STORAGE

MODULE IV

RAJENDRAN S

01.

RAID

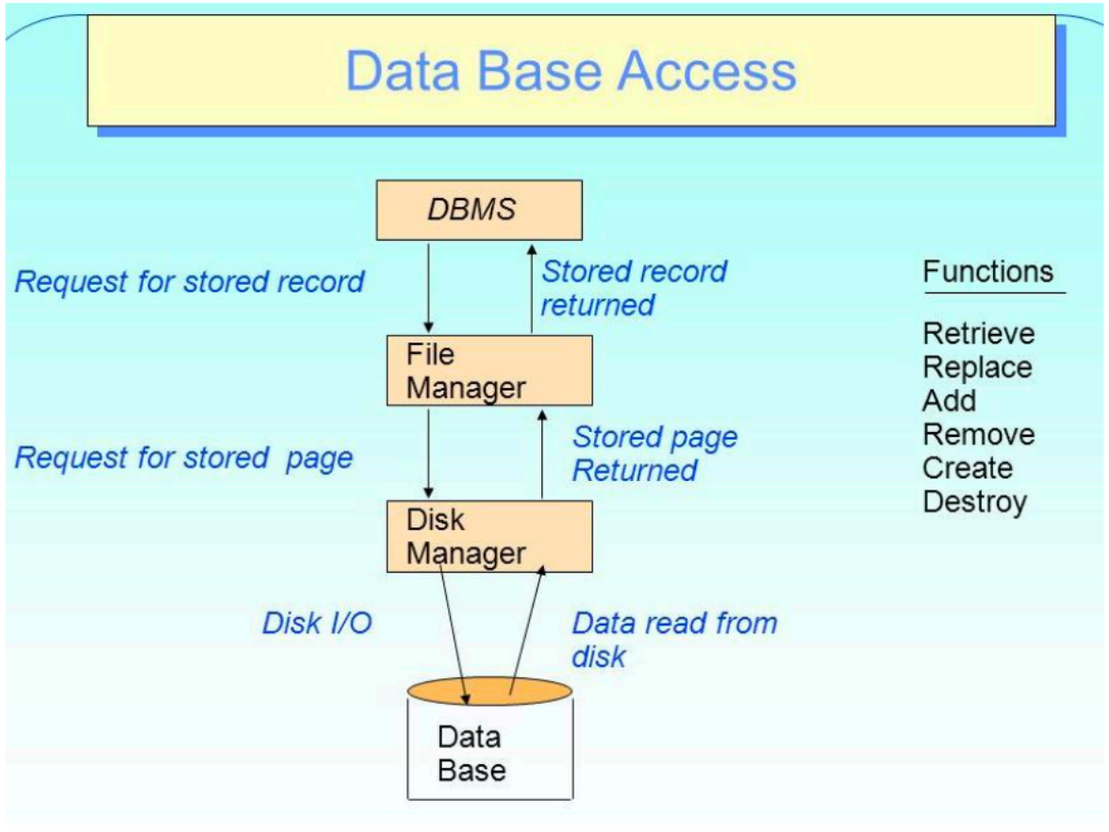
Before we dive into RAID, let's look into some storage types!

Lezzzzzz gooooooooo!!!!!

How data is being accessed?

- Data in a DBMS is stored on storage devices such as disks and tapes

- The disk space manager is responsible for keeping track of available disk space.
- The file manager, which provides the abstraction of a file of records to higher levels of DBMS code, issues requests to the disk space manager to obtain and relinquish space on disk.



Storage Manager

A Storage Manager is a component or program module that provides the interface between the low-level data stored in the database and the application programs/queries submitted to the system. The Storage Manager Components include –

1. **File Manager-** File manager manages the file space and it takes care of the structure of the file. It manages the allocation space on disk storage and the data structures used to represent info stored on other media.
2. **Buffer Manager –** It transfers blocks between disk (or other devices) and Main Memory. A DMA (Direct Memory Access) is a form of Input/Output that controls the exchange of blocks process. When a processor receives a request for a transfer of a block, it sends it to the DMA Controller which transfers the block uninterrupted.
3. **Authorization and Integrity Manager –** This Component of storage manager checks for the authority of the users to access and modify information, as well as integrity constraints (keys, etc).
4. **Disk Manager-** The block requested by the file manager is transferred by the Disk Manager.

Memory Hierarchy

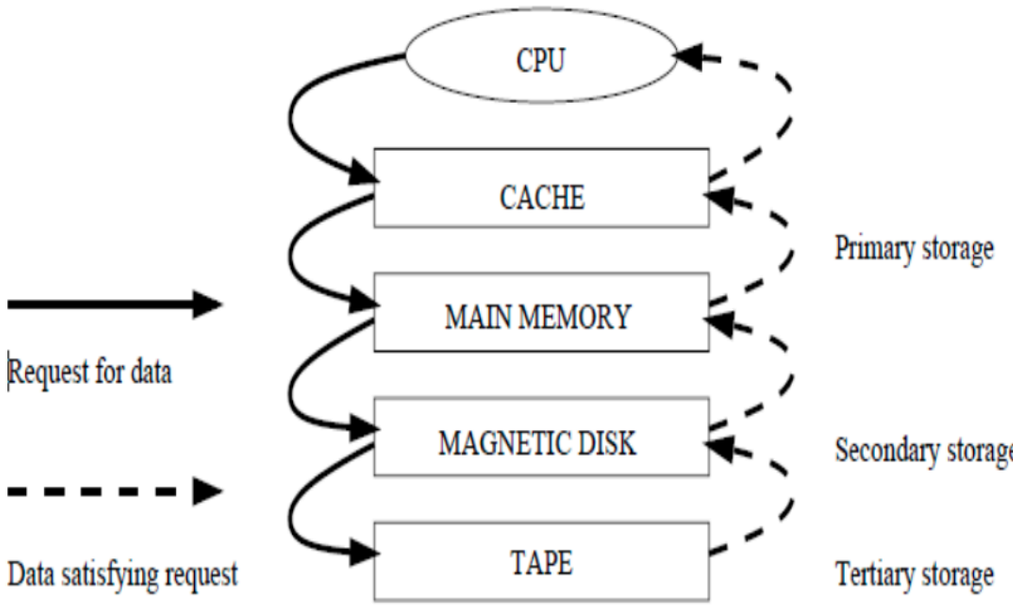
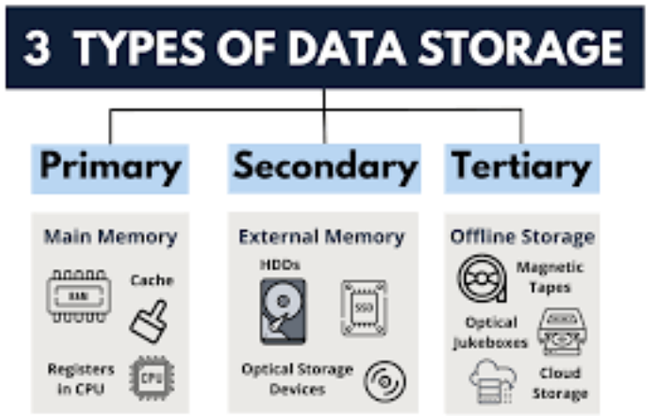


Figure: Memory Hierarchy

At the top, we have primary storage, which consists of cache and main memory , and provides very fast access to data. then comes secondary storage, which consists of slower devices such as magnetic disks. tertiary storage is the slowest class of storage devices; for example, optical disks and tapes.

3 Storages



Primary Storage:

1. all the primary storage level, the memory hierarchy includes at the most expensive end cache memory, which is a static RAM (Random Access Memory) cache memory is mainly used by the CPU to speedup execution programs.
2. the next level of primary storage is DRAM (Dynamic Random Access Memory), which provides the main work area for the CPU for keeping programs and data , which is popularly called as main memory .
3. the advantages of DRAM is its low cost, which continuous to decrease ; the drawback is its volatility and lower speed compared with static RAM.

Secondary Storage:

At the secondary storage level, the hierarchy includes magnetic disks, as well storage in the form of CD - ROM (Compact Disk - Read Only Memory) devices.

Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.

Tertiary storage:

At the tertiary storage level, the hierarchy includes optical disks and tapes as the least expensive end.

The storage capacity anywhere in the hierarchy is measured in kilobytes (k bytes or bytes), megabytes (M bytes or 1 million bytes), gigabytes (G byte or billion bytes), and even terabytes (1000 G bytes).

What's better than SRAM?

DRAM Bruh!

programs reside execute in **DRAM** . Generally, large permanent database reside on secondary storage, and portions of the database are read into and written from buffers is main memory as needed. personal computers and work stations have tens of megabytes of data in DRAM. it is become possible to load a large fraction of the database into main memory. an example is telephone switching applications, which store databases that contain routing and line information in main memory.

Flash Memory:

1. Between **DRAM and magnetic disk storage**, another form of memory resides, **flash memory**, which is becoming common, particularly because it is non - volatile.
2. flash memories are high density, high - performance memories using EEPROM (Electrically Erasable programmable Read -only Memory) technology.
3. the advantage of flash memory is the fast access speed;
4. the disadvantage is that an entire block must be erased and written over at a time.

Magnetic disk storage:

- primary medium for long-term storage.
- Typically the entire database is stored on disk.
- Data must be moved from disk to main memory in order for the data to be operated on.
- After operations are performed, data must be copied back to disk if any changes were made.
- Disk storage is called **direct access** storage as it is possible to read data on the disk in any order (unlike sequential access).
- Disk storage usually survives power failures and system crashes.

