SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 19CS402 - DATABASE MANAGEMENT SYSTEMS

II YEAR / III SEMESTER

Unit – 5

Storage Devices

P.REVATHI/AP/AIDS

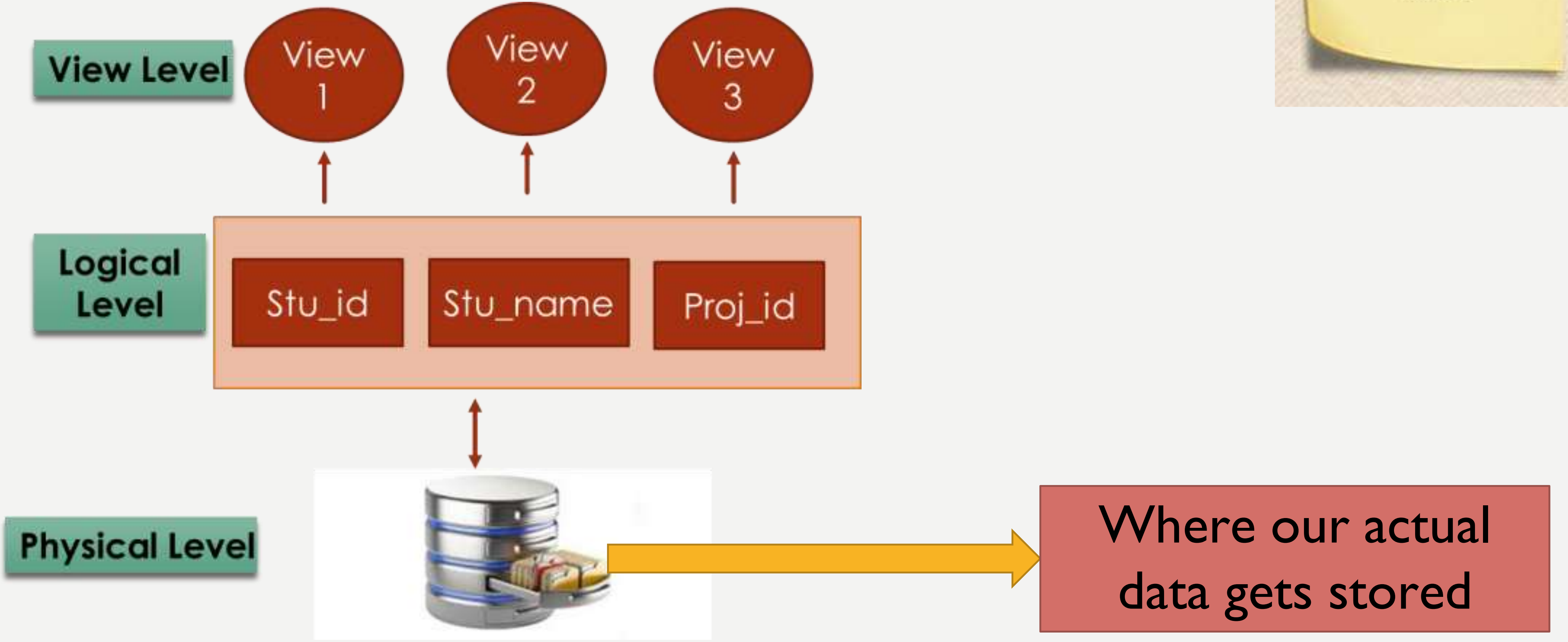
UNIT V

STORAGE & INDEXING

MODULE - IV	**STORAGE & INDEXING**	**10**
Overview of Storage Techniques – file organization - RAID –Indexing - Types of ordered indices - B & B+ tree – Hashing - Static & Dynamic Hashing. Query Processing & Optimization		

OVERVIEW OF STORAGE TECHNIQUES

Remember
This!



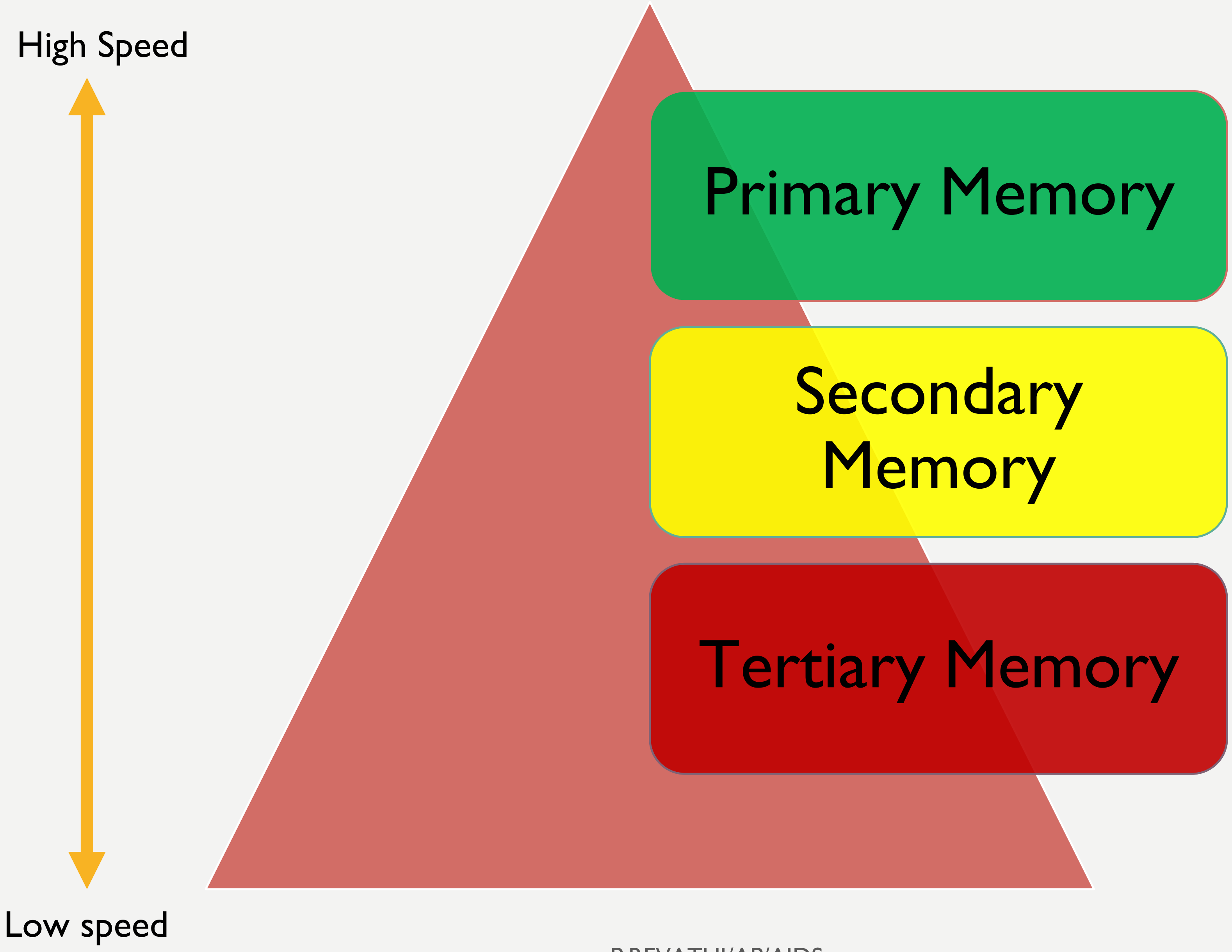
View Level
View 1
View 2
View 3
Logical Level
Stu_id
Stu_name
Proj_id
Physical Level
Where our actual data gets stored

