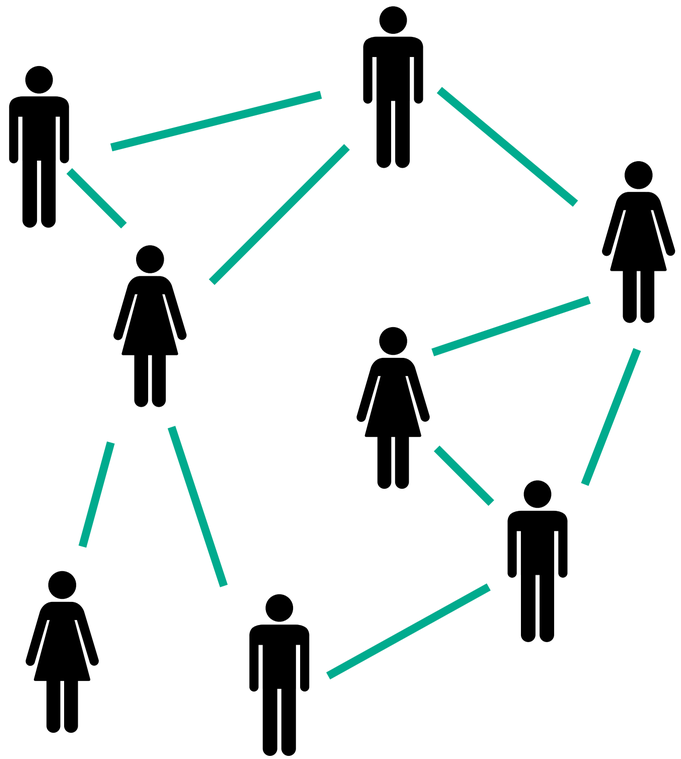
Caching on Flash: Kangaroo and Beyond

Sara McAllister
Carnegie Mellon University

Parallel Data Lab Meeting
Tuesday, March 1, 2022

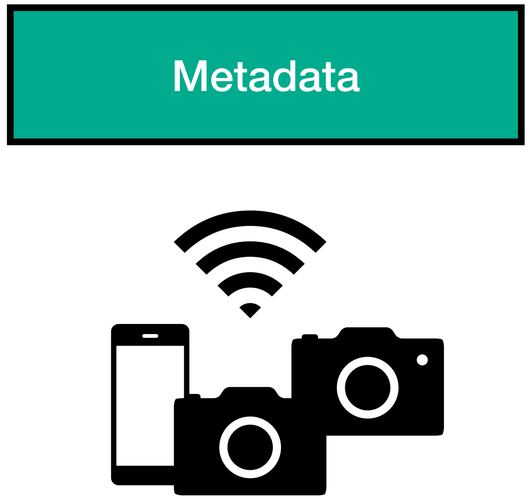
Tiny objects are prevalent



Social Graphs

Facebook social graph edges

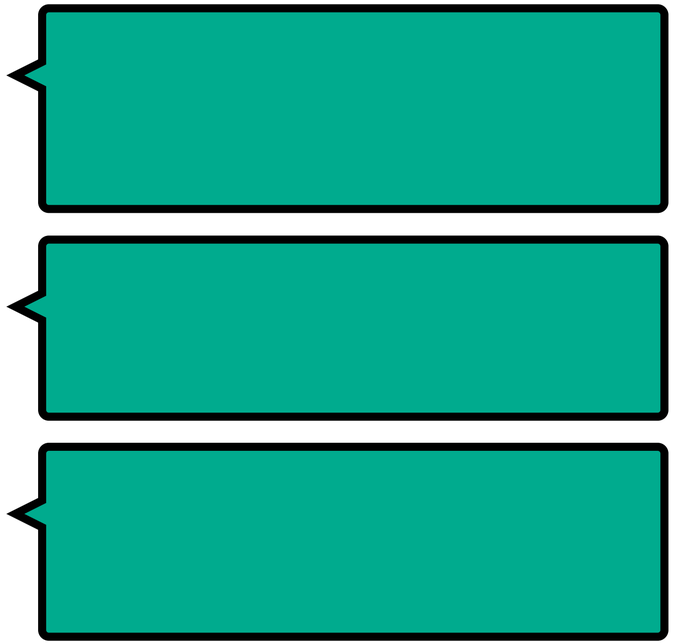
~100 bytes



IoT Metadata

Microsoft Azure sensor metadata

~300 bytes

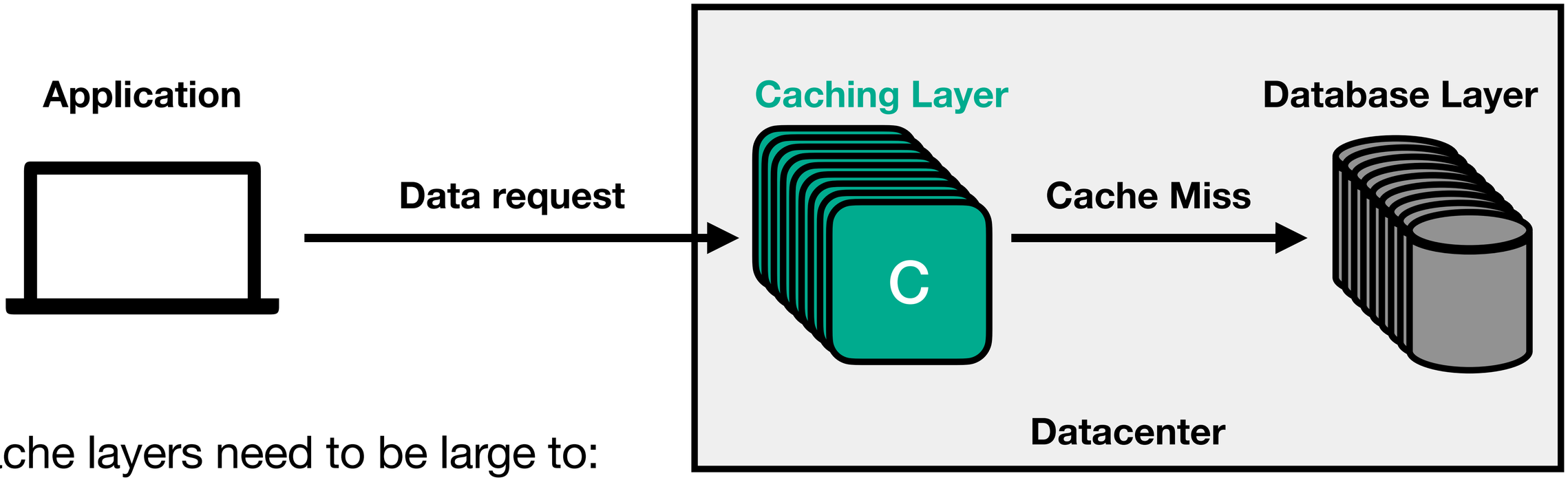


Tweets

Twitter tweets average

<33 characters

Caching at scale



Cache layers need to be large to:

1. lower average latency
2. keep load off of backend services

Flash is 100x cheaper per bit → Larger caches

Caching **billions of tiny objects** (~100 bytes) on flash

Prior Work → Too many flash writes or Large memory overhead →  **Wasted Money**

Kangaroo reduces misses by 29%

while keeping writes and memory under production constraints

McAllister SOSP 2021

Denser Flash → **Less flash writes**

New flash interfaces needed use **denser flash**

Talk Outline

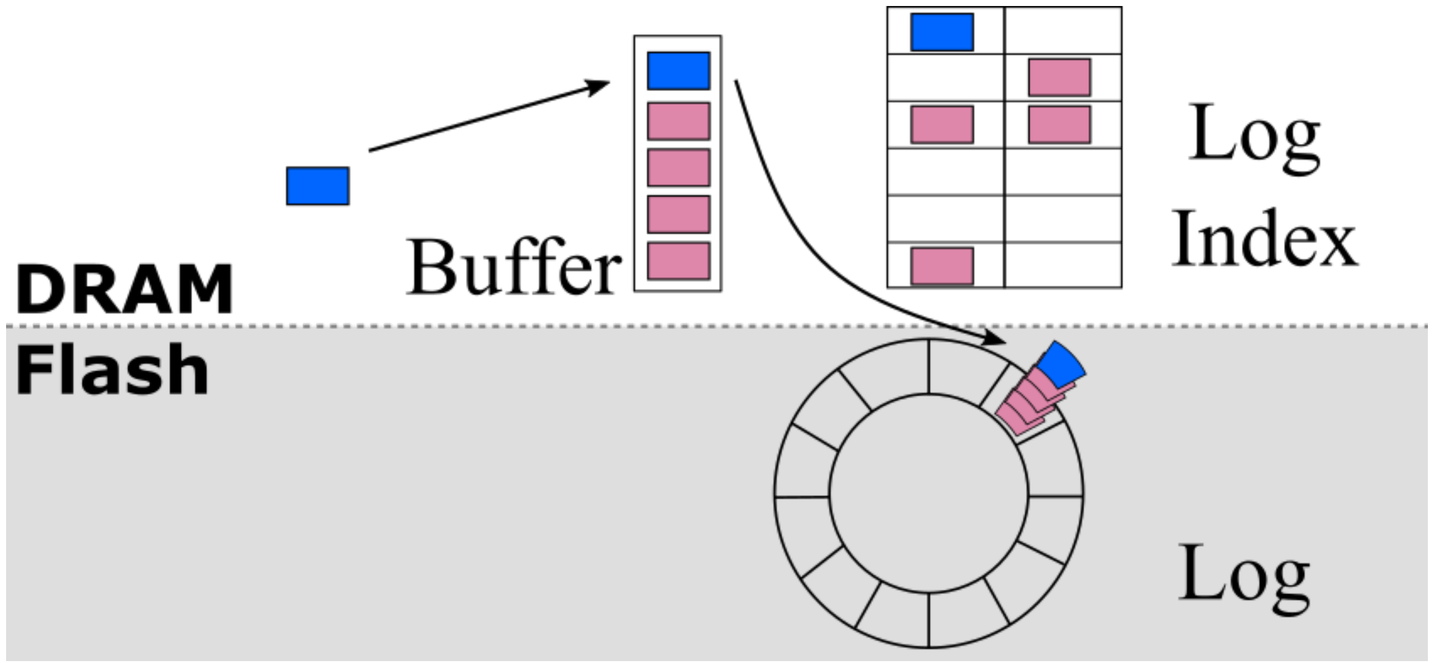
1) Kangaroo [McAllister SOSP 2021]
   - Introduction
   - Prior work: Too many writes or too much DRAM overhead
   - Kangaroo design
   - Results
2) Caching on new flash interfaces
   - Kangaroo isn’t enough for new generations of flash
   - New flash interfaces

Caching on flash → Write constraint

Flash allows cheaper than DRAM, but

- Flash has **limited write endurance**

Most flash caches use a **log-structured cache**



Caching on flash → Write constraint

Flash allows cheaper than DRAM, but

- Flash has **limited write endurance**

Most flash caches use a **log-structured cache**

+ **Buffered writes** minimize writes to flash

- **Full in-memory index** (with just 30 bits/object): Flashield (Eisenman NSDI '19)

2 TB flash cache → 75 GB memory overhead

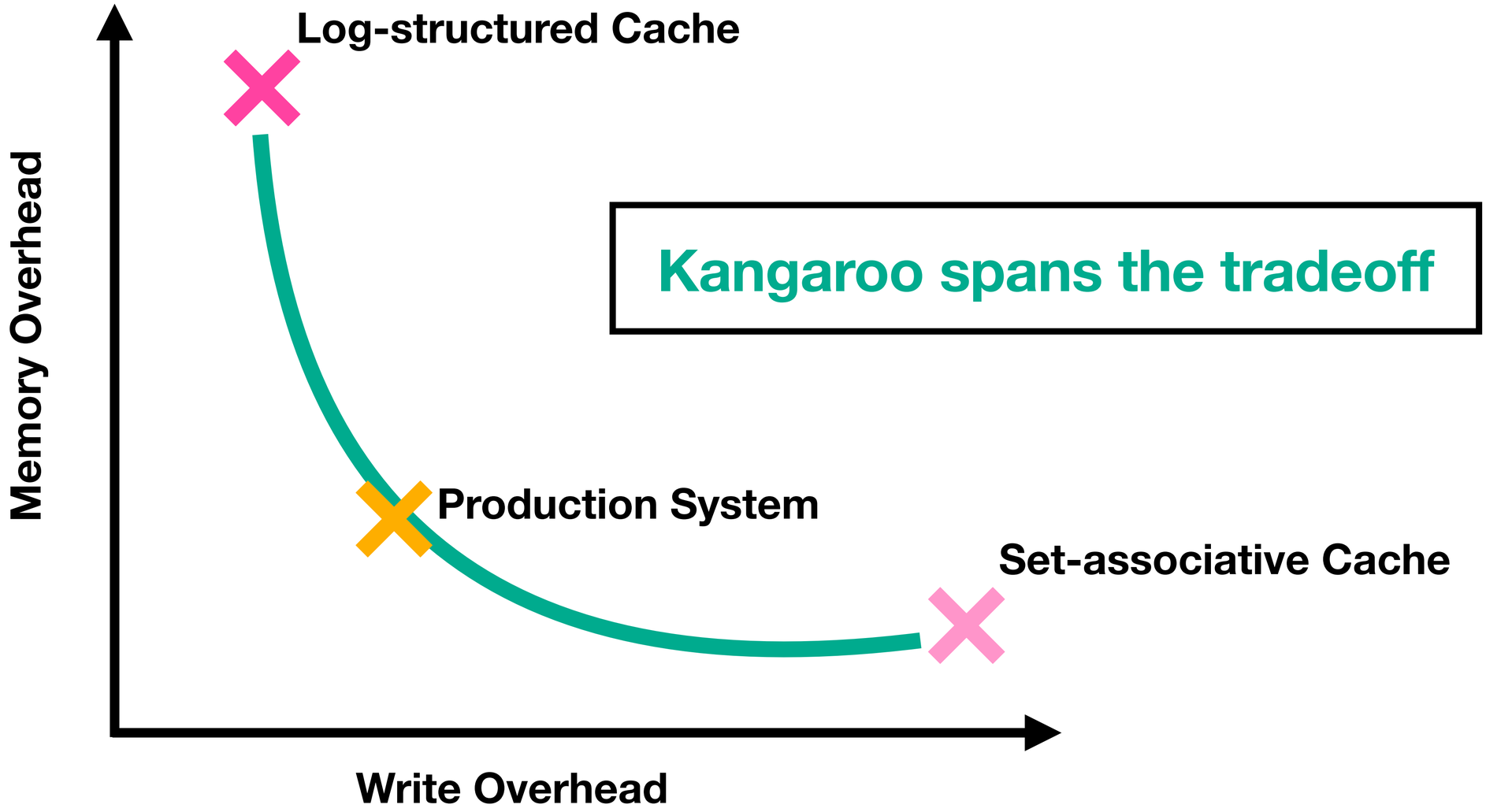
Low memory overhead → Set-associative cache

CacheLib (Berg OSDI '20)

+ **No index** → Low memory overhead

- **Large write amplification** (≥ 4 KB written for each object)

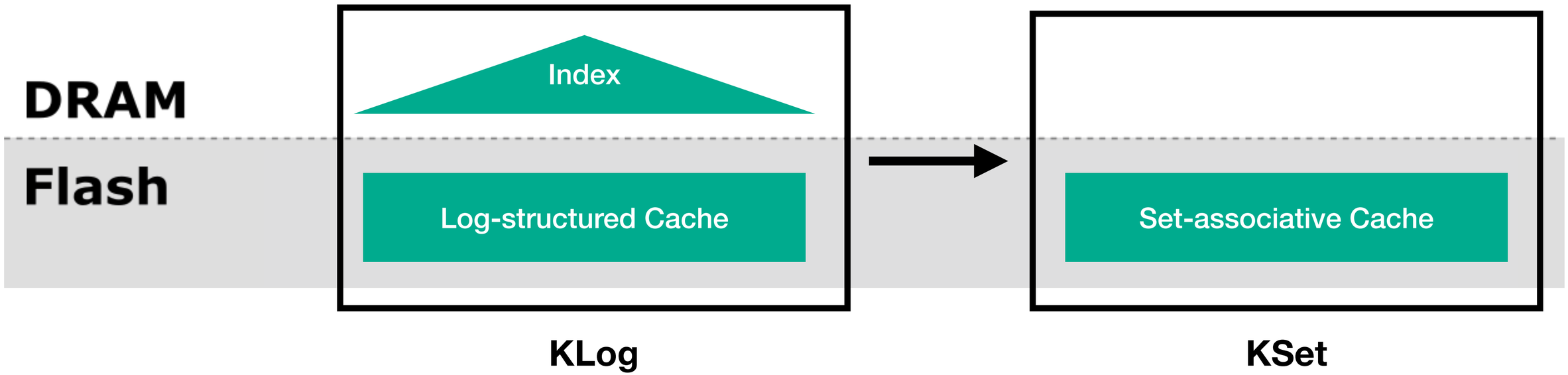
Prior work: Too much DRAM or too many writes



