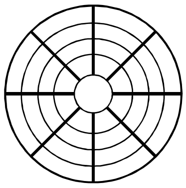
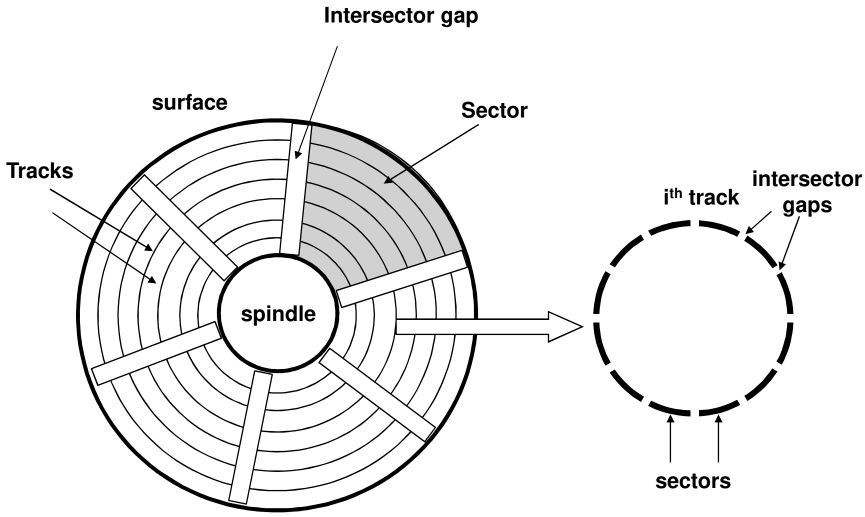
7 External Memory, Disk System

7.1 Magnetic Disk (External Memory):

- Both sides of a platter are used (double sided).
- Data is written/read on a concentric set of rings, called **tracks.**
- There are thousands of tracks per surface.
- Adjacent tracks are separated by intertrack **gaps** to prevent errors due to misalignment of the head.
- Data are transferred to and from the disk in **sectors**.
- There are typically hundreds of sectors per track.
- In most contemporary systems, fixed-length (512-byte) sectors are used.
- Adjacent sectors are separated by intersector gaps.

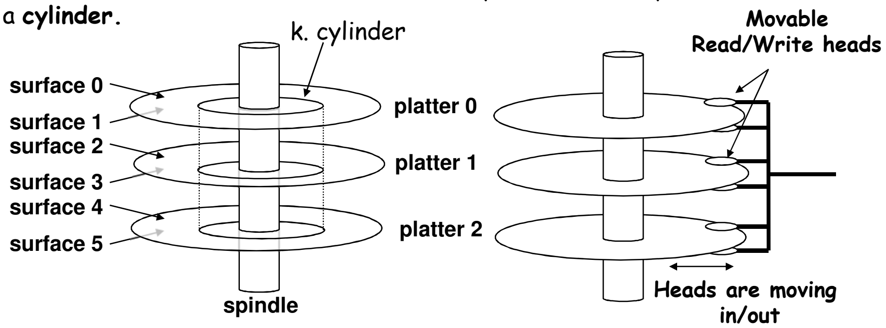


Disk data layout:



Physical Characteristics:

The set of all the tracks in the same relative position on the platter is referred to as a **cylinder**.



Capacity= (bytes/sector) x (avg. sectors/track) x (tracks/surface) x (surfaces/platter) x (platters/disk)

Example:

512 bytes/sector
300 sectors/track (average)
20,000 tracks/surface
2 surfaces/platter
5 platters/disk

Capacity= 512 x 300 x 20000 x 2 x 5
= 30,720,000,000 bytes
≈ 30.7 GB
≈ 28.6 GiB

Disk Performance

The average access time of a disk consists of three components:

Access time (Ta) = Seek time (Ts) + Rotational latency(Tr) + Transfer Time(Tt)

