



Redundancy in Fault Tolerant Computing

Lecture 1

Prof. Cinzia Bernardeschi
Department of Information Engineering
University of Pisa, Italy
cinzia.bernardeschi@unipi.it

May 7-10, 2019 – Thessaloniki, Greece

- **fault tolerant computing: why and what**
- **computer-based systems: faults and failures**
- **forms of redundancy:**
 - Hardware redundancy
 - Information redundancy
 - Timing redundancy
 - Software redundancy
- **effectiveness of fault tolerance**

Fault tolerant computing: why and what



UNIVERSITÀ DI PISA

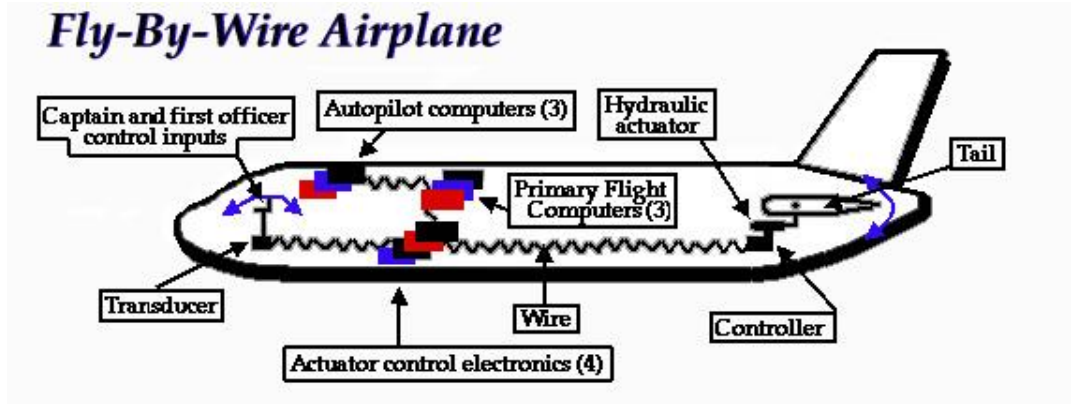
Computers are increasingly used in safety-critical systems:

- transport (automotive, railways, aerospace, ...)
- medicine
- process control
-

Future safety-critical systems will be more automated and more dependent on computers

Fault tolerant computing:

the ability of the system to deliver the expected functionality during its operational life also in case of malfunctions (important in safety-critical systems, systems whose failure may result in death or serious injury to people, loss or serious damage of equipment, or environmental harm)



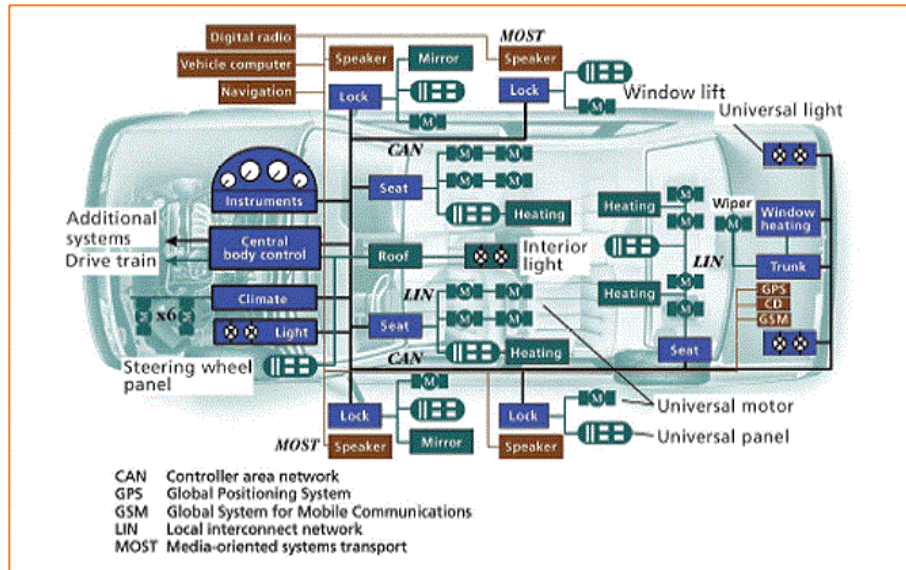
<http://www.aviationcoaching.com/wp-content/uploads/2015/08/fly-by-wire-system.jpg>

Earliest aircraft: controlled by the pilot using the steel cables, pulleys and hydraulic actuators

Fly-by-wire (FBW) system: all commands and signals are transmitted electrically along wires.

- These signals are sent to flight-control computers (FCS) that reconvert the electrical impulses into instructions for control surfaces like wing flaps or the tail.

Sensing and Computing in Cars



Over 80 different embedded processors, interconnected with each other.

Key ECUs (Electronic Control Unit):

- Engine Control Modul (ECM)
- Electronic Brake Control Module (EBCM)
- Transmission Control Module (TCM)
- Vehicle Vision System (VVS)
- Navigation Control Module (NCM)
- ...

Main vehicle control systems replaced with electronic controls (no physical connection):

- throttle - **Electronic Throttle Control**
- brakes - **Brake-by-Wire**
- steering - **Steer-by-Wire**

- For a computer based safety-critical system, the safety of the system depends strongly on its computers.
- Faults are unexpected events that may compromise the system functionality
- Faults in computer systems:
 - hardware faults (e.g., stuck-at 0 of a line, memory cell bit flip)
 - software faults (e.g., a bug in the code)
- General questions: How to build dependable computer-based systems.
Can we justifiably trust the dependability of such systems?

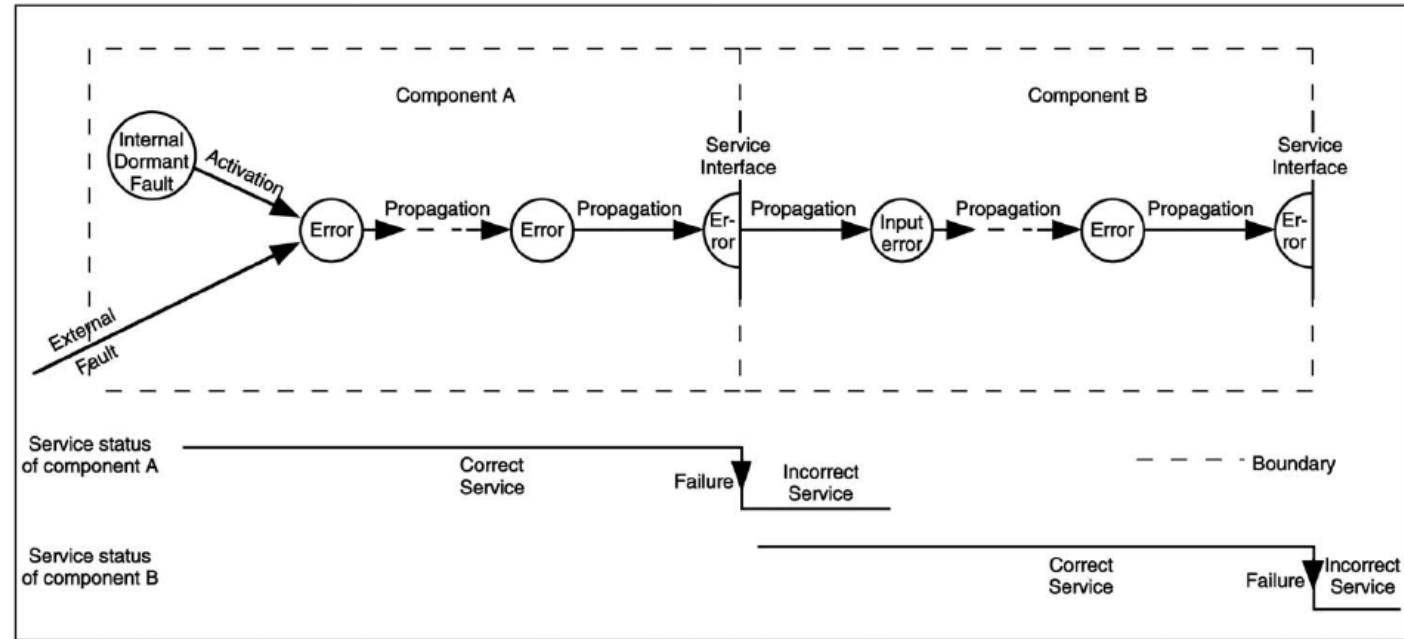
1. a system is as strong as its weakest component

- Hw and sw systems relaying on hidden components

2. small hidden faults may have large effects (digital machine)

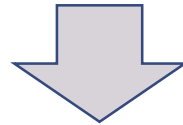
- Computer failures differ from failures of other equipment
- Subtler failures than “breaking down” or “stopping working”, ..
- The computer is used to store information: there are many ways information can be wrong, many different effects both within and outside the computer

Chain of threats: Faults -> Errors -> Failures



Taken from [Avizienis et al. 2004]

- Make available
 - two or more copies of data item that may be corrupted
 - a mechanism that compares them and declares an error if they differ
- The two copies must be unlikely to be corrupted together and in the same way
Examples: Duplicated circuitry, Transmit messages twice, Store data in two separate places (e.g. mirrored disks)



Replication:

can have a very important impact on a system in the area of performance, size, weight, power consumption and others

Redundancy in fault tolerant computing



UNIVERSITÀ DI PISA

HARDWARE REDUNDANCY

Physical replication of hw (the most common form of redundancy)

The cost of replicating hw within a system is decreasing because the costs of hw is decreasing

INFORMATION REDUNDANCY

Addition of redundant information to data in order to allow fault detection and fault masking

TIME REDUNDANCY

Attempt to reduce the amount of extra hw at the expense of using additional time

SOFTWARE REDUNDANCY

Tolerating faults in software

HARDWARE REDUNDANCY

Passive fault tolerant techniques

- use **fault masking** to hide the occurrence of faults
- rely upon voting mechanisms to mask the occurrence of faults
- do not require any action on the part of the system / operator
- generally do not provide for the detection of faults

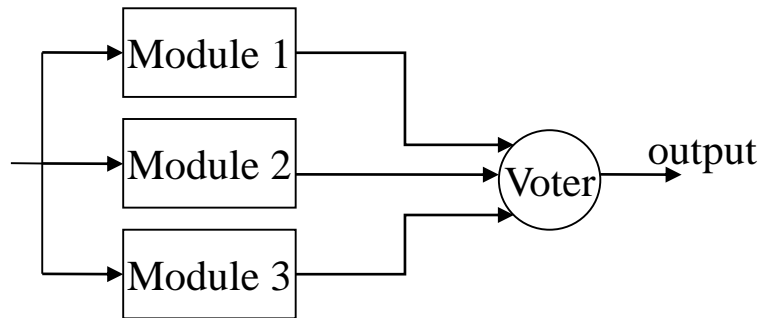
Active fault tolerance techniques

- *use* **fault detection, location and recovery**
- detect the existence of faults and perform some actions to remove the faulty hw from the system require the system to perform reconfiguration to tolerate faults
- common in applications where temporary, erroneous results are acceptable while the system reconfigures (satellite systems)

