BIG DATA MANAGEMENT

CE/CZ4123

OVERVIEW (1ST HALF)

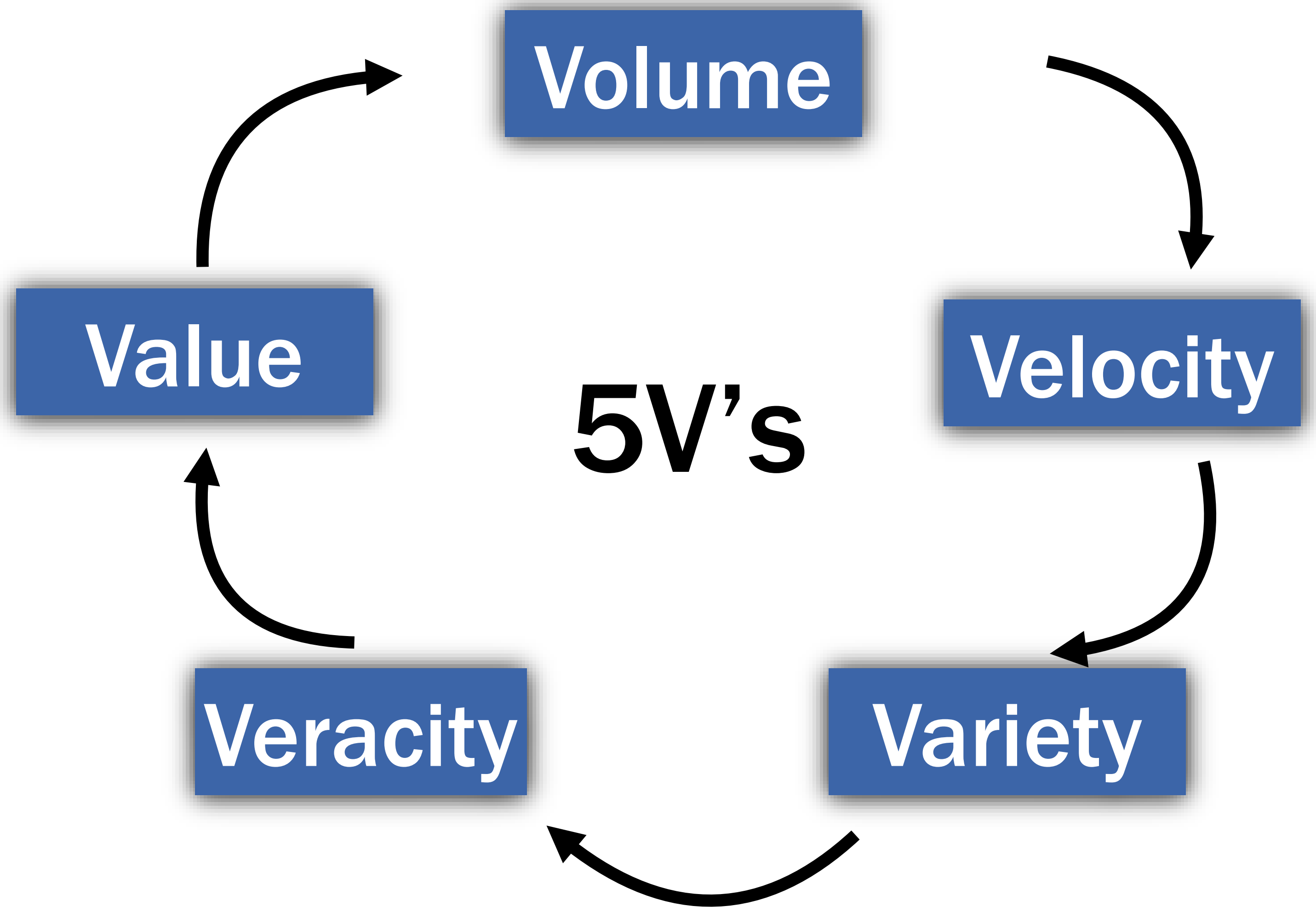
Siqiang Luo

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

Foreign Key

id	name	age	gender	Company Id
0001	Alex	25	M	c0001
0002	Mary	35	F	c0002

Employee (id, name, age, gender, company Id)

Primary Key

Company id	Company name	Country
c0001	Amazon	U.S.
c0002	Tesla	U.S.

Company (<u>Company Id</u>, Company name)

