IIT Kanpur

SOSP' 21

Kangaroo: Caching Billions of Tiny Objects on Flash

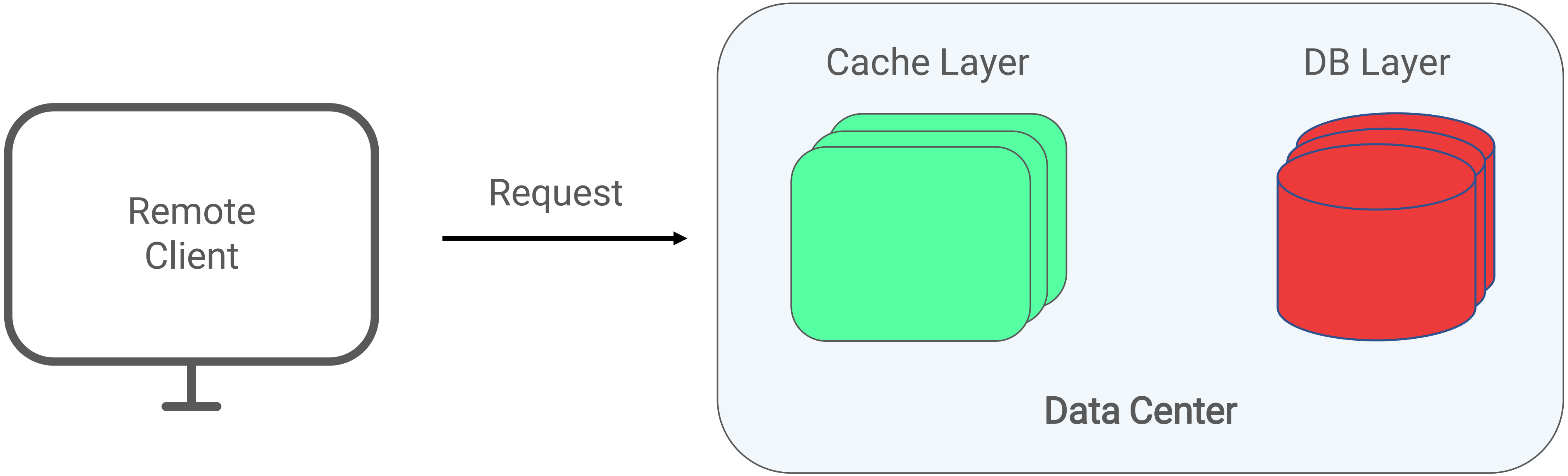
Sara McAllister, Benjamin Berg, Julian Tutuncu-Macias, Juncheng Yang, Sathya Gunasekar, Jimmy Lu, Daniel S. Berger, Nathan Beckmann, Gregory R. Ganger

Shiv Bhushan Tripathi

CDOS Talk
16th July 2022

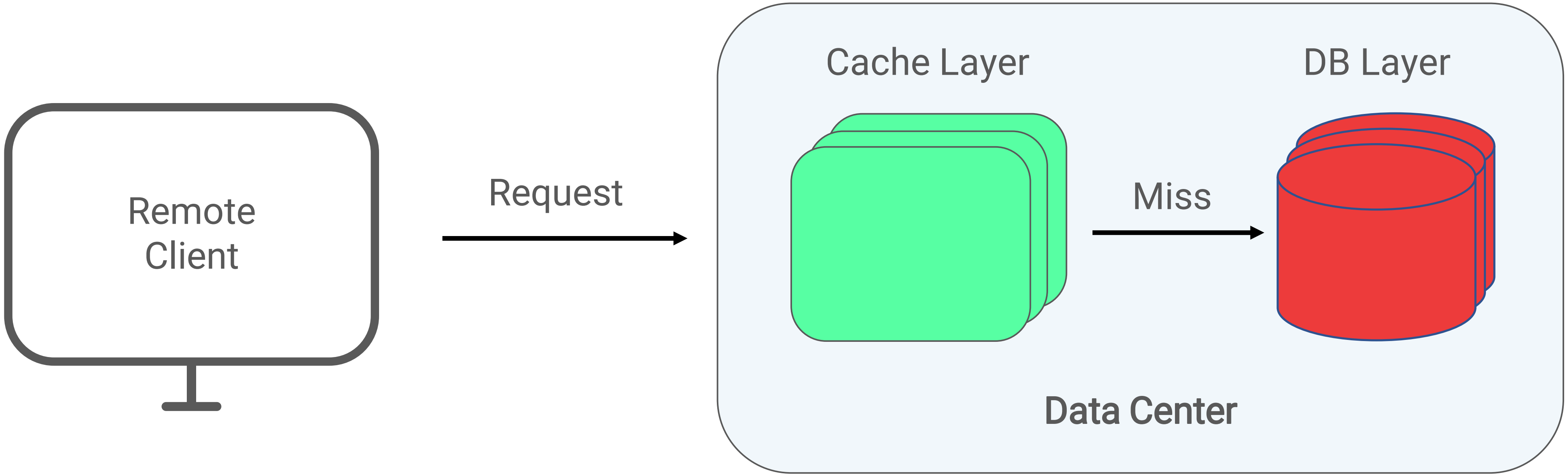
Caching in Data Center

Caching in Data Center



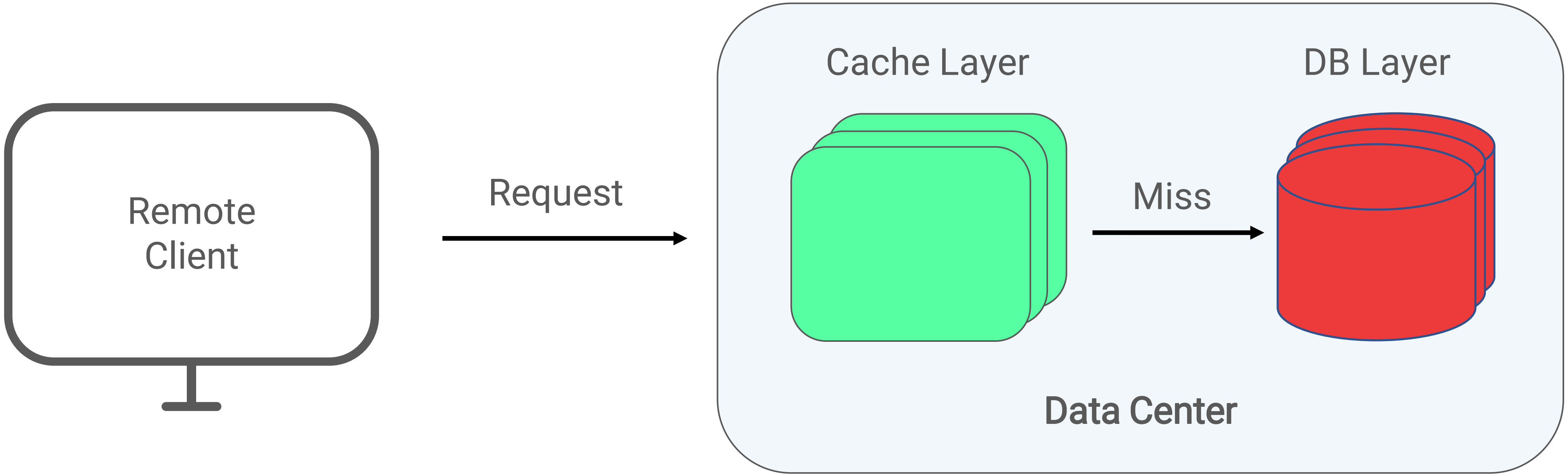
- Added Cache layer

Caching in Data Center



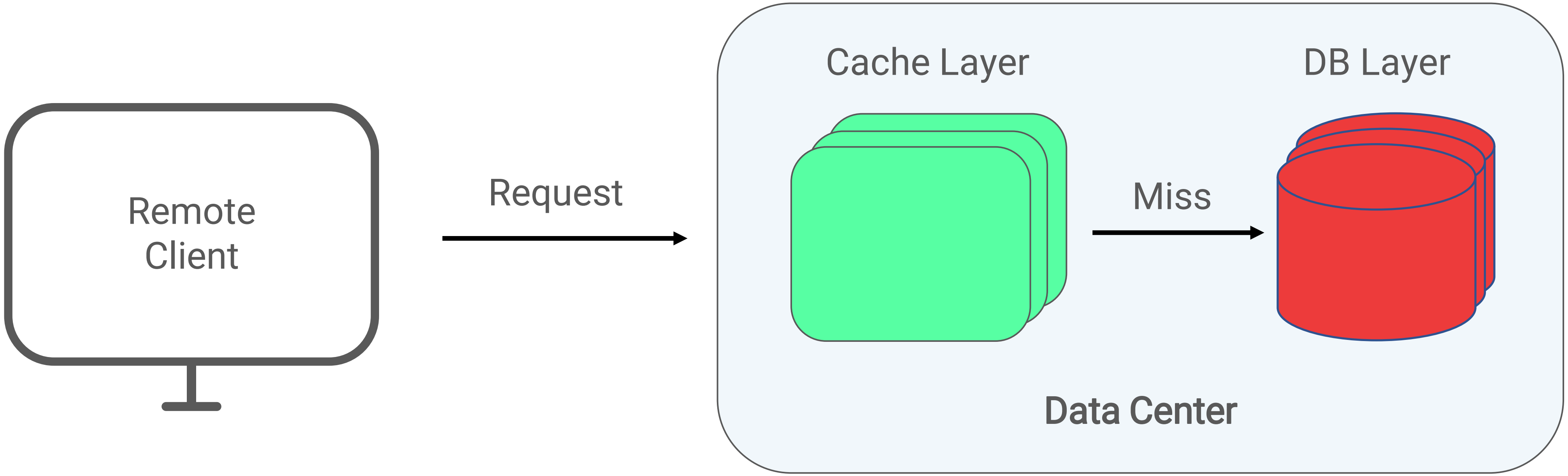
- Added Cache layer

Caching in Data Center



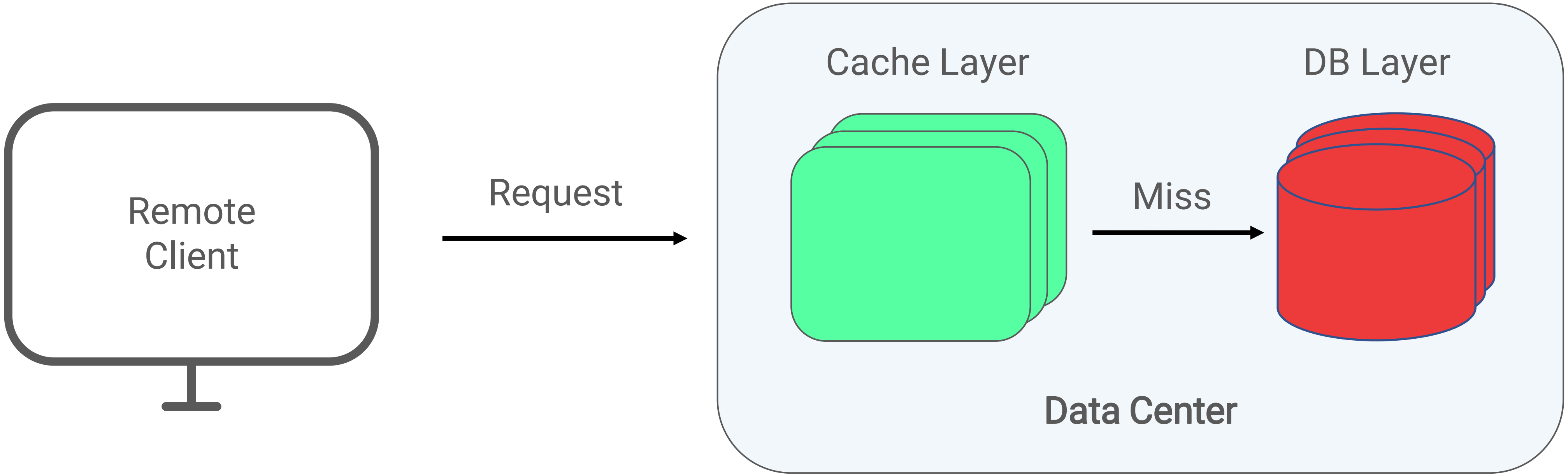
- Added Cache layer: faster response

Caching in Data Center



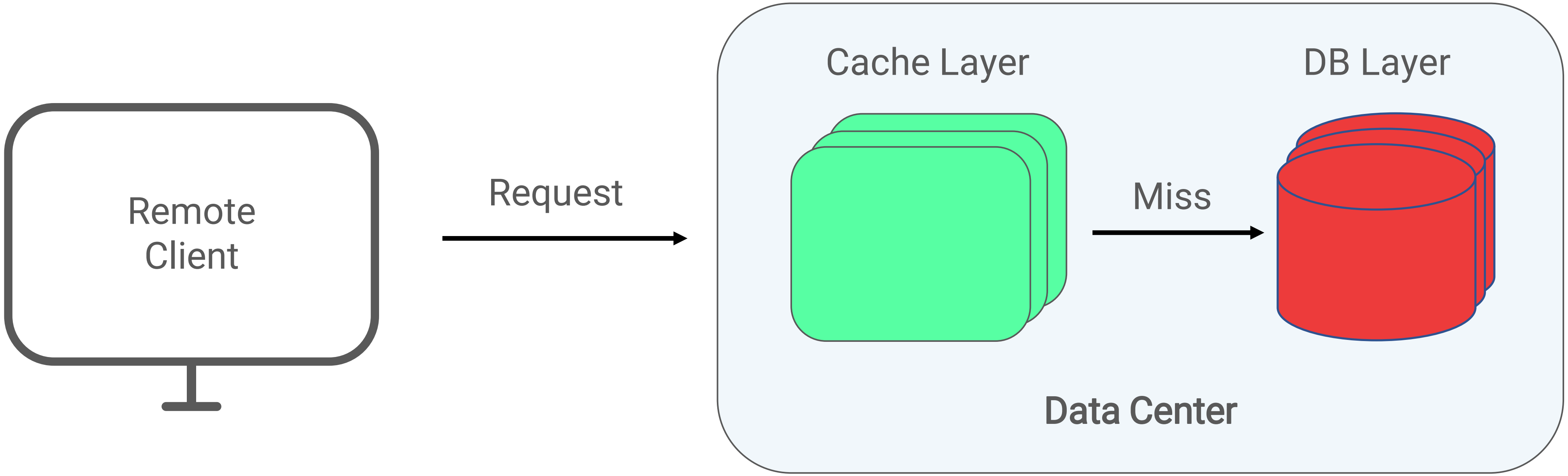
- Added Cache layer: faster response, less traffic to back end DB layer

Caching in Data Center



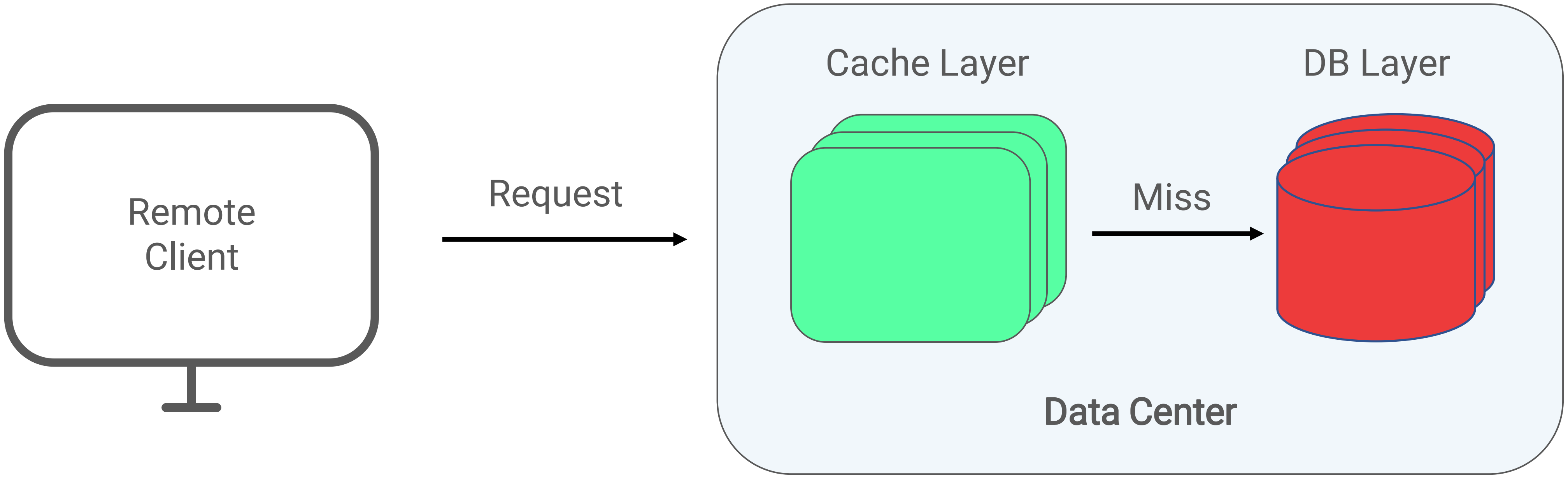
- Added Cache layer: faster response, less traffic to back end DB layer
- Caches should be high performant and low cost

Caching in Data Center



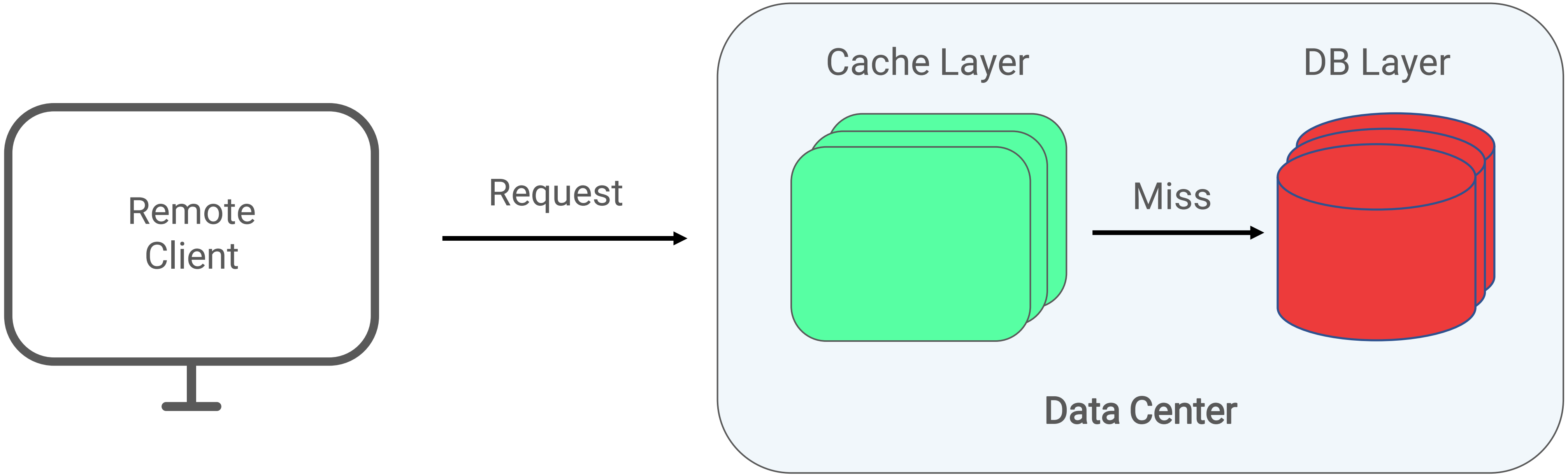
- Added Cache layer: faster response, less traffic to back end DB layer
- Caches should be high performant and low cost
- Flash as Cache