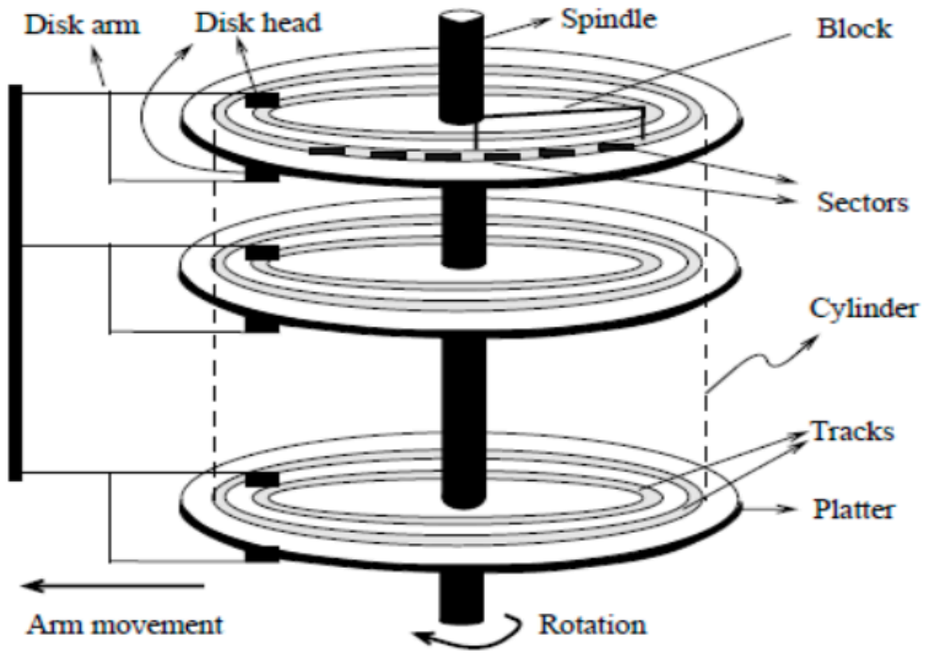


Figure 7.2 Structure of a Disk

Figure: Structure of magnetic disk

Access time: the time it takes from when a read or write request is issued to when data transfer begins.
Data-transfer rate– the rate at which data can be retrieved from or stored to the disk.
Mean time to failure (MTTF)– the average time the disk is expected to run continuously without any failure.

CD-ROM:

CD - ROM disks store data optically and are read by a laser. CD - ROM s contain pre - recorded data that cannot be overwritten. WORM (Write - Once - Read - Many disks) are a form of optical storage used for archiving data; they allow data to be written once and read any number of times without the possibility of erasing. the DVD (Digital Video Disks) is a recent standard for optical disks allowing fourteen to fifteen gigabytes of storage per disks.

Tapes:

1. Tapes are relatively not expensive and can store very large amount of data. when we maintain data for a long period but do expect to access it very often.

2. used primarily for backup and archival data.
3. Cheaper, but much slower access, since tape must be read sequentially from the beginning.
4. Used as protection from disk failures!
5. A Quantum DLT 4000 drive is a typical tape device; it stores 20 GB of data and can store about twice as much by compressing the data.

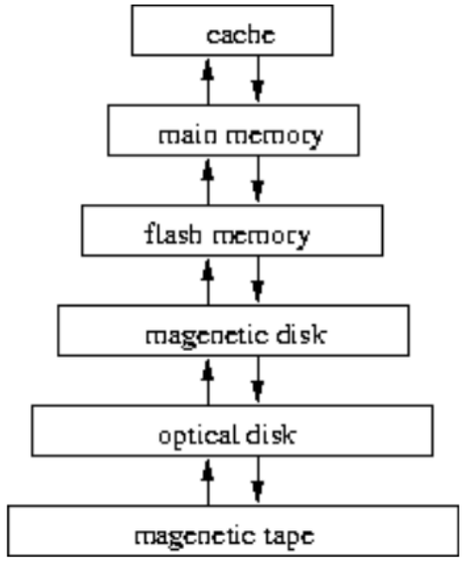


Figure: storage device hierarchy

It’s time to RAID now!!!

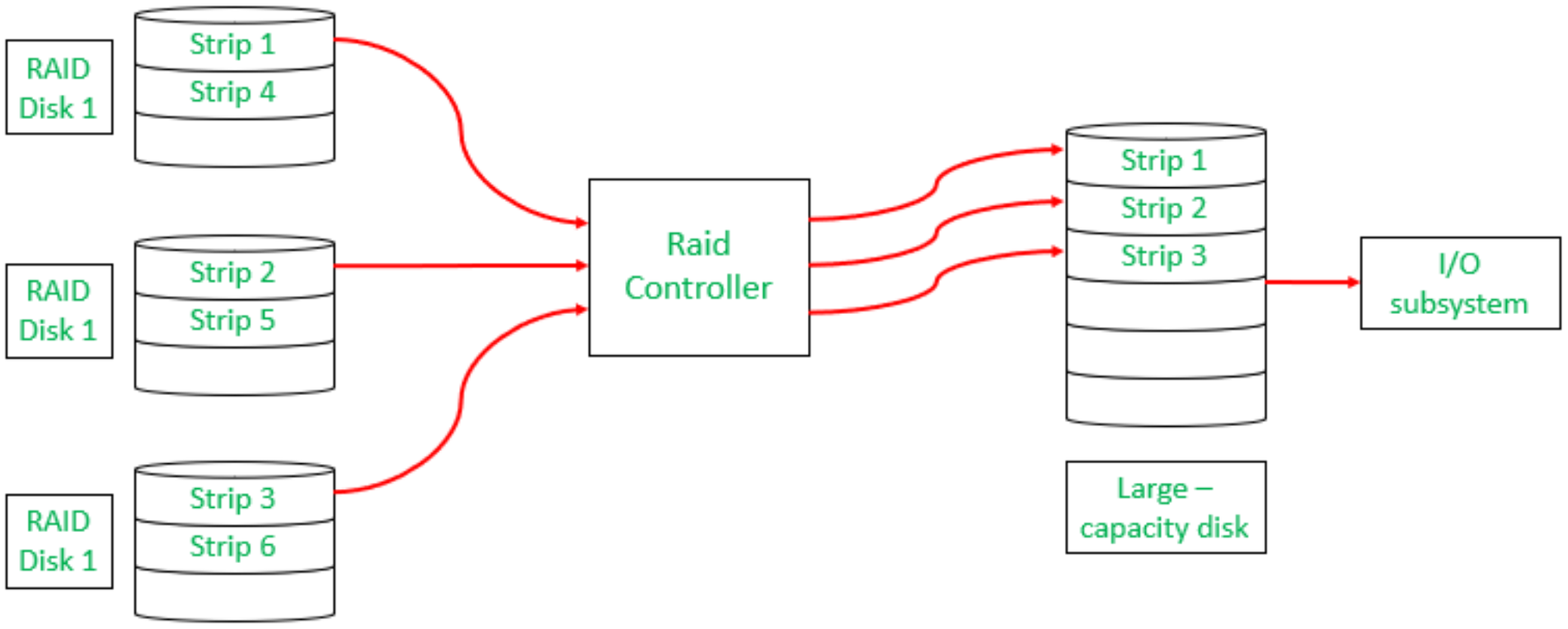
Redundant Array of Independent Disks

RAID

- RAID (Redundant Array of Independent Disks) is like having backup copies of your important files stored in different places on several hard drives or solid-state drives (SSDs).
- If one drive stops working, your data is still safe because you have other copies stored on the other drives. It's like having a safety net to protect your files from being lost if one of your drives breaks down.
- RAID (Redundant Array of Independent Disks) in a Database Management System (DBMS) is a technology that combines multiple physical disk drives into a single logical unit for data storage.
- The main purpose of RAID is to improve data reliability, availability, and performance. There are different levels of RAID, each offering a balance of these benefits.

RAID Controller

- A RAID controller is like a boss for your hard drives in a big storage system.
- It works between your computer's operating system and the actual hard drives, organizing them into groups to make them easier to manage.
- This helps speed up how fast your computer can read and write data, and it also adds a layer of protection in case one of your hard drives breaks down.
- So, it's like having a smart helper that makes your hard drives work better and keeps your important data safer


RAID
Disk 1
Strip 1
Strip 4
RAID
Disk 1
Strip 2
Strip 5
RAID
Disk 1
Strip 3
Strip 6
Raid
Controller
Strip 1
Strip 2
Strip 3
I/O
subsystem
Large –
capacity disk

REDUNDANT ARRAY OF INDEPENDENT DISKS(RAID)

RAID or **R**edundant **A**rray of **I**ndependent **D**isks, is a technology to connect multiple secondary storage devices and use them as a single storage media.

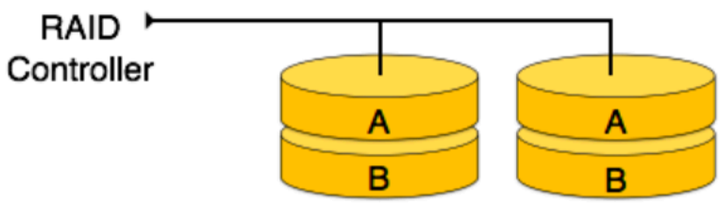
RAID consists of an array of disks in which multiple disks are connected together to achieve different goals. RAID levels define the use of disk arrays.

RAID 0

In this level, a striped array of disks is implemented. The data is broken down into blocks and the blocks are distributed among disks. Each disk receives a block of data to write/read in parallel. It enhances the speed and performance of the storage device. There is no parity and backup in Level 0.

