AniFilter: Parallel and Failure-Atomic Cuckoo Filter for Non-Volatile Memories

Hyungjun Oh[1], Bongki Cho[1], Changdae Kim[2], Heejin Park[1], **Jiwon Seo**[1]

[1] HANYANG UNIVERSITY [2] ETRI

Outline

- NVM and AMQs
- Cuckoo Filter
- Optimizations in AniFilter
  - Spillable Buckets
  - Lookahead Eviction
  - Bucket Primacy
- Logging and Recovery
- Evaluation

Non-Volatile Memories

- NVM Characteristics
  - High performance
  - Persistency
  - Byte-addressability

➔ Best of both DRAM and SSD (almost)

Approximate Membership Queries (AMQs)

- Approximate set data structures
- APIs
  - Insert(x) – inserts key x into the set
  - Lookup(x) – lookup key x and returns true or false
  - Delete(x) – removes key x from the set (optional)
- Small false-positives
  - lookup(x) true when x not in the set

NVM and AMQs

- NVMs are fast, but not as fast as DRAM
  - Read latency is 2~3x slower than DRAM

➔ DRAM versions of AMQs run slow on NVM

- AMQs' operations are cheap
  - Insert() and Lookup() need only handful of computation

➔ Cannot use complicated optimization techniques

Cuckoo Filter

- Fingerprint-based AMQ
- Bucketized Cuckoo Filter

➔ Each bucket has four slots

➔ Two hashes (H_1, H_2) for a bucket index

0	05A	0F2	0E9	000
1	600	27D	000	000
2	2F0	000	000	000
3	12D	A15	1EF	AFE
4	14A	FA0	5B0	000
5	1AA	5B0	000	000

Cuckoo Filter

- Fingerprint-based AMQ
- Bucketized Cuckoo Filter

➔ Each bucket has four slots

➔ Two hashes (H_1, H_2) for a bucket index

Bucket

0	05A	0F2	0E9	000
1	600	27D	000	000
2	2F0	000	000	000
3	12D	A15	1EF	AFE
4	14A	FA0	5B0	000
5	1AA	5B0	000	000

Slots

Cuckoo Filter

- Fingerprint-based AMQ
- Bucketized Cuckoo Filter

➔ Each bucket has four slots

➔ Two hashes (H_1, H_2) for a bucket index

- Insertion (without eviction)

Key x $\xrightarrow{H_1(x)}$

0	05A	0F2	0E9	000
1	600	27D	000	000
2	2F0	000	000	000
3	12D	A15	1EF	AFE
4	14A	FA0	5B0	000
5	1AA	5B0	000	000

Cuckoo Filter

- Fingerprint-based AMQ
- Bucketized Cuckoo Filter

➔ Each bucket has four slots

➔ Two hashes (H_1, H_2) for a bucket index

- Insertion (without eviction)

Key x $\xrightarrow{H_1(x)}$ 0

0	05A	0F2	0E9	AC0
1	600	27D	000	000
2	2F0	000	000	000
3	12D	A15	1EF	AFE
4	14A	FA0	5B0	000
5	1AA	5B0	000	000

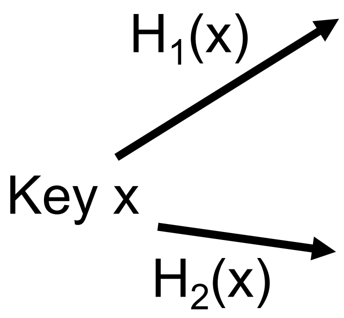
Cuckoo Filter

- Fingerprint-based AMQ
- Bucketized Cuckoo Filter

➔ Each bucket has four slots

➔ Two hashes (H_1, H_2) for a bucket index

- Insertion (with eviction)



0	05A	0F2	0E9	AC0
1	600	27D	000	000
2	2F0	000	000	000
3	12D	A15	1EF	AFE
4	14A	FA0	5B0	000
5	1AA	5B0	000	000

Cuckoo Filter

- Fingerprint-based AMQ
- Bucketized Cuckoo Filter

➔ Each bucket has four slots

➔ Two hashes (H_1, H_2) for a bucket index

- Insertion (with eviction)

Cuckoo Filter

- Fingerprint-based AMQ
- Bucketized Cuckoo Filter

➔ Each bucket has four slots

➔ Two hashes (H_1, H_2) for a bucket index

- Insertion (with eviction)

Cuckoo Filter

- Fingerprint-based AMQ
- Bucketized Cuckoo Filter

➔ Each bucket has four slots

➔ Two hashes (H_1, H_2) for a bucket index

- Lookup operation

Key x

$H_1(x)$

$H_2(x)$

0	05A	0F2	0E9	AC0
1	600	27D	000	000
2	2F0	000	000	000
3	12D	A15	1EF	AFE
4	14A	FA0	5B0	000
5	1AA	5B0	000	000

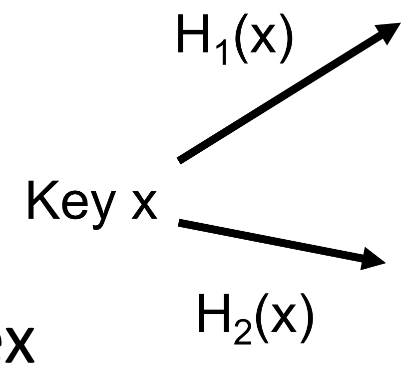
Cuckoo Filter

- Fingerprint-based AMQ
- Bucketized Cuckoo Filter

➔ Each bucket has four slots

➔ Two hashes (H_1, H_2) for a bucket index

- Lookup operation



0	05A	0F2	0E9	AC0
1	600	27D	000	000
2	2F0	000	000	000
3	12D	A15	1EF	AFE
4	14A	FA0	5B0	000
5	1AA	5B0	000	000

Cuckoo Filter Issue

1) Eviction overhead in high load factors (>75%)

- Worse in NVM with higher latency

2) Failure-atomicity issue

- Typical setting: 4 slots, 12 bit fingerprints
- A bucket is 6 bytes
- NVM's atomic write unit: 8 byte

6 bytes

0	05A	0F2	0E9	AC0
1	600	27D	000	000
2	2F0	000	000	000
3	12D	A15	1EF	AFE

Cuckoo Filter Issue

1) Eviction overhead in high load factors (>75%)

- Worse in NVM with higher latency

2) Failure-atomicity issue

- Typical setting: 4 slots, 12 bit fingerprints
- A bucket is 6 bytes
- NVM’s atomic write unit: 8 byte

8 byte boundaries

0	05A	0F2	0E9	AC0
1	600	27D	000	000
2	2F0	000	000	000
3	12D	A15	1EF	AFE

AniFilter*

- Cuckoo Filter optimized for NVM
- Optimization techniques
  - Spillable Buckets
  - Lookahead Evictions
  - Bucket Primacy
- Failure-atomic with minimal logging

* Anis are in the cuckoo family and have communal nests.

Spillable Buckets

- Spill a fingerprint in next 2 buckets
- Only spill in the first slot

A0F	D1F	E49	F8A
000	27D	2A0	F09
000	000	000	000
000	A15	1EF	AFE
A4A	DA1	EC0	F02
000	5B0	B8A	CAC

Spillable Buckets

- Spill a fingerprint in next 2 buckets
- Only spill in the first slot

Key x

H_1 (x)

A0F	D1F	E49	F8A
000	27D	2A0	F09
000	000	000	000
000	A15	1EF	AFE
A4A	DA1	EC0	F02
000	5B0	B8A	CAC

Spillable Buckets

- Spill a fingerprint in next 2 buckets
- Only spill in the first slot

Key x

Spill FP_x

A0F	D1F	E49	F8A
000	27D	2A0	F09
000	000	000	000
000	A15	1EF	AFE
A4A	DA1	EC0	F02
080	**B8A** >	**5B0**	CAC

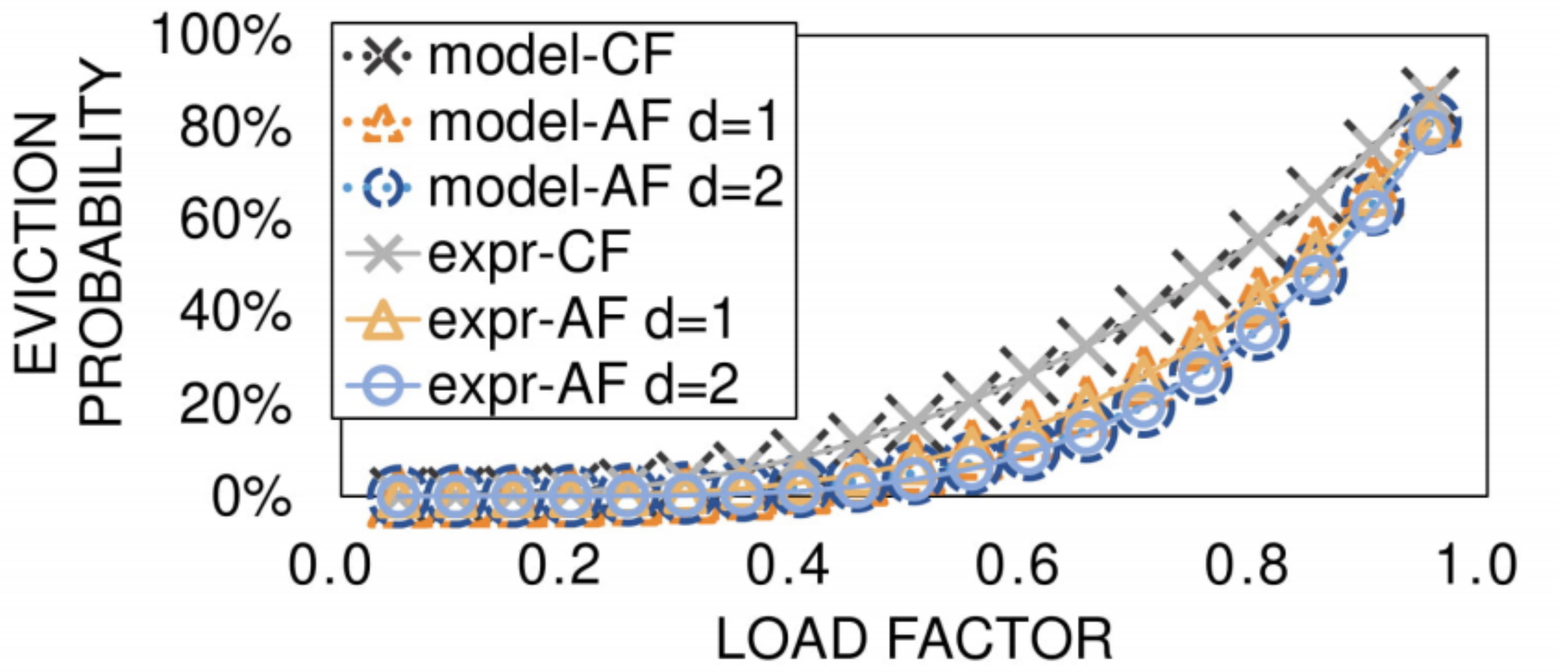
Swapped to encode spill

Spillable Buckets – Theoretic Analysis

- Eviction probability with and without Spillable Buckets
- Probabilistic model to compute Prob(X=k)