Hybrid approach

- very expensive
- often used in critical computations in which fault masking is required to prevent momentary errors and high reliability must be achieved

1. Triple Modular Redundancy (TMR) – fault masking



Voter is a single point of failure

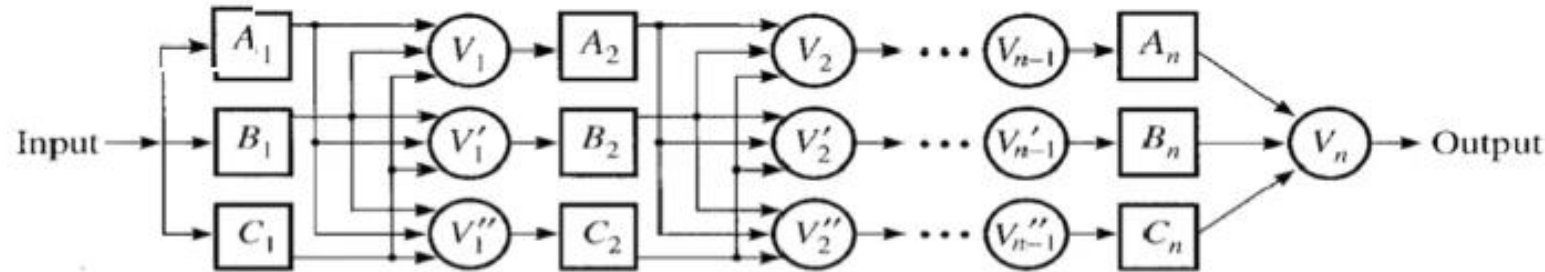
Triplicate the hw (processors, memories, ..) and perform a majority vote to determine the output

- 2/3 of the modules must deliver the correct results
- effects of faults neutralised without notification of their occurrence
- masking of a failure in any one of the three copies

Good for transient faults

For permanent faults, since the faulty module is not isolated, the fault tolerance decreases

Cascading TMR with triplicated voters



Taken from [Siewiorek et al., 1998]

The effect of partitioning of modules (A, B, C) is that the design can withstand more failures than the solution with only one large triplicated module

The partition cannot be extended to arbitrarily small modules, because reliability improvement is bounded by the reliability of the voter

Triplicated voters: voter errors propagates only of one step

Reliable Voters

Hardware voters are bit voters that compute the majority on n input bits.

Optimal designs of hardware voters with respect to circuit complexity, number of logic levels, fan-in and fan-out and power dissipation

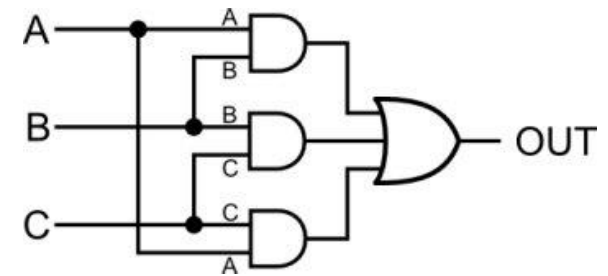
Difficulties

Delay in signal propagation:

- due to the voter
- due to multiple copies synchronisation

Trade-off : achieved fault tolerance vs hw required

1 bit voter



$$\text{OUT} = AB + BC + AC$$

using multiple analog to digital convertes and performing bit-by-bit voting on their digital output is not satisfactory

The three results from the analog to digital converters may not completely agree, for example, they could produce a result which differs for the least-significant bit even if the exact signal is passed through the same converter

- perform voting in the analog domain:
 - *average the three signals*
 - *choose the mean of the two most most similar signals*
 - *choose the median of the three signals (pseudo voting)*

2. NMR

N is made an odd number

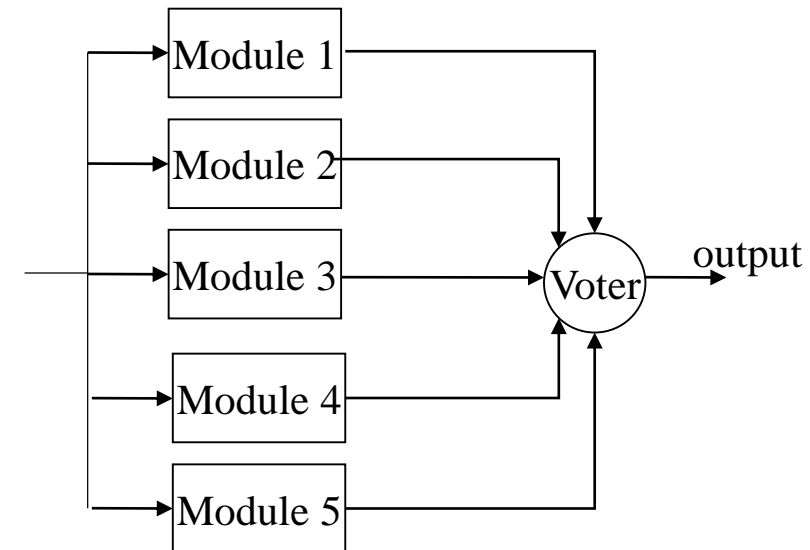
Coverage:

m faulty modules, with $N = 2m + 1$

5MR: tolerates 2 faulty modules

7MR: tolerates 3 faulty modules

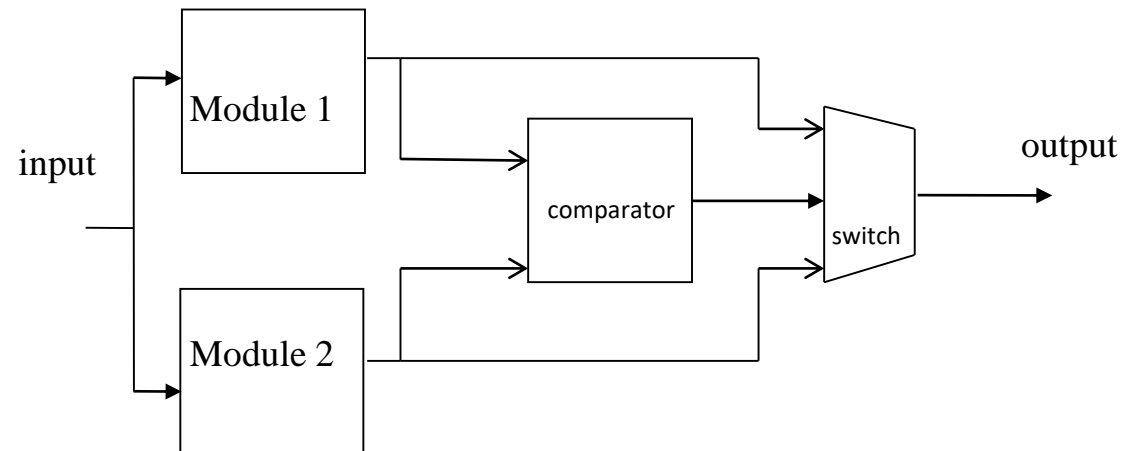
.....



1. Duplication with comparison scheme (Error detection)

Two identical pieces of hw (Module1 and Module 2) are employed

- they perform the same computation in parallel
- when a failure occurs, the two outputs are no more identical and a simple comparison detects the fault
Then the comparator (hw component) selects the correct output and reconfigure the switch to select the correct value
- the comparator must select the correct value



Dual-modular redundancy
(also Duplex system)

Active hw redundancy: the comparator

Assumption:

the two copies must be unlikely to be corrupted together in the same way

The comparator applies checks to select the correct output

Types of checks

-**Coding**

-**Self-checking circuitry**

-**Reversal Checks**

Assumption: the specified function of the system is to compute a mathematical function F and the function has an inverse function F' , such that $F'(F(x))=x$

Check: let output = $F(\text{input})$. Compute $F'(\text{output})$ and verify that $F'(\text{output}) = \text{input}$

-**Specification checks** (use the definition of “correct result”)

Example: Specification: find the solution of an equation
Check: substitute results back into the original equation

- **Reasonableness Checks**

Divide by 0

Acceptable ranges of variables

Acceptable transitions

Probable results

-.....

Problems

- need to check if the output data are valid. The comparator may not be able to perform an exact comparison, depending on the application area (digital control applications)
- faults in the comparator may cause an error indication when no error exists (false positive) or possible faults in duplicated modules are never detected (false negative)

Coverage

- detects all single faults except those of the comparison element

Advantages

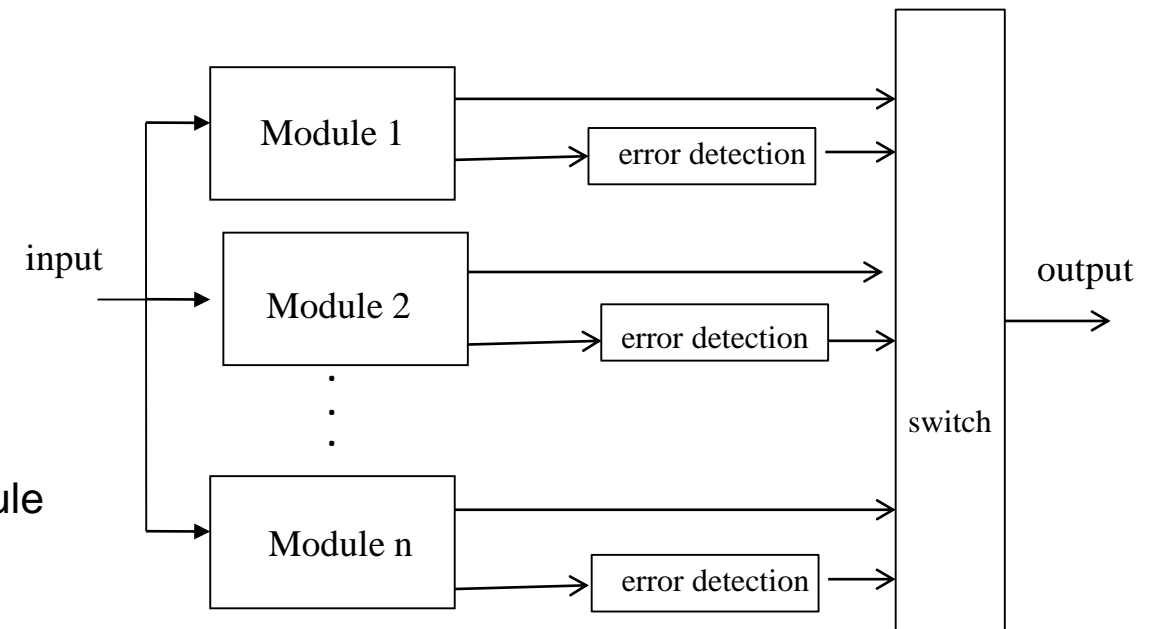
- simplicity, low cost, low performance impact of the comparison technique, applicable to all levels and areas

2. Stand-by sparing (error detection, reconfiguration)

Part of the modules are operational, part of the modules are spares modules (used as replacement modules)
The switch can decide no longer use the value of a module (fault detection and localization). The faulty module is removed and replaced with one of the spares.

