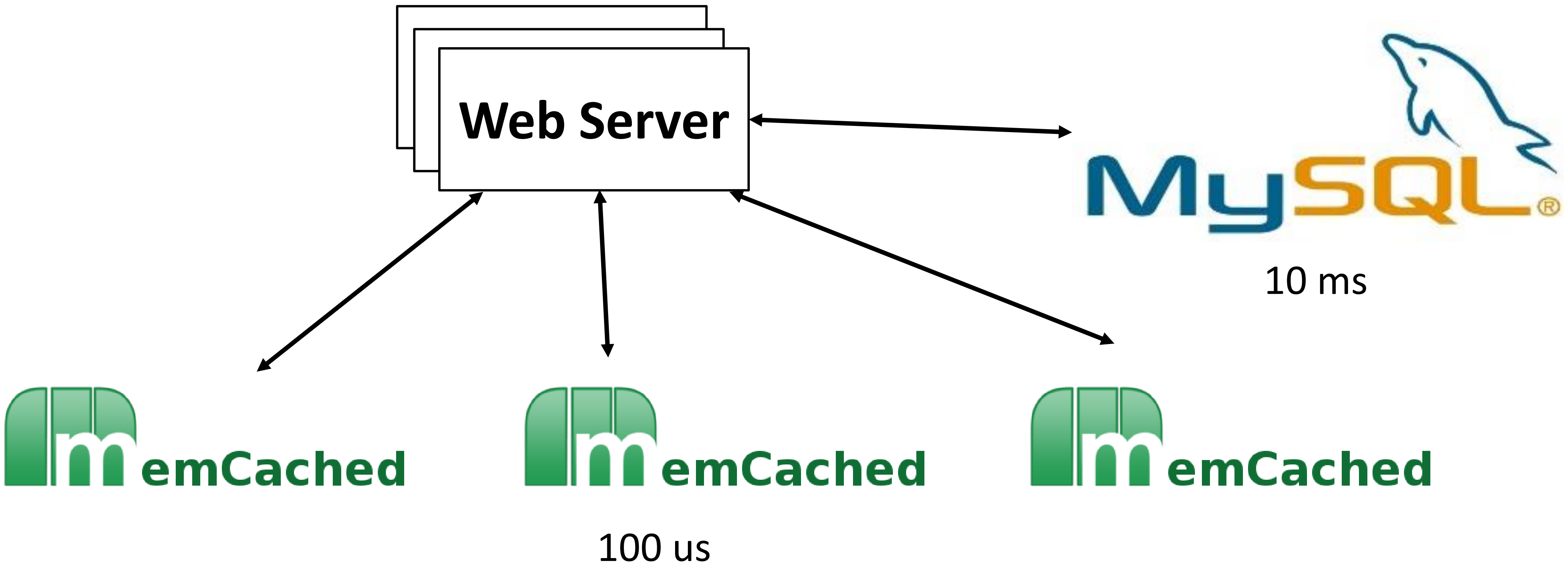
Memshare: a Dynamic Multi-tenant Key-value Cache

ASAF CIDON*, DANIEL RUSHTON†, STEPHEN M. RUMBLE‡, RYAN STUTSMAN†

*STANFORD UNIVERSITY, †UNIVERSITY OF UTAH, ‡GOOGLE INC.

Cache is 100X Faster Than Database



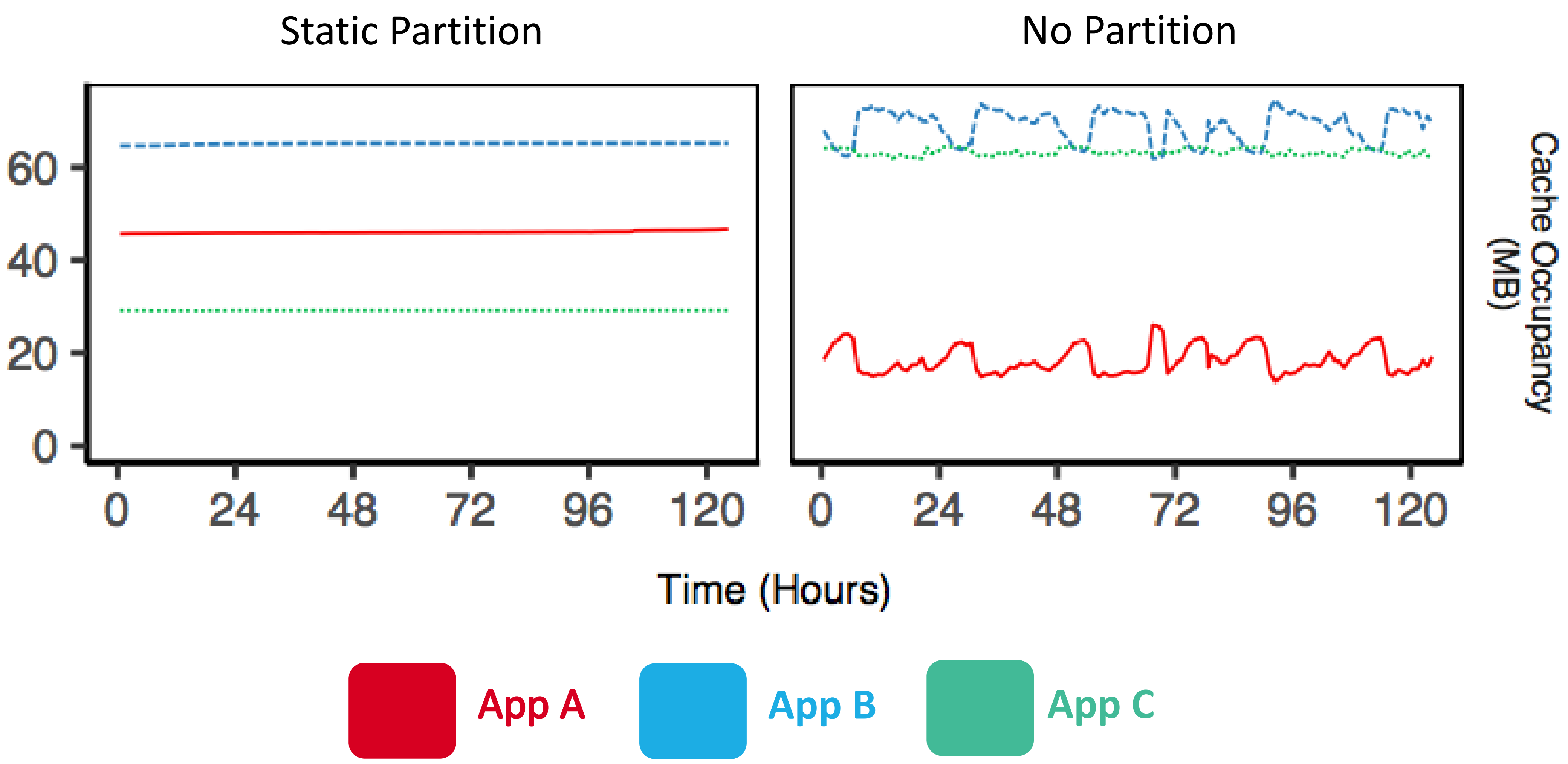
Cache Hit Rate Drives Cloud Performance

- Small improvements to cache hit rate make big difference:
- At 98% cache hit rate:
  - +1% hit rate → 35% speedup
  - Facebook study [Atikoglu '12]

Static Partitioning → Low Hit Rates

- Cache providers statically partition their memory among applications
- Examples:
  - Facebook
  - Amazon Elasticache
  - Memcachier

Partitioned Memory Over Time



Partitioned vs No Partition Hit Rates

Application	Hit Rate Partitioned	Hit Rate No Partition
Combined	**87.8%**	**88.8%**
A	97.6%	96.6%
B	98.8%	99.1%
C	30.1%	39.2%

Partitioned Memory: Pros and Cons

- Disadvantages:
  - Lower hit rate due to low utilization
  - Higher TCO
- Advantages:
  - Isolated performance and predictable hit rate
  - “Fairness”: customers get what they pay for

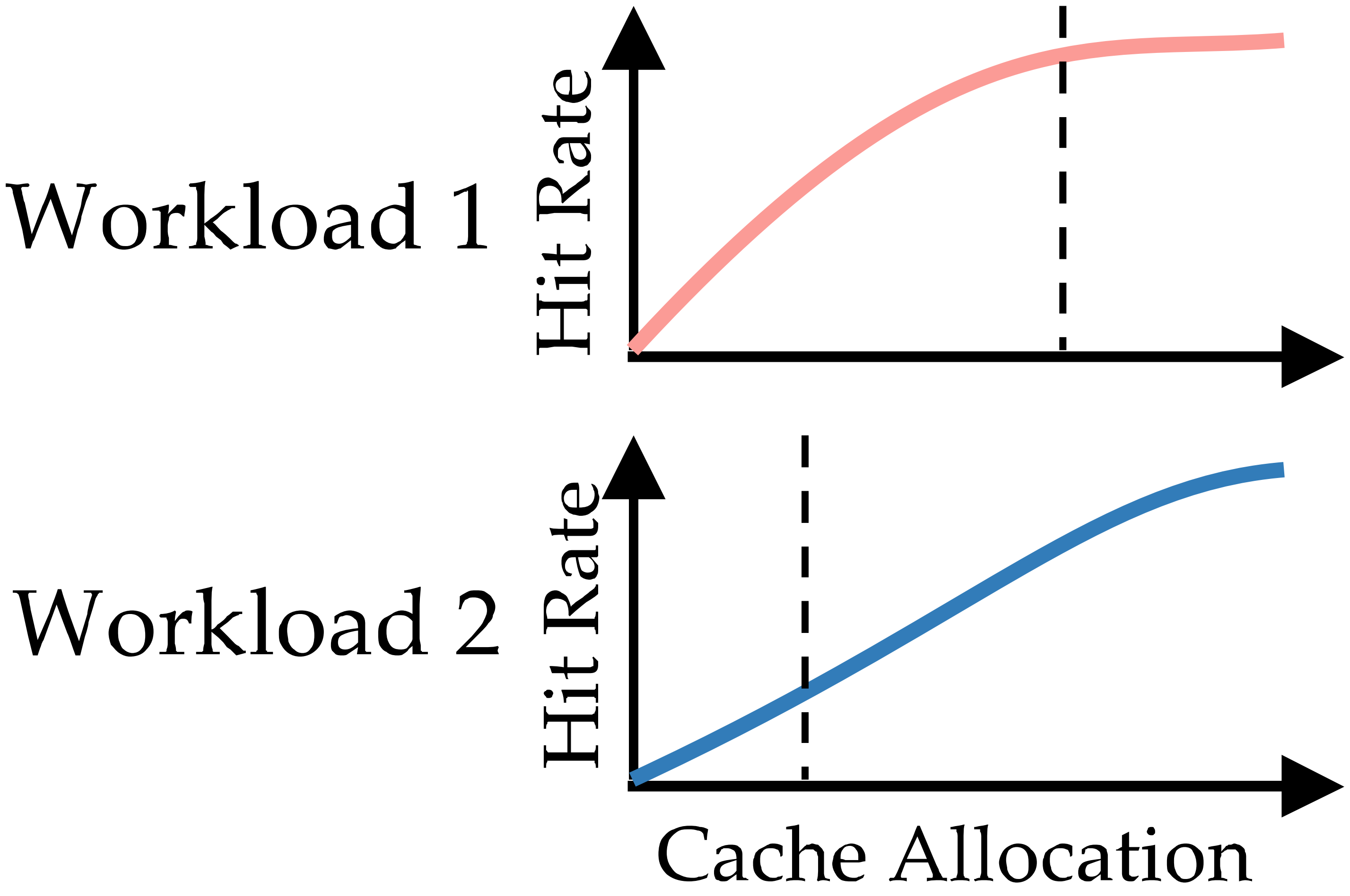
Memshare: the Best of Both Worlds

- Optimize memory allocation to maximize overall hit rate
- While providing minimal guaranteed memory allocation and performance isolation

