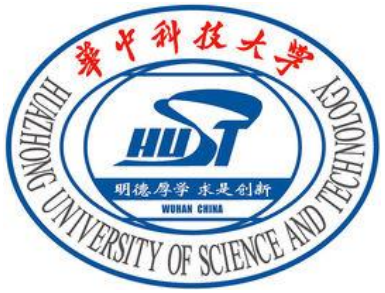


FORGE: Mitigating Synchronization Amplification for Memory-Disaggregated Caching Systems

Zhijun Yang, Yu Hua, Ming Zhang, Menglei Chen, Yixiao Wang

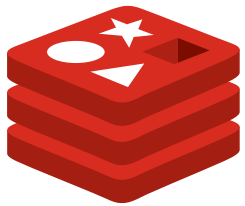
Huazhong University of Science and Technology, China

20th USENIX Symposium on Operating Systems Design and Implementation (**OSDI**), 2026

In-Memory Caching Systems

Fundamental infrastructure in cloud datacenters

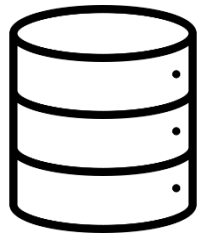
- Cache hot objects in memory
- Serve low-latency Get/Set
- Offload backend storage



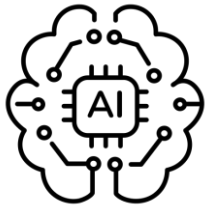
redis

MEMCACHED

Widely used in:



Databases

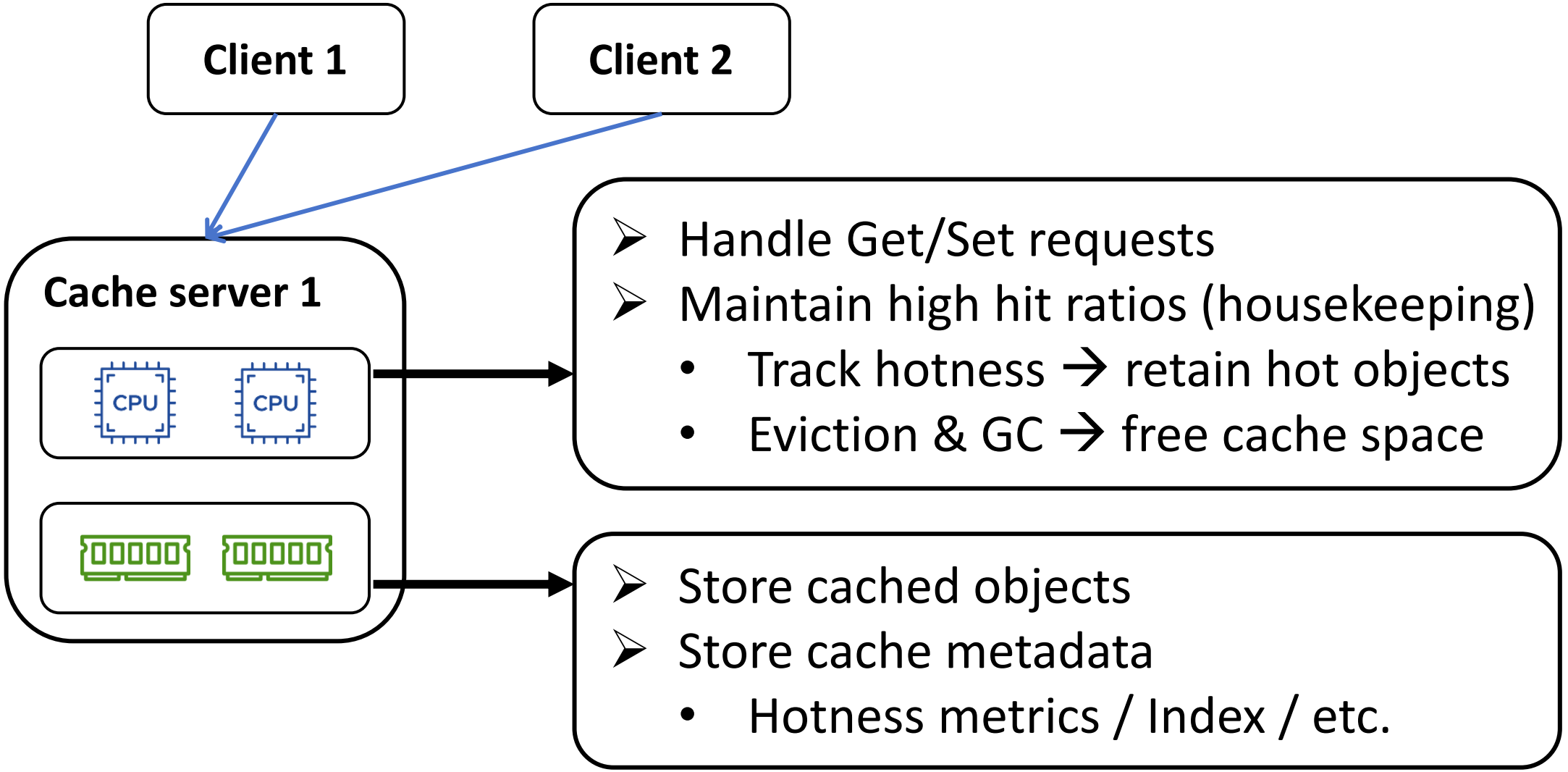


Machine Learning

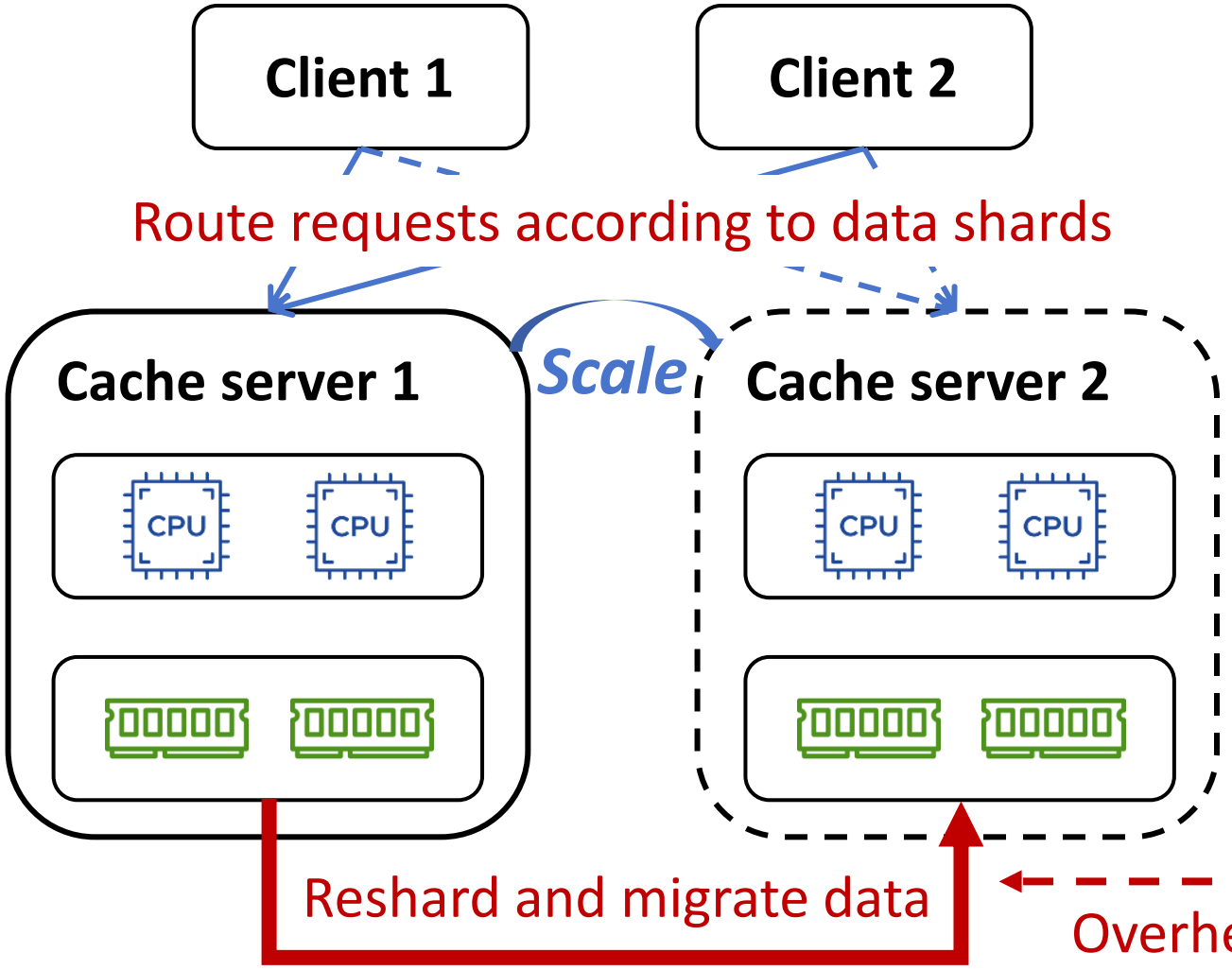
Serverless Computing

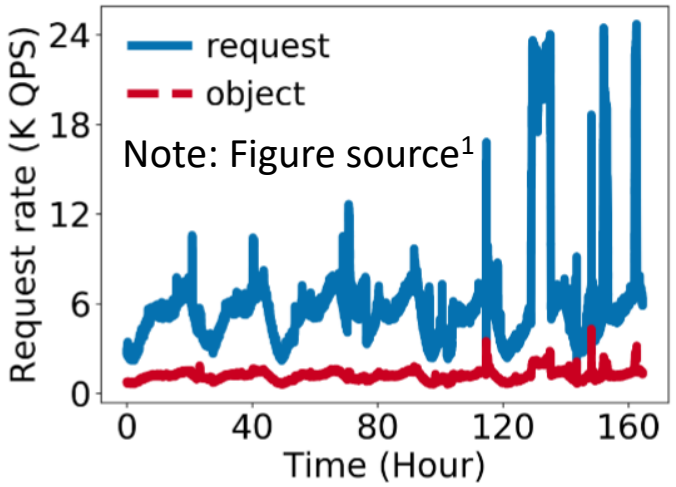
...

