



github link

! Try Hadoop / RocksDB
(use by FB/meh)

<https://github.com/search?q=Cz4123&type=repositories>

Course Schedule

COURSE SCHEDULE

Week 1 – Week 13: Course lectures
Venue: LT27
Time: 15:30pm – 17:20pm on Tuesday

Week 3 – Week 13: Tutorials
Venue: LT27
Time: 11:30am – 12:20pm on Monday

Quiz: Week 10 tutorial

Assessment

EVALUATION (TENTATIVE)

1 Quiz:
25%

1 Group Project:
25%

Final:
50%

deadline week 14!!

Page Limits of Project Report

Posted on: Wednesday, February 28, 2024 5:59:58 PM SGT

Dear Students,

This is a gentle reminder that your project report is allowed at most 5 pages (excluding cover pages and contribution forms). Please check Section 4 of the project description. The page limit is there to avoid unnecessarily lengthy reports that significantly increase your workloads.

Meanwhile, I understand some of you really would like to have a slight extension of page limits to accommodate big figures such as screenshots. To give you some flexibility, if you really want to place big figures such as screenshots (or design figures) which you cannot squeeze them into the 5 main pages after your best trial, you may **optionally** add an Appendix section to contain those figures. The section is fully optional, and it should only contain figures and corresponding captions. The Appendix section is allowed at most **two pages**. Note that the assessment would still be mainly based on the 5-page main content, so for those who already target 5-page report, it is perfectly fine.

A few tips when you are concerning page limits:

1. Can set/draw the screenshots/figures relatively small.
2. Avoid unnecessary screenshots. When needed, only screenshot the "informative part".

Hope that gives a bit flexibility to you.

- mapReduce → Solution for huge amount of data
- External Sorting → Solution for sorting huge amount of data without machine
- NoSQL → Key Value Stores that is more Scalable
- Column Store → if Search item has hundreds of properties

BIG DATA MANAGEMENT - COURSE OVERVIEW

We will discuss interesting big data techniques!

Most of the techniques are cutting-edge techniques in big data!

No text books – Slides contain everything



Big Data 5V's	Memory Hierarchy	Column Store	Distributed MapReduce Systems	NoSQL Key-Value store
---------------	------------------	--------------	-------------------------------	-----------------------



>\$3 trillion/year

>500,000 big data jobs

A report from McKinsey Global Institute estimates that Big Data could generate an additional \$3 trillion in value every year in just seven industries. The report also estimated that over half of this value could go to consumers. Examples such as learning new items, easier price comparisons, and better matching between educational institutions and students.

Data analysis has been called "the sexiest job of the 21st century." The United States already has an estimated shortage of Big Data workers. Recently, it was estimated that there is a shortage of between 140,000 and 190,000 workers with advanced degrees in statistics, computer engineering and other fields. People with these skills represent the shortage of 2.5 million managers and analysts who hold traditional jobs but are capable of integrating Big Data into their decision making.

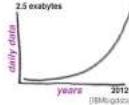
**Data is so big that we need
High-Performance Computation over big data**

efficiently store and query big data

DATA SIZE

- 1 Bit = 1/8 of a letter
- 1 Byte = a letter (8 bits)
- 1 Megabyte = a book (1024 kilobytes)
- 1 Gigabyte = ~1000 books (1024 megabytes)
- 1 Terabyte = ~1million books (1024 gigabytes)
- 1 Petabyte = ~1billion books (1024 terabytes)
- 1 Exabyte = ~ 10^{12} books (1024 Petabytes)

Nowadays, more than 2.5 exabytes of data are generated in every single day!



Every two days we create as much data as much we did from the dawn of humanity to 2003
[Eric Schmidt, Google]

CHALLENGES OF BIG DATA**How to store ?****How to query ?****1. Internet****2. Crowd sourcing**

Wikipedia, forums

3. Social networks

Facebook, Instagram

4. Sensors

Mobile phones, GPS

Structured data

Structured data consists of all data which can be stored in relational database in a table with rows and columns.

Unstructured data

Unstructured data is a data which is not organized in a predefined manner or does not have a predefined data model, thus it is not a good fit for a mainstream relational database.

Lec1 (1.2)**Features of big data**

- Big data 5Vs
- Volume
- Velocity
- Variety
- Veracity
- Value

[Benefits from analyzing the data]



large data
fast data generation
Various data types/sources/formats
Accurate and trustworthy
(↑Veracity = ↓data quality)



Some data points are given more weightage than others.

Data as software output are distorted.

Can you name some trustworthy data sources?
Text books, Research papers, Professional magazines

- Name ambiguity
- Sentence ambiguity

Summary of veracity

Big data should have high veracity;
that usually means the data are highly accurate and trustworthy, particularly

1. do not have statistics bias,
2. are not generated by software with bugs,
3. come from reliable data source,
4. have no/little data ambiguation

LeC2 Data models

DATA MODEL AND PHYSICAL STORAGE SCHEME

- Data model describes how data are logically organized.

- Each data model can have different storage schemes.

Example: Relational model can be stored in row-oriented or column oriented.

DATA MODELS

- Relational Data Model (contains set of relations)
 - ❖ Corresponding to relational database

- Key-Value Data Model (no SQL)
 - ❖ Corresponding to key-value systems

- Graph Data Model

❖ Corresponding to graph database

Primary key – Foreign key relationship

Employee (id, name, age, gender, company_id)				
Id	name	age	gender	Foreign Key
				Company Id
0001	Alex	25	M	c0001
0002	Mary	35	F	c0002

Company (Company Id, Company name)		
Company Id	Company name	Country
c0001	Amazon	U.S.
c0002	Tesla	U.S.

A foreign key is a set of one or more columns in a table that refers to the primary key in another table.

RELATIONAL DATA MODEL

Schema → specifies the relation name, and the attribute of each column.

Example:

Employee (id, name, age, gender) ← schema

ID	name	age	gender
0001	Alex	25	M
0002	Mary	35	F

Tuple: typically refers to a row of the relation

Attribute: corresponds to a column of the relation

Attribute			
Id	name	age	gender
Tuple 0001	Alex	25	M
0002	Mary	35	F

Primary Key: A set of attributes that uniquely specify a row (usually there is an ID column)

Primary Key

ID	name	age	gender
0001	Alex	25	M
0002	Mary	35	F

Employee (id, name, age, gender)

Primary Key underlined

TWO REPRESENTATIVE STORAGE SCHEMES

- A relation can be stored row-by-row (such a data system is often called a **row store**)

- A relation can be stored column-by-column (such a data system is often called a **column store**)

- We will discuss in more details when we in later lectures about "column store".

KEY-VALUE DATA MODEL

- The relational model has strict schemas.

- Some big data systems may require **schema-less** models.

- Key-value data model is one such kind.

- Data is represented as a collection of key-value pairs.

- Key uniquely decides the pair

Key	Value
Boston	Massachusetts
New York City	New York
Los Angeles	California

Above are (key, value) pairs describing the mapping between cities and states in the United States.

- It is usually less expressive than relational model but much **simpler**.

- It is preferred by a lot of real-systems including Facebook and Google in analyzing **big data**.

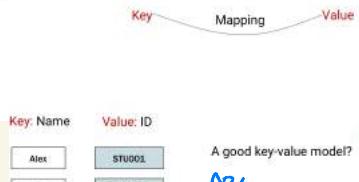
e.g., Google's levelDB, Facebook's RocksDB.

Key	Value
Boston	Massachusetts
New York City	New York
Los Angeles	California

found everywhere

A key-value data model is ubiquitous!
For any **A** that can determine **B** by **key**

CHOOSING THE RIGHT KEY



CHOOSING THE RIGHT KEY

How to put the Tweets data into key-value model?

Friends Tweets & tweets Media Likes

ADM RUMMO (Program - 0002) 2021 Making our community a cleaner place and a better place for everyone! The Data is now available in a key-value store. You can use it to analyze the data and gain insights into the behavior of users. The dataset contains information about users, their posts, and interactions. You may not have an explicit key in the dataset.

You may not have an explicit key in the dataset

Combine cool info to make it unique

AMAZON'S CASE



COMPARING RELATIONAL MODEL AND KEY-VALUE MODEL

- Key-Value model is more flexible, better for scaling up

▪ Favoured by a lot of industrial-level big data systems, e.g., Facebook's RocksDB, Google's LevelDB

▪ Assume most of the queries are simple (example: find a value corresponding to a key or a key range)

▪ It is schema-less, making it commonly used in real-time web-based applications (highly partitionable, easy scaling)

▪ Flexible to handle schema changes

▪ performance driven

▪ not hierarchical
▪ fast time access

- Relational model is more structured

▪ Suitable to handle tabular data

▪ Favoured by accuracy-sensitive systems, e.g., data systems in the bank

▪ It has strict schemas, and is easy to design query languages (e.g., SQLs)

- Data management
▪ Hierarchical

Key-value model can "store" the information of a relation

EXERCISE 1

Converting the following Relation/Table to key-value model

Primary Key

ID	name	age	gender
0001	Alex	25	M
0002	Mary	35	F

SOLUTION

Key	Value
Primary Key	Concatenate Other Attributes
id	name age gender
0001	Alex 25 M
0002	Mary 35 F

- Caching is example of key-value data model



access using
Some Sort of hash (key)

- Key value data models example : - Dynamo DB - Apache HDFS

REMARK 2

Key-value model can be mapped to a conceptual big table in the relational model!

EXERCISE 2



Given the above two tables (Employee and Company). If you do not worry about the storage and always want to query the information of employees, how would you convert them into a key-value model?

SOLUTION

Primary Key	Foreign Key	Primary Key
ID	name age gender	Company ID
0001	Alex 25 M	c0001 Amazon
0002	Mary 35 F	c0002 Tesla

Step 2: Join the table (Left outer-join from Employees)

	name	age	gender	Company name	Company ID
0001	Alex	25	M	Amazon	c0001
0002	Mary	35	F	Tesla	c0002

SOLUTION

Key: Id	Value: name; net worth; rank; company-name; company-id; CEO-name
0001	Jeff Bezos \$161.4B 1 Amazon c0001 Jeff Bezos
0002	Reinard Arnott & Family \$145.4B 2 LVMH c0002 Reinard Arnott

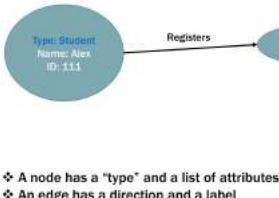
Note: We let key be ID because ID still uniquely defines a row in the big table.

GRAPH MODEL

- ❑ There is another type of database called **graph database**
 - ❑ E.g., Neo4j, OrientDB
- ❑ **Graphs** are the underlying data model of graph databases
 - ❑ A graph is formed by **nodes** and **edges**
 - ❑ A node represents an entity
 - ❑ An edge represents the relationship between entities

GRAPHS ARE UBIQUITOUS

Existing and can be found everywhere



- ❖ A node has a "type" and a list of attributes
- ❖ An edge has a direction and a label

Why do we need graphs (relational) model can do the same?

Suppose we want to model a social network like Facebook

One possible way is to build two relations:

User ID	Name
0001	Alex
0002	Mark
0003	Mary
0004	Bob

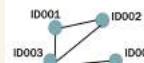
User Table

Friendship Table

Now, how do we answer a typical query :

"find the two most distant users"?

- ❑ There are some queries that require to explore the complex structures of the entities.
- ❑ For these queries, it is more suitable to consider the data as a graph.
- ❑ The social network is modeled as a graph as follows



We can then compute all-pair shortest path algorithms on the graph to answer the query



How to convert the above relations/tables into a graph model?

Primary Key	Foreign Key	Primary Key
ID	Company ID	Company ID
0001	c0001	c0001
0002	c0002	c0002

Type: Employee
id: 0001
Name: Alex
age: 25
gender: M

Type: Company
id: c0001
Name: Amazon

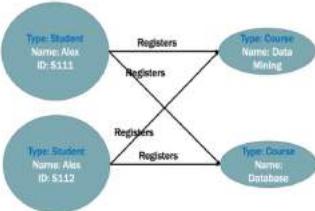
Type: Employee
id: 0002
Name: Mary
Net worth: 35
Gender: F

Type: Company
id: c0002
Name: Telsa

1 row (→) is 1 entity (•)

QUESTION 4

Given the following graph model, please convert it into relational data model.

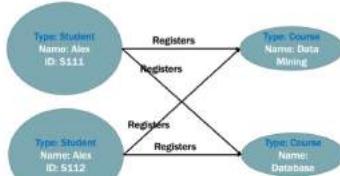


Step 1: consider how many tables are needed.

- Type Student → Student Table
- Type Course → Course Table

QUESTION 4

Given the following graph model, please convert it into relational data model.

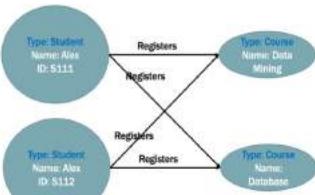


Step 2: find attributes for each table.

Student (Name, ID)
Course (Name)
Register (StudentID, CourseName)

QUESTION 4

Given the following graph model, please convert it into relational data model.

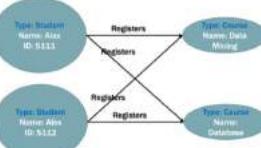


Step 1: consider how many tables are needed.

- Type "Student" → Student Table
- Type "Course" → Course Table
- Relationship "Registers" → Register Table

QUESTION 4

Given the following graph model, please convert it into relational data model.



Student	
ID	Name
S111	Alex
S112	Alex

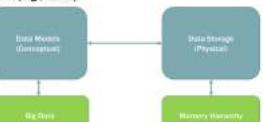
Course	
Name	
Data Mining	
Database	

Register	
StudentID	Name
S111	Data Mining
S111	Database
S112	Data Mining
S112	Database

Lec 3.1 memory Hierarchy I

PREPARATION

- In previous lectures, we have learnt (conceptual) data models. These data models need to be (physically) stored in the storage medium (e.g., disks).



- How to store the data must be related to how to retrieve/query them efficiently.



What is an ideal case for storing big data?



- Ideally, we should have infinite size of fast accessing storage, and they are persistent.
- Infinite size to store big data.
- Fast accessing guarantees fast read/write of the data.
- Persistency ensures keeping the data when we power off the system.

The dilemma of storage design

- Fast storage with large size is expensive; we may afford
- Faster storage with smaller size
- Slower storage with larger size

- Suppose you are given \$1000GD, how do you allocate your budget to buy different types of storage?



We will introduce the concept more formally

what is memory hierarchy?

- The storage space in the computer is used to store data and instructions

- Instructions are the "order" that tell the processor what to do

- The storage space is divided into multiple cells, each having an address

- Most modern computers are byte-addressable

- Each address identifies a single byte of storage

- For example, 256GB memory has $256 \times 1024^3 \times 1024^4$ addresses

THREE MAIN CATEGORY OF STORAGE

Register

Cache Memory

Main Memory

Secondary Memory

CACHE MEMORY

Volatile

- High speed
- Small capacity (often between 8 KB and 64 KB)
- Expensive
- Required as a buffer between the CPU and the slower main memory
- Holds data and program instructions which are most frequently used by the CPU

MAIN-MEMORY

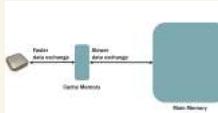
Volatile

- Holds data and instructions on which the computer is currently working
- Relatively high speed but slower than cache memory
- Data is lost if power fails
- Much larger capacity than cache memory (often between 2 GB and 32 GB)
- Less expensive than cache memory

SECONDARY-MEMORY

Non-Volatile

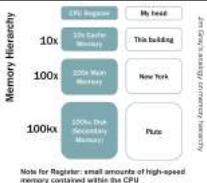
- Also known as external memory
- In many situations, we may simply use the most common example "disk".
- Much slower than main memory when accessing data
- Even with SSD (Solid State Disk), the access time is still higher
- Can store data permanently



MEMORY HIERARCHY

People often simply

- Cache memory → Cache
- Main Memory → Memory
- Secondary Memory → Disk



MORE COMPLICATED MEMORY HIERARCHY

- In fact, cache can be further divided into L1 cache, L2 cache, L3 cache.

- L1 cache is the fastest but smallest
- L2 cache is the slowest but largest
- L3 cache is in the middle

- For simplicity, in this course we simply merge them into a simple cache layer.

WHY WE NEED MEMORY HIERARCHY

- Suppose we need 256GB storage for a computer, can we design a computer with 256GB **cache memory** without main memory and disk/secondary memory?

- It can be too expensive

- No permanent storage

- Suppose we need 256GB storage for a computer, can we design a computer with 256GB **main memory** without cache memory and disk?

- It can still be relatively expensive

- No permanent storage

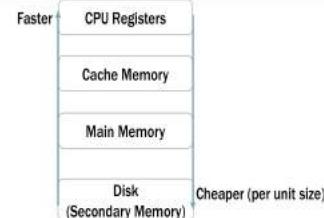
- No buffer between CPU and main memory. The CPU speed can be much faster than main memory, causing latencies.

- We need **fast** memory to sync up CPU speed
 - Processor speed is much faster than main memory access time

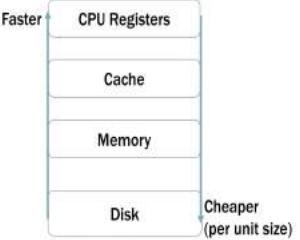
- We need **large** space for **permanently** storing **big data**

- Price becomes a concern
 - Faster memory (e.g., cache memory) is much more expensive

TRADE-OFF BETWEEN SPEED AND PRICE



Faster



SUMMARY OF MEMORY HIERARCHY

- Memory hierarchy consists of a set of memory layers, where a faster memory layer has a smaller capacity.

- Memory hierarchy is needed concerning the following factors (most important ones)

- Reasonable price
- Enough capacity to hold data
- Data persistency

Lec 3.2 Memory Hierarchy II

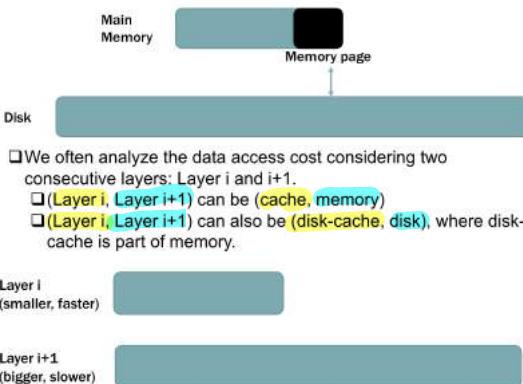
Data in Memory Hierarchy: Basic Cost Analysis



Big data cannot be fully loaded into main memory. Hence, often there are **data movements** between main memory and disks.

DATA ACCESS IN MEMORY HIERARCHY

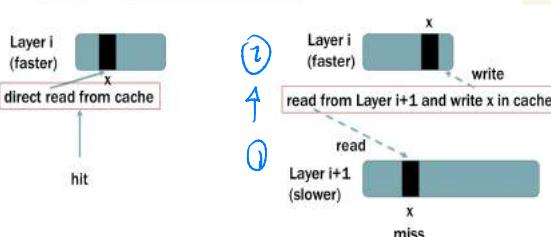
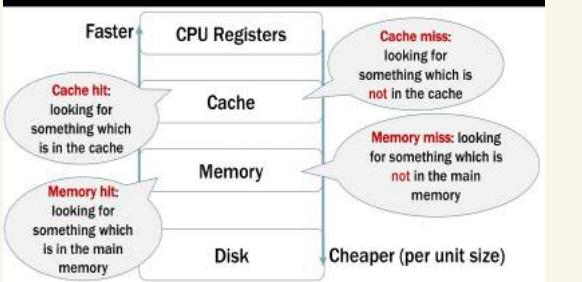
- Big data cannot be fully loaded into main memory. Hence, often there are data movements between main memory and disks.



- We often analyze the data access cost considering two consecutive layers: Layer i and i+1.
 - (Layer i, Layer i+1) can be **(cache, memory)**
 - (Layer i, Layer i+1) can also be **(disk-cache, disk)**, where disk-cache is part of memory.

- Some data item in Layer i+1 has been cached in Layer i
 - When reading a data item in Layer i+1, it will first check whether the data item is cached in Layer i.
 - If yes, directly read the data from the faster Layer i
 - If no, read the data from Layer i+1 and write the data into Layer i+1's cache (usually located in Layer i).

CACHE HIT/MISS AND MEMORY HIT/MISS



COST OF A CACHE MISS

Question:

Suppose a data item is located in **main memory** and can be **cached** in the **cache memory**.

1. accessing cache memory once incurs `cost_access_cache`;
2. accessing main memory once incurs `cost_access_mem`;
3. cache miss rate is h ($0 \leq h \leq 1$).

h = percentage of misses



How to estimate the cost of reading data item?

- If cache hit: `cost_access_cache` x (1-h)

- If cache miss: (`cost_access_cache` + `cost_access_mem`) x h

A simplified estimation

- Overall: `cost_access_mem_overall`

$$= \text{cost_access_cache} \times (1-h) + (\text{cost_access_cache} + \text{cost_access_mem}) \times h$$

Cache hit *Cache miss*

- Overall: `cost_access_mem_overall`

$$= \text{cost_access_cache} \times (1-h) + (\text{cost_access_cache} + \text{cost_access_mem}) \times h$$

- If $h = 1$ (cache miss rate is high), then the cost is

$$\text{cost_access_cache} + \text{cost_access_mem} \approx \text{cost_access_mem}$$

The above estimation is due to the fact that `cost_access_mem` is significantly higher (say, 1000x) than `cost_access_cache`

- If $h = 0$ (cache miss rate is low), then the cost is `cost_access_cache`

- Overall: `cost_access_mem_overall`

$$= \text{cost_access_cache} \times (1-h) + (\text{cost_access_cache} + \text{cost_access_mem}) \times h$$

- Summary

If cache miss rate is low, then data read performance is close to cache read performance.

If cache miss rate is high, then data read performance is close to memory read performance.



It is important to minimize the cache miss rate!

COST OF A MEMORY MISS

Question:

Suppose data items are located in the **disk** and can be cached in the **main memory** as well.

(We only consider **two layers**)

1. Accessing disk once incurs a cost of `cost_access_disk`;
2. Accessing main memory once incurs a cost of `cost_access_mem`;
3. Memory miss rate is h^* ($0 \leq h^* \leq 1$).

How to estimate the cost of reading the data item?



- If memory hit: `cost_access_mem` x (1-h*)

- If memory miss: (`cost_access_disk` + `cost_access_mem`) x h*

- Overall: `cost_access_disk_overall`

$$= \text{cost_access_mem} \times (1-h^*) + (\text{cost_access_disk} + \text{cost_access_mem}) \times h^*$$

- `cost_access_disk_overall` =

$$\text{cost_access_mem} \times (1-h^*) + (\text{cost_access_disk} + \text{cost_access_mem}) \times h^*$$

- If $h^*=1$ (high miss rate):

$$\text{cost_access_disk_overall}$$

$$= \text{cost_access_disk} + \text{cost_access_mem} \approx \text{cost_access_disk}$$

The above step is due to the fact that `cost_access_disk` significantly larger than (e.g., 1000x) `cost_access_mem`

- If $h^*=0$: `cost_access_disk_overall` = `cost_access_mem`

- `cost_access_disk_overall` =

$$\text{cost_access_mem} \times (1-h^*) + (\text{cost_access_disk} + \text{cost_access_mem}) \times h^*$$

- Summary

If $h^*=1$ (high miss rate), then the read performance is close to the read performance of disk

If $h^*=0$ (low miss rate), then the read performance is close to the read performance of memory.



What about considering more than two layers?

We will see this in tutorial questions.

How to obtain a smaller cache miss rate?

We will see this in the coming lectures about cache conscious designs.

Data in Memory Hierarchy:

Page-based Access

SMALLEST UNIT TRANSFERRED BETWEEN TWO LAYERS

- The data unit swapped between cache memory and main memory is called **cache line**
 - Cache line size is usually 32, 64 and 128 bytes
 - A cache line contains continuous memory segment



- The data unit swapped between cache memory and main memory is called **cache line**

- Cache line size is usually 32, 64 and 128 bytes
- A cache line contains continuous memory segment

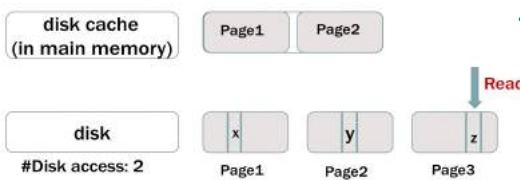
- The data unit swapped between main memory and disk is called **(memory) page**.

- Page size is usually about 4KB
- A page contains continuous memory segment

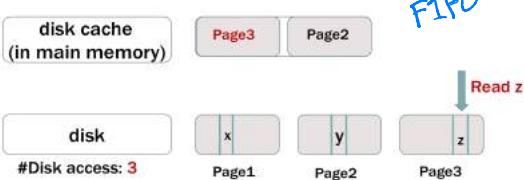
- We use "**page**" as a common term to refer to the transfer data unit between Layer i and Layer i+1.

Main memory refers to cache
Secondary memory refers to disk

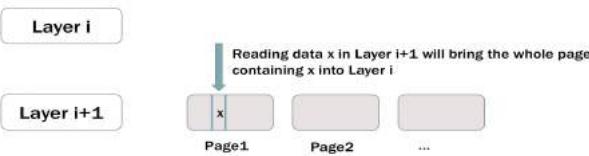
Now, what if we read the third page? Now disk cache is full.



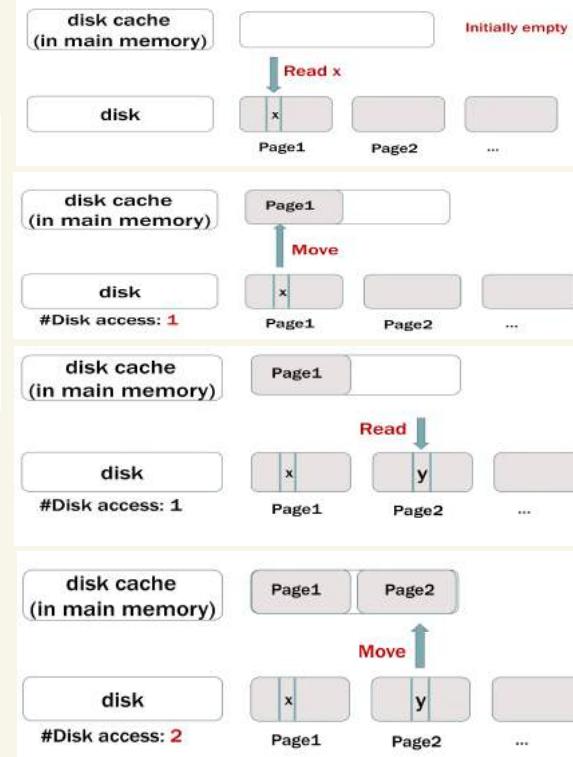
Now, what if we read the third page? Now disk cache is full.



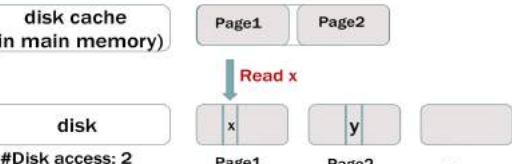
EXAMPLE OF PAGE-BASED ACCESS



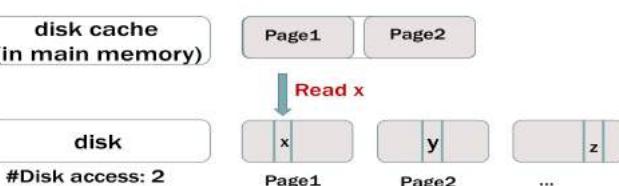
Note: A page can contain many data items.



Now, if we further read x, no additional disk accesses because it is in main memory already.



Now, what if we read the third page?



EXAMPLE: SCANNING ARRAYS

Query $x < 4$ from the following data

(size=120 bytes)
Memory Layer i

Memory Layer i+1

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

Cost: 40 Bytes

Query $x < 4$ from the following data

Scan → Qualified results



Memory Layer i+1

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

Cost: 80 Bytes

Query $x < 4$ from the following data

Scan → Qualified results



Memory Layer i+1

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

Cost: 120 Bytes

Query $x < 4$ from the following data

Scan → Qualified results



Memory Layer i+1

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

CAN WE DO BETTER ABOUT SCANNING?

- Consider a cost-free magic function for the example query:

By accessing a page, the function immediately tells you the positions of qualified keys (i.e., $x < 4$) within the page.

- Consider another cost-free magic function:

The function tells you which pages will contain the qualified keys (i.e., $x < 4$).

- Can we do better with such a function?
- Which one do you think is more useful?



TRY THE FIRST MAGIC FUNCTION

Consider a cost-free magic function for the example query:

By accessing a page, the function immediately tells you the positions of qualified keys (i.e., $x < 4$) within the page.



WITH MAGIC FUNCTION (FIRST)

Query $x < 4$ from the following data

(size=120 bytes)
Memory Layer i

Memory Layer i+1

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

Cost: 40 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

Memory Layer i+1 Magic function read

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

Cost: 40 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

Memory Layer i+1 Magic function read bring

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

Cost: 80 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

Memory Layer i+1 Magic function read

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

Cost: 80 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

Memory Layer i+1 Magic function read bring

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

Cost: 120 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

Memory Layer i+1 Magic function read

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

Cost: 120 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

Memory Layer i+1 Magic function read bring

5, 10, 7, 4, 12 2, 8, 9, 11, 7 7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

Cost: 120 Bytes

Query $x < 4$ from the following data

Qualified results

CAN WE DO BETTER ABOUT SCANNING?

WITH MAGIC FUNCTION (SECOND)

Query $x < 4$ from the following data

- Consider a cost-free magic function regarding the example query:

By accessing a page, the function immediately tells you the positions of qualified keys (i.e., $x < 4$) within the page.

This seemingly good function does not help much. The reason is that the data movement is based on pages.

TRY THE SECOND MAGIC FUNCTION

- Consider another cost-free magic function:

The function tells you **which pages** will contain the qualified keys (i.e., $x < 4$). Can we do better with such a function?

In some situations, yes!



(size=120 bytes)
Memory Layer i

Memory Layer i+1

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes $\times 5 = 40$ bytes

Cost: 40 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

Memory Layer i+1

Magic function read Page 2

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes Page 1

Page 2

Page 3

Each page: 8 bytes $\times 5 = 40$ bytes

Cost: 40 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

2, 8, 9, 11, 7

2

Memory Layer i+1

Magic function read Page 2

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes Page 1

Page 2

Page 3

Each page: 8 bytes $\times 5 = 40$ bytes

Cost: 80 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

2, 8, 9, 11, 7

7, 11, 3, 9, 8

2, 3

Memory Layer i+1

Magic function read Page 3

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes Page 1

Page 2

Page 3

Each page: 8 bytes $\times 5 = 40$ bytes

SUMMARY OF MAGIC FUNCTION EXERCISE

Take-away message

In general, if there is a magic function telling **which pages** locate the **qualified data**, and if these pages are only a subset of all pages, the read cost will be smaller than directly scanning all pages.

- In practice, we do not have such a magic function with a free cost. Typically, we design indexes, and put the indexes to a faster memory-layer so that its cost is minor compared with the slower memory-layer.

Lec 3.3 Memory Hierarchy III

Cache Conscious Data Processing

OVERVIEW

- In previous lectures, we show the basic concepts and analysis of memory hierarchy.
- In this lecture, we show some design principles to make use of memory hierarchy for processing big data. We will discuss following two cases.
 - Array access patterns
 - Spatial locality designs
 - Temporal locality designs
 - Big data sorting

MOTIVATING QUESTION

Suppose we have a 10000×10000 matrix of integers, which is stored in a 2-dimensional array $A[10000][10000]$. We want to set all of its value to 1. Cache size = 5000 integers. How would you write your code?

Row

```
Solution X
for (int i=0;i<10000;i++)
    for(int j=0;j<10000;j++)
        A[i][j]=1;
```

Column

```
Solution Y
for (int j=0;j<10000;j++)
    for(int i=0;i<10000;i++)
        A[i][j]=1;
```

X ? simulation X takes 22 Secs
Y takes 7.1 Secs

Which one is better, X or Y?

ACCESSING AN ARRAY

- We may access an array with different **access patterns**
- Access pattern means the position sequence of accessing an array.
- Different access patterns may impact the utility of cache

ACCESS PATTERN 1

We have a 10-integer array stored in main memory.
cache size = 5 integers, transfer size (cache line)= 5 integers

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

Memory

a	b	c	d	e	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

Memory

a	b	c	d	*	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

Memory

a	b	c	d	*	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

Memory

a	b	c	d	*	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

Memory

a	b	c	d	*	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

a	b	c	d	e
---	---	---	---	---

Cache Miss: 1
Cache Hit: 4

Memory

a	b	c	d	e	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

f	g	h	i	j
---	---	---	---	---

Replaced

Cache Miss: 2
Cache Hit: 4

Memory

a	b	c	d	e	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

f	g	h	i	j
---	---	---	---	---

Cache Miss: 2
Cache Hit: 5

Memory

a	b	c	d	e	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

f	g	h	i	j
---	---	---	---	---

Cache Miss: 2
Cache Hit: 6

Memory

a	b	c	d	e	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

f	g	h	i	j
---	---	---	---	---

Cache Miss: 2
Cache Hit: 7

Memory

a	b	c	d	e	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Cache

f	g	h	i	j
---	---	---	---	---

Cache Miss: 2
Cache Hit: 8

Memory

a	b	c	d	e	f	g	h	i	j
1	2	3	4	5	6	7	8	9	10

6 not in Cache
load new cache line

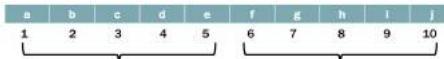
ACCESS PATTERN 2

Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache



Memory

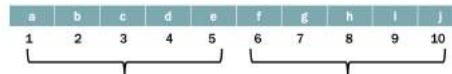


Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache



Memory

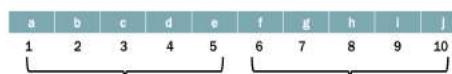


Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache



Memory

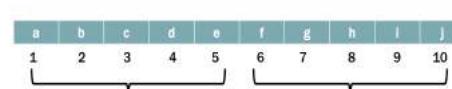


Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache



Memory

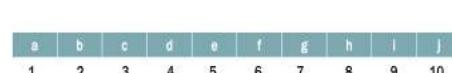


Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache

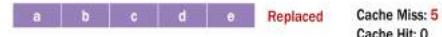


Memory



Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache



Memory

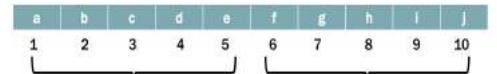


Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache



Memory

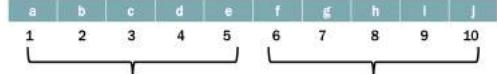


Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache



Memory

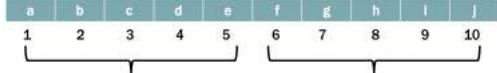


Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache

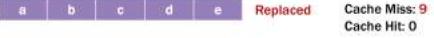


Memory

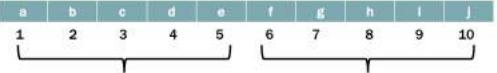


Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache



Memory

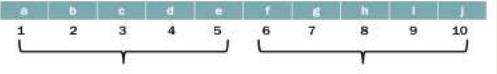


Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache



Memory



DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	2, 13
B	7, 10
C	6, 11
D	5, 12
E	1, 14

Write buffer (size 2)



2 digits

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	2, 13
B	7, 10
C	6, 11
D	5, 12
E	1, 14

Write buffer

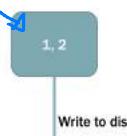


Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	13
B	7, 10
C	6, 11
D	5, 12
E	14

Write buffer



Write to disk

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	13
B	7, 10
C	6, 11
D	5, 12
E	14

Write buffer



Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	13
B	7, 10
C	6, 11
D	12
E	14

Write buffer



Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	13
B	15, 16
C	11
D	12
E	14

Write buffer



Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	13
B	7, 10
C	11
D	12
E	14

Write buffer

5, 6

Write to disk

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	13
B	7, 10
C	11
D	12
E	14

Write buffer

7

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	13
B	10
C	11
D	12
E	14

Write buffer

7, 10

empty=refill
1 cost

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	13
B	15, 16
C	11
D	12
E	14

Write buffer

7, 10

Write to disk

COST ANALYSIS

Let N be the data size, M be the memory size, and the B be the page (transfer unit) size.

- Step 1: Read all the N items once and write back to the disk once, incurring $O(N/B)$ I/Os.

$\frac{1}{B} \text{ I/O} = \frac{1 \text{ page transfer cost}}{\text{there are } N \text{ items}} = \frac{N}{B} = \text{Read}$

- Step 2&3: During merge and writeback, each item is read from and written to the disk exactly once, incurring $O(N/B)$ I/Os.

Does this analysis apply to all situations?



To make the analysis hold, we should have

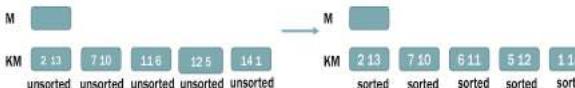
- The size of write buffer is at least one page size (size B)
- The process of in-memory merge sort should be fit in main memory.

Does this analysis apply to all situations?



SORTING

Suppose the size of main memory is M , the array size is $KM (=N)$.

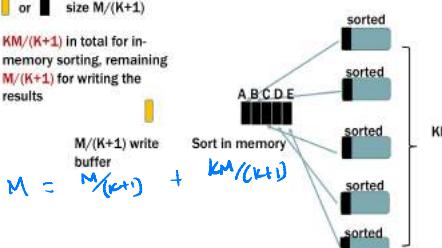


Step 1:

Cut KM arrays into K parts. Each part is put in main memory to sort (using any sorting algorithm we learnt, e.g., quick sort)

or size $M/(K+1)$

$KM/(K+1)$ in total for in-memory sorting, remaining $M/(K+1)$ for writing the results



Step 2:

Apply K-way merge sort to the first $M/(K+1)$ of each sorted part.

THE RANGE OF K

Condition 1:

A write buffer has at least one page size (denoted by B).

Hence, we have $M/(K+1) \geq B$, giving us $K \leq \frac{M}{B} - 1$

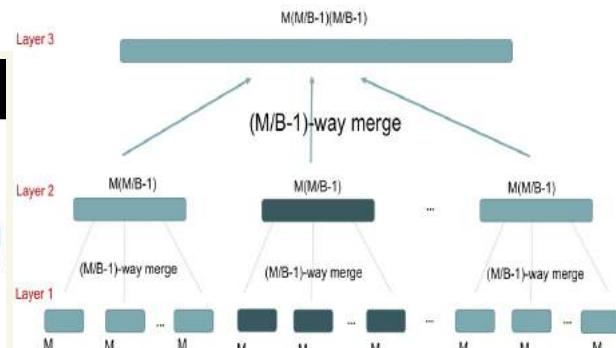
Condition 2:

The process of K -way merge sort should be fit in remaining memory size (excluding write buffer).

Hence, $K \times B \leq KM/(K+1)$, also giving us $K \leq \frac{M}{B-1}$

EXTEND TO GENERAL CASE

- The cost analysis holds when N is at most $(M/B-1)M$.
- We can extend the method to general N by recursively applying the $(M/B-1)$ -way merge.

RECURSIVELY APPLY $(M/B-1)$ -WAY MERGE Not testedGENERAL ANALYSIS not tested

- If there are L layers, then $M(M/B-1)^{L-1} = N$ because N items should be sorted, giving

$$L = O(\log_{M/B-1} N/M)$$

$$M(M/B-1)^{L-1} = N$$

$$\Leftrightarrow (M/B-1)^{L-1} = N/M$$

$$\Leftrightarrow \log_{M/B-1}(M/B-1)^{L-1} = \log_{M/B-1} N/M$$

$$\Leftrightarrow L-1 = \log_{M/B-1} N/M$$

$$\Leftrightarrow L = 1 + \log_{M/B-1} N/M$$

$$\Leftrightarrow L = O(\log_{M/B} N/M)$$

- The merges in each layer costs $O(N/B)$ I/Os.

- Hence, the total cost is $O((N/B) \log_{M/B} N/M)$ I/Os.

1. Column Store I

PREPARATION

- We learnt some principles of data models, memory hierarchy, and cache-conscious designs.

- In this lecture, we will see how the previous learnt concepts impact the design of modern big data system - column store

Main Designs of Column Store

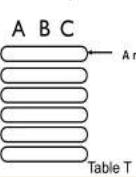
COLUMN STORE HANDLES TABULAR DATA

Column A	Column B	Column C
id	name	age
Row 1 0001	Alex	25
Row 2 0002	Mary	35

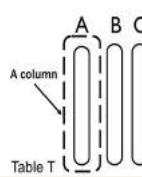
MAIN DESIGN OF COLUMN STORES

- Data in column stores are column-oriented
 - Also called column-oriented database, or columnar database.

row-store (traditional RDB)



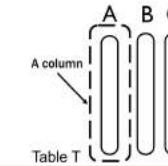
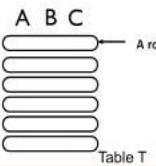
column-store



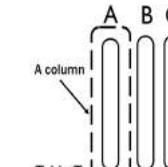
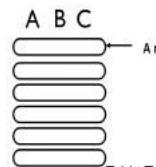
An excellent introduction video here:

First 3 minutes of <https://www.sisense.com/glossary/columnar-database/>

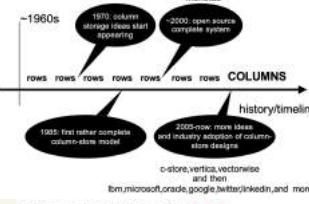
- Consider the following cases:
- select A from T where A>4



- Column store is efficient to read just the columns you need for a query.
 - No need to extract the other columns.
 - In fact, commercial data table may have tens of columns. Not every column is touched in a query.

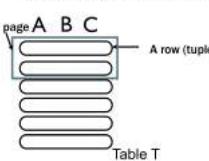


- Column stores are compression friendly.
 - Same type of data are stored consecutively → easier to compress
 - More compressed data can reduce the cost moving data from disks to main memory.

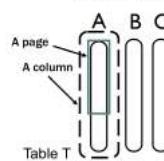


- Data are stored based on [pages](#)
- A file consists of multiple pages.
- A page is usually considered as the transfer unit between disk and memory
- A page may contain multiple rows (tuples) in row-store
- Data are stored based on [pages](#)

row-store (traditional RDB)



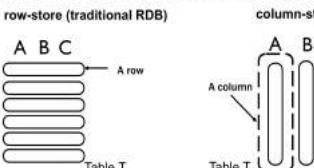
column-store



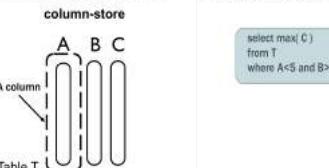
WHY COLUMN STORE

- Recently, column-store becomes one of the major systems in the industry. Can you work out in what situations column-stores may have benefits?

row-store (traditional RDB)



column-store

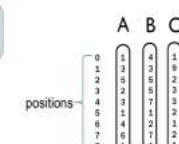


QUERYING WITH COLUMN STORE

- The whole table is stored in disk

- Have disk cache (in main memory) whose size is a multiple of page size

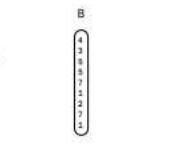
Table T



QUERYING WITH COLUMN STORE (FLOW CHART)

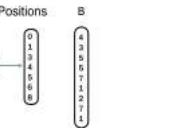
- The whole table is stored in disk

- Have disk cache (memory buffer) whose size is a multiple of page size



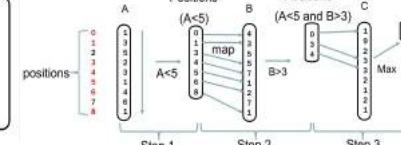
- The whole table is stored in disk

- Have disk cache (memory buffer) whose size is a multiple of page size



- The whole table is stored in disk

- Have disk cache (memory buffer) whose size is a multiple of page size



DETAILS OF STEP 1

- The whole table is stored in disk

- Have disk cache (memory buffer) whose size is a multiple of page size

- Step 1

- positions (A<5)

- Step 2

- positions (A<5 and B>3)

- Step 3

```
int j=0;
for (int i=0; i<#tuples; i++)
{
  if(A[i]<5)
  {
    Pos[i]=i;
    j++;
  }
}
```

- The whole table is stored in disk

- Have disk cache (memory buffer) whose size is a multiple of page size

- Step 1

- positions (A<5)

- Step 2

- positions (A<5 and B>3)

- Step 3

- positions (C)

- Max

- Int max;

- for (int i=0; i<#tuples; i++)

- {

- if(i==0) max=C[Pos2[i]];

- else if(max>C[Pos2[i]])

- {

- max=C[Pos2[i]];

- }

- }

- map

- B>3

- C

- Max

- Step 3

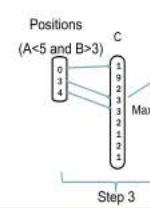
- Step 2

- Step 1

- positions (A<5 and B>3)

- C

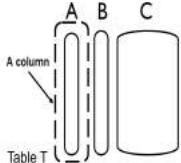
- Max



COLUMN CAN BE OF DIFFERENT WIDTHS

Different columns do not have to have the same width

e.g., in Employee(Name, Gender, Emails), all three attributes may occupy different bytes.



IN-MEMORY V.S. DISK-BASED IMPLEMENTATIONS

Previous pseudo codes are illustrated based on in-memory implementations.
Now we discuss the disk-based implementation.

The major difference is that

We cannot allocate a big array A or Pos because the data volume can be huge in practice.

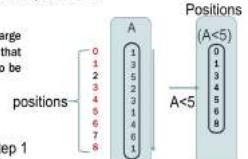
The data is mainly stored in files, but it allows some buffers in the main memory.

```
int j=0;
for (int i=0; i<#tuples; i++)
{
    if(A[i]<5)
    {
        Pos[i]=j;
        j++;
    }
}
```

Usually, each column is stored as a file (or multiple files) in the disk.
The scanning of the column involves file access, e.g.,

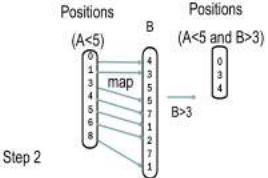
- ❖ For C programming: fread()
- ❖ For JAVA programming: various "FileInputStream"

The position list can be overly large
(larger than the buffer). If that happens, the position list can also be stored in a file.

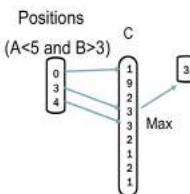


In step 2, the "map" process can be implemented in a disk-based manner.
Since column B is stored in a file, when given a position, we need to seek the corresponding values in the file containing column B.

❖ For example, using fseek() in C programming, it directly shifts to the desired starting point of the file.



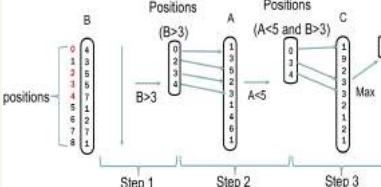
For step 3, similar "map" process can be done in a disk-based manner.



ALTERNATIVE SCHEMES

Can also check B>3 first.

Can you write down the procedure of this alternative approach?



Also, one can do the following:

Step 1: Scan A and obtain the qualified results

Step 2: Scan B and obtain the qualified results

Step 3: Merge the results (intersect the results in the previous case).

Option 1: Scan A first

Option 2: Scan B first

Option 3: Scan A and B independently

Which is the best?



Usually, Option 3 is not as good as Option 1 (or Option 2) because it needs to scan entire A and B

Option 1 and Option 2: which is better roughly depends on the number of qualified results after processing that column.

- ❑ If scanning A gets much less results than B, then scanning A first is better.
- ❑ If scanning B gets much less results than A, then scanning B first is better.

Query Cost Analysis

COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Suppose a column has Z values:

Column width = w bytes;

Page size = P bytes;

Storing a position costs 4 bytes;

Each column is sequentially stored in the disk.

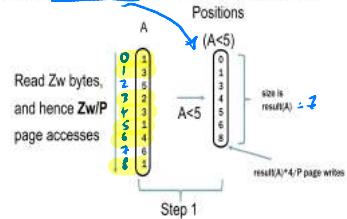
We assume position list is stored in files.

We also assume w < P and each value in a column is contained in a page.



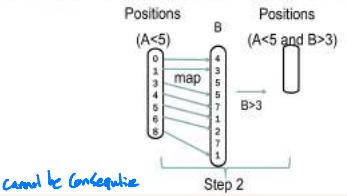
In this particular example, w=4 bytes (integer size)

Step 1: Scanning A costs Zw/P page reads and result(A)*4/P page writes.
(note: result(A) is the number of qualified positions after processing column A.)



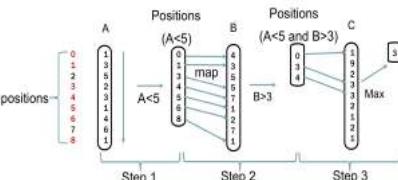
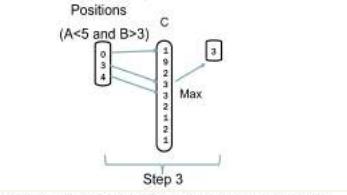
Step 2:

Reading the positions (for A<5) costs result(A)*4/P page reads;
Fetching B costs at most result(A) page reads and result(AB)*4/P page writes.



Why the cost of writing positions for B costs result(A)*4/P page writes instead of result(A)/4 page writes? Sequential writes!

Step 3: Reading the positions (A<5 and B>3) costs result(AB)*4/P page reads;
Fetching C costs at most result(AB) page reads.



In many cases, position list can be small enough to be simply stored in the memory buffer

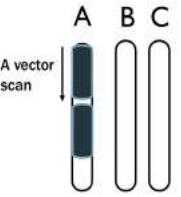
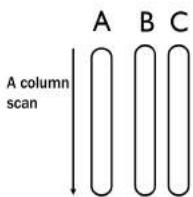
In these cases, cost for column store (number of page access):
 $Zw/P + 2(result(A)*4/P + result(A) + 2(result(AB)*4/P + result(AB)))$
 $= 2zw/P + result(A) + result(AB)$



CACHE OPTIMIZATION (extended discussion)

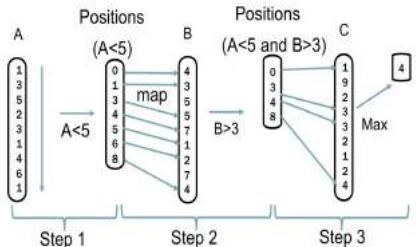
CACHE OPTIMIZATION: VECTOR AT A TIME

Modern column stores (e.g., VectorWise) employ **vector-at-a-time** procedure, utilizing cache locality. This is called **vectorized processing**.

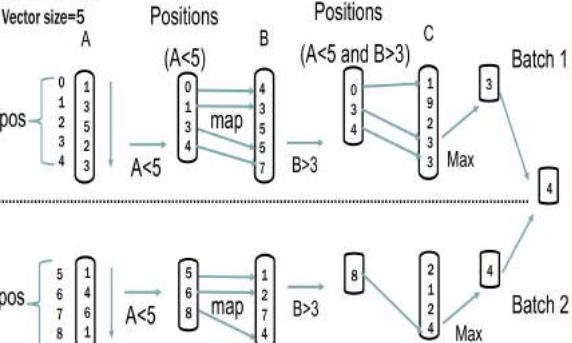


EXAMPLE

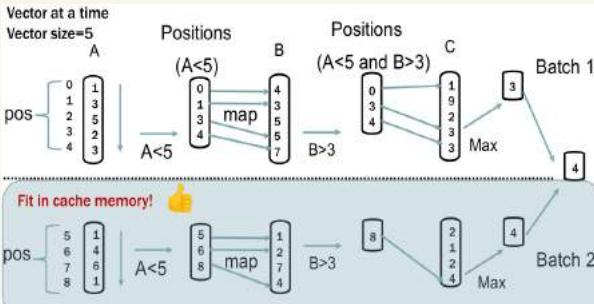
Column at a time



Vector at a time

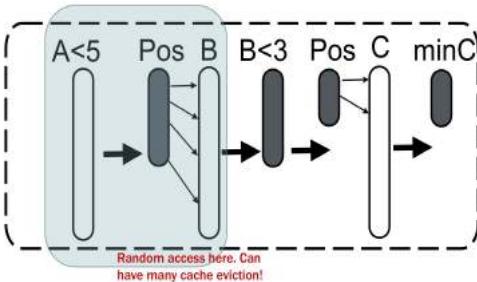


What's the benefits of vectorized processing?



VECTOR AT A TIME IS MORE CACHE FRIENDLY

Column at a time



BENEFITS OF VECTORIZED PROCESSING

- It is recommended to set the vector size to be smaller than cache size (L1 cache) to allow for auxiliary data structure. Hence, it guarantees each batch has a good cache locality.

- For some queries, we can get the results on the fly.

What can be a possible issue of a column store?



What if queries do not come at the same time?

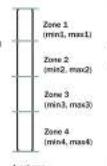
Gather queries in a batch

Schedule the queries on same data to run in parallel

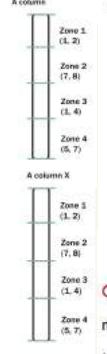
Each query gets a thread/core

ZONE MAP

- A common way to help column scan is the zone map.
- It separates a column into "zones", each is computed with max and min.
- In filtering, some zones can be skipped.



- Select from T where X>3 and X<6
- Which zones need to be scanned?



- Select from T where X>3 and X<6
- Which zones need to be scanned?
- We only need to scan Zone3 and Zone4 because
 - [1, 4] overlaps with [3, 6]
 - [5, 7] overlaps with [3, 6]
 - [1, 2] does not overlap with [3, 6]
 - [7, 8] does not overlap with [3, 6]



- Select from T where X>3 and X<6
- Which zones need to be scanned?
- We only need to scan Zone3 and Zone4 because
 - [1, 4] overlaps with [3, 6]
 - [5, 7] overlaps with [3, 6]
 - [1, 2] does not overlap with [3, 6]
 - [7, 8] does not overlap with [3, 6]
- Usually, a zone is of a page size
- We skip the scanning of two pages

- How to store min and max from each zones?
- They are stored together, probably in a few disk pages.
- These zone map pages can be loaded into memory when system is started
- So the additional cost of reading zone maps is very low

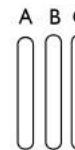
- Is zone map always effective? Why?
- Data can be uniformly distributed.



SELECT max(C) FROM T WHERE A>10 and A<40 and B>20



SELECT max(C) FROM T WHERE A>10 and A<40 and B>20



- Scan A and select A in (10, 40)
- Among those positions returned in 1), scan B and select B>20
- Among those positions returned in 2), select max(C)

SELECT max(C) FROM T WHERE A>10 and A<40 and B>20

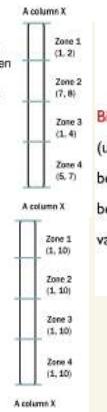
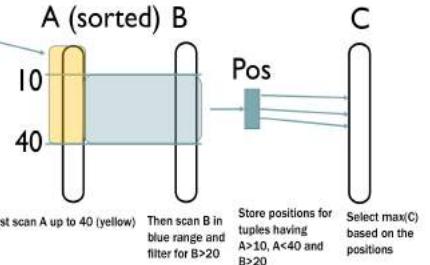


What if A is sorted?



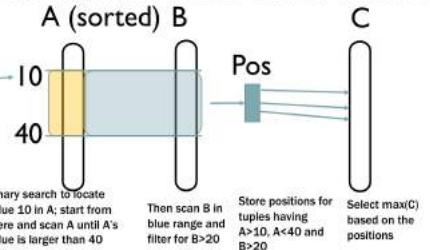
SELECT max(C) FROM T WHERE A>10 and A<40 and B>20

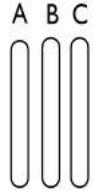
Optimization: No
need to scan from
the beginning.



SELECT max(C) FROM T WHERE A>10 and A<40 and B>20

Binary search
(usually faster
because there can
be many
values<10)





When sorting A,
do the other columns change?



ENHANCED WITH INDEX

What is an index?

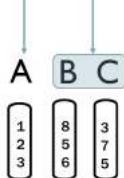
Roughly speaking, in column store, an index is an affiliated data structure built on a column, and it can efficiently help locate the position of a value in the column. It is a widely used technique in data management.

*more discussions
Hybrid layouts*

A B C To apply the above query scheme, the other columns need to be rearranged to follow the order of A



sorted followed



A B C

2	5	7
1	8	3
3	6	5

LIMITATION OF SORTING

- ❑ Can only sort one column (because the other columns need to follow the order of the sorted column)
- ❑ Filtering based on unsorted column is still as costly.
- ❑ Better solution: use index

Why do we need index?

Imaging how do you find a book in the library

Business Library

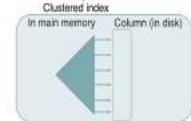
Available , Level B1: A-HG; Level B4: HJ-Z HM131.S431 2003

1. Find the bookrack corresponding to HJ-Z will help you find the book
2. Find the range for HM131

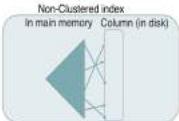
This is a kind of index

In column store, we can build index for a column

If the column is sorted

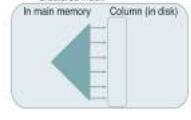


If the column is unsorted

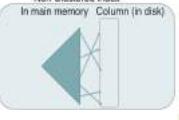


- ❑ In one table, we can apply both clustered index and non-clustered index.
- ❑ One clustered index for a table (why?)
- ❑ Can have multiple non-clustered index for a table
- ❑ Clustered index is typically faster in extracting values compared with non-clustered index (why?)

Clustered index

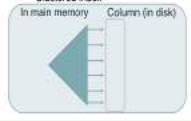


Non-Clustered index

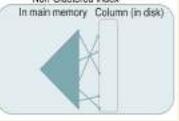


- ❑ Index access is much faster than data access, because index is stored in Main Memory, while data is stored in disk.
- ❑ So it is beneficial to have complicated index access to locate the exact pages where stores the data.

Clustered index

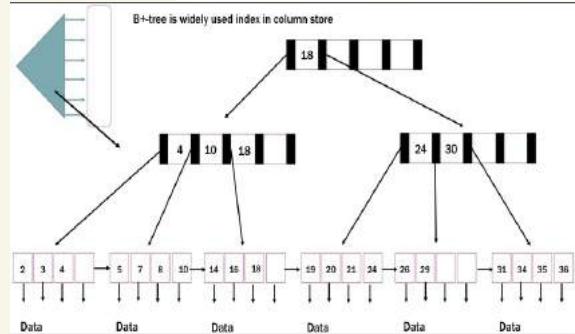


Non-Clustered index



*B+tree
not tested*

B+tree is widely used index in column store



QUESTION 1

Use shared scan to find the 2nd largest value and 2nd smallest value in an integer array A of size t. Assume that the integers in A are different.

```

int max = -1;
int second_max = -1;
for (i=0; i<t; i++) {
    if (A[i] > max) {second_max = max; max = A[i];}
    if (A[i] <= max && A[i] > second_max) second_max = A[i];
}

```

RECAP - WHAT IS SHARED SCAN

Shared scan minimizes the scan cost. Typically, one pass of scan is preferable.

Think about the question... How to write the code?

LET'S START WITH WHATEVER WE HAVE

First, we can use the following code to compute the max and second max of array A.

```

max = -∞;
second_max = -∞;
for (i=0; i<t; i++) {
    if (A[i] > max) {second_max = max; max = A[i];}
    if (A[i] <= max && A[i] > second_max) second_max = A[i];
}

```

Second, we can use the following code to compute the min and second min of array A.

```

min = ∞;
second_min = ∞;
for (i=0; i<t; i++) {
    if (A[i] < min) {second_min = min; min = A[i];}
    if (A[i] >= min && A[i] < second_min) second_min = A[i];
}

```

SHARED SCAN

Solution:

The idea of shared scan is to do multiple operations in one scan of the data.

```

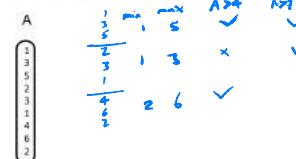
max = -∞;
second_max = -∞;
min = ∞;
second_min = ∞;
for (i=0; i<t; i++) {
    if (A[i] > max) {second_max = max; max = A[i];}
    if (A[i] <= max && A[i] > second_max) second_max = A[i];
    if (A[i] < min) {second_min = min; min = A[i];}
    if (A[i] >= min && A[i] < second_min) second_min = A[i];
}

```

QUESTION 2

Suppose we have the following column applied with zone map of size 3.

- 1) Illustrate the information recorded for each zone.
- 2) Give the steps of using zone map to answer queries "A>4"
- 3) Give the steps of using zone map to answer queries "A>7 or A<2"



Suppose we have the following column applied with zone map of size 3.

- 1) Illustrate the information recorded for each zone.
- 2) Give the steps of using zone map to answer queries "A>4"
- 3) Give the steps of using zone map to answer queries "A>7 or A<2"

QUESTION 3

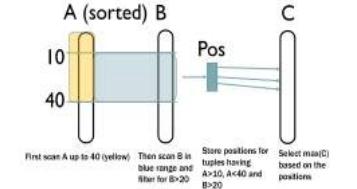
In lecture slides, we discuss two ways of scanning a sorted column: scanning from the start and scanning using binary search.

Give one extreme scenario where the first method is better.

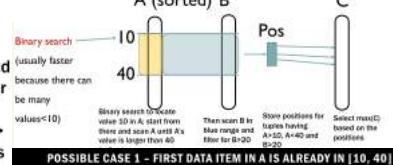
- if the item we scanning for is at the start

RECAP - SCANNING FROM THE START

SELECT max(C) FROM T WHERE A>10 and A<40 and B>20

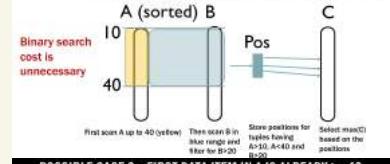


SELECT max(C) FROM T WHERE A>10 and A<40 and B>20



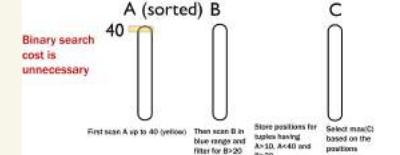
POSSIBLE CASE 1 - FIRST DATA ITEM IN A IS ALREADY IN [10, 40]

SELECT max(C) FROM T WHERE A>10 and A<40 and B>20



POSSIBLE CASE 2 - FIRST DATA ITEM IN A IS ALREADY >= 40

SELECT max(C) FROM T WHERE A>10 and A<40 and B>20



QUESTION 4

Given a table T of two columns A and B. Assume that the table is stored in a column store, and it has two copies: the first copy has column A sorted (while column B follows the order of column A); the second copy has column B sorted (while column A follows the order of column B).

(1) For query "select max(B) from T where A>3 and A<6", which copy should be used for lower cost?

1 \leftrightarrow

(2) For query "select max(A) from T where B>3 and B<6", which copy should be used for lower cost?

2 \leftrightarrow

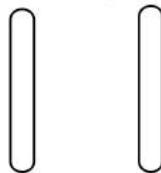
(3) We can also use multiple copies simultaneously. Please describe possible issues of using multiple copies.

check vid

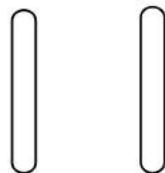
SOLUTIONS (1) (2)

Queries on sorted columns are faster!

A (sorted) B



A B (sorted)



For query "select max(B) from T where A>3 and A<6", which copy should be used for lower cost?

For query "select max(A) from T where B>3 and B<6", which copy should be used for lower cost?

QUESTION(3)-OPEN DISCUSSION

Describe possible issues of using multiple copies.

Think...

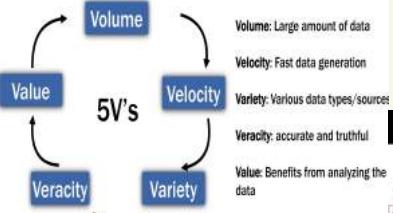


(1) Large space

(2) Updates have to be conducted on each copy

OVERVIEW (1ST HALF)

- Quickly go through key knowledge we have learnt, including
 - Big Data 5V's
 - Data Models
 - Memory Hierarchy
 - Column Store



DATA MODELS

- Relational Data Model
 - Corresponding to relational database
- Key-Value Data Model
 - Corresponding to key-value systems
- Graph Data Model
 - Corresponding to graph database

RELATIONAL DATA MODEL

Primary key – Foreign key relationship



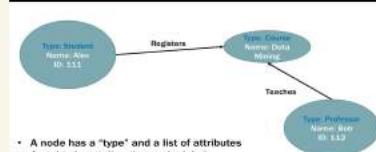
KEY-VALUE DATA MODEL

Key-value Data Model is ubiquitous!

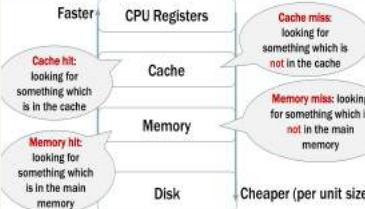
For any A that can determine B

Key Value

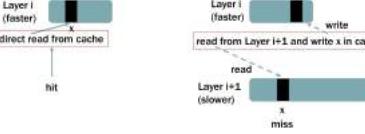
GRAPHS ARE UBIQUITOUS



MEMORY HIERARCHY



DATA ACCESS IN MEMORY HIERARCHY



$$\text{cost_access_mem_overall} = \text{cost_access_cache} \times (1 - \text{missrate}) + (\text{cost_access_cache} + \text{cost_access_mem}) \times \text{missrate}$$

CACHE CONSCIOUS DESIGNS

We show some design principles to make use of memory hierarchy for processing big data. We will discuss following two cases.

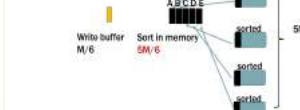
- Array access patterns
 - Spatial locality designs
 - Temporal locality designs

- Big data sorting

SORTING

or $\lceil \frac{n}{M} \rceil$

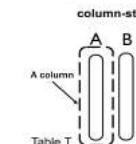
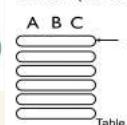
$5M/6$ is total in-memory sorting, remaining $M/6$ for writing the results



MAIN DESIGN OF COLUMN STORES

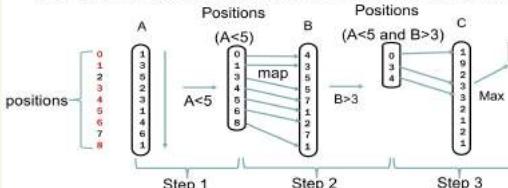
- Data in column stores are column-oriented
- Also called column-oriented database, or columnar database.

row-store (traditional RDB)



QUERYING WITH COLUMN STORE (FLOW CHART)

- The whole table is stored in disk
- Have disk cache (memory buffer) whose size is a multiple of page size



COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Without using an index, handling the query in the row store needs to read through the whole table. Namely, it costs $Z^w \cdot 3/P$ page accesses.

$$\begin{aligned} \text{Cost for column store (number of page access):} \\ Z \cdot w \cdot P + 2 \cdot \text{result}(A) \cdot 4/P + \text{result}(A) \cdot 2 \cdot \text{result}(AB) \cdot 4/P + \text{result}(AB) \\ = Z \cdot w \cdot P + \text{result}(A) \cdot (8/P + 1) + \text{result}(AB) \cdot (8/P + 1) \\ = Z \cdot w \cdot P + \text{result}(A) + \text{result}(AB) \end{aligned}$$

Cost for row store (number of page access):
 $3Z \cdot w \cdot P$

Note: the values of $\text{result}(A)$ and $\text{result}(AB)$ are typically much smaller than Z . So we can conclude in this case column store is faster.

OPTIMIZATIONS

- Compression
- Shared Scan
- Zone Map
- Sorting
- Indexing

COMPRESSION

Original data

8 bytes width
value1
value2
value3
value4
value5
...

Compressed

3 bits width
001
010
011
000
001
100
010
011
101
...

8 bytes width
value1
value2
value3
value4
value5
...

How many bits?

SHARED SCANS

loop fusion

```
for(i=0;i<n;i++)
    min = a[i]<min ? a[i] : min
for(i=0;i<n;i++)
    max = a[i]>max ? a[i] : max
```

Two passes of data

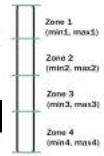
One pass of data

ZONE MAP

A common way to help column scan is the zone map.

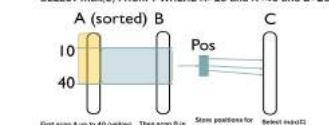
It separates a column into "zones", each is computed with max and min

In filtering, some zones can be skipped.



ENHANCED WITH SORTING

SELECT max(C) FROM T WHERE A > 10 and A < 40 and B > 20



ENHANCED WITH INDEX

- Index access is much faster than data access, because index is stored in Main Memory, while data is stored in disk
- So it is beneficial to have complicated index access to locate the exact pages where stores the data.



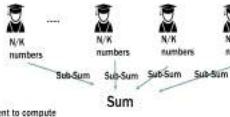
DISTRIBUTED SYSTEMS AND MAP-REDUCE

LEARNING OUTCOMES

- Understand the concept of distributed computation
- Understand the MapReduce computation model
- Can design proper algorithms based on MapReduce computation model
- Have a basic concept of Hadoop system

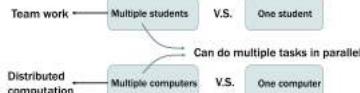
Give N integers to a team of K students to compute the sum. The numbers are given to the team leader. If you are the leader, how would you accomplish the computation task as soon as possible?

Strategy 1: Divide N numbers to K students to compute, and aggregate the results



Strategy 2: Ask one student to compute

Is there a better strategy?



If the task is to sum up 10000 integers, more people will be better.

Distributed computation is a very important approach in big data applications!

Basics of Distributed Computation

Distributed computation is often discussed in the context of big data

Not small data

DISTRIBUTED SYSTEMS

- Suppose an engineer from Amazon needs to process huge data of daily purchases.
- Just use a single commodity computer?
 - Lack of storage
 - Lack of computation power

Use multiple machines ~ Distributed system

WHAT IS DISTRIBUTED SYSTEM

A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another from any system. The components interact with one another in order to achieve a common goal.

EXAMPLE: DATA PARTITION

Name	Age
Alex	19
Bob	22
Kyle	18
Nelson	23
Ben	19
Terry	21

Name	Age
Alex	19
Bob	22
Kyle	18
Nelson	23
Ben	19
Terry	21

- Advantages:
- No space amplification (i.e., the total data size does not change)
 - Workload can be distributed to multiple machines.

DISTRIBUTED SYSTEMS FOR BIG DATA: CHALLENGES

DISTRIBUTED SYSTEMS

- The machines are connected via high-bandwidth networks.
- Two common models:
 - Fully distributed: Any two machines can access each other
 - Master-Slaves: There exists one coordinator



FULLY DISTRIBUTED MODELS

- Each machine has an IP address
- A knows machine B's IP address: A can send messages to B
- Two machines can communicate with each other via IP address
- i.e., sending messages between machines



COMMUNICATION IS NOT FOR FREE

Latency: time spent on sending/receiving messages

- Same datacenter: around 1ms
- One continent to another: around 100ms

Bandwidth: data volume per unit time

- 3G cellular data: around 1Mbit/s
- 4G cellular data: around 100Mbit/s

MASTER-SLAVE MODEL

- Each machine has an IP address
- There is a machine called **master**, and the other machines are called **slaves**.
- Master is the coordinator, being responsible to
 - distribute tasks to the slaves, and
 - receive the results from the slaves

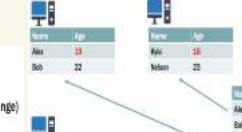


DATA PARTITION AND DATA REPLICATION

- Data partition: partition the data into different machines
- Data replication: each data item can be replicated to multiple copies.



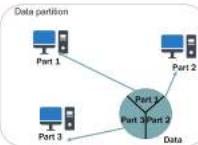
Select students whose ages < 20
(every machine does some computation)



DATA PARTITION

- Suppose the cost of doing a task on data D is $F(D)$.
- Data D is divided into $D = \{D_1, D_2, \dots, D_j\}$, which means machine j holds data D_j .
- The distributed computation cost can be estimated by $\max(F(D_1), \dots, F(D_j))$.
- Usually, we have $\max(F(D_1), \dots, F(D_j)) \ll F(D)$

DATA PARTITION



Disadvantages (if only using data partitioning)

- Machines can be off for many reasons.
- Data lost, and results are not accurate.

Only [Ben, Alex] are returned, no Kyle.



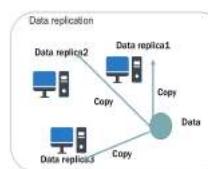
FAULT TOLERANCE

- Consider that a machine has 0.0001 probability of not being accessible at some time a day, and there are 1000 machines. What is the probability that all the machines in the cluster are always accessible within a month?
- $(1 - 0.0001)^{1000 \cdot 30}$ is about 5%.

- So, a large-scale distributed machine cluster always encounters machine failures. The process of handling machine failure is called fault tolerance.

- A **fault-tolerant** system ensures no loss of services when some machine fails.

DATA REPPLICATION



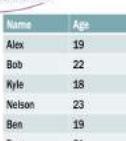
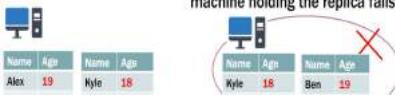
Advantages:

- Better fault tolerance
- When one machine fails, data may be restored from other replicas

- The same set of data (can be a subset of data) may be stored at different machines.



We can still get Kyle even when one machine holding the replica fails.



DATA REPPLICATION

There is no free lunch. What is the cost of using data replication?

- 1st Disadvantage: More data in each machine, and probably a high processing cost in each machine



Name	Age
Alex	19
Bob	22

Name	Age
Bon	19
Terry	21

DATA REPPLICATION: POSSIBLE SOLUTIONS

- Primary replica and secondary replicas
- During query, first access the primary replica. Only when the primary replica is not accessible, we go for one of secondary replicas.
- Maintains high query efficiency.

Name	Age
Alex	19
Bob	22

Name	Age
Kyle	18
Nelson	23

Name	Age
Kyle	18
Bon	19

Name	Age
Terry	21
Bob	22

DATA REPPLICATION

Updating one tuple requires updating all the replicas containing the tuple, causing delays (i.e., low availability).

Other options?



STRICT CONSISTENCY AND EVENTUAL CONSISTENCY

- Modern systems consider tradeoffs between consistency and availability.

Strict (replica) consistency

- When an update is committed to one replica, all the other replicas must be immediately updated as well, before handling any data reads.

- Strongly consistent between data replicas

- Low availability

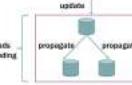
Eventual (replica) consistency

- Update is first committed to one replica, and once updated, the data reads are allowed. The updates will be ultimately propagated to other replicas.

- Weak consistency

- High availability

Strict consistency



REMARKS

- In distributed systems, there is a tradeoff between consistency and system availability

- Usually, bank or financial related systems may choose strict consistency; web applications which allow some inconsistency may choose eventual consistency.

- How to compute using multiple machines?

TYPICAL PROCEDURE OF DISTRIBUTED COMPUTATION

Master-Slave Model:

- Step 1: data is partitioned to different machines
- Step 2: master assigns tasks to each slave (or slaves request tasks from the master)
- Step 3: slaves accomplish their own tasks locally and send results to master if needed.
- Step 4: master aggregates the results and go back to Step 2 if needed.

Note: in practice, some detailed steps vary depending on the specific computation tasks.

Fully Distributed Model:

- Step 1: data is partitioned to different machines
- Step 2: each machine gathers the received messages (if any)
- Step 3: each machine conducts local computation
- Step 4: each machine sends messages to some other machines and goes to step 2 if needed.

Note: in practice, some detailed steps vary depending on the specific computation tasks.

SUMMARY OF DISTRIBUTED COMPUTATION

- Two ways of organizing machines
 - Fully distributed
 - Master-Slave

- Two data storage methods
 - Data partition
 - Data replication

- Two typical ways of distributed computation

DISTRIBUTED SYSTEMS AND MAP-REDUCE

MapReduce: A general distributed paradigm

DISTRIBUTED COMPUTATION PARADIGM

"The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues."



Jeff Dean
Lead of Google AI

PROGRAMMING WITH DISTRIBUTED MACHINES

- Programming with distributed machines is hard
- Compare with the codes to run in a single machine, coding for distributed systems requires
 - Handling communication between machines
 - Coping with machine failures
 - Data replicas
 - Data partitioning
- People start to develop ideas to ease the process:
 - OpenMPI
 - Hadoop
 - Spark
 - ...

Realization of MapReduce
One popular programming model

GOOGLE'S MAPREDUCE

MapReduce is now one of the most widely used programming paradigm in distributed processing.

Hides details such as parallelization, fault-tolerance, data distribution and load balancing

Realize the computation using map and reduce interfaces

PROGRAMMING MODEL OF MAPREDUCE

map(String key, String value)

reduce(String key, Iterator values)

more likely to be aggregated

map ($k_1, v_1 \rightarrow list(k_2, v_2)$) → Defined by you

Same k_2 will be gathered together and passed to the reduce workers (the system does it automatically).

reduce ($k_2, list(v_2) \rightarrow list(k_3, v_3)$)

EXECUTION OVERVIEW

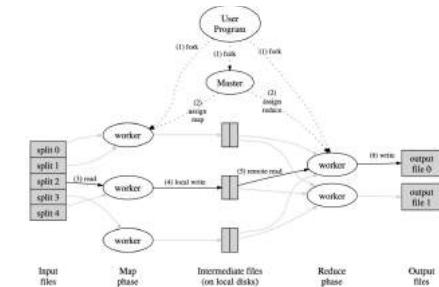
- The map invocations are distributed across machines by partitioning the input data into M splits (each split: 16-64MB).

One split → one map task (conducted in one map worker)

A set of key-value pairs. → Map function

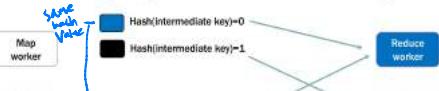
- The reduce invocations are distributed by partitioning the intermediate key space into R pieces using hash(key) mod R

- There are M map tasks and R reduce tasks.

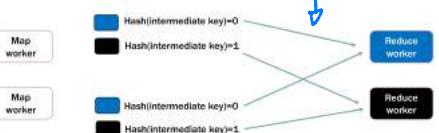


Reading: [MapReduce: Simplified Data Processing on Large Clusters](#)

- Intermediate map output is stored locally (in disk). The output is divided into R parts (each for one reduce worker to read)

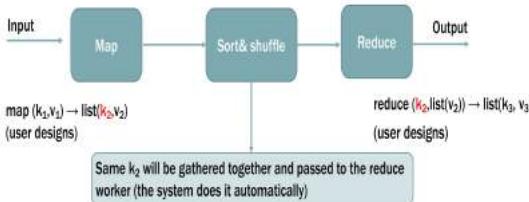


- Each reduce worker output results in local files



Remote procedure call to access data

FROM THE PERSPECTIVE OF ALGORITHMIC DESIGNER



EXAMPLE: COMPUTING FREQUENCIES

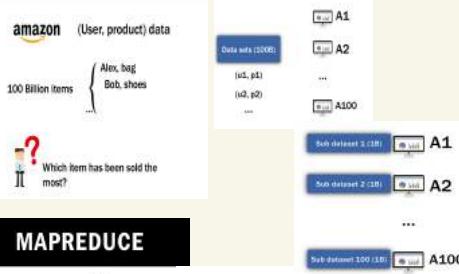
amazon (User, product) data

100 Billion items

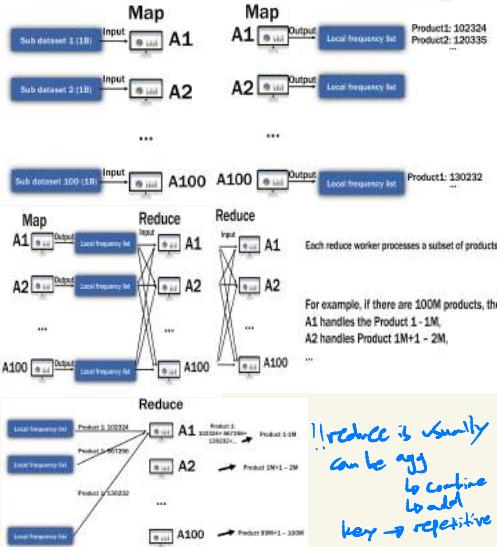
Alex, bag
Bob, shoes

Which item has been sold the most?

MAPREDUCE



MAPREDUCE



MAPREDUCE PSEUDOCODE

```
map(String key, String value) // key: user // value: product id
    Emit-Intermediate(value, "1");
}

reduce(String key, Iterator<String> values)
// key: a product
// values: a list of counts
{
    int result = 0;
    for (each v in values)
        result += ParseInt(v);
    Emit(Key,AsString(result));
}
```

Map: $\text{map}(\text{String key, String value}) \rightarrow \text{key: user, value: product id}$

Reduce: $\text{reduce}(\text{String key, Iterator<String> values}) \rightarrow \text{key: a product, value: a list of counts}$

INPUT/OUTPUT IN EACH PHASE

Input

(key, value) pairs

Map Output

(key, value) pairs

Input of Reduce

(key, value-list) pairs

Reduce Output

(key, value) pairs

Input

("Alex", "bag001") ("Alex", "bag002") ("Bob", "bag001") ("Bob", "bag003")

Map Output

("bag001", "1") ("bag002", "1") ("bag001", "1") ("bag003", "1")

Input of Reduce

("bag001", "1", "1") ("bag002", "1") ("bag003", "1")

Reduce Output

("bag001", "2") ("bag002", "1") ("bag003", "1")

ANOTHER EXAMPLE INPUT/OUTPUT

Input

("A", "001") ("A", "001") ("A", "002") ("A", "003") ("B", "004")

Map Output

("001", "1") ("001", "1") ("002", "1") ("003", "1") ("004", "1")

Input of Reduce

("001", "1", "1") ("002", "1") ("003", "1") ("004", "1")

Reduce Output

("001", "2") ("002", "1") ("003", "1") ("004", "1")

! Study pseudocode of map, reduce in C/java

EXAMPLE: INVERTED INDEX

Doc 1

NTU's 30th anniversary marks a history of sparking innovative ideas and addressing global challenges. Since its inauguration in 1991, NTU has been home to an exceptional community of faculty, students, and staff who have worked tirelessly to contribute to the university's welfare. Over our 30-year journey, we have created a digital time capsule that contains a collection of 30 past and present memories contributed by members of the NTU community. Sealed on 9 December 2022 to commemorate NTU's momentous achievements, the time capsule will be opened in 2041 at NTU's Golden Jubilee. View the artifacts at NTU's 30th anniversary exhibition, which is now open to public, or visit the time capsule website to learn more about the university's milestone moments.

EXAMPLE: INVERTED INDEX

Doc 2

The NTU School of Computer Science and Engineering (SCSE) was established in November 1988 as the School of Applied Science (SAS) offering Bachelor Degree in Computer Technology, the first of its kind in Singapore. We were part of NTU's predecessor, Nanyang Technological Institute (NTI) that was set up in August 1981 with a charter to train three-quarters of Singapore's engineers. Within 4 years of operation, NTI was singled out as one of the best engineering institutions in the world by the Commonwealth Engineering Council. The Council reached this verdict after an extensive 4-year study of the courses offered by engineering institutions worldwide.

A TYPICAL QUERY

Given a word, find out which documents contain that word.

Example:

"NTU": Doc 1, Doc 2
"August": Doc 2
"achievements": Doc 1

Extremely inefficient!

Given a word, find out which documents contain that word.

How about precompute all the document lists for each possible word in the corpus?

This is called inverted index.

→ precompute

EXAMPLE: INVERTED INDEX

Suppose we have a list of documents, each of which contains a set of words. We need to construct the inverted index, such that given a word, we can directly extract the list of documents (by IDs) containing the word. Please describe the algorithm using the MapReduce model.

Example input:
("doc1", "I study at NTU")
("doc2", "NTU is famous")

Example output:

("I", ["doc1"], ["study", ["doc1"]], ["at", ["doc1"]], ["NTU", ["doc1", "doc2"]], ["is", ["doc2"]], ["famous", ["doc2"]])

DESIGN: MAP FUNCTION

Input

("Doc1", "Big data management is a course.")

("Doc2", "Introduction to databases is a course.")

Map Output

("Big", {"Doc1"} ("data", "Doc1") ("management", "Doc1") ("is", "Doc1") ("course", "Doc1") ("Introduction", "Doc2") ("to", "Doc2") ("databases", "Doc2") ("is", "Doc2") ("course", "Doc2"))

Input of Reduce

("Big", {"Doc1"} ("data", "Doc1") ("management", "Doc1") ("is", "Doc1", "Doc2") ("course", "Doc1", "Doc2"))

Reduce Output

("Big", {"Doc1"} ("data", "Doc1") ("management", "Doc1") ("is", "Doc1", "Doc2") ("course", "Doc1", "Doc2"))

Looks like most of the tasks can be simply finished using one MapReduce job.

There are more complicated cases where we need multiple jobs

DESIGN: REDUCE FUNCTION

Key: document ID (string)

Value: document content (string)

map(String docID, String docContent):

{
 Iterator<String> words = split-to-words(docs);
 for(string word in words){
 Emit-Intermediate(word, docID);
 }
}

reduce(String word, Iterator<String> docIDs):

// key: a word // values: a list of document IDs

{
 Emit(word, ToString(docIDs));
}

DISTRIBUTED SYSTEMS AND MAP-REDUCE (MORE EXAMPLES)

MapReduce:
A general distributed paradigm

In this lecture, we will see more complicated examples of MapReduce-based algorithms.

- Table Join
- Shortest Path Computation
- PageRanks

NOTES ON PSEUDOCODE FOR MAPREDUCE ALGORITHMS

- The pseudocode focuses on "thinking in MapReduce", and so it is okay to use any understandable syntax (C-like or Java-like).
- For presenting complicated MapReduce Algorithms, we can use more complicated object class (data structure) than "String" for both key and value. Then we should describe the composition of the complicated data structure as well.

JOINING TWO TABLES

Primary key			Foreign key		
A	B	C	C	D	E
a1	b1	c1	c1	d1	e1
a2	b2	c2	c1	d2	e2
a3	b3	c3	c2	d3	e3
a4	b4	c4	c3	d4	e4
a5	b5	c5	c3	d5	e5

Map Input
Key: Line number
Value: A:B:C

Key: Line number
Value: C:D:E

If the tuples in two tables are in the same file, then we need to know the size of the 1st table to classify the tuples.

JOINING TWO TABLES

Primary key			Foreign key		
A	B	C	C	D	E
a1	b1	c1	c1	d1	e1
a2	b2	c2	c1	d2	e2
a3	b3	c3	c2	d3	e3
a4	b4	c4	c3	d4	e4
a5	b5	c5	c3	d5	e5

Map Output
Key: C
Value: T1:A:B

Key: C
Value: T2:D:E

Primary key			Foreign key		
A	B	C	C	D	E
a1	b1	c1	c1	d1	e1
a2	b2	c2	c1	d2	e2
a3	b3	c3	c2	d3	e3
a4	b4	c4	c3	d4	e4
a5	b5	c5	c3	d5	e5

Reduce Input
Key: C
Value: (T1:A:B, T2:D:E)

Key: c1
Value: (T1:a1:b1, T2:d1:e1, T2:d2:e2)
Key: c2
Value: (T1:a2:b2, T2:d2:e2)
Key: c3
Value: (T1:a3:b3, T2:d4:e4, T2:d5:e5)

Primary key			Foreign key		
A	B	C	C	D	E
a1	b1	c1	c1	d1	e1
a2	b2	c2	c1	d2	e2
a3	b3	c3	c2	d3	e3
a4	b4	c4	c3	d4	e4
a5	b5	c5	c3	d5	e5

Reduce Output
Key: c1
Value: (T1:a1:b1, T2:d1:e1, T2:d2:e2)
→ (T1:a1:b1, T2:d1:e1)
↓
(a1:b1:c1:d1:e1)
(a1:b1:c1:d2:e2)

A FEW TIPS

- It is crucial to determine which attributes/features should be aggregated on → Intermediate key
- The intermediate value can be complex; it can include a lot of useful information for you to finish the task.

WHAT ABOUT DISTANCE-BASED JOIN?

A	B	C
a5	b4	1
a2	b2	50
a3	b3	100

D	E
d1	e1
d2	e2
d3	e3

Join Column C of Table 1 and Column F of Table 2 based on the condition that |C-F| < 50. Assume that the values of Column C and Column F are positive integers.

If we directly use the values of Column C and Column F, qualified tuples will not be joined together. What should we do?



THE FIRST IDEA

Problem solved?



Aggregate the values in the same bucket

FIXED THE IDEA



bucket
(1,2) ← 98 ↗ 3
(1,2) ← 51 ↗ 47
1 ↗ 4 ← 0
2 ↗ 50 ↗ 50

Any two values whose difference is at most 50 should fall into at least one buckets

INPUT

Assume the input key-value pairs are the following (corresponding to the example):

1 a1:b1:1

2 a2:b2:50

3 a3:b3:100

4 d1:c1:1

5 d2:c2:2

6 d3:c3:3

JOB1: MAP (PSEUDOCODE)

```
Map(int key, String value){  
    if(key<table1.size){  
        int value_C=get_value_C(value);  
        Emit-Intermediate(floor(value_C/50), 'T1'+value);  
        Emit-Intermediate(floor(value_C/50)+1, 'T1'+value);  
    }  
    else{  
        int value_F=get_value_F(value);  
        Emit-Intermediate(floor(value_F/50), 'T2'+value);  
        Emit-Intermediate(floor(value_F/50)+1, 'T2'+value);  
    }  
}
```

JOB1: REDUCE (PSEUDOCODE)

```
Reduce(int key, Iterator<String> values){  
    List<value_C>list,value_T1_list,value_T2_list;  
    for(String value : values){  
        if(value starts with "T1"){  
            int value_C=get_value_C(value);  
            value_C_list.add(value_C);  
            value_T1_list.add(get_tuple(value));  
        }  
        else{  
            int value_F=get_value_F(value);  
            value_F_list.add(value_F);  
            value_T2_list.add(get_tuple(value));  
        }  
    }  
    for(int i=0;i<value_C_list.size();i++){  
        for(int j=0;j<value_F_list.size();j++){  
            int value_C=value_C_list[i];  
            int value_F=value_F_list[j];  
            if(|value_C-value_F|<50){  
                Emit("", value_T1_list[i]+value_T2_list[j]);  
            }  
        }  
    }  
}
```

JOB2: REMOVE DUPLICATES

The output of Job1 may contain duplicates (why?)

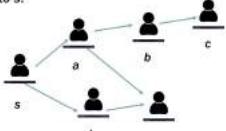
It is easy to remove duplicates by another MapReduce job.

EXAMPLE: SHORTEST PATH COMPUTATION

Given a directed social network in the following form, find all the people within k-hops from a given source node s, and the corresponding hop distance to s.

Example results (k=2):

- a,d,e,b
- dis(s,a)=1
- dis(s,d)=1
- dis(s,e)=2
- dis(s,b)=2



SINGLE-MACHINE ALGORITHM: Dijkstra's algorithm

- An efficient implementation of Dijkstra's algorithm often uses priorityqueue

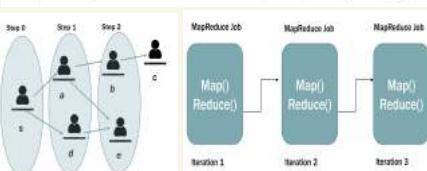
Not easy to parallelize the computation in MapReduce

IDEA FOR DESIGNING MAPREDUCE FUNCTIONS

- Solution to the problem can be defined inductively

Intuition

- Initial step (Step 0): $\text{dis}(s) = 0$, $\text{dis}(v) = +\infty$ for any other v.
- Step 1: For any node v_1 that is 1-hop from s, set $\text{dis}(v_1) = \min(\text{dis}(v_1), 1)$
- Step 2: For any node v_2 that can be reached in 2-hops from s, set $\text{dis}(v_2) = \min(\text{dis}(v_2), 2)$
- ...
- Step K: For any node v_K that can be reached K-hops from s, set $\text{dis}(v_K) = \min(\text{dis}(v_K), K)$



INPUT KEY-VALUE PAIRS

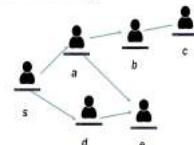
Data representation:

Each key-value pair stores the information of an edge

- Key: nodeID
- Value: nodeID

Example:

- * (s, a)
- * (s, d)
- * (a, e)
- * (d, c)
- * (a, b)
- * (b, c)



Each MapReduce iteration advances the "frontier" by one hop

- * Subsequent iterations include more and more reachable nodes as frontier expands
- * Multiple iterations are needed to explore the nodes reachable in K hops

How to preserve the graph structure?

- * Solution: map function emits (node, neighbor list) as well

FIRST MAPREDUCE JOB

Purpose:

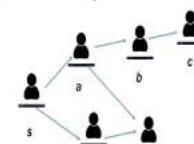
Each key-value pair stores the information of an edge

- * Key: nodeID

- * Value: nodeID

Example:

- * (s, a)
- * (s, d)
- * (a, e)
- * (d, c)
- * (a, b)
- * (b, c)



JOB0(STEP0):MAP

```
map(String nodeid_from, String nodeid_to) {
    Emit-Intermediate(nodeid_from, nodeid_to);
}
```

each node gather all neighbors

JOB0(STEP0):REDUCE

```
reduce(String nodeid, Iterator<String> neighbors) {
    if(neighbors.equals("s")){
        Emit("s", "D:0"); // "D" indicates distance
    }
}
```

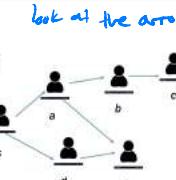
```
Emit(nodeid, "N:"+ToString(neighbors)); // "N" indicates
neighbors
```

Note: ToString() makes the list of neighbors as a string

EXAMPLE FOR JOB0

Job0:

- Input
 - * (s, a) (s, d) (a, e) (d, e) (a, b) (b, c)
- Output
 - * (s, "D:0")
 - * (s, "N:a;d")
 - * (a, "N:b;e")
 - * (d, "N:c")
 - * (b, "N;c")
 - * (a, "D:1")
 - * (d, "D:1")
 - * (s, "D:2")
 - * (a, "D:2")
 - * (b, "D:2")
 - * (c, "D:2")



SUBSEQUENT JOBS (STEPS 1-K): MAP

```
map(String nodeid, String value) {
{
    Emit-Intermediate(nodeid, value);
}}
```

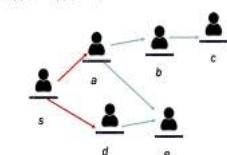
SUBSEQUENT JOBS (STEPS 1-K): REDUCE

```
reduce(String nodeid, Iterator<String> values) {
    d_min+=+∞;
    Iterator<String> neighbors;
    for(value in values){
        if(value starts with "N"){
            neighbors= ToNeighbors(value);
            Emit(nodeid, value); // always send neighbors out
        }
        else{
            if(ToInteger(value)< d_min)
                d_min=ToInteger(value);
        }
    }
    if(d_min!=+∞){
        for(neighbor in neighbors){
            Emit(neighbor, ToString("D:", d_min+1));
            // possible distances for neighbors
        }
        Emit(nodeid, ToString("D:", d_min));
    }
}
```

EXAMPLE FOR JOB1

Job1:

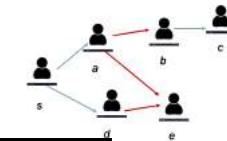
- Input
 - * (s, "D:0") (s, "N:a;d") (a, "N;b;e") (d, "N:c") (b, "N;c")
- Output
 - * (s, "D:0")
 - * (s, "N:a;d")
 - * (a, "N;b;e")
 - * (d, "N:c")
 - * (b, "N;c")
 - * (a, "D:1") From s
 - * (d, "D:1") From s
 - * (b, "D:2")



EXAMPLE FOR JOB2

Job2:

- Input
 - * (s, "D:0") (s, "N:a;d") (a, "N;b;e") (d, "N:c") (b, "N;c") (a, "D:1") (b, "D:2")
- Output
 - * (s, "D:0")
 - * (s, "N:a;d")
 - * (a, "N;b;e")
 - * (d, "N:c")
 - * (b, "N;c")
 - * (a, "D:1")
 - * (d, "D:1")
 - * (s, "D:2") From a
 - * (s, "D:2") From d
 - * (b, "D:2")



LAST JOB: MAP

```
map(String nodeid, String value) {
{
    Emit-Intermediate(nodeid, value);
}}
```

LAST JOB: REDUCE

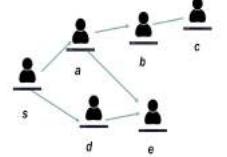
```
reduce(String nodeid, Iterator<String> values) {
    d_min=+∞;
    for value in values {
        if (value starts with "D"){
            if(ToInteger(value)<d_min)
                d_min=ToInteger(value);
        }
    }
    if(d_min!=+∞)
        Emit(nodeid, ToString("D:",d_min));
}
```

EXAMPLE LAST JOB

Job2:

Input

- Input
 - * (s, "D:0") (s, "N:a;d") (a, "N;b;e") (d, "N:c") (b, "N;c") (a, "D:1") (b, "D:2") (e, "D:2")
- Output
 - * (s, "D:0")
 - (a, "D:1")
 - (d, "D:1")
 - (b, "D:2")
 - (e, "D:2")



OPTIMIZATION?

FURTHER EXTENSION

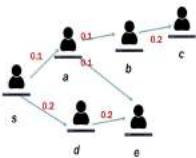
Previously, we considered the K-hop shortest path computation in social graphs using MapReduce. Extend the pseudo code to handle weighted graph whose edges may have different weights.

FIRST MAPREDUCE JOB

Purpose:

Each key-value pair stores the information of an edge

- Key: nodeID
- Value: nodeID; weight
- Example:
 - (s, "a: 0.1")
 - (s, "d: 0.2")
 - (a, "e: 0.1")
 - (d, "e: 0.2")
 - (a, "b: 0.1")
 - (b, "c: 0.2")



JOB0(STEP0):MAP

```
map(String nodeid_from, String nodeid_to_and_weight) {
    [
        Emit-Intermediate(nodeid_from, nodeid_to_and_weight);
    ]
}
```

JOB0(STEP0):REDUCE

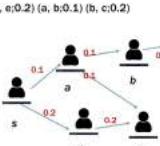
```
reduce(String nodeid, Iterator<String> neighbors) {
    if(nodeid.equals("s")){
        Emit("s", "D://" + "D" indicates distance
    }
    Emit(nodeid, "N:" + ToString(neighbors)); // "N" indicates neighbors
}
```

Note: ToString[] makes the list of neighbors as a string

EXAMPLE FOR JOB0

Job0:

Input
 • (s, a:0.1) (s, d:0.2) (a, e:0.1) (d, e:0.2) (a, b:0.1) (b, c:0.2)
 Output
 • ("s", "D")
 • ("s", "N:a:0.1;d:0.2")
 • ("a", "N:b:0.1;c:0.1")
 • ("d", "N:e:0.2")
 • ("b", "N:c:0.2")



SUBSEQUENT JOBS (STEPS 1-K): MAP

```
map(String nodeid, String value) {
    [
        Emit-Intermediate(nodeid, value);
    ]
}
```

SUBSEQUENT JOBS (STEPS 1-K): REDUCE

```
reduce(String nodeid, Iterator<String> values) {
    if(d_min==+∞)
        for(int u=0;u<neighbors.size();u++)
            Emit(neighbors[u], ToString("D:" + d_min + weights[u]));
    else
        Emit(nodeid, ToString("D:" + d_min));
}
for value in values {
    if(value starts with "N")
        neighbors=ToNeighborsNodes(value);
        weights=ToNeighborsWeights(value);
        Emit(nodeid, value);
    else
        If(GetDoubleValue(value)<d_min)
            d_min=GetDoubleValue(value); // GetDoubleValue removes 'N' and convert the value to double.
}
```

LAST JOB: MAP

```
map(String nodeid, String value) {
    [
        Emit-Intermediate(nodeid, value);
    ]
}
```

LAST JOB: REDUCE

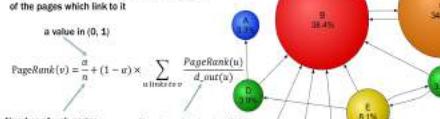
```
reduce(String nodeid, Iterator<String> values) {
    d_min=+∞;
    for(value in values){
        if(value starts with "D"){
            if(GetDoubleValue(value)<d_min)
                d_min=GetDoubleValue(value);
        }
    }
    if(d_min!=+∞)
        Emit(nodeid, ToString("D:" + d_min));
}
```

EXAMPLE: PAGERANK

- Google's famous algorithms for ranking webpages.
- A measure of the "importance/quality" of a page.
- Widely adopted for ranking search results
- Intuition
 - Consider a random surfer that starts at a random webpage
 - He follows out-going links in a random manner with probability $(1 - \alpha)$
 - He jumps to a random webpage with probability α
 - PageRank is the probability that the random surfer will arrive at a given webpage

MORE FORMALLY

PageRank of a page is based on the PageRank of the pages which link to it



PAGERANK COMPUTATION

Step 1: Initialize $\text{PageRank}(v)=1/n$ for every webpage.

Step 2: Iteratively apply the formula until convergence

For each webpage v , compute

$$\text{PageRank}(v) = \frac{\alpha}{n} + (1 - \alpha) \times \sum_{u \text{ links to } v} \frac{\text{PageRank}(u)}{d_{\text{out}}(u)}$$

MAPREDUCE DESIGNS

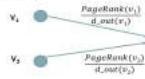
MAPREDUCE DESIGNS

Each iteration is a MapReduce job:

Map:

Input with the current PageRank value of a webpage v (or set to $\text{PageRank}(v)=1/n$ in the 1st job).

For each out-neighbor link (u) of current webpage v (namely, $v \rightarrow u$), compute $\frac{\text{PageRank}(v)}{d_{\text{out}}(v)}$ and Emit-Intermediate($u, \frac{\text{PageRank}(v)}{d_{\text{out}}(v)}$)



Reduce:

Each webpage v receives the PageRank portion (stored in values of $\text{reduce}()$) from its incoming links, and hence it can compute

$$\sum_{u \text{ links to } v} \frac{\text{PageRank}(u)}{d_{\text{out}}(u)}$$

Use the formula to update PageRanks for the next job

MAP

```
Map(String nodeid, String value_and_neighbors){
```

```
// value stores neighbor list and PageRank of nodeid
double PR=getPageRank(value_and_neighbors);
List<String> neighbors = getNeighbors(value_and_neighbors);
for(each neighbor in neighbors){
    Emit-Intermediate(neighbor, "P:" + PR / neighbors.size());
}
Emit-Intermediate(nodeid, "N:" + ToString(neighbors));
```

REDUCE

```
Reduce(String nodeid, Iterator<String> values){
    double PR=0.0;
    String neighbors;
    for(value in values){
        if(value starts with "P"){
            PR = PR + getPageRank(value) * (1 - α);
        }
        else{
            neighbors = getNeighbors(value);
        }
    }
    PR=PR +  $\frac{\alpha}{n}$ ;
    Emit(nodeid, PR + " + neighbors);
```

SUMMARY AND OPEN DISCUSSIONS

MapReduce model is very powerful.

- Database operations (join)
- Graph based queries (shortest paths, PageRanks)
- ...

1. Any more examples?
2. Possible optimization

Combiner



HADOOP → SPARK

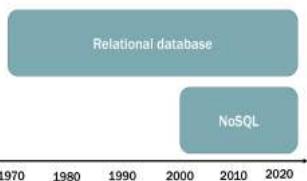


$$\text{PageRank}(v) = \frac{\alpha}{n} + (1 - \alpha) \times \sum_{u \text{ links to } v} \frac{\text{PageRank}(u)}{d_{\text{out}}(u)}$$

NoSQL

BASIC CONCEPT

You may have heard about the word "NoSQL"



RELATIONAL DATABASES (RECAP)

Query patterns	• Selection, Projection, Join, Aggregation
Query languages	• SQL (Structured Query Language) • Relational algebra
Systems	• Oracle Database, Microsoft SQL Server, IBM DB2, MySQL, PostgreSQL

RELATIONAL DATABASES (RECAP)



Normal Forms

Functional dependencies:
1NF, 2NF, 3NF, BCNF

Good: Reduces data redundancy, prevent update anomalies.
Bad: Data is divided into small pieces and so queries involve joining them (costly).

CURRENT TRENDS

Big data	• Volume, Variety, Velocity, ... • Various data formats • Strong consistency is no longer mission critical
Extensive user base	• Population online, hours spent online, devices online • Growing companies / web applications • Even millions of users within a few months
Cloud computing	• On-demand services of data storage and computing power
Real-time analytic processing	• Quality of services becomes more and more important.

RELATIONAL AND NOSQL DATABASES

- ❑ NoSQL also means "Not only SQL", where SQL refers to the relational database (not exactly the SQL language).
- ❑ A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.
- ❑ NoSQL databases are increasingly used in big data applications.



PROPERTIES OF NOSQL DATABASE

Well designed schema

e.g., Relations are well decomposed and connected by foreign keys

Benefit: standardized data model designs (normal forms)

Not that easy to scale

Reason: usually relational database requires strict consistency



Flexible schema (schemaless)

e.g., documents, key-value pairs

Benefit: programmers are more flexible in designing data models.

WIDE-COLUMN DATABASE

high availability
high write

Easier to scale

Reason: usually requires eventual consistency



Support all kinds of relational queries

Better supports query languages

e.g., SQL

Relational database

Queries are less flexible, but can have higher performance

Partially supports query language

Lack of standardized interface

NoSQL

DOCUMENT DATABASE

GRAPH DATABASE

TYPES OF NOSQL DATABASES

KEY-VALUE STORE

Key-Value Stores

Wide-Column Database

Document Database

Graph Database

Data model: 2-dimensional key-value models.

The names and format of the columns can vary across rows, even within the same table.

Columns are not separately stored, but some of them can be grouped as a column family. All values in a column family, it can be stored row-by-row.

It can have very large number of columns

Representatives:

- Google Bigtable, Apache Cassandra, Apache HBase, Apache Accumulo, HyperTable

❑ Data model: data are stored as documents

- A document describes an object
- Can be described in JSON format (i.e., a list of key-value pairs to describe the object attributes)
- Example: ["FirstName": "Bob", "Address": "5 Oak St.", "Hobby": "sailing"]
- A document is addressed by a "key", so also regarded as a subclass of key-value store. Differences is that the value is a document which can contain very rich information

❑ Representatives:

- MongoDB, Couchbase, Amazon DynamoDB, CouchDB, Redis, RavenDB, Terrascale

Data model: graphs

- A graph consists of nodes and edges
- Nodes represent entities; edges for relationships
- Easier to model data that contain many entities and interconnected relationships, e.g., social networks

Query patterns

- Create, update or remove a node / relationship in a graph
- Graph algorithms (shortest paths, spanning trees, ...)
- General graph traversals
- Sub-graph queries or super-graph queries
- Similarity based queries (approximate matching)

Representatives:

- Neo4j, Titan, Apache Giraph, InfiniteGraph, FlockDB

WHEN TO CHOOSE ONE OVER THE OTHER (RDB VS NOSQL)

There is NOT a golden rule that you should choose one over the other. Both systems are improving themselves, and some recent systems have integrated their advantages at a mixed system to serve needs.

How to choose one over the other?

- 1) When the data scale is small, relational database is great
- 2) When the project is always about relational queries and have various access patterns, relational database is great
- 3) When you want to better enforce data constraints, relational database is preferred

→ When the data scale is very large, and you need very high performance for some specific type of queries, NoSQL is great.



SCENARIOS

Suppose a company with 10000 employees want to build an employee management system. It would require to query the salaries of employees, the managers of an employee etc.

Would you choose a relational database or NoSQL database?

Relational database is good for this scenario.

SCENARIOS

Suppose we aim to build a stock database that records all the stock price changes every 5 seconds. The query pattern is given a stock id, search the lowest and highest stock prices of a given time period. The results should be returned in a short time when searched.

Would you choose a relational database or NoSQL database?

NoSQL key-value store is good choice for this scenario.

When we emphasize the benefit of NoSQL systems, do NOT take for granted that NoSQL databases are always better than Relational databases.

Relational databases are great in many aspects

Both relational databases and NoSQL databases are improving

Both relational databases and NoSQL databases can be chosen for different applications.

NOTE

KEY-VALUE STORE

LSM-TREE BASICS

PREPARATION

- In previous lectures, we introduced the basic concepts of NoSQL databases
- Key-Value Store/Key-Value Database
- Wide-column database
- Document database
- Graph database

- In the following lectures, we will introduce the mechanism of Key-Value Store, which is the most fundamental NoSQL database

BASIC CONCEPT

- Data model**
- Key-Value Store stores the data in key-value format
- Every key corresponds to a value
- Functions:** It supports four functions
 - Get: Given a key, search the value indexed by the key
 - Range-Get: Given a key range, search all the values indexed by any key within the range
 - Put a new key-value pair
 - Delete a key-value pair
- Data engine:** The log-structured merge tree (LSM-tree)

GET AND RANGE-GET

- Suppose a key-value store has the data $\{(1, \text{value}1), (2, \text{value}2), (3, \text{value}3)\}$
- Get(1) \rightarrow value1
- Get(2) \rightarrow value2
- Get(5) $\rightarrow ()$
- Range-Get(2,3) \rightarrow (value2, value3)
- Range-Get(3,5) \rightarrow (value3)
- Range-Get(4,5) $\rightarrow ()$

PUT AND DELETE

- Suppose a key-value store has the data $\{(1, \text{value}1), (2, \text{value}2), (3, \text{value}3)\}$, then logically
 - After Put(4,value4), we have $\{(1, \text{value}1), (2, \text{value}2), (3, \text{value}3), (4, \text{value}4)\}$ in the key-value store
 - After Delete(1), we have $\{(2, \text{value}2), (3, \text{value}3), (4, \text{value}4)\}$ in the key-value store
 - After Put(3,value5), we have $\{(2, \text{value}2), (3, \text{value}5), (4, \text{value}4)\}$ in the key-value store

THINK...

Suppose we need to store 10 billion key-value pairs in the database, what data structure you will use?

lvl 0 to 1

FIRST IDEA

An array with ten billion entries

Size: $10^{10} * 100 \text{ Bytes} = 1000 \text{ GB}$

Stored in memory? Stored in disk?

Too large to be stored in main memory

Main memory does not guarantee persistency

Disk is too slow to access, put and delete operations are costly.

SECOND IDEA



When handling Put(key,value), first insert the key-value pair into the main memory; when memory buffer is full, put all the buffer as a "sorted run" into the disk.



When handling Put(key,value), first insert the key-value pair into the main memory; when memory buffer is full, put all the buffer as a "sorted run" into the disk.

SECOND IDEA

Memory buffer

Put (key, value)

Full and sort merge

(key, value) pair

Disk

Any problems with this design?

High cost for GET function!

GET(key) consists of two steps

- Search the key in the main memory buffer; if the key exists in the buffer, directly return the value;
- Search the key in the disk. *costly*

Put(key, value) is also costly

MERGE CONDITION

3

Level 0 S*T

Level 1 S*(T+1)

Level 2 S*(T+1)

S*(T+1)

S*(T+1)

Trigger
merge
condition

THIRD IDEA: BASIC LSM-TREES

Memory buffer

Put (key, value)

Full and sort merge

T-size ratio

Two levels to multiple levels

Let $L[i]$ be the i -th level

Mem buffer capacity (i.e., $L[0]$ capacity) = S (key-value pairs)

Capacity is different from the actual size. $L[i]$ capacity may be different

from the number of actually stored (key,value) pairs in $L[i]$

Memory buffer

Put (key, value)

Full and sort merge

Disk

Disk

T times

T times

n of key-value pair

EXAMPLE

Memory buffer

Level 0

Level 1

Level 2

Memory buffer

Level 0

Level 1

Level 2

Memory buffer

Level 0

Level 1

Level 2

Memory buffer

Level 0

Level 1

Level 2

Memory buffer

Level 0

Level 1

Level 2

Memory buffer

Level 0

Level 1

Level 2

void Put(Key k, Value v) void SortMerge(Level i)

```
{
    L[0].insert(k,v);
    if(L[0].size==S){
        SortMerge(0);
    }
}
```

if (L[i+1] not exists)

create L[i+1];

L[i+1].SortMergeWith(L[i]);

L[i].clear();

if(L[i+1]>S*(T+1-T))

SortMerge(i+1);

}

The actual number of (key,value) pairs after merging $L[i]$ into $L[i+1]$ may be less than actual size($L[i]$) + actual size($L[i+1]$)

user
due to updates new value to existing key but it outdated values remain

wiring

[LSM can help control the write cost, as at most the merge sort size between $L[i]$ and $L[i+1]$ is the size of itself]

How about deleting a key?

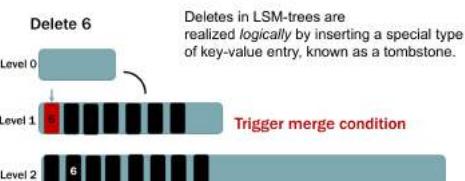
DELETE?



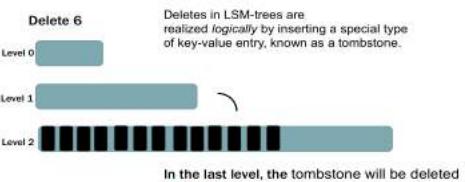
DELETE-TOMBSTONE



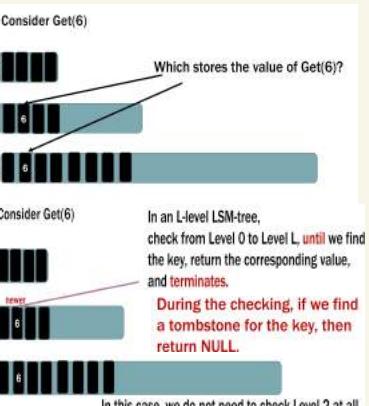
DELETE-TOMBSTONE



DELETE-TOMBSTONE



How to implement Get(K)?



Consider Get(6)

Level 0



Level 1



How to check each level on disk (starting from Level 1)?

This is a sorted run

Level 2



Consider Get(6)

Level 0



Level 1



How to check each level?

Sorted based on keys

Level 2



First idea:

Searching from left to right, and stops when we find 6

THE COST FOR GET(6)

Consider Get(6)

Level 0



How to check each level?

No I/Os (because in main memory)

Level 1



Can have many I/Os

Level 2



No I/Os

I/O cost: Number of disk page reads or writes

SOME OTHER POSSIBLE CASES (FOR OTHER KEYS)

Consider Get(6)

Level 0



How to check each level?

No I/Os (because in main memory)

Level 1



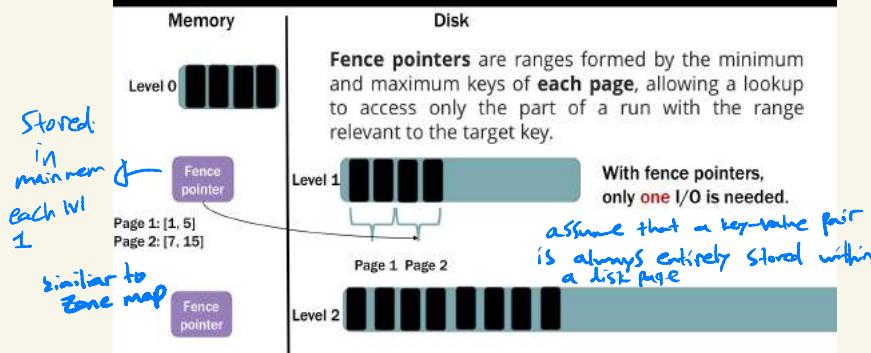
Can have many I/Os

Level 2



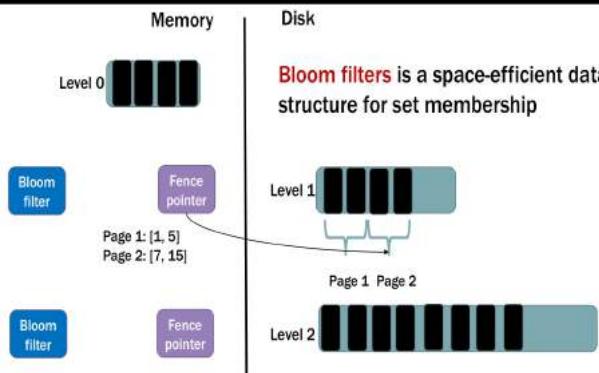
Can have many I/Os

OPTIMIZATION – FENCE POINTERS



Each level at most 1 I/O

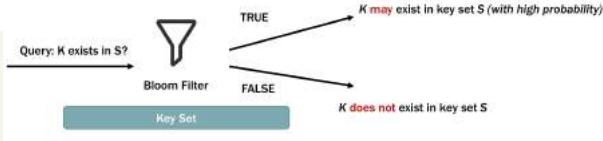
OPTIMIZATION – BLOOM FILTERS



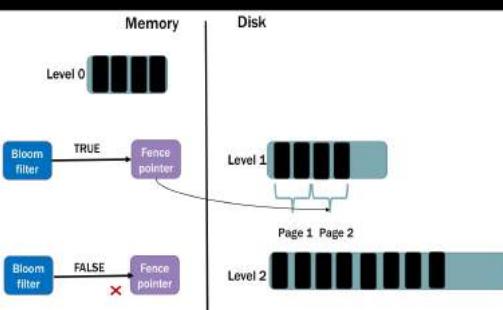
BLOOM FILTER

1. Stored in main memory
2. Built on a set S of keys
3. Given a key K , the Bloom filter answers TRUE or FALSE for key K
4. If it answers FALSE, it means the key K **does not** exist in key set S
5. If it answers TRUE, it means the key K **may** exist in key set S , and it is still possible that the key K does not exist in key set S .
6. FPR (False-Positive Rate) is the probability that the filter returns TRUE for a key K , but actually K does not exist in set S . We usually use P to denote FPR. Clearly, P is in $[0, 1]$ (e.g., $P=0.3$)

ILLUSTRATION OF BLOOM FILTER



WHY BLOOM FILTER IS HELPFUL?



SUMMARY

1. Check buffer level (Level 0)
2. Starting from Level 1, always check its Bloom filter first, and then its Fence pointer.
3. If Bloom filter returns FALSE, then do not access the fence pointer and directly go to the next level for checking.
4. If Bloom filter returns TRUE, then access the fence pointer to fetch the corresponding page in that level. If the key is found, return the value and terminate the program; otherwise go to the next level for checking.
5. If the key has not been found in any level, return NULL (or empty set) and terminate the program.

INTERNAL DESIGNS OF BLOOM FILTER

The main purpose of the filter:

- Built on a set of keys $S = \{K_1, K_2, \dots, K_n\}$, where each key is from a large universe U . *(if false)*
- Bloom filter can 100% determine if a key is NOT in S , and with HIGH probability it can tell if a key is in S .
- Space efficient: on average, each key only occupies a few bits in Bloom filter.
- Inserting a new element into the Bloom filter should be fast
- Querying a key from the Bloom filter should be fast

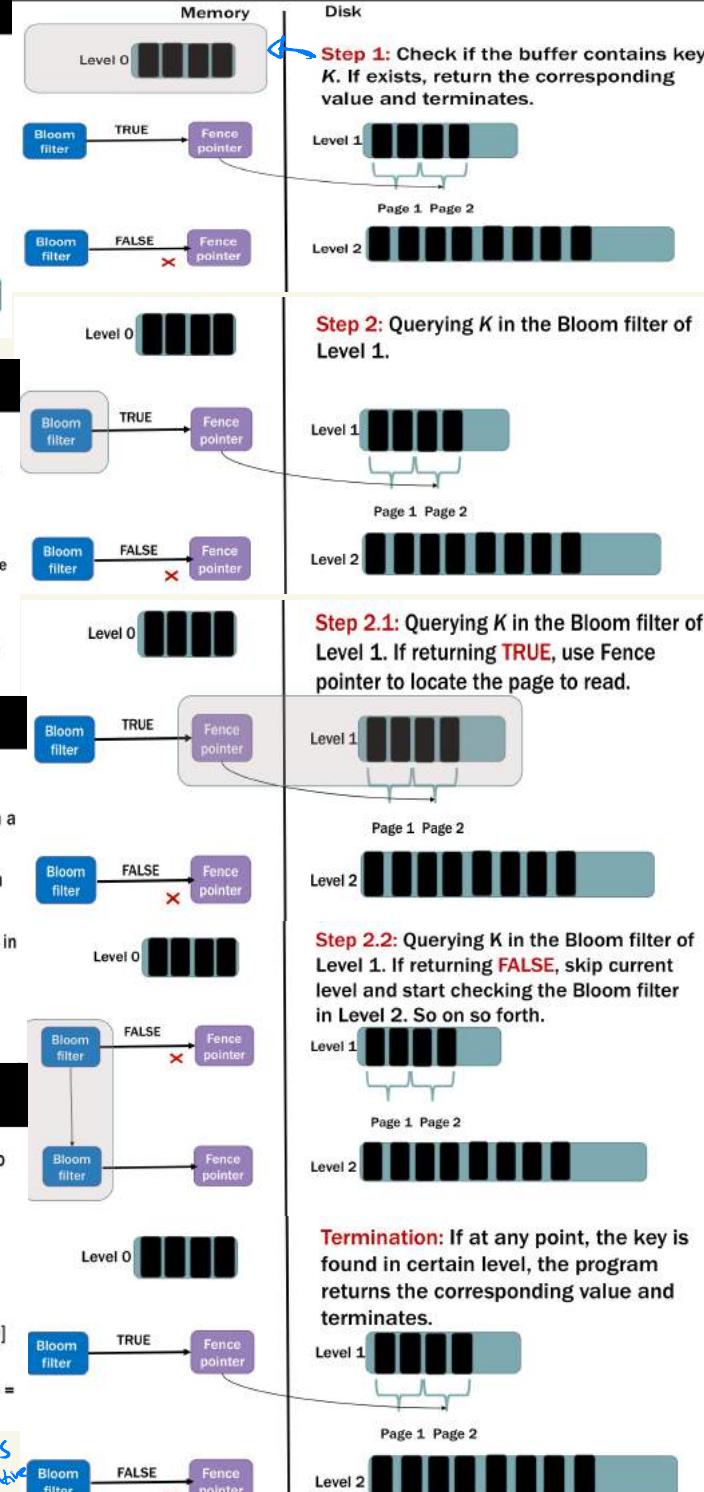
THE CORE OF BLOOM FILTER - HASH FUNCTION

A hash function h maps a uniformly at random chosen key $x \in U$ to an integer from $R_m = [0, m-1]$ such that each element in R_m is mapped with equal probability.

A Bloom filter is a bit vector B of m bits, with k independent hash functions h_1, \dots, h_k *bit we control*

- Initially all the m bits are 0.
- Insert x into S : compute $h_1(x), \dots, h_k(x)$ and set $B[h_1(x)] = B[h_2(x)] = \dots = B[h_k(x)] = 1$.
- Query if $x \in S$: Compute $h_1(x), \dots, h_k(x)$. If $B[h_1(x)] = B[h_2(x)] = \dots = B[h_k(x)] = 1$, then answer TRUE, else answer FALSE.

no way this is False negative



EXAMPLE

- $m=5, k=2$
- $h_1(x) = x \bmod 5$
- $h_2(x) = (2x+3) \bmod 5$
- Initially $B[0]=B[1]=B[2]=B[3]=B[4]=B[5]=0$
- Then insert 9 and 11

$B_0, B_1, B_2, B_3, B_4, B_5$

	$h_1(x)$	$h_2(x)$	B					
Initialize:			<table border="1"> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	0
0	0	0	0	0				
Insert 9:	4	1	<table border="1"> <tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> </table>	0	1	0	0	1
0	1	0	0	1				
Insert 11:	1	0	<table border="1"> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr> </table>	1	0	0	0	1
1	0	0	0	1				

Now query 15 and 16

	$h_1(x)$	$h_2(x)$	Answer
Query 15:	0	3	No, not in B (correct answer)
Query 16:	1	0	Yes, in B (wrong answer: false positive)

ANALYSIS

- When m/n is fixed (m/n is often called bits-per-key), the optimal k is $\ln 2 \times (m/n)$ (See [here](#) for proof if you are interested)
- Then, the optimal FPR is about $0.6185^{m/n}$
- So, larger m means small FPR (approaches to 0); smaller m means higher FPR (approaches to 1).

SUMMARY OF LSM-TREE

- Multi-level structured
- Out-of-place updates (delete acts as a special insert)
- Fence pointers \rightarrow reduce I/Os of Get()
- Bloom filters \rightarrow reduce I/Os of Get()

CLARIFICATIONS

In practice, fence pointers can be implemented in different ways.

Option 1: containing the min/max of each page;

Option 2: containing only the min (or max) of each page;

For analysis purpose, we reasonably assume that when using Fence pointers, it always incurs one I/O.

VARIANTS OF LSM-TREE

- Leveled LSM-tree (or Leveling LSM-tree)
 - The LSM-tree with leveling merge policy.
- Tiered LSM-tree (or Tiering LSM-tree)
 - The LSM-tree with tiering merge policy.
- Until now, the LSM-tree we introduce belongs to Leveling LSM-tree.
- Next, we introduce what is Tiering LSM-tree

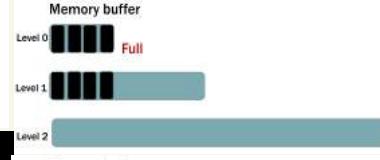
LEVELING LSM-TREE VS TIERING LSM-TREE

- read optimized* *write optimized*
- The difference lies in the way of merging a full level to its next level

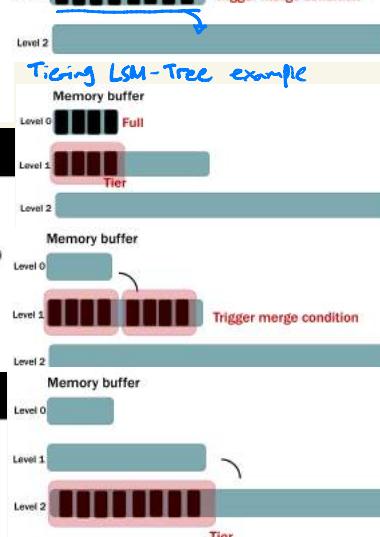
Leveling LSM-tree:

- When a level needs to be merged, always sort-merge to the next level
- Tiering LSM-tree:
 - When a level needs to be merged, does not sort but put it as a **tier** stored in the next level

LEVELING LSM-TREE EXAMPLE



Tiering LSM-TREE example



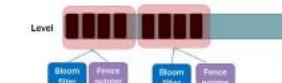
SOME PROPERTIES

- A tier in Level i roughly has the size of the capacity of Level $(i-1)$
- Usually, in tiering LSM-tree, starting from Level 1, the merge is triggered when there are T tiers in it, where T is the size ratio.

- Question: In each level of tiering LSM-tree, is a key unique?

FENCE POINTERS AND BLOOM FILTERS IN TIERING LSM-TREES

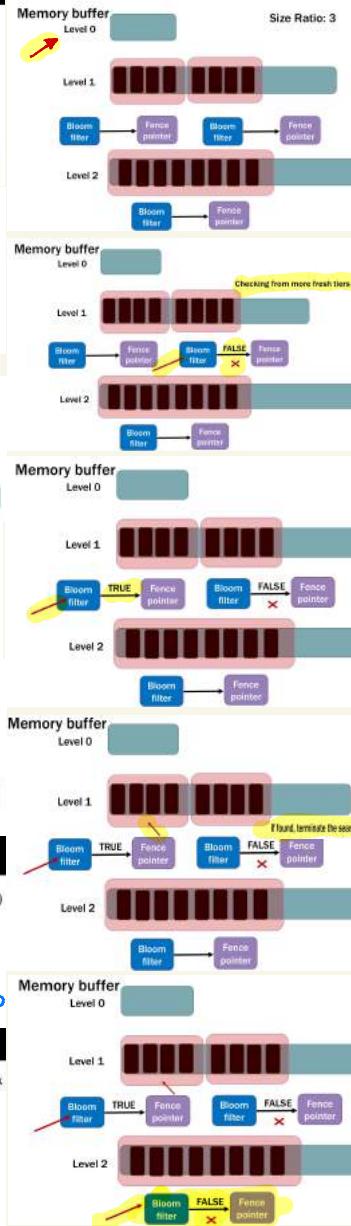
Bloom filter and fence pointers are built for each tier of each disk level (except Level 0)



PROS AND CONS OF TIERING LSM-TREE

- Advantages:
 - Avoid costly sort merges
 - Put/Delete is faster
- Disadvantages:
 - Get is slower
- Summary
 - Leveling LSM-tree: faster data reads, slower data writes
 - Tiering LSM-trees: slower data reads, faster data writes

PERFORMING GET(K) IN TIERING LSM-TREE

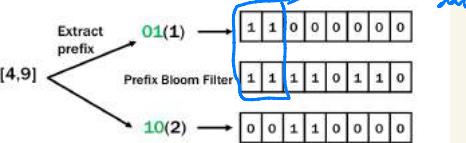


RANGE FILTER - PREFIX BLOOM FILTER

Level 2

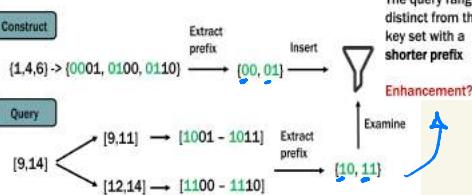
1 3 5 7 8 11 13 15

Prefix Bloom Filter 1 1 1 1 0 1 1 0



Consider:

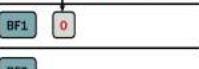
How to construct Prefix Bloom filter with the key set {1,4,6} and query with the range [9-14]?



Idea: use hierarchical prefix Bloom filters to encode different length of prefixes

Example: set prefix length from 1 to 3.

Key set: {0001, 0100, 0110}



For BF1:

0 -> [0000, 0111] Range size = 8

For clarity, the elements inserted into a BF are listed instead of the actual BF

Key set: {0001, 0100, 0110}

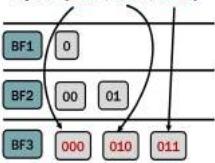


Key set: {0001, 0100, 0110}

Each element represents smaller range compared to BF1

00 -> [0000, 0011] Range size = 4
01 -> [0100, 0111]

Key set: {0001, 0100, 0110}



000 -> [0000, 0001] Range size = 2

010 -> [0100, 0101]

011 -> [0110, 0111]

Query

-> [9,14]

-> [1001, 1110]

Key set: {0001, 0100, 0110}

BF1 0

BF2 00 01

BF3 000 010 011

NO

Do not need to query the following BFs due to there is not key in the extended query range [1000,1111]

Example: set prefix length from 1 to 3.

Query

-> [9,14]

-> [1001, 1110]

[1100, 1110]

Key set: {0001, 0100, 0110}

BF1 0

BF2 00 01

BF3 000 010 011

This step is unnecessary for the reason mentioned

Query 1 bit is sufficient, more efficient.

Key set: {0001, 0100, 0110}

BF1 0 BPK=2

BF2 00 01 BPK=2

BF3 000 010 011 BPK=2

BF1 0 BPK=1 Diverse BF memory allocation

BF2 00 01 BPK=2

BF3 000 010 011 BPK=3

More flexible structure to adapt to different work condition.

