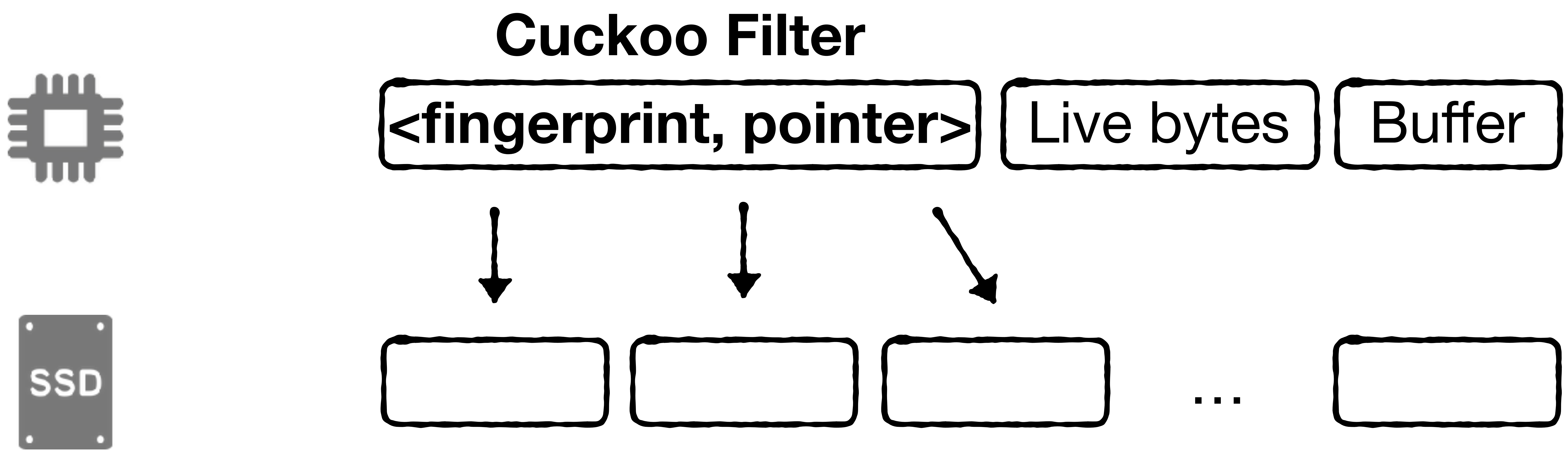
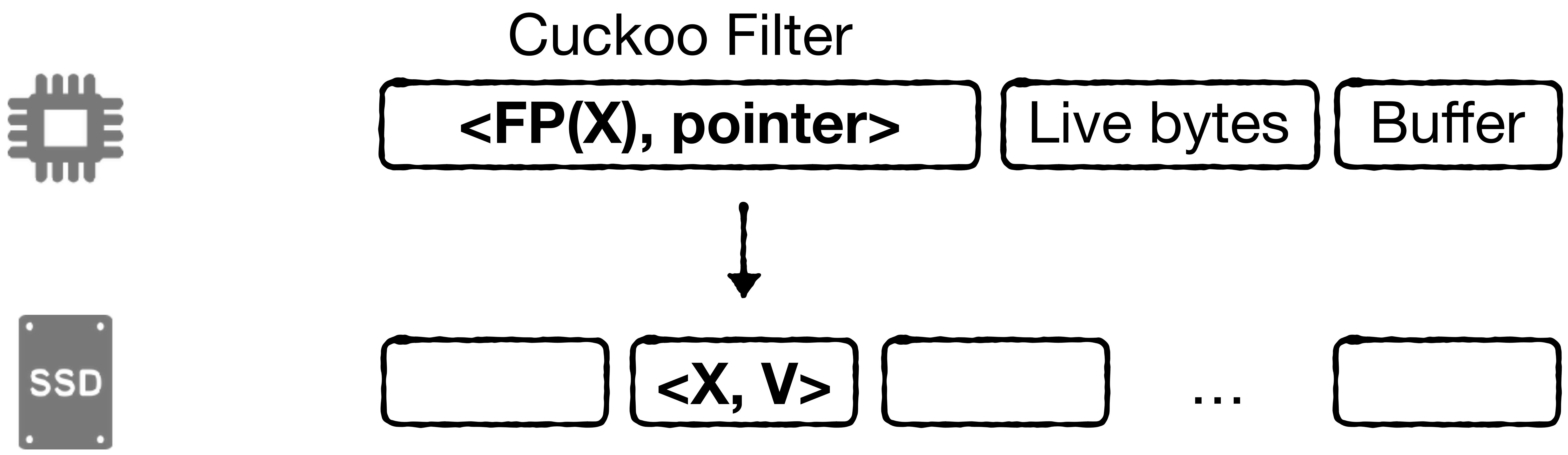
Tutorial on Circular Logs & Cuckoo Filters

Database System Technology

Niv Dayan

	Cuckoo Filter	Bloom filter
false positive rate:	$\approx 2^{-M+3}$	$2^{-M \cdot \ln(2)}$
Memory accesses for positive query	1.5	$M \cdot \ln(2)$
Memory accesses for negative query	2	2
Insertion cost	O(1)	$M \cdot \ln(2)$
Deletes?	**1.5**	**N/A**
Storing Payloads?	**Yes**	**N/A**


Cuckoo Filter
<fingerprint, pointer>
Live bytes
Buffer
SSD
...


Cuckoo Filter
<FP(X), pointer>
Live bytes
Buffer
SSD
<X, V>
...

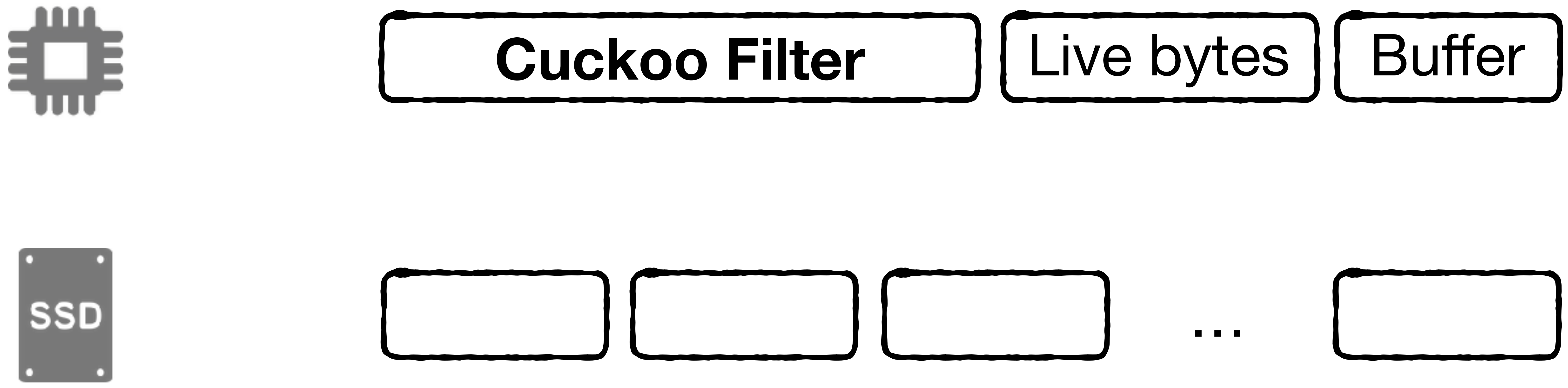
Index size = N * (P + **K**) * α

N = data size

P = pointer size = $O(\log_2 N/B)$

Our goal was reducing this → **K = key size = $\Omega(\log_2 N)$**

α = collision resolution overheads ≈0.8 → ≈0.95



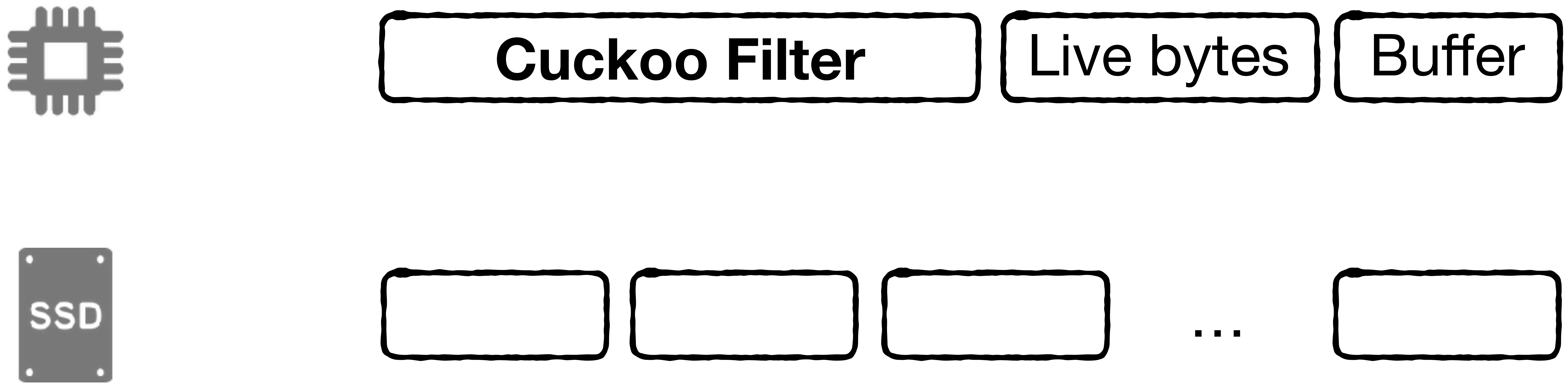
Index size = $N * (P + \mathbf{M}) / \alpha$

N = data size

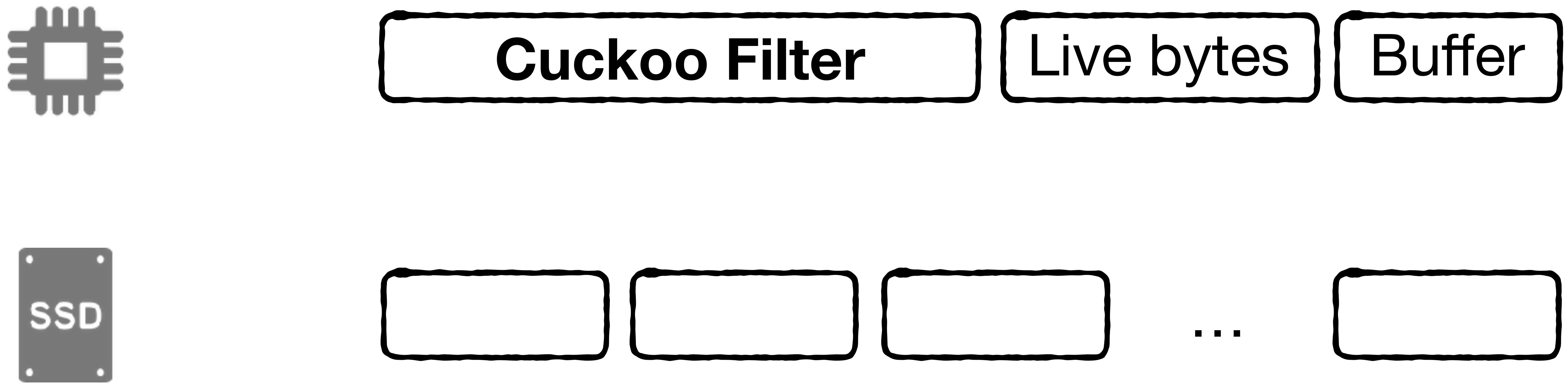
P = pointer size = $O(\log_2 N/B)$

Our goal was reducing this → **K = key size = $\Omega(\log_2 N)$** → **M bits / entry**

