# From DRAM to SSDs

Challenges with caching @ Facebook

## Agenda:

1. Introduce CacheLib & hybrid caching
2. Hybrid caching challenges
3. Call to action

Sathya Gunasekar
Software engineer

# Caching is important for social media

### Reuse

Users share and consume content through overlapping social circles

### Recency

Users interact with recently shared content

### Scale & Efficiency

Efficiency is important to scale products to billions of users

# Evolution of FB infrastructure

| | 2005 - 2010 | 2010 - 2015 | 2015 - today |
|---|---|---|---|
| **Users** | 10s of millions | 100s of million | billions |
| **Product** | likes, comments, profile | + timeline, messaging, videos, photos, location.. | facebook |
| **Infrastructure** | Memcached<br>MySQL | fb-Memcached, TAO<br>MySQL,<br>CDN, Haystack | hundreds of services |

# Diverse caching needs @ Facebook

| Application Domain | Architecture | Data format | HW |
|---|---|---|---|
| General purpose KV cache | Look-aside | Blob | DRAM |
| Social graph cache | Read & write through | Structured | SSD |
| CDN caches | Write back | Semi-structured | |
| Ranking/ML systems | | | |
| Developer infrastructure | | | |
| Analytics systems | | | |

## Should we generalize or specialize ?

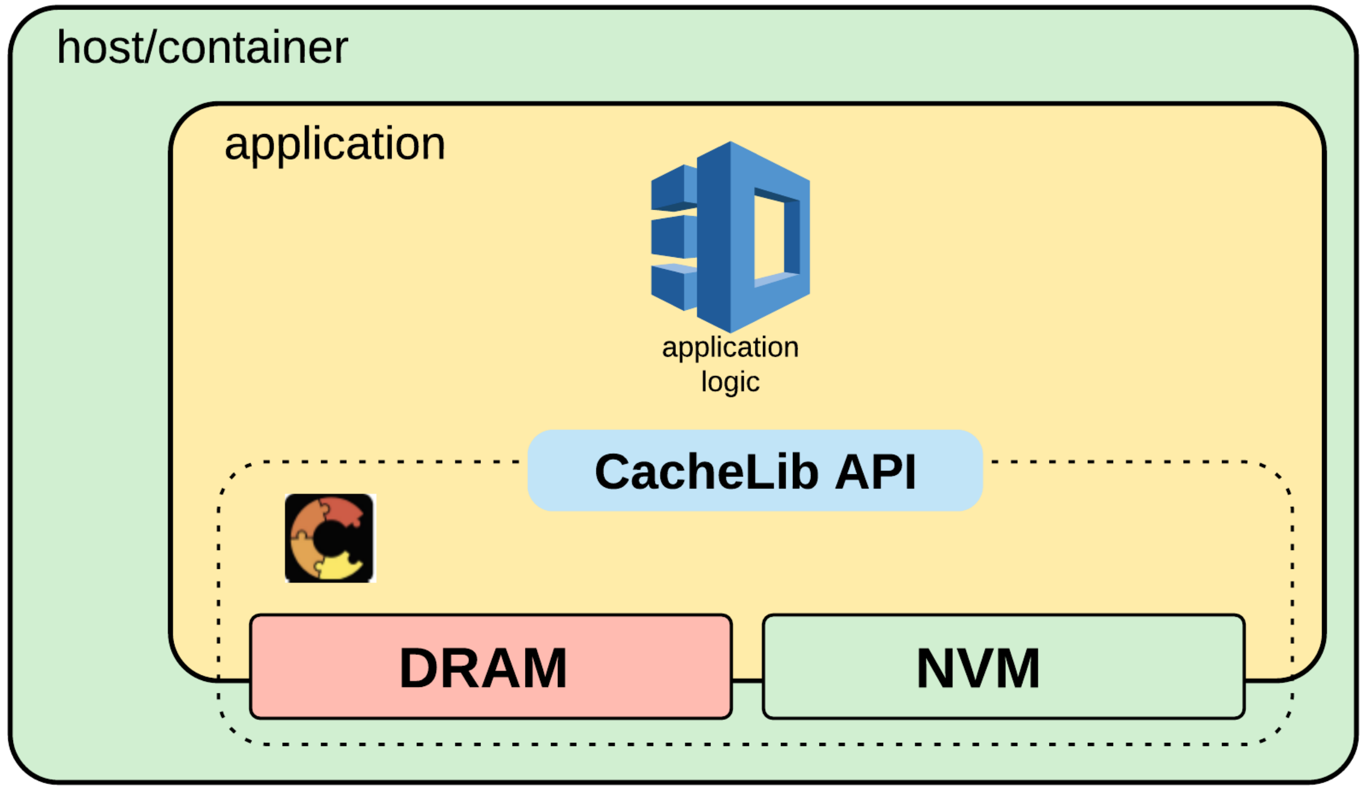
# CacheLib – Embedded C++ cache engine

## Aggregation point for innovation

### CacheLib solves:

1) What to cache ? - Heuristics
2) How to cache ? – Storage design

CacheBench – workload benchmarking tool



# CacheLib @ Facebook

**Online Data**
- Social graph
- Look aside KV Cache

**Traffic**
- CDN
- Proxygen

**Storage**
- HDD file storage
- HDD video storage

**AI/ML**
- Training
- Ranking

**Dev Infra**
- Data analytics
- Source control
- Stream processing

70+ services, XXX K+ instances, XX PB DRAM, XXX PB NVM

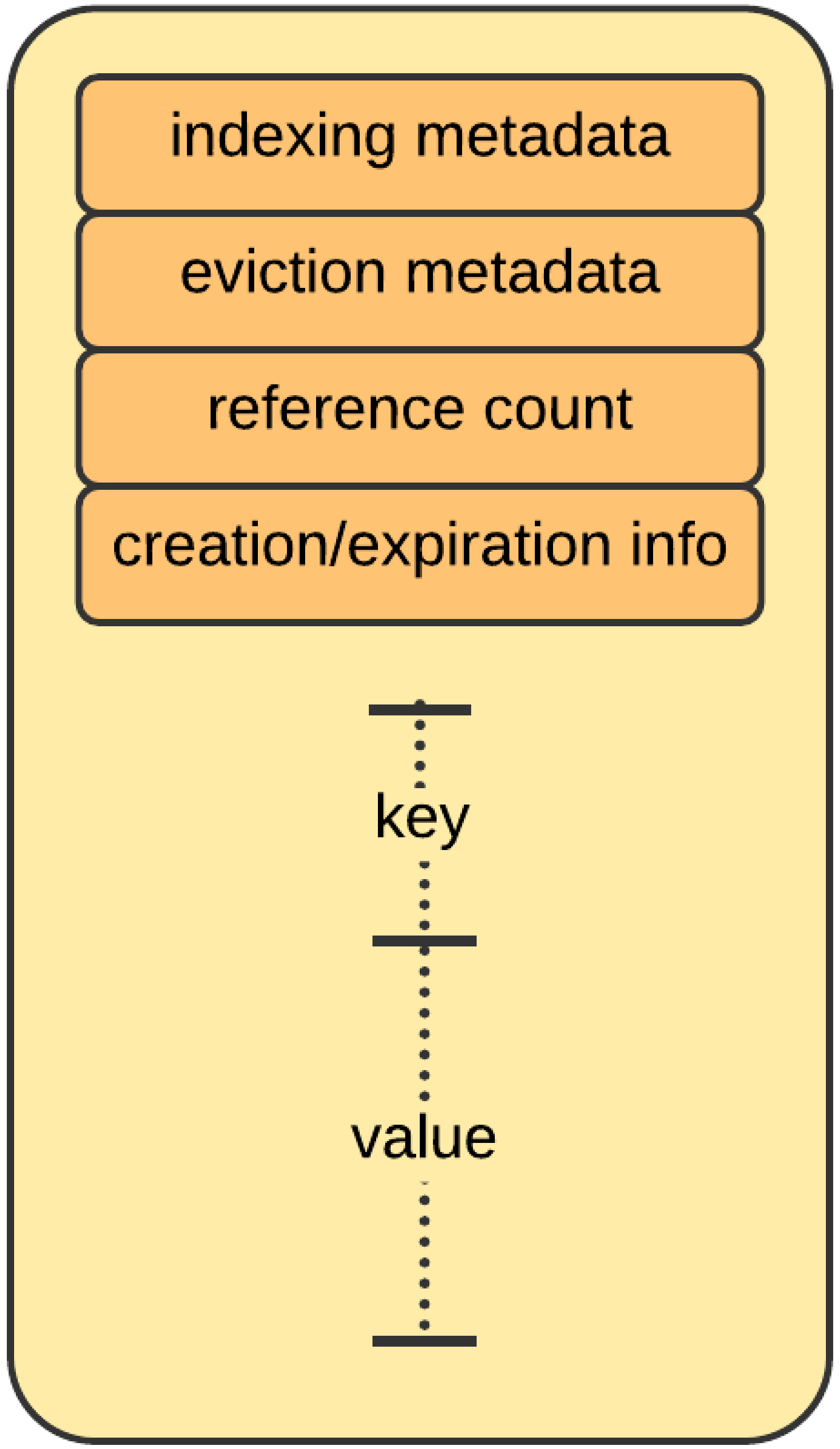
## Open sourced by Facebook

www.cachelib.org

# CacheLib: Item

Unit of object managed in the cache

- Sequence of bytes (<4MB) identified by a key (<255 bytes)
- Exposed through an ItemHandle (shared_ptr<Item>)



# CacheLib: API

ItemHandle allocate(key, size, ttl)

void* Item::getMemory( )

ItemHandle insertOrReplace(handle)

ItemHandle find(key)

bool remove(key)