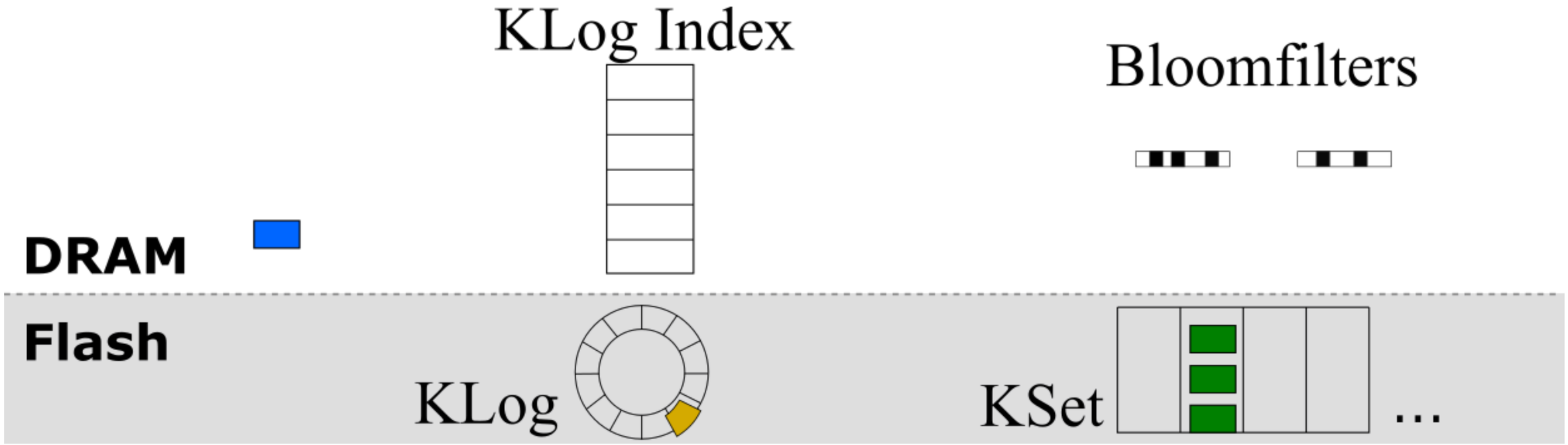
Talk Outline

1) Kangaroo [McAllister SOSP 2021]
   - Introduction
   - Prior work: Too many writes or too much DRAM overhead
   - Kangaroo design
   - Results
2) Caching on new flash interfaces
   - Kangaroo isn’t enough for new generations of flash
   - New flash interfaces

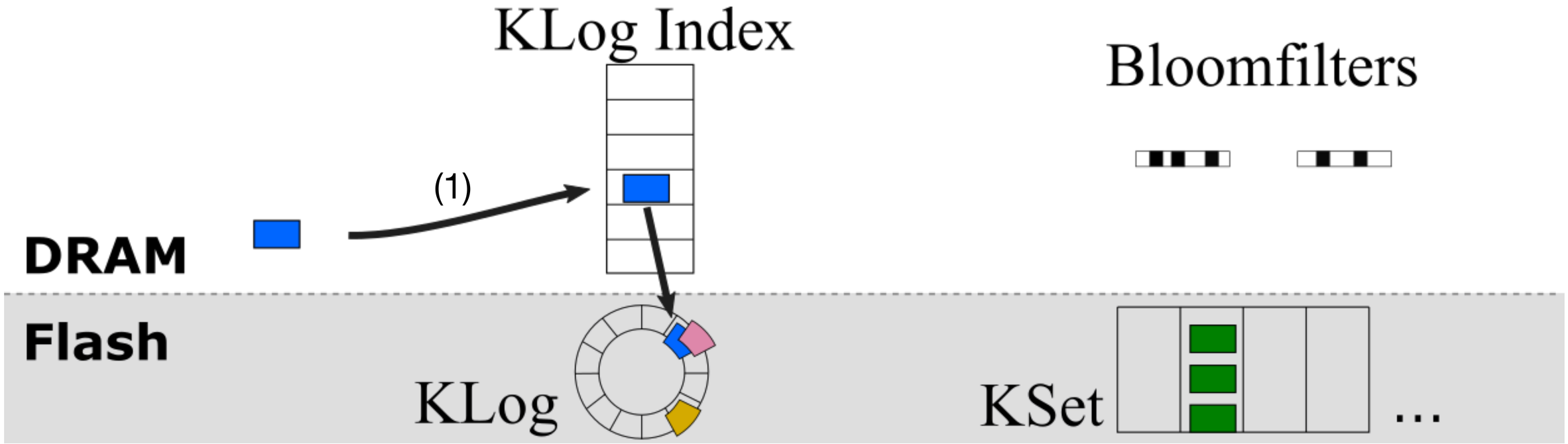
Kangaroo Overview



Inserting Objects in Kangaroo

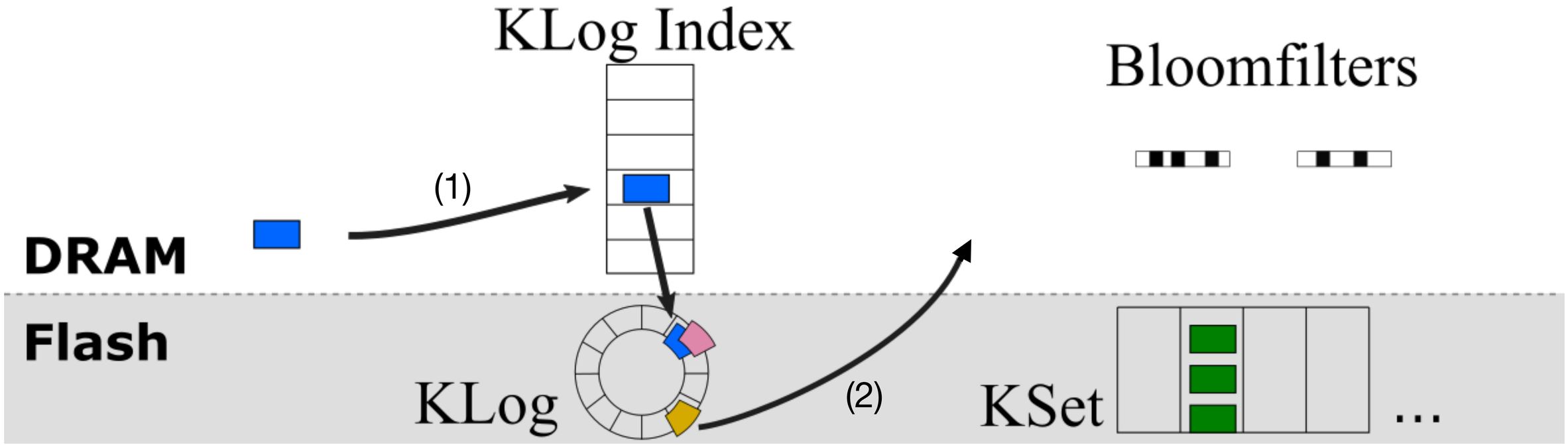


Inserting Objects in Kangaroo



1) Insert to KLog via buffered write

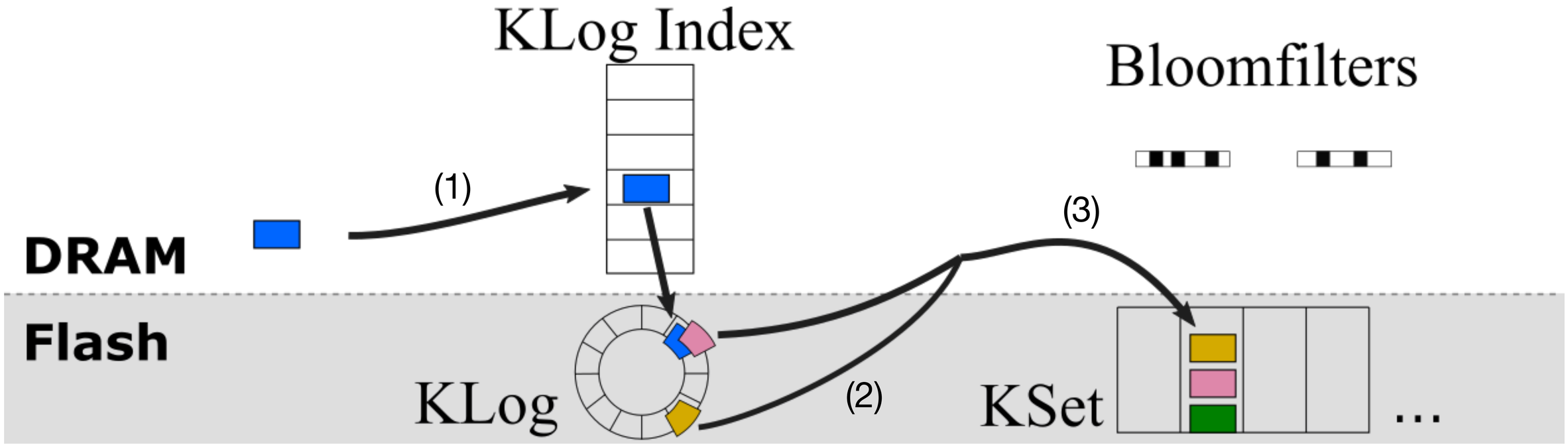
Inserting Objects in Kangaroo



1) Insert to KLog via buffered write

2) Flush object from KLog to KSet

Inserting Objects in Kangaroo

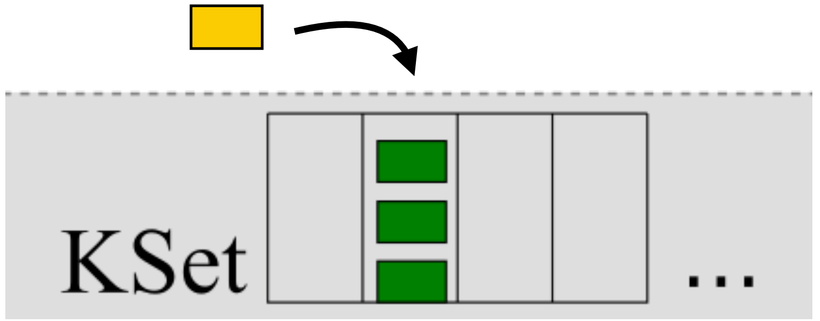


1) Insert to KLog via buffered write
2) Flush object from KLog to KSet
3) Move **all objects** in KLog that map to the same set