Caching in Data Center



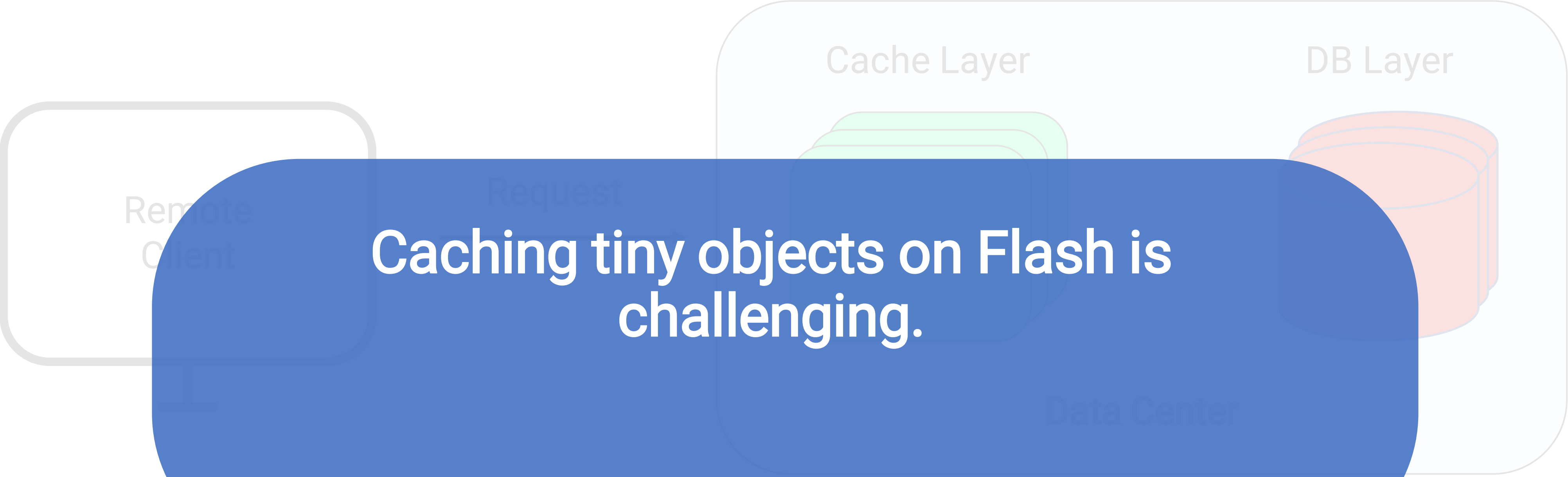
- Added Cache layer: faster response, less traffic to back end DB layer
- Caches should be high performant and low cost
- Flash as Cache: cheaper than DRAM and NVMs

Caching in Data Center



- Added Cache layer: faster response, less traffic to back end DB layer
- Caches should be high performant and low cost
- Flash as Cache: cheaper than DRAM and NVMs, better performance than HDD

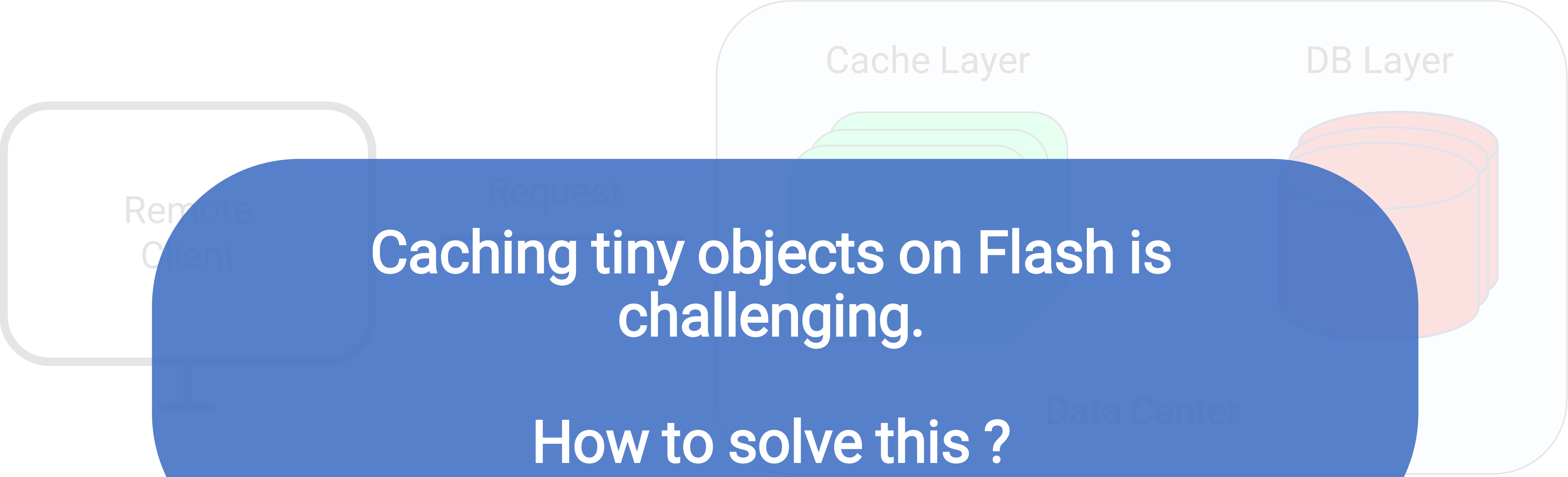
Caching in Data Center



Caching tiny objects on Flash is challenging.

- Added Cache layer: faster response, less traffic to back end DB layer
- Caches should be high performant and low cost
- Flash as Cache: cheaper than DRAM and NVMs, better performance than HDD

Caching in Data Center



Caching tiny objects on Flash is challenging.

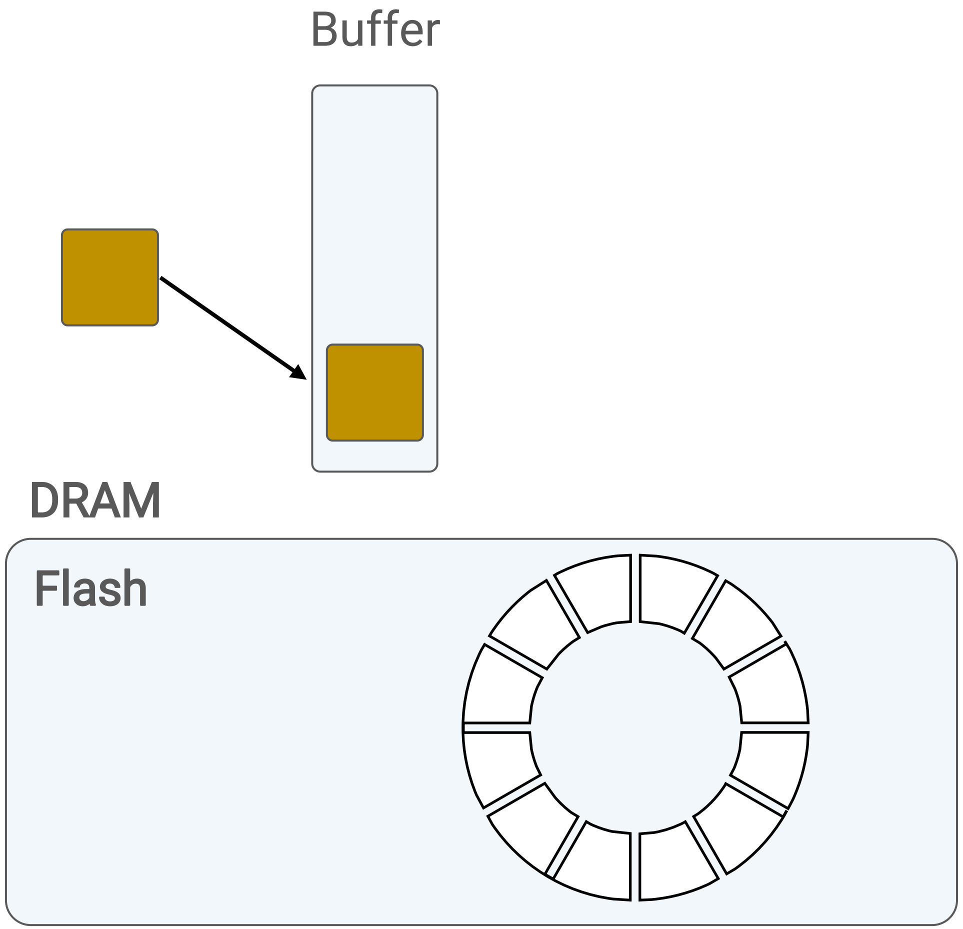
How to solve this ?

- Added Cache layer: faster response, less traffic to back end DB layer
- Caches should be high performant and low cost
- Flash as Cache: cheaper than DRAM and NVMs, better performance than HDD