Features

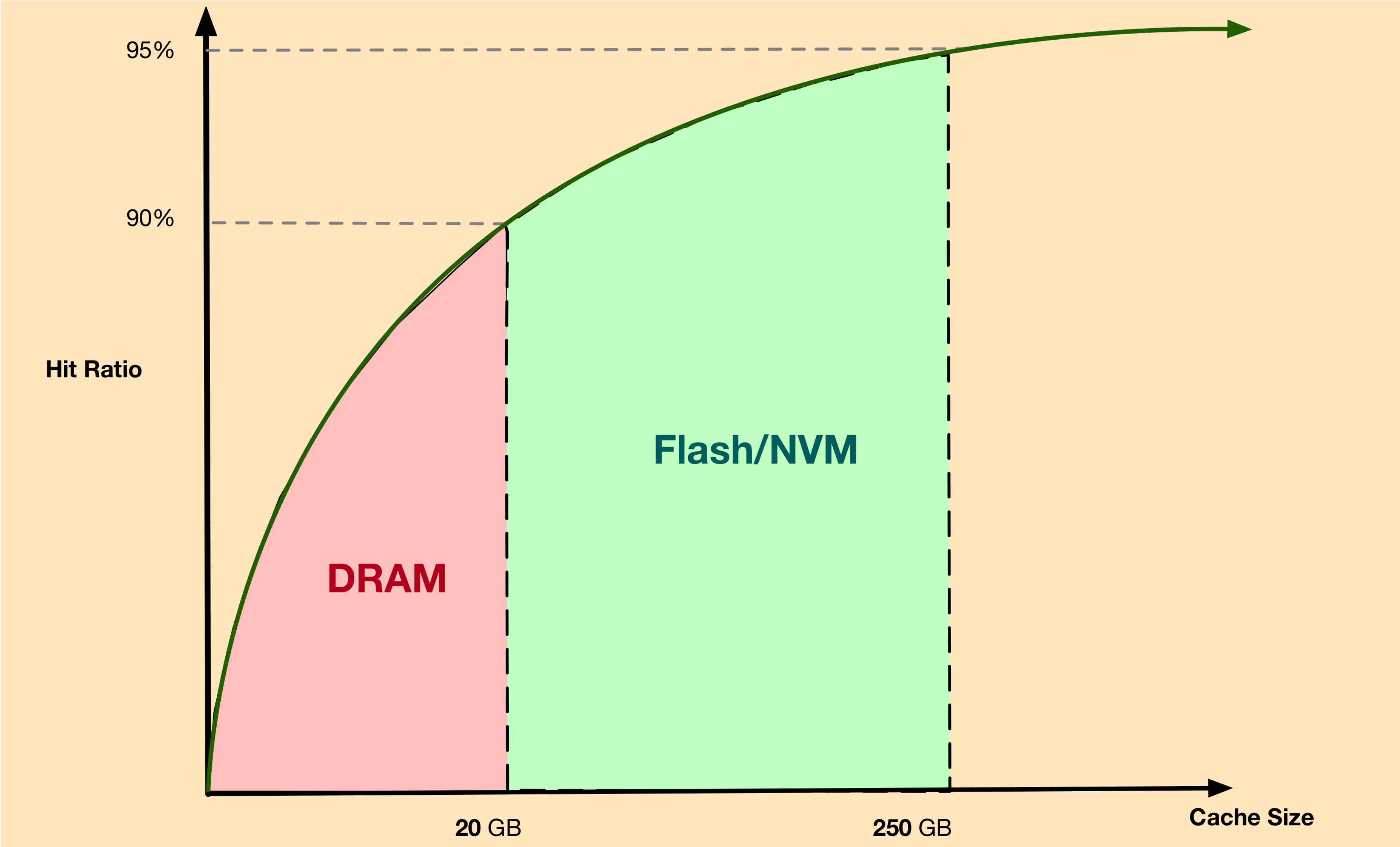
- variety of cache policies
- memory pools & isolation
- resource adaptiveness
- zero-copy access
- high concurrency
- structured data cache
- hybrid cache

# Hardware trade-offs

Explore cost and power tradeoffs within a cache system

- marginal cost of an additional hit goes up
  - Hot objects in DRAM
  - Warm objects in NVM

Cache workload



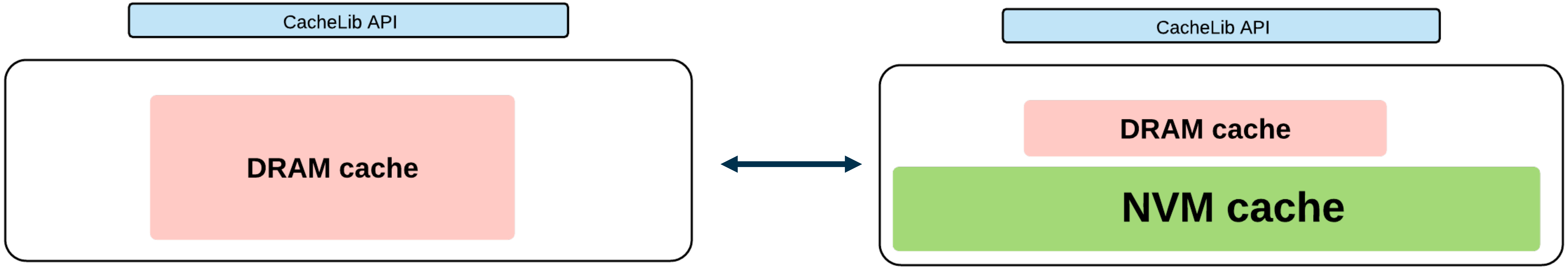
# Hybrid Cache goals

Fluid usage of storage mediums based on cost-sensitivity

- DRAM, NVM technologies etc.