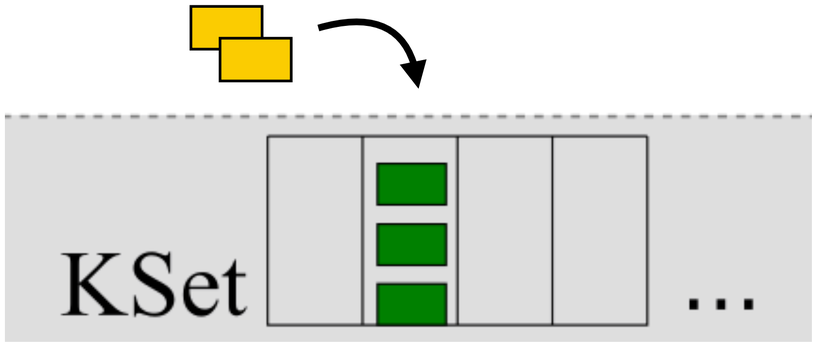
Amortizing KSet flash writes using KLog

Two small objects **halve** write amplification (WA) to KSet

1 new object



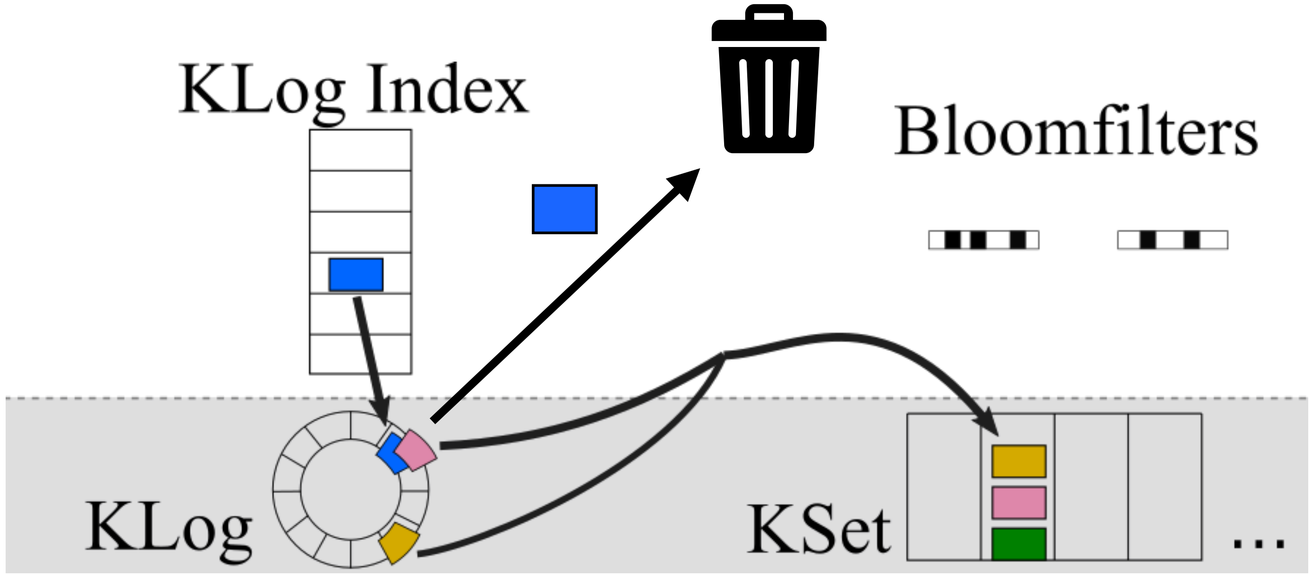
2 new objects



KLog allows more time to find **set collisions** and **amortize WA**

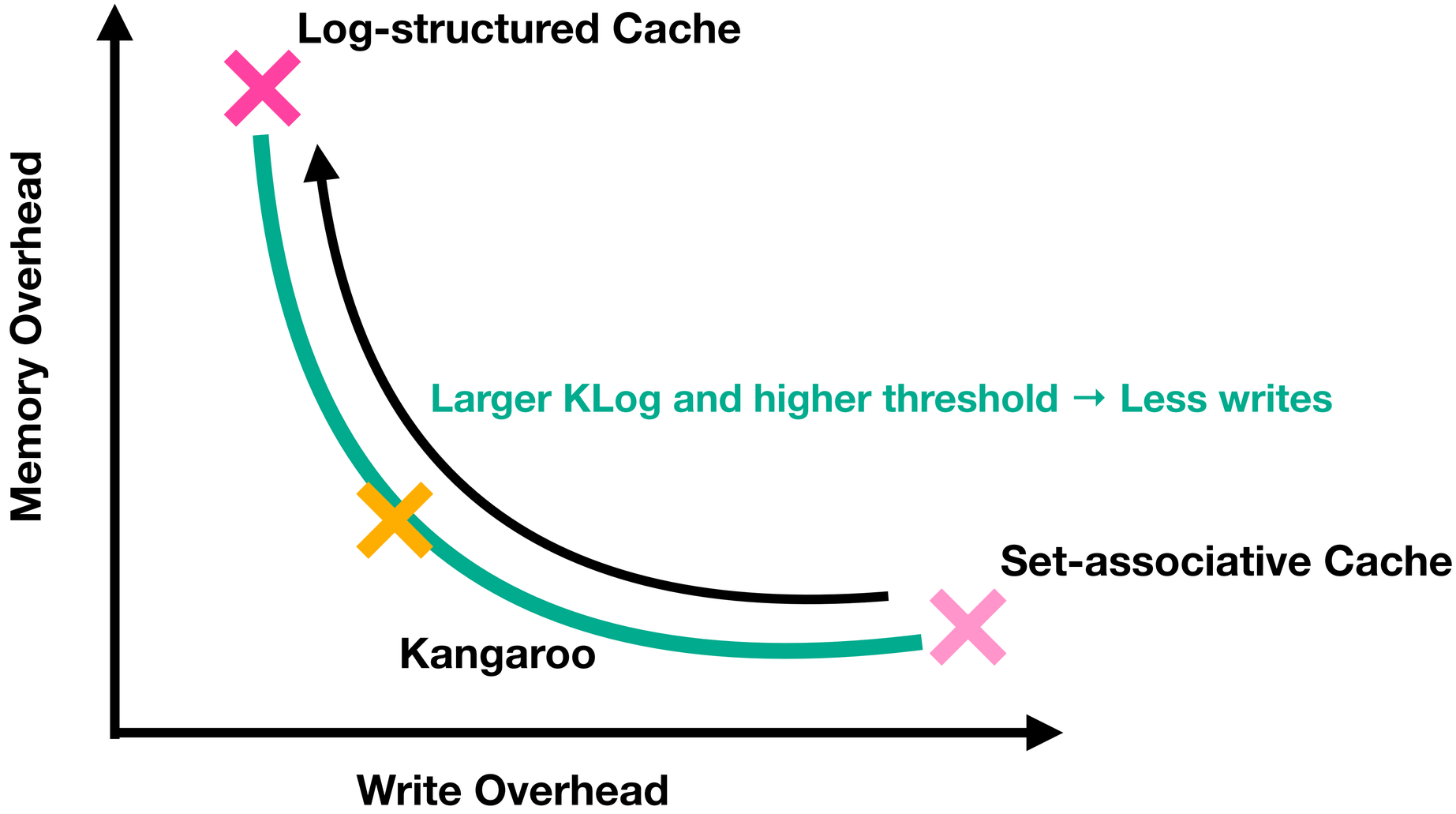
Threshold admission

We can **choose** which objects to discard based on write cost



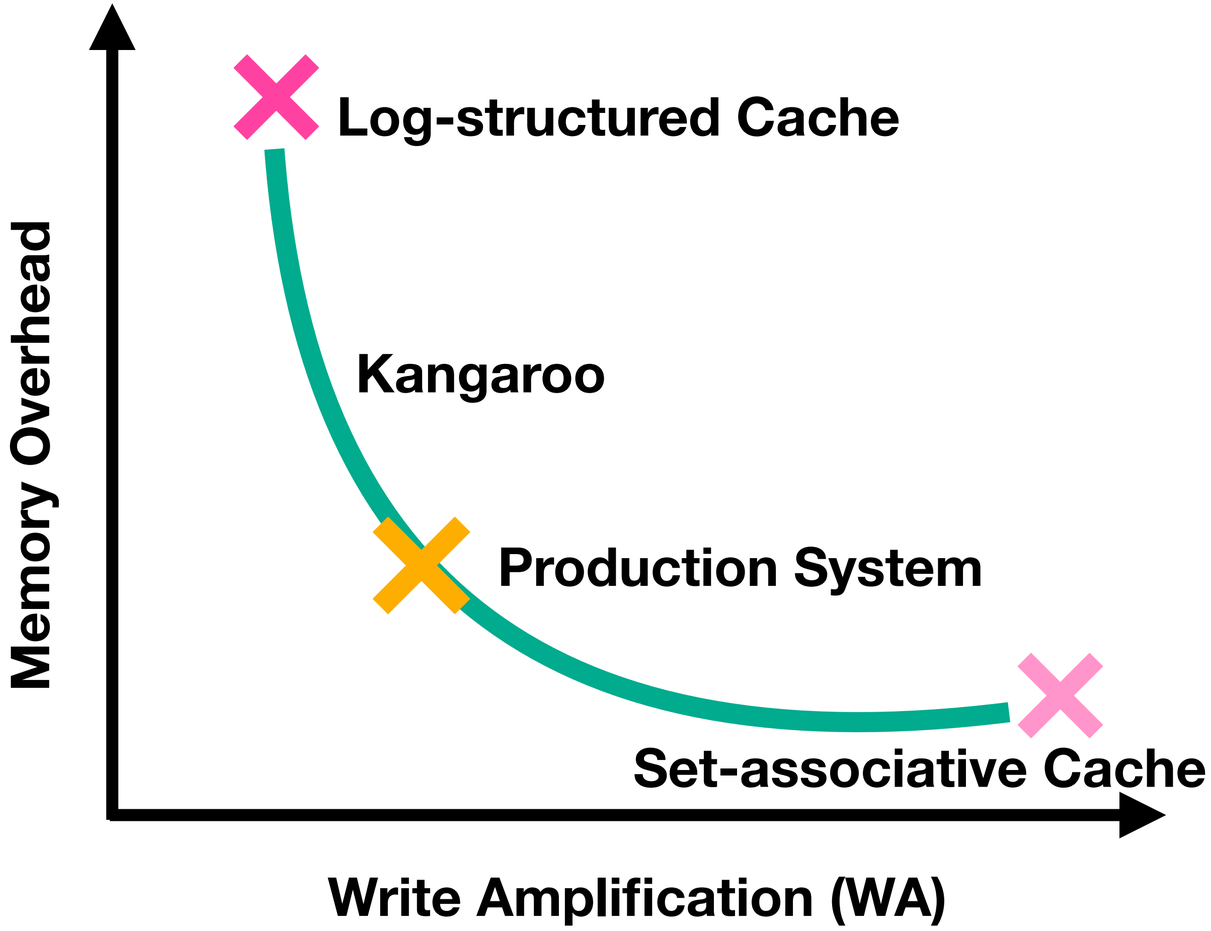
Only rewrite a set in KSet if at least threshold, **n**, number of objects

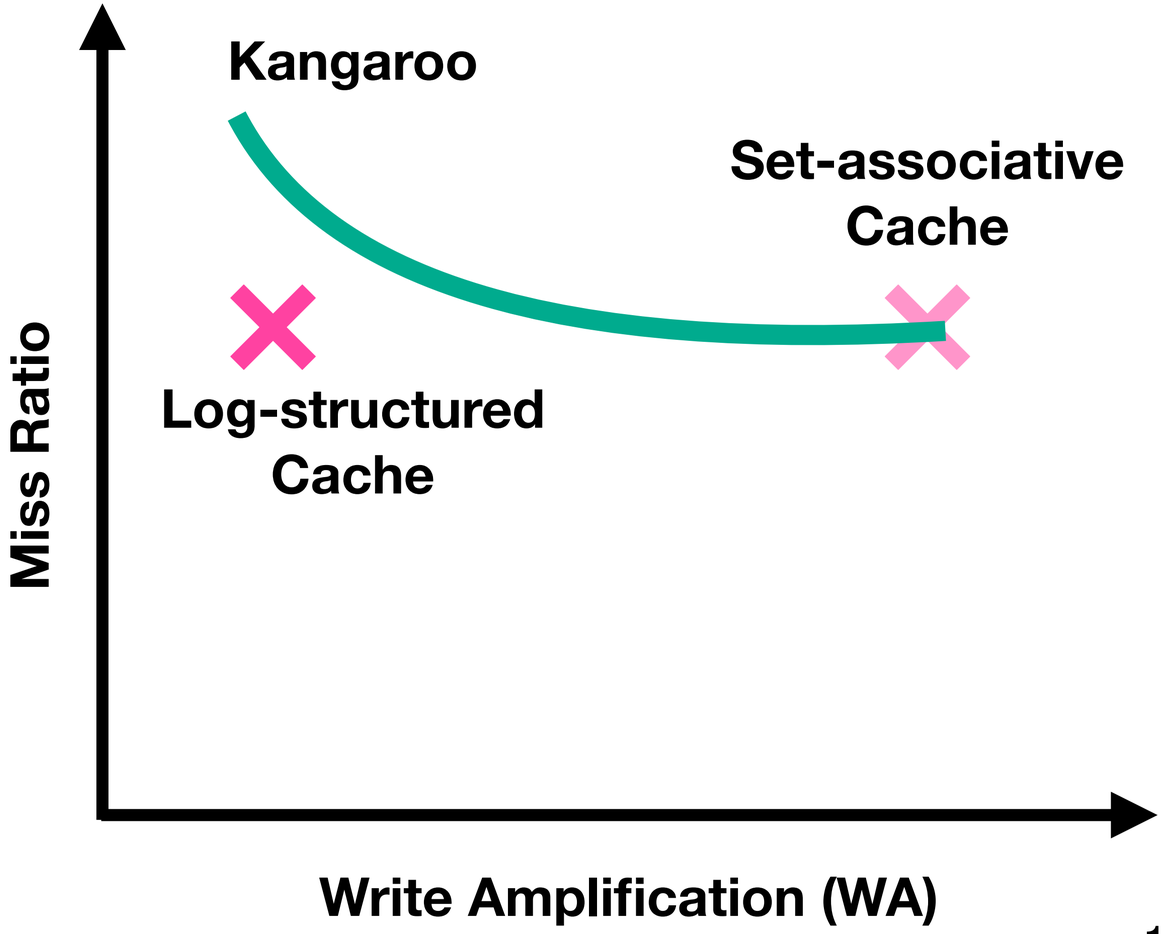
Kangaroo can trade off overheads



Miss ratio: Another tradeoff

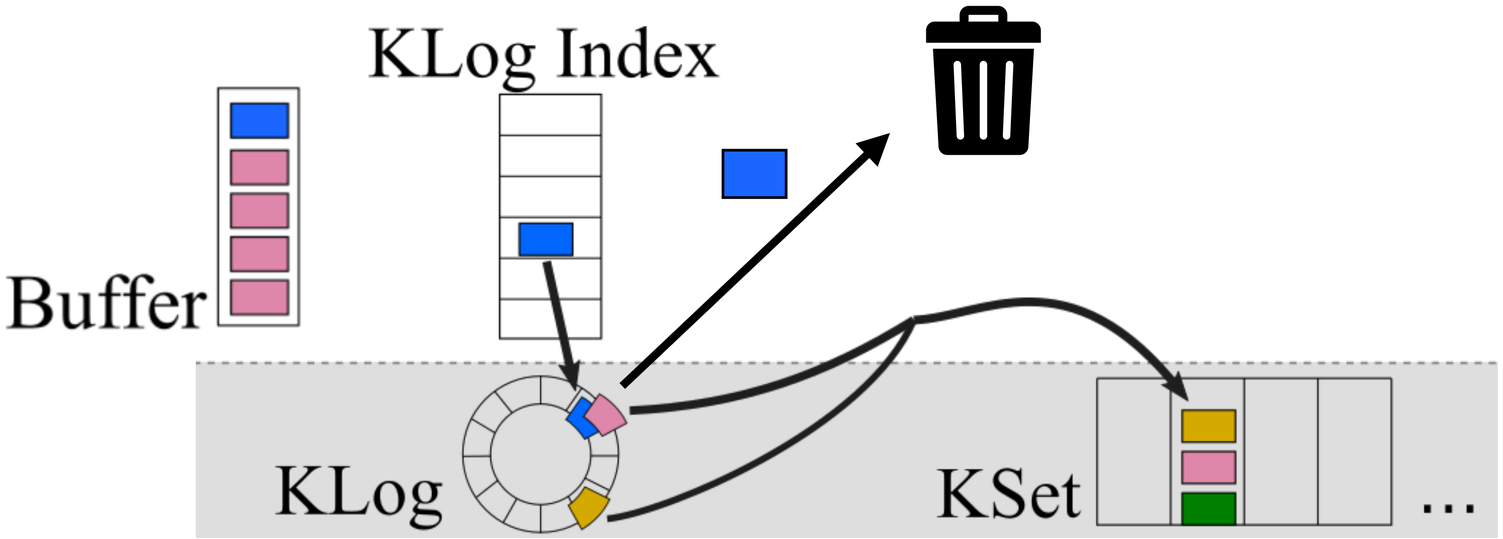
Does discarding objects cause miss ratio losses?





