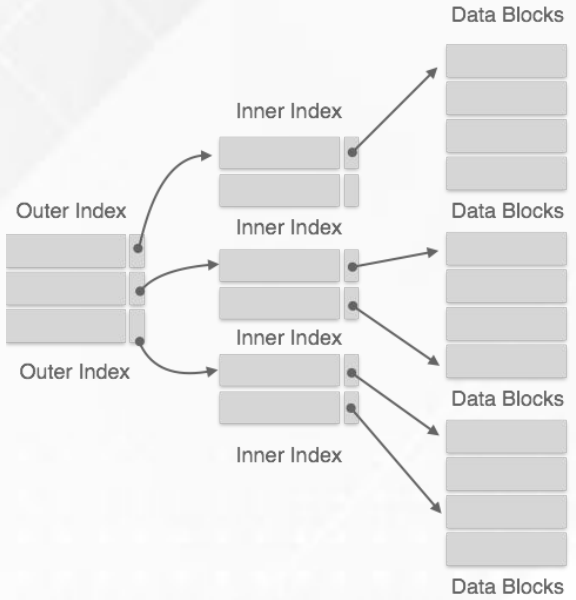
Unit-4

Hashing & File Structure (File Structure)



What is File?

- A **file** is a **collection of records** where a record consists of one or more fields. Each contains the same sequence of fields.
- Each **field** is normally of **fixed length**.
- A sample file with four records is shown below:

Name	Roll No.	Year	Marks
AMIT	1000	1	82
KALPESH	1005	2	54
JITENDRA	1009	1	75
RAVI	1010	1	79

- There are **four records**
- There are four fields (Name, Roll No., Year, Marks)
- Records can be uniquely identified on the field 'Roll No.' Therefore, **Roll No. is the key field**.
- A database is a collection of files.

File Organizations

File Organizations

1. Sequential files
2. Relative files
3. Direct files
4. Indexed Sequential files
5. Index files

Primitive Operations on a File

1. Creation
2. Reading
3. Insertion
4. Deletion
5. Updation
6. Searching

Sequential Files

- It is the most common type of file.
- A **fixed format** is used for **record**.
- All **records** are of the **same length**.
- **Position** of each **field** in record and **length** of field is **fixed**.
- Records are **physically ordered** on the value of one of the fields - called the **ordering field**.

Block 1

Name	Roll No.	Year	Marks
AMIT	1000	1	82
KALPESH	1005	1	54
JITENDRA	1009	1	75
RAVI	1010	1	79

Block 2

Name	Roll No.	Year	Marks
RAMESH	1015	1	75
ROHIT	1025	1	65
JANAK	1026	1	75
AMAR	1029	1	79