Caching Systems on Monolithic Servers



Scaling Monolithic Caching Systems





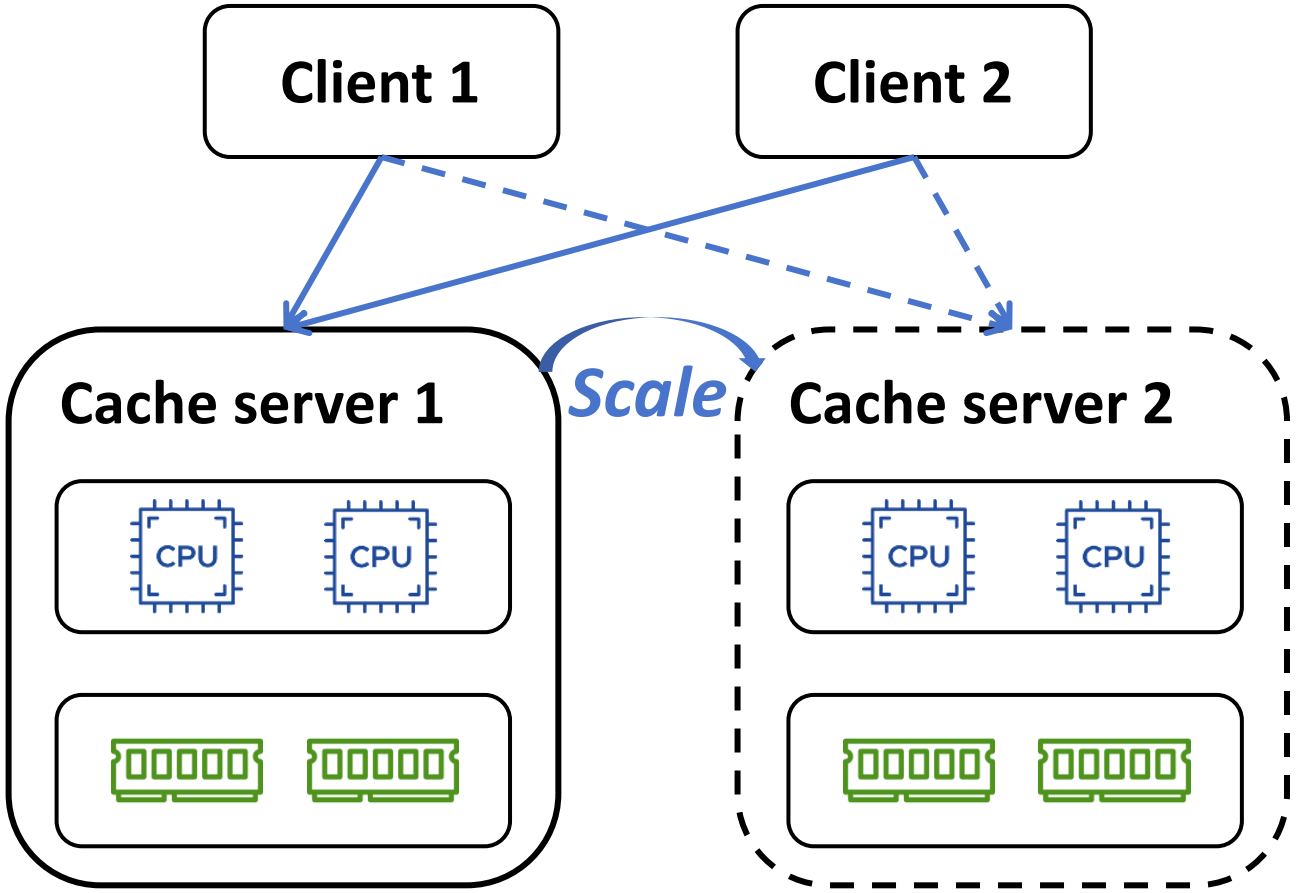
Dynamic resource demands

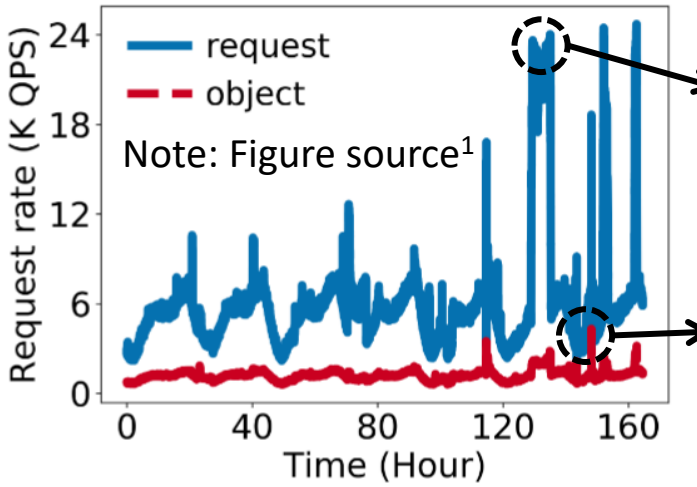
Need higher throughput or larger cache capacity?

