

CS460: Intro to Database Systems

# Class 14: Log-Structured-Merge Trees

Instructor: Manos Athanassoulis

<https://midas.bu.edu/classes/CS460/>

# Useful when?

- Massive dataset
- Rapid updates/insertions
- Fast lookups

⇒ LSM-trees are for you.

# Why now?

Patrick O'Neil  
UMass Boston



Invented in  
1996



1980

1990

2000

2010

Time



levelDB



DynamoDB



amazon  
webservices™  
riak

rocksdb

cassandra

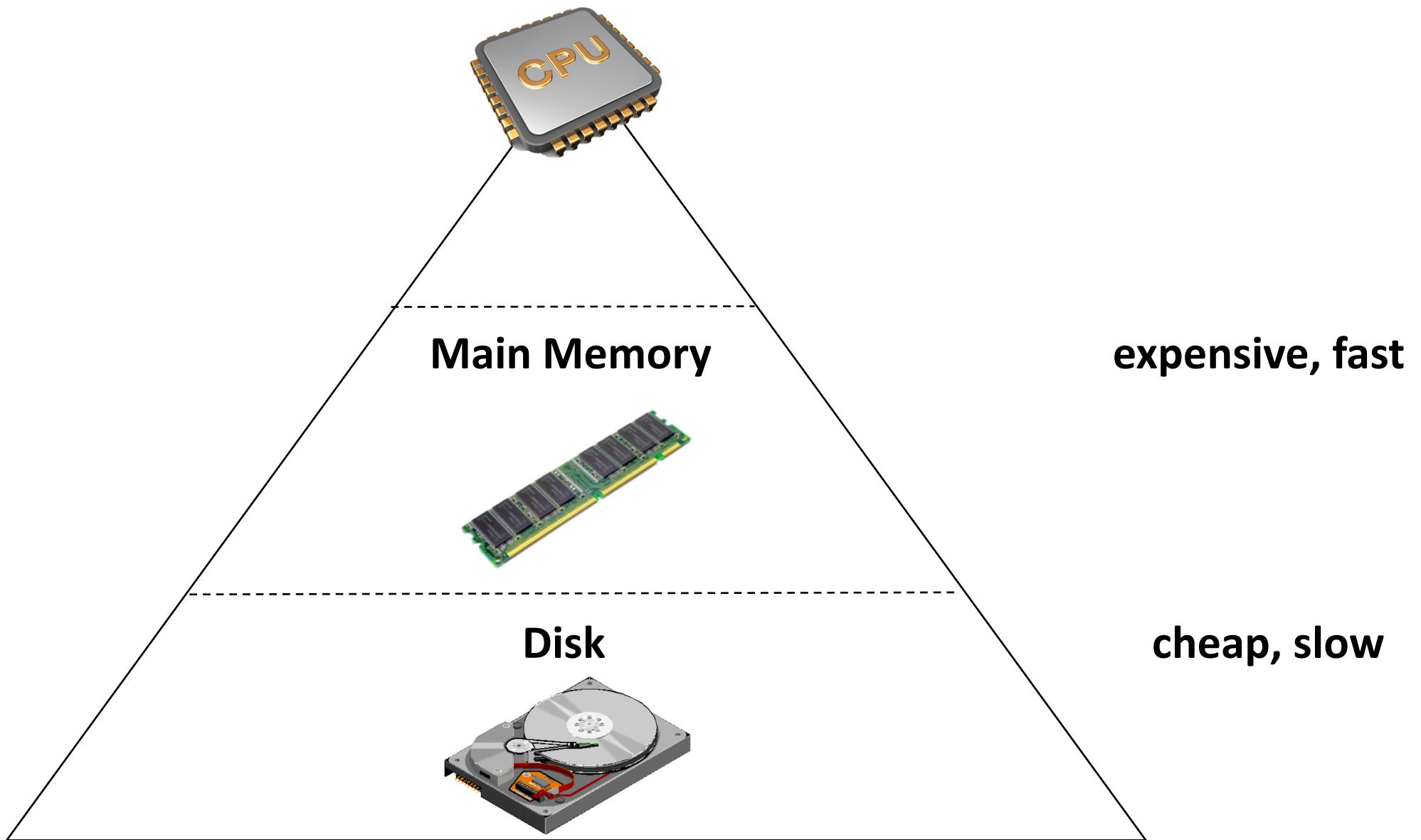
hbase

accumulo™

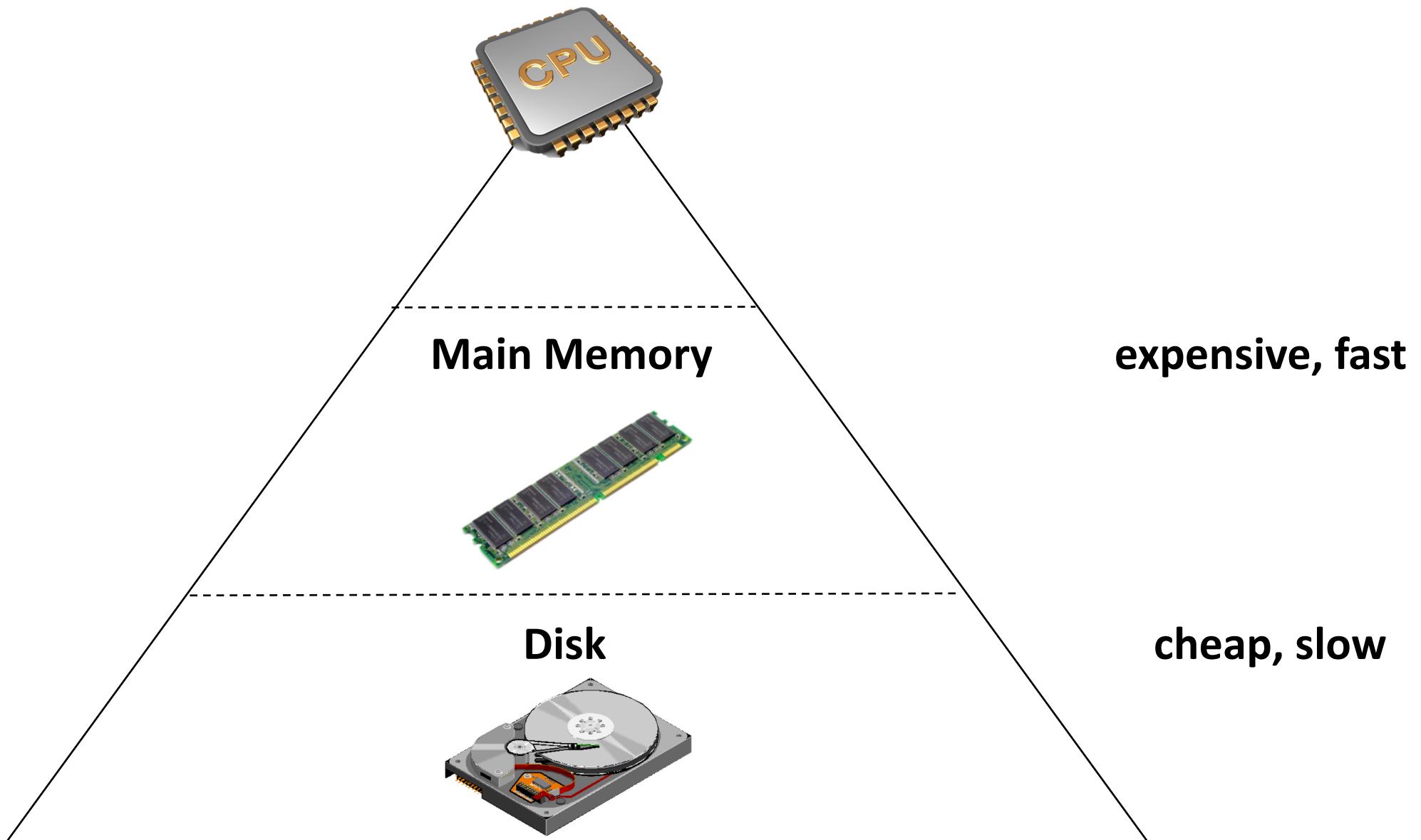
# Outline

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

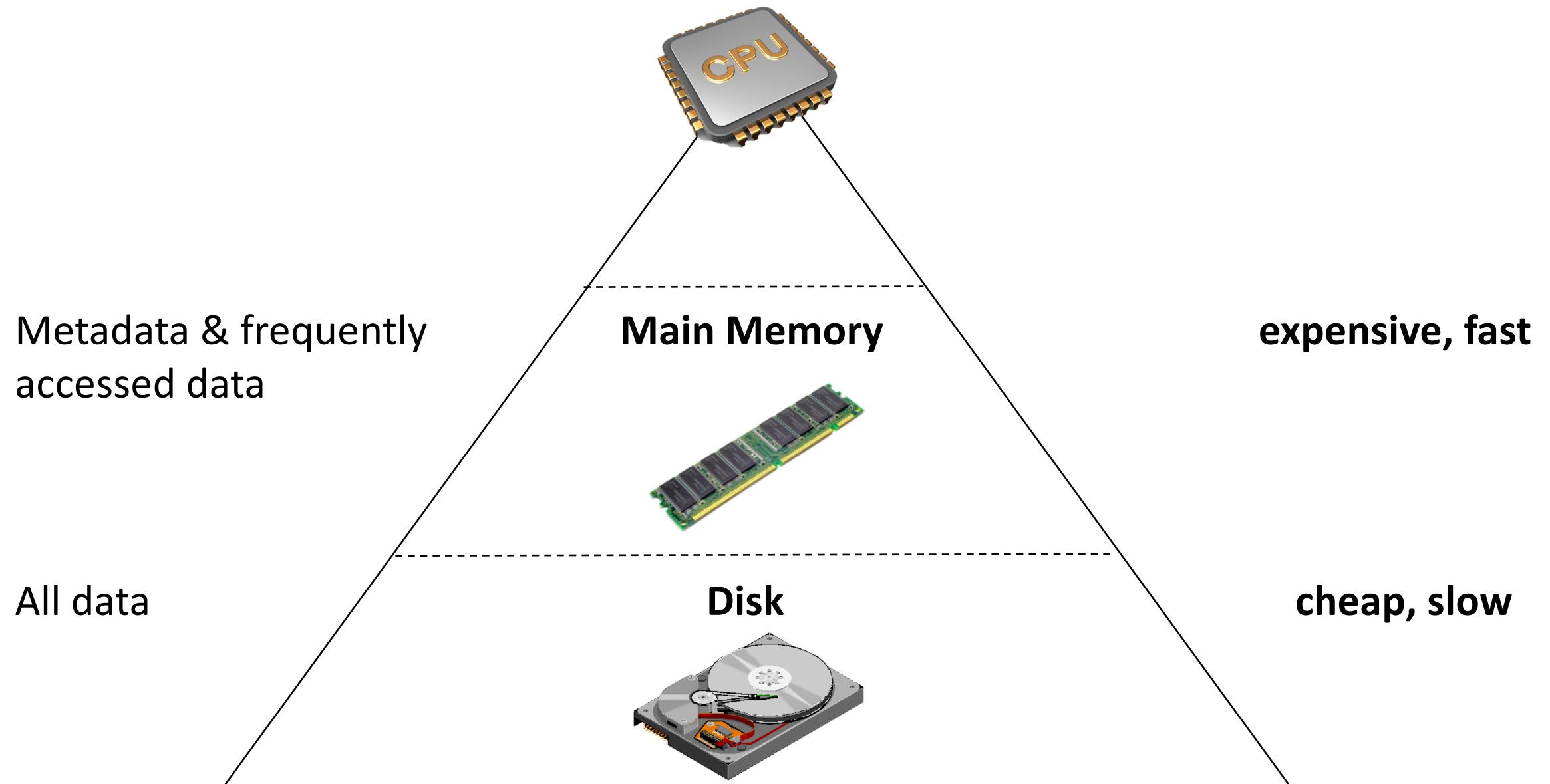
# Storage devices

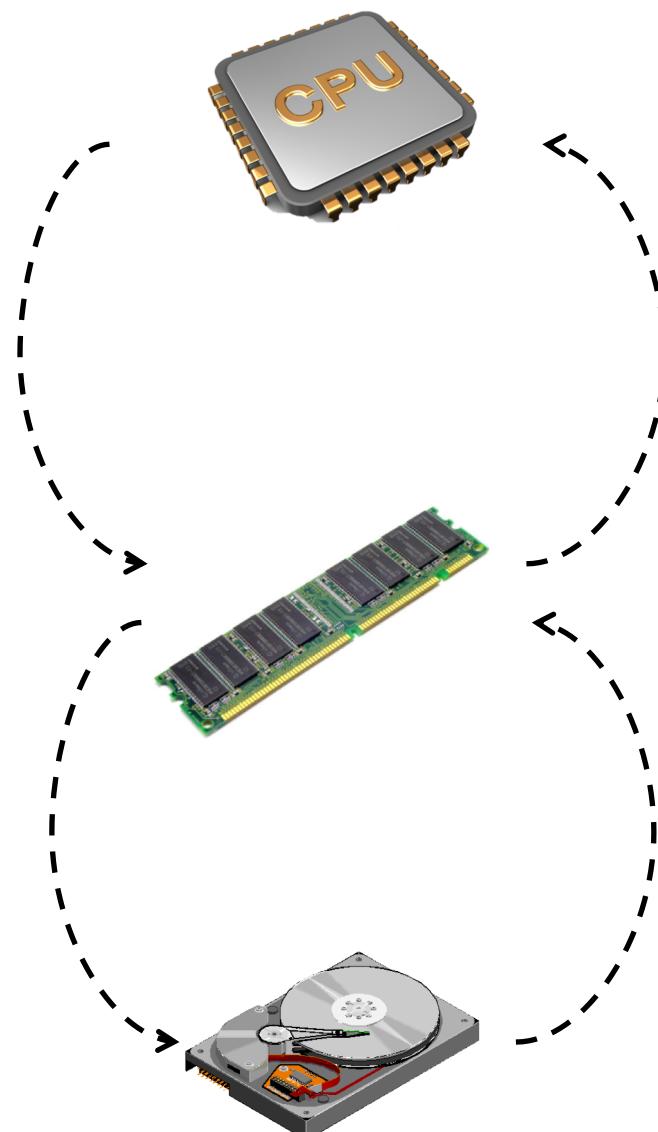


# The Memory Hierarchy

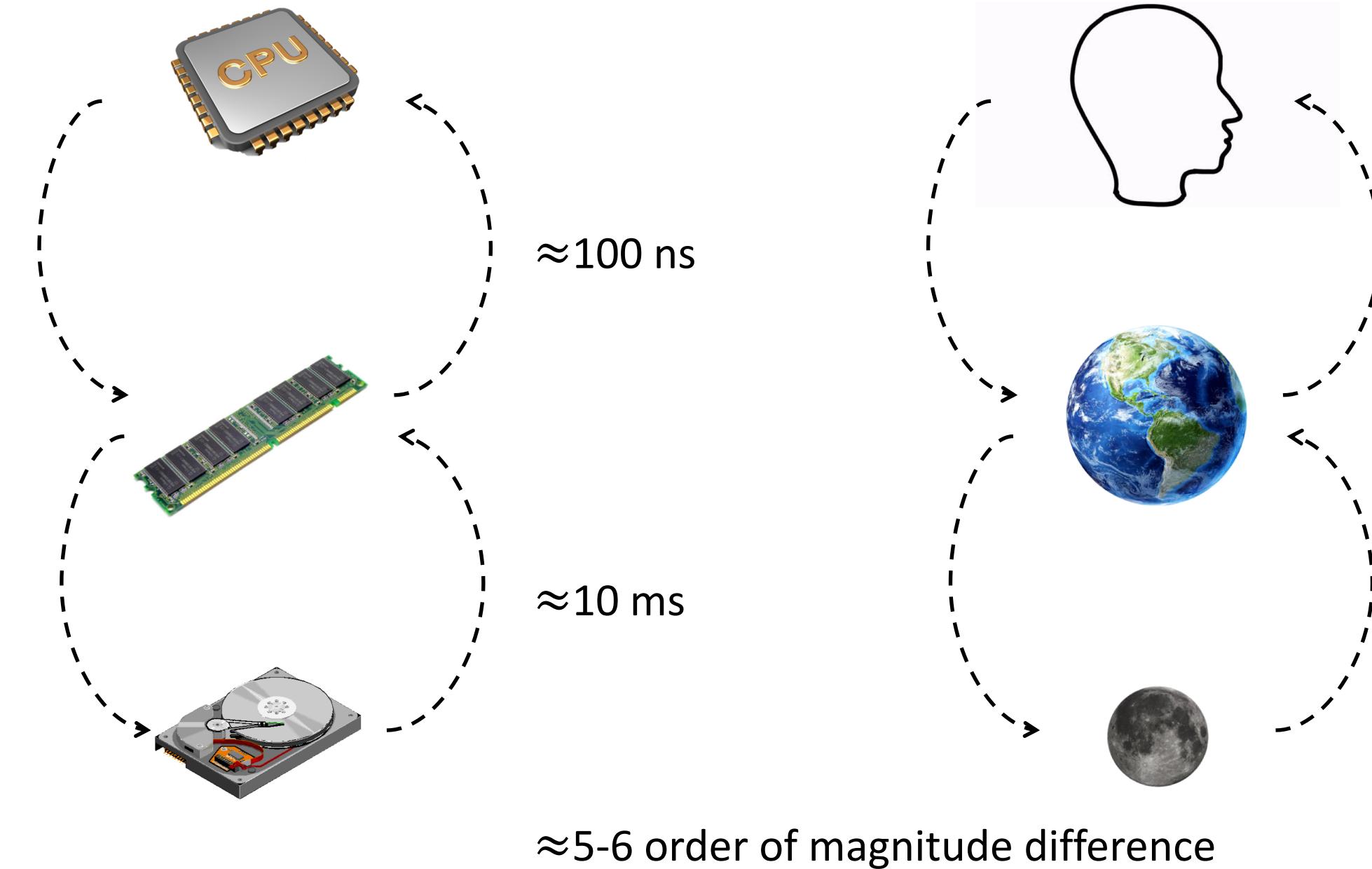


# The Memory Hierarchy

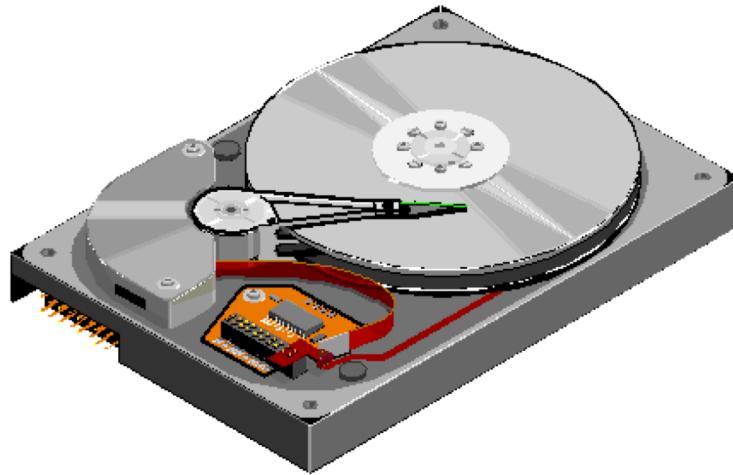




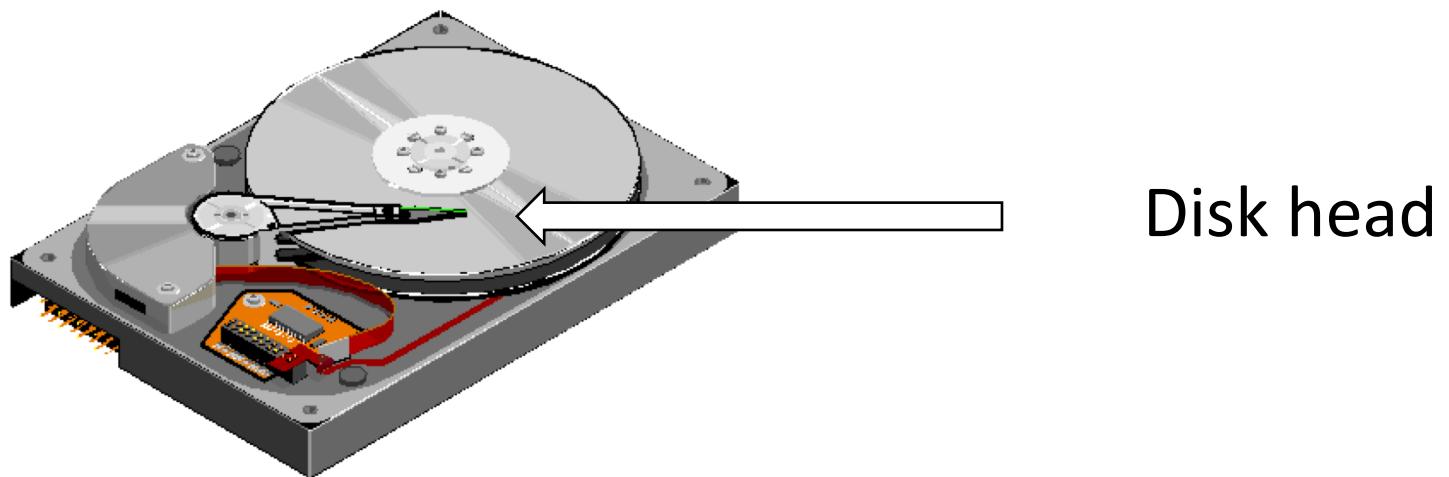
$\approx 5\text{-}6 \text{ order of magnitude difference}$



# Why is disk slow?

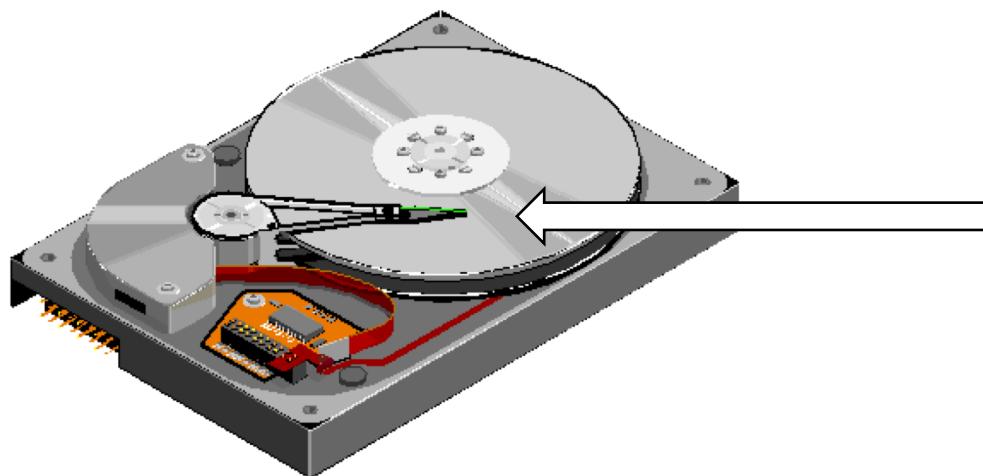


# Why is disk slow?



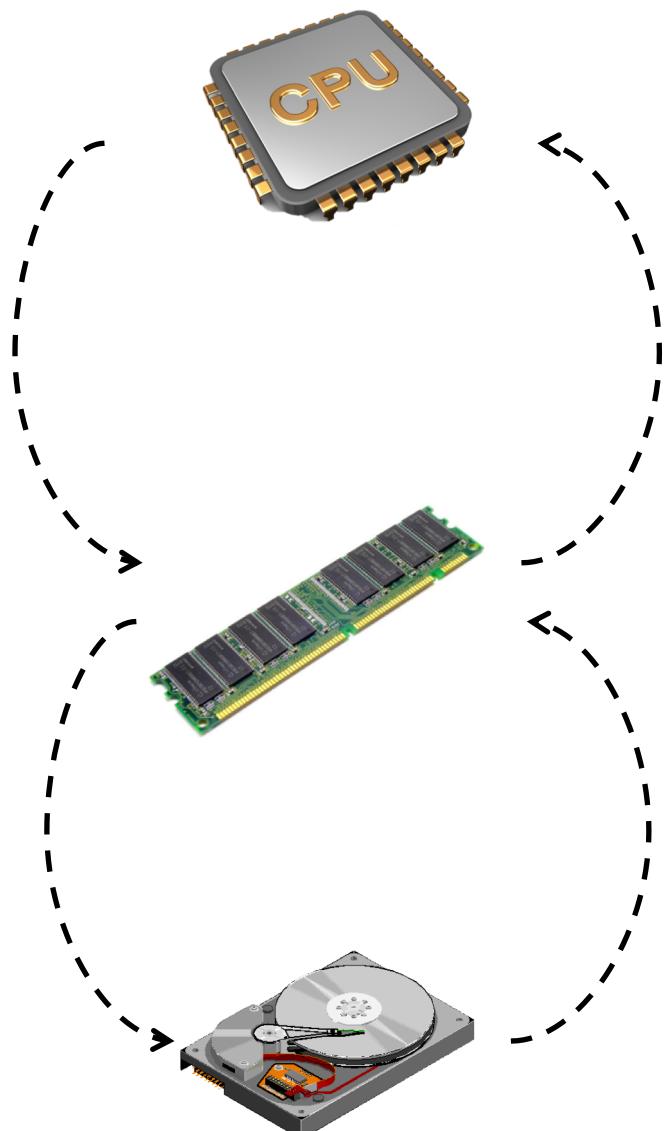
Disk head

# Why is disk slow?



Disk head

- Random access is slow       $\Rightarrow$       move disk head
- Sequential access is faster       $\Rightarrow$       let disk spin

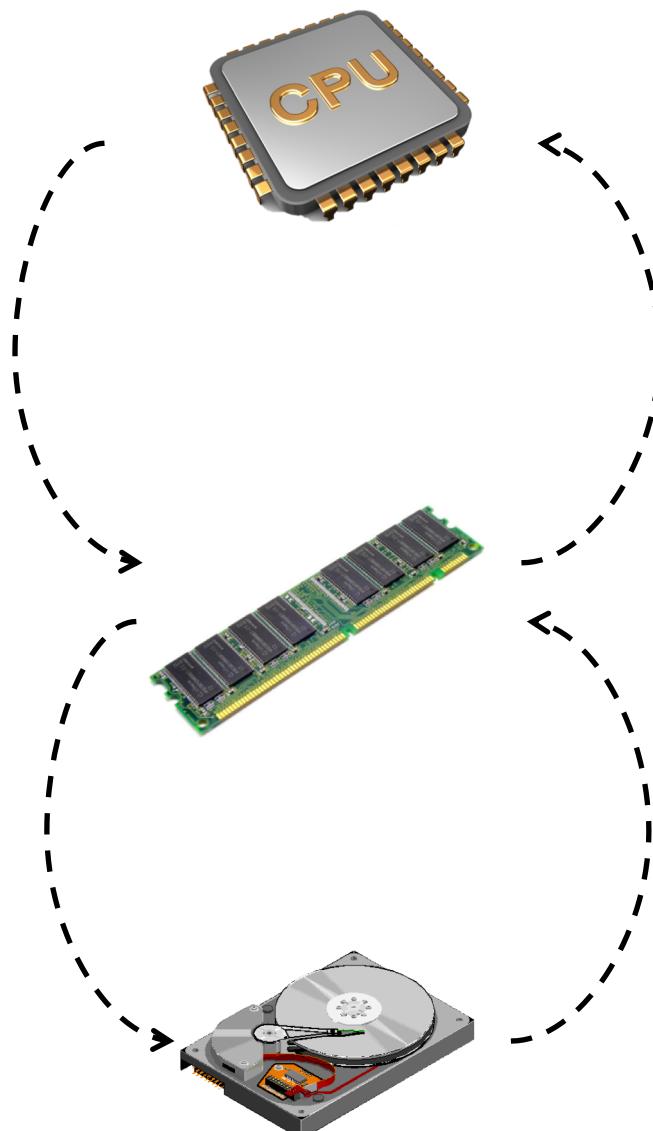


64 byte chunks  
Words

**Fine access granularity**

4 kilobyte chunks  
Blocks

**Coarse access granularity**



64 byte chunks  
Words

**Fine access granularity**

4 kilobyte chunks  
**Blocks**

**Coarse access granularity**

# 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

# Indexing Problem

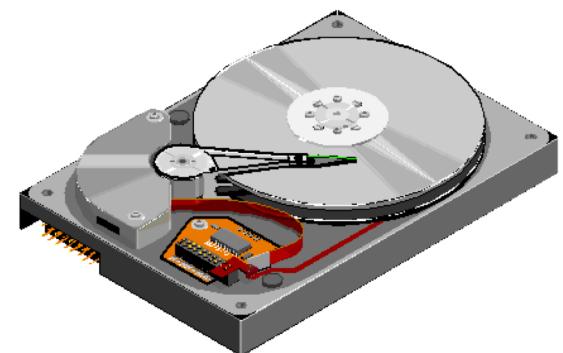


names → phone numbers

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		

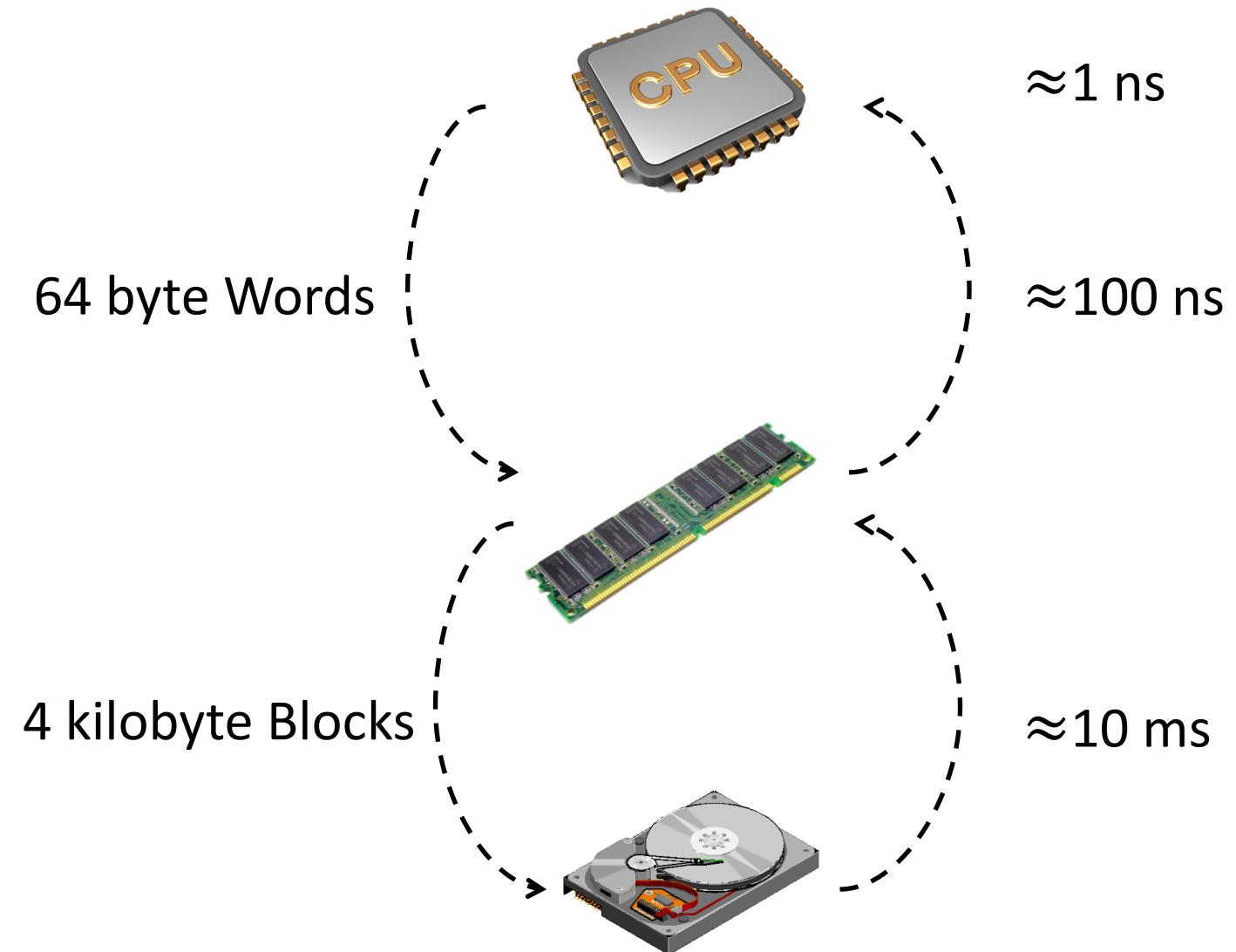
# Results Catalogue

Compare and contrast data structures.

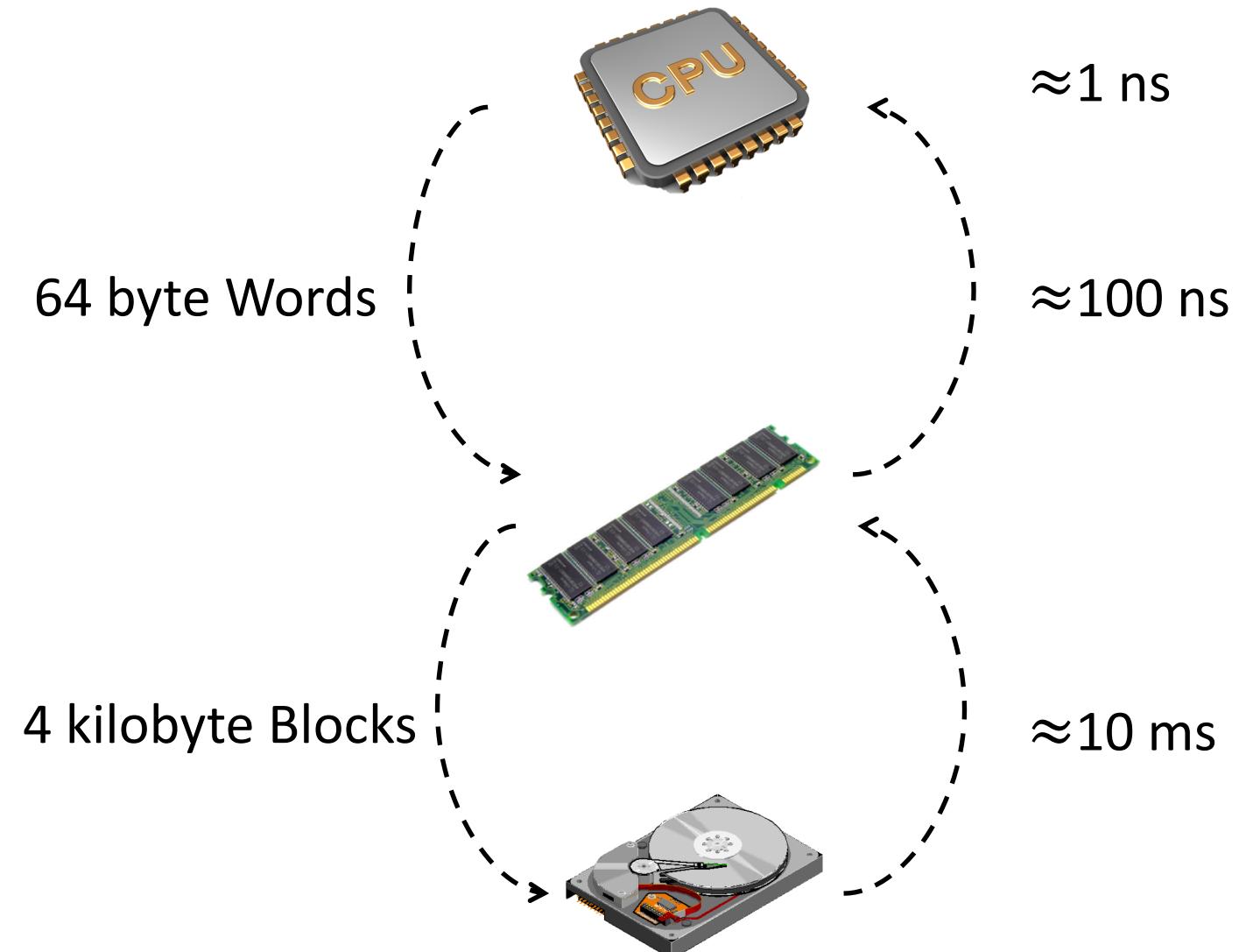
What to use when?