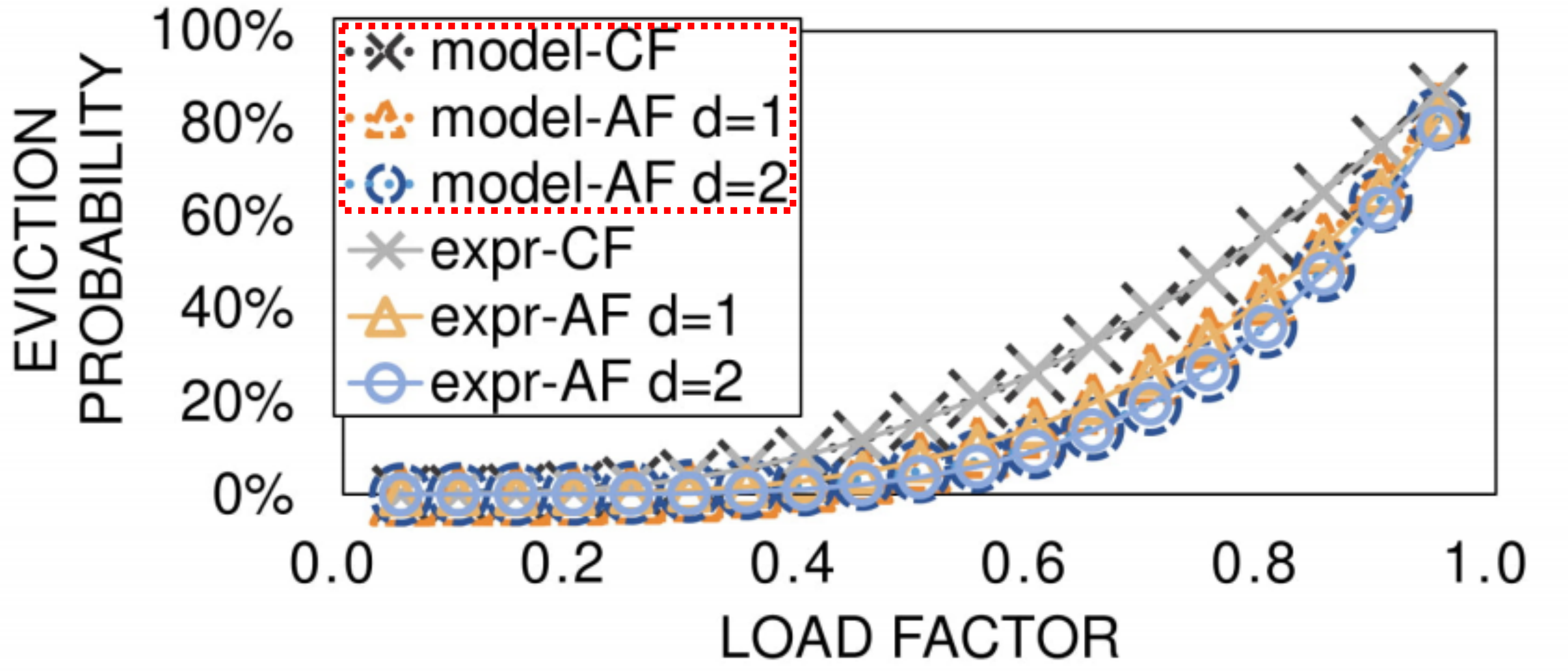
Spillable Buckets – Theoretic Analysis

- Eviction probability with and without Spillable Buckets
- Probabilistic model to compute Prob(X=k)



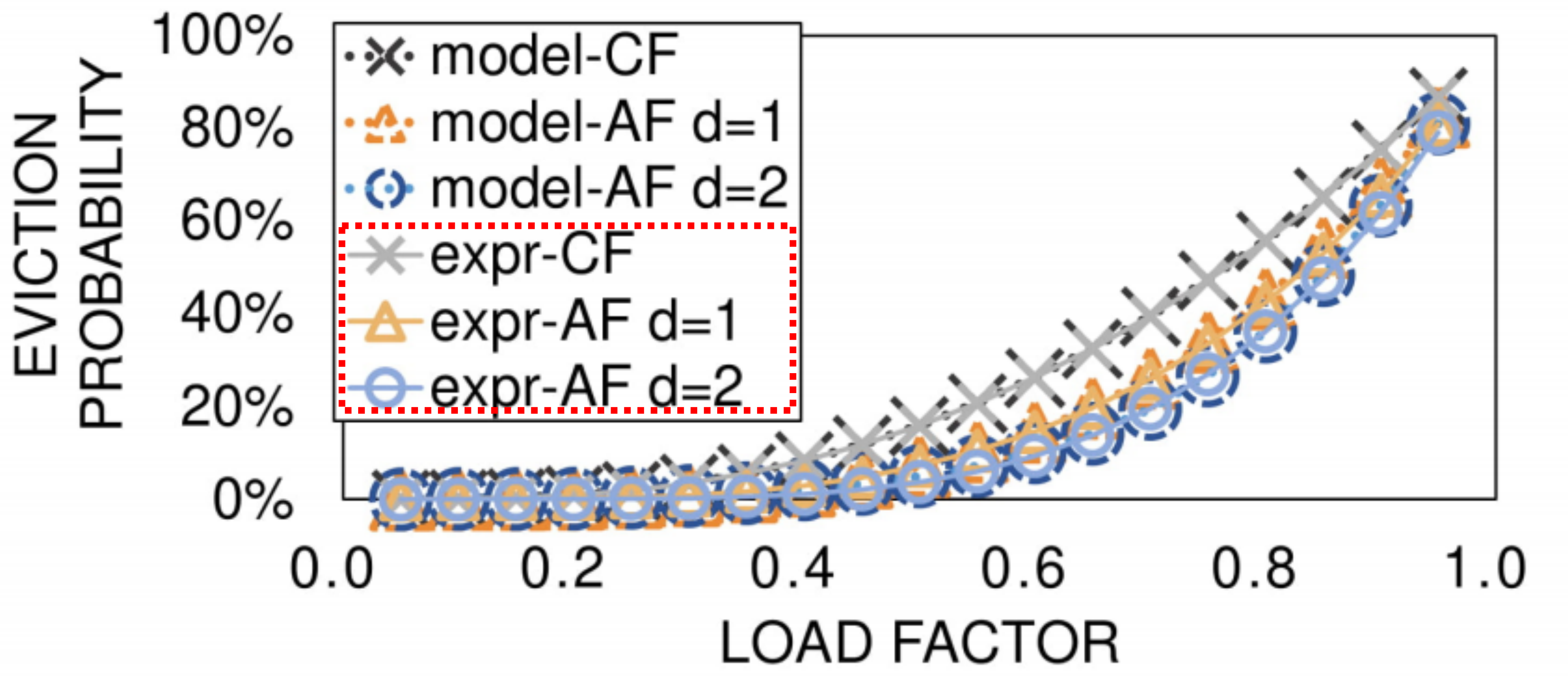
Spillable Buckets – Theoretic Analysis

- Eviction probability with and without Spillable Buckets
- Probabilistic model to compute Prob(X=k)



