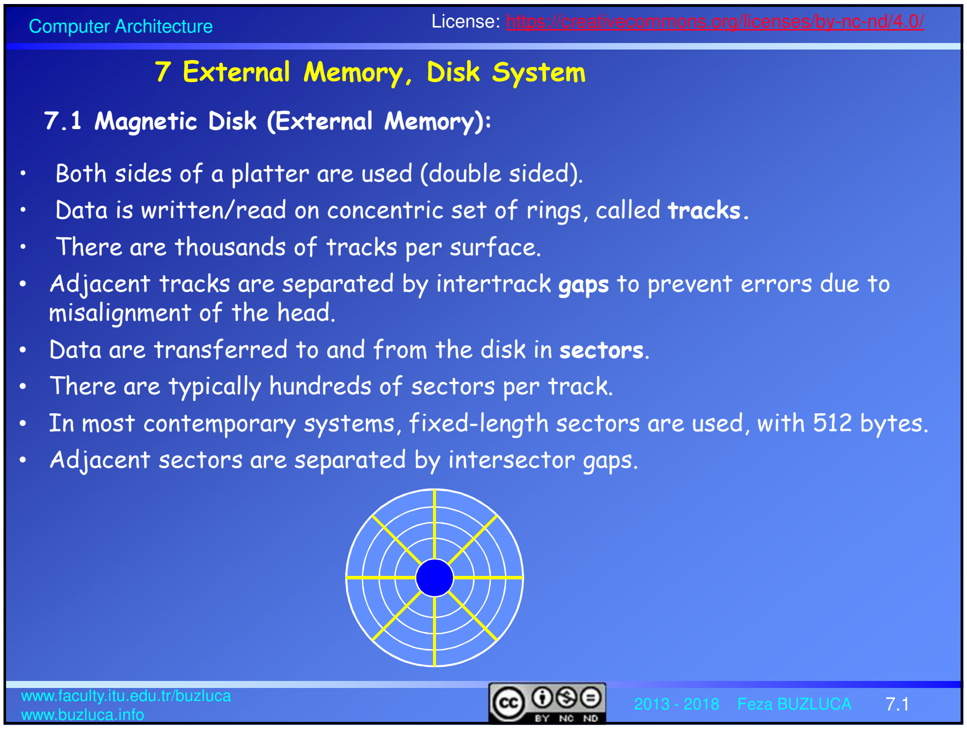
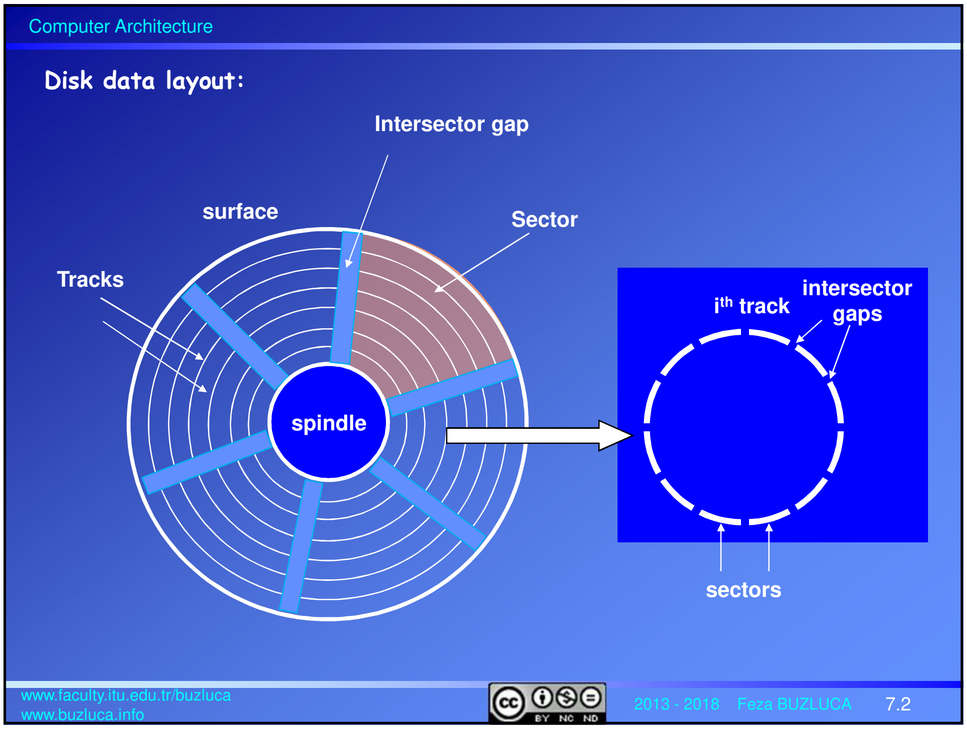


7 External Memory, Disk System

7.1 Magnetic Disk (External Memory):

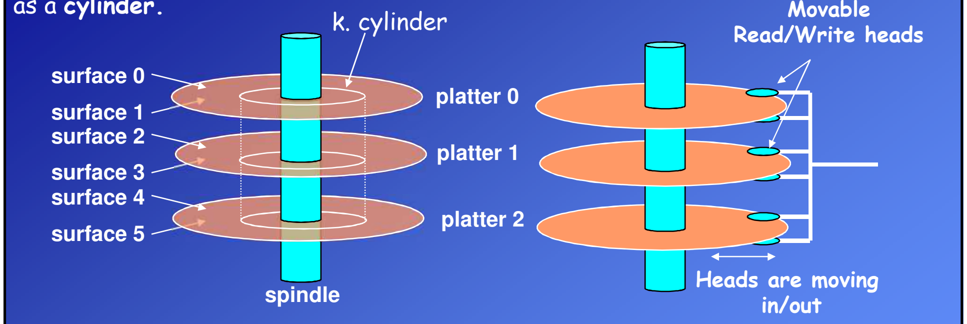
- Both sides of a platter are used (double sided).
- Data is written/read on concentric set of rings, called **tracks**.
- There are thousands of tracks per surface.
- Adjacent tracks are separated by intertrack **gaps** to prevent errors due to misalignment of the head.
- Data are transferred to and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

Example:

512 byte/sector
300 sector/track (average)
20,000 track/surface
2 surface/platter
5 platter/disk

Capacity= 512 x 300 x 20000 x 2 x 5
= 30,720,000,000 byte
≈ 28.6 GB

Disk Performance

The average access time of 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. Average 9ms (3-15ms)

- **Average rotational latency Tr:**

The time it takes for the beginning of the sector to reach the head.

