

CSE 421/521 - Operating Systems Fall 2018

LECTURE - XXI

MASS STORAGE & IO - II

Tevfik Koşar

University at Buffalo
November 15th, 2018

Storage System Reliability

- ◆ What can happen if disk loses power or machine software crashes?
 - Some operations in progress may complete
 - Some operations in progress may be lost
 - Overwrite of a block may only partially complete
- ◆ File system wants durability (as a minimum!)
 - Data previously stored can be retrieved (maybe after some recovery step), regardless of failure

1. Caching Issues

- ◆ For performance, all must be cached!
This is OK for reads but what about writes?
- ◆ Options for writing data:
 - Write-through:** write change immediately to disk
Problem: slow! Have to wait for write to complete before you go on
 - Write-back:** delay writing modified data back to disk (for example, until replaced)
Problem: can lose data on a crash!

2. Multiple Update Issues

- ◆ If multiple updates needed to perform some operations, crash can occur between them!

- Moving a file between directories:

- * Delete file from old directory
 - * Add file to new directory

- Create new file

- * Allocate space on disk for header, data
 - * Write new header to disk
 - * Add the new file to directory

What if there is a crash in the middle? Even with write-through it can still have problems

Storage Reliability Problem

- ◆ Single logical file operation can involve updates to multiple physical disk blocks
 - inode, indirect block, data block, bitmap, ...
 - With remapping, single update to physical disk block can require multiple (even lower level) updates
- ◆ At a physical level, operations complete one at a time
 - Want concurrent operations for performance
- ◆ How do we guarantee consistency regardless of when crash occurs?

Transaction Concept

- ◆ Transaction is a group of operations
 - Atomic: operations appear to happen as a group, or not at all (at logical level)
 - * At physical level, only single disk/flash write is atomic
 - Durable: operations that complete stay completed
 - * Future failures do not corrupt previously stored data
 - Isolation: other transactions do not see results of earlier transactions until they are committed
 - Consistency: sequential memory model

Transaction Implementation

- ◆ Key idea: fix problem of how you make multiple updates to disk, by turning multiple updates into a single disk write!
- ◆ Example: money transfer from account x to account y:

Begin transaction

$$x = x - a$$

$$y = y + a$$

Commit

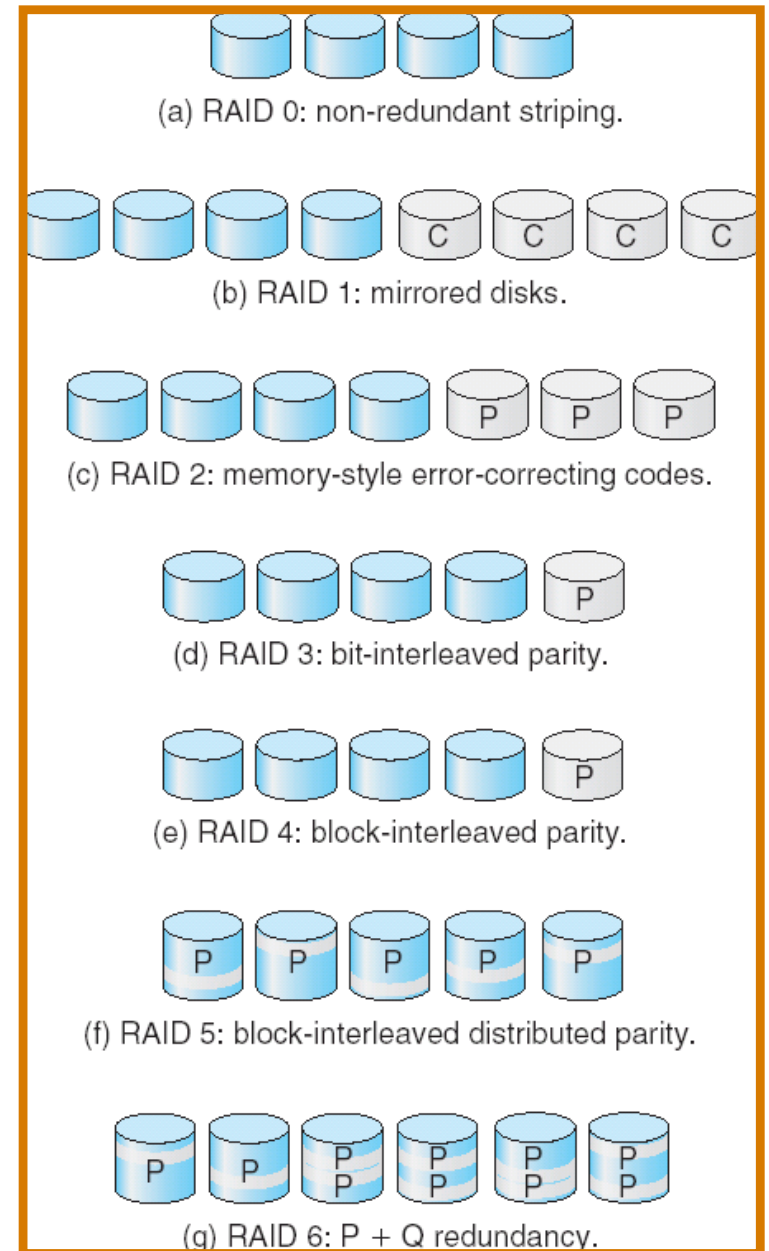
- ◆ Keep “redo” log on disk of all changes in transaction.
 - A log is like a journal, never erased, record of everything you’ve done
 - Once both changes are on log, transaction is committed.
 - Then can “write behind” changes to disk --- if crash after commit, replay log to make sure updates get to disk

