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

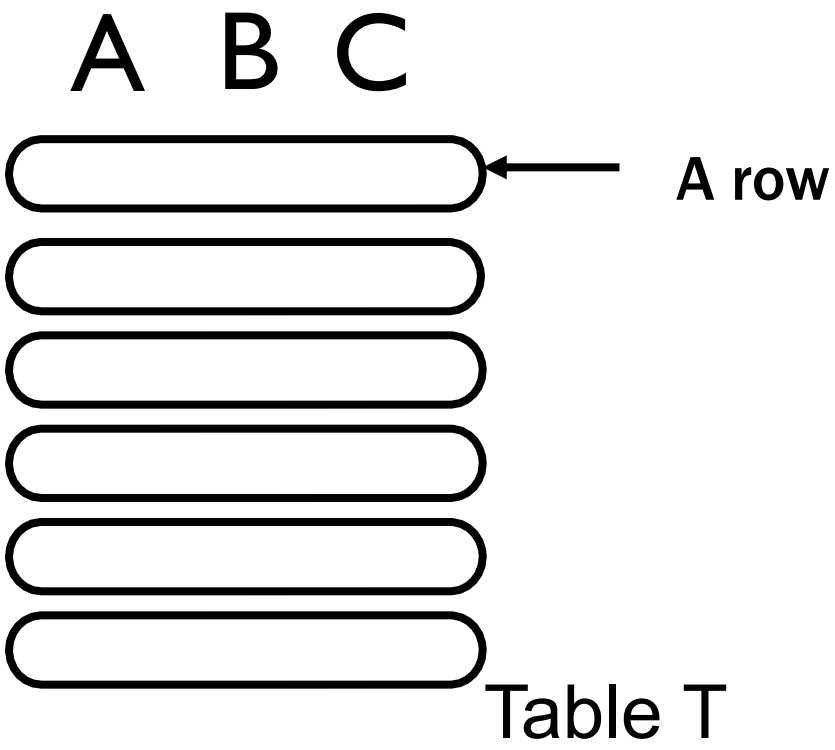
COLUMN STORE HANDLES TABULAR DATA

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

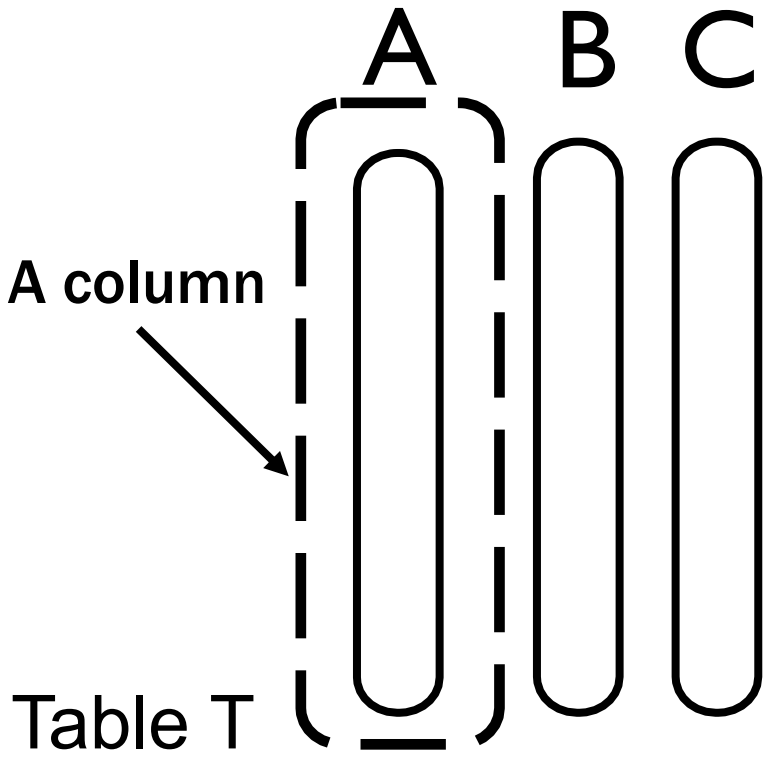
MAIN DESIGN OF COLUMN STORES

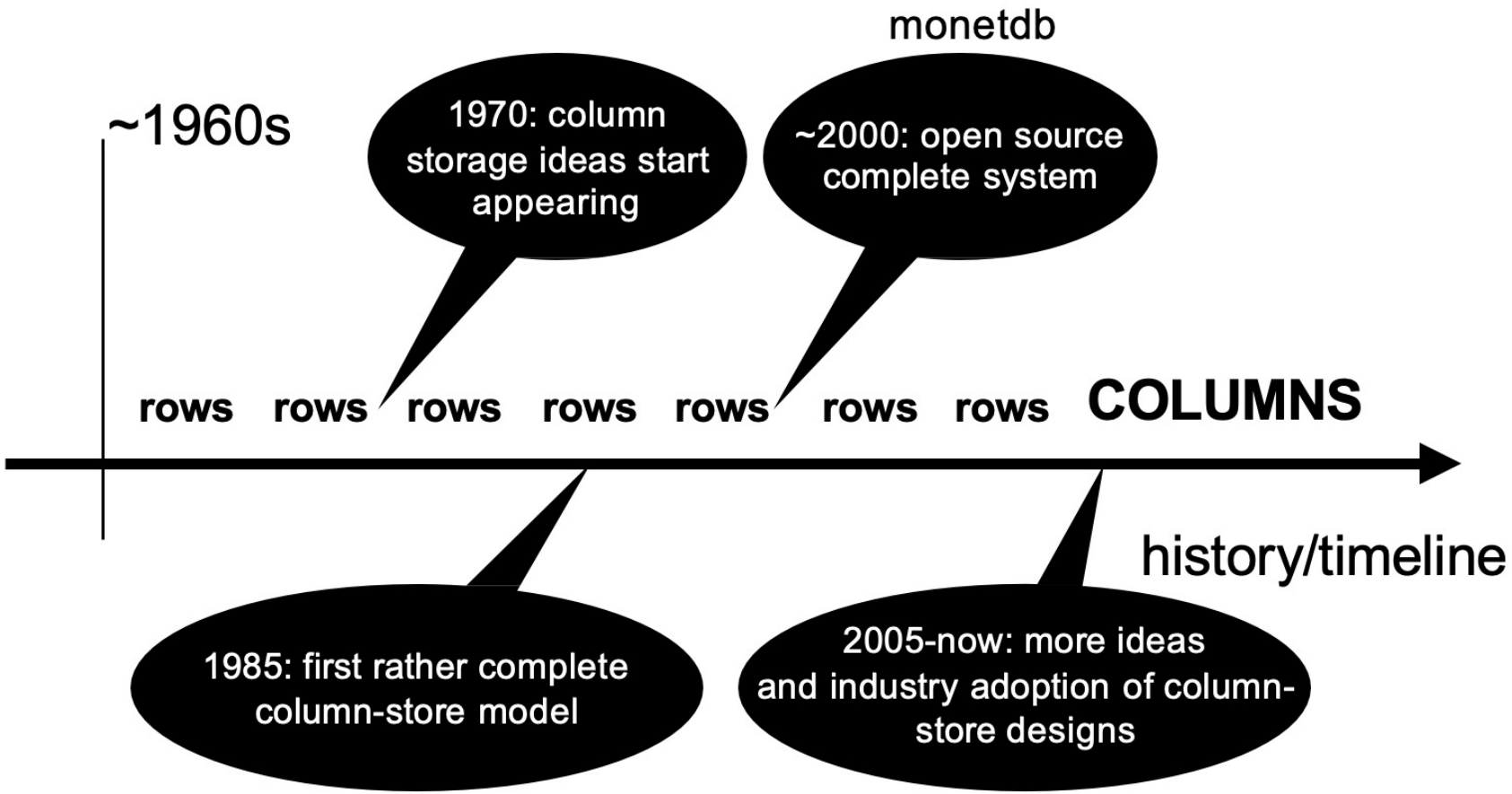
- Data in column stores are column-oriented
  - Also called column-oriented database, or columnar database.

row-store (traditional RDB)



column-store





Credit to
Prof. Stratos Idreos
at Harvard University.

c-store,vertica,vectorwise
and then
Ibm,microsoft,oracle,google,twitter,linkedin,and more

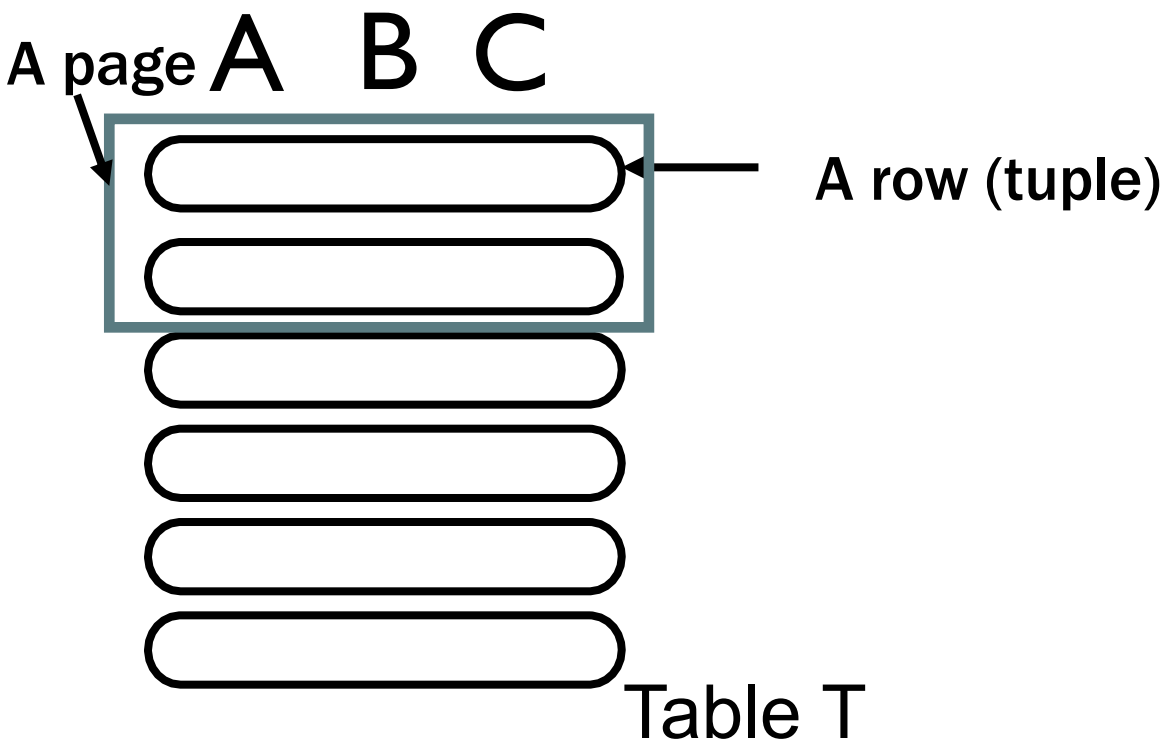
MAIN DESIGN OF COLUMN STORES

- 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

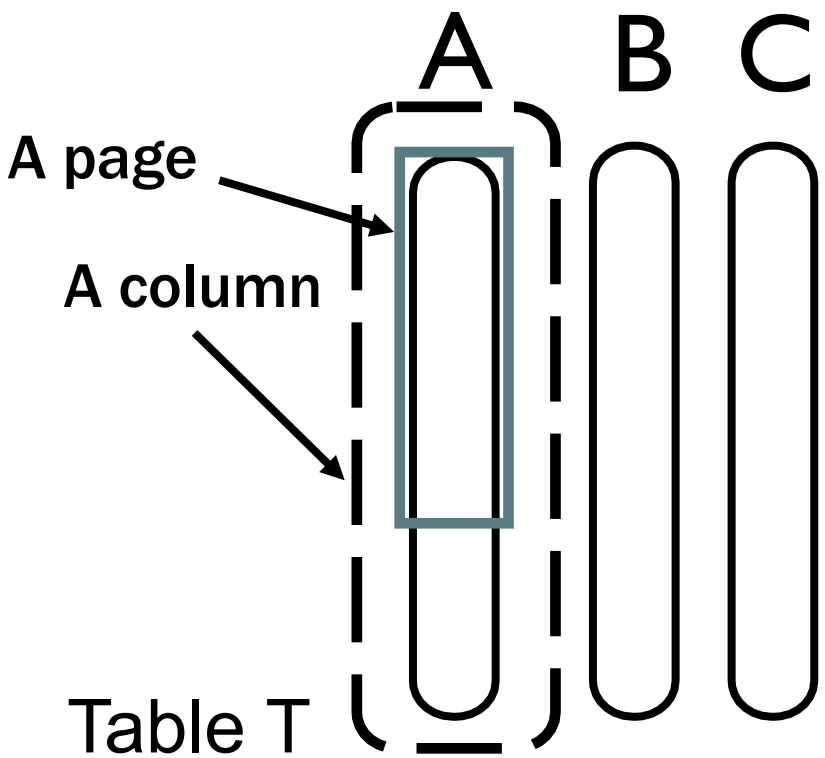
MAIN DESIGN OF COLUMN STORES

- Data are stored based on pages

row-store (traditional RDB)



