CS460: Intro to Database Systems

Class 14: Log-Structured-Merge Trees

Instructor: Manos Athanassoulis

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

based on slides from Niv Dayan

Useful when?

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

⟹ LSM-trees are for you.

Why now?

Patrick O'Neil
UMass Boston

Invented in
1996

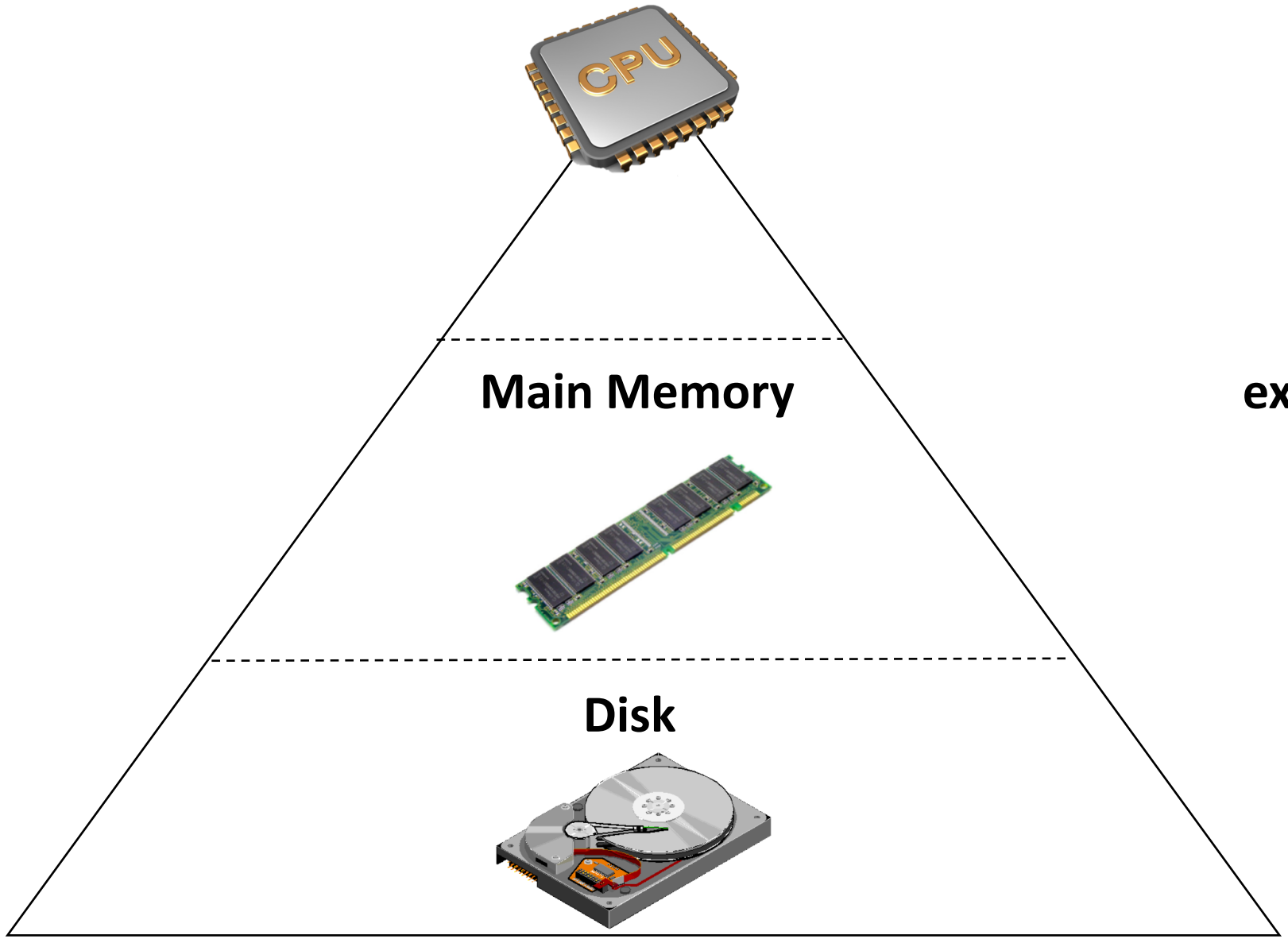
1980 1990 2000 2010

Time

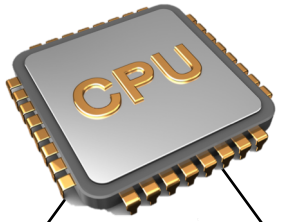
Outline

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

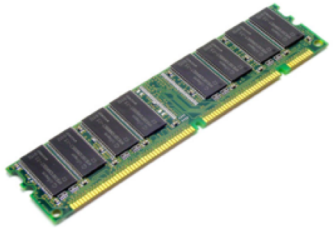

CPU
Main Memory
Disk
expensive, fast
cheap, slow

The Memory Hierarchy



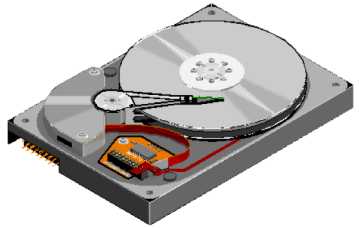
Main Memory

expensive, fast

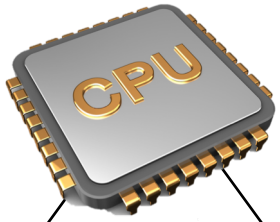


Disk

cheap, slow



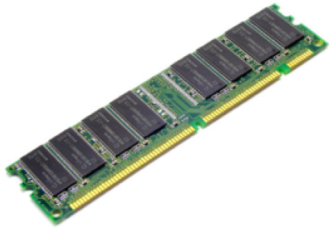
The Memory Hierarchy



Metadata & frequently accessed data

Main Memory

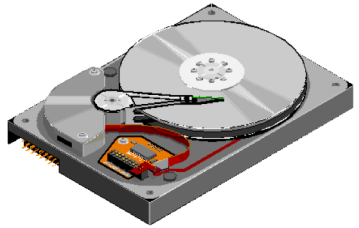
expensive, fast

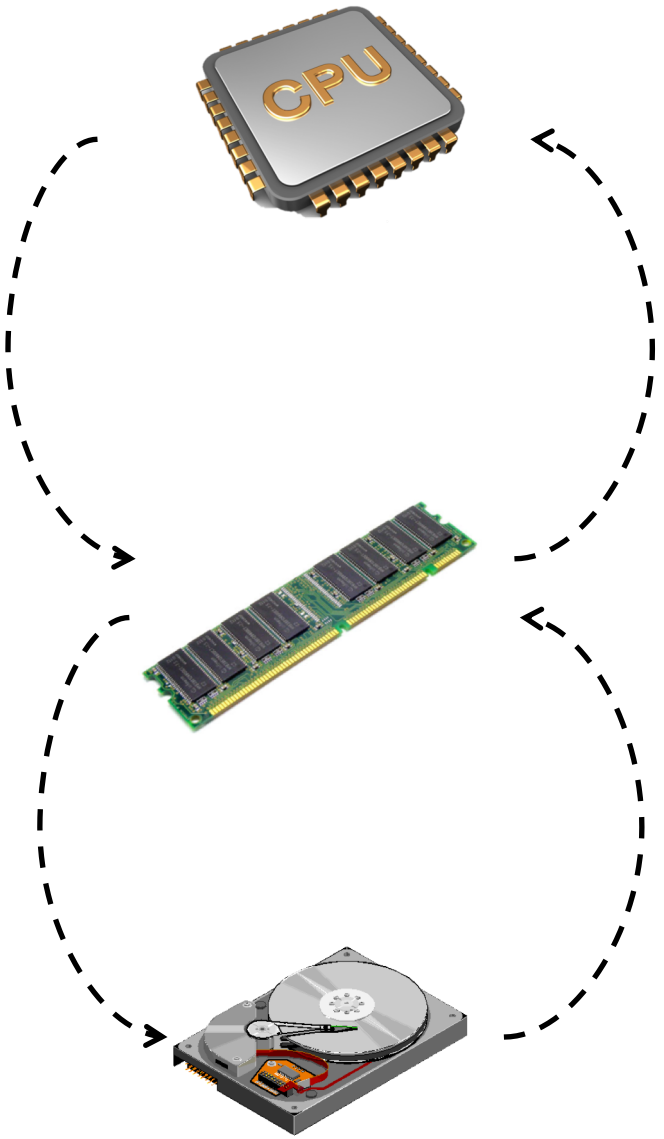


All data

Disk

cheap, slow

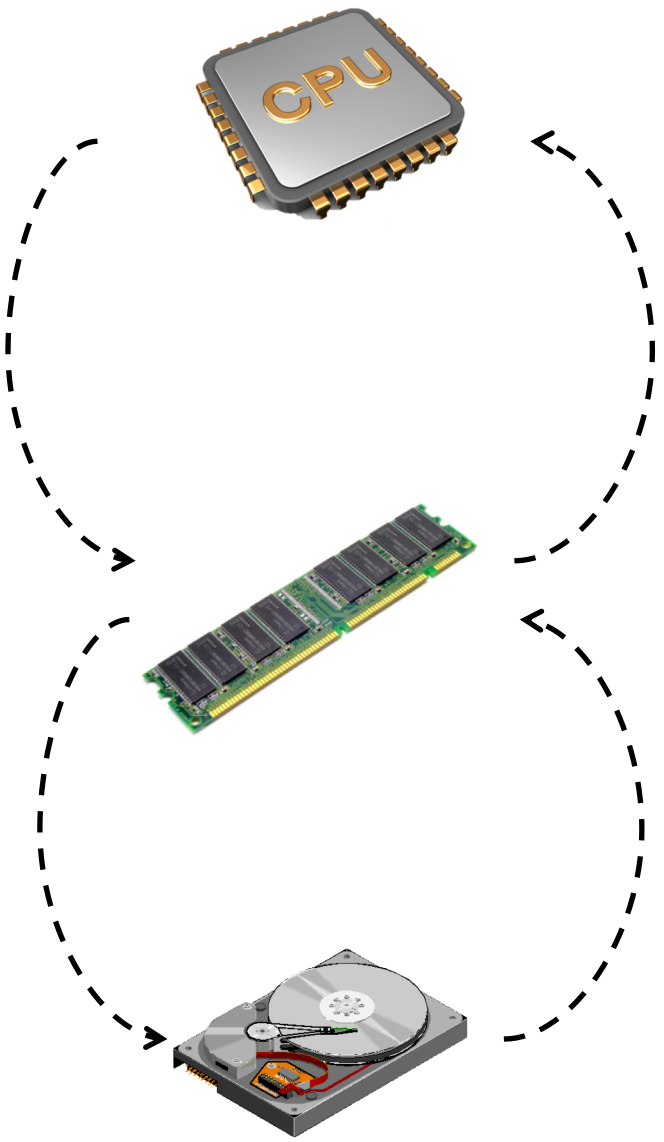




≈100 ns

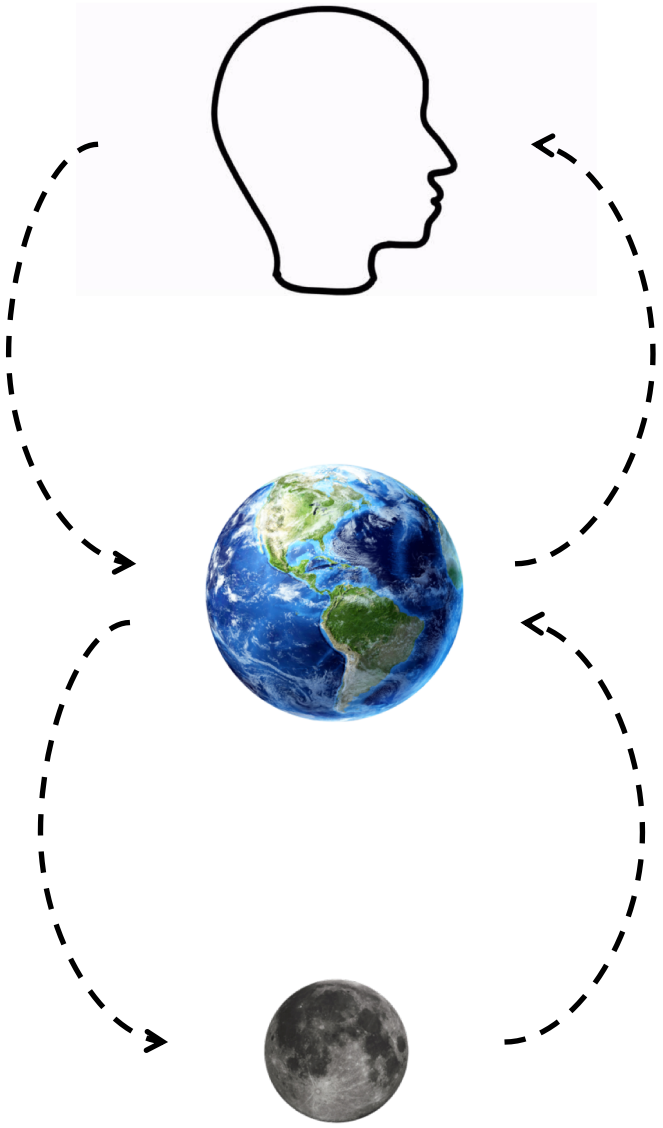
≈10 ms

≈5-6 order of magnitude difference



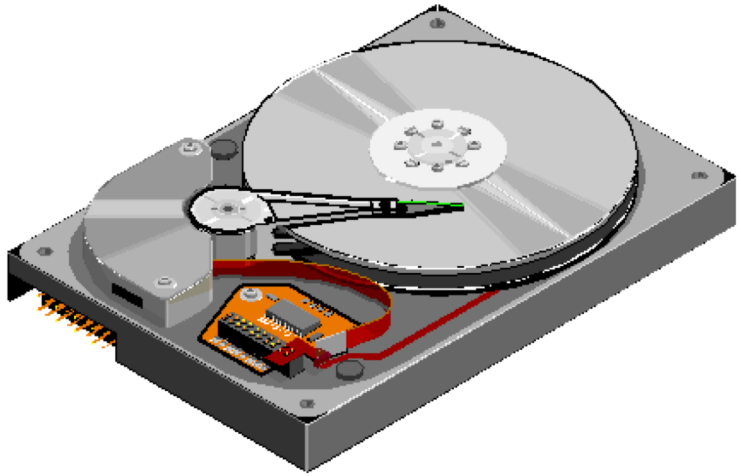
≈100 ns

≈10 ms

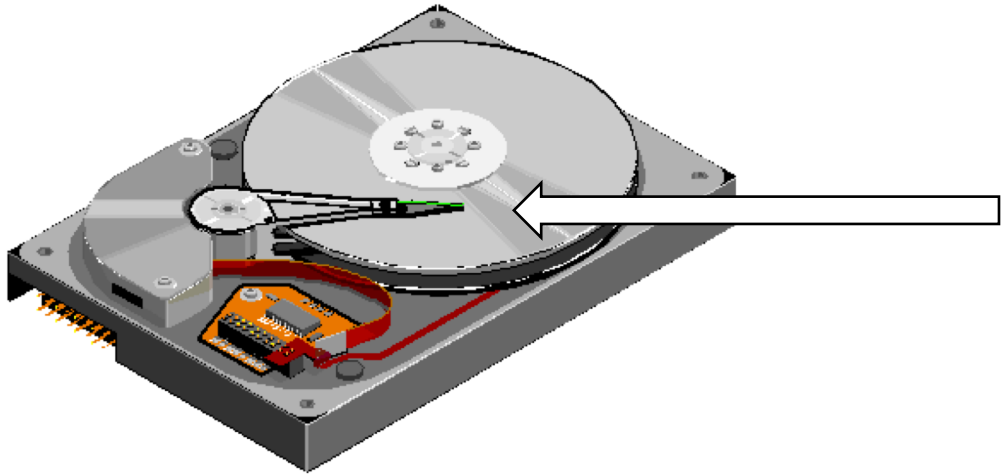


≈5-6 order of magnitude difference

Why is disk slow?

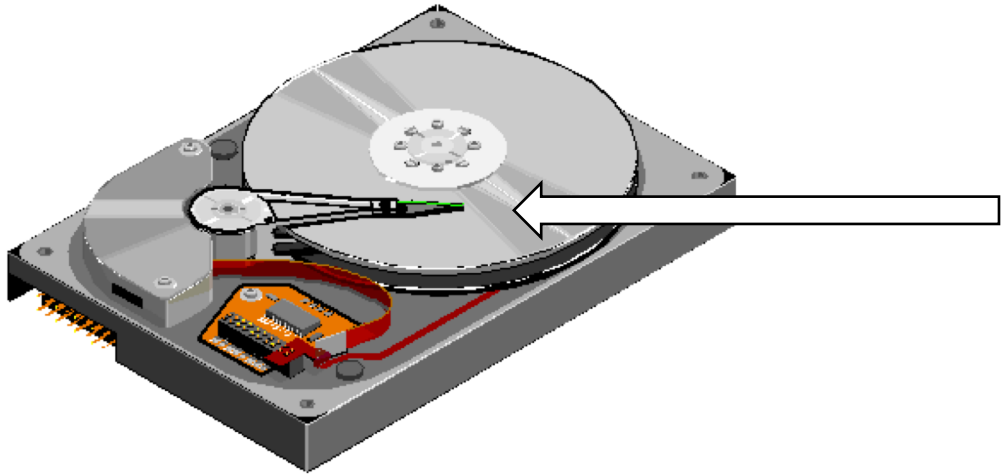


Why is disk slow?



Disk head

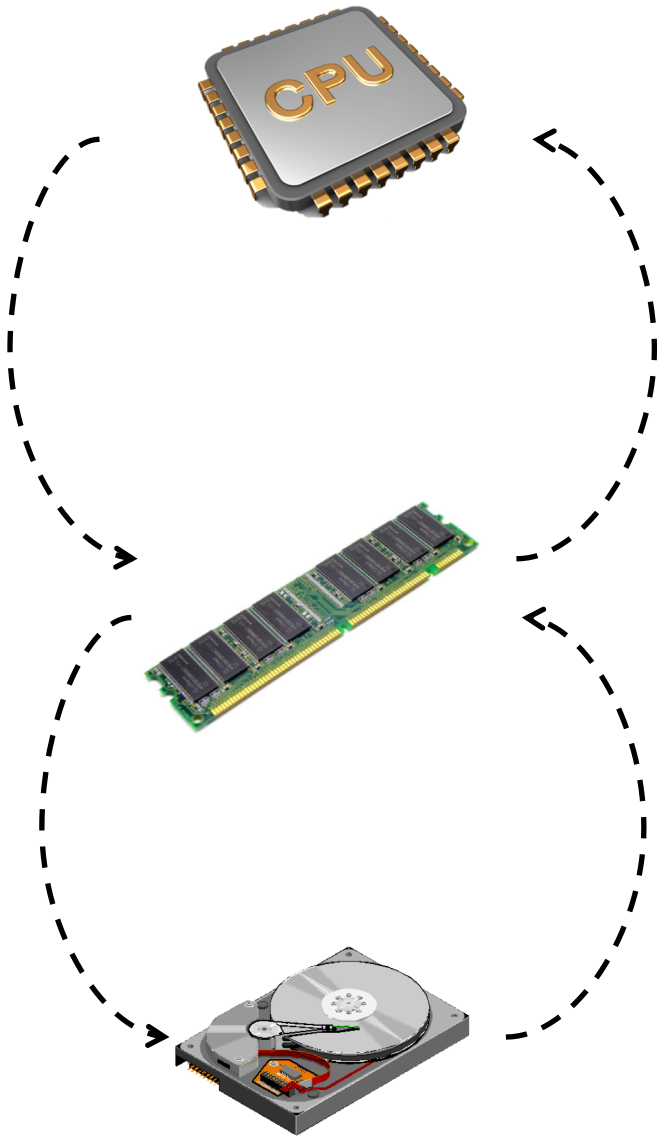
Why is disk slow?



Disk head

Random access is slow ⇒ move disk head

Sequential access is faster ⇒ let disk spin

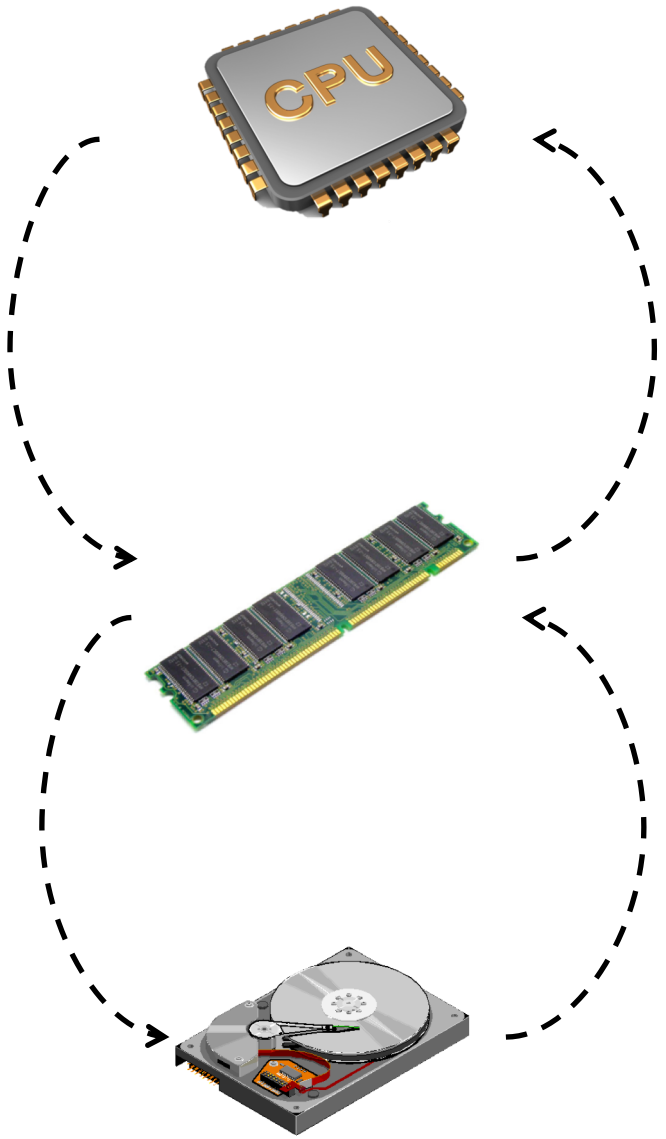


64 byte chunks
Words

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

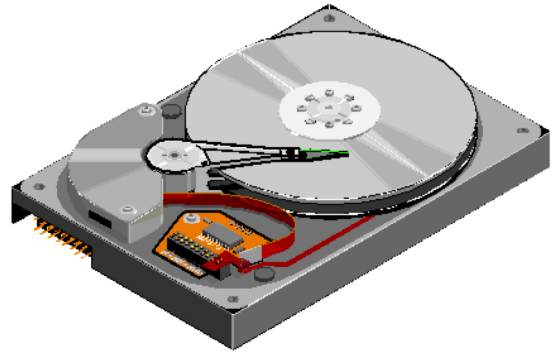
Indexing Problem

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		

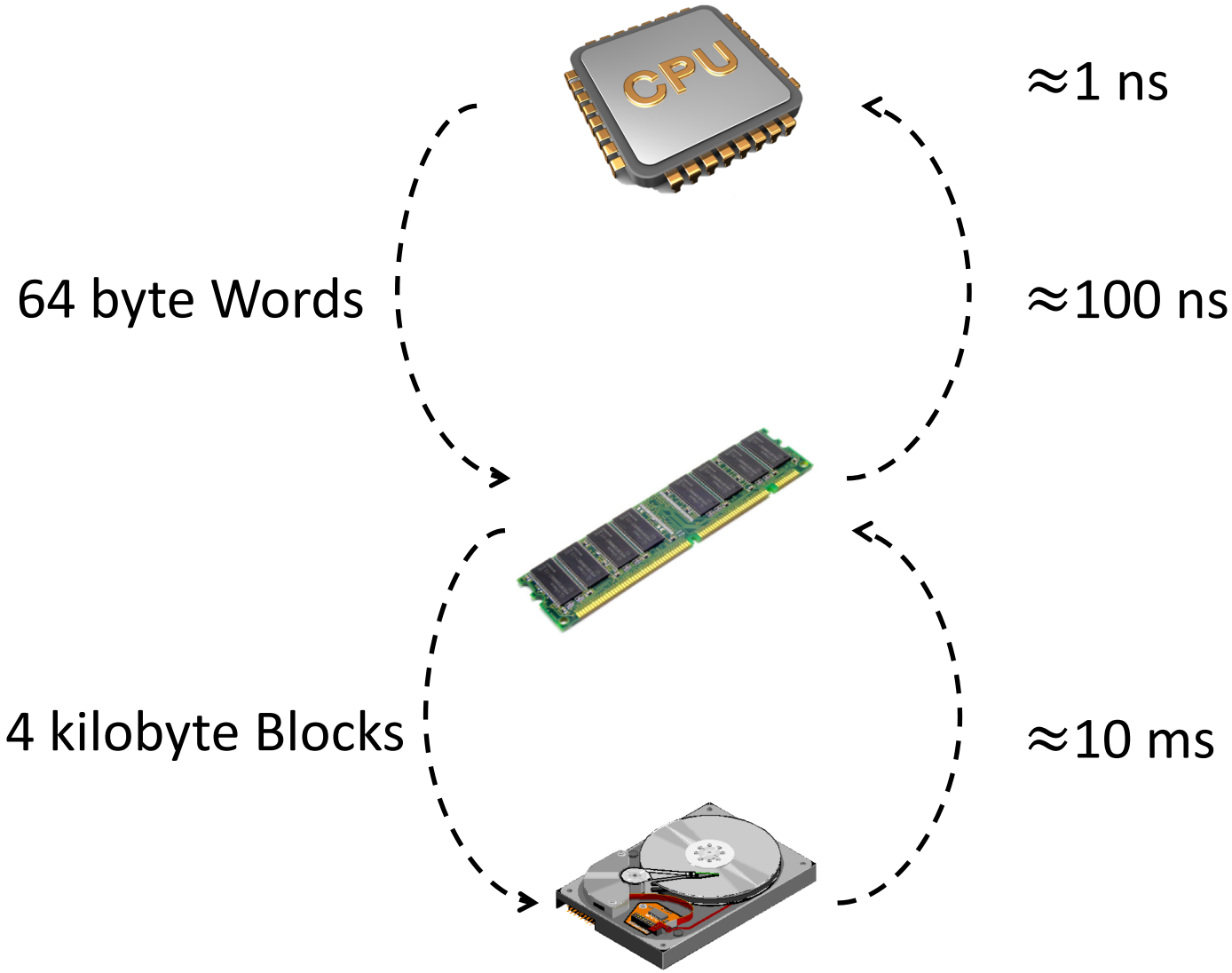
Results Catalogue

Compare and contrast data structures.