RAID Concept

- As disks get cheaper, adding multiple disks to the same system provides increased **storage space**, as well as increased **reliability** and **performance**.
- RAID: Redundant Array of Inexpensive Disks
 - multiple disk drives provides reliability via redundancy.
- RAID is arranged into seven standard levels.
 - (RAID 0 -- 6)

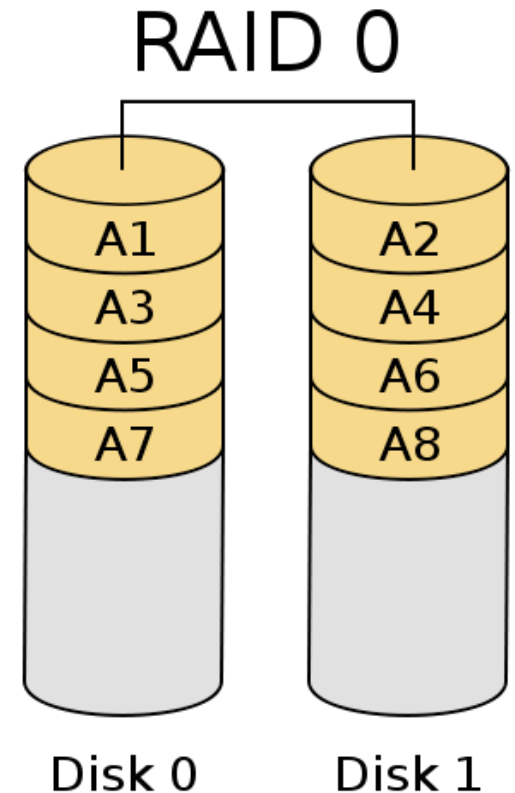
RAID (cont)

- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data.
 - **Data Striping**: splitting each bit (or block) of a file across multiple disks.
 - **Mirroring (shadowing)**: duplicate each disk
 - Simplest but most expensive approach
 - **Block interleaved parity** uses much less redundancy.



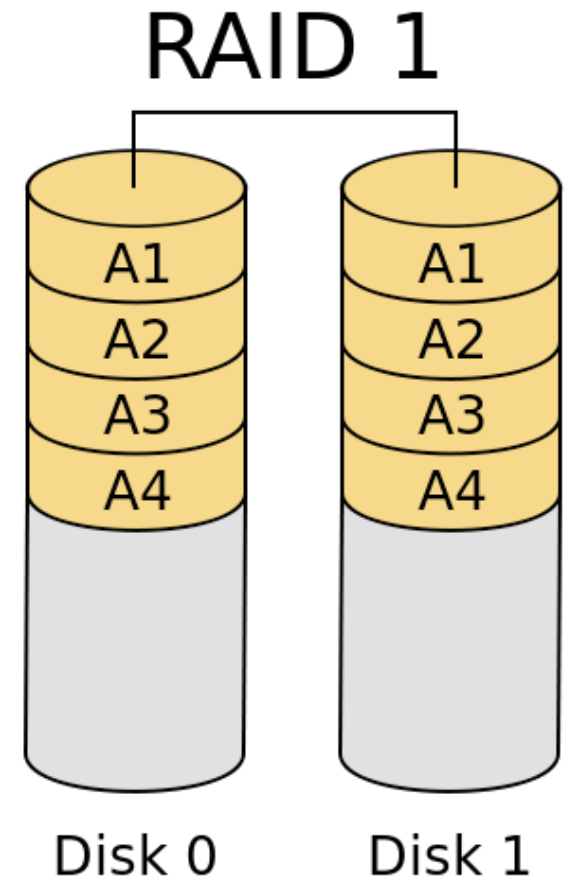
RAID Level 0

- Data is divided into blocks and is spread in a fixed order among all the disks in the array
- also known as **disk striping**
- + improves read and write performance via parallel access
- - does not provide any fault tolerance



