INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING
ICPP/2021/CHICAGO/USA
acm In-Cooperation
sighpc
AUGUST 9-12, 2021

Dual-KV : Improving Performance of Key-value Caching Systems on Non-volatile Memory

Zong-Ming Ke, Yun-Ze Li, Da-Wei Chang

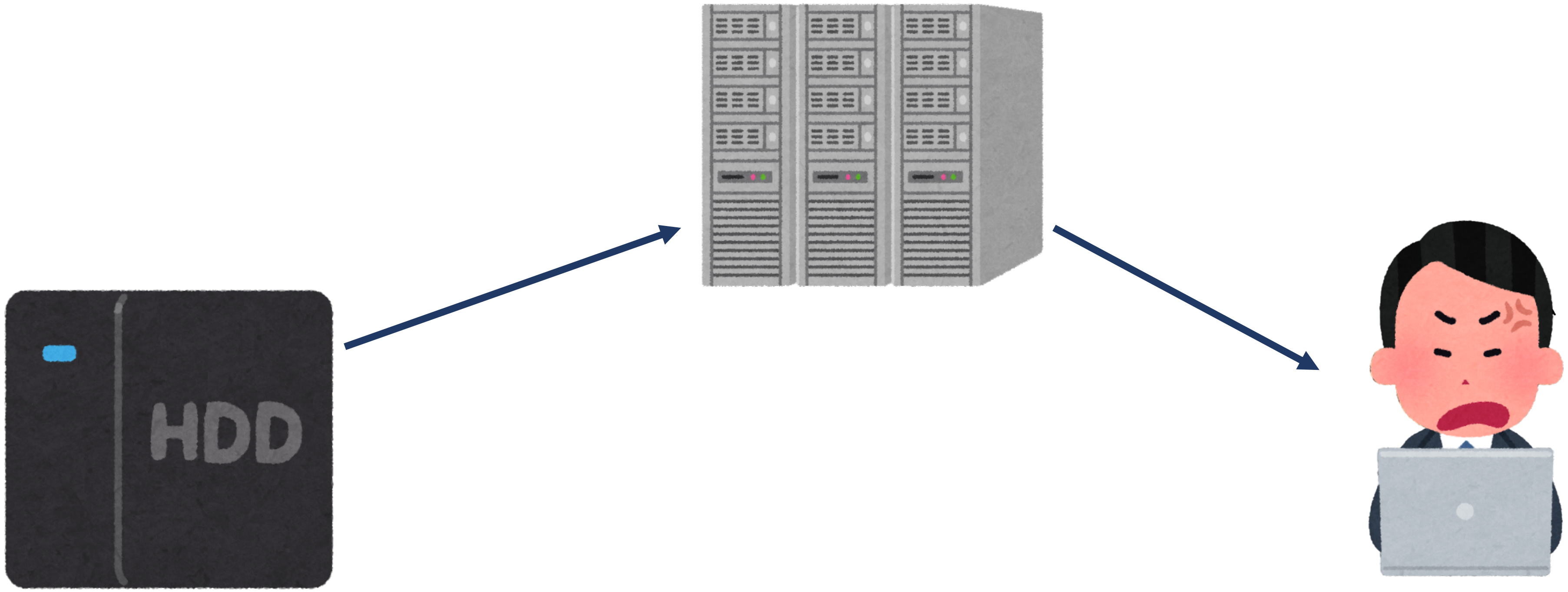
National Cheng Kung University

國立成功大學
1931
National Cheng Kung University

Operating Systems and Embedded Systems Lab

In-Memory Key-value Cache System

Server

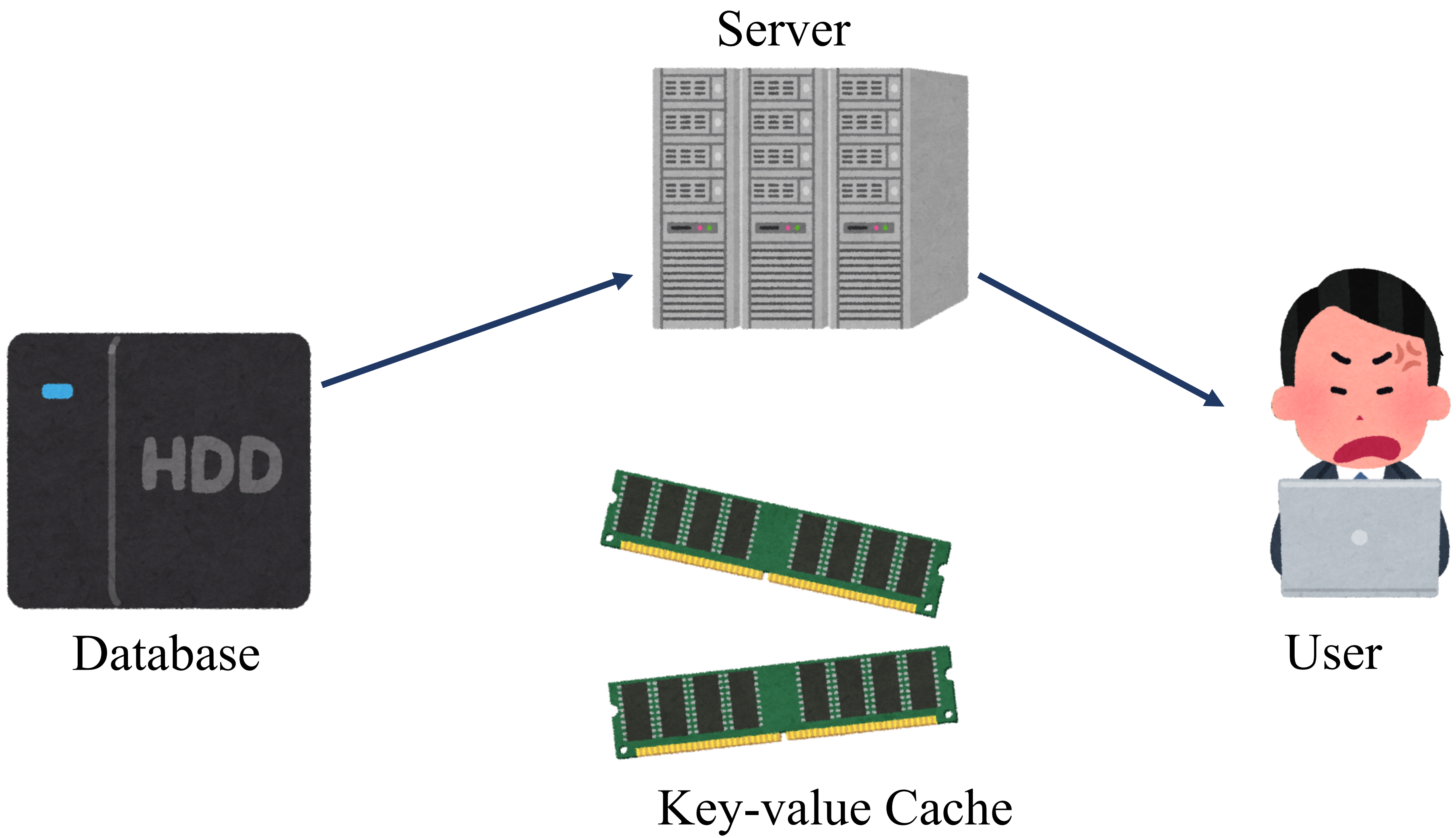


Database

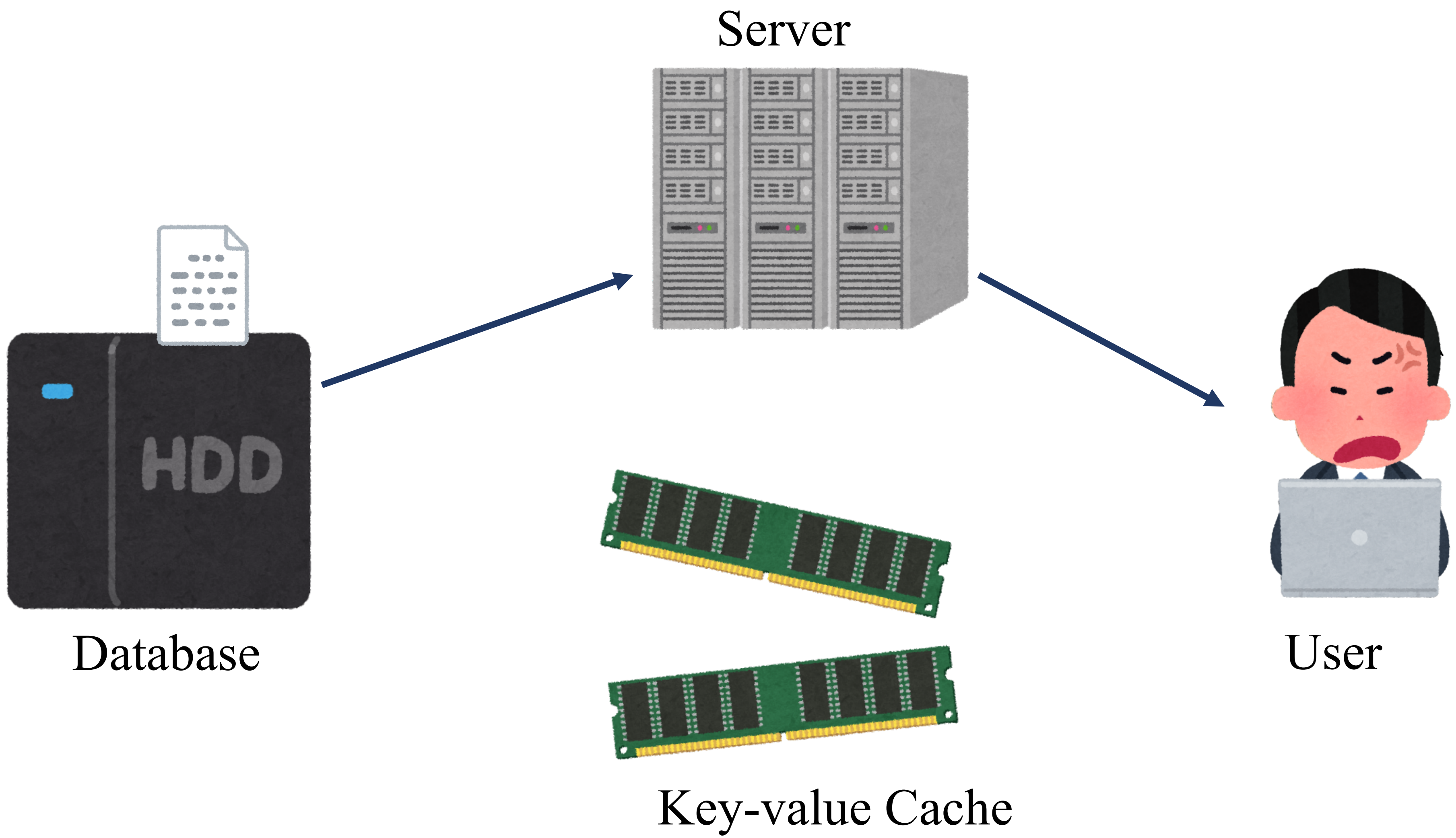
User

Key-value Cache

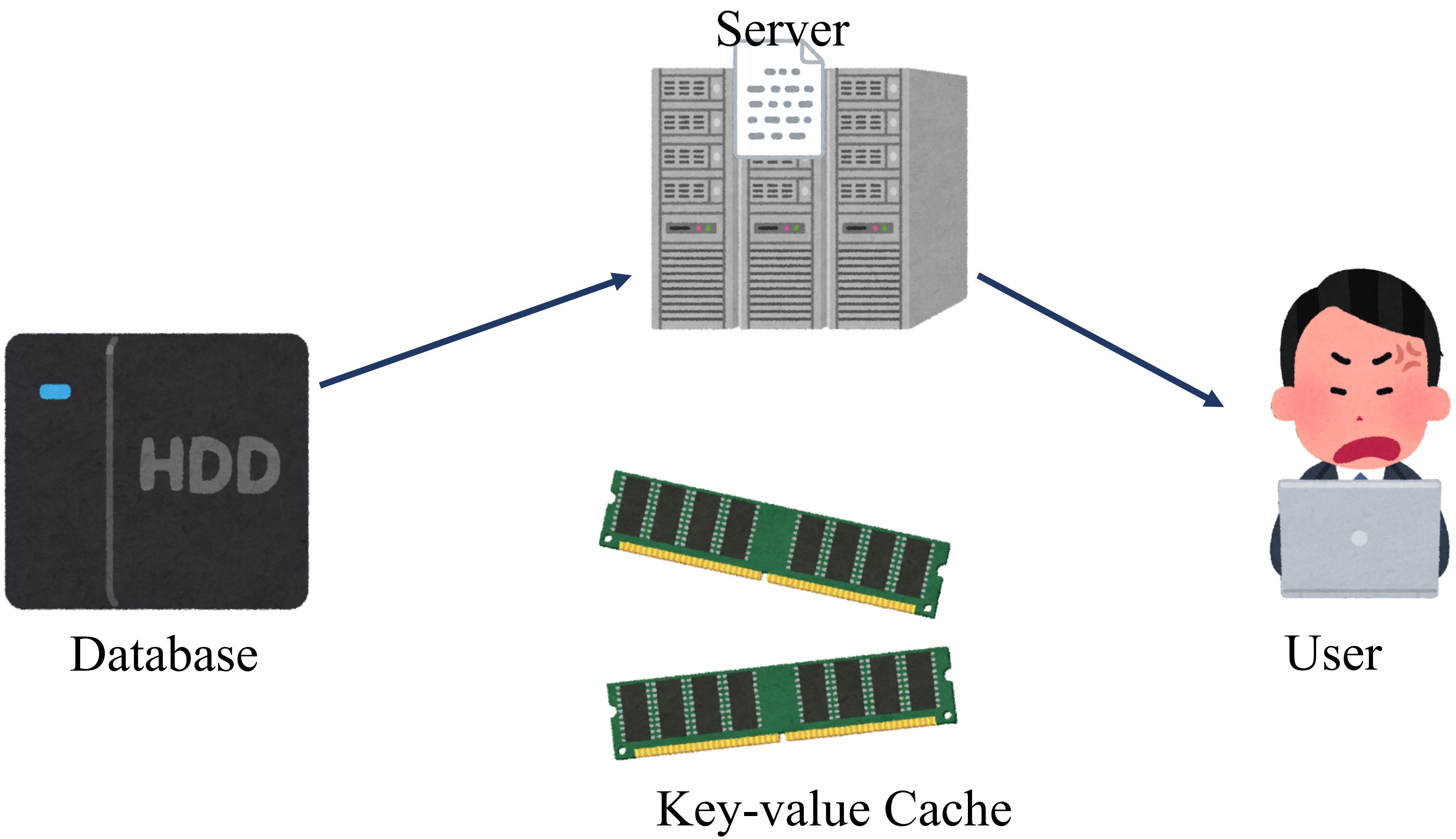
In-Memory Key-value Cache System



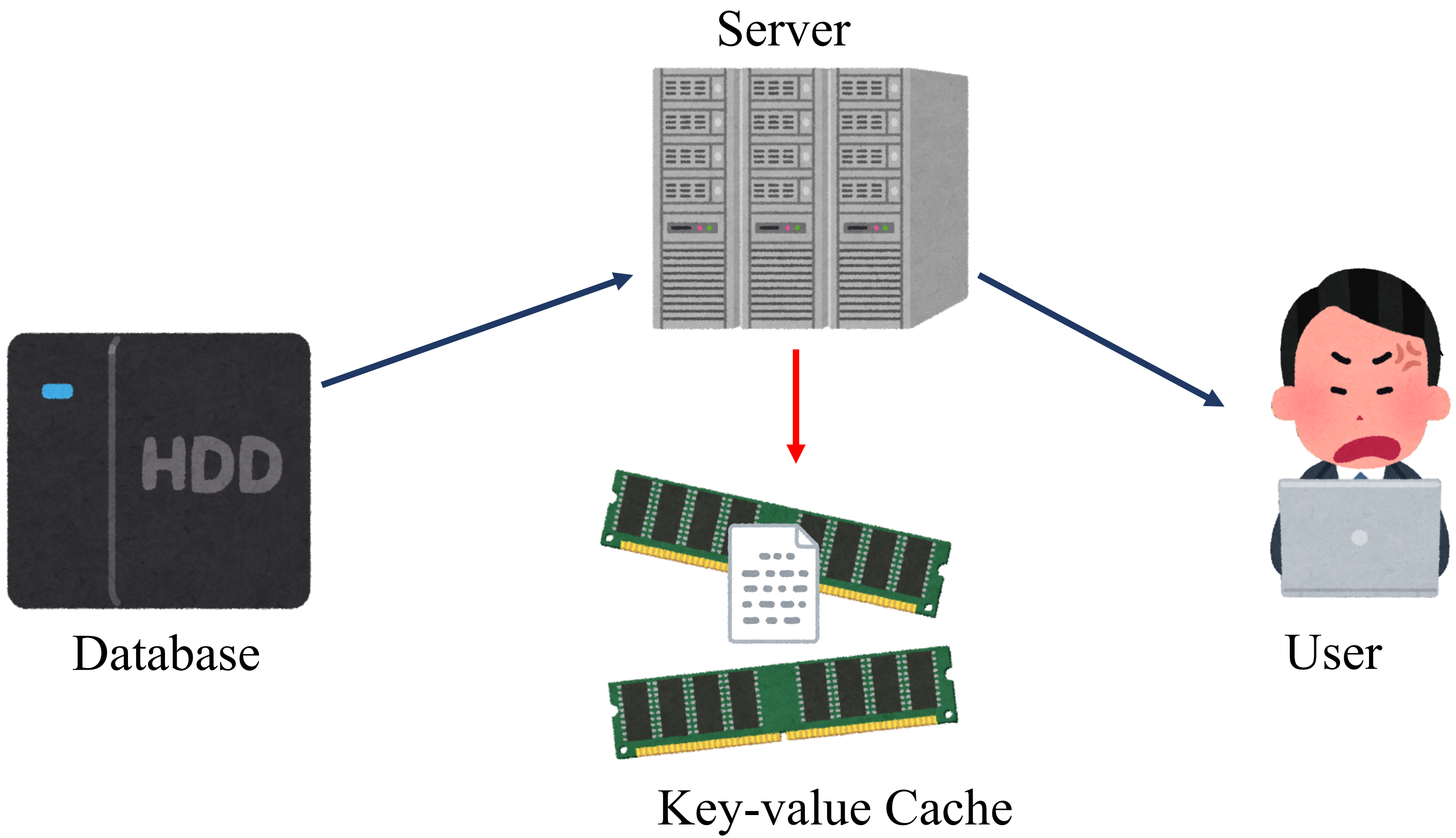
In-Memory Key-value Cache System



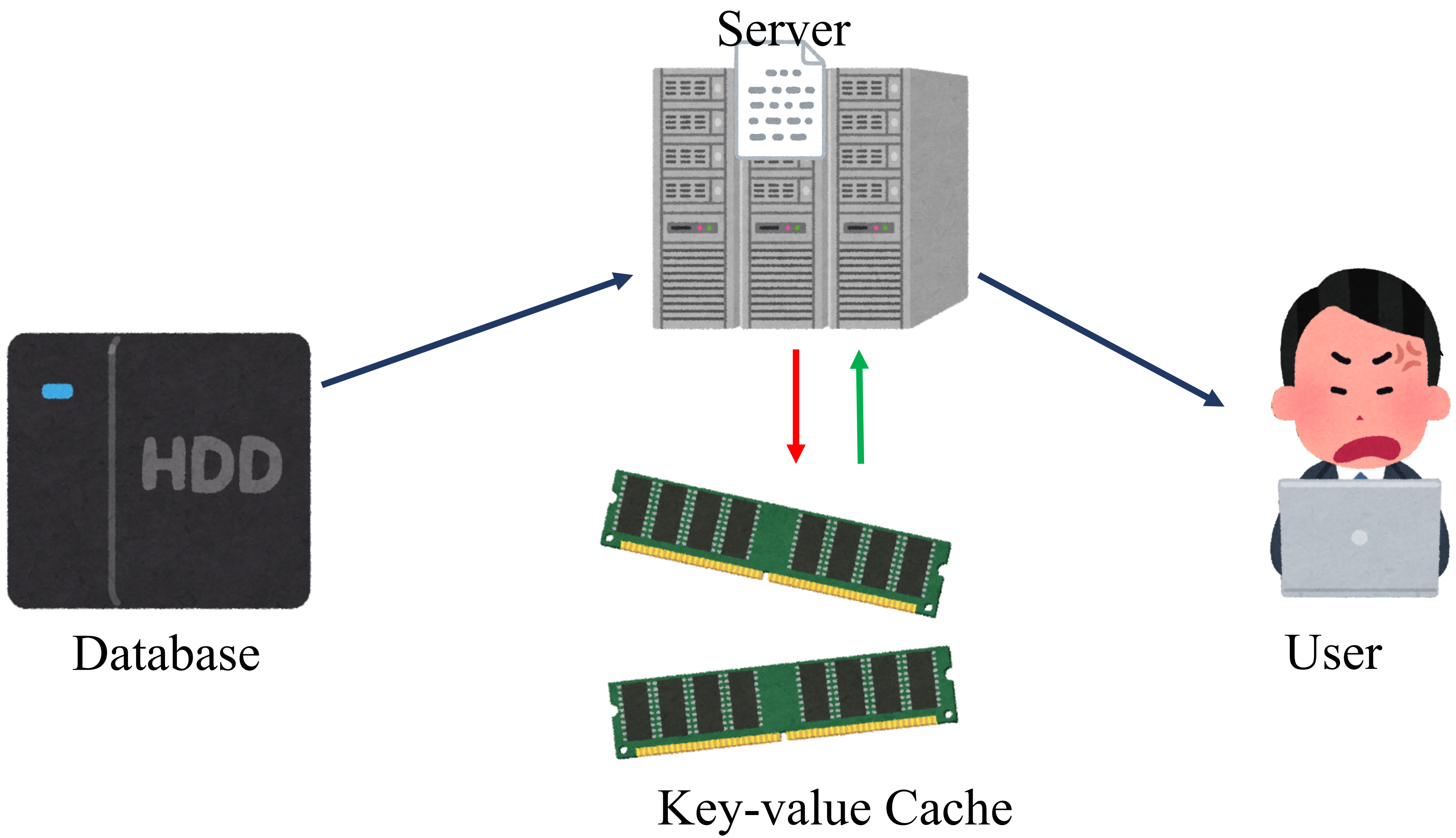
In-Memory Key-value Cache System



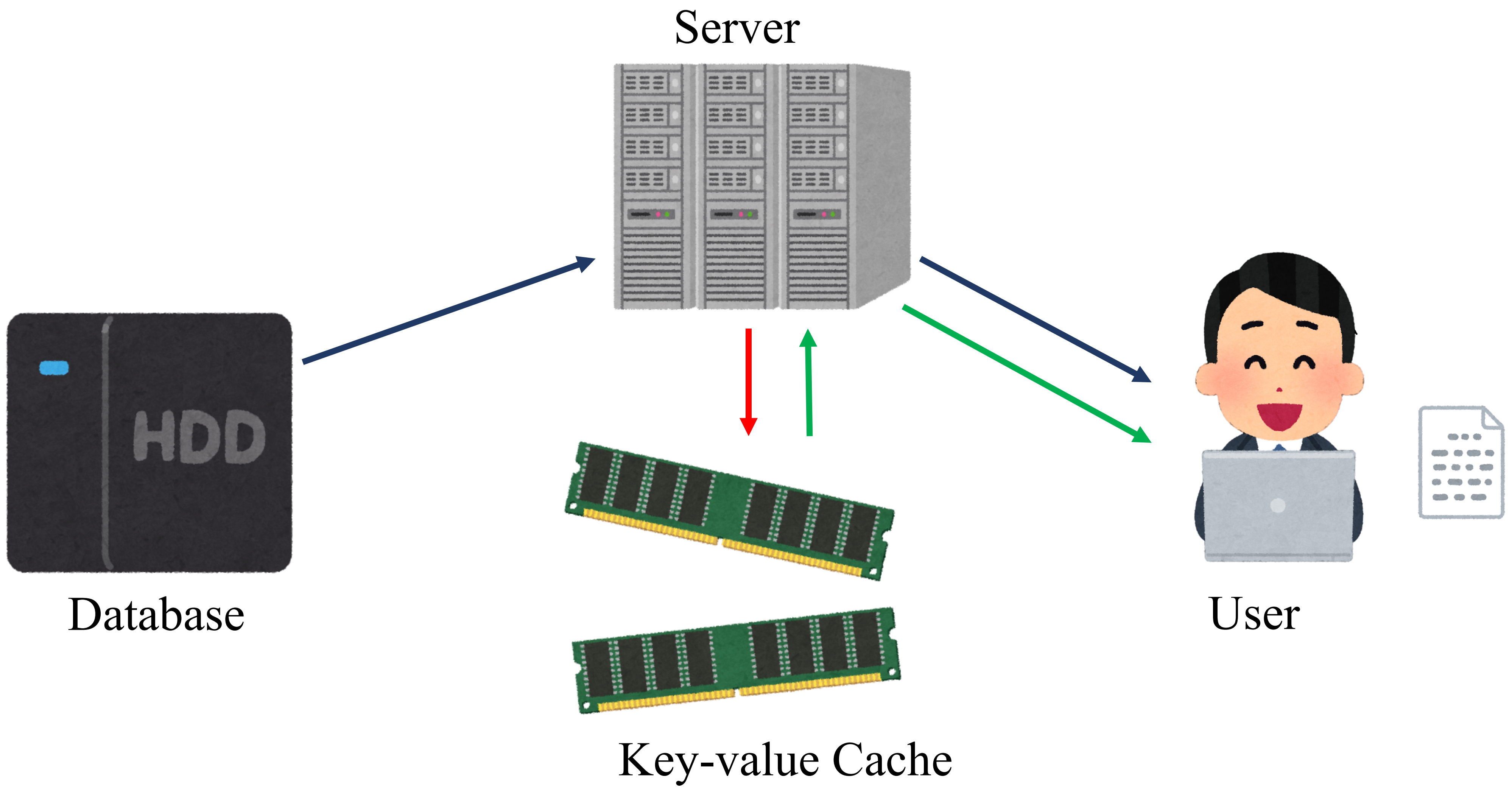
In-Memory Key-value Cache System



In-Memory Key-value Cache System

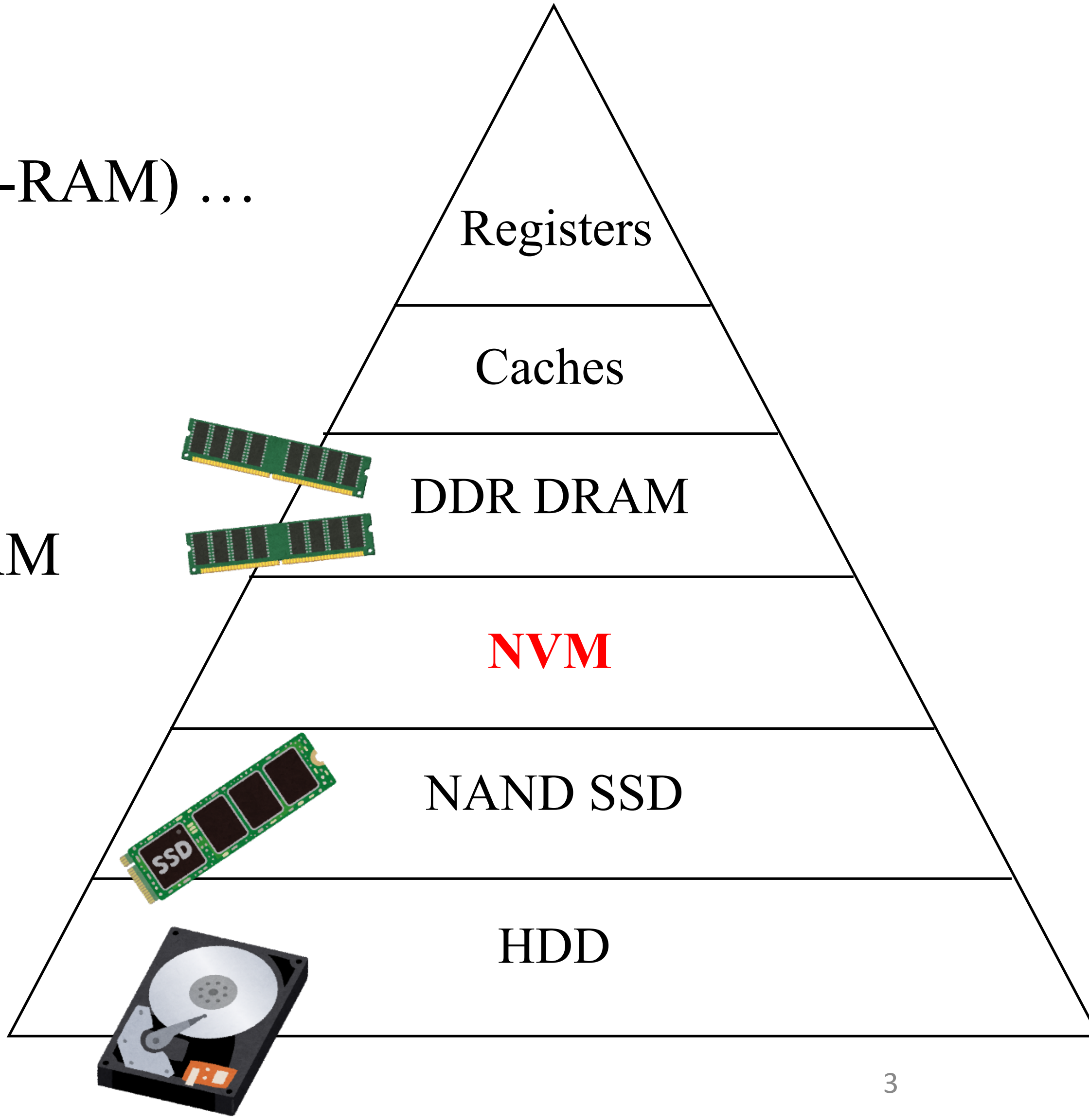


In-Memory Key-value Cache System

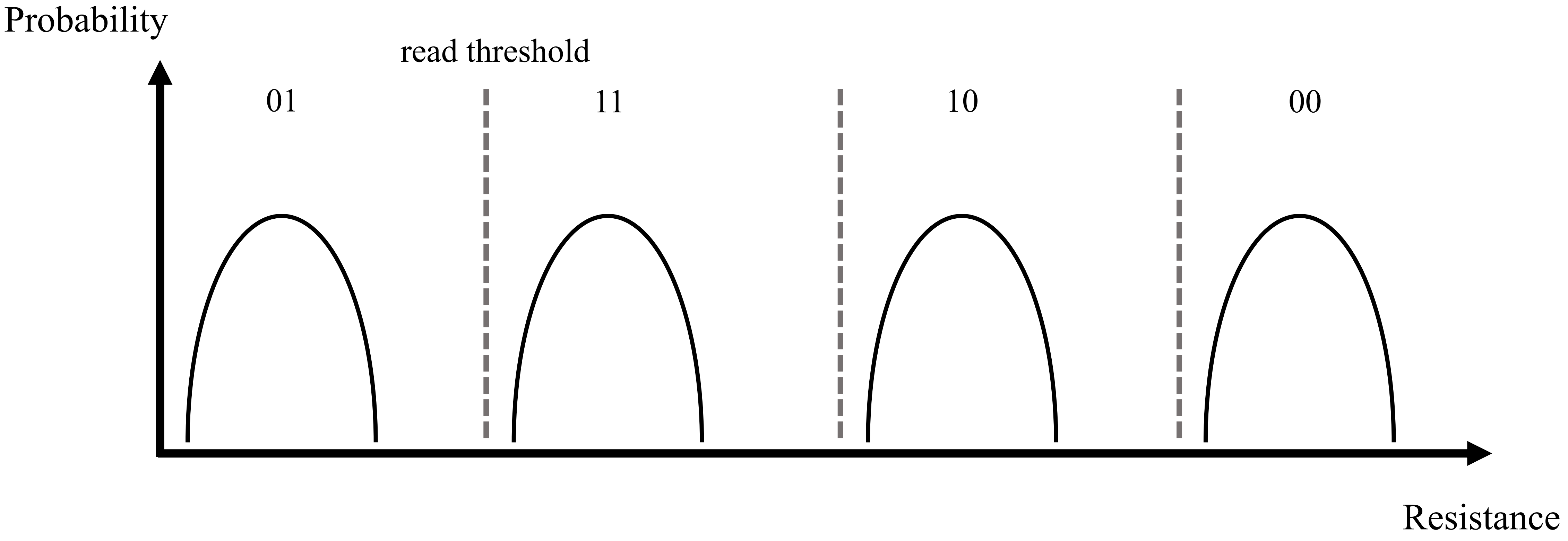


Emerging Non-volatile Memory

- Phase Change Memory (PCM), Resistive RAM (R-RAM) …
- Byte addressable
- Comparable performance (especially read) to DRAM
- Larger capacity and non-volatile

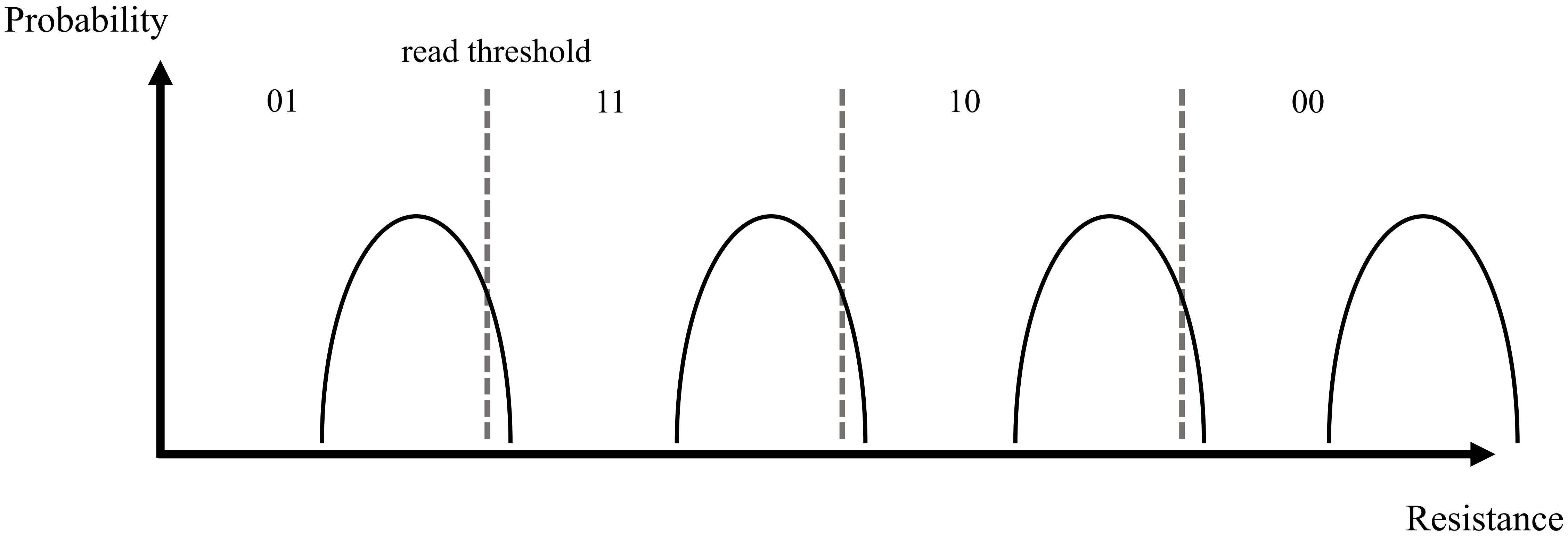


Dual Retention of Phase Change Memory (PCM) Cells