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

# Modeling Performance

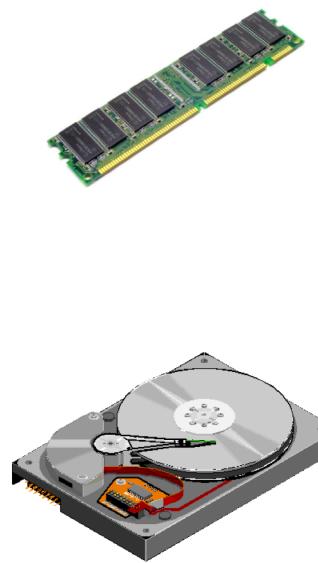


# Modeling Performance



Measure bottleneck:  
Number of block reads/writes (I/O)

# Sorted Array



Buffer
James
Sara

Array size | Pointer



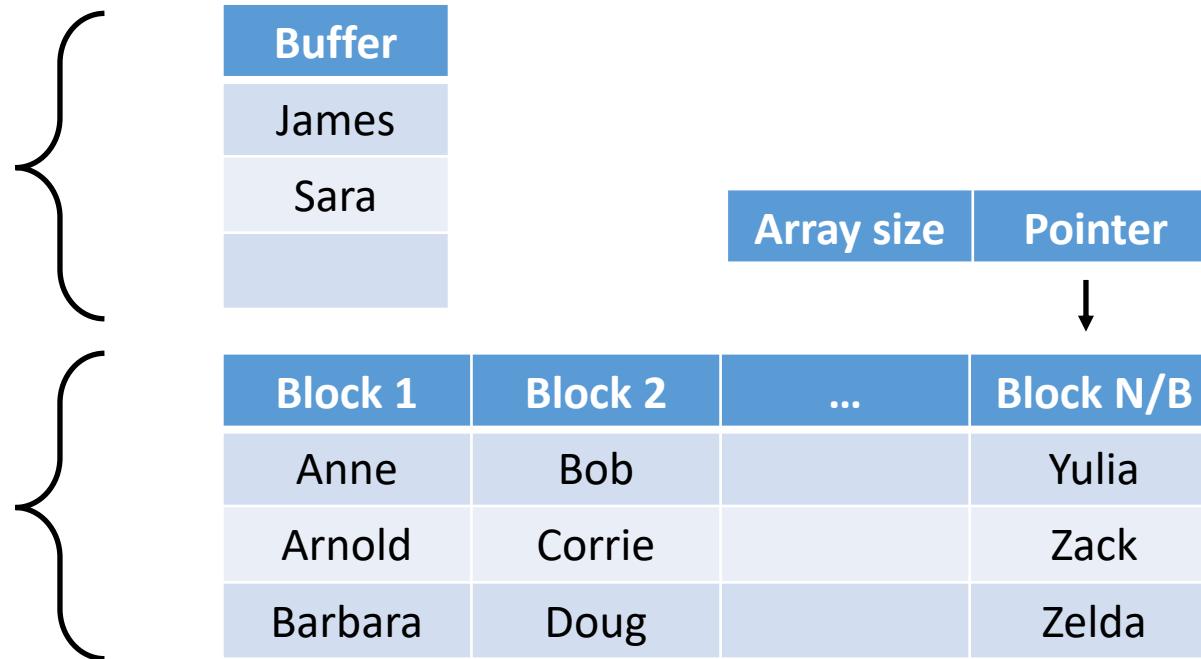
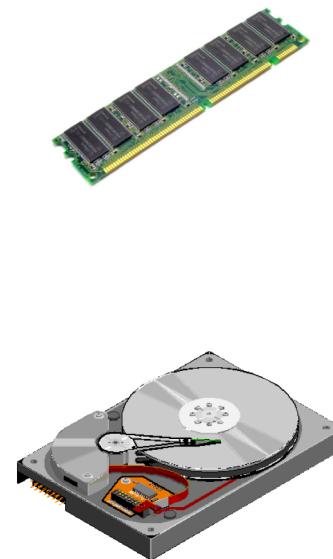
Block 1	Block 2	...	Block N/B
Anne	Bob		Yulia
Arnold	Corrie		Zack
Barbara	Doug		Zelda

# Sorted Array

**N** entries

**B** entries fit into a disk block

Array spans **N/B** disk blocks



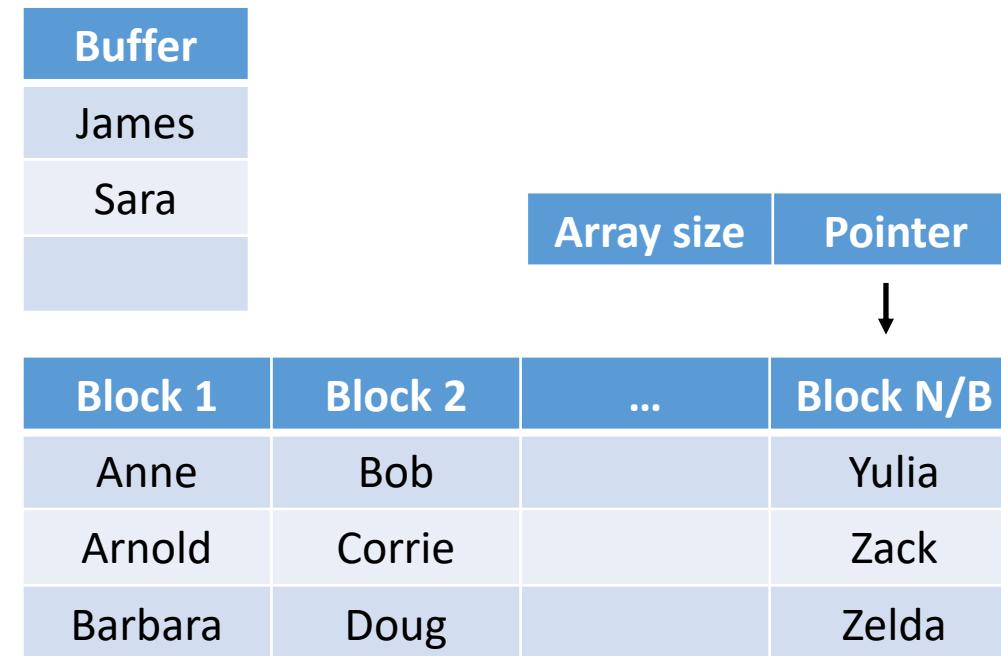
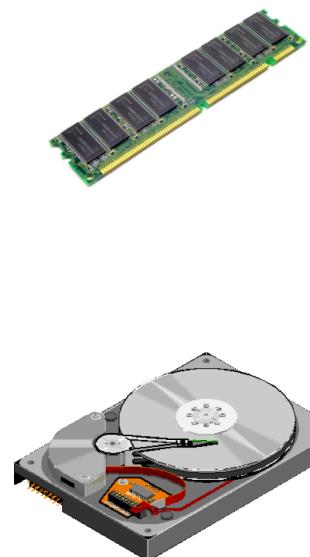
# Sorted Array

**N** entries

**B** entries fit into a disk block

Array spans **N/B** disk blocks

Lookup method & cost?



# Sorted Array

**N** entries

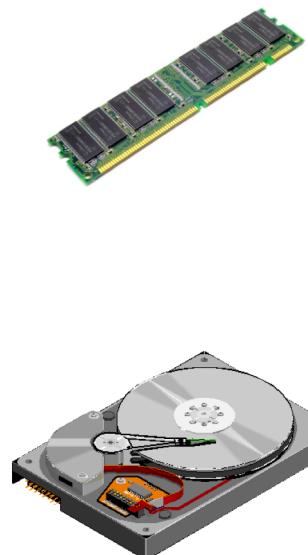
**B** entries fit into a disk block

Array spans **N/B** disk blocks

Lookup method & cost?

Binary search:

$$O\left(\log_2\left(\frac{N}{B}\right)\right) \text{ I/Os}$$



Buffer		Array size	Pointer
James			
Sara			
Block 1	Block 2	...	Block N/B
Anne	Bob		Yulia
Arnold	Corrie		Zack
Barbara	Doug		Zelda

# Sorted Array

**N** entries

**B** entries fit into a disk block

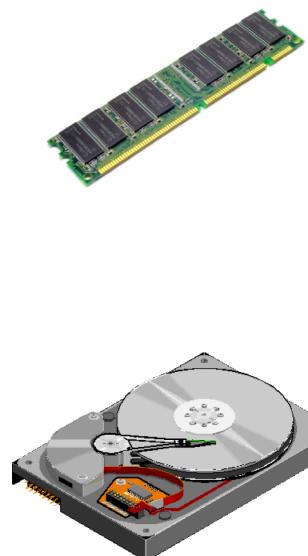
Array spans **N/B** disk blocks

Lookup method & cost?

Binary search:

$O\left(\log_2\left(\frac{N}{B}\right)\right)$  I/Os

Insertion cost?



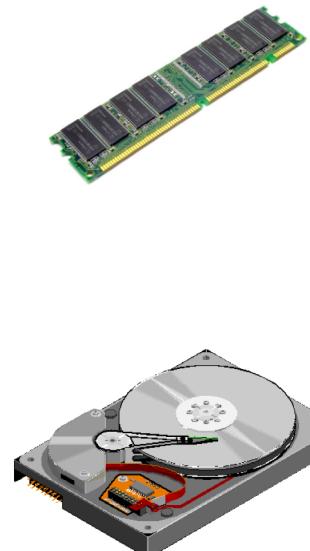
Buffer			
James			
Sara			
		Array size	Pointer
			↓
Block 1	Block 2	...	Block N/B
Anne	Bob		Yulia
Arnold	Corrie		Zack
Barbara	Doug		Zelda

# Sorted Array

**N** entries

**B** entries fit into a disk block

Array spans **N/B** disk blocks



Buffer
James
Sara

Lookup method & cost?

Binary search:  $O\left(\log_2\left(\frac{N}{B}\right)\right)$  I/Os

Insertion cost?

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

Array size			
Block 1	Block 2	...	Block N/B
Anne	Bob		Yulia
Arnold	Corrie		Zack
Barbara	Doug		Zelda

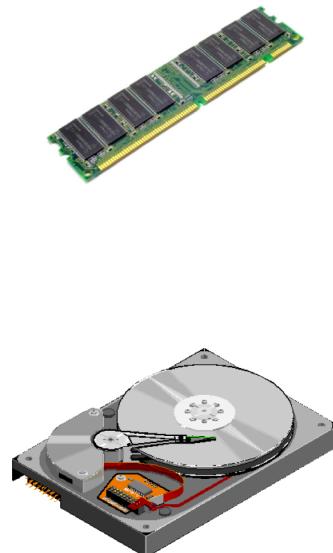
# Results Catalogue

	Lookup cost	Insertion cost
<b>Sorted array</b>	$O(\log_2(N/B))$	$O(N/B^2)$
Log		
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

# Results Catalogue

	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N/B))$	$O(N/B^2)$
<b>Log</b>		
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

# Log (append-only array)



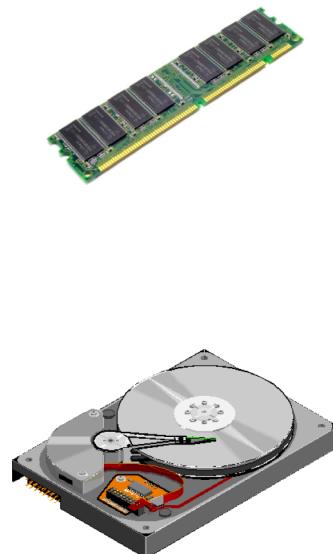
Buffer		Array size	Pointer
James			
Sara			
Block 1	Block 2	...	Block N/B
Doug	Yulia		Anne
Zelda	Zack		Bob
Arnold	Barbara		Corrie

# Log (append-only array)

**N** entries

**B** entries fit into a disk block

Array spans **N/B** disk blocks

The diagram illustrates the structure of an append-only array. It shows a 'Buffer' structure containing two entries: 'James' and 'Sara'. To the right of the buffer is a header consisting of 'Array size' and 'Pointer' fields. An arrow points from the 'Pointer' field to a 'Block' array. The 'Block' array is organized into multiple disk blocks, labeled 'Block 1', 'Block 2', ..., 'Block N/B'. Each block contains two entries: 'Doug', 'Yulia', 'Zelda', 'Zack', 'Arnold', 'Barbara', 'Anne', 'Bob', and 'Corrie'.

Block 1	Block 2	...	Block N/B
Doug	Yulia		Anne
Zelda	Zack		Bob
Arnold	Barbara		Corrie

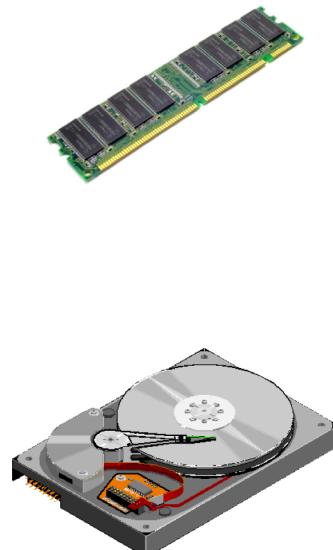
# Log (append-only array)

**N** entries

Lookup method & cost?

**B** entries fit into a disk block

Array spans **N/B** disk blocks



Buffer
James
Sara

Array size	Pointer



Block 1	Block 2	...	Block N/B
Doug	Yulia		Anne
Zelda	Zack		Bob
Arnold	Barbara		Corrie

# Log (append-only array)

**N** entries

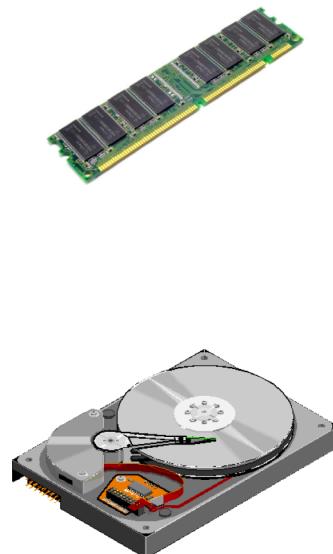
**B** entries fit into a disk block

Array spans **N/B** disk blocks

Lookup method & cost?

Scan:

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



Buffer
James
Sara

Array size | Pointer



Block 1	Block 2	...	Block N/B
Doug	Yulia		Anne
Zelda	Zack		Bob
Arnold	Barbara		Corrie

# Log (append-only array)

**N** entries

**B** entries fit into a disk block

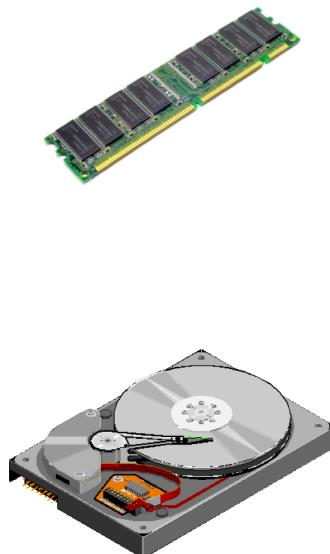
Array spans **N/B** disk blocks

Lookup method & cost?

Scan:

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

Insertion cost?



Buffer
James
Sara

Array size	Pointer



Block 1	Block 2	...	Block N/B
Doug	Yulia		Anne
Zelda	Zack		Bob
Arnold	Barbara		Corrie

# Log (append-only array)

**N** entries

**B** entries fit into a disk block

Array spans **N/B** disk blocks

Lookup method & cost?

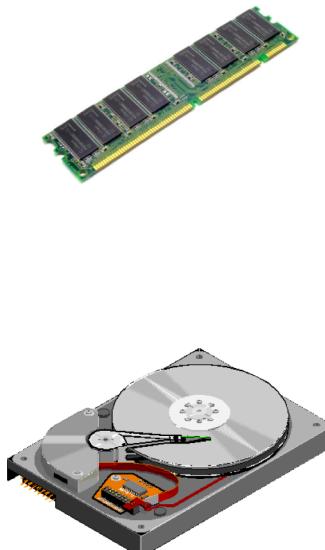
Scan:

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

Insertion cost?

Append:

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



Buffer
James
Sara

Array size	Pointer
------------	---------



Block 1	Block 2	...	Block N/B
Doug	Yulia		Anne
Zelda	Zack		Bob
Arnold	Barbara		Corrie

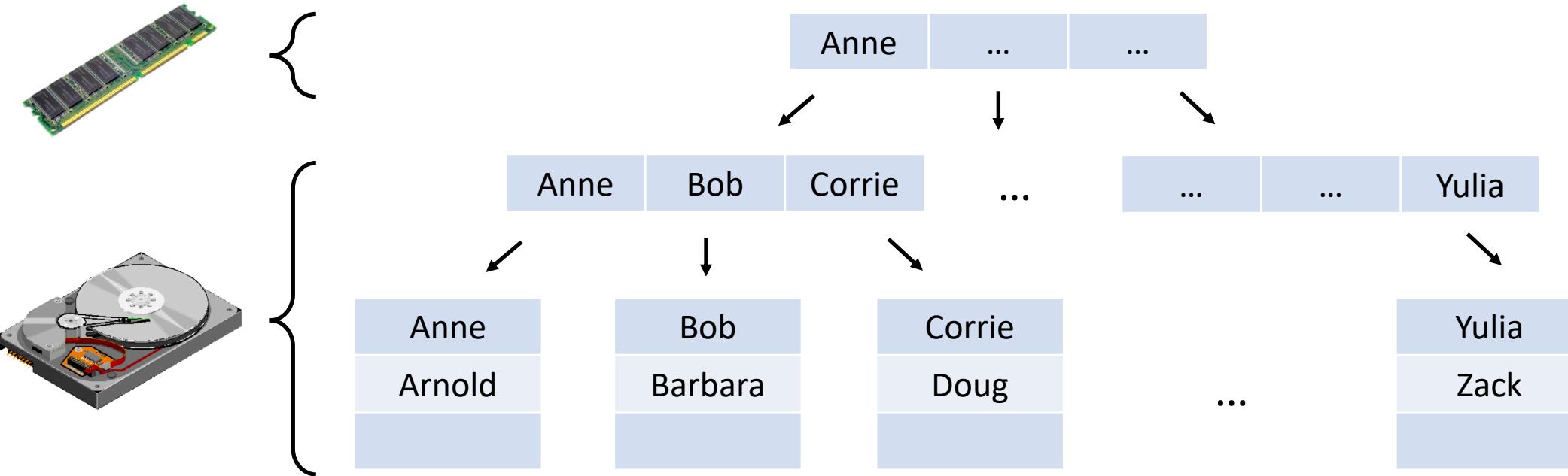
# Results Catalogue

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

# Results Catalogue

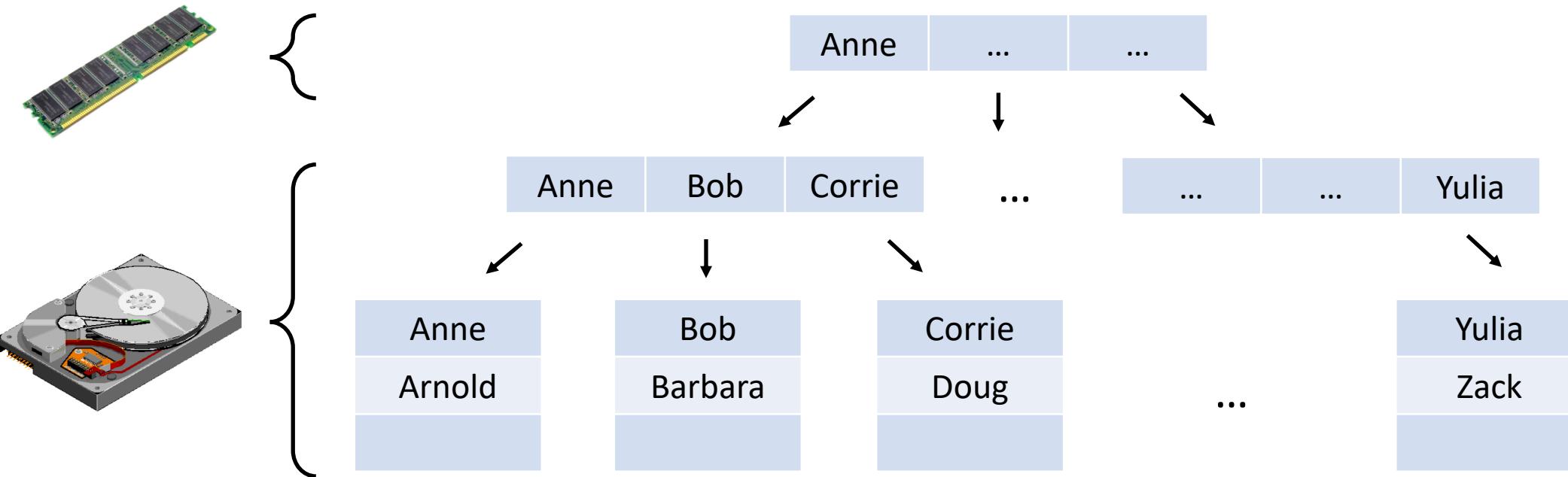
	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N/B))$	$O(N/B^2)$
Log	$O(N/B)$	$O(1/B)$
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

# B-tree



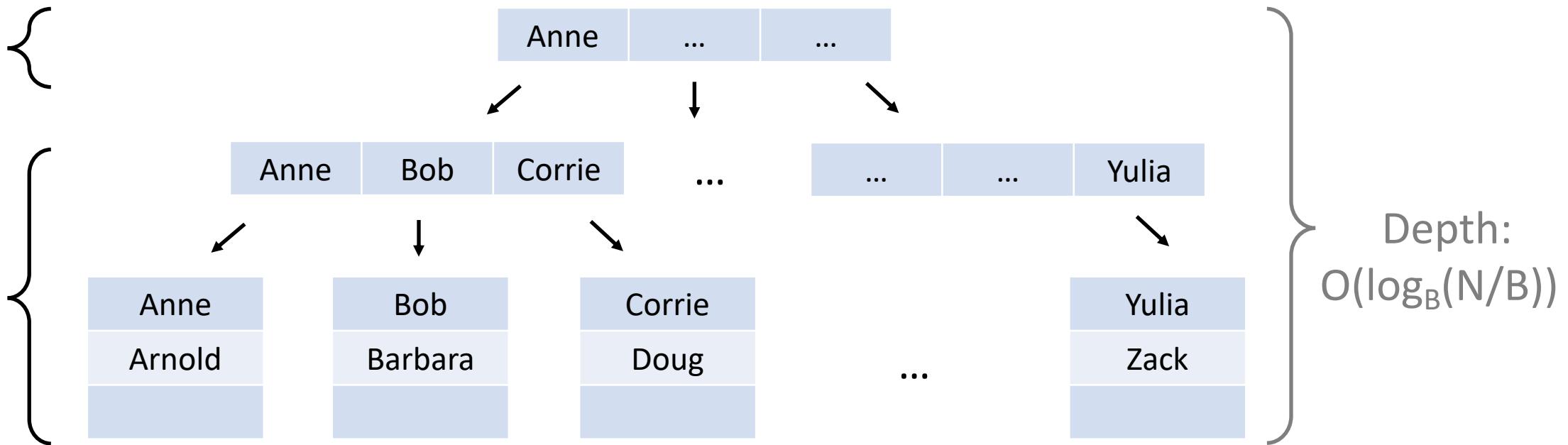
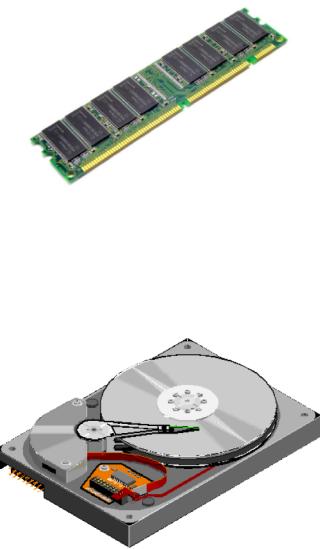
# B-tree

Lookup method & cost?



# B-tree

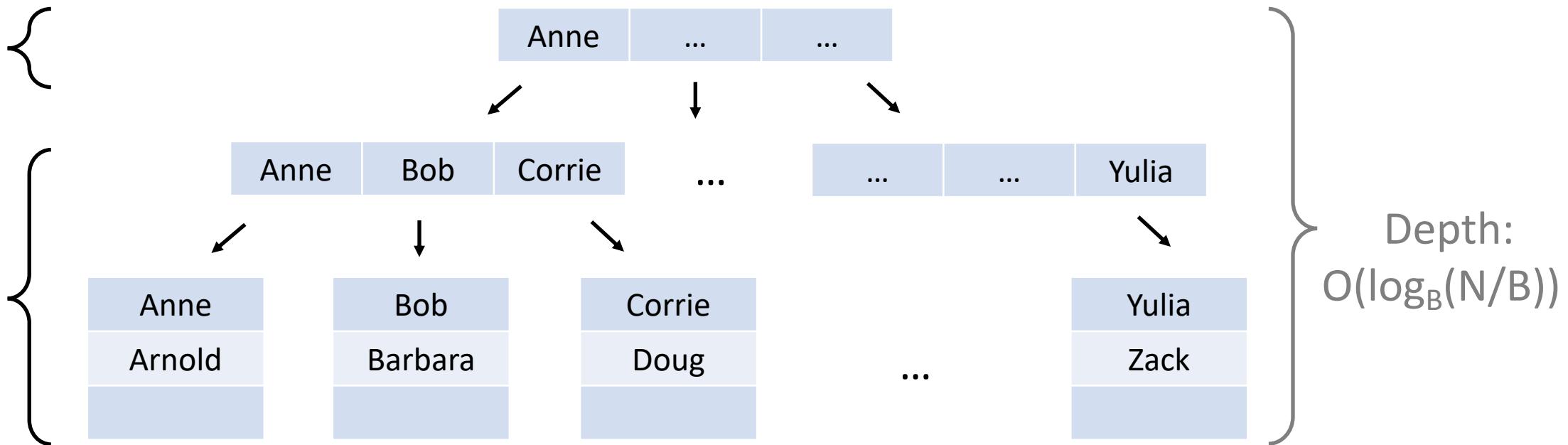
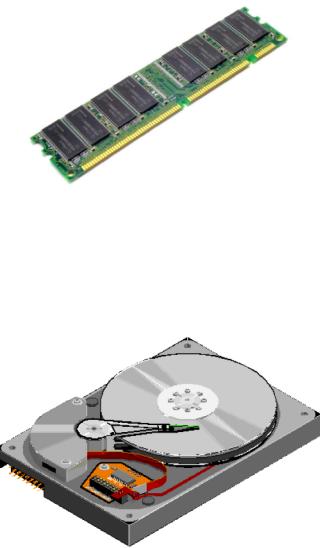
Lookup method & cost?



# B-tree

Lookup method & cost?

Tree search:  $O\left(\log_B \left(\frac{N}{B}\right)\right)$

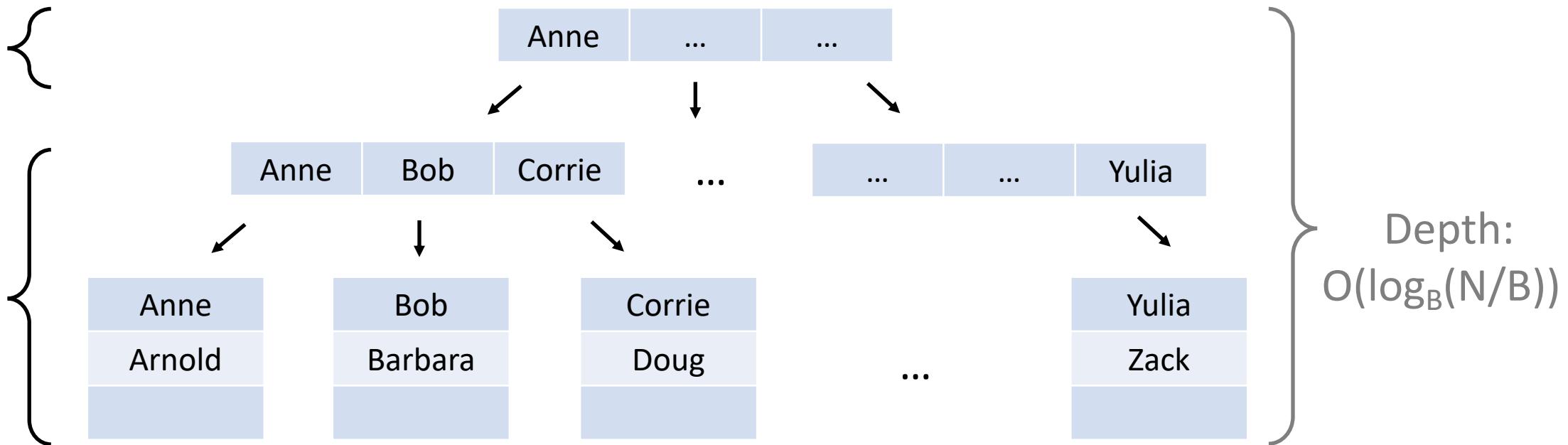
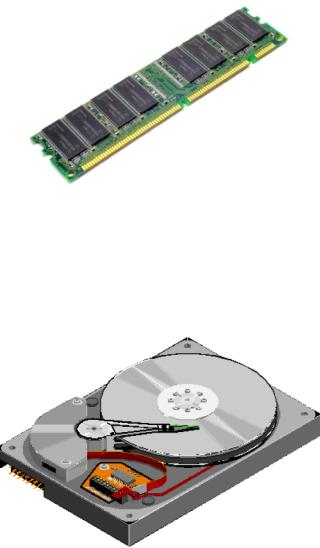


# B-tree

Lookup method & cost?

Tree search:  $O\left(\log_B \left(\frac{N}{B}\right)\right)$

Insertion method & cost?



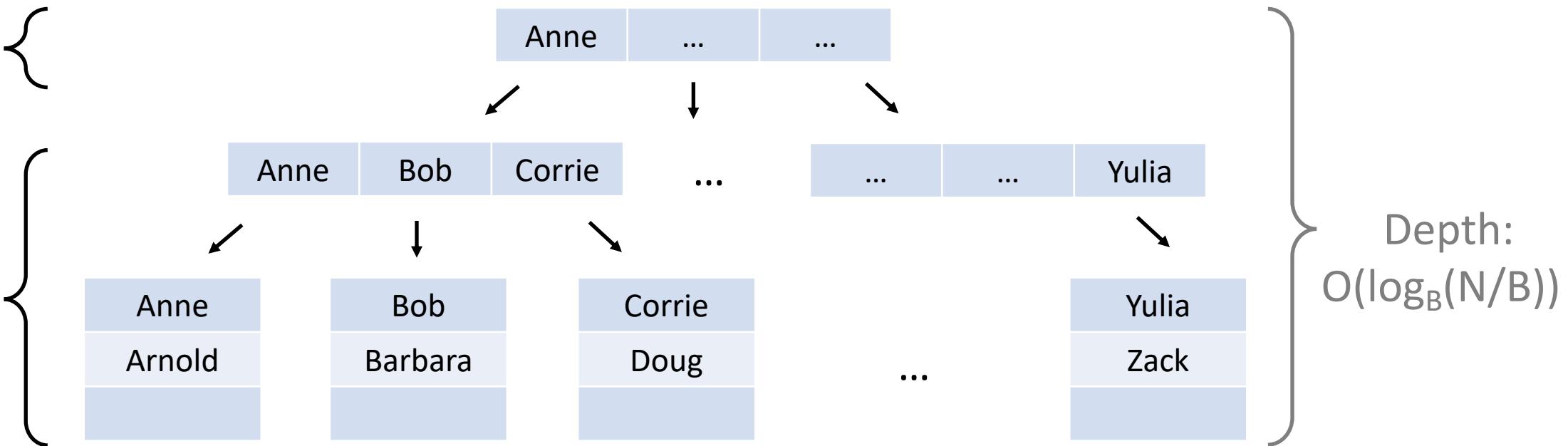
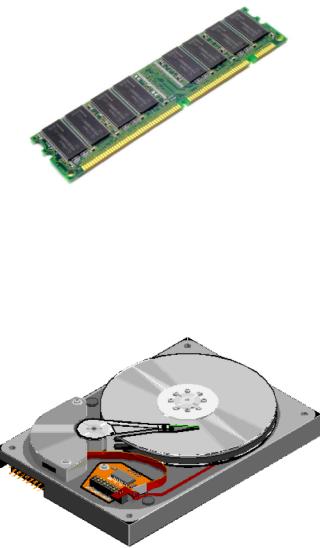
# B-tree

Lookup method & cost?

Tree search:  $O\left(\log_B \left(\frac{N}{B}\right)\right)$

Insertion method & cost?