After the head was 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 (one whole revolution) of the platter:

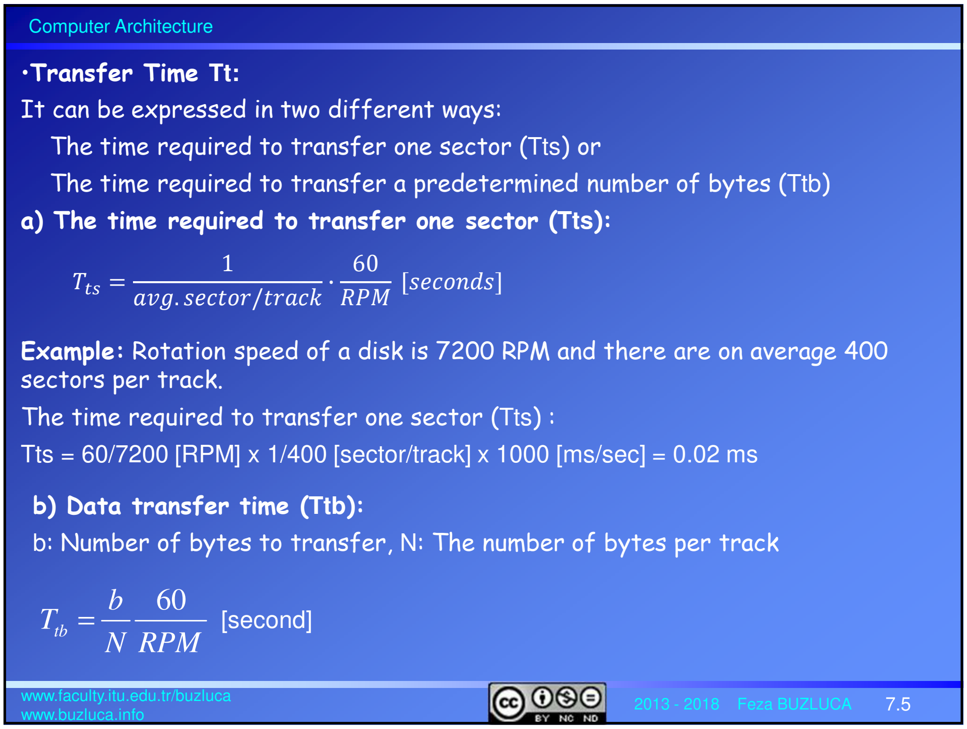
$$T_r = \frac{1}{2} r$$ r: Rotation time of a disk (in seconds)

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 rotates one whole revolution in 8.3 ms. Hence the average rotational latency is 4ms.

10000 rpm: 3ms, 15000 rpm: 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: Rotation speed of a disk is 7200 RPM and there are on average 400 sectors per track.

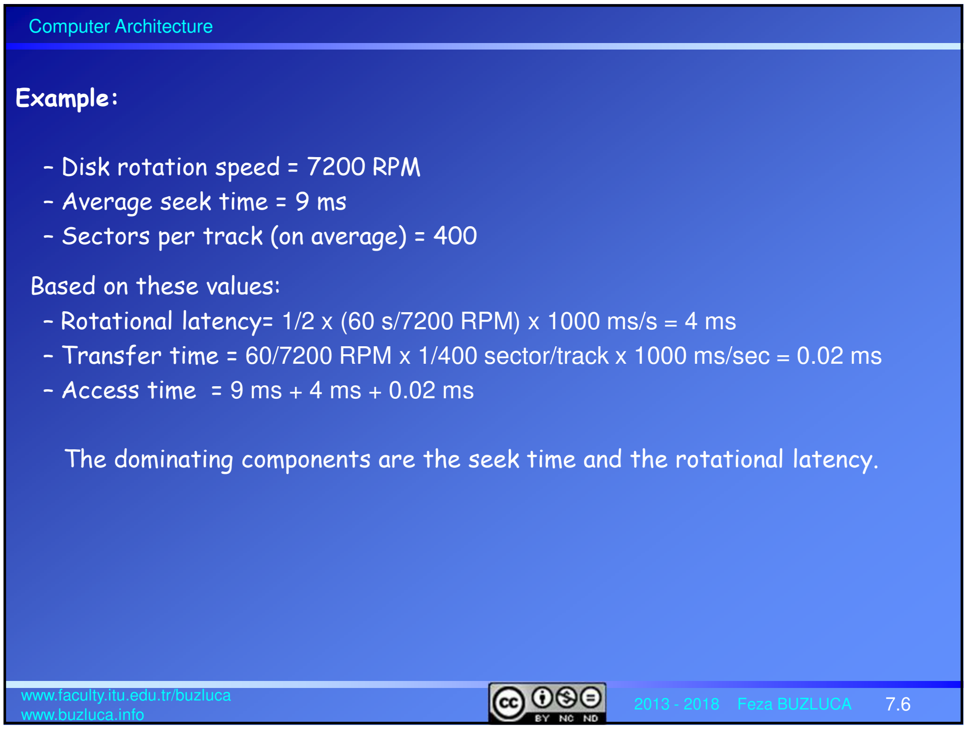
The time required to transfer one sector (Tts) :

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

b) Data transfer time (Ttb):