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

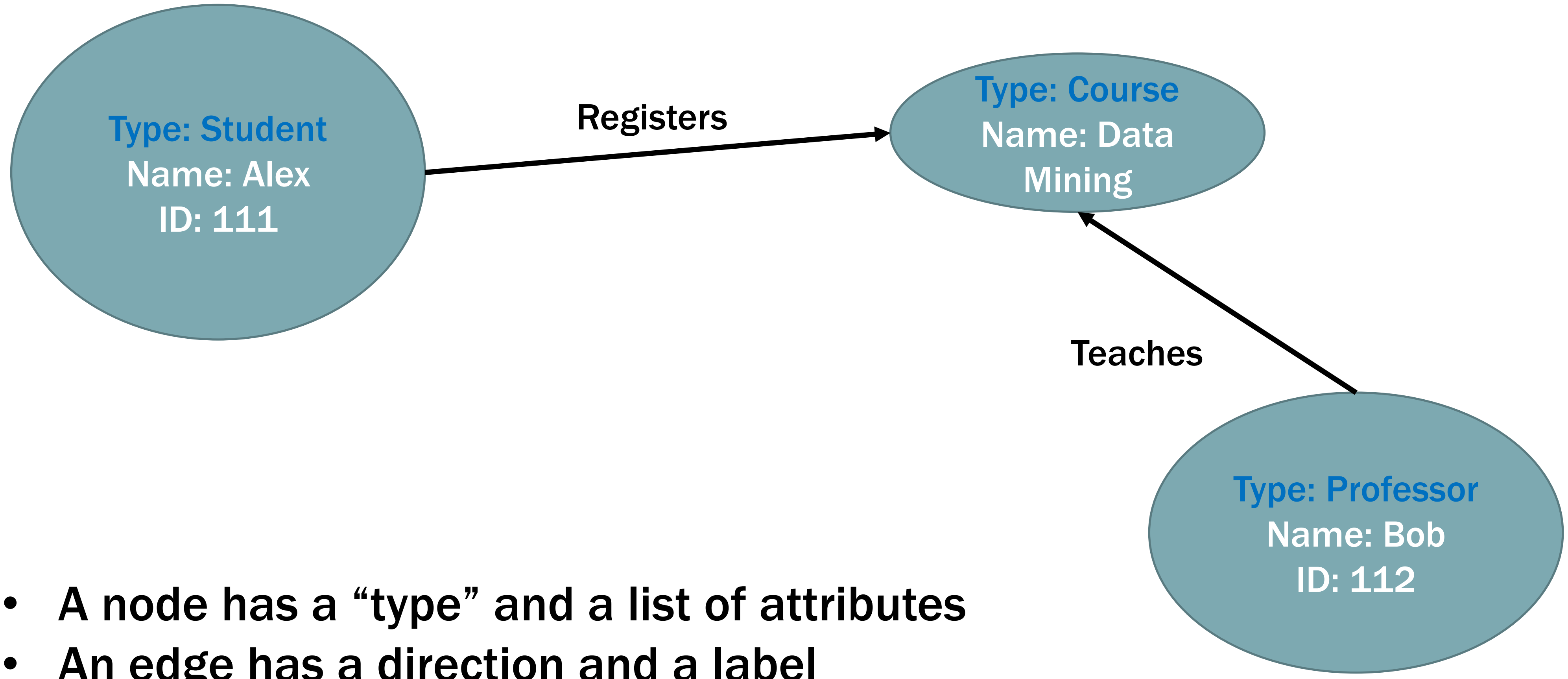
KEY-VALUE DATA MODEL

Key-value Data Model is ubiquitous!