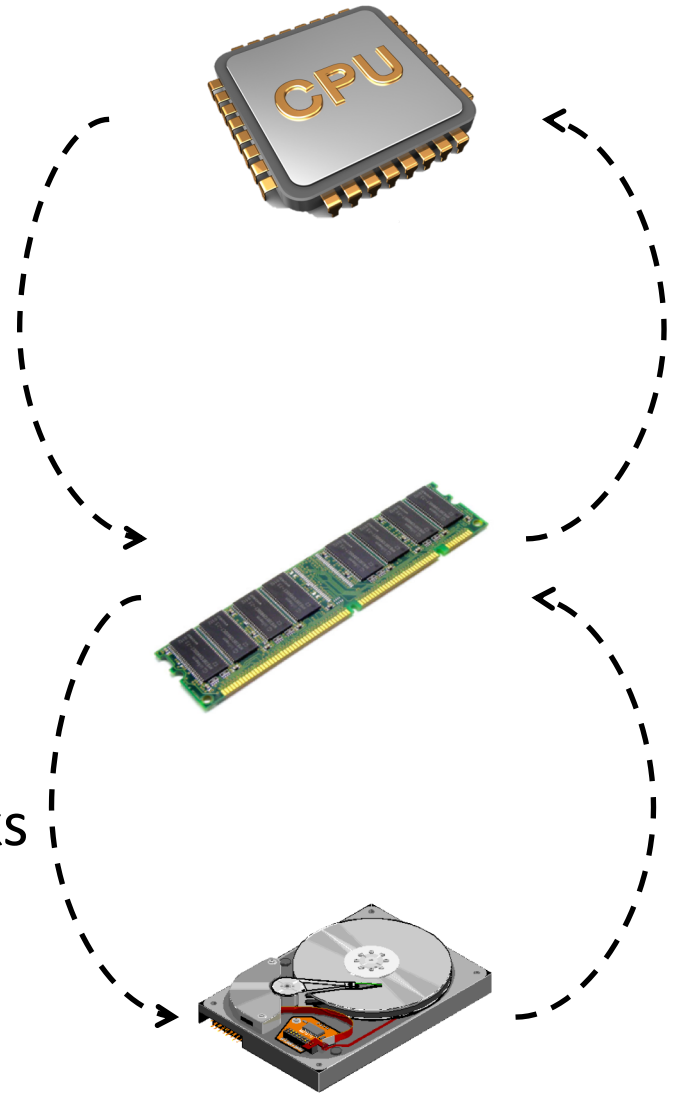
What to use when?

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

Modeling Performance



Modeling Performance



≈1 ns

64 byte Words

≈100 ns

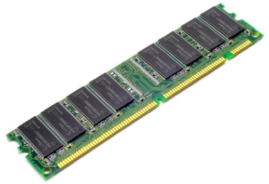
4 kilobyte Blocks

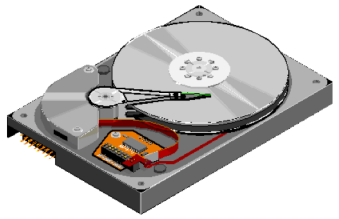
≈10 ms

Measure bottleneck:

Number of block reads/writes (I/O)

Sorted Array





Buffer
James
Sara

Array size	Pointer

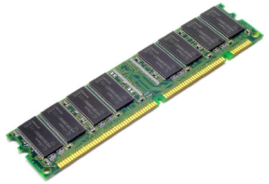
Block 1	Block 2	…	Block N/B
Anne	Bob		Yulia
Arnold	Corrie		Zack
Barbara	Doug		Zelda

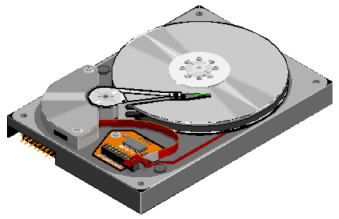
Sorted Array

N entries

B entries fit into a disk block

Array spans **N/B** disk blocks





Buffer
James
Sara

Array size	Pointer

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

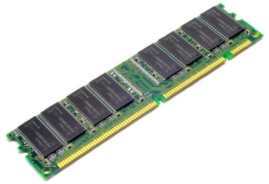
Sorted Array

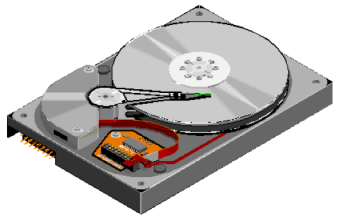
N entries

B entries fit into a disk block

Array spans **N/B** disk blocks

Lookup method & 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

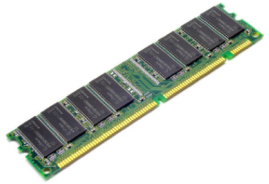
N entries

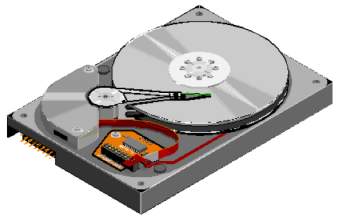
B entries fit into a disk block

Array spans **N/B** disk blocks

Lookup method & cost?

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





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

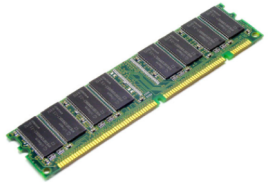
B entries fit into a disk block

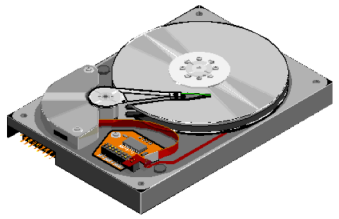
Array spans **N/B** disk blocks

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

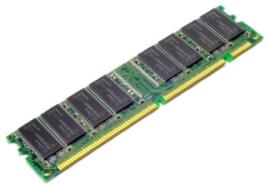
Array spans **N/B** disk blocks

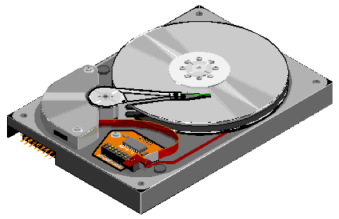
Lookup method & cost?

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





Buffer
James
Sara

Array size	Pointer

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

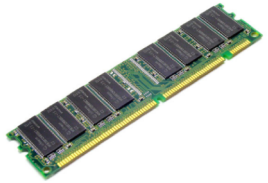
Results Catalogue

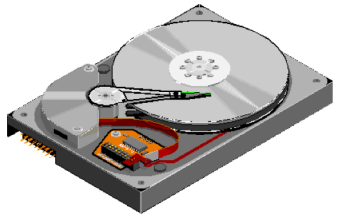
	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N/B))$	$O(N/B^2)$
Log		
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue

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

Log (append-only array)





Buffer
James
Sara

Array size	Pointer

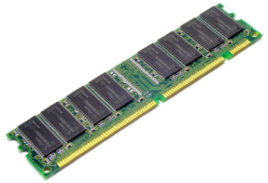
Block 1	Block 2	…	Block N/B
Doug	Yulia		Anne
Zelda	Zack		Bob
Arnold	Barbara		Corrie

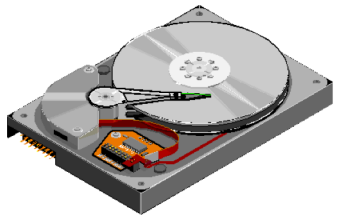
Log (append-only array)

N entries

B entries fit into a disk block

Array spans **N/B** disk blocks





Buffer
James
Sara

Array size	Pointer

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

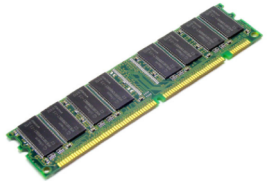
Log (append-only array)

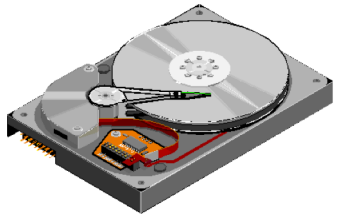
N entries

B entries fit into a disk block

Array spans **N/B** disk blocks

Lookup method & cost?





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)

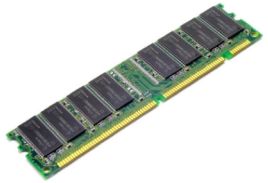
N entries

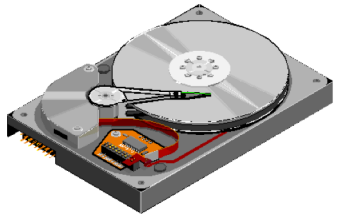
B entries fit into a disk block

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

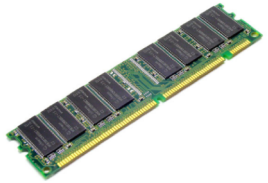
B entries fit into a disk block

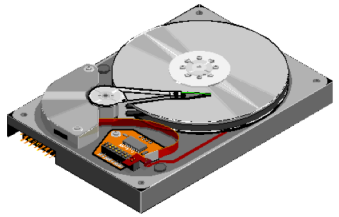
Array spans **N/B** disk blocks

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

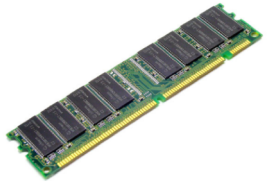
Array spans **N/B** disk blocks

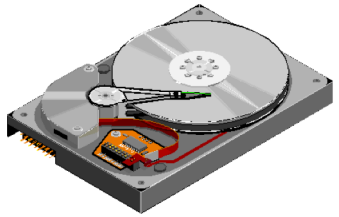
Lookup method & cost?

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

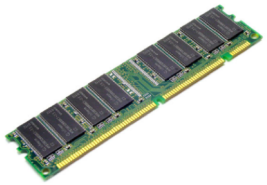
Results Catalogue

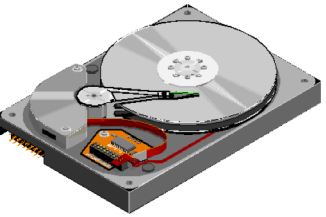
	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N/B))$	$O(N/B^2)$
Log	$O(N/B)$	$O(1/B)$
B-tree		
Basic LSM-tree		
Leveled LSM-tree		
Tiered LSM-tree		

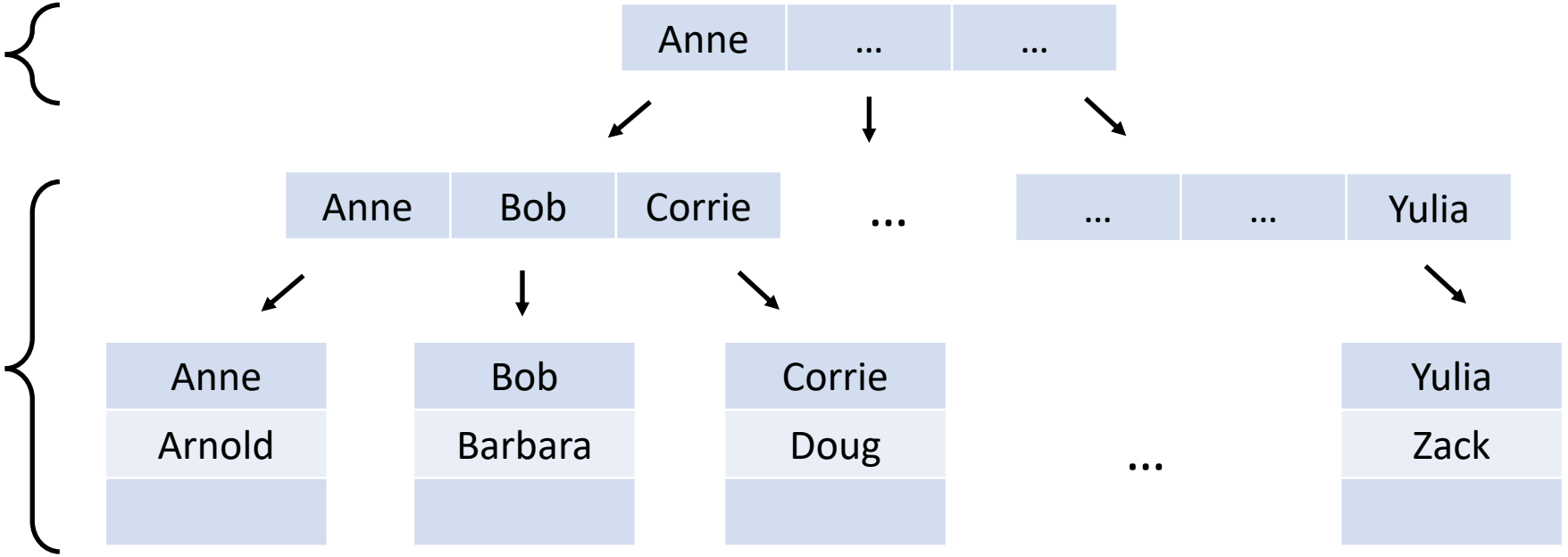
Results Catalogue

	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

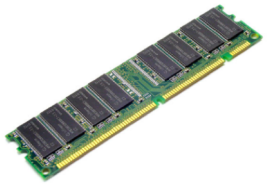


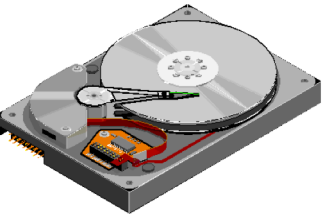


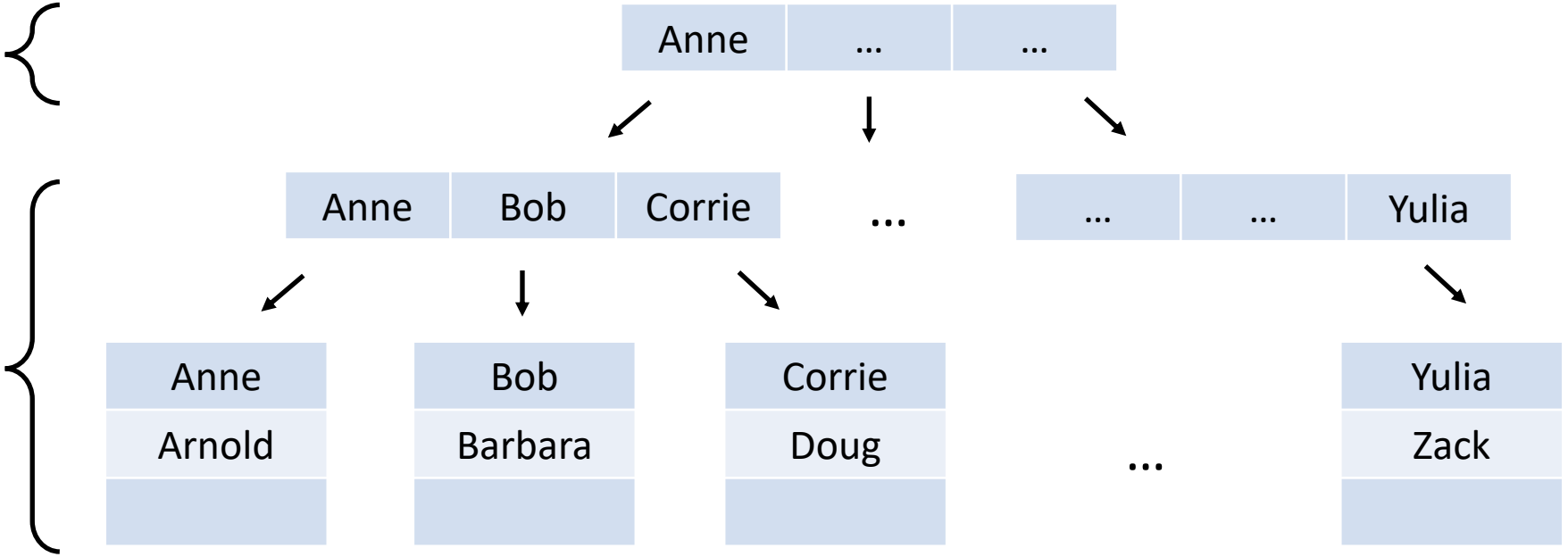

Anne
…
…
Anne
Bob
Corrie
…
…
…
Yulia
Anne
Arnold
Bob
Barbara
Corrie
Doug
…
Yulia
Zack

B-tree

Lookup method & cost?

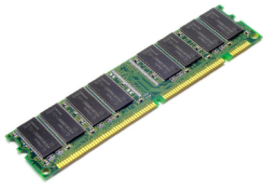


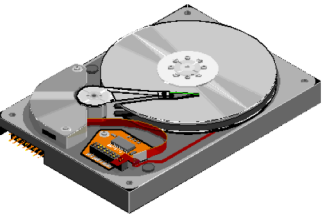


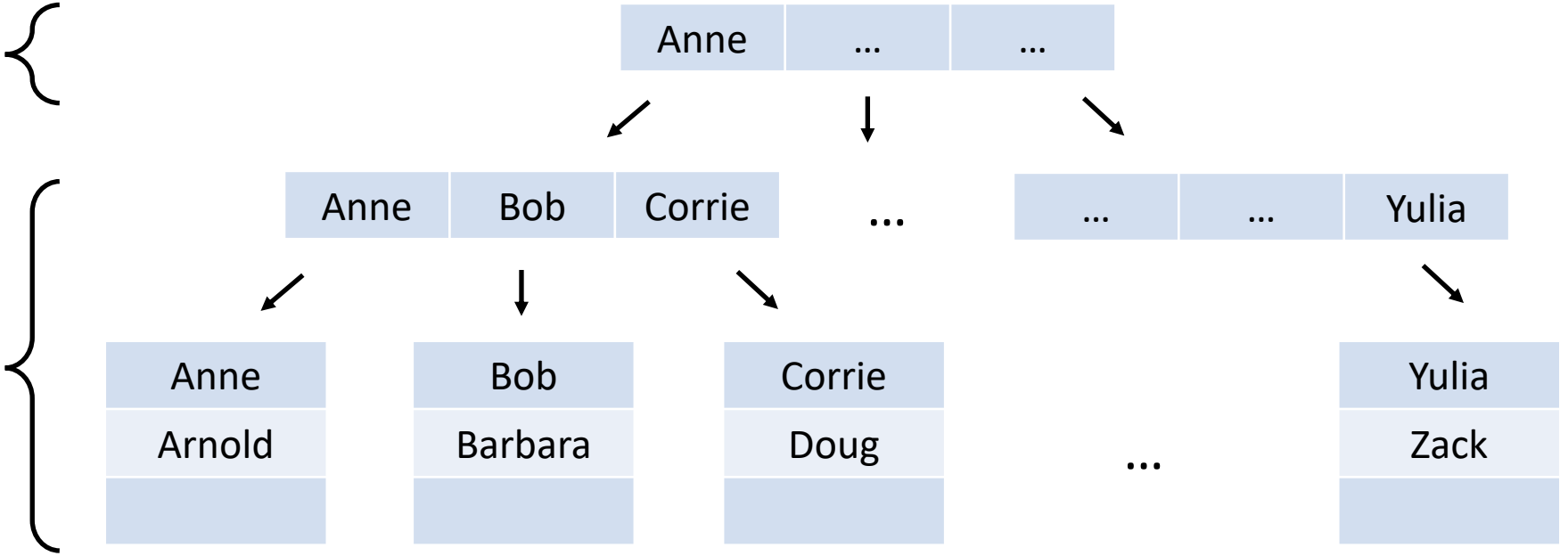


B-tree

Lookup method & cost?



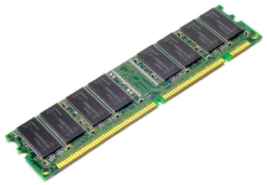


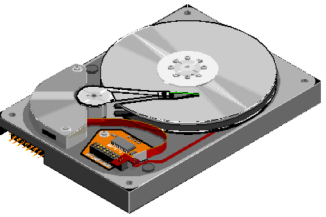


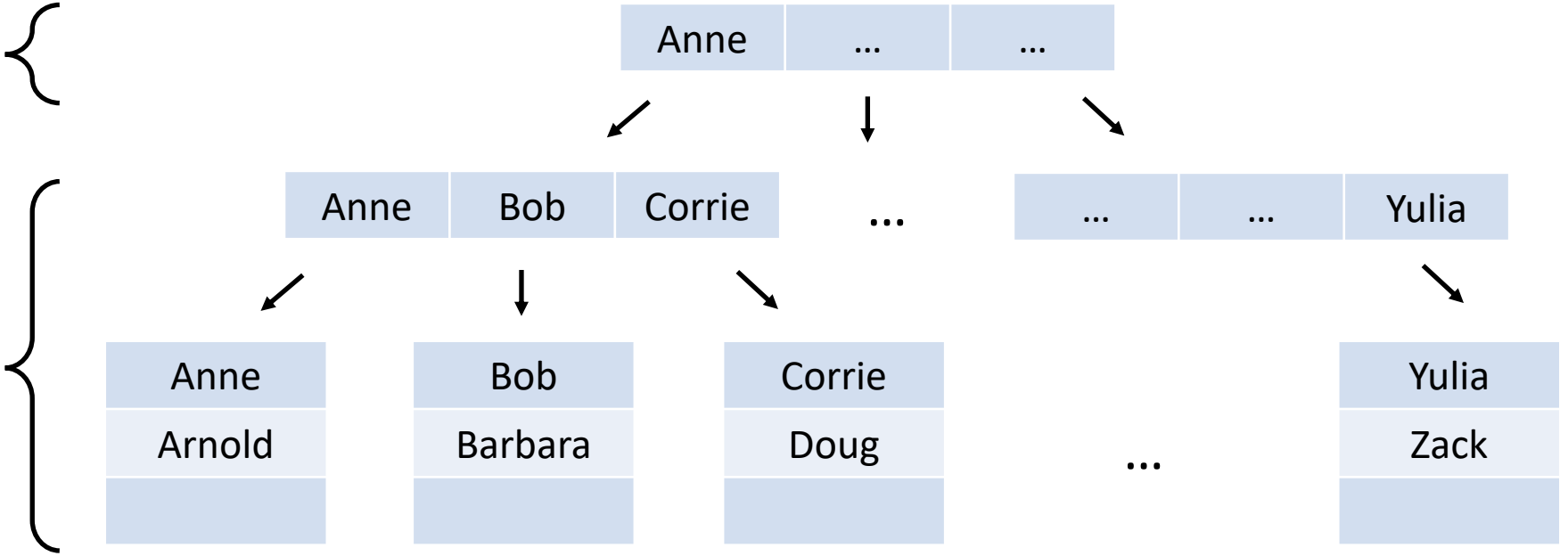
B-tree

Lookup method & cost?

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





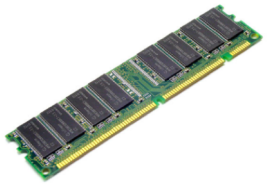


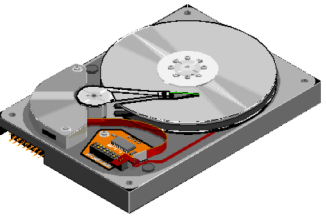
B-tree

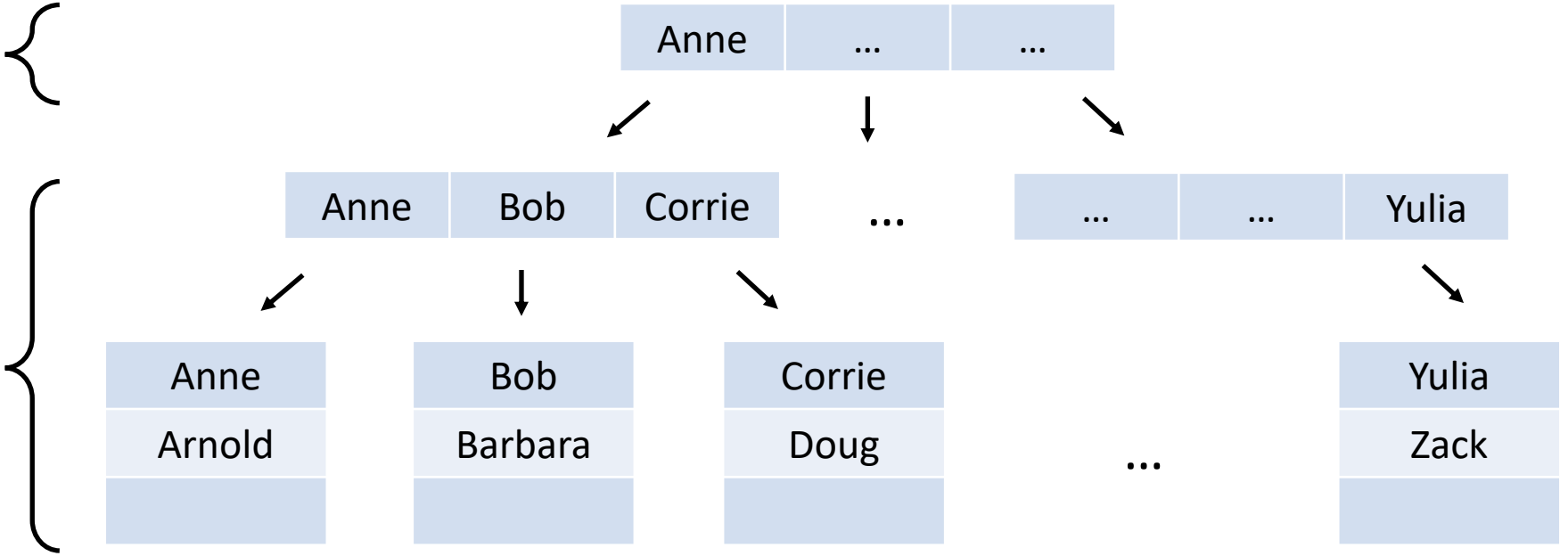
Lookup method & cost?

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

Insertion method & cost?







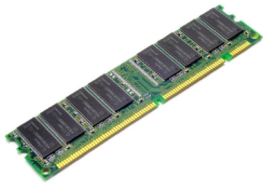
B-tree

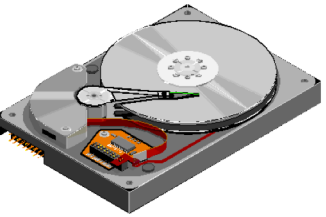
Lookup method & cost?

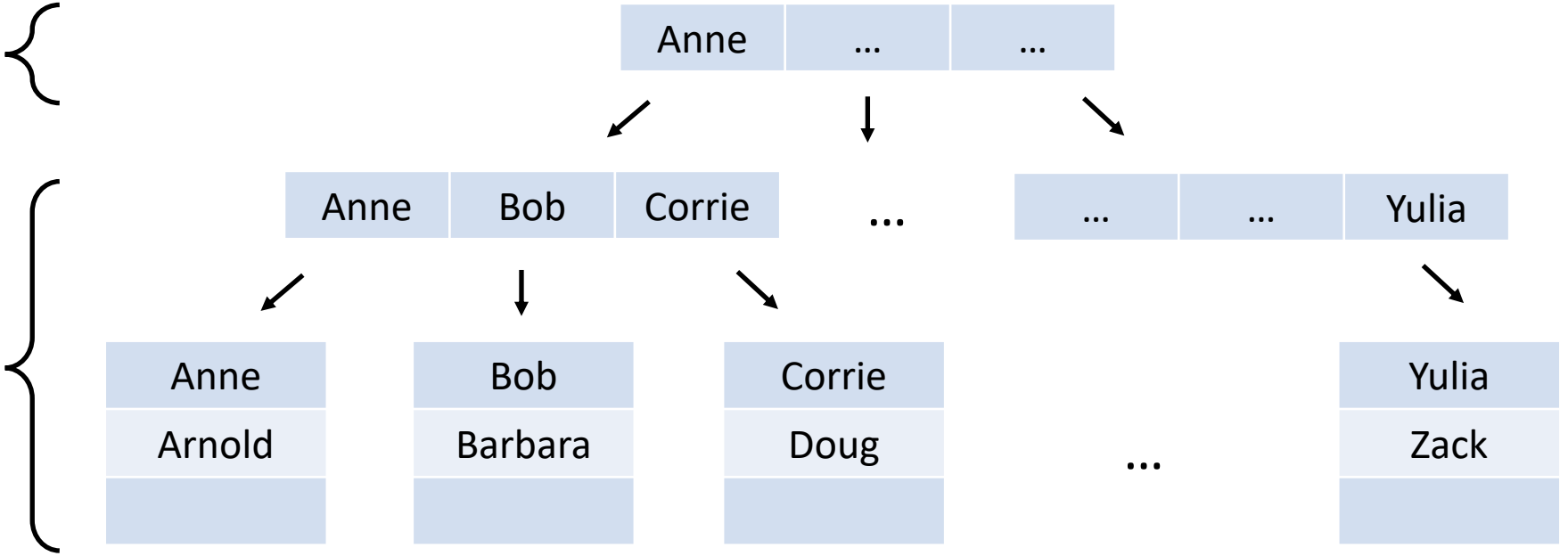
Tree search: $O\left(\log_B\left(\frac{N}{B}\right)\right)$

Insertion method & cost?

