Dependable Systems

Design for Dependability:
Fault Detection / Fault Tolerance

Luca Cassano
luca.cassano@polimi.it
cassano.faculty.polimi.it/ds.html

Most of the material of these slides has been provided by Prof. Cristiana Bolchini, Politecnico di Milano, Italy

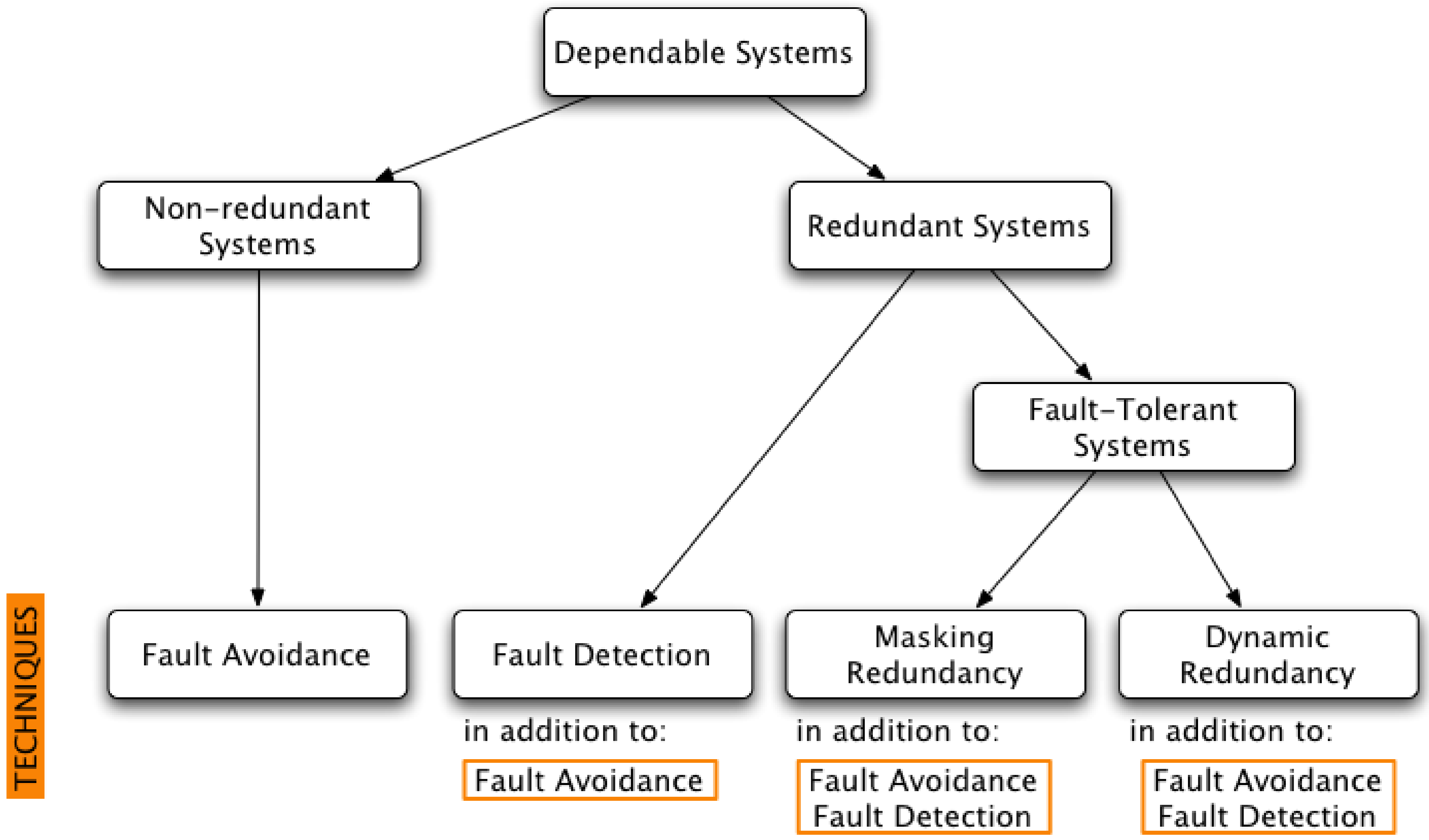
TOPIC QUESTIONS

How can we detect and handle the occurrence of faults?

What technique is more effective?

What are the costs?

Fault Management



Fault management strategies

<u>Fault avoidance</u>
preventing faults from entering the system during design phase (this should be the goal of the entire design process)

Fault management strategies

Fault avoidance

preventing faults from entering the system during design phase (this should be the goal of the entire design process)

Fault detection

Identification of fault occurrence within the operational system during normal functioning

Fault management strategies

Fault avoidance

Preventing faults from entering the system during design phase (this should be the goal of the entire design process)

Fault detection

Identification of fault occurrence within the operational system during normal functioning

Masking Redundancy (Fault tolerance)

The system is equipped with additional resources to tolerate the occurrence of faults; these resources are always on

Fault management strategies

Fault avoidance

Preventing faults from entering the system during design phase (this should be the goal of the entire design process)

Fault detection

Identification of fault occurrence within the operational system during normal functioning

Masking Redundancy (Fault tolerance)

The system is equipped with additional resources to tolerate the occurrence of faults; these resources are always on

Dynamic Redundancy (Fault tolerance)

The system is equipped with additional resources to tolerate the occurrence of faults; these resources can be switched on when needed

Considered approaches

Redundancy techniques

- Space/Hardware and time
- Information

Space/Hardware and Time Redundancy

TOPIC QUESTIONS

How can I use additional resources/processing to detect/manage faults/errors?

How do I protect processing/data from the outside?

Space Redundancy

Area (or Space or Hardware) Redundancy

Additional modules, not necessary for performing the “nominal” device functionality, are introduced

From the design point of view, the approach is the one requiring the lowest effort

Space Redundancy | 2

Passive redundancy

- fault masking

Active redundancy techniques

- detection, localization, containment, recovery

Hybrid redundancy techniques

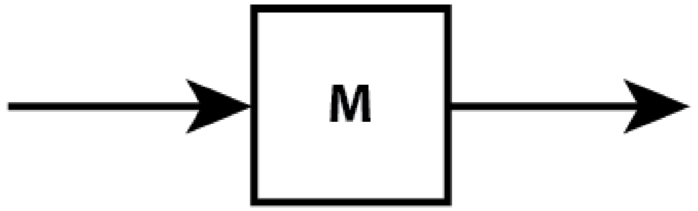
- static & dynamic
- fault masking & reconfiguration

Space Redundancy | 3

Fault Detection

Duplication With Comparison (DWC)

Two replicas of the nominal module receive the same inputs and their outputs are compared for mismatch

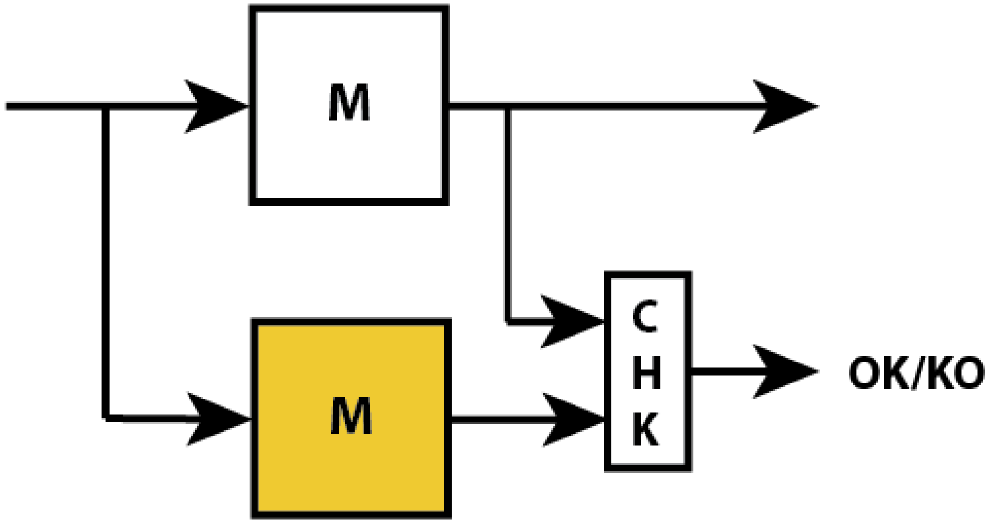


Space Redundancy | 3

Fault Detection

Duplication With Comparison (DWC)

Two replicas of the nominal module receive the same inputs and their outputs are compared for mismatch