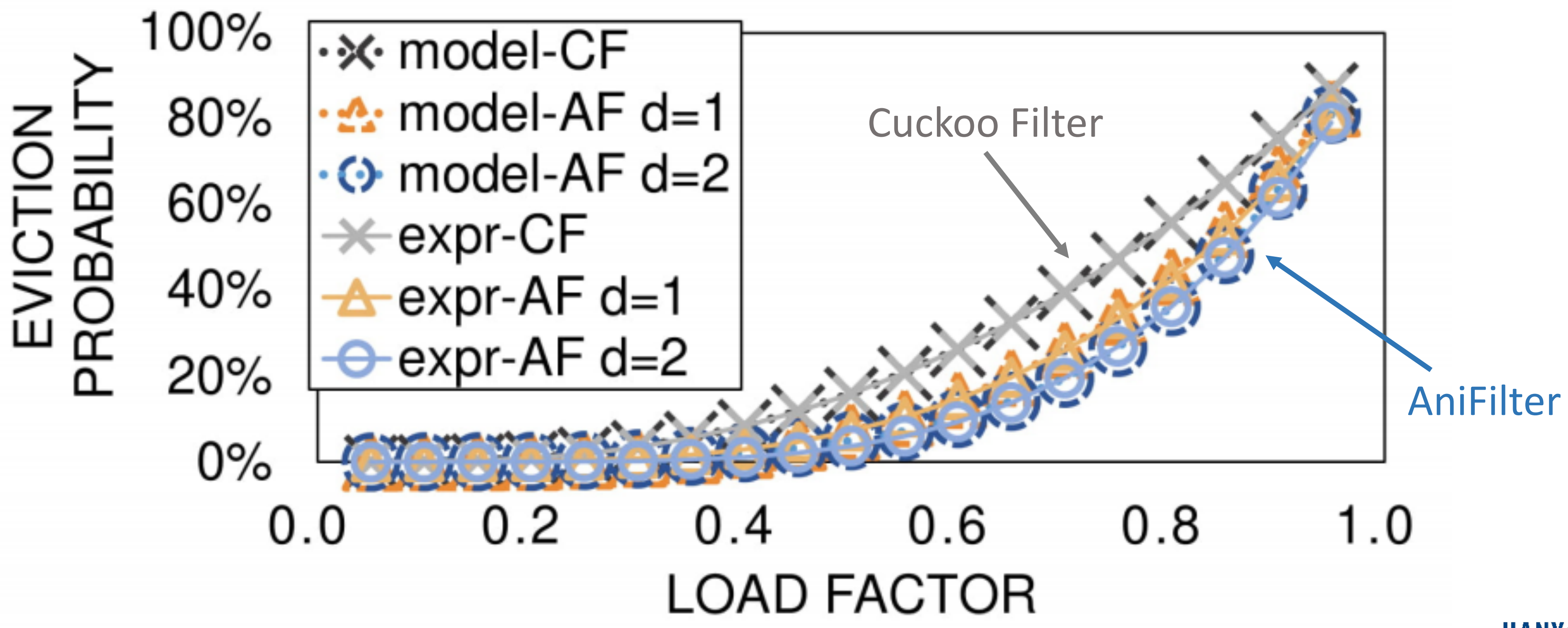
Spillable Buckets – Theoretic Analysis

- Eviction probability with and without Spillable Buckets
- Probabilistic model to compute Prob(X=k)



Spillable Buckets – Theoretic Analysis

- Eviction probability with and without Spillable Buckets
- Probabilistic model to compute Prob(X=k)



Lookahead Eviction

- Evict a fingerprint that does not incur further eviction

A0F	D1F	E49	F8A
000	27D	2A0	F09
000	000	000	000
000	A15	1EF	AFE
A4A	DA1	EC0	F02
080	B8A	5B0	CAC

Occupancy flags

1
0
0
0
1
1

Bucket Primacy

- Primary bucket (H_1) and secondary bucket (H_2)

Swapped to encode overflow

A0F	D1F	**F8A**	>	**E49**
1A0	27D	2A0	<	F09
000	000	000		000
000	A15	1EF		AFE
A4A	DA1	**F02**	>	**EC0**
080	B8A	5B0	<	CAC

Buckets previously not overflown

Logging for Failure Atomicity

- Type-A, -B, -C buckets
- Requires different # of loggings
- Logging example for Type-B buckets

A0F	D1F	F8A	E49	Type A
2A0	27D	000	000	Type B
AC7	000	000	000	Type C
009	A15	1EF	AFE	Type A

Logging for Failure Atomicity

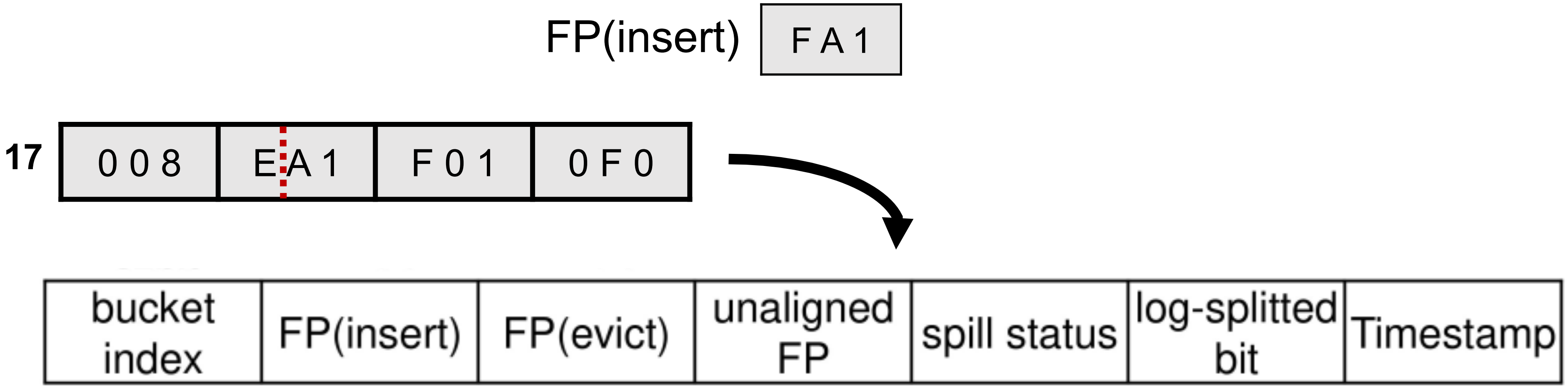
- Type-A, -B, -C buckets
- Requires different # of loggings
- Logging example for Type-B buckets

- 8-byte logging record

32bit	12bit	12bit	4bit	1bit	1bit	2bit
bucket index	FP(insert)	FP(evict)	unaligned FP	spill status	log-splitted bit	Timestamp

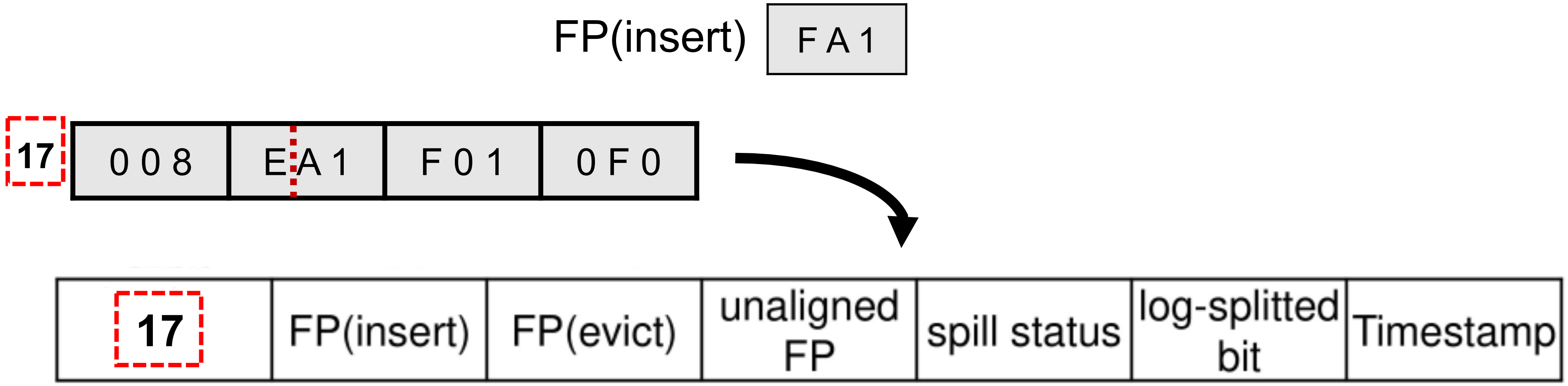
Logging for Failure Atomicity

- Type-A, -B, -C buckets
- Requires different # of loggings
- Logging example for Type-B buckets



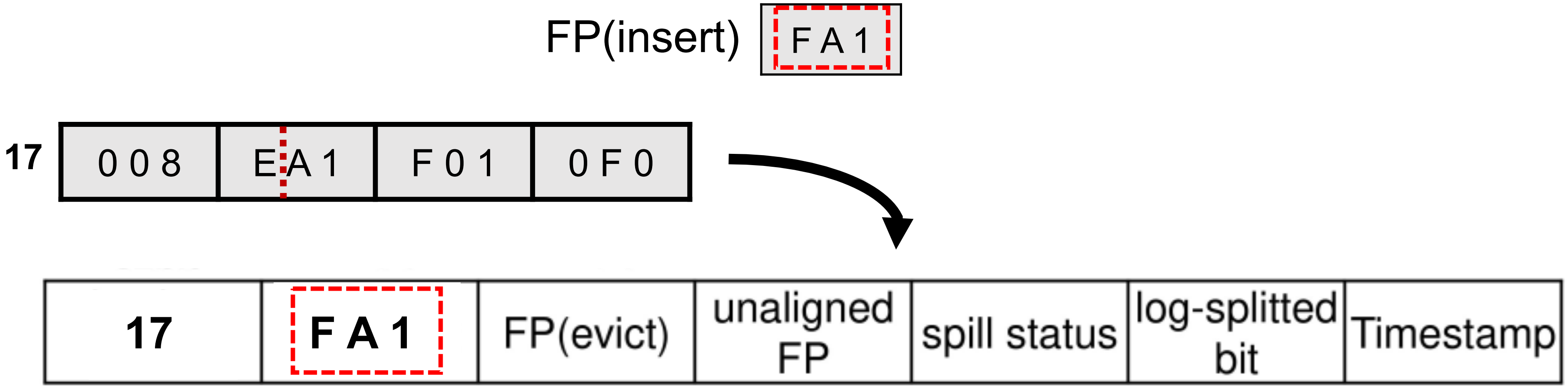
Logging for Failure Atomicity

- Type-A, -B, -C buckets
- Requires different # of loggings
- Logging example for Type-B buckets