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







Results Catalogue

	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		

B-trees

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

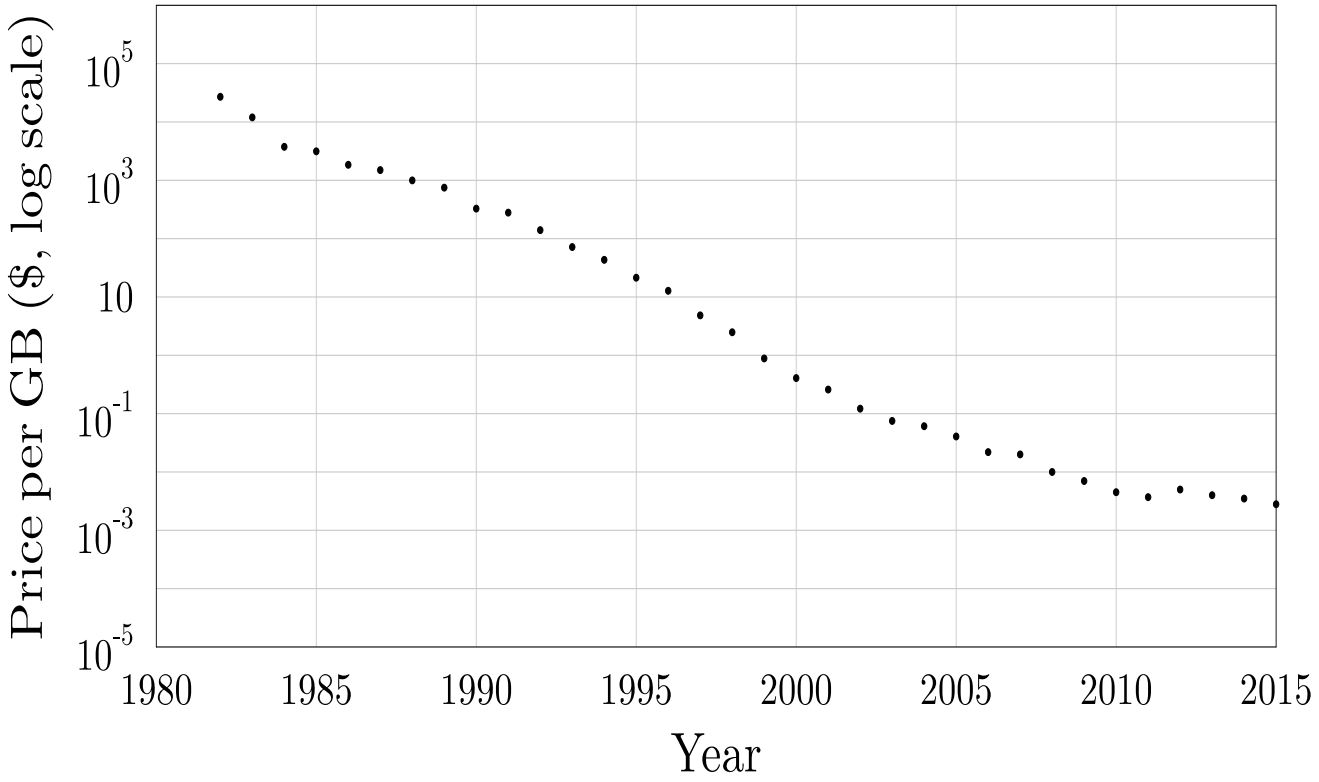
"It could be said that the world's information is at our fingertips because of B-trees"

B-trees are no longer sufficient

Cheaper to store data

Workloads more insert-intensive

We need better insert-performance.



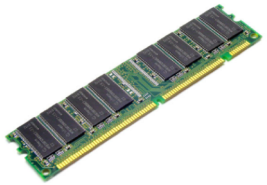
Results Catalogue

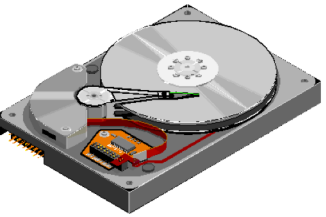
Goal to combine sub-constant insertion cost
logarithmic lookup cost

	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





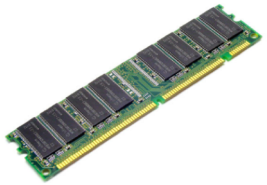
Level

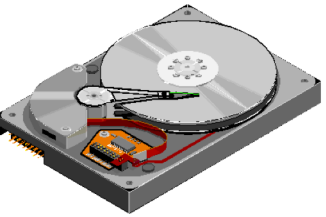
Buffer { 0

Sorted arrays { 1, 2, 3

Basic LSM-tree

Design principle #1: optimize for insertions by buffering





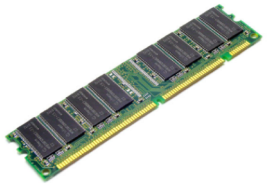
Level

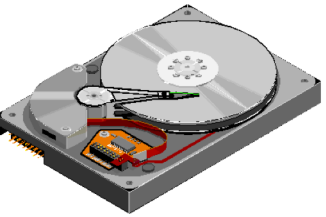
Buffer { 0

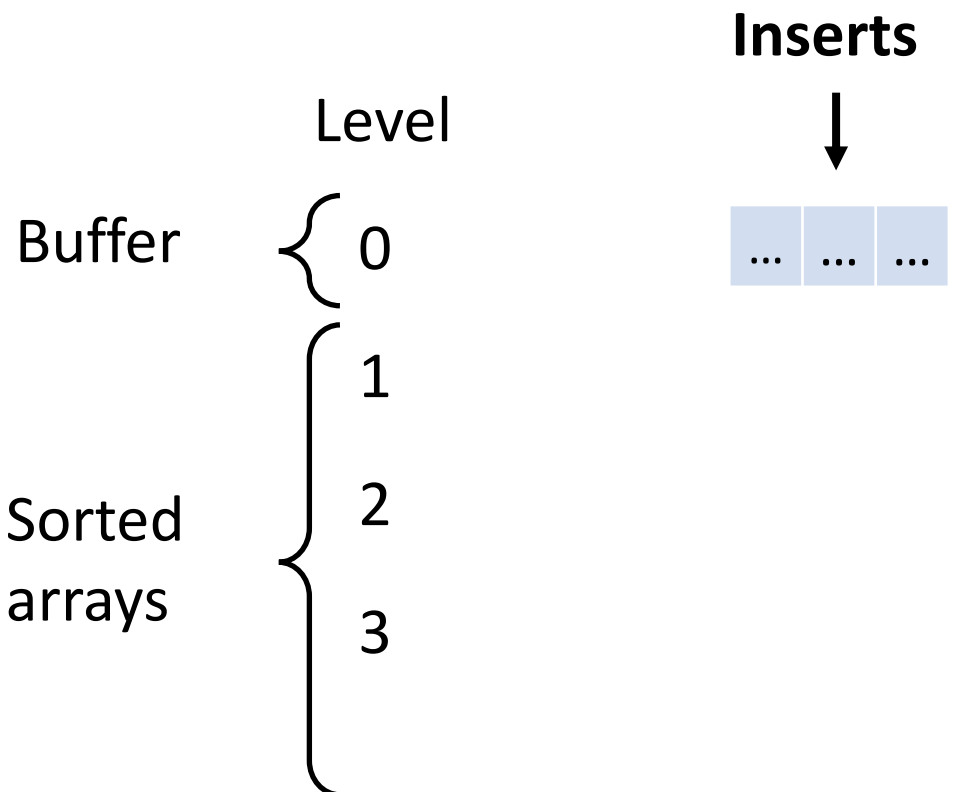
Sorted arrays { 1, 2, 3

Basic LSM-tree

Design principle #1: optimize for insertions by buffering

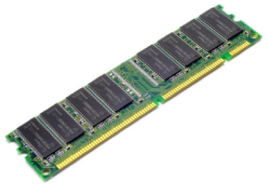


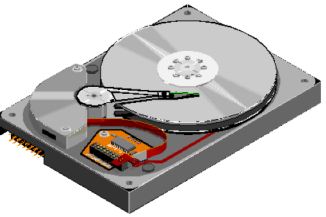


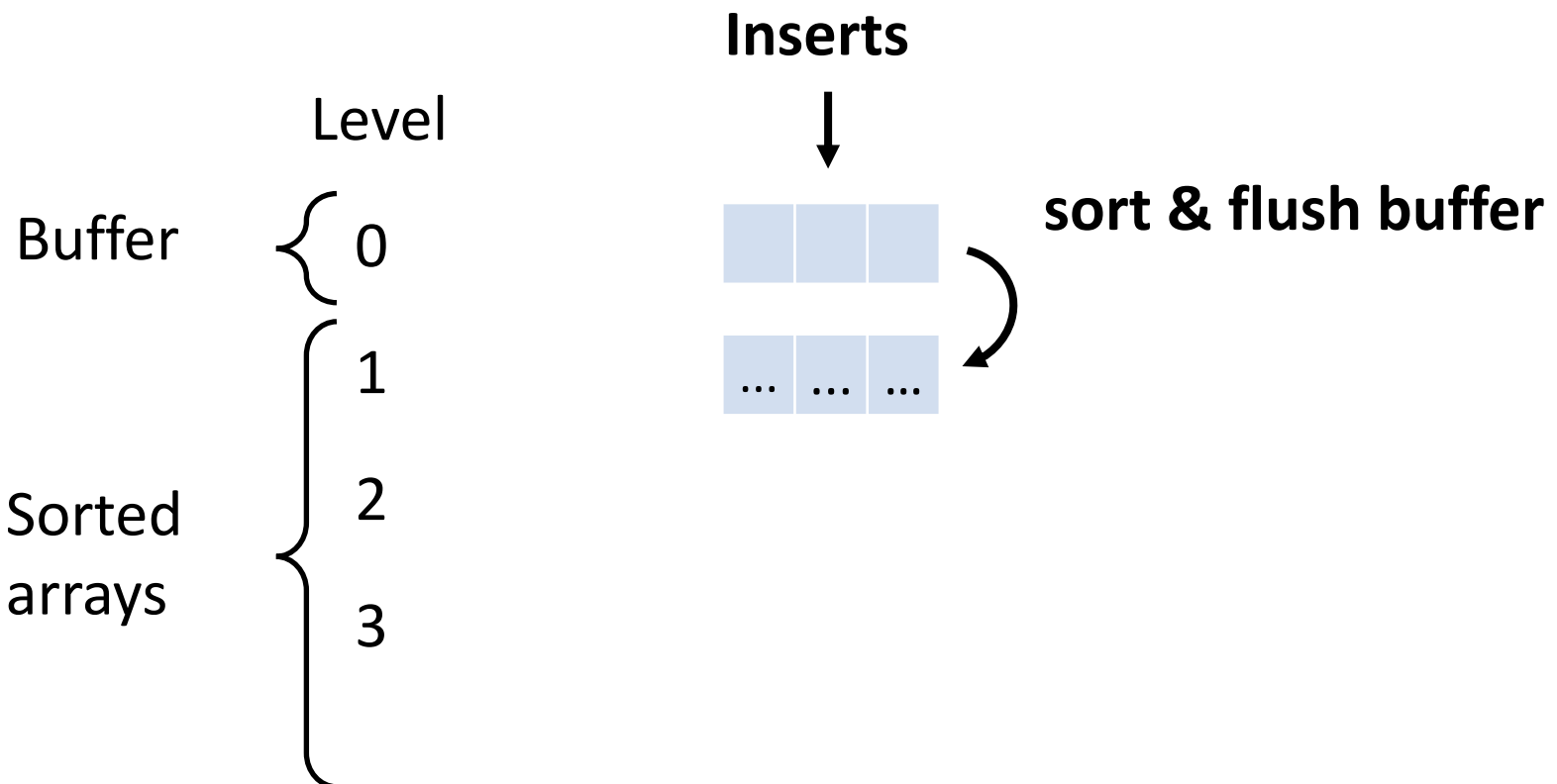


Basic LSM-tree

Design principle #1: optimize for insertions by buffering

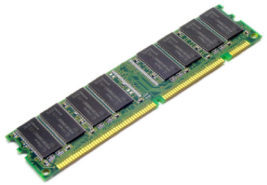


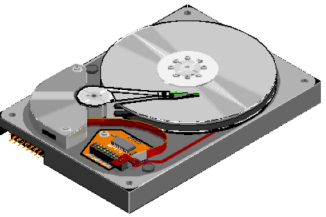


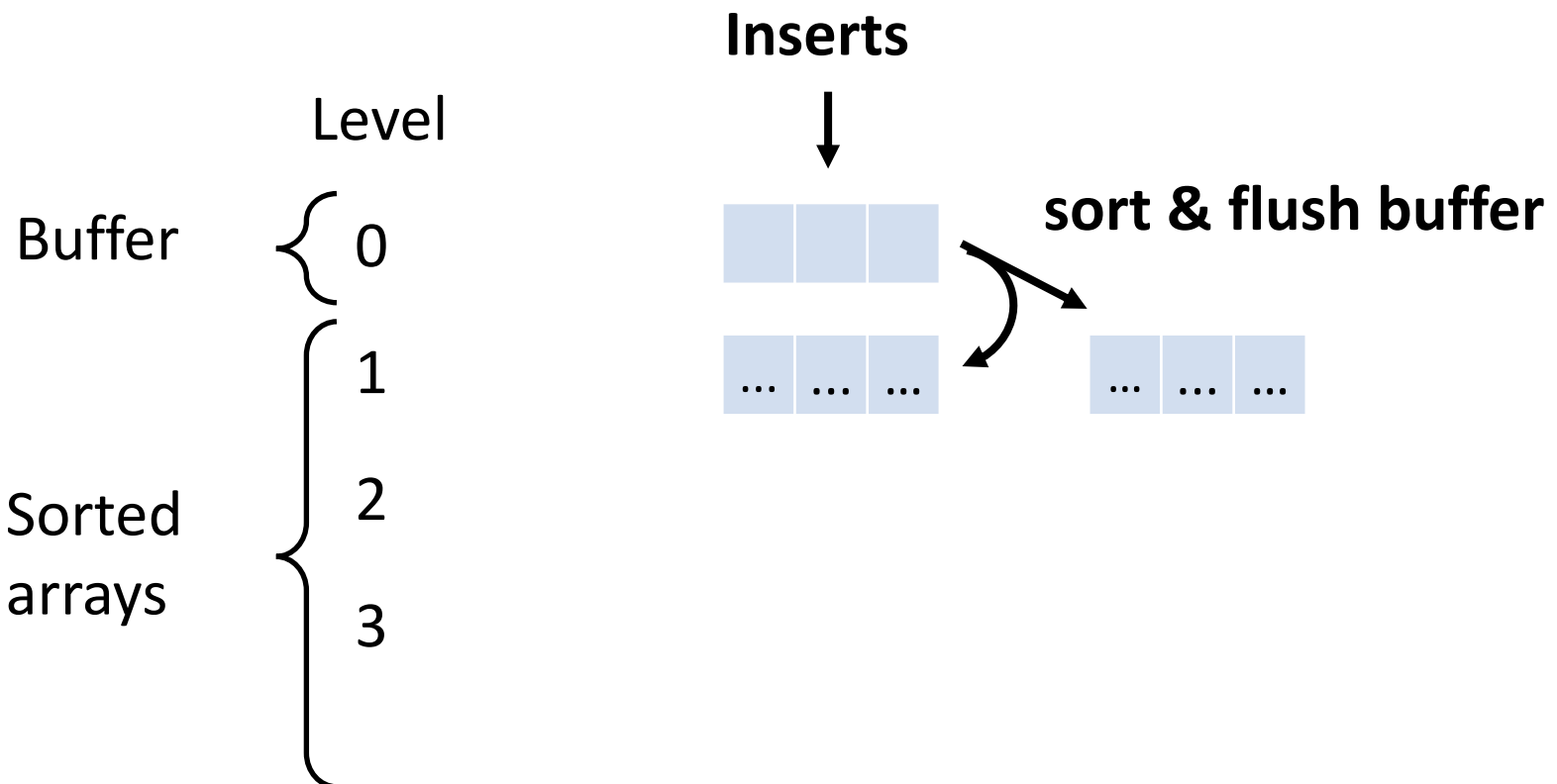


Basic LSM-tree

Design principle #1: optimize for insertions by buffering



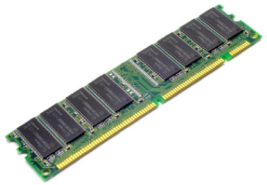


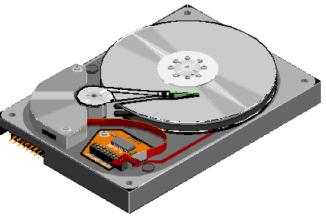


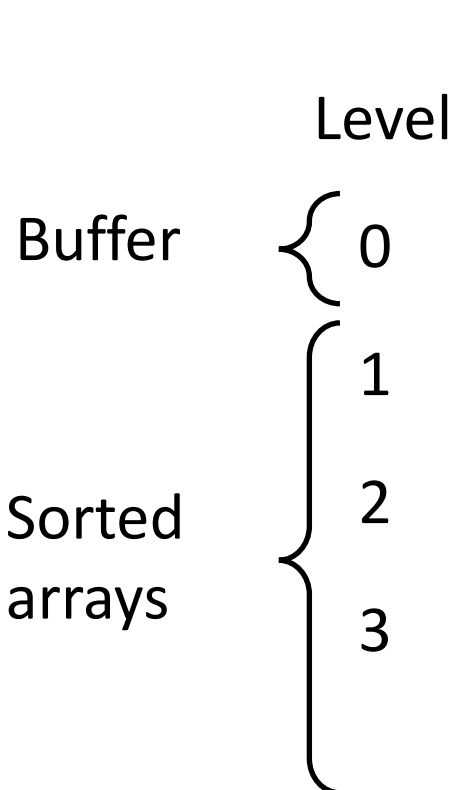
Basic LSM-tree

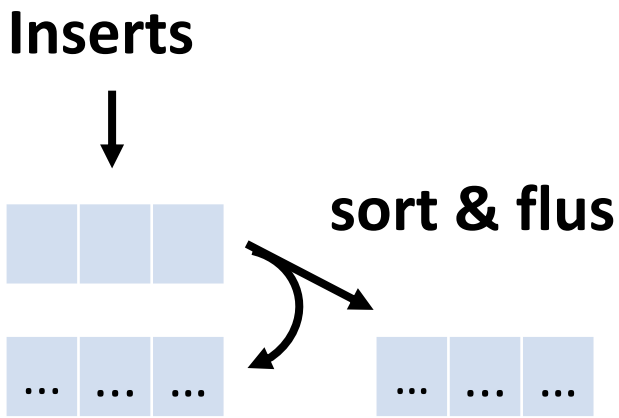
Design principle #1: optimize for insertions by buffering

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





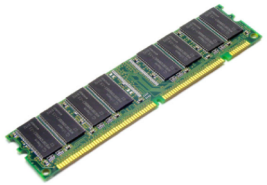


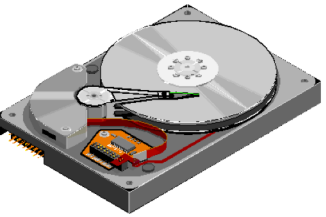


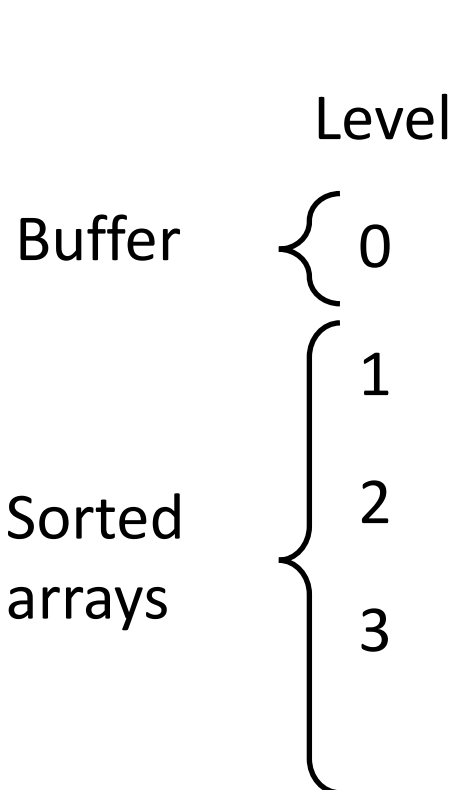
Basic LSM-tree

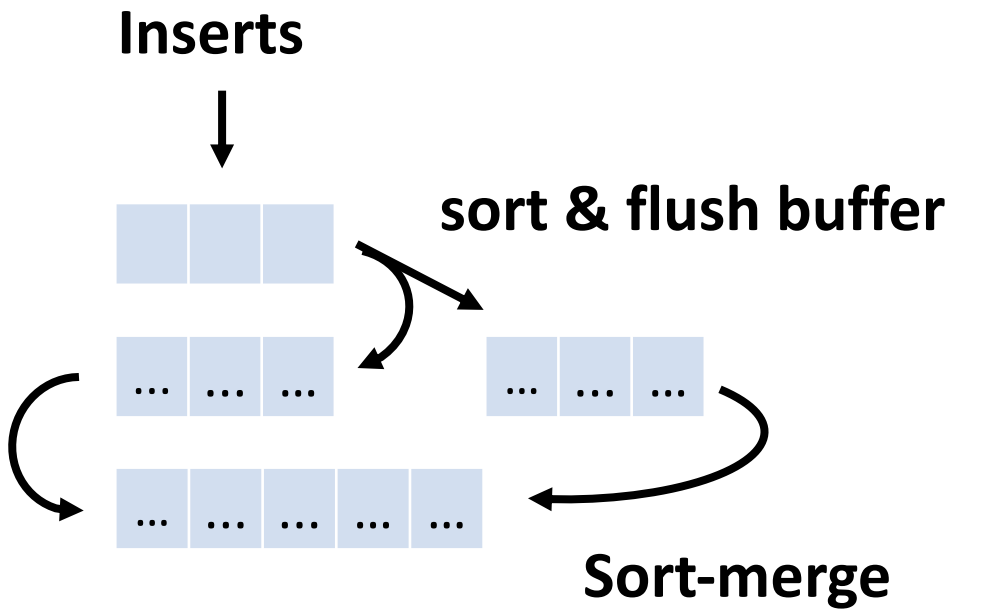
Design principle #1: optimize for insertions by buffering