Tree search & append:  $O\left(\log_B \left(\frac{N}{B}\right)\right)$



# Results Catalogue

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

# B-trees



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

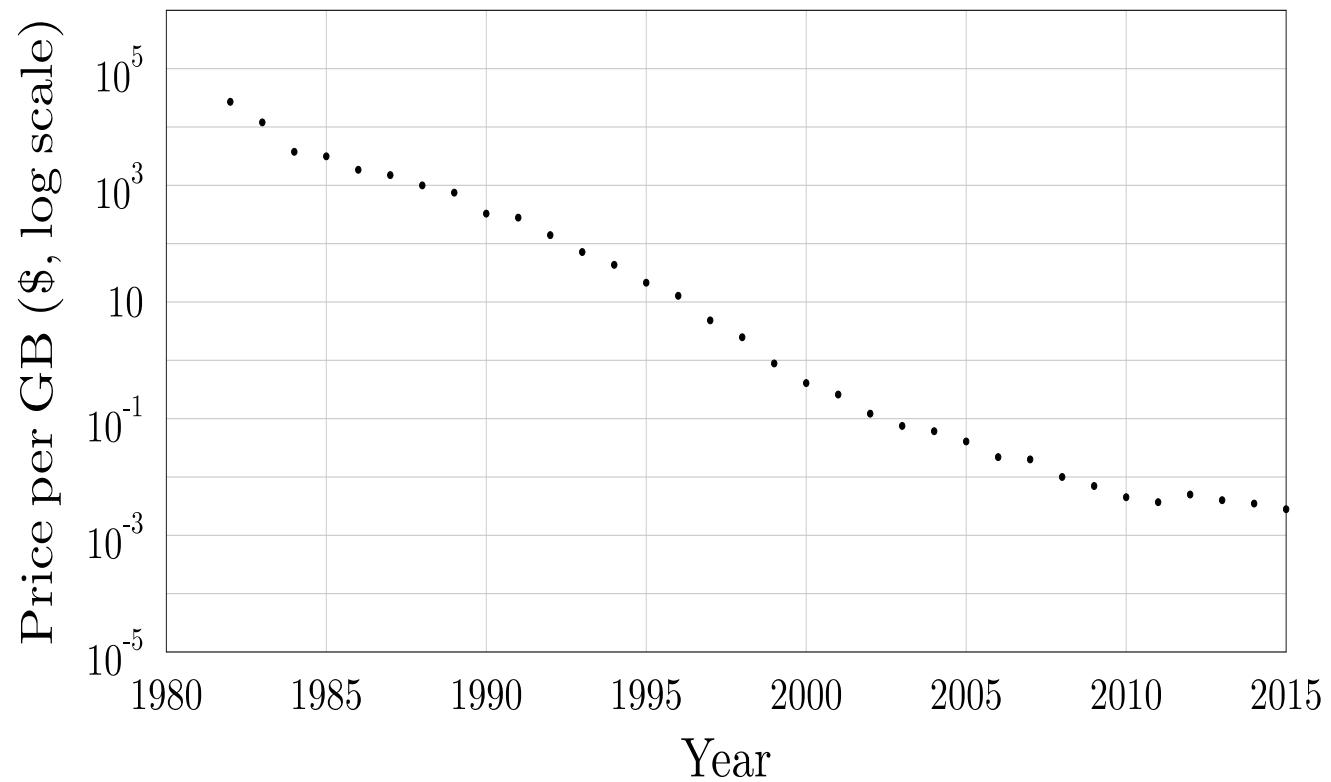
Goetz Graefe Microsoft, HP Fellow, now  
Google ACM Software System Award

# B-trees are no longer sufficient

Cheaper to store data

Workloads more insert-intensive

We need better insert-performance.



# Results Catalogue

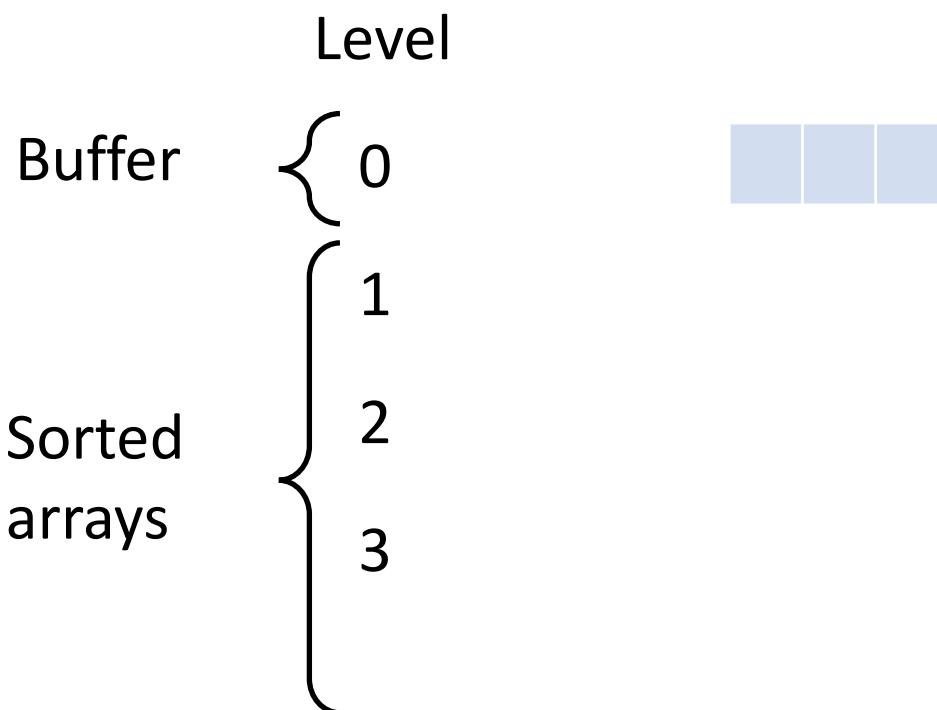
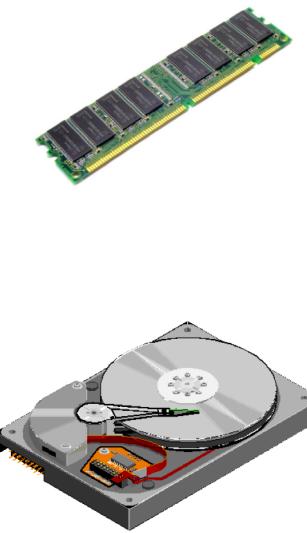
Goal to combine

sub-constant insertion cost  
logarithmic lookup cost

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

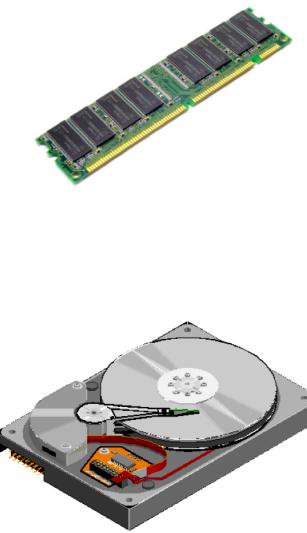
# Basic LSM-trees

# Basic LSM-tree



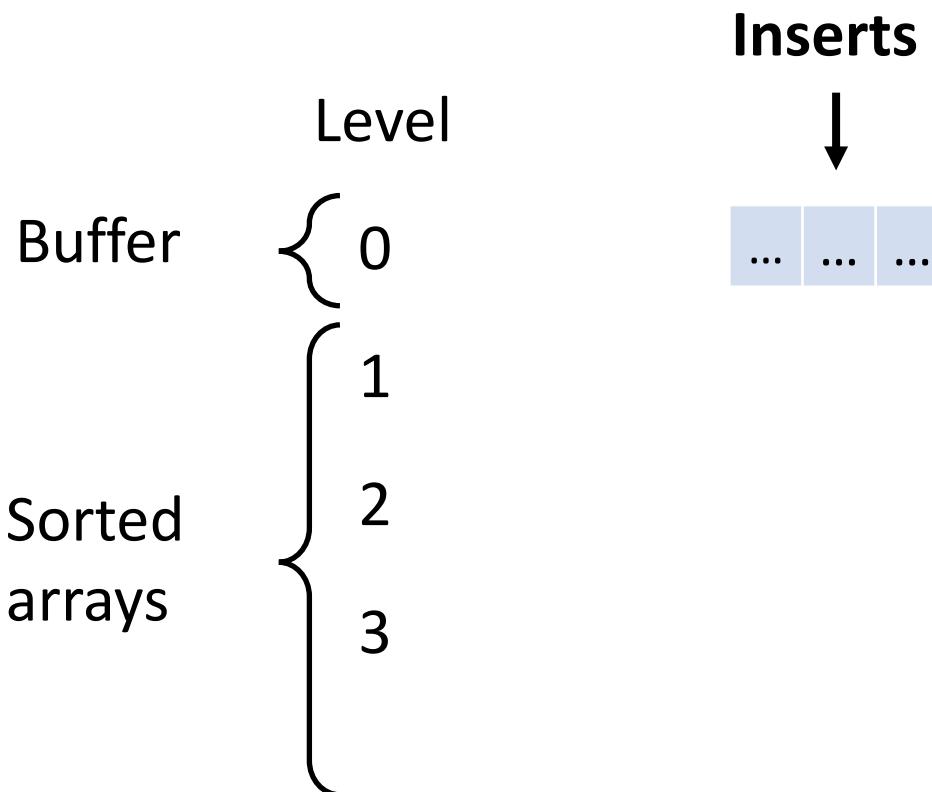
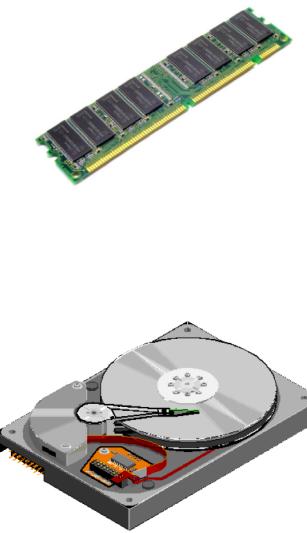
# Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering



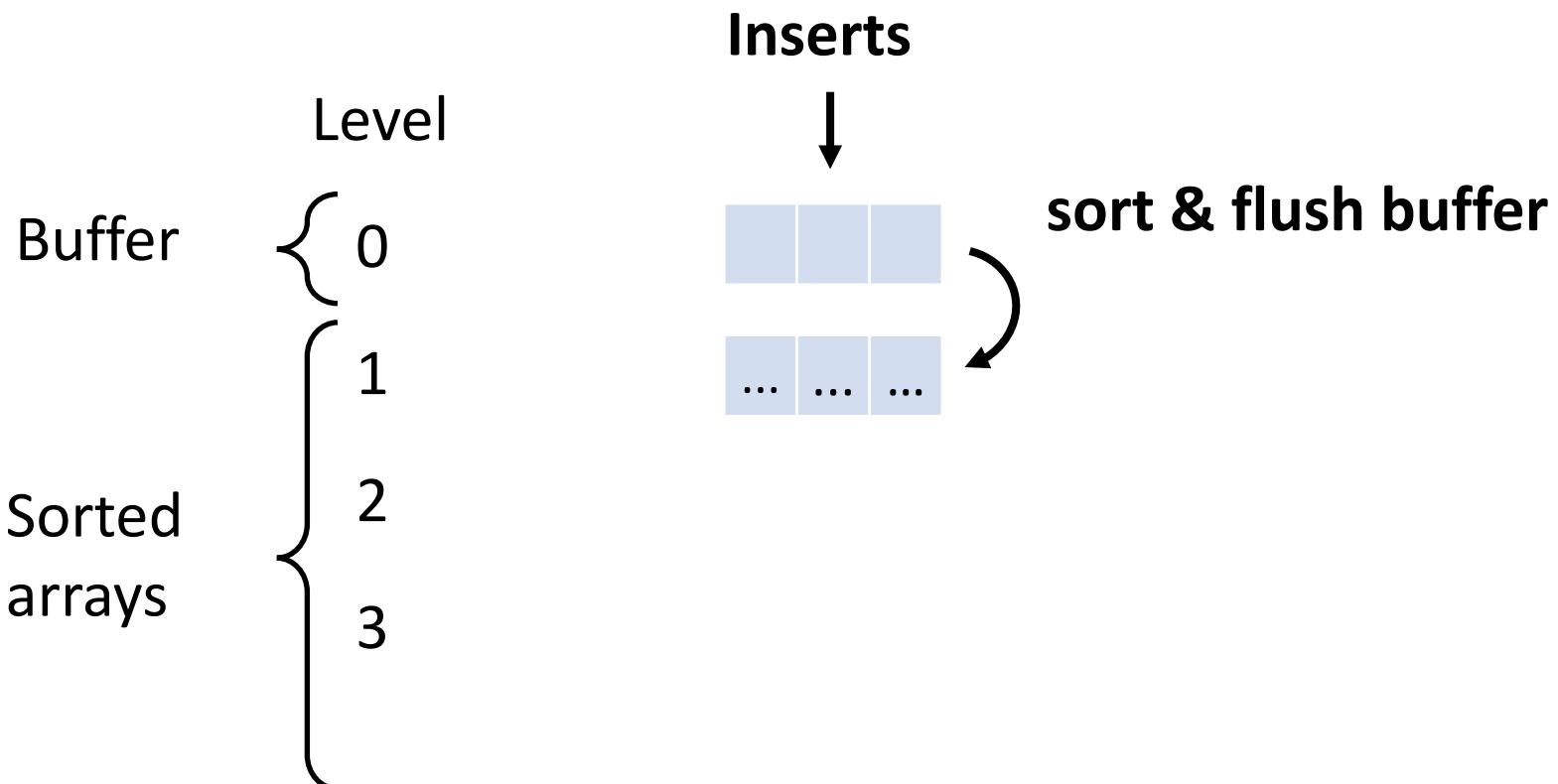
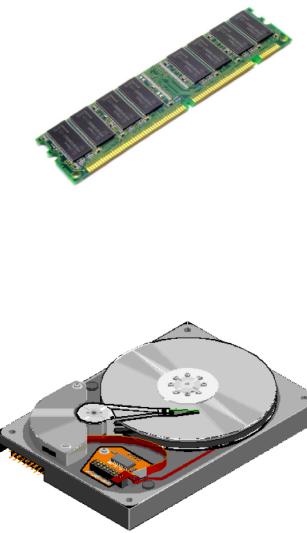
# Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering



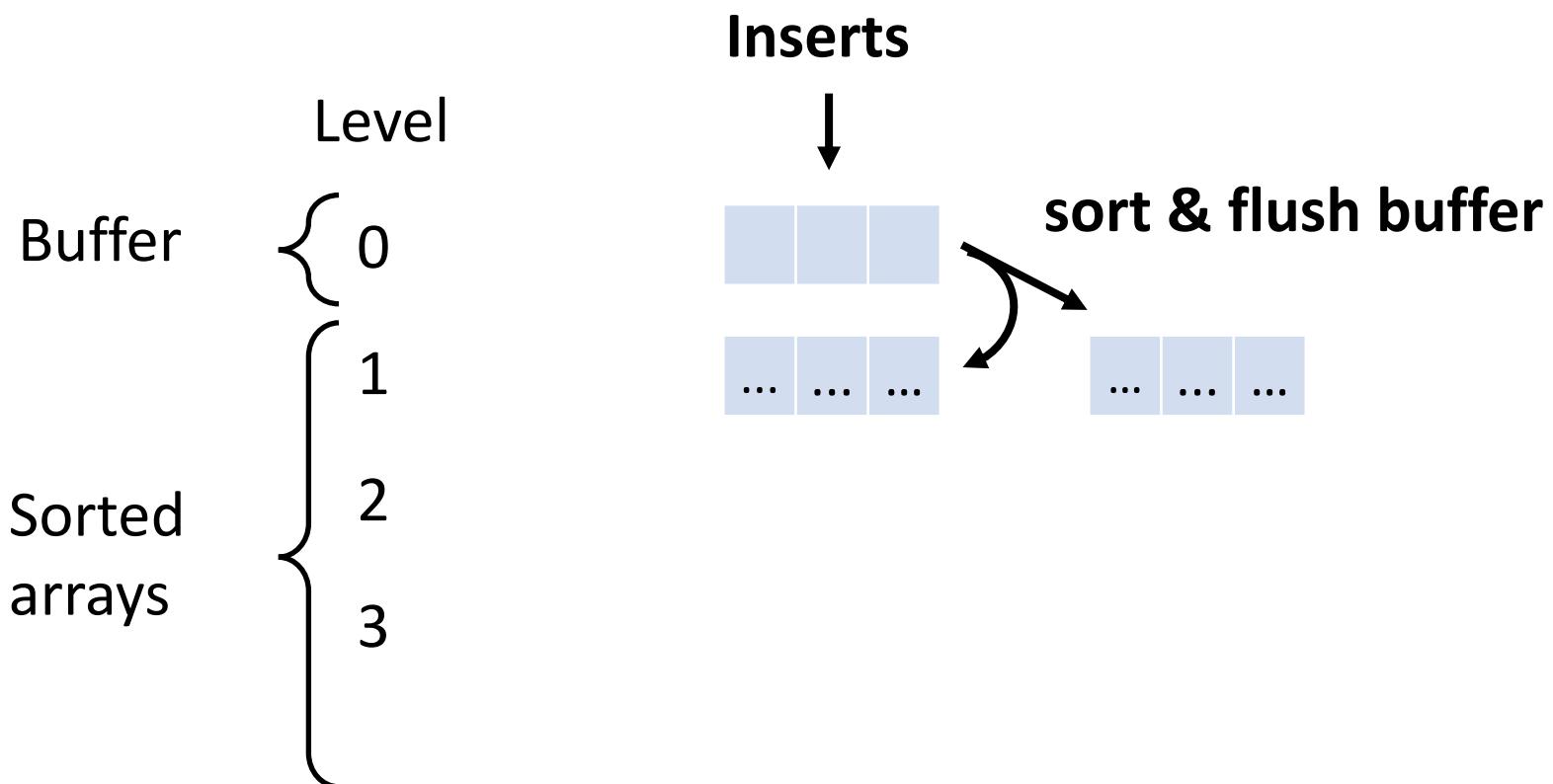
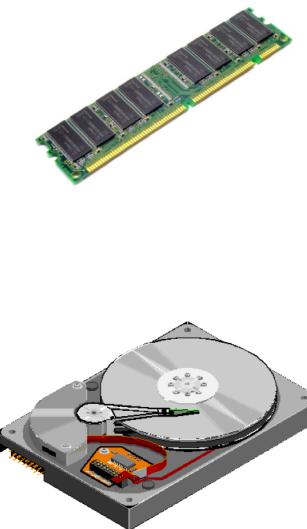
# Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering



# Basic LSM-tree

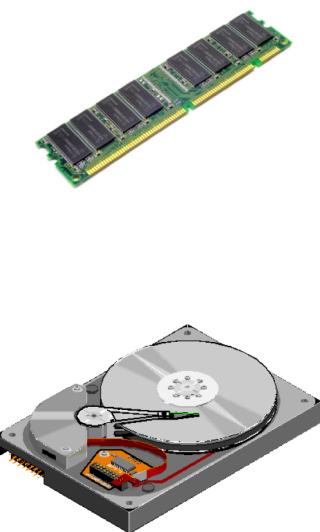
*Design principle #1:* optimize for insertions by buffering



# Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

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



**Inserts**

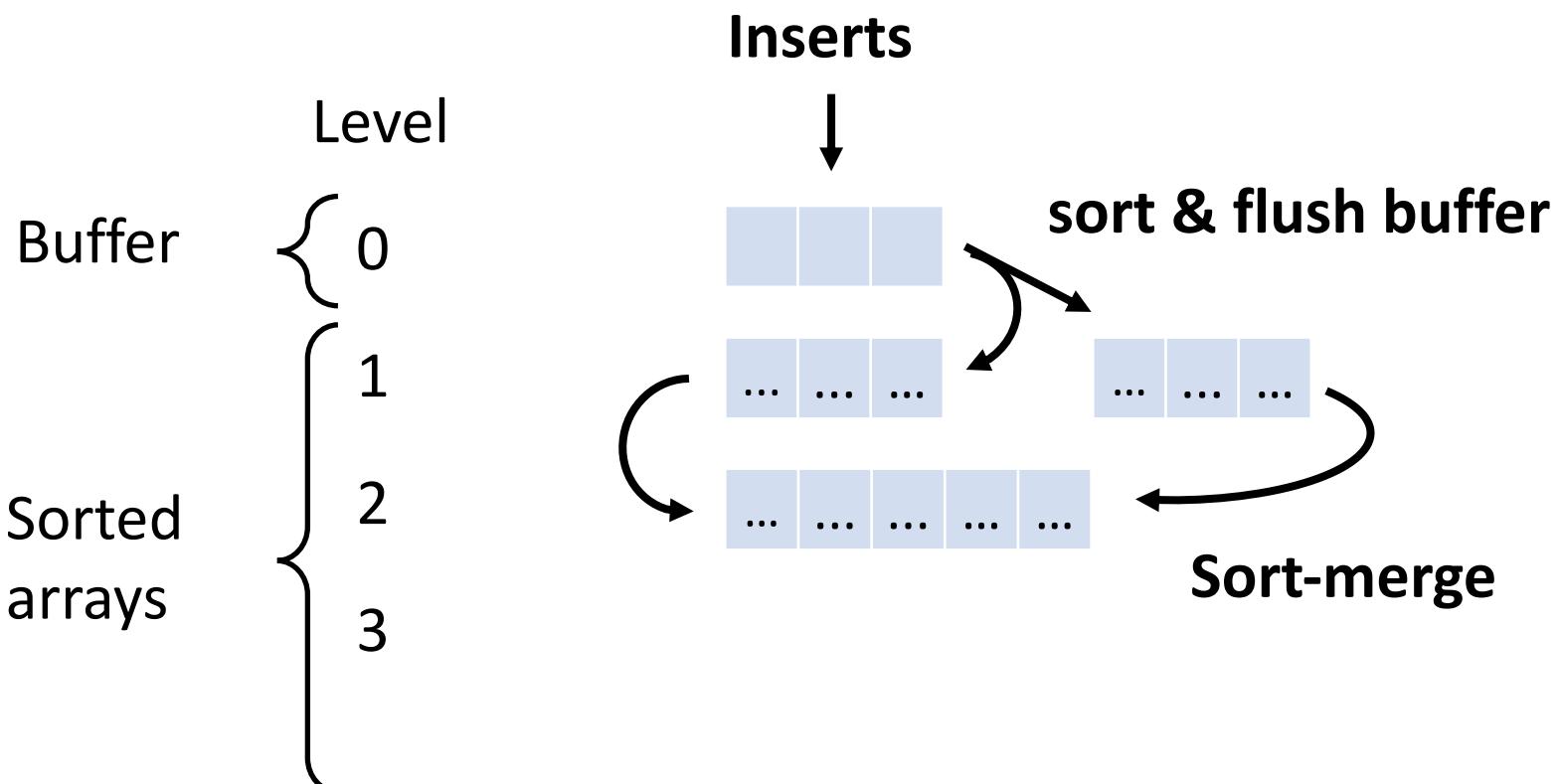
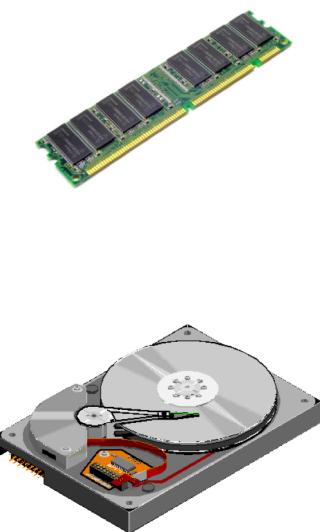


**sort & flush buffer**

# Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

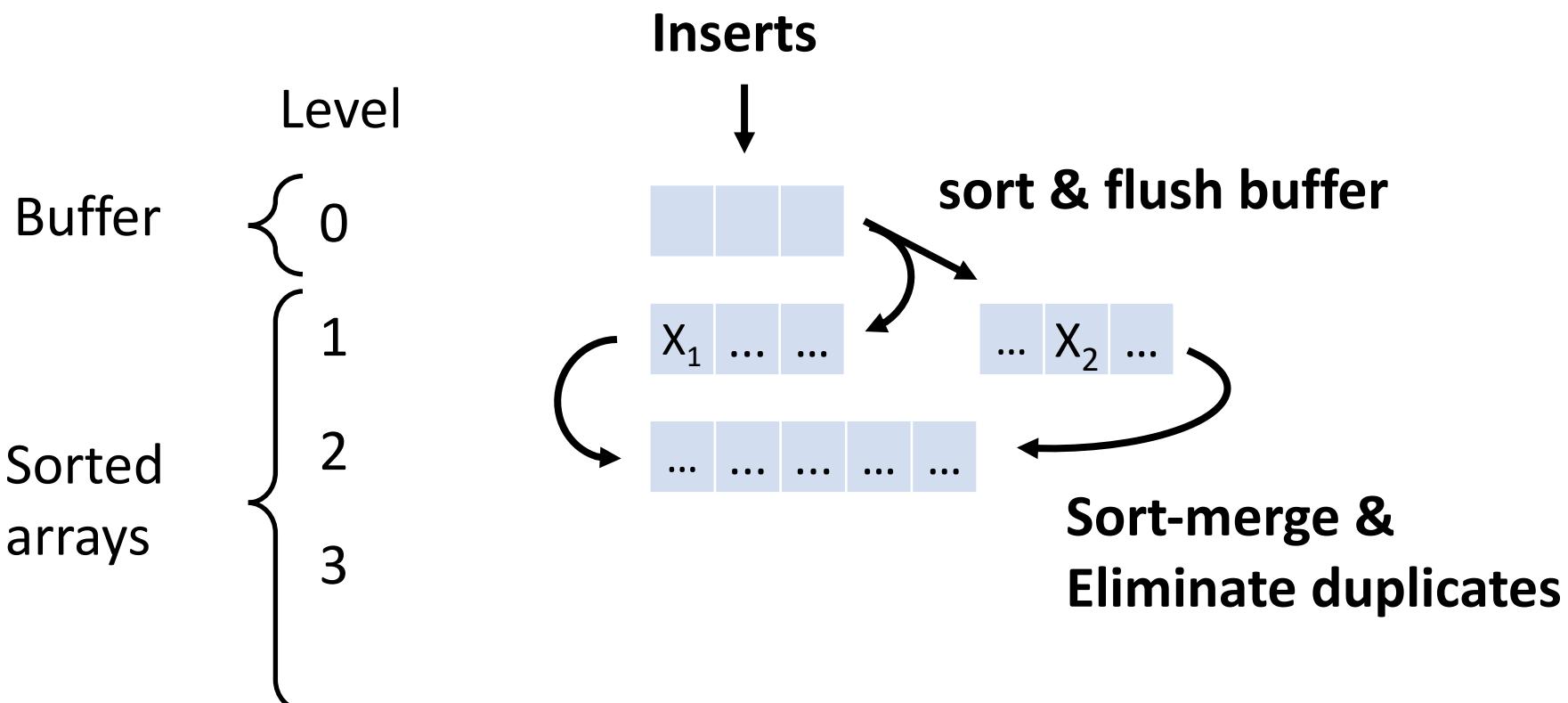
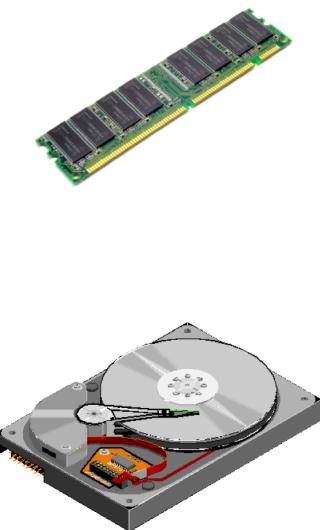
*Design principle #2:* optimize for lookups by sort-merging arrays



# Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

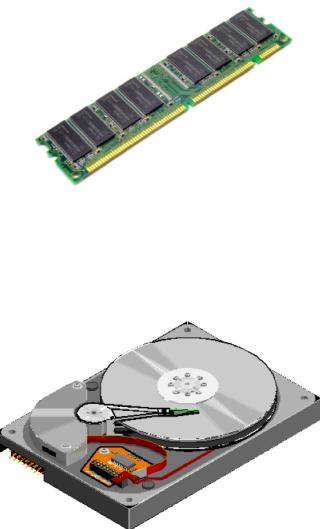
*Design principle #2:* optimize for lookups by sort-merging arrays



# Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

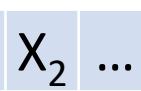
*Design principle #2:* optimize for lookups by sort-merging arrays



**Inserts**



**sort & flush buffer**

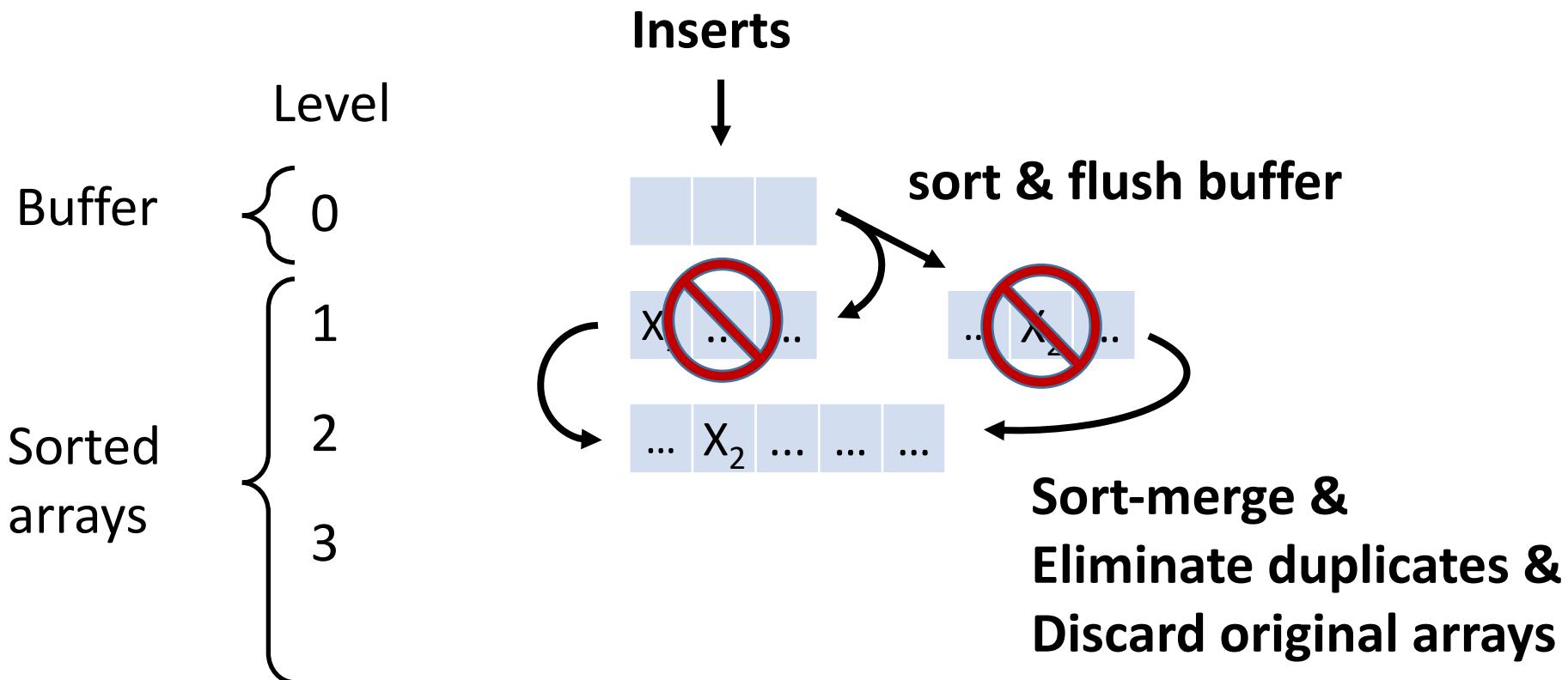
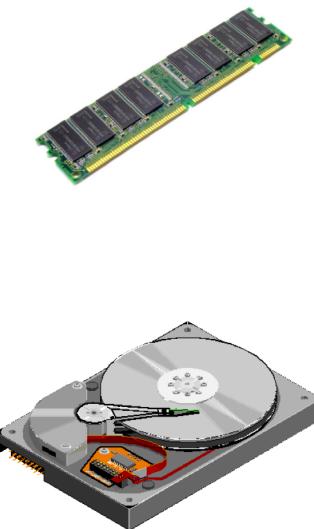


**Sort-merge &  
Eliminate duplicates**

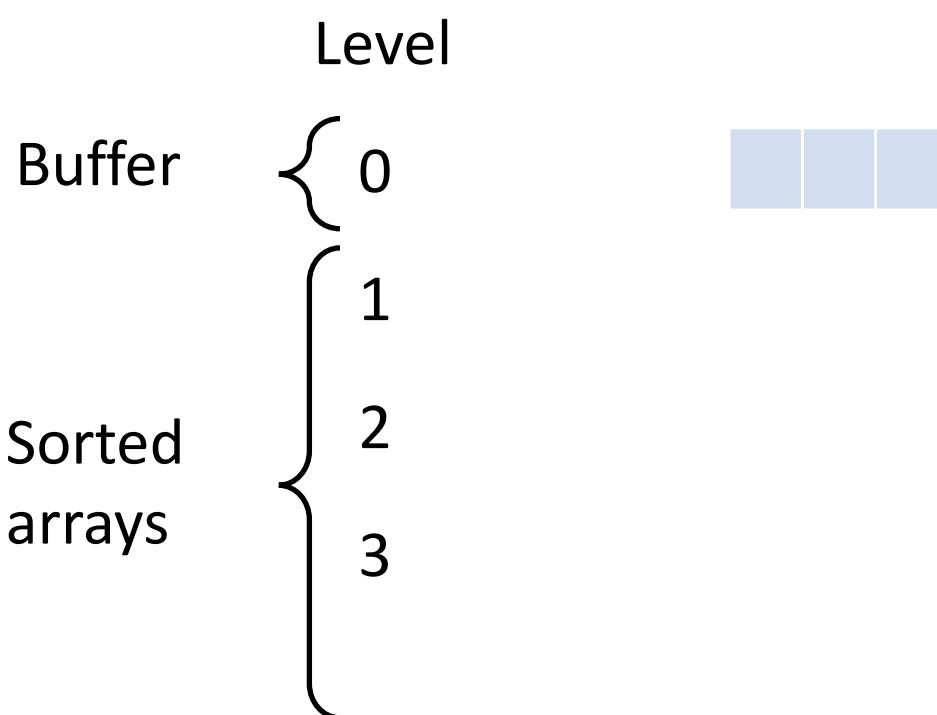
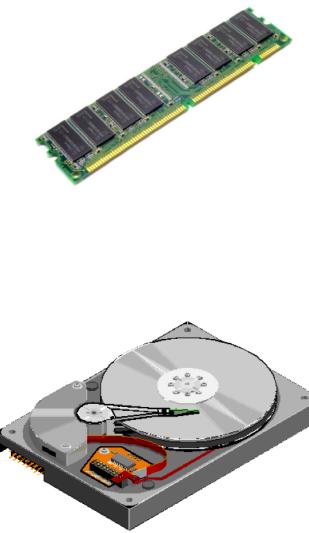
# Basic LSM-tree