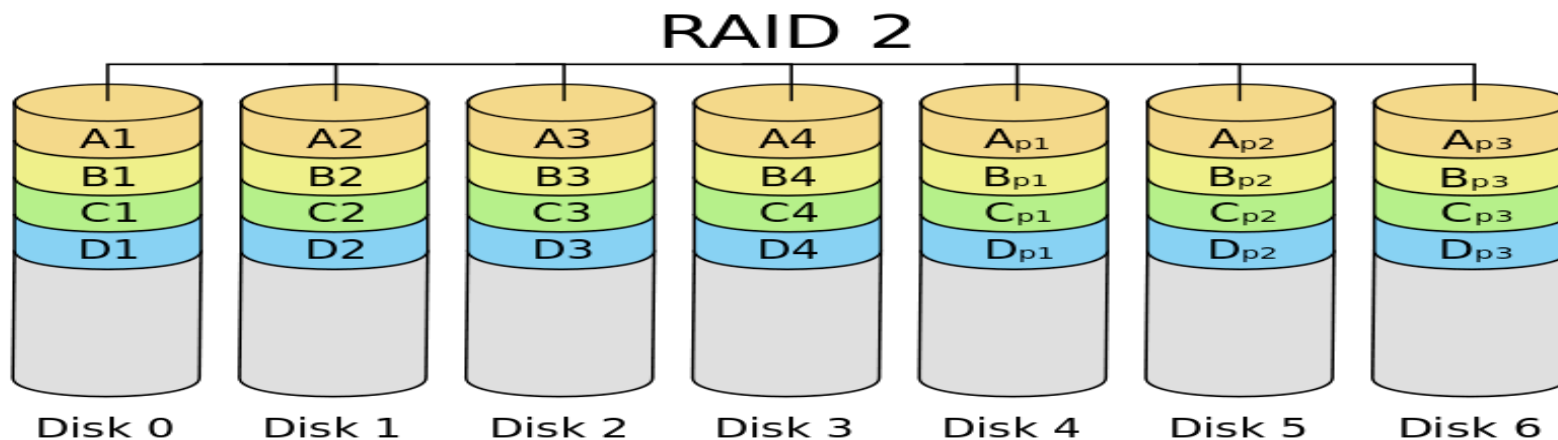
RAID Level 1

- All data written to the primary disk is written to the mirror disk
- provides a redundant, identical copy of all data
- also known as **disk mirroring**
- + provides fault tolerance
- + also generally improves read performance (but may degrade write performance)
- - doubles the storage requirement



RAID Level 2

- uses memory-style error correcting code (ECC) that employs disk-striping strategy that **breaks a file into bits** and spreads it across multiple disks
- Used Hamming Code (7,4) method requires three extra disks for four data disks



Hamming Code

- Linear error correcting code (named after its inventor - Richard Hamming)
- Hamming (7,4) -> 4 data bits, 3 parity bits
 - Can find and correct 1-bit errors
 - Can find but not correct 2-bit errors
- Hamming bits table:
 - $p1 = d1 \oplus d2 \oplus d4$
 - $p2 = d1 \oplus d3 \oplus d4$
 - $p3 = d2 \oplus d3 \oplus d4$

Bit position		1	2	3	4	5	6	7	
Encoded data bits		p1	p2	d1	p3	d2	d3	d4	
Parity bit coverage	p1	X		X		X		X	...
	p2		X	X			X	X	
	p3				X	X	X	X	

Example

Byte 1011 0001

Two data blocks, 1011 and 0001.

Expand the first block to 7 bits: 1 0 1 1.

Bit 1 is 0, because $b_3 + b_5 + b_7$ is even.

Bit 2 is 1, $b_3 + b_6 + b_7$ is odd.

bit 4 is 0, because $b_5 + b_6 + b_7$ is even.

Our 7 bit block is: 0 1 1 0 0 1 1

Repeat for right block giving 1 1 0 1 0 0 1

Examples

Data bits

Data + parity bits

(0 0 0 0)	→	(0 0 0 0 0 0 0)
(0 0 0 1)	→	(1 1 0 1 0 0 1)
(0 0 1 0)	→	(0 1 0 1 0 1 0)
(0 0 1 1)	→	(1 0 0 0 0 1 1)
(0 1 0 0)	→	(1 0 0 1 1 0 0)
(0 1 0 1)	→	(0 1 0 0 1 0 1)
(0 1 1 0)	→	(1 1 0 0 1 1 0)
(0 1 1 1)	→	(0 0 0 0 1 1 1)