Portability of caching applications

- Hide the complexity of hardware technologies behind the API



# Hybrid caches @ Facebook

Social graph

30GB DRAM,
1TB SSD

Look-aside key-value cache

40GB DRAM, 1TB
SSD

CDN

30–120 GB DRAM, 2–
8 TB SSD


HDD Storage

10GB DRAM, 400GB
SSD

# Hybrid Cache efficiency in practice

256GB DRAM -> 60GB DRAM + 2TB SSD

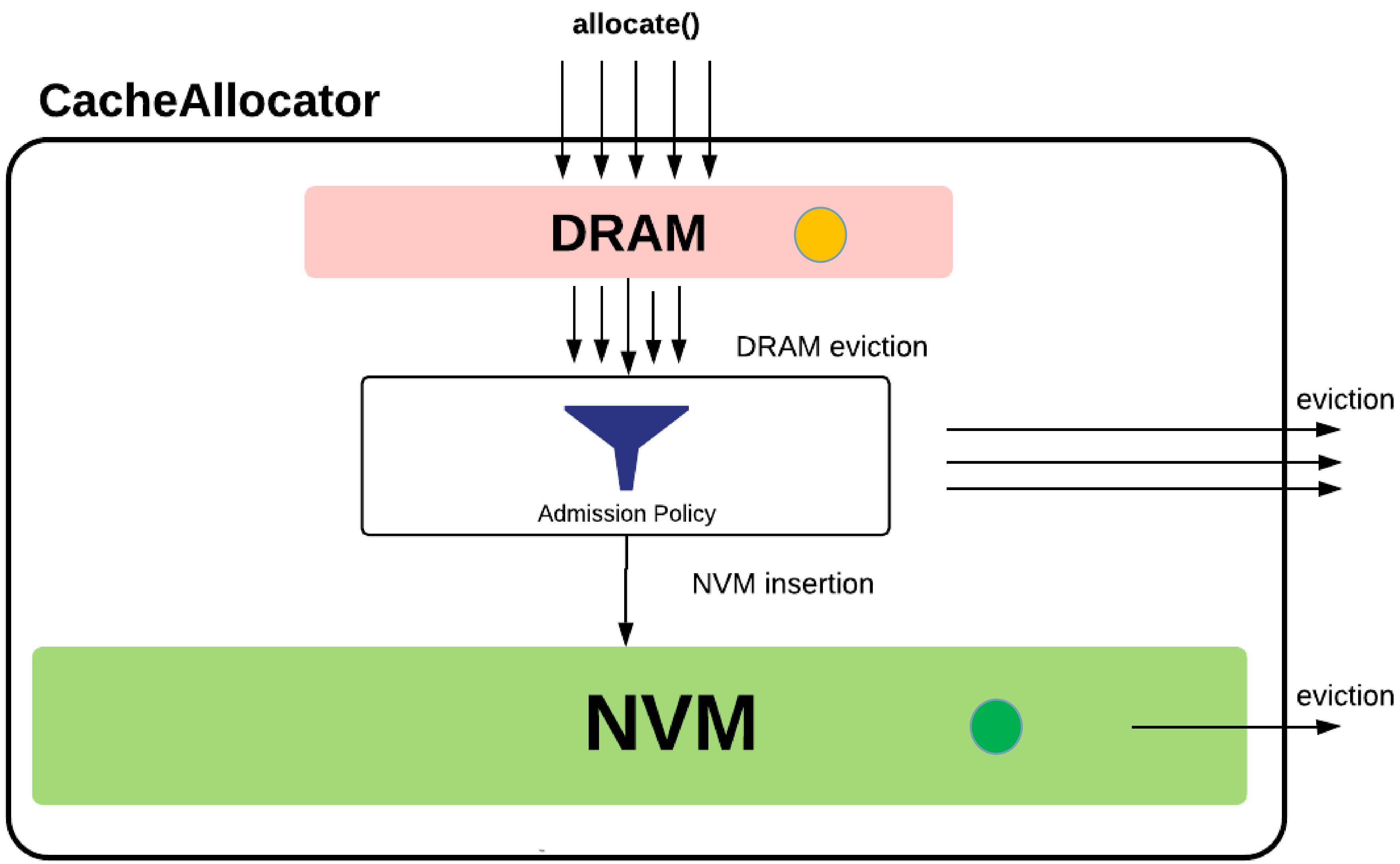
- 10x  more cache capacity
- 50% reduction in misses
- 25% reduction power and cost of ownership

# Hybrid cache: Challenges

| DRAM | NVM |
| --- | --- |
| mutable bytes | mutable blocks |
| small working set | large working set |
| infinite R/W BW | limited R/W BW |
| low & steady latency | latency is complicated |
| infinite endurance | limited endurance |

