Chapter 4

Data Storage & Access Methods

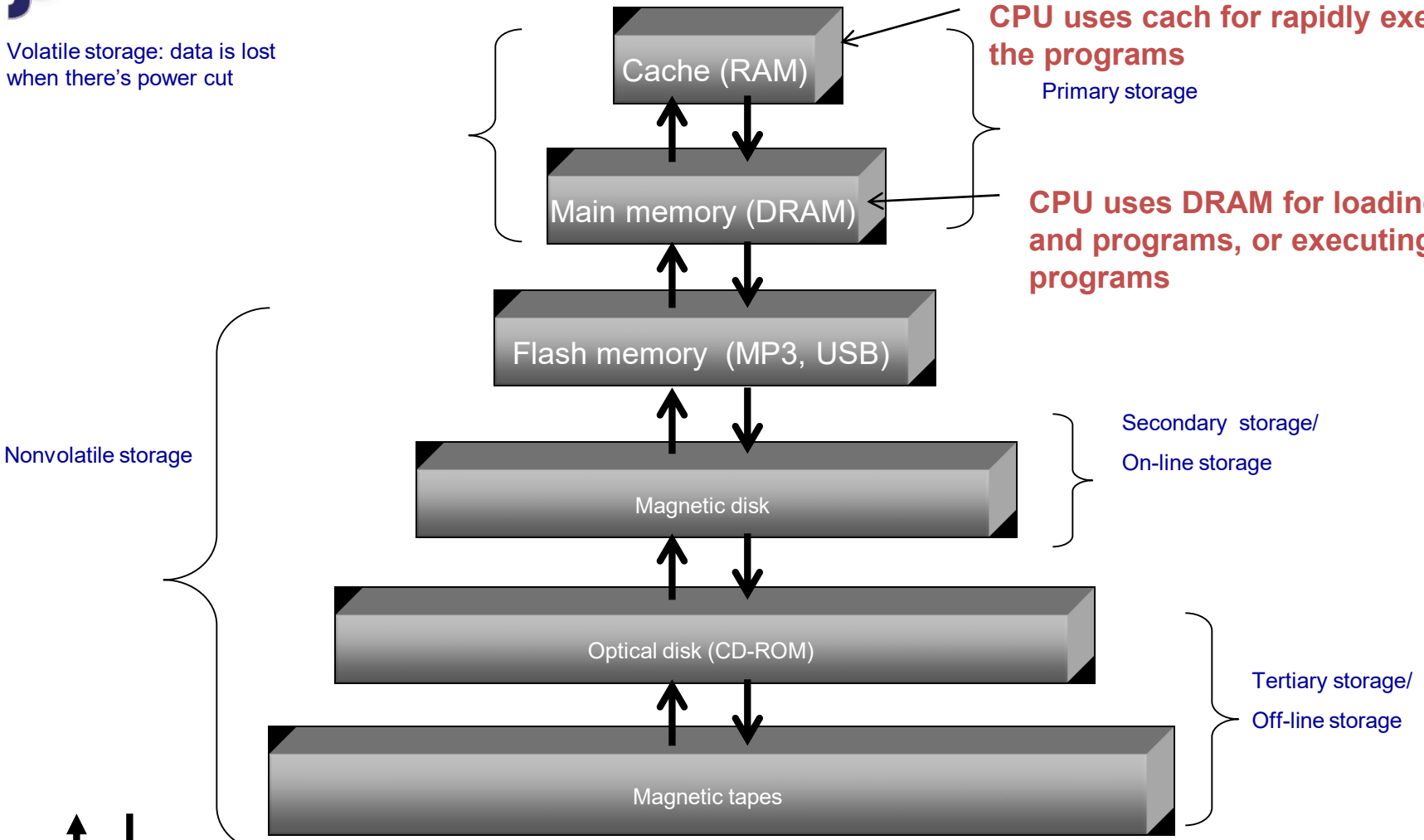
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Outline

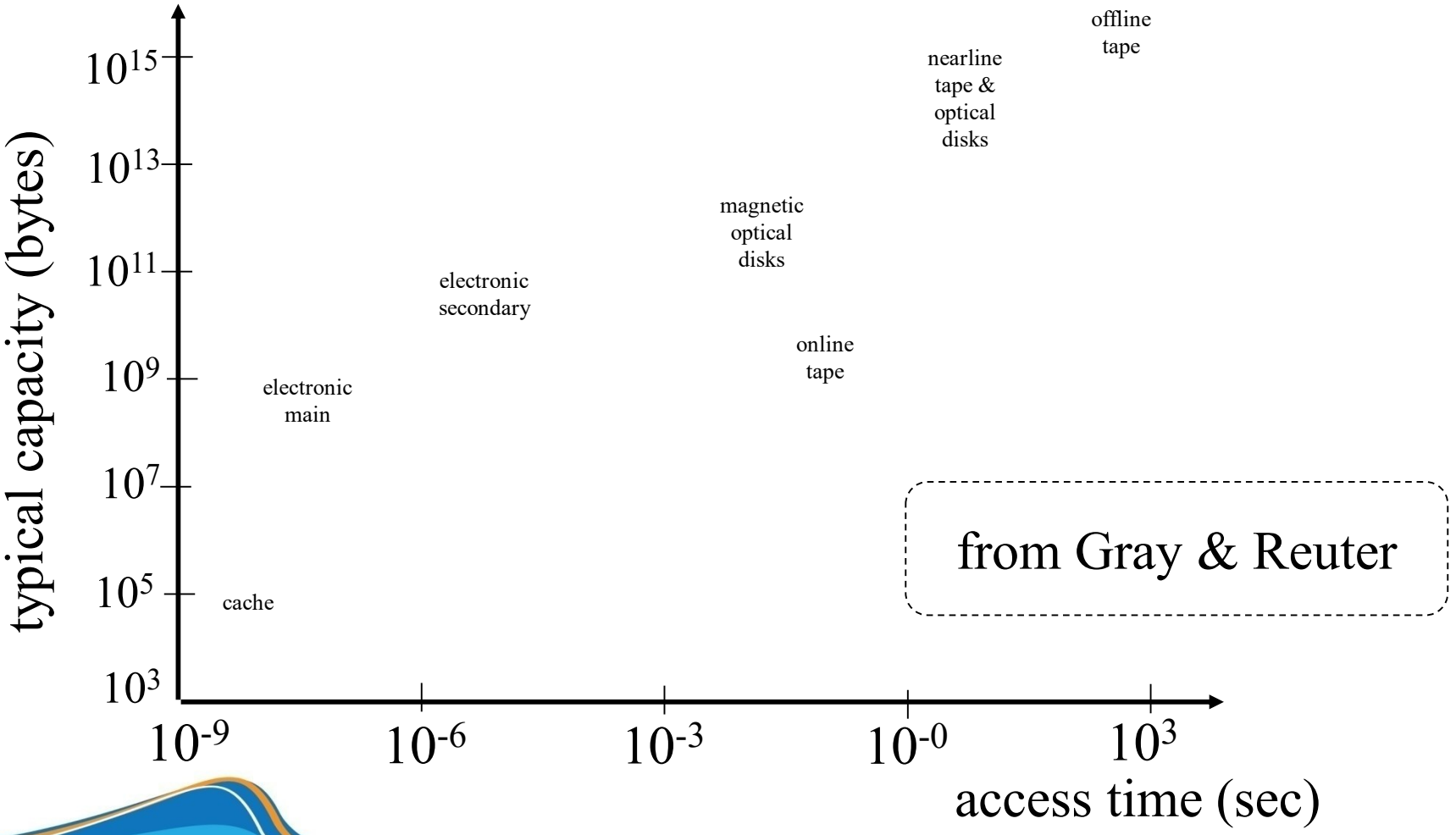
1. Concepts
2. Basic file structures and hashing
3. Index