⋮

How to detect errors

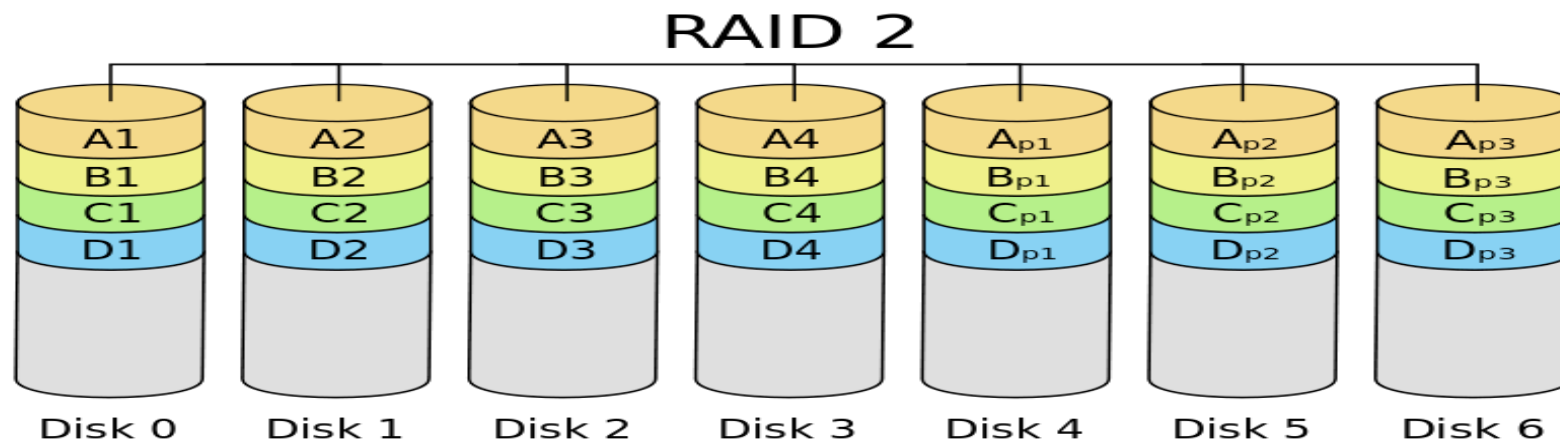
- Check the parity bits, if they do not match, they will reveal the corrupted data bit:

- $p1 \ \& \ p2 \longrightarrow d1$
- $p1 \ \& \ p3 \longrightarrow d2$
- $p2 \ \& \ p3 \longrightarrow d3$
- $p1 \ \& \ p2 \ \& \ p3 \longrightarrow d4$

Bit position		1	2	3	4	5	6	7	
Encoded data bits		p1	p2	d1	p3	d2	d3	d4	
Parity bit coverage	p1	X		X		X		X	...
	p2		X	X			X	X	
	p3				X	X	X	X	

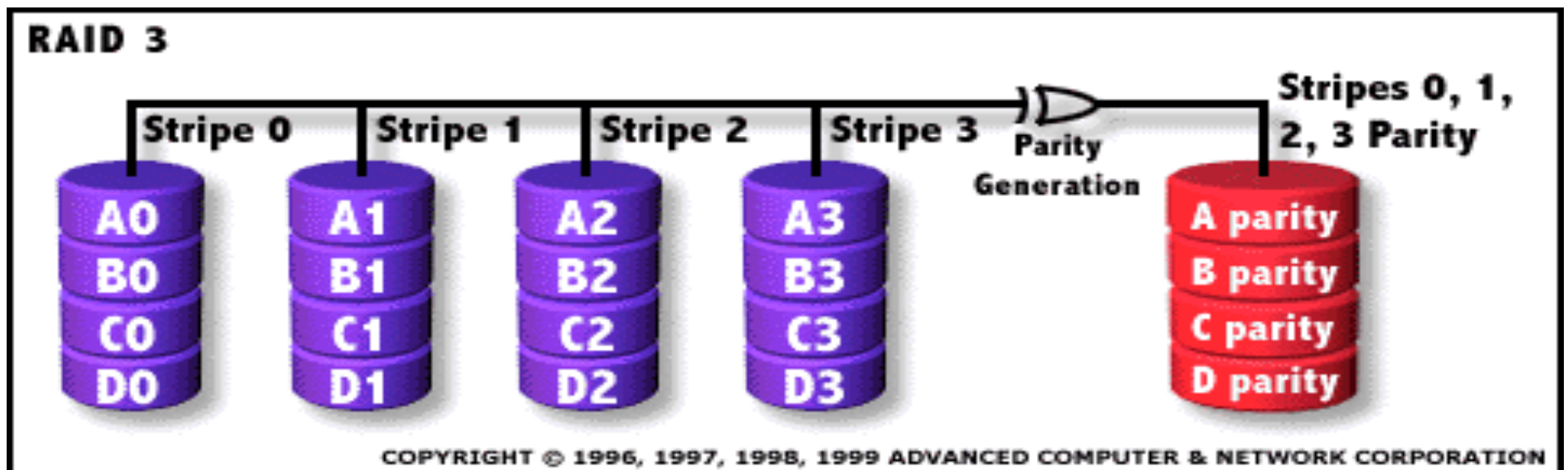
RAID Level 2 - wrap up

- + It provides fault tolerance
- - Commercially one of the most inefficient solutions
- - Overhead of Hamming code calculation slows down data access rates
- - Almost doubles the storage requirement