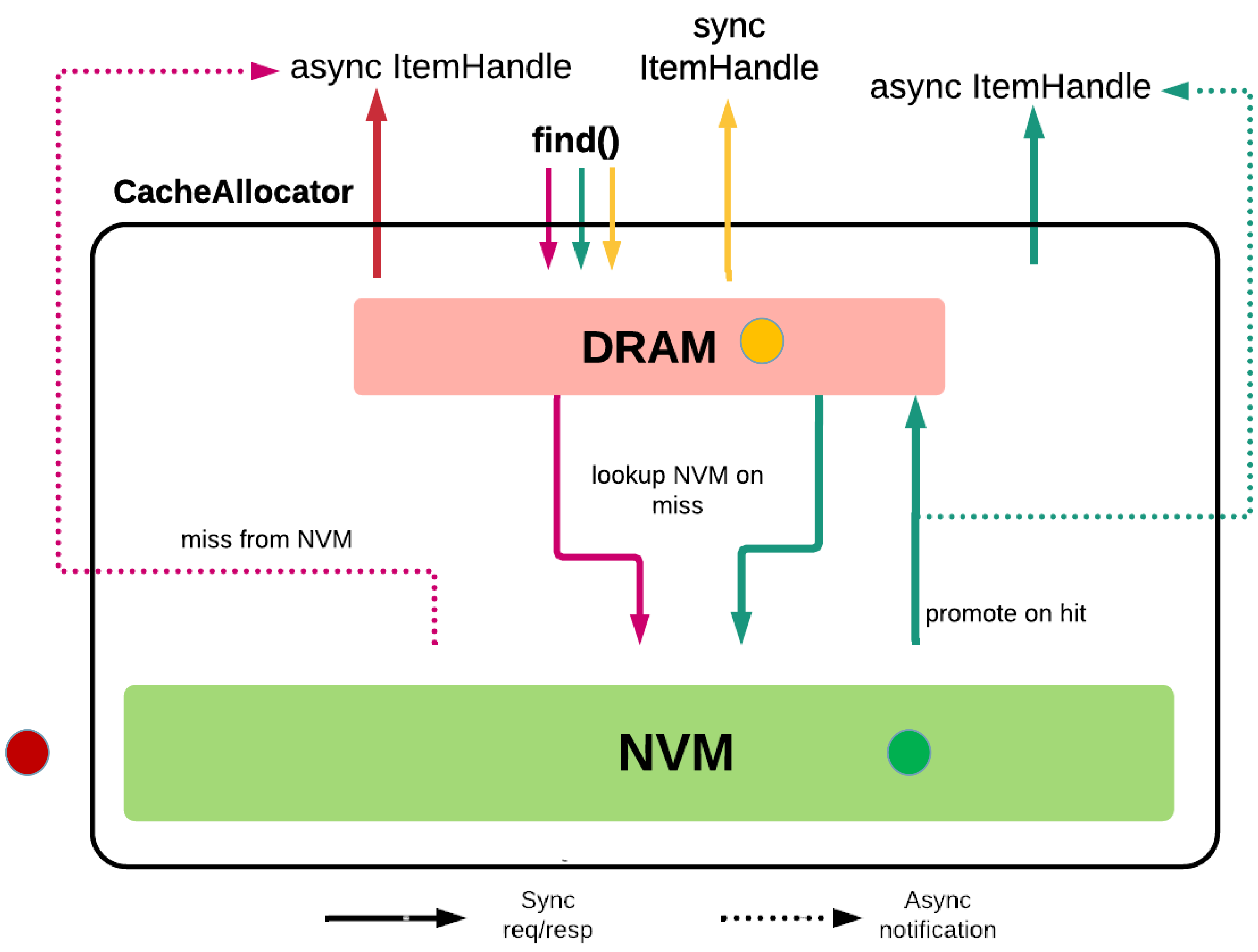
# Hybrid cache: Design

- Portability: Applications always see a DRAM cache
- Items are allocated in DRAM and then evicted to NVM
  - Evicted items can be rejected by NVM admission policy



# Latency: accessing Items through find()

- Portability: getMemory() blocks until Item is in DRAM
- Async interface ItemHandle
  - isReady( ) - is Item in DRAM ?
  - wait( ) - wait until Item is in DRAM
  - SemiFuture compatibility

