

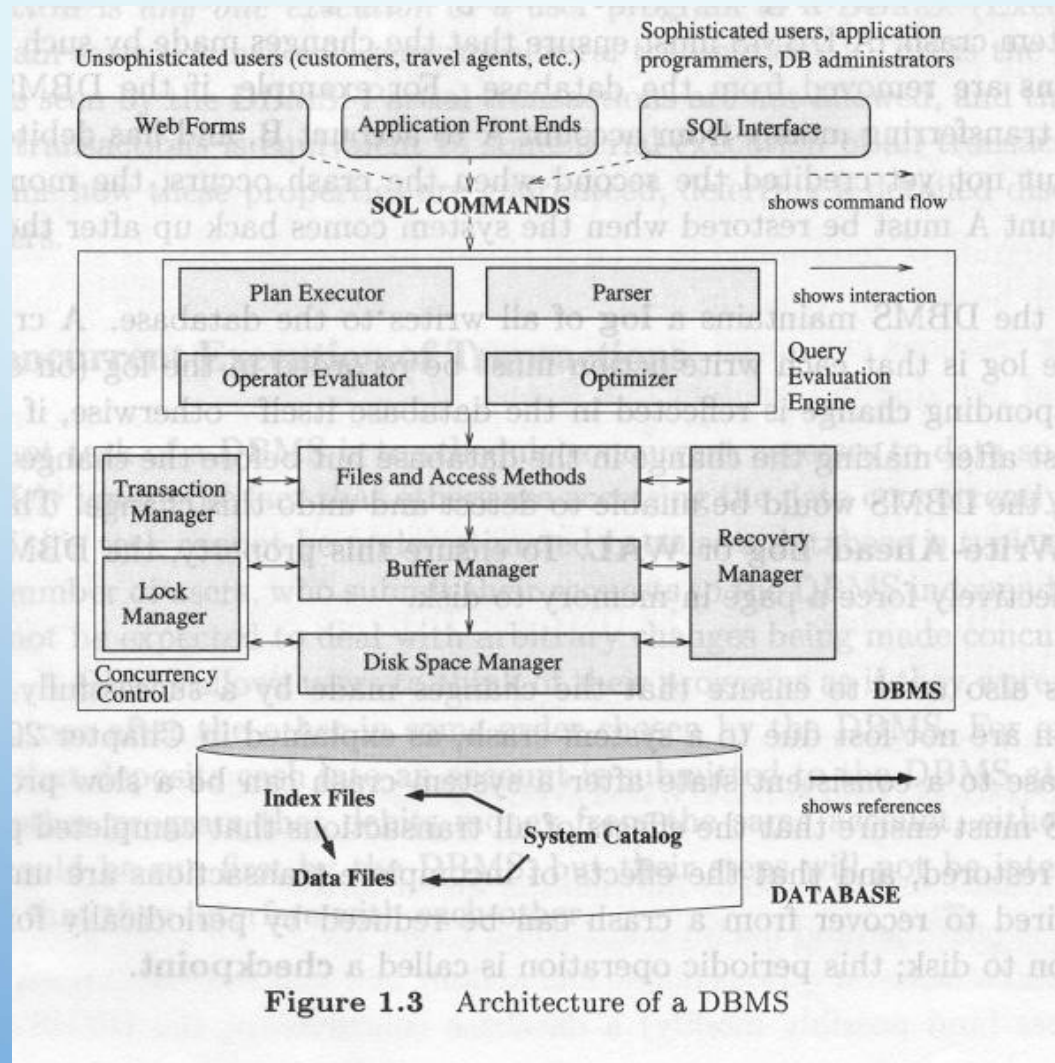
The background of the slide features a light blue to medium blue gradient. Scattered across this background are numerous water droplets of various sizes. Some droplets are large and prominent, showing highlights and shadows that give them a three-dimensional appearance. Others are smaller and more numerous, adding texture to the overall design.

# **COMPONENTS OF A DBMS, DISKS AND FILES**

# THIS LECTURE...

- COMPONENTS OF A DBMS...
- DATA STORAGE: DISKS AND FILES...

# STRUCTURE OF A DBMS



# STRUCTURE OF A DBMS

- DBMSs accept SQL statements
- The queries are **parsed** and presented to the **optimizer** (for query optimization)
- An **execution plan** is produced after optimization
- **Execution plan** consists of a set of **relational operators** and extra information

# STRUCTURE OF A DBMS

- All data is stored in logical storage called **files**
- **Files and Access Methods** provide an interface to manipulate files
- **Files** are considered to be a set of **pages**
- **Buffer manager** brings pages from disk to main memory

# STRUCTURE OF A DBMS

- **Disk Space Manger** deals with the management of space on disk.
- DBMS supports concurrency control and crash recovery
- This requires users scheduling the transactions carefully and maintaining a log of all changes to the database

# STRUCTURE OF A DBMS

- DBMS components for transactions processing include
  - **Transaction Manager:** ensures that transactions requests/releases locks according to a appropriate locking protocol
  - **Lock Manager:** Keeps track of locks for each database object
  - **Recovery Manager:** Maintains the log and restores the system after a crash

# DATA STORAGE: DISKS AND FILES

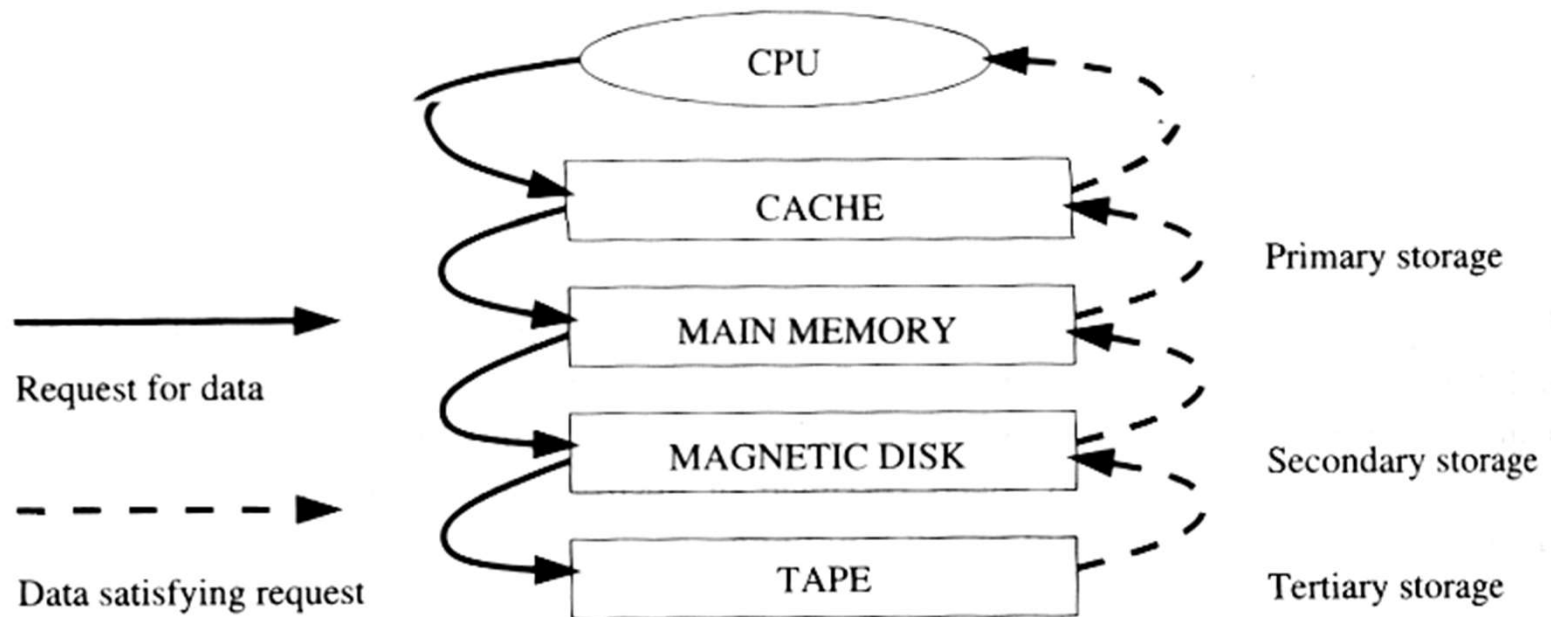
- DBMS stores information on (“hard”) disks.
- This has major implications for DBMS design!
  - **READ**: transfer data from disk to main memory (RAM).
  - **WRITE**: transfer data from RAM to disk.
  - Both are high-cost operations, relative to in-memory operations, so must be planned carefully!



# WHY NOT STORE EVERYTHING IN MAIN MEMORY?



- *Costs too much.* Rs. 9000 will buy you either 256MB of RAM or 40GB of disk.
- *Main memory is volatile.* We want data to be saved between runs.  
(Obviously!)
- Typical storage hierarchy:
  - Main memory (RAM) for currently used data (primary storage).
  - Disk for the main database (secondary storage).
  - Tapes for archiving older versions of the data (tertiary storage).

# MEMORY HIERARCHY



**Figure 7.1** The Memory Hierarchy

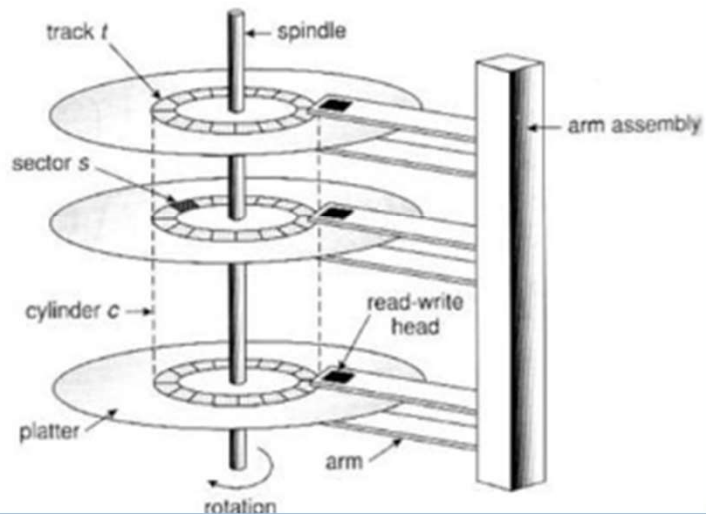
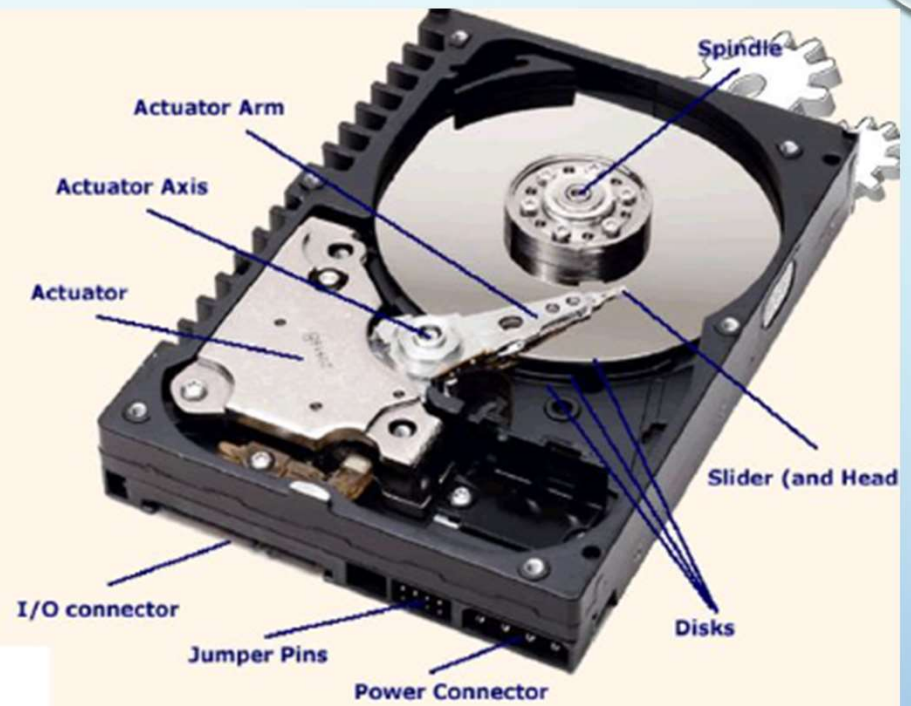
# CHARACTERISTICS OF STORAGE MEDIUM