Secondary storage devices

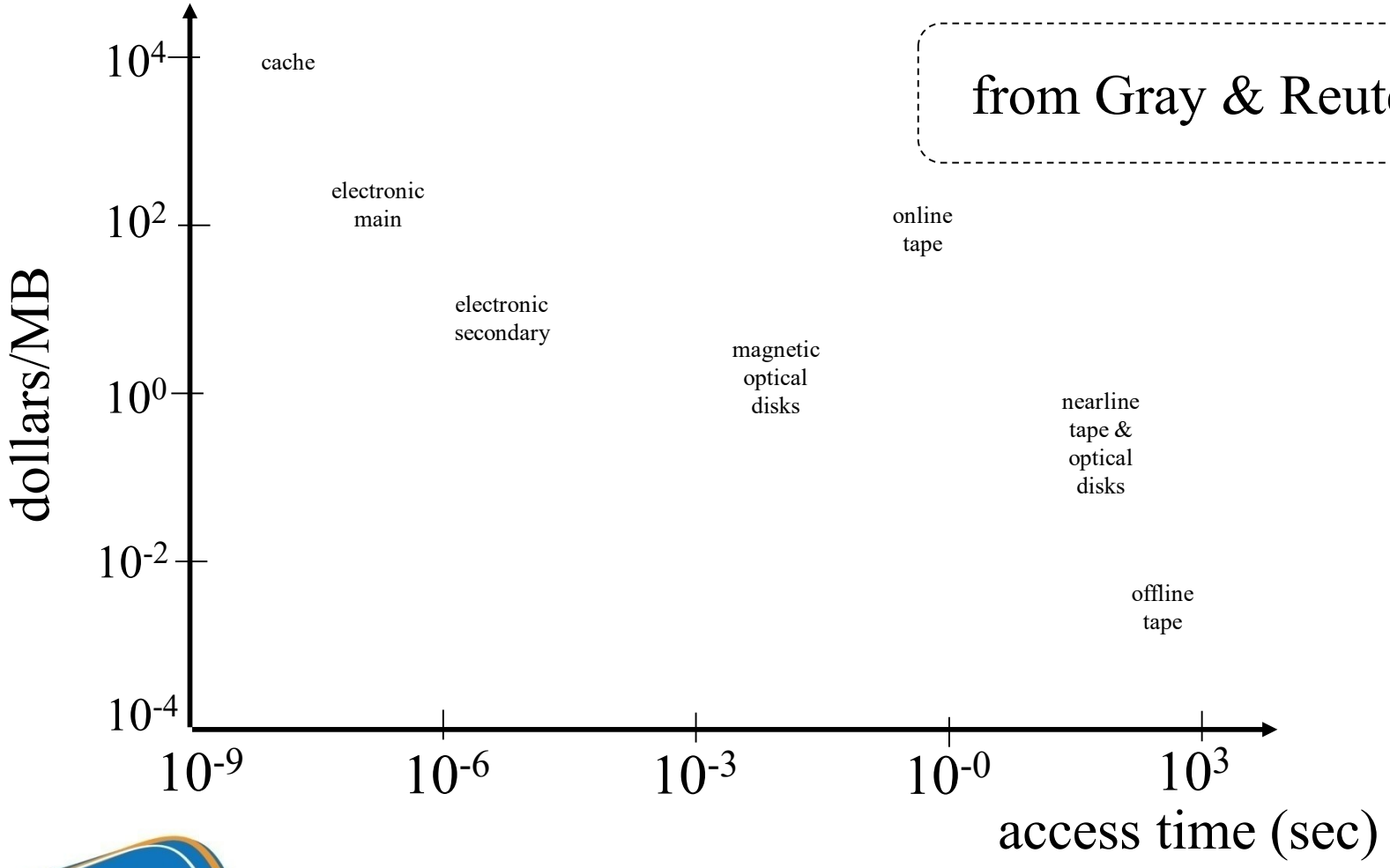
Volatile storage: data is lost when there's power cut



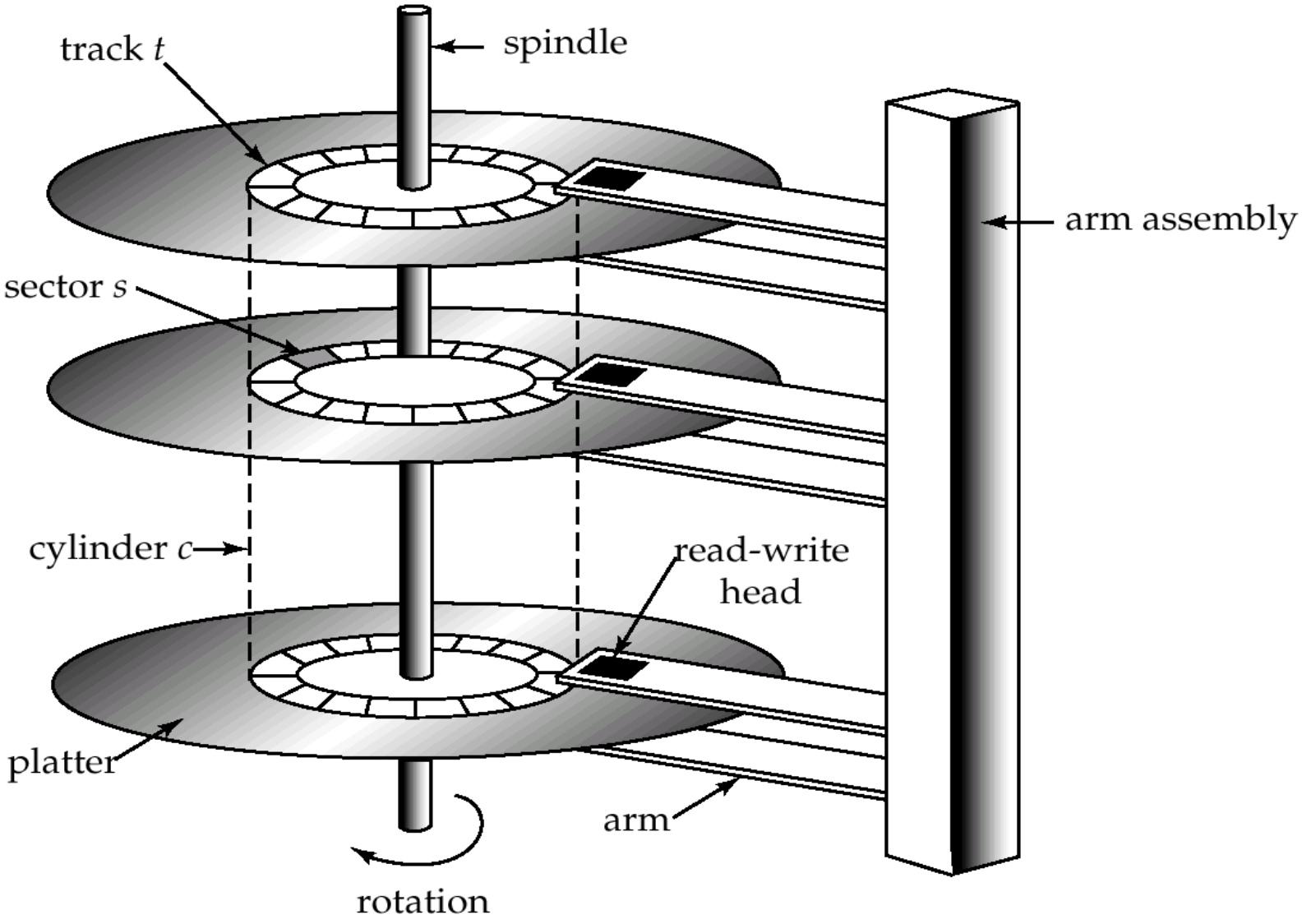
Storage Capacity



Storage Cost



Magnetic disk



Disk management

- Disk surface is divided into many tracks, 1 track is divided into multiple blocks (pages). 1 cluster = n blocks.
- Use magnetic disks to store databases because:
  - Massive data cannot be stored in main memory
  - Persistent, long-term storage for repeated access and processing (main memory can not response)
  - Low cost.
- Data on disk must be input to main memory when it needs processing. If this data is changed, it will be output back to disk.
- Disk controller (DC): communicates between the disk and the computer, receives an I / O command, locates the reader and makes the R / W action.
- Block is also the unit to store and transfer data.

Data transfer

- Average time to find and transfer 1 block = s + rd + btt
  - Seek time (s): Time for DC to locate the reader / writer on the right track, about 7-10 msec (desktop), 3-8 msec (server).
  - Rotational delay / latency (rd):Ttime for the reader to be at the block to be read, depends on rpm, about 2 msec on average.
  - Block transfer time (btt): Time to transfer data, depends on block size, track size, and rpm.
- Accessing consecutive blocks will save time.
- Some search techniques exploit this aspect.

Some principles

- DBMS minimizes the number of blocks transferred between disk and MM (main memory) → reduces the number of disk I/O
  - Save as many blocks as possible in the MM, increasing the chance of finding the block to be accessed in MM.
- Buffer is a part of the MM used to store copies (newer version) of blocks read/write from/to disk, managed by Buffer manager.