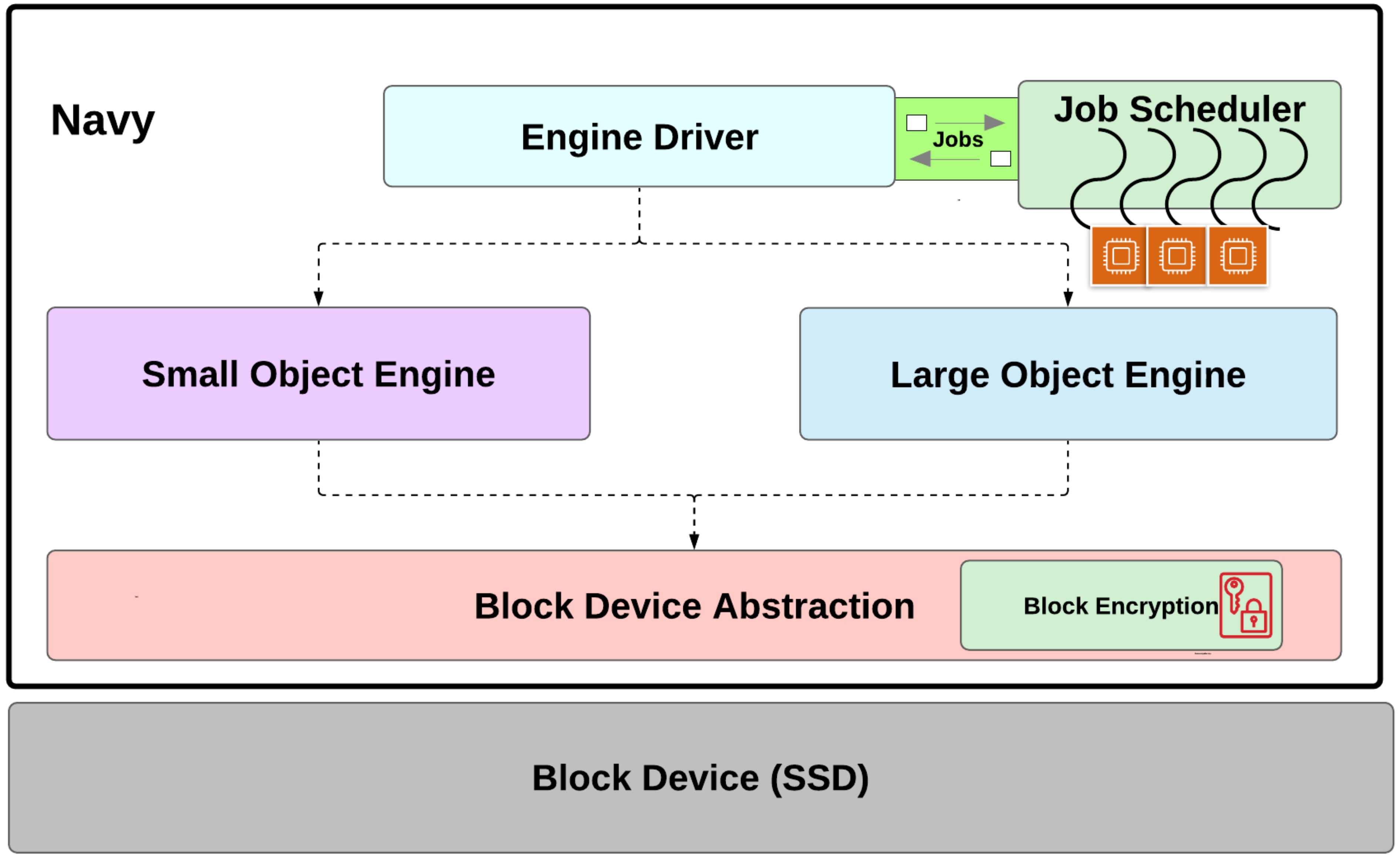


# Navy

## NVM storage engine

Goal: CPU & IO efficient point & negative lookups with low DRAM overhead

Optimized for caching tiny objects (100s of bytes)




Asynchronous IO

Optimized for caching large objects (>4KB)

File system/Block interfaces

# Small Item engine design

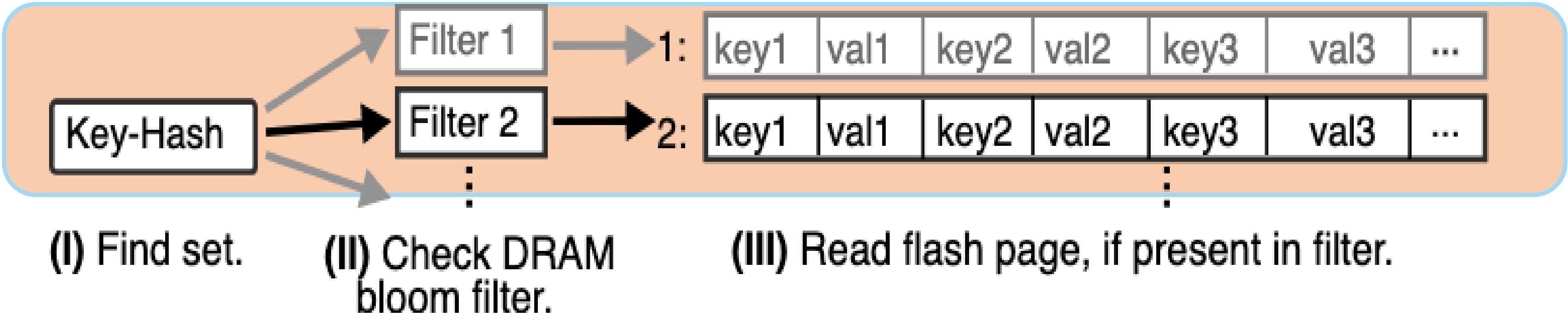
Accessing billions of objects per TB

- With single IO
- Low DRAM overhead

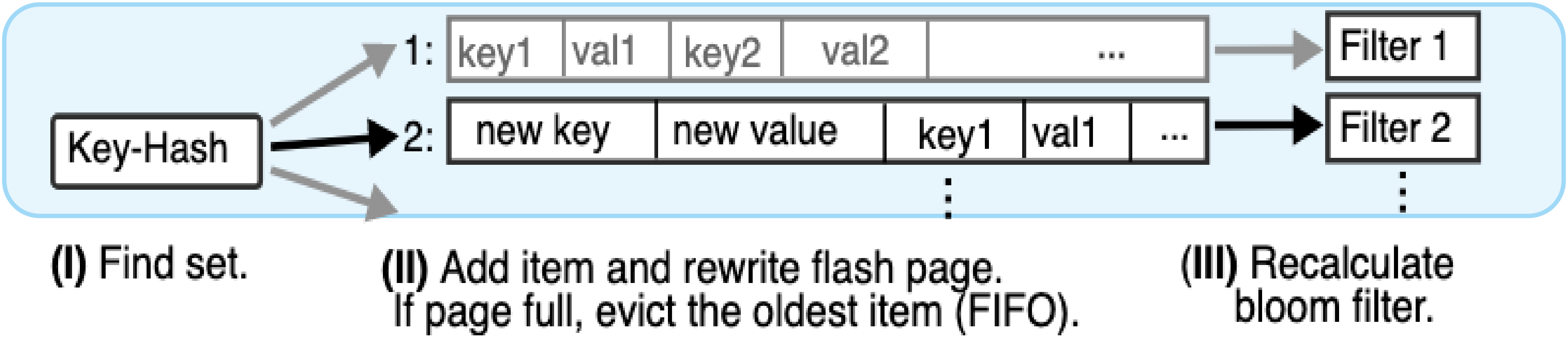
## Design

- Set-associative index (no DRAM)
- Optional DRAM bloom filters
- FIFO eviction (no DRAM)
- No space amp, but large write-amp