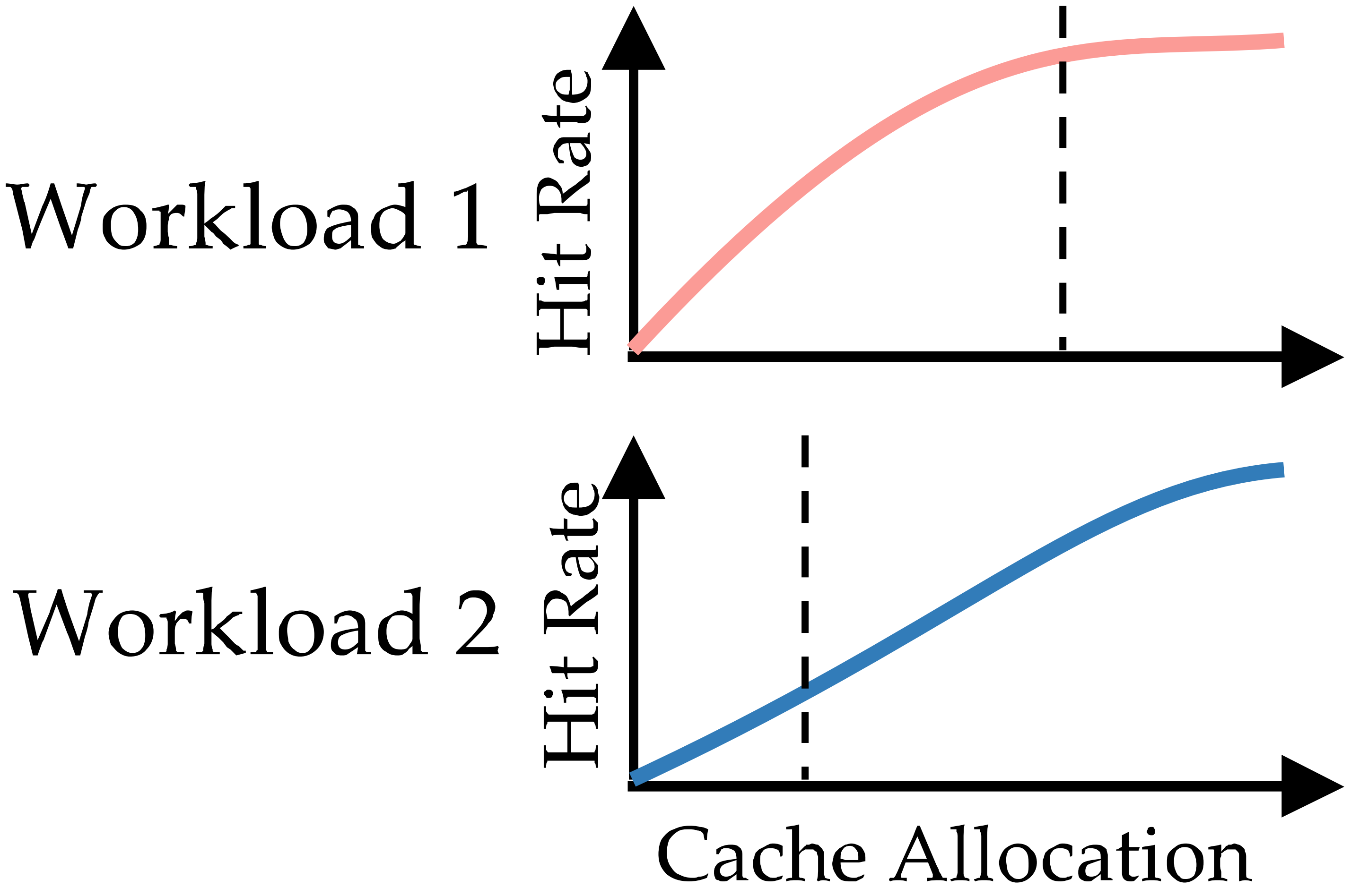
Multi-tenant Cache Design Challenges

1. **Decide application memory allocation to optimize hit rate**
2. Enforce memory allocation among applications

Estimate Hit Rate Curve Gradient to Optimize Hit Rate



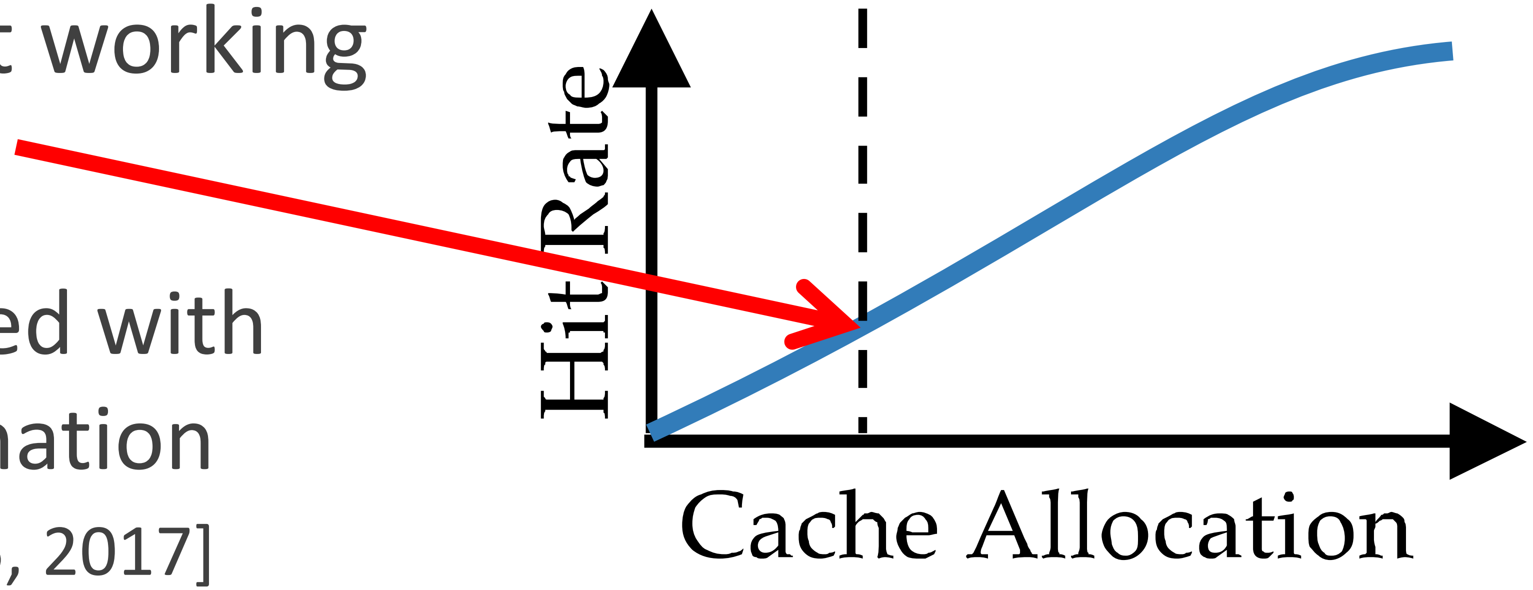
Estimate Hit Rate Curve Gradient to Optimize Hit Rate



$\nabla w_1 < \nabla w_2 \rightarrow$ Keep items from w_2

Estimating Hit Rate Gradient

- Track access frequency to recently evicted objects to determine gradient at working point
- Can be further improved with full hit rate curve estimation
  - SHARDS [Waldspurger 2015, 2017]
  - AET [Hu 2016]



Multi-tenant Cache Design Challenges

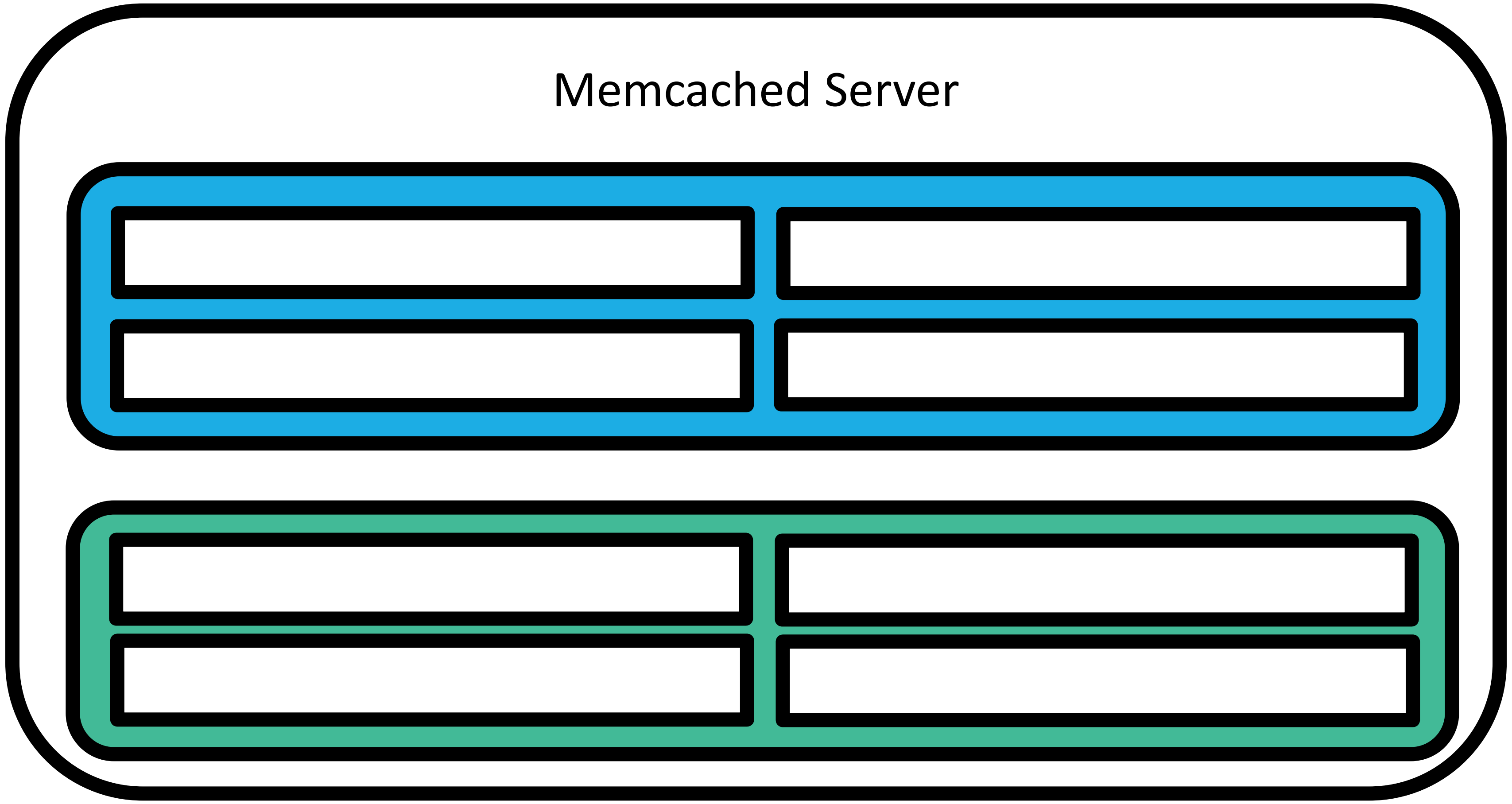
1. Decide application memory allocation to optimize hit rate
2. **Enforce memory allocation among applications**