*Design principle #1:* optimize for insertions by buffering

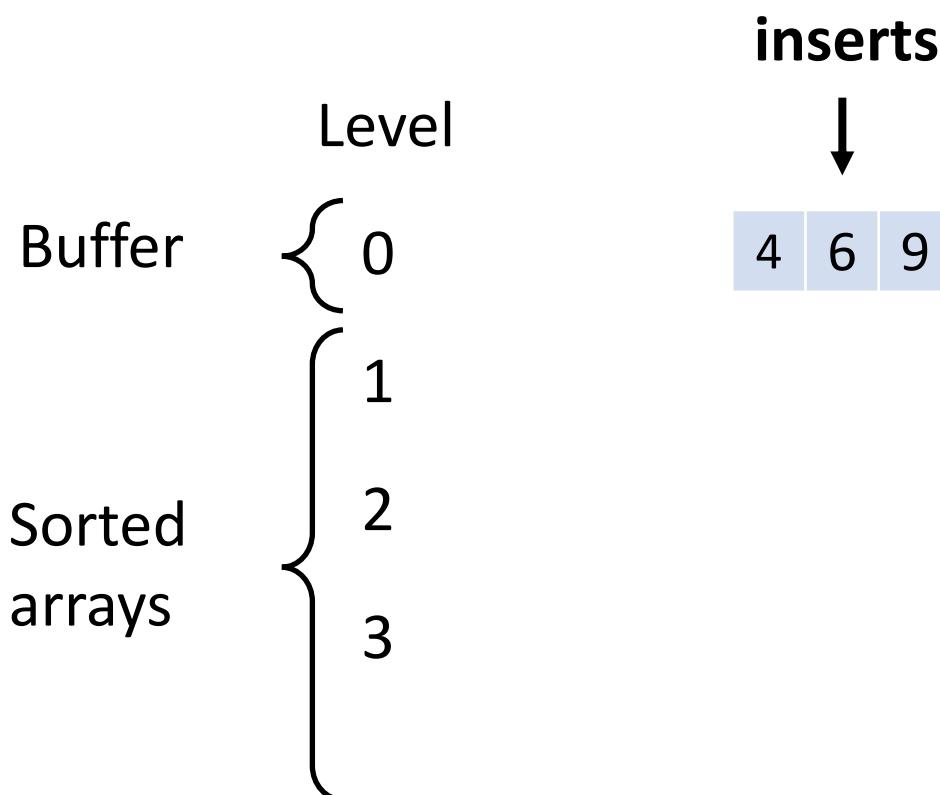
*Design principle #2:* optimize for lookups by sort-merging arrays



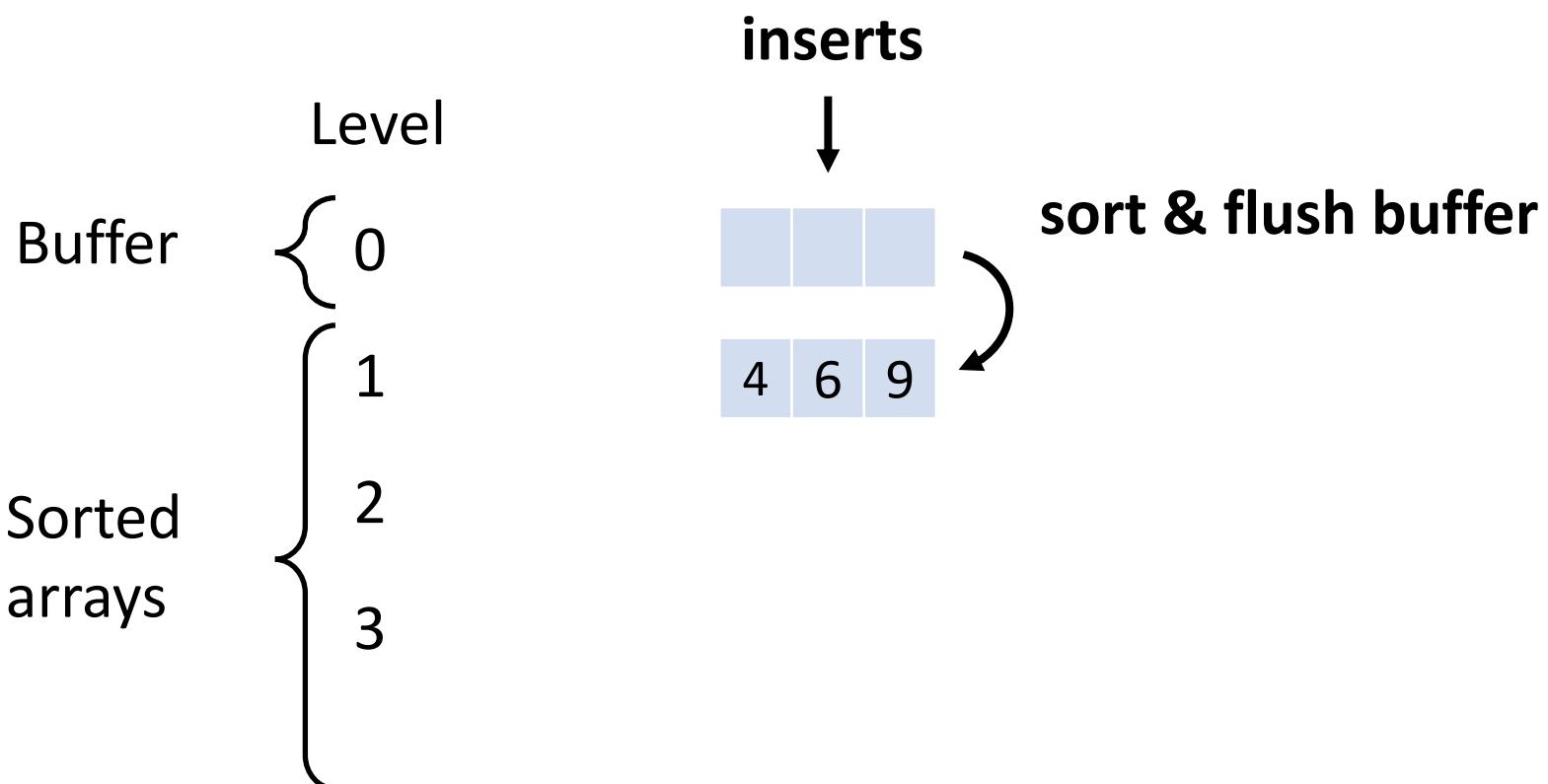
# Basic LSM-tree – Example



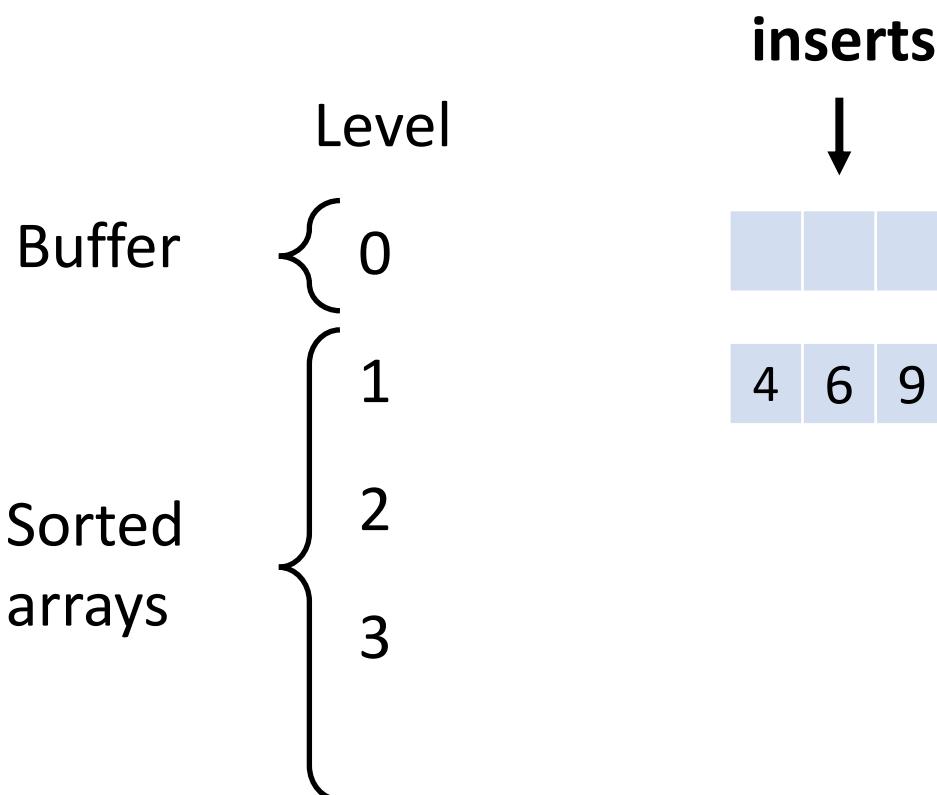
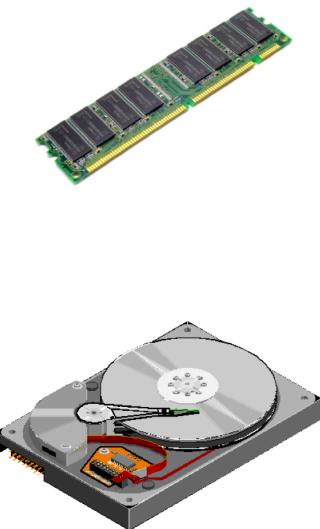
# Basic LSM-tree – Example



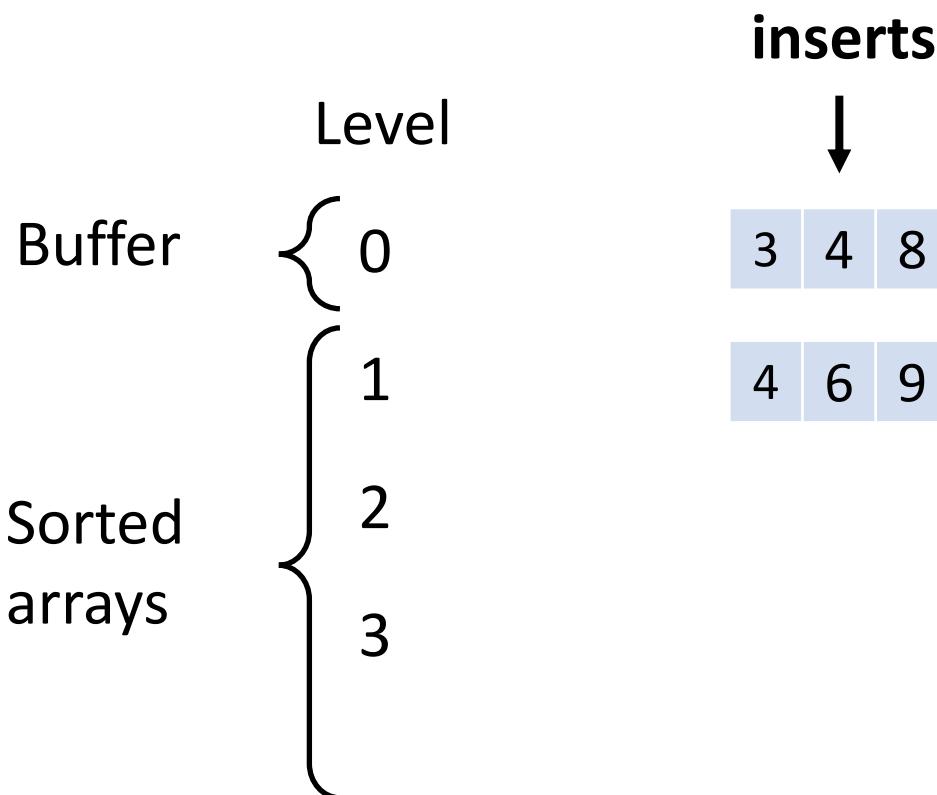
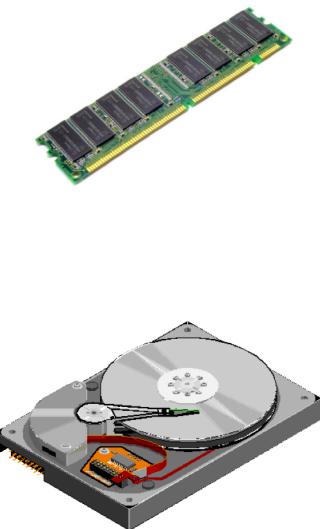
# Basic LSM-tree – Example



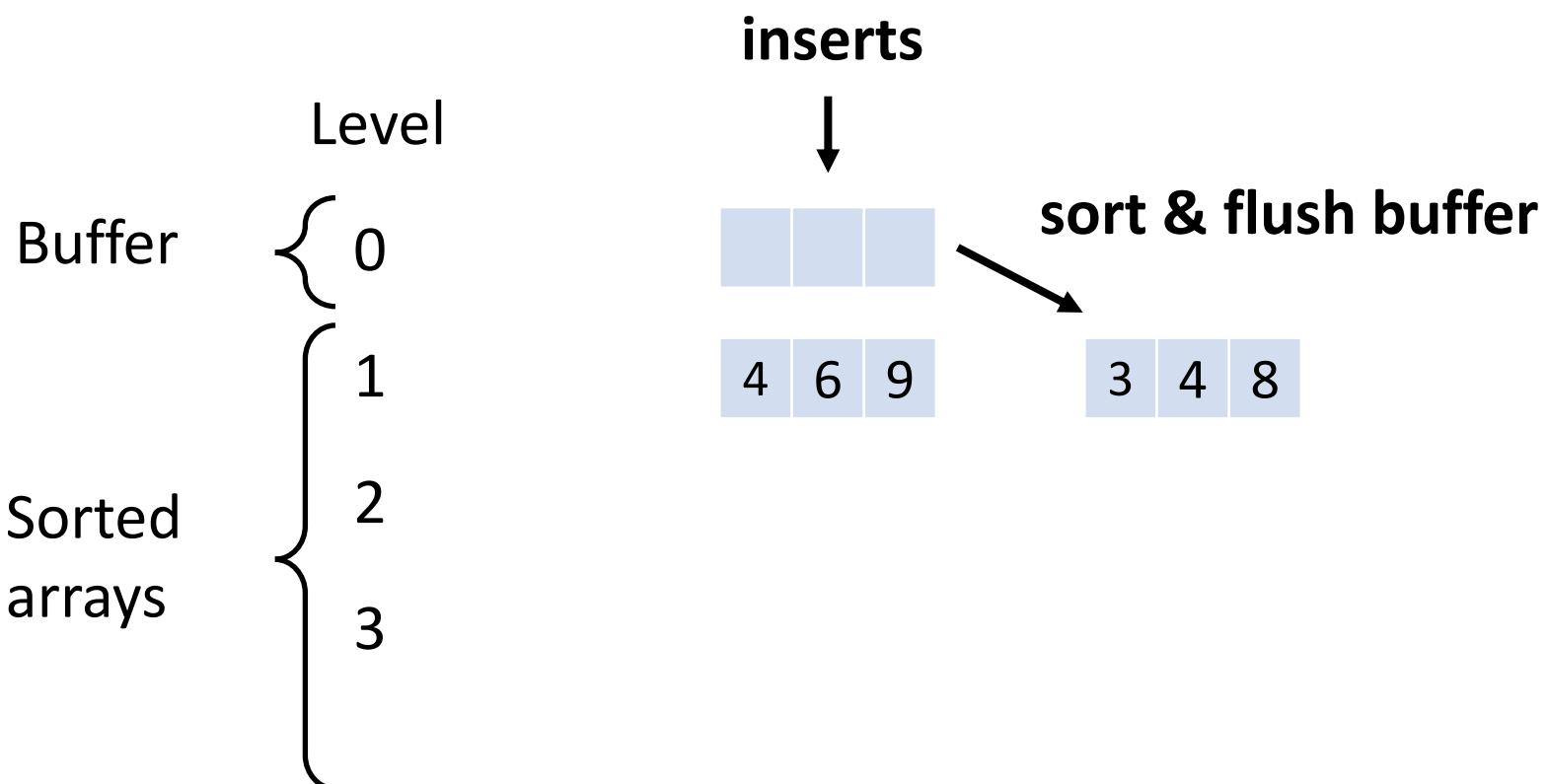
# Basic LSM-tree – Example



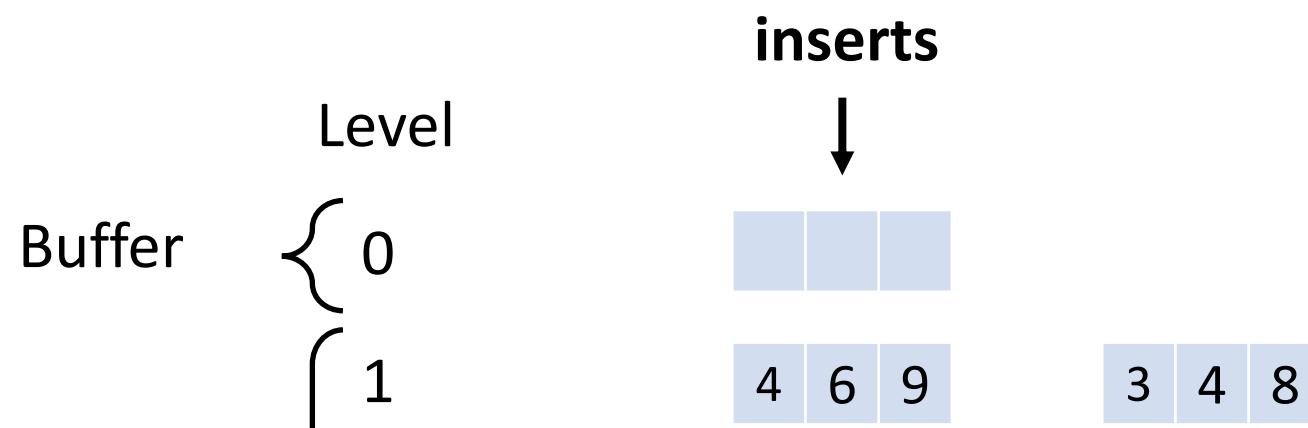
# Basic LSM-tree – Example



# Basic LSM-tree – Example

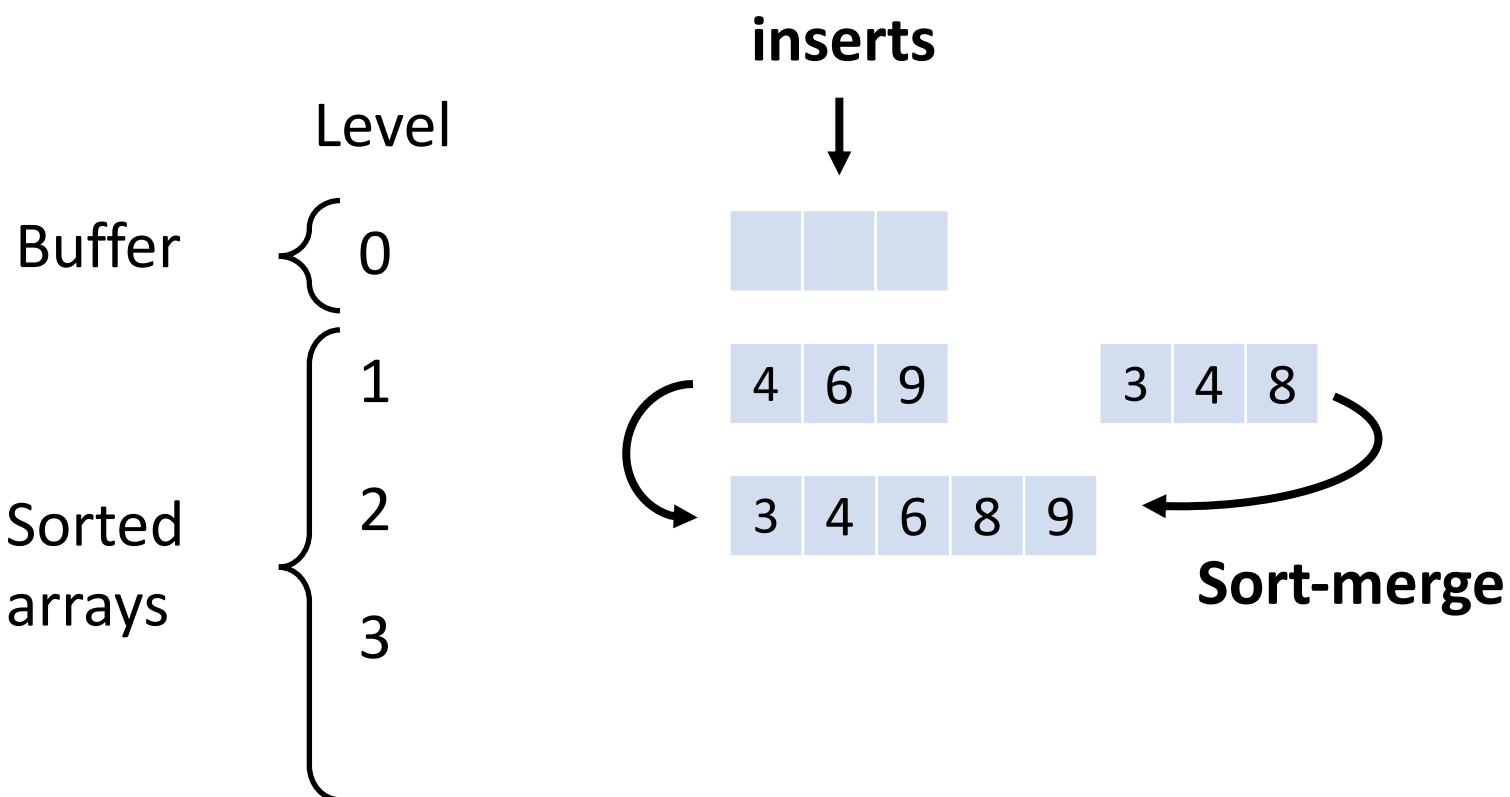
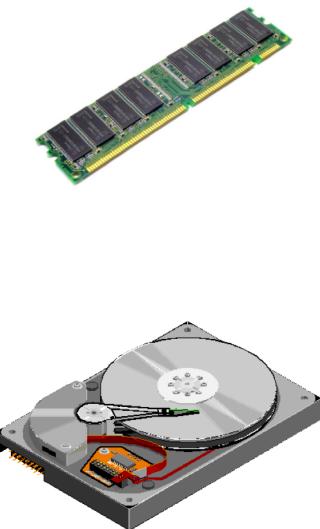


# Basic LSM-tree – Example

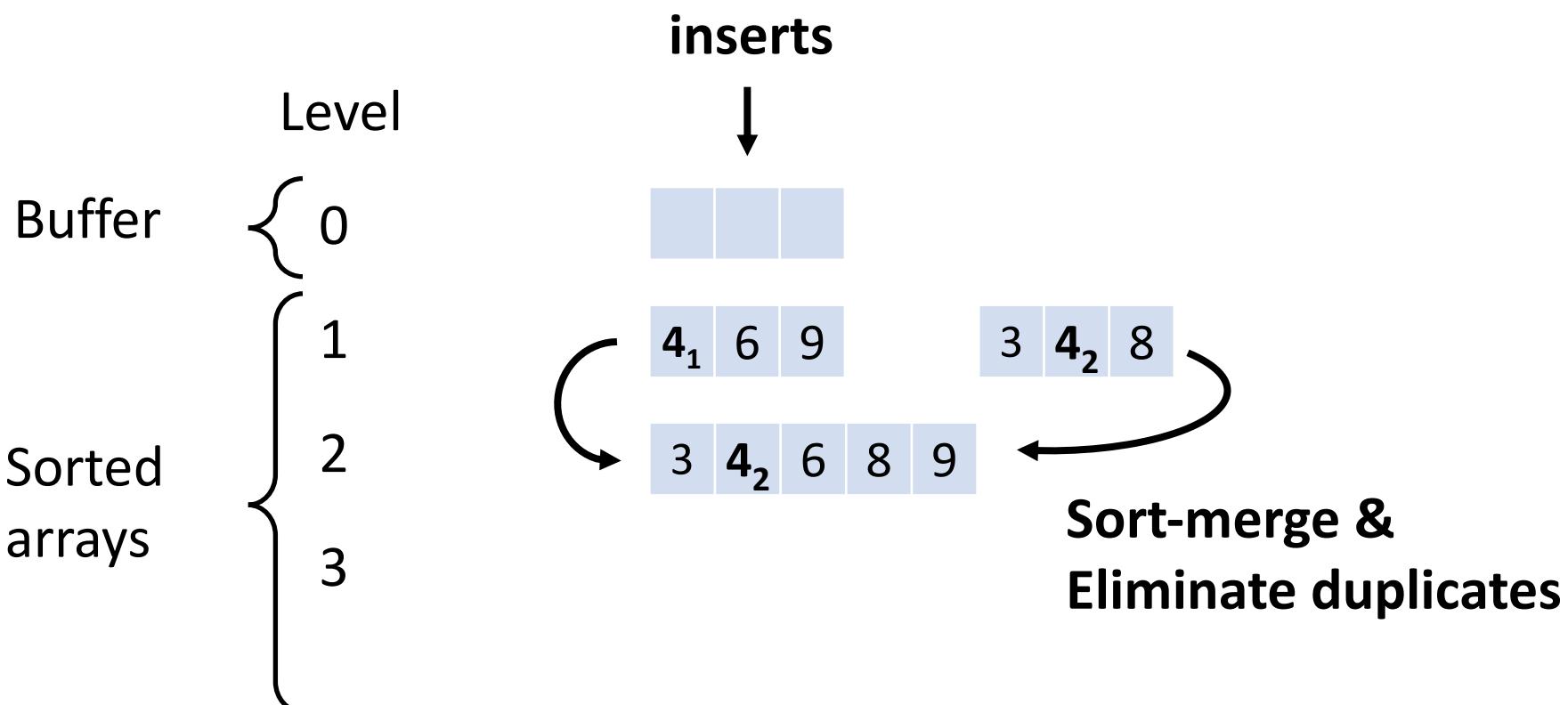
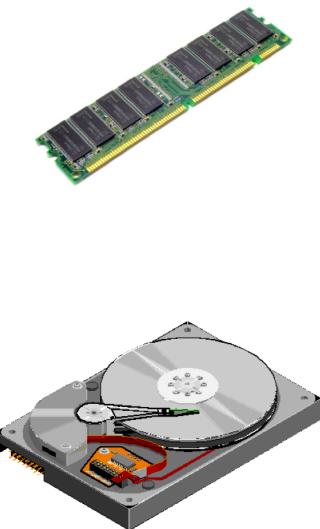


Sorted  
arrays

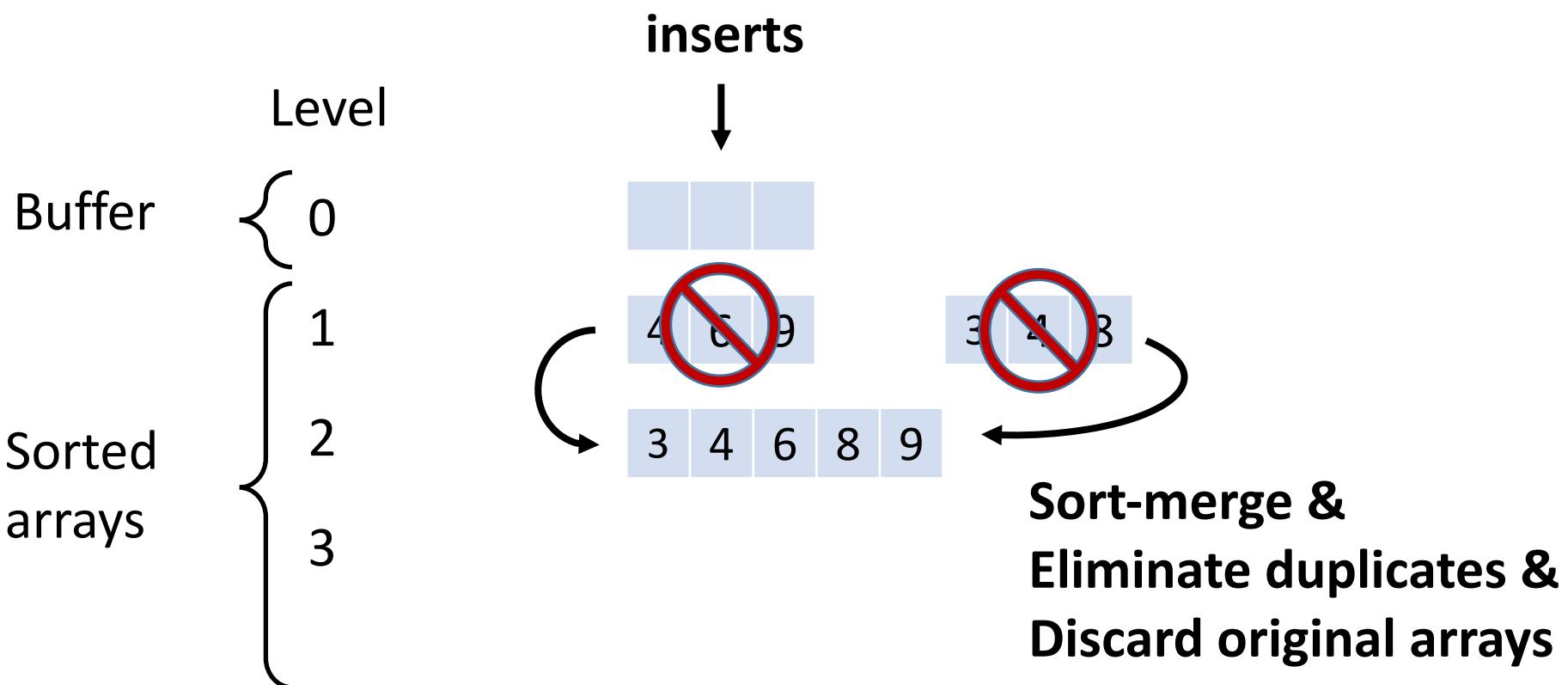
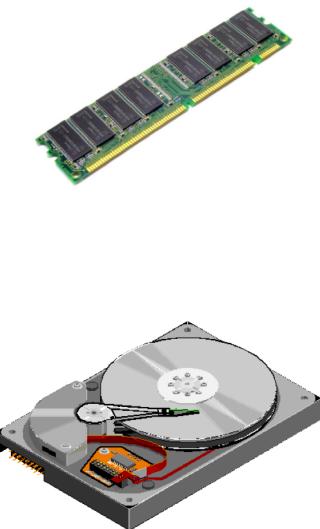
# Basic LSM-tree – Example