DEVICES

your
data

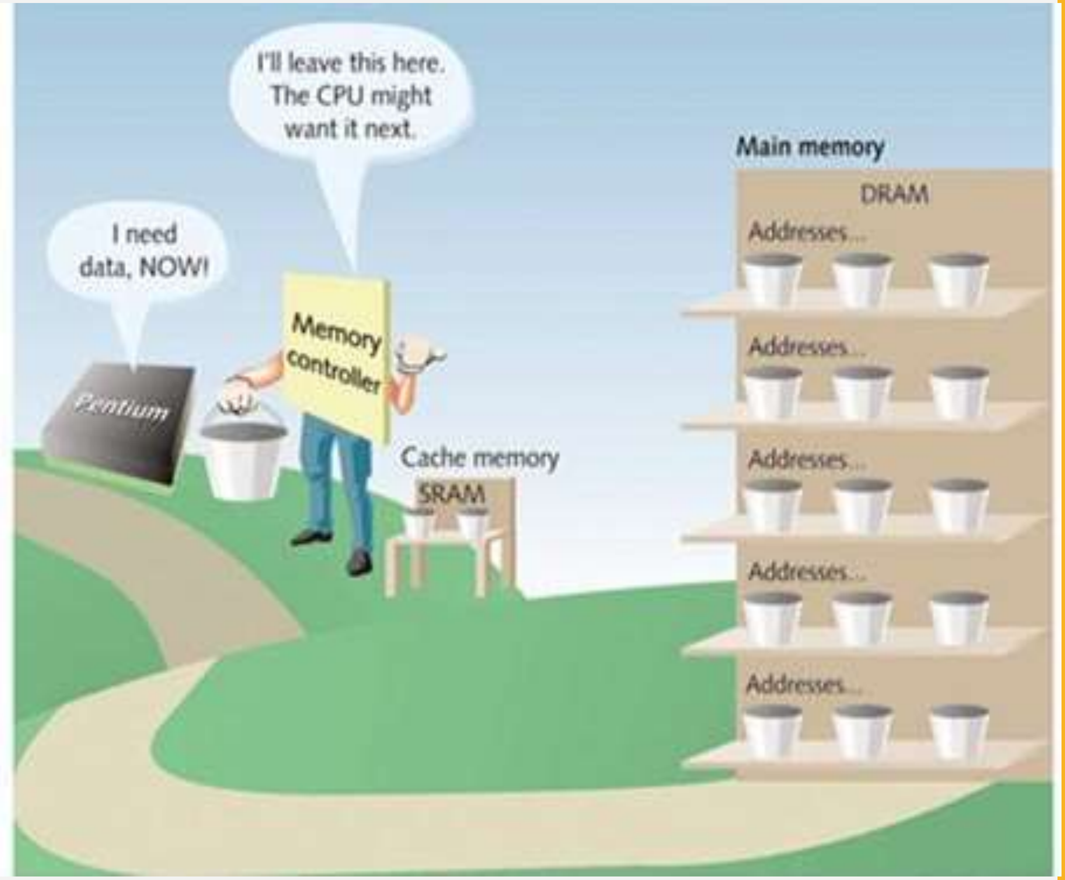
STORAGE

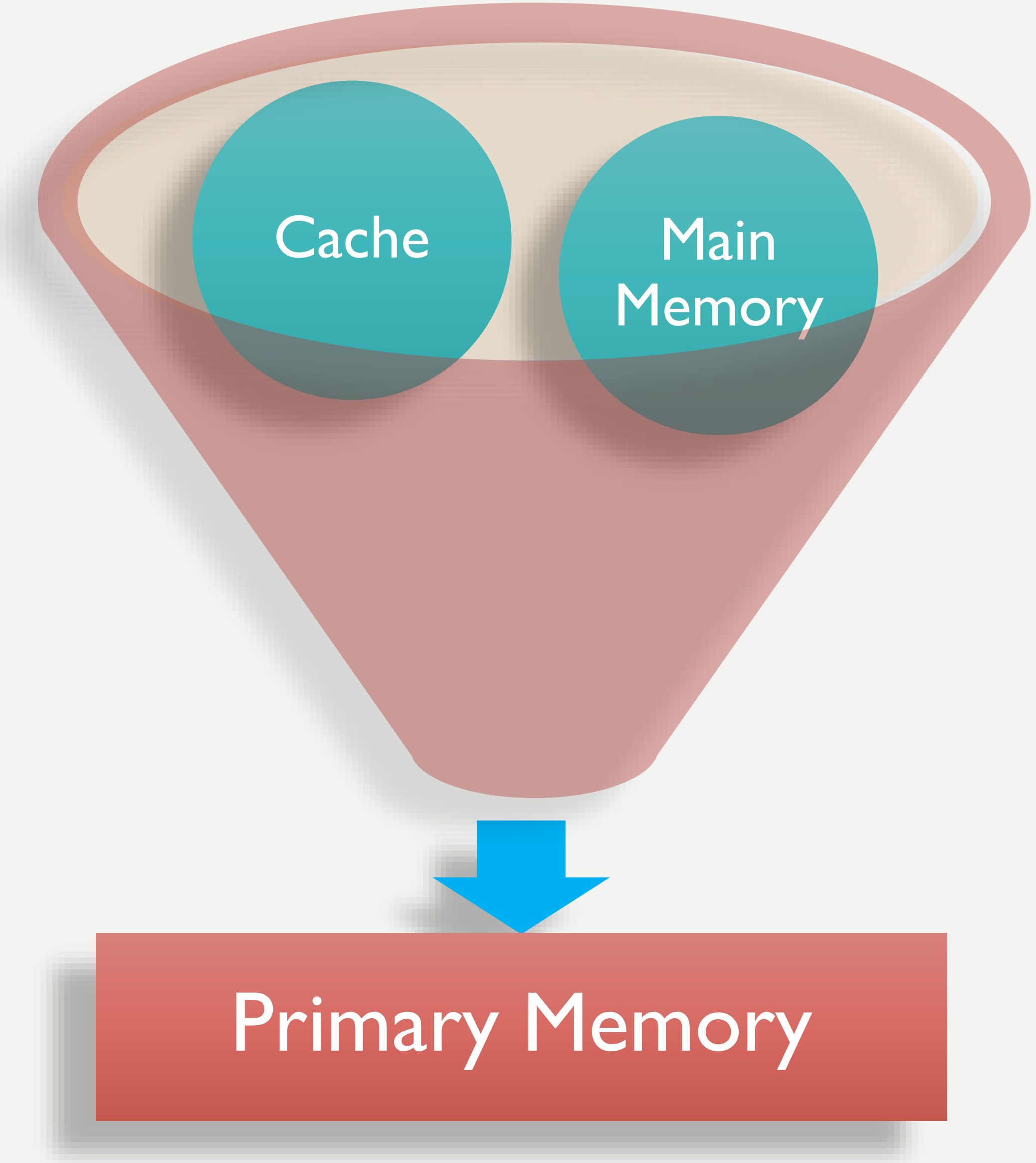
- data in the form of bits, bytes get stored in different storage devices.
- A database system provides an ultimate view of the stored data
- For storing the data, there are different types of storage options available
- These storage types differ from one another as per the speed and accessibility


High Speed
Primary Memory
Secondary Memory
Tertiary Memory
Low speed

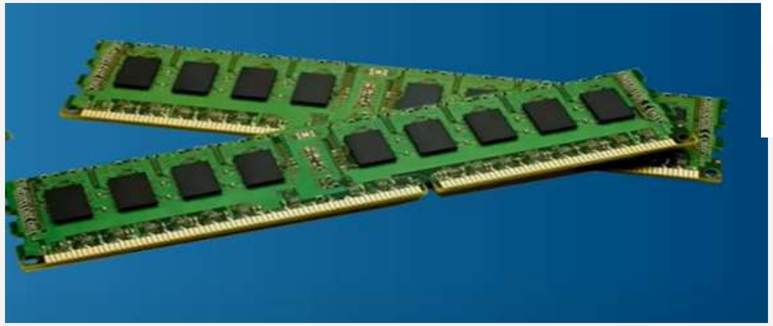
PRIMARY MEMORY

- memory storage that is directly accessible to the CPU
- primary area that offers quick access to the stored data
- volatile storage - does not permanently store the data
- CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset




Cache
Main Memory
Primary Memory

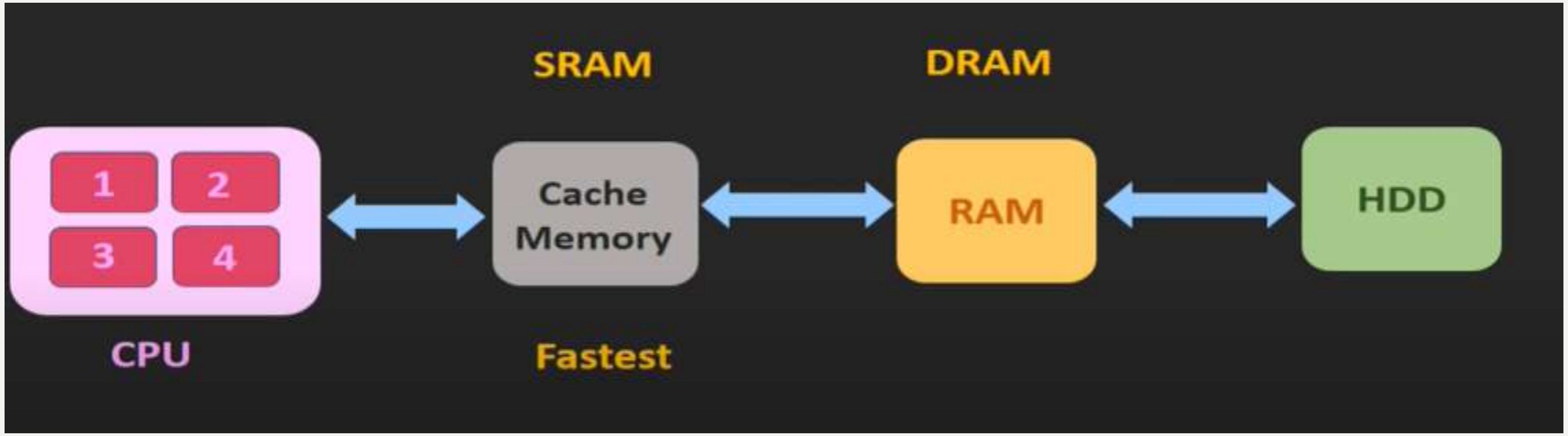
MAIN MEMORY



- one that is responsible for operating the data that is available by the storage medium
- handles each instruction of a computer machine
- can store gigabytes of data on a system but is small enough to carry the entire database
- Contents of main memory is lost if power failure or system crash occurs
- Storage is limited

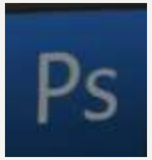
CACHE MEMORY

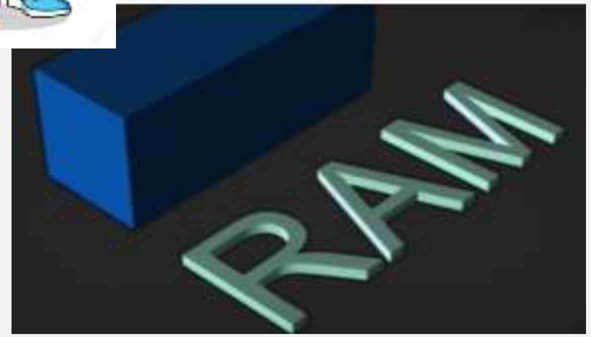
- one of the costly storage media and fastest one
- tiny storage media which is maintained by the computer hardware
- database implementors do pay attention to cache effects when designing query processing data structures and algorithms.




Ps
CPU

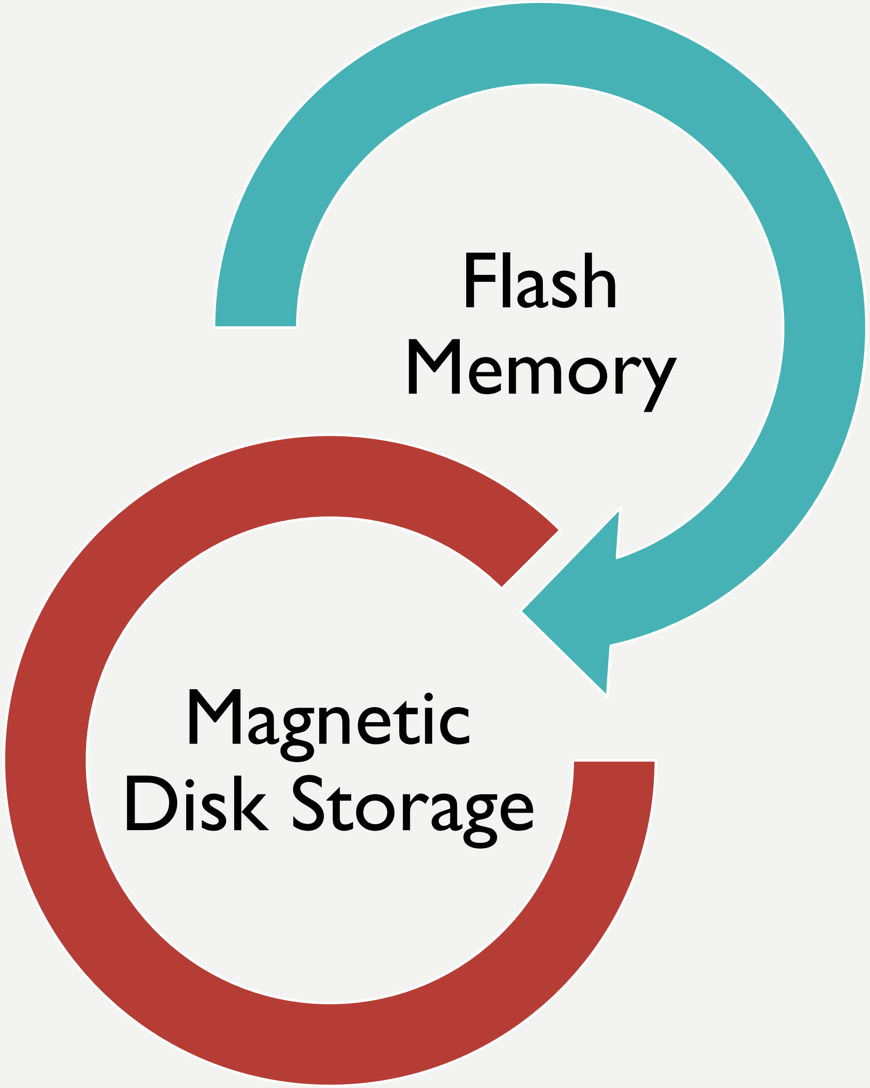

CACHE


Ps


RAM

SECONDARY MEMORY

- storage area that allows the user to save and store data permanently
- does not lose the data due to any power failure or system crash
- 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.