α = collision resolution overheads $\approx 0.8 \rightarrow \approx 0.95$

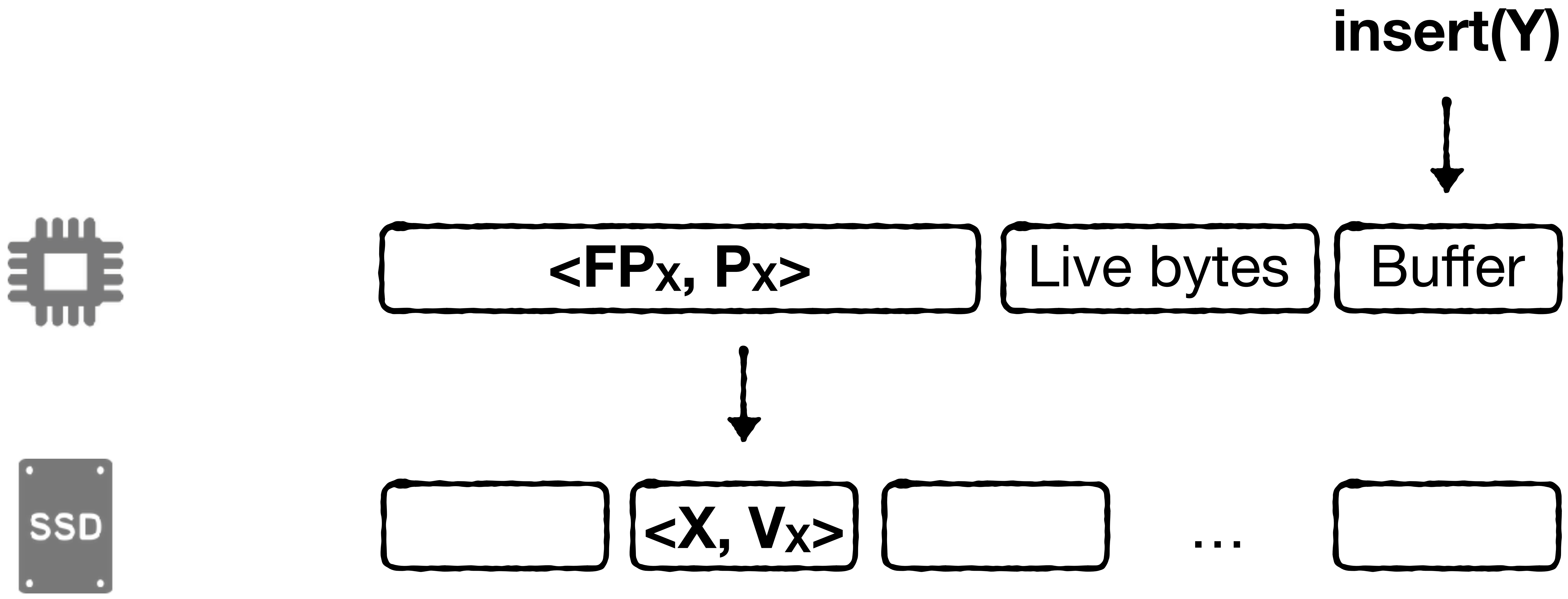


Implication of Using a Filter on Insertions/Deletes/Updates



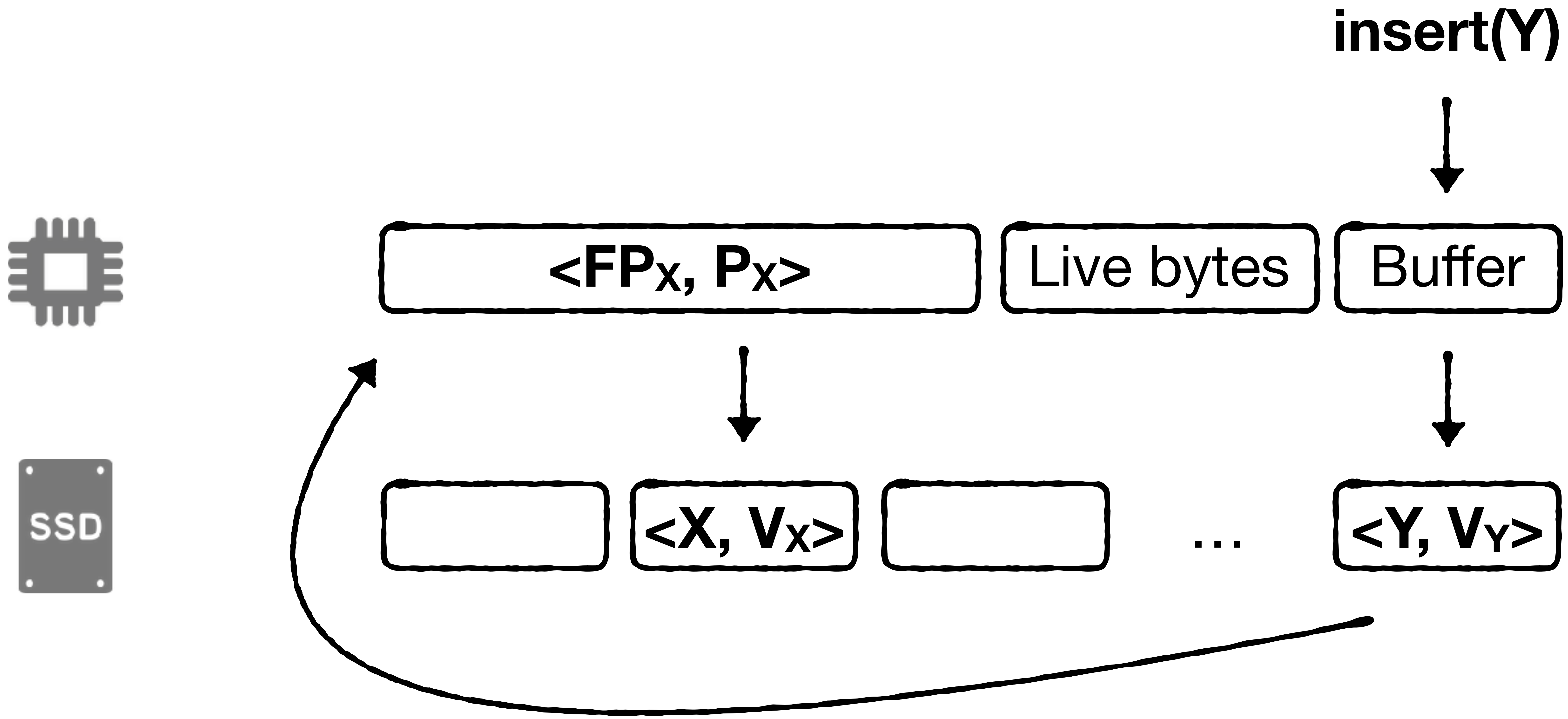
Implication of Using a Filter on Insertions/Deletes/Updates

Suppose we insert key Y that has a matching fingerprint to existing key X ($FP_X = FP_y$)



Implication of Using a Filter on Insertions/Deletes/Updates

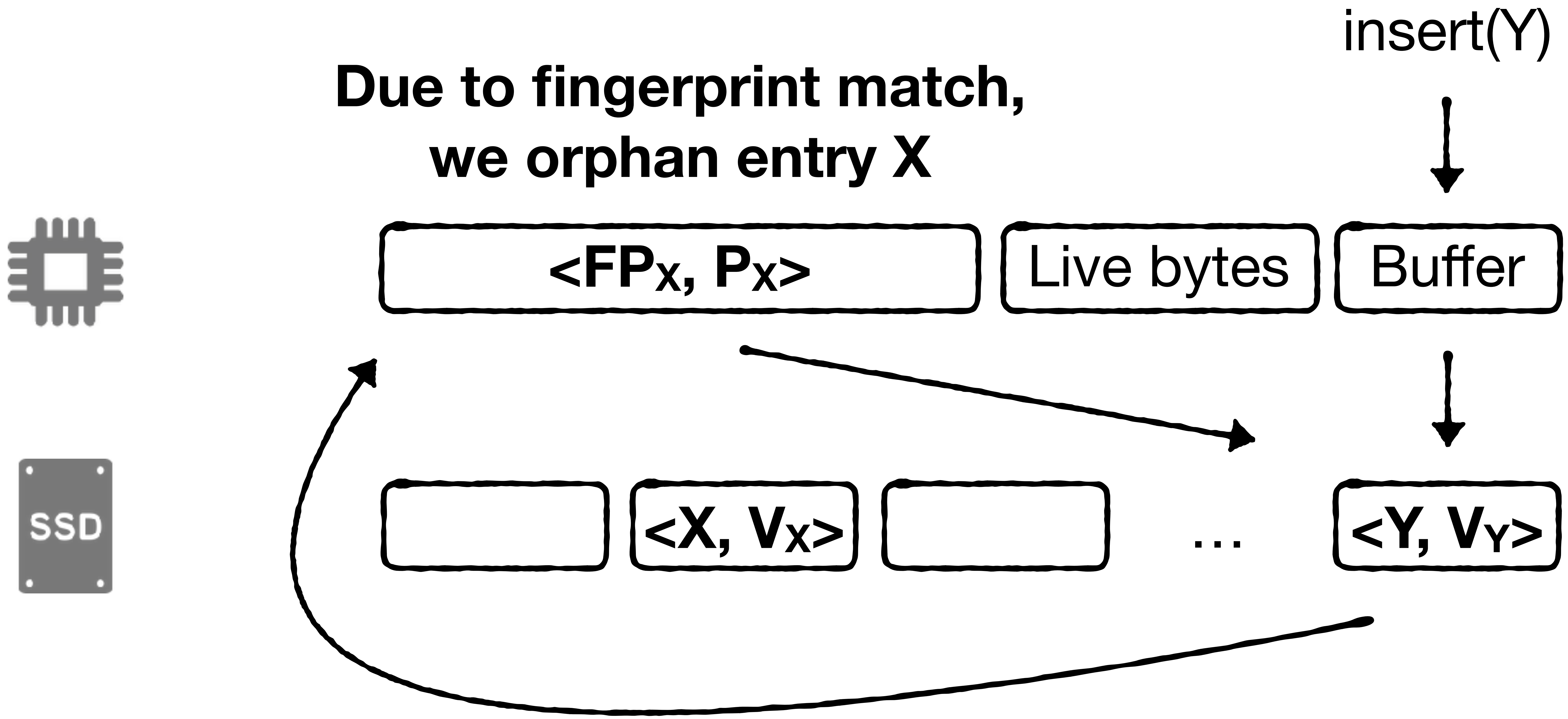
Suppose we insert key Y that has a matching fingerprint to existing key X