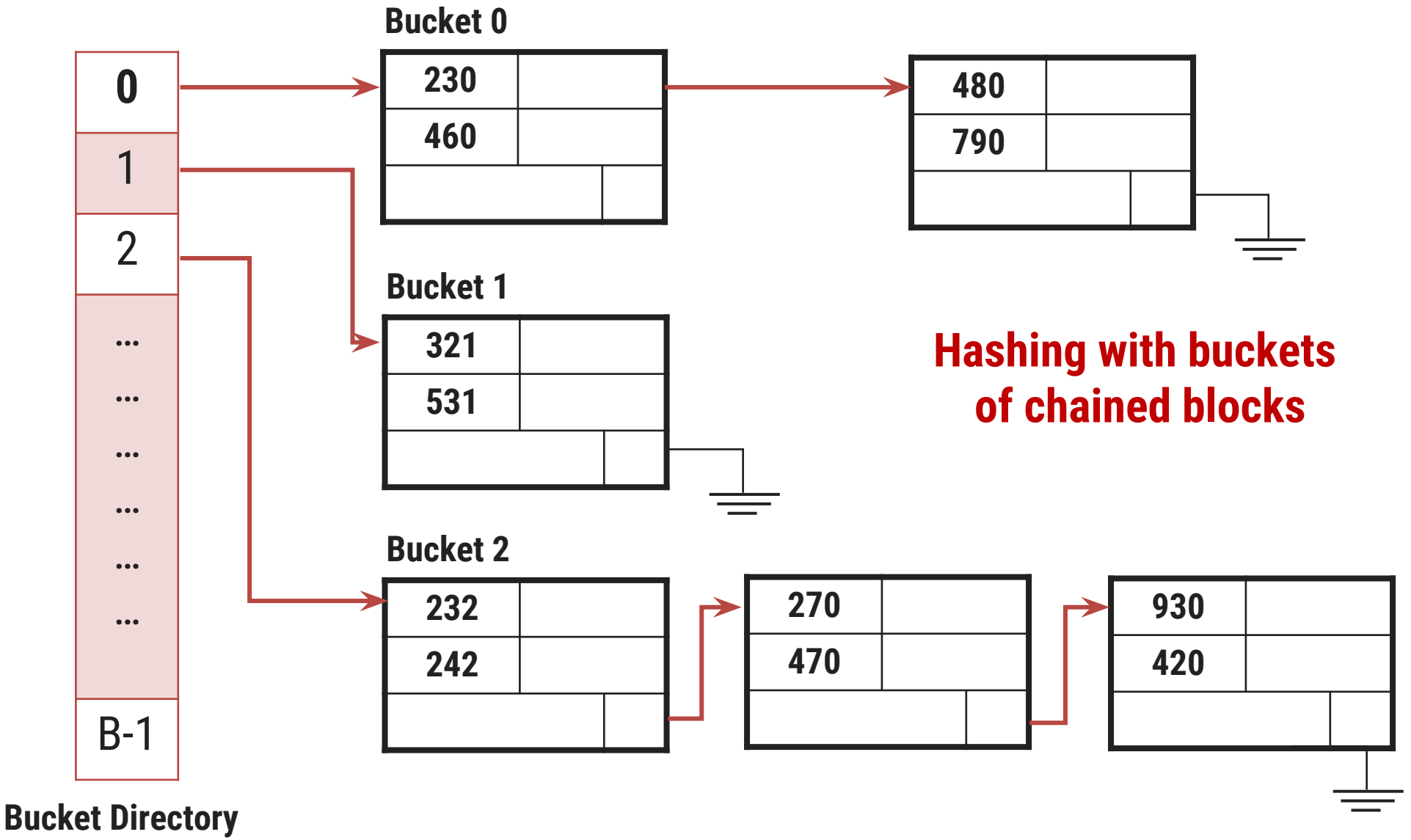
Advantages of Sequential Files

- **Reading** of records **in** order of the **ordering key** is extremely **efficient**.
- **Finding** the **next record in** order of the **ordering key** usually, does **not require additional block** access. Next record may be found in the same block.
- **Searching** operation **on ordering key** is must **faster**. **Binary search can be utilized**. A binary search will require $\log_2 b$ block accesses where b is the total number of blocks in the file.

Disadvantages of Sequential Files

- Sequential file **does not** give any **advantage** when the **search** operation is to be carried out **on non- ordering field**.
- **Inserting** a **record** is an **expensive operation**. Insertion of a new record requires finding of place of insertion and then all records ahead of it must be moved to create space for the record to be inserted. This could be very expensive for large files.
- **Deleting** a **record** is an **expensive operation**. Deletion too requires movement of records.
- **Modification** of **field value** of **ordering key** could be **time consuming**. Modifying the ordering field means the record can change its position. This requires deletion of the old record followed by insertion of the modified record.

Hashing (Direct file organization)



Hashing (Direct file organization)

- It is a common technique used for **fast accessing of records** on secondary storage.
- **Records** of a file are **divided among buckets**.
- A **bucket** is either **one disk block** or **cluster of contiguous blocks**.
- A **hashing function** maps a **key** into a **bucket number**. The buckets are numbered 0, 1,2...b-1.
- A hash function **f** maps each **key** value into one of the integers **0 through b - 1**.
- If **x is a key**, **f(x)** is the number of **bucket** that contains the record **with key x**.
- The blocks making up each bucket could either be contiguous blocks or they can be chained together in a linked list.