RAID Level 3

- it uses **byte-level** striping
- **+** requires only one disk for parity for 4 data disks
- **-** suffers from a write bottleneck, because all parity data is written to a single drive
- **-** RAID 2 & 3 cannot serve multiple requests simultaneously



Parity Block Calculation (using XOR)

- ◆ Parity block: $\text{Block1} \text{ xor } \text{block2} \text{ xor } \text{block3} \dots$

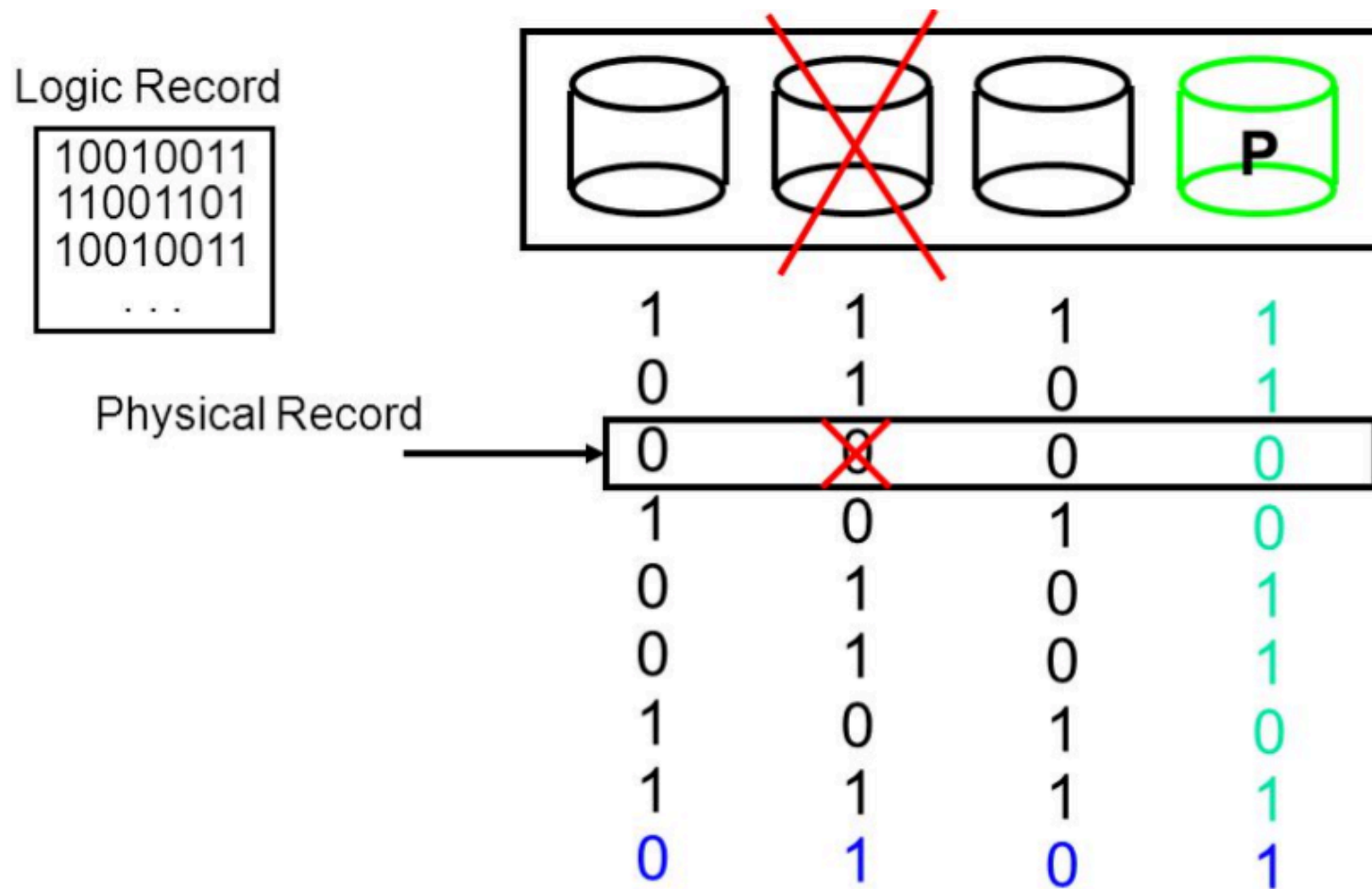
10001101	block1
01101100	block2
11000110	block3