Flash as Cache: Log Structured

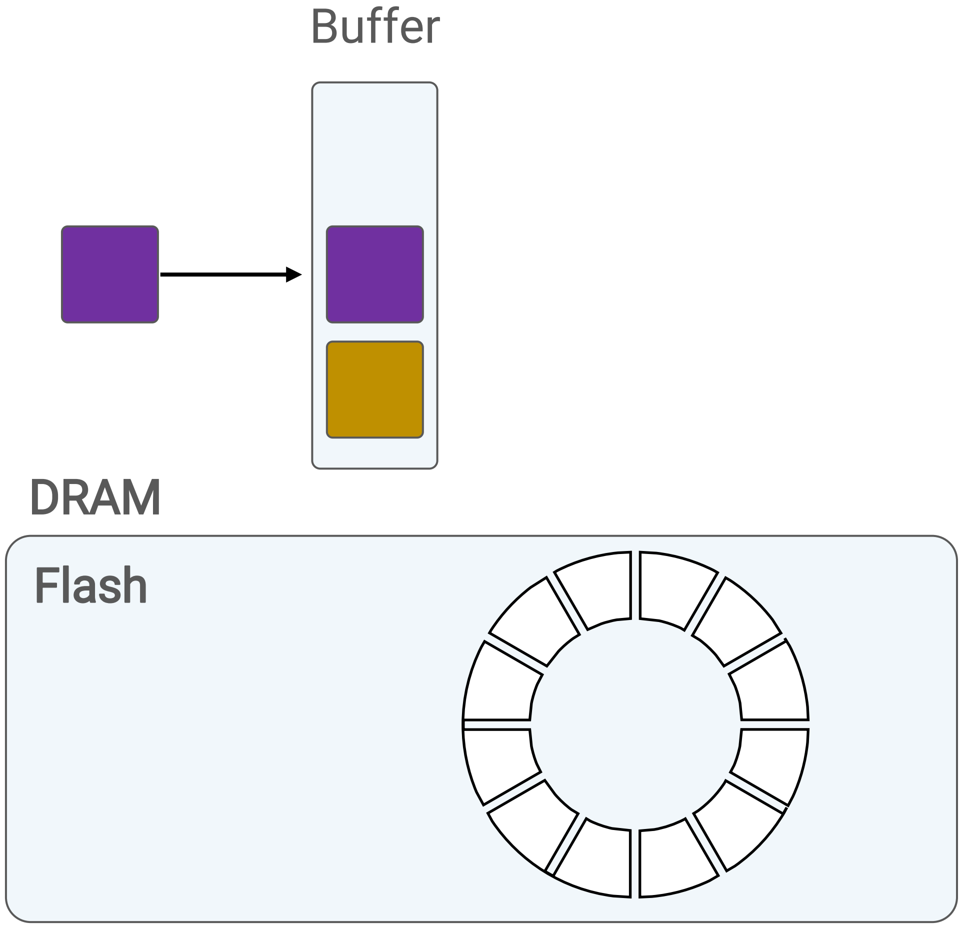
Flash as Cache: Log Structured

- Tiny objects are buffered in DRAM



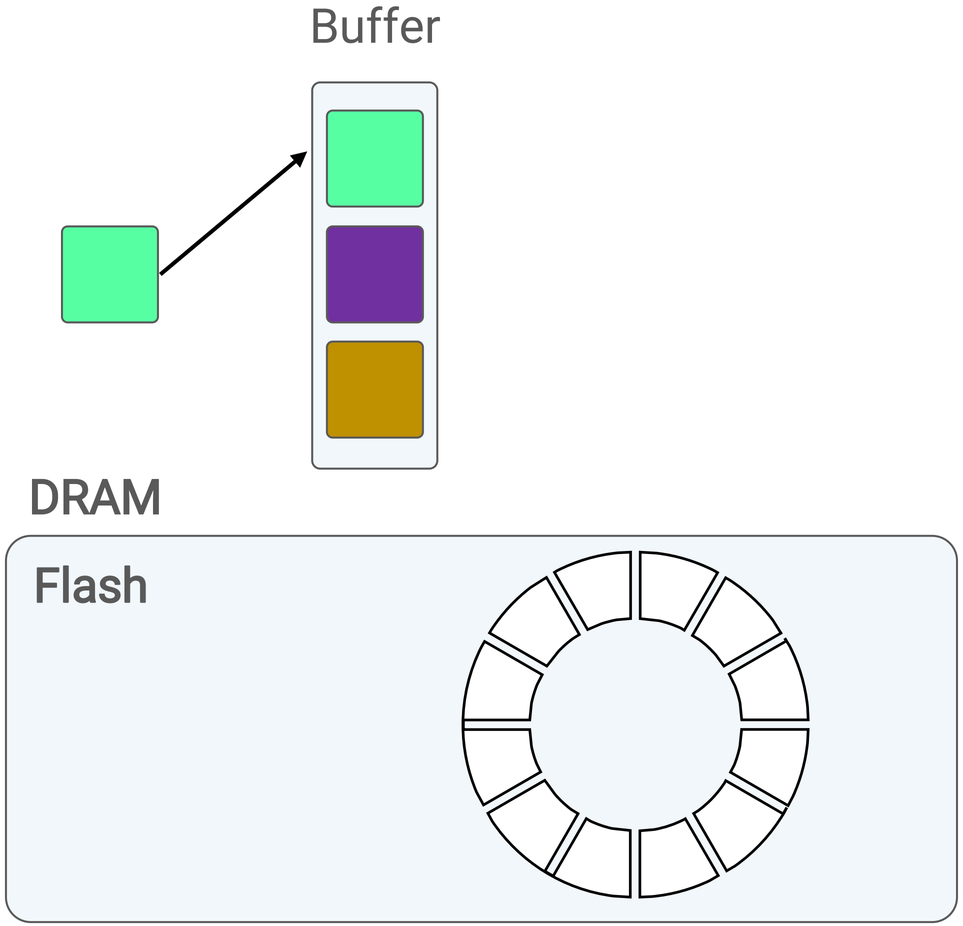
Flash as Cache: Log Structured

- Tiny objects are buffered in DRAM



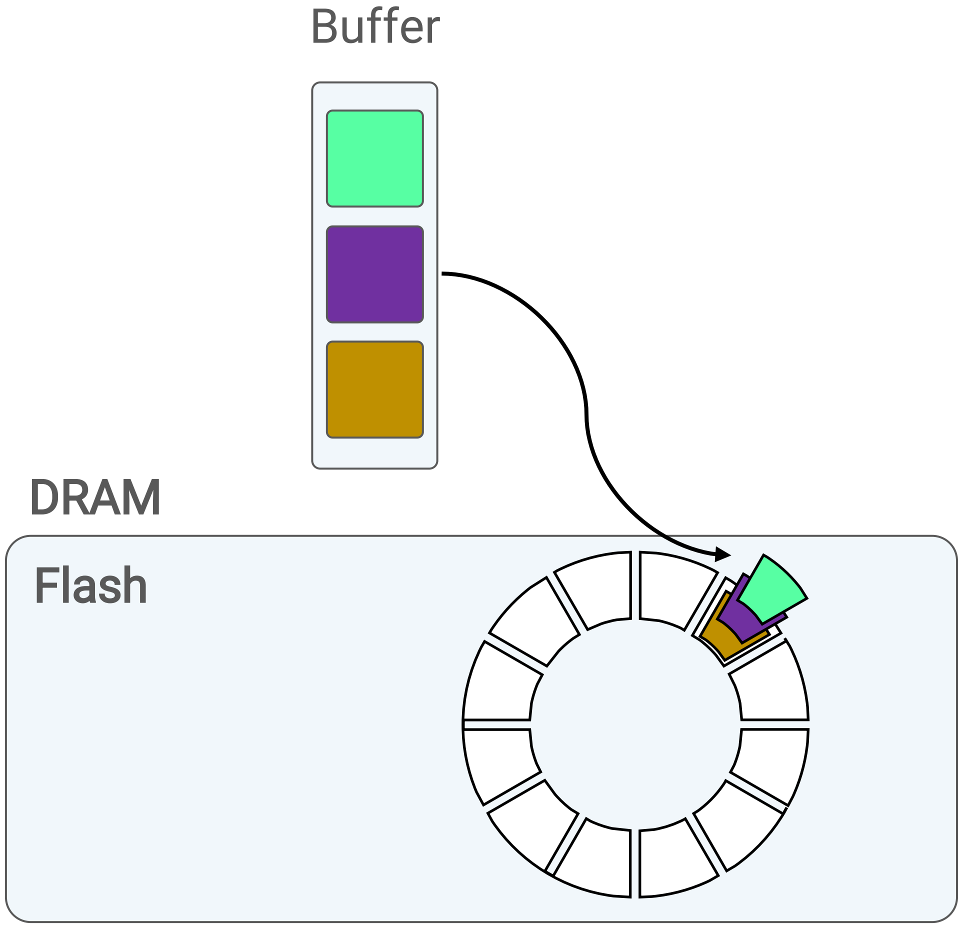
Flash as Cache: Log Structured

- Tiny objects are buffered in DRAM



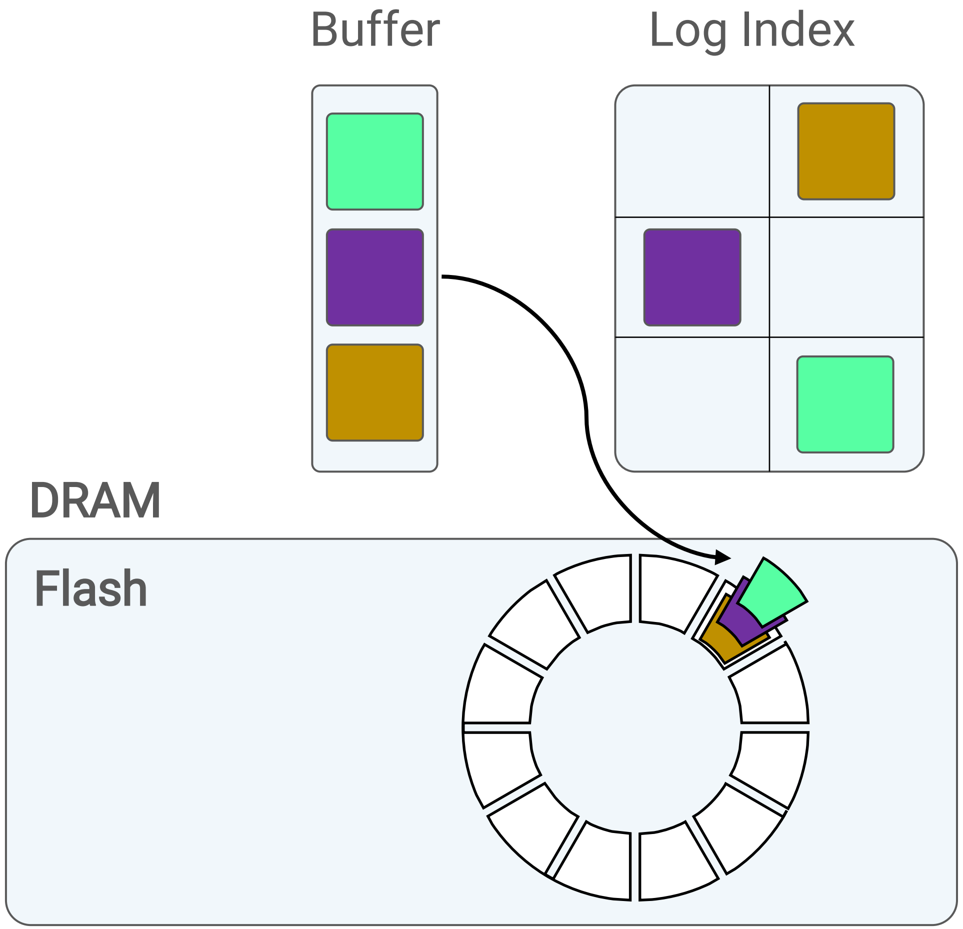
Flash as Cache: Log Structured

- Tiny objects are buffered in DRAM
- Moved to flash, when buffer is full



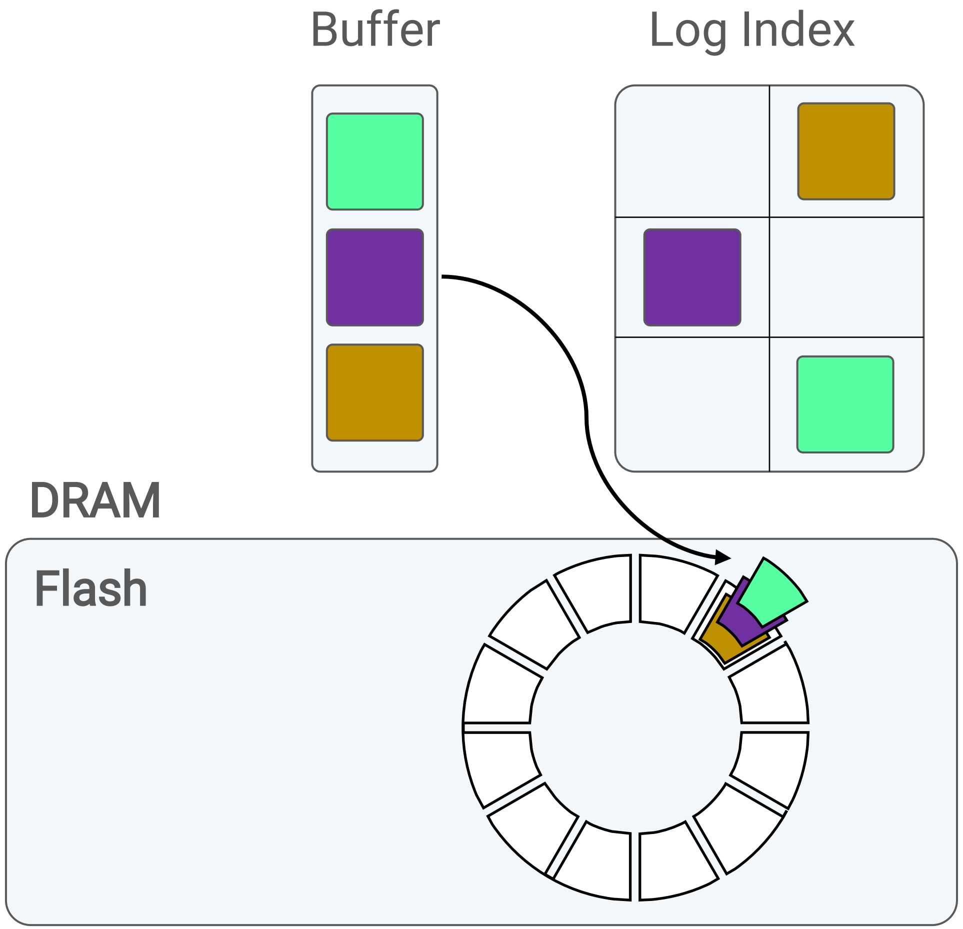
Flash as Cache: Log Structured

- Tiny objects are buffered in DRAM
- Moved to flash, when buffer is full
- A log index in DRAM to track each object



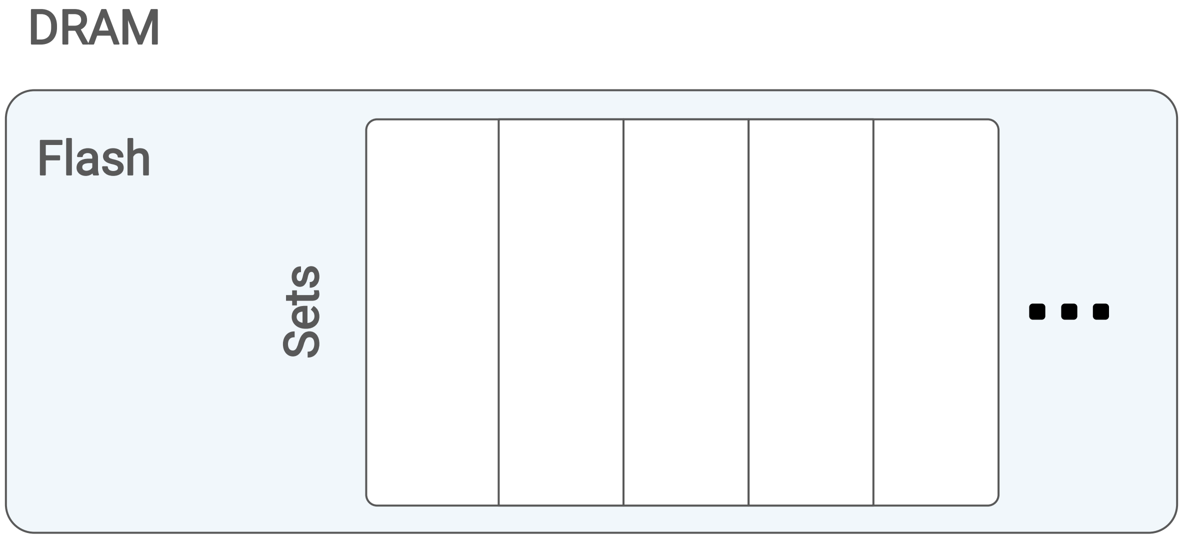
Flash as Cache: Log Structured

- Tiny objects are buffered in DRAM
- Moved to flash, when buffer is full
- A log index in DRAM to track each object
- Reduces flash writes but huge amount of DRAM



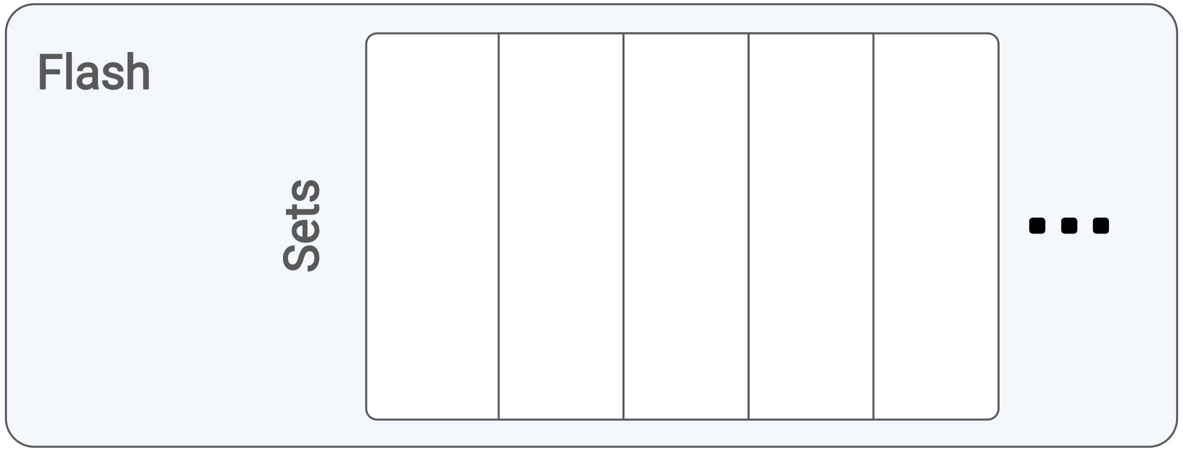
Flash as Cache: Set Associative

Flash as Cache: Set Associative



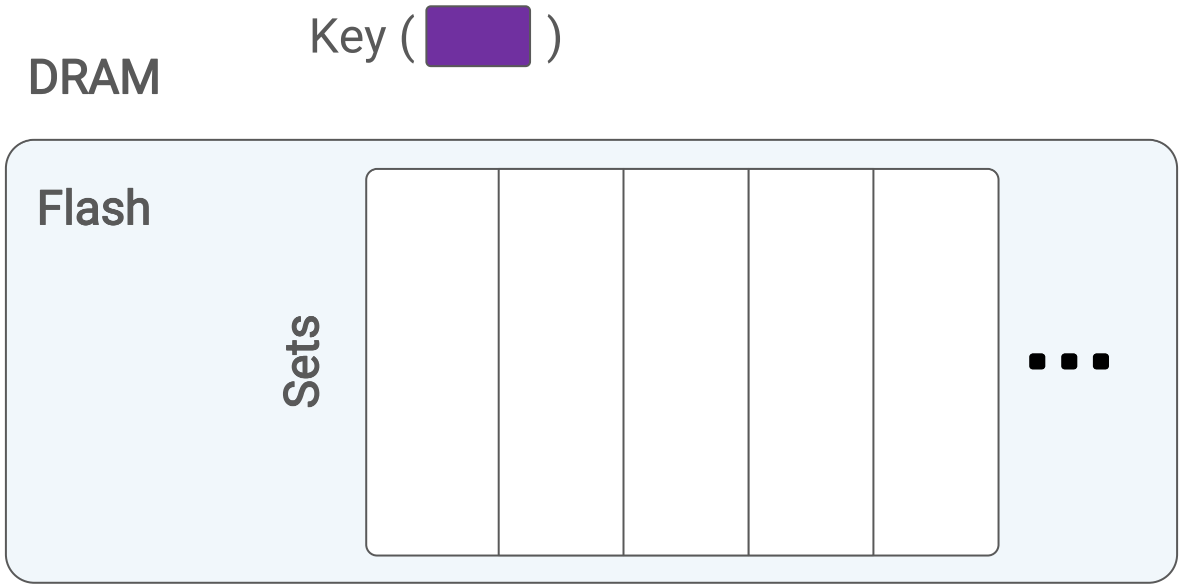
Flash as Cache: Set Associative (Insert)

DRAM



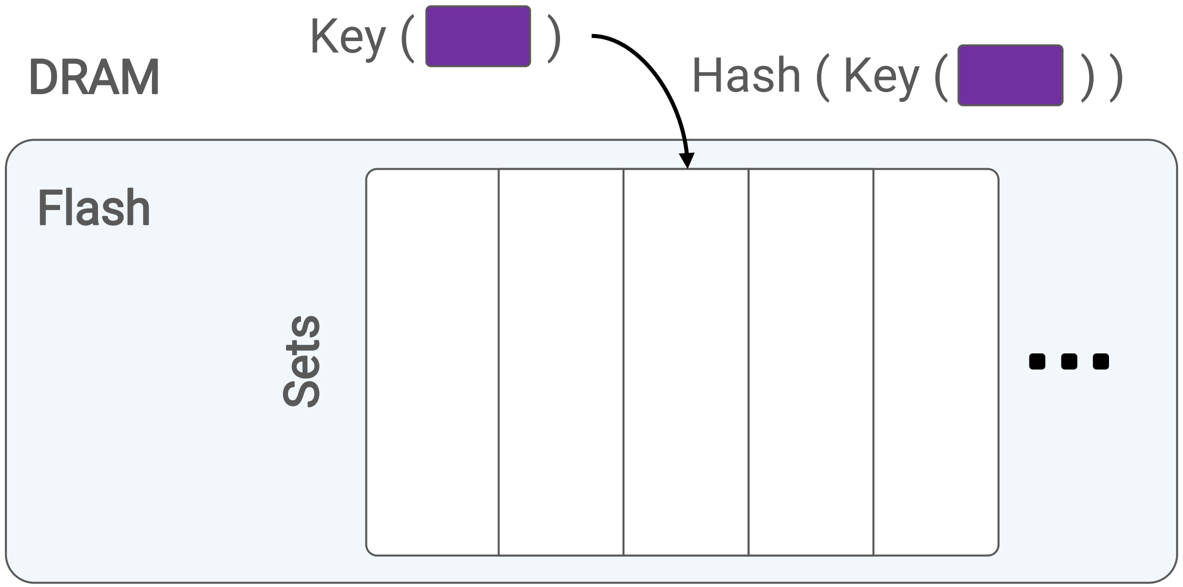
- Objects are inserted into sets using Hashing

Flash as Cache: Set Associative (Insert)



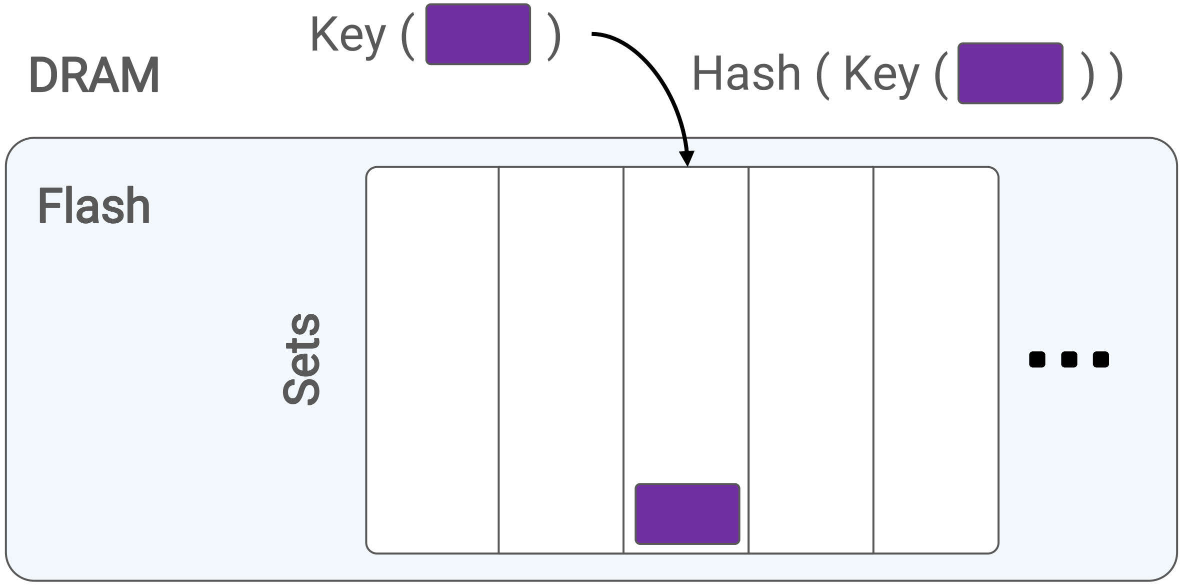
- Objects are inserted into sets using Hashing

Flash as Cache: Set Associative (Insert)



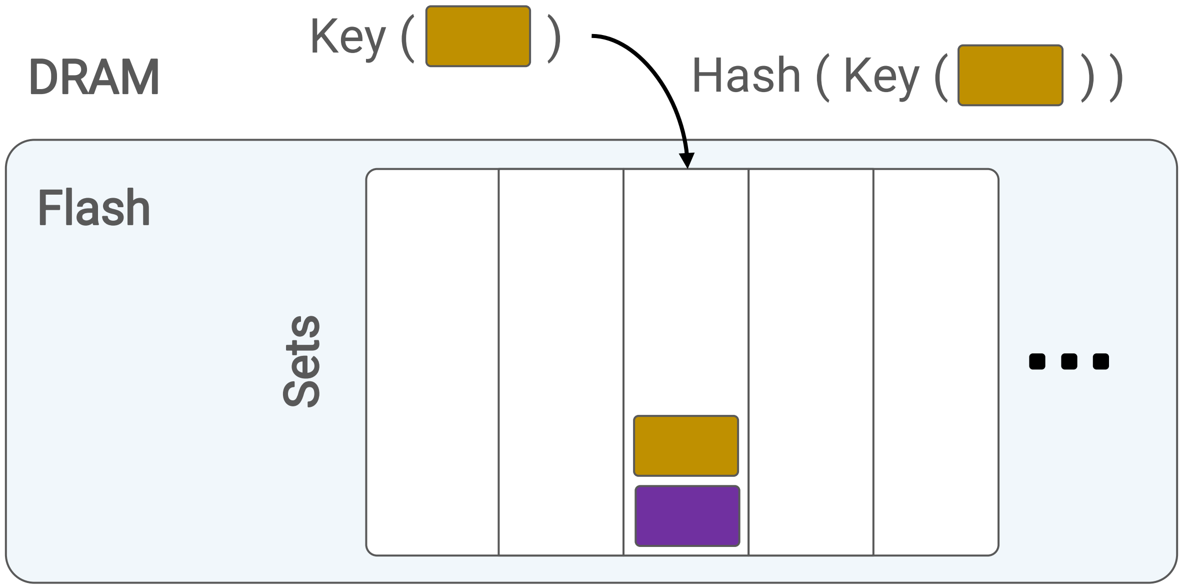
- Objects are inserted into sets using Hashing

Flash as Cache: Set Associative (Insert)