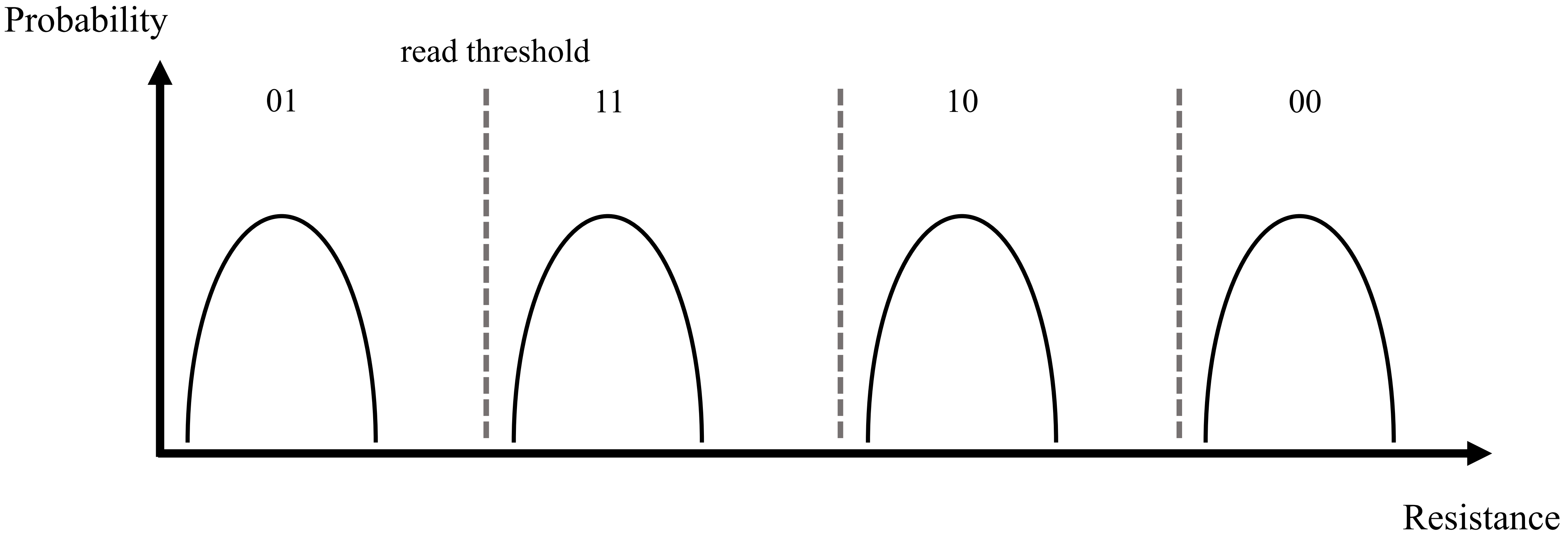
R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Dual Retention of Phase Change Memory (PCM) Cells



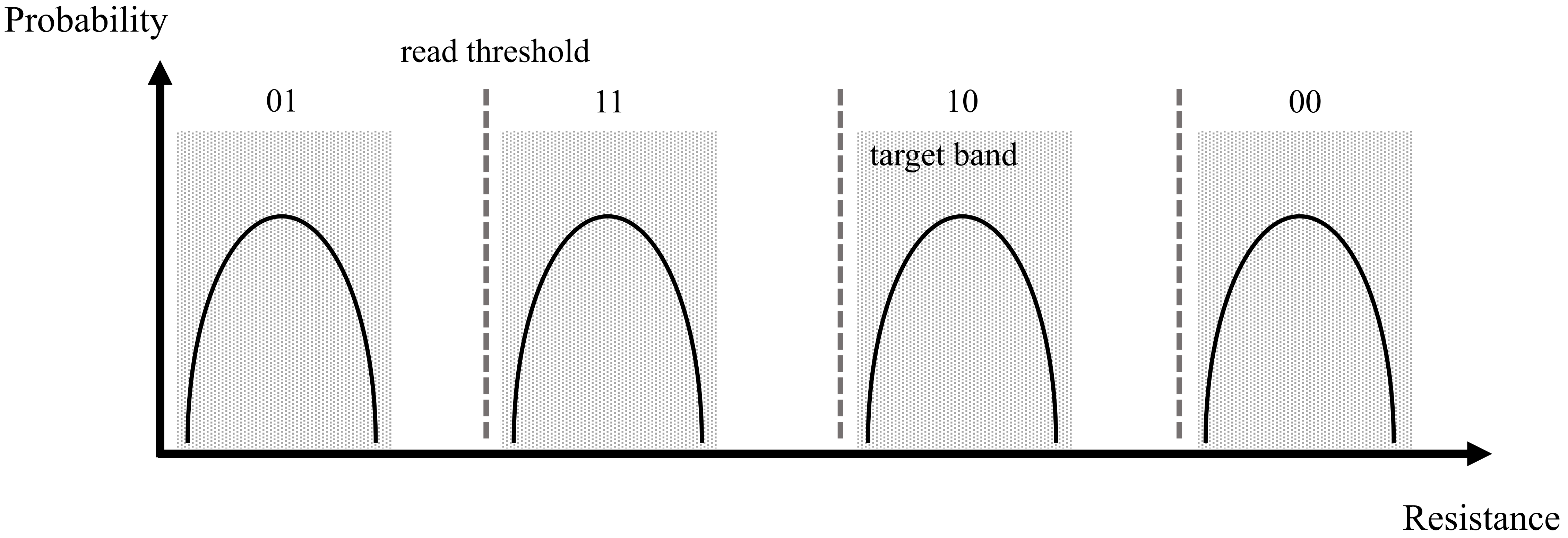
R.-S. Liu *et al.*, “NVM duet: Unified working memory and persistent store architecture,” *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Dual Retention of Phase Change Memory (PCM) Cells



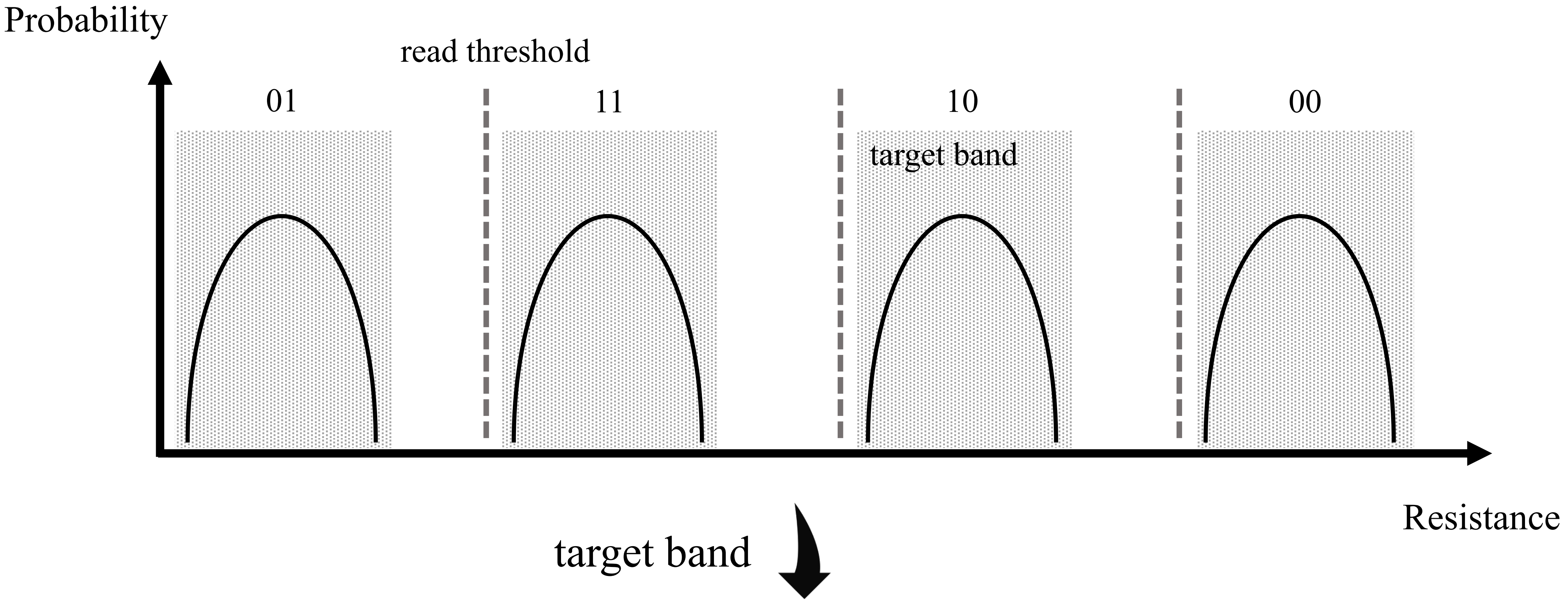
R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Dual Retention of Phase Change Memory (PCM) Cells



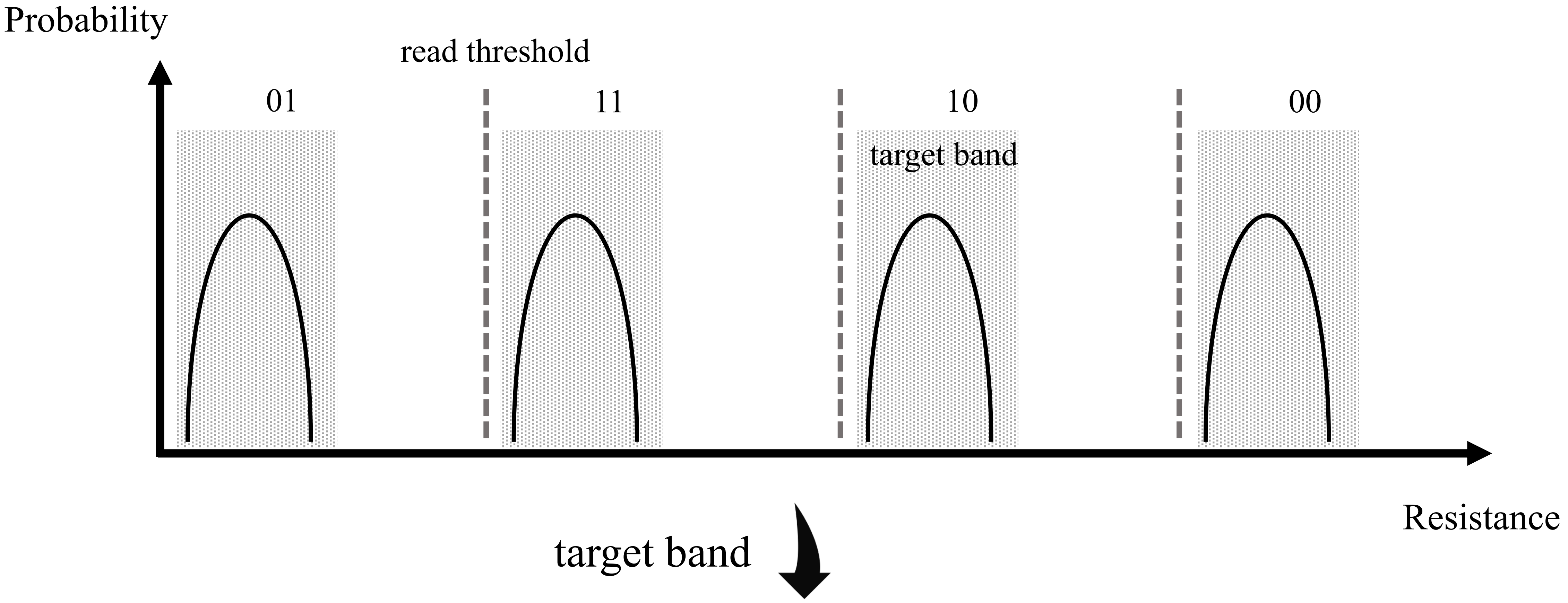
R.-S. Liu *et al.*, “NVM duet: Unified working memory and persistent store architecture,” *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Dual Retention of Phase Change Memory (PCM) Cells



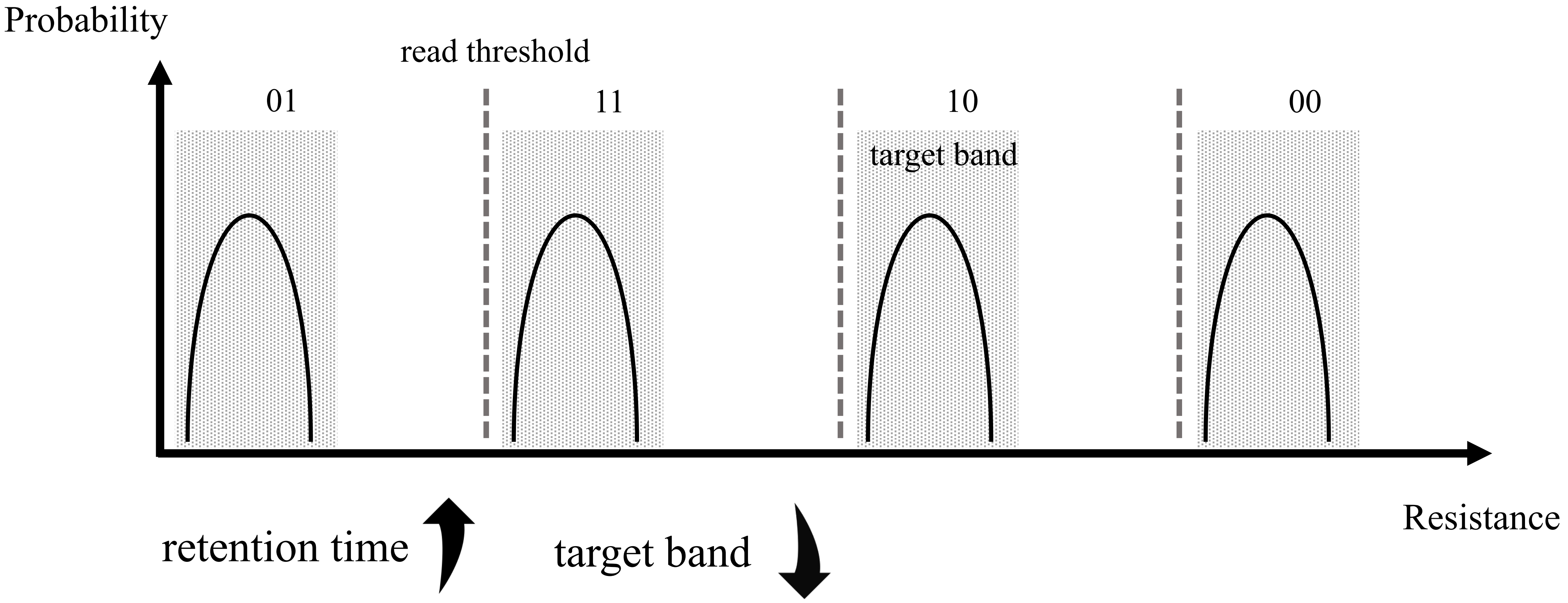
R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Dual Retention of Phase Change Memory (PCM) Cells



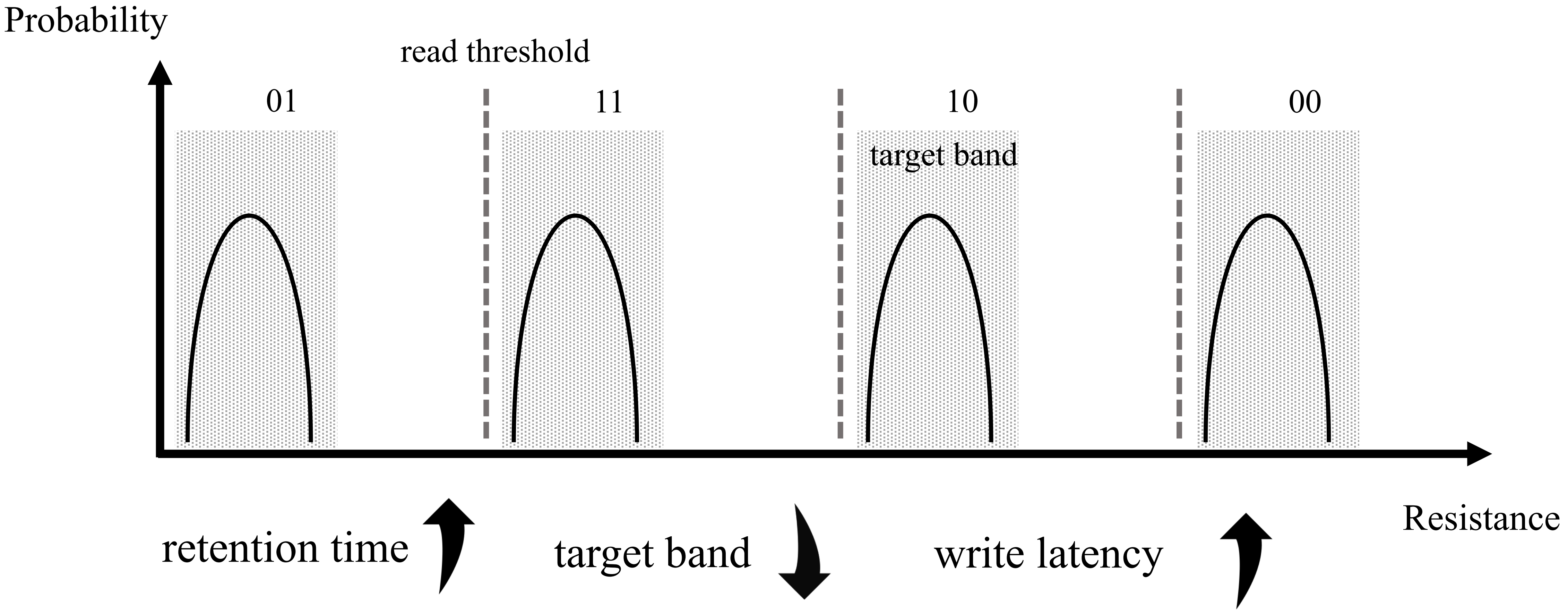
R.-S. Liu *et al.*, ‘‘NVM duet: Unified working memory and persistent store architecture,’’ *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Dual Retention of Phase Change Memory (PCM) Cells



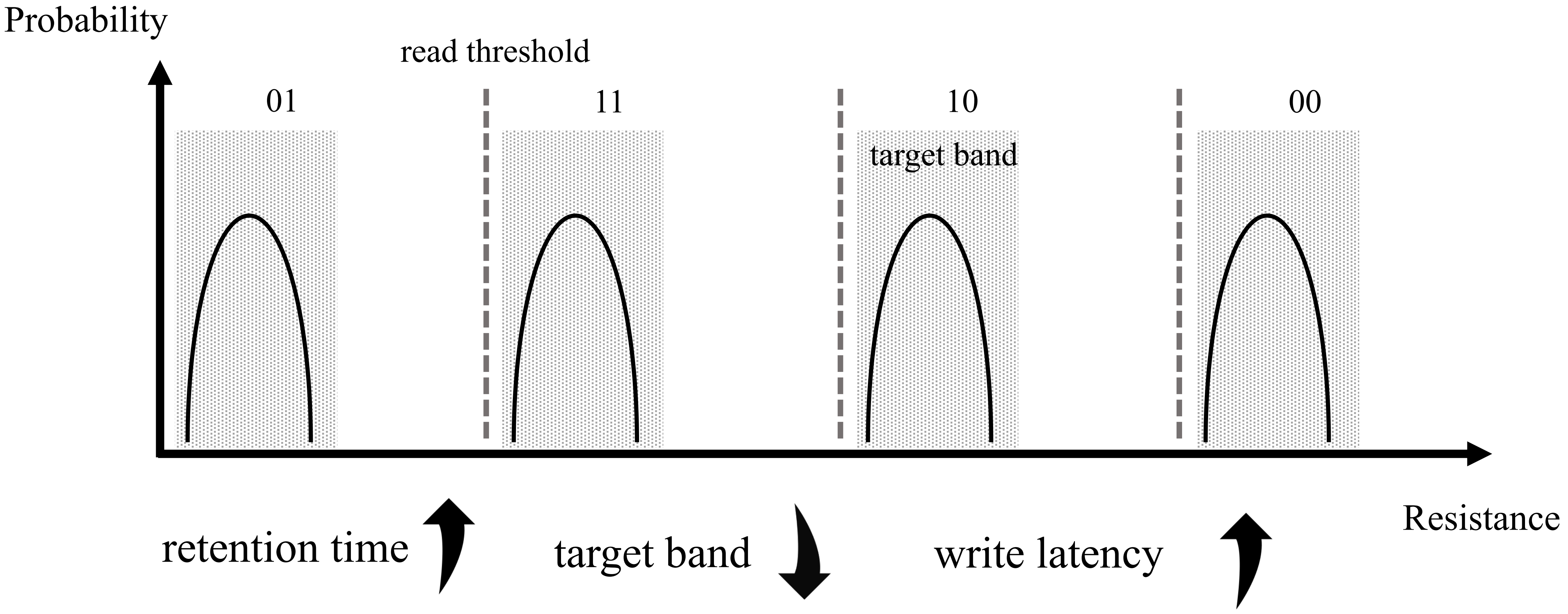
R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Dual Retention of Phase Change Memory (PCM) Cells