00100111	parity block

- ◆ Can reconstruct any missing block from the others

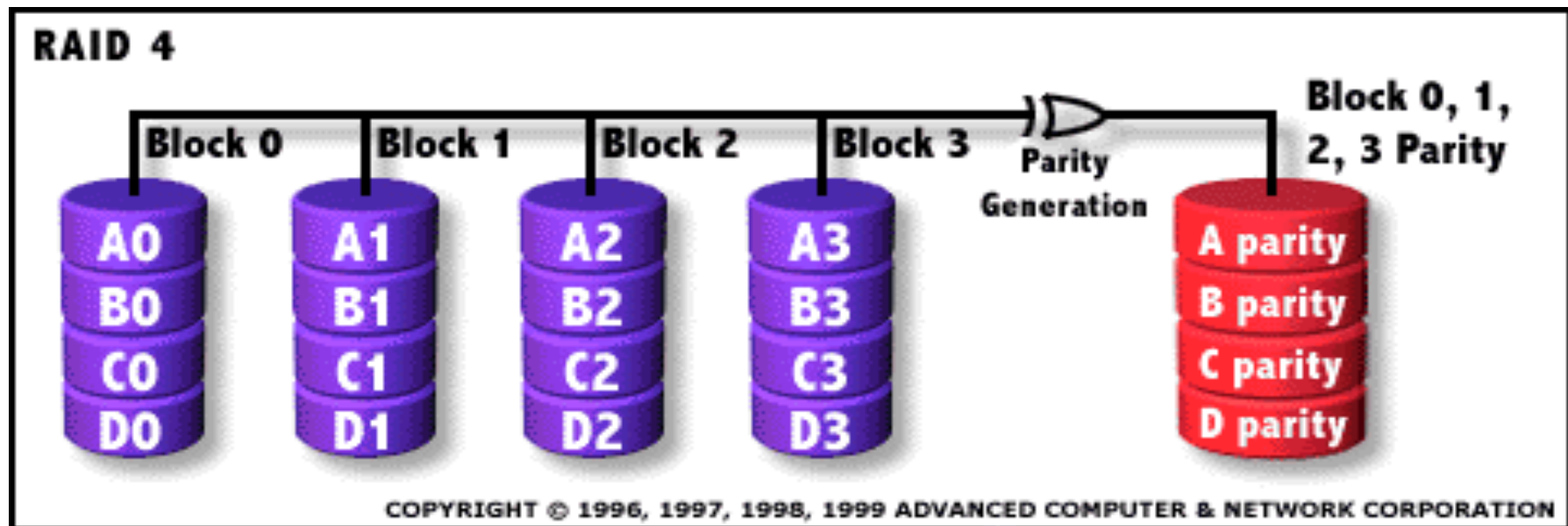
RAID 3 - Example

- ECC is not required, since the controller knows which disk has failed. So parity is enough to recover the failing disk.



RAID Level 4

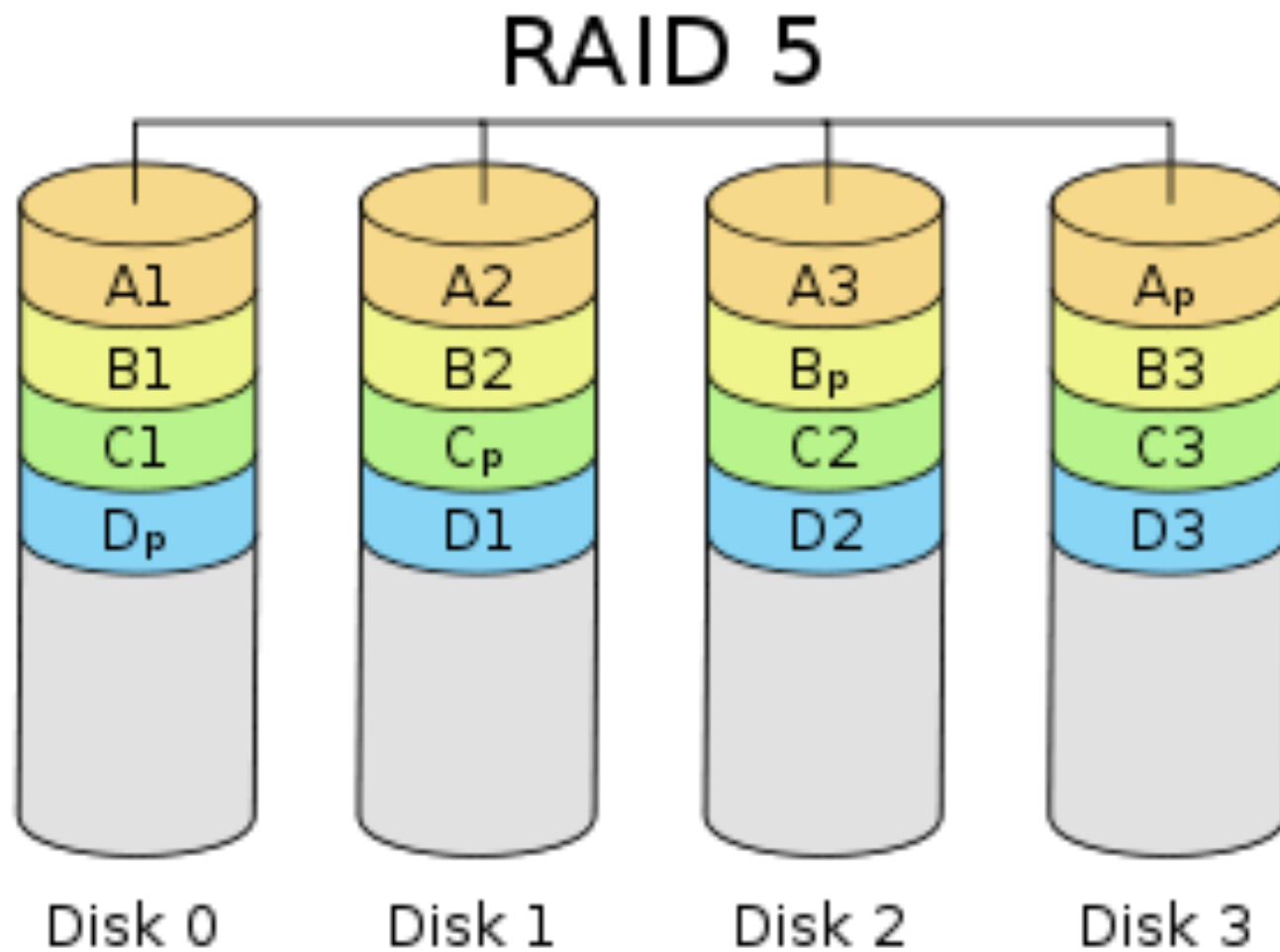
- Similar to RAID level 3, but it employs striped data in much larger **blocks** or segments
- - RAID level 4 suffers from a write bottleneck (due to parity disk).
- - Not as efficient as RAID level 5, because (as in RAID level 3) all parity data is written to a single drive