Multi-tenant Cache Design Challenges

1. Decide application memory allocation to optimize hit rate
2. **Enforce memory allocation among applications**

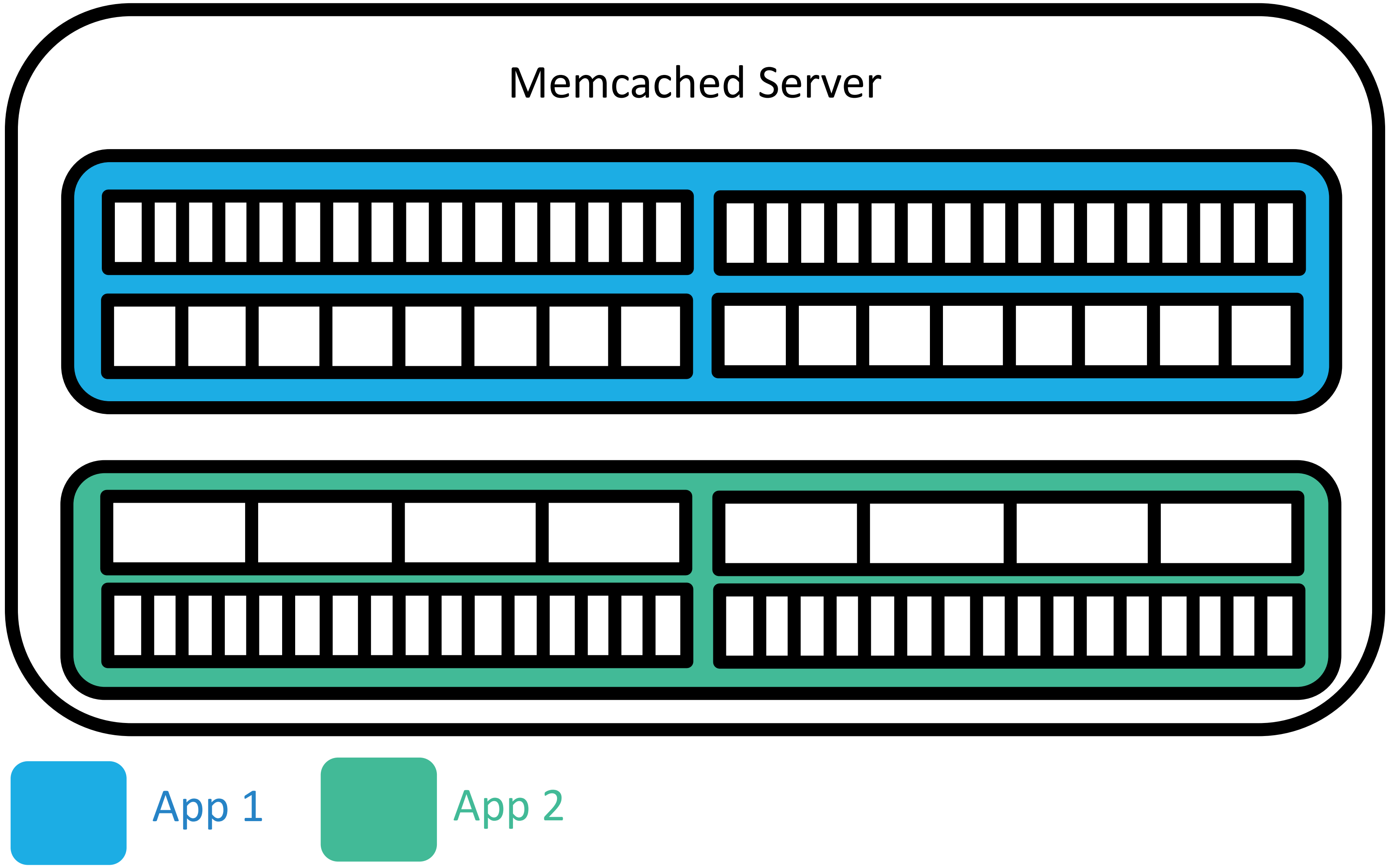
Not so simple

Slab Allocation Primer

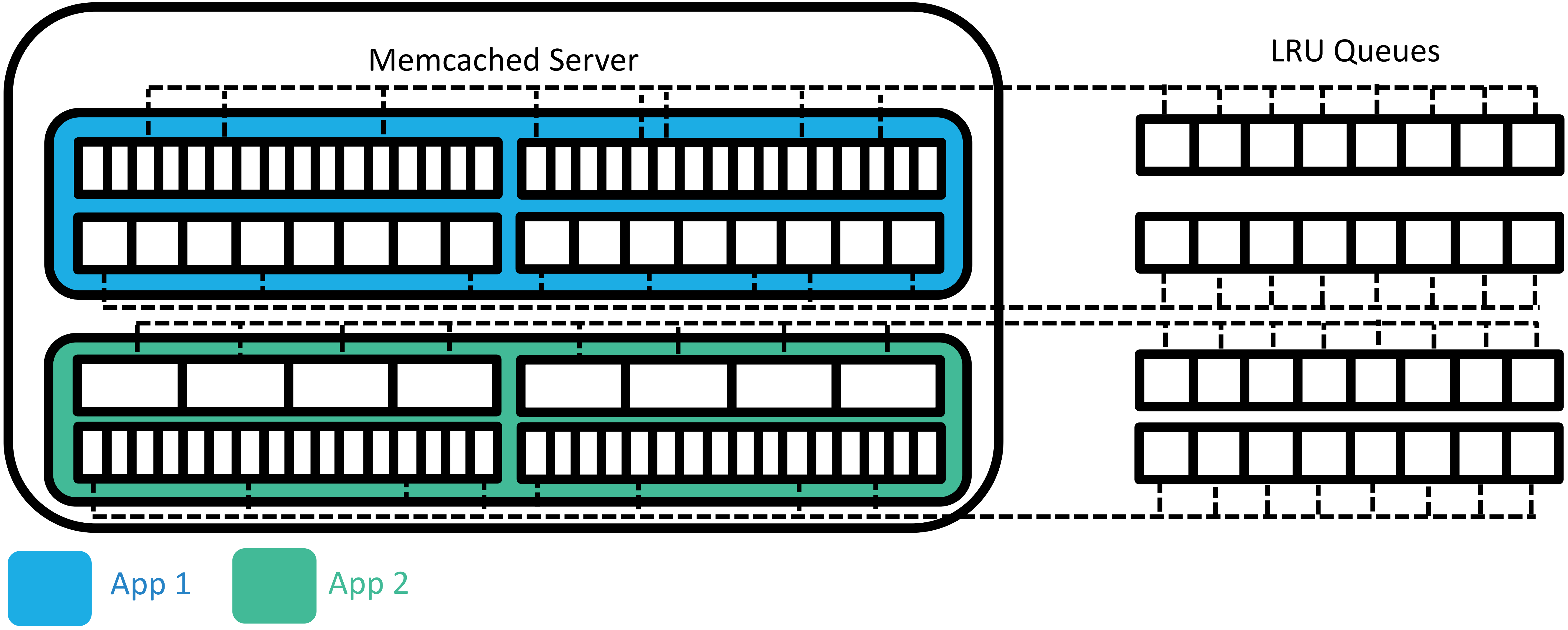


App 1 App 2

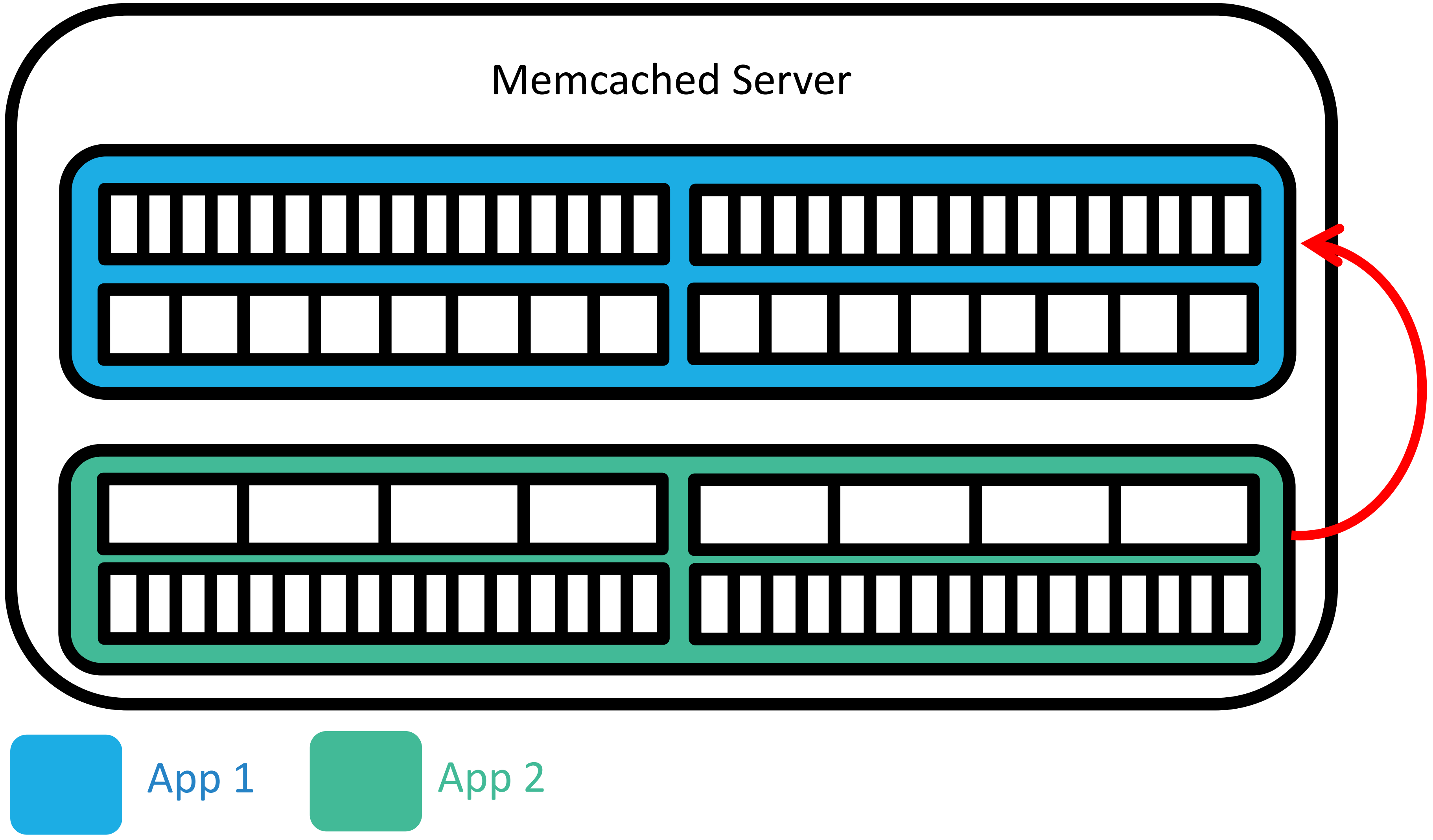
Slab Allocation Primer



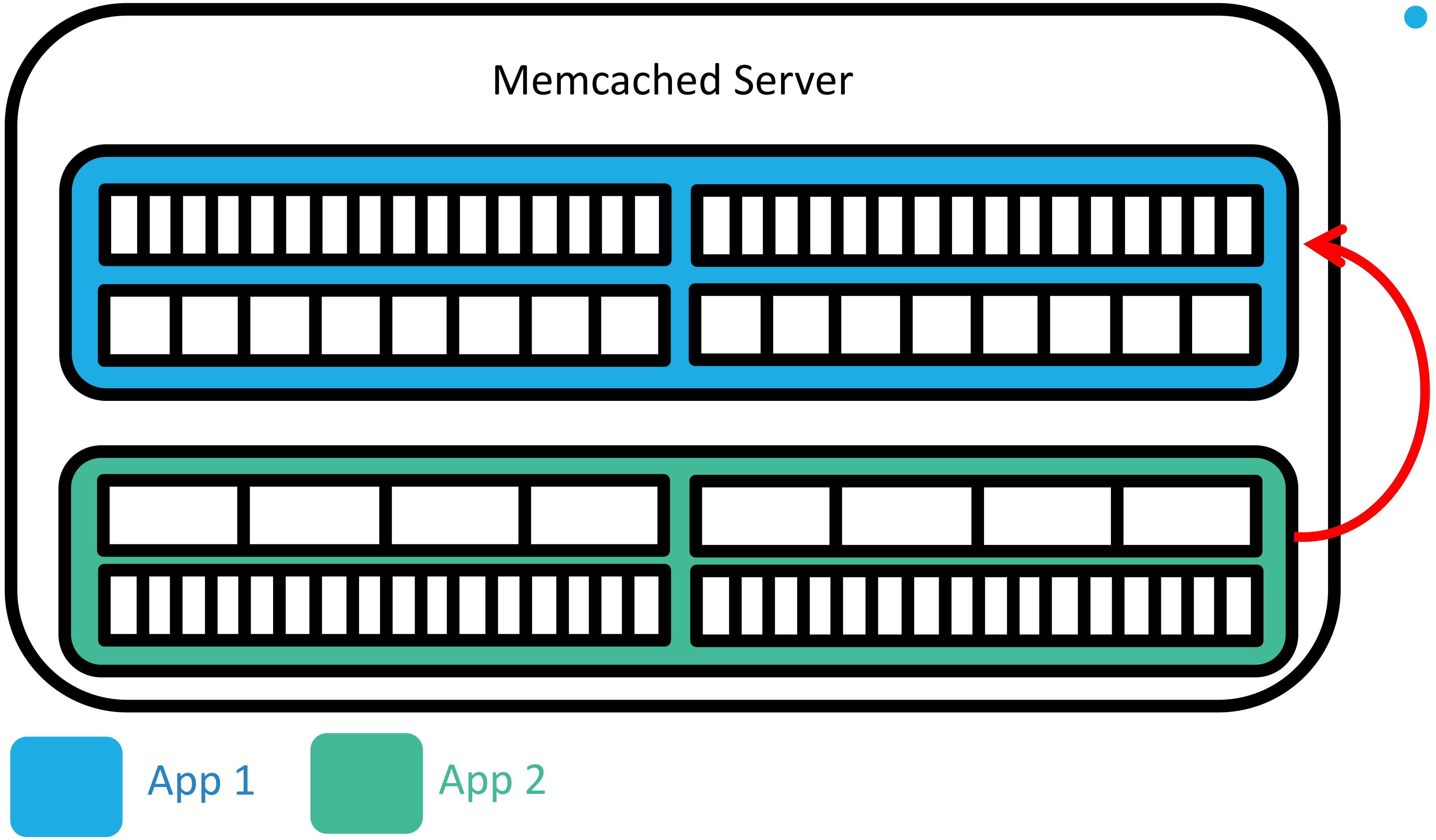
Slab Allocation Primer



Goal: Move 4KB from App 2 to App 1

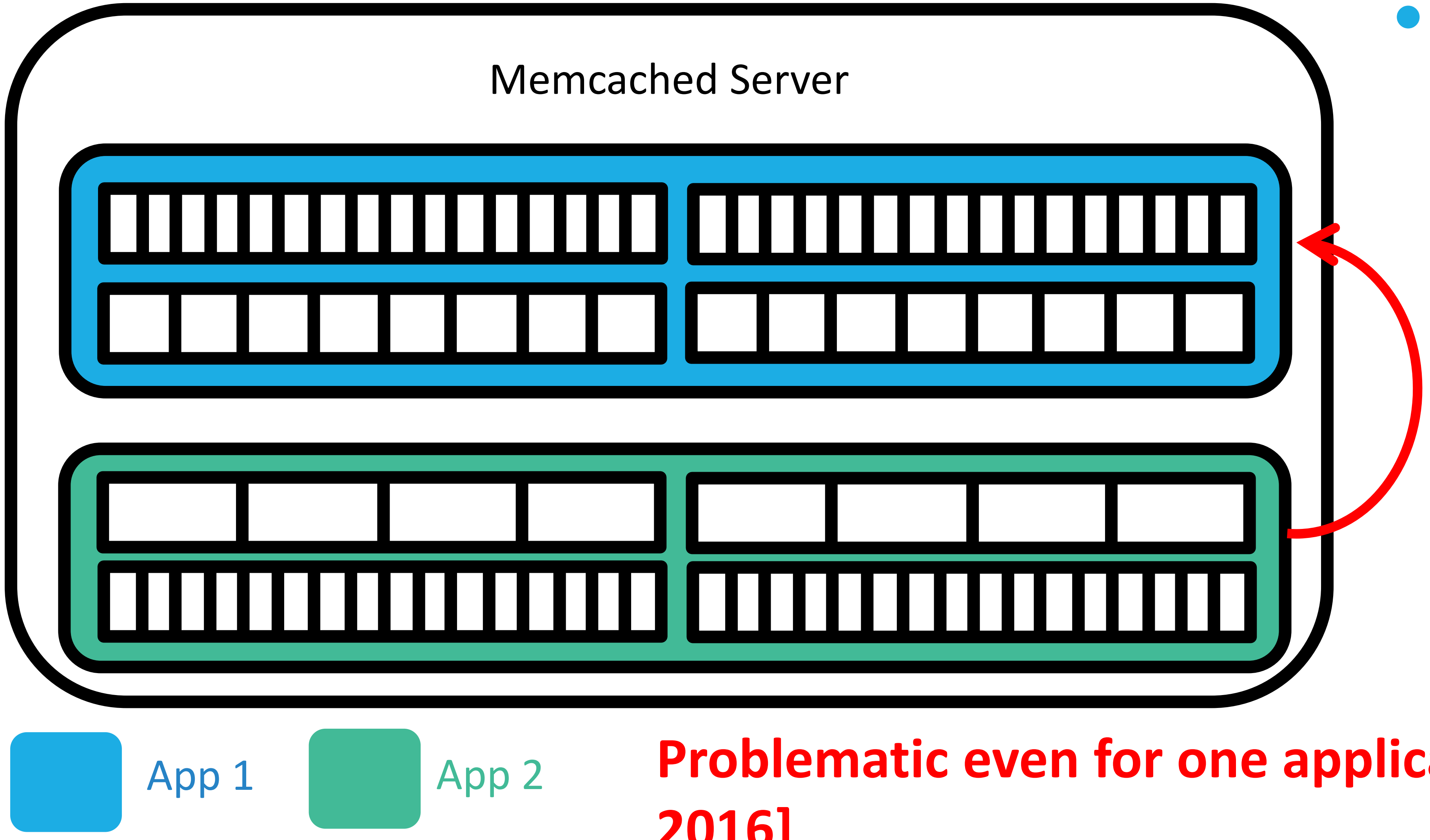


Goal: Move 4KB from App 2 to App 1



- Problems:
  - Need to evict 1MB
    - Contains many small objects, some are hot
  - App 1 can only use extra space for objects of certain size

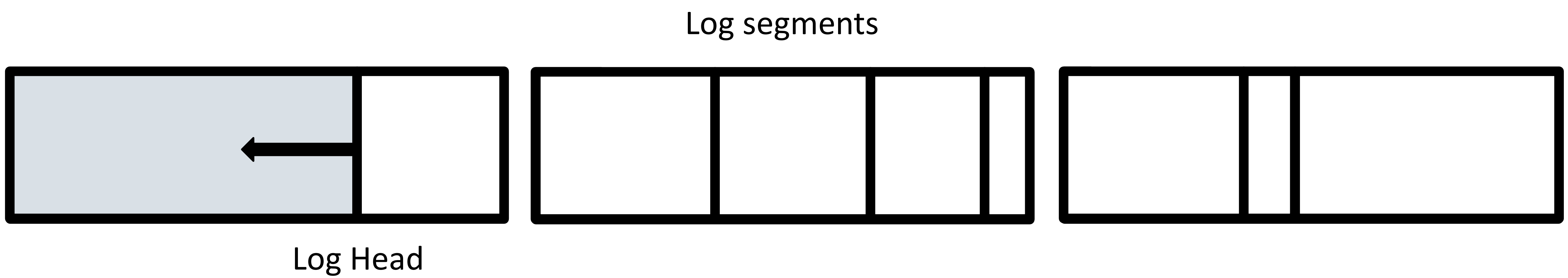