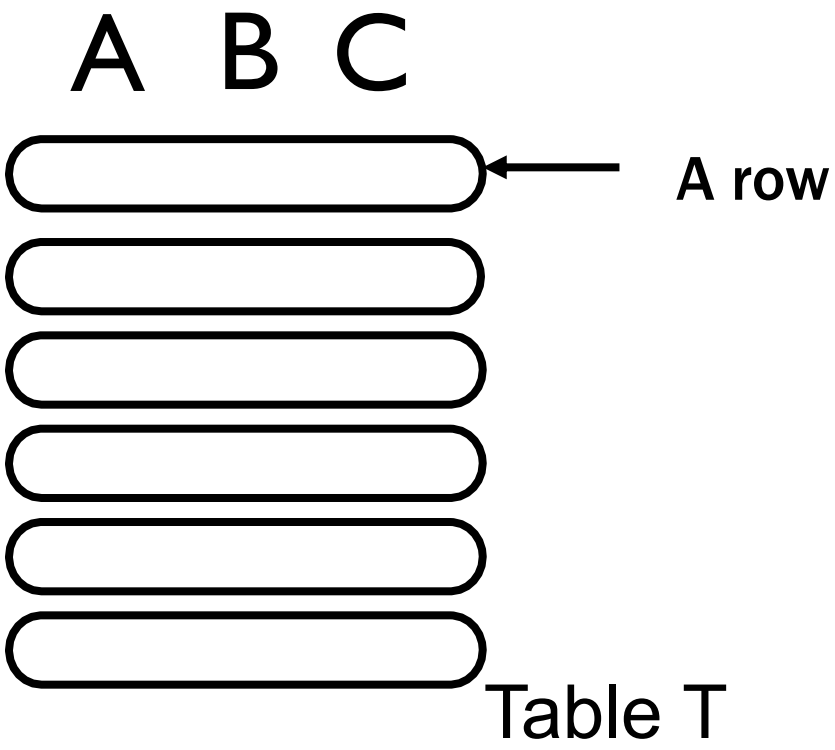
column-store



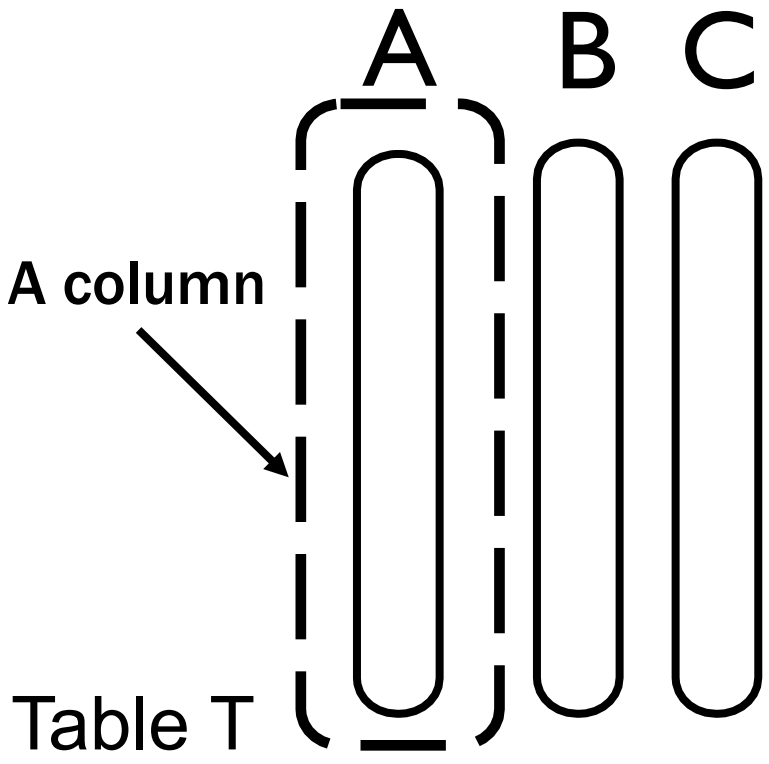
WHY COLUMN STORE

- 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

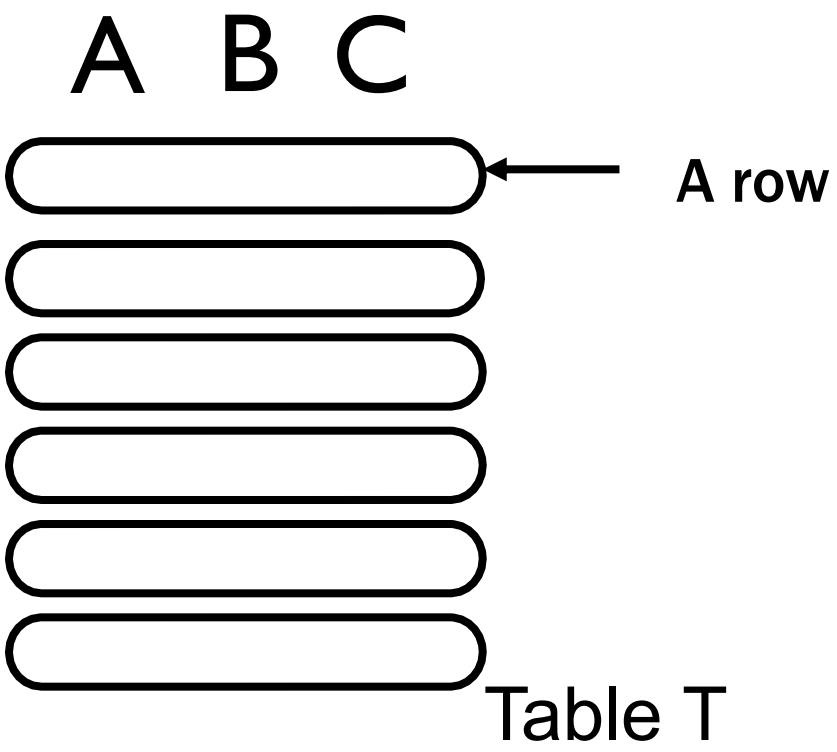


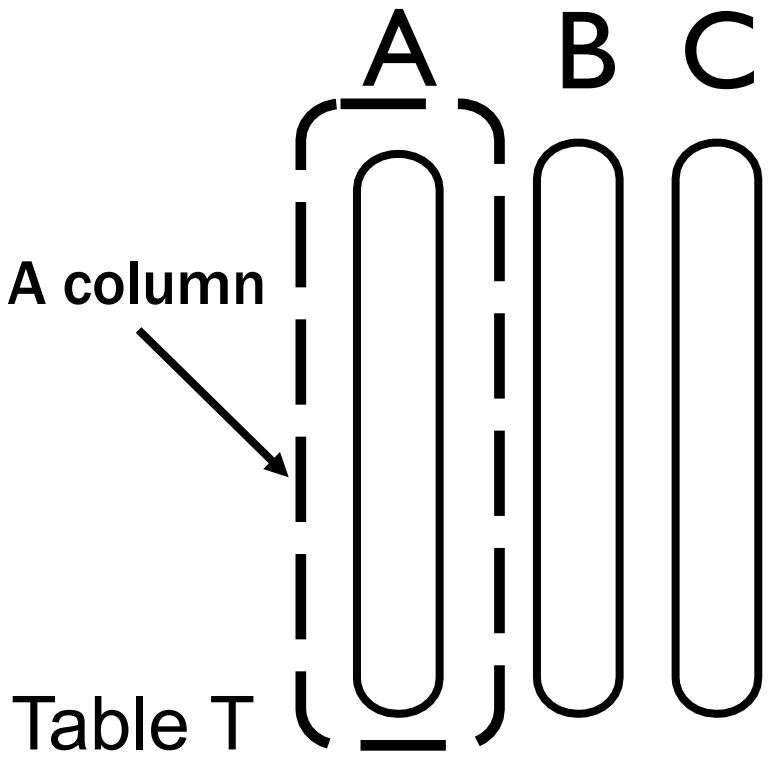
An excellent introduction video here.

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

WHY COLUMN STORE

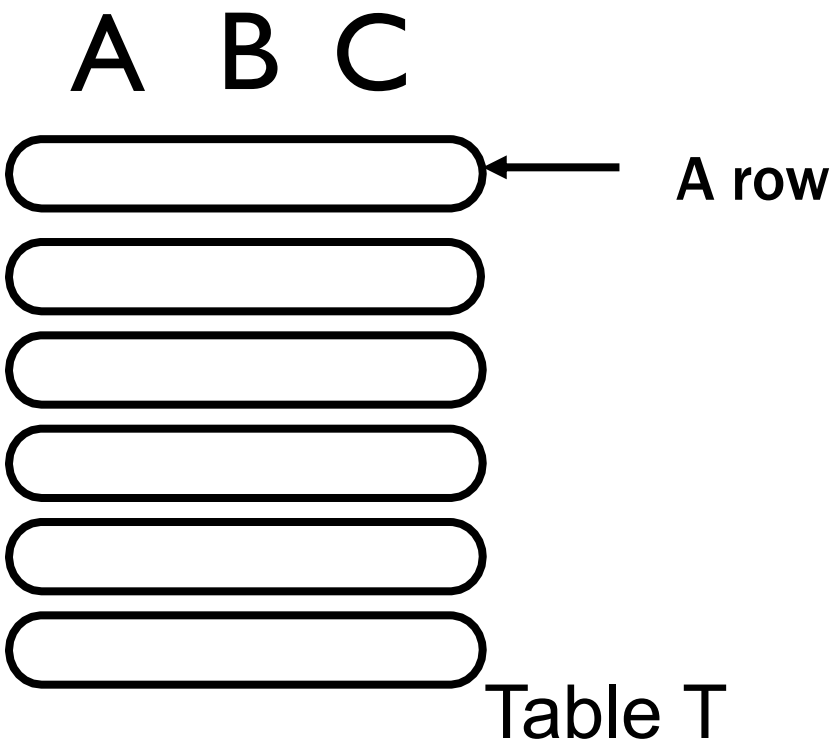
- Consider the following cases:
- select A from T where A>4

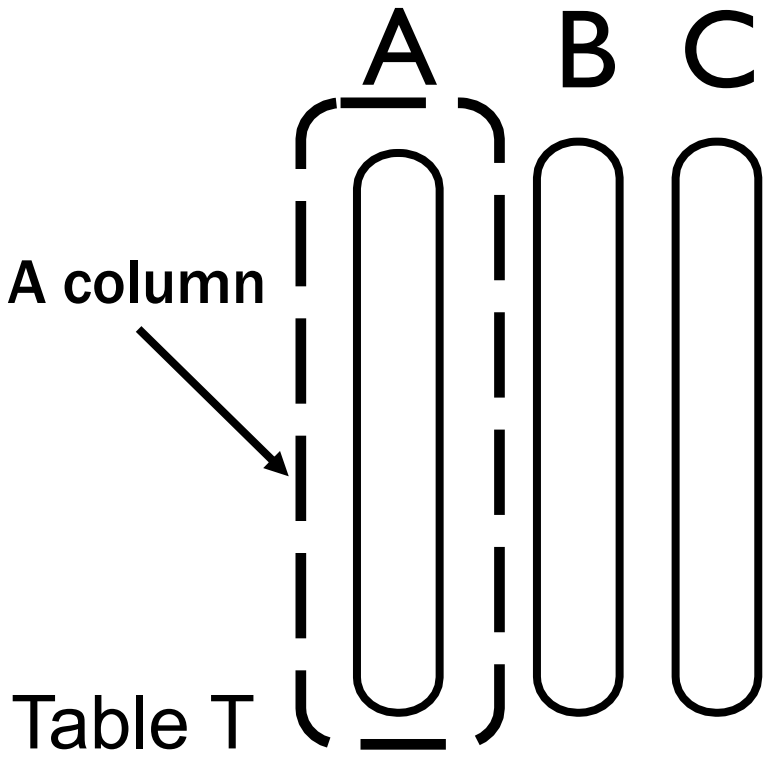




WHY COLUMN STORE

- 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.