R.-S. Liu *et al.*, “NVM duet: Unified working memory and persistent store architecture,” *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Dual Retention of Phase Change Memory (PCM) Cells



R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Dual Retention of PCM Cell : Retention Time vs. Write Latency

- NVM data retention time/write speed tradeoff

Retention Time (sec)	Write Speedup	Average Write Iterations
10^7	Baseline (1,425 ns)	5.7
10^6	1.2 x	5.7/1.2
10^5	1.5 x	5.7/1.5
10^4	1.7 x	5.7/1.7
10^3	1.9 x	5.7/1.9
10^2	2.1 x	5.7/2.1

R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Dual Retention of PCM Cell : Retention Time vs. Write Latency

- NVM data retention time/write speed tradeoff

	Retention Time (sec)	Write Speedup	Average Write Iterations
slow write	10^7	Baseline (1,425 ns)	5.7
	10^6	1.2 x	5.7/1.2
	10^5	1.5 x	5.7/1.5
	10^4	1.7 x	5.7/1.7
	10^3	1.9 x	5.7/1.9
fast write	10^2	2.1 x	5.7/2.1

R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

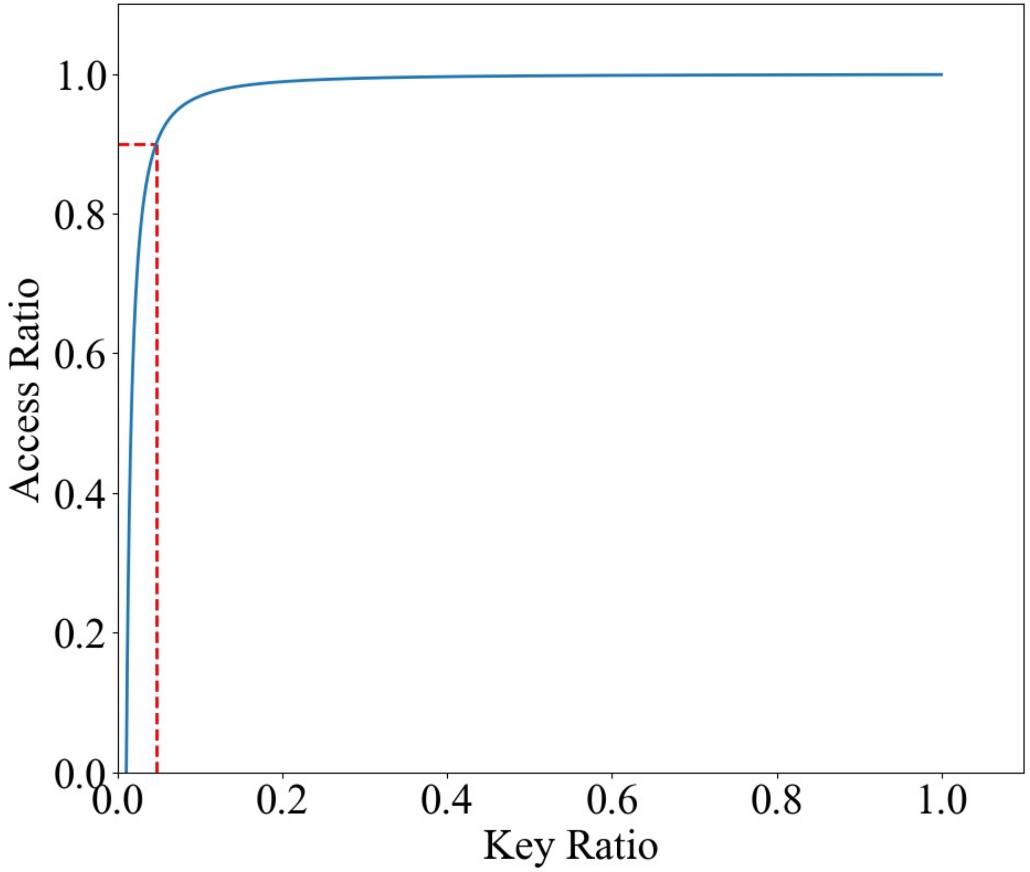
Key-value Cache System on NVM

- Pros
  - Lower system warm up cost
  - Lower hardware cost
  - Higher capacity density -> higher hit rate

- Cons
  - Performance decline due to longer **write latency**

The Hotspot Issue of Key-value Cache Systems

- According to the previous studies, in many real life workloads in key-value cache, only a small portion of items are accessed frequently



Dual-KV : Main Idea

- We introduce Dual-KV, which
  - Identify frequently-updated hot items

  - Speedup hot items writes using fast writes

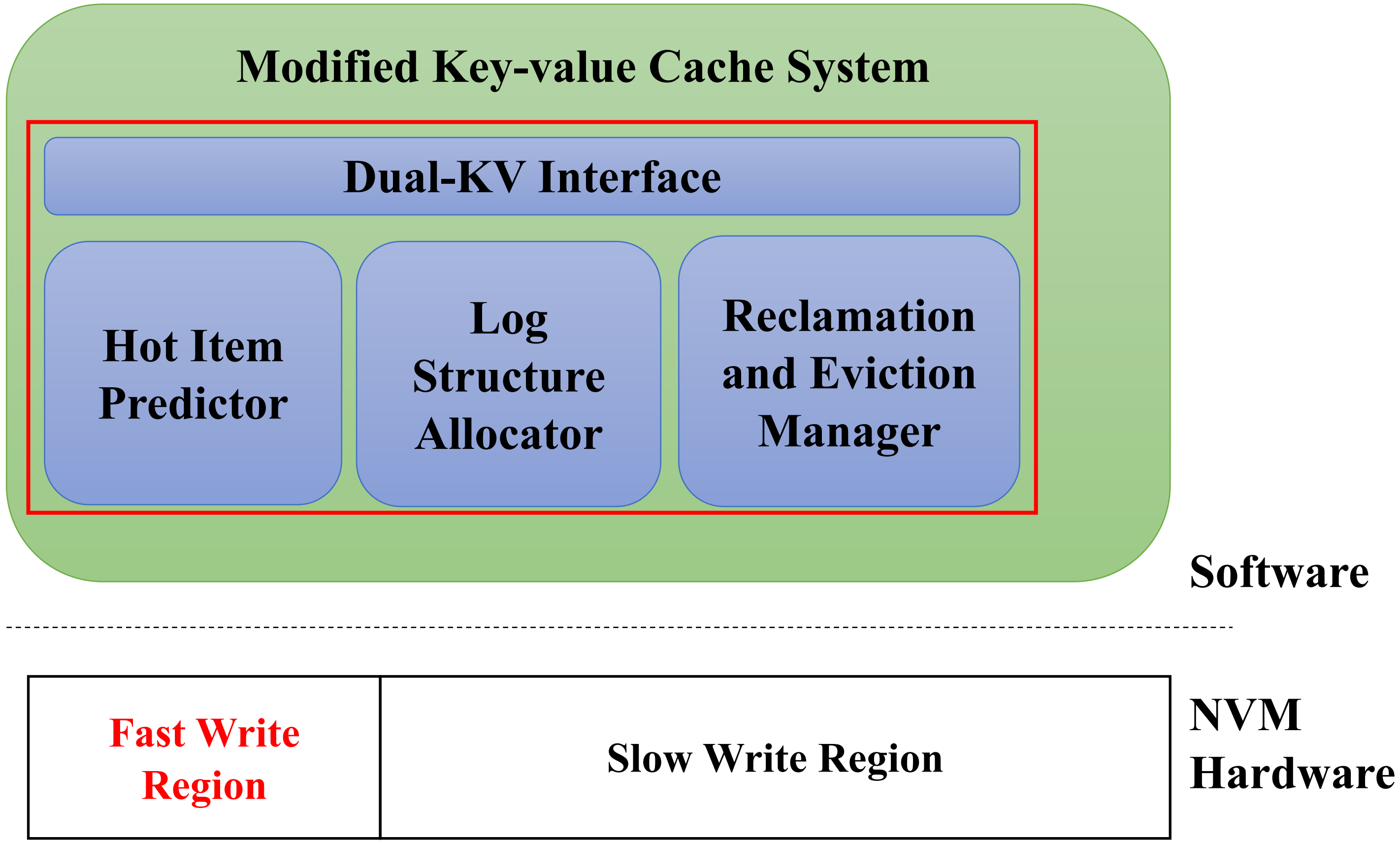
Contributions

- Single-threaded and 16-threaded YCSB workloads
  - 43% and 83% throughput improvement respectively
  - 30% and 45% request latency reduction respectively

- Based on our interface, Dual-KV can be easily integrated into existing KV cache systems
  - Only about 30 LoC insertions/modifications to be embedded into Memcached

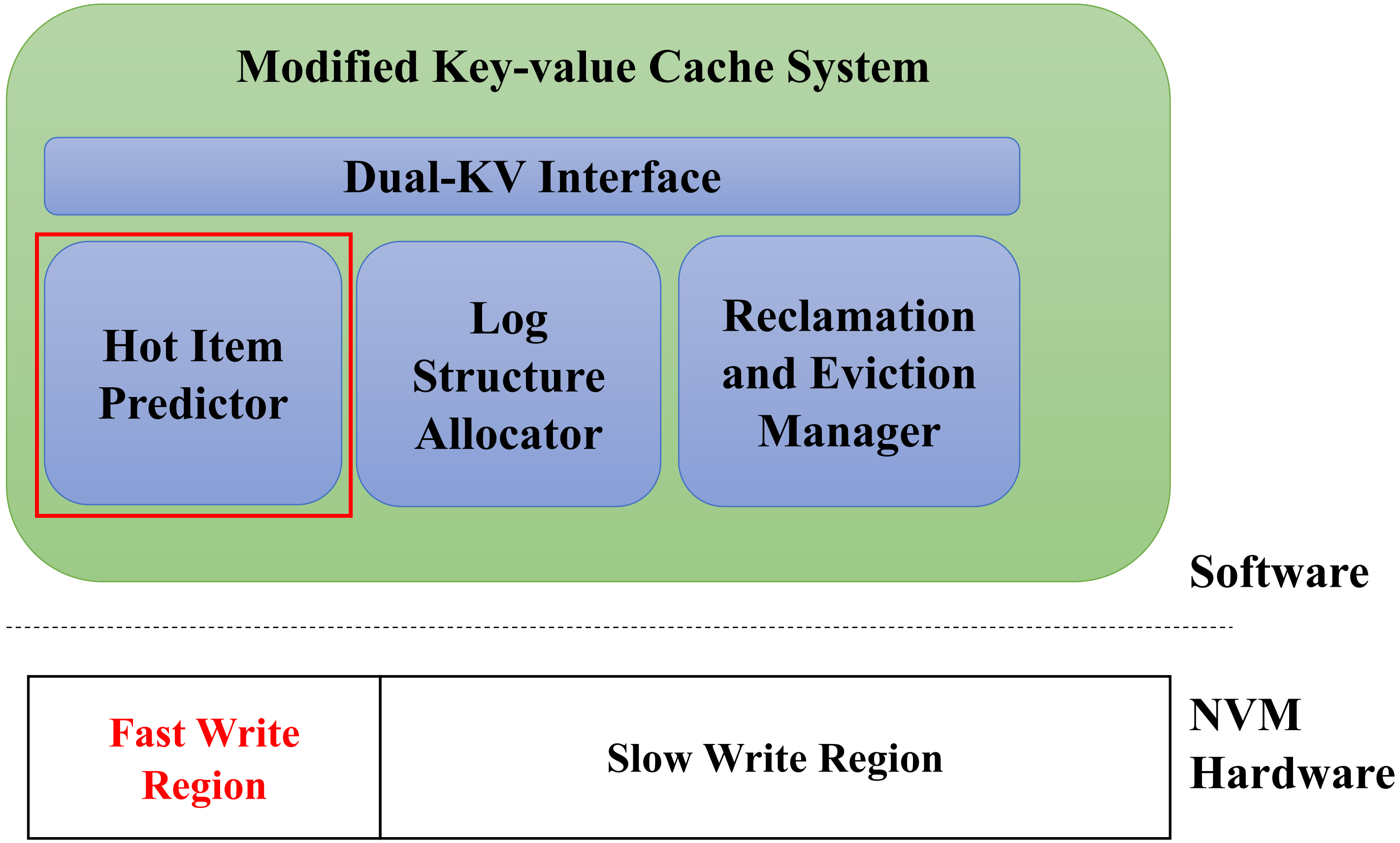
Dual-KV Model

- Dual-KV can be embedded into an existing key-value cache system
  - Dual-KV interface
  - Hot item predictor
  - Log structure allocator
  - Reclamation and eviction manager

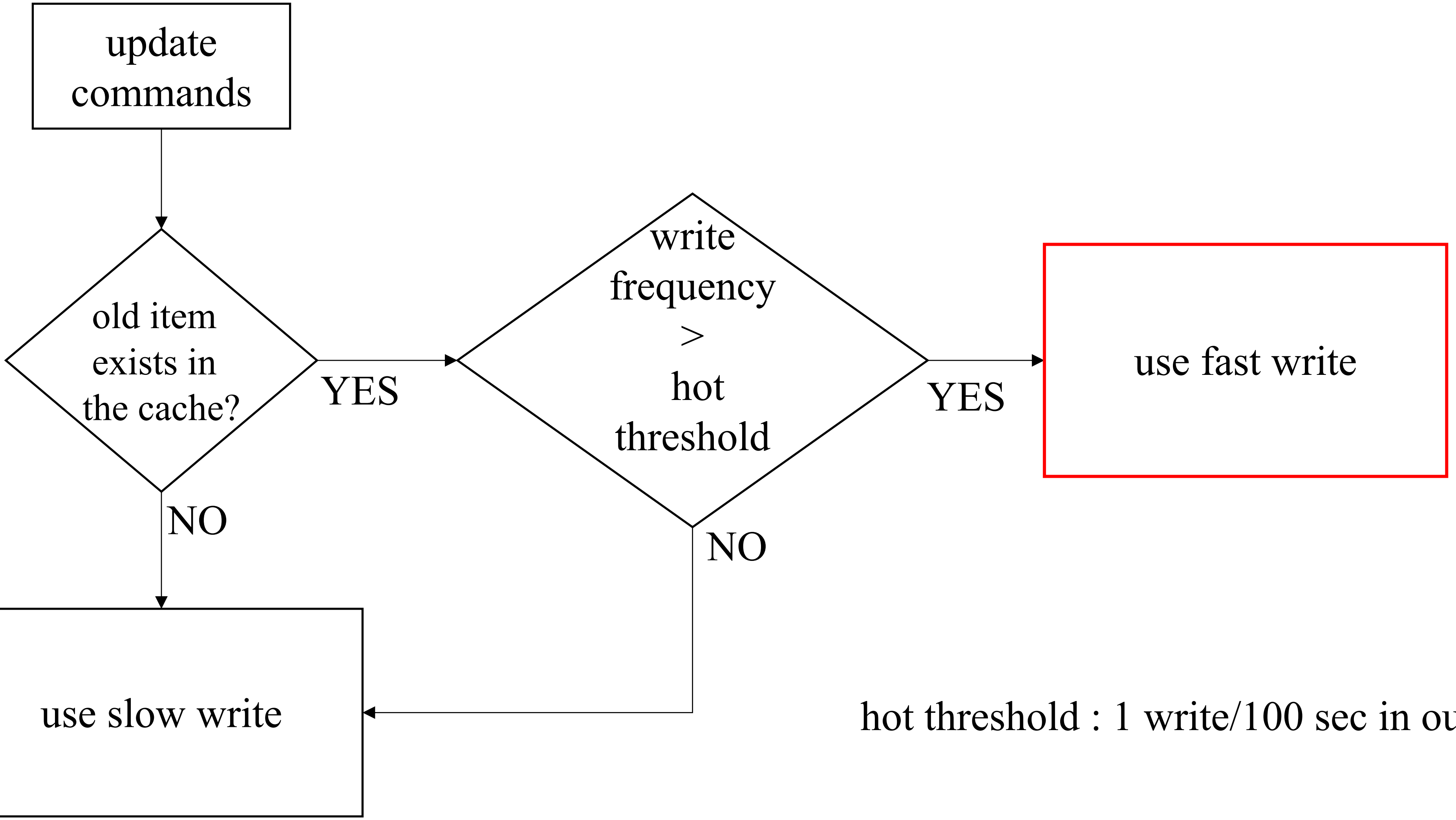


Hot Item Predictor

- Calculate the "hotness" of each item and determine the write mode (fast/slow)



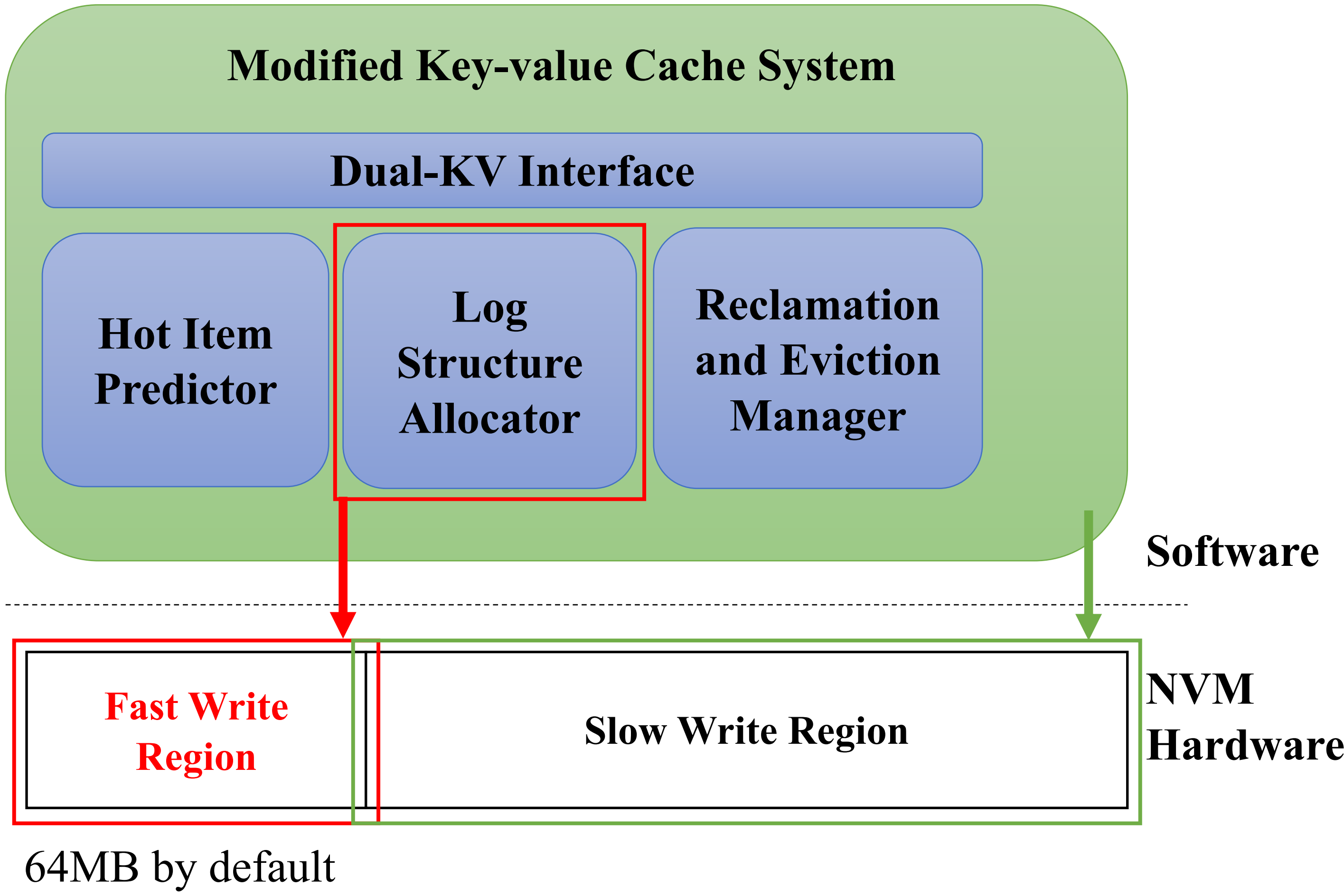
Hot Item Predictor



hot threshold : 1 write/100 sec in our work

Managing NVM

- The NVM is divided into
  - Fast write region : managed by our log structure allocator
  - Slow write region : managed by the original **Memcached slab allocator**



Memcached Slab Allocator : Problem

A Slab of
Slab class #1

A Slab of
Slab class #2

…

A Slab of
Slab class #6

Memcached Slab Allocator : Problem

- Slab: pre-divided 1MB pages

A Slab of
Slab class #1

A Slab of
Slab class #2

…

A Slab of
Slab class #6

Memcached Slab Allocator : Problem

- Slab: pre-divided 1MB pages
  - A page is further cut into chunks based on the slab class it belongs to

A Slab of
Slab class #1

A Slab of
Slab class #2

…

A Slab of
Slab class #6

Memcached Slab Allocator : Problem

- Slab: pre-divided 1MB pages
  - A page is further cut into chunks based on the slab class it belongs to
  - Chunks are the allocation unit for items

A Slab of
Slab class #1

A Slab of
Slab class #2

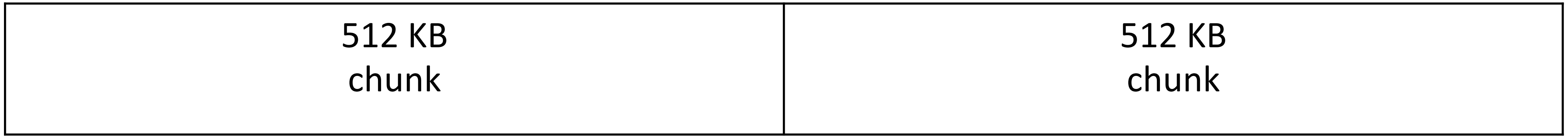
…

A Slab of
Slab class #6

Memcached Slab Allocator : Problem

- Slab: pre-divided 1MB pages
  - A page is further cut into chunks based on the slab class it belongs to
  - Chunks are the allocation unit for items

A Slab of
Slab class #1



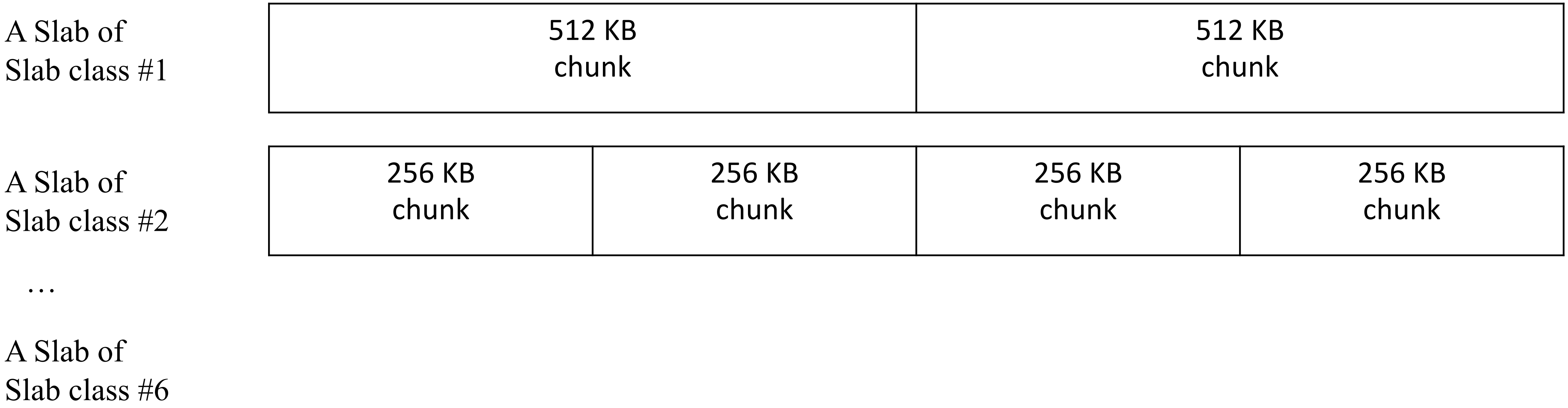
A Slab of
Slab class #2

…

A Slab of
Slab class #6

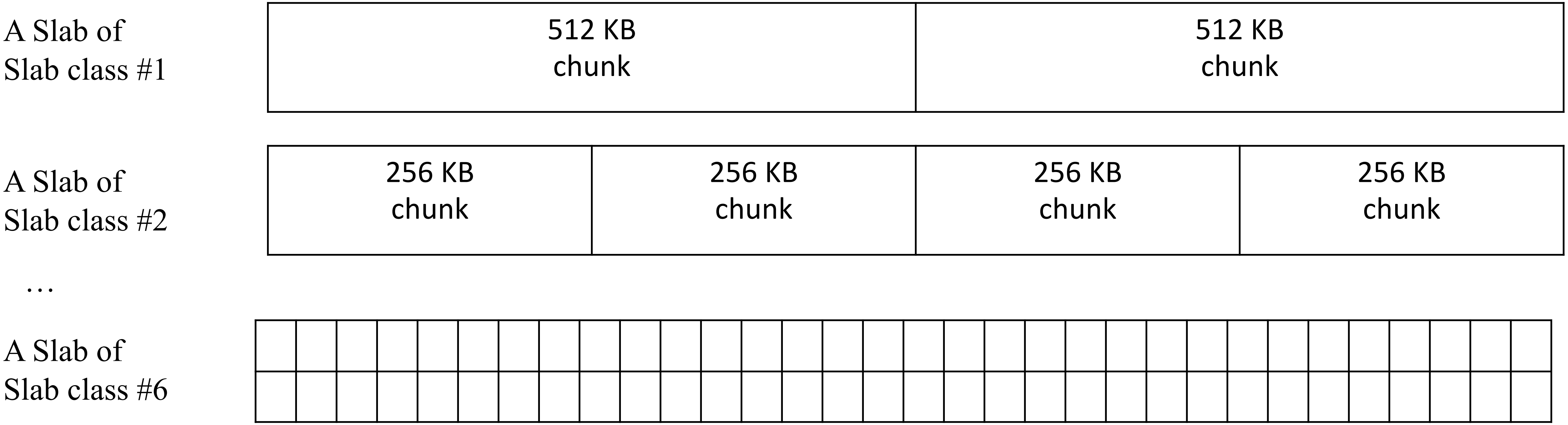
Memcached Slab Allocator : Problem

- Slab: pre-divided 1MB pages
  - A page is further cut into chunks based on the slab class it belongs to
  - Chunks are the allocation unit for items



