CS460: Intro to Database Systems

Class 16: Log-Structured-Merge Trees

Instructor: Manos Athanassoulis

https://bu-disc.github.io/CS460/

based on slides from Niv Dayan

Useful when?

- ☑ Massive dataset
- ☑ Rapid updates/insertions
- ☑ Fast lookups

$\Rightarrow$ LSM-trees are for you.

Why now?

Patrick O'Neil
UMass Boston

Invented in 1996

DynamoDB

1980 1990 2000 2010

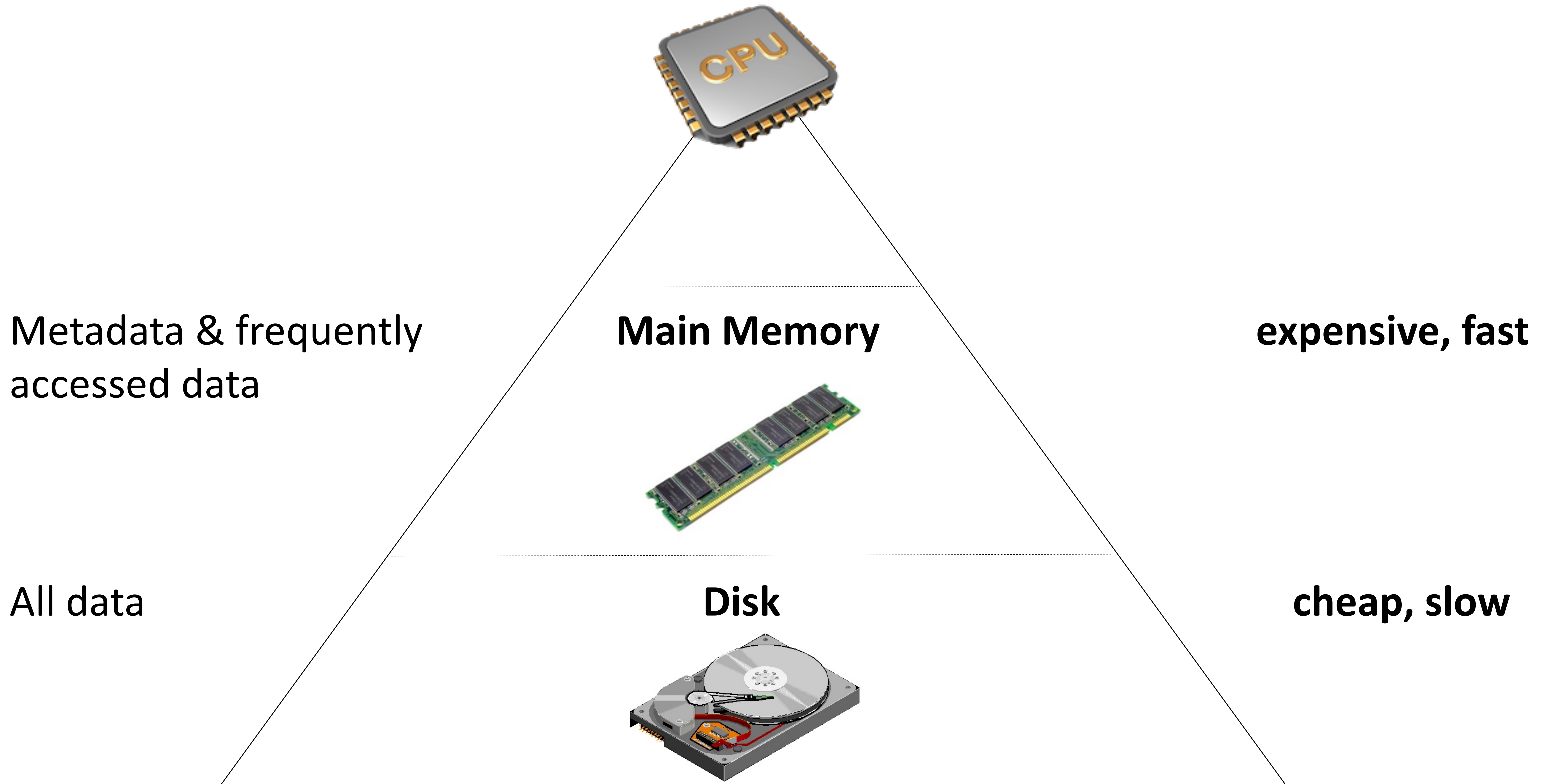
Time

Outline

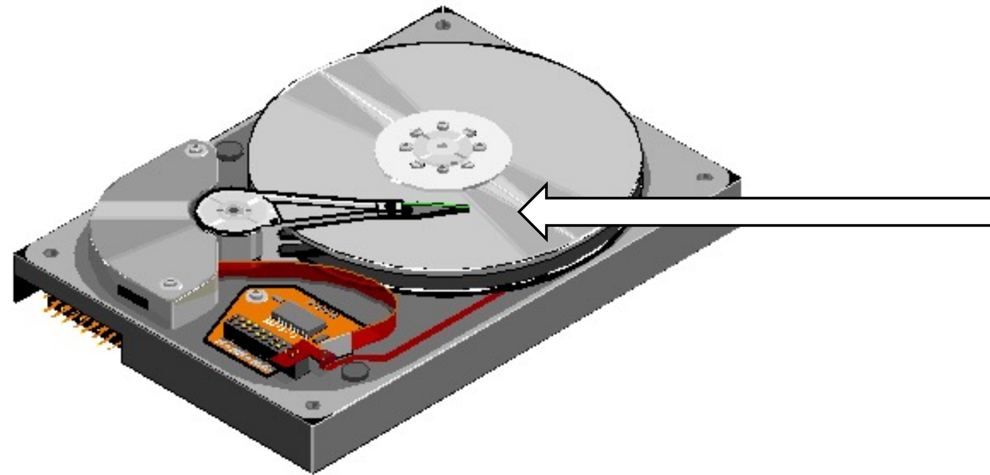
1. Storage devices
2. Indexing problem & basic solutions
3. Basic LSM-trees
4. Leveled LSM-trees
5. Tiered LSM-trees
6. Bloom filters

Storage devices

The Memory Hierarchy



Why is disk slow?



Random access is slow move disk head

Sequential access is faster let disk spin

Outline

1. **Storage devices**
2. Indexing problem & basic solutions
3. Basic LSM-trees
4. Leveled LSM-trees
5. Tiered LSM-trees
6. Bloom filters

Outline

1. Storage devices
2. **Indexing problem & basic solutions**
3. Basic LSM-trees
4. Leveled LSM-trees
5. Tiered LSM-trees
6. Bloom filters

Indexing Problem & Basic Solutions

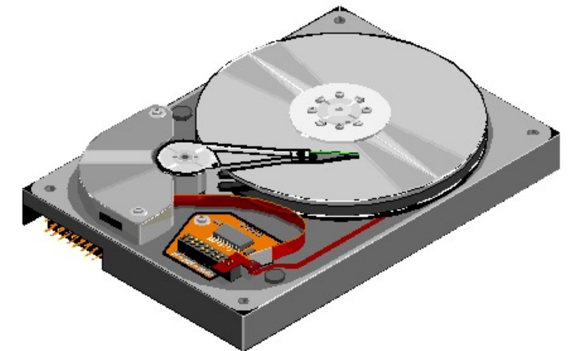
Indexing Problem

names ⟹ phone numbers

Structure on disk?

Lookup cost?

Insertion cost?



Results Catalogue

Compare and contrast data structures.

What to use when?

Data Structure	Lookup cost	Insertion cost
Sorted array		
Log		
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue

Compare and contrast data structures.

What to use when?

Data Structure	Lookup cost	Insertion cost
Sorted array		
Log		
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

Sorted Array

Measure Performance in I/Os

$\mathfrak{n}$ entries

$\boldsymbol{B}$ entries fit into a disk block

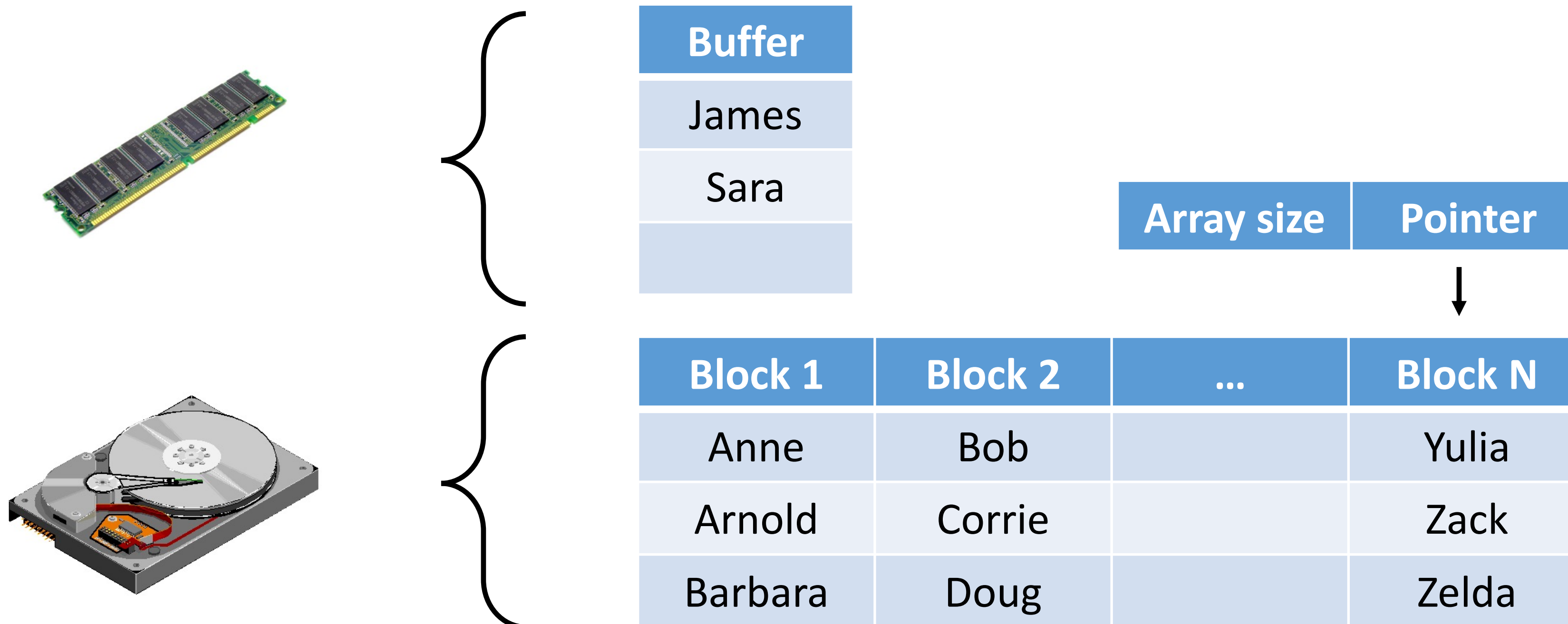
Array spans $\mathbf{N} = \frac{\mathfrak{n}}{\boldsymbol{B}}$ disk blocks

Lookup method & cost?

Binary search: $O(\log_2(N))$ I/Os

Insertion cost?

Push entries: $O\left(\frac{1}{B} \cdot N\right)$ I/Os



Results Catalogue

	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log		
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue

	Lookup cost	**Insertion cost**
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log		
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

Log (append-only array)

$\mathfrak{n}$ entries

$\boldsymbol{B}$ entries fit into a disk block

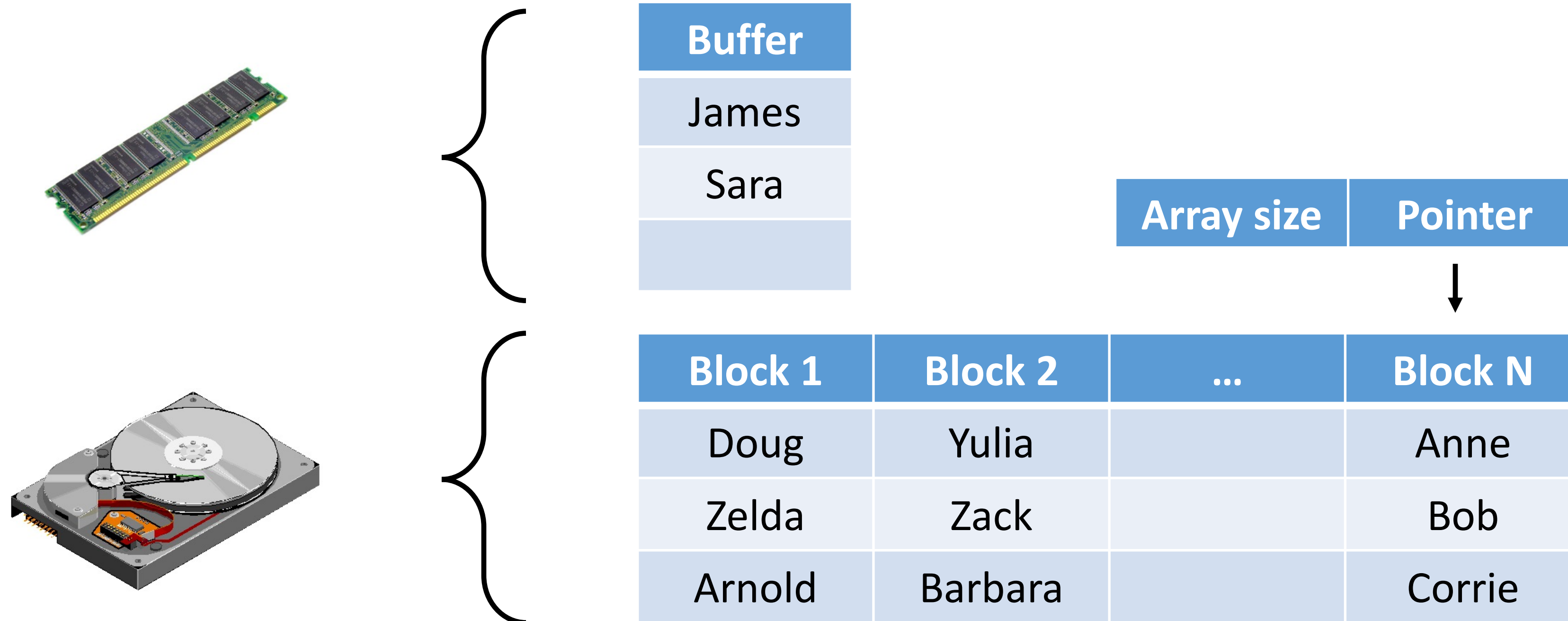
Array spans $\mathbf{N} = \frac{\mathfrak{n}}{\boldsymbol{B}}$ disk blocks

Lookup method & cost?

Scan: $O(N)$

Insertion cost?

Append: $O\left(\frac{1}{B}\right)$



Results Catalogue

	Lookup cost	**Insertion cost**
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue

	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

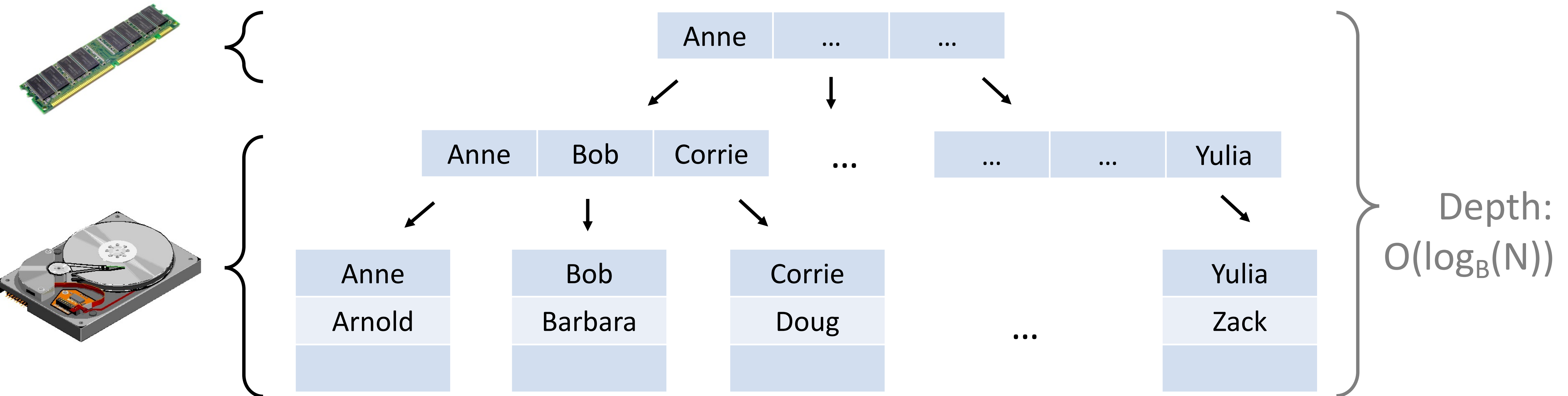
B-tree

Lookup method & cost?

Tree search: $O(\log_B(N))$

Insertion method & cost?

Tree search & append: $O(\log_B(N))$



Results Catalogue

	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

B-trees

Goetz Graefe Microsoft, HP Fellow, now
Google ACM Software System Award

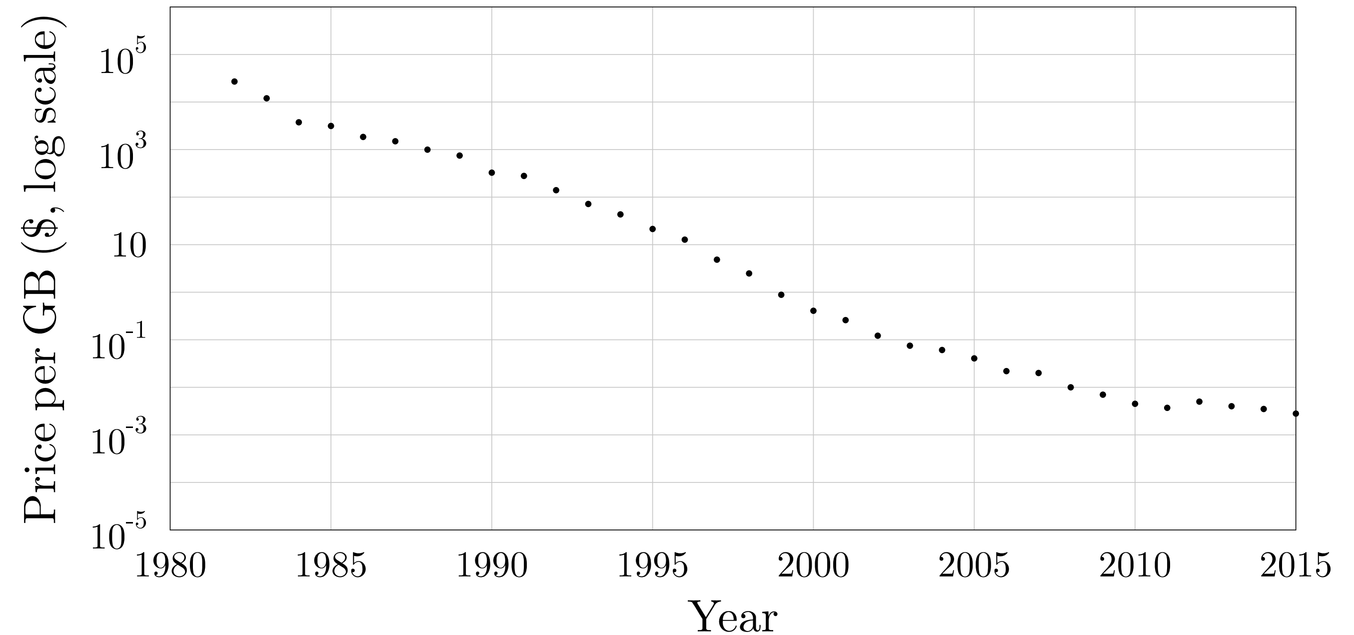
“It could be said that the world’s information is at our fingertips because of B-trees”

B-trees are no longer sufficient

Cheaper storage

Workloads more **insert-intensive**

We need **better insert-performance**



Results Catalogue

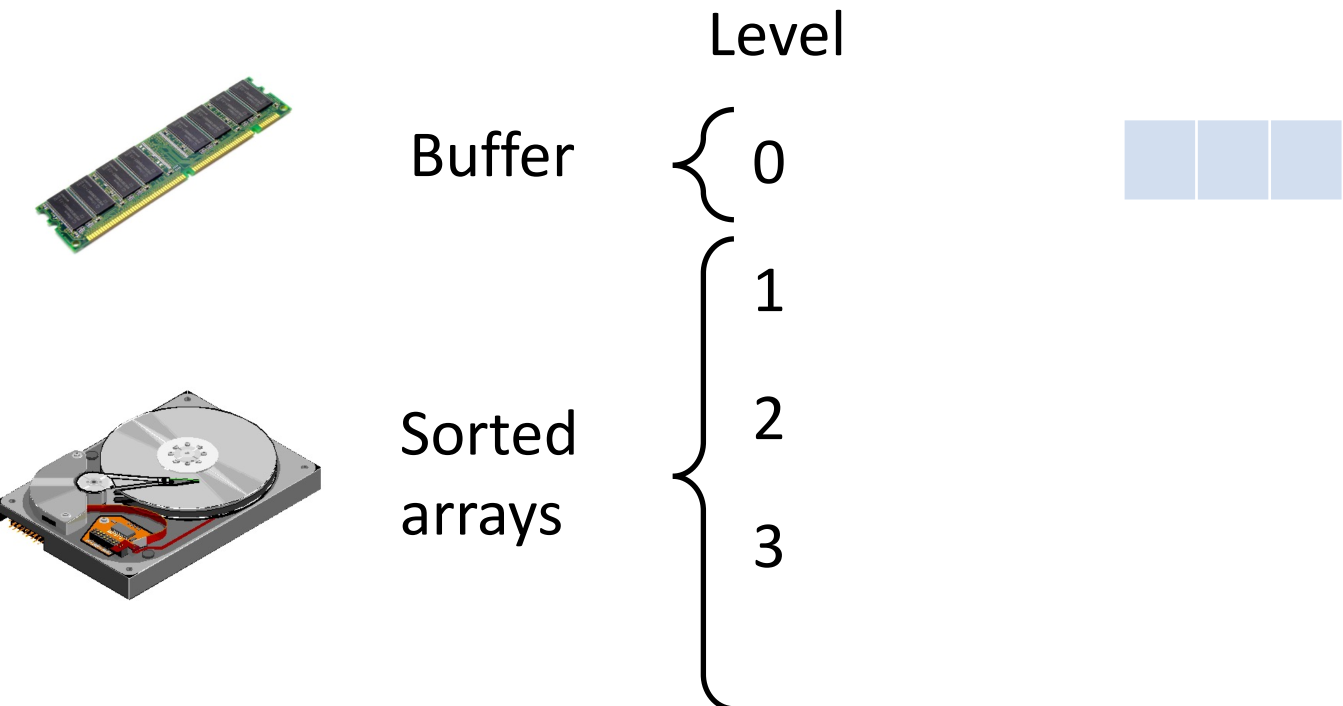
Goal to combine

sub-constant insertion cost
logarithmic lookup cost

	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log	$O(N)$	**$O(1/B)$**
B-tree	**$O(\log_B(N))$**	$O(\log_B(N))$
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