Scale at instance granularity

[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





❶Need higher throughput

❷Need larger cache size

❶ Need more CPUs

❷ Need more memory

Monolithic scaling adds both CPUs & memory

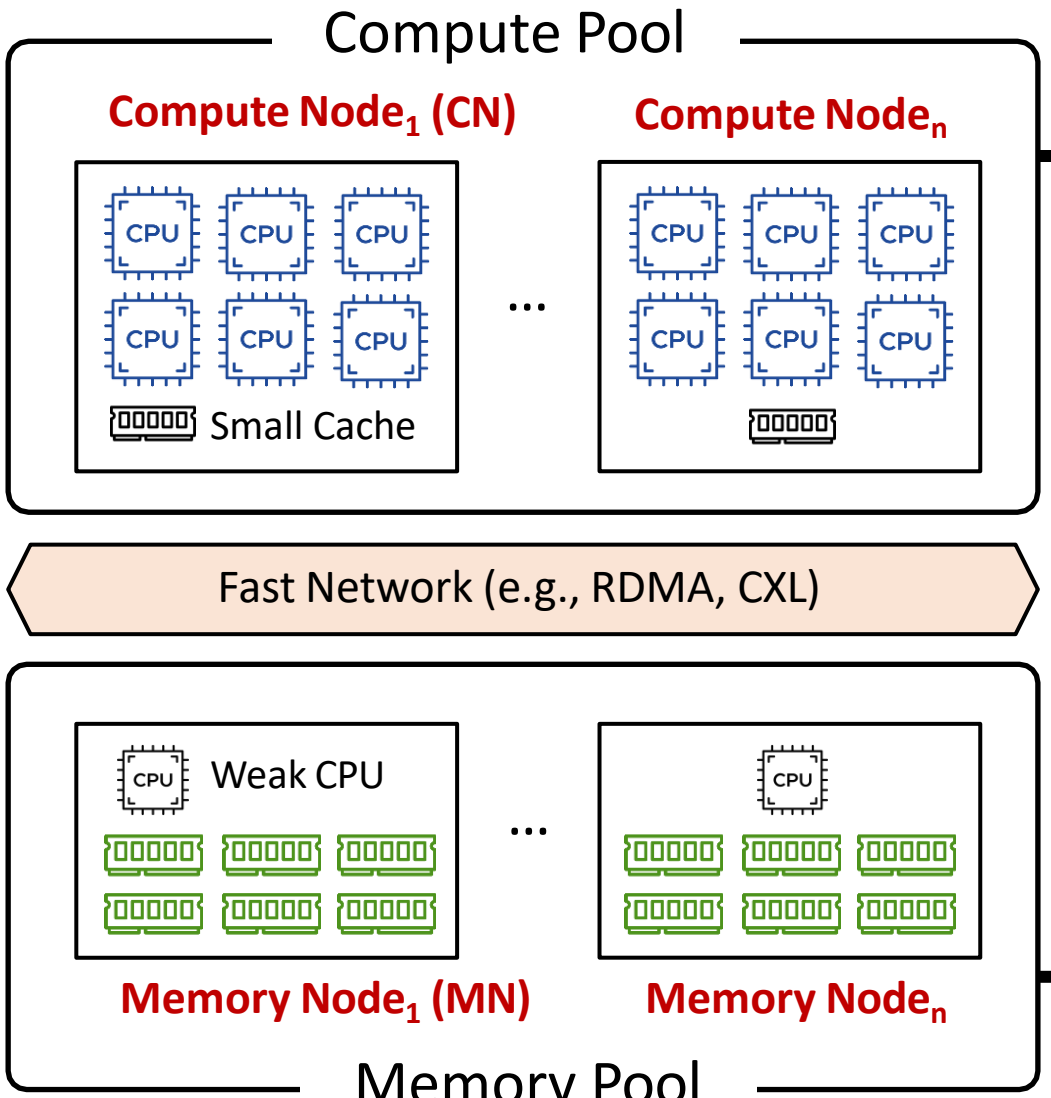
Waste memory

Waste CPUs

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

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

Disaggregated Memory (DM) for Caching



- Handle Get/Set requests
- Cache housekeeping for high hit ratios

❶ Need higher throughput → add CNs

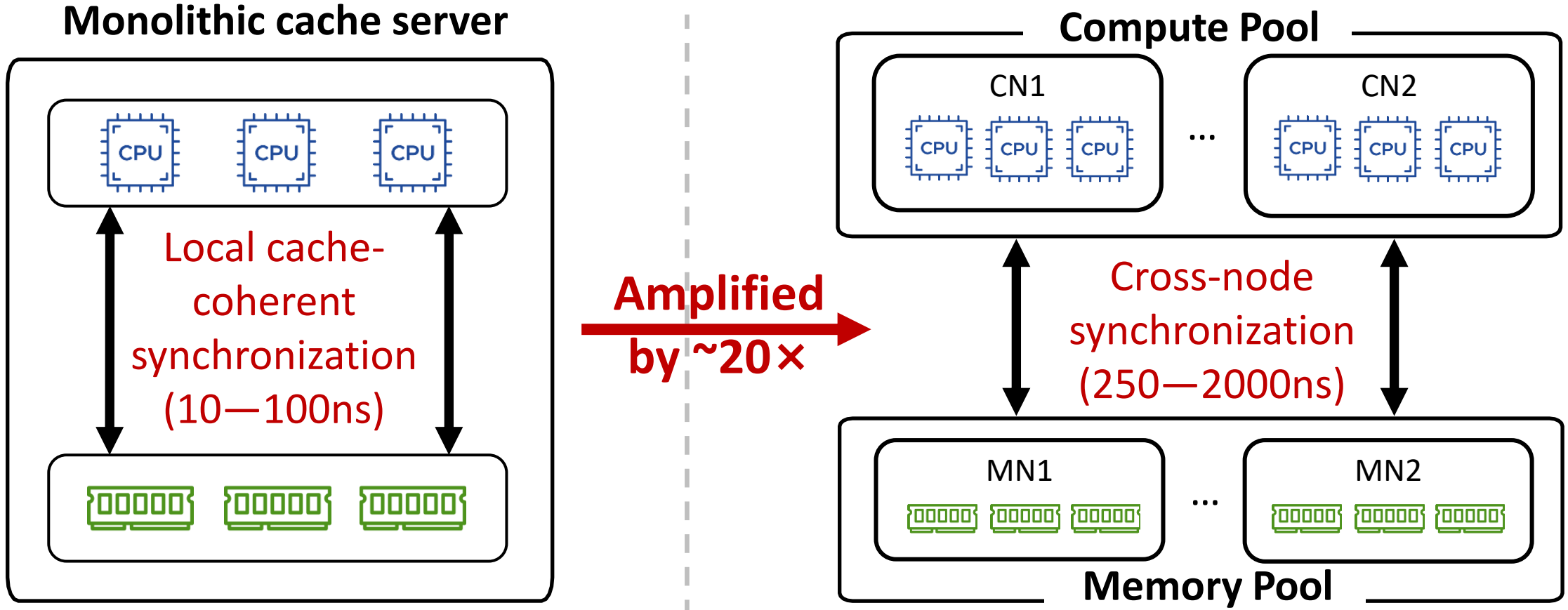
Benefits: higher resource utilization & elasticity

❷ Need larger cache capacity → add MNs

- Store cached objects and metadata
- Shared across CNs to avoid migration

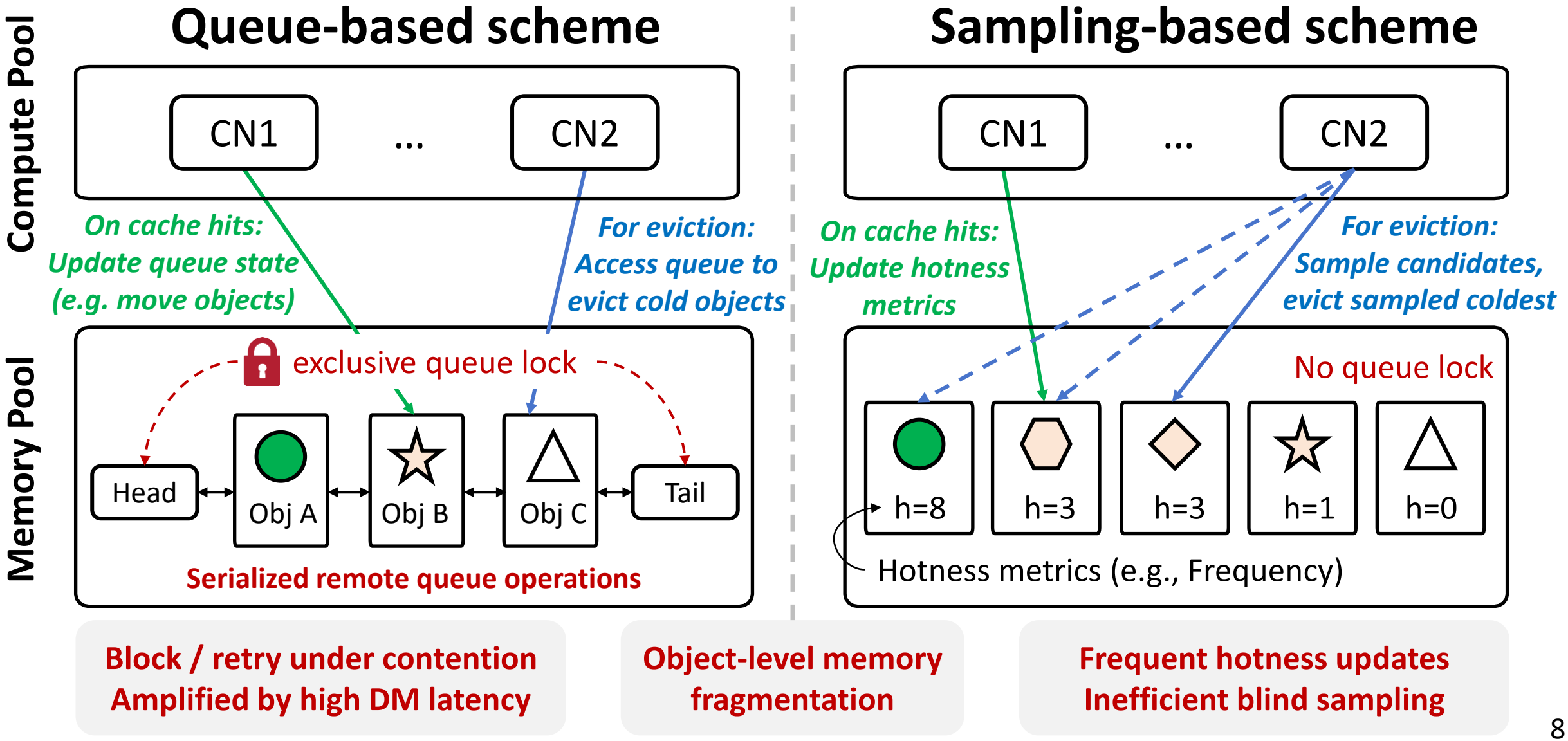
The Cost: Synchronization Amplification

Cache housekeeping: Hotness Tracking / Eviction Coordination / Garbage Collection



DM turns lightweight housekeeping into expensive cross-node synchronization.