- **Average seek time Ts:**

The time it takes to position the head at the track. Typically, 9ms on contemporary hard disks (3-15ms , depending on device)

- **Average rotational latency, Tr:**

The time it takes for the beginning of the sector to reach the head.
After the head is positioned on the track, the disk controller waits until the appropriate sector rotates to line up with the head.
This waiting time is, on average, the half of the rotation time (time it takes to do a full rotation) of the platter:

$$T_r = \frac{1}{2} r$$

r: Rotation time of a disk (in seconds)

Rotation speeds of disks are given in **revolutions per minute** (RPM).
The rotational latency in seconds can be calculated as follows:

$$T_r = \frac{1}{2} \frac{60}{RPM}$$

Example: A disk with a speed of 7200 RPM completes a full rotation in 8.3 ms. Hence, the average rotational latency is 4ms.
For 10000 RPM: Tr = 3ms, For 15000 RPM: Tr = 2ms.

- **Transfer Time, Tt:**

It can be expressed in two different ways:

The time required to transfer one sector (Tts) or

The time required to transfer a predetermined number of bytes (Ttb)

a) The time required to transfer one sector (Tts):

$$T_{ts} = \frac{1}{avg.sector/track} \cdot \frac{60}{RPM} \ [seconds]$$

Example: The rotation speed of a disk is 7200 RPM and, there are, on average, 400 sectors per track.

The time required to transfer one sector (Tts) :

Tts = 60/7200 [RPM] x 1/400 [sectors/track] x 1000 [ms/sec] = 0.02 ms

b) Data transfer time (Ttb):

b: Number of bytes to transfer, N: Number of bytes per track

$$T_{tb} = \frac{b}{N}\frac{60}{RPM} \ \text{[seconds]}$$

Example:

- Disk rotation speed = 7200 RPM
- Average seek time = 9 ms
- Sectors per track (on average) = 400

Based on these values:

- Rotational latency= 1/2 x (60s / 7200 RPM) x 1000 ms/s = 4 ms
- Transfer time = 60/7200 RPM x (1/400 sectors/track) x 1000 ms/sec = 0.02 ms
- Access time = 9 ms + 4 ms + 0.02 ms

The dominating components are the seek time and the rotational latency.

Evolution of disks:

Improvement in capacity is customarily expressed as improvement in *areal density*, measured in bits per square inch (or centimeter):

Areal density = $\frac{Tracks}{Inch}$ on a disk surface $\times$ $\frac{bits}{Inch}$ *on a track*

Through about 1988, the rate of improvement of areal density was 29% per year, thus doubling density every three years.

Between 1988 and about 1996, the rate improved to 60% per year.

From 1997 to about 2003, the rate increased to 100%, doubling every year.

Between 2003 and 2011, the rate dropped back to about 40% per year.

In 2011, the highest density in commercial products was 400 billion bits per square inch.

Cost per gigabyte has dropped at least as fast as areal density has increased, with smaller diameter drives playing the larger role in this improvement.

Costs per gigabyte improved by almost a factor of 1,000,000 between 1983 and 2011.

Source: John L. Hennessy, David A. Patterson "Computer Architecture, A Quantitative Approach", 6 ed., Morgan Kaufmann, 2017.

Evolution of disks (cont'd):

After 2011, disk improvement has slowed to less than 5% per year.

One way to increase disk capacity is to add more platters, but there are already seven platters within the one-inch depth of the 3.5-inch form factor disks.

There is room for at most one or two more platters.

The last hope for real density increase is the Heat-Assisted Magnetic Recording (HAMR).

The HAMR uses a small laser on each disk read-write head to heat a 30 nm spot to 400°C so that it can be written magnetically before it cools.

It is unclear whether Heat Assisted Magnetic Recording can be manufactured economically and reliably.

Hard drive manufacturer Seagate promises that hard drives in 18TB and 20TB models will be available in retail channels in 2020.