Store files in disk

- Database is organized in disk as one / many files, each file has many records, each record has many fields.
- records must be stored on disk in such a way that they can be accessed and accessed effectively.
  - Primary file organization: Decides how physical records are physically located in disk, thereby knows how to retrieve them.
  - Secondary organization/ auxiliary access structure: for efficient access to records on file.

Purpose

- Data storage techniques are useful for database designers, DBAs, and DBMS installers.
  - Database designer, DBA : Knows the advantages and disadvantages of each storage technique to design, implement, and manipulate databases on a specific DBMS.
    - Characteristics of magnetic disks + how to organize data files on magnetic disks → Propose a database design to be able to store and exploit effectively.
    - DBMS often has many options for organizing data, the physical design needs to select appropriate data organization techniques for specific application requirements..
  - Database installer need to know data organization techniques, install DB properly and effectively to provide DBA and users with a full range of options..

Content

1. Some concepts
2. File organization and access method
   - Record, record type, file, fixed size record, changeable size record
   - Locate the file block on disk
   - File header
   - File Manipulation
   - Heap file, Sorted file, Hashing technique
3. Index

Record

- Data ≡ records ≡ data values/ items
  ≡ entity/relationship and their proprties
- Record type/ Record format = set of properties' name and data type.
- Ex: struct EMPLOYEE {char NAME[30];
  char ID[9];
  int SALARY;
  char DEPARTMENT[20];}

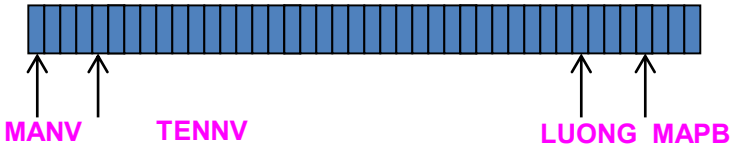
- The data type of each property(field) indicates the value that the field can take.
  - Number: integer(4 bytes), long integer (8 bytes), floating point (4 bytes)
  - String (1 character ≡ 1 byte): fixed or variable length
  - Boolean (1 byte)
  - Date/ time (YYYY-MM-DD: 10 byte)

File

- File organization:
  - Save multiple records, the same type or different type of records.
  - Including records of the same length (bytes) → files of fixed-length records or
  - Composed of different length records → file of variable-length record size
    - Case 1: Same type of records, has field of variable length. For example, a string field
    - Case 2: Different type of records, the case where the related records are grouped (clustered) and stored on the same block for quick access.
    - Case 3: Same type of records, having properties of many different values.
    - Case 4: Same type of records, having properties with / without values.
    - Note: Case 3 and Case 4 do not occur when storing data in a relational database.

File

- Fixed length record: Easy to access



- Variable length record:
  - Case 1:
    - Use special characters (not confused with attribute values) to mark the end of variable length fields or
    - Save the actual size in bytes immediately before the property value.

001	Peter John	1000	LAB

  - Case 2: Before each record, note record type.
  - Case 3: Need a special character separating between repeated values and a ending character.
  - Case 4:
    - If the number of properties is large but the number of properties actually having value is small, you can save the <field-name, field-value> pairs instead of just saving the field-value.
    - Use 3 special characters to separate: between field-name and field-value, between fields, and at the end each record. The same special character can be used for the first 2 purposes.
    - Or encode the data type of each field being a field-type, such as using an integer, and stored <field-type, field-value>

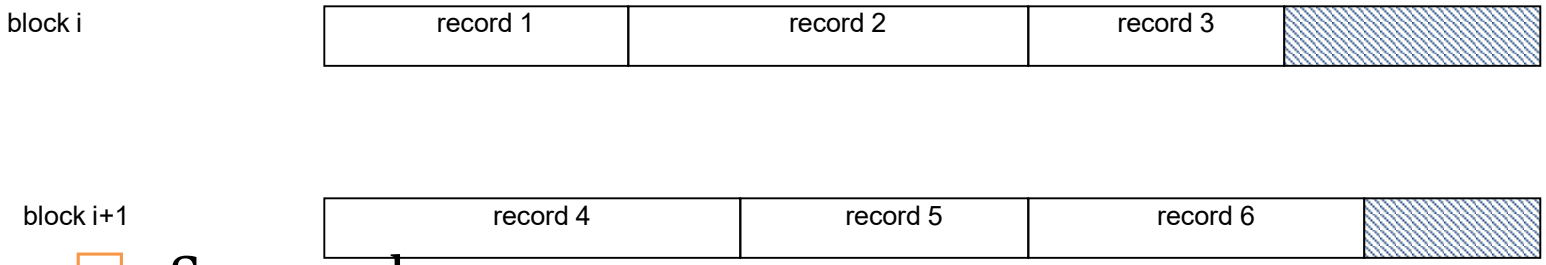
File

- We know: the disk is divided into blocks.
- Often:
  - Block size B> record size R (fixed, in bytes, B> = R)

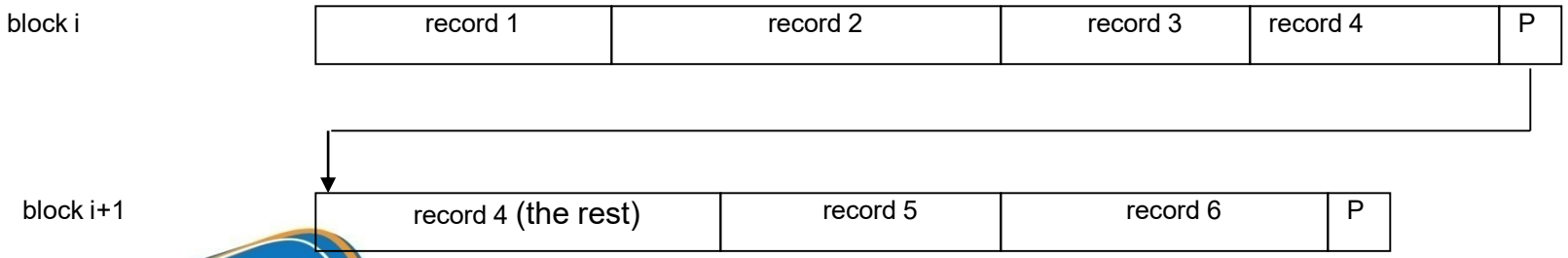
- 1 block contains many records, the number of records:
  - bfr = floor (B / R) records / block
  - bfr is called the blocking factor of the file
  - B - bfr * R is the number of bytes not used in a block.

Allocating records on blocks

- Unspanned



- Spanned



Allocating blocks on disk

- Contiguous: the file blocks are allocating to consecutive disk blocks.
  - Read the whole file very fast using double buffering.
    - While CPU is processing this block on main memory, the disk I/O processor can be reading and transferring the next block into a different buffer.
  - Difficult to expand the file.
- Link allocation: each file block contains a pointer to the next file block.
  - Easy to expand the file.
  - Slow to read the whole file.
- Combination of these two techniques.
  - Clusters of consecutive disk blocks and the clusters are linked.
  - Cluster are called file segment or extent.

File Header

- File header of file descriptor contains information about a file that is needed by the system programs that access the records.
- Includes:
  - Disk addresses of the file blocks.
  - Record format descriptions:
    - fixed-length unspanned records: field lengths, order of fields within a records.
    - variable-length records: field type codes, seperator characters, record type codes.
  - For searching a record on disk:
    - One or more disk blocks are copied into main memory buffers.
    - Programs then search for the desired record(s) within the buffers, using the information in the file header.
    - Time consuming → good file organization for locating quicky

Operations on file

- Major operations:
  - Find a specific record.
  - Insert a record.
  - Delete a record.
  - Update a record.
- Depending on the file organization, we have an appropriate operations.