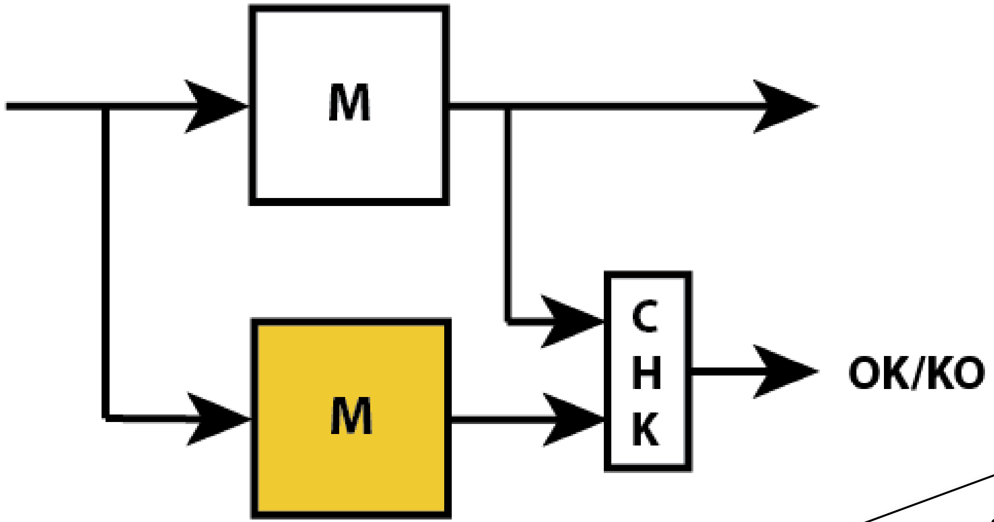


Space Redundancy | 3

Fault Detection

Duplication With Comparison (DWC)

Two replicas of the nominal module receive the same inputs and their outputs are compared for mismatch



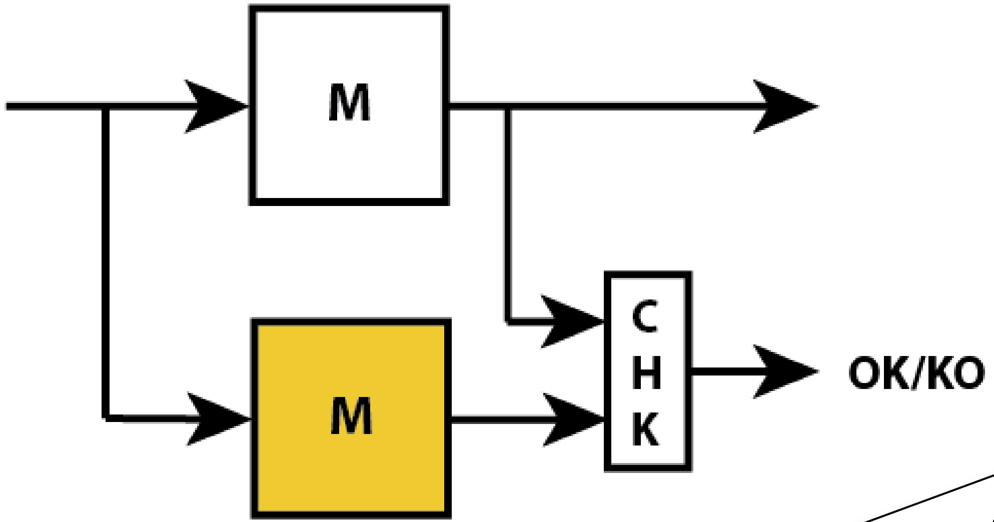
Under which assumption?

Space Redundancy | 3

Fault Detection

Duplication With Comparison (DWC)

Two replicas of the nominal module receive the same inputs and their outputs are compared for mismatch



No faults in the checker!

Space Redundancy | 4

Duplication With Comparison (DWC)

Applied at various abstraction levels:

- Module M is a circuit element (an adder, a multiplier, …)
- Module M is a processor (Dual processor architectures)

Space Redundancy | 5

Duplication With Comparison (DWC)

Costs/Benefits:

- Area: More than twice the original area
- Checker design: Two-Rail Code Checkers
- Performance: no degradation
- Effort: low

Space Redundancy | 6

Masking Redundancy

N-Modular redundancy

N replicas of the same module are fed with the same inputs and their outputs are compared and voted to produce the output

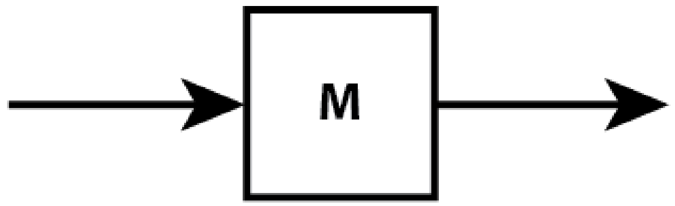
Implementation of the *M*-out-of-*N* systems

- The system survives if there are *M* out of *N* working modules

Space Redundancy | 7

N-Modular redundancy

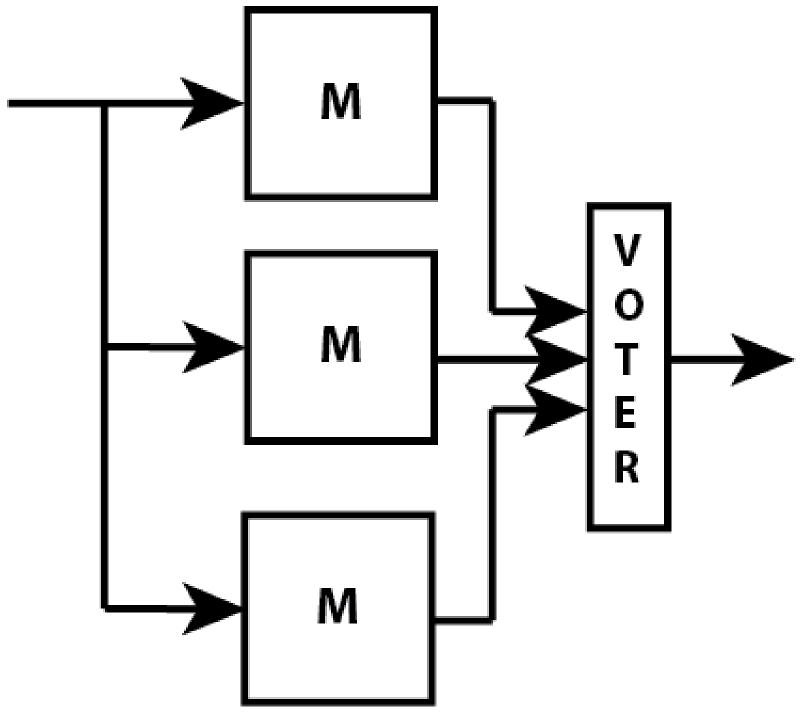
N = 3 is the smallest realization, called Triple Modular Redundancy (TMR)



Space Redundancy | 7

N-Modular redundancy

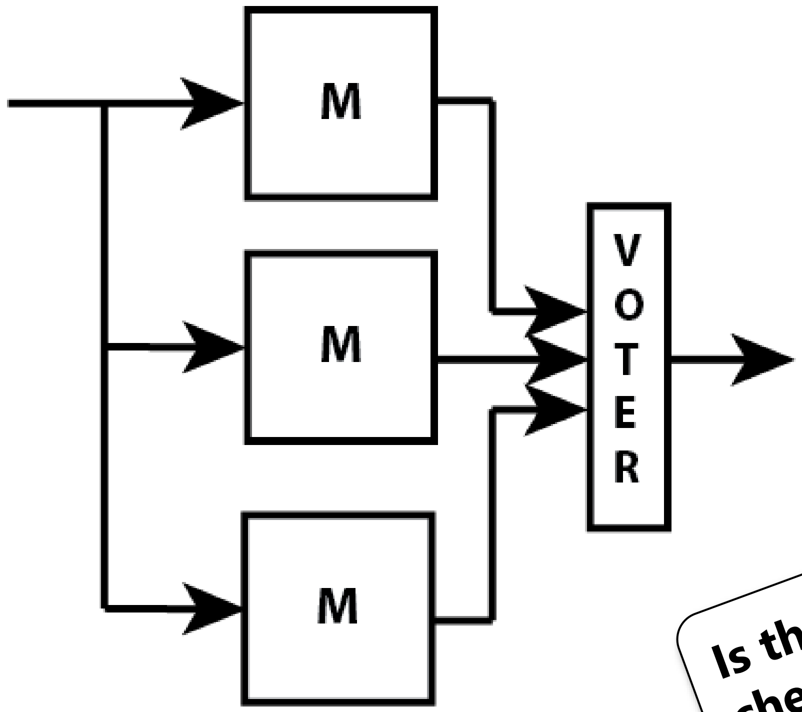
N = 3 is the smallest realization, called Triple Modular Redundancy (TMR)



Space Redundancy | 7

_N_-Modular redundancy

$N = 3$ is the smallest realization, called Triple Modular Redundancy (TMR)



Is the assumption on the checker still required?