	Cost	Speed
<b>Primary Storage</b> (main memory, cache, etc.)	 Increase	 Decrease
<b>Secondary Storage</b> (magnetic disks, optical disks, etc.)		
<b>Tertiary Storage</b> (tapes)		

# DISKS

- Secondary storage device of choice.
- Main advantage over tapes: *random access* vs. *sequential*.
- Data is stored and retrieved in units called *disk blocks* or *pages*.
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
  - Therefore, relative placement of pages on disk has major impact on DBMS performance!

# *Disks*



# COMPONENTS OF A DISK

- The platters spin (say, 90rps).

**Track:** concentric rings on platters

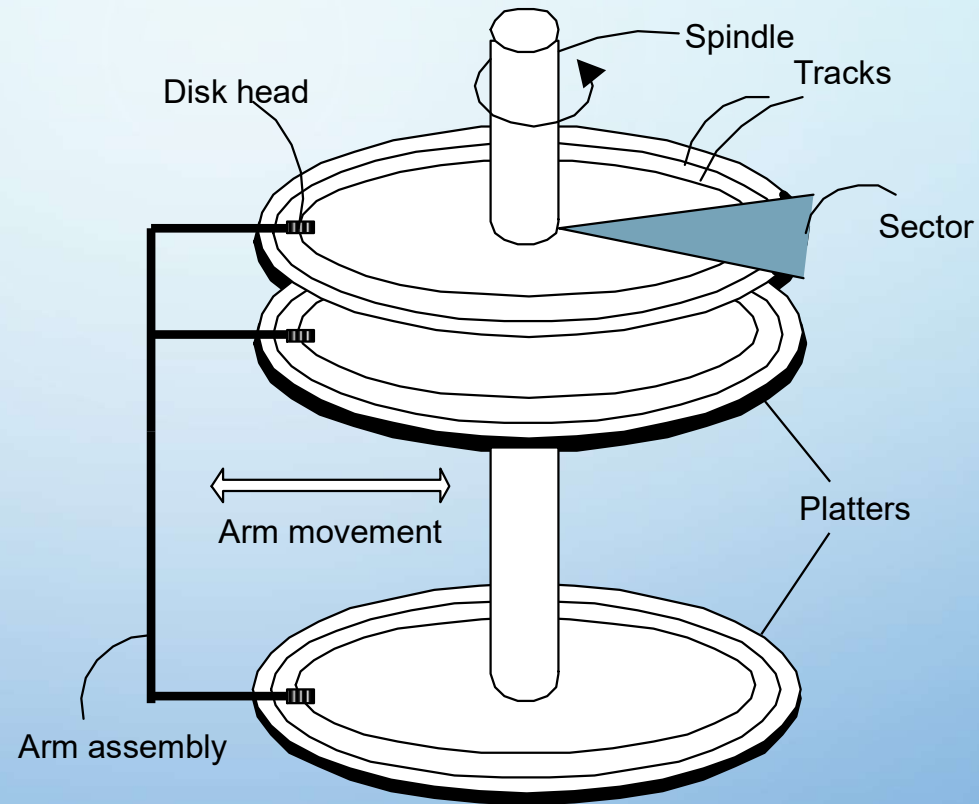
**Cylinder:** the set of all tracks with the same diameter

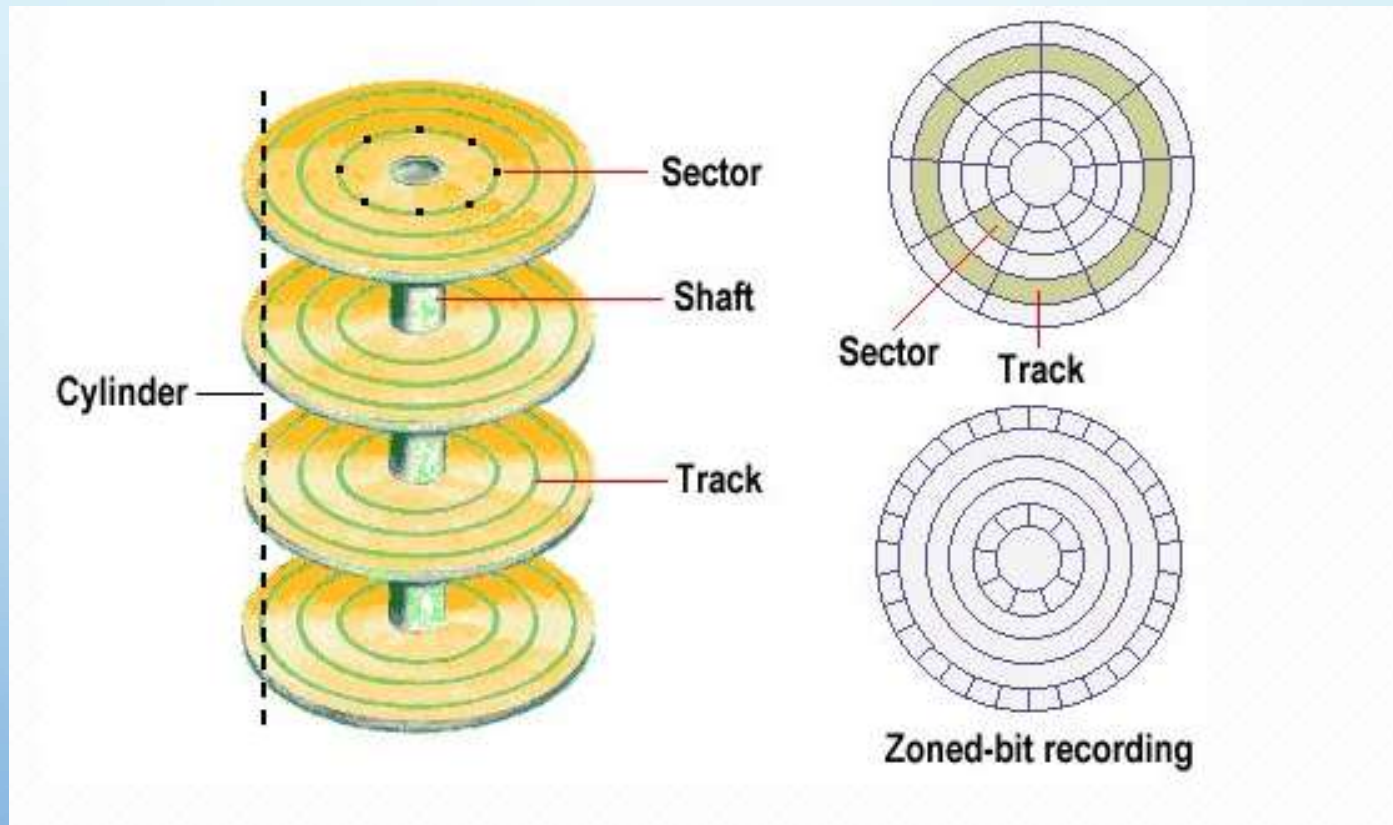
**Sector:** each track is divided into fixed-sized arcs, called sectors

**Block:** the unit in which data is written and read from disk; block size is a multiple of sector size.

**Disk heads:** move as a unit; only one head read/write at any one time.

**Arm assembly:** move in or out to position a disk head on a desired track.



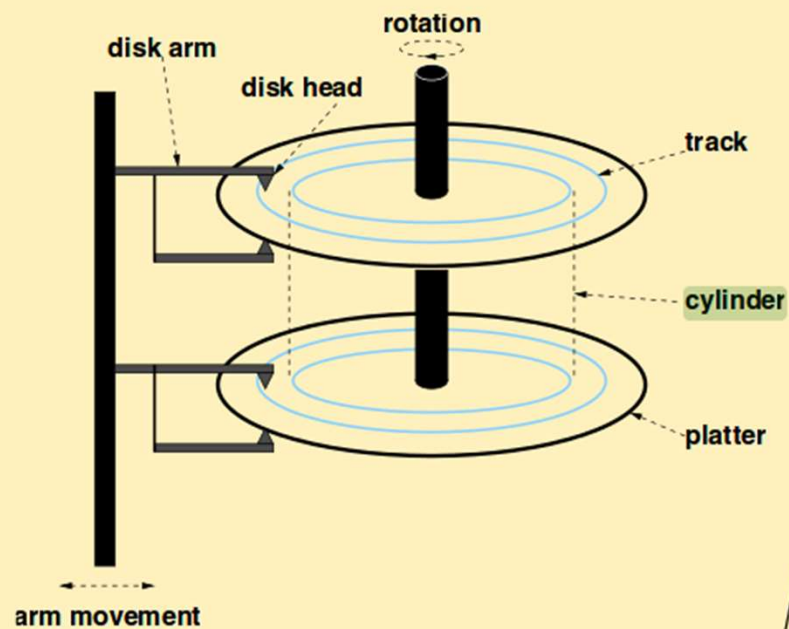




## PERMANENT MEMORY: DISK

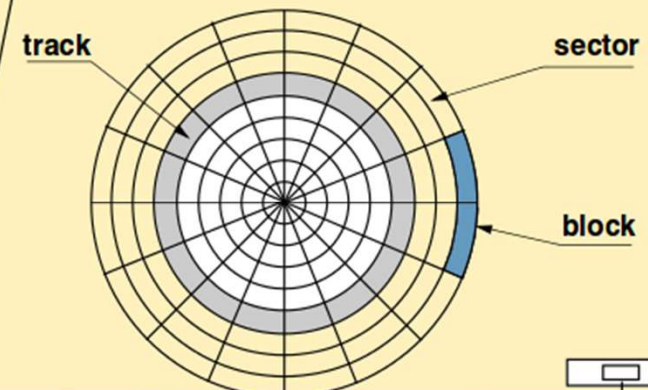
8

In an electronic world, disks are a mechanical/anachronism!



