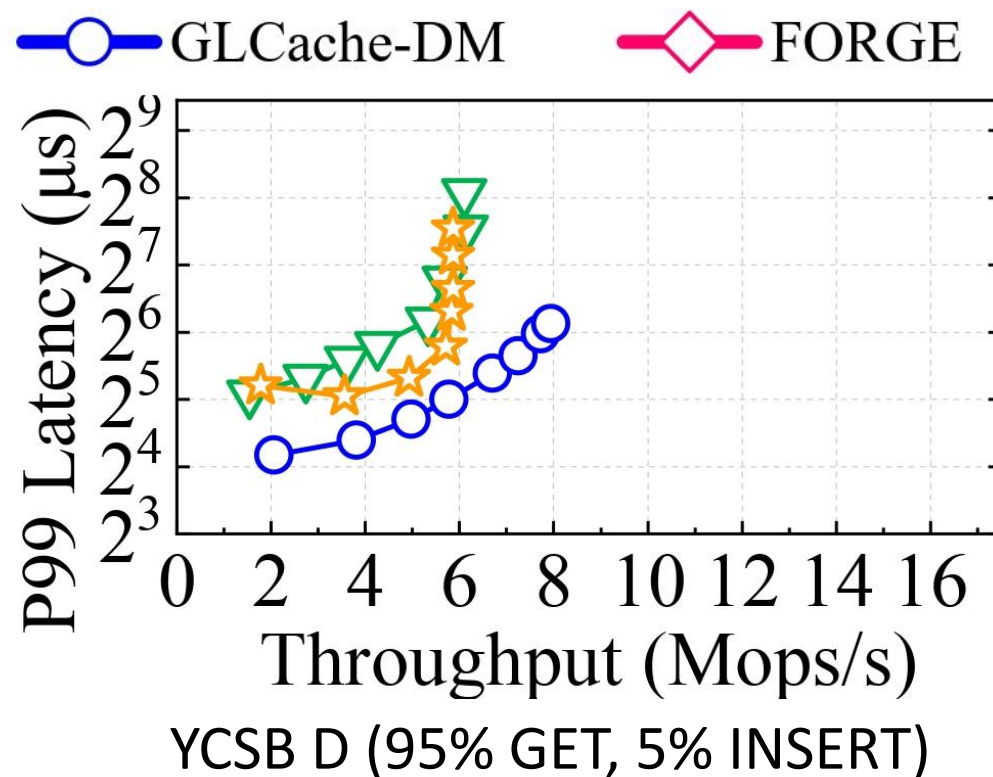
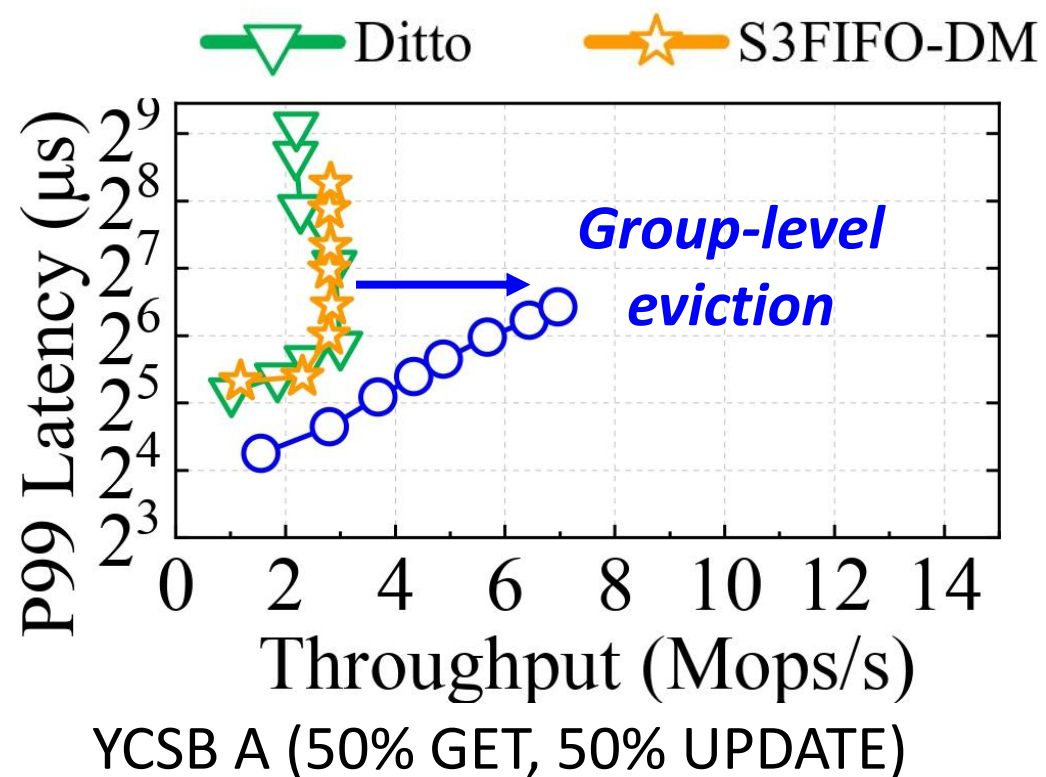
Evaluation: YCSB workloads