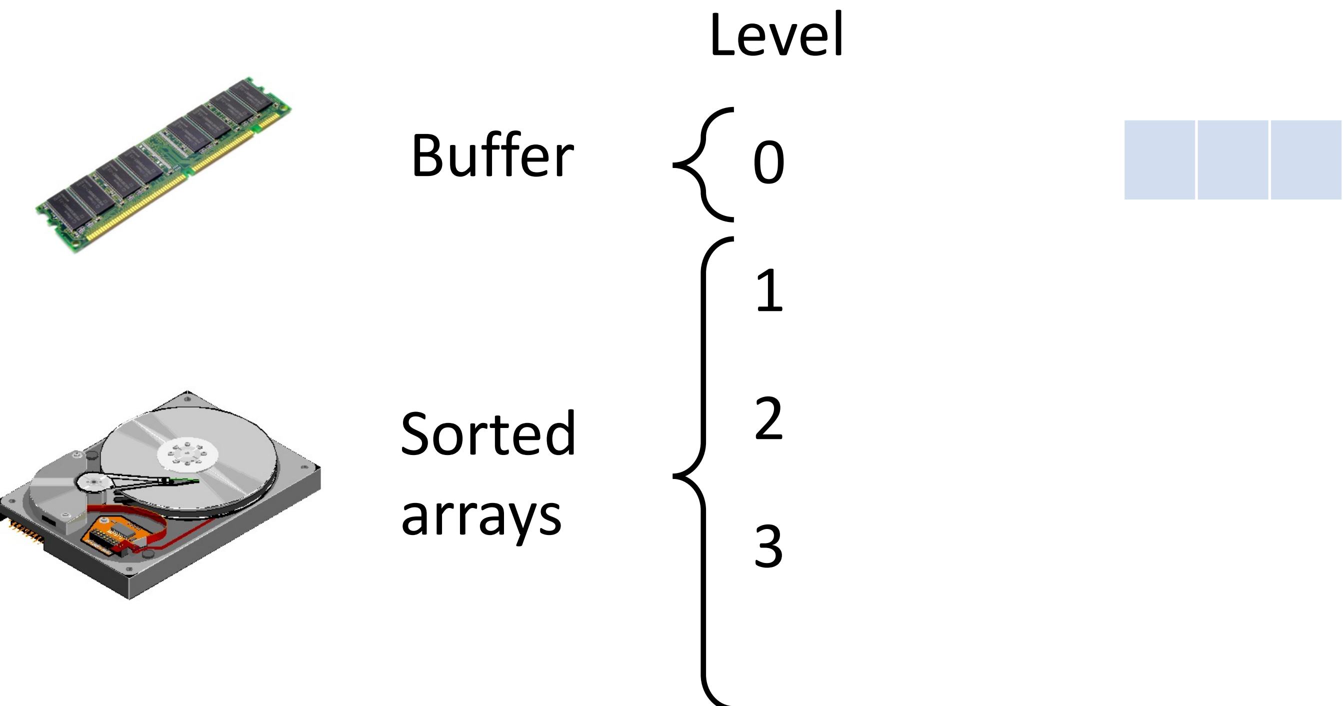
Basic LSM-trees

Basic LSM-tree



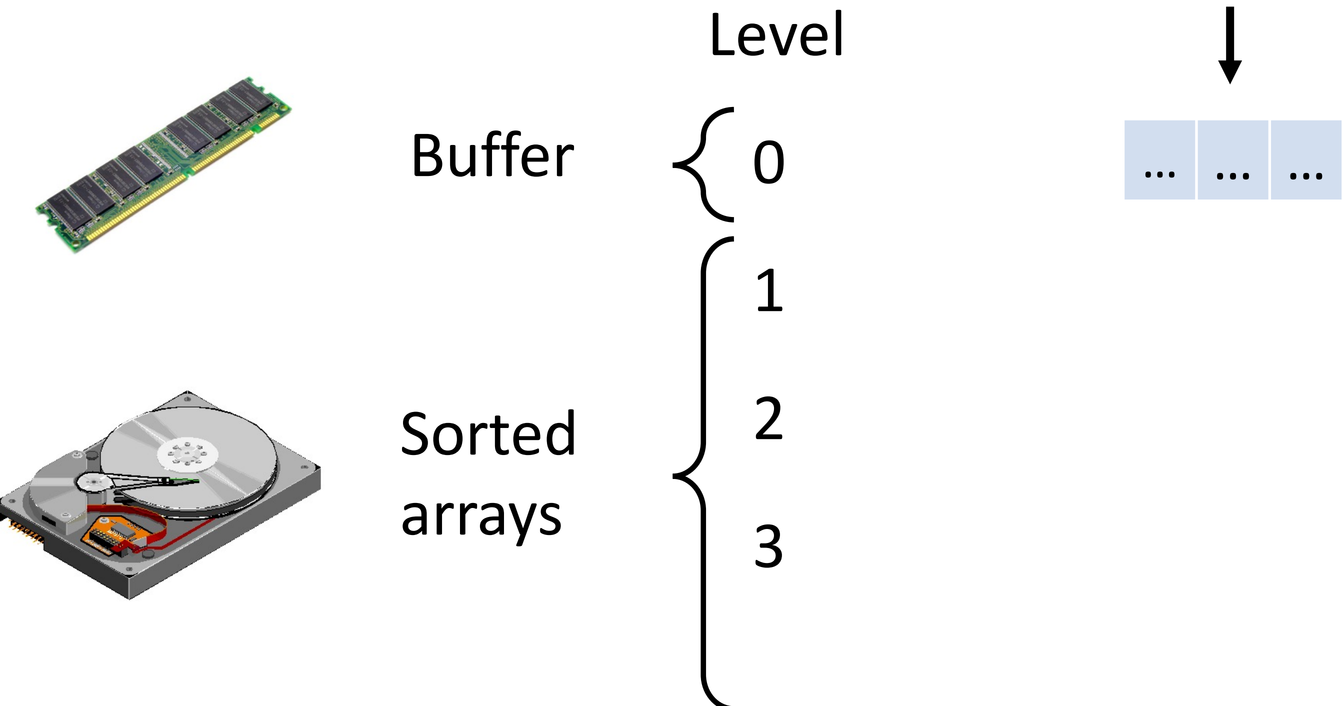
Basic LSM-tree

Design principle #1: optimize for insertions by buffering



Basic LSM-tree

Design principle #1: optimize for insertions by buffering

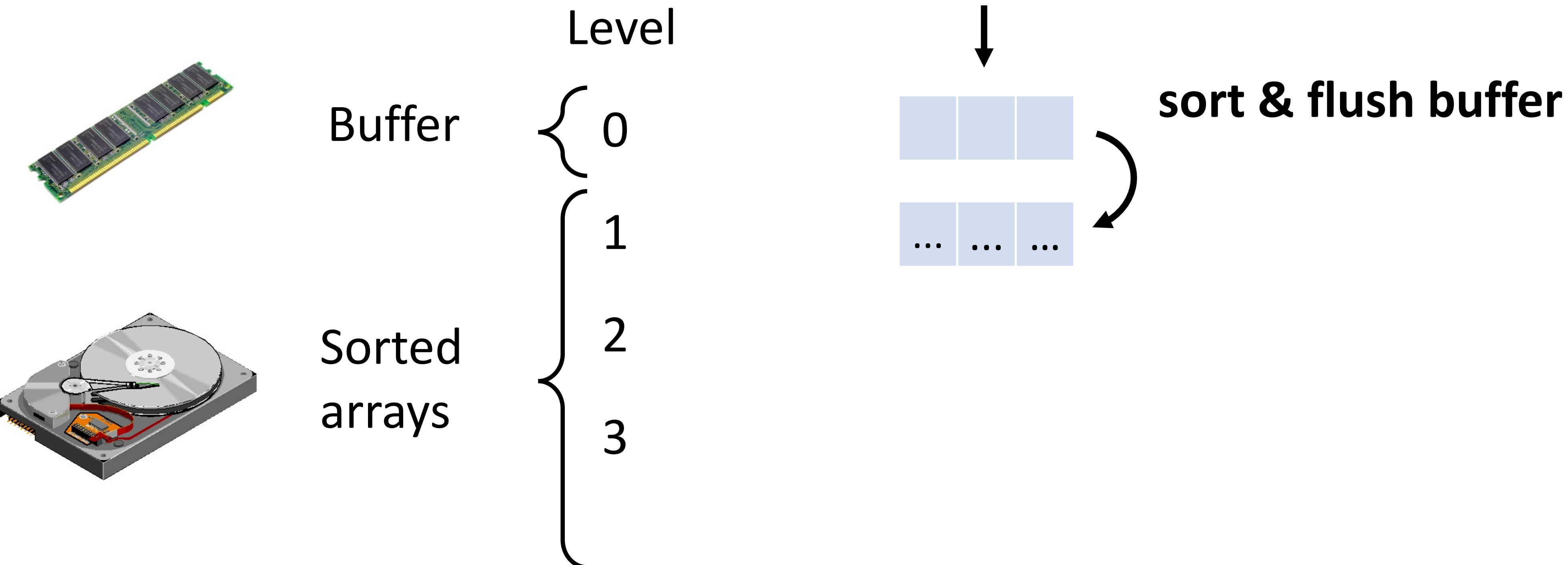


Basic LSM-tree

Design principle #1: optimize for insertions by buffering

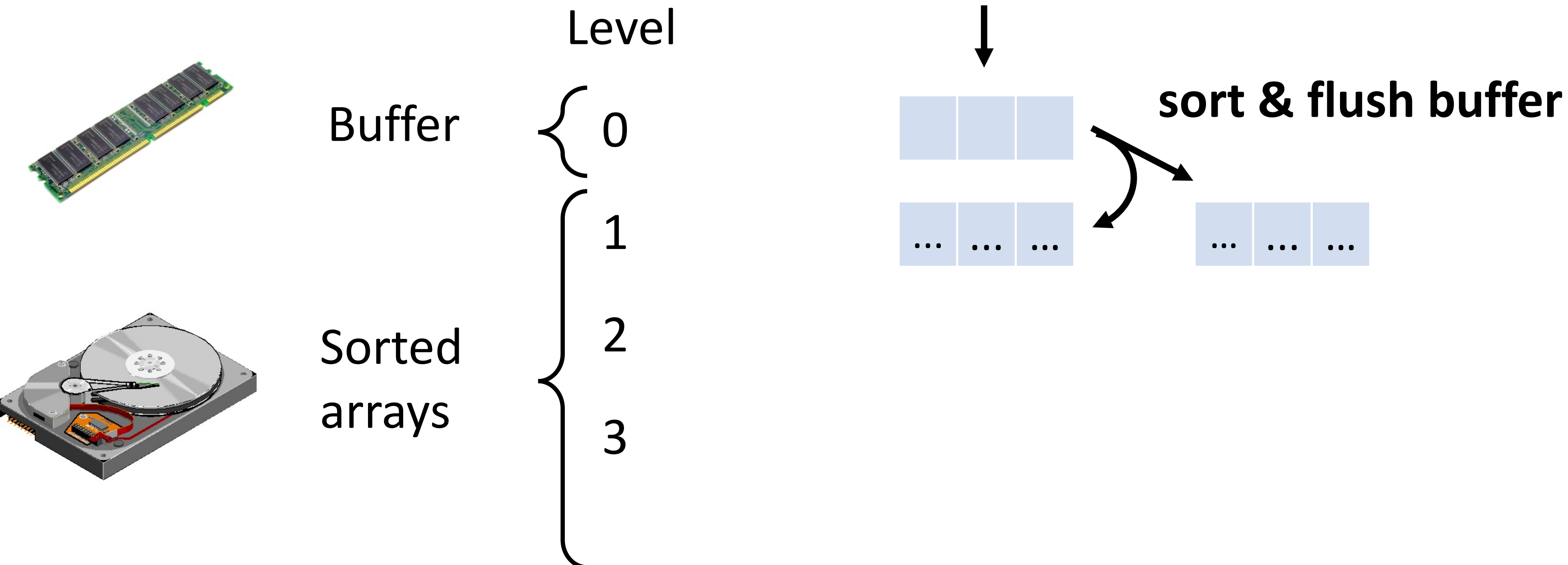
Inserts

sort & flush buffer



Basic LSM-tree

Design principle #1: optimize for insertions by buffering



Inserts

sort & flush buffer

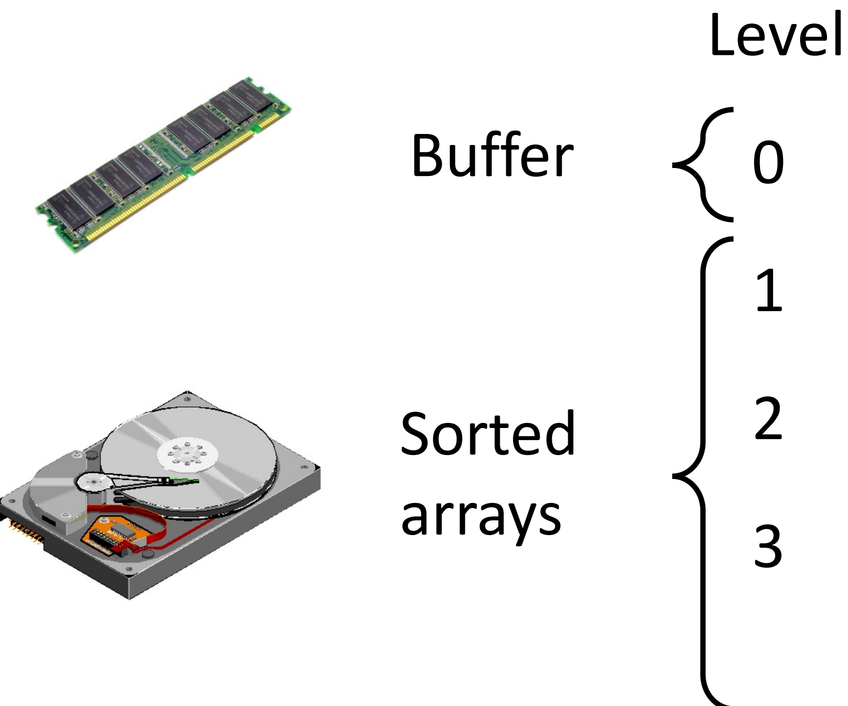
... ...

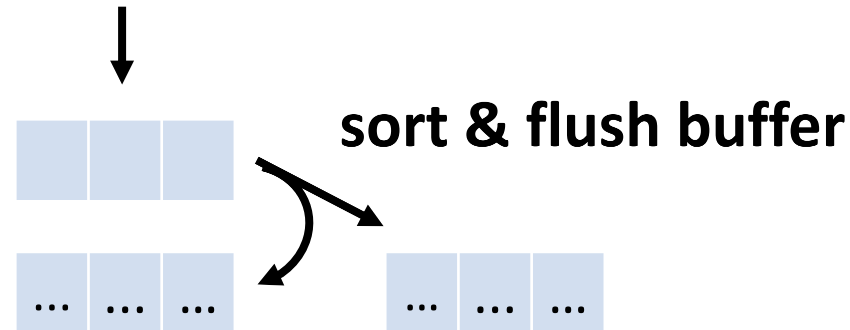
... ...

Basic LSM-tree

Design principle #1: optimize for insertions by buffering

Design principle #2: optimize for lookups by sort-merging arrays

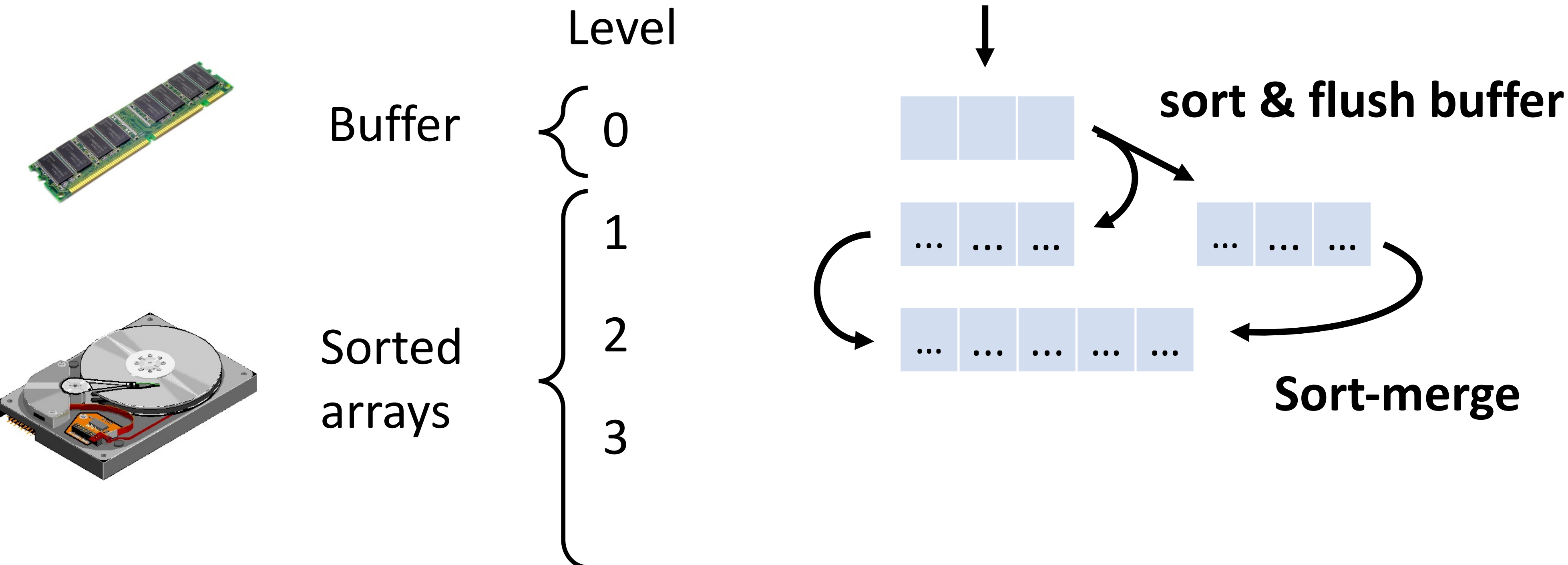




Basic LSM-tree

Design principle #1: optimize for insertions by buffering

Design principle #2: optimize for lookups by sort-merging arrays



Basic LSM-tree

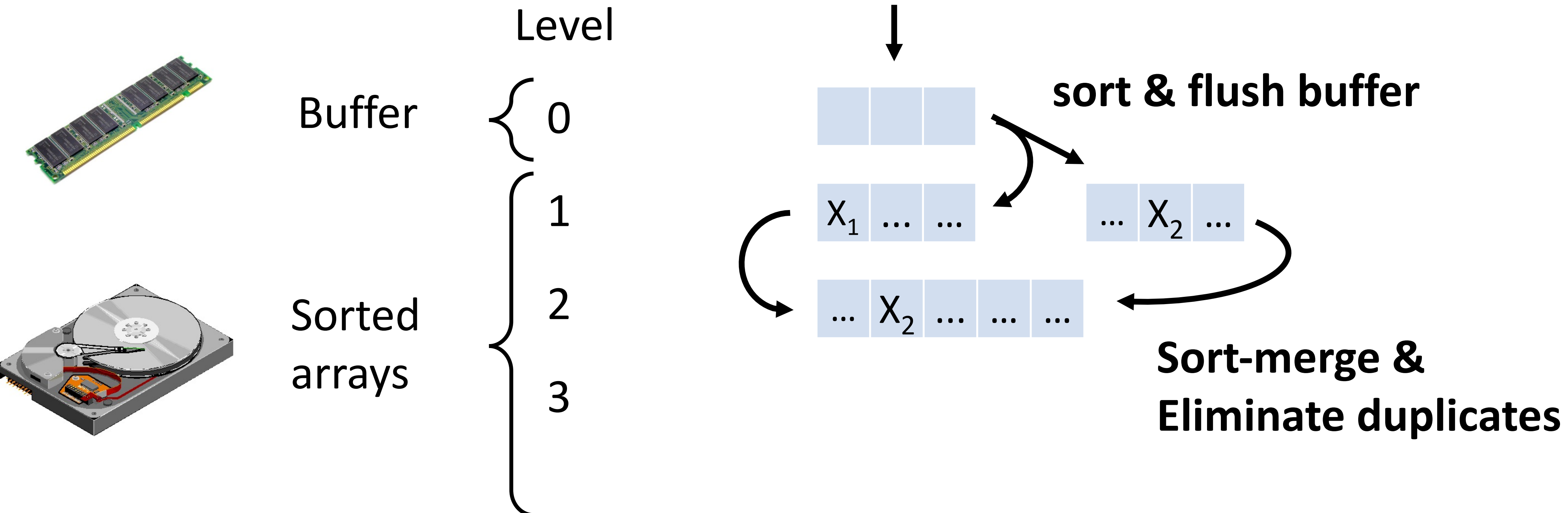
Design principle #1: optimize for insertions by buffering

Design principle #2: optimize for lookups by sort-merging arrays

Basic LSM-tree

Design principle #1: optimize for insertions by buffering

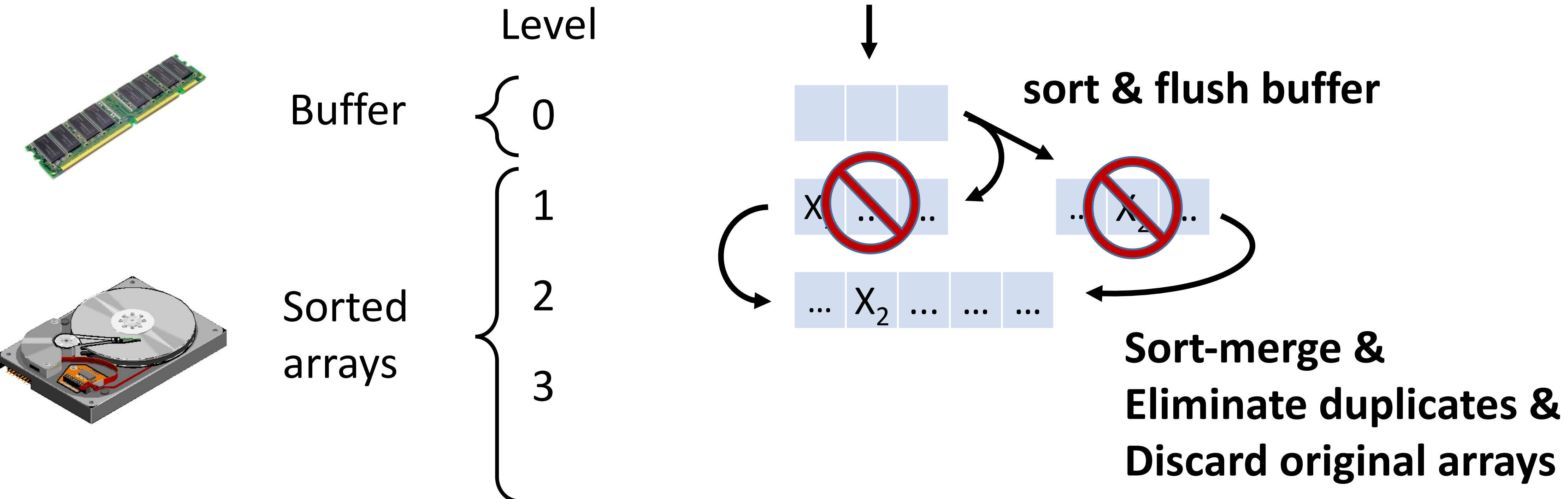
Design principle #2: optimize for lookups by sort-merging arrays



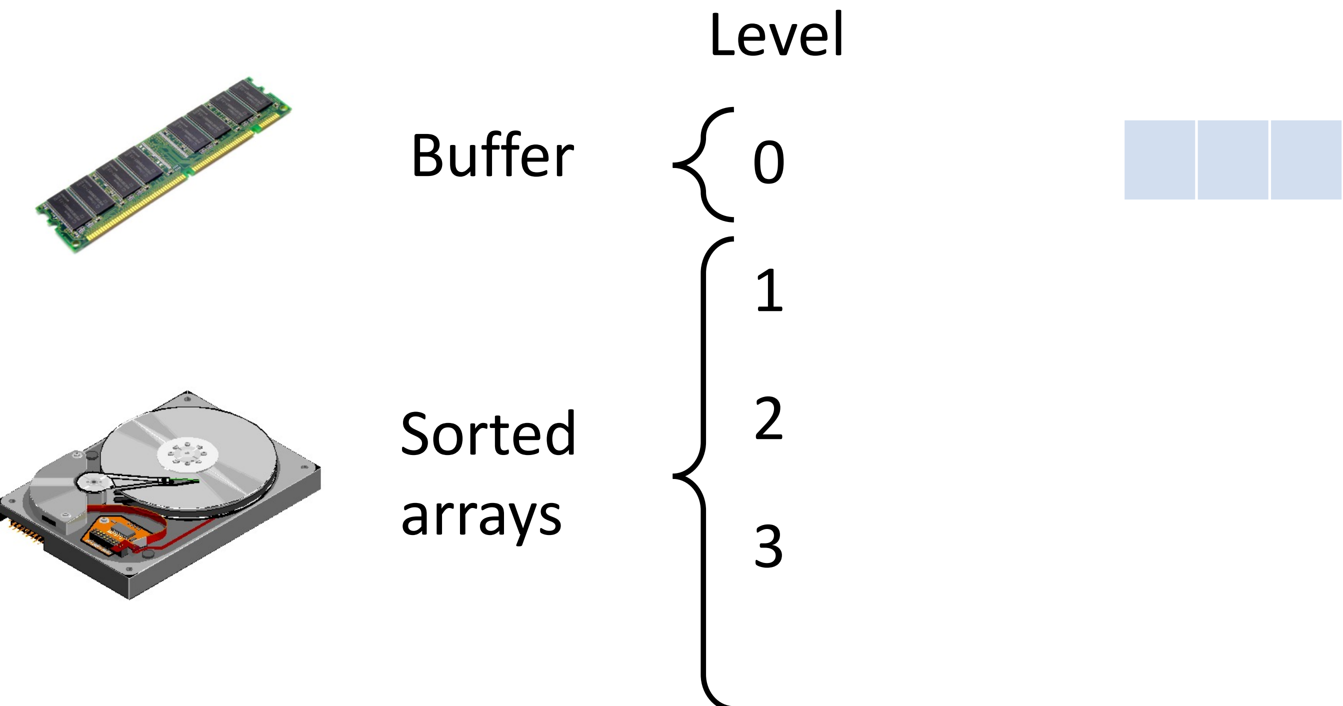
Basic LSM-tree

Design principle #1: optimize for insertions by buffering

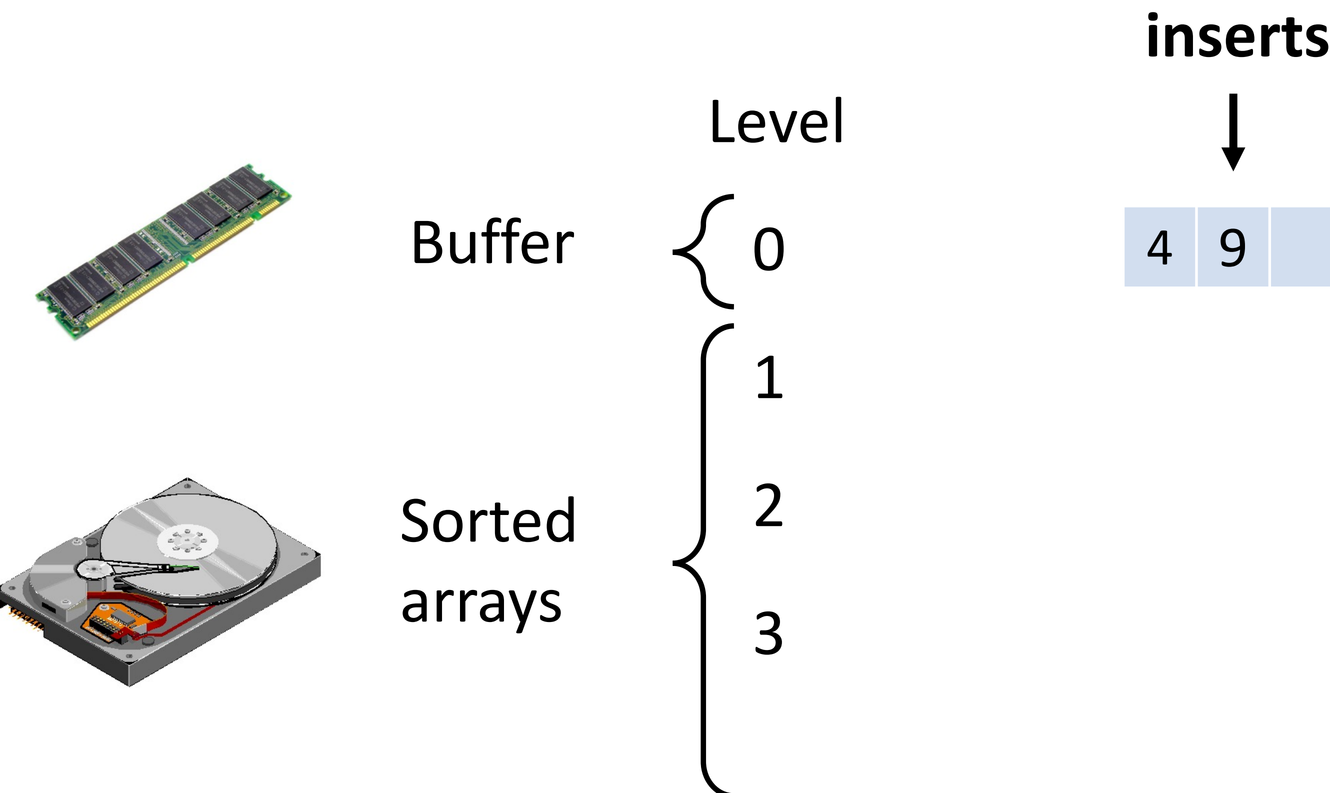
Design principle #2: optimize for lookups by sort-merging arrays