In November 2020, hard drive manufacturer Seagate started to ship 20TB hard disk drives on a limited basis in their Enterprise Data Solutions (EDS) products and to select datacenter customers, as they continue collecting production and field data.

(https://www.techradar.com/news/seagate-confirms-20tb-hamr-hard-disk-drives-have-been-shipped).

Disk vs. DRAM vs. Flash Memory

(Source: Hennessy, Patterson)

Disk vs. DRAM (main memory):

DRAM latency is about between 100,000 and 1,000,000 times less than disk.

The typical access time of a DRAM is between 50ns and 100 ns.

The typical access time of a disk is between 5ms and 100 ms (nano vs. mili!).

A disk is 200–300 times cheaper per bit than DRAM.

Flash Memory:

Many have tried to invent a technology cheaper than DRAM but faster than disk to fill that gap, but thus, far all have failed.

The closest challenger is **Flash memory**.

Flash memory is a type of EEPROM (electronically erasable programmable read-only memory), which is normally read-only but can be erased.

This semiconductor memory is nonvolatile.

Flash is popular in mobile devices and laptops because it is more power efficient than disks.

Flash Memory (cont'd):

Flash uses a different architecture and has different properties than standard DRAM and magnetic disk.

Properties of flash memory:

- Read operations: Reads to Flash are sequential and read an entire page, which can be 512 bytes, 2 KiB, or 4 KiB.

  Thus Flash has a long delay to access the first byte from a random address (about 25 µs), but can supply the remainder of a page block at about 40 MiB/s.

  Comparing the time to transfer 2 KiB, Flash is about 150 times slower than DRAM.

  Compared to magnetic disk, a 2 KiB read from Flash is 300 to 500 times faster.

  We can see why Flash is not a candidate to replace DRAM for main memory, but is a candidate to replace magnetic disk.

- Write operations: Flash memory must be erased (thus the name flash for the "flash" erase process) before it is overwritten, and it is erased in blocks rather than individual bytes or words.

  For writing, Flash is about 1500 times slower then SDRAM, and about 8–15 times as fast as magnetic disk.

Flash Memory (cont'd):

Properties of flash memory (cont'd):

- Power consumption: It draws significantly less power when not reading or writing (from less than half in standby mode to zero when completely inactive).
- Lifetime: Flash memory limits the number of times that any given block can be written, typically between 100,000 and 1 million.

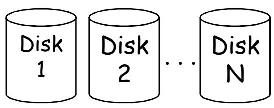
  They are not suitable for large servers. Some servers combine disk and Flash-based storage.
- Cost: Flash memory is cheaper than SDRAM but more expensive than disks: roughly \$2/GiB for Flash, \$20 to \$40/GiB for SDRAM, and \$0.09/GiB for magnetic disks.

  In the past five years, Flash has decreased in cost at a rate that is almost twice as fast as that of magnetic disks.

7.2 RAID: (*Redundant Array of Independent/Inexpensive Disks*)*

Data are distributed across the physical drives of an array.

Purpose:



Improving the **performance** and the **reliability**.

- The parallel operation of independent disks improves the rate at which data can be read or written.
- Redundancy increases the reliability.

Raid Levels:

RAID 0 – RAID 6: There are 7 main levels and their combinations.

RAID 0, is not a real member of the RAID family due to lack of redundancy.

RAID 3 and 5 are used more frequently.

Common properties:

1. There is a set of physical disk drives viewed by the operating system as a single logical drive.
2. Logically sequential data are distributed across the physical drives of an array in a scheme known as striping.
3. Redundant disk capacity is used to store **parity** information.