For any A that can determine B

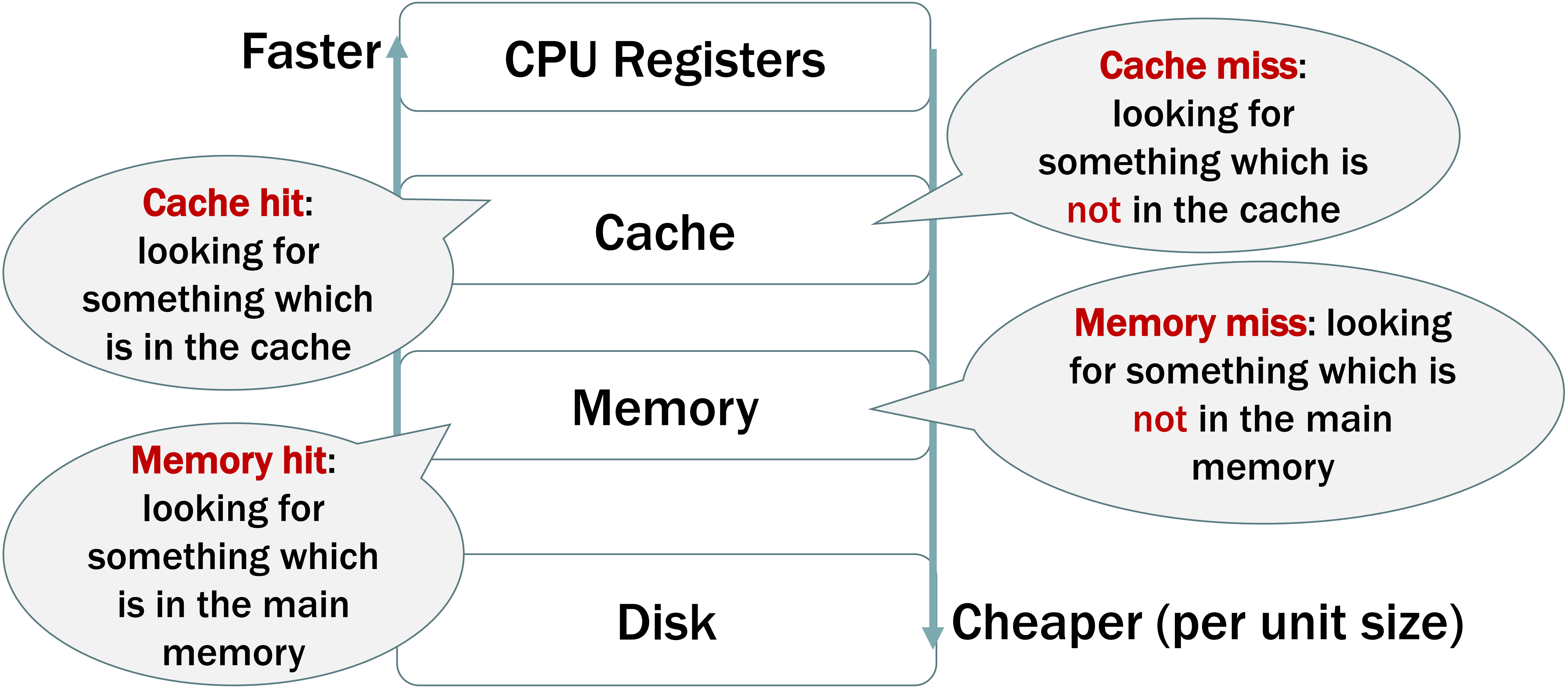
Key Value

GRAPHS ARE UBIQUITOUS

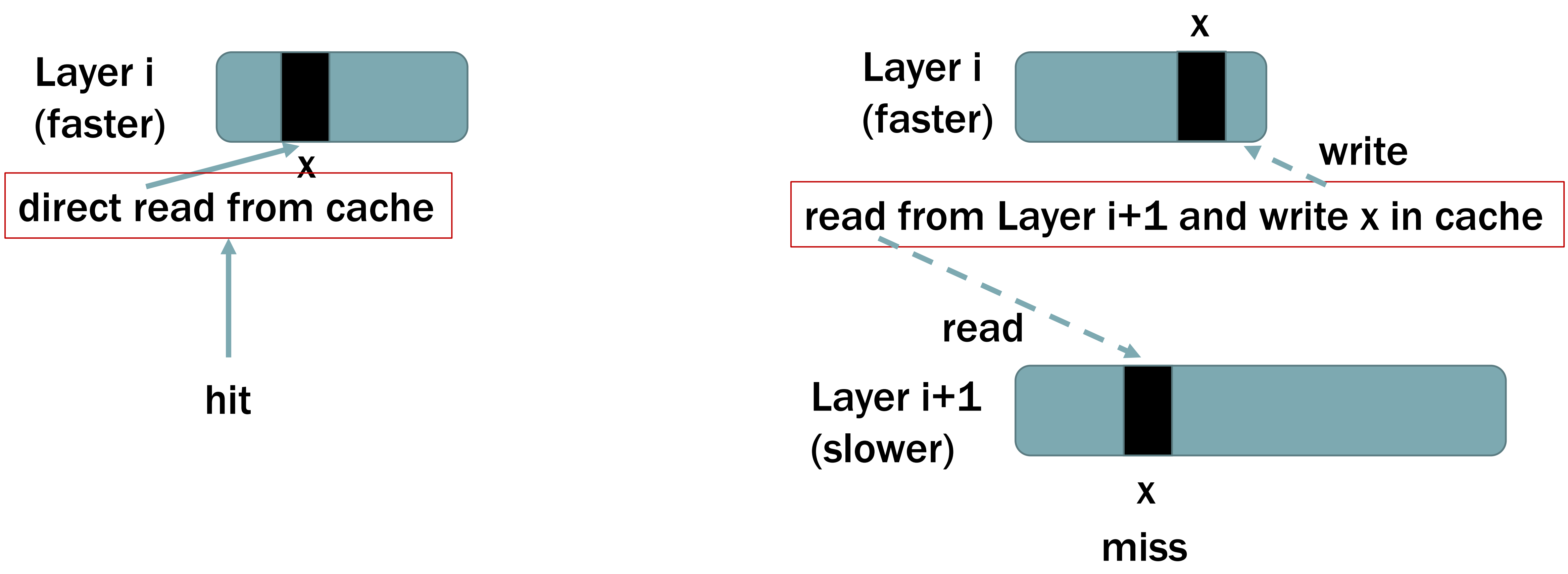


- A node has a “type” and a list of attributes
- An edge has a direction and a label

MEMORY HIERARCHY



DATA ACCESS IN MEMORY HIERARCHY



COST OF A CACHE MISS

- cost_access_mem_overall =cost_access_cache x (1-missrate) +
  (cost_access_cache + cost_access_mem) x 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 x 5 = 40 bytes

PAGE-BASED ACCESS EXAMPLE: SCANNING ARRAYS

Cost: 40 Bytes

Query x < 4 from the following data

Scan

Qualified results

(size=120 bytes)

Memory Layer i

5, 10, 7, 4, 12

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 x 5 = 40 bytes

PAGE-BASED ACCESS EXAMPLE: SCANNING ARRAYS

Cost: 80 Bytes

Query x < 4 from the following data

Scan

Qualified results

(size=120 bytes)
Memory Layer i

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 x 5 = 40 bytes

PAGE-BASED ACCESS EXAMPLE: SCANNING ARRAYS

Cost: 120 Bytes

Query x < 4 from the following data

Scan

Qualified results

(size=120 bytes)
Memory Layer i

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 x 5 = 40 bytes

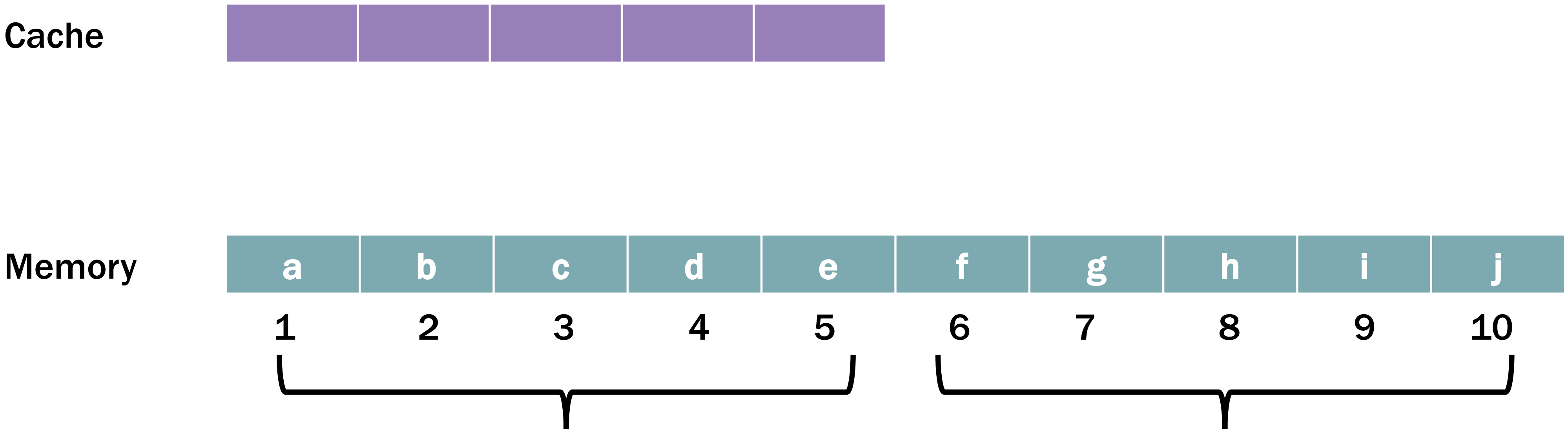
CACHE CONSCIOUS DESIGNS

- 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	a	b	c	d	e

Cache Miss: 1
Cache Hit: 0

Memory	a	b	c	d	e	f	g	h	i	j
	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

a	b	c	d	e

Cache Miss: 1
Cache Hit: 1

Memory

a	b	c	d	e	f	g	h	i	j
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					
	a	b	c	d	e