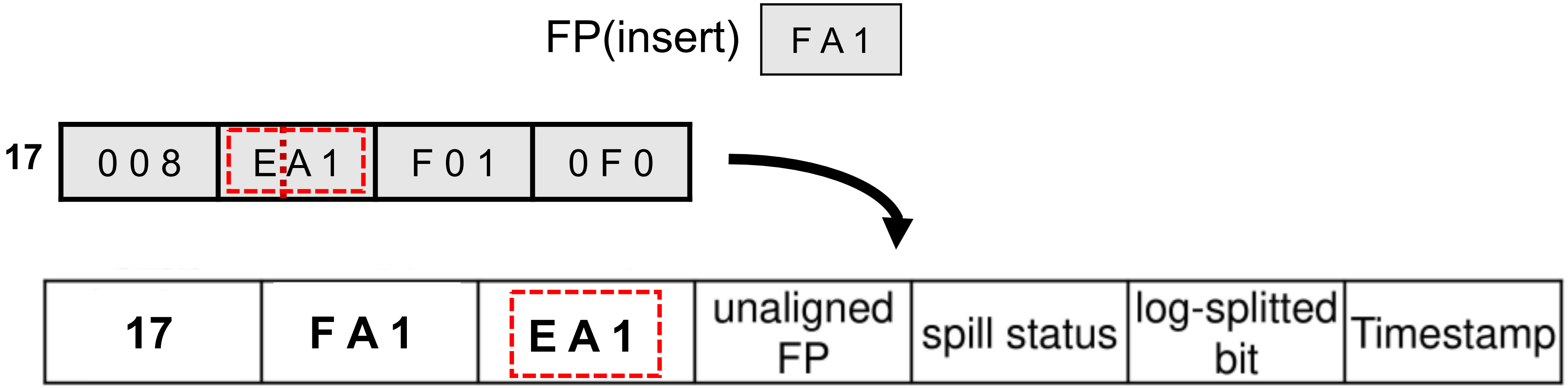
Logging for Failure Atomicity

- Type-A, -B, -C buckets
- Requires different # of loggings
- Logging example for Type-B buckets



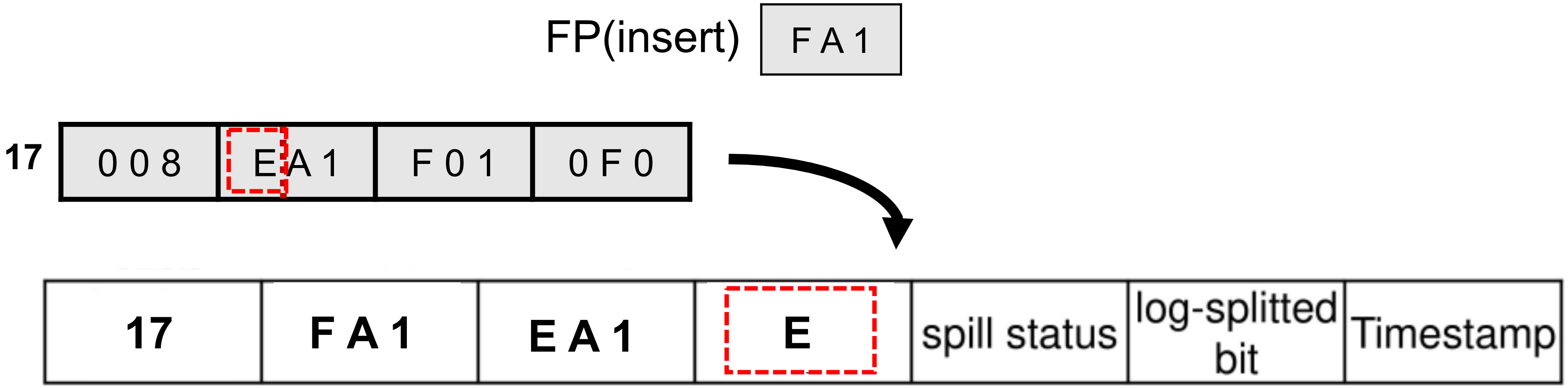
Logging for Failure Atomicity

- Type-A, -B, -C buckets
- Requires different # of loggings
- Logging example for Type-B buckets



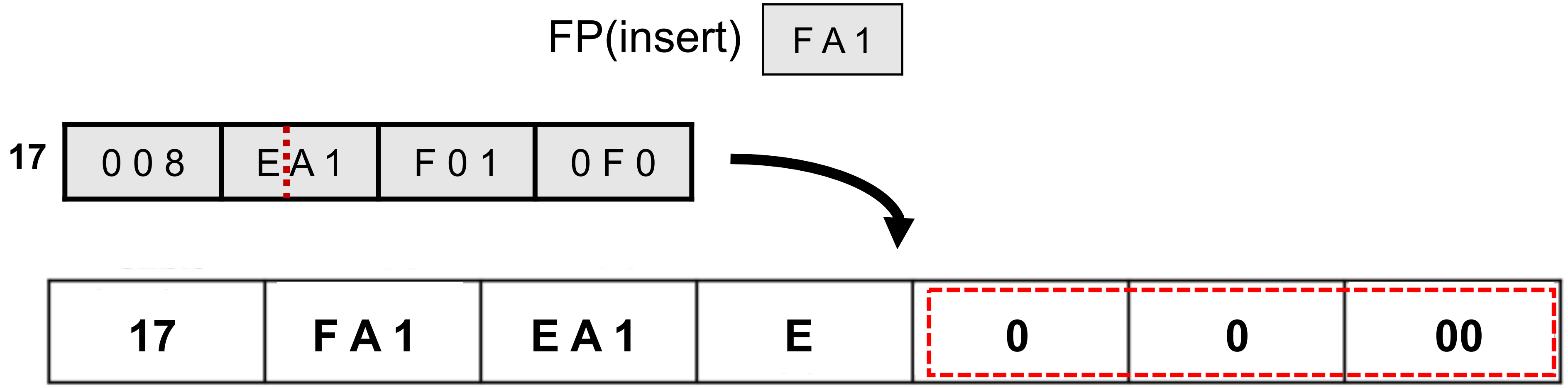
Logging for Failure Atomicity

- Type-A, -B, -C buckets
- Requires different # of loggings
- Logging example for Type-B buckets



Logging for Failure Atomicity

- Type-A, -B, -C buckets
- Requires different # of loggings
- Logging example for Type-B buckets



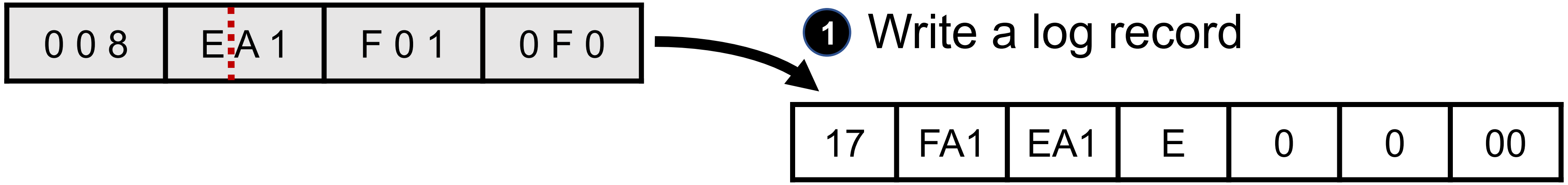
Recovery

1. Check log record and read FP_i, FP_e, and bucket B
2. Test if FP_i and FP_e are in B
   a) $FP_e \in B$ and $FP_i \in B$
   b) $FP_e \notin B$ and $FP_i \notin B$
   c) $FP_e \in B$ and $FP_i \notin B$
   d) $FP_e \notin B$ and $FP_i \notin B$
3. For (a), (b), (c) insertion is incomplete

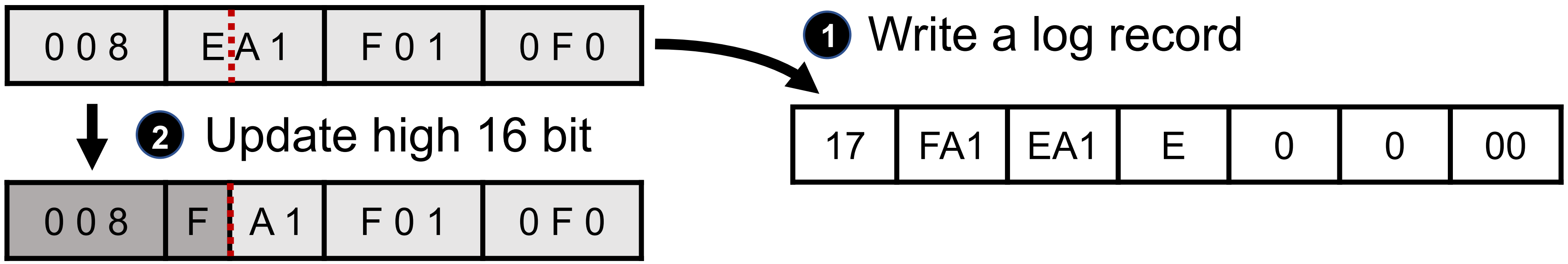
   For (d) examine meta-data in log record

➔ Example recovery process for (d)