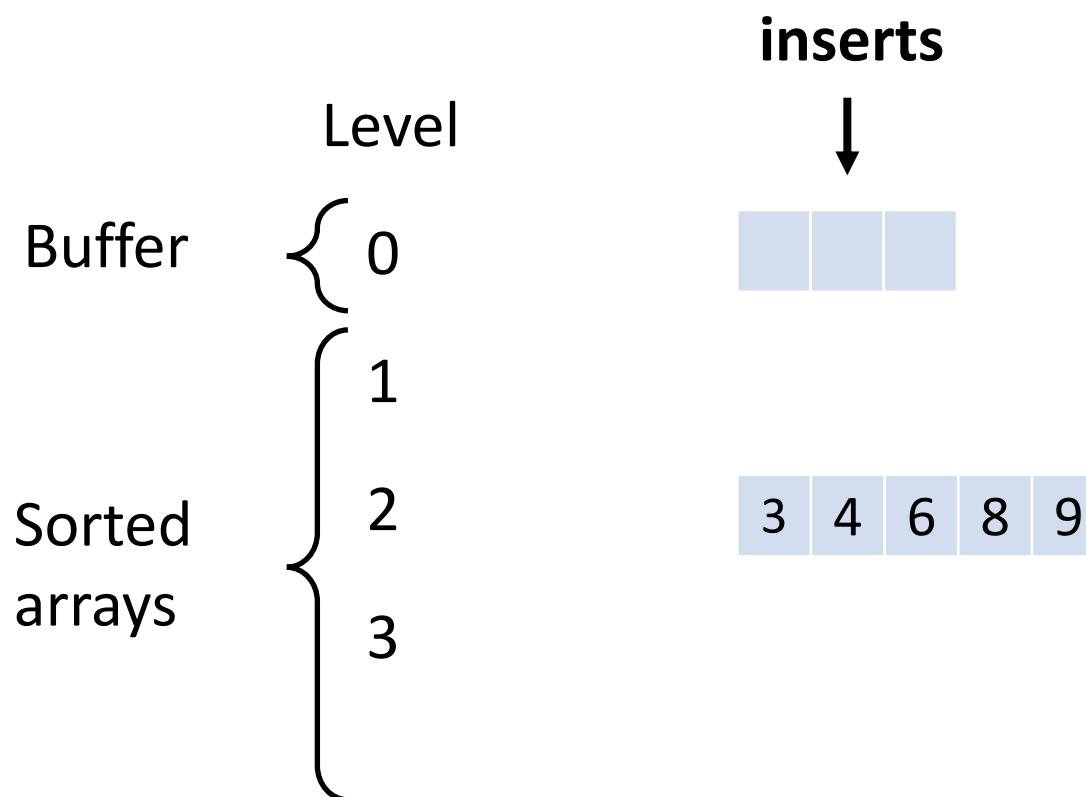
# Basic LSM-tree – Example



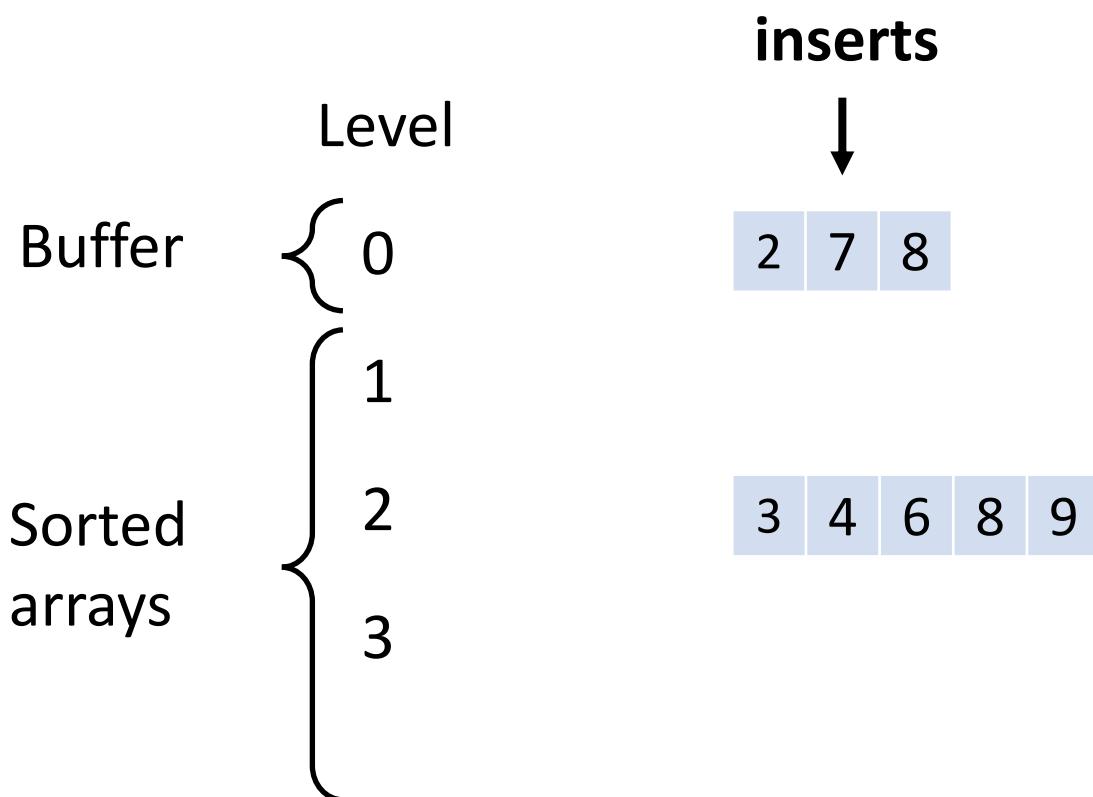
# Basic LSM-tree – Example



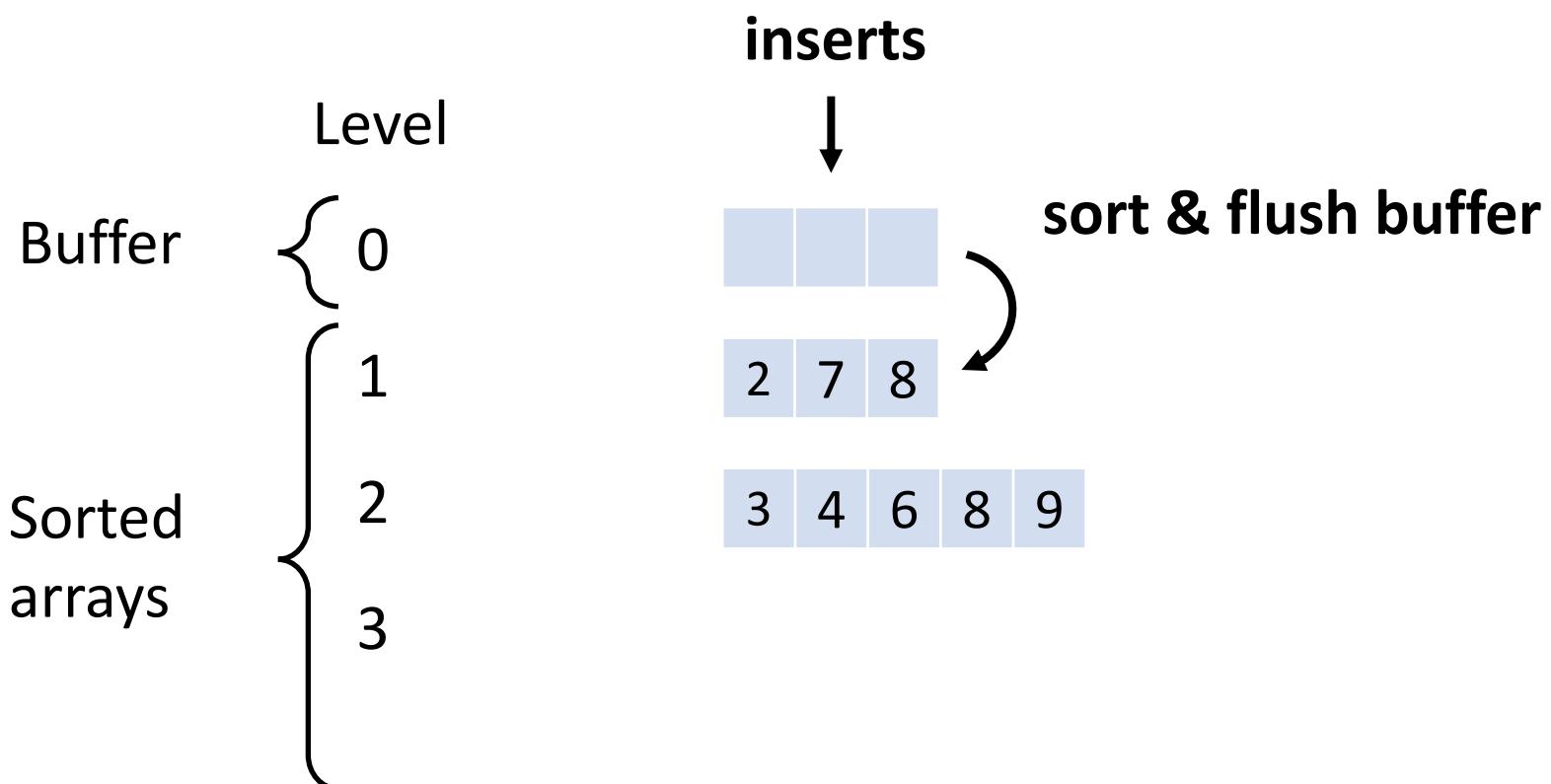
# Basic LSM-tree – Example



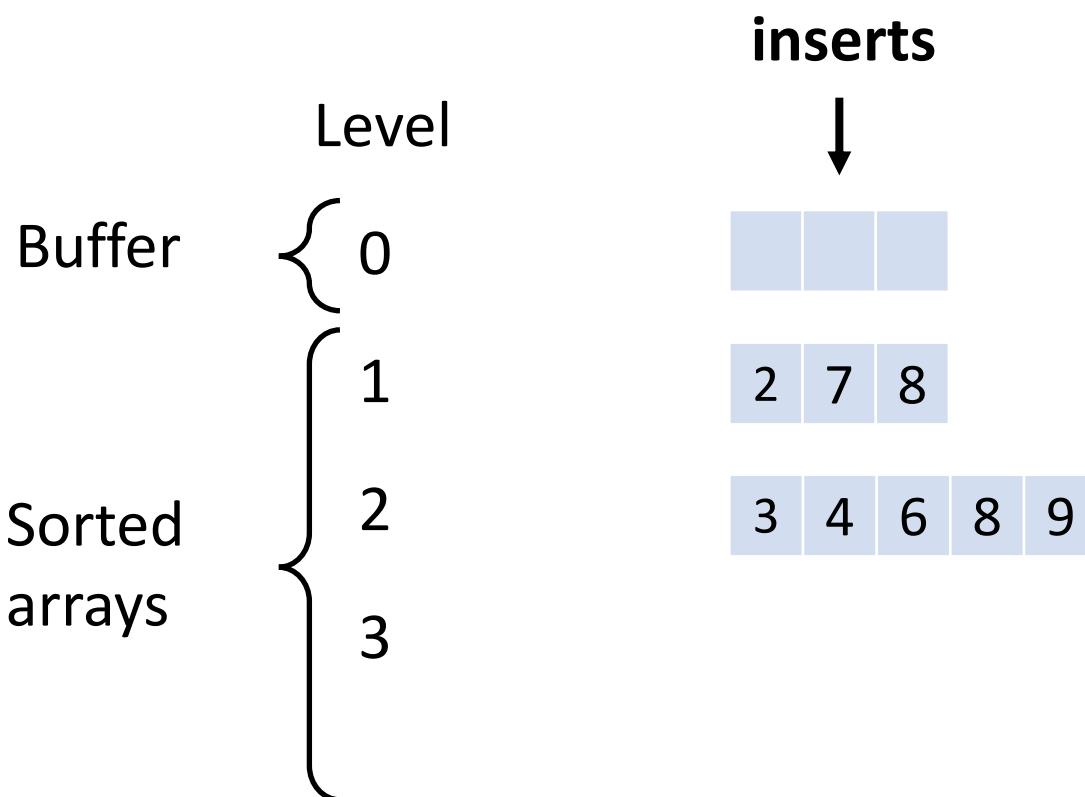
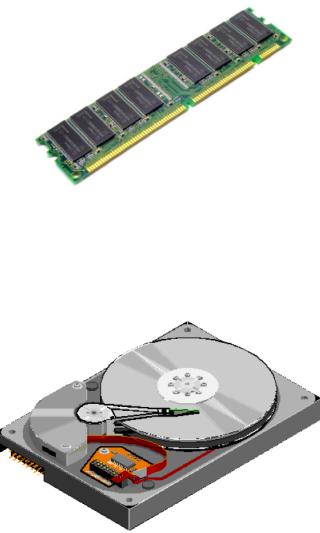
# Basic LSM-tree – Example



# Basic LSM-tree – Example

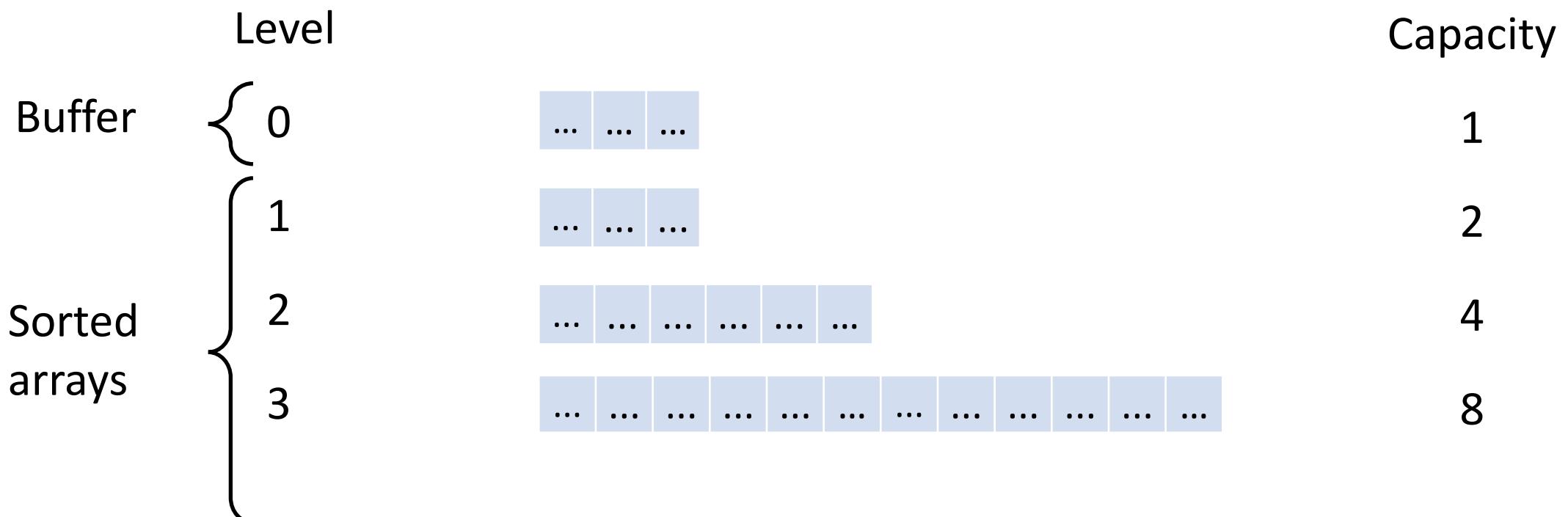
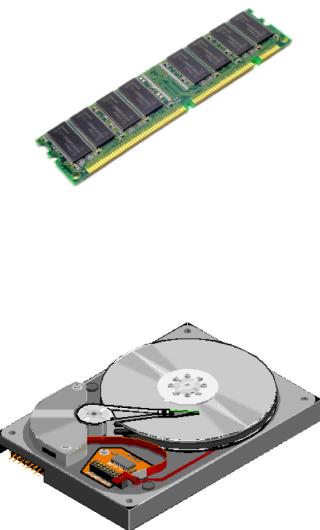


# Basic LSM-tree – Example

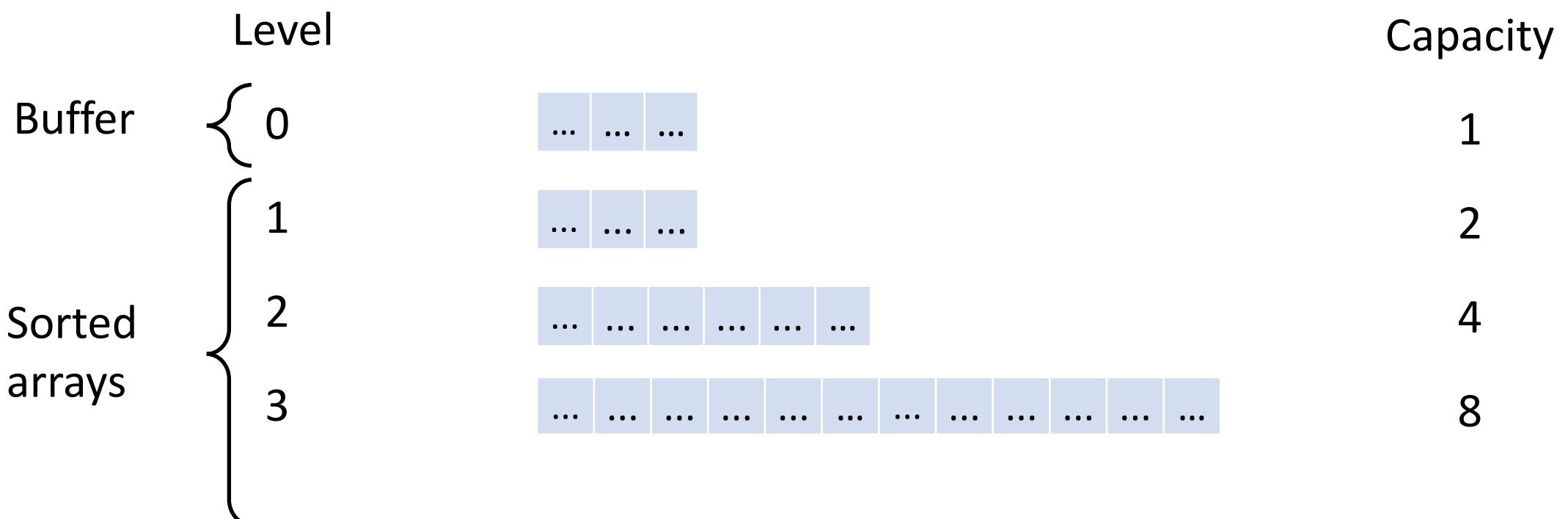
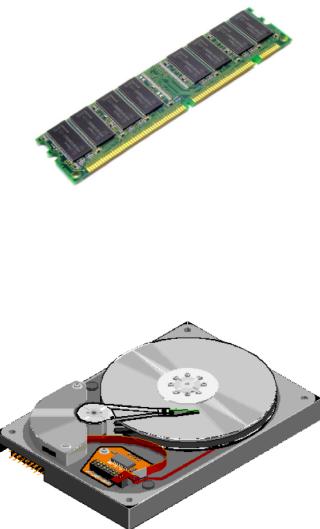


# Basic LSM-tree

Levels have exponentially increasing capacities.

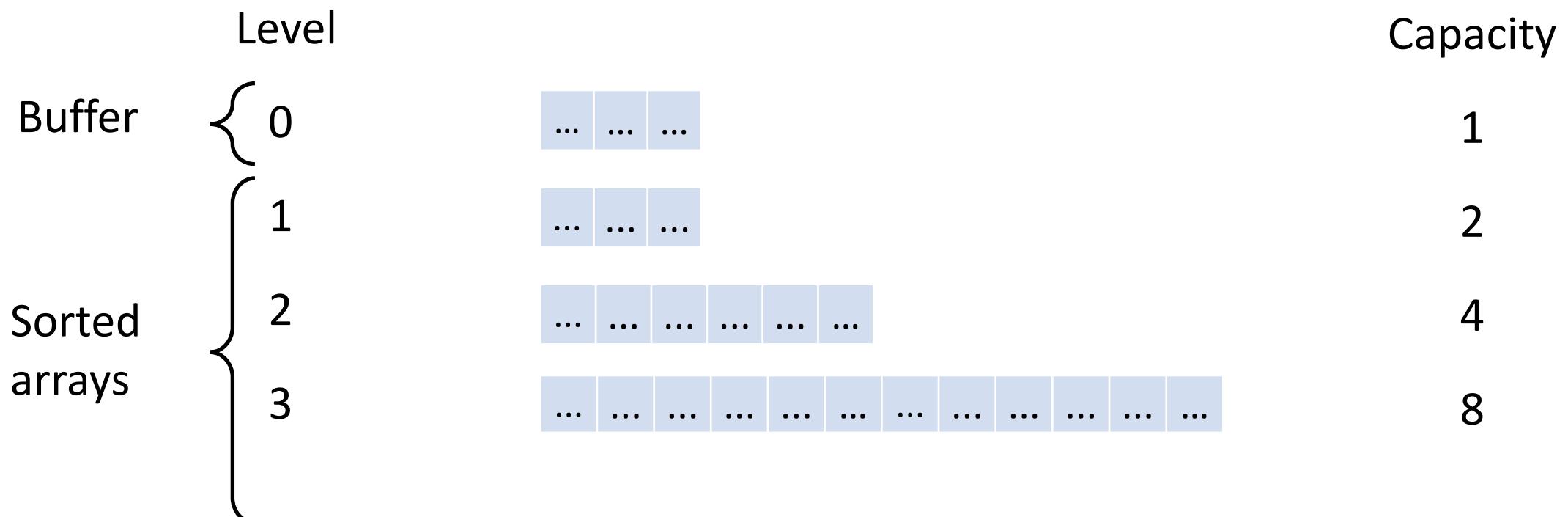
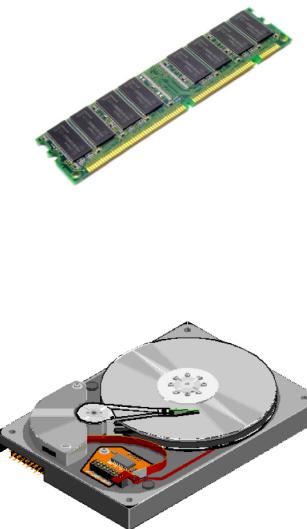


# Basic LSM-tree – Lookup cost



# Basic LSM-tree – Lookup cost

*Lookup method?*

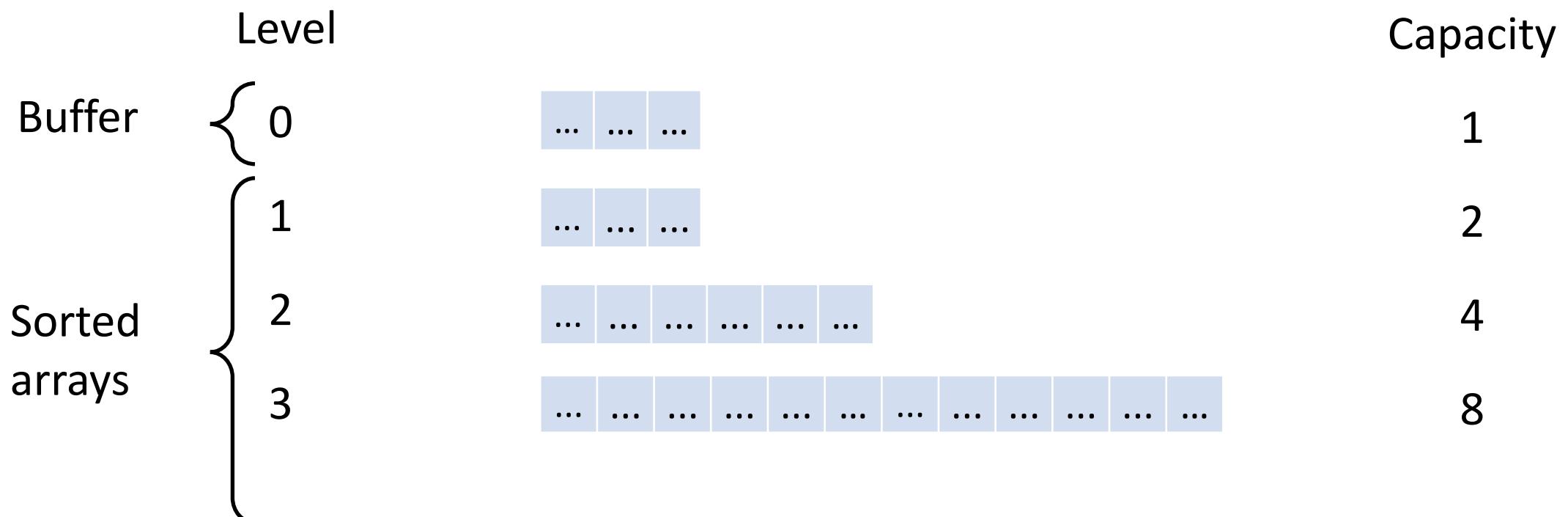
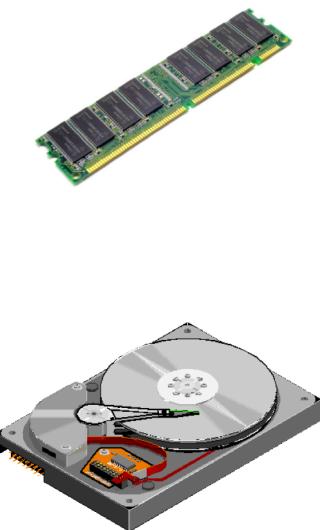


# Basic LSM-tree – Lookup cost

*Lookup method?*

Search youngest to oldest.

$$O\left(\log_2 \left(\frac{N}{B}\right)\right)$$



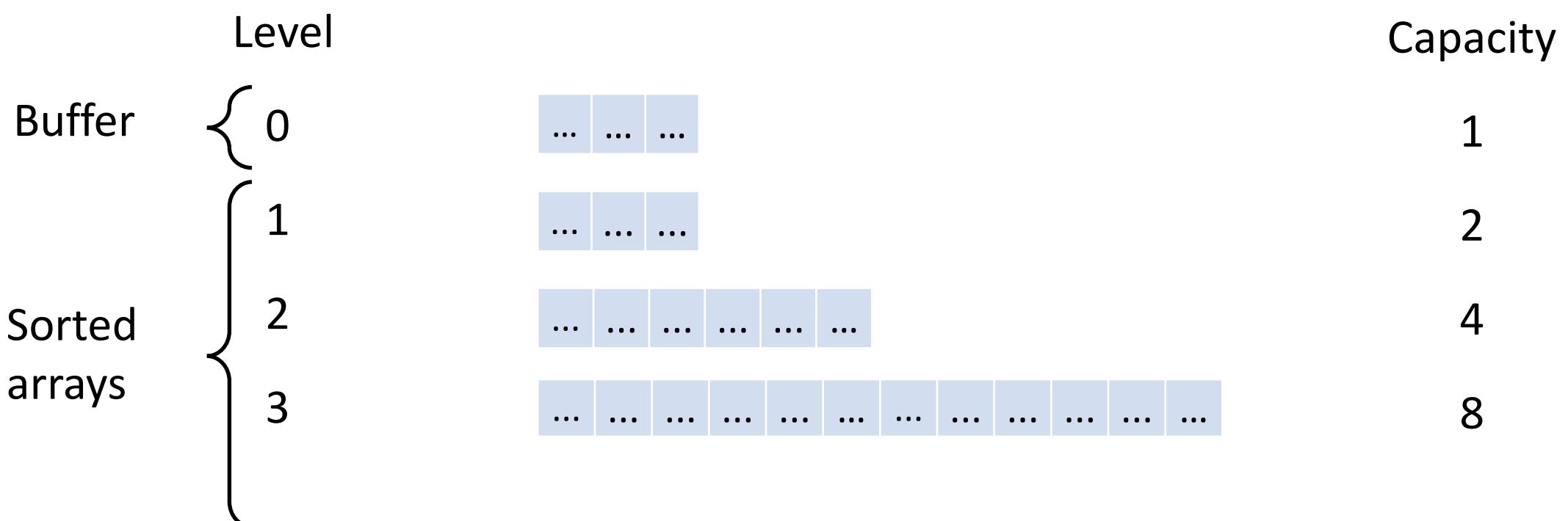
# Basic LSM-tree – Lookup cost

*Lookup method?*

Search youngest to oldest.

$$O\left(\log_2 \left(\frac{N}{B}\right)\right)$$

*How?*



# Basic LSM-tree – Lookup cost

*Lookup method?*

Search youngest to oldest.

$$O\left(\log_2 \left(\frac{N}{B}\right)\right)$$

*How?*

Binary search.

$$O\left(\log_2 \left(\frac{N}{B}\right)\right)$$



	Level	Capacity
Buffer	0	1
Sorted arrays	1	2
	2	4
	3	8

The diagram illustrates the levels of an LSM-tree. It shows four levels: Buffer (Level 0), and three levels of sorted arrays (Level 1, Level 2, and Level 3). Each level is represented by a stack of blue rectangular boxes. The Buffer level has one box. The Level 1 has two boxes. The Level 2 has four boxes. The Level 3 has eight boxes. Ellipses (...) are used to indicate that there are more boxes in each level than shown.



# Basic LSM-tree – Lookup cost

*Lookup method?*

Search youngest to oldest.

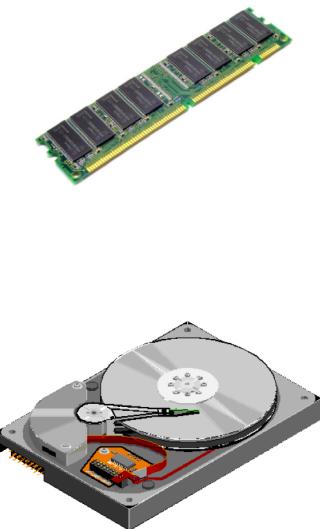
$$O\left(\log_2 \left(\frac{N}{B}\right)\right)$$

*How?*

Binary search.

$$O\left(\log_2 \left(\frac{N}{B}\right)\right)$$

*Lookup cost?*



	Level	Capacity
Buffer	0	1
Sorted arrays	1	2
	2	4
	3	8

The diagram illustrates the levels of an LSM-tree. It shows four levels: Buffer (Level 0), and Sorted arrays at Levels 1, 2, and 3. Each level is represented by a stack of blue rectangular boxes. Level 0 has 1 box, Level 1 has 2 boxes, Level 2 has 4 boxes, and Level 3 has 8 boxes. Ellipses (...) are used to indicate that there are more boxes than shown in each level.

# Basic LSM-tree – Lookup cost

*Lookup method?*

Search youngest to oldest.

$$O\left(\log_2 \left(\frac{N}{B}\right)\right)$$

*How?*

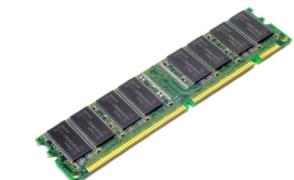
Binary search.

$$O\left(\log_2 \left(\frac{N}{B}\right)\right)$$

*Lookup cost?*

$$O\left(\log_2 \left(\frac{N}{B}\right)^2\right)$$

Capacity



Buffer  
Level 0



1

Sorted  
arrays  
Level 1



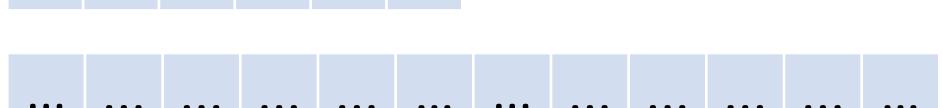
2

Sorted  
arrays  
Level 2



4

Sorted  
arrays  
Level 3



8



# Basic LSM-tree – Insertion cost

*How many times is each entry copied?*

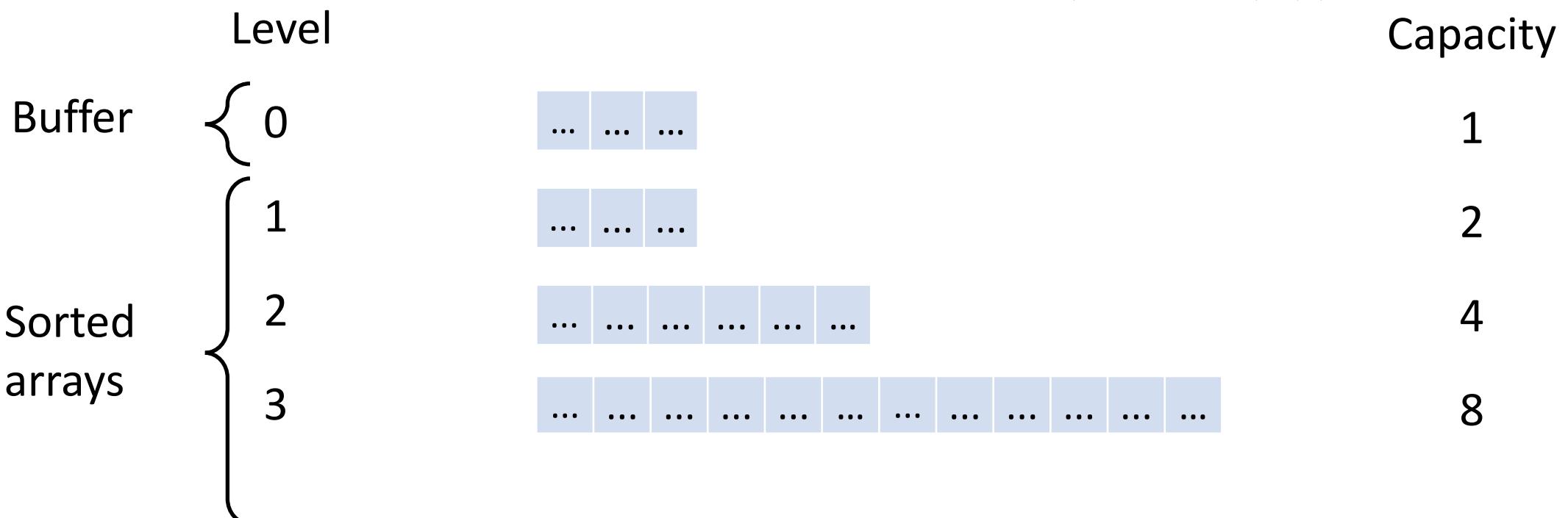
$$O\left(\log_2\left(\frac{N}{B}\right)\right)$$

*What is the price of each copy?*

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

Total insert cost?

$$O\left(\frac{1}{B} \cdot \log_2\left(\frac{N}{B}\right)\right)$$



# Results Catalogue

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

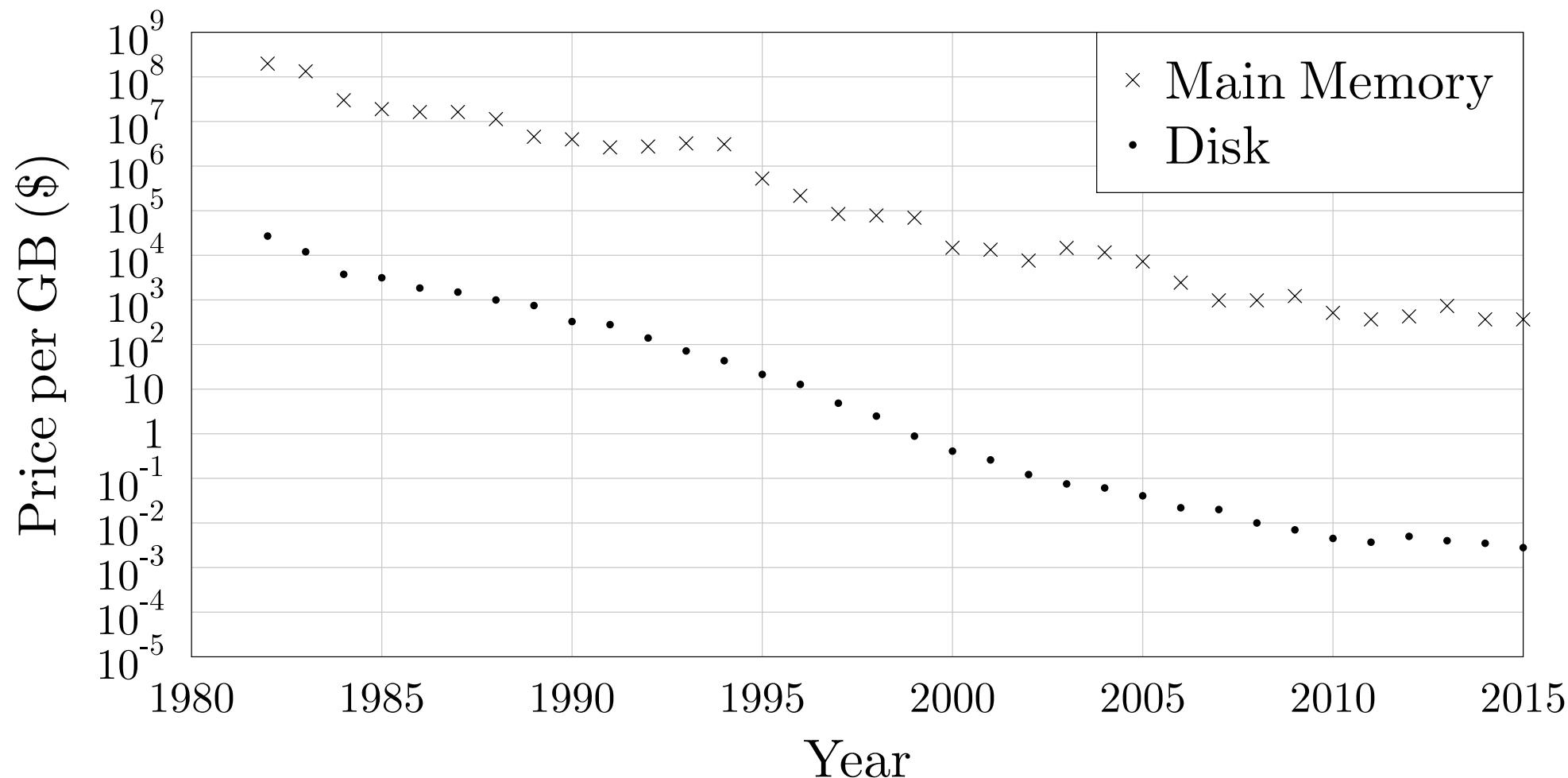
# Results Catalogue

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

Can we improve lookup cost?

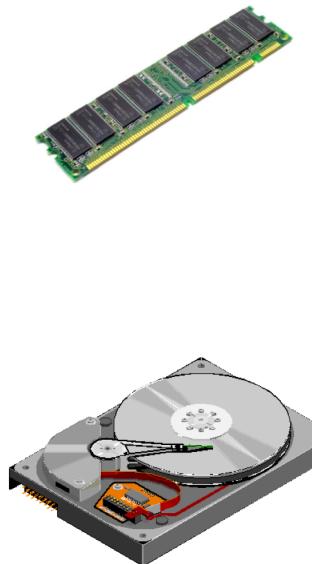
	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N/B))$	$O(N/B)$
Log	$O(N/B)$	$O(1/B)$
B-tree	$O(\log_B(N/B))$	$O(\log_B(N/B))$
<b>Basic LSM-tree</b>	$O(\log_2(N/B)^2)$	$O(1/B \cdot \log_2(N/B))$
Leveled LSM-tree		
Tiered LSM-tree		

# Declining Main Memory Cost



# Declining Main Memory Cost

Store a fence pointer for every block in main memory



Fence  
pointers {  
  
array {

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

# Results Catalogue – with fence pointers

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

# Results Catalogue – with fence pointers

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

# Results Catalogue – with fence pointers

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

# Results Catalogue – with fence pointers

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

# Results Catalogue – with fence pointers

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

# Results Catalogue – with fence pointers

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

# Results Catalogue – with fence pointers

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

# Results Catalogue – with fence pointers

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

# Results Catalogue – with fence pointers

Quick sanity check:

suppose  
and

$N = 2^{42}$   
 $B = 2^{10}$

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

# Results Catalogue – with fence pointers

Quick sanity check:

suppose  
and

$N = 2^{42}$   
 $B = 2^{10}$

	Lookup cost	Insertion cost
Sorted array	$O(1)$	$O(2^{32})$
Log	$O(2^{32})$	$O(2^{-10})$
B-tree	$O(1)$	$O(1)$
<b>Basic LSM-tree</b>	$O(5)$	$O(2^{-10} \cdot 5)$
Leveled LSM-tree		
Tiered LSM-tree		

# Leveled LSM-tree



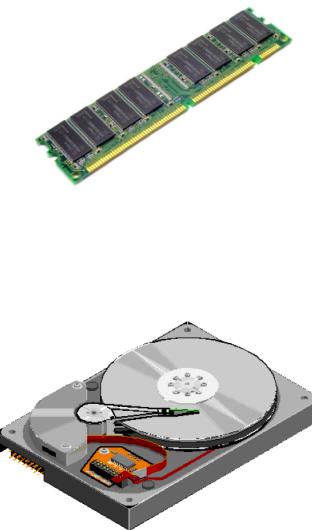
Lookup cost



Update cost

# Leveled LSM-tree

Lookup cost depends on number of levels



Level

Buffer

0

...