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

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



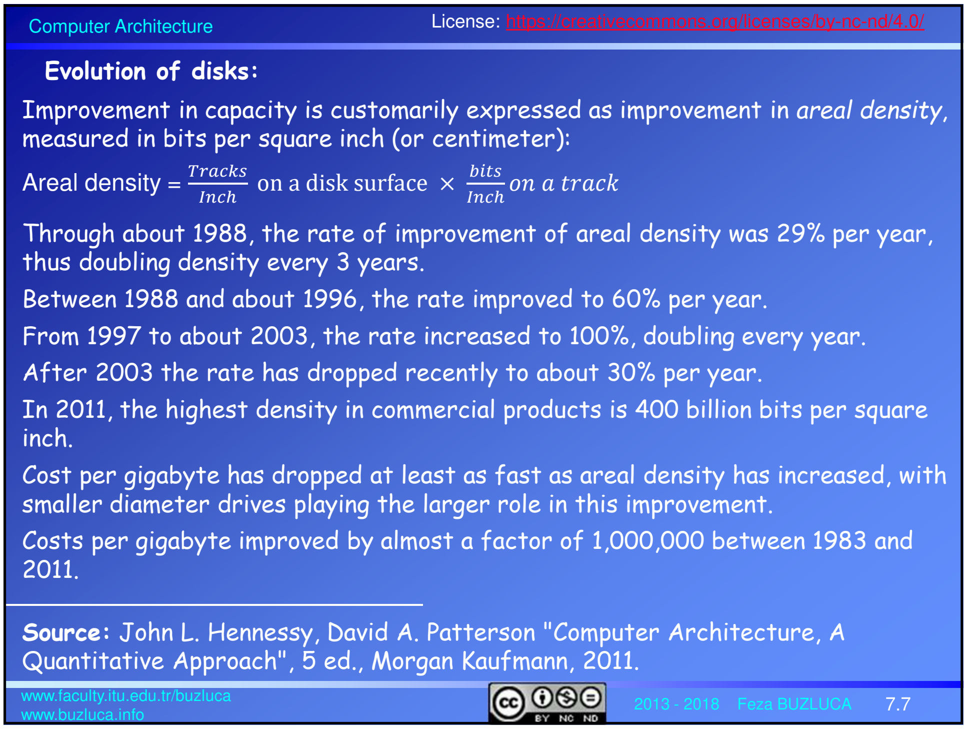
Example:

- 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 (60 s/7200 RPM) x 1000 ms/s = 4 ms
- Transfer time = 60/7200 RPM x 1/400 sector/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 × $\frac{bits}{Inch}$ *on a track*

Through about 1988, the rate of improvement of areal density was 29% per year, thus doubling density every 3 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.

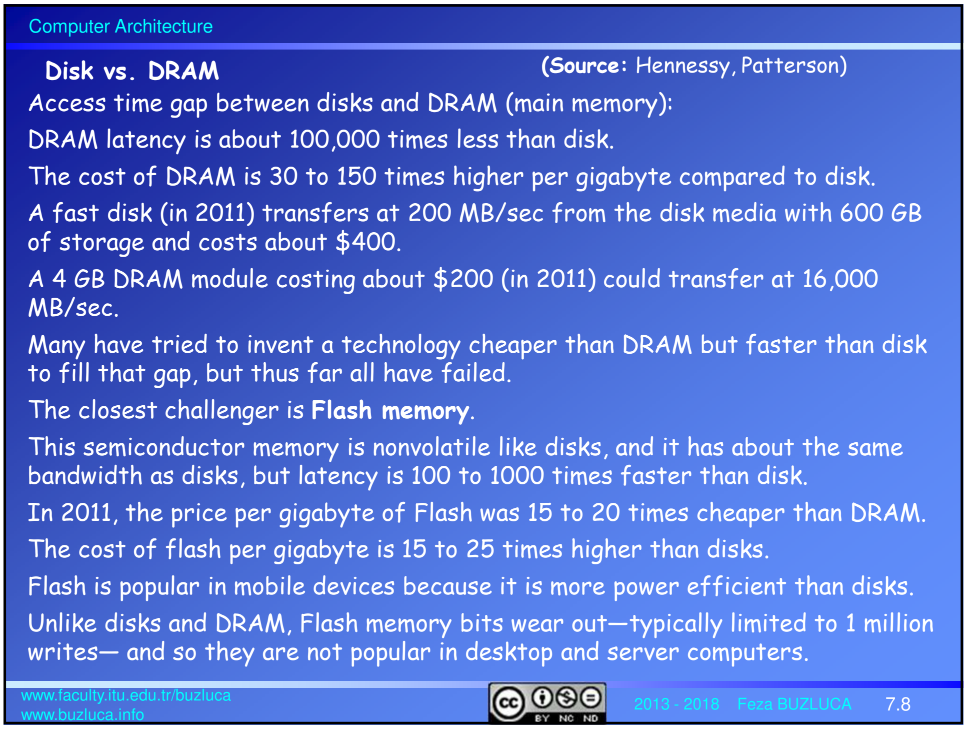
After 2003 the rate has dropped recently to about 30% per year.

In 2011, the highest density in commercial products is 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", 5 ed., Morgan Kaufmann, 2011.



Disk vs. DRAM

(Source: Hennessy, Patterson)

Access time gap between disks and DRAM (main memory):

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

The cost of DRAM is 30 to 150 times higher per gigabyte compared to disk.

A fast disk (in 2011) transfers at 200 MB/sec from the disk media with 600 GB of storage and costs about $400.

A 4 GB DRAM module costing about $200 (in 2011) could transfer at 16,000 MB/sec.

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**.

This semiconductor memory is nonvolatile like disks, and it has about the same bandwidth as disks, but latency is 100 to 1000 times faster than disk.

In 2011, the price per gigabyte of Flash was 15 to 20 times cheaper than DRAM.

The cost of flash per gigabyte is 15 to 25 times higher than disks.

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

Unlike disks and DRAM, Flash memory bits wear out—typically limited to 1 million writes— and so they are not popular in desktop and server computers.

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

Data are distributed across the physical drives of an array.

Purpose:

Increasing the **performance** and the **reliability**.