**Access Time =**

Seek Time (0-10 ms) +  
Rotational Time (0-3 ms) +  
Transfer time (.02 ms per 8K)





## COMPONENTS OF A DISK

- A **disk controller** is an interface which controls the disk drive.
- This interface commands to read/write sectors by moving the arm assembly and transferring data to/from the disk surfaces
- A well-known interface for PCs is SCSI (Small Computer Storage Interface)

## ACCESSING A DISK PAGE

- The unit for data transfer between disk and main memory is a block;
  - ✓ if a single item on a block is needed, the entire block is transferred.
- Reading or writing a disk block is called an I/O (for input/output) operation.

## ACCESSING A DISK PAGE

- *Seek time* (moving arms to position disk head on track): 1-20 ms
- *Rotational delay* (waiting for block to rotate under head) 0-10ms  
usually less than seek time
- *Transfer time* (actually moving data to/from disk surface) 1 ms per 4KB block

## ACCESSING A DISK PAGE

*access time = seek time + rotational delay + transfer time*

# DISK PERFORMANCE

- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20msec
  - Rotational delay varies from 0 to 10msec
  - Transfer rate is about 1msec per 4KB page
- Key to lower I/O cost: **reduce seek/rotation delays!** Hardware vs. software solutions?

# NEXT BLOCK

- The "Next Block" is related to how data is organized on a disk to optimize access speed, particularly in the context of minimizing seek time and rotational delay.

## **Order of Block Arrangement**

1. Blocks on the Same Track
2. Blocks on the Same Cylinder
3. Blocks on Adjacent Cylinders

# NEXT BLOCK

- **Blocks on the Same Track:** The first priority is to store consecutive blocks on the same track. This minimizes both seek time (the time it takes for the read/write head to move to a different track) and rotational delay (the time it takes for the desired block to rotate under the read/write head).
- **Blocks on the Same Cylinder:** If the track fills up, the next best place to store the following blocks is on another track within the same cylinder. This way, the read/write head only needs to move vertically to the next track on a different platter, which is faster than moving to a completely different cylinder.
- **Blocks on Adjacent Cylinders:** If the entire cylinder is full, the next blocks should be placed on the tracks of an adjacent cylinder. This still minimizes the seek time since the head only needs to move slightly to the neighboring cylinder.

# DISK PERFORMANCE

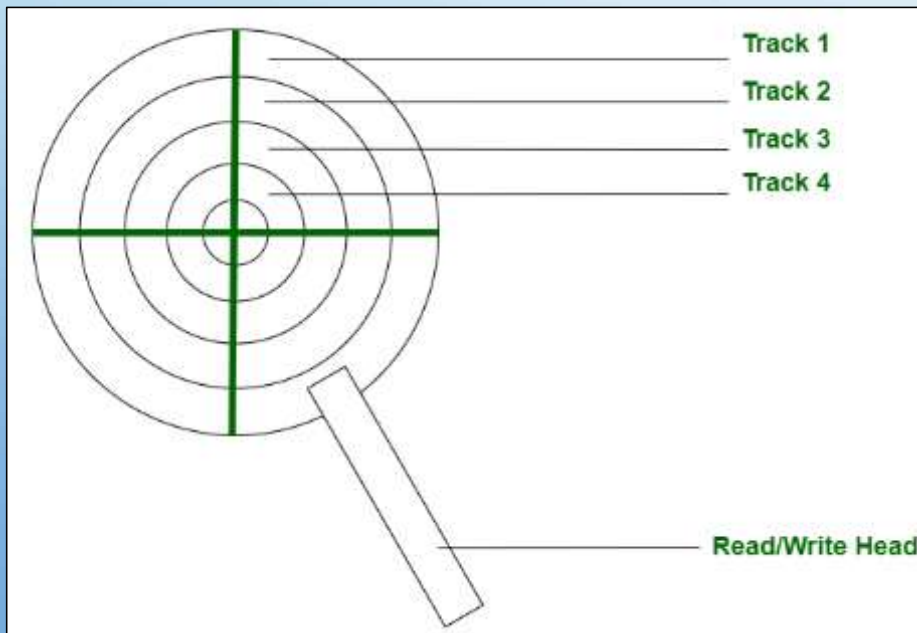
- Blocks in a file should be arranged sequentially on disk (by 'next'), to minimize seek and rotational delay.
- For a sequential scan, pre-fetching several pages at a time is a big win!

***Pre-fetching:*** Pre-fetching is a technique where the system anticipates the need for subsequent blocks and loads them into memory before they are actually requested by the process.



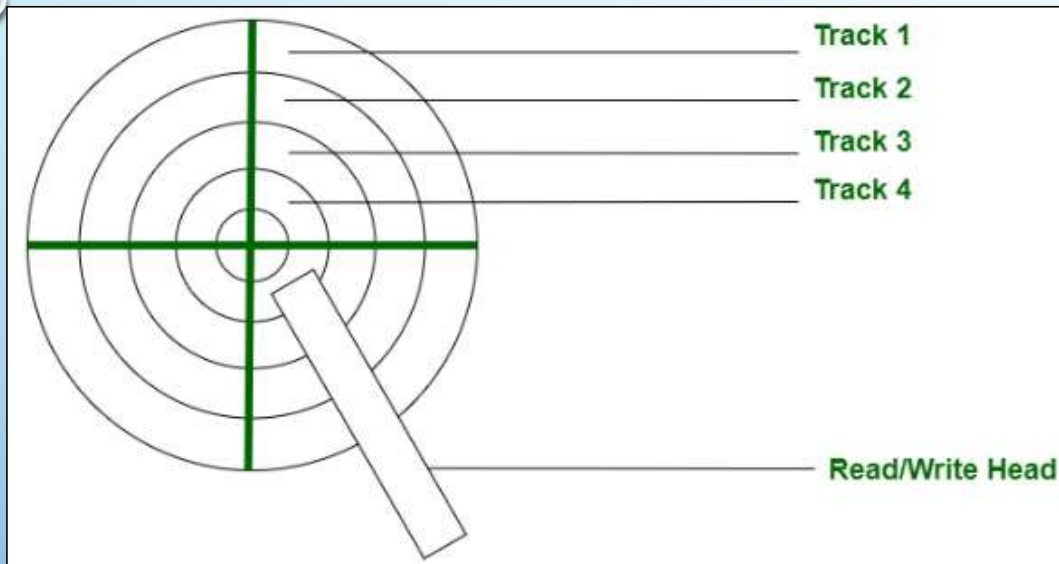
# SEEK TIME

- Time required by the read/write head to move from one track to another.



- The read/write head is currently on track 1.
- Now, on the next read/write request, we may want to read data from Track 4, in this case, our read/write head will move to track 4.

❖ The time it will take to reach track 4 is the seek time.



- It can be reduced if subsequent request belongs to same track or near.

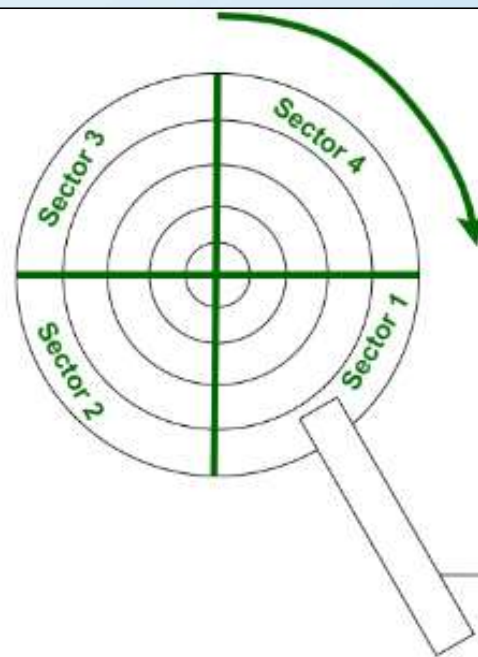
**Seek Time = (Time to cross 1 cylinder(track))\*(No of cylinder(track) crossed).**

# ROTATIONAL LATENCY

- The time required by the read/write head to rotate to the requested sector from the current position
- The read/write head is currently in sector 3.
- The data block may be present in sector 1. The time required by read/write head to move from sector 3 to sector 1 is the **rotational latency**.



Read/Write Head



Text

Read/Write Head

**Rotational Latency = (Angle between current sector and required sector) / (Rotational frequency)**

**Average Rotational Latency = (1/2) \* One rotation time**

- **Best Case** = When the header is already in the desired sector.
- **Worst Case** = When the header is at the sector far away from the desired sector, you have to wait for one complete rotation.
- **Average Case** = Half of the rotation time.

# QUESTIONS

- Consider a disk with a sector size of 512 bytes, 2000 tracks per surface, 50 sectors per track, five double-sided platters, and average seek time of 10 msec.
  1. What is the capacity of a track in bytes? What is the capacity of each surface? What is the capacity of the disk?
  2. How many cylinders does the disk have?
  3. Give examples of valid block sizes. Is 256 bytes a valid block size? 2048? 51200?
  4. If the disk platters rotate at 5400 rpm (revolutions per minutes) what is the maximum rotational delay?
  5. If one track of data can be transferred per revolution, what is the transfer rate?

**Answer 9.5** 1.

$$\text{bytes}/\text{track} = \text{bytes}/\text{sector} \times \text{sectors}/\text{track} = 512 \times 50 = 25K$$

$$\text{bytes}/\text{surface} = \text{bytes}/\text{track} \times \text{tracks}/\text{surface} = 25K \times 2000 = 50,000K$$

$$\text{bytes}/\text{disk} = \text{bytes}/\text{surface} \times \text{surfaces}/\text{disk} = 50,000K \times 5 \times 2 = 500,000K$$

2. The number of cylinders is the same as the number of tracks on each platter, which is 2000.
3. The block size should be a multiple of the sector size. We can see that 256 is not a valid block size while 2048 is. 51200 is not a valid block size in this case because block size cannot exceed the size of a track, which is 25600 bytes.
4. If the disk platters rotate at 5400rpm, the time required for one complete rotation, which is the maximum rotational delay, is

$$\frac{1}{5400} \times 60 = 0.011 \text{seconds}$$

- . The average rotational delay is half of the rotation time, 0.006 seconds.
5. The capacity of a track is 25K bytes. Since one track of data can be transferred per revolution, the data transfer rate is

$$\frac{25K}{0.011} = 2,250K \text{bytes}/\text{second}$$



## EXERCISE

Following are the specifications of a disk drive used in a database server:

- **Rotational Speed:** 7200 revolutions per minute (RPM)
- **Average Seek Time:** 9 milliseconds (ms)
- **Number of Tracks:** 200
- **Number of Sectors per Track:** 512
- **Size of Each Sector:** 512 bytes

1. Calculate the average rotational latency for this disk drive
2. If the disk is spinning at 7200 RPM, how long (in milliseconds) does it take for the disk to complete one full rotation?



