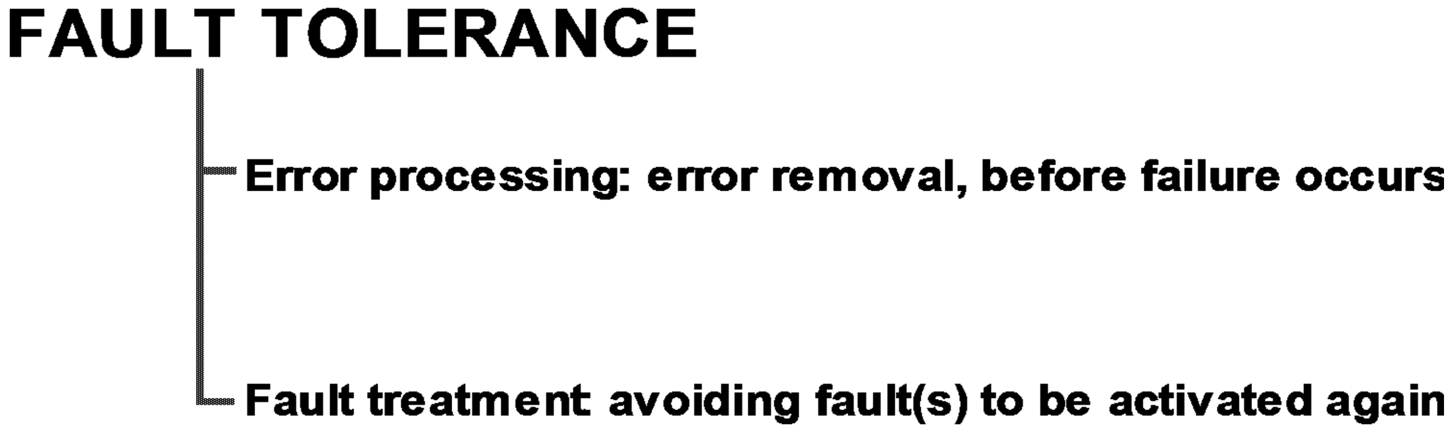
Fault-tolerant design techniques

slides made with the collaboration of: Laprie, Kanoon, Romano

Fault Tolerance – Key Ingredients



FAULT TOLERANCE

- Error processing: error removal, before failure occurs
- Fault treatment avoiding fault(s) to be activated again

Error Processing

ERROR PROCESSING

- **Error detection:** identification of erroneous state(s)
- **Error diagnosis**: damage assessment
- **Error recovery:** error-free state substituted to erroneous state
  - **Backward recovery**: system brought back in state visited before error occurrence

    $\downarrow$

    Recovery points (checkpoint)
  - **Forward recovery:** Erroneous state is discarded and correct one is determined Without losing any computation.

Fault Treatment

FAULT TREATEMENT

- **Fault diagnosis**

  determination of error causes

- **Fault isolation**

  removing faulty components from subsequent execution process

♍ system no longer able to deliver same service

↓

Reconfiguration

Modification of system structure, such that nor failed components deliver degraded service

Fault Tolerant Strategies

- Fault tolerance in computer system is achieved through redundancy in hardware, software, information, and/or time.
  Such redundancy can be implemented in static, dynamic, or hybrid configurations.

- Fault tolerance can be achieved by the following techniques:
  - **Fault masking** is any process that prevents faults in a system from introducing errors. Example: Error correcting memories and majority voting.
  - **Reconfiguration** is the process of eliminating faulty component from a system and restoring the system to some operational state.

Reconfiguration Approach

- **Fault detection** is the process of recognizing that a fault has occurred. Fault detection is often required before any recovery procedure can be initiated.
- **Fault location** is the process of determining where a fault has occurred so that an appropriate recovery can be initiated.
- **Fault containment** is the process of isolating a fault and preventing the effects of that fault from propagating throughout the system.
- **Fault recovery** is the process of regaining operational status via reconfiguration even in the presence of faults.

The Concept of Redundancy

- *Redundancy* is simply the addition of information, resources, or time beyond what is needed for normal system operation.
- **Hardware redundancy** is the addition of extra hardware, usually for the purpose either detecting or tolerating faults.
- **Software redundancy** is the addition of extra software, beyond what is needed to perform a given function, to detect and possibly tolerate faults.
- **Information redundancy** is the addition of extra information beyond that required to implement a given function; for example, error detection codes.

The Concept of Redundancy (Cont’d)

- **Time redundancy** uses additional time to perform the functions of a system such that fault detection and often fault tolerance can be achieved. *Transient faults* are tolerated by this approach.