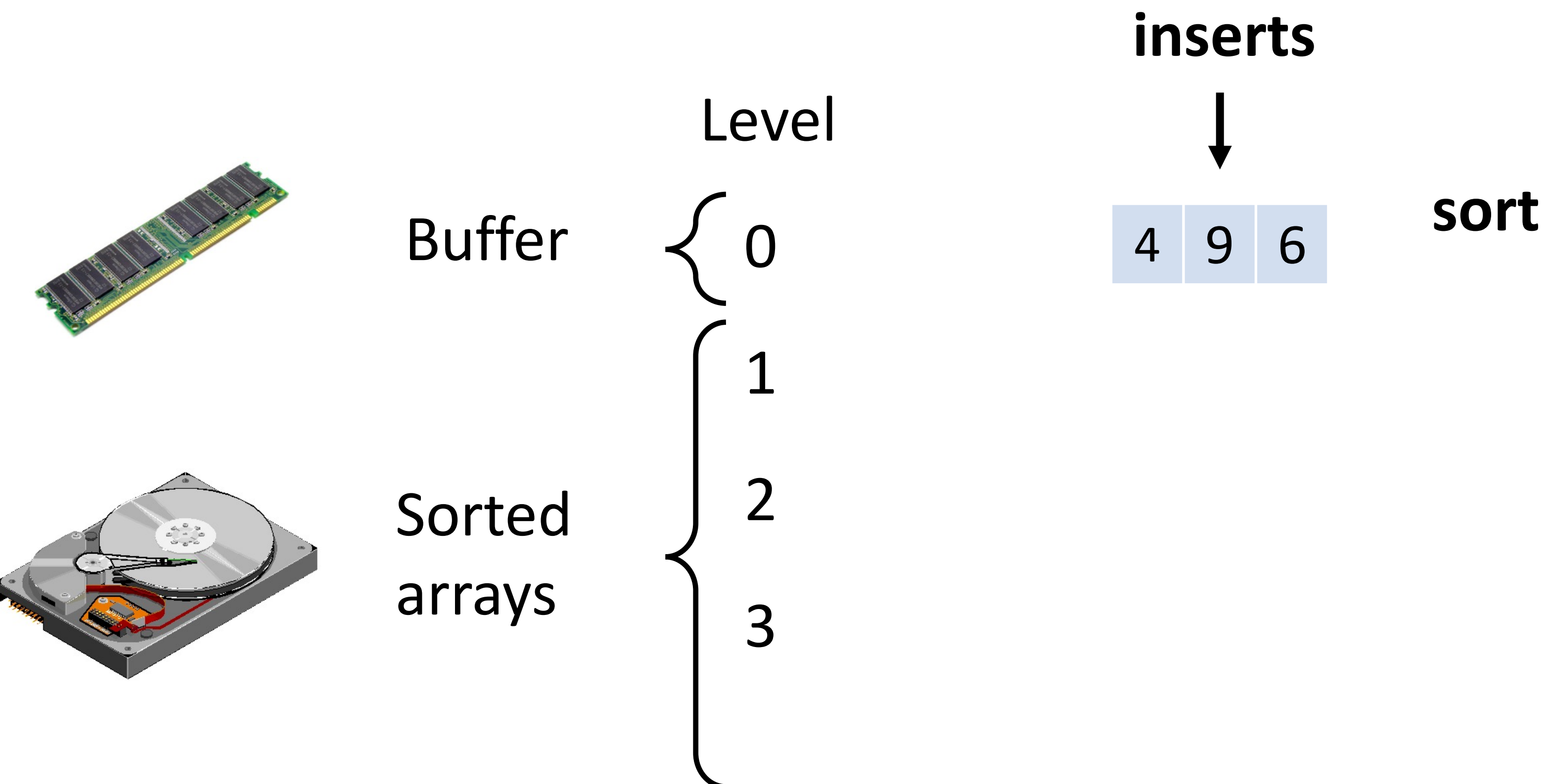
Basic LSM-tree – Example



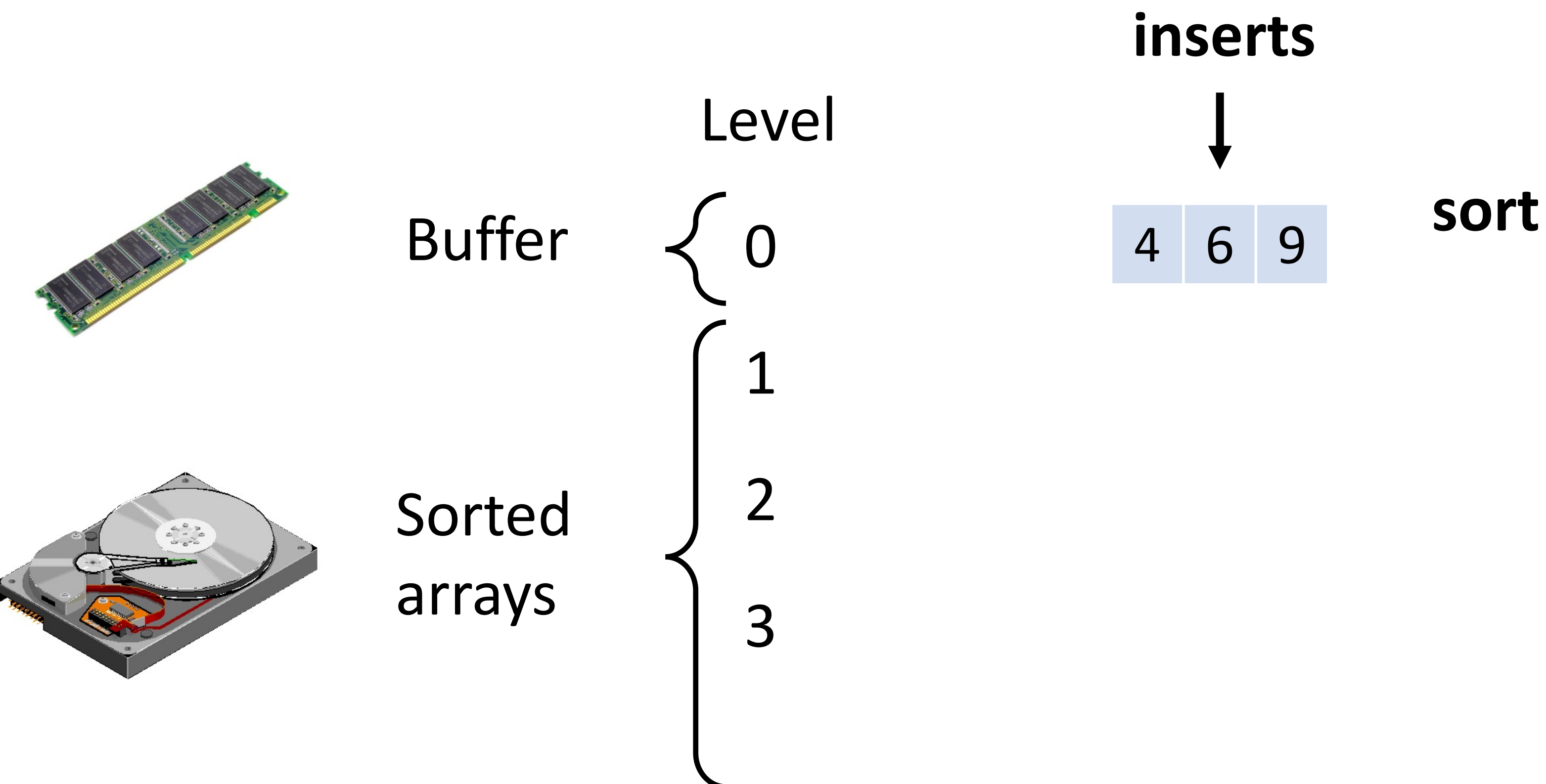
Basic LSM-tree – Example



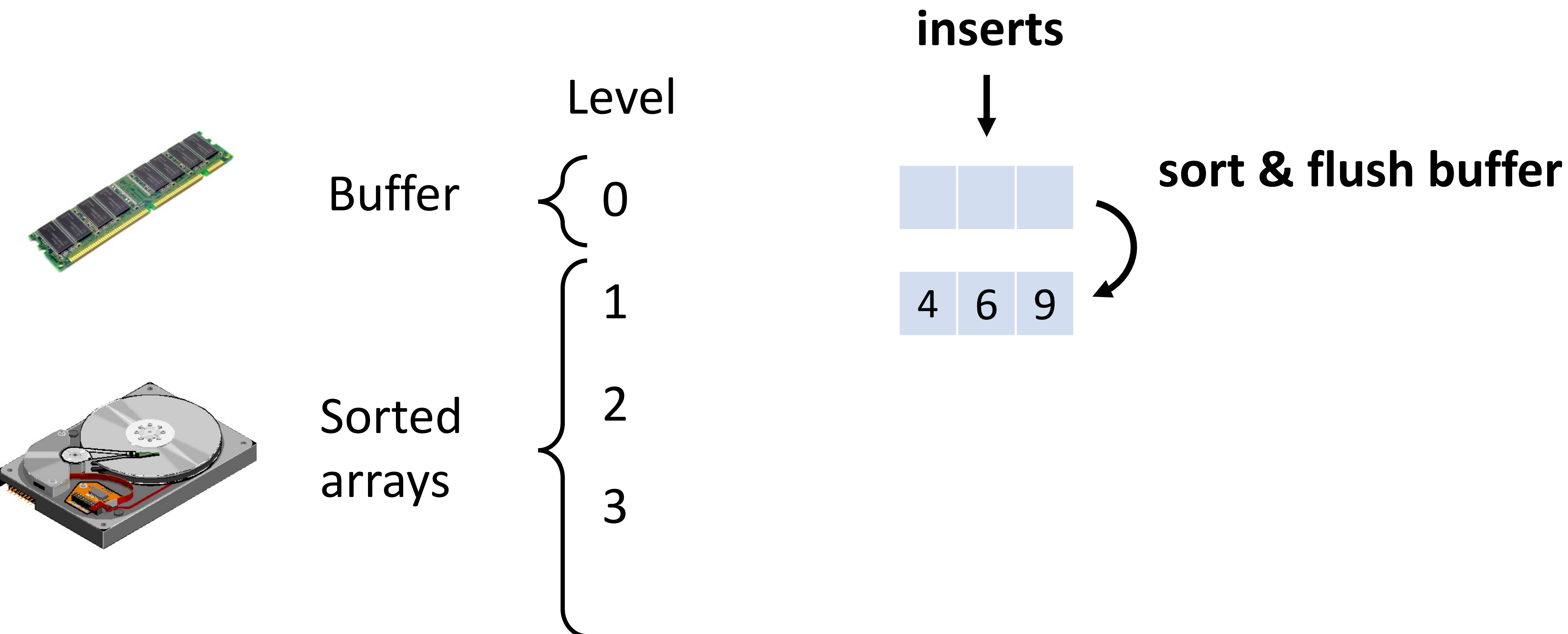
Basic LSM-tree – Example



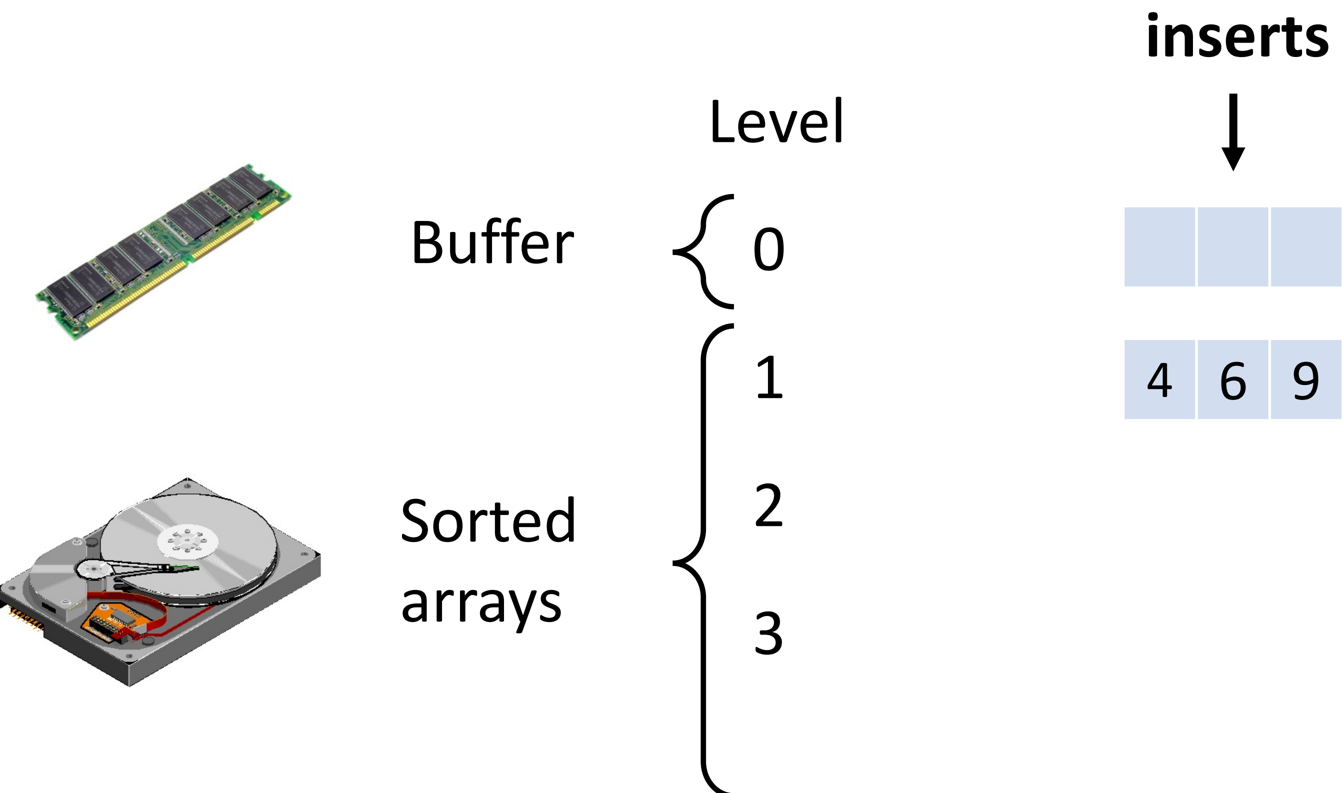
Basic LSM-tree – Example



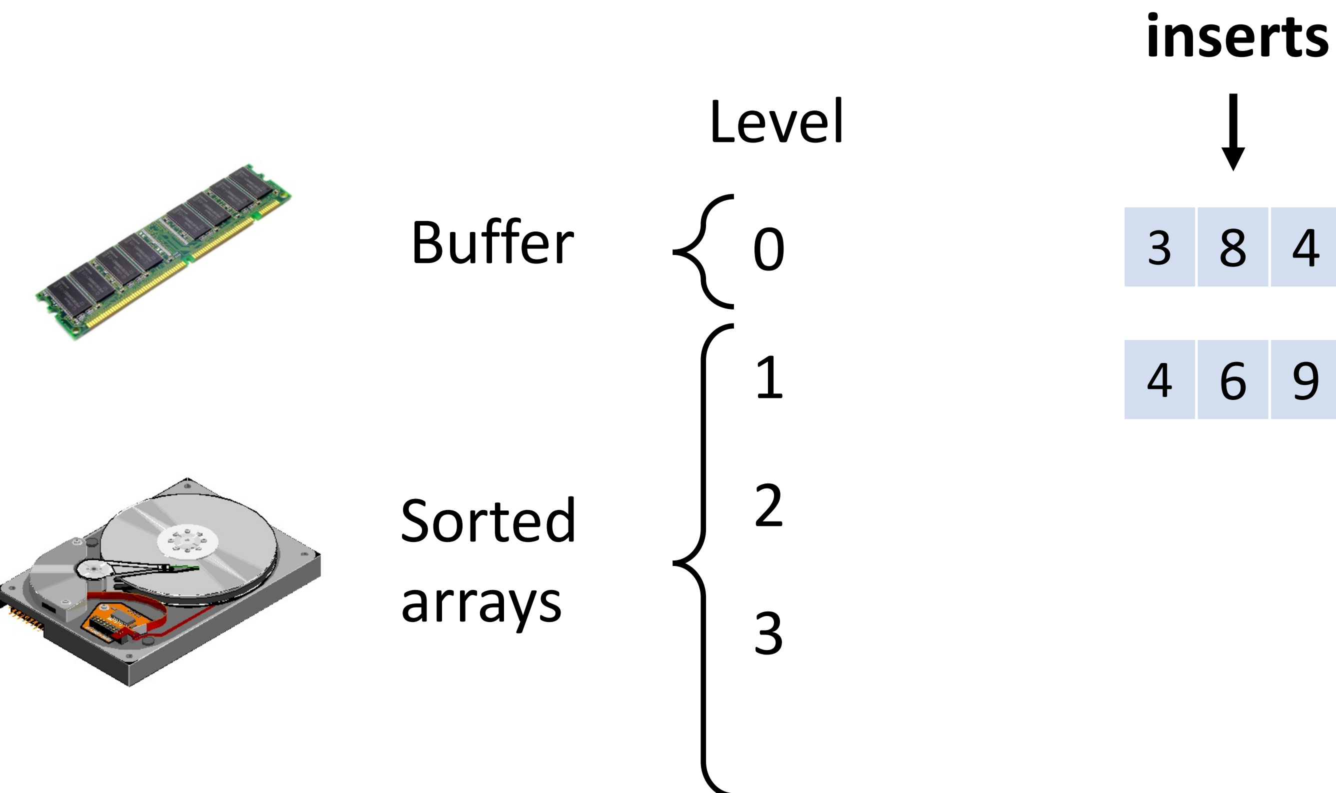
Basic LSM-tree – Example



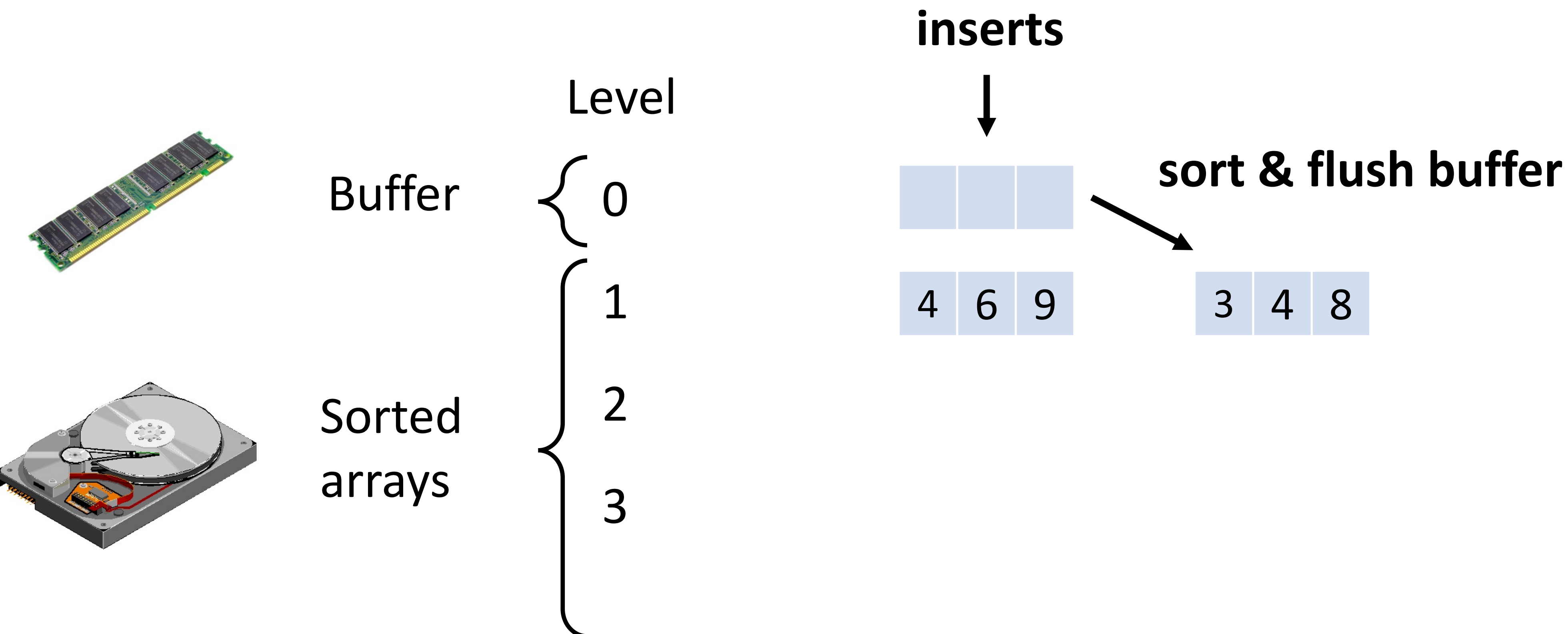
Basic LSM-tree – Example



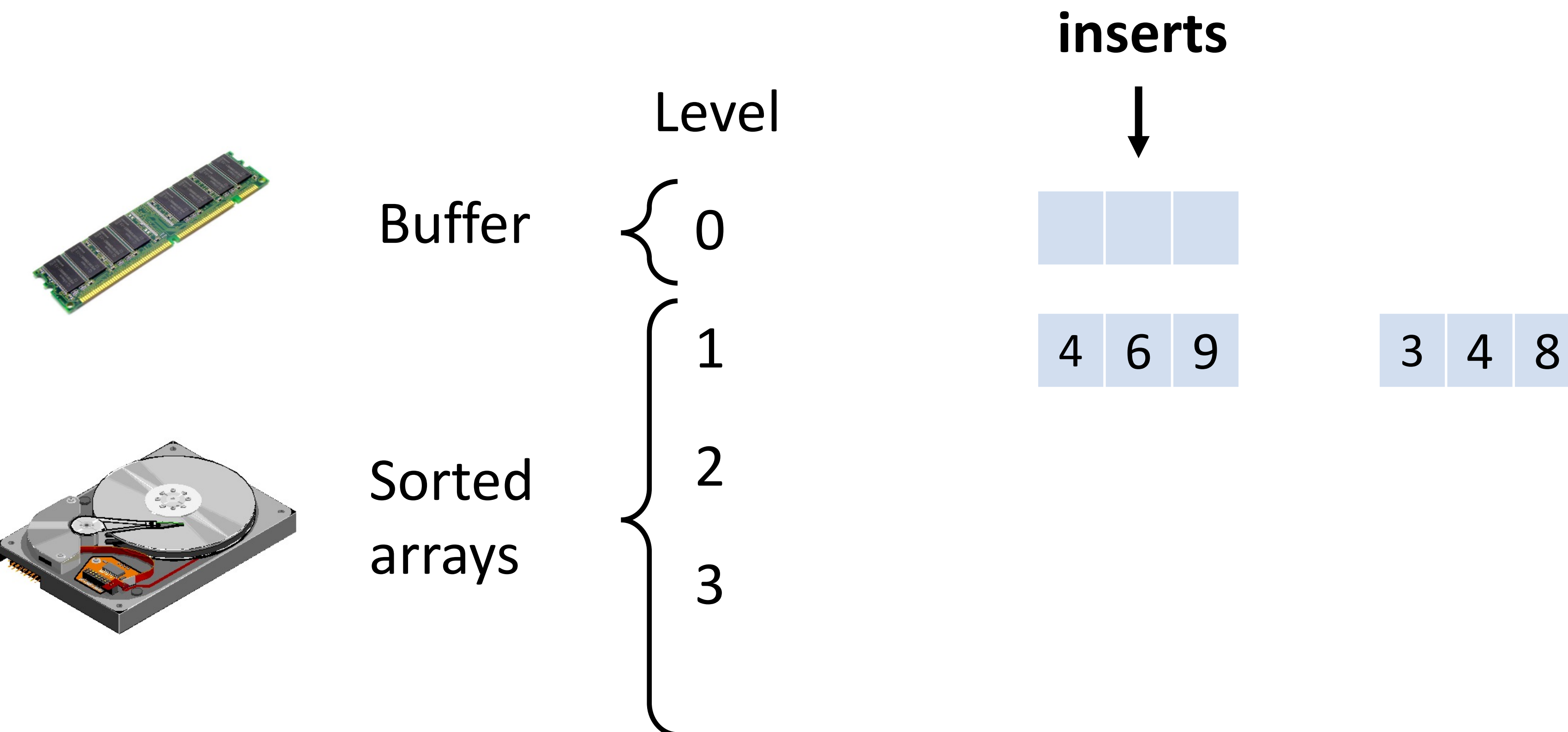
Basic LSM-tree – Example



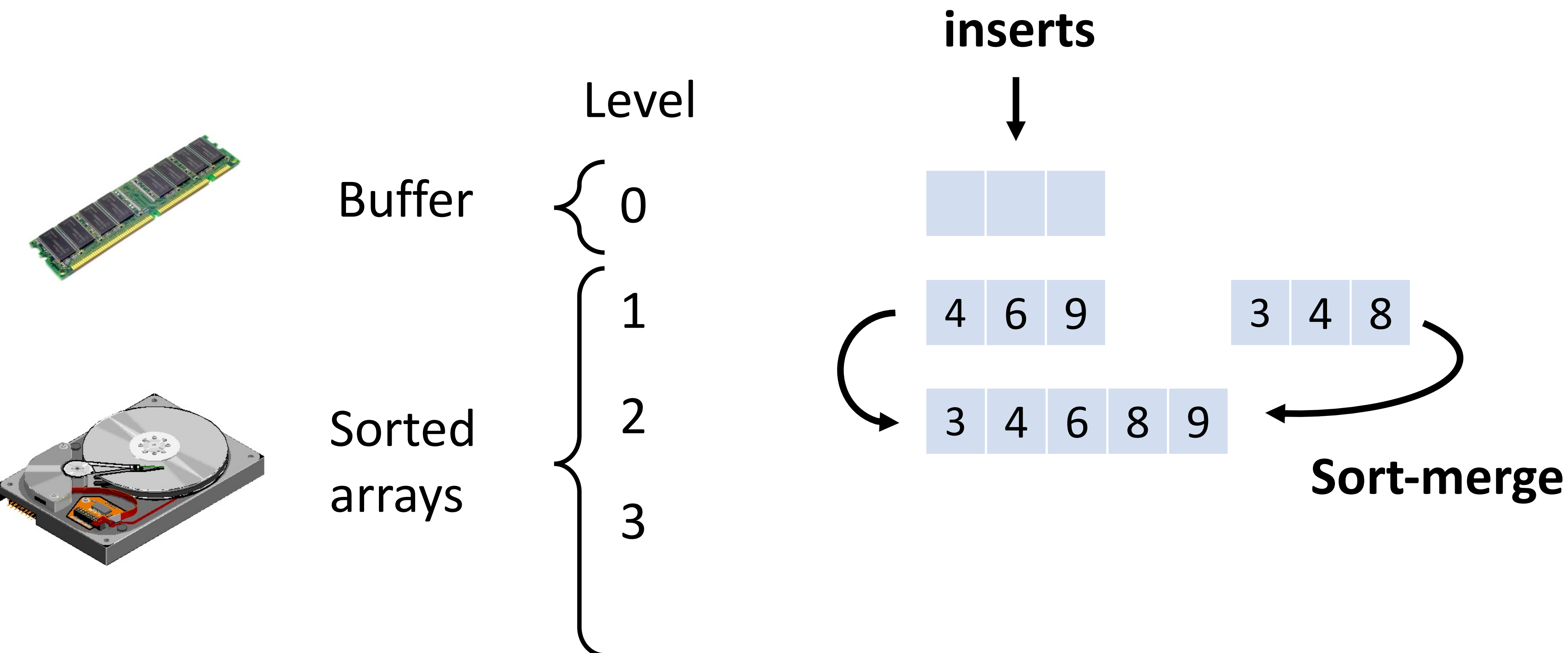
Basic LSM-tree – Example



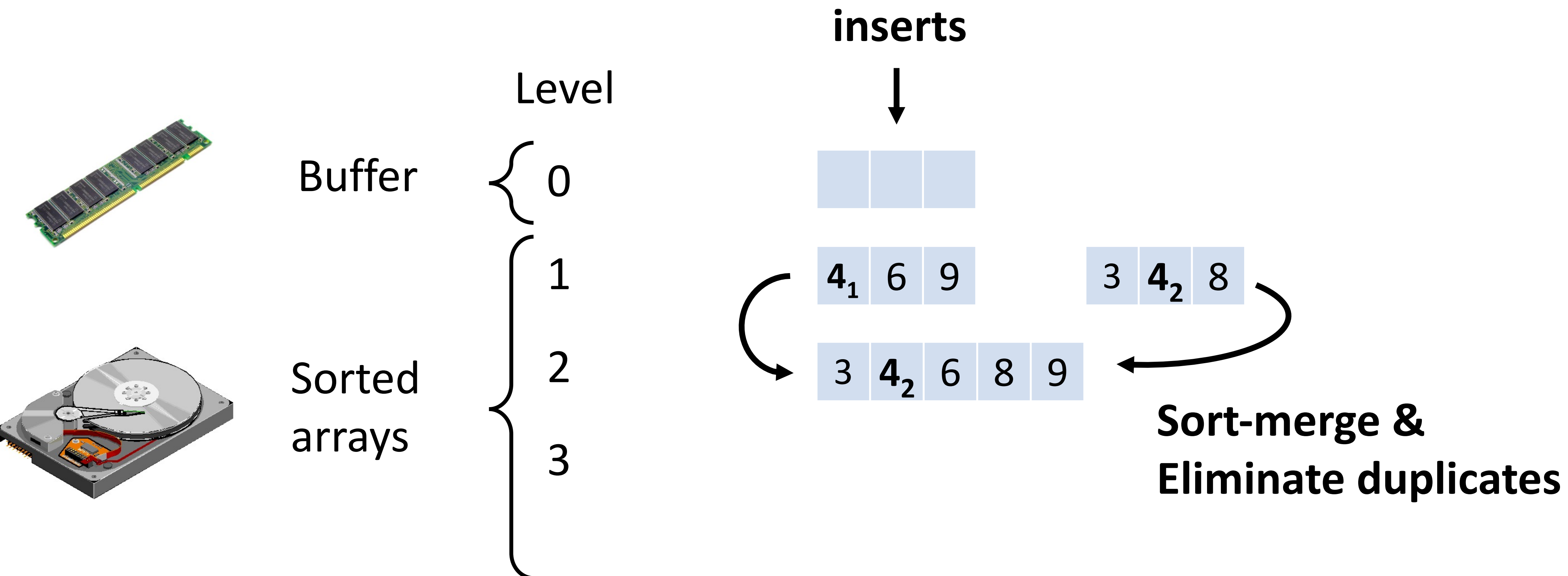
Basic LSM-tree – Example



Basic LSM-tree – Example



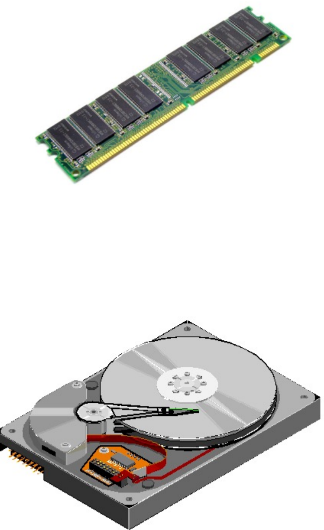
Basic LSM-tree – Example