Flash Memory
Magnetic Disk Storage

FLASH MEMORY

- stores data in USB (Universal Serial Bus) keys which are further plugged into the USB slots of a computer system
- USB keys help transfer data to a computer system
- it is possible to get back the stored data which may be lost due to a power cut or other reasons
- high performance and is capable of storing large amounts of databases than the main memory

MAGNETIC DISK STORAGE

- online storage media
- used for storing the data for a long time
- *capable of storing an entire database*
- It is the responsibility of the computer system to make availability of the data from a disk to the main memory for further accessing
- *modified data should be written back to the disk*
- does not affect the data due to a system crash or failure, but a disk failure can easily ruin as well as destroy the stored data

TERTIARY MEMORY

- used to store huge volumes of data
- storage type that is external from the computer system
- slowest speed
- Offline storage
- generally used for data backup

Optical Storage

Tape Storage

OPTICAL STORAGE

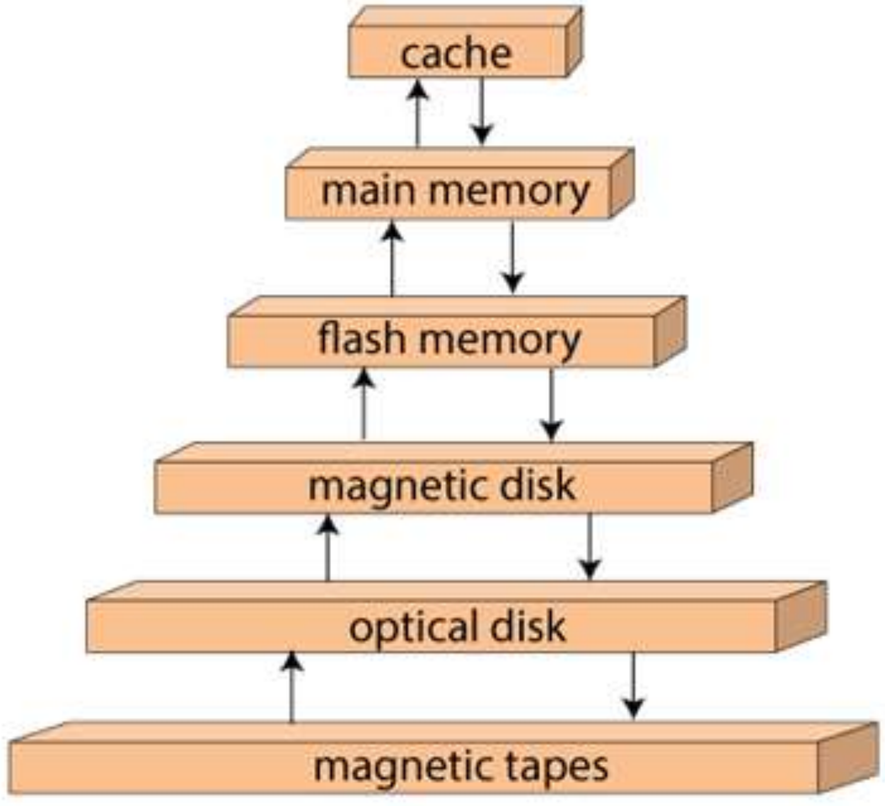
- can store megabytes or gigabytes of data
- Compact Disk (CD) and Digital Video Disk or a DVD are optical storage devices

TAPE STORAGE

- cheapest storage medium than disks
- tapes are used for archiving or backing up the data
- provides slow access to data as it accesses data sequentially from the start

High Speed

Expensive



Storage device hierarchy

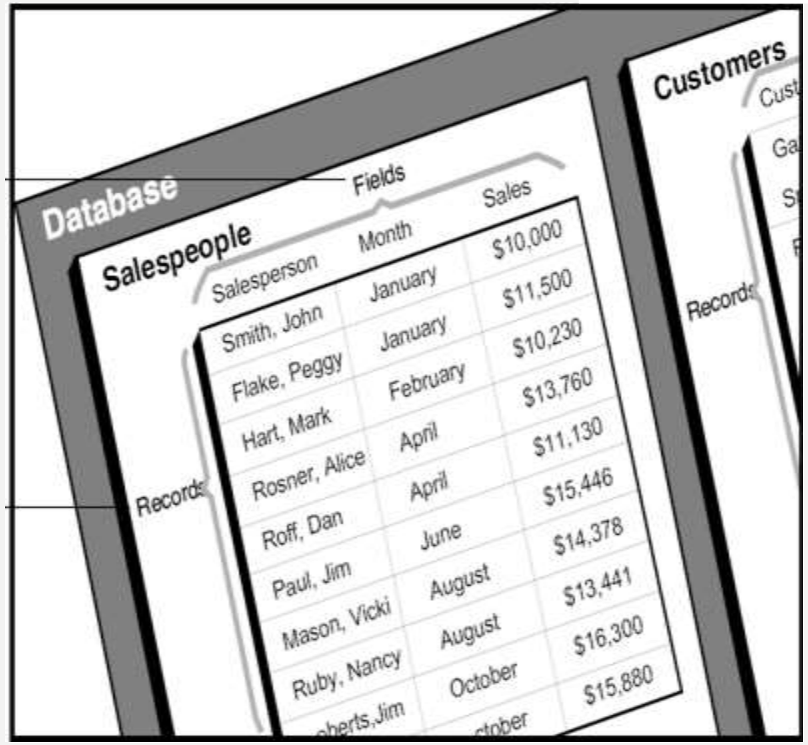
Low speed

Low cost

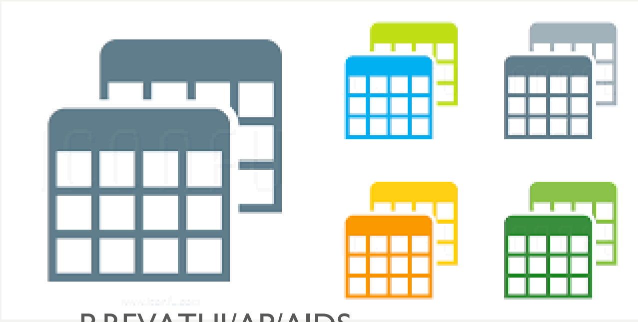
FILE ORGANIZATION

RDBMS

huge amount of data



Files in physical memory



FILE ORGANIZATION

- data is grouped within a table in RDBMS, and each table have related records
- user can see that the data is stored in form of tables, but in actual this huge amount of data is stored in physical memory in form of files

What is a File?

A file is named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tables and optical disks

FILE ORGANIZATION

- File Organization refers to the logical relationships among various records that constitute the file, particularly with respect to the means of identification and access to any specific record.
- refers to the way in which data is stored in a file and, consequently the methods by which it can be accessed
- In simple terms, Storing the files in certain order is called file Organization.
- Database is stored as a collection of files. Each file is a sequence of records. A record is a sequence of fields.

One file ⟷ One Table

One record ⟷ One Tuple

WHY TO ORGANIZE?

- in order to access the contents of the files – records in the physical memory, it is not that easy.
- They are not stored as tables there and our SQL queries will not work.
- We need some accessing methods.
- To access these files, we need to store them in certain order so that it will be easy to fetch the records

OBJECTIVE OF FILE ORGANIZATION

- 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.

FILE ORGANIZATION

- Each file is divided into **fixed-length storage units** known as **Blocks**. These blocks are the units of storage allocation as well as data transfer
- Most database use block size of 4 to 8 kilobytes
- Block may contain several records
- Each record is entirely contained in a single block to avoid partial storage of record in a block
- In RDBMS, the size of tuples varies in different relations. Thus, we need to structure our files in multiple lengths for implementing the records
- 2 ways
  - Fixed length records
  - Variable Length Records

FIXED LENGTH RECORDS

- setting a length and storing the records into the file
- If the record size exceeds the fixed size, it gets divided into more than one block

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

→ 53 bytes for each record

FIXED LENGTH RECORDS

- 2 problems
  - Unless the block size happens to be a multiple of 53 (which is unlikely), some records will cross block boundaries. That is, part of the record will be stored in one block and part in another. It would thus require two block accesses to read or write such a record.
  - It is difficult to delete a record from this structure. The space occupied by the record to be deleted must be filled with some other record of the file, or we must have a way of marking deleted records so that they can be ignored.

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

Solution

- allocate only as many records to a block as would fit entirely in the block
- When record is deleted, move next records to occupy space of deleted record

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

insertions tend to be more frequent than deletions

Leave the space of deleted record which can be used for next insertion

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

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

Need a Marker here to find the space to insert

File Header

At beginning of file, allocate certain number of bytes to file header

It stores the address of first empty space of the record

Like Pointers?

Yes

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

deleted records thus form a linked list

On insertion of a new record, we use the record pointed to by the header.

If no space is available, we add the new record to the end of the file.

VARIABLE LENGTH RECORDS

- records that vary in size
- requires the *creation of multiple blocks of multiple sizes to store them*
- 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.
  - Record types that allow repeating fields, such as arrays or multisets.
- 2 problems exists:
  - Defining the way of representing a single record so as to extract the individual attributes easily.
  - Defining the way of storing variable-length records within a block so as to extract that record in a block easily.

Problem

Variable length record

Initial Part:

with fixed length attr

1. numeric values,
2. Dates
3. fixed-length character attributes

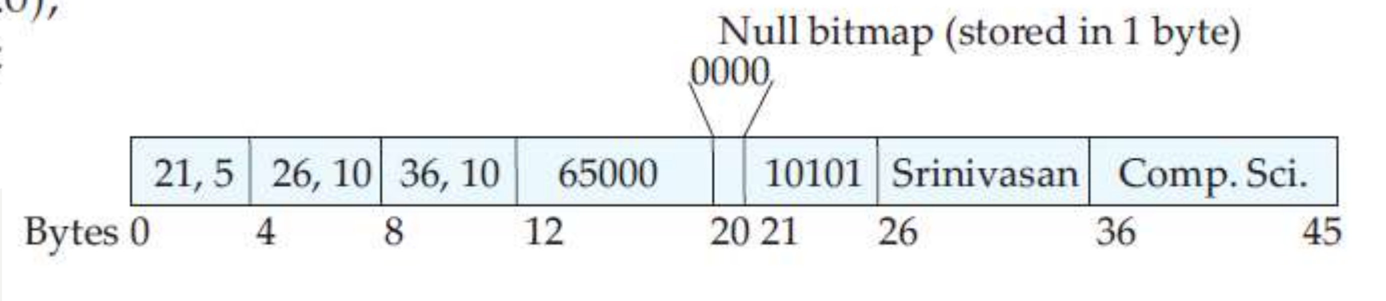
Followed by:

data for variable-length attribut

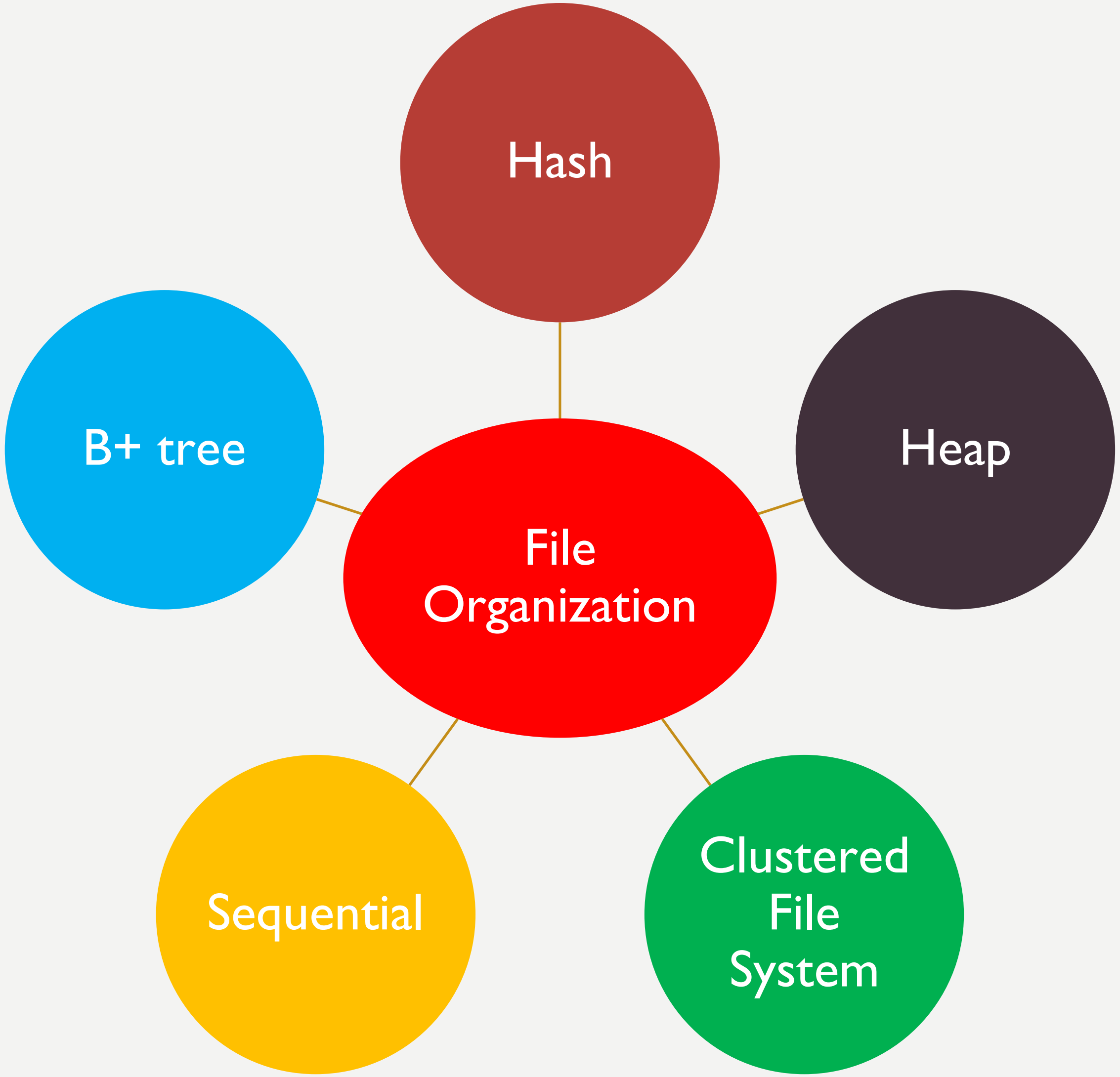
varchar type represented in the initial part of the record by (offset, length) pair

offset - place where that record begins
length - length of the variable-size attribute

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

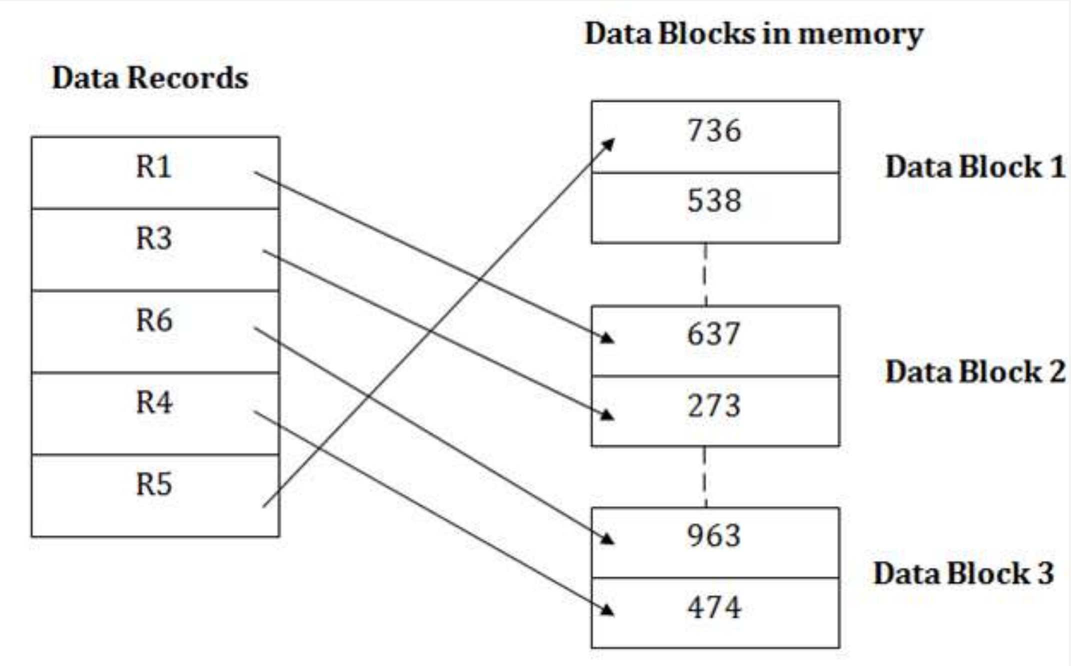


WAYS TO ORGANIZE THE FILES



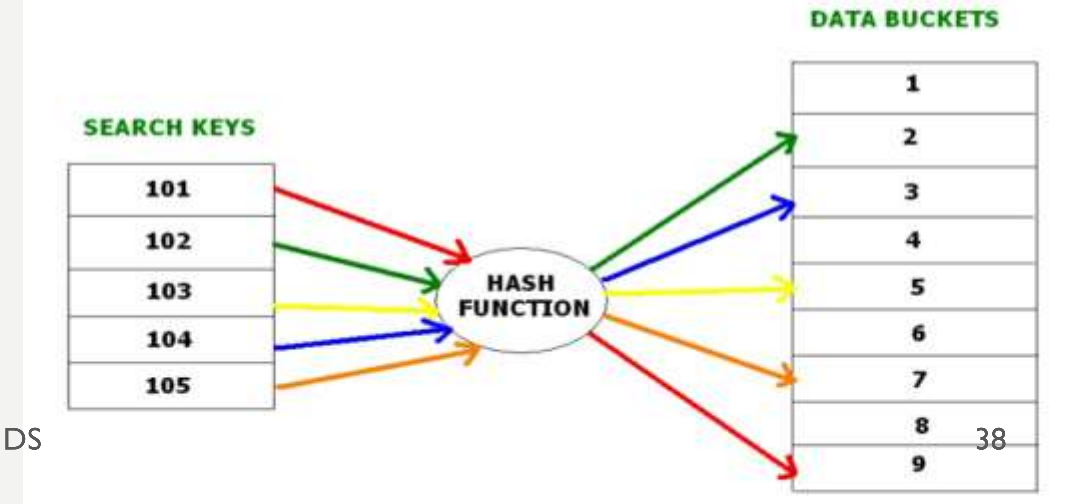
HEAP FILE ORGANIZATION

- ***Simplest file structure***
- Contains ***records in no particular order***
- ***Any record can be placed anywhere*** in the file where there is space for the record
- There is a single file for each relation.
- If a ***data block is full***, the *new record is stored in some other block*
- ***records are inserted at the file's end***



HASH

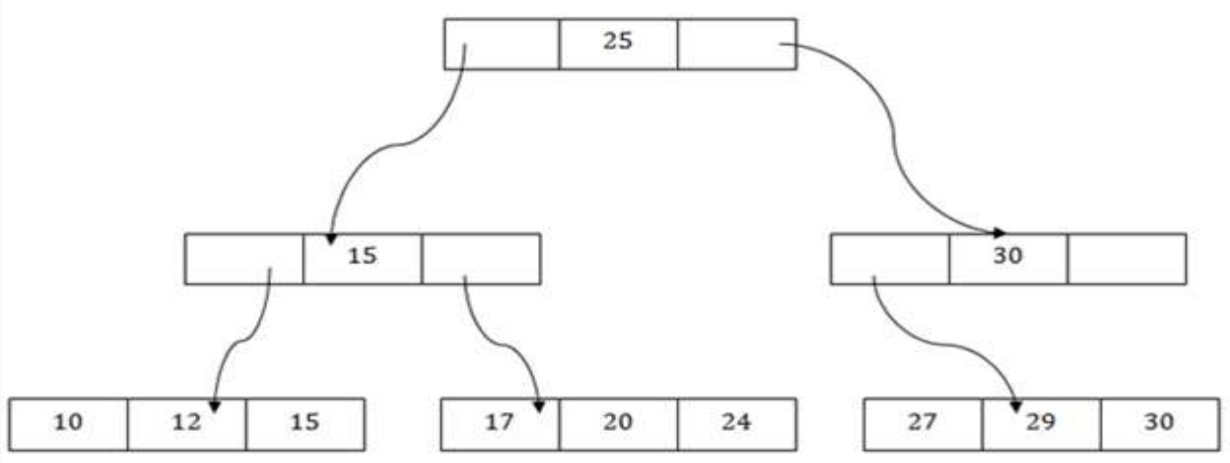
- uses the ***computation of hash function*** on some fields of the records
- *hash function's output* ***determines the location of disk block*** where the records are to be placed.
- *Hashing is a* ***technique to directly search the location of desired data on the disk without using index structure.***
- *used to* ***index and retrieve items*** *in a database as it is* ***faster to search*** *that specific item using the shorter hashed key instead of using its original value.*



B+ FILE ORGANIZATION

- uses a ***tree-like structure*** to store records in File.
- concept of ***key-index*** where the ***primary key is used to sort the records***.
- For each primary key, the ***value of the index is generated*** and ***mapped with the record***

- balanced binary search tree. It follows a multi-level index format.
- Leaf nodes denote actual data pointers
- B+ tree ensures that all leaf nodes remain at the same height.
- Leaf nodes are linked using a link list.
- A B+ tree can support random access as well as sequential access.