Need to set up pointer to Y

Implication of Using a Filter on Insertions/Deletes/Updates

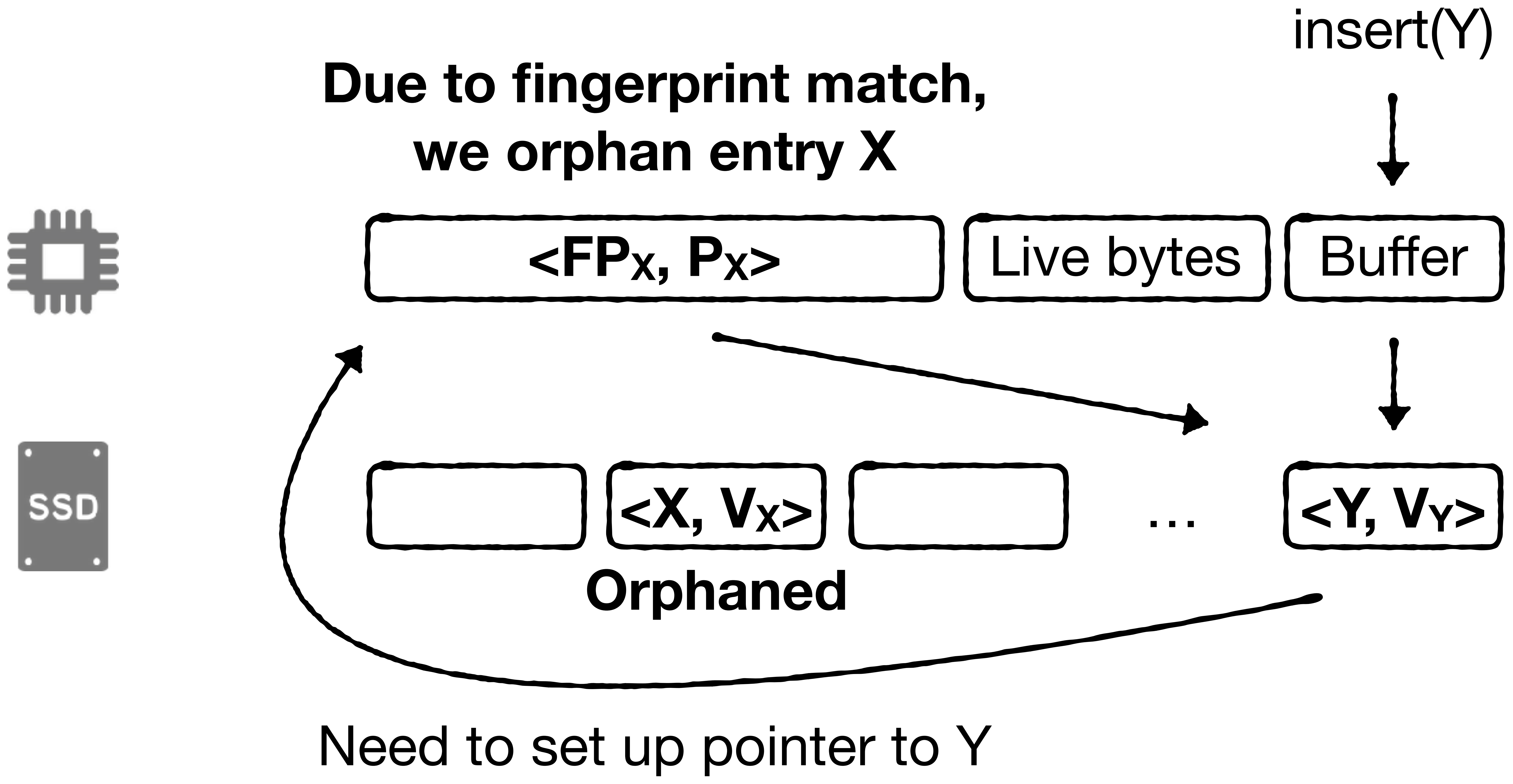
Suppose we insert key Y that has a matching fingerprint to existing key X



Need to set up pointer to Y

Implication of Using a Filter on Insertions/Deletes/Updates

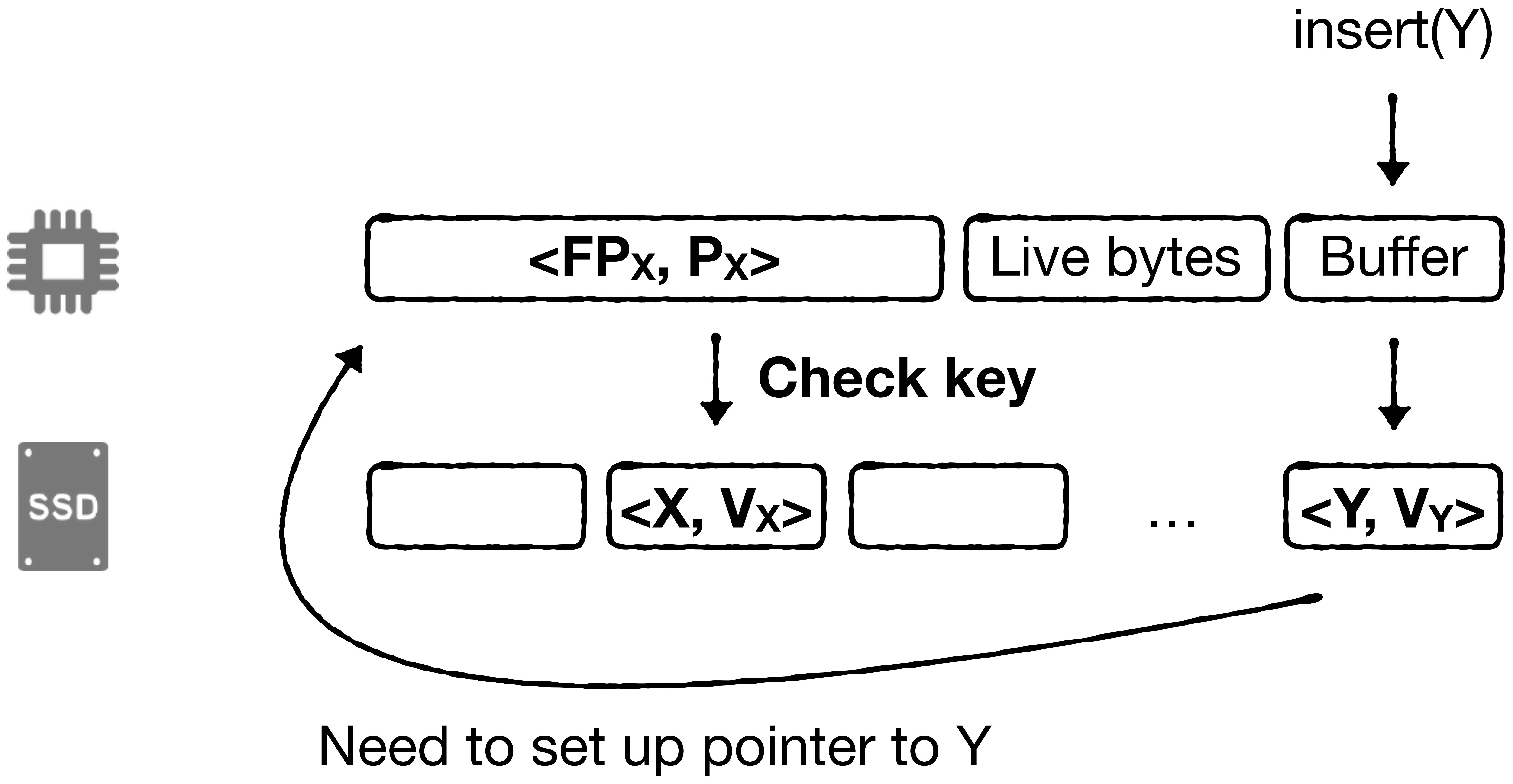
Suppose we insert key Y that has a matching fingerprint to existing key X



Implication of Using a Filter on Insertions/Deletes/Updates

Suppose we insert key Y that has a matching fingerprint to existing key X

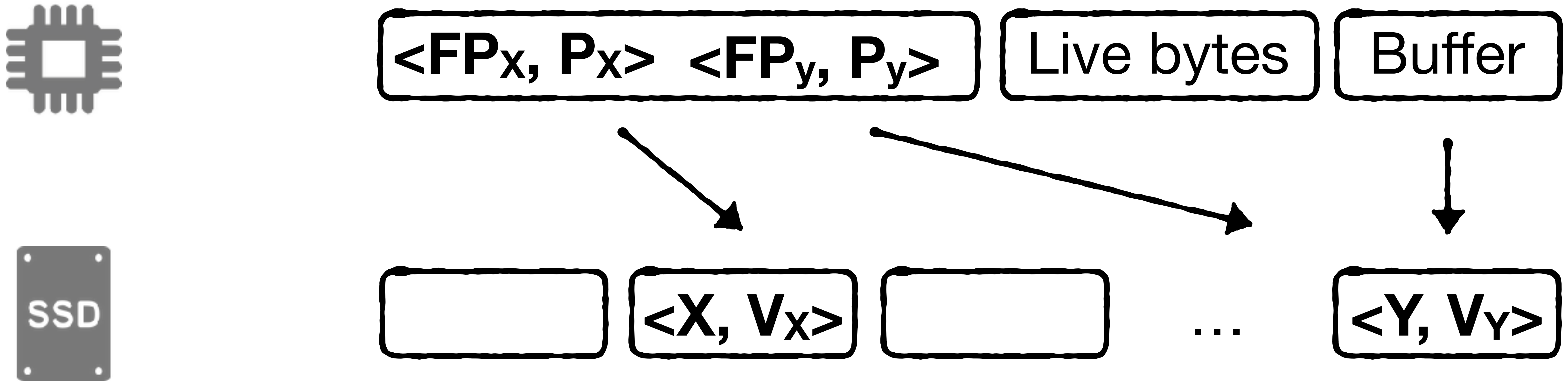
To safeguard against orphaning, we must issue read-before-write



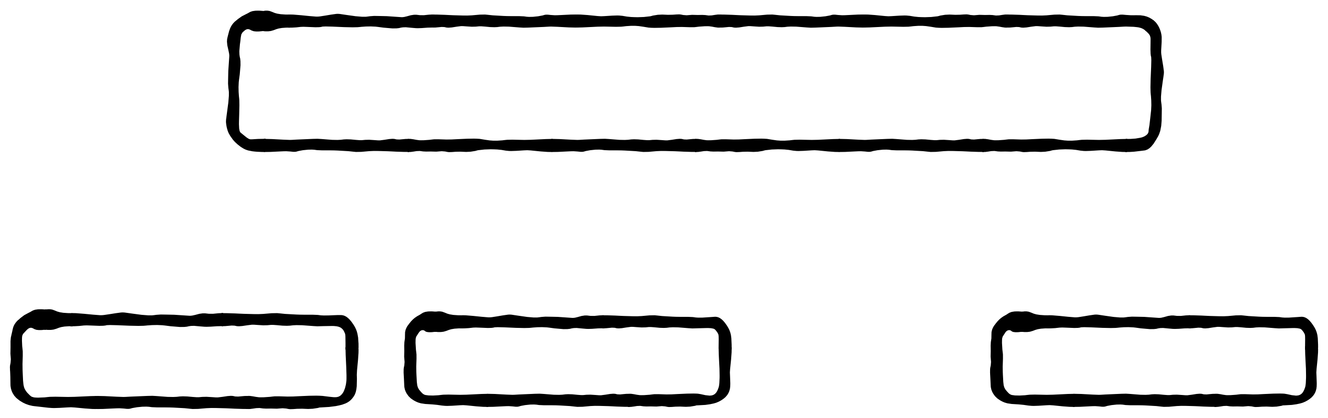
Implication of Using a Filter on Insertions/Deletes/Updates