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





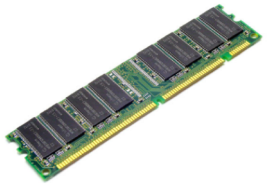


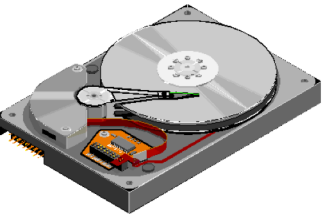


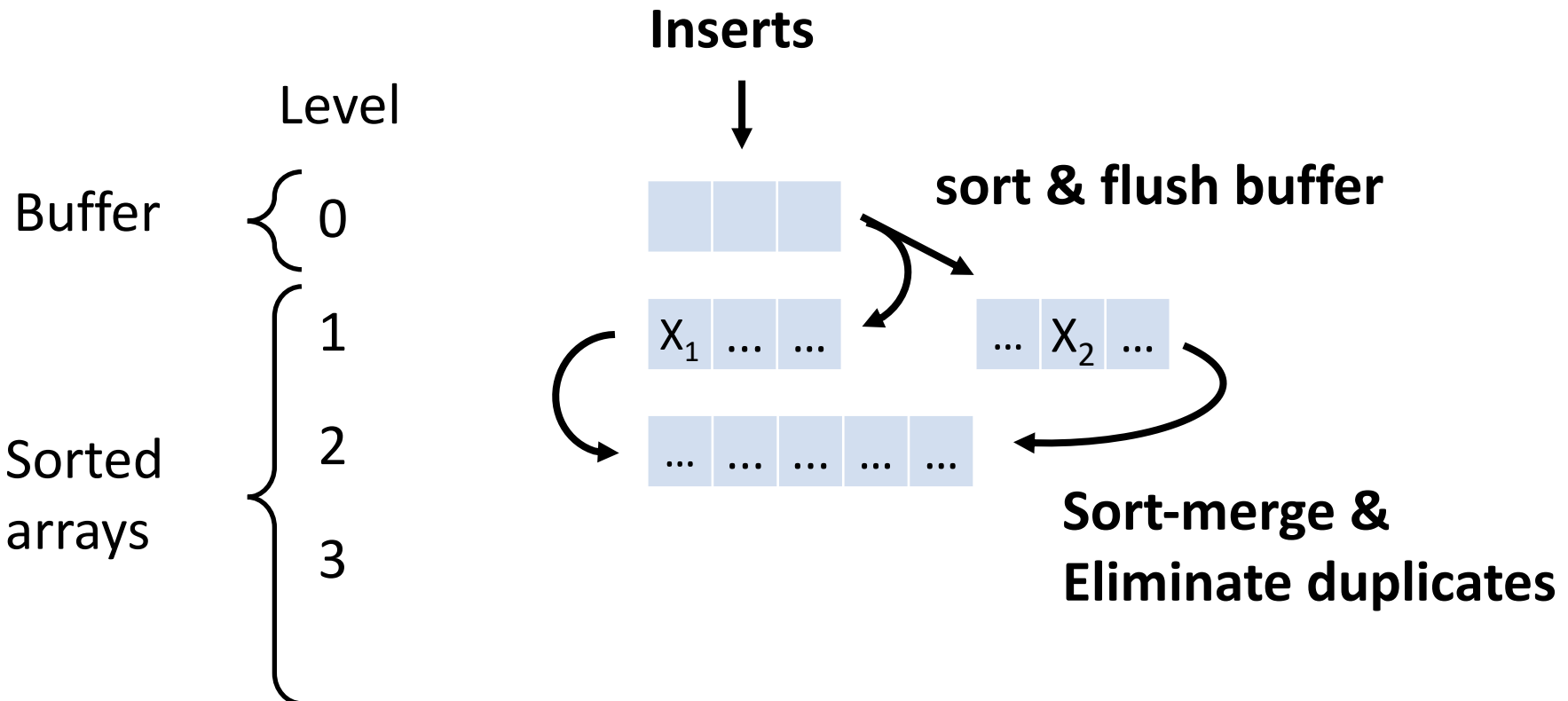
Basic LSM-tree

Design principle #1: optimize for insertions by buffering

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



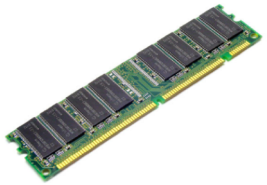


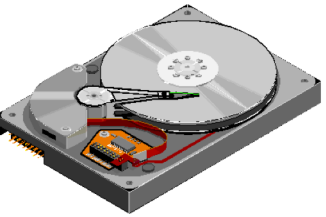


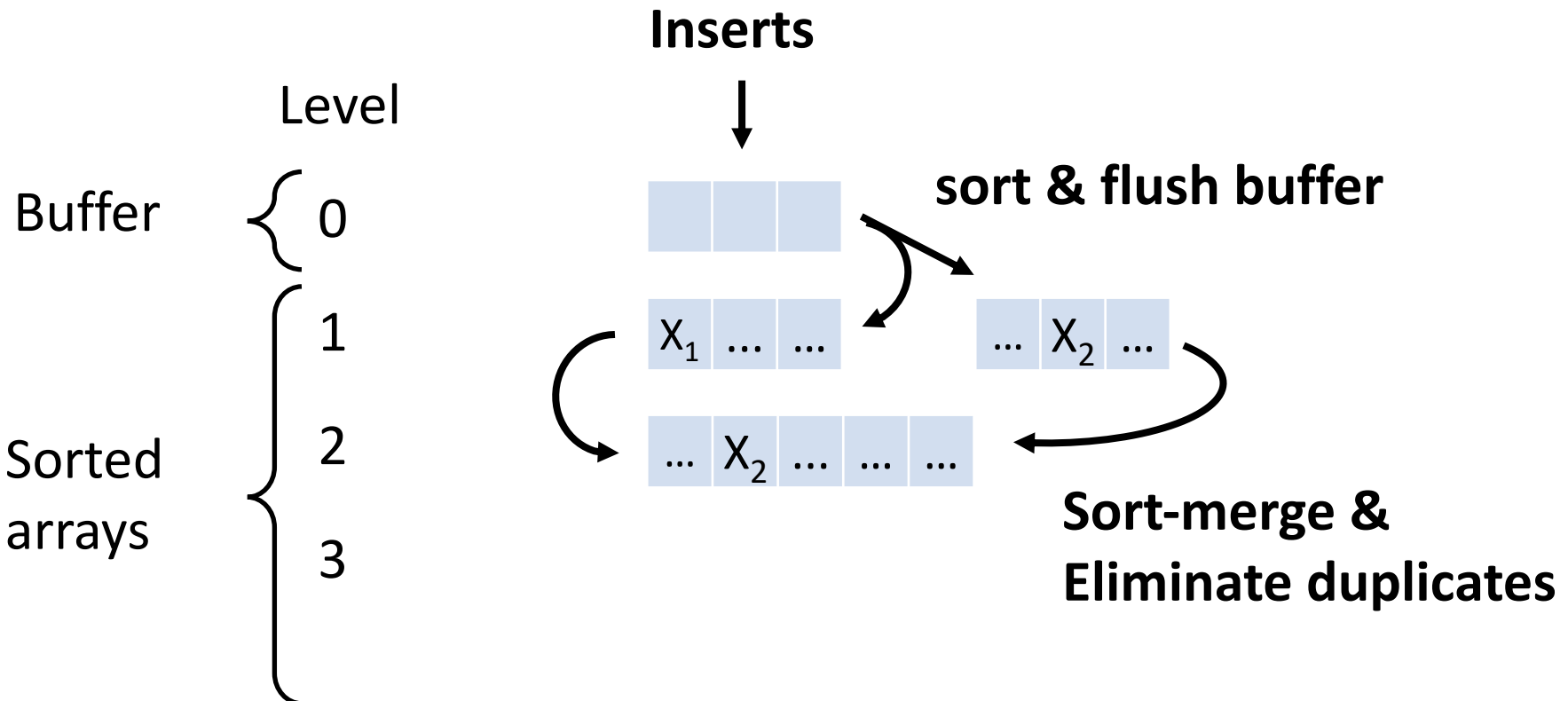
Basic LSM-tree

Design principle #1: optimize for insertions by buffering

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



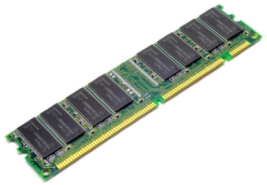


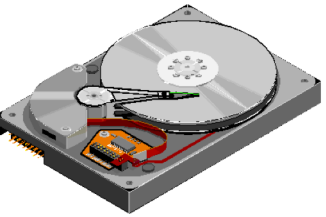


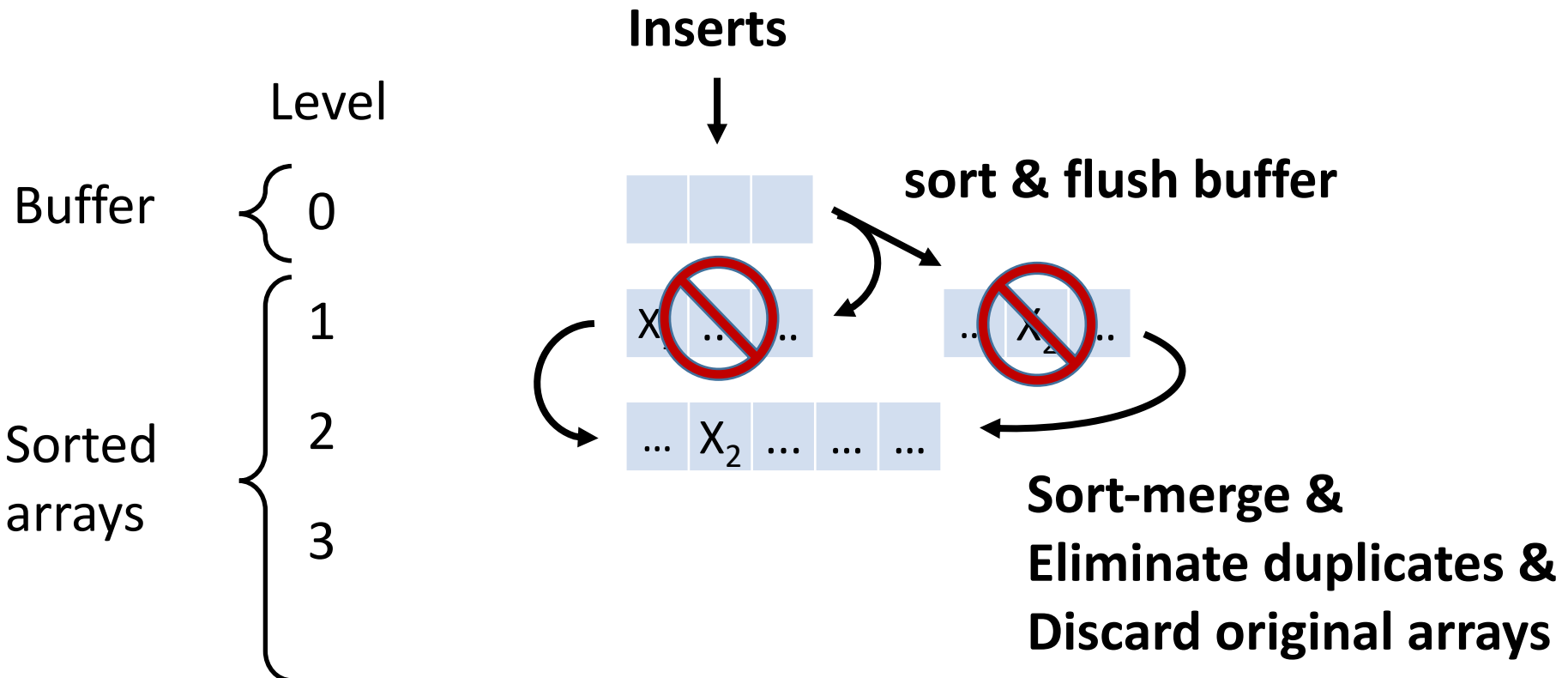
Basic LSM-tree

Design principle #1: optimize for insertions by buffering

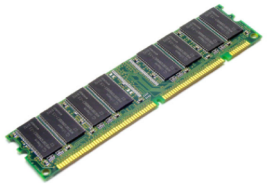
Design principle #2: optimize for lookups by sort-merging arrays