Hashing (Direct file organization)

- Translation of bucket number to disk block address is done with the help of **bucket directory**. It **gives** the **address of the first block** of the chained blocks in a linked list.
- Hashing is quite efficient in retrieving a record on hashed key. The average number of block accesses for retrieving a record.

$$= \textbf{1 (bucket directory) +} \frac{No\ of\ records}{No\ of\ buckets\ x\ No\ of\ records\ per\ block}$$

- Thus the **operation is b times faster** (b = number of buckets) than unordered file.
- To **insert a record with key value x**, the **new record** can added to the **last block** in the chain for bucket f(x). If the record does not fit into the existing block, record is stored in a new block and this new block is added at the end of the chain for bucket f(x).
- A well designed hashed structure requires two block accesses for most operations

Indexing

- **Indexing** is **used** to **speed** up **retrieval** of records.
- It is done with the help of a **separate sequential file**.
- **Each record** of in the **index file** consists of two fields, a key field and a pointer into the main file.
- To **find** a specific **record** for the given **key value**, **index** is **searched** for the given key value.
- **Binary search** can **used** to search in **index file**. After getting the address of record from index file, the record in main file can easily be retrieved.

Indexing

Index File

Keyc	
1000	
1009	
1010	
1012	
1016	
1089	
1100	
1200	

Main File

Name	Roll No.	Year	Marks
AMIT	1010	1	82
KALPESH	1016	1	54
JITENDRA	1000	1	75
RAVI	1012	1	79
NILESH	1089	1	85
NITIN	1100	1	98
JAYESH	1200	1	99
UMESH	1009	1	74

Index file is ordered on the ordering key Roll No. each record of index file points to the corresponding record. Main file is not sorted.

Advantages of Indexing

- Sequential file can be searched effectively on ordering key. When it is necessary to **search** for a **record** on the **basis of** some **other attribute** than the ordering key field, sequential file representation is inadequate.
- **Multiple indexes** can be **maintained** for **each** type of **field** used for searching. Thus, indexing provides much better flexibility.
- An **index file** usually requires **less storage** space than the main file.
- A **binary search** on **sequential file** will **require** accessing of **more blocks**.
- This can be explained with the help of the following example.
- Consider the example of a sequential file with r = **1024 records** of fixed length with **record size R = 128 bytes** stored on disk with **block size B = 2048 bytes**.

Advantages of Indexing

Size of Sequential File

- **Number of blocks** required to **store the file**
  - **(1024 x 128) / 2048 = 64**
- **Number of block accesses** for **searching** a record
  - **$\log_2 64 = 6$**