Recovery – Example

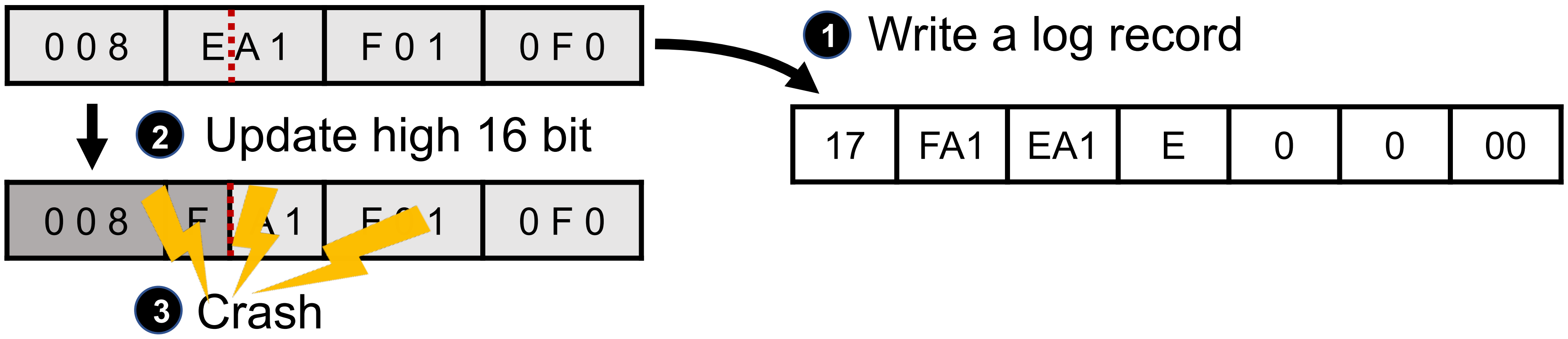


Recovery



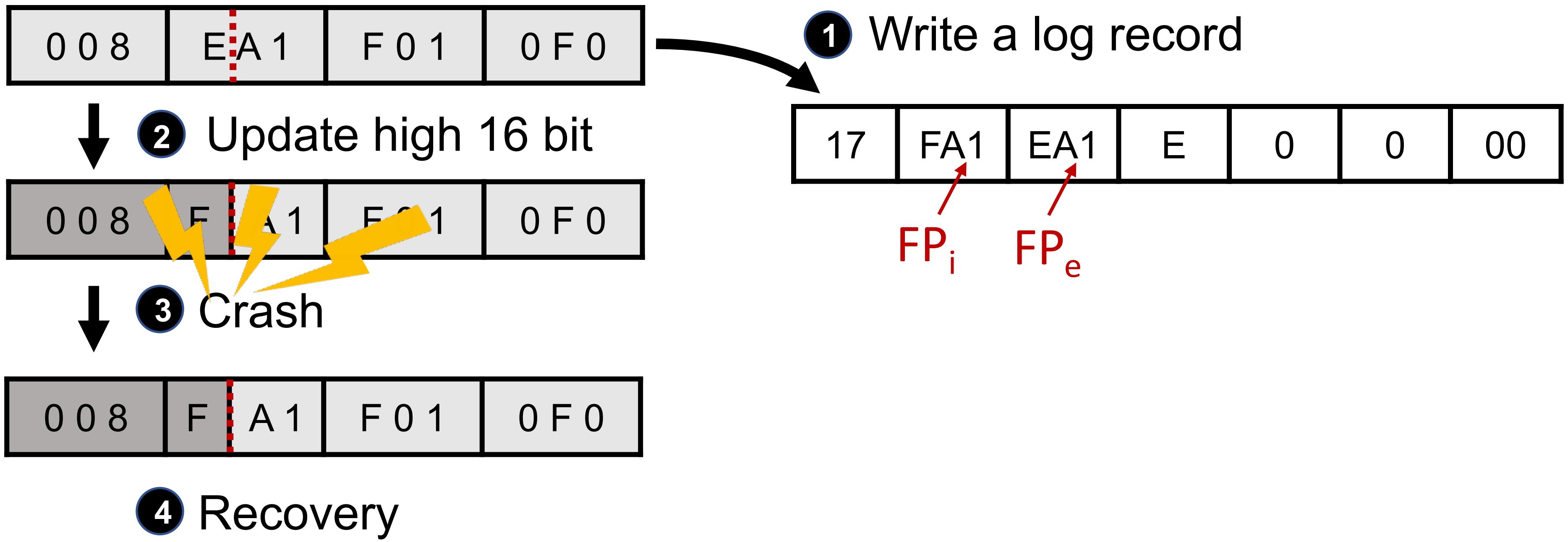
0 0 8
E A 1
F 0 1
0 F 0
1 Write a log record
17
FA1
EA1
E
0
0
00
2 Update high 16 bit
0 0 8
F
A 1
F 0 1
0 F 0

Recovery

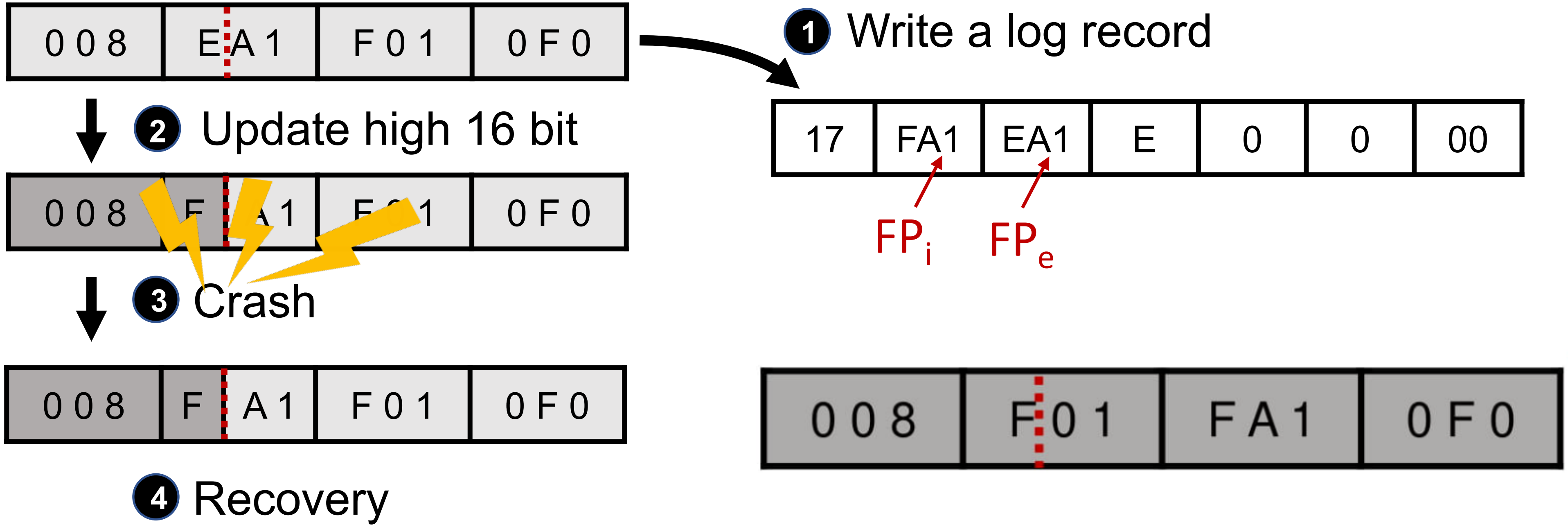


0 0 8
E A 1
F 0 1
0 F 0
1 Write a log record
17
FA1
EA1
E
0
0
00
2 Update high 16 bit
0 0 8
F
A 1
0 F 0
3 Crash

Recovery

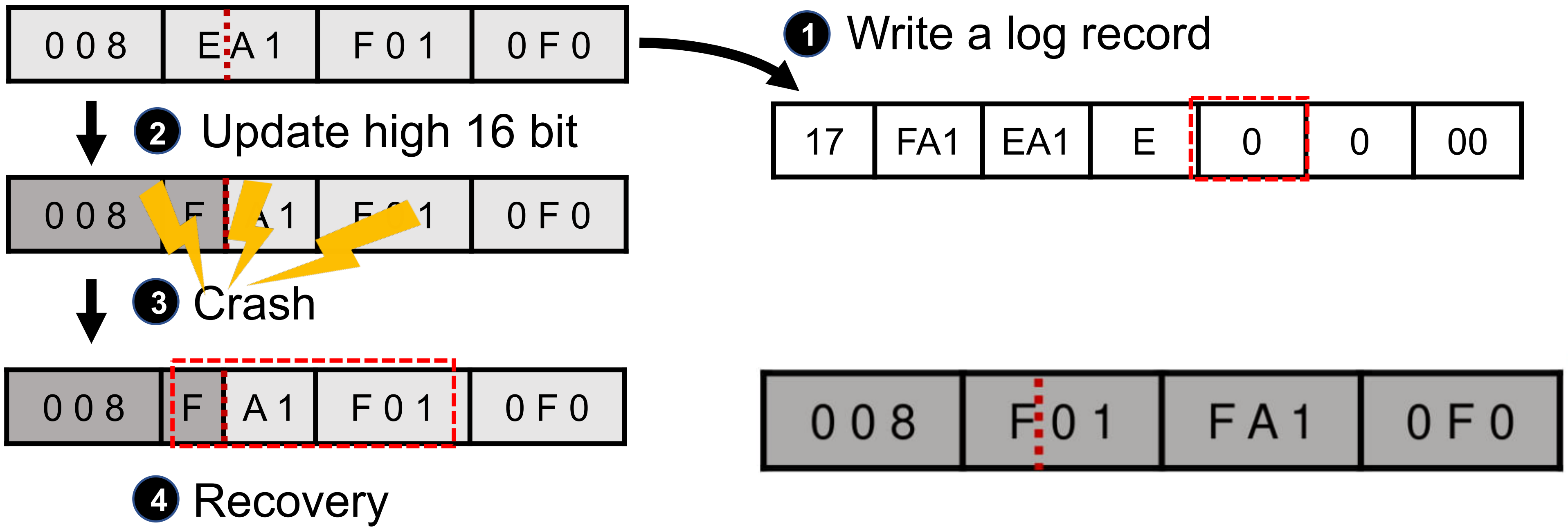

0 0 8
E A 1
F 0 1
0 F 0
Update high 16 bit
0 0 8
0 F 0
Crash
0 0 8
F
A 1
F 0 1
0 F 0
Recovery
Write a log record
17
FA1
EA1
E
0
0
00
FPi
FPe