Suppose we insert key Y that has a matching fingerprint to existing key X

To safeguard against orphaning, we must issue read-before-write



Circular Log w.
Cuckoo Filter



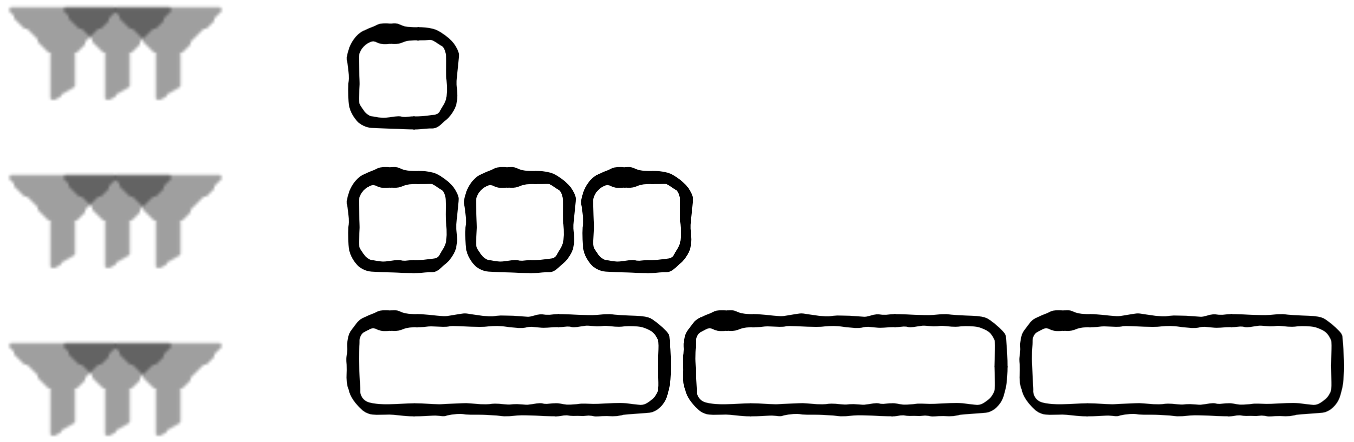
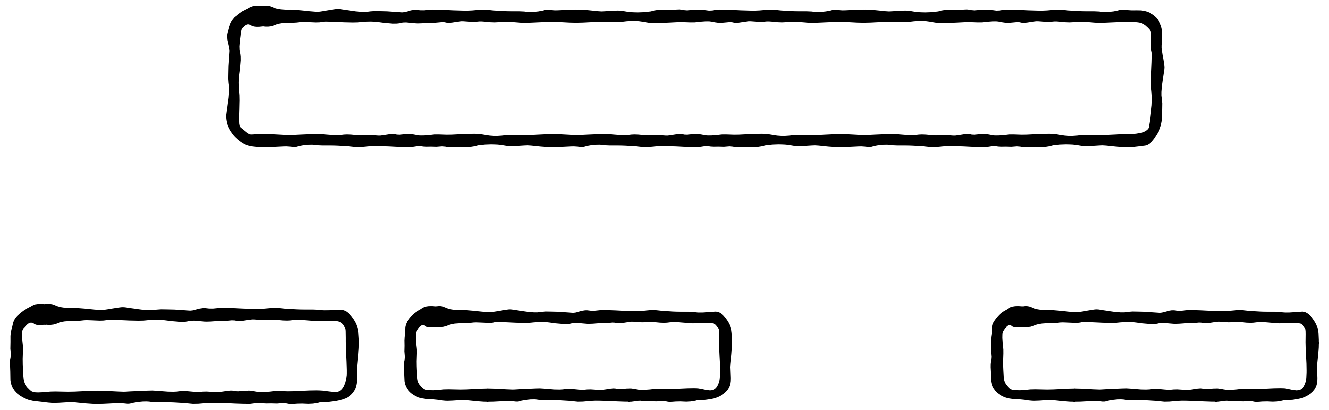
Updates/
Deletes: $O(1+2^{-M+3})$ reads & $O(GC/B)$ writes (excluding checkpointing)

Insert: $O(2^{-M+3})$ read & $O(GC/B)$ writes

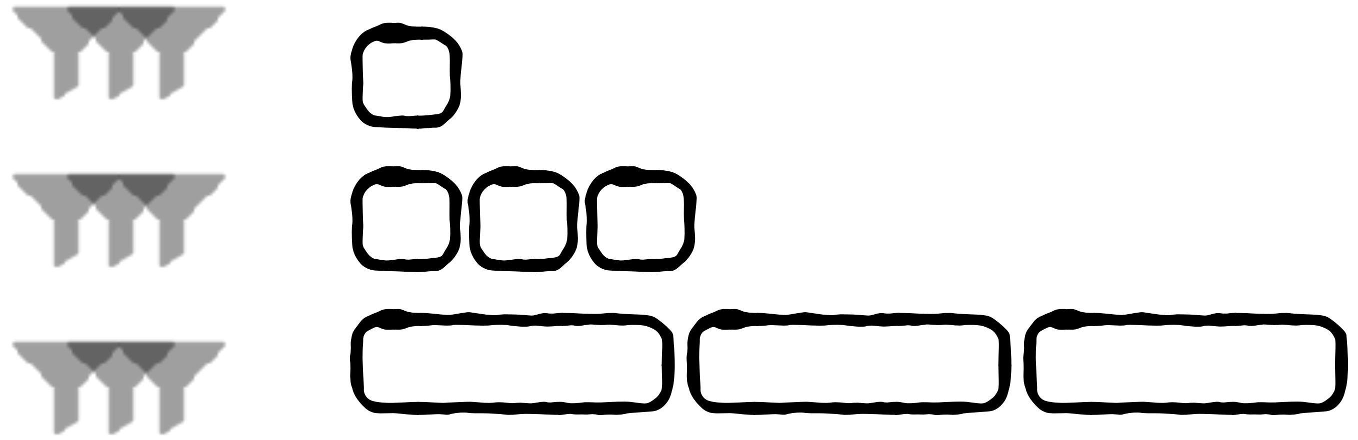
Gets: $O(1+2^{-M+3})$

Scan: $O(N/B)$

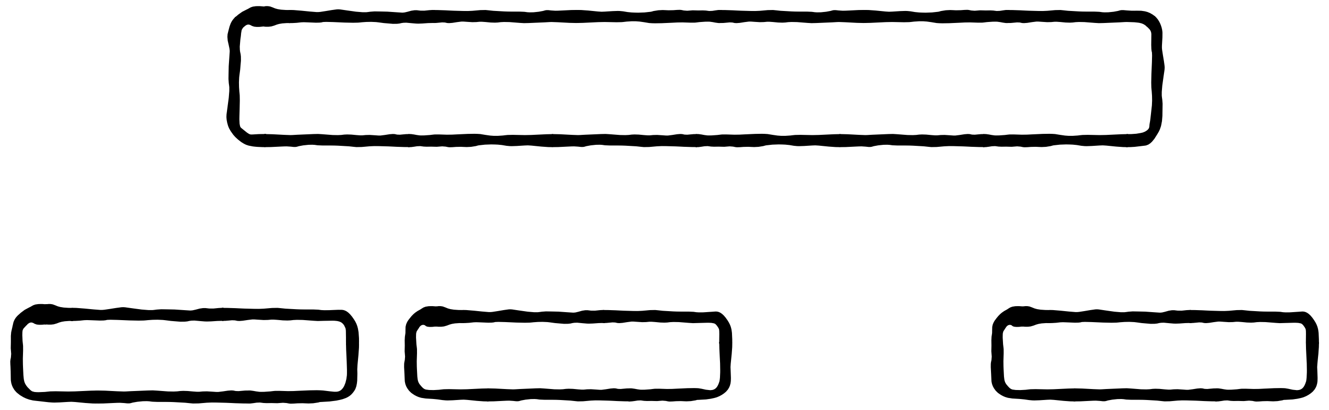
Memory
(bits / entry) $O(\log_2(N/B) + M))$

	Basic LSM-tree w. Monkey	Circular Log w. Cuckoo Filter
		
Updates/ Deletes:	$O(\log_2(N/P) / B)$ reads & writes	$O(1+2^{-M+3})$ reads & $O(GC/B)$ writes
Insert:	$O(\log_2(N/P) / B)$ reads & writes	$O(2^{-M+3})$ read & $O(GC/B)$ writes
Gets:	$O(1+2^{-M * \ln(2)})$	$O(1+2^{-M+3})$
Scan:	$O(\log_2 N/P + S/B)$	$O(N/B)$
Memory (bits / entry)	$O(K/B + M)$	$O(\log_2(N/B) + M))$

Basic LSM-tree w. Monkey



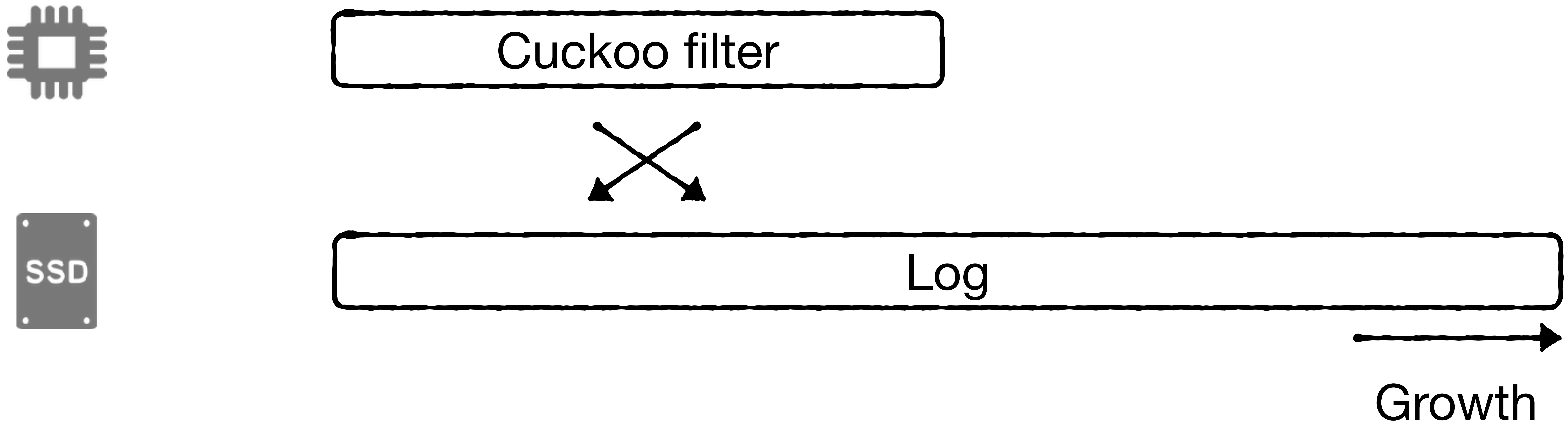
Circular Log w. Cuckoo Filter



	Basic LSM-tree w. Monkey	Circular Log w. Cuckoo Filter
Updates/ Deletes:	$O(\log_2(N/P) / B)$ reads & writes	$O(1+2^{-M+3})$ reads & $O(GC/B)$ writes
Insert:	$O(\log_2(N/P) / B)$ reads & writes	$O(2^{-M+3})$ read & $O(GC/B)$ writes
Gets:	$O(1+2^{-M * \ln(2)})$	$O(1+2^{-M+3})$
Scan:	$O(\log_2 N/P + S/B)$	$O(N/B)$
Memory (bits / entry)	$O(K/B + M)$	$O(\log_2(N/B) + M))$
Recovery	**Swift**	**Longer**

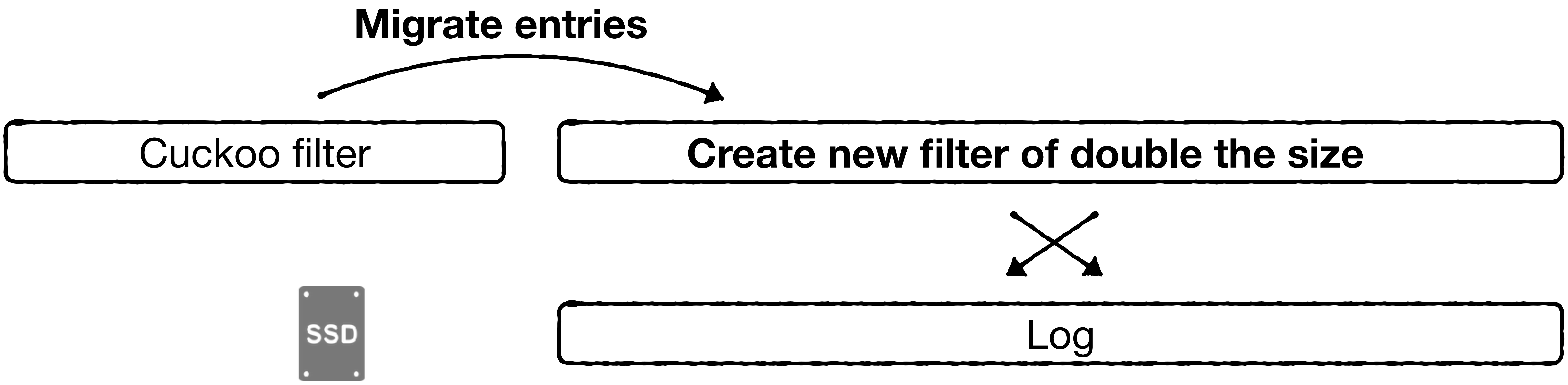
Question 1 - Expanding Cuckoo Filters

A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. Comment on any trade-offs or downsides.