- Zipfian distribution, $\theta = 0.99$, object sizes = 256 bytes, cache size = 20%



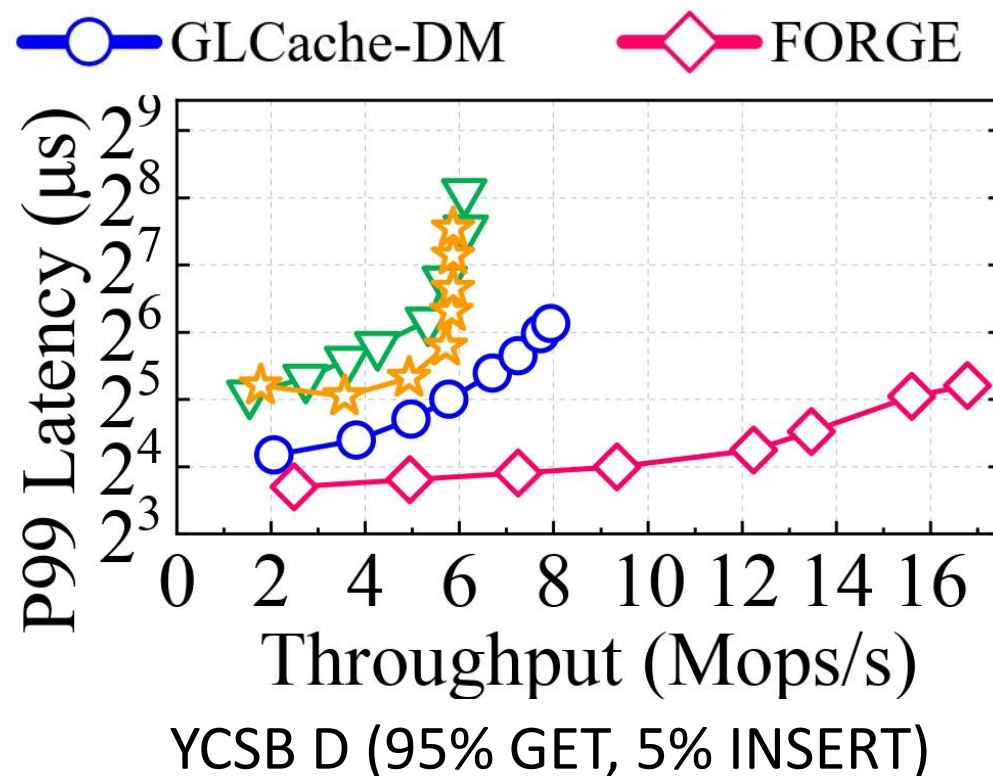