Why DM Cache Housekeeping Is Costly

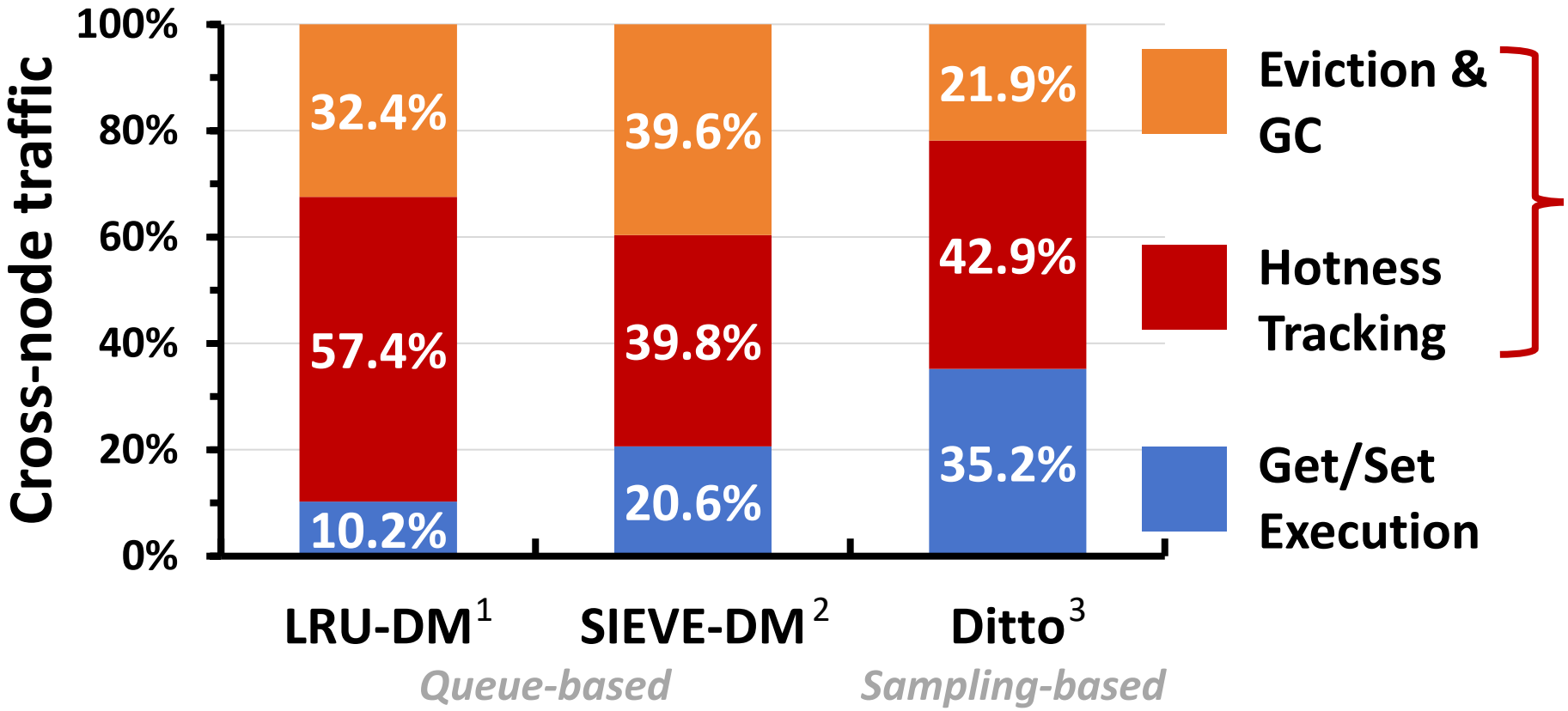


Block / retry under contention
Amplified by high DM latency

Object-level memory fragmentation

Frequent hotness updates
Inefficient blind sampling

Synchronization Amplification is Severe

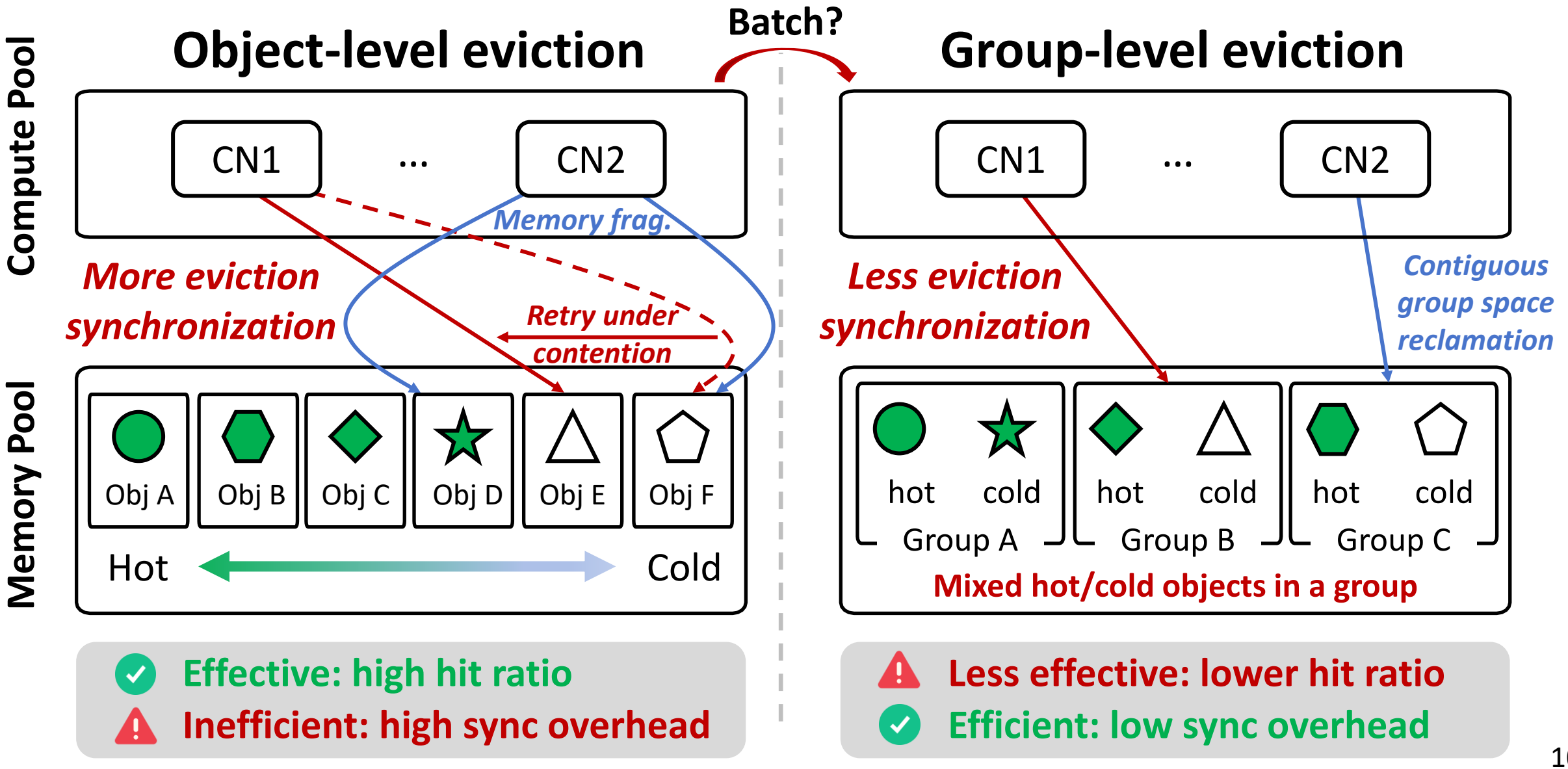


Housekeeping traffic can exceed useful Get/Set traffic

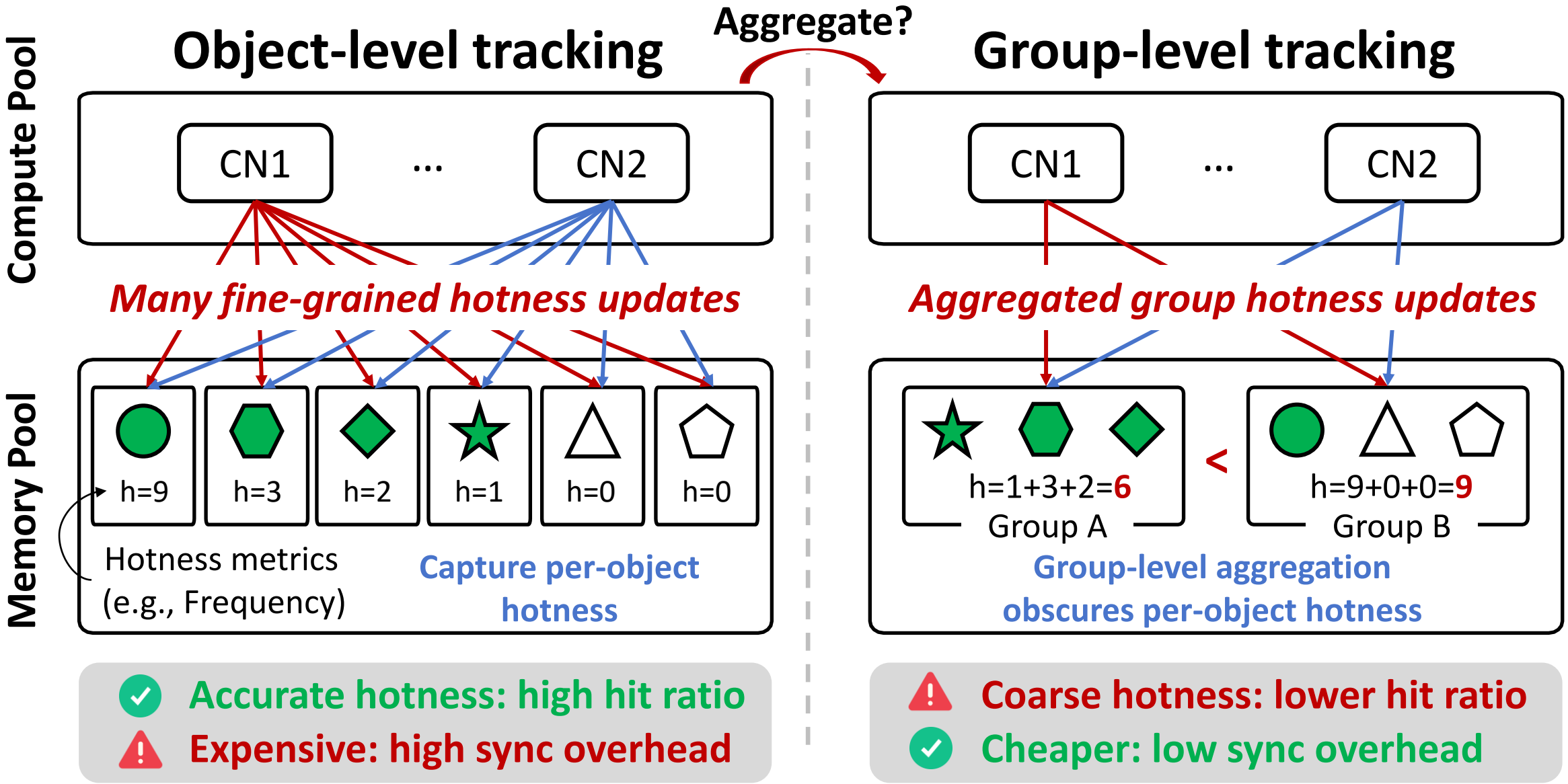
How can we redesign DM cache housekeeping for efficient synchronization?

[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.

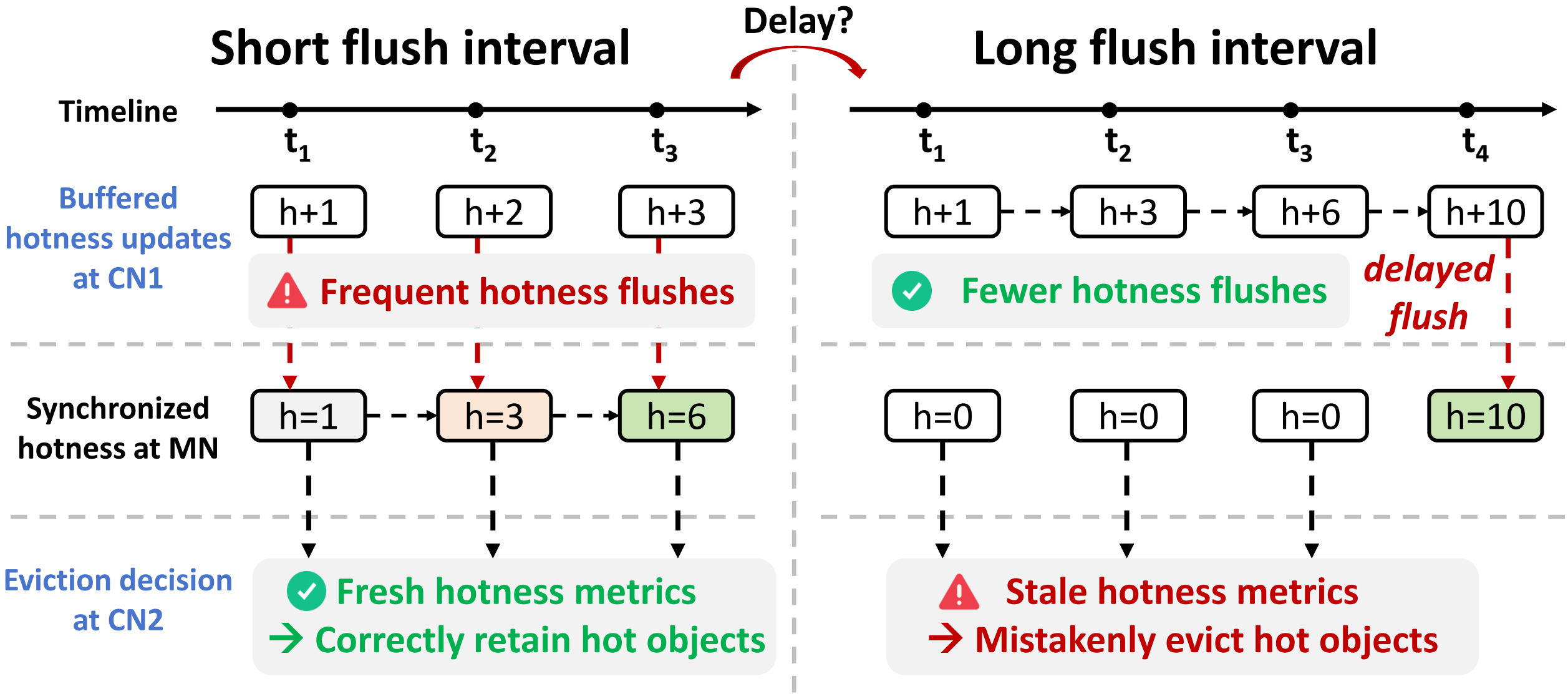
C1: Efficiency-Effectiveness Trade-off in Eviction