Question 1 - Expanding Cuckoo Filters

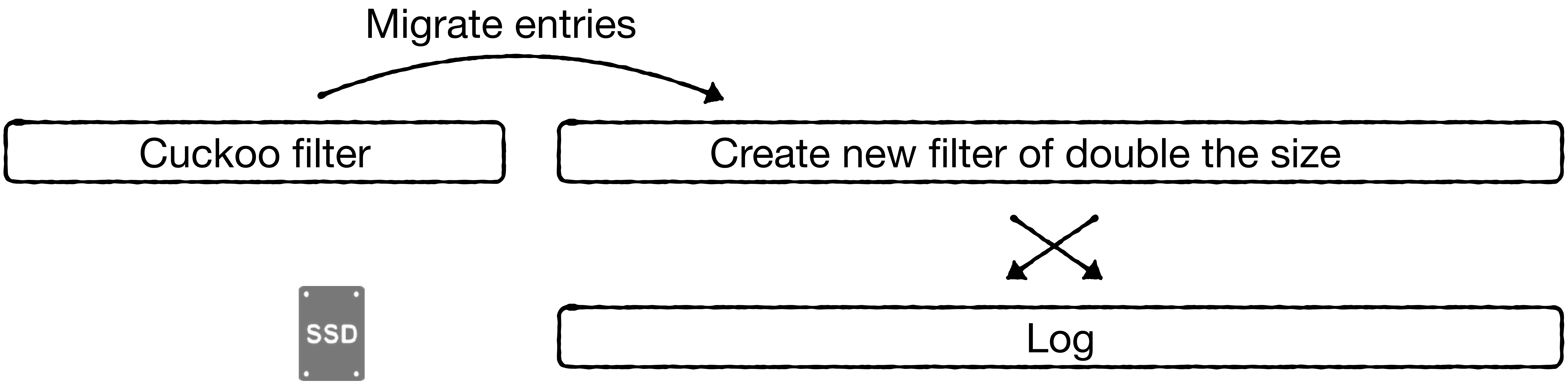
A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. Comment on any trade-offs or downsides.



Question 1 - Expanding Cuckoo Filters

A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. Comment on any trade-offs or downsides.

Challenge: we do not have the full keys to rehash. We only have fingerprints.



Question 1 - Expanding Cuckoo Filters

A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. Comment on any trade-offs or downsides.

Challenge: we do not have the full keys to rehash. We only have fingerprints.

Approach: view fingerprint and first bucket address as components of the same hash

address fingerprint

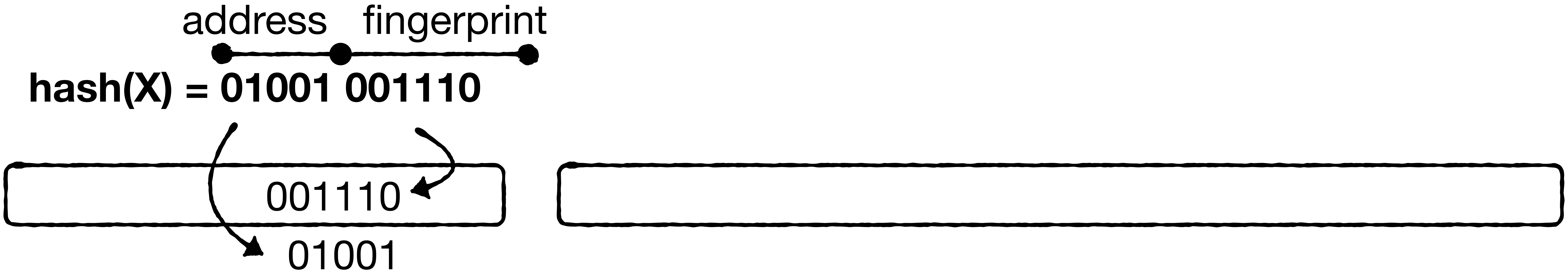
hash(X) = 01001 001110

Question 1 - Expanding Cuckoo Filters

A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. Comment on any trade-offs or downsides.

Challenge: we do not have the full keys to rehash. We only have fingerprints.

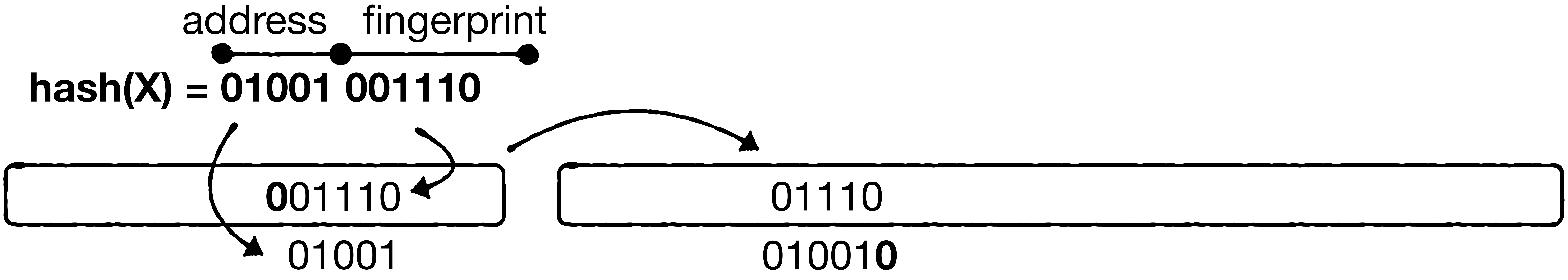
Approach: view fingerprint and first bucket address as components of the same hash



Question 1 - Expanding Cuckoo Filters

A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. Comment on any trade-offs or downsides.

To migrate, transfer one bit from fingerprint to address

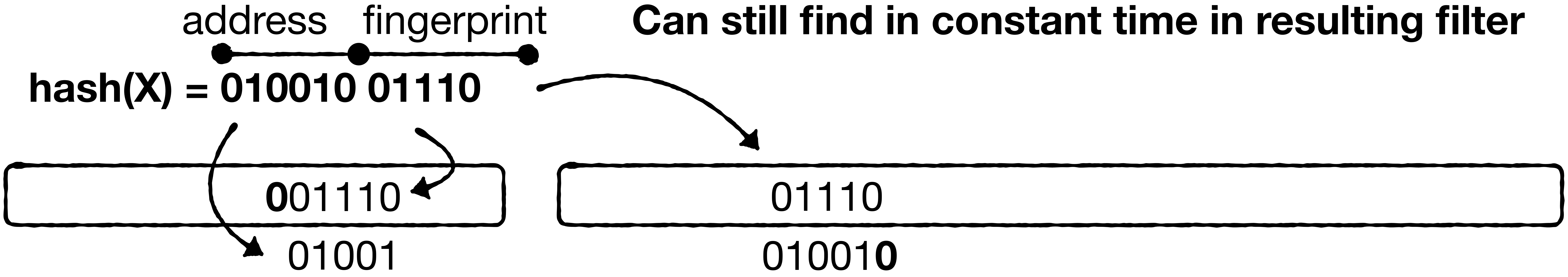


Question 1 - Expanding Cuckoo Filters

A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. Comment on any trade-offs or downsides.

To migrate, transfer one bit from fingerprint to address

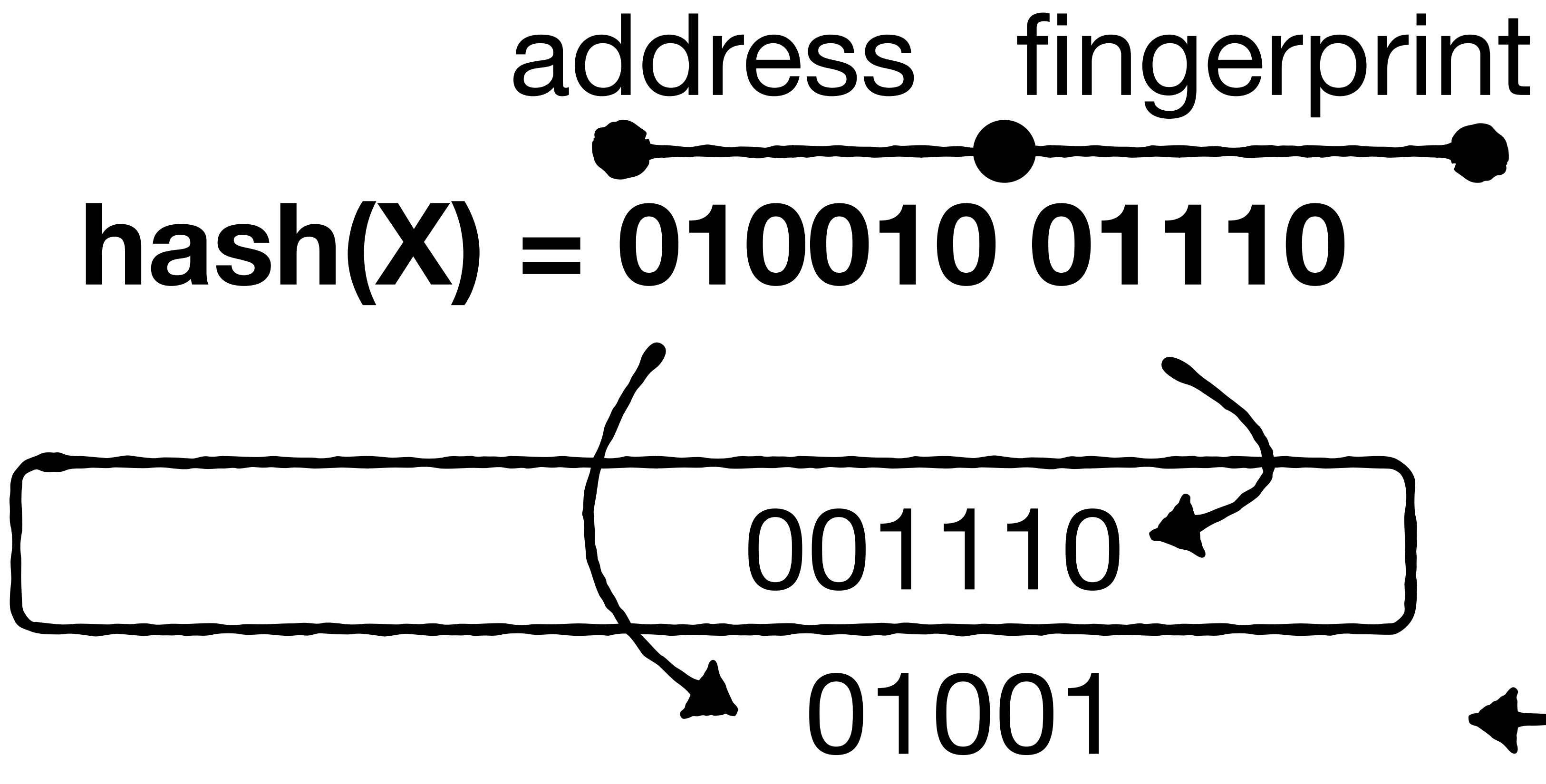
Can still find in constant time in resulting filter



Question 1 - Expanding Cuckoo Filters

A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. Comment on any trade-offs or downsides.