Key set: {0001, 0100, 0110}

BF1 0 BPK=2

BF2 00 01 BPK=2

BF3 000 010 011 BPK=2

BF1 0 BPK=1 Diverse BF memory allocation

BF2 00 01 BPK=2

BF3 000 010 011 BPK=3

However, more BF could take up more memory space

OVERVIEW

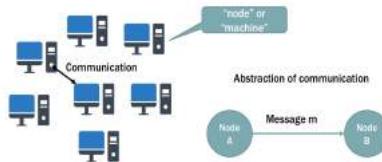
2ND HALF

DISTRIBUTED SYSTEMS FOR BIG DATA: CHALLENGES

- ❑ How to organize the machines?
 - ❑ Fully-Distributed Mode
 - ❑ Master-Slave Mode
 - ❑ Fault-Tolerant
- ❑ How to store data across machines?
 - ❑ Data Partition
 - ❑ Data Replication
- ❑ How to compute using multiple machines?

FULLY DISTRIBUTED MODELS

- ❑ Each machine has an IP address
- ❑ A knows machine B's IP address: A can send messages to B
- ❑ Two machines can communicate with each other via IP address
 - ❑ i.e., sending messages between machines



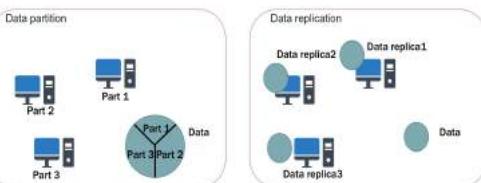
MASTER-SLAVE MODEL

- ❑ Each machine has an IP address
- ❑ There is a machine called **master**, and the other machines are called **slaves**.
- ❑ Master is the coordinator, being responsible to
 - ❑ distribute tasks to the slaves, and
 - ❑ receive the results from the slaves



DATA PARTITION AND DATA REPLICATION

- ❑ Data partition: partition the data into different machines
- ❑ Data replication: each data item can be replicated to multiple copies.



MAPREDUCE

- ❑ Understand the basic model of MapReduce
 - ❑ Map function
 - ❑ Reduce function
 - ❑ Job
- ❑ Understand the execution workflow of MapReduce
 - ❑ Within a job, reduce function receives the pairs with the same intermediate key
- ❑ Know how to design algorithms (pseudo-code)
 - ❑ Wordcount
 - ❑ Table Join
 - ❑ Shortest distance
 - ❑ PageRank

NOSQL

- ❑ Property of NoSQL
 - ❑ Flexible schema (schemaless)
 - ❑ Easier to scale
 - ❑ Partially supports query language
 - ❑ Queries are less flexible, but can have higher performance
- ❑ Types of NoSQL Systems
 - ❑ Key-Value Stores
 - ❑ Wide-Column Database
 - ❑ Document Database
 - ❑ Graph Database

KEY-VALUE STORES

- ❑ LSM-tree
 - ❑ Get
 - ❑ Put
 - ❑ Delete
 - ❑ Fence Pointers
 - ❑ Bloom filters
 - ❑ FPR
 - ❑ I/O cost analysis
 - ❑ Tiering LSM-tree
 - ❑ Range Filter (Not in the scope of final exam)

FINAL EXAM TIME AND VENUE

- ❑ May 8 1pm-3pm (Come Early!)
- ❑ Hall 7

Hall 7	Function Hall (former Meranli Hall)
--------	-------------------------------------

- ❑ Closed-Book
- ❑ Covers whole semester lectures (including those before quiz)
- ❑ Instructions to Examination Candidates
https://entuedu.sharepoint.com/sites/Student/dept/sasd/oas/Shared%20Documents/ExamAndAssessment/Exam/Instructions_to_candidates_physical_examinations_on-campus.pdf

BIG DATA MANAGEMENT

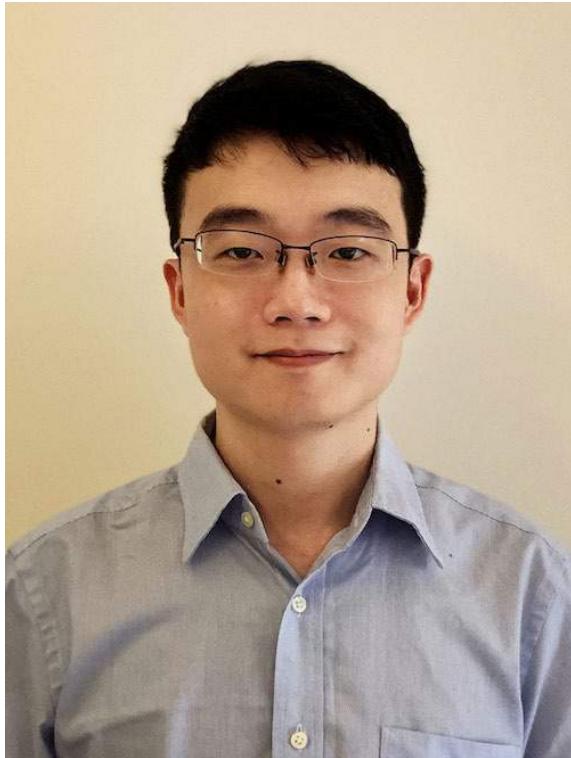
CZ/CE4123

Course Overview



COURSE INSTRUCTOR

LUO Siqiang (Assistant Professor at SCSE)



Email
siqiang.luo@ntu.edu.sg

Research area
Big data / data management

Office
N4 Level 2 c-110

Teaching Assistant to help the course project

Wang Fan

FAN008@e.ntu.edu.sg



For **all the lecture related questions**, please directly email me at siqiang.luo@ntu.edu.sg

For **project related problems**, you can consult TAs for specifics.

WHAT IS BIG DATA MANAGEMENT?



WHAT IS THIS COURSE ABOUT?



Understand
important concepts
of big data



Analyze important
big data processing
techniques



Explain various big
data systems

AFTER THIS COURSE, YOU WILL BE ABLE TO...

- Understand what is big data

Solving math?

AFTER THIS COURSE, YOU WILL BE ABLE TO...

- Understand what is big data

Solving math?
No!

AFTER THIS COURSE, YOU WILL BE ABLE TO...

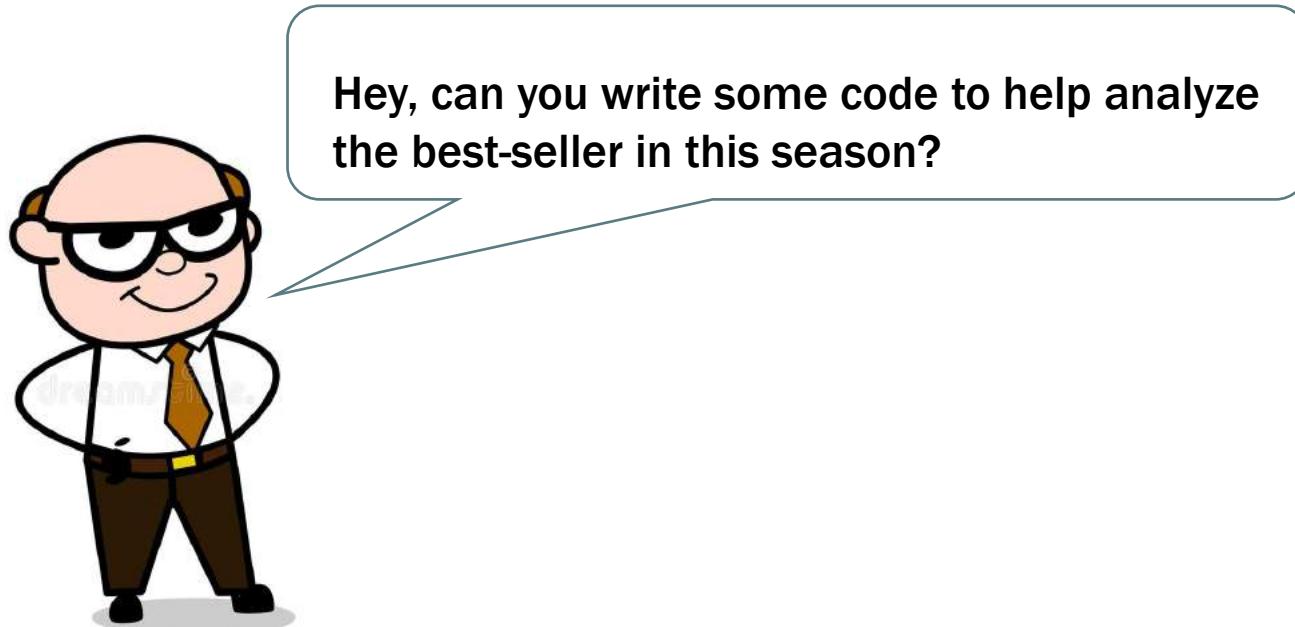
- Understand what is big data

Big data 5V's

AFTER THIS COURSE, YOU WILL BE ABLE TO...

- Consider a scenario:

- You are a big data engineer/ analyst in Amazon, your boss assigns you a task:





Sure! I can scan each record and get the selling frequency of each product, and then get the item with the max frequency!

AFTER THIS COURSE, YOU WILL BE ABLE TO...



Umm ... Not bad, but forgot to tell you.
We have **1 trillion** sale-item records...
Will your method be efficient?

AFTER THIS COURSE, YOU WILL BE ABLE TO...

- You will learn some solutions from the course

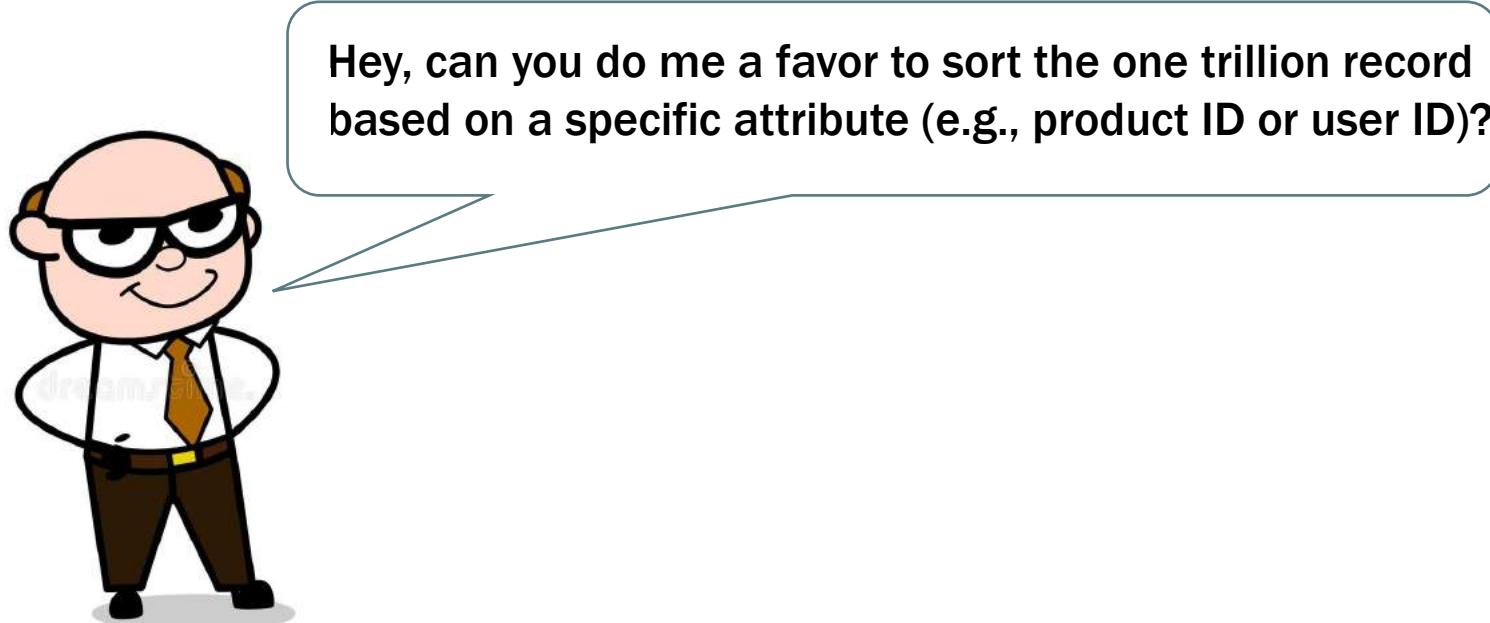


MapReduce!

- Use a few lines of code to efficiently analyze the “best seller” in retail applications in a distributed system!

AFTER THIS COURSE, YOU WILL BE ABLE TO...

- Consider another scenario:
 - The other day, you receive an urgent task from your boss:





Easy! I get quite familiar with different kinds of sorting algorithms such as quick-sort, heap-sort, ...

AFTER THIS COURSE, YOU WILL BE ABLE TO...



Umm... I am afraid we do not have a machine that can hold all the data in main memory...

AFTER THIS COURSE, YOU WILL BE ABLE TO...

- After the course, you will probably have some solutions

External Sorting

AFTER THIS COURSE, YOU WILL BE ABLE TO...

- Next scenario:

- 2 years later, you are promoted to a tech leader. Your boss wants you to design a system for easy queries of product item information.

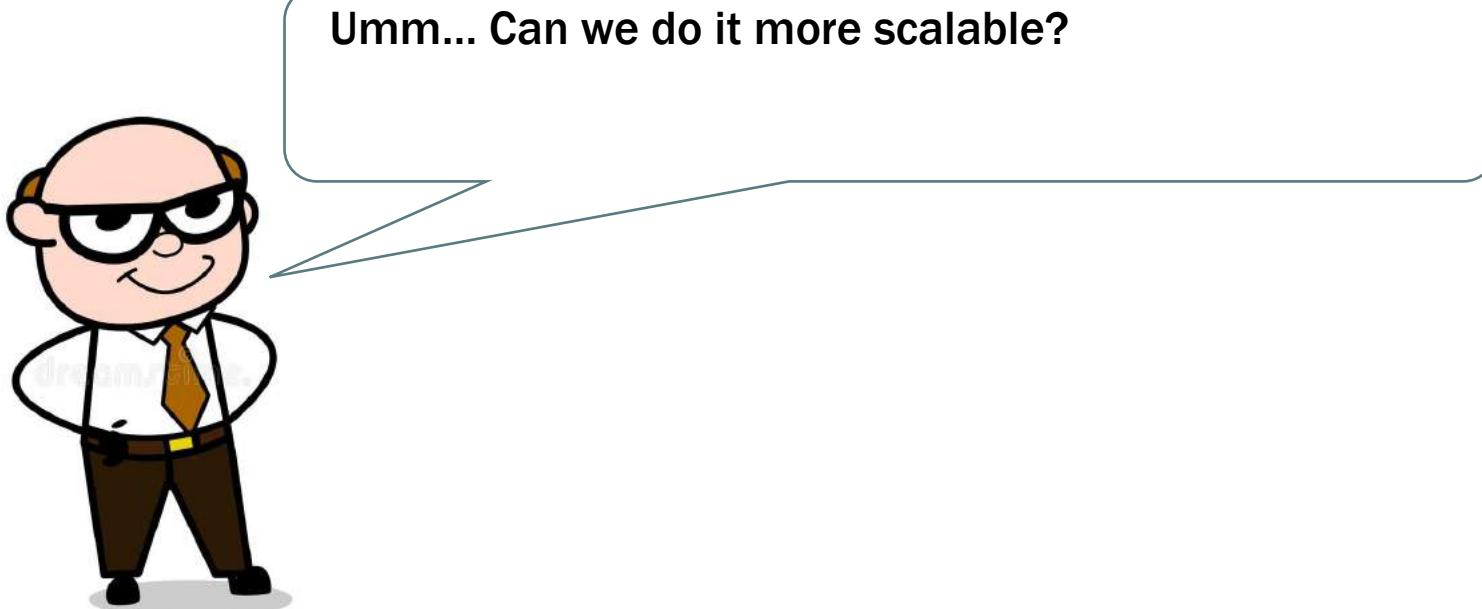


Can you design a system to hold one trillion records, so that user can easily query the information of any given product?



Yeah! I get familiar with SQLServer, ...

AFTER THIS COURSE, YOU WILL BE ABLE TO...



AFTER THIS COURSE, YOU WILL BE ABLE TO...

- After the course, you will probably have alternative solutions

NoSQL Key-Value Stores

AFTER THIS COURSE, YOU WILL BE ABLE TO...

- Next scenario:
 - 2 years later, your boss wants you to redesign the system.



Hey, can you redesign the system to hold the product-purchase data, so that user can easily **query and filter** the product information based on ID and name?



**Well, this time maybe SQLServer is a
better solution?**

AFTER THIS COURSE, YOU WILL BE ABLE TO...



I do not disagree. But can we do even better given that the product may have hundreds of properties?

AFTER THIS COURSE, YOU WILL BE ABLE TO...

- After the course, you will probably have alternative solutions

Column Store

PREREQUISITES

Course CZ2007:
Introduction to Databases

~~-CE/CZ4031:
Database system principles~~

THIS COURSE IS

- NOT a programming language course
 - ✓ Will not teach SQL (had learnt it in CZ2007)
 - ✓ Will not teach C or Java (may have learnt it in other courses)
 - ✓ Though we assume you understand **one of** basic SQL or C or Java
- NOT a traditional database course
 - ✓ Will not teach relational database (may recap if necessary)

BIG DATA MANAGEMENT – COURSE OVERVIEW

We will discuss interesting big data techniques!



**Big Data
5V's**

**Memory
Hierarchy**

**Column
Store**

**Distributed
MapReduc
e Systems**

**NoSQL
Key-Value
store**

BIG DATA MANAGEMENT – COURSE OVERVIEW

Most of the techniques are cutting-edge techniques in big data!

No text books – Slides contain everything



Big Data
5V's

Memory
Hierarchy

Column
Store

Distributed
MapReduc
e Systems

NoSQL
Key-Value
store

LECTURE STYLE

- The purpose of the course is to expand your vision, both conceptually and technically.
- I tend to encourage questions and (open-ended) discussions during the class.
- I tend to link the techniques to some real industrial systems.

TUTORIAL STYLE

In the tutorial of this course, you will be

- Taught with solving some theory questions related to big data techniques;

- Hands-on tutorials to walk you through some widely adopted big-data systems, such as Hadoop (used by many companies) and RocksDB (used by Facebook/Meta). You will not be examined on the procedure; instead, our purpose is to give you some resources which might be useful when you join the industry in the future.

COURSE SCHEDULE

Week 1 – Week 13: Course lectures

Venue: LT27

Time: 15:30pm – 17:20pm on Tuesday

Week 3 – Week 13: Tutorials

Venue: LT27

Time: 11:30am – 12:20pm on Monday

Quiz: Week 10 tutorial

EVALUATION (TENTATIVE)

1 Quiz:

25%

1 Group Project:

25%

Final:

50%

QUESTIONS WANTED!!

I welcome all kinds of questions during the course!!

Questions wanted!

BIG DATA MANAGEMENT

CZ/CE4123

Lec 1.1

Why big data?

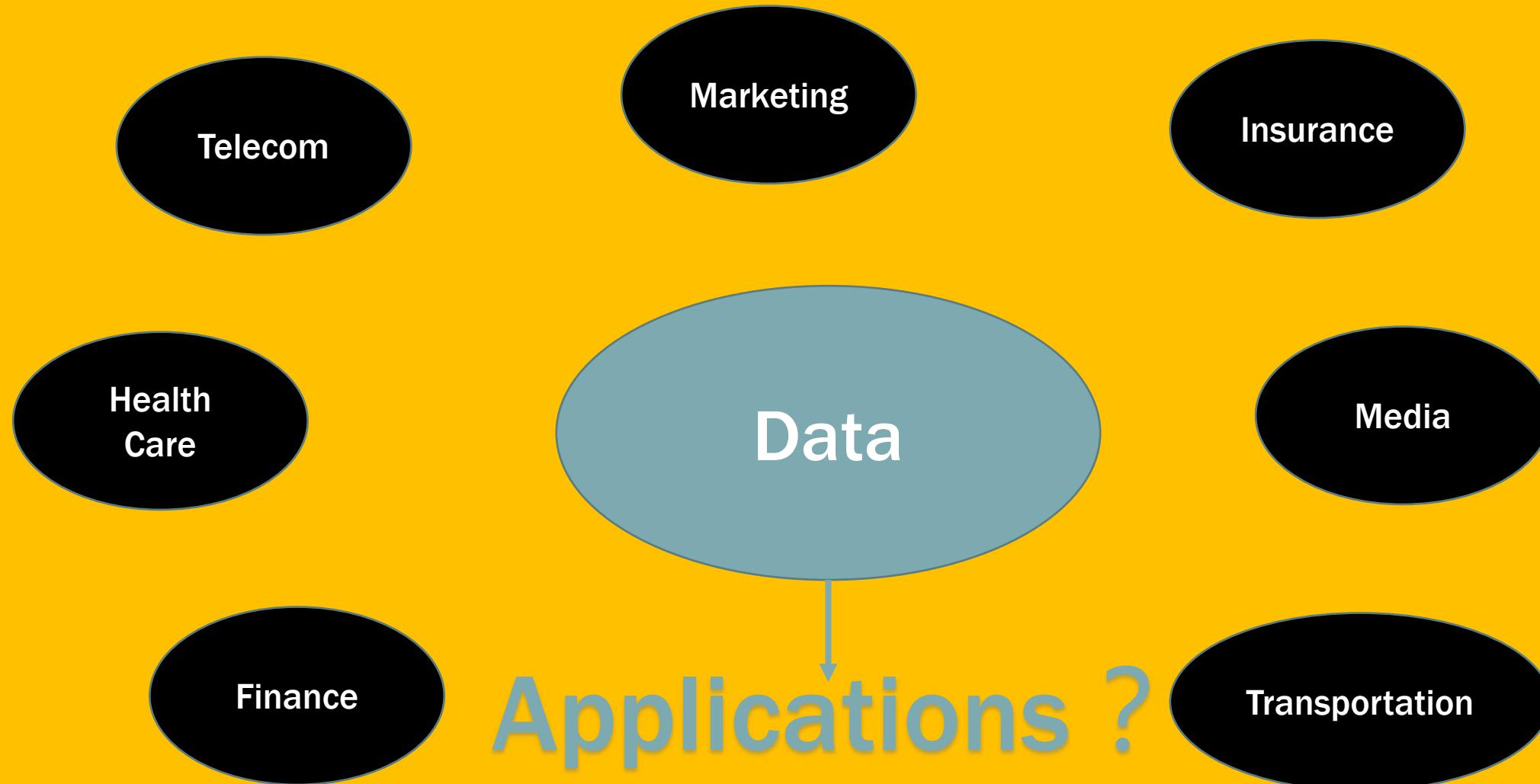


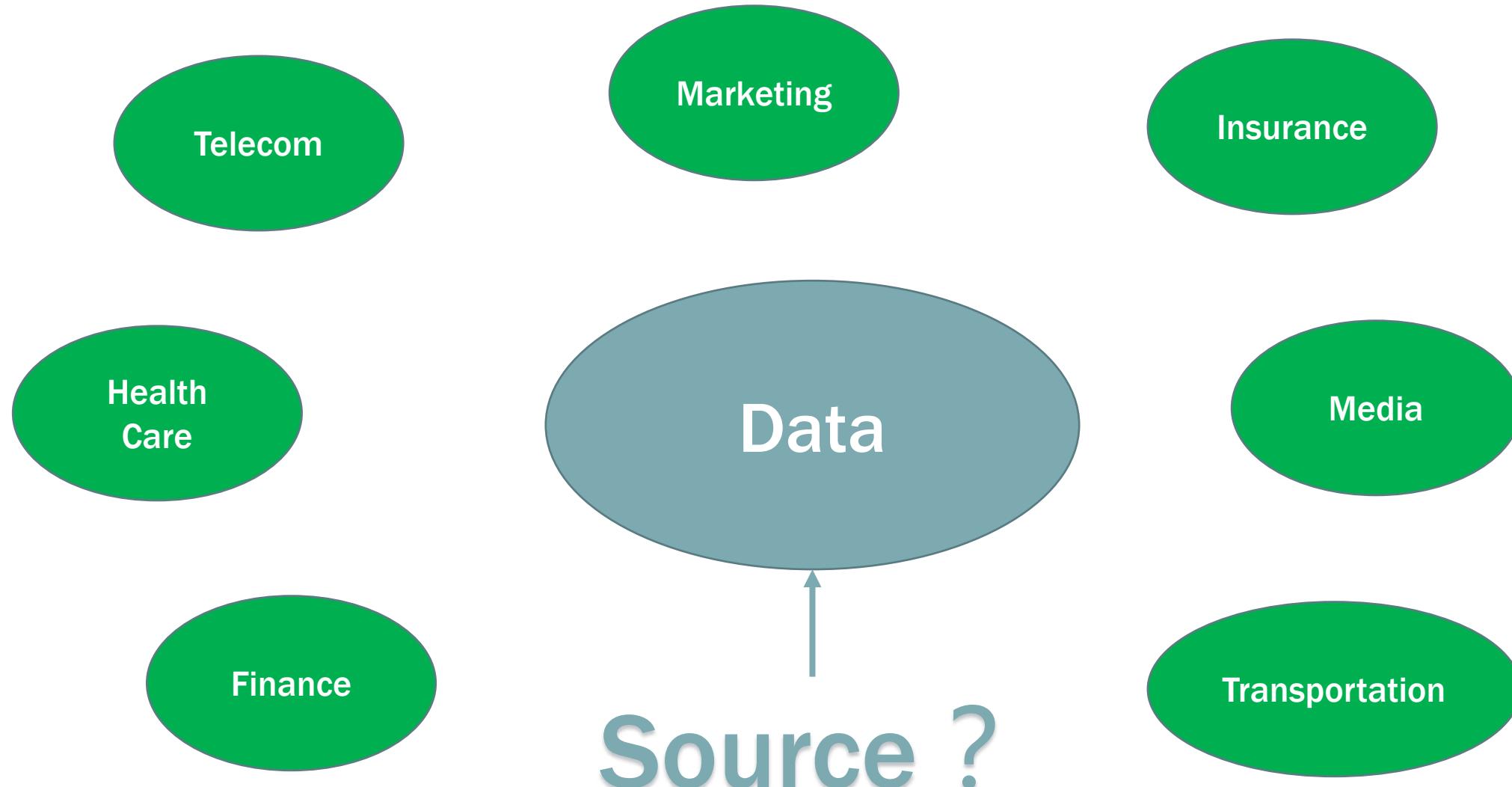
WHY WE NEED TO UNDERSTAND BIG DATA MANAGEMENT

Data

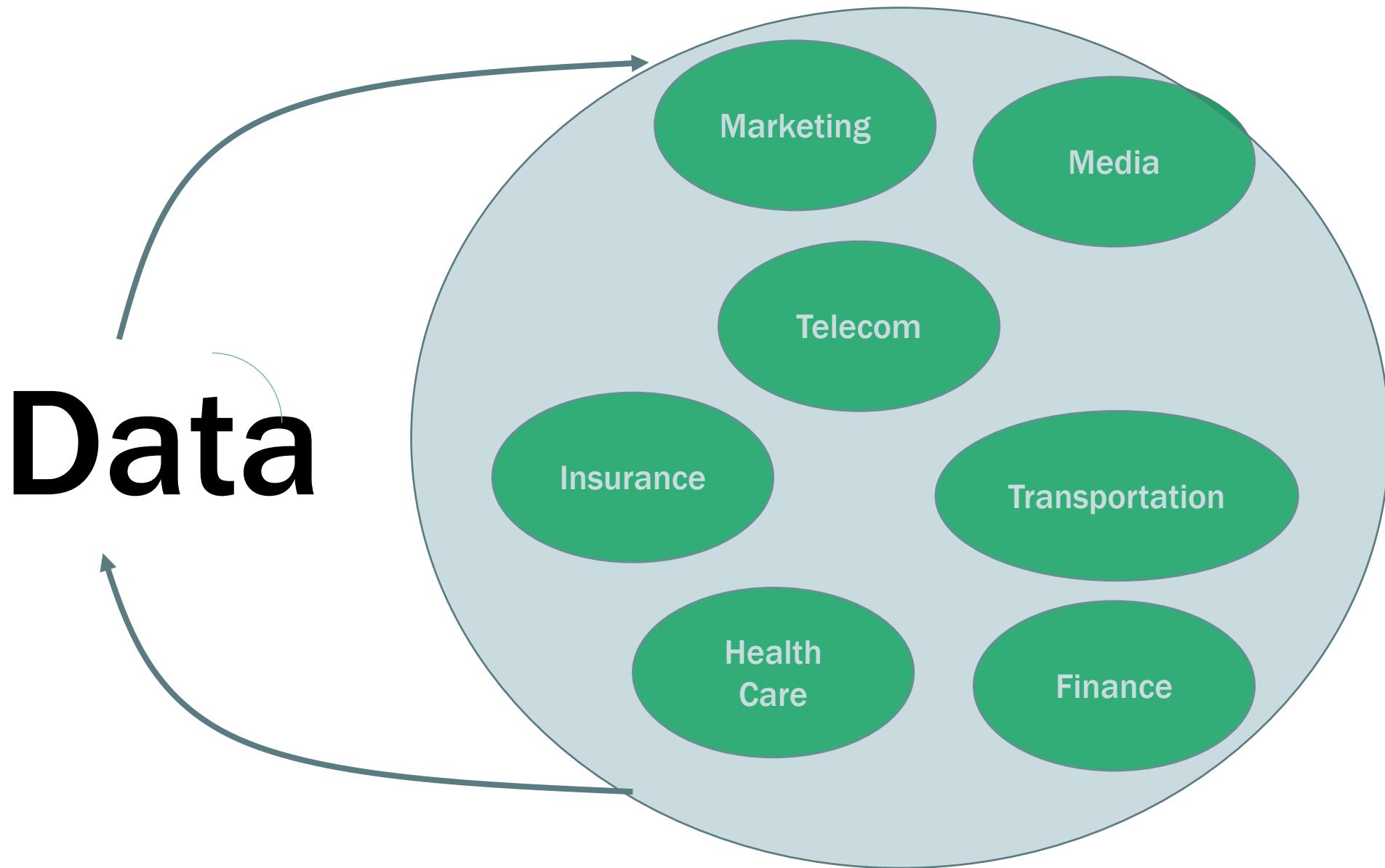
Data

Is at the center of most things





DATA LOOP



APPLICATIONS WE USE DAILY



Google



DATA SIZE

1 Bit – 1/8 of a letter

1 Byte – a letter (8 bits)

1 Megabyte – a book (1024 kilobytes)

1 Gigabyte -- ~1000 books (1024 megabytes)

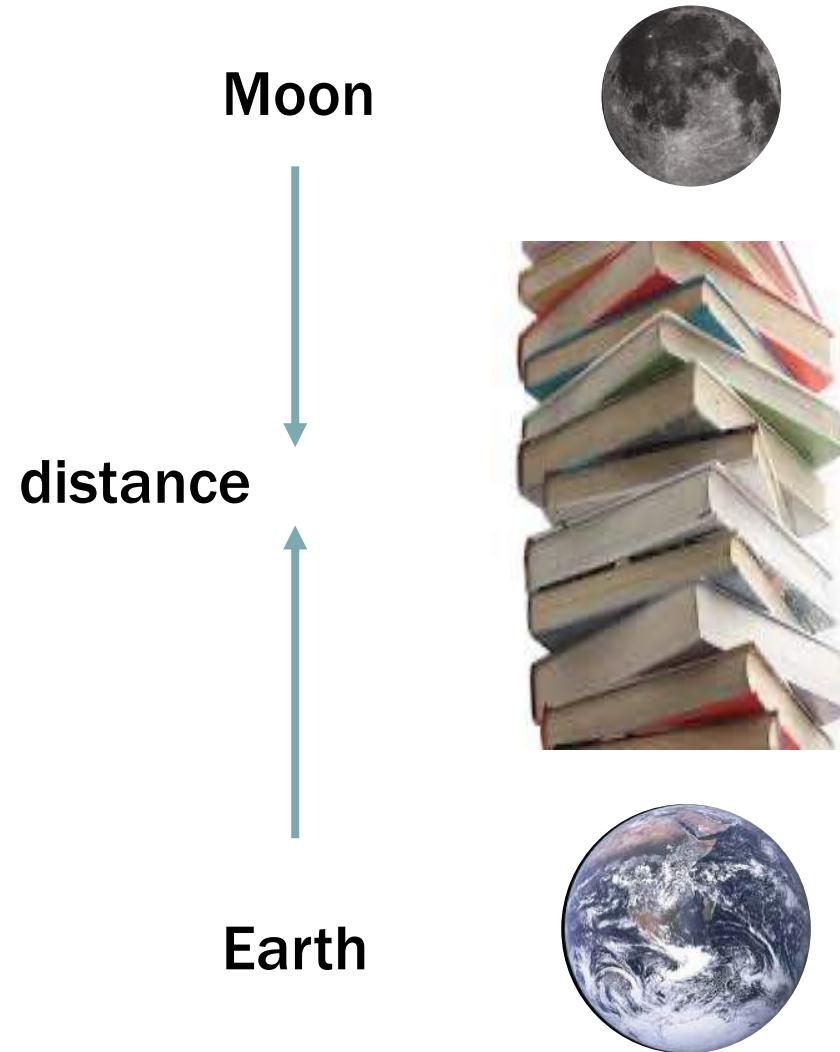
1 Terabyte -- ~ 1million books (1024 gigabytes)

1 Petabyte -- ~ 1billion books (1024 terabytes)

1 Exabyte -- ~ 10^{12} books (1024 Petabytes)

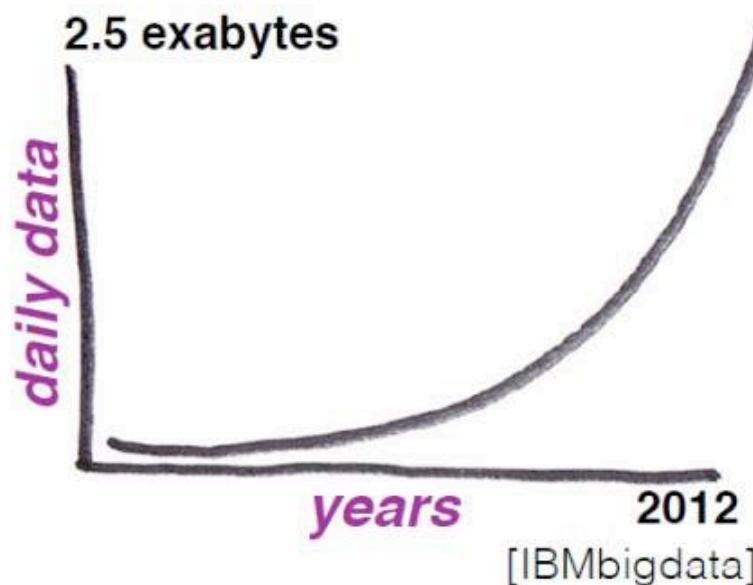
BOOKS PILING UP TO REACH MOON

1 Exabyte



HOW MUCH DATA GENERATED?

Nowadays, more than 2.5 exabytes of data are generated in every single date!



Every two days we create as much data as much we did from the dawn of humanity to 2003.

[Eric Schmidt, Google]

We are in the era of big data!

BIG DATA ECONOMY

>\$3 trillion/per year

A report from McKinsey Global Institute estimates that Big Data could generate an additional \$3 trillion in value every year in just seven industries. The report also estimated that over half of this value would go to customers in forms such as fewer traffic jams, easier price comparisons, and better matching between educational institutions and students.

>500,000 big data jobs

Data analysis has been called “the sexiest job of the 21st century.” The United States already has an estimated 500,000 Big Data jobs. But McKinsey estimates that there is a shortage of between 140,000 and 190,000 workers with advanced degrees in statistics, computer engineering and other applied fields. Perhaps more important is the shortage of 1.5 million managers and analysts who hold traditional jobs but are capable of integrating Big Data into their decision making.

THE KEY CHALLENGE OF BIG DATA

Data is so big that we need
High-Performance Computation over big data



efficiently store and query big data

CHALLENGES OF BIG DATA

How to store ?



<https://www.freeiconspng.com/images/question-mark-icon>



CHALLENGES OF BIG DATA

How to query ?



An SQL query walks into a bar and sees two tables.
He walks up to them and says "Can I join you?"

THE FIRST THING WE NEED TO KNOW

What are the characteristics of big data?

Next lecture:

Big Data 5V's

BIG DATA MANAGEMENT

CZ/CE4123

Lec 1, 2



What are important features of big data?



**Does your course schedule information
belong to a kind of big data?**



LEARNING OUTCOMES

- Understand what characterizes big data?
- Big data 5Vs
 - Volume
 - Velocity
 - Variety
 - Veracity
 - Value

What are important features?

5V's

What are important features?

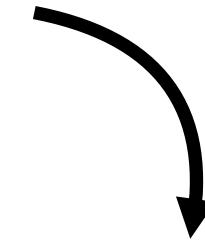
Volume

5V's

What are important features?

Volume

5V's



Velocity

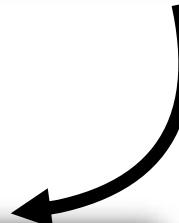
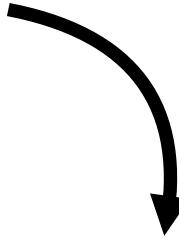
What are important features?

Volume

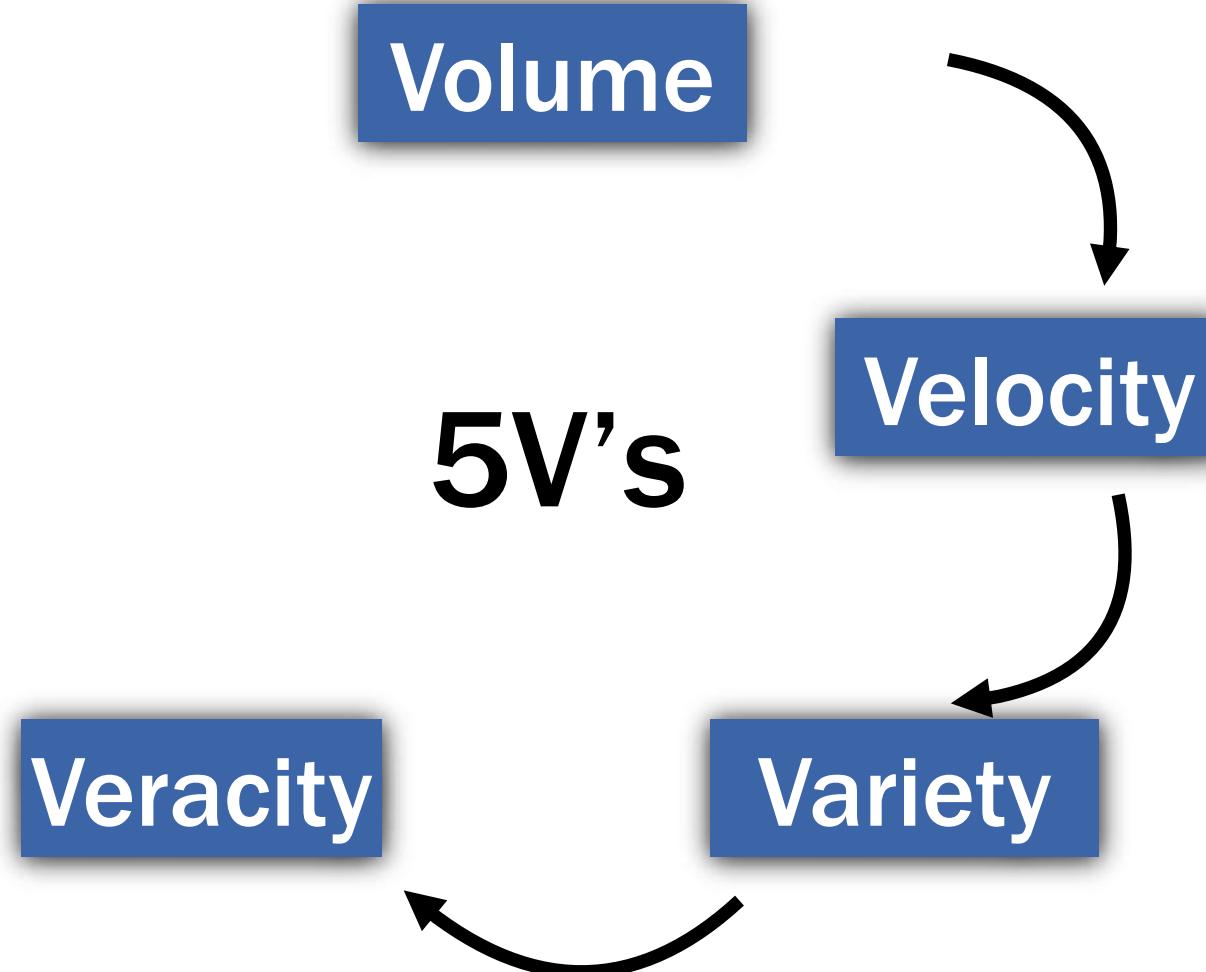
5V's

Velocity

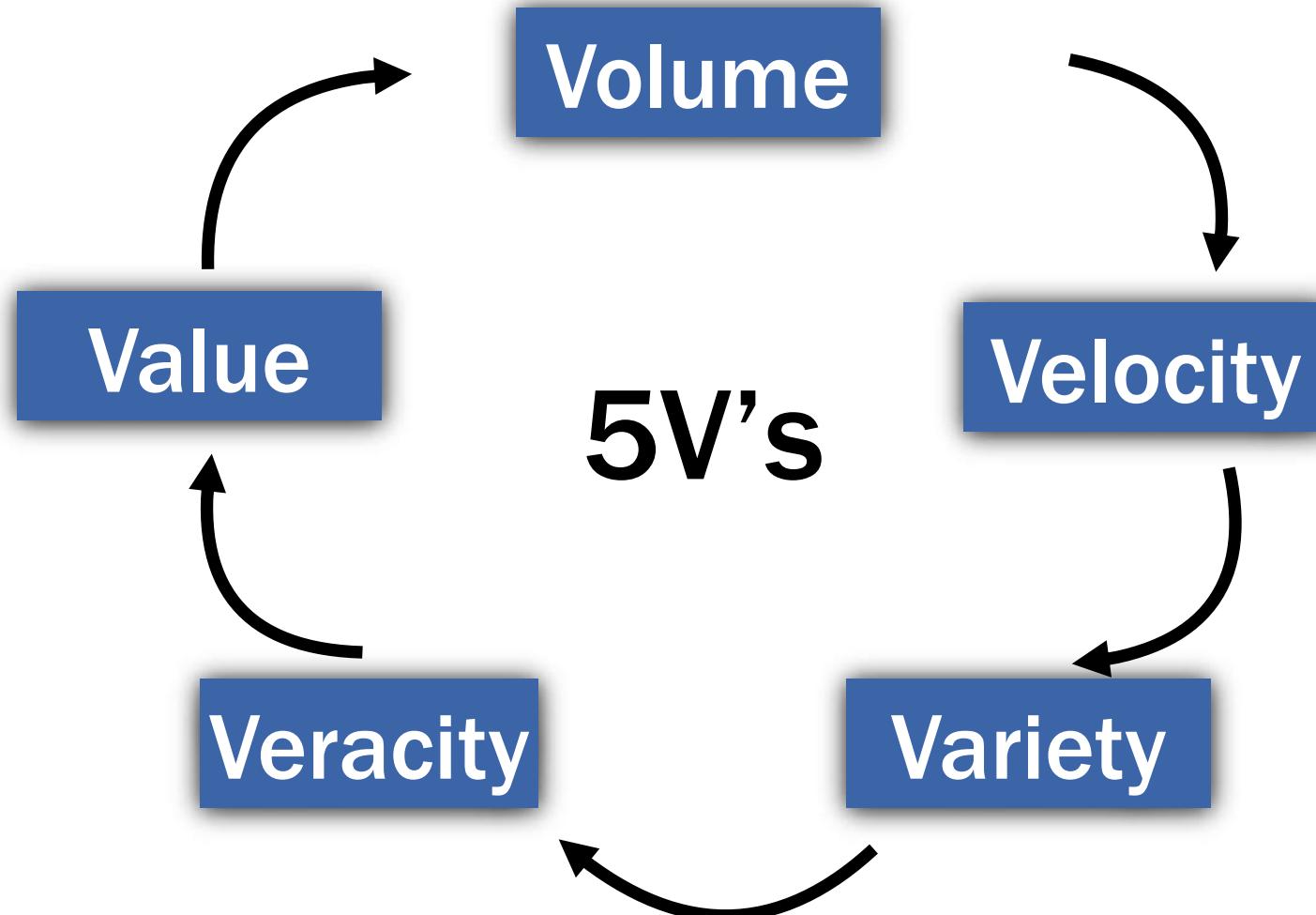
Variety



What are important features?



What are important features?



Volume

large amount of data

Volume

large amount of data



The monthly message data volume

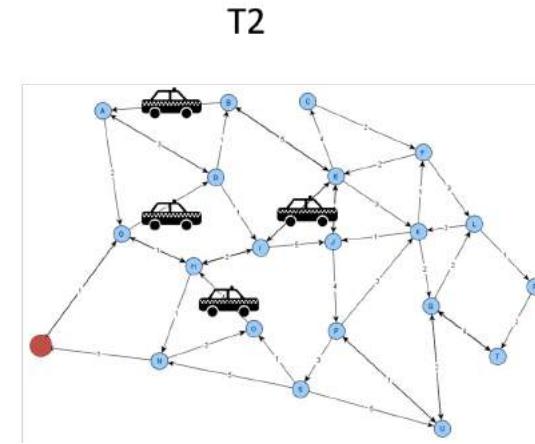
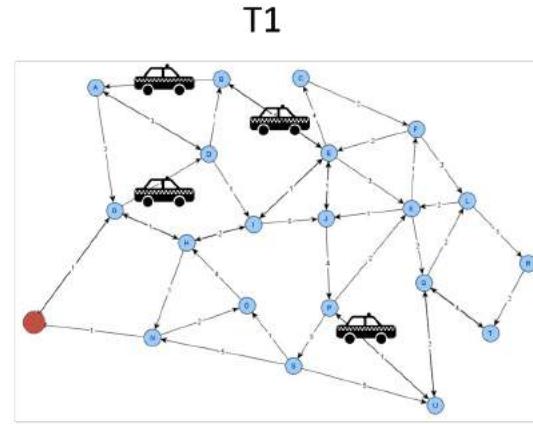
300 billion messages ~ 30TB

large!

Velocity

fast data generation

Velocity fast data generation



Example:
Taxi locations updated **every 5 seconds**

Frequent!

Variety

various data types/sources

Variety various data types

Structured data

Structured data concerns all data which can be stored in relational database in a table with rows and columns.



Excel

Unstructured data

Unstructured data is a data which is not organized in a predefined manner or does not have a predefined data model, thus it is not a good fit for a mainstream relational database.



Video

Variety

various data sources

1. Internet

2. Crowd sourcing
Wikipedia, forums

3. Social networks
Facebook, instagram

4. Sensors
Mobile phones, GPS



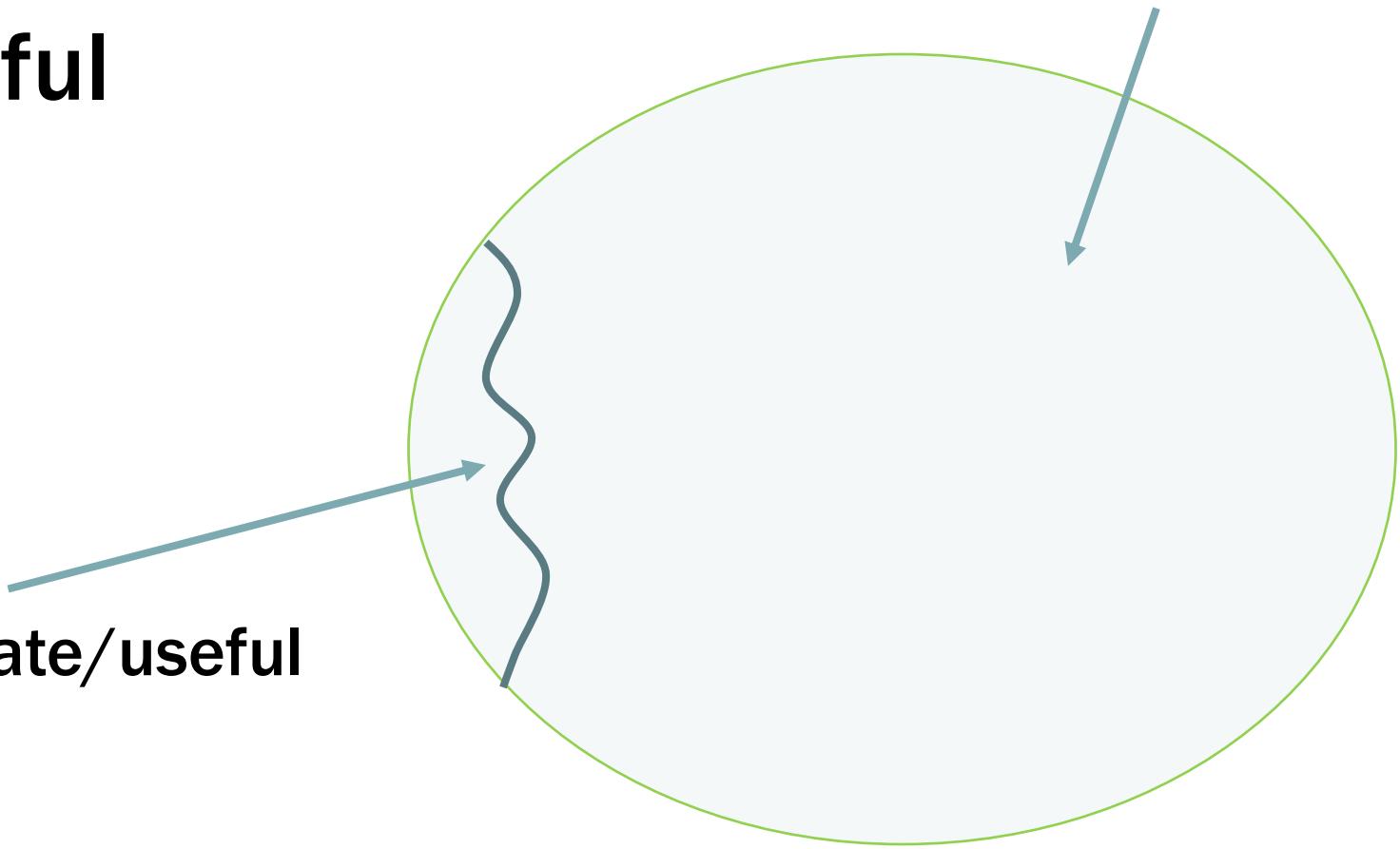
Veracity

accurate and truthful

Not accurate/not useful

Accurate/useful

Large volume



Veracity

accurate and truthful

High veracity \approx High data quality

Veracity

accurate and truthful



Statistical bias



Software bugs



Untrustworthy source

Veracity

accurate and truthful



Statistical bias

Some data points are given more weightage than others.



Software bugs



Untrustworthy source

Veracity

accurate and truthful



Statistical bias



Software bugs



Untrustworthy source

Data as software output
are distorted.

Veracity

accurate and truthful



Statistical bias



Software bugs



Untrustworthy source

Can you name some
trustworthy data sources?



Veracity

accurate and truthful



Statistical bias

Some data points are given more weightage than others.



Software bugs

Data as software output are distorted.



Untrustworthy source

Can you name some trustworthy data sources?

Text books, Research papers,
Professional magazines

Veracity

accurate and truthful

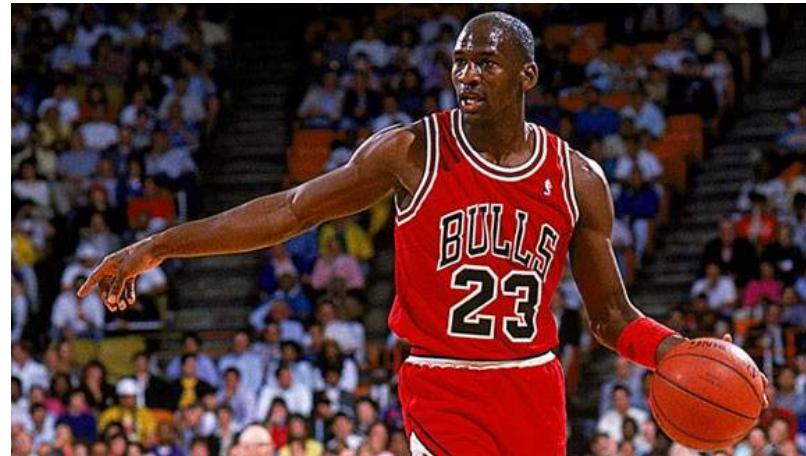
Name ambiguity

Who is Michael Jordan?

Veracity

accurate and truthful

In many people's mind:



Former NBA star Michael Jordan

Veracity

accurate and truthful

**Machine learning
people may think about**



Prof. Michael Jordan

Great contributions in ML/AI

Veracity

accurate and truthful

Sentence ambiguity



Summary of veracity

Big data should have high veracity;
that usually means the data are highly accurate and trustworthy,
particularly

- 1. do not have statistics bias,**
- 2. are not generated by software with bugs,**
- 3. come from reliable data source,**
- 4. have no/little data ambiguation**

Value

Benefits from analyzing the data

Value

Benefits from analyzing the data

Recommendation



Retail Marketing



Health Care



Computer Vision



Discussions & Questions

We finish big data 5V's!



Next lecture:

Data models

BIG DATA MANAGEMENT

CZ4123

DATA MODELS

Siqiang Luo

Assistant Professor

- ❑ In previous lectures, we discussed Big Data 5V's
 - ❖ Understand how to classify a big data application

- ❑ In this lecture, we will learn typical data models in big data systems
 - ❖ Geared to the mainstream big data systems

DATA MODEL AND PHYSICAL STORAGE SCHEME

- **Data model** describes how data are logically organized.
- Each data model can have different **storage schemes**.
 - Example: Relational model can be stored in row-oriented or column oriented.

DATA MODELS

- Relational Data Model
 - ❖ Corresponding to relational database

- Key-Value Data Model
 - ❖ Corresponding to key-value systems

- Graph Data Model
 - ❖ Corresponding to graph database

Relational Data Model

RELATIONAL DATA MODEL

Relational data contains a set of **Relations**

ID	name	age	gender
0001	Alex	25	M
0002	Mary	35	F

An example relation (Employee table)

RELATIONAL DATA MODEL

Schema -- specifies the **relation name**, and the **attribute** of each column.

❖ Example:

Employee (id, name, age, gender) ← schema

ID	name	age	gender
0001	Alex	25	M
0002	Mary	35	F

RELATIONAL DATA MODEL

Tuple: typically refers to a row of the relation

Attribute: corresponds to a column of the relation

	Attribute			
	id	name	age	gender
Tuple	0001	Alex	25	M
	0002	Mary	35	F

RELATIONAL DATA MODEL

Primary Key: A set of attributes that uniquely specify a row (usually there is an ID column)

Primary Key

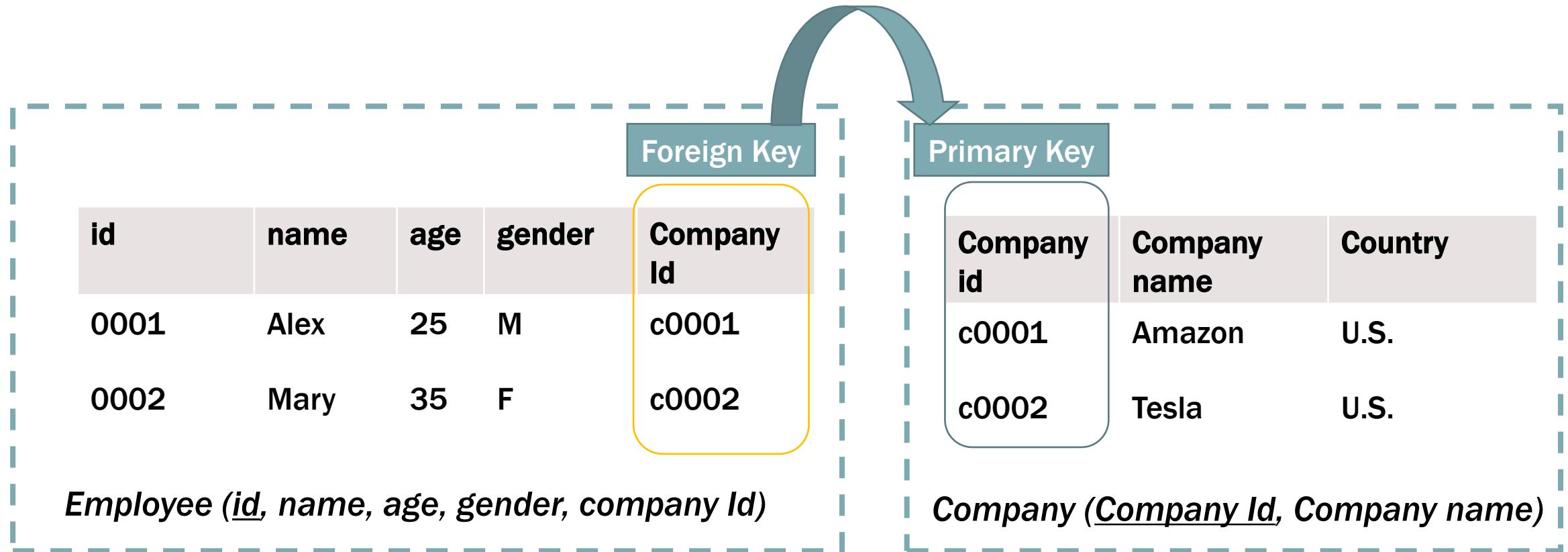
id	name	age	gender
0001	Alex	25	M
0002	Mary	35	F

Employee (id, name, age, gender)

→ Primary Key underlined

RELATIONAL DATA MODEL

Primary key – Foreign key relationship



A foreign key is a set of one or more columns in a table that refers to the primary key in another table.

TWO REPRESENTATIVE STORAGE SCHEMES

- A relation can be stored row-by-row (such a data system is often called a **row store**)
- A relation can be stored column-by-column (such a data system is often called a **column store**)
- We will discuss in more details when we in later lectures about “column store”.

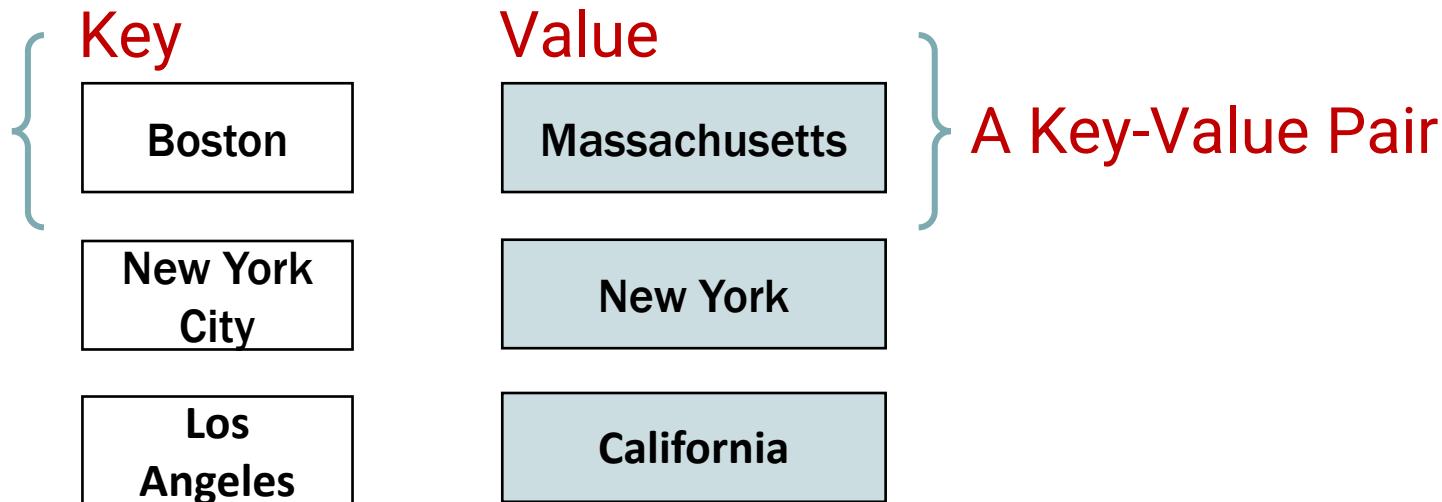
Key-Value Data Model

KEY-VALUE DATA MODEL

- The relational model has strict schemas.
- Some big data systems may require **schema-less** models.
- Key-value data model is one such kind.

KEY-VALUE DATA MODEL

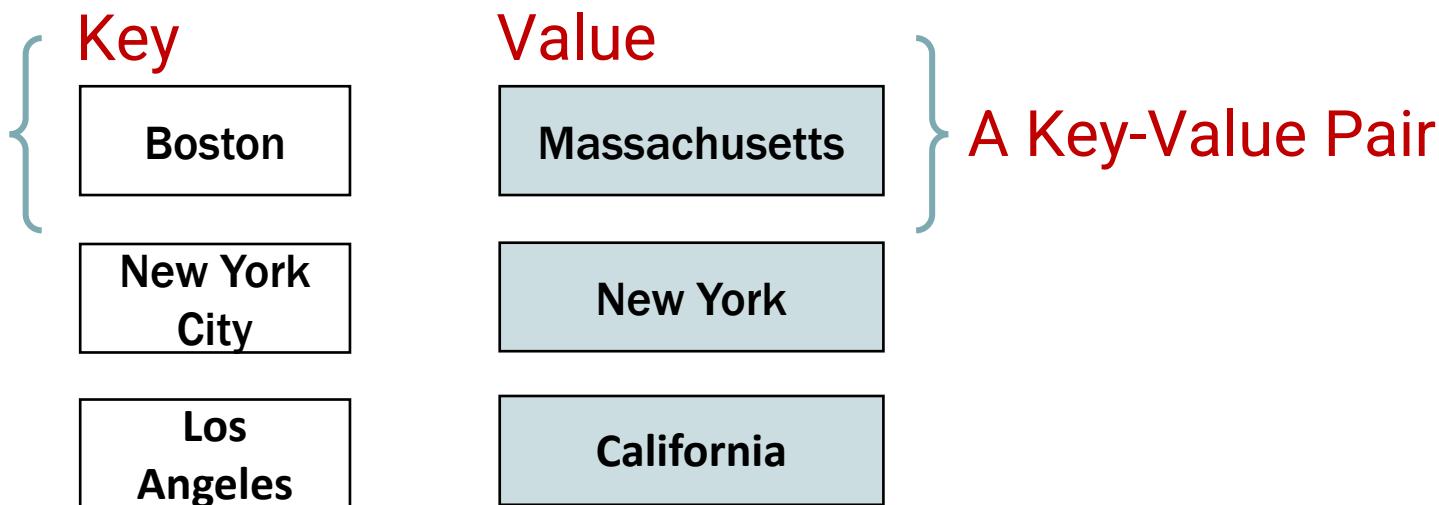
- Data is represented as a collection of key–value pairs.
- Key uniquely decides the pair



Above are (key, value) pairs describing the mapping between cities and states in the United States.

KEY-VALUE DATA MODEL

- ❑ It is usually less expressive than relational model but much **simpler**.
- ❑ It is preferred by a lot of real-systems including Facebook and Google in **analyzing big data**.
 - ❖ e.g., Google's levelDB, Facebook's RocksDB.



Key-value Data Model is ubiquitous!

For any A that can determine B

Key

Value

CHOOSING THE RIGHT KEY

Key Mapping Value

Key: Name

Value: ID

Alex

STU001

Bob

STU002

A good key-value model?



CHOOSING THE RIGHT KEY

How to put the Tweets data into key-value model?

Tweets **Tweets & replies** Media Likes

ACM SIGMOD @sigmod · Jan 3
 Wishing our community a happy new year from SIGMOD Record! The Dec 2021 issue is out now. We have very interesting medley of articles :) Check it all out at sigmodrecord.org/sigmod-record-...

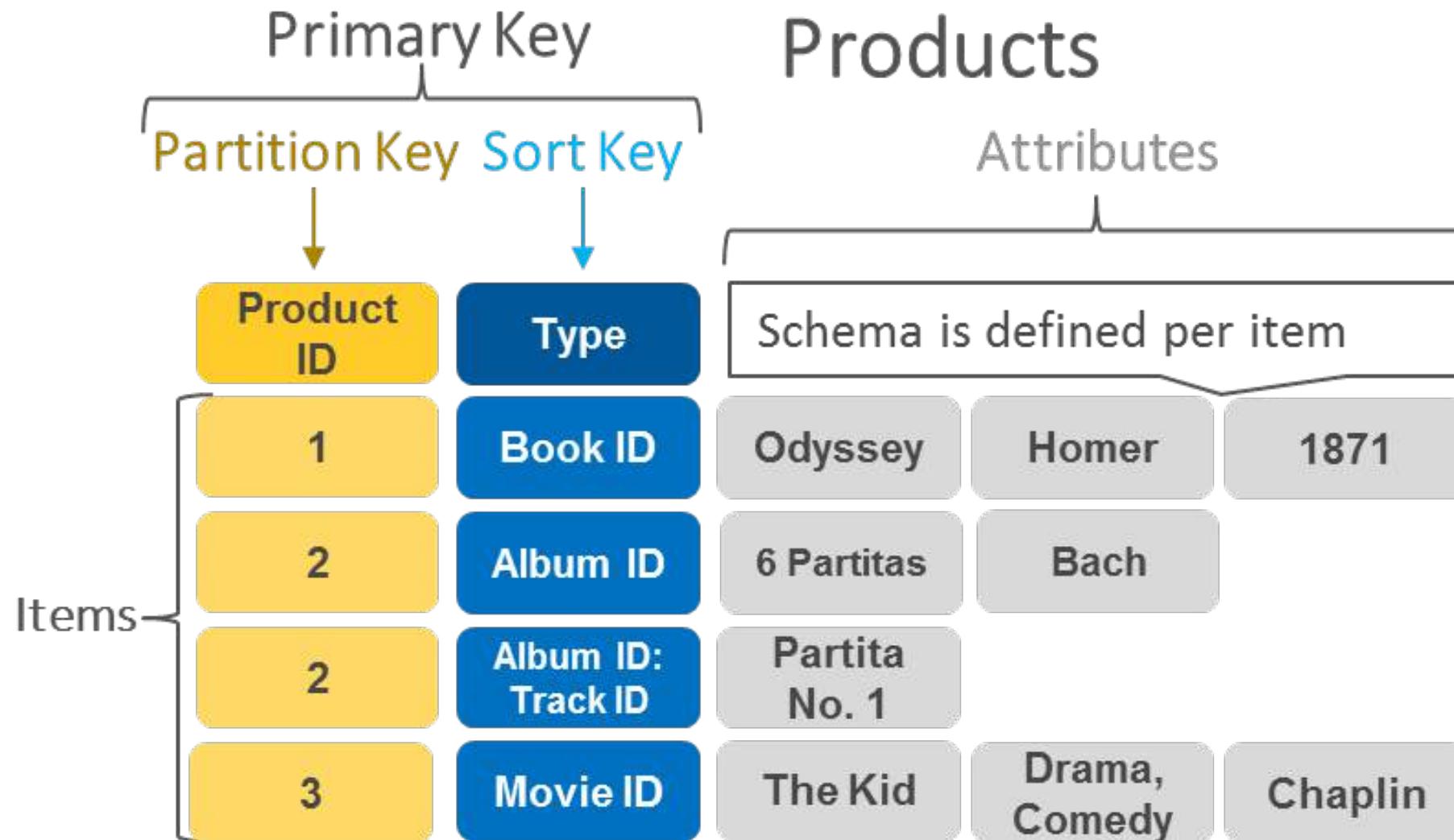
4 15 

ACM SIGMOD @sigmod · Oct 23, 2021
 New SIGMOD Record issue is out! We have some exciting articles including great advise from Jag to mid-career researchers, VLDB panel summary on the future of data(base) education, the DBrainstorming article on DB tuning, among others.

You may not have an explicit key in the dataset



AMAZON'S CASE



REMARK 1

Key-value model can “store”
the information of a relation

EXERCISE 1

Converting the following Relation/Table to key-value model

Primary Key

id	name	age	gender
0001	Alex	25	M
0002	Mary	35	F

SOLUTION

Key	Value		
Primary Key	Concatenate Other Attributes		
id	name	age	gender
0001	Alex	25	M
0002	Mary	35	F

0001 Alex;25;M

0001 Mary;35;F

REMARK 2

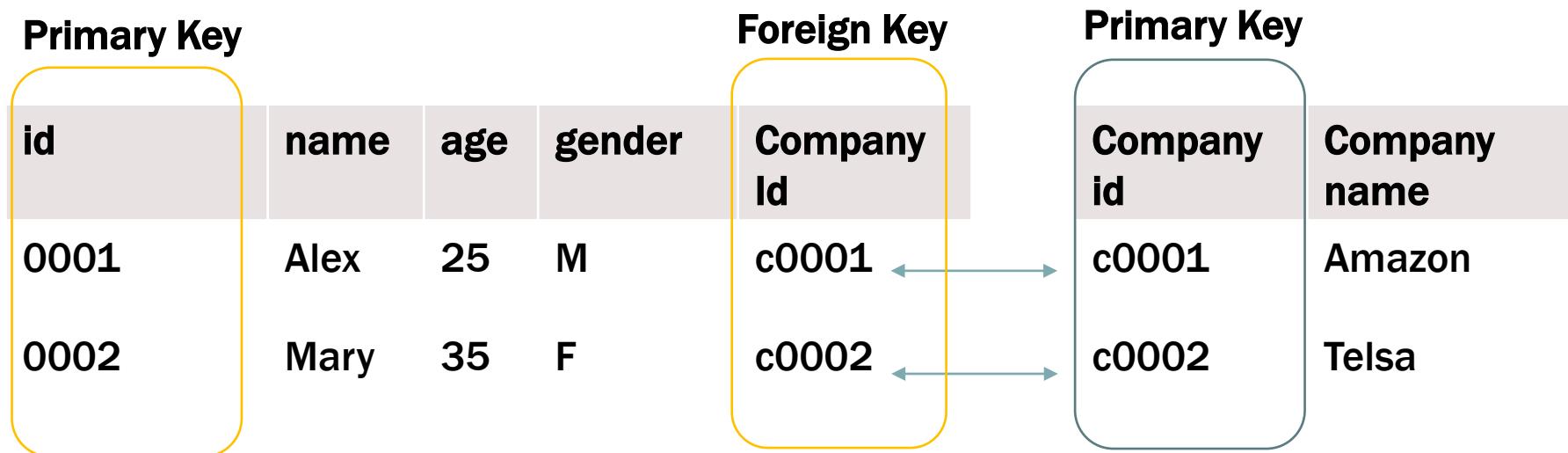
Key-value model can be mapped to a conceptual big table in the relational model!

EXERCISE 2

Primary Key				Foreign Key	Primary Key	
ID	name	age	gender	Company ID	Company ID	Company name
0001	Alex	25	M	c0001	c0001	Amazon
0002	Mary	35	F	c0002	c0002	Telsa

Given the above two tables (Employee and Company). If you do not worry about the storage and **always want to query the information of employees**, how would you convert them into a key-value model?

SOLUTION



Step 1: Join the table (Left outer-join from Employees)

ID	name	age	gender	Company-name	Company-ID
0001	Alex	25	M	Amazon	c0001
0002	Mary	35	F	Telsa	c0002

SOLUTION

Step 2: Make primary key as “Key”, and the others concatenated as values

Key: Id

Value: name; net worth; rank; company-name; company-id; CEO-name

ID	name	Net worth	rank	Company-name	Company-ID	CEO-name
0001	Jeff Bezos	\$201.4B	1	Amazon	c0001	Jeff Bezos
0002	Bernard Arnault & Family	\$181.6B	2	LVMH	c0003	Bernard Arnault

Note: We let key be ID because ID still uniquely defines a row in the big table.

COMPARING RELATIONAL MODEL AND KEY-VALUE MODEL

Advantages,
disadvantages?



COMPARING RELATIONAL MODEL AND KEY-VALUE MODEL

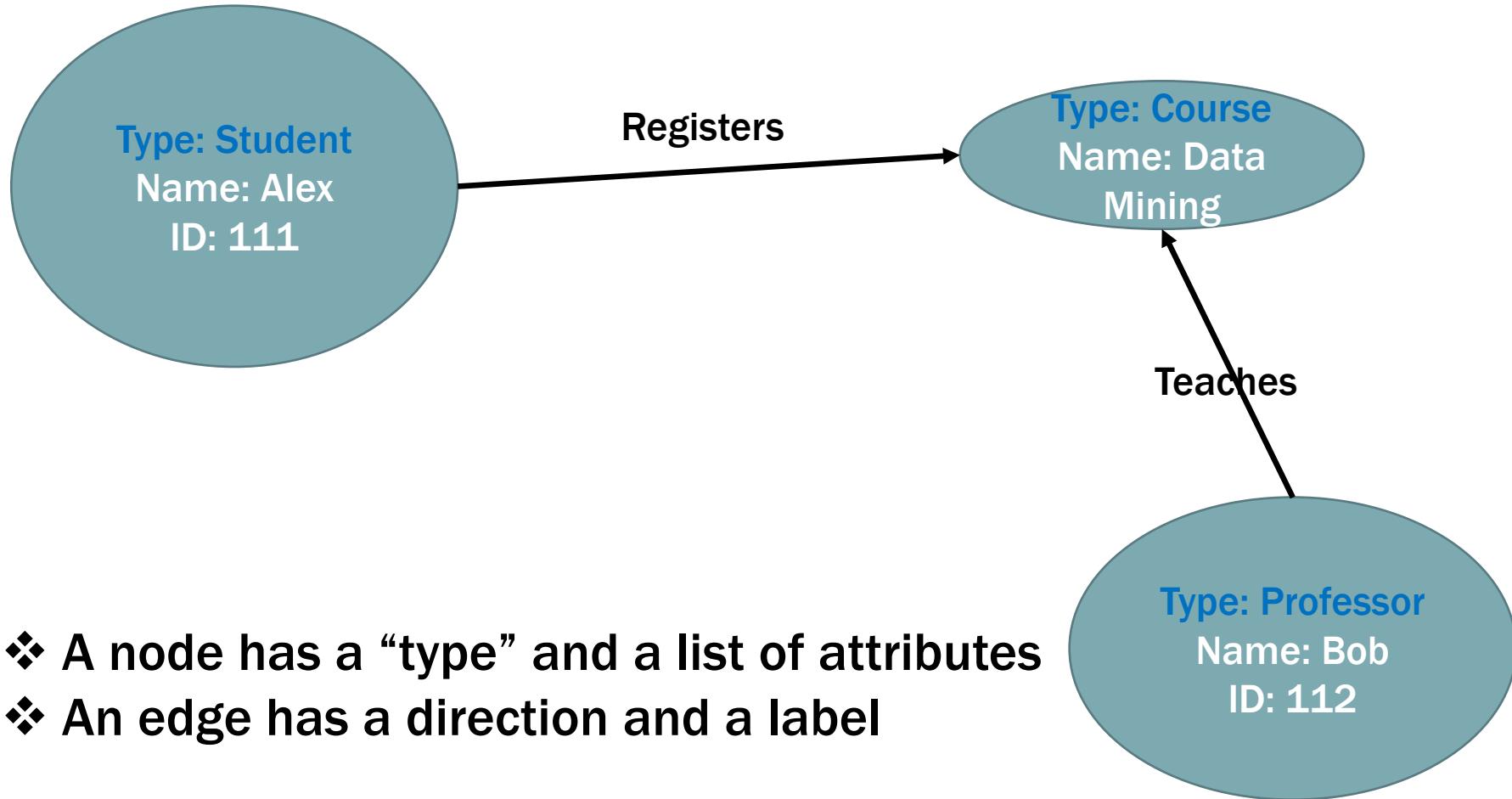
- Key-Value model is more flexible
 - Favoured by a lot of industrial-level big data systems, e.g., Facebook's RocksDB, Google's LevelDB
 - Assume most of the queries are simple (example: find a value corresponding to a key or a key range)
 - It is schema-less, making it commonly used in real-time web-based applications (highly partitionable, easy scaling)
 - Flexible to handle schema changes
- Relational model is more structured
 - Suitable to handle tabular data
 - Favoured by accuracy-sensitive systems, e.g., data systems in the bank
 - It has strict schemas, and is easy to design query languages (e.g., SQLs)

Graph Data Model

GRAPH MODEL

- There is another type of database called **graph database**
 - E.g., Neo4j, OrientDB
- **Graphs** are the underlying data model of graph databases
 - A graph is formed by **nodes** and **edges**
 - A node represents an entity
 - An edge represents the relationship between entities

GRAPHS ARE UBIQUITOUS



- ❖ A node has a “type” and a list of attributes
- ❖ An edge has a direction and a label

Primary Key

id	name	age	gender
0001	Alex	25	M
0002	Mary	35	F

Foreign Key

Company Id
c0001
c0002

Primary Key

Company id	Company name
c0001	Amazon
c0002	Telsa

How to convert the above relations/tables into a graph model?

Primary Key

ID	name	age	gender
0001	Alex	25	M
0002	Mary	35	F

Foreign Key

Company ID
c0001
c0002

Primary Key

Company ID	Company name
c0001	Amazon
c0002	Telsa

Type: Employee

id: 0001

Name: Alex

age: 25

gender: M



Type: Company

id: c0001

Name: Amazon

Type: Employee

id: 0002

Name: Mary

Net worth: 35

Gender: F



Type: Company

id: c0002

Name: Telsa

**Looks like relational model can do
the same thing. Why do we need
graphs?**



Suppose we want to model a social network like Facebook

One possible way is to build two relations:

User ID	Name
ID001	Alex
ID002	Mark
ID003	Mary
ID004	Bob

User Table

User ID1	User ID2
ID001	Id002
ID001	Id003
ID002	Id003
ID003	Id004

Friendship Table

Suppose we want to model a social network like Facebook

One possible way is to build two relations:

User ID	Name
ID001	Alex
ID002	Mark
ID003	Mary
ID004	Bob

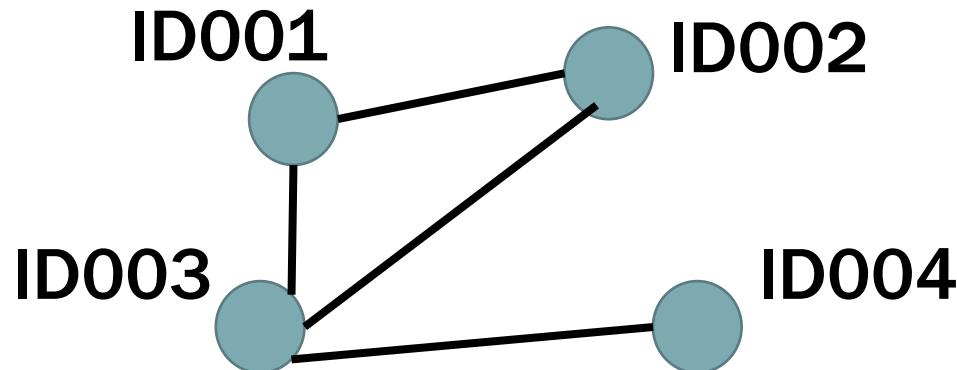
User Table

User ID1	User ID2
ID001	Id002
ID001	Id003
ID002	Id003
ID003	Id004

Friendship Table

Now, how do we answer a typical query :
“find the two most distant users”?

- There are some queries that require to explore the complex structures of the entities.
- For these queries, it is more suitable to consider the data as a graph.
- The social network is modeled as a graph as follows



We can then compute all-pair shortest path algorithms on the graph to answer the query

We finish Data Models!



Next lecture:

Big data and Memory Hierarchy

BIG DATA MANAGEMENT

CZ4123

MEMORY HIERARCHY

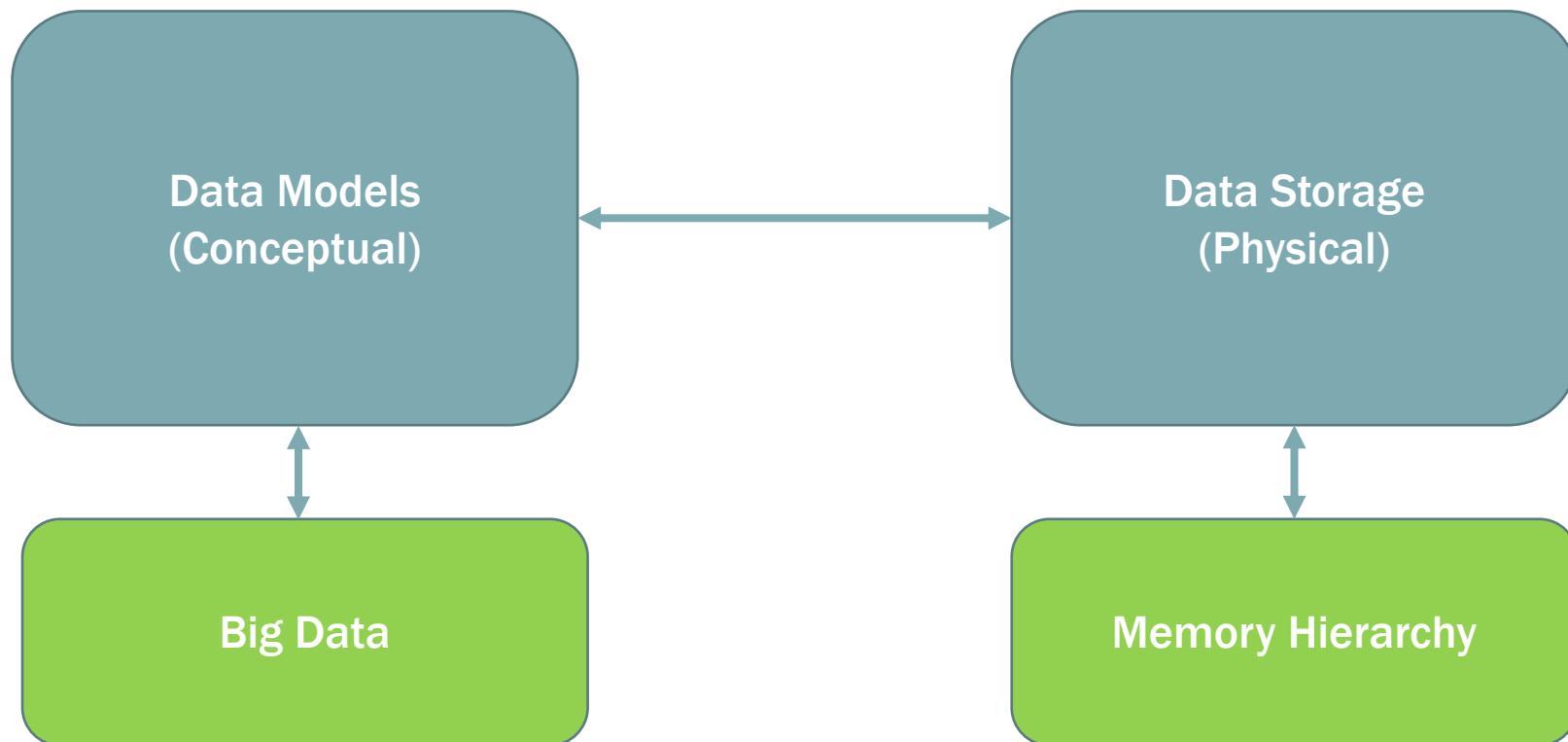
(PART I)

Siqiang Luo

Assistant Professor

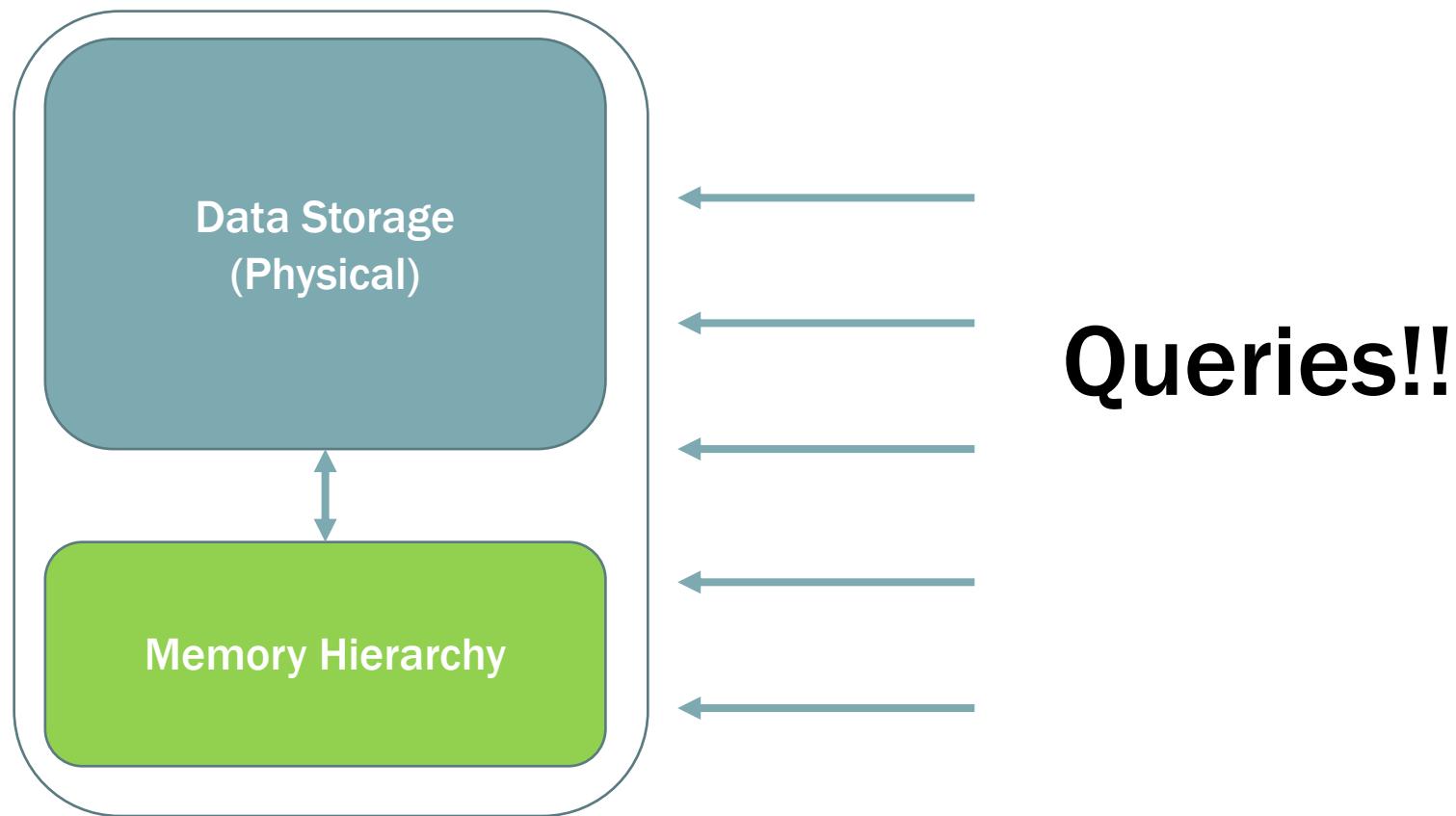
PREPARATION

- In previous lectures, we have learnt (conceptual) data models. These data models need to be (physically) stored in the storage medium (e.g., disks).



PREPARATION

- ❑ “How to **store** the data” must be related to “how to **retrieve/query them efficiently**”.



PREPARATION

What is an ideal case for storing big data?



PREPARATION

What is an ideal case for storing big data?



- Ideally, we should have infinite size of fast accessing storage, and they are persistent.
 - Infinite size to store big data.
 - Fast accessing guarantees fast read/write of the data.
 - Persistency ensures keeping the data when we power off the system.

PREPARATION

**However, the ideal case is not easy
to realize**

PREPARATION

The **dilemma** of storage design

- Fast storage with a large size is expensive; we may afford
 - Faster storage with a smaller size
 - Slower storage with a larger size

PREPARATION

The dilemma of storage design

- Fast storage with large size is expensive; we may afford
 - Faster storage with smaller size
 - Slower storage with larger size
- Suppose you are given 800SGD, how do you allocate your budget to buy different types of storage?
 - Cache: \$20/MB
 - Main Memory: \$20/GB
 - Disk: \$20/TB



PREPARATION

If all the budget is for disk

large storage
(40TB)

Slow access



If all the budget is for cache

small storage
(40MB)

Fast access



PREPARATION

Get large and fast enough storage



Memory Hierarchy

We will introduce the concept more formally

OUTLINE

- ❑ What is memory hierarchy?
- ❑ Why we need memory hierarchy?

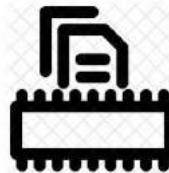
OUTLINE

- ❑ What is memory hierarchy?
- ❑ Why we need memory hierarchy?

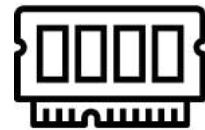
CONCEPTS

- The storage space in the computer is used to store data and instructions
 - Instructions are the “codes” that tell the processor what to do
- The storage space is divided into multiple cells, each having an address
 - Most modern computers are byte-addressable.
 - Each address identifies a single byte of storage.
 - For example, 256GB memory has $256 * 1024 * 1024 * 1024$ addresses

THREE MAIN CATEGORY OF STORAGE



Cache Memory



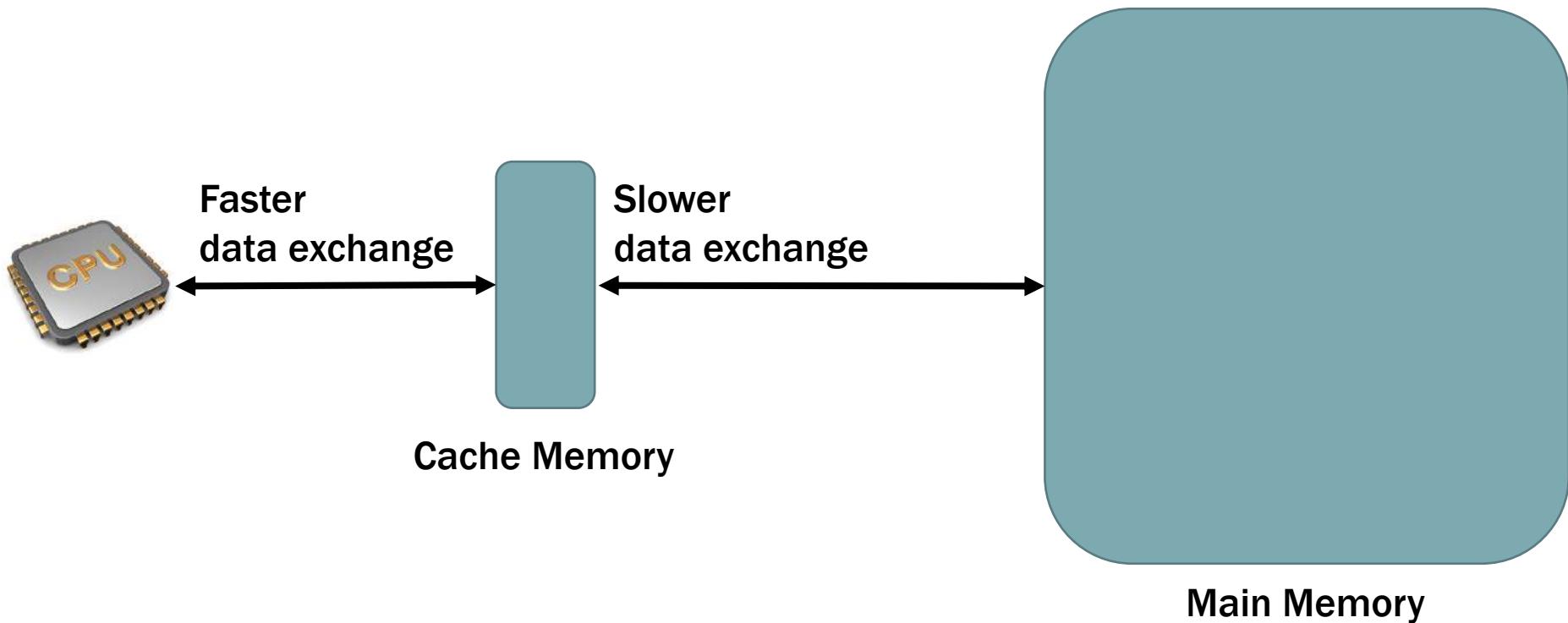
Main Memory



Secondary Memory

CACHE-MEMORY

- ❑ High speed
- ❑ Small capacity (often between 8 KB and 64 KB)
- ❑ Expensive
- ❑ Regarded as a buffer between the CPU and the slower main memory
- ❑ Hold data and program instructions which are most frequently used by the CPU



MAIN-MEMORY

- Holds data and instructions on which the computer is currently working
- Relatively high speed but slower than cache-memory
- Data is lost if power-offed
- Much larger capacity than cache memory (often between 2 GB and 32 GB)
- Less expensive than cache memory

SECONDARY-MEMORY

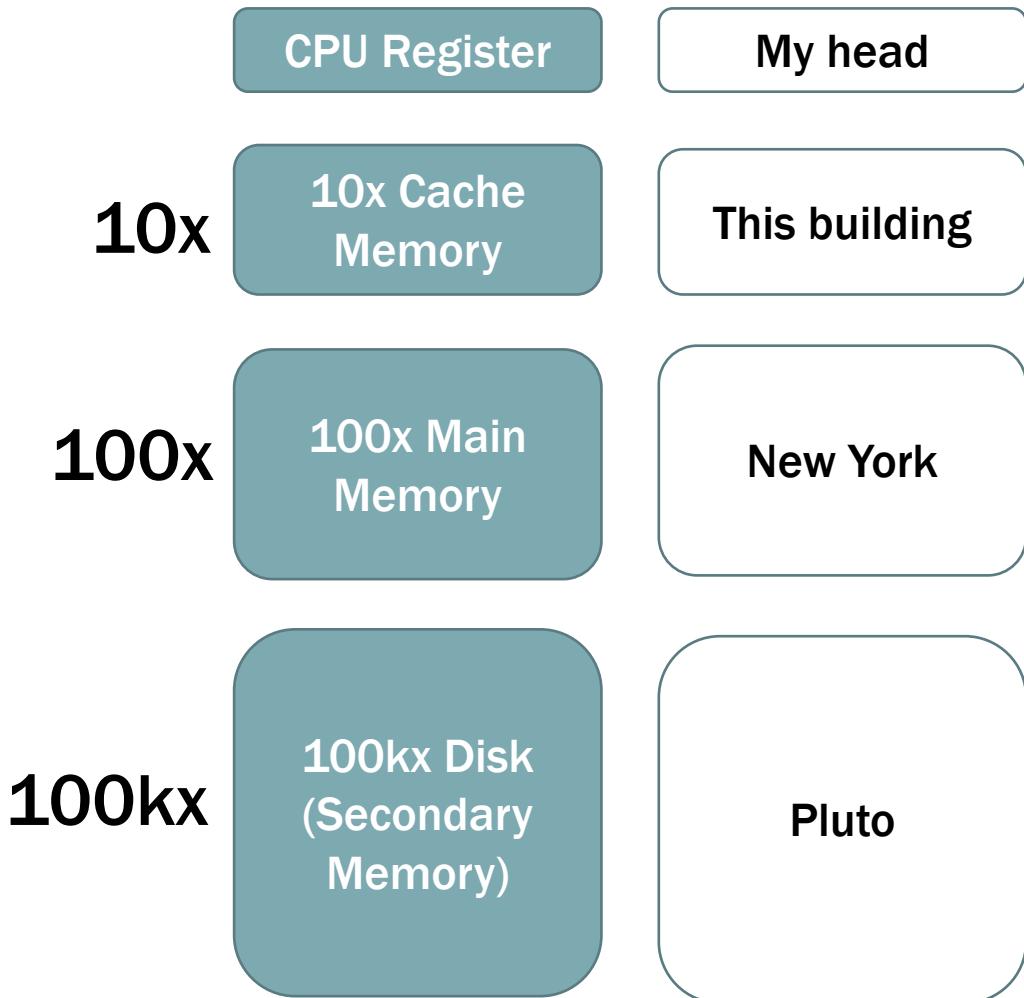
- Also known as external memory
 - in many situations, we may simply use the most common example “disk”.
- Much slower than main memory when accessing data
 - Even with SSD (Solid State Disk), the accessing cost is still higher
- Can store data permanently
- Example: Hard Disk, SSD

MEMORY HIERARCHY

People often simplify

- Cache memory → Cache
- Main Memory → Memory
- Secondary Memory → Disk

Memory Hierarchy



Note for Register: small amounts of high-speed memory contained within the CPU

Jim Gray's analogy on memory hierarchy

MORE COMPLICATED MEMORY HIERARCHY

- ❑ In fact, cache can be further divided into L1 cache, L2 cache, L3 cache.
 - ❑ L1 cache is the fastest but smallest
 - ❑ L3 cache is the slowest but largest
 - ❑ L2 cache is in the middle
- ❑ For simplicity, in this course we simply merge them into a simple cache layer.

OUTLINE

- ❑ What is memory hierarchy?
- ❑ Why we need memory hierarchy?

WHY WE NEED MEMORY HIERARCHY

- ❑ Suppose we need 256GB storage for a computer, can we design a computer with 256GB **cache memory** without main memory and disk/secondary memory?



WHY WE NEED MEMORY HIERARCHY

❑ Suppose we need 256GB storage for a computer, can we design a computer with 256GB **cache memory** without main memory and disk/secondary memory?

❑ -- It can be too expensive



❑ -- No permanent storage



WHY WE NEED MEMORY HIERARCHY

- ❑ Suppose we need 256GB storage for a computer, can we design a computer with 256GB **main memory** without cache memory and disk?



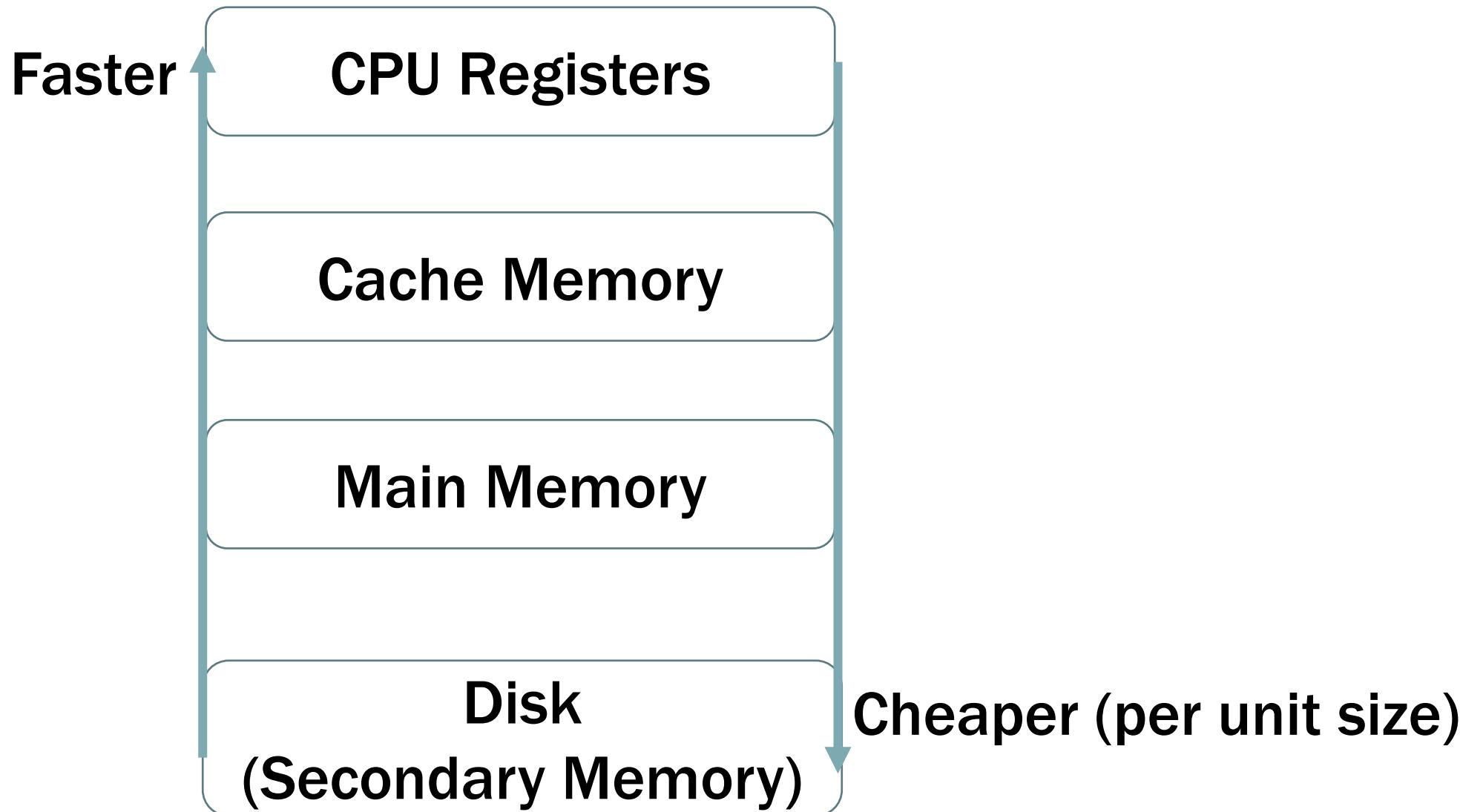
WHY WE NEED MEMORY HIERARCHY

- ❑ Suppose we need 256GB storage for a computer, can we design a computer with 256GB **main memory** without cache memory and disk?
- ❑ -- It can still be relatively expensive 
- ❑ -- No permanent storage 
- ❑ -- No buffer between CPU and main memory. The CPU speed can be much faster than main memory, causing latencies. 

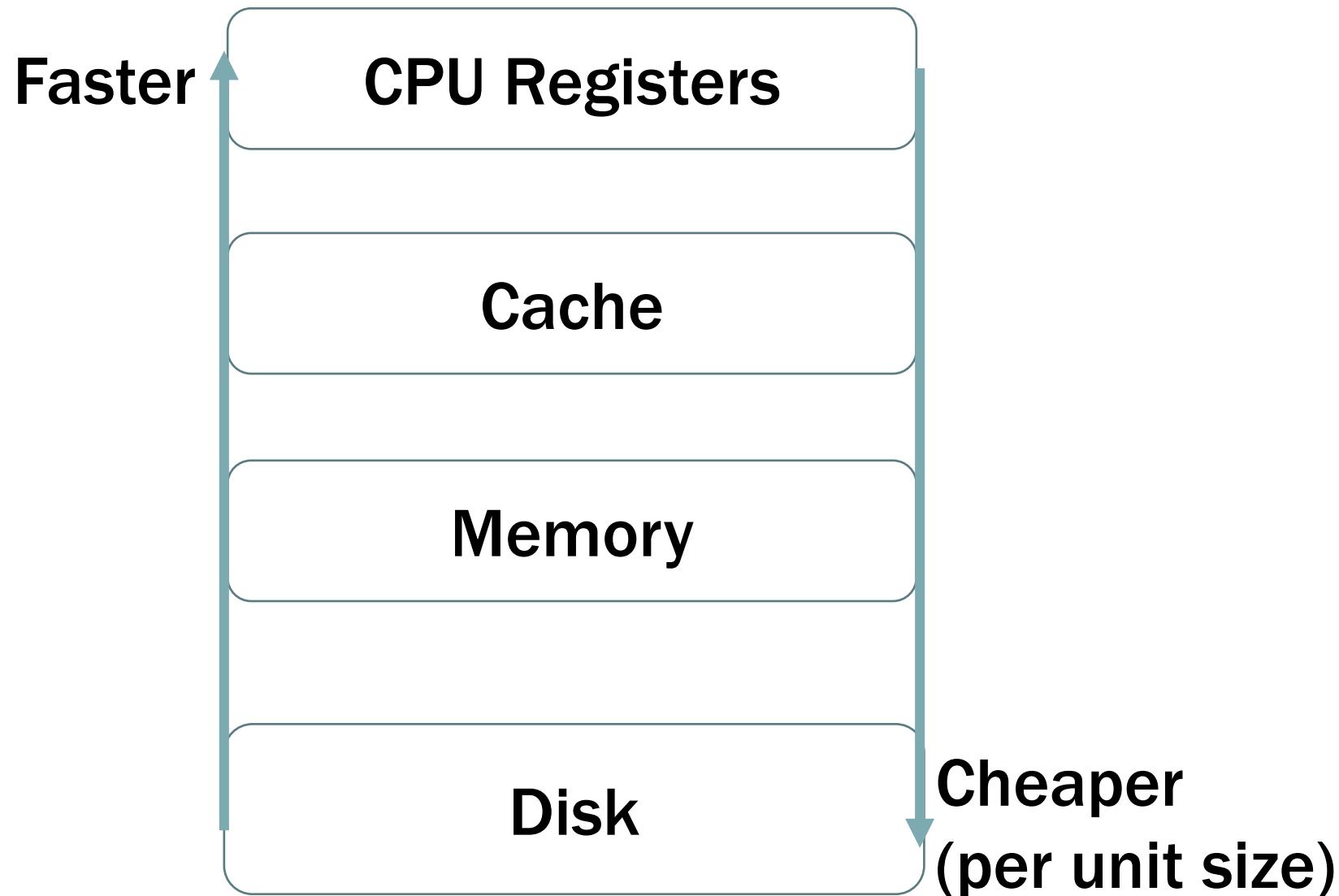
WHY WE NEED MEMORY HIERARCHY

- We need **fast** memory to sync up CPU speed → **Need Cache Memory**
 - Processor speed is much faster than main memory access time
- We need **large** space for **permanently** storing **big data**
 - Price becomes a concern
 - Faster memory (e.g., cache memory) is much more expensive

TRADE-OFF BETWEEN SPEED AND PRICE



TRADE-OFF BETWEEN SPEED AND PRICE



SUMMARY OF MEMORY HIERARCHY

- ❑ Memory hierarchy consists of a set of memory layers, where a faster memory layer has a smaller capacity.
- ❑ Memory hierarchy is needed concerning the following factors (most important ones)
 - ❑ Reasonable price
 - ❑ Enough capacity to hold data
 - ❑ Data persistency

BIG DATA MANAGEMENT

CZ4123

MEMORY HIERARCHY (PART II)

Siqiang Luo

Assistant Professor

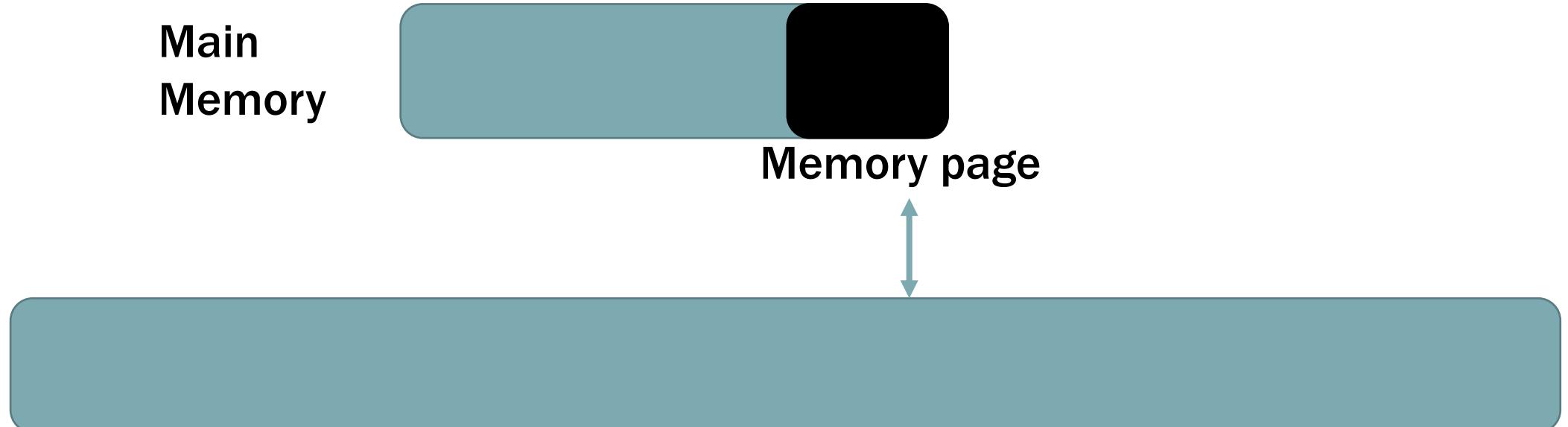
Data in Memory Hierarchy: Basic Cost Analysis



Big data cannot be fully loaded into main memory. Hence, often there are **data movements** between main memory and disks.

DATA ACCESS IN MEMORY HIERARCHY

- Big data cannot be fully loaded into main memory. Hence, often there are data movements between main memory and disks.



DATA ACCESS IN MEMORY HIERARCHY

- We often analyze the data access cost considering two consecutive layers: Layer i and i+1.
 - (Layer i, Layer i+1) can be (cache, memory)
 - (Layer i, Layer i+1) can also be (disk-cache, disk), where disk-cache is part of memory.

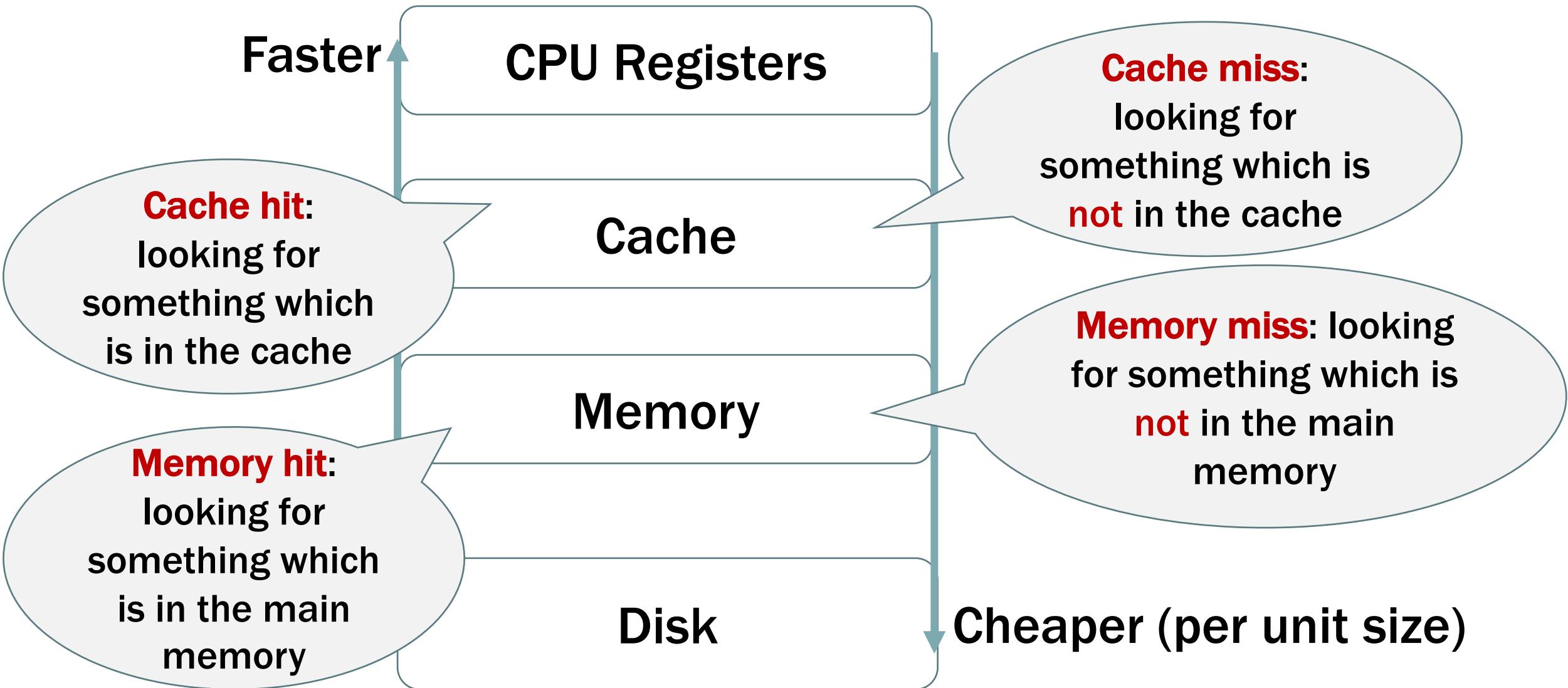
Layer i
(smaller, faster)

Layer i+1
(bigger, slower)

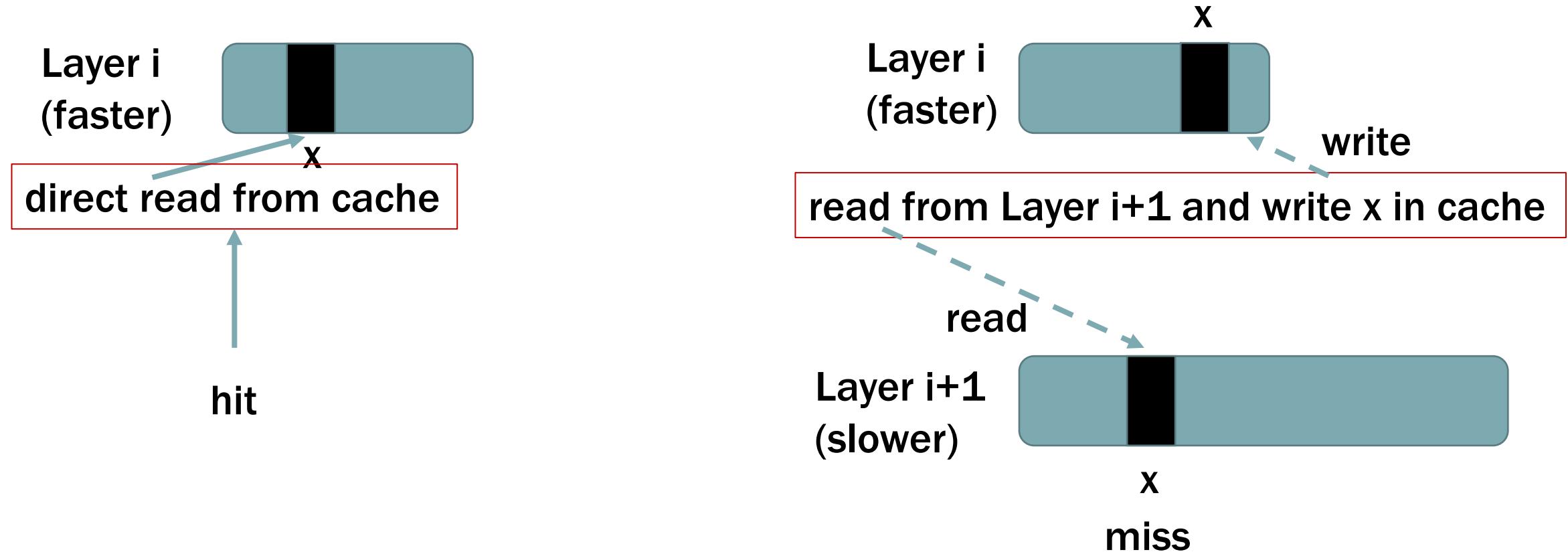
DATA ACCESS IN MEMORY HIERARCHY

- Some data item in Layer $i+1$ has been cached in Layer i
- When reading a data item in Layer $i+1$, it will first check whether the data item is cached in Layer i .
 - If yes, directly read the data from the faster Layer i
 - If no, read the data from Layer $i+1$ and write the data into Layer $i+1$'s cache (usually located in Layer i).

CACHE HIT/MISS AND MEMORY HIT/MISS



DATA ACCESS IN MEMORY HIERARCHY



COST OF A CACHE MISS

Question:

Suppose a data item is located in **main memory** and can be **cached** in the **cache memory**.

1. accessing cache memory once incurs *cost_access_cache*;
2. accessing main memory once incurs *cost_access_mem*;
3. cache miss rate is h ($0 \leq h \leq 1$).

How to estimate the cost of reading data item?



COST OF A CACHE MISS

- ❑ If cache hit: $\text{cost_access_cache} \times (1-h)$
 - ❑ If cache miss: $(\text{cost_access_cache} + \text{cost_access_mem}) \times h$
 - ❑ Overall: $\text{cost_access_mem_overall}$
 $= \text{cost_access_cache} \times (1-h) + (\text{cost_access_cache} + \text{cost_access_mem}) \times h$
- A simplified estimation

COST OF A CACHE MISS

- Overall: $\text{cost_access_mem_overall}$

$$= \text{cost_access_cache} \times (1-h) + (\text{cost_access_cache} + \text{cost_access_mem}) \times h$$

- If $h = 1$ (cache miss rate is high), then the cost is

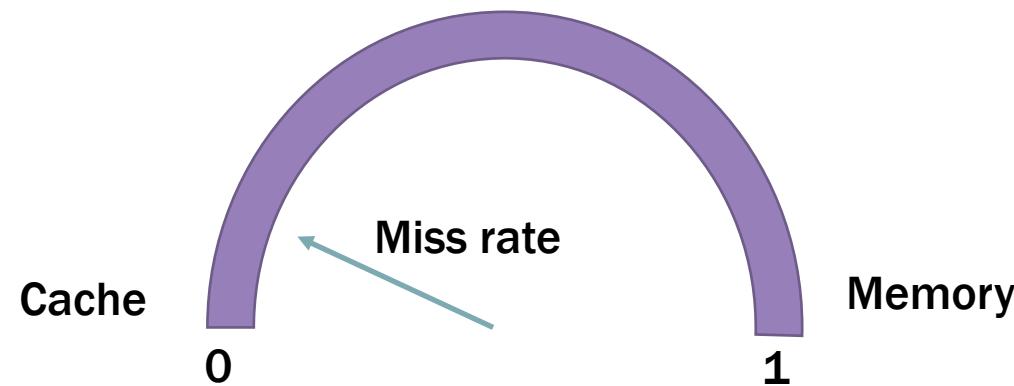
$$\text{cost_access_cache} + \text{cost_access_mem} \approx \text{cost_access_mem}$$

- The above estimation is due to the fact that cost_access_mem is significantly higher (say, 1000x) than cost_access_cache

- If $h = 0$ (cache miss rate is low), then the cost is cost_access_cache

COST OF A CACHE MISS

- ❑ Overall: `cost_access_mem_overall`
$$= \text{cost_access_cache} \times (1-h) + (\text{cost_access_cache} + \text{cost_access_mem}) \times h$$
- ❑ Summary
 - ❑ If cache miss rate is low, then data read performance is close to cache read performance.
 - ❑ If cache miss rate is high, then data read performance is close to memory read performance.



COST OF A CACHE MISS

It is important to minimize the cache miss rate!

COST OF A MEMORY MISS

Question:

Suppose data items are located in the **disk** and can be cached in **main memory** as well.

(We only consider **two layers**)

1. Accessing disk once incurs a cost of *cost_access_disk*;
2. Accessing main memory once incurs a cost of *cost_access_mem*;
3. Memory miss rate is h^* ($0 \leq h^* \leq 1$).

How to estimate the cost of reading the data Item?



COST OF A MEMORY MISS

- ❑ If memory hit: $\text{cost_access_mem} \times (1-h^*)$
- ❑ If memory miss: $(\text{cost_access_disk} + \text{cost_access_mem}) \times h^*$
- ❑ Overall: $\text{cost_access_disk_overall}$
 $= \text{cost_access_mem} \times (1-h^*) + (\text{cost_access_disk} + \text{cost_access_mem}) \times h^*$

COST OF A MEMORY MISS

- ❑ $\text{cost_access_disk_overall} = \text{cost_access_mem} \times (1-h^*) + (\text{cost_access_disk} + \text{cost_access_mem}) \times h^*$

- ❑ If $h^*=1$ (high miss rate):
$$\begin{aligned}\text{cost_access_disk_overall} \\ = \text{cost_access_disk} + \text{cost_access_mem} \approx \text{cost_access_disk}\end{aligned}$$

The above step is due to the fact that cost_access_disk significantly larger than (e.g., 1000x) cost_access_mem

- ❑ If $h^*=0$: $\text{cost_access_disk_overall} = \text{cost_access_mem}$

COST OF A MEMORY MISS

- $\text{cost_access_disk_overall} = \text{cost_access_mem} \times (1-h^*) + (\text{cost_access_disk} + \text{cost_access_mem}) \times h^*$
- Summary
 - If $h^*=1$ (high miss rate), then the read performance is close to the read performance of disk
 - If $h^*=0$ (low miss rate), then the read performance is close to the read performance of memory.



What about considering more than two layers?

We will see this in tutorial questions.

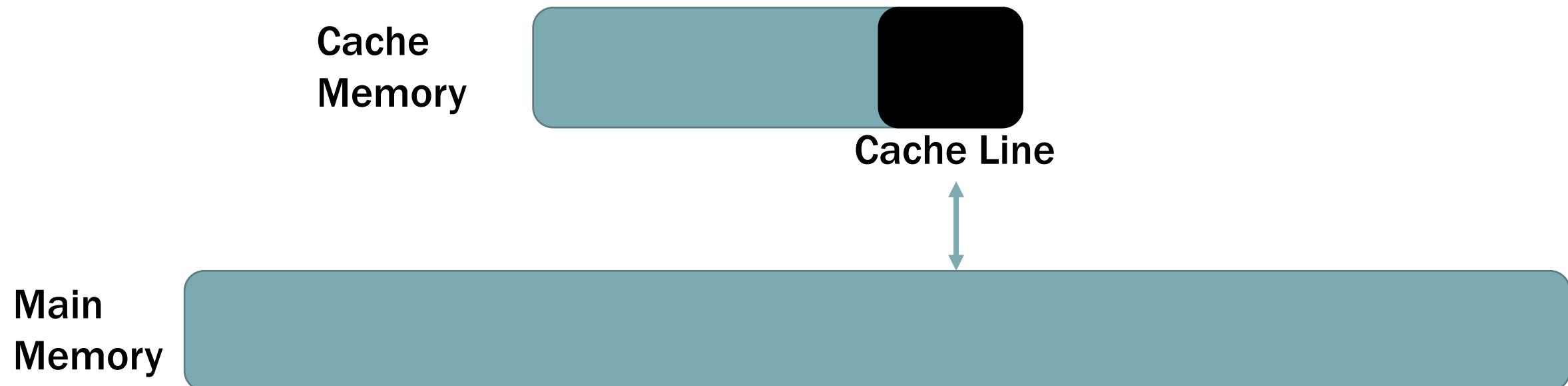
How to obtain a smaller cache miss rate?

We will see this in the coming lectures about cache conscious designs.

Data in Memory Hierarchy: Page-based Access

SMALLEST UNIT TRANSFERRED BETWEEN TWO LAYERS

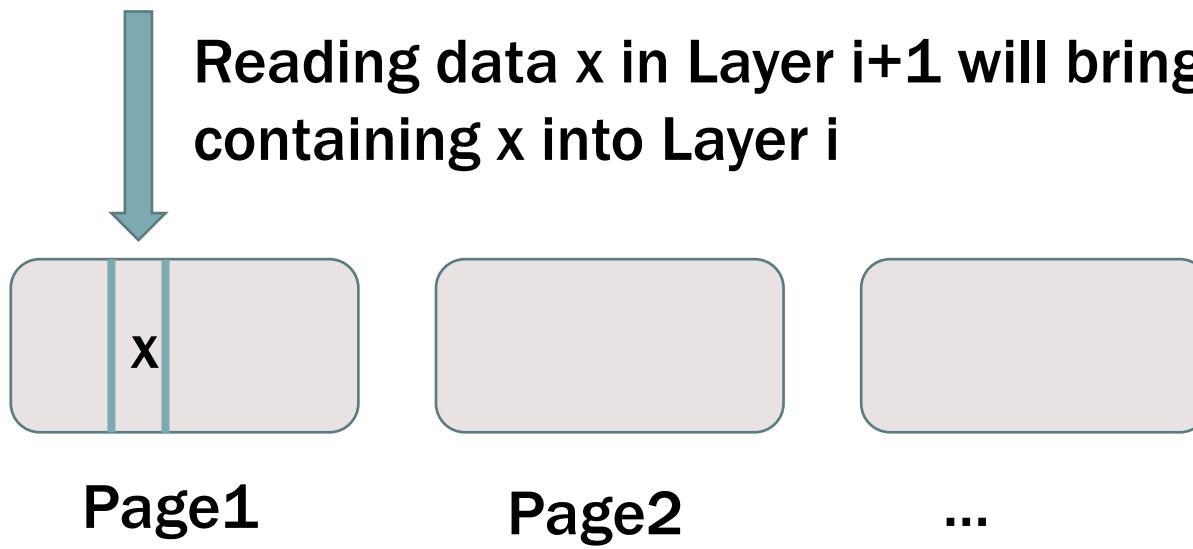
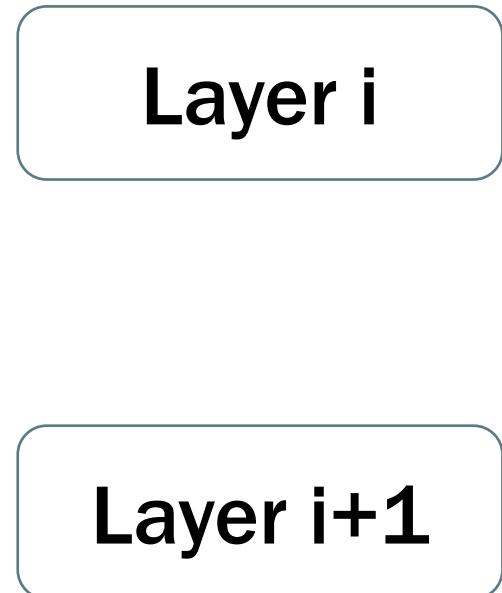
- ❑ The data unit swapped between cache memory and main memory is called **cache line**
 - ❑ Cache line size is usually 32, 64 and 128 bytes
 - ❑ A cache line contains continuous memory segment



SMALLEST UNIT TRANSFERRED BETWEEN TWO LAYERS

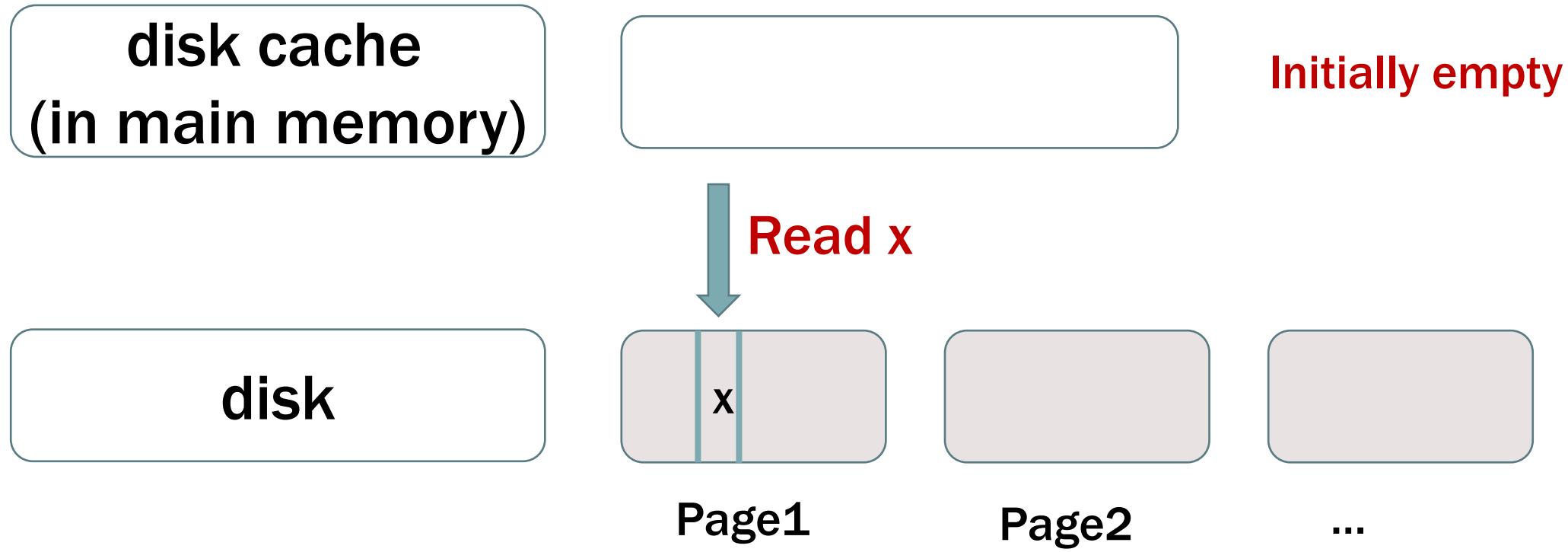
- The data unit swapped between cache memory and main memory is called **cache line**
 - Cache line size is usually 32, 64 and 128 bytes
 - A cache line contains continuous memory segment
- The data unit swapped between main memory and disk is called (memory) **page**.
 - Page size is usually about 4KB
 - A page contains continuous memory segment
- We use “**page**” as a common term to refer to the transfer data unit between Layer i and Layer i+1.