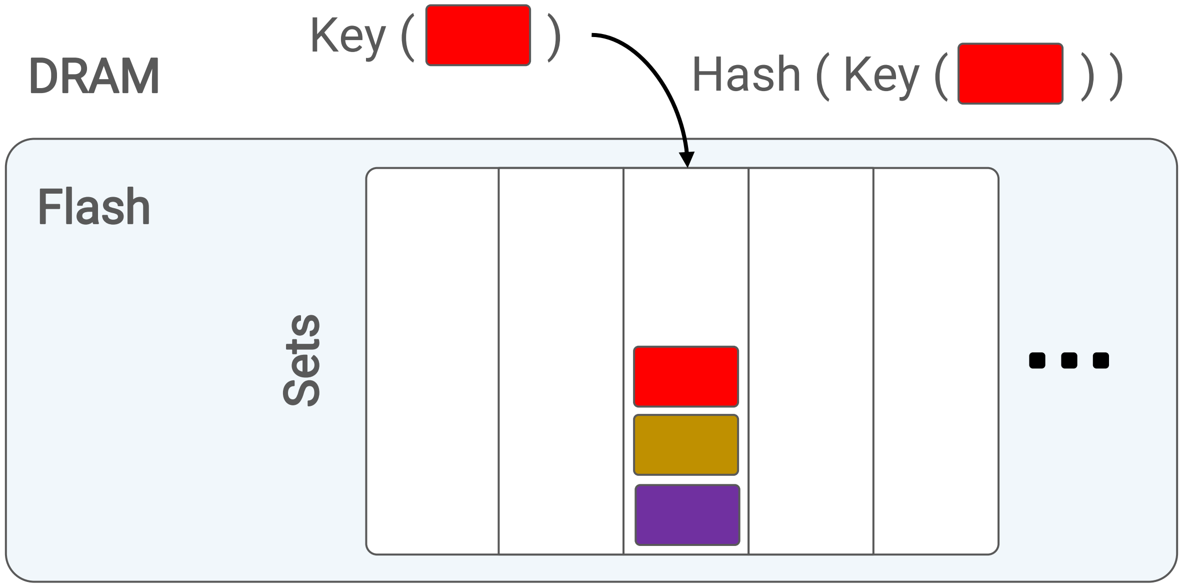
- Objects are inserted into sets using Hashing

Flash as Cache: Set Associative (Insert)



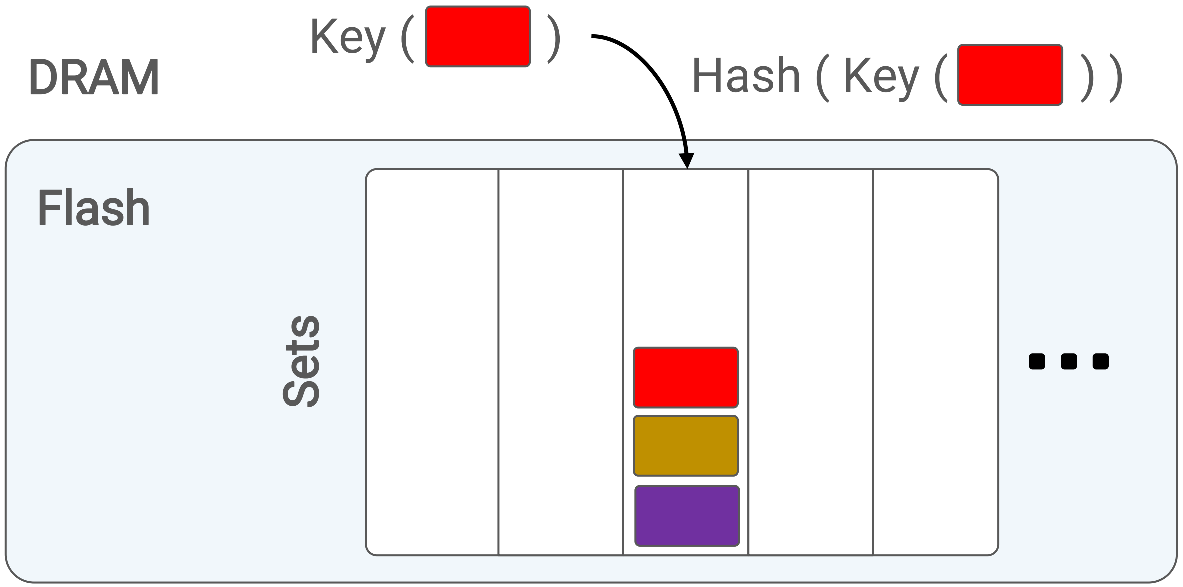
- Objects are inserted into sets using Hashing

Flash as Cache: Set Associative (Insert)



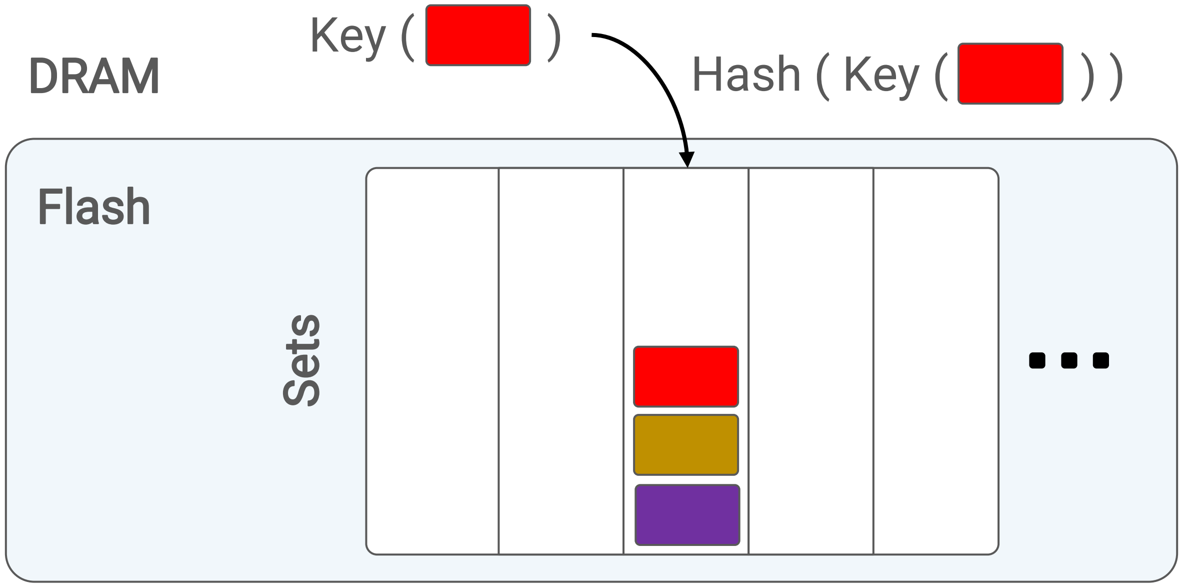
- Objects are inserted into sets using Hashing

Flash as Cache: Set Associative (Insert)



- Objects are inserted into sets using Hashing
- Write amplification is too high for tiny objects

Flash as Cache: Set Associative (Insert)

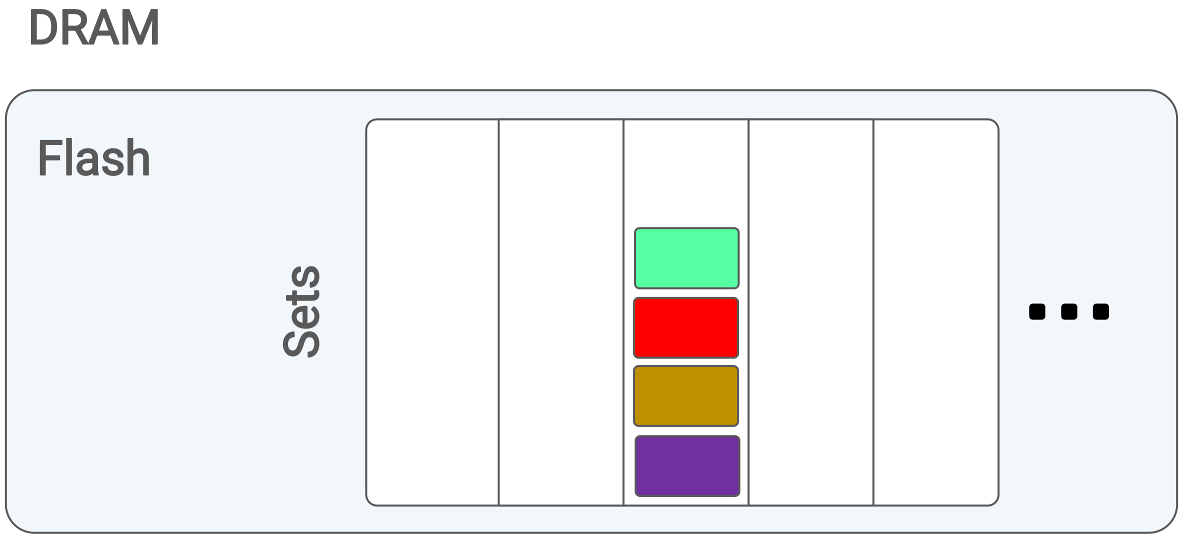


- Objects are inserted into sets using Hashing
- Write amplification is too high for tiny objects:

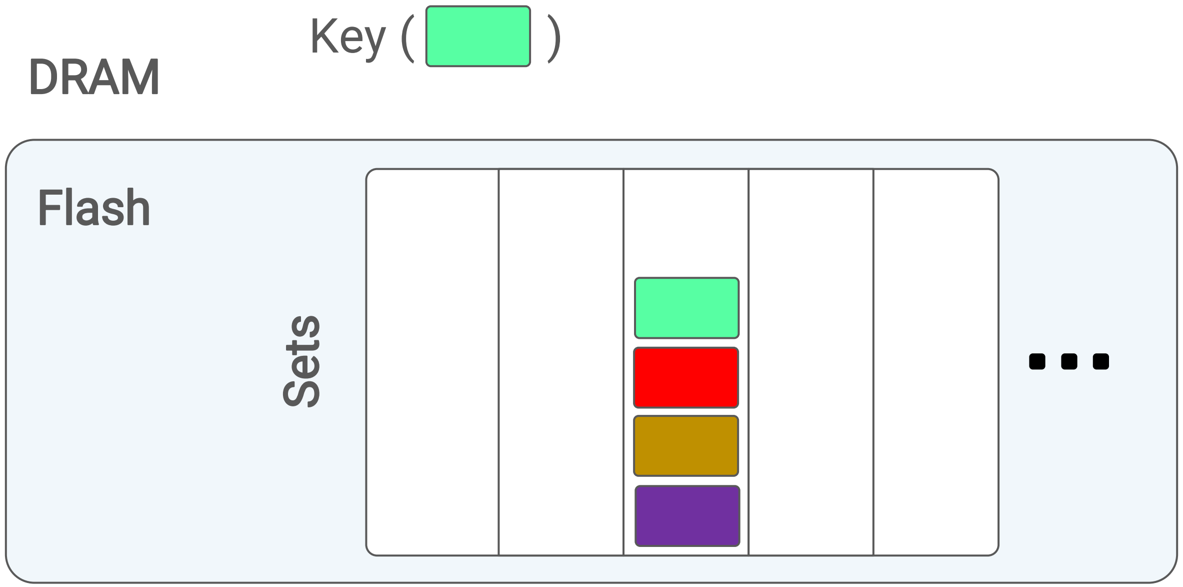
Write Amplification = 4096 / 100 = 40x

Flash as Cache: Set Associative (Search)

Flash as Cache: Set Associative (Search)

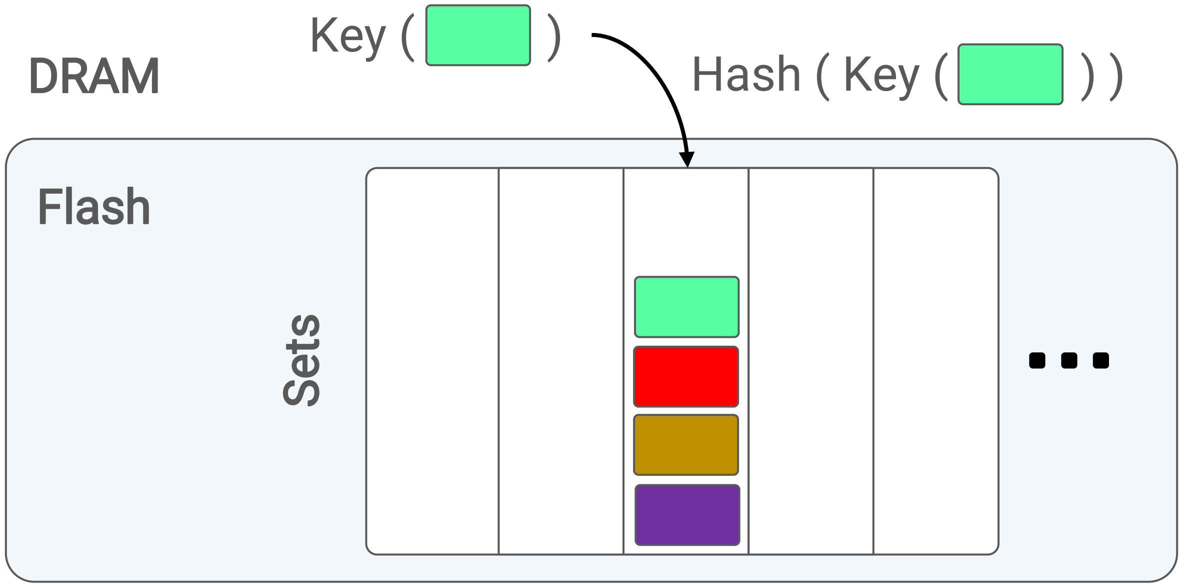


Flash as Cache: Set Associative (Search)



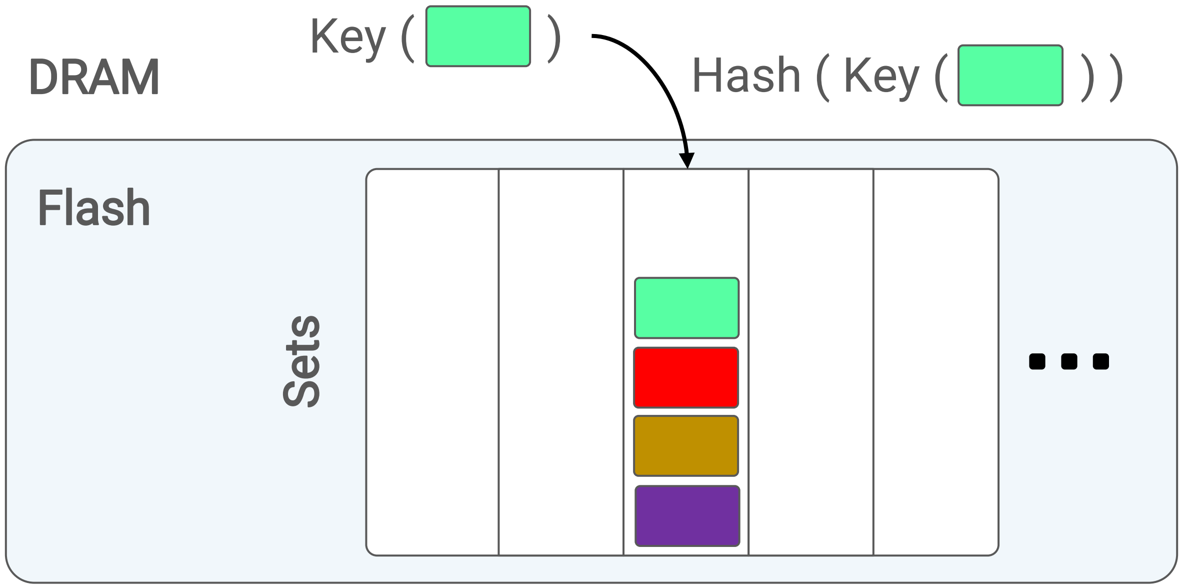
- Objects are searched based on key

Flash as Cache: Set Associative (Search)



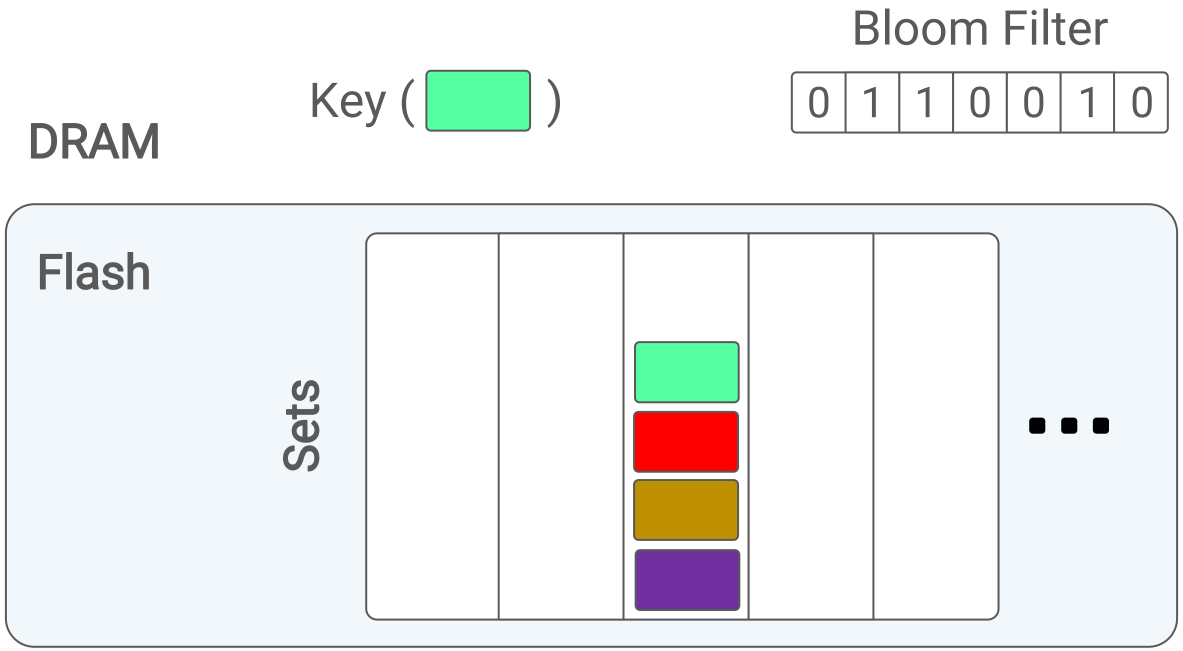
- Objects are searched based on key

Flash as Cache: Set Associative (Search)



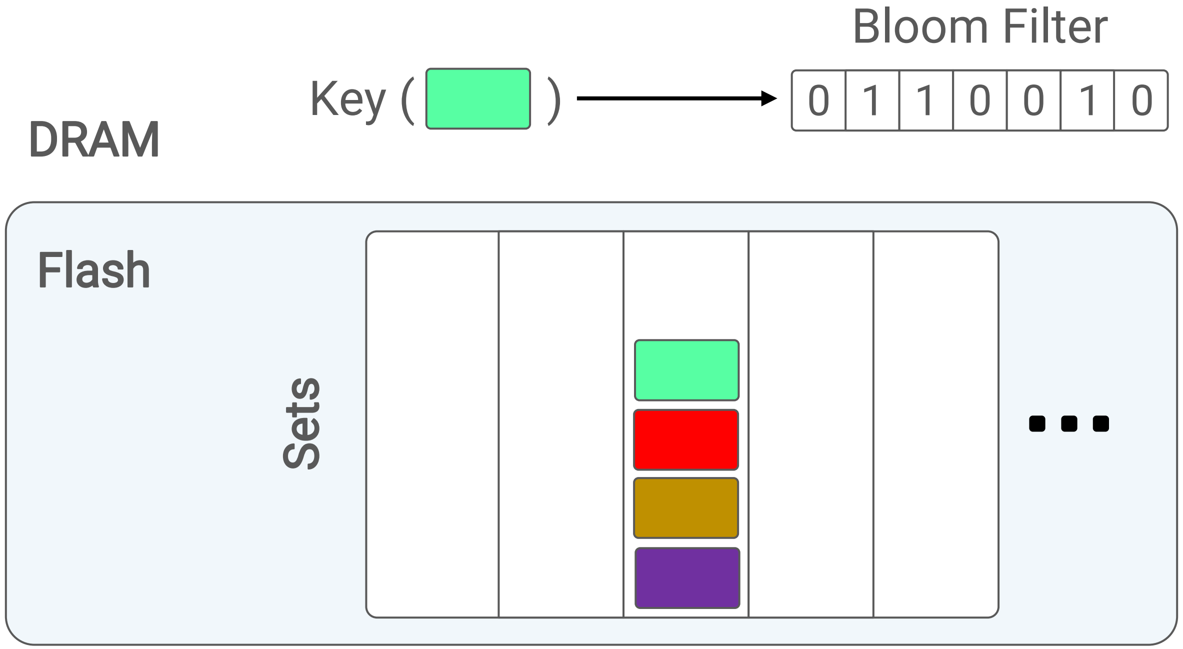
- Objects are searched based on key
- Have to read all tiny objects in the set (flash read)