CLUSTER FILE

- When ***two or more records are stored in the same file***, it is known as clusters.
- These files will ***have two or more tables in the same data block***, and ***key attributes*** which are used to map these tables together are ***stored only once.***
- ***reduces the cost of searching*** for various records in different files
- used when there is a ***frequent need for joining the tables*** with the same condition

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEP_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

DEPARTMENT

DEP_ID	DEP_NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry

Cluster Key

DEP_ID	DEP_NAME	EMP_ID	EMP_NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

SEQUENTIAL

- ***Records are stored in sequential order***
- efficient processing of ***records in sorted order based*** on ***some search key***
- **search key** is any attribute or set of attributes; it need not be the primary key, or even a super key
- allows records to be read in sorted order

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

INSERTION IN SEQUENTIAL FILE

For insertion, we apply the following rules:

1. **Locate the record** in the file that comes before the record to be inserted in search-key order.
2. **If there is a free record** (that is, space left after a deletion) within the same block as this record, ***insert the new record there***.
3. Otherwise, insert the new record in an ***overflow block***. In either case, ***adjust the pointers*** so as to chain together the records in search-key order.

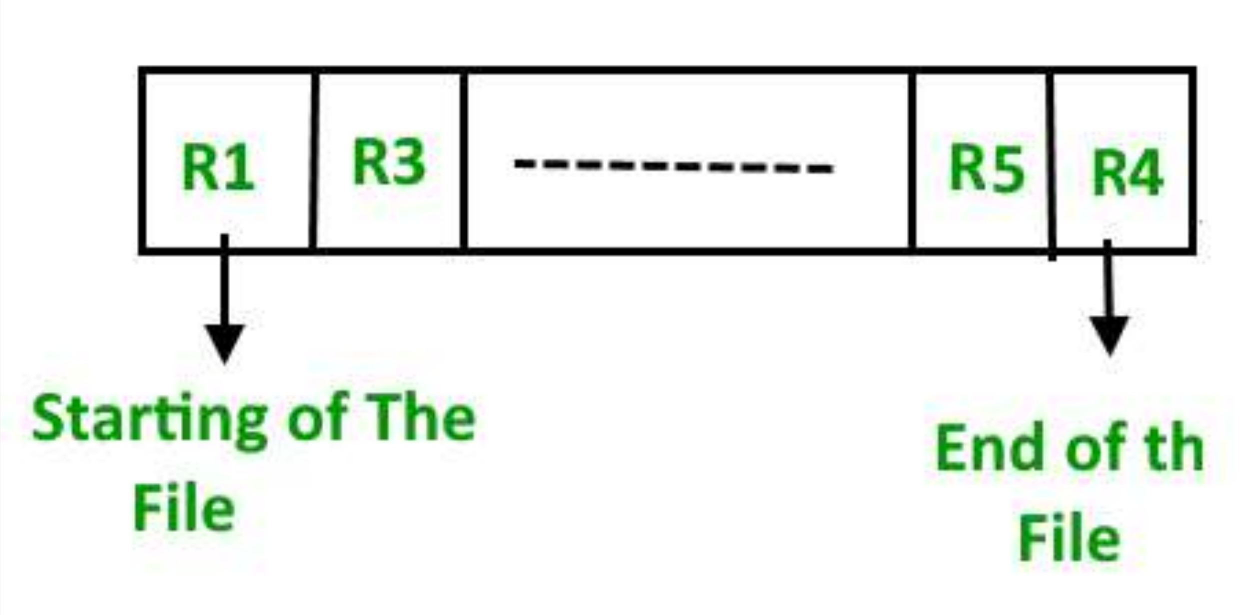
10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp.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

32222	Verdi	Music	48000	

2 WAYS

Pile File Method

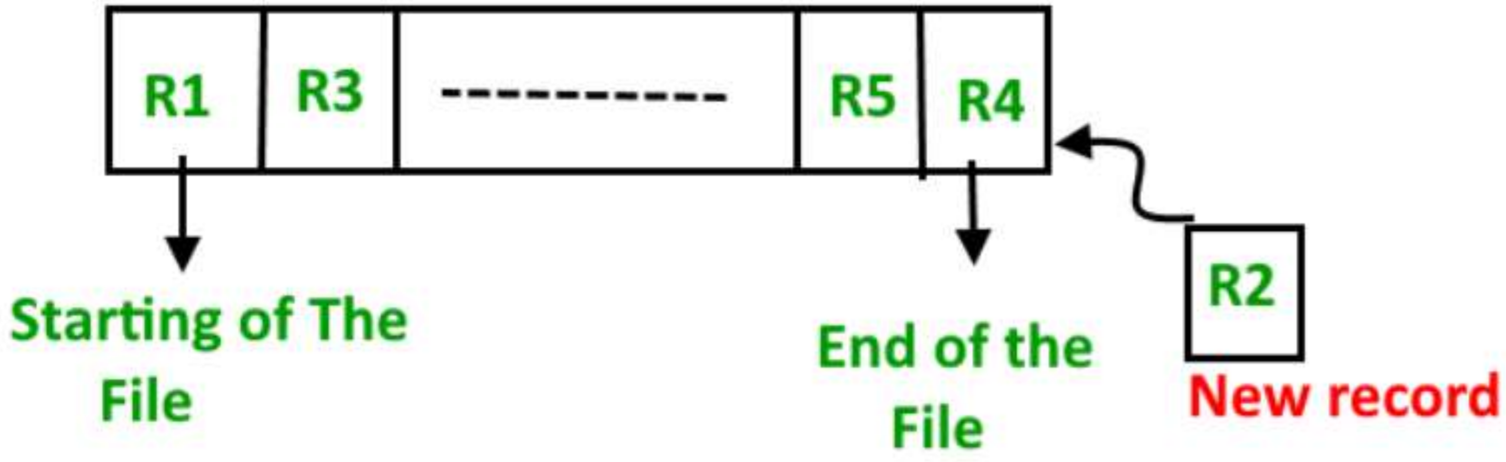
The records are stored in a sequence i.e one after other in the order in which they are inserted into the tables.



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.

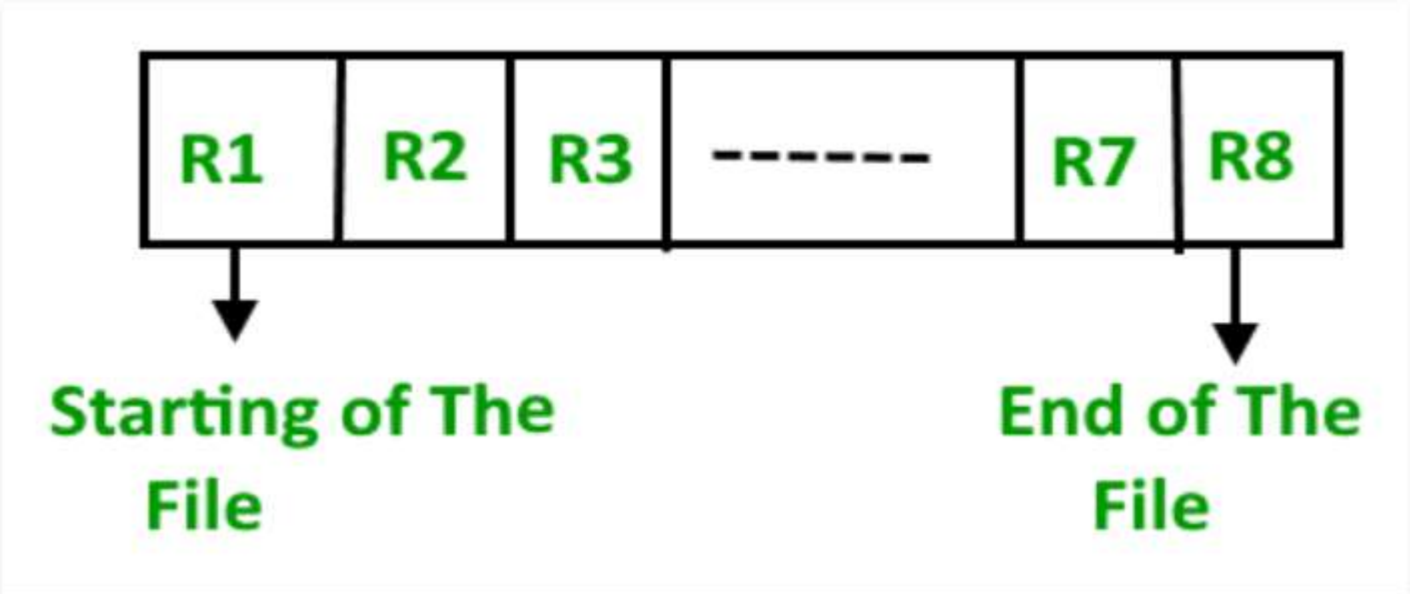


Sorted File Method

whenever a new record has to be inserted, it is always inser a sorted (ascending or descending) manner. Sorting of records may be based on any primary key or any other key.

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 .



Pros and Cons of Sequential File Organization

Pros –

1. **Fast and efficient** method for huge amount of data.
2. **Simple design**.
3. 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.

	Sequential	Heap/Direct	Hash	ISAM	B+ tree	Clustered
Method of storing	Stored as they come or sorted as they come	Stored at the end of the file. But the address in the memory is random.	Stored at the hash address generated	Address index is appended to the record	Stored in a tree like structure	Frequently joined tables are clubbed into one file based on cluster key
Types	Pile file and sorted file Method		Static and dynamic hashing	Dense, Sparse, multilevel indexing		Indexed and Hash
Design	Simple Design	Simplest	Medium	Complex	Complex	Simple
Storage Cost	Cheap (magnetic tapes)	Cheap	Medium	Costlier	Costlier	Medium

RAID

REDUNDANT ARRAY OF INDEPENDENT DIS

- way of storing the same data in different places on multiple hard disks or solid-state drives to protect data in the case of a drive failure
- connect multiple secondary storage devices for **increased performance,** data redundancy or both
- gives you the ability to **survive one or more drive failure** depending upon the RAID level used
- consists of an array of disks in which **multiple disks are connected** to achieve different goals

- Redundancy Array of the Independent Disk
- technology which is used to connect multiple secondary storage devices for increased performance, data redundancy or both.
- gives the ability to survive one or more drive failure depending upon the RAID level used.
- It consists of an array of disks in which multiple disks are connected to achieve different goals
- RAID 0, RAID 1, RAID 2,RAID 3,RAID 4,RAID 5, RAID 6

- It contains a set of physical disk drives.
- In this technology, the operating system views these separate disks as a single logical disk.
- In this technology, data is distributed across the physical drives of the array.
- Redundancy disk capacity is used to store parity information.
- In case of disk failure, the parity information can be helped to recover the data.

WHY REDUNDANCY?

- although taking up extra space, adds to disk reliability
- in case of disk failure, if the same data is also backed up onto another disk, we can retrieve the data and go on with the operation
- if the data is spread across just multiple disks without the RAID technique, the loss of a single disk can affect the entire data.