RAID 1

RAID 1 uses mirroring techniques. When data is sent to a RAID controller, it sends a copy of data to all the disks in the array. RAID level 1 is also called **mirroring** and provides 100% redundancy in case of a failure.

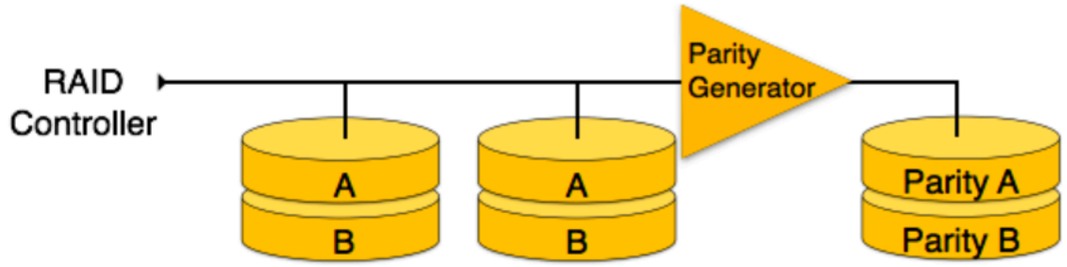


RAID 2

RAID 2 records Error Correction Code using Hamming distance for its data, striped on different disks. Like level 0, each data bit in a word is recorded on a separate disk and ECC codes of the data words are stored on a different set disks. Due to its complex structure and high cost, RAID 2 is not commercially available.

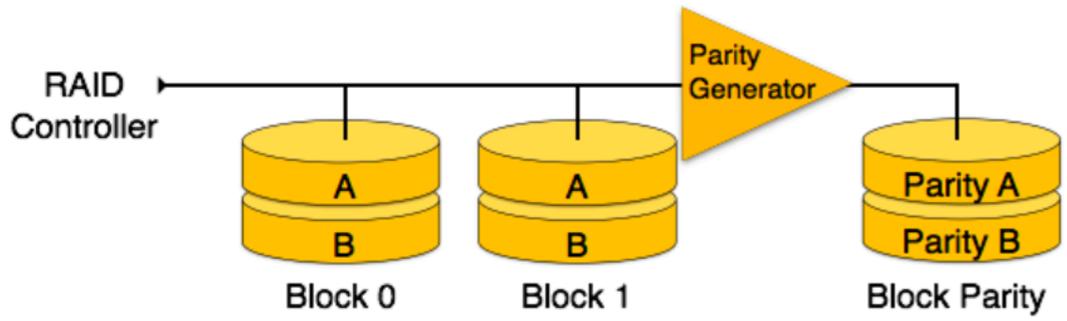
RAID 3

RAID 3 stripes the data onto multiple disks. The parity bit generated for data word is stored on a different disk. This technique makes it to overcome single disk failures.



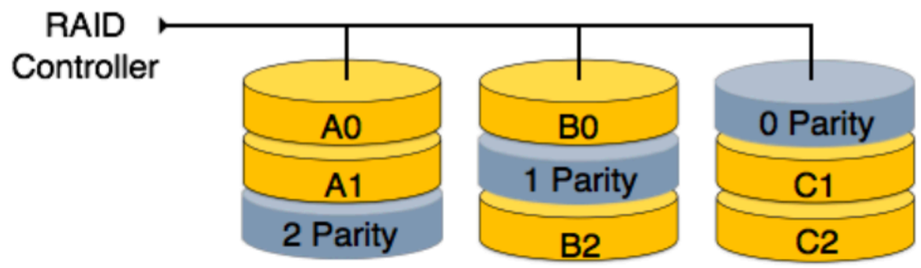
RAID 4

In this level, an entire block of data is written onto data disks and then the parity is generated and stored on a different disk. Note that level 3 uses byte-level striping, whereas level 4 uses block-level striping. Both level 3 and level 4 require at least three disks to implement RAID.



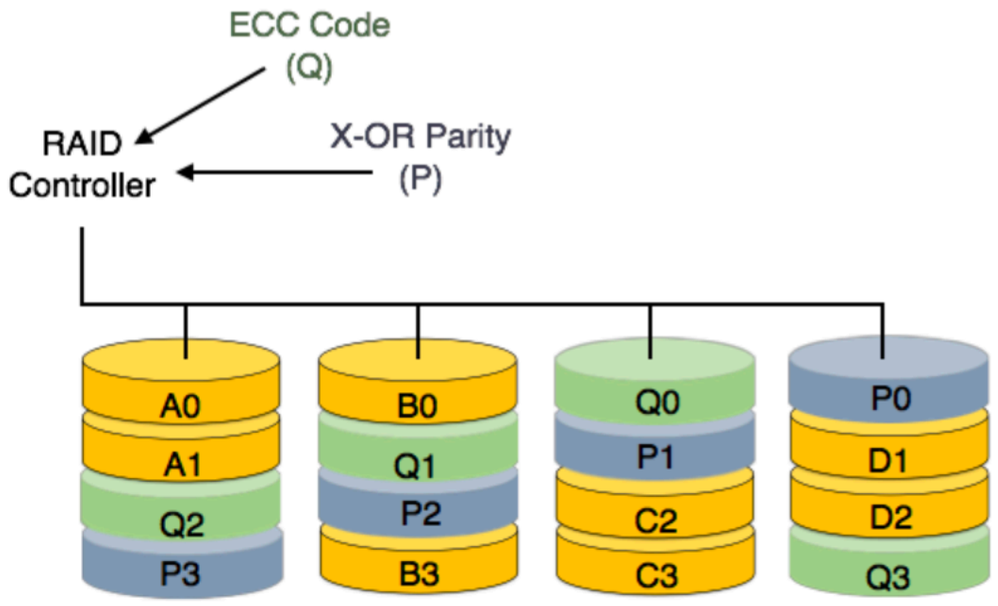
RAID 5

RAID 5 writes whole data blocks onto different disks, but the parity bits generated for data block stripe are distributed among all the data disks rather than storing them on a different dedicated disk.



RAID 6

RAID 6 is an extension of level 5. In this level, two independent parities are generated and stored in distributed fashion among multiple disks. Two parities provide additional fault tolerance. This level requires at least four disk drives to implement RAID.



RAID – Redundant Array of Independent Disks

- The storage which we are used in laptop computers are single disk but in servers, data centers, cloud computing we used multiple disk, means RAID technology.
- RAID saves data in multiple disk.
- RAID is a set of multiple hard disk, which are in the network.
- As additional memory in form of cache, can improve system performance, in the same way additional disks can also improve system performance.
- In RAID, we use an array (multiple) of disks. These disks operate independently.
- Since there are many disks, separate and multiple I/O requests can be handled in parallel if the data required is on separate disks.
- A single I/O operation can be handled in parallel if the data required is distributed across multiple disks

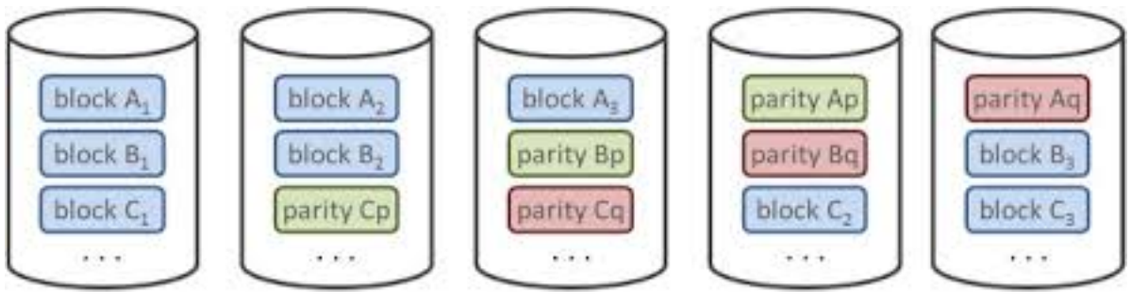
Benefits of RAID:

- RAID technology prevents data loss due to disk failure.
- RAID technology can be implemented in hardware or software.
- Servers make use of RAID technology.
- To prevent fail of Operating system RAID technology is beneficial.

PARITY CHECK: A parity check is the process in which accurate data transmission between nodes during communication is checked.

HOW TO PARITY CHECK:

1. A parity bit is appended to the original data bits to create an even or odd bit number; the number of bits with value one.
2. The source then transmits this data via a link to the destination.
3. Transferred bits are checked and verified at the destination.
4. If number of bits at destination are same as the number of bits at source than data is considered as accurate.



To know more about RAID, redirect to TechTarget website

RAID resources

02.

FILE ORGANIZATION

2.1 TERMINOLOGIES

- The database is stored as a collection of *files*.
  - Each file is a sequence of *records.*
    - A record is a sequence of fields.