EXAMPLE OF PAGE-BASED ACCESS

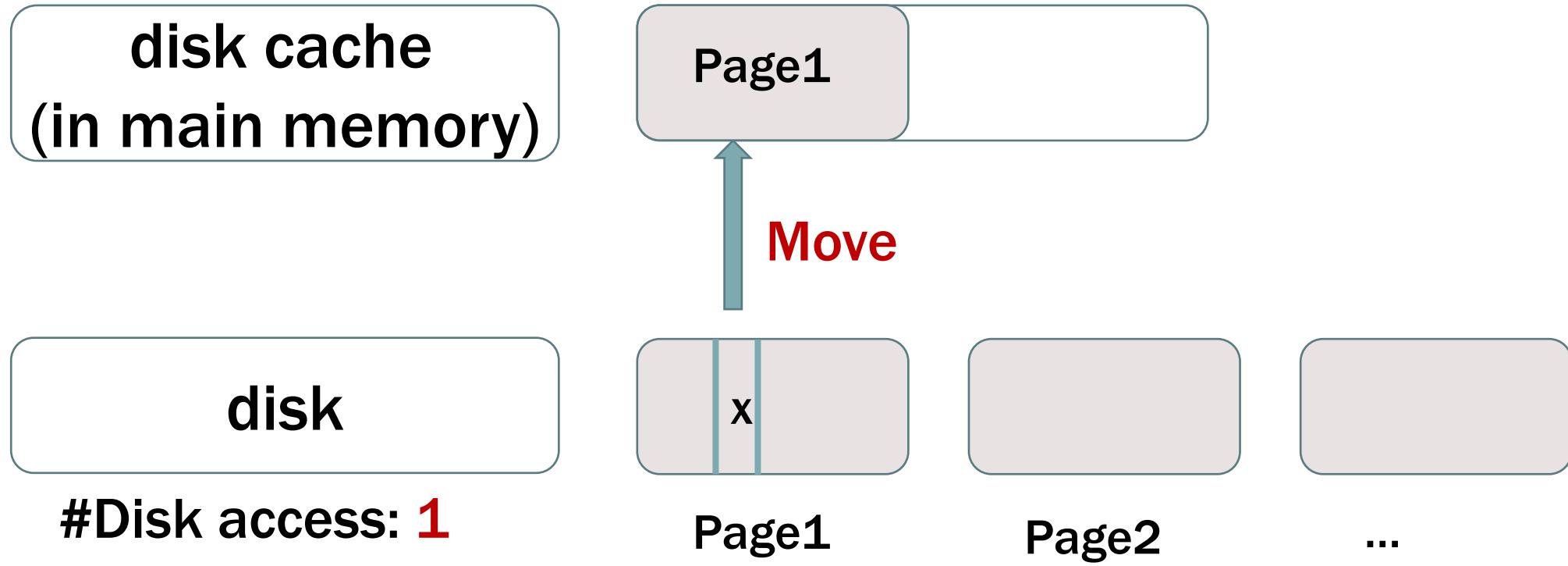


Reading data x in Layer $i+1$ will bring the whole page containing x into Layer i

EXAMPLE OF PAGE-BASED ACCESS



EXAMPLE OF PAGE-BASED ACCESS



EXAMPLE OF PAGE-BASED ACCESS

disk cache
(in main memory)

Page1

disk

x

Page1

Read

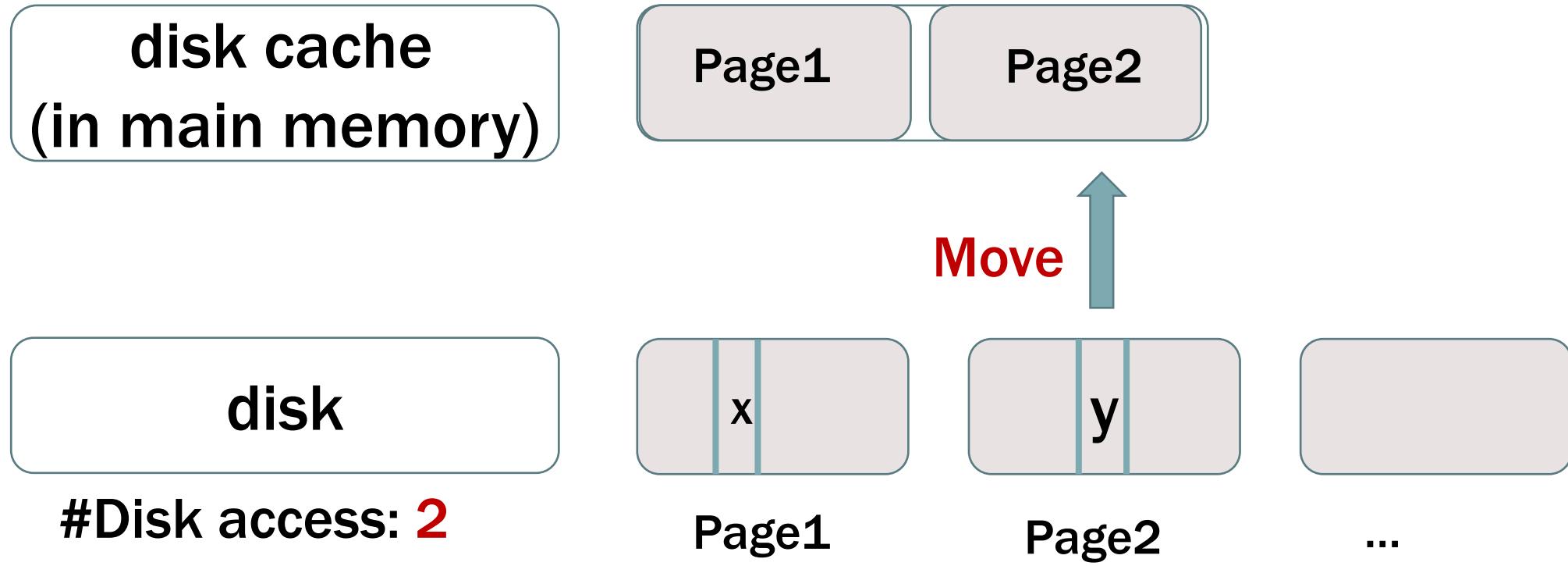
y

Page2

...

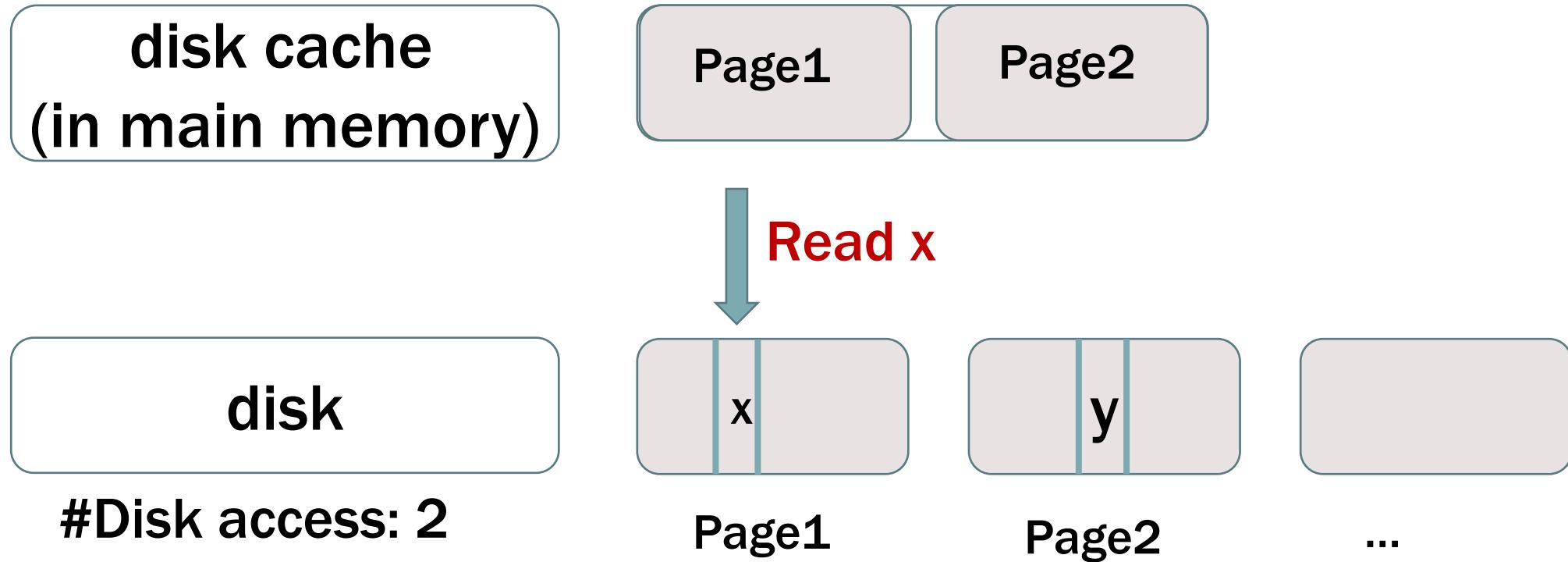
#Disk access: 1

EXAMPLE OF PAGE-BASED ACCESS



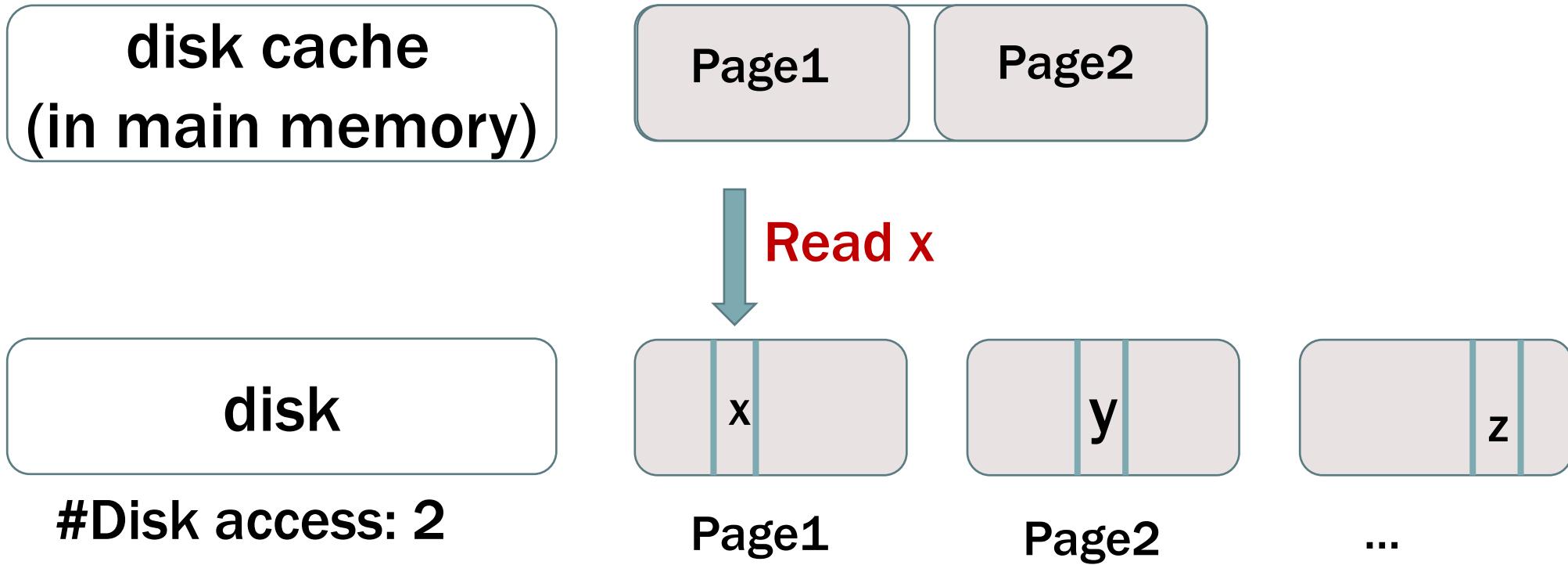
EXAMPLE OF PAGE-BASED ACCESS

Now, if we further read x, no additional disk accesses because it is in main memory already.



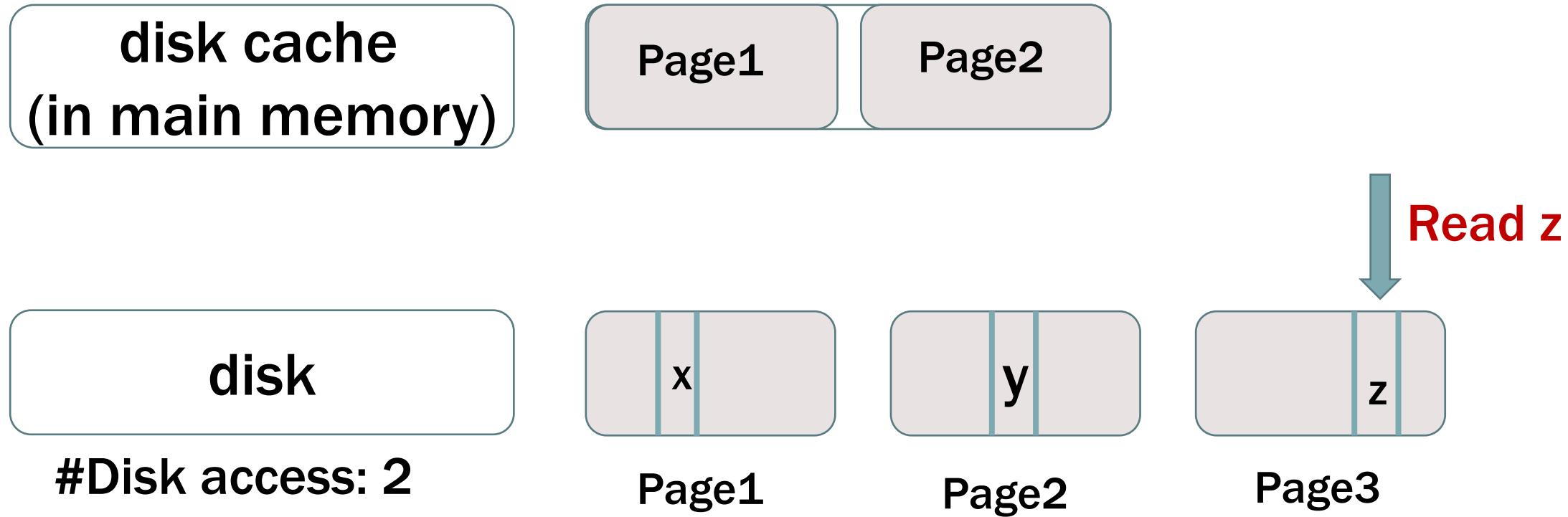
EXAMPLE OF PAGE-BASED ACCESS

Now, what if we read the third page?



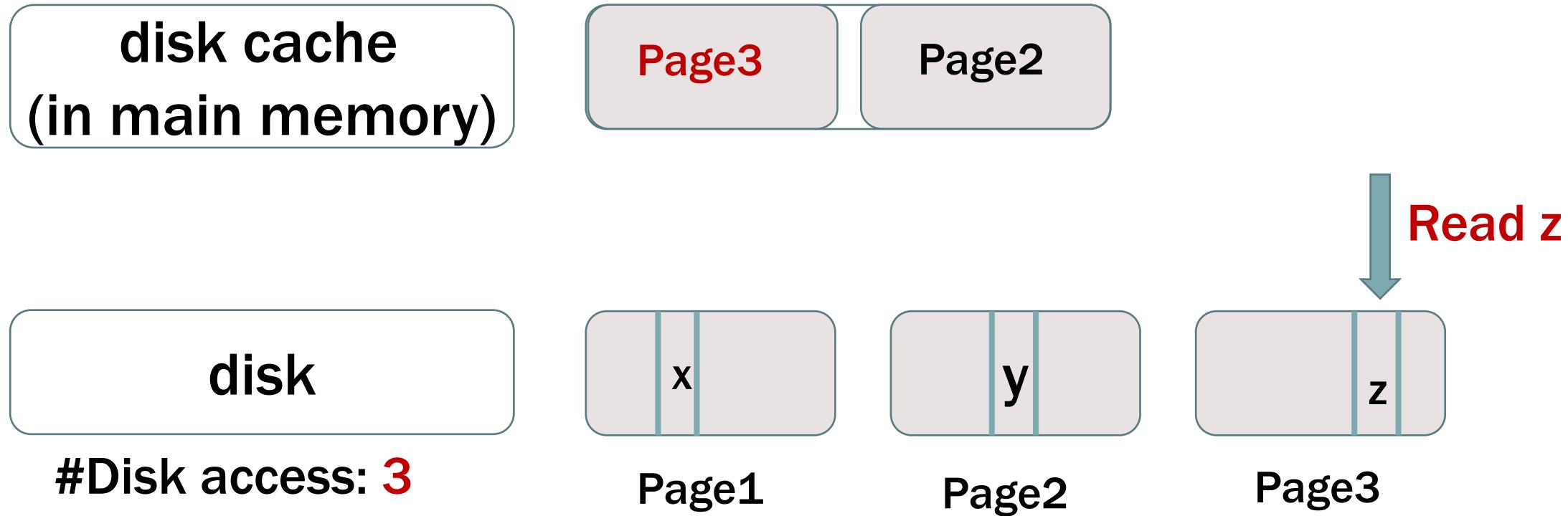
EXAMPLE OF PAGE-BASED ACCESS

Now, what if we read the third page? Now disk cache is full.



EXAMPLE OF PAGE-BASED ACCESS

Now, what if we read the third page? Now disk cache is full.



EXAMPLE: SCANNING ARRAYS

Query $x < 4$ from the following data

(size=120 bytes)

Memory Layer i

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

EXAMPLE: SCANNING ARRAYS

Cost: **40 Bytes**

(size=120 bytes)

Memory Layer i

Query $x < 4$ from the following data

Scan

5, 10, 7, 4, 12

Qualified results

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

EXAMPLE: SCANNING ARRAYS

Cost: **80 Bytes**

(size=120 bytes)
Memory Layer i

Query $x < 4$ from the following data

Scan

5, 10, 7, 4, 12

2, 8, 9, 11, 7

2

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

EXAMPLE: SCANNING ARRAYS

Cost: **120 Bytes**

(size=120 bytes)

Memory Layer i

Query $x < 4$ from the following data
Scan

7, 11, 3, 9, 8

2, 8, 9, 11, 7

2, 3

Qualified results

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

CAN WE DO BETTER ABOUT SCANNING?

- ❑ Consider a cost-free magic function for the example query:

By accessing a page, the function immediately tells you the positions of qualified keys (i.e., $x < 4$) within the page.

- ❑ Consider another cost-free magic function:

The function tells you **which pages** will contain the qualified keys (i.e., $x < 4$).

- ❖ Can we do better with such a function?

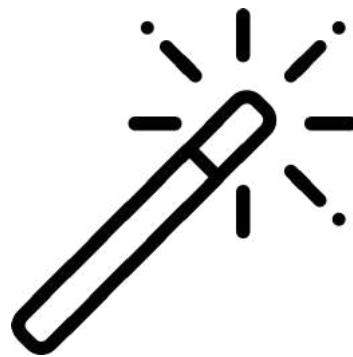
- ❖ Which one do you think is more useful?



TRY THE FIRST MAGIC FUNCTION

- Consider a cost-free magic function for the example query:

By accessing a page, the function immediately tells you the positions of qualified keys (i.e., $x < 4$) within the page.



WITH MAGIC FUNCTION (FIRST)

Query $x < 4$ from the following data

(size=120 bytes)

Memory Layer i

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 40 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)

Memory Layer i

Memory Layer $i+1$

Magic function read



5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 40 Bytes

(size=120 bytes)
Memory Layer i

Query $x < 4$ from the following data

Qualified results

5, 10, 7, 4, 12



Memory Layer $i+1$

Magic function read

5, 10, 7, 4, 12

bring

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 80 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

5, 10, 7, 4, 12

2

Memory Layer $i+1$

Magic function read

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 80 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)
Memory Layer i

5, 10, 7, 4, 12

2, 8, 9, 11, 7

2

Memory Layer $i+1$

Magic function read

bring

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 120 Bytes

(size=120 bytes)
Memory Layer i

Query $x < 4$ from the following data

Qualified results

7, 11, 3, 9, 8

2, 8, 9, 11, 7

2, 3

Memory Layer $i+1$

Magic function read

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 120 Bytes

(size=120 bytes)
Memory Layer i

Query $x < 4$ from the following data

7, 11, 3, 9, 8

2, 8, 9, 11, 7

2, 3

Qualified results

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

bring

Magic function read

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

CAN WE DO BETTER ABOUT SCANNING?

- Consider a cost-free magic function regarding the example query:

By accessing a page, the function immediately tells you the positions of qualified keys (i.e., $x < 4$) within the page.

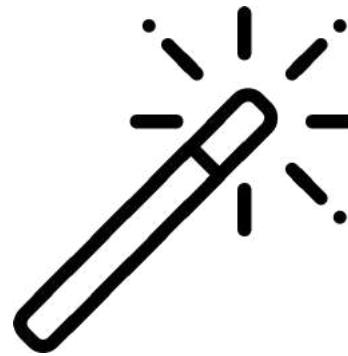
This seemingly good function does not help much. The reason is that the data movement is based on pages.

TRY THE SECOND MAGIC FUNCTION

- Consider another cost-free magic function:

The function tells you **which pages** will contain the qualified keys (i.e., $x < 4$).
Can we do better with such a function?

In some situations, **yes!**



WITH MAGIC FUNCTION (SECOND)

Query $x < 4$ from the following data

(size=120 bytes)

Memory Layer i

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

WITH MAGIC FUNCTION (SECOND)

Cost: 40 Bytes

Query $x < 4$ from the following data

Qualified results

(size=120 bytes)

Memory Layer i

Memory Layer $i+1$

Each integer: 8 bytes

5, 10, 7, 4, 12

Page 1

Magic function read Page 2

2, 8, 9, 11, 7

Page 2

7, 11, 3, 9, 8

Page 3

Each page: 8 bytes \times 5 = 40 bytes

WITH MAGIC FUNCTION (SECOND)

Cost: 40 Bytes

(size=120 bytes)

Memory Layer i

Query $x < 4$ from the following data

Qualified results

2, 8, 9, 11, 7

2

bring

Memory Layer $i+1$

Magic function read Page 2

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Page 1

Page 2

Page 3

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

WITH MAGIC FUNCTION (SECOND)

Cost: 80 Bytes

(size=120 bytes)
Memory Layer i

Query $x < 4$ from the following data

2, 8, 9, 11, 7

7, 11, 3, 9, 8

2, 3

Qualified results

bring

Memory Layer $i+1$

Magic function read Page 3

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Page 1

Page 2

Page 3

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

SUMMARY OF MAGIC FUNCTION EXERCISE

Take-away message

In general, if there is a magic function telling **which pages locate the qualified data**, and if these pages are only a subset of all pages, the read cost will be smaller than directly scanning all pages.

- In practice, we do not have such a magic function with a free cost. Typically, we design indexes, and put the indexes to a faster memory-layer so that its cost is minor compared with the slower memory-layer.

BIG DATA MANAGEMENT

CZ4123

MEMORY HIERARCHY

(PART III)

CACHE CONSCIOUS DATA-PROCESSING

Siqiang Luo

Assistant Professor

OVERVIEW

- ❑ In previous lectures, we show the basic concepts and analysis of memory hierarchy.
- ❑ In this lecture, we show some design principles to make use of memory hierarchy for processing big data. We will discuss following two cases.
 - ❑ Array access patterns
 - ❑ Spatial locality designs
 - ❑ Temporal locality designs
 - ❑ Big data sorting

MOTIVATING QUESTION

Suppose we have a 10000x10000 matrix of integers, which is stored in a 2-dimentional array A[10000][10000]. We want to set all of its value to 1. **Cache size = 5000 integers.** How would you write your code?

Solution X

```
for (int i=0;i<10000;i++)  
    for(int j=0;j<10000;j++)  
        A[i][j]=1;
```

Solution Y

```
for (int j=0;j<10000;j++)  
    for(int i=0;i<10000;i++)  
        A[i][j]=1;
```

Which one is better, X or Y?



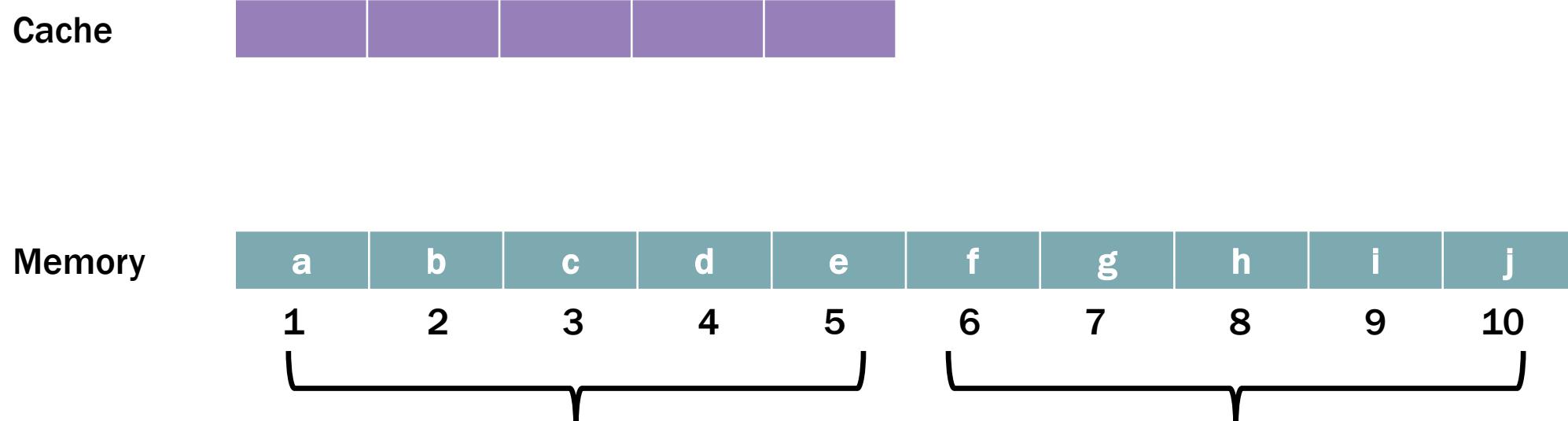
ACCESSING AN ARRAY

- We may access an array with different **access patterns**
 - Access pattern means the position sequence of accessing an array.
 - Different access patterns may impact the utility of cache

ACCESS PATTERN 1

We have a 10-integer array stored in main memory,
cache size = 5 integers, transfer size (cache line)= 5 integers

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10



ACCESS PATTERN 1

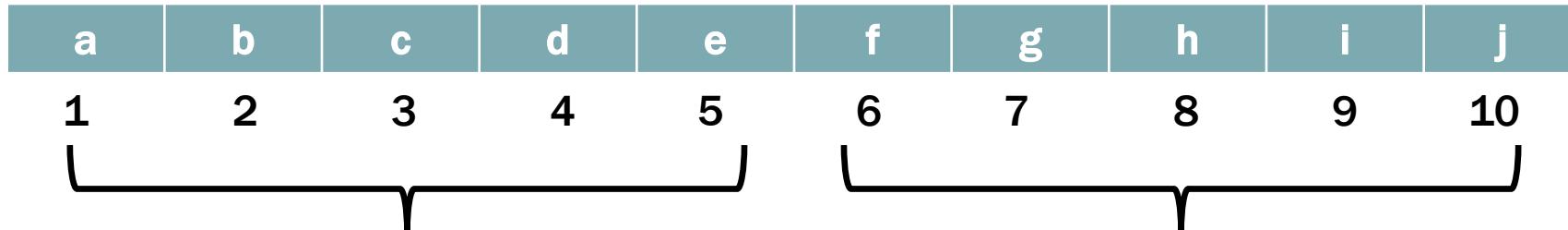
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: **1**
Cache Hit: 0

Memory



ACCESS PATTERN 1

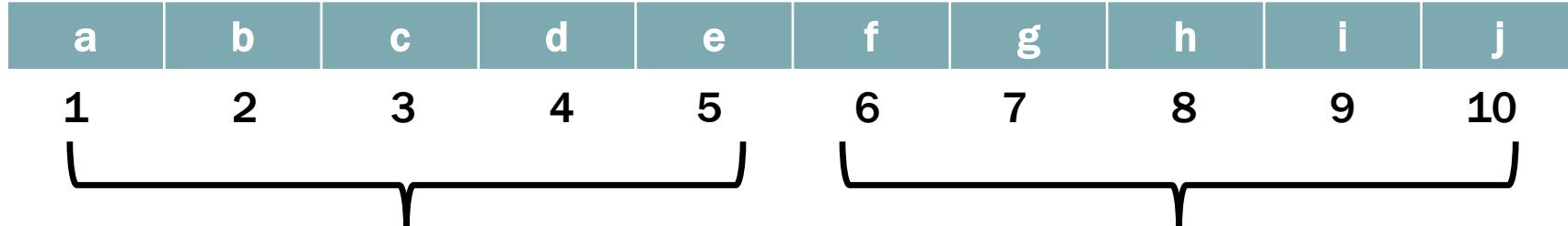
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: 1
Cache Hit: **1**

Memory



ACCESS PATTERN 1

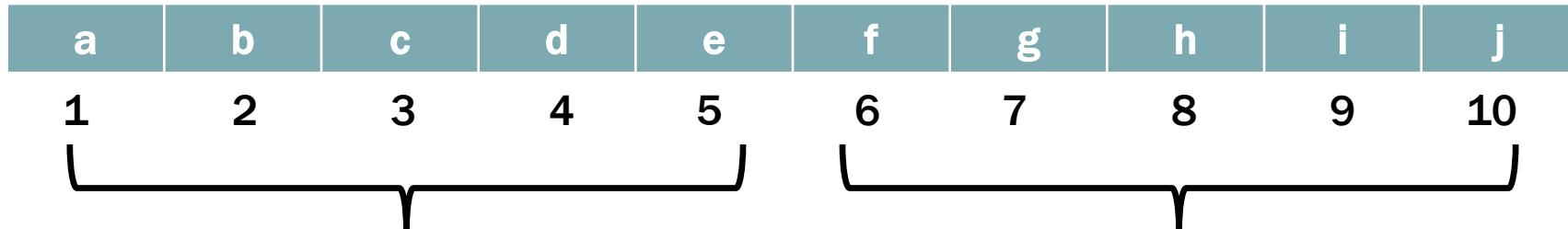
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: 1
Cache Hit: 2

Memory



ACCESS PATTERN 1

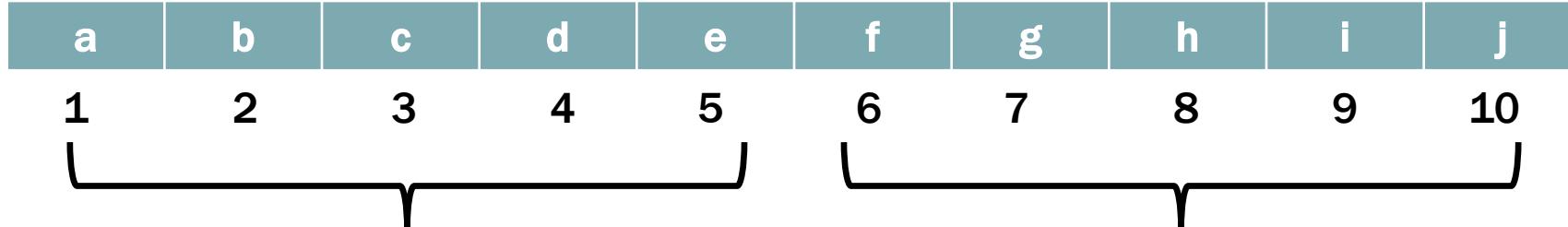
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: 1
Cache Hit: 3

Memory



ACCESS PATTERN 1

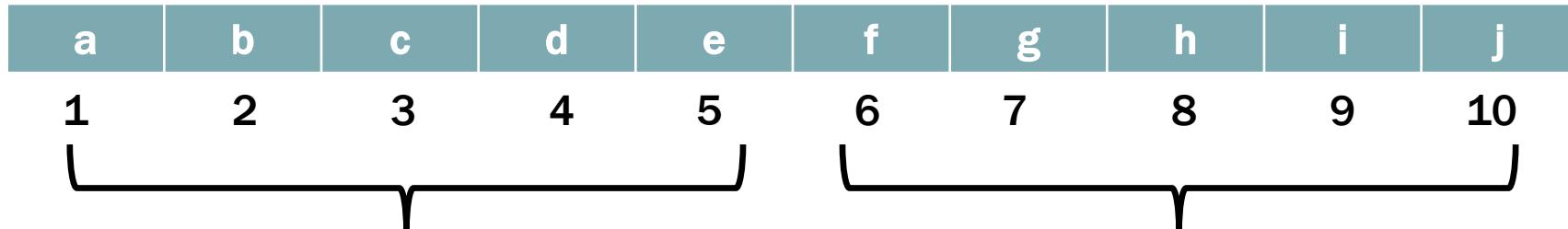
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



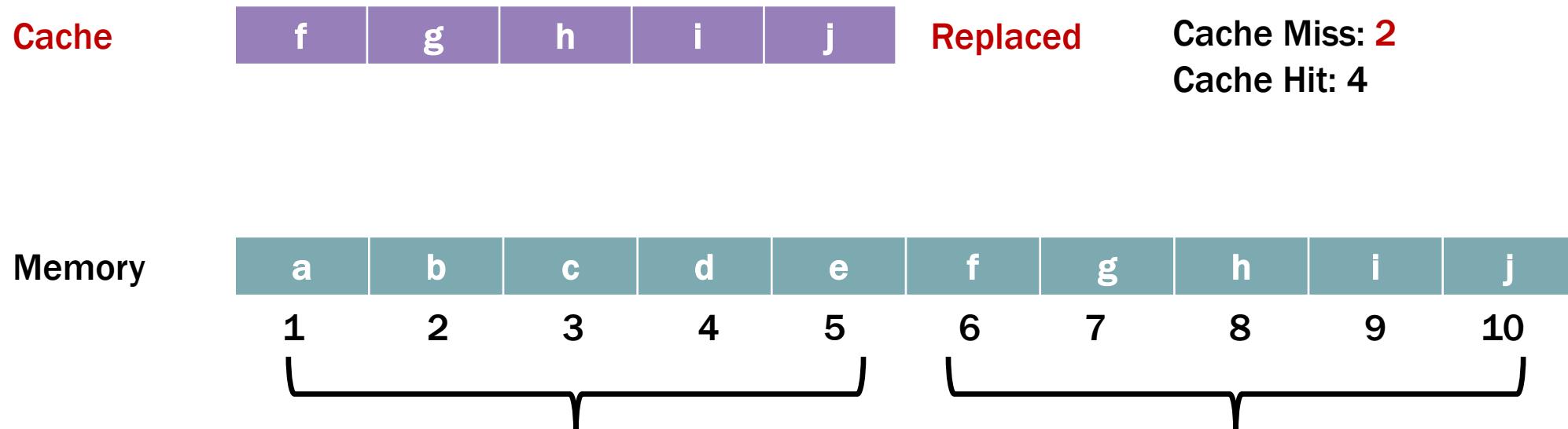
Cache Miss: 1
Cache Hit: 4

Memory



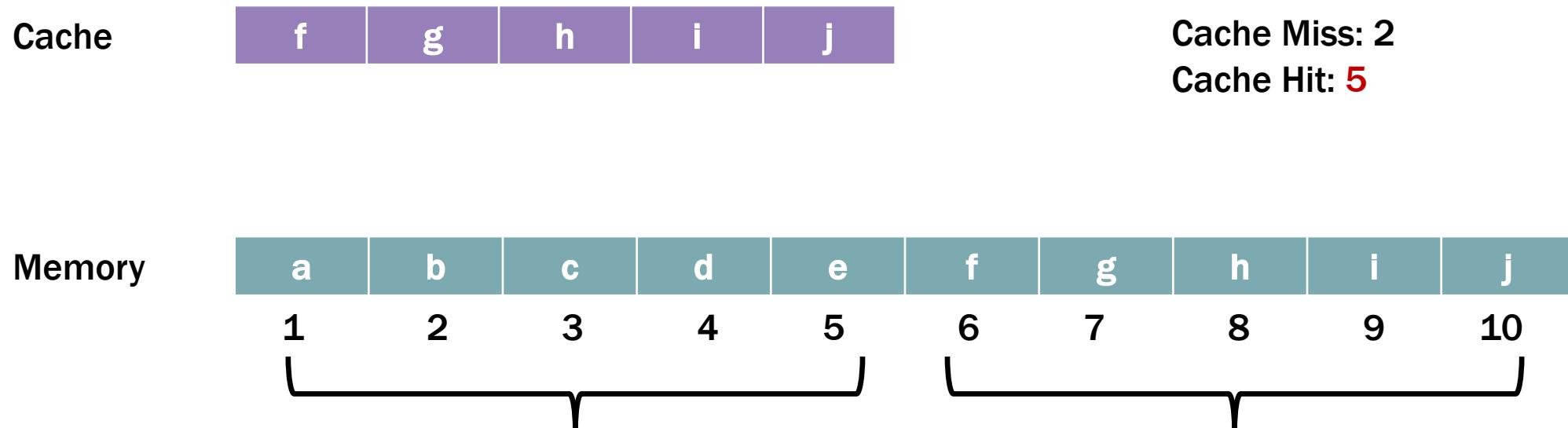
ACCESS PATTERN 1

Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**



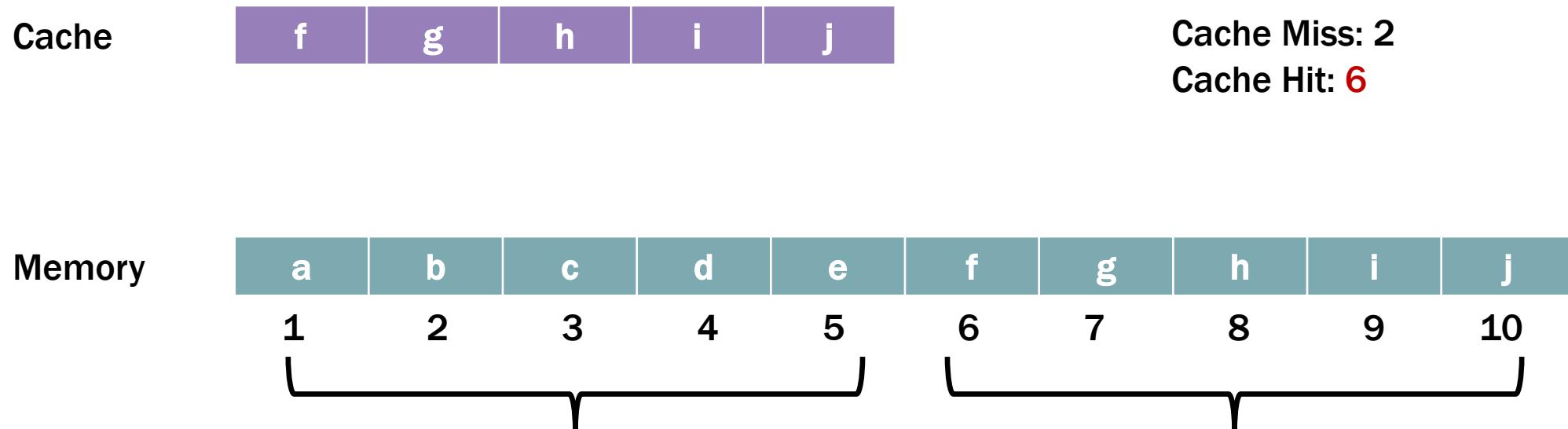
ACCESS PATTERN 1

Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**



ACCESS PATTERN 1

Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**



ACCESS PATTERN 1

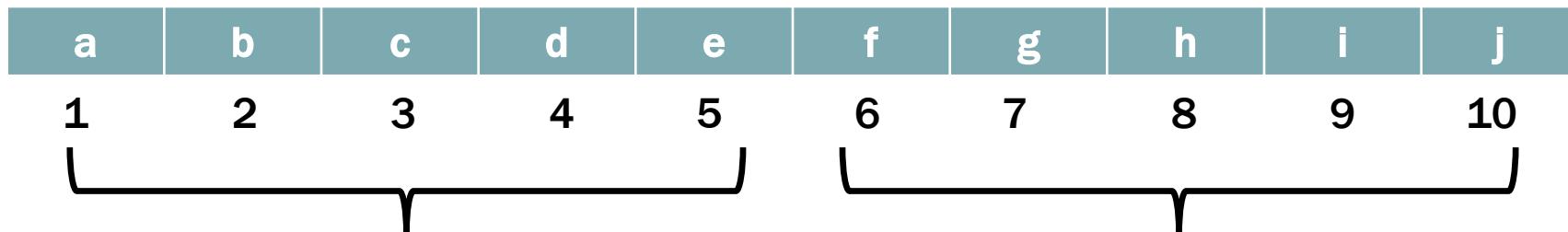
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: 2
Cache Hit: 7

Memory



ACCESS PATTERN 1

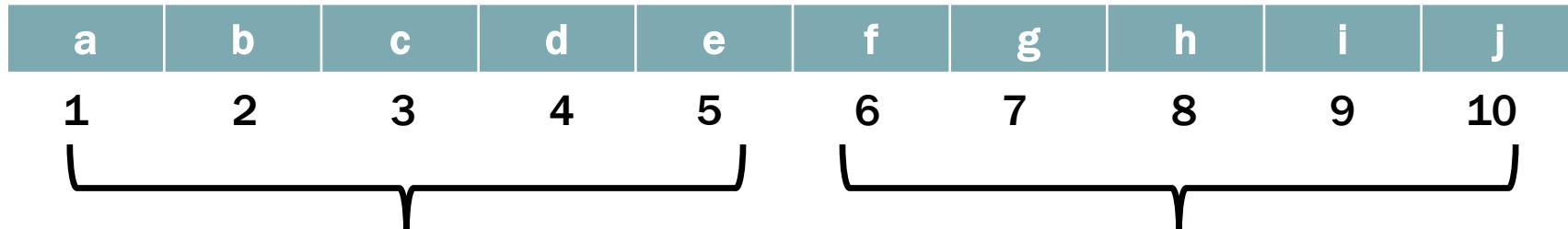
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: 2
Cache Hit: 8

Memory



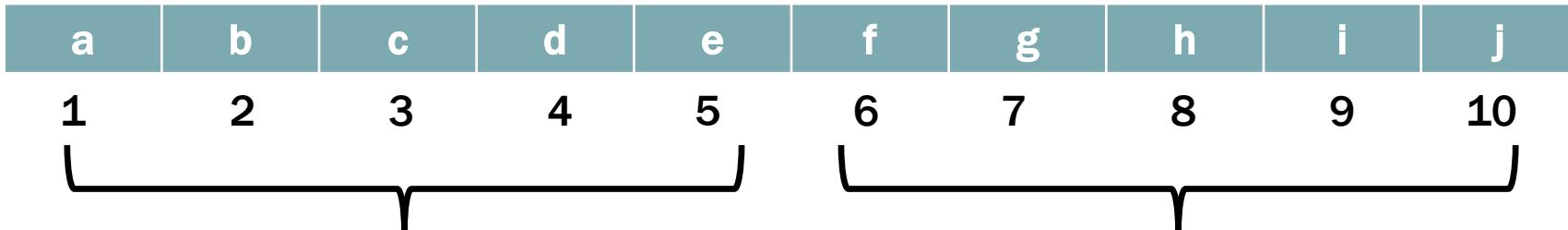
ACCESS PATTERN 2

Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10

Cache



Memory



ACCESS PATTERN 2

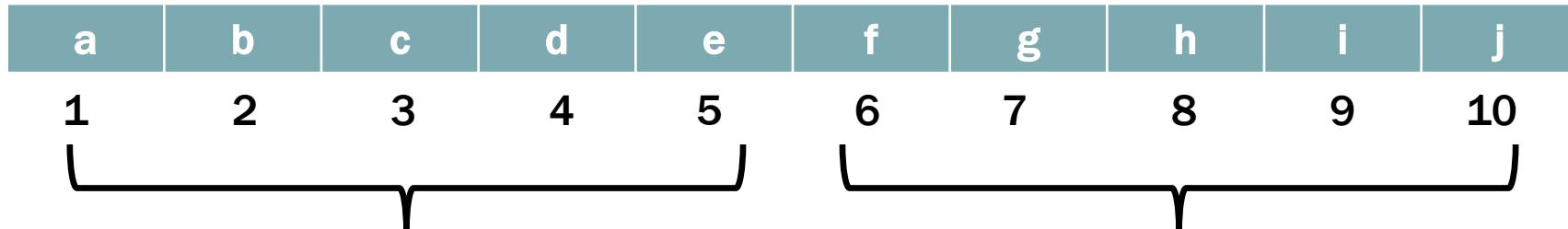
Access pattern: **1, 6, 2, 7, 3, 8, 4, 9, 5, 10**

Cache



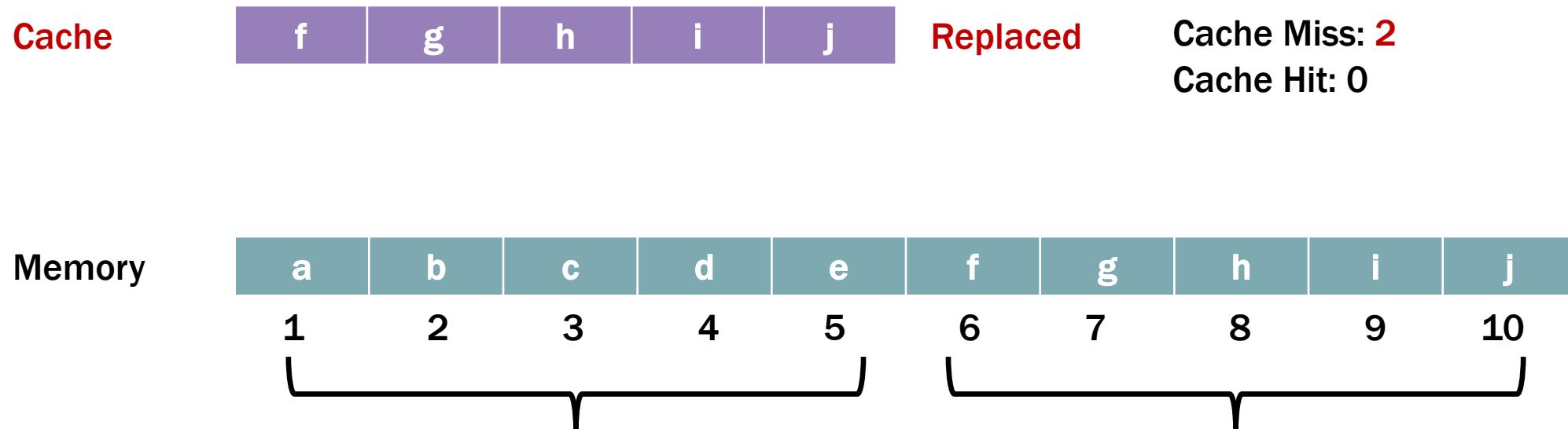
Cache Miss: **1**
Cache Hit: 0

Memory



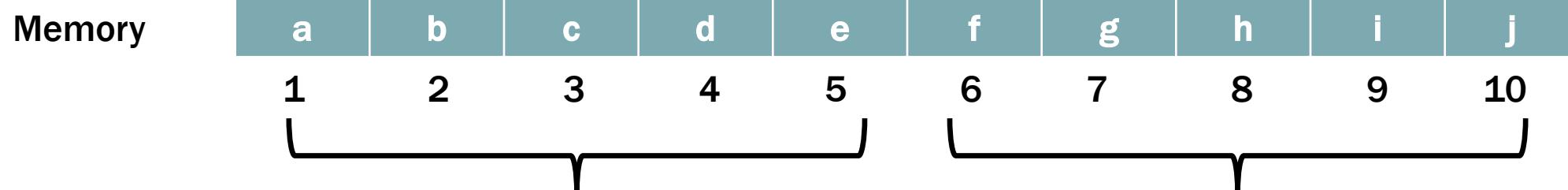
ACCESS PATTERN 2

Access pattern: **1, 6, 2, 7, 3, 8, 4, 9, 5, 10**



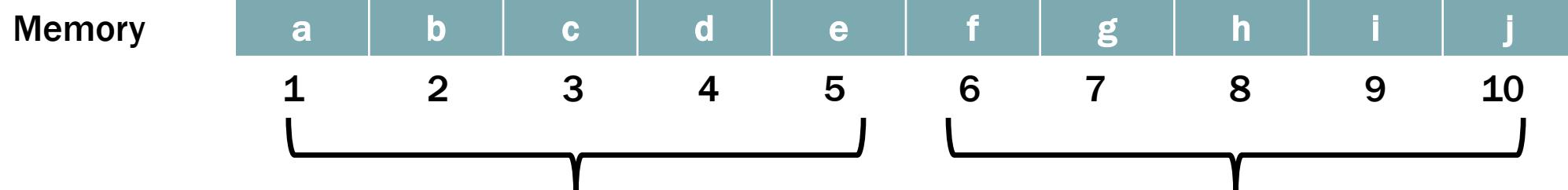
ACCESS PATTERN 2

Access pattern: **1, 6, 2, 7, 3, 8, 4, 9, 5, 10**



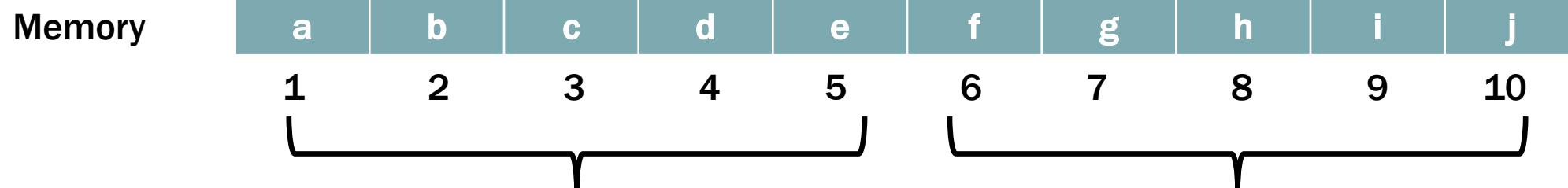
ACCESS PATTERN 2

Access pattern: **1, 6, 2, 7, 3, 8, 4, 9, 5, 10**



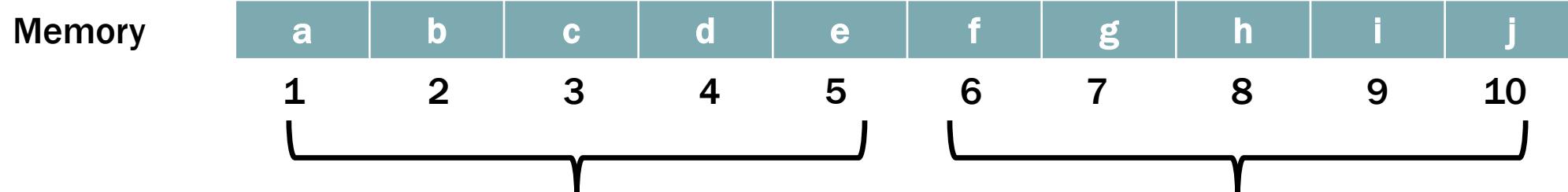
ACCESS PATTERN 2

Access pattern: **1, 6, 2, 7, 3, 8, 4, 9, 5, 10**



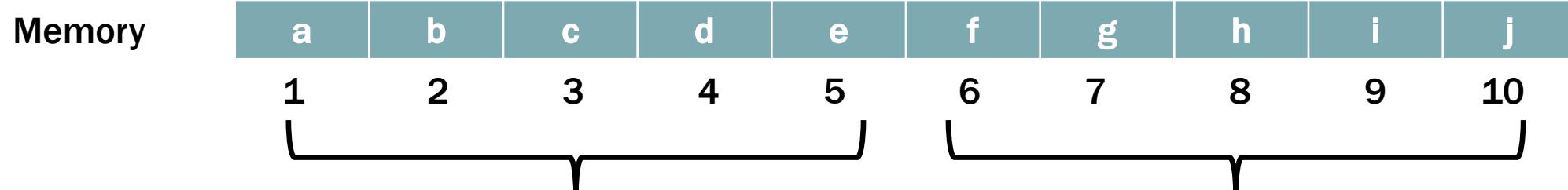
ACCESS PATTERN 2

Access pattern: **1, 6, 2, 7, 3, 8, 4, 9, 5, 10**



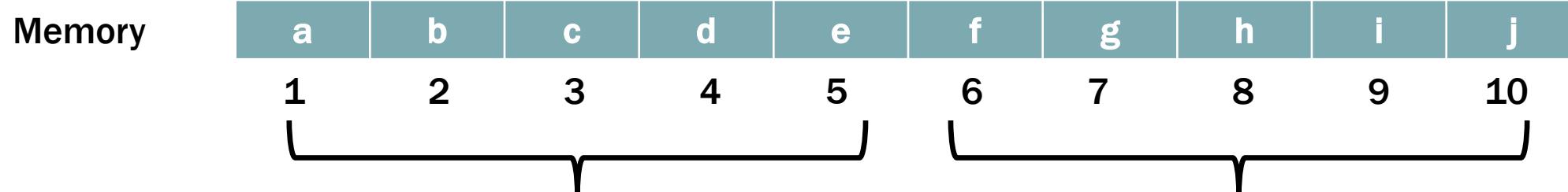
ACCESS PATTERN 2

Access pattern: **1, 6, 2, 7, 3, 8, 4, 9, 5, 10**



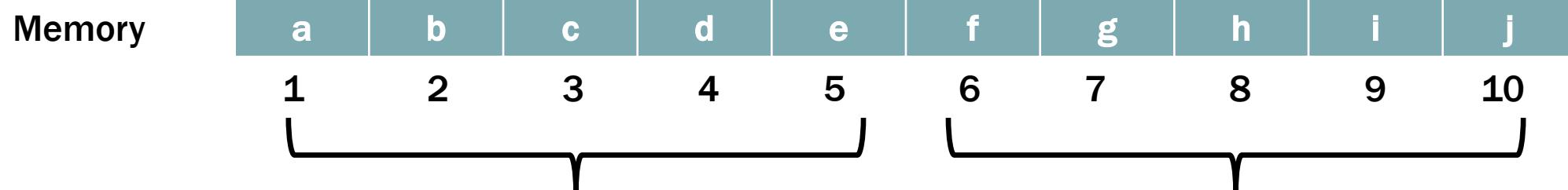
ACCESS PATTERN 2

Access pattern: **1, 6, 2, 7, 3, 8, 4, 9, 5, 10**



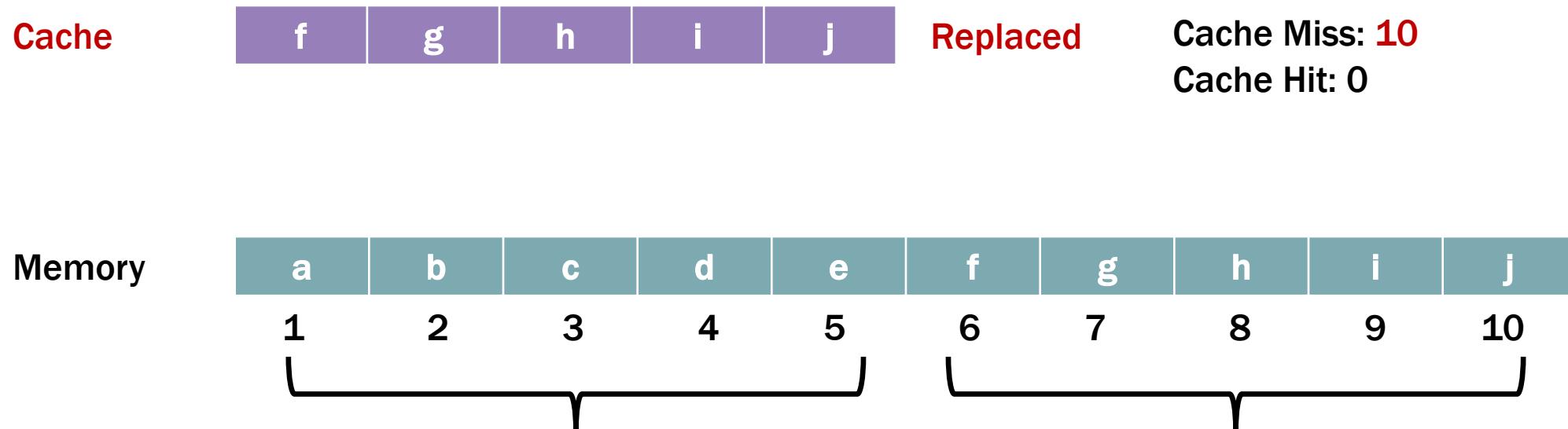
ACCESS PATTERN 2

Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10



ACCESS PATTERN 2

Access pattern: 1, 6, 2, 7, 3, 8, 4, 9, 5, 10



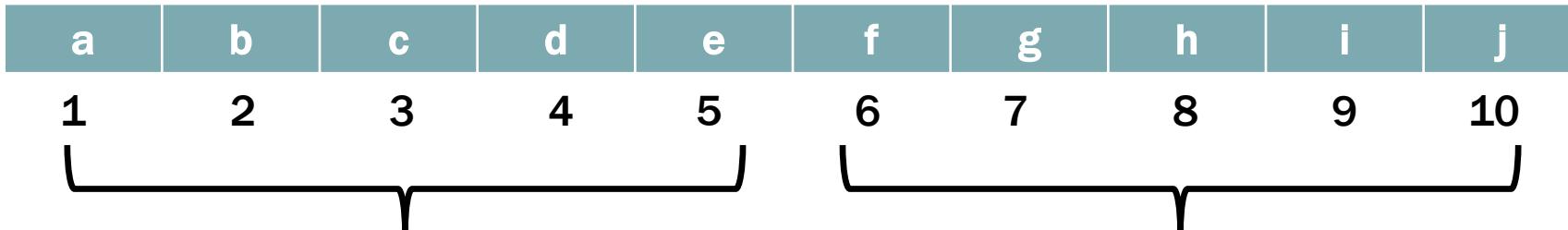
ACCESS PATTERN 3

Access pattern: 1, 6, 8, 7, 7, 8, 9, 9, 6, 10

Cache



Memory



ACCESS PATTERN 3

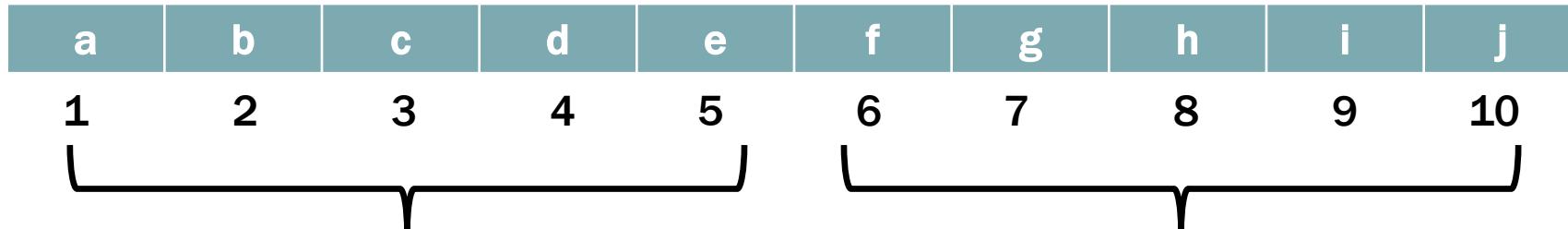
Access pattern: 1, 6, 8, 7, 7, 8, 9, 9, 6, 10

Cache



Cache Miss: 2
Cache Hit: 8

Memory



SUMMARY

- Access pattern with spatial locality (pattern 1) or temporal locality (pattern 3) is cache friendly.
- The adversarial access (pattern 2) may cause the worst case read performance
- When data is big, the impact is very significant
- Whenever possible, we store the data in a way that the access pattern has certain locality.

2D ARRAYS

Suppose we have a 10000×10000 matrix of integers, which is stored in a 2-dimensional array $A[10000][10000]$. We want to set all of its value to 1. **Cache size = 5000 integers.** How would you write your code?

Solution X

```
for (int i=0;i<10000;i++)  
    for(int j=0;j<10000;j++)  
        A[i][j]=1;
```

Solution Y

```
for (int j=0;j<10000;j++)  
    for(int i=0;i<10000;i++)  
        A[i][j]=1;
```

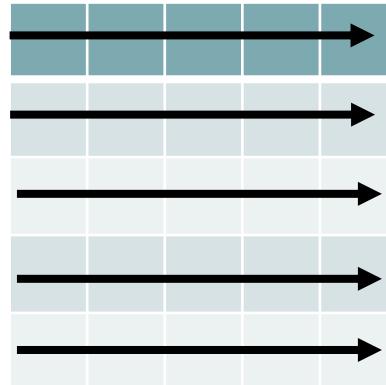
Which one is better, X or Y?



2D ARRAYS

Solution X

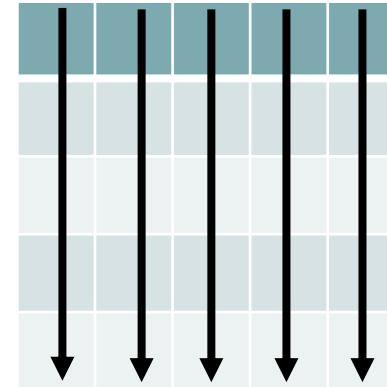
```
for (int i=0;i<10000;i++)  
    for(int j=0;j<10000;j++)  
        A[i][j]=1;
```



Cache friendly

Solution Y

```
for (int j=0;j<10000;j++)  
    for(int i=0;i<10000;i++)  
        A[i][j]=1;
```



Always evict
cache

Which one is better, X or Y?



SORTING

In the big data era, it happens very often that the data volume is too big to hold in main memory.

Suppose we have 1 TB of integers stored in disk and we want to sort the array. My computer only has 16GB main memory, what should I do?



SORTING

In the big data era, it happens too often that the data is too big to be stored in main memory.

Suppose we have 1 TB of integers stored in disk and we want to sort the array. My computer only has 16GB main memory, what should I do?

We have learnt a lot of sorting algorithms, can we simply use them?



– We can't allocate enough large array in memory when coding.

SORTING

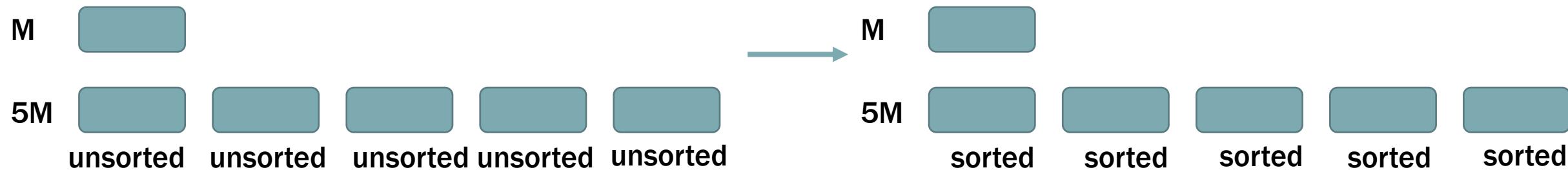
The problem of sorting a big array larger than the main memory is called **external memory sorting**.

Suppose the size of main memory is M , the array size is $5M$.



SORTING

Suppose the size of main memory is M , the array size is $5M$.

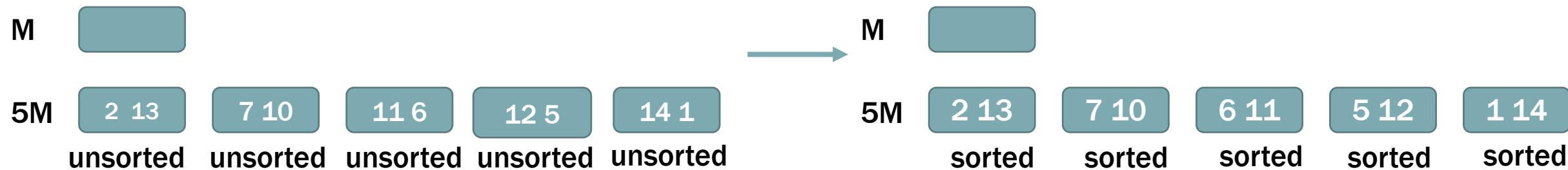


Step 1:

Cut $5M$ arrays into 5 parts. Each part is put in main memory to sort
(using any sorting algorithm we learnt, e.g., quick sort)

SORTING

Suppose the size of main memory is M , the array size is $5M$.



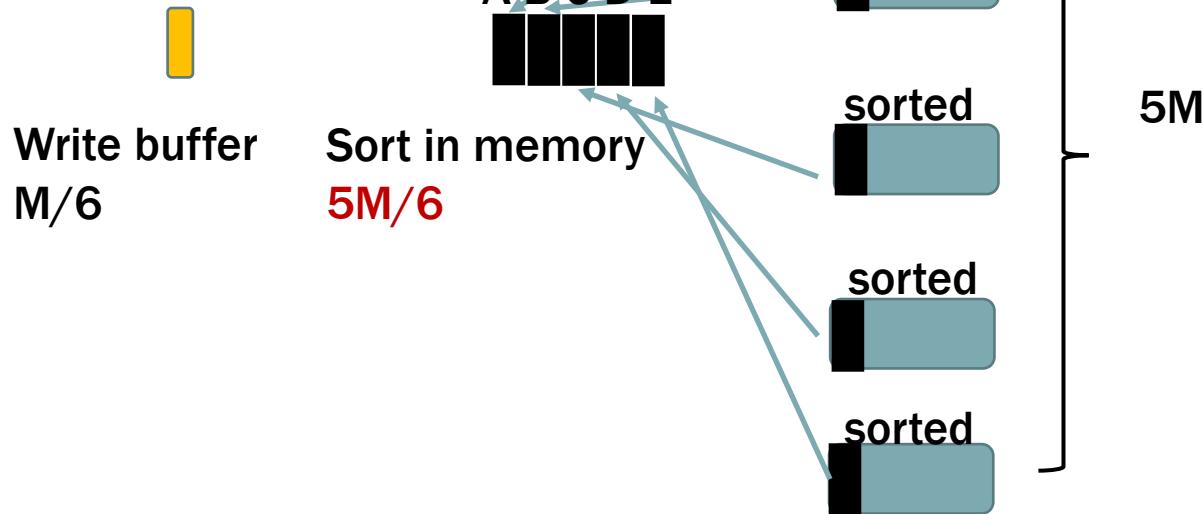
Step 1:

Cut $5M$ arrays into 5 parts. Each part is put in main memory to sort
(using any sorting algorithm we learnt, e.g., quick sort)

SORTING

or size $M/6$

$5M/6$ in total for in-memory sorting, remaining $M/6$ for writing the results



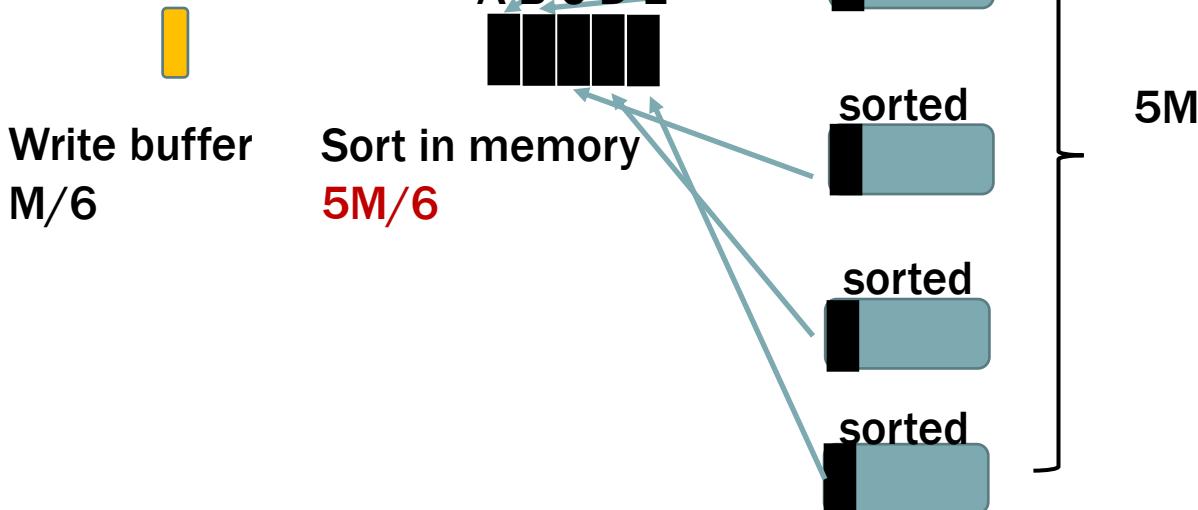
Step 2:

Apply 5-way merge sort to the first $M/6$ of each sorted part.

SORTING

or size $M/6$

5M/6 in total for in-memory sorting, remaining **M/6** for writing the results



Details of merge sort

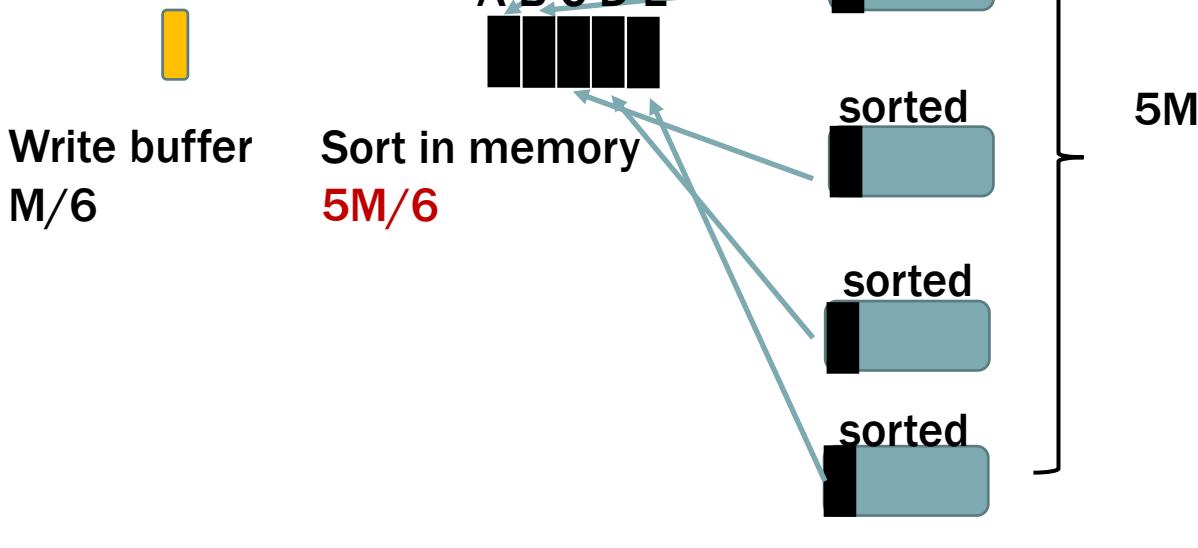
1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	2, 13
B	7, 10
C	6, 11
D	5, 12
E	1, 14

SORTING

■ or ■ size $M/6$

5M/6 in total for in-memory sorting, remaining **M/6** for writing the results



Step 3:

Whenever one of $\{A, B, C, D, E\}$ is empty, refill it from the source part.

Whenever the write-buffer is full, write it to the disk.

Details of merge sort

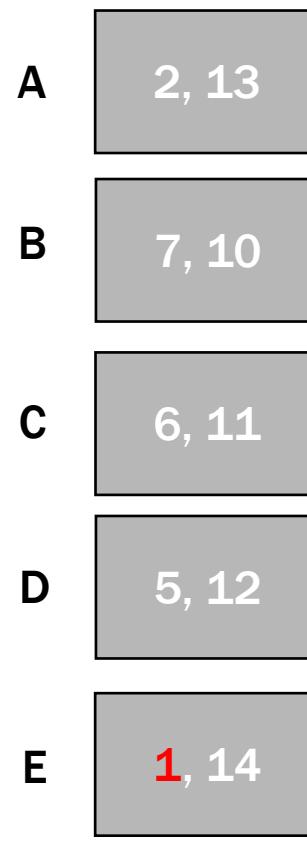
1. 5 iterators scanning each of $\{A, B, C, D, E\}$ from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	2, 13
B	7, 10
C	6, 11
D	5, 12
E	1, 14

DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;



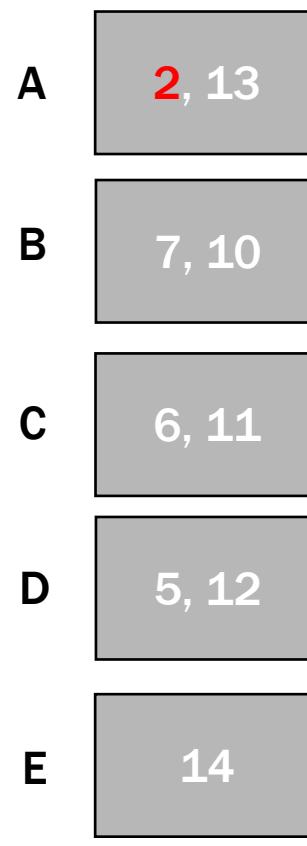
Write buffer (size 2)



DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;



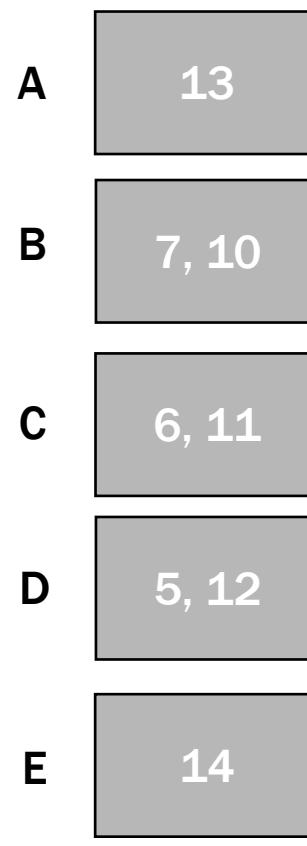
Write buffer



DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;



Write buffer



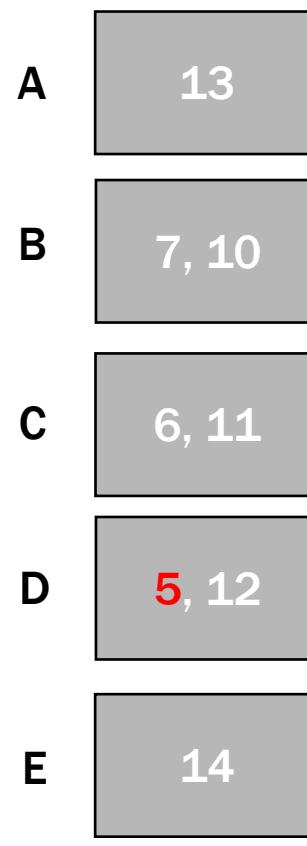
Write to disk



DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;



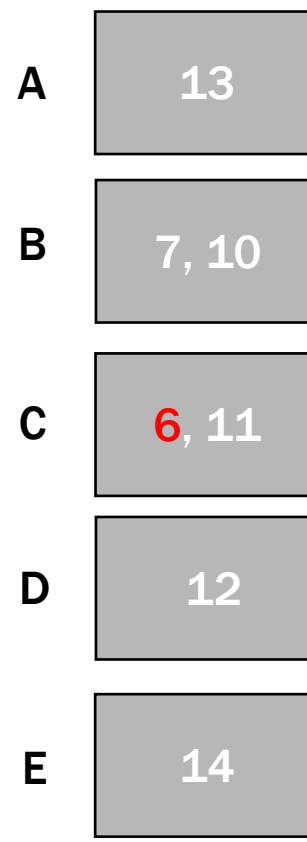
Write buffer



DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;



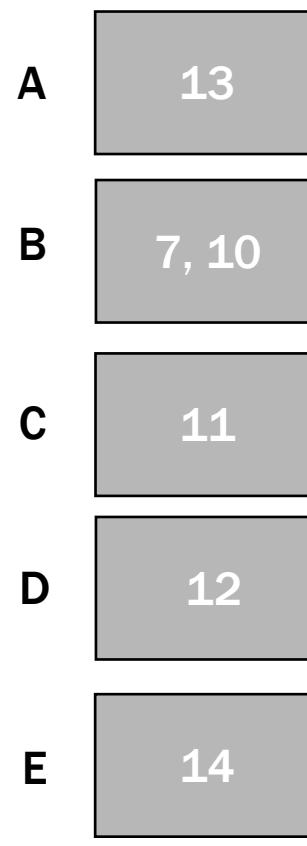
Write buffer



DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;



Write buffer



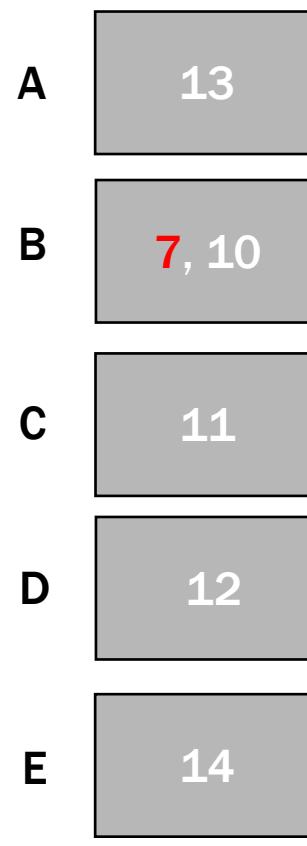
Write to disk



DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;



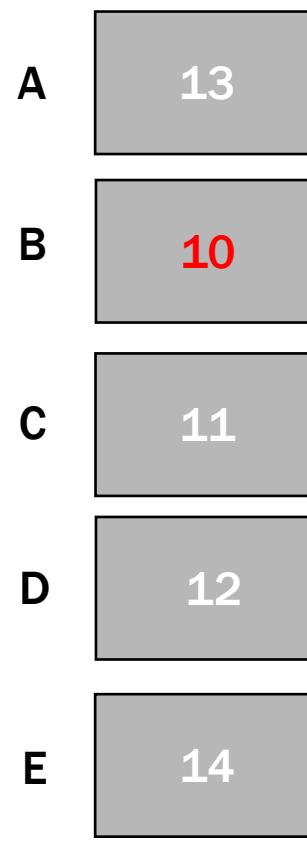
Write buffer



DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;



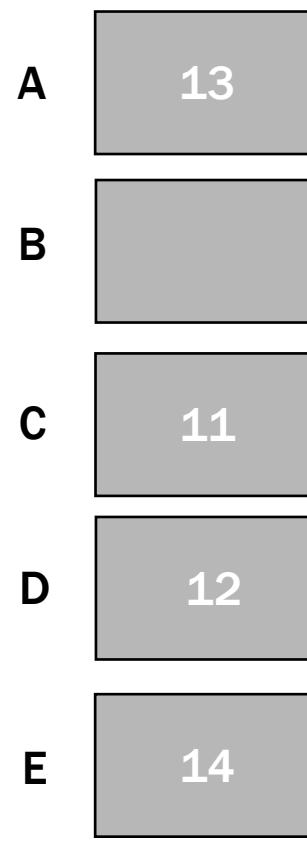
Write buffer



DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;



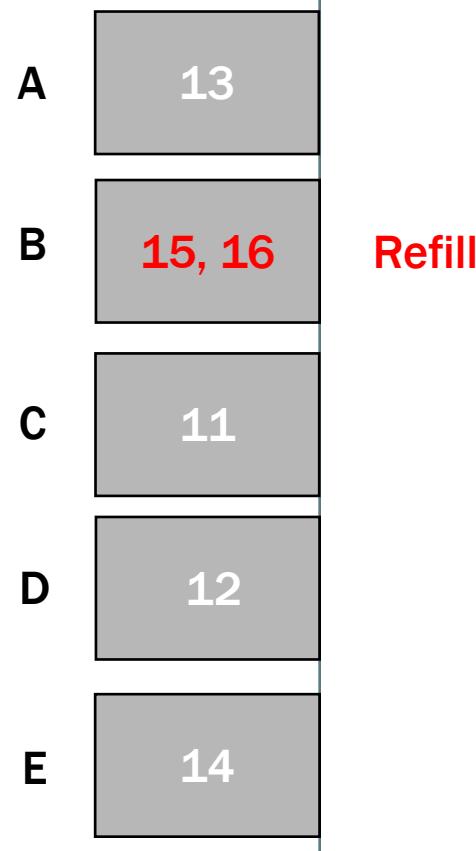
Write buffer

7, 10

DETAILED EXAMPLE FOR (STEP 2&3)

Details of merge sort

1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;



Write buffer



7, 10

Write to disk

What's the cost?



COST ANALYSIS

Let N be the data size, M be the memory size, and the B be the page (transfer unit) size.

- Step 1: Read all the N items once and write back to the disk once, incurring $O(N/B)$ I/Os.

COST ANALYSIS

Let N be the data size, M be the memory size, and the B be the page (transfer unit) size.

- Step 1: Read all the N items once and write back to the disk once, incurring $O(N/B)$ I/Os.
- Step 2&3: During merge and writeback, each item is read from and written to the disk exactly once, incurring $O(N/B)$ I/Os.

COST ANALYSIS

Let N be the data size, M be the memory size, and the B be the page (transfer unit) size.

- Step 1: Read all the N items once and write back to the disk once, incurring $O(N/B)$ I/Os.
- Step 2&3: During merge and writeback, each item is read from and written to the disk exactly once, incurring $O(N/B)$ I/Os.

Does this analysis apply to all situations?



COST ANALYSIS

To make the analysis hold, we should have

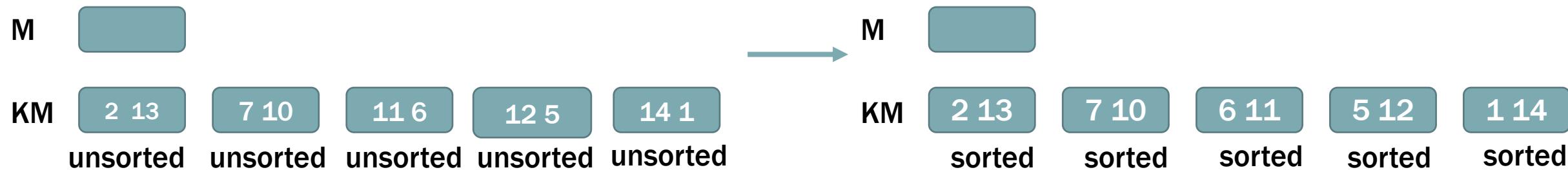
- The size of write buffer is at least one page size (size B)
- The process of in-memory merge sort should be fit in main memory.

Does this analysis apply to all situations?



SORTING

Suppose the size of main memory is M , the array size is $KM (=N)$.



Step 1:

Cut KM arrays into K parts. Each part is put in main memory to sort (using any sorting algorithm we learnt, e.g., quick sort)

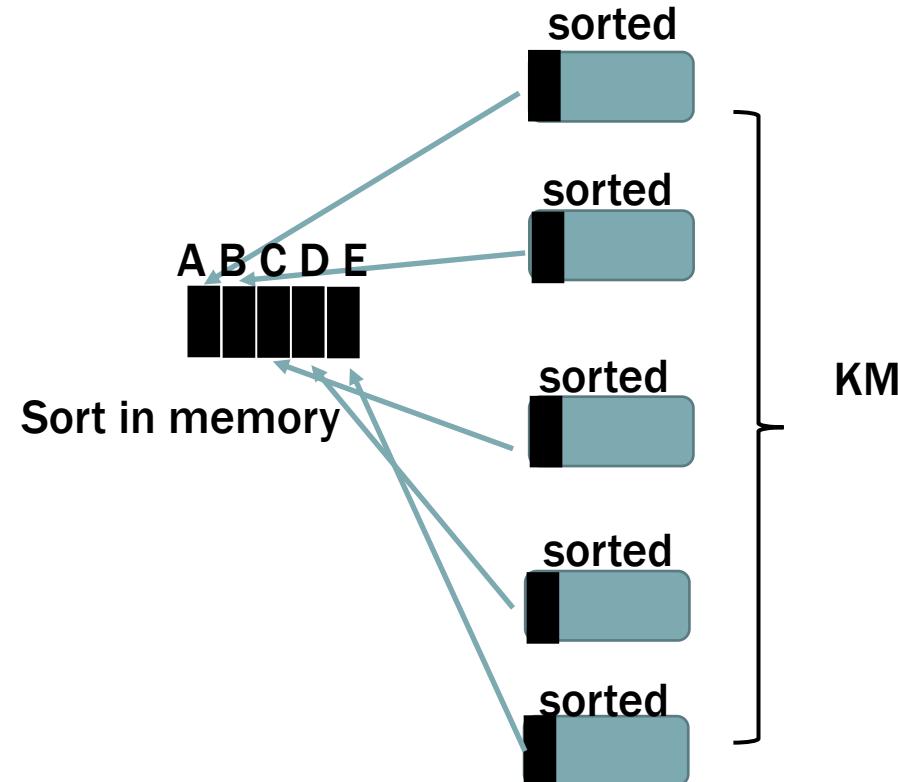
SORTING

■ or ■ size $M/(K+1)$

$KM/(K+1)$ in total for in-memory sorting, remaining $M/(K+1)$ for writing the results



$M/(K+1)$ write buffer



Step 2:

Apply K-way merge sort to the first $M/(K+1)$ of each sorted part.

THE RANGE OF K

- Condition 1:

A write buffer has at least one page size (denoted by B).

Hence, we have $M/(K+1) \geq B$, giving us **$K \leq M/B - 1$**

- Condition 2:

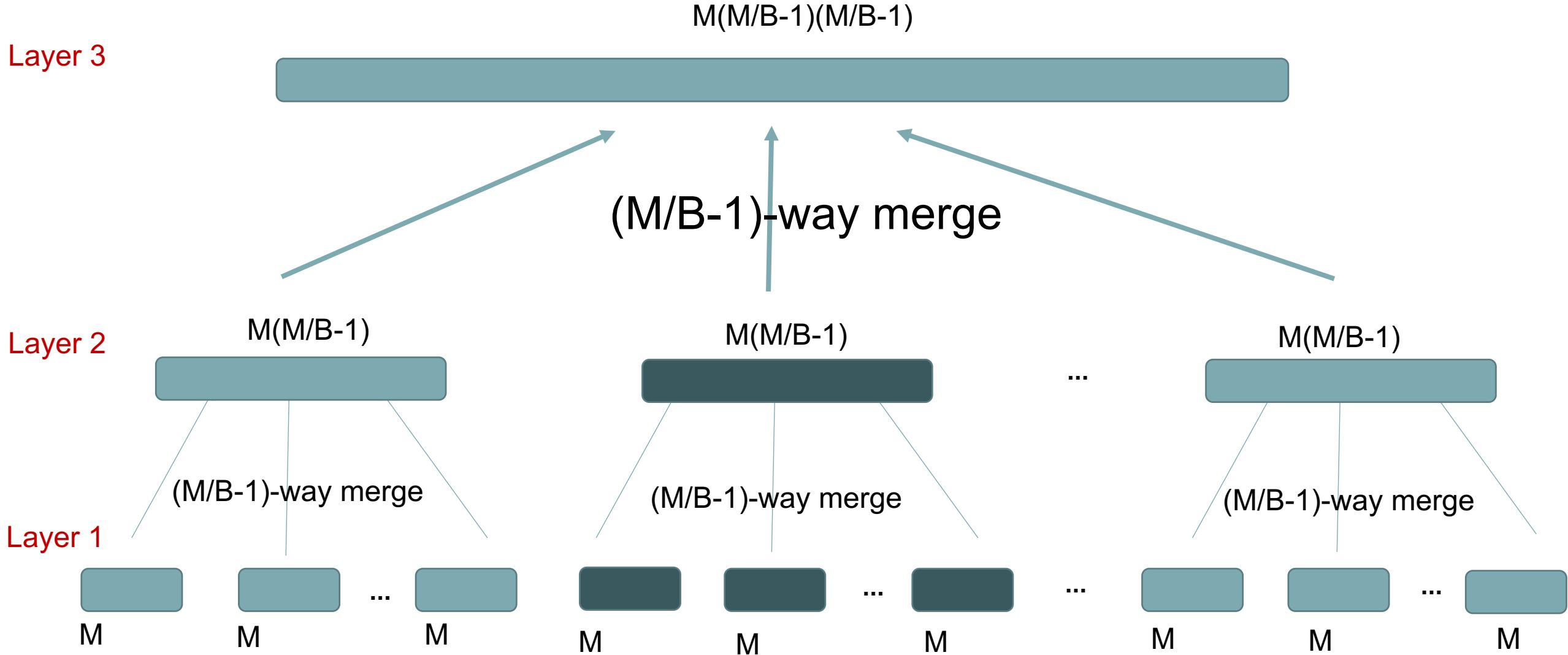
The process of K-way merge-sort should be fit in remaining memory size (excluding write buffer).

Hence, $K \times B \leq KM/(K+1)$, also giving us **$K \leq M/B - 1$**

EXTEND TO GENERAL CASE

- The cost analysis holds when N is at most $(M/B-1)M$.
- We can extend the method to general N by recursively applying the $(M/B-1)$ -way merge.

RECURSIVELY APPLY (M/B-1)-WAY MERGE



GENERAL ANALYSIS

- If there are L layers, then $M(M/B-1)^{L-1} = N$ because N items should be sorted, giving

$$L = O(\log_{M/B} N/M)$$

$$\begin{aligned} M(M/B - 1)^{L-1} &= N \\ \Leftrightarrow (M/B - 1)^{L-1} &= N/M \\ \Leftrightarrow \log_{M/B-1}(M/B - 1)^{L-1} &= \log_{M/B-1} N/M \\ \Leftrightarrow L - 1 &= \log_{M/B-1} N/M \\ \Leftrightarrow L &= 1 + \log_{M/B-1} N/M \\ \Leftrightarrow L &= O(\log_{M/B} N/M) \end{aligned}$$

- The merges in each layer costs $O(N/B)$ I/Os.

- Hence, the total cost is $O((N/B) \log_{M/B} N/M)$ I/Os.

We finish Memory Hierarchy!



Next lecture:

Column Store

BIG DATA MANAGEMENT

CE/CZ4123

COLUMN STORE

PART I

Siqiang Luo

Assistant Professor

PREPARATION

- We learnt some principles of data models, memory hierarchy, and cache-conscious designs.

- In this lecture, we will see how the previous learnt concepts impact the design of modern big data system – column store

Main Designs of Column Store

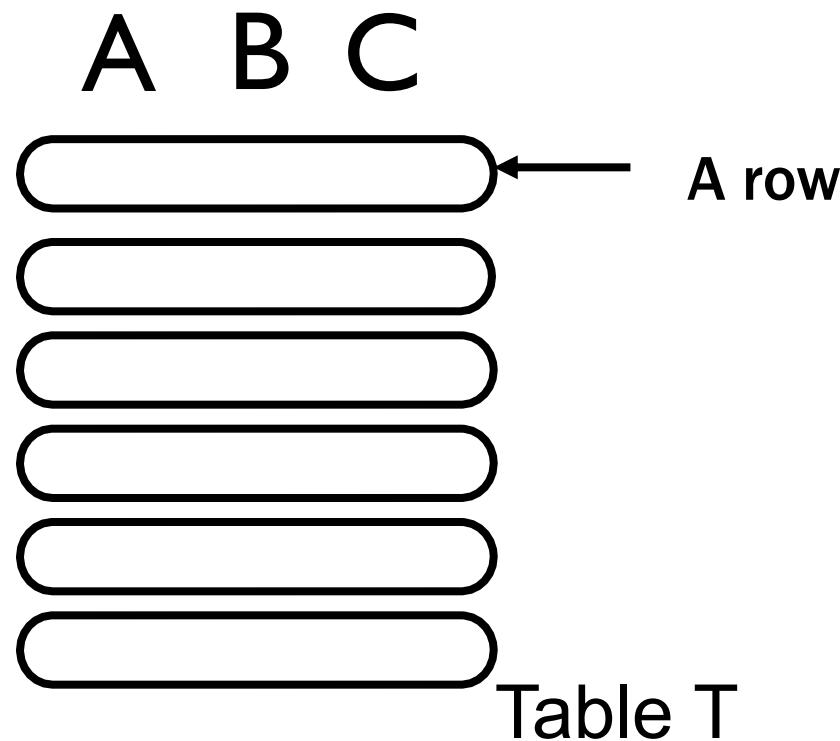
COLUMN STORE HANDLES TABULAR DATA

	Column A	Column B	Column C
	id	name	age
Row 1	0001	Alex	25
Row 2	0002	Mary	35

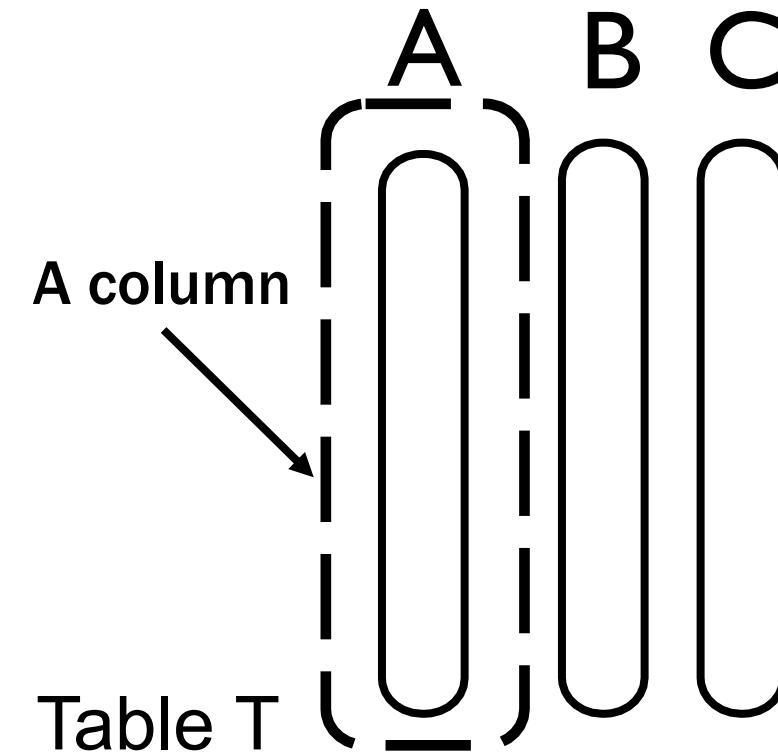
MAIN DESIGN OF COLUMN STORES

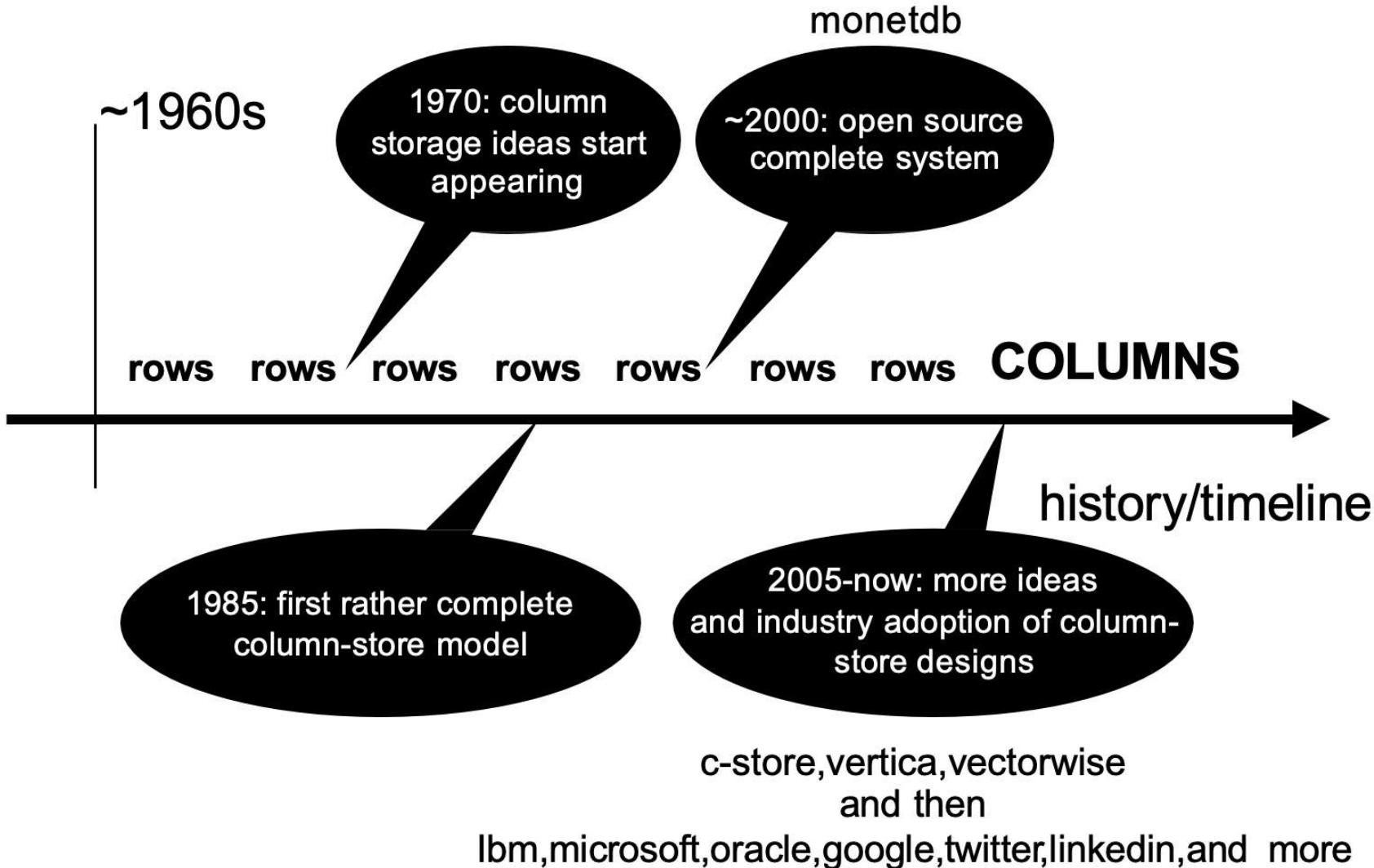
- Data in column stores are column-oriented
 - Also called column-oriented database, or columnar database.

row-store (traditional RDB)



column-store





Credit to
Prof. Stratos Idreos
at Harvard University.

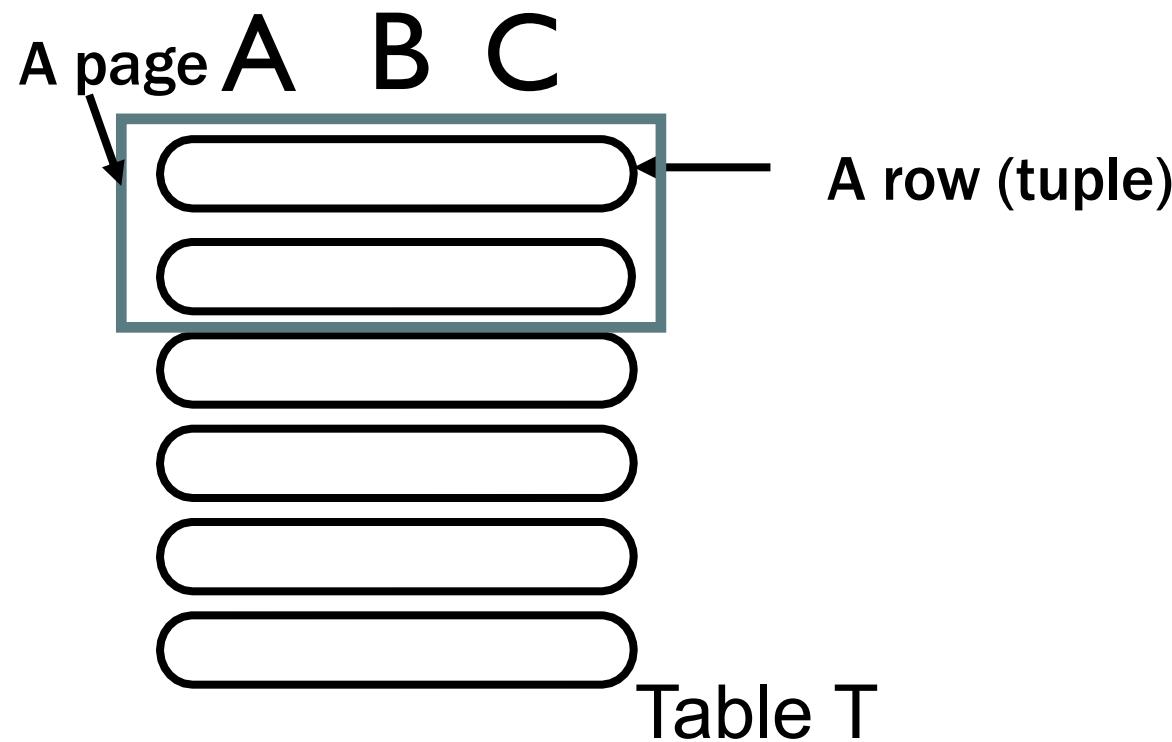
MAIN DESIGN OF COLUMN STORES

- Data are stored based on **pages**
- A file consists of multiple pages.
- A page is usually considered as the transfer unit between disk and memory
- A page may contain multiple rows (tuples) in row-store

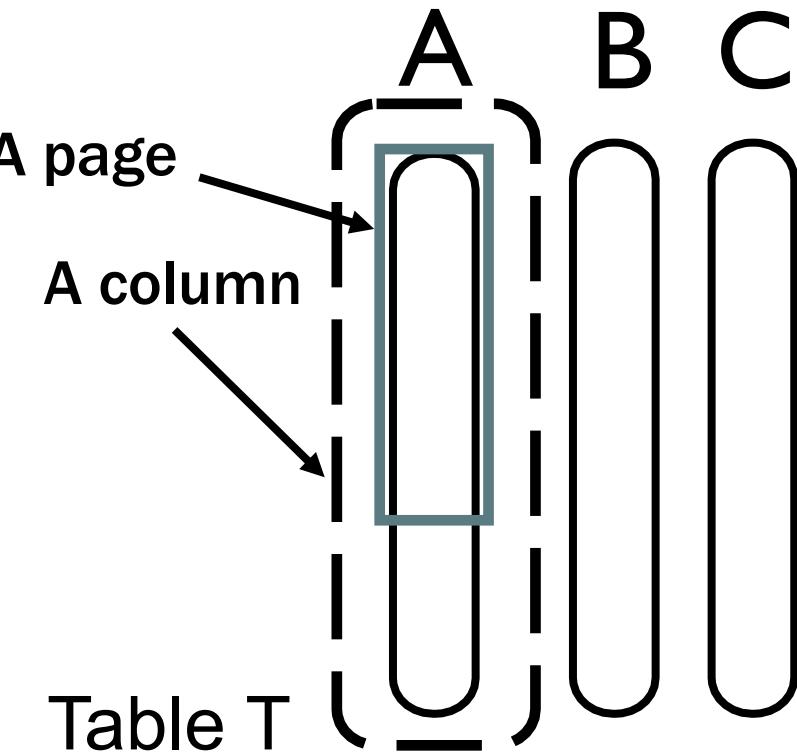
MAIN DESIGN OF COLUMN STORES

- Data are stored based on **pages**

row-store (traditional RDB)



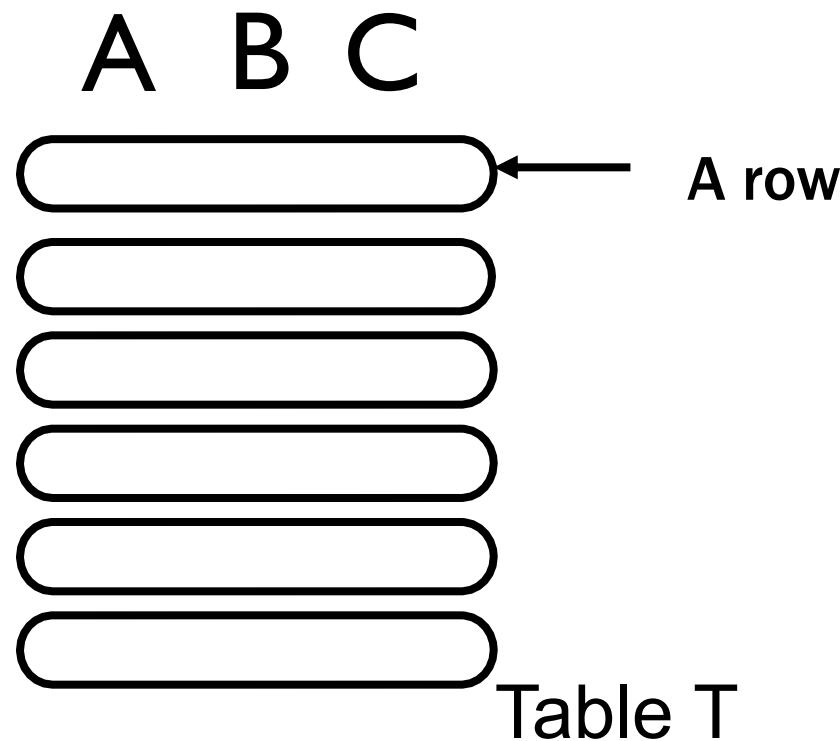
column-store



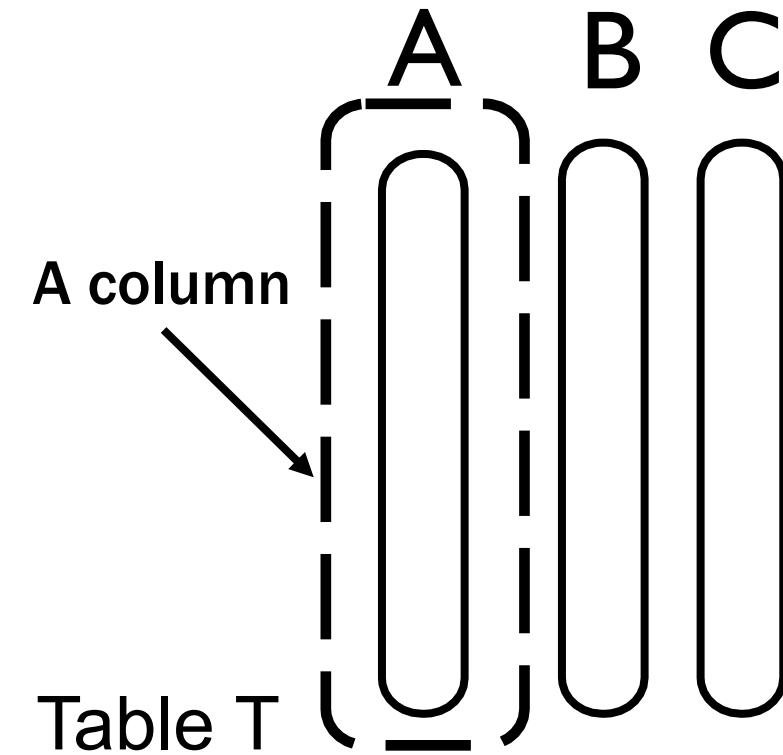
WHY COLUMN STORE

- Recently, column-store becomes one of the major systems in the industry.
Can you work out in what situations column-stores may have benefits?

row-store (traditional RDB)



column-store



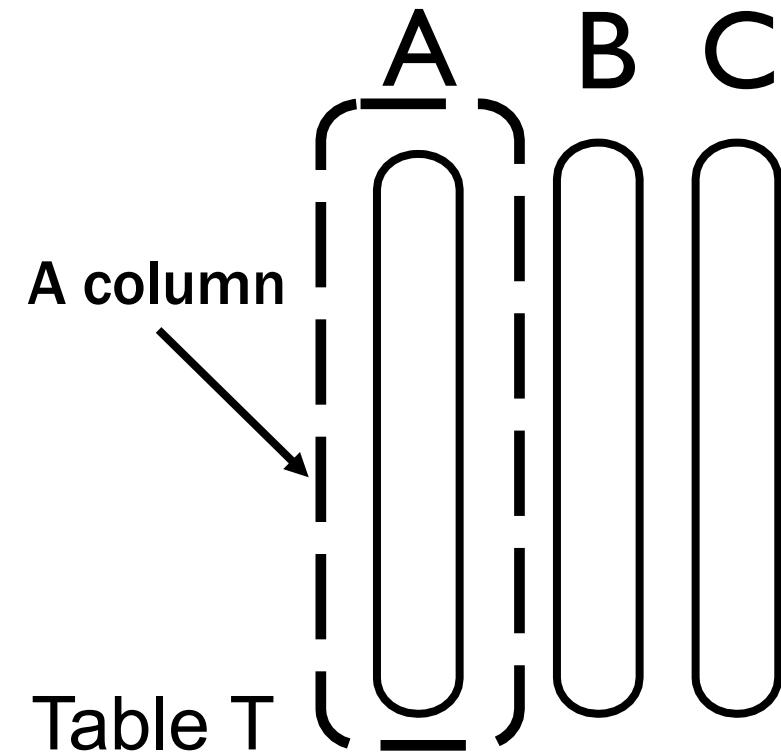
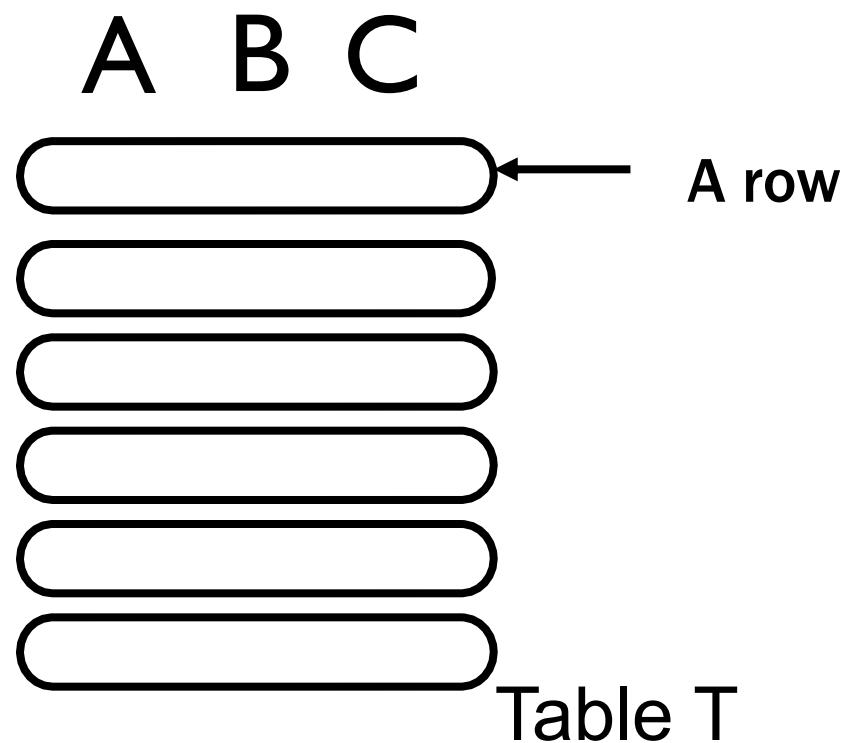
An excellent introduction video here.

First 3 minutes of

<https://www.sisense.com/glossary/columnar-database/>

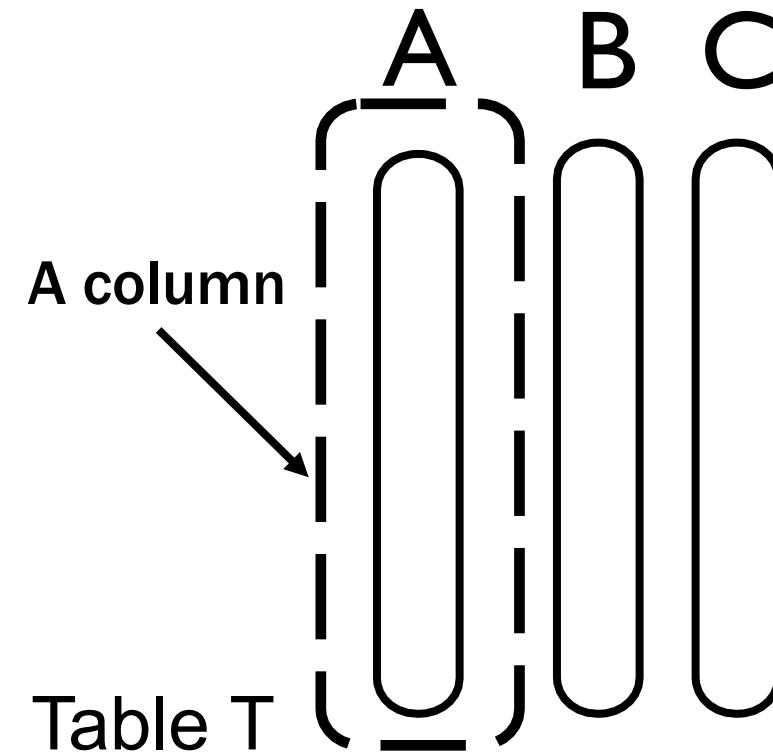
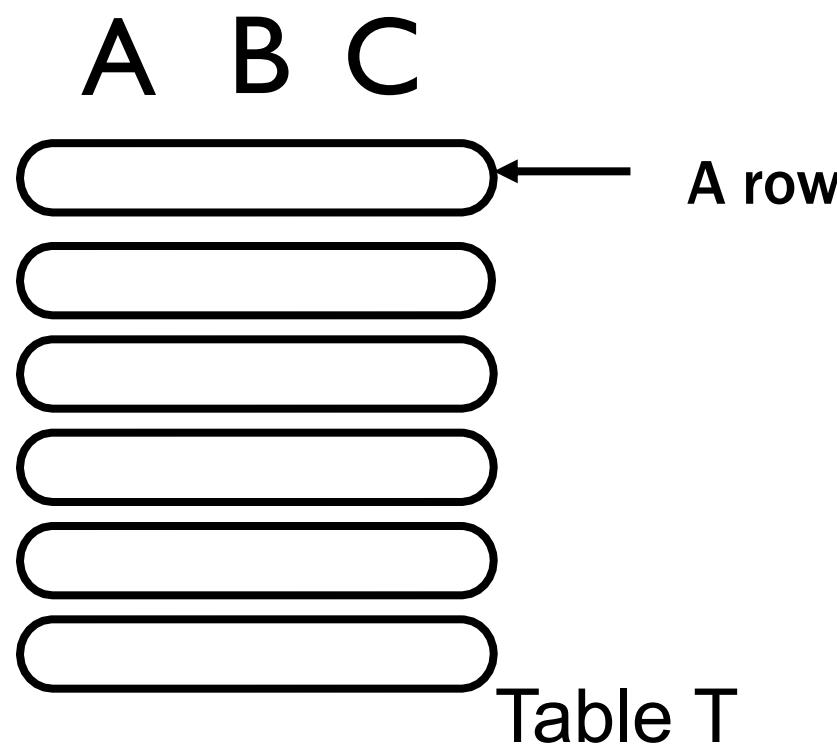
WHY COLUMN STORE

- Consider the following cases:
- select A from T where A>4



WHY COLUMN STORE

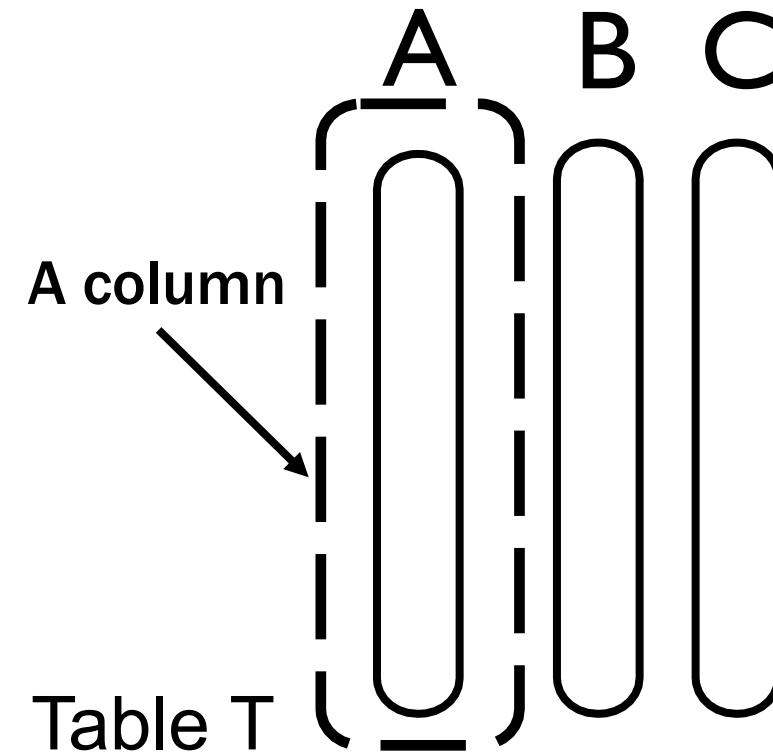
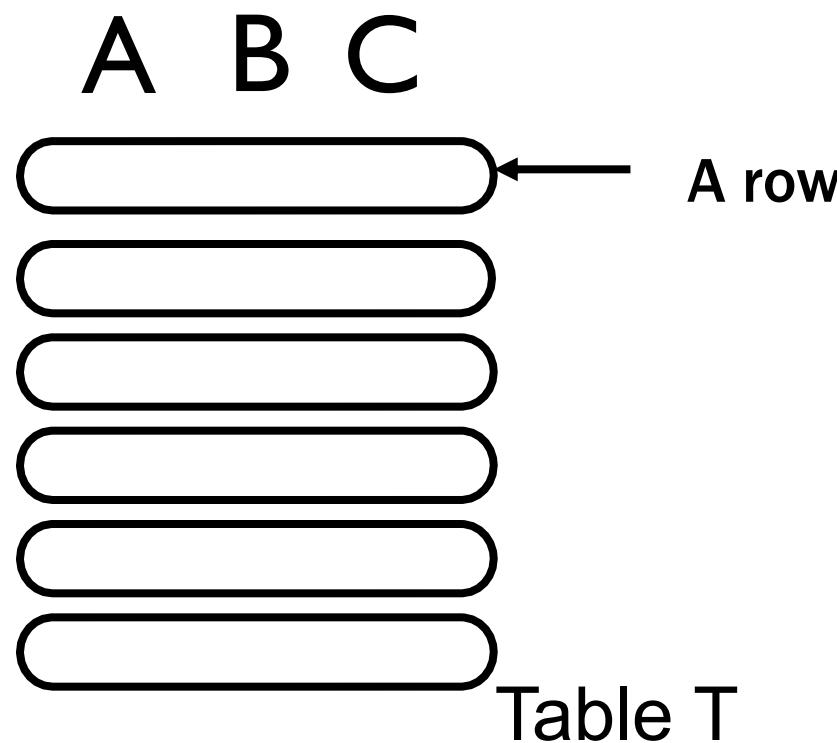
- ❑ Column store is efficient to read just the columns you need for a query.
 - ❑ No need to extract the other columns.
 - ❑ In fact, commercial data table may have tens of columns. Not every column is touched in a query.



Any other benefits?

WHY COLUMN STORE

- ❑ Column stores are compression friendly.
 - ❑ Same type of data are stored consecutively → easier to compress
 - ❑ More compressed data can reduce the cost moving data from disks to main memory.



Querying with Column Store

QUERYING WITH COLUMN STORE

- The whole table is stored in disk
- Have disk cache (in main memory) whose size is a multiple of page size

```
select max( C )
from T
where A<5 and B>3
```

Table T

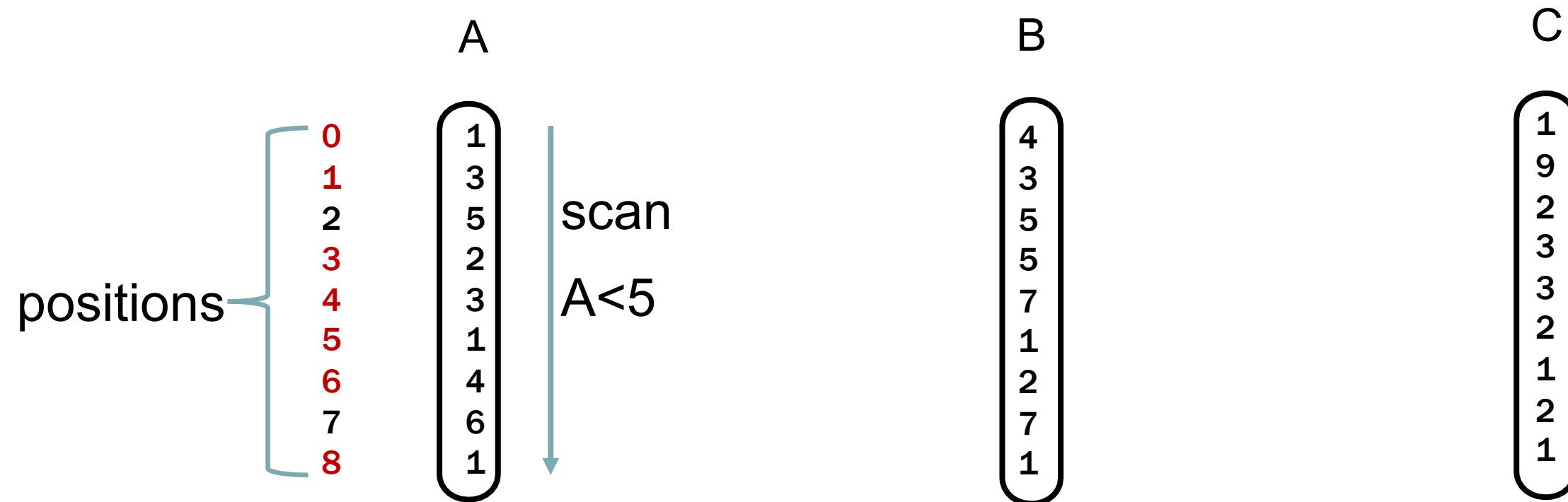
	A	B	C
0	1	3	4
1	3	5	3
2	5	2	5
3	2	3	5
4	3	7	7
5	1	1	1
6	4	2	2
7	6	2	2
8	1	7	1

positions

The diagram illustrates the structure of the column store. A bracket labeled "positions" spans the first three columns (A, B, and C), indicating that they are stored sequentially. The fourth column, C, is shown separately to the right, likely representing a compressed or processed version of the data.

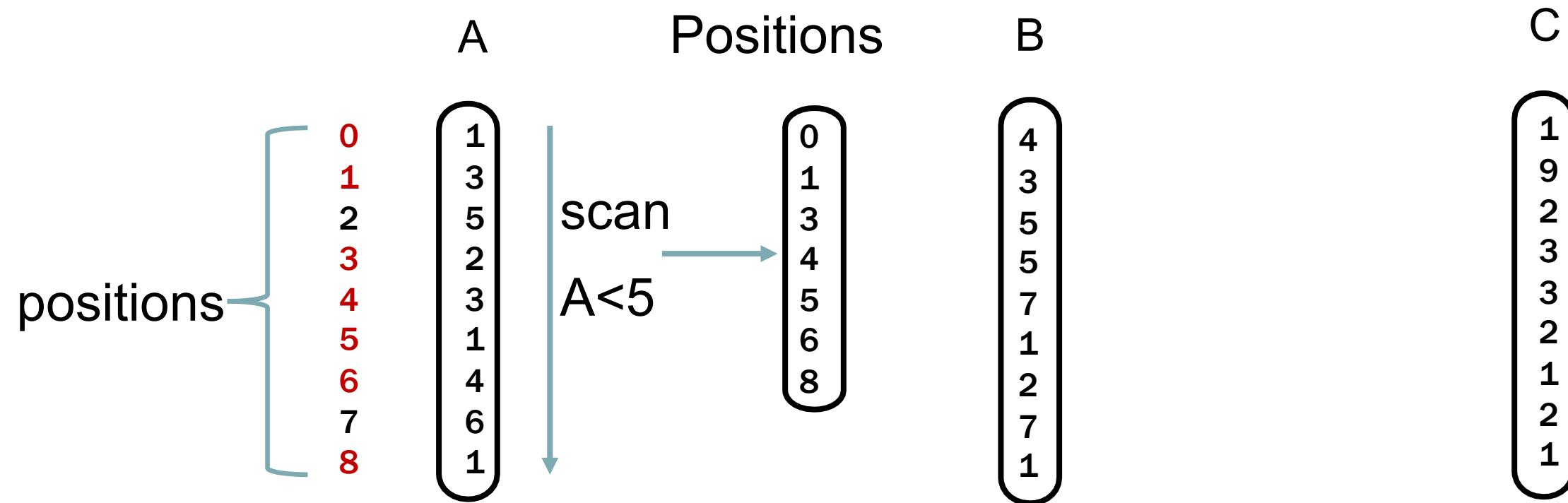
QUERYING WITH COLUMN STORE (FLOW CHART)

- The whole table is stored in disk
- Have disk cache (memory buffer) whose size is a multiple of page size



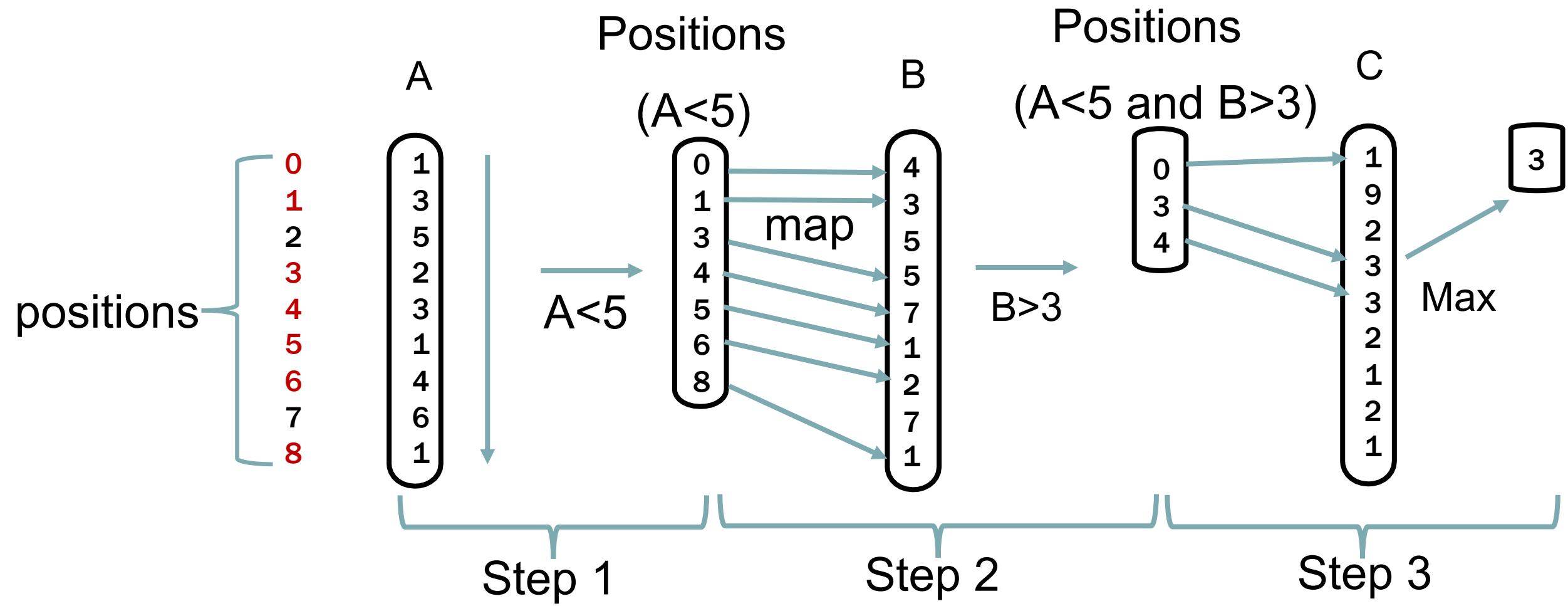
QUERYING WITH COLUMN STORE (FLOW CHART)

- The whole table is stored in disk
- Have disk cache (memory buffer) whose size is a multiple of page size

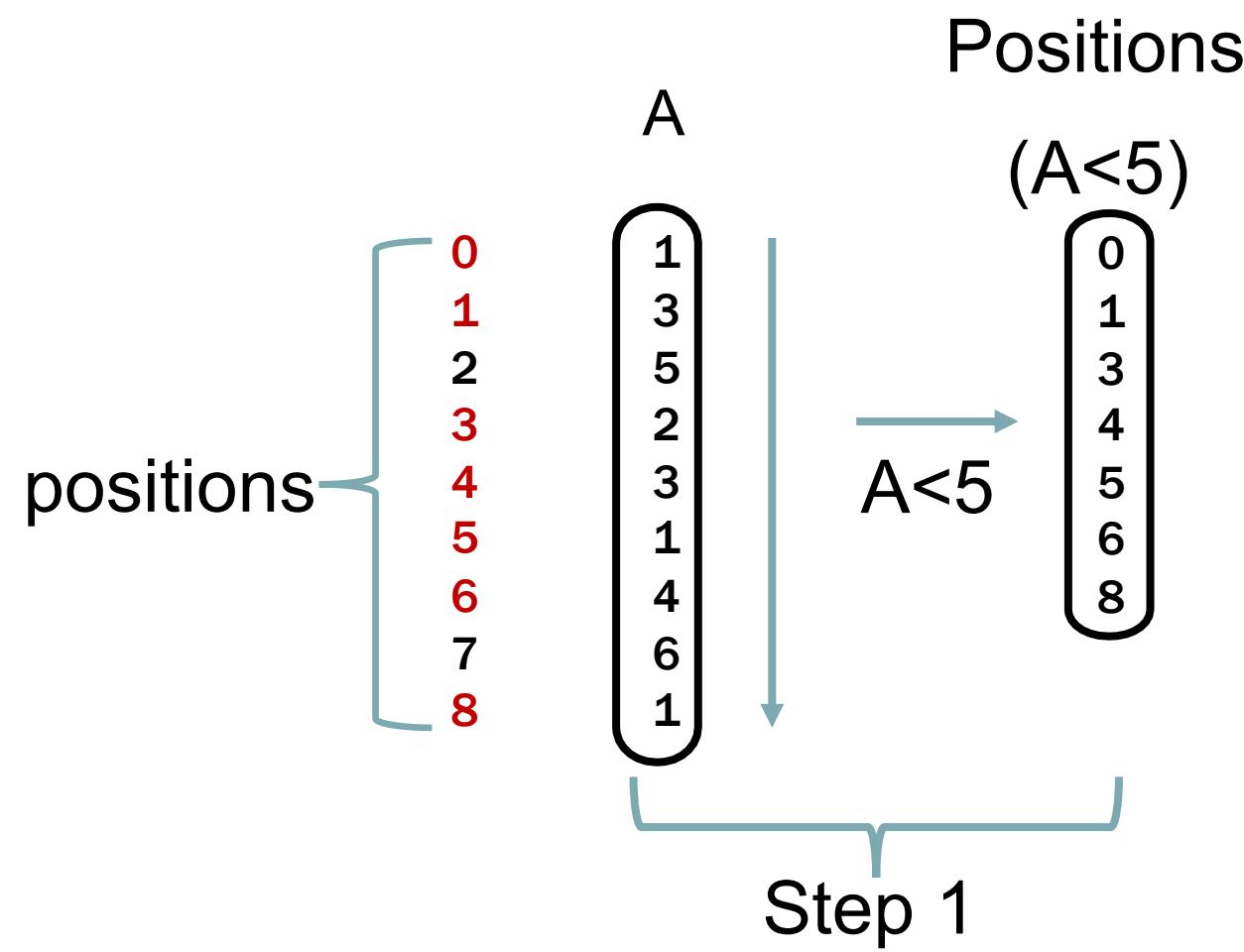


QUERYING WITH COLUMN STORE (FLOW CHART)

- The whole table is stored in disk
- Have disk cache (memory buffer) whose size is a multiple of page size



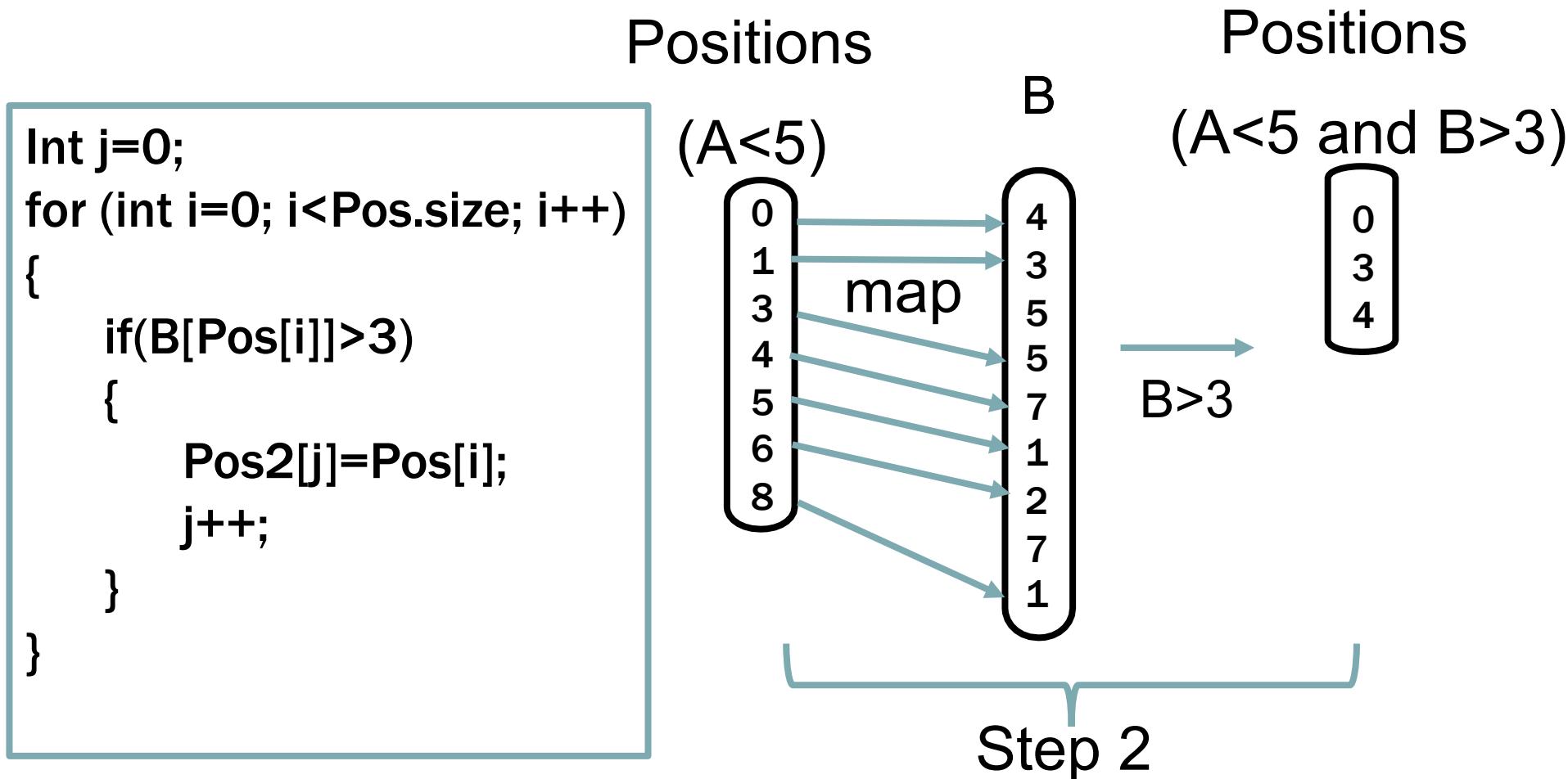
DETAILS OF STEP 1



```
int j=0;  
for (int i=0; i<#tuples; i++)  
{  
    if(A[i]<5)  
    {  
        Pos[j]=i;  
        j++;  
    }  
}
```

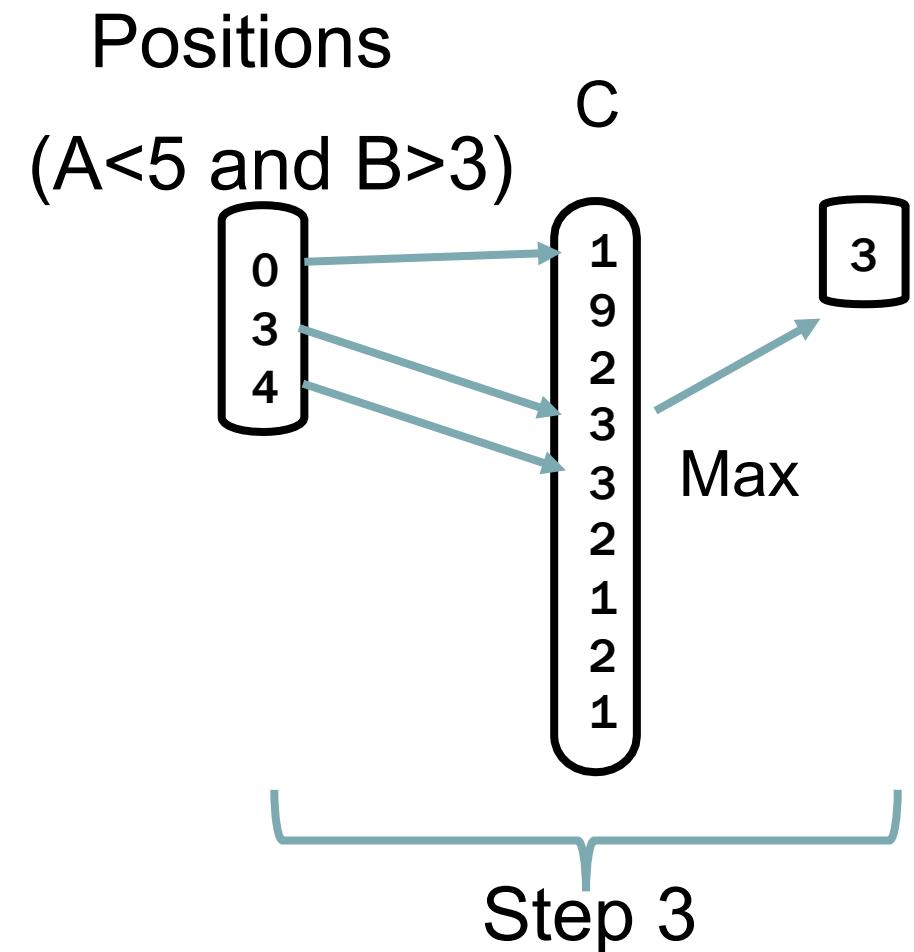
QUERYING WITH COLUMN STORE

- The whole table is stored in disk
- Have disk cache (memory buffer) whose size is a multiple of page size



QUERYING WITH COLUMN STORE

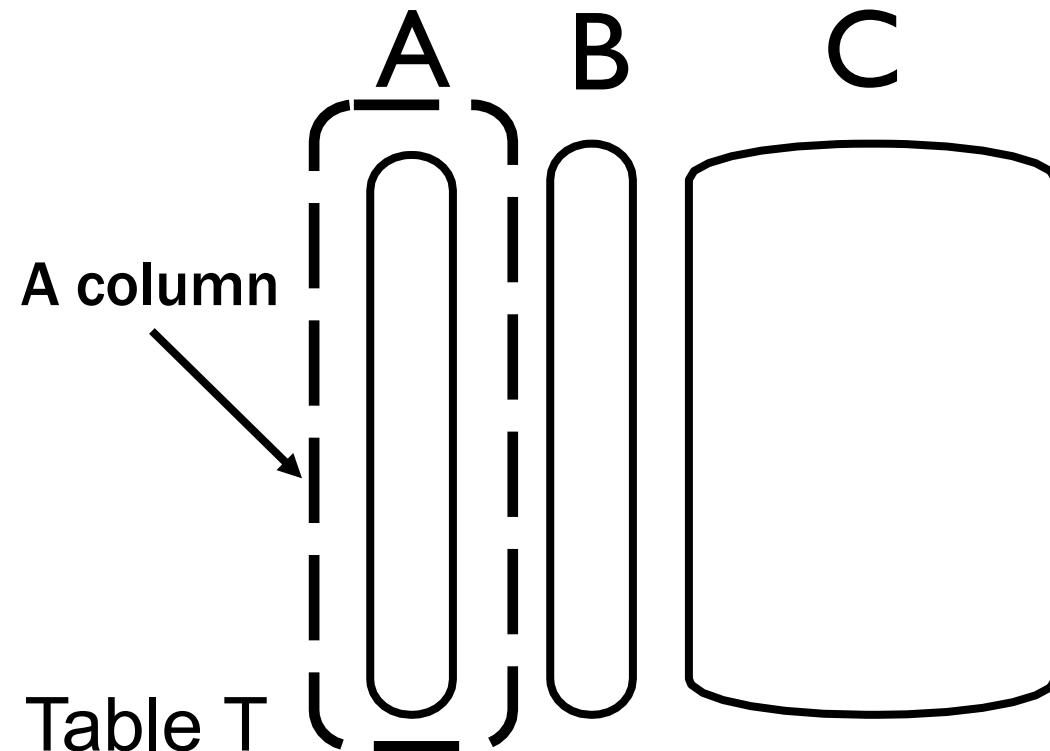
```
int max;  
for (int i=0; i<Pos2.size; i++)  
{  
    if(i==0) max=C[Pos2[i]];  
    else if(max<C[Pos2[i]])  
    {  
        max=C[Pos2[i]];  
    }  
}
```



COLUMN CAN BE OF DIFFERENT WIDTHS

Different columns do not have to have the same width

❑ e.g., in Employee(Name, Gender, Emails), all three attributes may occupy different bytes.