Reading an object



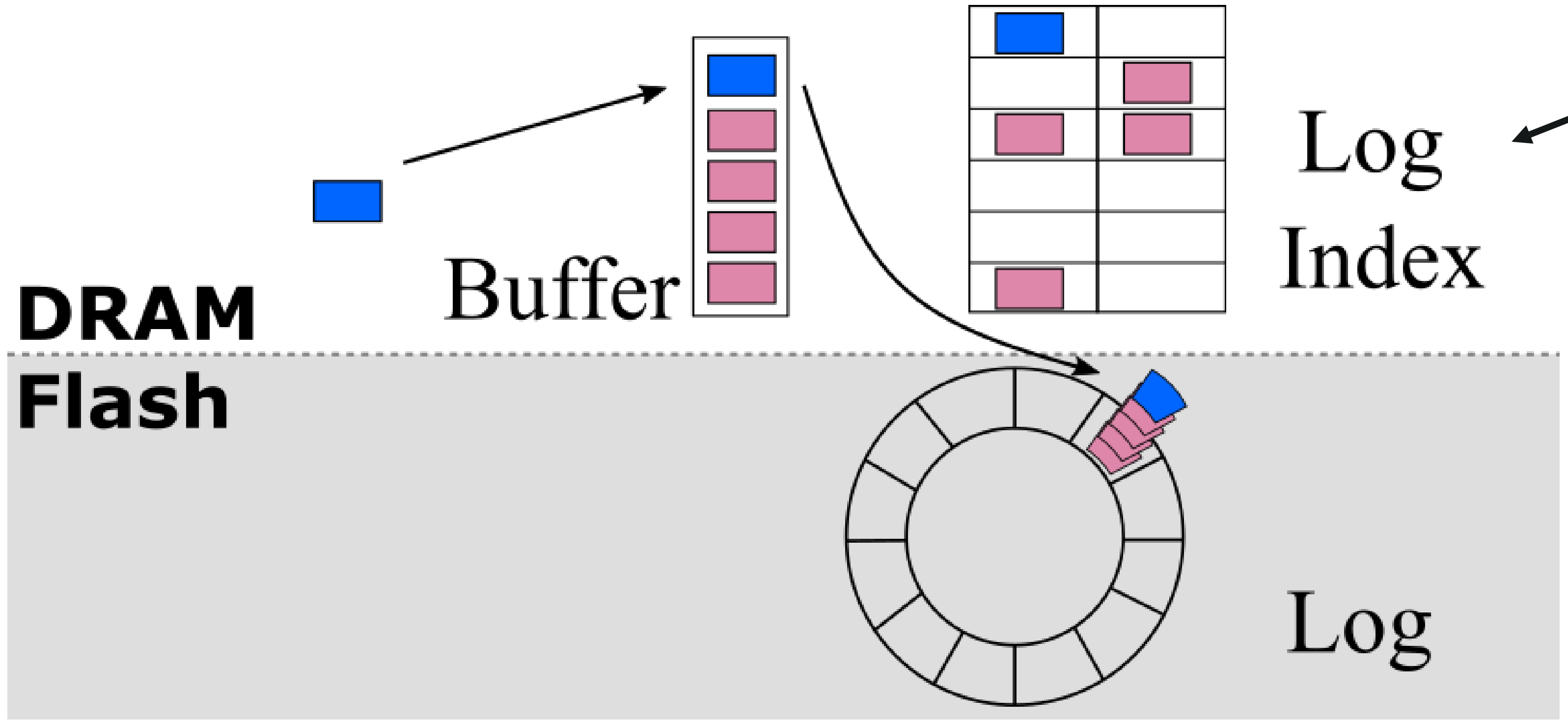

Writing an object




Due to endurance constraints, caches can’t write all the objects to SSD -> **poor miss ratios**

# Improving small object caching

## Log structured storage ?




To be compute and IO efficient for lookups, log structured cache needs a full DRAM index.

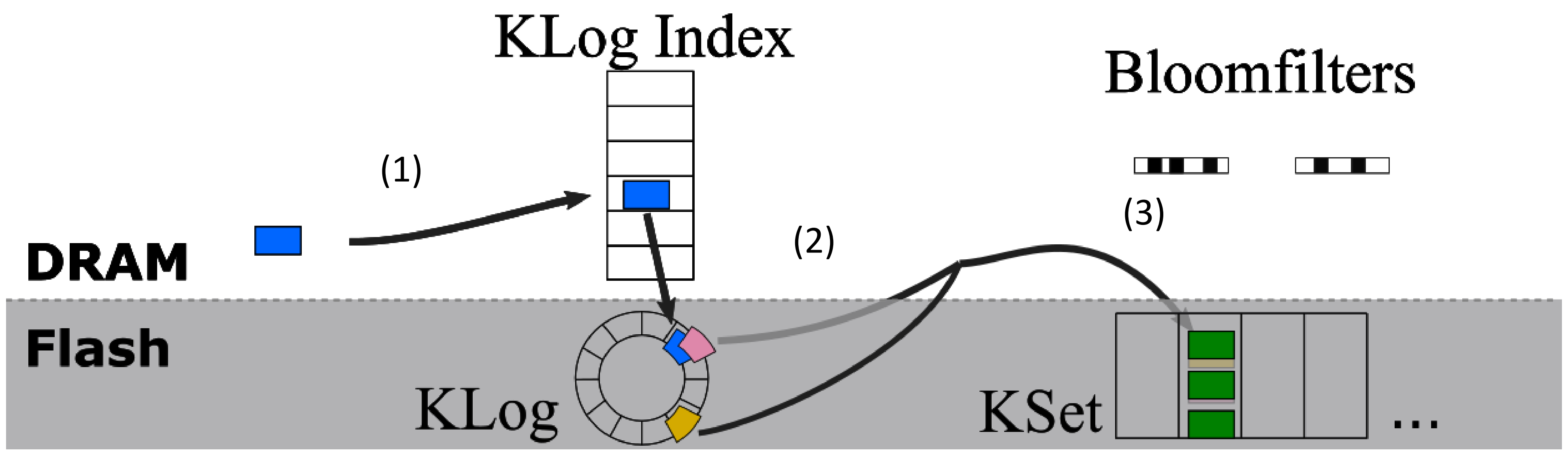
| | Set-Associative Cache | Log-Structured Cache |
|---|---|---|
| DRAM overhead | **Low** | **High** |
| Write amplification | **High** | **Low** |