- *hot spares*
the spares operate in synchrony with the on line modules, and they are prepared to take over
- *warm spares*
the spares are running but receive inputs only after switching
- *cold spares*
the spares are unpowered until needed to replace a faulty module

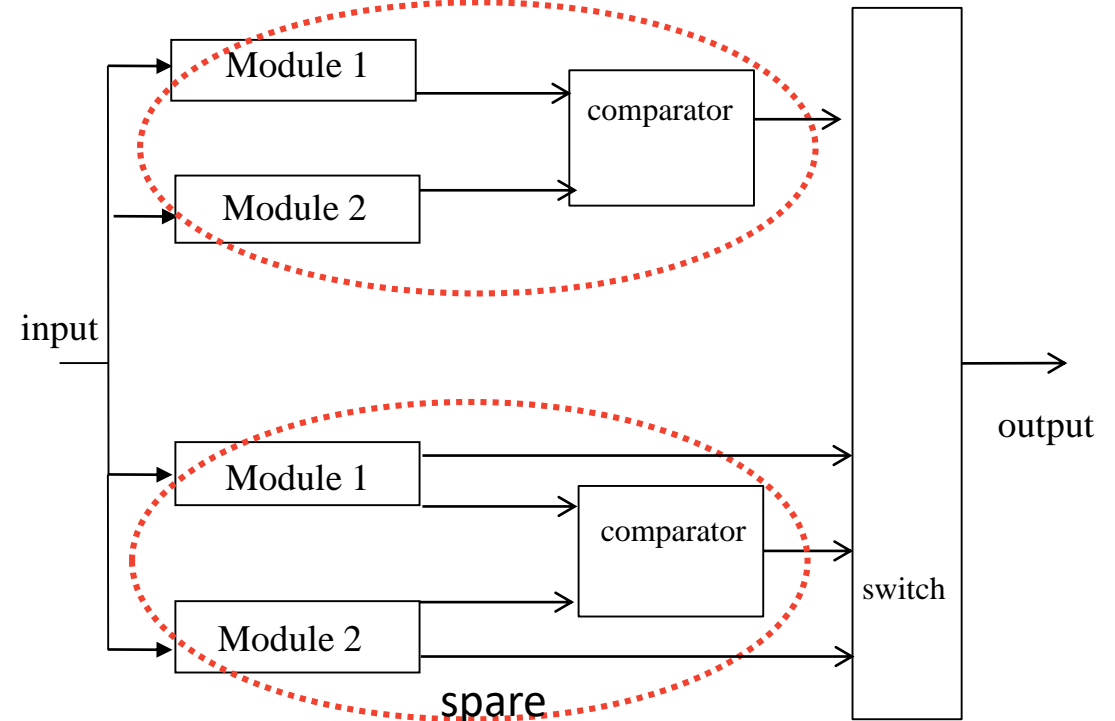
As long as the outputs agree, the spares are not used.



Different schemes can be implemented

Pair-and-spare approach

- A module is a duplex system, pairs connected by a comparator
- Duplex systems are connected to spares by a switch
- As long as the two outputs agree, or the comparator can detect the right value, the spare is not used.
- Otherwise, the comparator signals the switch that it is not able to compute the right value and the switch operates a replacement using the spare.



Pair results are used in a spare arrangement. Spare components at coarser granularity.
Not all four copies must be synchronised (only the two pairs)

Hybrid approaches

Combine both the active and passive approaches

Very expensive in terms of the amount of hw required to implement a system

Applied in commercial systems, safety critical system (aviation, railways, ...)

NMR with spares (Reconfigurable NMR):

Modules arranged in a voting configuration

- spares to replace faulty units
- rely on detection of disagreements and determine the module(s) not agreeing with the majority

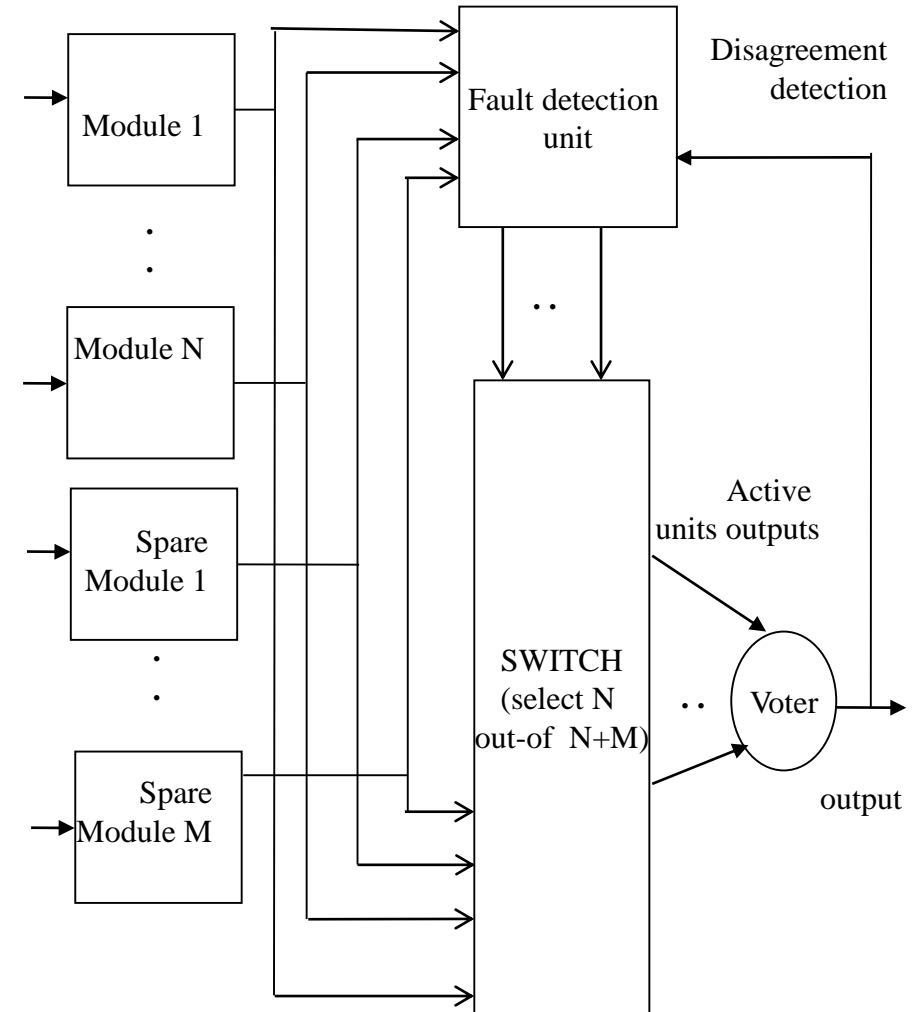
- N redundant module configuration (active modules)
- Voter (votes on the output of active modules)
- The Fault detection units
 - 1) compares the output of the Voter with the output of the active modules
 - 2) replaces modules whose output disagree with the output of the voter with spares

Reliability

as long as the spare pool is not empty

Coverage

TMR with one spare can tolerate 2 faulty modules
(mask the first faulty module; replace the module;
mask the second faulty module)



Key differences

Passive: rely on fault masking

Active: rely on error detection, location and recovery

Hybrid: employ both masking and recovery

- **Passive** provides fault masking but requires investment in hw (5MR can tolerate 2 faulty modules)
- **Active** has the disadvantage of additional hw for error detection and recovery, sometimes it can produce momentary erroneous outputs
- **Hybrid** techniques have the highest reliability but are the most costly (3MR with one spare can tolerate 2 faulty modules)

INFORMATION REDUNDANCY

Information is represented with more bits than strictly necessary: says, an n -bit information chunk is represented by

$$n+c = m \text{ bits}$$

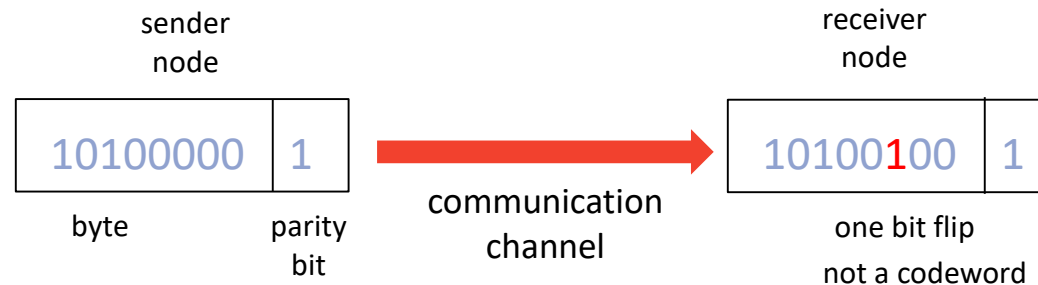
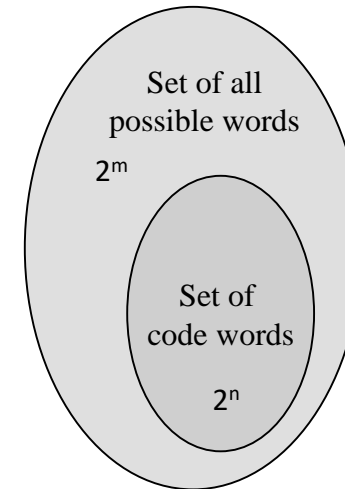
Among all the possible 2^m configurations of the m bits, only 2^n represent acceptable values (code words)

if a non-code word appears, it indicates an error in transmitting, or storing, or retrieving ...