Cache Miss: 1
Cache Hit: 2

Memory										
	a	b	c	d	e	f	g	h	i	j
	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

a	b	c	d	e

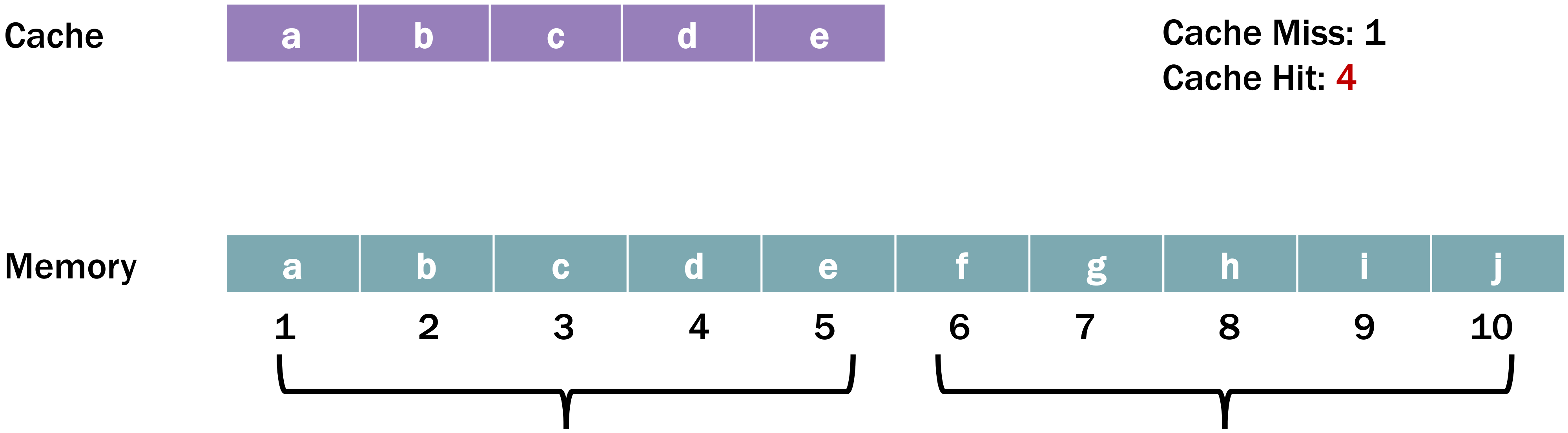
Cache Miss: 1
Cache Hit: 3

Memory

a	b	c	d	e	f	g	h	i	j
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 Miss: 1
Cache Hit: 4