Sorted  
arrays

1

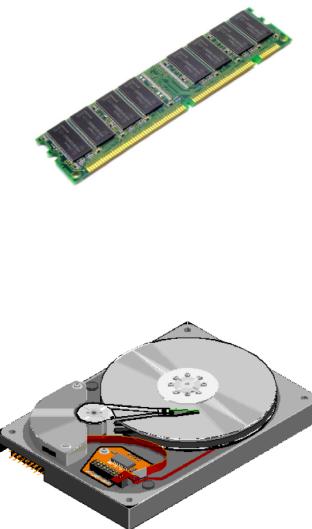
2

3

# Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

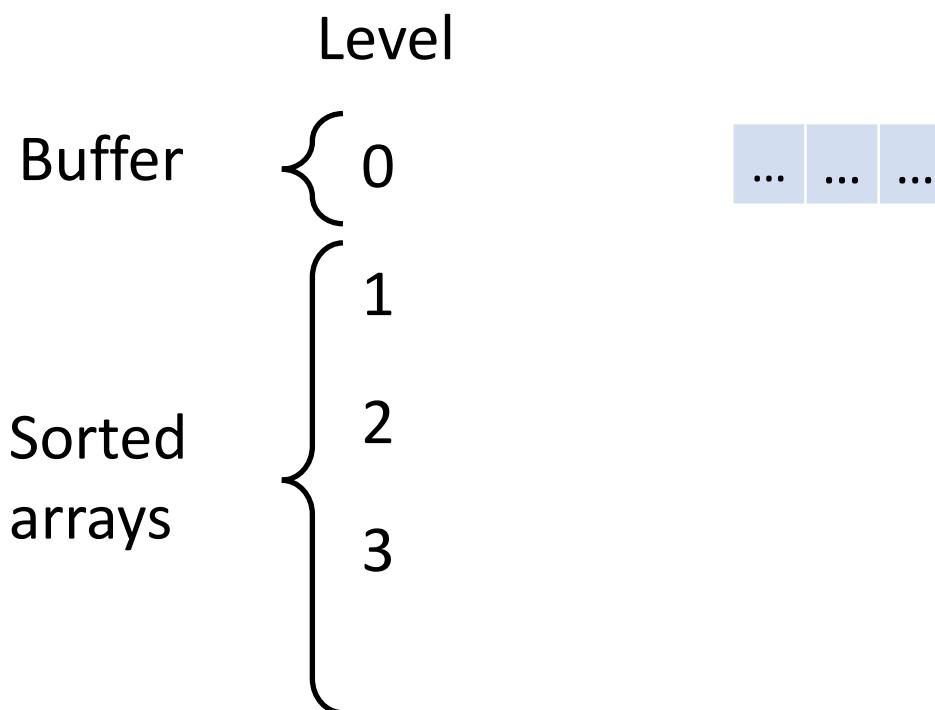
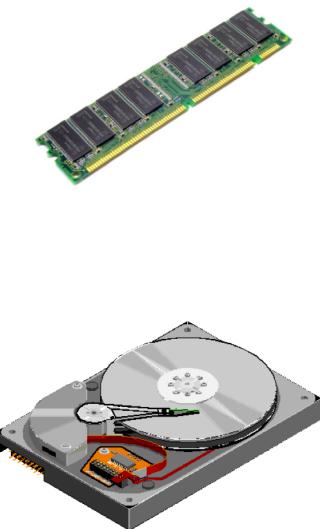


# Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

Increase size ratio T

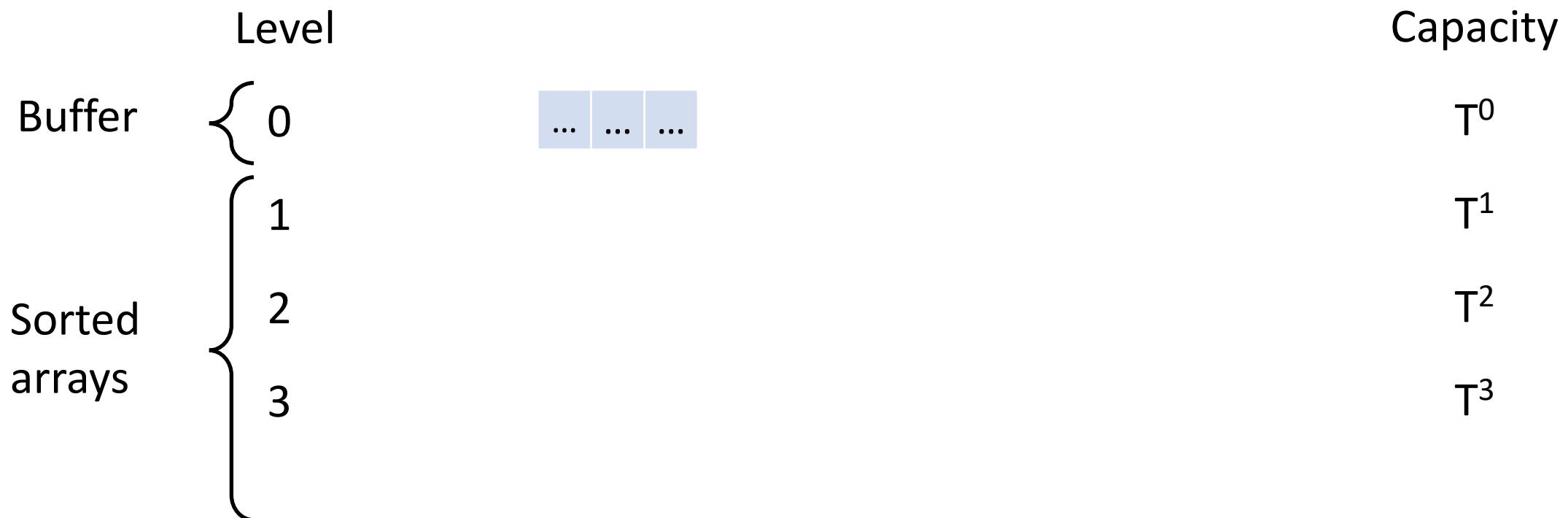
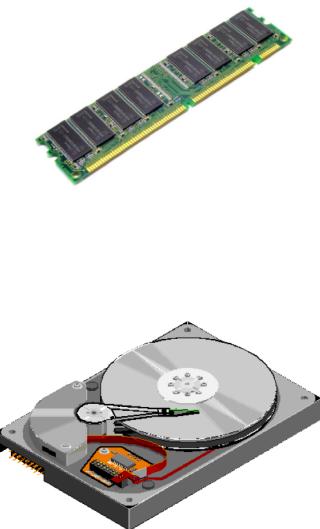


# Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

Increase size ratio T



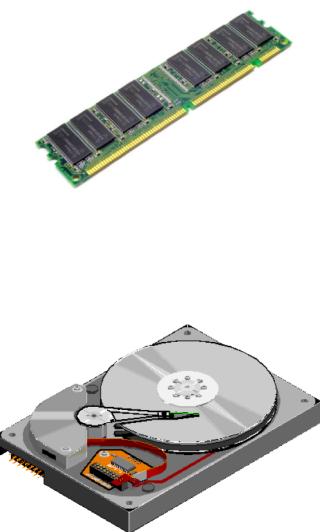
# Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

E.g. size ratio of 4

Increase size ratio T



	Level	Capacity
Buffer	0	1
Sorted arrays	1	4
	2	16
	3	64

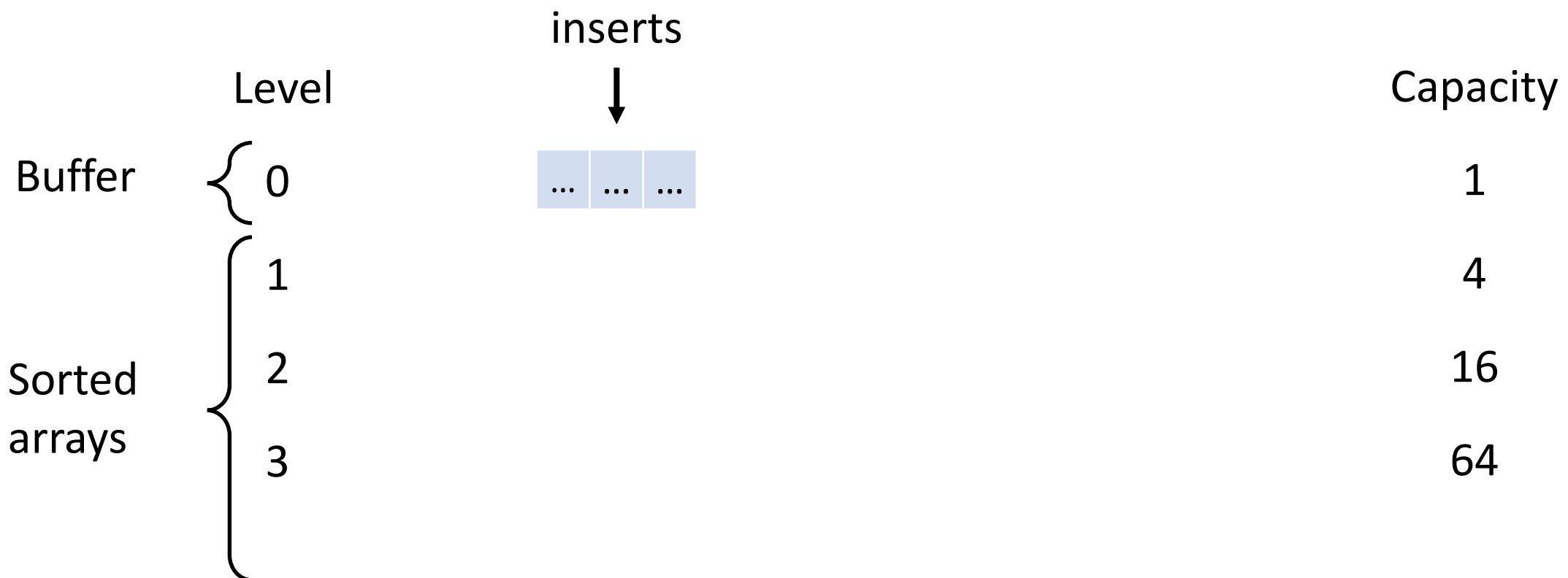
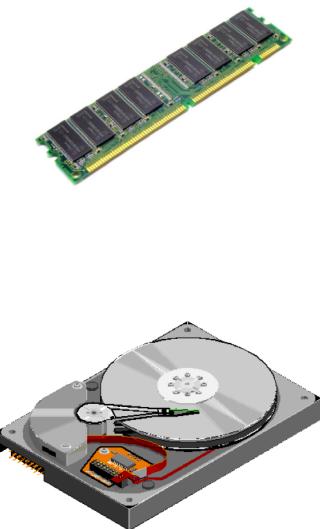
# Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

E.g. size ratio of 4

Increase size ratio T



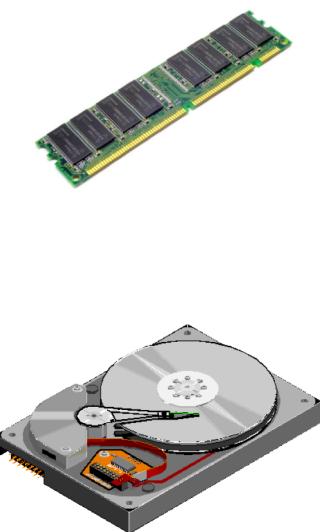
# Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

E.g. size ratio of 4

Increase size ratio T



Level	Capacity
Buffer	1
Sorted arrays	4
2	16
3	64

The table shows a leveled LSM-tree structure with four levels. The first level is a buffer with a capacity of 1, represented by a single RAM module icon. The second level is sorted arrays with a capacity of 4, represented by a hard disk drive icon. The third level has a capacity of 16, and the fourth level has a capacity of 64. An arrow labeled "inserts" points down from the top of the table towards the sorted arrays level. A curved arrow labeled "flush" points from the sorted arrays level back up towards the buffer level, indicating the flow of data between memory and storage.

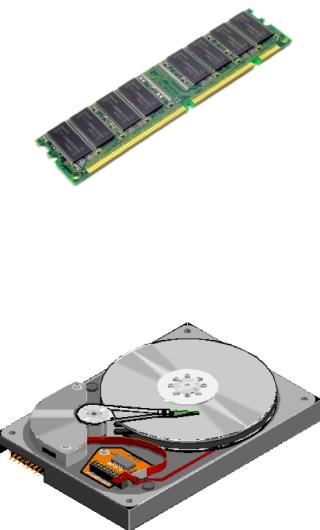
# Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

E.g. size ratio of 4

Increase size ratio T



	Level	Capacity
Buffer	0	1
Sorted arrays	1	4
	2	16
	3	64

The table shows a leveled LSM-tree structure with four levels. The first level (Level 0) is a Buffer with a capacity of 1. The second level (Level 1) is a Sorted arrays with a capacity of 4. The third level (Level 2) has a capacity of 16, and the fourth level (Level 3) has a capacity of 64. The table includes a column for 'Level' and a column for 'Capacity'. A curly brace on the left indicates the levels from Buffer to Level 3. A curly brace on the right indicates the levels from Level 0 to Level 3. An arrow labeled 'flush & sort-merge' points from the Buffer level to the Sorted arrays level. Above the table, there is a downward-pointing arrow labeled 'inserts' pointing from the Buffer level to the Sorted arrays level. To the left of the table, there is a RAM module icon and a hard disk drive icon.

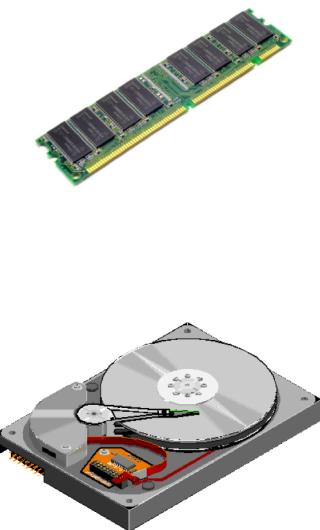
# Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

E.g. size ratio of 4

Increase size ratio T



inserts



flush & sort-merge

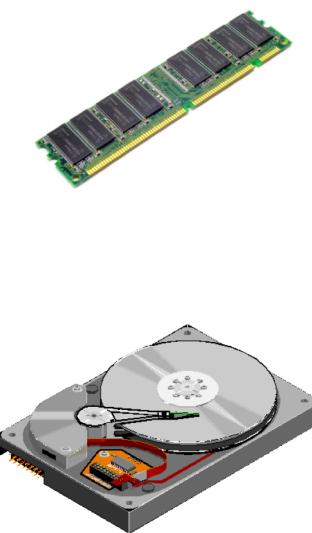
# Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

E.g. size ratio of 4

Increase size ratio T



	Level	Capacity
Buffer	0	1
Sorted arrays	1	4
	2	16
	3	64

Diagram illustrating the Leveled LSM-tree structure:

- The structure consists of four levels: Buffer (Level 0), Sorted arrays (Level 1), Level 2, and Level 3.
- Each level has a capacity: Level 0 has capacity 1, Level 1 has capacity 4, Level 2 has capacity 16, and Level 3 has capacity 64.
- Insertions (indicated by a downward arrow) occur at the Buffer level.
- After insertions, a process labeled "flush & sort-merge" (indicated by a curved arrow) moves data from the Buffer level to the Sorted arrays level.
- The Sorted arrays level contains multiple sorted arrays, each represented by a sequence of three blue boxes with ellipses (...).

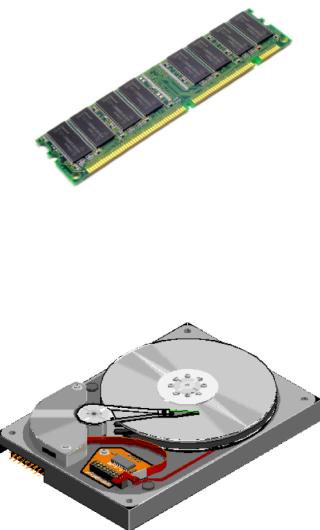
# Leveled LSM-tree

Lookup cost depends on number of levels

How to reduce it?

E.g. size ratio of 4

Increase size ratio T



	Level	Capacity
Buffer	0	1
Sorted arrays	1	4
	2	16
	3	64

The table illustrates a leveled LSM-tree structure. It shows four levels: Buffer (Level 0), Sorted arrays (Level 1), Level 2, and Level 3. The capacity of each level increases exponentially: Level 0 has a capacity of 1, Level 1 has 4, Level 2 has 16, and Level 3 has 64. A vertical brace on the left indicates the grouping of these levels. An arrow labeled "inserts" points downwards from the top of the table towards the first three levels (Buffer, Sorted arrays, and Level 1). A curved arrow labeled "move" points from the end of the Level 1 array towards the start of the Level 2 array, indicating the movement of data between levels during an insertion operation.

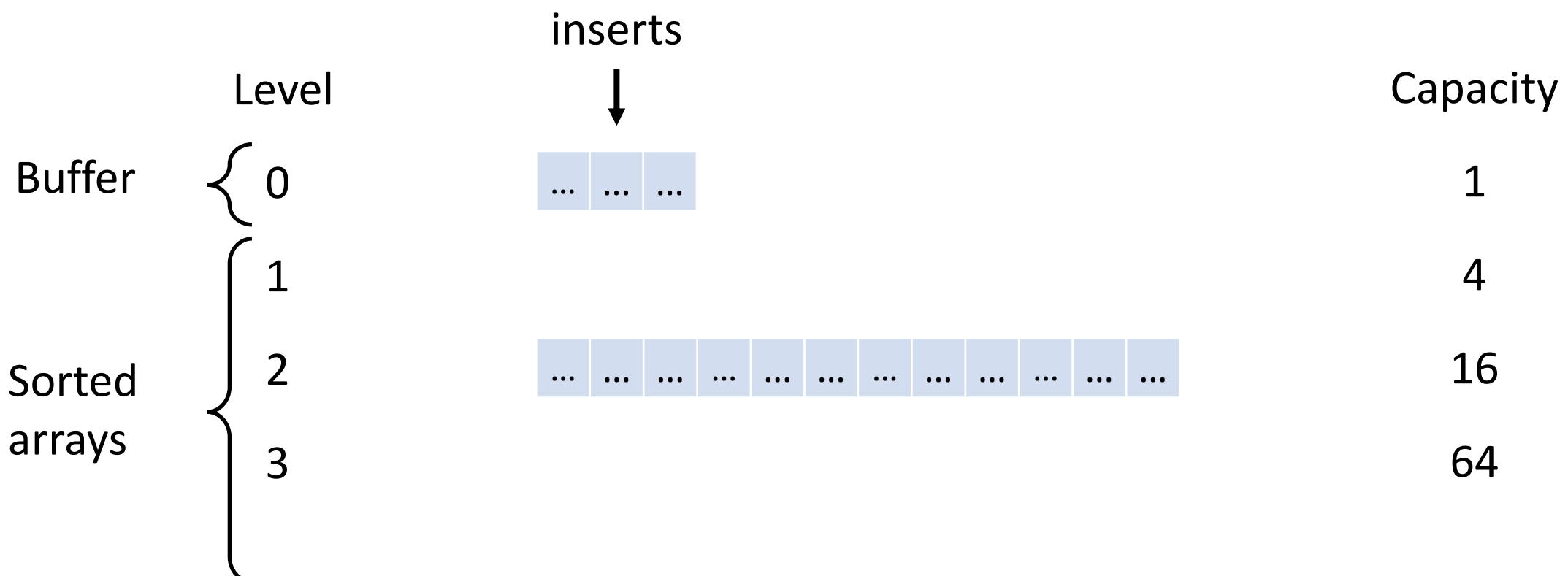
# Leveled LSM-tree

Lookup cost depends on number of levels

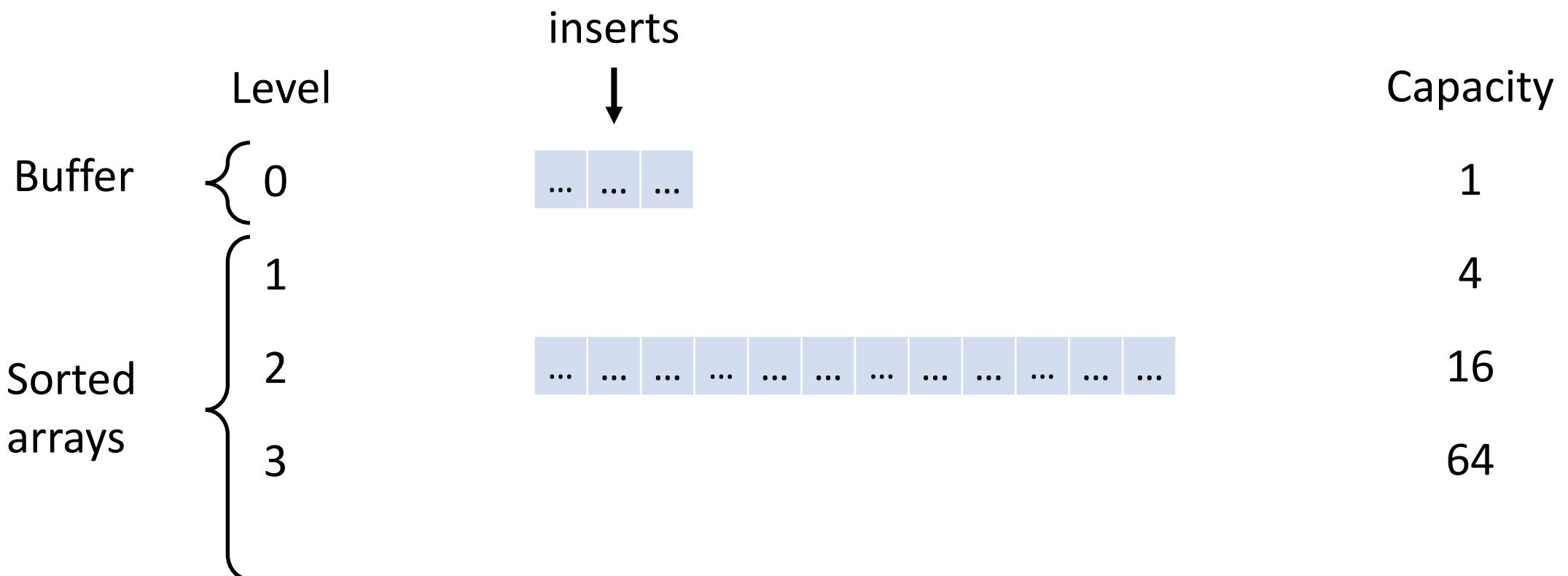
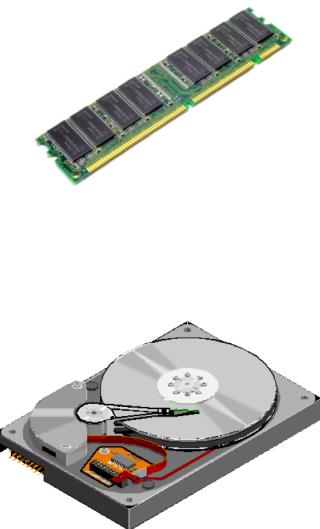
How to reduce it?

E.g. size ratio of 4

Increase size ratio T

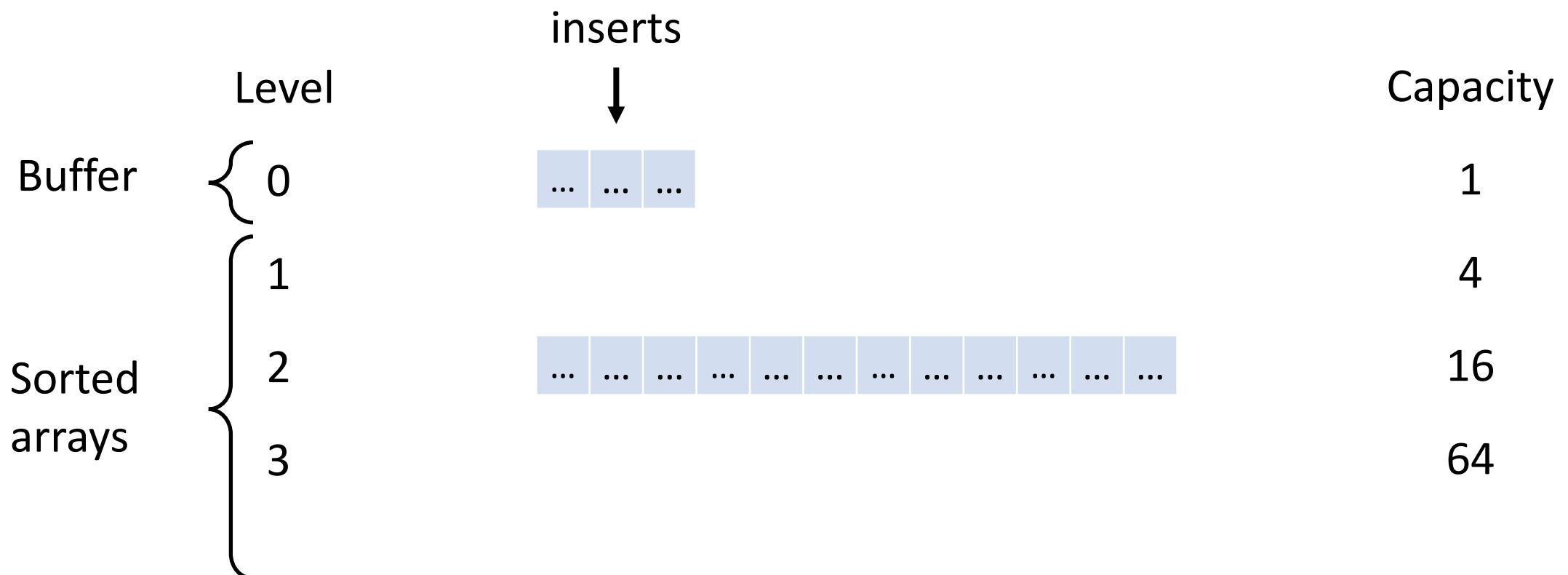
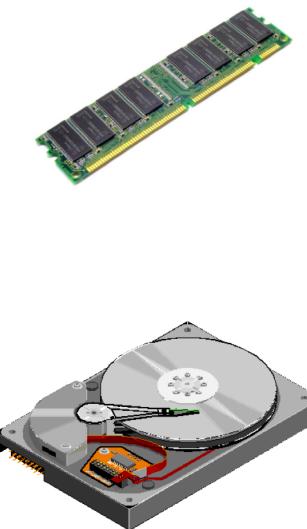


# Leveled LSM-tree



# Leveled LSM-tree

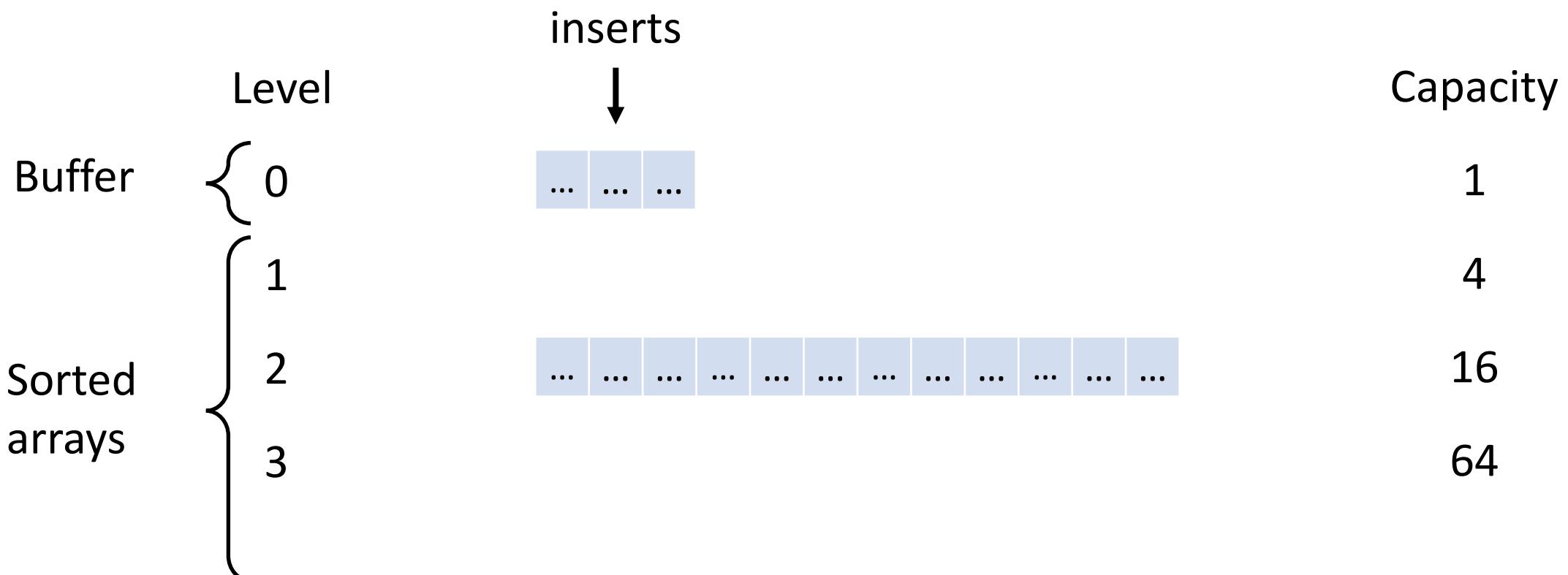
Lookup cost?



# Leveled LSM-tree

Lookup cost?

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

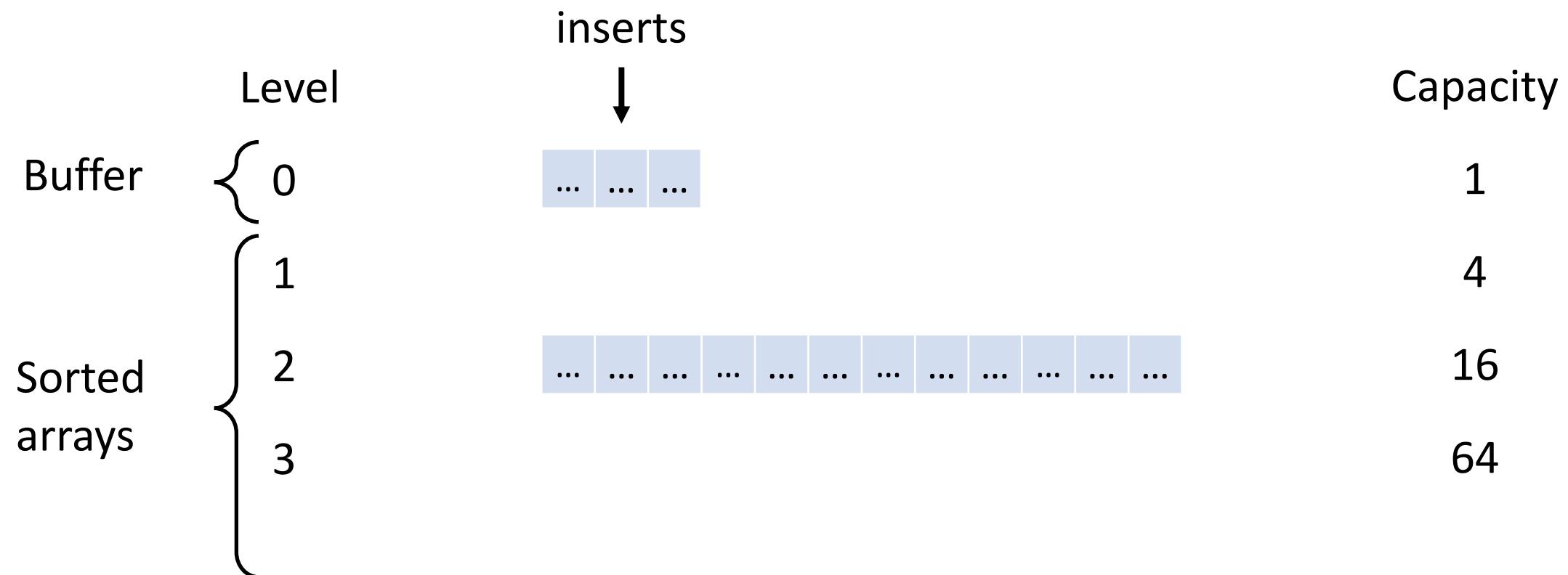


# Leveled LSM-tree

Lookup cost?

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

Insertion cost?



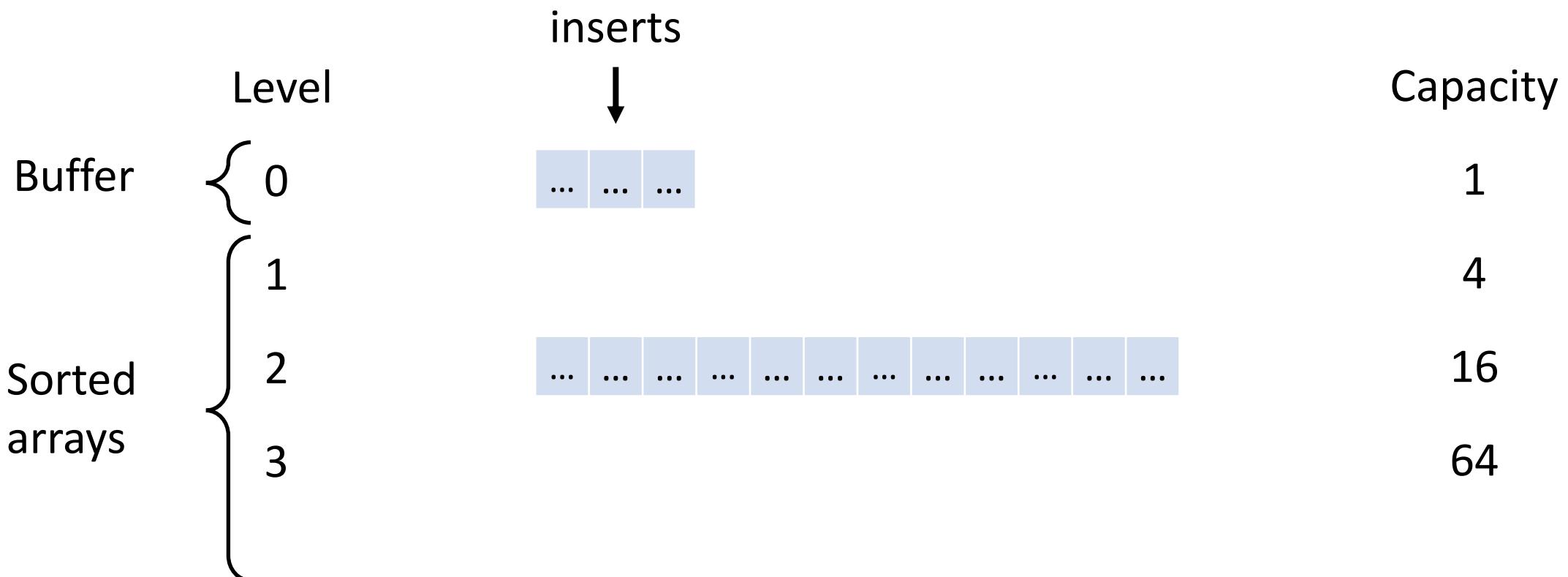
# Leveled LSM-tree

Lookup cost?