MIRRORING

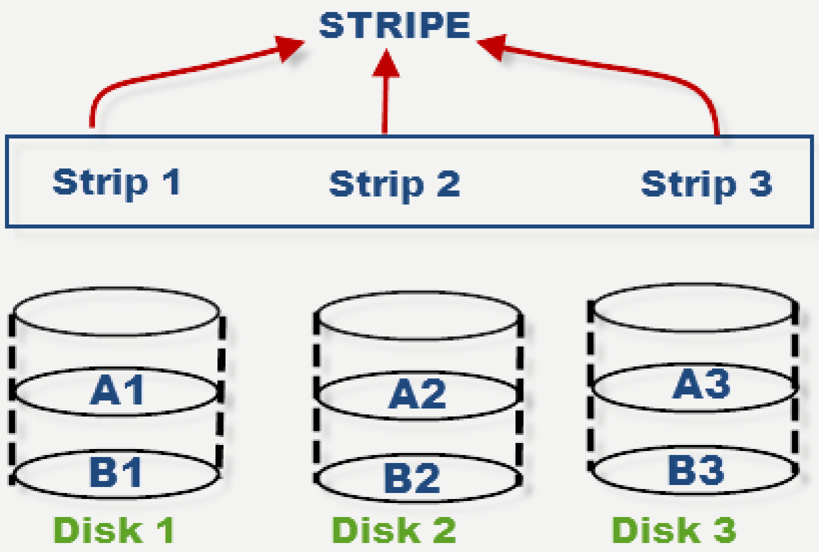
- approach to **introduce redundancy** is to duplicate every disk . This is called mirroring
- A logical disk then consists of two physical disks, and every write is carried out on both disks.
  If one of the disks fails, the data can be read from the other.
- Data will be lost only if the second disk fails before the first failed disk is repaired

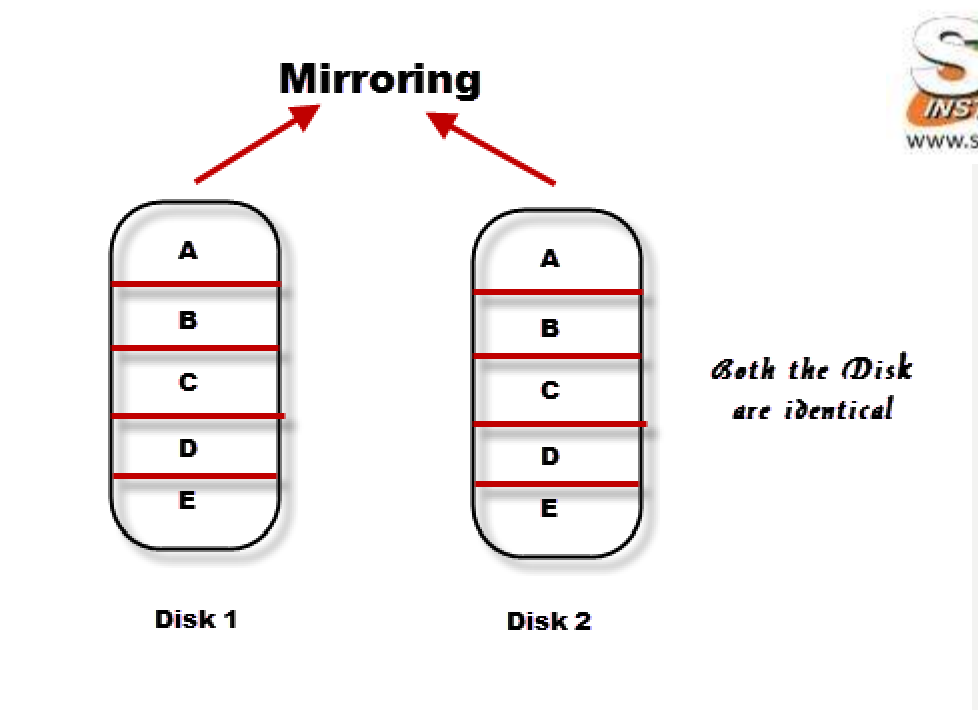
IMPROVEMENT IN PERFORMANCE VIA PARALLELISM

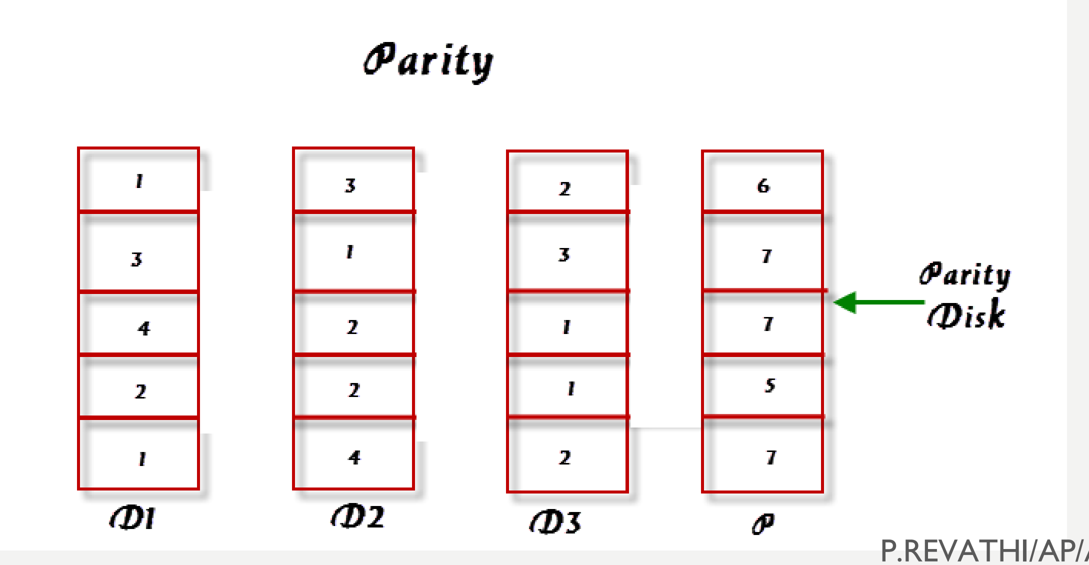
- with **Disk Mirroring** - rate at which read requests can be handled is doubled, since read requests can be sent to either disk
- we can improve the transfer rate as well (or instead) by **striping** data across multiple disks
- data striping consists of splitting the bits of each byte across multiple disks; such striping is called bit level striping.
- For e.g.,
  - if we have an array of eight disks, we write bit i of each byte to disk I
  - array of eight disks can be treated as a single disk - eight times the normal size – eight times the transfer rate

IMPROVEMENT IN PERFORMANCE VIA PARALLELISM

- Block-level striping stripes blocks across multiple disks
- treats the array of disks as a single large disk, and it gives blocks logical numbers
- array of n disks, block-level striping assigns logical block i of the disk array to disk (i mod n) + 1


STRIPE
Strip 1
Strip 2
Strip 3
A1
B1
A2
B2
A3
B3
Disk 1
Disk 2
Disk 3


Mirroring
A
B
C
D
E
A
B
C
D
E
Both the Disk are identical
Disk 1
Disk 2


Parity
1
3
4
2
1
3
1
2
2
4
2
3
1
1
2
6
7
7
5
7
Parity Disk
D1
D2
D3
P

RAID LEVELS

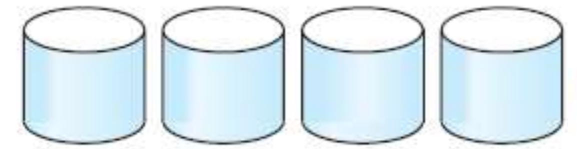
high reliability | **Mirroring** | expensive

Does not improve high reliability | **Striping** | high data-transfer rates

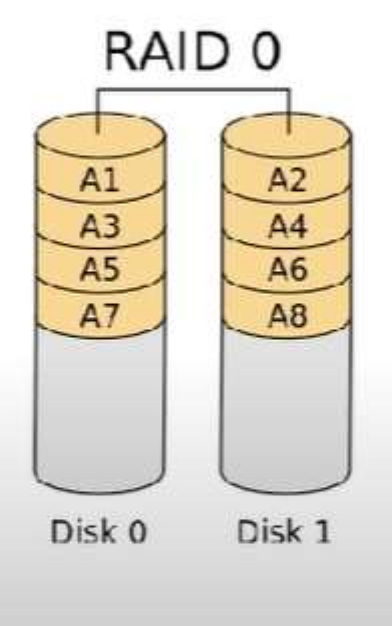
1. Provides redundancy
2. Lower cost
3. Disk striping with "parity" bits

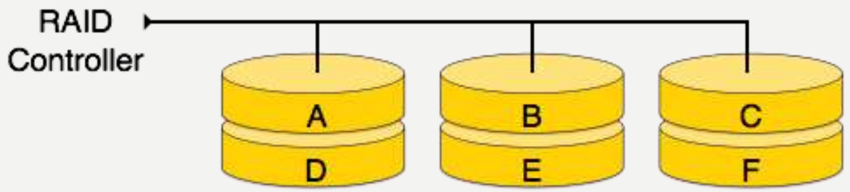
→ RAID Levels

RAID LEVELS – RAID 0

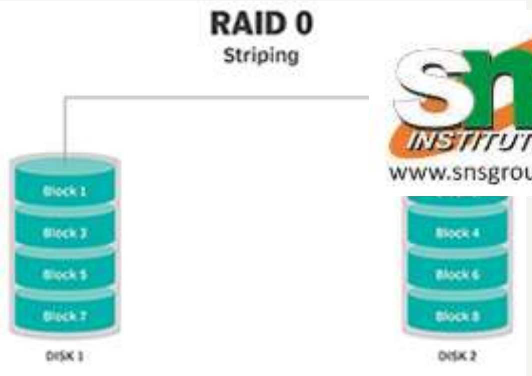

(a) RAID 0: nonredundant striping

- provides ***data stripping*** i.e., a ***data can place across multiple disks***
- if **one disk fails then all data in the array is lost**.
- 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***
- **Doesn't provide fault tolerance** but **increases the system performance**





RAID LEVELS – RAID 0



Disk 0	Disk 1	Disk 2	Disk 3
20	21	22	23
24	25	26	27
28	29	30	31
32	33	34	35

instead of placing just one block into a disk at a time, we can work with two or more blocks placed it into a disk before moving on to the next one

there is **no duplication of data**. Hence, **a block once lost cannot be recovered**.

Pros of RAID 0:

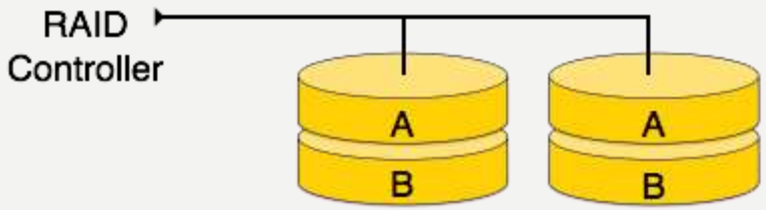
- In this level, throughput is increased because multiple data requests probably not on the same disk.
- This level full utilizes the disk space and provides high performance.
- It requires minimum 2 drives.