C2: Accuracy–Overhead Dilemma in Hotness Tracking

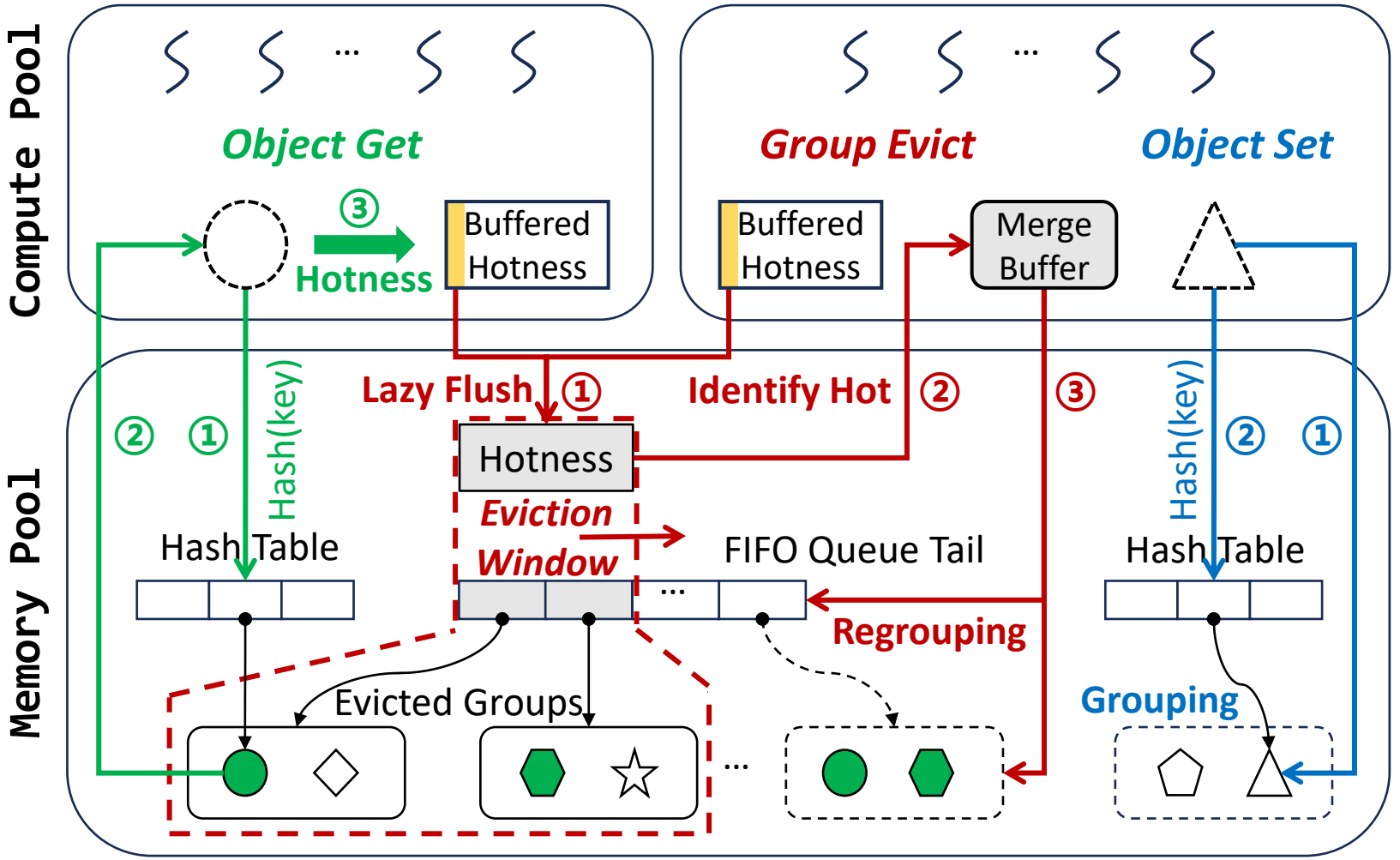


C3: Timeliness–Cost Conflict in Synchronization



FORGE Overview

Principle: prioritize synchronization efficiency

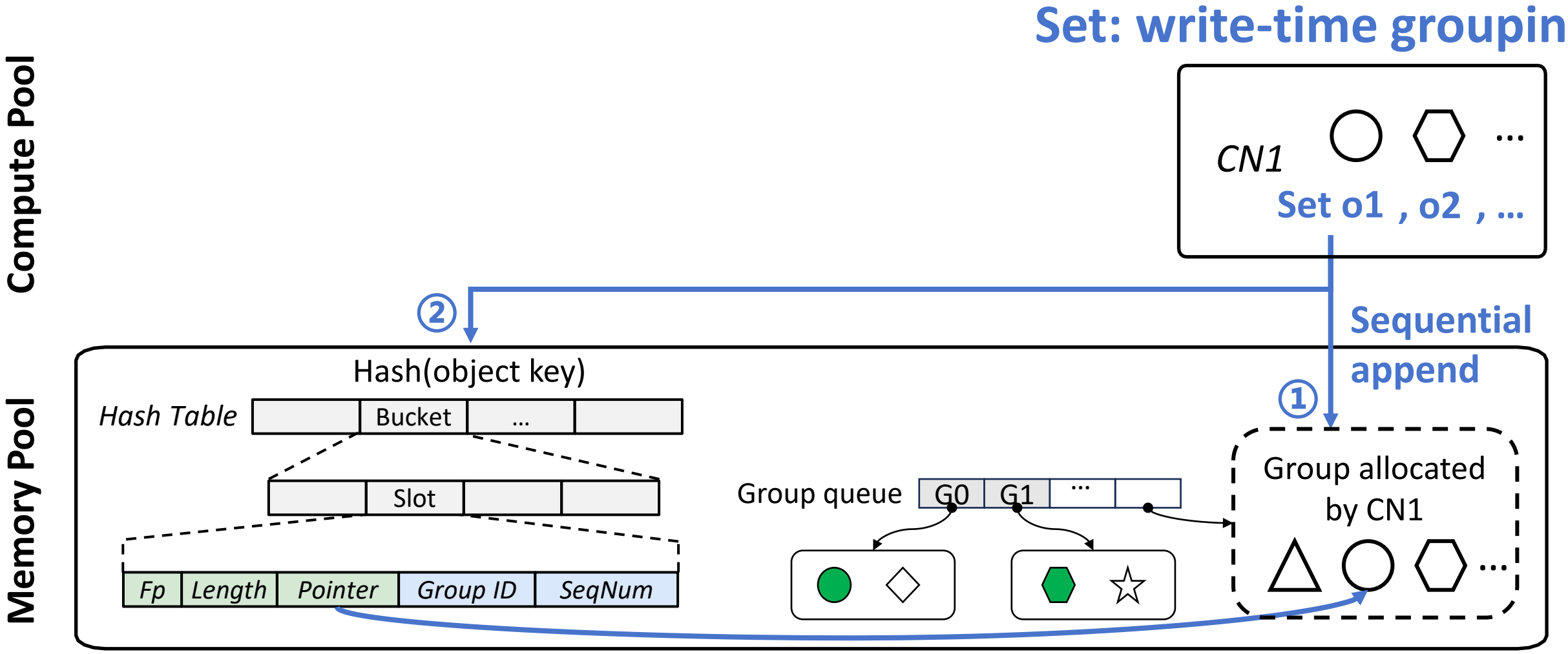


Group-level cache management

Contention-free, hotness-aware FIFO

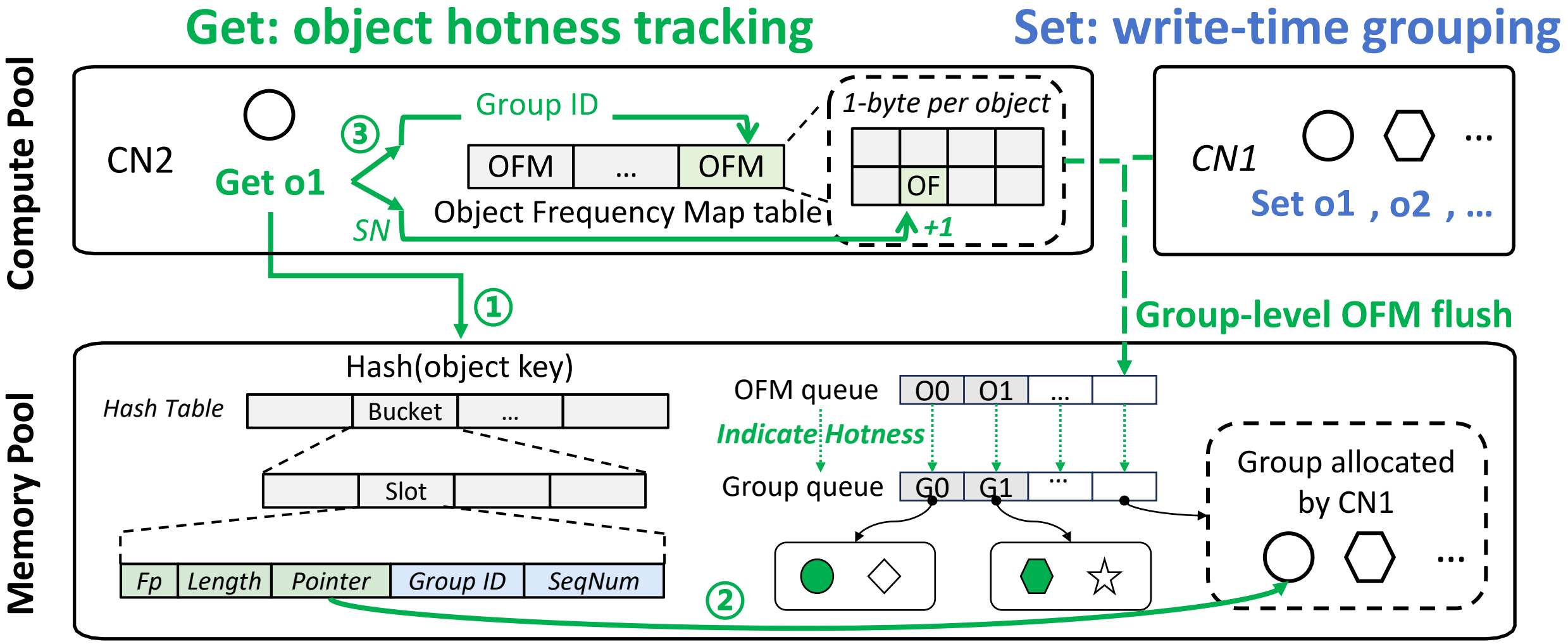
Object-level tracking, lazy hotness sync