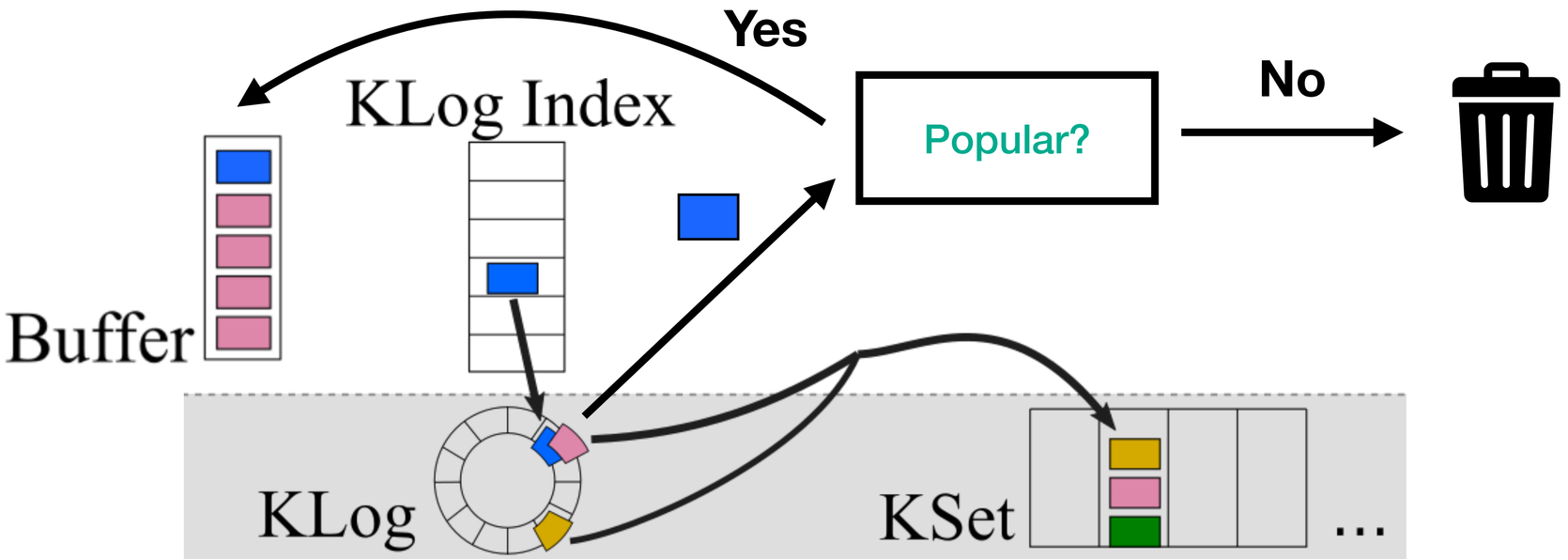
Readmission to KLog

Popular objects rewritten to KLog to minimize write cost

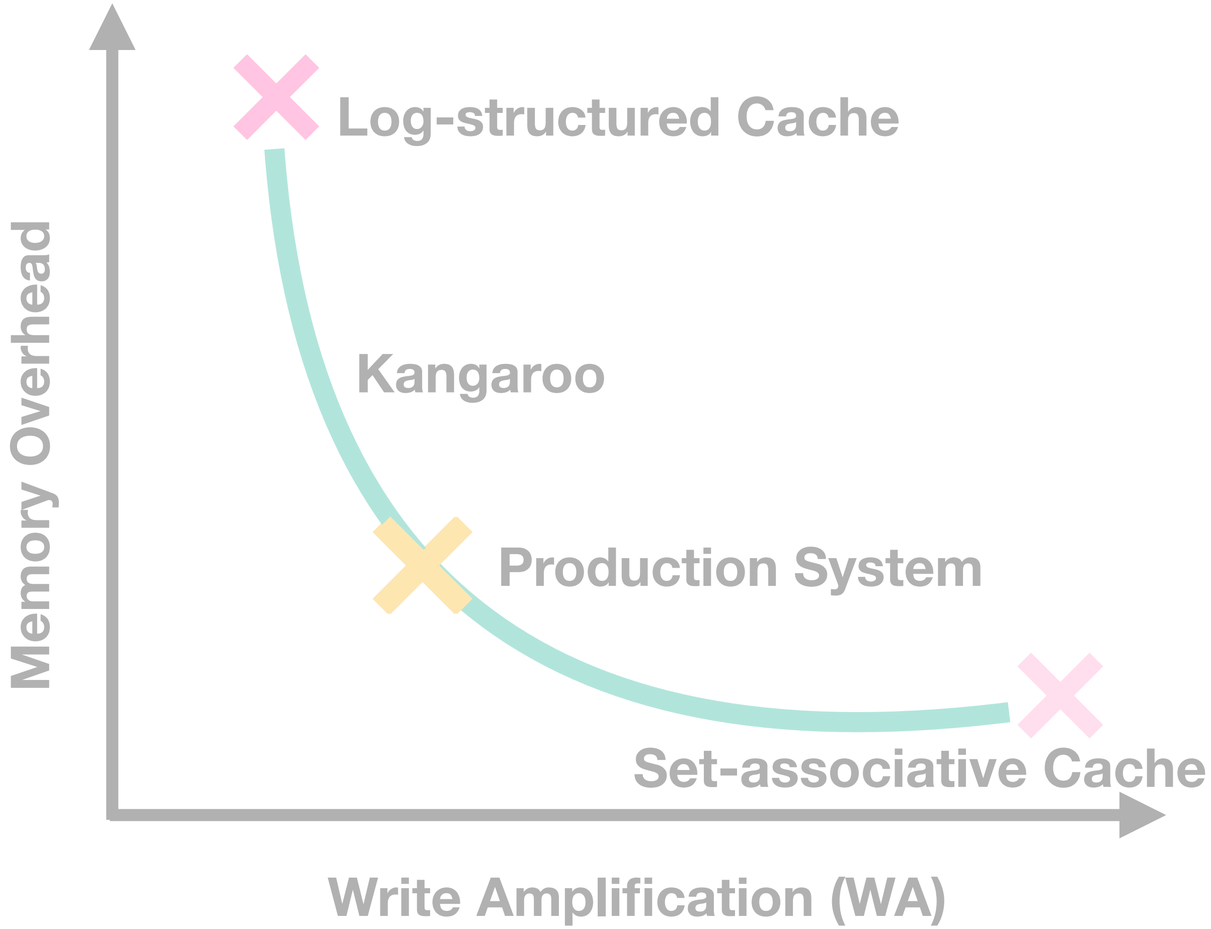


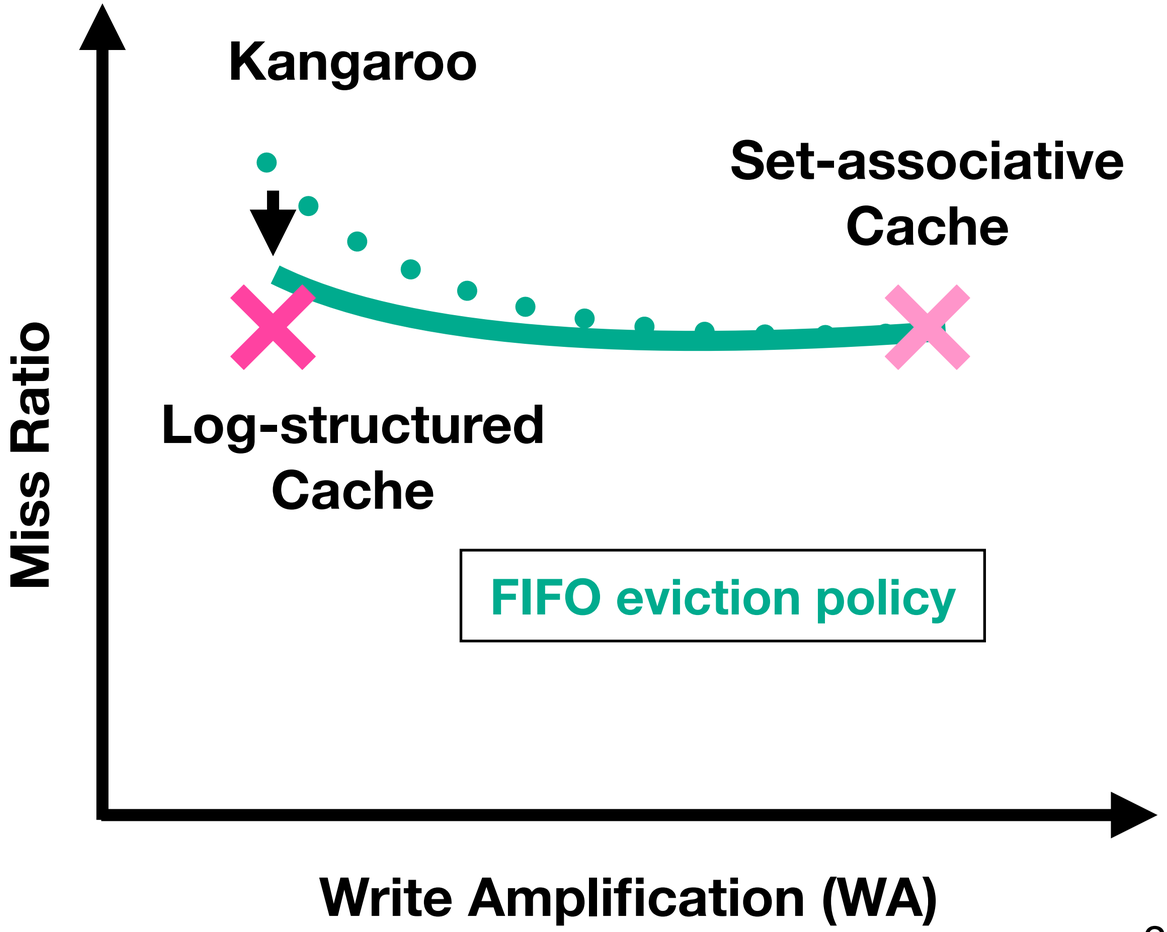
Readmission to KLog

Popular objects rewritten to KLog to minimize write cost

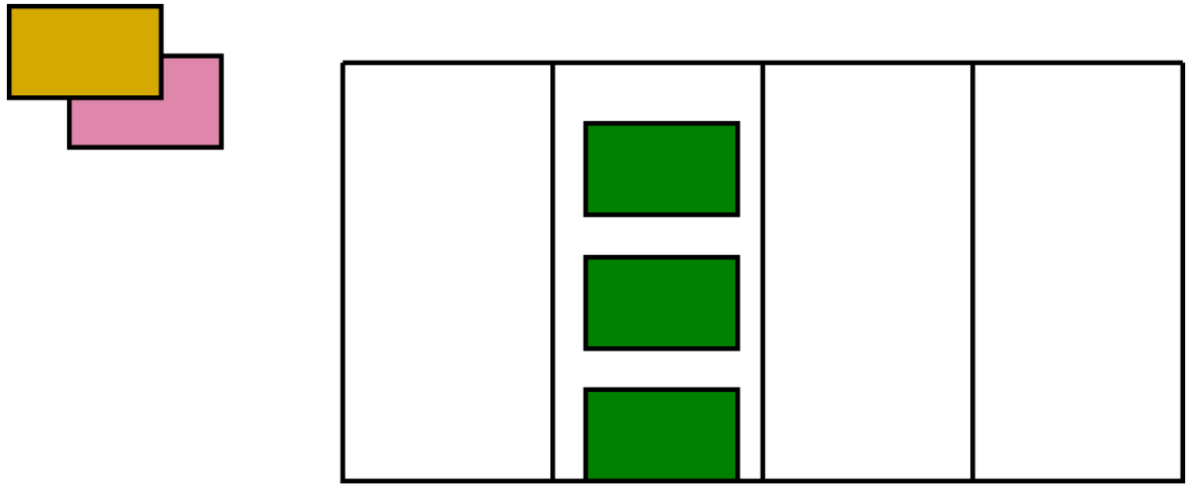


Readmission improves miss ratio





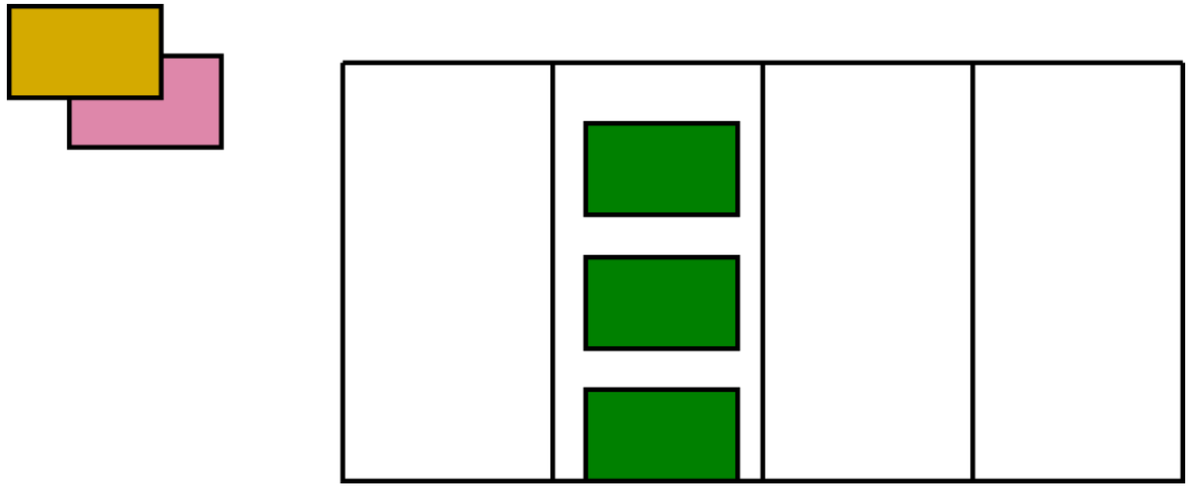
RRIParoo eviction in KSet helps miss ratio



Problem: Evict from set to make room for log objects while:

- Retaining more popular objects
- Maintaining small memory overhead

RRIParoo eviction in KSet helps miss ratio



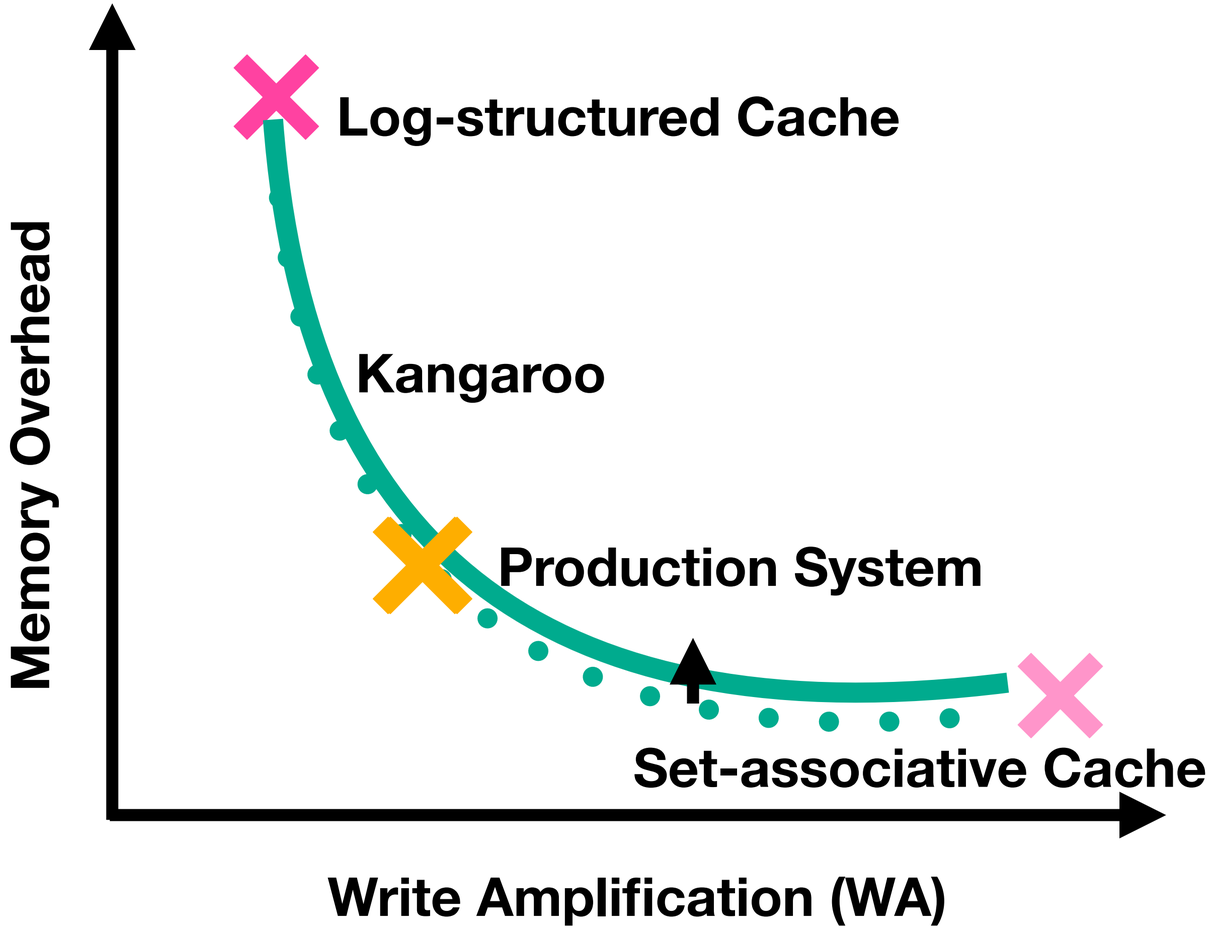
Problem: Evict from set to make room for log objects while:

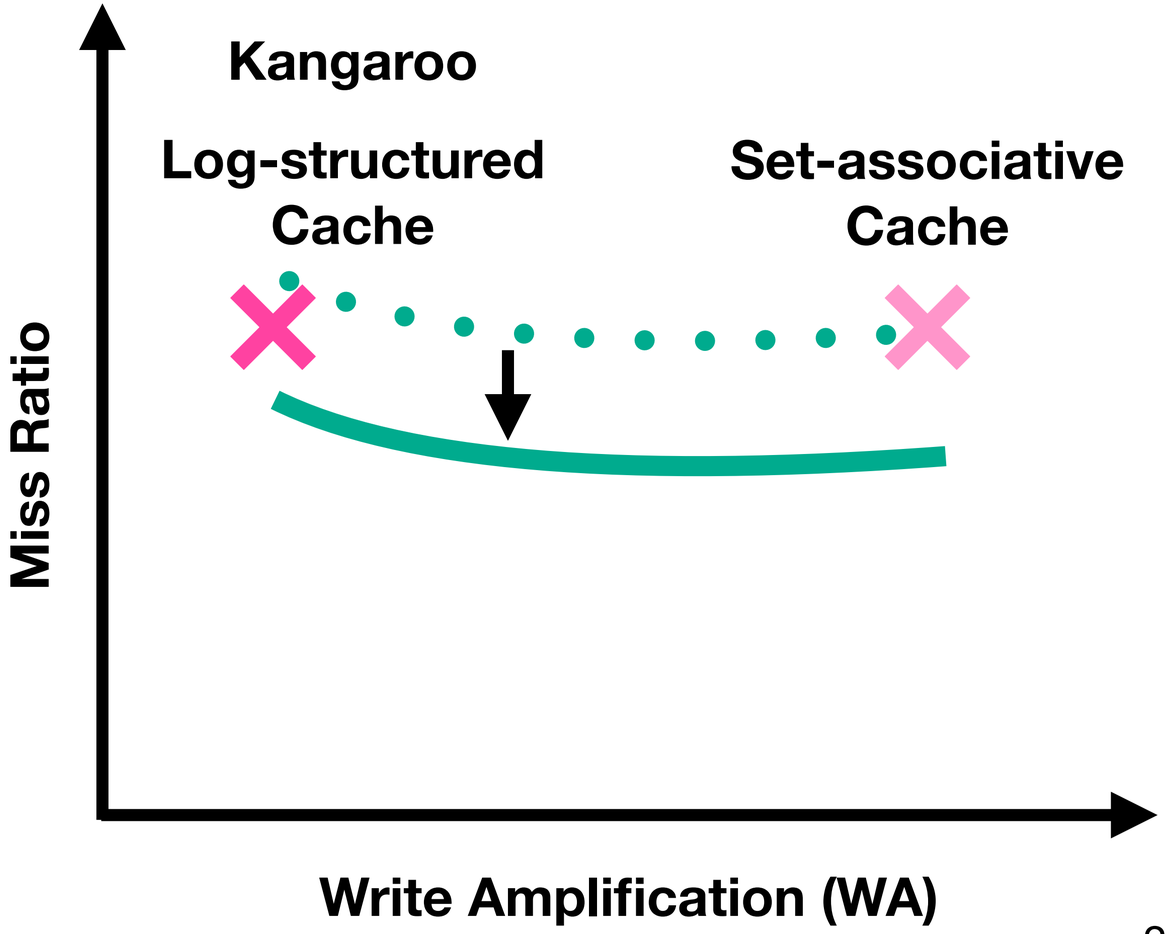
- Retaining more popular objects
- Maintaining small memory overhead

Solution: RRIParoo, a modified version of RRIP RRIP (Jaleel ISCA'10)

- **1 bit DRAM/object** in KSet with RRIParoo

RRIParoo improves miss ratio



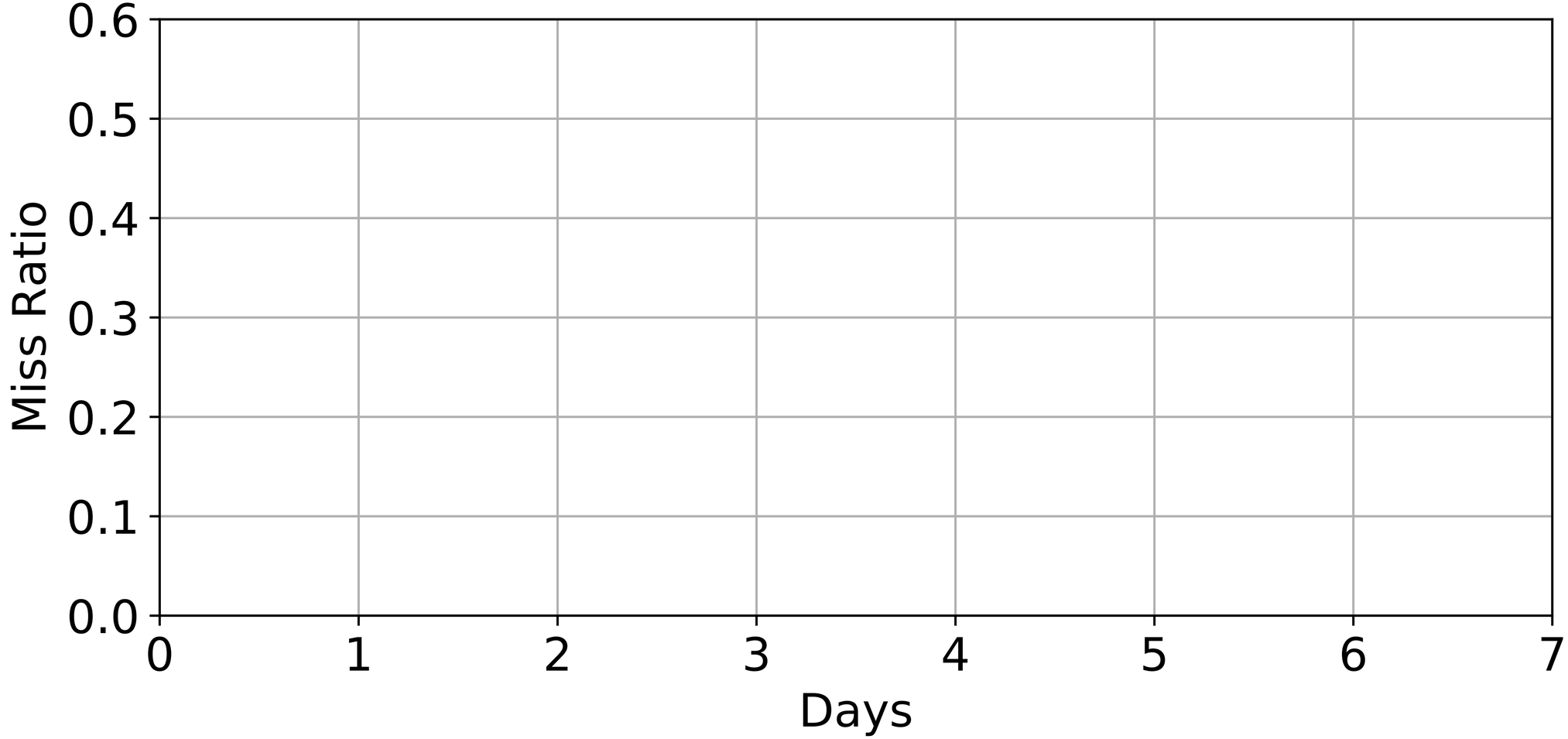


Talk Outline

1) Kangaroo [McAllister SOSP 2021]
   - Introduction
   - Prior work: Too many writes or too much DRAM overhead
   - Kangaroo design
   - Results
2) Caching on new flash interfaces
   - Kangaroo isn’t enough for new generations of flash
   - New flash interfaces