- Parallel independent disks increase the performance.
- 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:

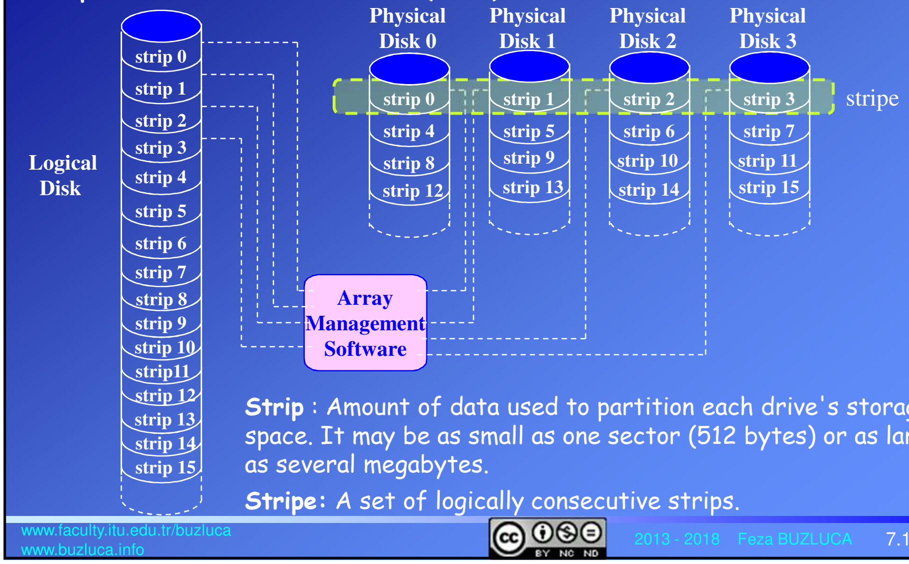
1. There are many physical disk drives viewed by the operating system as a single logical drive.
2. Logical sequential data are distributed across the physical drives of an array in a scheme known as striping.
3. Redundant disk capacity is used to store **parity** information.
   In case of a disk **failure** with the help of the parity data can be recovered.

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

RAID 0

No redundancy (parity). Data cannot be recovered. Not a real RAID method.
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)

Increasing the throughput:

- 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 parallel data disks).
- 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 the strip size on the performance

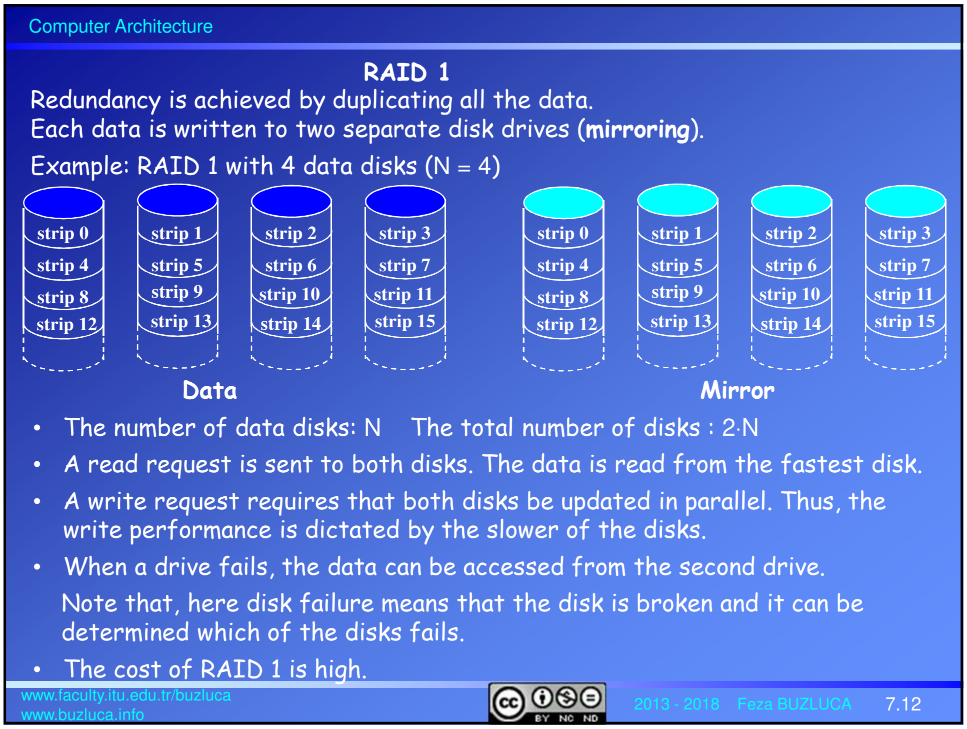
a) If the typical request is for **large amounts of logically contiguous data**, compared to the size of a strip:

Transfer rate is important: Copying files or video playback.

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 data, such as 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)

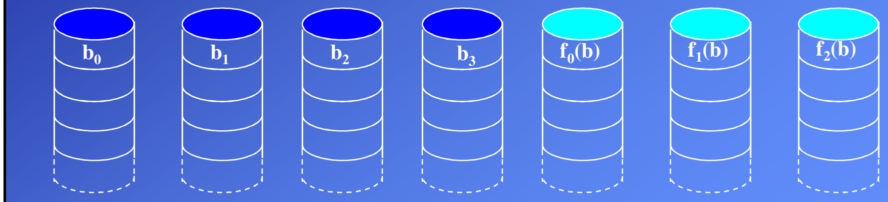
- The number of data disks: N The total number of disks : 2·N
- A read request is sent to both disks. The data is read from the fastest disk.
- A write request requires that both disks be updated in parallel. Thus, the write performance is dictated by the slower of the disks.
- When a drive fails, the data can be accessed from the second drive.

  Note that, here disk failure means that the disk is broken and it can be determined which of the disks fails.
- The cost of RAID 1 is high.

RAID 2

For data recovery, **a Hamming** code is used, which is able to correct single-bit errors and detect double-bit errors.

- Hamming code is used in memories or 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.
- As shown below, to 4-bit data, 3-bit parity is added.
- Hamming code is unnecessary for disk systems, because it is possible to physically determine which disk fails.
- Although RAID 2 requires fewer disks than RAID 1, it is still rather costly.