Space Redundancy | 7

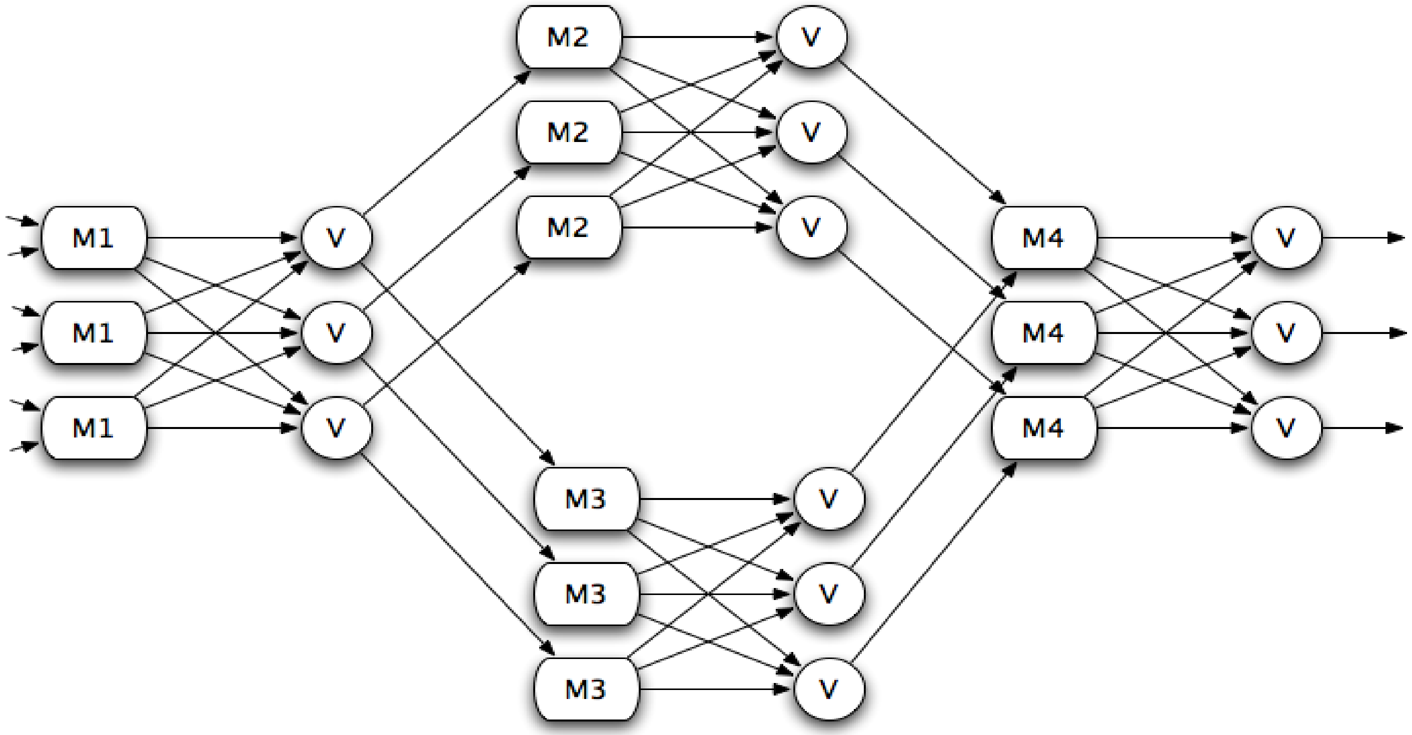
N = 3 is the smallest realization, called Triple Modular Redundancy (TMR)

$$R_{TMR}(t) = R_{voter}(t)\,(3R(t) + R^3(t) - 3R^2(t))$$

Space Redundancy | 8

N-Modular redundancy

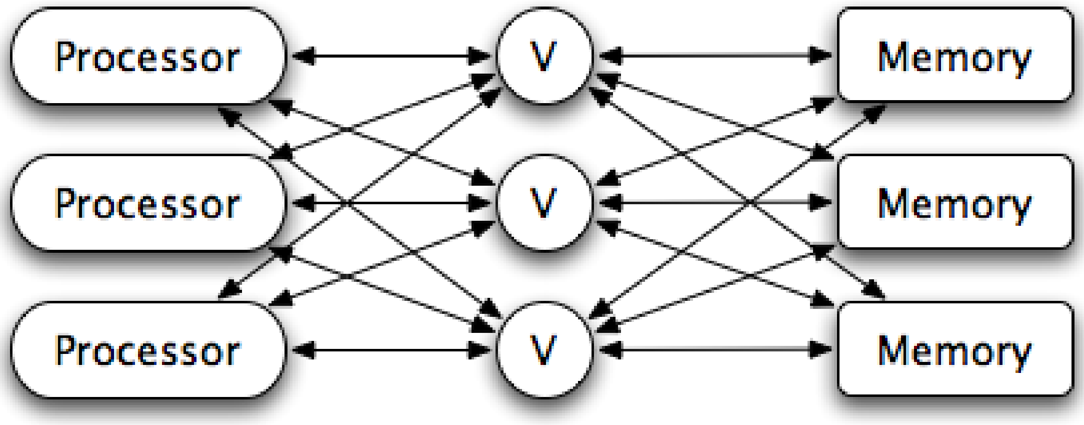
Application at system level or at module level



Space Redundancy | 8

N-Modular redundancy

Application at system level or at module level

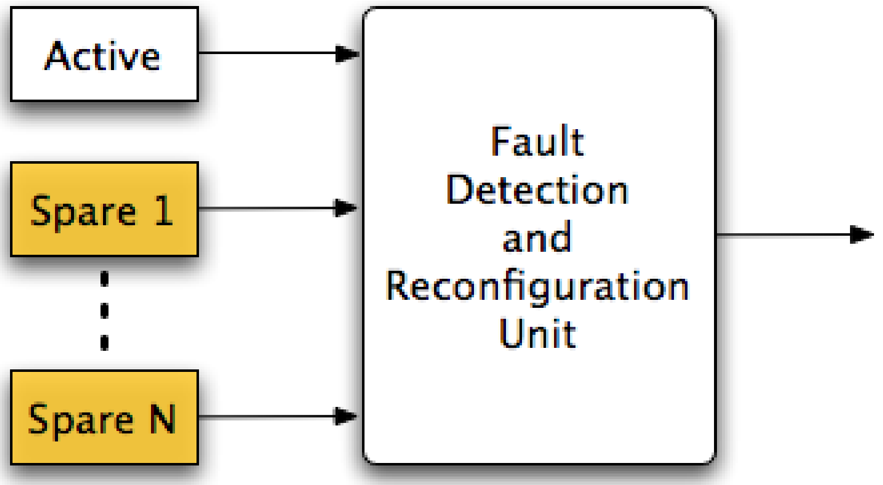


Space Redundancy | 9

Dynamic Redundancy

Standby Sparing

When a fault is detected the system reconfigures itself to use one of the redundant modules



Standby sparing

Hot standby

- all modules are powered up
- spares can be switched into use immediately after the primary module becomes failed

Cold standby

- the primary modules are powered up
- the spares are powered down, and then are powered up and switched into use when the primary modules fail

Warm standby

Standby sparing | 2

Various fault detection or error detection schemes are used to determine whether a module has become faulty