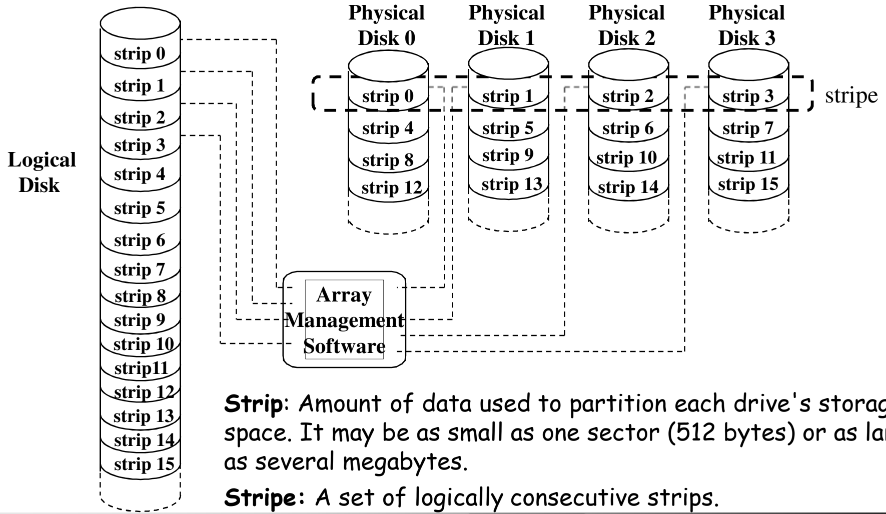
   In case of a disk **failure**, data can be recovered with the help of the parity.

* Source: W. Stallings, " Computer Organization and Architecture", 8/e, 2010.

RAID 0

No redundancy (parity). Data cannot be recovered. Not a true RAID structure.
Data is split across drives, resulting in higher data throughput.

Example: RAID 0 with 4 data disks (N = 4)



Strip: Amount of data used to partition each drive's storage space. It may be as small as one sector (512 bytes) or as large as several megabytes.

Stripe: A set of logically consecutive strips.

RAID 0 (cont'd)

Increase in throughput (Two cases):

1. If a single I/O request consists of multiple logically contiguous strips, then up to N strips for that request can be handled in parallel, greatly reducing the I/O transfer time (N : number of data disks operating in parallel).
2. If two different I/O requests are pending for two different blocks of data, then there is a good chance that the requested blocks are on different disks.

   Thus, the two requests can be issued in parallel, reducing the I/O queuing time.

Effect of strip size on performance

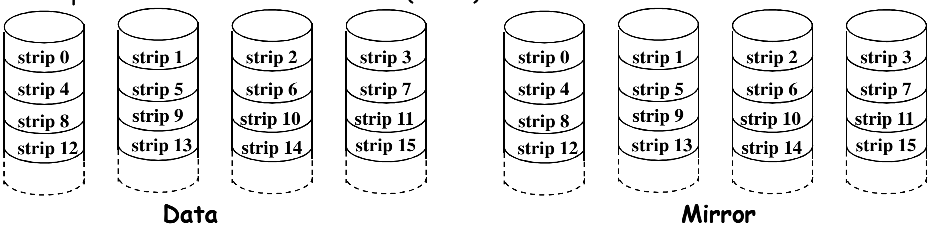
a) If the typical request is for **large amounts of logically contiguous data**, compared to the size of a strip (e.g., copying files or video playback) :
   - Transfer rate is important:
   - Using **small strips** makes sense: A single I/O request involves the parallel transfer of data from multiple disks.

b) If it is a **transaction-oriented environment**, where many **random** I/O requests per second are for small amounts of data (e.g., database access):
   - If the **strip size is relatively large**, so that a single I/O request only involves a single disk access, then multiple waiting I/O requests can be handled in parallel.

RAID 1

Redundancy is achieved by duplicating all the data.
Each data is written to two separate disk drives (**mirroring**).