Memory VS Disk

Implementations

IN-MEMORY V.S. DISK-BASED IMPLEMENTATIONS

Previous pseudo codes are illustrated based on in-memory implementations.
Now we discuss the disk-based implementation.

The major difference is that

We cannot allocate a big array A or Pos because the data volume can be huge in practice.

The data is mainly stored in files, but it allows some buffers in the main memory.

```
int j=0;
for (int i=0; i<#tuples; i++)
{
    if(A[i]<5)
    {
        Pos[j]=i;
        j++;
    }
}
```

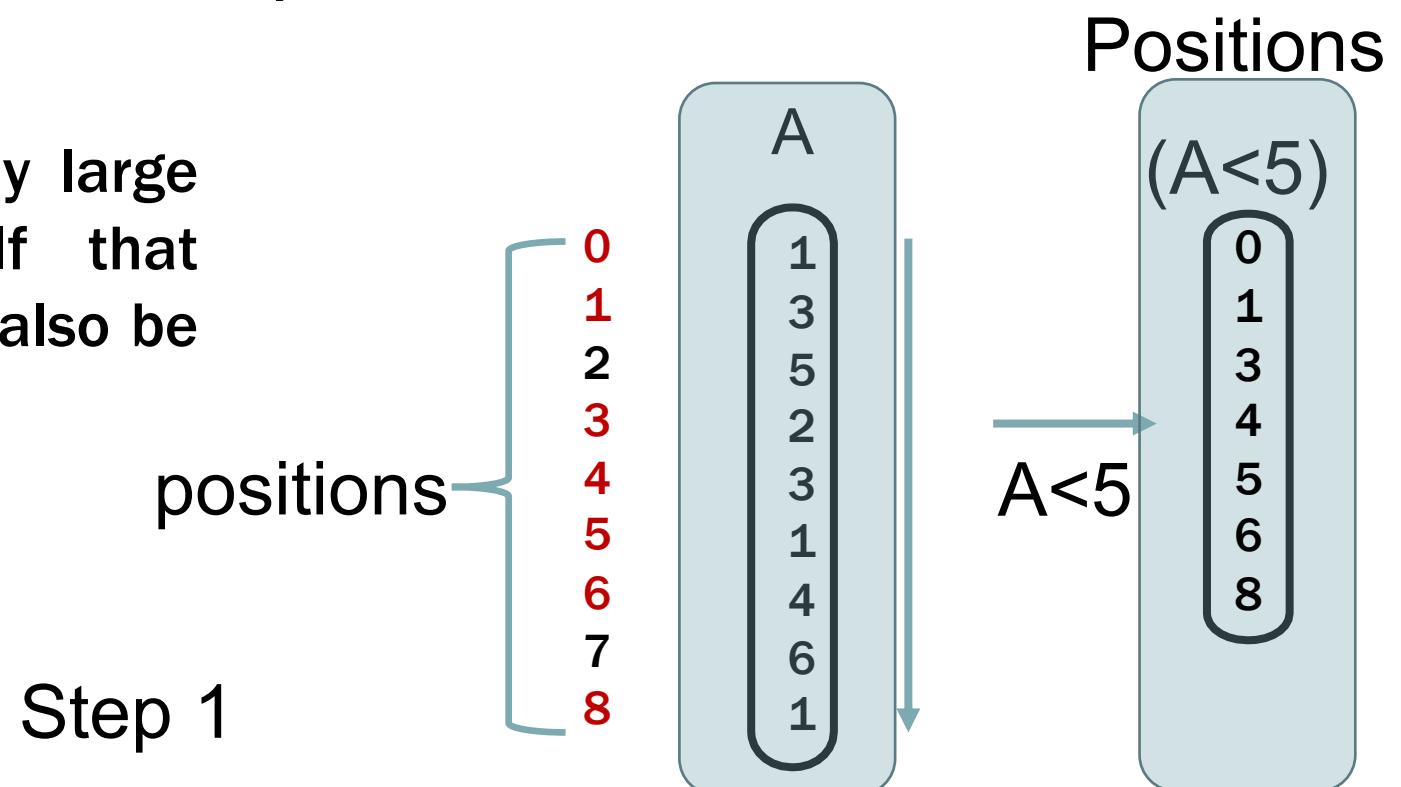
IN-MEMORY V.S. DISK-BASED IMPLEMENTATIONS

Usually, each column is stored as a file (or multiple files) in the disk.

The scanning of the column involves file access, e.g.,

- ❖ For C programming: `fread()`
- ❖ For JAVA programming: various “`FileInputStream`”s

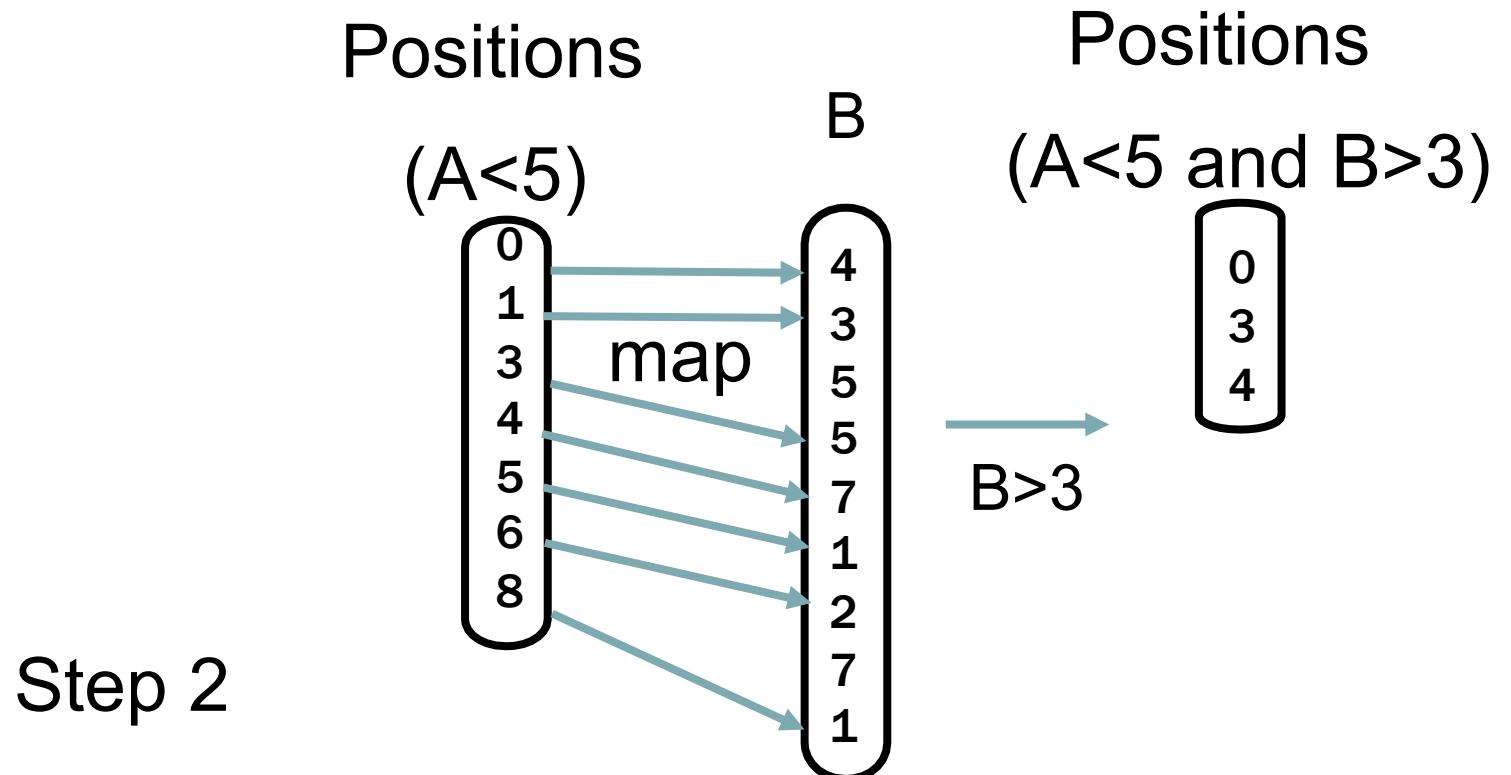
The position list can be overly large (larger than the buffer). If that happens, the position list can also be stored in a file.



IN-MEMORY V.S. DISK-BASED IMPLEMENTATIONS

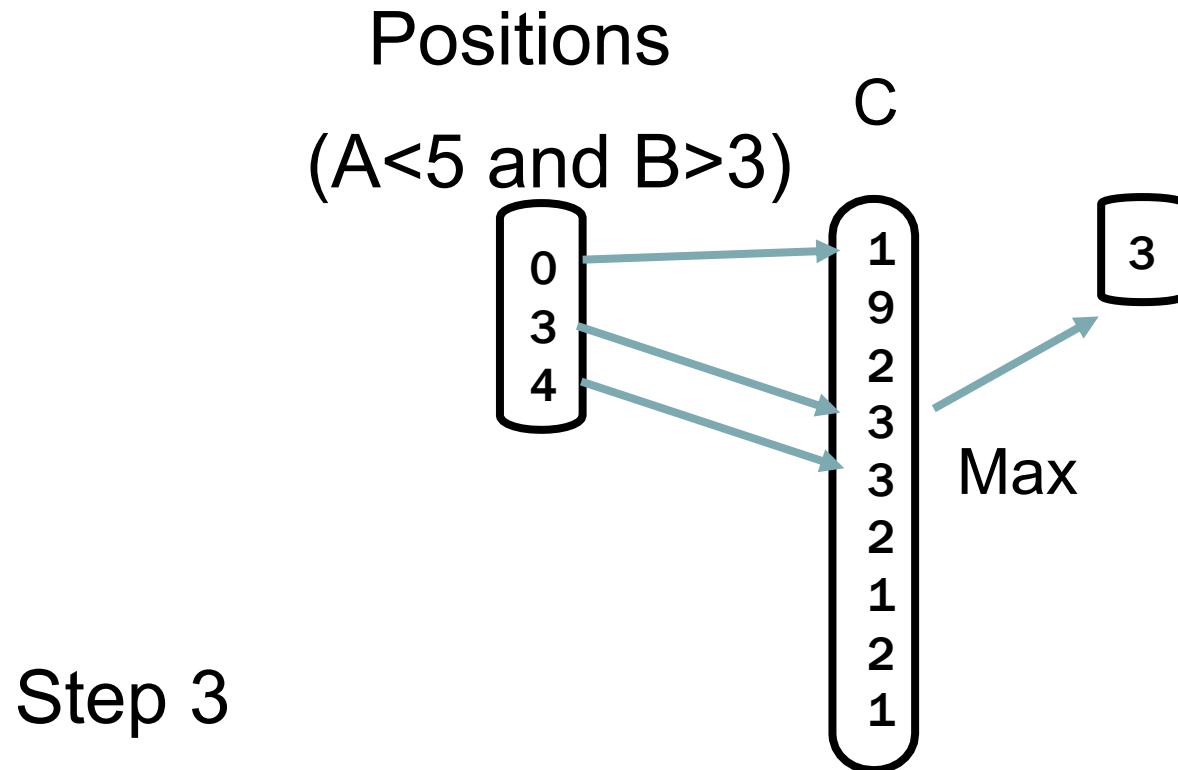
In step 2, the “map” process can be implemented in a disk-based manner. Since column B is stored in a file, when given a position, we need to seek the corresponding values in the file containing column B.

- ❖ For example, using fseek() in C programming. It directly shifts to the desired starting point of the file.



IN-MEMORY V.S. DISK-BASED IMPLEMENTATIONS

For step 3, similar “map” process can be done in a disk-based manner.

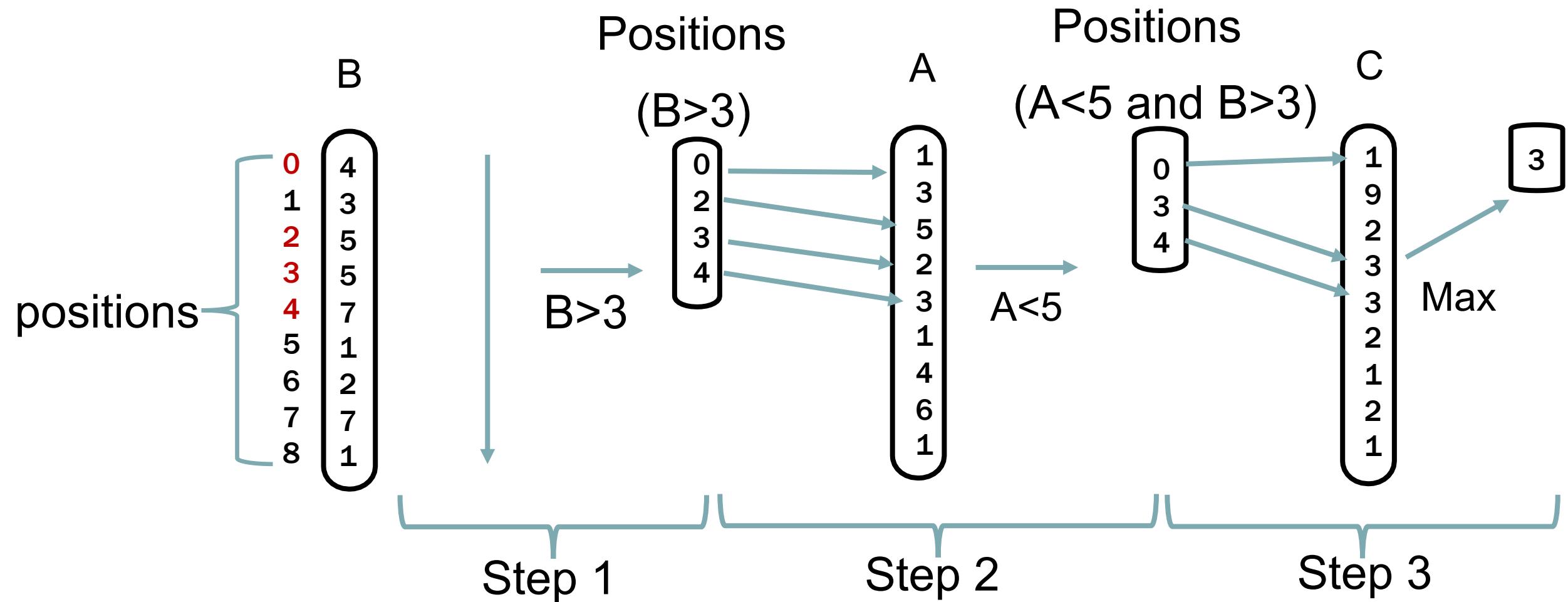


Discuss Alternative Schemes

ALTERNATIVE SCHEMES

Can also check $B > 3$ first.

Can you write down the procedure of this alternative approach?



ALTERNATIVE SCHEMES

Also, one can do the following:

Step 1: Scan A and obtain the qualified results

Step 2: Scan B and obtain the qualified results

Step 3: Merge the results (intersect the results in the previous case).

Option 1: Scan A first

Option 2: Scan B first

Option 3: Scan A and B independently

Which is the best?



- ❑ Usually, Option 3 is not as good as Option 1 (or Option 2) because it needs to scan entire A and B
- ❑ Option 1 and Option 2: which is better roughly depends on the number of qualified results after processing that column.
 - ❑ If scanning A gets much less results than B, then scanning A first is better.
 - ❑ If scanning B gets much less results than A, then scanning B first is better.

Query Cost Analysis

COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Suppose a column has Z values;

Column width = w bytes;

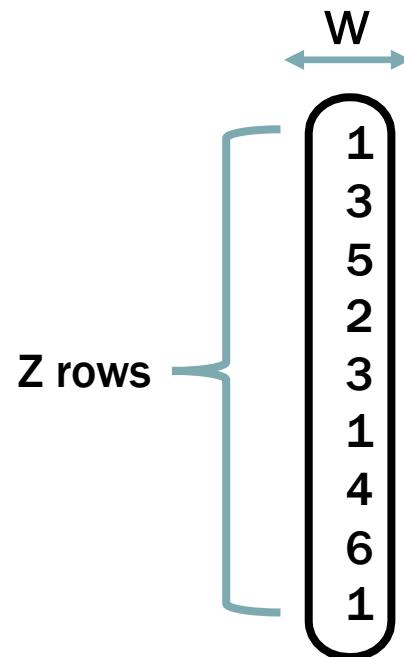
Page size = P bytes;

Storing a position costs 4 bytes;

Each column is sequentially stored in the disk.

We assume position list is stored in files.

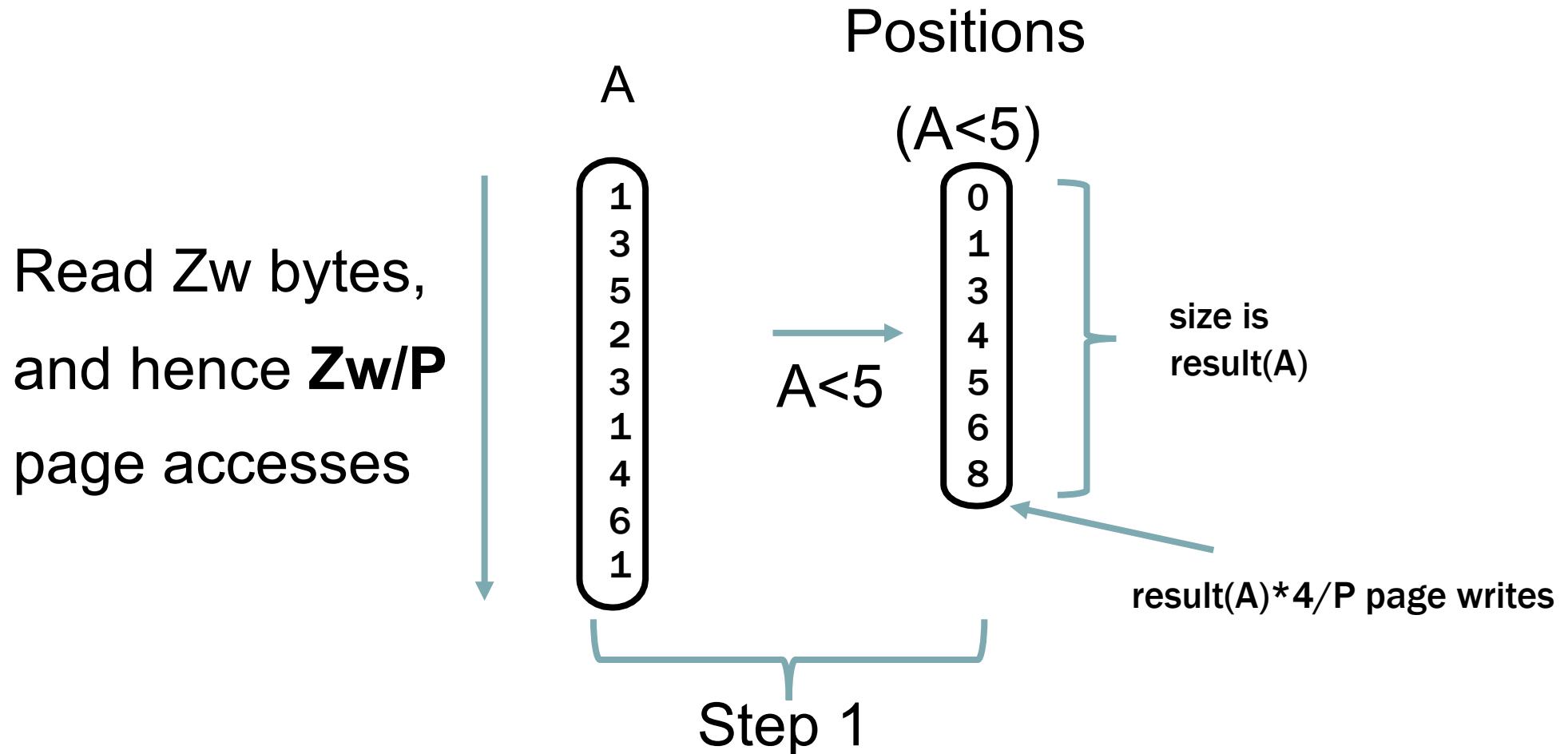
We also assume $w < P$ and each value in a column is contained in a page.



In this particular example, $w=4$ bytes (integer size)

COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Step 1: Scanning A costs Zw/P page reads and $\text{result}(A)*4/P$ page writes.
(note: $\text{result}(A)$ is the number of qualified positions after processing column A.)

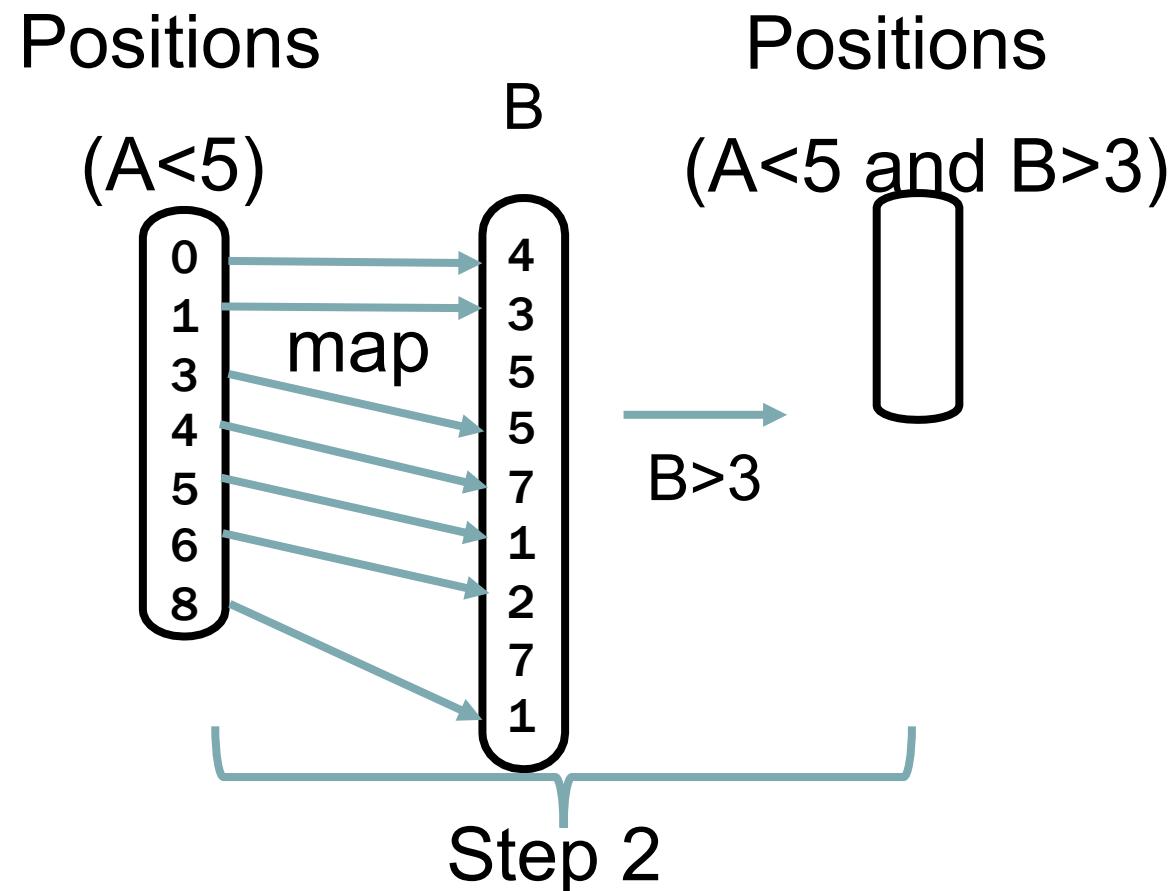


COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Step 2:

Reading the positions (for $A < 5$) costs $\text{result}(A) * 4/P$ page reads;

Fetching B costs **at most** $\text{result}(A)$ page reads and $\text{result}(AB) * 4/P$ page writes.



Why the cost of fetching B costs **at most** $\text{result}(A)$ page reads instead of $\text{result}(A)/P$ page reads?



Why the cost of fetching B costs **at most** $\text{result}(A)$ page reads instead of $\text{result}(A)/P$ page reads?

Random reads!

Why the cost of writing positions for B costs $\text{result}(A) * 4/P$ page writes instead of $\text{result}(A)$ page writes?

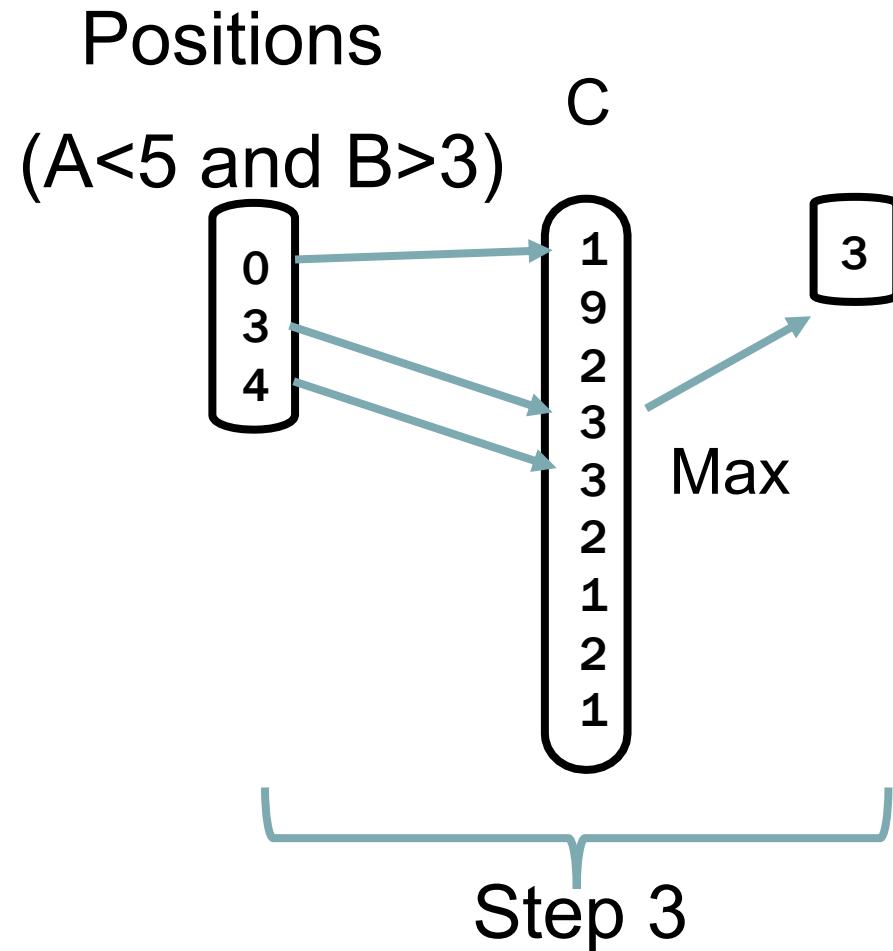


Why the cost of writing positions for B costs $\text{result}(A) * 4/P$ page writes instead of $\text{result}(A)$ page writes?

Sequential writes!

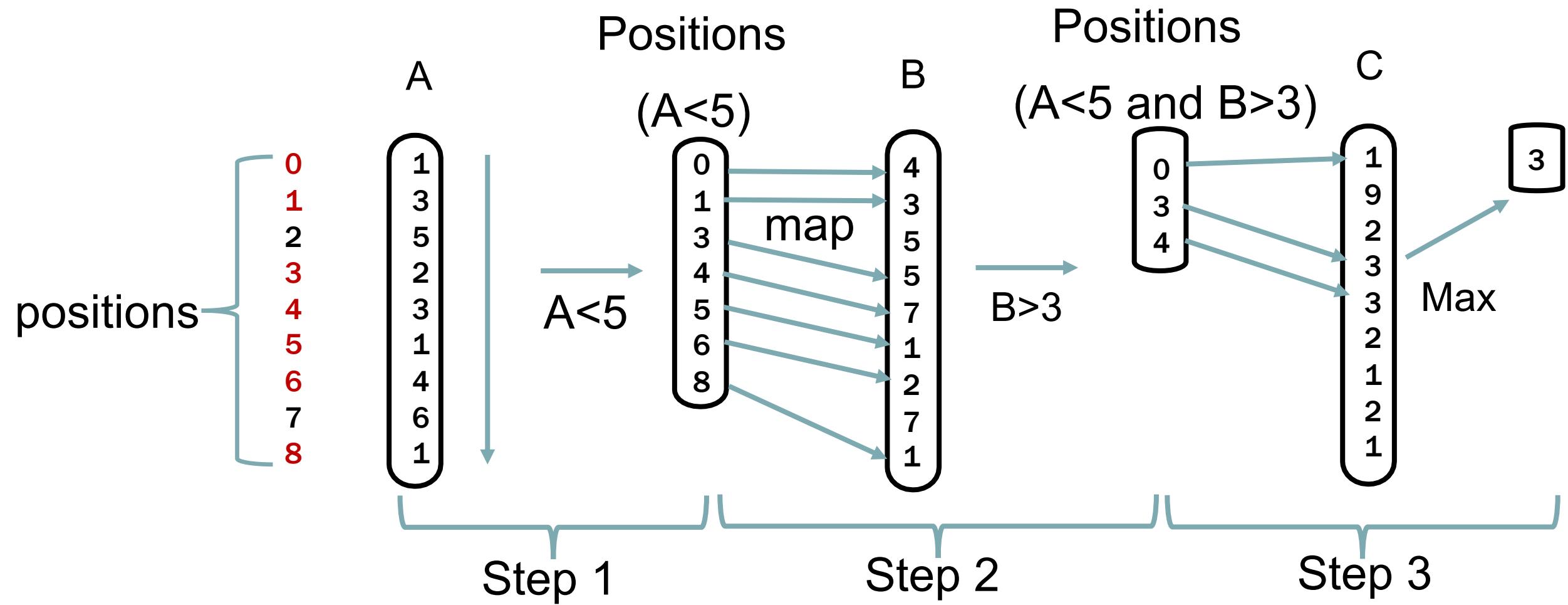
COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Step 3: Reading the positions ($A < 5$ and $B > 3$) costs $\text{result}(AB) * 4 / P$ page reads;
Fetching C costs at most $\text{result}(AB)$ page reads.



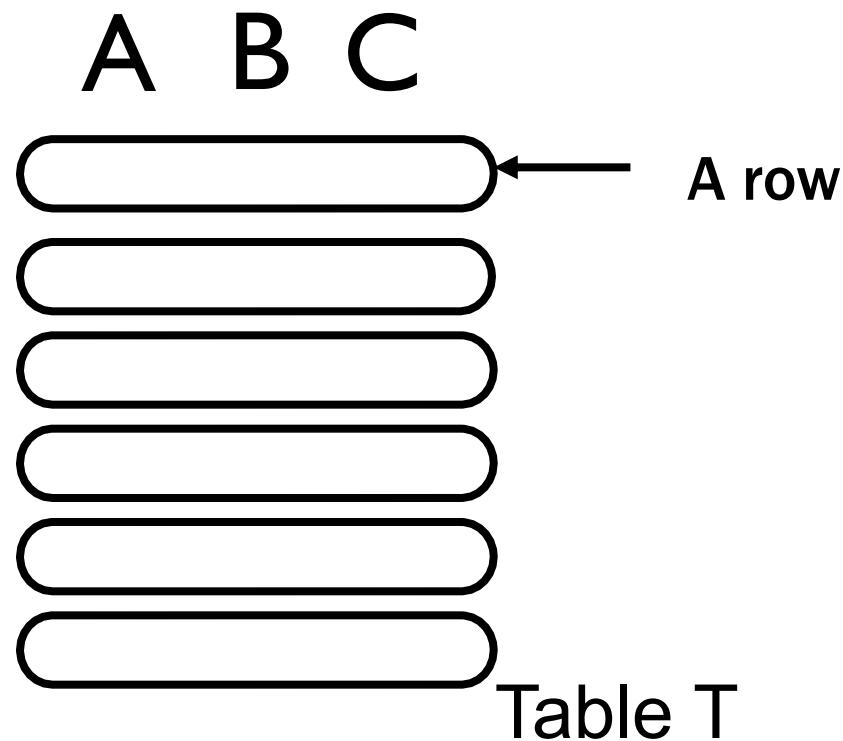
COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Overall: $Zw/P + 2\text{result}(A)*4/P + \text{result}(A) + 2\text{result}(AB)*4/P + \text{result}(AB)$ page accesses.



COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Cost of using row store?



Discussion

COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Without using an index, handling the query in the row store needs to read through the whole table. Namely, it costs $Z \cdot w \cdot 3/P$ page accesses.

Cost for column store (number of page access):

$$\begin{aligned} & Zw/P + 2\text{result}(A) \cdot 4/P + \text{result}(A) + 2\text{result}(AB) \cdot 4/P + \text{result}(AB) \\ & = Zw/P + \text{result}(A) \cdot (8/P + 1) + \text{result}(AB) \cdot (8/P + 1) \\ & \approx Zw/P + \text{result}(A) + \text{result}(AB) \end{aligned}$$

Cost for row store (number of page access):

$$3Zw/P$$

Note: the values of $\text{result}(A)$ and $\text{result}(AB)$ are typically much smaller than Z . So we can conclude in this case column store is faster.

REMARK

In many cases, position list can be small enough to be simply stored in the memory buffer

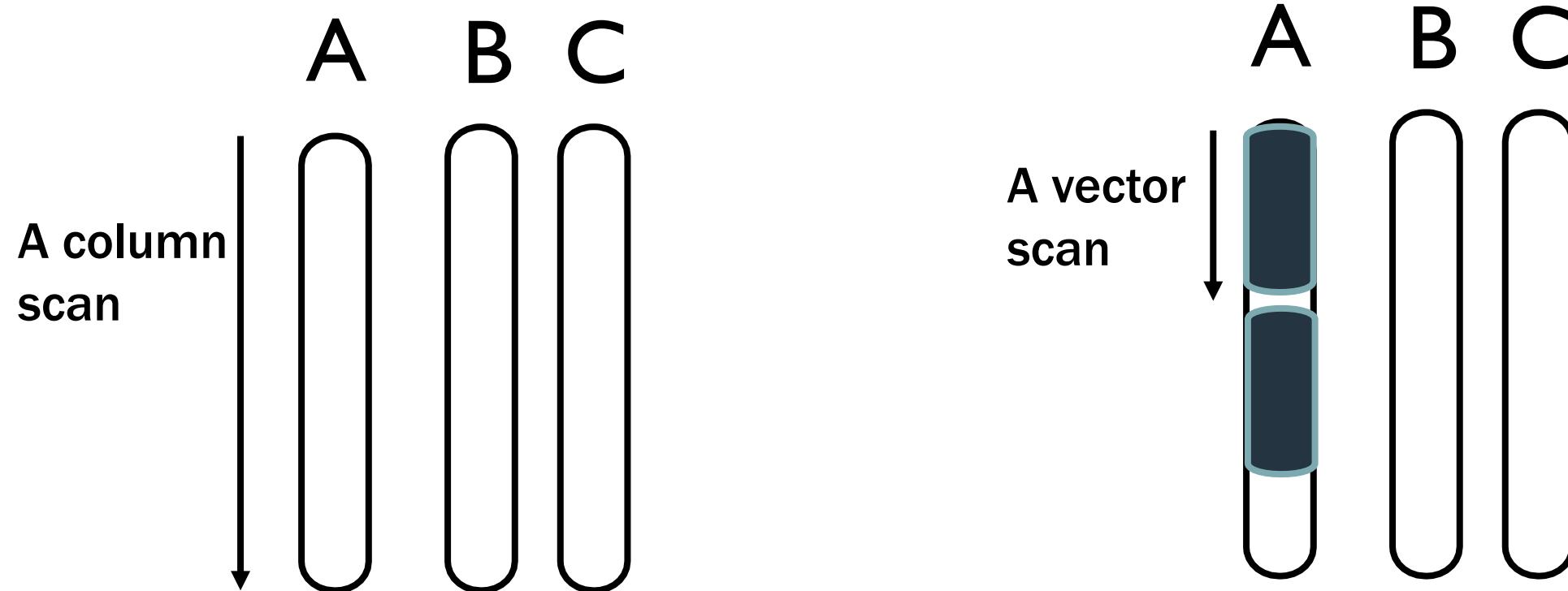
In these cases, cost for column store (number of page access):

$$\begin{aligned} & \cancel{Zw/P + 2\text{result}(A)*4/P + \text{result}(A)} + \cancel{2\text{result}(AB)*4/P + \text{result}(AB)} \\ & = Zw/p + \text{result}(A) + \text{result}(AB) \end{aligned}$$

Cache Optimization (extended discussion)

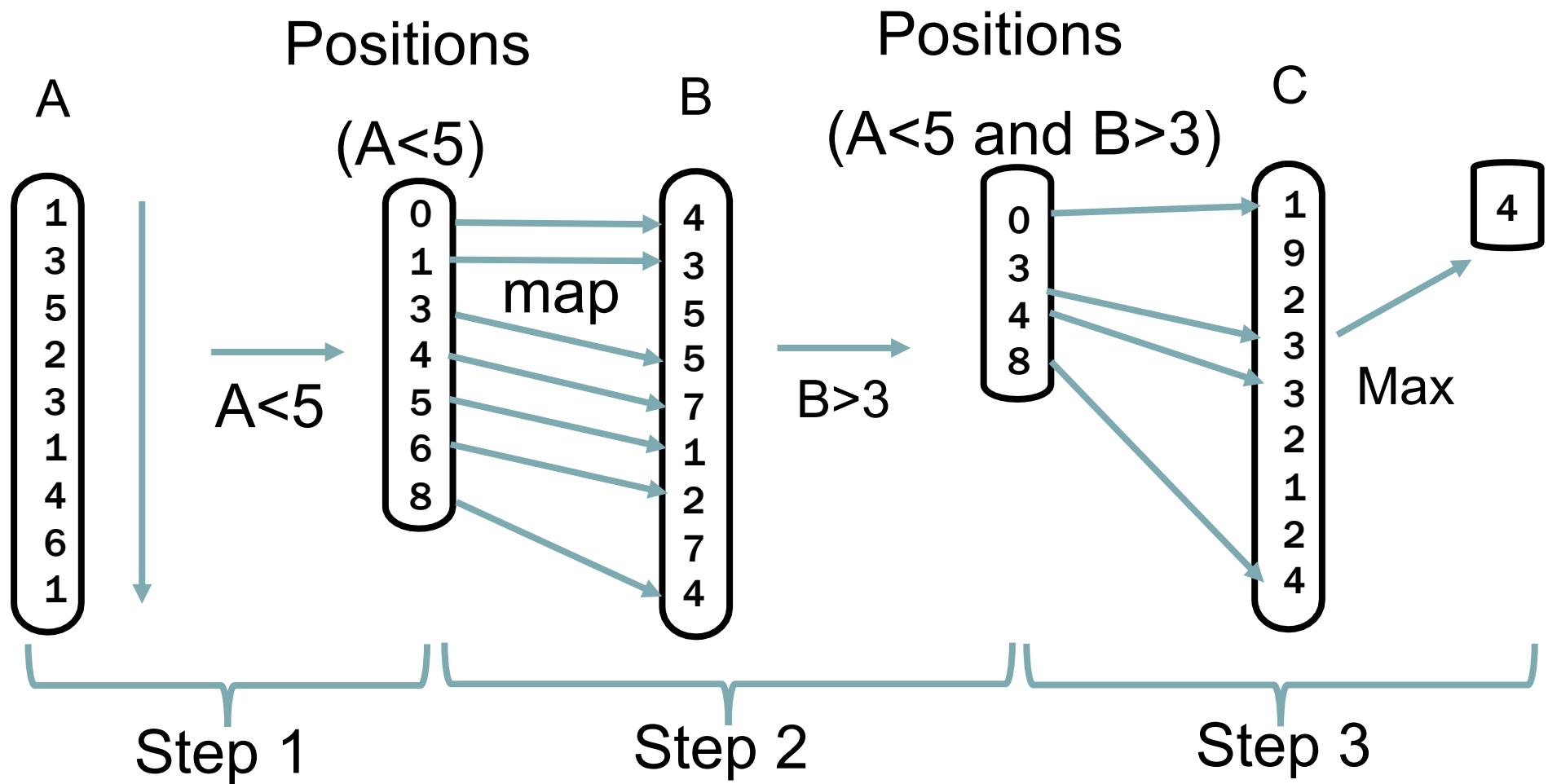
CACHE OPTIMIZATION: VECTOR AT A TIME

Modern column stores (e.g., VectorWise) employ **vector-at-a-time** procedure, utilizing cache locality. This is called **vectorized processing**.



EXAMPLE

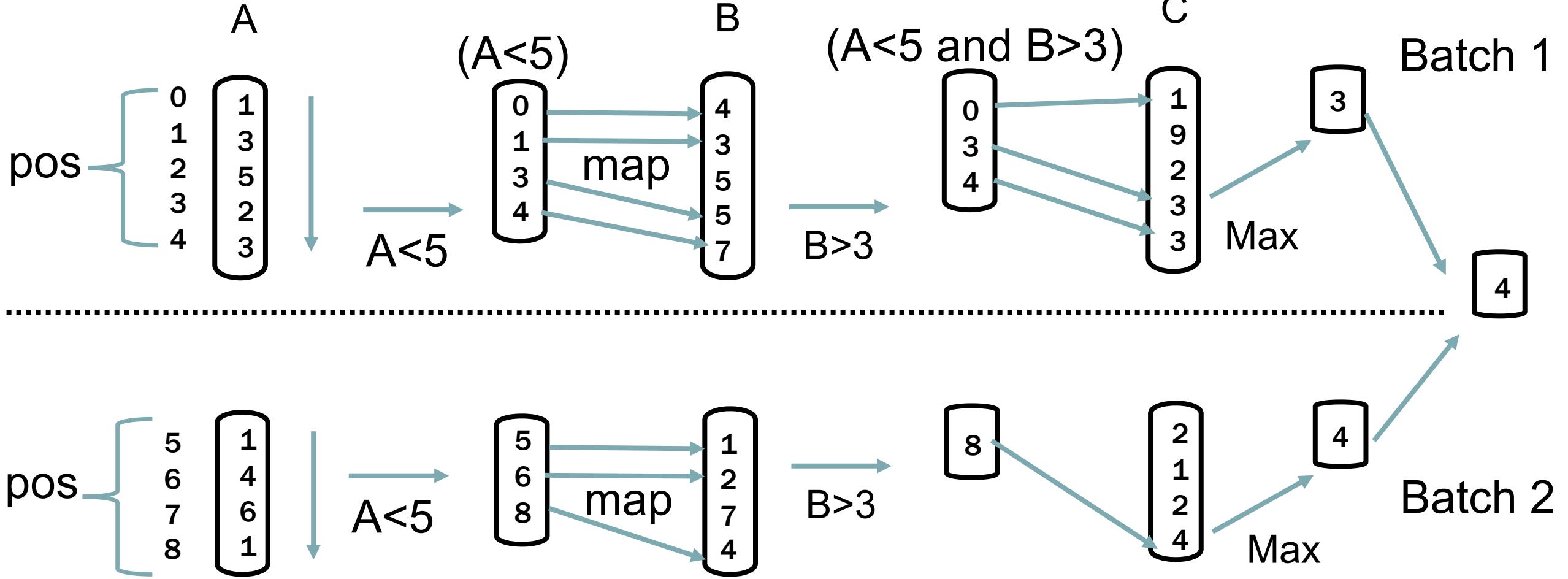
Column at a time



EXAMPLE

Vector at a time

Vector size=5



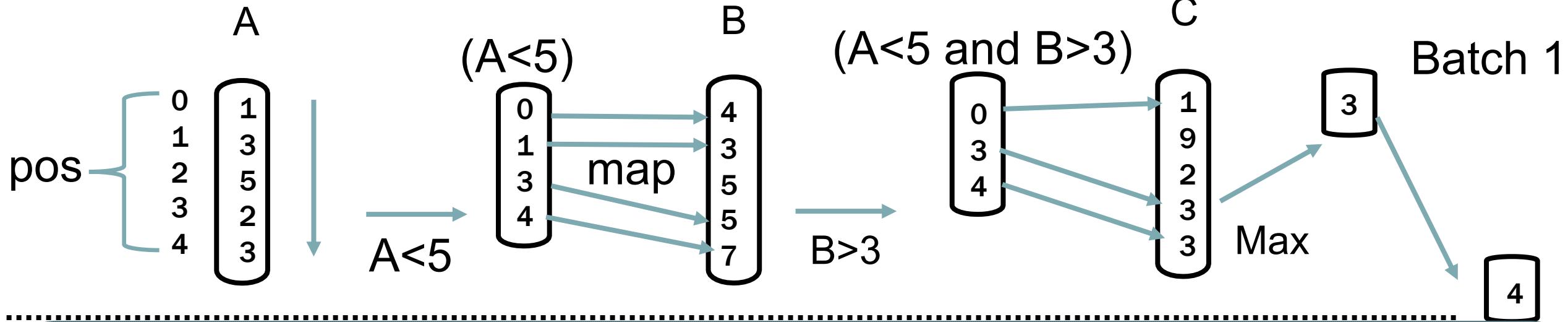
What's the benefits of vectorized processing?



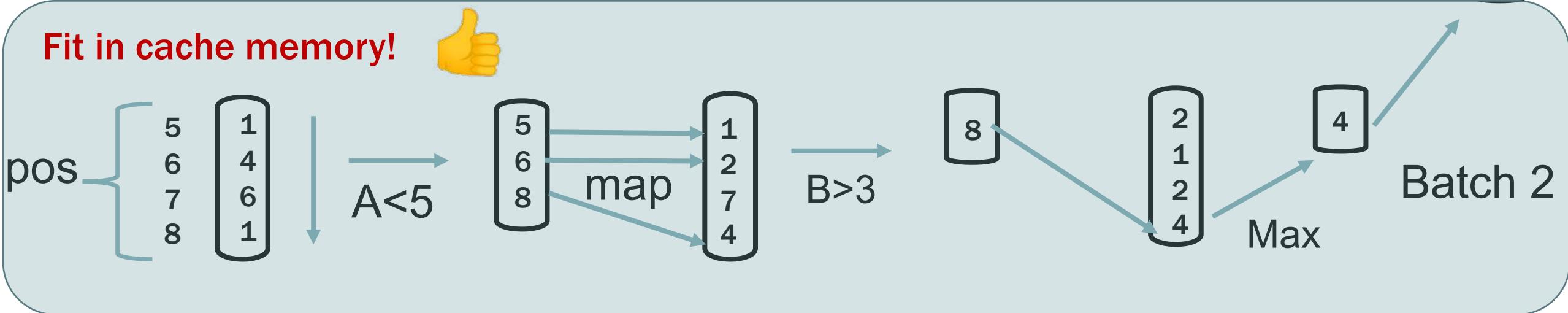
EXAMPLE

Vector at a time

Vector size=5

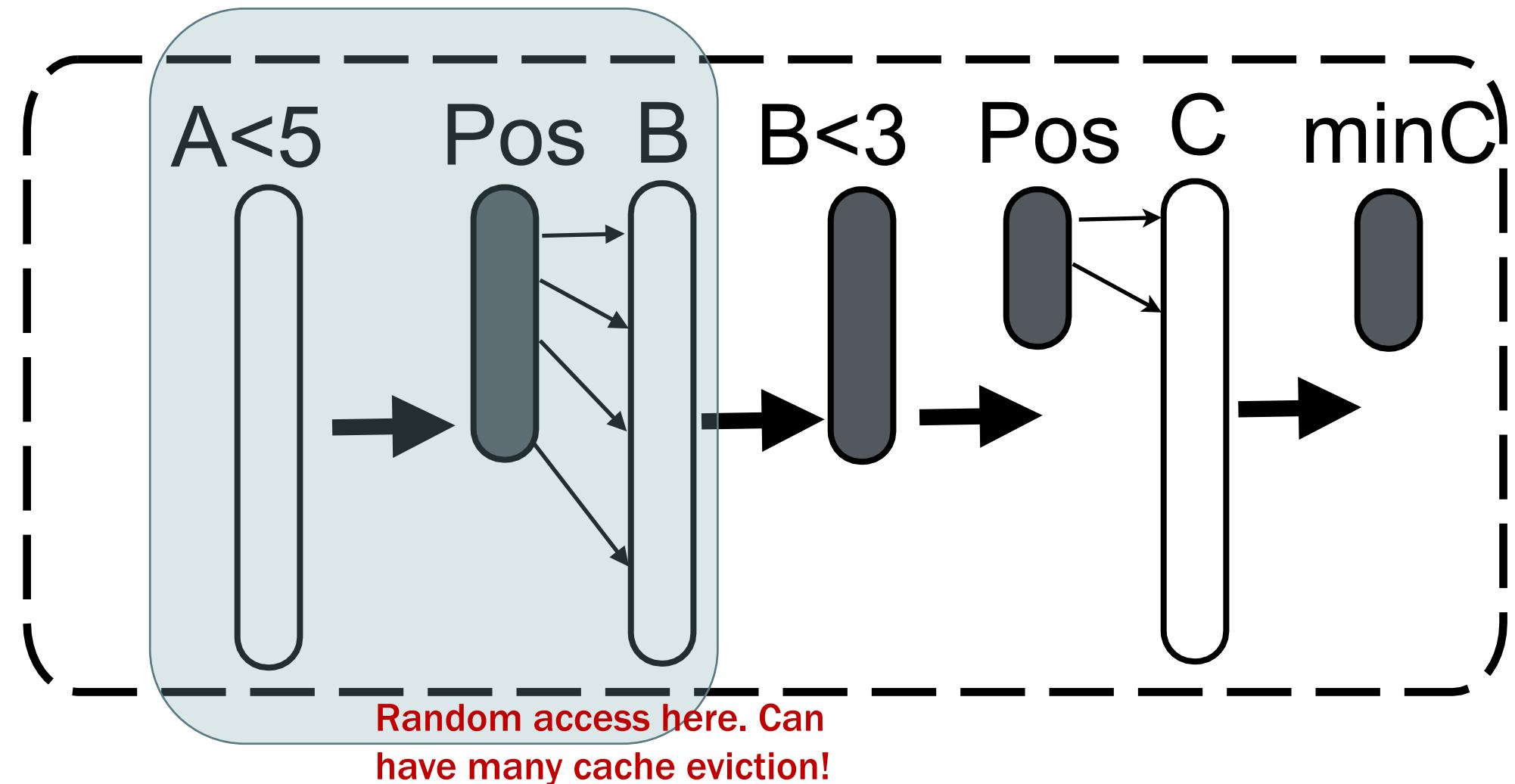


Fit in cache memory!



VECTOR AT A TIME IS MORE CACHE FRIENDLY

Column at a time



BENEFITS OF VECTORIZED PROCESSING

- ❑ It is recommended to set the vector size to be smaller than cache size (L1 cache) to allow for auxiliary data structure. Hence, it guarantees each batch has a good cache locality.
- ❑ For some queries, we can get the results on the fly.

What can be a possible issue of a column store?



BIG DATA MANAGEMENT

CE/CZ4123

COLUMN STORE

PART II

Siqiang Luo

Assistant Professor

IN PREVIOUS LECTURES

- ❑ What is the main design of column stores?
- ❑ Why we need column stores?
- ❑ How to conduct queries with column store?

IN THIS LECTURE

- ❑ How to conduct updates with column store?
- ❑ More techniques (ideas) about optimizing column stores
 - ❑ Compression
 - ❑ Shared scans
 - ❑ Zone maps
 - ❑ Sorting
 - ❑ Indexing

STARTING QUESTION

The limitations of column stores?

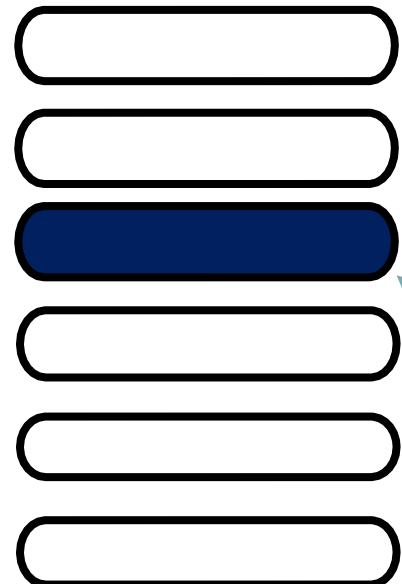


**Updates in
column stores**

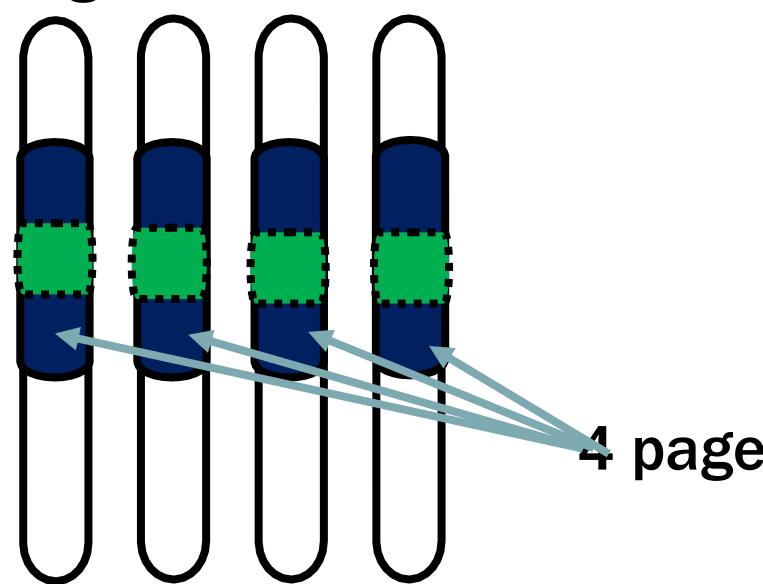
UPDATE WITH COLUMN STORE

- We assume One Row occupies a Page for ease of illustration.

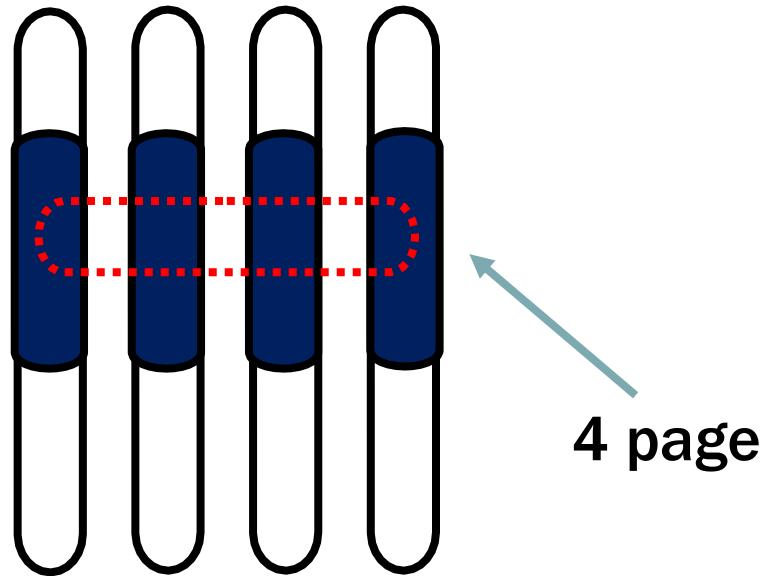
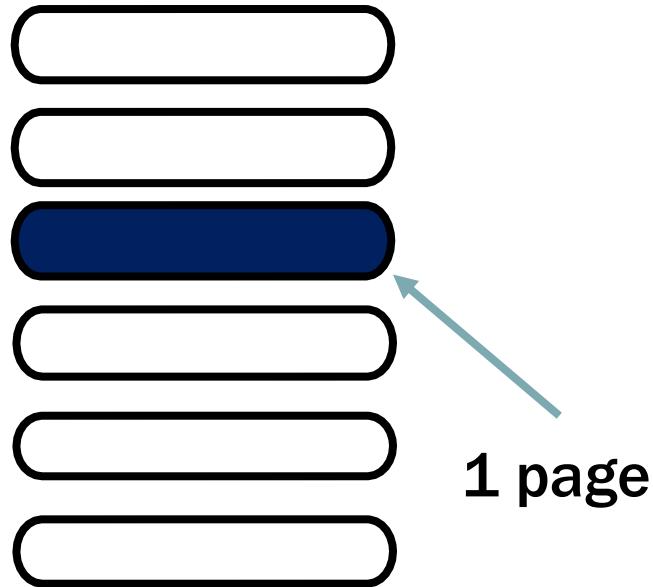
Updating the 3rd row in row store



Updating the 3rd row in column store



UPDATE WITH COLUMN STORE



- ❑ Row store: when we know which row to update, we only update the page containing the tuple
- ❑ Column store: when we know which row to update, we still need to update N pages intersecting with the tuple. (note: N is the number of columns)



One issue with column store

Column store is **NOT** update-friendly,
if the whole row needs to be updated.

SUMMARY OF COLUMN STORE

- ❑ Column store is often great for read-heavy workloads (e.g., analytical tasks)
- ❑ Row store is often great for updates, but column store's update performance is also improving with latest technologies.
- ❑ In system design, no system is perfect for every kind of workload (e.g., analytical workloads, update-heavy workloads). Always design your data system based on the targeted workloads.

**Join in
column stores**

Table R

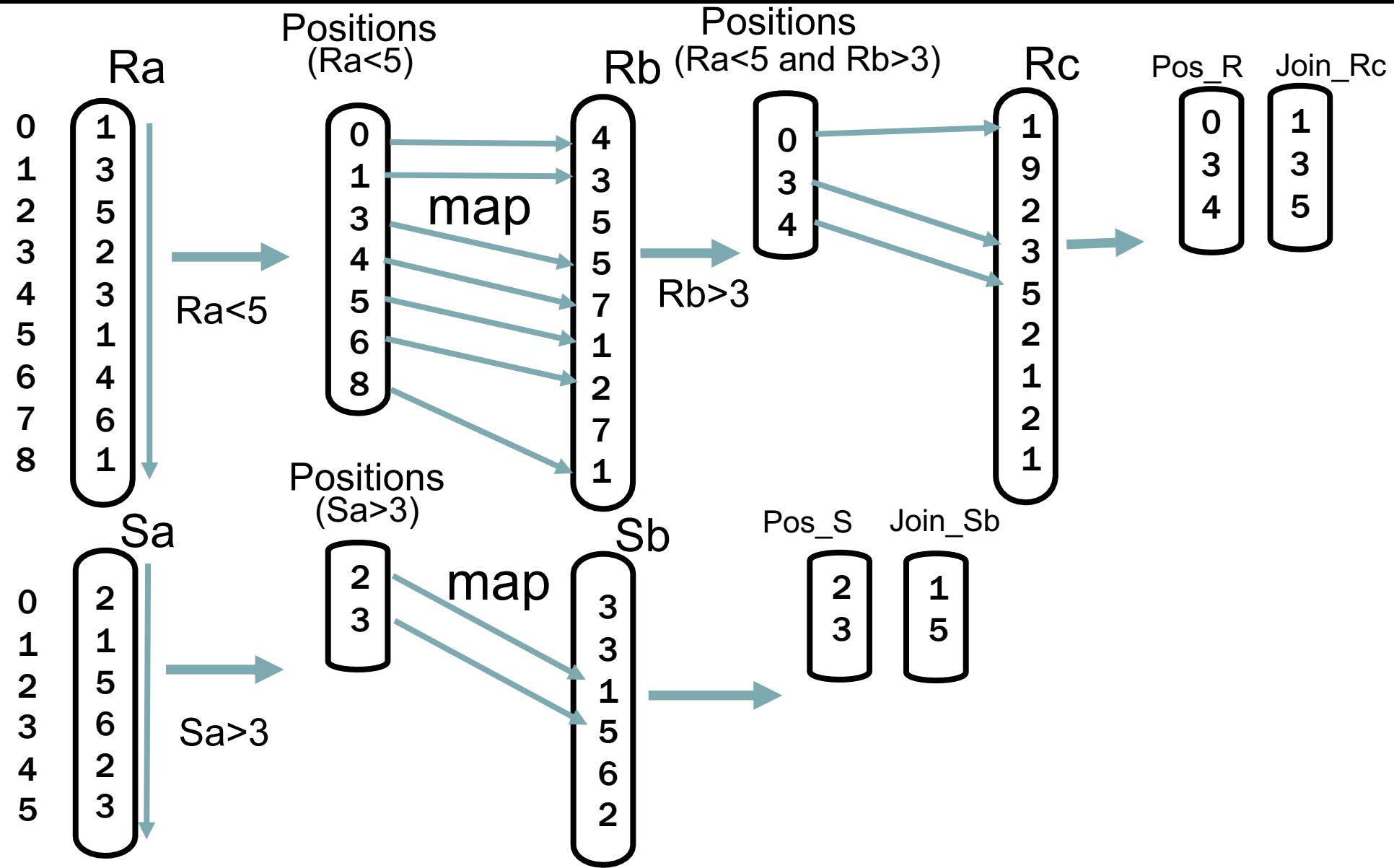
positions	Ra	Rb	Rc
0	1	4	1
1	3	3	9
2	5	5	2
3	2	5	3
4	3	7	5
5	1	1	2
6	4	2	1
7	6	7	2
8	1	1	1

```
SELECT SUM(R.a)
FROM R,S
WHERE R.c=S.b and R.a<5 and
R.b>3 and S.a>3
```

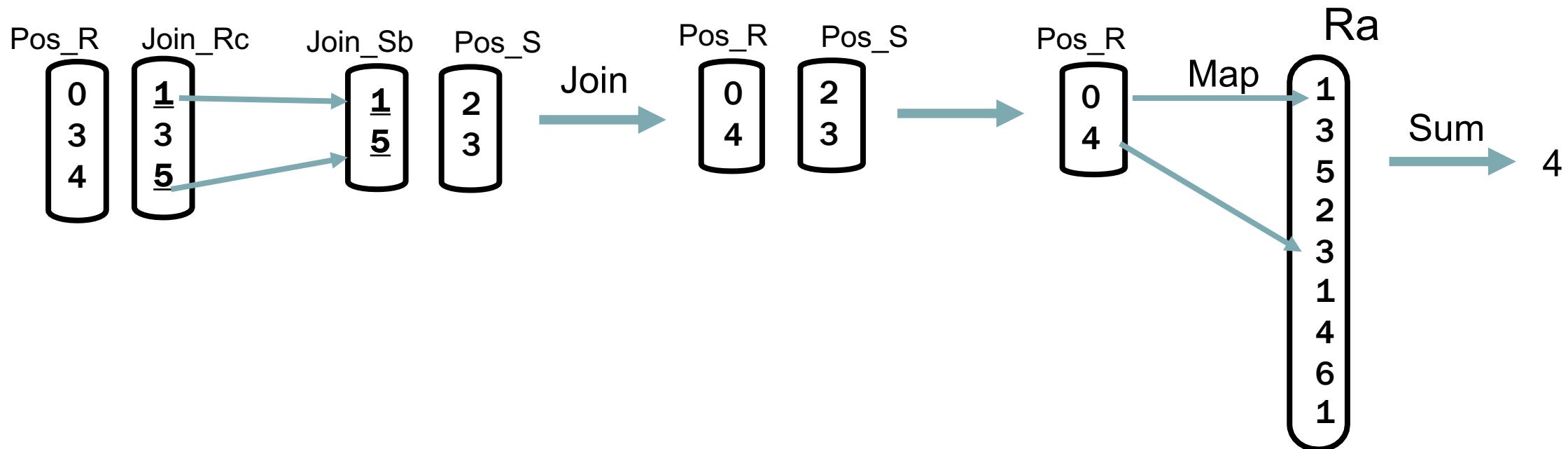
Table S

positions	Sa	Sb
0	2	3
1	1	3
2	5	1
3	6	5
4	2	6
5	3	2

COLUMN-BASED FILTERING



JOIN & AGGREGATE



**More techniques to enhance
column stores**

COMPRESSION

Original data

8 bytes
width

value1
value2
value3
value1
value1
value4
value2
value3
value5
...

Compressed Dictionary

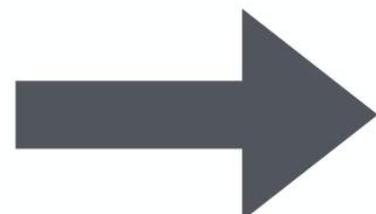
3 bits
width

001
010
011
001
001
100
010
011
101
...

8 bytes
width

value1
value2
value3
value4
value5

If there are only 5 possible values in a column, then 3 bits encoding can be applied.

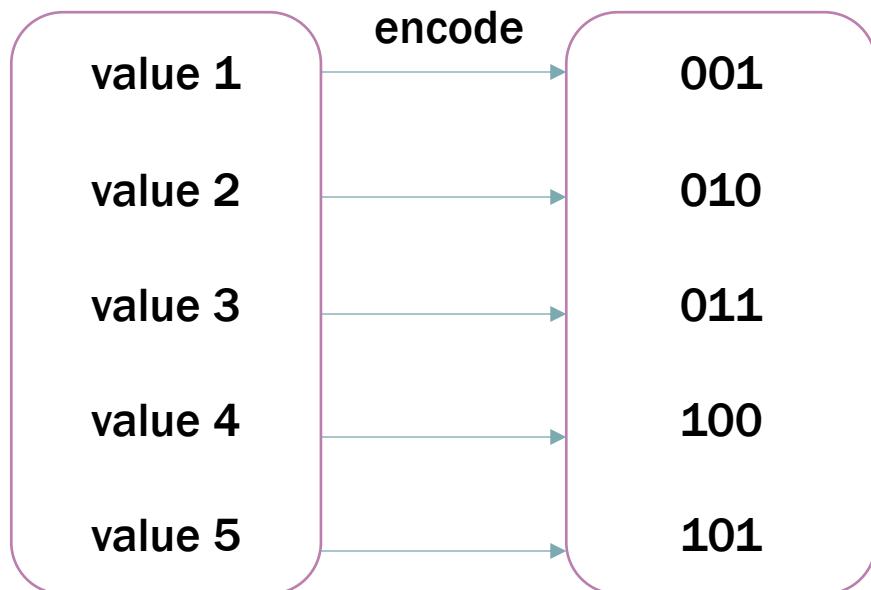


How many bits?



COMPRESSION

If there are only 5 possible values in a column, then 3 bits encoding can be applied.



Original data

8 bytes width

value1
value2
value3
value1
value1
value4
value2
value3
value5
...



Compressed

3 bits width

001
010
011
001
001
100
010
011
101
...

Dictionary

8 bytes width

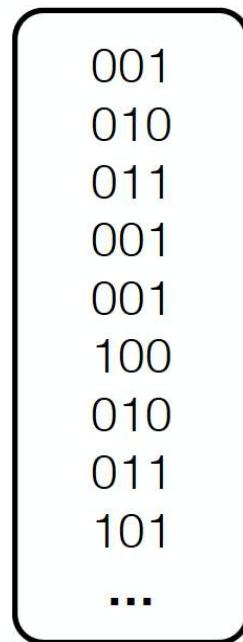
value1
value2
value3
value4
value5

Compressed ~21x

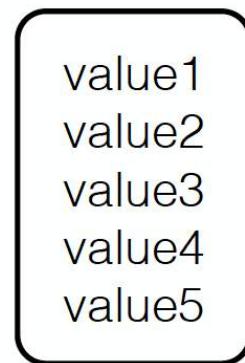
COMPRESSION

Compressed Dictionary

3 bits
width



8 bytes
width



**How do we
process data?**



**Check “=100”?
Check “<100”?**

COMPRESSION

In general, if there are N possible values in the column, we can use $\log N$ bits to encode (compress) the column values

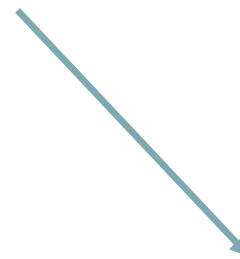
Comparison of data can be operated on the compressed data
For “= 100”, we can process the data by:

- (1) find the compressed code for 100 from the dictionary; if not found, then must be “unequal”
- (2) Otherwise compare with the code directly in the compressed column

COMPRESSION

In general, if there are N possible values in the column, we can use $\log N$ bits to encode (compress) the column values

Comparison of data can be operated on the compressed data
For “ <100 ”, we can apply **order-preserving encoding**.



value 1 < value 2 is equal to code 1 < code 2

SHARED SCANS

loop fusion

```
for(i=0;i<n;i++)  
    min = a[i]<min ? a[i] : min  
  
for(i=0;i<n;i++)  
    max = a[i]>max ? a[i] : max
```

```
for(i=0;i<n;i++)  
    min = a[i]<min ? a[i] : min  
    max = a[i]>max ? a[i] : max
```

Which one is better?



SHARED SCANS

loop fusion

```
for(i=0;i<n;i++)  
    min = a[i]<min ? a[i] : min  
  
for(i=0;i<n;i++)  
    max = a[i]>max ? a[i] : max
```

Two passes of data

```
for(i=0;i<n;i++)  
    min = a[i]<min ? a[i] : min  
    max = a[i]>max ? a[i] : max
```

One pass of data



SHARED SCAN

Real-case code in C++

```
7 # define LENGTH 500000000
8 int array[LENGTH];
9 int main(int argc, char* argv[]){
10     for(int i = 0; i < LENGTH; i++){
11         array[i] = rand();
12     }
13     int min, max;
14     struct timespec t1, t2;
```

Two Pass Solution

```
16     clock_gettime(CLOCK_MONOTONIC, &t1);
17     min = RAND_MAX;
18     max = 0;
19     for(int i = 0; i < LENGTH; i++){
20         min = array[i]<min?array[i]:min;
21     }
22     for(int i = 0; i < LENGTH; i++){
23         max = array[i]>max?array[i]:max;
24     }
25     clock_gettime(CLOCK_MONOTONIC, &t2);
26     double time = (t2.tv_sec - t1.tv_sec) +
27         (double)(t2.tv_nsec - t1.tv_nsec) / 1000000000;
28     std::cout << "Two pass time: " << time << " s" << std::endl;
```



One Pass Solution

```
30     clock_gettime(CLOCK_MONOTONIC, &t1);
31     min = RAND_MAX;
32     max = 0;
33     for(int i = 0; i < LENGTH; i++){
34         min = array[i]<min?array[i]:min;
35         max = array[i]>max?array[i]:max;
36     }
37     clock_gettime(CLOCK_MONOTONIC, &t2);
38     time = (t2.tv_sec - t1.tv_sec) +
39         (double)(t2.tv_nsec - t1.tv_nsec) / 1000000000;
40     std::cout << "One pass time: " << time << " s" << std::endl;
```

SHARED SCAN

Real-case code in C++

```
7 # define LENGTH 500000000
8 int array[LENGTH];
9 int main(int argc, char* argv[]){
10     for(int i = 0; i < LENGTH; i++){
11         array[i] = rand();
12     }
13     int min, max;
14     struct timespec t1, t2;
```

Two Pass Solution

```
16     clock_gettime(CLOCK_MONOTONIC, &t1);
17     min = RAND_MAX;
18     max = 0;
19     for(int i = 0; i < LENGTH; i++){
20         min = array[i]<min?array[i]:min;
21     }
22     for(int i = 0; i < LENGTH; i++){
23         max = array[i]>max?array[i]:max;
24     }
25     clock_gettime(CLOCK_MONOTONIC, &t2);
26     double time = (t2.tv_sec - t1.tv_sec) +
27         (double)(t2.tv_nsec - t1.tv_nsec) / 1000000000;
28     std::cout << "Two pass time: " << time << " s" << std::endl;
```

Execution result

```
Two pass time: 2.20189 s
One pass time: 1.34021 s
```

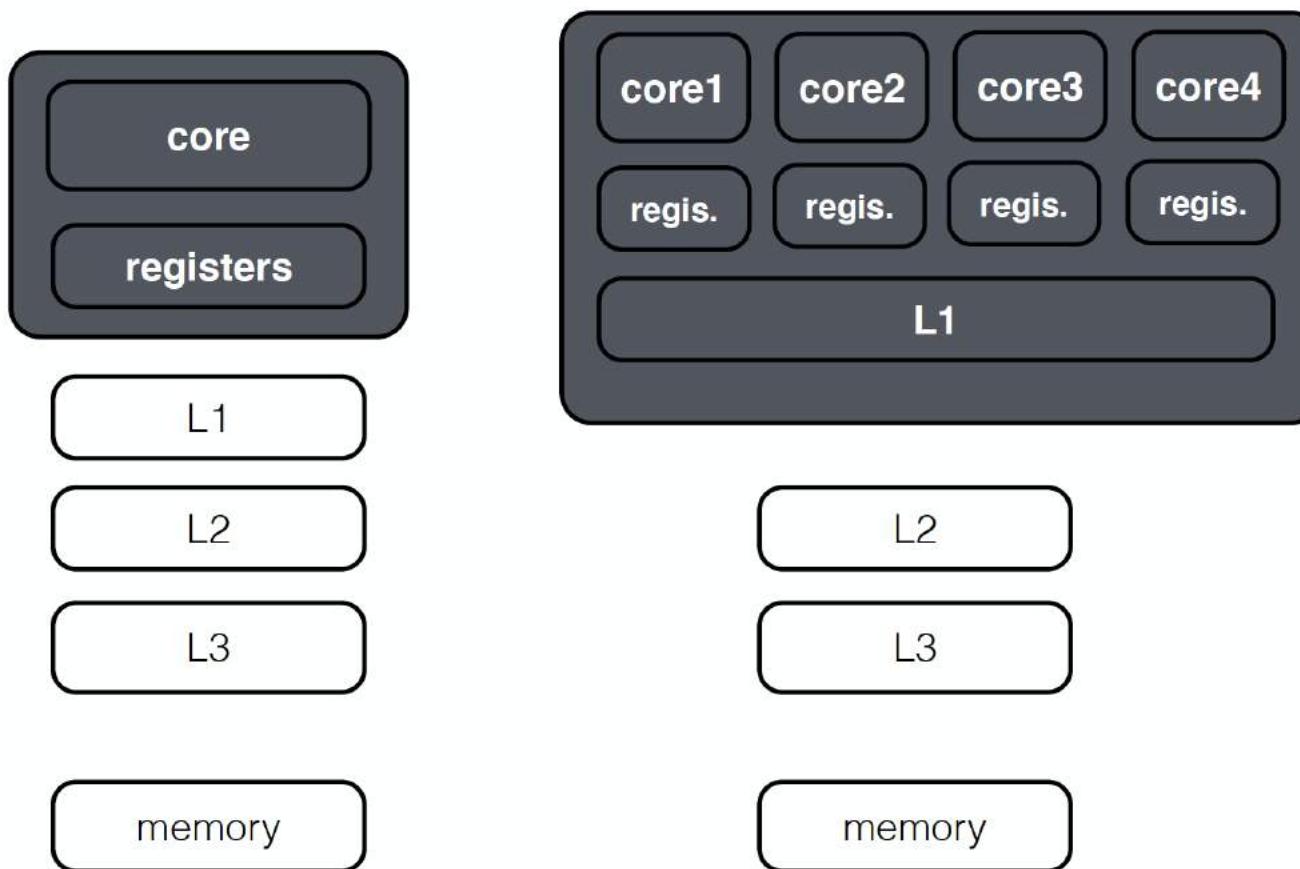


One Pass Solution

```
30     clock_gettime(CLOCK_MONOTONIC, &t1);
31     min = RAND_MAX;
32     max = 0;
33     for(int i = 0; i < LENGTH; i++){
34         min = array[i]<min?array[i]:min;
35         max = array[i]>max?array[i]:max;
36     }
37     clock_gettime(CLOCK_MONOTONIC, &t2);
38     time = (t2.tv_sec - t1.tv_sec) +
39         (double)(t2.tv_nsec - t1.tv_nsec) / 1000000000;
40     std::cout << "One pass time: " << time << " s" << std::endl;
```

SHARED SCANS

Modern CPUs can do more than 1 tasks at the same time.



SHARED SCANS



can work in parallel

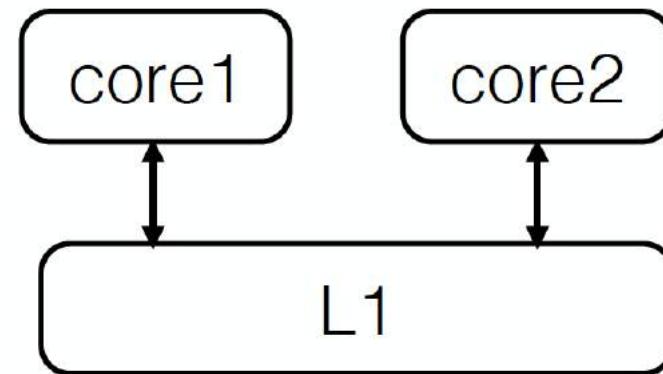
SHARED SCANS

In column store, multiple queries scanning the same column.

We can always maximize the CPU core usage and minimize the data scanned.
This idea is called **shared scan**.

Query 1: Select > 5 (handled by core 1)

Query 2: Select < 8 (handled by core 2)



A
1
3
5
2
3
1
4
6
1

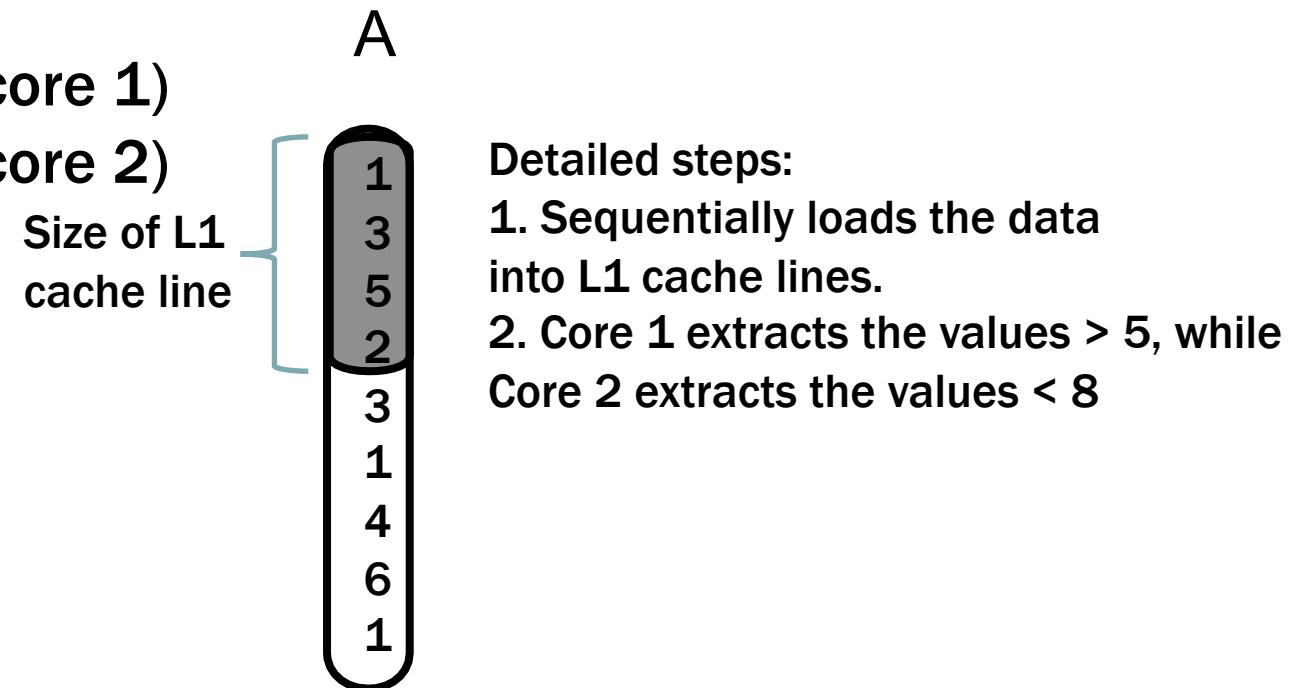
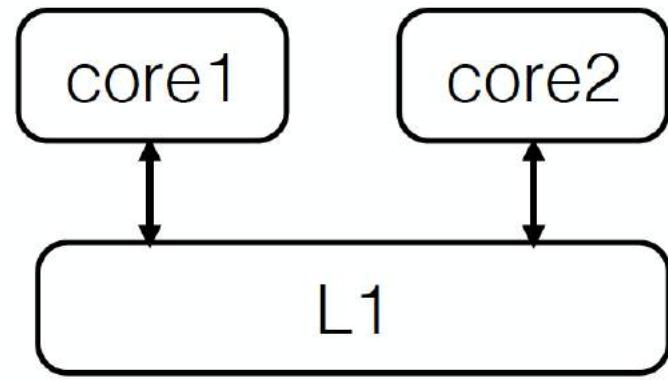
SHARED SCANS

In column store, multiple queries scanning the same column.

We can always maximize the CPU core usage and minimizing the scanning data. This idea is called **shared scan**.

Query 1: Select > 5 (handled by core 1)

Query 2: Select < 8 (handled by core 2)



QUESTION

What if queries do not come at the same time?



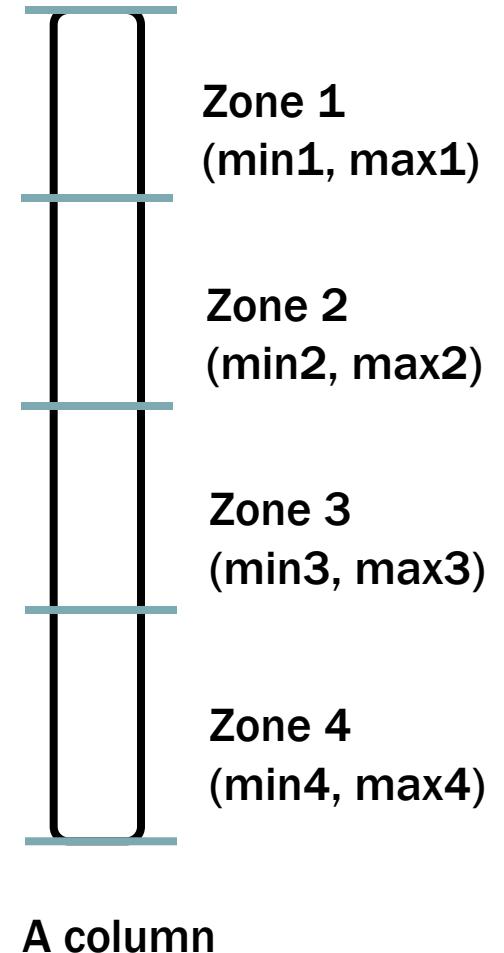
Gather queries in a batch

Schedule the queries on same data to run in parallel

Each query gets a thread/core

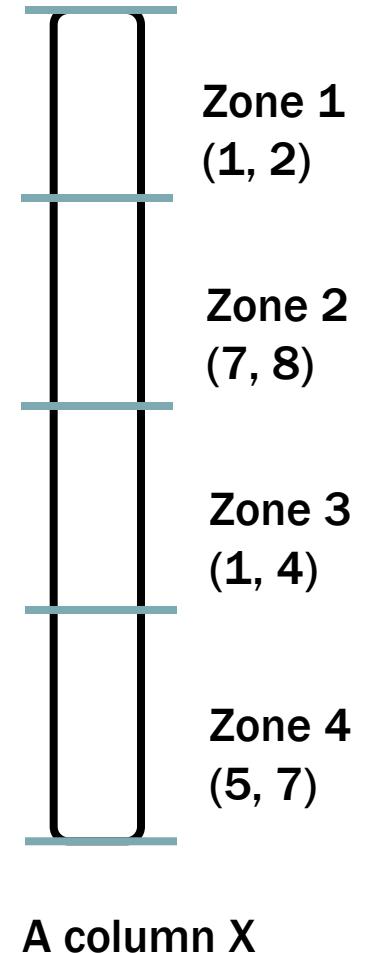
ZONE MAP

- ❑ A common way to help column scan is the zone map.
- ❑ It separates a column into “zones”, each is computed with max and min
- ❑ In filtering, some zones can be skipped.



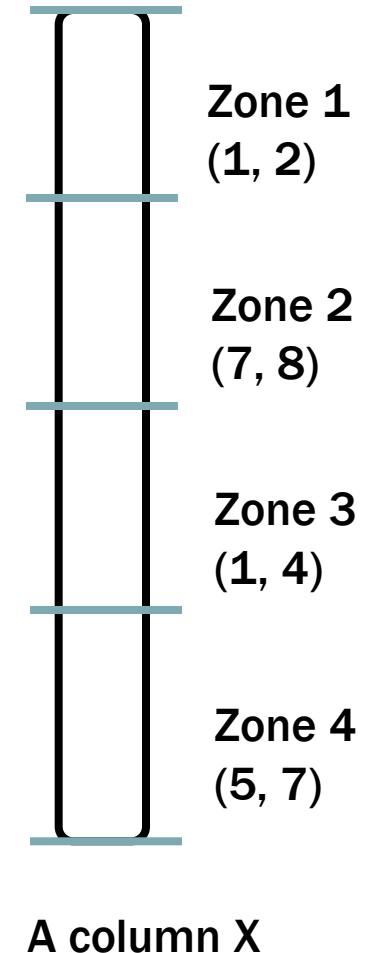
ZONE MAP

- Select from T where $X > 3$ and $X < 6$
- Which zones need to be scanned?



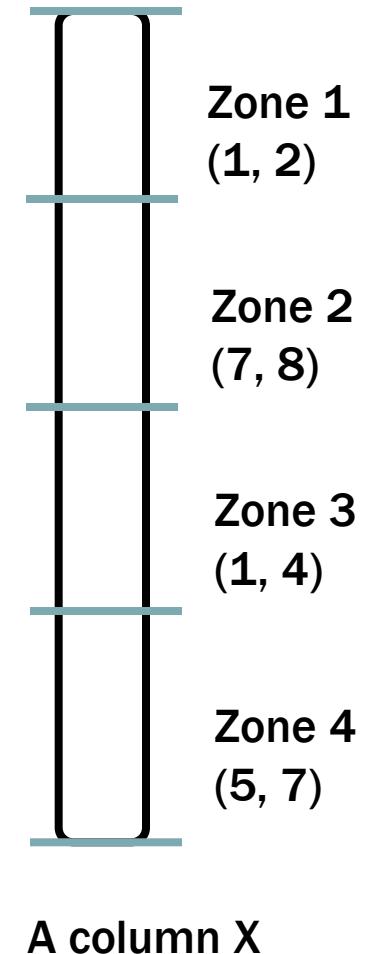
ZONE MAP

- Select from T where $X > 3$ and $X < 6$
- Which zones need to be scanned?
- We only need to scan Zone3 and Zone4 because
 - $[1, 4]$ overlaps with $(3, 6)$
 - $[5, 7]$ overlaps with $(3, 6)$
 - $[1, 2]$ does not overlap with $(3, 6)$
 - $[7, 8]$ does not overlap with $(3, 6)$



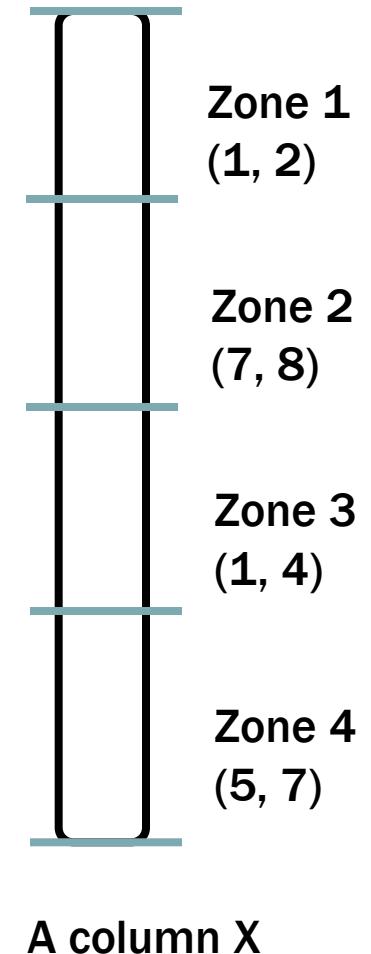
ZONE MAP

- Select from T where $X > 3$ and $X < 6$
- Which zones need to be scanned?
- We only need to scan Zone3 and Zone4 because
 - $[1, 4]$ overlaps with $(3, 6)$
 - $[5, 7]$ overlaps with $(3, 6)$
 - $[1, 2]$ does not overlap with $(3, 6)$
 - $[7, 8]$ does not overlap with $(3, 6)$
- Usually, a zone is of a page size
- We skip the scanning of two pages



ZONE MAP

- How to store min and max from each zones?
- They are stored together, probably in a few disk pages.
- These zone map pages can be loaded into memory when system is started
- So the additional cost of reading zone maps is very low



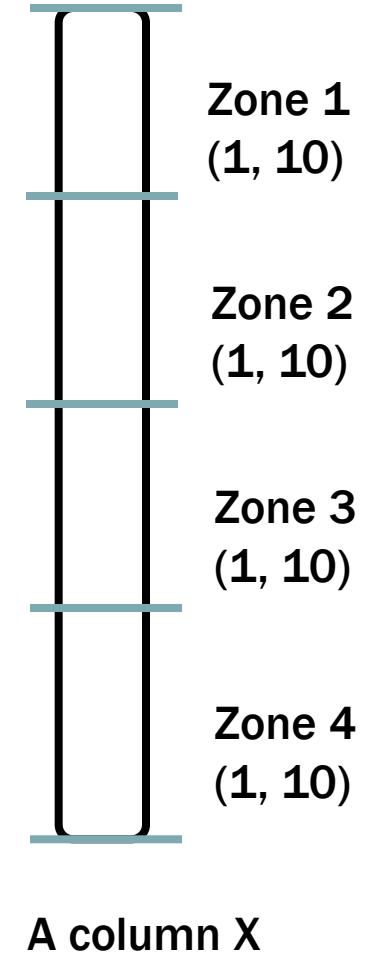
ZONE MAP

- Is zone map always effective? Why?



ZONE MAP

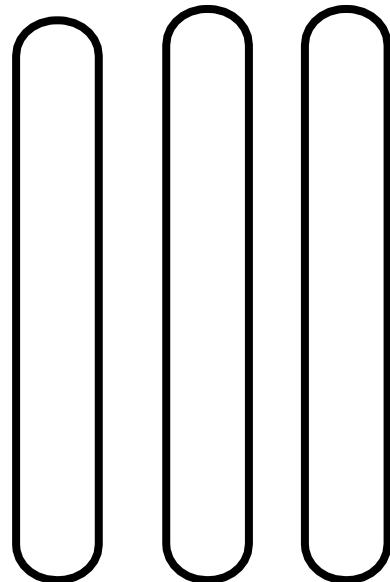
- Is zone map always effective? Why?
- Data can be uniformly distributed.



ENHANCED WITH SORTING

SELECT max(C) FROM T WHERE A>10 and A<40 and B>20

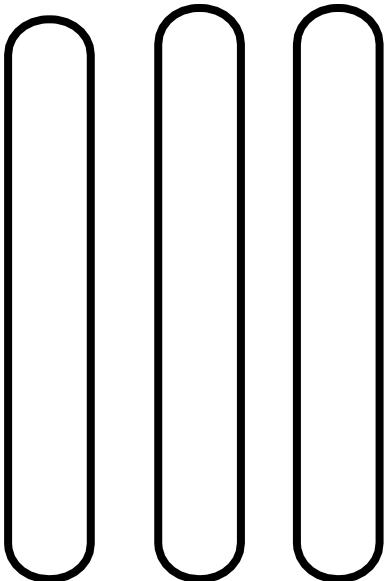
A B C



ENHANCED WITH SORTING

SELECT max(C) FROM T WHERE A>10 and A<40 and B>20

A B C

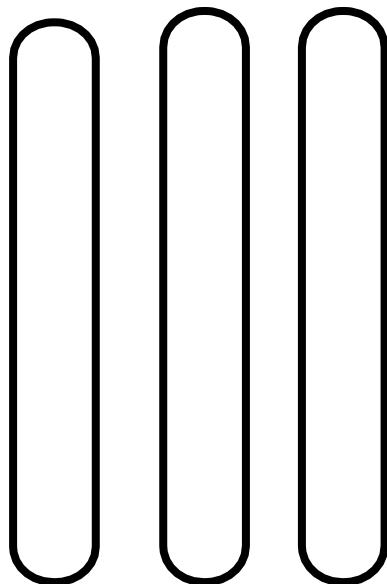


- 1) Scan A and select A in (10, 40)
- 2) Among those positions returned in 1), scan B and select B>20
- 3) Among those positions returned in 2), select max(C)

ENHANCED WITH SORTING

SELECT max(C) FROM T WHERE A>10 and A<40 and B>20

A B C

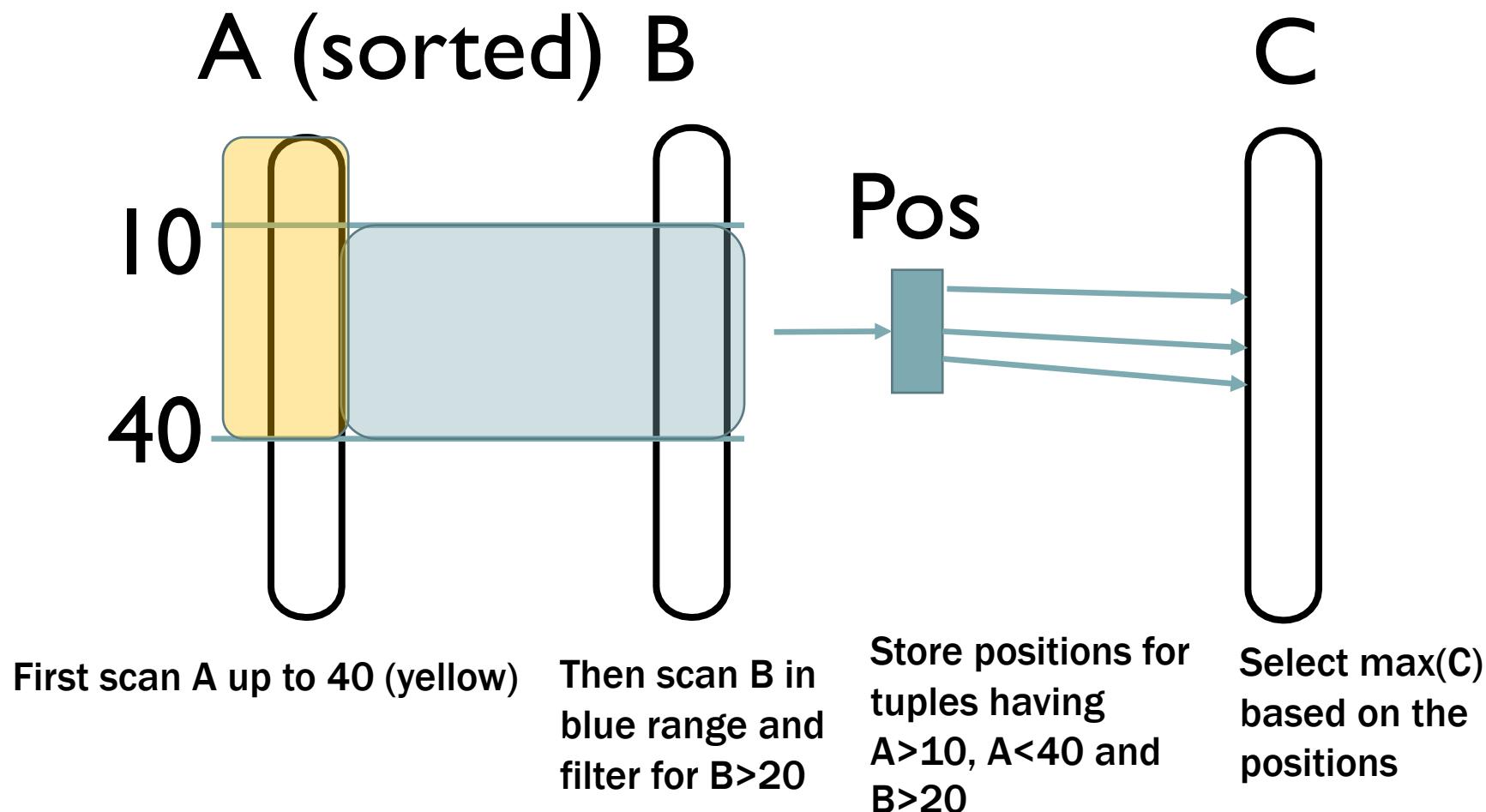


What if A is sorted?



ENHANCED WITH SORTING

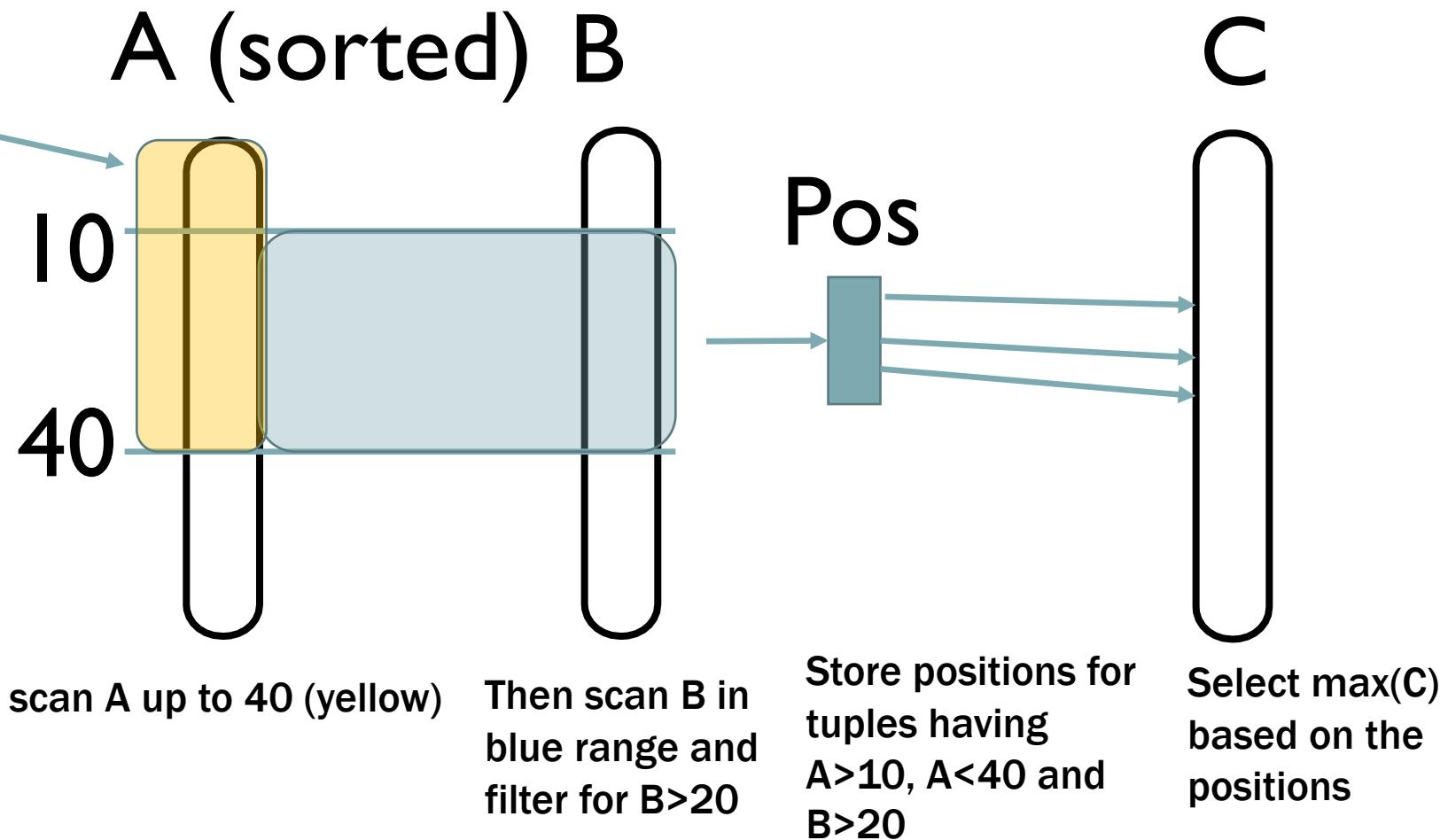
SELECT max(C) FROM T WHERE A>10 and A<40 and B>20



ENHANCED WITH SORTING

SELECT max(C) FROM T WHERE A>10 and A<40 and B>20

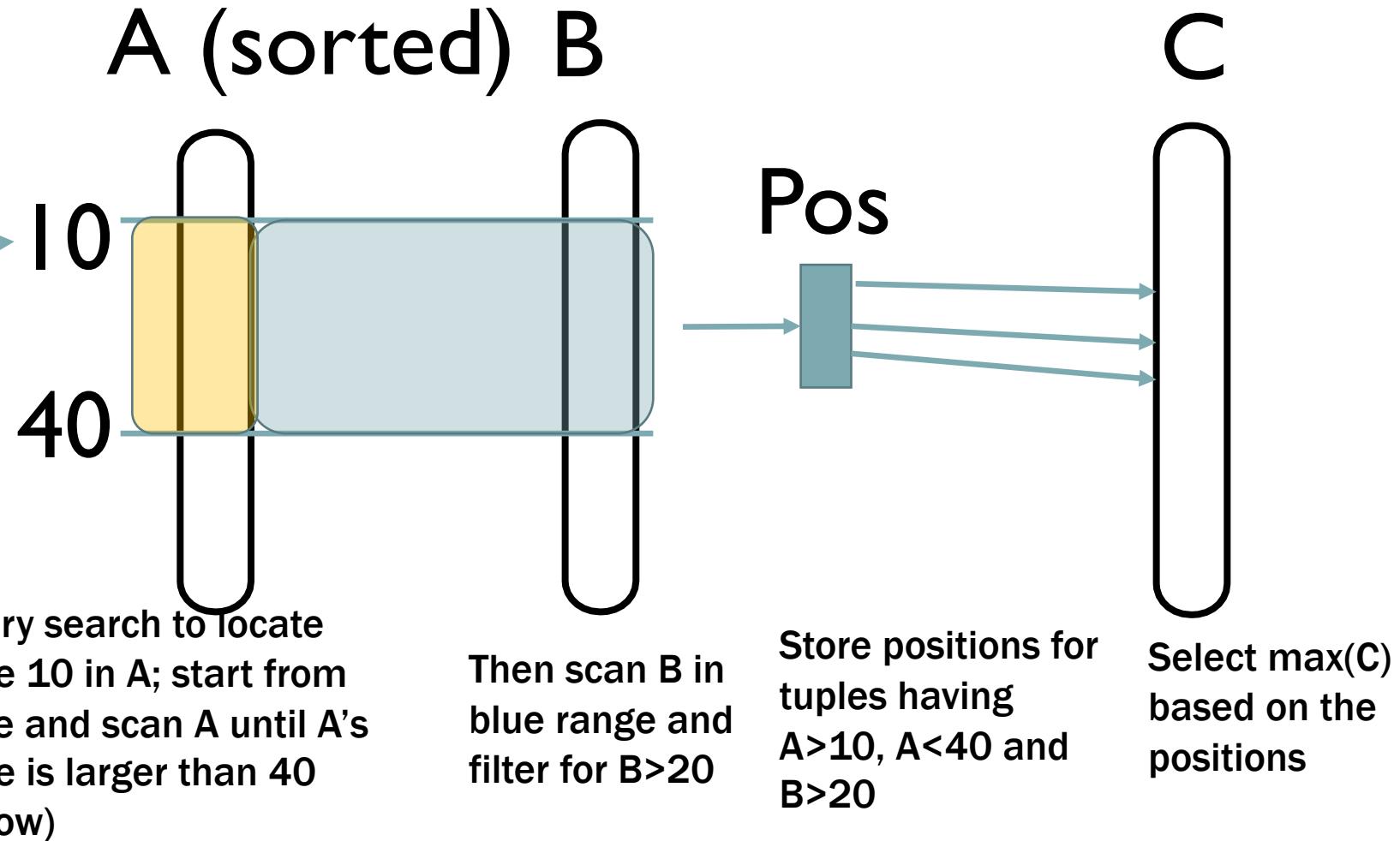
Optimization: No
need to scan from
the beginning.



ENHANCED WITH SORTING

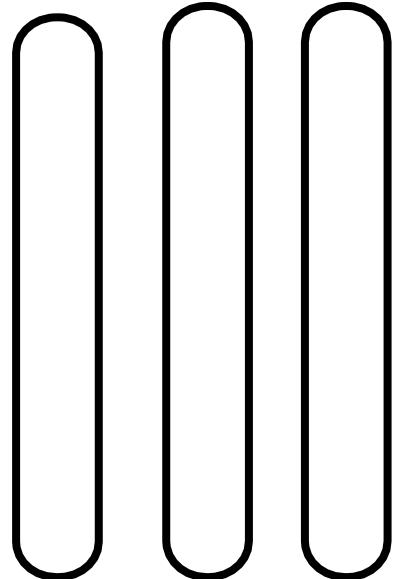
`SELECT max(C) FROM T WHERE A>10 and A<40 and B>20`

Binary search
(usually faster
because there can
be many
values < 10)



QUESTIONS FOR YOU

A B C

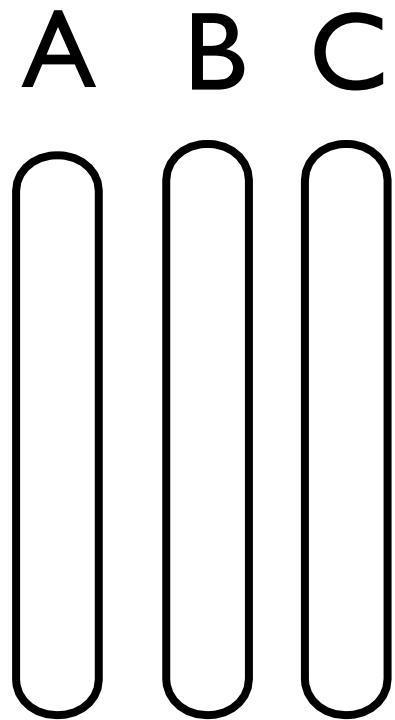


**When sorting A,
do the other columns change?**



QUESTIONS FOR YOU

A B C



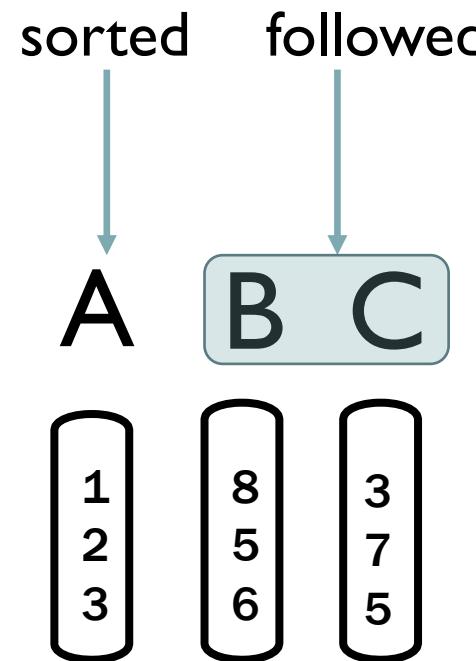
The diagram shows three vertical rectangles of equal height and width, positioned side-by-side. They represent the columns A, B, and C mentioned in the text above them.

To apply the above query scheme, the other columns need to be rearranged to follow the order of A

QUESTIONS FOR YOU

A B C

2	5	7
1	8	3
3	6	5



LIMITATION OF SORTING

- ❑ Can only sort one column (because the other columns need to follow the order of the sorted column)
- ❑ Filtering based on unsorted column is still as costly.
- ❑ Better solution: use index

ENHANCED WITH INDEX

What is an index?

ENHANCED WITH INDEX

What is an index?

Roughly speaking, in column store, an index is an affiliated data structure built on a column, and it can efficiently help locate the position of a value in the column. It is a widely used technique in data management.

ENHANCED WITH INDEX

Why do we need index?

Imaging how do you find a book in the library

Business Library

Available , Level B1: A-HG; Level B4: HJ-Z HM131.S431 2003

1. Find the bookrack corresponding to HJ-Z will help you find the book
2. Find the range for HM131

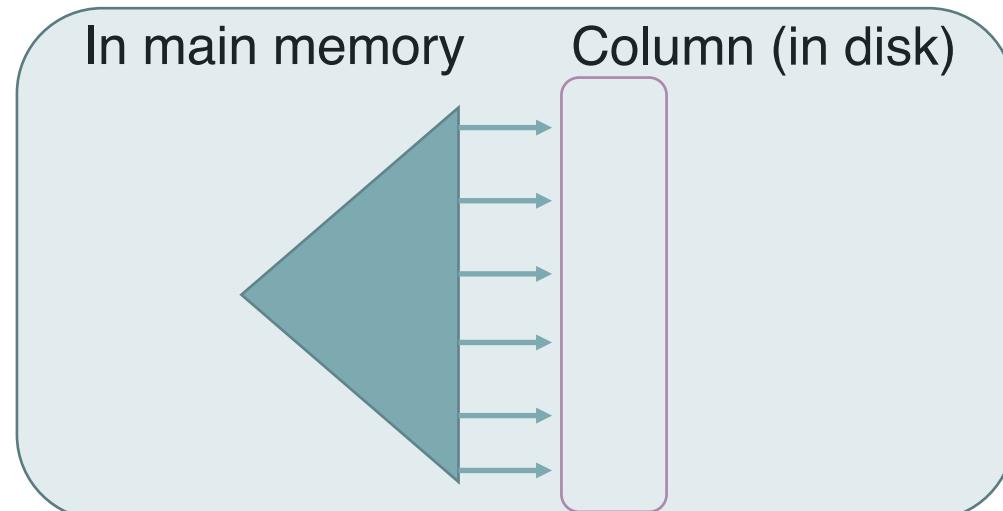
This is a kind of index

ENHANCED WITH INDEX

In column store, we can build index for a column

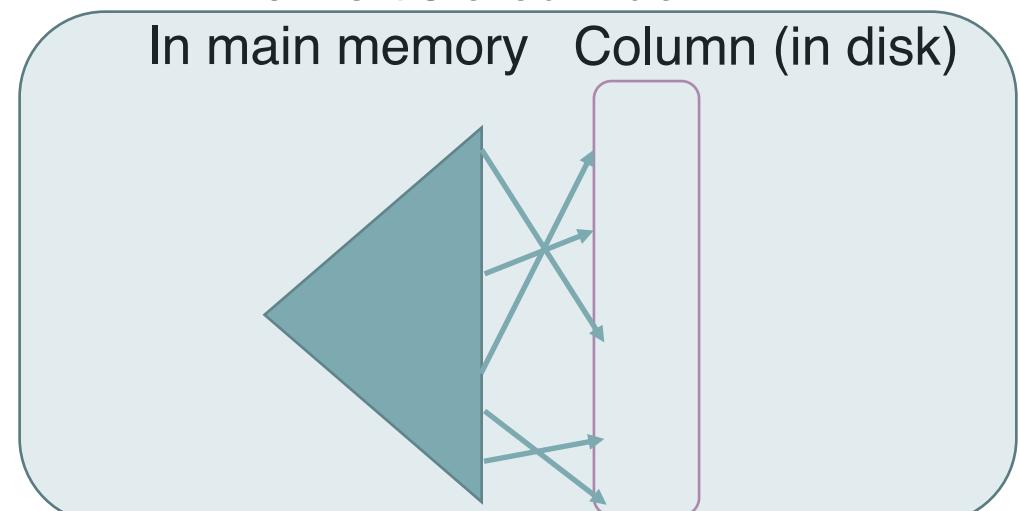
If the column is sorted

Clustered index



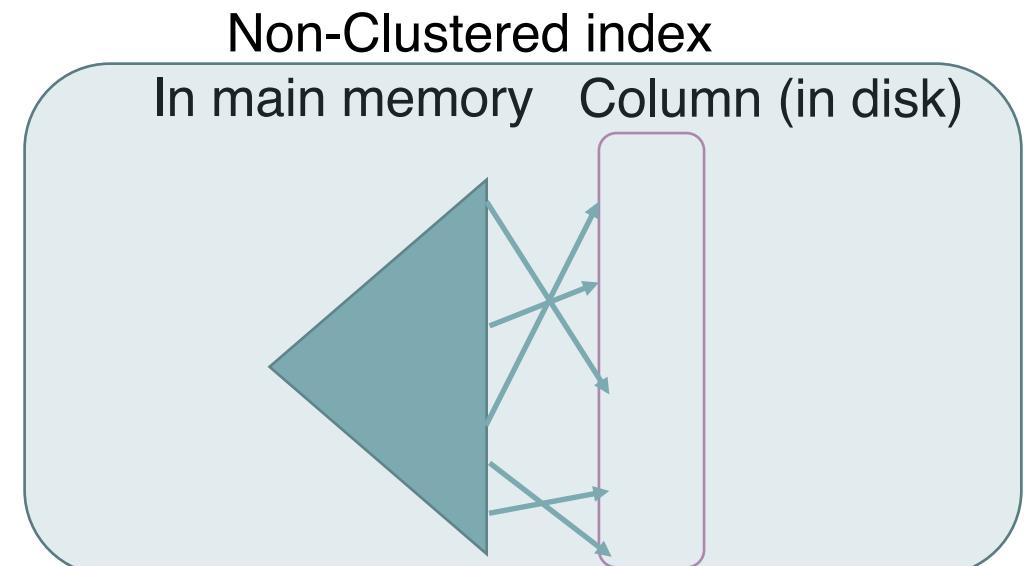
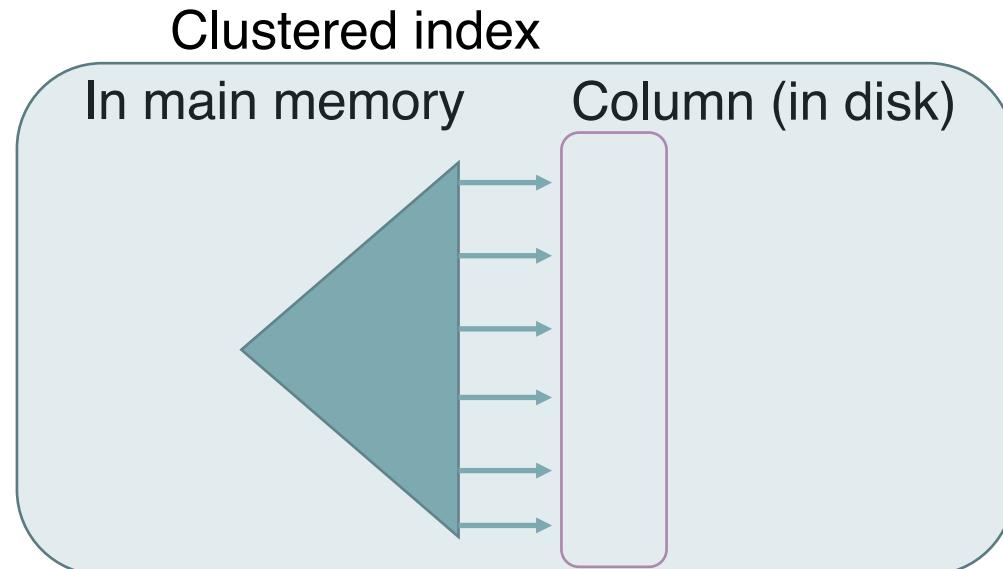
If the column is unsorted

Non-Clustered index



ENHANCED WITH INDEX

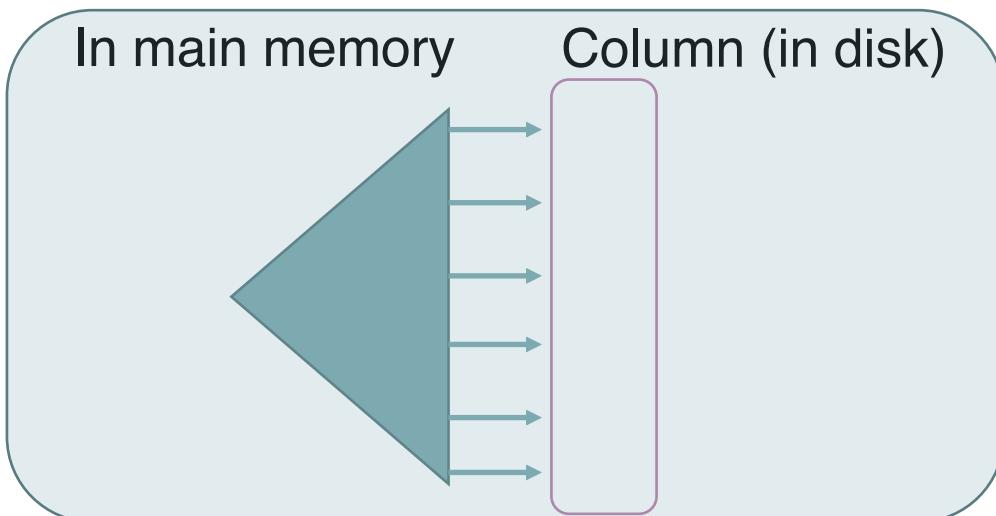
- ❑ In one table, we can apply both clustered index and non-clustered index.
- ❑ One clustered index for a table (why?)
- ❑ Can have multiple non-clustered index for a table
- ❑ Clustered index is typically faster in extracting values compared with non-clustered index (why?)



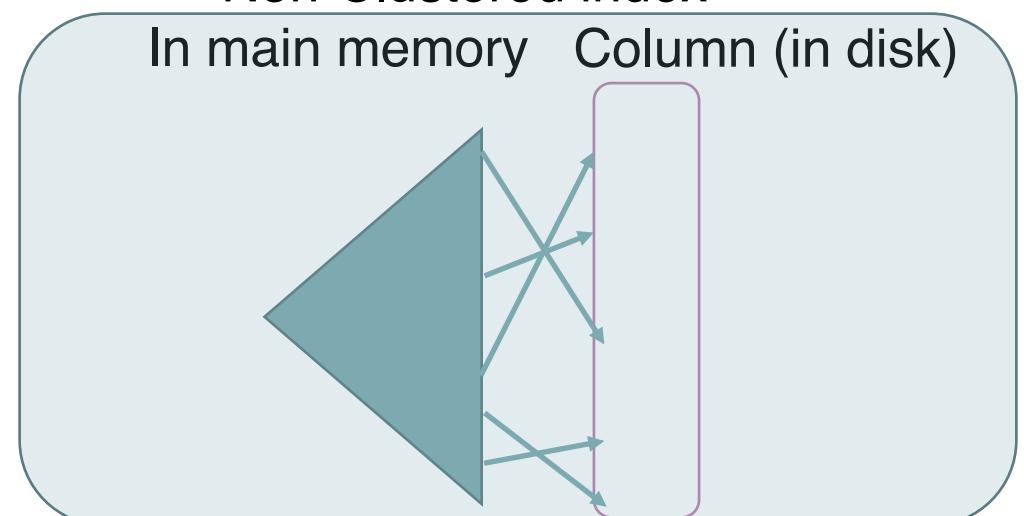
ENHANCED WITH INDEX

- Index access is much faster than data access, because index is stored in Main Memory, while data is stored in disk.
- So it is beneficial to have complicated index access to locate the exact pages where stores the data.

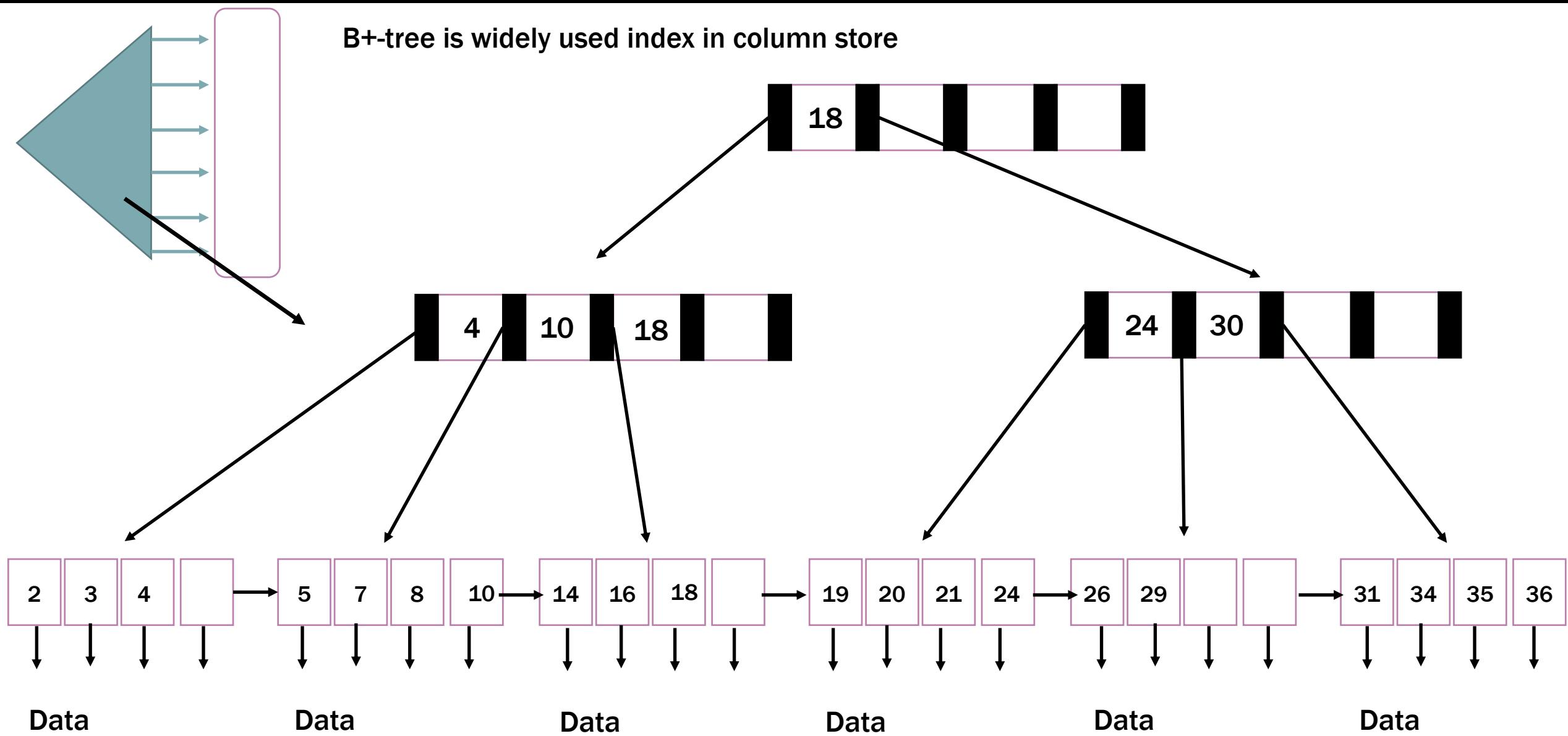
Clustered index



Non-Clustered index



ENHANCED WITH INDEX



More discussions



Hybrid layouts



Anastasia Ailamaki, EPFL
ACM SIGMOD Innovations award, 2019

PAX: store all data about a row in a single disk page but organize data in a column way inside each page

Weaving Relations for Cache Performance, VLDB 2001

**More practice for
Column Stores**



QUESTION1

Use **shared scan** to find the 2nd largest value and 2nd smallest value in an integer array A of size t. Assume that the integers in A are different.

RECAP-WHAT IS SHARED SCAN

Shared scan minimizes the scan cost. Typically, **one** pass of scan is preferable.

Think about the question... How to write the code?

LET'S START WITH WHATEVER WE HAVE

First, we can use the following code to compute the max and second max of array A.

```
max= - ∞;  
second_max= - ∞;  
  
for (i=0;i<t;i++){  
    if(A[i]>max) {second_max=max; max=A[i]; }  
    if(A[i]<= max && A[i]>second_max) second_max=A[i];  
}
```

LET'S START WITH WHATEVER WE HAVE

Second, we can use the following code to compute the min and second min of array A.

```
min= ∞;  
second_min= ∞;  
for (i=0;i<t;i++){  
    if(A[i]<min) {second_min=min; min=A[i]; }  
    if(A[i]>= min && A[i]<second_min) second_min=A[i];  
}
```

SHARED SCAN

Solution:

The idea of shared scan is to do multiple operations in one scan of the data.

```
max= - ∞;  
second_max= - ∞;  
min= ∞;  
second_min= ∞;  
for (i=0;i<t;i++){  
    if(A[i]>max) {second_max=max; max=A[i]; }  
    if(A[i]<= max && A[i]>second_max) second_max=A[i];  
    if(A[i]<min) {second_min=min; min=A[i]; }  
    if(A[i]>= min && A[i]<second_min) second_min=A[i];  
}
```

QUESTION 2

Suppose we have the following column applied with zone map of size 3.