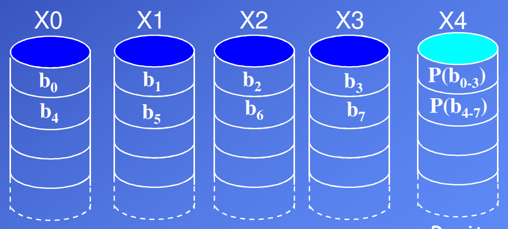
Data

Parity bits

RAID 3

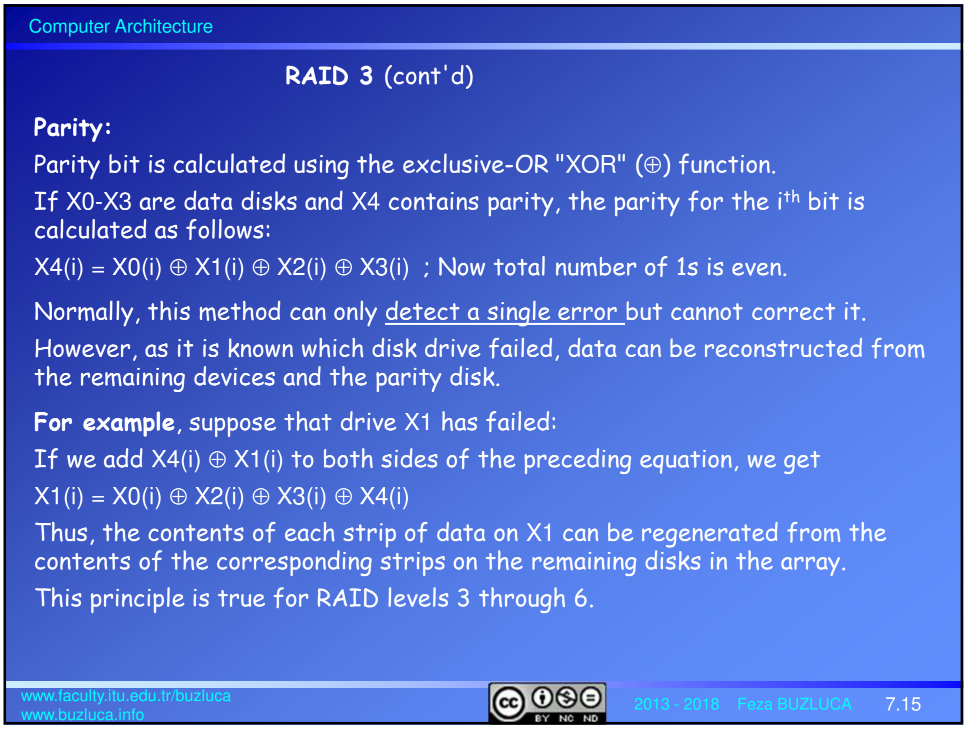
- As in RAID 2 the spindles of the individual drives are **synchronized;** each disk head is in the same position on each disk at any given time.
- **Small strips** are used. Suitable for large amounts of logically contiguous data.
- RAID 3 requires only a single redundant disk, no matter how large the 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.

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



Data

Parity



RAID 3 (cont'd)

Parity:

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

If X0-X3 are data disks and X4 contains parity, the parity for the ith bit is calculated as follows:

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

Normally, this method can only detect a single error but cannot correct it.

However, as it is known which disk drive failed, data can be reconstructed from the remaining devices and the parity disk.

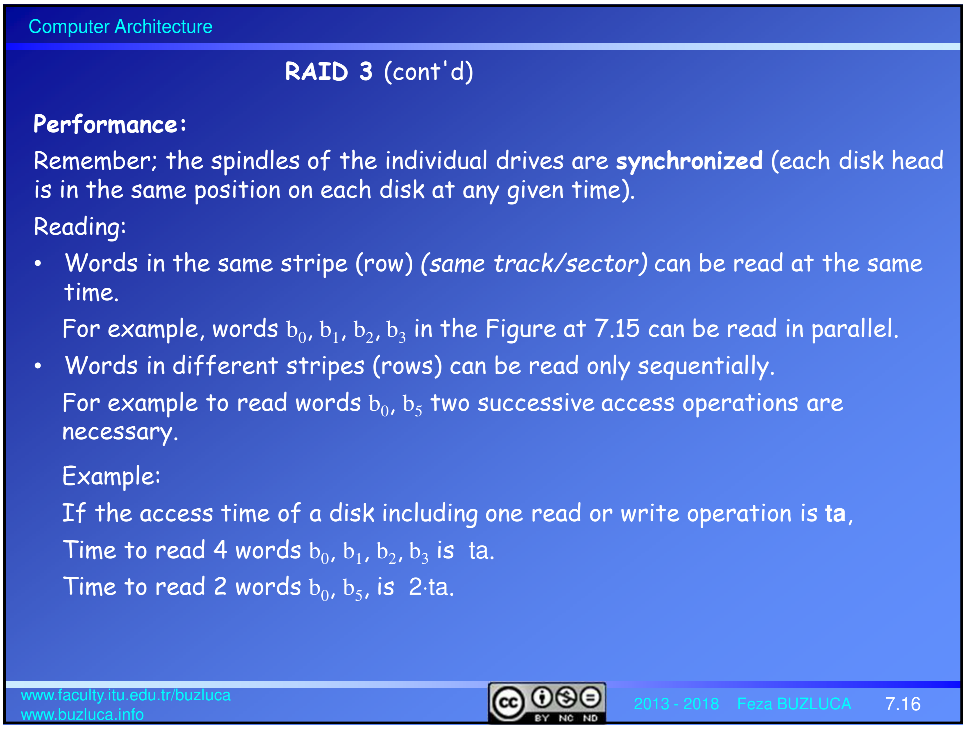
For example, suppose that drive X1 has failed:

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 true for RAID levels 3 through 6.



RAID 3 (cont'd)

Performance:

Remember; 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, b_3 in the Figure at 7.15 can be read in parallel.
- Words in different stripes (rows) can be read only sequentially.