## EXERCISE

1. The disk drive has an average seek time of 9 milliseconds. Explain what this means in the context of disk operations.
2. If the read/write head is currently on track 50 and needs to move to track 150, estimate the seek time assuming the seek time is proportional to the number of tracks traversed.
3. Calculate the time it takes to transfer one sector of data from the disk to the computer.
4. If the system needs to read 100 consecutive sectors from the disk, estimate the total time taken, including the time for seek, rotational latency, and transfer.
5. Combine your results from the previous questions to estimate the total time required to access and read 100 consecutive sectors, starting from when the disk is idle and the read/write head is on a random track.

# **RAID-REDUNDANT ARRAY OF INDEPENDENT DISKS**

- It is a setup consisting of multiple disks for data storage.
- They are linked together to prevent data loss and/or speed up performance.
- Having multiple disks allows the employment of various techniques like disk striping, disk mirroring and parity.

# RAID

- Disk Array: Arrangement of several disks that gives abstraction of a single, large disk.
- Goals: Increase performance and reliability.
- Two main techniques:
  - Data striping: Data is partitioned; size of a partition is called the striping unit. Partitions are distributed over several disks.
  - Redundancy: More disks -> more failures. Redundant information allows reconstruction of data if a disk fails.

# RAID

- Data Striping Example...
  - $D$  disks =  $D$  blocks are transferred simultaneously
  - Transfer Rate =  $D$  times faster
- An array of disks reduces reliability
  - Example
    - 1 disk (MTTF)  $\sim 50,000$  hours
    - 100 disks  $50,000 \text{ hours} / 100 \sim 21 \text{ days}$

# RAID

- In RAID, redundant information is stored to increase reliability
- Example.. parity scheme

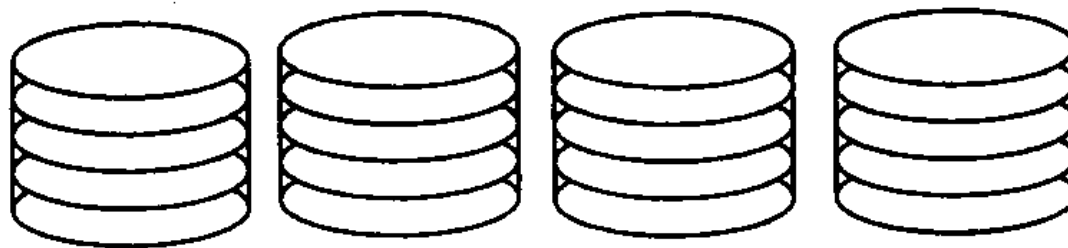
## **RAID levels or RAID arrays**

- 👉 **RAID 0** configuration is used for striping
- 👉 **RAID 1** configuration is used for mirroring
- 👉 **RAID 5** configuration is used for striping with parity
- 👉 **RAID 6** configuration is used for striping with double parity
- 👉 **RAID 10** configuration is used for combining, mirroring and striping

# RAID LEVELS

- **Level 0: No redundancy**

- Uses data striping
- Reliability an issue
- Best Write Performance (No writing of redundant data)



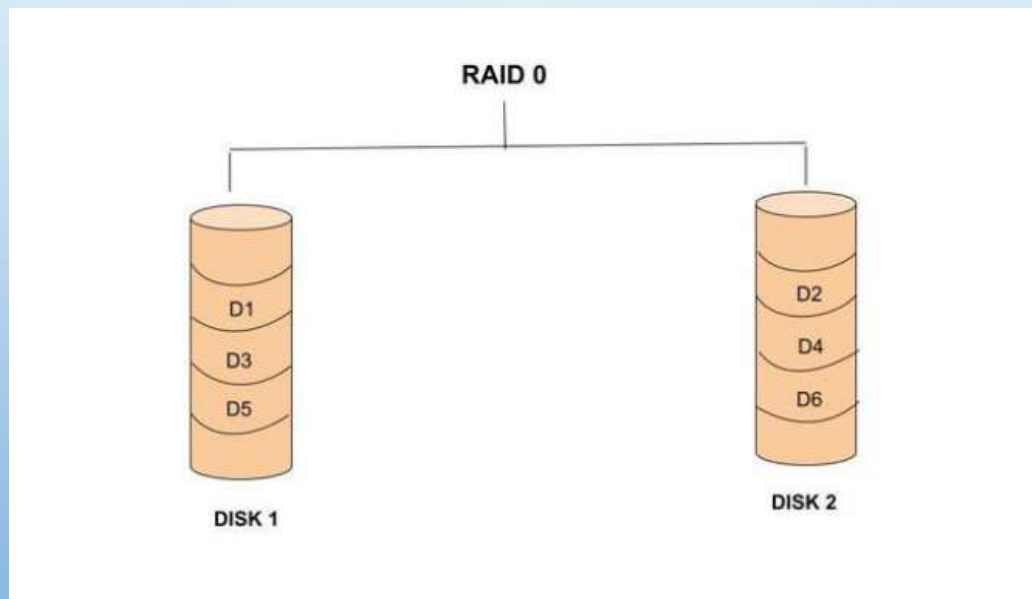
Non-Redundant (RAID Level 0)

## **DISK STRIPPING (RAID 0)**

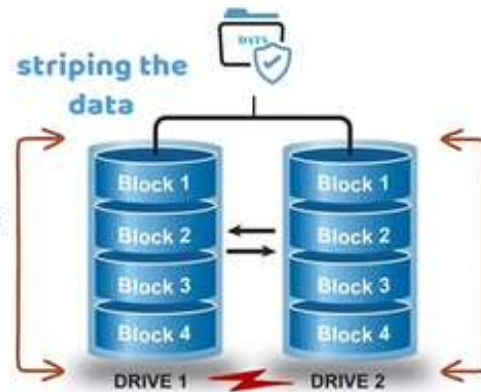
- Disk Stripping is the technique of breaking data into multiple blocks and storing those blocks on several storage disks.
- A minimum of 2 disks are used in RAID 0.
- RAID 0 does not provide data redundancy and therefore may be used for temporary storage of data where the original data is recoverable from any other storage device.



There's data named D and break it into multiple blocks of data namely D1, D2, D3, D4, D5, and D6, and store them into different disks.



## RAID 0

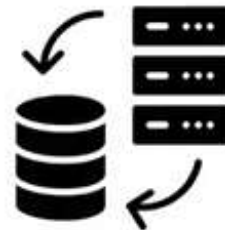


All of the **data** that was stored on both of the **drives** is **lost**



By using multiple disks (at least 2) at the same time this offers superior **I/O performance**

**Very frequent backups**



you get two **drives**  
double the  
**Read** and  
**Write** performance

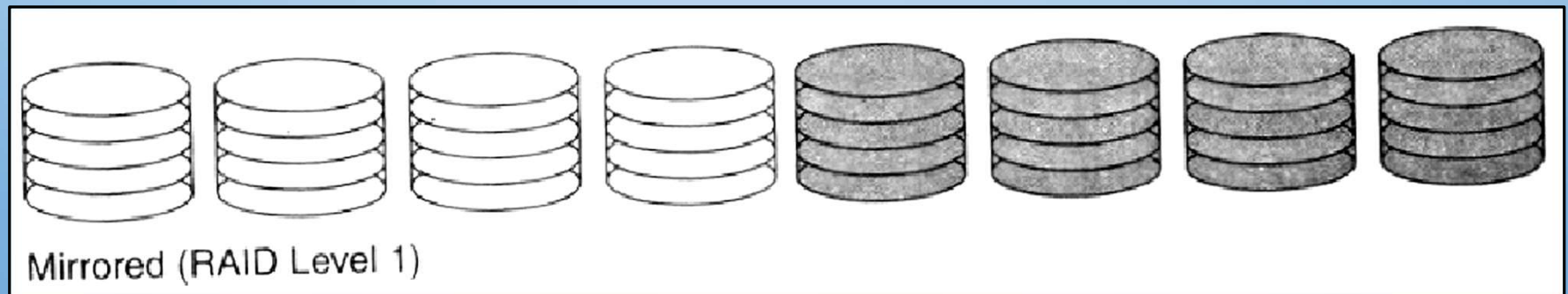
One of the drives undergoes a **hardware failure**



# RAID LEVELS

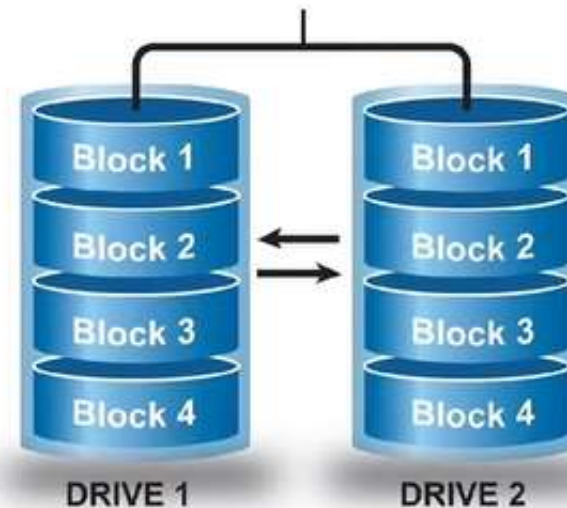
## Level 1: Mirrored (two identical copies)

- No data striping
- Most expensive solution
- Each disk has a mirror image (check disk)
- Parallel reads, a write involves two disks.
- Maximum transfer rate = transfer rate of one disk (this level does not stripe the data).
- Space utilization = 50%



There's no performance overhead for running RAID 1

## RAID 1



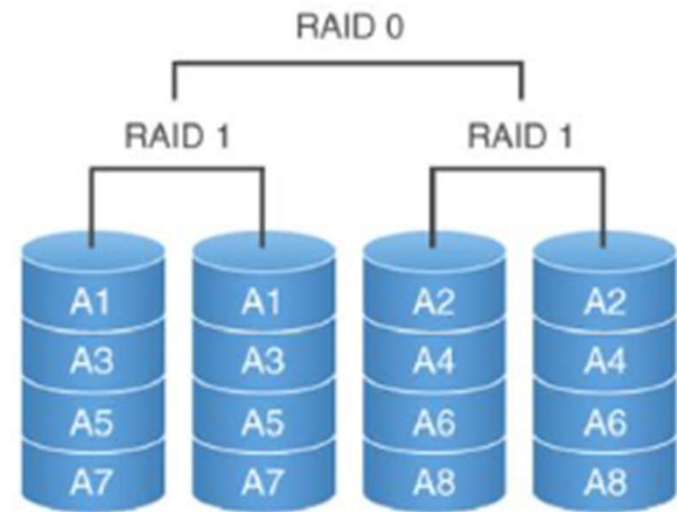
you're always only getting  
get half the capacity that  
you would otherwise have

Full performance of the drive

# RAID LEVELS

## Level 0+1: Striping and Mirroring

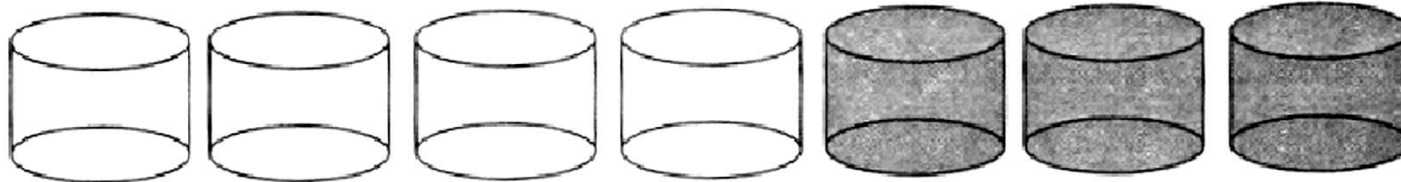
- Parallel reads, a write involves two disks.
- Maximum transfer rate = aggregate bandwidth
- Space utilization = 50%



# RAID LEVELS

## Level 2: Error-Correcting Codes

- Striping unit = 1 bit
- Redundancy scheme = Hamming Code (An error correcting code)



Memory-Style ECC (RAID Level 2)

# RAID LEVELS

## Level 2:

- It consist of Bit-level Striping.
- RAID 2 records Error Correction Code (ECC) using hamming code parity.
- In this level 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 of disk.