- 1) Illustrate the information recorded for each zone.
- 2) Give the steps of using zone map to answer queries “A>4”
- 3) Give the steps of using zone map to answer queries “A>7 or A<2”

A

1
3
5
2
3
1
4
6
2

QUESTION 2

Suppose we have the following column applied with zone map of size 3.

- 1) Illustrate the information recorded for each zone.
- 2) Give the steps of using zone map to answer queries “A>4”
- 3) Give the steps of using zone map to answer queries “A>7 or A<2”

A

1	(1, 5)
3	
5	
2	
3	
1	(1, 3)
4	
6	
2	(2, 6)

**Compute the min, max for each zone
and record them in memory.**

QUESTION 2

Suppose we have the following column applied with zone map of size 3.

- 1) Illustrate the information recorded for each zone.
- 2) Give the steps of using zone map to answer queries “ $A>4$ ”
- 3) Give the steps of using zone map to answer queries “ $A>7$ or $A<2$

A
1
3
5
2
3
1
4
6
2

Diagram illustrating the zone map for column A. The values are grouped into three zones:

- Zone 1: Values 1, 3, 5 (Range: (1, 5))
- Zone 2: Values 2, 3, 1 (Range: (1, 3))
- Zone 3: Values 4, 6, 2 (Range: (2, 6))

Querying $A>4$:

Check the min, max of each zone, and skip those do not contain value > 4 . The 2nd zone does not contain value > 4 , and hence no need to access that zone.

QUESTION 2

Suppose we have the following column applied with zone map of size 3.

- 1) Illustrate the information recorded for each zone.
- 2) Give the steps of using zone map to answer queries “ $A>4$ ”
- 3) Give the steps of using zone map to answer queries “ $A>7$ or $A<2$ ”

A
1
3
5
2
3
1
4
6
2

(1, 5)

(1, 3)

(2, 6)

Querying $A>7$ or $A<2$:

Check the min, max of each zone, and skip those do not contain value > 7 or <2 .

The 3rd zone does not contain value > 7 or <2 , and hence no need to access that zone.

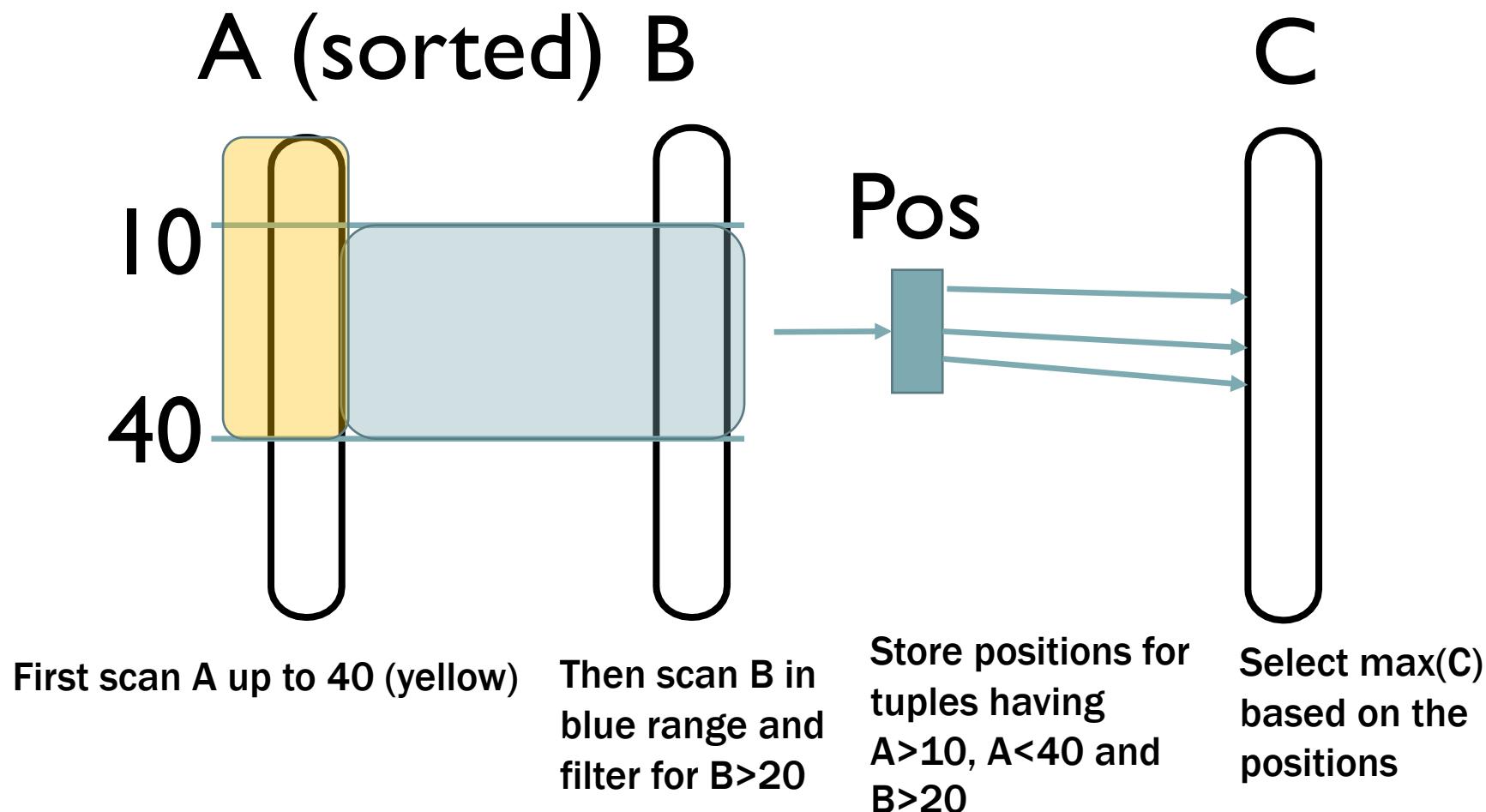
QUESTION 3

In lecture slides, we discuss two ways of scanning a sorted column:
scanning from the start and scanning using binary search.

Give one extreme scenario where the first method is better.

RECAP – SCANNING FROM THE START

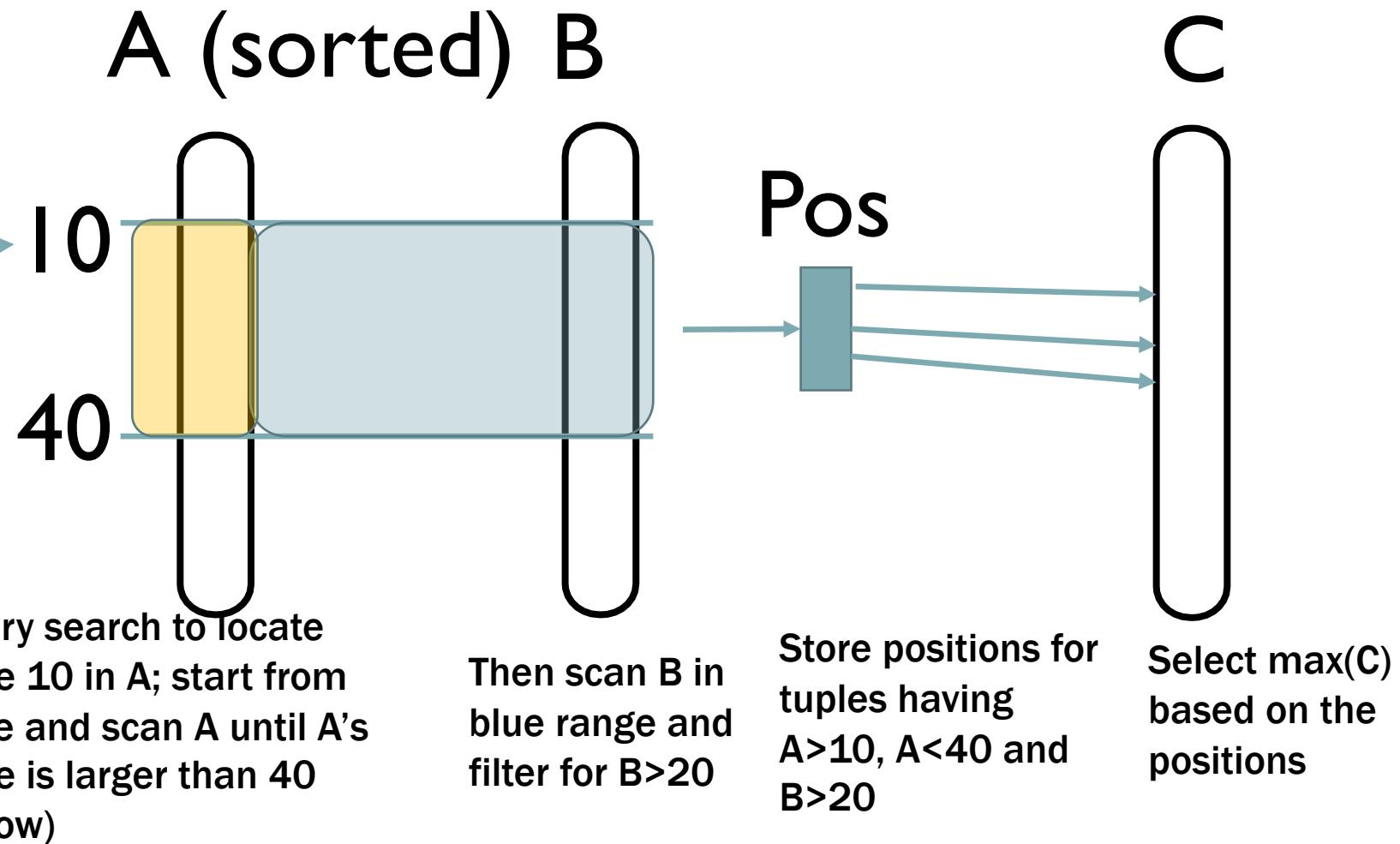
`SELECT max(C) FROM T WHERE A>10 and A<40 and B>20`



RECAP– USE BINARY SEARCH

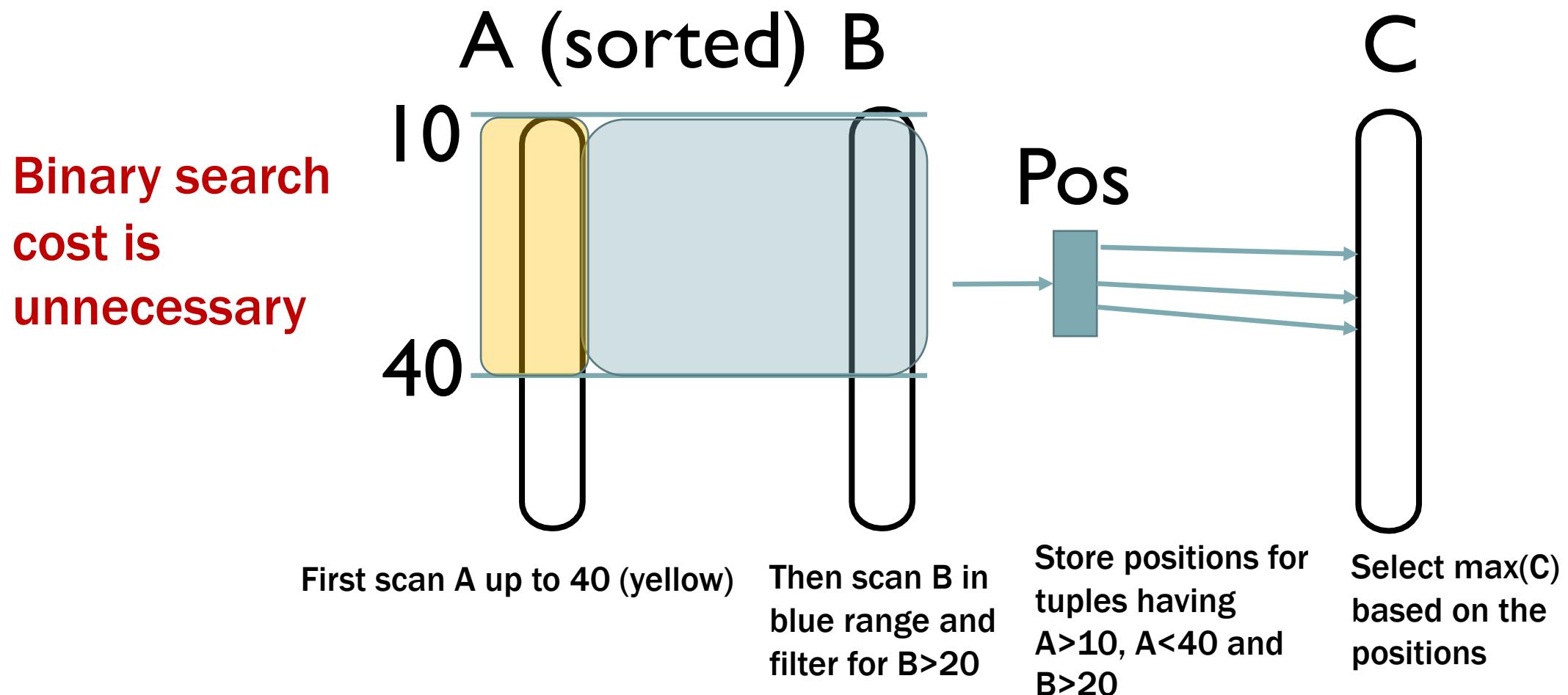
`SELECT max(C) FROM T WHERE A>10 and A<40 and B>20`

Binary search
(usually faster
because there can
be many
values < 10)



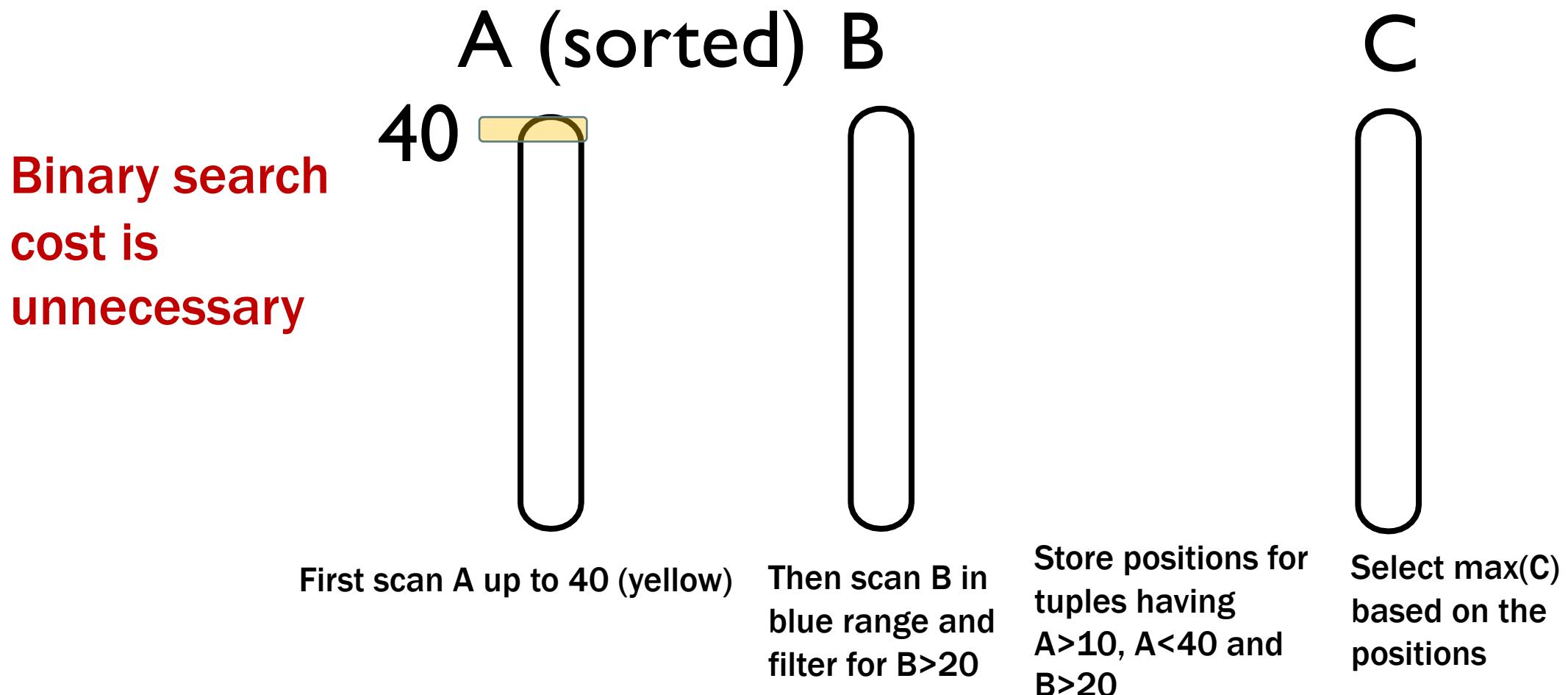
POSSIBLE CASE 1 - FIRST DATA ITEM IN A IS ALREADY IN [10, 40]

SELECT max(C) FROM T WHERE A>10 and A<40 and B>20



POSSIBLE CASE 2 – FIRST DATA ITEM IN A IS ALREADY ≥ 40

`SELECT max(C) FROM T WHERE A>10 and A<40 and B>20`



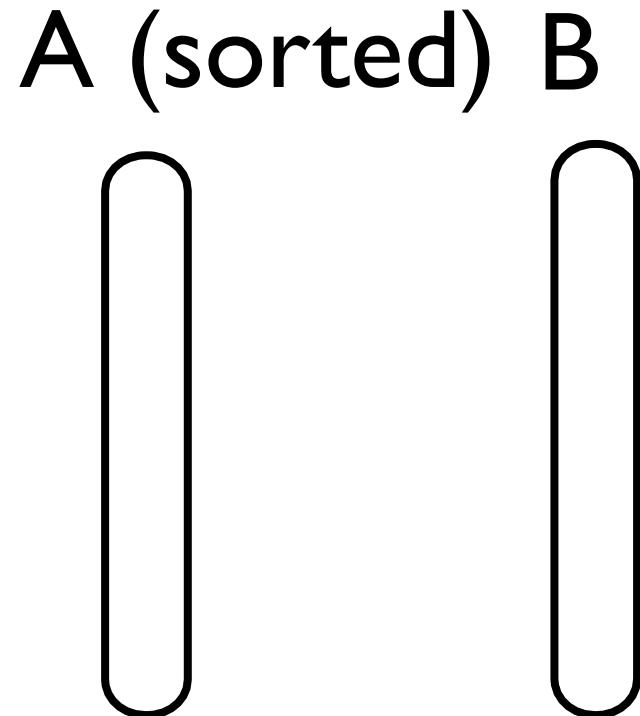
QUESTION 4

Given a table T of two columns A and B. Assume that the table is stored in a column store, and it has two copies: the first copy has column A sorted (while column B follows the order of column A); the second copy has column B sorted (while column A follows the order of column B).

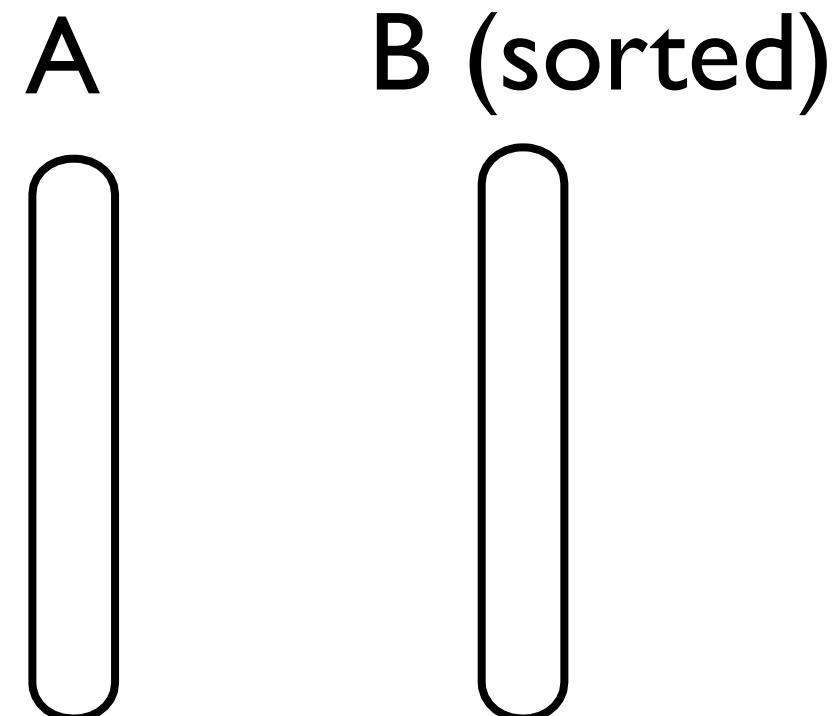
- (1) For query “select max(B) from T where A>3 and A<6”, which copy should be used for lower cost?
- (2) For query “select max(A) from T where B>3 and B<6”, which copy should be used for lower cost?
- (3) We can also use multiple copies simultaneously. Please describe possible issues of using multiple copies.

SOLUTIONS (1) (2)

Queries on sorted columns are faster!



For query “select max(B) from T **where A>3 and A<6**”, which copy should be used for lower cost?



For query “select max(A) from T **where B>3 and B<6**”, which copy should be used for lower cost?

QUESTION(3)-OPEN DISCUSSION

Describe possible issues of using multiple copies.

Think...



QUESTION(3)-OPEN DISCUSSION

Describe possible issues of using multiple copies.

- (1) Large space
- (2) Updates have to be conducted on each copy

BIG DATA MANAGEMENT

CE/CZ4123

OVERVIEW (1ST HALF)

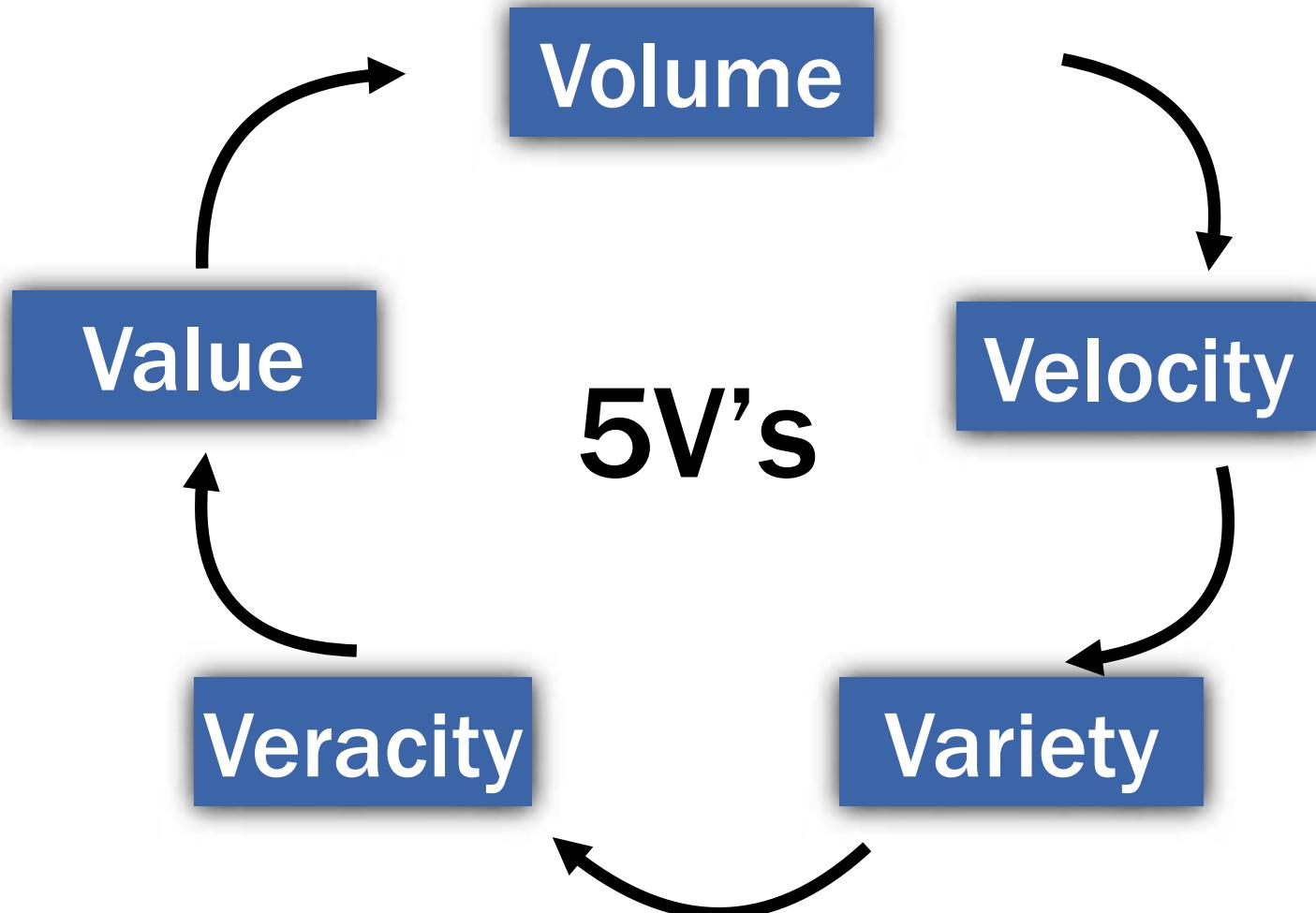
Siqiang Luo

Assistant Professor

THIS LECTURE

- Quickly go through key knowledge we have learnt, including
 - Big Data 5V's
 - Data Models
 - Memory Hierarchy
 - Column Store

BIG DATA 5V'S



Volume: Large amount of data

Velocity: Fast data generation

Variety: Various data types/sources

Veracity: accurate and truthful

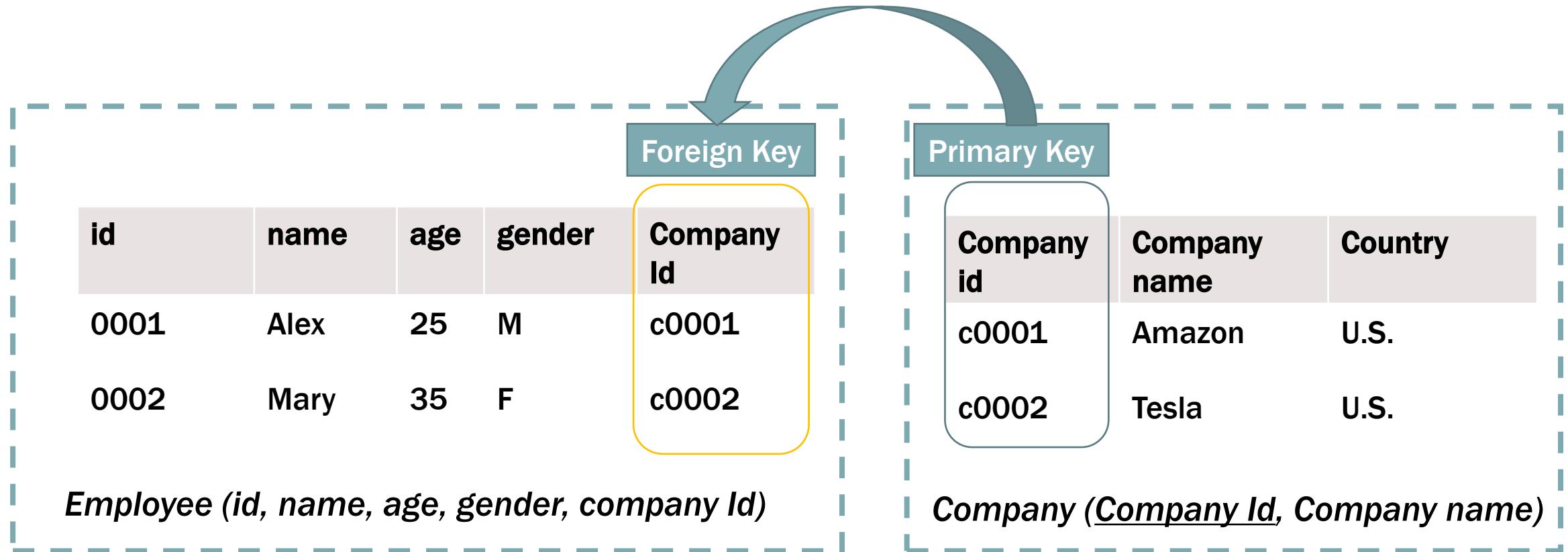
Value: Benefits from analyzing the data

DATA MODELS

- ❑ Relational Data Model
 - ❑ Corresponding to relational database
- ❑ Key-Value Data Model
 - ❑ Corresponding to key-value systems
- ❑ Graph Data Model
 - ❑ Corresponding to graph database

RELATIONAL DATA MODEL

Primary key – Foreign key relationship



A foreign key is a set of one or more columns in a table that refer to the primary key in another table.

KEY-VALUE DATA MODEL

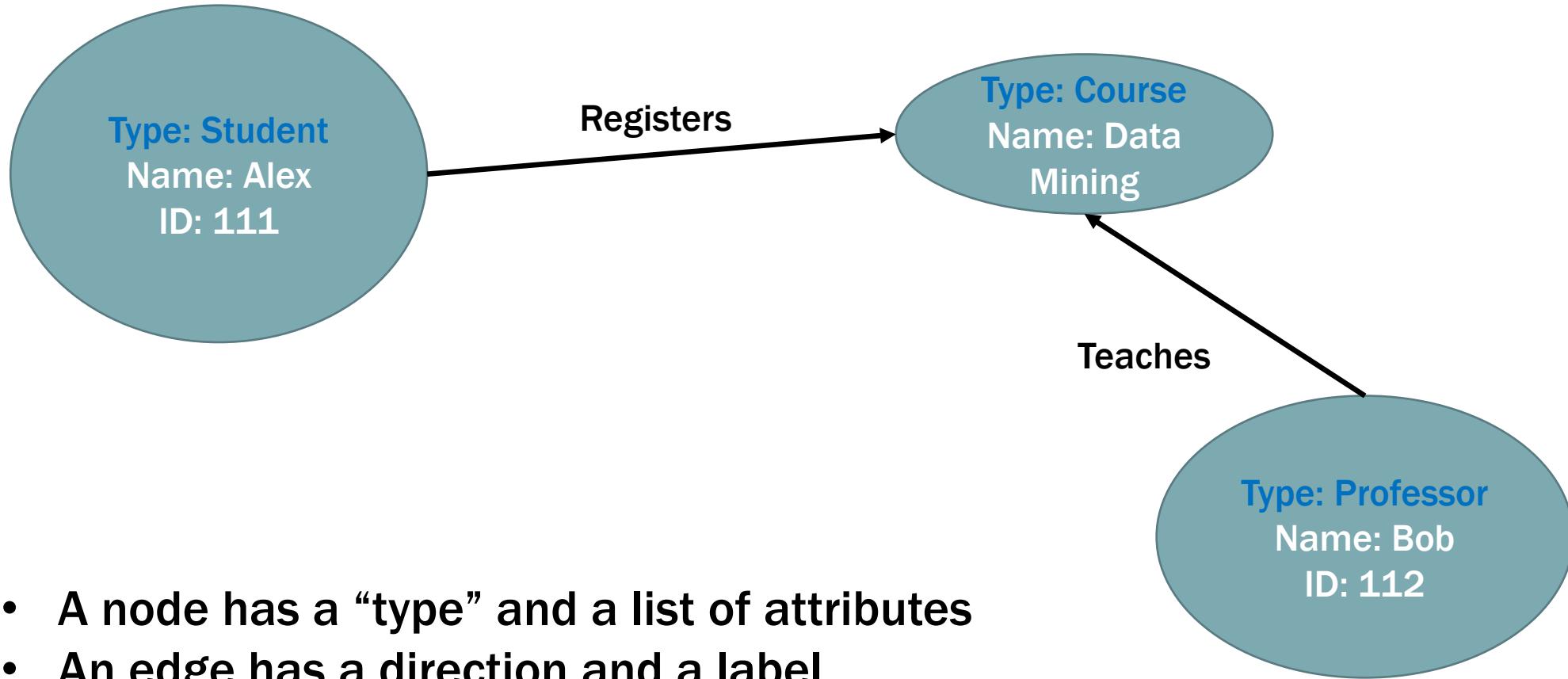
Key-value Data Model is ubiquitous!

For any A that can determine B

Key

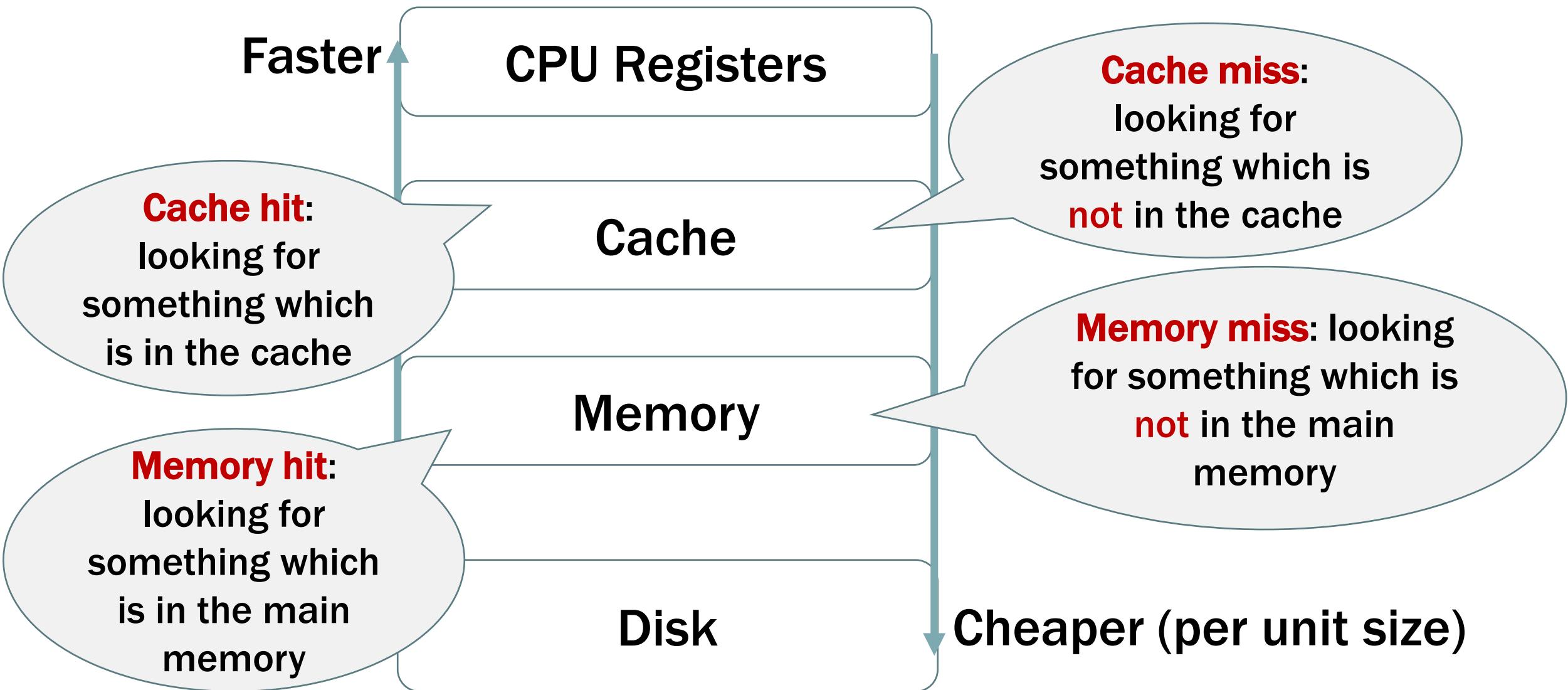
Value

GRAPHS ARE UBIQUITOUS

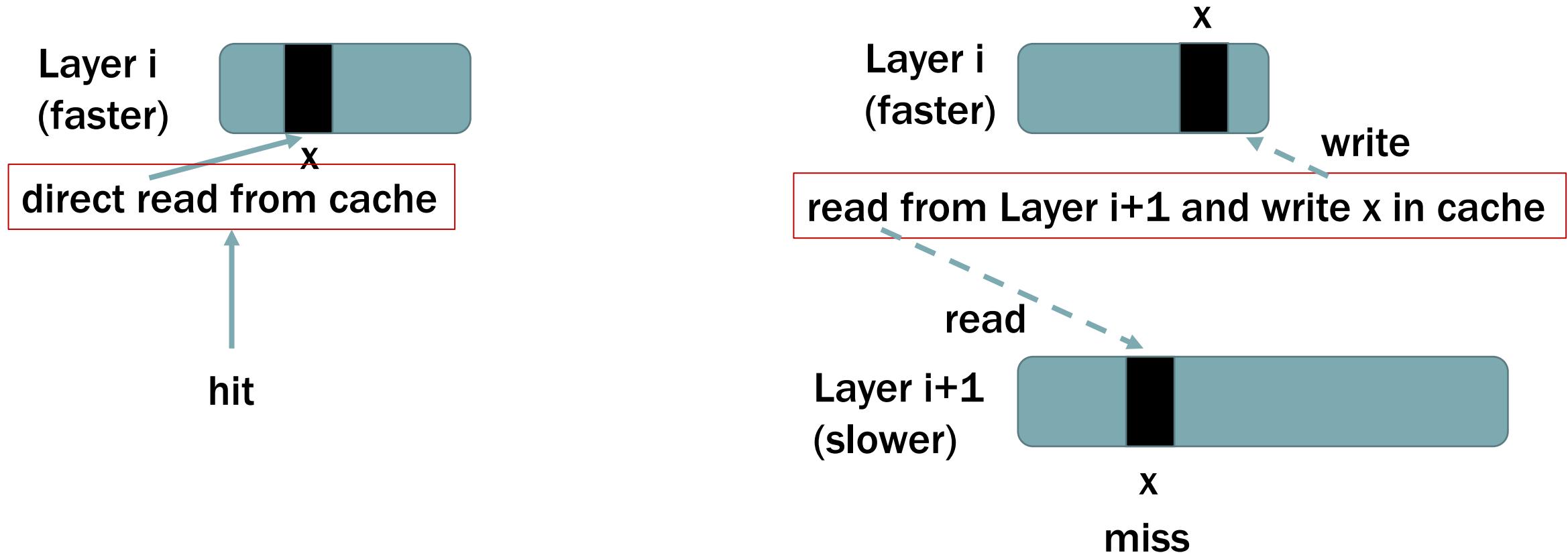


- A node has a “type” and a list of attributes
- An edge has a direction and a label

MEMORY HIERARCHY



DATA ACCESS IN MEMORY HIERARCHY



COST OF A CACHE MISS

- ❑ $\text{cost_access_mem_overall} = \text{cost_access_cache} \times (1 - \text{missrate}) + (\text{cost_access_cache} + \text{cost_access_mem}) \times \text{missrate}$

PAGE-BASED ACCESS EXAMPLE: SCANNING ARRAYS

Query $x < 4$ from the following data

(size=120 bytes)

Memory Layer i

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

PAGE-BASED ACCESS EXAMPLE: SCANNING ARRAYS

Cost: **40 Bytes**

(size=120 bytes)
Memory Layer i

Query $x < 4$ from the following data

Scan
→

5, 10, 7, 4, 12

Qualified results

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

PAGE-BASED ACCESS EXAMPLE: SCANNING ARRAYS

Cost: **80 Bytes**

(size=120 bytes)
Memory Layer i

Query $x < 4$ from the following data

Scan

5, 10, 7, 4, 12

2, 8, 9, 11, 7

2

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

PAGE-BASED ACCESS EXAMPLE: SCANNING ARRAYS

Cost: **120 Bytes**

(size=120 bytes)
Memory Layer i

Query $x < 4$ from the following data
Scan →

7, 11, 3, 9, 8

2, 8, 9, 11, 7

2, 3

Memory Layer $i+1$

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes \times 5 = 40 bytes

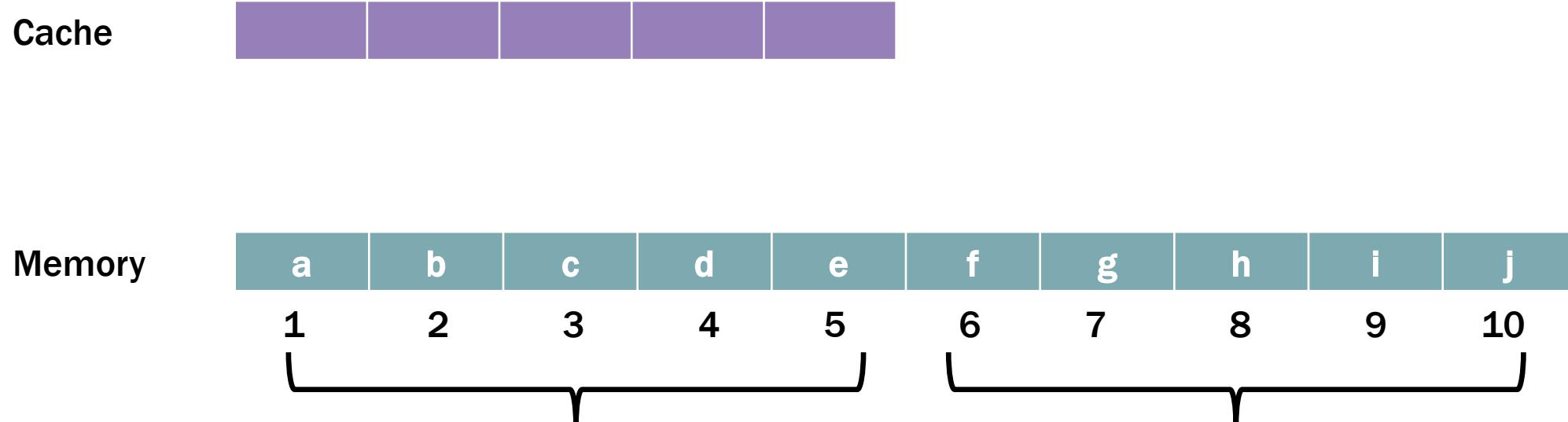
CACHE CONSCIOUS DESIGNS

- We show some design principles to make use of memory hierarchy for processing big data. We will discuss following two cases.
 - Array access patterns
 - Spatial locality designs
 - Temporal locality designs
 - Big data sorting

ACCESS PATTERN 1

We have a size-10 array stored in main memory,
cache size = 5, transfer size (cache line)= 5

Access pattern: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10



ACCESS PATTERN 1

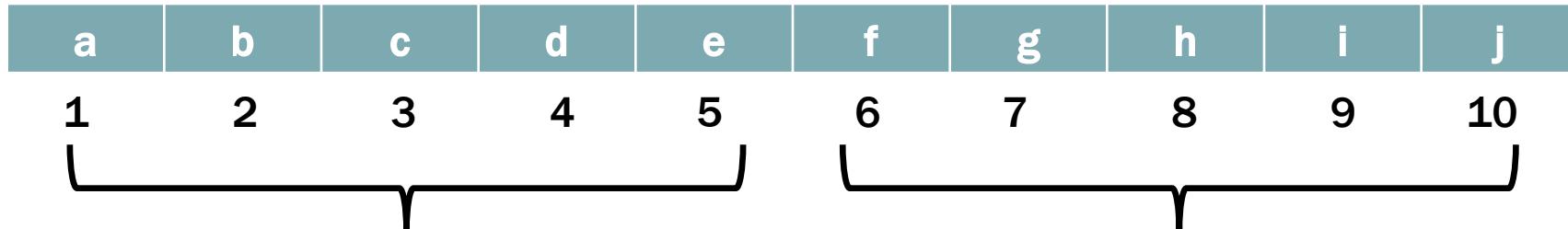
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: **1**
Cache Hit: 0

Memory



ACCESS PATTERN 1

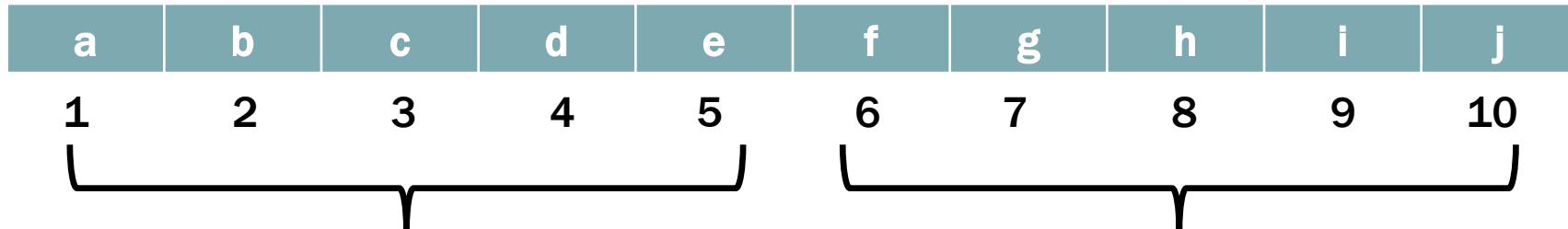
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: 1
Cache Hit: **1**

Memory



ACCESS PATTERN 1

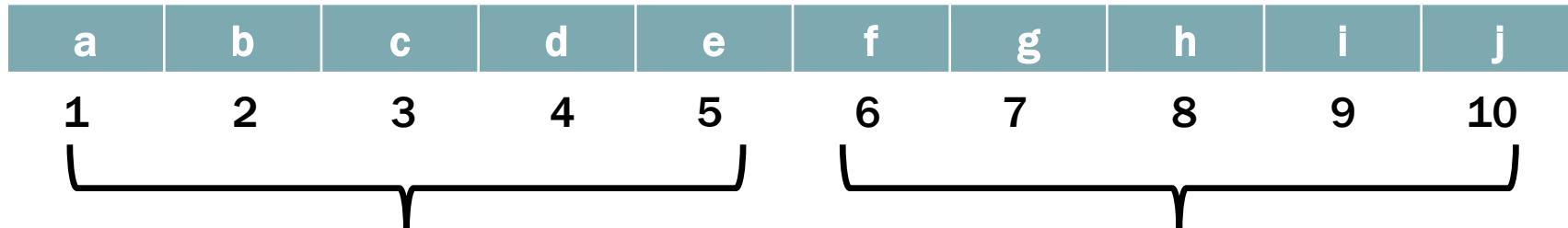
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: 1
Cache Hit: 2

Memory



ACCESS PATTERN 1

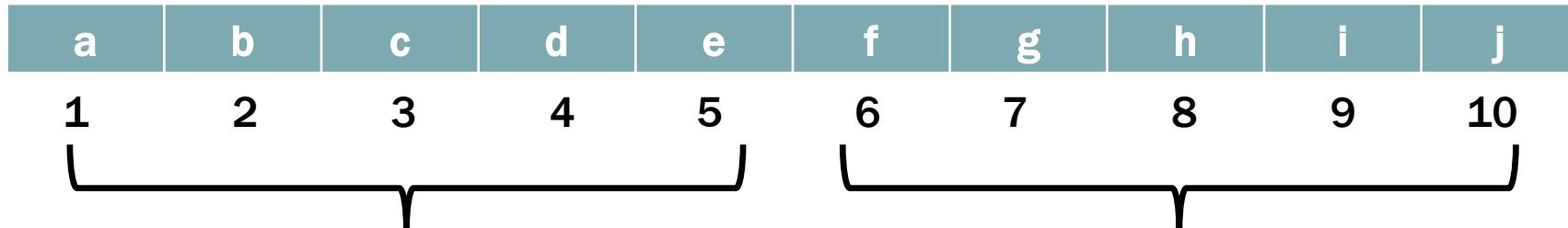
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: 1
Cache Hit: 3

Memory



ACCESS PATTERN 1

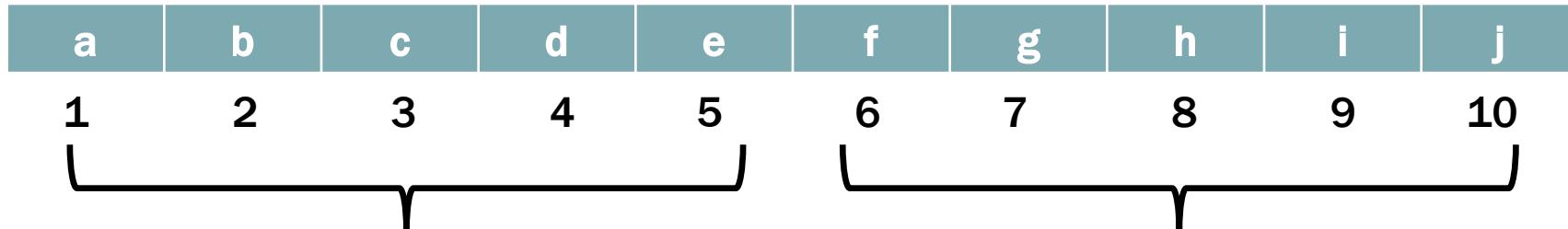
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



Cache Miss: 1
Cache Hit: 4

Memory



ACCESS PATTERN 1

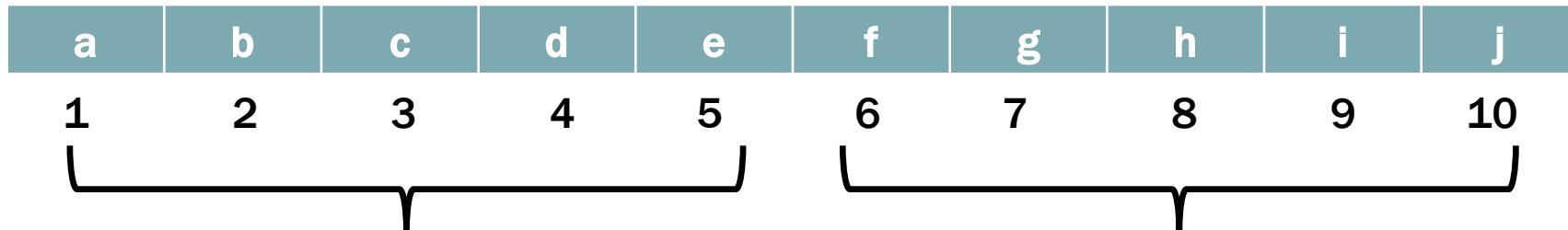
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



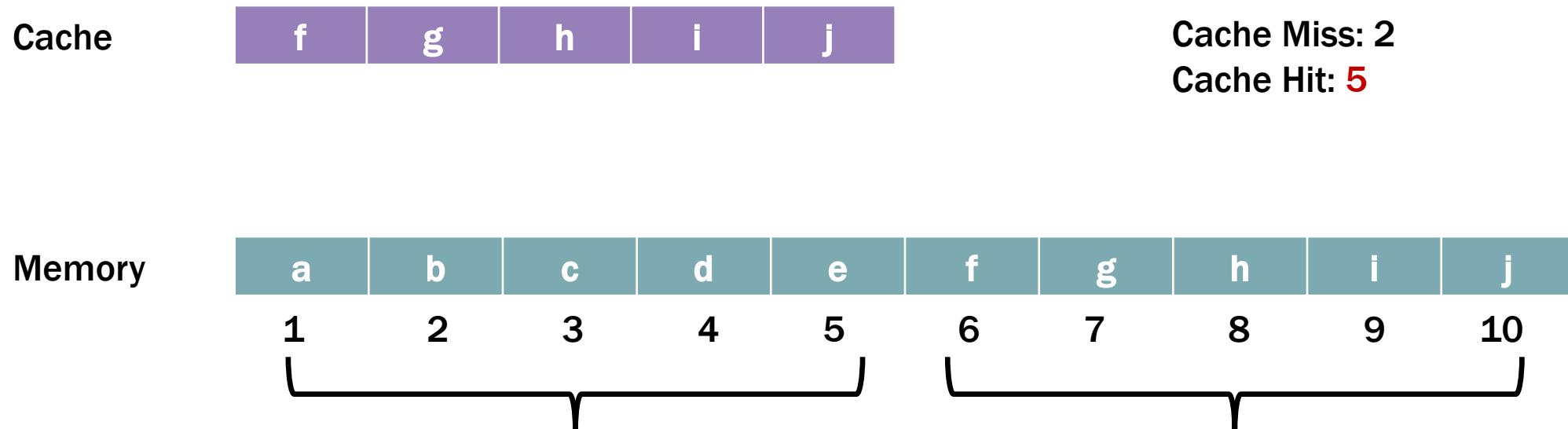
Cache Miss: **2**
Cache Hit: 4

Memory



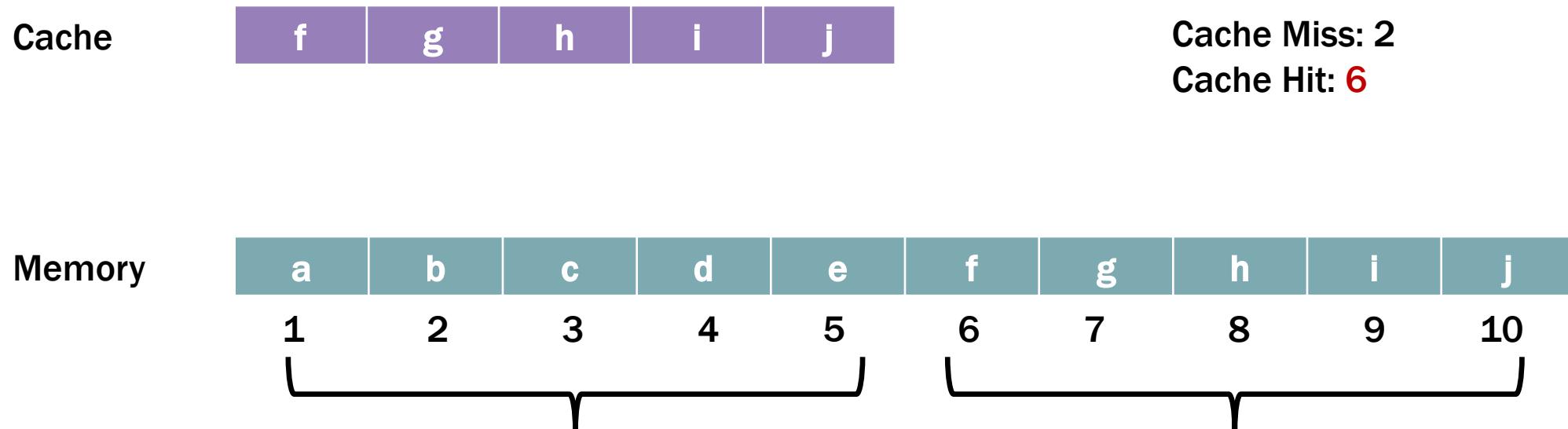
ACCESS PATTERN 1

Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**



ACCESS PATTERN 1

Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**



ACCESS PATTERN 1

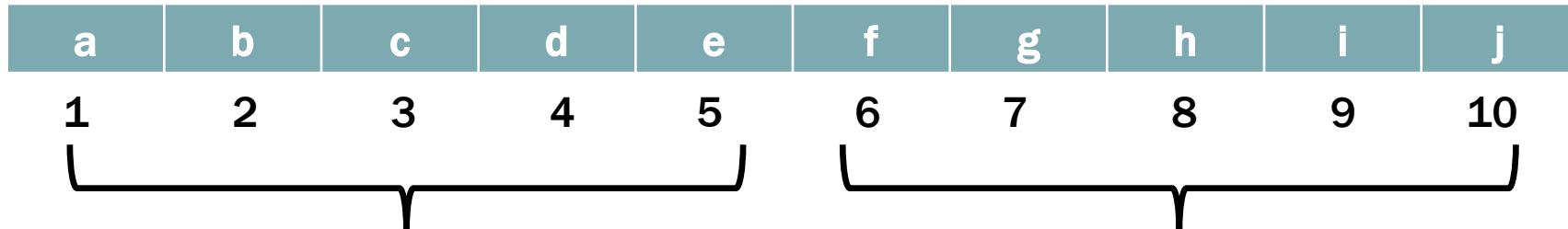
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**

Cache



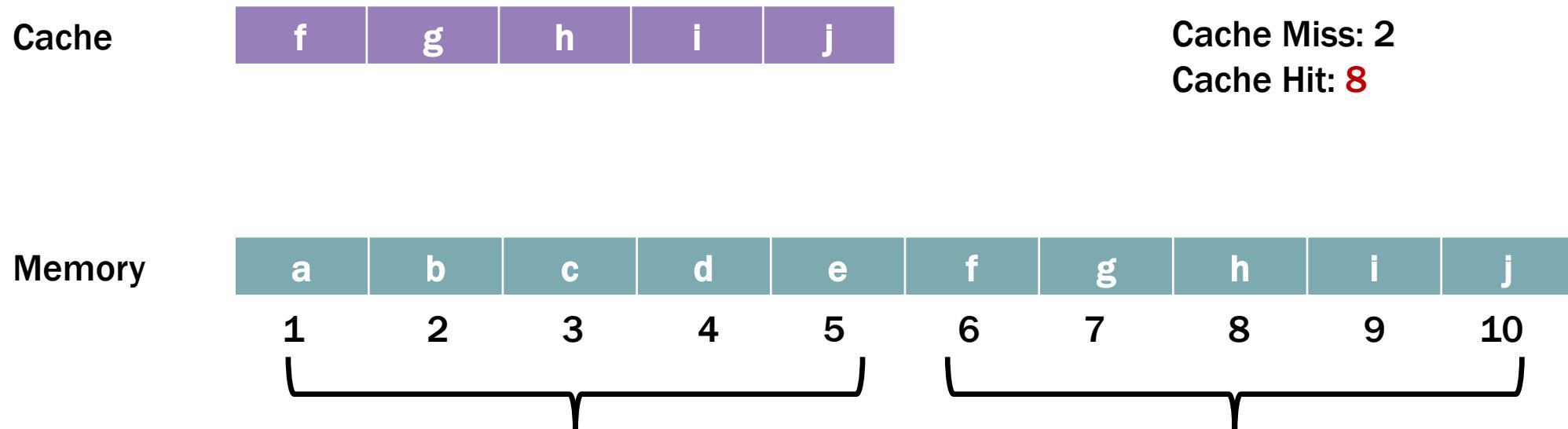
Cache Miss: 2
Cache Hit: **7**

Memory



ACCESS PATTERN 1

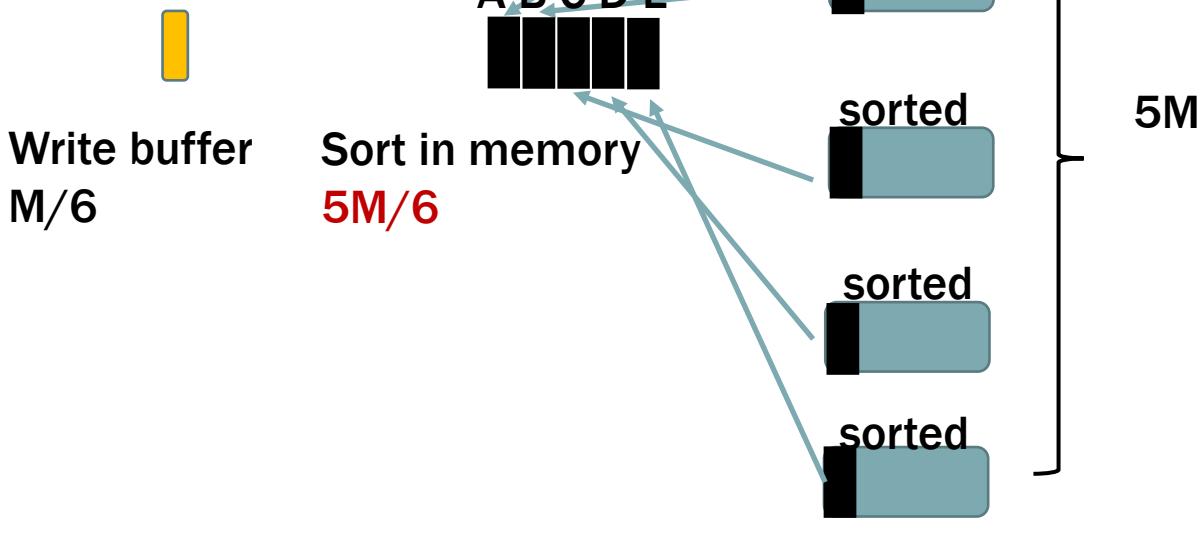
Access pattern: **1, 2, 3, 4, 5, 6, 7, 8, 9, 10**



SORTING

■ or ■ size $M/6$

5M/6 in total for in-memory sorting, remaining **M/6** for writing the results



Details of merge sort

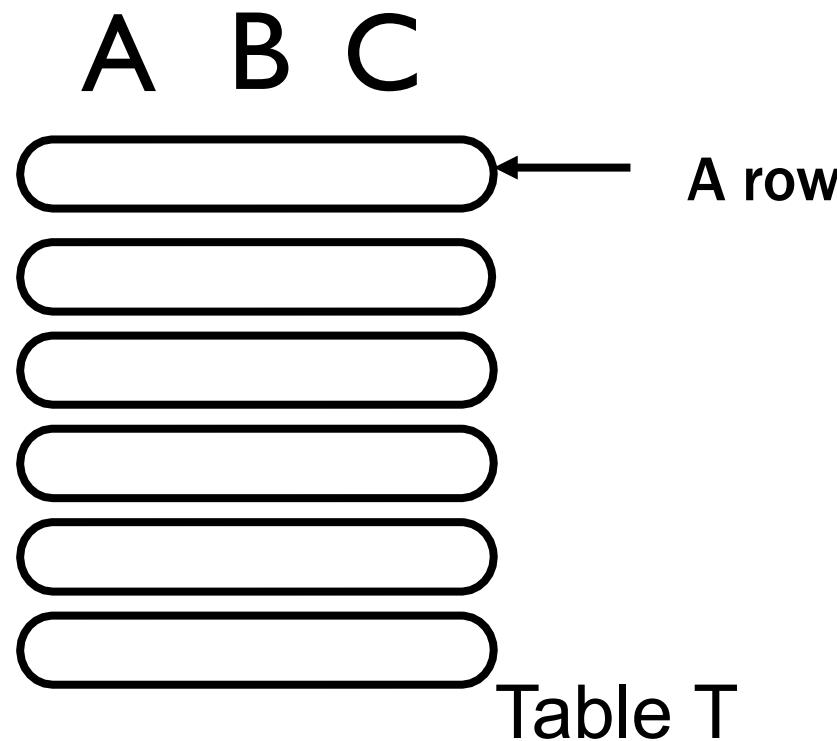
1. 5 iterators scanning each of {A,B,C,D,E} from left to right;
2. Put the smallest-value (pointed by the 5 iterators) in the write-buffer; forward the corresponding iterator;

A	2, 13
B	7, 10
C	6, 11
D	5, 12
E	1, 14

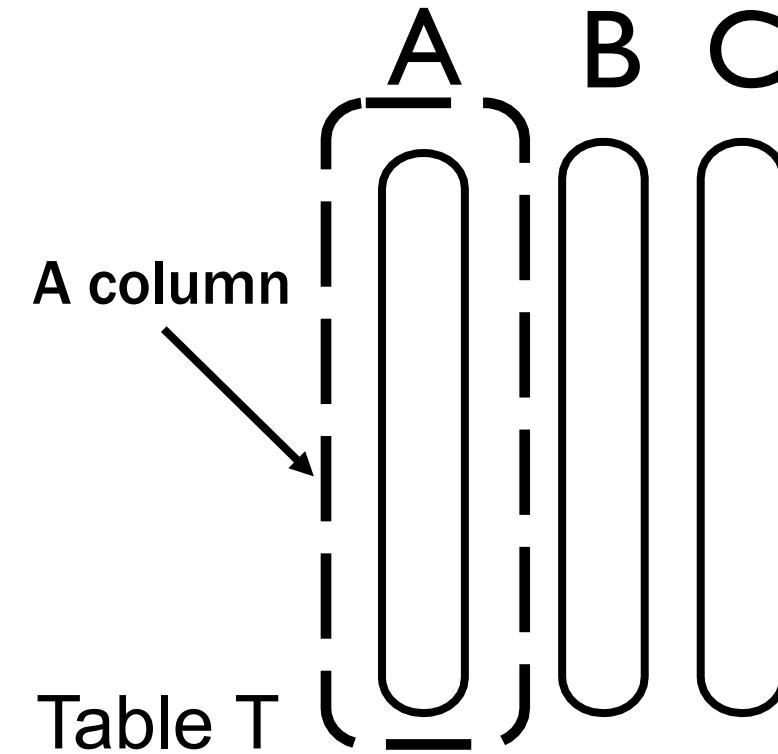
MAIN DESIGN OF COLUMN STORES

- Data in column stores are column-oriented
 - Also called column-oriented database, or columnar database.

row-store (traditional RDB)

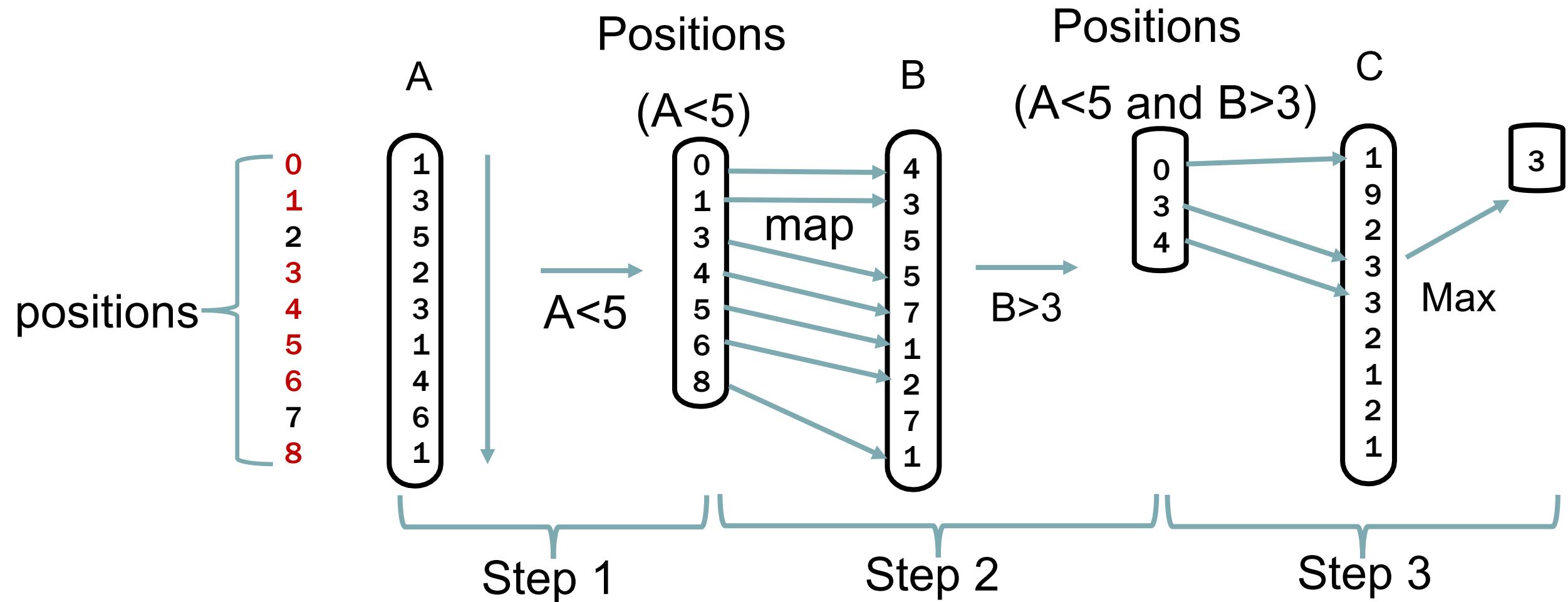


column-store



QUERYING WITH COLUMN STORE (FLOW CHART)

- The whole table is stored in disk
- Have disk cache (memory buffer) whose size is a multiple of page size



COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Without using an index, handling the query in the row store needs to read through the whole table. Namely, it costs Z^*w^*3/P page accesses.

Cost for column store (number of page access):

$$\begin{aligned} & Zw/P + 2\text{result}(A)^*4/P + \text{result}(A) + 2\text{result}(AB)^*4/P + \text{result}(AB) \\ & = Zw/P + \text{result}(A)^*(8/P+1) + \text{result}(AB)^*(8/P+1) \\ & \approx Zw/P + \text{result}(A) + \text{result}(AB) \end{aligned}$$

Cost for row store (number of page access):

$$3Zw/P$$

Note: the values of $\text{result}(A)$ and $\text{result}(AB)$ are typically much smaller than Z . So we can conclude in this case column store is faster.

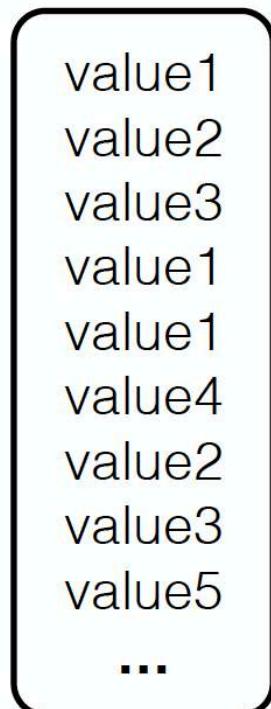
OPTIMIZATIONS

- Compression
- Shared Scan
- Zone Map
- Sorting
- Indexing

COMPRESSION

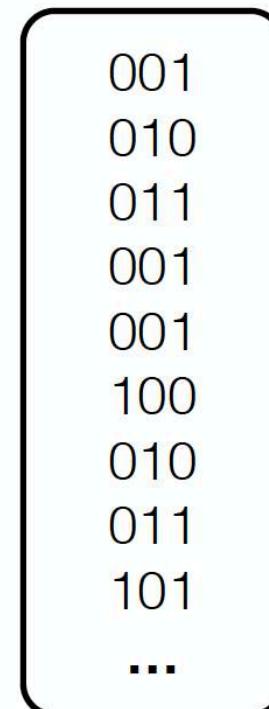
Original data

8 bytes
width

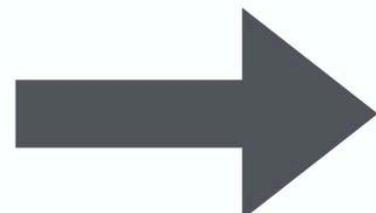
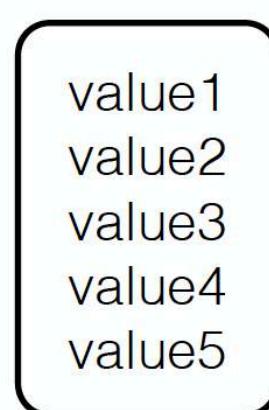


Compressed Dictionary

3 bits
width



8 bytes
width



How many bits?



SHARED SCANS

loop fusion

```
for(i=0;i<n;i++)  
    min = a[i]<min ? a[i] : min  
  
for(i=0;i<n;i++)  
    max = a[i]>max ? a[i] : max
```

Two passes of data

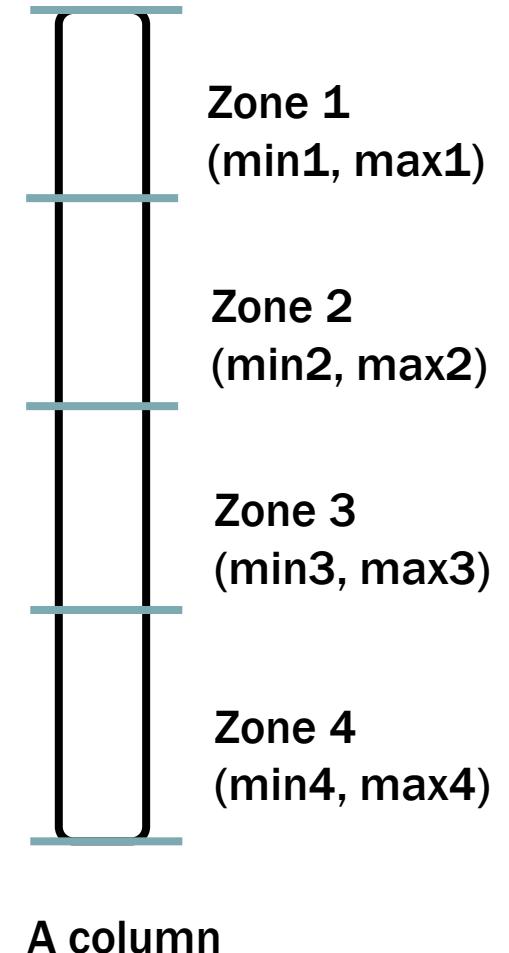
```
for(i=0;i<n;i++)  
    min = a[i]<min ? a[i] : min  
    max = a[i]>max ? a[i] : max
```

One pass of data



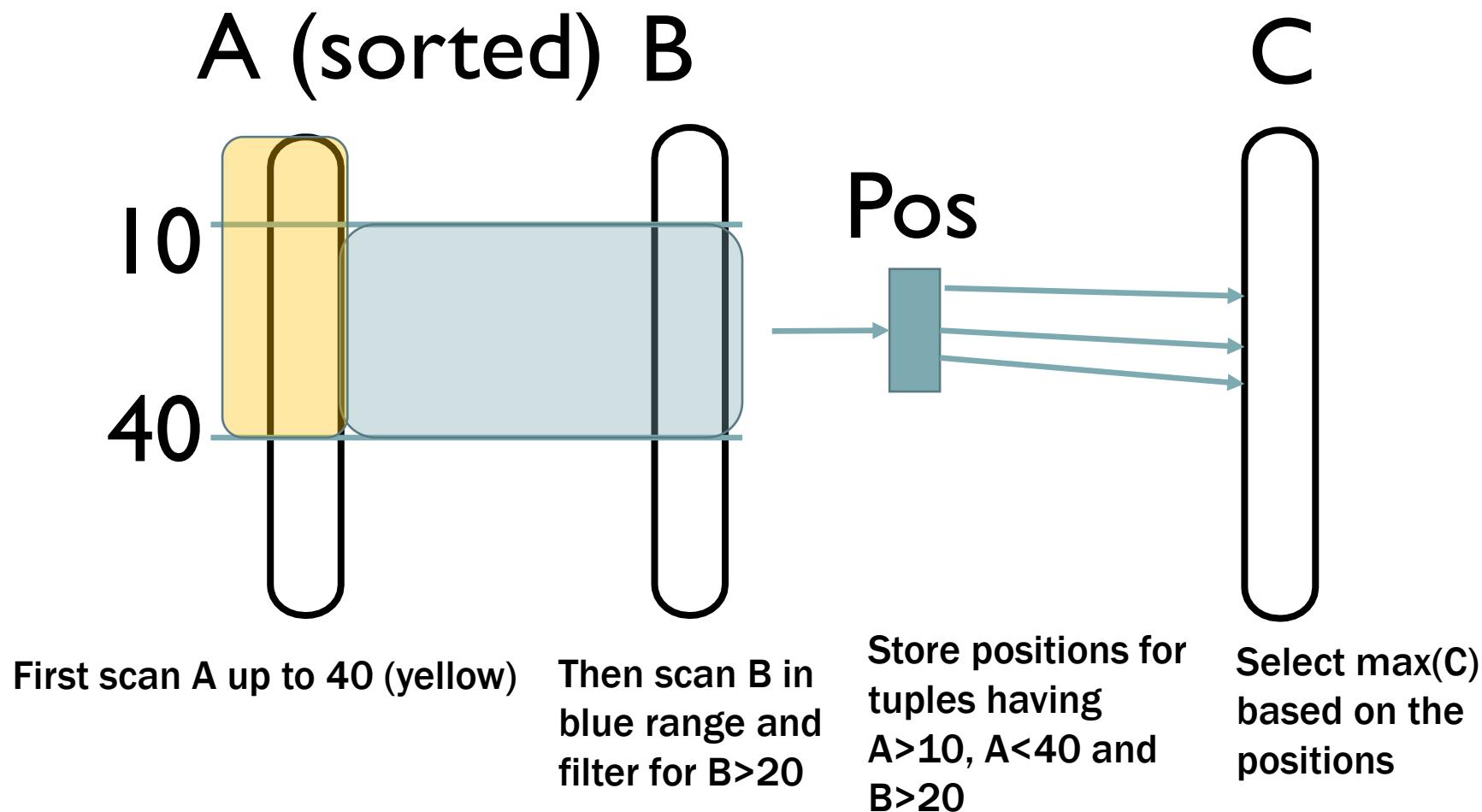
ZONE MAP

- ❑ A common way to help column scan is the zone map.
- ❑ It separates a column into “zones”, each is computed with max and min
- ❑ In filtering, some zones can be skipped.



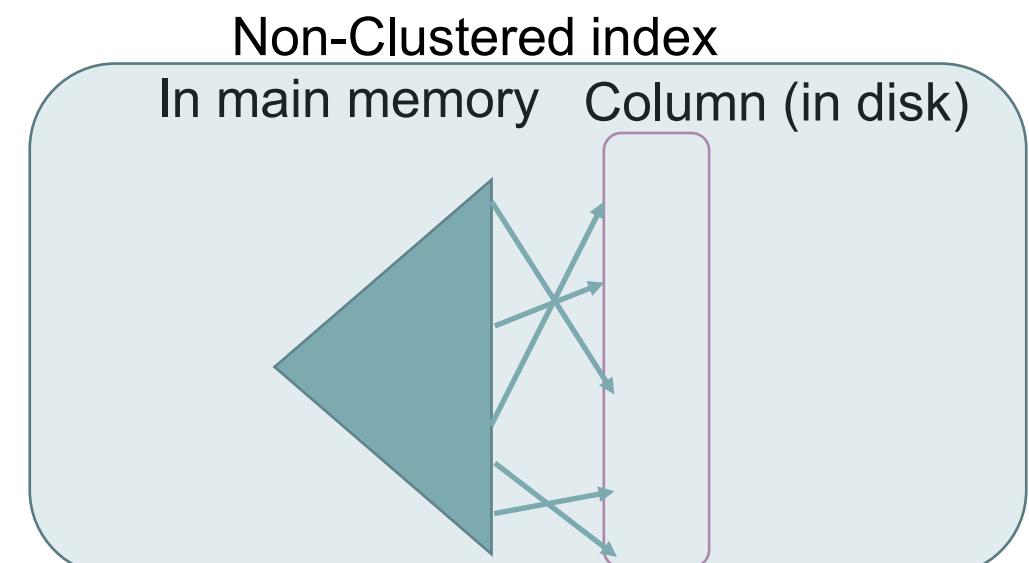
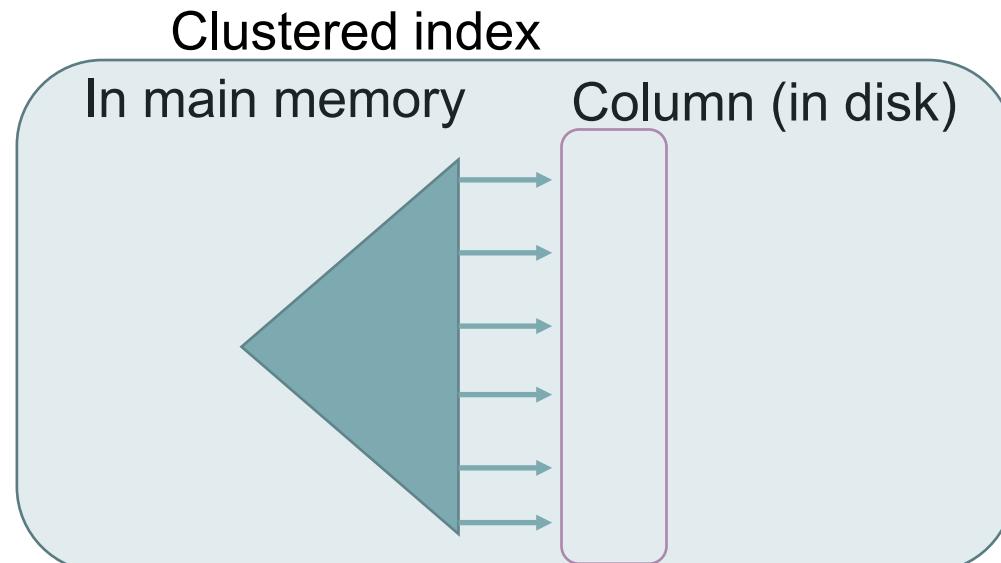
ENHANCED WITH SORTING

SELECT max(C) FROM T WHERE A>10 and A<40 and B>20



ENHANCED WITH INDEX

- Index access is much faster than data access, because index is stored in Main Memory, while data is stored in disk.
- So it is beneficial to have complicated index access to locate the exact pages where stores the data.



**The End of First-Half.
Thank you!**

We finish lectures for Column Store!



Next lecture:

**Distributed Systems and
MapReduce**

BIG DATA MANAGEMENT

CE/CZ4123

DISTRIBUTED SYSTEMS AND MAP-REDUCE

Siqiang Luo

Assistant Professor

LEARNING OUTCOMES

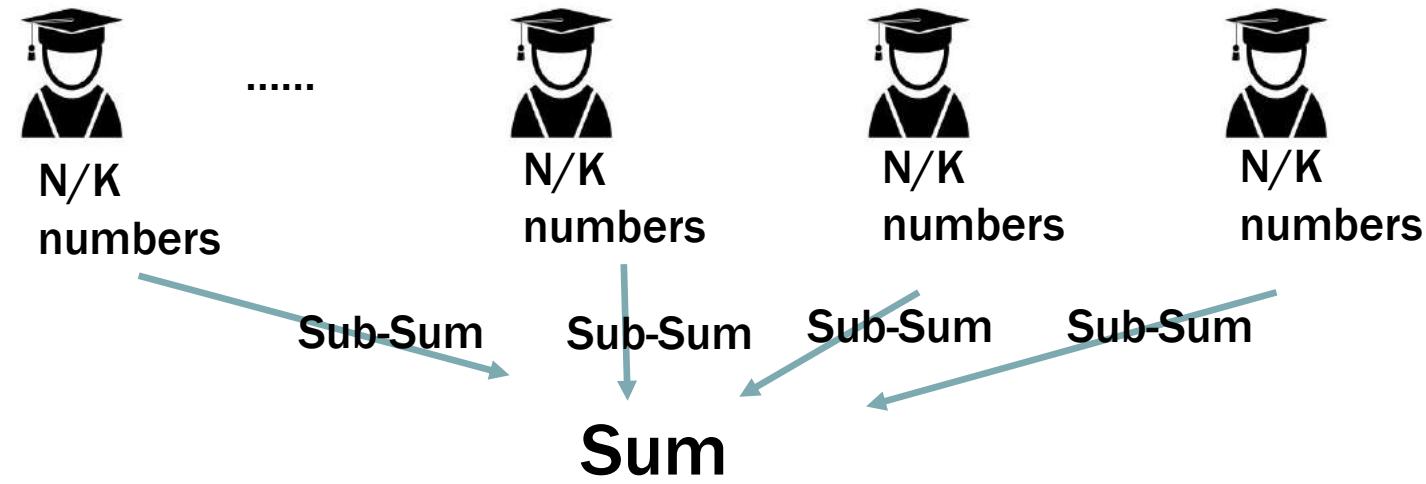
- Understand the concept of distributed computation
- Understand the MapReduce computation model
- Can design proper algorithms based on MapReduce computation model
- Have a basic concept of Hadoop system

STARTING QUESTION

Give N integers to a team of K students to compute the sum. The numbers are given to the team leader. If you are the leader, how would you accomplish the computation task as soon as possible?

STARTING QUESTION

Strategy 1: Divide N numbers to K students to compute, and aggregate the results



Strategy 2: Ask one student to compute

Is there a better strategy?

Team work

Multiple students

V.S.

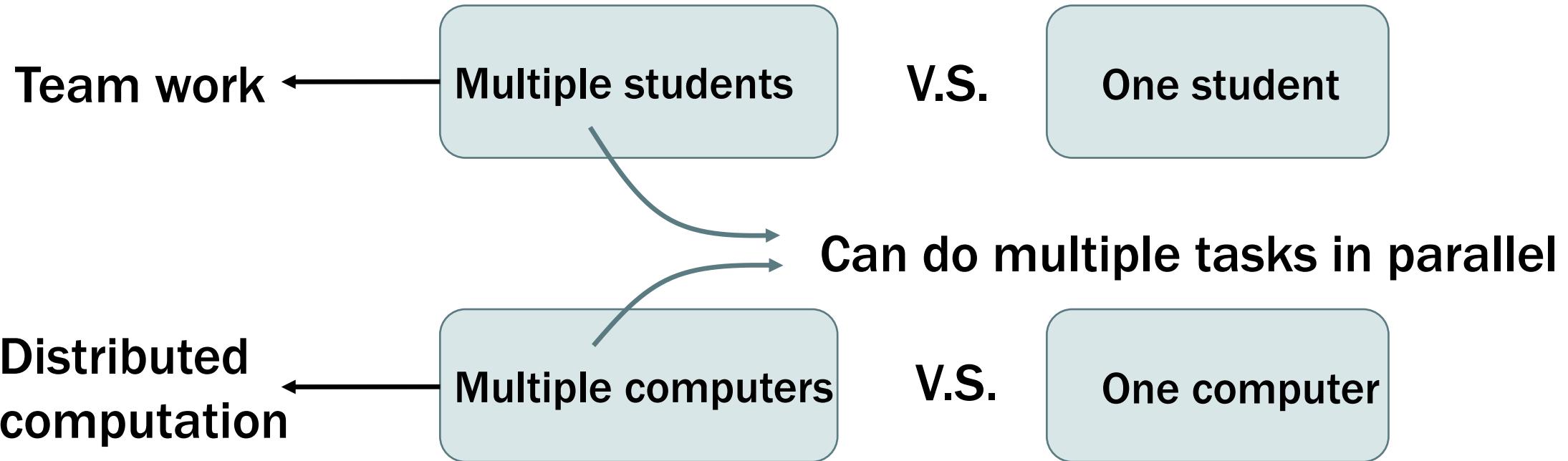
One student

**Distributed
computation**

Multiple computers

V.S.

One computer



If the task is to sum up 10000 integers,
more people will be better.

Distributed computation is a very important
approach in big data applications!

Basics of Distributed Computation

**Distributed computation is often discussed
in the context of big data**

Not small data

DISTRIBUTED SYSTEMS

- ❑ Suppose an engineer from Amazon needs to process huge data of daily purchases.
- ❑ Just use a single commodity computer?



DISTRIBUTED SYSTEMS

- Suppose an engineer from Amazon needs to process huge data of daily purchases.
- Just use a single commodity computer?
 - Lack of storage
 - Lack of computation power

DISTRIBUTED SYSTEMS

- Suppose an engineer from Amazon needs to process huge data of daily purchases.
- Just use a single commodity computer?
 - Lack of storage
 - Lack of computation power

Use multiple machines ~ Distributed system

WHAT IS DISTRIBUTED SYSTEM

A *distributed system* is a system whose components are located on different **networked computers**, which communicate and coordinate their actions by passing messages to one another from any system. The components interact with one another in order to achieve a common goal.

DISTRIBUTED SYSTEMS FOR BIG DATA: CHALLENGES

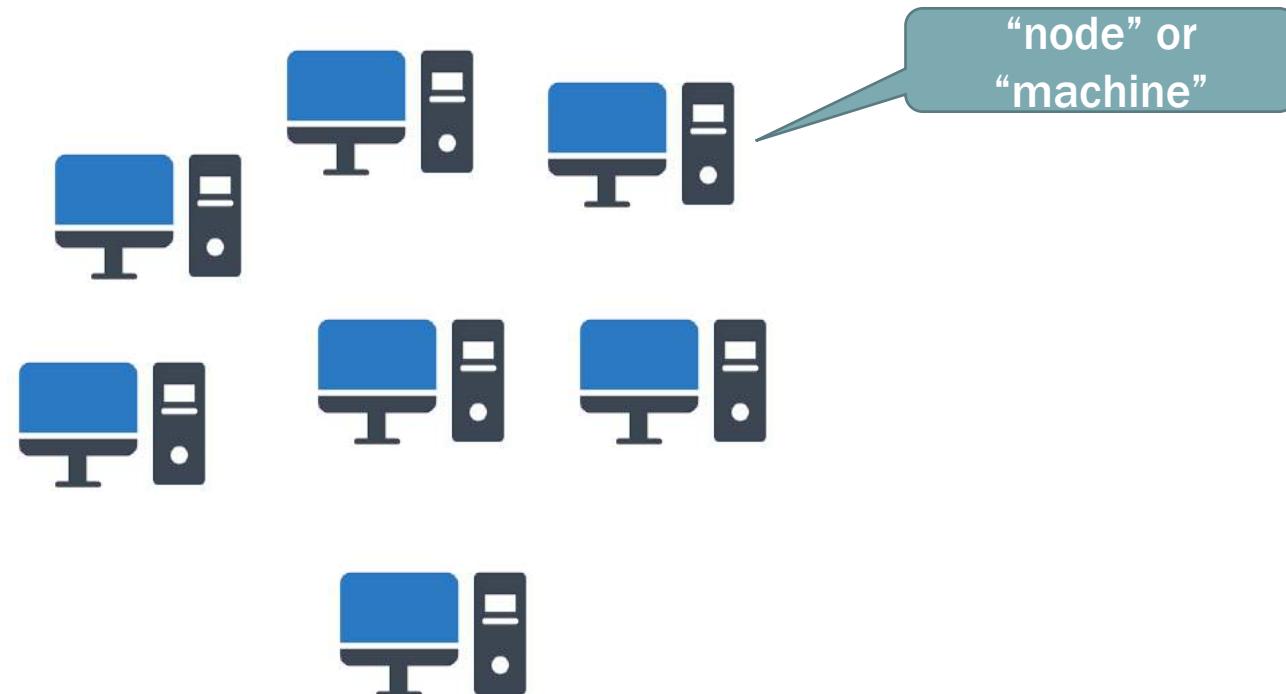
- How to organize the machines?
- How to store data across machines?
- How to compute using multiple machines?

DISTRIBUTED SYSTEMS FOR BIG DATA: CHALLENGES

- How to organize the machines?
- How to store data across machines?
- How to compute using multiple machines?

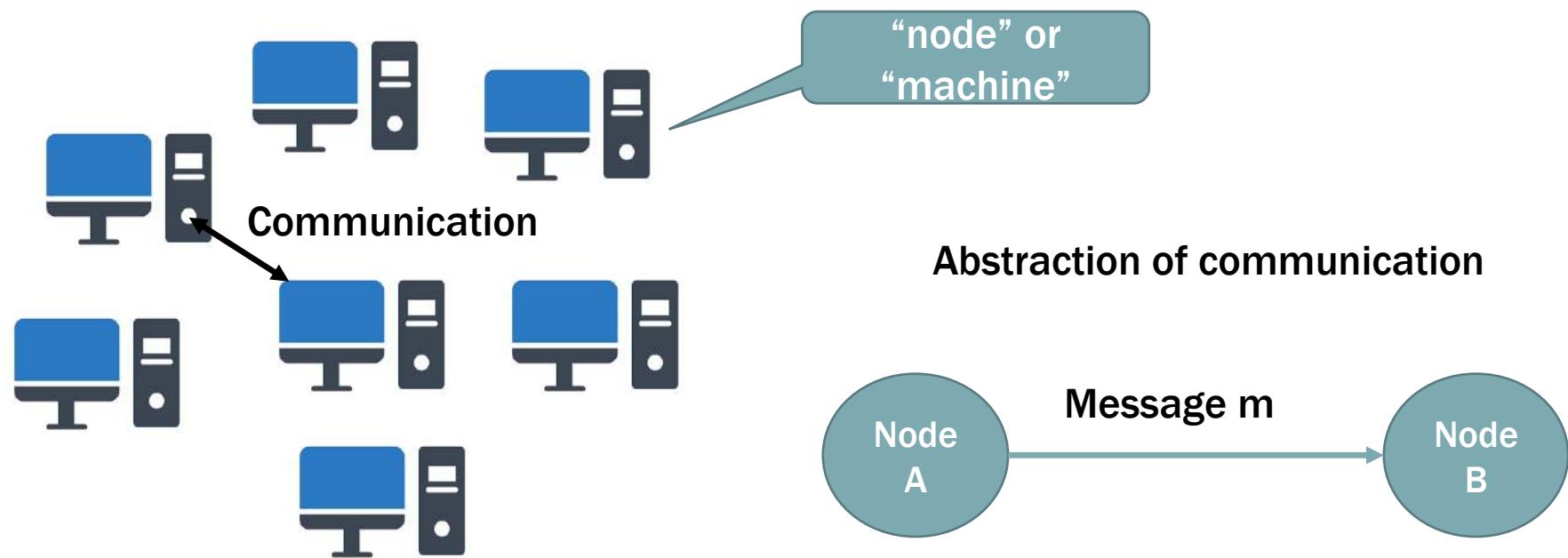
DISTRIBUTED SYSTEMS

- The machines are connected via high-bandwidth networks.
- Two common models:
 - Fully distributed: Any two machines can access each other
 - Master-Slaves: There exists one coordinator



FULLY DISTRIBUTED MODELS

- ❑ Each machine has an IP address
- ❑ A knows machine *B*'s IP address: A can send messages to *B*
- ❑ Two machines can communicate with each other via IP address
 - ❑ i.e., sending messages between machines



COMMUNICATION IS NOT FOR FREE

Latency: time spent on sending/receiving messages

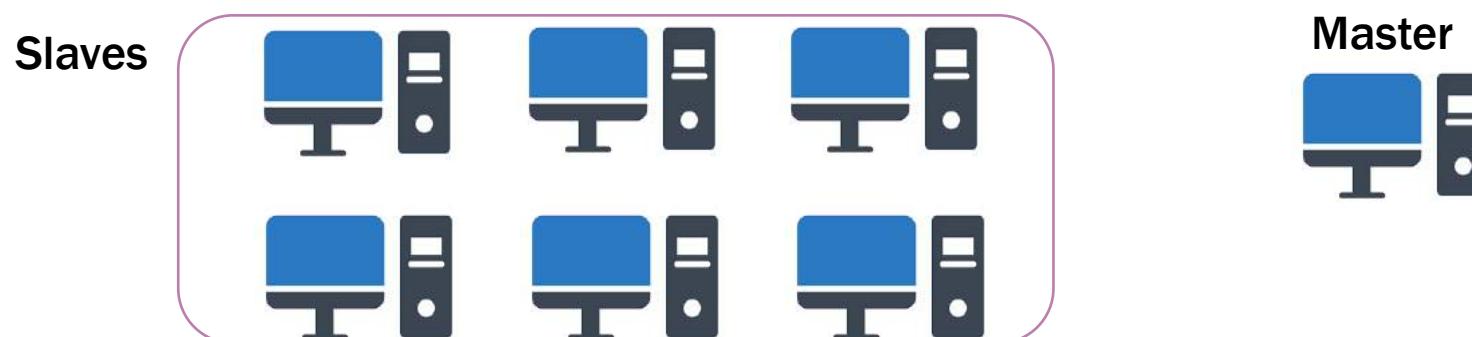
- Same datacenter: around 1ms
- One continent to another: around 100ms

Bandwidth: data volume per unit time

- 3G cellular data: around 1Mbit/s
- 4G cellular data: around 100Mbit/s

MASTER-SLAVE MODEL

- Each machine has an IP address
- There is a machine called **master**, and the other machines are called **slaves**.
- Master is the coordinator, being responsible to
 - distribute tasks to the slaves, and
 - receive the results from the slaves

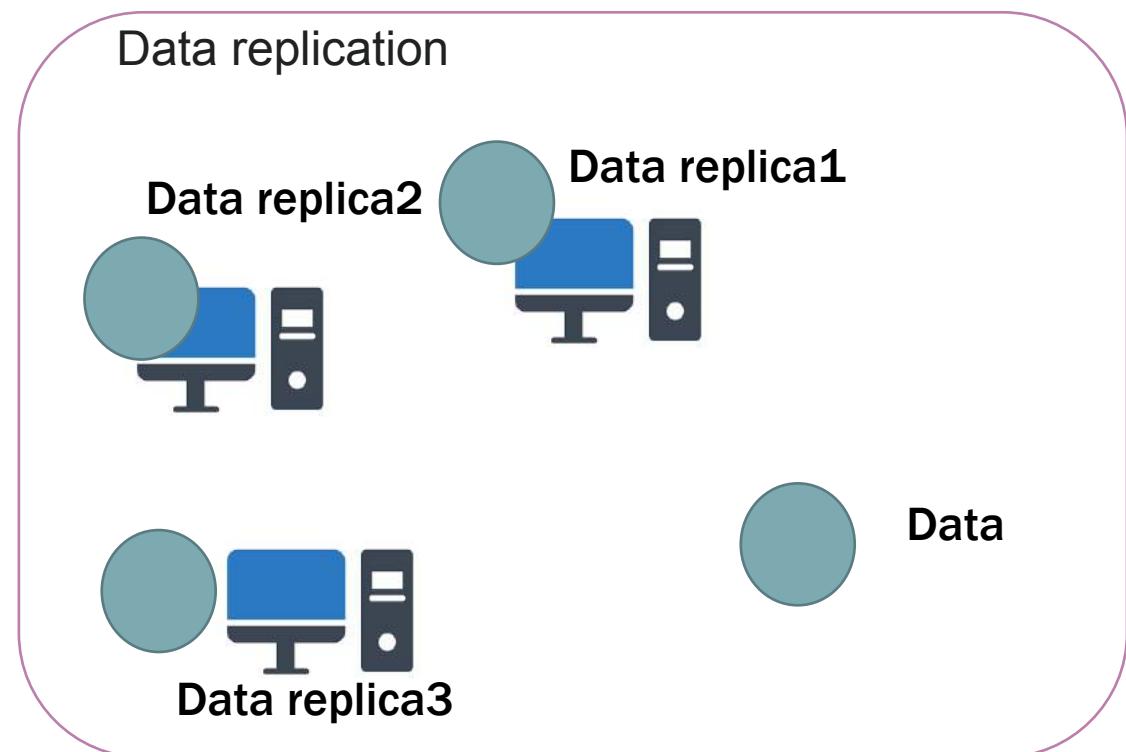
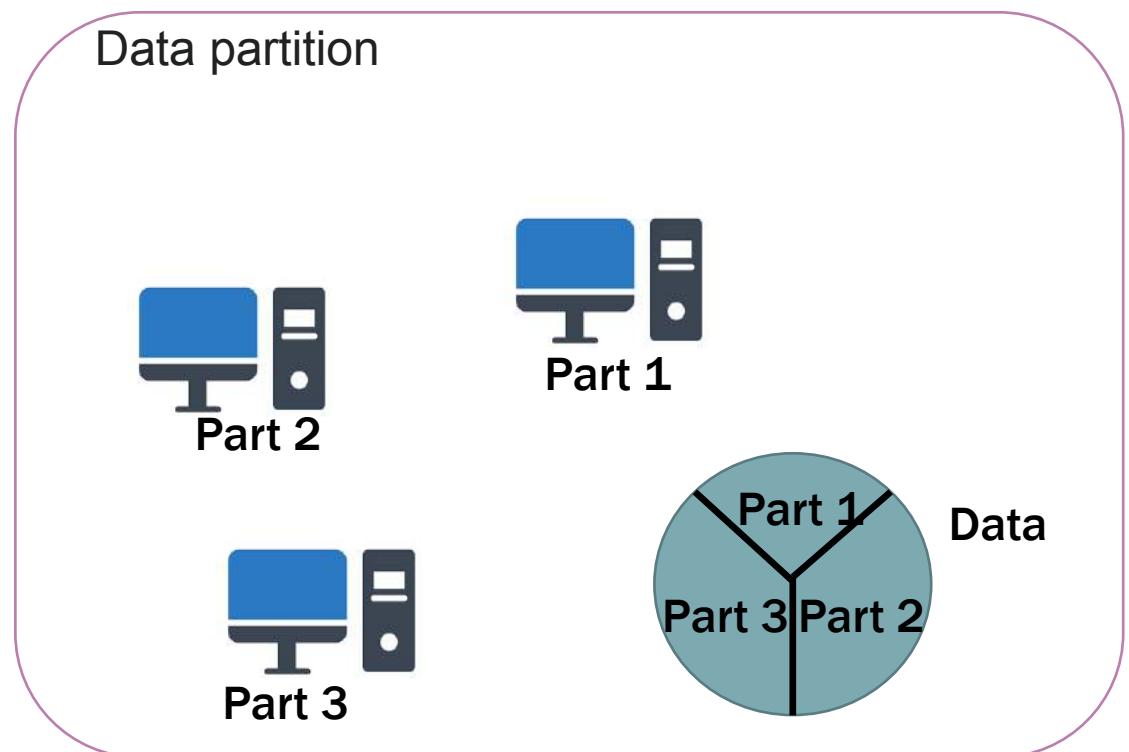


DISTRIBUTED SYSTEMS FOR BIG DATA: CHALLENGES

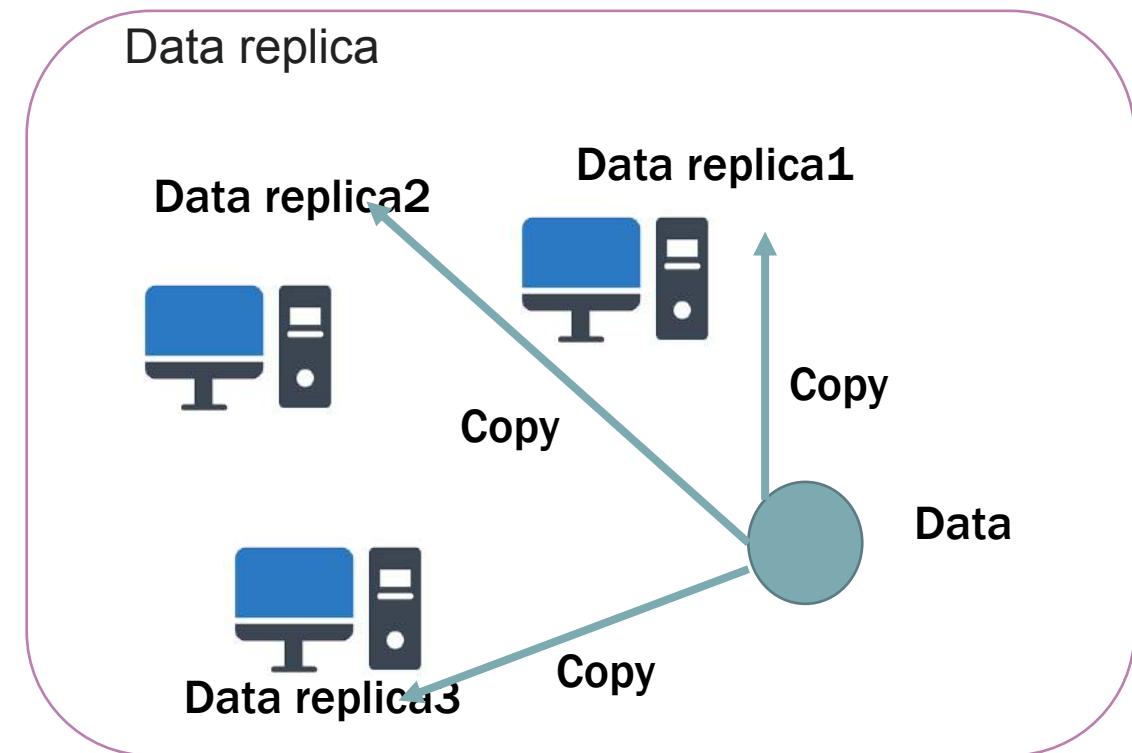
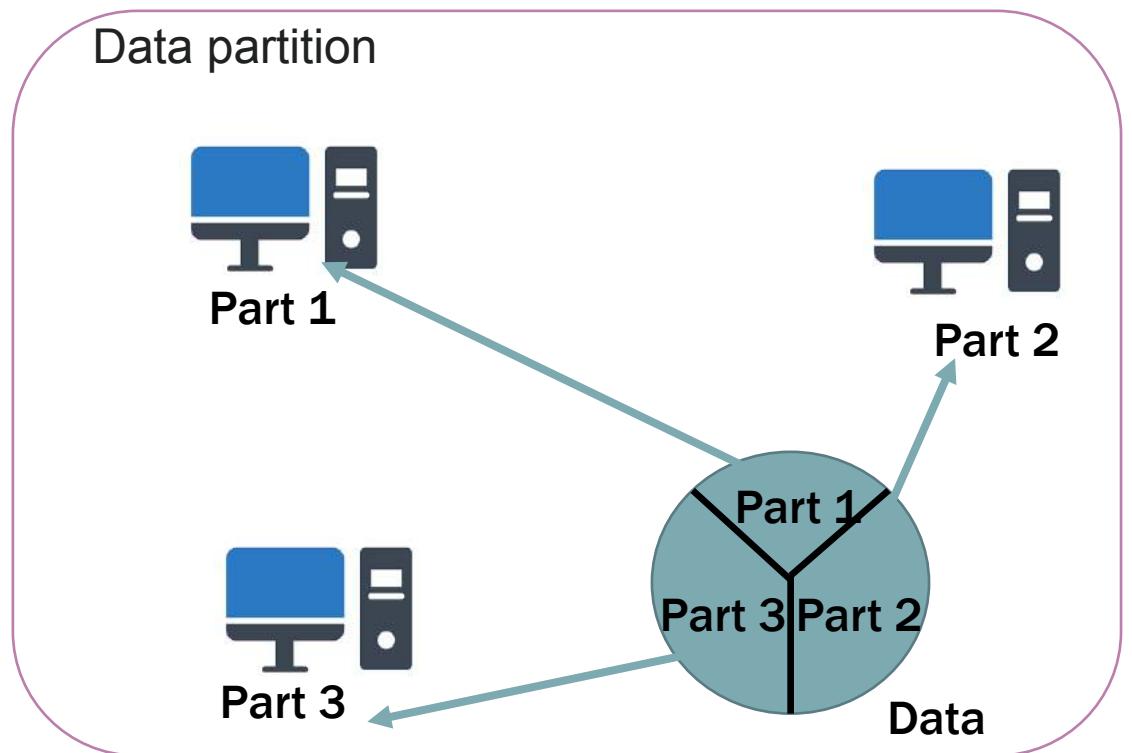
- How to organize the machines?
- **How to store data across machines?**
- How to compute using multiple machines?

DATA PARTITION AND DATA REPLICATION

- Data partition: partition the data into different machines
- Data replication: each data item can be replicated to multiple copies.



DATA PARTITION AND DATA REPLICA



EXAMPLE: DATA PARTITION

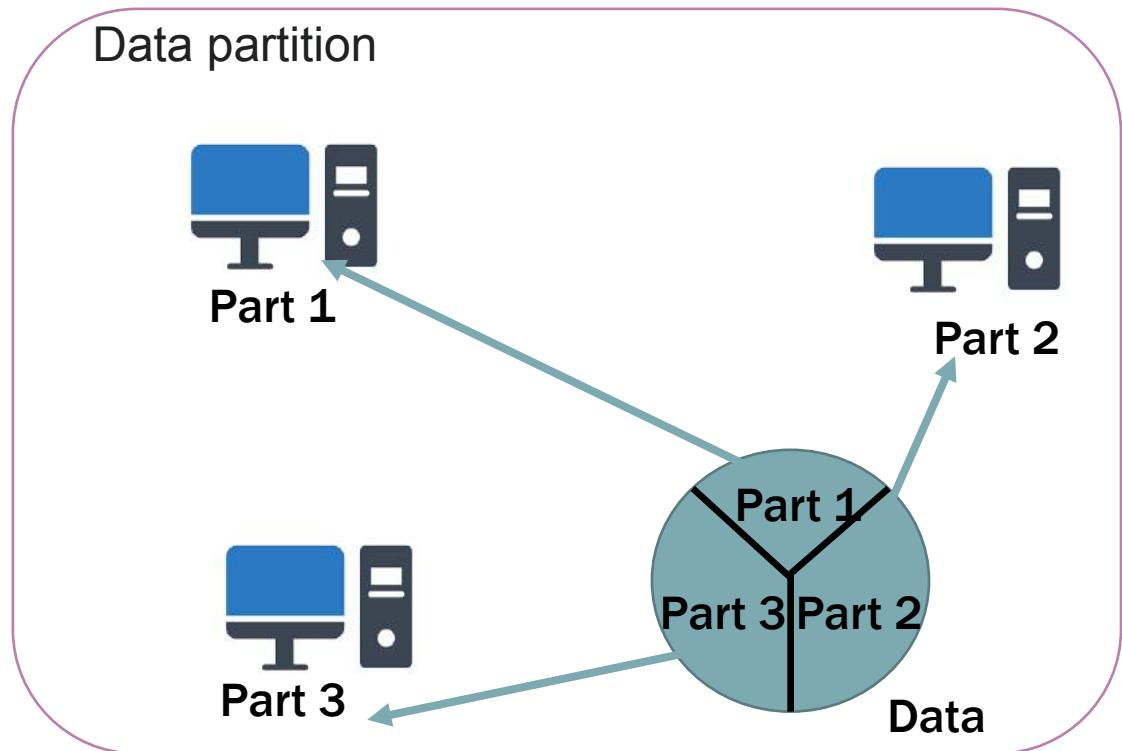
Name	Age
Alex	19
Bob	22
Kyle	18
Nelson	23
Ben	19
Terry	21

Name	Age
Alex	19
Bob	22

Name	Age
Kyle	18
Nelson	23

Name	Age
Ben	19
Terry	21

DATA PARTITION



Advantages:

- No space amplification
(i.e., the total data size does not change)
- Workload can be distributed to multiple machines.

DATA PARTITION

Select students whose ages < 20



Name	Age
Alex	19
Bob	22



Name	Age
Kyle	18
Nelson	23



Name	Age
Ben	19
Terry	21

Name	Age
Alex	19
Bob	22
Kyle	18
Nelson	23
Ben	19
Terry	21

DATA PARTITION

Select students whose ages < 20
(every machine does some computation)



Name	Age
Alex	19
Bob	22



Name	Age
Kyle	18
Nelson	23



Name	Age
Ben	19
Terry	21

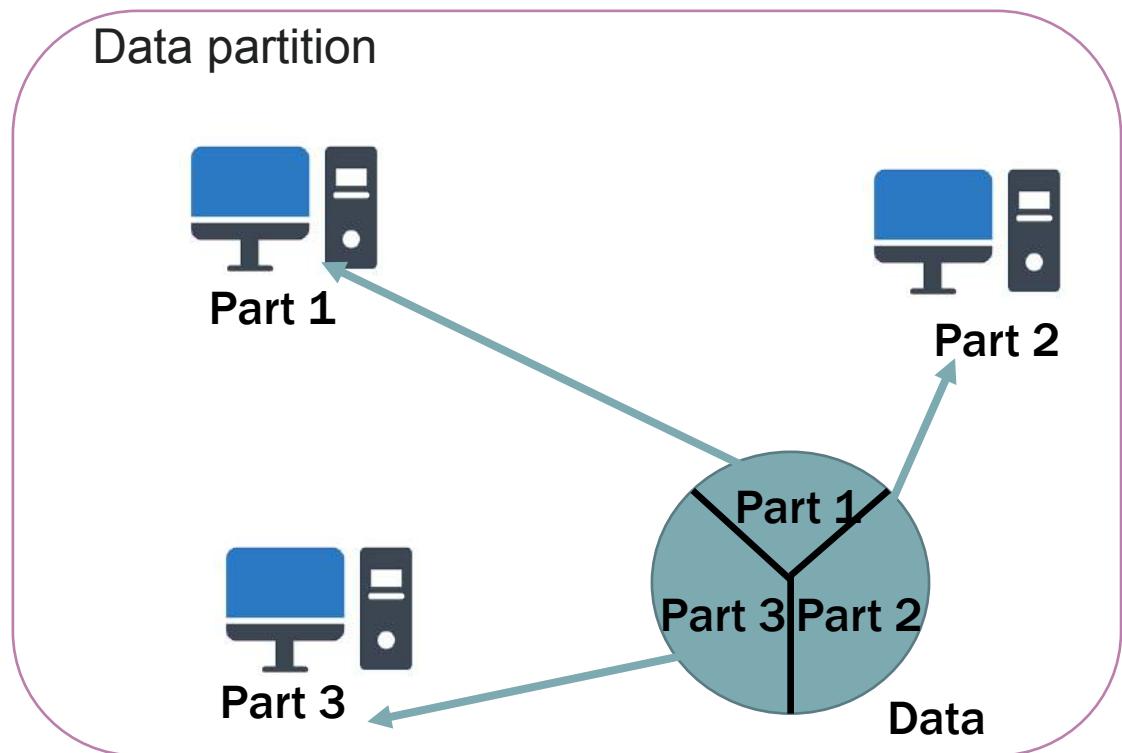
Name	Age
Alex	19
Bob	22
Kyle	18
Nelson	23
Ben	19
Terry	21



ANALYSIS

- Suppose the cost of doing a task on data D is $F(D)$.
- Data D is divided into $D=\{D_1, D_2, \dots, D_i\}$, which means machine j holds data D_j .
- The distributed computation cost can be **estimated** by $\max\{F(D_1), \dots, F(D_i)\}$.
- Usually, we have $\max\{F(D_1), \dots, F(D_i)\} \ll F(D)$

DATA PARTITION



Disadvantages (if **only** using data partitioning):

- Machines can be off for many reasons.
- Data lost, and results are not accurate.

DATA PARTITION

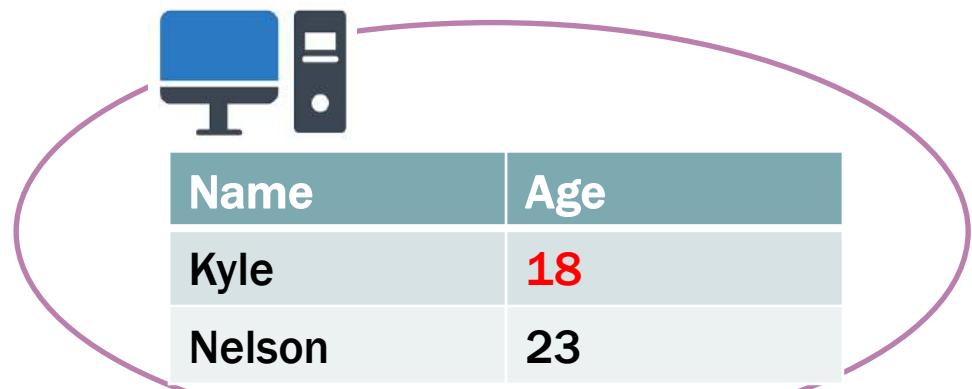
Only {Ben, Alex} are returned, no Kyle.



Name	Age
Alex	19
Bob	22



Name	Age
Ben	19
Terry	21

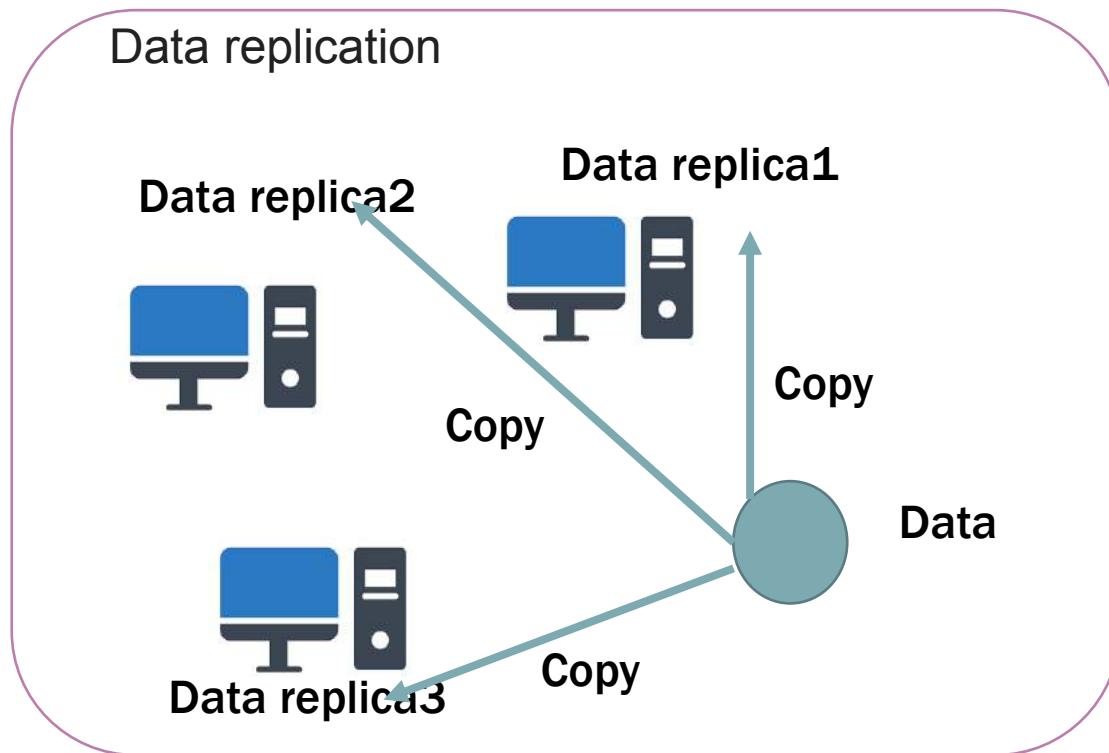


Name	Age
Alex	19
Bob	22
Kyle	18
Nelson	23
Ben	19
Terry	21

FAULT TOLERANCE

- Consider that a machine has 0.0001 probability of not being accessible at some time a day, and there are 1000 machines. What is the probability that all the machines in the cluster are always accessible within a month?
 - $(1-0.0001)^{1000*30}$ is about 5%.
- So, a large-scale distributed machine cluster always encounters machine failures. The process of handling machine failure is called fault tolerance.
- A **fault-tolerant** system ensures no loss of services when some machine fails.

DATA REPLICATION



Advantages:

- Better fault tolerance
- When one machine fails, data may be restored from other replicas

- The same set of data (can be a **subset** of data) may be stored at different machines.

DATA REPLICATION



Name	Age
Alex	19
Bob	22

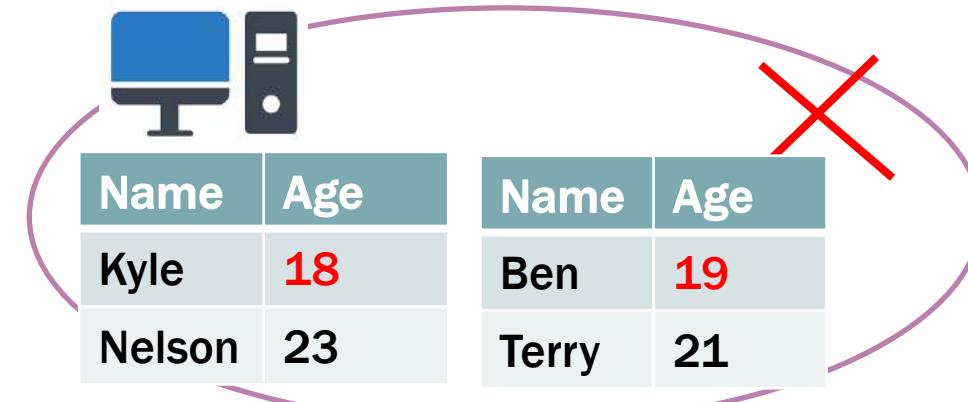
Name	Age
Kyle	18
Nelson	23



Name	Age
Ben	19
Terry	21

Name	Age
Alex	19
Bob	22

We can still get Kyle even when one machine holding the replica fails.



Name	Age
Alex	19
Bob	22
Kyle	18
Nelson	23
Ben	19
Terry	21

DATA REPLICATION

There is no free lunch. What is the cost of using data replication?



DATA REPLICATION

1st Disadvantage: More data in each machine, and probably a high processing cost in each machine



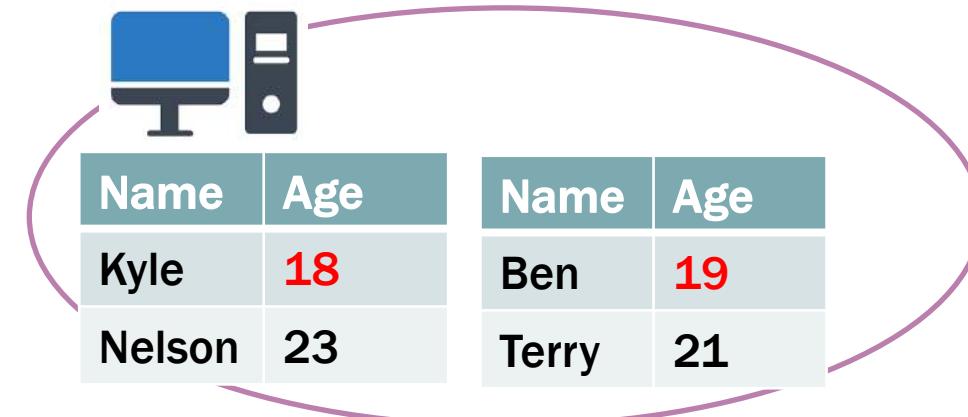
Name	Age
Alex	19
Bob	22

Name	Age
Kyle	18
Nelson	23



Name	Age
Ben	19
Terry	21

Name	Age
Alex	19
Bob	22



Name	Age
Alex	19
Bob	22
Kyle	18
Nelson	23
Ben	19
Terry	21

DATA REPLICATION: POSSIBLE SOLUTIONS

- Primary replica and secondary replicas
- During query, first access the primary replica. Only when the primary replica is not accessible, we go for one of secondary replicas.
- Maintains high query efficiency.

DATA REPLICATION



Name	Age
Alex	19
Bob	22

Name	Age
Kyle	18
Nelson	23



Name	Age
Kyle	18
Nelson	23

Name	Age
Ben	19
Terry	21



Name	Age
Ben	19
Terry	21

Name	Age
Alex	19
Bob	22

2nd disadvantage: Consistency between the replicas

Name	Age
Alex	19
Bob	22
Kyle	18
Nelson	23
Ben	19
Terry	21

DATA REPLICATION

Updating one tuple requires updating all the replicas containing the tuple, causing delays (i.e., low availability).

Other options?



STRICT CONSISTENCY AND EVENTUAL CONSISTENCY

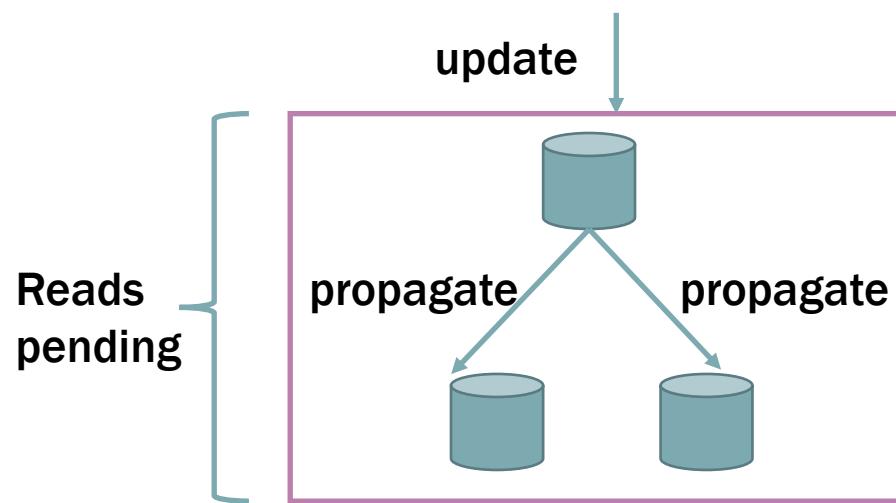
- Modern systems consider tradeoffs between **consistency** and **availability**.
- **Strict** (replica) consistency
 - When an update is committed to one replica, all the other replicas must be immediately updated as well, before handling any data reads.
 - Strongly consistent between data replicas
 - Low availability

STRICT CONSISTENCY AND EVENTUAL CONSISTENCY

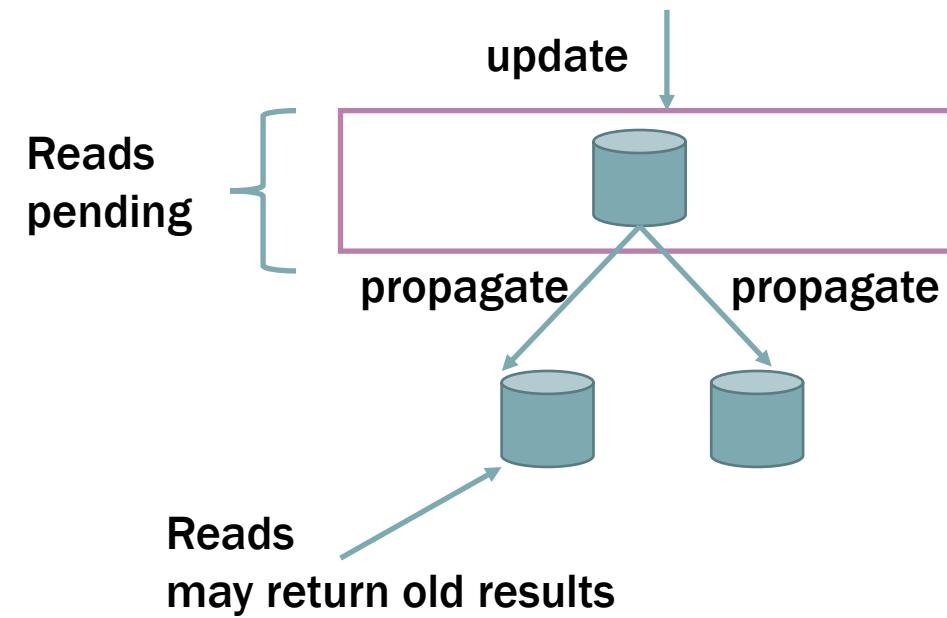
- Modern systems consider tradeoffs between consistency and availability.
- **Eventual (replica) consistency**
 - Update is first committed to one replica, and once updated, the data reads are allowed. The updates will be ultimately propagated to other replicas.
 - Weak consistency
 - High availability

STRICT CONSISTENCY AND EVENTUAL CONSISTENCY

Strict consistency



Eventual consistency



REMARKS

- In distributed systems, there is a tradeoff between consistency and system availability

- Usually, bank or financial related systems may choose strict consistency; web applications which allow some inconsistency may choose eventual consistency.

DISTRIBUTED SYSTEMS FOR BIG DATA: CHALLENGES

- How to organize the machines?
- How to store data across machines?
- **How to compute using multiple machines?**

TYPICAL PROCEDURE OF DISTRIBUTED COMPUTATION

Master-Slave Model:

Step 1: data is partition to different machines

Step 2: master assigns tasks to each slave (or slaves request tasks from the master)

Step 3: slaves accomplish their own tasks locally and send results to master if needed.

Step 4: master aggregates the results and go back to Step 2 if needed.

Note: in practice, some detailed steps vary depending on the specific computation tasks.

TYPICAL PROCEDURE OF DISTRIBUTED COMPUTATION

Fully Distributed Model:

Step 1: data is partitioned to different machines

Step 2: each machine gathers the received messages (if any)

Step 3: each machine conducts local computation

Step 4: each machine sends messages to some other machines and goes to step 2 if needed.

Note: in practice, some detailed steps vary depending on the specific computation tasks.

SUMMARY OF DISTRIBUTED COMPUTATION

- Two ways of organizing machines
 - Fully distributed
 - Master-Slave
- Two data storage methods
 - Data partition
 - Data replication
- Two typical ways of distributed computation

BIG DATA MANAGEMENT

CE/CZ4123

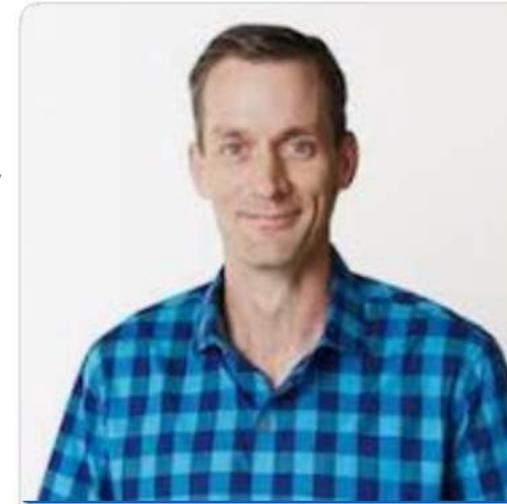
DISTRIBUTED SYSTEMS AND MAP-REDUCE

MapReduce:

A general distributed paradigm

DISTRIBUTED COMPUTATION PARADIGM

“The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.”



Jeff Dean

Lead of Google AI

PROGRAMMING WITH DISTRIBUTED MACHINES

- Programming with distributed machines is hard
- Compared with the codes to run in a single machine, coding for distributed systems requires
 - Handling communication between machines
 - Coping with machine failures
 - Data replicas
 - Data partitioning
 - ...

**Coding from scratch for distributed Machines is painful,
even for simple computation.**



People start to develop ideas to ease the process:

- OpenMPI

- Hadoop

Enhancement

- Spark

Realization of MapReduce

One popular programming model

- ...

GOOGLE'S MAPREDUCE

MapReduce is now one of the most widely used programming paradigm in distributed processing.