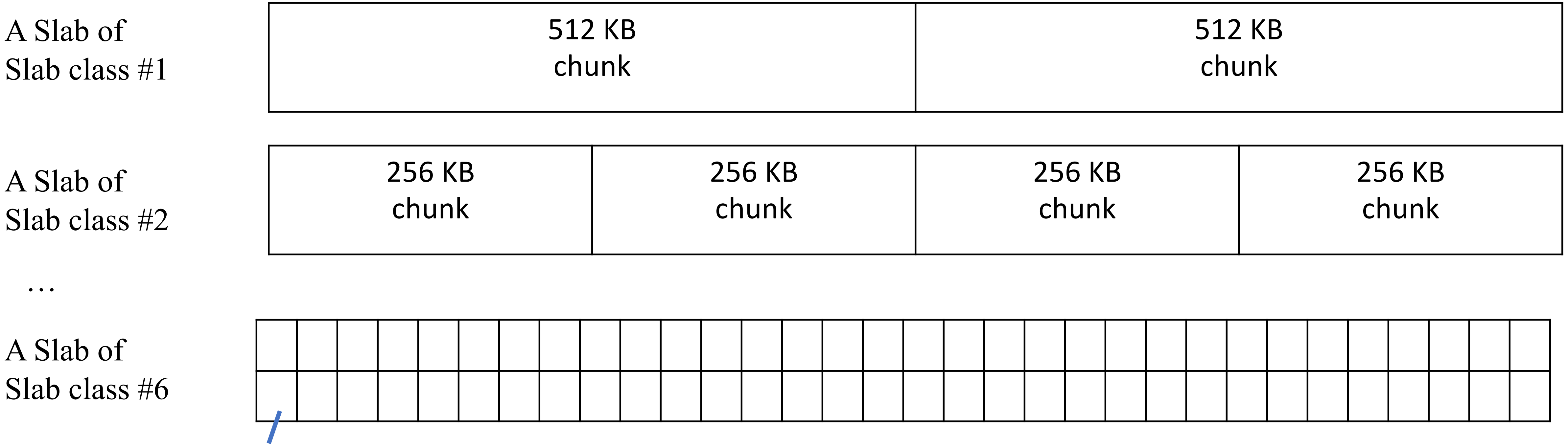
Memcached Slab Allocator : Problem

- Slab: pre-divided 1MB pages
  - A page is further cut into chunks based on the slab class it belongs to
  - Chunks are the allocation unit for items



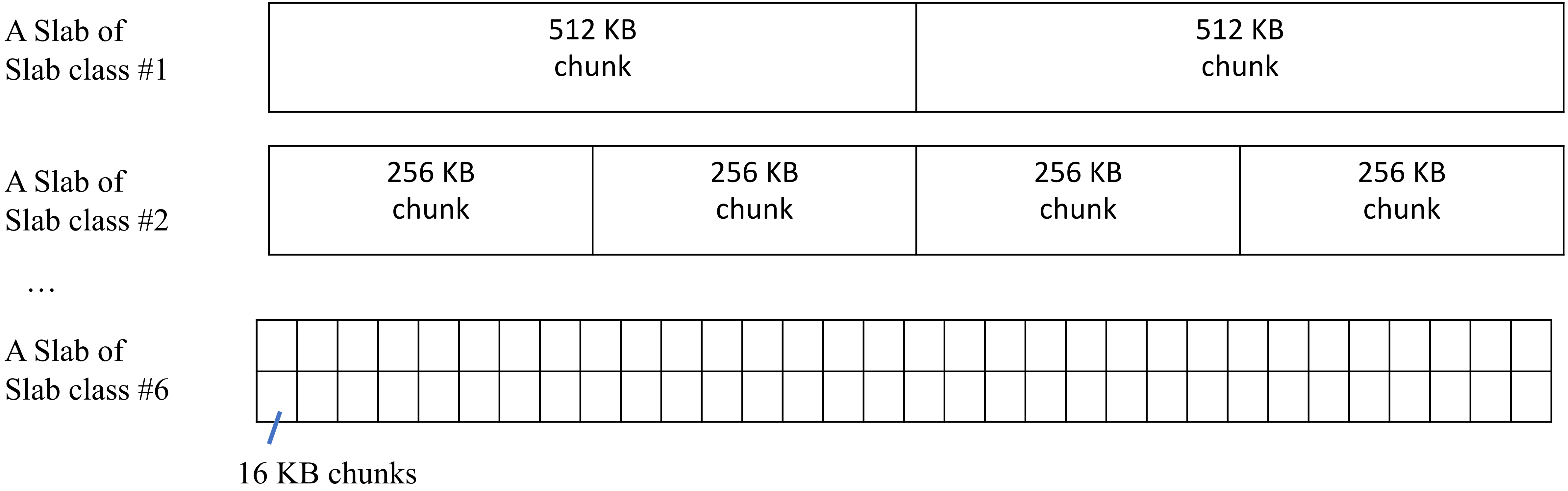
Memcached Slab Allocator : Problem

- Slab: pre-divided 1MB pages
  - A page is further cut into chunks based on the slab class it belongs to
  - Chunks are the allocation unit for items

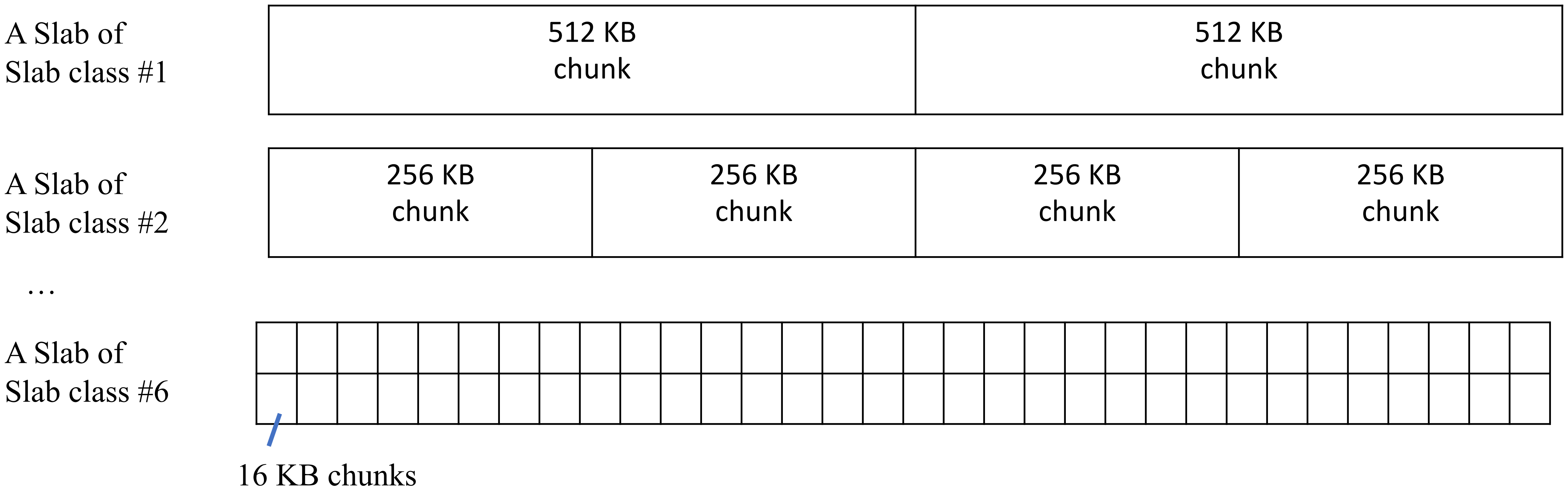


Memcached Slab Allocator : Problem

- Slab: pre-divided 1MB pages
  - A page is further cut into chunks based on the slab class it belongs to
  - Chunks are the allocation unit for items

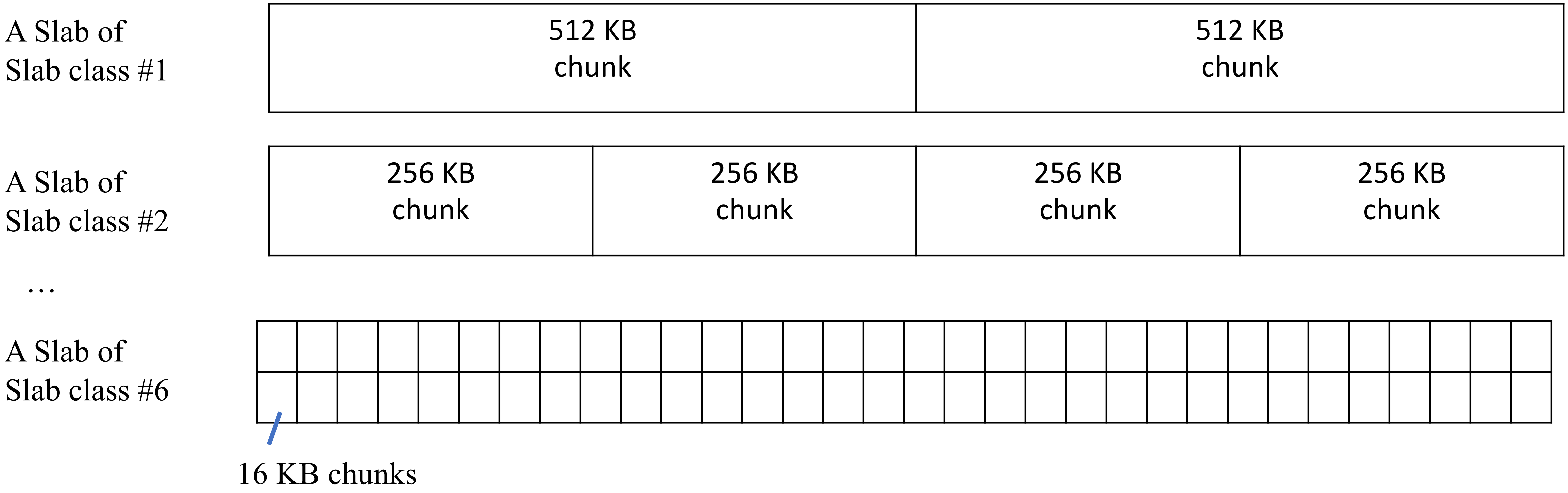


Memcached Slab Allocator : Problem



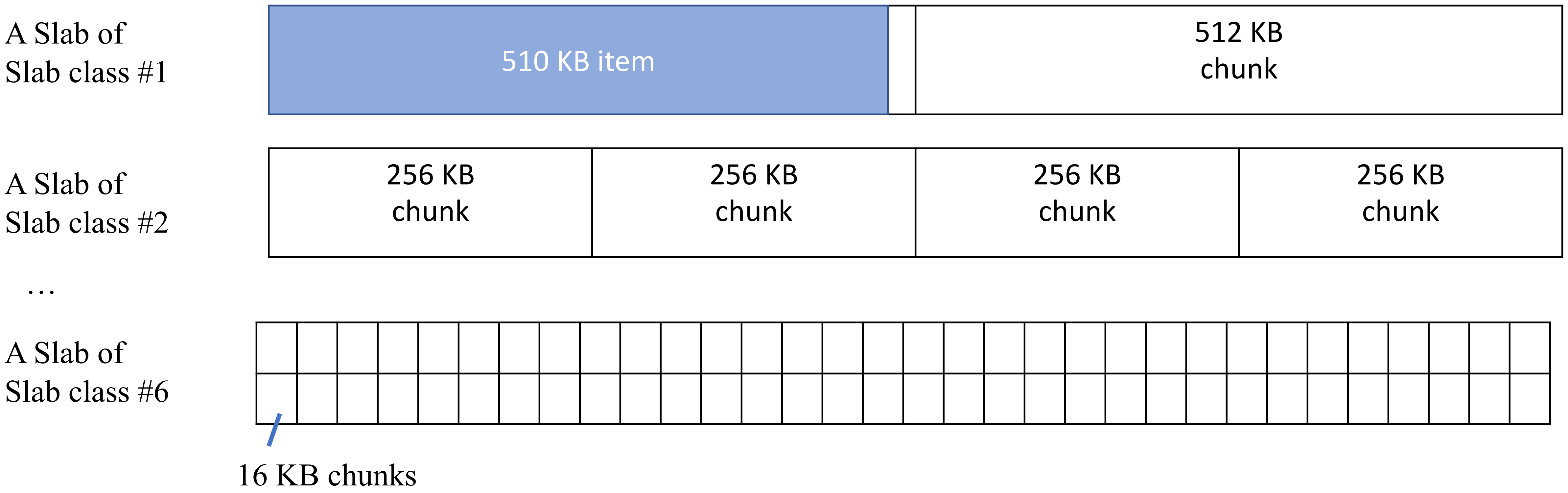
Memcached Slab Allocator : Problem

- When an item demands a memory space, the best fitting chunk is selected for it



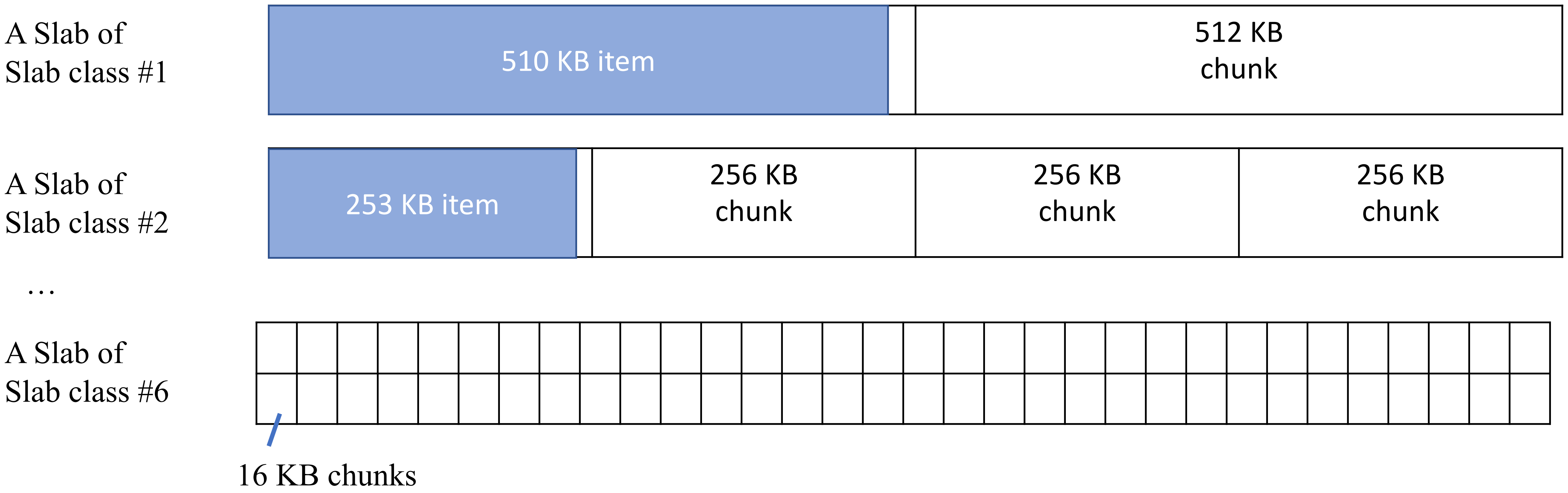
Memcached Slab Allocator : Problem

- When an item demands a memory space, the best fitting chunk is selected for it



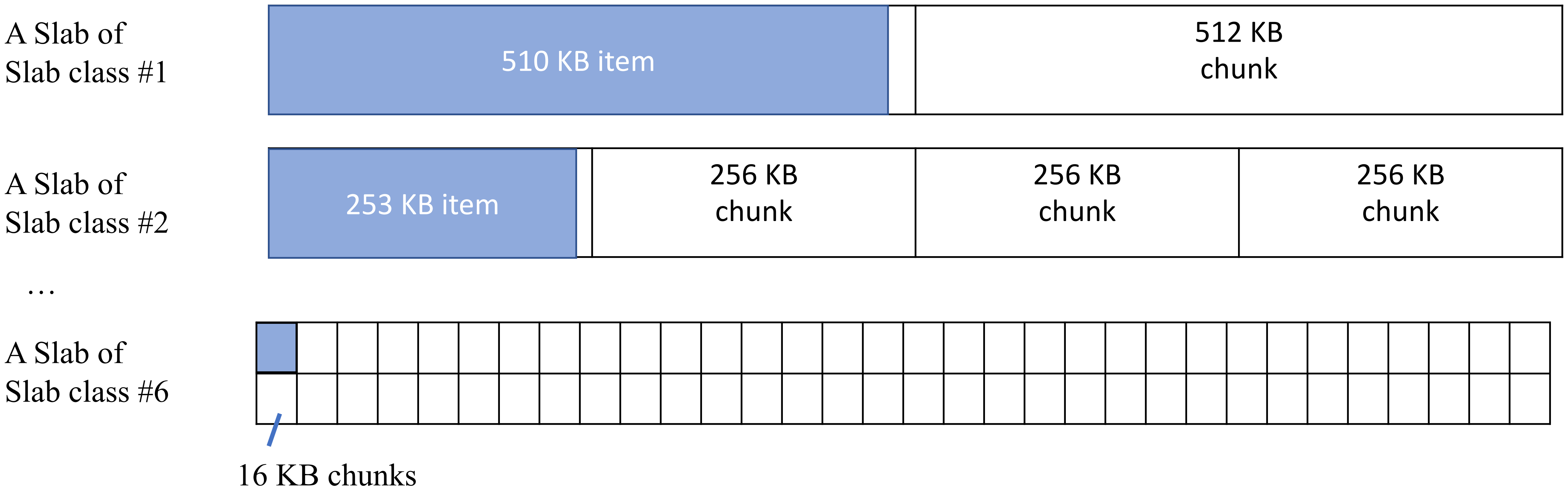
Memcached Slab Allocator : Problem

- When an item demands a memory space, the best fitting chunk is selected for it



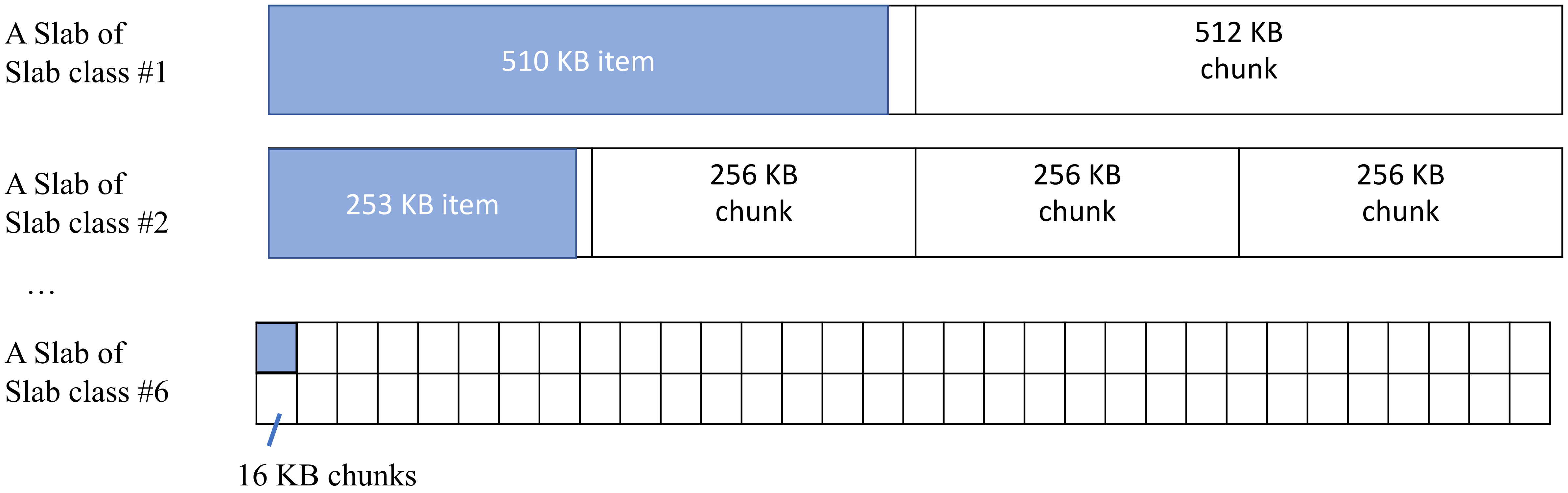
Memcached Slab Allocator : Problem

- When an item demands a memory space, the best fitting chunk is selected for it



Memcached Slab Allocator : Problem

- When an item demands a memory space, the best fitting chunk is selected for it
  - Most hot items are small items -> slabs with *big* chunks have low utilities



Memcached Slab Allocator : Problem

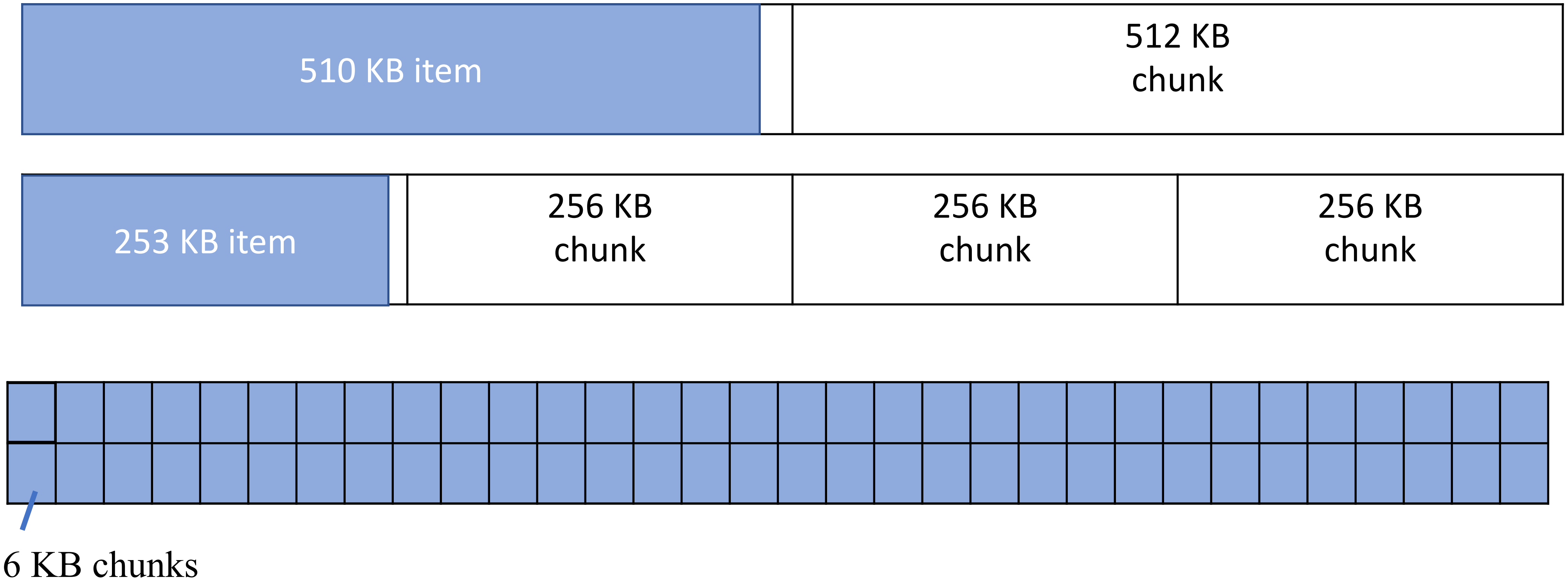
- When an item demands a memory space, the best fitting chunk is selected for it
  - Most hot items are small items -> slabs with *big* chunks have low utilities

A Slab of
Slab class #1

A Slab of
Slab class #2

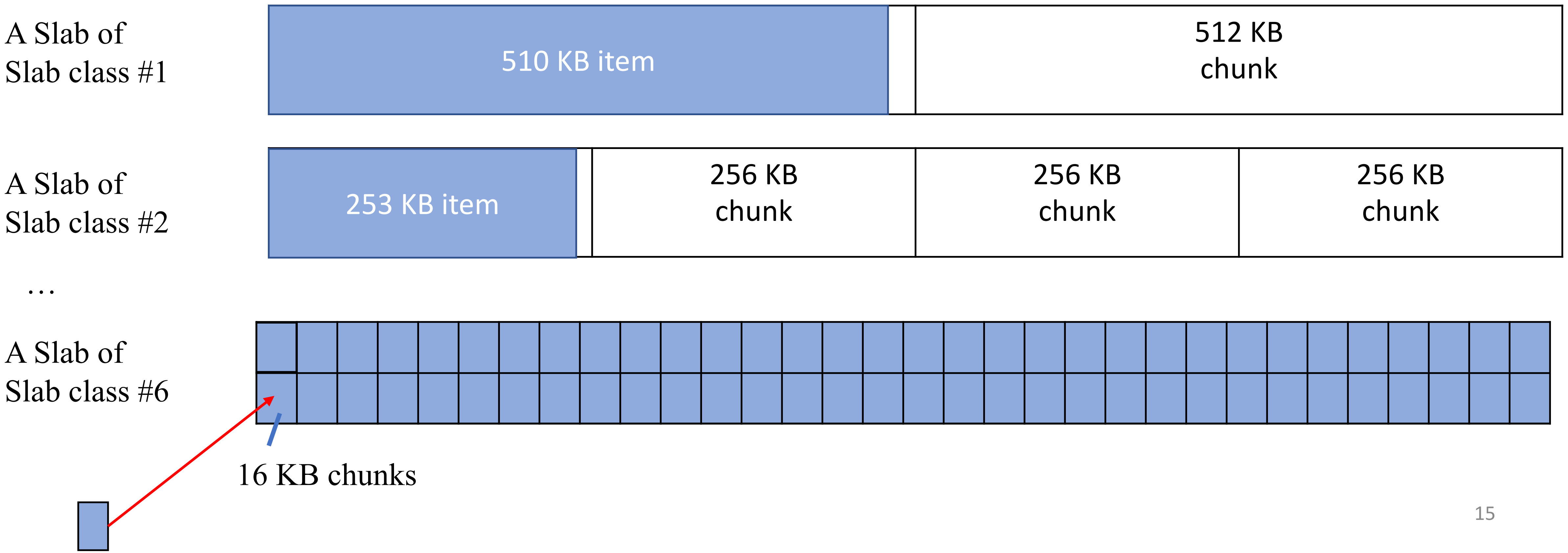
…

A Slab of
Slab class #6



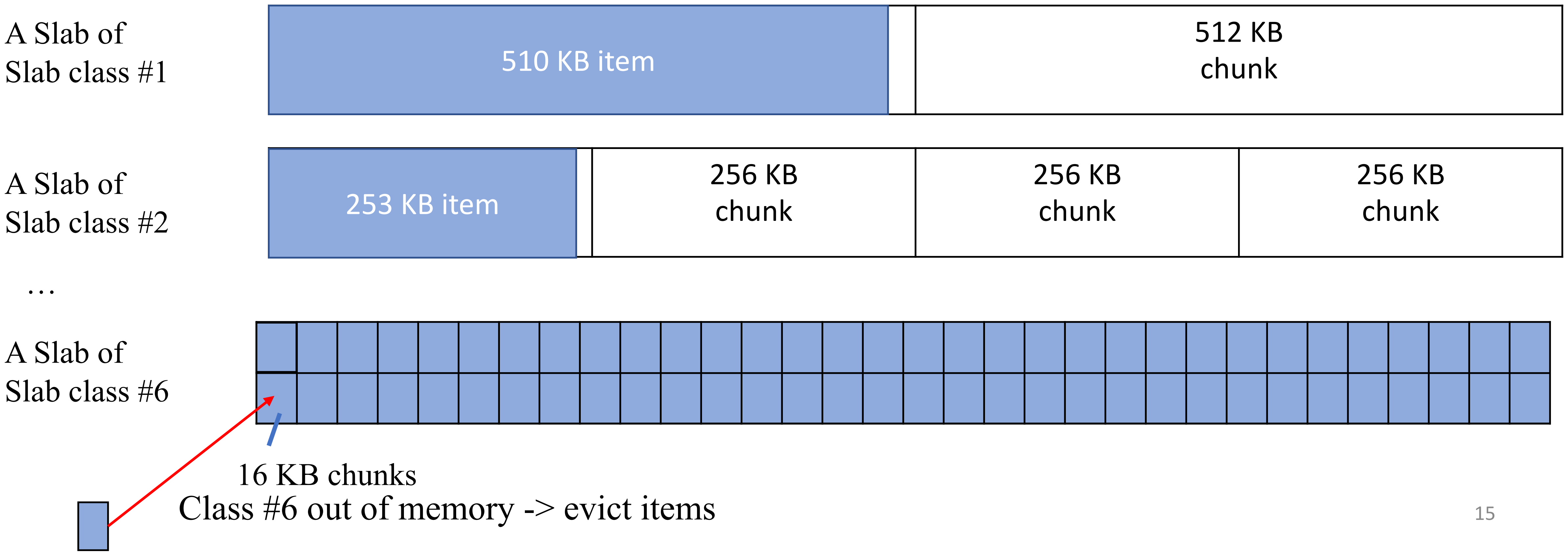
Memcached Slab Allocator : Problem

- When an item demands a memory space, the best fitting chunk is selected for it
  - Most hot items are small items -> slabs with *big* chunks have low utilities