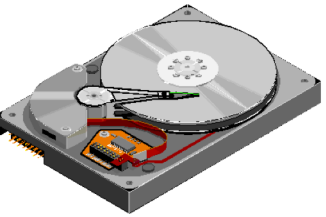






Basic LSM-tree – Example



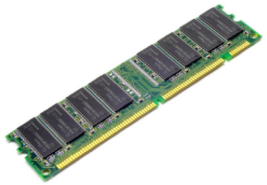


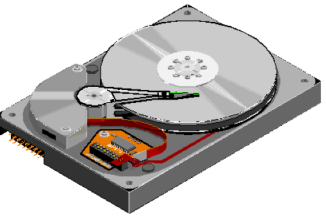
Level

Buffer { 0

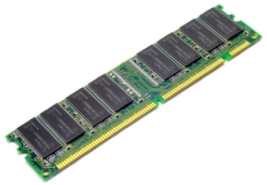
Sorted arrays { 1, 2, 3

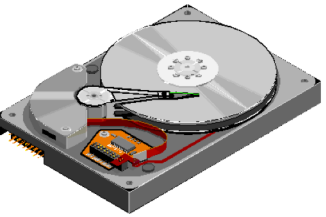
Basic LSM-tree – Example

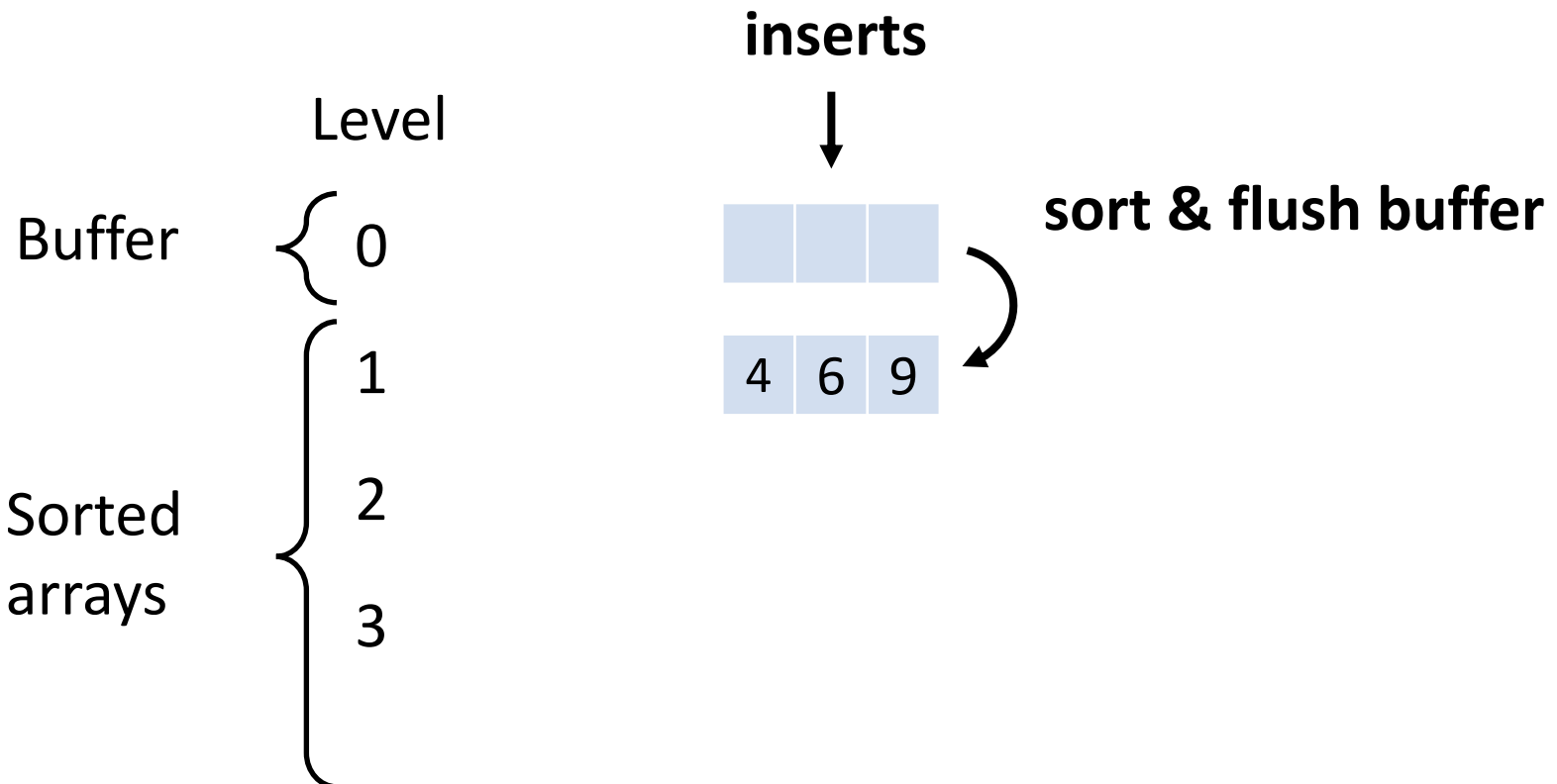




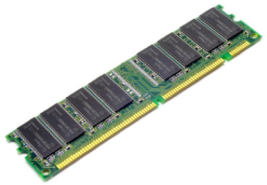
Basic LSM-tree – Example

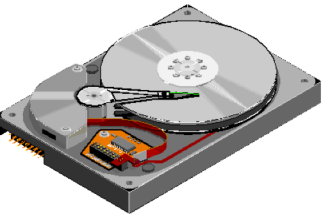




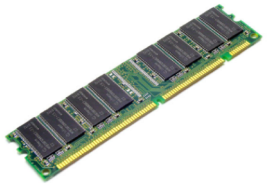


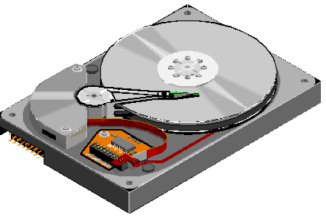
Basic LSM-tree – Example

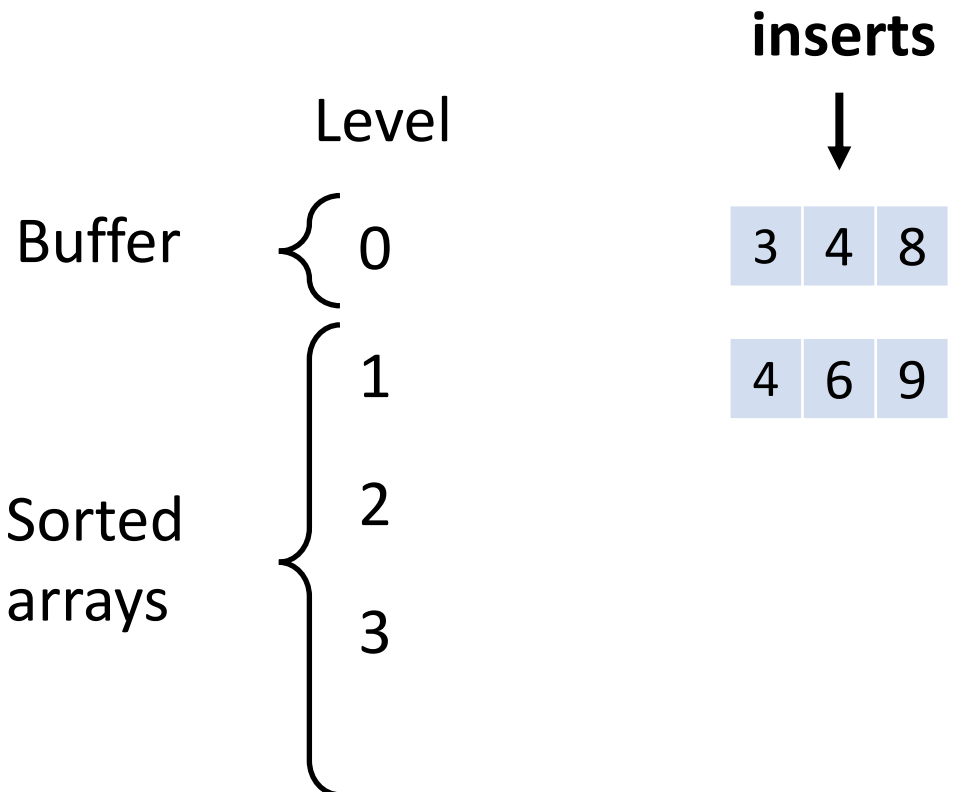




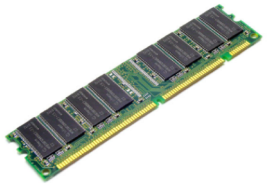
Basic LSM-tree – Example

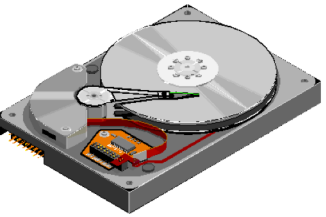


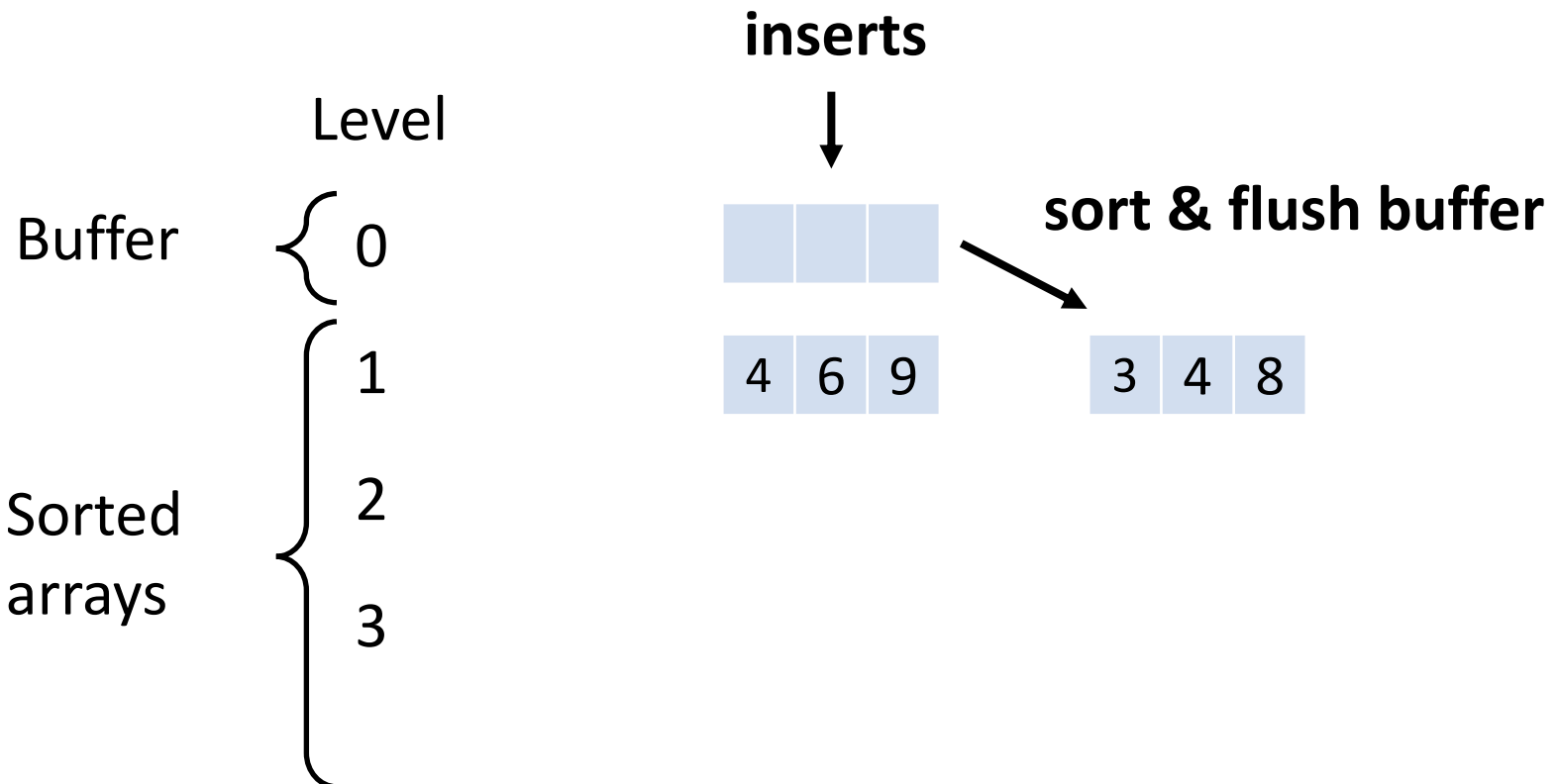




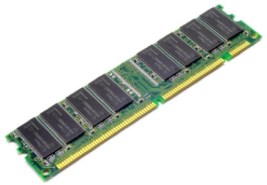
Basic LSM-tree – Example

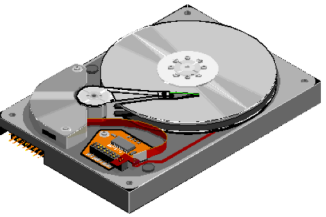




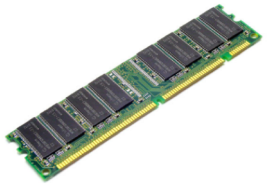


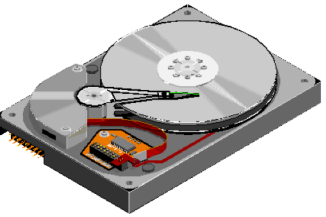
Basic LSM-tree – Example

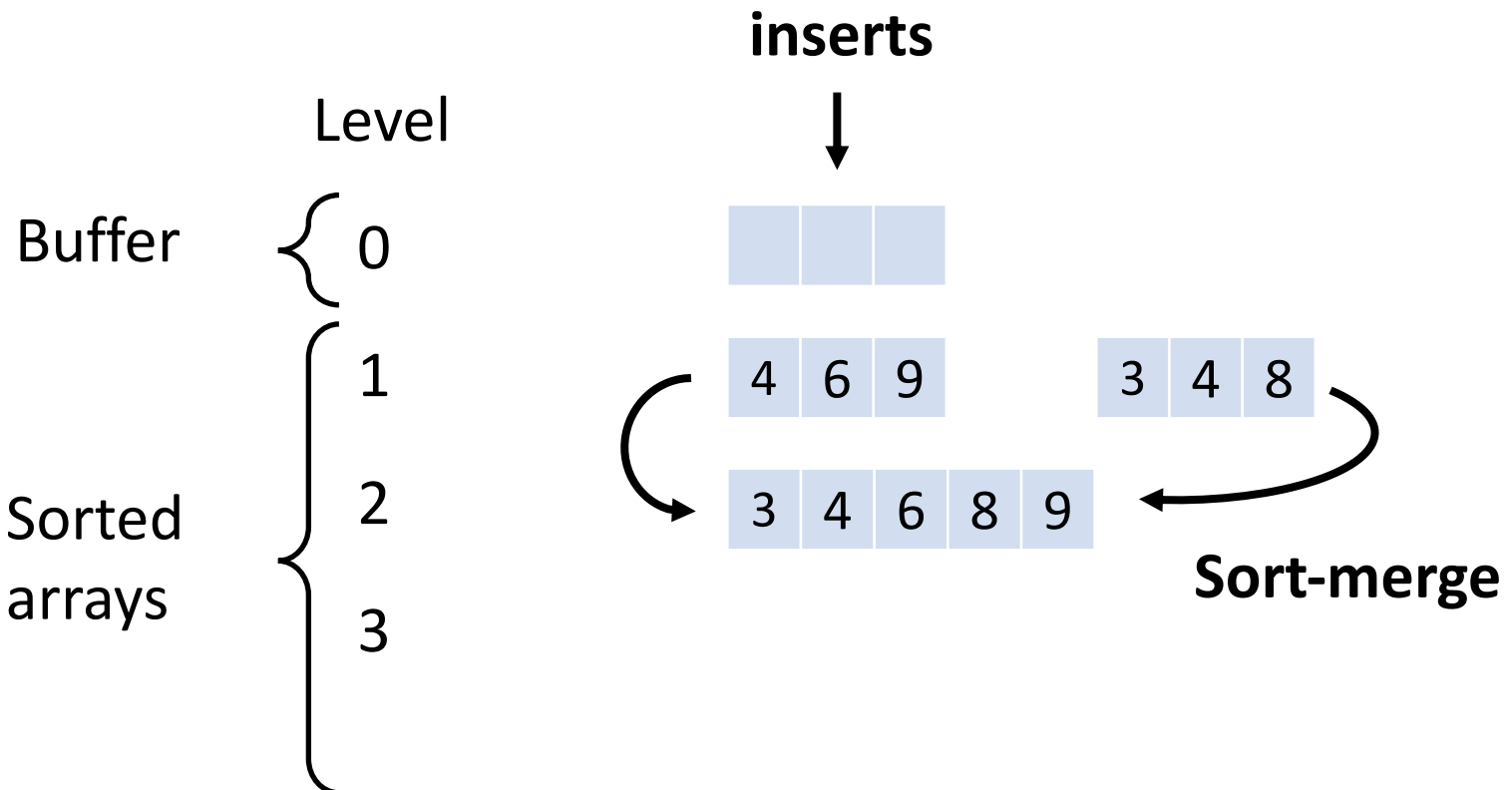




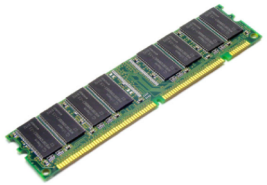
Basic LSM-tree – Example

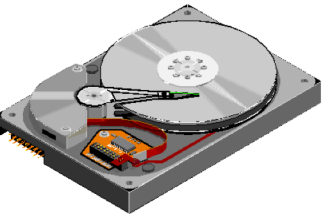


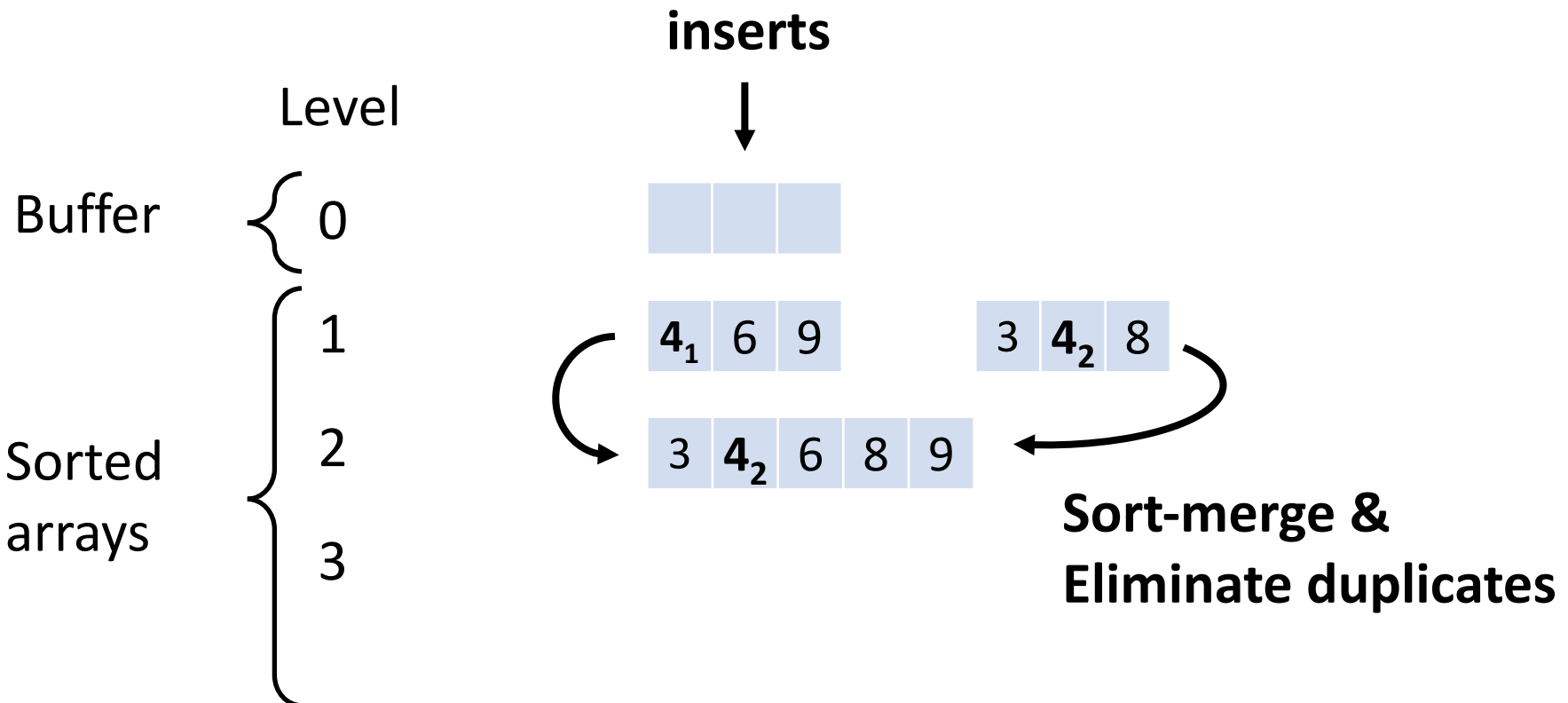




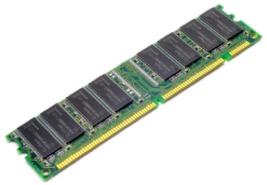
Basic LSM-tree – Example

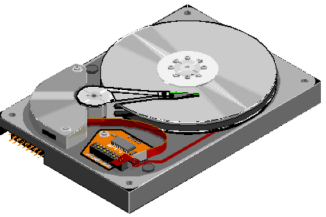






Basic LSM-tree – Example





Buffer

Sorted arrays

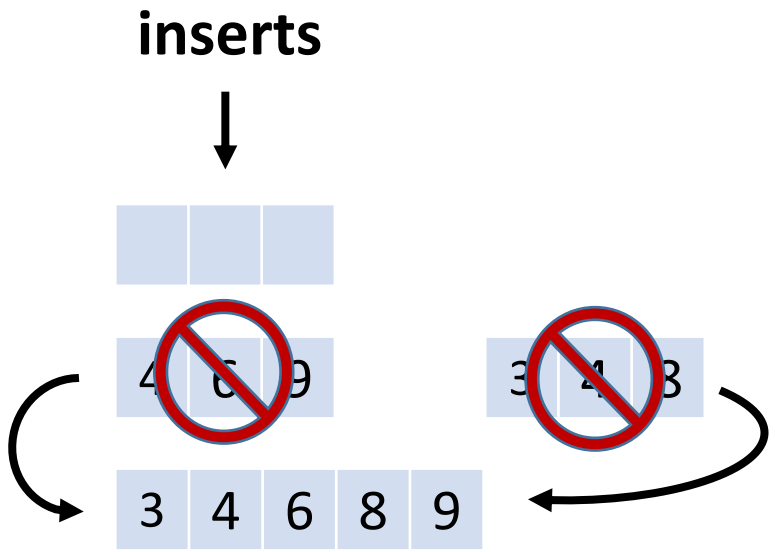
Level

0

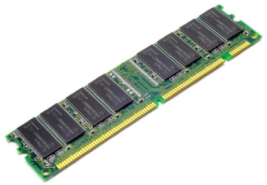
1

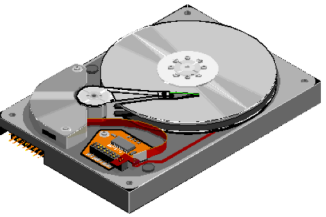
2

3



Basic LSM-tree – Example





Level

Buffer

0

Sorted arrays

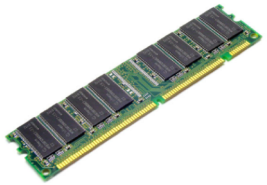
1

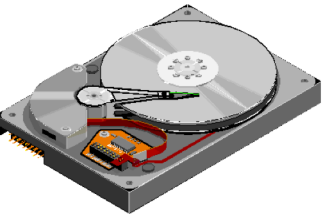
2

3

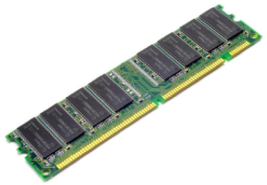
inserts

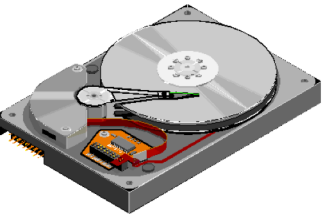
Basic LSM-tree – Example

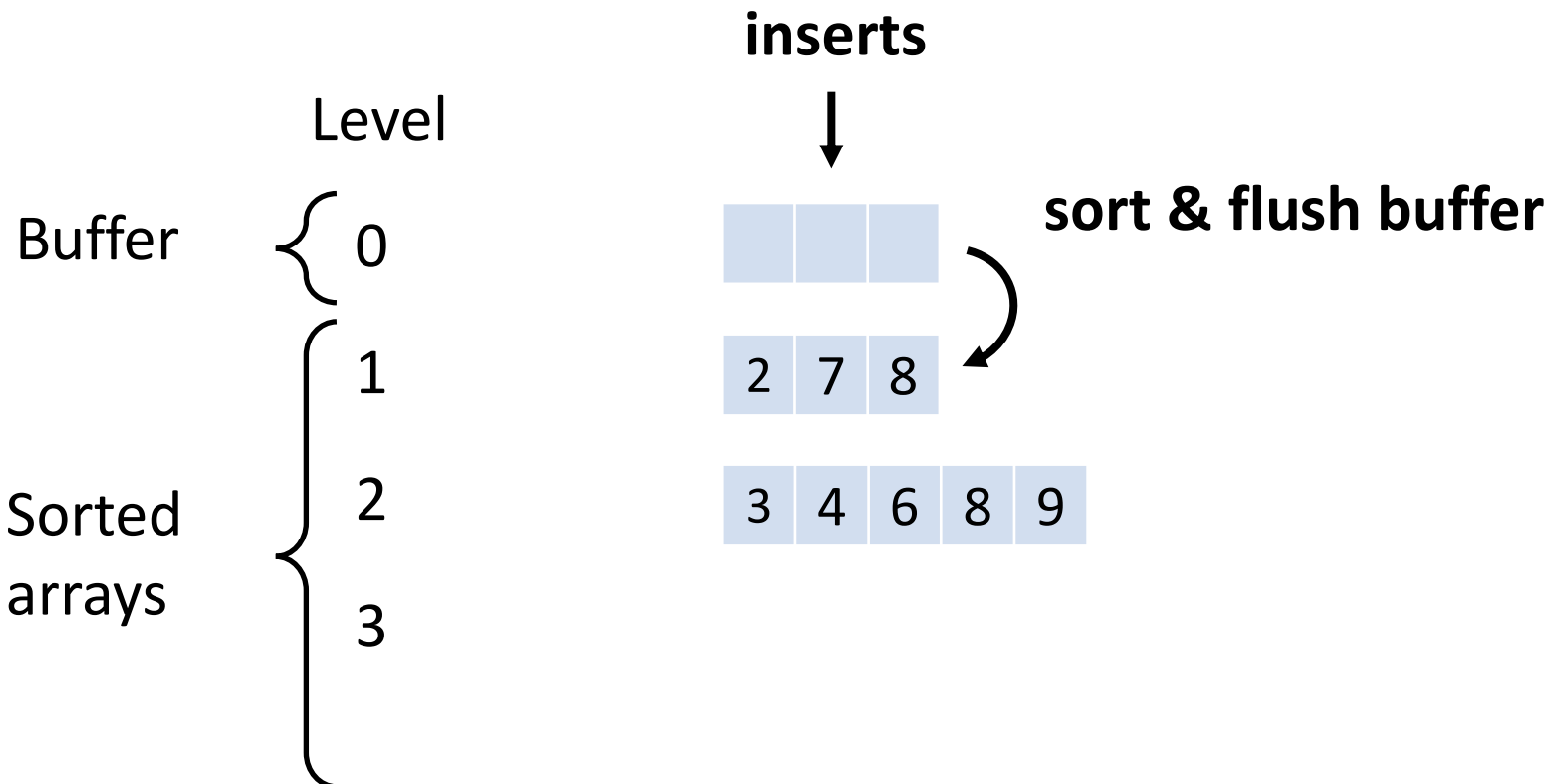




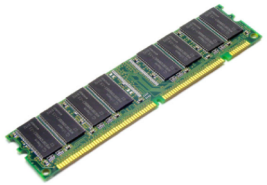
Basic LSM-tree – Example

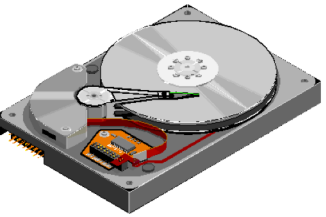






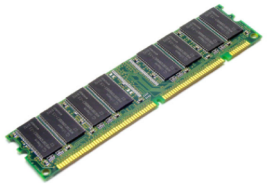
Basic LSM-tree – Example

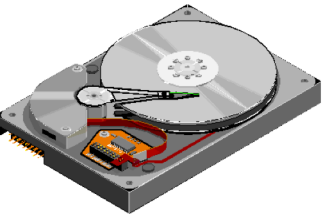


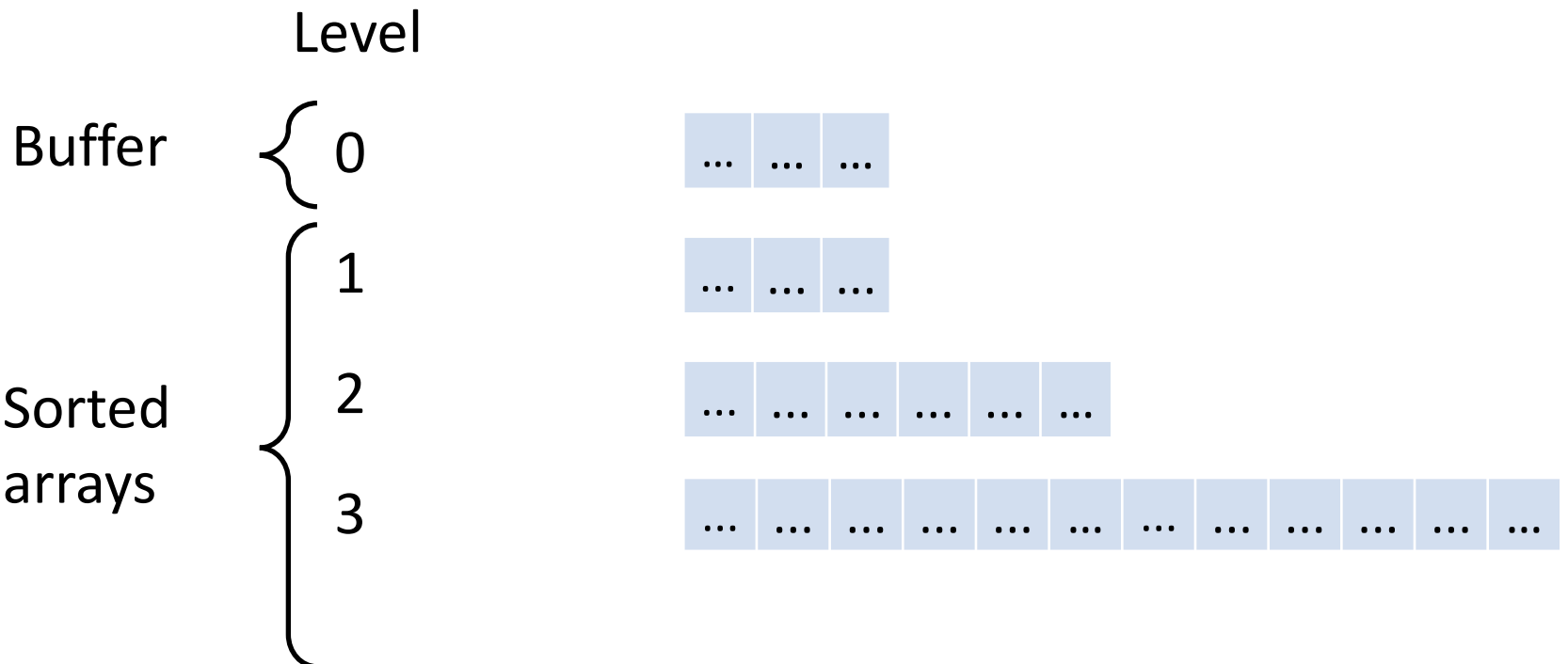


Basic LSM-tree

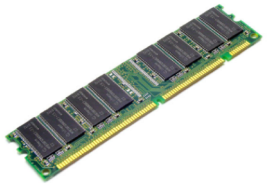
Levels have exponentially increasing capacities.

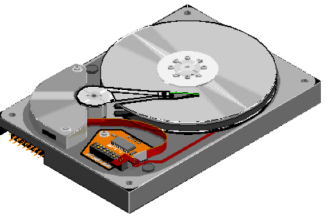






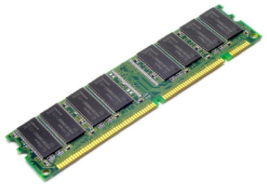
Basic LSM-tree – Lookup cost

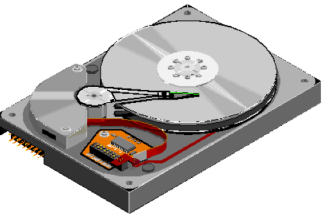




Basic LSM-tree – Lookup cost

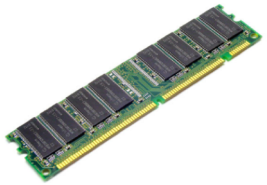
Lookup method?

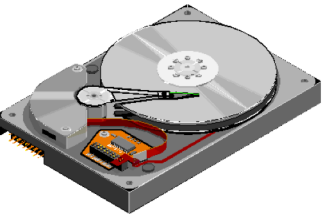


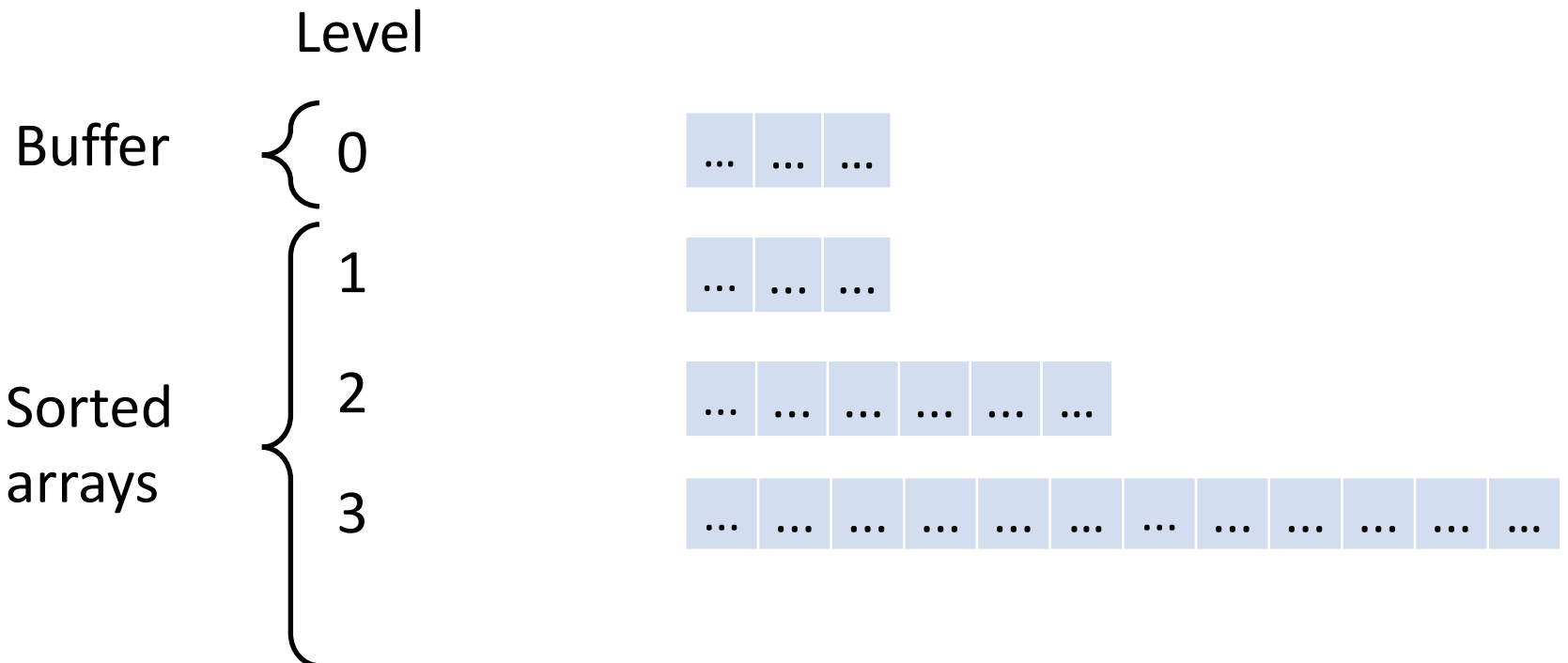


Basic LSM-tree – Lookup cost

Lookup method? Search youngest to oldest. $O\left(\log_2\left(\frac{N}{B}\right)\right)$



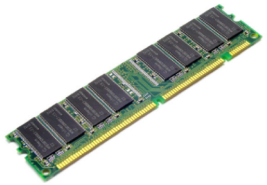


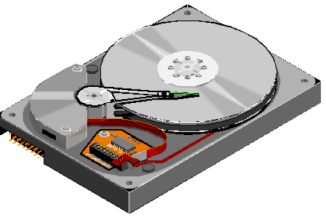


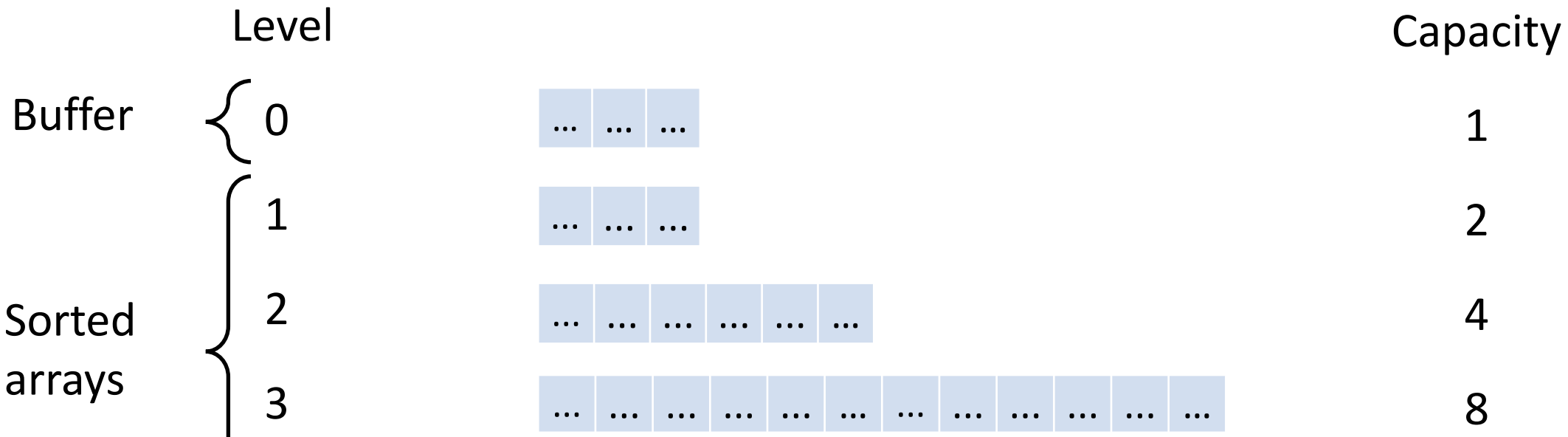
Basic LSM-tree – Lookup cost

Lookup method? Search youngest to oldest. $O\left(\log_2\left(\frac{N}{B}\right)\right)$

How?



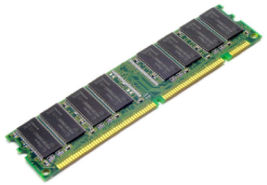


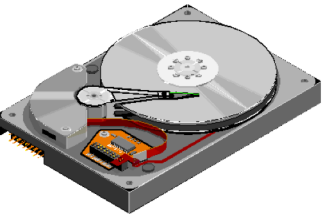


Basic LSM-tree – Lookup cost

Lookup method? Search youngest to oldest. $O\left(\log_2\left(\frac{N}{B}\right)\right)$

How? Binary search. $O\left(\log_2\left(\frac{N}{B}\right)\right)$



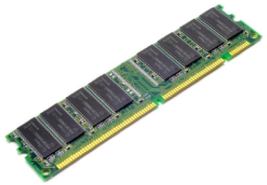


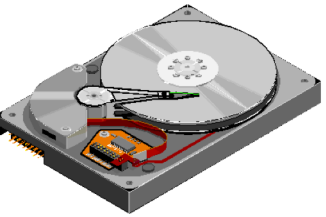
Basic LSM-tree – Lookup cost

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?



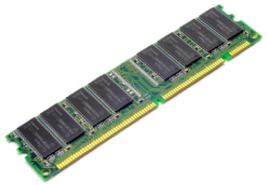


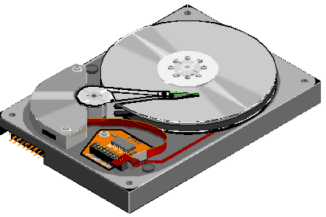
Basic LSM-tree – Lookup cost

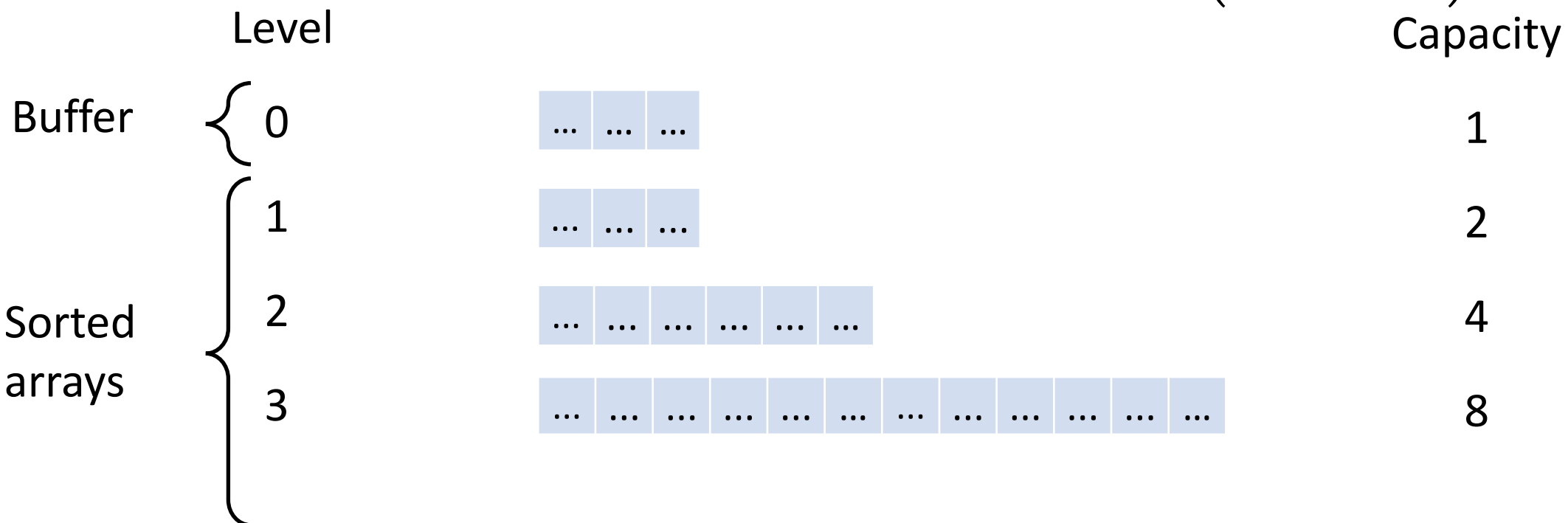
Lookup method? Search youngest to oldest. $O\left(\log_2\left(\frac{N}{B}\right)\right)$

How? Binary search. $O\left(\log_2\left(\frac{N}{B}\right)\right)$

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





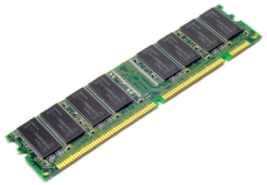


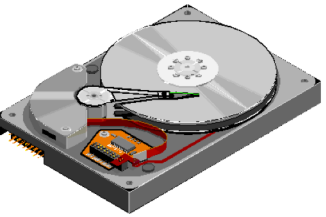
Basic LSM-tree – Insertion cost

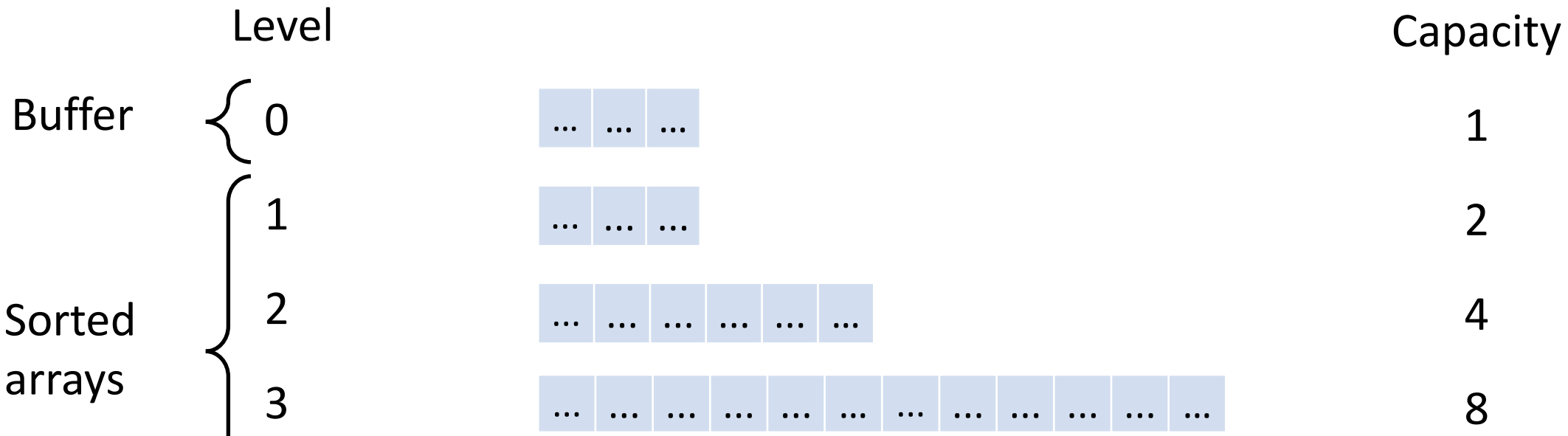
How many times is each entry copied? $O\left(\log_2\left(\frac{N}{B}\right)\right)$