Hides details such as parallelization, fault-tolerance, data distribution and load balancing

Realize the computation using *map* and *reduce* interfaces

PROGRAMMING MODEL OF MAPREDUCE

`map(String key, String value)`

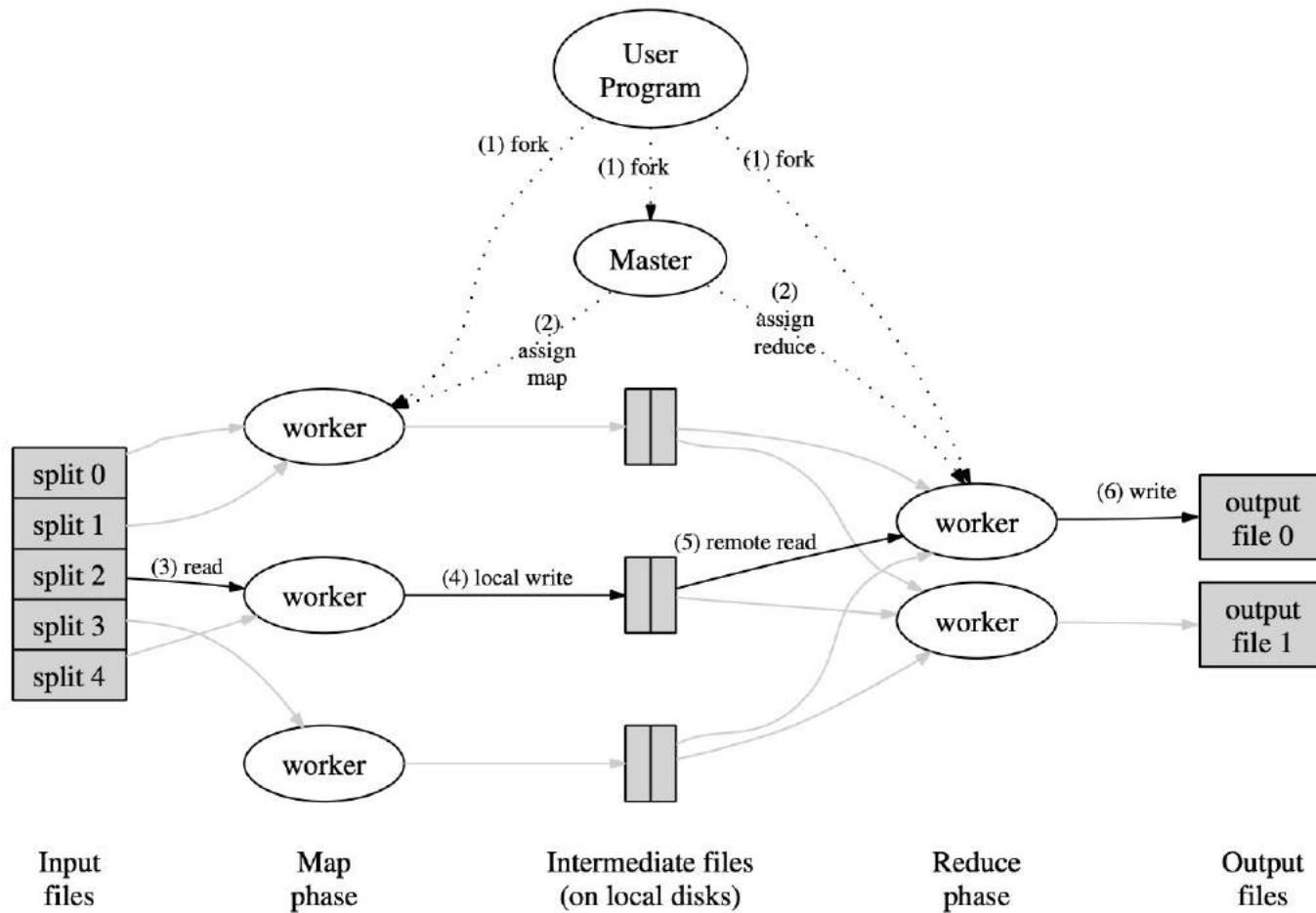
`reduce(String key, Iterator values)`

`map (k1,v1) → list(k2,v2)`  **Defined by you**

Same k₂ will be gathered together and passed to the reduce workers (the system does it automatically)

`reduce (k2,list(v2)) → list(k3, v3)`

EXECUTION OVERVIEW



Reading: [MapReduce: Simplified Data Processing on Large Clusters](#)

EXECUTION OVERVIEW

- The map invocations are distributed across machines by partitioning the input data into M splits (each split: 16-64MB).

One split → one map task (conducted in one map worker)

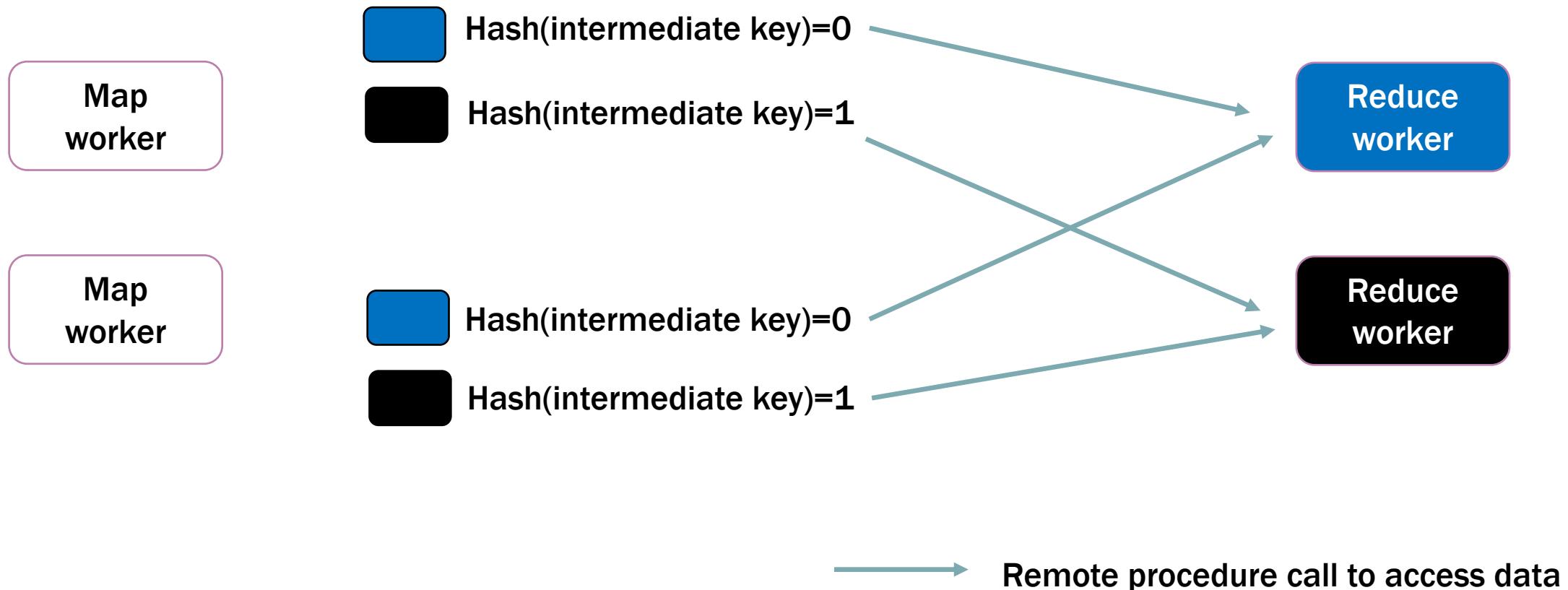


A set of key-value pairs. → Map function

- The reduce invocations are distributed by partitioning the **intermediate key space** into R pieces using $\text{hash(key)} \bmod R$
- There are M **map tasks** and R **reduce tasks**.

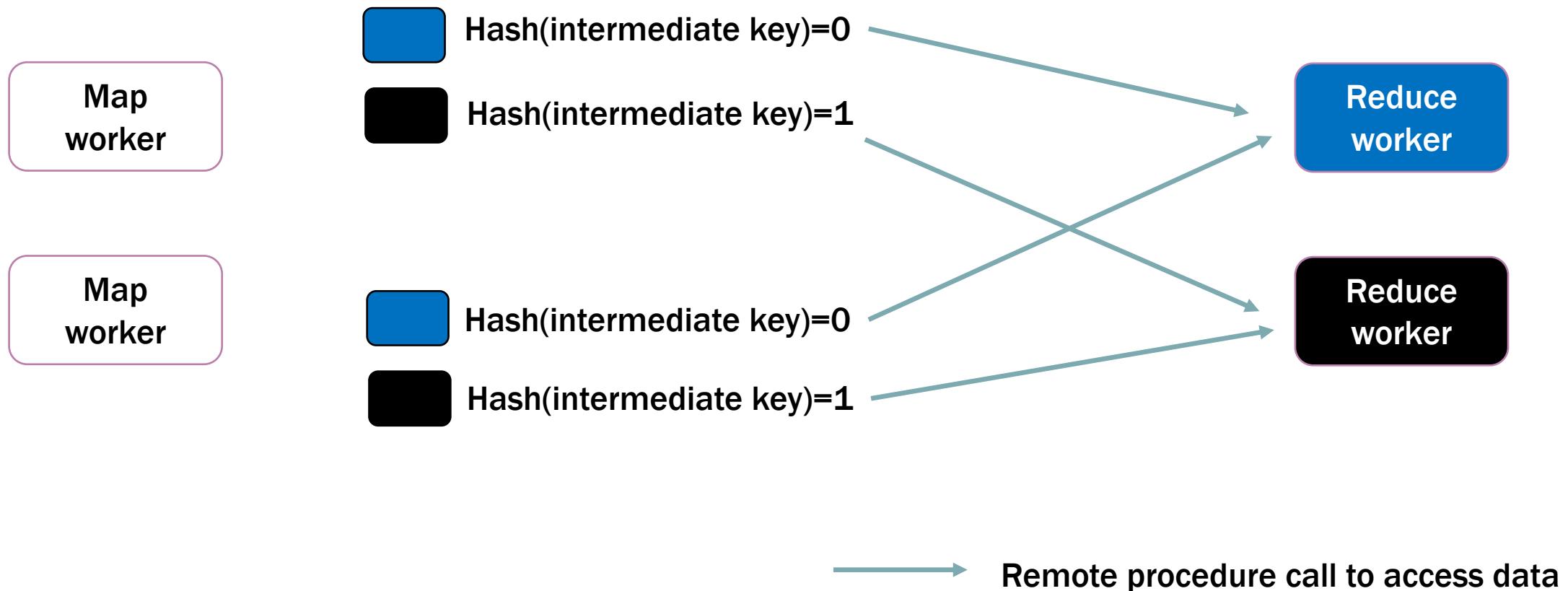
EXECUTION OVERVIEW

- ❑ Intermediate map output is stored locally (in disk). The output is divided into R parts (each for one reduce worker to read)

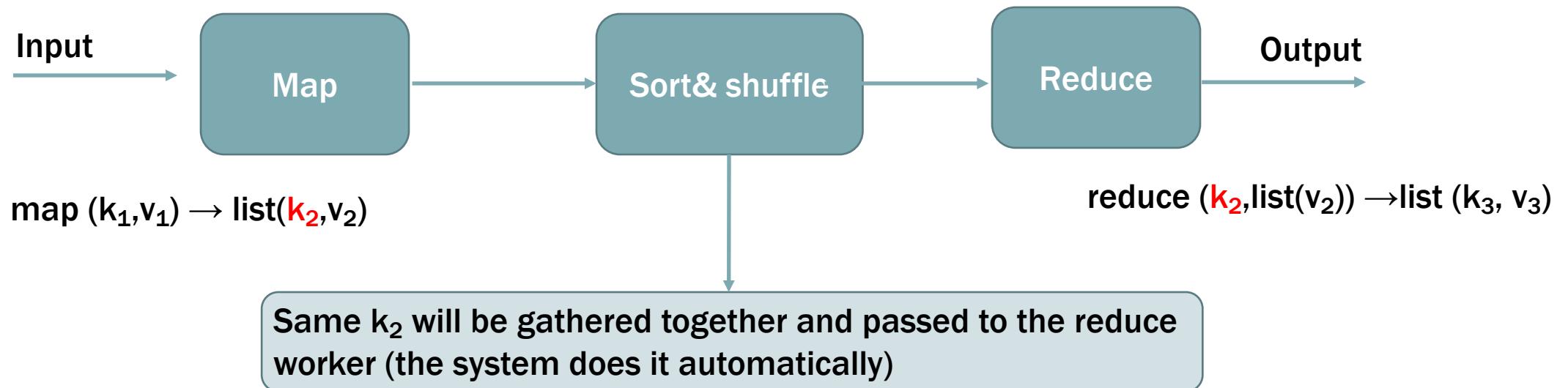


EXECUTION OVERVIEW

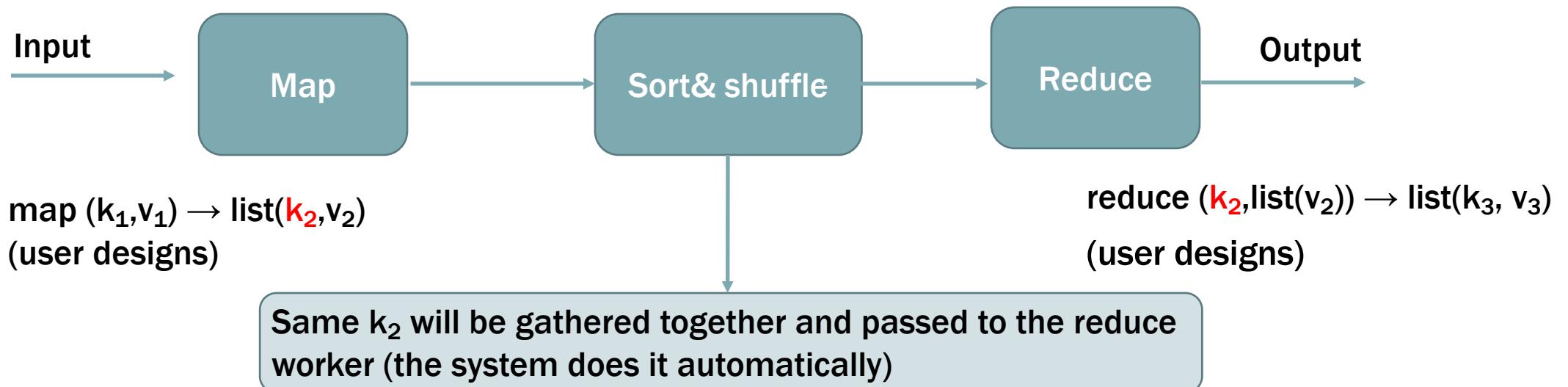
- Each reduce worker output results in local files



FROM THE PERSPECTIVE OF ALGORITHMIC DESIGNER



FROM THE PERSPECTIVE OF ALGORITHMIC DESIGNER



EXAMPLE: COMPUTING FREQUENCIES



(User, product) data

100 Billion items

{
Alex, bag
Bob, shoes
...}



Which item has been sold the most?

MAPREDUCE

Data sets (100B)

(u1, p1)

(u2, p2)

...



A1



A2

...

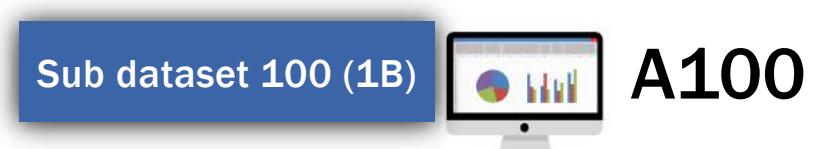


A100

MAPREDUCE



...



MAPREDUCE

Map

Sub dataset 1 (1B)

Input



A1

Sub dataset 2 (1B)

Input



A2

...

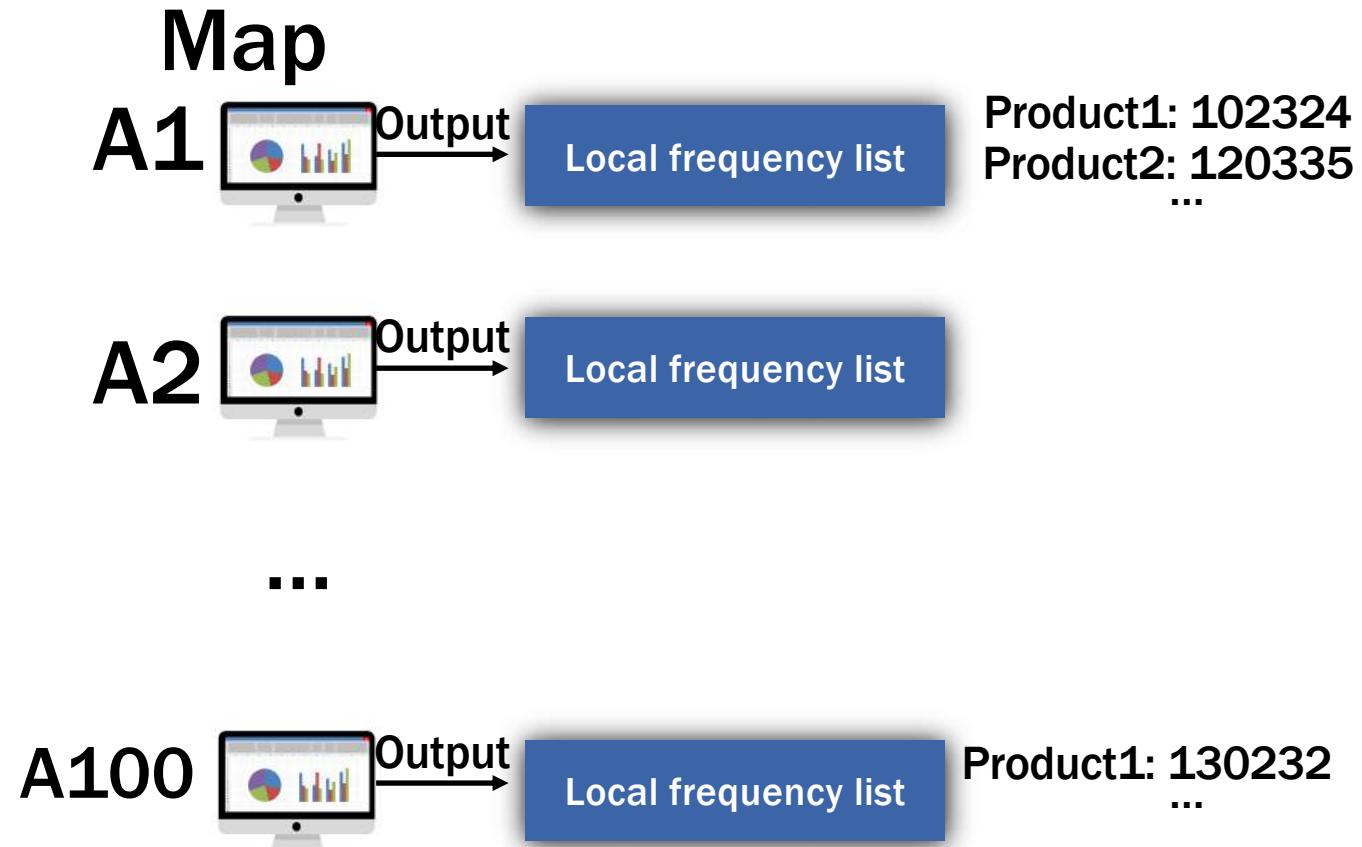
Sub dataset 100 (1B)

Input

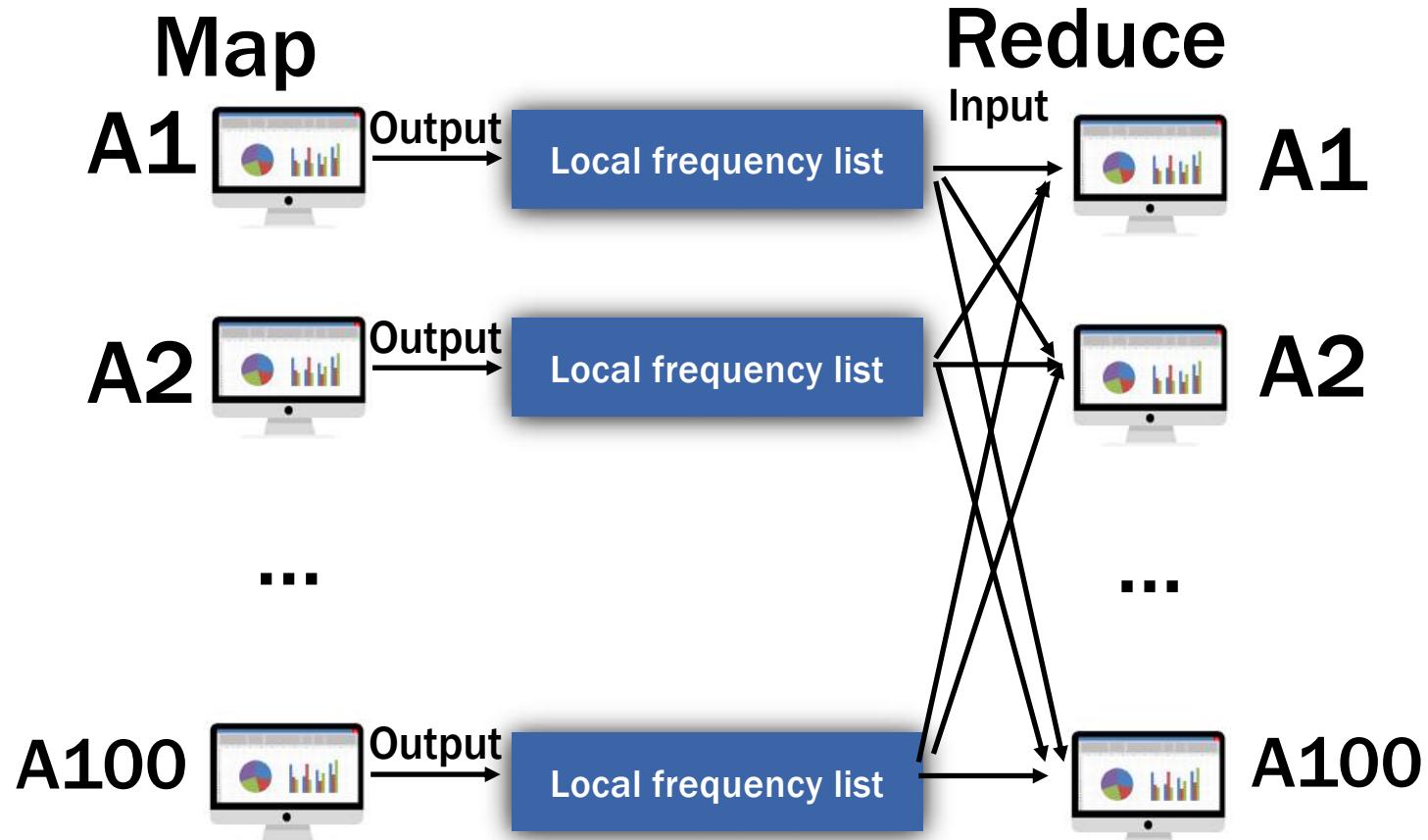


A100

MAPREDUCE

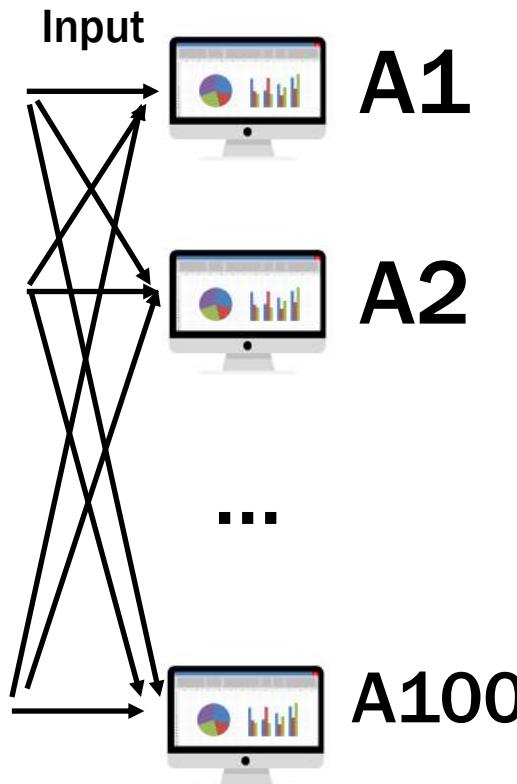


MAPREDUCE



MAPREDUCE

Reduce

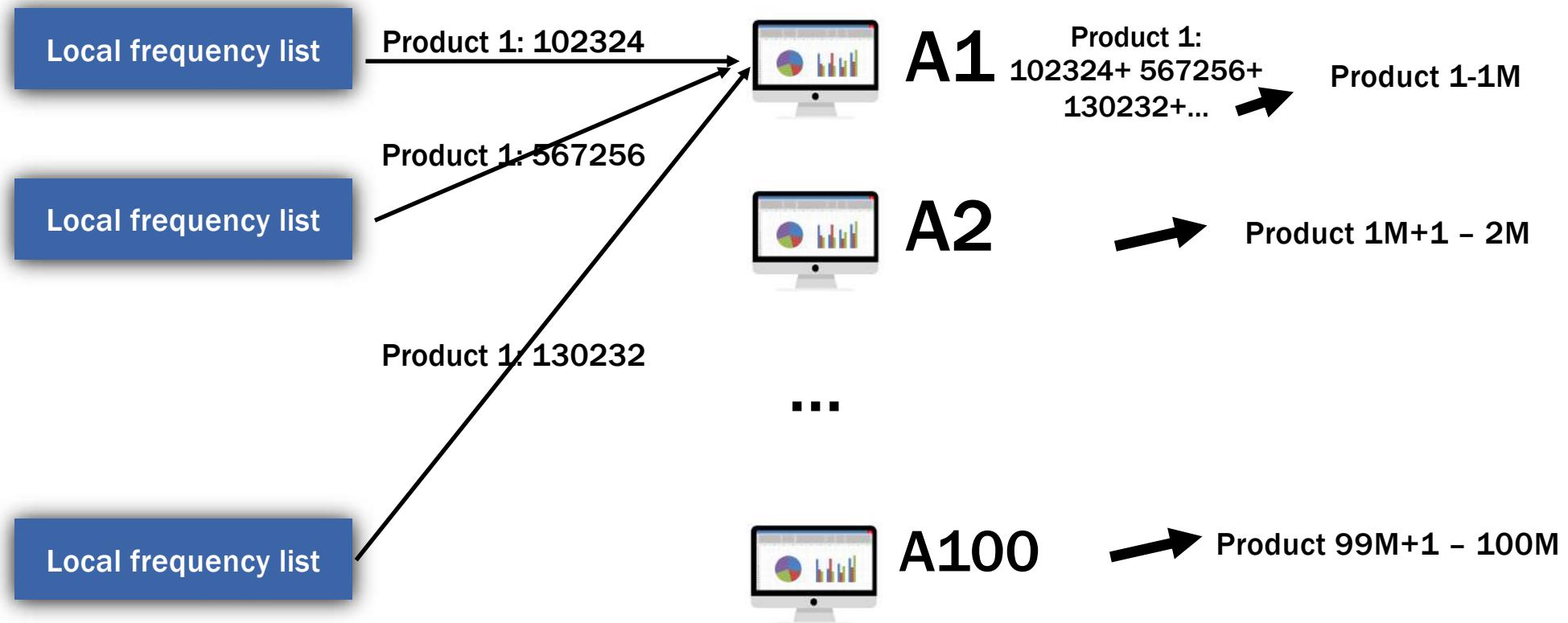


Each reduce worker processes a subset of products.

For example, if there are 100M products, then
A1 handles the Product 1 - 1M,
A2 handles Product 1M+1 – 2M,
...

MAPREDUCE

Reduce



MAPREDUCE PSEUDOCODE

```
map(String key, String value) // key: user // value: product id
{
    Emit-Intermediate(value, "1");
}
```

Note: Map() can be implemented in a simple way that it does not even need to compute the local frequency list for each subset.

MAPREDUCE PSEUDOCODE

```
reduce(String key, Iterator<String> values)
// key: a product
// values: a list of counts
{
    int result = 0;
    for (each v in values){
        result += ParseInt(v);
    }
    Emit(Key,AsString(result));
}
```

INPUT/OUTPUT IN EACH PHASE

Input

(key, value) pairs

Map Output

(key, value) pairs

Input of Reduce

(key, value-list) pairs

Reduce Output

(key, value) pairs

EXAMPLE INPUT/OUTPUT

Input

(Alex, product 1) (Alex, product 2) (Bob, product 1) (Bob, product 3)

Map Output

(key, value) pairs

Input of Reduce

(key, value-list) pairs

Reduce Output

(key, value) pairs

EXAMPLE INPUT/OUTPUT

Input

(“Alex”, “bag001”) (“Alex”, “bag002”) (“Bob”, “bag001”) (“Bob”, “bag003”)

Map Output

(“bag001”, “1”) (“bag002”, “1”) (“bag001”, “1”) (“bag003”, “1”)

Input of Reduce

?

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Alex”, “bag001”) (“Alex”, “bag002”) (“Bob”, “bag001”) (“Bob”, “bag003”)

Map Output

(“bag001”, “1”) (“bag002”, “1”) (“bag001”, “1”) (“bag003”, “1”)

Input of Reduce

?

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Alex”, “bag001”) (“Alex”, “bag002”) (“Bob”, “bag001”) (“Bob”, “bag003”)

Map Output

(“bag001”, “1”) (“bag002”, “1”) (“bag001”, “1”) (“bag003”, “1”)

Input of Reduce

(“bag001”, {“1”, “1”}) (“bag002”, “1”) (“bag003”, “1”)

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Alex”, “bag001”) (“Alex”, “bag002”) (“Bob”, “bag001”) (“Bob”, “bag003”)

Map Output

(“bag001”, “1”) (“bag002”, “1”) (“bag001”, “1”) (“bag003”, “1”)

Input of Reduce

(“bag001”, {“1”, “1”}) (“bag002”, “1”) (“bag003”, “1”)

Reduce Output

(“bag001”, “2”) (“bag002”, “1”) (“bag003”, “1”)

ANOTHER EXAMPLE INPUT/OUTPUT

Input

(“A”, “001”) (“A”, “001”) (“A”, “002”) (“A”, “003”) (“B”, “004”)

Map Output

(“001”, “1”) (“001”, “1”) (“002”, “1”) (“003”, “1”) (“004”, “1”)

Input of Reduce

(“001”, {"1", "1"}) (“002”, “1”) (“003”, “1”) (“004”, “1”)

Reduce Output

(“001”, “2”) (“002”, “1”) (“003”, “1”) (“004”, “1”)

EXAMPLE: INVERTED INDEX

Doc 1

NTU's 30th anniversary marks a history of sparking innovative ideas and addressing global challenges. Since its inauguration in 1991, NTU has been home to an exceptional community of faculty, students and staff who have helped propel the university to excellence. To celebrate our birthday year we have created a digital time capsule that contains a collection of 30 past and present memories contributed by members of the NTU community. Sealed on 9 December 2021 to commemorate NTU's momentous achievements, the time capsule will be opened in 2041 at NTU's Golden Jubilee. View the artefacts at NTU's 30th anniversary exhibition, which is now open to public, or visit the time capsule website to learn more about the university's milestone moments.

EXAMPLE: INVERTED INDEX

Doc 2

The NTU School of Computer Science and Engineering (SCSE) was established in November 1988 as the School of Applied Science (SAS) offering Bachelor Degree in Computer Technology, the first of its kind in Singapore. We were part of NTU's predecessor, Nanyang Technological Institute (NTI) that was set up in August 1981 with a charter to train three-quarters of Singapore's engineers. Within 4 years of operation, NTI was singled out as one of the best engineering institutions in the world by the Commonwealth Engineering Council. The Council reached this verdict after an extensive 4-year study of the courses offered by engineering institutions worldwide.

A TYPICAL QUERY

Given a word, find out which documents contain that word.

Example:

“NTU”: Doc 1, Doc 2

“August”: Doc 2

“achievements”: Doc 1

A TYPICAL QUERY

Given a word, find out which documents contain that word.

Example:

“NTU”: Doc 1, Doc 2

“August”: Doc 2

“achievements”: Doc 1

Extremely inefficient!

A TYPICAL QUERY

Given a word, find out which documents contain that word.

How about precompute all the document lists for each possible word in the corpus?

This is called **inverted index**.

EXAMPLE: INVERTED INDEX

Suppose we have a list of documents, each of which contains a set of words. We need to construct the inverted index, such that given a word, we can directly extract the list of documents (by Ids) containing the word. Please describe the algorithm using the MapReduce model.

Example input:

(“doc1”, “I study at NTU”)

(“doc2”, “NTU is famous”)

Example output:

(“I”, {"doc1"}), (“study”, {"doc1"}), (“at”, {"doc1"}), (“NTU”, {"doc1, doc2"}), (“is”, {"doc2"}), (“famous”, {"doc2"})

DESIGN: THE INPUT TO MAP

Key: document ID (string)

Value: document content (string)

DESIGN: MAP FUNCTION

Key: document ID (string)

Value: document content (string)

```
map(String docID, String docs){  
    Iterator<string> words = split-to-words(docs);  
    for(string word in words){  
        Emit-Intermediate(word, docID);  
    }  
}
```

DESIGN: REDUCE FUNCTION

Key: document ID (string)

Value: document content (string)

```
reduce(String word, Iterator<String> docIDs):  
    // key: a word // values: a list of document ids  
    {  
        Emit(word, ToString(docIDs));  
    }
```

EXAMPLE INPUT/OUTPUT

Input

(“Doc1”, “Big data management is a course.”)

(“Doc2”, “Introduction to databases is a course.”)

Map Output

?

Input of Reduce

?

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Doc1”, “Big data management is a course.”)

(“Doc2”, “Introduction to databases is a course.”)

Map Output

(“Big”, “Doc1”) (“data”, “Doc1”) (“management”, “Doc1”) (“is”, “Doc1”) (“a”, “Doc1”)
 (“course”, “Doc1”) (“Introduction”, “Doc2”) (“to”, “Doc2”) (“databases”, “Doc2”) (“is”,
 “Doc2”) (“a”, “Doc2”) (“course”, “Doc2”)

Input of Reduce

?

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Doc1”, “Big data management is a course.”)

(“Doc2”, “Introduction to databases is a course.”)

Map Output

(“Big”, “Doc1”) (“data”, “Doc1”) (“management”, “Doc1”) (“is”, “Doc1”) (“a”, “Doc1”)
 (“course”, “Doc1”) (“Introduction”, “Doc2”) (“to”, “Doc2”) (“databases”, “Doc2”) (“is”,
 “Doc2”) (“a”, “Doc2”) (“course”, “Doc2”)

Input of Reduce

(“Big”, {"Doc1"}) (“data”, {"Doc1"}) (“management”, {"Doc1"}) (“is”, {"Doc1", "Doc2"})
 (“a”, {"Doc1", "Doc2"}) (“course”, {"Doc1", "Doc2"})

Reduce Output

?

EXAMPLE INPUT/OUTPUT

Input

(“Doc1”, “Big data management is a course.”)

(“Doc2”, “Introduction to databases is a course.”)

Map Output

(“Big”, “Doc1”) (“data”, “Doc1”) (“management”, “Doc1”) (“is”, “Doc1”) (“a”, “Doc1”)
 (“course”, “Doc1”) (“Introduction”, “Doc2”) (“to”, “Doc2”) (“databases”, “Doc2”) (“is”,
 “Doc2”) (“a”, “Doc2”) (“course”, “Doc2”)

Input of Reduce

(“Big”, {"Doc1"}) (“data”, {"Doc1"}) (“management”, {"Doc1"}) (“is”, {"Doc1", "Doc2"})
 (“a”, {"Doc1", "Doc2"}) (“course”, {"Doc1", "Doc2"})

Reduce Output

(“Big”, “Doc1”) (“data”, “Doc1”) (“management”, “Doc1”) (“is”, “Doc1;Doc2”) (“a”,
 “Doc1;Doc2”) (“course”, “Doc1;Doc2”)

Looks like most of the tasks can be simply finished using one
MapReduce job

There are more complicated cases where we need multiple jobs

BIG DATA MANAGEMENT

CE/CZ4123

DISTRIBUTED SYSTEMS AND MAP-REDUCE (MORE EXAMPLES)

MapReduce:

A general distributed paradigm

In this lecture, we will see more complicated examples of MapReduce based algorithms.

- Table Join
- Shortest Path Computation
- PageRanks

NOTES ON PSEUDOCODE FOR MAPREDUCE ALGORITHMS

- The pseudocode focuses on “thinking in MapReduce”, and so it is okay to use any understandable syntax (C-like or Java-like).
- For presenting complicated MapReduce Algorithms, we can use more complicated object class (data structure) than “String” for both key and value. Then we should describe the composition of the complicated data structure as well.

EXAMPLE: JOINING TWO TABLES

Primary key		
A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Foreign key		
C	D	E
c1	d1	e1
c1	d2	e2
c2	d3	e3
c3	d4	e4
c3	d5	e5

Map Input

Key: Line number
Value: A;B;C

Key: Line number
Value: C;D;E

JOINING TWO TABLES

Primary key		
A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5



Map Input

Key: Line number
Value: A;B;C

Foreign key		
C	D	E
c1	d1	e1
c1	d2	e2
c2	d3	e3
c3	d4	e4
c3	d5	e5



Key: Line number
Value: C;D;E

- ❑ If the tuples in two tables are in the same file, then we need to know the size of the 1st table to classify the tuples.

JOINING TWO TABLES

Primary key		
A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Foreign key		
C	D	E
c1	d1	e1
c1	d2	e2
c2	d3	e3
c3	d4	e4
c3	d5	e5

Map Output

Key: C
Value: **T1;A;B**

Key: C
Value: **T2;D;E**

JOINING TWO TABLES

Primary key		
A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Reduce Input

Key: C

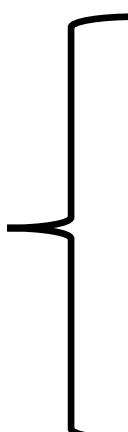
Value: {T1;A;B, T2;D;E}

Foreign key

C	D	E
c1	d1	e1
c1	d2	e2
c2	d3	e3
c3	d4	e4
c3	d5	e5

Key: c1

Value: {T1;a1;b1, T2;d1;e1, T2;d2;e2}



Key: c2

Value: {T1;a2;b2, T2;d3;e3}

Key: c3

Value: {T1;a3;b3, T2;d4;e4, T2;d5;e5}



JOINING TWO TABLES

Primary key		
A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Foreign key		
C	D	E
c1	d1	e1
c1	d2	e2
c2	d3	e3
c3	d4	e4
c3	d5	e5

Reduce

Key: c1

Value: {T1;a1;b1, T2;d1;e1, T2;d2;e2}

→ (T1;a1;b1, T2;d1;e1)

(T1;a1;b1, T2;d2;e2)

↓ (a1;b1;c1;d1;e1)
(a1;b1;c1;d2;e2)

A FEW TIPS

- ❑ It is crucial to determine which attributes/features should be aggregated on → Intermediate **key**
- ❑ The intermediate **value** can be complex; it can include a lot of useful information for you to finish the task.

WHAT ABOUT DISTANCE-BASED JOIN?

A	B	C
a1	b1	1
a2	b2	50
a3	b3	100

F	D	E
2	d1	e1
99	d2	e2
49	d3	e3
...
...

Join Column C of Table 1 and Column F of Table 2 based on the condition that $|C-F|<50$. Assume that the values of Column C and Column F are positive integers.

If we directly use the values of Column C and Column F, qualified tuples will not be joined together. What should we do?



THE FIRST IDEA

Problem solved?



0

50

100

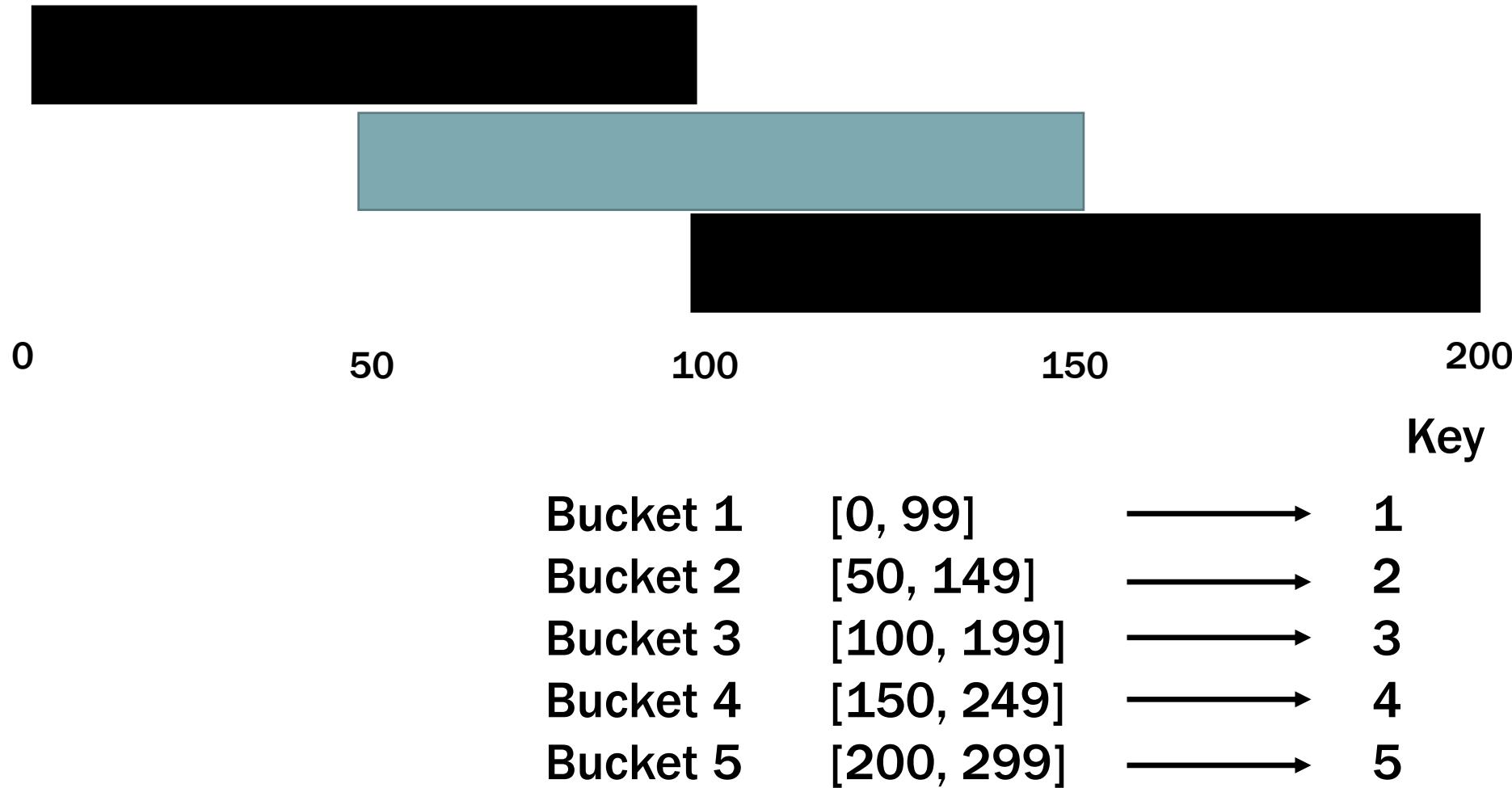
150

Key

Bucket 1	[0, 49]	→	1
Bucket 2	[50, 99]	→	2
Bucket 3	[100, 149]	→	3
Bucket 4	[150, 199]	→	4
Bucket 5	[200, 249]	→	5

Aggregate the values in the same bucket

FIXED THE IDEA



Any two values whose difference is at most 50 should fall into **at least one** buckets

INPUT

Assume the input key-value pairs are the following (corresponding to the example):

- 1 a1;b1;1
- 2 a2;b2;50
- 3 a3;b3;100
- 4 2;d1;e1
- 5 99;d2;e2
- 6 49;d3;e3

JOB1: MAP (PSEUDOCODE)

```
Map(int key, String value){  
    if (key<table1_size){  
        int value_C=get_value_C(value);  
        Emit-Intermediate(floor(value_C/50), “T1:”+value);  
        Emit-Intermediate(floor(value_C/50)+1, “T1:”+value);  
    }  
    else{  
        int value_F=get_value_F(value);  
        Emit-Intermediate(floor(value_C/50), “T2:”+value);  
        Emit-Intermediate(floor(value_C/50)+1, “T2:”+value);  
    }  
}
```

JOB1: REDUCE (PSEUDOCODE)

```
Reduce(int key, iterator<String> values){  
  
    List value_C_list, value_F_list, value_T1_list, value_T2_list;  
  
    for(String value : values){  
  
        if(value starts with "T1"){  
  
            int value_C=get_value_C(value);  
  
            value_C_list.add(value_C);  
  
            value_T1_list.add(get_tuple(value));  
  
        } else{  
  
            int value_F=get_value_F(value);  
  
            value_F_list.add(value_F);  
  
            value_T2_list.add(get_tuple(value));  
  
        }  
  
    }  
  
}
```

```
        for(int i=0;i<value_C_list.size();i++)  
  
            for(int j=0;j<value_F_list.size();j++){  
  
                int value_C=value_C_list[i];  
  
                int value_F=value_F_list[j];  
  
                if(|value_C-value_F|<50){  
  
                    Emit("", value_T1_list[i]+value_T2_list[j]);  
  
                }  
  
            }  
  
        }  
  
    }
```

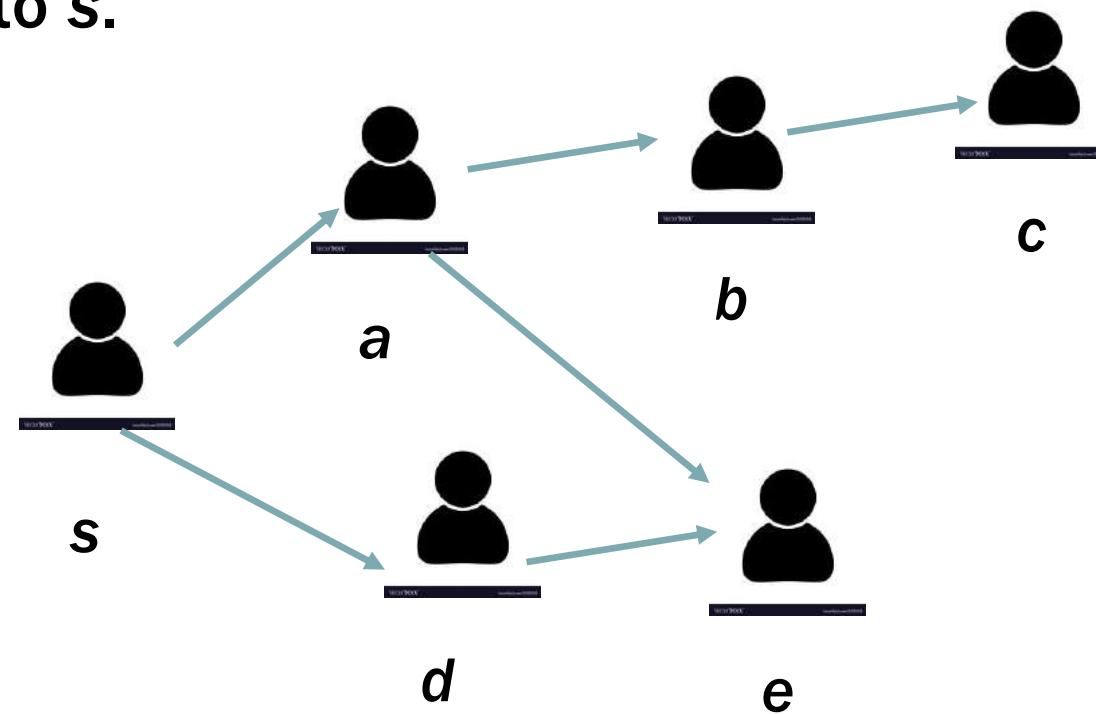
JOB2: REMOVE DUPLICATES

The output of Job1 may contain duplicates (why?)

It is easy to remove duplicates by another MapReduce job.

EXAMPLE: SHORTEST PATH COMPUTATION

Given a directed social network in the following form, find all the people within k-hops from a given source node s , and the corresponding hop distance to s .



EXAMPLE: SHORTEST PATH COMPUTATION

Given a directed social network in the following form, find all the people within k-hops from a given source node s , and the corresponding hop distance to s .

Example results ($k=2$):

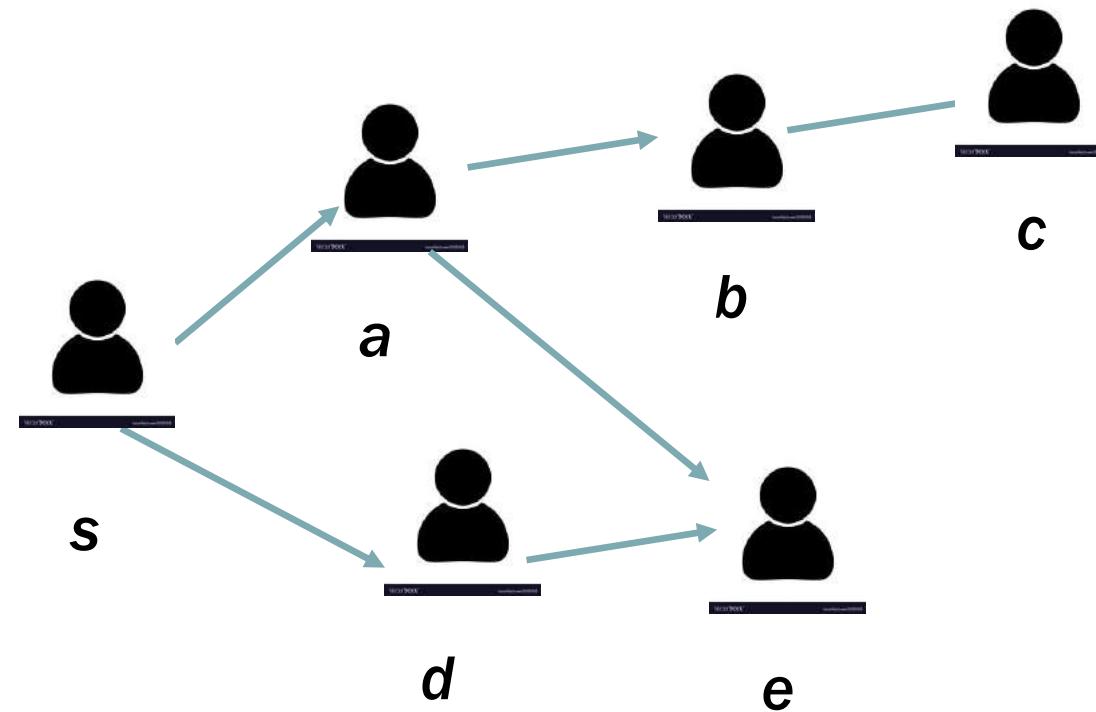
a,d,e,b

$\text{dis}(s,a)=1$

$\text{dis}(s,d)=1$

$\text{dis}(s,e)=2$

$\text{dis}(s,b)=2$



EXAMPLE: SHORTEST PATH COMPUTATION

Single-Machine Algorithm: Dijkstra's algorithm

- An efficient implementation of Dijkstra's algorithm often uses priorityqueue

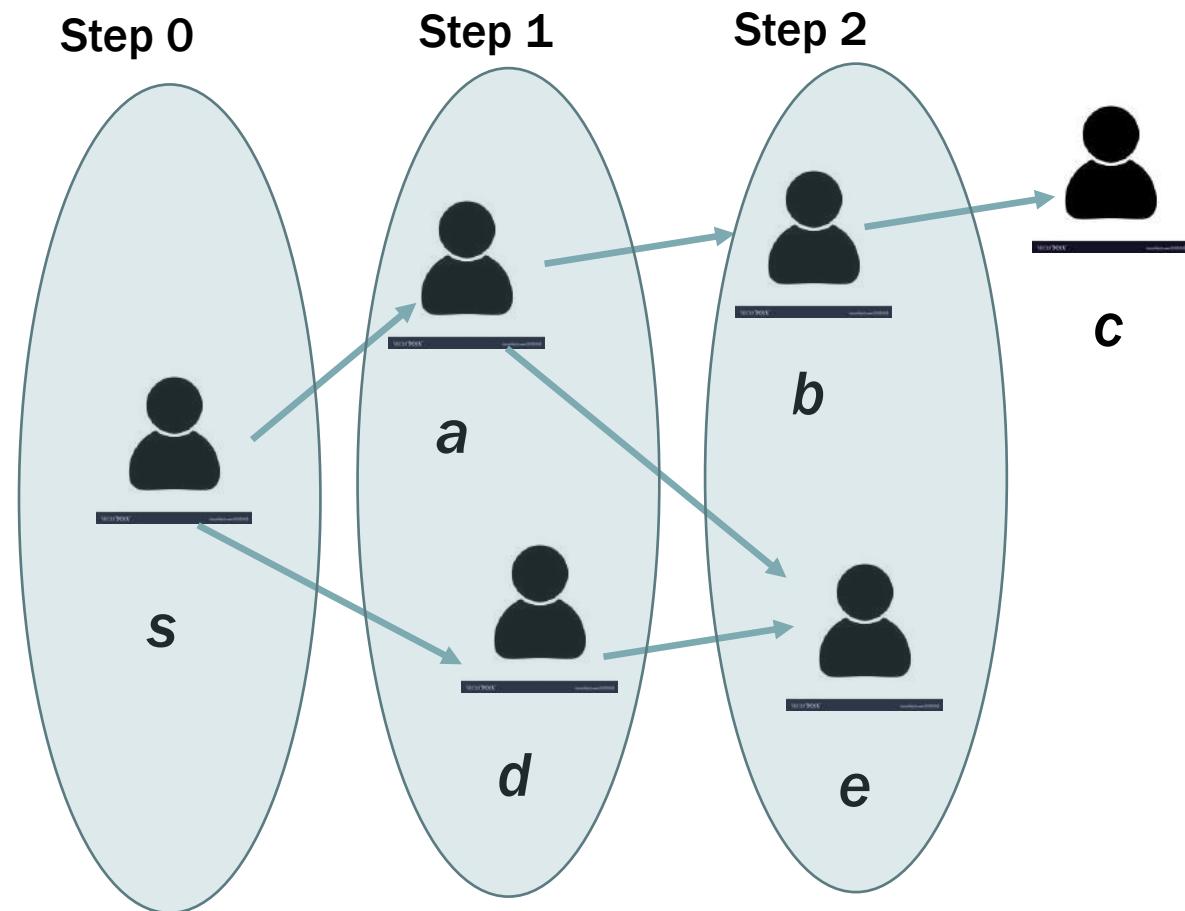
Not easy to parallelize the computation in MapReduce

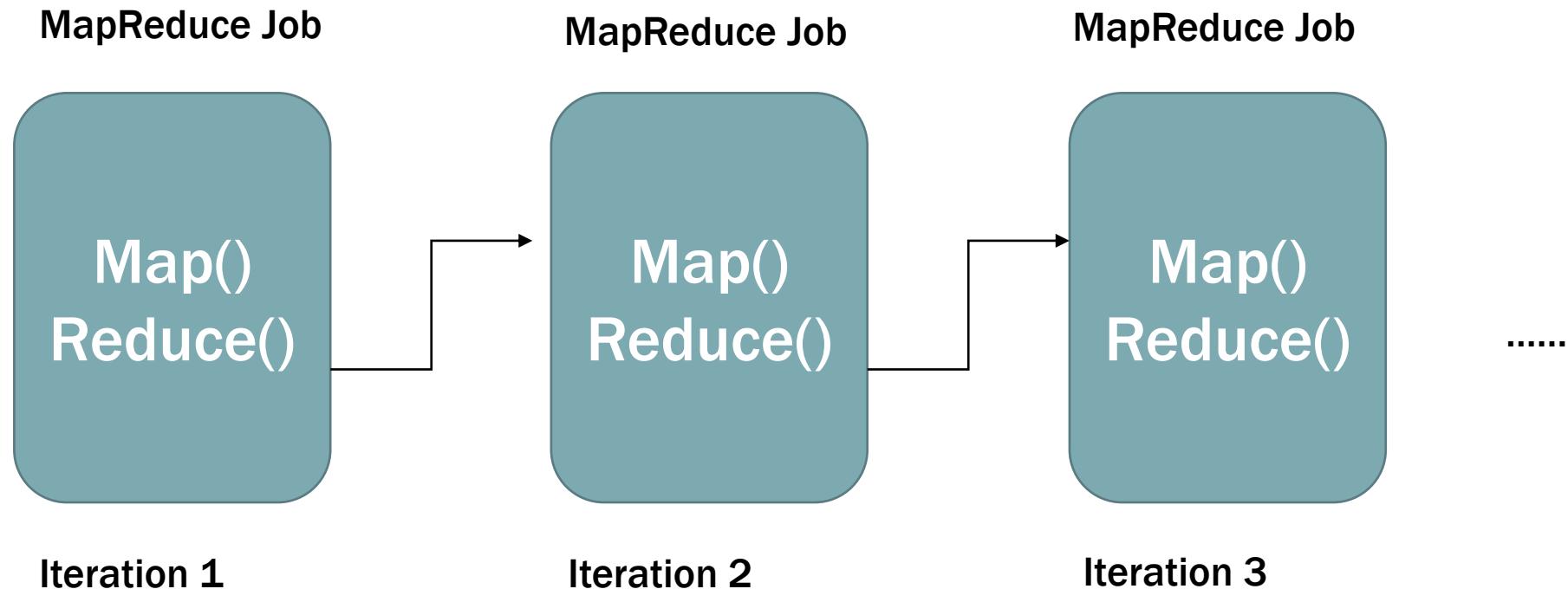
IDEA FOR DESIGNING MAPREDUCE FUNCTIONS

- Solution to the problem can be defined inductively

Intuition

- Initial step (Step 0): $\text{dis}(s)=0$ $\text{dis}(v)=+\infty$ for any other v .
- Step 1: For any node v_1 that is 1-hop from s , set $d(v_1)=\min\{d(v_1), 1\}$
- Step 2: For any node v_2 that can be reached in 2-hops from s , set $d(v_2)=\min\{d(v_2), 2\}$
- ...
- Step K : For any node v_K that can be reached K -hops from s , set $d(v_K)=\min\{d(v_K), K\}$





INPUT KEY-VALUE PAIRS

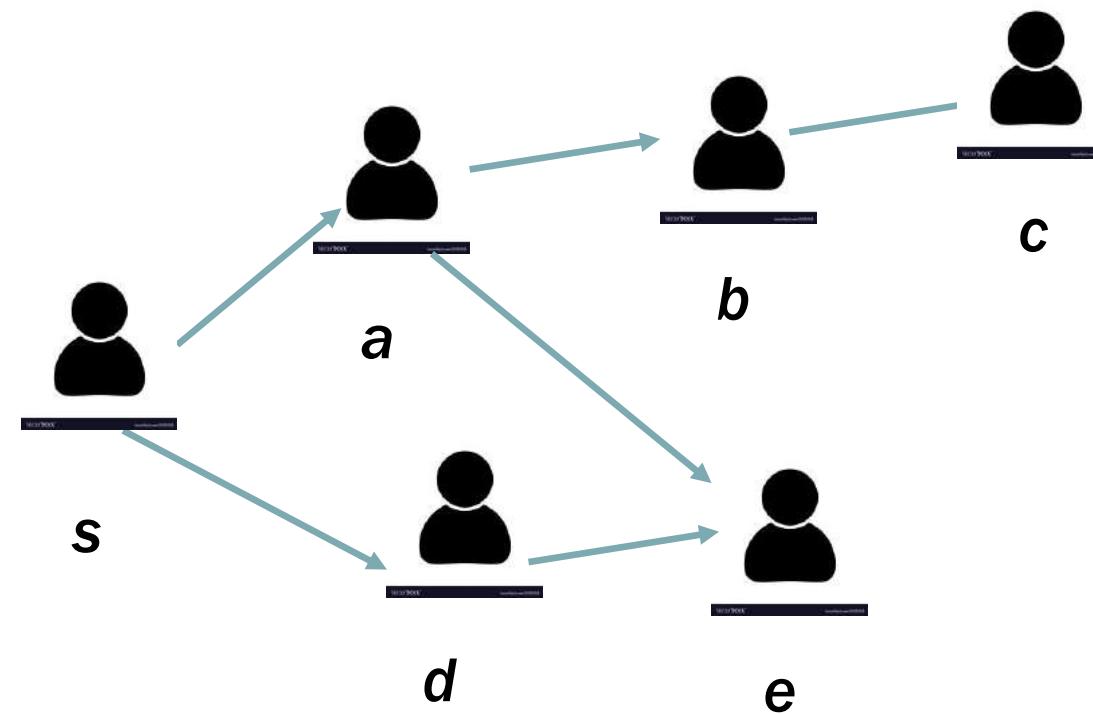
Data representation:

Each key-value pair stores the information of an edge

- Key: nodeID
- Value: nodeID

Example:

- (s, a)
- (s, d)
- (a, e)
- (d, e)
- (a, b)
- (b, c)



Each MapReduce iteration advances the “frontier” by one hop

- Subsequent iterations include more and more reachable nodes as frontier expands
- Multiple iterations are needed to explore the nodes reachable in K hops

How to preserve the graph structure?

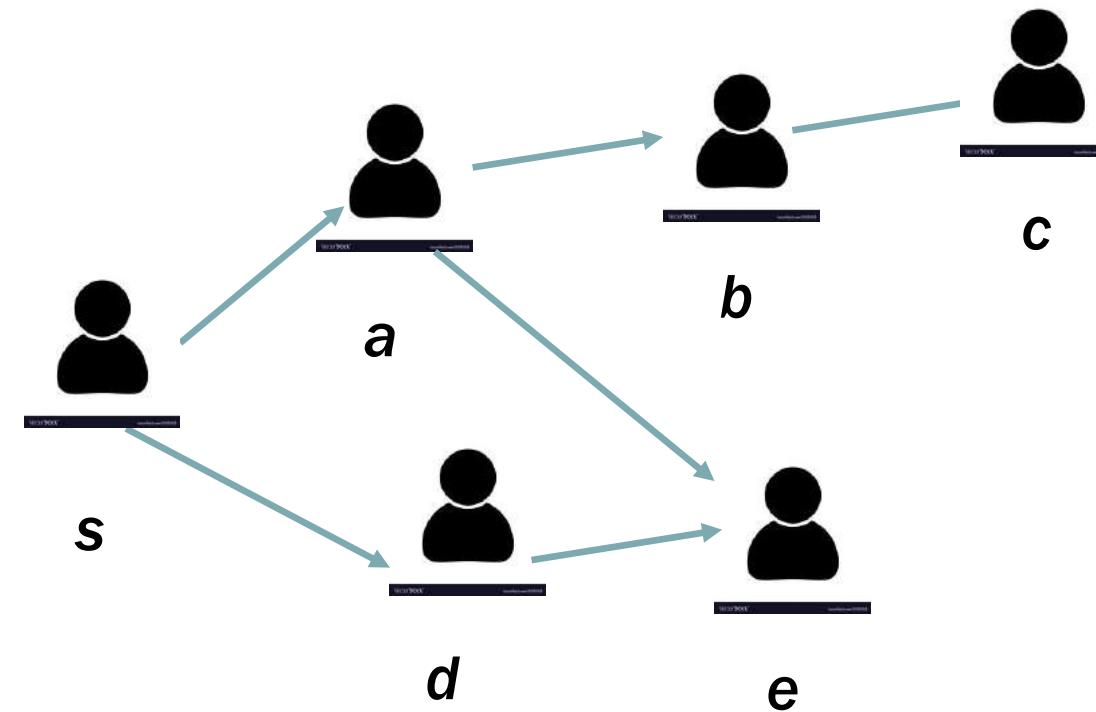
- Solution: map function emits (node, neighbor list) as well

FIRST MAPREDUCE JOB

Purpose:

Each key-value pair stores the information of an edge

- Key: nodeID
- Value: nodeID
- Example:
- (s, a)
- (s, d)
- (a, e)
- (d, e)
- (a, b)
- (b, c)



JOB0(STEP0):MAP

```
map(String nodeid_from, String nodeid_to) {  
    {  
        Emit-Intermediate(nodeid_from, nodeid_to);  
    }  
}
```

JOB0(STEP0):REDUCE

```
reduce(String nodeid, Iterator<String> neighbors) {  
    if(nodeid.equals("s")){  
        Emit("s", "D:0");//"D" indicates distance  
    }  
  
    Emit(nodeid, "N:"+ToString(neighbors));//"N" indicates  
neighbors  
}
```

Note: `ToString()` makes the list of neighbors as a string

EXAMPLE FOR JOBO

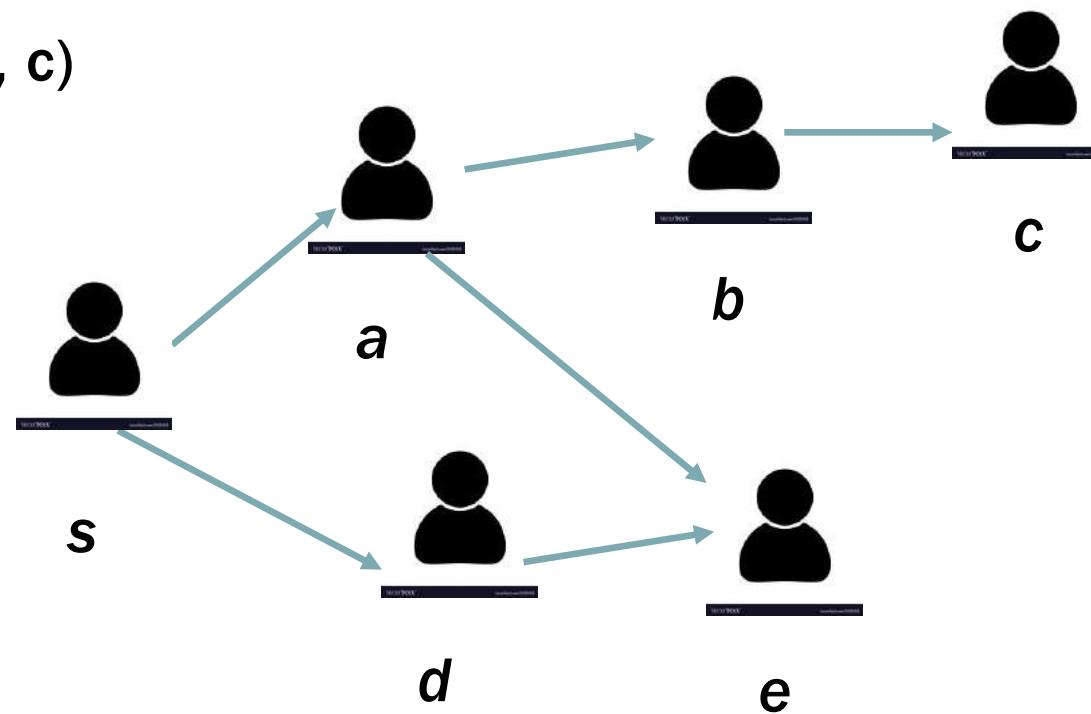
Job0:

Input

- (s, a) (s, d) (a, e) (d, e) (a, b) (b, c)

Output

- (s, "D:0")
- (s, "N:a;d")
- (a, "N:b;e")
- (d, "N:e")
- (b, "N:c")



SUBSEQUENT JOBS (STEPS 1-K): MAP

```
map(String nodeid, String value) {  
    {  
        Emit-Intermediate(nodeid, value);  
    }  
}
```

SUBSEQUENT JOBS (STEPS 1-K): REDUCE

```
reduce(String nodeid, Iterator<String> values) {  
    d_min=+∞;  
    Iterator<String> neighbors;  
    for( value in values ){  
        if (value starts with “N”){  
            neighbors= ToNeighbors(value);  
            Emit(nodeid, value); // always send neighbors out  
        }  
        else{  
            if( ToInteger(value)<d_min)  
                d_min=ToInteger(value);  
        }  
    }  
    if(d_min!=+∞){  
        for (neighbor in neighbors){  
            Emit(neighbor, ToString(“D:”,d_min+1));  
            // possible distances for neighbors  
        }  
        Emit(nodeid, ToString(“D:”,d_min));  
    }  
}
```



EXAMPLE FOR JOB1

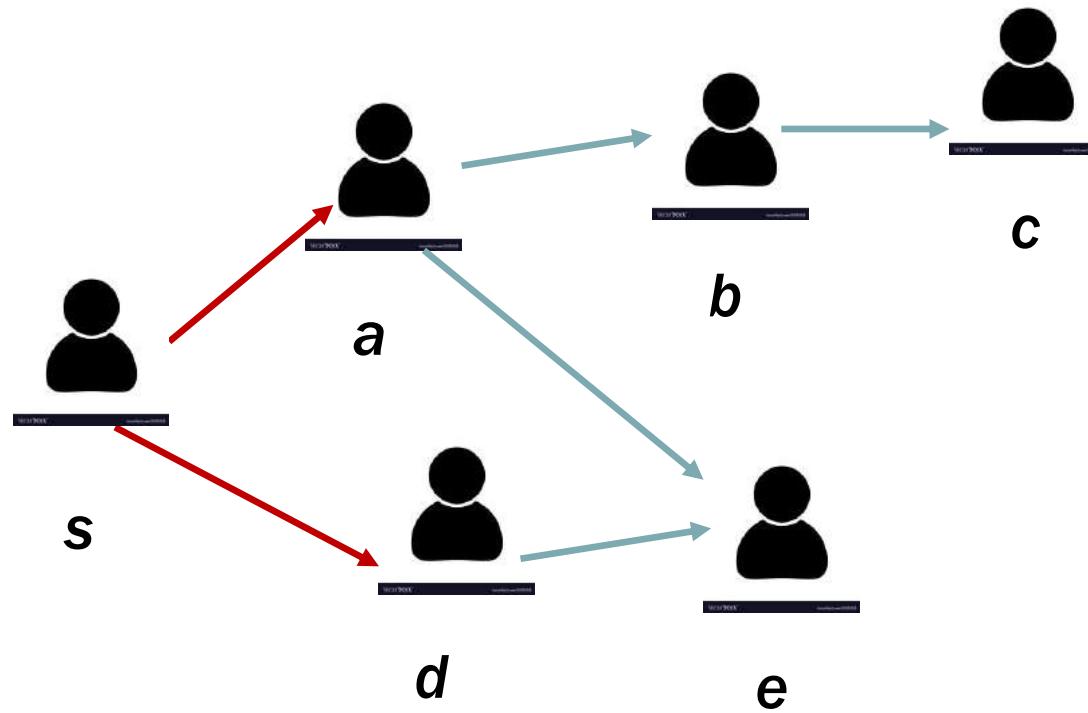
Job1:

Input

- (s,"D:0") (s, "N:a;d") (a, "N:b;e") (d, "N:e") (b, "N:c")

Output

- (s,"D:0")
- (s, "N:a;d")
- (a, "N:b;e")
- (d, "N:e")
- (b, "N:c")
- (a, "D:1")
- (d, "D:1")



EXAMPLE FOR JOB2

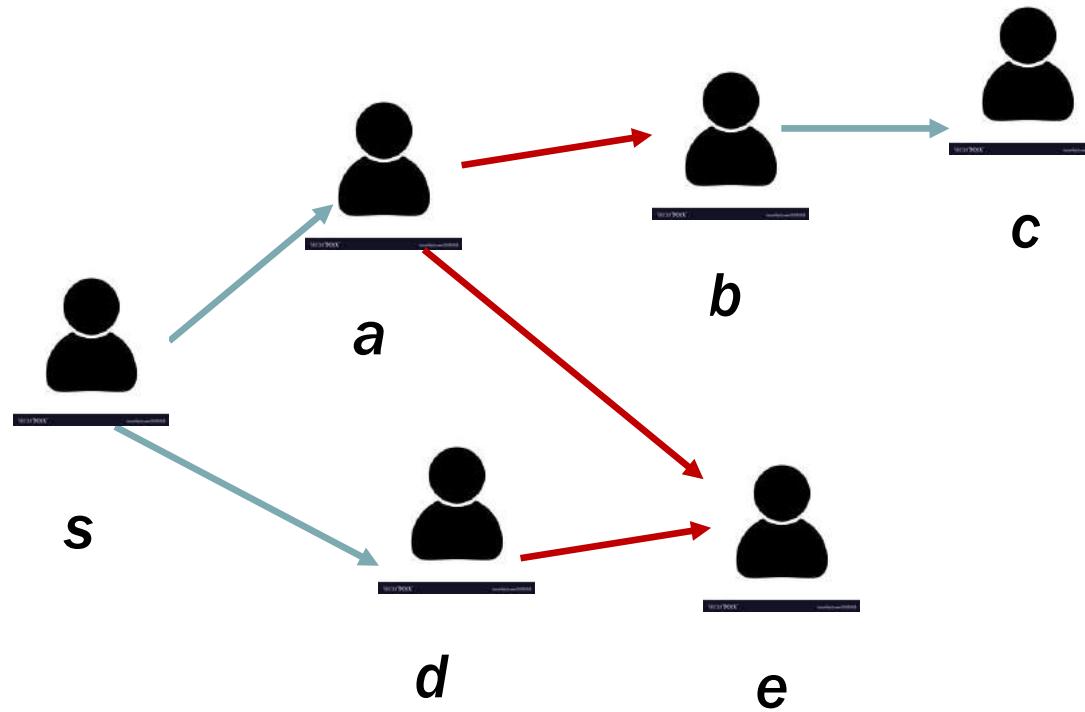
Job2:

Input

- (s,"D:0") (s, "N:a;d") (a, "N:b;e") (d, "N:e") (b, "N:c") (a, "D:1") (b, "D:2")

Output

- (s,"D:0")
- (s, "N:a;d")
- (a, "N:b;e")
- (d, "N:e")
- (b, "N:c")
- (a, "D:1")
- (d, "D:1")
- (b, "D:2")
- (e, "D:2")
- (e, "D:2")



LAST JOB: MAP

```
map(String nodeid, String value) {  
    {  
        Emit-Intermediate(nodeid, value);  
    }  
}
```

LAST JOB: REDUCE

```
reduce(String nodeid, Iterator<String> values) {  
    d_min=+∞;  
    for( value in values ){  
        if (value starts with “D”){  
            if( ToInteger(value)<d_min)  
                d_min=ToInteger(value);  
        }  
    }  
    if(d_min!=+∞)  
        Emit(nodeid, ToString(“D:”,d_min));  
}
```

EXAMPLE LAST JOB

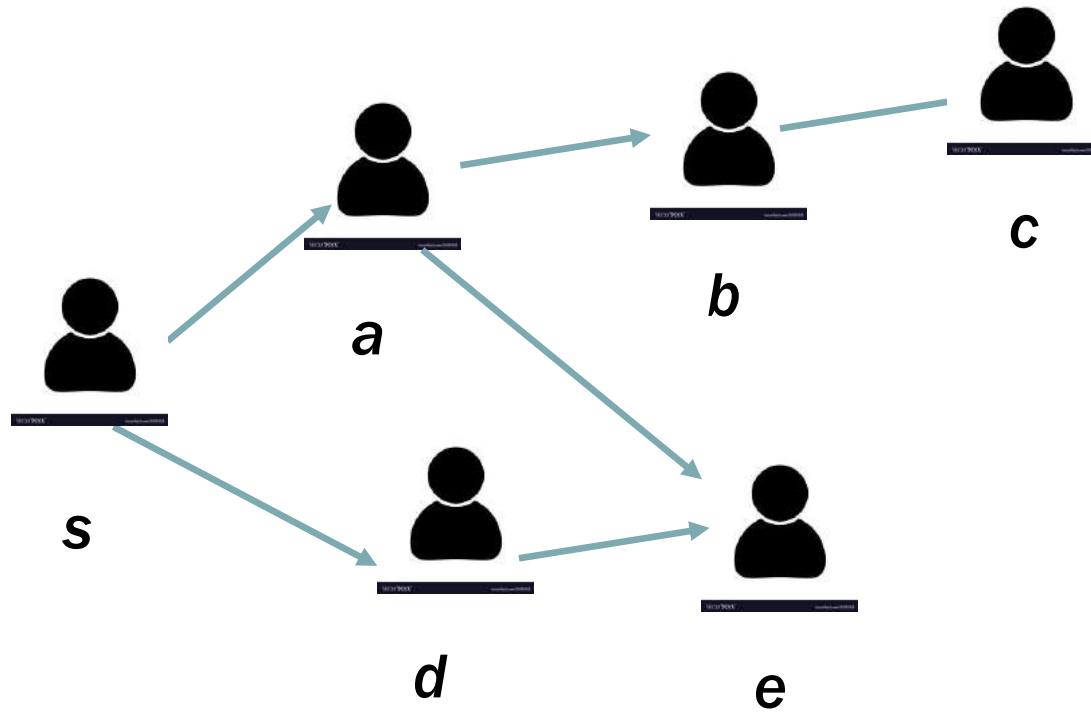
Job2:

Input

- (s, "D:0") (s, "N:a;d") (a, "N:b;e") (d, "N:e") (b, "N:c") (a, "D:1") (d, "D:1") (b, "D:2") (e, "D:2") (e, "D:2")

Output

- (s, "D:0")
- (a, "D:1")
- (d, "D:1")
- (b, "D:2")
- (e, "D:2")



OPTIMIZATION?



FURTHER EXTENSION

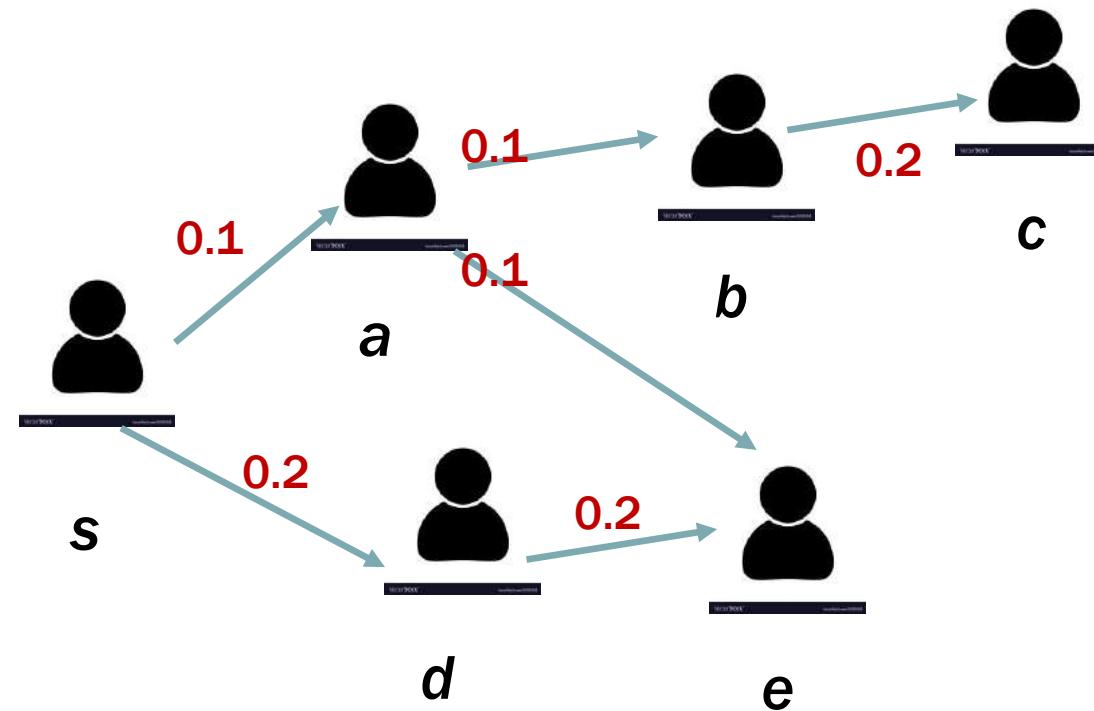
Previously, we considered the K -hop shortest path computation in social graphs using MapReduce. Extend the pseudo code to handle weighted graph whose edges may have different weights.

FIRST MAPREDUCE JOB

Purpose:

Each key-value pair stores the information of an edge

- Key: nodeID
- Value: nodeID; **weight**
- Example:
- (s, “a; **0.1**”)
- (s, “d; **0.2**”)
- (a, “e; **0.1**”)
- (d, “e; **0.2**”)
- (a, “b; **0.1**”)
- (b, “c; **0.2**”)



JOB0(STEP0):MAP

```
map(String nodeid_from, String nodeid_to_and_weight) {  
    {  
        Emit-Intermediate(nodeid_from, nodeid_to_and_weight);  
    }  
}
```

JOB0(STEP0):REDUCE

```
reduce(String nodeid, Iterator<String> neighbors) {  
    if(nodeid.equals("s")){  
        Emit("s", "D:0");//"D" indicates distance  
    }  
  
    Emit(nodeid, "N:"+ToString(neighbors));//"N" indicates neighbors  
}
```

Note: `ToString()` makes the list of neighbors as a string

EXAMPLE FOR JOB0

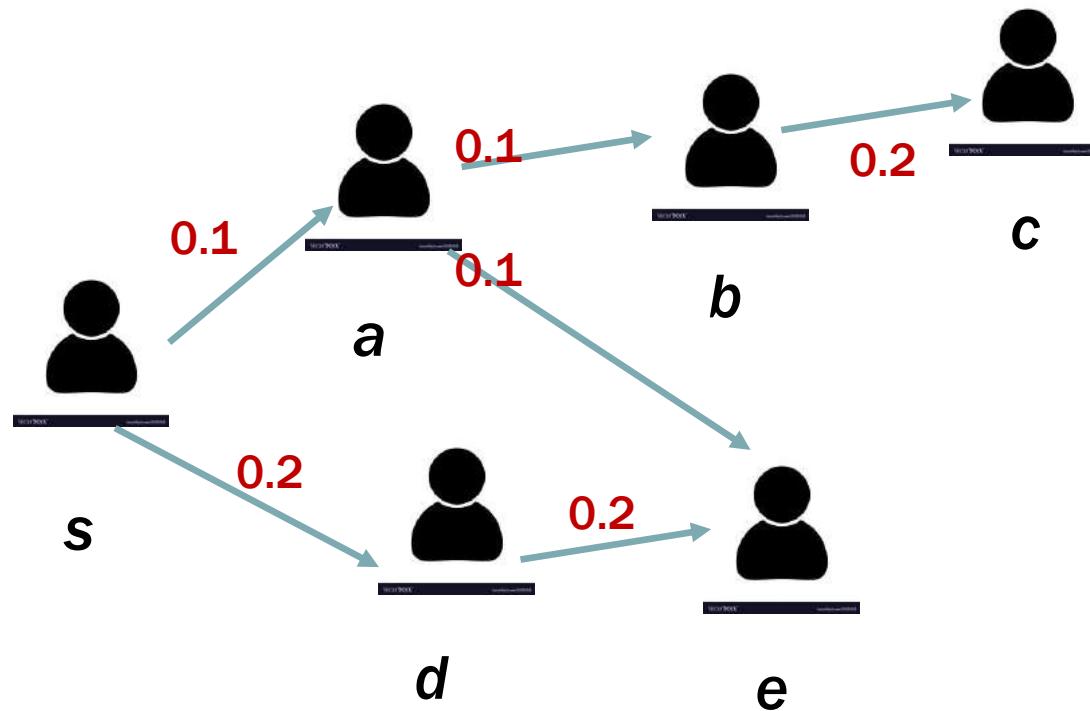
Job0:

Input

- (s, a;0.1) (s, d;0.2) (a, e;0.1) (d, e;0.2) (a, b;0.1) (b, c;0.2)

Output

- (s, "D:0")
- (s, "N:a;0.1;d;0.2")
- (a, "N:b;0.1;e;0.1")
- (d, "N:e;0.2")
- (b, "N:c;0.2")

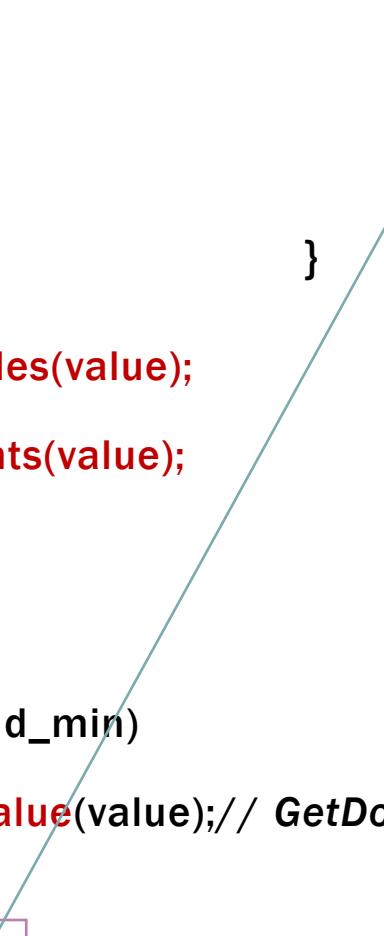


SUBSEQUENT JOBS (STEPS 1-K): MAP

```
map(String nodeid, String value) {  
    {  
        Emit-Intermediate(nodeid, value);  
    }  
}
```

SUBSEQUENT JOBS (STEPS 1-K): REDUCE

```
reduce(String nodeid, Iterator<String> values) {  
    d_min=+∞;  
    Iterator<String> neighbors;  
    Iterator<Double> weights;  
    for( value in values ){  
        if (value starts with "N"){  
            neighbors= ToNeighborsNodes(value);  
            weights= ToNeighborsWeights(value);  
            Emit(nodeid, value);  
        }else{  
            if( GetDoubleValue(value)<d_min)  
                d_min=GetDoubleValue(value); // GetDoubleValue removes "N" and convert the value to double.  
        }  
    }  
    if(d_min!=+∞){  
        for (int u=0;u<neighbors.size();u++){  
            Emit(neighbors[i], ToString("D:",d_min+weights[i]));  
        }  
        Emit(nodeid, ToString("D:",d_min));  
    }  
}
```



LAST JOB: MAP

```
map(String nodeid, String value) {  
    {  
        Emit-Intermediate(nodeid, value);  
    }  
}
```

LAST JOB: REDUCE

```
reduce(String nodeid, Iterator<String> values) {
```

```
    d_min=+∞;
```

```
    for( value in values ){
```

```
        if (value starts with “D”){
```

```
            if(GetDoubleValue(value)<d_min)
```

```
                d_min=GetDoubleValue(value);
```

```
        }
```

```
}
```

```
    if(d_min!=+∞)
```

```
        Emit(nodeid, ToString(“D:”,d_min));
```

```
}
```

EXAMPLE: PAGERANK

- Google's famous algorithms for ranking webpages.
- A measure of the “importance/quality” of a page.
- Widely adopted for ranking search results
- Intuition
 - Consider a random surfer that starts at a random webpage
 - He follows out-going links in a random manner with probability $(1 - \alpha)$
He jumps to a random webpage with probability α
 - PageRank is the probability that the random surfer will arrive at a given webpage

MORE FORMALLY

PageRank of a page is based on the PageRank of the pages which link to it

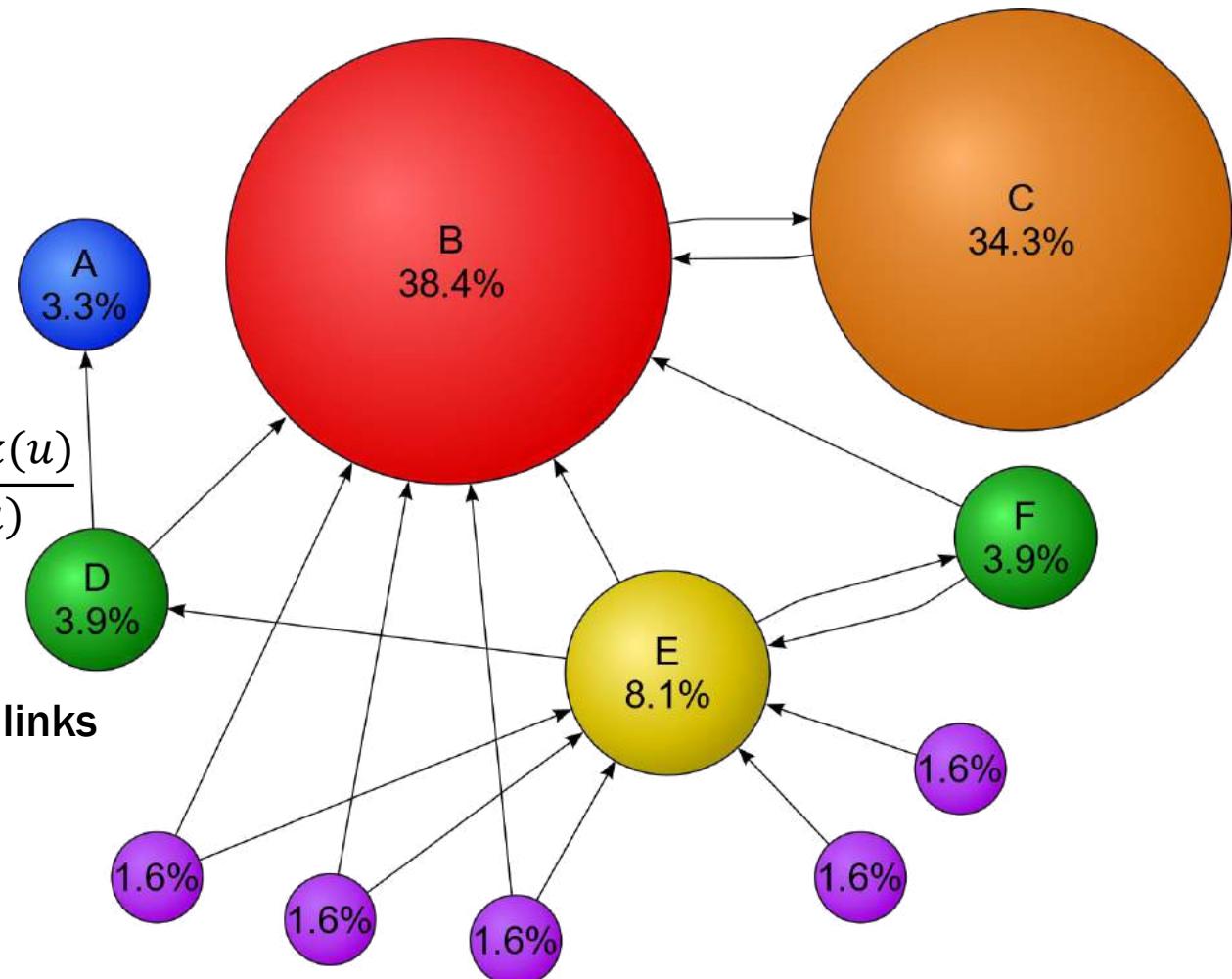
a value in $(0, 1)$

$$\text{PageRank}(v) = \frac{\alpha}{n} + (1 - \alpha) \times$$

$$\sum_{u \text{ links to } v} \frac{\text{PageRank}(u)}{d_{\text{out}}(u)}$$

Number of web pages

Number of out-going links



PAGERANK COMPUTATION

Step 1: Initialize $\text{PageRank}(v) = 1/n$ for every webpage.

Step 2: Iteratively apply the formula until convergence

For each webpage v , compute

$$\text{PageRank}(v) = \frac{\alpha}{n} + (1 - \alpha) \times \sum_{u \text{ links to } v} \frac{\text{PageRank}(u)}{d_{\text{out}}(u)}$$

Value of the previous iteration



MAPREDUCE DESIGNS

Each iteration is a MapReduce job:

Question:

Which features should be aggregated for a page?

$$\text{PageRank}(v) = \frac{\alpha}{n} + (1 - \alpha) \times \sum_{u \text{ links to } v} \frac{\text{PageRank}(u)}{d_{out}(u)}$$

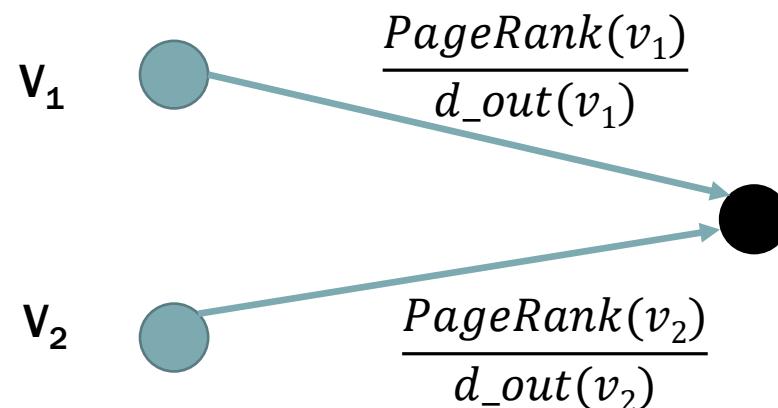
MAPREDUCE DESIGNS

Each iteration is a MapReduce job:

Map:

Input with the current PageRank value of a webpage v (or set to $\text{PageRank}(v)=1/n$ in the 1st job).

For each out-neighbor link (u) of current webpage v (namely, $v \rightarrow u$),
compute $\frac{\text{PageRank}(v)}{d_{out}(v)}$ and Emit-Intermediate($u, \frac{\text{PageRank}(v)}{d_{out}(v)}$)



Reduce:

Each webpage u receives the PageRank portion (stored in **values of reduce()**) from its incoming links, and hence it can compute

$$\sum_{u \text{ links to } v} \frac{\text{PageRank}(u)}{d_{out}(u)}$$

Use the formula to update PageRanks for the next job

MAP

```
Map(String nodeid, String value_and_neighbors){  
    // value stores neighbor list and PageRank of nodeid  
    double PR=getPageRank(value_and_neighbors);  
    List<String> neighbors = getNeighbors(value_and_neighbors);  
    for(each neighbor in neighbors){  
        Emit-Intermediate(neighbor, "P:"+PR/neighbors.size());  
    }  
    Emit-Intermediate(nodeid, "N:"+ToString(neighbors));  
}
```

REDUCE

```
Reduce(String nodeid, Iterator values){
```

```
    double PR=0.0;
```

```
    String neighbors;
```

```
    for(value in values){
```

```
        if(value starts with "P"){
```

```
            PR = PR + getPageRank(value) × (1 - α);
```

```
        }
```

```
        else{
```

```
            neighbors = getNeighbors(value);
```

```
        }
```

```
}
```

```
PR=PR+  $\frac{\alpha}{n}$  ;
```

```
Emit(nodeid, PR+";"+ neighbors);
```

```
}
```

SUMMARY AND OPEN DISCUSSIONS

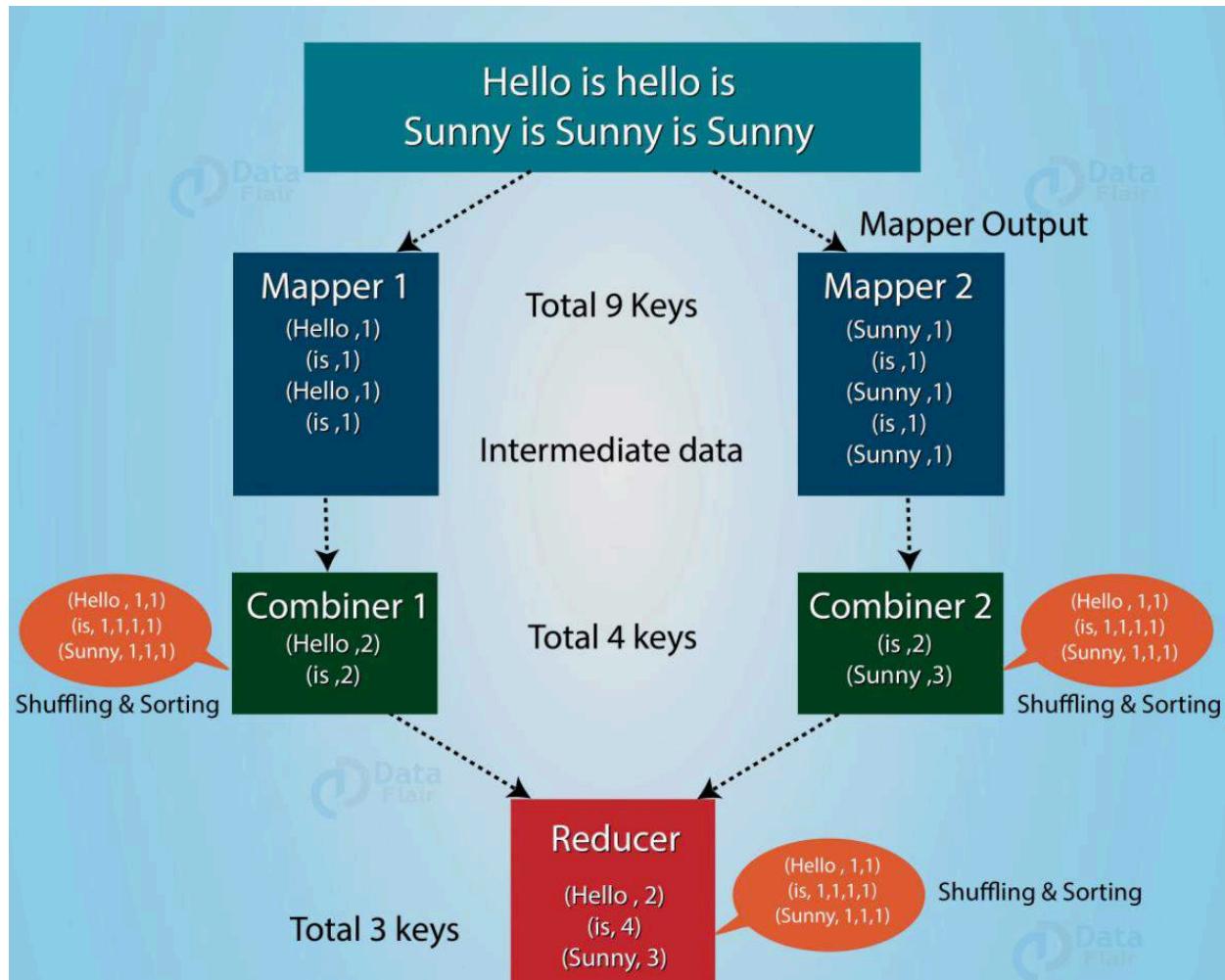
MapReduce model is very powerful.

- Database operations (join)
- Graph based queries (shortest paths, PageRanks)
- ...

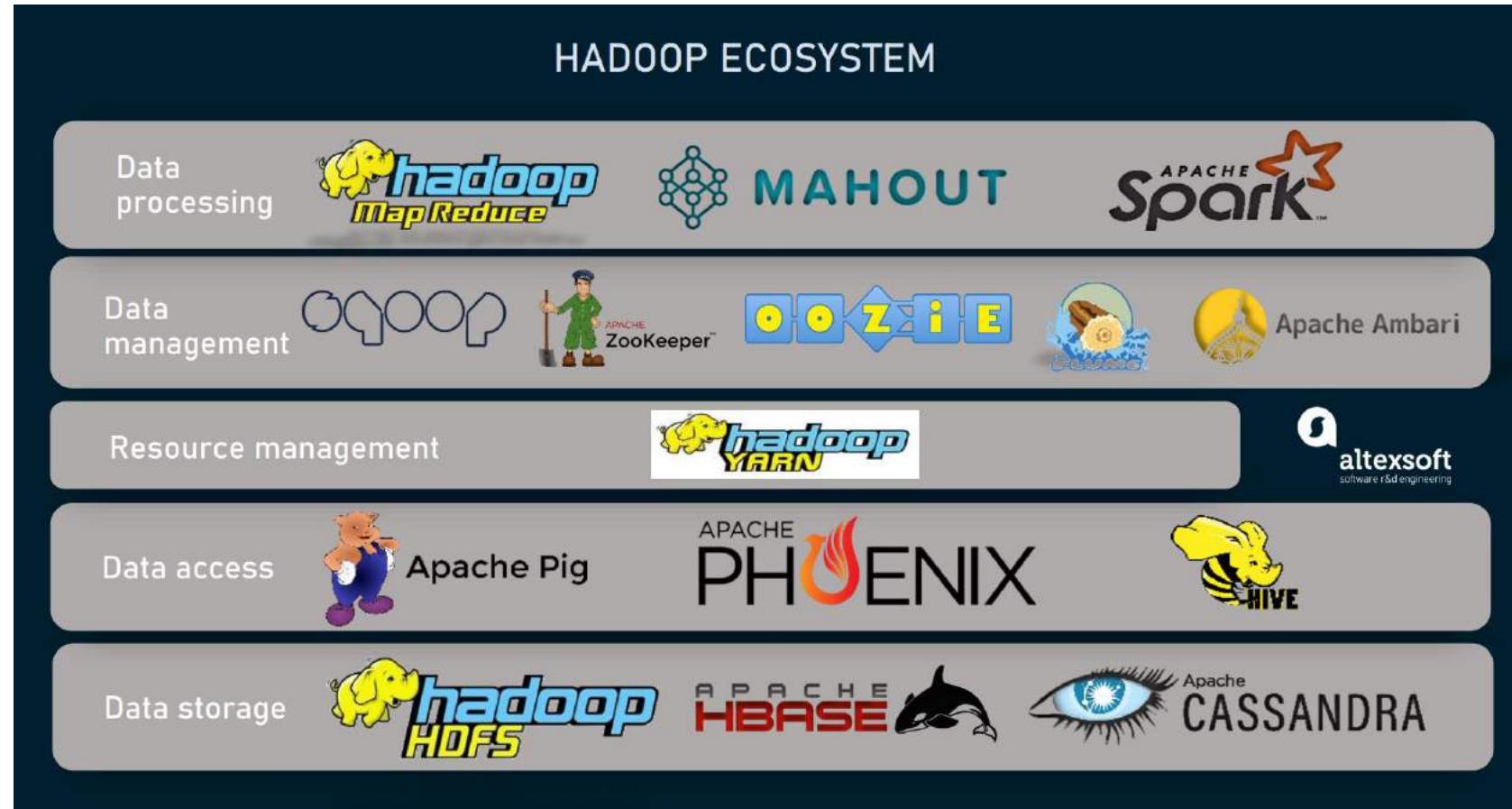
1. Any more examples?
2. Possible optimization



COMBINER



HADOOP → SPARK



Source: <https://www.altexsoft.com/blog/hadoop-vs-spark/>

We finish lectures for Distributed Systems!



Next lecture:

NoSQL and Key-Value Stores

BIG DATA MANAGEMENT

CE/CZ4123

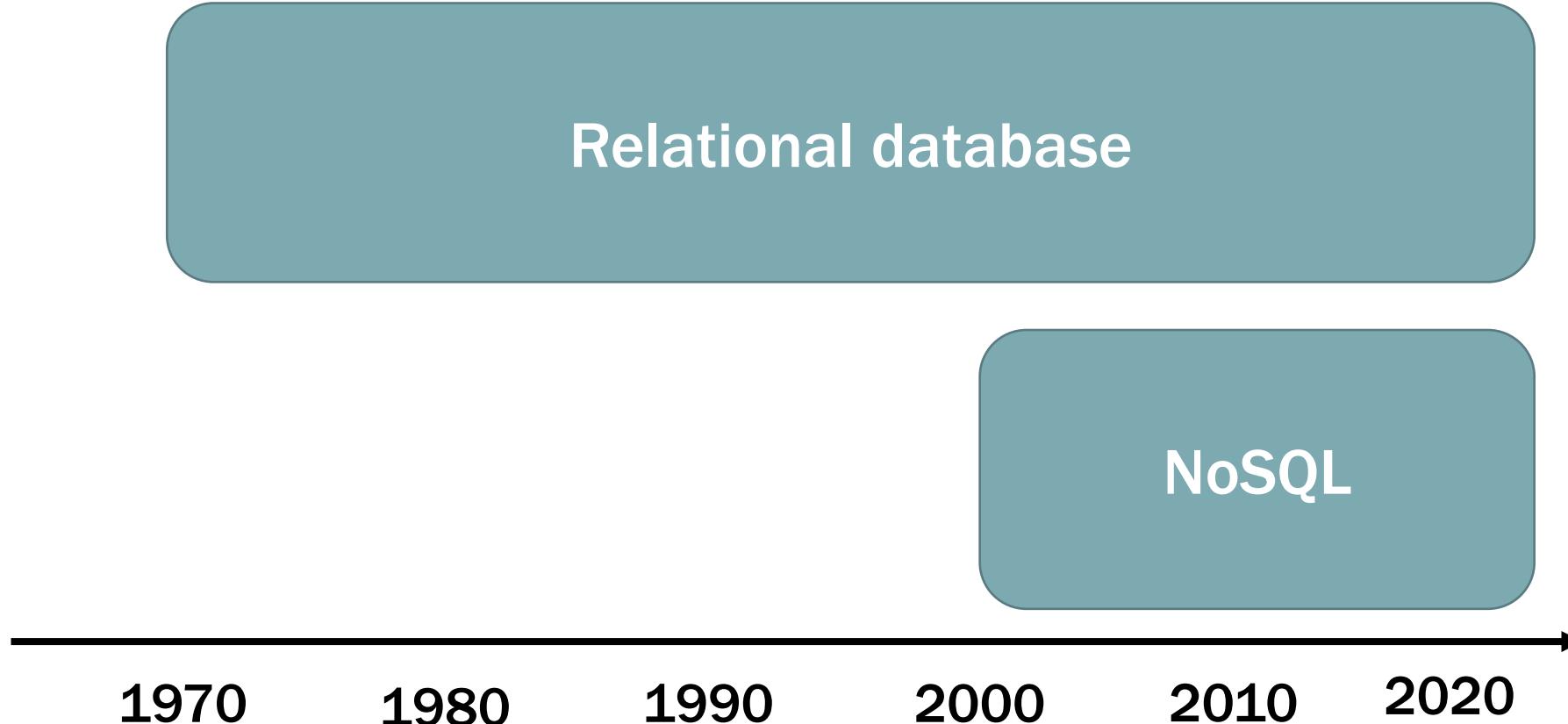
NOSQL

Siqiang Luo

Assistant Professor

BASIC CONCEPT

You may have heard about the word “NoSQL”



RELATIONAL DATABASES (RECAP)

Query patterns

- Selection, Projection, Join, Aggregation

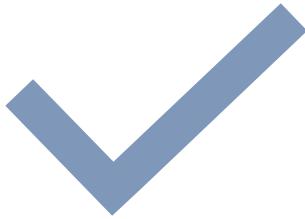
Query languages

- SQL (Structured Query Language)
- Relational algebra

Systems

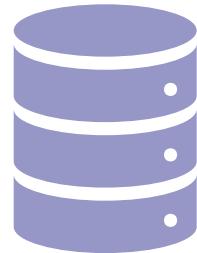
- Oracle Database, Microsoft SQL Server, IBM DB2, MySQL, PostgreSQL

RELATIONAL DATABASES (RECAP)



Normal Forms

Functional dependencies
1NF, 2NF, 3NF, BCNF



Purpose

Good: Remove data redundancy, prevent update anomalies
Bad: Data is divided into small pieces and so queries involve joining them (costly).

CURRENT TRENDS

Big data

- Volume, Variety, Velocity, ...
- Various data formats
- Strong consistency is no longer mission-critical

Extensive user base

- Population online, hours spent online, devices online
- Growing companies / web applications
 - Even millions of users within a few months

Cloud computing

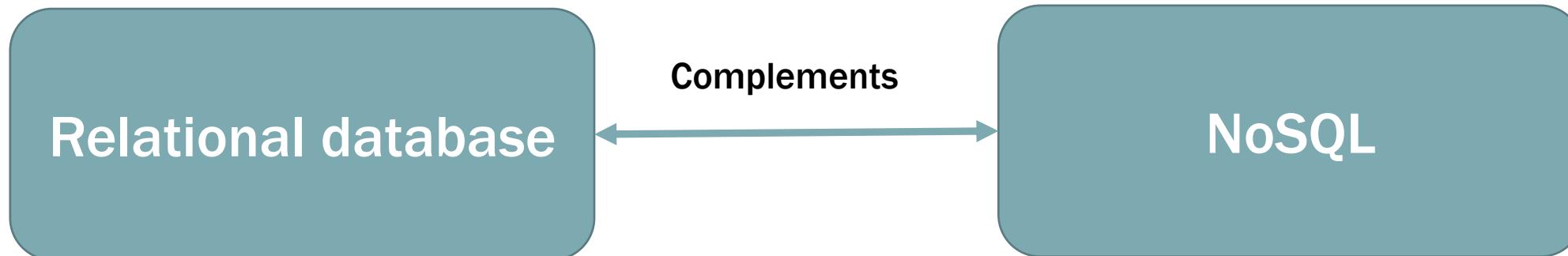
- On-demand services of data storage and computing power

Real-time analytic processing

- Quality of services becomes more and more important.

RELATIONAL AND NOSQL DATABASES

- ❑ NoSQL also means “Not only SQL”, where SQL refers to the relational database (not exactly the SQL language).
- ❑ A **NoSQL** database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.
- ❑ NoSQL databases are increasingly used in big data applications.



PROPERTIES OF NOSQL DATABASE

Well designed schema

e.g., Relations are well decomposed and connected by foreign keys

Benefit: standardized data model designs (normal forms)

Relational database

Flexible schema (schemaless)

e.g., documents, key-value pairs

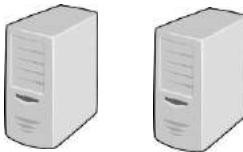
Benefit: programmers are more flexible in designing data models.

NoSQL

PROPERTIES OF NOSQL DATABASE

Not that easy to scale

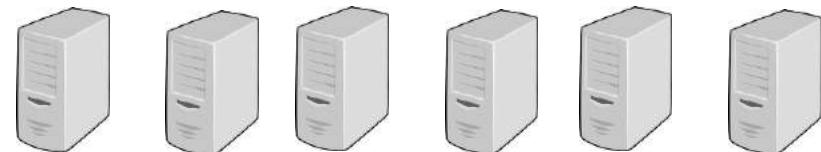
Reason: usually relational database
requires strict consistency



Relational database

Easier to scale

Reason: usually requires
eventual consistency

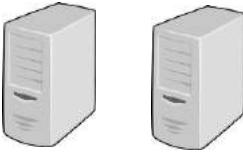


NoSQL

PROPERTIES OF NOSQL DATABASE

Better supports query languages

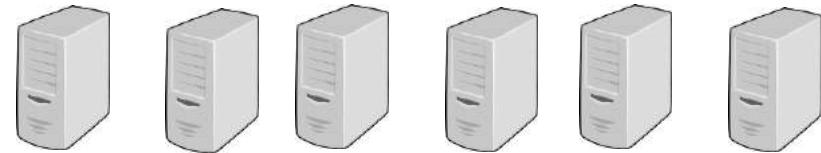
e.g., SQL



Relational database

Partially supports query language

Lack of standardized interface



NoSQL

PROPERTIES OF NOSQL DATABASE

Support all kinds of relational queries

Relational database

Queries are less flexible, but can have higher performance

NoSQL

TYPES OF NOSQL DATABASES



Key-Value Stores



Wide-Column Database



Document Database



Graph Database

KEY-VALUE STORE

- Data model
 - key-value pairs
 - The simplest NoSQL database type
 - Works as a simple hash table (mapping)
- Query patterns
 - Create, update or remove value for a given key
 - Get value for a given key
- Characteristics
 - Simple model → great performance, easily scaled, ...
 - Simple model → not for complex queries nor complex data

KEY-VALUE STORE

- ❑ Very fundamental database, and widely used in Web applications with billion users (e.g., Facebook, Amazon)
- ❑ Use cases: Session data, user profiles, user preferences, shopping carts, i.e., when values are only accessed via keys
- ❑ Representatives:
 - RocksDB, LevelDB, Redis, Memcache,...

WIDE-COLUMN DATABASE

Data model: 2-dimensional key-value models.

- The names and format of the columns can vary across rows, even within the same table.
- Columns are not separately stored, but some of them can be grouped as a “column family”. Given a column family, it can be stored row-by-row.
- It can have very large number of columns

Representatives:

- Google Bigtable, Apache Cassandra, Apache HBase, Apache Accumulo, Hypertable

DOCUMENT DATABASE

- Data model: data are stored as documents
 - A document describes an object
 - Can be described in JSON format (i.e., a list of key-value pairs to describe the object attributes)
 - Example: { "FirstName": "Bob", "Address": "5 Oak St.", "Hobby": "sailing" }
 - A document is addressed by a “key”, so also regarded as a subclass of key-value store. Differences is that the value is a document which can contain very rich information.

□ Representatives:

- MongoDB, Couchbase, Amazon DynamoDB, CouchDB, RethinkDB, RavenDB, Terrastore

GRAPH DATABASE

Data model: graphs

- A graph consists of nodes and edges
- Nodes represent entities; edges for relationships
- Easier to model data that contain many entities and interconnected relationships, e.g., social networks

Query patterns

- Create, update or remove a node / relationship in a graph
- Graph algorithms (shortest paths, spanning trees, ...)
- General graph traversals
- Sub-graph queries or super-graph queries
- Similarity based queries (approximate matching)

Representatives

- Neo4j, Titan, Apache Giraph, InfiniteGraph, FlockDB

WHEN TO CHOOSE ONE OVER THE OTHER (RDB VS NOSQL)

There is NOT a golden rule that you should choose one over the other. Both systems are improving themselves, and some modern systems have integrated their advantages as a mixed system in some sense.

However, there are some guidelines one can consider

- 1) When the data scale is small, relational database is great
- 2) When the project is always about relational queries and have various access patterns, relational database is great
- 3) When you want to better enforce field constraints, relational database is preferred.
- 4) When the data scale is very large, and you need very high performance for some specific type of queries, NoSQL is great.



SCENARIOS

Suppose a company with 10000 employees want to build an employee management system. It would require to query the salaries of employees, the managers of an employee etc.

Would you choose a relational database or NoSQL database?



SCENARIOS

Suppose a company with 10000 employees want to build an employee management system. It would require to query the salaries of employees, the managers of an employee etc.

Would you choose a relational database or NoSQL database?

Relational database is good for this scenario.

SCENARIOS

Suppose we aim to build a stock database that records all the stock price changes every 5 seconds. The query pattern is given a stock id, search the lowest and highest stock prices of a given time period. The results should be returned in a short time when searched.

Would you choose a relational database or NoSQL database?



SCENARIOS

Suppose we aim to build a stock database that records all the stock price changes every 5 seconds. The query pattern is given a stock id, search the lowest and highest stock prices of a given time period. The results should be returned in a short time when searched.

Would you choose a relational database or NoSQL database?

NoSQL key-value store is good choice for this scenario.

NOTE

When we emphasize the benefit of NoSQL systems, do NOT take for granted that NoSQL databases are always better than Relational databases.

Relational databases are great in many aspects

Both relational databases and NoSQL databases are improving

Both relational databases and NoSQL databases can be chosen for different applications.

ACKNOWLEDGEMENT

**Part of the slide contents are inspired by the course materials of
Prof. Martin Svoboda**

DISCUSSION

Debate between SQL and NoSQL

https://www.youtube.com/watch?v=rRoy6I4gKWU&list=RDCMUC_x5XG1OV2P6uZZ5FSM9Tw&start_radio=1&rv=rRoy6I4gKWU&t=774



BIG DATA MANAGEMENT

CE/CZ4123

KEY-VALUE STORE LSM-TREE BASICS

Siqiang Luo

Assistant Professor

PREPARATION

- ❑ In previous lectures, we introduced the basic concepts of NoSQL databases
 - ❑ Key-Value Store/Key-Value Database
 - ❑ Wide-column database
 - ❑ Document database
 - ❑ Graph database
- ❑ In the following lectures, we will introduce the mechanism of Key-Value Store, which is the most fundamental NoSQL database

BASIC CONCEPT

□ Data model

- Key-Value Store stores the data in key-value format
- Every key corresponds to a value

□ Functions: It supports four functions

- Get: Given a key, search the value indexed by the key
- Range-Get: Given a key range, search all the values indexed by any key within the range
- Put a new key-value pair
- Delete a key-value pair

□ Data engine: The log-structured merge tree (LSM-tree)

GET AND RANGE-GET

- Suppose a key-value store has the data $\{(1, \text{value1}), (2, \text{value2}), (3, \text{value3})\}$
 - **Get(1)** \rightarrow value1
 - **Get(2)** \rightarrow value2
 - **Get(5)** \rightarrow {}

- **Range-Get([2,3])** \rightarrow {value2, value3}
- **Range-Get([3,5])** \rightarrow {value3}
- **Range-Get([4,5])** \rightarrow {}

PUT AND DELETE

- Suppose a key-value store has the data $\{(1, \text{value1}), (2, \text{value2}), (3, \text{value3})\}$, then **logically**
 - After **Put(4,value4)**, we have $\{(1, \text{value1}), (2, \text{value2}), (3, \text{value3}), (4, \text{value4})\}$ in the key-value store
 - After **Delete(1)**, we have $\{(2, \text{value2}), (3, \text{value3}), (4, \text{value4})\}$ in the key-value store
 - After **Put(3, value5)**, we have $\{(2, \text{value2}), (3, \text{value5}), (4, \text{value4})\}$ in the key-value store



THINK...

Suppose we need to store 10 billion key-value pairs in the database, what data structure you will use?

FIRST IDEA



An array with ten billion entries

Size: $10^{10} * 100$ Bytes=1000GB

Stored in memory? Stored in disk?

FIRST IDEA



An array with ten billion entries

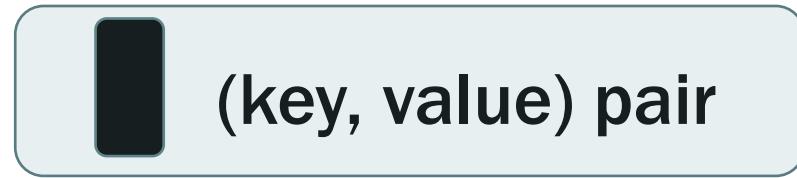
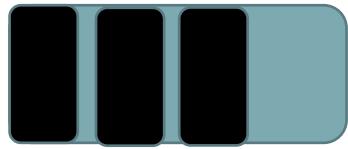
Too large to be stored in main memory

Main memory does not guarantee persistency

Disk is too slow to access, put and delete operations are costly.

SECOND IDEA

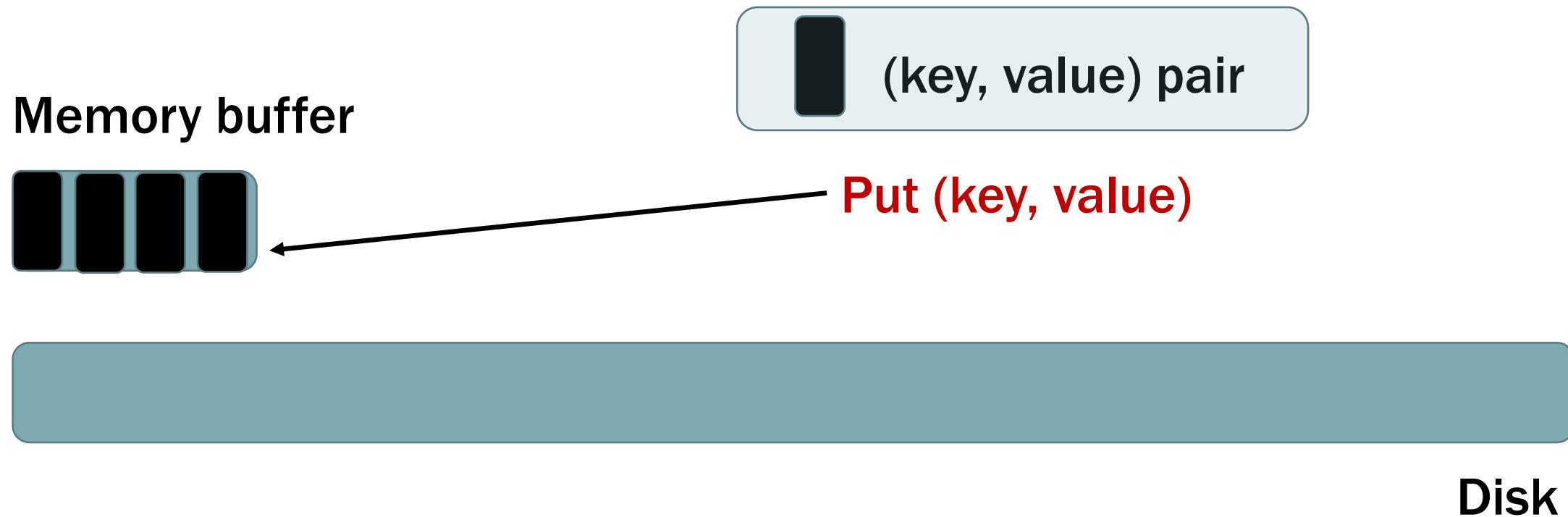
Memory buffer



Disk

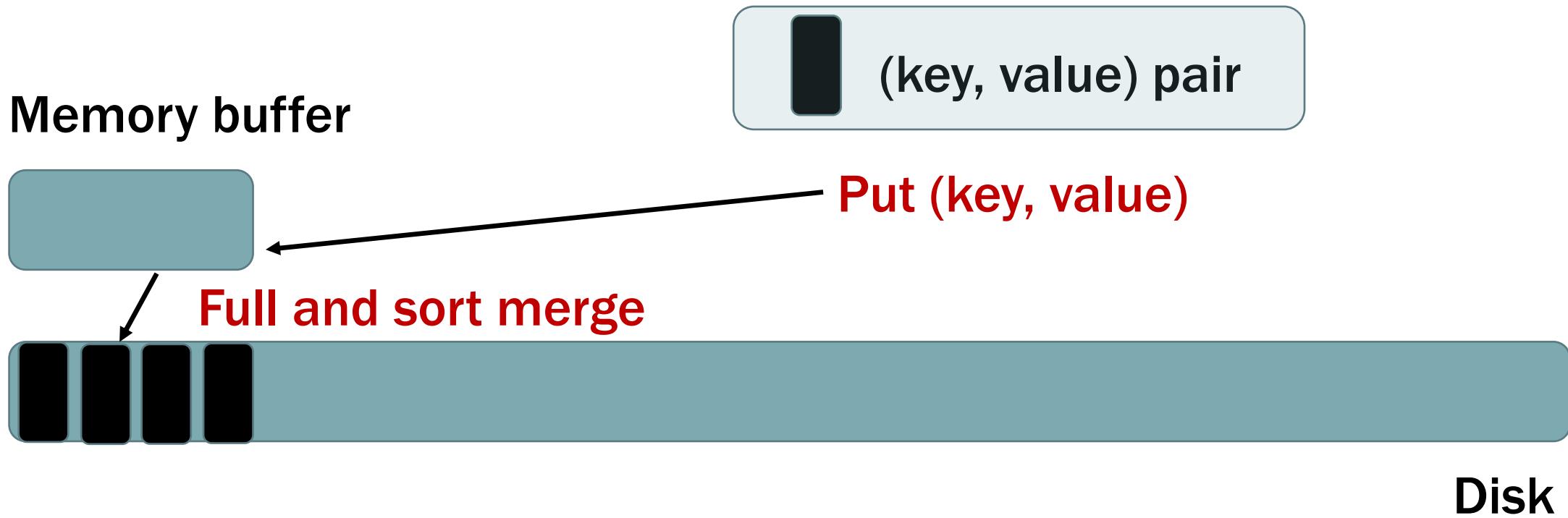
When handling Put(key,value), first insert the key-value pair into the main memory; when memory buffer is full, put all the buffer as a “run” into the disk.

SECOND IDEA



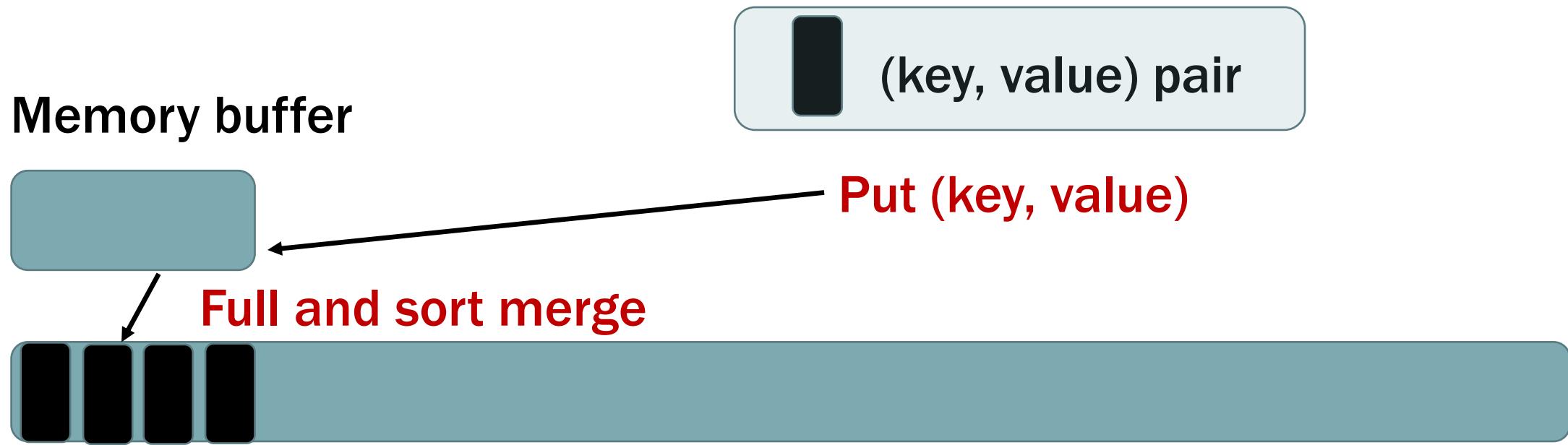
When handling **Put(key,value)**, first insert the key-value pair into the main memory; when memory buffer is full, put all the buffer as a “**sorted run**” into the disk.

SECOND IDEA



When handling Put(key,value), first insert the key-value pair into the main memory; when memory buffer is full, put all the buffer as a “sorted run” into the disk.

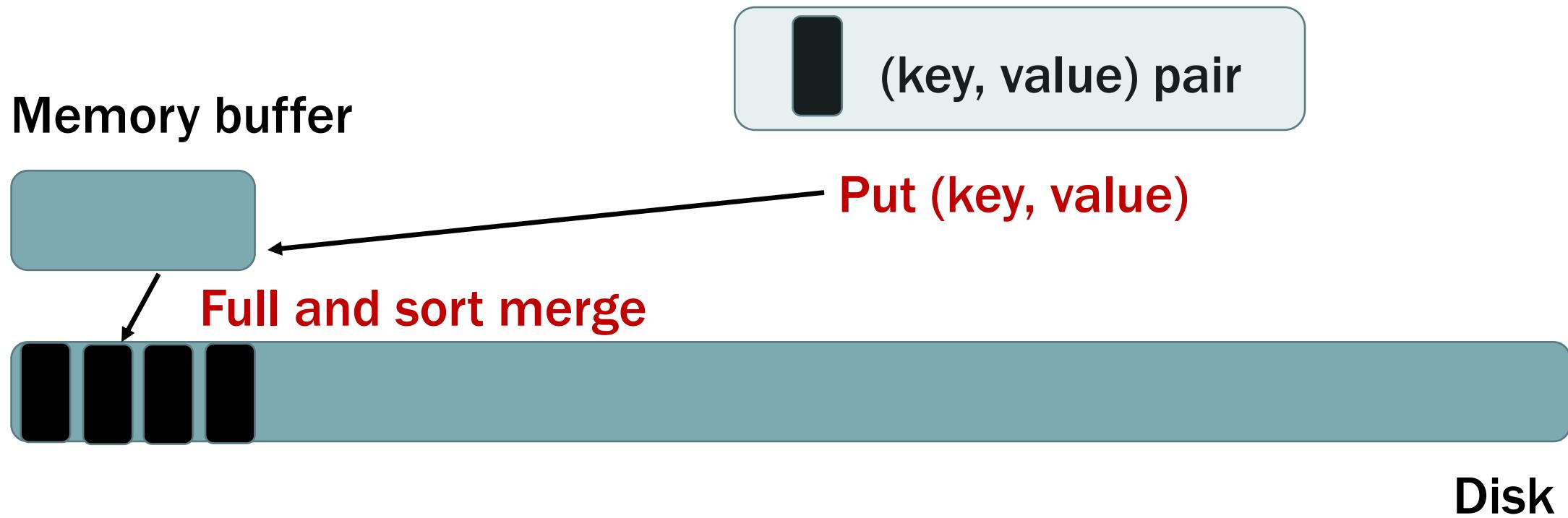
SECOND IDEA



Any problems with this design?



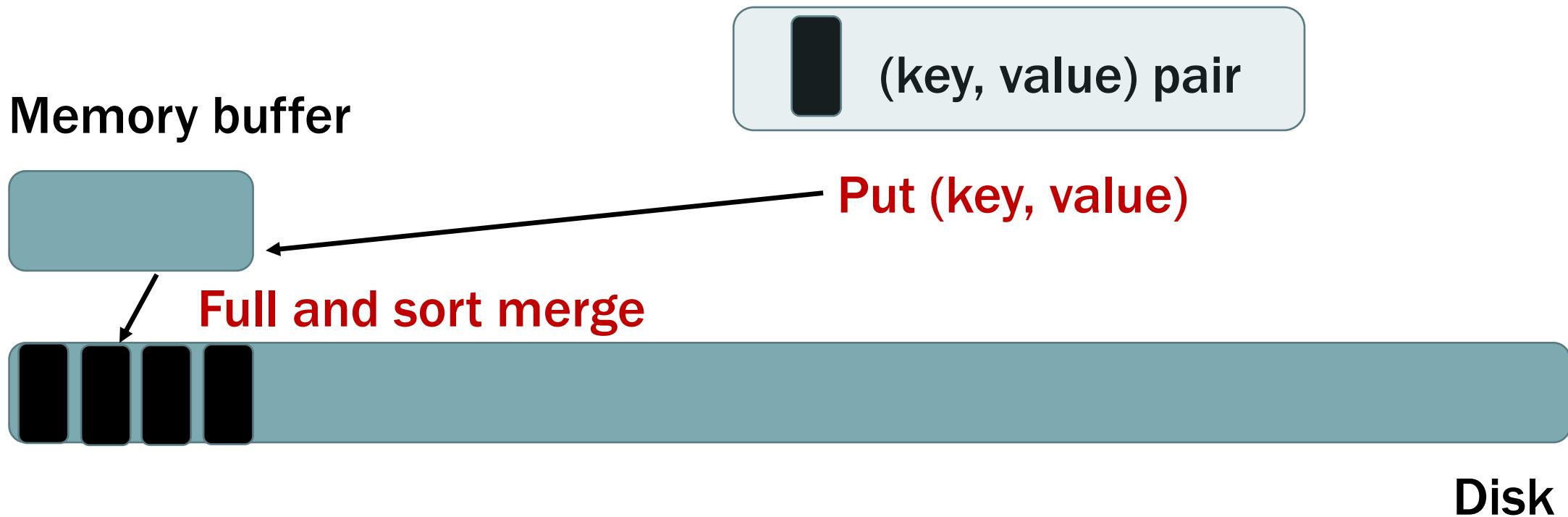
SECOND IDEA



Any problems with this design?

High cost for GET function!

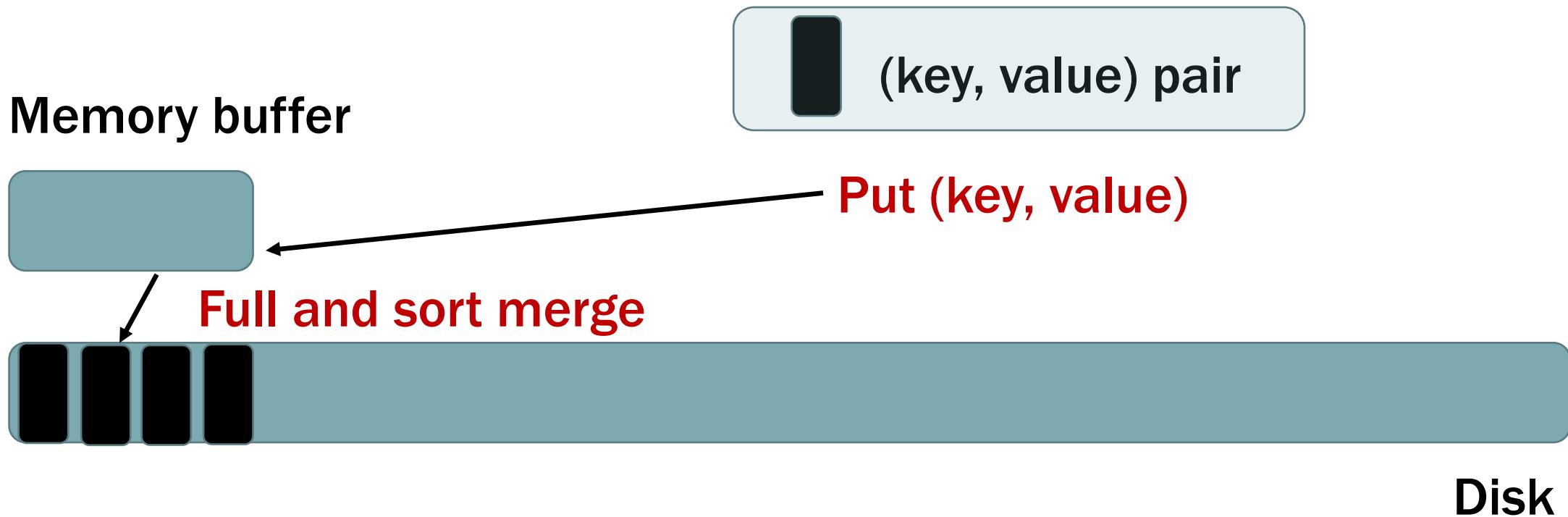
SECOND IDEA



GET(key) consists of two steps

- 1) Search the key in the main memory buffer; if the key exists in the buffer, directly return the value;
- 2) Search the key in the disk.