Flash as Cache: Set Associative (Search)



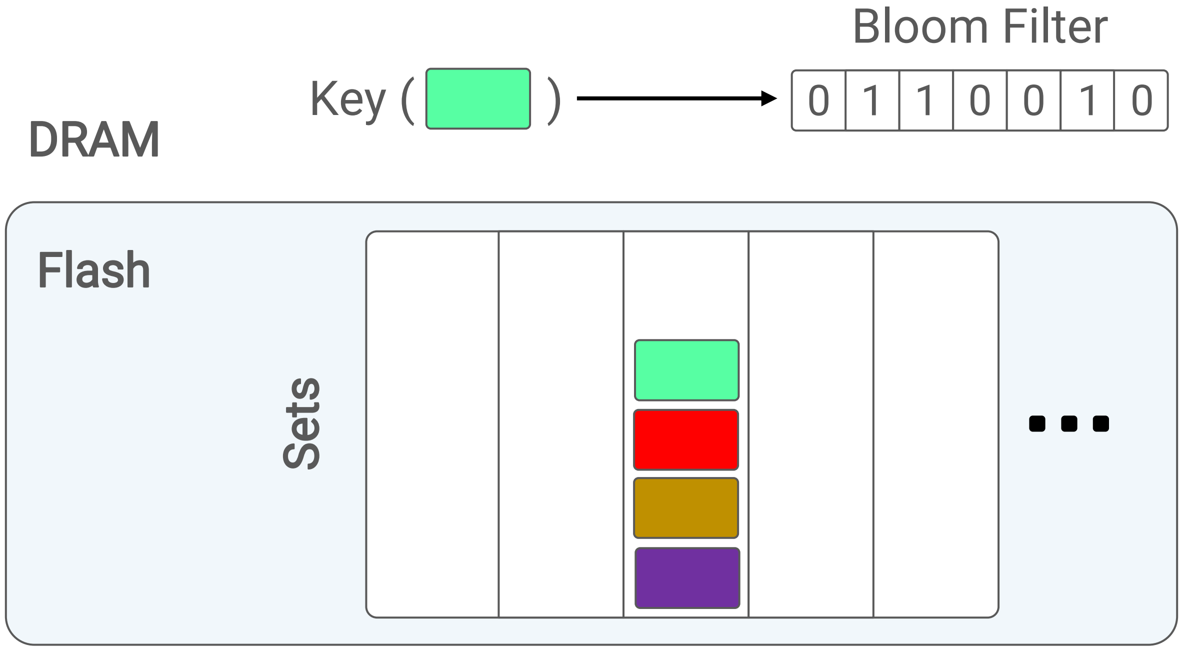
- Objects are searched based on key
- Have to read all tiny objects in the set (flash read): Add bloom filter

Flash as Cache: Set Associative (Search)



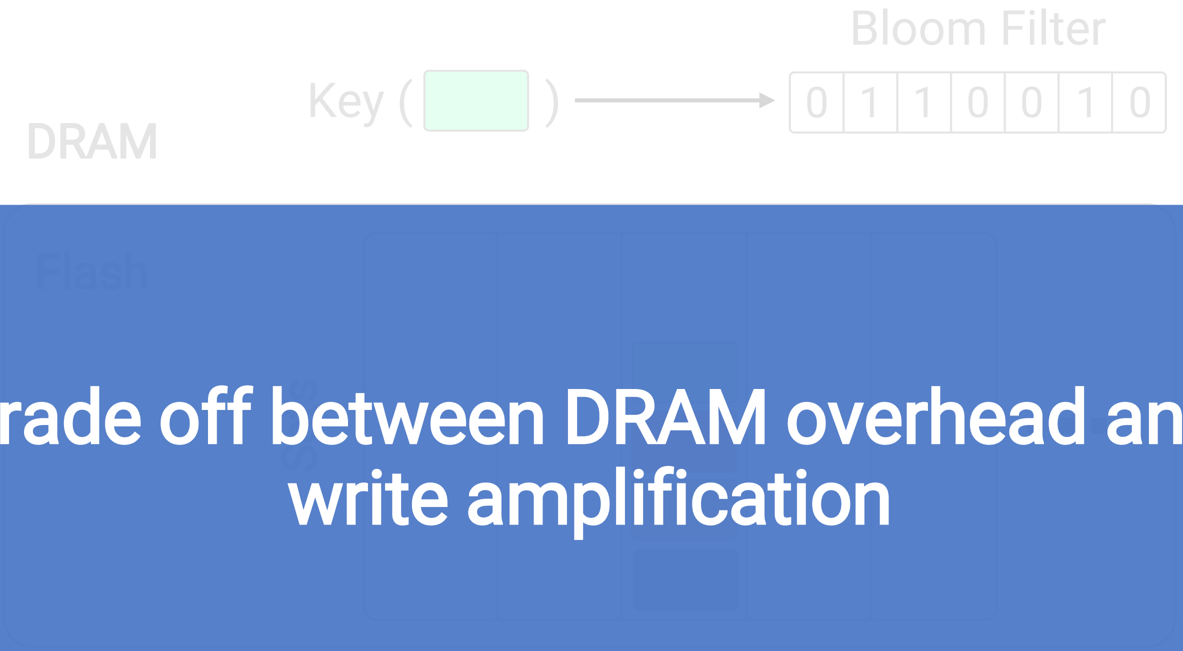
- Objects are searched based on key
- Have to read all tiny objects in the set (flash read): Add bloom filter

Flash as Cache: Set Associative (Search)



- Objects are searched based on key
- Have to read all tiny objects in the set (flash read): Add bloom filter
- Less DRAM overhead but have huge write amplification

Flash as Cache



Trade off between DRAM overhead and write amplification

- Objects are searched based on key
- Have to read all tiny objects in the set (flash read): Add bloom filter
- Less DRAM overhead but have huge write amplification