- One approach
  - Assume record size is fixed
  - Each file has records of one particular type only
  - Different files are used for different relations

  This case is easiest to implement; will consider variable length records later

- We assume that records are smaller than a disk block

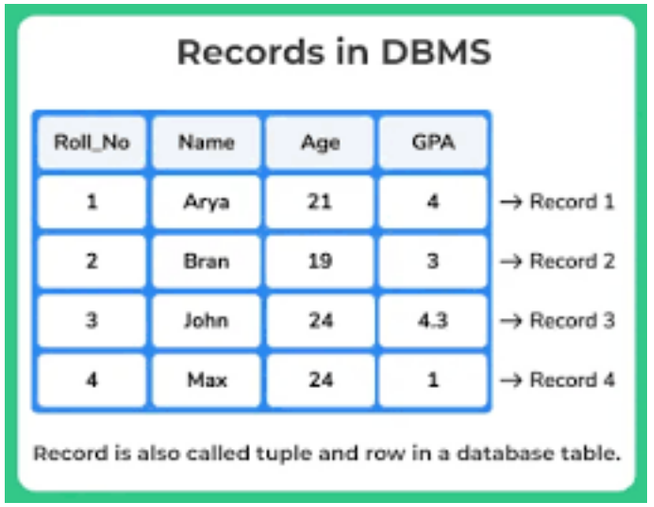
Example,

- Assume that each character occupies 1 byte and that numeric (8,2) occupies 8 bytes.
- Let's assume we allocate the maximum number of bytes that each attribute can hold.
- Then, the *instructor* record is 53 bytes long. A simple approach is to use the first 53 bytes for the first record, the next 53 bytes for the second record, and so on

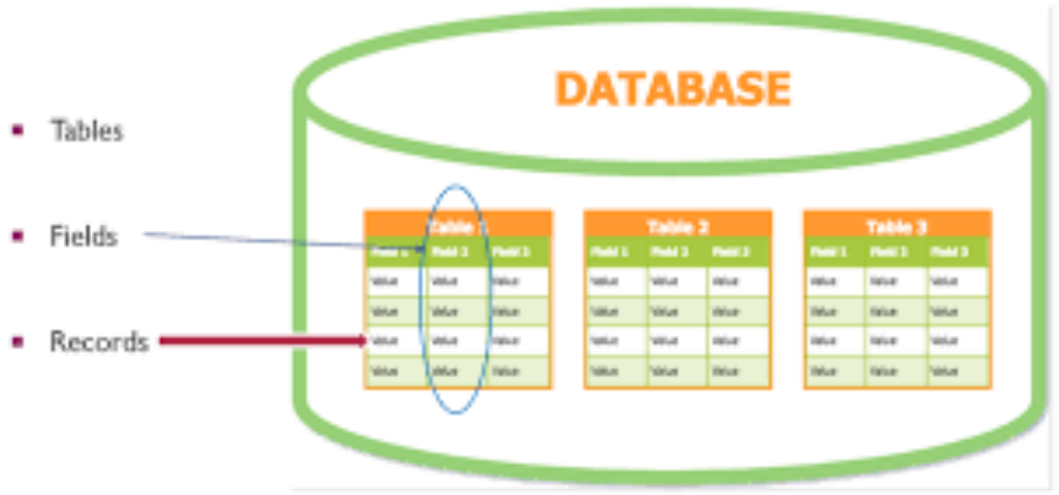
```
type instructor = record
                      ID varchar (5);
                      name varchar(20);
                      dept_name varchar (20);
                      salary numeric (8,2);
                  end
```

2.2 RECORDS

DEFINITION & TYPES



- A record is sequence of files.
- It is basically classified into two types,
  - Fixed - length records
  - Variable - length records



Fixed Length Records

- Simple approach:
  - Store record *i* starting from byte $n * (i - 1)$, where *n* is the size of each record.
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed Length Records (Cont.)

- Deletion of record *i:* alternatives*:*
  - **move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1**
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

Record 3 deleted

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Fixed Length Records (Cont.)

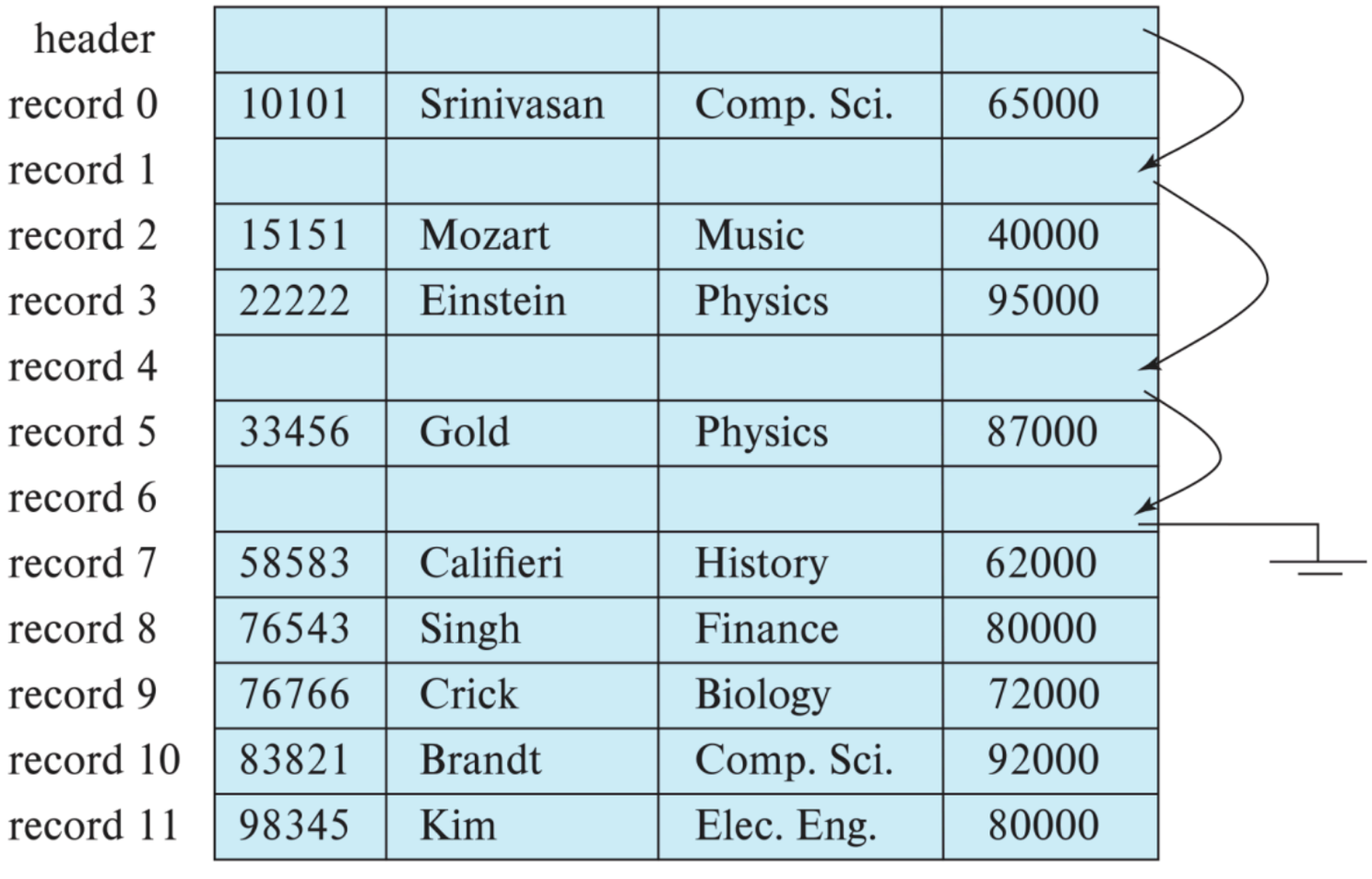
- Deletion of record *i:* alternatives*:*
  - move records *i* + 1, . . ., *n* to *i, . . . , n* – 1
  - **move record *n* to *i***
  - do not move records, but link all free records on a *free list*

Record 3 deleted and replaced by record 11

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Fixed Length Records (Cont.)

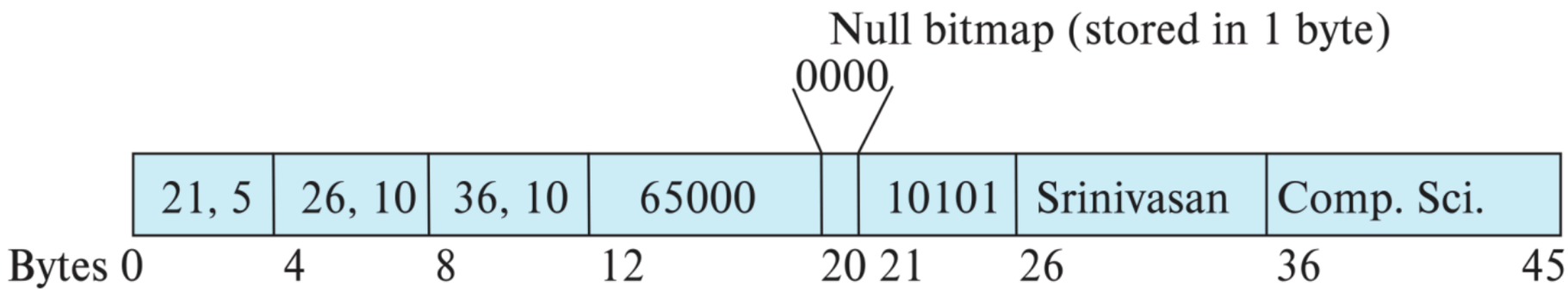
- Deletion of record *i:* alternatives*:*
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record n to i
  - **do not move records, but link all free records on a *free list***