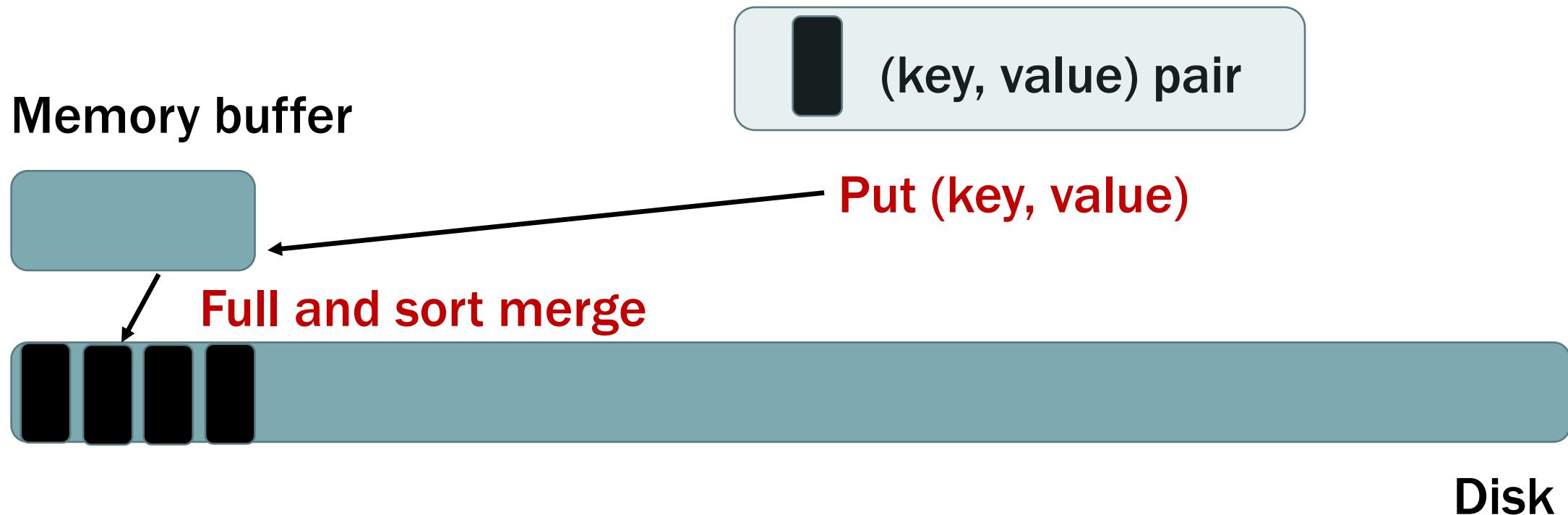
SECOND IDEA



GET(key) consists of two steps

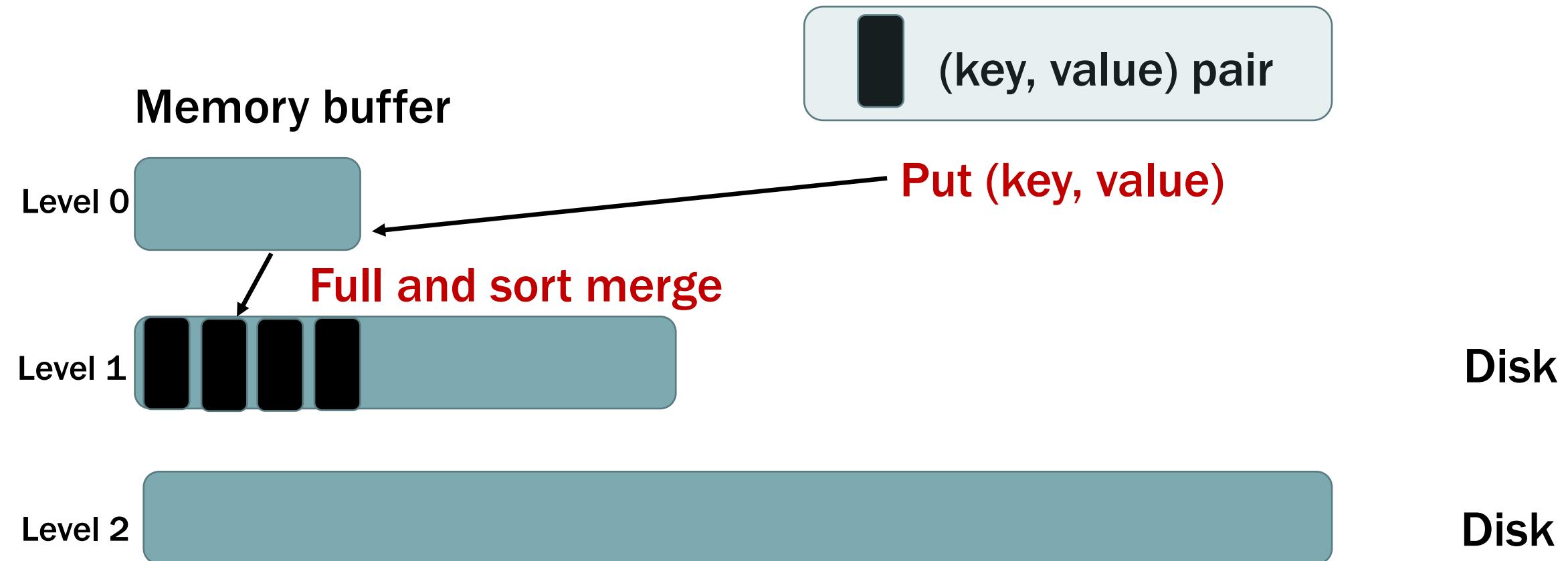
- 1) Search the key in the main memory buffer; if the key exists in the buffer, directly return the value;
- 2) Search the key in the disk. ← Very costly

SECOND IDEA



Put(key, value) is also **costly**

THIRD IDEA: BASIC LSM-TREES

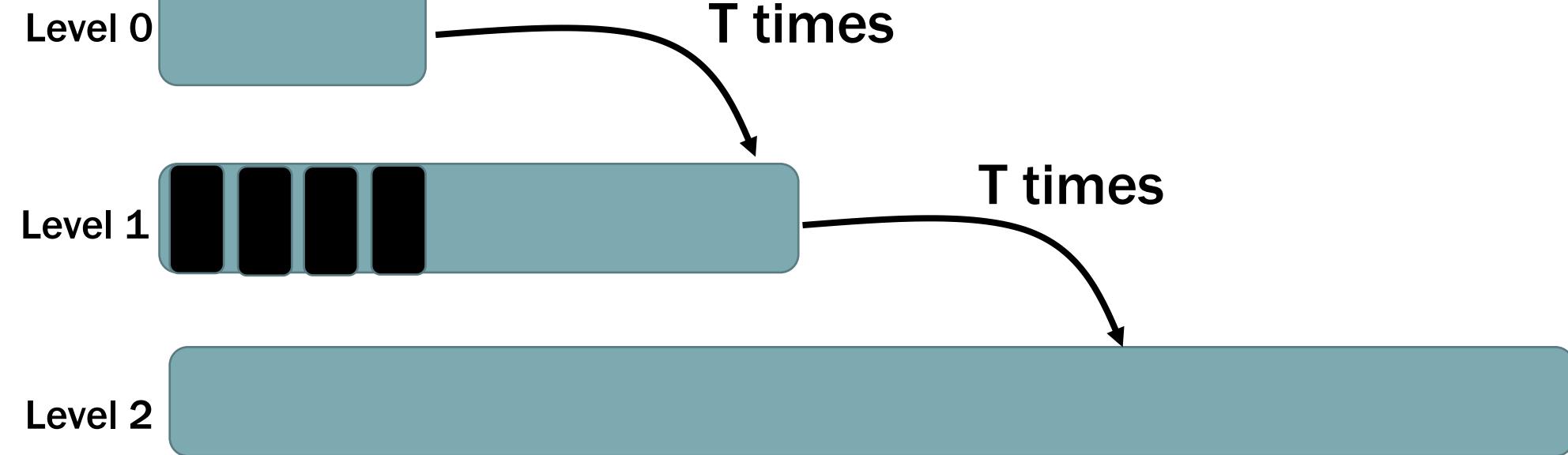


Two levels to multiple levels

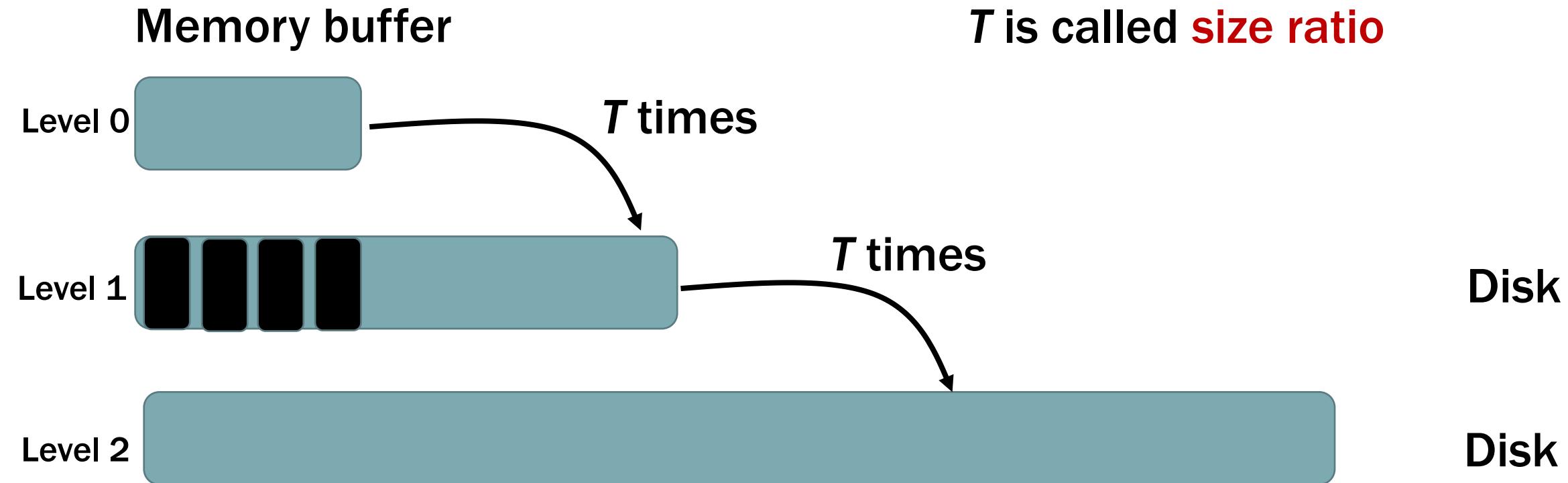
BASIC LSM-TREES

Memory buffer

T is called **size ratio**



BASIC LSM-TREES



Let $L[i]$ be the i -th level

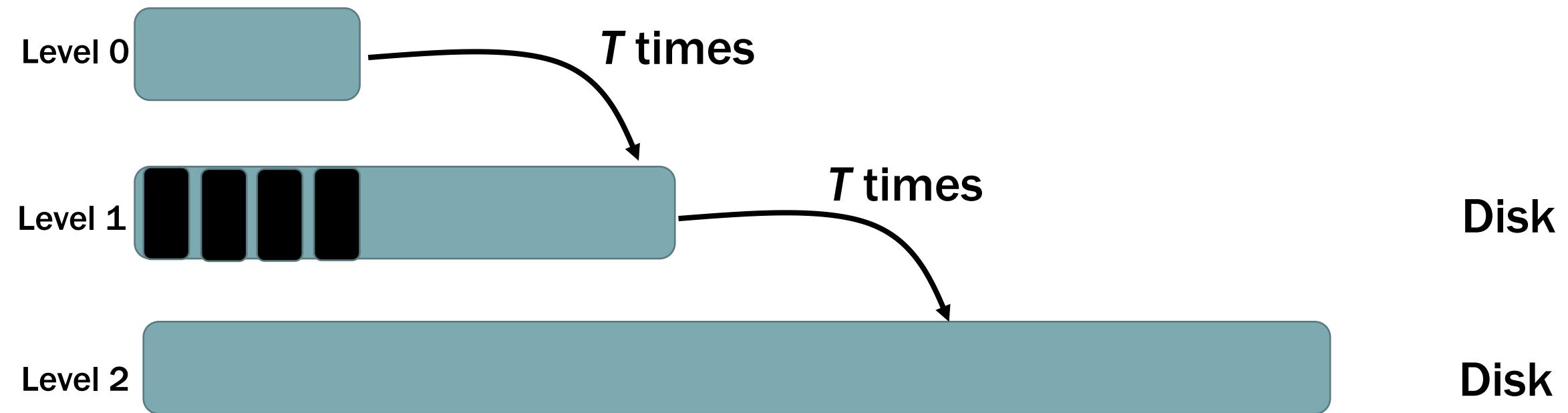
$$L[i] \text{ capacity} = S \times T^i$$

Mem buffer capacity (i.e., $L[0]$ capacity) = S (key-value pairs)

BASIC LSM-TREES

Memory buffer

T is called **size ratio**



Capacity is different from the actual size. $L[i]$ capacity may be different from the number of actually stored (key,value) pairs in $L[i]$

$$L[i] \text{ capacity} = S \times T^i$$

BASIC LSM-TREES

Memory buffer

Level 0



Level 1



Level 2



```
void Put(Key k, Value v) void SortMerge(Level i)
{
    L[0].insert(k,v);
    if(L[0].size==S){
        SortMerge(0);
    }
}
if (L[i+1] not exists)
    create L[i+1];
L[i+1].SortMergeWith(L[i]);
L[i].clear();
if(L[i+1]>S*(Ti+1-Ti)
    SortMerge(i+1);
}
```

BASIC LSM-TREES

Memory buffer

Level 0



Level 1



Level 2



```
void Put(Key k, Value v) { L[0].insert(k,v); if(L[0].size==S){ SortMerge(0); } }
```

```
void SortMerge(Level i) { if (L[i+1] not exists) create L[i+1]; L[i+1].SortMergeWith(L[i]); L[i].clear(); if(L[i+1]>S*(Ti+1-Ti)) SortMerge(i+1); }
```



BASIC LSM-TREES

Memory buffer

Level 0



Level 1



Level 2



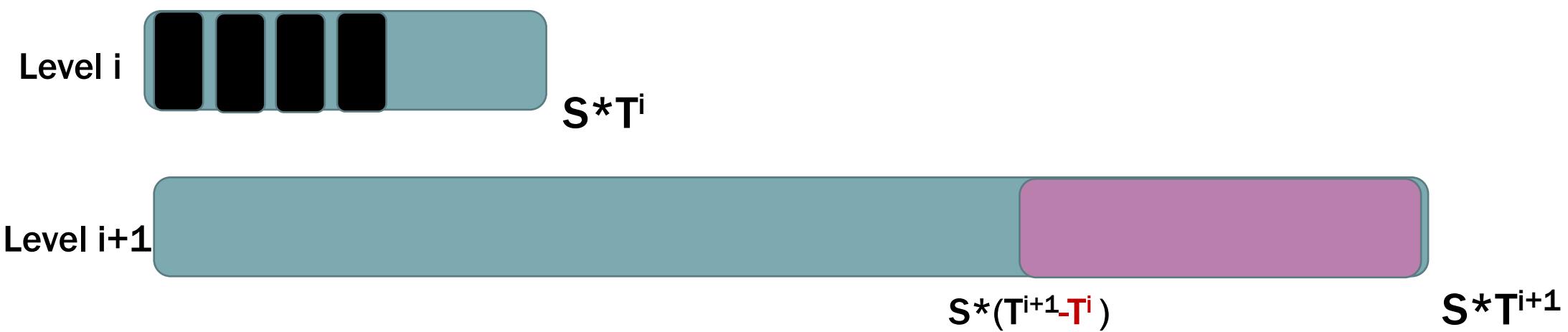
```
void Put(Key k, Value v) void SortMerge(Level i)
{
    L[0].insert(k,v);
    if(L[0].size==S){
        SortMerge(0);
    }
}
```

if ($L[i+1]$ not exists)
create $L[i+1]$;
 $L[i+1].SortMergeWith(L[i]);$
 $L[i].clear();$
if($L[i+1] > S * (T^{i+1} - T^i)$
 SortMerge($i+1$);

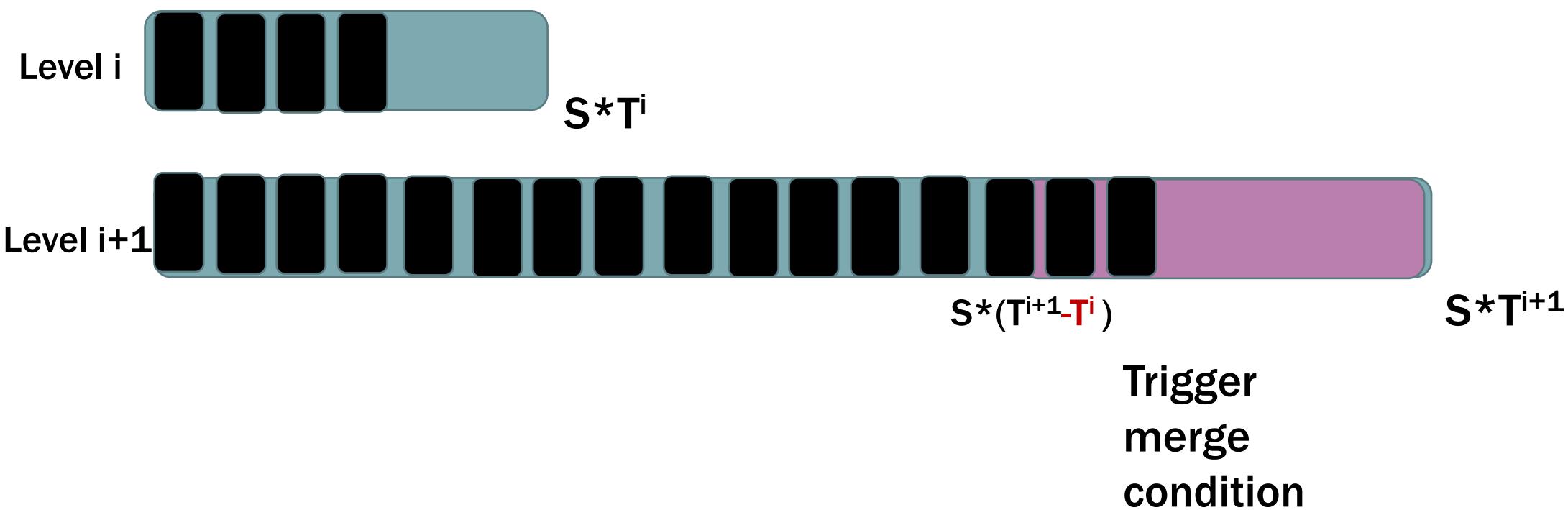
The actual number of (key,value) pairs after merging $L[i]$ into $L[i+1]$ may be less than actual size($L[i]$)+actual size($L[i+1]$)



MERGE CONDITION



MERGE CONDITION



EXAMPLE

Memory buffer

Level 0



Level 1



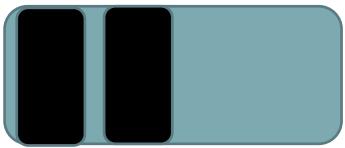
Level 2



EXAMPLE

Memory buffer

Level 0



Level 1



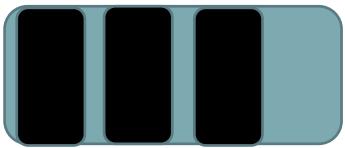
Level 2



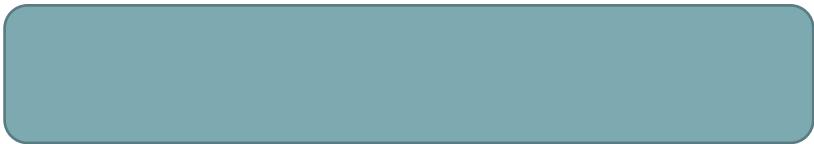
EXAMPLE

Memory buffer

Level 0



Level 1



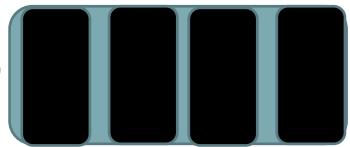
Level 2



EXAMPLE

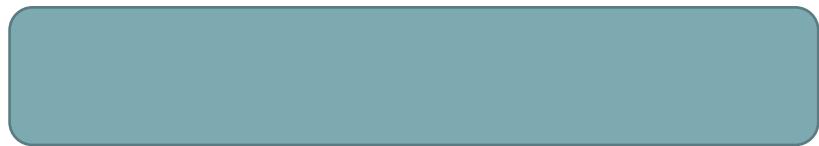
Memory buffer

Level 0



Full

Level 1



Level 2



EXAMPLE

Memory buffer



EXAMPLE

Memory buffer

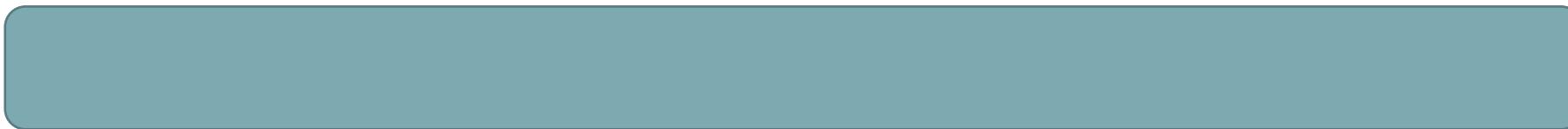
Level 0



Level 1



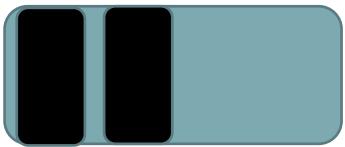
Level 2



EXAMPLE

Memory buffer

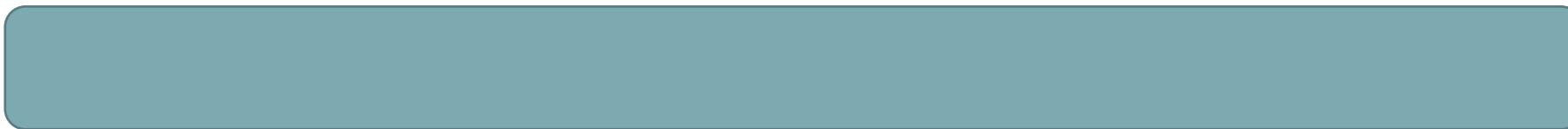
Level 0



Level 1



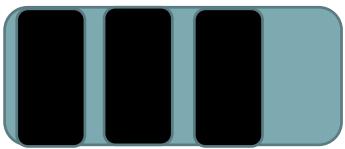
Level 2



EXAMPLE

Memory buffer

Level 0



Level 1



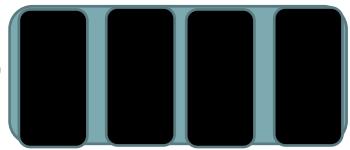
Level 2



EXAMPLE

Memory buffer

Level 0



Full

Level 1

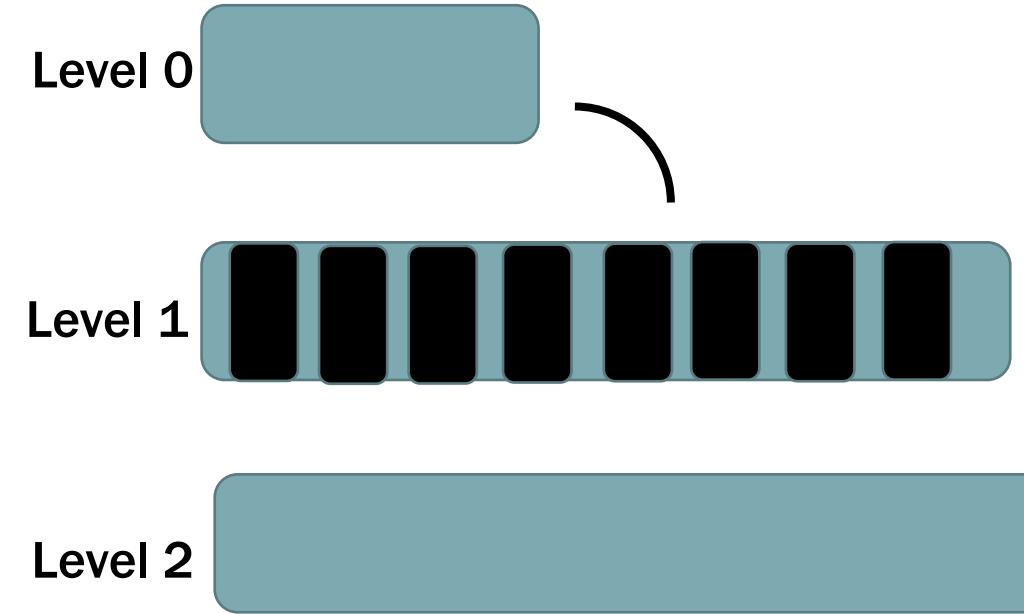


Level 2



EXAMPLE

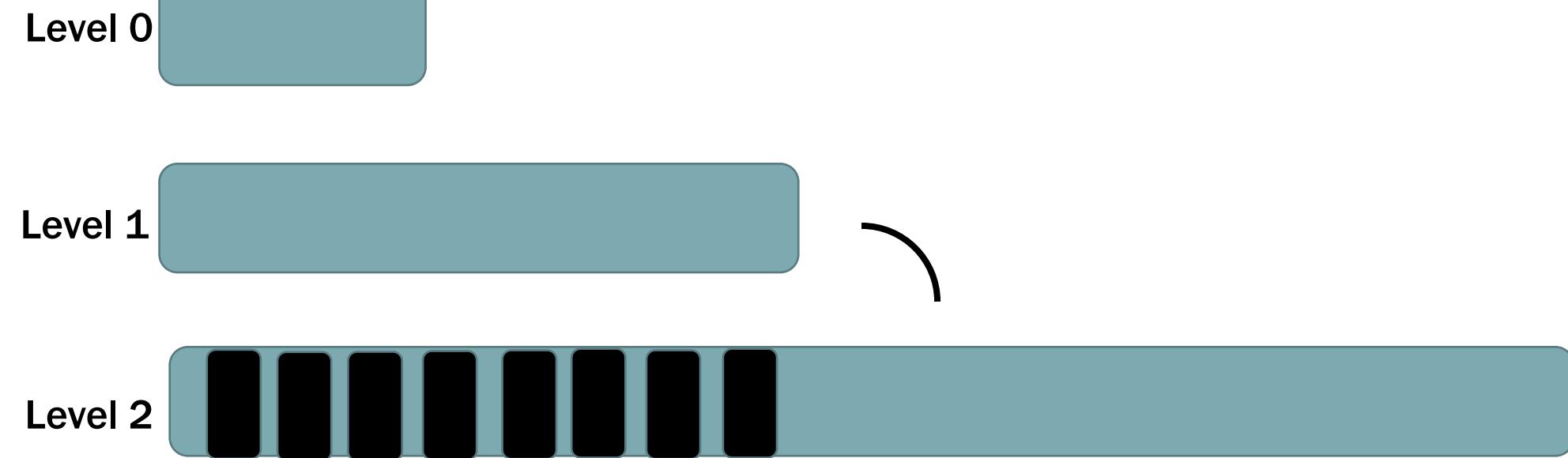
Memory buffer



Trigger merge condition

EXAMPLE

Memory buffer

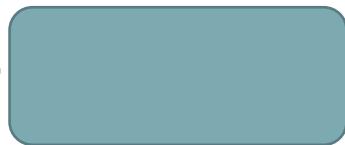


How about deleting a key?

DELETE?

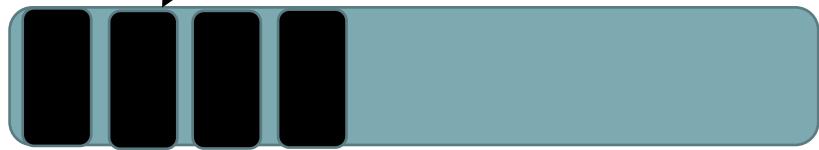
Memory buffer

Level 0



Full and sort merge

Level 1



Naïve way: directly find the key in each level and delete them...



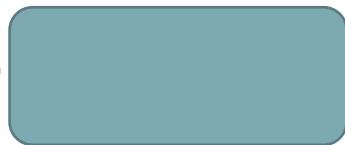
Level 2



DELETE?

Memory buffer

Level 0

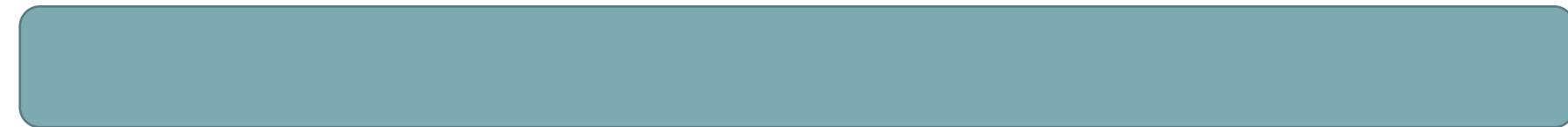


Full and sort merge

Level 1



Level 2



Naïve way: directly find the key in each level and delete them...

Too costly!!



DELETE--TOMBSTONE

Delete 6

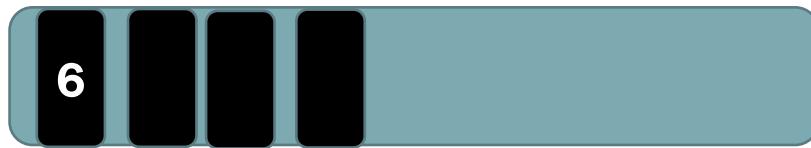
Level 0



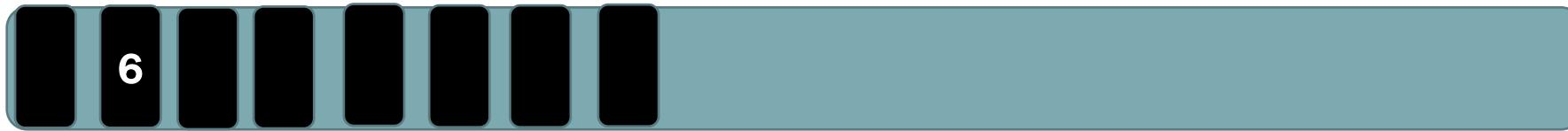
Deletes in LSM-trees are realized *logically* by inserting a special type of key-value entry, known as a tombstone.



Level 1



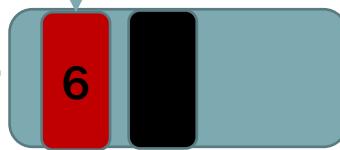
Level 2



DELETE--TOMBSTONE

Delete 6

Level 0



Deletes in LSM-trees are realized *logically* by inserting a special type of key-value entry, known as a tombstone.

Level 1



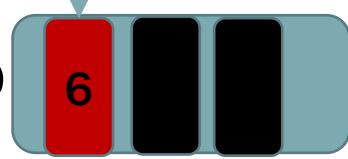
Level 2



DELETE--TOMBSTONE

Delete 6

Level 0

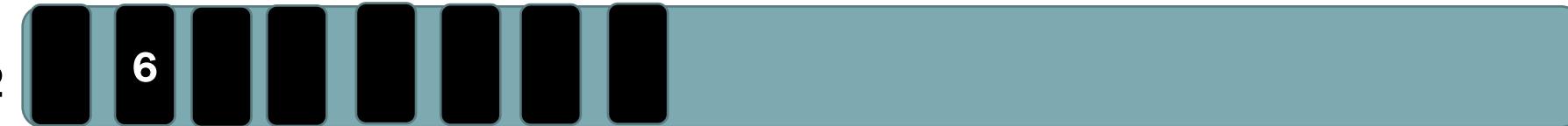


Deletes in LSM-trees are realized *logically* by inserting a special type of key-value entry, known as a tombstone.

Level 1



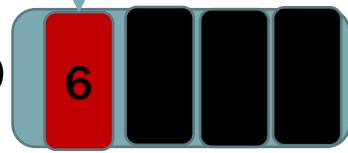
Level 2



DELETE--TOMBSTONE

Delete 6

Level 0

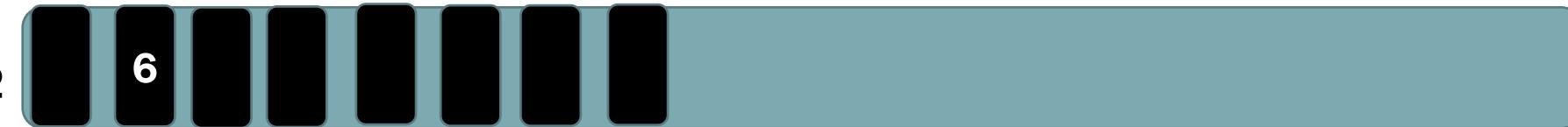


Deletes in LSM-trees are realized *logically* by inserting a special type of key-value entry, known as a tombstone.

Level 1

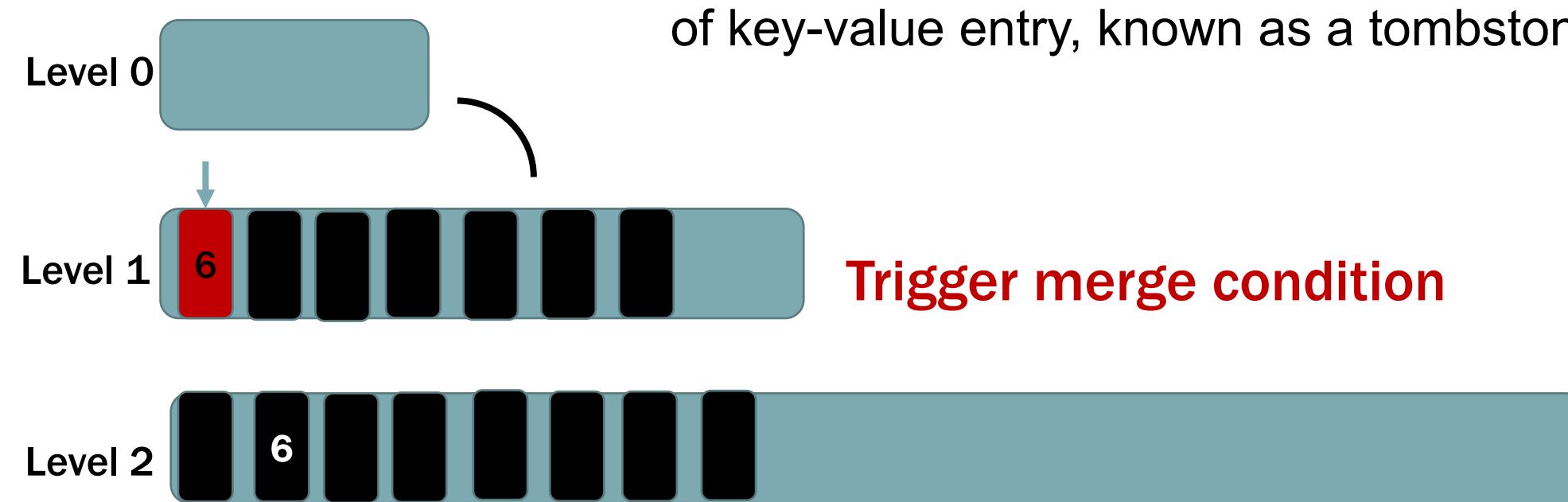


Level 2



DELETE--TOMBSTONE

Delete 6



Deletes in LSM-trees are realized *logically* by inserting a special type of key-value entry, known as a tombstone.

DELETE--TOMBSTONE

Delete 6

Level 0



Deletes in LSM-trees are realized *logically* by inserting a special type of key-value entry, known as a tombstone.

Level 1



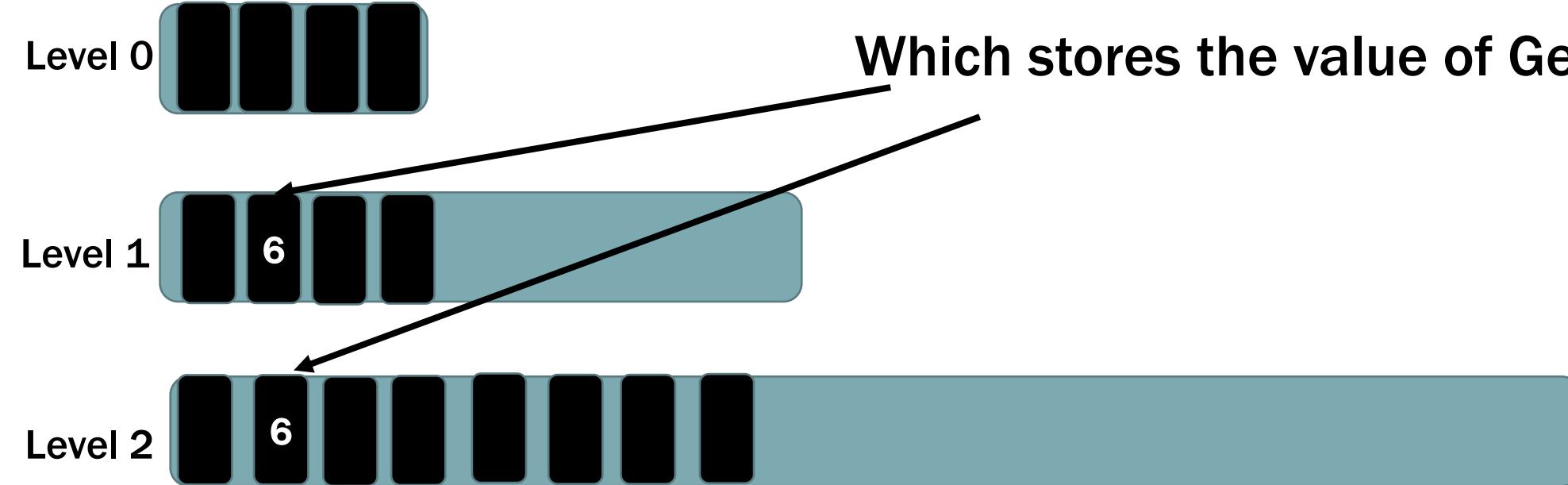
Level 2



In the last level, the tombstone will be deleted
“Red 6” and “black 6” cancel out

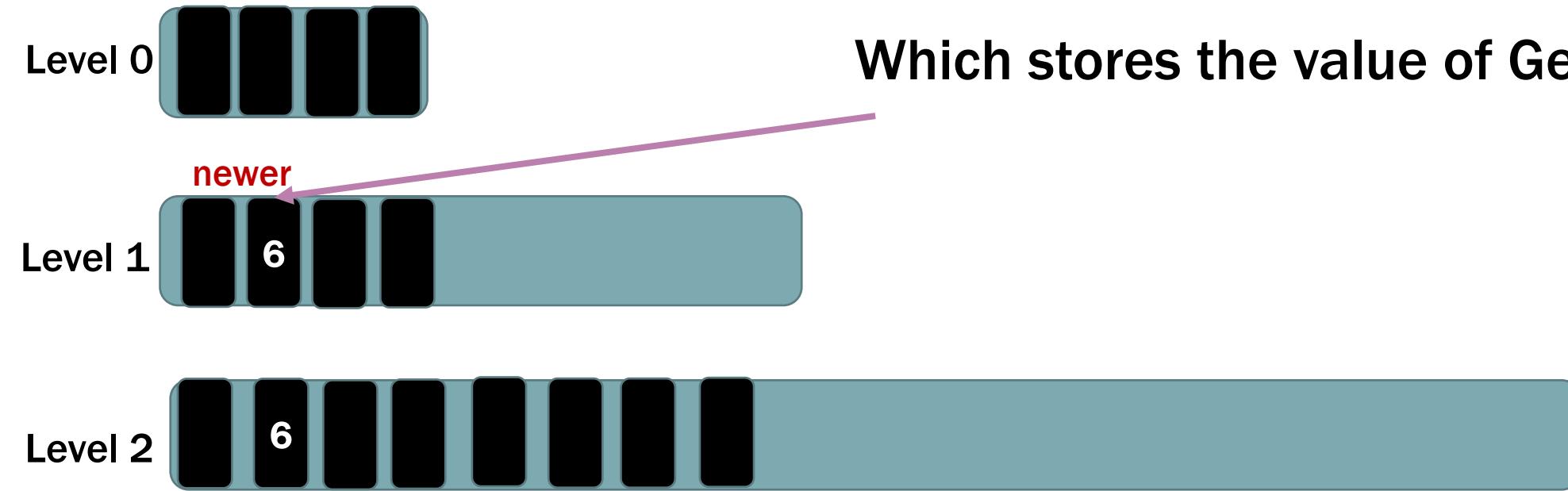
How to implement Get(K)?

Consider Get(6)

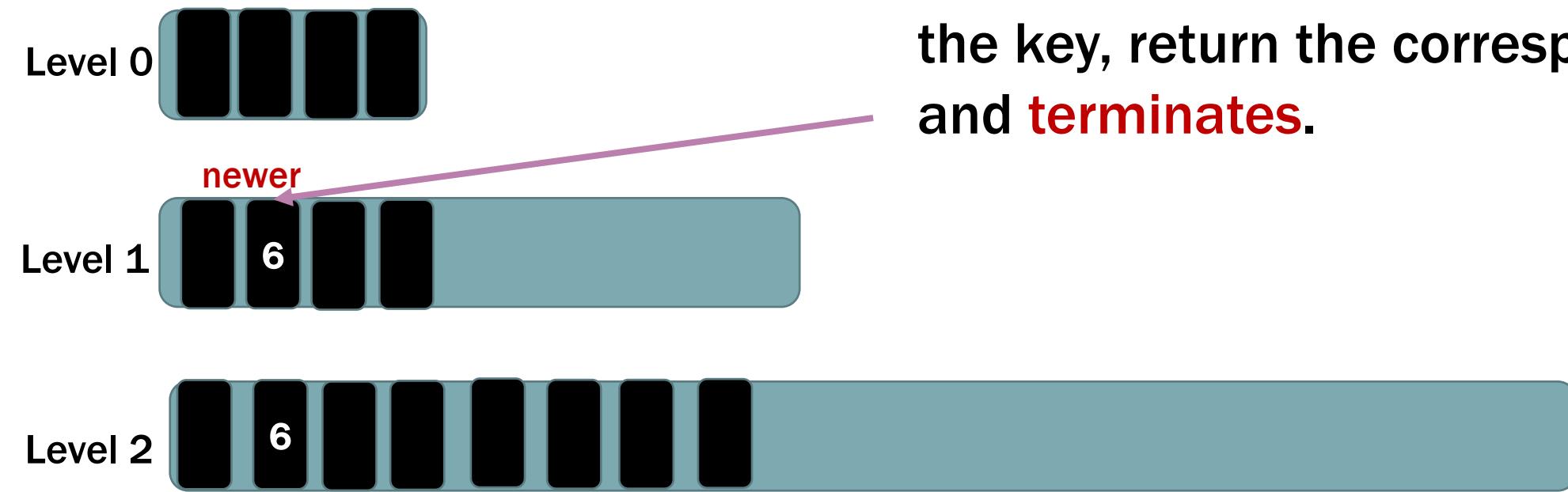


Which stores the value of Get(6)?

Consider Get(6)

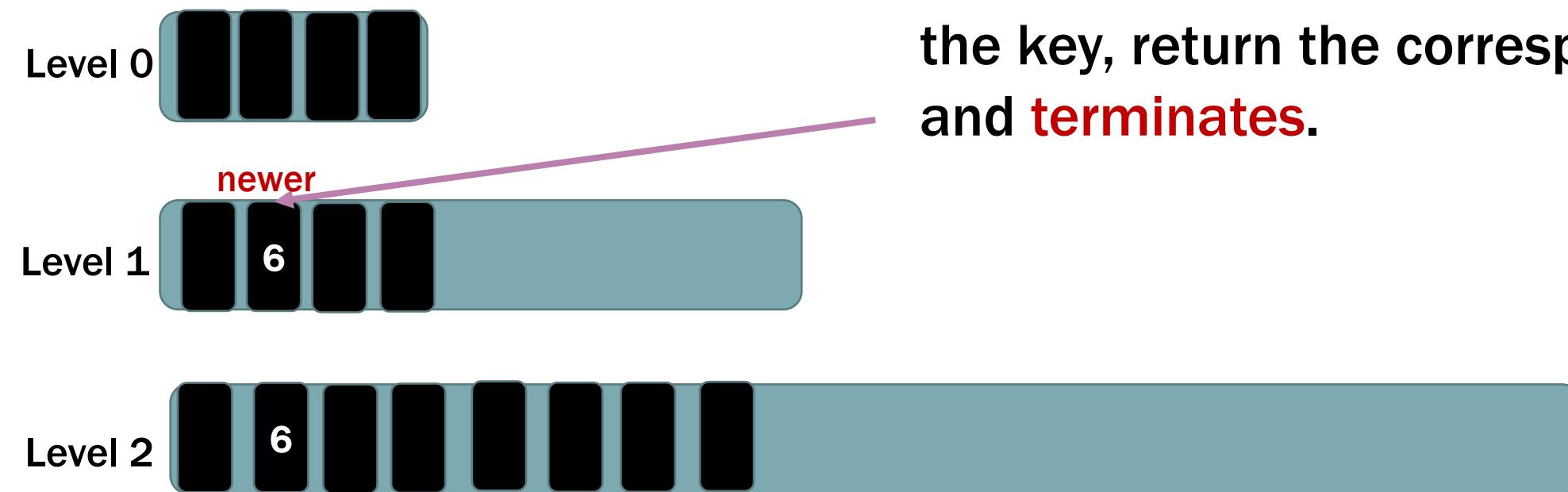


Consider Get(6)



In an L-level LSM-tree,
check from Level 0 to Level L, **until** we find
the key, return the corresponding value,
and **terminates**.

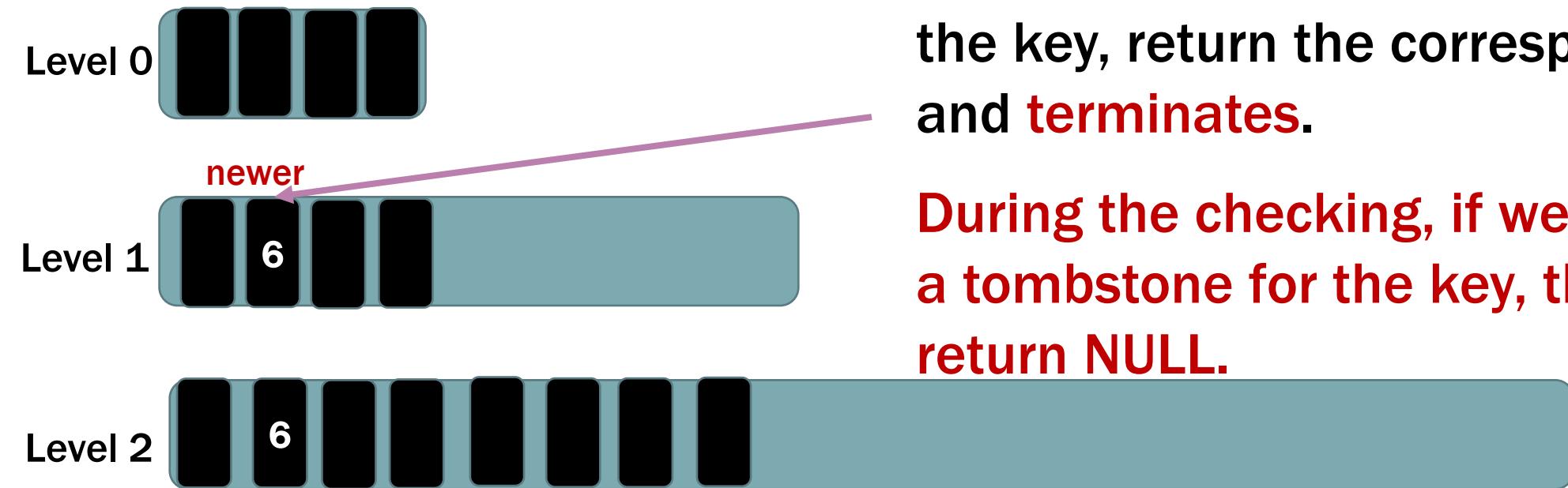
Consider Get(6)



In an L-level LSM-tree,
check from Level 0 to Level L, **until** we find
the key, return the corresponding value,
and **terminates**.

In this case, we do not need to check Level 2 at all.

Consider Get(6)



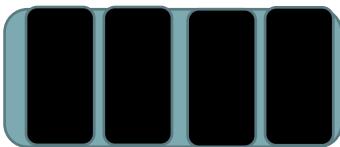
In an L-level LSM-tree,
check from Level 0 to Level L, **until** we find
the key, return the corresponding value,
and **terminates**.

**During the checking, if we find
a tombstone for the key, then
return NULL.**

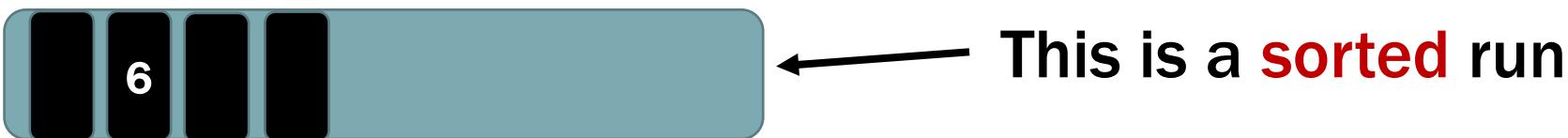
Consider Get(6)

How to **check** each level on disk
(starting from Level 1)?

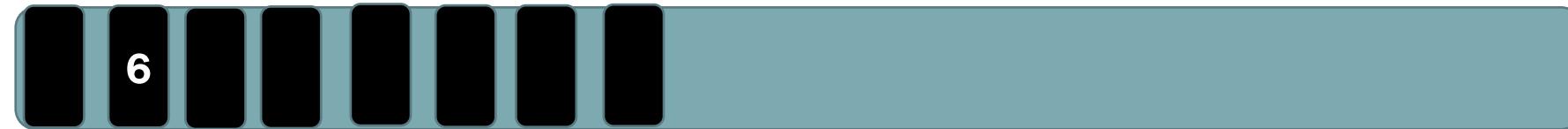
Level 0



Level 1



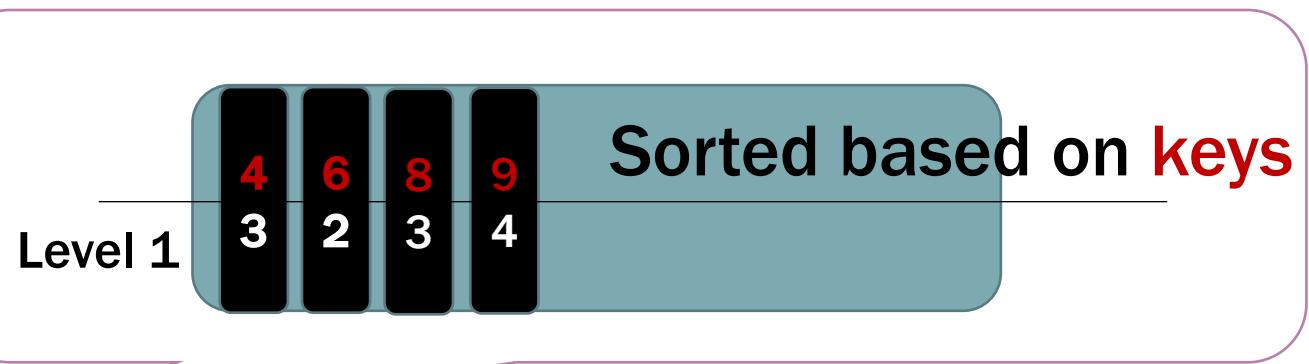
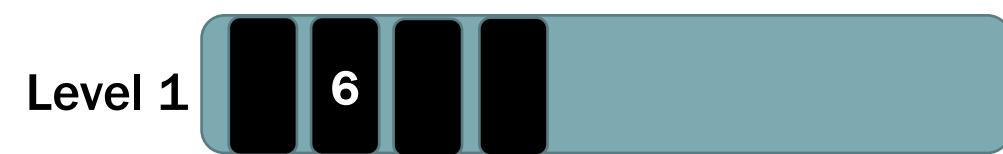
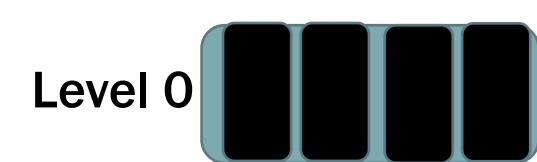
Level 2



This is a **sorted run**

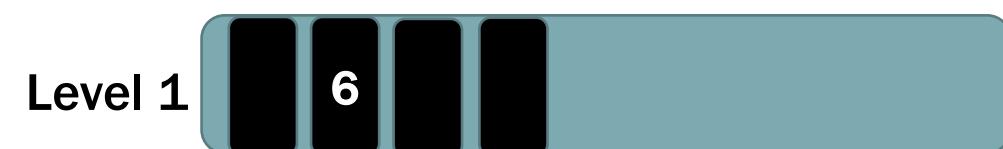
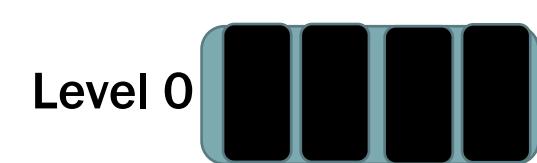
Consider Get(6)

How to **check** each level?

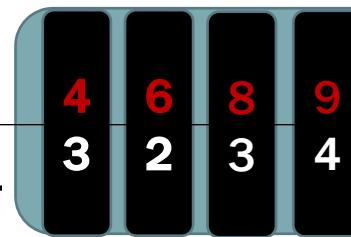


Consider Get(6)

How to **check** each level?



Level 1



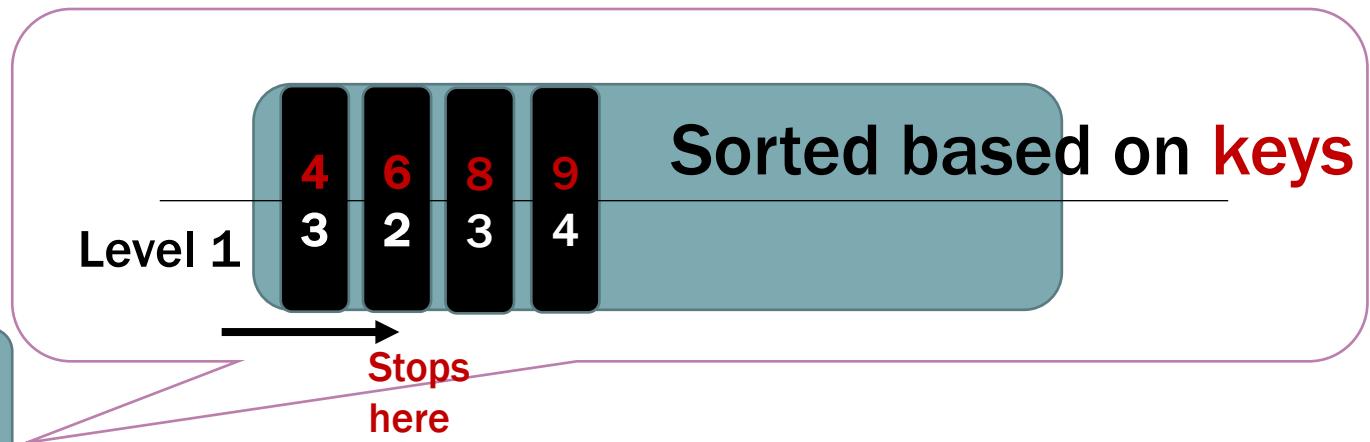
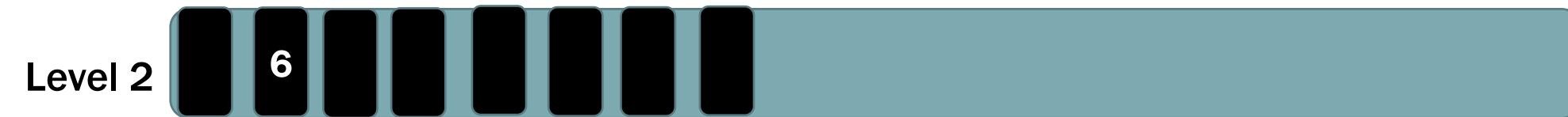
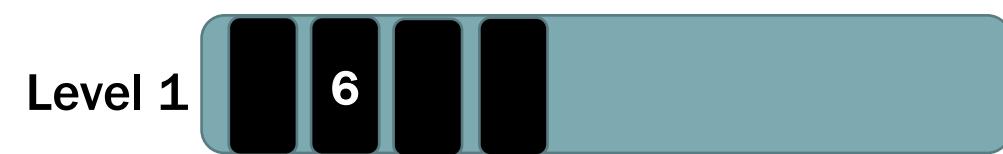
Sorted based on **keys**

First idea:

Searching from left to right, and stops when we find 6

Consider Get(6)

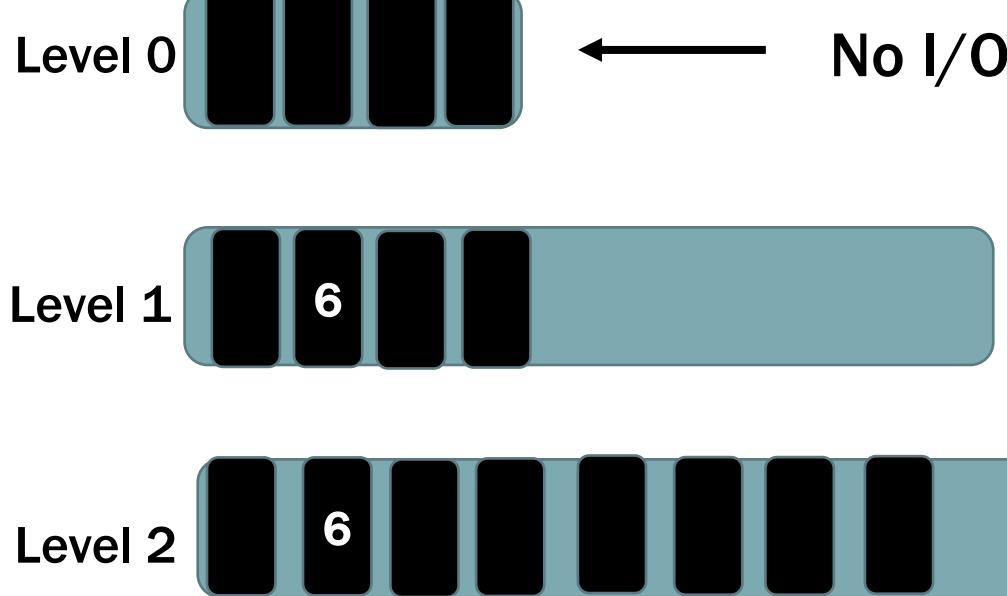
How to **check** each level?



THE COST FOR GET(6)

Consider Get(6)

How to **check** each level?



I/O cost: Number of disk page reads or writes

SOME OTHER POSSIBLE CASES (FOR OTHER KEYS)

Consider Get(6)

How to **check** each level?

Level 0



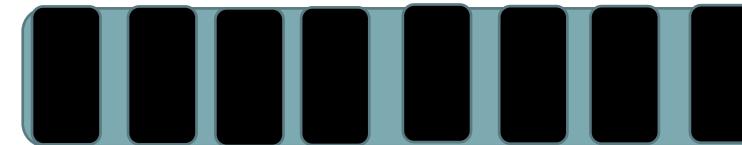
No I/Os (because in main memory)

Level 1



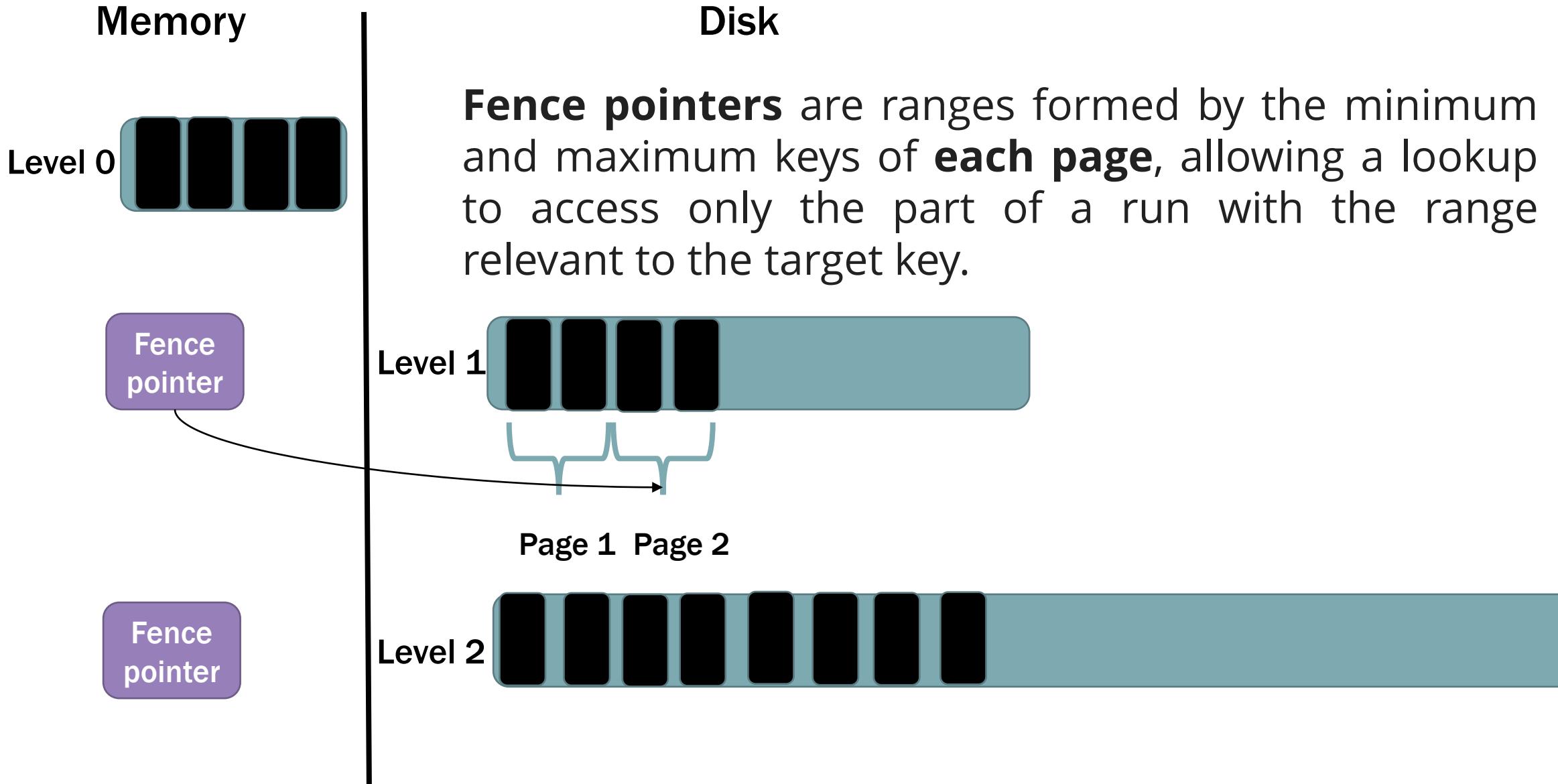
Can have many I/Os

Level 2

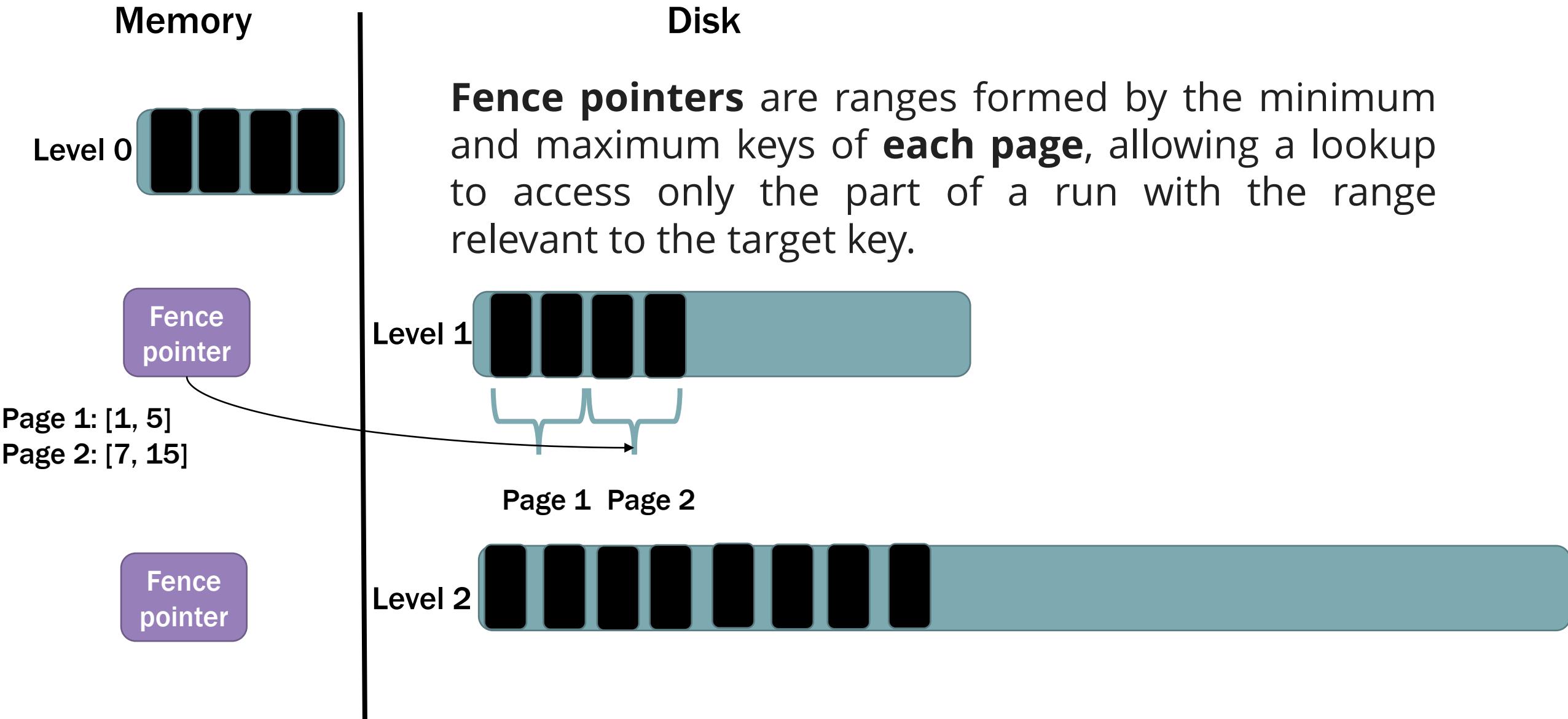


Can have many I/Os

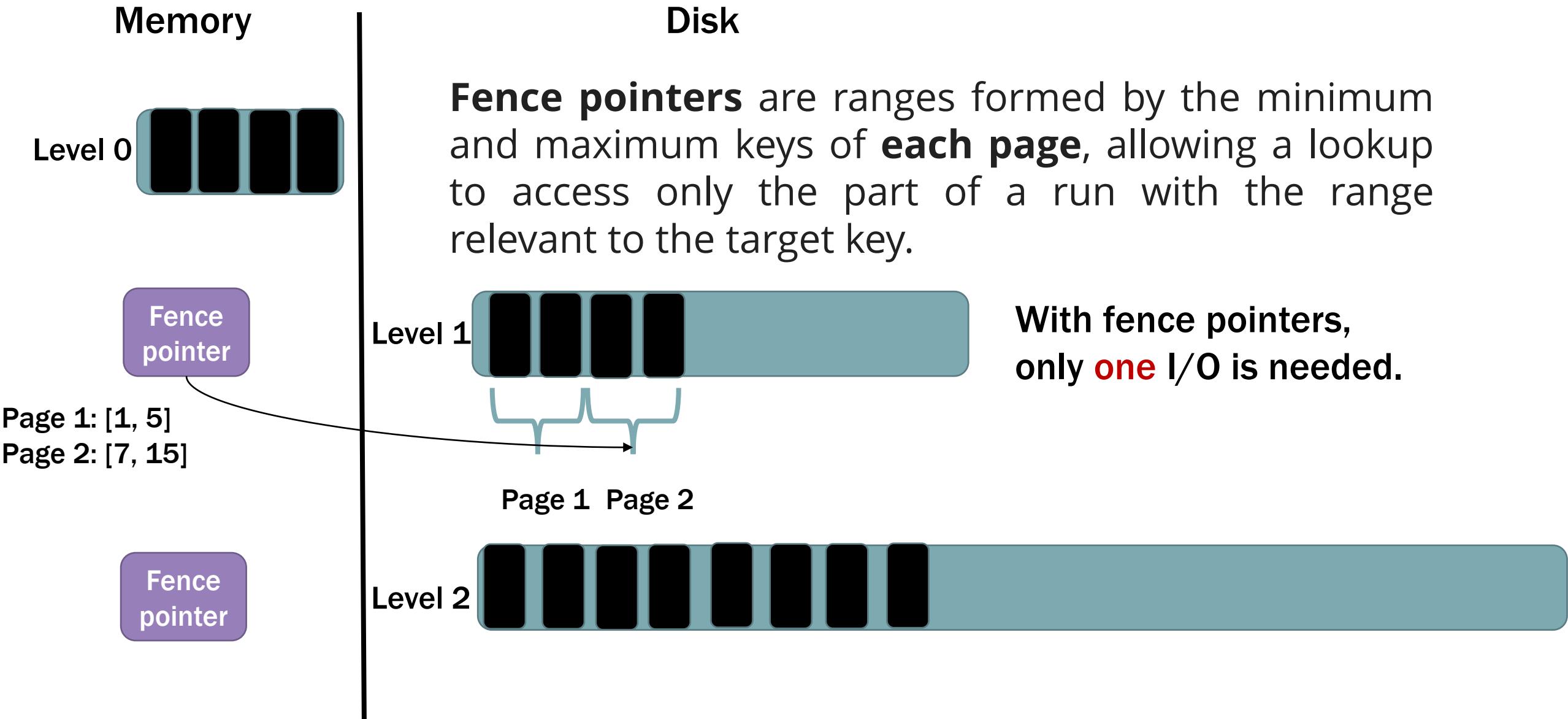
OPTIMIZATION – FENCE POINTERS



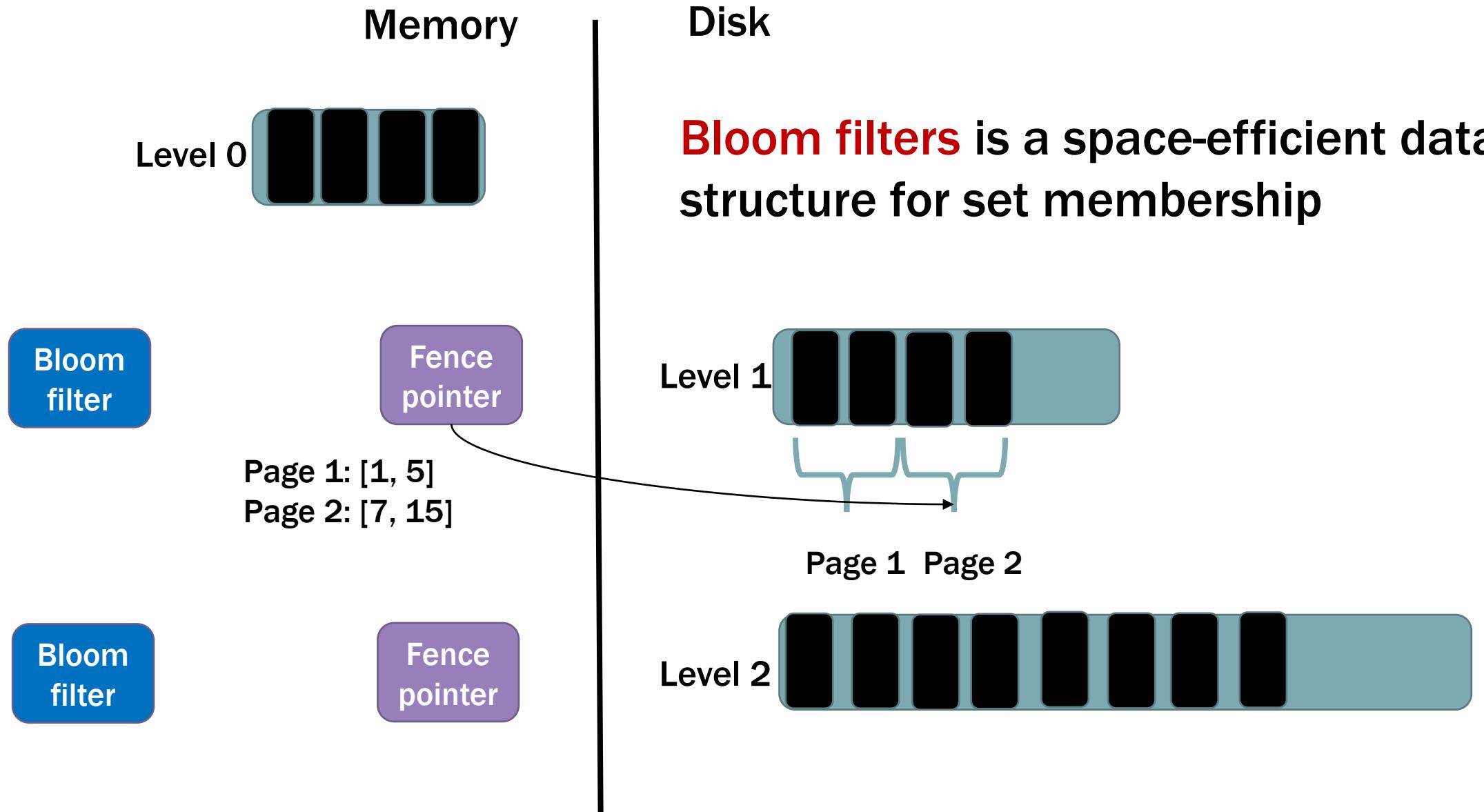
OPTIMIZATION – FENCE POINTERS



OPTIMIZATION – FENCE POINTERS



OPTIMIZATION – BLOOM FILTERS



BIG DATA MANAGEMENT

CE/CZ4123

KEY-VALUE STORE LSM-TREE BASICS

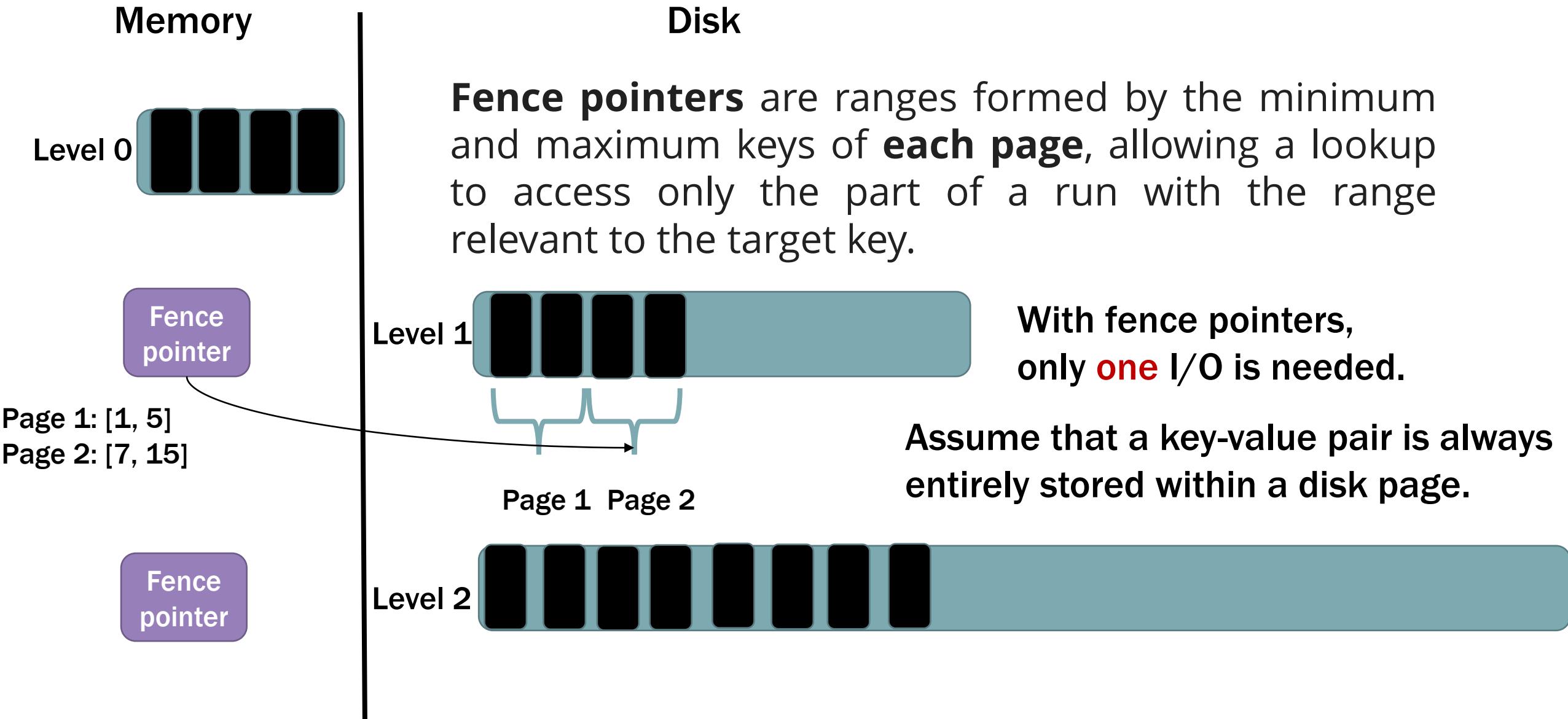
Siqiang Luo

Assistant Professor

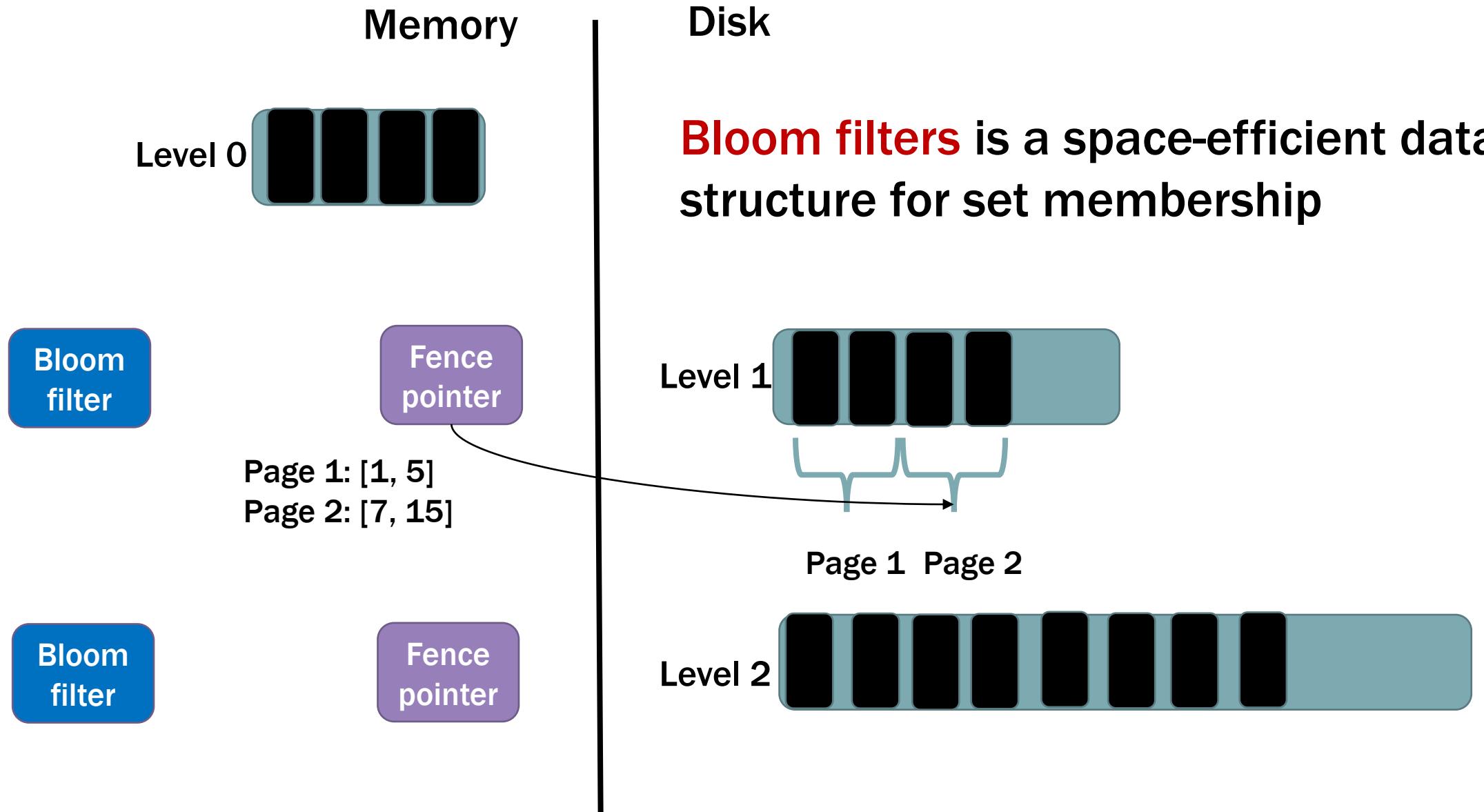
IN PREVIOUS LECTURES

- We discuss the basic structure of an LSM-tree
- We introduce the main functions of an LSM-tree
 - (e.g., Get(), Put(), Delete())
- We introduce
 - (e.g., **fence pointers, Bloom filters**)

OPTIMIZATION – FENCE POINTERS



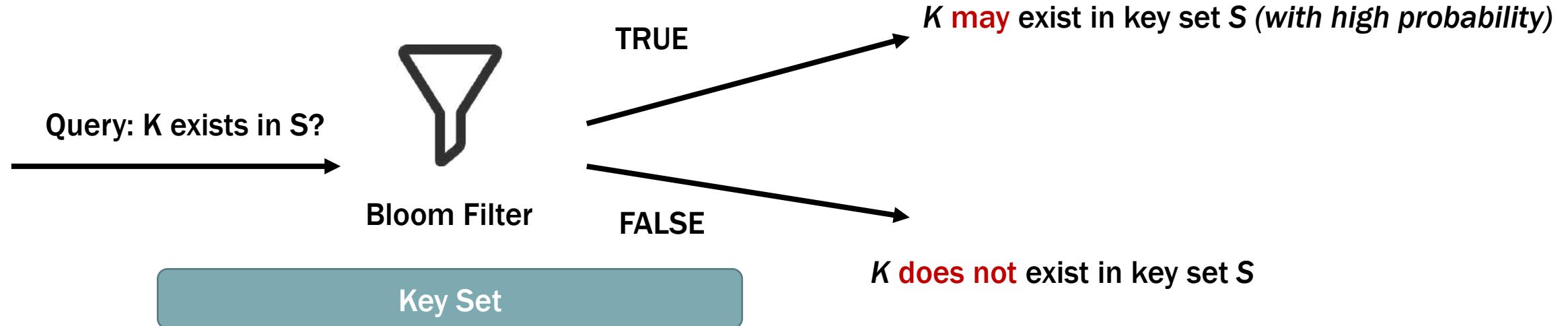
OPTIMIZATION – BLOOM FILTERS



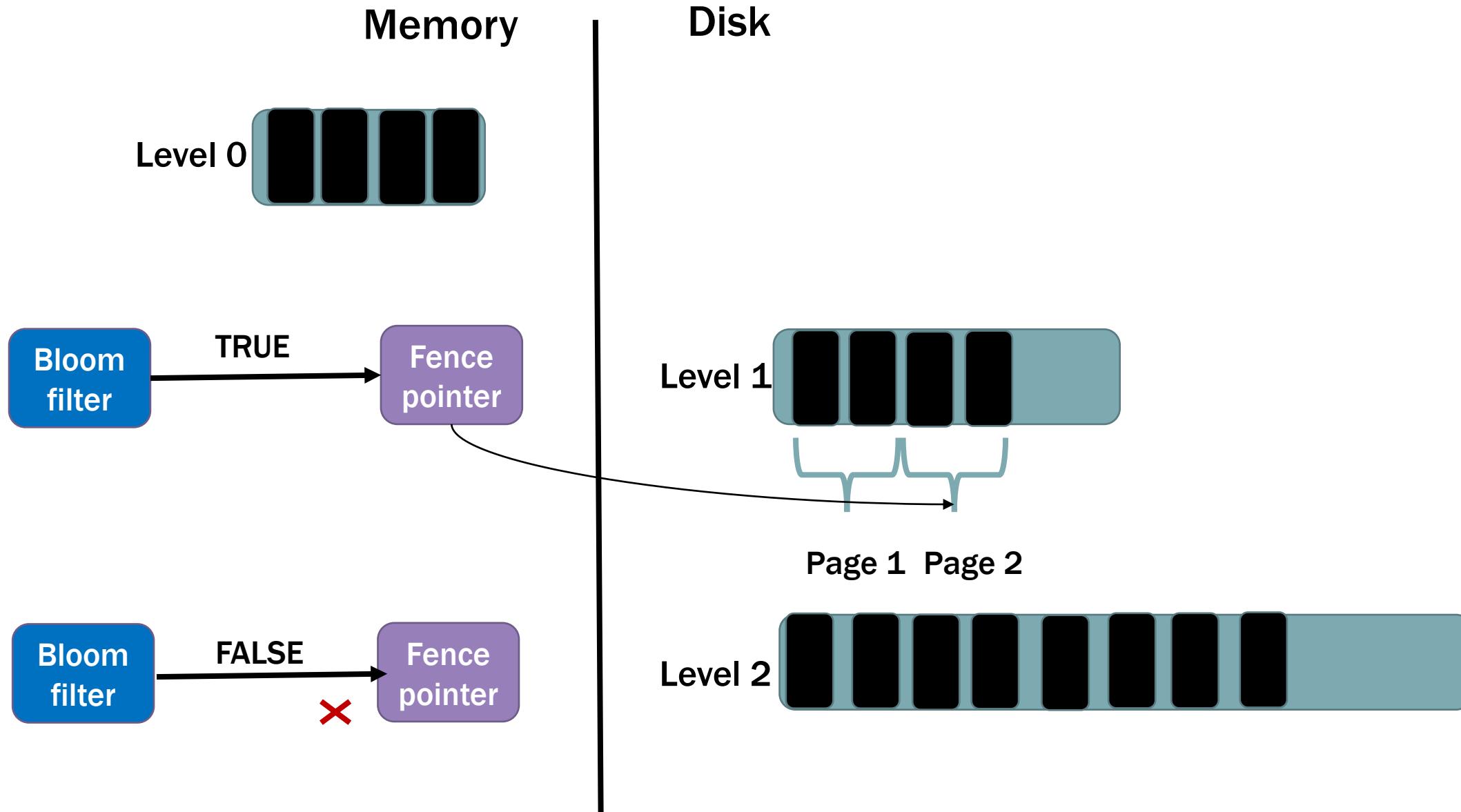
BLOOM FILTER

1. Stored in main memory
2. Built on a set S of keys
3. Given a key K , the Bloom filter answers TRUE or FALSE for key K
4. If it answers FALSE, it means the key K **does not** exist in key set S
5. If it answers TRUE, it means the key K **may** exist in key set S , and it is still possible that the key K does not exist in key set S .
6. FPR (False-Positive Rate) is the probability that the filter returns TRUE for a key K , but actually K does not exist in set S . We usually use P to denote FPR. Clearly, P is in $[0, 1]$ (e.g., $P=0.3$)

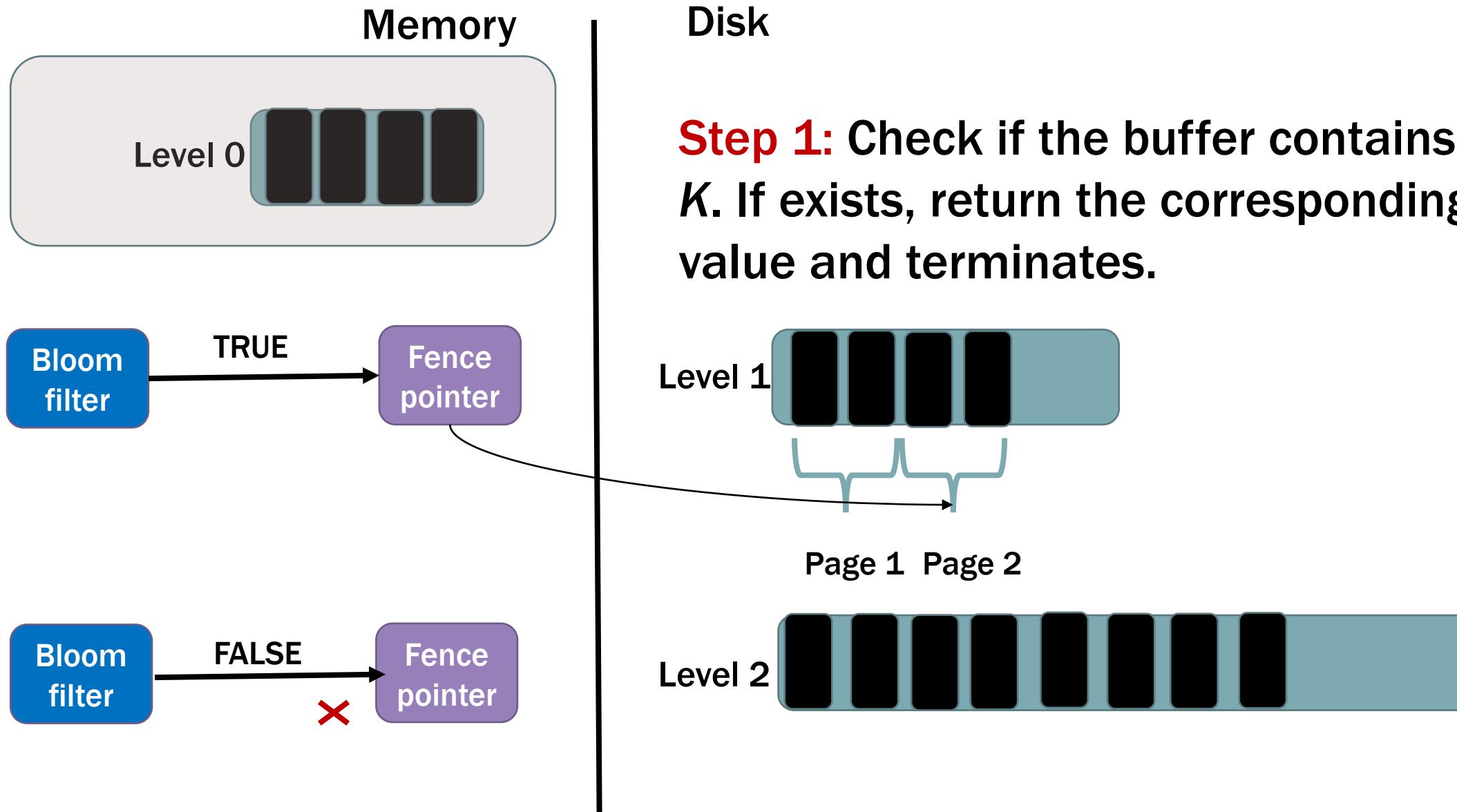
ILLUSTRATION OF BLOOM FILTER



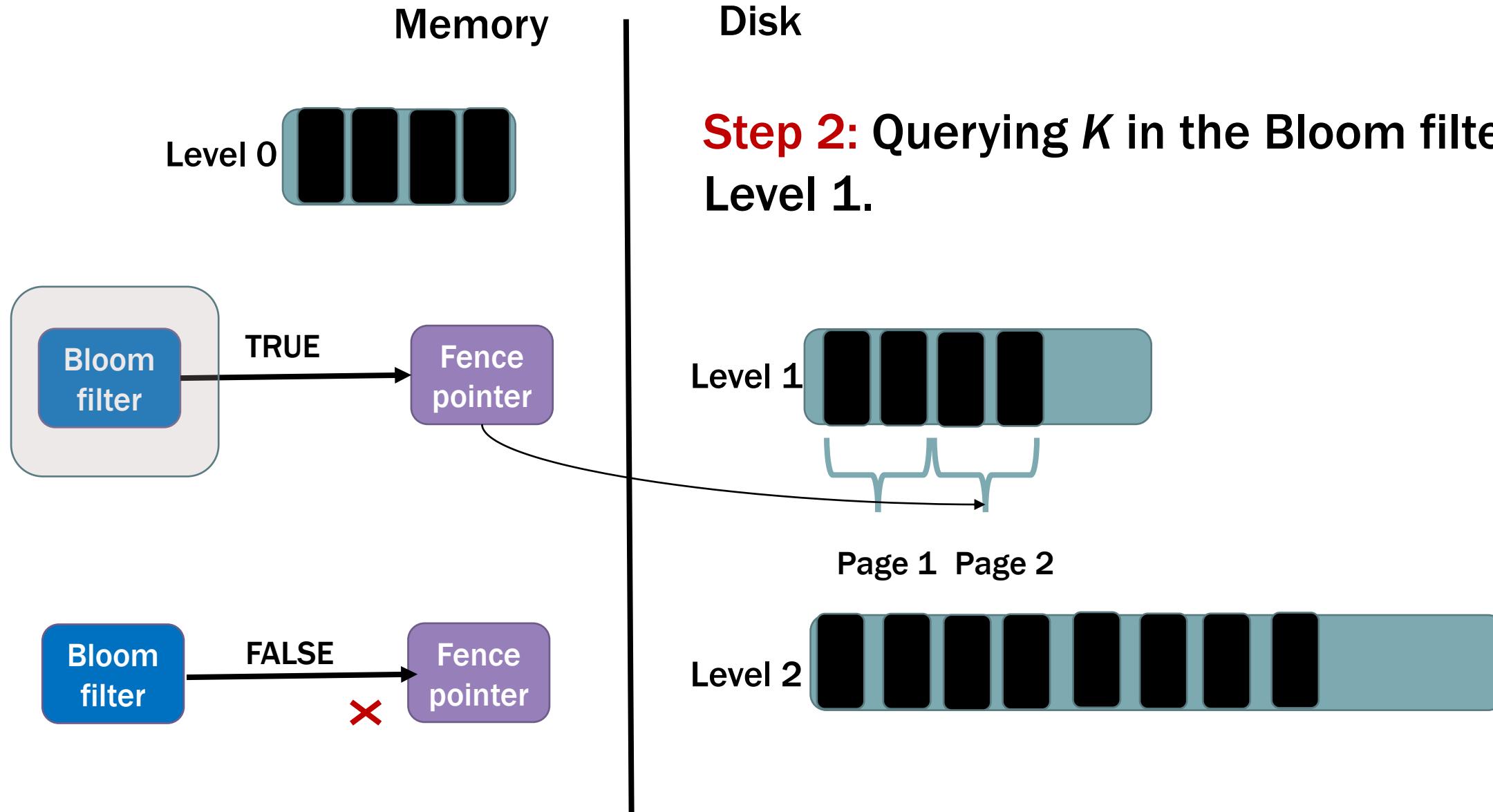
WHY BLOOM FILTER IS HELPFUL?



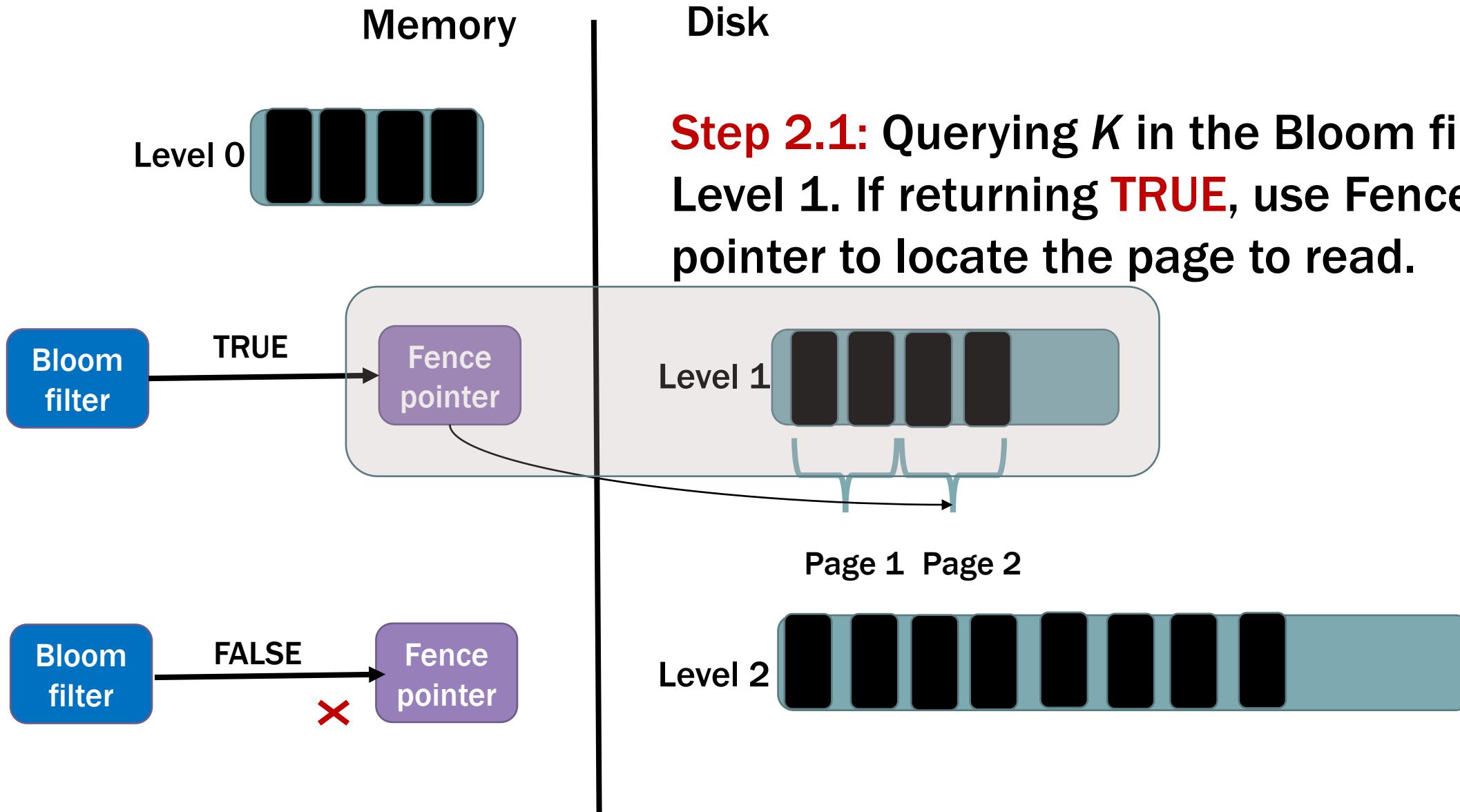
THE PROCEDURE OF GET(K) - WITH FENCE POINTERS AND BLOOM FILTERS



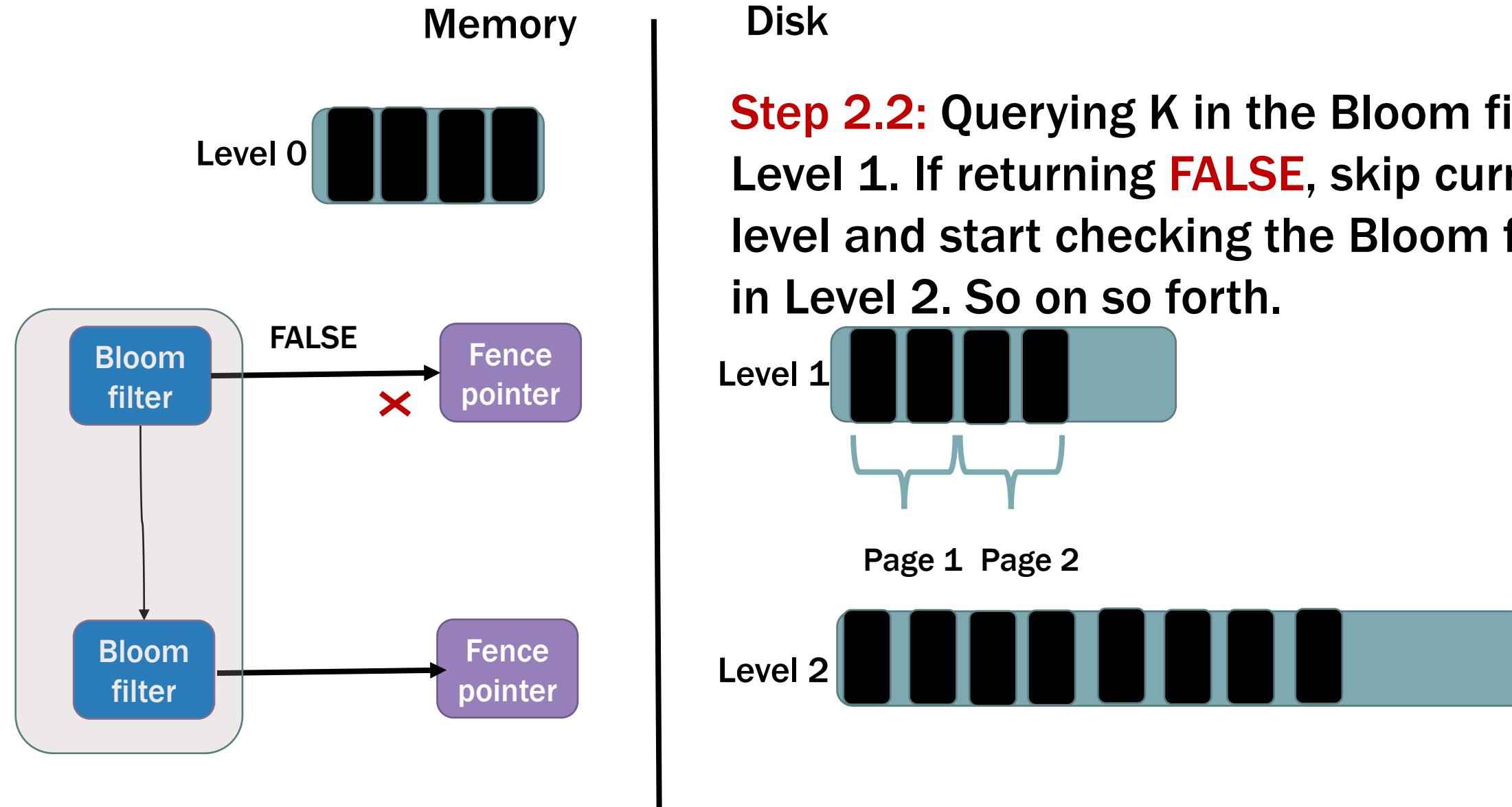
THE PROCEDURE OF GET(K) - WITH FENCE POINTERS AND BLOOM FILTERS



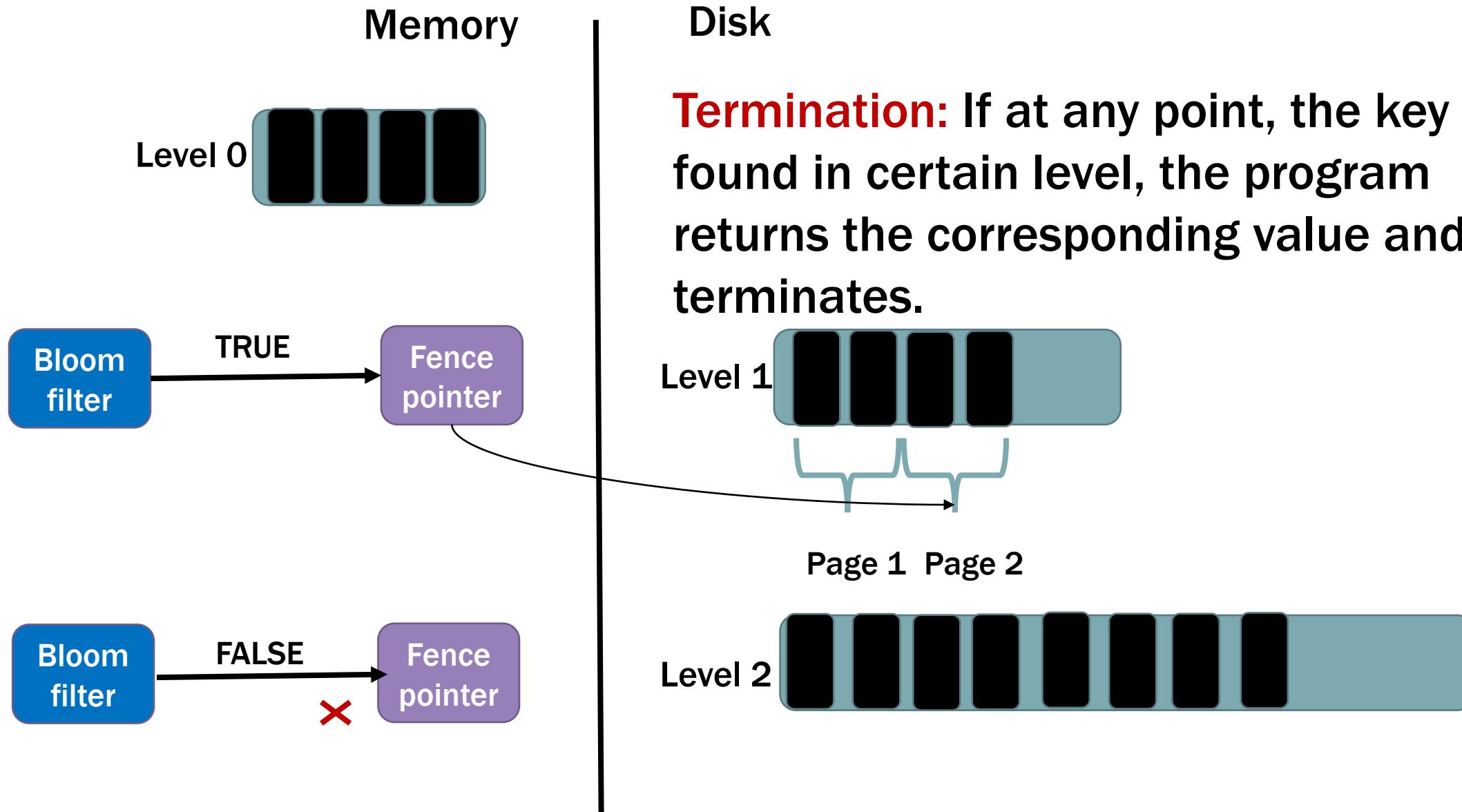
THE PROCEDURE OF GET(K) - WITH FENCE POINTERS AND BLOOM FILTERS



THE PROCEDURE OF GET(K) - WITH FENCE POINTERS AND BLOOM FILTERS



THE PROCEDURE OF GET(K) - WITH FENCE POINTERS AND BLOOM FILTERS



SUMMARY

1. Check buffer level (Level 0)
2. Starting from Level 1, always check its Bloom filter first, and then its Fence pointer.
3. If Bloom filter returns FALSE, then do not access the fence pointer and directly go to the next level for checking.
4. If Bloom filter returns TRUE, then access the fence pointer to fetch the corresponding page in that level. If the key is found, return the value and terminate the program; otherwise go to the next level for checking.
5. If the key has not been found in any level, return NULL (or empty set) and terminate the program.

INTERNAL DESIGNS OF BLOOM FILTER

The main purpose of the filter:

- Built on a set of keys $S=\{K_1, K_2, \dots, K_n\}$, where each key is from a large universe U .
- Bloom filter can 100% determine if a key is NOT in S , and with HIGH probability it can tell if a key is in S .
- Space efficient: on average, each key only occupies a few bits in Bloom filter.
- Inserting a new element into the Bloom filter should be fast
- Querying a key from the Bloom filter should be fast

THE CORE OF BLOOM FILTER – HASH FUNCTION

A hash function h maps a uniformly at random chosen key $x \in U$ to an integer from $R_m = [0, m-1]$ such that each element in R_m is mapped with equal probability.

A Bloom filter is a bit vector B of m bits, with k independent hash functions h_1, \dots, h_k

- Initially all the m bits are 0.
- Insert x into S : compute $h_1(x), \dots, h_k(x)$ and set $B[h_1(x)] = B[h_2(x)] = \dots = B[h_k(x)] = 1$.
- Query if $x \in S$: Compute $h_1(x), \dots, h_k(x)$. If $B[h_1(x)] = B[h_2(x)] = \dots = B[h_k(x)] = 1$, then answer TRUE, else answer FALSE.

EXAMPLE

- $m=5, k=2$
- $h_1(x) = x \bmod 5$
- $h_2(x) = (2x+3) \bmod 5$
- Initially $B[0]=B[1]=B[2]=B[3]=B[4]=B[5]=0$
- Then insert 9 and 11

	$h_1(x)$	$h_2(x)$	B					
Initialize:			0	0	0	0	0	
Insert 9:	4	1	0	1	0	0	1	
Insert 11:	1	0	1	1	0	0	1	

EXAMPLE

Now query 15 and 16

	$h_1(x)$	$h_2(x)$	Answer
Query 15:	0	3	No, not in B (correct answer)
Query 16:	1	0	Yes, in B (wrong answer: false positive)

ANALYSIS

- When m/n is fixed (m/n is often called bits-per-key), the optimal k is $\ln 2 \times (m/n)$ (See [here](#) for proof if you are interested)
- Then, the optimal FPR is about $0.6185^{m/n}$
- So, larger m means small FPR (approaches to 0); smaller m means higher FPR (approaches to 1).

SUMMARY OF LSM-TREE

- ❑ Multi-level structured
- ❑ Out-of-place updates (delete acts as a special insert)
- ❑ Fence pointers → reduce I/Os of Get()
- ❑ Bloom filters → reduce I/Os of Get()

CLARIFICATIONS

In practice, fence pointers can be implemented in different ways.

Option 1: containing the min/max of each page;

Option 2: containing only the min (or max) of each page;

For analysis purpose, we reasonably assume that when using Fence pointers, it always incurs one I/O.

VARIANTS OF LSM-TREE

- **Leveled LSM-tree (or Leveling LSM-tree)**
 - The LSM-tree with leveling merge policy.
- **Tiered LSM-tree (or Tiering LSM-tree)**
 - The LSM-tree with tiering merge policy.
- Until now, the LSM-tree we introduce belongs to Leveling LSM-tree.
- Next, we introduce what is Tiering LSM-tree

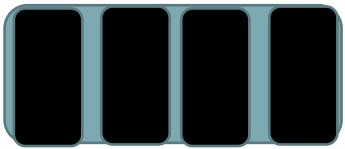
LEVELING LSM-TREE VS TIERING LSM-TREE

- The difference lies in the way of merging a full level to its next level
- Leveling LSM-tree:
 - When a level needs to be merged, always sort-merge to the next level
- Tiering LSM-tree:
 - When a level needs to be merged, does not sort but put it as a *tier* stored in the next level

LEVELING LSM-TREE EXAMPLE

Memory buffer

Level 0

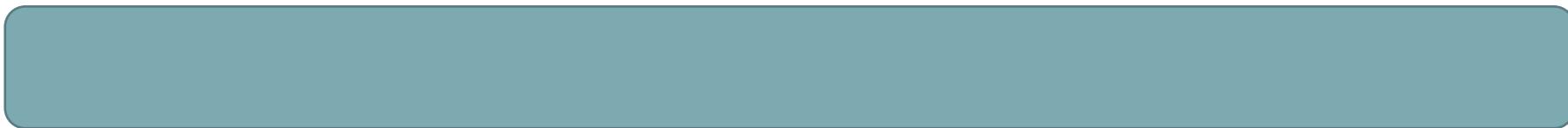


Full

Level 1

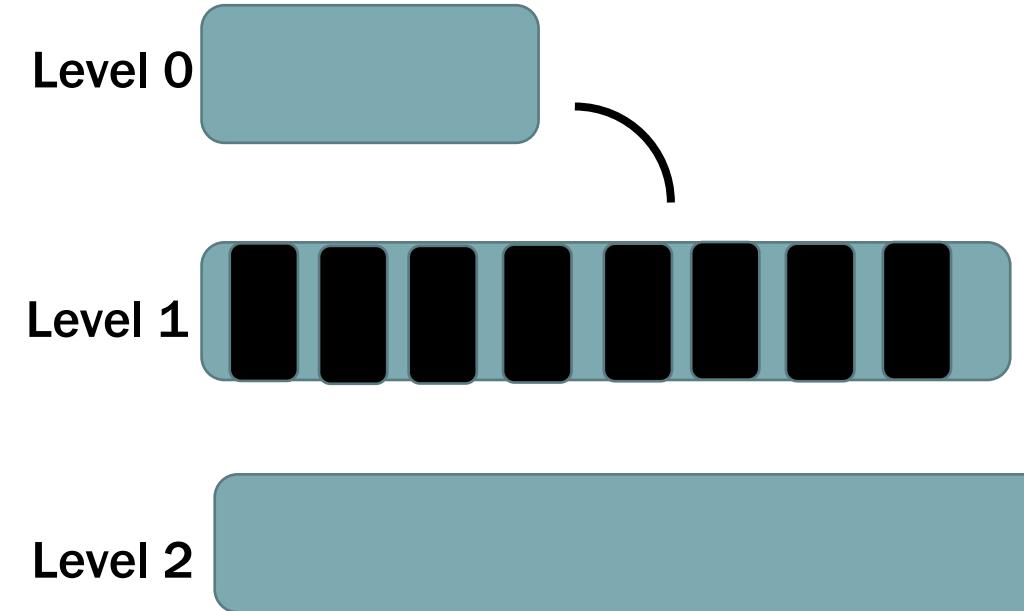


Level 2



LEVELING LSM-TREE EXAMPLE

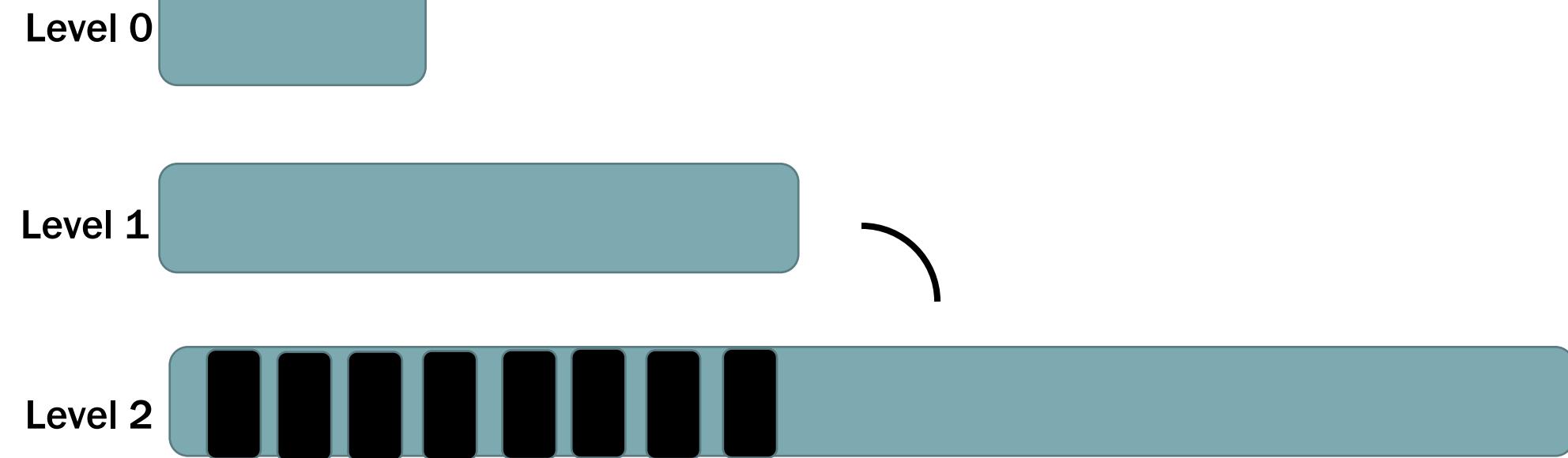
Memory buffer



Trigger merge condition

LEVELING LSM-TREE EXAMPLE

Memory buffer



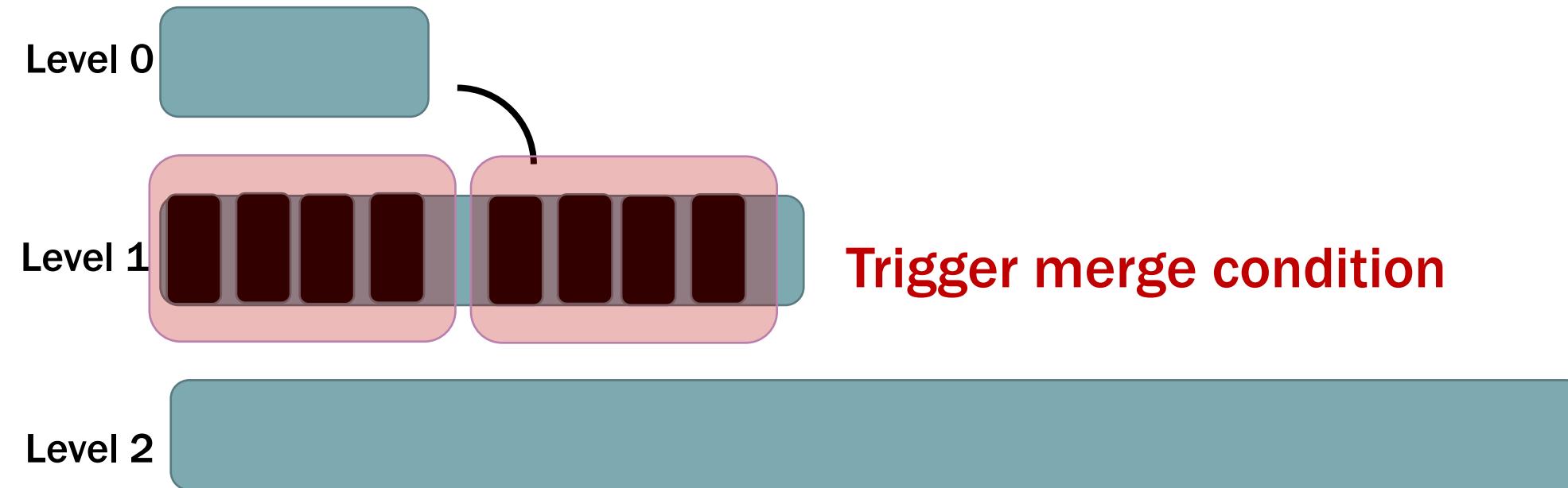
TIERING LSM-TREE EXAMPLE

Memory buffer



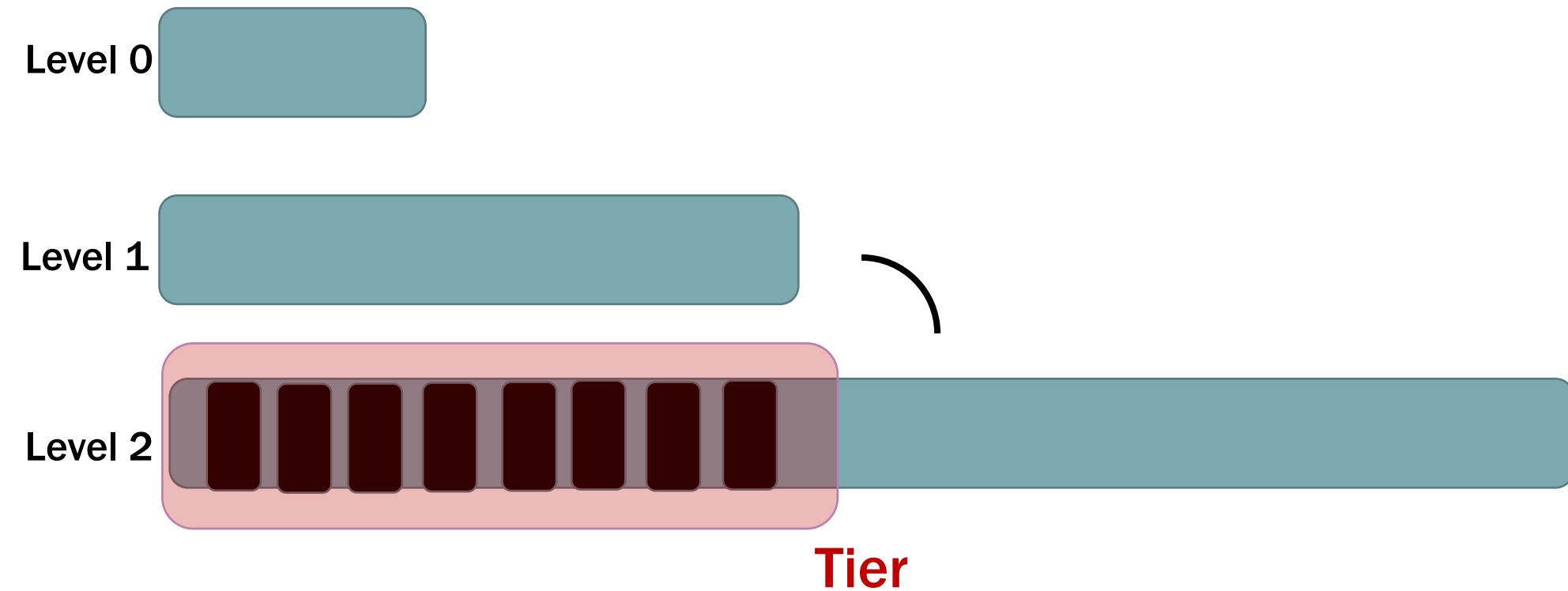
TIERING LSM-TREE EXAMPLE

Memory buffer



TIERING LSM-TREE EXAMPLE

Memory buffer



SOME PROPERTIES

- A tier in Level i **roughly** has the size of the capacity of Level $(i-1)$

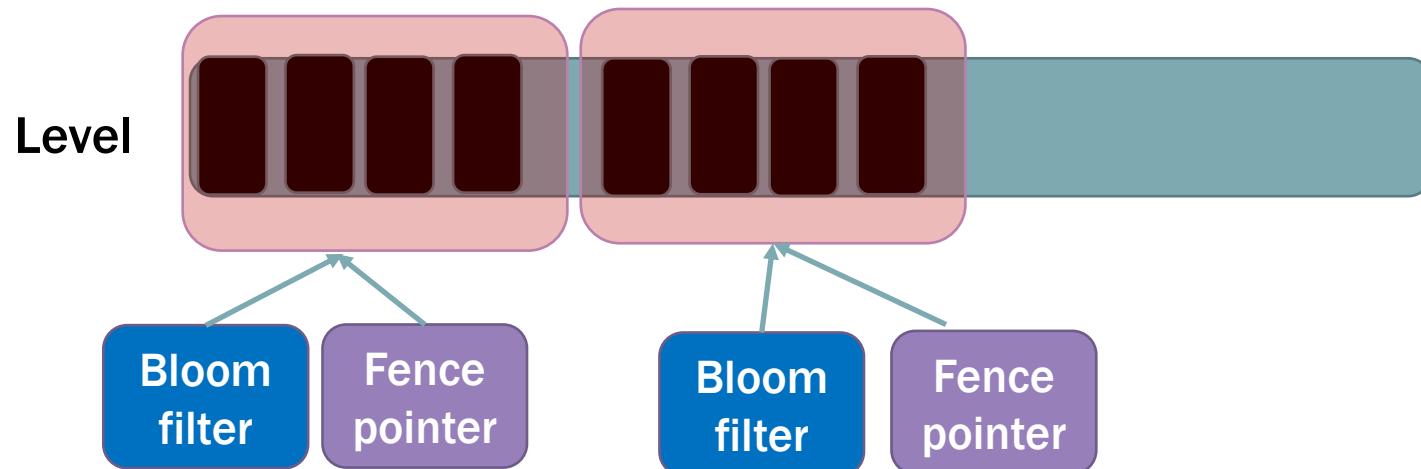
- Usually, in tiering LSM-tree, starting from Level 1, the merge is triggered when there are T tiers in it, where T is the size ratio.

- Question: In each level of tiering LSM-tree, is a key unique?