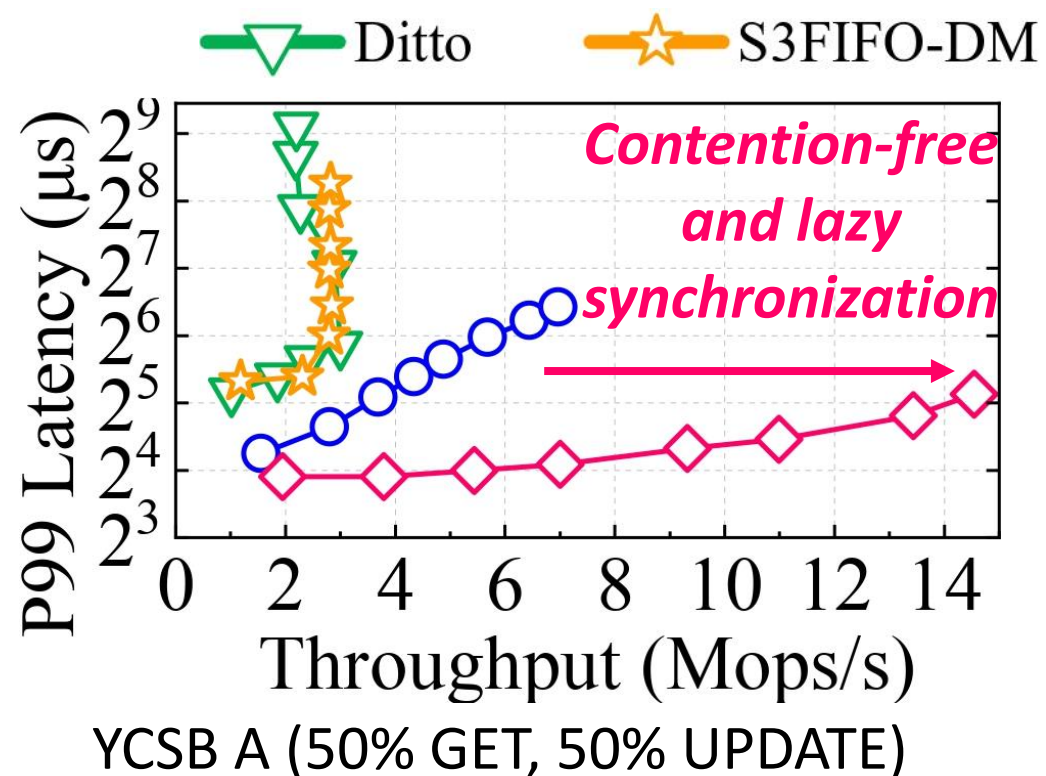
Evaluation: YCSB workloads

- Zipfian distribution, $\theta = 0.99$, object sizes = 256 bytes, cache size = 20%