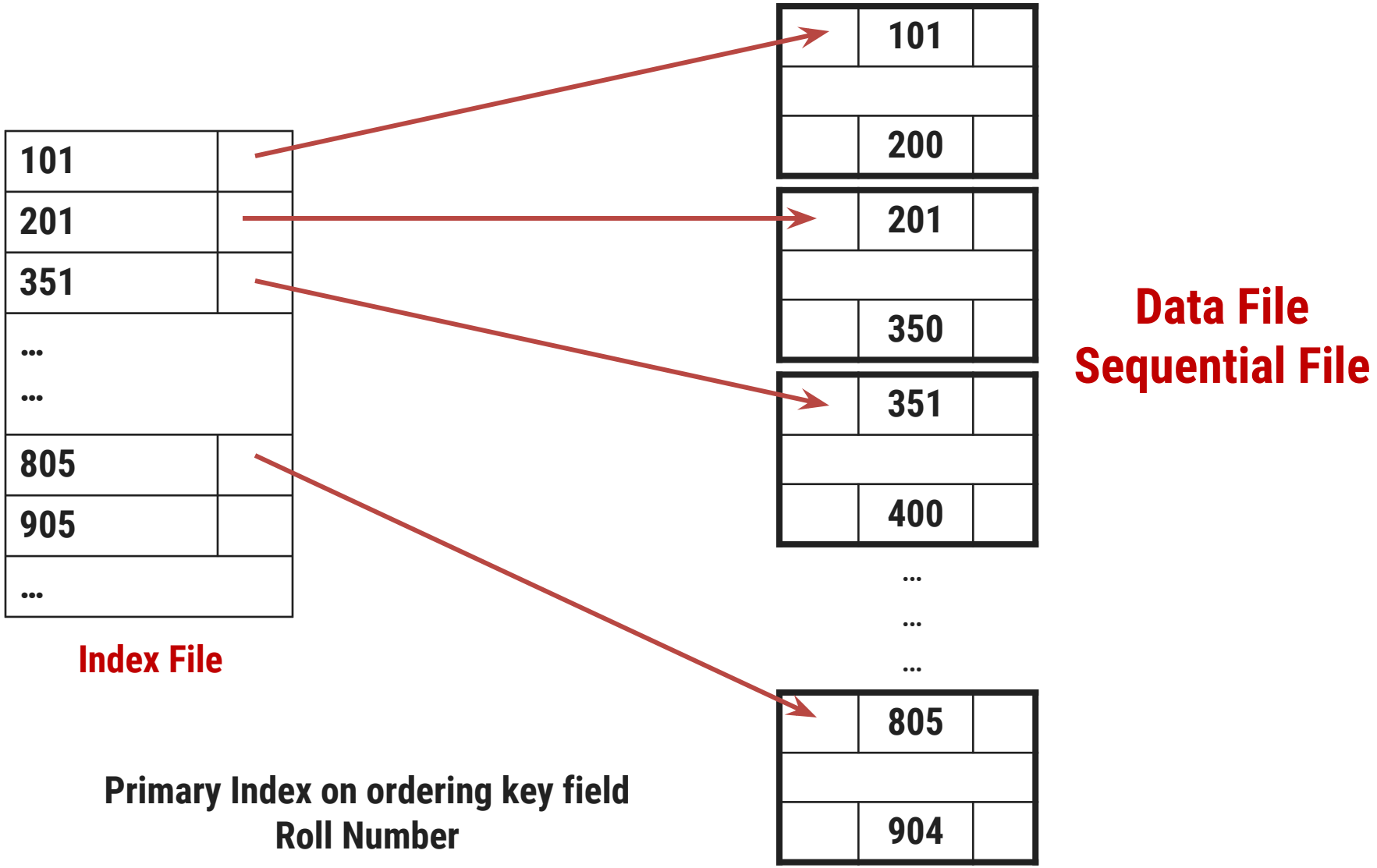
Size of Index File

- Suppose, we want to **construct an index** on a **key field** that is V = **4 bytes** long and the block **pointer is P = 4 bytes long**.
- A **record** of an **index file** needs **8 bytes** per entry.
- Total **Number** of **index entries** = **1024**
- **Number of blocks required to store the index file**
  - **(1024x8) / 2048 = 4**
- **Number of block accesses for searching a record = $\log_2 4 = 2$**

Types of Indexes

- With indexing, **new records** can be **added** at the **end of the main file**. It will not require movement of records as in the case of sequential file.
- **Updation of index file requires fewer block** accesses compare to sequential file
- Types of Indexes:
  1. Primary indexes
  2. Clustering indexes
  3. Secondary indexes