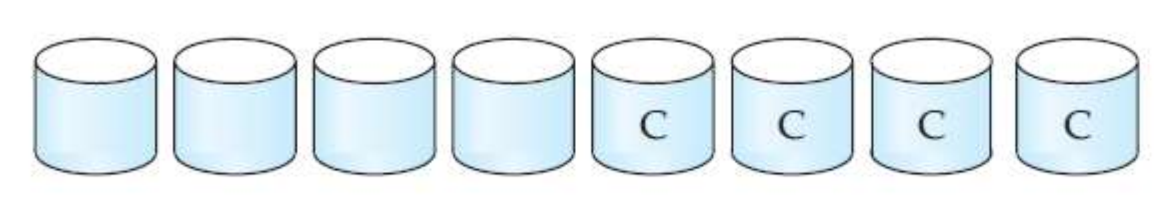
Cons of RAID 0:

- It doesn't contain any error detection mechanism.
- The RAID 0 is not a true RAID because it is not fault-tolerance.
- In this level, failure of either disk results in complete data loss in respective array.

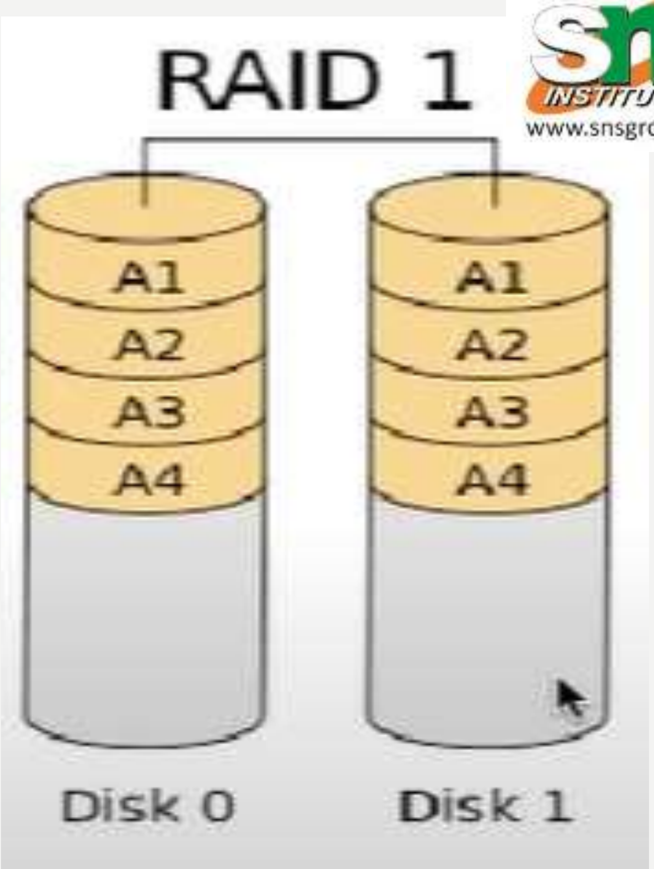
RAID LEVELS – RAID 1

- This level is called ***mirroring*** of data
- **copies the data from drive 1 to drive 2**
- It provides **100% redundancy** in case of a failure





(b) RAID 1: mirrored disks



RAID LEVELS – RAID 1

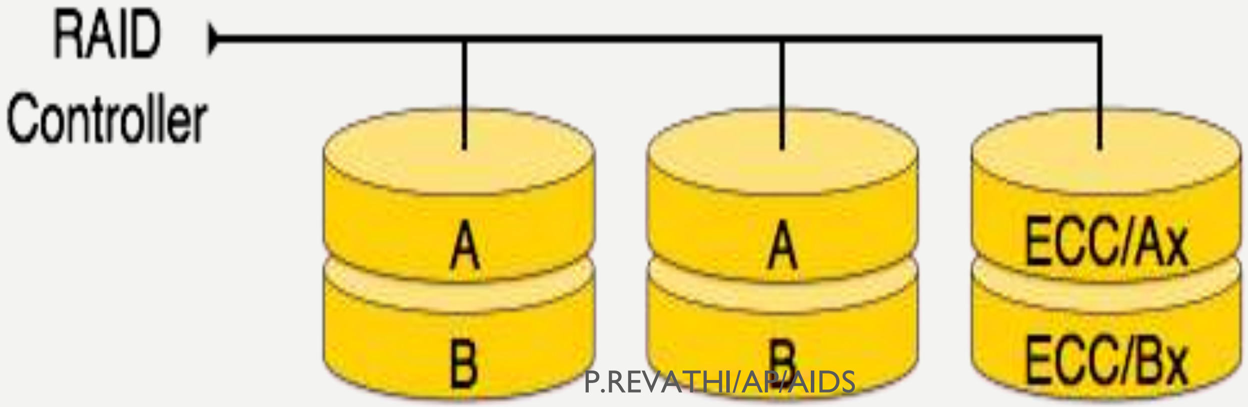
Disk 0	Disk 1	Disk 2	Disk 3
A	A	B	B
C	C	D	D
E	E	F	F
G	G	H	H

Only half space of the drive is used to store the data.

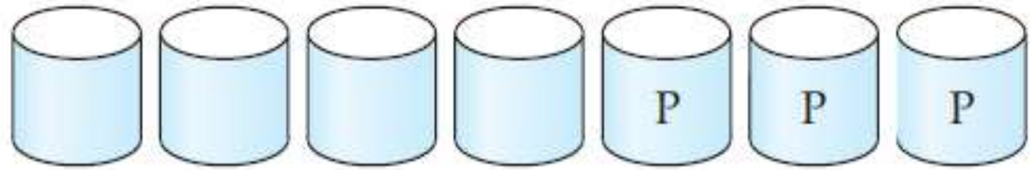
The other half of drive is just a mirror to the already stored data.

RAID LEVELS – RAID 2

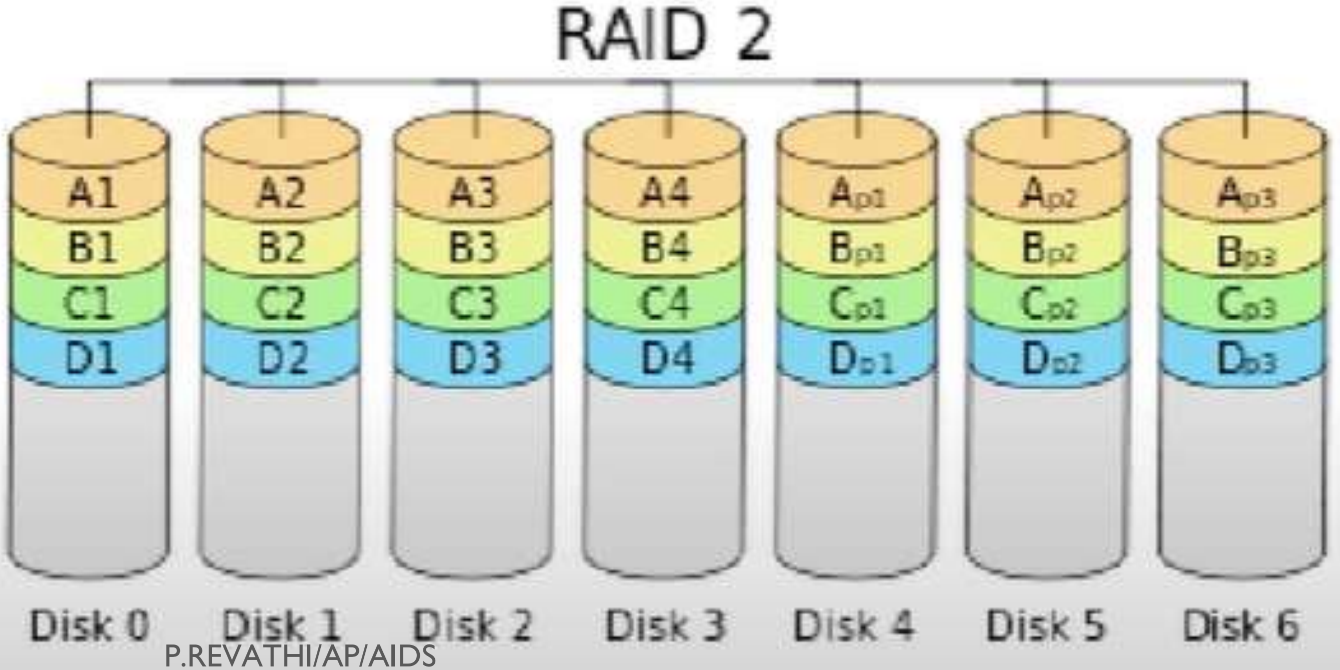
- RAID 2 **records Error Correction Code using Hamming distance** for its data, striped on different disks
- employs **parity bits**
- Each byte in a memory system may have a parity bit associated with it that records whether the numbers of bits in the byte that are set to 1 is even (parity = 0) or odd (parity = 1)
- If one of the bits in the byte gets damaged (either a 1 becomes a 0, or a 0 becomes a 1), the parity of the byte changes and thus will not match the stored parity