Fault location is used to determine exactly which module, if any, is faulty

Standby sparing | 3

The reconfiguration can be viewed as a switch

Can bring a system back to full operation after the occurrence of a fault

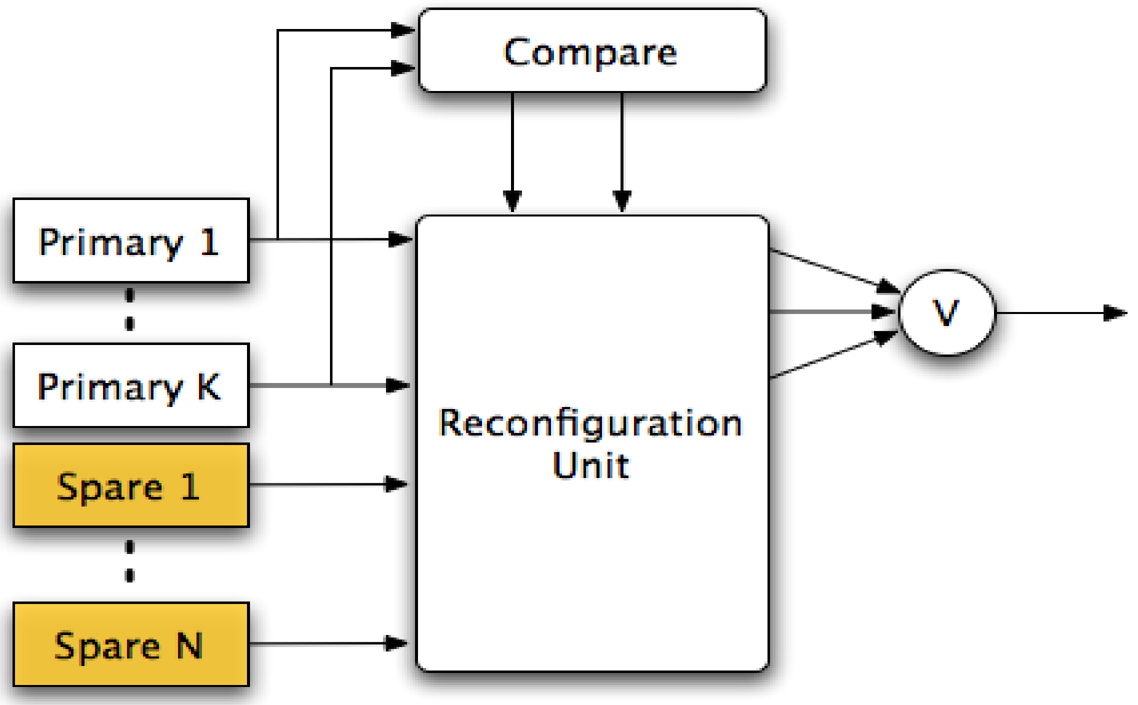
Require momentary cost in performance when reconfiguration is performed

Space Redundancy | 10

Hybrid Redundancy

Pair-and-a-spare

There is both fault masking and dynamic reconfiguration to cope with faulty modules



Space Redundancy | 11

Pair-and-a-spare

Combine the features in **standby sparing and duplication with comparison**

2 modules are operated in parallel at all times and their results are compared to provide the error protection capability

The error signal from the comparison is used to initiate the reconfiguration process (switch) that removes faulty modules and replaces them with spares

Time redundancy

All the concepts seen so far related to area/hardware redundancy can be applied to *time redundancy*

The application is executed multiple times and the results are checked…

Adopted for temporary, not permanent faults

In not “real-time” systems

Information Redundancy

TOPIC QUESTIONS

How can I modify data itself to allow for error detection/correction?

How do I protect (stored) data?

Information Redundancy

To detect/tolerate errors, additional information is introduced into data, allowing detection/tolerance

Information Redundancy

To detect/tolerate errors, additional information is introduced into data, allowing detection/tolerance