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	$O(\log_B(N/B))$	$O(\log_B(N/B))$
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

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	$O(\log_B(N/B))$	$O(\log_B(N/B))$
Basic LSM-tree	$O(\log_2(N/B)^2)$	$O(1/B \cdot \log_2(N/B))$
Leveled LSM-tree		
Tiered LSM-tree		

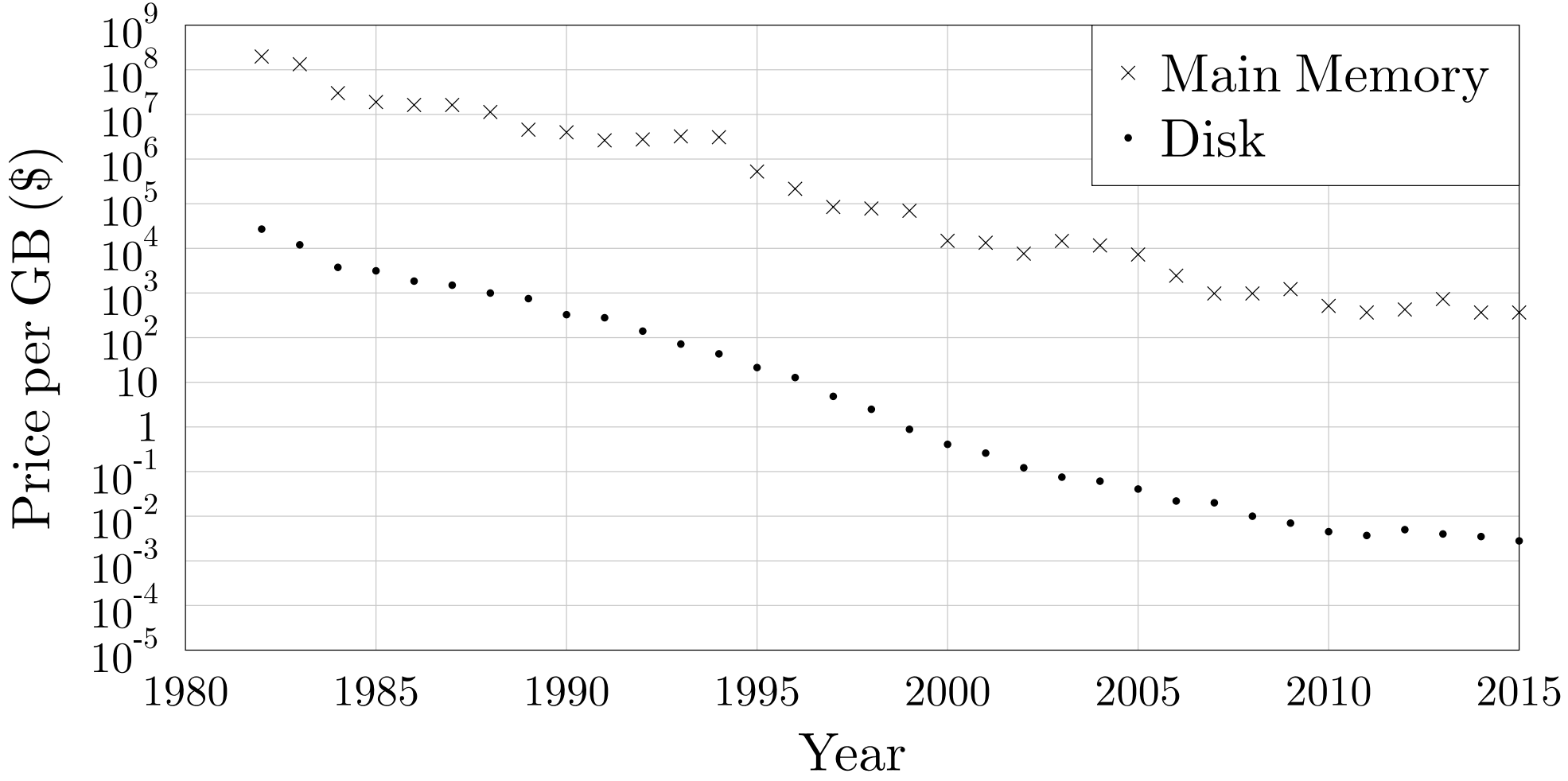
Results Catalogue

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

Can we improve lookup cost?

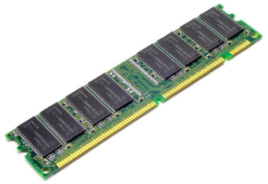
	Lookup cost	Insertion cost
Sorted array	$O(\log_2(N/B))$	$O(N/B)$
Log	$O(N/B)$	$O(1/B)$
B-tree	$O(\log_B(N/B))$	$O(\log_B(N/B))$
Basic LSM-tree	$O(\log_2(N/B)^2)$	$O(1/B \cdot \log_2(N/B))$
Leveled LSM-tree		
Tiered LSM-tree		

Declining Main Memory Cost

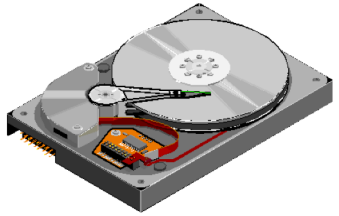


Declining Main Memory Cost

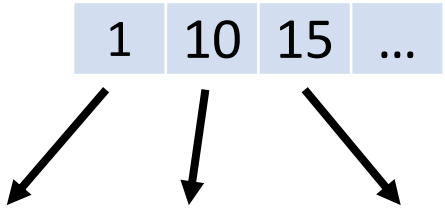
Store a fence pointer for every block in main memory



Fence pointers



array



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

Results Catalogue – with fence pointers

	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	$O(\log_2(N/B))$	$O(N/B)$
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)
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)
Log	O(N/B)	O(1/B)
B-tree	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)
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)
Basic LSM-tree	$O(\log_2(N/B))$	$O(1/B \cdot \log_2(N/B))$
Leveled LSM-tree		
Tiered LSM-tree		

Results Catalogue – with fence pointers

Quick sanity check: suppose $N = 2^{42}$ and $B = 2^{10}$

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

Results Catalogue – with fence pointers

Quick sanity check: suppose $N = 2^{42}$ and $B = 2^{10}$

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

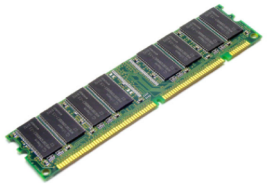
Leveled LSM-tree

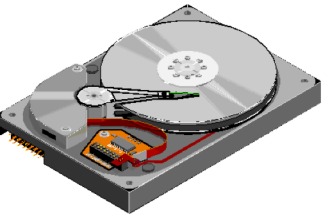
 Lookup cost

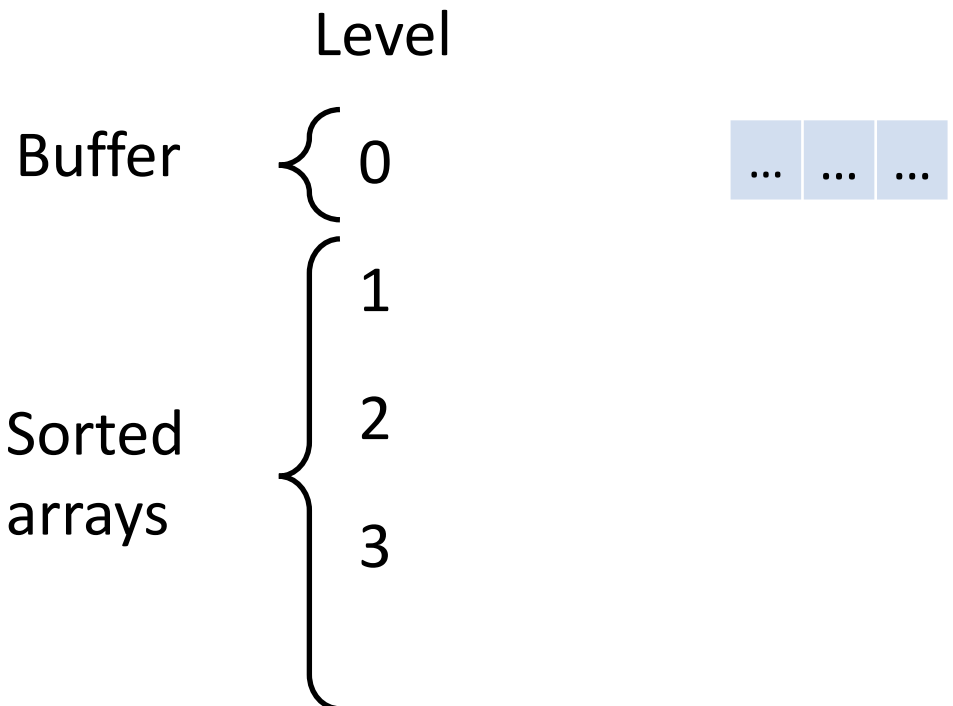
 Update cost

Leveled LSM-tree

Lookup cost depends on number of levels

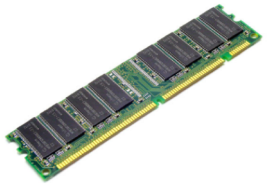


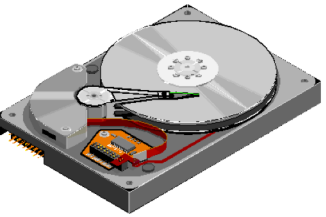




Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?

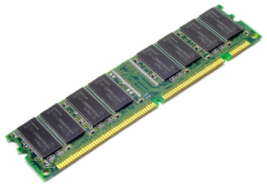


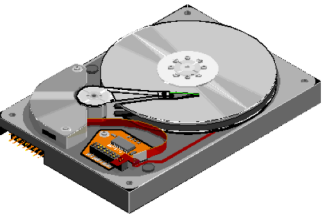


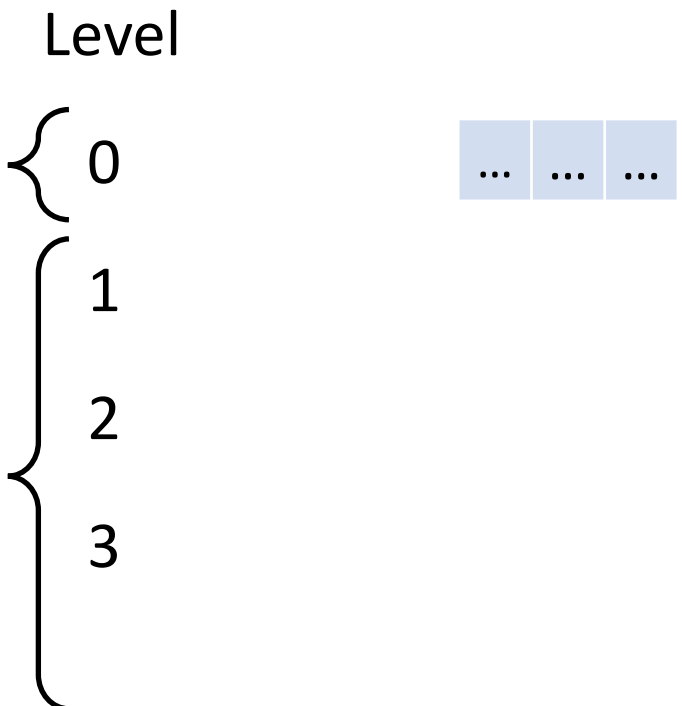
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?

Increase size ratio T



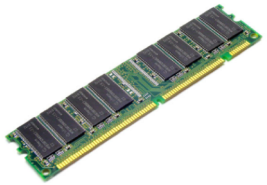


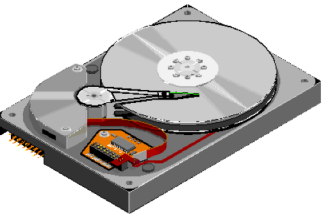


Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?

Increase size ratio T





Level

Capacity

Buffer { 0 … … … T^0

Sorted arrays {

1 T^1

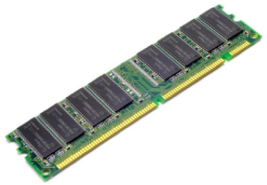
2 T^2

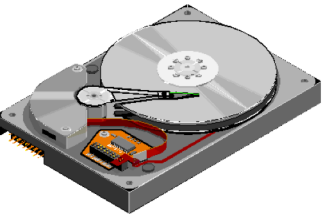
3 T^3

Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio T



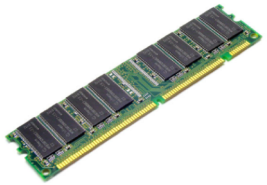


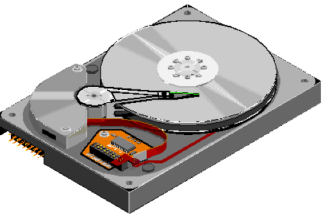
	Level	Capacity
Buffer	0	1
Sorted arrays	1	4
	2	16
	3	64

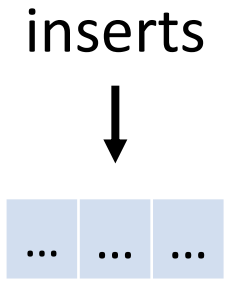
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio T



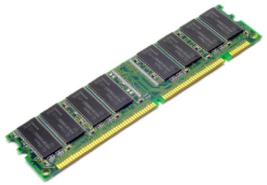


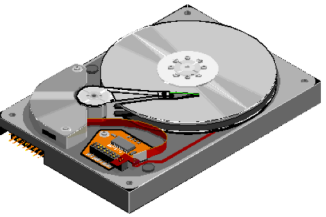


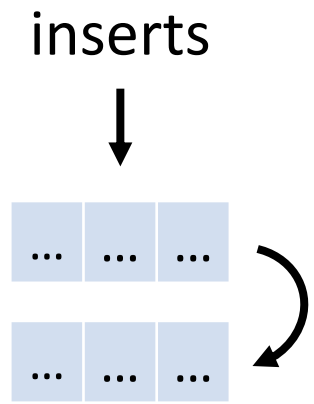
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio T





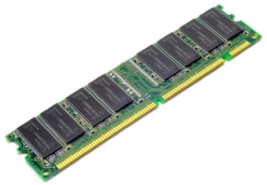


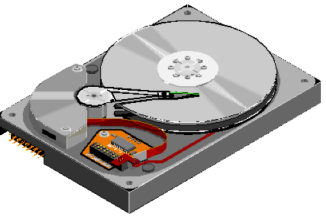
Level		Capacity
Buffer	0	1
Sorted arrays	1	4
	2	16
	3	64

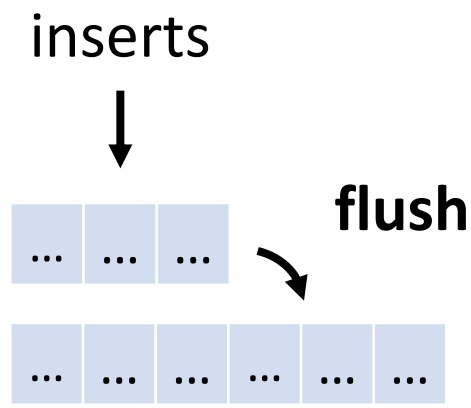
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio T





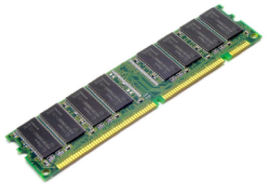


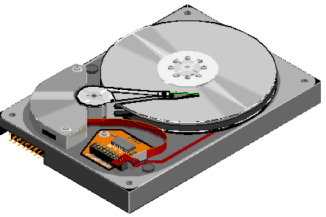
Level		Capacity
Buffer	0	1
Sorted arrays	1	4
	2	16
	3	64

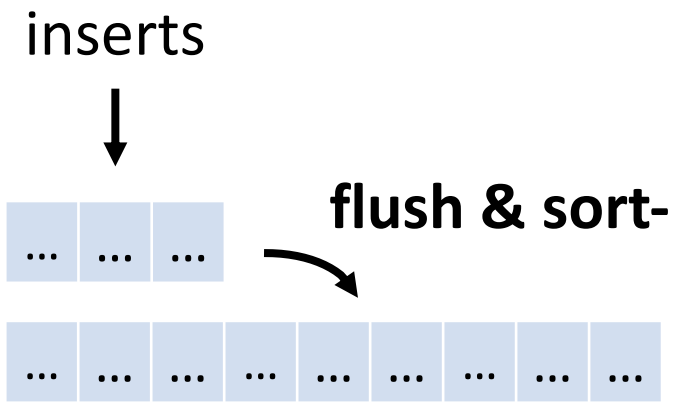
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio T



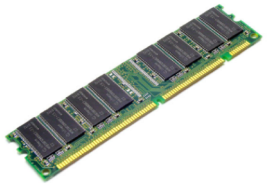


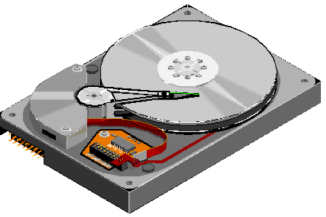


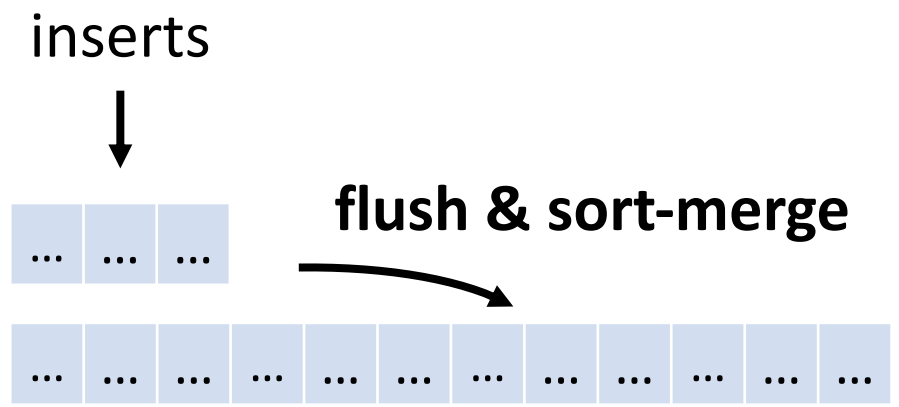
Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio T



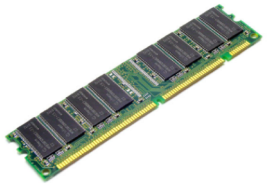


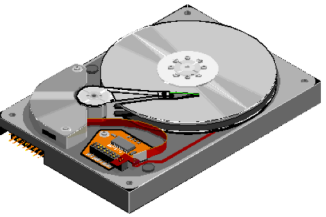


Leveled LSM-tree

Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

Increase size ratio T

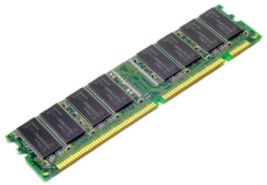


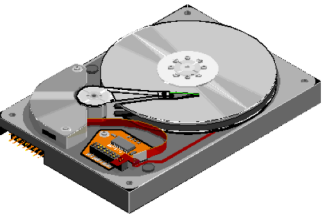


Leveled LSM-tree

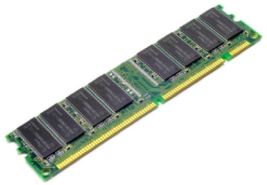
Lookup cost depends on number of levels
How to reduce it?
E.g. size ratio of 4

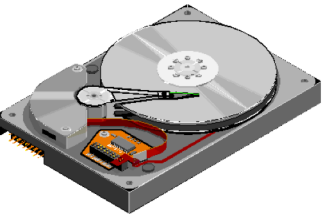
Increase size ratio T





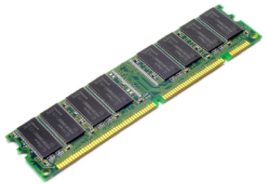
Leveled LSM-tree

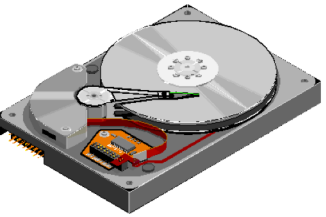


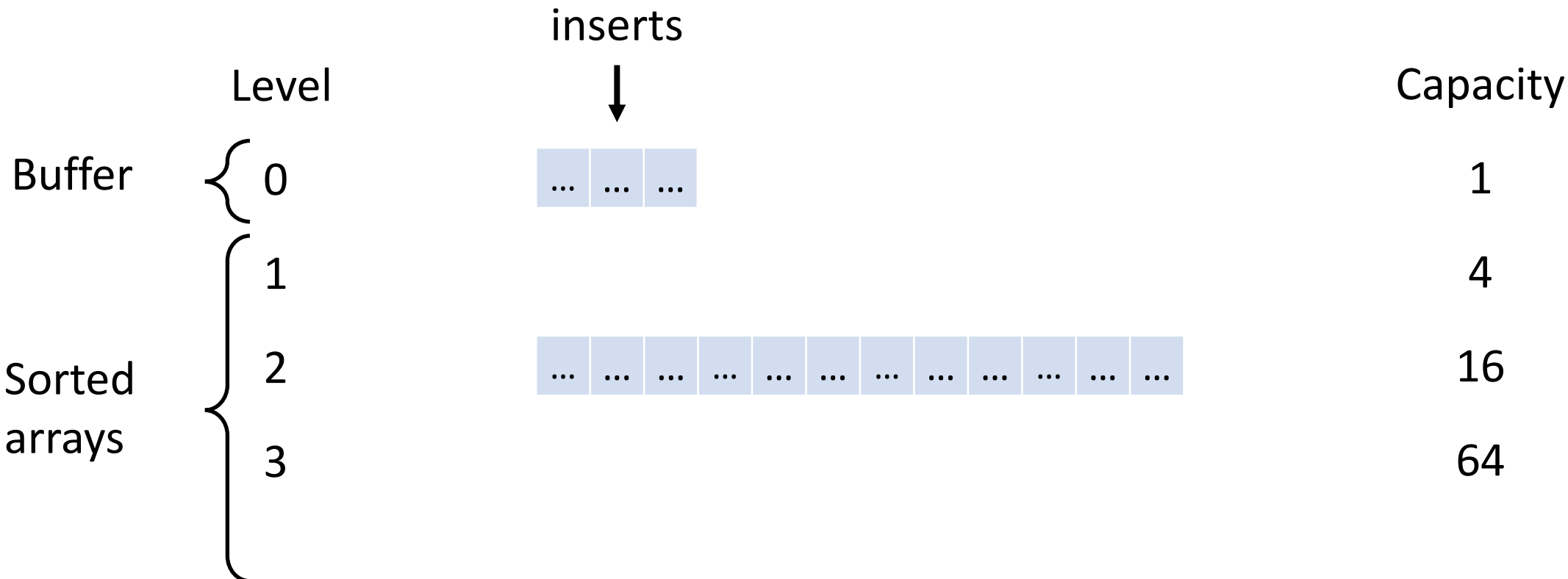


Leveled LSM-tree

Lookup cost?



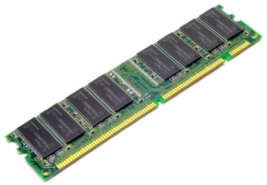


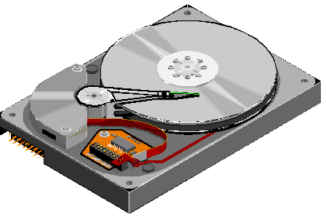


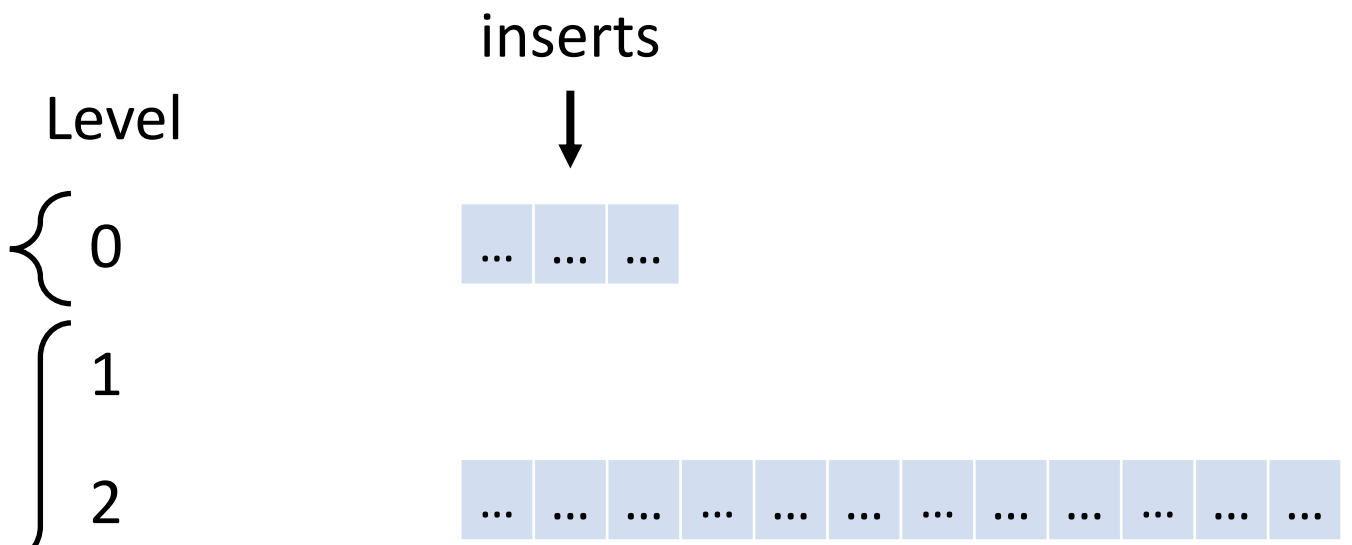
Leveled LSM-tree

Lookup cost?

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





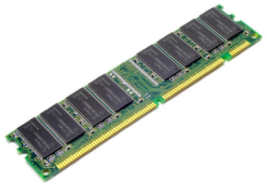


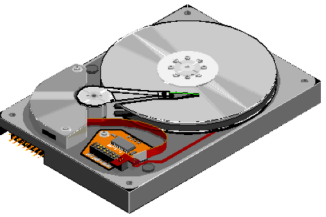
Leveled LSM-tree

Lookup cost?

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

Insertion cost?





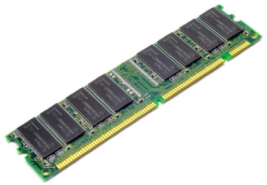
Leveled LSM-tree

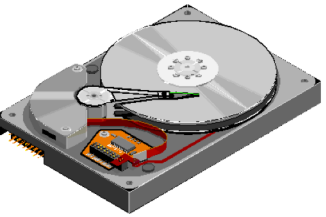
Lookup cost?