Complication: in a cuckoo filter an entry can be in one of two buckets, the canonical address and the alternative address. Only the canonical address should be viewed as a part of the original hash.



Alternative address = Canonical address XOR h(fingerprint)

Canonical or alternative address?

Question 1 - Expanding Cuckoo Filters

A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. Comment on any trade-offs or downsides.

Complication: in a cuckoo filter an entry can be in one of two buckets, the canonical address and the alternative address. Only the canonical address should be viewed as a part of the original hash.

Solution: add a bit to indicate whether the current address is canonical or alternative. If alternative, switch to canonical via XOR.

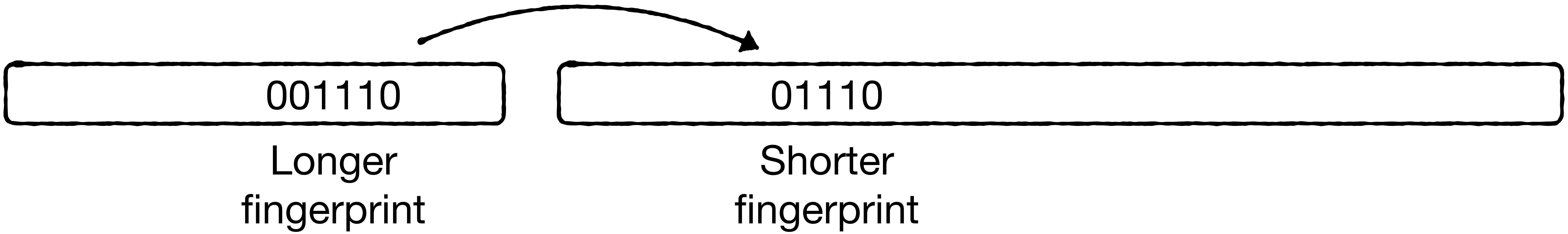
0 001110

01001

Question 1 - Expanding Cuckoo Filters

A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. **Comment on any trade-offs or downsides.**

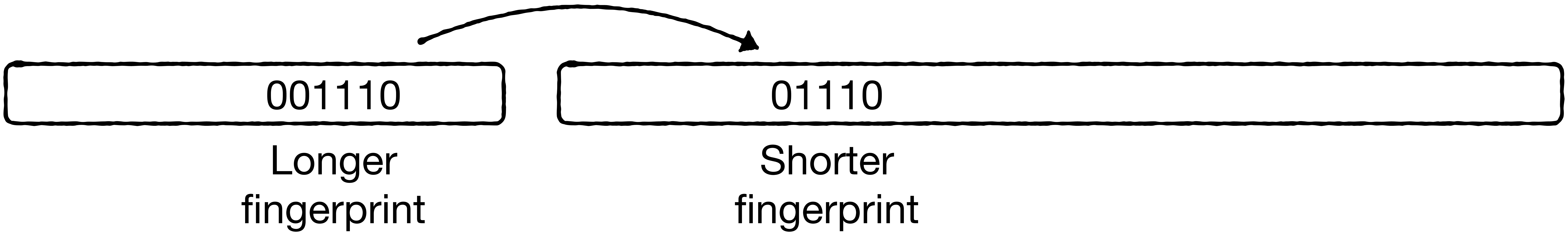
Every time we double the data size, we lose one bit from all fingerprints. Hence, the false positive rate increases: $O(2^{-M + 3 + \log(N)})$



Question 1 - Expanding Cuckoo Filters

A Cuckoo filter is allocated with a fixed capacity. As a circular log grows, we must expand its cuckoo filter to map more data. Devise a Cuckoo filter expansion algorithm. Ideally, this algorithm should maintain constant time performance and not have to read any data from storage. **Comment on any trade-offs or downsides.**

Every time we double the data size, we lose one bit from all fingerprints. Hence, the false positive rate increases: $O(N \cdot 2^{-M+3})$

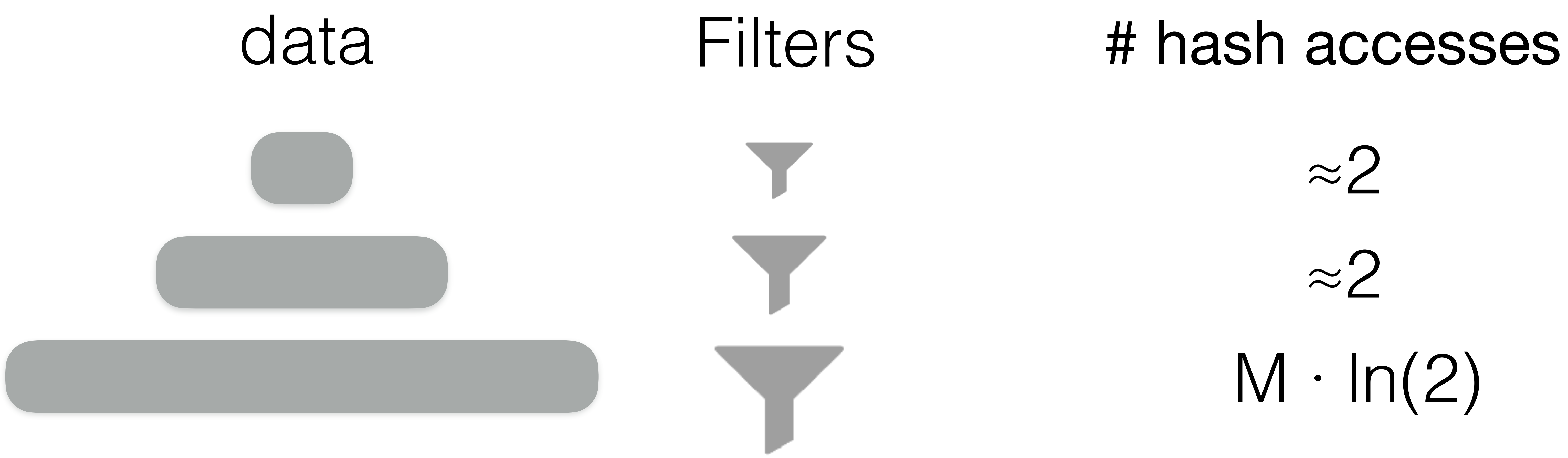


Question 2 - Cuckoo Filter for LSM-tree

As we have seen, the expected worst case query cost over Bloom filters for a basic LSM-tree is $O(L + M)$, where L is the number of levels and M is the number of bits per entry. (Assume only unique entries in the tree).

(A) How can we employ a cuckoo filter to achieve constant time?

(B) What are the implications on the false positive rate and memory footprint? Any downsides compared to plain Bloom filters?



Question 2 - Cuckoo Filter for LSM-tree

As we have seen, the expected worst case query cost over Bloom filters for a basic LSM-tree is O(L + M), where L is the number of levels and M is the number of bits per entry. (Assume only unique entries in the tree).

(A) How can we employ a cuckoo filter to achieve constant time?

(B) What are the implications on the false positive rate and memory footprint? Any downsides compared to plain Bloom filters?

Question 2 - Cuckoo Filter for LSM-tree

As we have seen, the expected worst case query cost over Bloom filters for a basic LSM-tree is O(L + M), where L is the number of levels and M is the number of bits per entry. (Assume only unique entries in the tree).

(A) How can we employ a cuckoo filter to achieve constant time?
(B) What are the implications on the false positive rate and memory footprint? Any downsides compared to plain Bloom filters?

Unified Cuckoo filter

Filter accesses:	$O(1)$
False positive rate:	$O(2^{-M+3})$
Memory (bits/entry)	$O(M + \log_2(L))$
Construction:	$O(L)$

FP(X), 2
FP(Y), 3
…

<M bit Fingerprint, level>

Question 2 - Cuckoo Filter for LSM-tree

As we have seen, the expected worst case query cost over Bloom filters for a basic LSM-tree is $O(L + M)$, where L is the number of levels and M is the number of bits per entry. (Assume only unique entries in the tree).

(A) How can we employ a cuckoo filter to achieve constant time?

(B) What are the implications on the false positive rate and memory footprint? Any downsides compared to plain Bloom filters?

	Unified Cuckoo filter	**With Bloom filters**
Filter accesses:	$O(1)$	$O(M+L)$
False positive rate:	$O(2^{-M+3})$	$O(L \cdot 2^{-M \cdot \ln(2)})$
Memory (bits/entry)	$O(M + \log_2(L))$	$O(M)$
Construction:	$O(L)$	$O(L \cdot M)$

Question 2 - Cuckoo Filter for LSM-tree

As we have seen, the expected worst case query cost over Bloom filters for a basic LSM-tree is $O(L + M)$, where L is the number of levels and M is the number of bits per entry. (Assume only unique entries in the tree).

(A) How can we employ a cuckoo filter to achieve constant time?
(B) What are the implications on the false positive rate and memory footprint? Any downsides compared to plain Bloom filters?

	Unified Cuckoo filter	**With Bloom filters**	**Monkey**
Filter accesses:	$O(1)$	$O(M+L)$	$O(M+L)$
False positive rate:	$O(2^{-M+3})$	$O(L \cdot 2^{-M \cdot \ln(2)})$	$O(2^{-M \cdot \ln(2)})$
Memory (bits/entry)	$O(M + \log_2(L))$	$O(M)$	$O(M)$
Construction:	$O(L)$	$O(L \cdot M)$	$O(L \cdot (L + M))$

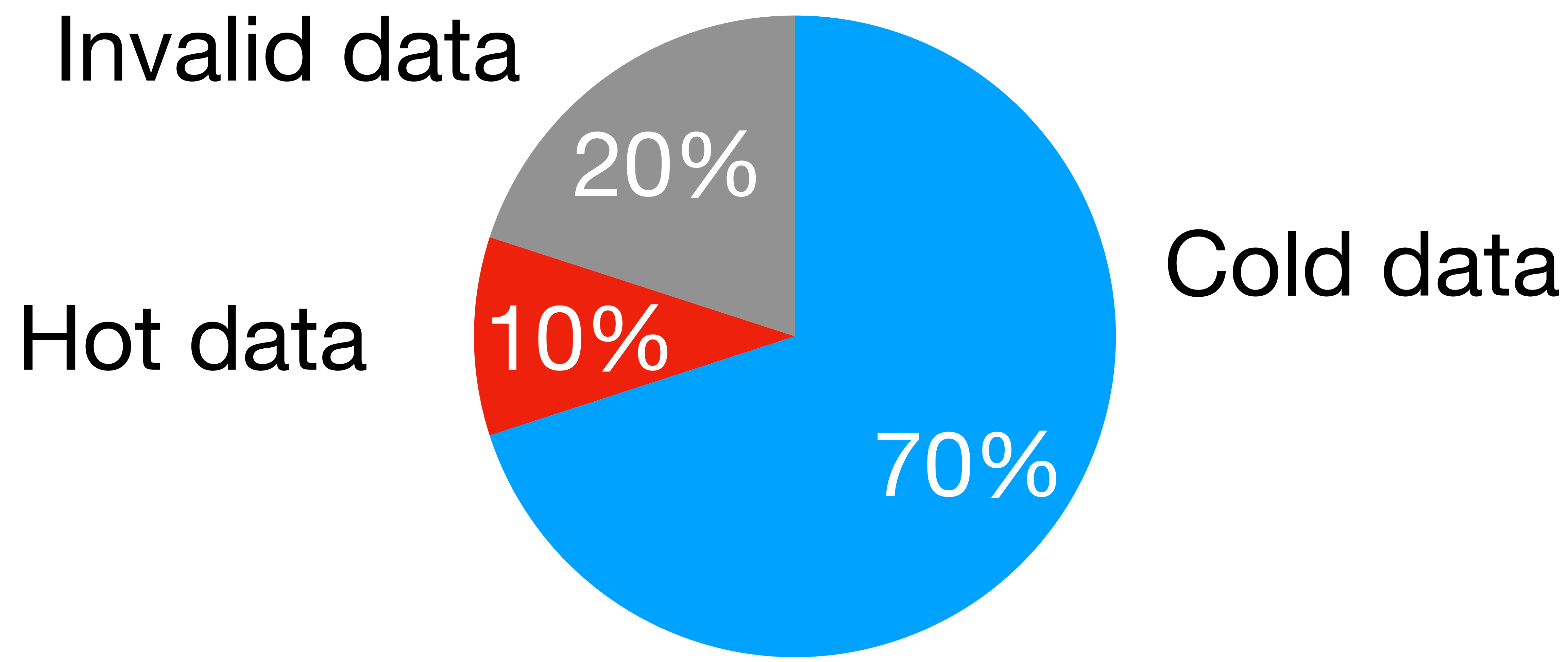
(This memory analysis here only account for the filters and not the fence pointers (internal nodes) being stored in memory)