Fixed-length records

- Consider a file with the records:

```
type deposit = record
        account-number: char(10);
        branch-name: char(22);
        balance: real;
end
```

- Suppose that:
  - 1 *char* : 1 byte
  - *Real* : 8 bytes
  - 1 record **deposit**: 40 bytes
  - The first 40 bytes store the 1st record
  - The next 40 bytes store 2rd record…

A-102	Perryridge	400
A-305	Round Hill	350
A-215	Mianus	700
A-101	Downtown	500
A-222	Redwood	700
A-201	Perryridge	900
A-217	Brighton	750
A-110	Downtown	600
A-218	Perryridge	700

Fixed-length records (cont)

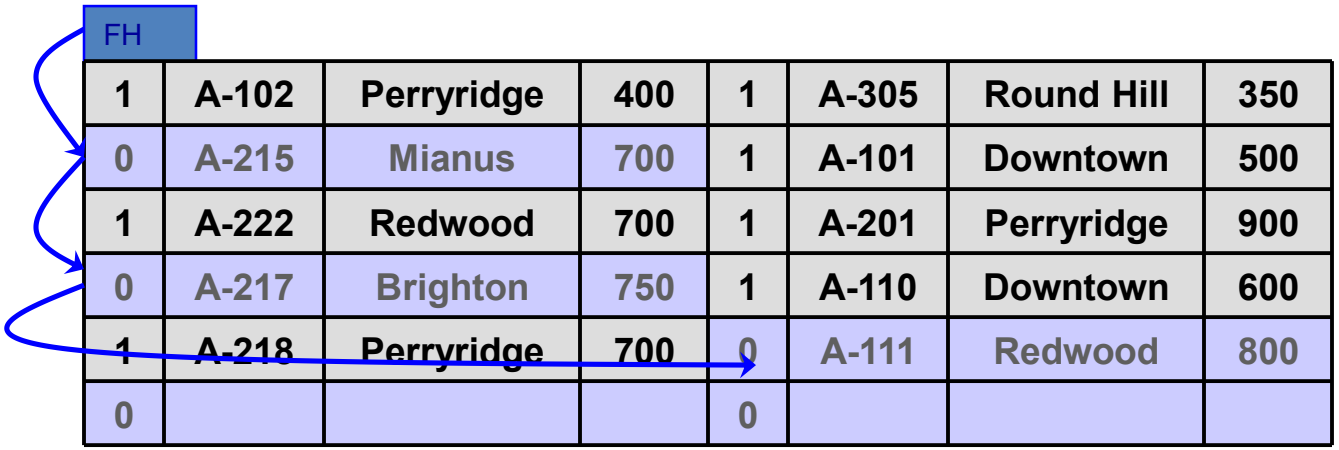
- Each record has an additional bit
  - =0: Deleted
  - =1: In use
- File header stores the address of the first free (unused) record.
  - List of unused records (free list)
- The physical storing of *account* records

0 1 10 11 32 33 40 41

1	A-102	Perryridge	400	1	A-305	Round Hill	350
1	A-215	Mianus	700	1	A-101	Downtown	500
1	A-222	Redwood	700	1	A-201	Perryridge	900
1	A-217	Brighton	750	1	A-110	Downtown	600
1	A-218	Perryridge	700	0			
0				0			

Fixed-length records (cont)

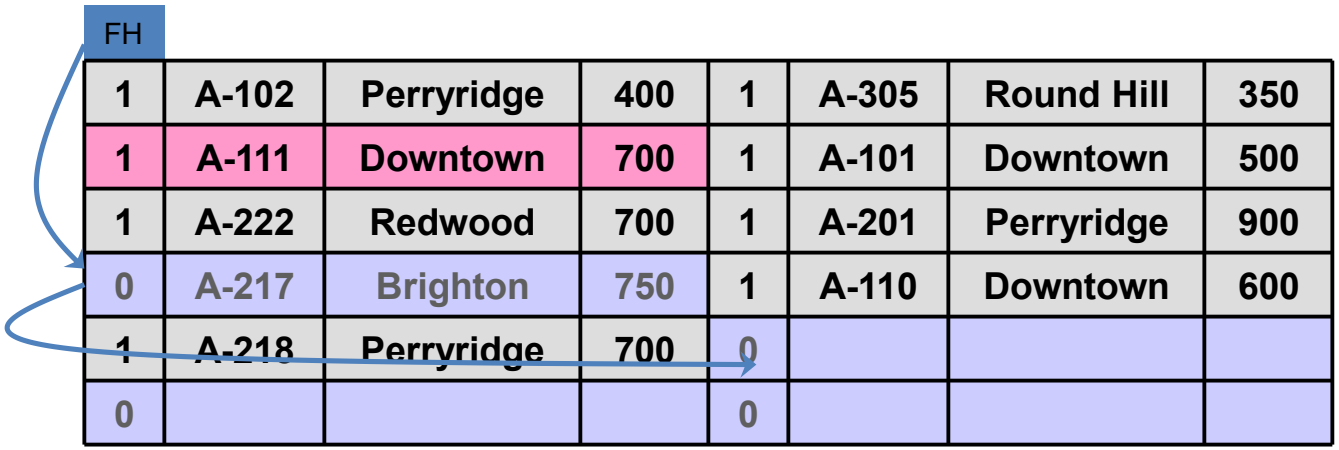
- Deleting a record
  - Mark “del” to the info bit
  - Add the deleted records to free list.



1	A-102	Perryridge	400	1	A-305	Round Hill	350
0	A-215	Mianus	700	1	A-101	Downtown	500
1	A-222	Redwood	700	1	A-201	Perryridge	900
0	A-217	Brighton	750	1	A-110	Downtown	600
1	A-218	Perryridge	700	0	A-111	Redwood	800
0				0			

Fixed-length records (cont)

- Add a record
  - Insert at the position of free (deleted) records or insert at the end of file.
  - Update the free list



1	A-102	Perryridge	400	1	A-305	Round Hill	350
1	A-111	Downtown	700	1	A-101	Downtown	500
1	A-222	Redwood	700	1	A-201	Perryridge	900
0	A-217	Brighton	750	1	A-110	Downtown	600
1	A-218	Perryridge	700	0			
0				0			

- Search
  - Sequential scanning on file

Variable-length records

- A file may have variable-length records for several cases:
  - The file records are of the same record type, but one or more of the fields are of varying size.
  - The file records are of the same record type, but one or more of the fields may have multiple values of individual records.
  - The file records are of the same record type, but one or more of the fields are optional.
  - The file contains records of different record types and hence varying in size.
- Consider this example:

```
type account-list = record
        branch-name: char(22);
        account-info: array [1..n] of
                record
                        account-number: char(10);
                        balance: real;
                end
end
```

Variable – length records

- Byte-String Representation
  - Should use a special seperator character to terminate each record.
  - Use a special character for terminating variable-length fields, or store the length in bytes together with the field value.
  - Optional fields: <field name, field value> or <field type, field value>

Perryridge	A-102	400	A-201	900	A-218	700	-
Round Hill	A-305	350	-	Brighton	A-217	750	-
Downtown	A-101	500	A-110	600	-		
Mianus	A-215	700	-				
Redwood	A-222	700	-				

Variable – length records (cont)

- Byte-String Representation
  - Reuse free space after deleting a record is ineffective
  - Lead to fragmentation

Perryridge	A-102	400	A-201	900	A-218	700	-
Round Hill	A-305	350	-	Brighton	A-217	750	-
Downtown	A-101	500	A-110	600	-		
Mianus	A-215	700	-				
Redwood	A-222	700	-				

Variable – length records (cont)

- Byte-String Representation
  - It is costly when the record length changes

Perryridge	A-102	400	A-201	900	A-218	700	-
Round Hill	A-305	350	-	Brighton	A-217	750	-
Downtown	A-101	500	A-110	600	-		
Mianus	A-215	700	-				
Redwood	A-222	700	-				

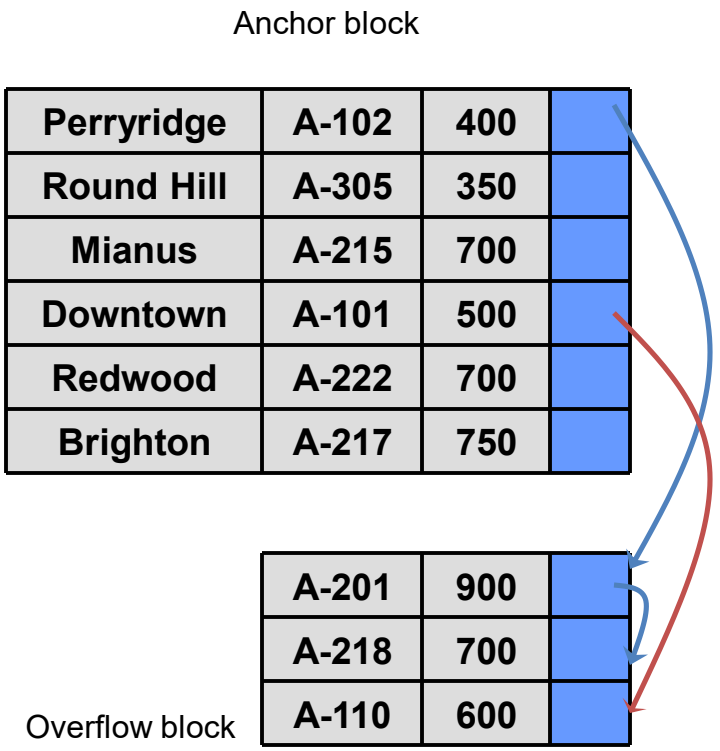
Variable – length records (cont)