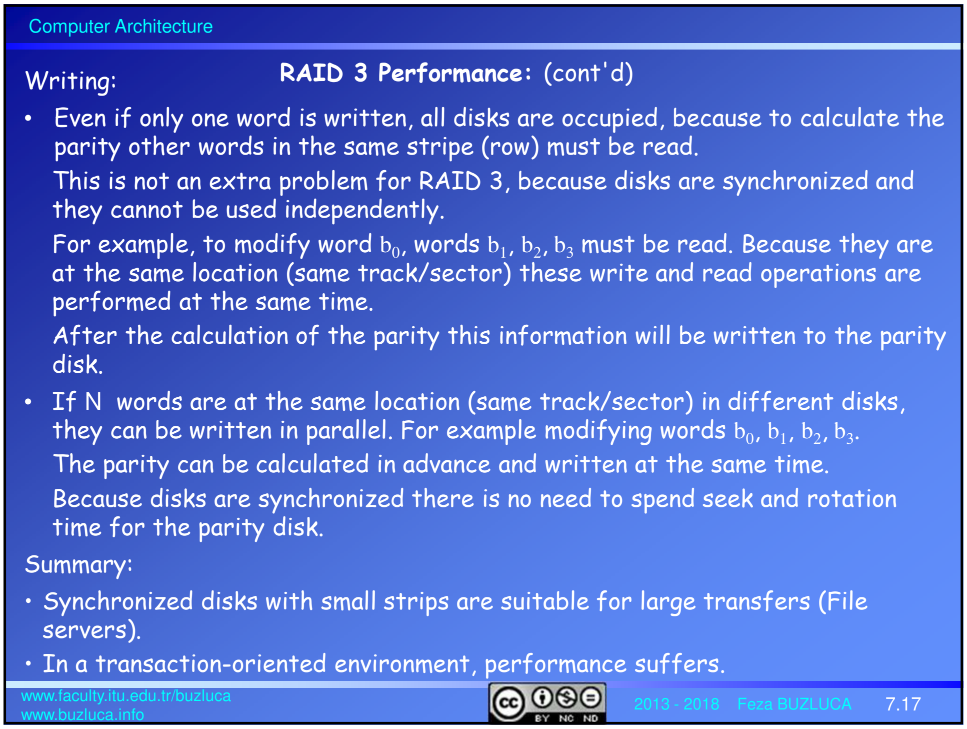
  For example to read words b_0, b_5 two successive access operations are necessary.

  Example:

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

  Time to read 4 words b_0, b_1, b_2, b_3 is ta.

  Time to read 2 words b_0, b_5, is 2·ta.



RAID 3 Performance: (cont'd)

Writing:

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

  This is not an extra problem for RAID 3, because disks are synchronized and they cannot be used independently.

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

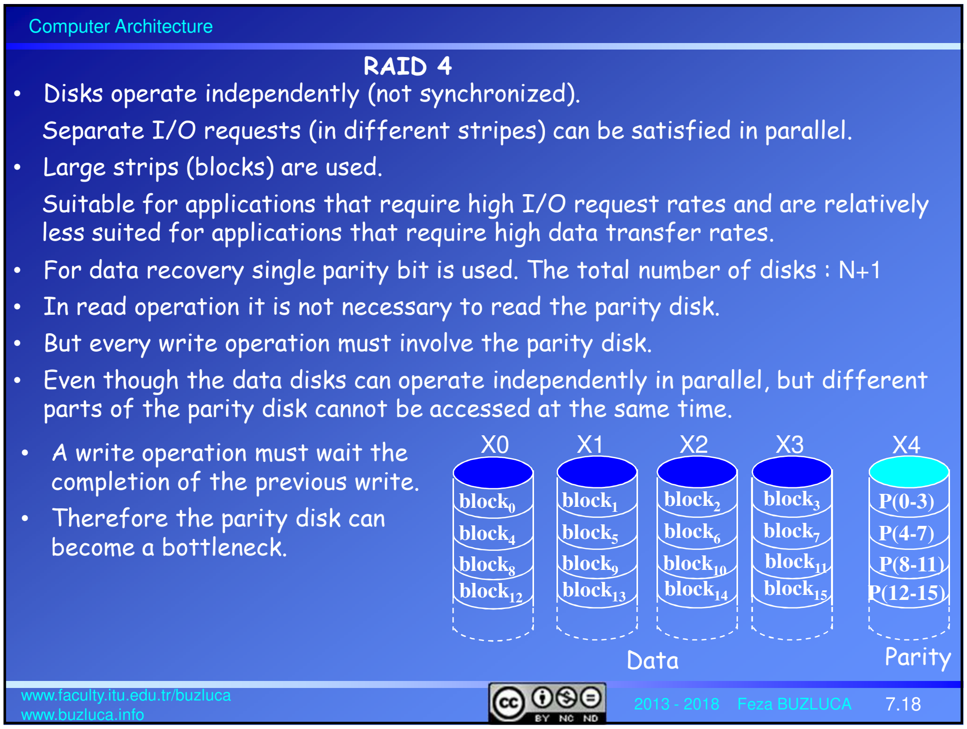
  After the calculation of the parity this information will be written to the parity disk.
- If N words are at the same location (same track/sector) in different disks, they can be written in parallel. For example modifying words b_0, b_1, b_2, b_3.

  The parity can be calculated in advance and written at the same time.

  Because 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.

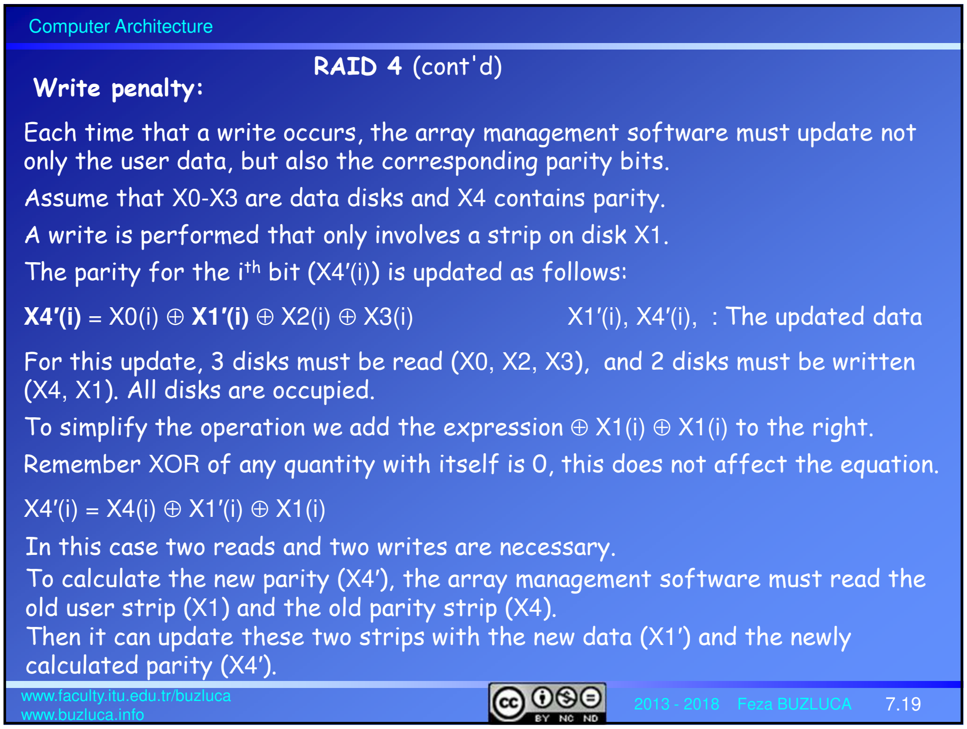


RAID 4

- 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 are relatively less suited for applications that require high data transfer rates.
- For data recovery single parity bit is used. The total number of disks : N+1
- In read operation it is not necessary to read the parity disk.
- But every write operation must involve the parity disk.
- Even though the data disks can operate independently in parallel, but different parts of the parity disk cannot be accessed at the same time.
- A write operation must wait the completion of the previous write.
- Therefore the parity disk can become a bottleneck.