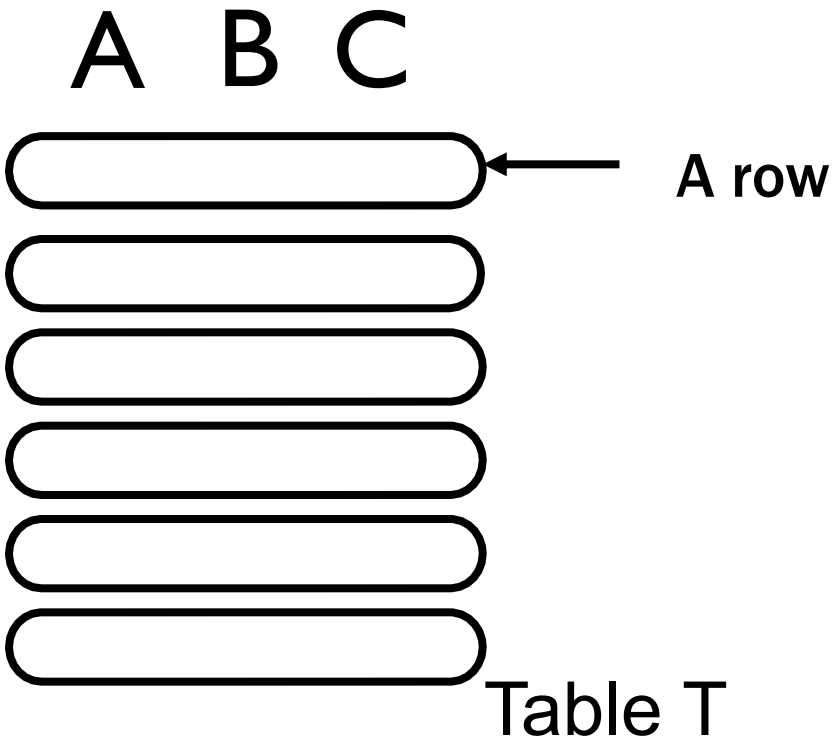


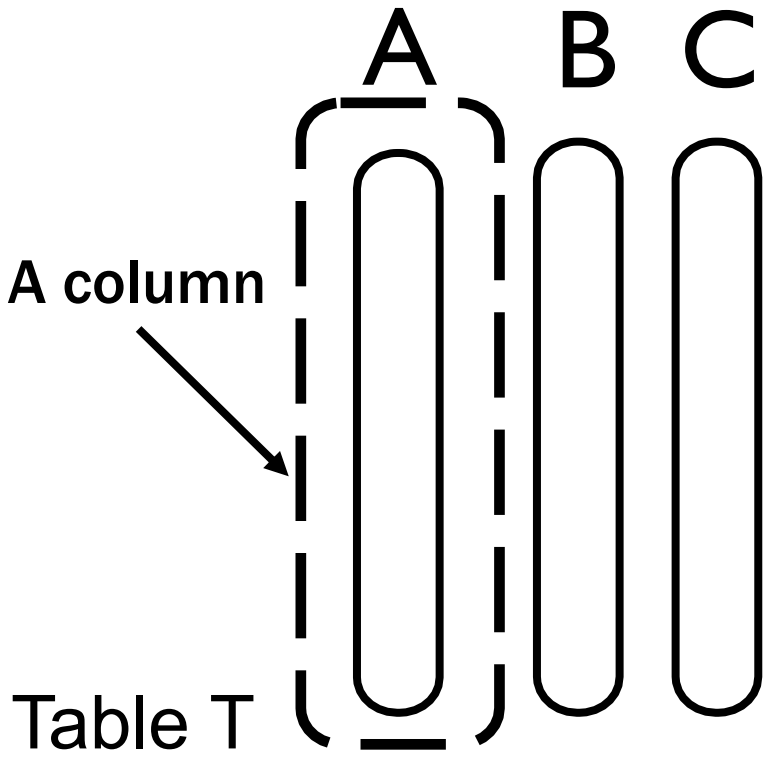


Any other benefits?

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.



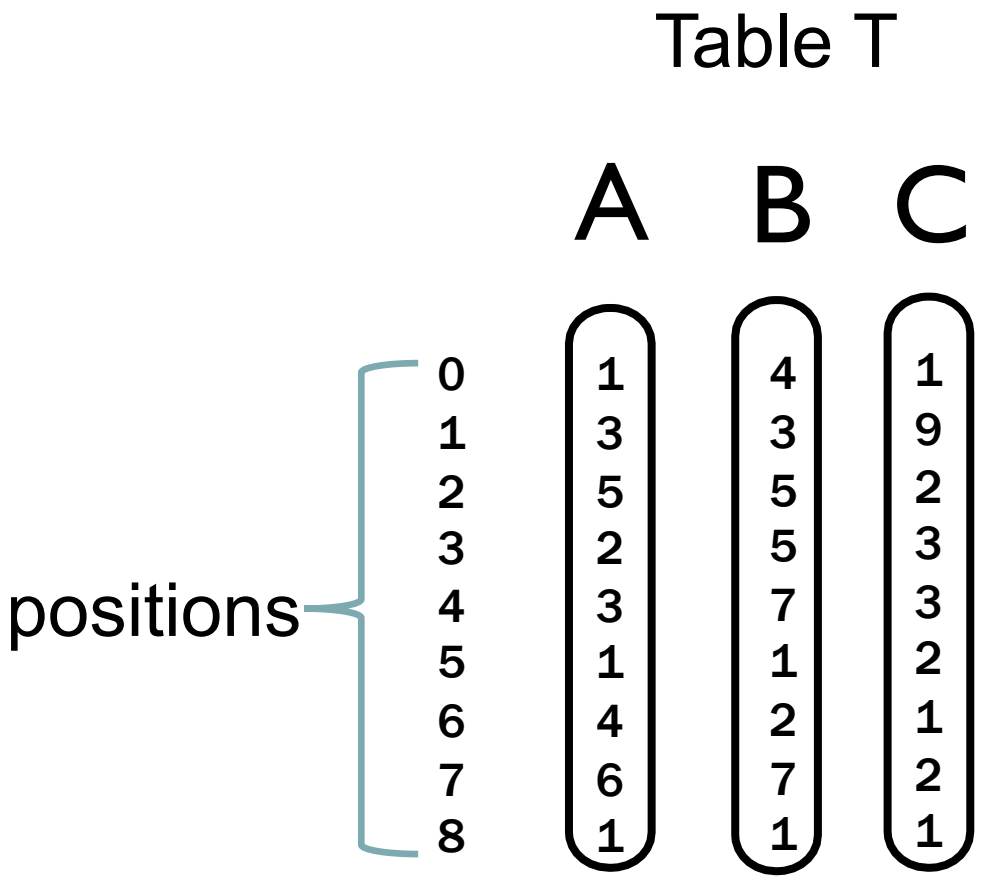


Querying with Column Store

QUERYING WITH COLUMN STORE

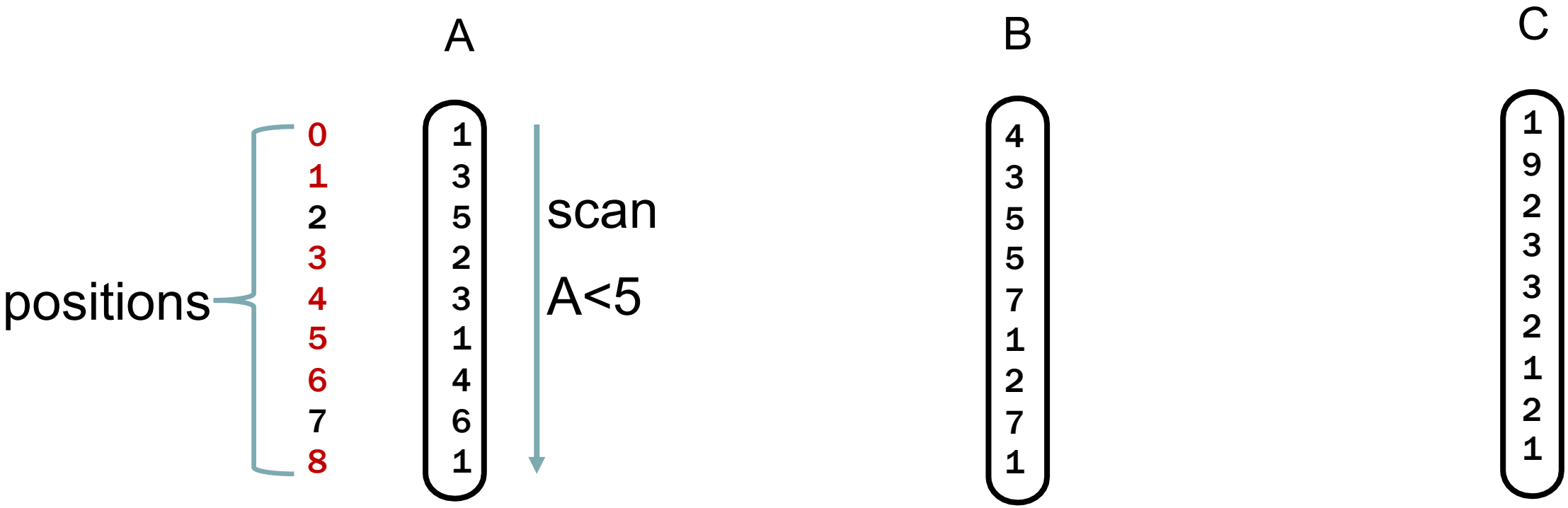
- The whole table is stored in disk
- Have disk cache (in main memory) whose size is a multiple of page size

```
select max( C )
from T
where A<5 and B>3
```



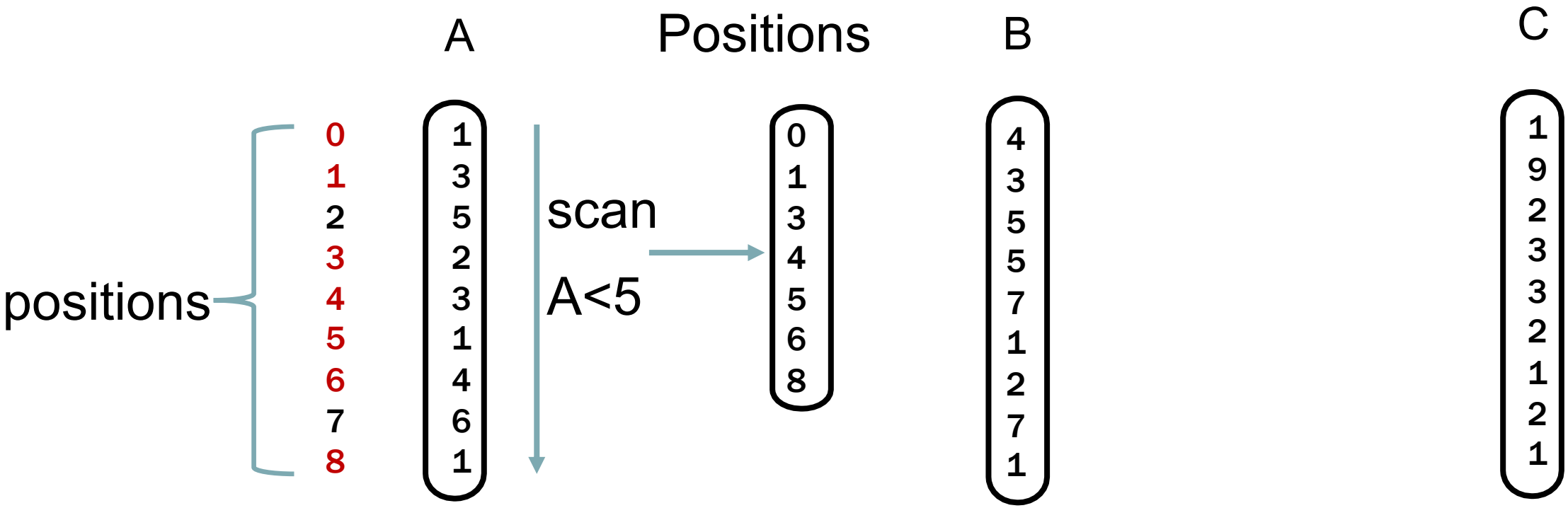
QUERYING WITH COLUMN STORE (FLOW CHART)