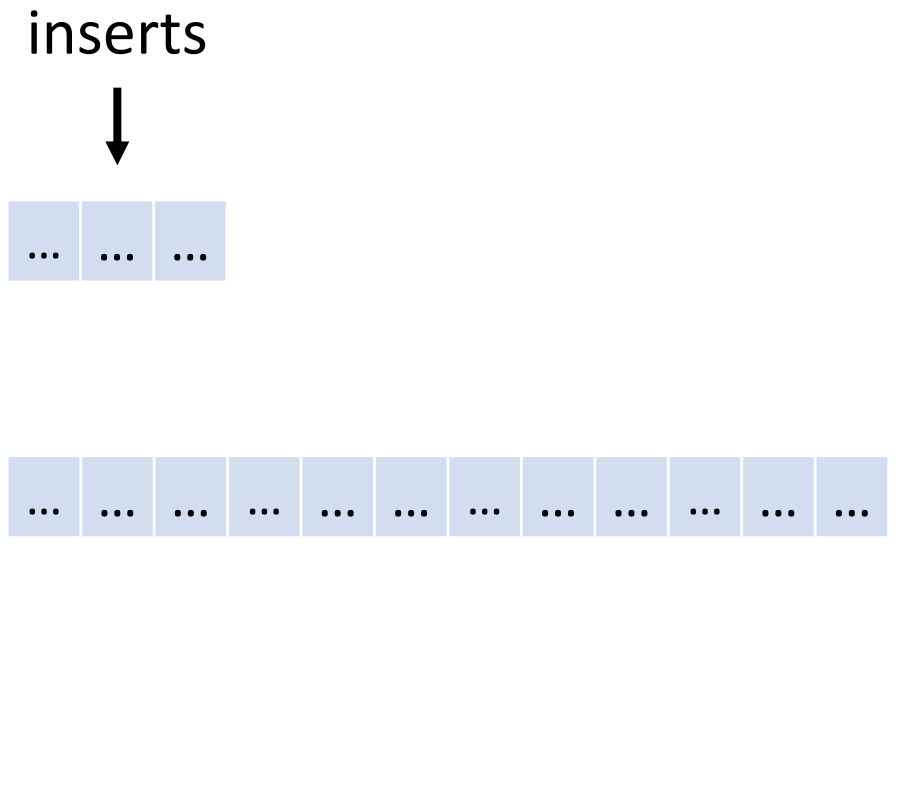
$$O\left(\log_T\left(\frac{N}{B}\right)\right)$$

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?

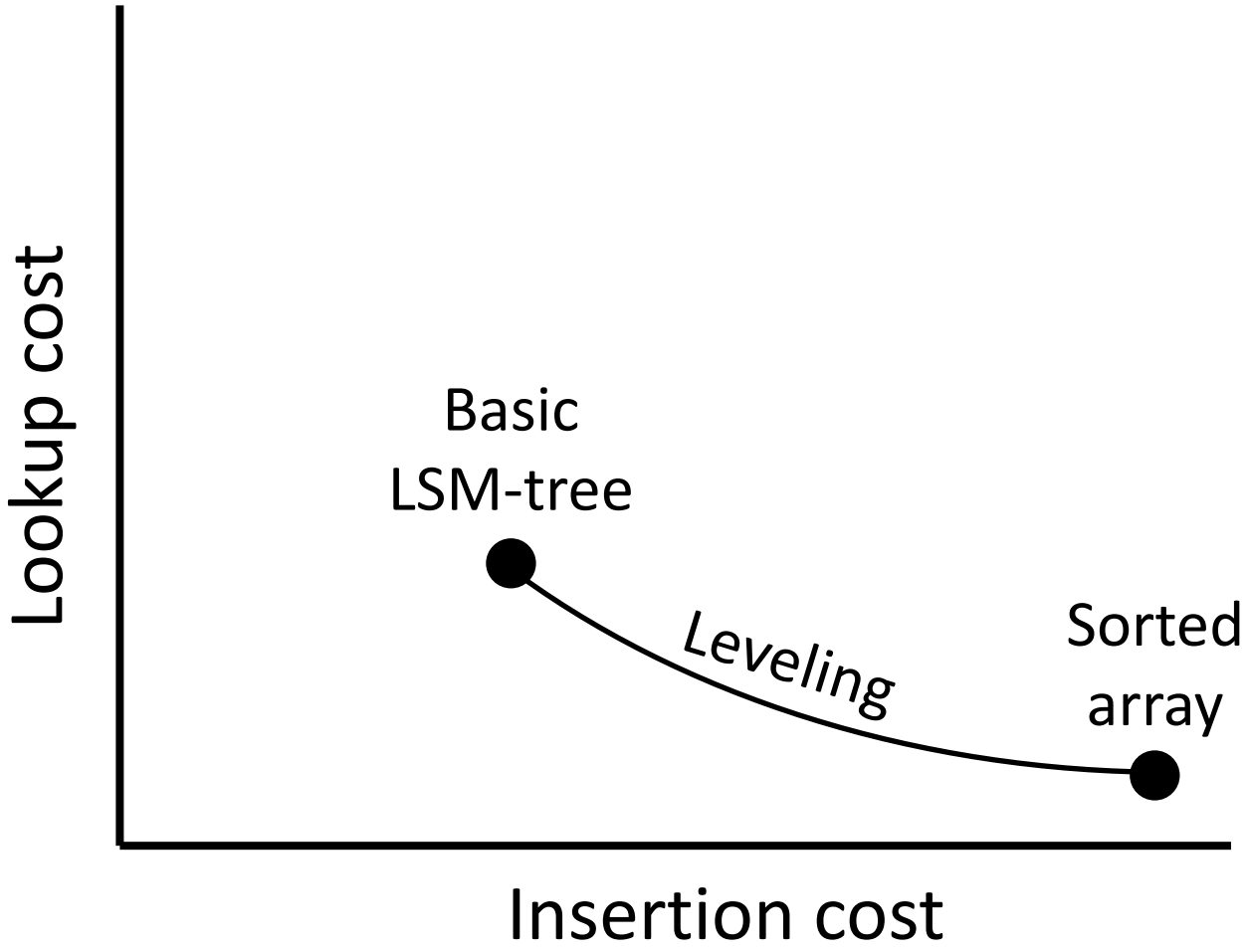
Lookup cost becomes:

O(1)

Insert cost becomes:

$O(N/B^2)$

The LSM-tree becomes a sorted array!


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/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	$O(\log_T(N/B))$	$O(T/B \cdot \log_T(N/B))$
Tiered LSM-tree		

Tiered LSM-tree

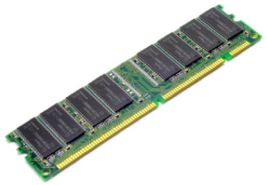
Tiered LSM-tree

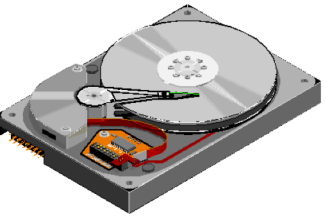

Lookup cost


Insertion cost

Tiered LSM-tree

Reduce the number of levels by increasing the size ratio.





Level

Capacity

Buffer

0 … … … T^0

Sorted arrays

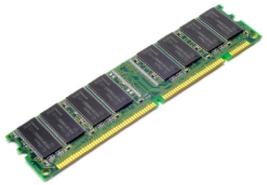
1 T^1

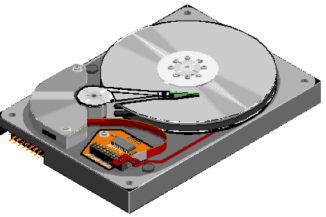
2 T^2

3 T^3

Tiered LSM-tree

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





Level

Capacity

Buffer

0 … … … T^0

Sorted arrays

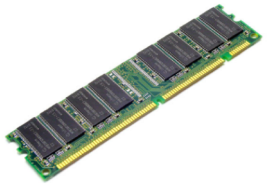
1 T^1

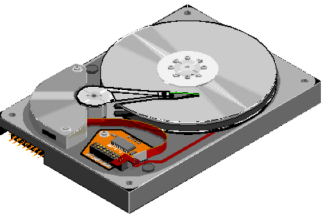
2 T^2

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

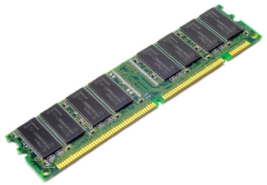


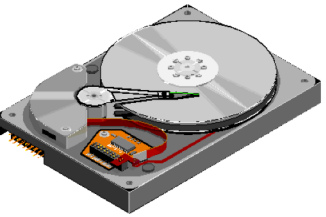


Level		Capacity
Buffer	0	1
Sorted arrays	1	4
	2	16
	3	64

Tiered LSM-tree

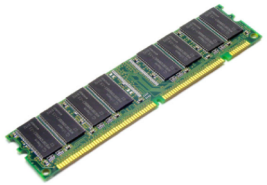
Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4

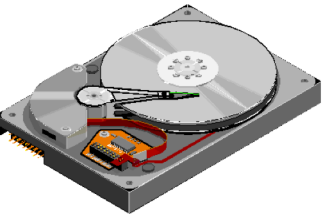




Tiered LSM-tree

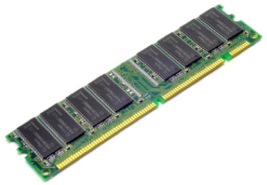
Reduce the number of levels by increasing the size ratio.
Do not merge within a level.
E.g. size ratio of 4

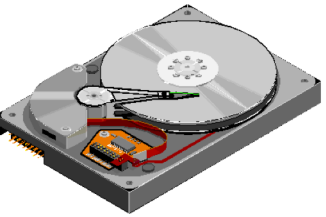


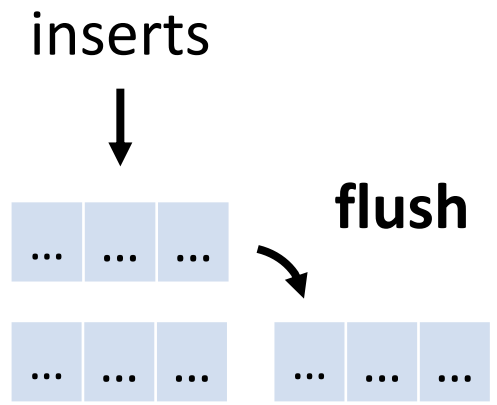


Tiered LSM-tree

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

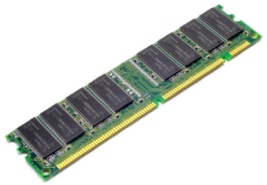


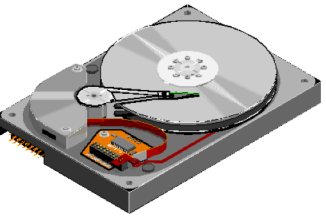


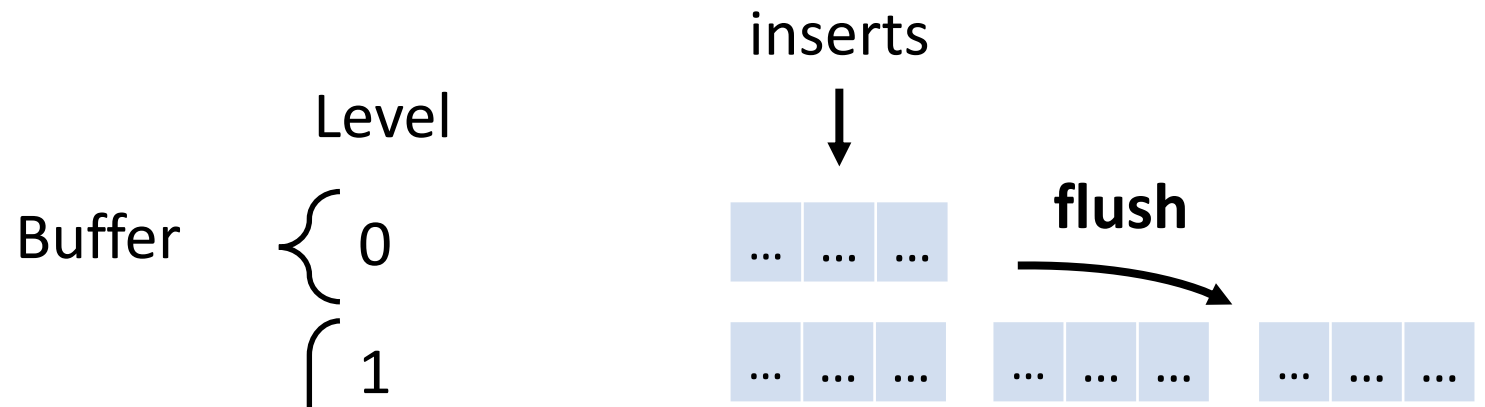


Tiered LSM-tree

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

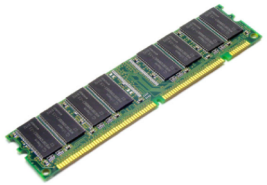


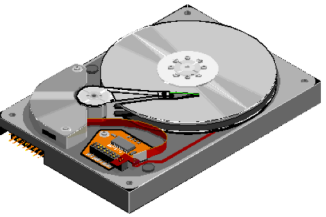


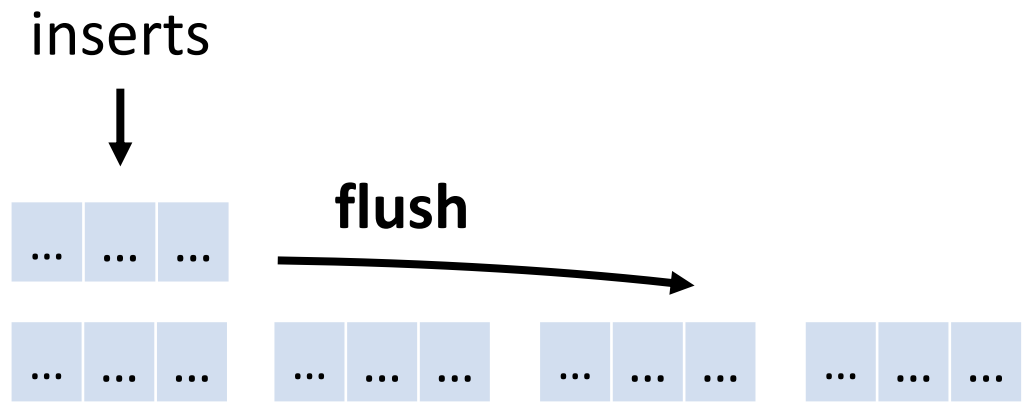


Tiered LSM-tree

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

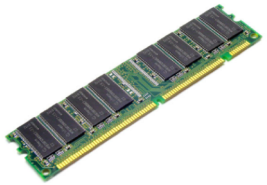


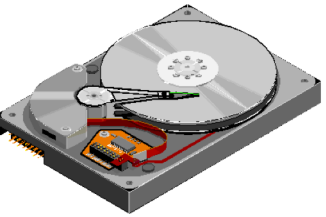


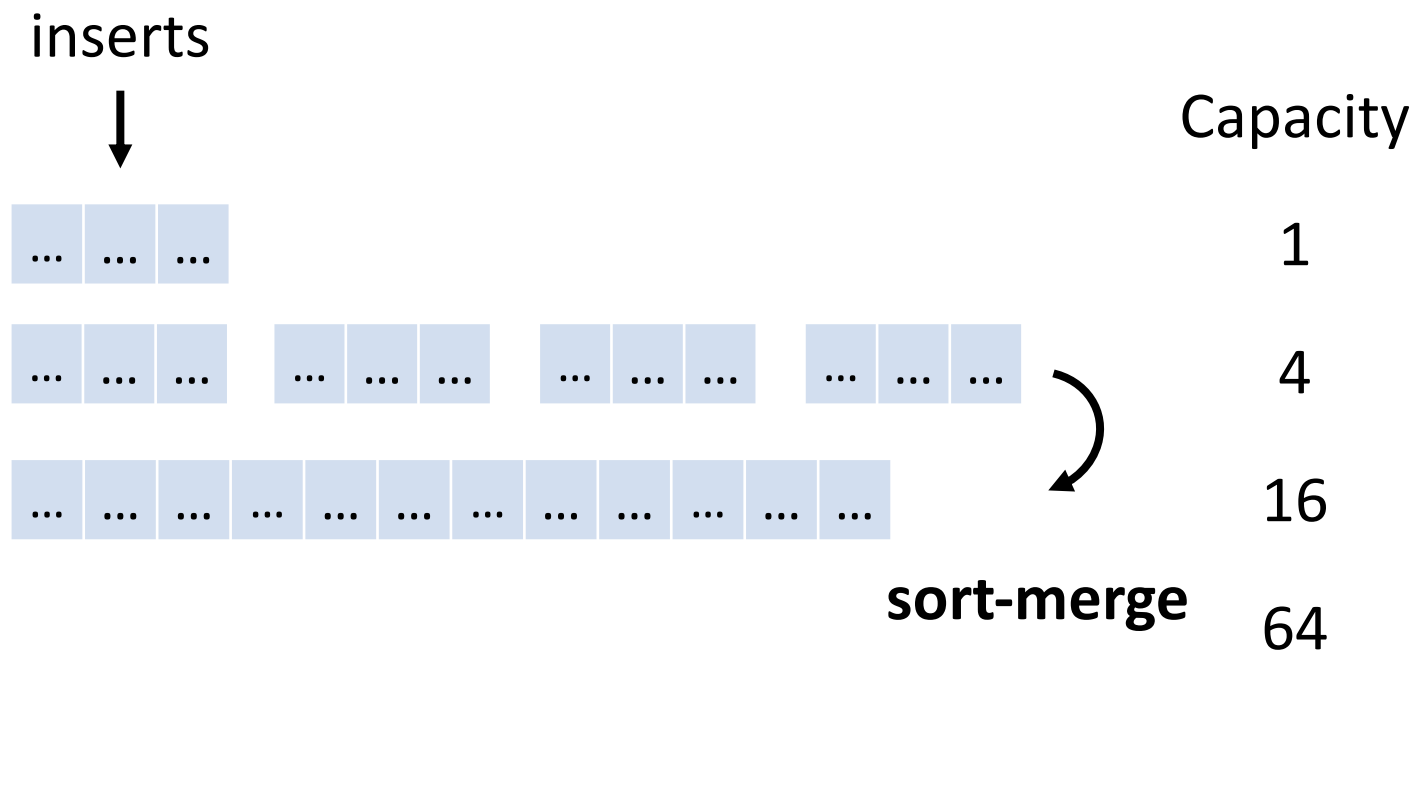


Tiered LSM-tree

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

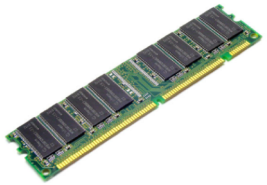


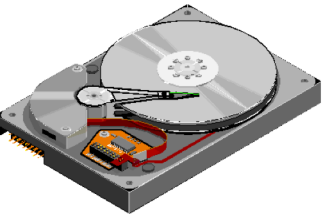


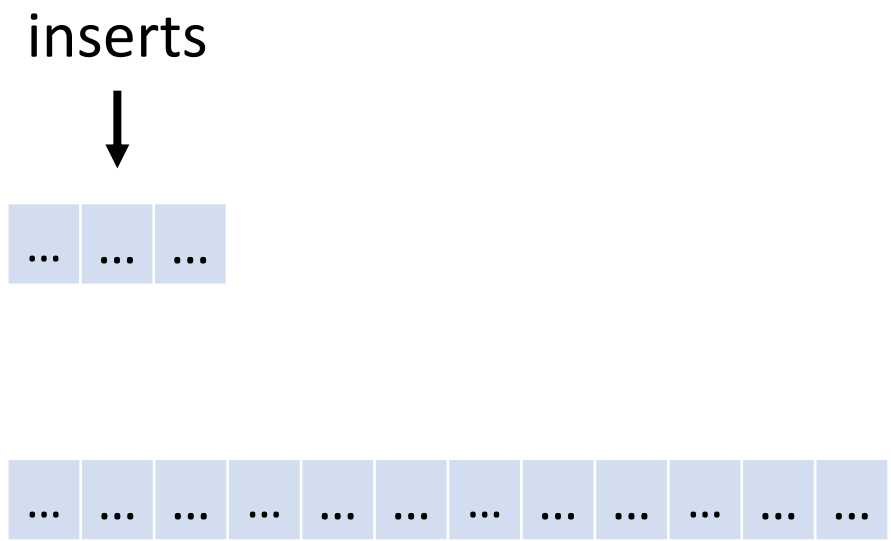


Tiered LSM-tree

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

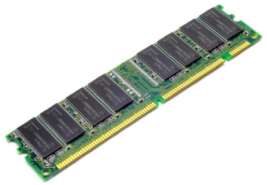


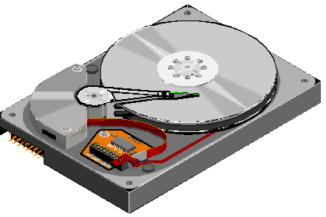


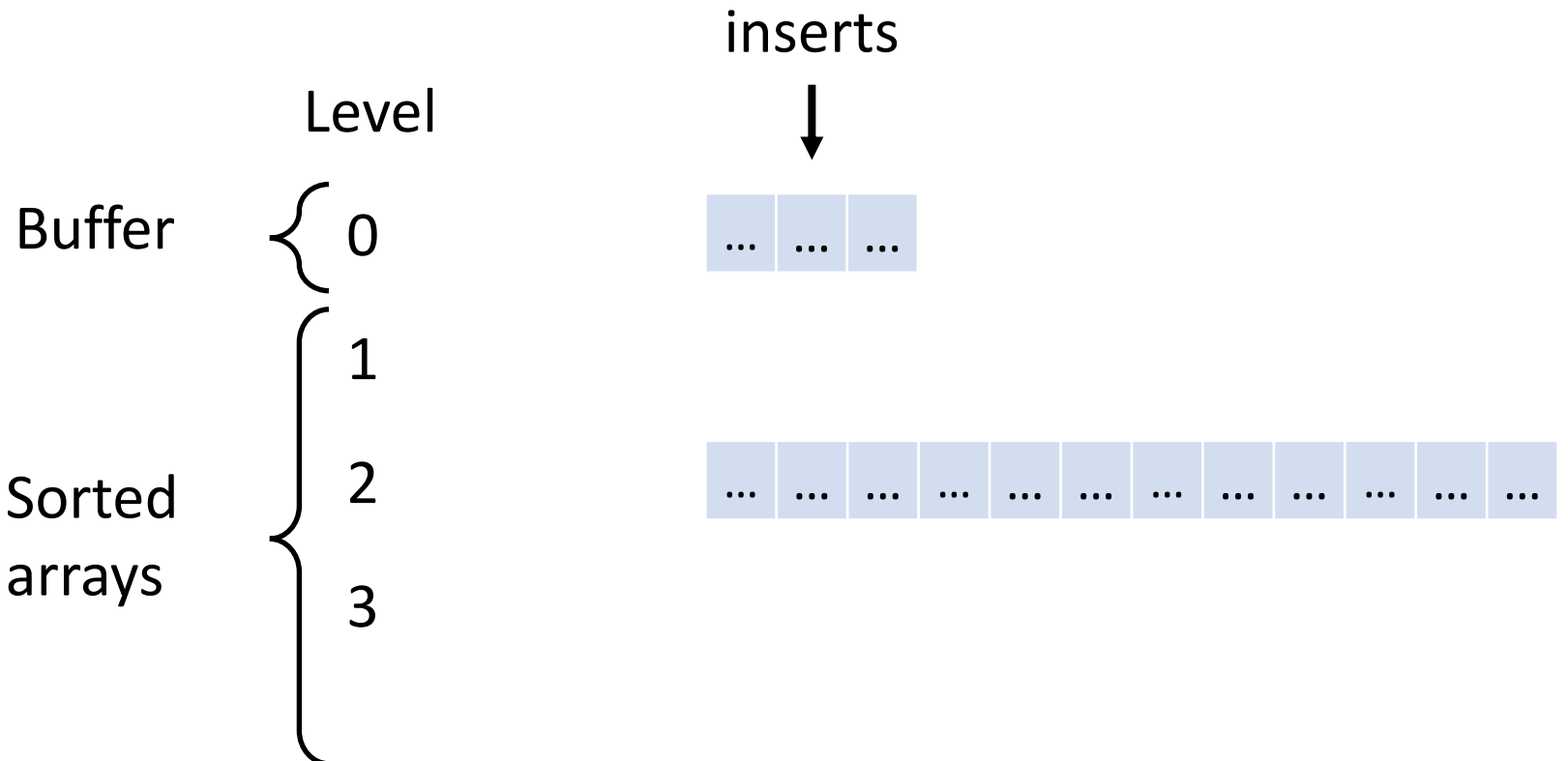


Tiered LSM-tree

Lookup cost?



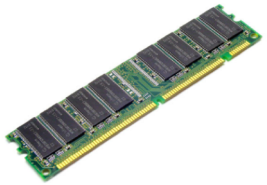


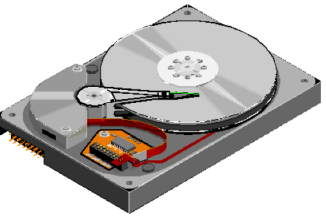


Tiered LSM-tree

Lookup cost?

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



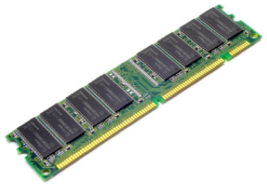


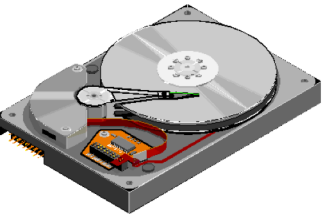
Tiered LSM-tree

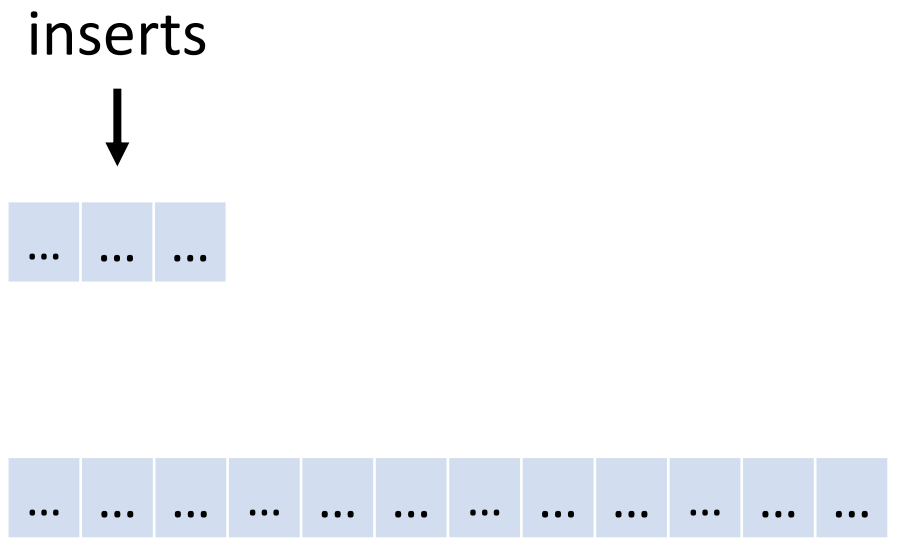
Lookup cost?