Grouping-aware Get/Set



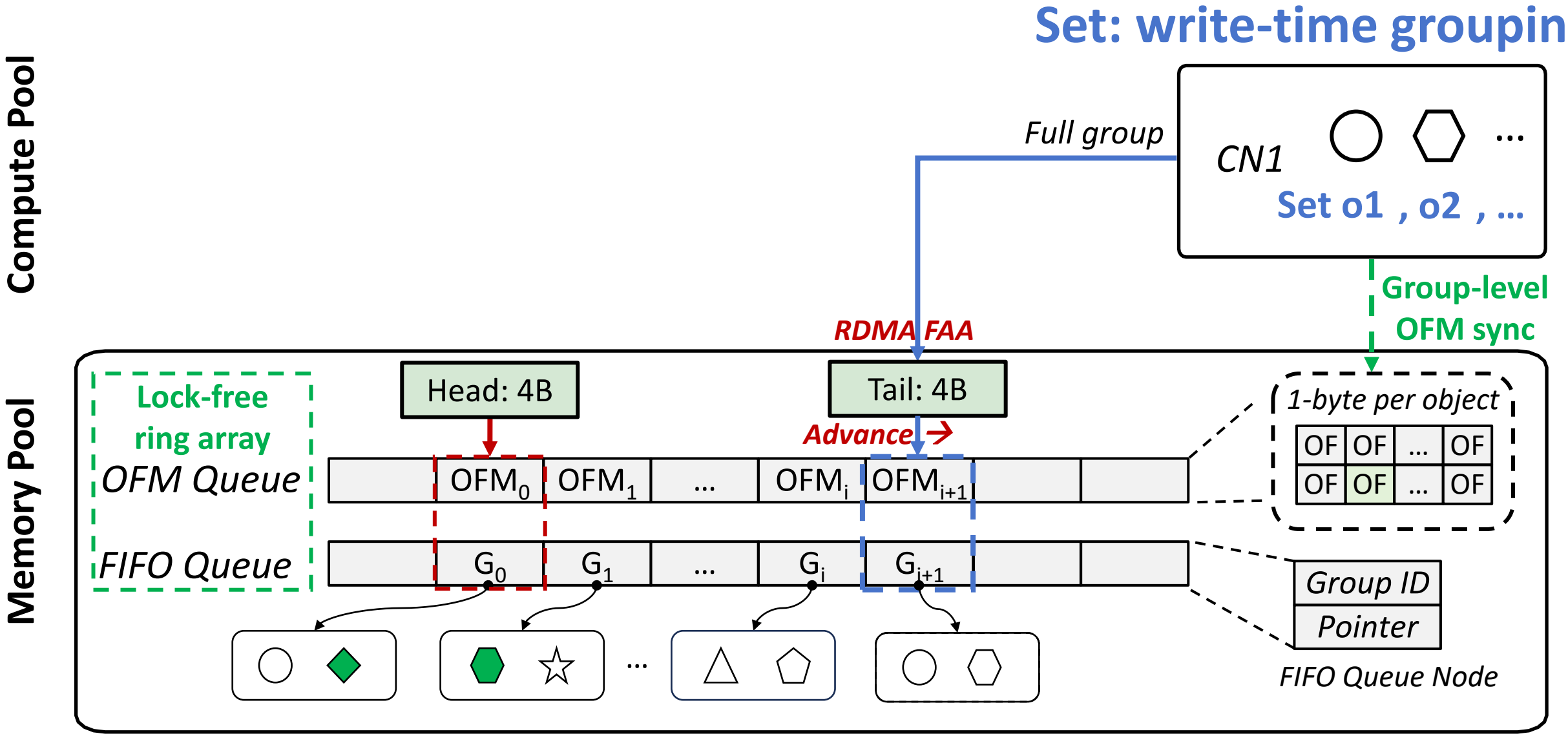
Temporal locality: objects written close in time often exhibit correlated access patterns.

Grouping-aware Get/Set

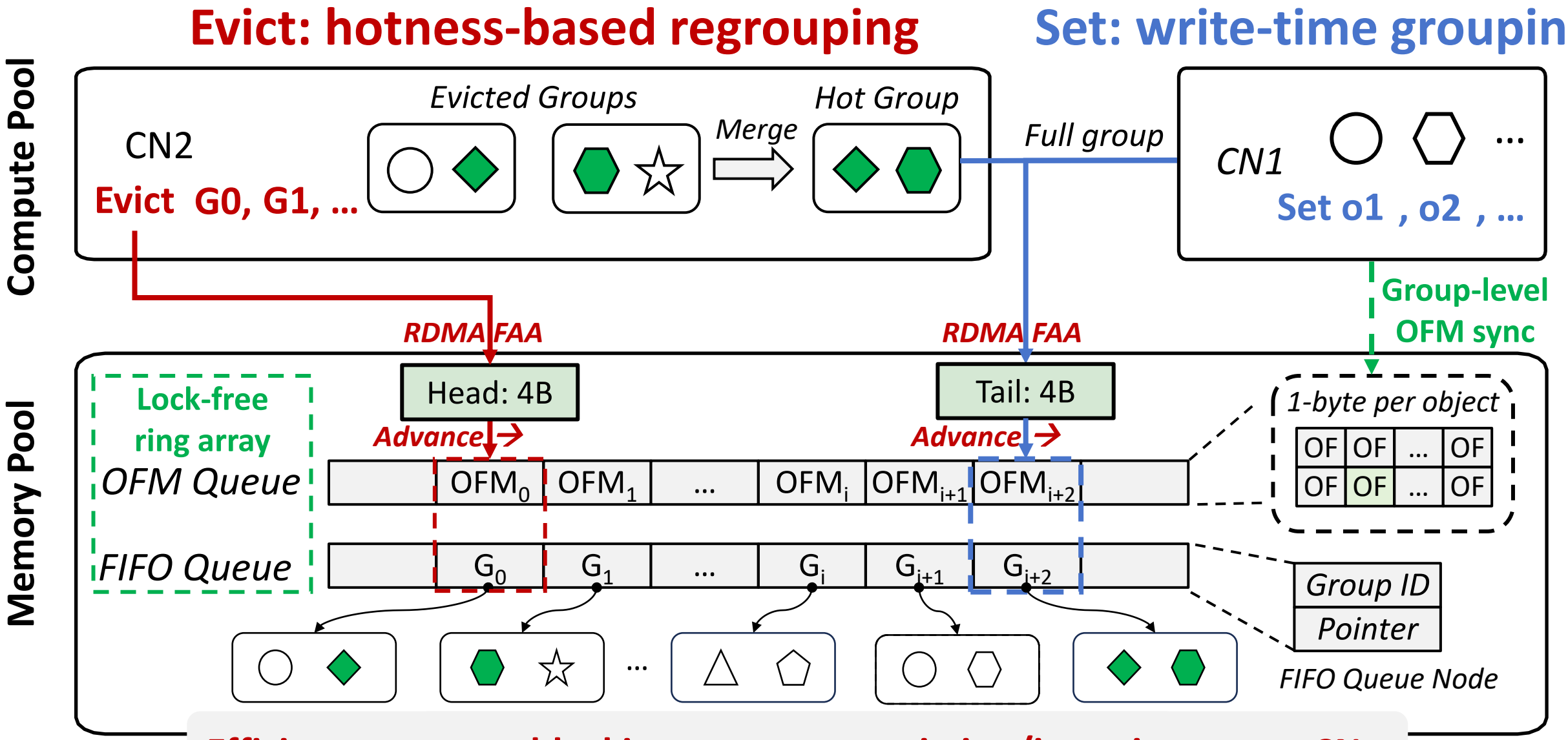


Hybrid granularity: accurate object-level hotness tracking, batched group-level sync.