Recovery

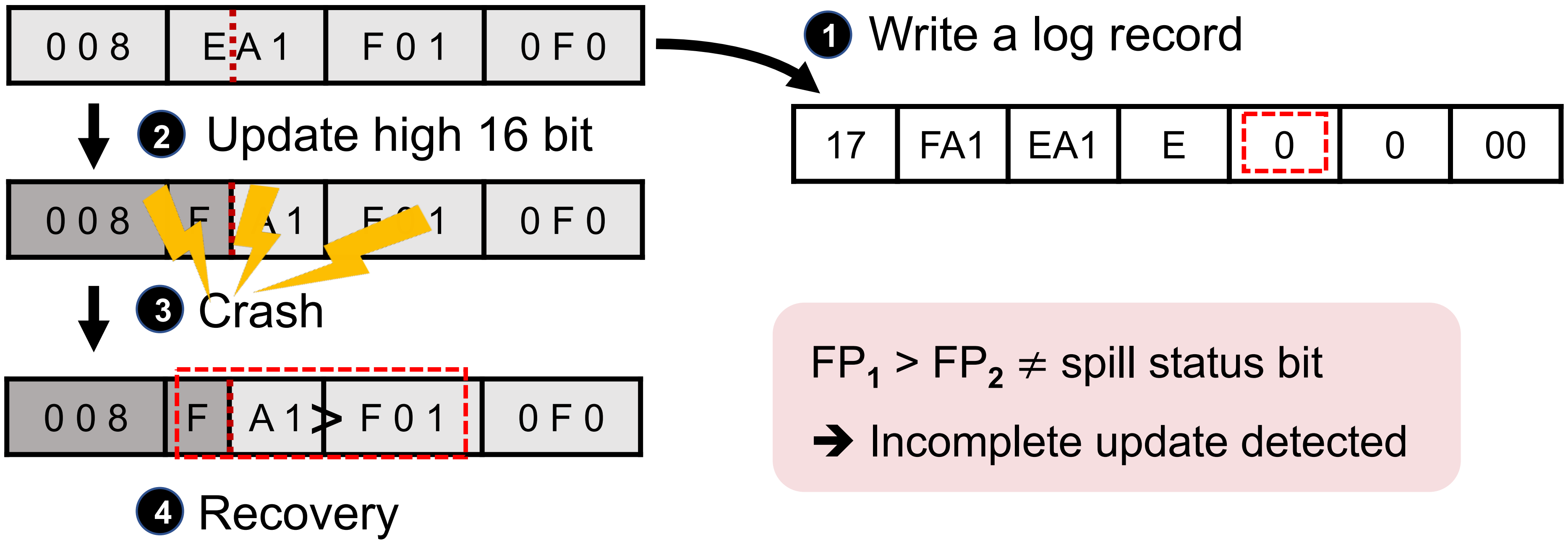


0 0 8
E A 1
F 0 1
0 F 0
1 Write a log record
17
FA1
EA1
E
0
0
00
FP i
FP e
2 Update high 16 bit
0 0 8
F A 1
0 F 0
3 Crash
0 0 8
F A 1
F 0 1
0 F 0
4 Recovery
0 0 8
F 0 1
F A 1
0 F 0

Recovery


0 0 8
E A 1
F 0 1
0 F 0
1 Write a log record
17
FA1
EA1
E
0
0
00
2 Update high 16 bit
3 Crash
4 Recovery
0 0 8
F 0 1
F A 1
0 F 0