RAID Level 5

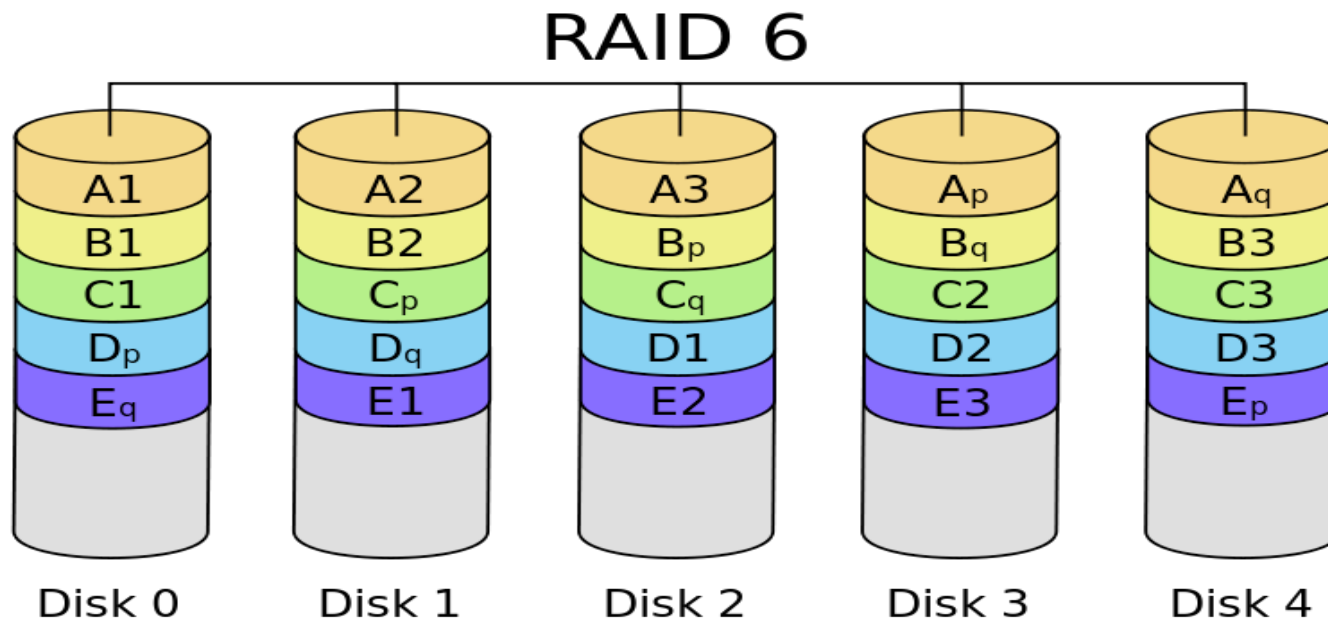
- known as **striping with parity**
- the most popular RAID level, replaced RAID 3 & 4
- similar to level 4 in that it stripes the data in large **blocks** across all the disks in the array
- It differs in that it **writes the parity across all the disks**
- The data redundancy is provided by the parity information
- The data and parity information are arranged on the disk array so that the two are always on different disks