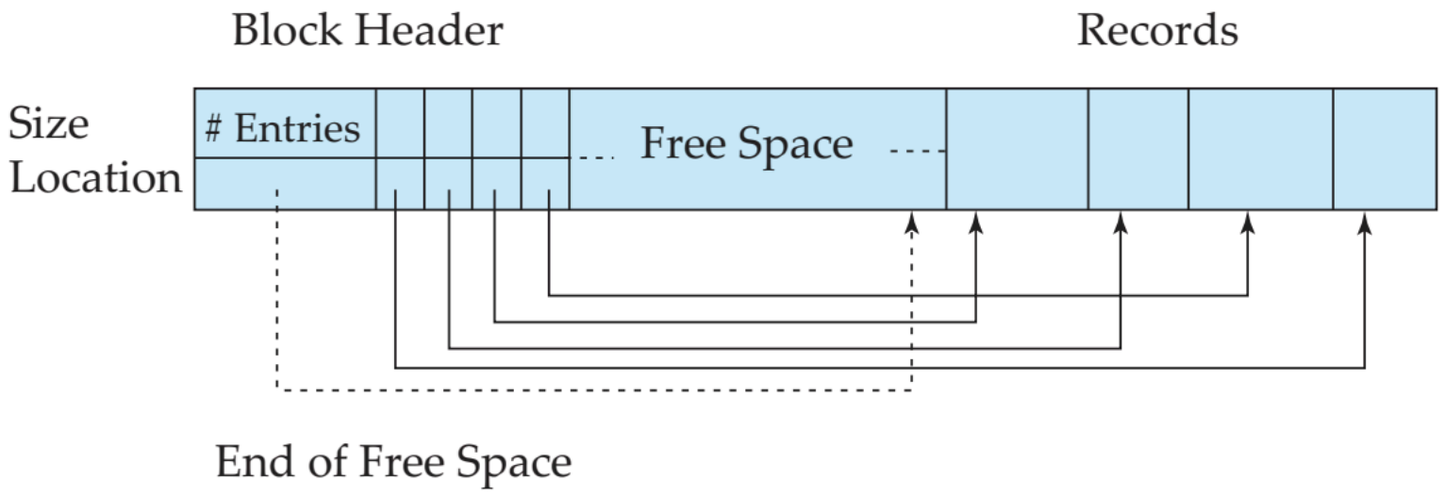
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Variable Length Records

- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**)
  - Record types that allow repeating fields (used in some older data models).
- Attributes are stored in order
- Variable length attributes represented by fixed size (offset, length), with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap , which indicates which attributes of the record have a null value.



VLR - Slotted Page Structure



- **Slotted page** header contains:
  - number of record entries
  - end of free space in the block
  - An array whose entries contain the location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Records can be grown or shrunk, as long as there is space in the block.
  - The cost of moving the records is not too high, since the size of a block is limited: typical values are around 4 to 8 kilobytes.

How to store large objects?

- E.g., blob/clob types
- Records must be smaller than pages
- Alternatives:
  - Store as files in file systems
  - Store as files managed by database
  - Break into pieces and store in multiple tuples in separate relation
    - PostgreSQL TOAST

How are records organised in files?

FILE ORGANISATION TECHNIQUES

- **Heap** – record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O
- **B+-tree file organization**
  - Ordered storage even with inserts/deletes
- **Hashing** – a hash function computed on search key; the result specifies in which block of the file the record should be placed

Btw, Why do we need to organise the files?

Objective of Organisation

Storing the files in certain order is called file organization. The main objective of file organization is

- Optimal selection of records i.e.; records should be accessed as fast as possible.
- Any insert, update or delete transaction on records should be easy, quick and should not harm other records.
- No duplicate records should be induced as a result of insert, update or delete
- Records should be stored efficiently so that cost of storage is minimal.

2.3 File Organisation Methods

6 Methods

Some of the file organizations are

1. Sequential File Organization
2. Heap File Organization
3. Hash/Direct File Organization
4. Indexed Sequential Access Method
5. B+ Tree File Organization
6. Cluster File Organization

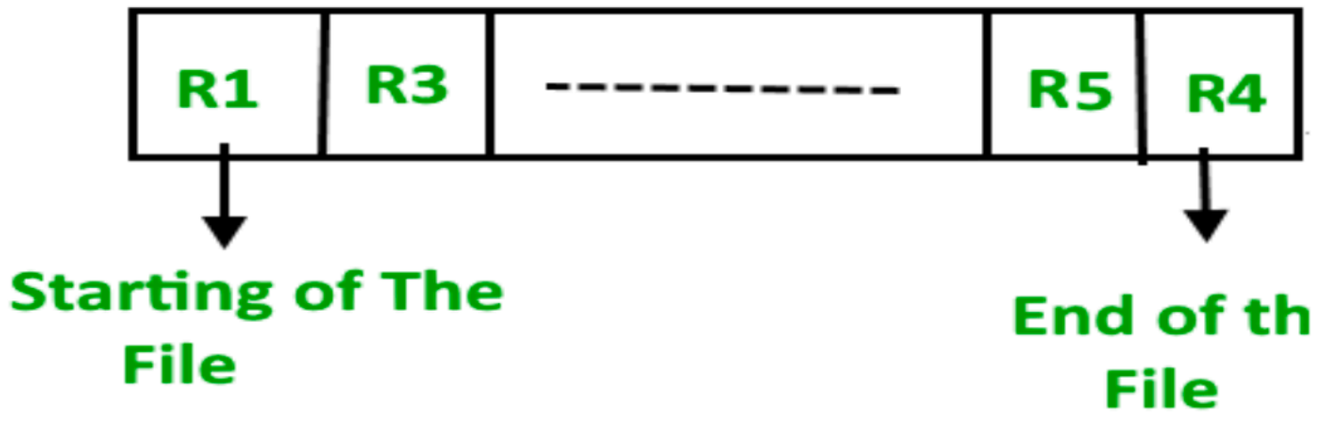
2.3.1 Sequential File Organisation

In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key.

The easiest method for file Organization is Sequential method. In this method the the file are stored one after another in a sequential manner. There are two ways to implement this method:

1. Pile FIle Method
2. Sorted File

1. **Pile File Method –** This method is quite simple, in which we store the records in a sequence i.e one after other in the order in which they are inserted into the tables.



Sequential File Organisation (Cont.)

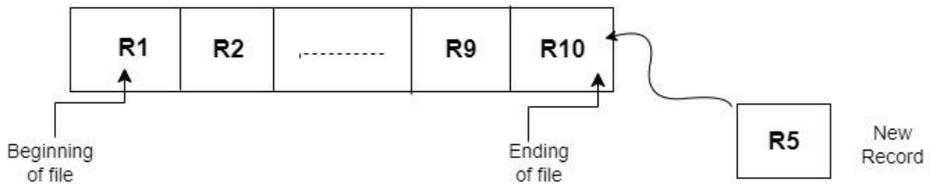
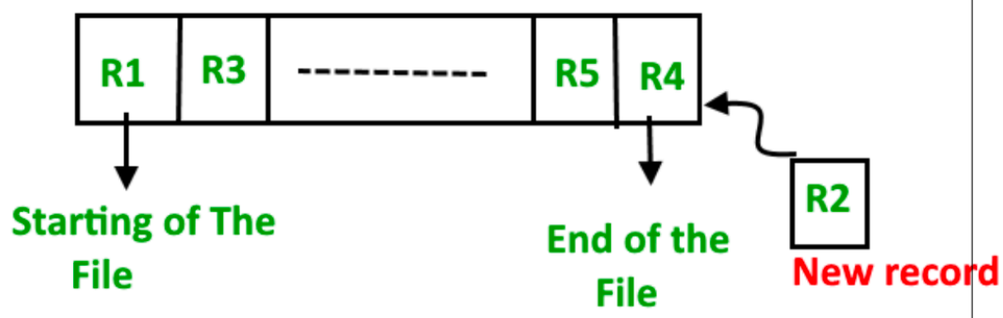


Fig: Insertion of record in pile file method

Insertion of new record –
Let the R1, R3 and so on upto R5 and R4 be four records in the sequence. Here, records are nothing but a row in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.



2. **Sorted File Method –**In this method, As the name itself suggest whenever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner. Sorting of records may be based on any primary key or any other key.

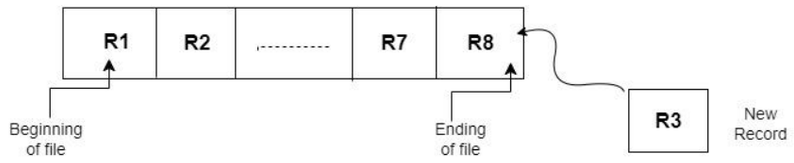
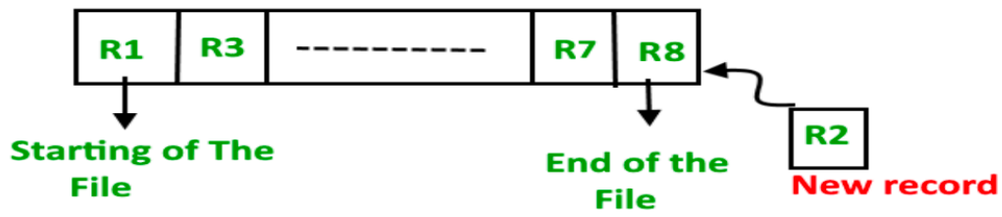


Fig: Insertion of record in sorted file method



Insertion of new record –
Let us assume that there is a preexisting sorted sequence of four records R1, R3, and so on upto R7 and R8. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence .

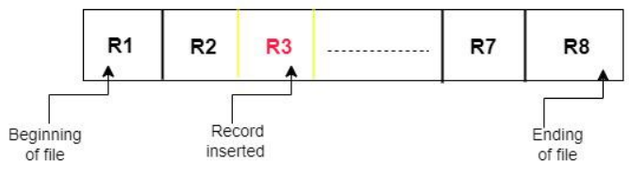
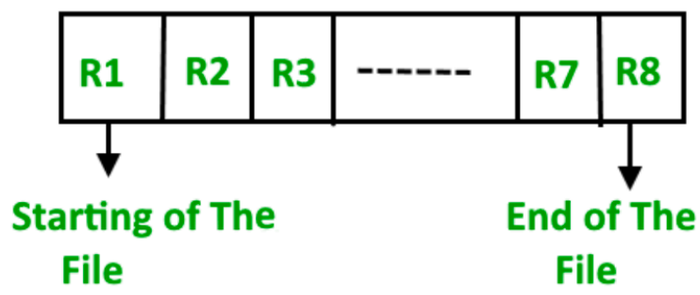


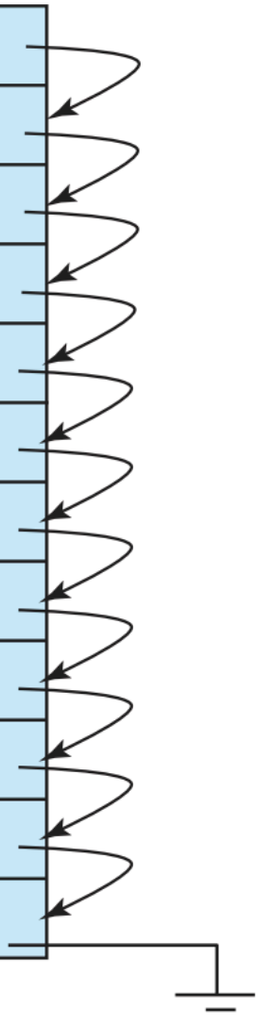
Fig: Record inserted in sorted manner



Sequential File Organisation (Cont.)

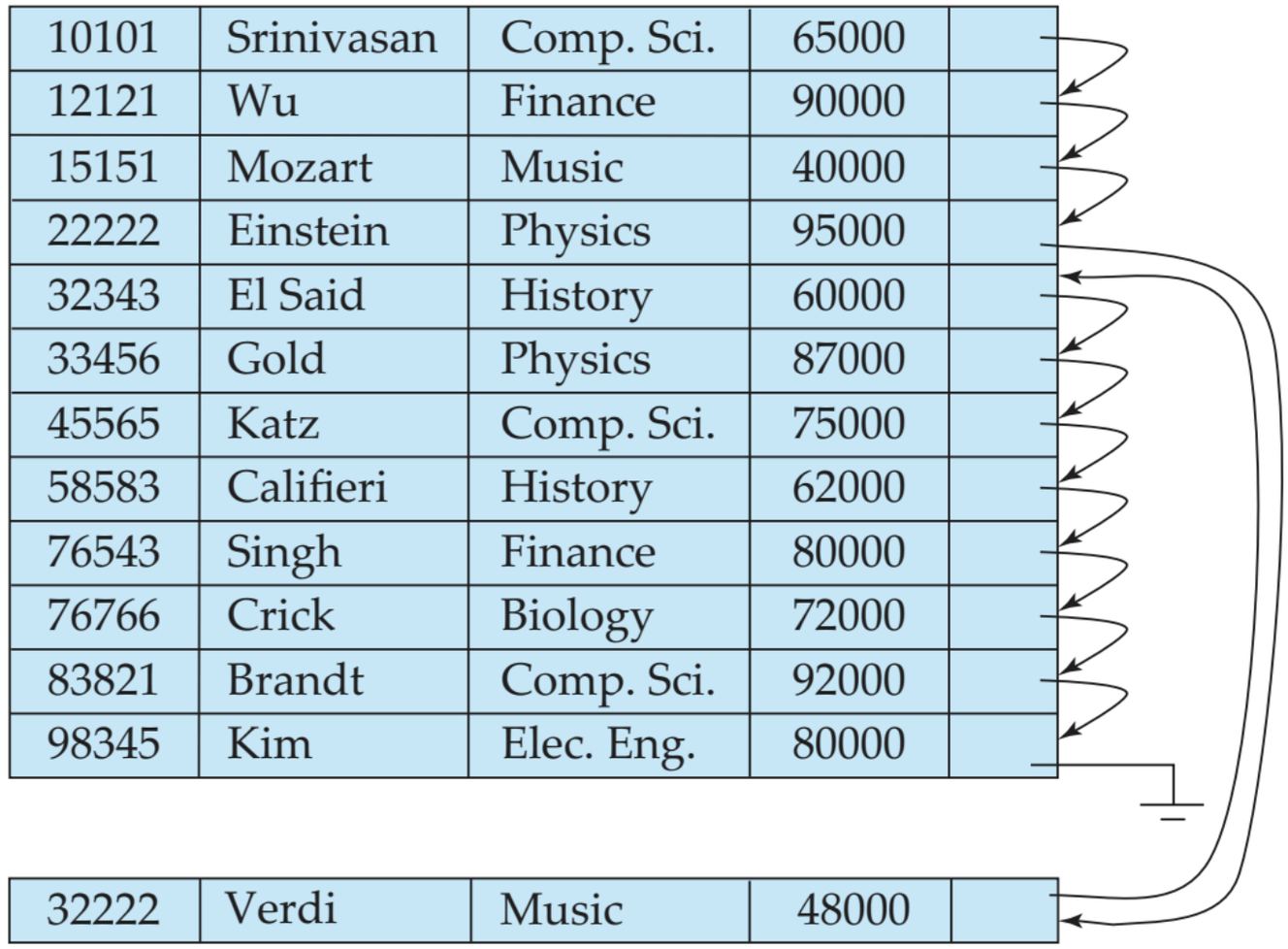
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	



Sequential File Organisation (Cont.)

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order



Sequential File Organisation (Cont.)

Advantages & Disadvantages

Pros –

- Fast and efficient method for huge amount of data.
- Simple design.
- Files can be easily stored in magnetic tapes i.e cheaper storage mechanism.

Cons –

- Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner which takes our time.
- Sorted file method is inefficient as it takes time and space for sorting records.

Advantages of Sequential File Organization

- **Design:** Compared to other file organization methods, sequential file organization boasts a simple design and minimal complexity, making it easy to develop and implement.
- **Small Datasets:** Sequential processing, can be efficient for small amounts of data. But for large datasets, other methods might be more efficient such as hash file and heap file.
- **Accuracy:** Sequential file organization is inherently accurate as it maintains a simple order, and there's no manipulation involved that could introduce errors.
- **Processing:** For generating reports or calculations that require processing every record, sequential file organization is straightforward and accurate.

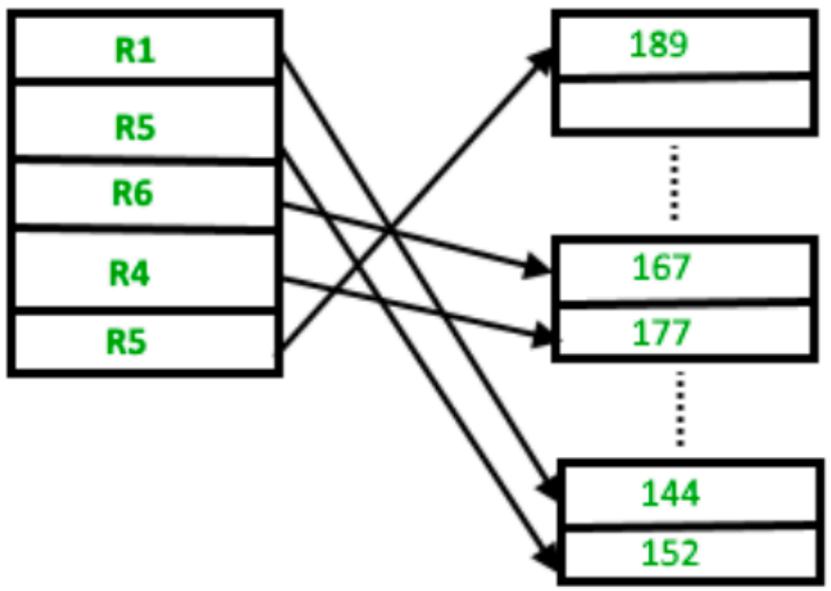
Disadvantages of Sequential File Organization

- **Time-Consuming:** Random record access within a pile file presents a time-consuming challenge. Unlike indexed methods, each record must be sequentially traversed from left to right.
- **Memory Wastage:** When a record is deleted the space it occupied usually remains unused. This can lead to memory waste, especially for large datasets with frequent deletions, and act as an inefficient memory method.
- **System Processing:** While using sorted files the record must always be sorted and adding or changing records often requires re-sorting the entire set, which can slow down your system, especially for larger files.