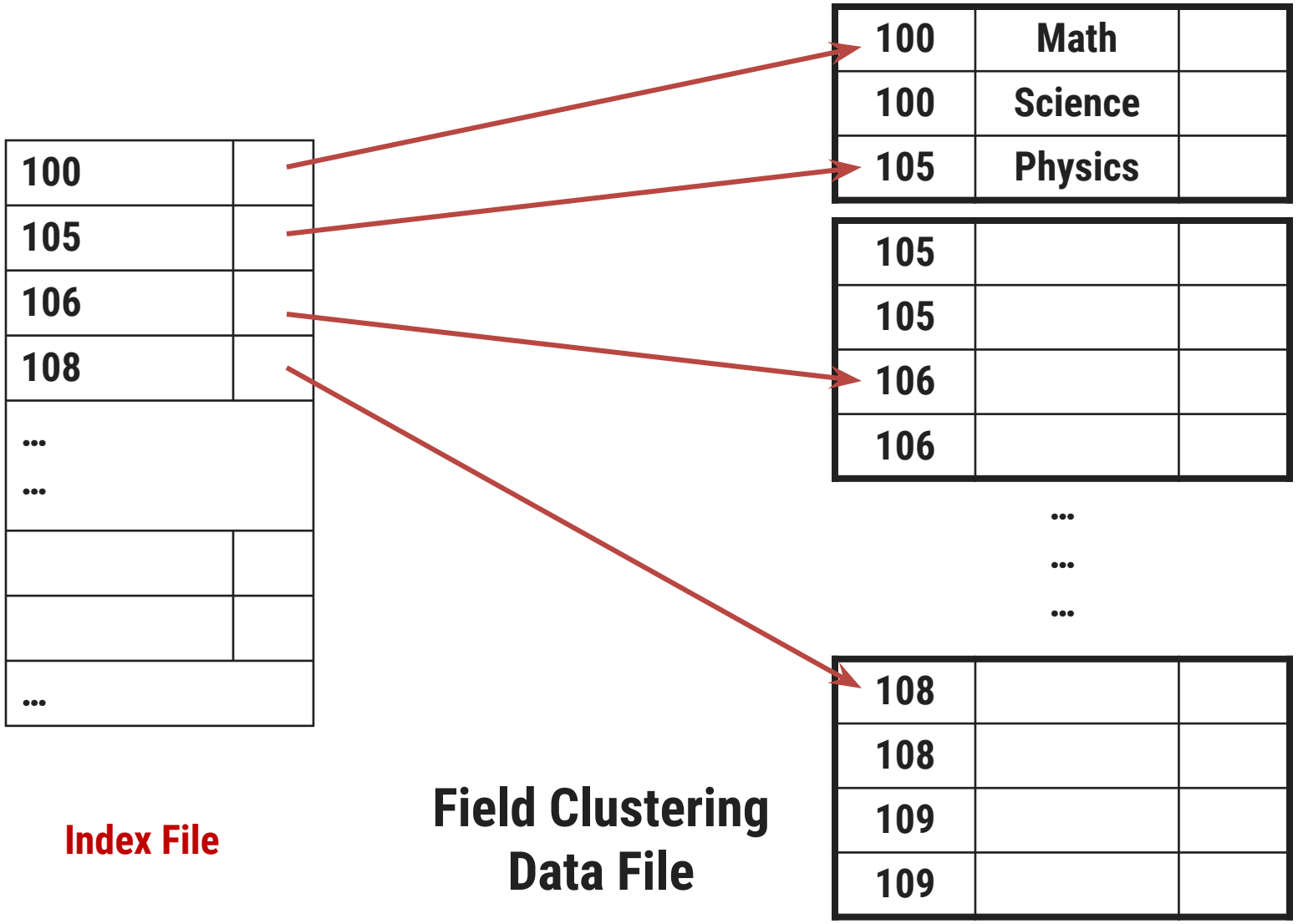
Primary Indexes (Indexed Sequential File)



Primary Indexes (Indexed Sequential File)

- An **indexed sequential** file is **characterized** by
  - Sequential organization (ordered on primary key)
  - Indexed on primary key
- **An indexed sequential** file is both **ordered** and **indexed**.
- **Records** are **organized** in **sequence** based **on** a **key field,** known as **primary key**.
- An **index** to the file is **added** to **support random access**. Each record in the index file consists of two fields: a key field, which is the same as the key field in the main file.
- **Number of records** in the **index file** is equal to the **number of blocks** in the **main file** (data file) and not equal to the number of records in the main file (data file).