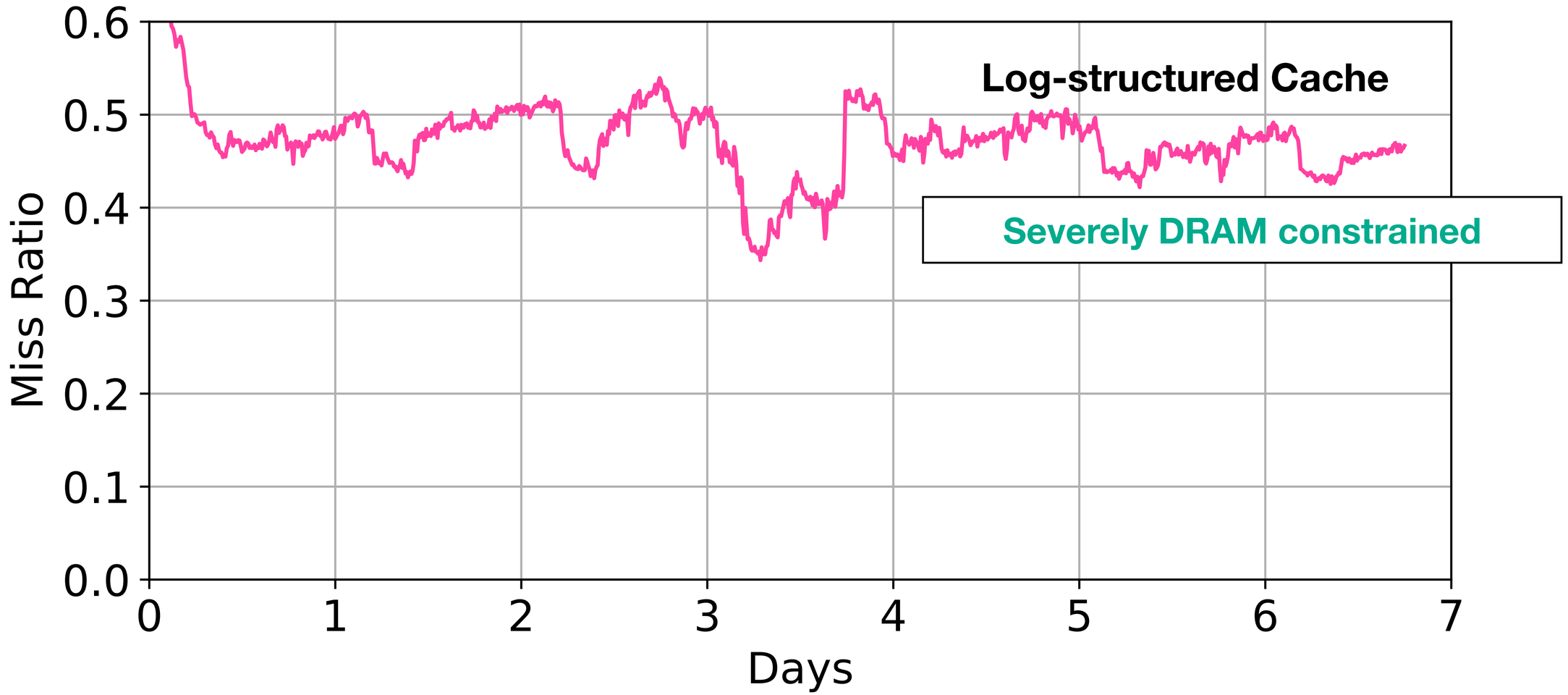
Kangaroo has best miss ratio

Run on 2 TB flash drive with a 7-day Facebook trace with 16 GB DRAM and 3 DWPD



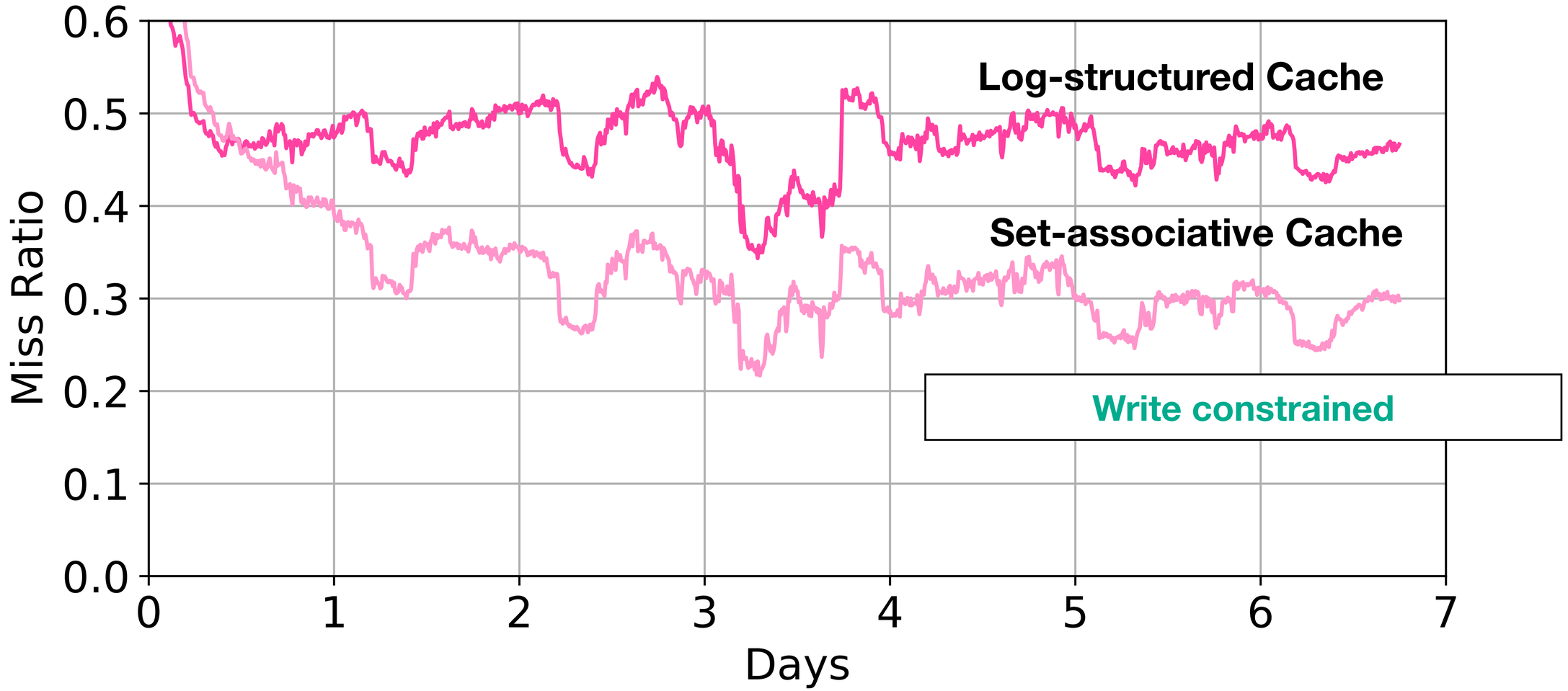
Kangaroo has best miss ratio

Run on 2 TB flash drive with a 7-day Facebook trace with 16 GB DRAM and 3 DWPD



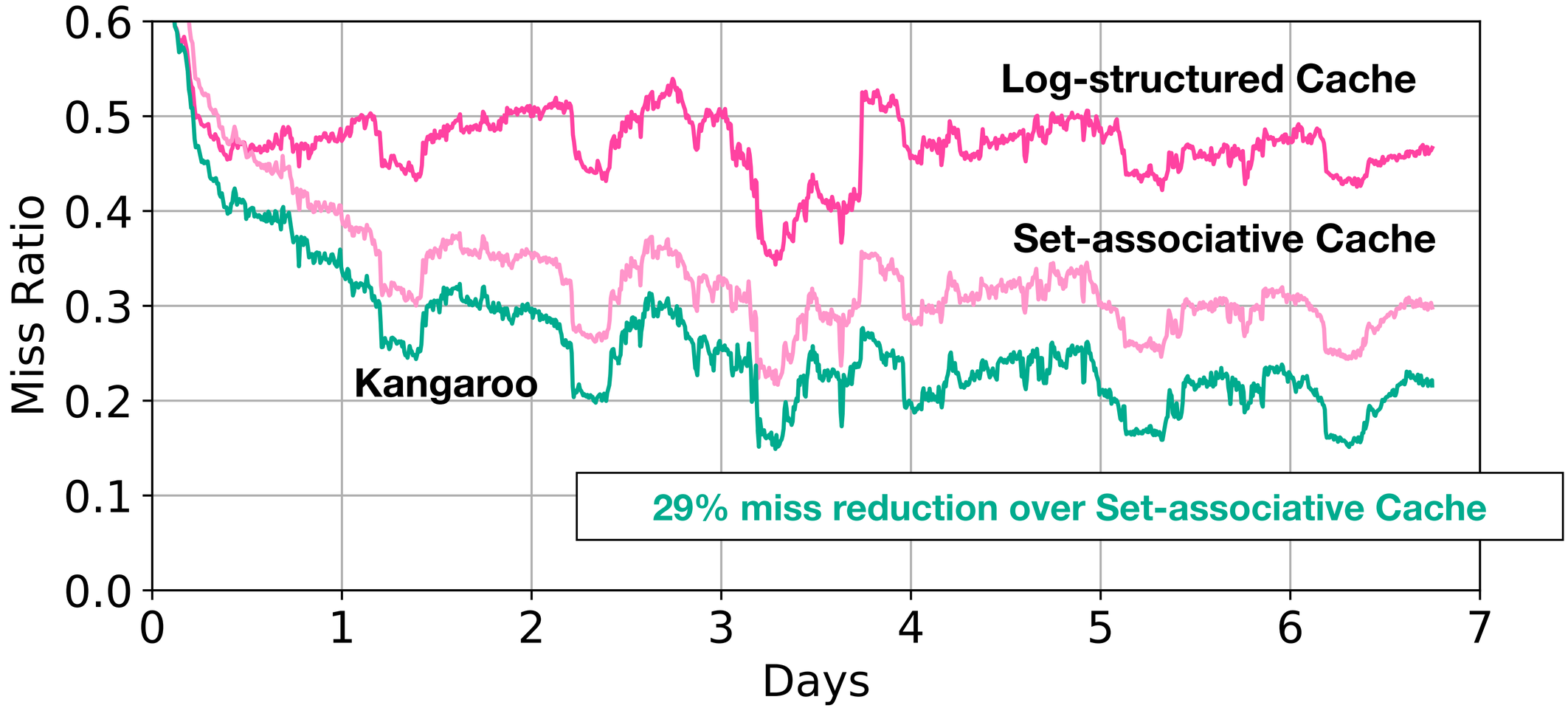
Kangaroo has best miss ratio

Run on 2 TB flash drive with a 7-day Facebook trace with 16 GB DRAM and 3 DWPD



Kangaroo has best miss ratio

Run on 2 TB flash drive with a 7-day Facebook trace with 16 GB DRAM and 3 DWPD



Kangaroo: Caching Billions of Tiny Objects on Flash

A flash cache for tiny objects that has:

1. Write rate within bounds for device lifetime by amortizing write costs
2. Low memory metadata overhead at **7.0 bits/object**
3. **29%** decrease in misses over than competitors

More on Kangaroo can be found in our SOSP ’21 paper:

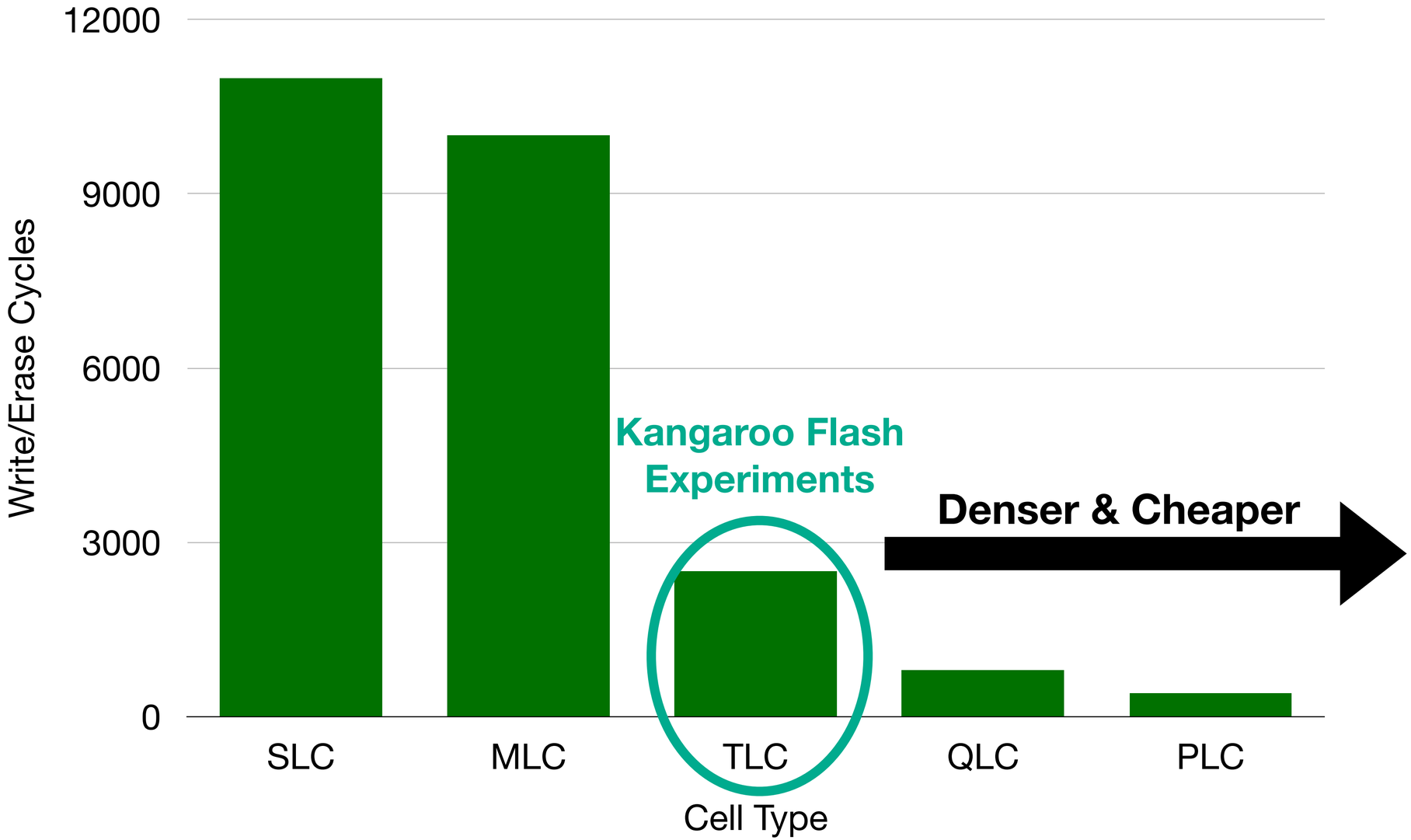
Kangaroo: Caching Billions of Tiny Objects on Flash.