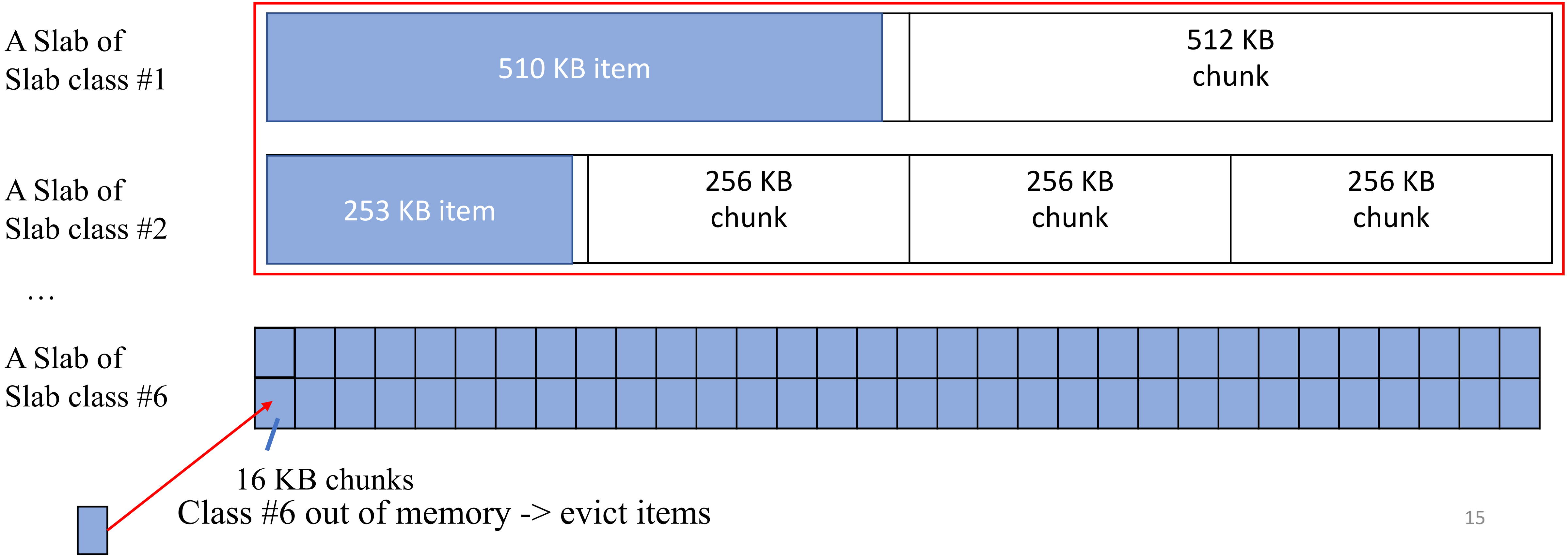
Memcached Slab Allocator : Problem

- When an item demands a memory space, the best fitting chunk is selected for it
  - Most hot items are small items -> slabs with *big* chunks have low utilities



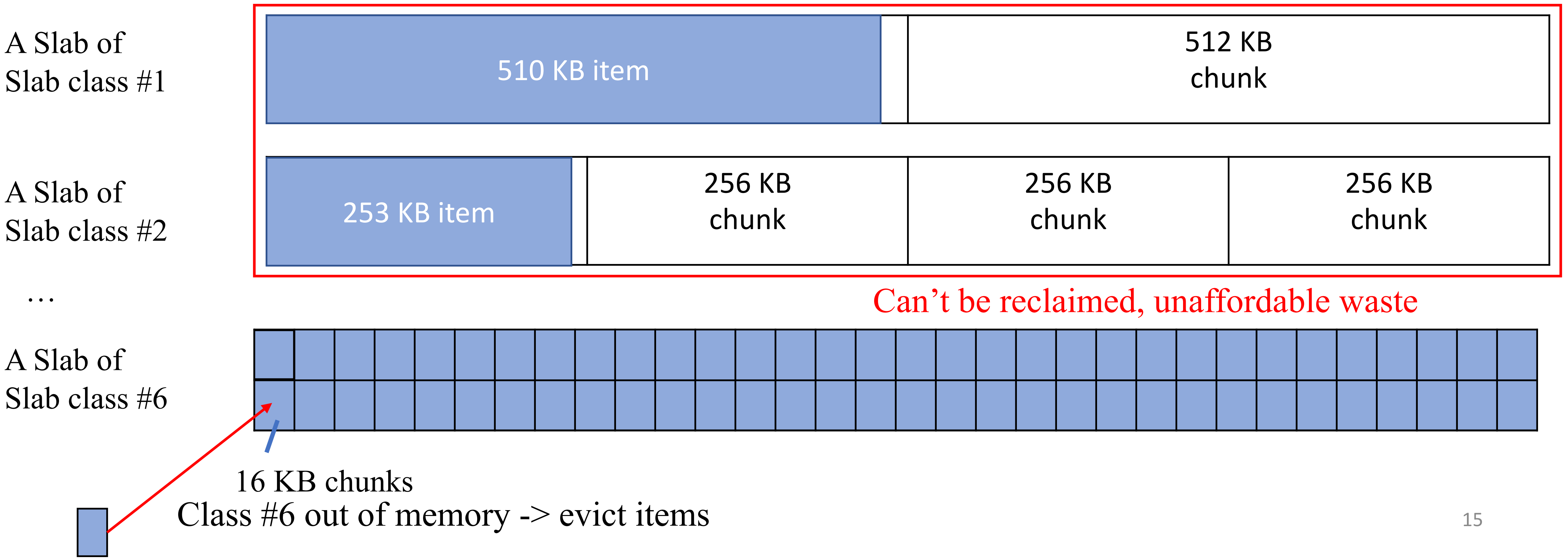
Memcached Slab Allocator : Problem

- When an item demands a memory space, the best fitting chunk is selected for it
  - Most hot items are small items -> slabs with *big* chunks have low utilities
  - Slabs with low utility are *NOT* reclaimable -> unaffordable waste



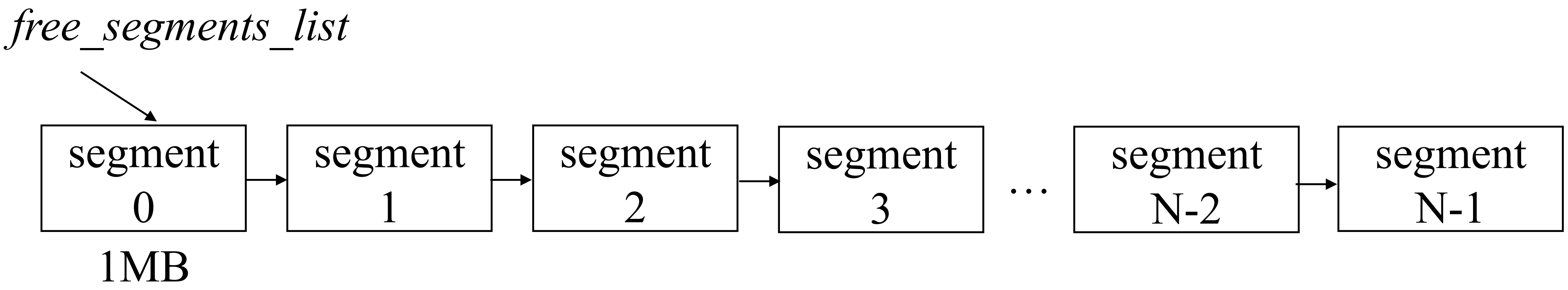
Memcached Slab Allocator : Problem

- When an item demands a memory space, the best fitting chunk is selected for it
  - Most hot items are small items -> slabs with *big* chunks have low utilities
  - Slabs with low utility are *NOT* reclaimable -> unaffordable waste



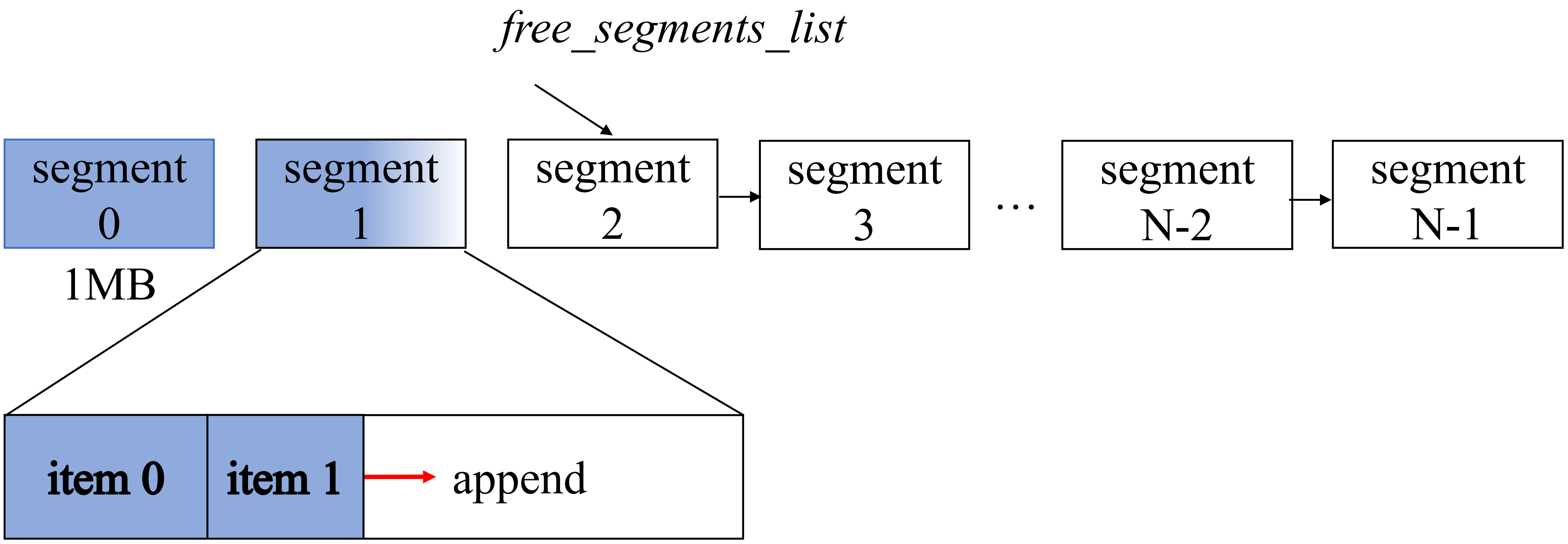
Log Structure Allocator

- The fast write region is organized as linked 1MB segments



Log Structure Allocator

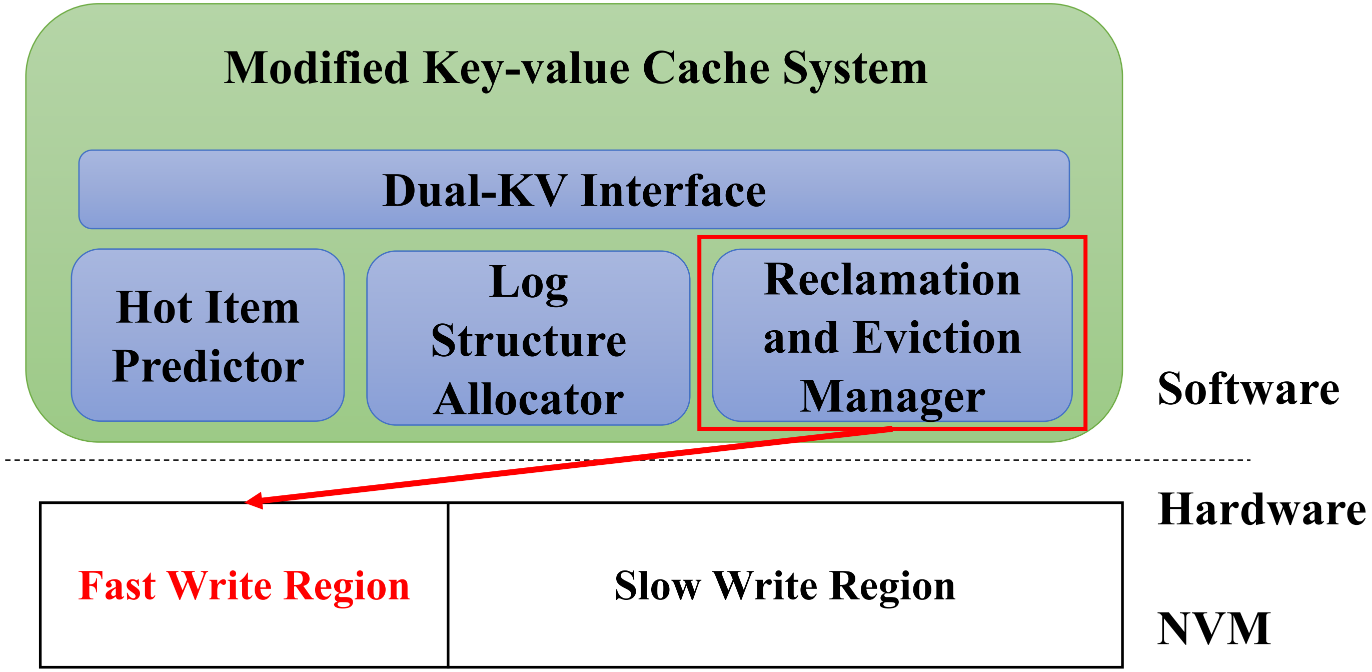
- Items are sequentially appended to the segment



Managing the Fast Write Region

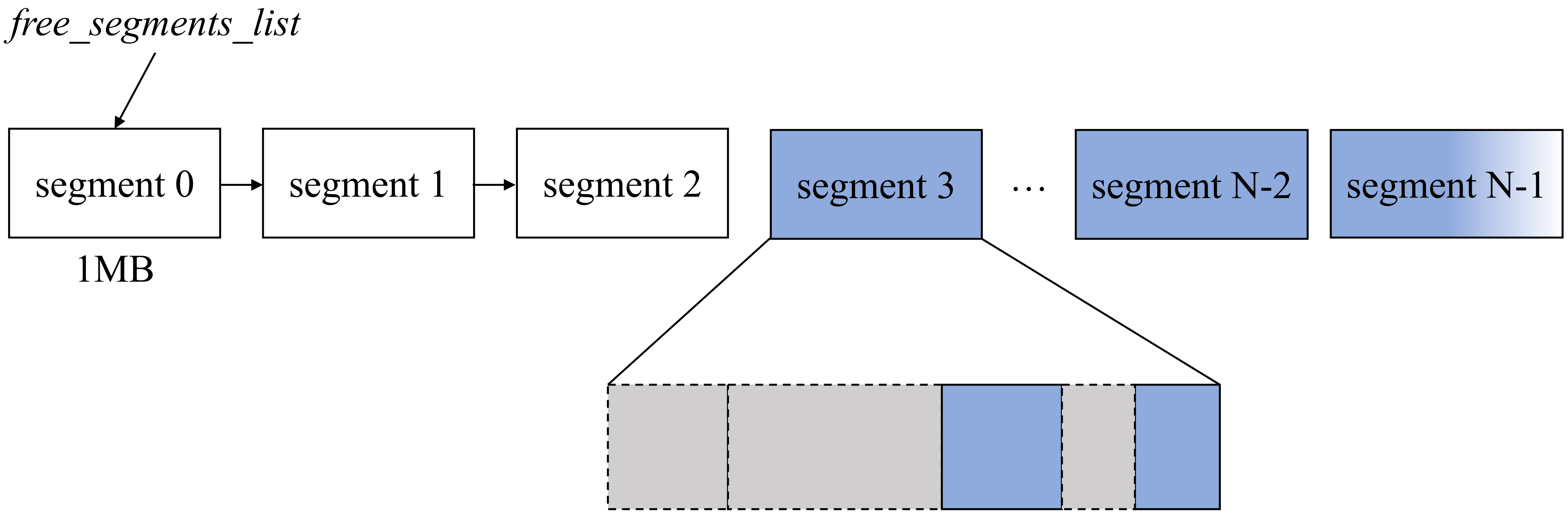
- Reclaim fast write region space when the free segments # is low

- Evict the out-of-date data
  - Since the data in the fast write region have limited retention time (i.e., 100 seconds in our research)



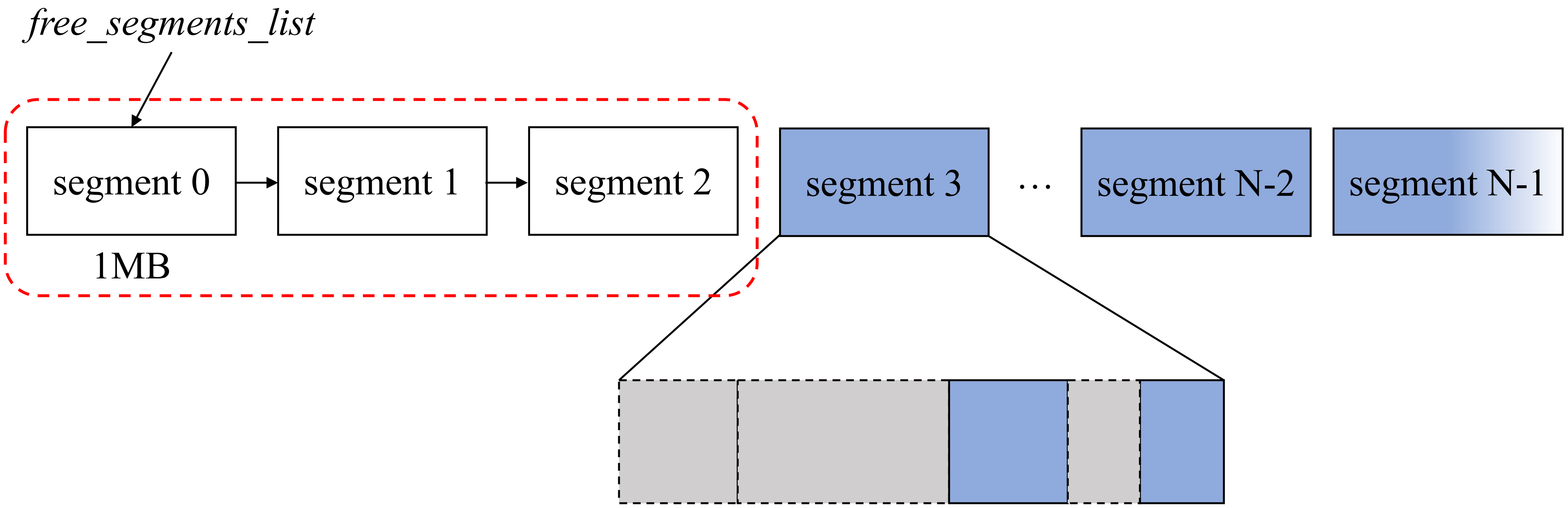
Memory Reclamation : When

- When : The number of free segments falls below a threshold (4 in our research)



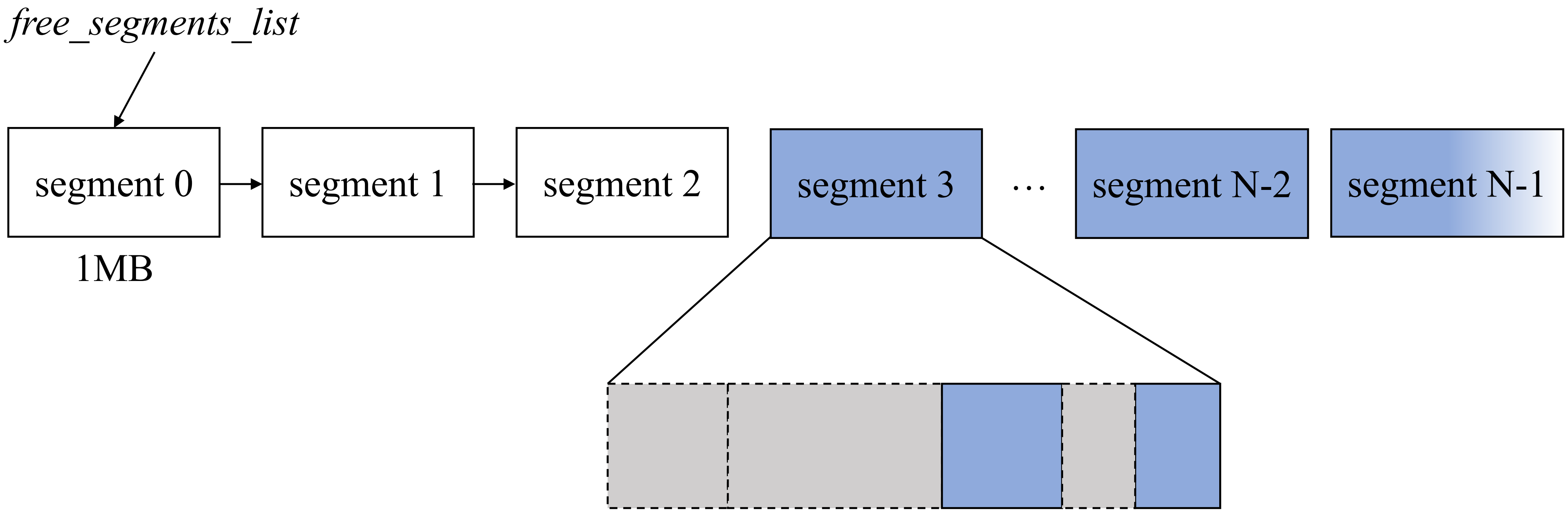
Memory Reclamation : When

- When : The number of free segments falls below a threshold (4 in our research)



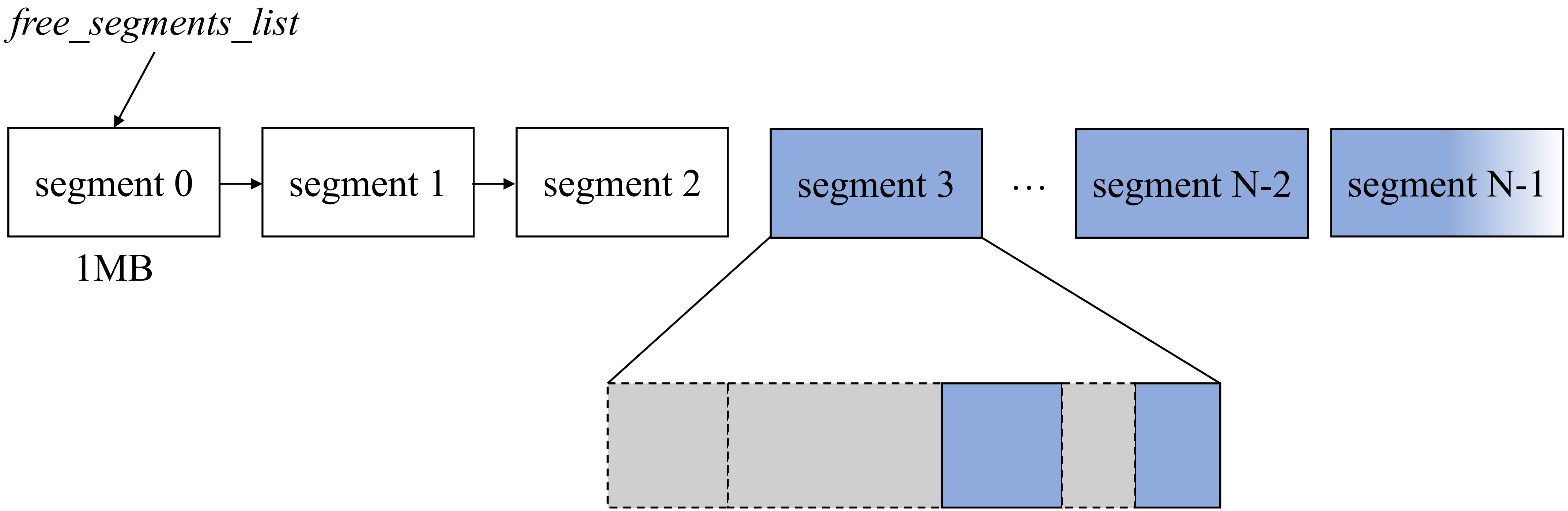
Memory Reclamation : Which and How

- Which : The oldest segment
- How : Drop the live items



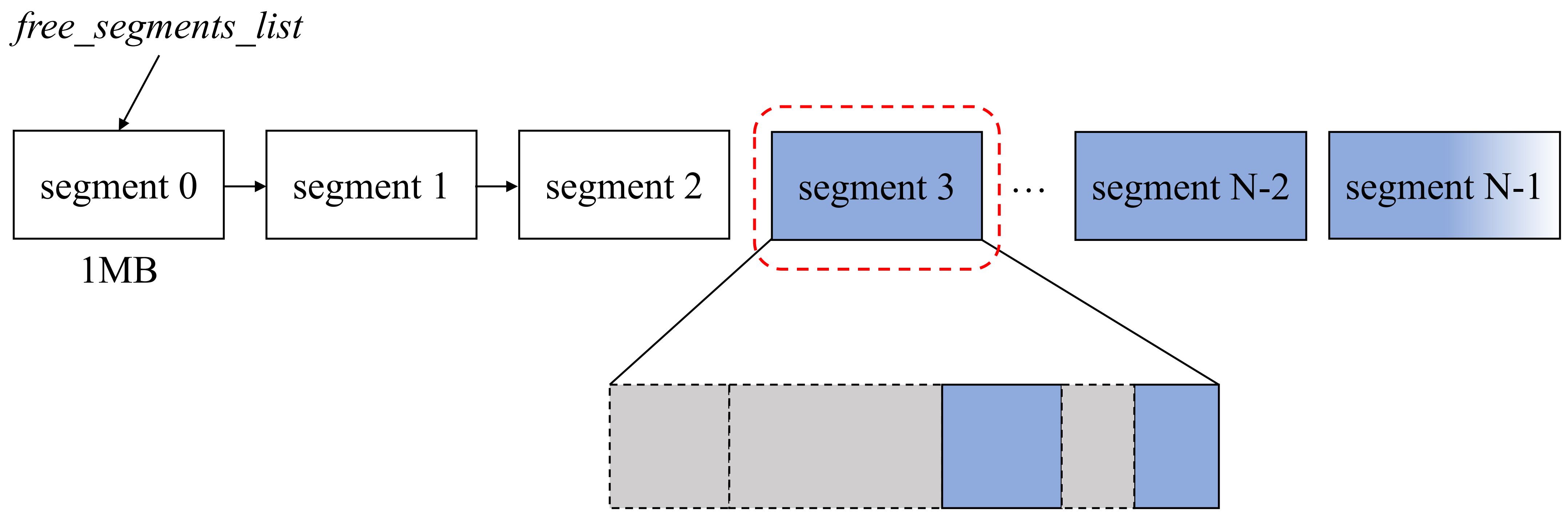
Memory Reclamation : Which and How

- Which : The oldest segment ➡ tend to contain the fewest live items
- How : Drop the live items