Parity code

for each unit of data, e.g. 8 bits, add a parity bit so that the total number of 1's in the resulting 9 bits is odd

Two bit flips are not detected



Codes

encoding: the process of determining the c bit configuration for a n bit data item

decoding: the process of recovering the original n bit data from the m bit total bit

Separable code: a code in which the original information is appended with new information to form the codeword. The decoding process consists of simply removing the additional information and keeping the original data

Nonseparable code: requires more complicated decoding procedures

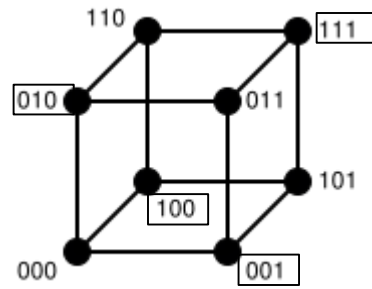
Parity code is a separable code

Additional information can be used for error detection and may be for error correction

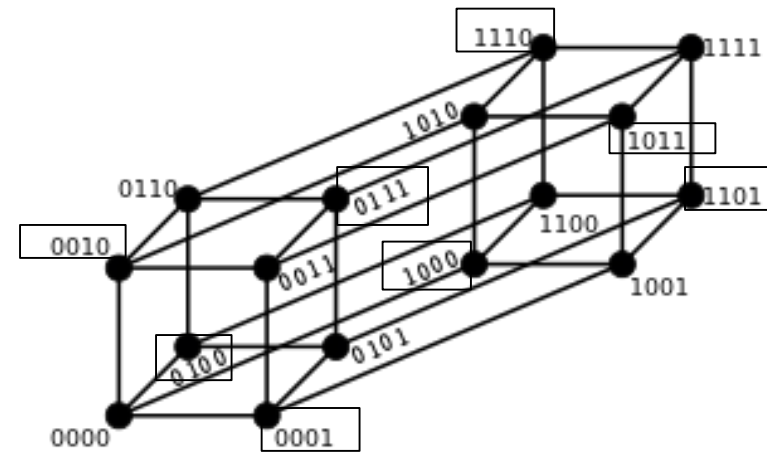
Examples of codes

- boxed words: code words

Parity-code
odd parity



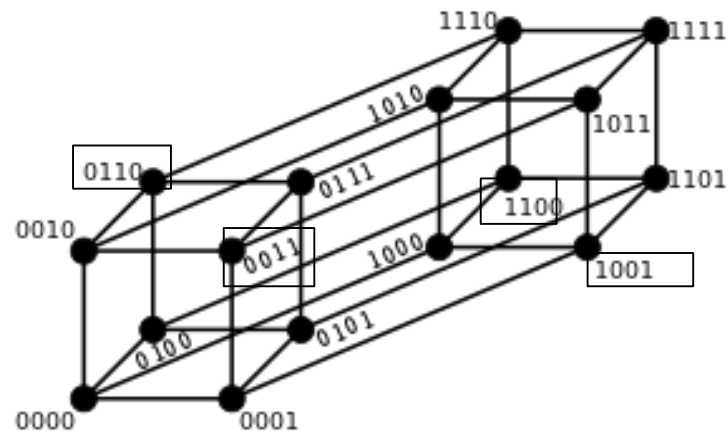
3-bit words – 4 code words



4-bit words – 8 code words

Examples of codes

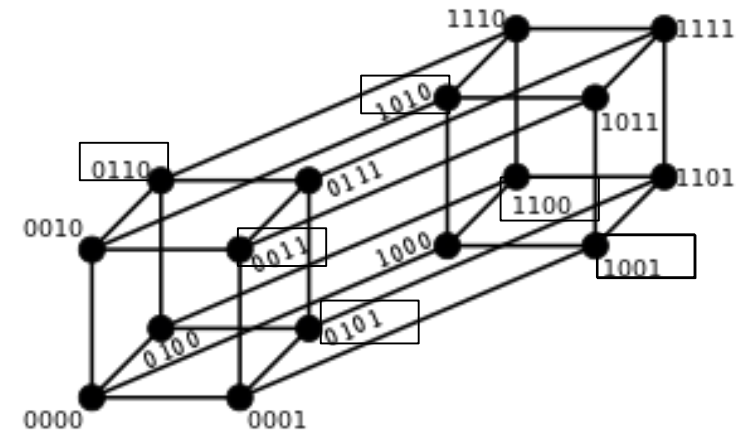
CD - complemented
duplication



4-bit words - 4 code words

m/n code:
m bit equal to 1

2/4 code



4-bit words - 6 code words

Distances and data spaces



UNIVERSITÀ DI PISA

Hamming distance between two data items: count the number of bits that are different

A code such that the Hamming distance between two code words is $> k$ will detect all errors that flip up to k bits

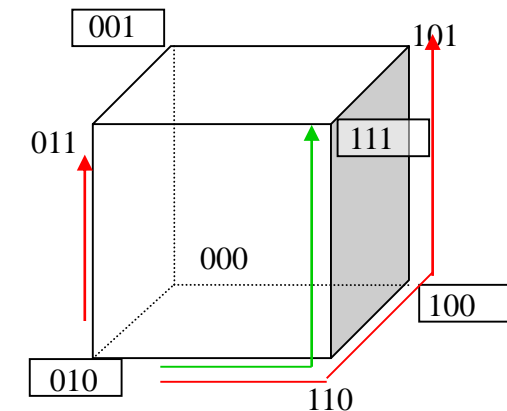
Memories of computer systems.

Parity bit added before writing the memory. Parity bit is checked when reading.

Useful distance measures depend on type of data and faults

Bank account numbers should be such that mistyping a digit does not credit the wrong account.

Parity-code: Hamming distance 2



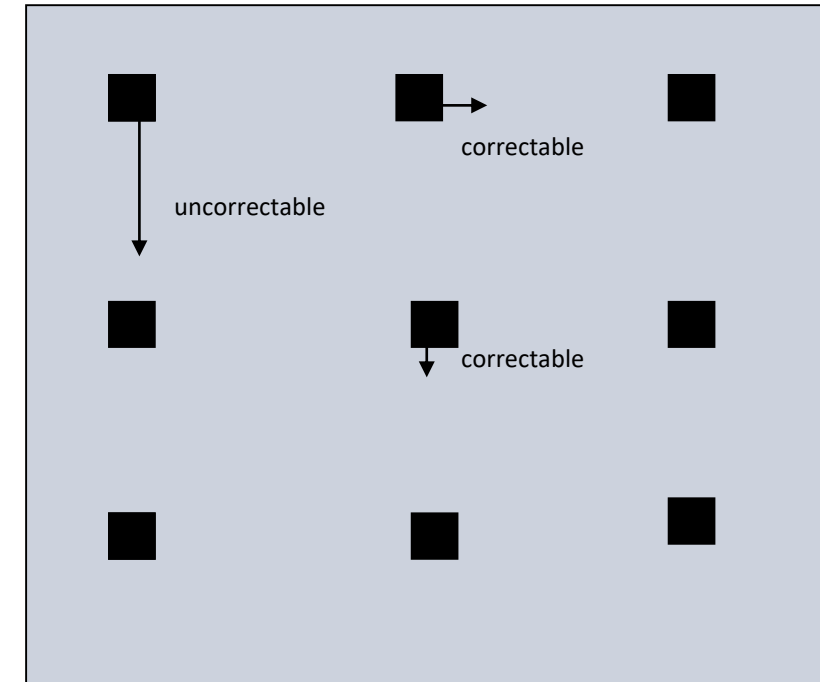
→ undetectable
→ detectable

Minimum Hamming distance:
minimum distance between two code words

A code such that the minimum Hamming distance is k will detect up to $k-1$ single bit errors

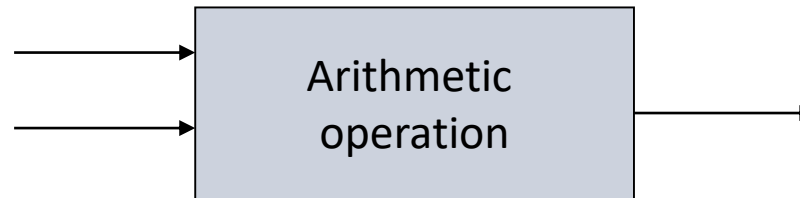
A code such that the minimum Hamming distance is k will correct up to d errors, where $k = 2d + 1$

Hamming distance 3:
detects 1 or 2 bits errors
correct 1 bit error



The corrupted data is closer to the correct data than to any other code word

An arithmetic code guarantees that
if inputs are code words and the operation is performed correctly results are code words too



Implementation of the arithmetic operation (hardware or software) must be modified to operate on the code

The set of code words of an arithmetic code A is closed with respect to a specific set of operations.

$$A(b * c) = A(b) * A(c) \text{ where } * \text{ is one of a set of operations}$$

3N codes

Multiply the data by 3 (this add 2 bits of redundancy)

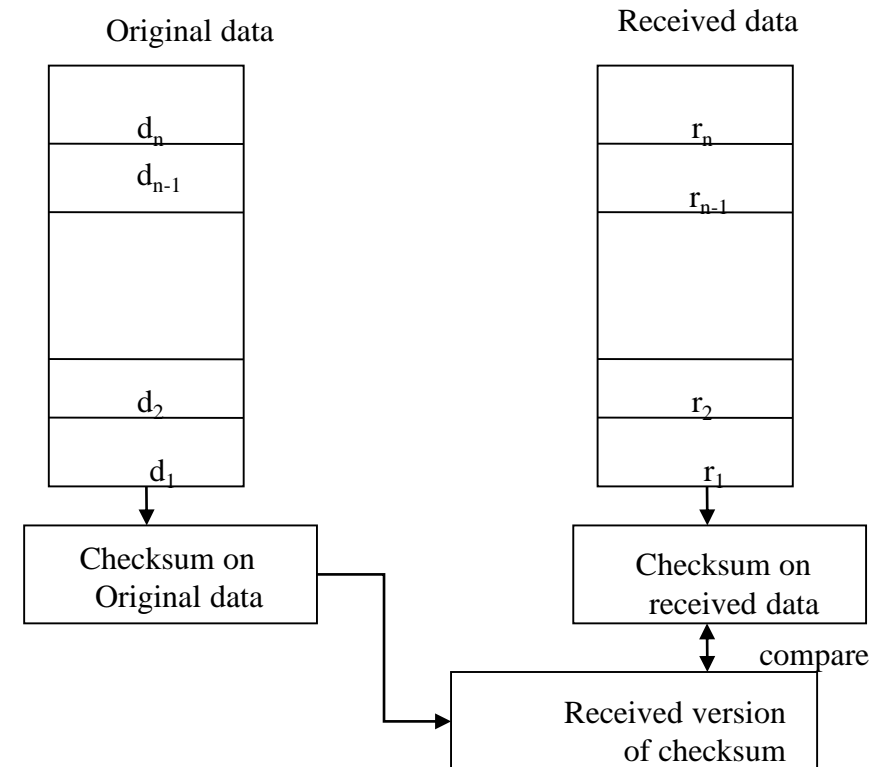
Error checking is performed by confirming that the received word is divisible by 3