Sara McAllister (CMU), Benjamin Berg (CMU), Julian Tutuncu-Macias (CMU), Juncheng Yang (CMU), Sathya Gunasekar (FB), Jimmy Lu (FB), Daniel S. Berger (MSR/UW), Nathan Beckmann (CMU), and Gregory R. Ganger (CMU)

Talk Outline

1) Kangaroo [McAllister SOSP 2021]
   - Introduction
   - Prior work: Too many writes or too much DRAM overhead
   - Kangaroo design
   - Results
2) Caching on new flash interfaces
   - Kangaroo isn’t enough for new generations of flash
   - New flash interfaces

Flash's future write endurance will drop



https://blocksandfiles.com/2019/08/07/penta-level-cell-flash/, Jaffer FAST 2022

Modeling cost of caching

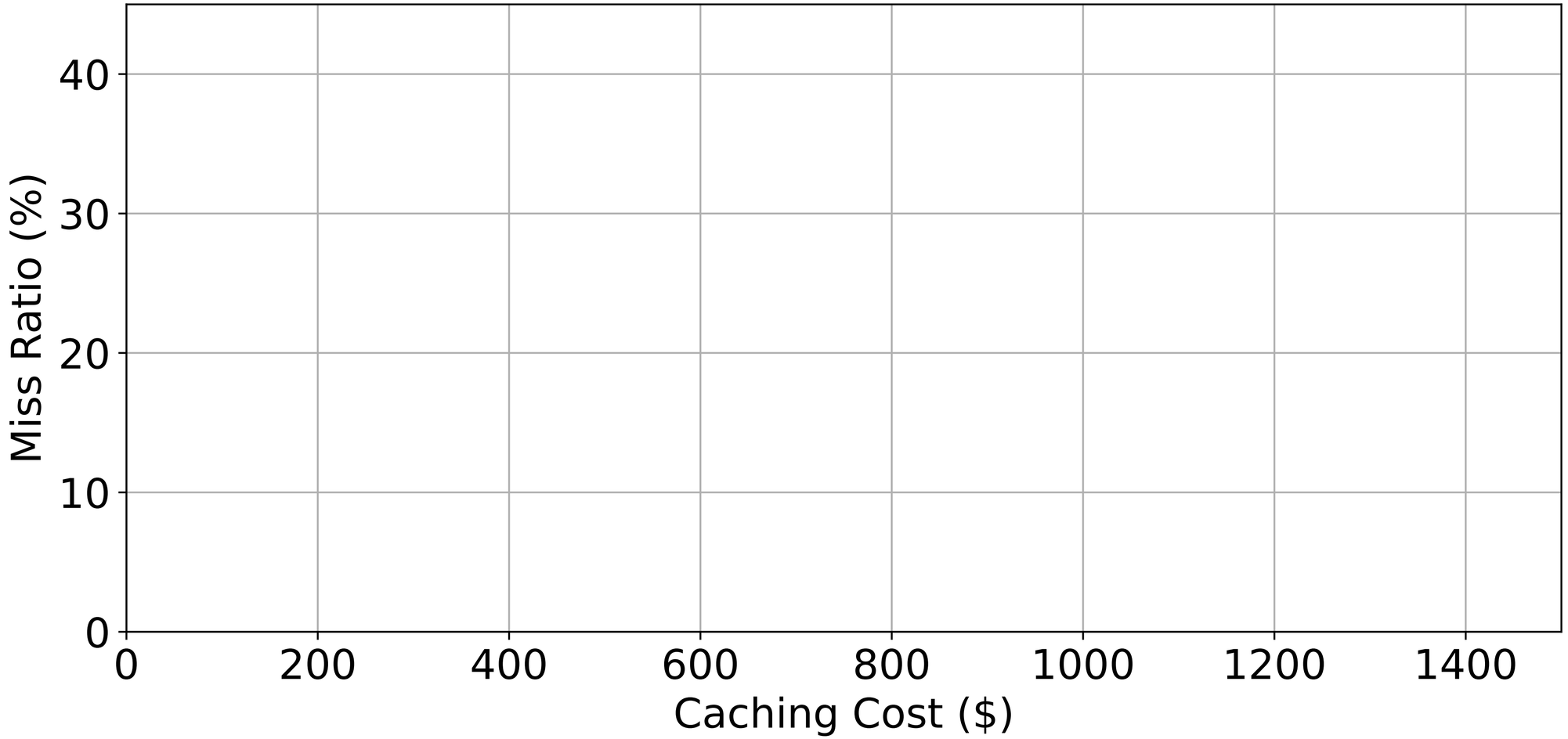
Assuming fixed stream of requests through one server:

- Fixed memory size
- Can buy different flash capacities
  - **Linear cost model** for different sized flash devices
  - Extrapolated from public TLC flash device costs

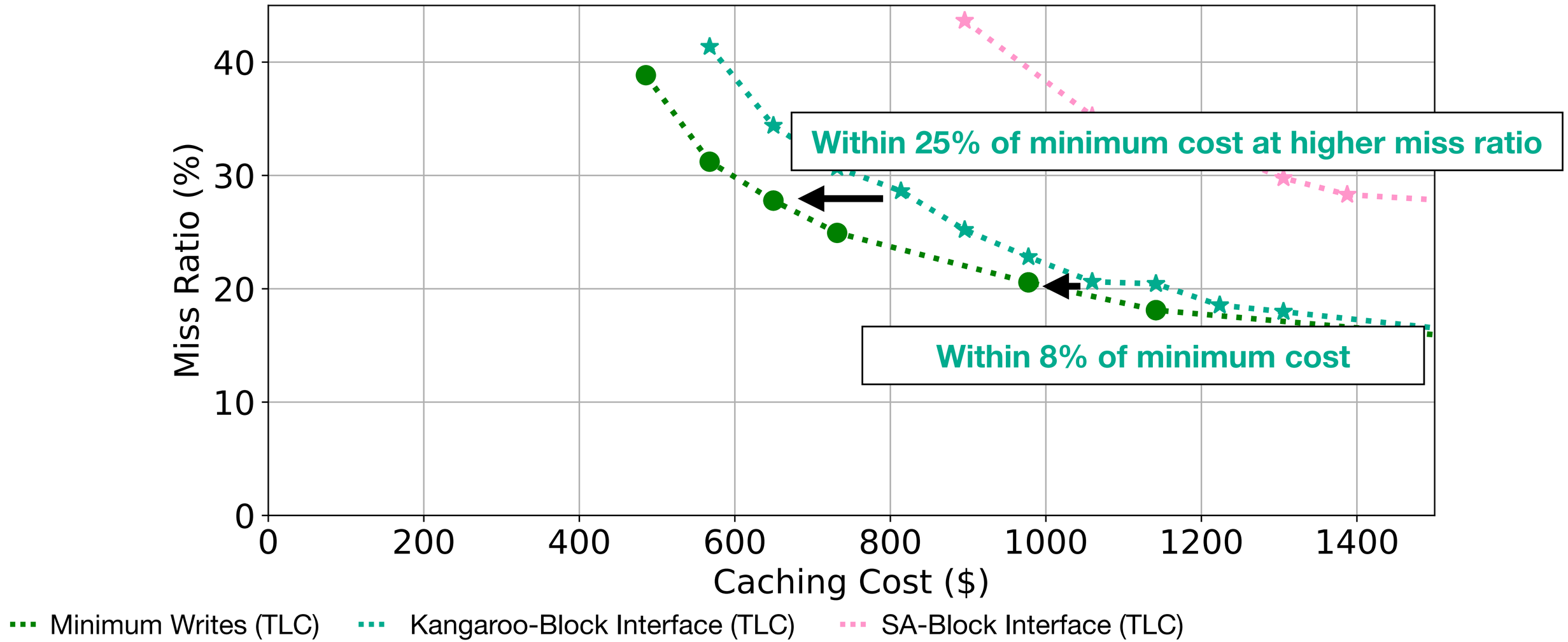
Minimizing cost for a given miss ratio under two main constraints:

- Maximum usable **flash capacity**
- Average device write rate needs to be **below device limit** to avoid wear out

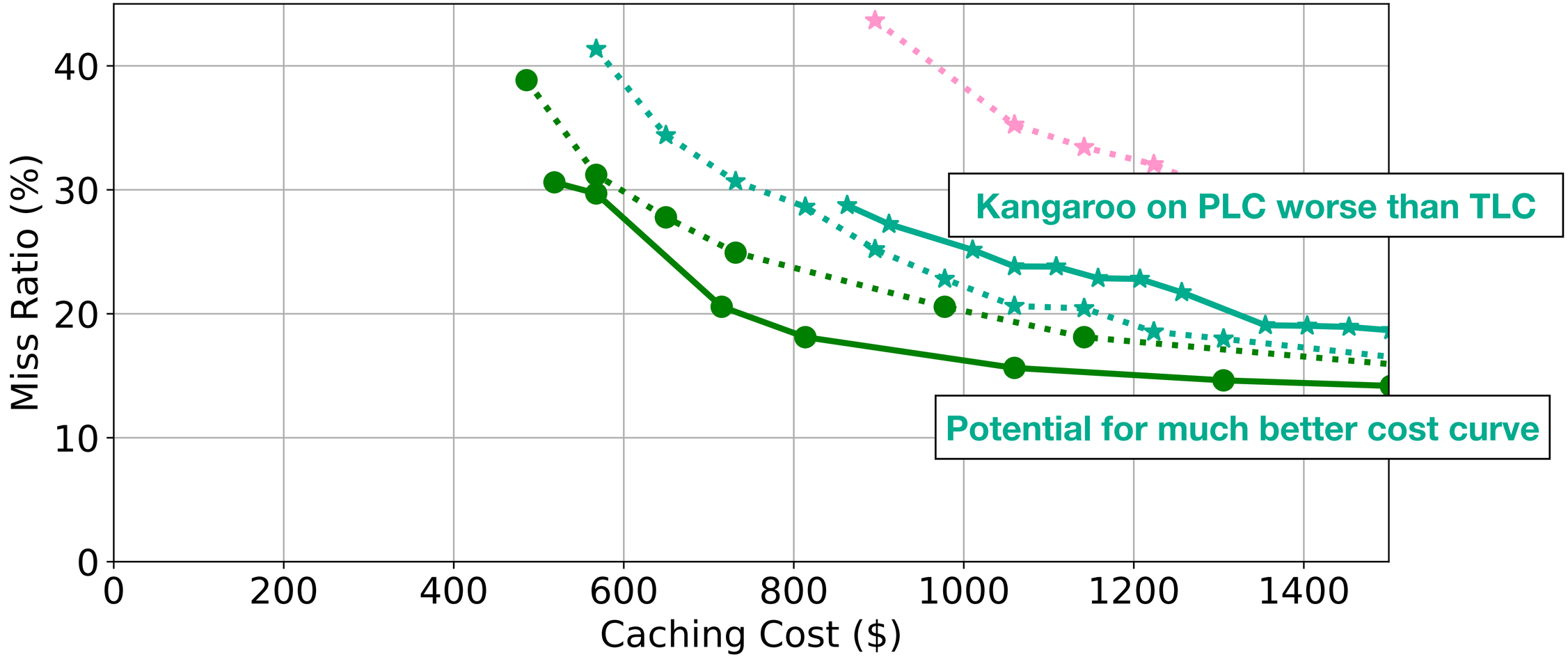
Kangaroo close to minimum for TLC



Kangaroo close to minimum for TLC



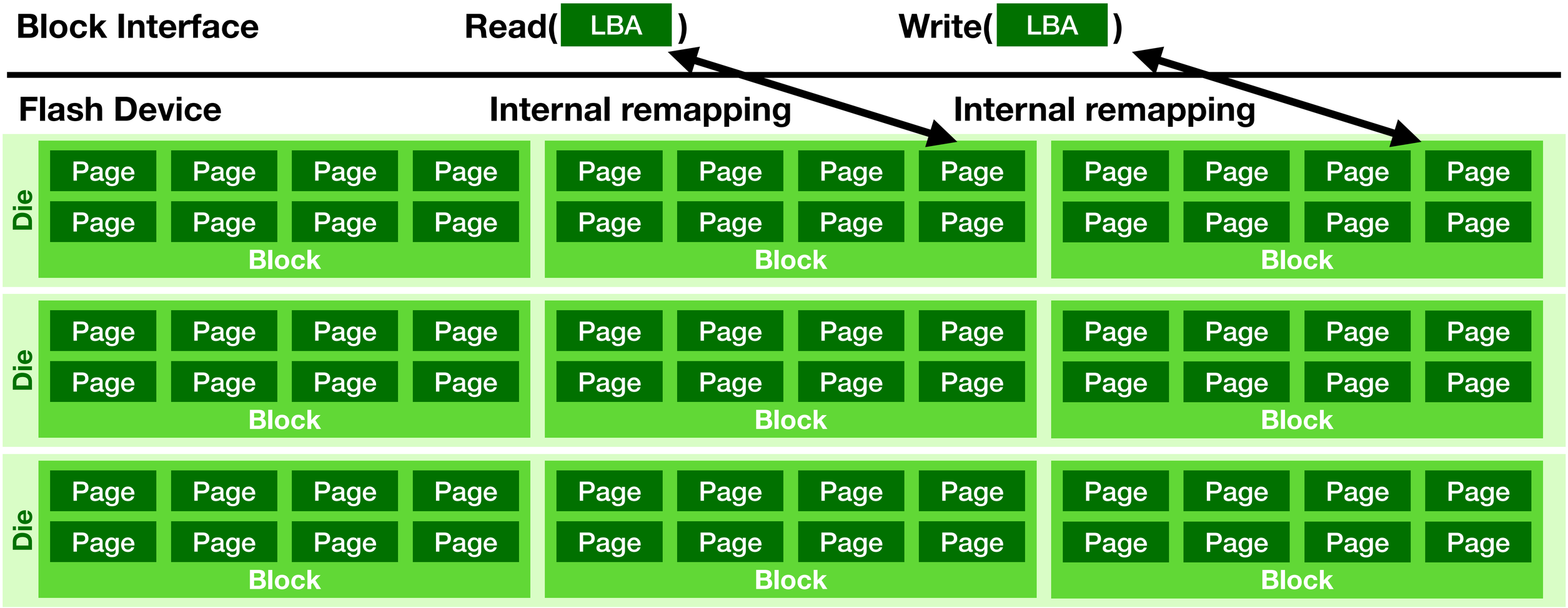
Kangaroo not enough for PLC



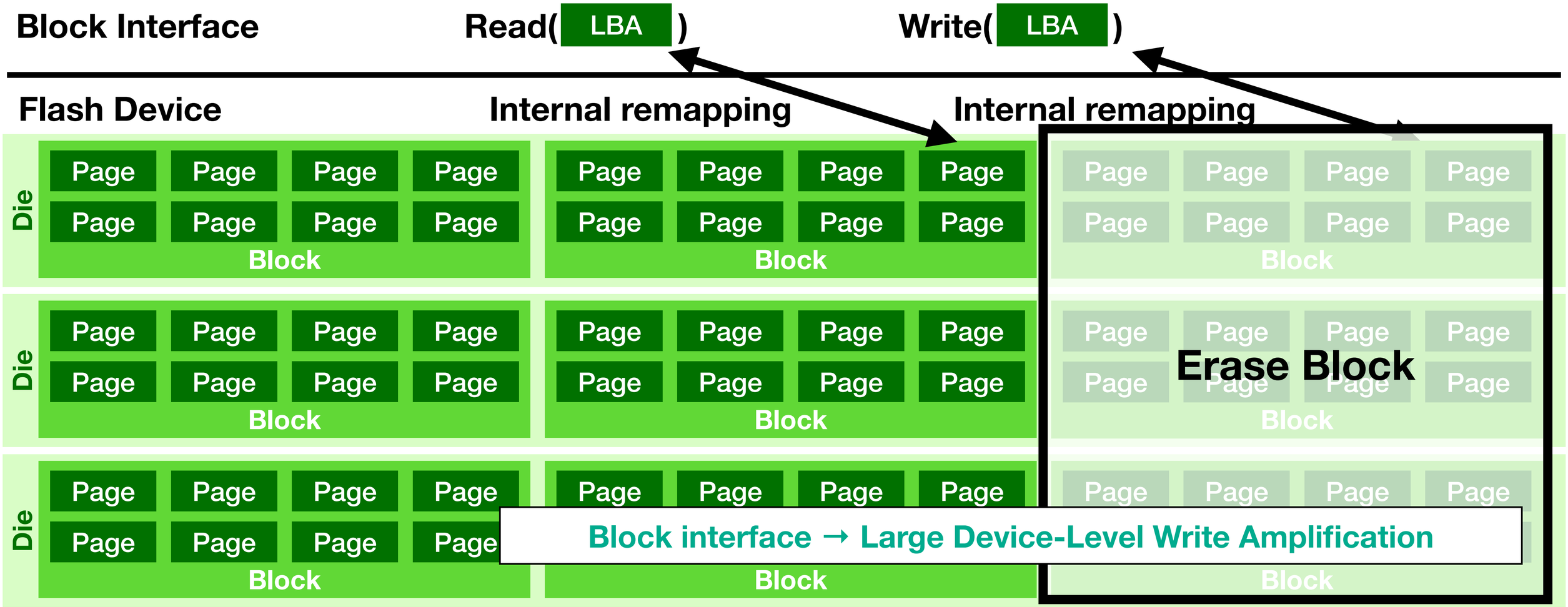
Talk Outline

1) Kangaroo [McAllister SOSP 2021]
   - Introduction
   - Prior work: Too many writes or too much DRAM overhead
   - Kangaroo design
   - Results
2) Caching on new flash interfaces
   - Kangaroo isn’t enough for new generations of flash
   - New flash interfaces