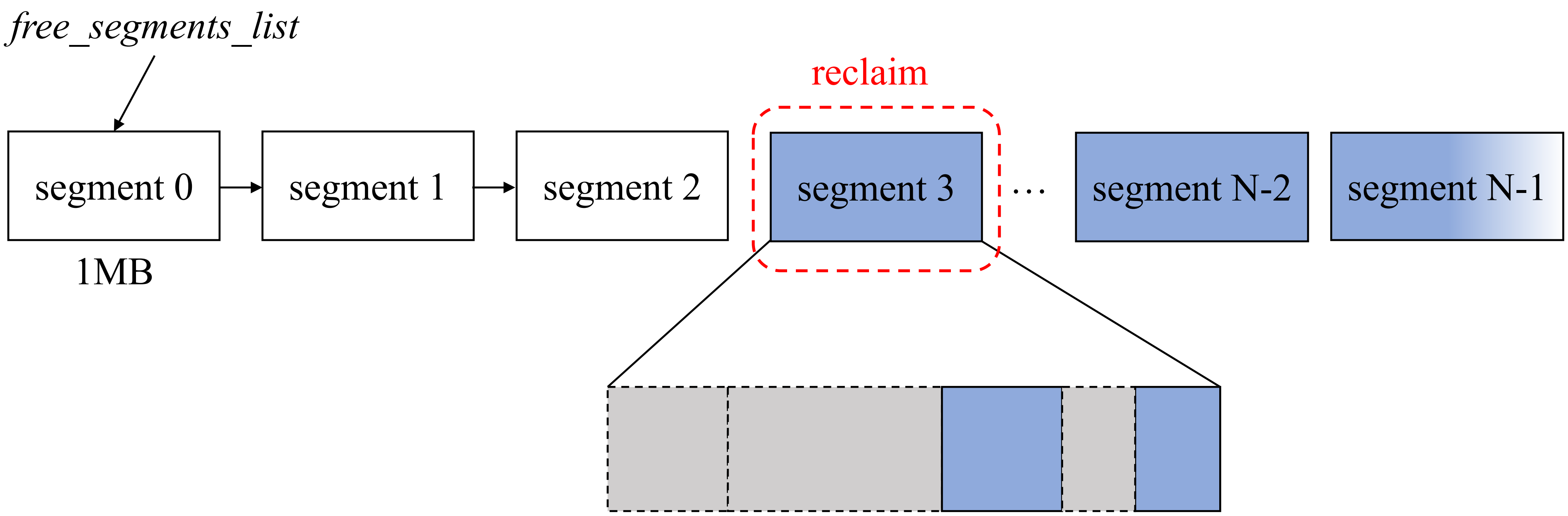
Memory Reclamation : Which and How

- Which : The oldest segment ➡ tend to contain the fewest live items
- How : Drop the live items



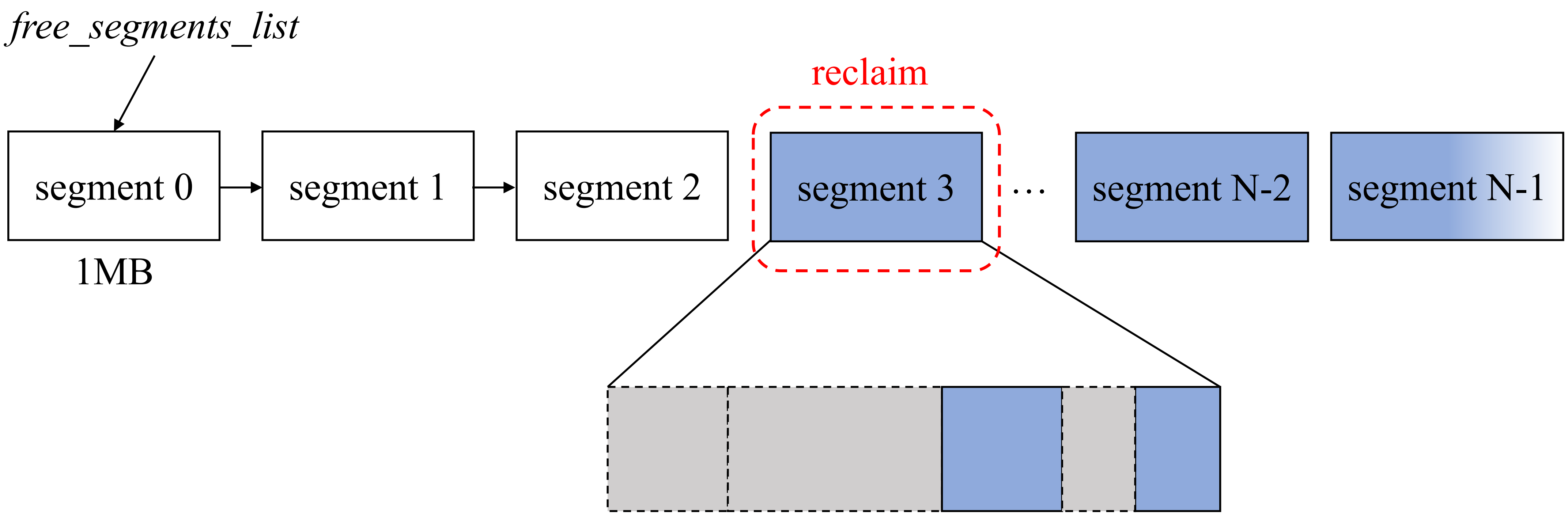
Memory Reclamation : Which and How

- Which : The oldest segment ➡ tend to contain the fewest live items
- How : Drop the live items



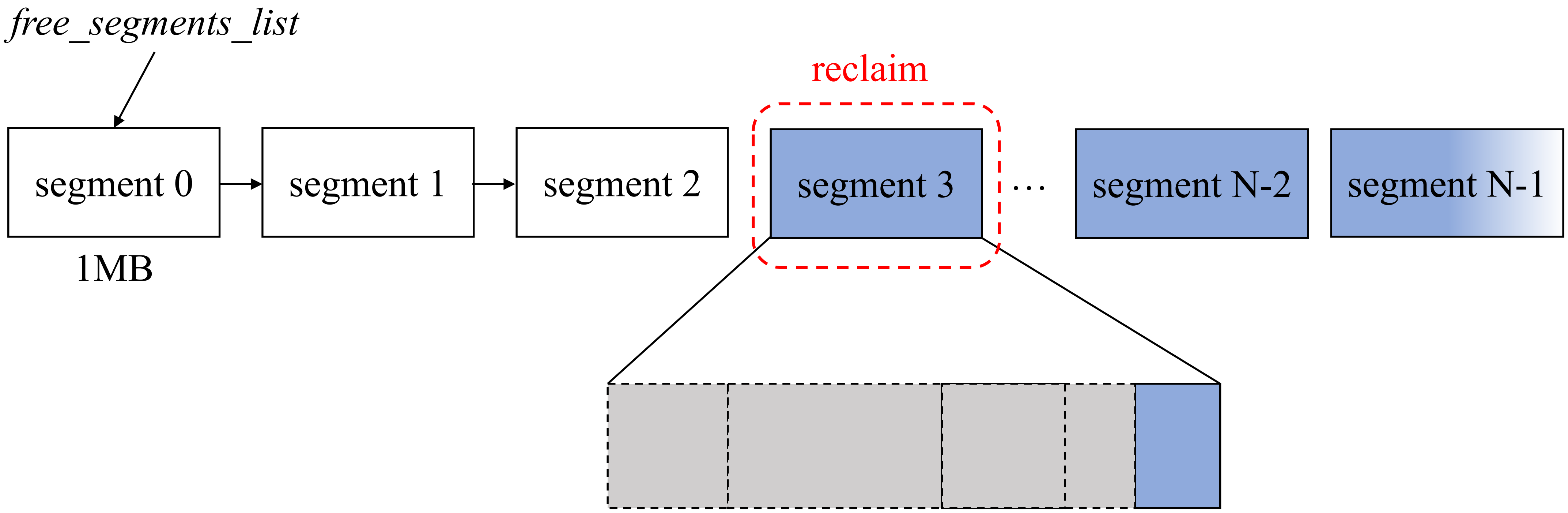
Memory Reclamation : Which and How

- Which : The oldest segment → tend to contain the fewest live items
- How : Drop the live items → easy for implementation, hit rate penalty (but negligible)



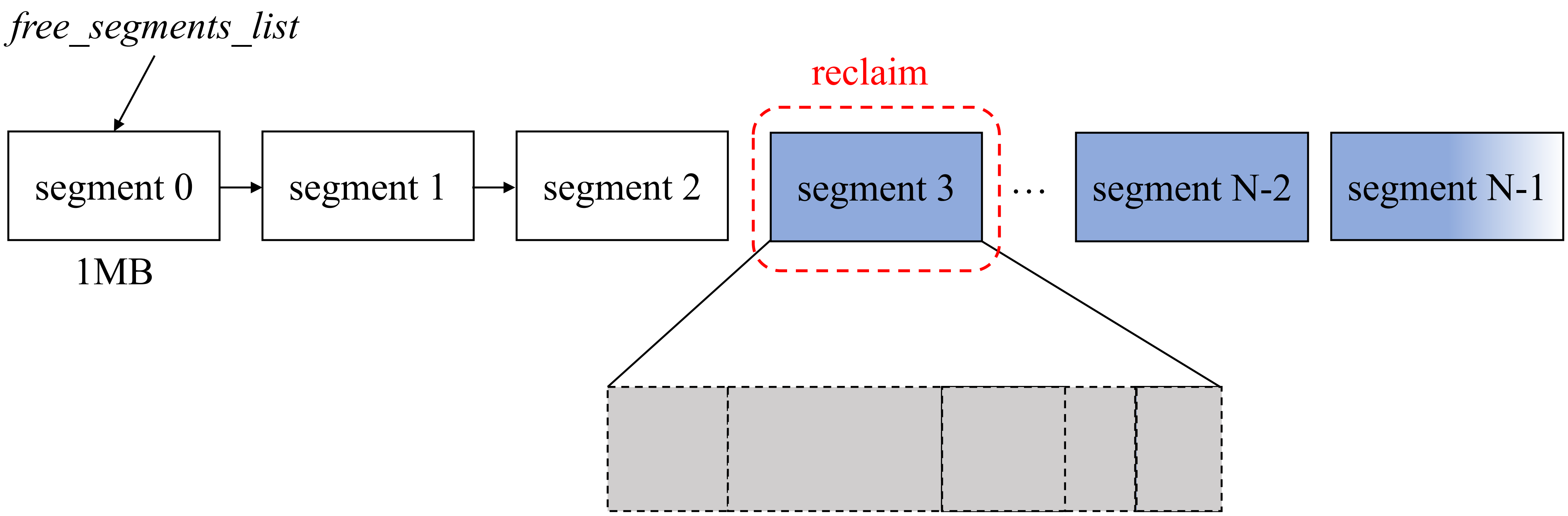
Memory Reclamation : Which and How

- Which : The oldest segment → tend to contain the fewest live items
- How : Drop the live items → easy for implementation, hit rate penalty (but negligible)

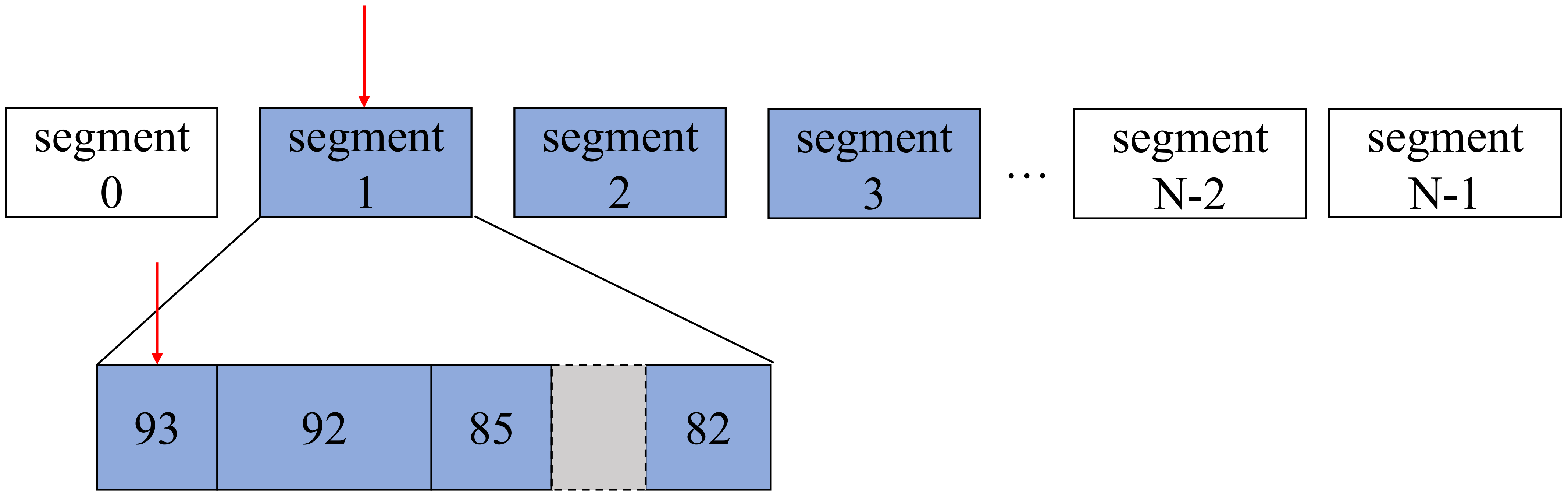


Memory Reclamation : Which and How

- Which : The oldest segment → tend to contain the fewest live items
- How : Drop the live items → easy for implementation, hit rate penalty (but negligible)



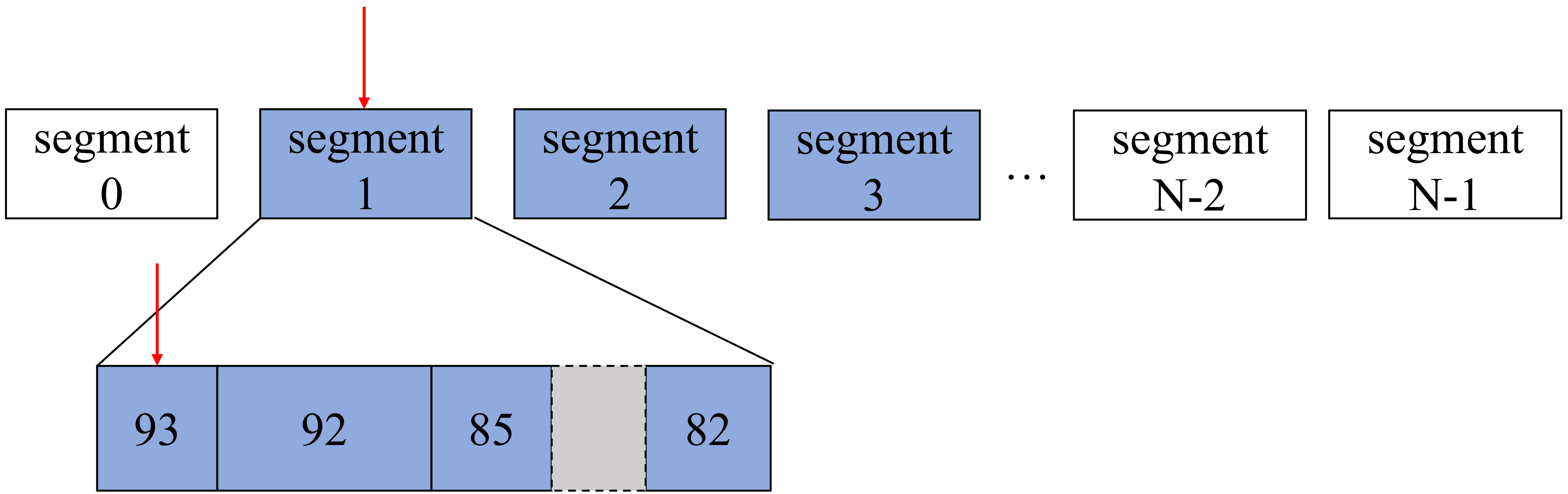
Aged Items Eviction



Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

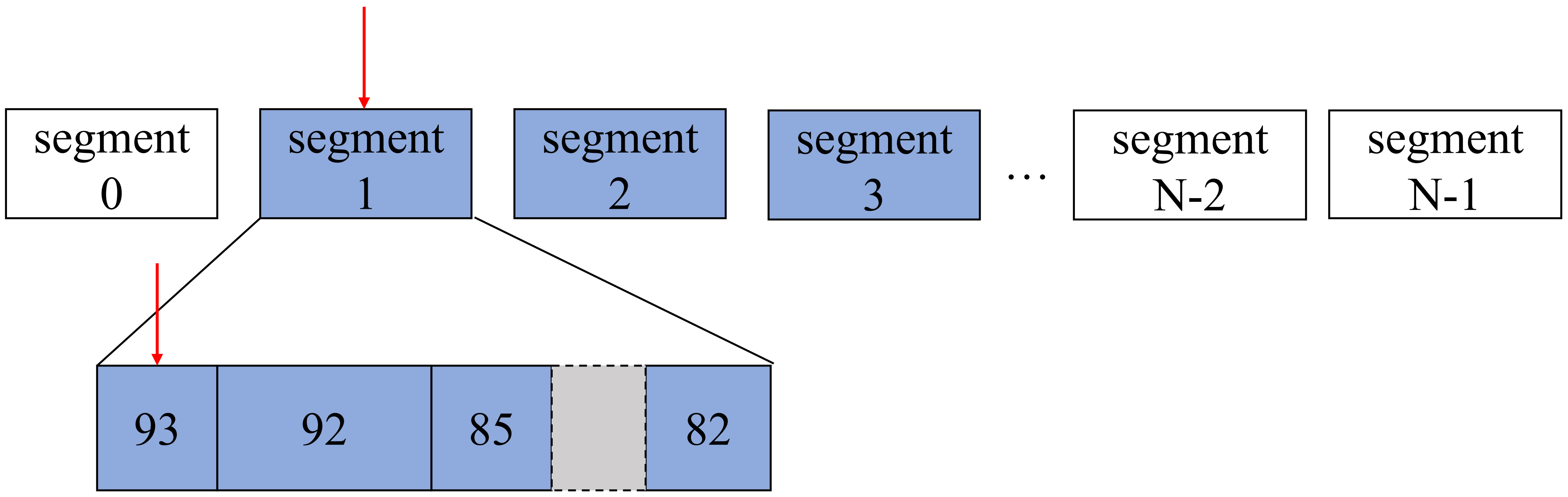
  -> need to be evict before they become out-of-date



Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

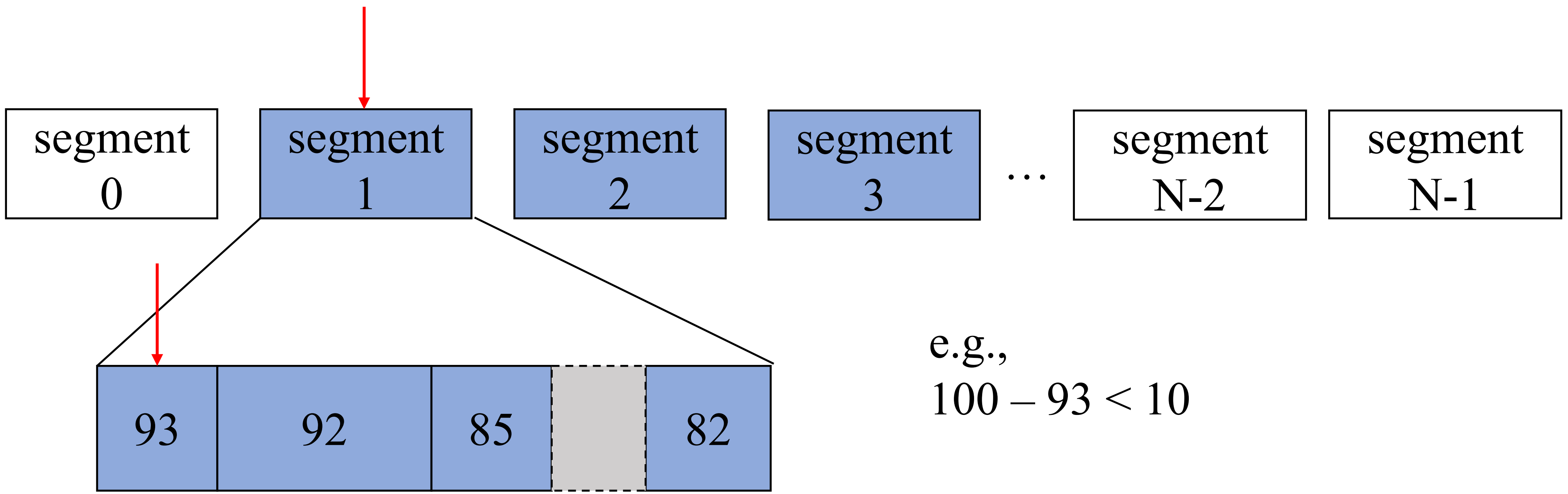
  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



e.g.,

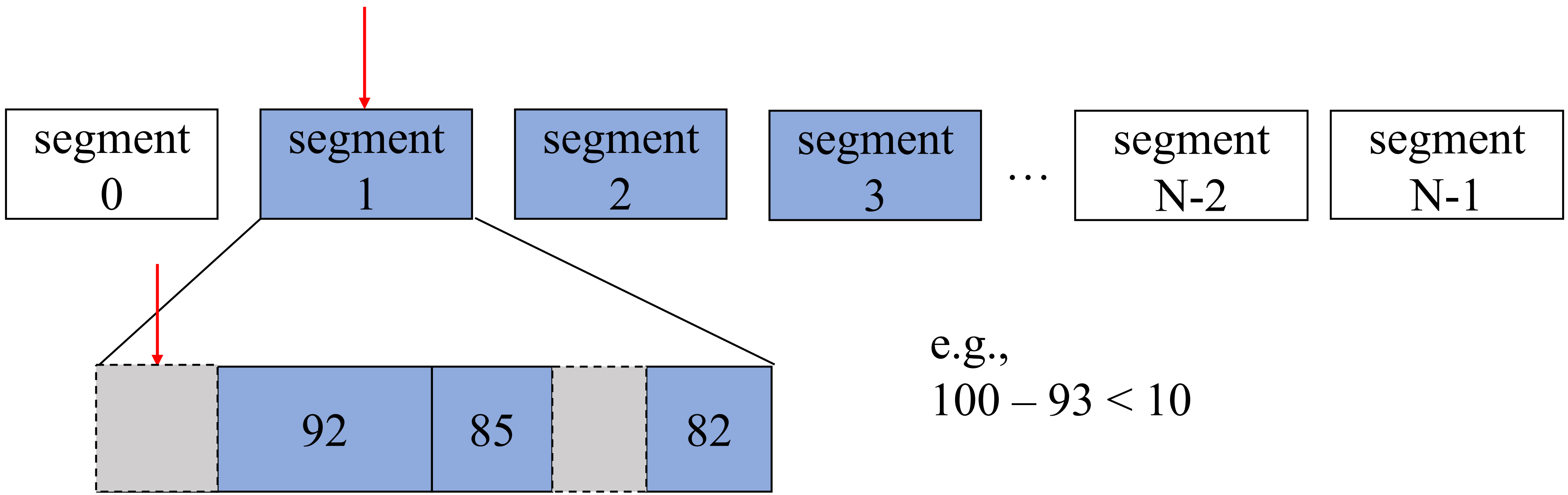
100 – 93 < 10

evict the item

Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)

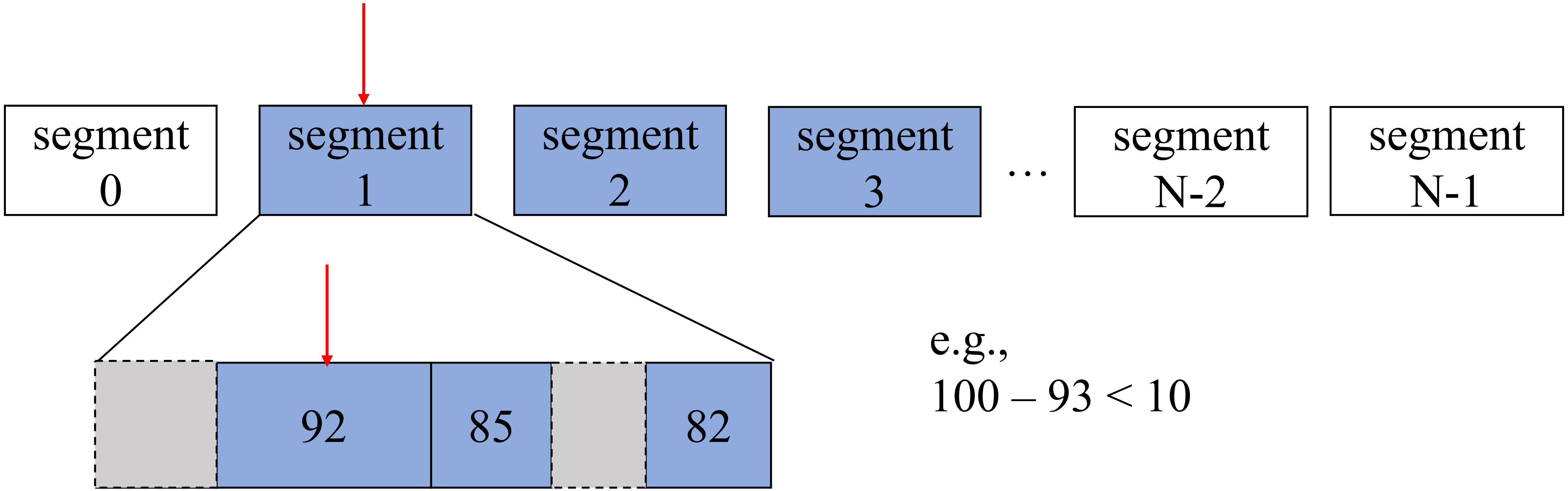


e.g.,
$100 - 93 < 10$

evict the item

Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds
  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



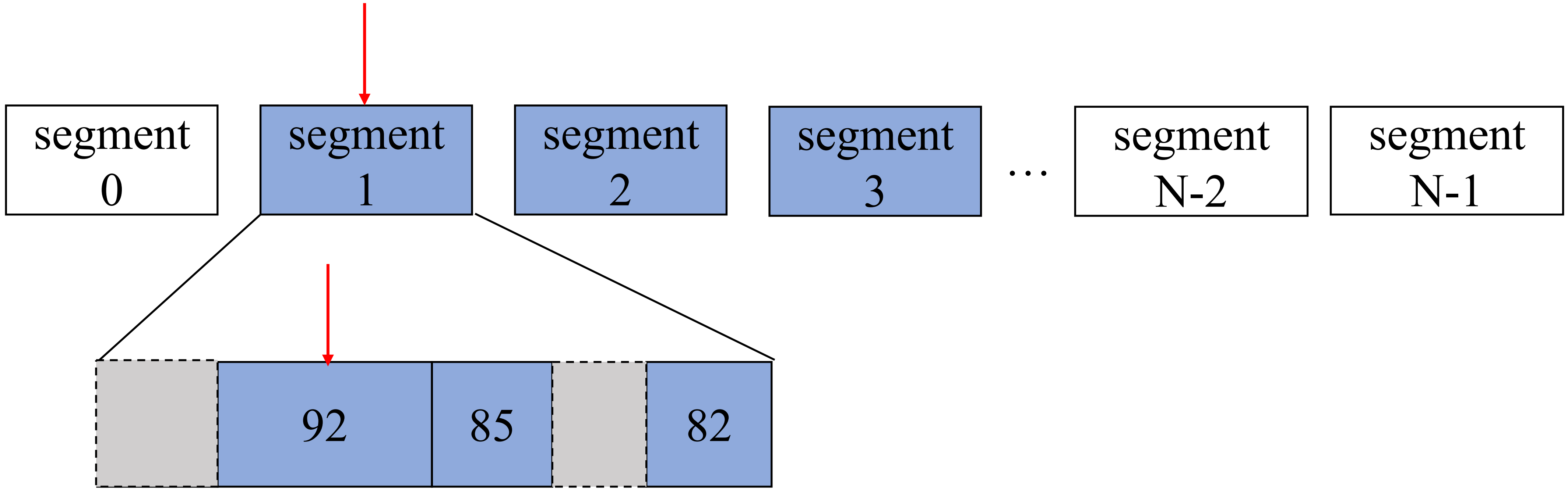
e.g.,
100 – 93 < 10

evict the item

Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)