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



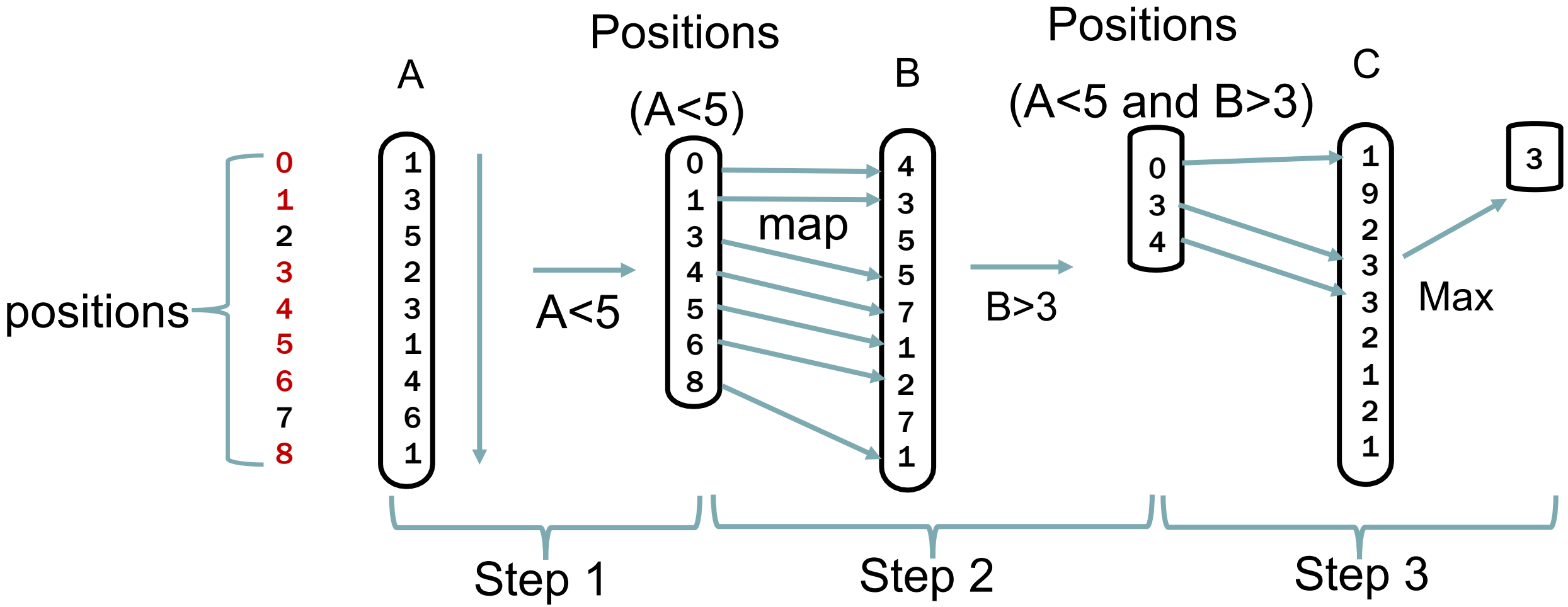
QUERYING WITH COLUMN STORE (FLOW CHART)

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

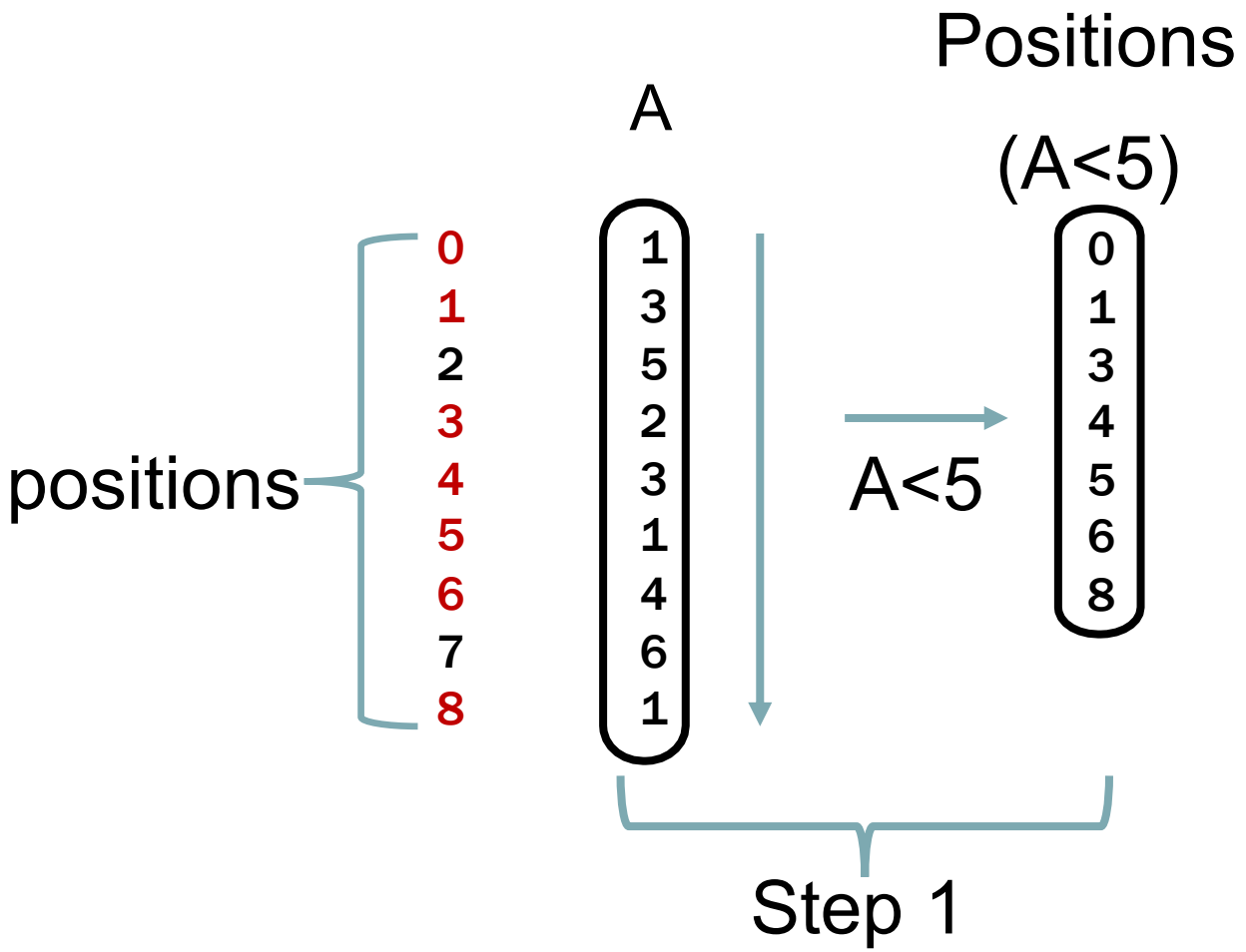


QUERYING WITH COLUMN STORE (FLOW CHART)

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



DETAILS OF STEP 1

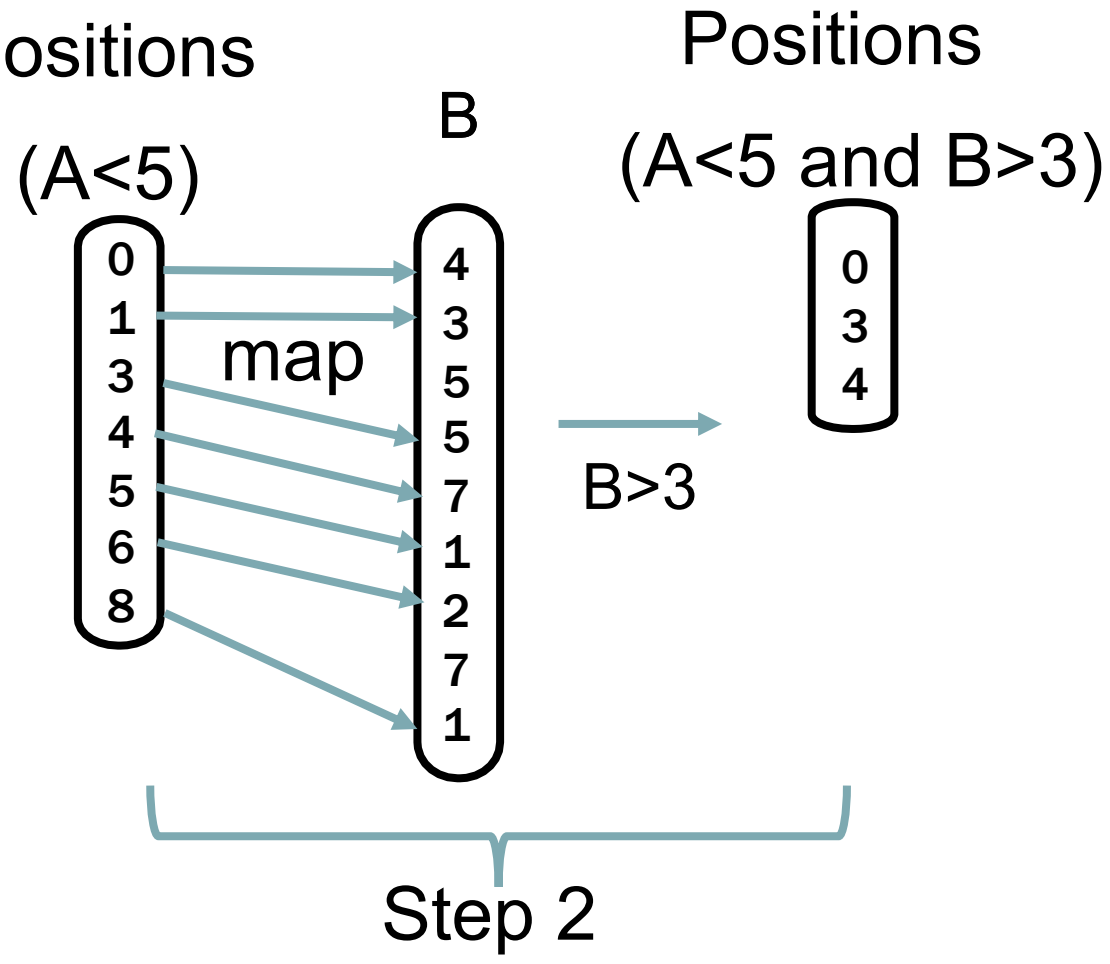



```
int j=0;
for (int i=0; i<#tuples; i++)
{
    if(A[i]<5)
    {
        Pos[j]=i;
        j++;
    }
}
```

QUERYING WITH COLUMN STORE

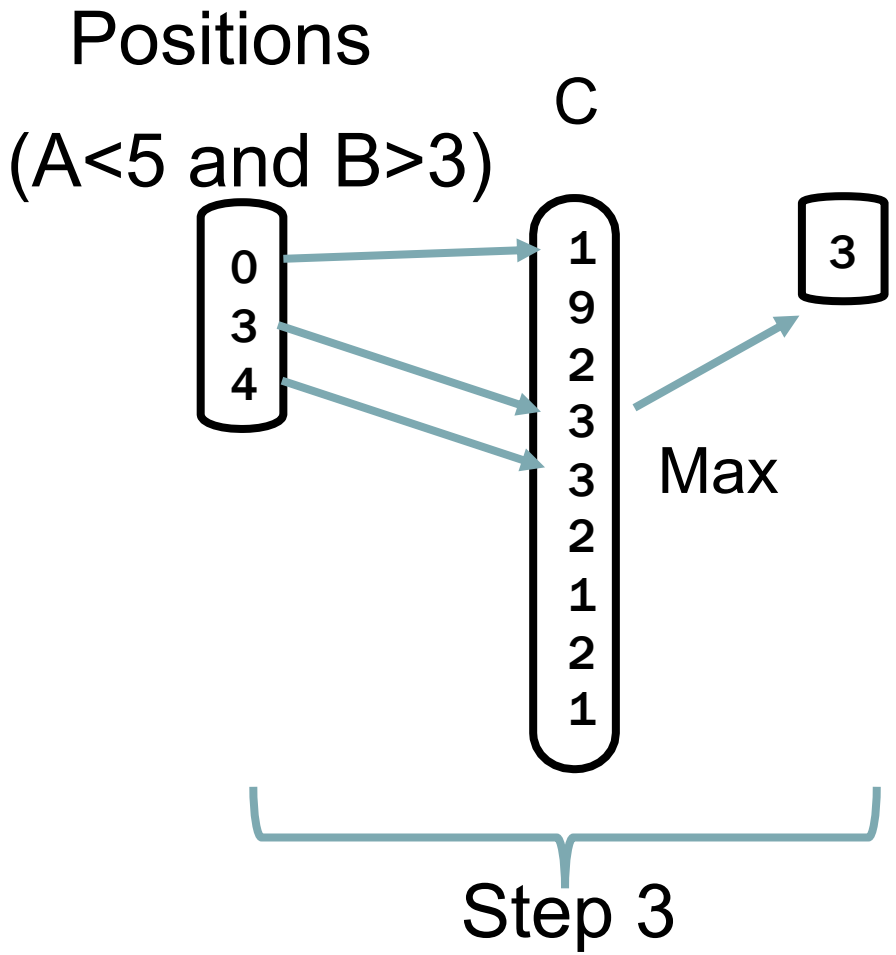
- The whole table is stored in disk
- Have disk cache (memory buffer) whose size is a multiple of page size

```
Int j=0;
for (int i=0; i<Pos.size; i++)
{
      if(B[Pos[i]]>3)
      {
            Pos2[j]=Pos[i];
            j++;
      }
}
```