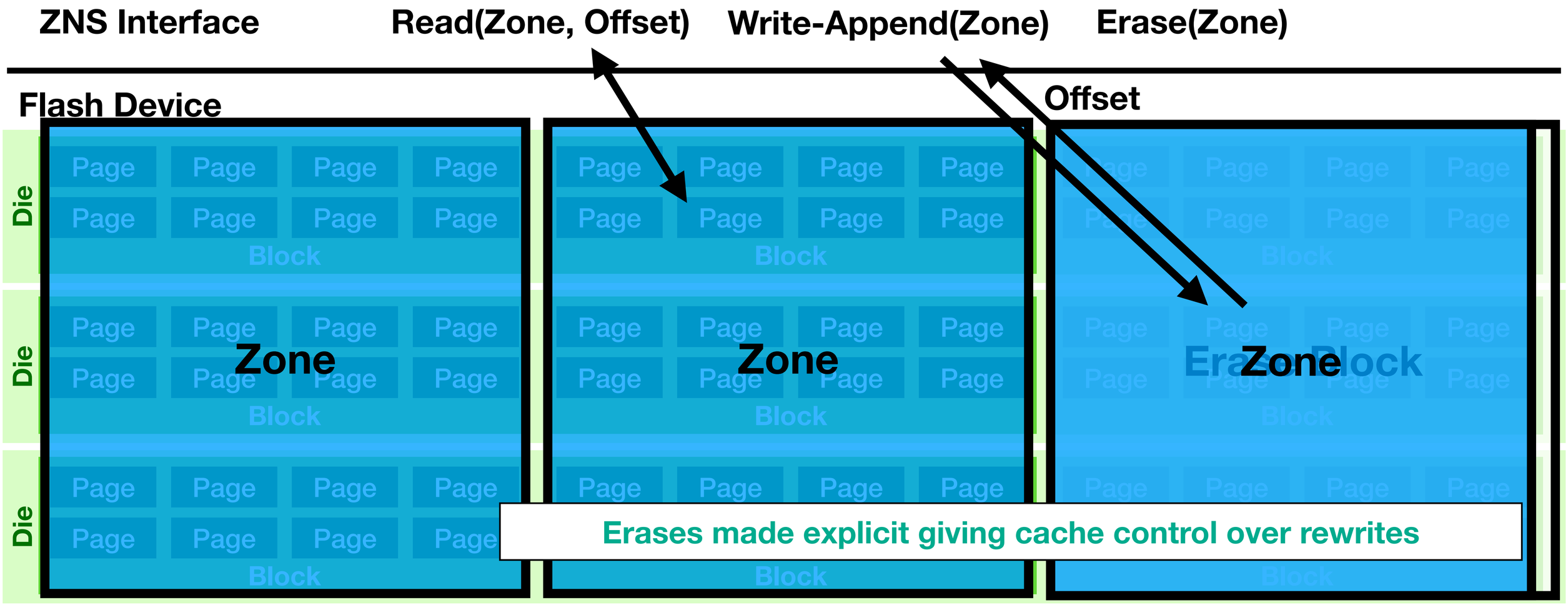
Block interface → bad flash abstraction



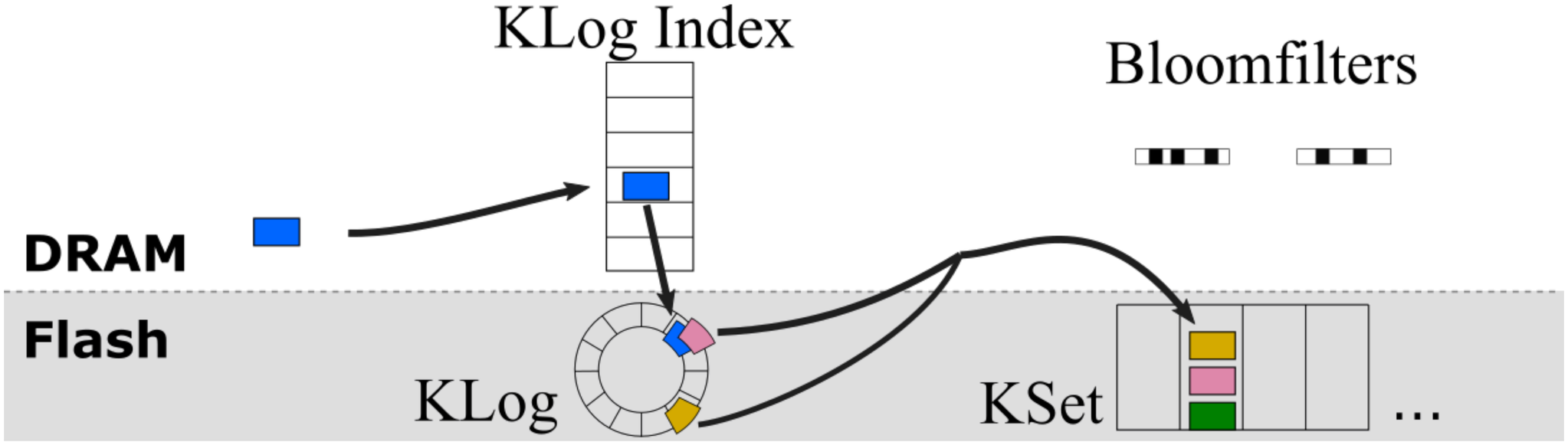
Block interface → Bad flash abstraction



Zoned Namespaces (ZNS) →Closer to device



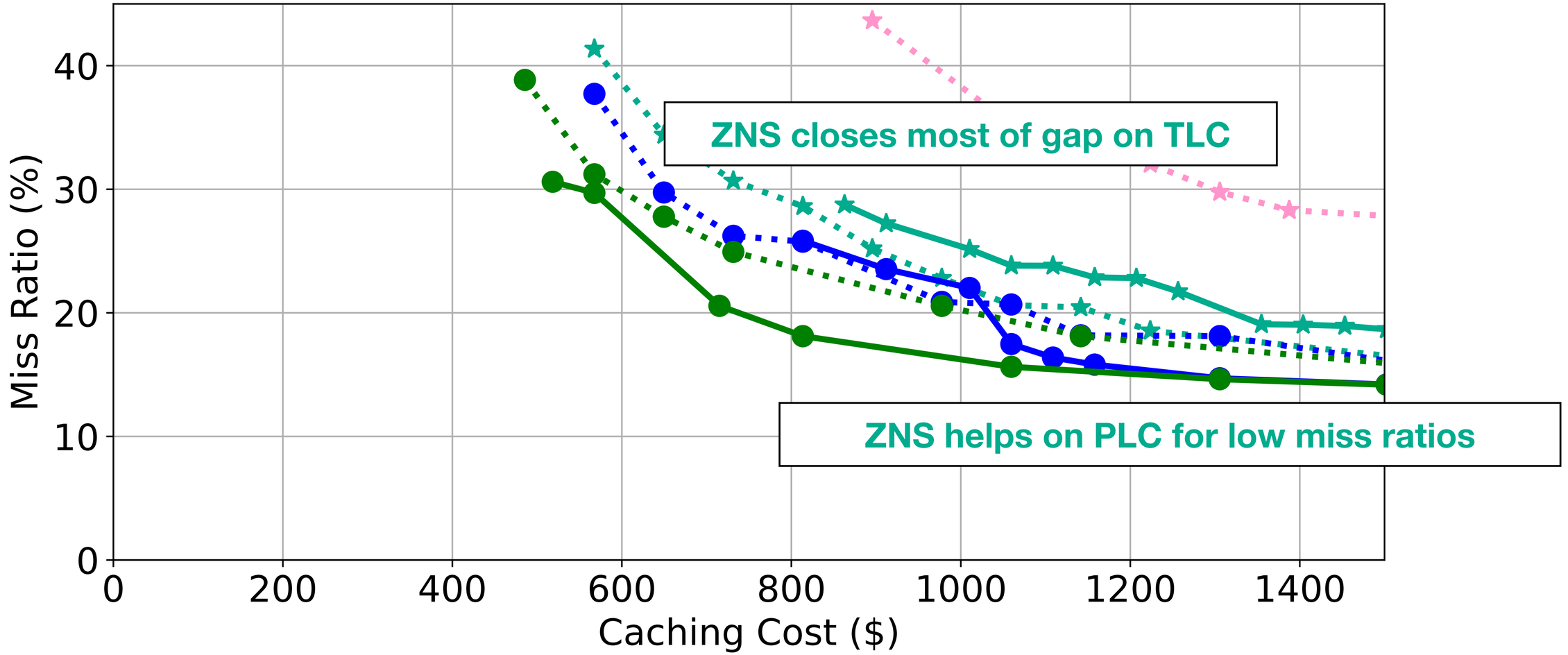
Revisiting Kangaroo with ZNS



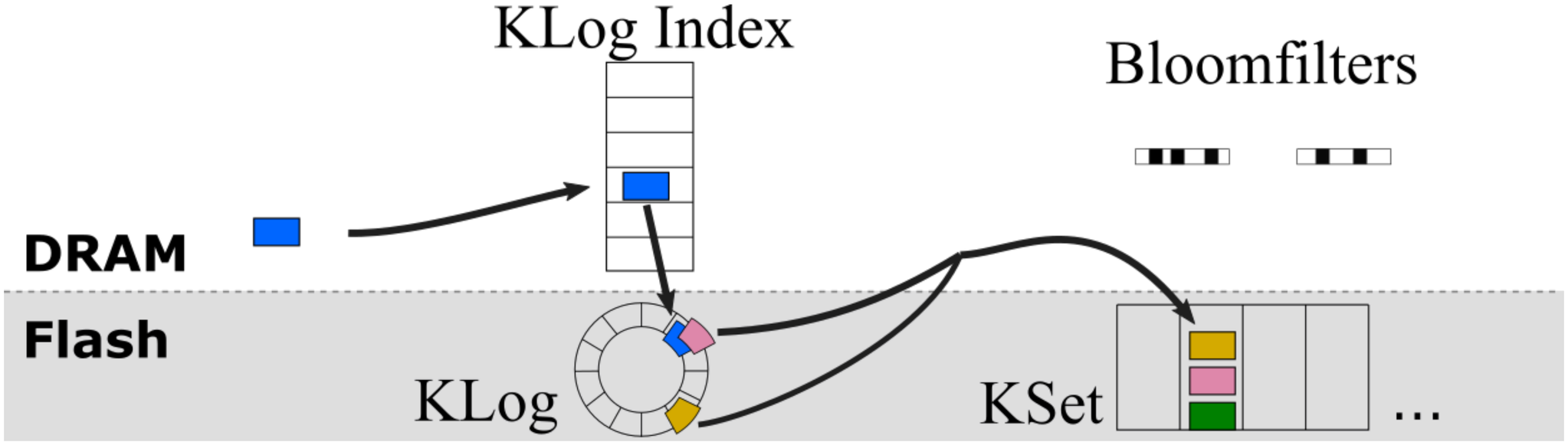
Opportunity: Combine Kangaroo eviction with ZNS

- Align KLog segments with erase blocks
- Flush to set in KSet if it has to be rewritten

ZNS starts to close the gap



Revisiting Kangaroo with ZNS



Opportunity: Combine Kangaroo eviction with ZNS

- Align KLog segments with erase blocks
- Flush to set in KSet if it has to be rewritten

Challenge: Set Capacity (4 KB - 64 KB) <<< Zone Capacity (1 - 8 GB)

Future Directions

SmartFTL interface out of Google

- allows **smaller writes** at the cost of device-level ECC

Further cache and erase co-design

- pair **admission policies** with future eviction decisions
- hotness-based object separation for small objects

Caching **billions of tiny objects** (~100 bytes) on flash

Prior Work → Too many flash writes or Large memory overhead →  **Wasted Money**

Kangaroo reduces misses by 29%

while keeping writes and memory under production constraints

McAllister SOSP 2021

Denser Flash → **Less flash writes**

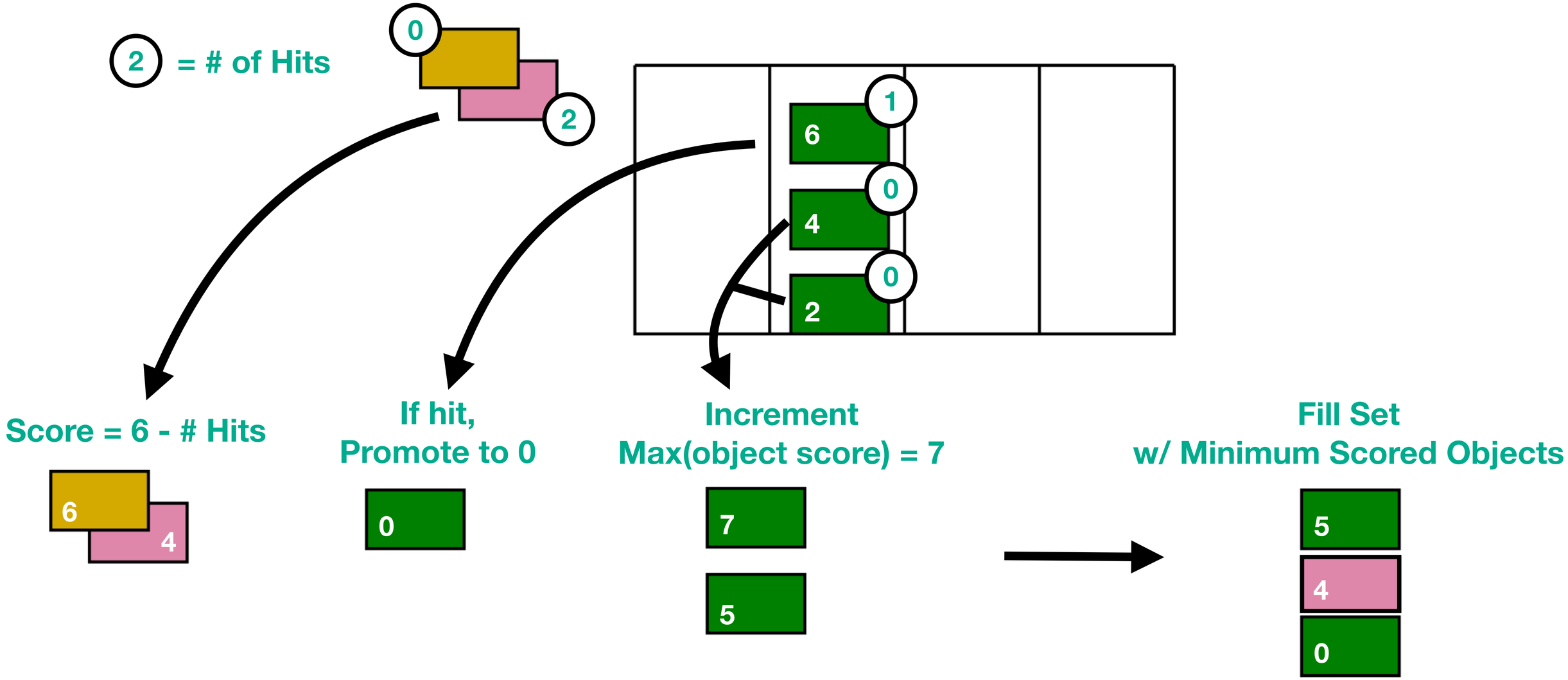
New flash interfaces needed use **denser flash**

Extra Slides

Kangaroo

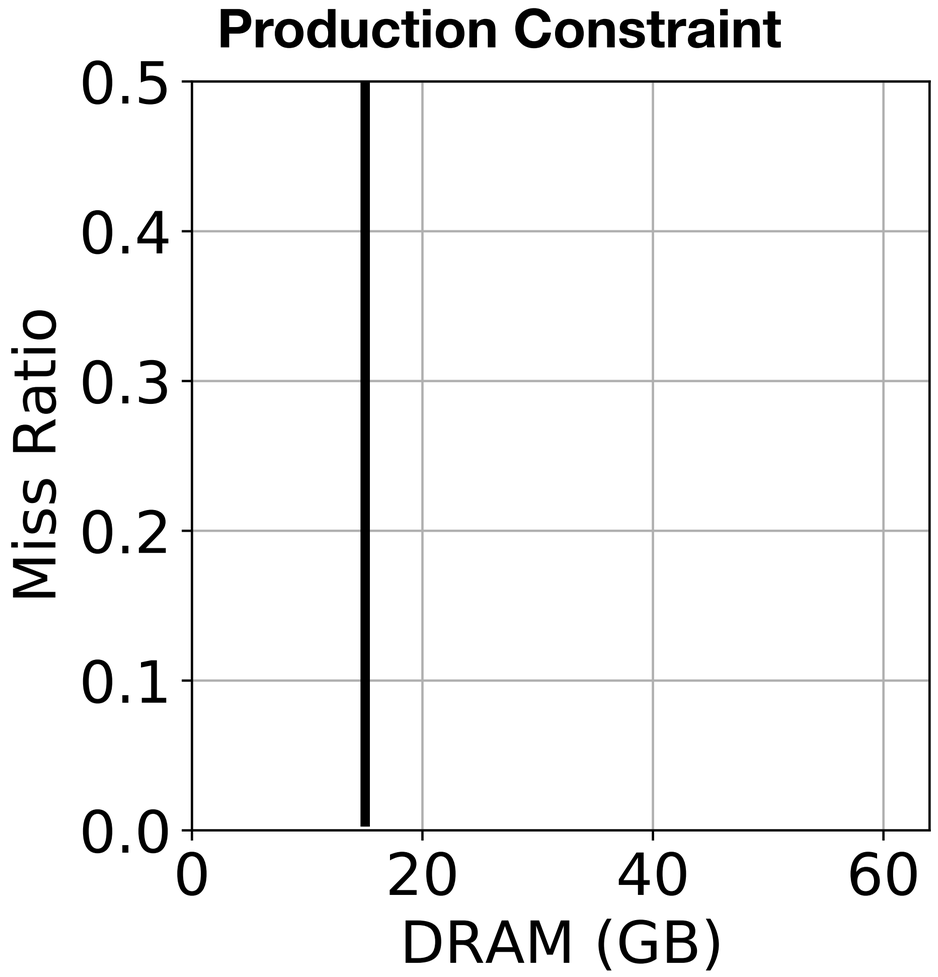
- Readmission to KLog
- RRIParoo
- Simulation experiments