# Parity

- ❖ Parity information refers to the additional data stored in a physical array or RAID group to provide data protection in case of a hard disk drive failure.
- ❖ Use the XOR gate

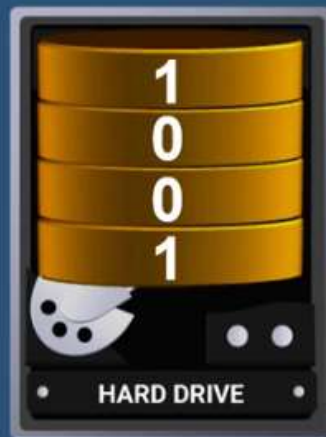


# PARITY

1 0 0 1 1 0 1 0

## RAID 5

Striping with  
parity



DISK A



DISK B



DISK C

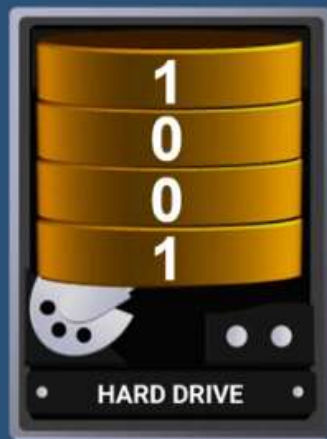
PARITY

XOR	
0 + 0	0
0 + 1	1
1 + 0	1
1 + 1	0

# PARITY

1 0 0 1 1 0 1 0

**RAID 5**  
Striping with  
parity



DISK A



DISK B



DISK C  
PARITY

XOR	
0 + 0	0
0 + 1	1
1 + 0	1
1 + 1	0

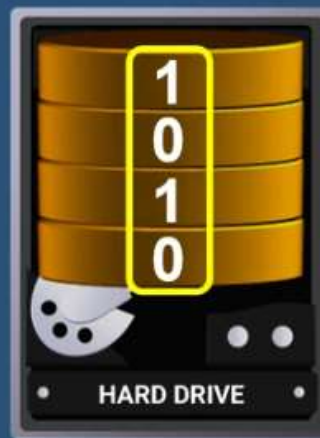
# PARITY

1 0 0 1 1 0 1 0

**RAID 5**  
Striping with  
parity



**DISK A**



**DISK B**  
RESTORED

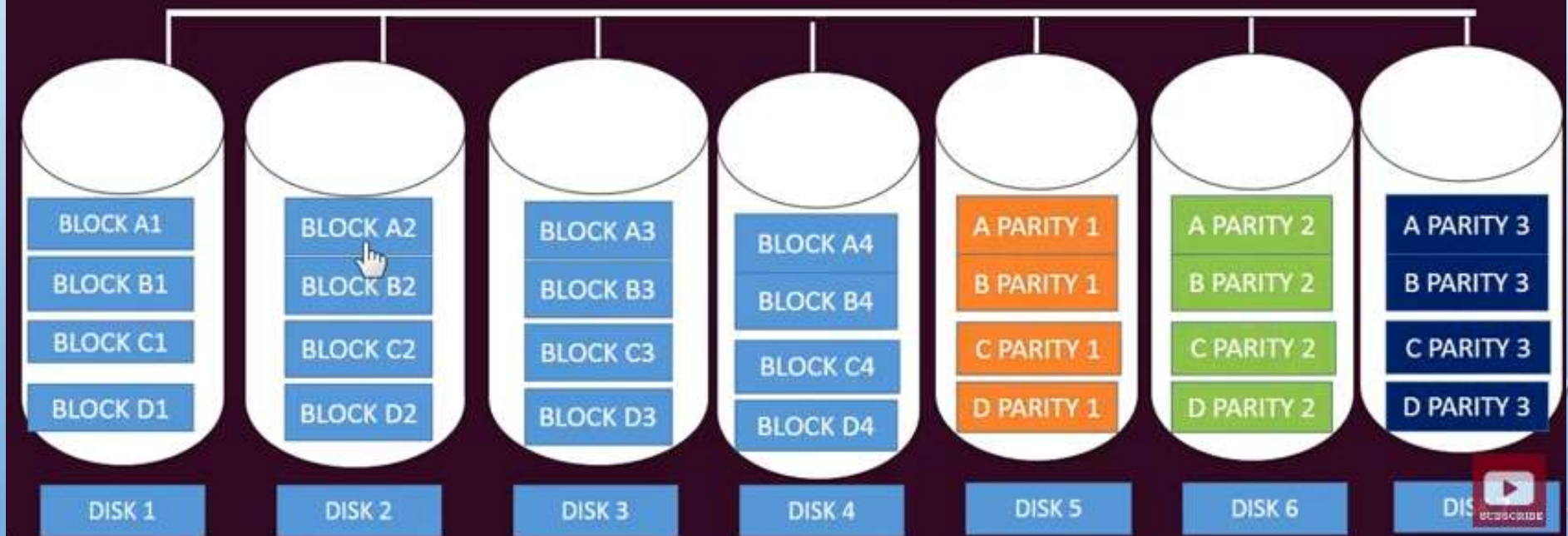


**DISK C**  
PARITY

XOR	
0 + 0	0
0 + 1	1
1 + 0	1
1 + 1	0

## RAID 2

### RAID 2 Bit Striping with Parity



## Exercise: Understanding Parity in RAID Level 2

RAID Level 2 uses bit-level striping with Hamming code error correction. In this setup, data is distributed across multiple disks at the bit level, and additional disks are used to store error-correcting codes (ECC) based on Hamming code. These ECC disks provide the ability to detect and correct single-bit errors.

You have a RAID 2 configuration with four data disks (Disk 1, Disk 2, Disk 3, Disk 4) and three ECC disks (ECC1, ECC2, ECC3). Each bit of data is stored across the four data disks, and the corresponding parity bits are stored on the ECC disks.

### 1. Data Distribution and Parity Calculation:

- Given the following 4-bit data block, calculate the parity bits using Hamming code and distribute the data across the four data disks and three ECC disks.
- Data block: **1011**



## 2. Simulate a Bit Error:

- Assume a single-bit error occurs during data transmission, corrupting the third bit on Disk 3. Identify the error using the ECC disks and correct the corrupted bit.

# Answers

**Step 1:** Distribute the data bits across the four data disks.

- Data block: **1011**
- Distribute across disks:
  - ✓ **Disk 1: 1**
  - ✓ **Disk 2: 0**
  - ✓ **Disk 3: 1**
  - ✓ **Disk 4: 1**

# Answers

**Step 2:** Calculate the Hamming code parity bits to determine the values for ECC1, ECC2, and ECC3 and store the parity bits on the ECC disks.

- **Hamming Code Parity Calculation:**

- ✓ Assume ECC1 corresponds to the parity for bits 1, 2, and 4.
- ✓ Assume ECC2 corresponds to the parity for bits 1, 3, and 4.
- ✓ Assume ECC3 corresponds to the parity for bits 2, 3, and 4.
- ✓ Calculate ECC1:  $1 \text{ XOR } 0 \text{ XOR } 1 = 0$  (Even parity)
- ✓ Calculate ECC2:  $1 \text{ XOR } 1 \text{ XOR } 1 = 1$  (Odd parity)
- ✓ Calculate ECC3:  $0 \text{ XOR } 1 \text{ XOR } 1 = 0$  (Even parity)
- ✓ ECC1 = 0, ECC2 = 1, ECC3 = 0

- **Store parity bits:**

- ✓ ECC1: 0
- ✓ ECC2: 1
- ✓ ECC3: 0



# Answers

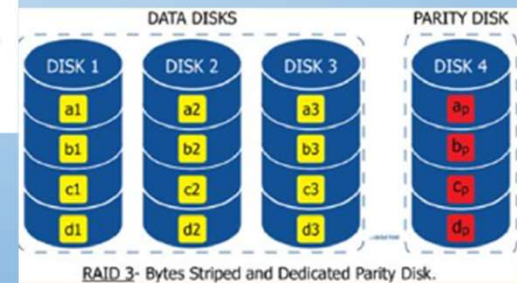
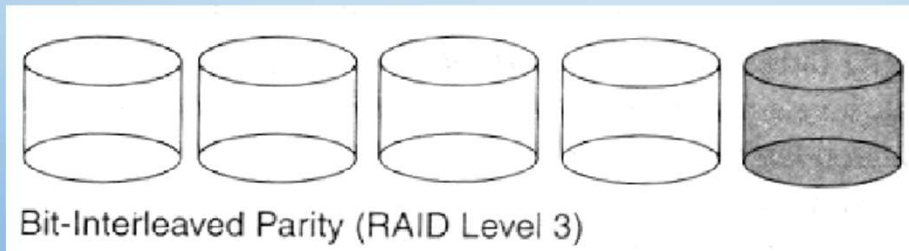
## Simulate a Bit Error:

- Introduce a single-bit error: **Change the third bit on Disk 3 from 1 to 0.**
- Data received: **1, 0, 0, 1**
- Check parity using Hamming code:
  - Calculate new ECC bits:
    - New ECC1:  **$1 \text{ XOR } 0 \text{ XOR } 1 = 0$**
    - New ECC2:  **$1 \text{ XOR } 0 \text{ XOR } 1 = 0$**
    - New ECC3:  **$0 \text{ XOR } 0 \text{ XOR } 1 = 1$**
  - Compare received ECC (0, 0, 1) with stored ECC (0, 1, 0).
  - The discrepancy indicates an error in the third bit.
- Correct the error:
  - Flip the bit on Disk 3 from 0 back to 1.
- Corrected data: **1, 0, 1, 1**

# RAID LEVELS

## Level 3: Bit-Interleaved Parity

- Striping Unit: One bit. One check disk.
- Each read and write request involves all disks; disk array can process one request at a time.



# RAID LEVELS

## **Level 3: Bit-Interleaved Parity**

- RAID 3 is a type of RAID configuration that uses byte-level striping with a dedicated parity disk.
- This means that each byte of data is divided into smaller pieces (usually individual bytes) and distributed across multiple data disks.
- A separate disk is dedicated to storing parity information, which can be used to recover data if one of the data disks fails.