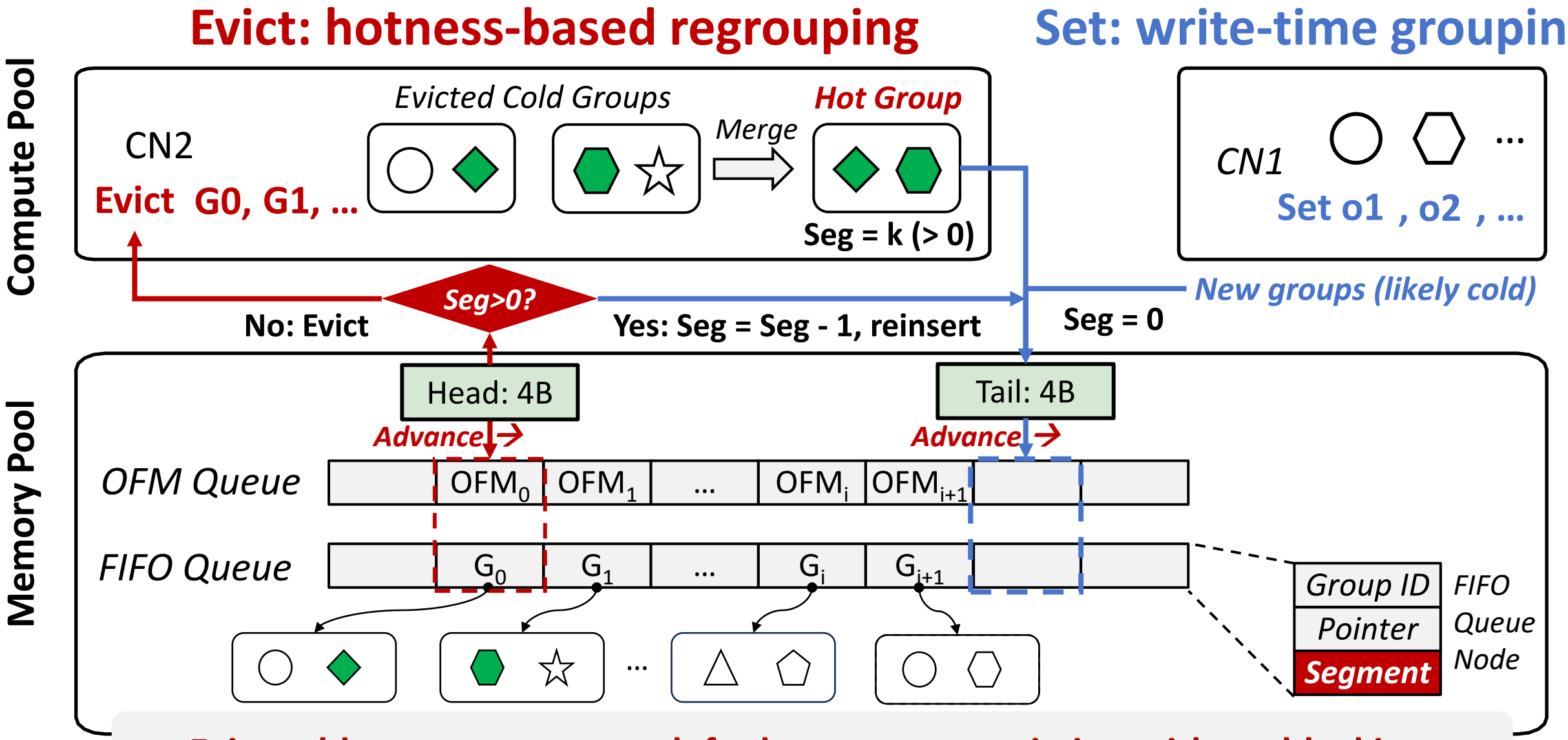
Contention-free FIFO Management



Contention-free FIFO Management

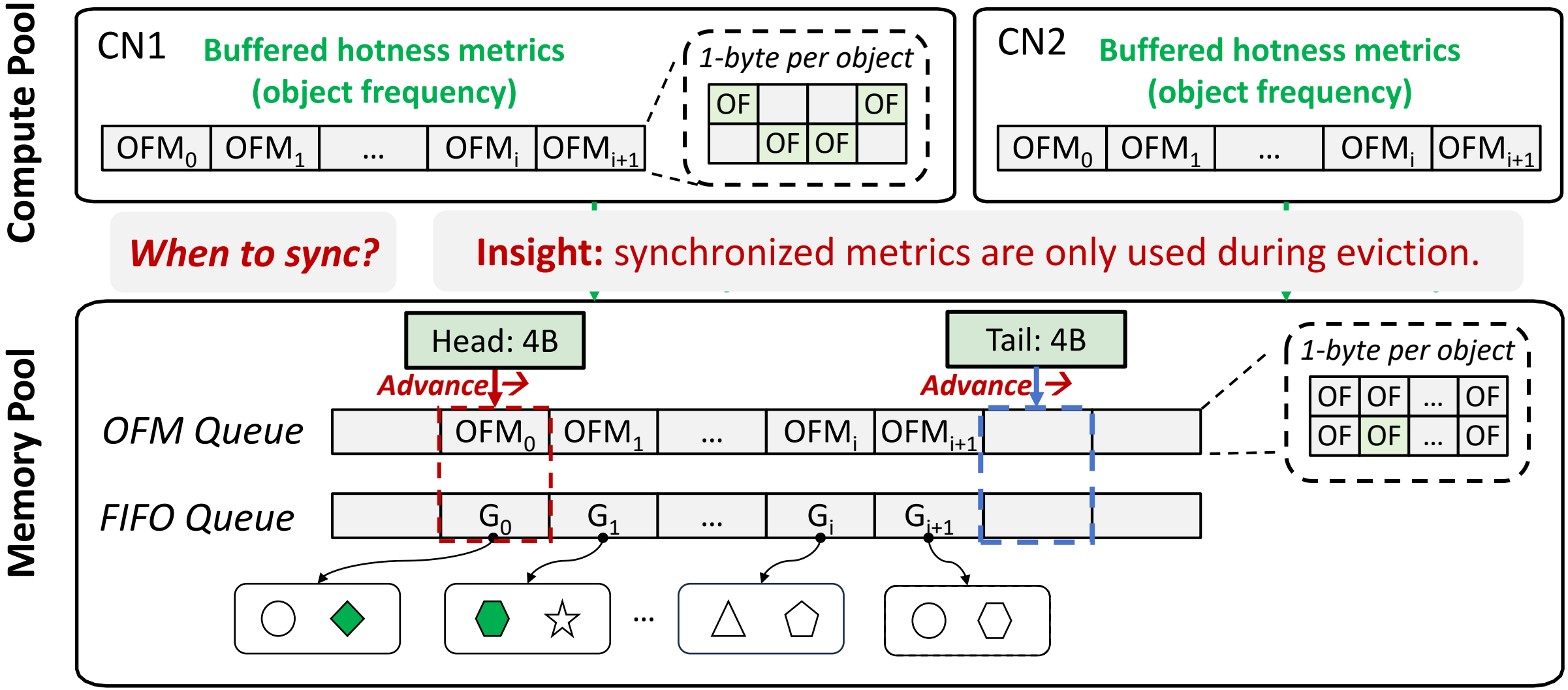


Hotness-aware FIFO Segmentation

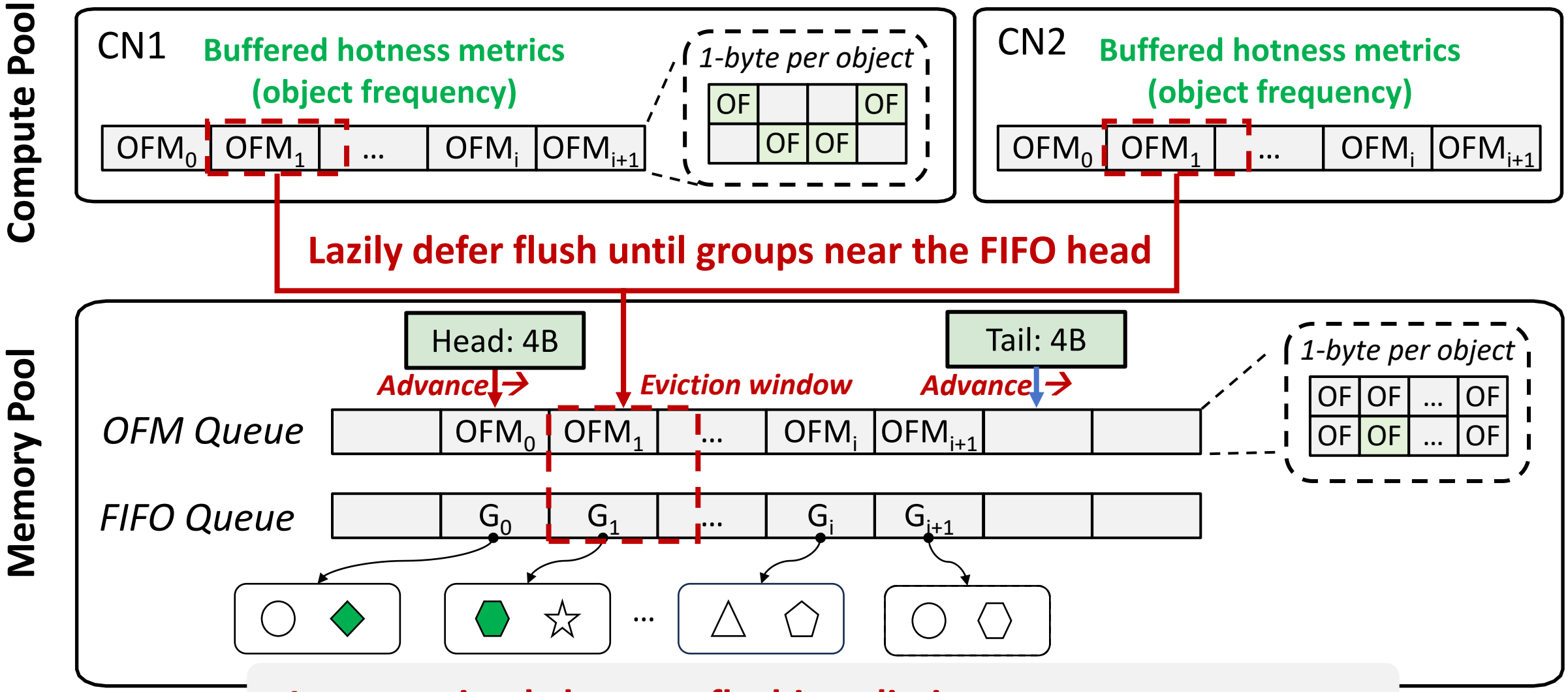


Evict colder groups sooner; defer hotter-group eviction without blocking.

Lazy Hotness Synchronization



Lazy Hotness Synchronization



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
- …

FORGE: Mitigating Synchronization Amplification for Memory-Disaggregated Caching Systems

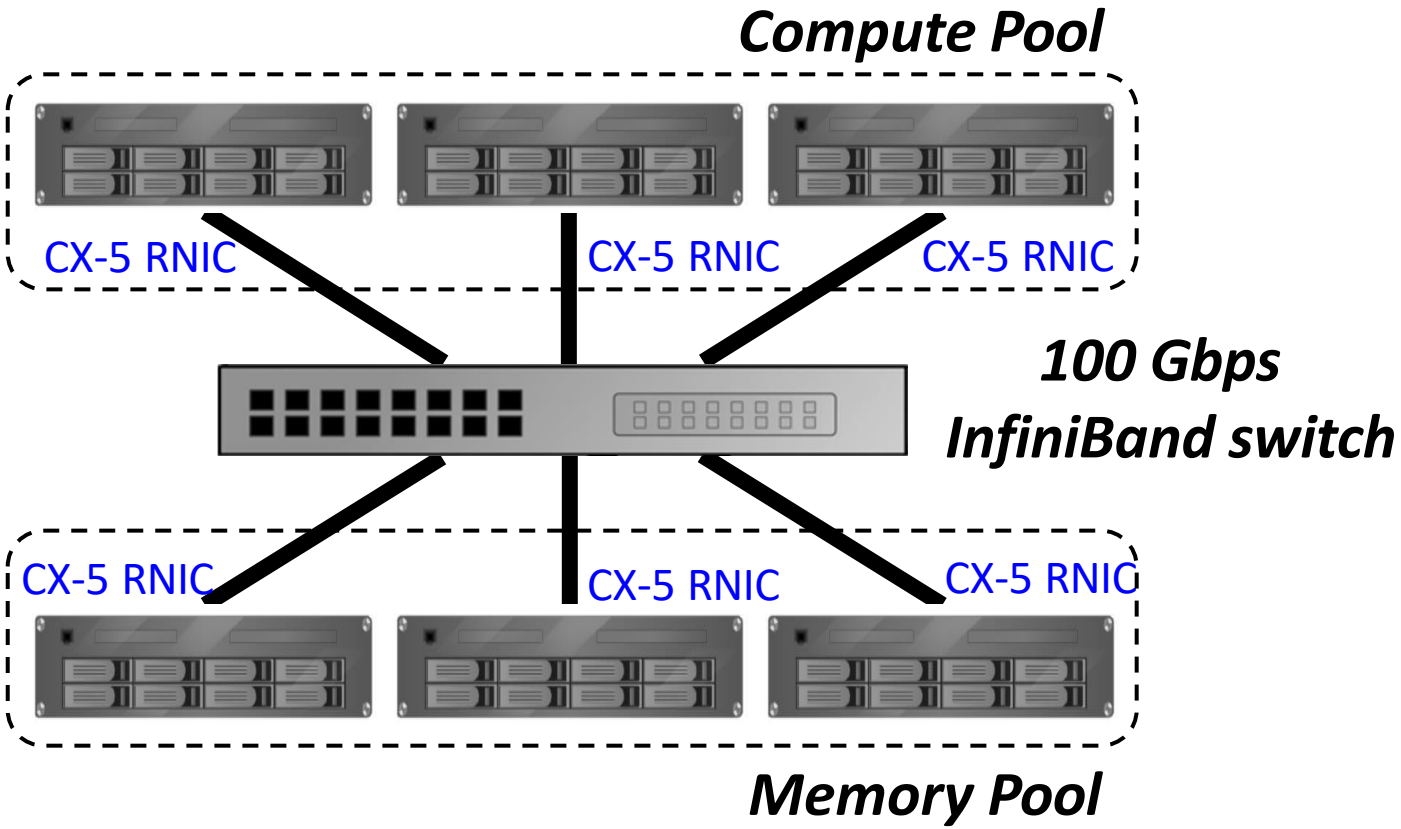
Zhijun Yang, Yu Hua*, Ming Zhang, Menglei Chen, Yixiao Wang
Huazhong University of Science and Technology
*Corresponding Author: Yu Hua (csyhua@hust.edu.cn)