QUERYING WITH COLUMN STORE

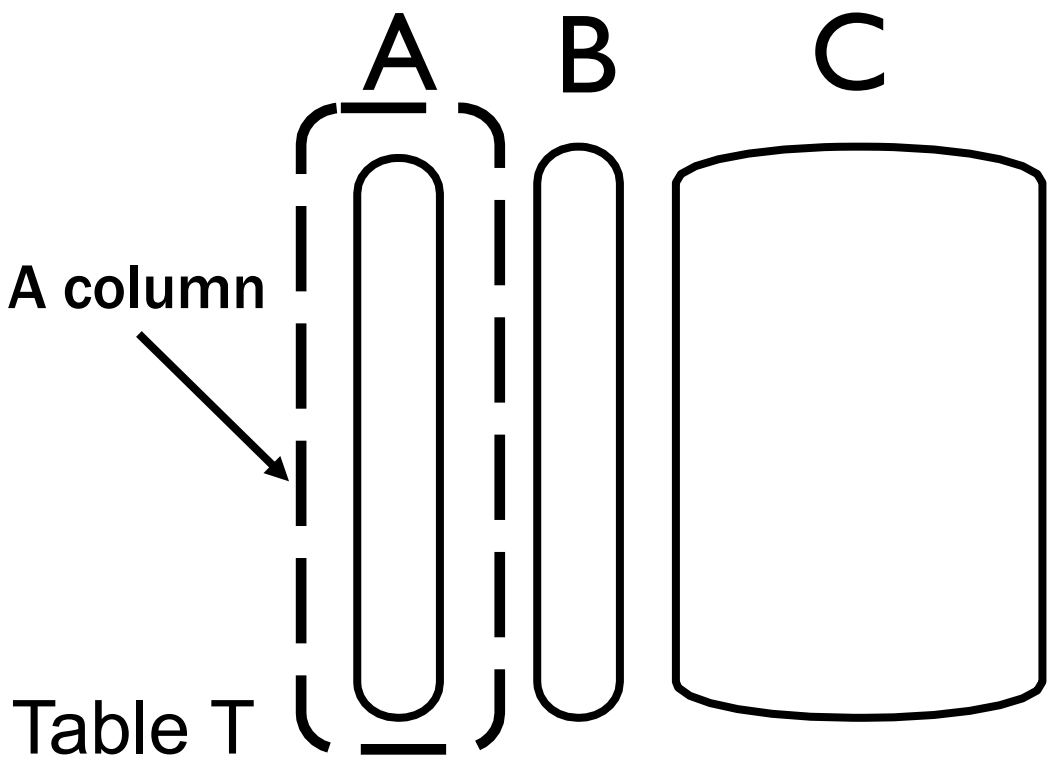
```
int max;
for (int i=0; i<Pos2.size; i++)
{
    if(i==0) max=C[Pos2[i]];
    else if(max<C[Pos2[i]])
    {
        max=C[Pos2[i]];
    }
}
```