The use of redundancy can provide additional capabilities within a system. But, redundancy can have very important impact on a system's performance, size, weight and power consumption.

HARDWARE REDUNDANCY

Hardware Redundancy

- **Static techniques** use the concept of fault masking. These techniques are designed to achieve fault tolerance without requiring any action on the part of the system. Relies on voting mechanisms.

  (also called passive redundancy or fault-masking)

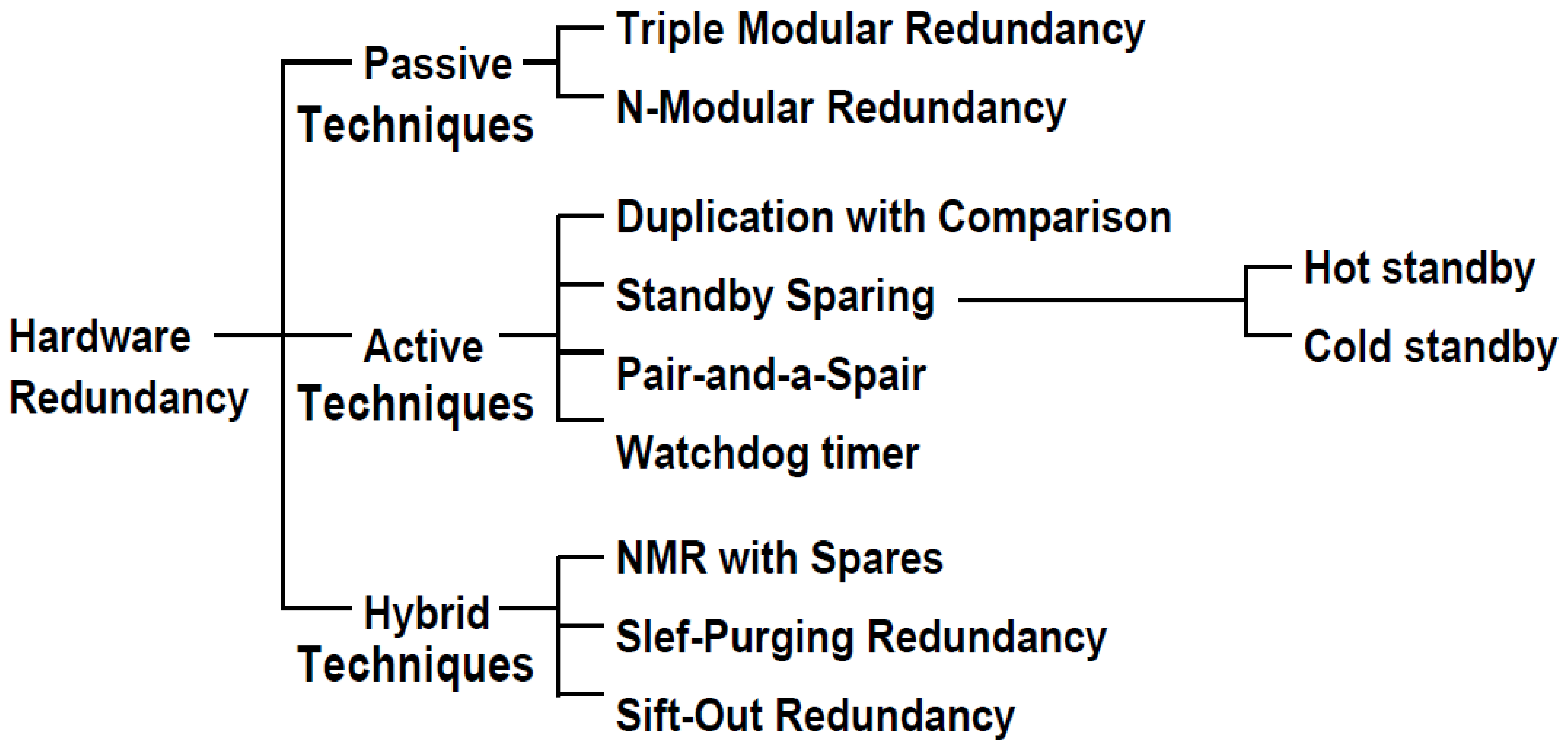
- **Dynamic techniques** achieve fault tolerance by detecting the existence of faults and performing some action to remove the faulty hardware from the system. That is, active techniques use fault detection, fault location, and fault recovery in an attempt to achieve fault tolerance.