2.3.2 Heap File Organisation

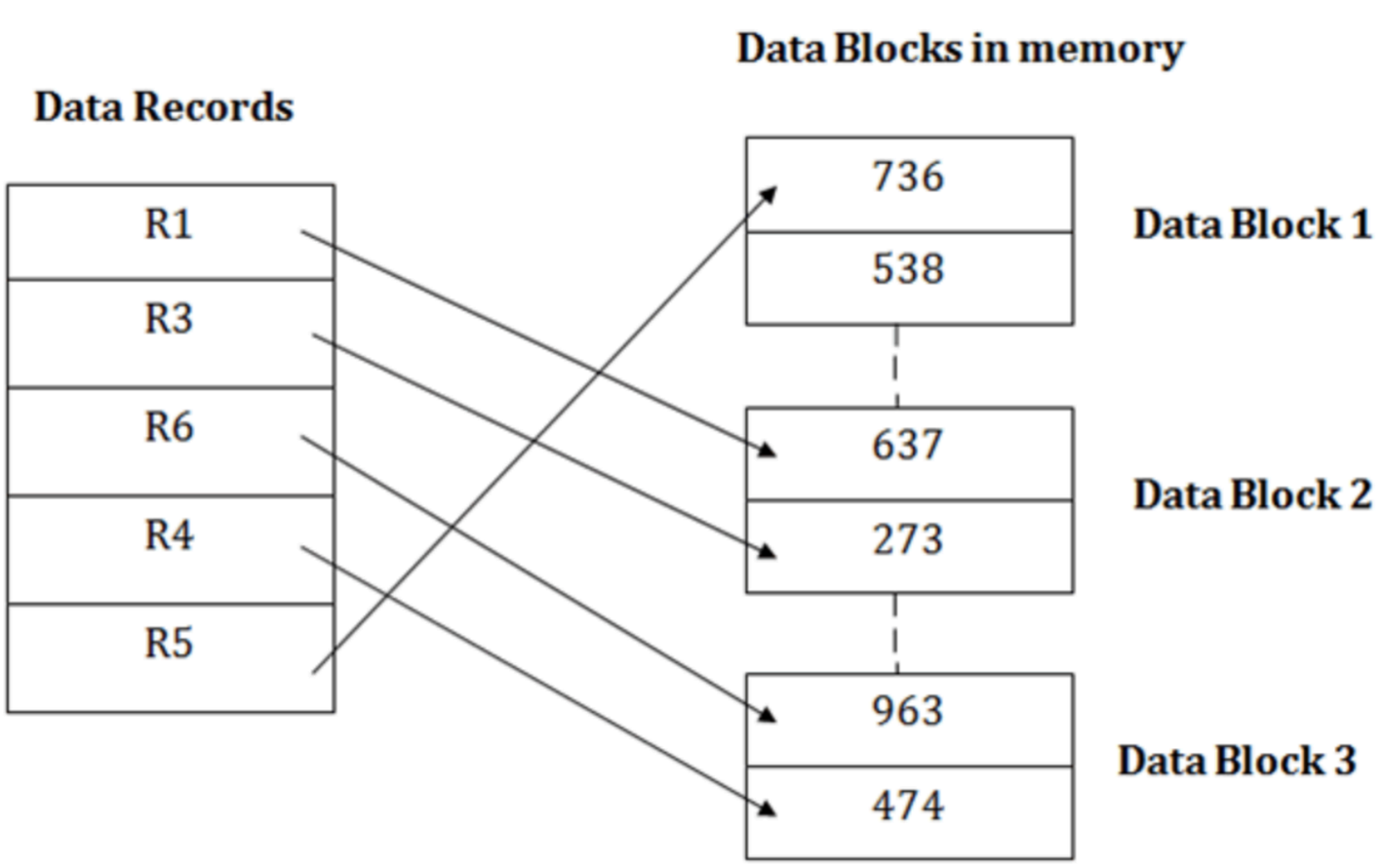
When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area.

Heap File Organization works with data blocks. In this method records are inserted at the end of the file, into the data blocks. No Sorting or Ordering is required in this method. If a data block is full, the new record is stored in some other block, Here the other data block need not be the very next data block, but it can be any block in the memory. It is the responsibility of DBMS to store and manage the new records.



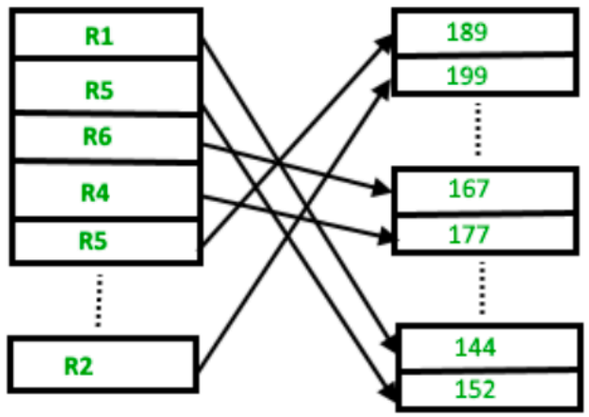
Heap File Organisation (Cont.)

- It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.
- When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but it can select any data block in the memory to store new records. The heap file is also known as an unordered file.
- In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Heap File Organisation (Cont.)

Insertion of new record –
Suppose we have four records in the heap R1, R5, R6, R4 and R3 and suppose a new record R2 has to be inserted in the heap then, since the last data block i.e data block 3 is full it will be inserted in any of the database selected by the DBMS, lets say data block 1.



If we want to search, delete or update data in heap file Organization the we will traverse the data from the beginning of the file till we get the requested record. Thus if the database is very huge, searching, deleting or updating the record will take a lot of time.

Pros and Cons of Heap File Organization –
Pros –

- Fetching and retrieving records is faster than sequential record but only in case of small databases.
- When there is a huge number of data needs to be loaded into the database at a time, then this method of file Organization is best suited.

Cons –

- Problem of unused memory blocks.
- Inefficient for larger databases.

Heap File Organisation (Cont.)

- Records can be placed anywhere in the file where there is free space
- Records usually do not move once allocated
- Important to be able to efficiently find free space within file
- **Free-space map is** to track which blocks have free space to store records.
  - Array with 1 entry per block. Each entry is a few bits to a byte, and records fraction of block that is free
  - If 3 bits are used to store the occupancy fraction; the value at position *i* should be divided by 8 to get the free-space fraction for block *i*.
  - For example, a value of 7 indicates that at least 7/8*th* of the space in the block is free.

4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

  - Can have second-level free-space map. In example below, each entry stores maximum from 4 entries of first-level free-space map

4	7	2	6

- Free space map written to disk periodically, OK to have wrong (old) values for some entries (will be detected and fixed)

2.3.3 Hash File Organisation

Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.

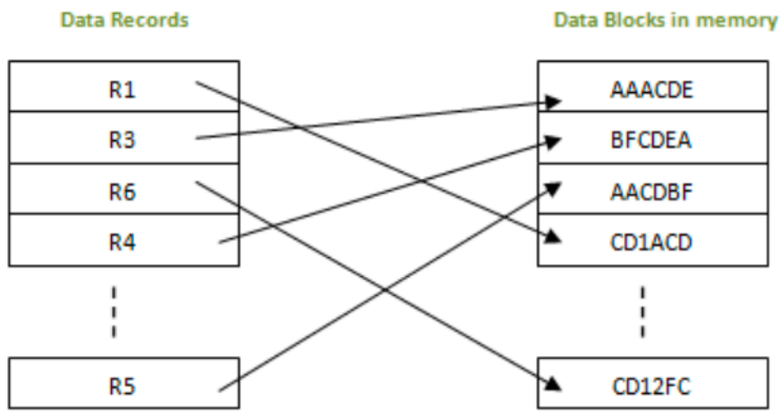
In this method of file organization, hash function is used to calculate the address of the block to store the records.

The hash function can be any simple or complex mathematical function.

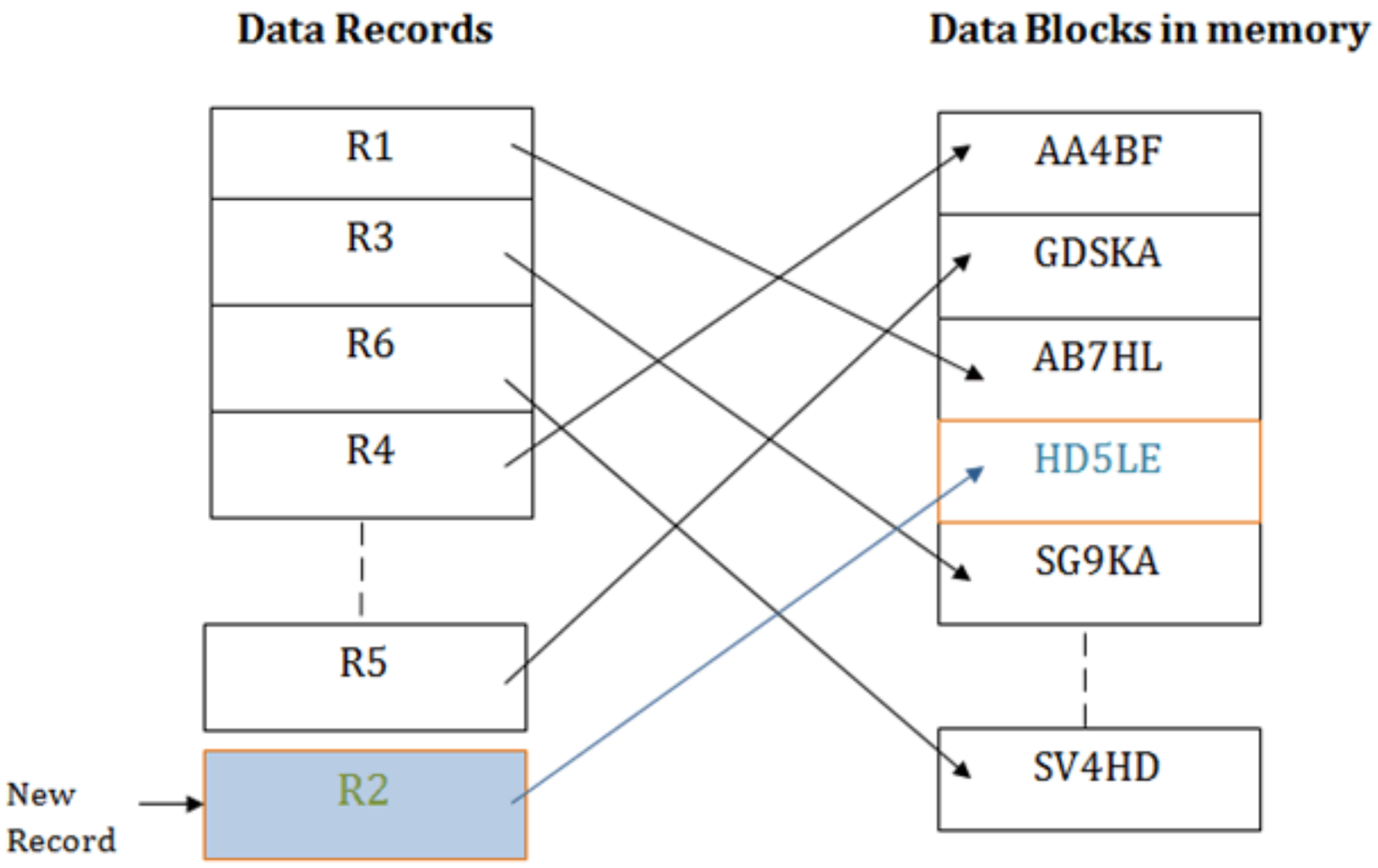
The hash function is applied on some columns/attributes – either key or non-key columns to get the block address.