Goal: Move 4KB from App 2 to App 1



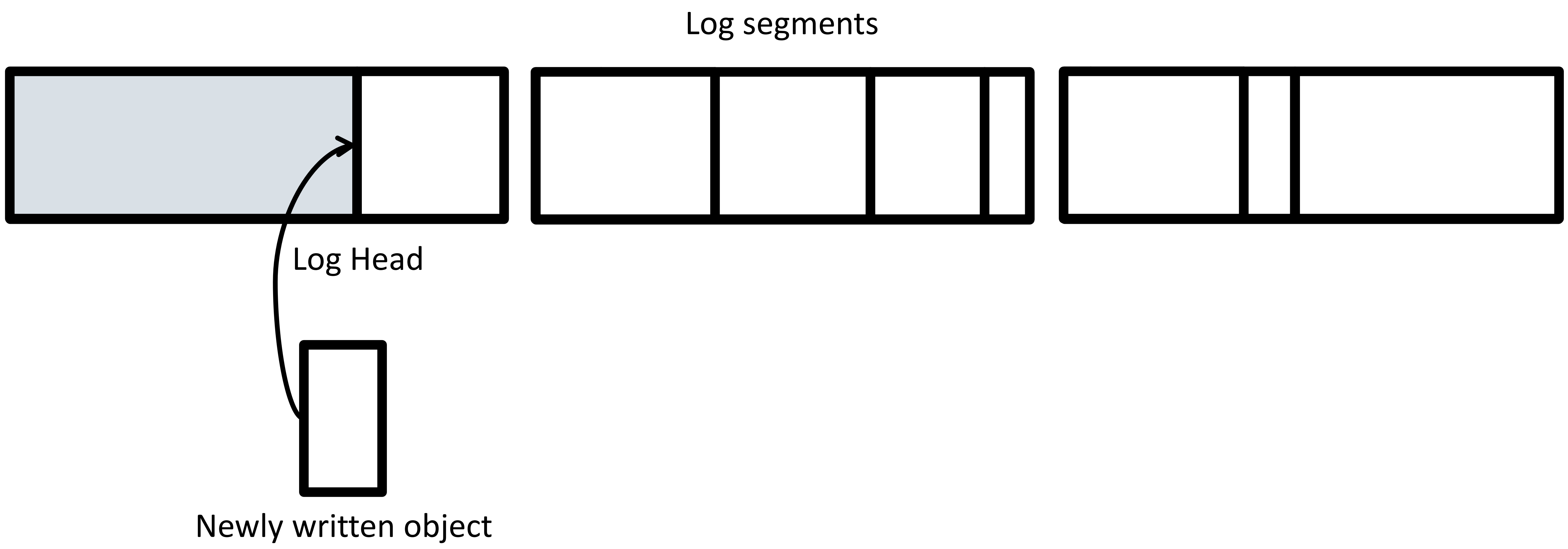
- Problems:
  - Need to evict 1MB
    - Contains many small objects, some are hot
  - App 1 can only use extra space for objects of certain size

Problematic even for one application, see Cliffhanger [Cidon 2016]

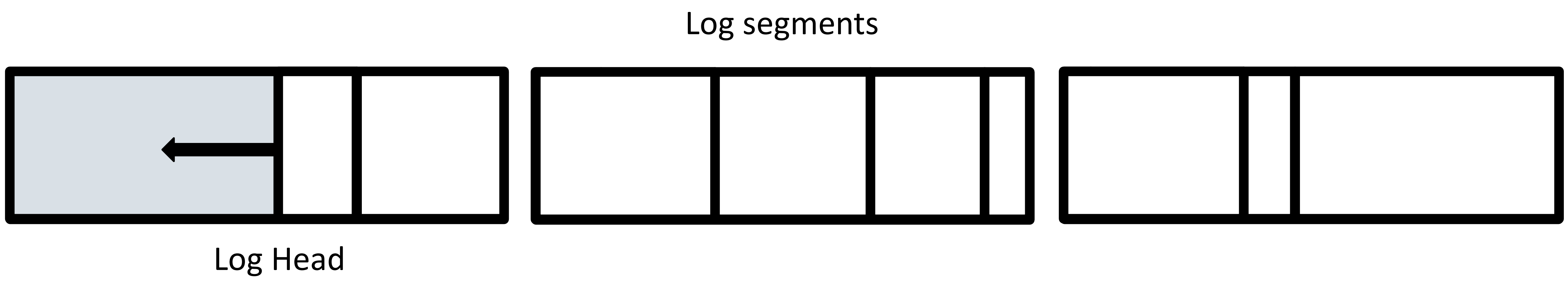
Instead of Slabs: Log-structured Memory



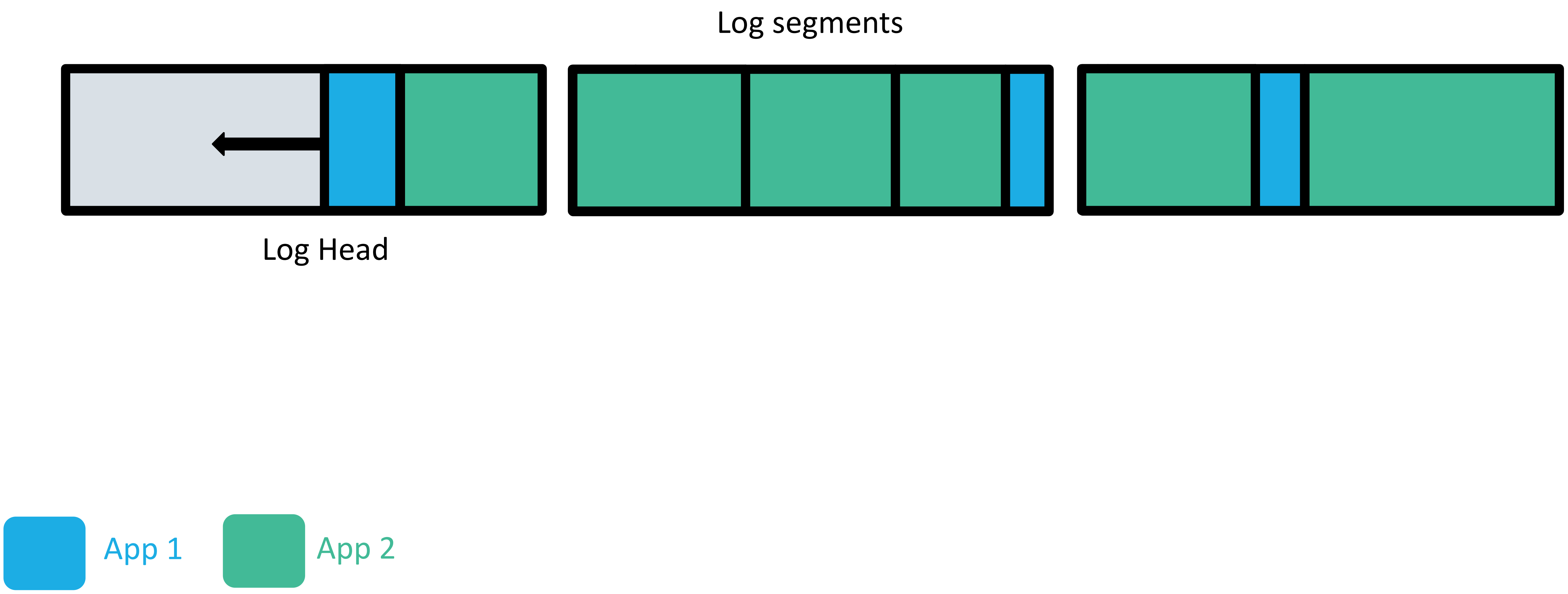
Instead of Slabs: Log-structured Memory



Instead of Slabs: Log-structured Memory



Applications are Physically Intermixed



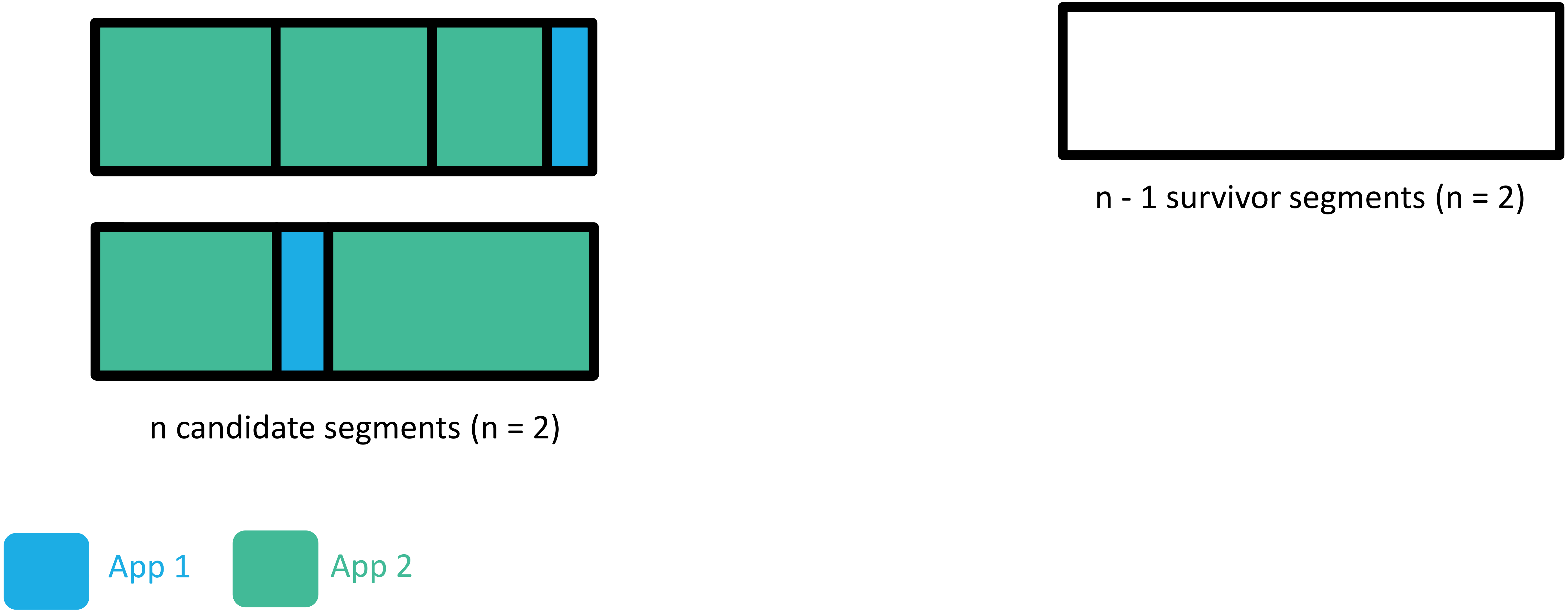
Memshare's Sharing Model

- Reserved Memory: guaranteed static memory
- Pooled Memory: application's share of pooled memory
- Target Memory = Reserved Memory + Pooled Memory