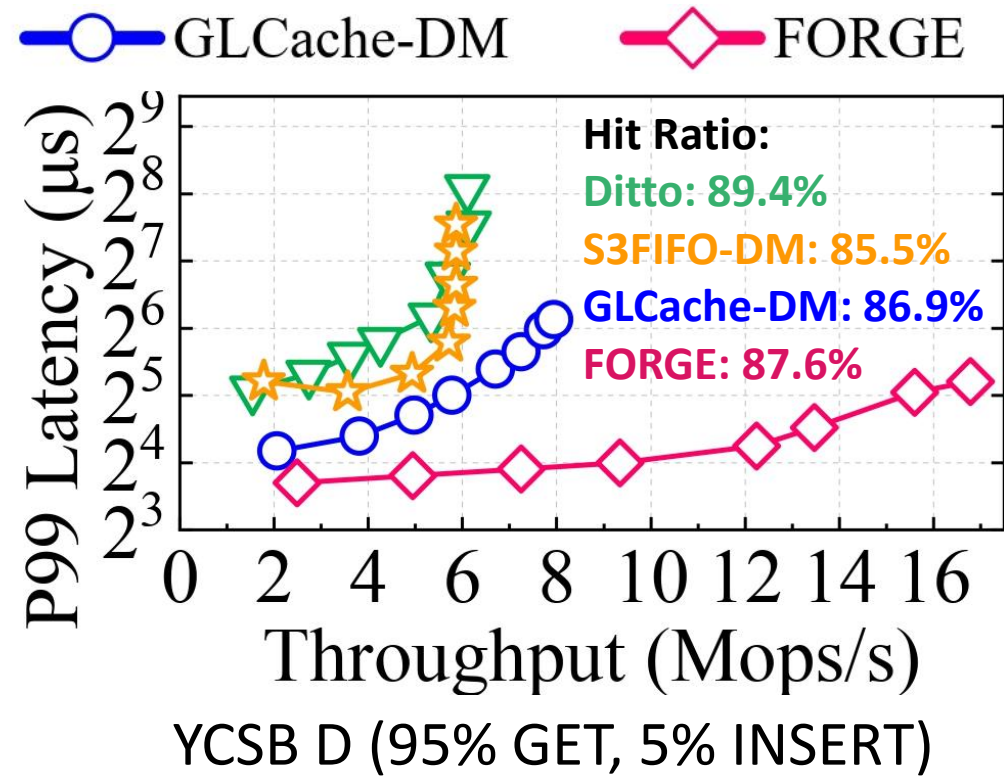
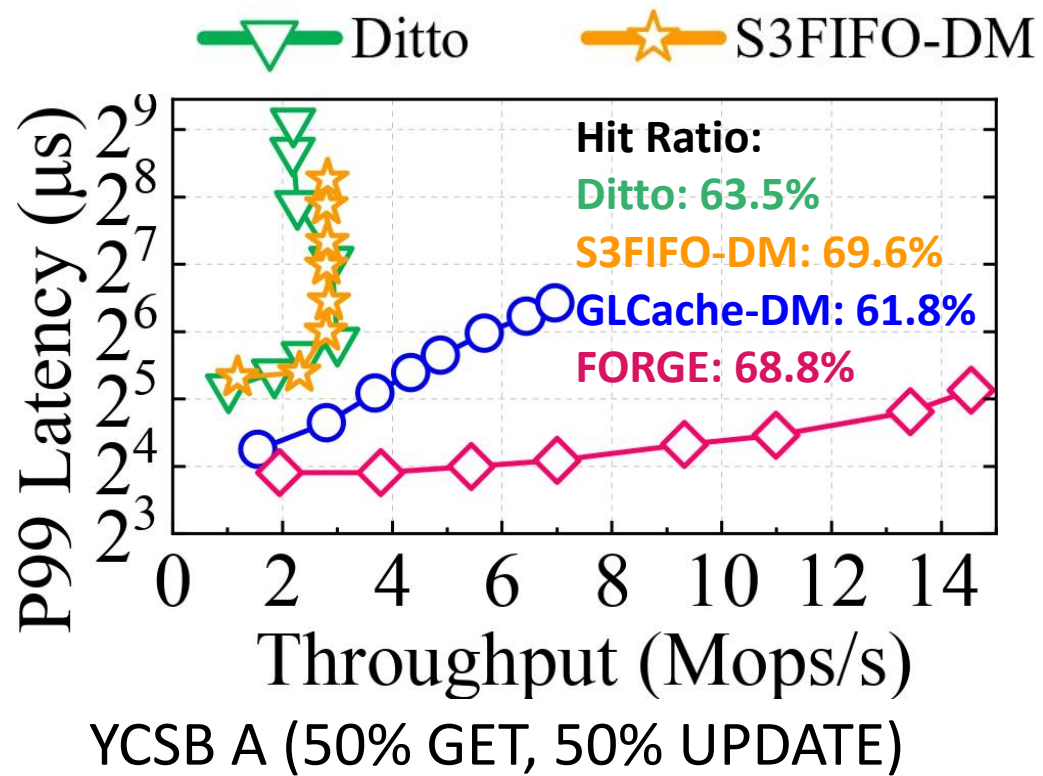
Evaluation: YCSB workloads