Hence each record is stored randomly irrespective of the order they come. Hence this method is also known as Direct or Random file organization.

If the hash function is generated on key column, then that column is called hash key, and if hash function is generated on non-key column, then the column is hash column.



When a record has to be retrieved, based on the hash key column, the address is generated and directly from that address whole record is retrieved. Here no effort to traverse through whole file. Similarly when a new record has to be inserted, the address is generated by hash key and record is directly inserted. Same is the case with update and delete.


Data Records
Data Blocks in memory
R1
R3
R6
R4
R5
R2
New
Record
AA4BF
GDSKA
AB7HL
HD5LE
SG9KA
SV4HD

Terms in Hash File Organisation

Hash file organization may be a strategy of putting away and getting to information in a record employing a hash work to calculate the address of the information within the record.

This permits for fast recovery of information based on a key.

In Hashing we mainly refer the following terms:

- **Data Bucket:** A data bucket is a storage location where records are stored. These buckets are also considered storage units.

- **Hash Function:** A hash function is a mapping function that maps all search keys to actual record addresses. Generally, a hash function uses a primary key to generate a hash index (address of a data block). Hash functions range from simple to complex mathematical functions.

- **Hash Index:** The prefix of the entire hash value is used as the hash index. Each hash index has a depth value that indicates the number of bits used to calculate the hash function.

Advantages & Disadvantages

Hashing File Organisation

Advantages

- Records need not be sorted after any of the transaction. Hence the effort of sorting is reduced in this method.
- Since block address is known by hash function, accessing any record is very faster. Similarly updating or deleting a record is also very quick.

- This method can handle multiple transactions as each record is independent of other. i.e.; since there is no dependency on storage location for each record, multiple records can be accessed at the same time.
- It is suitable for online transaction systems like online banking, ticket booking system etc.

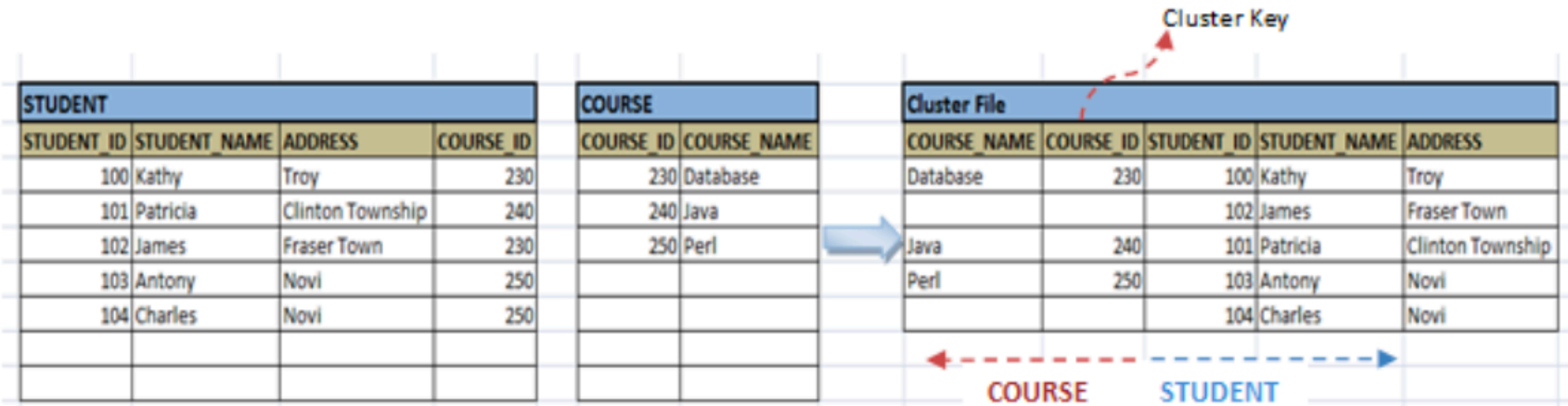
Disadvantages of Hash File Organization

1. Can cause accidental deletion of data, if columns are not selected properly for hash function. For example, while deleting an Employee `"Steve"` using `Employee_Name` as hash column can cause accidental deletion of other employee records if the other employee name is also `"Steve"`. This can be avoided by selecting the attributes properly, for example in this case combining age, department or SSN with the employee_name for hash key can be more accurate in finding the distinct record.
2. Memory is not efficiently used in hash file organization as records are not stored in consecutive memory locations.
3. If there are more than one hash columns, searching a record using a single attribute will not give accurate results.

2.3.4 Cluster File Organisation

Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

In this method two or more table which are frequently used to join and get the results are stored in the same file called clusters. These files will have two or more tables in the same data block and the key columns which map these tables are stored only once. This method hence reduces the cost of searching for various records in different files. All the records are found at one place and hence making search efficient.



Cluster File Organisation (Cont.)

Store several relations in one file using a **multitable clustering** file organization

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

multitable clustering of *department* and *instructor*

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

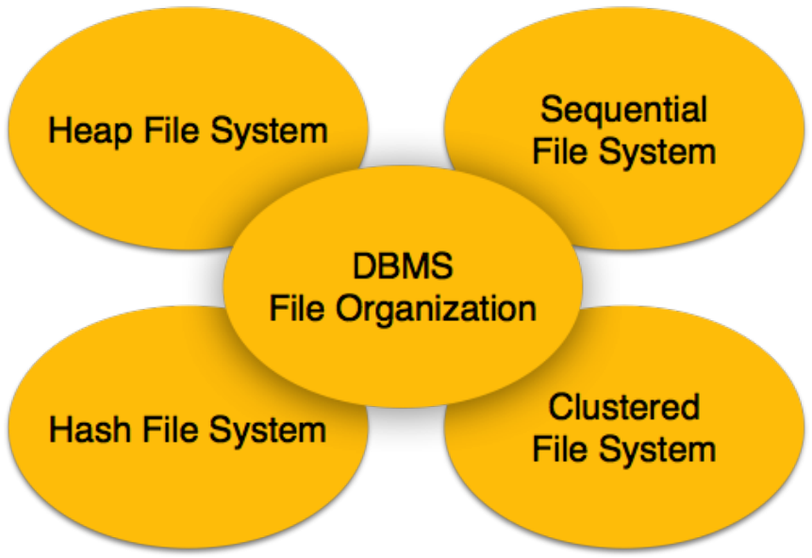
Cluster File Organisation (Cont.)

- Good for queries involving *department* ⋈ *instructor*, and for queries involving one single department and its instructors
- Bad for queries involving only, for example, *department*
  - Would require more block accesses since each block now contains significantly fewer *department* records.
- Results in variable size records
- Can add pointer chains to link records of a particular relation

COMPARISON
of File Organisation

The operations to be considered for comparisons of file organizations are below:

- **Scan:** Fetch all records in the file. The pages in the file must be fetched from disk into the buffer pool. There is also a CPU overhead per record for locating the record on the page (in the pool).
- **Search with equality selection:** Fetch all records that satisfy an equality selection, for example, "Find the Students record for the student with *sid* 23." Pages that contain qualifying records must be fetched from disk, and qualifying records must be located within retrieved pages.
- **Search with range selection:** Fetch all records that satisfy a range selection, for example, "Find all Students records with *name* alphabetically after 'Smith.' "
- **Insert:** Insert a given record into the file. We must identify the page in the file into which the new record must be inserted, fetch that page from disk, modify it to include the new record, and then write back the modified page. Depending on the file organization, we may have to fetch, modify, and write back other pages as well.
- **Delete:** Delete a record that is specified using its rid. We must identify the page that contains the record, fetch it from disk, modify it, and write it back. Depending on the file organization, we may have to fetch, modify, and write back other pages as well.



File Type	*Scan*	*Equality Search*	*Range Search*	*Insert*	*Delete*
Heap	BD	$0.5BD$	BD	$2D$	$Search + D$
Sorted	BD	$Dlog_2B$	$Dlog_2B$+# *matches*	$Search + BD$	$Search + BD$
Hashed	$1.25BD$	D	$1.25BD$	$2D$	$Search + D$

Figure 8.1 A Comparison of I/O Costs

B - number of data pages

R records per page

D- average time to read or write a disk page

C- average time to process a record

Partitioning

- **Table partitioning**: Records in a relation can be partitioned into smaller relations that are stored separately
- E.g., *transaction* relation may be partitioned into *transaction_2018, transaction_2019, etc.*
- Queries written on *transaction* must access records in all partitions
  - Unless query has a selection such as *year*=2019, in which case only one partition in needed
- Partitioning
  - Reduces costs of some operations such as free space management
  - Allows different partitions to be stored on different storage devices
    - E.g., *transaction* partition for current year on SSD, for older years on magnetic disk

Data Dictionary Storage

DEFINITION

The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as

- Information about relations
  - names of relations
  - names, types and lengths of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential/hash/…)
  - Physical location of relation
- Information about indices