  (also called active redundancy)

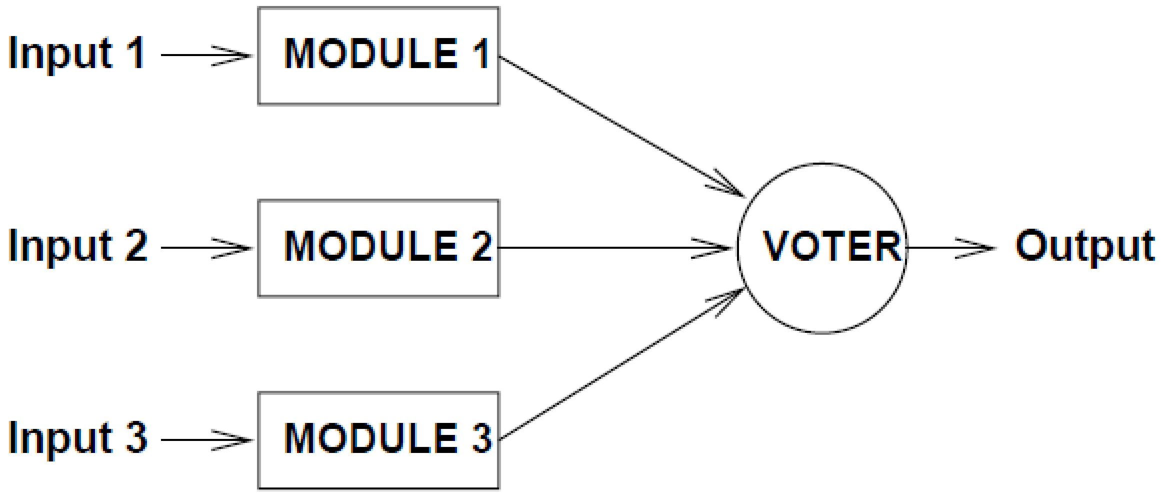
Hardware Redundancy (Cont’d)

- **Hybrid techniques** combine the attractive features of both the passive and active approaches.
  - Fault masking is used in hybrid systems to prevent erroneous results from being generated.
  - Fault detection, location, and recovery are also used to improve fault tolerance by removing faulty hardware and replacing it with spares.

Hardware Redundancy - A Taxonomy



Triple Modular Redundancy (TMR)



Masks failure of a single component.
Voter is a SINGLE POINT OF FAILURE.

Reliability of TMR

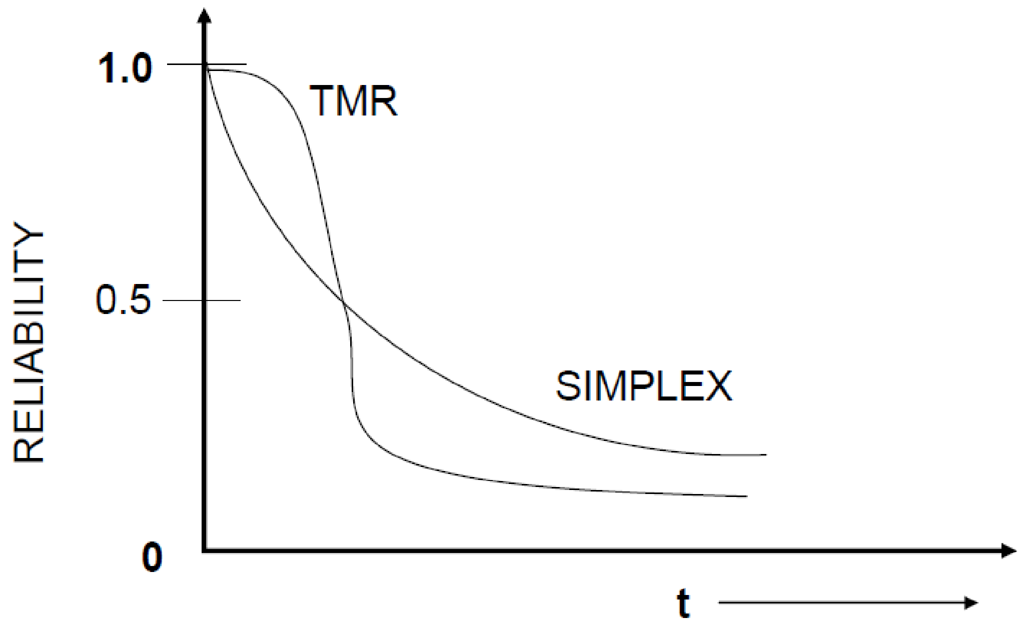
- Ideal Voter ($R_V(t)=1$)