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 contains parity.

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 (X4, X1). All disks are occupied.

To simplify the operation we add the expression $\oplus X1(i) \oplus X1(i)$ to the right.

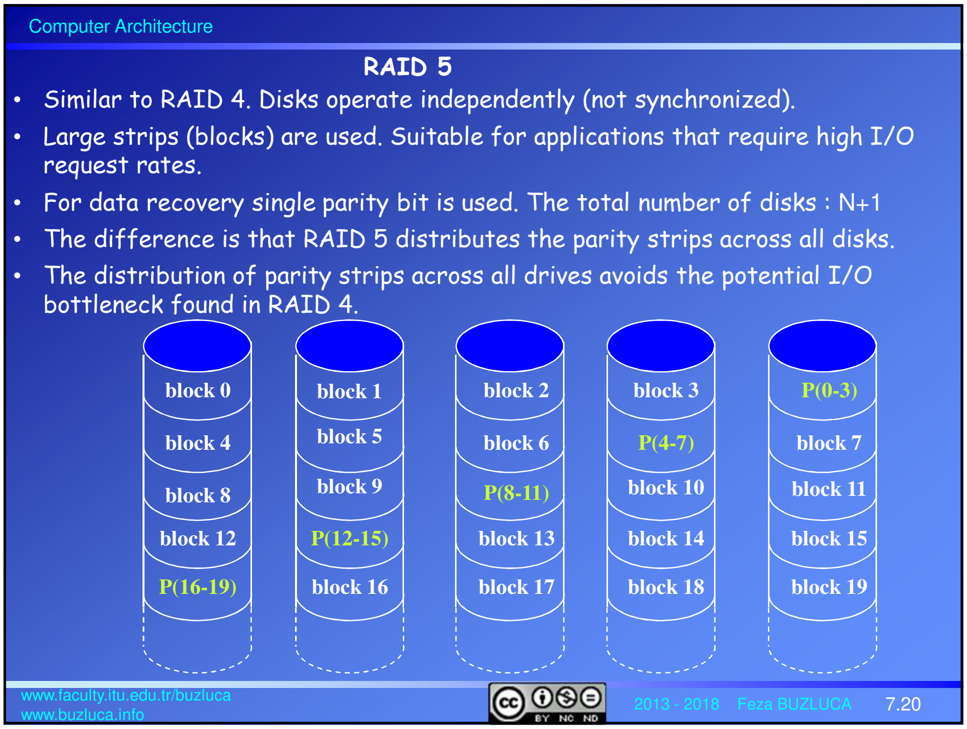
Remember XOR of any quantity with itself is 0, this does not affect the equation.

$X4'(i) = X4(i) \oplus X1'(i) \oplus X1(i)$

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).

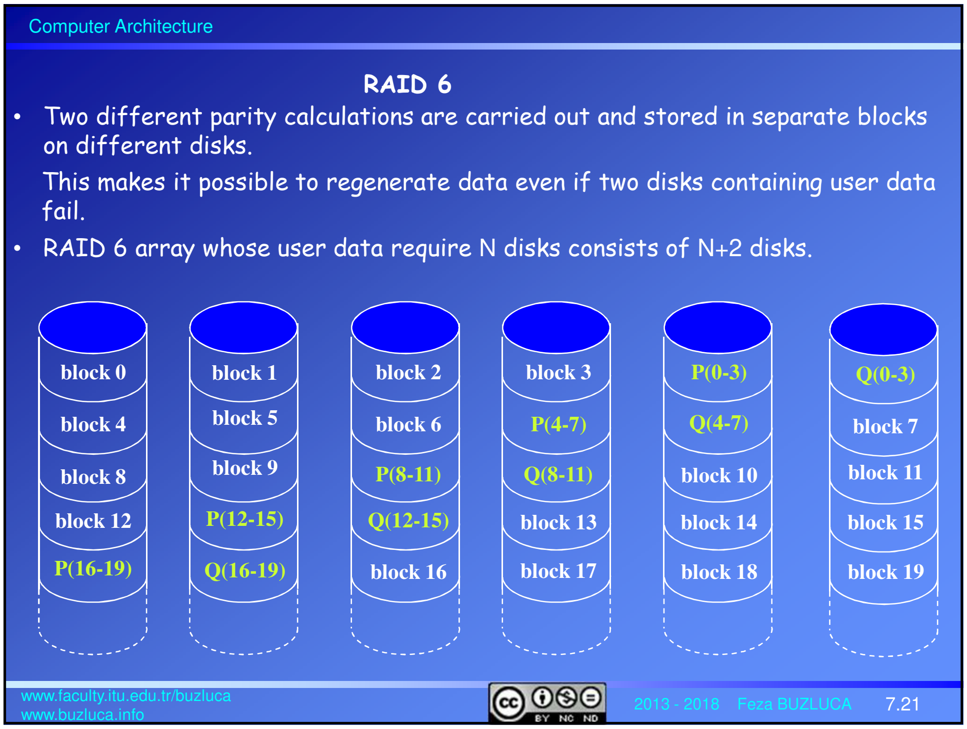
Then it can update these two strips with the new data (X1') and the newly calculated parity (X4').