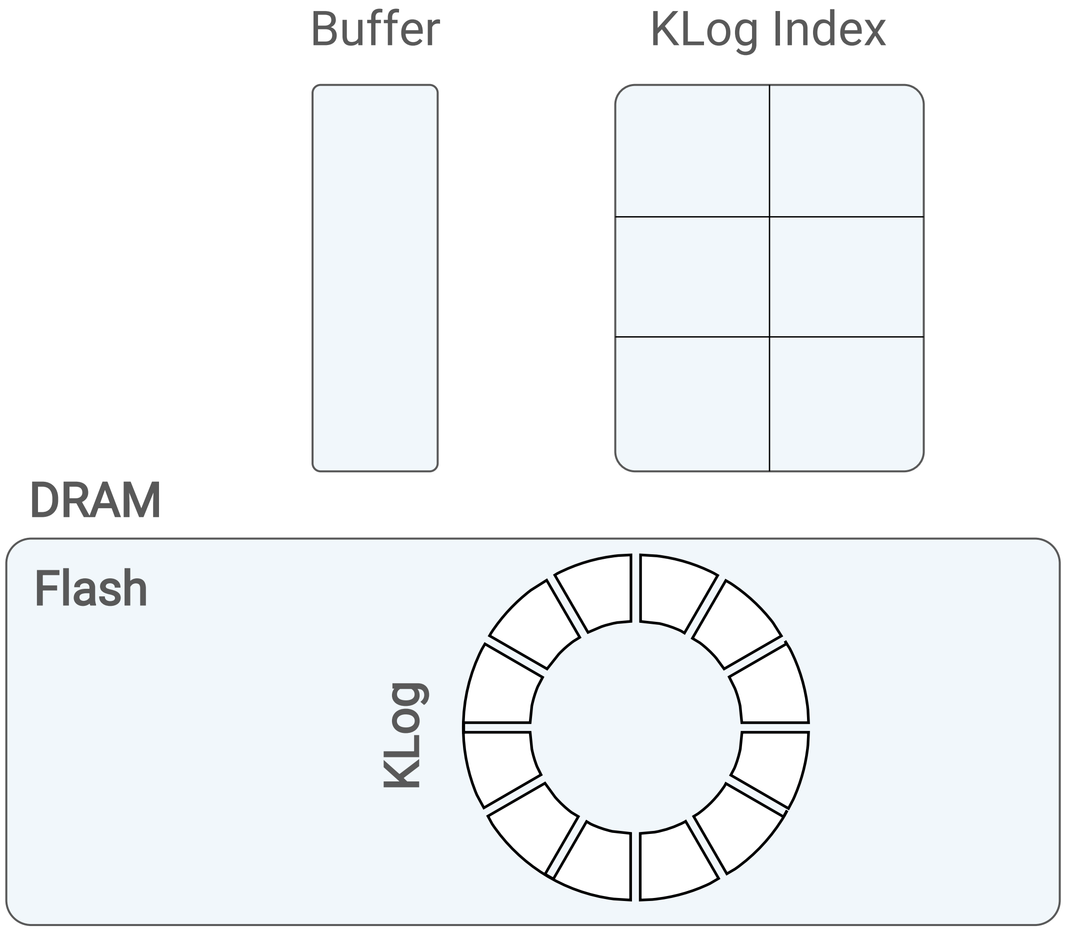
Kangaroo

16-July-2022

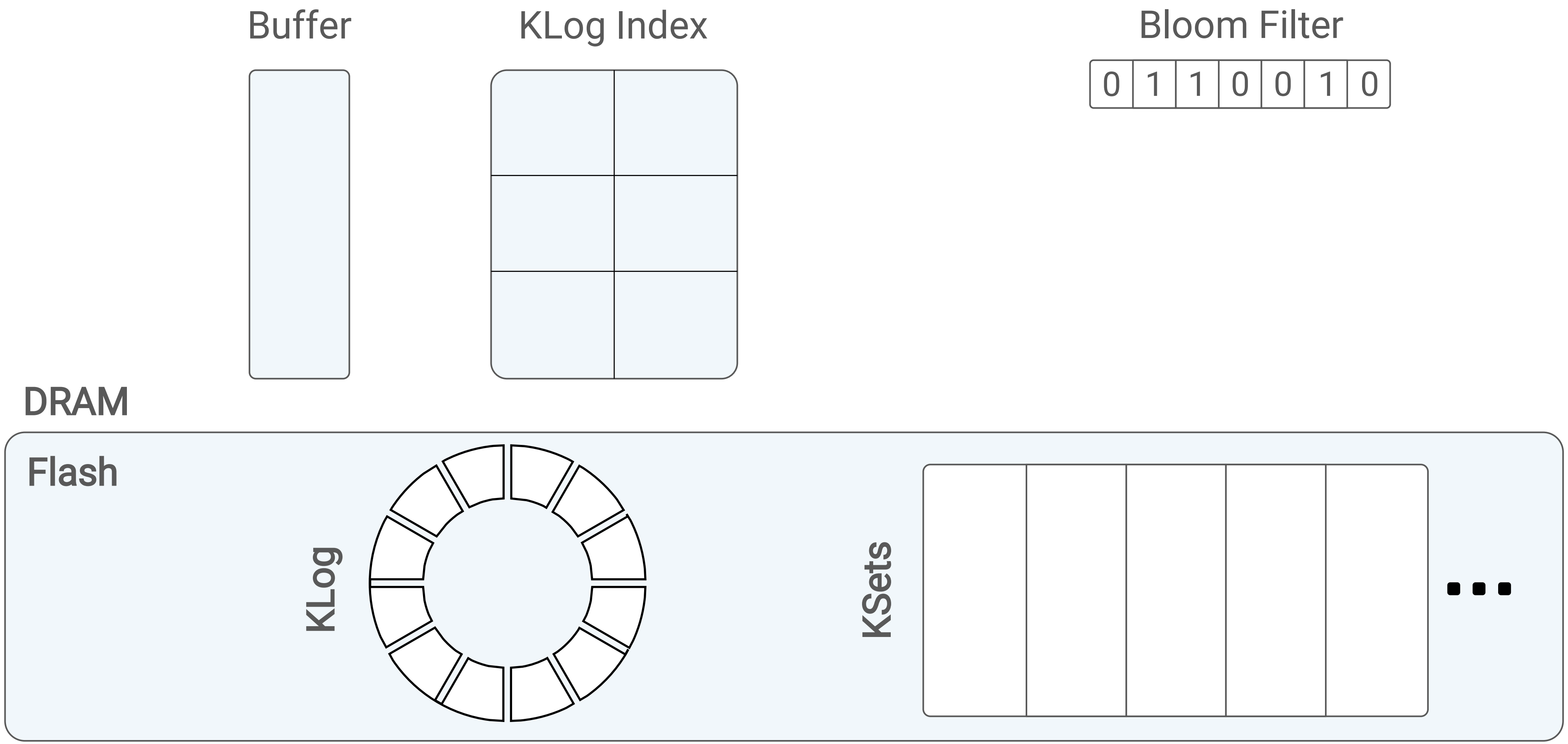
Kangaroo: Design

16-July-2022

Kangaroo: Design

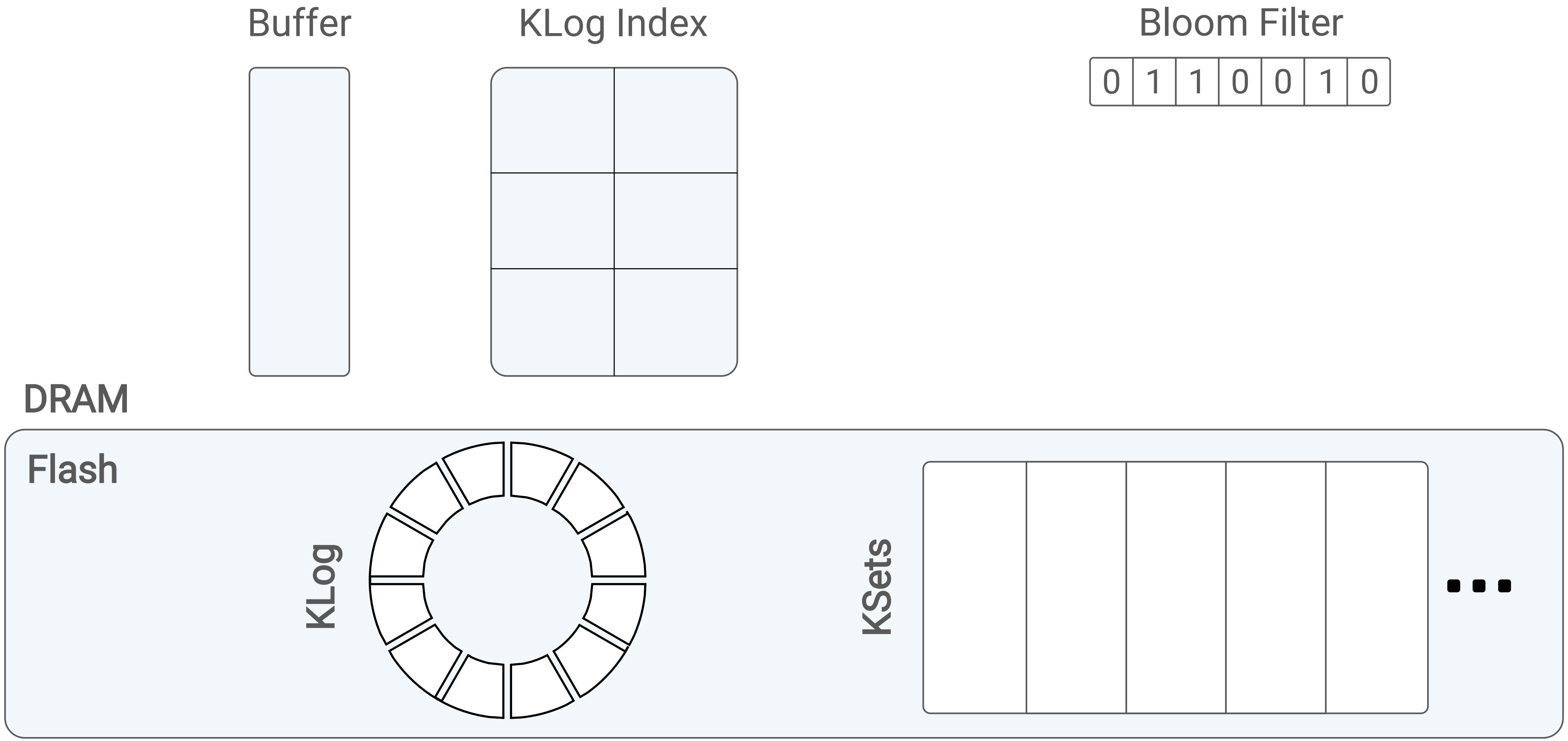


Kangaroo: Design



- ○ Have advantage of both the design

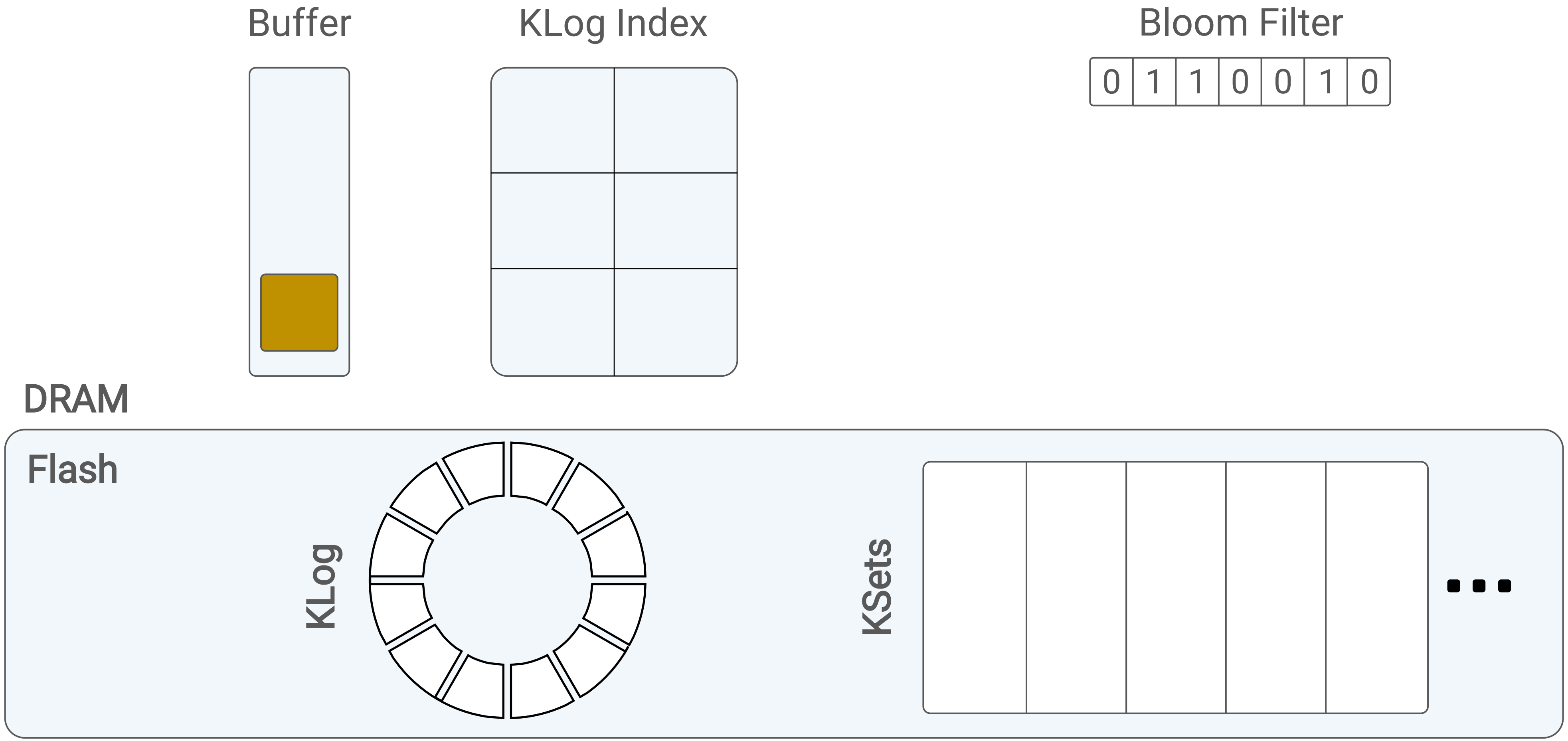
Kangaroo: Design



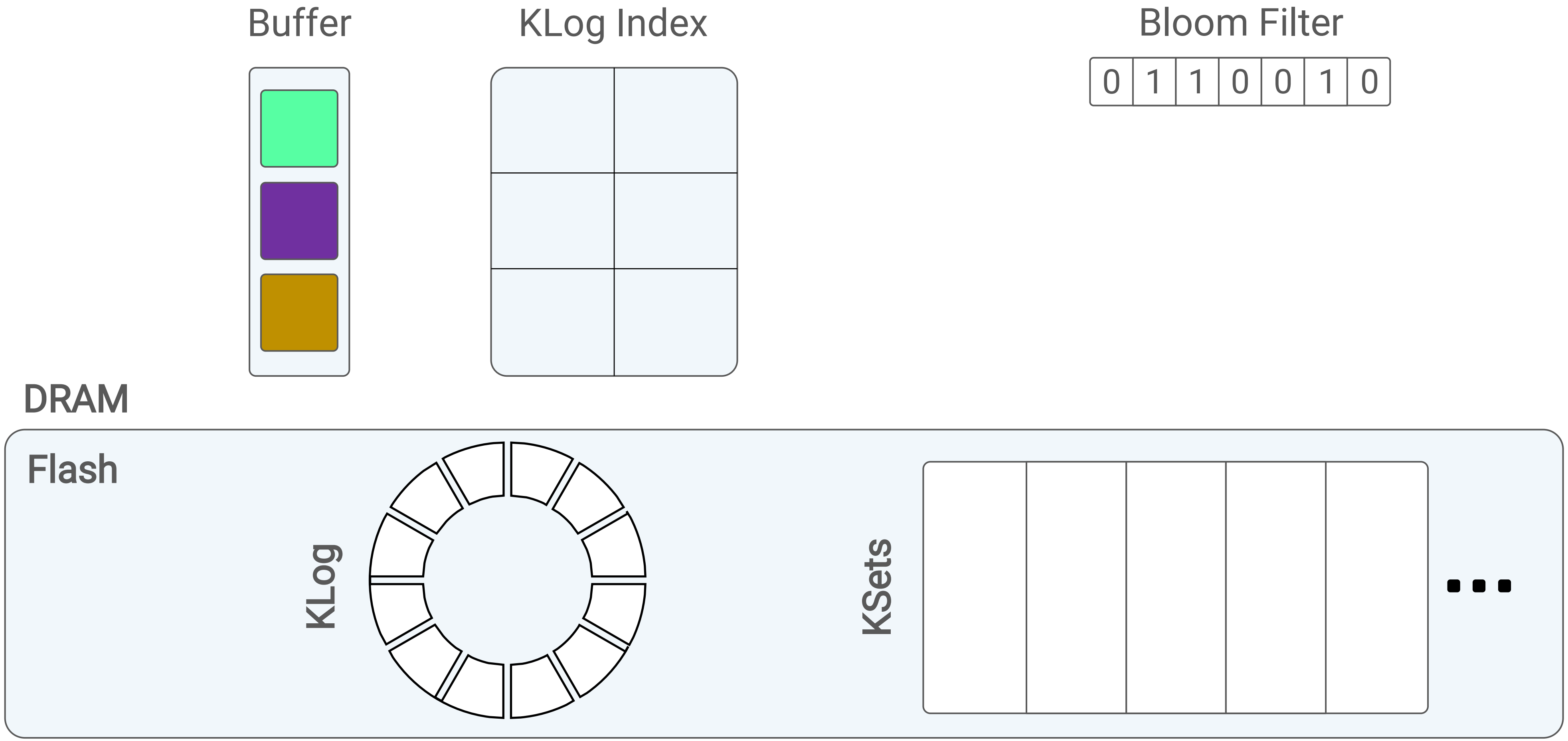
0	1	1	0	0	1	0

- Have advantage of both the design: 5% to KLog

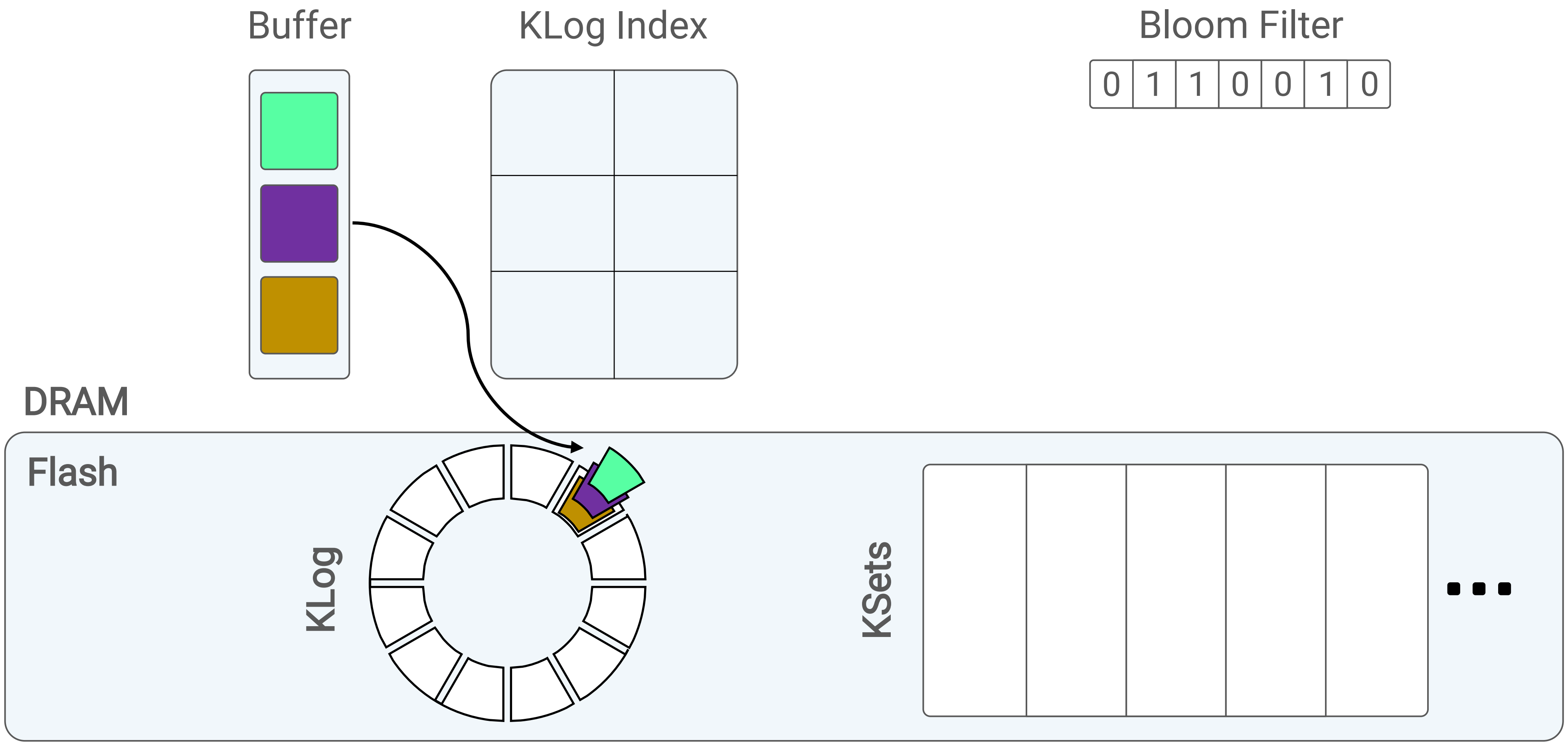
Kangaroo: Insert (to KLog)



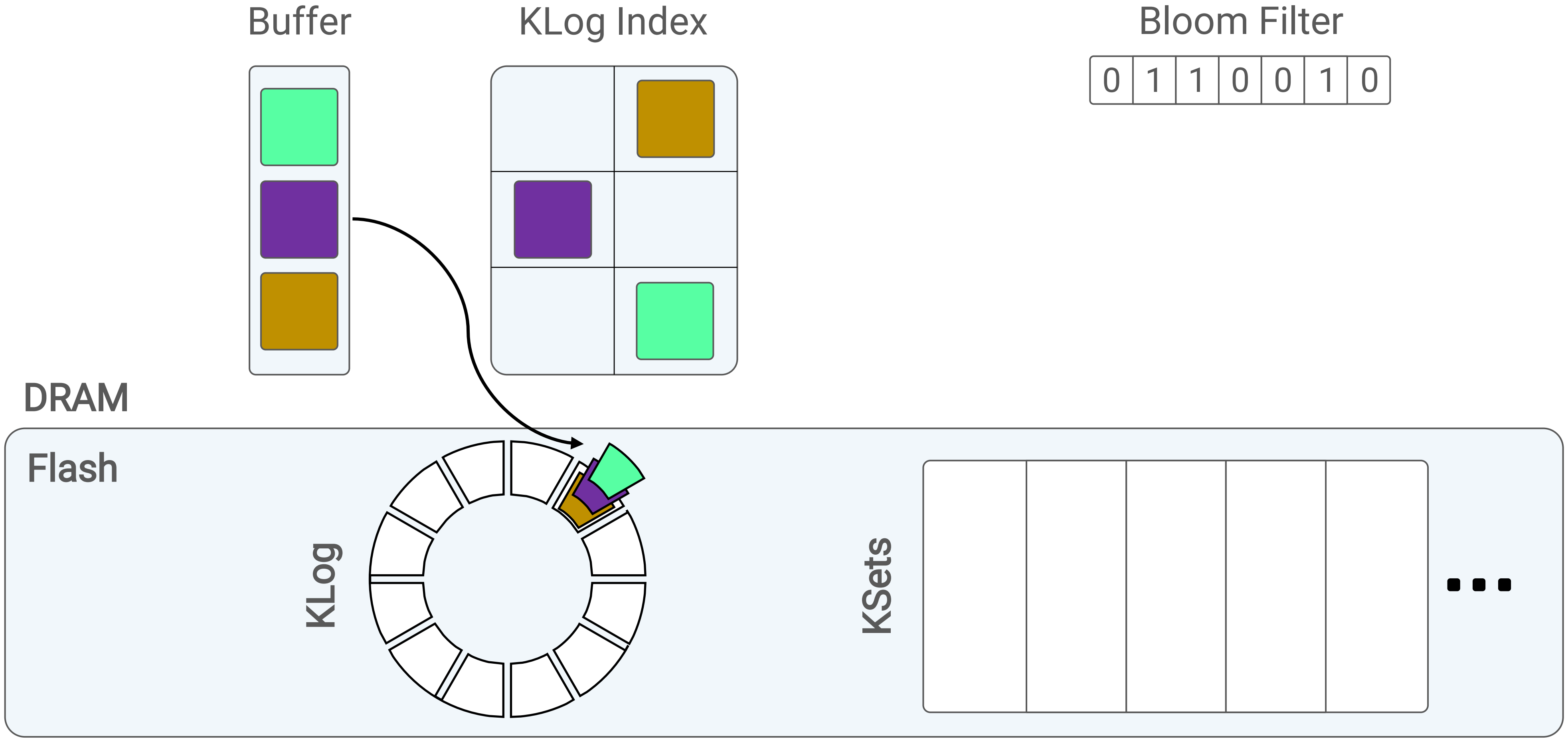
Kangaroo: Insert (to KLog)



Kangaroo: Insert (to KLog)

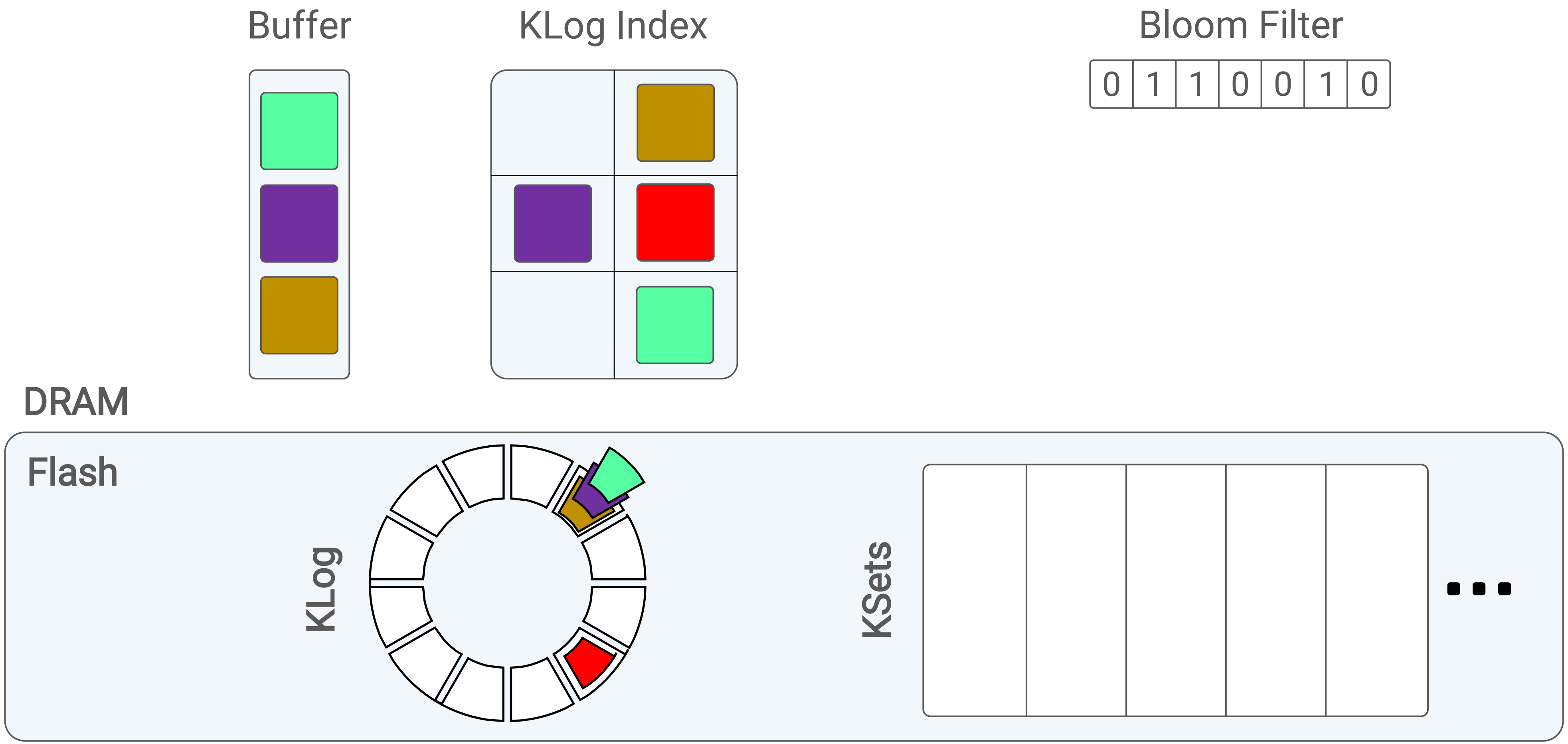


Kangaroo: Insert (to KLog)

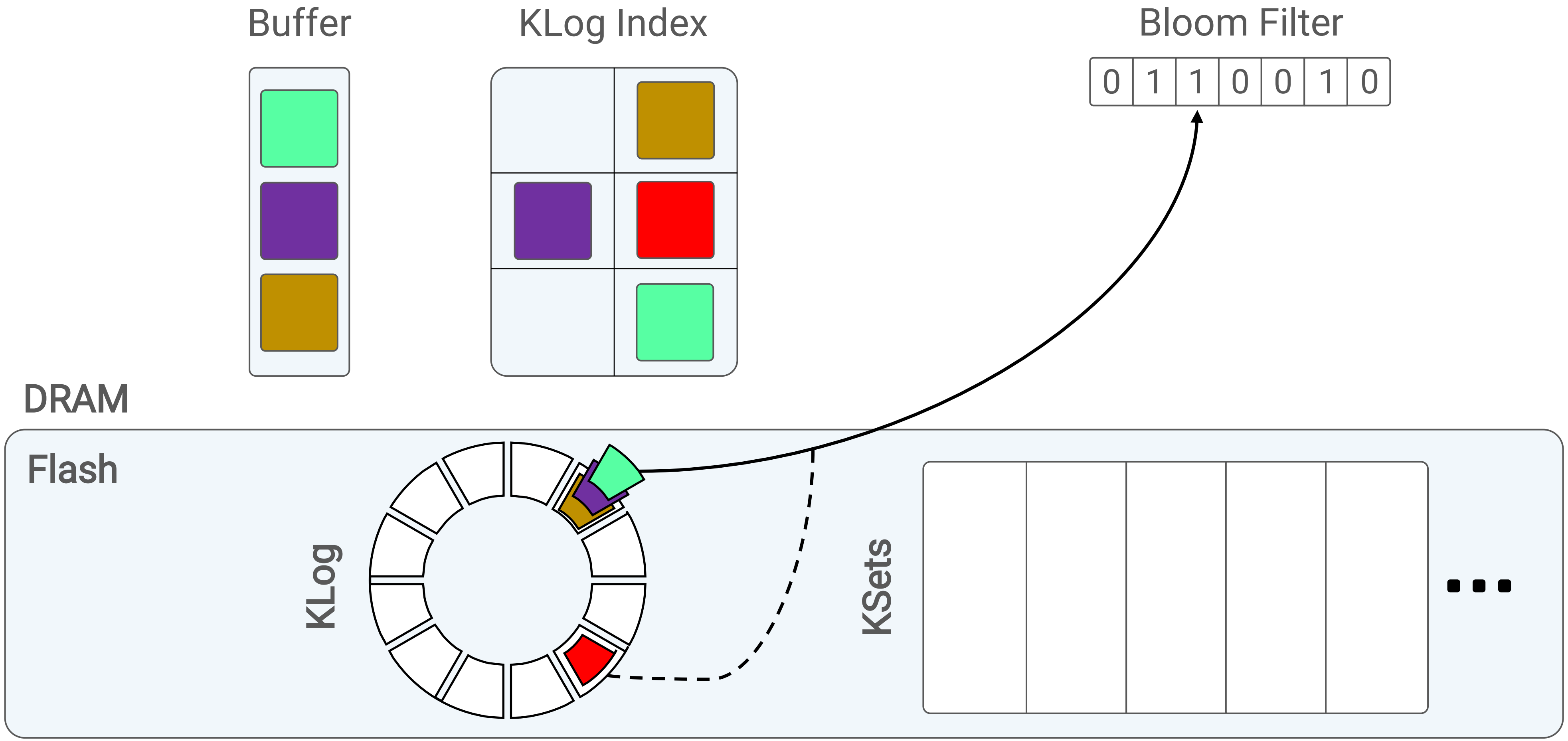


Kangaroo: Insert (KLog to KSets)