Checksumming

- applied to large block of data in memories
- coverage: single fault

checksum for a block of n words is formed by adding together all of the words in the block modulo- k , where k is arbitrary (one of the least expensive method)

- the checksum is stored with the data block
- when blocks of data are transferred (e.g. data transfer between mass-storage device) the sum is recalculated and compared with the checksum
- checksum is basically the sum of the original data



Code word = block + checksum

- Disadvantages

- if any word in the block is changed, the checksum must also be modified at the same time
- allow error detection, no error location: the detected fault could be in the block of s words, the stored checksum or the checking circuitry
- single point of failures for the comparison and encoder/detector element

- Different methods differ for how summation is executed

Two-dimensional parity

| Odd parity | | n-bit words | row parity | |
|------------|---------------|--------------|------------|----------------|
| k words | | 1 0 1 ... 0 | 1 | |
| | | 0 0 1 ... 1 | 1 | ← parity error |
| | | 1 1 1 ... 0 | 0 | |
| | column parity | 1 0 0 ... 0 | 0 | |
| | | ↑ | | 0 |
| | | parity error | | |

Error location is possible for single-bit error:

one error in the row parity vector, one error in the column parity vector

A single-bit error in the parity column or parity row column is detected

Single-error correcting code (SEC): detect and correct 1-bit error

Hamming Code (I)



UNIVERSITÀ DI PISA

Parity bits spread through all the data word

| 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 | X | | X | | X | | X | | X | | X | | X | | X | | X | | X |
| | p2 | | X | X | | | X | X | | | X | X | | | X | X | | | X | X |
| | p4 | | | | X | X | X | X | | | | | X | X | X | X | | | | X |
| | p8 | | | | | | | | X | X | X | X | X | X | X | | | | | |
| | p16 | | | | | | | | | | | | | | | X | X | X | X | X |

Taken from: http://en.wikipedia.org/wiki/Hamming_code#Hamming_codes

Parity bits

all bit positions that are powers of two : 1, 2, 4, 8, etc.

Data bits

all other bit positions

(number the bit positions starting from 1: bit 1, 2, 3, etc..)

Parity bit p_j covers all bits whose position has the j least significant bit equal to 1

Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position

Hamming code (II)



UNIVERSITÀ DI PISA

| 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 | X | | X | | X | | X | | X | | X | | X | | X | | X | | X |
| | p2 | | X | X | | | X | X | | | X | X | | | X | X | | | X | X |
| | p4 | | | | X | X | X | X | | | | | X | X | X | X | | | | X |
| | p8 | | | | | | | | X | X | X | X | X | X | X | | | | | |
| | p16 | | | | | | | | | | | | | | | X | X | X | X | X |

Taken from: http://en.wikipedia.org/wiki/Hamming_code#Hamming_codes

Parity bit p1 covers all bit positions which have the least significant bit set:
bit 1 (the parity bit itself), 3, 5, 7, 9, etc.

Parity bit 4 covers all bit positions which have the third least significant bit set:
bits 4–7, 12–15, 20–23, etc.

Parity bit p2 covers all bit positions which have the second least significant bit set:
bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.

Parity bit 8 covers all bit positions which have the fourth least significant bit set:
bits 8–15, 24–31, 40–47, etc.

Hamming code (III)



UNIVERSITÀ DI PISA

| Parity bits | Total bits | Data bits | Name | Rate |
|-------------|------------|---------------|---------------------------------------|---------------------------|
| 2 | 3 | 1 | Hamming(3,1) (Triple repetition code) | $1/3 \approx 0.333$ |
| 3 | 7 | 4 | Hamming(7,4) | $4/7 \approx 0.571$ |
| 4 | 15 | 11 | Hamming(15,11) | $11/15 \approx 0.733$ |
| 5 | 31 | 26 | Hamming(31,26) | $26/31 \approx 0.839$ |
| ... | | | | |
| m | $2^m - 1$ | $2^m - m - 1$ | Hamming($2^m - 1, 2^m - m - 1$) | $(2^m - m - 1)/(2^m - 1)$ |

Taken from: http://en.wikipedia.org/wiki/Hamming_code#Hamming_codes

Overlap of control bit: a data bit is controlled by more than one parity bits

Minimum Hamming distance: 3

Double-error detection code
Single-error correction code



SEC-DED code

Necessity of reliance on the correct operation of **comparators** and **code checkers** that are used as hard-core for fault tolerant systems

Self-checking circuit

given a set of faults, a circuit that has the ability to automatically detect the existence of the fault and the detection occurs during the normal course of its operations

Typically obtained using coding techniques:

circuit inputs and outputs are encoded (also different codes can be used)

Basic idea:

- fault free + code input \rightarrow output: correct code word
- fault + code input \rightarrow output: (correct code word) or (non code word)

- **Self-testing circuit:** if, for every fault from the set, the circuit produces a noncode output for at least one code input (each single fault is detectable)
- **Fault-secure circuit:** if, for every fault from the set, the circuit never produces an incorrect code output for a code input (i.e. correct code output or noncode output)
- **Totally self-checking (TSC):** if the circuit is self-testing and fault-secure

two-input TSC comparator



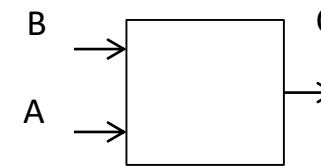
UNIVERSITÀ DI PISA

two signal input comparator (A, B)

output equal to 0 if inputs are equal; 1 otherwise

Fault assumption:

- single fault
- stuck-at-1/stuck-at-0 of each line in the circuit



| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Coding: 1/2

(dual-rail signal: coded signal whose two bits are always complementary)

A : A1 A2

0 : 0 1

1 : 1 0

two-input TSC comparator

output 0 if inputs are equal; 1 otherwise

Fault free

A = 0, B = 1 different input

m=1, n =1, q=0

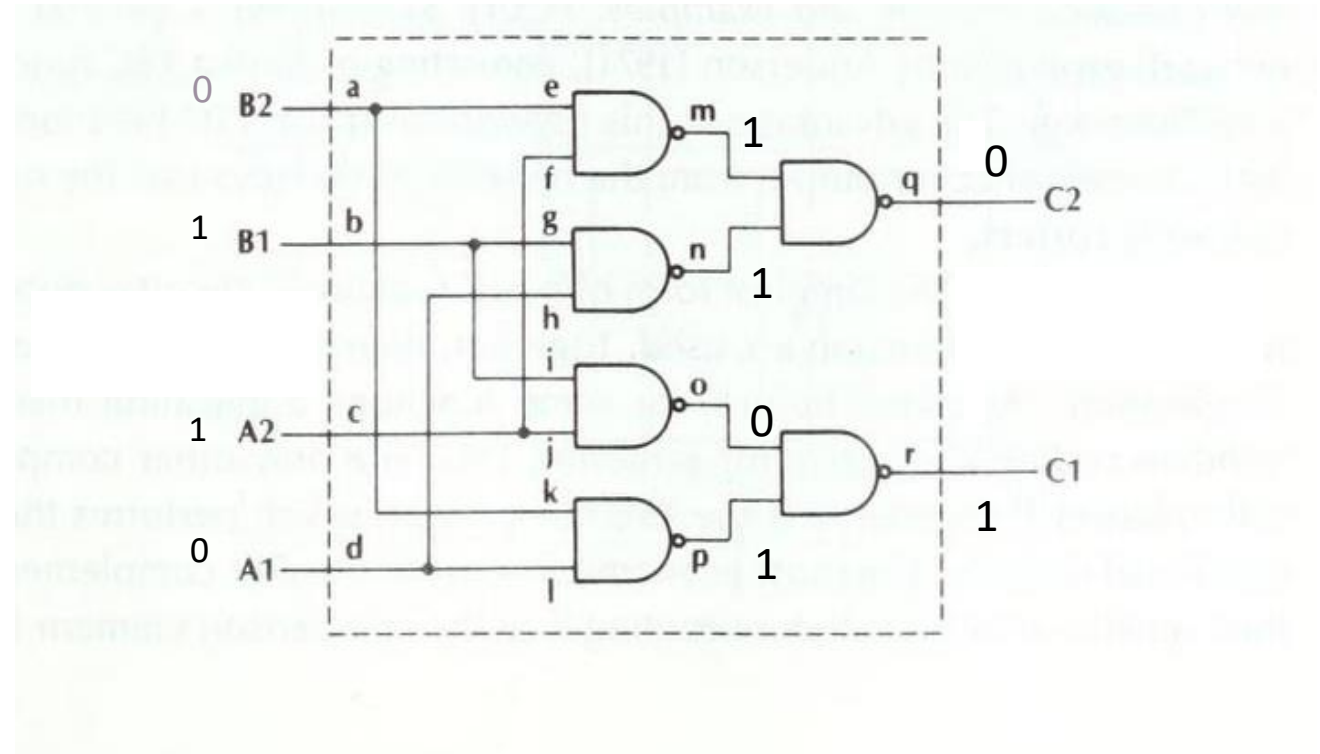
o = 0, p=1, r= 1

c2=0

c1=1

c1c2: code word

Output = c1 = 1 correct



Taken from:[Siewiorek et al., 1998]

two-input TSC comparator

output 0 if inputs are equal; 1 otherwise

Faulty:

A=0, B=1 different input

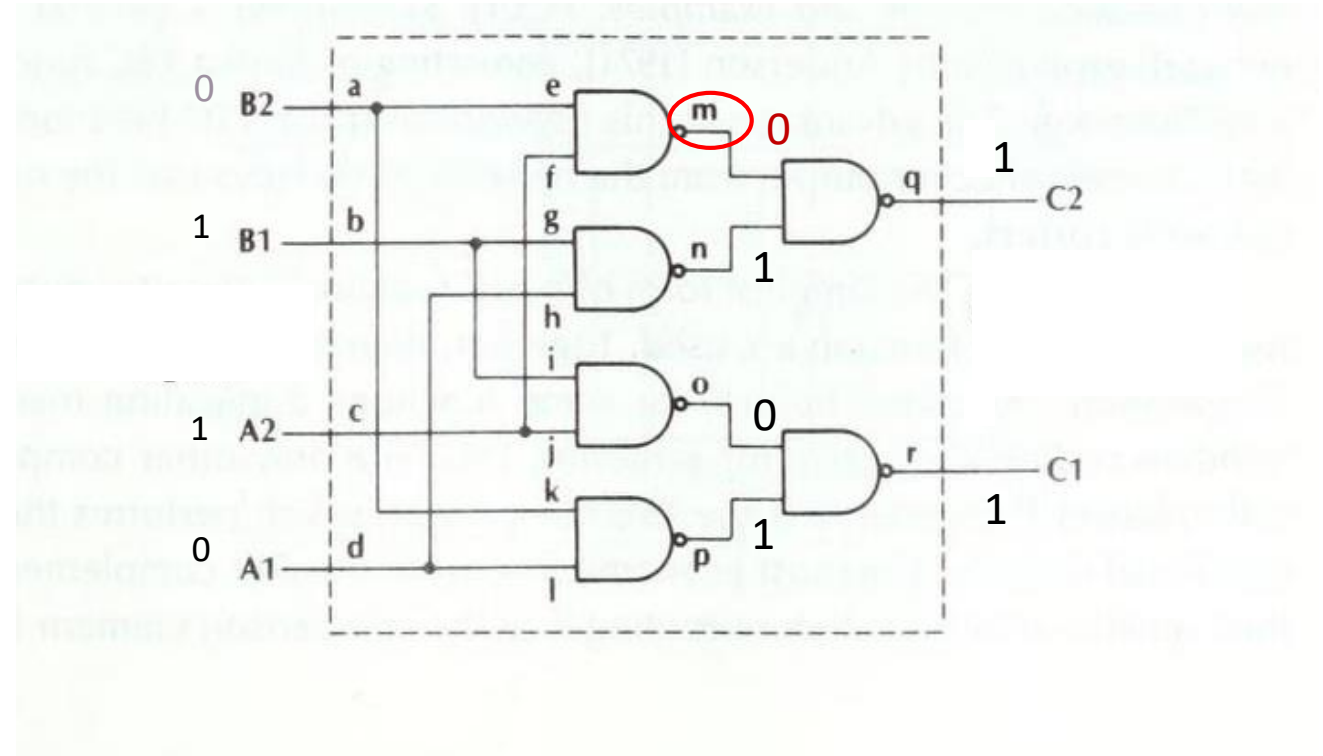
m: stuck-at-0

c2 = 1

c1 = 1

c1c2: non code word

Output = error



Taken from:[Siewiorek et al., 1998]

two-input TSC comparator

output 0 if inputs are equal; 1 otherwise

Faulty:

A=0, B=1 different input

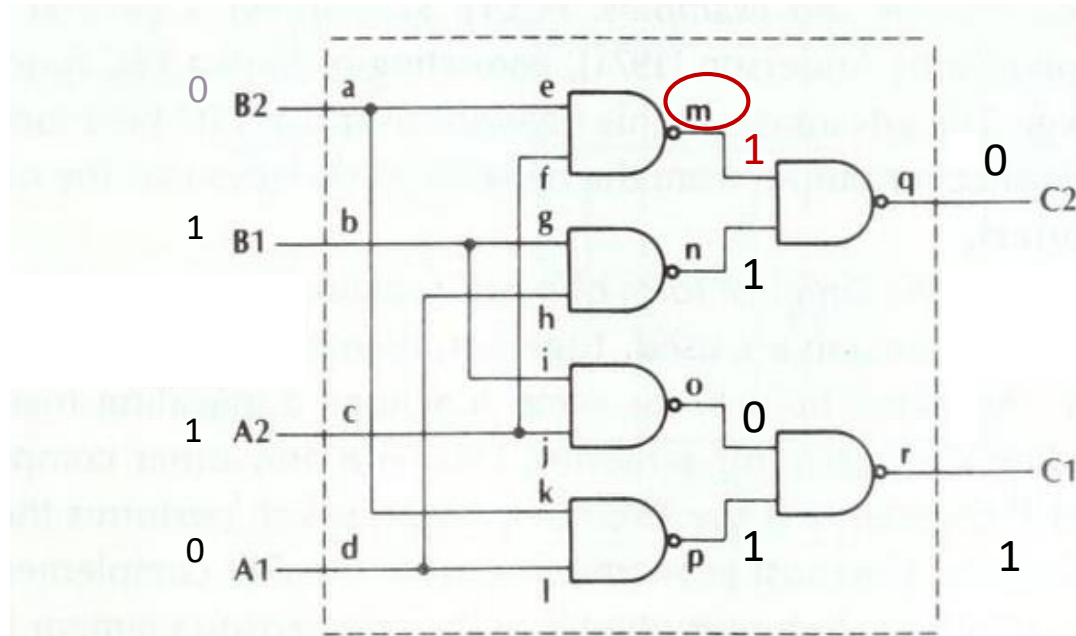
m: stuck-at-1

c2=0

c1=1

c1c2: code word

output = c1 = 1 correct



Taken from:[Siewiorek et al., 1998]

two-input TSC comparator

| Inputs | | | Outputs C2C1 Resulting from Single Stuck-at-1 Faults | | | | | | | | | | | | | | | | | |
|--------|------|---------------|------------------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| B2B1 | A2A1 | Normal Output | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r |
| 01 | 01 | 10 | 11 | 10 | 11 | 10 | 10 | 10 | 10 | 10 | 10 | 11 | 11 | 10 | 10 | 00 | 10 | 10 | 10 | 11 |
| 01 | 10 | 01 | 11 | 01 | 01 | 11 | 11 | 01 | 01 | 11 | 01 | 01 | 01 | 01 | 01 | 01 | 00 | 01 | 11 | 01 |
| 10 | 01 | 01 | 01 | 11 | 11 | 01 | 01 | 11 | 11 | 01 | 01 | 01 | 01 | 01 | 01 | 01 | 01 | 00 | 11 | 01 |
| 10 | 10 | 10 | 10 | 11 | 10 | 11 | 10 | 10 | 10 | 10 | 11 | 10 | 10 | 11 | 00 | 10 | 10 | 10 | 10 | 11 |

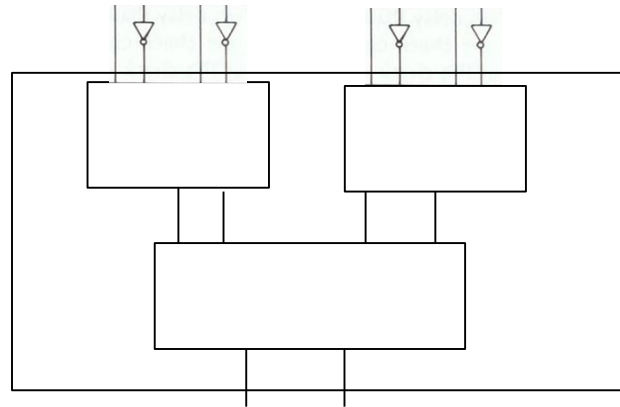
| Inputs | | | Outputs C2C1 Resulting from Single Stuck-at-0 Faults | | | | | | | | | | | | | | | | | |
|--------|------|---------------|------------------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| B2B1 | A2A1 | Normal Output | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r |
| 01 | 01 | 10 | 10 | 00 | 10 | 00 | 10 | 10 | 00 | 00 | 10 | 10 | 10 | 10 | 10 | 10 | 11 | 11 | 00 | 10 |
| 01 | 10 | 01 | 01 | 00 | 00 | 01 | 01 | 01 | 01 | 01 | 00 | 00 | 01 | 01 | 11 | 11 | 01 | 01 | 01 | 00 |
| 10 | 01 | 01 | 00 | 01 | 01 | 00 | 01 | 01 | 01 | 01 | 01 | 01 | 00 | 00 | 11 | 11 | 01 | 01 | 01 | 00 |
| 10 | 10 | 10 | 00 | 10 | 00 | 10 | 00 | 00 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 11 | 11 | 00 | 10 |

Taken from:[Siewiorek et al., 1998]

- For each fault, there exists at least one input configuration such that the output is a non code word
- If the output is a code word, the output is correct

n-input TSC comparator

- n-input TSC comparator:
tree of two input self checking comparators



TIME REDUNDANCY

Attempt to reduce the amount of extra hw at the expense of using additional time

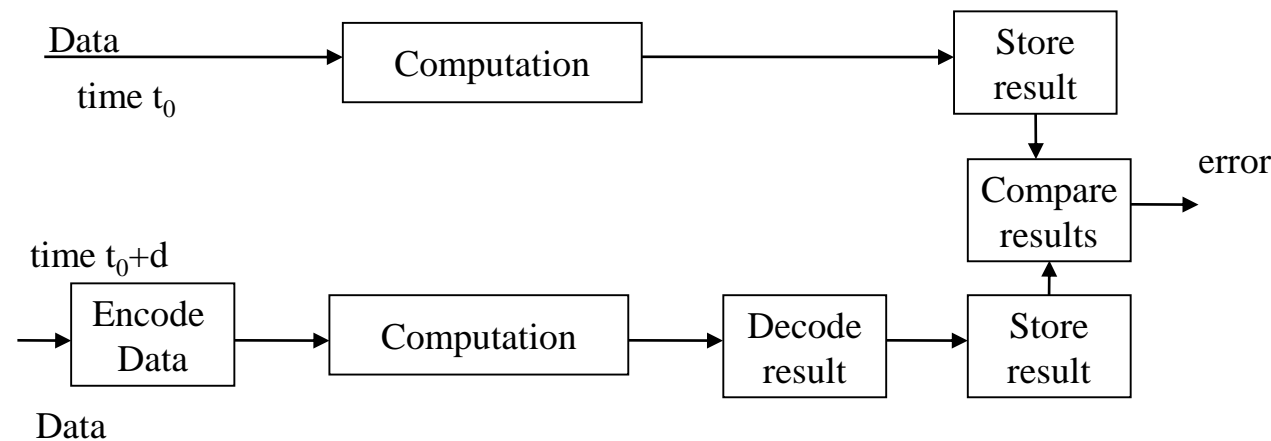
- Repetition of computations

- compare the results to detect faults
- re-execute computations (disagreement disappears or remains)

good for transient faults

no protection against permanent fault

- problem of guaranteeing the same data when a computation is executed



Time redundancy techniques (II)



UNIVERSITÀ DI PISA

- Use a minimum of extra hw to detect also permanent faults
 - encode data before executing the second computation

Example: data transmitted over a parallel bus

- stuck at of a line of the bus

t0: transmit original data

t0+d : transmit complement data

When a fault occurs: received data not complements of each other

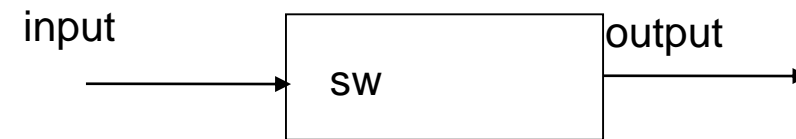
↓ line stuck at 0

| | | | |
|--------|---------|----|---------|
| t0 : | 1 0 1 1 | -> | 1 0 0 1 |
| t0+d : | 0 1 0 0 | -> | 0 1 0 0 |