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

Insertion cost?







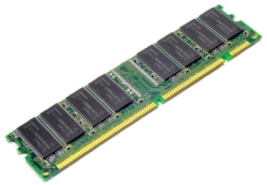
Tiered LSM-tree

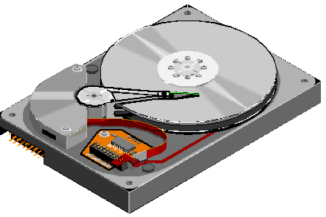
Lookup cost?

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

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?

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

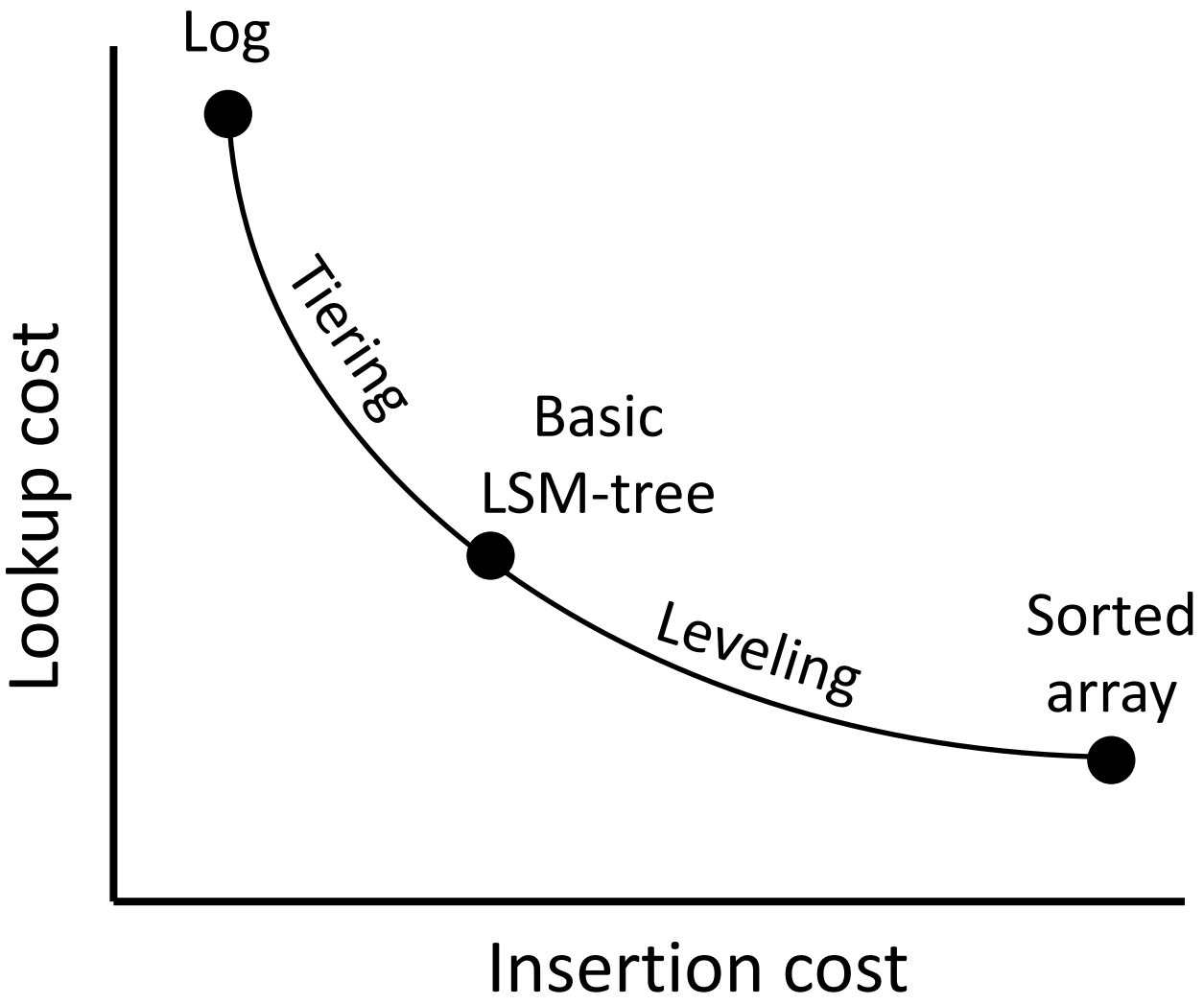
What happens as we increase the size ratio T?

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!

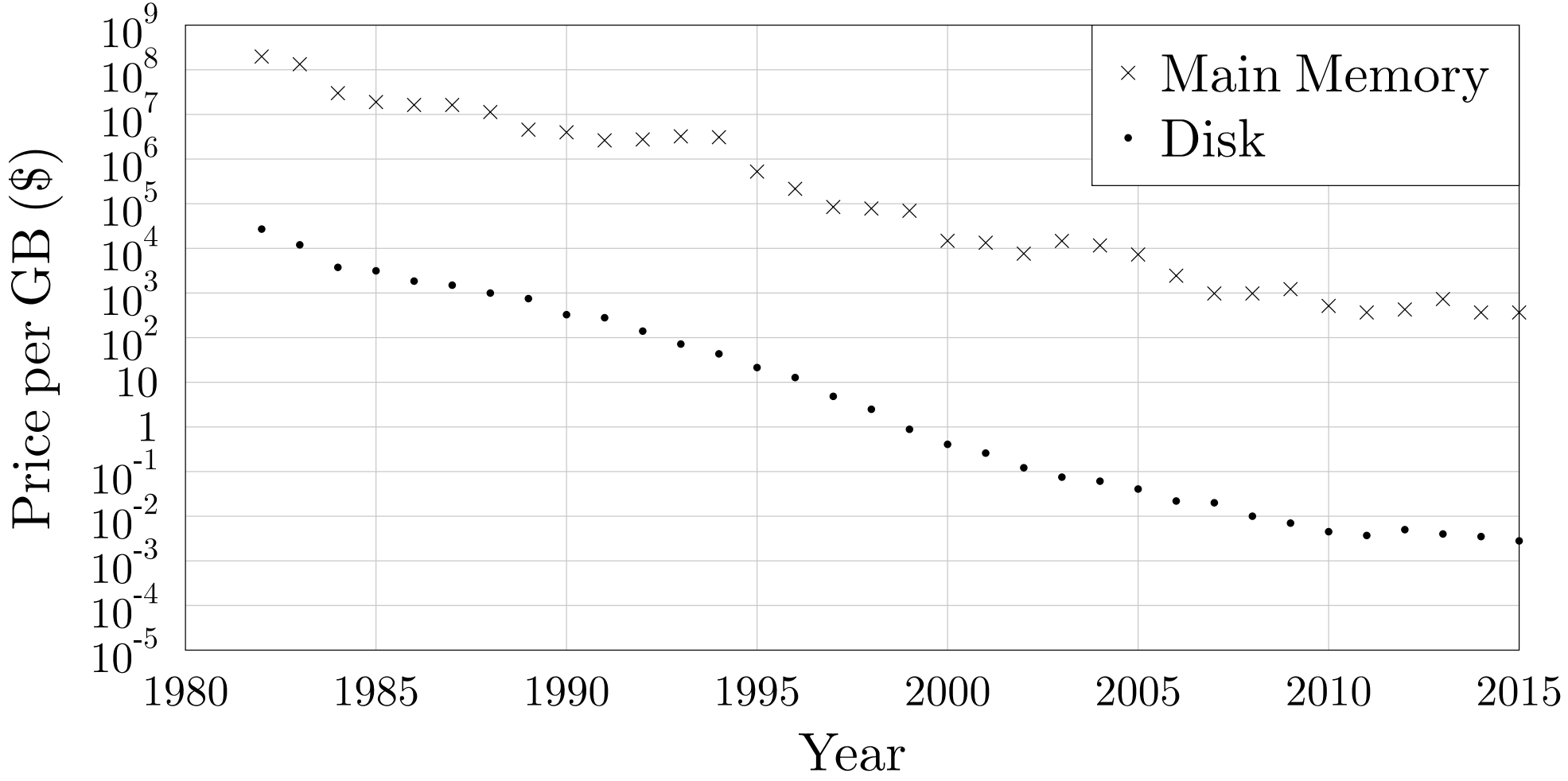

Log
Tiering
Basic
LSM-tree
Leveling
Sorted
array
Lookup cost
Insertion cost

Results Catalogue – with fence pointers

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

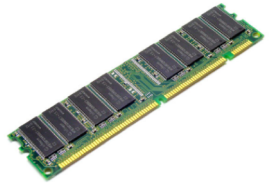
Bloom filters

Declining Main Memory Cost



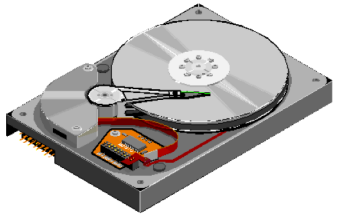
Bloom Filters

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.



filters

Bloom filter



array

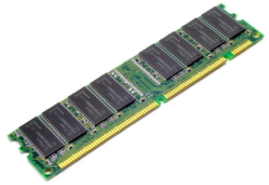
Bloom Filters

Answers set-membership queries

Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

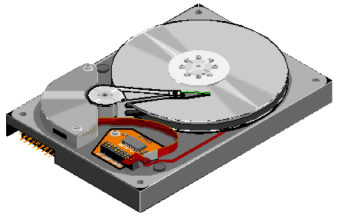
Subtlety: may return false positives.



filters

Lookup for X

Bloom filter



array

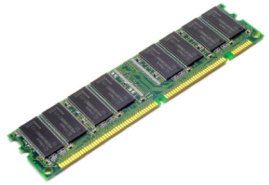
Bloom Filters

Answers set-membership queries

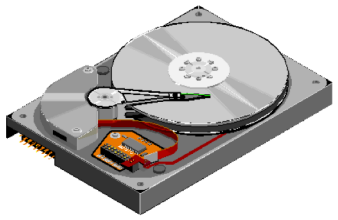
Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

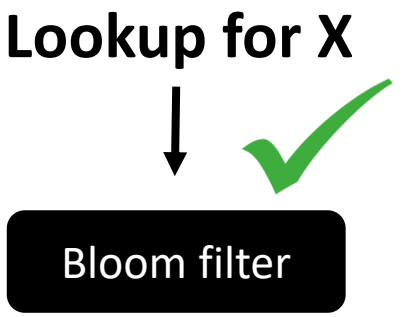
Subtlety: may return false positives.



filters



array



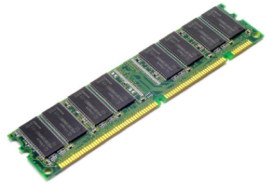
Bloom Filters

Answers set-membership queries

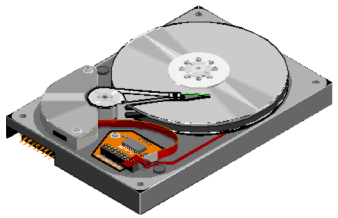
Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

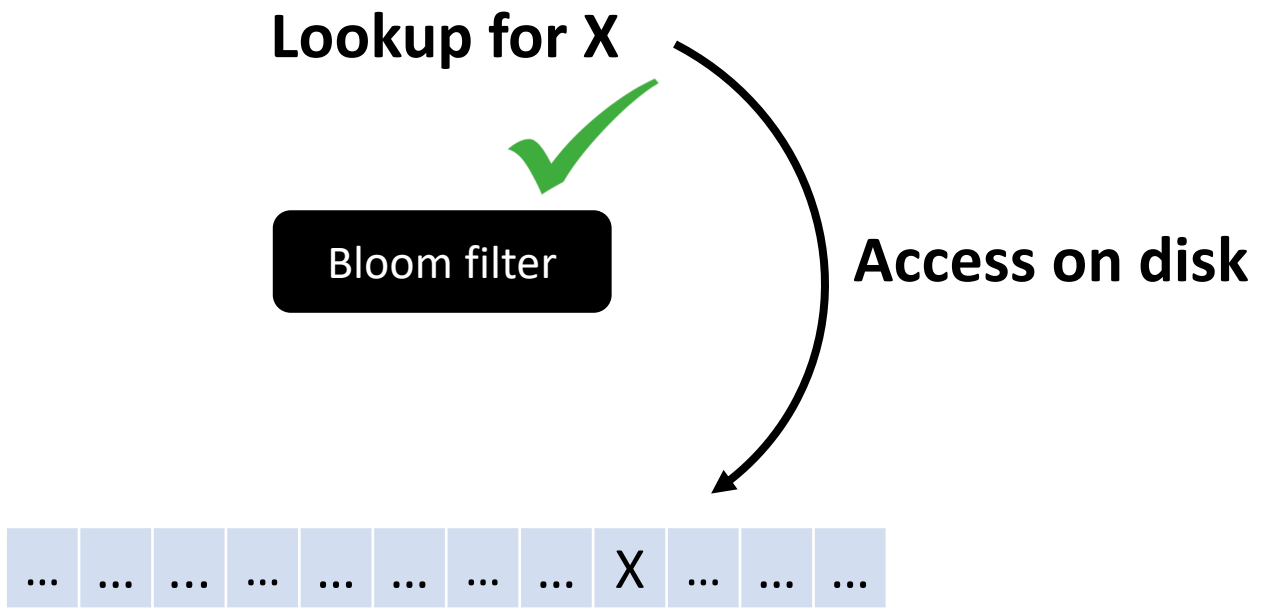
Subtlety: may return false positives.



filters



array



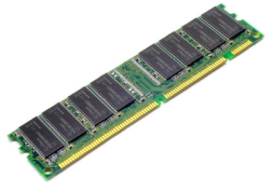
Bloom Filters

Answers set-membership queries

Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

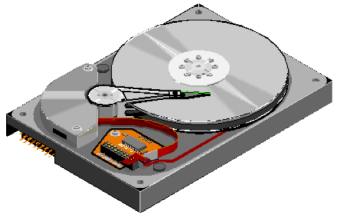
Subtlety: may return false positives.



filters

Lookup for Y

Bloom filter



array

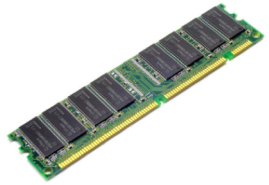
Bloom Filters

Answers set-membership queries

Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

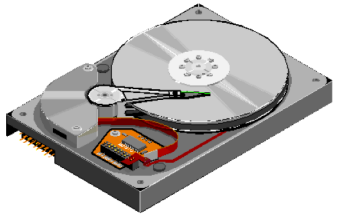
Subtlety: may return false positives.



filters

Lookup for Y

Bloom filter



array

Bloom Filters

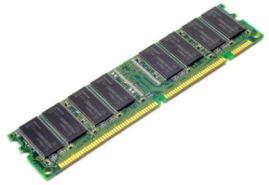
Answers set-membership queries

Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

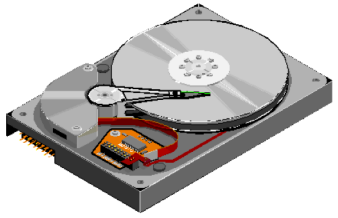
Subtlety: may return false positives.

Lookup for Y



filters

Bloom filter



array

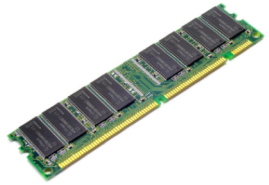
Bloom Filters

Answers set-membership queries

Smaller than array, and stored in main memory

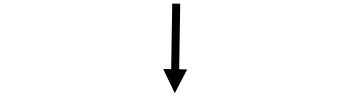
Purpose: avoid accessing disk if entry is not in array

Subtlety: may return false positives.

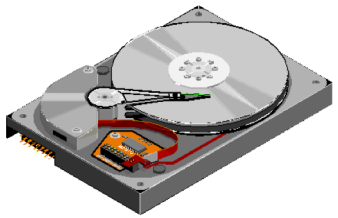


filters

Lookup for Z



Bloom filter



array

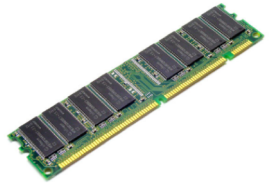
Bloom Filters

Answers set-membership queries

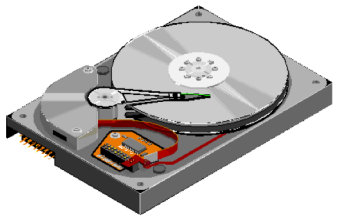
Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

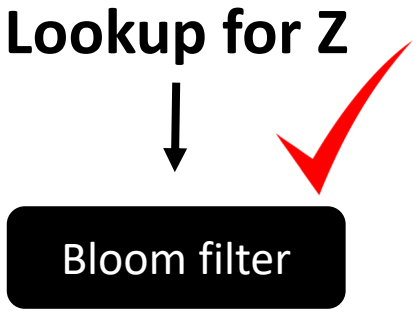
Subtlety: may return false positives.



filters



array



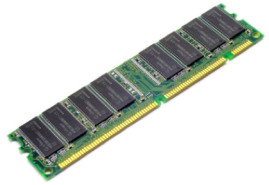
Bloom Filters

Answers set-membership queries

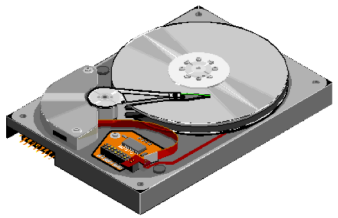
Smaller than array, and stored in main memory

Purpose: avoid accessing disk if entry is not in array

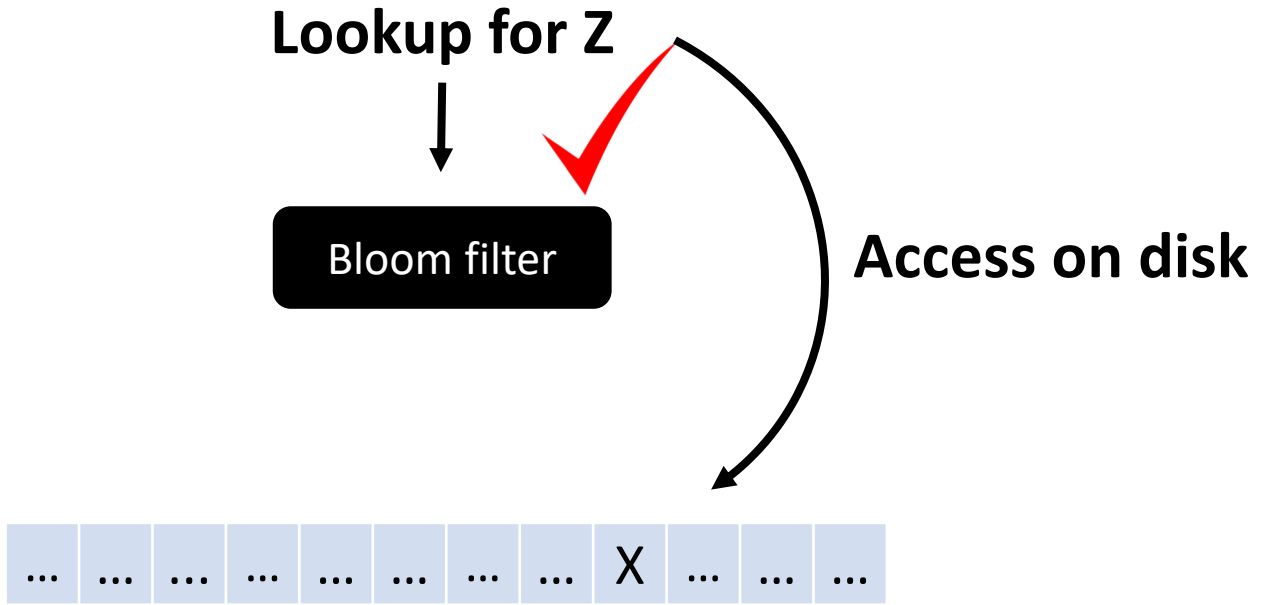
Subtlety: may return false positives.



filters

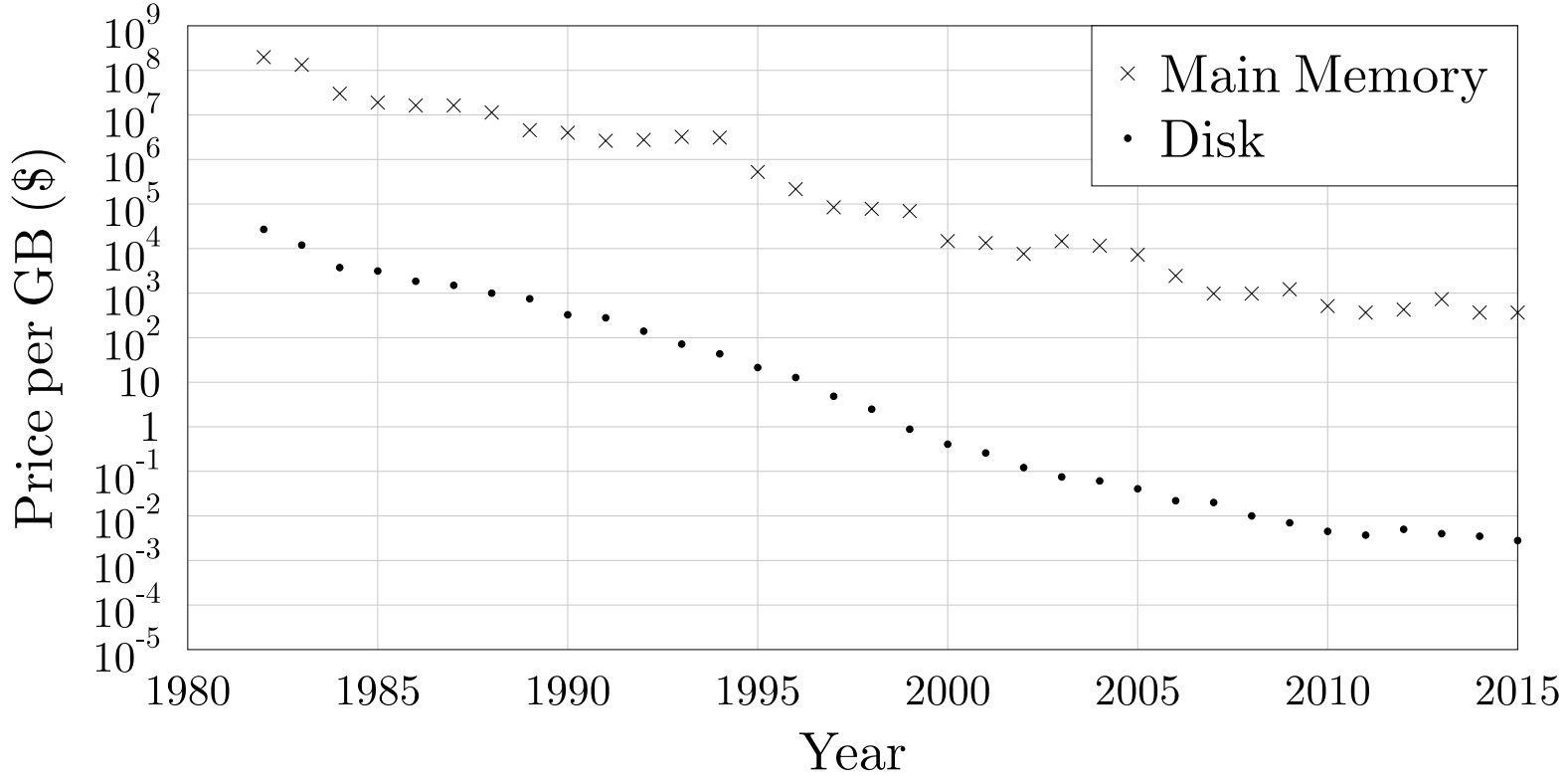


array



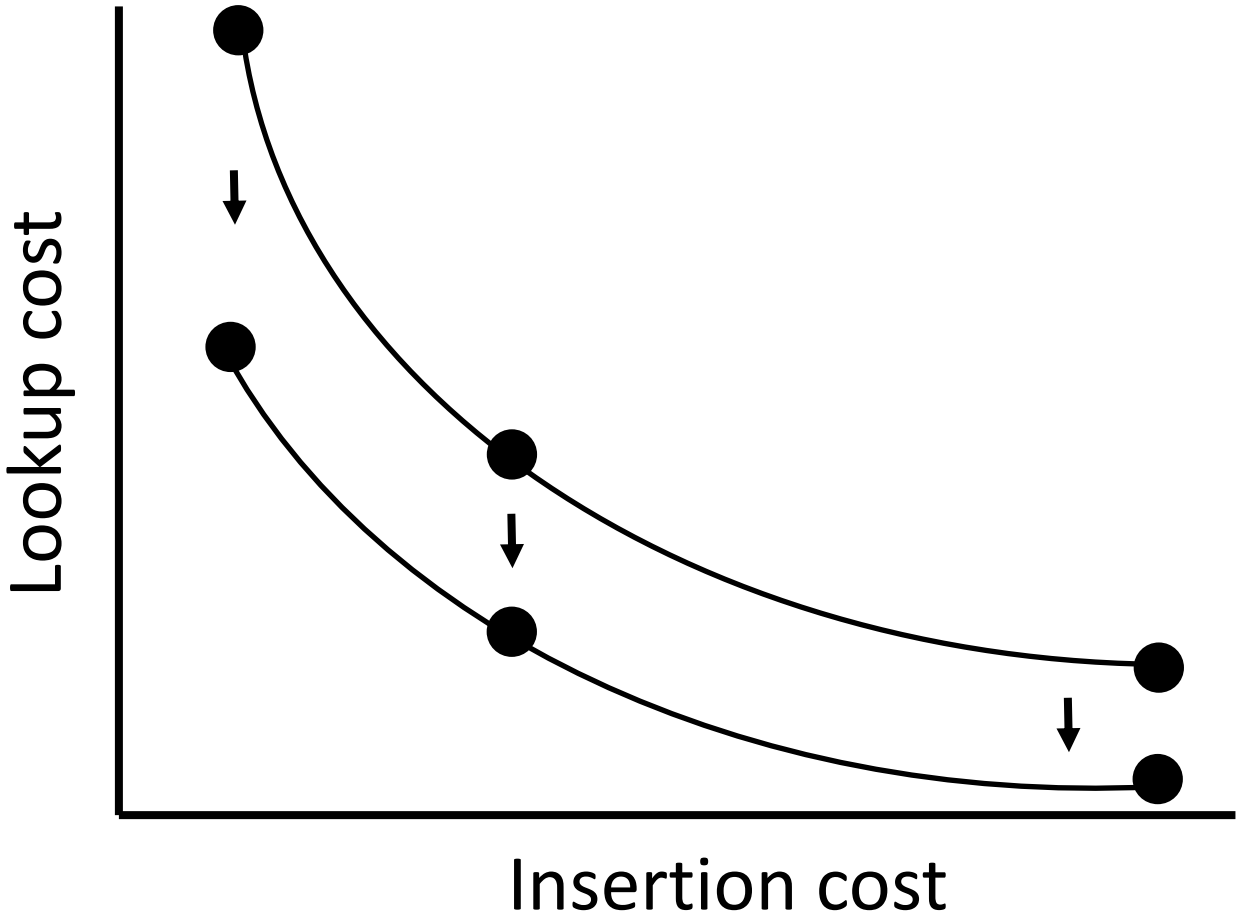
Bloom Filters

The more main memory, the fewer false positives ⟹ 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!