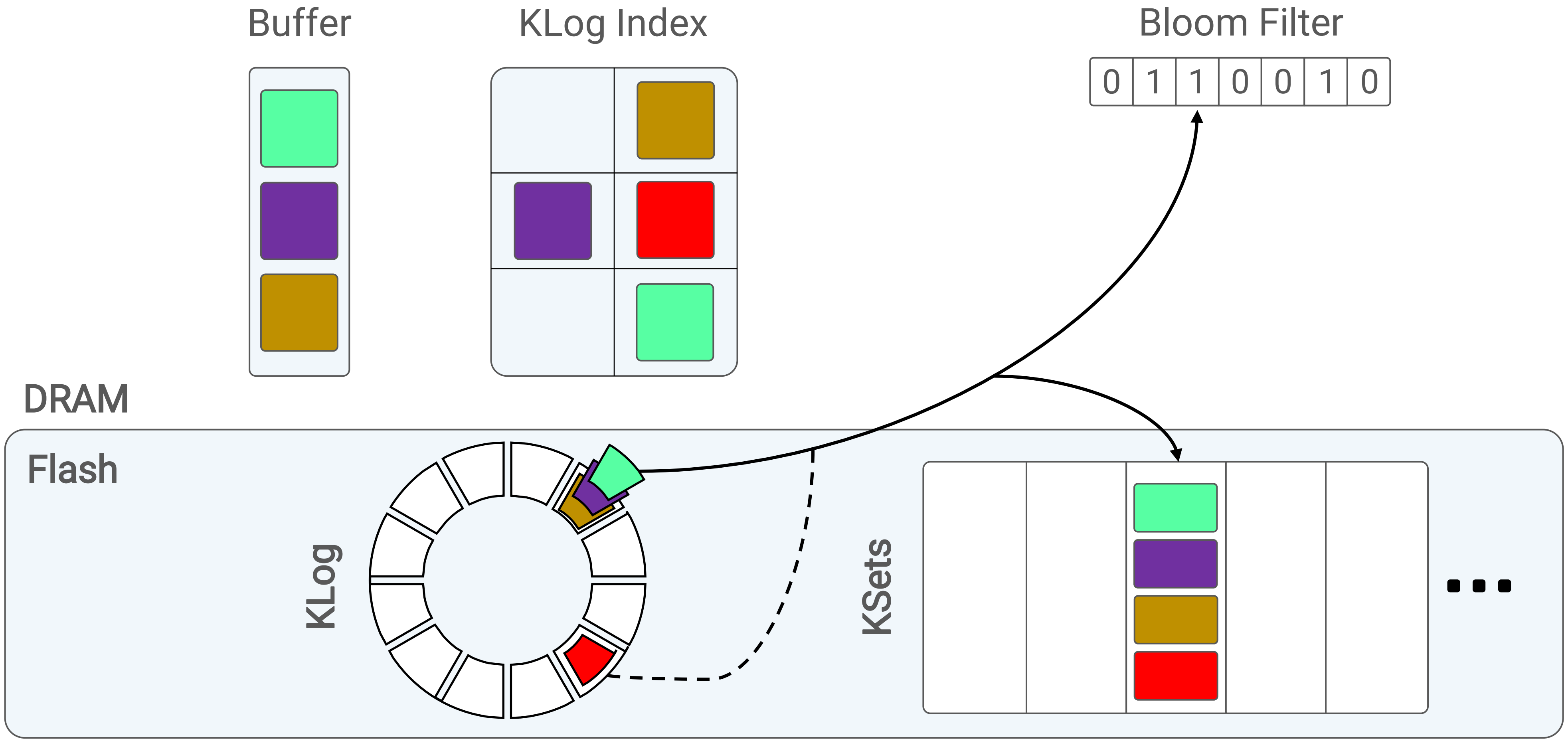
Kangaroo: Insert (KLog to KSets)



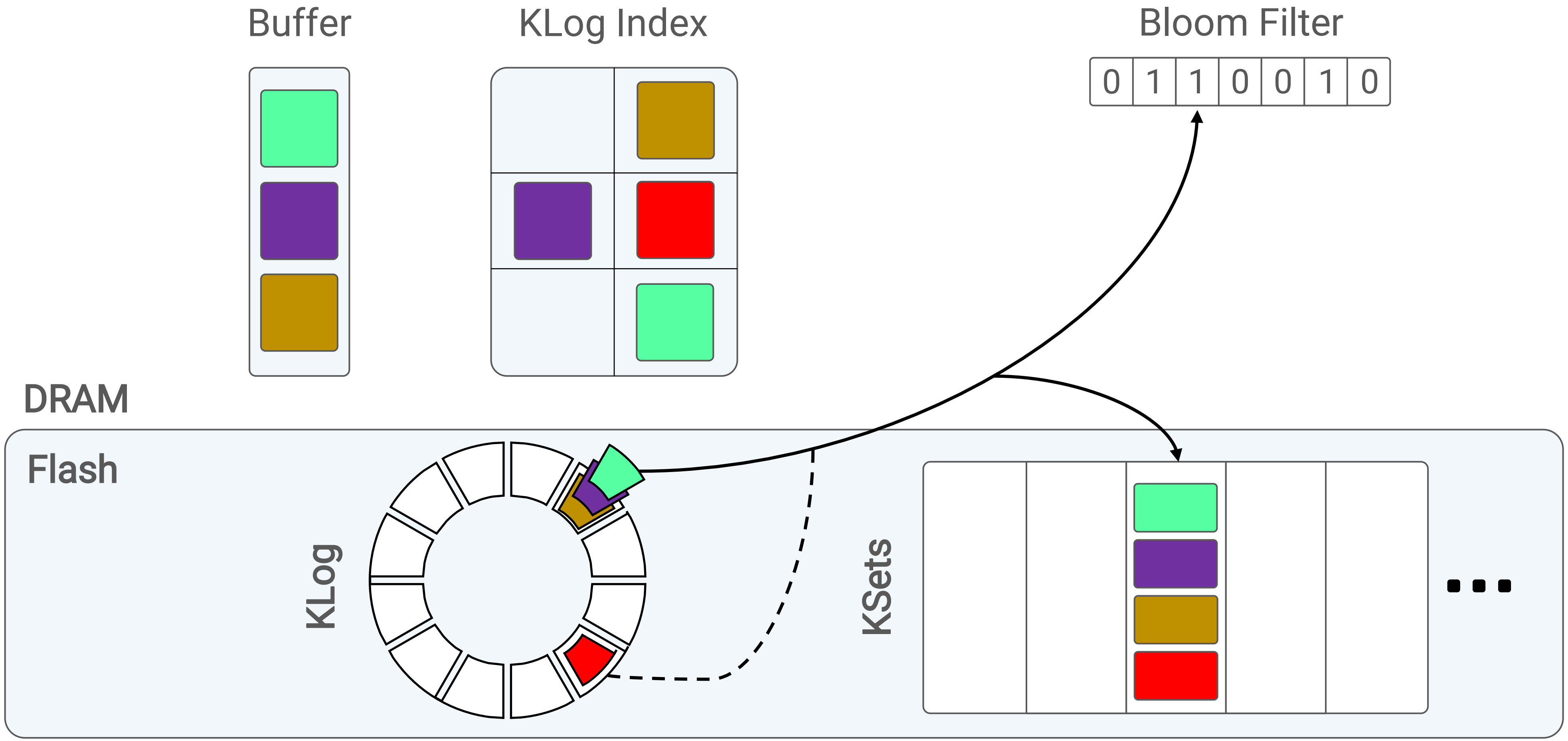
Kangaroo: Insert (KLog to KSets)



Kangaroo: Insert (KLog to KSets)



Kangaroo: Insert (KLog to KSets)



- Admission policies to reduce flash write

Kangaroo: Evaluation

Kangaroo: Evaluation (Miss Ratio)

Kangaroo: Evaluation (Miss Ratio)

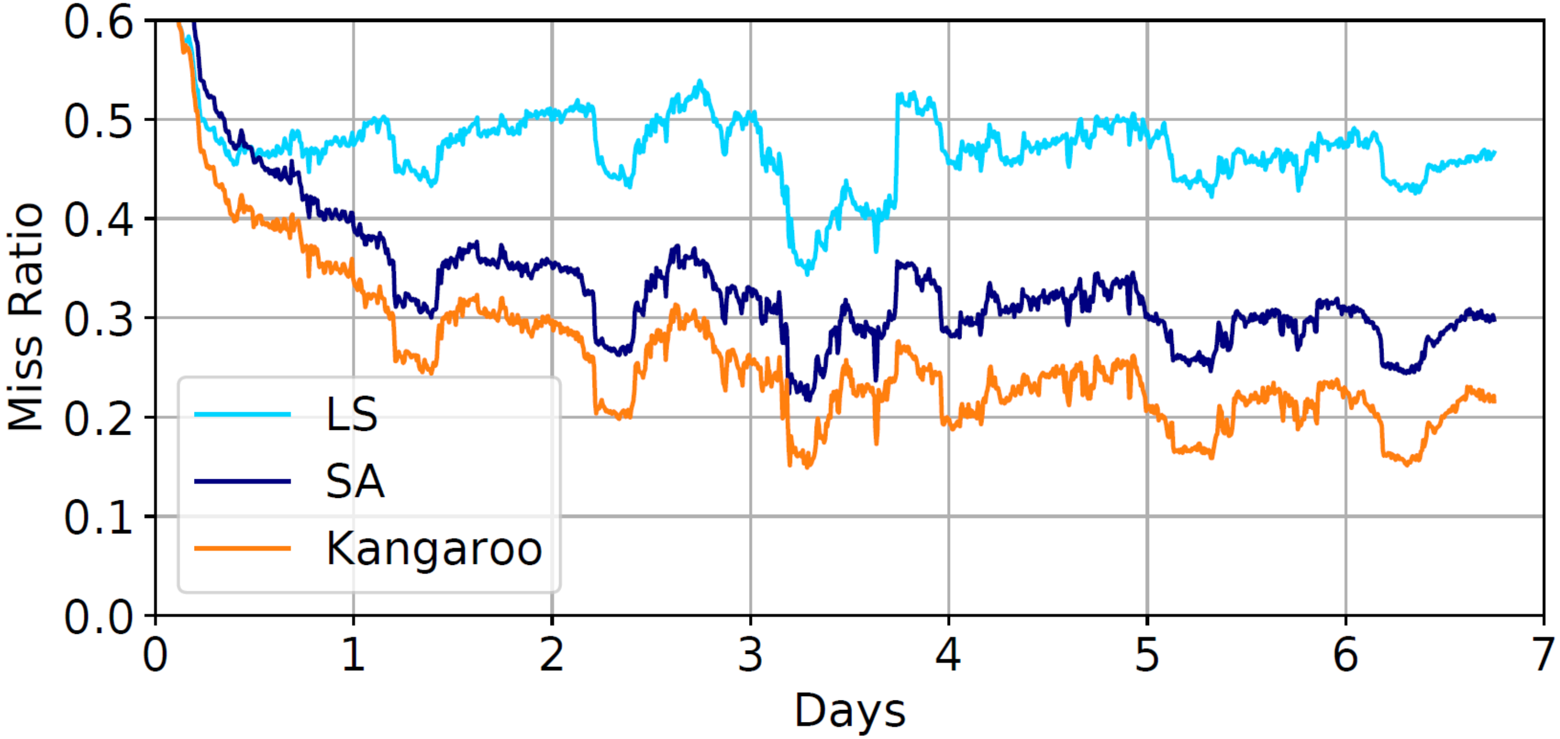


Fig: Miss ratio for all three systems over a 7-day Facebook trace. All systems are run with 16 GB DRAM, a 1.9 TB drive, and with write rates less than 62.5 MB/s

Kangaroo: Evaluation (Miss Ratio)

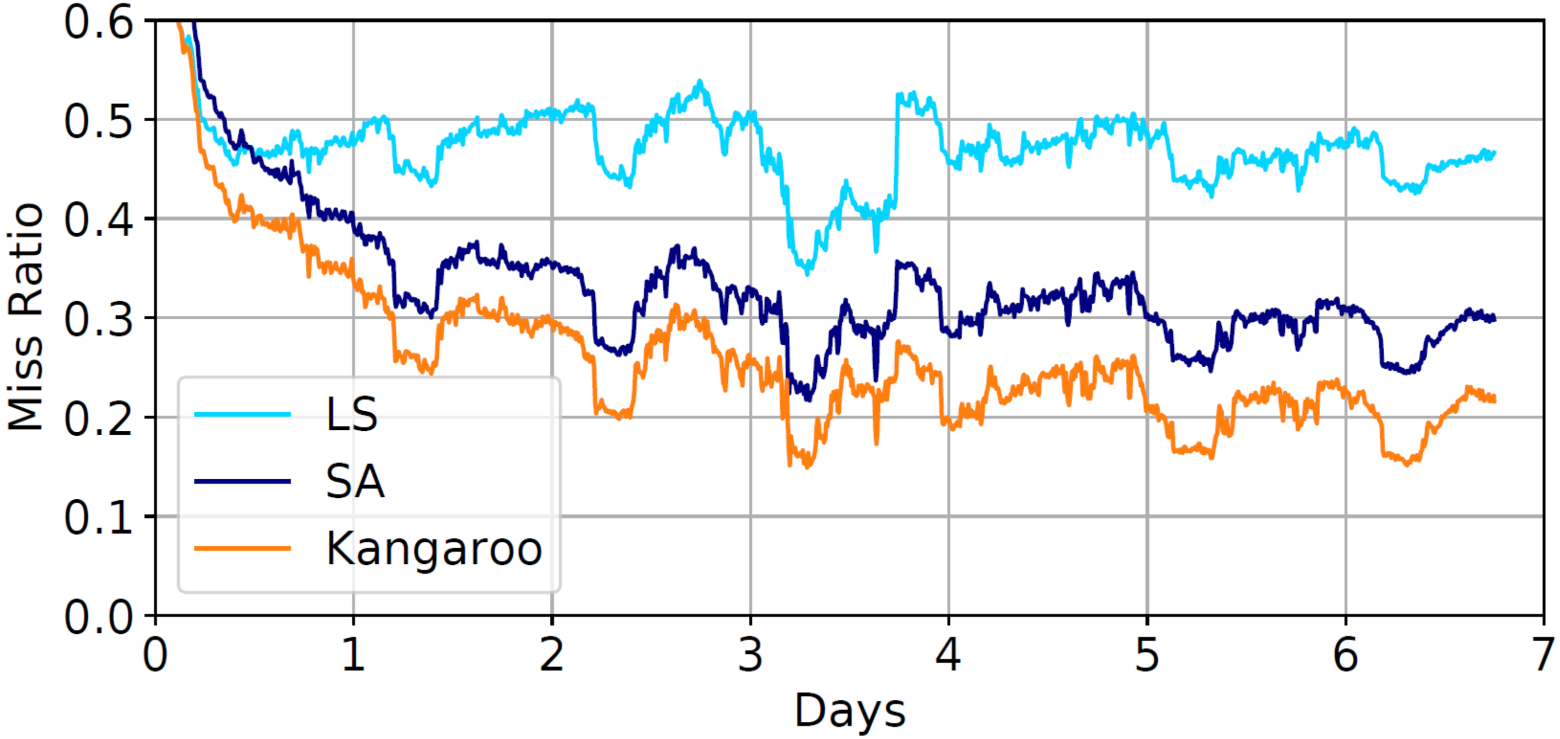
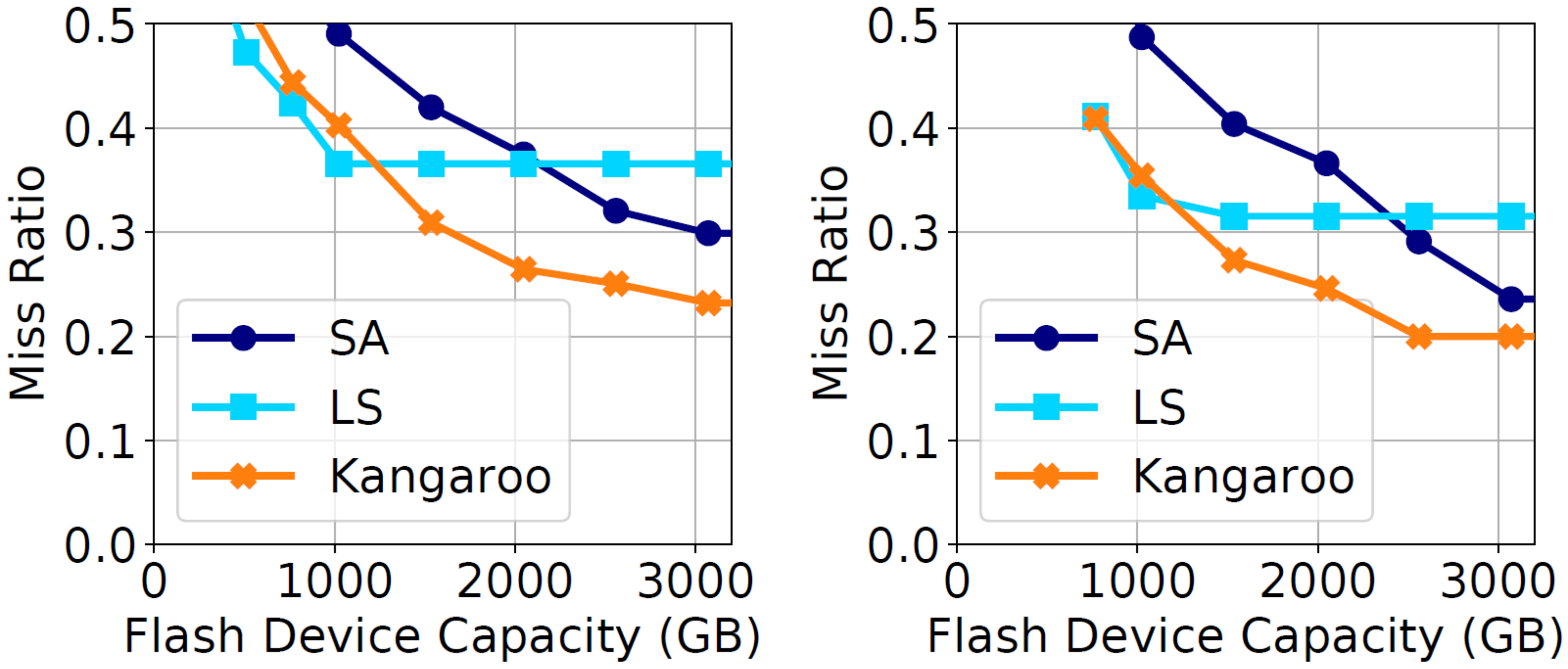