Example: RAID 1 with 4 data disks (N = 4)

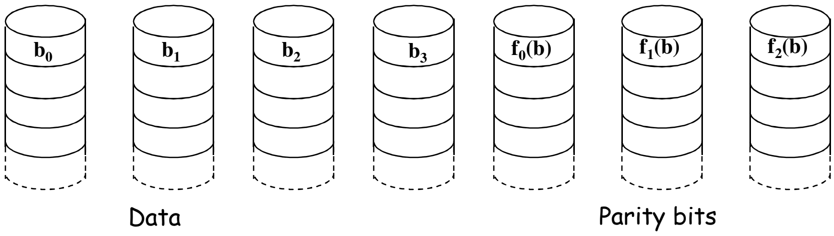


- The number of data disks: N The total number of disks : 2·N
- A read request is sent to both disks. The data is read from the fastest disk.
- A write request requires that both disks be updated in parallel. Thus, the write performance is dictated by the slower of the disks.
- When a drive fails, the data may still be accessed from the second drive.
  Note that, here, disk failure means that the disk is broken, and the failed disk can be identified.
- The cost of RAID 1 is high.

RAID 2

For error detection/correction, **a Hamming** code is used, which is able to correct single-bit errors and detect double-bit errors.

- Hamming codes are used in memory systems as well as in data communications for error detection/correction (explained in section 7.3.1).
- The spindles of the individual drives are **synchronized** so that each disk head is in the same position on each disk at any given time.
- **Small strips** are used. Suitable for large amounts of logically contiguous data.
- In the figure below, 3 parity bits are added to 4 bits of data.
- Since it is possible to physically determine which disk fails, Hamming codes unnecessarily increase the complexity and cost for disk systems.
- Although RAID 2 requires fewer disks than RAID 1, it is still rather costly.

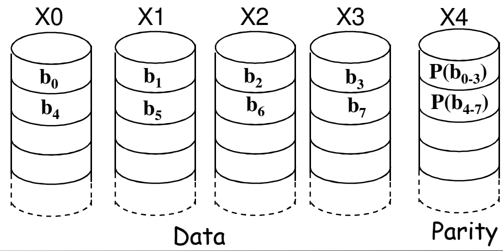


RAID 3

- As in RAID 2, the spindles of the individual drives are **synchronized** so that each disk head is in the same position on each disk at any given time.
- **Small strips** are used. Suitable for large amounts of logically contiguous data.
- RAID 3 requires only a single redundant disk, no matter how large the disk array.
- The number of data disks: N   The total number of disks : N+1
- Instead of an error-correcting code, a simple parity bit is computed for the set of individual bits in the same position on all of the data disks.

  In the event of a 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 and operation resumed.

Example:
RAID 3 with 4 Data + 1 Parity Disks
(N = 4)



RAID 3 (cont'd)

Parity:

- The parity bit is calculated using the exclusive-OR "XOR" (⊕) function.

  The parity bit is set so that the total number of 1s in data + parity is an even number.

  If X0-X3 are data disks and X4 is the parity disk, the parity for the i^{th} bit is calculated as follows:

  $X4(i) = X0(i) \oplus X1(i) \oplus X2(i) \oplus X3(i)$; Now, the total number of 1s is even.

  Examples (4-bit data, 1 even parity bit):

  0000 0; 0001 1; 1001 0; 1110 1; 1111 0
- Normally, parity checking can only <u>detect a single error</u> but not correct it.

  However, in the case of a disk error, we can reasonably expect to know which disk has failed, and hence which bit. Data can be reconstructed from the remaining devices and the parity disk.

  For example, suppose that drive X1 has failed:

  If we add $X4(i) \oplus X1(i)$ to both sides of the preceding equation, we get

  $X1(i) = X0(i) \oplus X2(i) \oplus X3(i) \oplus X4(i)$

  Thus, the contents of each strip of data on X1 can be regenerated from the contents of the corresponding strips on the remaining disks in the array.
- This principle is used for RAID levels 3 through 6.

RAID 3 (cont'd)

Performance:

Reminder: the spindles of the individual drives are **synchronized** (each disk head is in the same position on each disk at any given time).