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++;
    }
}
```

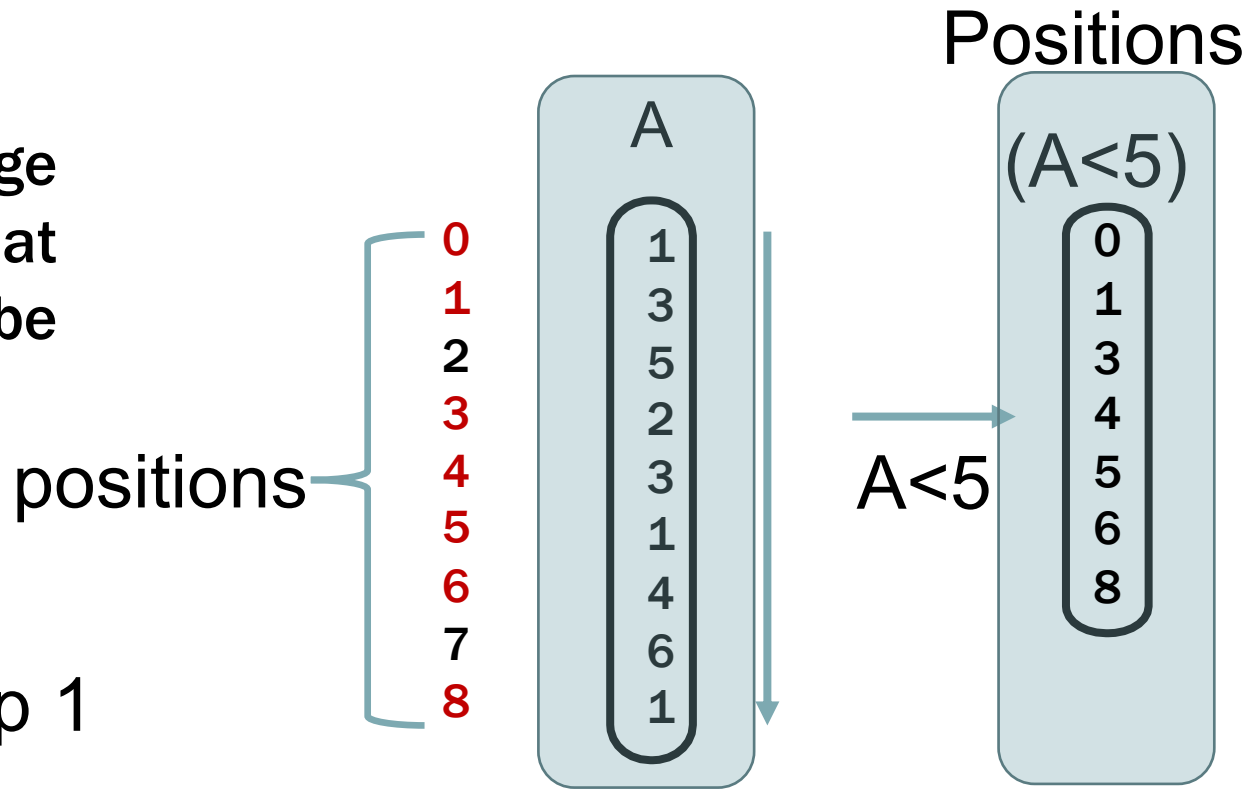
IN-MEMORY V.S. DISK-BASED IMPLEMENTATIONS

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 “FileInputReader”s

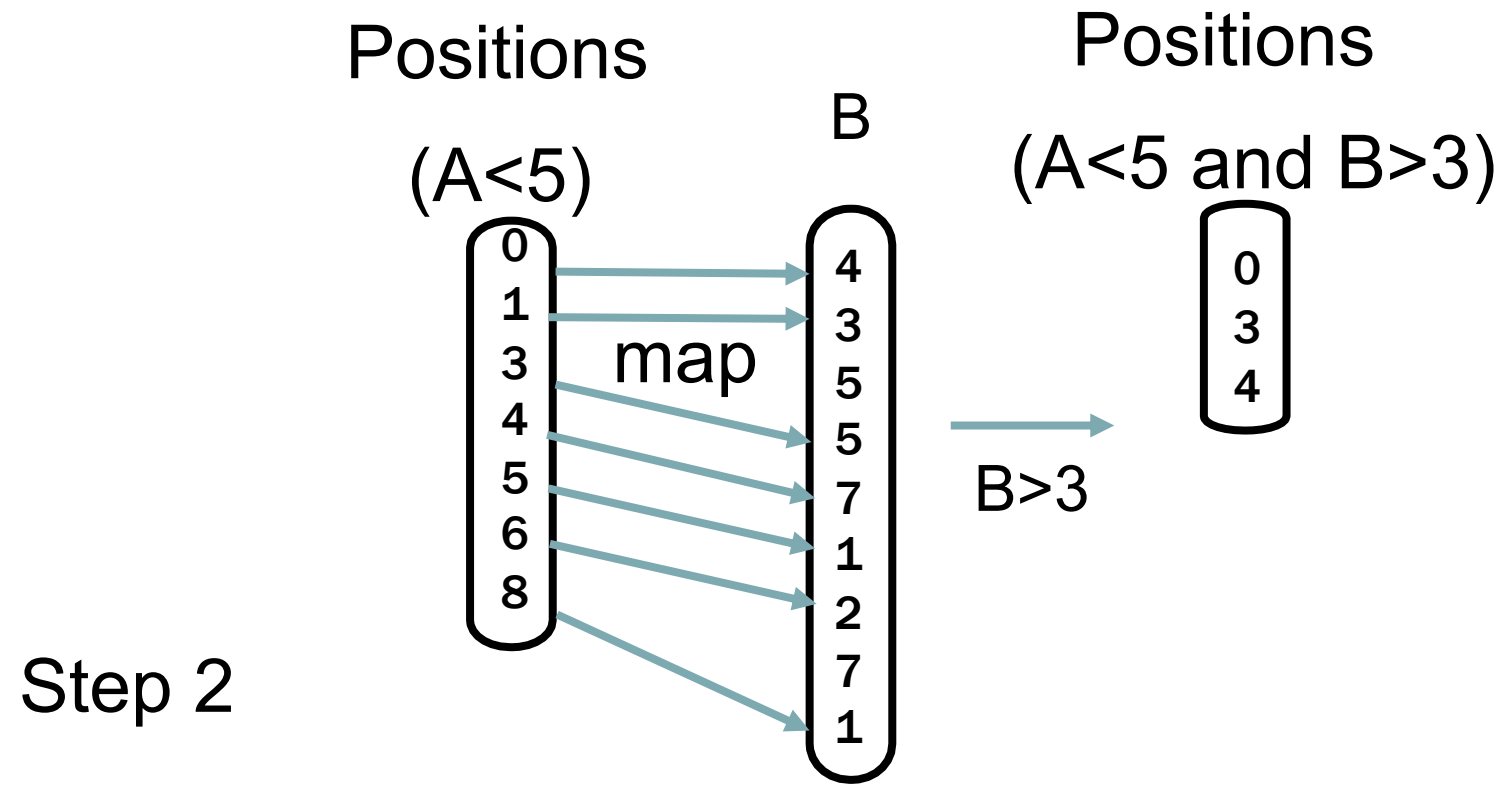
The position list can be overly large (larger than the buffer). If that happens, the position list can also be stored in a file.



IN-MEMORY V.S. DISK-BASED IMPLEMENTATIONS

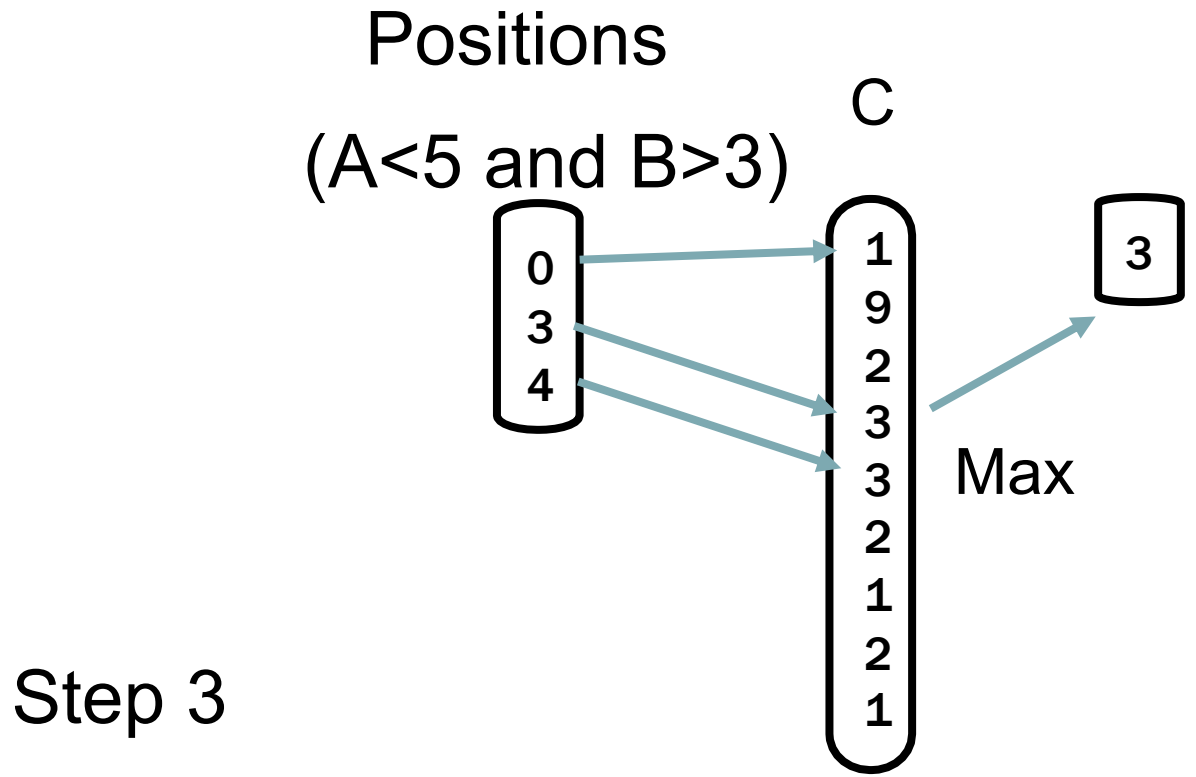
In step 2, the “map” process can be implemented in a disk-based manner. Since column B is stored in a file, when given a position, we need to seek the corresponding values in the file containing column B.

❖ 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.

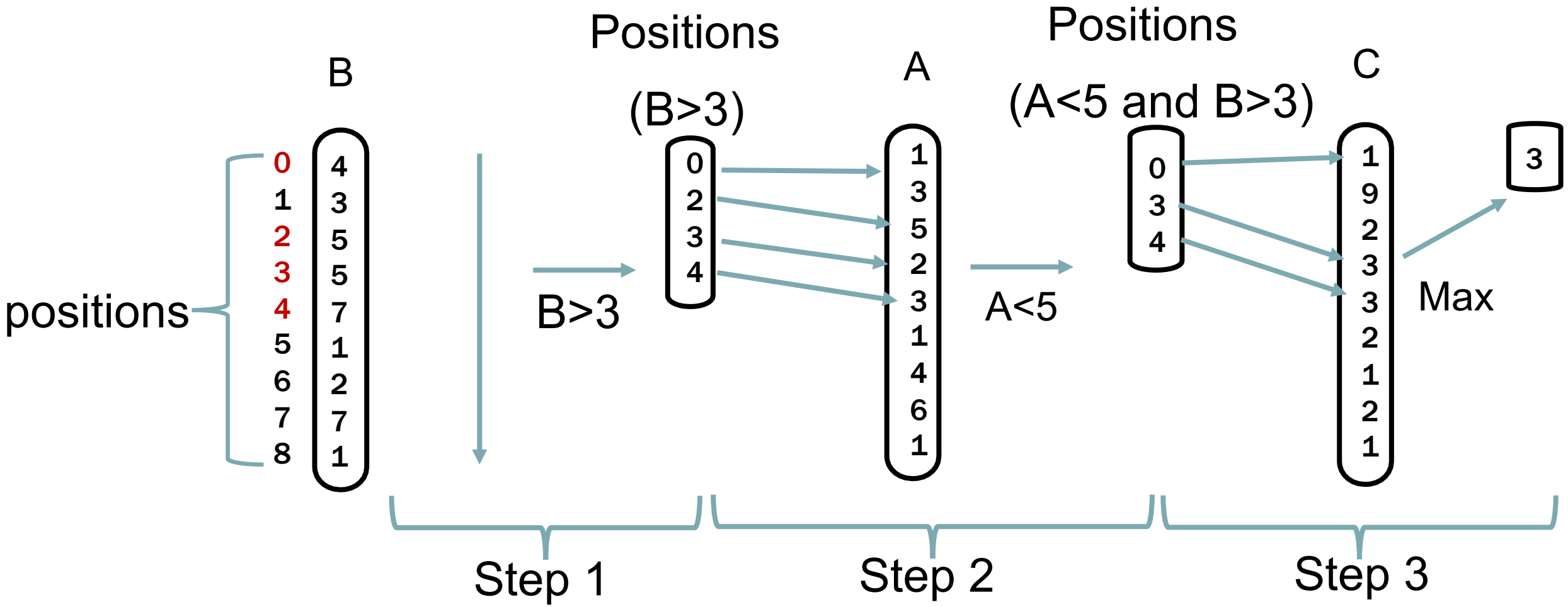


Discuss Alternative Schemes

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?

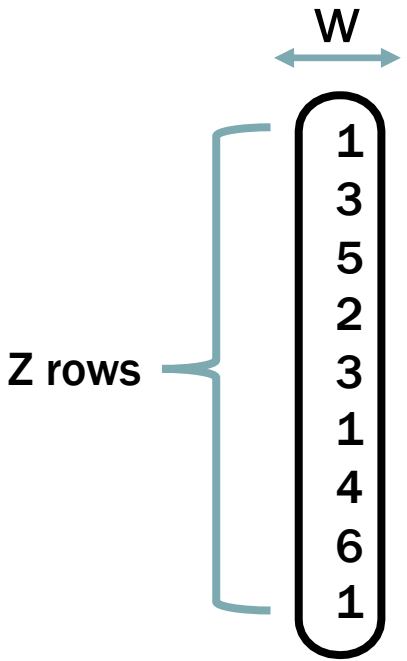
- Usually, Option 3 is not as good as Option 1 (or Option 2) because it needs to scan entire A and B

- 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

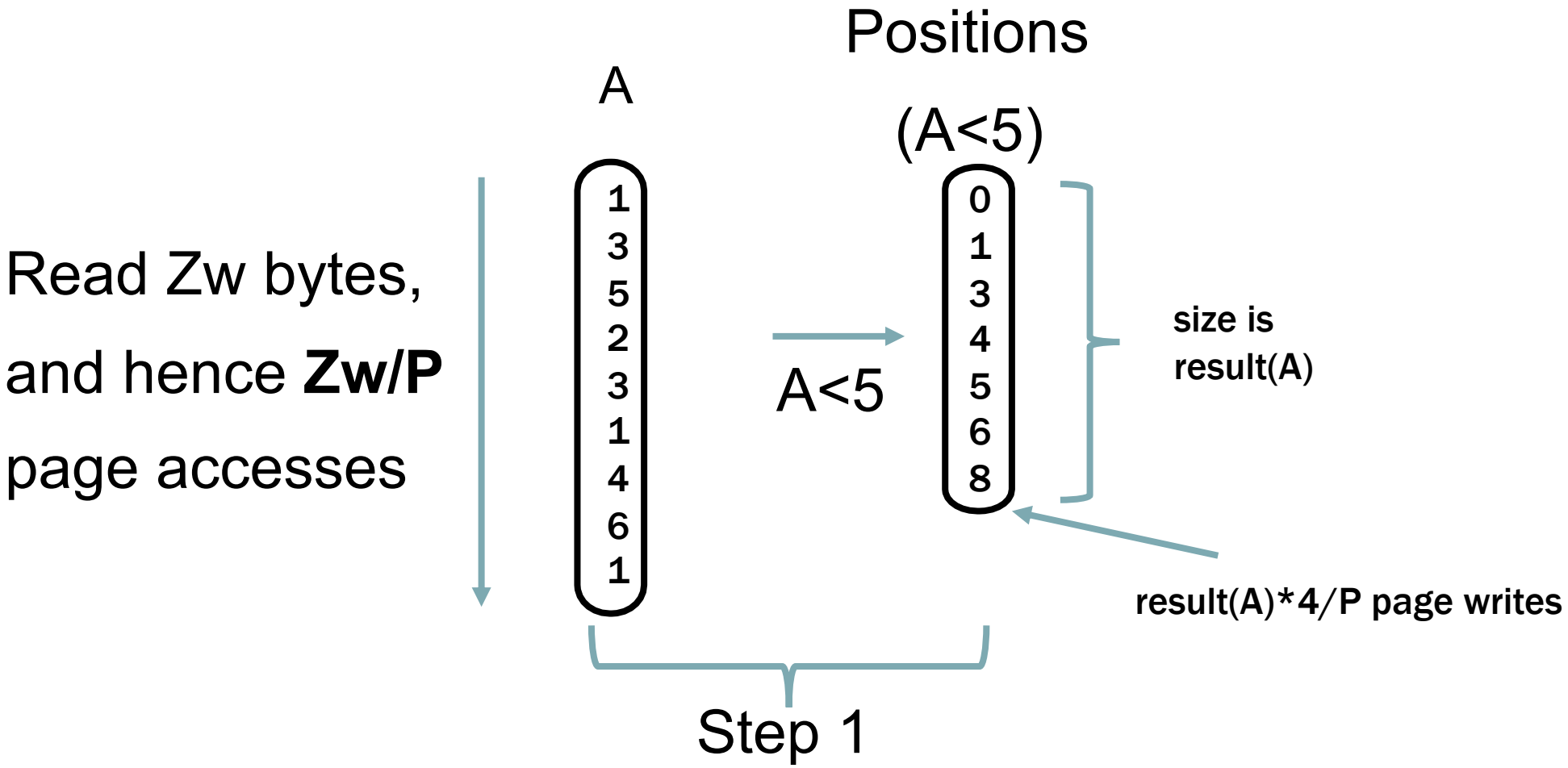
Suppose a column has Z values;
Column width = w bytes;
Page size = P bytes;
Storing a position costs 4 bytes;
Each column is sequentially stored in the disk.