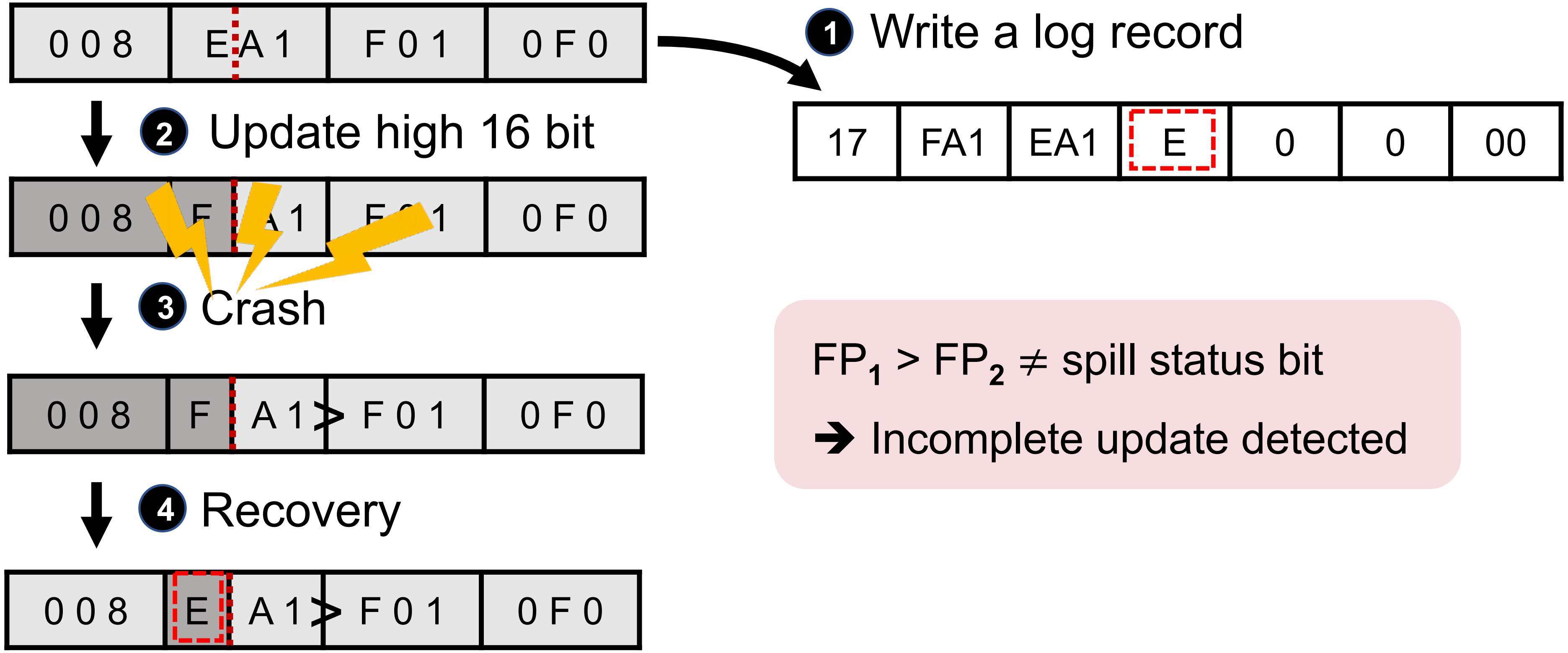
Recovery



$FP_1 > FP_2 \neq$ spill status bit

➔ Incomplete update detected

Recovery



$FP_1 > FP_2 \neq$ spill status bit

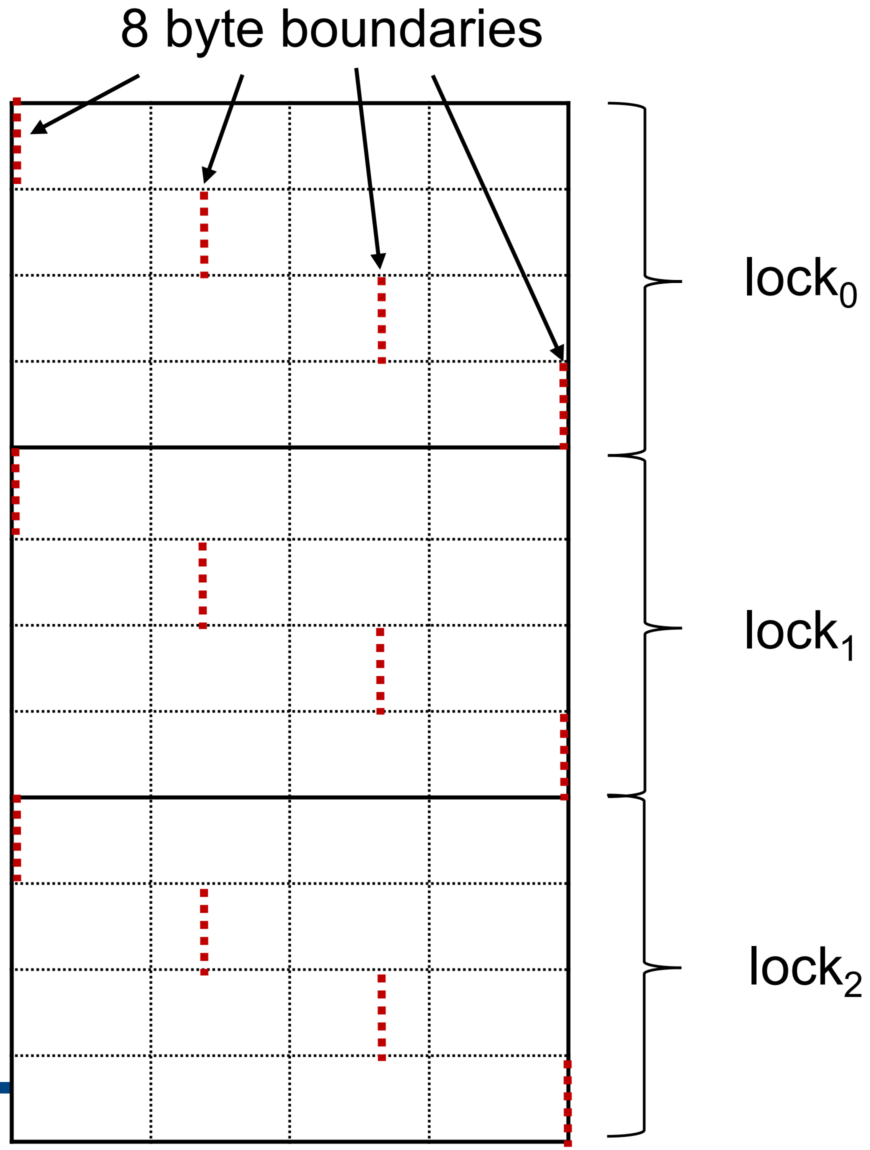
➔ Incomplete update detected

Parallel Implementation

- Synchronization
- Logging

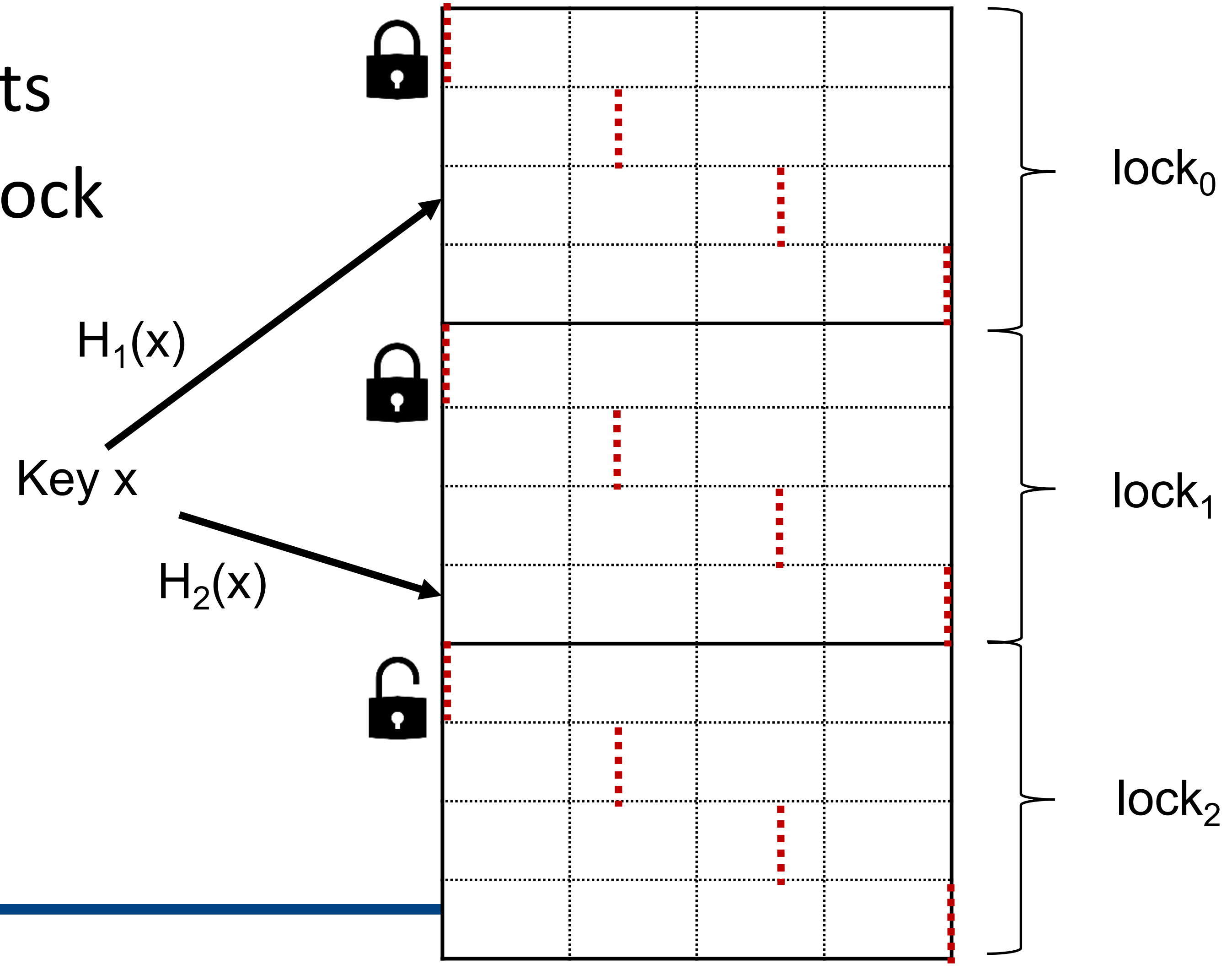
Parallel Implementation – Sync

- Shared lock for buckets
- Holding locks for both buckets
- Lock ordering – for spill, try lock



Parallel Implementation – Sync

- Shared lock for buckets
- Holding locks for both buckets
- Lock ordering – for spill, try lock



Parallel Implementation – Sync

- Shared lock for buckets
- Holding locks for both buckets
- Lock ordering – for spill, try lock

➔ No deadlock, yet may have livelock

1) Single insertion and multiple lookups

– Bounded wait time for lookups

2) Multiple insertions

– Extremely unlikely ($< 10^{-28}$)

Parallel Implementation – Logging

- Thread-local log entries per thread
- Additional log write for commit mark
  - To determine write order between 2 threads accessing a same bucket

Evaluation

- System setting

	Intel Optane DC Persistent Memory*	Quartz Emulation
CPU	Xeon Gold 5215M 2.5GHz (10 cores)	Xeon E5-2620 2.4GHz
Memory	384 GB DRAM + 1512 GB NVM	96 GB DRAM
OS	Linux Kernel 4.18.0	Linux Kernel 4.8.12

*Used AppDirect mode

Evaluation

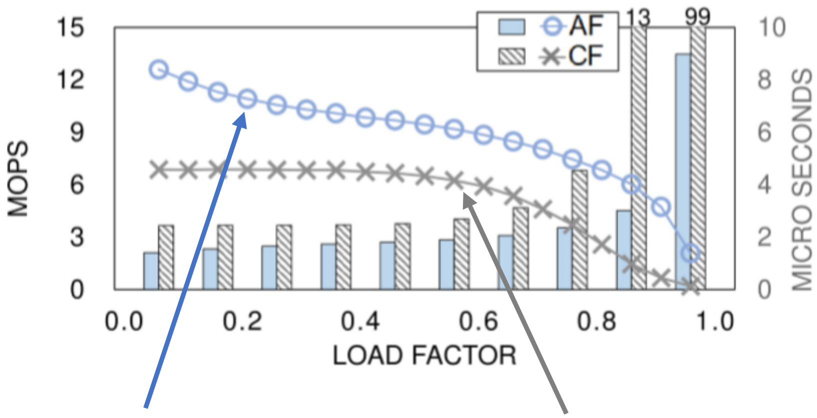
- Evaluated Filters

Filter	Notation	Description
Cuckoo Filter	CF	Bucketized Cuckoo Filter
Morton Filter	MF	DRAM-optimized Cuckoo Filter
Rank-and-Select Quotient Filter	RSQF	SSD-optimized Quotient Filter
Bloom Filter	BF	Bitmap-based AMQ