Question 3 - Hot/Cold Data Separation

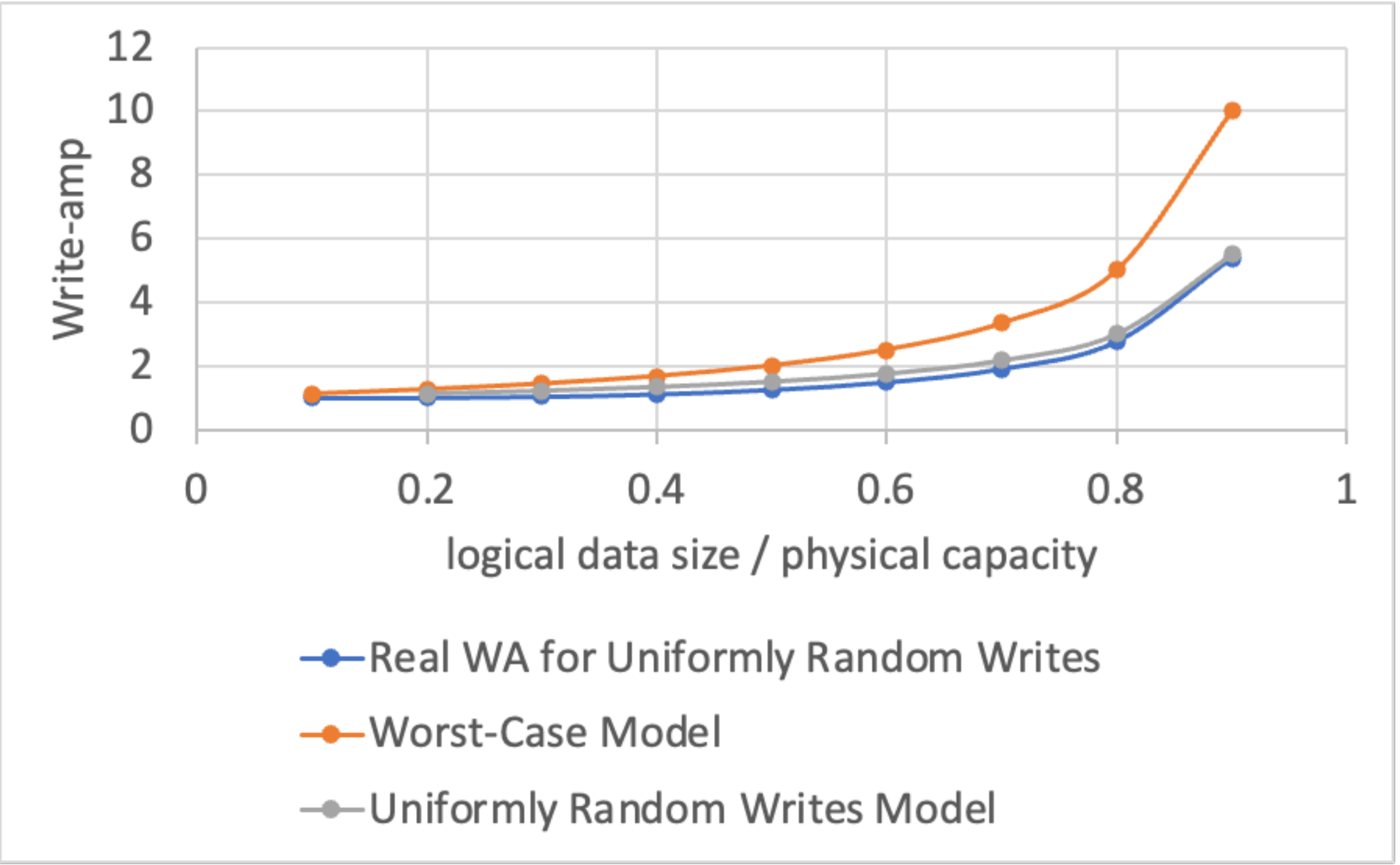
Consider a circular log where the physical capacity consists of 70% static data (never updated), 10% hot data, and 20% over-provisioning.

(A) Estimate a lower bound and upper bound for write-amplification assuming no hot/cold data separation.

(B) Estimate write-amplification assuming perfect hot/cold data separation.



Garbage-Collection Write-Amplification



$$1 + \frac{L/P}{1 - L/P}$$ **Worst case**

$$1 + \frac{1}{2} \cdot \frac{L/P}{1 - L/P}$$ **Uniformly random**

L = logical data size

P = physical data size

Garbage-Collection Write-Amplification

$$1 + \frac{L/P}{1 - L/P} = 1 + \frac{L}{P - L}$$ **Worst case**

$$1 + \frac{1}{2} \cdot \frac{L/P}{1 - L/P} = 1 + \frac{1}{2} \cdot \frac{L}{P - L}$$ **Uniformly random**

L = logical data size

P = physical data size

Garbage-Collection Write-Amplification

$$1 + \frac{L/P}{1 - L/P} = 1 + \frac{L}{P - L} = 1 + \frac{L}{O}$$ **Worst case**

$$1 + \frac{1}{2} \cdot \frac{L/P}{1 - L/P} = 1 + \frac{1}{2} \cdot \frac{L}{P - L} = 1 + \frac{1}{2} \cdot \frac{L}{O}$$ **Uniformly random**

L = logical data size

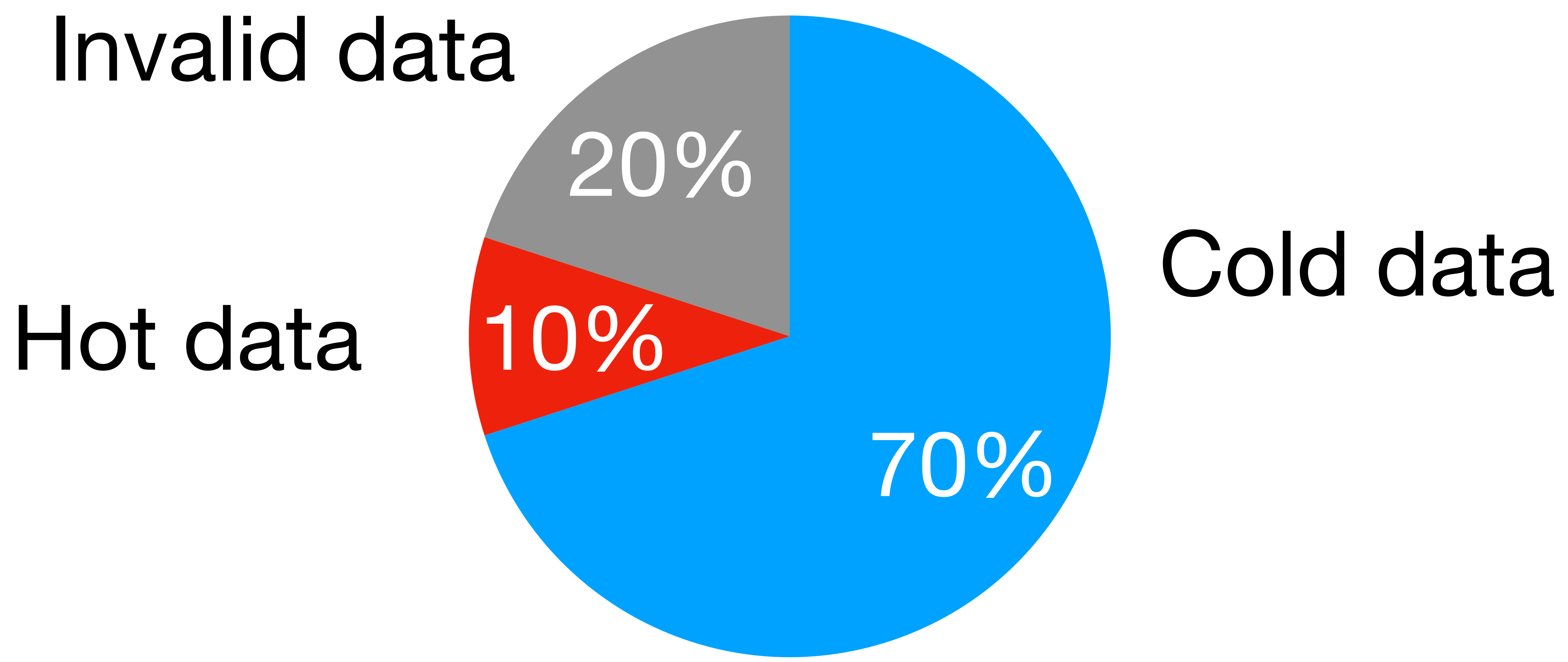
P = physical data size

O = Overprovisioned space (P-L)

Question 3 - Hot/Cold Data Separation

Consider a circular log where the physical capacity consists of 70% static data (never updated), 10% hot data, and 20% over-provisioning.

(A) Estimate a lower bound and upper bound for write-amplification assuming no hot/cold data separation.



Worst case WA: $= 1 + \frac{L}{O}$

Uniformly random: $= 1 + \frac{1}{2} \cdot \frac{L}{O}$

Question 3 - Hot/Cold Data Separation

Consider a circular log where the physical capacity consists of 70% static data (never updated), 10% hot data, and 20% over-provisioning.

(A) Estimate a lower bound and upper bound for write-amplification assuming no hot/cold data separation.

Let C=0.7, H=0.1 and O=0.2

Worst case WA: $= 1 + \frac{L}{O}$

Uniformly random: $= 1 + \frac{1}{2} \cdot \frac{L}{O}$

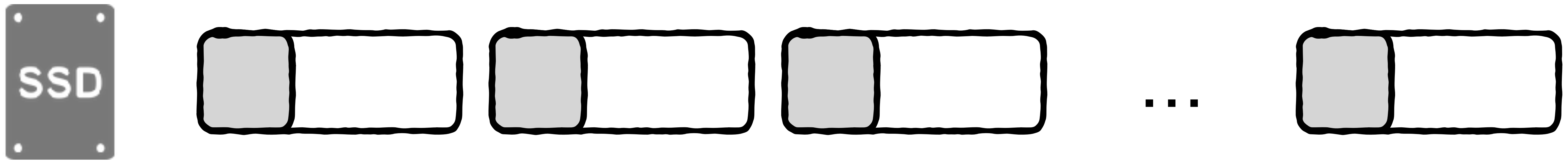
Question 3 - Hot/Cold Data Separation

Consider a circular log where the physical capacity consists of 70% static data (never updated), 10% hot data, and 20% over-provisioning.

(A) Estimate a lower bound and upper bound for write-amplification assuming no hot/cold data separation.