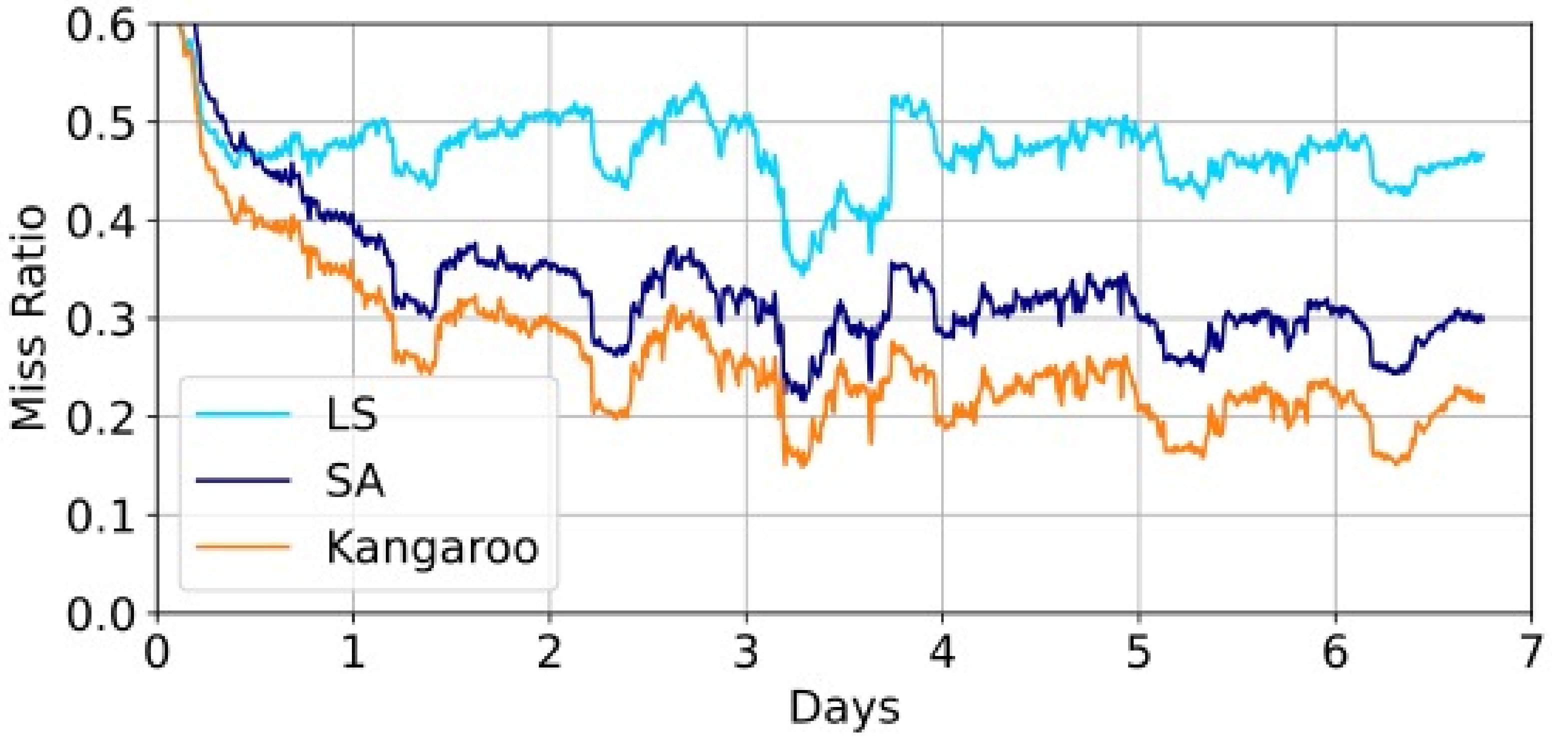
**Kangaroo: Realizing the best of both worlds**

Collaboration with Carnegie Mellon University, SOSP'21

# Kangaroo: SOSP'21

## Design




**Reducing the write amplification improves miss ratios**

1) Insert to Klog after buffering in DRAM

2)Periodically flush objects from KLog to KSet

3)Move all objects in KLog that map to the same bucket in KSet

# Summary: call to action

## Hybrid caching is gaining popularity, lots of new challenges

Caching small objects on block devices

- With lower DRAM overheads
- With lower endurance overheads
- With more IO efficiency

New hardware technologies on the horizon

need innovative solutions

collaborate among industry & academia

CacheLib offers an OSS platform to collaborate and explore solutions

www.cachelib.org

FACEBOOK