➔ Configured to have the same false-positive rates

Parallel Throughput*

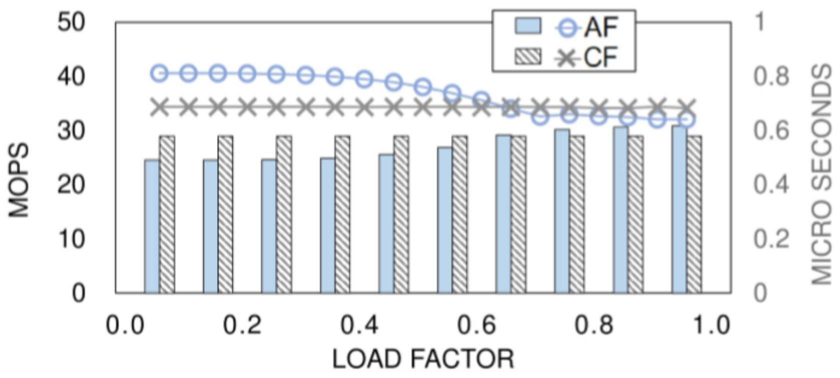
Insertion



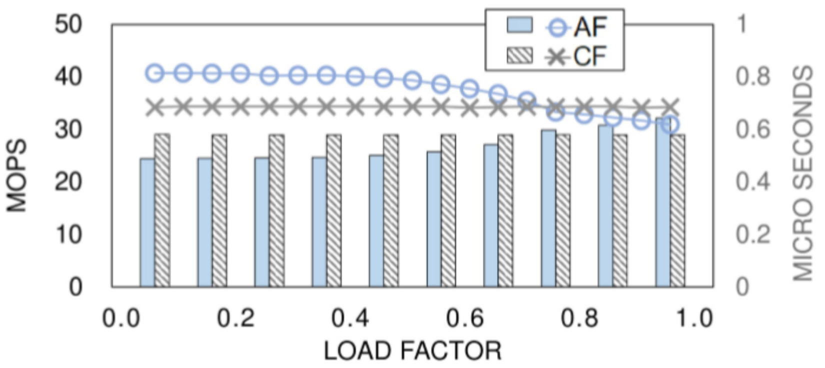
AniFilter

Cuckoo Filter

Successful Lookup



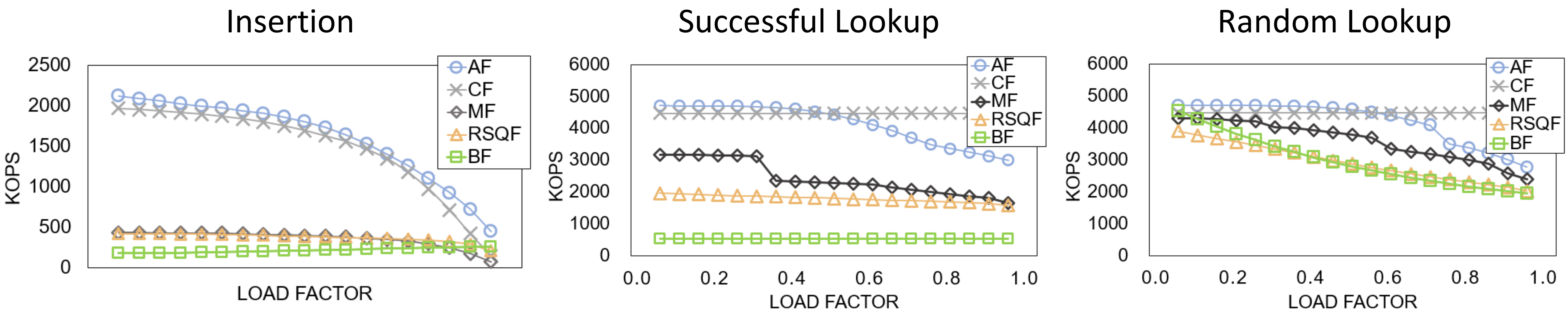
Random Lookup



AniFilter upto 10.7x faster
(2.6x faster on avg) for insertion

*Intel Optane DC PM, 10 threads

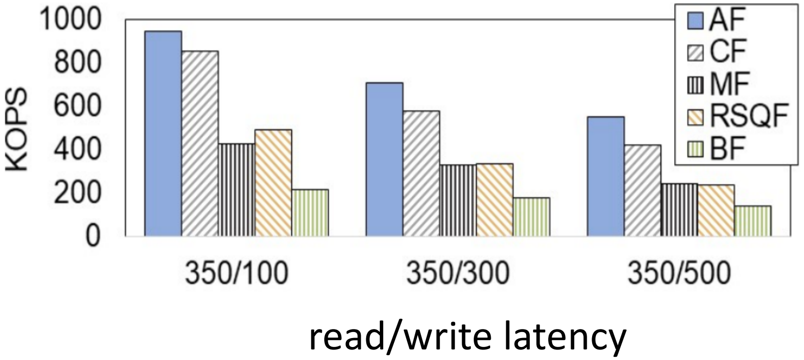
Sequential Throughput*



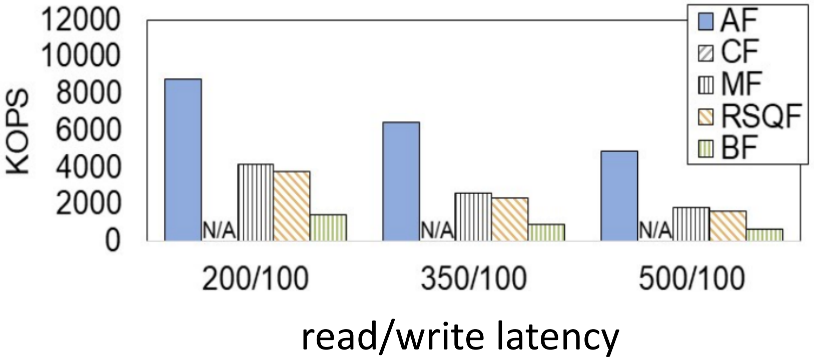
*Intel Optane DC PM

Sequential Throughput*

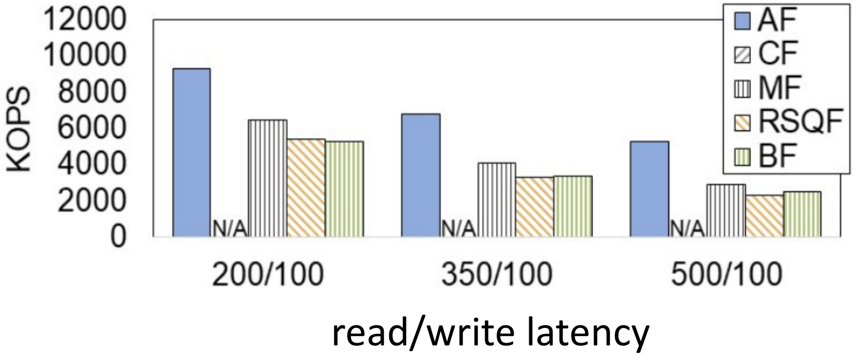
Insertion



Successful Lookup

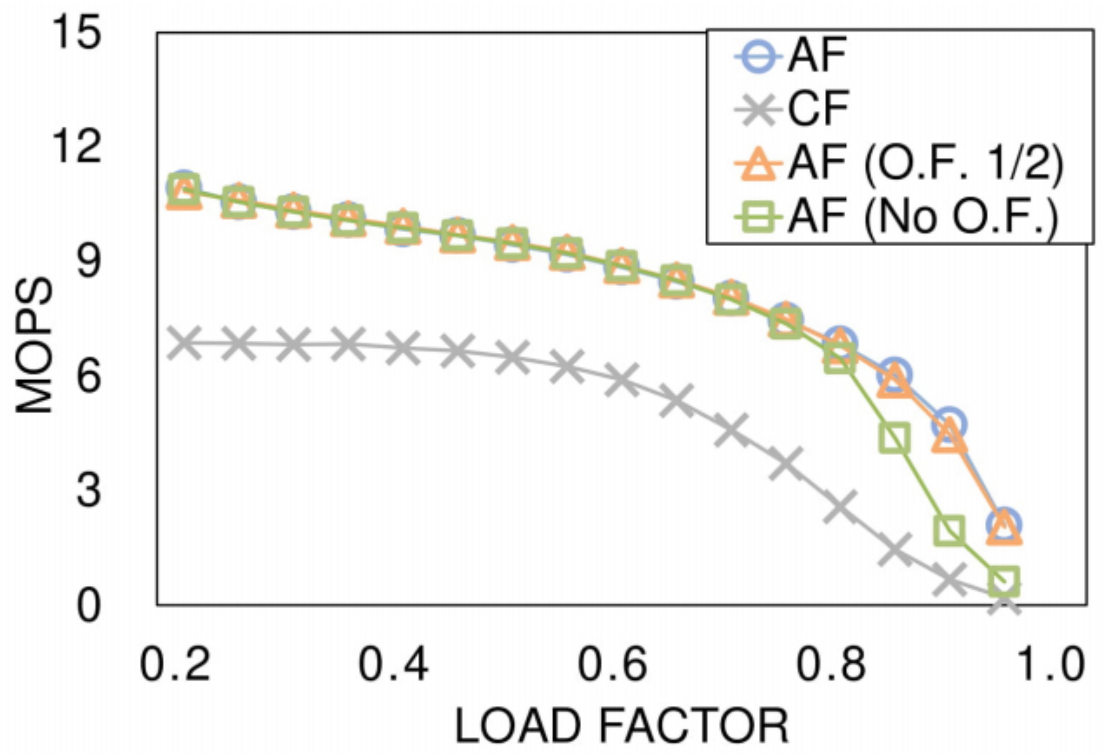


Random Lookup



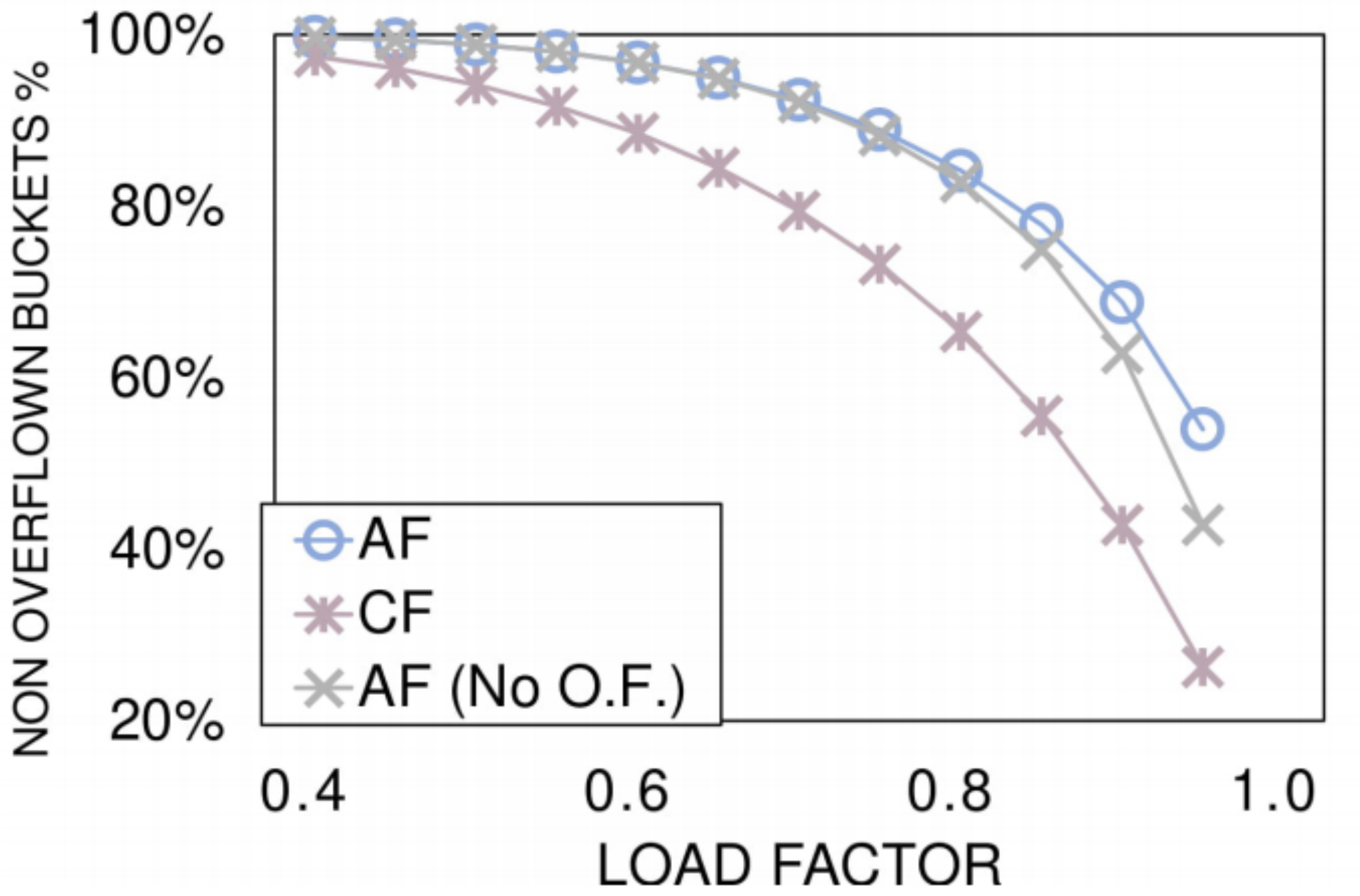
*Quartz emulation, 75% load factor

Effect of Lookahead Eviction – Occupancy Flags



Synergy between Optimizations

- Spillable Buckets and Lookahead Eviction's Impact on Bucket Primacy



Conclusion

- AniFilter – Optimized Cuckoo Filter for NVM
- Optimizations
  - Spillable Buckets
  - Lookahead Evictions
  - Bucket Primacy
- Logging for Failure-Atomicity
- Evaluation on NVM

Q/A

seojiwon@gmail.com