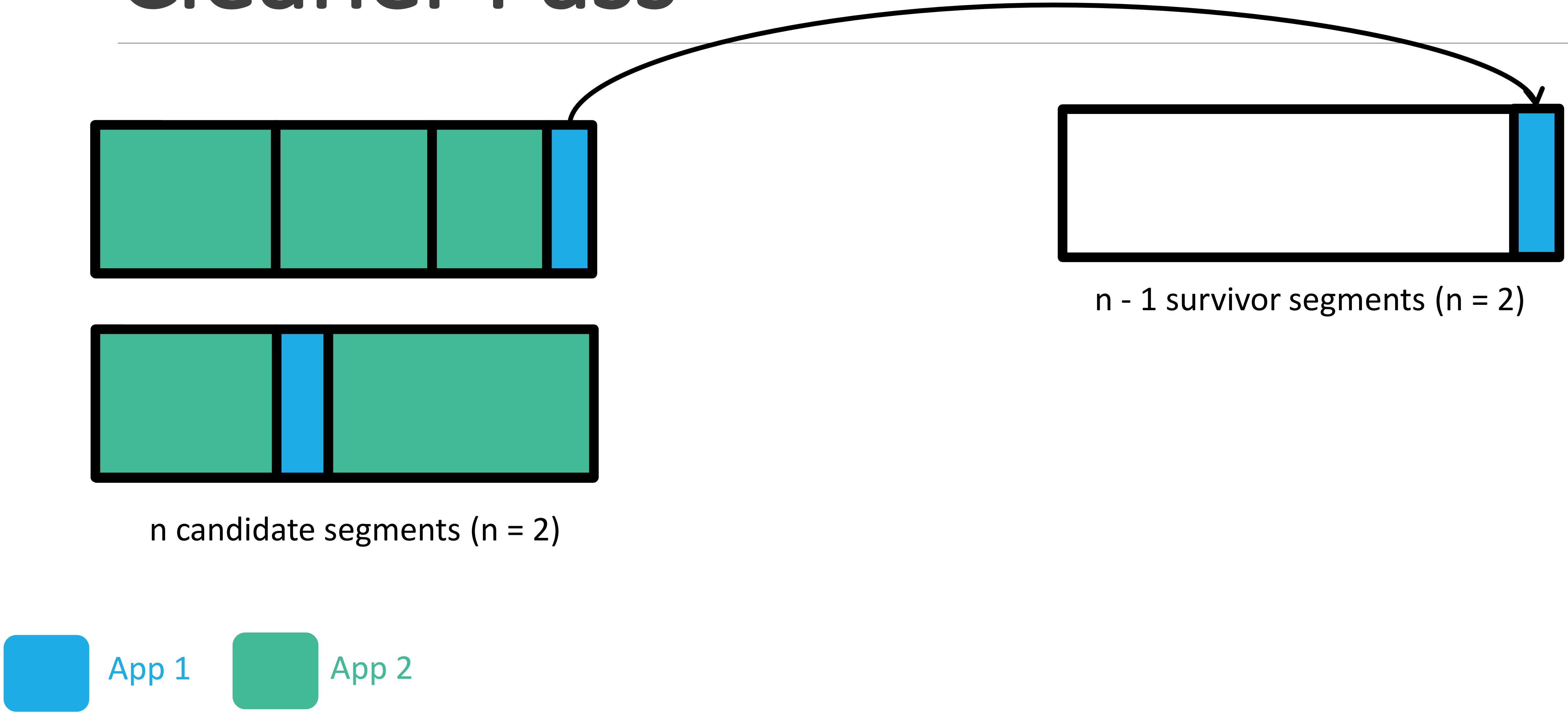
Cleaning Priority Determines Eviction Priority

- Q: When does Memshare evict?
- A: Newly written objects evict old objects, but not in critical path
  - Cleaner keeps 1% of cache empty
  - Cleaner tries to enforce actual memory allocation to be equal to Target Memory

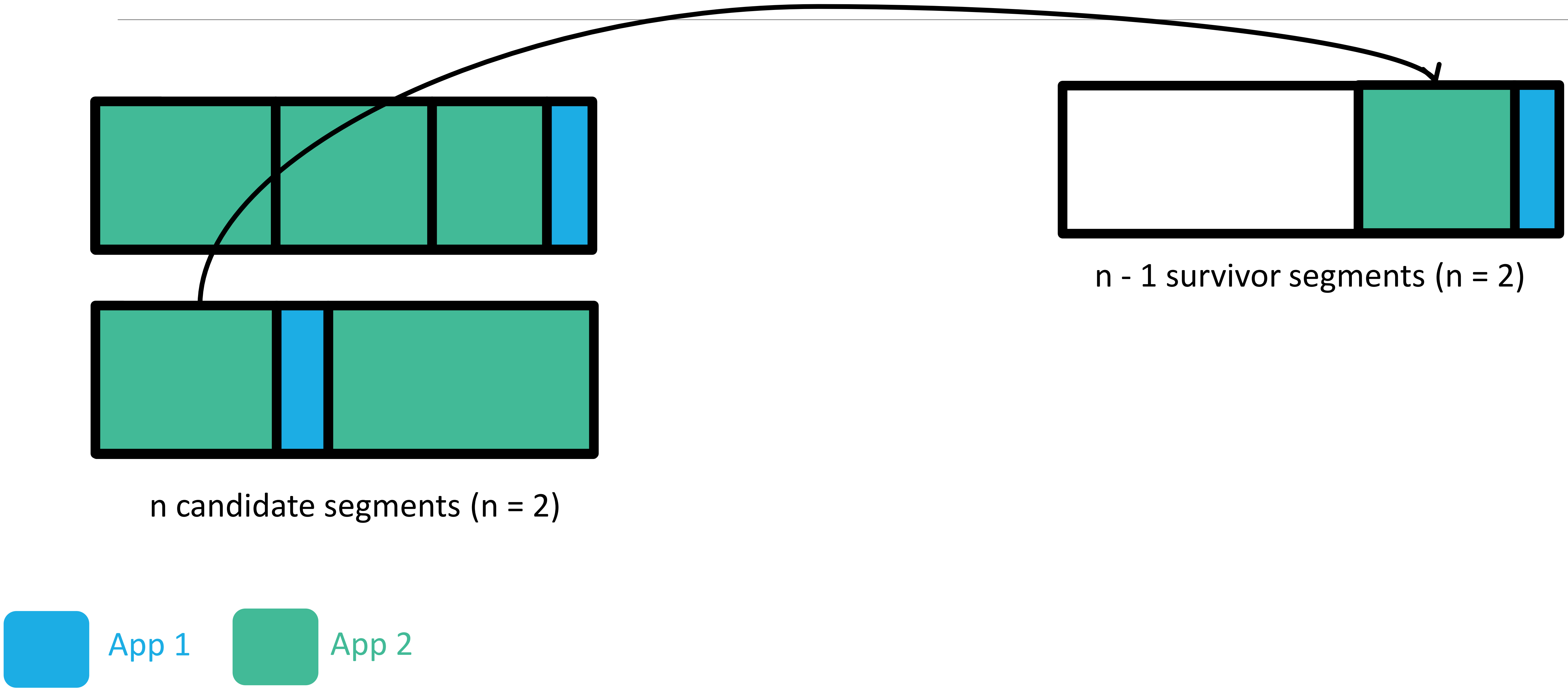
Cleaner Pass



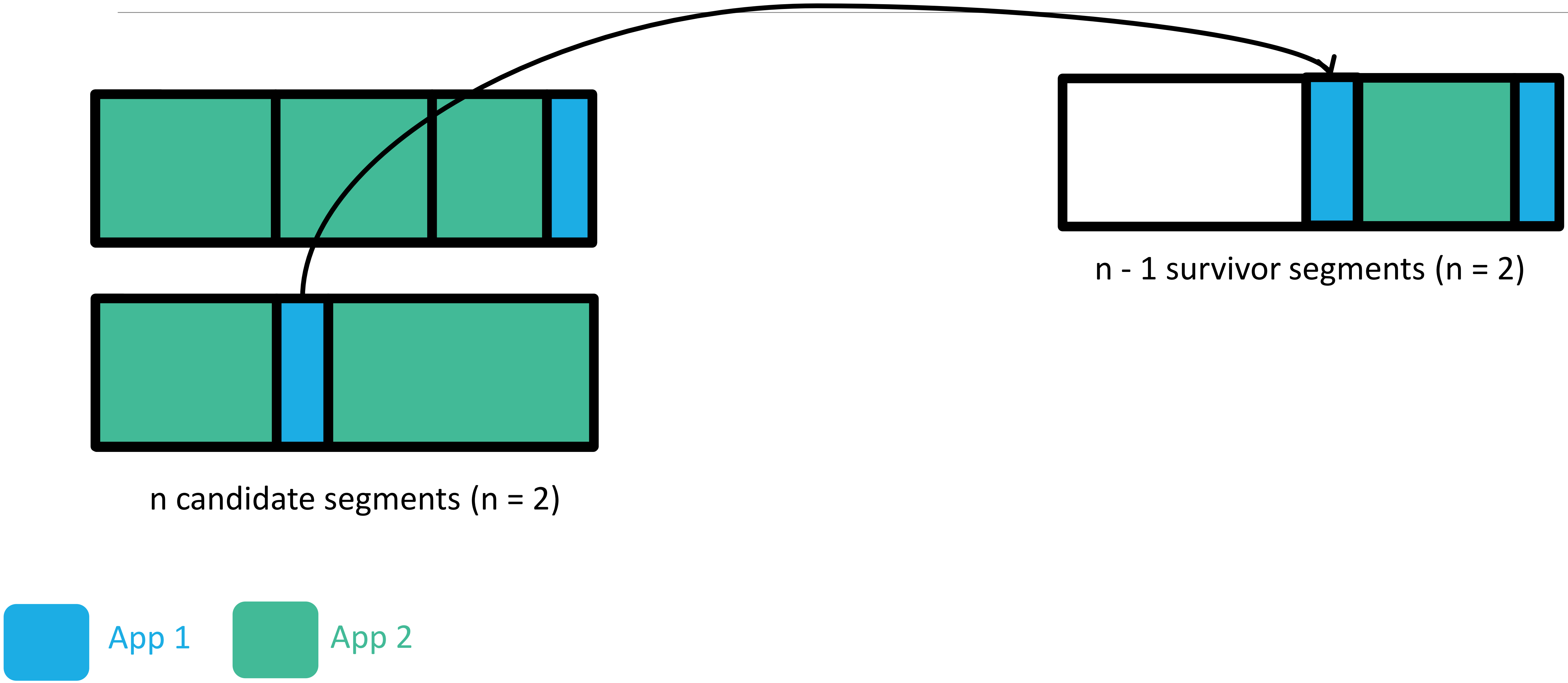
Cleaner Pass



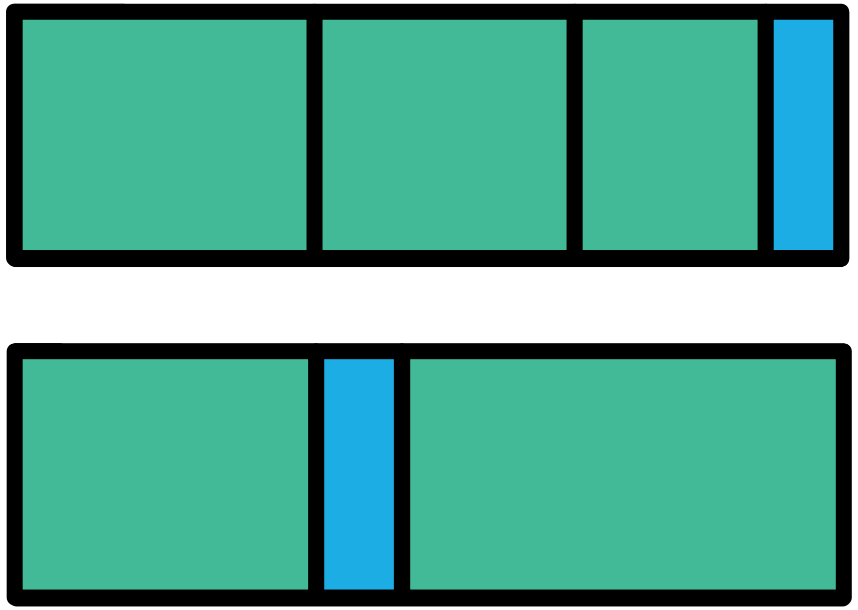
Cleaner Pass



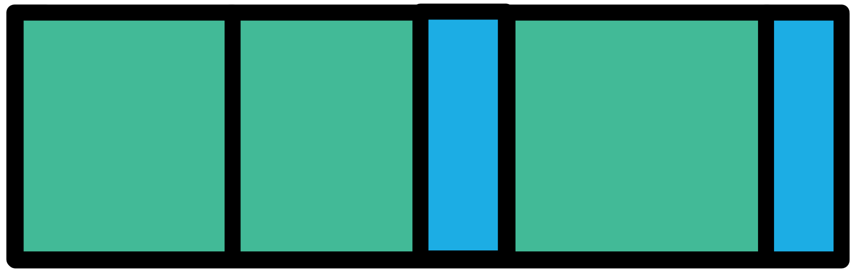
Cleaner Pass



Cleaner Pass



n candidate segments (n = 2)