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 data recovery 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.

block 0	block 1	block 2	block 3	P(0-3)
block 4	block 5	block 6	P(4-7)	block 7
block 8	block 9	P(8-11)	block 10	block 11
block 12	P(12-15)	block 13	block 14	block 15
P(16-19)	block 16	block 17	block 18	block 19

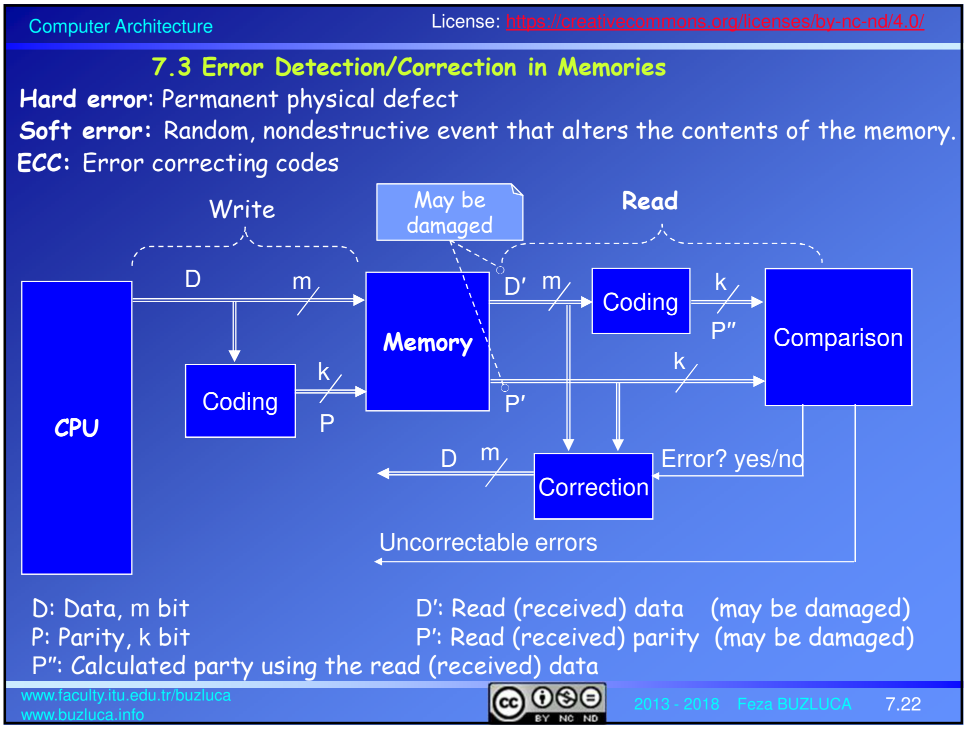


RAID 6

- 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.
- RAID 6 array whose user data require N disks consists of N+2 disks.

block 0	block 1	block 2	block 3	P(0-3)	Q(0-3)
block 4	block 5	block 6	P(4-7)	Q(4-7)	block 7
block 8	block 9	P(8-11)	Q(8-11)	block 10	block 11
block 12	P(12-15)	Q(12-15)	block 13	block 14	block 15
P(16-19)	Q(16-19)	block 16	block 17	block 18	block 19



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 bit
P: Parity, k bit
P'': Calculated party using the read (received) data

D': Read (received) data (may be damaged)
P': Read (received) parity (may be damaged)

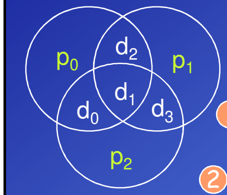
7.3.1 Single Error-Correcting (Hamming Code) (*Single Error Correction – SEC*)

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 (saved).



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 effected 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

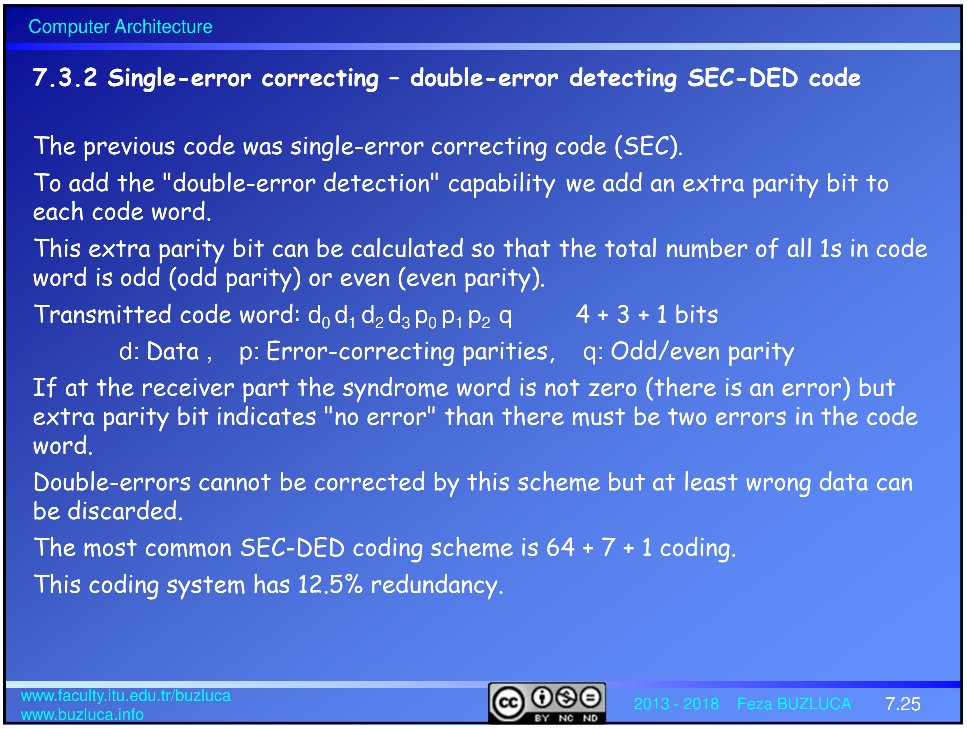
Number parity bits: k

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.

Because an error can occur on any of the m data bits or k parity bits, we must have: $\mathbf{m + k \leq 2^k - 1}$.



7.3.2 Single-error correcting – double-error detecting SEC-DED code

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 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

If at the receiver part the syndrome word is not zero (there is an error) but extra parity bit indicates "no error" than there must be two errors in the code word.

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

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

This coding system has 12.5% redundancy.