FENCE POINTERS AND BLOOM FILTERS IN TIERING LSM-TREES

Bloom filter and fence pointers are built for each tier of each disk level (except Level 0)



PROS AND CONS OF TIERING LSM-TREE

❑ Advantages:

- ❑ Avoid costly sort merges
- ❑ Put/Delete is faster

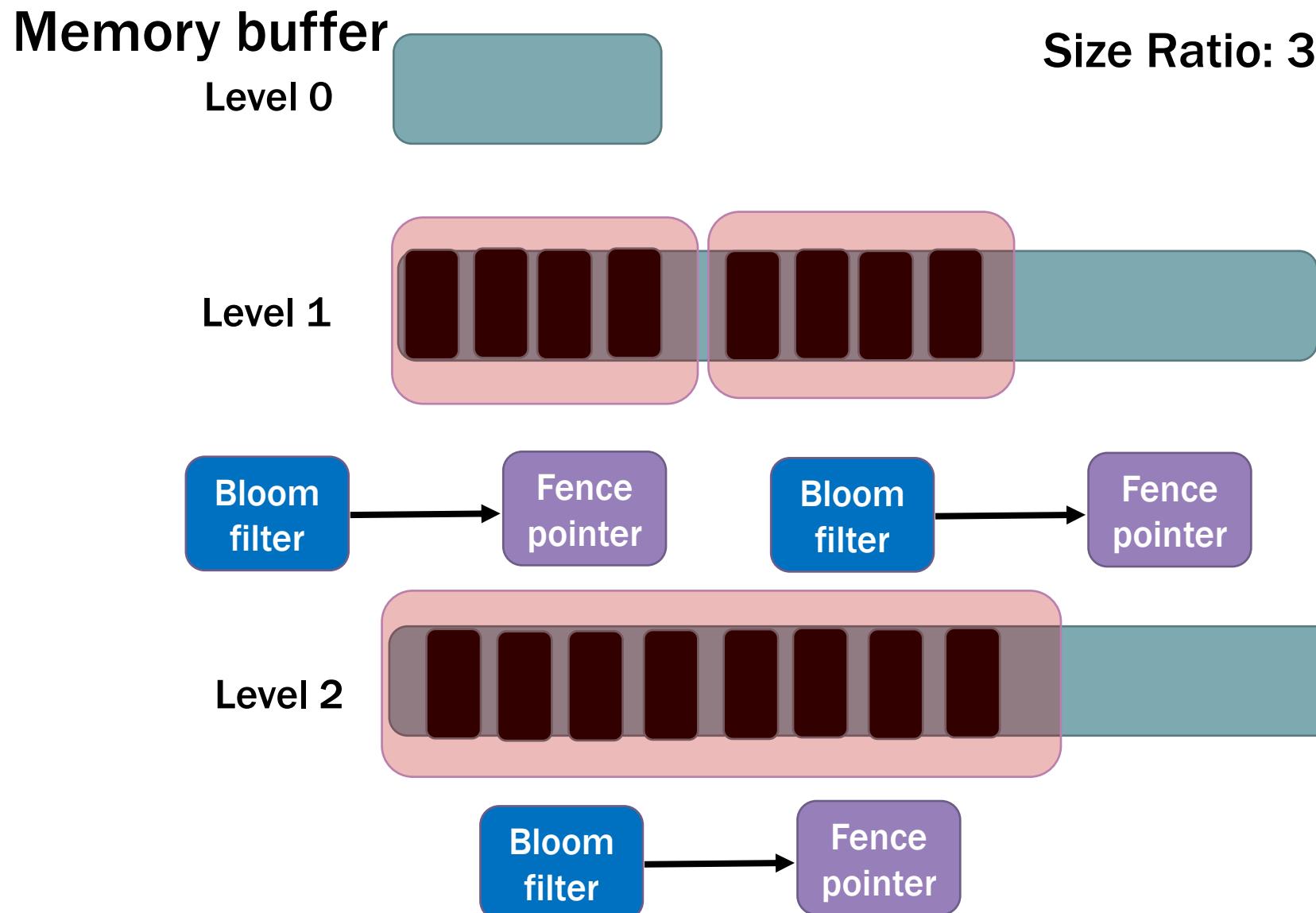
❑ Disadvantages:

- ❑ Get is slower

❑ Summary

- ❑ Leveling LSM-tree: faster data reads, slower data writes
- ❑ Tiering LSM-trees: slower data reads, faster data writes

PERFORMING GET(K) IN TIERING LSM-TREE



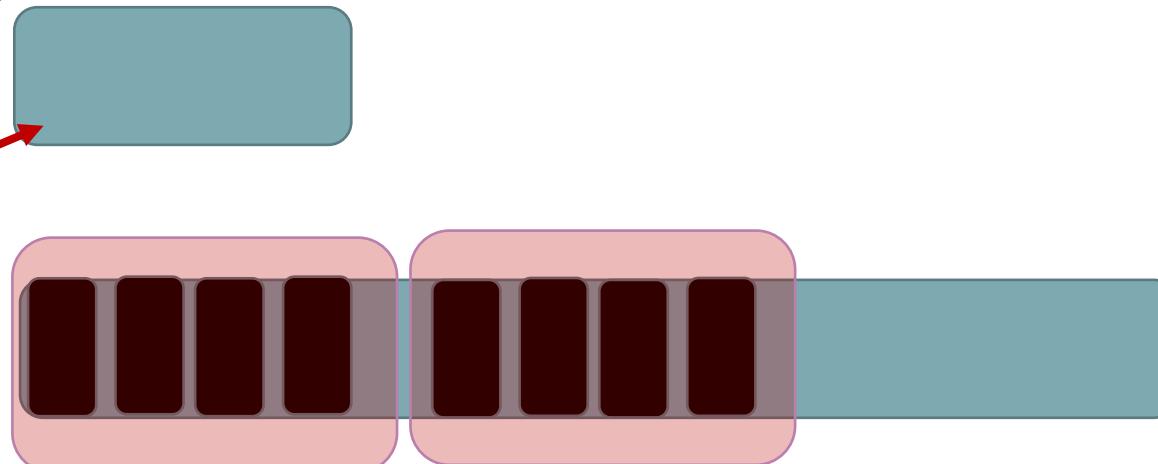
PERFORMING GET(K) IN TIERING LSM-TREE

Memory buffer

Level 0



Level 1



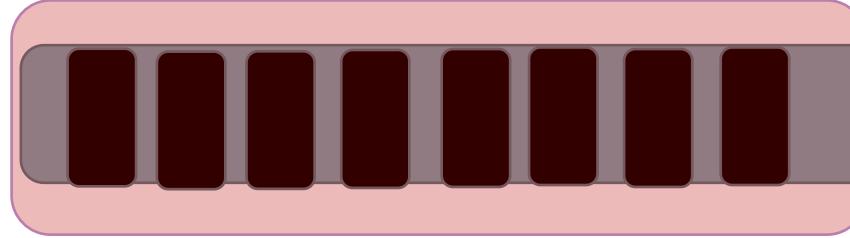
Bloom
filter

Fence
pointer

Bloom
filter

Fence
pointer

Level 2



Bloom
filter

Fence
pointer

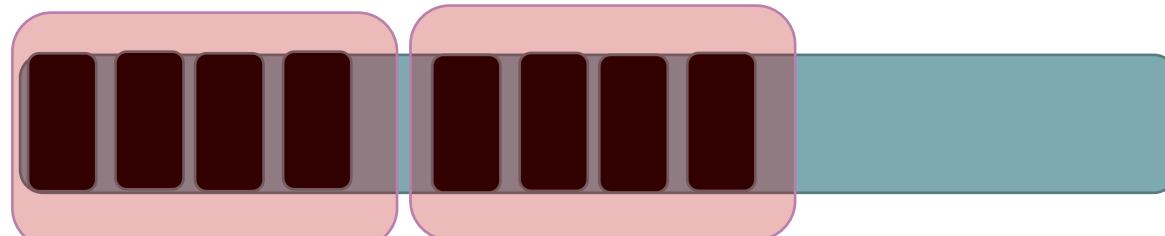
PERFORMING GET(K) IN TIERING LSM-TREE

Memory buffer

Level 0



Level 1



Bloom
filter

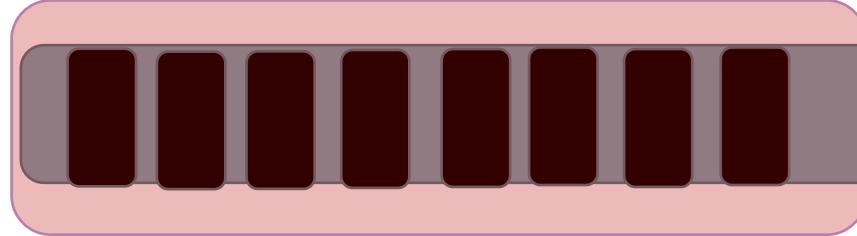
Fence
pointer

Bloom
filter

Fence
pointer

FALSE


Level 2



Bloom
filter

Fence
pointer

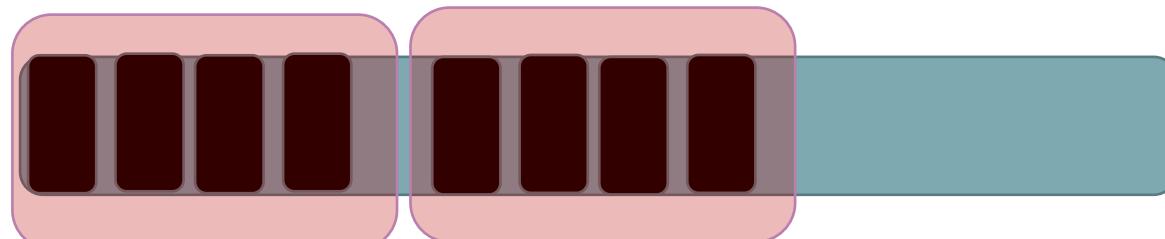
PERFORMING GET(K) IN TIERING LSM-TREE

Memory buffer

Level 0



Level 1



Bloom
filter

TRUE

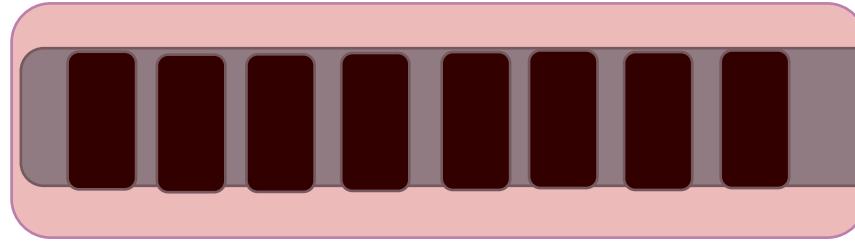
Fence
pointer

Bloom
filter

FALSE

Fence
pointer

Level 2



Bloom
filter

Fence
pointer

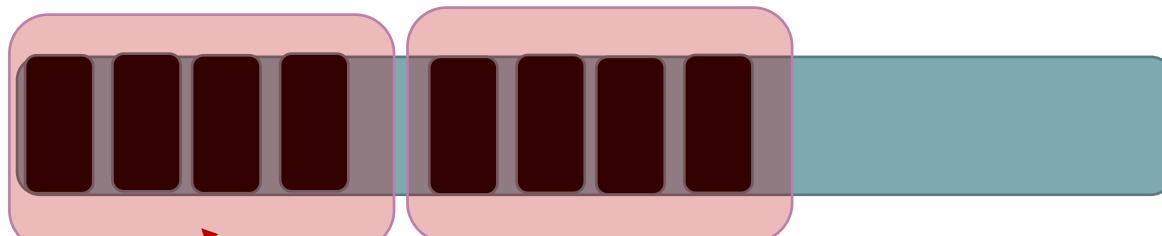
PERFORMING GET(K) IN TIERING LSM-TREE

Memory buffer

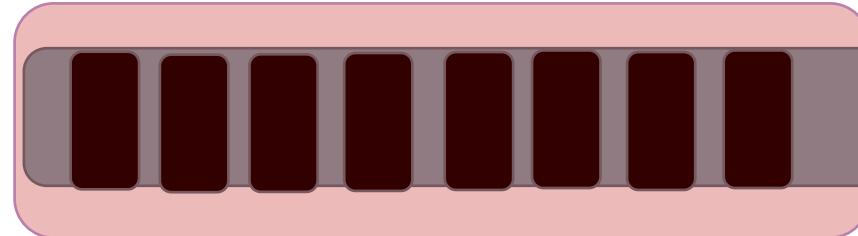
Level 0



Level 1



Level 2



Bloom
filter

Fence
pointer

Fence
pointer

Fence
pointer

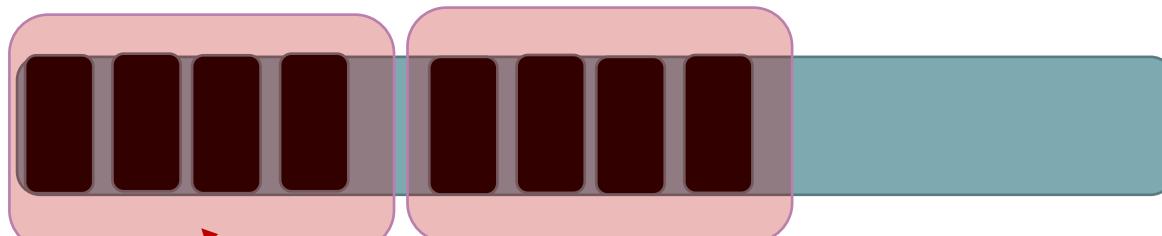
PERFORMING GET(K) IN TIERING LSM-TREE

Memory buffer

Level 0



Level 1



If found, terminate the search

Bloom
filter

TRUE

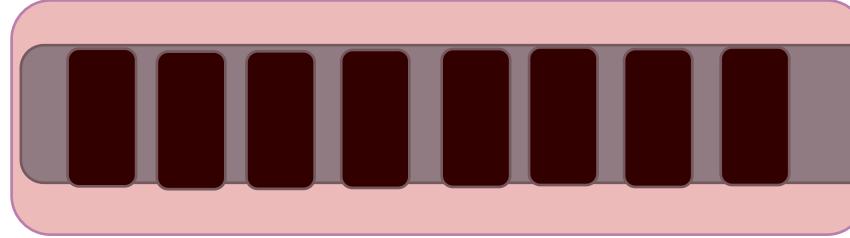
Fence
pointer

Bloom
filter

FALSE

Fence
pointer

Level 2



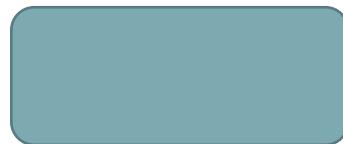
Bloom
filter

Fence
pointer

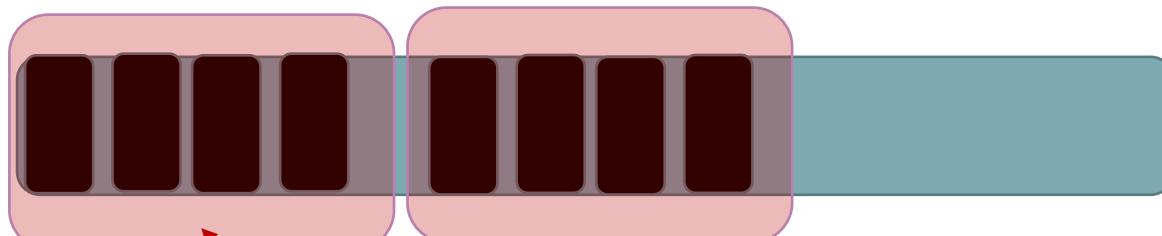
PERFORMING GET(K) IN TIERING LSM-TREE

Memory buffer

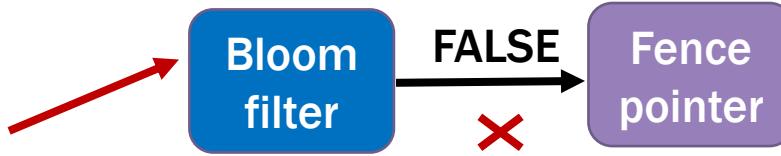
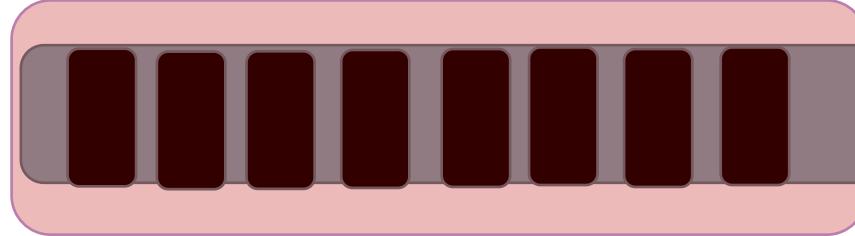
Level 0



Level 1



Level 2



**The End
Thank you!**

BIG DATA MANAGEMENT

CE/CZ4123

KEY-VALUE STORE LSM-TREE (EXTENSIONS)

Siqiang Luo

Assistant Professor

How to implement Range-Get ([4,9])?

Range-Get([4,9]) → find out **ALL** values with keys larger than 4 and smaller than 9 in the LSM-tree (For simplicity, we assume key and value are equal in the example.)

Level 0

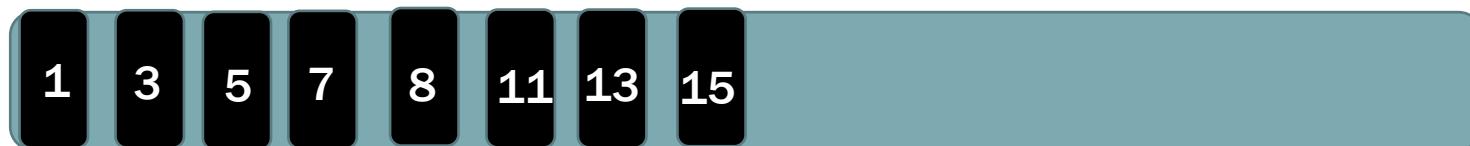


Idea: top-to-down search like GET function

Level 1



Level 2



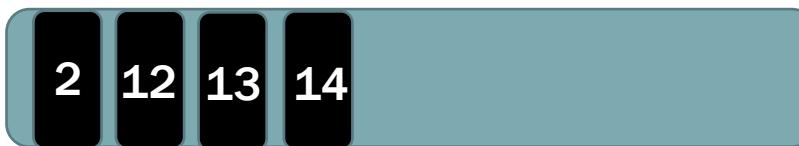
Leveling Structure

Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9

Level 0



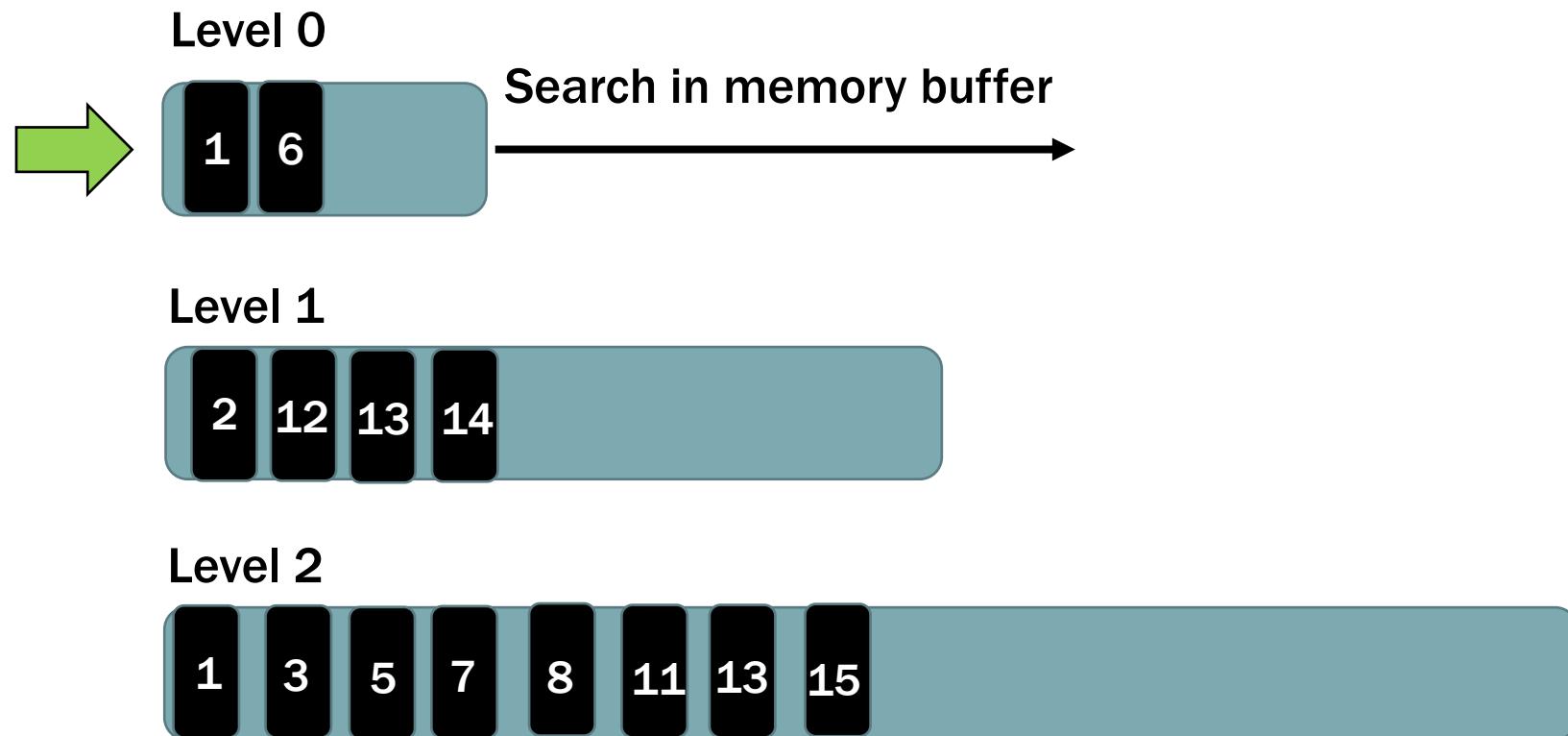
Level 1



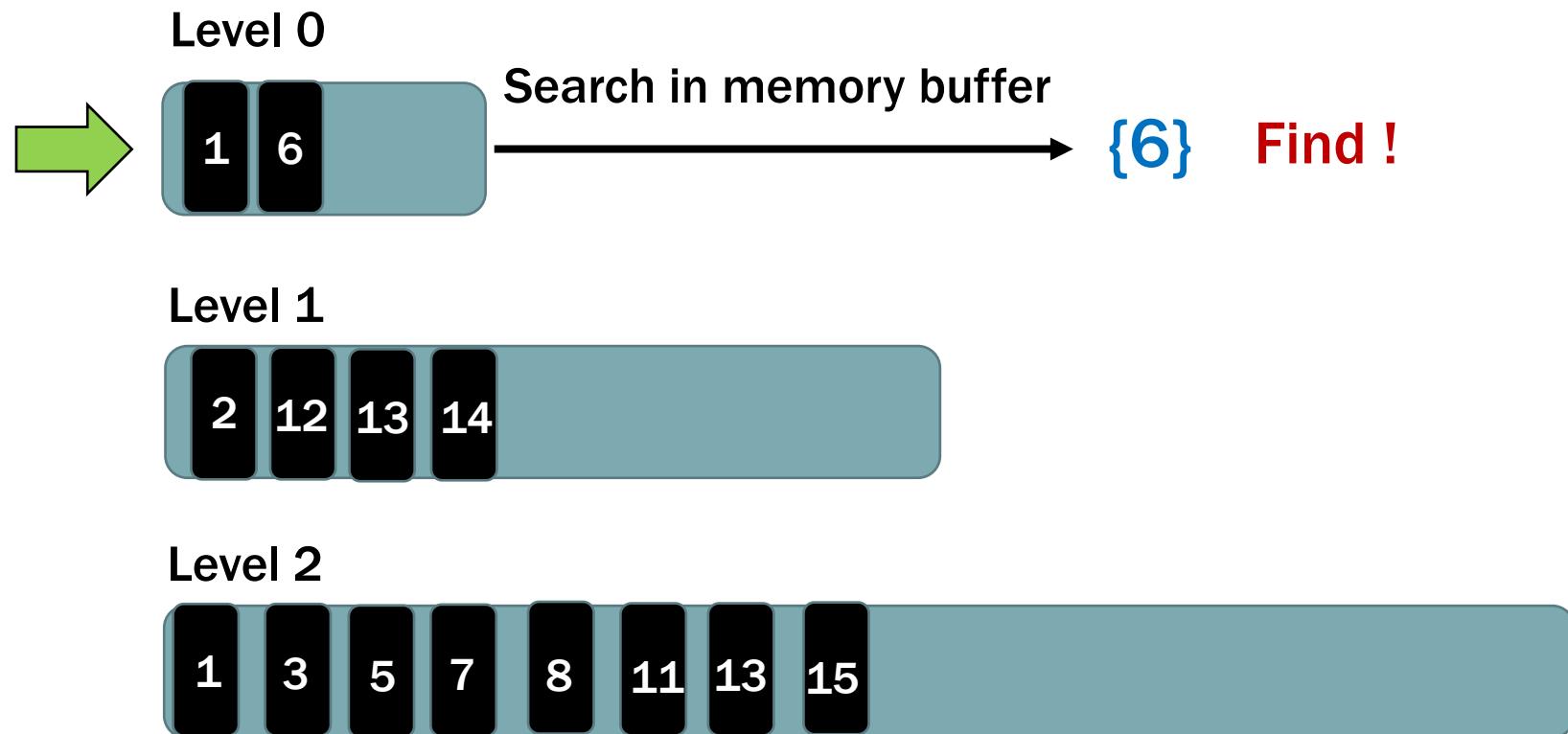
Level 2



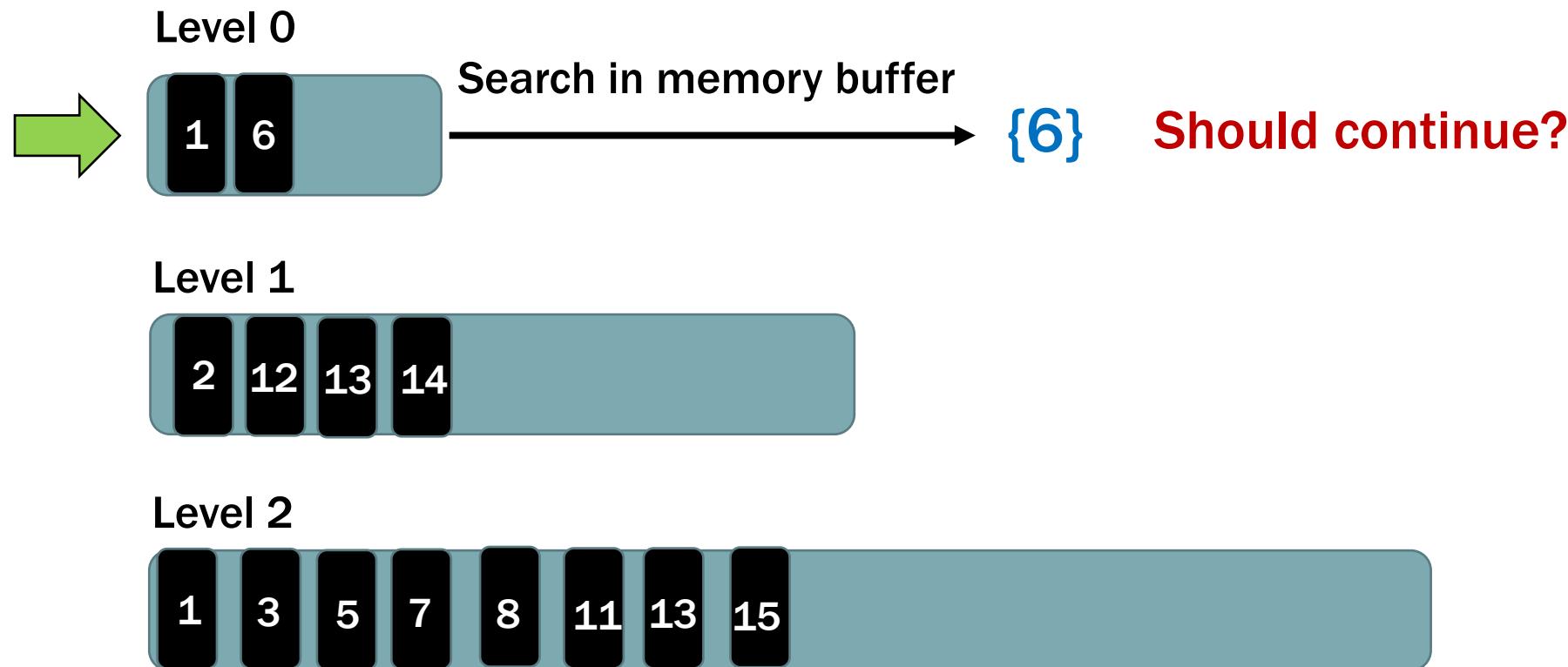
Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9



Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9



Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9



Range-Get([4,9]) → find out **ALL keys** larger than 4 and smaller than 9

Level 0



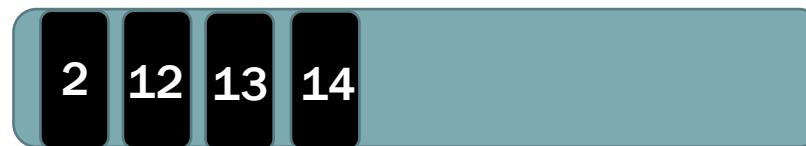
Search in memory buffer

{6}

Should continue?

Yes!

Level 1

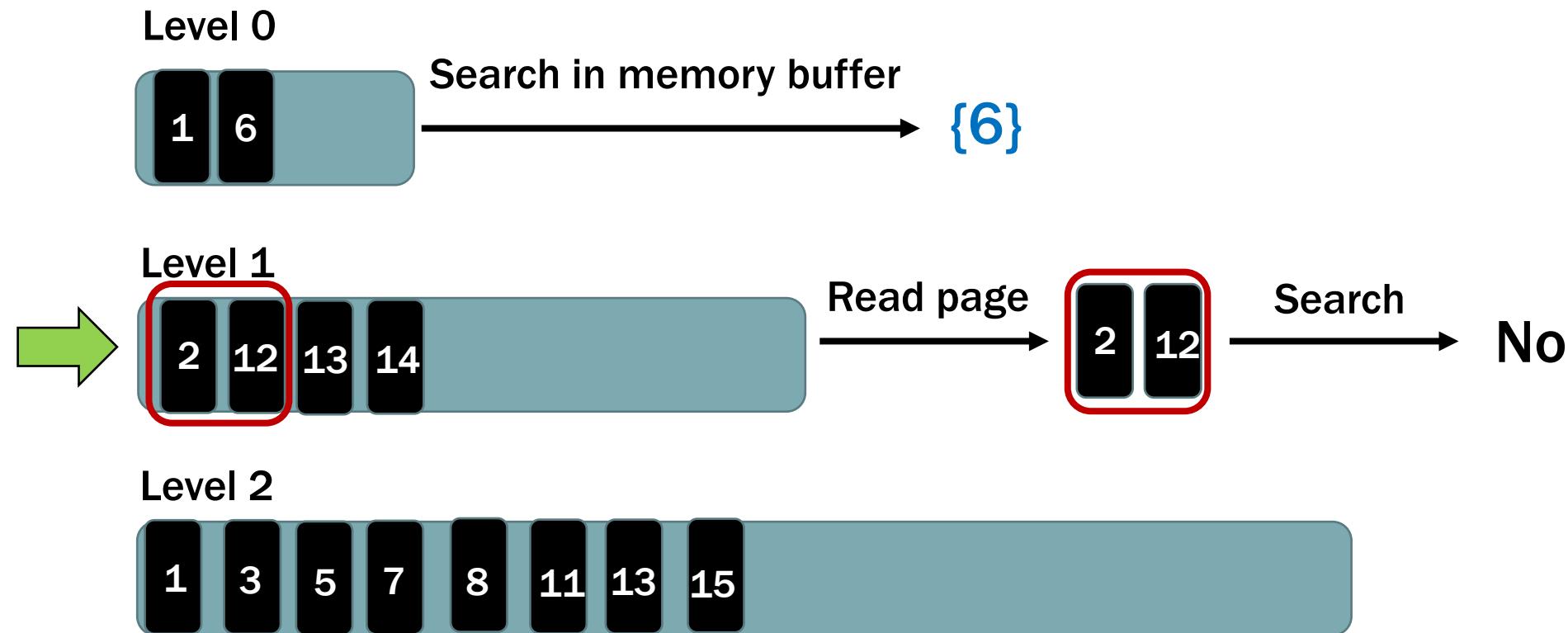


Note: This is different from
the Get function.

Level 2

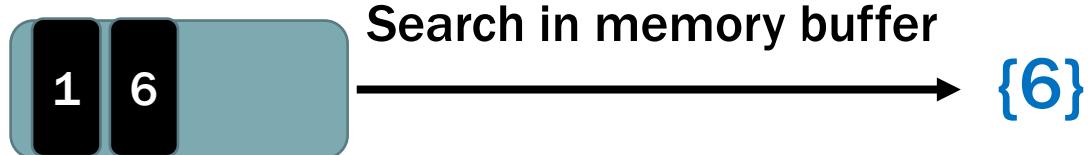


Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9



Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9

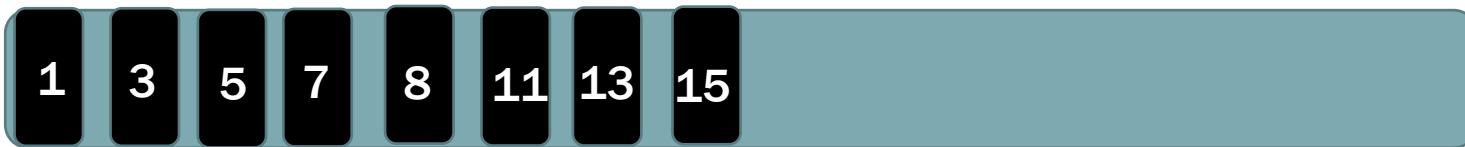
Level 0



Level 1

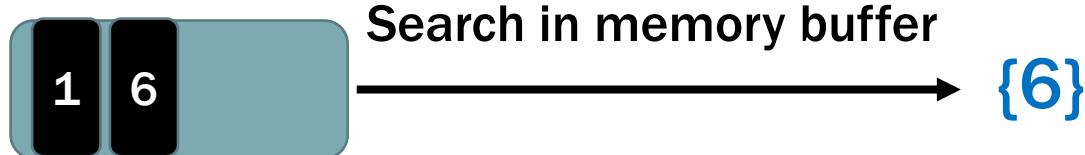


Level 2



Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9

Level 0



Level 1



Level 2



Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9

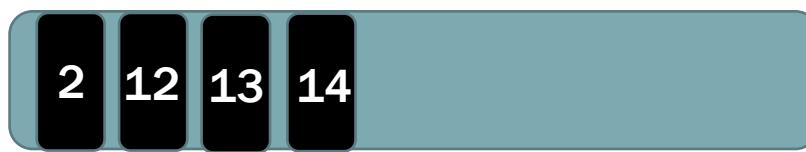
Level 0



Search in memory buffer

{6}

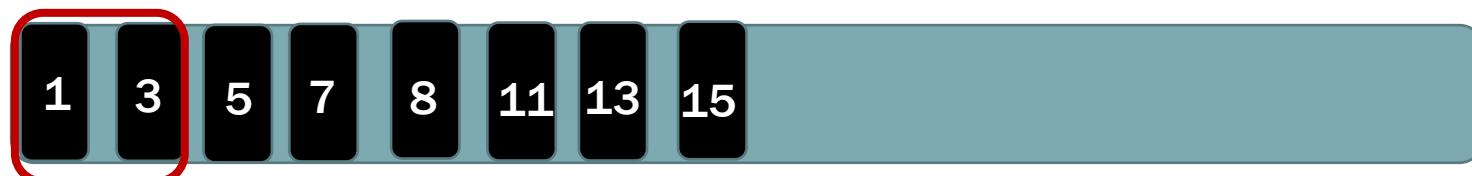
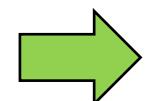
Level 1



Search

No

Level 2



Read page



Search

No

Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9

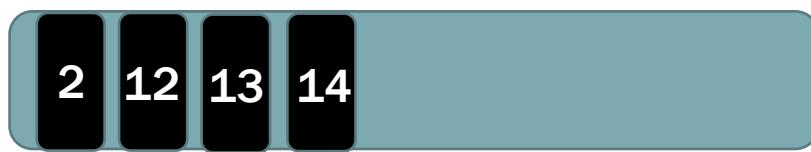
Level 0



Search in memory buffer

{6}

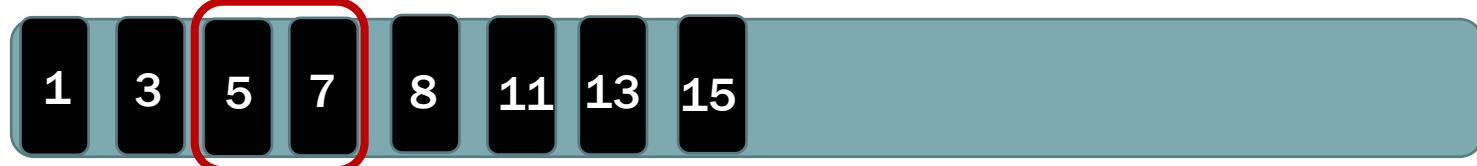
Level 1



Search

No

Level 2



Read page



Search

{5,7}

Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9

Level 0



Search in memory buffer

{6}

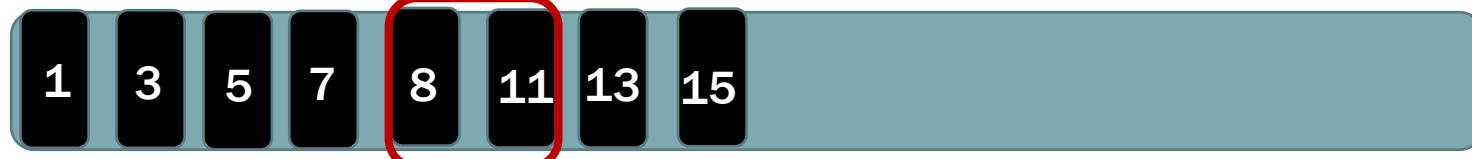
Level 1



Search

No

Level 2



Read page



Search

{8}

Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9

Level 0



Search in memory buffer

{6}

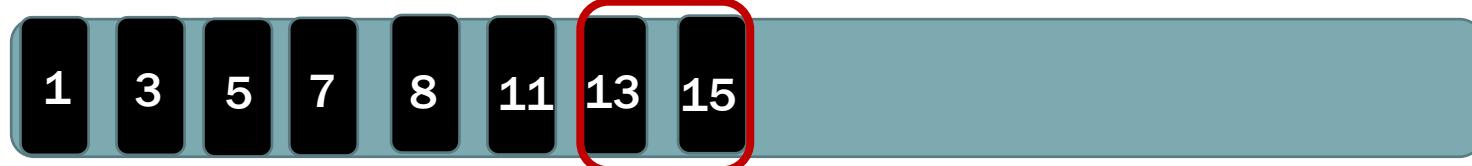
Level 1



Search

No

Level 2



Read page



Search

No

Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9

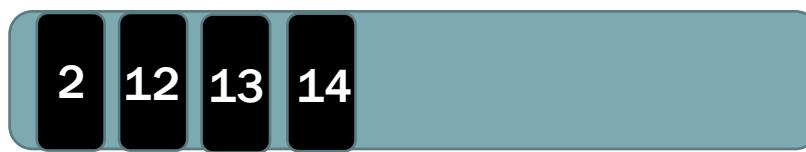
Level 0



Search in memory buffer

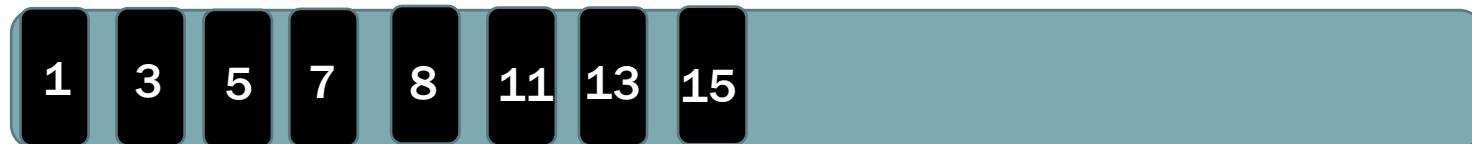
{6}

Level 1



Search → No

Level 2



Search → {5,7,8}

Range-Get([4,9]) → find out **ALL** keys larger than 4 and smaller than 9

Level 0



Level 1



Level 2



Range-Get([4,9]) → {5,6,7,8}

Level 0



Level 1



Level 2



THE COST FOR RANGE_GET([4,9])

Range-Get([4,9]) → {5,6,7,8}

All the existing data in the LSM-tree should be retrieved which introduces huge I/O cost.

Level 0



No I/O (because in main memory)

Level 1



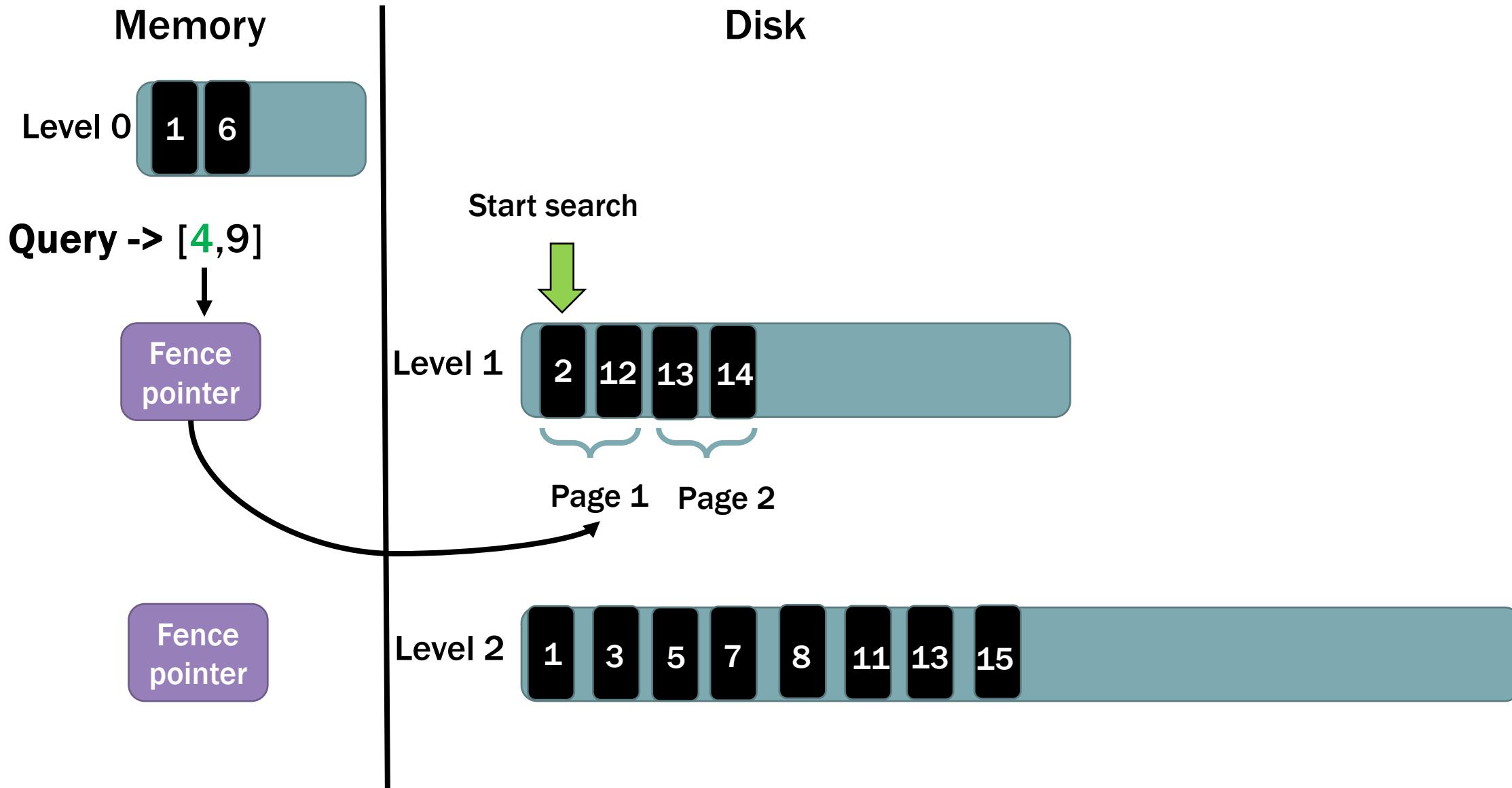
2 I/Os

Level 2

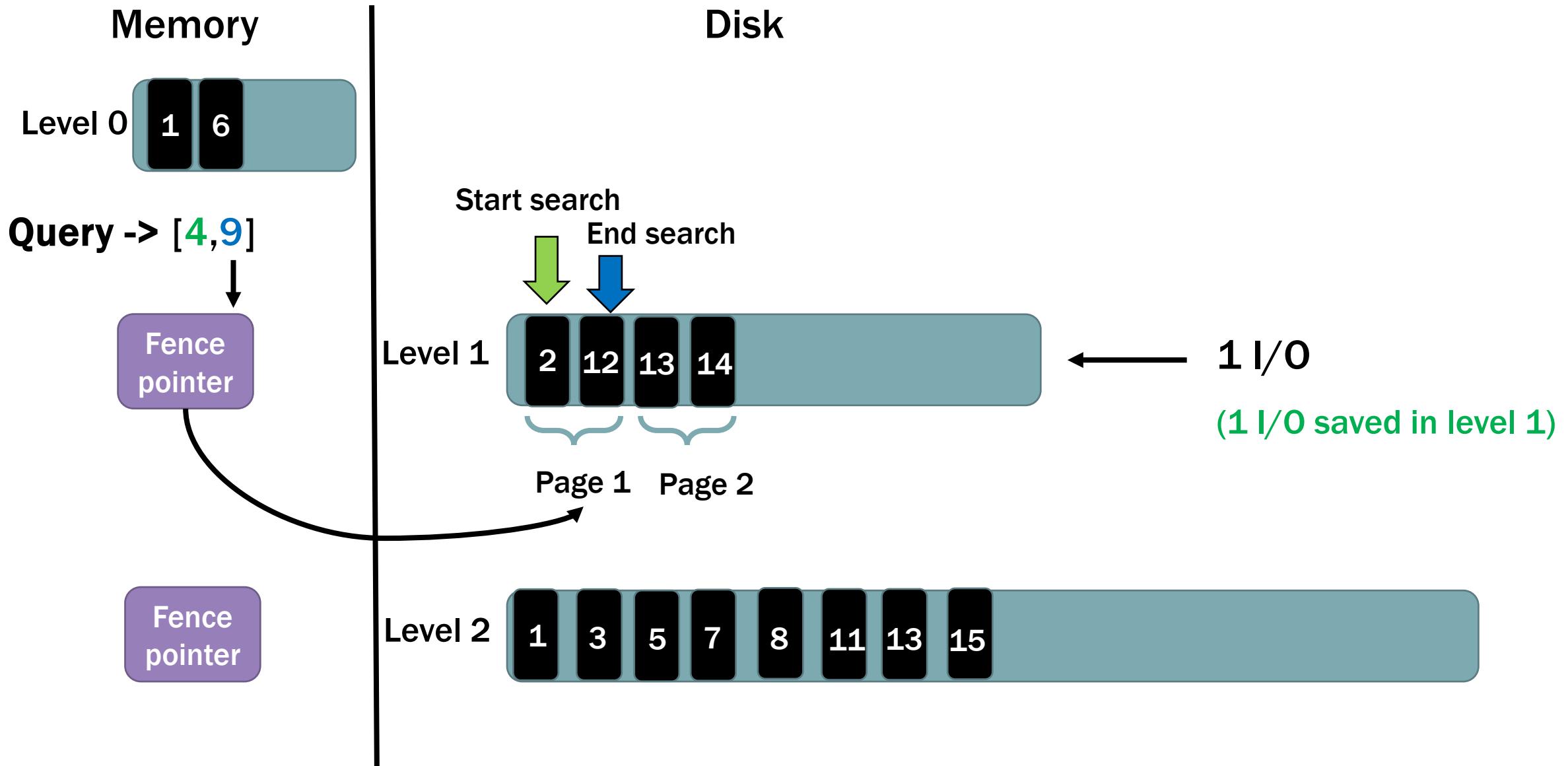


4 I/Os

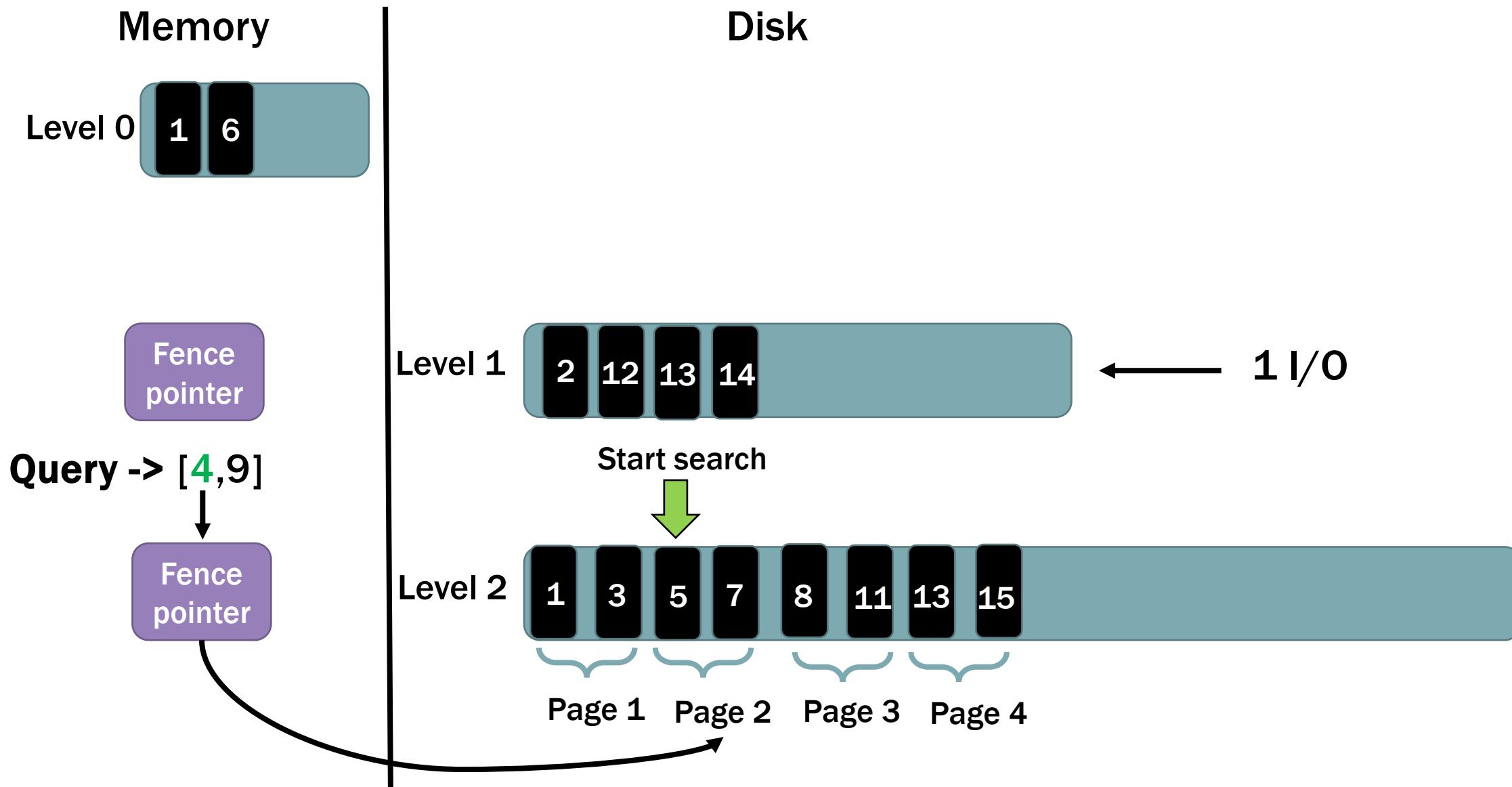
OPTIMIZATION – FENCE POINTERS



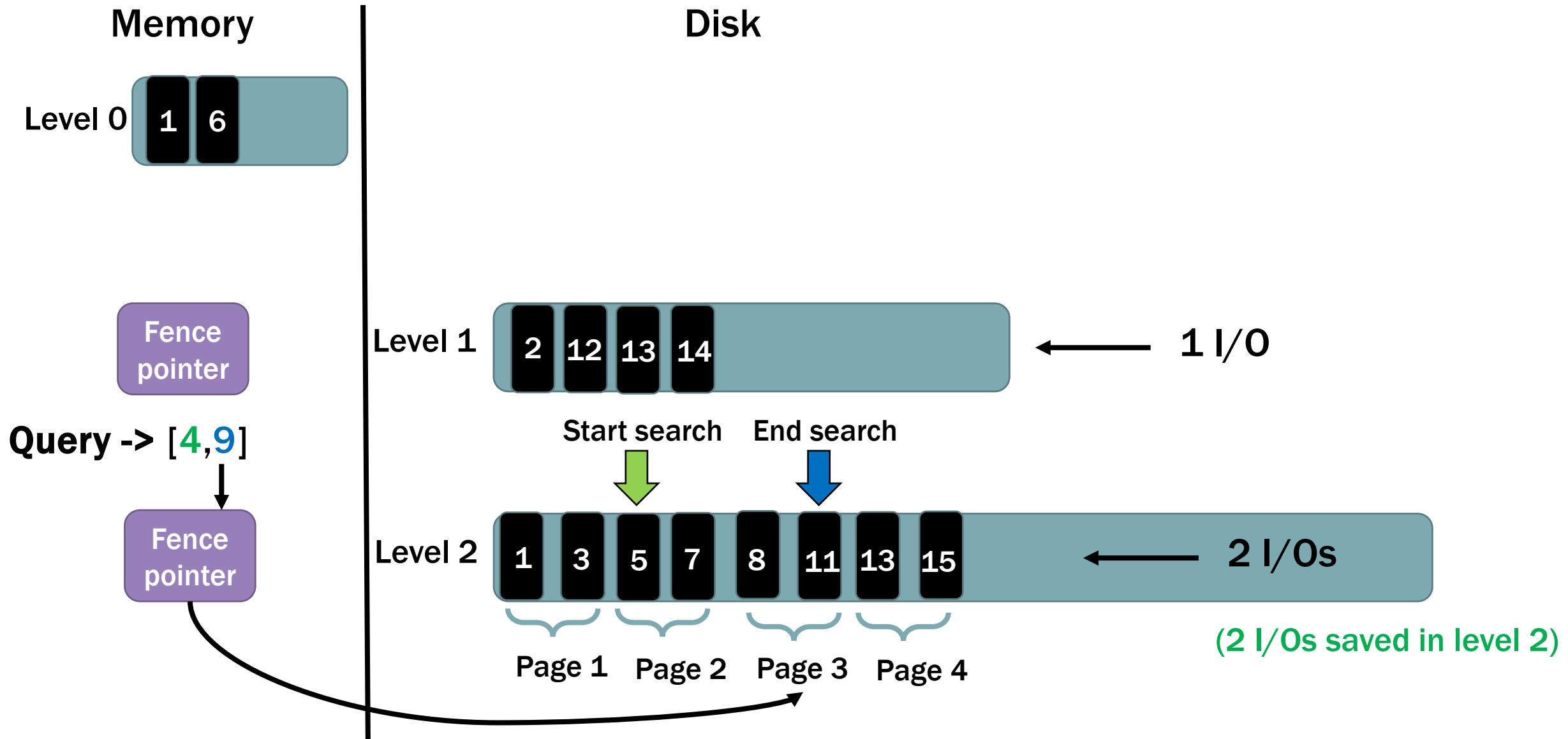
OPTIMIZATION – FENCE POINTERS



OPTIMIZATION – FENCE POINTERS



OPTIMIZATION – FENCE POINTERS



OPTIMIZATION – RANGE FILTERS (NOT EXAMINABLE)

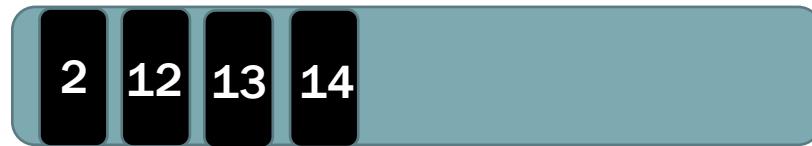
Consider:

Can we introduce a filter for Range_Get like GET function to reduce I/O cost?

Level 0



Level 1



Level 2



OPTIMIZATION – RANGE FILTERS

Consider:

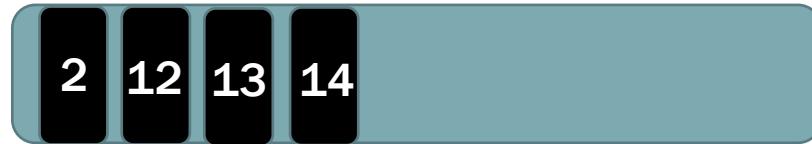
Can we introduce a filter for Range_Get like GET function to reduce I/O cost?

Level 0



Sure ! Let's see how does it work.

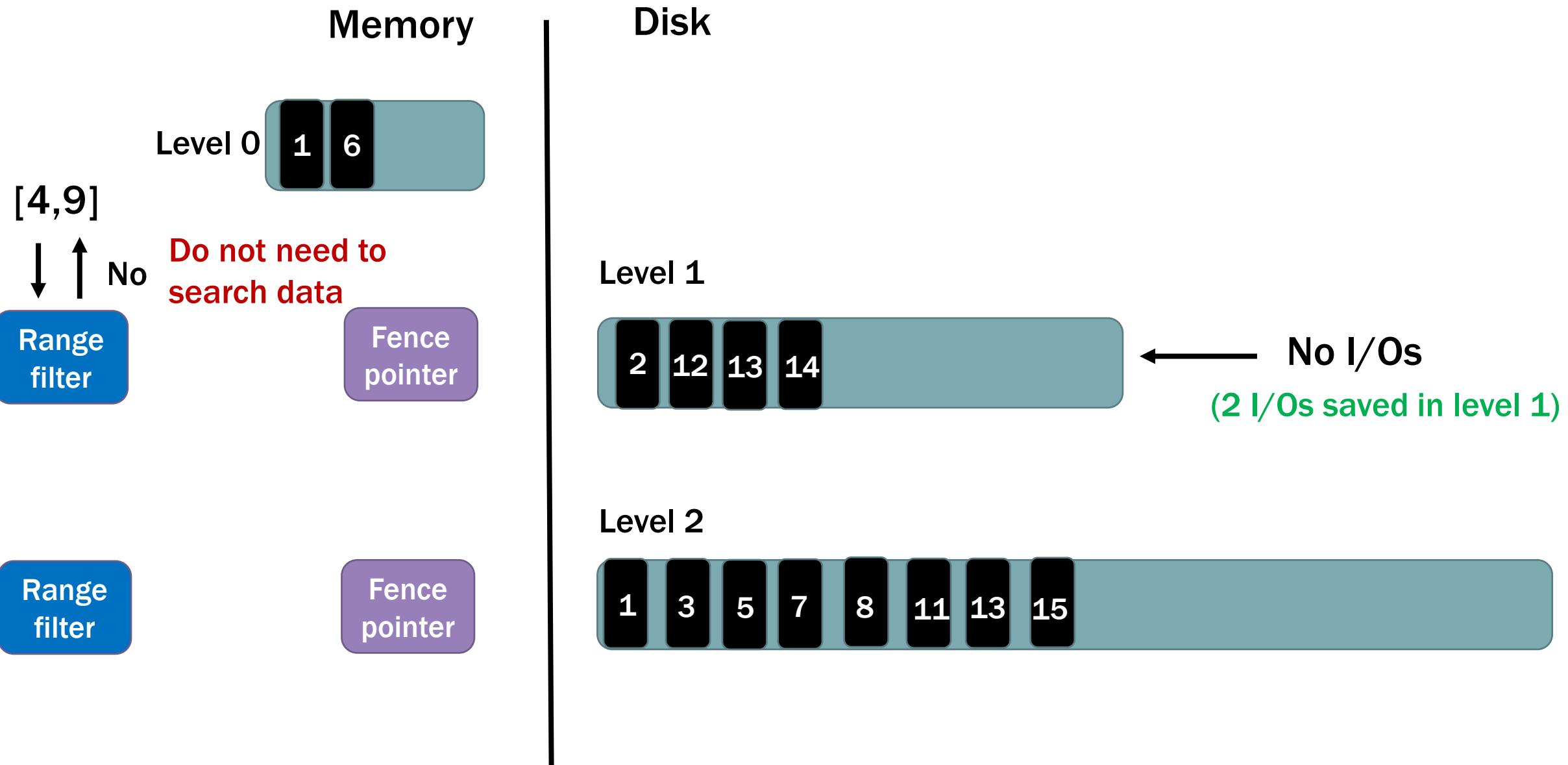
Level 1



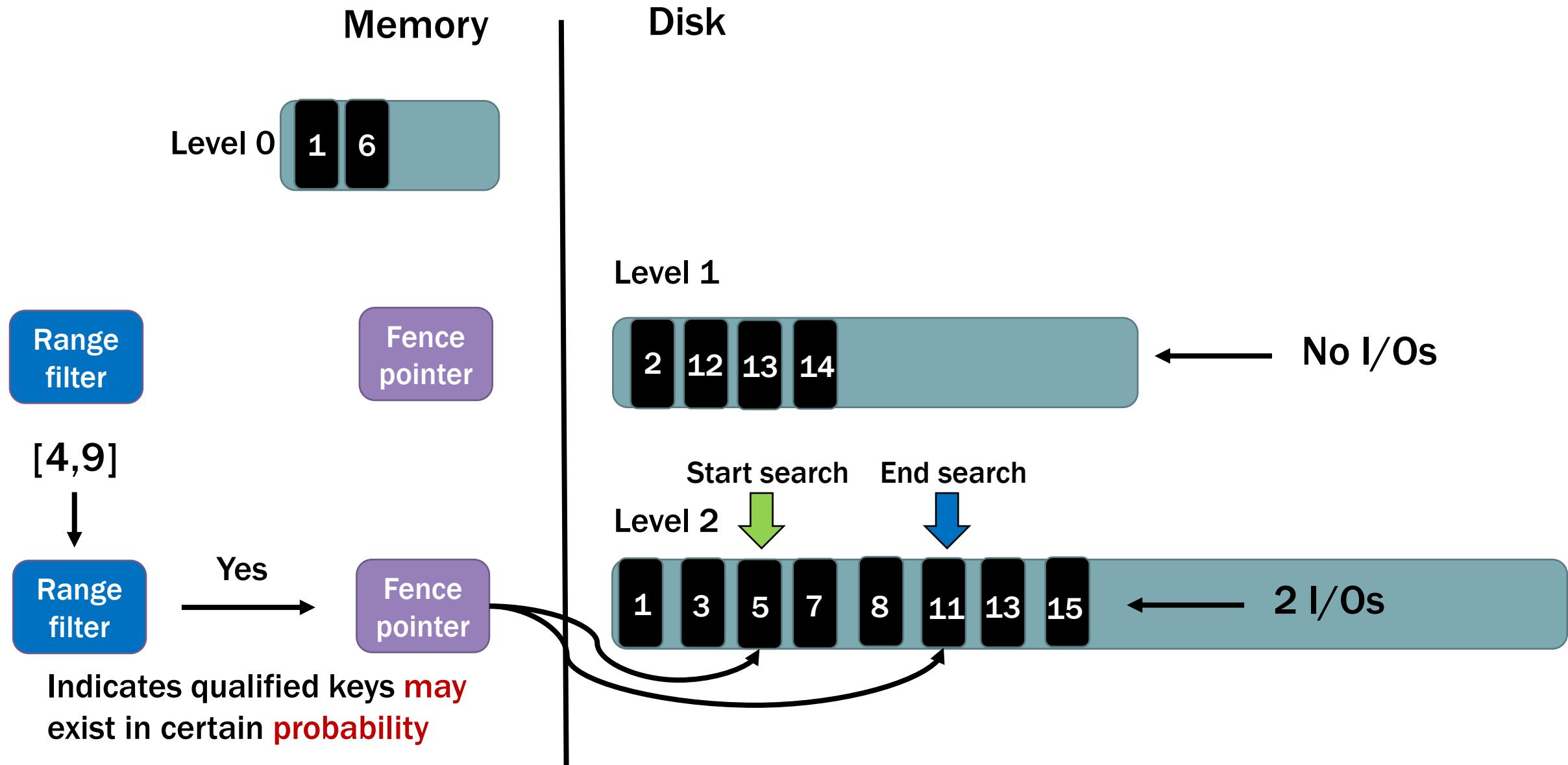
Level 2



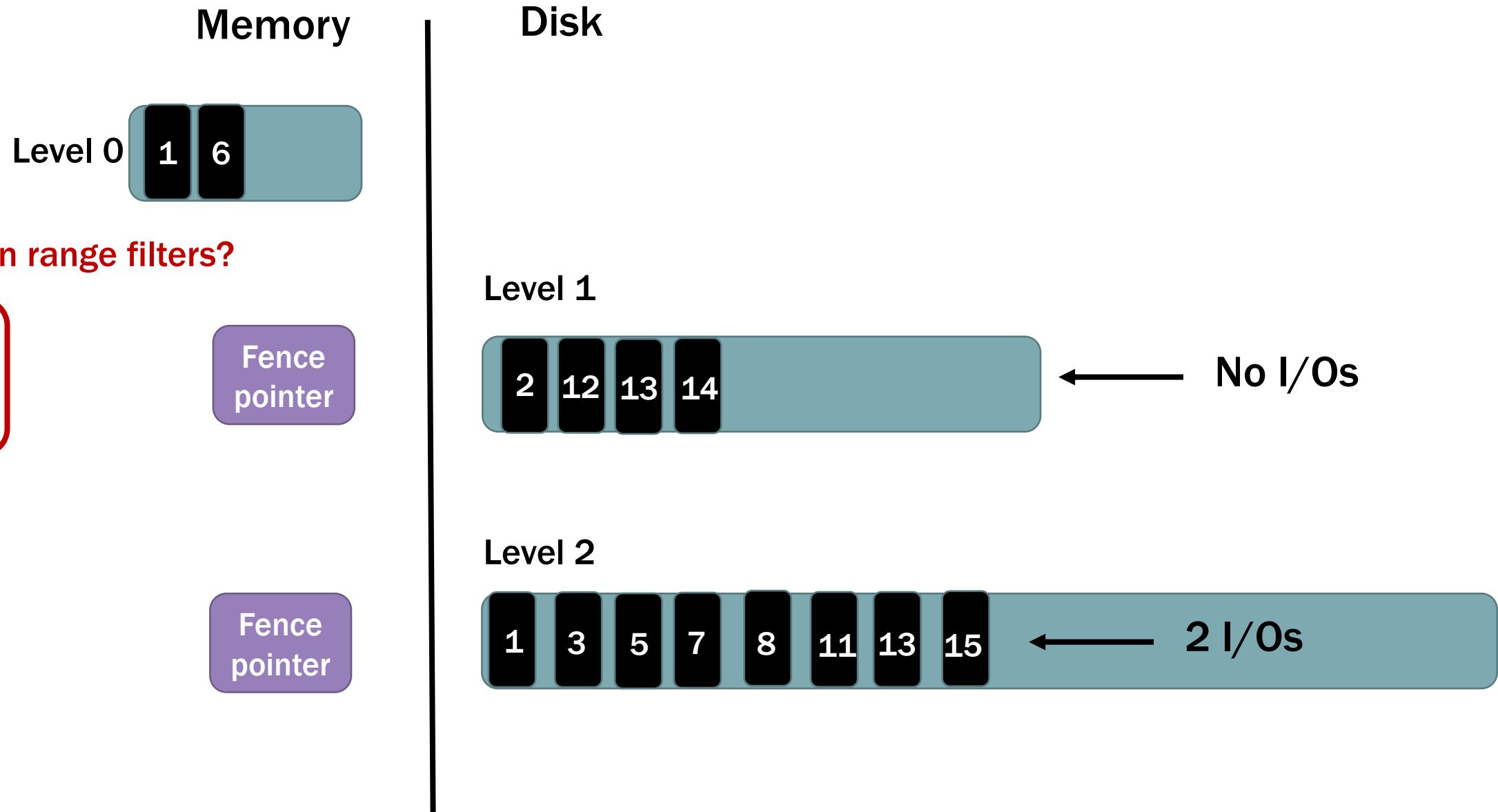
OPTIMIZATION – RANGE FILTERS



OPTIMIZATION – RANGE FILTERS

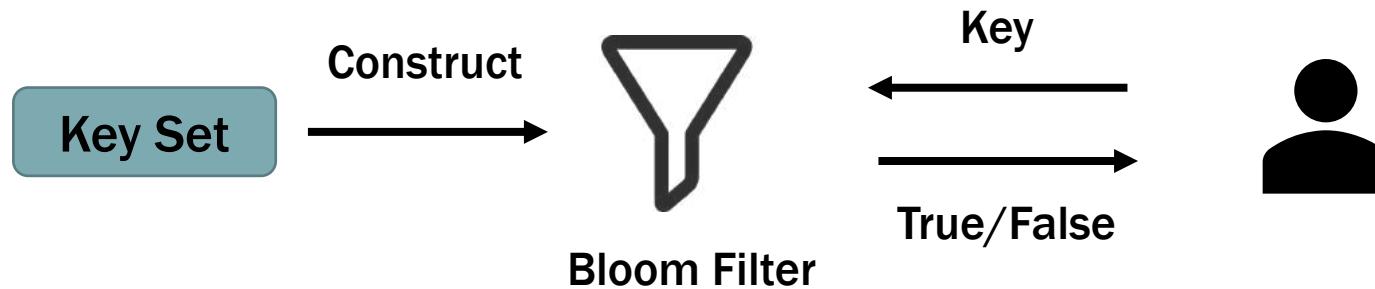


OPTIMIZATION – RANGE FILTERS



RANGE FILTER – PREFIX BLOOM FILTER

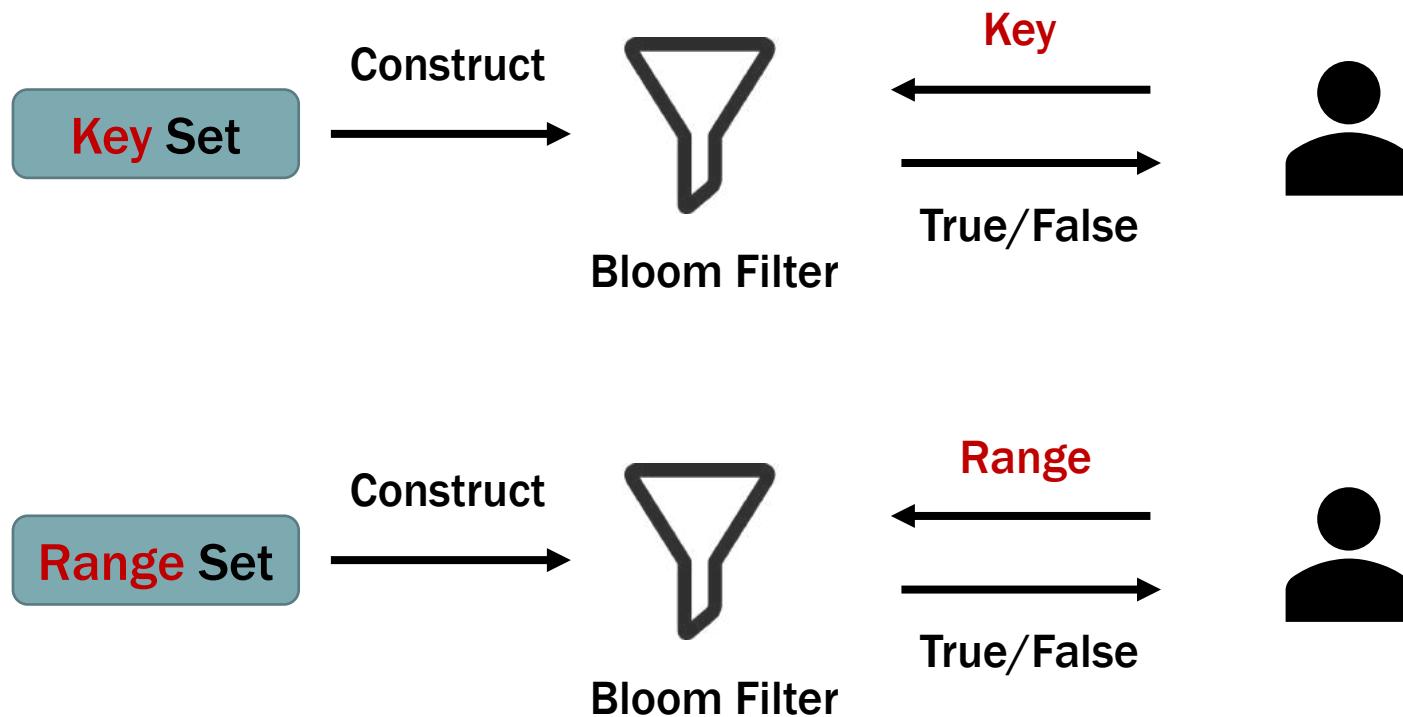
Revisit our old friend: Bloom filter



Can we extend to range query?

RANGE FILTER – PREFIX BLOOM FILTER

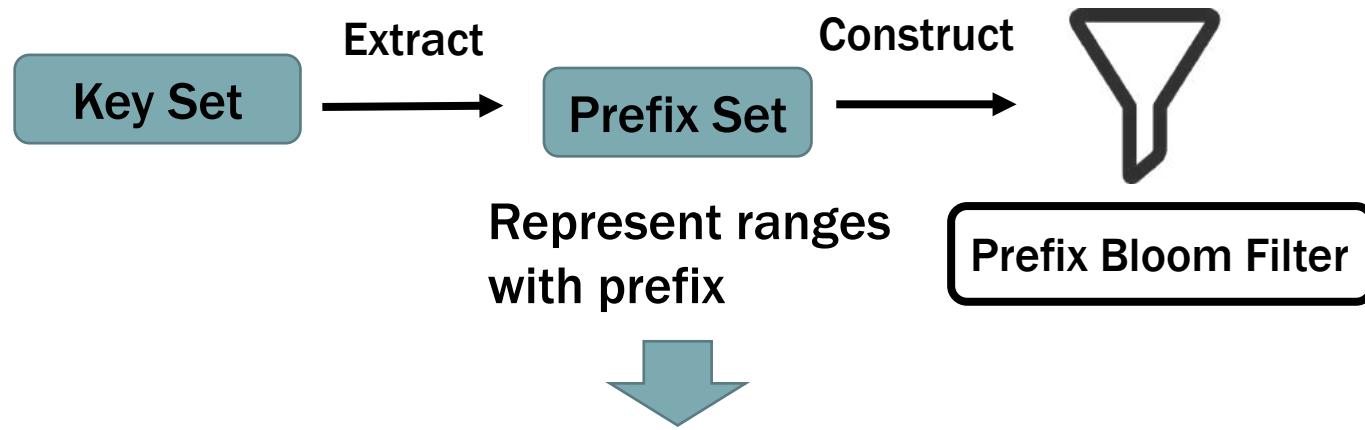
Extend Bloom filter to facilitate range query



RANGE FILTER – PREFIX BLOOM FILTER



RANGE FILTER – PREFIX BLOOM FILTER

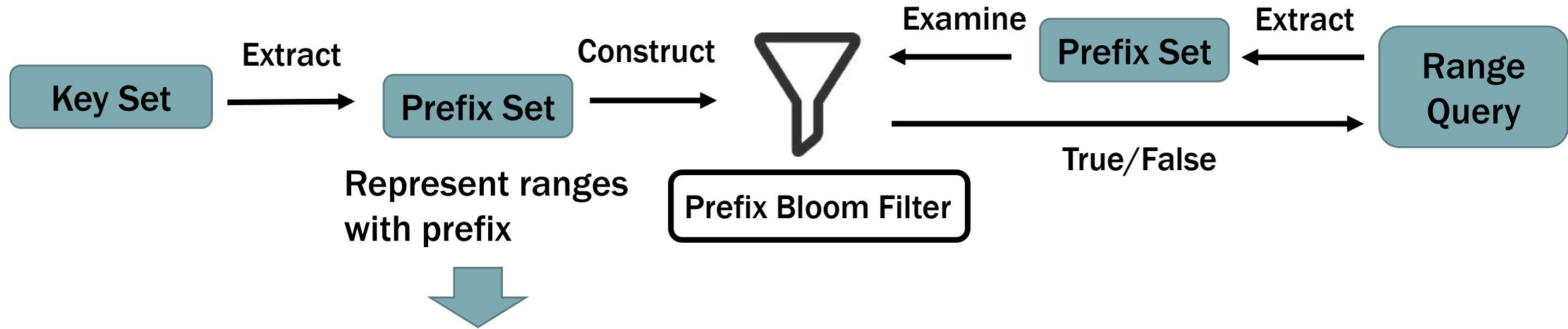


Example: Assume the key space is [0-15] and prefix length is 2

6 -> 0110
↑
prefix

prefix	range
00	[0-3]
01	[4-7]
10	[8-11]
11	[12-15]

RANGE FILTER – PREFIX BLOOM FILTER



Example: Assume the key space is [0-15] and prefix length is 2

6 -> 0110
↑
prefix

prefix	range
00	[0-3]
01	[4-7]
10	[8-11]
11	[12-15]

RANGE FILTER – PREFIX BLOOM FILTER

Example: construct prefix bloom filter for level 1

Level 1



Bloom filter info.

$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

RANGE FILTER – PREFIX BLOOM FILTER

Example: construct prefix bloom filter for level 1

Level 1



$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

2 -> 0010

Extract
prefix

00(0)

12 -> 1100



11(3)

13 -> 1101

14 -> 1110

RANGE FILTER – PREFIX BLOOM FILTER

Example: construct prefix bloom filter for level 1

Level 1



$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

2 -> 0010

12 -> 1100

13 -> 1101

14 -> 1110

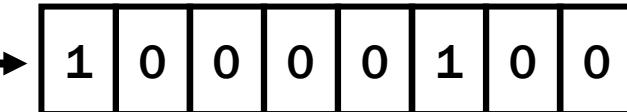
Extract
prefix

00(0)

11(3)

$$h_1(0) = 0$$

$$h_2(0) = 5$$



RANGE FILTER – PREFIX BLOOM FILTER

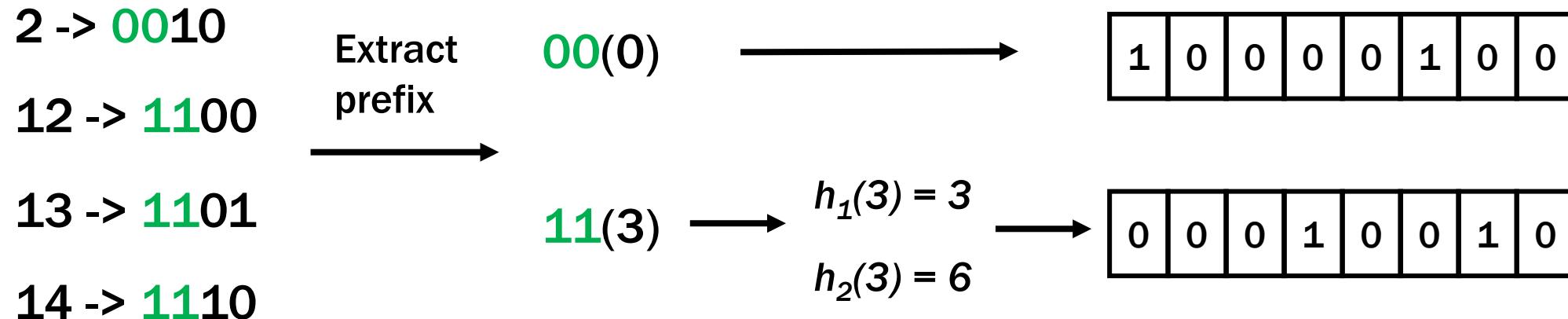
Example: construct prefix bloom filter for level 1

Level 1

2	12	13	14
---	----	----	----

$$h_1(x) = x \bmod 8$$

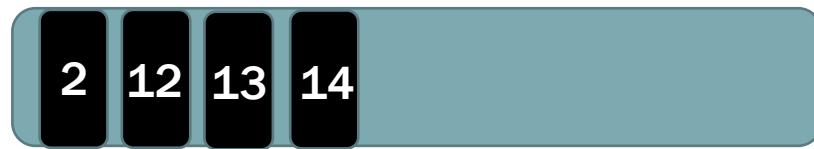
$$h_2(x) = (3x+5) \bmod 8$$



RANGE FILTER – PREFIX BLOOM FILTER

Response to the range query

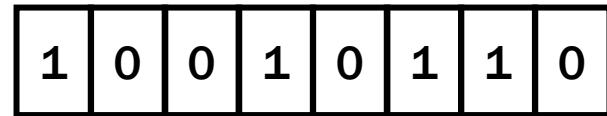
Level 1



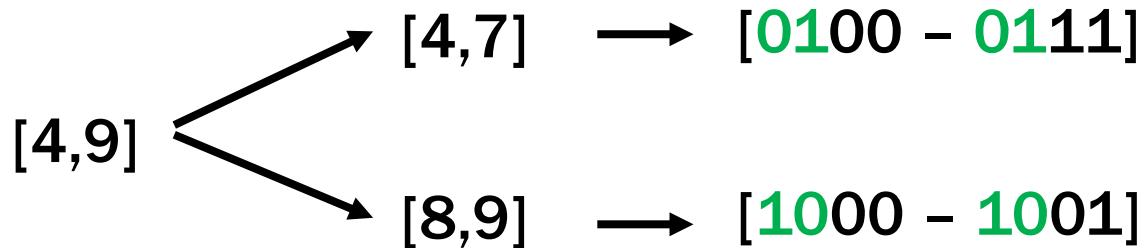
$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

Prefix Bloom Filter



Query -> [4,9]



RANGE FILTER – PREFIX BLOOM FILTER

Response to the range query

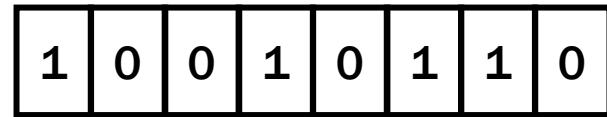
Level 1



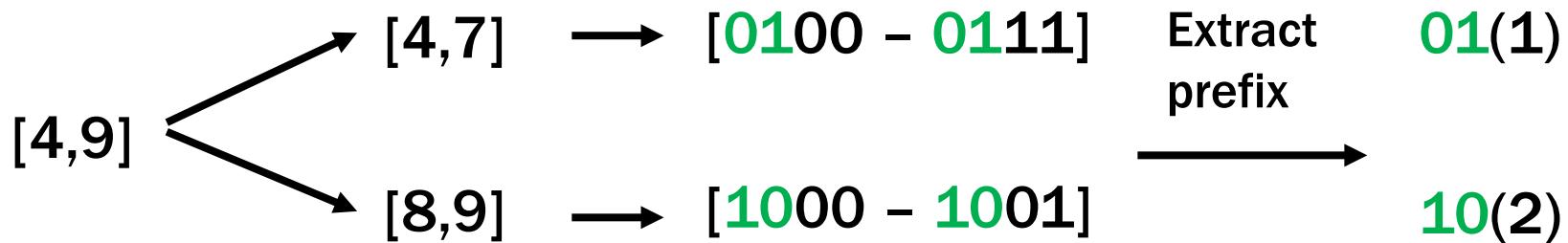
$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

Prefix Bloom Filter

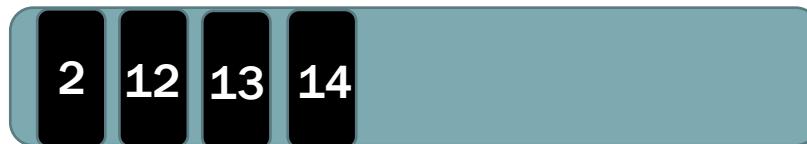


Query -> [4,9]



RANGE FILTER – PREFIX BLOOM FILTER

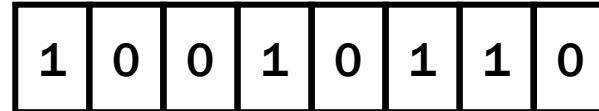
Level 1



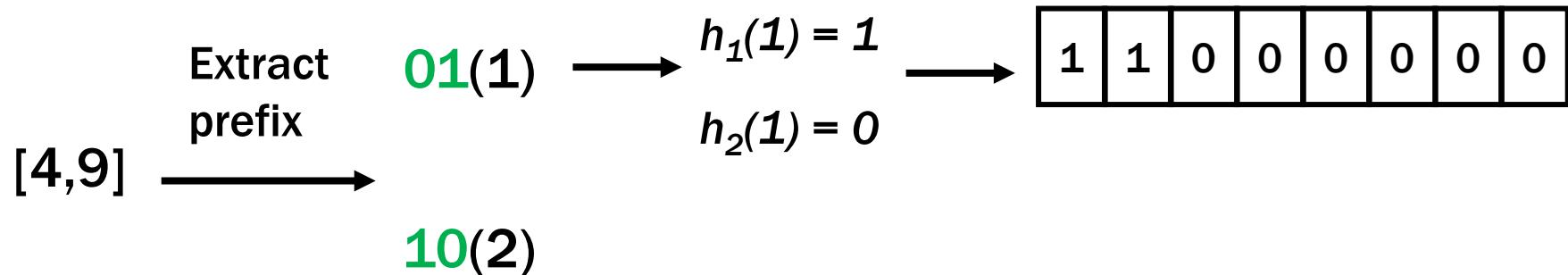
$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

Prefix Bloom Filter



Query $\rightarrow [4, 9]$



RANGE FILTER – PREFIX BLOOM FILTER

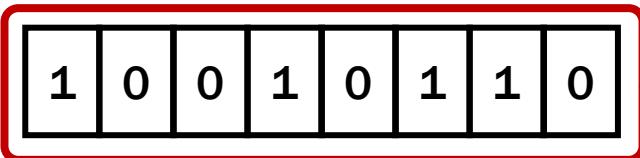
Level 1



$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

Prefix Bloom Filter

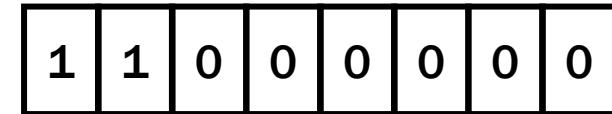


Query $\rightarrow [4, 9]$

Extract
prefix
 $[4, 9] \longrightarrow 01(1)$

$$\begin{aligned} h_1(1) &= 1 \\ h_2(1) &= 0 \end{aligned}$$

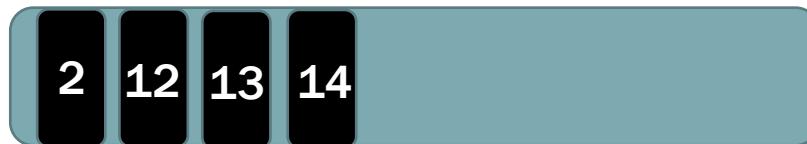
Compare



10(2)

RANGE FILTER – PREFIX BLOOM FILTER

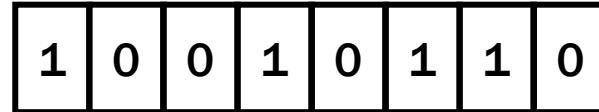
Level 1



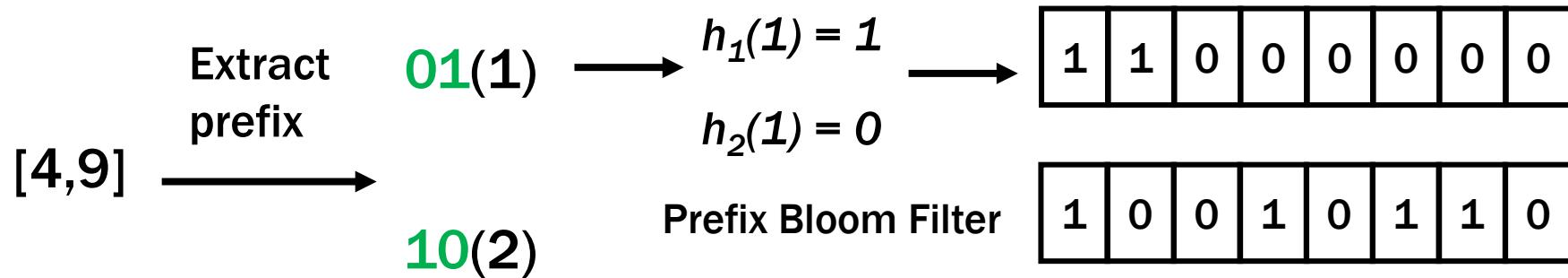
$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

Prefix Bloom Filter



Query $\rightarrow [4, 9]$



If every “1” bit is set to “1” in Prefix Bloom filter:
Response YES
Else:
Response NO

RANGE FILTER – PREFIX BLOOM FILTER

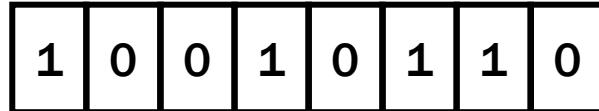
Level 1



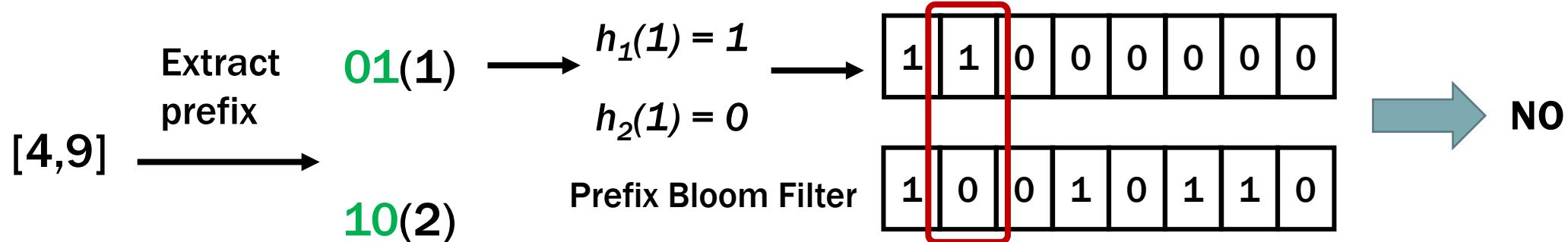
$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

Prefix Bloom Filter



Query $\rightarrow [4, 9]$



If every “1” bit is set to “1” in Prefix Bloom filter:
Response YES

Else:

Response NO

RANGE FILTER – PREFIX BLOOM FILTER

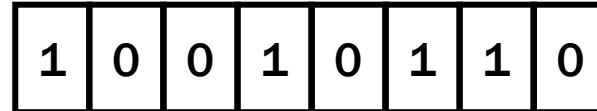
Level 1



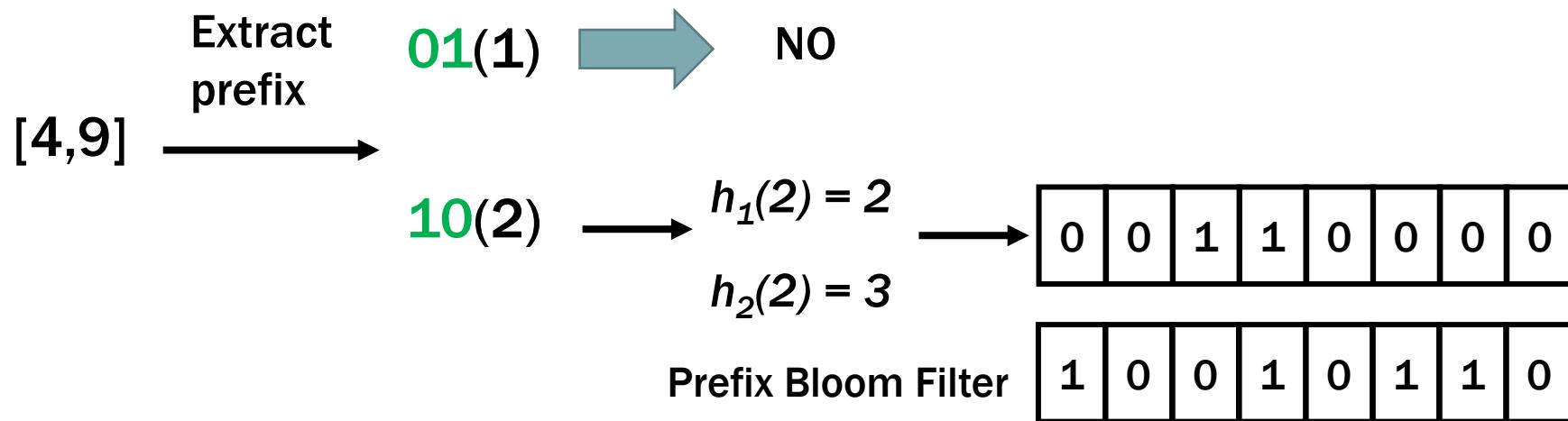
$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

Prefix Bloom Filter

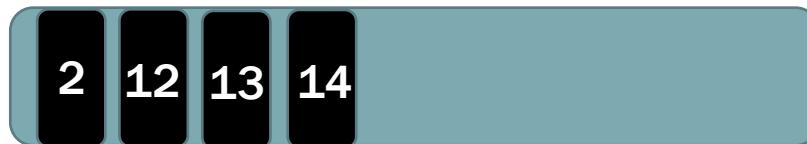


Query $\rightarrow [4, 9]$



RANGE FILTER – PREFIX BLOOM FILTER

Level 1



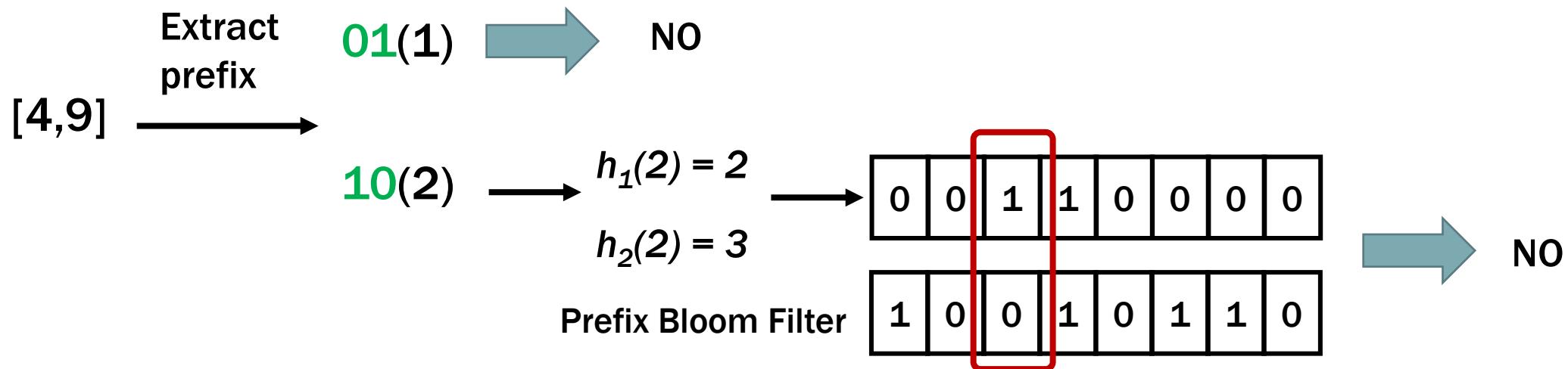
$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

Prefix Bloom Filter

1	0	0	1	0	1	1	0
---	---	---	---	---	---	---	---

Query $\rightarrow [4, 9]$



RANGE FILTER – PREFIX BLOOM FILTER

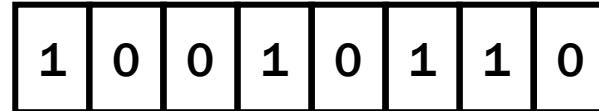
Level 1



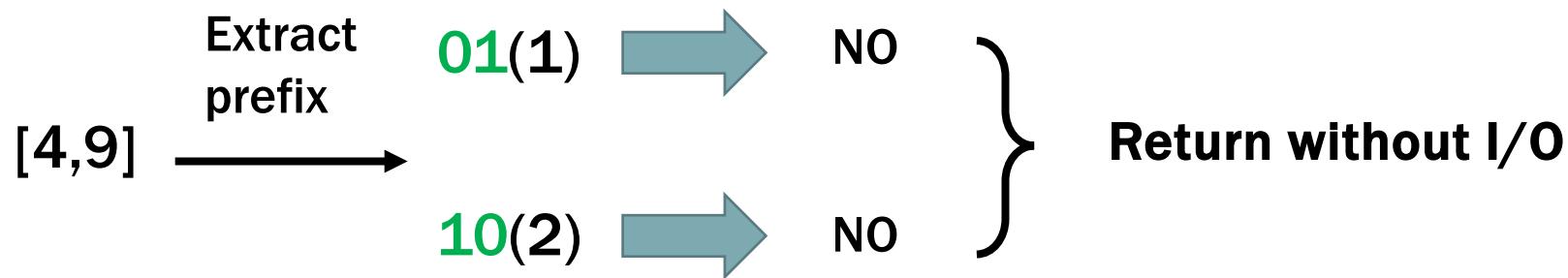
$$h_1(x) = x \bmod 8$$

$$h_2(x) = (3x+5) \bmod 8$$

Prefix Bloom Filter

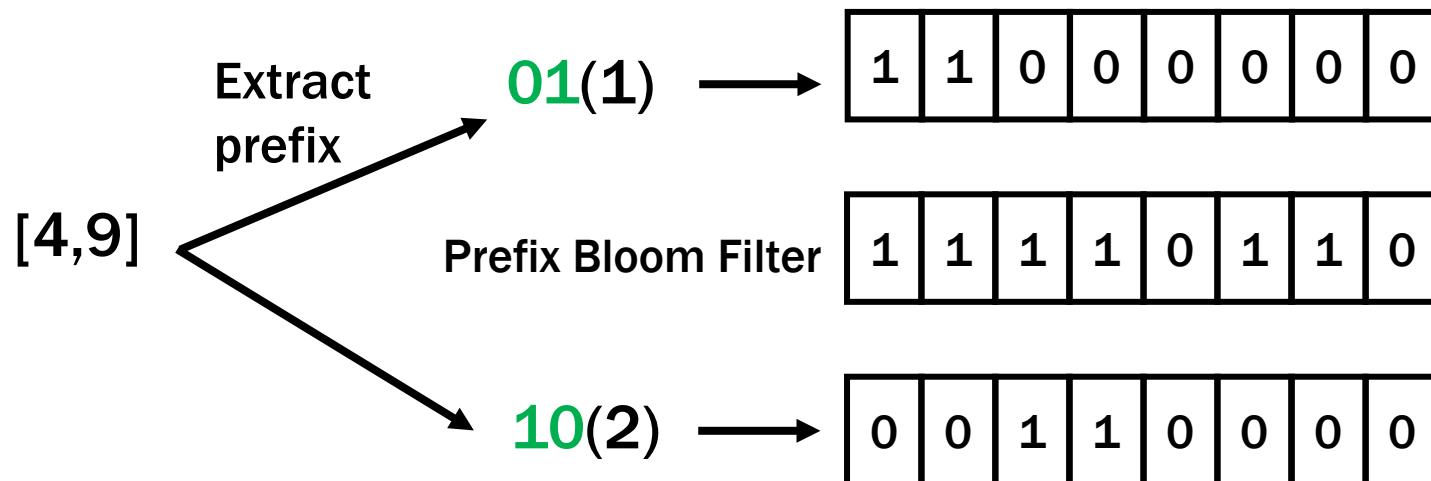
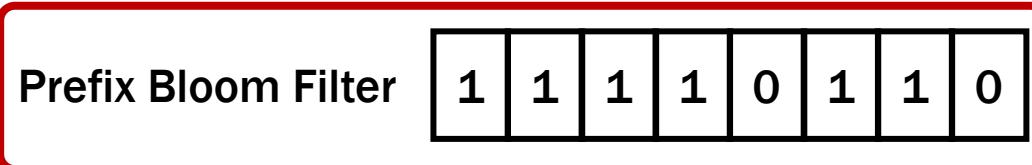


Query $\rightarrow [4, 9]$



RANGE FILTER – PREFIX BLOOM FILTER

Level 2

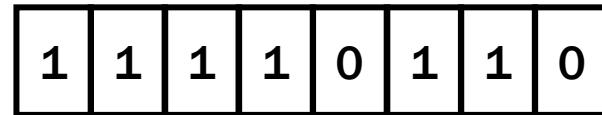


RANGE FILTER – PREFIX BLOOM FILTER

Level 2

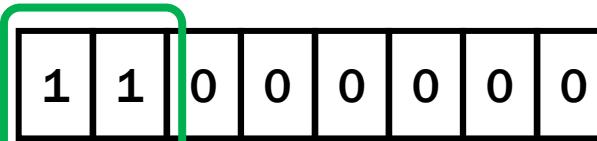


Prefix Bloom Filter

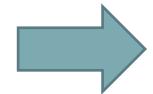


Extract
prefix

01(1)



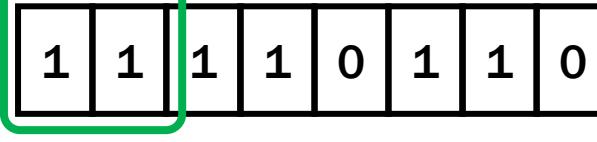
YES



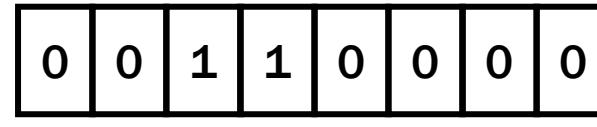
Search data

[4,9]

Prefix Bloom Filter



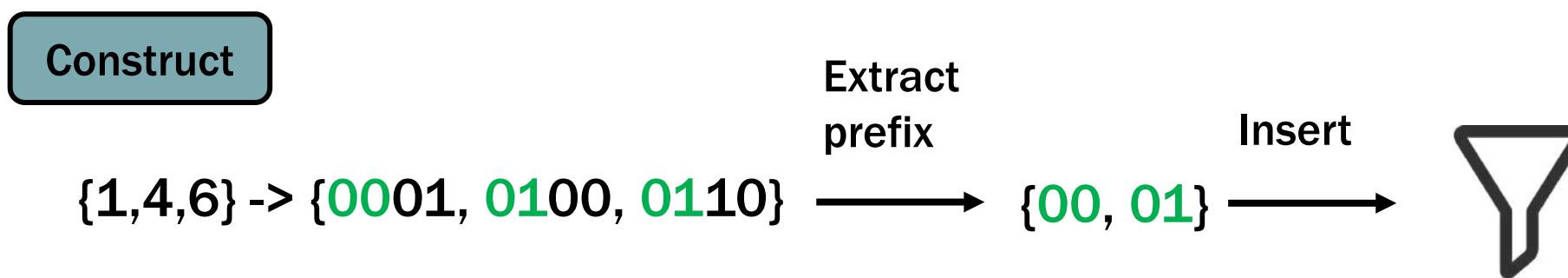
10(2)



RANGE FILTER – PREFIX BLOOM FILTER

Consider:

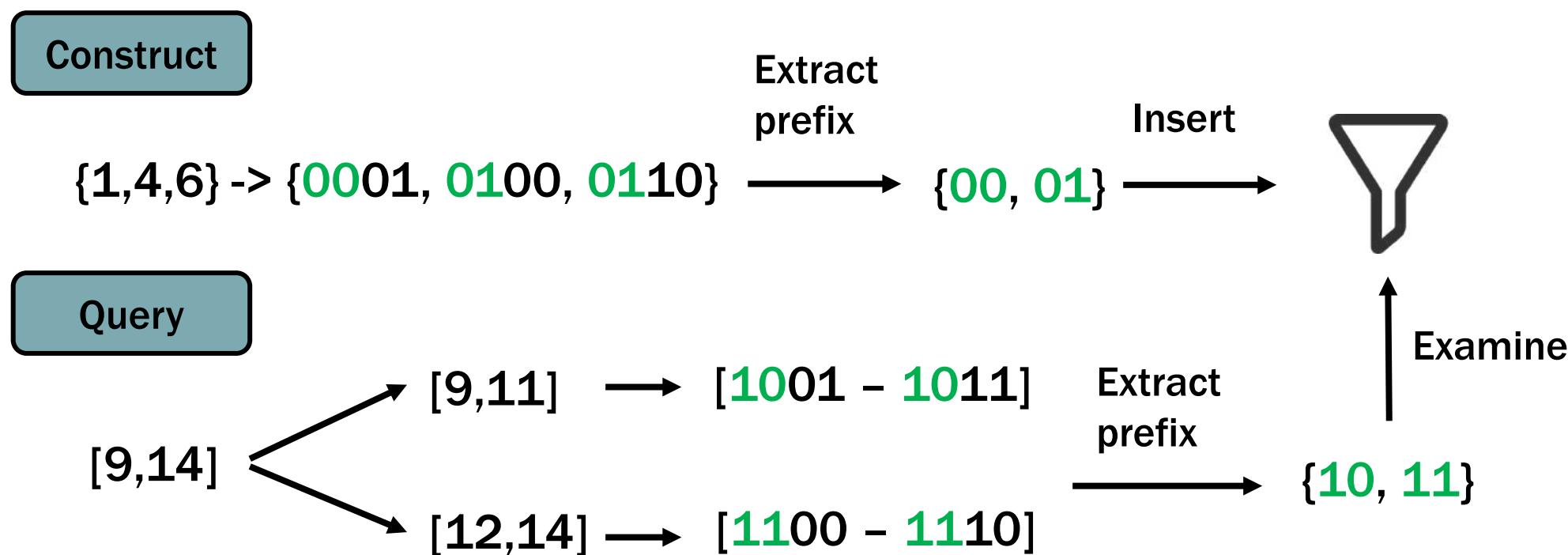
How to construct Prefix Bloom filter with the key set {1,4,6} and query with the range [9-14]?



RANGE FILTER – PREFIX BLOOM FILTER

Consider:

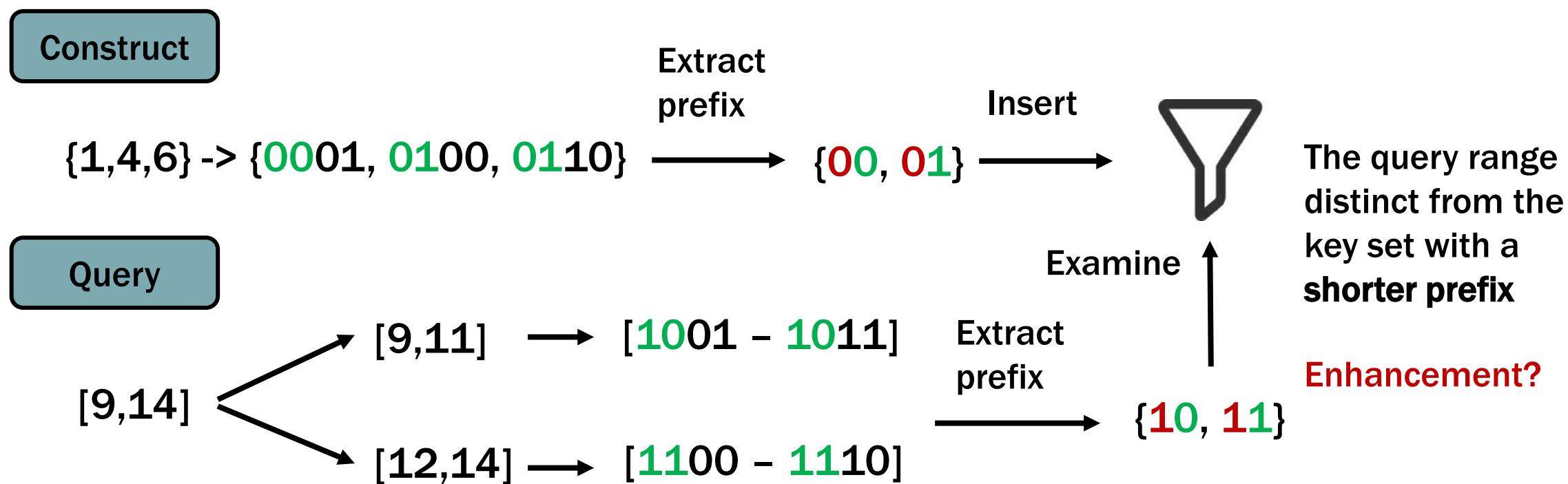
How to construct Prefix Bloom filter with the key set {1,4,6} and query with the range [9-14]?



RANGE FILTER – PREFIX BLOOM FILTER

Consider:

How to construct Prefix Bloom filter with the key set {1,4,6} and query with the range [9-14]?



RANGE FILTER – PREFIX BLOOM FILTER

Idea: use hierarchical prefix Bloom filters to encode different length of prefixes

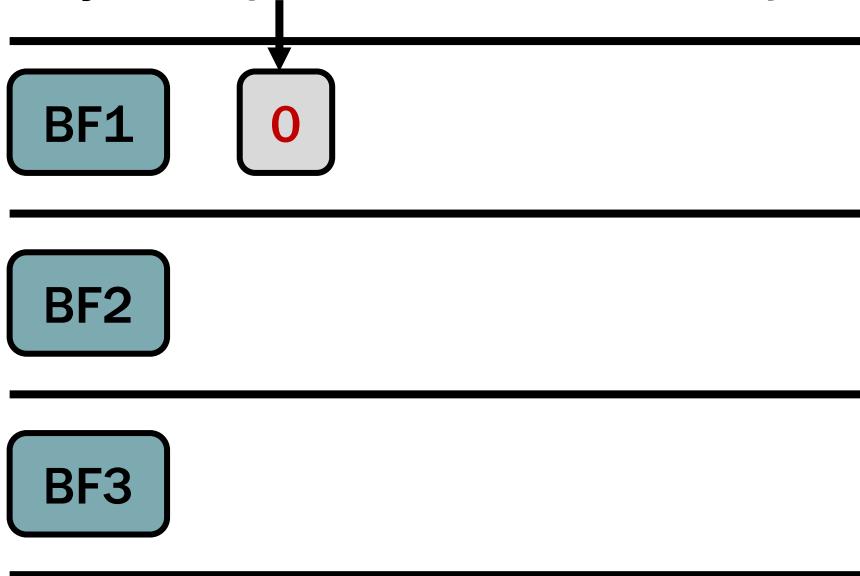
Example: set prefix length from 1 to 3.

RANGE FILTER – PREFIX BLOOM FILTER

Idea: use hierarchical prefix Bloom filters to encode different length of prefixes

Example: set prefix length from 1 to 3.

Key set: {0001, 0100, 0110}



For BF1:

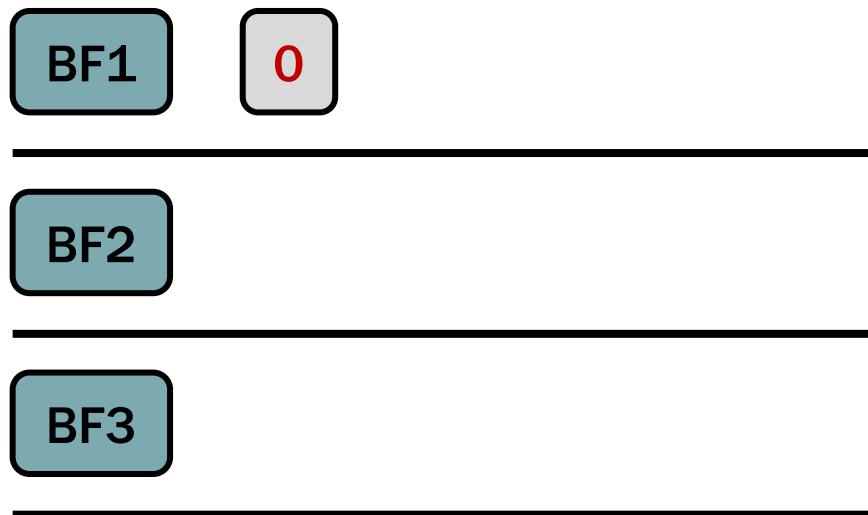
0 -> [0000,0111] Range size = 8

For clarity, the elements
inserted into a BF are listed
instead of the actual BF

RANGE FILTER – PREFIX BLOOM FILTER

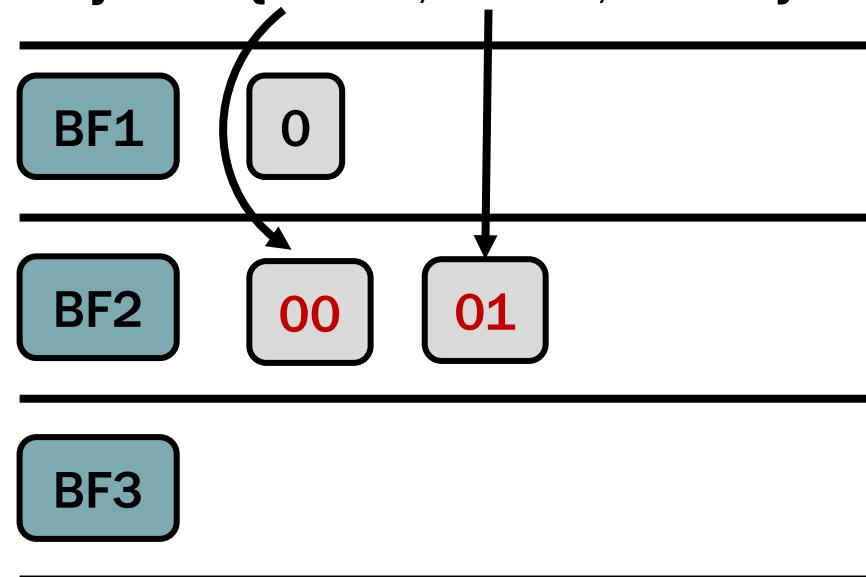
Example: set prefix length from 1 to 3.

Key set: {0001, 0100, 0110}



Each element represents
smaller range compared to BF1

Key set: {0001, 0100, 0110}



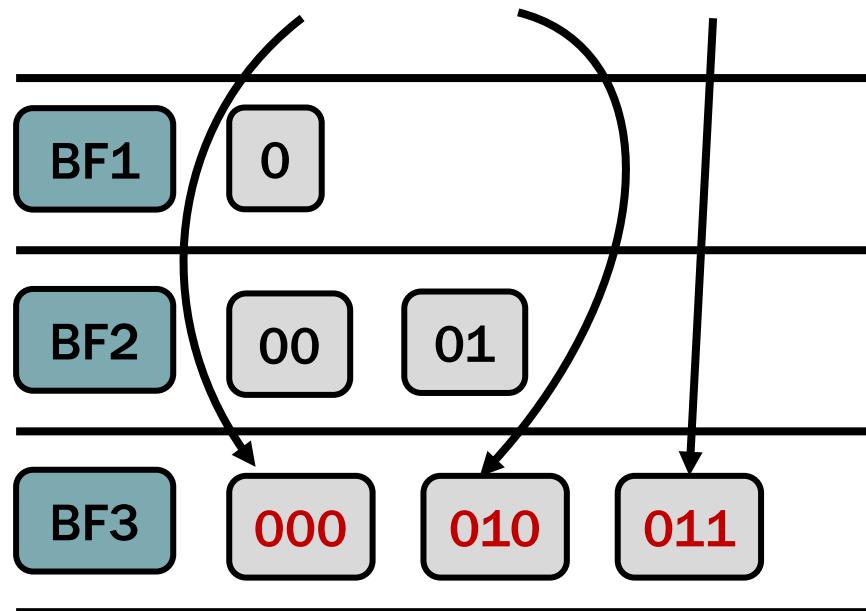
00 -> [0000,0011] Range size = 4

01 -> [0100,0111]

RANGE FILTER – PREFIX BLOOM FILTER

Example: set prefix length from 1 to 3.

Key set: {**0001**, **0100**, **0110**}



000	-> [0000,0001]	Range size = 2
010	-> [0100,0101]	
011	-> [0110,0111]	

RANGE FILTER – PREFIX BLOOM FILTER

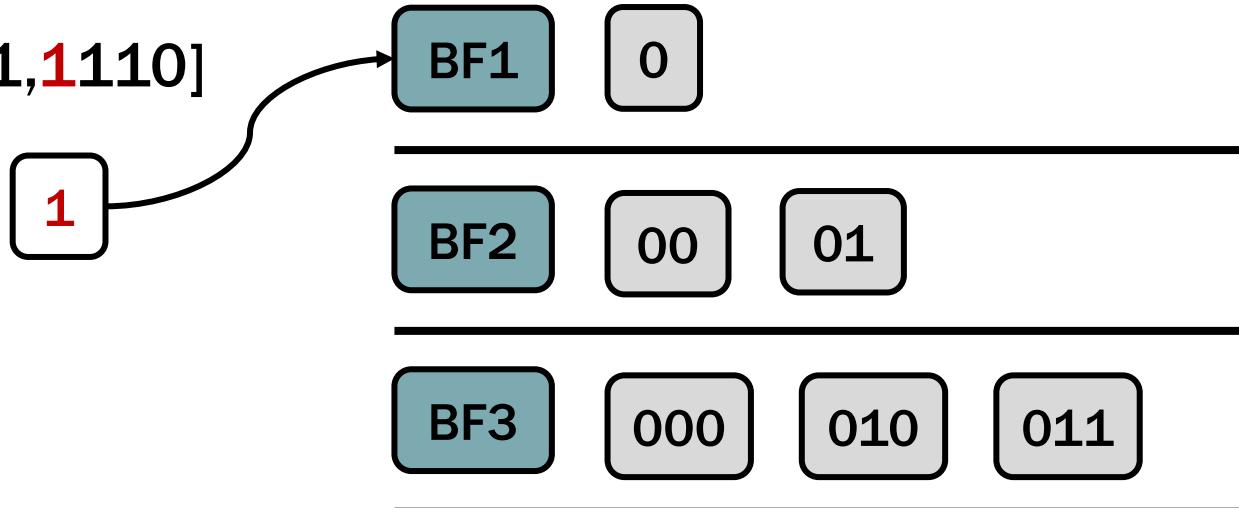
Example: set prefix length from 1 to 3.

Query

-> [9,14]

-> [1001,1110]

Key set: {0001, 0100, 0110}



→ NO

Do not need to query the following BFs due to there is not key in the extended query range [1000,1111]

RANGE FILTER – PREFIX BLOOM FILTER

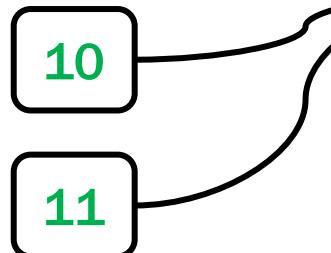
Example: set prefix length from 1 to 3.

Query

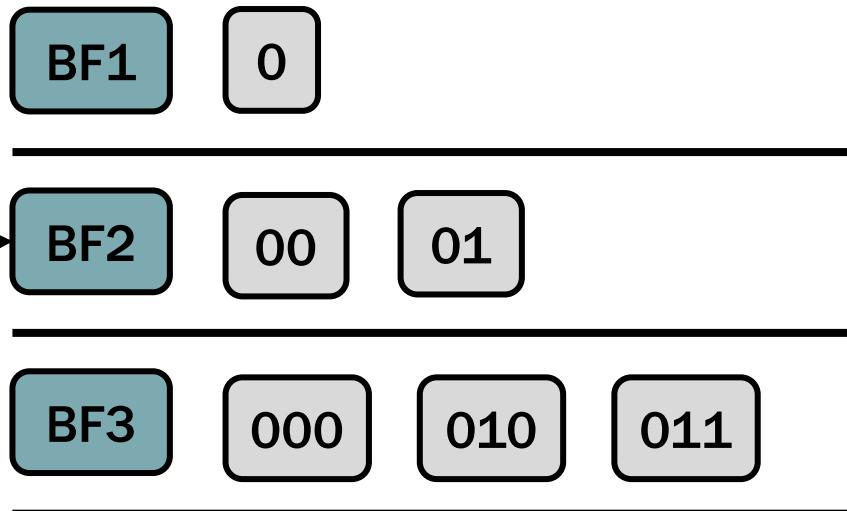
-> [9,14]

-> [1001, 1011]

[1100, 1110]



Key set: {0001, 0100, 0110}



This step is **unnecessary** for the reason mentioned

Query **1 bit** is sufficient,
more **efficient**.

RANGE FILTER – PREFIX BLOOM FILTER

Key set: {0001, 0100, 0110}

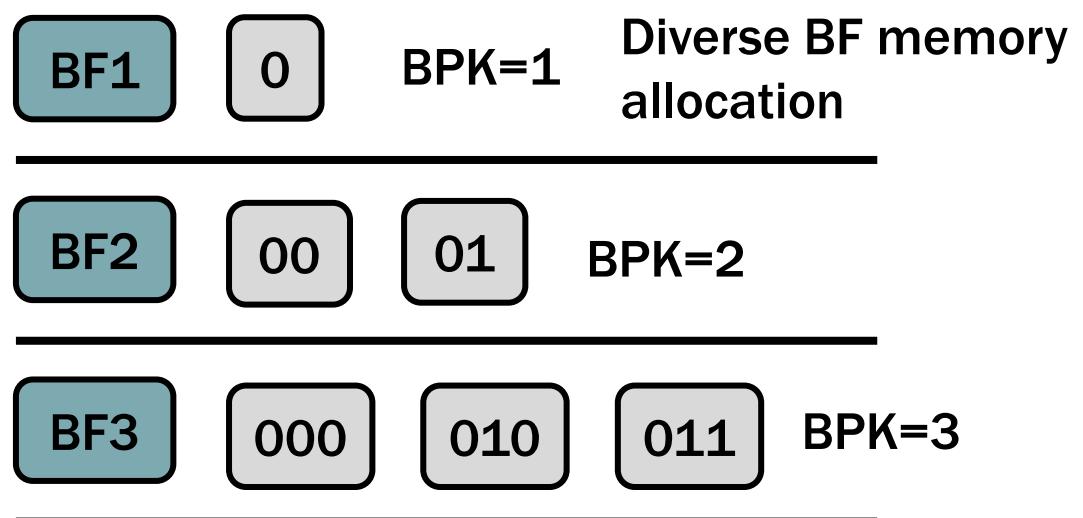
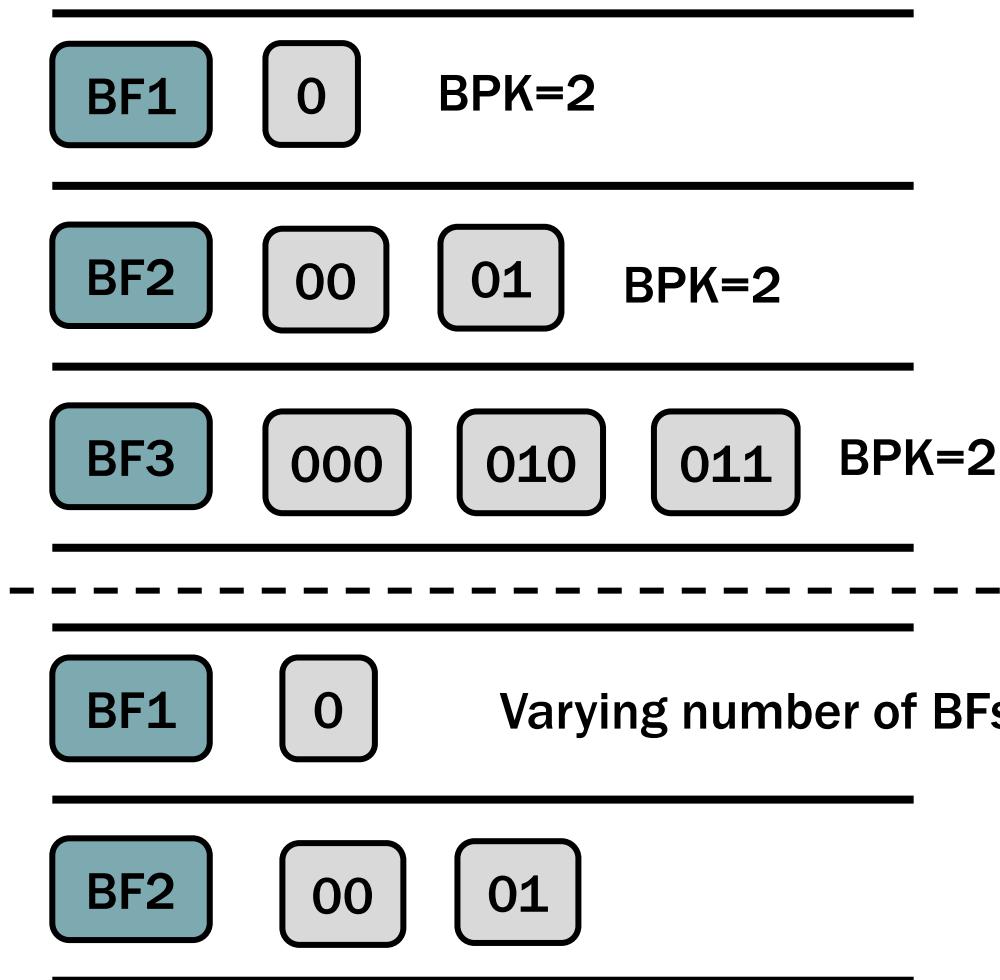
BF1	0	BPK=2
BF2	00	01
BF3	000	010
BPK=2		
BF1	0	Varying number of BFs
BF2	00	01

BF1	0	BPK=1	Diverse BF memory allocation
BF2	00	01	BPK=2
BF3	000	010	011
			BPK=3

More **flexible structure** to adapt to different work condition.

RANGE FILTER – PREFIX BLOOM FILTER

Key set: {0001, 0100, 0110}



However, more BF could take up
more **memory space**

**The End
Thank you!**

BIG DATA MANAGEMENT

CE/CZ4123

OVERVIEW

2ND HALF

Siqiang Luo

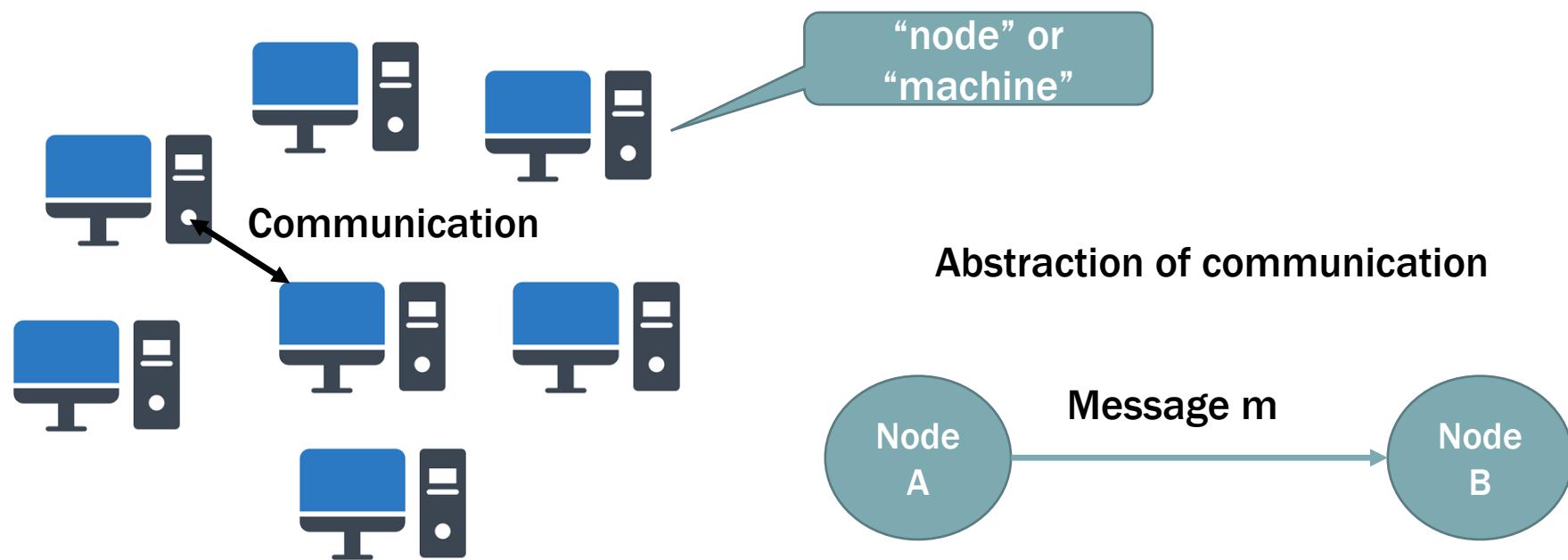
Assistant Professor

DISTRIBUTED SYSTEMS FOR BIG DATA: CHALLENGES

- How to organize the machines?
 - Fully-Distributed Mode
 - Master-Slave Mode
 - Fault-Tolerant
- How to store data across machines?
 - Data Partition
 - Data Replication
- How to compute using multiple machines?

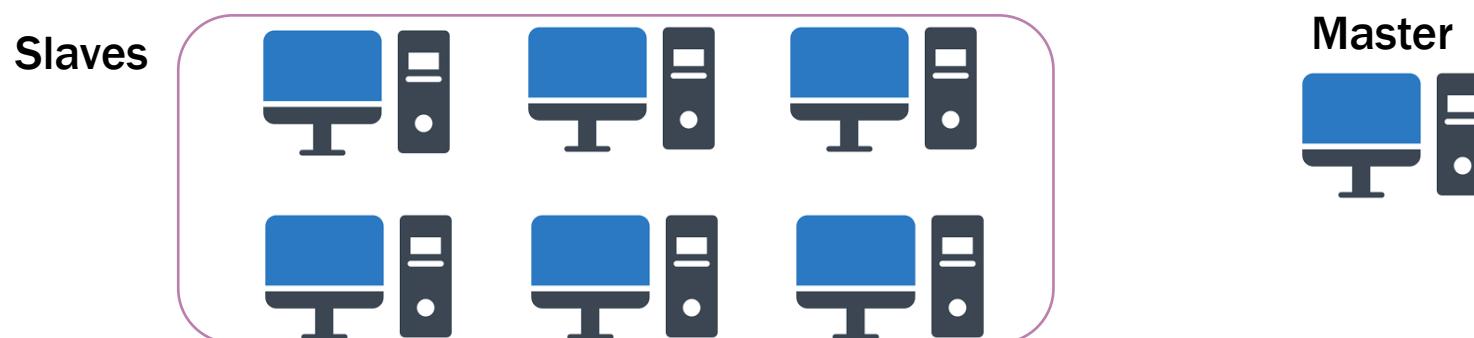
FULLY DISTRIBUTED MODELS

- Each machine has an IP address
- A knows machine B's IP address: A can send messages to B
- Two machines can communicate with each other via IP address
 - i.e., sending messages between machines



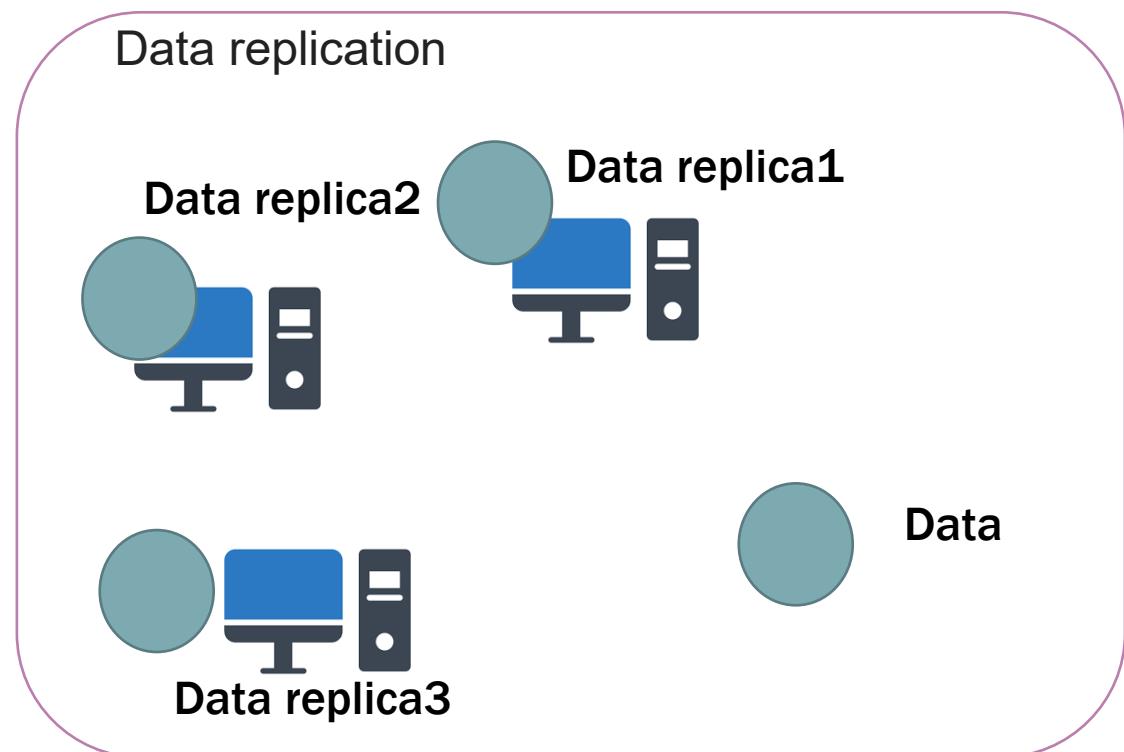
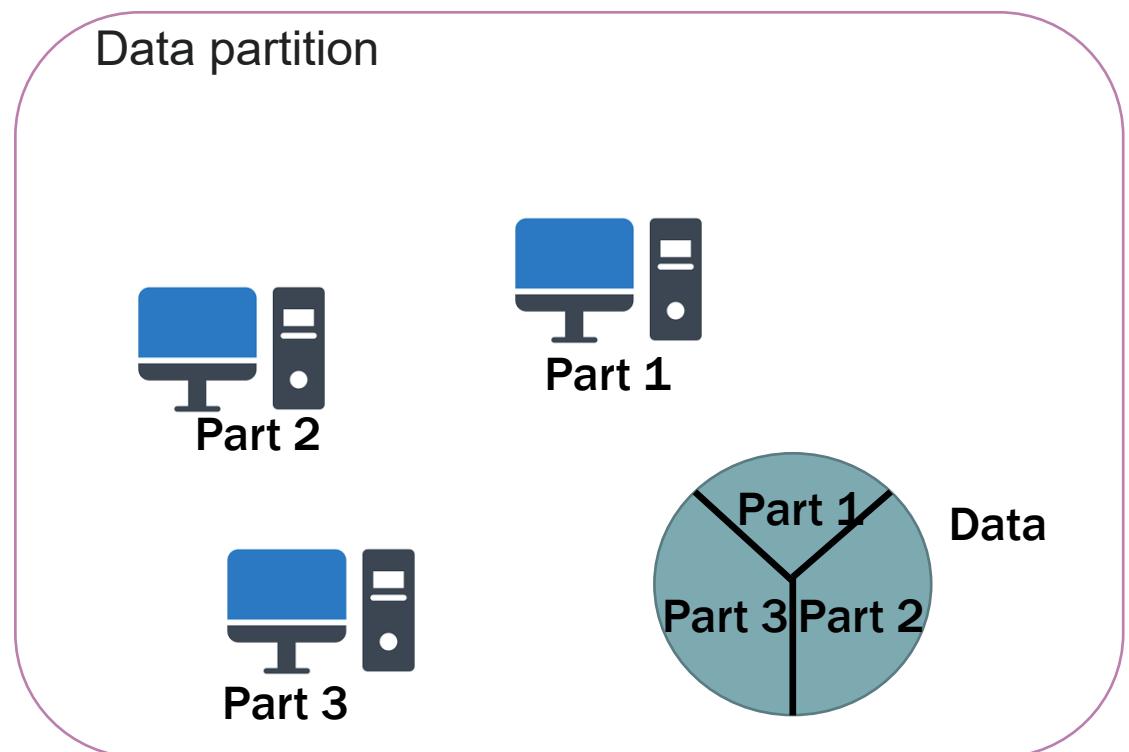
MASTER-SLAVE MODEL

- Each machine has an IP address
- There is a machine called **master**, and the other machines are called **slaves**.
- Master is the coordinator, being responsible to
 - distribute tasks to the slaves, and
 - receive the results from the slaves



DATA PARTITION AND DATA REPLICATION

- Data partition: partition the data into different machines
- Data replication: each data item can be replicated to multiple copies.



MAPREDUCE

- Understand the basic model of MapReduce
 - Map function
 - Reduce function
 - Job
- Understand the execution workflow of MapReduce
 - Within a job, reduce function receives the pairs with the same intermediate key
- Know how to design algorithms (pseudo-code)
 - Wordcount
 - Table Join
 - Shortest distance
 - PageRank

NOSQL

- Property of NoSQL
 - Flexible schema (schemaless)
 - Easier to scale
 - Partially supports query language
 - Queries are less flexible, but can have higher performance
- Types of NoSQL Systems
 - Key-Value Stores
 - Wide-Column Database
 - Document Database
 - Graph Database

KEY-VALUE STORES

- LSM-tree
 - Get
 - Put
 - Delete
 - Fence Pointers
 - Bloom filters
 - FPR
 - I/O cost analysis
 - Tiering LSM-tree
 - Range Filter (Not in the scope of final exam)

FINAL EXAM TIME AND VENUE

- May 8 1pm-3pm (Come Early!)**
- Hall 7**

Hall 7

Function Hall (former Meranti Hall)

- Closed-Book**
- Covers whole semester lectures (including those before quiz)**
- Instructions to Examination Candidates**
https://entuedu.sharepoint.com/sites/Student/dept/sasd/oas/Shared%20Documents/ExamAndAssessment/Exam/Instructions_to_candidates_physical_examinations_on-campus.pdf

**The End
Thank you!**