Reading:

- Words in the same stripe (row) *(same track/sector)* can be read at the same time.

  For example, words b_0, b_1, b_2, and b_3 in the figure on slide 7.17 can be read in parallel.
- Words in different stripes (rows) can only be read sequentially.

  For example, to read words b_0, and b_5, two successive access operations are necessary.

  Example:

  If the access time of a disk including, one read or write operation is **ta**,

  The time to read 4 words, b_0, b_1, b_2, and b_3, is ta.

  The time to read 2 words, b_0 and b_5, is $2 \cdot$ta.

RAID 3 Performance: (cont'd)

Writing:

- Even if only one word is written, all disks are busy because to calculate parity, the other words in the same stripe (row) must be read.

  This does not create any additional problems for RAID 3 because disks are synchronized and different stripes cannot be accessed independently.

  For example, to modify word b_0, words b_1, b_2, and b_3 must be read. Since these words reside in the same location (same track/sector), these write and read operations are performed at the same time.

  Parity is calculated and then written to the parity disk.
- N words can be written in parallel to the same location (same track/sector) on different disks (for example, words b_0, b_1, b_2, and b_3 can be modified at the same time).

  The parity can be calculated in advance, then written with the data.

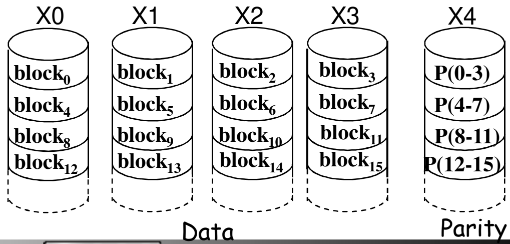
  Since disks are synchronized, there is no need to spend seek and rotation time for the parity disk.

Summary:

- Synchronized disks with small strips are suitable for large transfers (file servers).
- In a transaction-oriented environment, performance suffers.

RAID 4

- Disks operate independently (not synchronized).
  Separate I/O requests (in different stripes) can be satisfied in parallel.
- Large strips (blocks) are used.
  Suitable for applications that require high I/O request rates and relatively less suited for applications that require high data transfer rates.
- For error detection/correction, single parity bit is used. The total number of disks : N+1
- In a read operation, it is not necessary to read the parity disk.
- However, every write operation must involve the parity disk.
- Even though the independent data disks can operate in parallel, different parts of the parity disk cannot be accessed at the same time.
- There is only one parity disk. A write operation must wait for the completion of the previous write.
- **Therefore, the parity disk can become a bottleneck for write operations**.



RAID 4 (cont'd)

Write penalty:

Each time that a write occurs, the array management software must update not only the user data but also the corresponding parity bits.

Assume that X0-X3 are data disks and X4 is the parity disk.

A write is performed that only involves a strip on disk X1.

The parity for the i^{th} bit (X4′(i)) is updated as follows:

$\mathbf{X4'(i)} = X0(i) \oplus \mathbf{X1'(i)} \oplus X2(i) \oplus X3(i)$ X1′(i), X4′(i) : The updated data

For this update, 3 disks must be read (X0, X2, X3), and 2 disks must be written to (X4, X1). **All disks are occupied**.

To simplify the equation, we add the terms $\oplus$ X1(i) $\oplus$ X1(i) to the right.