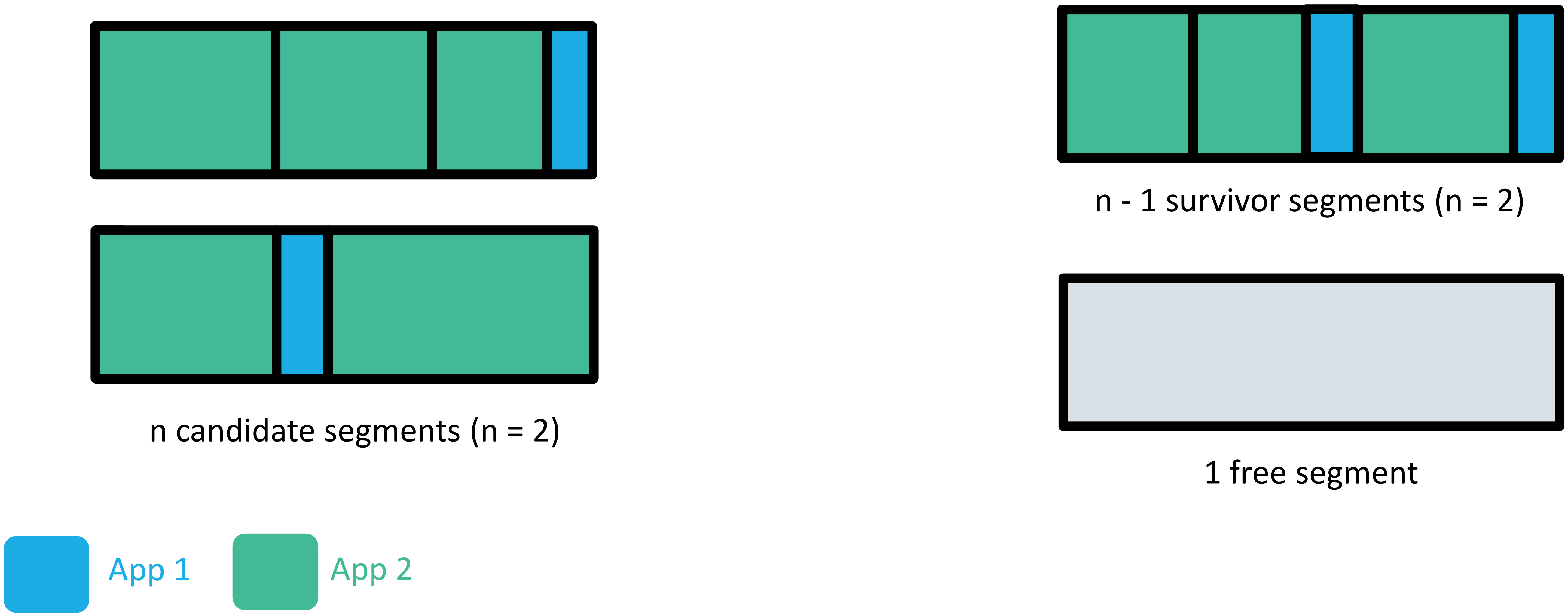


n - 1 survivor segments (n = 2)

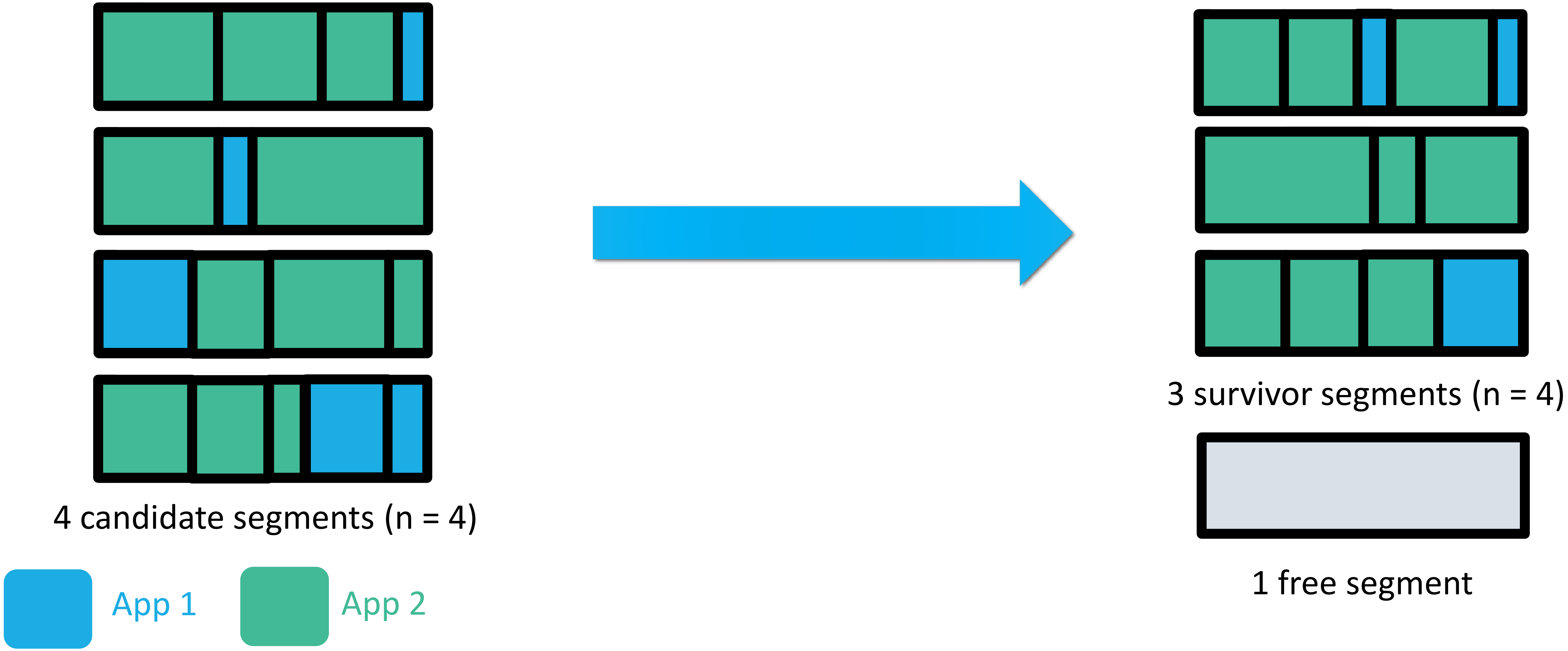
App 1

App 2

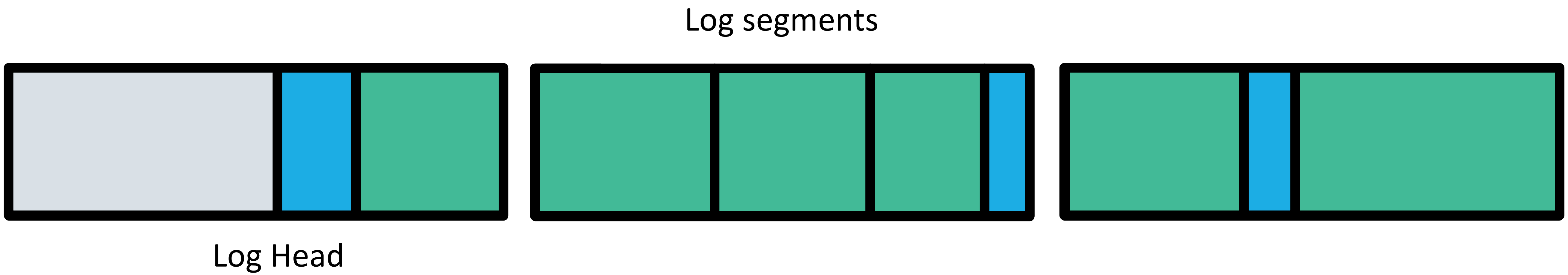
Cleaner Pass



Cleaner Pass (n = 4): Twice the Work



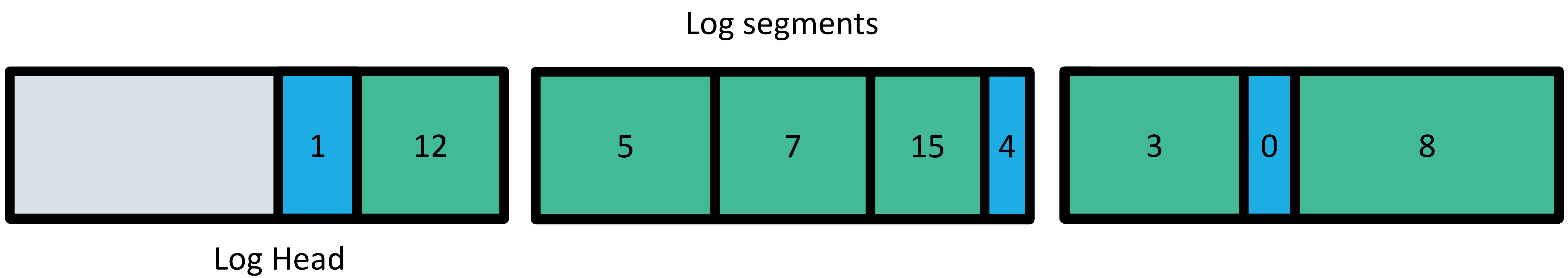
Application Need: How Far is Memory Allocation from Target Memory?



$$need(app) = \frac{targetMemory(app)}{actualMemory(app)}$$

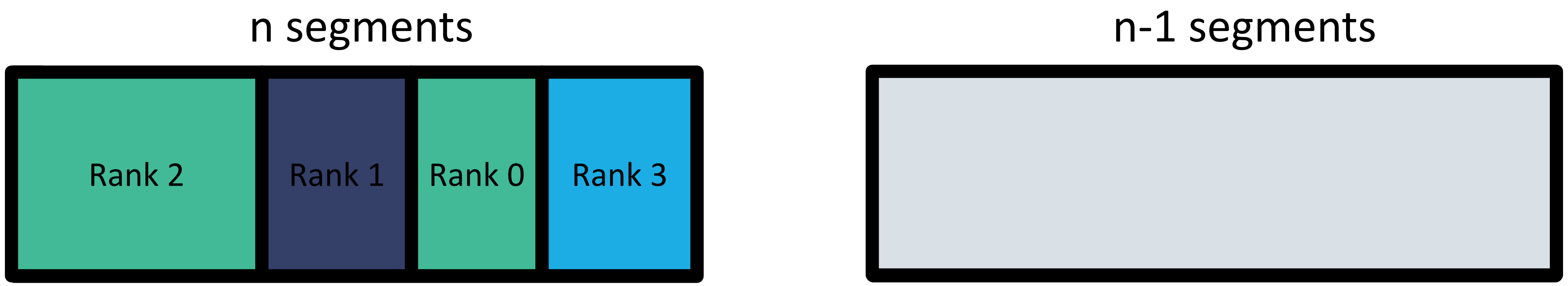
App 1 App 2

Within Each Application, Evict by Rank



- To implement LRU: rank = last access time

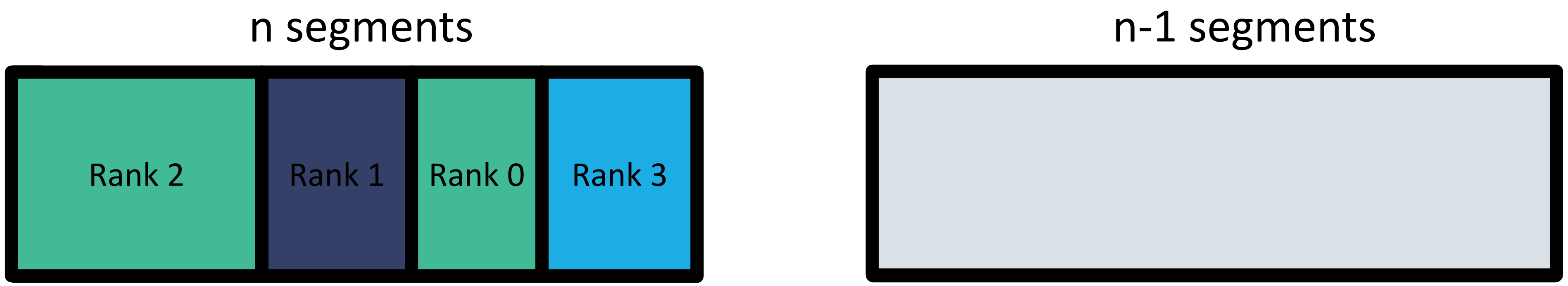
Cleaning: Max Need and then Max Rank



Max Need?
Max Rank?