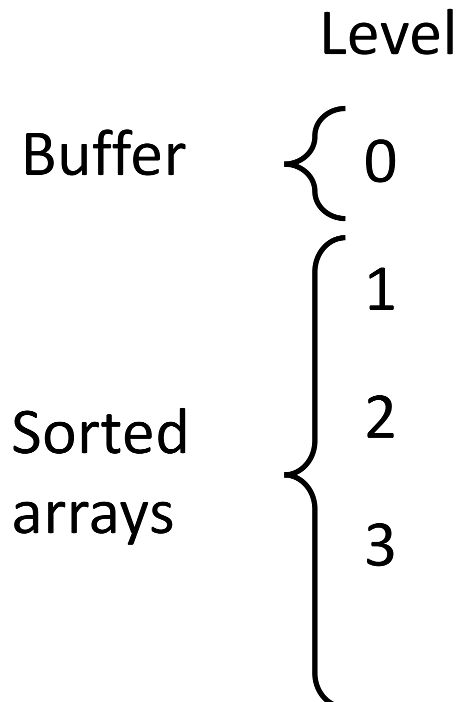
Basic LSM-tree – Example

inserts

Level



Buffer



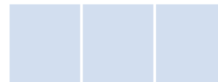
0

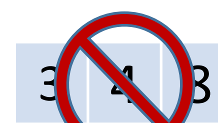
1

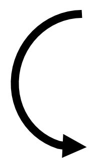
2

3

Sorted arrays



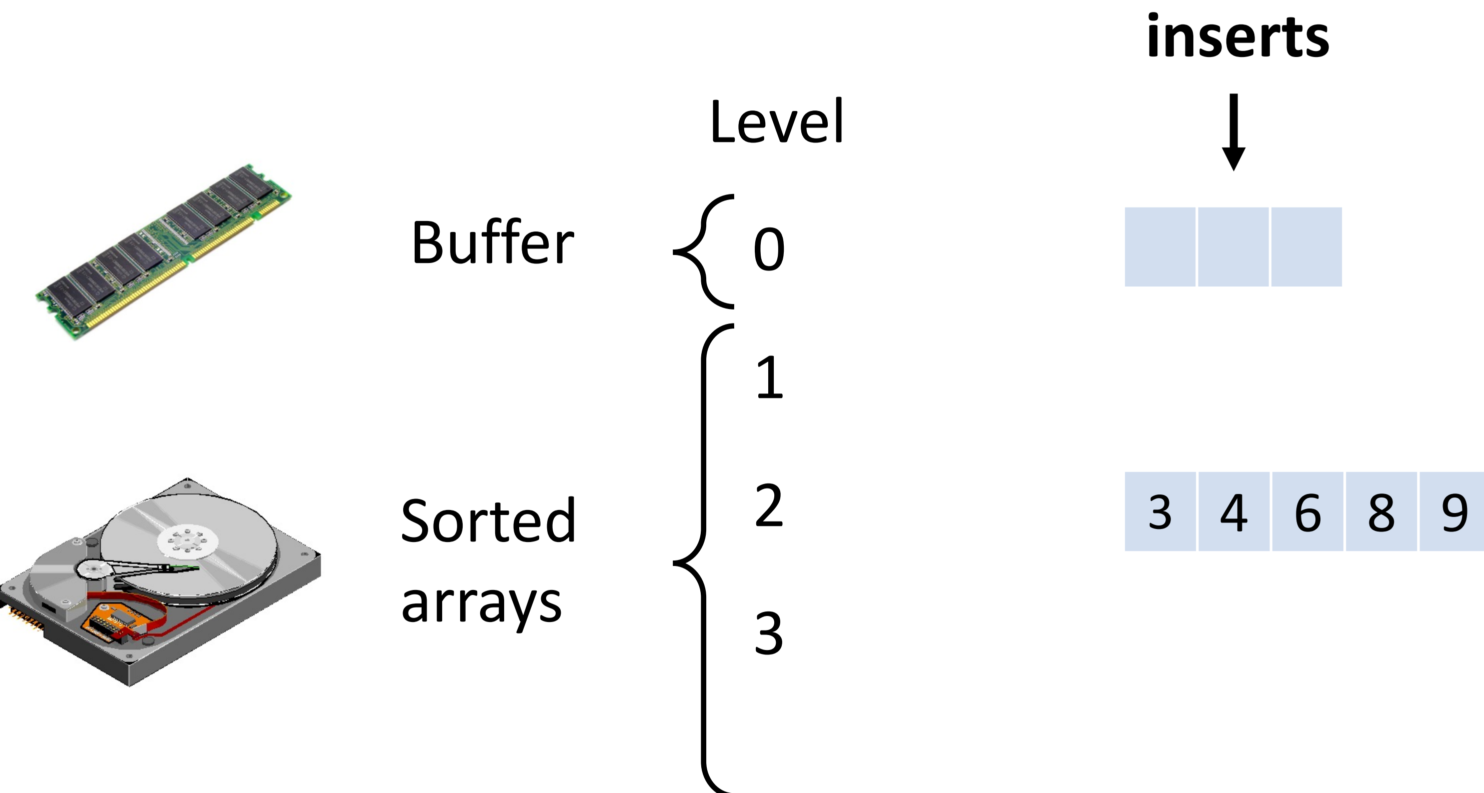




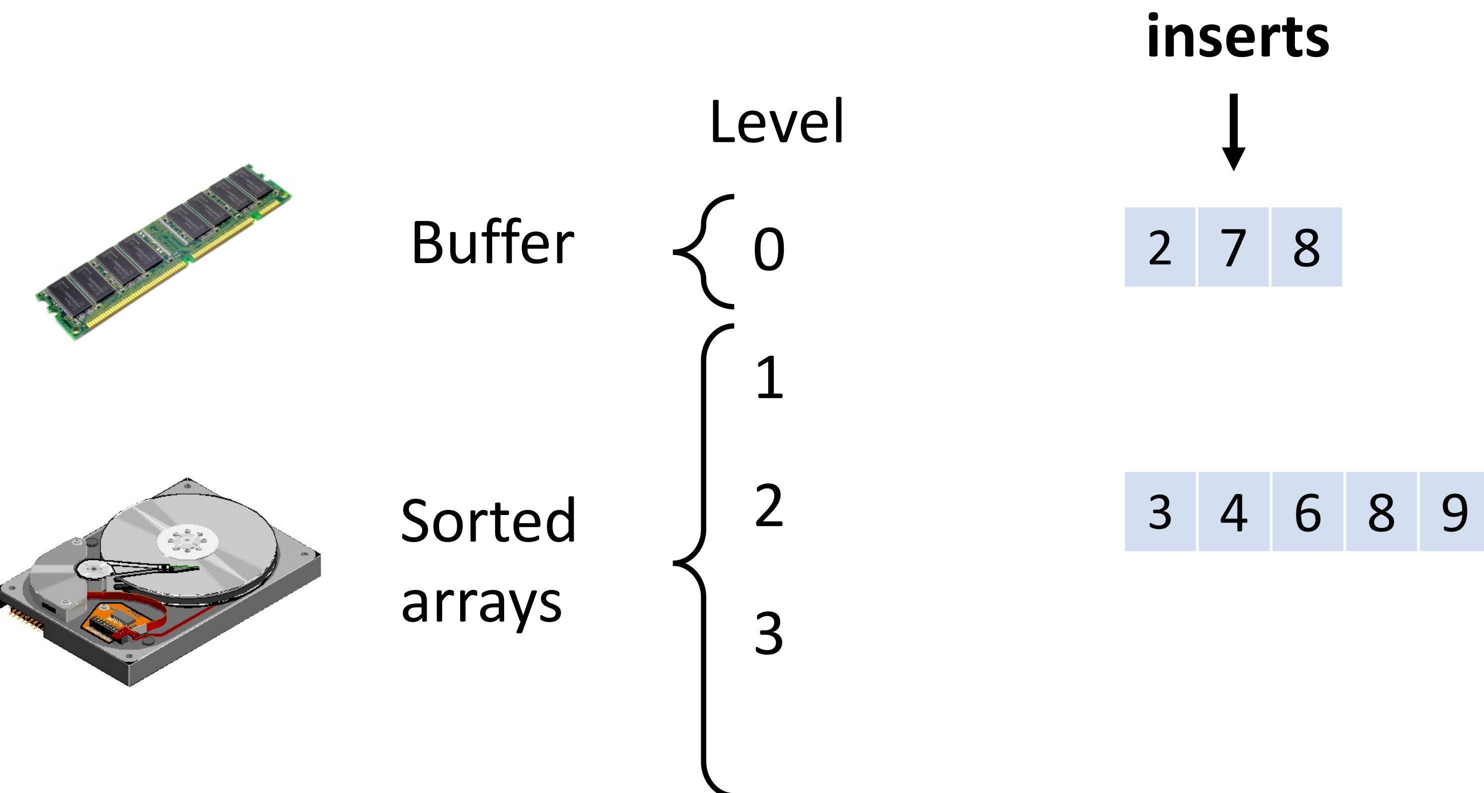
3	4	6	8	9

Sort-merge &
Eliminate duplicates &
Discard original arrays

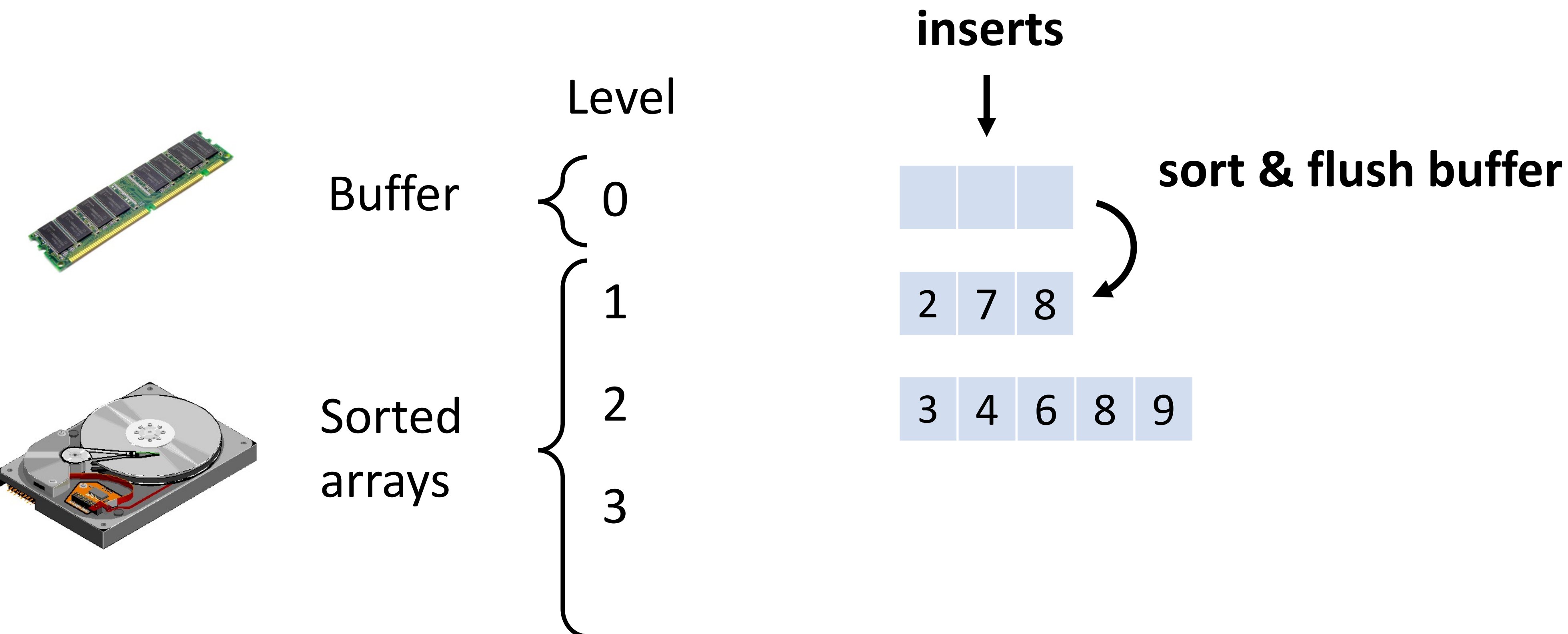
Basic LSM-tree – Example



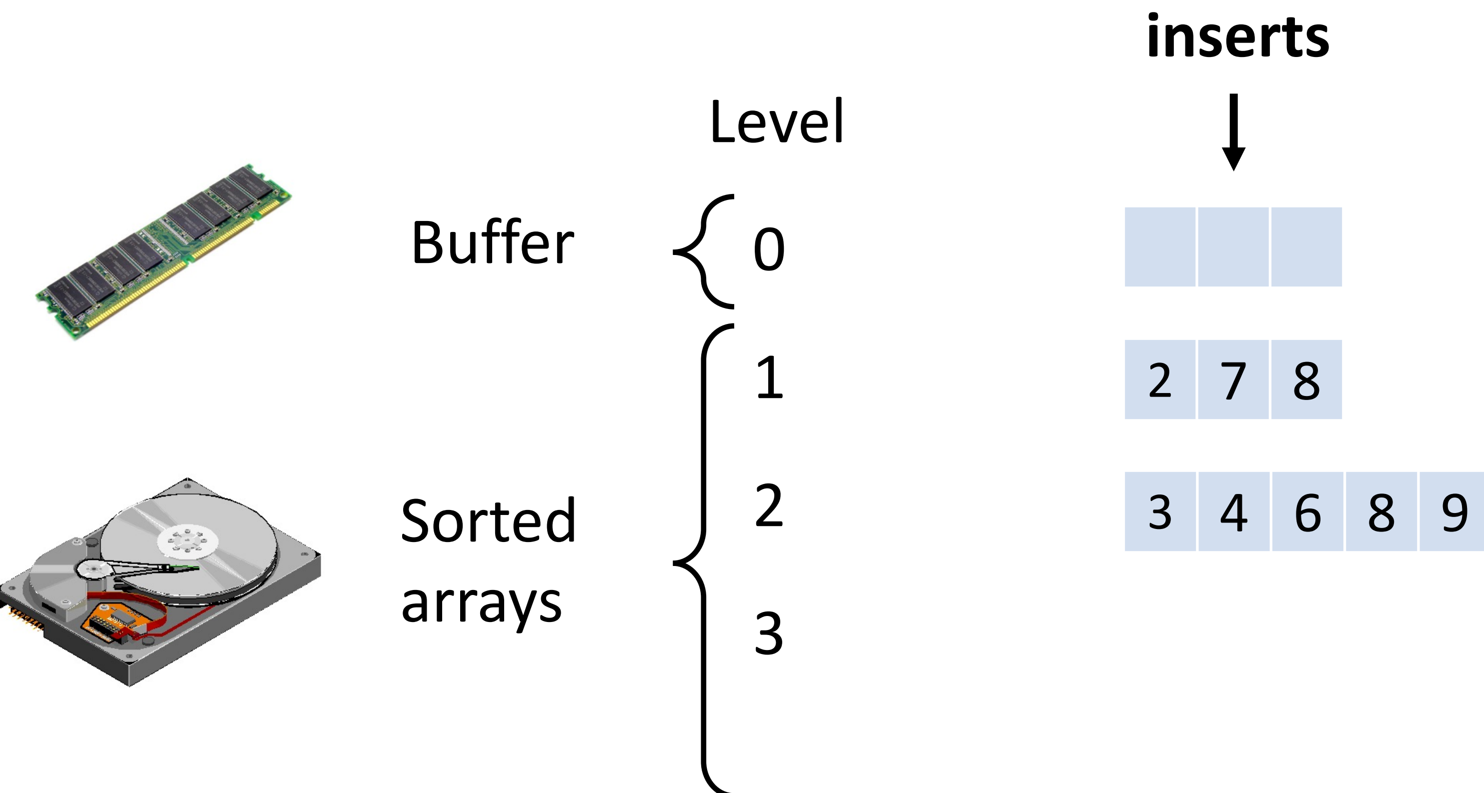
Basic LSM-tree – Example



Basic LSM-tree – Example

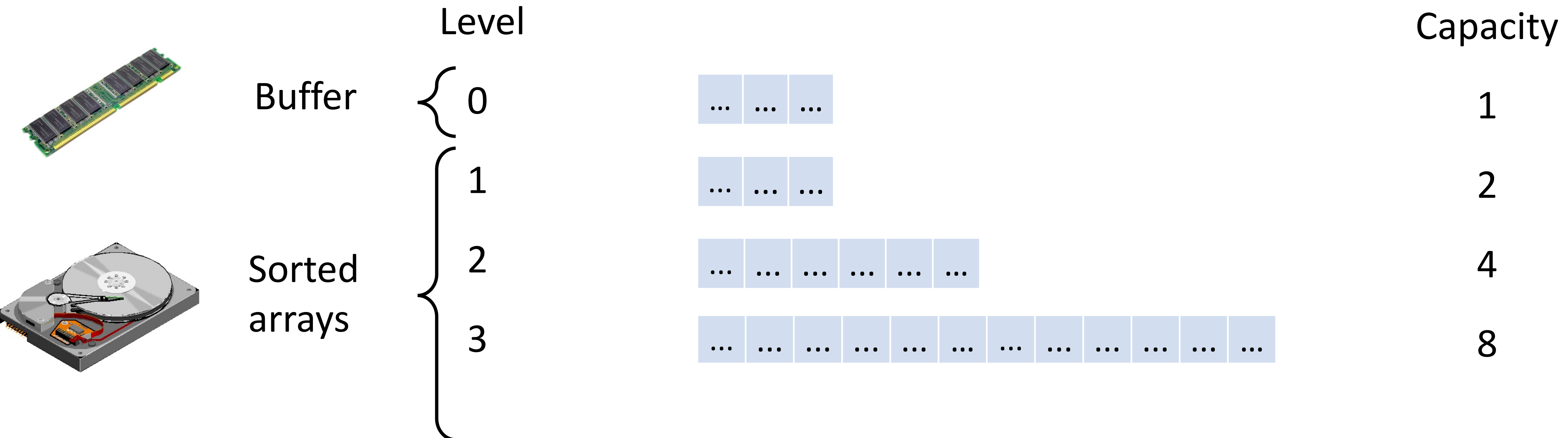


Basic LSM-tree – Example



Basic LSM-tree

Levels have exponentially increasing capacities.