- Fixed-Length Representation
  - Use 1 or more fixed-length records to represent variable length records
  - There are 2 techniques
    - Reserved space
      - Use the maximum length of a record to implement the remaining records
      - This length must be guaranteed to never be longer

Perryridge	A-102	400	A-201	900	A-218	700
Round Hill	A-305	350				
Mianus	A-215	700				
Downtown	A-101	500	A-110	600		
Redwood	A-222	700				
Brighton	A-217	750				

Variable – length records (cont)

- Fixed-Length Representation
  - Pointer
    - Variable length records link together through a list of fixed-length records
    - Thera are 2 block types in file:
      - Anchor block – store the first record of table *account-info*
      - Overflow block – Store the rest records of table *account-info*



Record organization in file

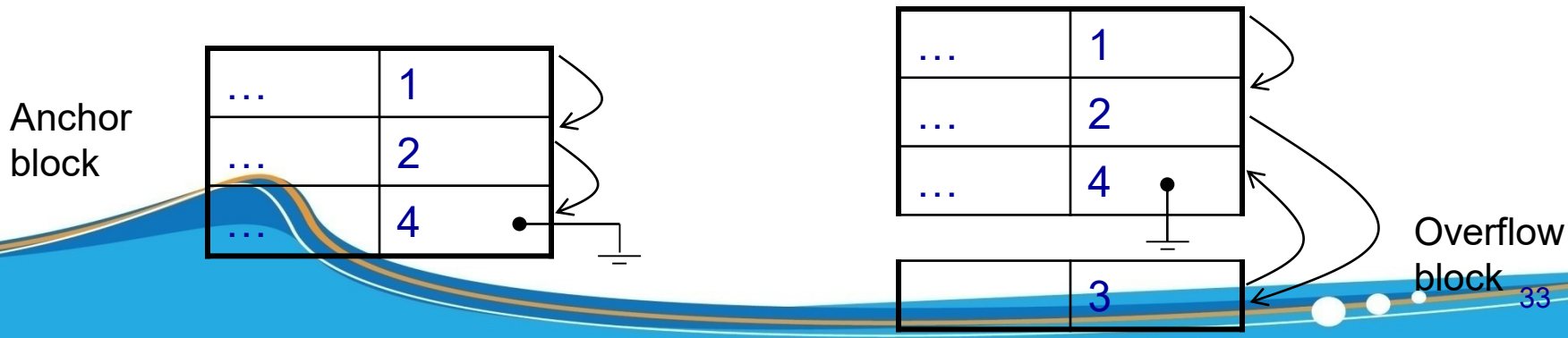
- Unordered : records are inserted anywhere in the file.
- Ordered: records saved in right position to ensure order by a certain field.
- Hashing: locating records on a storage device using a hash function.

Heap file

- A form of data storage in which the records are not stored in any logical order (but in the order of inserting).
- Often, data of each relation is stored in a file.
  - Search: scan.
  - Insert: fast.
- Easy to store and manipulate data, suitable only for small files, system will be very slow when the file is large.

Sequential file

- A form of storing data in which the records are stored in the order of the field used as search key.
- Records in order are linked by pointers.
- Suitable for specific applications that work on sorted data (by search key).
  - Search: scan or search sequentially.
- Should store physically in the order of the search key to minimize the number of blocks read. But :
  - With large data, Insert and Delete operations are complicated
    - Insert: Positioning -> insert into overflow block (≠ anchor block) -> break physical order, must reorganize



Hashing file

- A hash function is set on an property which is the search key of the relation.
- Principle: "where to save, find there".
- Divide the file into buckets depending on the value of the search key. Each lot has a number of blocks, linked by pointers. The data in the block is organized as a heap.
- B is the number of lots. The hash function’s return value is an integer $\in$ [0,B-1] indicating the lot containing the record. If the key is a string, we define a rule for converting the string into a number.

Hashing file

- Search record with key v
  - Calculate h(v) for the lot, and perform a search in this lot.
- Insert
  - Calculate h(v) for the lot. Search for the last block of the lot, if there is room then insert, otherwise allocate another block at the end of the list of lot h(v).
- Delete/ Update
  - Search and update or delete
  - After deletion, it may be necessary to perform a correction step (consolidating data in the block) to reduce the number of blocks in this lot.

Clustering file

10	Dept	Loc
	KT	HCM

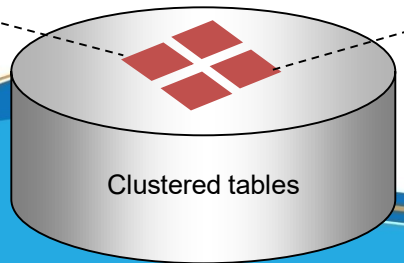
MANV	TENNV	…
N2	B	…
N5	E	…
N6	G	…

20	Dept	Loc
	KT	HCM

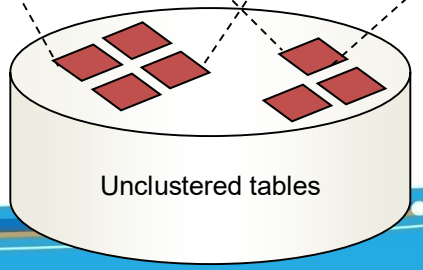
EmpID	Name	…
N1	A	…
N3	C	…
N4	D	…

EmpID	Name	DeptID	…
N1	A	20	…
N2	B	10	…
N3	C	20	…
N4	D	20	…
N5	E	10	…
N6	G	10	…

DepID	DeptName	Loc
10	KT	HCM
20	KD	HN



Related data is stored together → save space.



Related data is separated, need more space

Clustering file

- A cluster is formed by storing data of several tables on the same blocks.
- A cluster key is one or more common fields of the tables involved in Clustering. Cluster key is specified when the user creates the cluster.
- These tables are often co-accessed or joined.
- Advantages:
  - Reduce disk access time because the number of blocks to read is reduced
  - The value at the cluster key field is only saved once, no matter how many records in another table refer to this row → save space
- Organizing data as a cluster does not affect index creation on tables that participate in the cluster.

Using cluster file

- Identify cluster at the physical design stage.
- Select the tables to cluster :
  - The tables mainly for select, rarely for insert or update.
  - Contains data that is queried together with high frequency.
- Choose columns to be cluster key.
  - The cluster key must have enough distinct values so that the records associated with each cluster key value fill up a data block.
    - If there are too few rows, it will take up storage space but the effect is negligible.
    - SIZE can be specified when creating the cluster, which is the estimated average number of bytes that can be stored in a cluster
    - If too many rows are also ineffective.

Index (1)

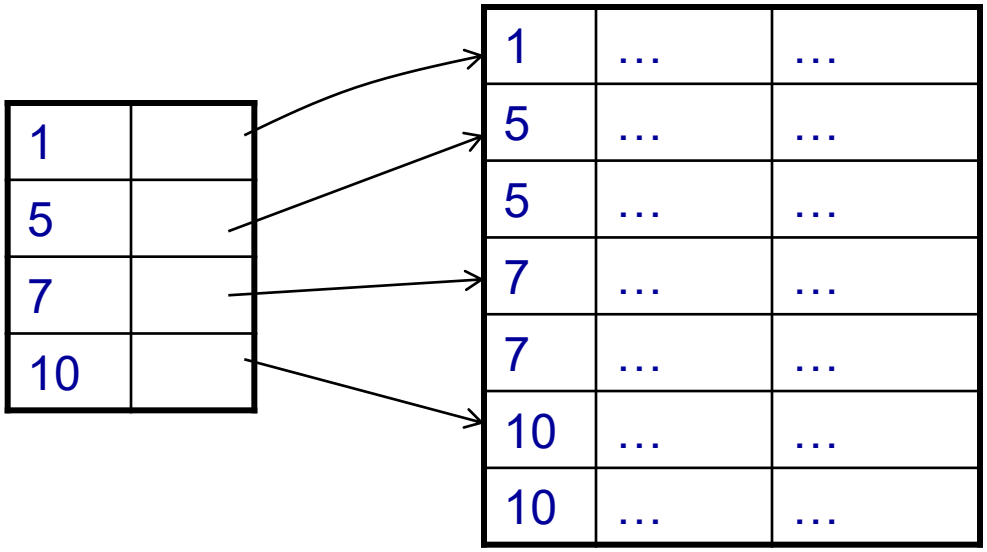
- Using the index for a file is the same as using the catalog in a library.
  - Information in the catalog has been sorted → quick search without having to scan all.
- Technically, there are 2 basic types of indexes :
  - Ordered indices: Base on index valued.
    - Use binary search in index file.
    - The index is an ordered file of fixed-length records of 2 fields.
      - Field 1: search key.
      - Field 2: pointer to the blocks.
  - Hash indices.
- Evaluate techniques for using index.
  - Access type.
  - Access time.
  - Insert / delete time.
  - Disk space for index.