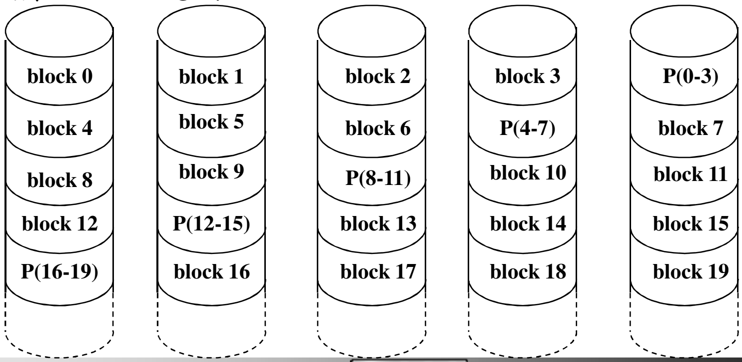
Remember XOR of any quantity with itself is 0, this does not affect the equation.

$$X4'(i) = X4(i) \oplus X1'(i) \oplus X1(i) \qquad \boxed{X4(i) = X0(i) \oplus X1(i) \oplus X2(i) \oplus X3(i)}$$

In this case, two reads and two writes are necessary.
To calculate the new parity (X4′), the array management software must read the old user strip (X1) and the old parity strip (X4).
Then, it can update these two strips with the new data (X1′) and the newly calculated parity (X4′).

RAID 5

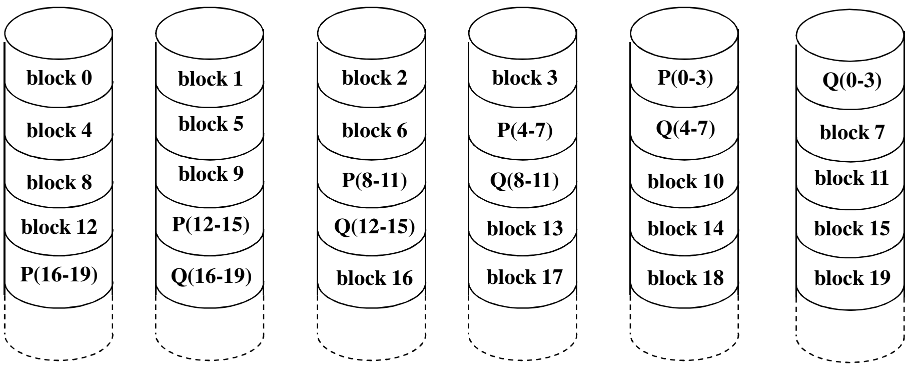
- Similar to RAID 4. Disks operate independently (not synchronized).
- Large strips (blocks) are used. Suitable for applications that require high I/O request rates.
- For error detection/correction, a single parity bit is used. The total number of disks : N+1
- The difference is that RAID 5 distributes the parity strips across all disks.
- The distribution of parity strips across all drives avoids the potential I/O bottleneck found in RAID 4.



RAID 6

- Two different parity calculations are carried out and stored in separate blocks on different disks.

  This makes it possible to regenerate data even if two disks containing user data fail.
- A RAID 6 array whose user data require N disks consists of N+2 disks.

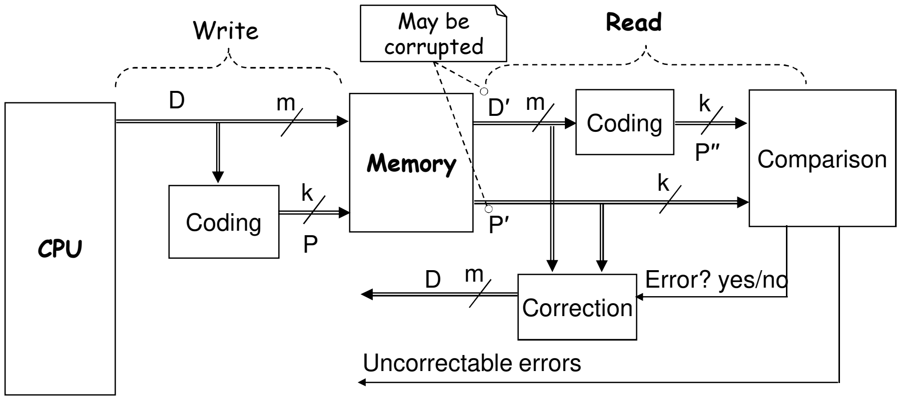


7.3 Error Detection/Correction in Memories

Hard error: Permanent physical defect

Soft error: Random, nondestructive event that alters the contents of the memory.