- Zipfian distribution, $\theta = 0.99$, object sizes = 256 bytes, cache size = 20%



Evaluation: YCSB workloads

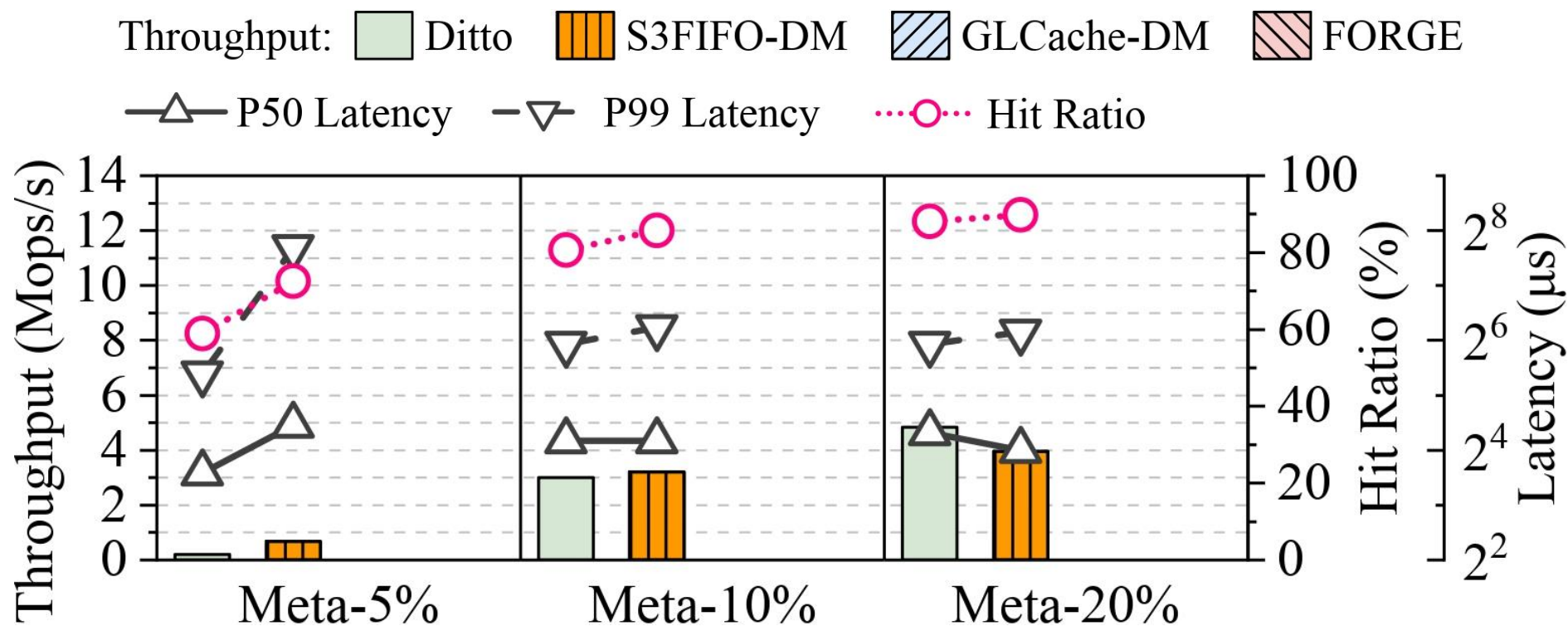
- Zipfian distribution, $\theta = 0.99$, object sizes = 256 bytes, cache size = 20%



FORGE achieves 2.0–8.7× higher throughput and 3.9–13.3× lower P99 latency while maintaining top-2 hit ratios.