# RAID LEVELS

## Level 3: Bit-Interleaved Parity

- Imagine you have a file that is 4 bytes long, and you want to store it on a RAID 3 array with 4 data disks (Disk 1, Disk 2, Disk 3, Disk 4) and 1 parity disk (Disk P).
- Example Data:

***File Content: 11001001 00110110 10101100 01111001 (4 bytes)***

**Byte 1 (11001001)**

Disk 1 (D1): 11001001

**Byte 2 (00110110)**

Disk 2 (D2): 00110110

**Byte 3 (10101100)**

Disk 3 (D3): 10101100

**Byte 4 (01111001)**

Disk 4 (D4): 01111001

- Each disk now holds one byte of the original file.

## Exercise: Understanding Parity in RAID Level 3

RAID Level 3 uses byte-level striping with a dedicated parity disk. Data is striped across multiple disks at the byte level, and a single disk is used to store the parity information. This setup allows for the recovery of data in case one of the data disks fails.

You have the following RAID 3 configuration with four data disks (Disk 1, Disk 2, Disk 3, Disk 4) and one dedicated parity disk (Disk P).

### 1. Data Distribution and Parity Calculation:

Given the following data bytes, calculate the parity byte for each set of striped bytes and store it on the parity disk.

- **Byte 1:** 10110101
- **Byte 2:** 01101011
- **Byte 3:** 11010010
- **Byte 4:** 00111001

## 2. Simulate a Disk Failure:

- Assume Disk 2 fails. Demonstrate how you can recover the lost data byte using the remaining data bytes and the parity byte.

## 3. Write and Recover New Data:

- After recovering the data, write the following new data bytes to the RAID array, calculate the new parity byte, and demonstrate the recovery process if Disk 3 fails.
  - **Byte 1:** 11001100
  - **Byte 2:** 10110110
  - **Byte 3:** 01100101
  - **Byte 4:** 10011011

# Answers

## 1. Data Distribution and Parity Calculation:

### Data Bytes:

- Disk 1: **10110101**
- Disk 2: **01101011**
- Disk 3: **11010010**
- Disk 4: **00111001**

### Parity Calculation (using XOR):

- **Parity Byte:** 10110101 XOR 01101011 XOR 11010010 XOR 00111001
- **Resulting Parity Byte:** **00110101**

### Data and Parity Disk Setup:

- Disk 1: **10110101**
- Disk 2: **01101011**
- Disk 3: **11010010**
- Disk 4: **00111001**
- Disk P (Parity): **00110101**



# Answers

## 2. Simulate a Disk Failure:

- Assume Disk 2 fails.

To recover the data on Disk 2, XOR the remaining data bytes and the parity byte:

- **Recovered Byte 2:**  $00100111 \text{ XOR } 10110101 \text{ XOR } 11010010 \text{ XOR } 00110101$
- **Recovered Byte 2:** **01101011**

# Answers

## 3. Write and Recover New Data:

### New Data Bytes:

- Disk 1: **11001100**
- Disk 2: **10110110**
- Disk 3: **01100101**
- Disk 4: **10011011**

### New Parity Calculation:

- **New Parity Byte:** 11001100 XOR 10110110 XOR 01100105 XOR 10011011
- **Resulting Parity Byte:** 10000100

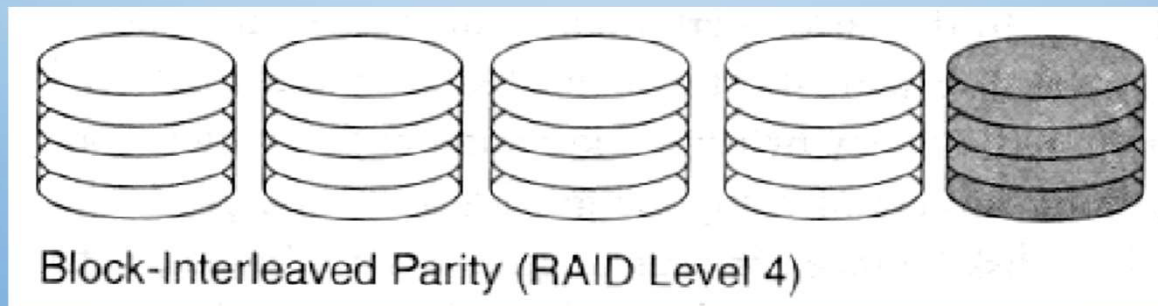
### Assume Disk 3 fails.

- To recover the data on Disk 3, XOR the remaining data bytes and the new parity byte:
  - **Recovered Byte 3:** 10011110 XOR 11001100 XOR 10110110 XOR 10000100
  - **Recovered Byte 3:** **01100101**

# RAID LEVELS

## Level 4: Block-Interleaved Parity

- Striping Unit: One disk block. One check disk.
- Parallel reads possible for small requests, large requests can utilize full bandwidth
- Writes involve modified block and check disk



# RAID LEVELS

- **Level 4: Block-Interleaved Parity**

Assume you have a file that is divided into blocks. For simplicity, let's say the file has 4 blocks of data:

*Block 1: 10101100*

*Block 2: 11010111*

*Block 3: 01101001*

*Block 4: 10011010*

# RAID LEVELS

## • Level 4: Block-Interleaved Parity

In RAID 4, each block of data is written to a different data disk, while the parity block is calculated and stored on the dedicated parity disk:

- *Block 1 (10101100): Stored on Disk 1 (D1)*
- *Block 2 (11010111): Stored on Disk 2 (D2)*
- *Block 3 (01101001): Stored on Disk 3 (D3)*
- *Block 4 (10011010): Stored on Disk 4 (D4)*

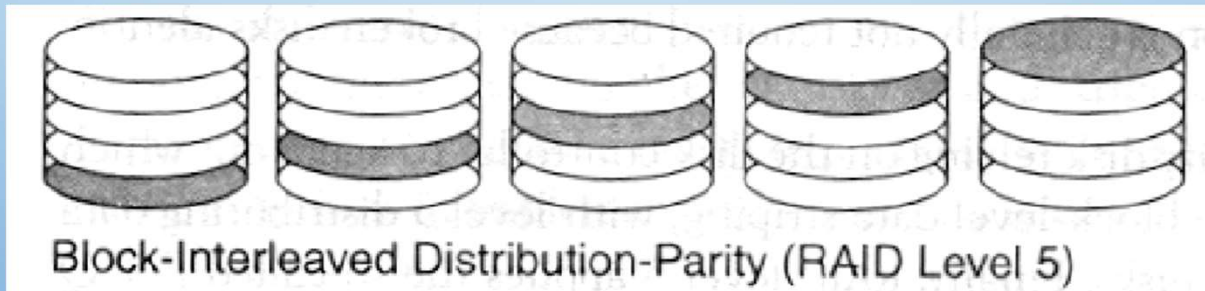
*Parity Block = Block 1 XOR Block 2 XOR Block 3 XOR Block 4*

**Find the parity block value...**

# RAID LEVELS

## Level 5: Block-Interleaved Distributed Parity

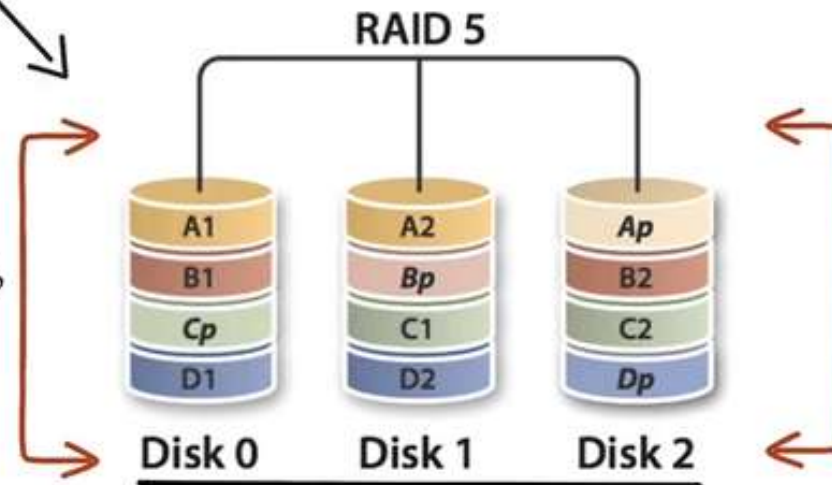
- Similar to RAID Level 4, but parity blocks are distributed over all disks





The parity data is not written to a fixed drive, they are spread across all drives

Data is not duplicated



but it's striped or spread across multiple disks

**On one drive a parity checksum of all the block data is written**

## Exercise: Understanding Parity in RAID Level 5

- RAID 5 uses block-level striping with distributed parity. This means that data and parity information are striped across all disks in the array, with parity blocks distributed among the disks. This setup allows for data recovery if any one disk fails.

You have a RAID 5 array with 4 disks: Data Disks 1, 2, 3, and 4. Each disk holds a block of data or parity information.

Assume you have the following blocks of data and need to calculate the parity blocks.

Data Blocks:

- Block A1: 11010010
- Block A2: 10111100
- Block A3: 01101011
- Block A4: 10010101
- Block A5: 11100011
- Block A6: 01011101

1. **Calculate Parity Blocks**
2. **Assign Data and Parity Blocks to Disks**
3. **Assume disk 1 fails, handle the disk failure**



# Answers

## Calculate Parity Blocks

Calculate Parity Block (P1) for the first block position:

Parity Block (P1) = Block A1 XOR Block A2 XOR Block A3

Perform the XOR operation:

Parity (P1): 00000101

Parity Block (P2) = Block A4 XOR Block A5 XOR Block A6

Perform the XOR operation:

Parity (P2): 00101011

# Answers

## **Data Blocks and Parity:**

### **Row 1:**

- Disk 1: 110010 (Block 1)
- Disk 2: 101110 (Block 2)
- Disk 3: 011011 (Block 3)
- Disk 4: 000111 (Parity P1)

### **Row 2:**

- Disk 1: 100101 (Block 4)
- Disk 2: 111000 (Block 5)
- Disk 3: 001010 (Parity P2)
- Disk 4: 010111 (Block 6)