ACCESS PATTERN 1

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

Cache: f, g, h, i, j

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

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

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

Cache

f	g	h	i	j

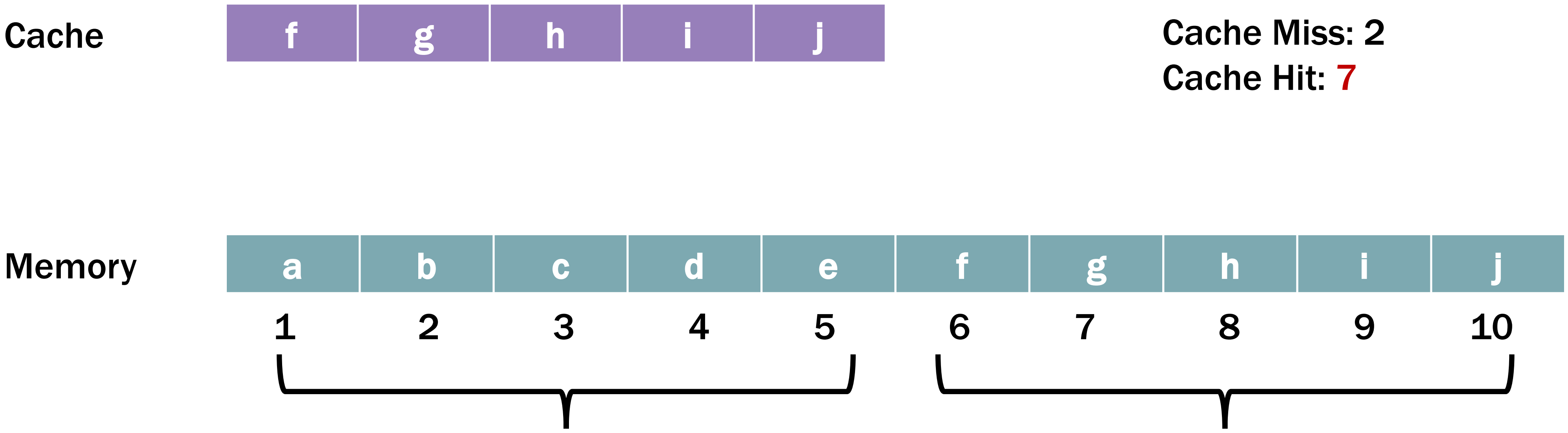
Cache Miss: 2
Cache Hit: 6

Memory

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

ACCESS PATTERN 1

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



Cache Miss: 2
Cache Hit: 7

ACCESS PATTERN 1

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

Cache

f	g	h	i	j

Cache Miss: 2
Cache Hit: 8

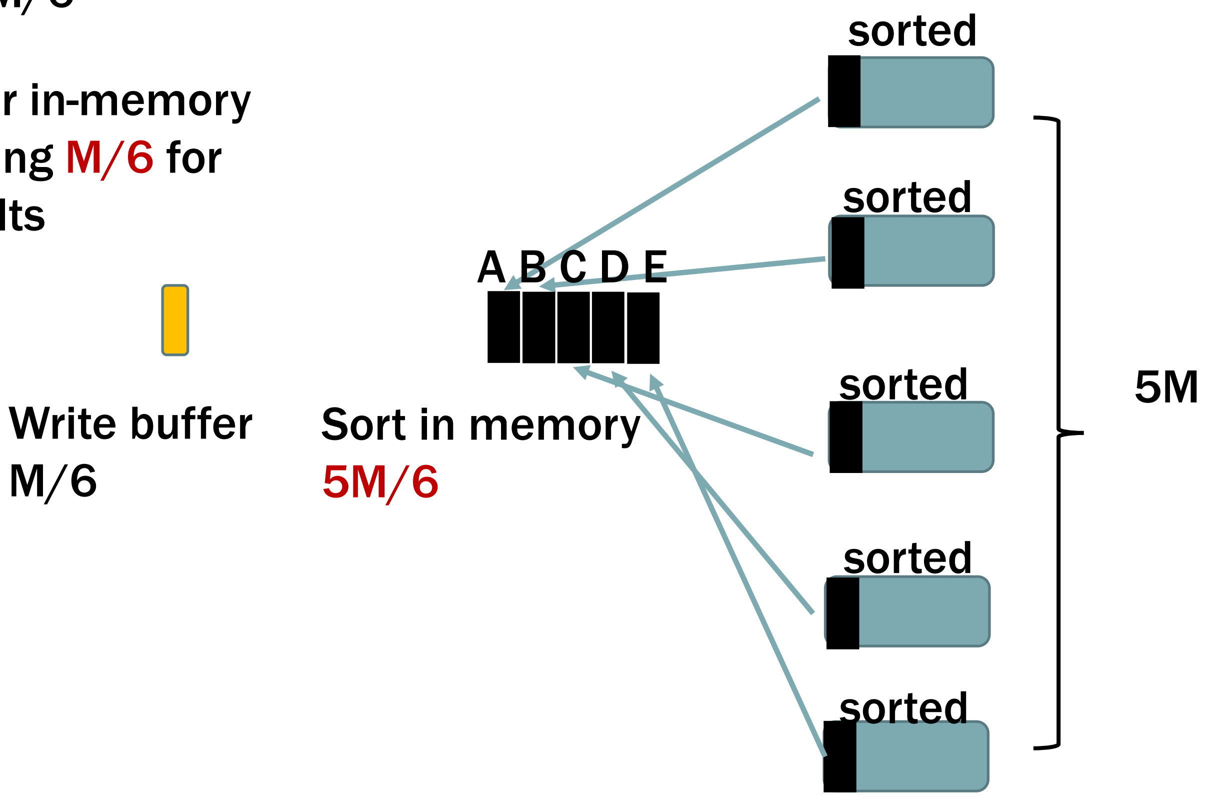
Memory

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

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

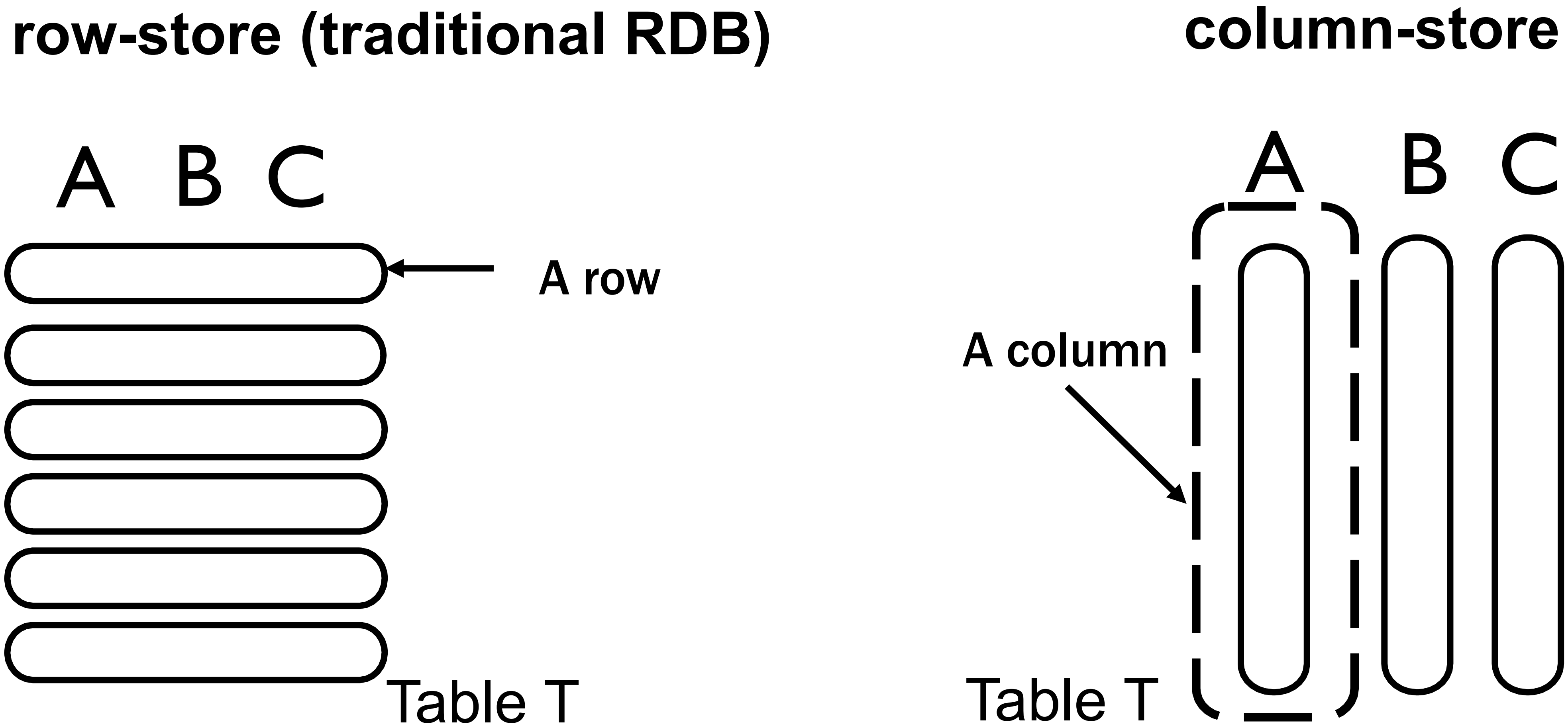
Step 3:

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.

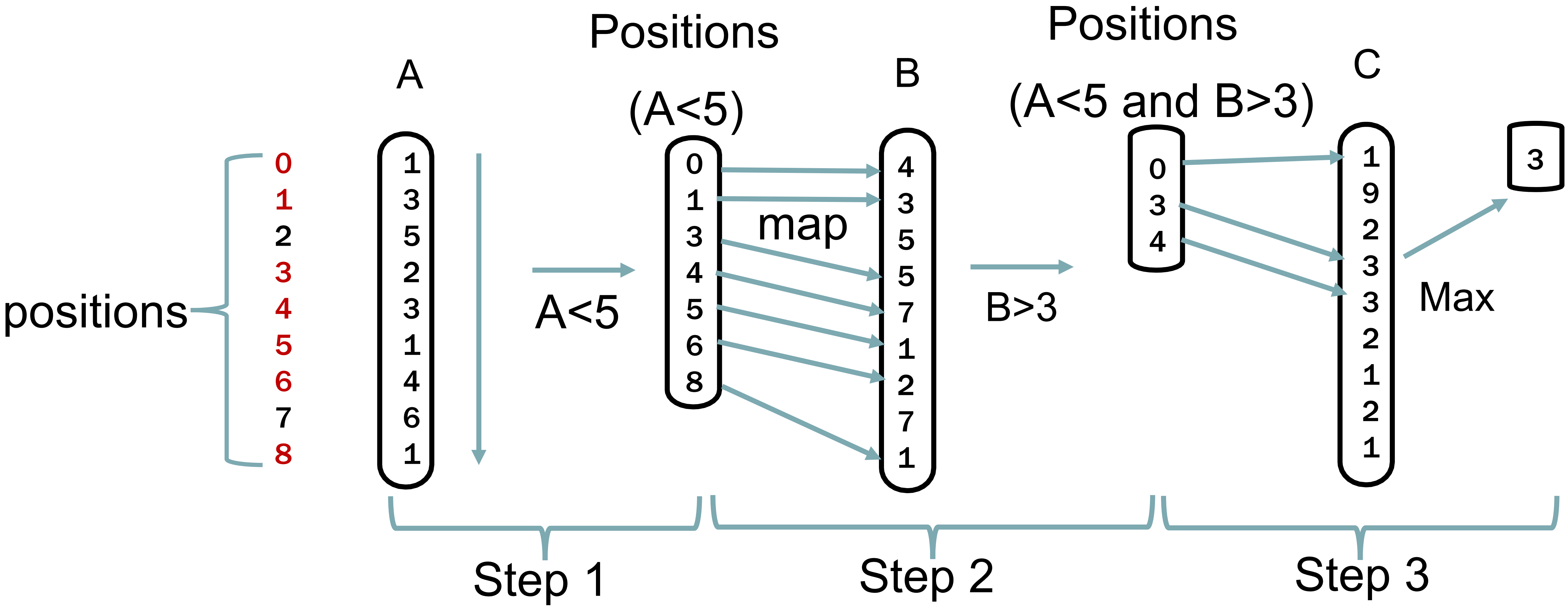
MAIN DESIGN OF COLUMN STORES

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



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):

$Zw/P+2result(A)*4/P+result(A)+2result(AB)*4/P+result(AB)$

$=Zw/P+result(A)*(8/P+1)+result(AB)*(8/P+1)$

$\approx Zw/P+result(A)+result(AB)$

Cost for row store (number of page access):