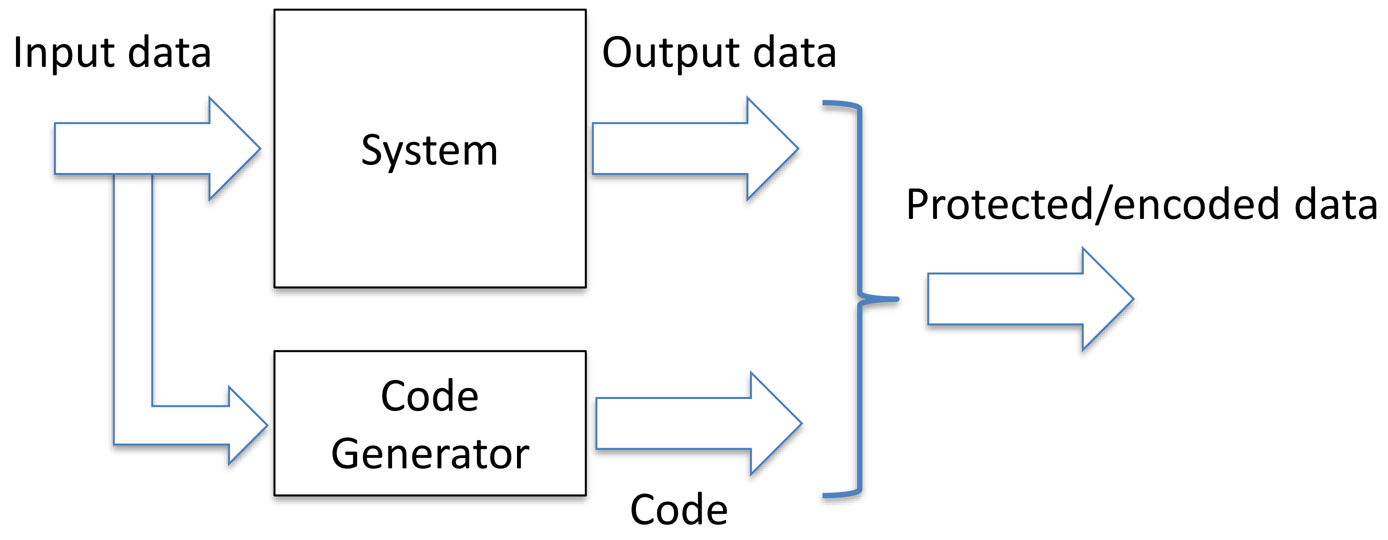
Information coding

Redundant Array of Independent Disks (RAID) :: for larger data structures

Information coding

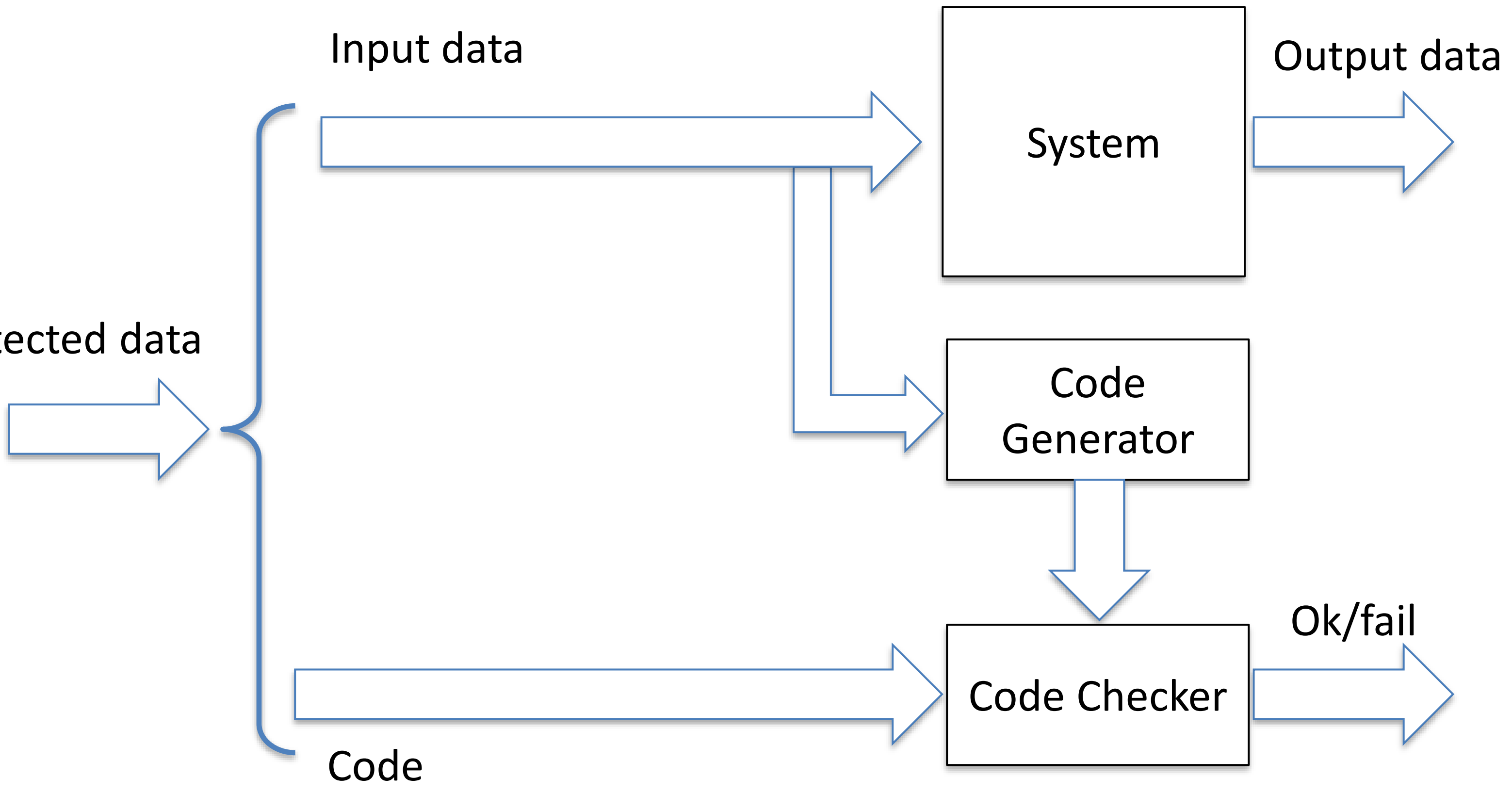
Information Redundancy

Data production



Information Redundancy

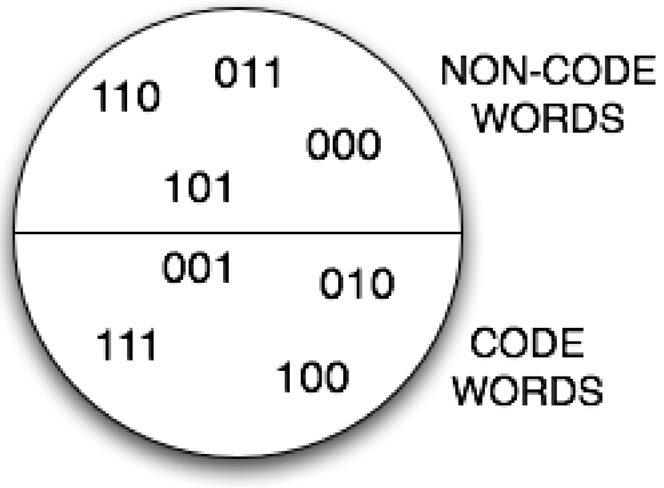
Data usage



Information Redundancy | 2

Partition the set of possible configurations in **codewords** and **non-codewords**

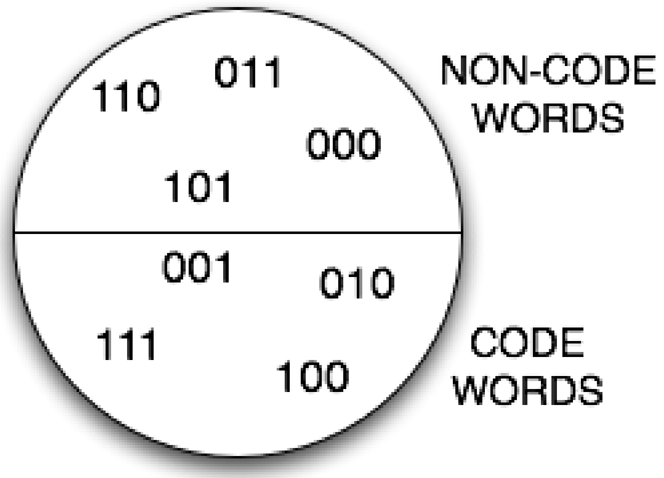
- In a fault-free situation data is a codeword
- In a faulty situation data is a non-codeword



Information Redundancy | 2

Partition the set of possible configurations in **codewords** and **non-codewords**

- In a fault-free situation data is a codeword
- In a faulty situation data is a non-codeword



Fault detection

Information Redundancy | 3

Associated with the adoption of a code, there must be a synthesis strategy to guarantee that **faults cause only detectable errors**, according to the adopted code

In other words, faults may cause data to be a non-codeword

Information Redundancy | 4

Combinational circuits

- Output encoding
- Input encoding

  allows covering faults on primary inputs

Sequential circuits

- Next-state function
- Output function

Error Detecting Codes (EDC)

Allow the detection of a fault when it produces an error

- Parity Bit
- Berger
- m-out-of-n
- checksum (single/double precision, residue, Honeywell, …)
- Cyclic codes
- Arithmetic codes

Parity Bit code

Least redundant bit is used to *code* the remaining bits

Odd parity bit is 1 if data contains an even number of ones

Even parity bit is 1 if data contains an odd number of ones

Parity on the encoded information

- Odd parity: 100010***1***
- Odd parity: 101010***0***
- Even parity: 100010***0***
- Even parity: 110010***1***

Parity Bit code

Examples:

- odd: 1000100***1***
- even: 1000100***0***

Parity Bit code

Fault detection:

- odd: 1000100***1***
- even: 1000100***0***

- odd: 1010100***1***
- even: 1010100***0***

Parity Bit code

Fault detection:

- odd: 1000100***1***
- even: 1000100***0***

- odd: 10101001***1***
- even: 10101000***0***

Should be 0

Parity check mismatch!
Error detected!

Should be 1

Parity Bit code

Fault detection:

- odd: 10001001***1***
- even: 10001000***0***

- odd: 10101001***1***
- even: 10101000***0***

Should be 1

mismatch!
Error detected!

- odd: 10001000***0***
- even: 10001001***1***

Parity check mismatch!
Error detected!

Should be 0

Parity Bit code

Examples:

- odd: 1000100***1***
- even: 1000100***0***

What about multiple faults?

Parity Bit code

Examples:

- odd: 1000100***1***
- even: 1000100***0***

- odd: 1001000***1***
- even: 1001000***0***

Should be 1

No parity check mismatch!
Error undetected!

Should be 0

Parity Bit code

Examples:

- odd: 10001001***1***
- even: 10001000***0***

- odd: 10010001***1***
- even: 10010000***0***

Should be 0

eck mismatch!
Error undetected!

- odd: 10011000***0***
- even: 10011001***1***

No parity check mismatch!
Error undetected!

Should be 1

Parity Bit code

Examples:

- odd: 1000100***1***
- even: 1000100***0***

- odd: 0001000***1***
- even: 0001000***0***

Parity check mismatch!
Error detected!

Parity Bit code

Examples:

- odd: 1000100***1***
- even: 1000100***0***

- odd: 0001000***1***
- even: 0001000***0***

Parity check mismatch!
Error detected!

- odd: 1101100***0***
- even: 1101100***1***

Parity check mismatch!
Error detected!

Parity Bit code

Examples:

- odd: 10001001***1***
- even: 10001000***0***

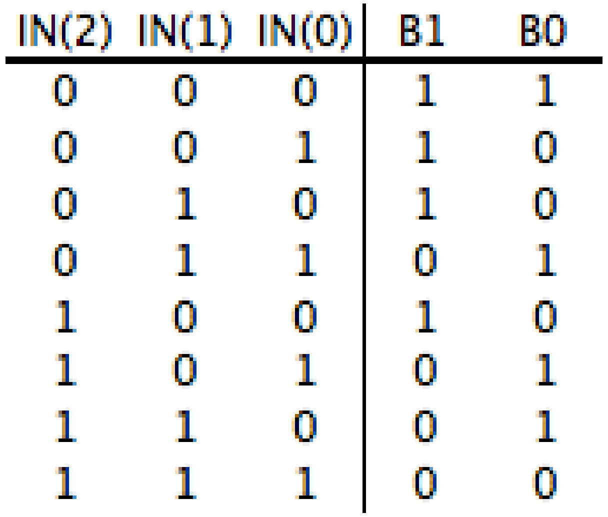
What about multiple faults?

Single parity bit detects an odd number of faults

Berger code

Code bits encode the number of 0s in the information bits

Costs varies with information size



IN(2)	IN(1)	IN(0)	B1	B0
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	0

Berger code

Code bits encode the number of 0s in the information bits

Costs varies with information size

IN(3)	IN(2)	IN(1)	IN(0)	B2	B1	B0
0	0	0	0	1	0	0
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	1	0
0	1	1	0	0	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	0	1	0	1	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	0	1	0
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	0	0

Berger code

Code bits encode the number of 0s in the information bits

Costs varies with information size

Able to detect any number of *unidirectional* faults

IN(3)	IN(2)	IN(1)	IN(0)	B2	B1	B0
0	0	0	0	1	0	0
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	1	0
0	1	1	0	0	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	0	1	0	1	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	0	1	0
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	0	0

Berger code

Code bits encode the number of 0s in the information bits

Costs varies with information size

***unidirectional* faults: only 0s becoming 1s or 1s becoming 0s**

IN(3)	IN(2)	IN(1)	IN(0)	B2	B1	B0
0	0	0	0	1	0	0
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	1	0
0	1	1	0	0	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	0	1	0	1	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	0	1	0
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	0	0

Berger code

Examples:

- 0000 ***100***

- 0001 ***100***
- 1010 ***100***

Number of 0s decreases and count keeps unaltered

- 0000 ***110***
- 0000 ***010***

- 0100 ***101***
- 0110 ***111***

Berger code

Examples:

- 0000 ***100***

- 0001 ***100***
- 1010 ***100***

Number of 0s decreases and count keeps unaltered

- 0000 ***110***
- 0000 ***010***

Number of 0s keeps unaltered and count increases

- 0100 ***101***
- 0110 ***111***

Berger code

Examples:

- 0000 ***100***

- 0001 ***100***
- 1010 ***100***

Number of 0s decreases and count keeps unaltered

- 0000 ***110***
- 0000 ***010***

Number of 0s keeps unaltered and count increases

- 0100 ***101***
- 0110 ***111***

Number of 0s decreases and count increases

Berger code

Examples:

- 0000 ***100***

- 0001 ***100***
- 1010 ***100***

- 0000 ***110***
- 0000 ***010***

- 0100 ***101***
- 0110 ***111***

Number of 0s decreases
and count keeps unaltered

unidirectional **faults are always detected**

Number of 0s decreases
and count increases

Berger code

Examples:

- 0010 ***011***

- 0001 ***011***

- 0101 ***011***

- 0001 ***001***

- 1010 ***010***

bidirectional **faults may be undetected**

m-out-of-n code

Given m-bit data strings, n-bit keys are added so that the number of 1s (or 0s) in the resulting bit string is constant

m-out-of-n code

Given m-bit data strings, n-bit keys are added so that the number of 1s (or 0s) in the resulting bit string is constant

d1	d2	d3	k1	k2	k3
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1	0	0	0

Error Correcting Codes (ECC)

Allow also to correct a non-codeword, identifying the codeword corrupted by the error

Error Correcting Codes (ECC)

Allow also to correct a non-codeword, identifying the codeword corrupted by the error

Given a non-codeword *nc* there is only one codeword *c* such that an error transforms *c* in *nc*…

- Hamming
- BCH
- Reed-Solomon & Reed-Muller,
- Binary Golay,
- convolutional & turbo

Hamming code

Detects double errors (2 erroneous bits), correct single errors

Introduces additional check bits, each one checking a subset of the information bits

Each check bit computes the parity

Several versions exist:

- 1bit data + 2bit check
- 4bit data + 3bit check
- 11bit data + 4bit check
- …

Hamming code | 2

Bits of position 2^k are check bits
(1, 2, 4, 8, 16, 32, 64, …)

Each parity bit calculates the parity for some of the bits in the code word

Hamming code | 2

Bits of position 2^k are check bits
(1, 2, 4, 8, 16, 32, 64, …)

Each parity bit calculates the parity for some of the bits in the code word

- Parity bit 1 covers all bit positions which have the least significant bit set to 1 => bit 1 (the parity bit itself), 3, 5, 7, 9, etc.

Hamming code | 2

Bits of position 2^k are check bits
(1, 2, 4, 8, 16, 32, 64, …)

Each parity bit calculates the parity for some of the bits in the code word

- Parity bit 1 covers all bit positions which have the least significant bit set to 1 => bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
- Parity bit 2 covers all bit positions which have the second least significant bit set to 1 => bits 2-3, 6-7, 10-11, etc.

Hamming code | 2

Bits of position 2^k are check bits
(1, 2, 4, 8, 16, 32, 64, …)

Each parity bit calculates the parity for some of the bits in the code word

- Parity bit 1 covers all bit positions which have the least significant bit set to 1 => bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
- Parity bit 2 covers all bit positions which have the second least significant bit set to 1 => bits 2-3, 6-7, 10-11, etc.
- Parity bit 4 covers all bit positions which have the third least significant bit set to 1 => bits 4–7, 12–15, 20–23, etc.

Hamming code | 2

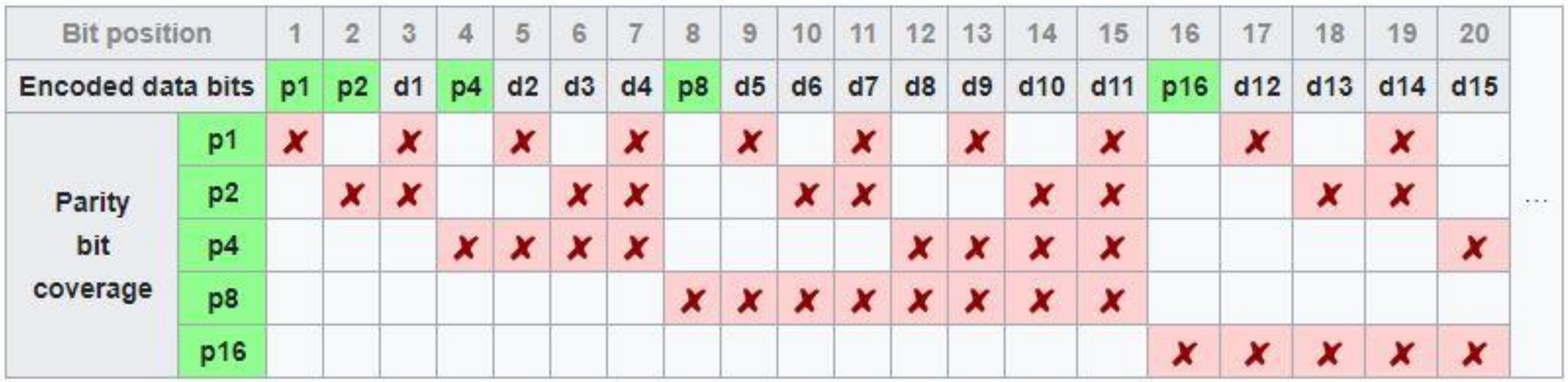
Bits of position 2^k are check bits
(1, 2, 4, 8, 16, 32, 64, …)

Each parity bit calculates the parity for some of the bits in the code word

- Parity bit 1 covers all bit positions which have the least significant bit set to 1 => bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
- Parity bit 2 covers all bit positions which have the second least significant bit set to 1 => bits 2-3, 6-7, 10-11, etc.
- Parity bit 4 covers all bit positions which have the third least significant bit set to 1 => bits 4–7, 12–15, 20–23, etc.
- …
- In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.

Hamming code | 2

Bits of position 2^k are check bits
(1, 2, 4, 8, 16, 32, 64, …)



Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		…
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

bit set to 1 → bits 4–7, 12–15, 20–23, etc.

- …
- In general each parity bit covers all bits where the bitwise AND of the parity position and the bit position is non-zero.

Hamming code | 3

Coding (example 4 data bits + 3 check bits)

Data to be coded: 1010

The encoded string will be:

1	2	3	4	5	6	7
P_1	P_2	1	P_3	0	1	0

Hamming code | 3

Coding (example 4 data bits + 3 check bits)

Data to be coded: 1010

The encoded string will be:

1	2	3	4	5	6	7
P_1	P_2	1	P_3	0	1	0

P_1=EvenParity(P_1,1,0,0) => P_1 = 1

Hamming code | 3

Coding (example 4 data bits + 3 check bits)

Data to be coded: 1010

The encoded string will be:

1	2	3	4	5	6	7
1	P_2	1	P_3	0	1	0

P_2=EvenParity(P_2,1,1,0) => $P_2 = 0$

Hamming code | 3

Coding (example 4 data bits + 3 check bits)

Data to be coded: 1010

The encoded string will be:

1	2	3	4	5	6	7
1	0	1	P_3	0	1	0

P_3=EvenParity(P_3,0,1,0) => P_3 = 1

Hamming code | 3

Coding (example 4 data bits + 3 check bits)

Data to be coded: 1010

The encoded string is:

1	2	3	4	5	6	7
1	0	1	1	0	1	0

Hamming code | 3

Coding (example 4 data bits + 3 check bits)

Data to be coded: 1010

The received string is:

1	2	3	4	5	6	7
1	0	1	1	0	1	0

$$\left.\begin{array}{l} \text{EvenParity}(1,3,5,7)=\text{EvenParity}(1,1,0,0)=0 \\ \text{EvenParity}(2,3,6,7)=\text{EvenParity}(0,1,1,0)=0 \\ \text{EvenParity}(4,5,6,7)=\text{EvenParity}(1,0,1,0)=0 \end{array}\right\} \text{No fault occurred}$$

Hamming code | 3

Coding (example 4 data bits + 3 check bits)

Data to be coded: 1010

The received string is:

1	2	3	4	5	6	7
1	0	1	1	0	0	0

EvenParity(1,3,5,7)=EvenParity(1,1,0,0)=0

EvenParity(2,3,6,7)=EvenParity(0,1,0,0)=1

EvenParity(4,5,6,7)=EvenParity(1,0,0,0)=1

$\}$ Fault occurred in position 6

Hamming code | 3

Coding (example 4 data bits + 3 check bits)

Data to be coded: 1010

The received string is:

1	2	3	4	5	6	7
1	1	1	1	0	1	0

$$\left.\begin{array}{l}\text{EvenParity}(1,3,5,7)=\text{EvenParity}(1,1,0,0)=0\\ \text{EvenParity}(2,3,6,7)=\text{EvenParity}(1,1,1,0)=1\\ \text{EvenParity}(4,5,6,7)=\text{EvenParity}(1,0,1,0)=0\end{array}\right\}\text{Fault occurred in position 2}$$

Hamming code | 3

Coding (example 4 data bits + 3 check bits)

Data to be coded: 1010

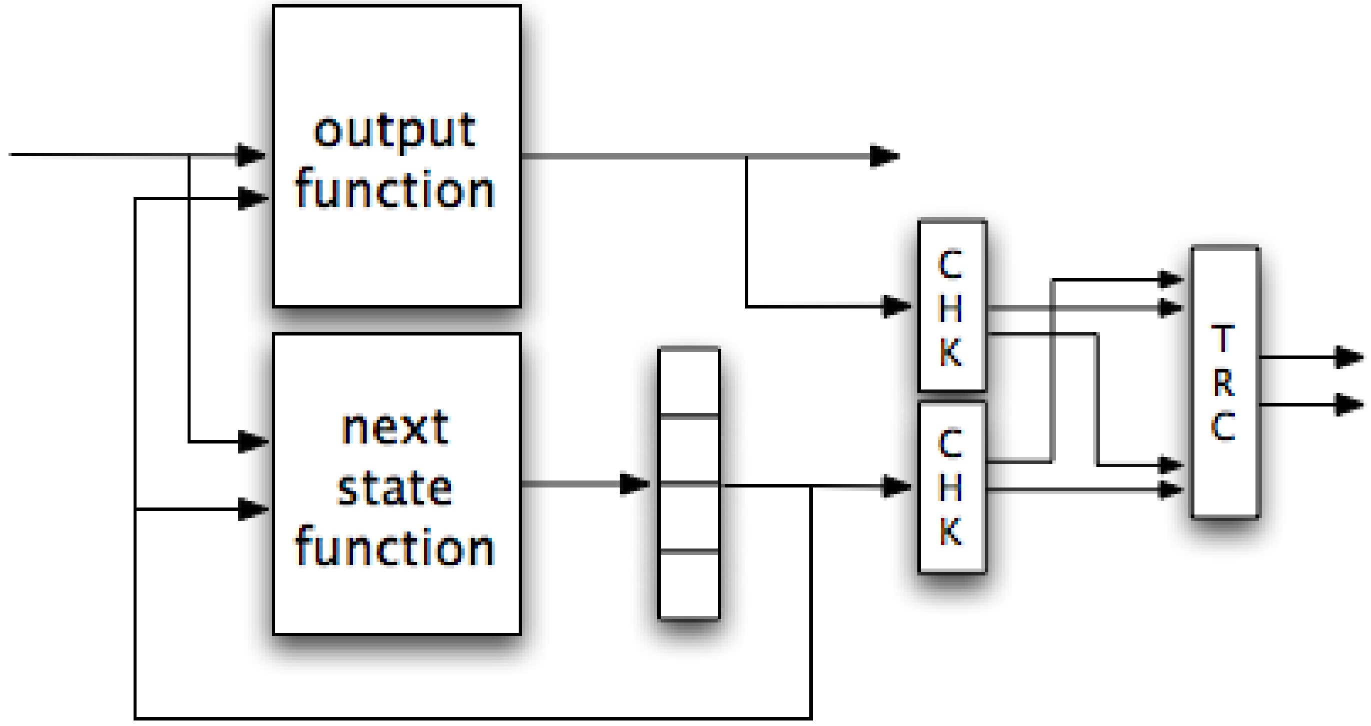
The received string is:

1	2	3	4	5	6	7
0	0	1	1	1	1	0

$$\left.\begin{array}{l}\text{EvenParity}(1,3,5,7)=\text{EvenParity}(0,1,1,0)=0\\ \text{EvenParity}(2,3,6,7)=\text{EvenParity}(0,1,1,0)=0\\ \text{EvenParity}(4,5,6,7)=\text{EvenParity}(1,1,1,0)=1\end{array}\right\}\ \begin{array}{l}\text{Double faults detected}\\ \text{but not corrected}\end{array}$$

Sequential circuits

Encoding of the next-state and of the output function of sequential circuits



Redundant Array of Independent Disks (RAID)

RAID

Multiple Hard Disks are deployed to introduce data redundancy

Several versions of RAID have been proposed

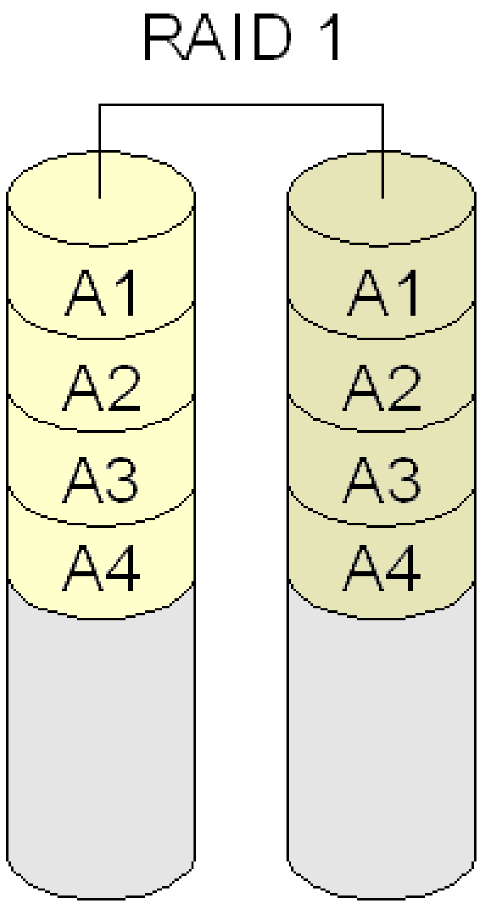
RAID 1

Simply two (or more) mirrored hard disks

The more mirrors, the more reliability

Low scalability

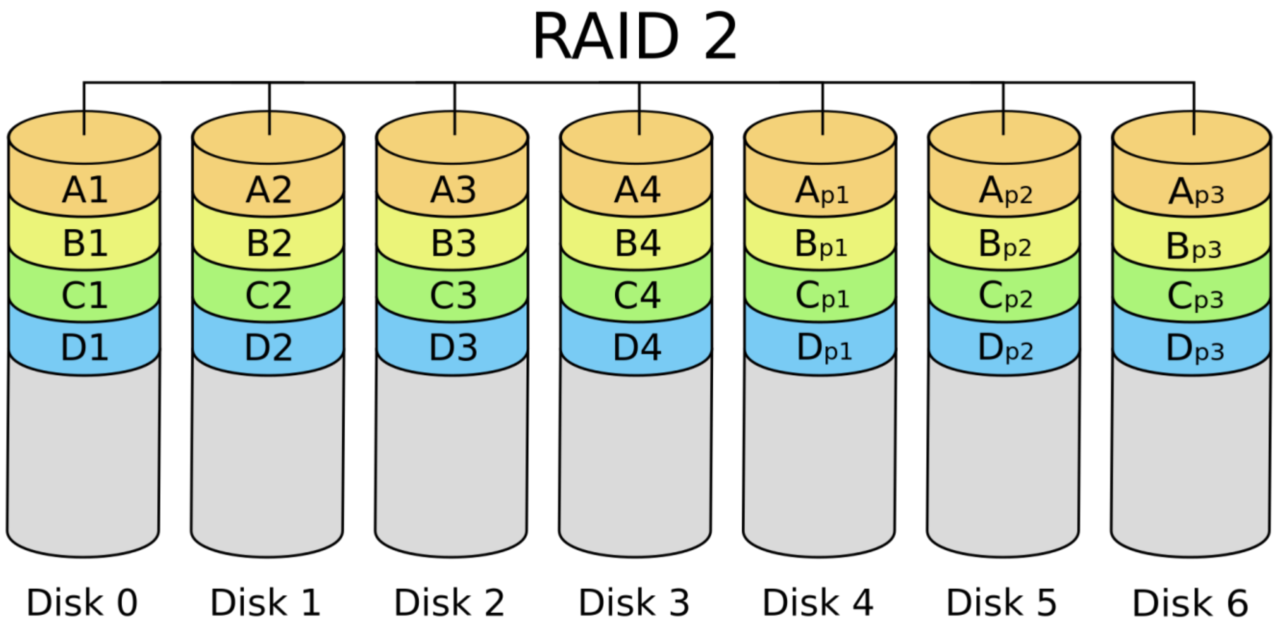
The slowest disk affects the speed of the entire system