Let C=0.7, H=0.1 and O=0.2

In worst-case, same amount of live data in each area



Worst case WA: $= 1 + \frac{L}{O}$

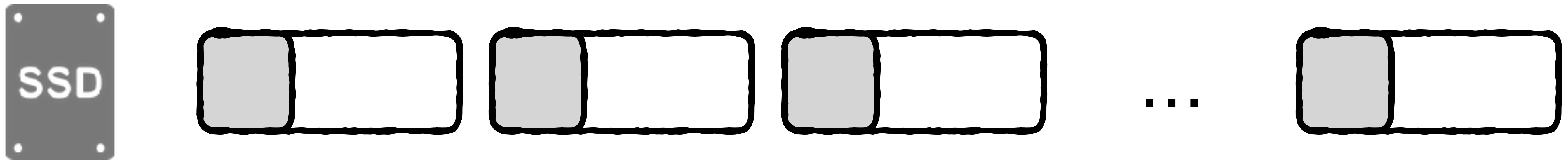
Question 3 - Hot/Cold Data Separation

Consider a circular log where the physical capacity consists of 70% static data (never updated), 10% hot data, and 20% over-provisioning.

(A) Estimate a lower bound and upper bound for write-amplification assuming no hot/cold data separation.

Let C=0.7, H=0.1 and O=0.2

In worst-case, same amount of live data in each area



Upper bound: $= 1 + \frac{L}{O} = 1 + \frac{H + C}{O} = 1 + \frac{0.8}{0.2} = 5$

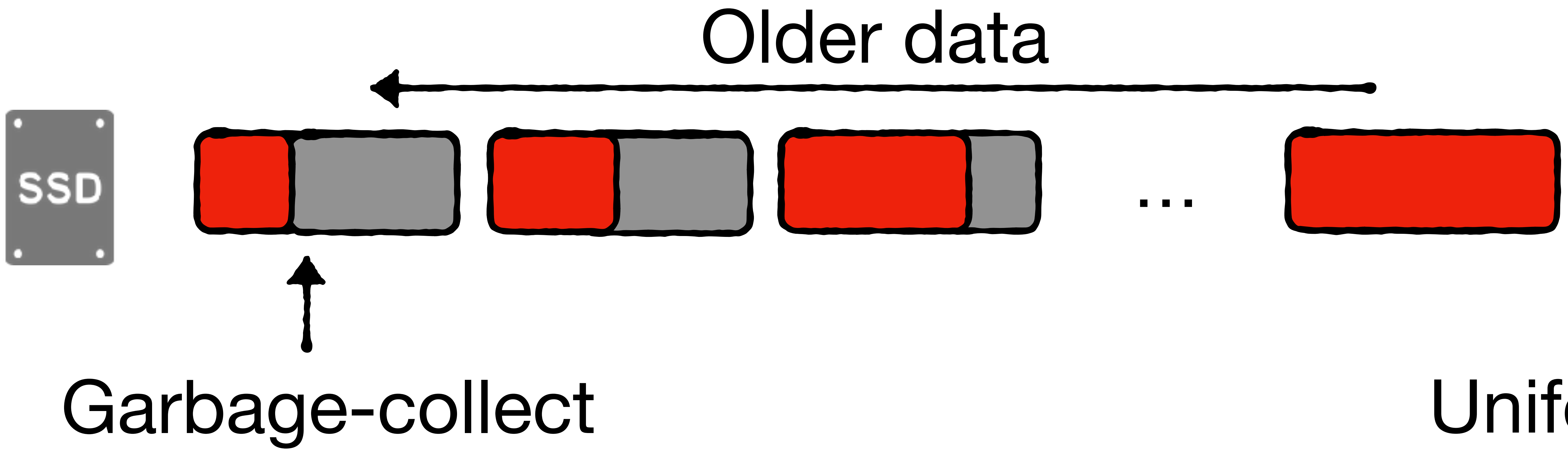
Question 3 - Hot/Cold Data Separation

Consider a circular log where the physical capacity consists of 70% static data (never updated), 10% hot data, and 20% over-provisioning.

(A) Estimate a lower bound and upper bound for write-amplification assuming no hot/cold data separation.

Let C=0.7, H=0.1 and O=0.2

For lower bound, let's use our uniform workload distribution estimation.



Uniformly random: $= 1 + \frac{1}{2} \cdot \frac{L}{O}$

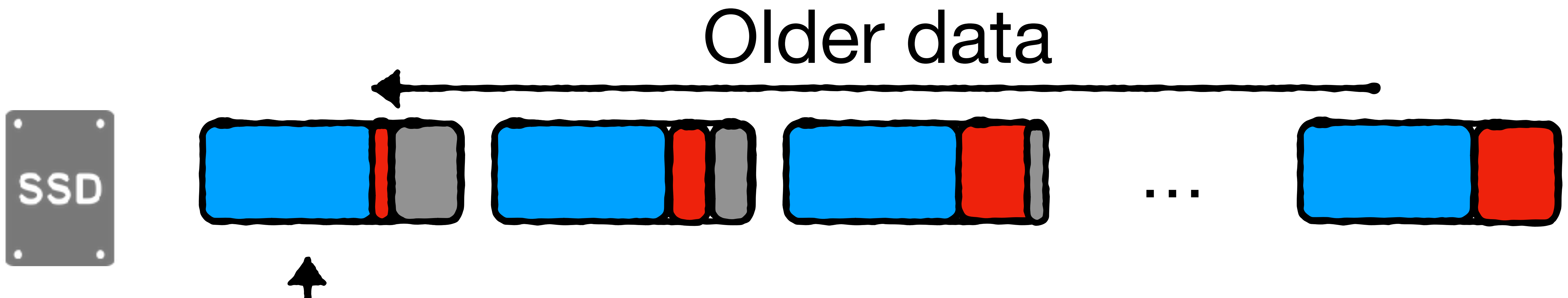
Question 3 - Hot/Cold Data Separation

Consider a circular log where the physical capacity consists of 70% static data (never updated), 10% hot data, and 20% over-provisioning.

(A) Estimate a lower bound and upper bound for write-amplification assuming no hot/cold data separation.

Let C=0.7, H=0.1 and O=0.2

For lower bound, let's use our uniform workload distribution estimation.



Garbage-collect

It works as a Lower bound since static data can only increase fraction of valid data in areas we garbage-collect

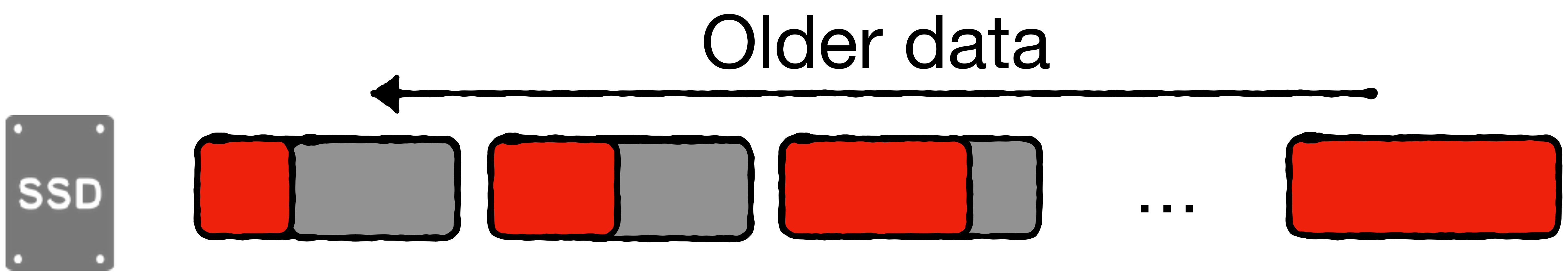
Question 3 - Hot/Cold Data Separation

Consider a circular log where the physical capacity consists of 70% static data (never updated), 10% hot data, and 20% over-provisioning.

(A) Estimate a lower bound and upper bound for write-amplification assuming no hot/cold data separation.

Let C=0.7, H=0.1 and O=0.2

For lower bound, let's use our uniform workload distribution estimation.



Lower bound: $= 1 + \frac{1}{2} \cdot \frac{L}{O} \quad = 1 + \frac{1}{2} \cdot \frac{H+C}{O} = 1 + \frac{1}{2} \cdot \frac{0.8}{0.2} = 3$

Question 3 - Hot/Cold Data Separation

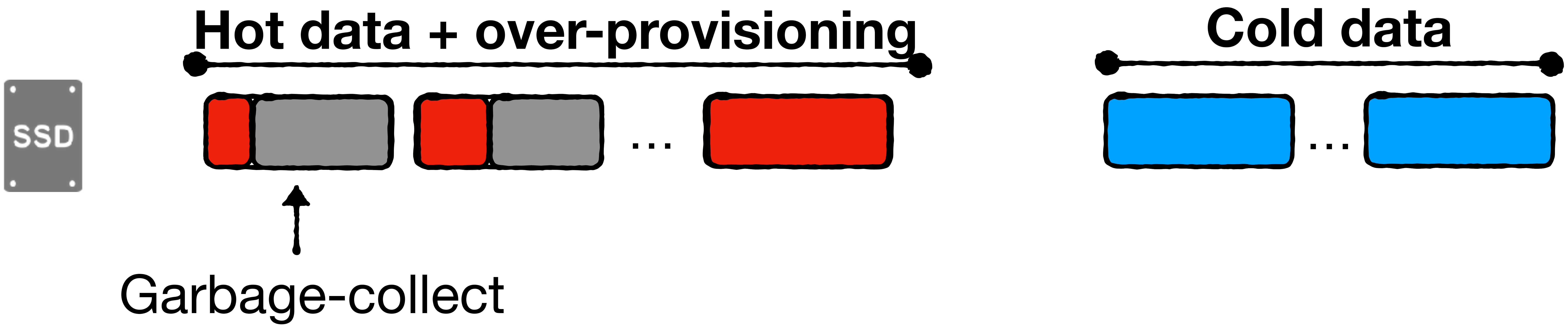
Consider a circular log where the physical capacity consists of 70% static data (never updated), 10% hot data, and 20% over-provisioning.

(A) Estimate a lower bound and upper bound for write-amplification assuming no hot/cold data separation.

(B) Estimate write-amplification assuming perfect hot/cold data separation.

Let C=0.7, H=0.1 and O=0.2

For hot/cold separation estimation, assume all over-provisioned space is applied on hot areas.



Question 3 - Hot/Cold Data Separation

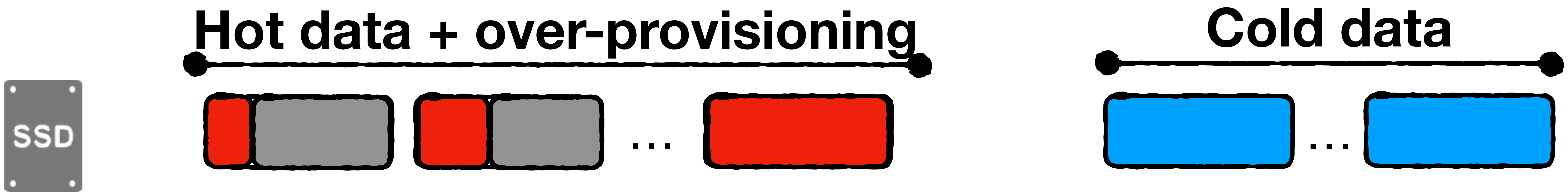
Consider a circular log where the physical capacity consists of 70% static data (never updated), 10% hot data, and 20% over-provisioning.

(A) Estimate a lower bound and upper bound for write-amplification assuming no hot/cold data separation.

(B) Estimate write-amplification assuming perfect hot/cold data separation.

Let C=0.7, H=0.1 and O=0.2

For hot/cold separation estimation, assume all over-provisioned space is applied on hot areas.



Estimation: $= 1 + \frac{1}{2} \cdot \frac{L}{O} = 1 + \frac{1}{2} \cdot \frac{H}{O} = 1 + \frac{1}{2} \cdot \frac{0.1}{0.2} = 1.25$