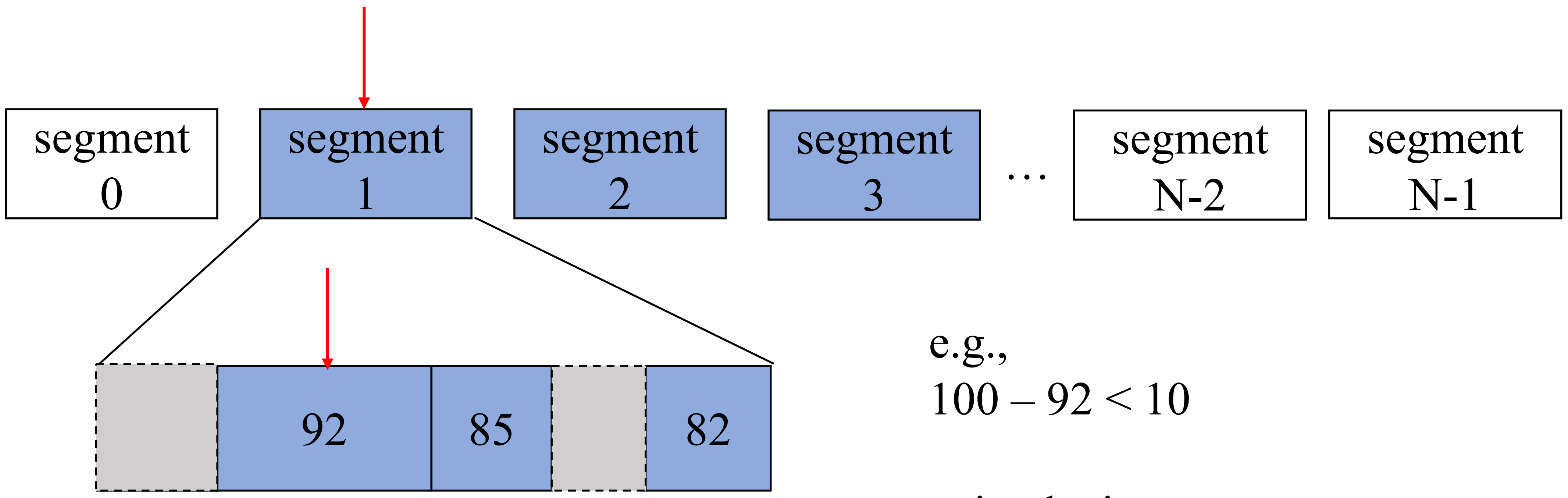


Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date

- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



e.g.,

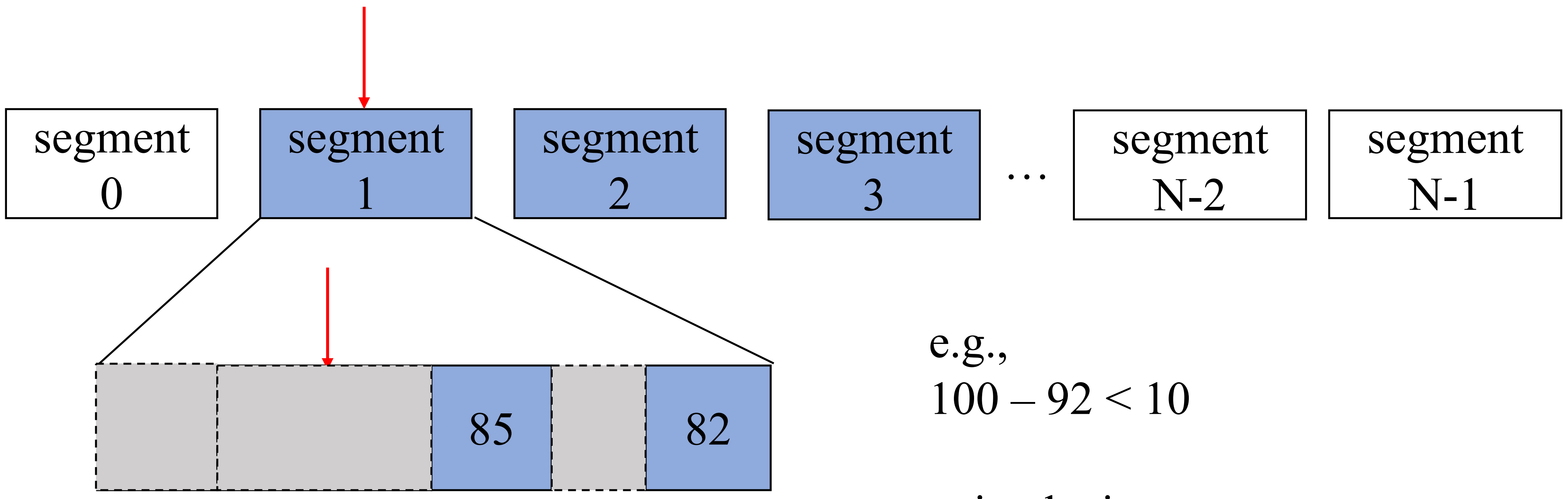
100 – 92 < 10

evict the item

Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



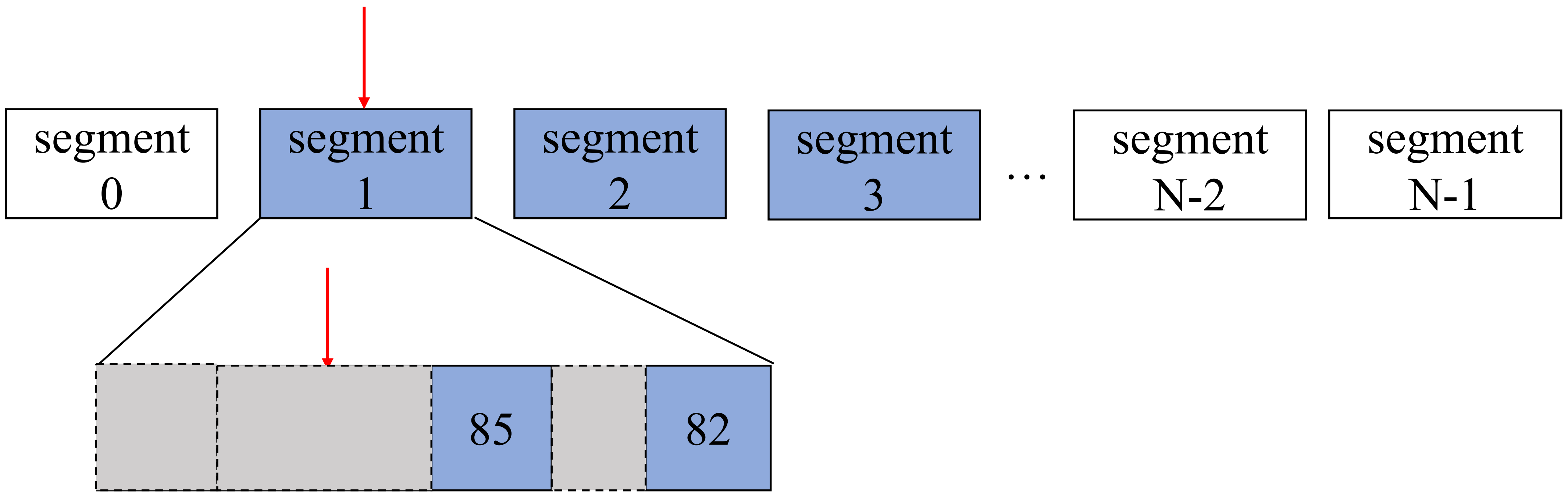
e.g.,
100 – 92 < 10

evict the item

Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)

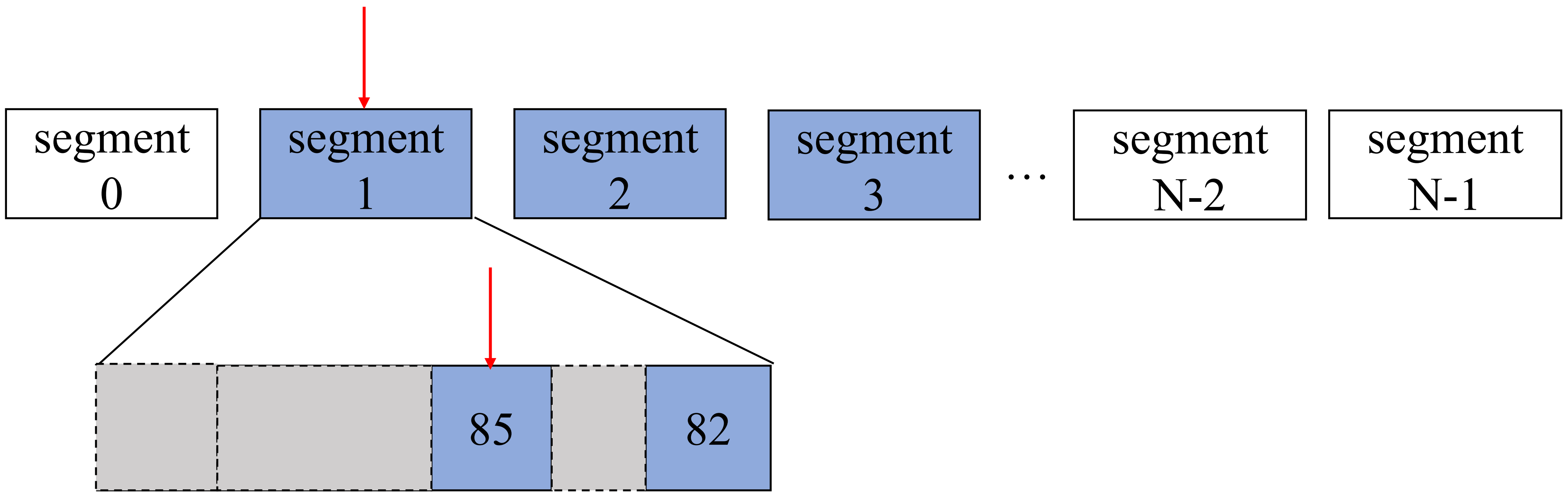


Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date

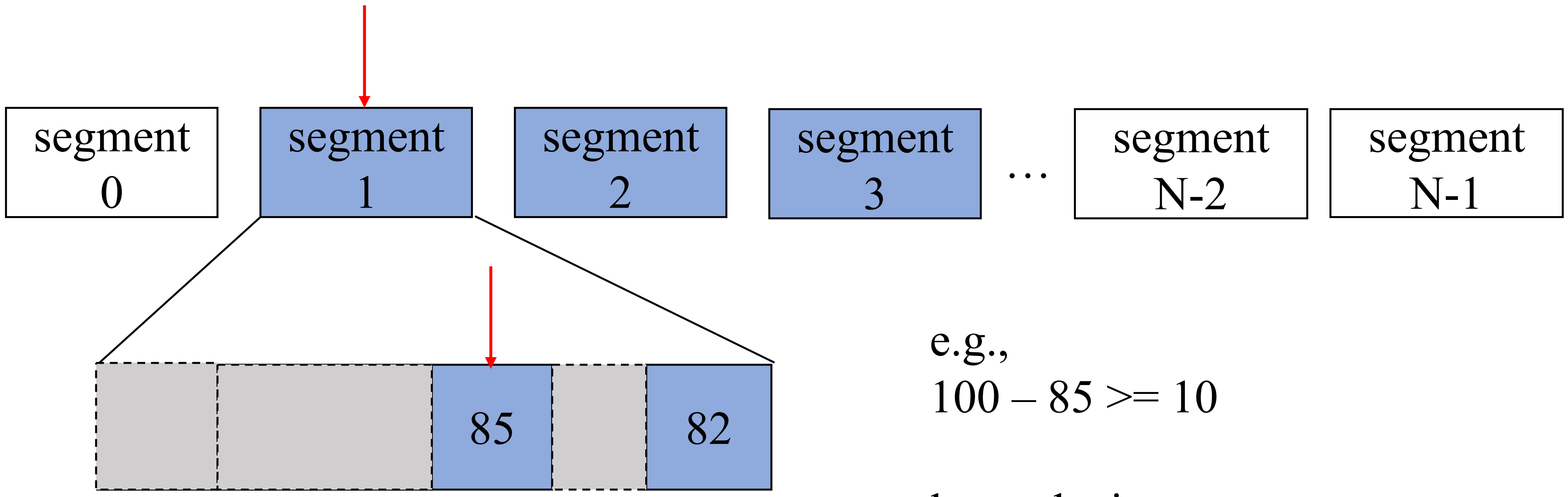
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



e.g.,

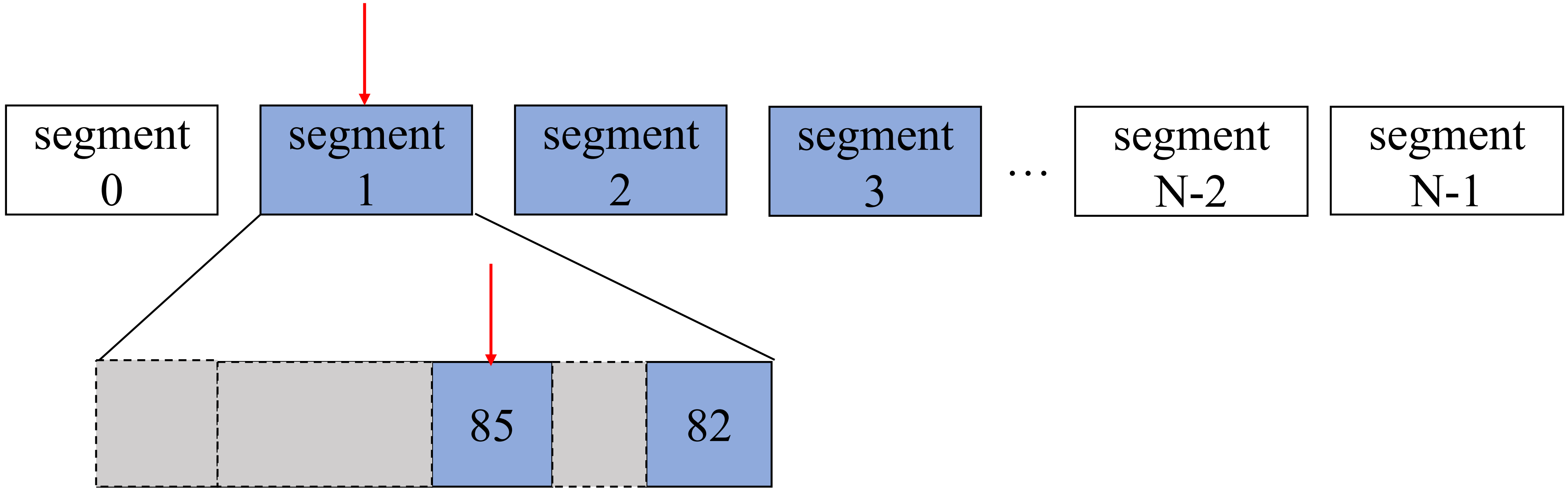
100 – 85 >= 10

leave the item

Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)

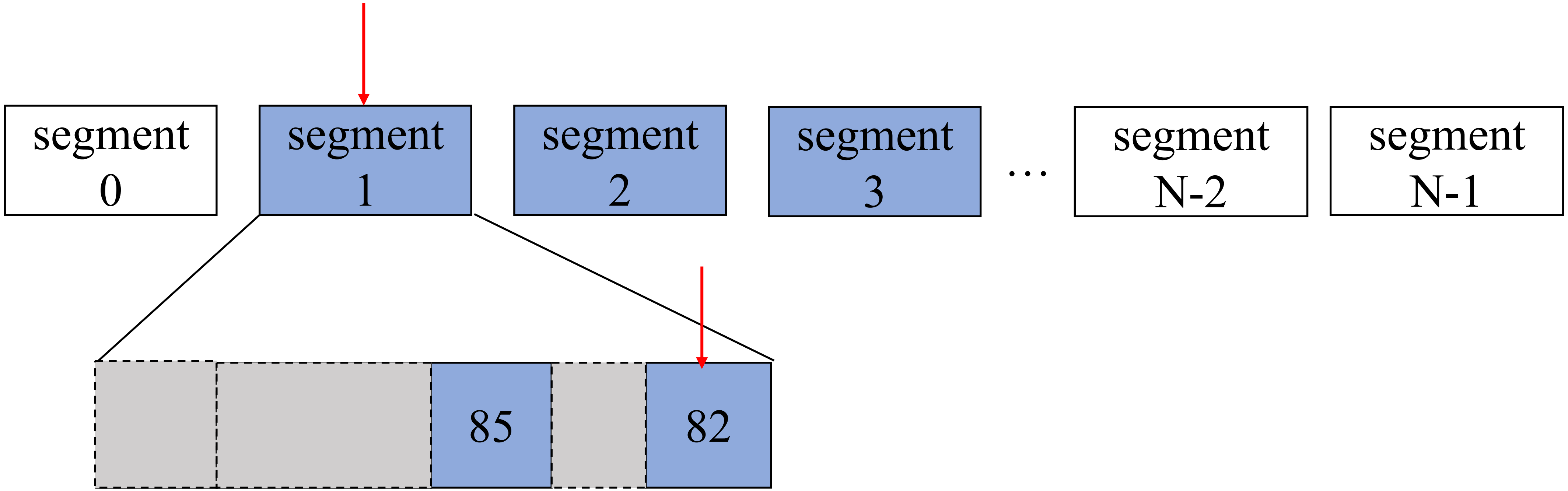


Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date

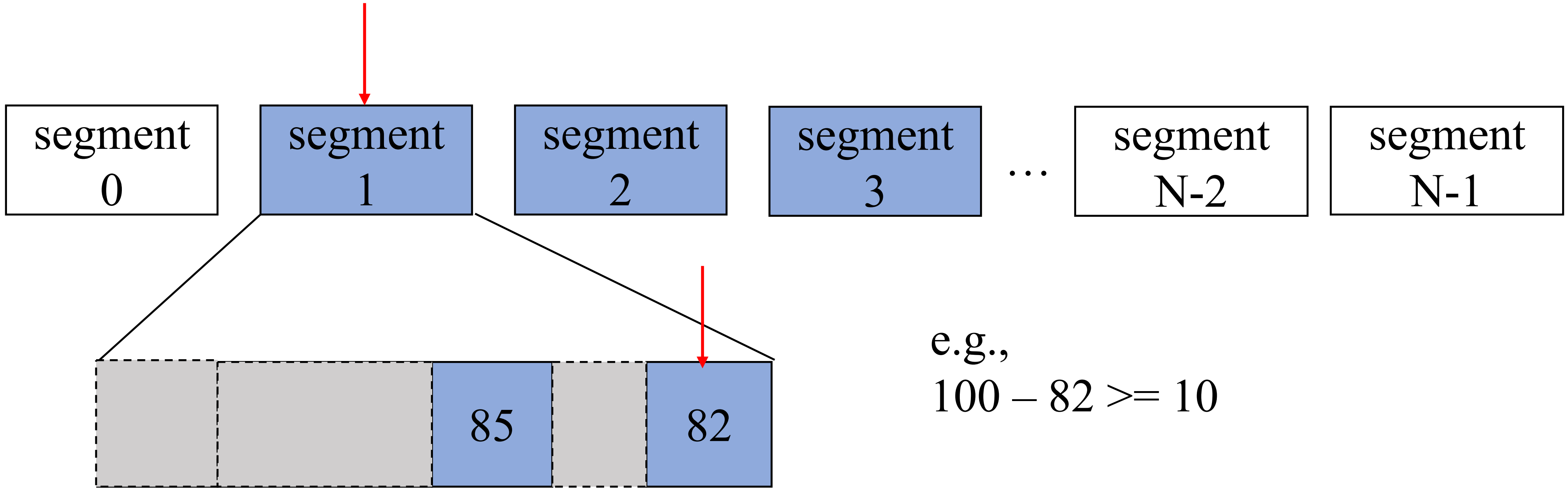
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



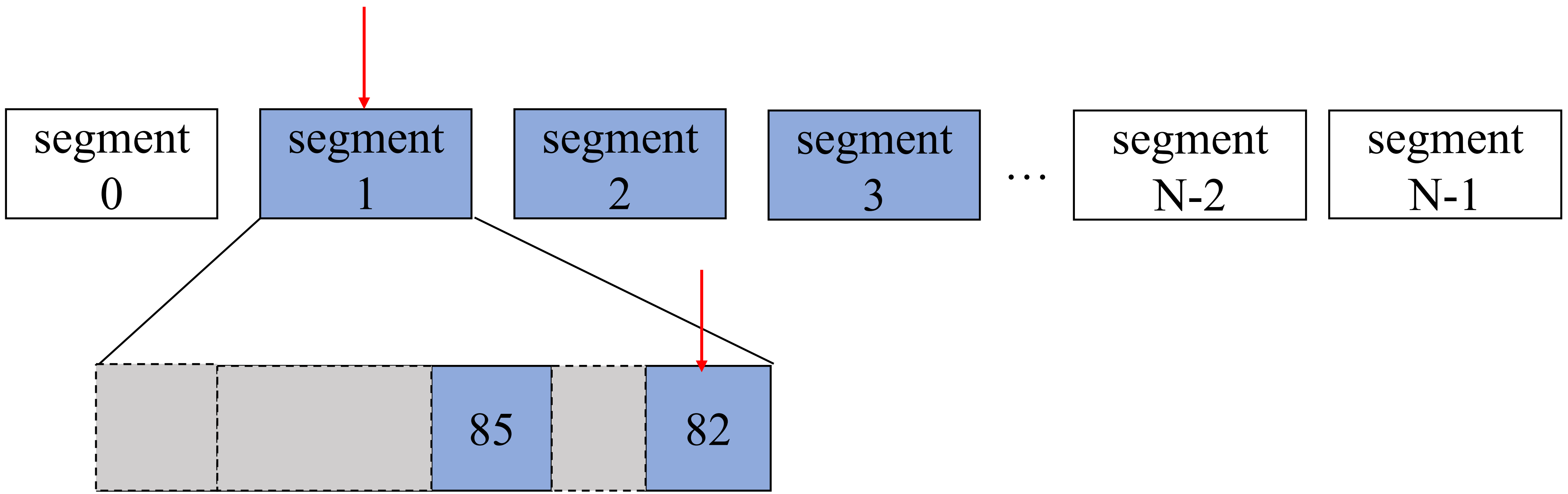
e.g.,
100 – 82 >= 10

leave the item

Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

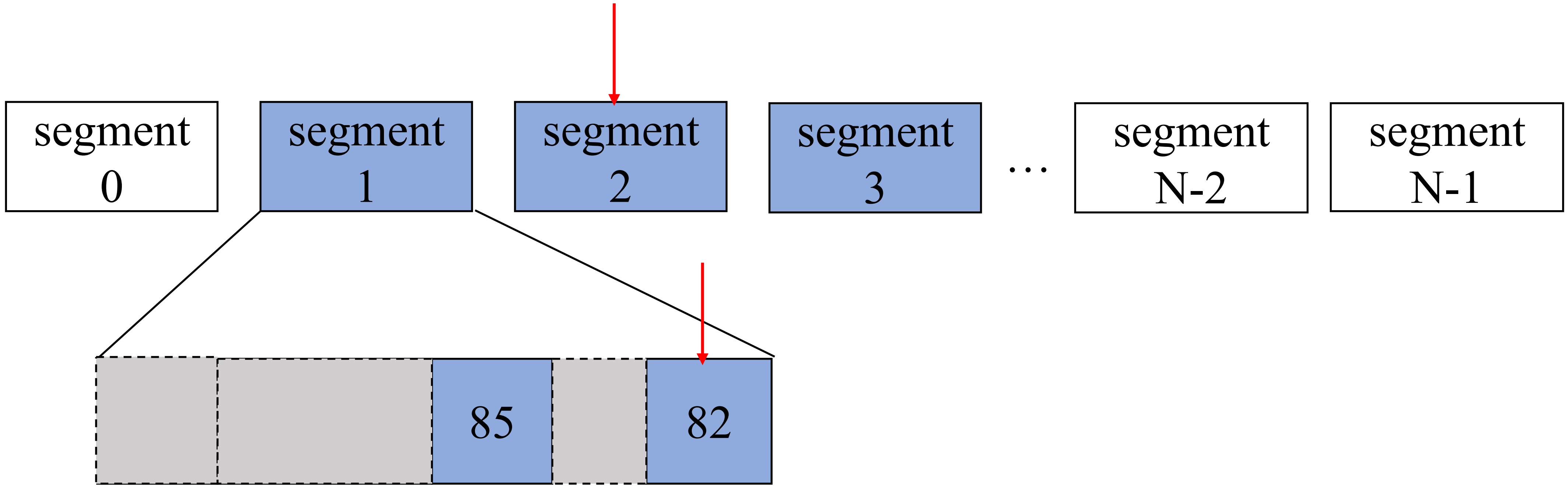
  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



Aged Items Eviction

- Why: The retention time of fast written items are 100 seconds

  -> need to be evict before they become out-of-date
- How: An eviction thread periodically scans the segments and evicts out-of-date items
  - Eviction period: 10 seconds in our work (discussed in the next slide)



Computation Overhead vs. Hit Rate

eviction period ↓ computational overhead ↑

eviction period ↑ Items may be evicted too early, hit rate penalty ↑

Dual-KV Interface

- Allow key-value cache systems to take advantage of the dual retention write scheme without significant software refactoring

Dual-KV Interface	**Feature**
classify_item_write_mode()	Predict appropriate write mode for each item
segment_alloc_item()	Allocate memory in the fast write region
segment_free_item()	Release memory in the fast write region

Hardware Environment: Dual Retention Support & Super Capacity

- Hardware support includes

Hardware Environment: Dual Retention Support & Super Capacity

- Hardware support includes
  - Dual retention NVM
    - Previous studies provided an interface for dual retention mechanisms

Hardware Environment: Dual Retention Support & Super Capacity

- Hardware support includes
  - Dual retention NVM
    - Previous studies provided an interface for dual retention mechanisms

Hardware Environment: Dual Retention Support & Super Capacity

- Hardware support includes
  - Dual retention NVM
    - Previous studies provided an interface for dual retention mechanisms

M. Zhang, L. Zhang, L. Jiang, Z. Liu, and F. T. Chong, "Balancing performance and lifetime of MLC PCM by using a region retention monitor," in *Symp. on High-Performance Computer Architecture (HPCA)*, 2017, pp. 385–396

R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Hardware Environment: Dual Retention Support & Super Capacity

- Hardware support includes
  - Dual retention NVM
    - Previous studies provided an interface for dual retention mechanisms
  - Supercapacitor
    - Supply power to refresh the data into NVM with slow writes upon a power failure
    - Existing NVDIMMs support data refreshing in DRAM to NAND flash chips upon a power failure

M. Zhang, L. Zhang, L. Jiang, Z. Liu, and F. T. Chong, "Balancing performance and lifetime of MLC PCM by using a region retention monitor," in *Symp. on High-Performance Computer Architecture (HPCA)*, 2017, pp. 385–396

R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Hardware Environment: Dual Retention Support & Super Capacity

- Hardware support includes
  - Dual retention NVM
    - Previous studies provided an interface for dual retention mechanisms
  - Supercapacitor
    - Supply power to refresh the data into NVM with slow writes upon a power failure
    - Existing NVDIMMs support data refreshing in DRAM to NAND flash chips upon a power failure

M. Zhang, L. Zhang, L. Jiang, Z. Liu, and F. T. Chong, "Balancing performance and lifetime of MLC PCM by using a region retention monitor," in *Symp. on High-Performance Computer Architecture (HPCA)*, 2017, pp. 385–396

R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Hardware Environment: Dual Retention Support & Super Capacity

- Hardware support includes
  - Dual retention NVM
    - Previous studies provided an interface for dual retention mechanisms
  - Supercapacitor
    - Supply power to refresh the data into NVM with slow writes upon a power failure
    - Existing NVDIMMs support data refreshing in DRAM to NAND flash chips upon a power failure

M. Zhang, L. Zhang, L. Jiang, Z. Liu, and F. T. Chong, "Balancing performance and lifetime of MLC PCM by using a region retention monitor," in *Symp. on High-Performance Computer Architecture (HPCA)*, 2017, pp. 385–396

R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

DDR4 SDRAM NVRDIMM MTA36ASS4G72XF1Z," Micron Technology Inc., 2020. Micron Inc.