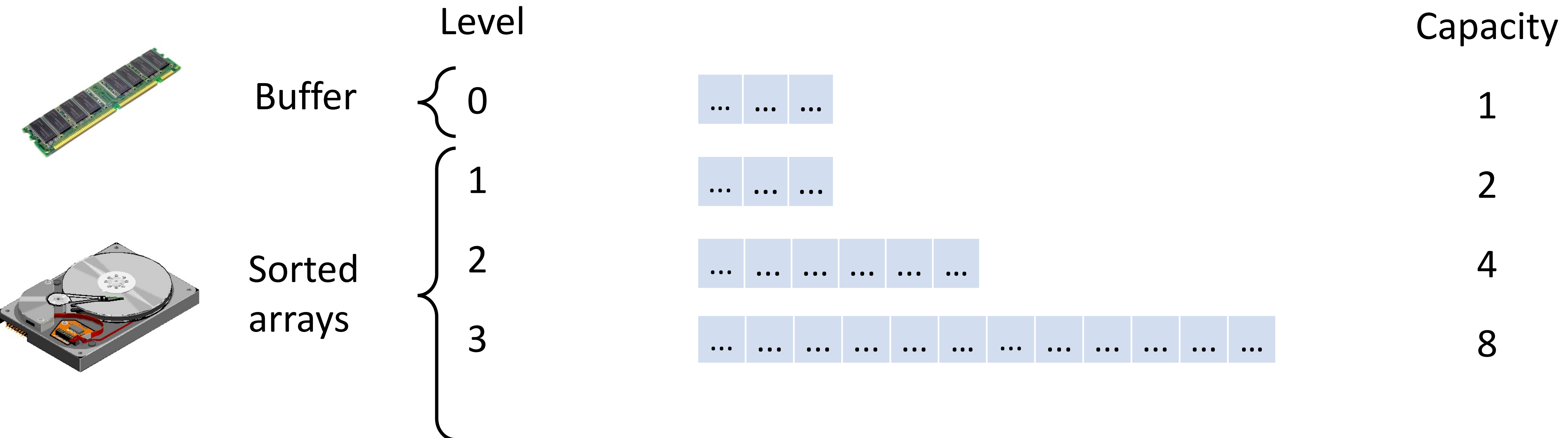


Basic LSM-tree – Lookup cost

Lookup method? Search youngest to oldest. $O(\log_2(N))$

How? Binary search. $O(\log_2(N))$

Lookup cost? $O(\log_2(N)^2)$

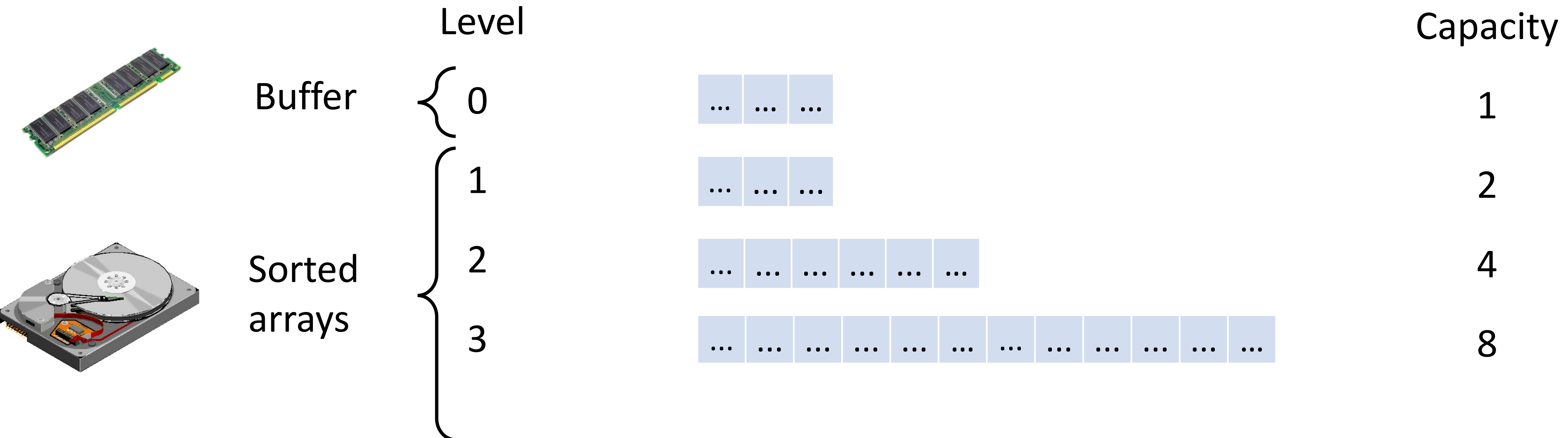


Basic LSM-tree – Insertion cost

How many times is each entry copied? $O(\log_2(N))$

What is the price of each copy? $O\left(\frac{1}{B}\right)$

Total insert cost? $O\left(\frac{1}{B} \cdot \log_2(N)\right)$



Results Catalogue

	Lookup cost	**Insertion cost**
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N)^2)$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue

Better insert cost and worst lookup cost compared with B-trees

	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N)^2)$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

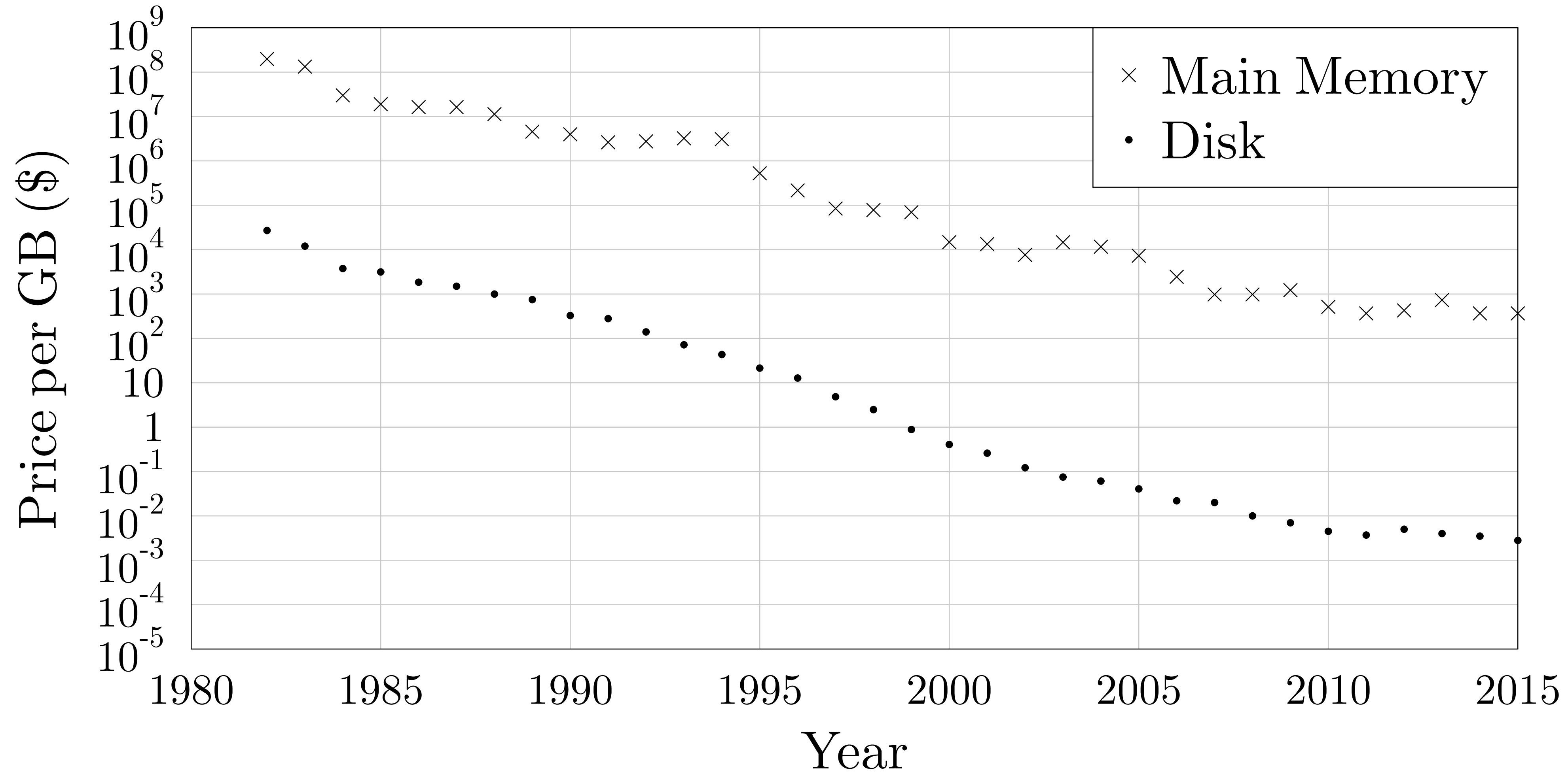
Results Catalogue

Better insert cost and worst lookup cost compared with B-trees

Can we improve the lookup cost?

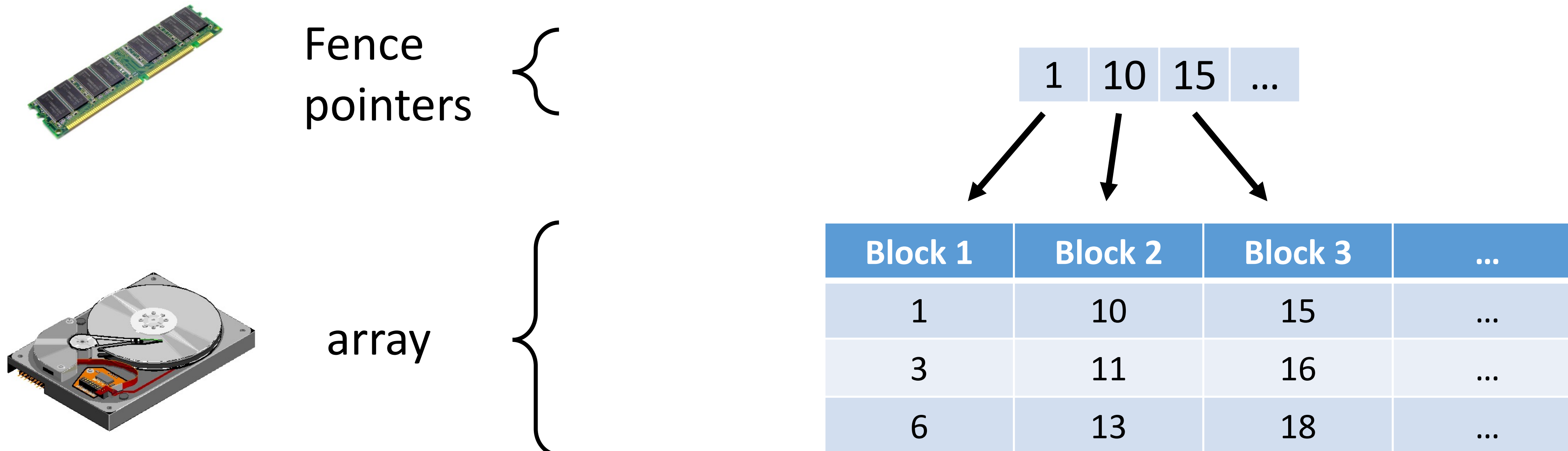
	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N)^2)$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

Declining Main Memory Cost



Declining Main Memory Cost

Store a fence pointer for every block in main memory



Block 1	Block 2	Block 3	...
1	10	15	...
3	11	16	...
6	13	18	...

Results Catalogue – with fence pointers

	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N)^2)$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue – with fence pointers

	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N))$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N)^2)$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue – with fence pointers

	Lookup cost	Insertion cost
Sorted array	$O(1)$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N)^2)$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue – with fence pointers

	Lookup cost	Insertion cost
Sorted array	$O(1)$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N)^2)$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue – with fence pointers

	Lookup cost	Insertion cost
Sorted array	O(1)	O(N/B)
Log	O(N)	O(1/B)
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N)^2)$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue – with fence pointers

	Lookup cost	Insertion cost
Sorted array	$O(1)$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N)^2)$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue – with fence pointers

	Lookup cost	Insertion cost
Sorted array	O(1)	O(N/B)
Log	O(N)	O(1/B)
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N))$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue – with fence pointers

Quick sanity check: suppose $N = 2^{32}$ and $B = 2^{10}$

	Lookup cost	Insertion cost
Sorted array	$O(1)$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N))$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue – with fence pointers

Quick sanity check: suppose $N = 2^{32}$ and $B = 2^{10}$

	Lookup cost	Insertion cost
Sorted array	$O(1)$	$O(2^{22})$
Log	$O(2^{32})$	$O(2^{-10})$
B-tree	$O(4)$	$O(4)$
Basic LSM-tree	$O(32)$	$O(2^{-10} \cdot 32)$
Leveled LSM-tree		
Tiered LSM-tree		

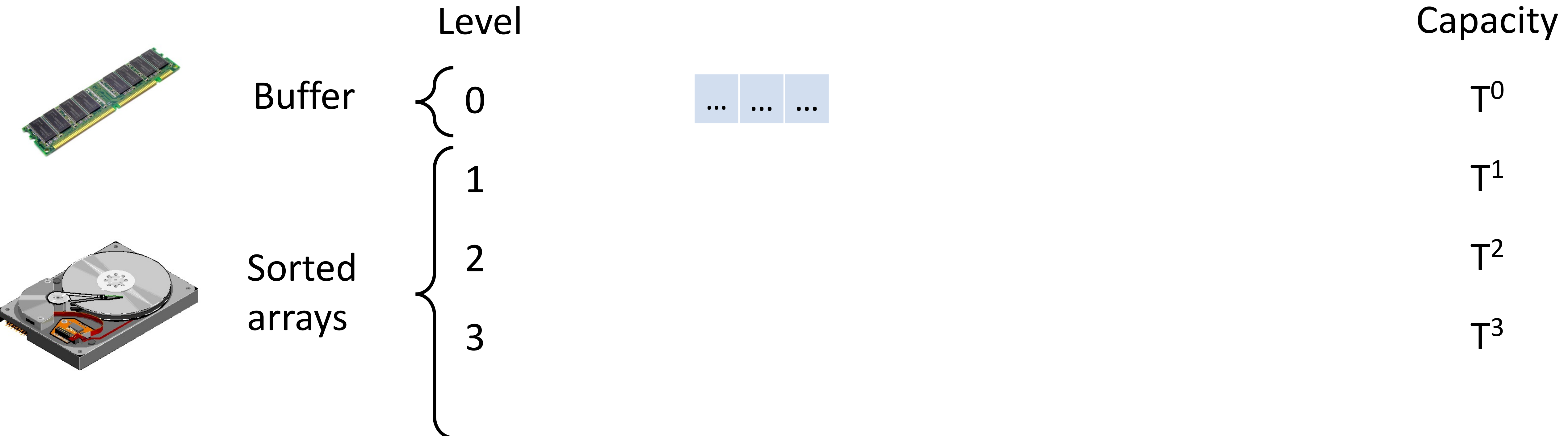
Leveled LSM-tree

Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

Increase size ratio T



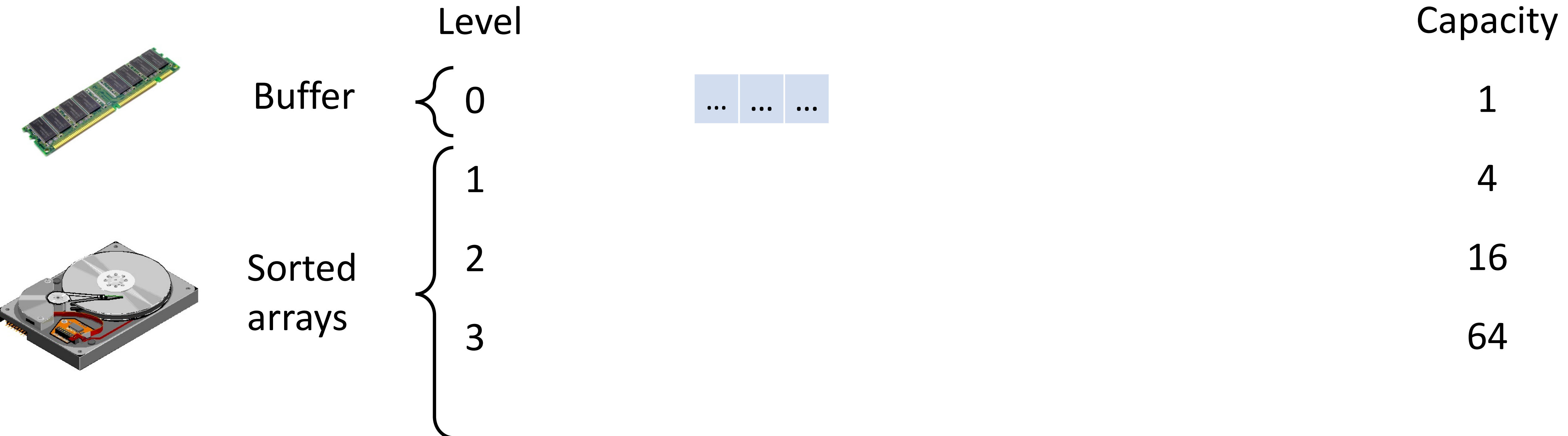
Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

Increase size ratio T

E.g. size ratio of 4



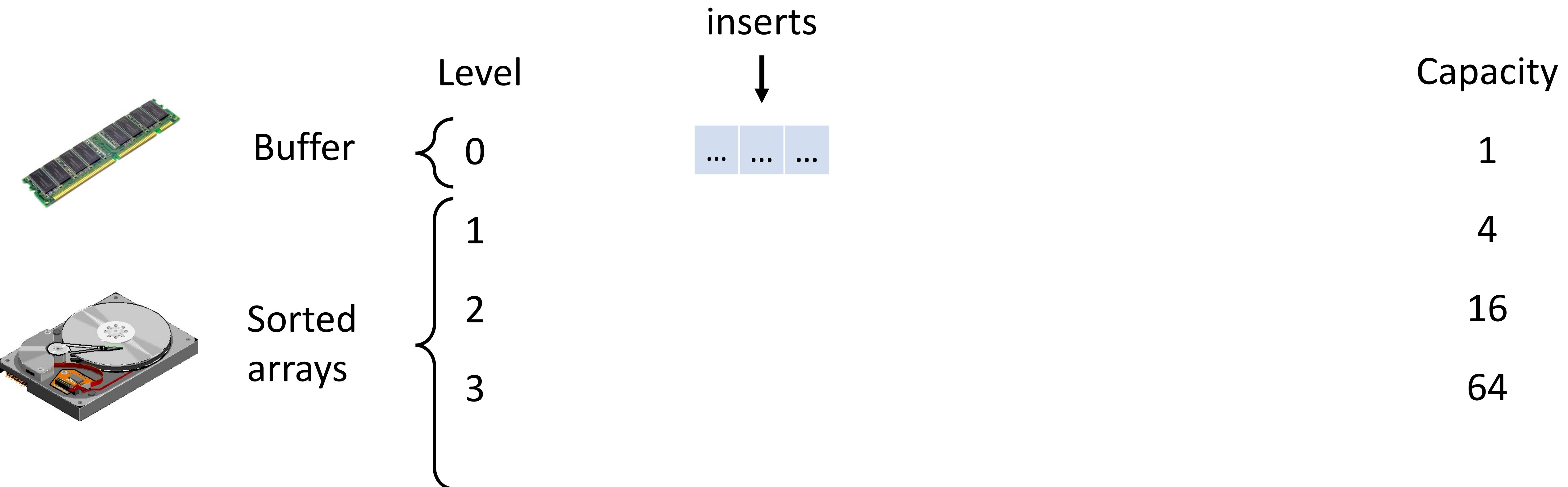
Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

Increase size ratio T

E.g. size ratio of 4



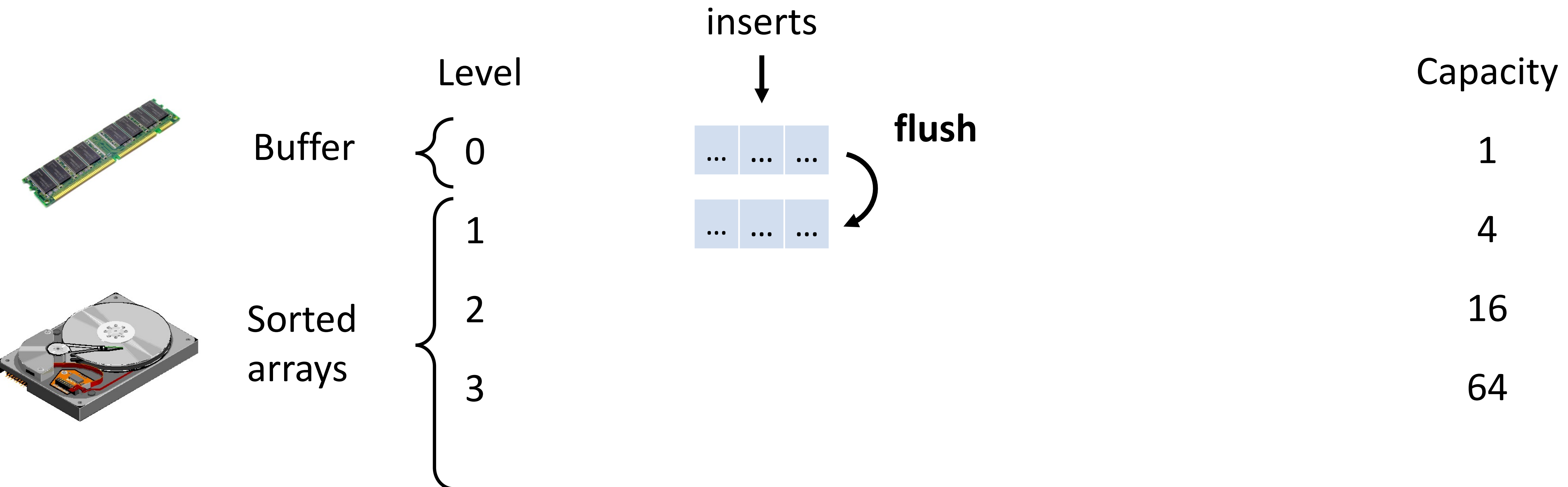
Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

Increase size ratio T

E.g. size ratio of 4



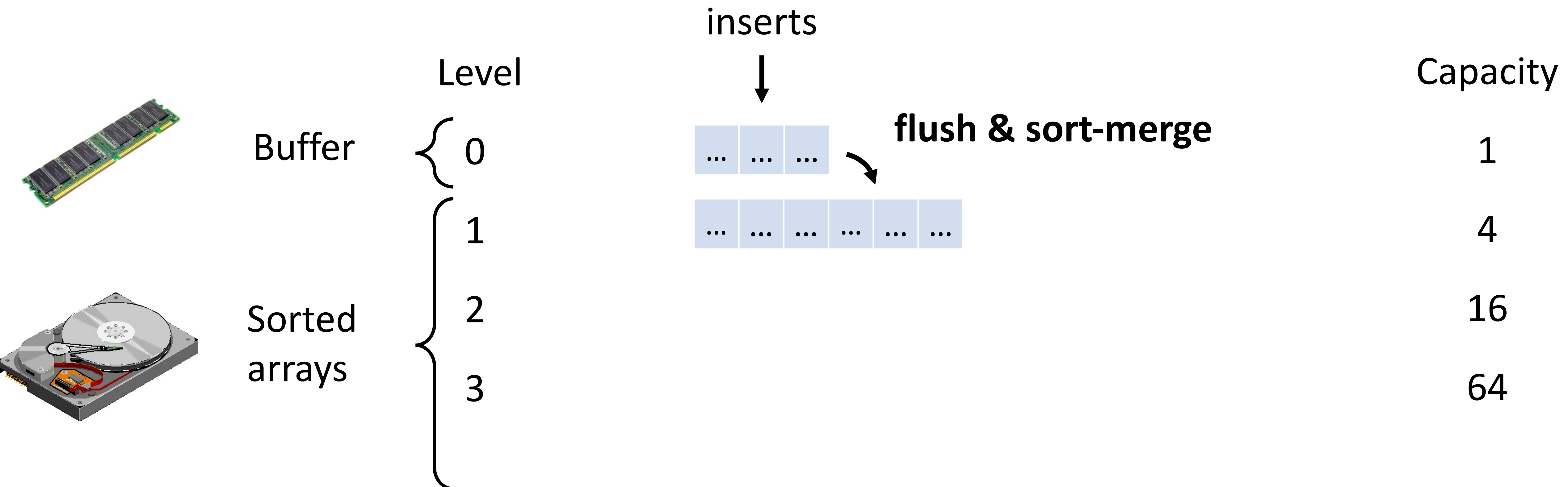
Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