# Answers

## **Recovered Block D1- R1**

= Block D2-1 XOR Block D3-1 XOR Parity Block P1

= 101110 XOR 011011 XOR 000111

= 110010

## **Recovered Block D1- R2**

= Block D2-2 XOR Parity Block P2 XOR Block D4-2

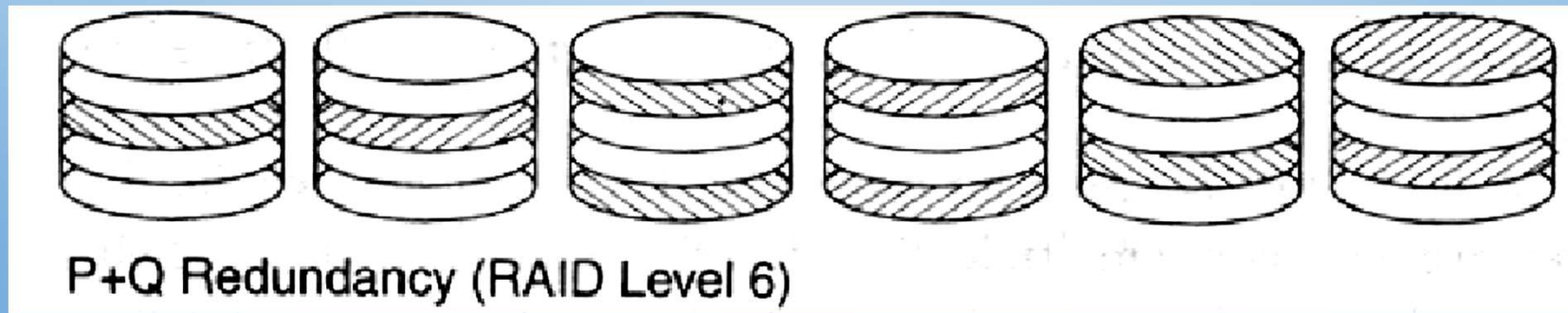
= 111000 XOR 001010 XOR 010111

= 100101

# RAID LEVELS

## Level 6: P+Q Redundancy

- Similar to RAID Level 5
- Two check disks (uses Reed-Solomon codes)
- Able to recover from 2 failures

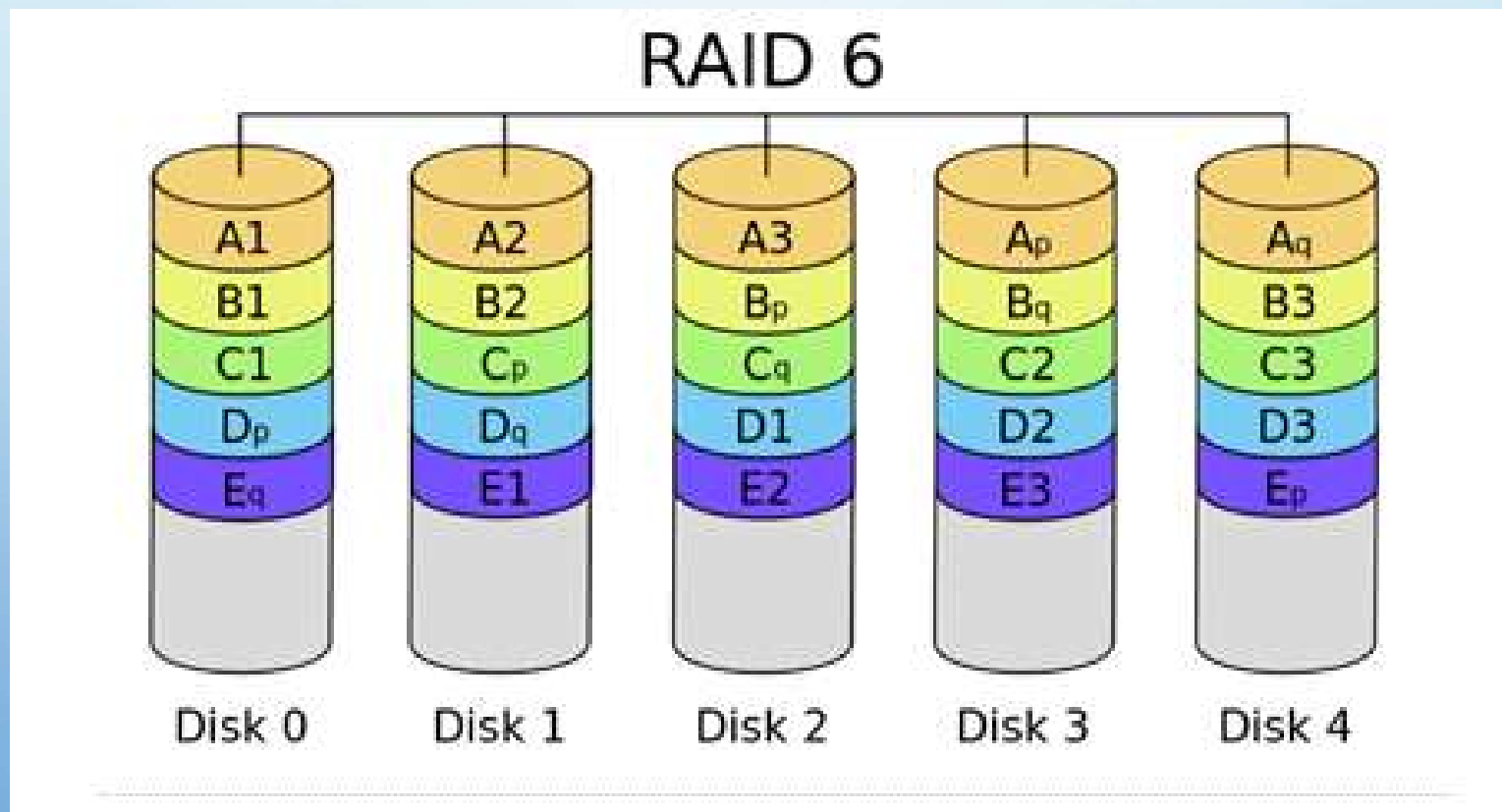


# RAID LEVELS

## Double Parity

- Single parity protects data against the loss of a single storage device.
- But, what if you want protection against the loss of two devices?
- Can we extend the parity idea to protect the data even in that scenario?

# RAID LEVELS



# EXAMPLES

Consider defining P and Q like this:

$$P = a + b + c + d + e$$

$$Q = 1a + 2b + 3c + 4d + 5e$$

# DISK SPACE MANAGEMENT

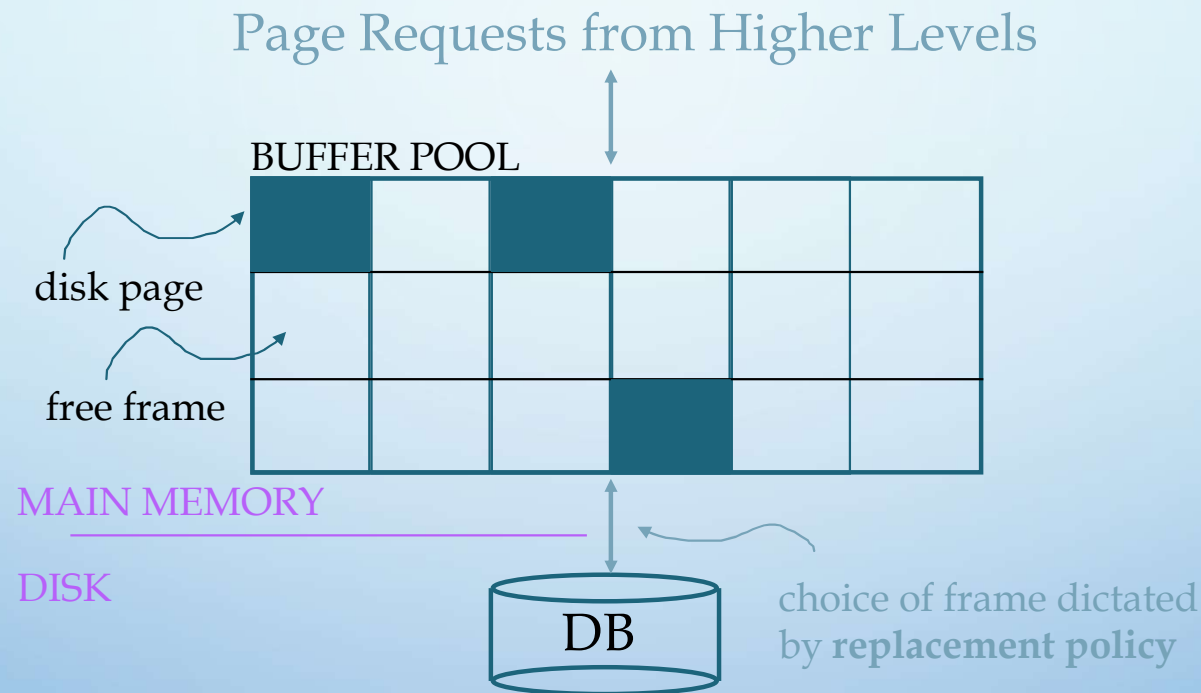
- ❖ Lowest layer of DBMS software manages space on disk.
- ❖ Higher levels call upon this layer to:
  - ✓ allocate/de-allocate a page
  - ✓ read/write a page
- ❖ Page = 1 disk block
- ❖ 1 page request = 1 disk block
- ❖ Request for a *sequence* of pages must be satisfied by allocating the pages sequentially on disk!



# DISK SPACE MANAGEMENT

- Higher levels don't need to know how this is done, or how free space is managed.
- Maintaining free blocks...
  - Keeping a list of free blocks & a pointer to the first free block
  - Maintaining a Bitmap

# BUFFER MANAGEMENT IN A DBMS



- *Data must be in RAM for DBMS to operate on it!*
- *Table of  $\langle \text{frame\#}, \text{pageid} \rangle$  pairs is maintained.*
- *For each frame#, 2 variables (pin count: no of current users & dirty bit: page is modified)*

## WHEN A PAGE IS REQUESTED ...

- The buffer manager checks the buffer pool to see if some frame contains the requested page.
  - If so, (pin count++) for that frame
- If requested page is not in pool:
  - Choose a frame for *replacement*
  - If frame is dirty, write it to disk
  - Read requested page into chosen frame
- *Pin* the page and return its address.

*If requests can be predicted (e.g., sequential scans)  
pages can be pre-fetched several pages at a time!*

## MORE ON BUFFER MANAGEMENT

- If a requested page is not in the buffer pool and a free frame is not available.
  - A frame with `pin_count = 0` is chosen for replacement according to the buffer replacement policy.
  - If no page in the pool has `pin_count 0`, the buffer manager has to wait till some page is released; the requesting transaction may be simply aborted.

## MORE ON BUFFER MANAGEMENT

- Requestor of page must unpin it, and indicate whether page has been modified:
  - *dirty* bit is used for this.
- If a page is requested by several different transactions
  - Each transaction should obtain a lock on the page before it read or modify the page (locking protocol)

# BUFFER REPLACEMENT POLICY

- Least recently used(LRU)
  - A queue of pointers to frames with pin\_count 0
  - A frame is added to the end of the queue. When it becomes a candidate for replacement.
  - The frame at the head of queue is chosen for replacement.
- Clock
- First in first out(FIFO)
- Most recently used (MRU)

## BUFFER REPLACEMENT POLICY

- Frame is chosen for replacement by a *replacement policy*:
  - Least-recently-used (LRU), Clock, MRU, random etc.
- Policy can have big impact on # of I/O's; depends on the *access pattern*.
- Sequential flooding: Nasty situation caused by LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O. MRU much better in this situation (but not in all situations, of course).

# DBMS VS. OS FILE SYSTEM

OS does disk space & buffer mgmt.: why not let OS manage these tasks?

- Differences in OS support: portability issues
- Some limitations, e.g., files can't span disks.
- Buffer management in DBMS requires ability to:
  - **pin a page** in buffer pool, **force a page** to disk (important for implementing CC & recovery),
  - adjust *replacement policy*, and **pre-fetch pages** based on access patterns in typical DB operations.