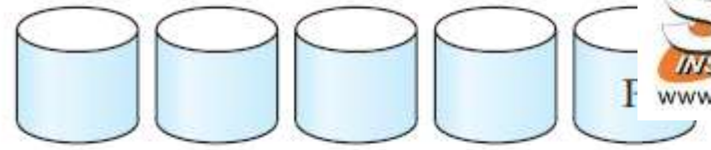
RAID LEVELS – RAID 2



(c) RAID 2: memory-style error-correcting codes

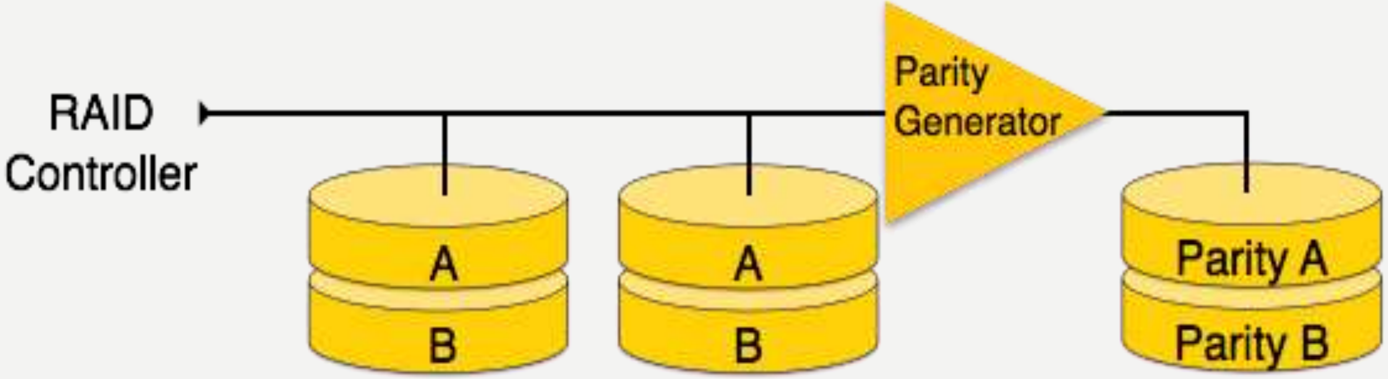


RAID LEVELS – RAID 3



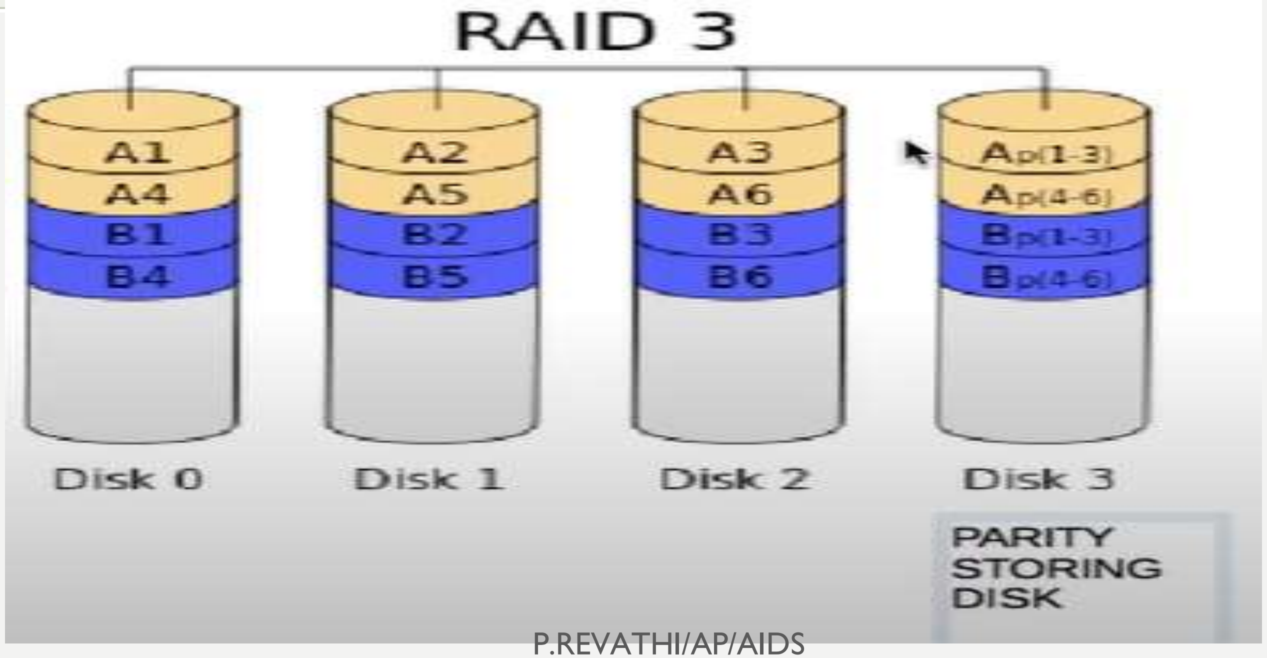
(d) RAID 3: bit-interleaved parity

- RAID 3 **stripes the data onto multiple disks**
- The parity bit generated for data word is stored on a different disk
- **In case of drive failure**, **the parity drive is accessed**, and data is reconstructed from the remaining devices.
- Once the failed drive is replaced, the missing data can be restored on the new drive.



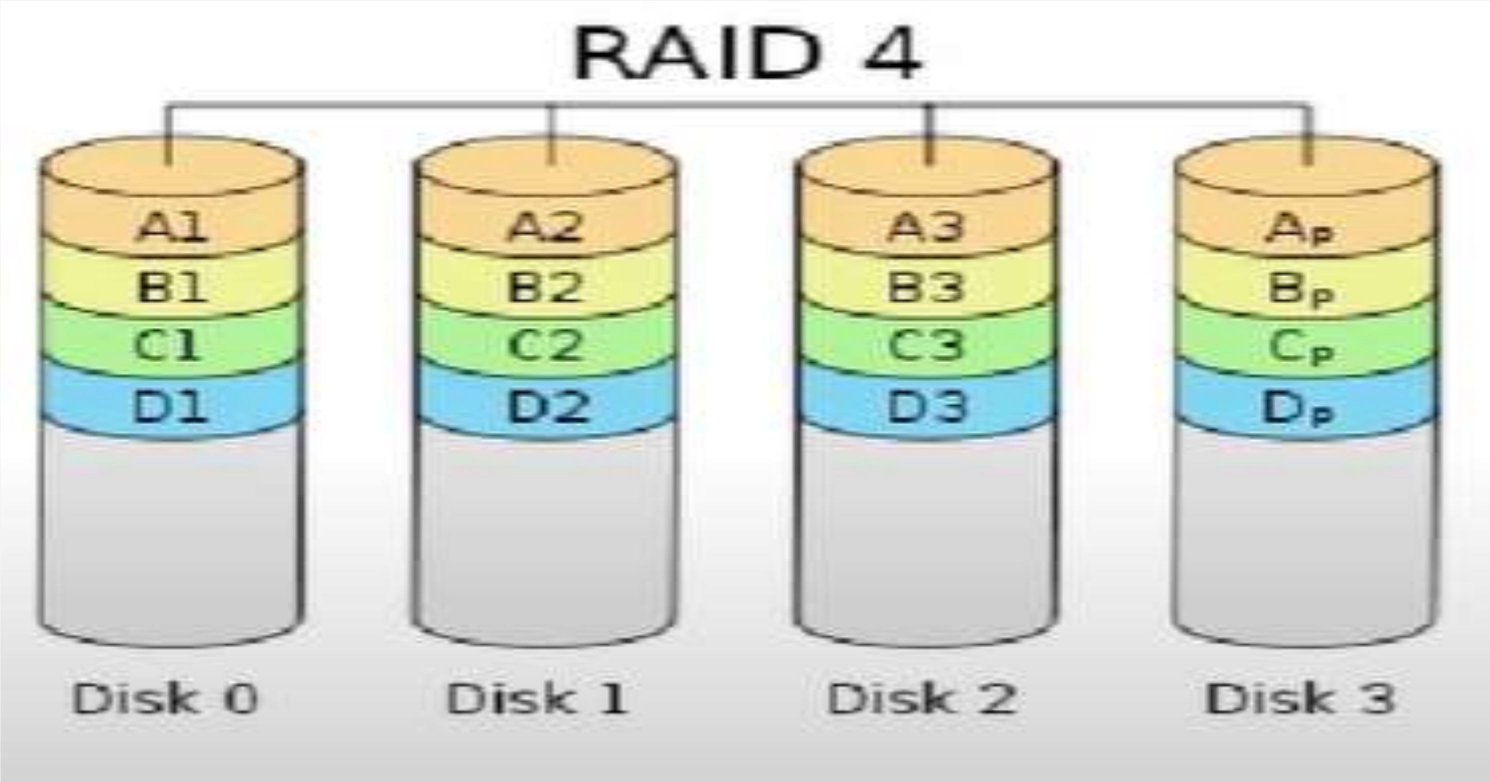
RAID LEVELS – RAID 3

Disk 0	Disk 1	Disk 2	Disk 3
A	B	C	P(A, B, C)
D	E	F	P(D, E, F)
G	H	I	P(G, H, I)
J	K	L	P(J, K, L)



RAID LEVELS – RAID 4

- RAID 4 consists of **block-level stripping with a parity disk**
- This level **allows recovery of at most 1 disk failure due to the way parity works.**
- In this level, if more than one disk fails, then there is no way to recover the data



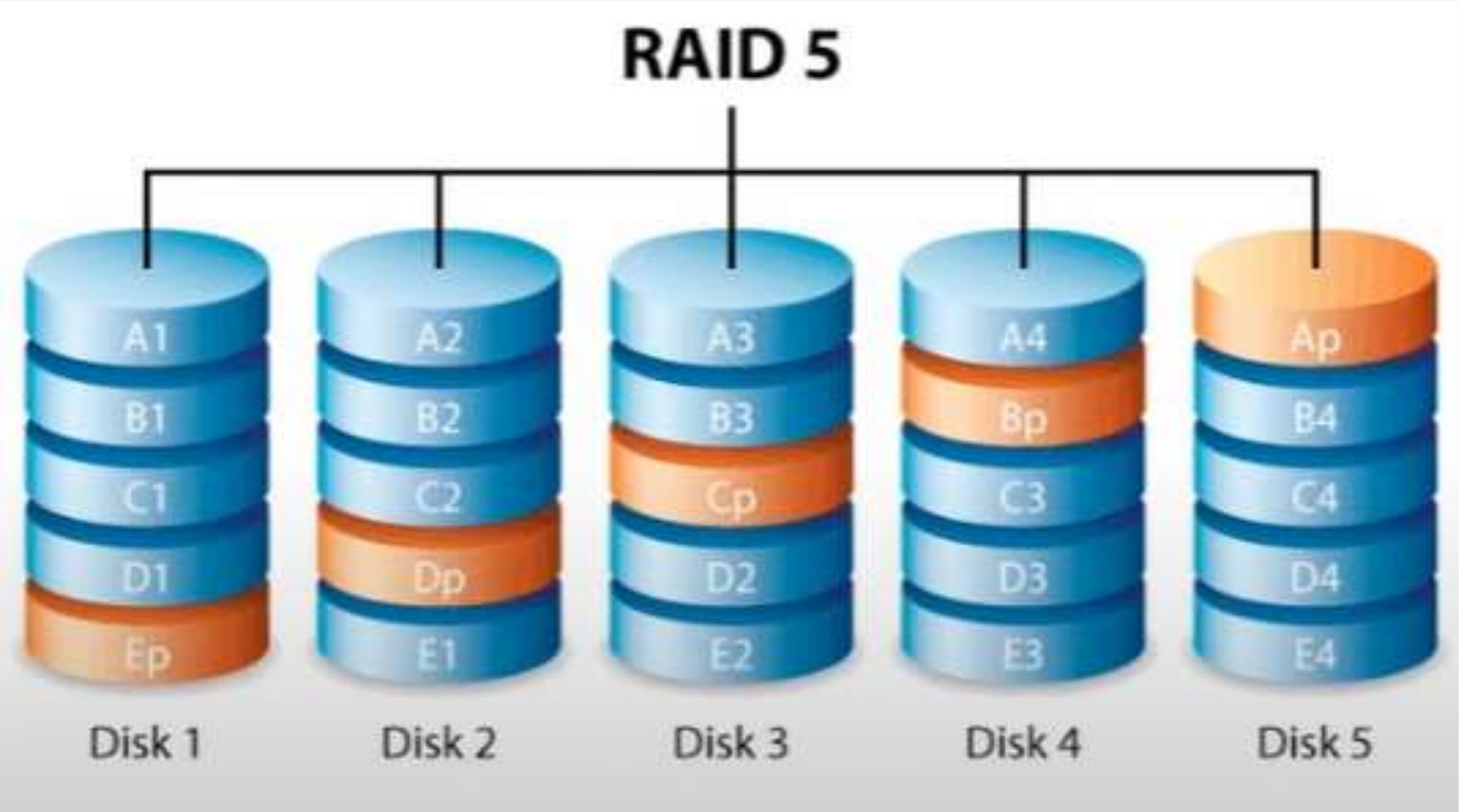
RAID LEVELS – RAID 5

- RAID 5 is a slight modification of the RAID 4 system.
- The only difference is that in RAID 5, the **parity rotates among the drives**
- It **consists of block-level striping with DISTRIBUTED parity**

Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

RAID LEVELS – 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 LEVELS – 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**