Transmission error free, each bit line alternate between a logic 1 and a logic 0 (*alternating logic*)

SOFTWARE REDUNDANCY

Software is subject to



design flaws:

- mistakes in the **interpretation of the specification** that the software is supposed to satisfy (ambiguities)
- mistakes in the **implementation of the specification**: carelessness or incompetence in writing code, inadequate testing

operational faults:

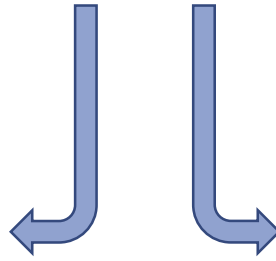
incorrect or unexpected usage faults (operational profile)

Design flaws:

hard to visualize, classify, detect, and correct.

closely related to human factors and the design process, of which we don't have a solid understanding

only some type of inputs will exercise that fault to cause failures. Number of failures depend on how often these inputs exercise the sw flaw



apparent reliability of a piece of software is correlated to how frequently design faults are exercised as opposed to number of design faults present

Due to the large cost of developing software, most of the software dependability effort has focused on
fault prevention techniques and testing strategies

Multi-version approaches

replicate the software
mainly used in safety-critical systems (due to cost)

Software diversity

a simple duplication and comparison procedure will not detect software faults if the duplicated software modules are identical

Independent generation of $N \geq 2$ functionally equivalent programs, called *versions*, **from the same initial specification**.

Upon disagreement among the versions?

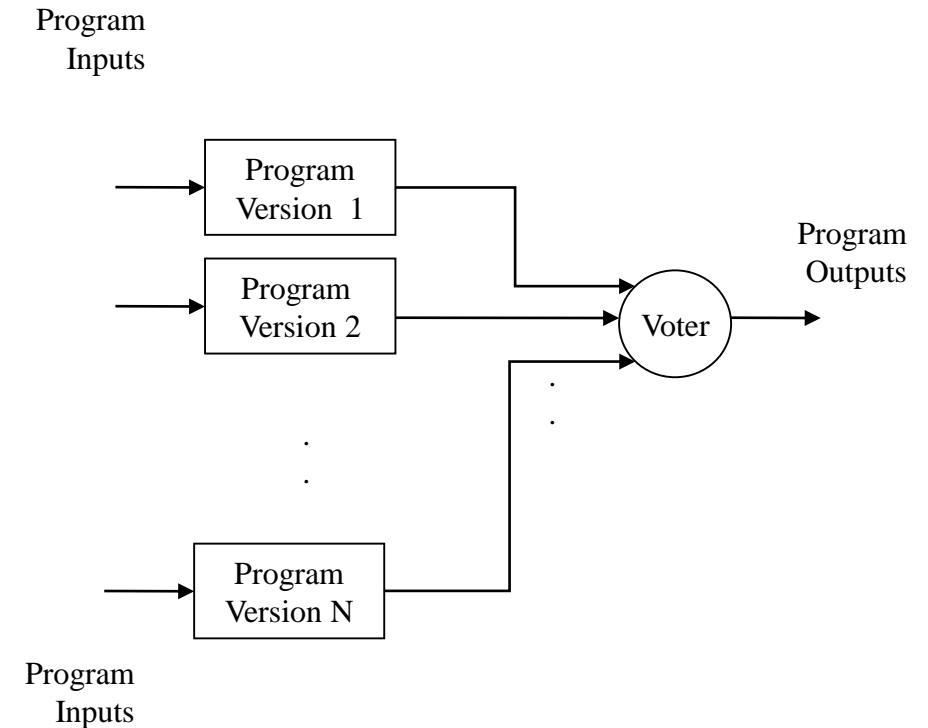
- retry or restart (fault containment)
- transition to a predefined safe state
- reliance on one of the versions

N-version programming (I)

independently developed versions
of **design** and **code**

Technique: independent
design teams using different design
methodologies, algorithms, compilers,
run-time systems and hardware
components

- vote on the N results produced



Disadvantages

- cost of software development
- cost of concurrent executions
- potential source of correlated errors, such as the original specification.

Specification mistakes: not tolerated

Practical problem

in implementing the software Voter for comparing the results generated by the copies because of the differences in compilers, numerical techniques and format conversions.

Software voter (single point of failure)

- not replicated: must be simple and verifiable
- must assure that the input data vector to each of the versions is identical
- must receive data from each version in identical formats or make efficient conversions must implement some sort of communication protocol to wait until all versions complete their processing or recognize the versions that do not complete

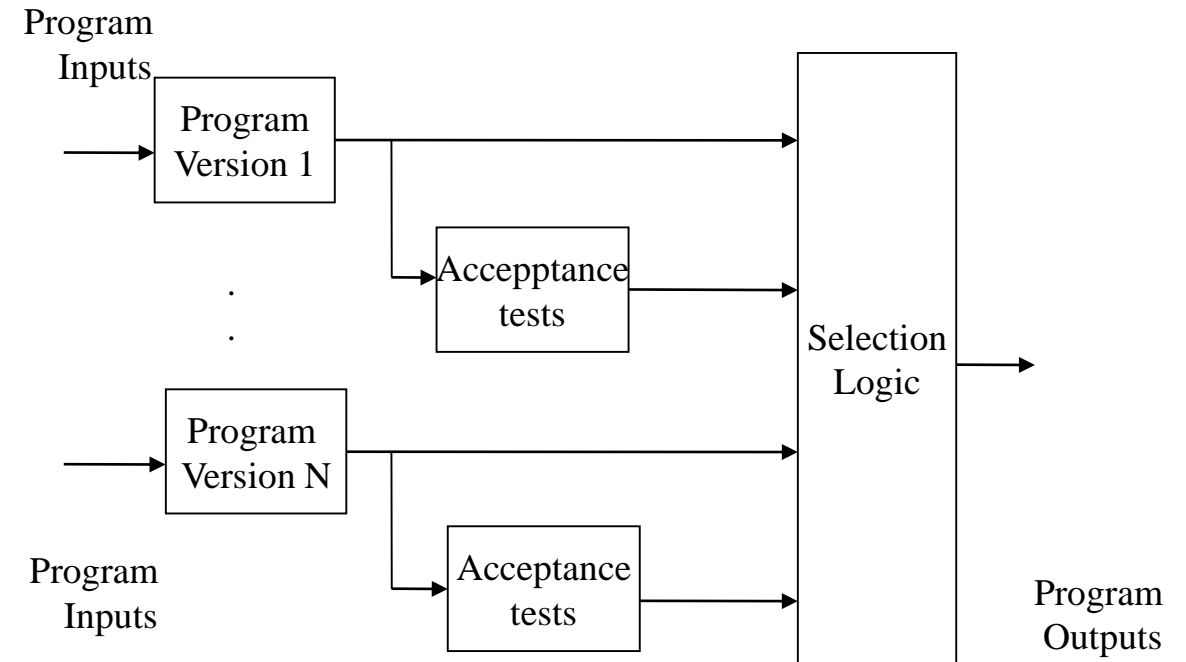
N-self-checking programming

based on acceptance tests rather than comparison with equivalent versions

N versions of the program are written
- each version is running simultaneously
and includes its acceptance tests

The selection logic chooses the results
from one of the programs that passes
the acceptance tests

Tolerates N-1 faults (independent faults)



Design diversity

1. Cannot adopt the hardware analogy and assume versions fail independently
2. Empirical evidence that there will be common faults
There is evidence that diversity delivers some improvement over single versions
3. Related faults may result from dependencies in the separate designs and implementations (example: specification mistakes)

Functional diversity

Assign to independent software versions diverse functions that compute the same task

For example, in a plant, diverse measurement signals, actuators and functions exists to monitoring the same phenomenon

Diverse functions

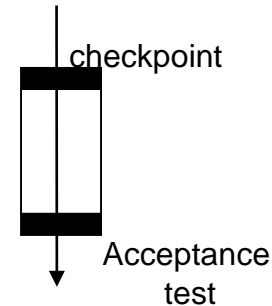
for example, functions that ensure independently that the plant safety targets are met.

Recovery-block technique

based on an one acceptance test and a single alternate is run at a time

Basic structure:

Ensure T
By P
else by Q
else error



Acceptability of the result is decided by an acceptance test **T**. Primary alternate **P**, secondary alternates **Q**

1. variables global to the block automatically checkpointed if they are altered within the block
2. the primary alternate is executed and subjected to an acceptance test to detect any error in the result. If the test is passed, the block is exited; otherwise the content of the recovery cache pertinent to the block is reinstated, and the second alternate is executed.
3. This cycle is executed until either an alternative is successful or no more alternatives exist. In this last case an error is reported.

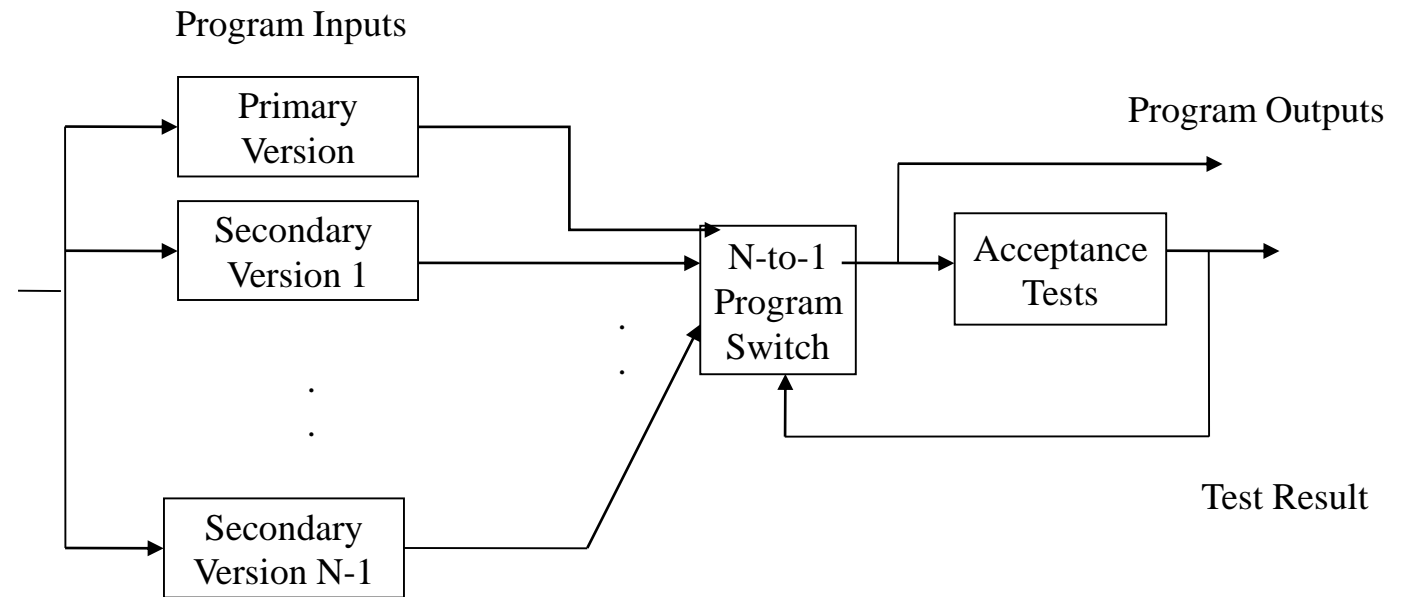
Recovery-block technique



UNIVERSITÀ DI PISA

Combines elements of checkpointing and backup

- checkpoint: a copy of the current state for possible use in fault tolerant techniques
- releases the programmer from determining which variables should be checkpointed and when
- linguistic structure for recovery blocks requires a suitable mechanism for providing automatic backward error recovery.