## **EXERCISE**

1. How will the buffer manager identify a dirty page?
2. How to handle a dirty page?
3. How will buffer manager know if a page is in use?
4. If buffer manager is full, which page should be replaced?

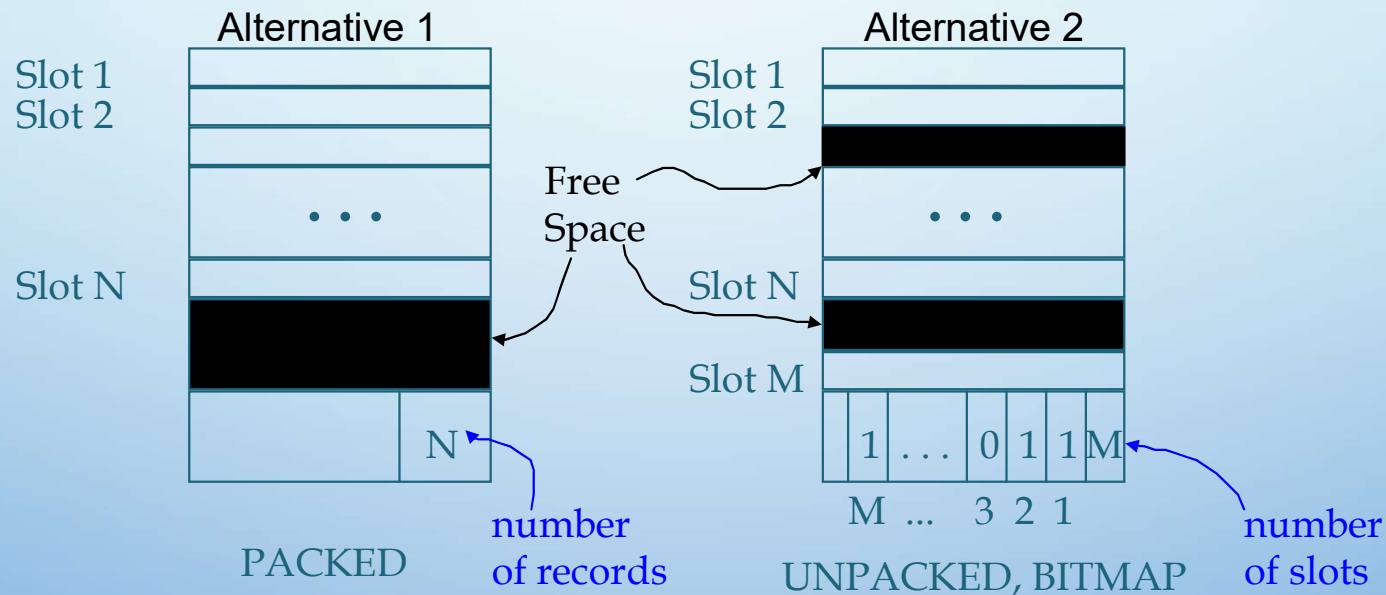
# ANSWERS

1. A dirty bit on a page
2. Write back to the disk
3. Page pin count
4. Page replacement policy

## PAGE FORMATS

- DBMS see data as a collection of records
- Hence, for fixed-length records, pages can be considered as a collection of **slots**
- Each slot contains a record
- Each record contains a **record id**

# PAGE FORMATS: FIXED LENGTH RECORDS

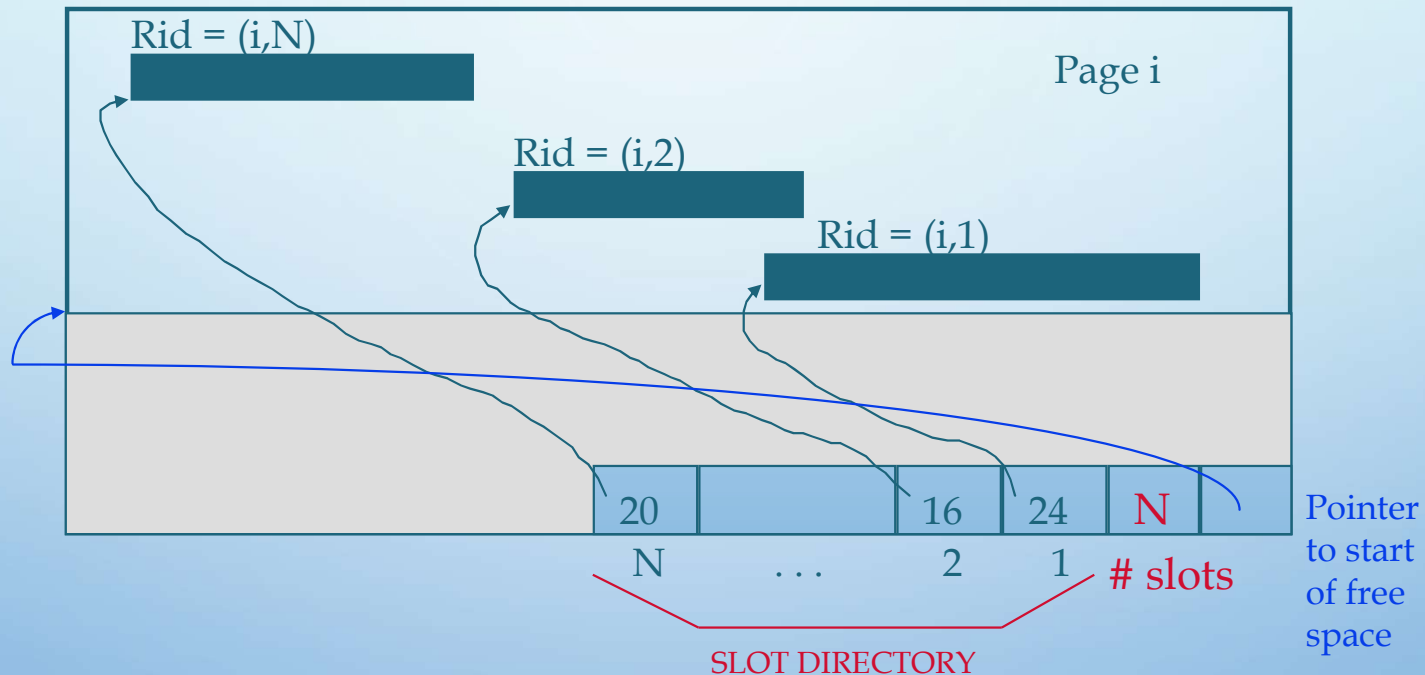


- Record id =  $\langle \text{page id}, \text{slot \#} \rangle$ . In first alternative, moving records for free space management changes rid; may not be acceptable.

# PAGE FORMATS: VARIABLE LENGTH RECORDS

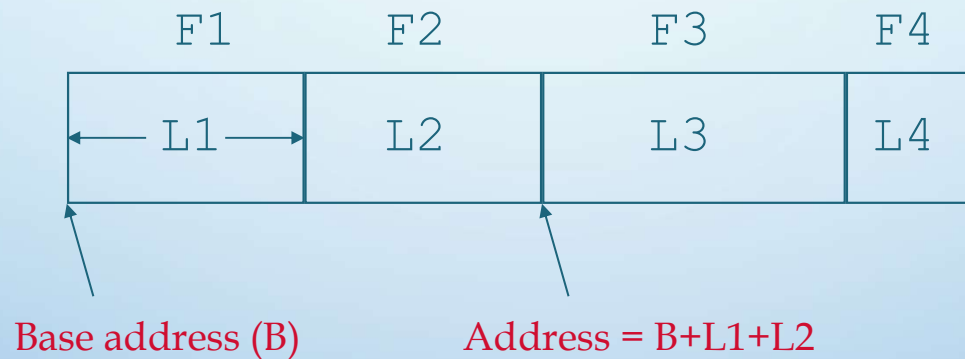
- Fixed length slots not possible
- Slot should be as large as the largest record
- If not, waste space for smaller records
- In variable-length records, having contiguous free space needs is highly desirable (avoid free blocks in the middle of pages)

# PAGE FORMATS: VARIABLE LENGTH RECORDS



- Maintains  $\langle \text{record offset}, \text{record length} \rangle$  parameters. Offset in slot directory & length first few bytes of the record or catalog for fixed length records
- Can move records on page without changing rid; so, attractive for fixed-length records too.

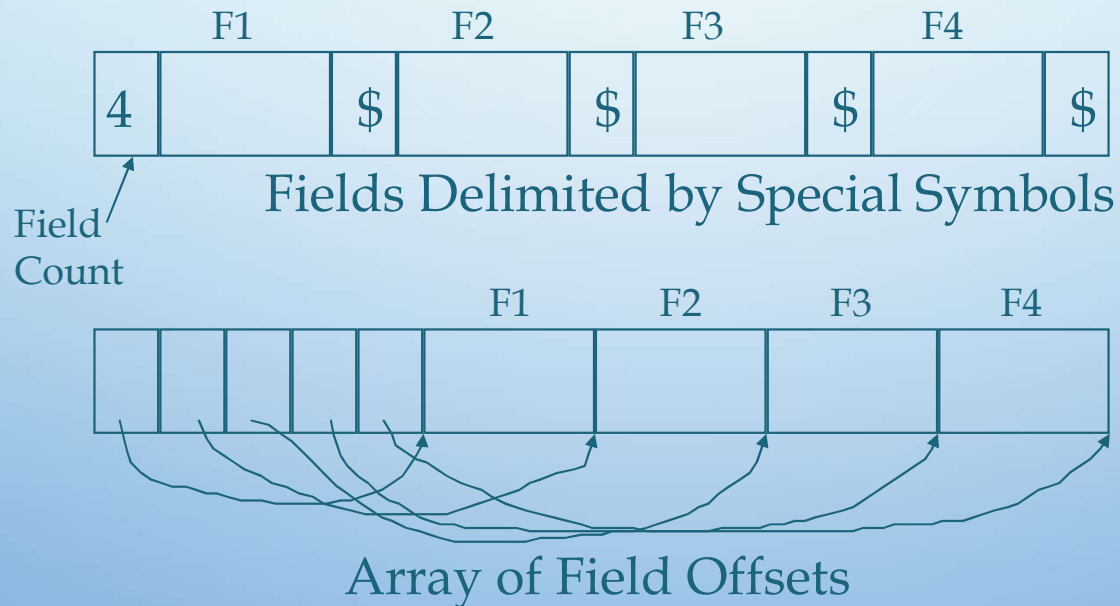
## RECORD FORMATS: FIXED LENGTH



- Information about field types same for all records in a file; stored in *system catalogs*.
- Finding *i*'th field requires scan of record.

# RECORD FORMATS: VARIABLE LENGTH

- Two alternative formats (# fields is fixed):



Second offers direct access to  $i$ 'th field, efficient storage of nulls (special *don't know* value); small directory overhead.



# SUMMARY

- COMPONENTS OF A DBMS
- DISKS
- RAID
- DISK SPACE MANAGEMENT
- BUFFER MANAGEMENT
- PAGE FORMATS
- RECORD FORMATS



**THANK YOU**

**COMPONENTS OF A DBMS,  
DISKS AND FILES**