RAID 2

Data is bit-wise partitioned among disks

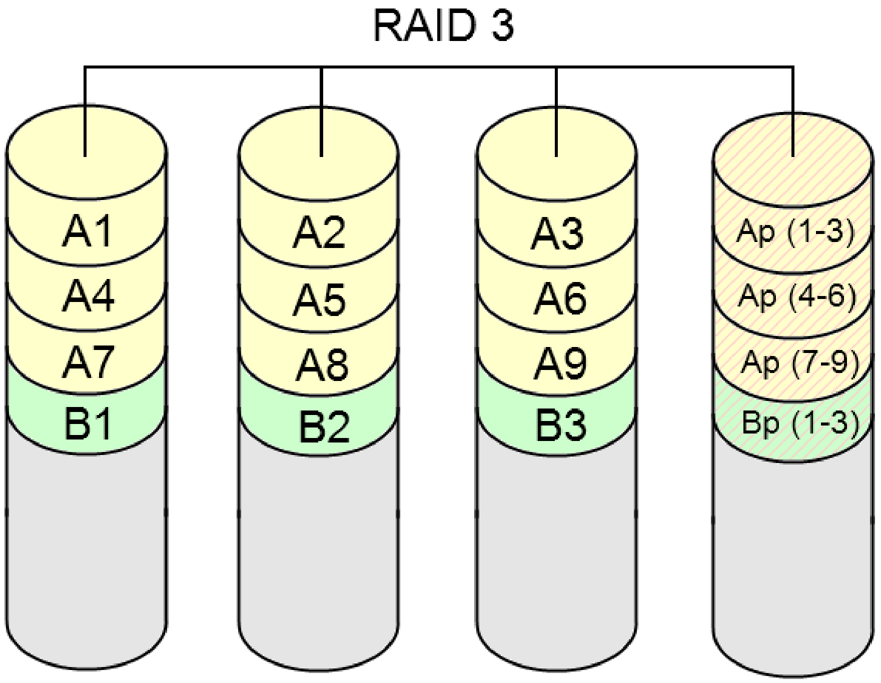
Additional disks for hamming code bits are added



RAID 3

Data is Byte-wise partitioned among disks

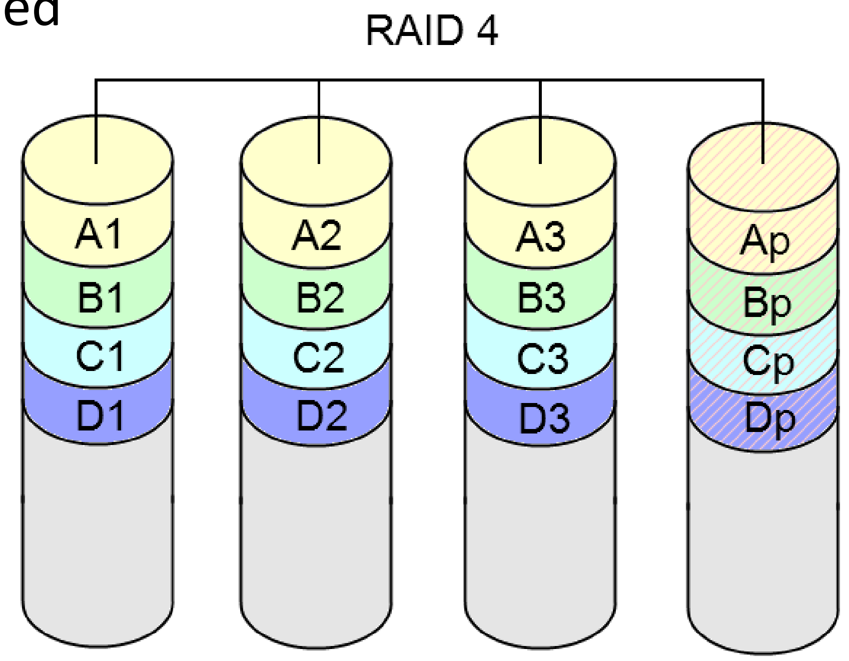
An additional disk for parity is added



RAID 4

Like RAID 3 but data is block-wise partitioned among disks

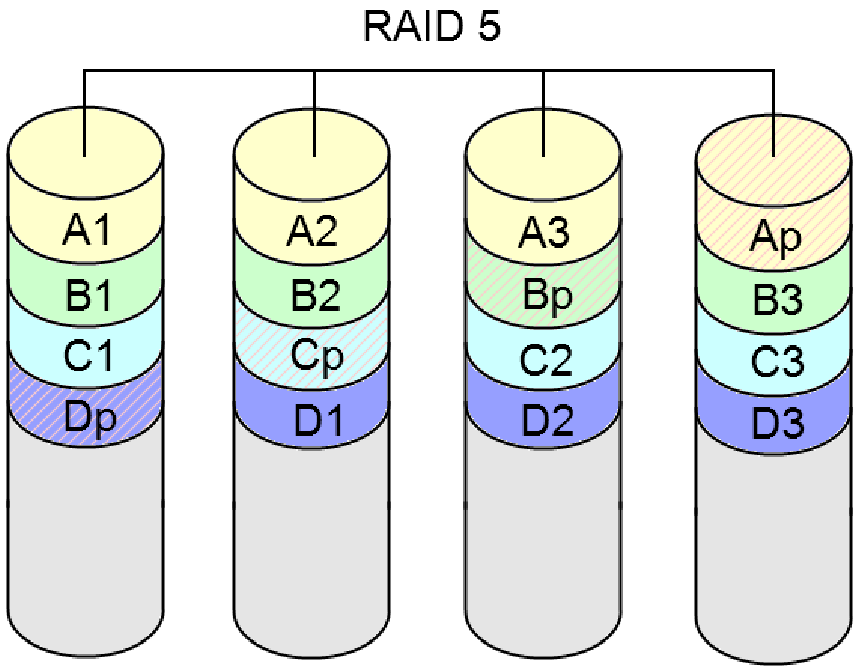
An additional disk for parity is added



RAID 5

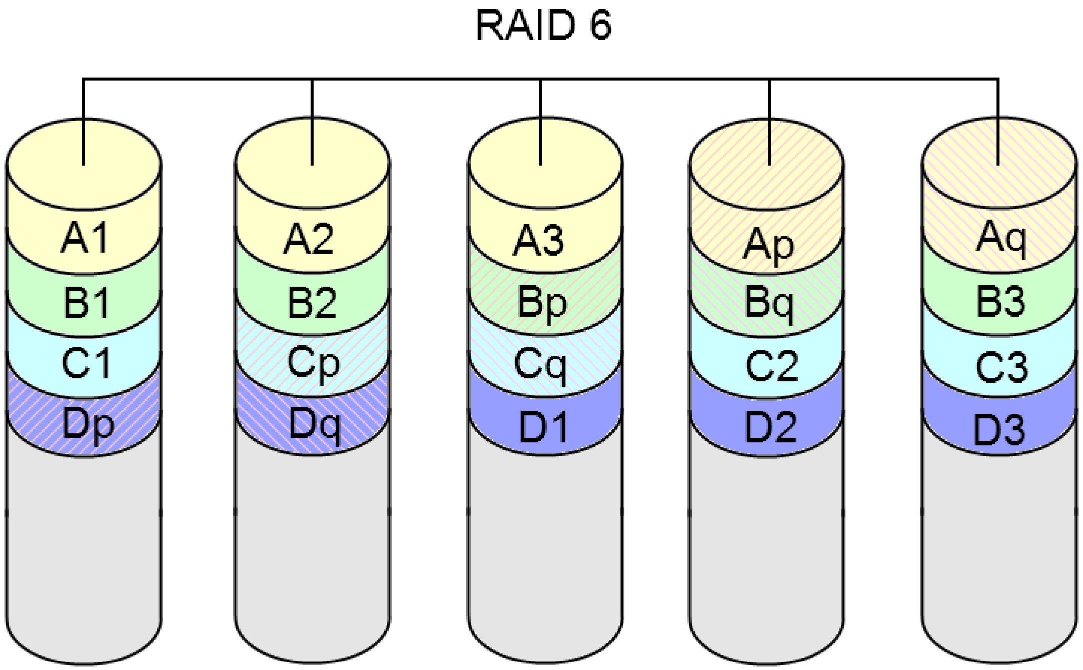
Like RAID 4, data is block-wise partitioned among disks

Parity is distributed among disks



RAID 6

Like RAID 5 but with double parity distributed among disks



TOPIC QUESTIONS

How can we handle the occurrence of faults?

Is there a more appropriate kind of redundancy than others?

What technique is more effective?

What are the costs?

TOPICS

Key: Redundancy

Techniques exploiting different kinds of redundancy

Relevant aspects: costs, performance, power, fault coverage