Index (2)

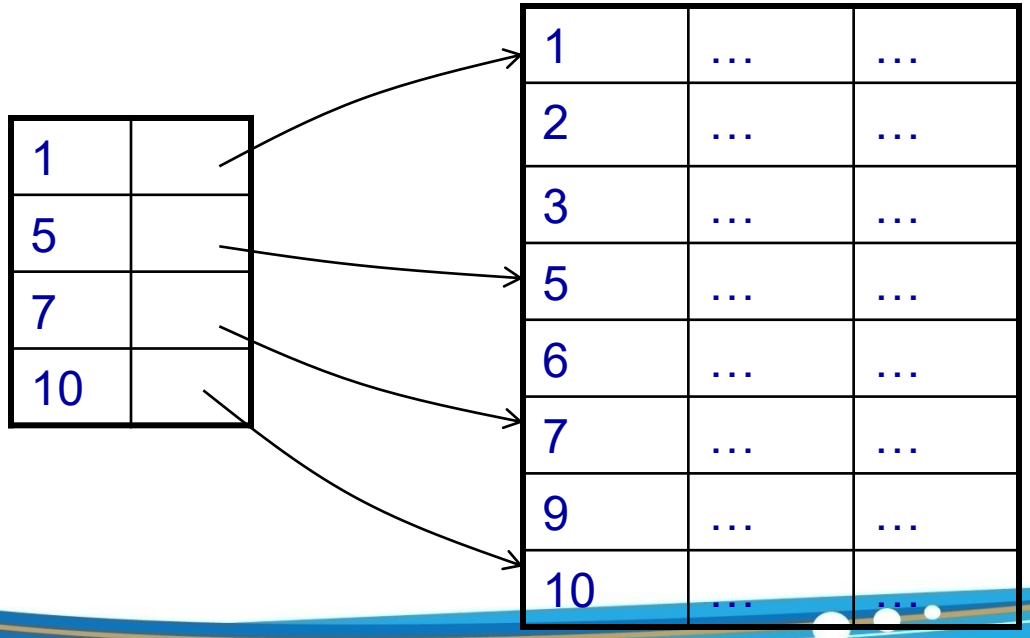
- Dense / Sparse index
  - Dense index
    - Index file has as many records as the distinct values of the search key?
    - Each record of the index file contains a search key value and a pointer to the first record on the data file having the same value of the search key.
  - Sparse index
    - Records on the index file are only correspond to some values of the search key (not all search key values like dense index)
    - To find a value, search the index file for the record having the maximum value of the search key <= value to be searched, and scan data file from the first record that the pointer points to.

Example

- Dense index



- Sparse index



Index (3)

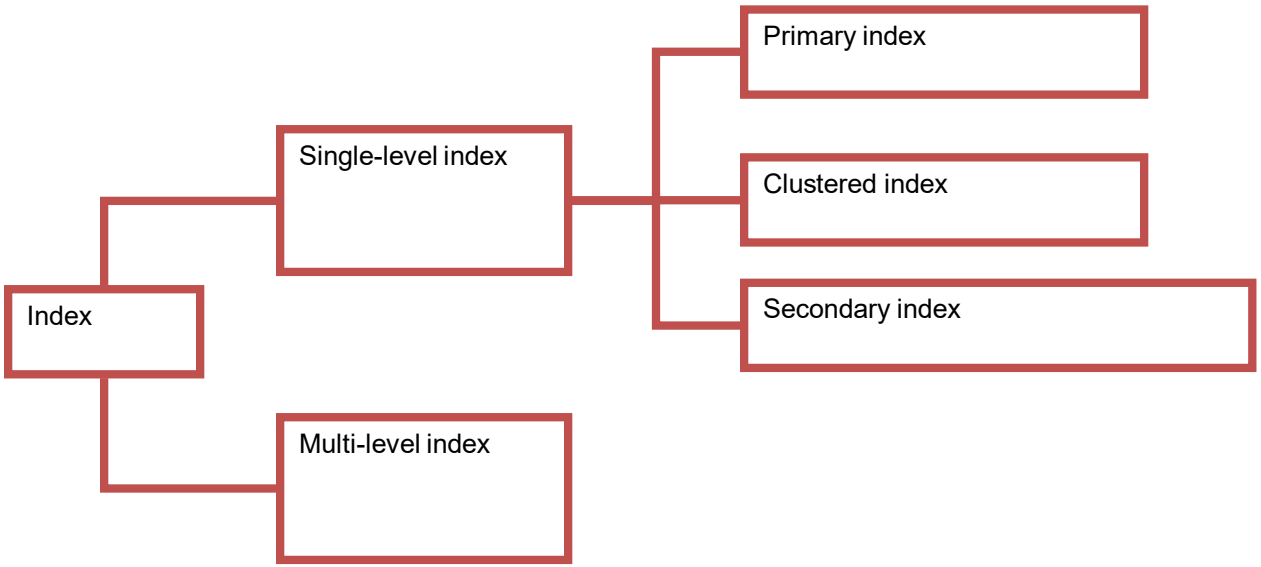
- Files have their own organization and corresponding operation.
- In addition, we have to add an access structure that supports quick access on file.
- Index is a mechanism that helps DBMS access data quickly.
- The index key refers to one or more fields used as indexes.
  - Simple Index: index of only 1 field.
  - Composite index: index with many fields, but <=16

Index(4)

- Each index is associated with a specific index key.
- To access the database using an index, we must use one or more fields that are index keys in the WHERE clause of the SQL statement.
- Basically, any field can be indexed and we can have multiple indexes on the same file. The problem is whether the index is effective or not .
- Index is effective or not based on :
  - The data type of index key.
  - Whether the value on the index key is distinguished or not.
  - The type of SQL statement used.
    - When executing an SQL statement, if more than 20% of the data rows in a table are accessed, using an index is not as beneficial as not using an index (table scan).
  - Multiple updates on the field(s) being index key will slow down the system.
  - Having too many indexes will slow down the system.

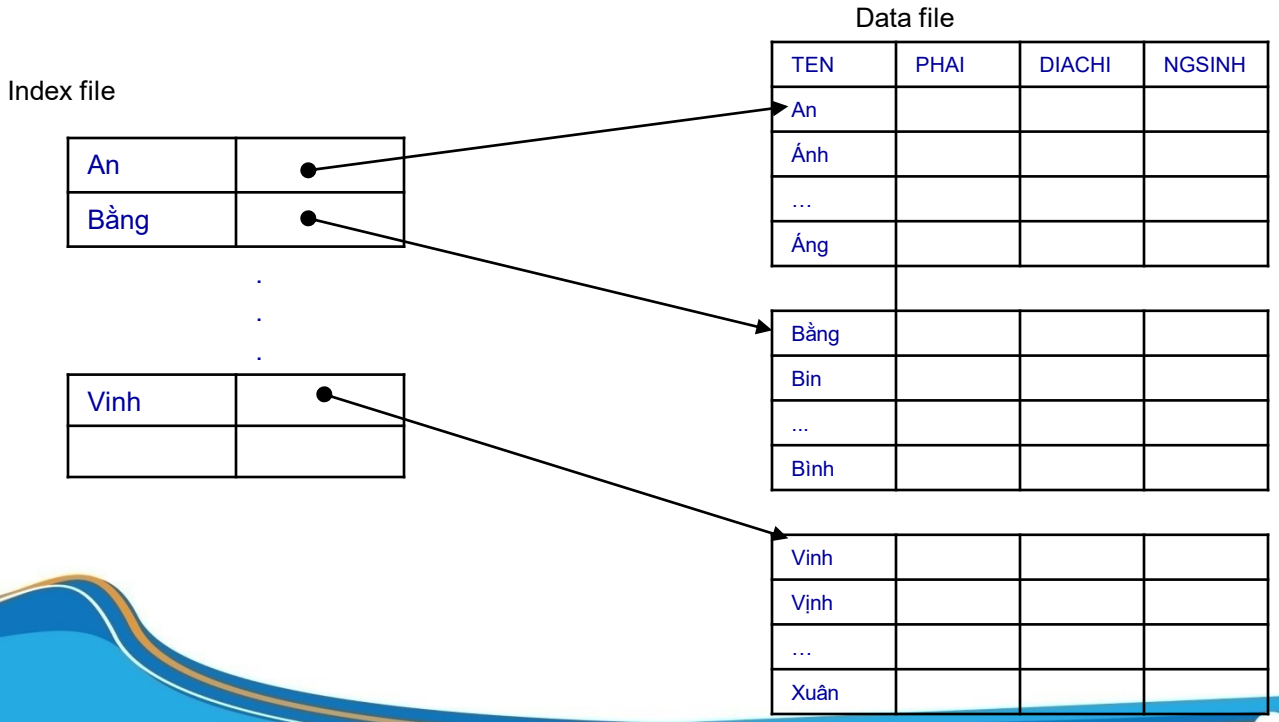
Index (5)