Need	
App 1	0.8
App 2	1.4
App 3	0.9

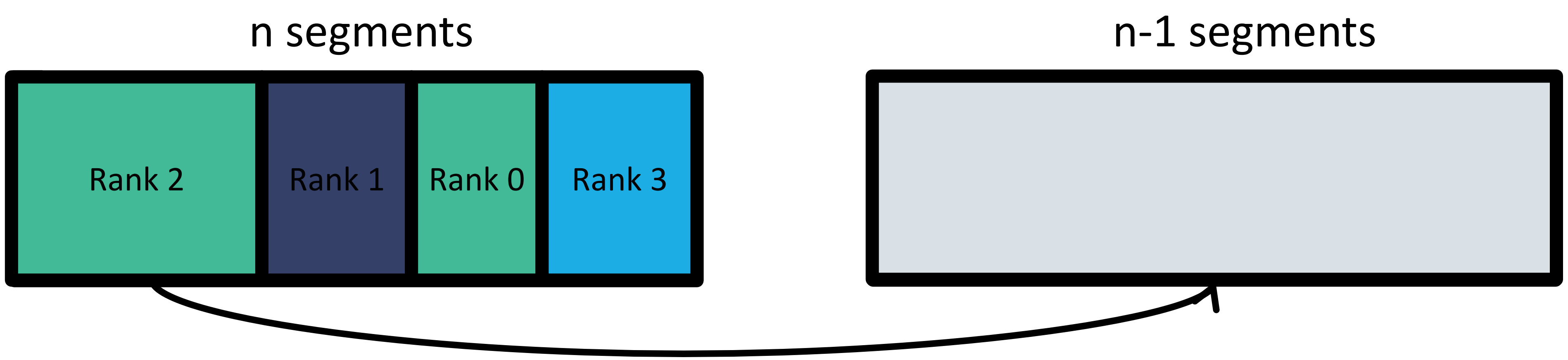
Cleaning: Max Need and then Max Rank



Max Need? → App 2
Max Rank?

Need	
App 1	0.8
App 2	1.4
App 3	0.9

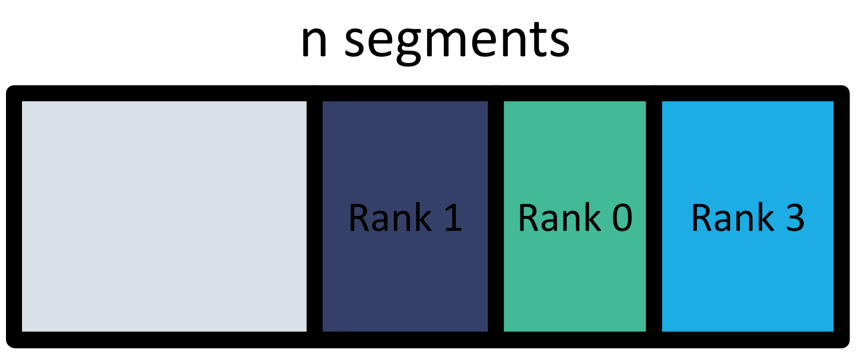
Cleaning: Max Need and then Max Rank

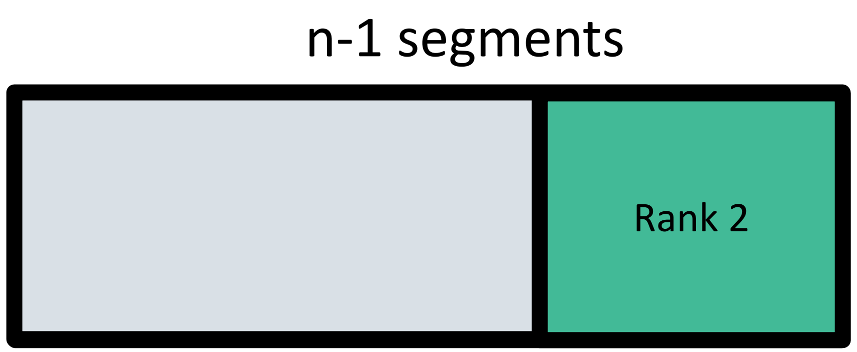


Max Need? → App 2
Max Rank? → Rank 2

Need	
App 1	0.8
App 2	1.4
App 3	0.9

Cleaning: Max Need and then Max Rank

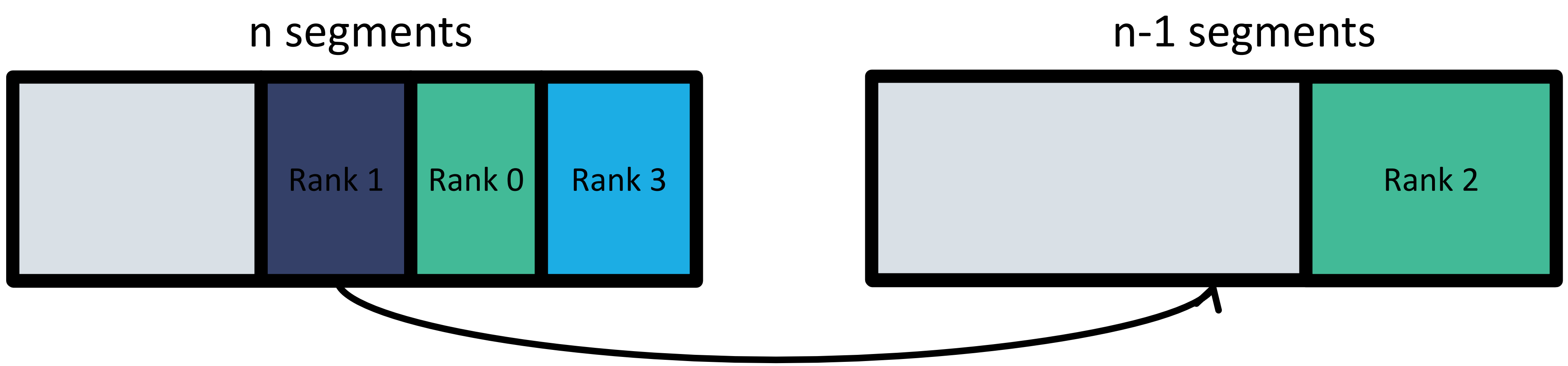




Max Need?
Max Rank?

	Need
App 1	0.9
App 2	0.8
App 3	1.2

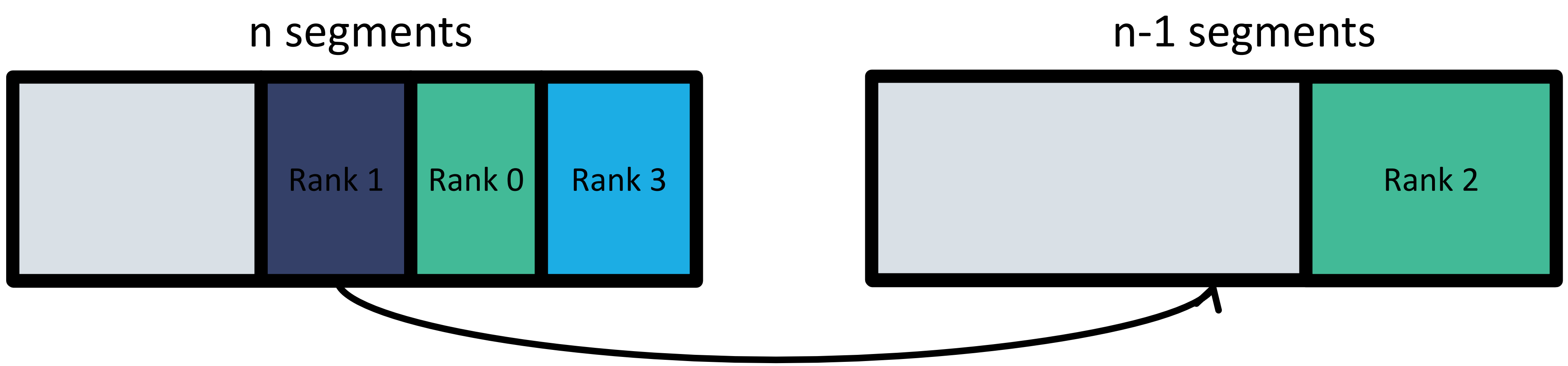
Cleaning: Max Need and then Max Rank



Max Need? → App 3
Max Rank?

Need	
App 1	0.9
App 2	0.8
App 3	1.2

Cleaning: Max Need and then Max Rank



Max Need? → App 3
Max Rank? → Rank 1

Need	
App 1	0.9
App 2	0.8
App 3	1.2

Trading Off Eviction Accuracy and Cleaning Cost

- Eviction accuracy is determined by n
  - For example: rank = time of last access
  - When n → # segments: ideal LRU
  - Intuition: n is similar to cache associativity
- CPU consumption is determined by n

Trading Off Eviction Accuracy and Cleaning Cost

- Eviction accuracy is determined by n
  - For
  - Wh
  - Intu ity
- CPU by n

> "In practice Memcached is never CPU-bound in our data centers. Increasing CPU to improve the hit rate would be a good trade off."
>
> - Nathan Bronson, Facebook

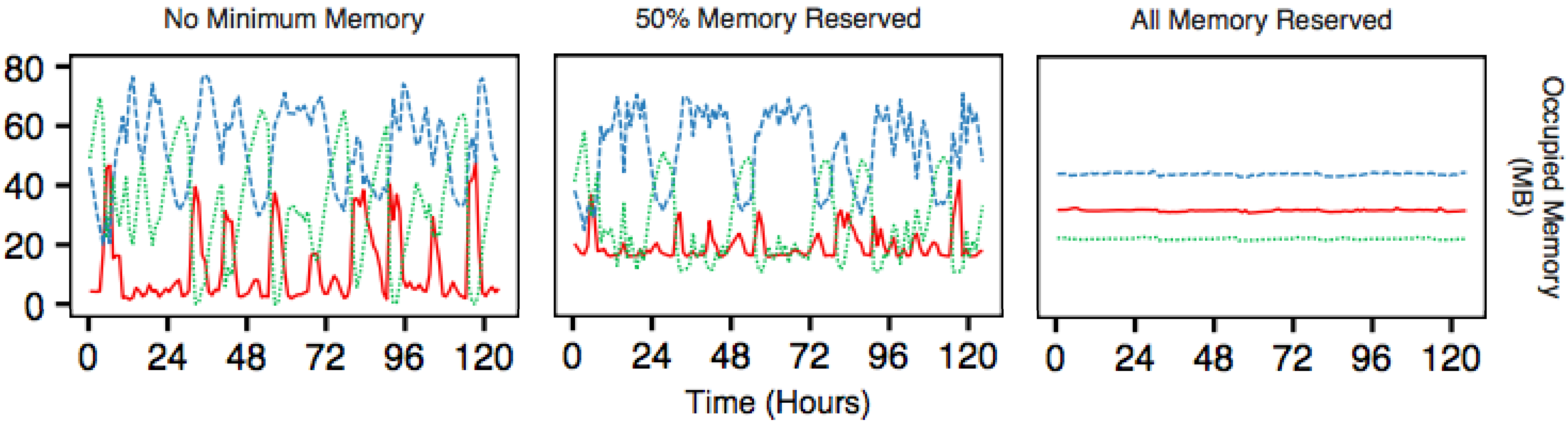
Implementation

- Implemented in C++ on top of Memcached
- Reuse Memcached's hash table, transport, request processing
- Implemented log-structured memory allocator

Partitioned vs. Memshare

Application	Hit Rate Partitioned	Hit Rate Memshare (50% Reserved)
Combined	**87.8%**	**89.2%**
A	97.6%	99.4%
B	98.8%	98.8%
C	30.1%	34.5%