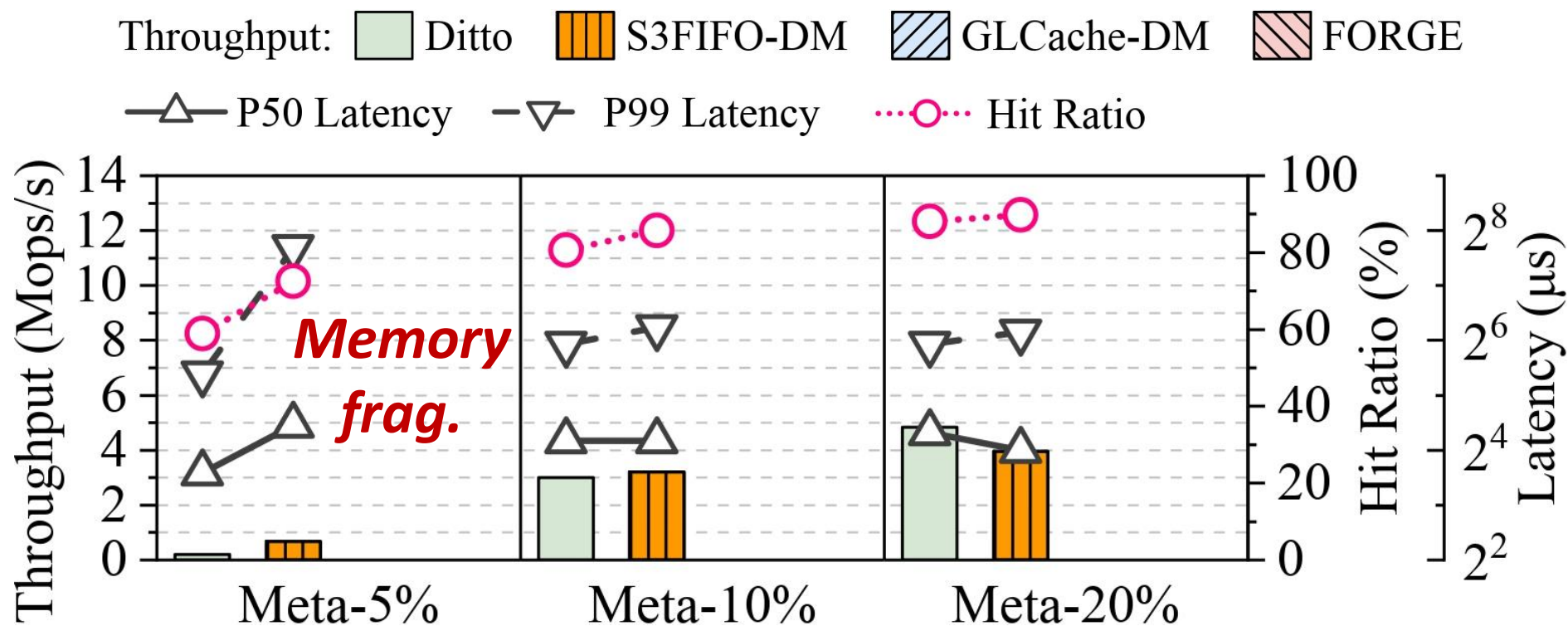
Evaluation: real-world workloads

➤ Dynamic object sizes, cache size = 5%, 10%, 20%



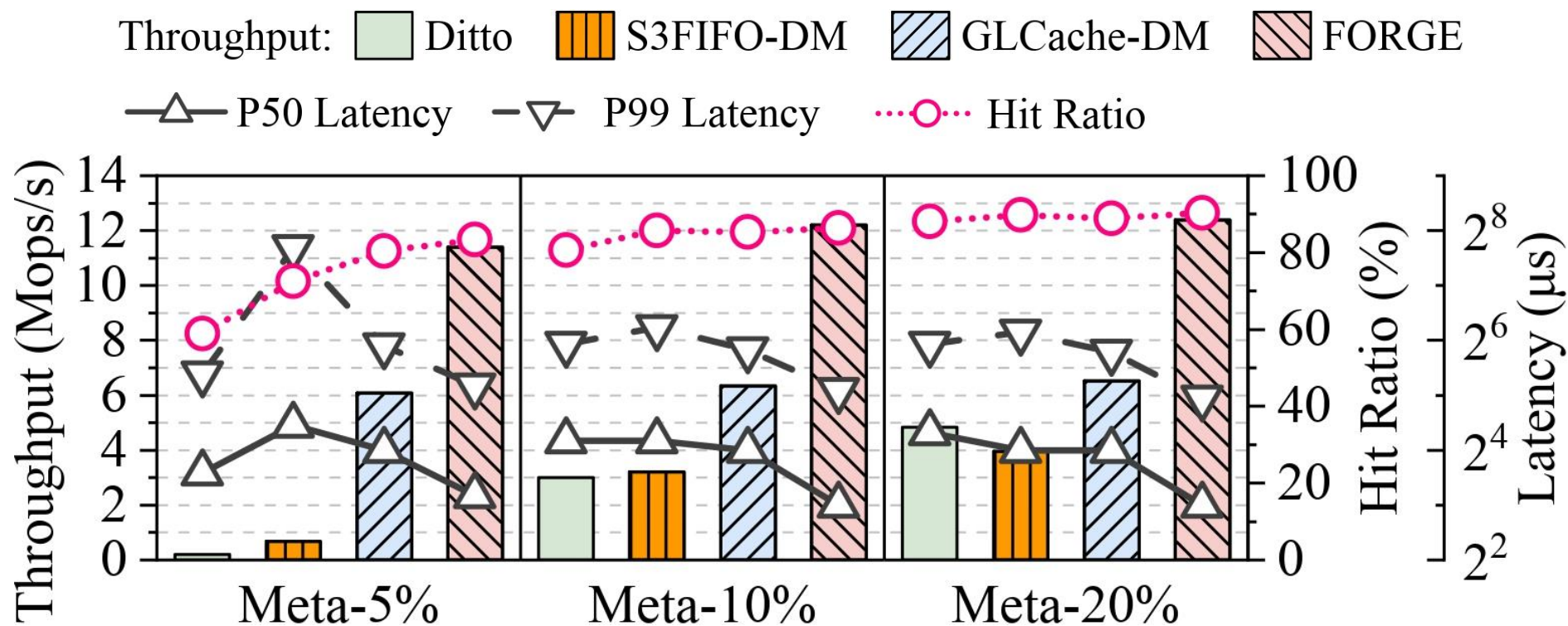
Evaluation: real-world workloads

- Dynamic object sizes, cache size = 5%, 10%, 20%



Evaluation: real-world workloads

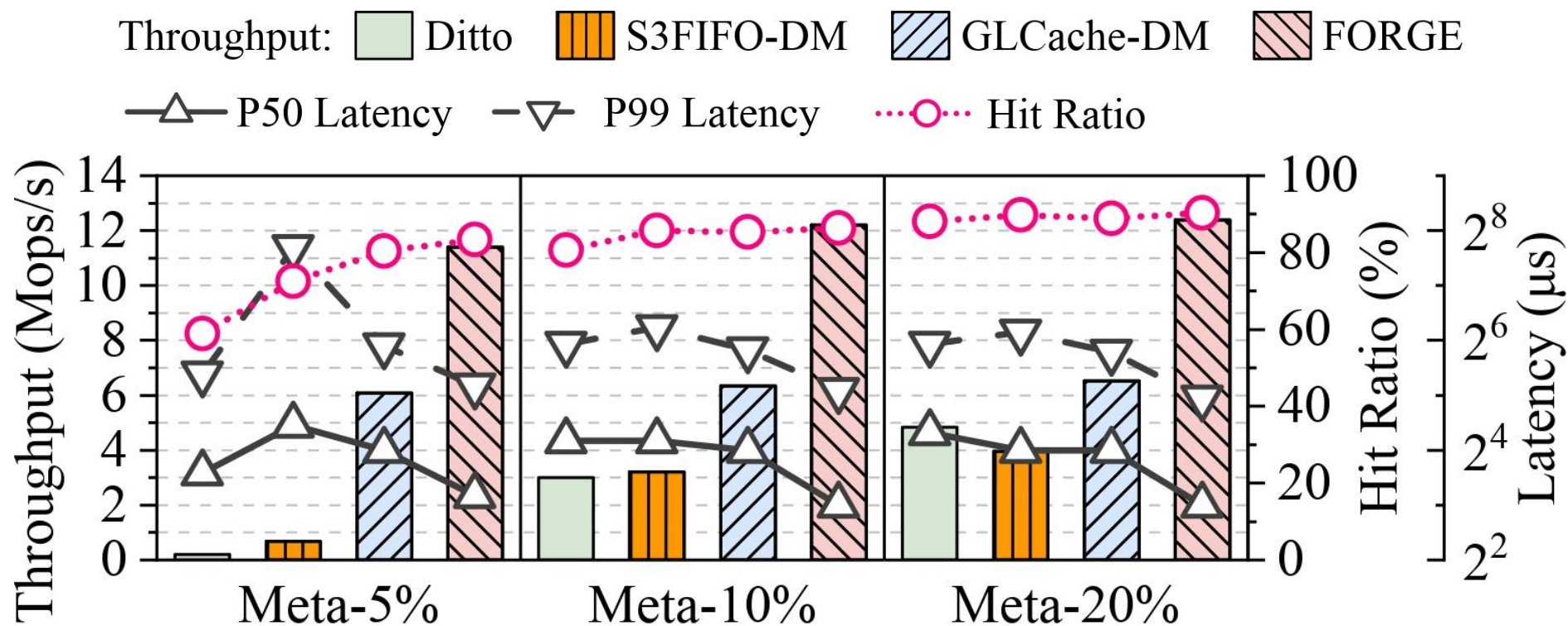
- Dynamic object sizes, cache size = 5%, 10%, 20%



Group-level management reduces fragmentation.

Evaluation: real-world workloads

- Dynamic object sizes, cache size = 5%, 10%, 20%



Compared with the best baseline, FORGE achieves 1.6–1.9× higher throughput and 1.4–2.0× lower latency.

Conclusion

- DM amplifies the synchronization overheads in cache housekeeping
 - Hotness Tracking, Eviction Coordination, Garbage Collection

Conclusion

- DM amplifies the synchronization overheads in cache housekeeping
 - Hotness Tracking, Eviction Coordination, Garbage Collection
- **FORGE**: a DM caching system prioritizing *synchronization efficiency*
 - DM-tailored *group-level* management
 - Contention-free and hotness-aware **FIFO** policy
 - *Lazy-yet-timely* hotness synchronization mechanism

Conclusion

- DM amplifies the synchronization overheads in cache housekeeping
 - Hotness Tracking, Eviction Coordination, Garbage Collection
- **FORGE**: a DM caching system prioritizing *synchronization efficiency*
 - DM-tailored *group-level* management
 - Contention-free and hotness-aware **FIFO** policy
 - *Lazy-yet-timely* hotness synchronization mechanism
- **Benefits**

High Throughput

Low Latency

Comparable Hit Ratios



Thank you! Q&A