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

Insertion cost?

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



# Leveled LSM-tree

Lookup cost?

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

Insertion cost?

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

# Leveled LSM-tree

Lookup cost?

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

Insertion cost?

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

What happens as we increase the size ratio T?

# Leveled LSM-tree



Lookup cost?

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

Insertion cost?

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



What happens as we increase the size ratio T?

# Leveled LSM-tree



Lookup cost?

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

Insertion cost?

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



What happens as we increase the size ratio T?

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

# Leveled LSM-tree



Lookup cost?

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

Insertion cost?

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



What happens as we increase the size ratio T?

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

Lookup cost becomes:

$$O(1)$$

Insert cost becomes:

$$O(N/B^2)$$

# Leveled LSM-tree



Lookup cost?

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

Insertion cost?

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



What happens as we increase the size ratio T?

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

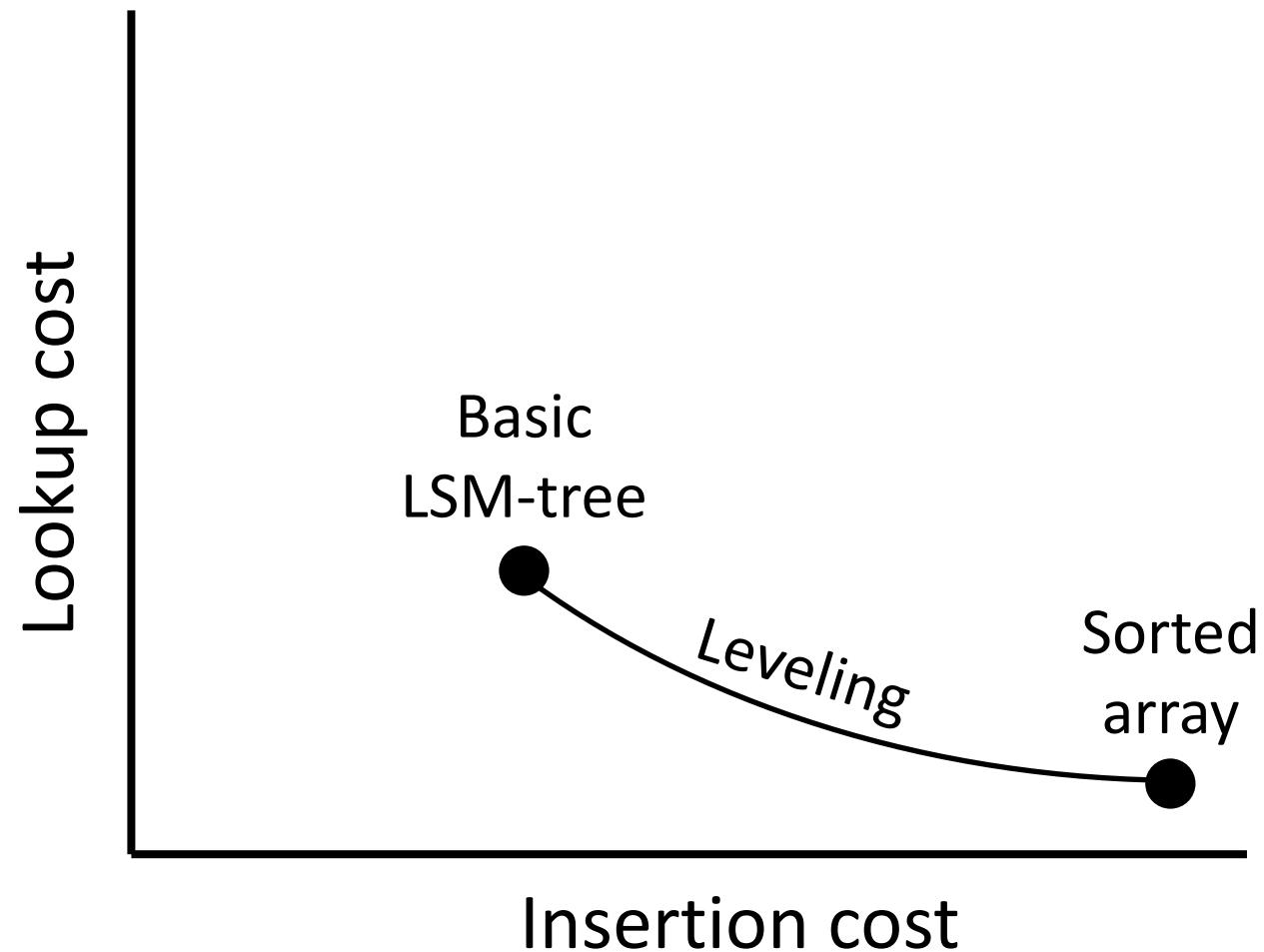
Lookup cost becomes:

$$O(1)$$

Insert cost becomes:

$$O(N/B^2)$$

The LSM-tree becomes a sorted array!



# Results Catalogue – with fence pointers

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

# Tiered LSM-tree

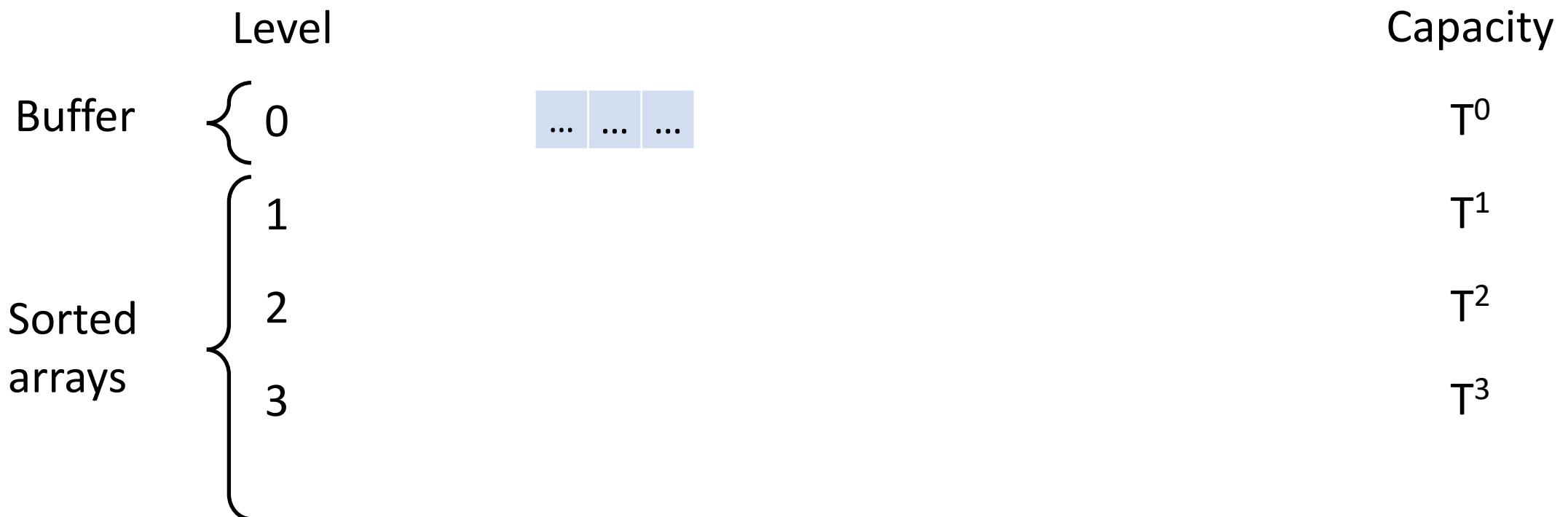
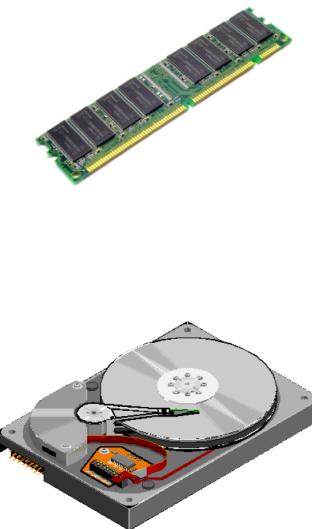
# Tiered LSM-tree

 Lookup cost

 Insertion cost

# Tiered LSM-tree

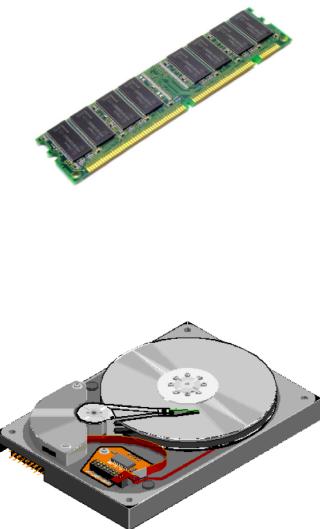
Reduce the number of levels by increasing the size ratio.



# Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.

Do not merge within a level.



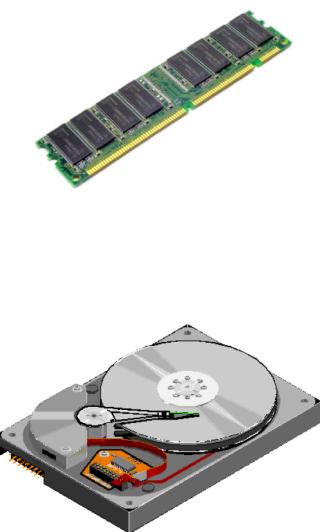
	Level	Capacity
Buffer	0	$T^0$
Sorted arrays	1	$T^1$
	2	$T^2$
	3	$T^3$

# 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



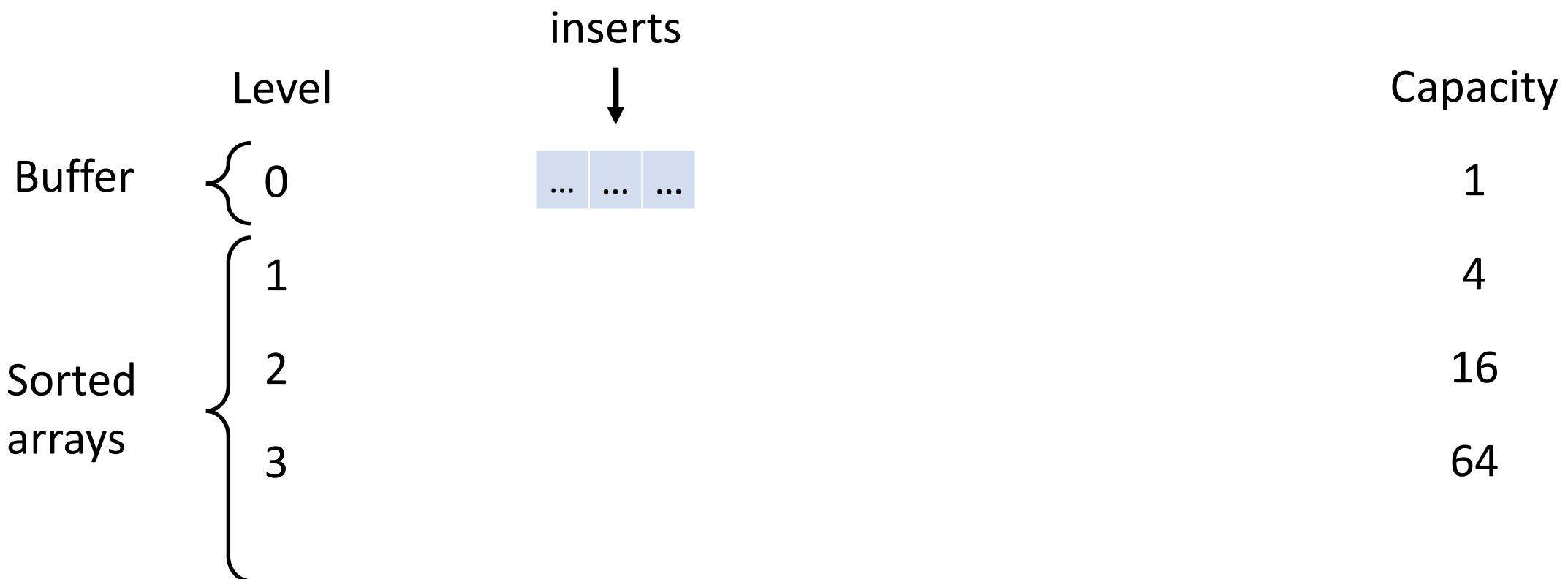
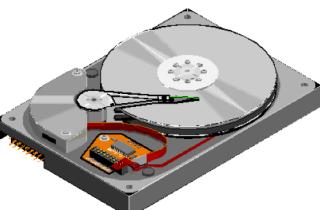
	Level	Capacity
Buffer	0	1
Sorted arrays	1	4
	2	16
	3	64

# 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

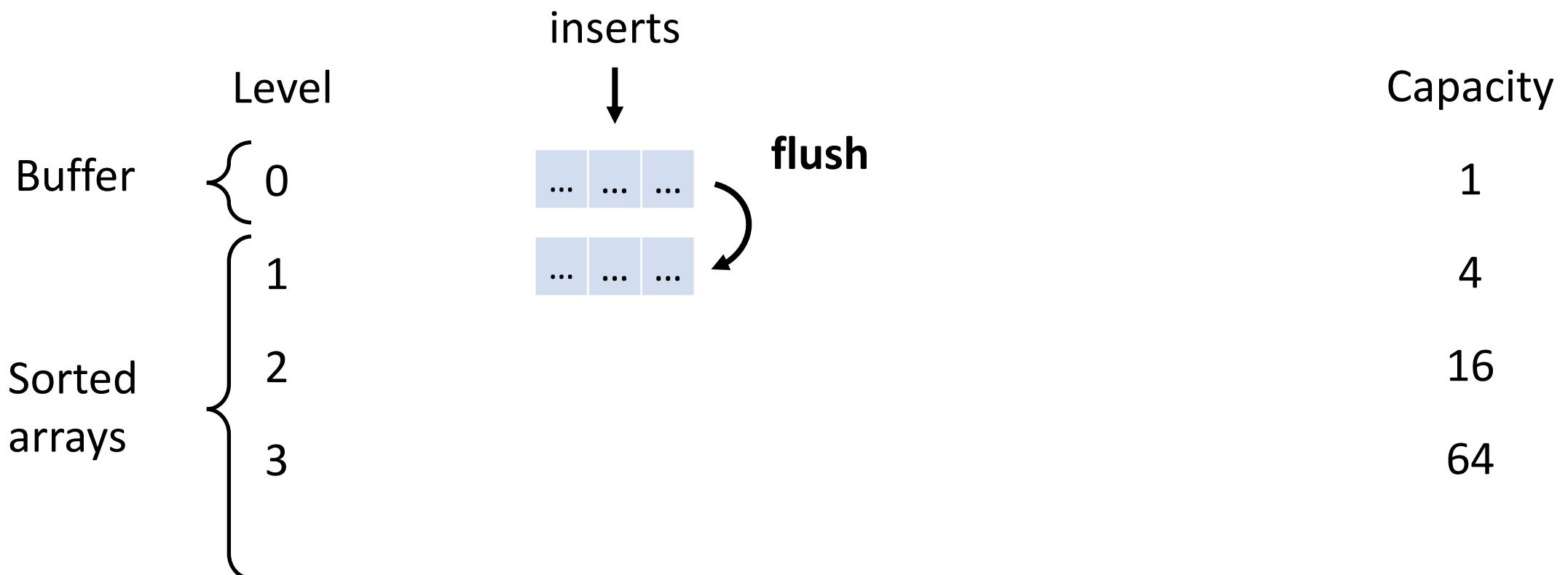
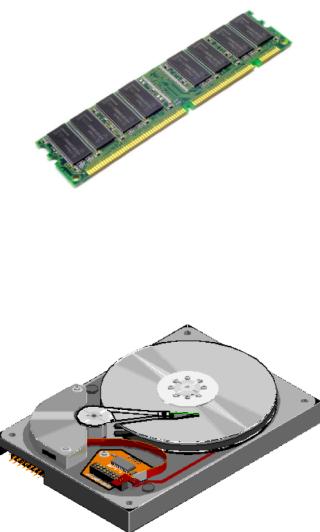


# Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.

Do not merge within a level.

E.g. size ratio of 4

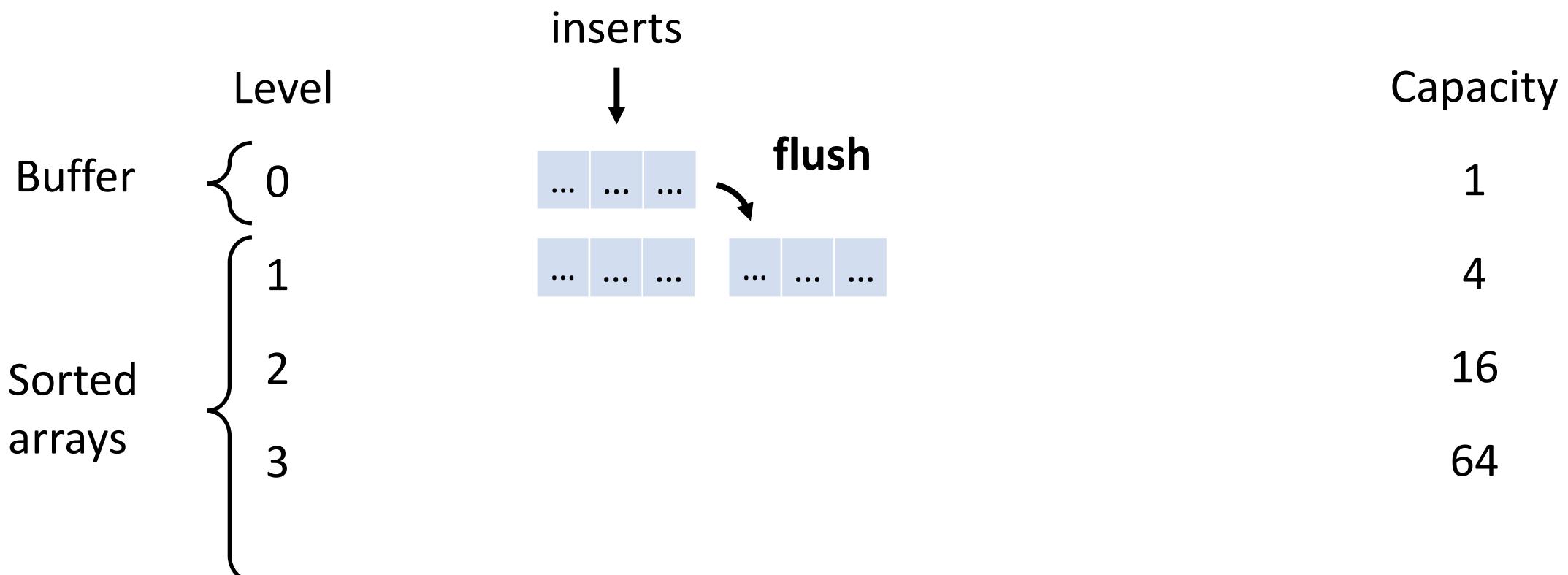
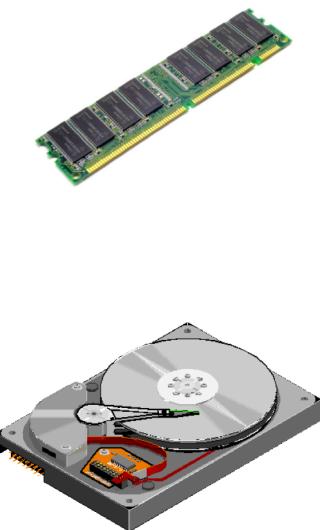


# Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.

Do not merge within a level.

E.g. size ratio of 4

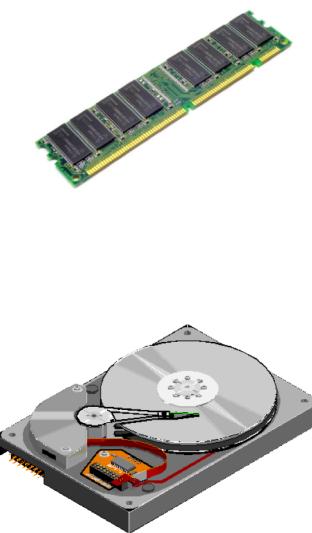


# Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.

Do not merge within a level.

E.g. size ratio of 4



	Level	Capacity
Buffer	0	1
Sorted arrays	1	4
	2	16
	3	64

The diagram illustrates the insertion process and flushing mechanism:

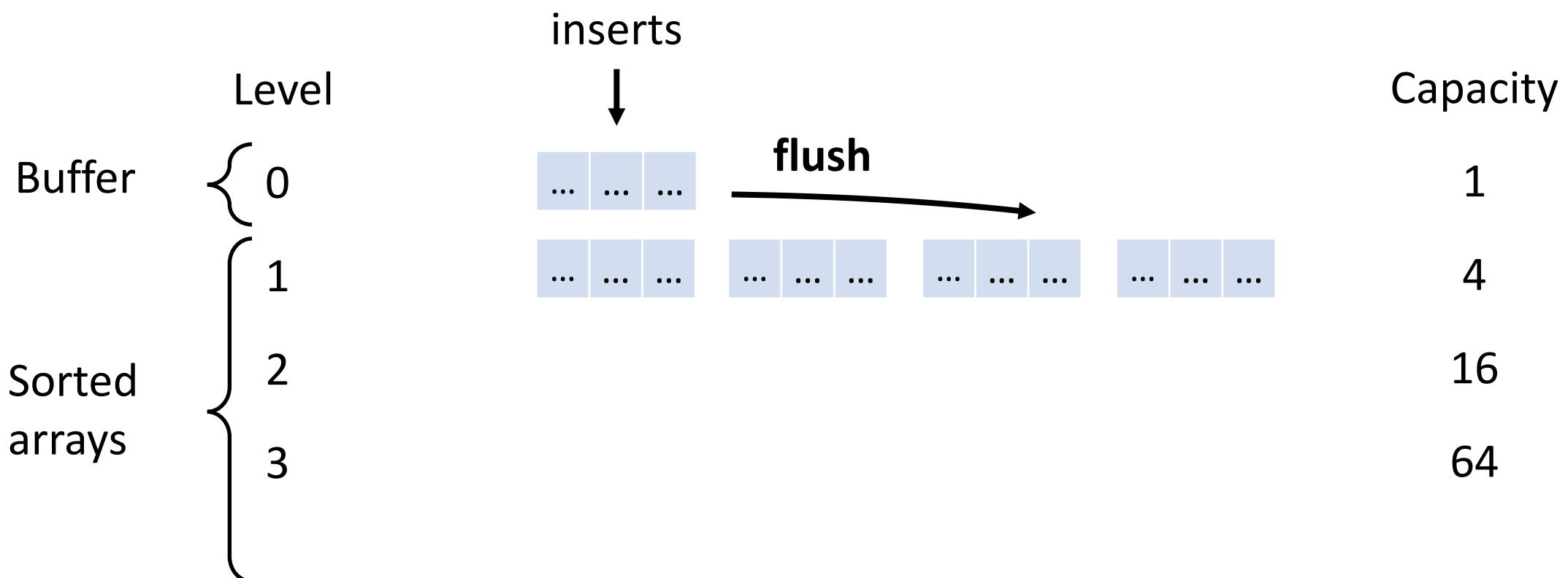
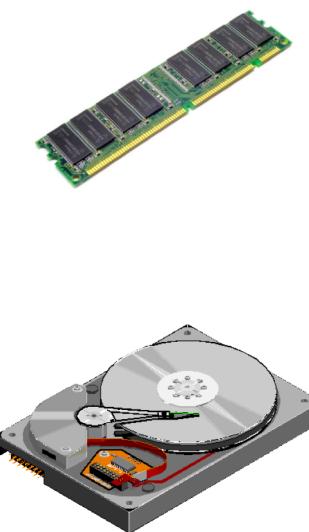
- inserts:** An arrow labeled "inserts" points downwards from the Buffer level to the Sorted arrays level.
- flush:** A curved arrow labeled "flush" points from the second-to-last element of the first sorted array to the last element of the second sorted array.
- The table shows the capacity of each level: Level 0 has a capacity of 1, Level 1 has a capacity of 4, Level 2 has a capacity of 16, and Level 3 has a capacity of 64.

# 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

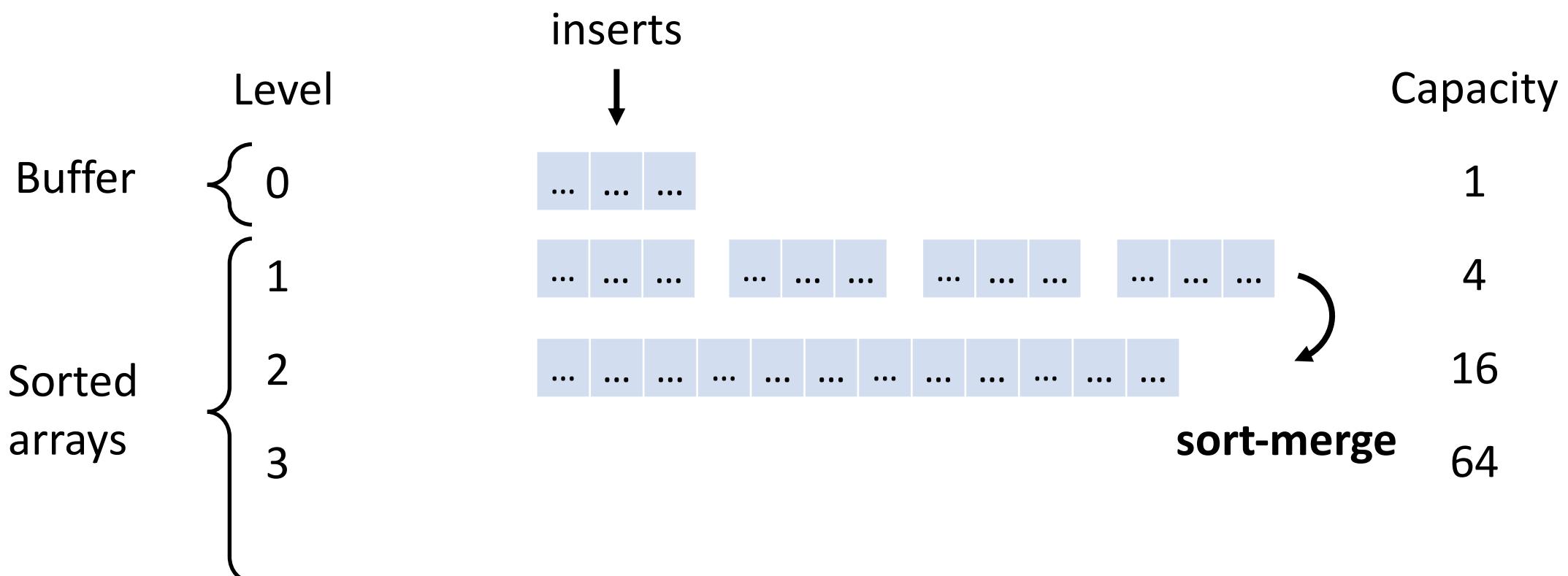
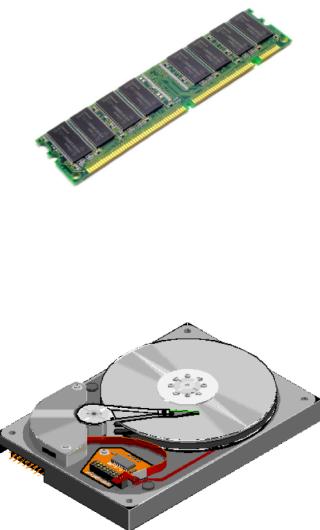


# Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.

Do not merge within a level.

E.g. size ratio of 4

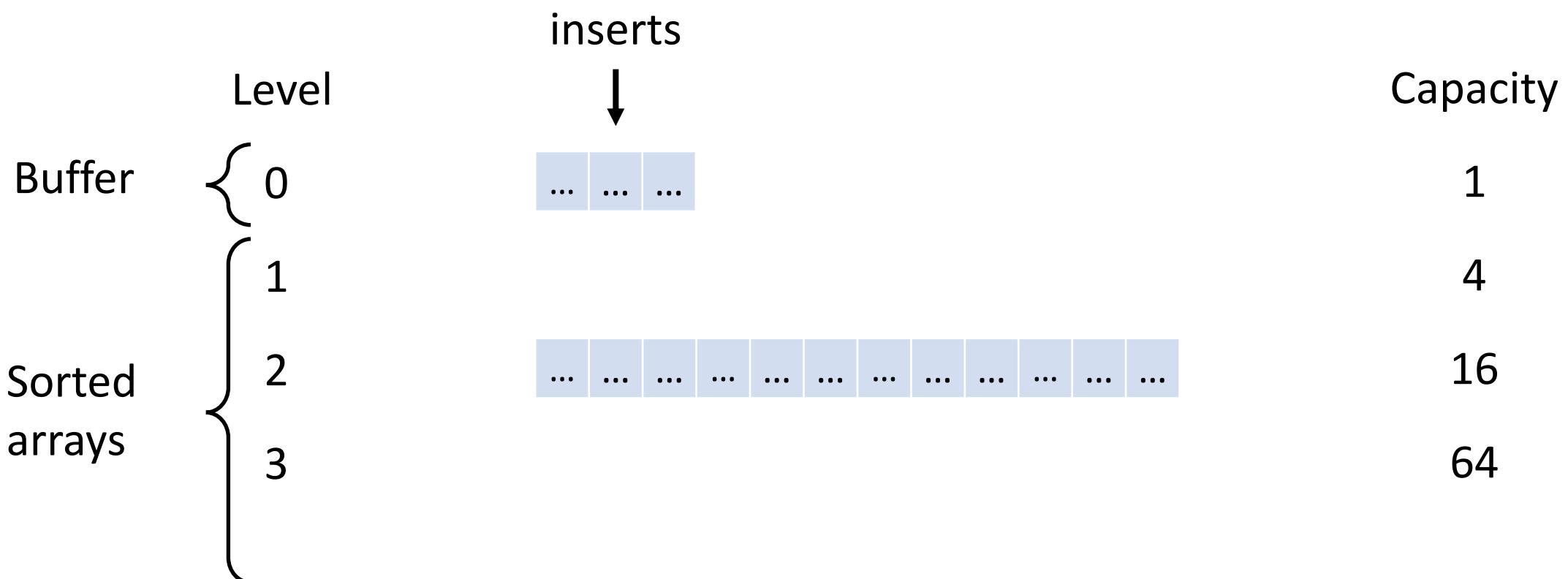
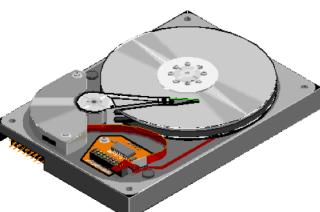


# Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.

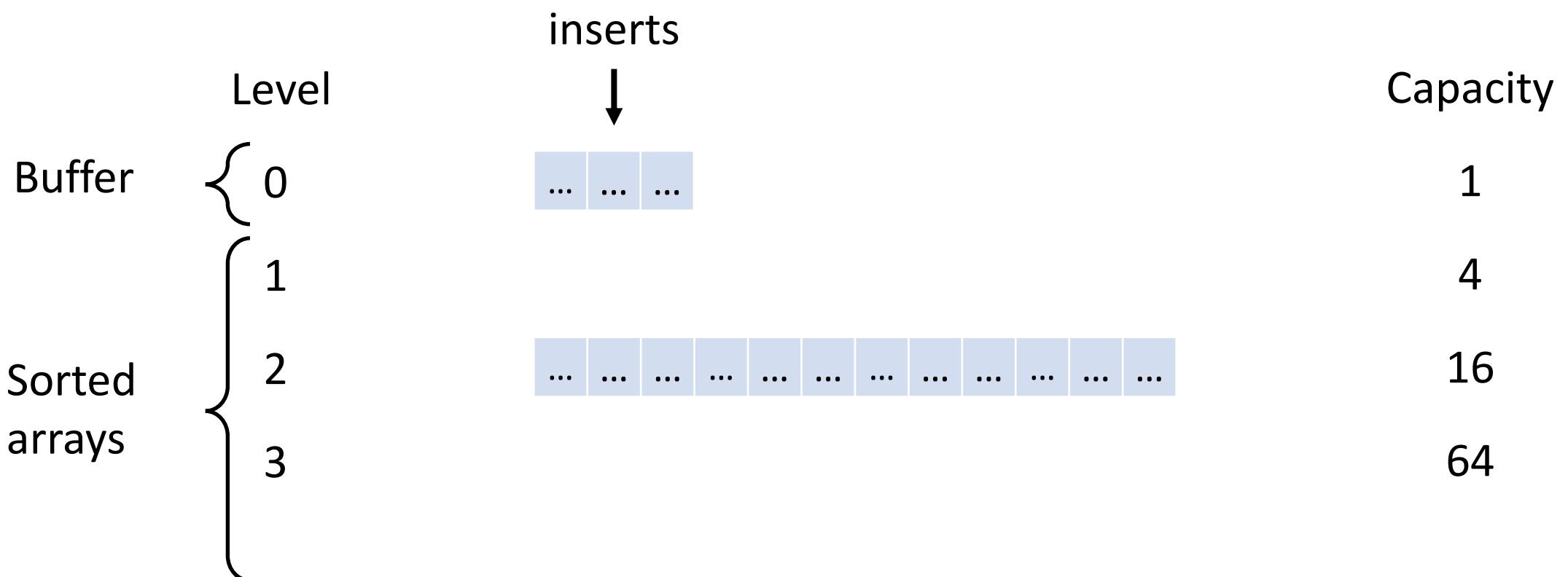
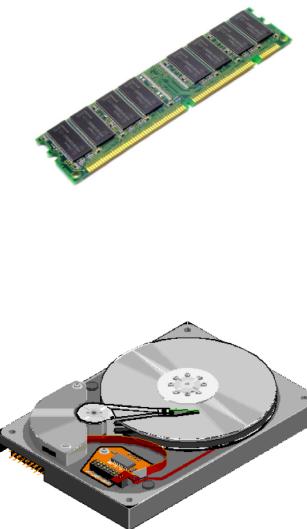
Do not merge within a level.