Terms



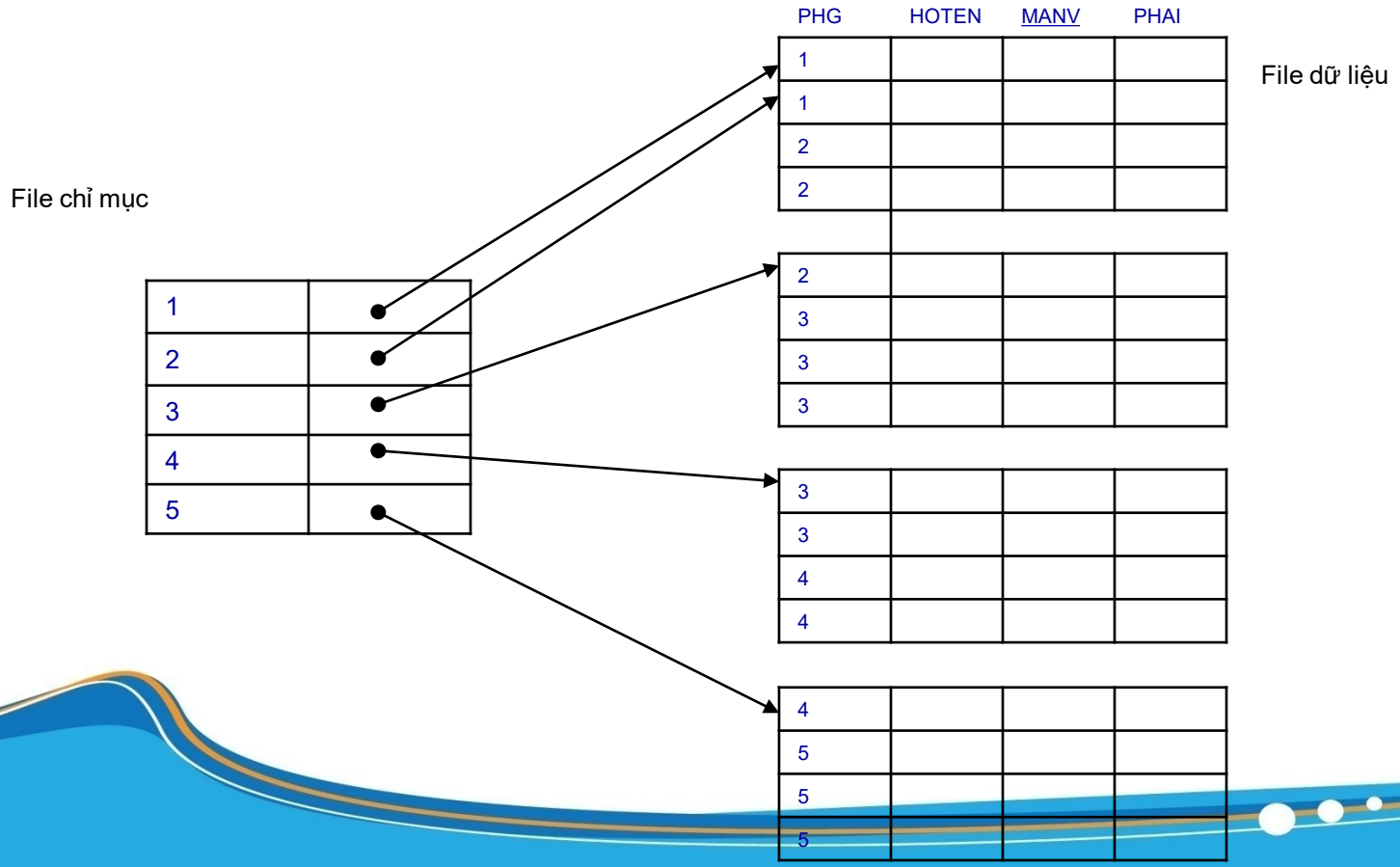
Primary index

- Created on the field as the sort key for the data file. The physical order of the records on disk is also based on this field, and on which the records have a unique value.
  - If there are multiple records with the same value in the field used to sort, we will create a clustering index on this field.
- One index record in the index file corresponding to a block in the data file.
- The primary index file is much smaller than the data file.
  - The first record in each block of the data file is called the anchor record or block anchor.
- There can only be 1 primary index, or 1 clustered index in 1 data file, there cannot be both types of index in the same file..



Clustering index

- If the data on the data file is physically arranged by a field that is not a key (not unique to each message), that field is a clustering field.
- Clustering indexes are created on clustering fields to speed up the search for records with the same value of clustering fields.
- There is a record in the index file containing a clustering field value, the pointer to the first block contains the discriminant value.



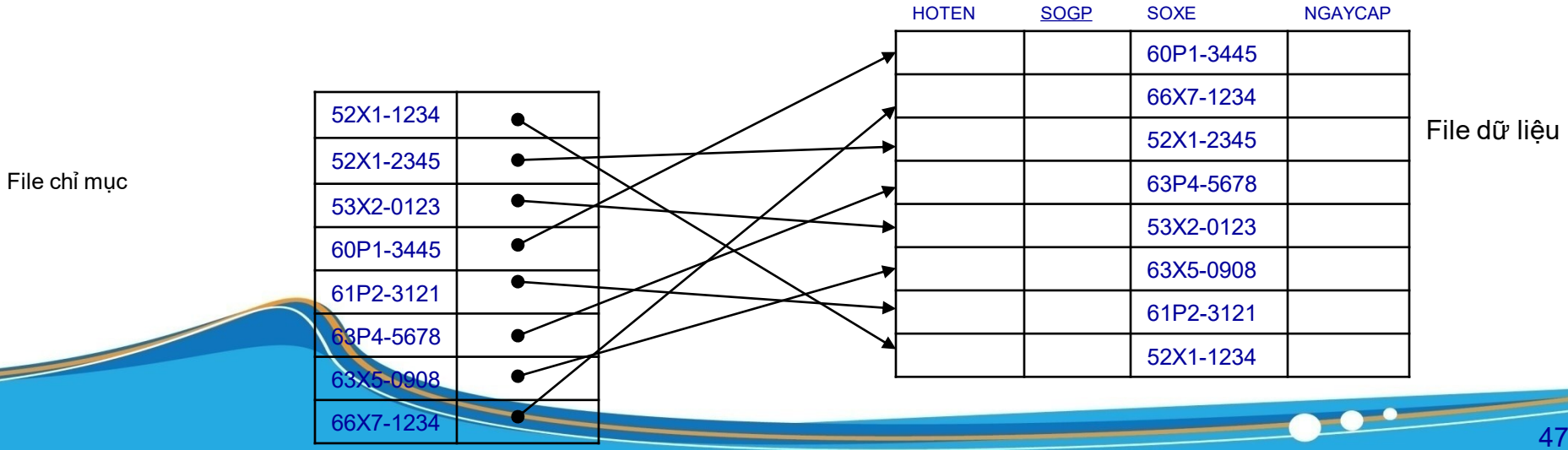
Secondary Index

A secondary index provides additional means to access files, in addition to the primary index.