We assume position list is stored in files.
We also assume w < P and each value in a column is contained in a page.



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 result(A)*4/P page writes.
(note: result(A) is the number of qualified positions after processing column A.)

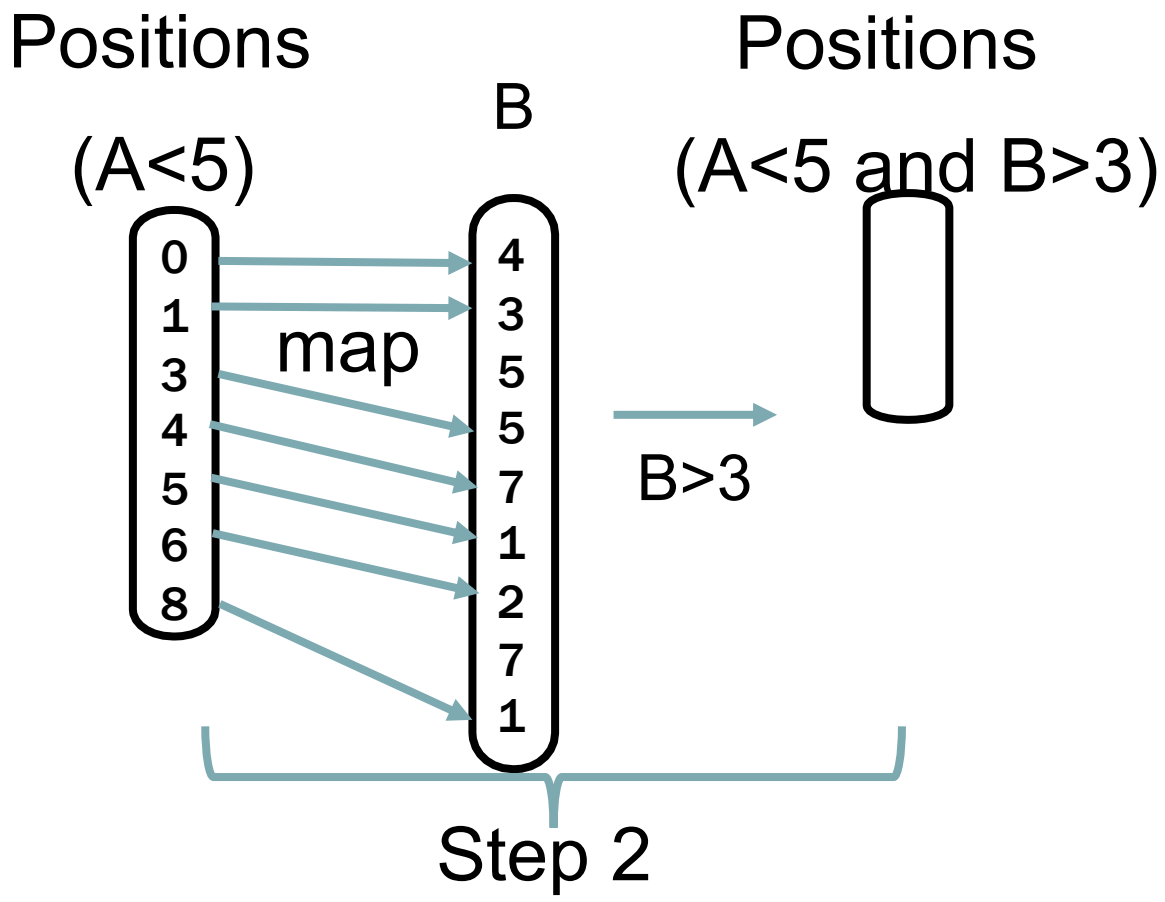


COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Step 2:

Reading the positions (for A<5) costs 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 fetching B costs at most result(A) page reads instead of result(A)/P page reads?

Why the cost of fetching B costs at most result(A) page reads instead of result(A)/P page reads?

Random reads!

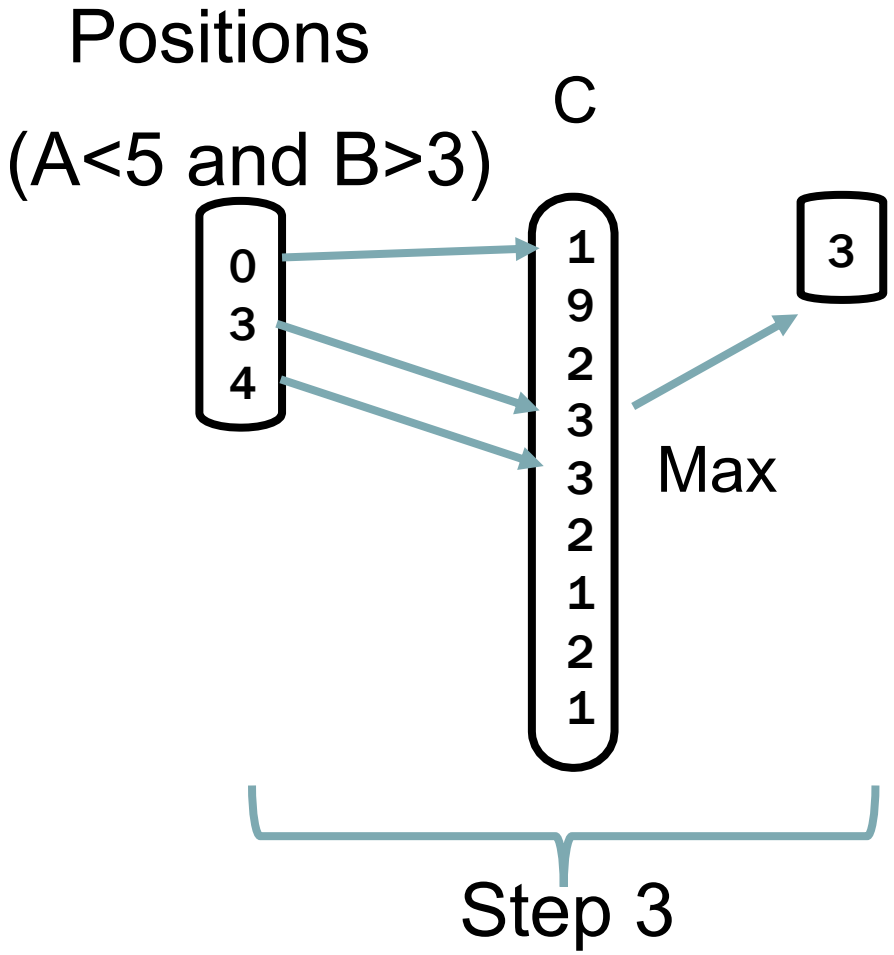
Why the cost of writing positions for B costs result(A)*4/P page writes instead of result(A) page writes?

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

Sequential writes!

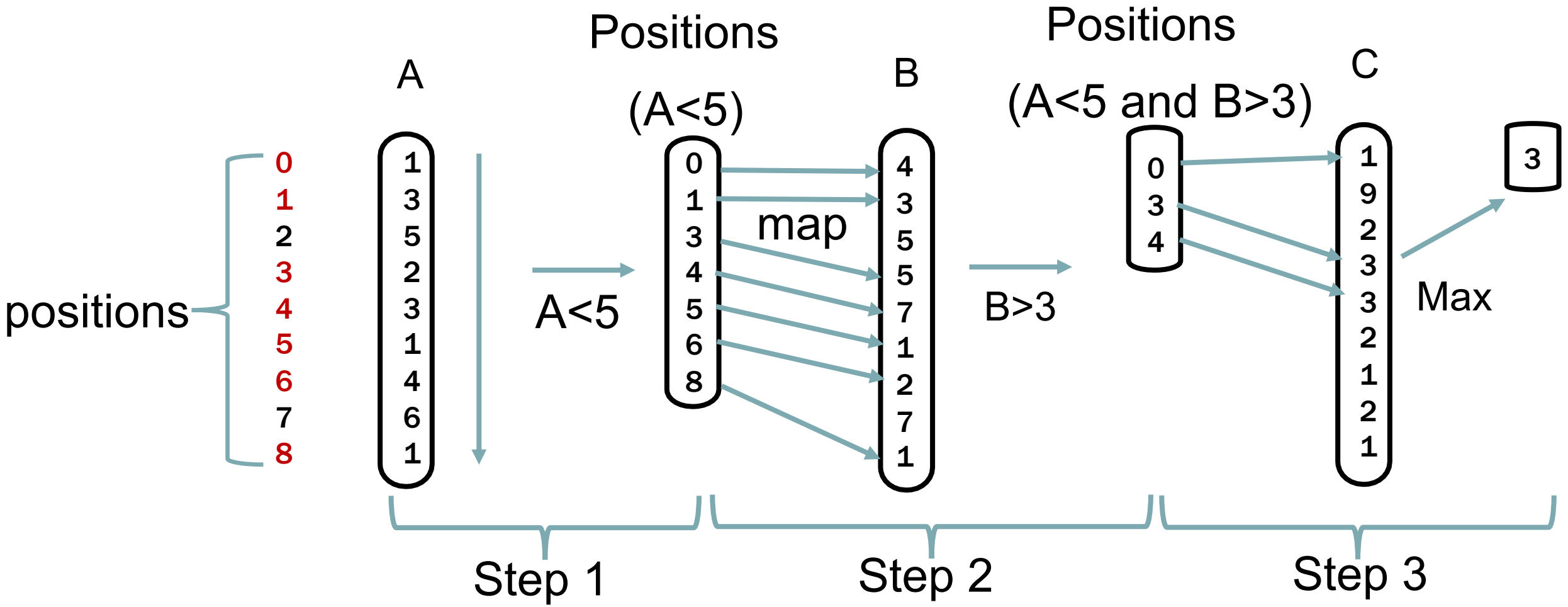
COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

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



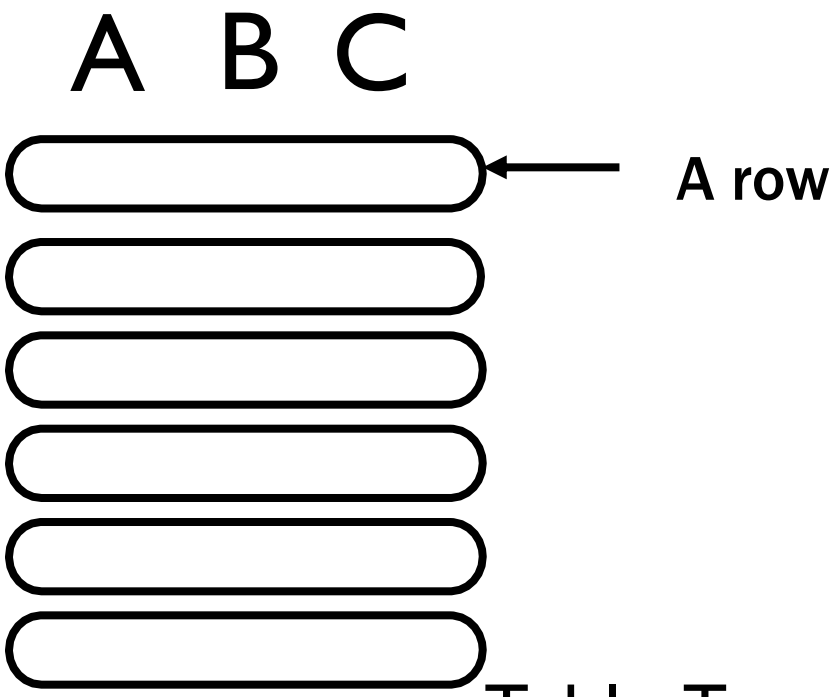
COMPARING QUERYING WITH ROW-STORE AND COLUMN-STORE

Overall: Zw/P+2result(A)*4/P+result(A)+2result(AB)*4/P+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*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)