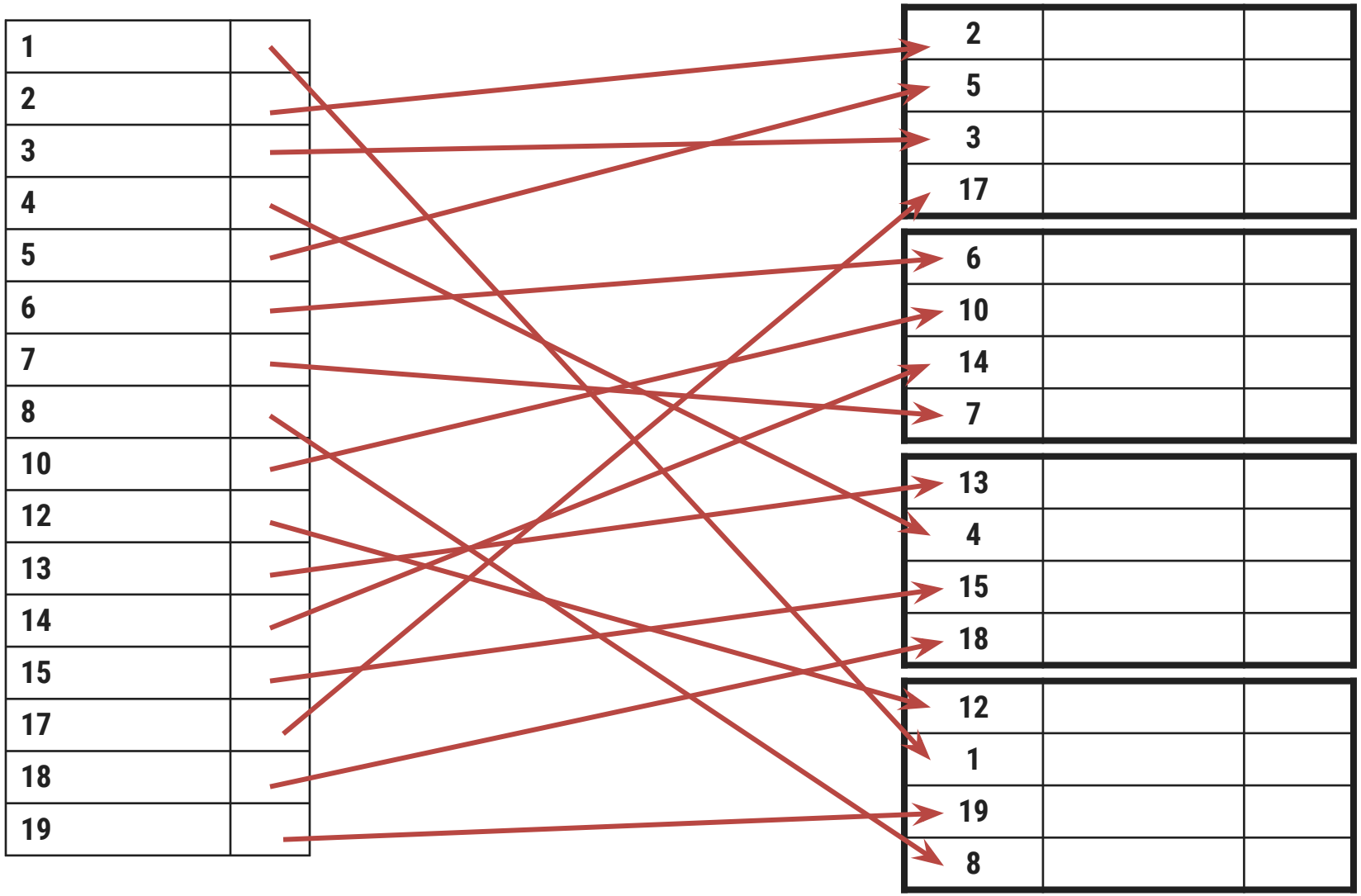
Clustering Indexes



Clustering Indexes

- If **records** of a file are **ordered on a non-key field**, we can create a different type of index known as **clustering index**.
- A non-key field does not have distinct value for each record.
- A Clustering index is also an ordered file with two fields.

Secondary Indexes (Simple Index File)



A secondary index on a non-ordering key field

Secondary Indexes (Simple Index File)

- While the **hashed, sequential** and **indexed sequential** files are suitable for operations based on ordering key or the hashed key. Above file organizations are **not suitable** for operations involving a search on **a field other than ordering or hashed key**.
- If searching is required on various keys, **secondary indexes** on these fields must be **maintained**.
- A **secondary index** is an ordered file with **two fields**.
  - Some **non-ordering field** of the data file.
  - A **block pointer**
- There could be **several secondary indexes** for the **same file**.
- One **could use binary search** on index file as entries of the index file are ordered on secondary key field.
- Records of the **data files are not ordered** on secondary key field.

Secondary Indexes (Simple Index File)

- A **secondary index** requires **more storage** space and **longer search time** than does a **primary index**.
- A secondary index file has an **entry for every record** whereas primary index file has an entry for every block in data file.
- There is **a single primary index** file but the **number of secondary indexes** could be quite a few.

Thank You