Fig: Miss ratio for all three systems over a 7-day Facebook trace. All systems are run with 16 GB DRAM, a 1.9 TB drive, and with write rates less than 62.5 MB/s

○ Kangaroo reduces cache misses by 29% vs. SA and by 56% vs. LS

Kangaroo: Evaluation (Flash Size)

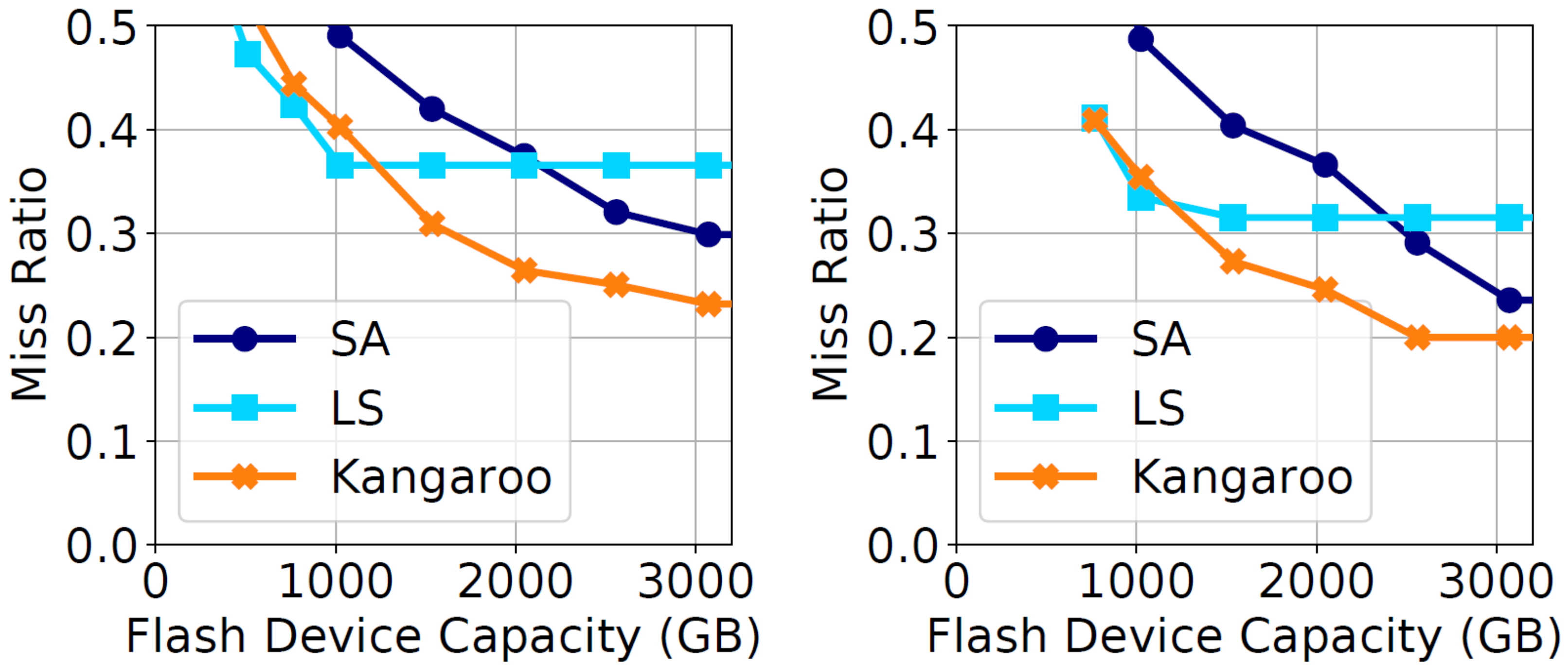
Kangaroo: Evaluation (Flash Size)



(a) Facebook

(b) Twitter

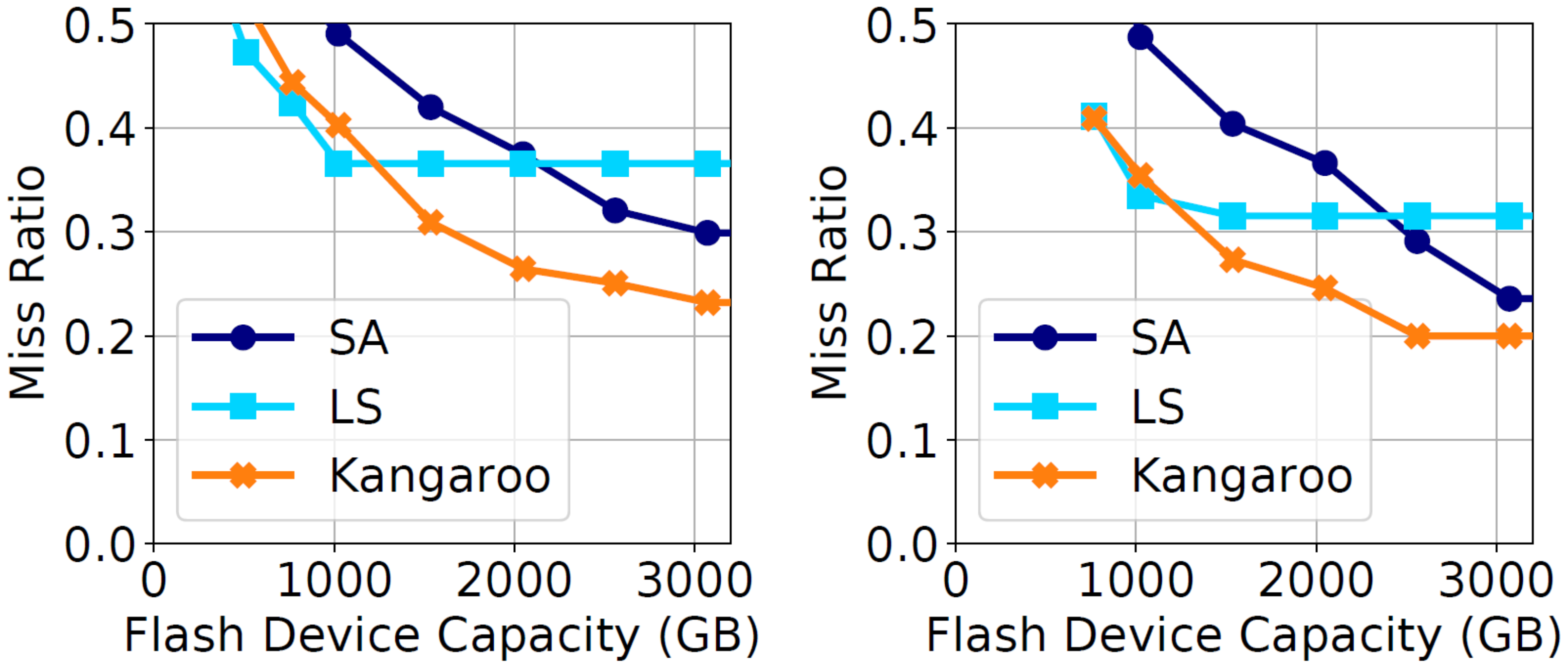
Kangaroo: Evaluation (Flash Size)



(a) Facebook ***(b)*** Twitter

- LS saturates faster: Fixed DRAM size

Kangaroo: Evaluation (Flash Size)

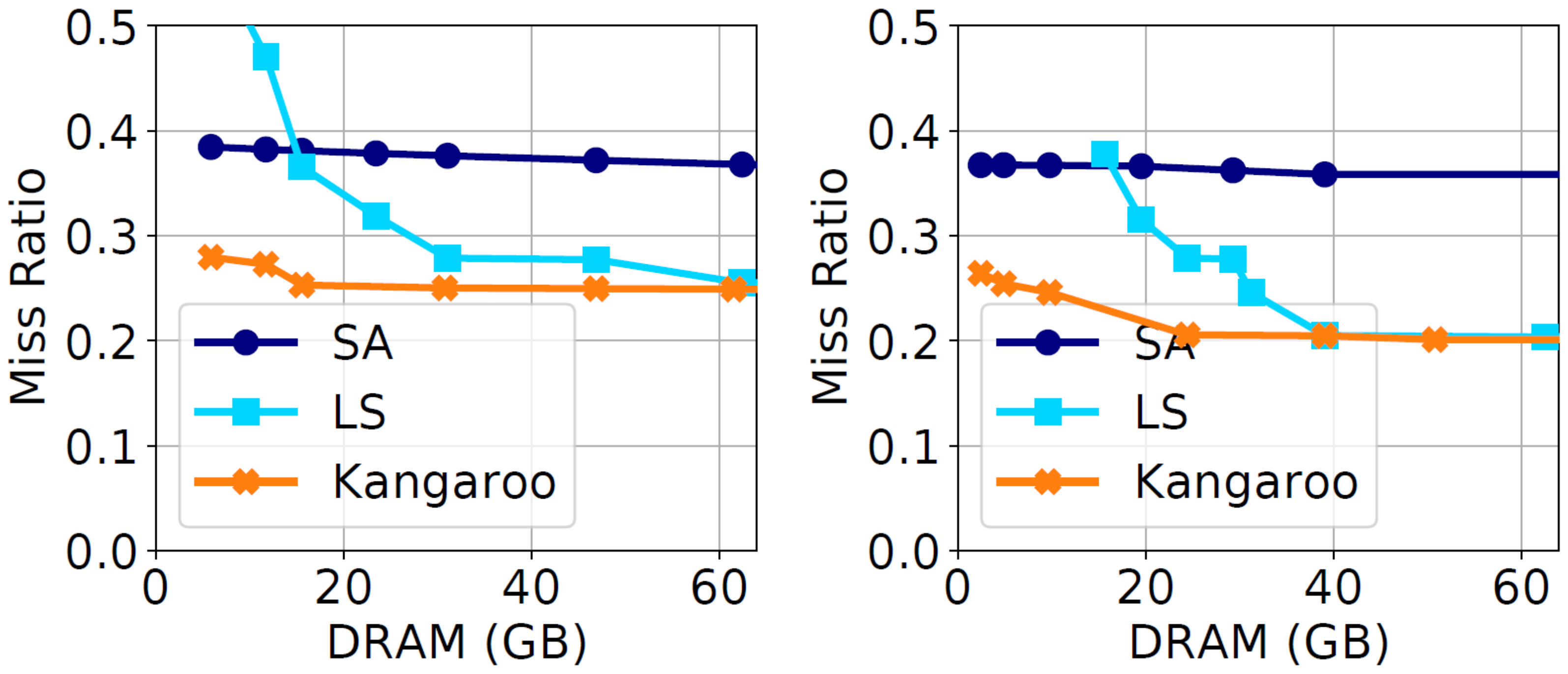


(a) Facebook

(b) Twitter

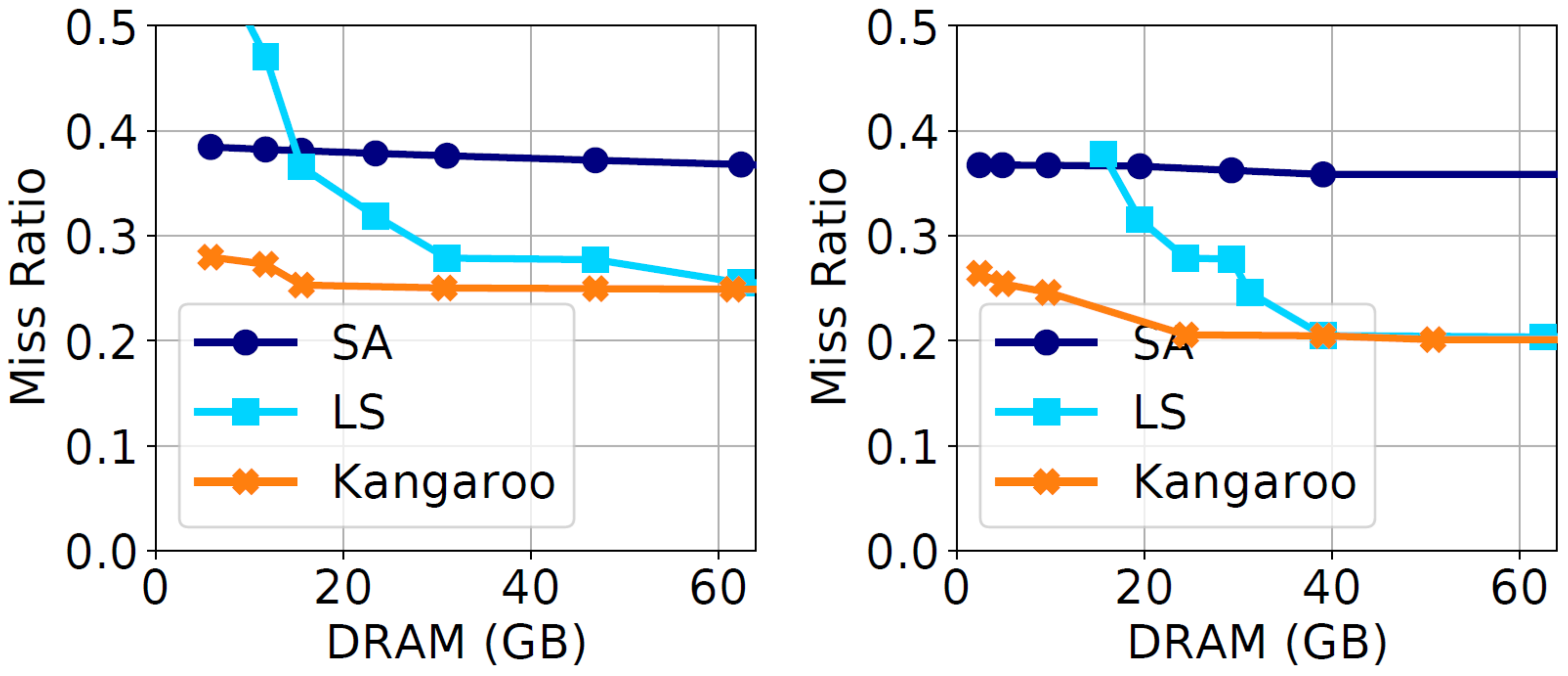
- LS saturates faster: Fixed DRAM size
- SA performs worst than Kangaroo: FIFO eviction and higher write amplification

Kangaroo: Evaluation (DRAM Size)



(a) Facebook *(b)* Twitter

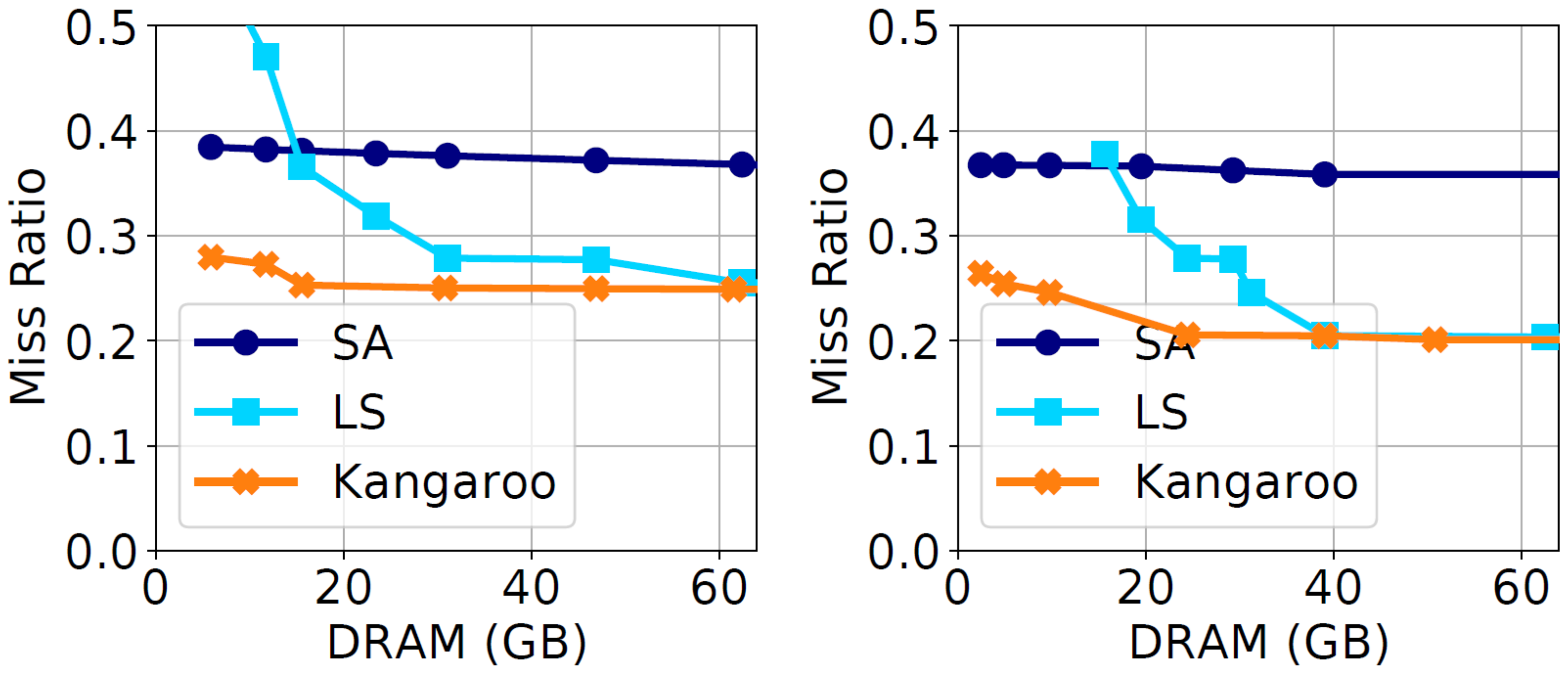
Kangaroo: Evaluation (DRAM Size)



(a) Facebook ***(b)*** Twitter

- ○ LS improves miss ratio: uses available DRAM

Kangaroo: Evaluation (DRAM Size)



(a) Facebook ***(b)*** Twitter

- LS improves miss ratio: uses available DRAM
- SA has no effect, however Kangaroo perform better than all

Conclusion

- Have advantages of both the designs, i.e. LS and SA

Conclusion

- Have advantages of both the designs, i.e. LS and SA

- Better miss rate than both the designs

Conclusion

- Have advantages of both the designs, i.e. LS and SA
- Better miss rate than both the designs
- Throughput and latency are not better than both designs but very well suited in production

Questions ?

Thank You