RAID Level 5



RAID Level 6

- stores extra redundant info to recover from multiple disk failures
- would need 2 additional disks for each 4 data disks
 - more reliability versus less data space
- uses **Reed-Solomon** error correcting code



RAID Level 6 - dual parity

Drive 1	Drive 2	Drive 3	Drive 4	Drive 5
D0	D1	D2	P0	Q0
D3	D4	P1	Q1	D5
D6	P2	Q2	D7	D8
P3	Q3	D9	D10	D11

⋮ (Dx: Data Block; Px: Parity of data; Qx: Q parity of data) ⋮

$$P0 = D0 \text{ XOR } D1 \text{ XOR } D2;$$

$$P1 = D3 \text{ XOR } D4 \text{ XOR } D5;$$

$$P2 = D6 \text{ XOR } D7 \text{ XOR } D8;$$

$$P3 = D9 \text{ XOR } D10 \text{ XOR } D11;$$

⋮

⋮

⋮

$$Q0 = GF(D0) \text{ XOR } GF(D1) \text{ XOR } GF(D2);$$

$$Q1 = GF(D3) \text{ XOR } GF(D4) \text{ XOR } GF(D5);$$

$$Q2 = GF(D6) \text{ XOR } GF(D7) \text{ XOR } GF(D8);$$

$$Q3 = GF(D9) \text{ XOR } GF(D10) \text{ XOR } GF(D11);$$

⋮

⋮

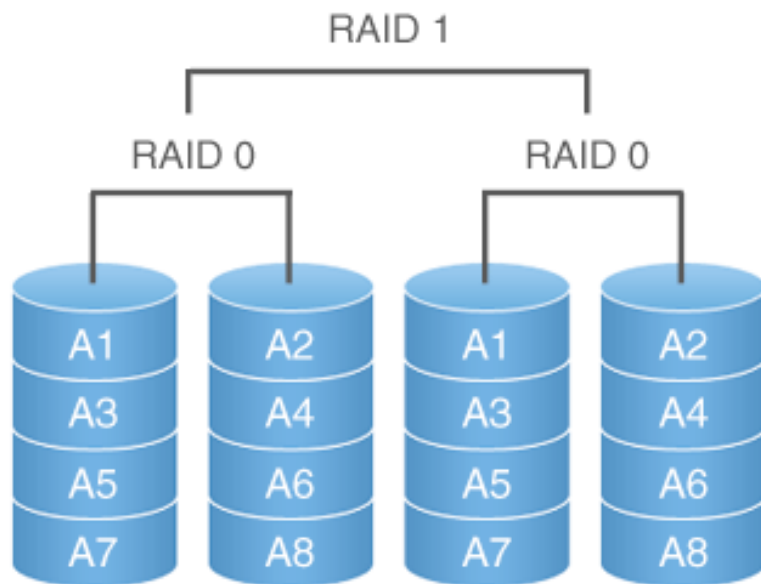
⋮

**GF() is Reed-Solomon transformation function;*

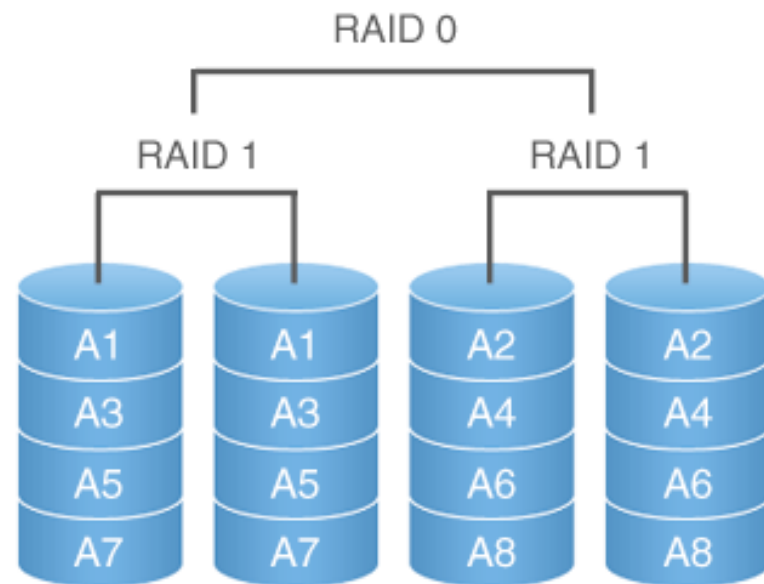
RAID (0+1) and (1+0)

- Combination of RAID 0 & 1
- better performance & reliability, but doubles the disk storage requirement

RAID 01 (0+1)



RAID 10 (1+0)



Any Questions?



Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- Z. Shao from Yale; S. Hansen from Univ of Wisconsin; J. Hall from MSU