E.g. size ratio of 4



# Tiered LSM-tree

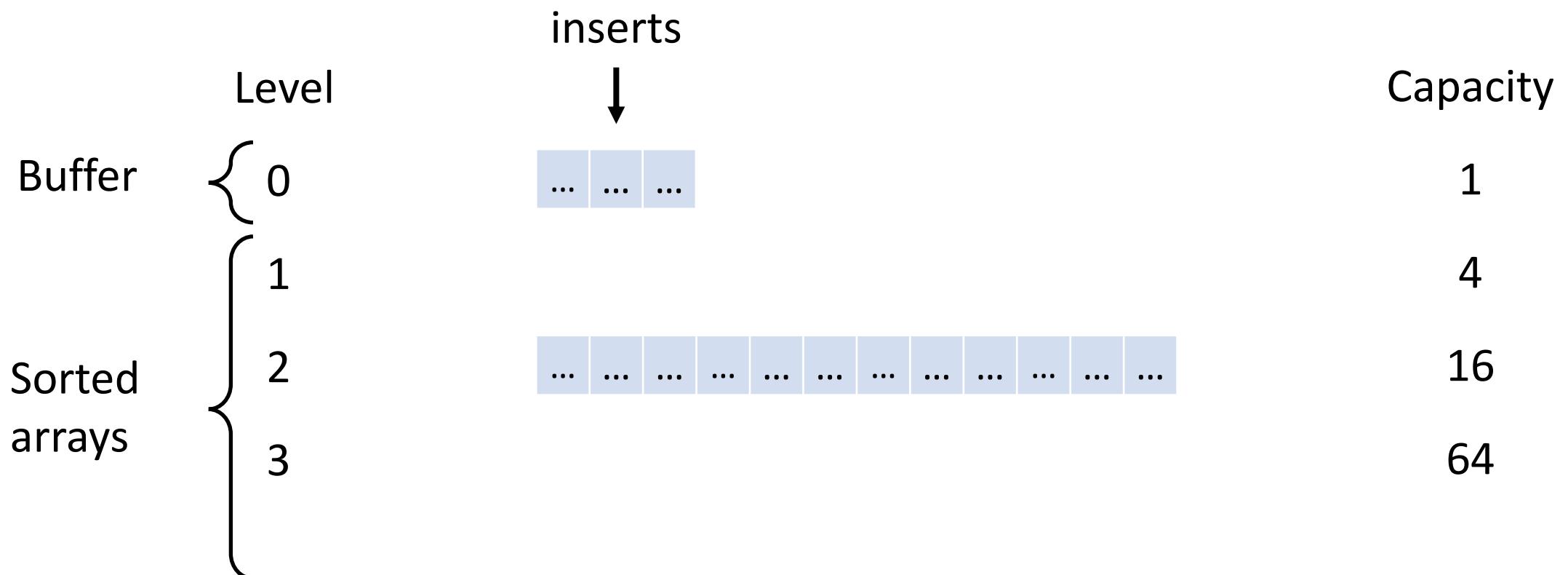
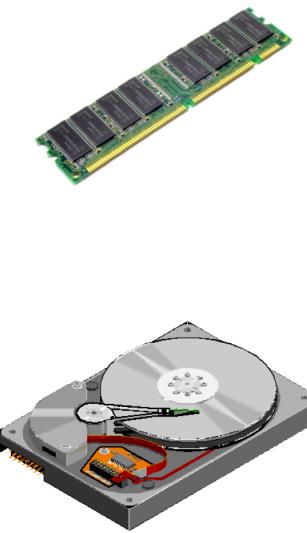
Lookup cost?



# Tiered LSM-tree

Lookup cost?

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

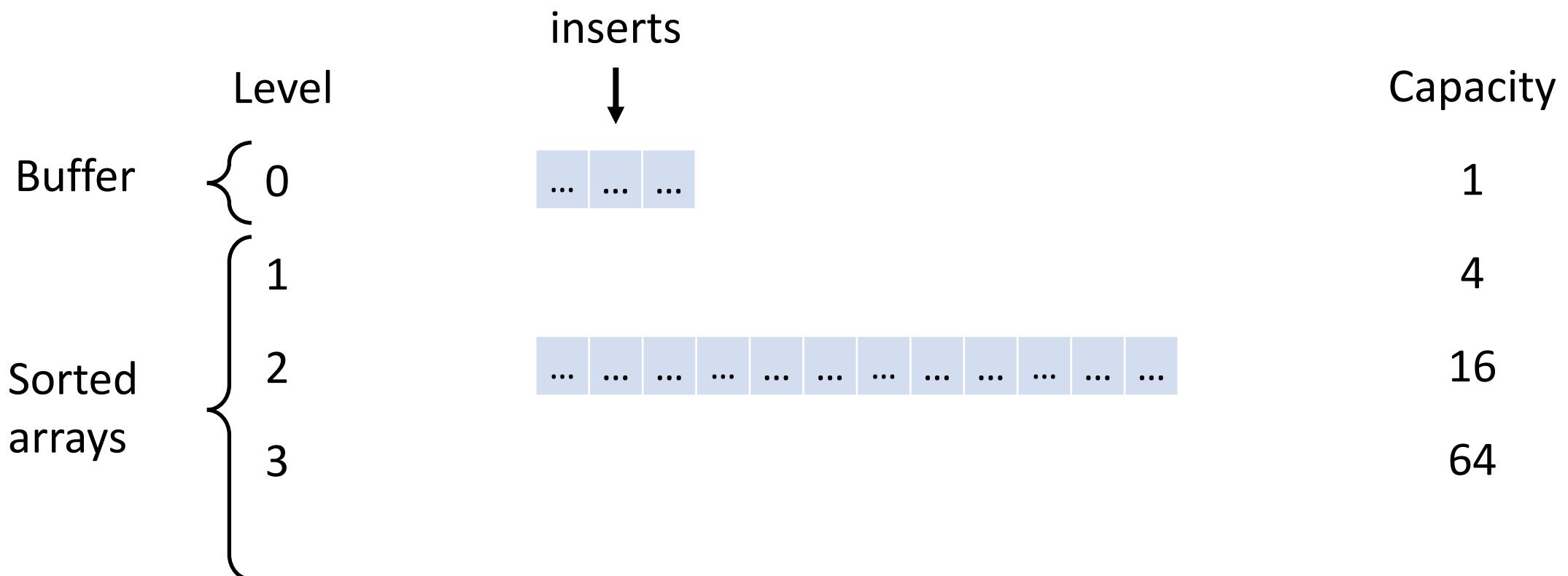
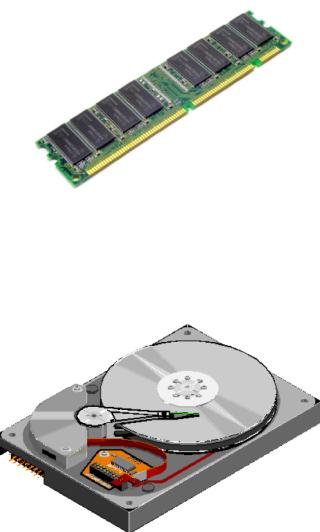


# Tiered LSM-tree

Lookup cost?

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

Insertion cost?



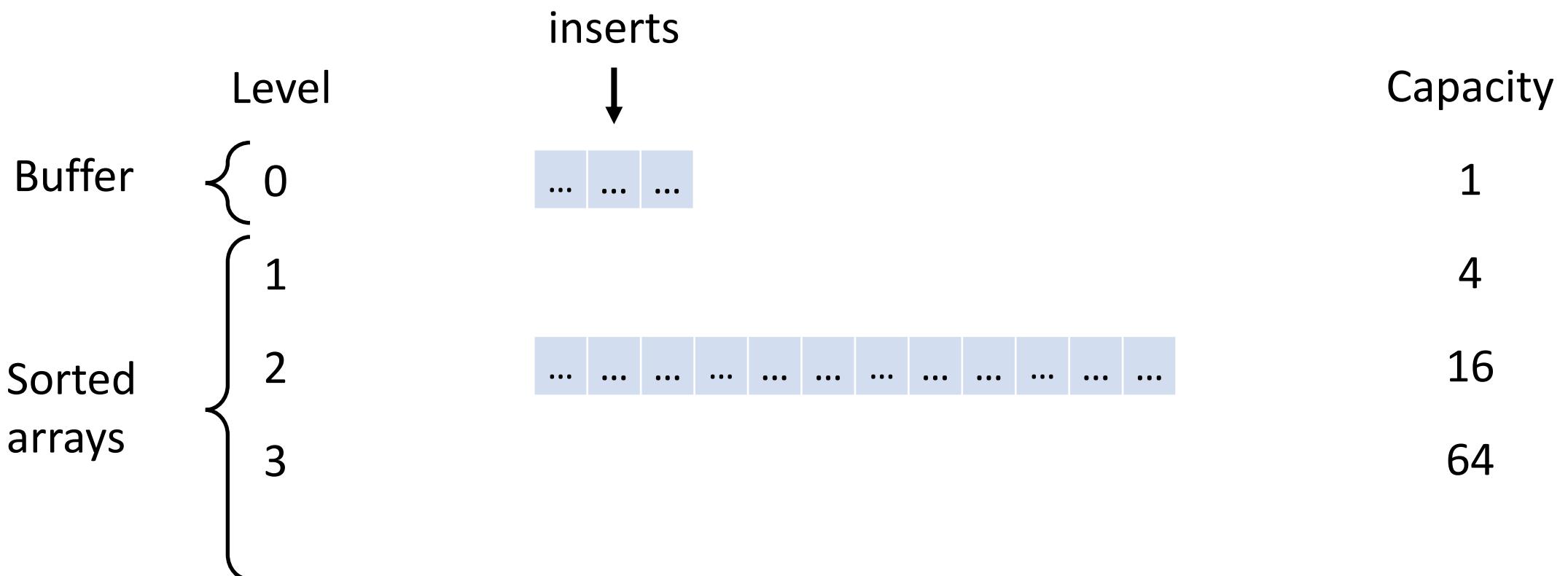
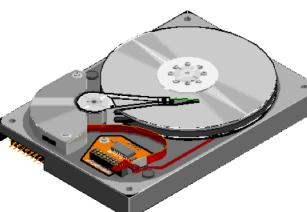
# Tiered LSM-tree

Lookup cost?

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

Insertion cost?

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



# Tiered LSM-tree

Lookup cost?

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

Insertion cost?

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

# Tiered LSM-tree

Lookup cost?

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

Insertion cost?

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

What happens as we increase the size ratio T?

# Tiered LSM-tree

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

Insertion cost?  
 $O\left(\frac{1}{B} \cdot \log_T\left(\frac{N}{B}\right)\right)$  ↓

What happens as we increase the size ratio T?

# Tiered LSM-tree

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

Insertion cost?  
 $O\left(\frac{1}{B} \cdot \log_T\left(\frac{N}{B}\right)\right)$  ↓

What happens as we increase the size ratio T?

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

# Tiered LSM-tree

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

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


What happens as we increase the size ratio T?

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

Lookup cost becomes:  
 $O(N/B)$

Insert cost becomes:  
 $O(1/B)$

# Tiered LSM-tree

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

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

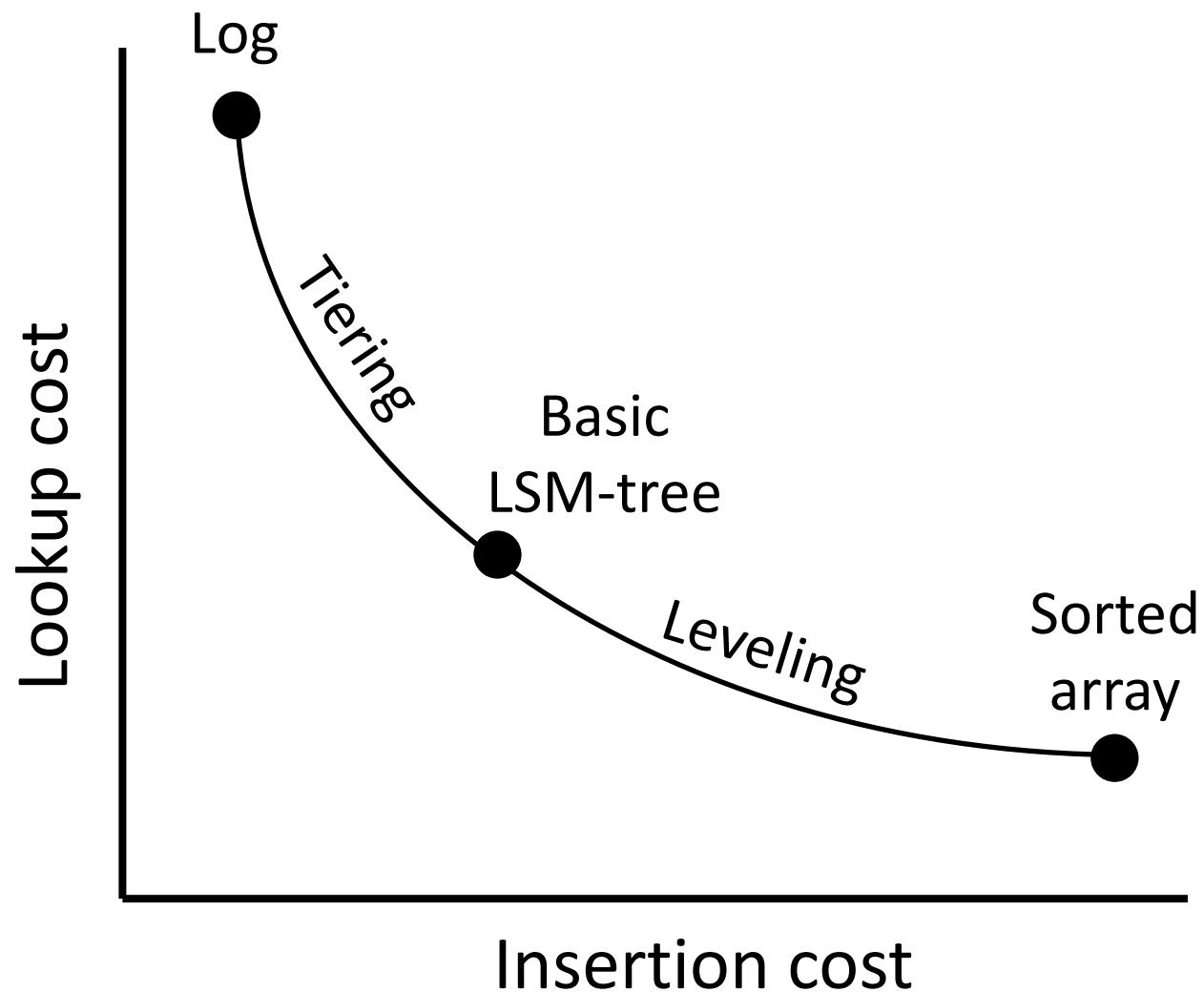
What happens as we increase the size ratio T?

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

Lookup cost becomes:  
 $O(N/B)$

Insert cost becomes:  
 $O(1/B)$

The tiered LSM-tree becomes a log!

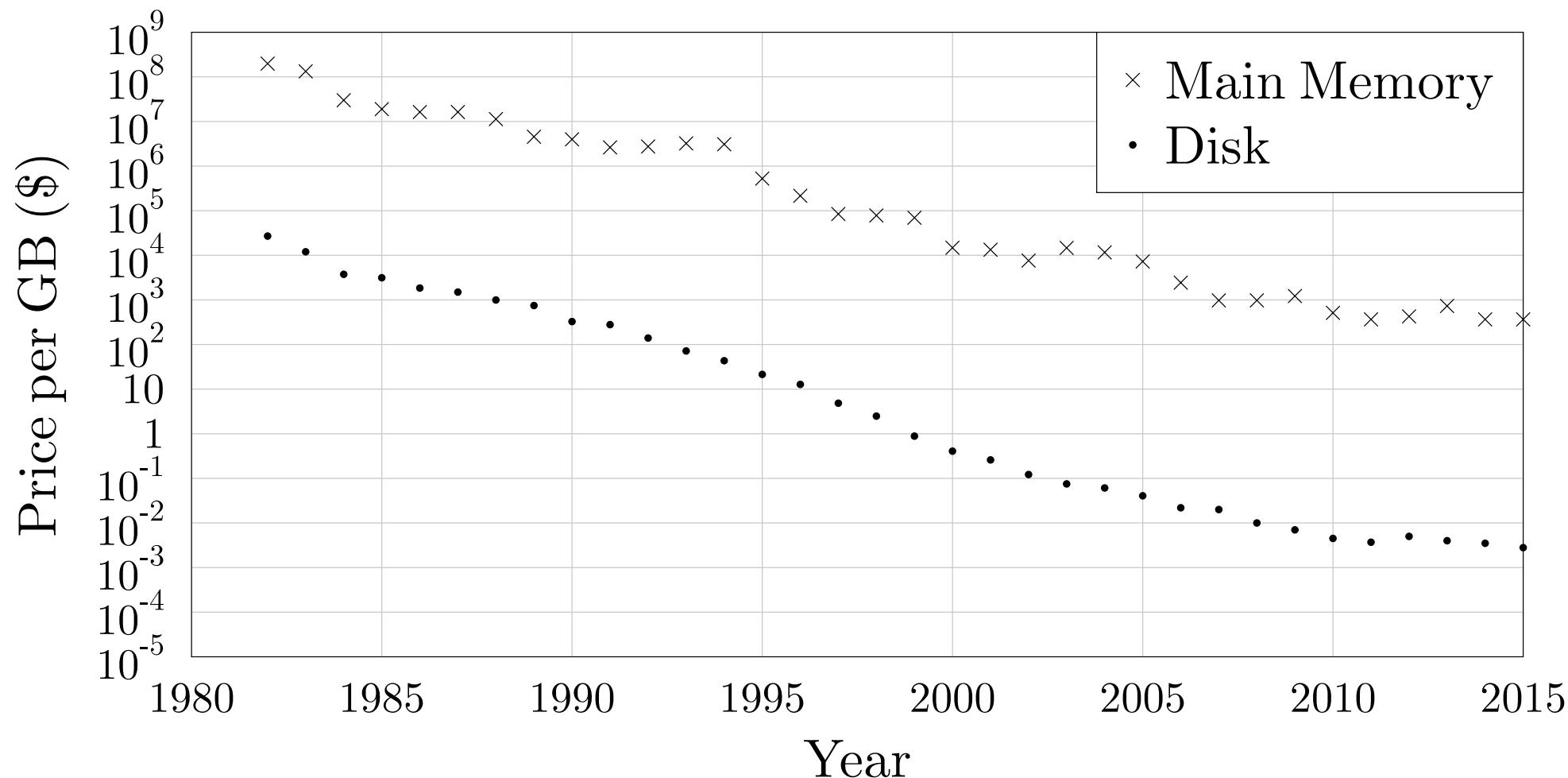


# Results Catalogue – with fence pointers

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

# Bloom filters

# Declining Main Memory Cost



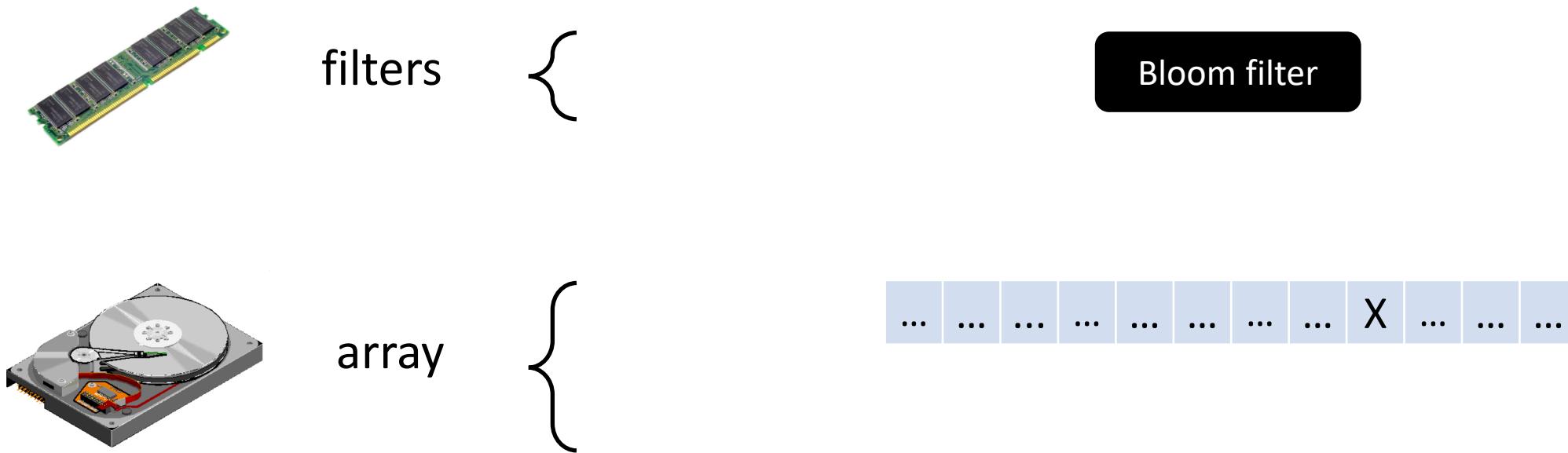
# Bloom Filters

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.



filters {



array {

**Lookup for X**



Bloom filter



# 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.



filters {



array {

Lookup for X



Bloom filter



# Bloom Filters

Answers set-membership queries

Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

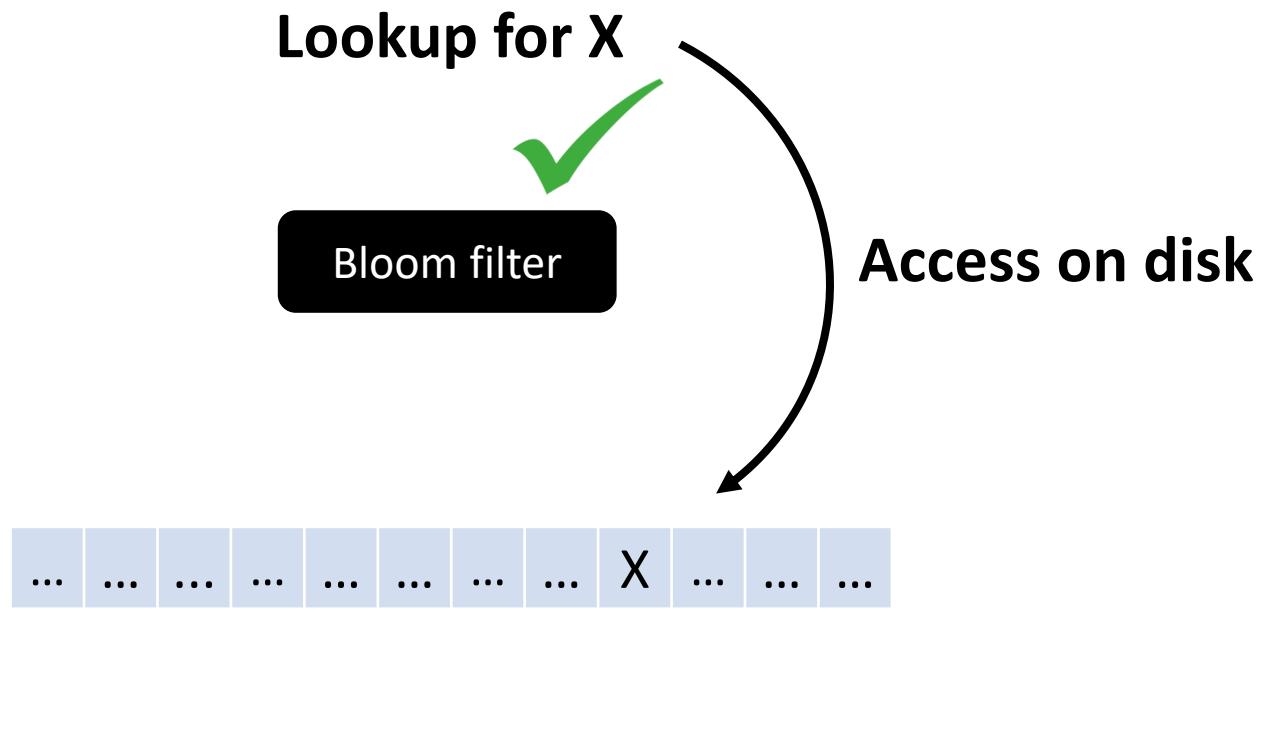
Subtlety: may return false positives.



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.



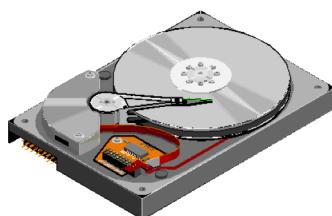
filters



**Lookup for Y**



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.



filters {



array {



Lookup for Y



Bloom filter

# 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



Bloom filter



array



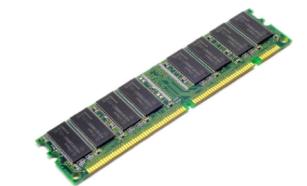
# Bloom Filters

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.



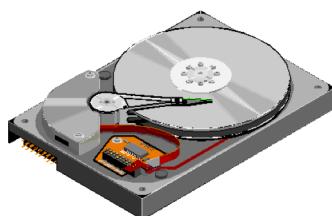
filters



**Lookup for Z**



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.



filters {



array {



Lookup for Z



Bloom filter



# Bloom Filters

Answers set-membership queries

Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

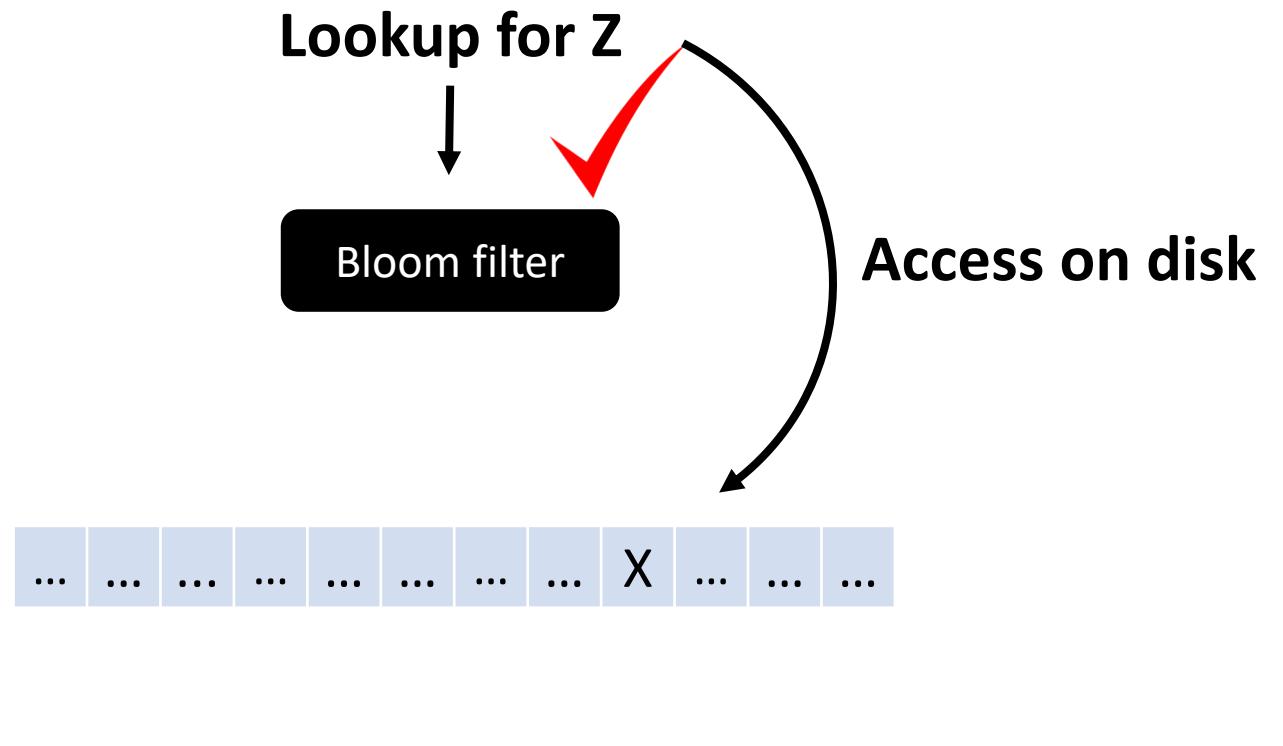
Subtlety: may return false positives.



filters

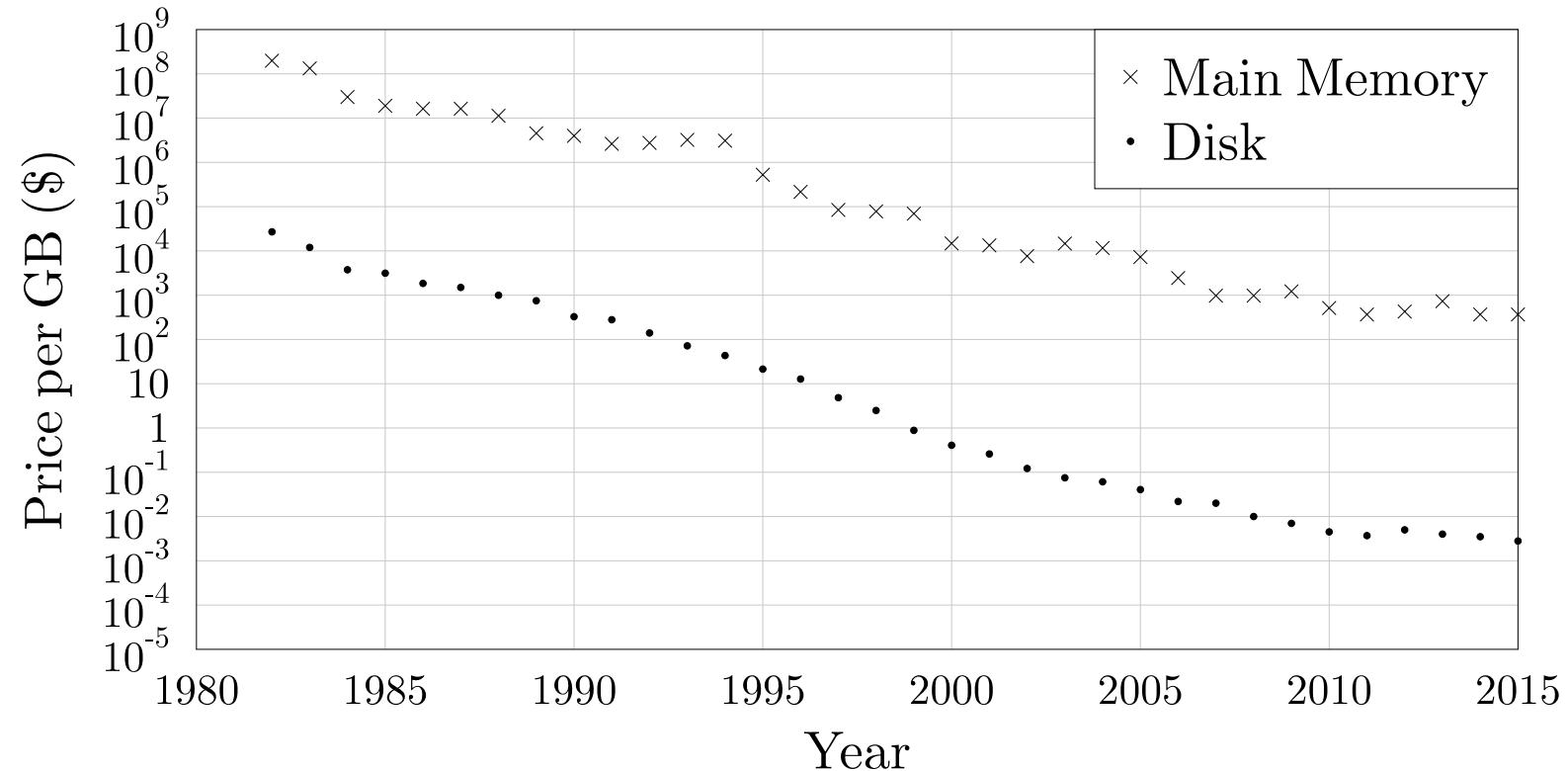


array



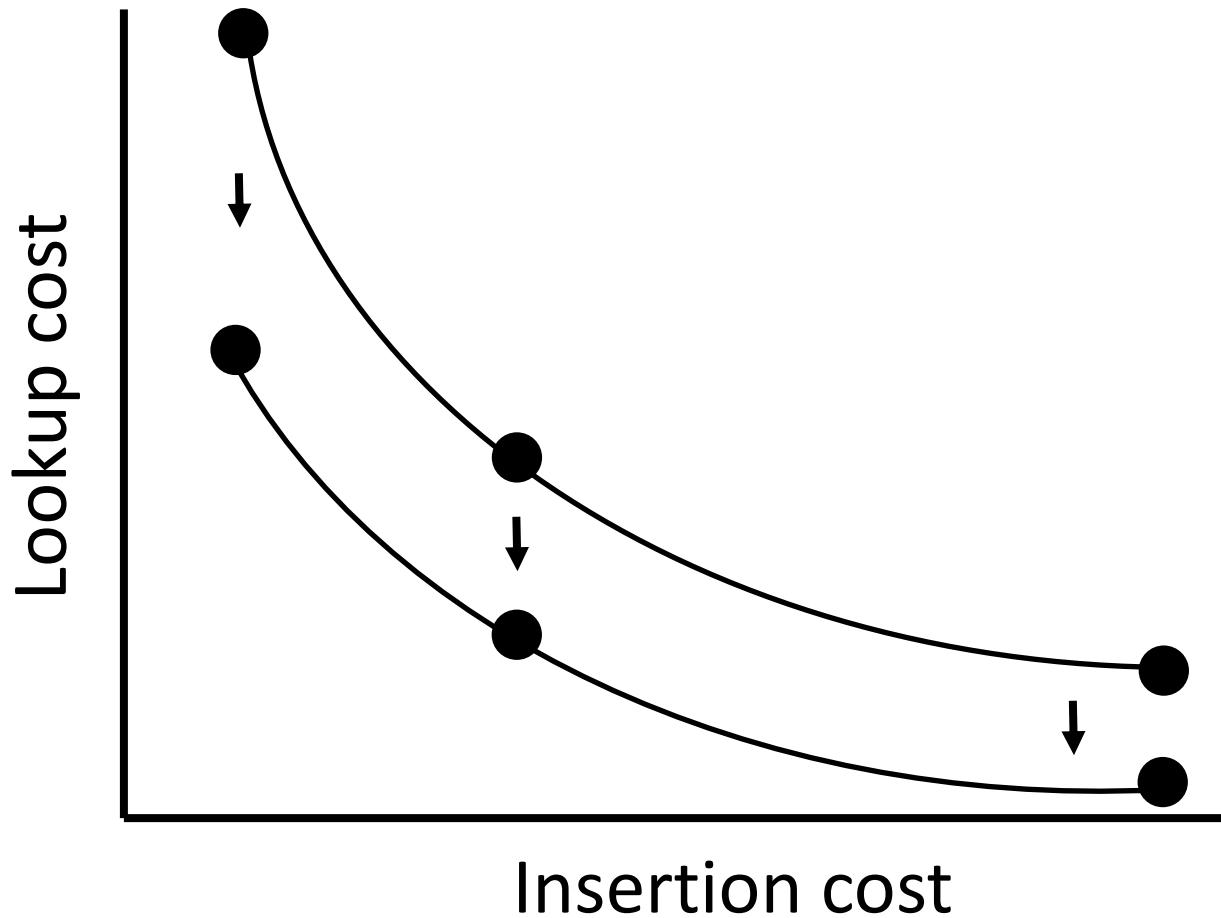
# Bloom Filters

The more main memory, the fewer false positives  $\Rightarrow$  cheaper lookups



# Bloom Filters

The more main memory, the fewer false positives  $\Rightarrow$  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!**