RRIParoo eviction in KSet helps miss ratio



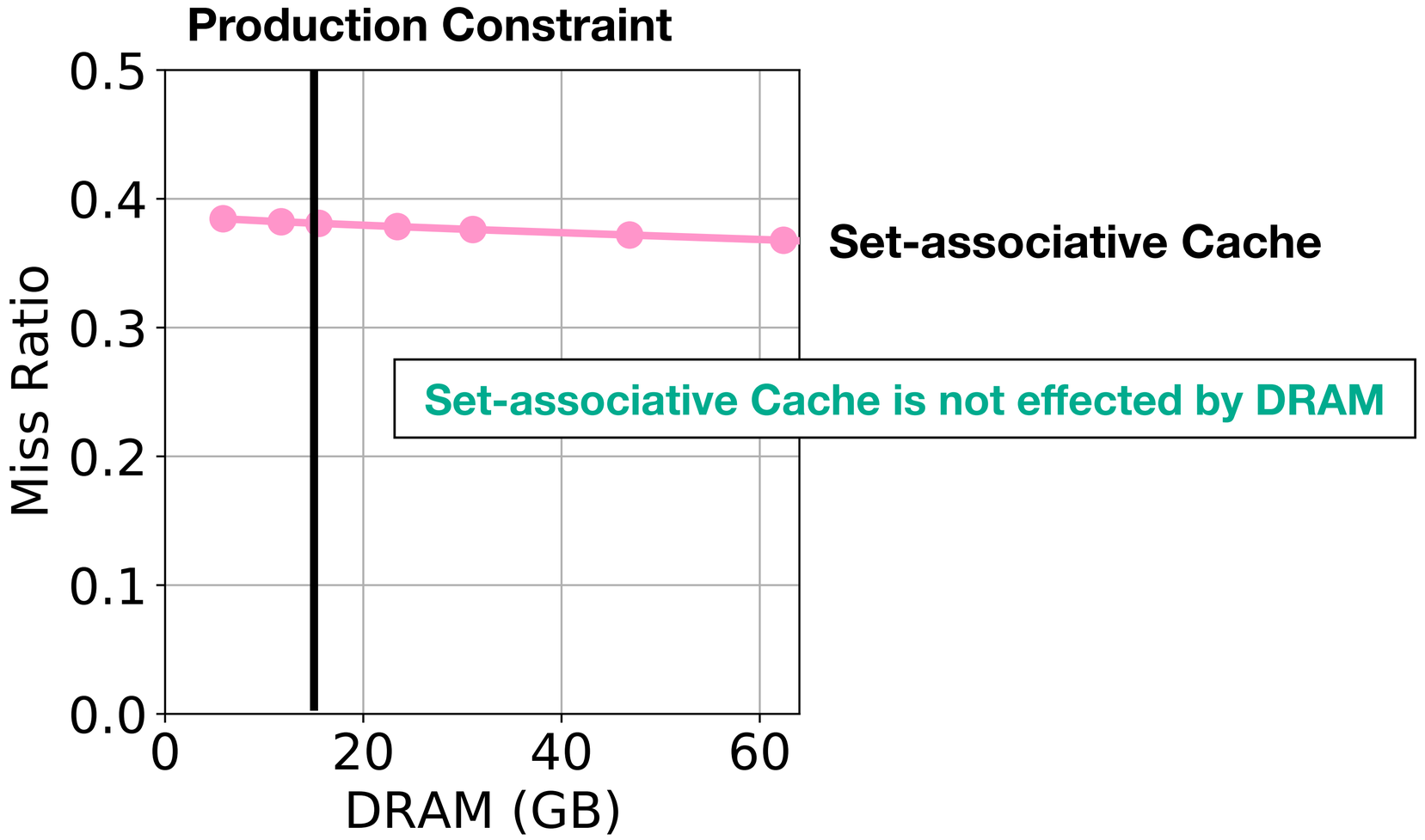
Varying DRAM budget

Simulating caches under different DRAM budgets on a 2 TB flash drive with 3 DWPD



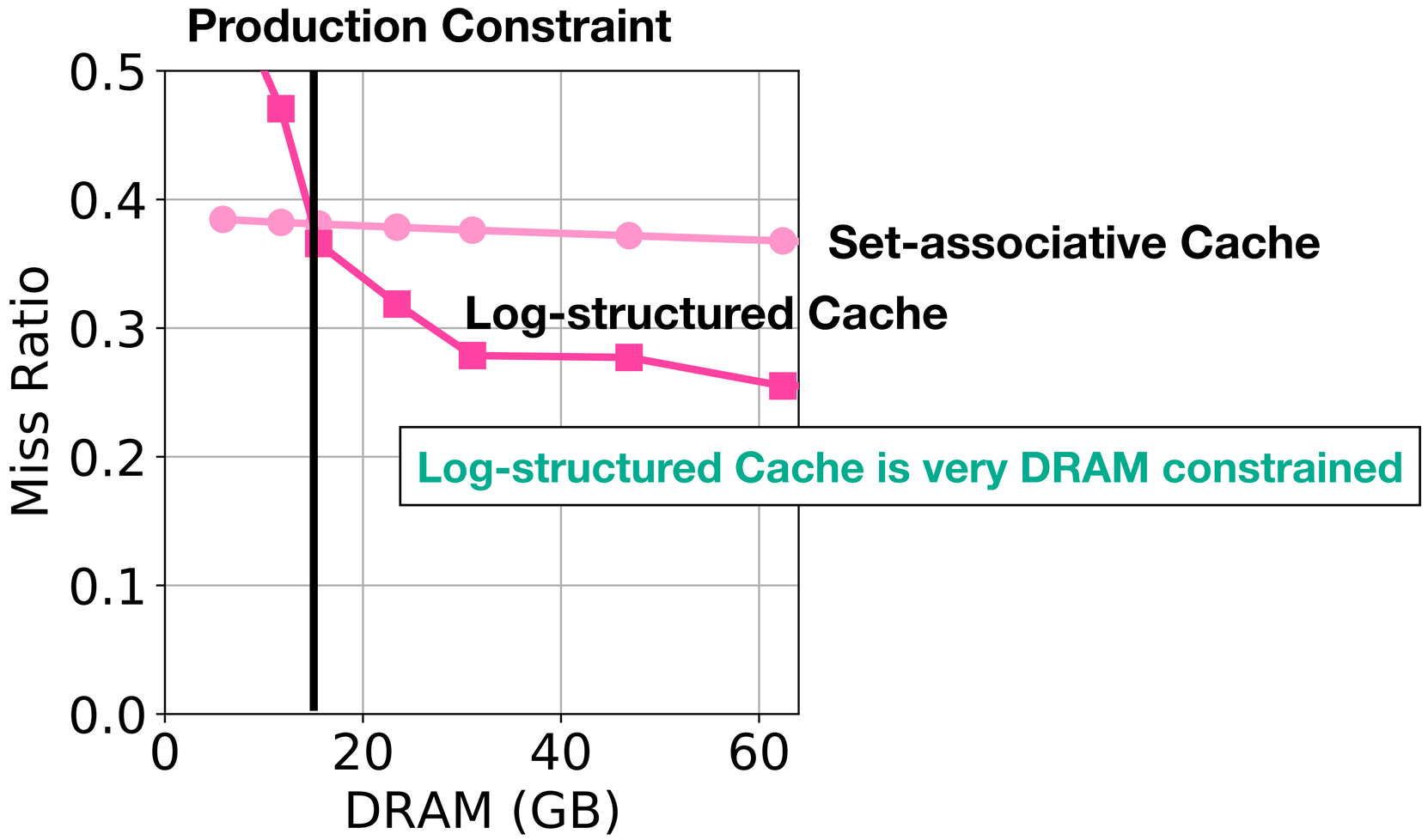
Varying DRAM budget

Simulating caches under different DRAM budgets on a 2 TB flash drive with 3 DWPD



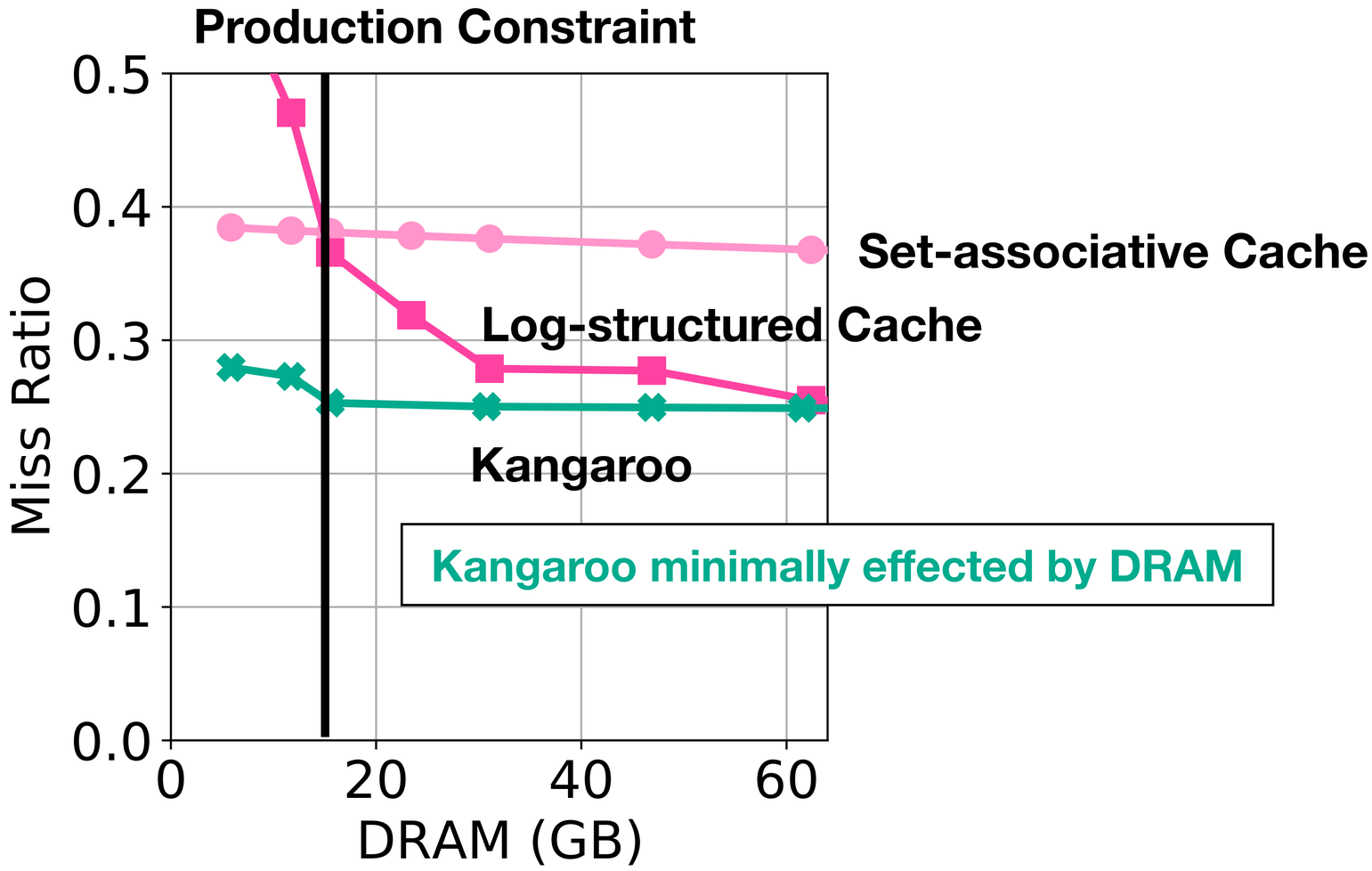
Varying DRAM budget

Simulating caches under different DRAM budgets on a 2 TB flash drive with 3 DWPD



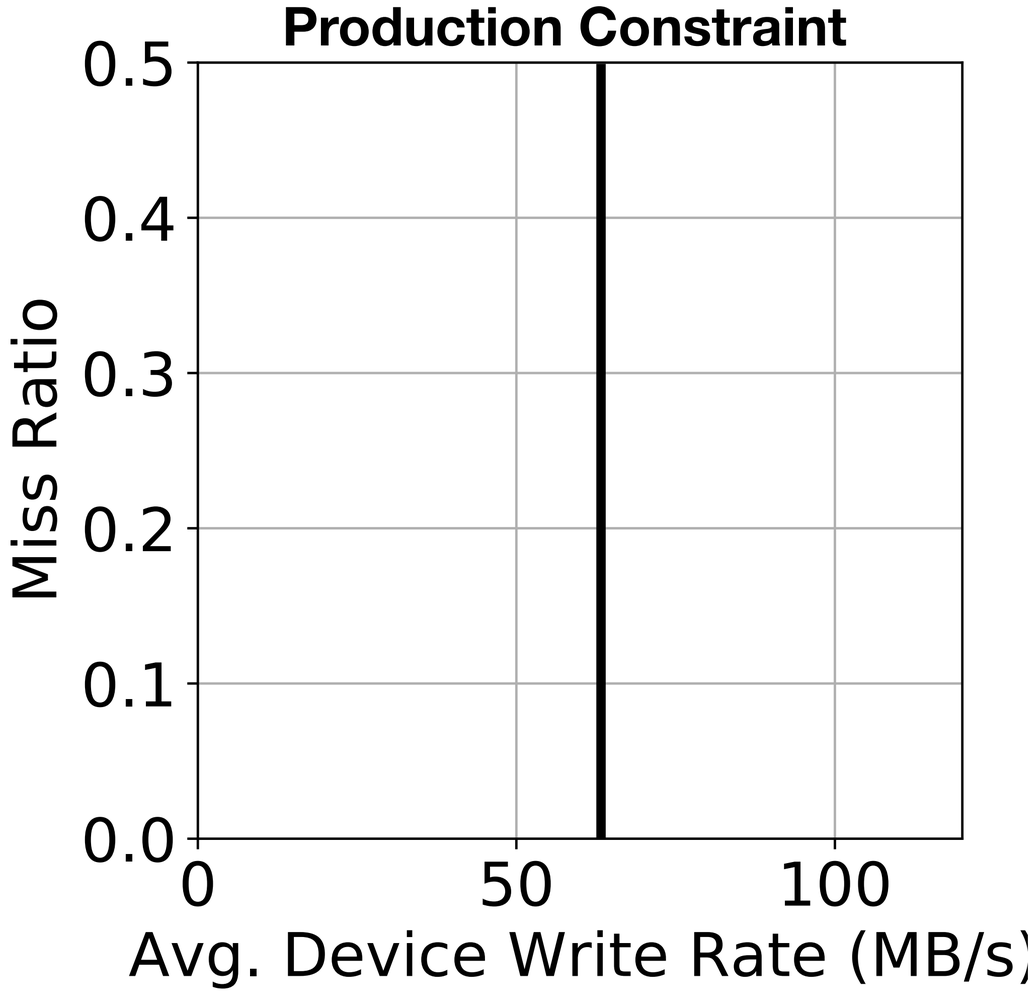
Varying DRAM budget

Simulating caches under different DRAM budgets on a 2 TB flash drive with 3 DWPD



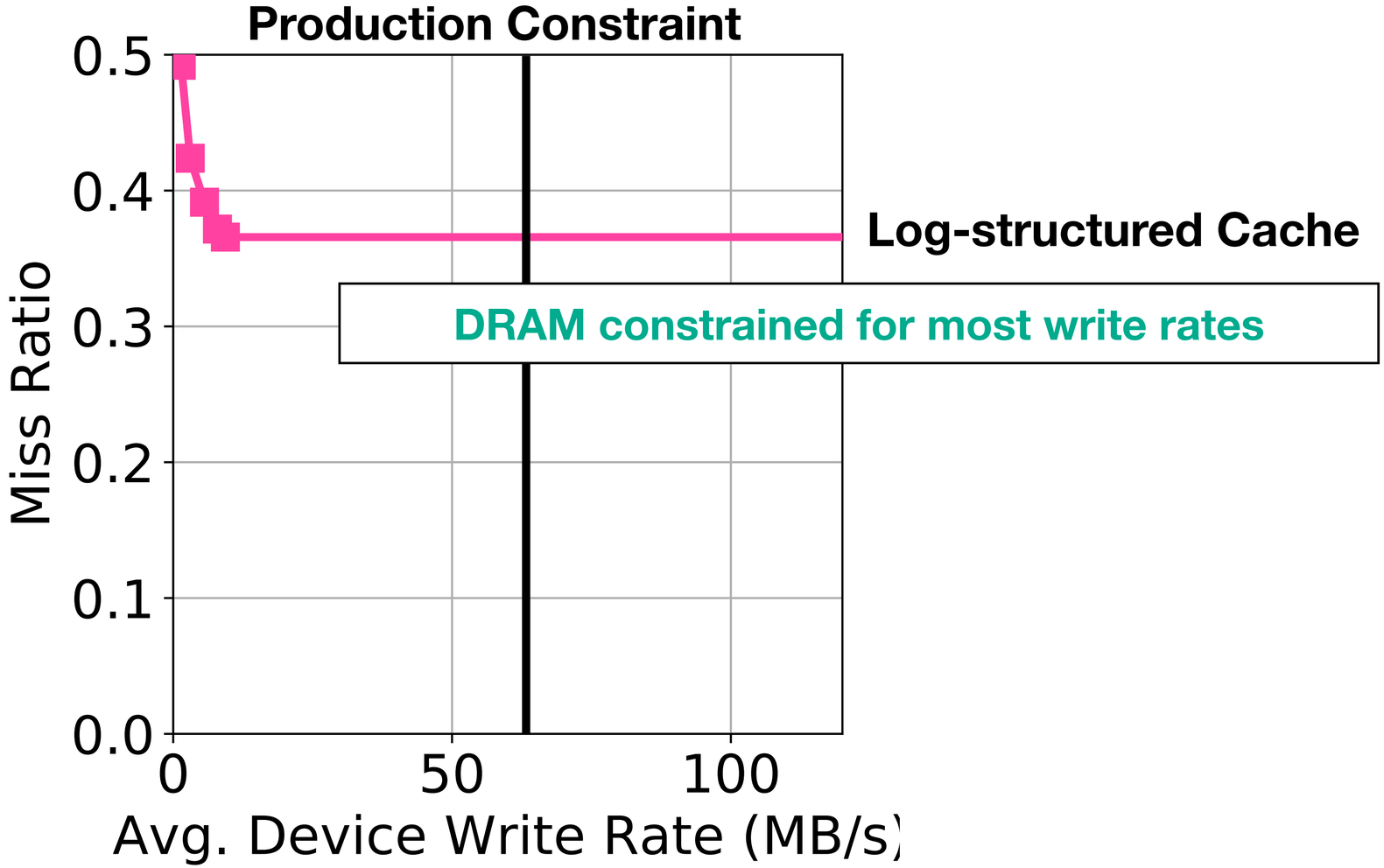
Varying write rate

Simulating caches under different write rate on a 2 TB flash drive with 16 GB memory



Varying write budget

Simulating caches under different write budgets on a 2 TB flash drive with 16 GB memory



Varying write budget

Simulating caches under different write budgets on a 2 TB flash drive with 16 GB memory