≈ 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.

REMARK

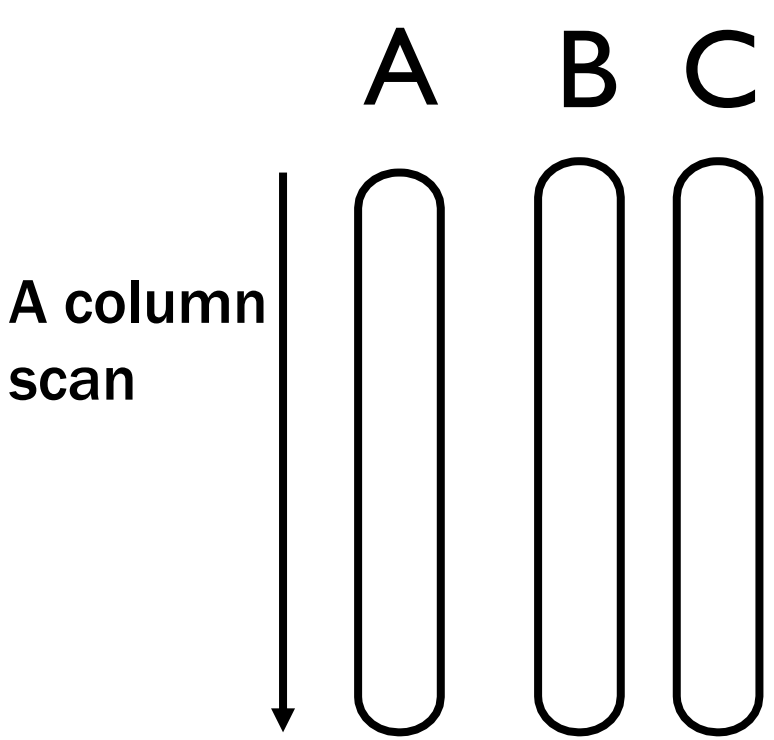
In many cases, position list can be small enough to be simply stored in the memory buffer

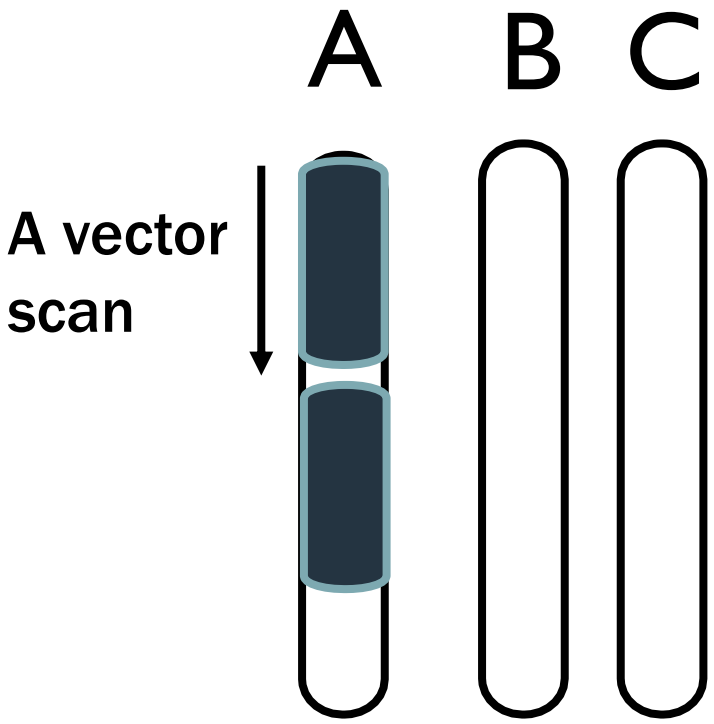
In these cases, cost for column store (number of page access):
Zw/P+~~2result(A)*4/P~~+result(A)+~~2result(AB)*4/P~~+result(AB)
=Zw/p+result(A)+result(AB)

Cache Optimization (extended discussion)

CACHE OPTIMIZATION: VECTOR AT A TIME

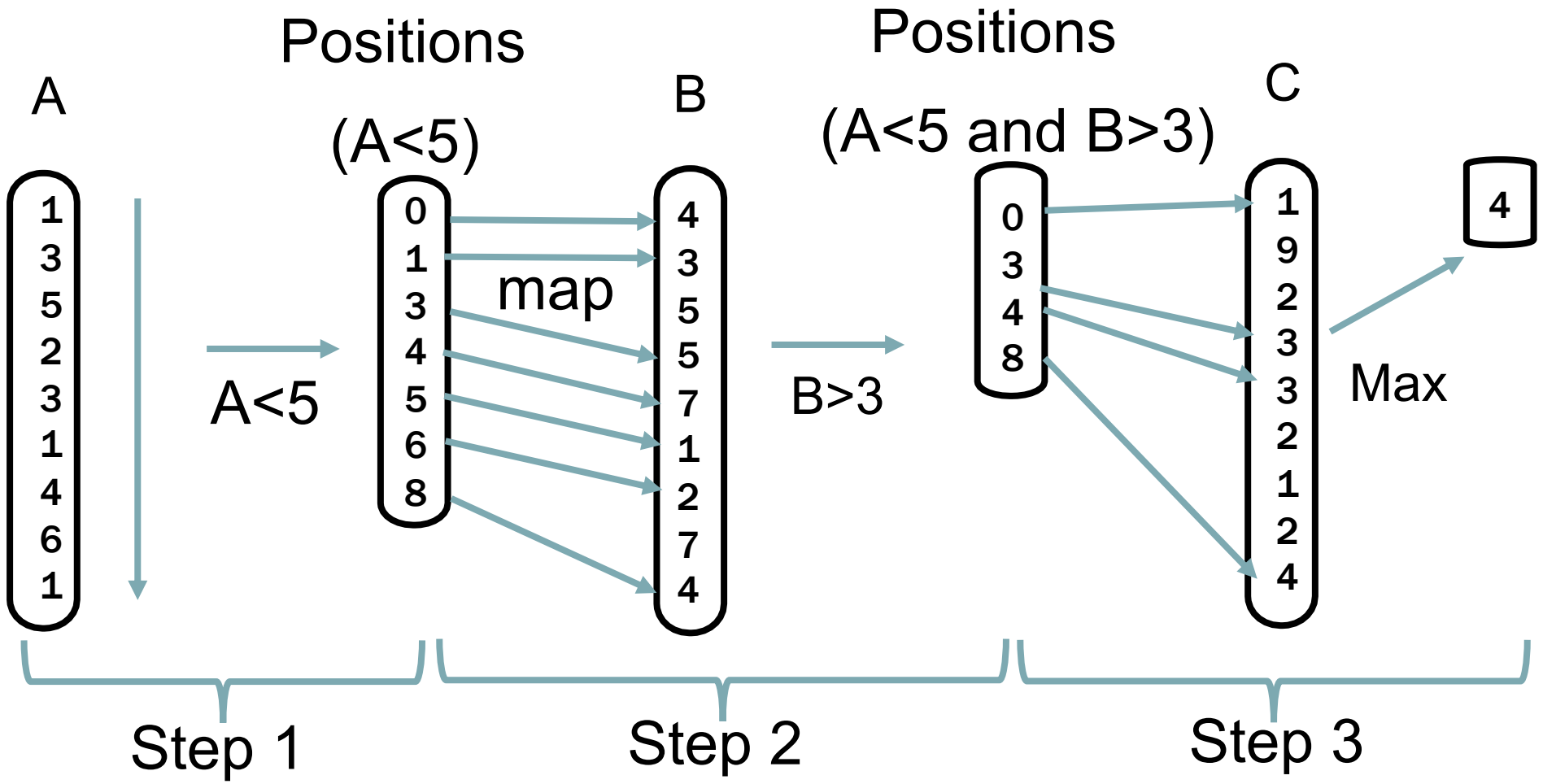
Modern column stores (e.g., VectorWise) employ vector-at-a-time procedure, utilizing cache locality. This is called vectorized processing.



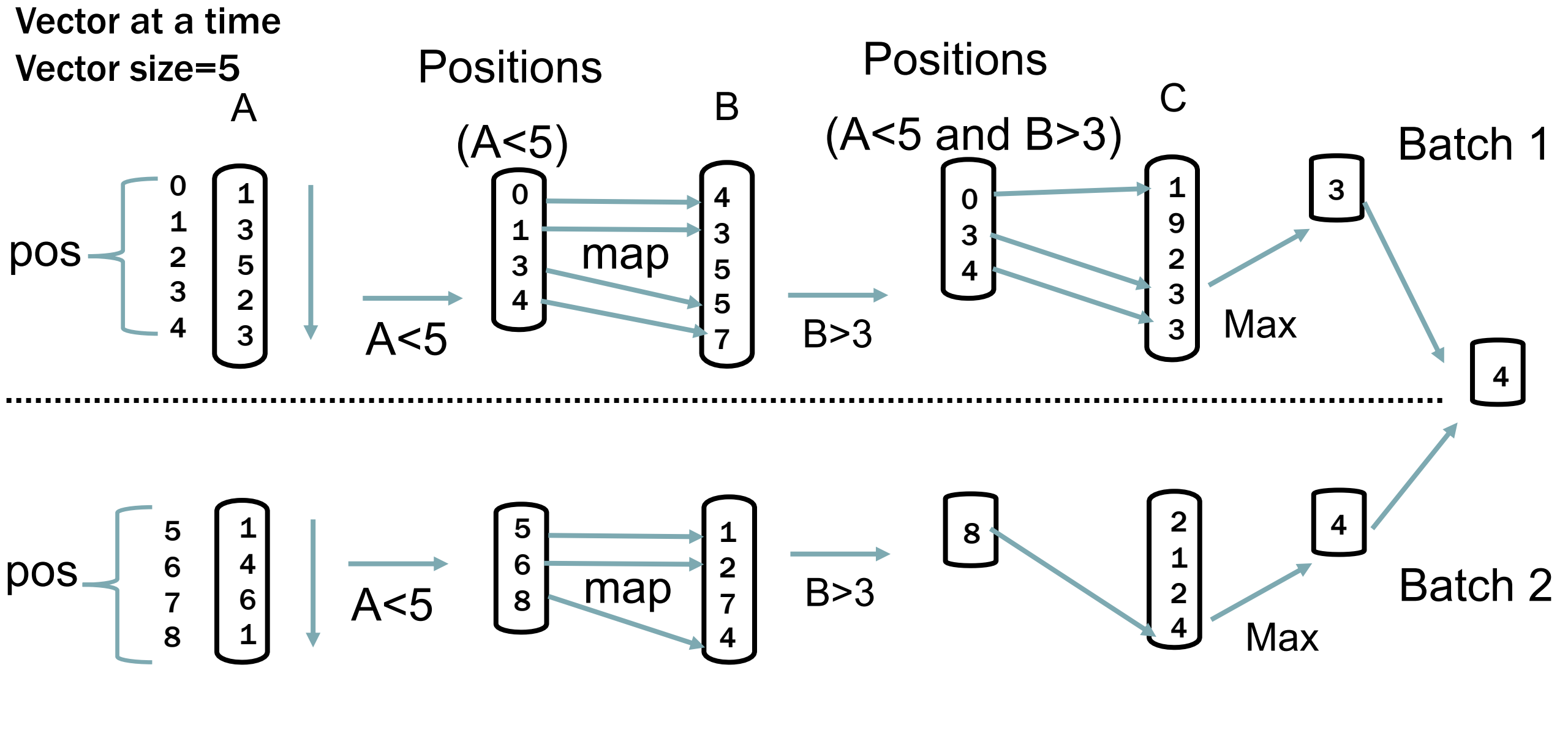


EXAMPLE

Column at a time



EXAMPLE

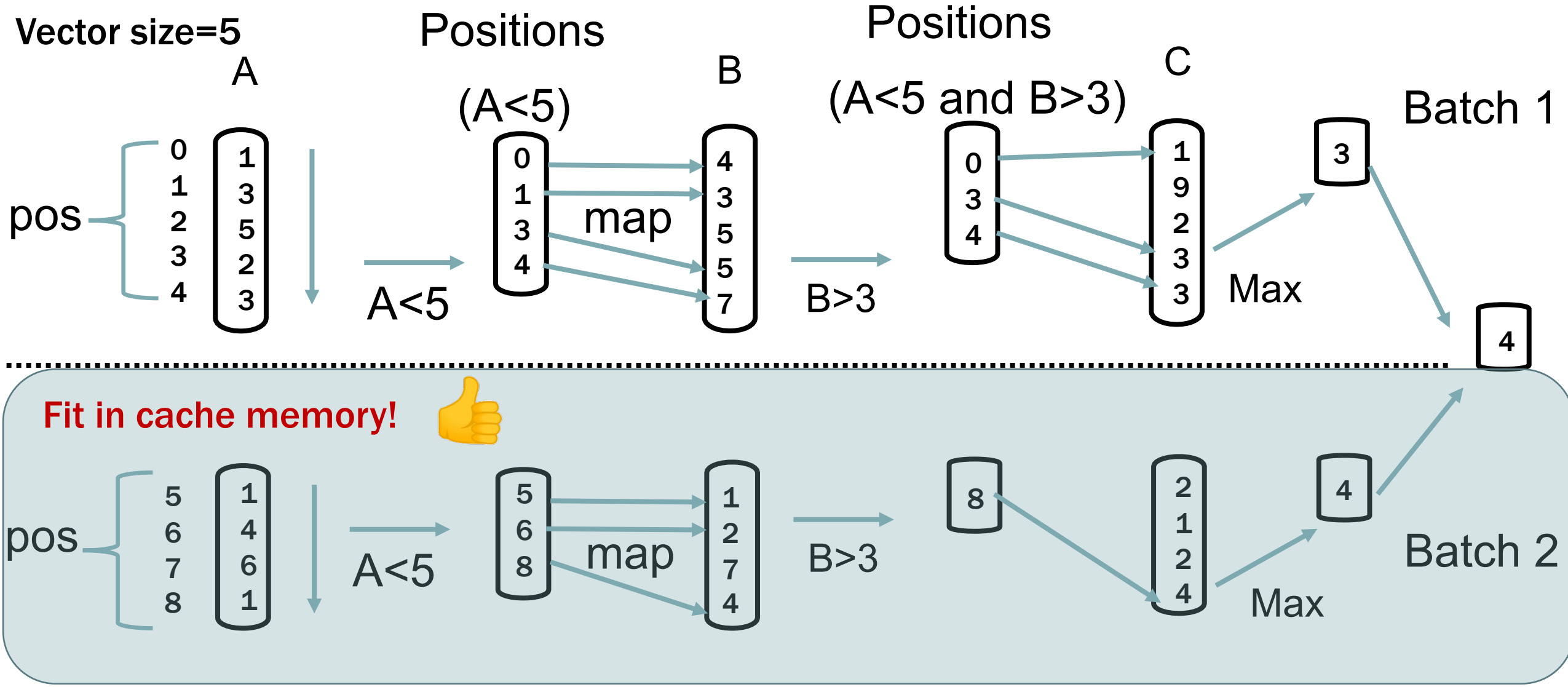

Vector at a time
Vector size=5
A
Positions
(A<5)
B
Positions
(A<5 and B>3)
C
Batch 1
pos
0
1
2
3
4
1
3
5
2
3
A<5
0
1
3
4
map
4
3
5
5
7
B>3
0
3
4
1
9
2
3
3
Max
3
4
pos
5
6
7
8
1
4
6
1
A<5
5
6
8
map
1
2
7
4
B>3
8
2
1
2
4
4
Max
Batch 2

What’s the benefits of vectorized processing?

EXAMPLE

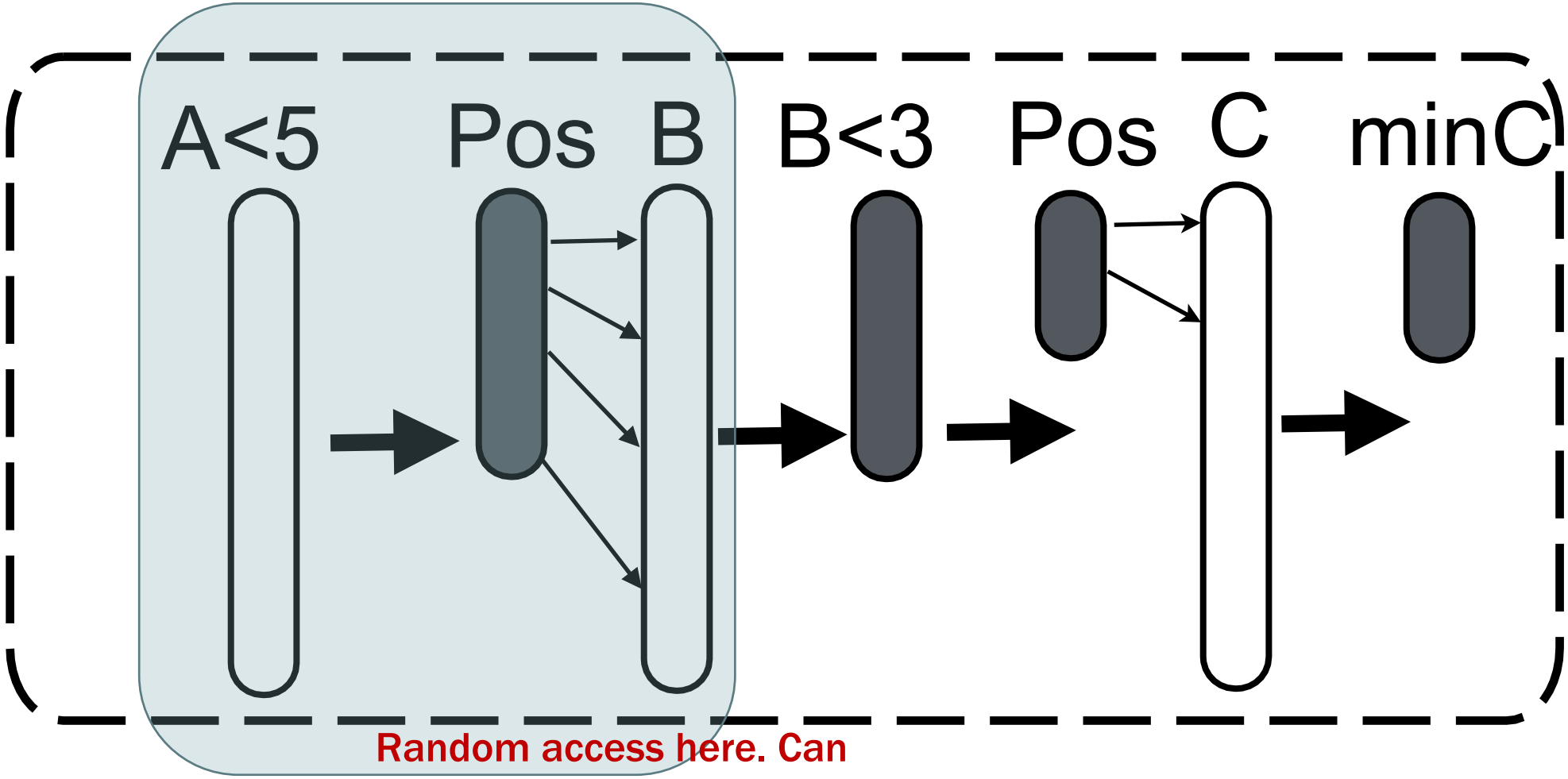
Vector at a time

Vector size=5



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?