Abstract

1 Introduction

Evaluation

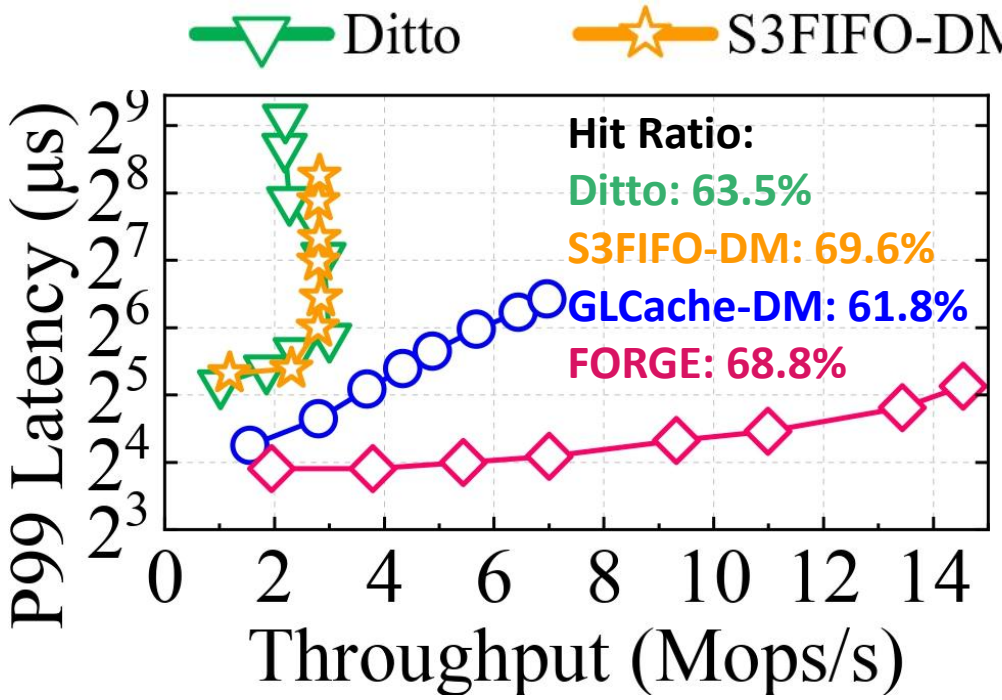
- **Workloads**
  - YCSB A/B/C/D/E
  - CloudPhysics
  - Wikimedia
  - Microsoft Research Cambridge
  - IBM
  - Meta
  - Twitter



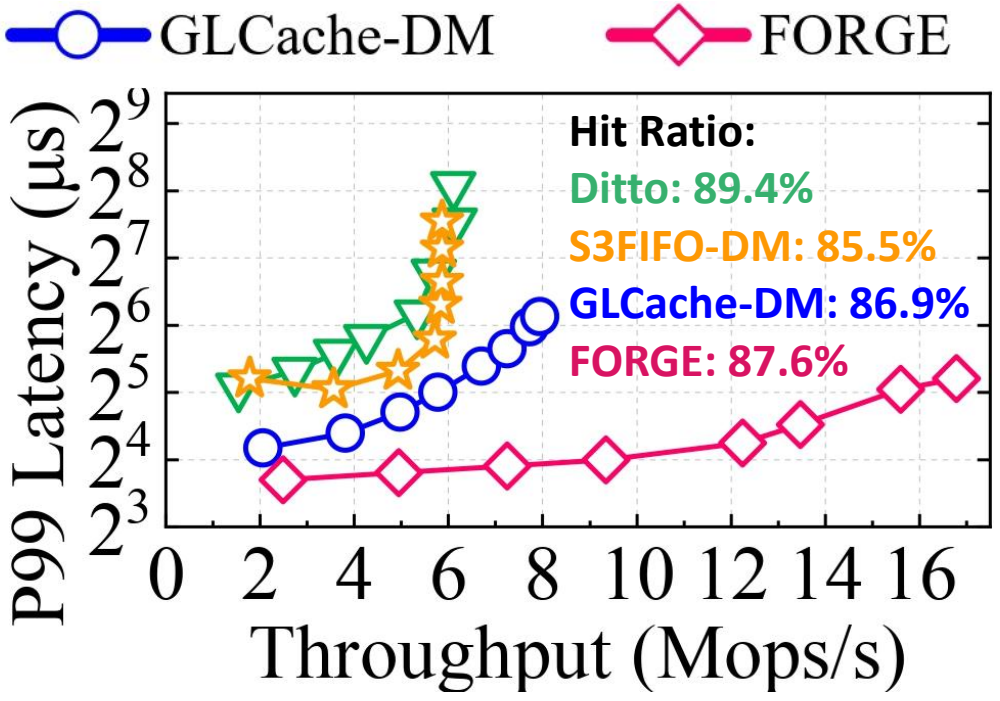
- **Comparisons**
  - Ditto [SOSP ’23] (state-of-the-art in RDMA-based DM caching systems)
  - S3FIFO-DM (adapted from S3-FIFO [SOSP ’23], a recent object-level FIFO scheme)
  - GLCache-DM (adapted from GL-Cache [FAST ’24], a recent group-level scheme)

Evaluation: YCSB workloads

- **Zipfian distribution, θ = 0.99, object sizes = 256 bytes, cache size = 20%**



YCSB A (50% GET, 50% UPDATE)

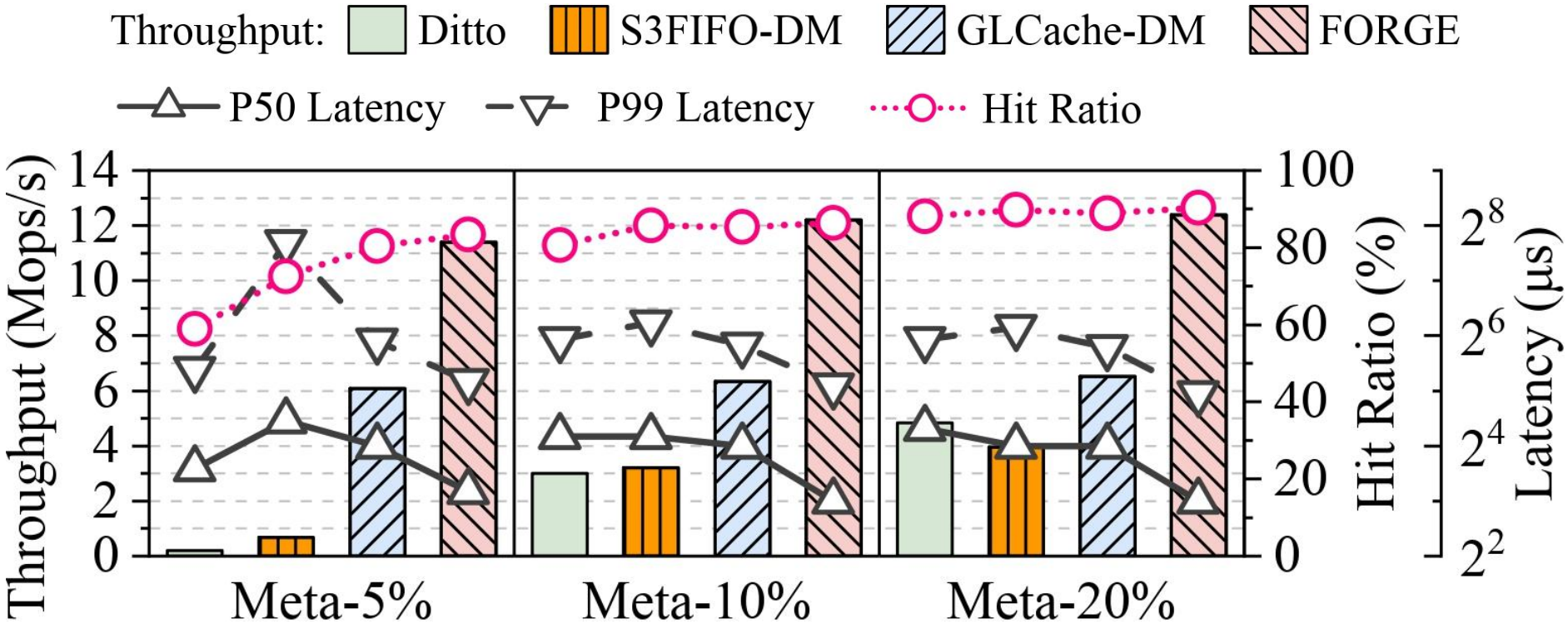


YCSB D (95% GET, 5% INSERT)

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%**



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
- ***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**

https://github.com/cszjyang/forge

Thank you! Q&A