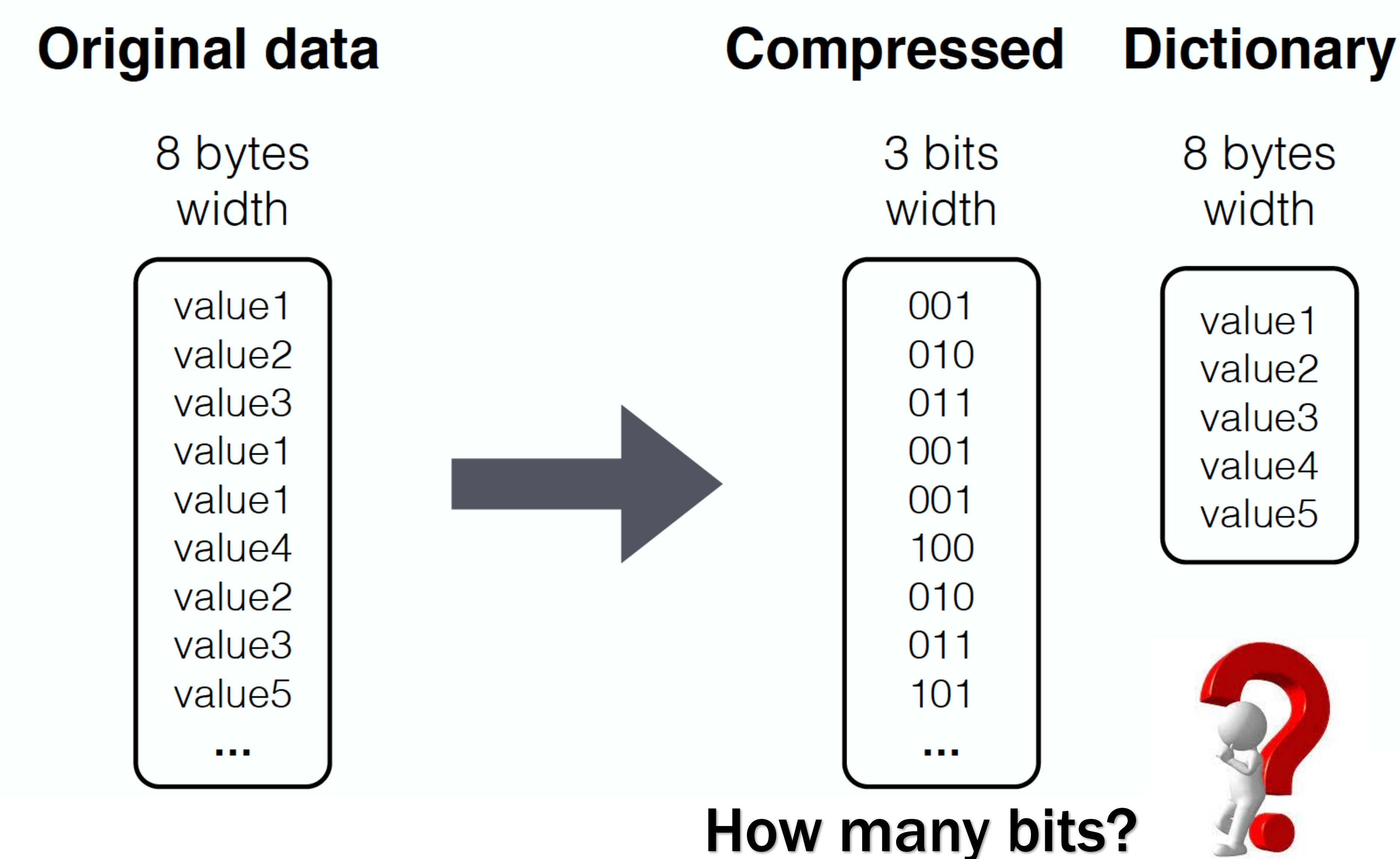
$3Zw/P$

Note: the values of result(A) and 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