Increase size ratio T

E.g. size ratio of 4



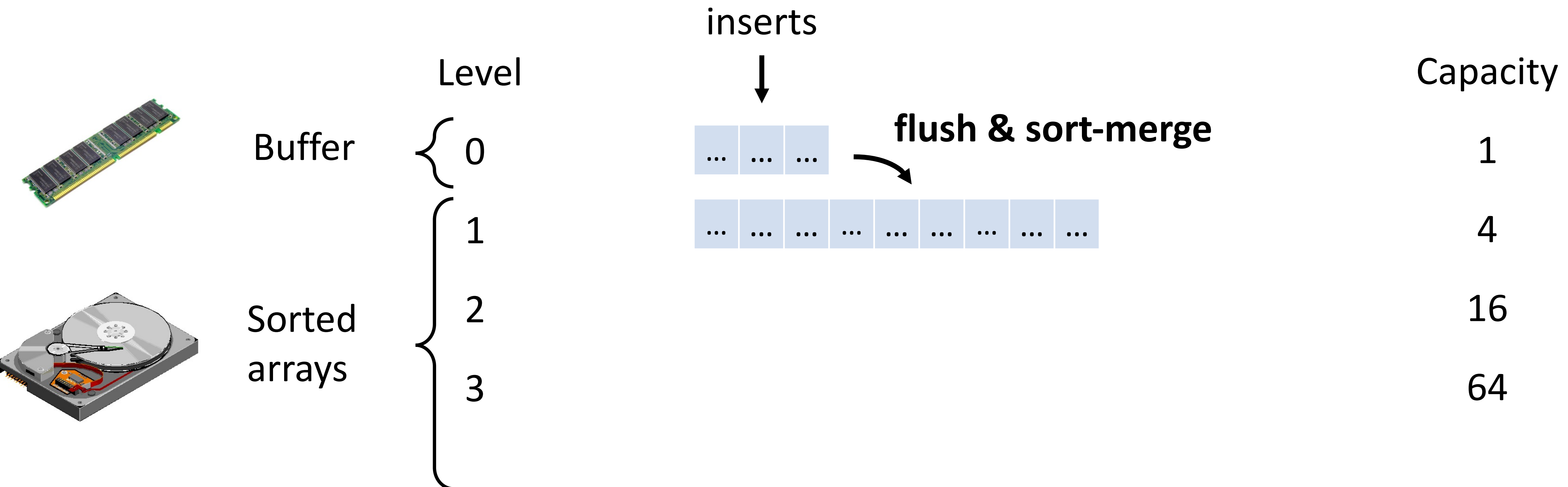
Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

Increase size ratio T

E.g. size ratio of 4



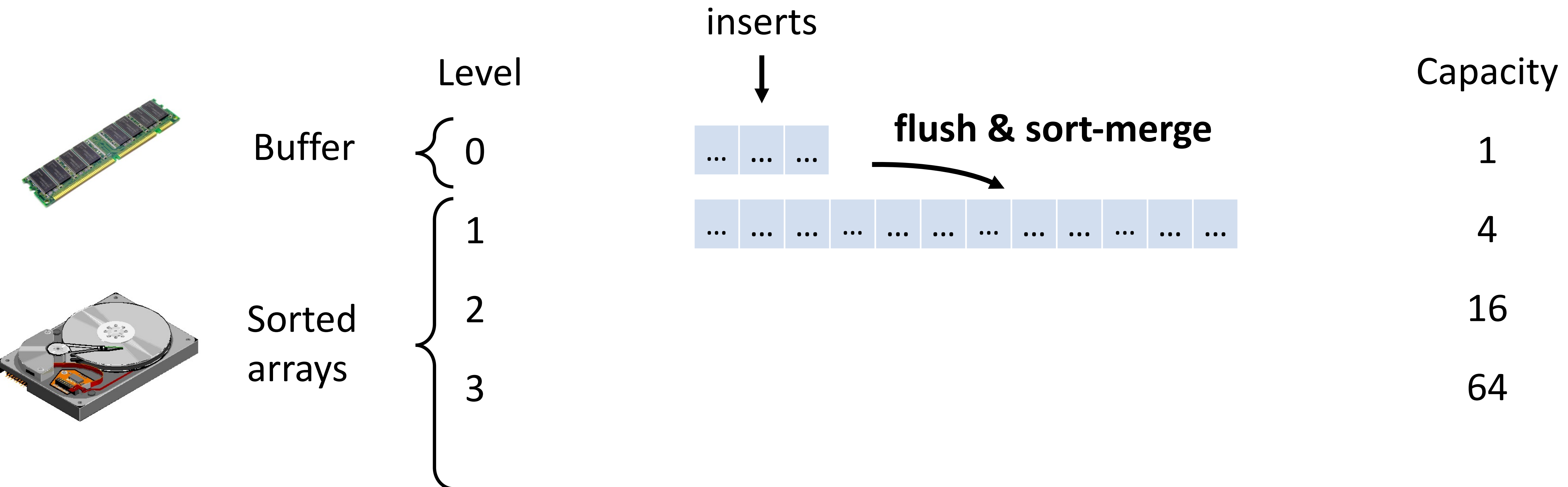
Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

Increase size ratio T

E.g. size ratio of 4



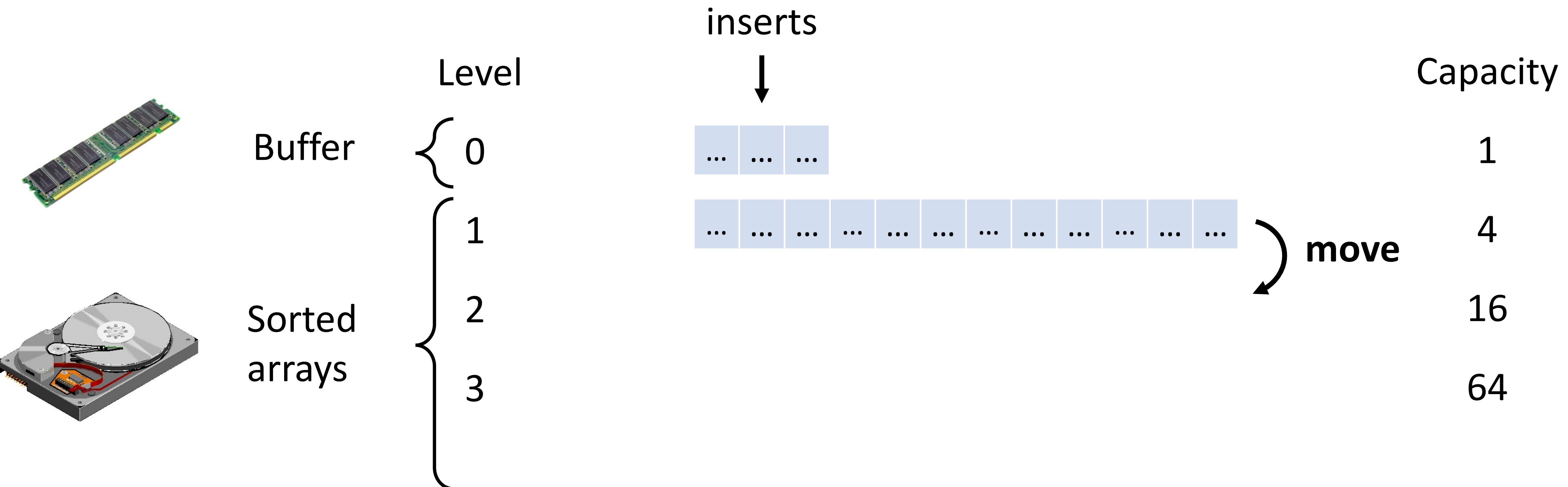
Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

Increase size ratio T

E.g. size ratio of 4

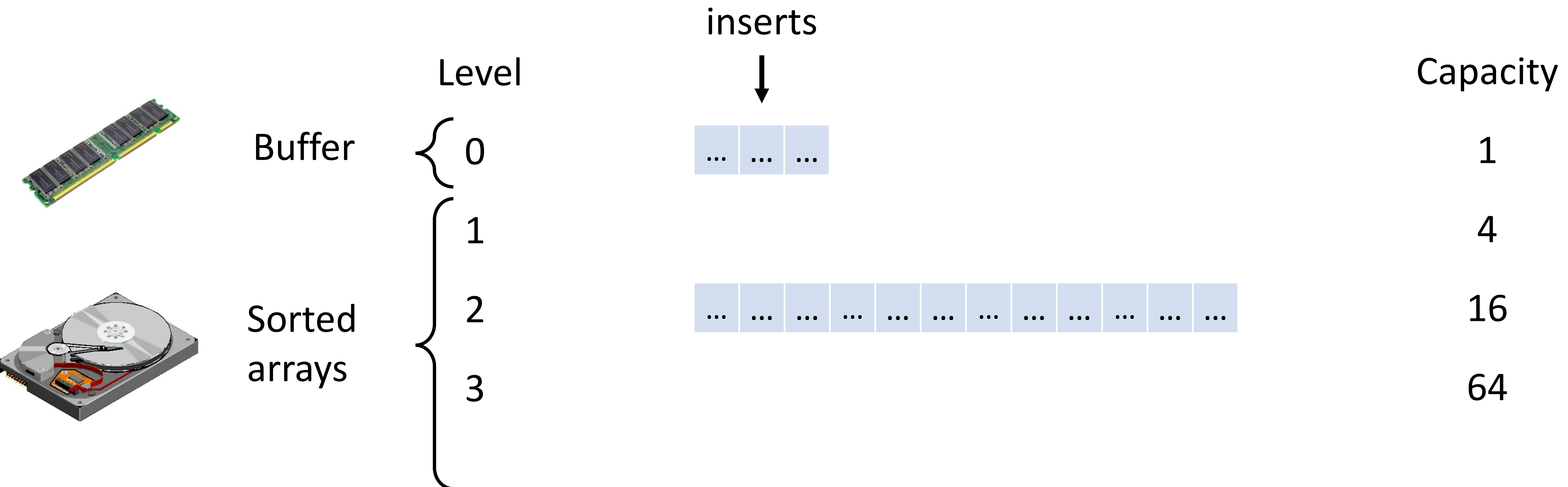


Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it? Increase size ratio T

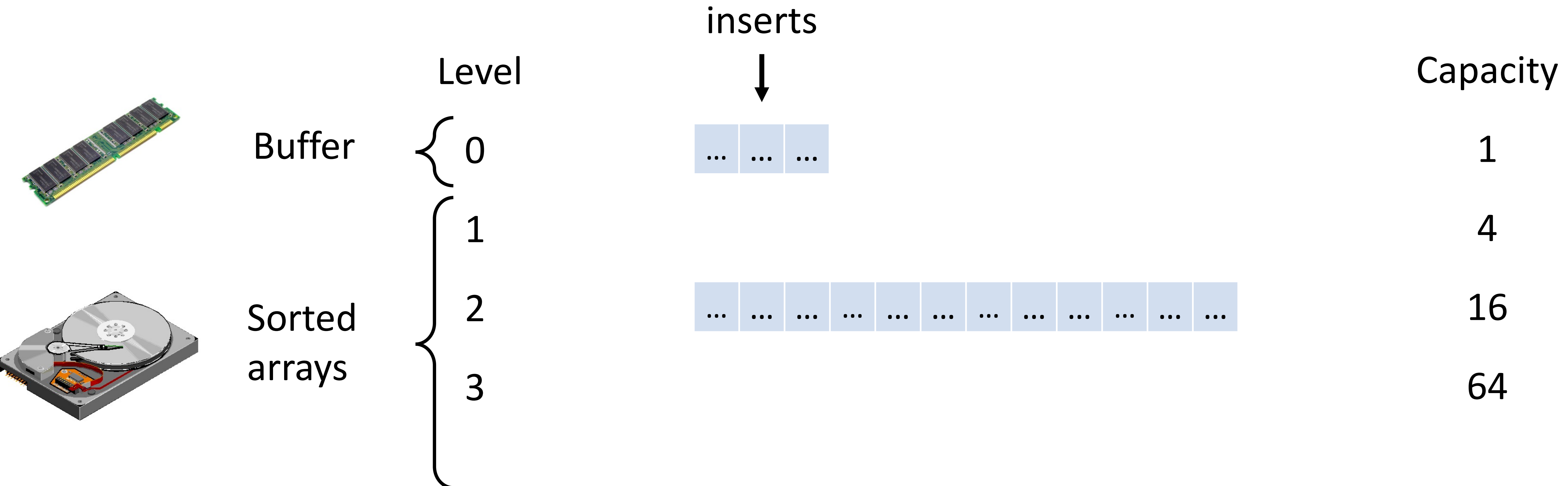
E.g. size ratio of 4



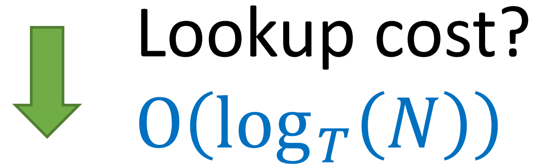
Leveled LSM-tree

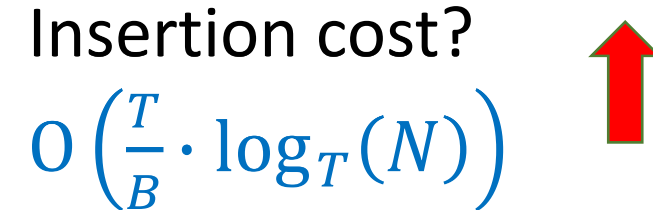
Lookup cost?
$O(\log_T(N))$

Insertion cost?
$O\left(\frac{T}{B} \cdot \log_T(N)\right)$



Leveled LSM-tree





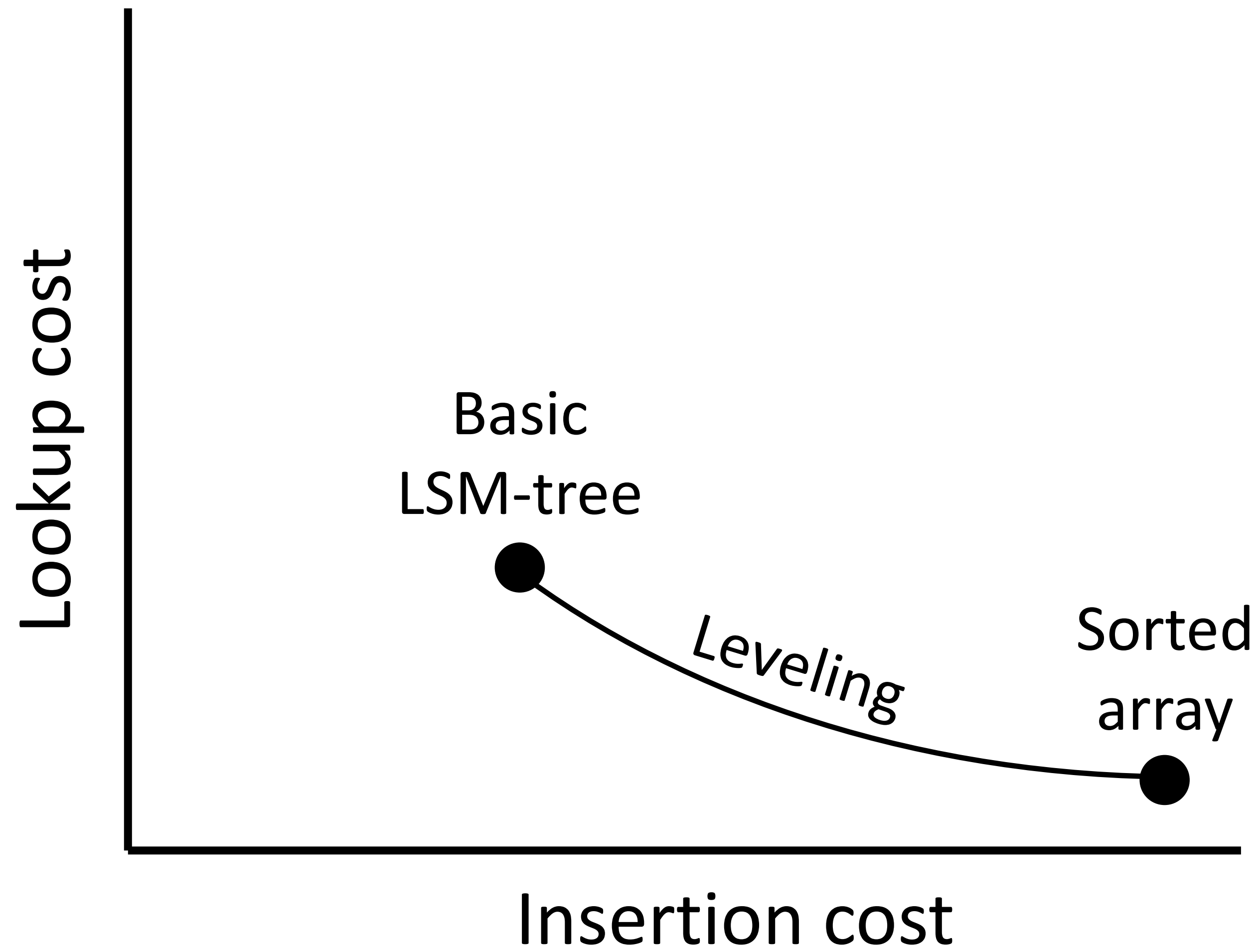
What happens as we increase the size ratio T?

What happens when size ratio T is set to be N?

Lookup cost becomes:
O(1)

Insert cost becomes:
O(N/B)

The LSM-tree becomes a sorted array!


Lookup cost
Basic
LSM-tree
Leveling
Sorted
array
Insertion cost

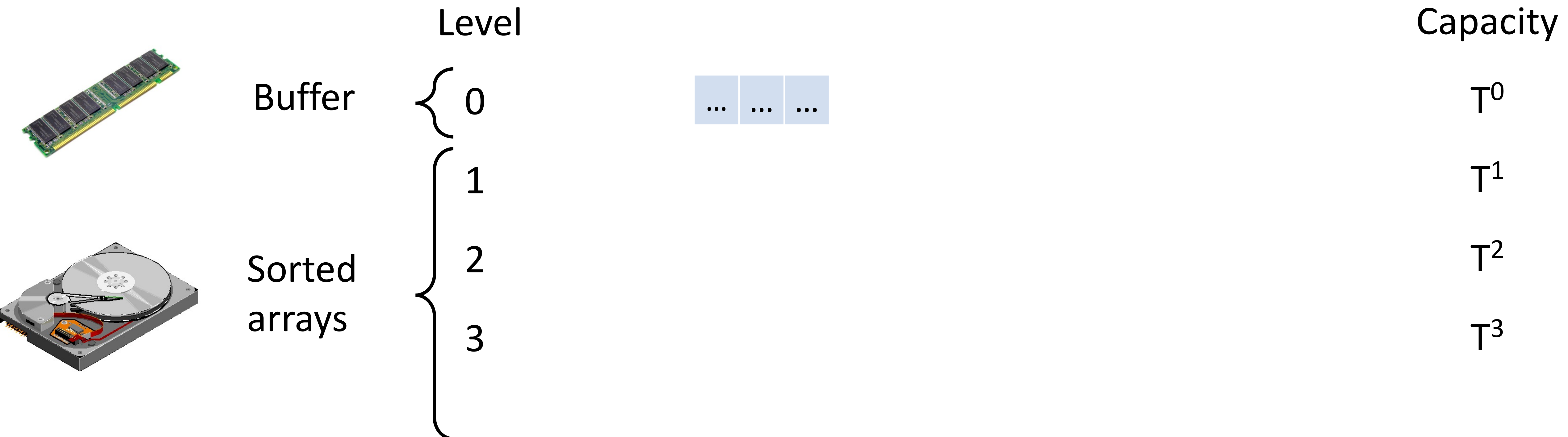
Results Catalogue – with fence pointers

	Lookup cost	Insertion cost
Sorted array	O(1)	O(N/B)
Log	O(N)	O(1/B)
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N))$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree	$O(\log_T(N))$	$O(T/B \cdot \log_T(N))$
Tiered LSM-tree		

Tiered LSM-tree

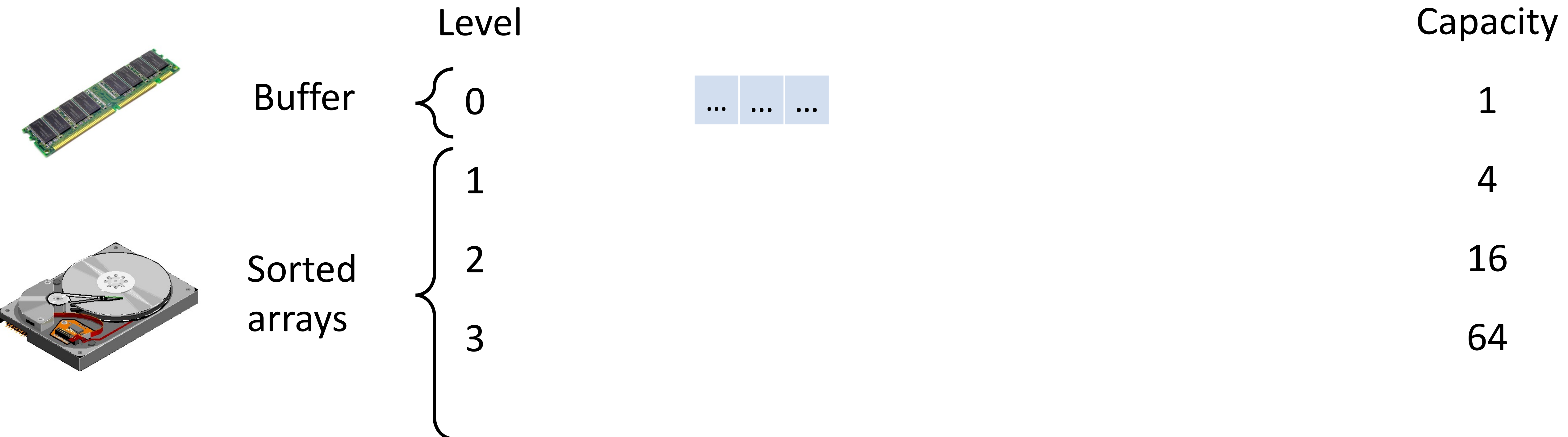
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.



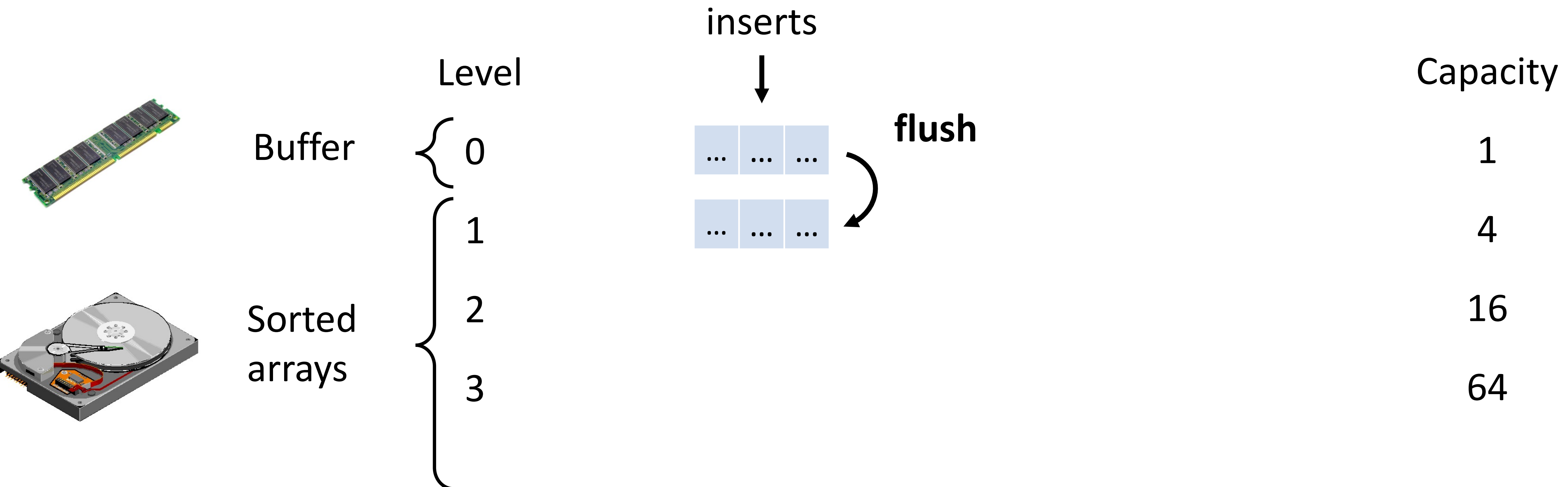
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4



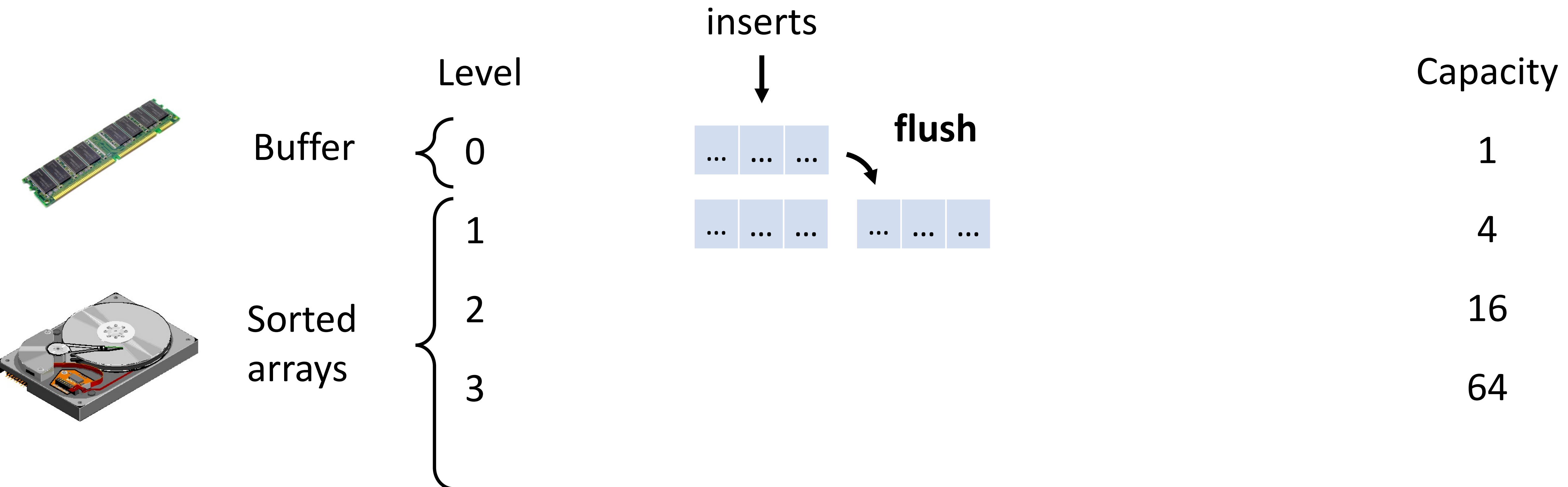
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4