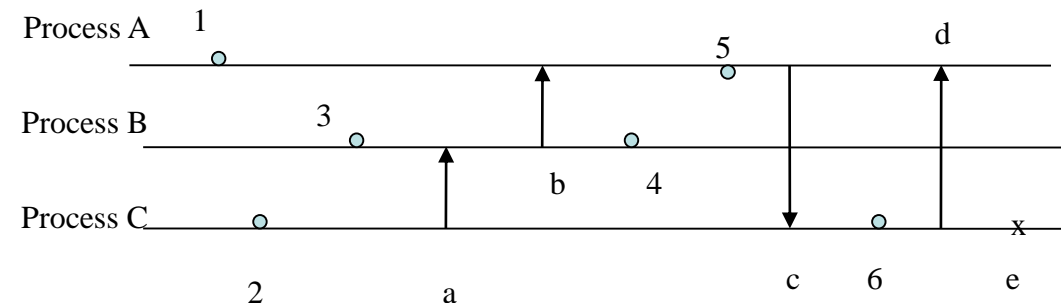
Checkpointing: basic issues

- may be taken automatically (periodically) or upon request by program
- resetting the system and process state to the state stored at the latest checkpoint needs mechanisms in run-time support (rollback)

- need to be correct
(consistency of checkpoints)

- need eventually to be discarded

- overhead of saving system state
(minimize the amount of state information that must be saved)



○ Checkpoint

x Error

→ Message passed

domino effect

Example: Stable storage

RAID (Redundant Arrays of Independent Disks) technology
disk organization techniques that manage a large numbers of disks,
providing a view of a single disk of high capacity and high speed by
using multiple disks in parallel, and high reliability by storing data
redundantly

Read failure

To deal with read failure, computes and attaches **checksums** to each sector to verify that data is read back correctly

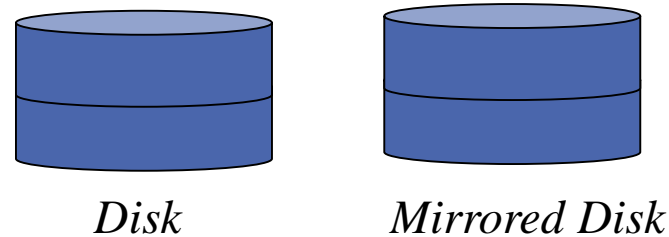
If data is corrupted, with very high probability stored checksum won't match recomputed checksum

Write failure

Ensure successful writing by reading back sector after writing it

Redundant information stored on multiple disks to recover from failures

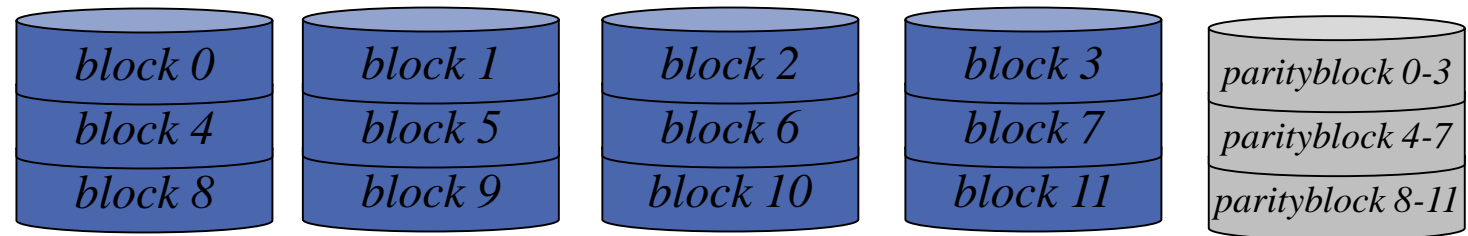
- Mirroring



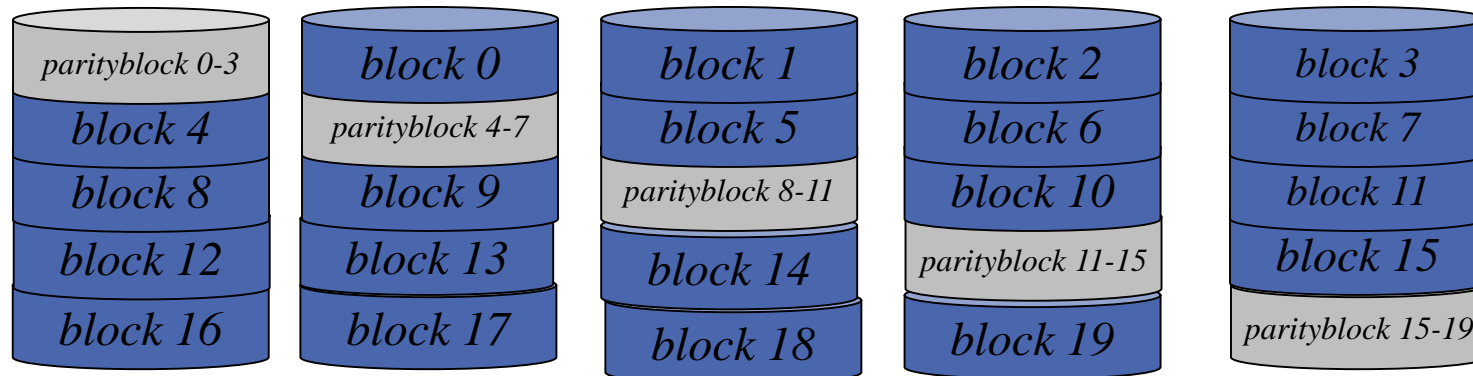
- Coding: Block-Interleaved Parity

Block-level striping

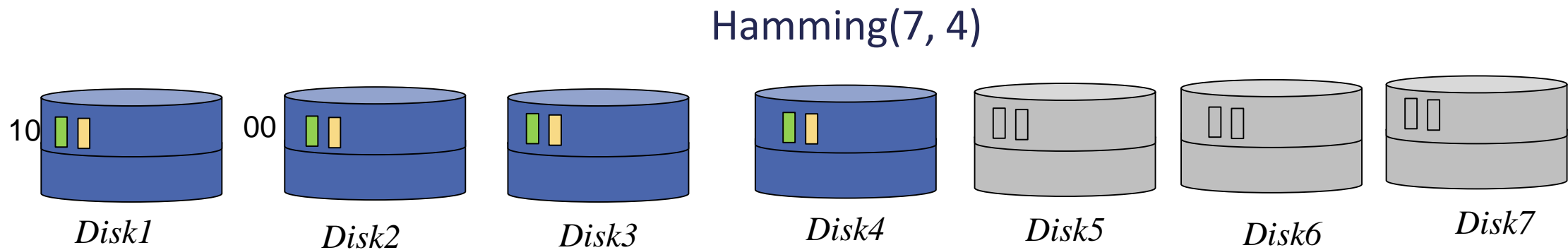
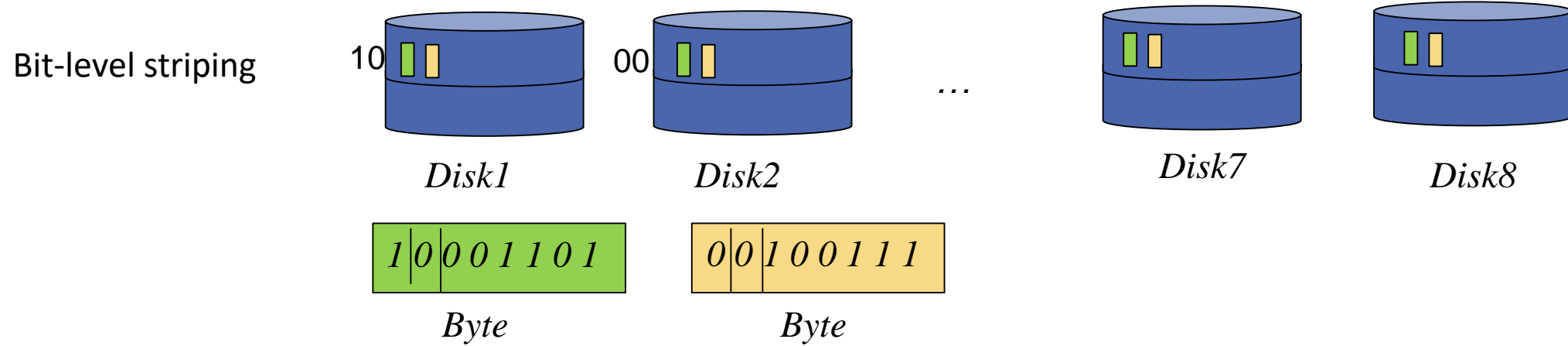
Parity block on a different disk for tolerating disk failure



- Coding: Block-Interleaved Distributed Parity



- Coding: Hamming code



- Fault tolerance uses **replication for error detection** and system recovery
- Fault tolerance relies on the **independency of redundancies** with respect to the process of fault creation and activations
- **Fault masking** will conceal a possibly progressive and eventually fatal loss of protective redundancy
- Practical implementations of masking generally involve error detection (and possibly fault handling), leading to **masking and error detection and recovery**

- When tolerance to physical faults is foreseen, the channels may be identical, based on the assumption that hardware components fail **independently**
- When tolerance to design faults is foreseen, channels have to provide identical service through separate designs and implementation (through **design diversity**)