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
```

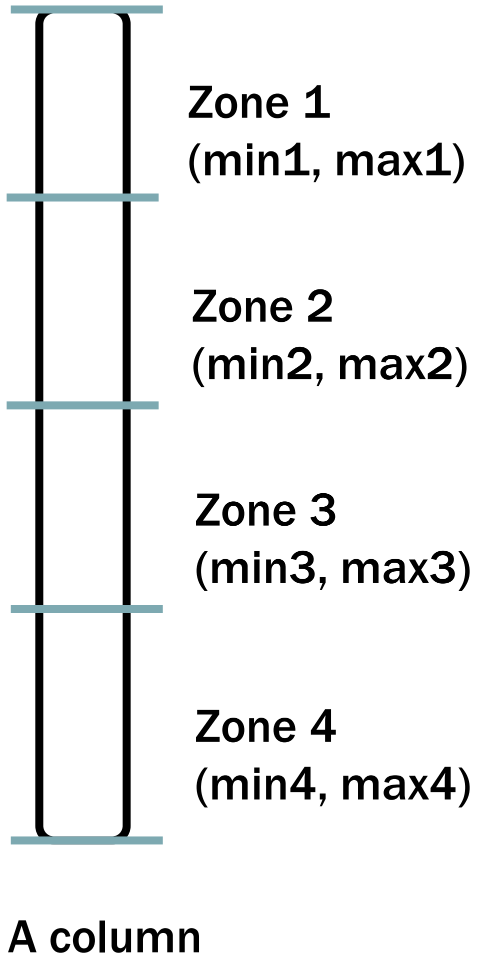
Two passes of data

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

One pass of data

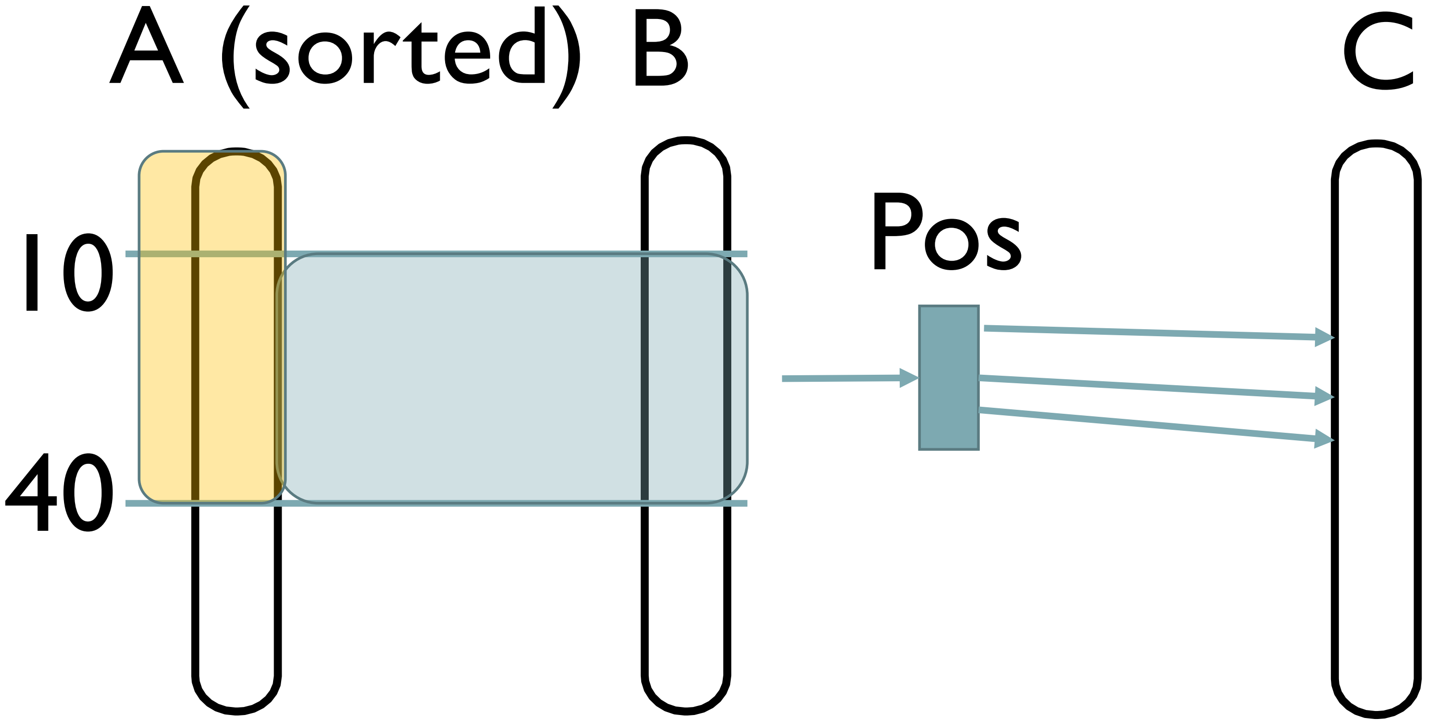
ZONE MAP

- 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



First scan A up to 40 (yellow)

Then scan B in blue range and filter for B>20

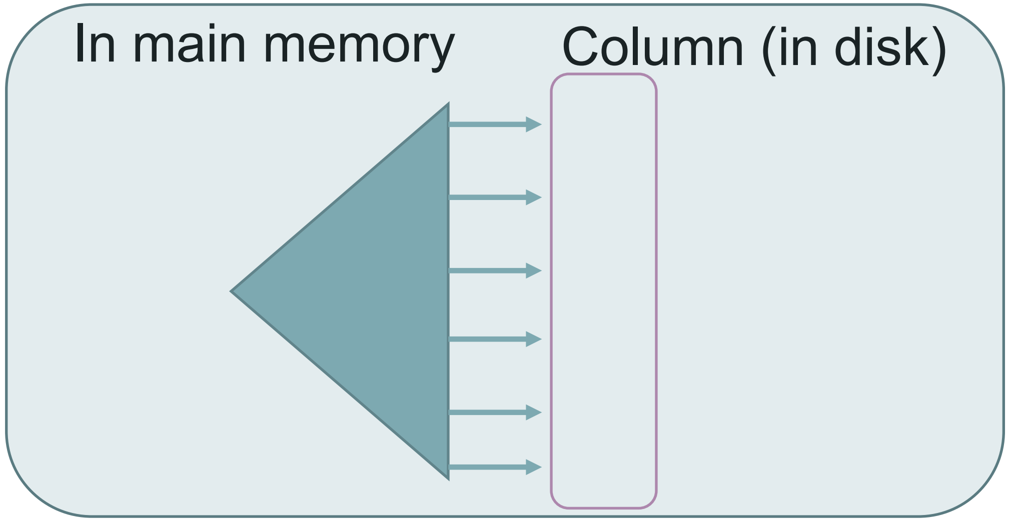
Store positions for tuples having A>10, A<40 and B>20

Select max(C) based on the positions

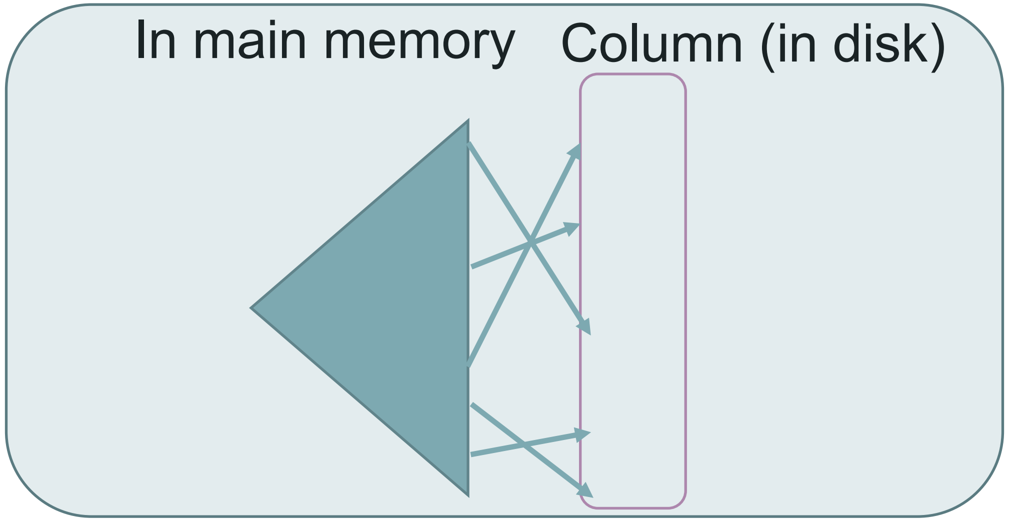
ENHANCED WITH INDEX

- 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



The End of First-Half.
Thank you!