ECC: Error-correcting codes



D: Data, m bits

P: Parity, k bits

D': Read (received) data (may be corrupted)

P': Read (received) parity (may be corrupted)

P": Parity calculated using the data read (received)

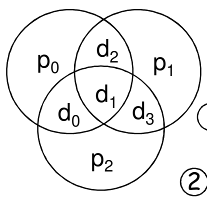
7.3.1 Single Error-Correcting Hamming Codes (*Single Error Correction – SEC*)

Parity bits are added to data bits to identify the incorrect bit.

Parity bits can be calculated in different ways.

Example: 4:7 Hamming Code (Richard Wesley Hamming (1915-1998), USA)

4-bit data + 3-bit parity. As a result, a 7-bit **code word** is transferred (stored).



d_i: data bit
p_i: parity bit

$p_0 = d_0 \oplus d_1 \oplus d_2$
$p_1 = d_1 \oplus d_2 \oplus d_3$
$p_2 = d_0 \oplus d_1 \oplus d_3$

① Transferred code word: $d_0\, d_1\, d_2\, d_3\, p_0\, p_1\, p_2$ 4 bit data + 3 bit parity

② Received (read) word: $d_0'd_1'd_2'd_3'p_0'p_1'p_2'$ may be corrupted

③ At the receiver, parity bits are calculated again:

$p_0'' = d_0' \oplus d_1' \oplus d_2'$
$p_1'' = d_1' \oplus d_2' \oplus d_3'$
$p_2'' = d_0' \oplus d_1' \oplus d_3'$

④ Received parity bits are compared to recalculated bits (XORing):

Syndrome word

$s_0 = p_0' \oplus p_0''$
$s_1 = p_1' \oplus p_1''$
$s_2 = p_2' \oplus p_2''$

⑤ If all syndrome bits (s_i) are zero, it means that the received word is error-free. If syndrome word is not zero, the incorrect bit is identified and corrected by inverting.

Syndrome impact table :

This table shows which syndrome bit is affected by which code bit.

	d_0	d_1	d_2	d_3	p_0	p_1	p_2
s_0	x	x	x		x		
s_1		x	x	x		x	
s_2	x	x		x			x

Syndrome Table:

s_0 s_1 s_2	**Meaning**
0 0 0	No error
0 0 1	p_2 (incorrect)
0 1 0	p_1
0 1 1	d_3
1 0 0	p_0
1 0 1	d_0
1 1 0	d_2
1 1 1	d_1

Determining the number of parity bits:

Number of data bits: m

Number parity bits: k

If we have k parity bits, then the syndrome word is also k bits wide and has a range between 0 and $2^k - 1$.

The value 0 indicates that no error was detected.

Remaining $2^k - 1$ values indicate which bit was in error.

Since an error can occur on any of the m data bits or k parity bits, we must have:

$$\mathbf{m + k \le 2^k - 1} .$$

7.3.2 Single-Error Correcting, Double-Error Detecting (SEC-DED) Codes

The previous code was single-error correcting code (SEC).

To add the "double-error detection" capability, we add an extra parity bit to each code word.

This extra parity bit can be calculated so that the total number of all 1s in the code word is odd (odd parity) or even (even parity).

Transmitted code word: $d_0\, d_1\, d_2\, d_3\, p_0\, p_1\, p_2\, q$ 4 + 3 + 1 bits

d: Data , p: Error-correcting parities, q: Odd/even parity

At the receiving end, if the syndrome word is not zero (there is an error), but the extra parity bit indicates "no error", then there must be two errors in the code word.

Double-errors cannot be corrected by this scheme, but at least, corrupt data can be discarded.

The most common SEC-DED coding scheme is 64 + 7 + 1 coding.

This coding system has 12.5% redundancy.