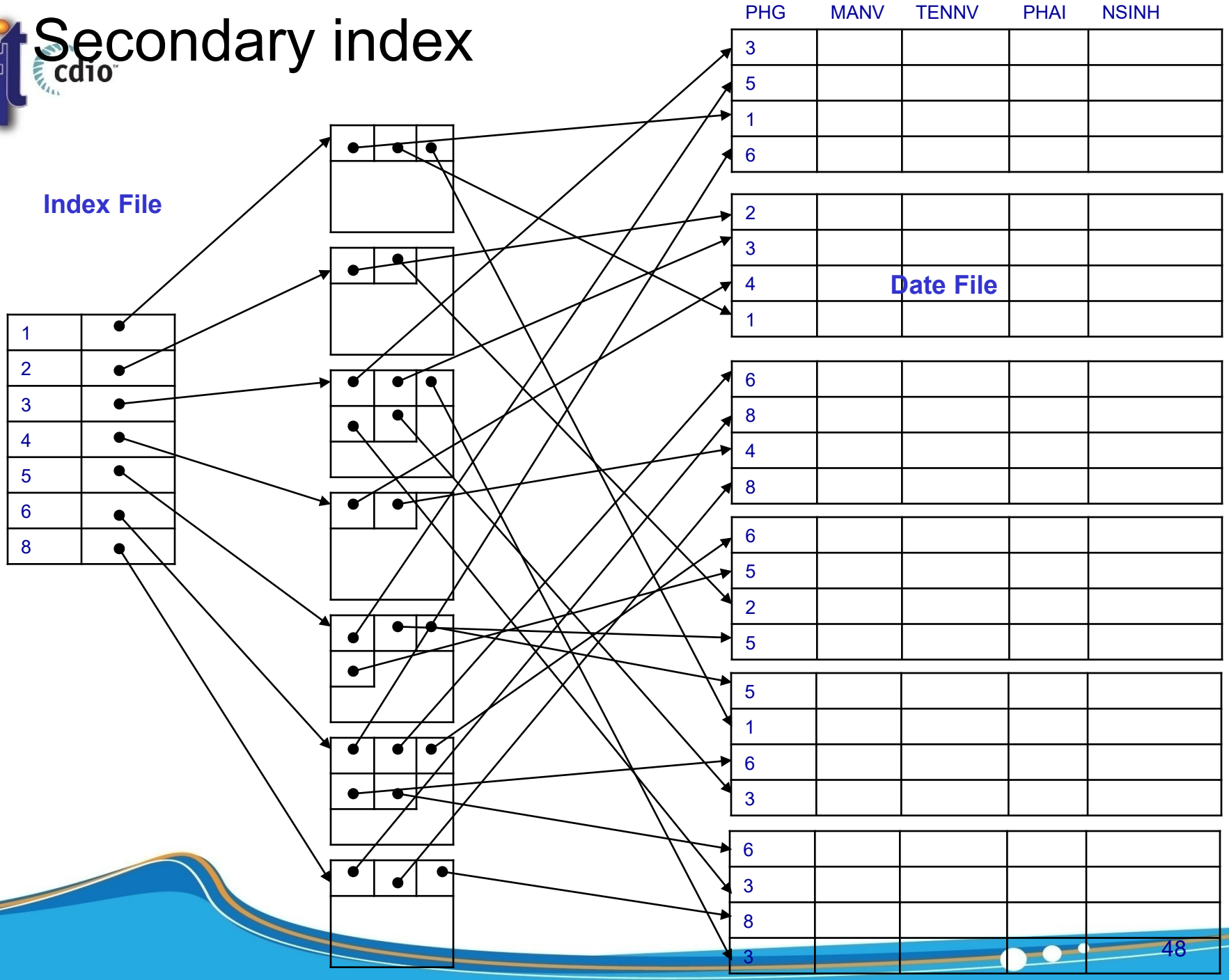
- Created on the field being the candidate key and having a unique value on each record, the data file is not sorted by this field.
- It is also possible to create a field that is not a key and has identical values.
- Field 1 is the unordered data field of the data file, being the searched value.
- Field 2 is the pointer to the first block that contains the value, or to the record containing the value.

It is possible to create multiple secondary indexes for 1 data file.

- Case 1: Created on a field with a unique value, this field is also known as a secondary key.



Secondary index

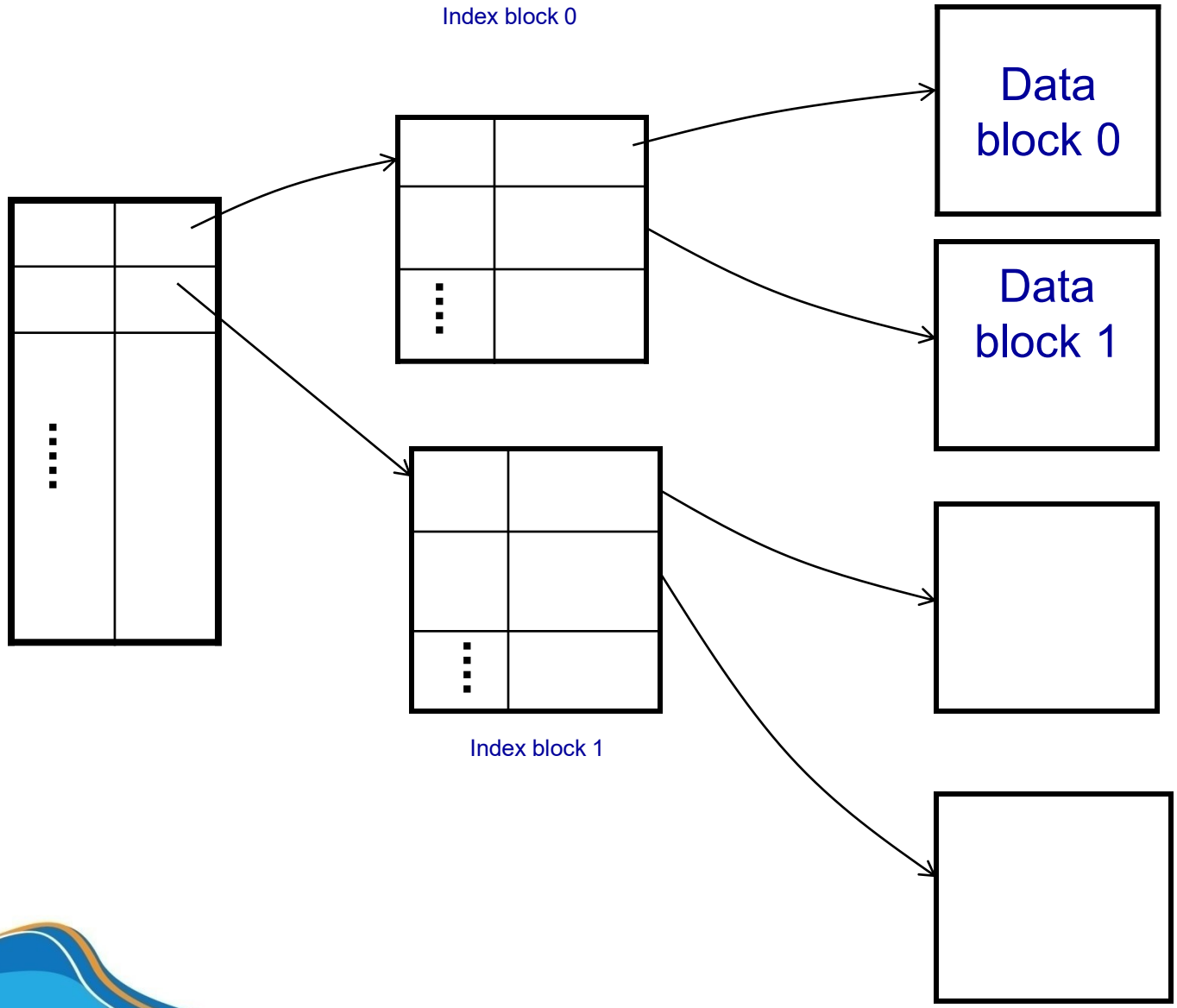


Comments

	Data file is sorted by Index field	Data file is not sorted by Index field
Index field is key	Primary index	Secondary index (Key)
Index field is not key	Clustering index	Secondary index (non key)

- A file can have many indexes because there may be many search criteria on the file.
- For SQL Server, can have a maximum of 1 primary index & 249 secondary index.

Multi – level Index



Guide

- Creating index on the distinct value field, that will be accessed with a high frequency, that produce a relation with multiple tuples.
- Creating index on the attributes involving the range selection condition (refer to the operators BETWEEN, >, >=, <, <=)
- Index keys should be the fields used in the join conditions, or used in the GROUP BY, ORDER BY clauses.
- Should not create clustered index on the fields with a high update frequency.

End.