Reserved vs. Pooled Behavior

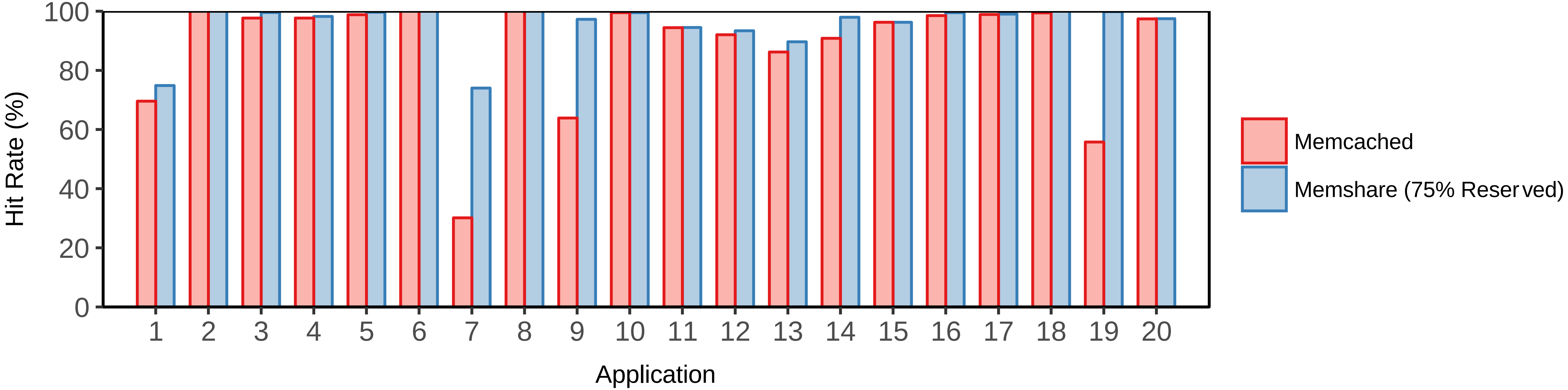


Combined Hit Rates

90.2%	89.2%	88.8%

App A App B App C

State-of-the-art Hit rate

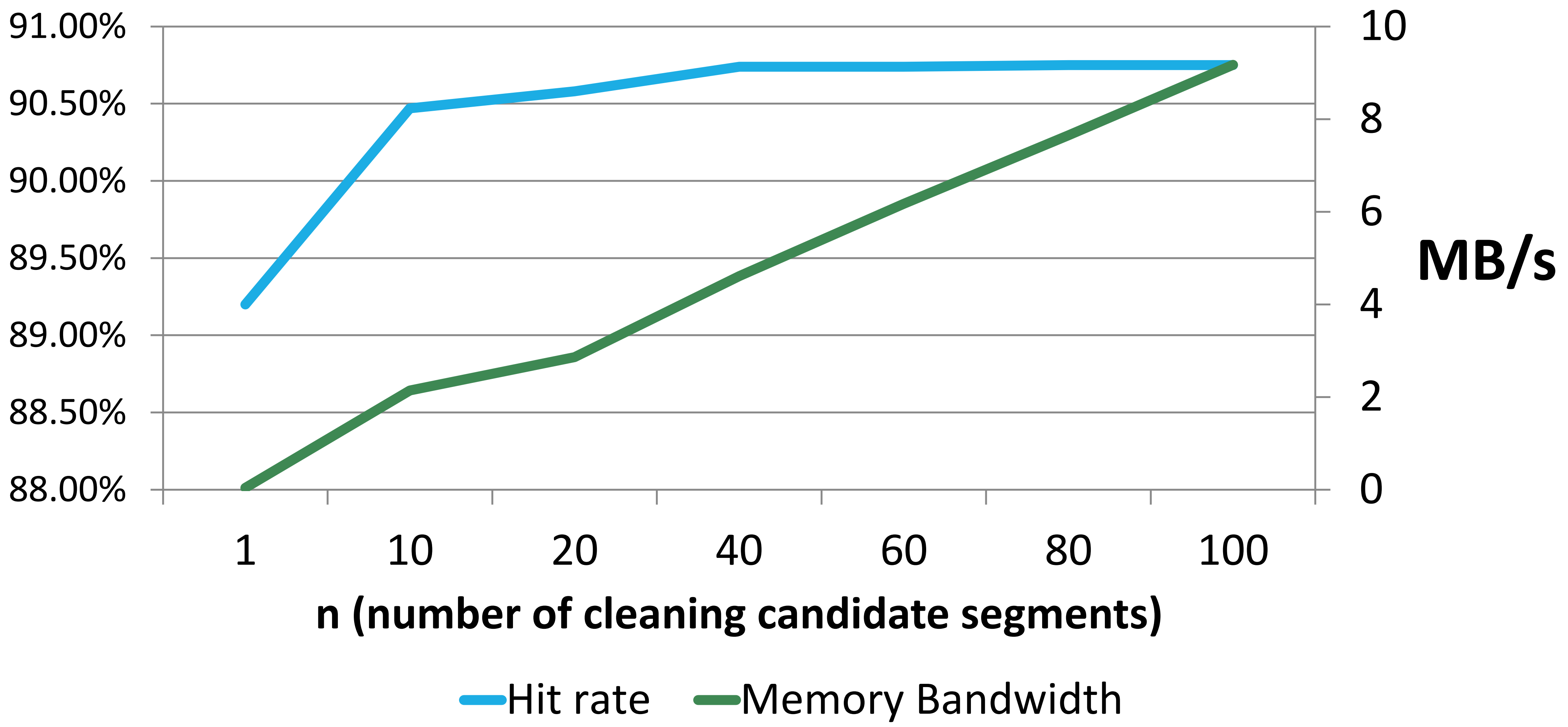


- Misses reduced by 40%
- Combined hit rate increase: 6% (85% → 91%)

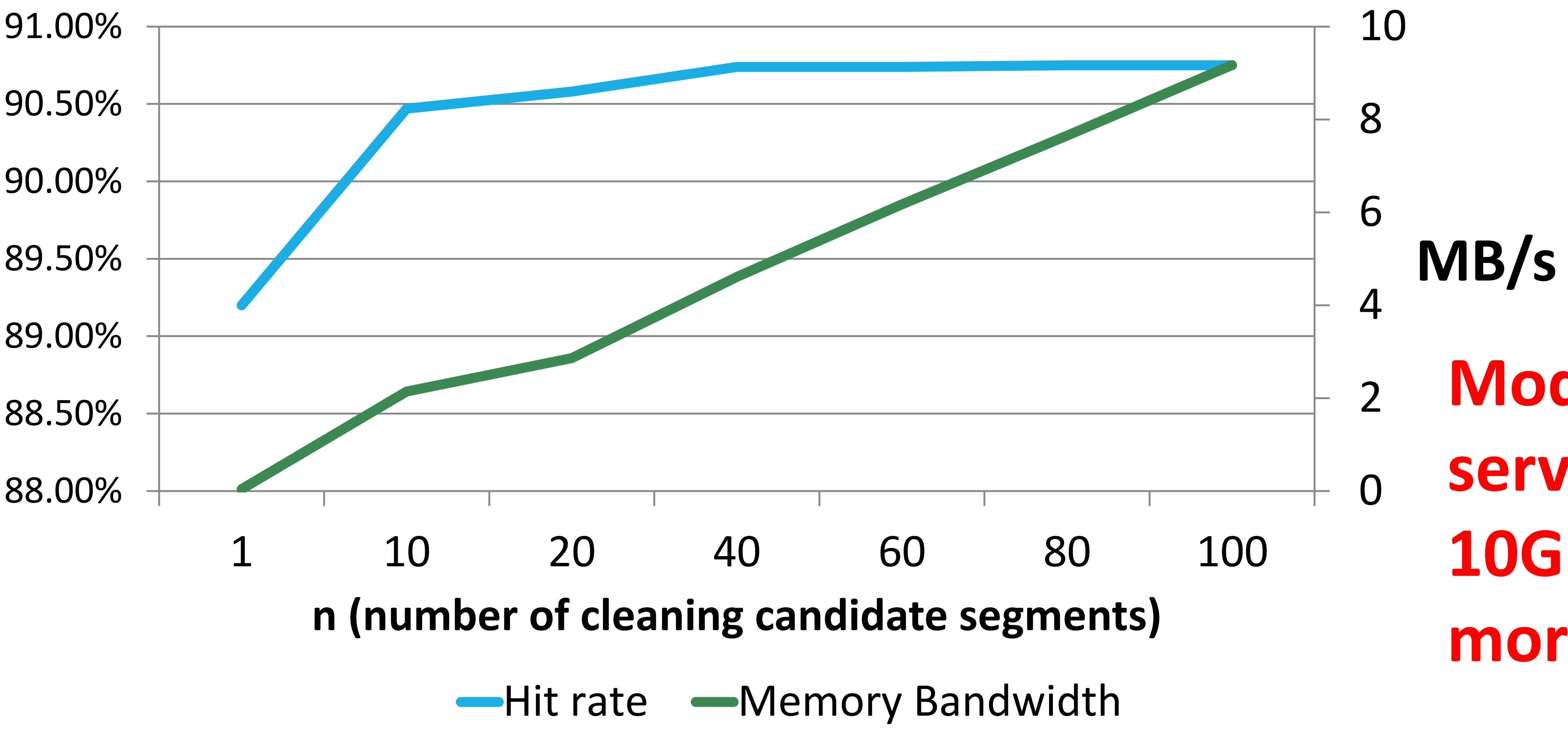
State-of-the-art Hit Rate Even for Single Tenant Applications

Policy	Memcached	Memshare (100% Reserved)
Average Single Tenant Hit Rate	88.3%	95.5%

Cleaning Overhead is Minimal



Cleaning Overhead is Minimal



Modern servers have 10GB/s or more!

Related Work

- Optimizing memory allocation using shadow queues
  - Cliffhanger [Cidon 2016]
- Log-structured single-tenant key-value stores
  - RAMCloud [Rumble 2014] and MICA [Lim 2014]
- Taxing idle memory
  - ESX Server [Waldspurger 2002]

Summary

- First multi-tenant key-value cache that:
  - Optimizes share for highest hit rate
  - Provides minimal guarantees
- Novel log-structured design
  - Use cleaner as enforcer

Appendix

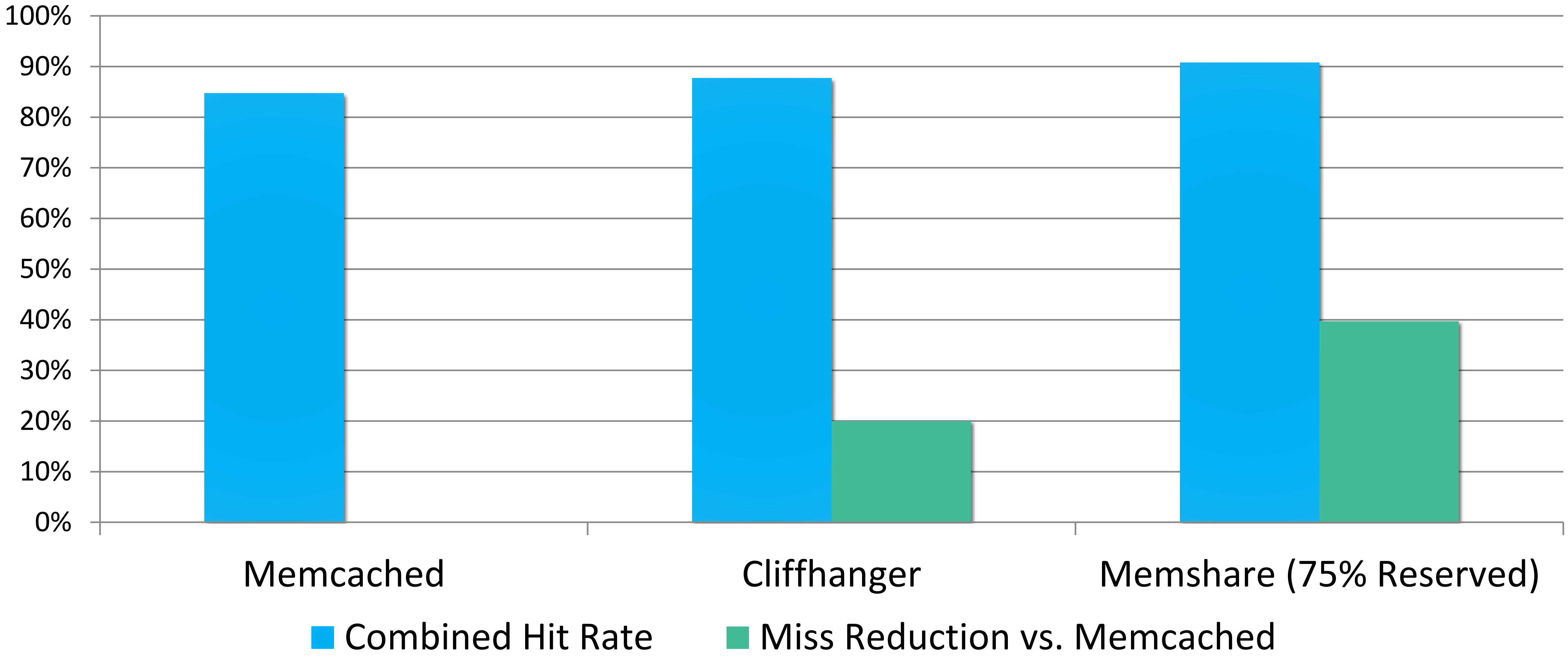
Idle Tax for Selfish Applications

- Some sharing models do not support pooled memory, each application is selfish
  - For example: Memcachier's Cache-as-a-Service
- Idle tax: reserved memory can be reassigned if idle
- Tax rate: determines portion of idle memory that can be reassigned
- If all memory is active: target memory = reserved memory

Partitioned vs. Idle Tax

Application	Hit Rate Partitioned	Hit Rate Memshare Idle Tax
Combined	**87.8%**	**88.8%**
A	97.6%	99.4%
B	98.8%	98.6%
C	30.1%	31.3%

State-of-the-art Hit rate



Nearly Identical Latency