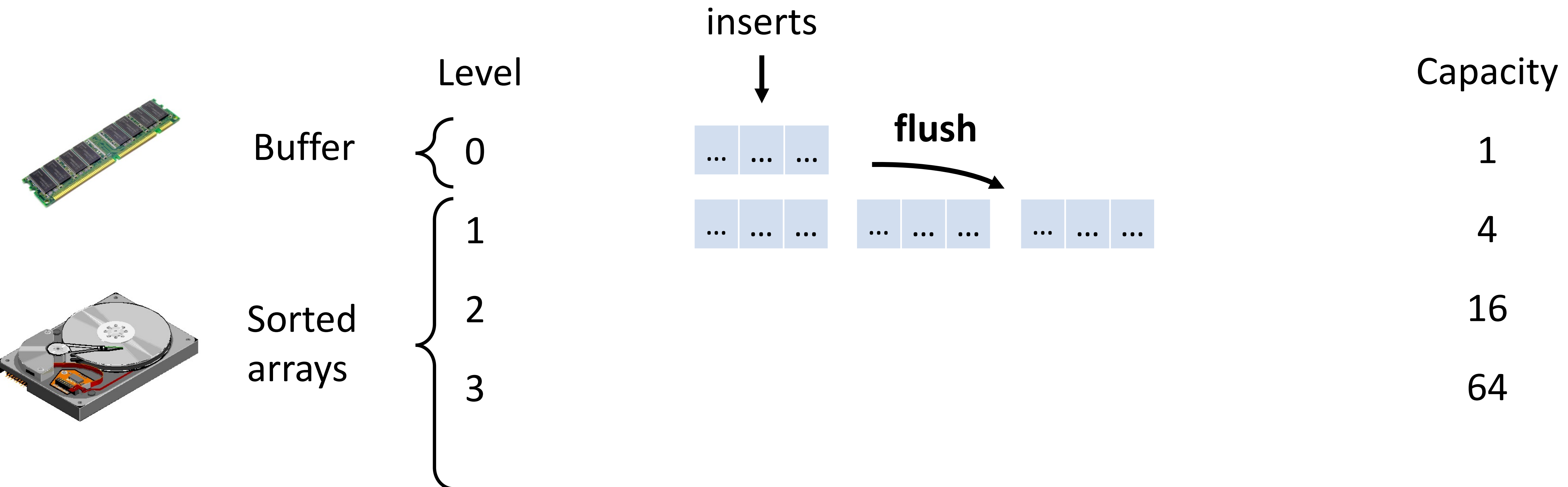
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4



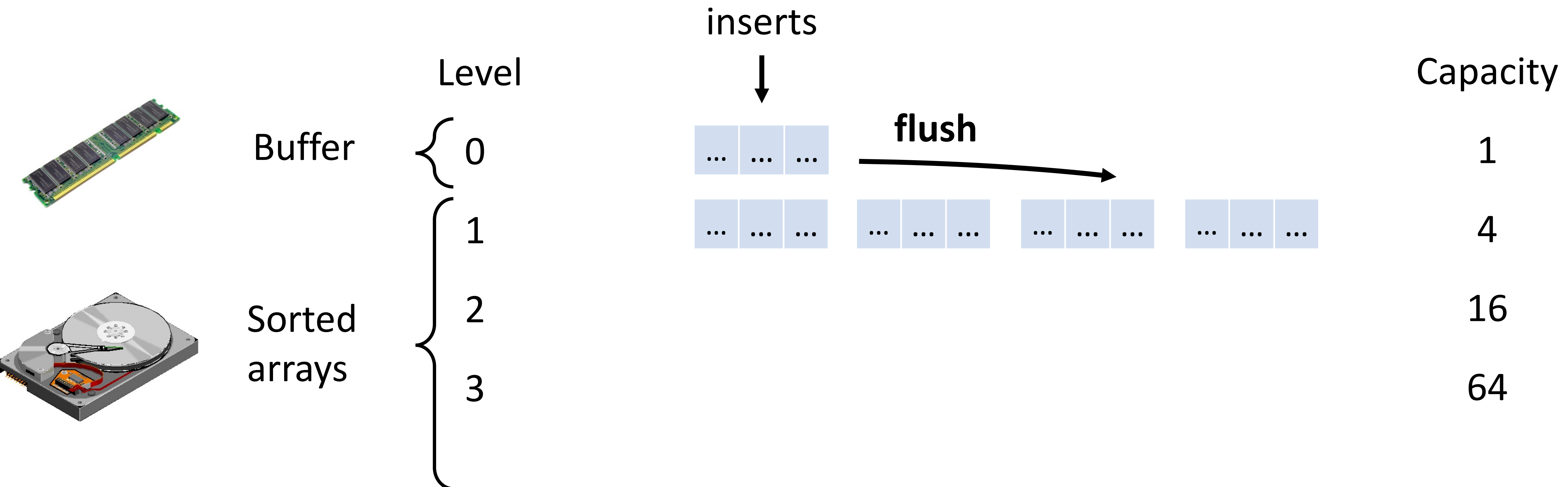
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4



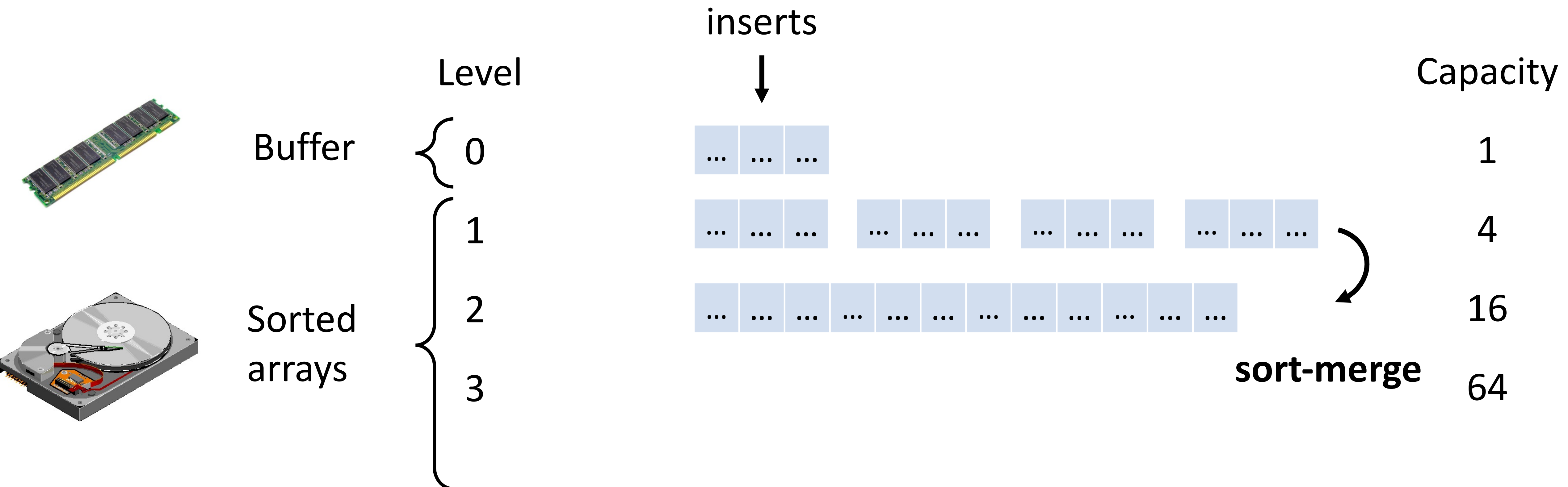
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4



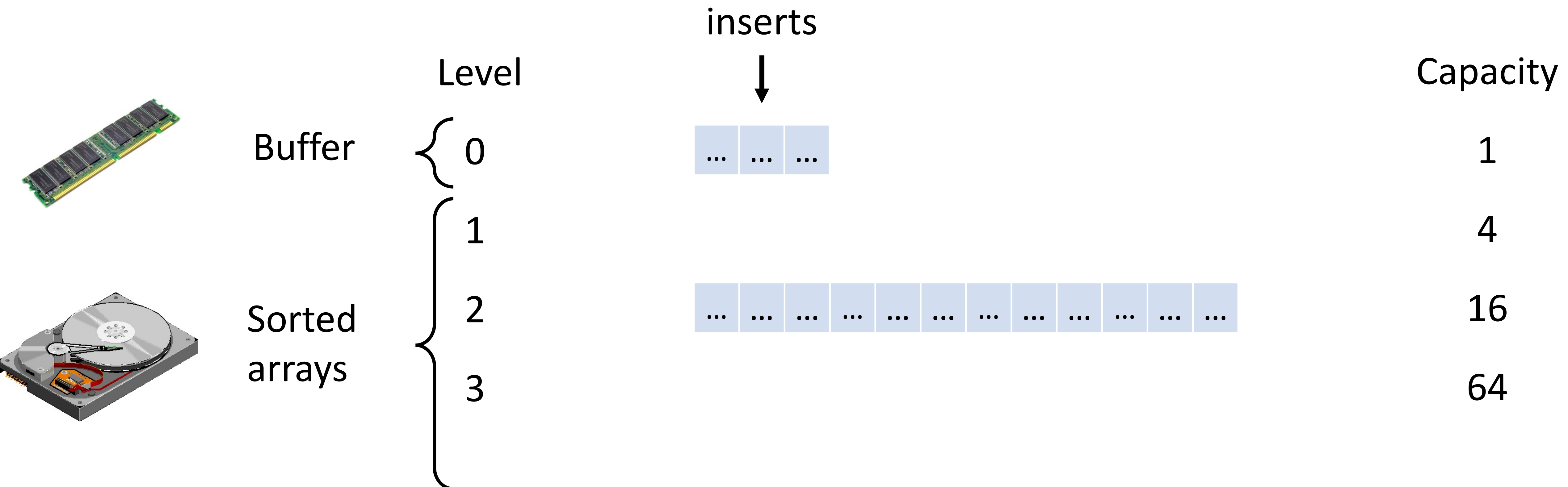
Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4



Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4



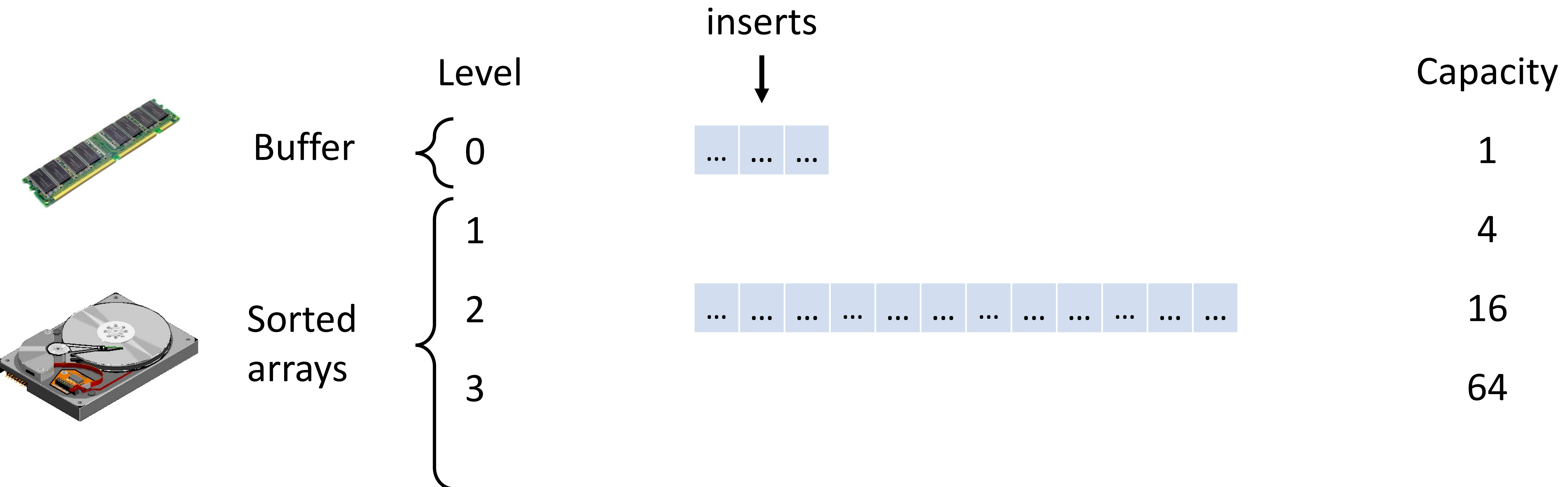
Tiered LSM-tree

Lookup cost?

$O(T \cdot \log_T(N))$

Insertion cost?

$O\left(\frac{1}{B} \cdot \log_T(N)\right)$



Tiered LSM-tree

Lookup cost?

$O(T \cdot \log_T(N))$

Insertion cost?

$O\left(\frac{1}{B} \cdot \log_T(N)\right)$

What happens as we increase the size ratio T?

What happens when size ratio T is set to be N?

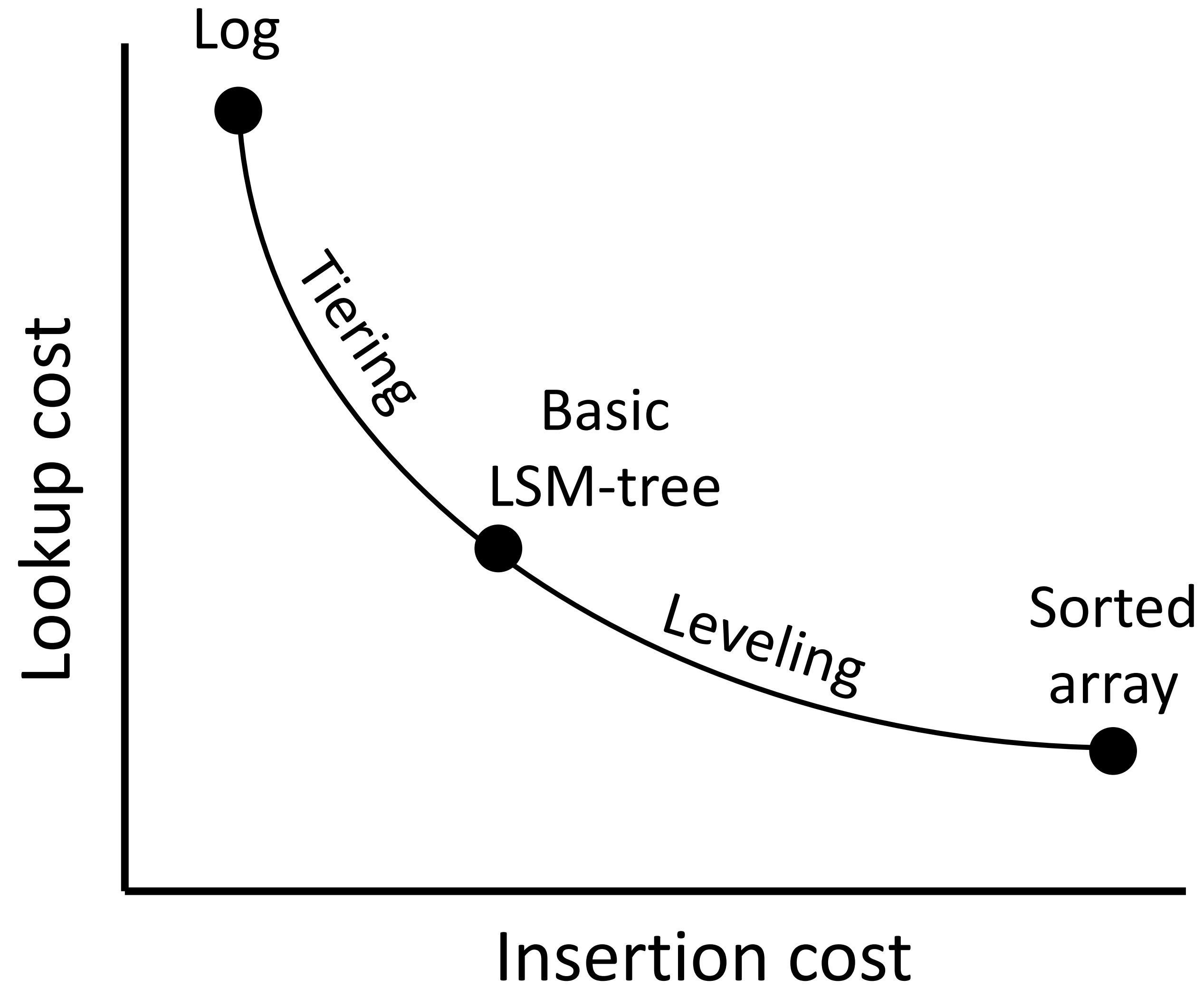
Lookup cost becomes:

O(N)

Insert cost becomes:

O(1/B)

The tiered LSM-tree becomes a log!


Log
Tiering
Basic
LSM-tree
Leveling
Sorted
array
Lookup cost
Insertion cost

Results Catalogue – with fence pointers

	Lookup cost	Insertion cost
Sorted array	O(1)	O(N/B)
Log	O(N)	O(1/B)
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N))$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree	$O(\log_T(N))$	$O(T/B \cdot \log_T(N))$
Tiered LSM-tree	$O(T \cdot \log_T(N))$	$O(1/B \cdot \log_T(N))$

Results Catalogue – with fence pointers

Quick sanity check: suppose $N = 2^{32}$ and $B = 2^{10}$ and $T = 2^{2}$

	Lookup cost	Insertion cost
Sorted array	$O(1)$	$O(N/B)$
Log	$O(N)$	$O(1/B)$
B-tree	$O(\log_B(N))$	$O(\log_B(N))$
Basic LSM-tree	$O(\log_2(N))$	$O(1/B \cdot \log_2(N))$
Leveled LSM-tree	$O(\log_T(N))$	$O(T/B \cdot \log_T(N))$
Tiered LSM-tree	$O(T \cdot \log_T(N))$	$O(1/B \cdot \log_T(N))$

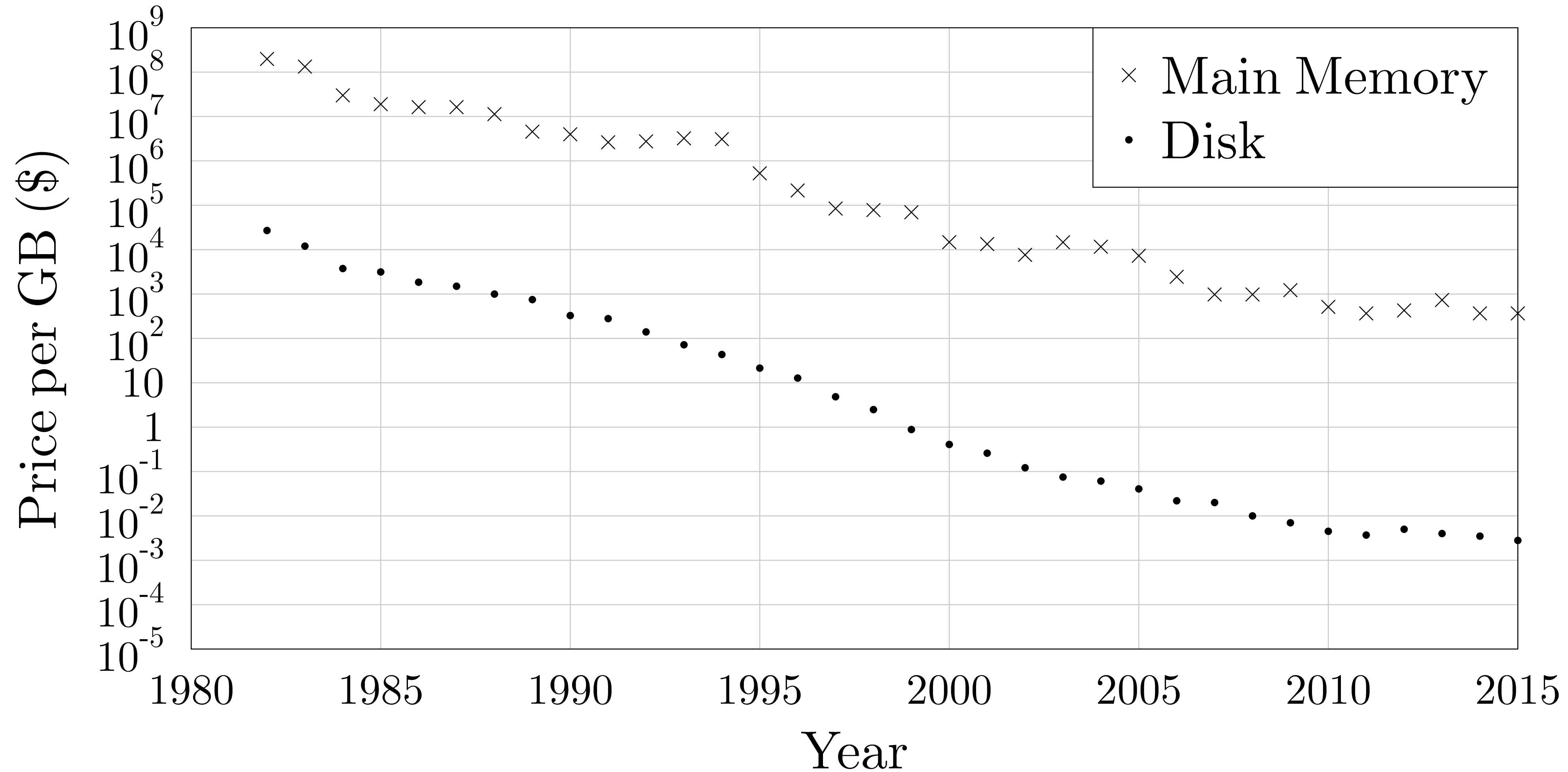
Results Catalogue – with fence pointers

Quick sanity check: suppose $N = 2^{32}$
and $B = 2^{10}$
and $T = 2^{2}$

	Lookup cost	Insertion cost
Sorted array	2^{0}=1	2^{22}=4K
Log	2^{32}=4M	2^{-10}=0.001
B-tree	2^{2}=4	2^{2}=4
Basic LSM-tree	2^{5}=32	2^{-5}=0.031
Leveled LSM-tree	2^{4}=16	2^{-4}=0.063
Tiered LSM-tree	2^{6}=64	2^{-6}=0.016

Bloom filters

Declining Main Memory Cost



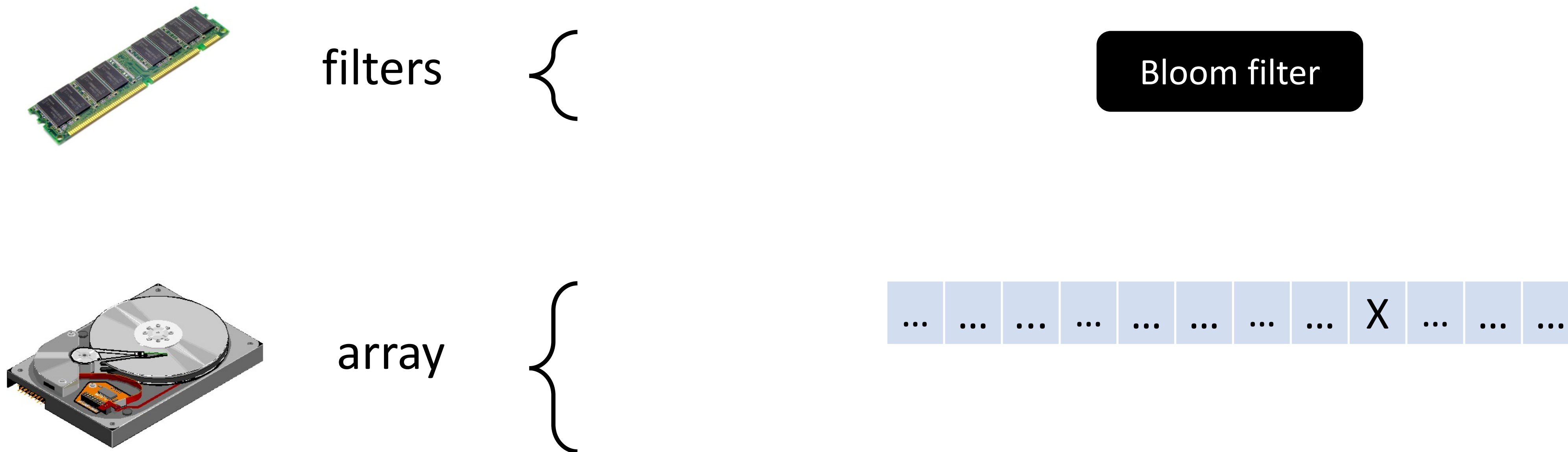
Bloom Filters

Answers set-membership queries

Smaller than array, and stored in main memory

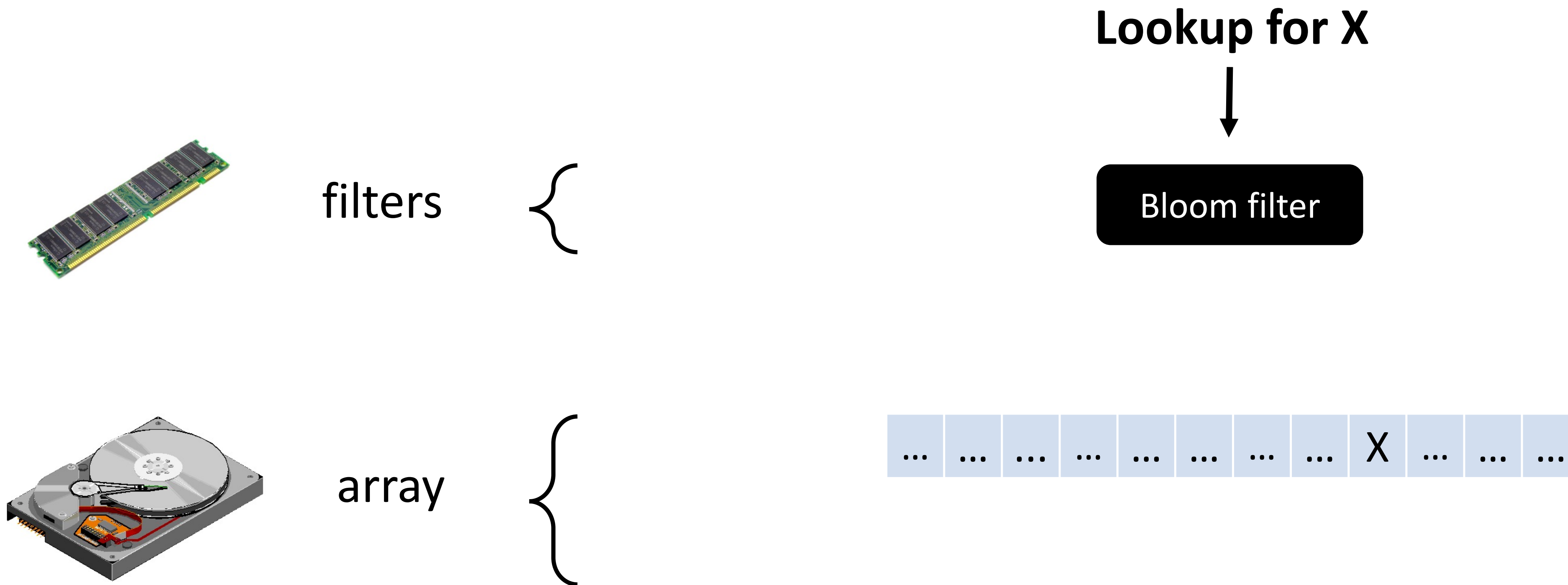
Purpose: avoid accessing disk if entry is not in array

Subtlety: may return false positives.



Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.

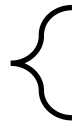


Bloom Filters

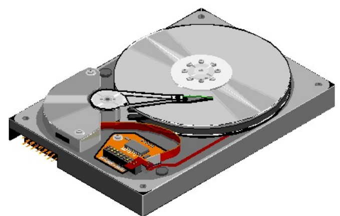
Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.

Lookup for X

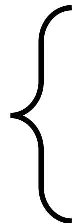
filters



Bloom filter

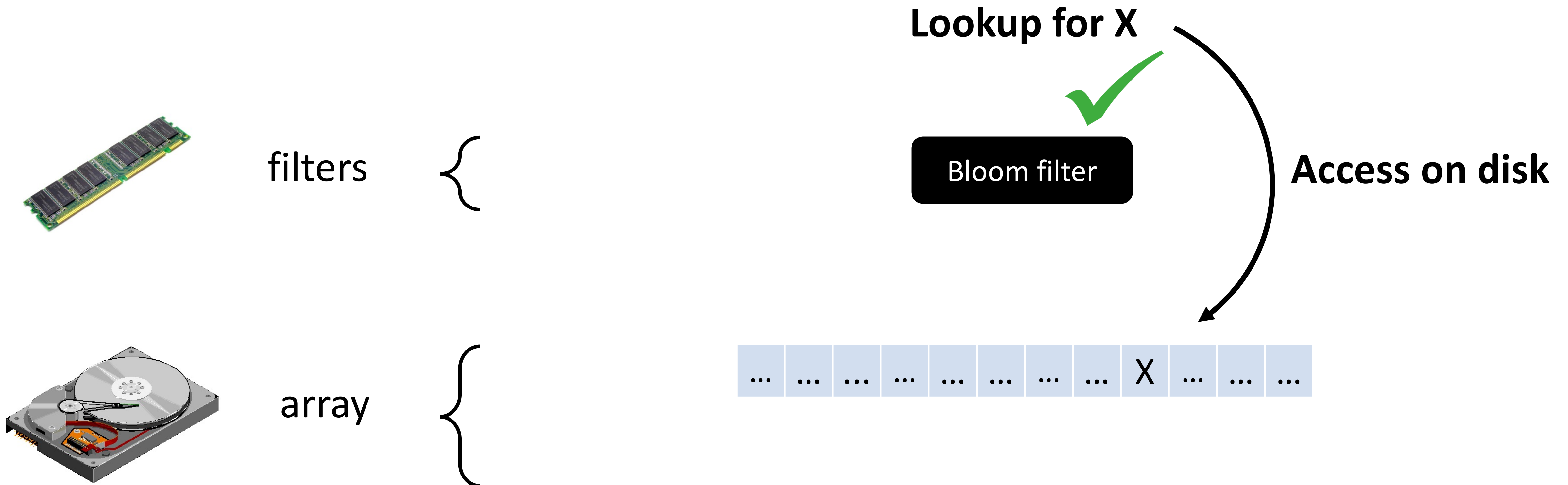


array



Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.



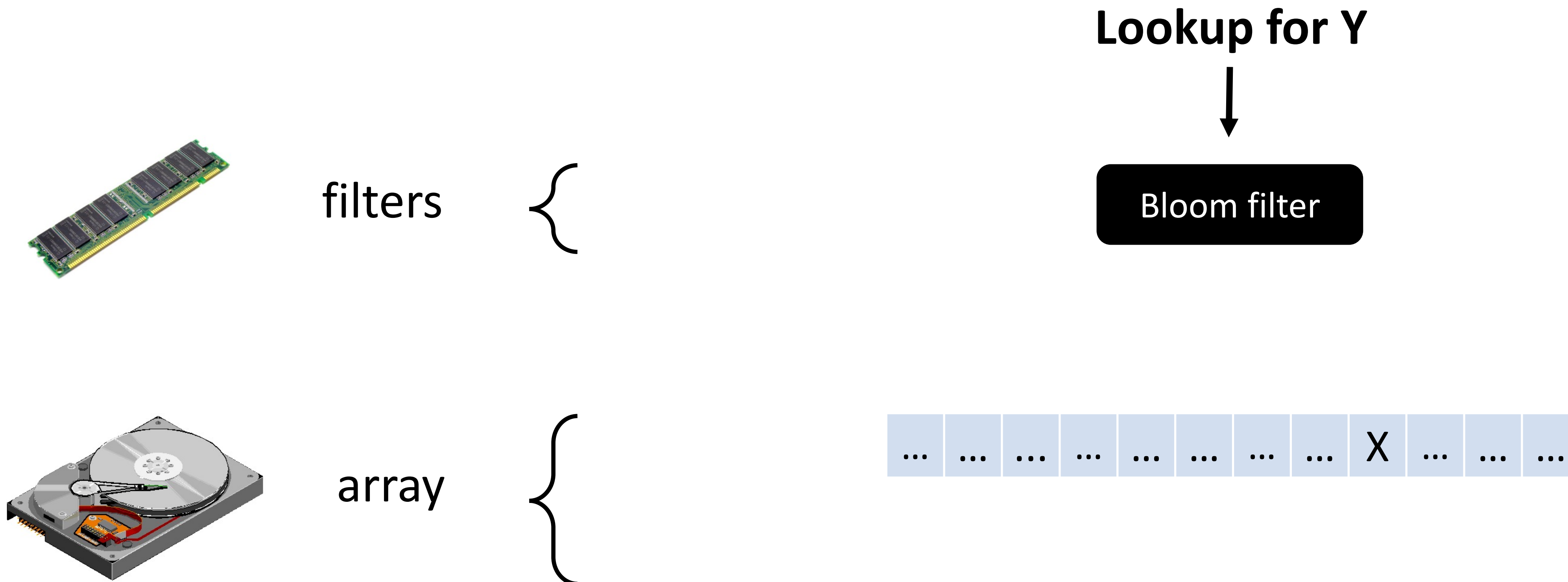
Bloom Filters

Answers set-membership queries

Smaller than array, and stored in main memory

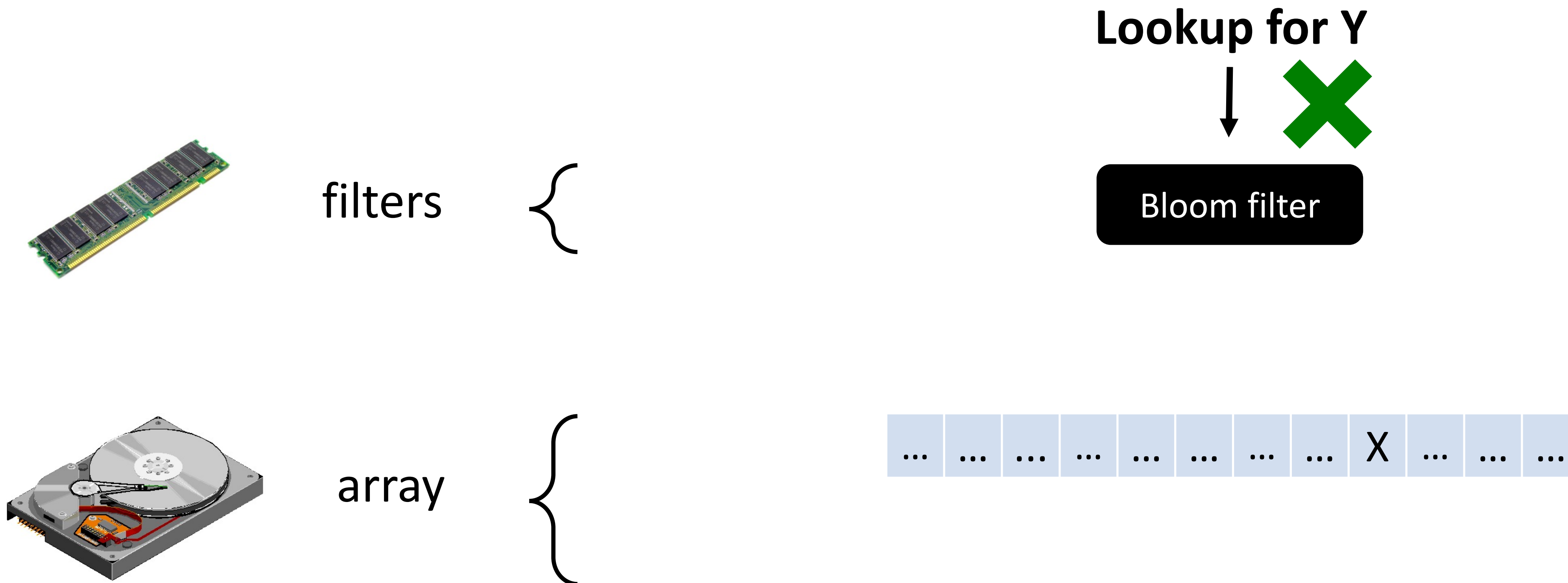
Purpose: avoid accessing disk if entry is not in array

Subtlety: may return false positives.



Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.



Bloom Filters

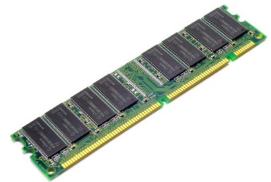
Answers set-membership queries

Smaller than array, and stored in main memory

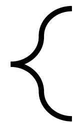
Purpose: avoid accessing disk if entry is not in array

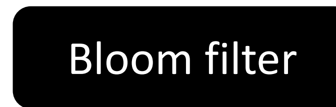
Subtlety: may return false positives.

Lookup for Y

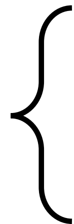


filters





array



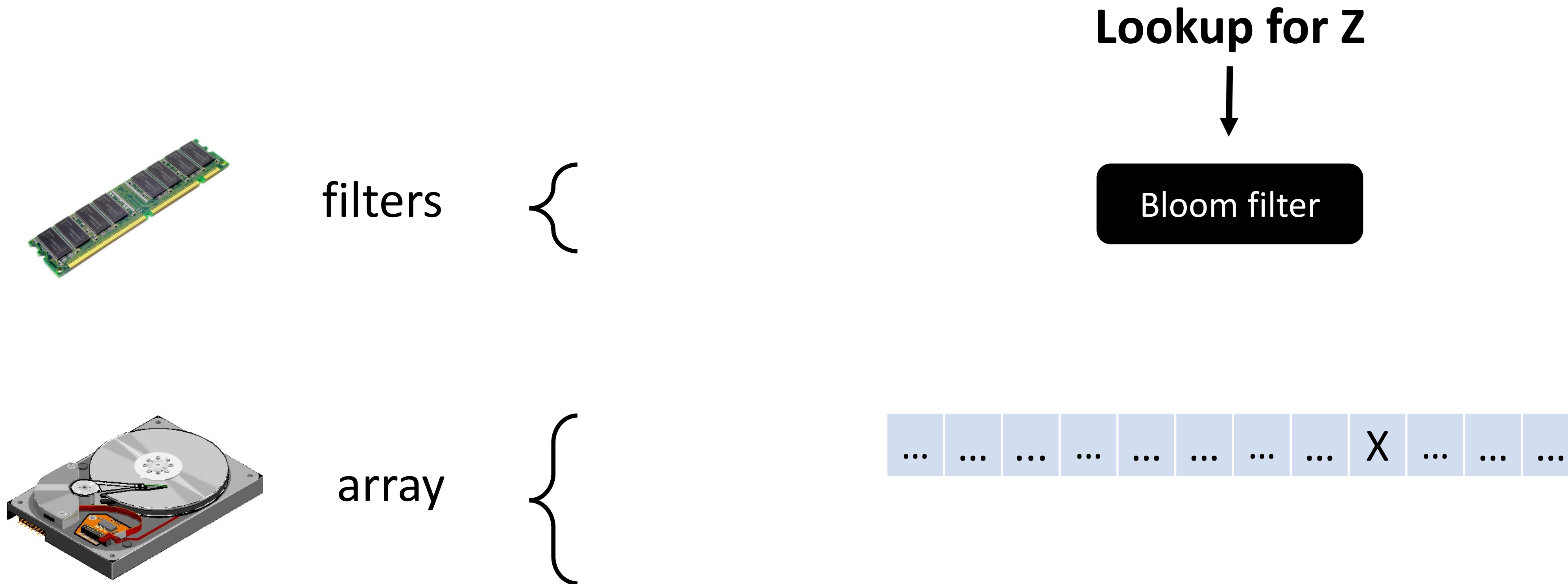
Bloom Filters

Answers set-membership queries

Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

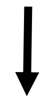
Subtlety: may return false positives.



Bloom Filters

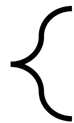
Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.

Lookup for Z

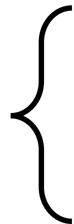


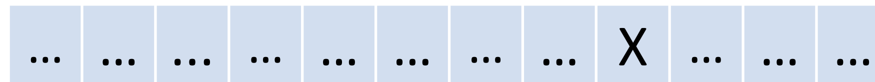
Bloom filter

filters



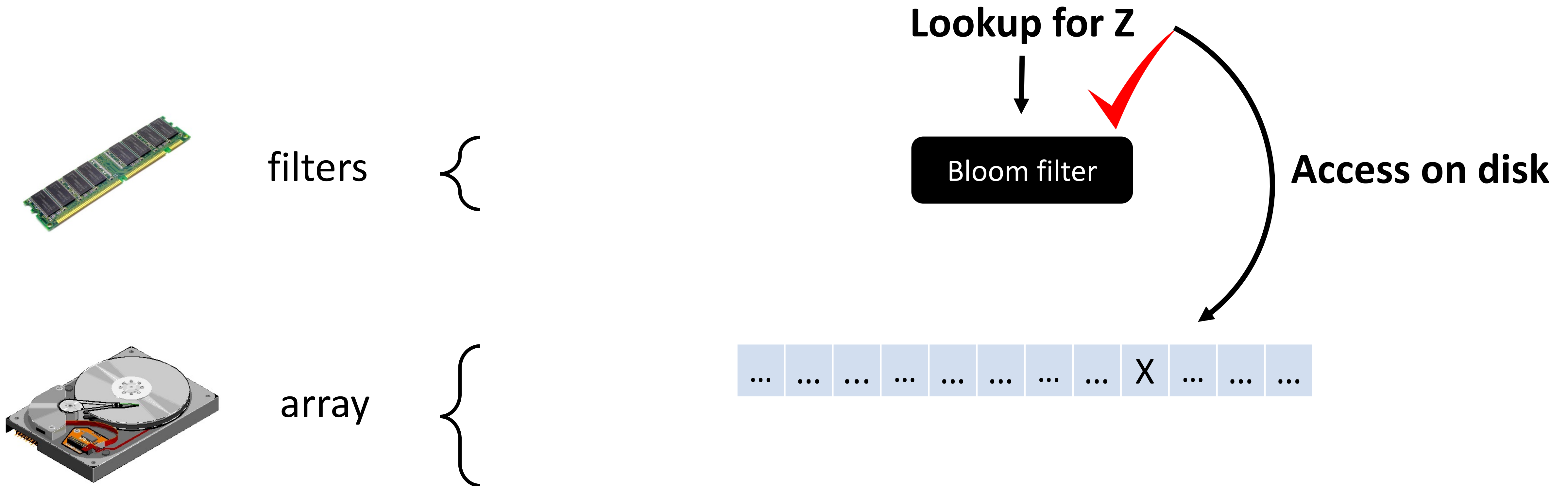
array





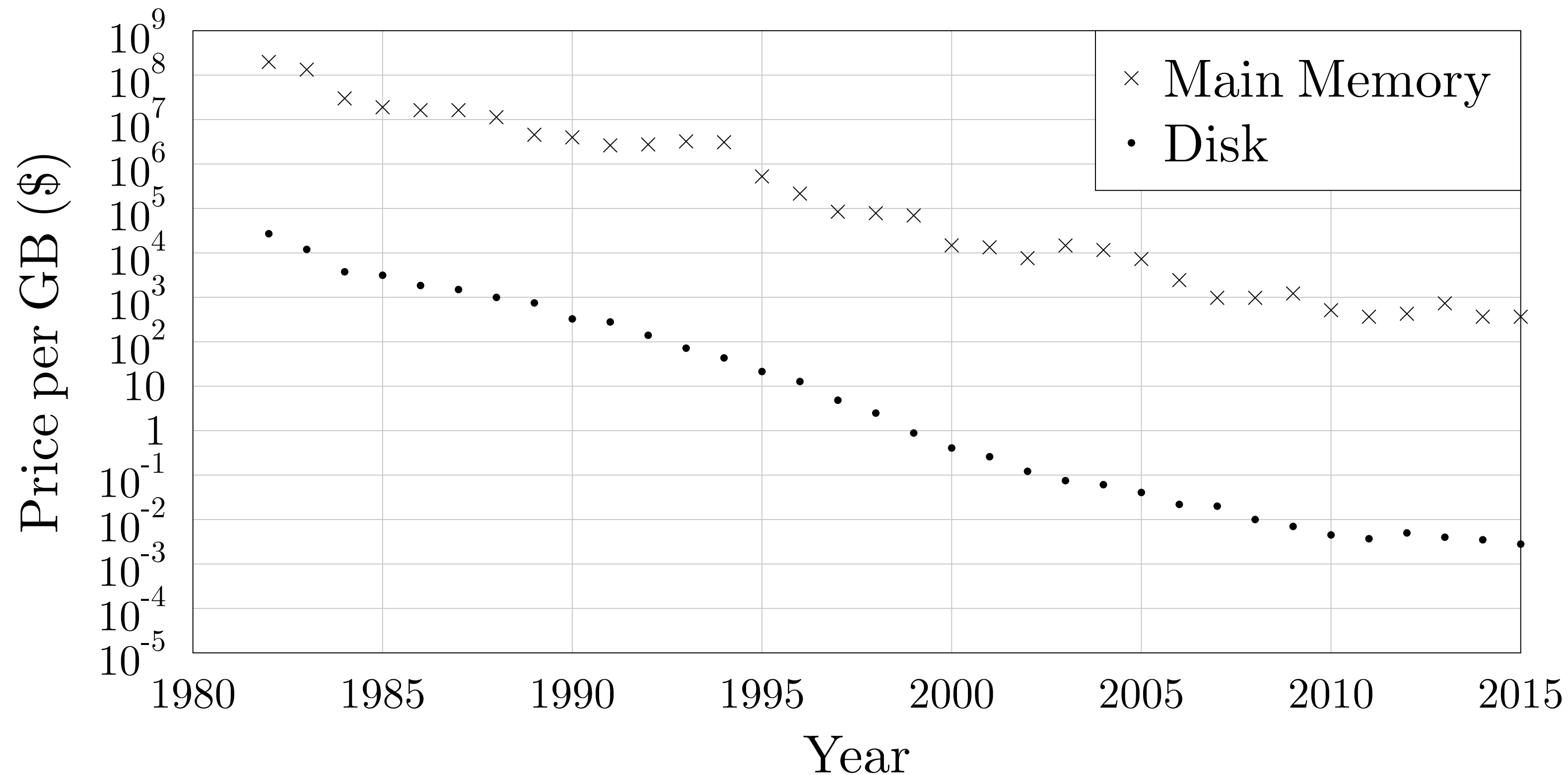
Bloom Filters

Answers set-membership queries
Smaller than array, and stored in main memory
Purpose: avoid accessing disk if entry is not in array
Subtlety: may return false positives.



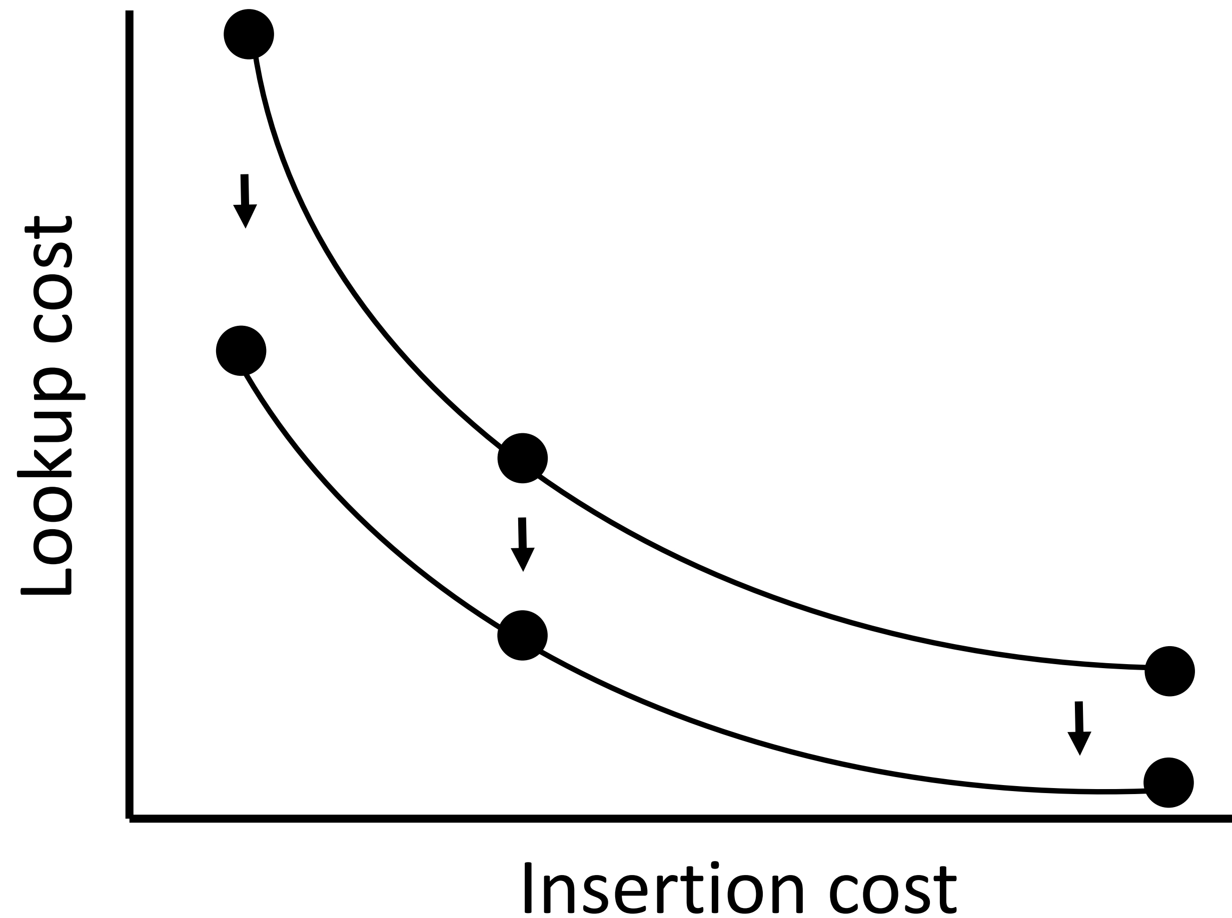
Bloom Filters

The more main memory, the fewer false positives ⟹ cheaper lookups



Bloom Filters

The more main memory, the fewer false positives ⟹ cheaper lookups



Conclusions

Write-optimized

Highly tunable

Backbone of many modern systems

Trade-off between lookup and insert cost (tiering/leveling, size ratio)

Trade main memory for lookup cost (fence pointers, Bloom filters)

Thank you!