Evaluation Setup

CPU	Intel i7-8700
Main Memory	16GB DDR4 for Main Memory 48GB DDR4 for NVM Emulation
Operating System	Linux Kernel 5.4
Default Fast Write Region Size	64MB

NVM Evaluation	Latency (ns)	Write Iterations	Retention Time (sec)
Slow Write	1,425	5.7	10^7
Fast Write	678	2.7	10^2

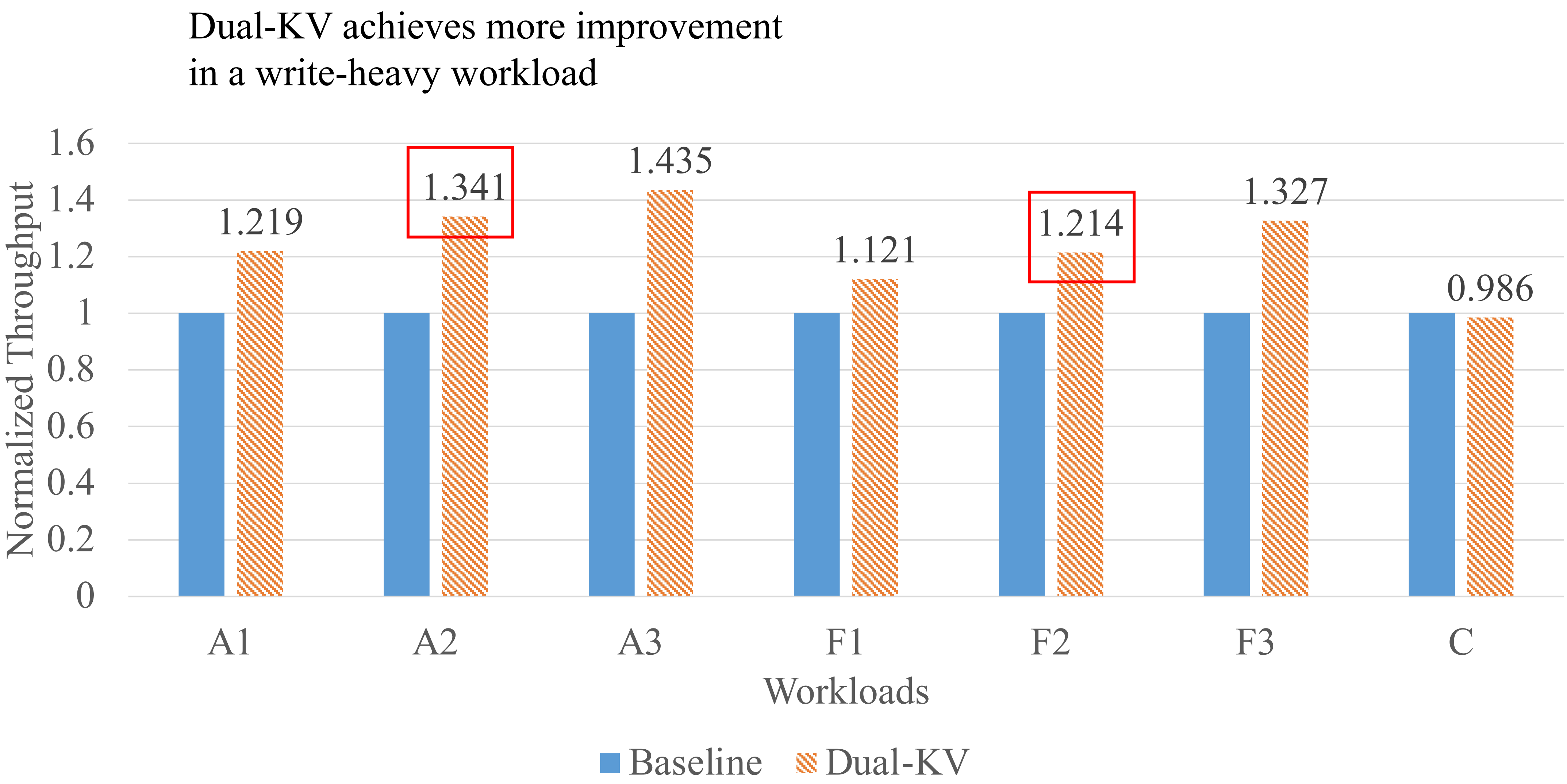
R.-S. Liu *et al.*, "NVM duet: Unified working memory and persistent store architecture," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 455–470, 2014

Workloads

- Yahoo! Cloud Serving Benchmark (YCSB)
  - 10 M operations on 10M items
  - Zipfian constant is 0.99 -> most selected items are small

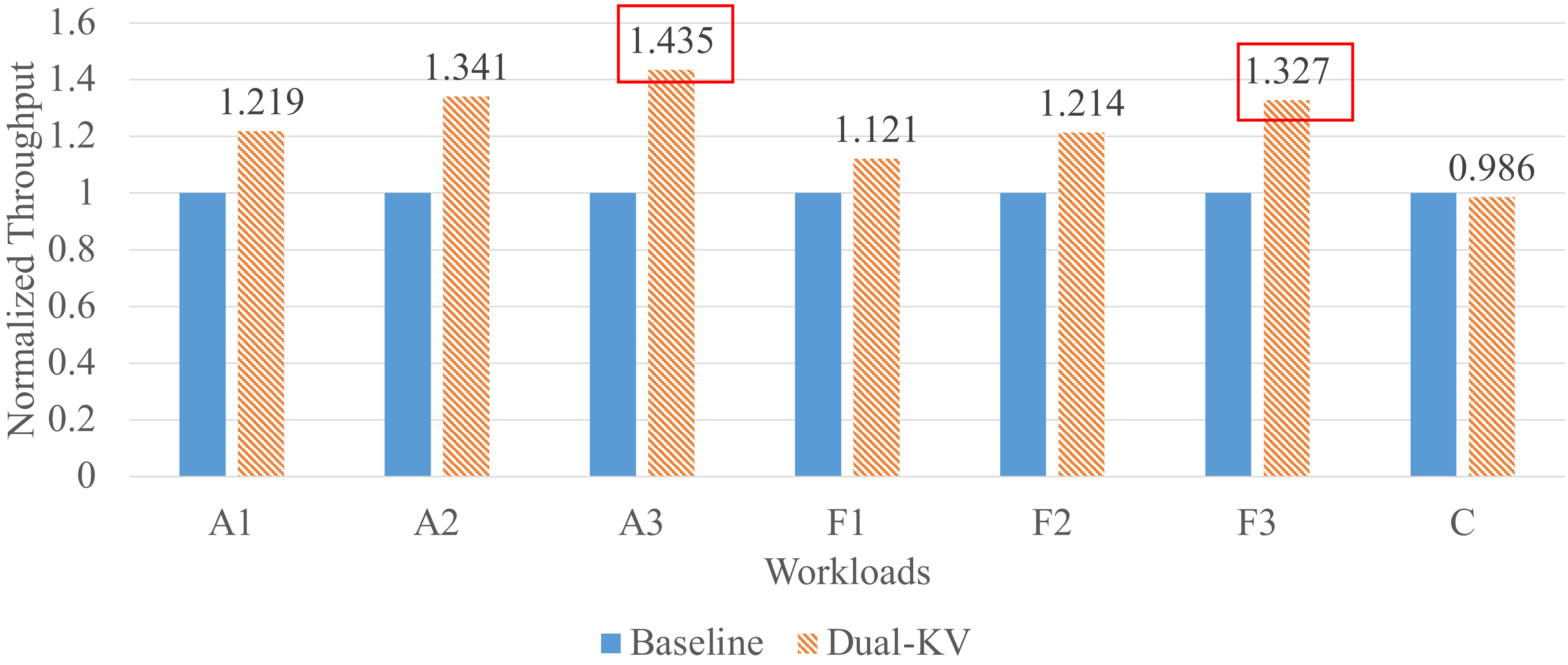
Workload	Operation	Value Size
A1	read 50%, update 50%	64B ~ 10KB
A2	read 5%, update 95%	64B ~ 10KB
A3	read 50%, update 50%	64B ~ 100KB
F1	read 50%, read-modify-write 50%	64B ~ 10KB
F2	read 5%, read-modify-write 95%	64B ~ 10KB
F3	read 50%, read-modify-write 50%	64B ~ 100KB
C	read 100%	64B ~ 10KB

Throughput Improvement

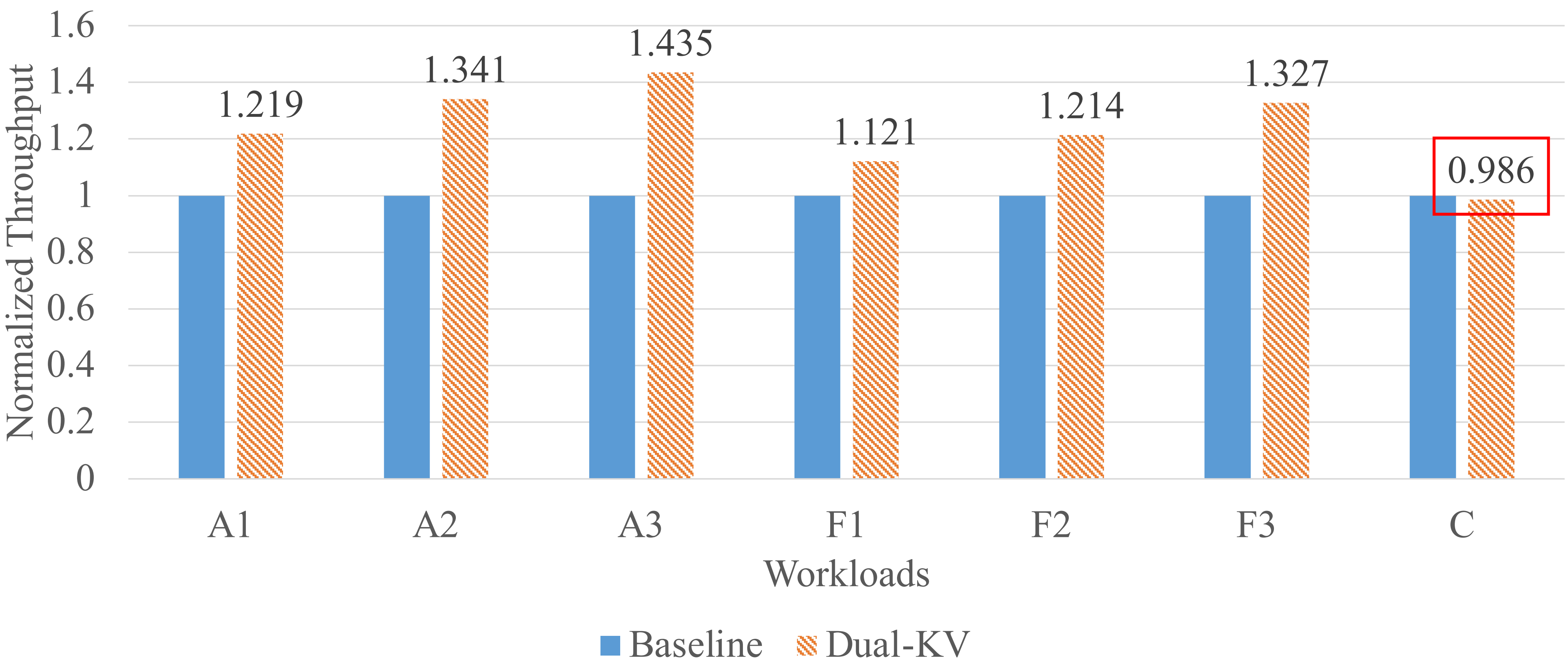


Throughput Improvement

Fast writes reduce even more **queueing delays** for the following requests when the selected fast write item is large

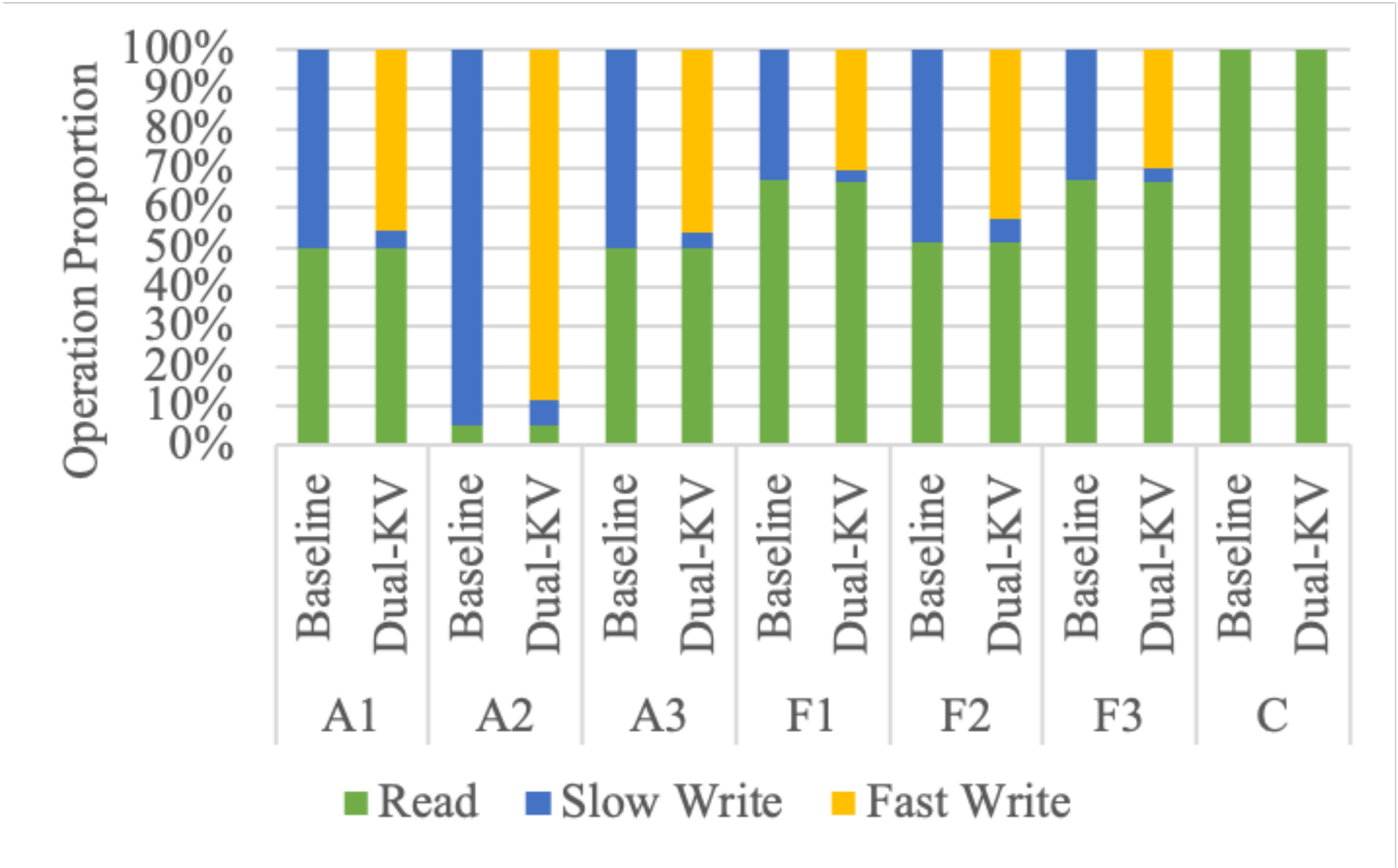


Read Overhead



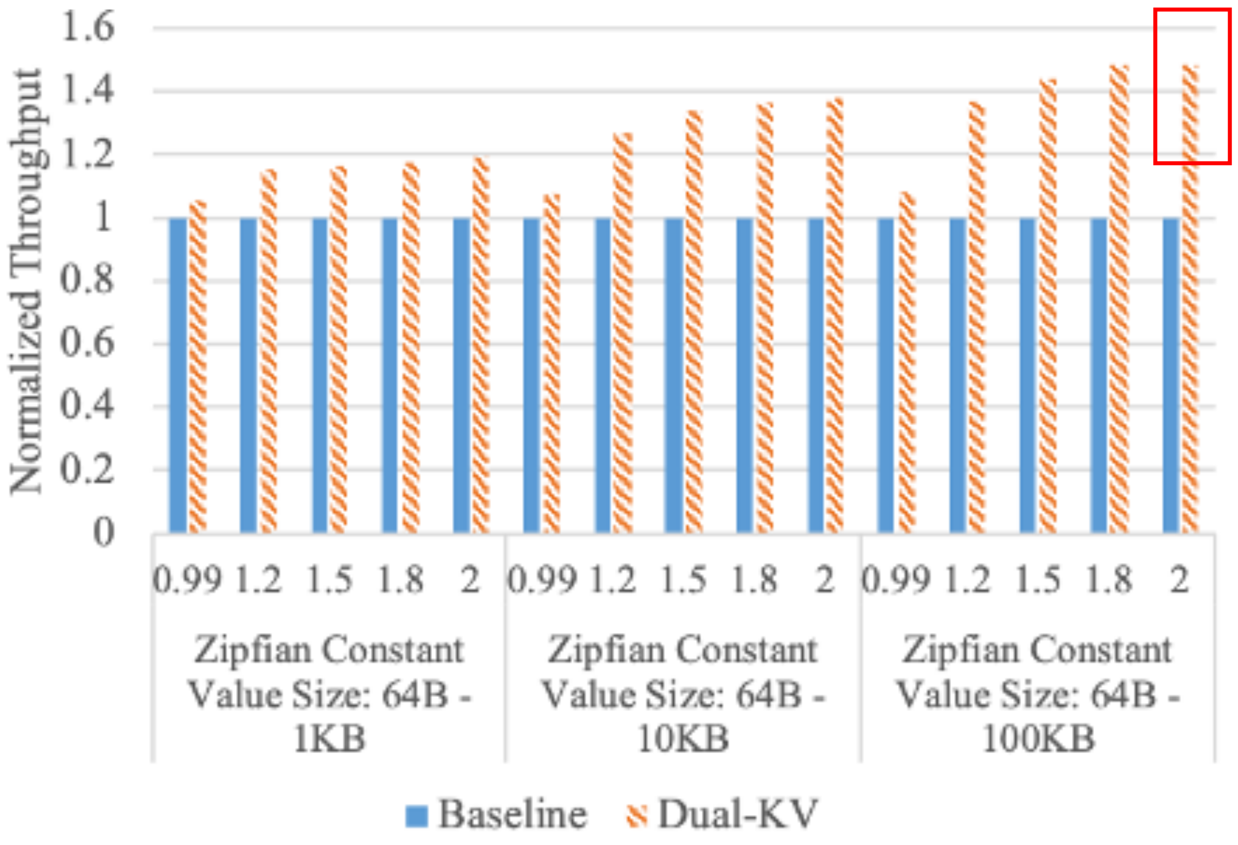
Operation Ratios

- Most writes are sped up using fast writes



Workload Skewness

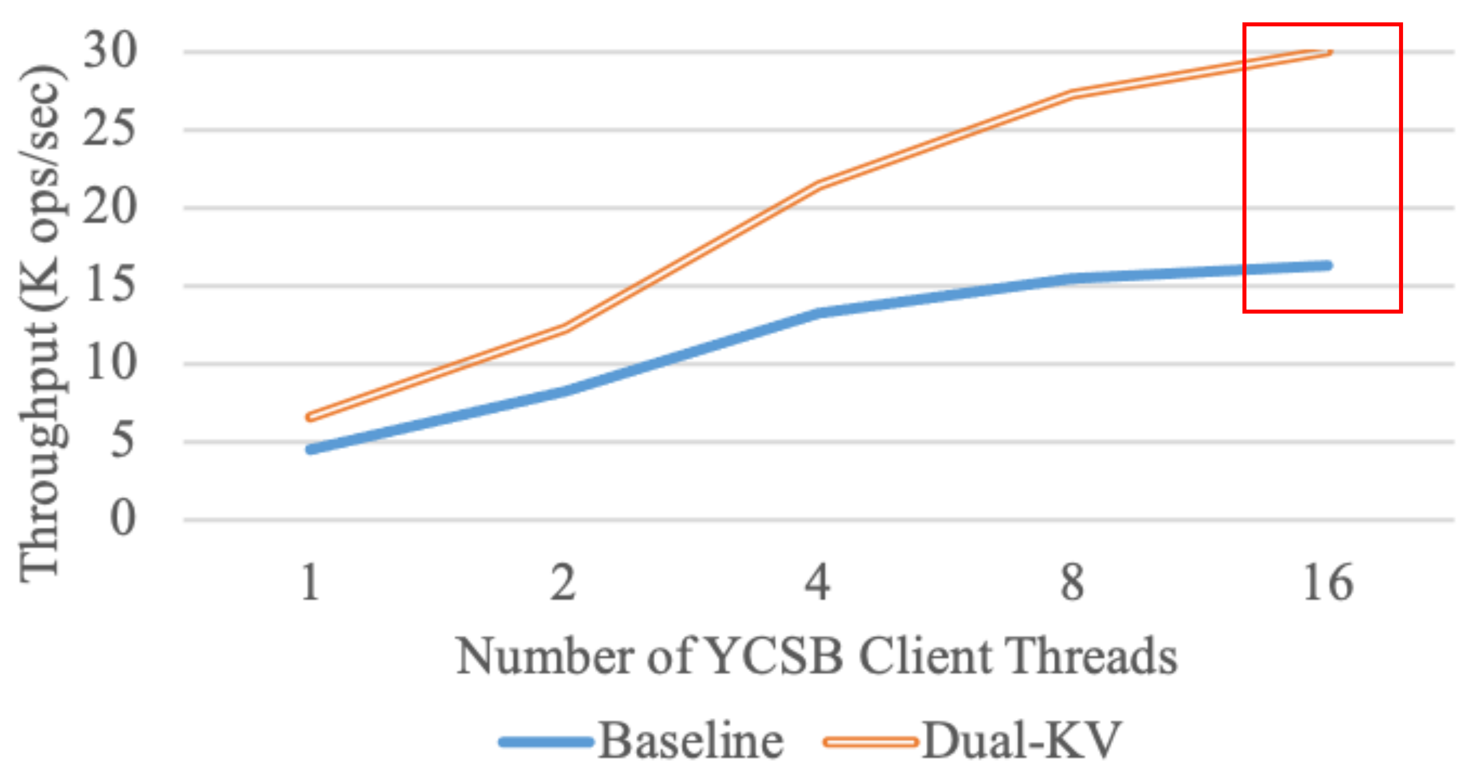
- Workloads A1, A2, A3 -> different access skewness
- Dual-KV improves up to 49% of throughput under a highly skewed workload



Scalability (Clients)

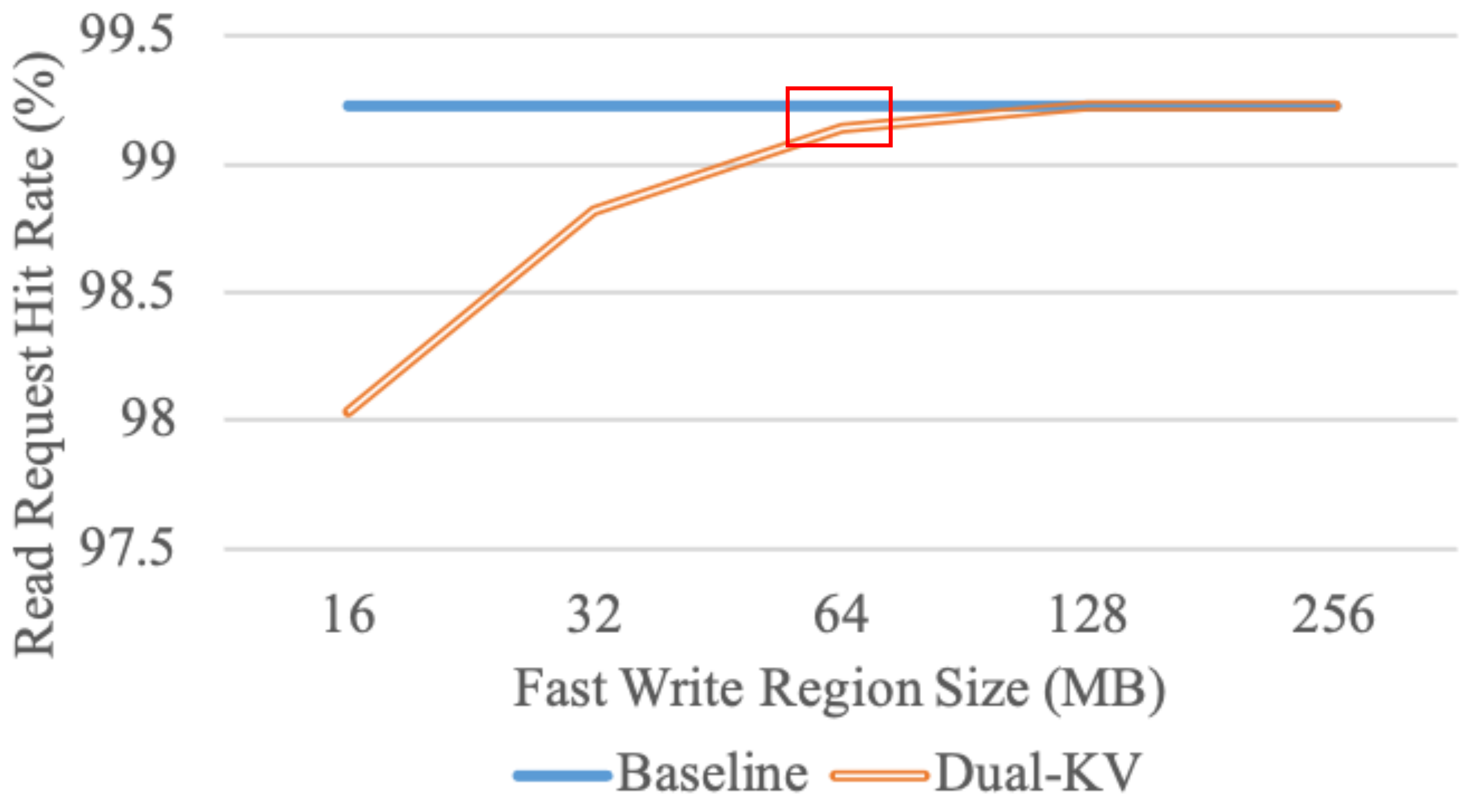
- Workload A3 (read 50%, update 50%, 64B ~ 100KB)
- 4 Memcached worker threads

83 % improvement



Fast Write Region Size vs. Hit Rate

- Larger fast write region ->
  - higher hit rate (less dropped live items)
  - **but with more hardware cost (larger supercapacitor)**



Thank You

Q & A

國立成功大學
National Cheng Kung University
1931

Operating System and Embedded System Lab