$$R_{SYS}(t)=R_M(t)^3+3R_M(t)^2[1-R_M(t)]=3R_M(t)^2-2R_M(t)^3$$

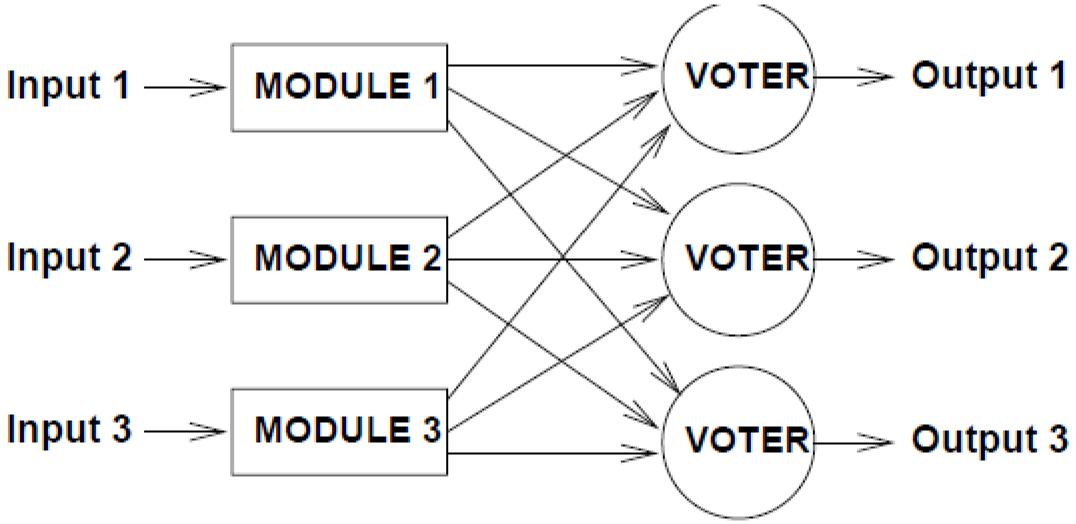
- Non-ideal Voter

$$R'_{SYS}(t)=R_{SYS}(t)R_V(t)$$

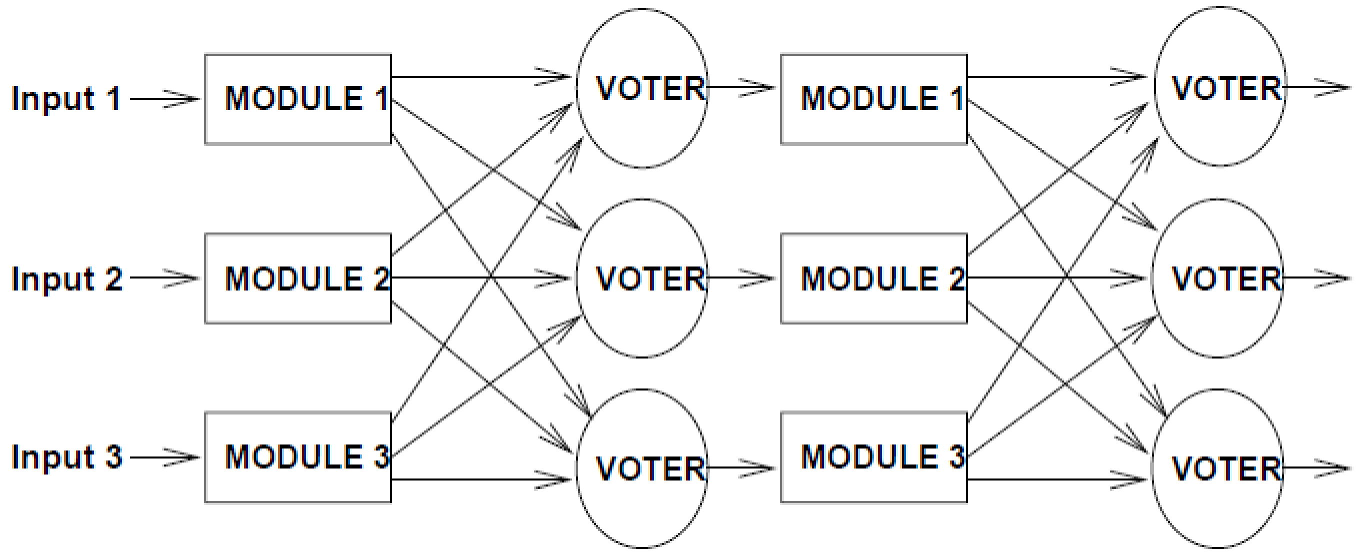
- $R_M(t)=e^{-\lambda t}$
- $R_{SYS}(t)=3\,e^{-2\lambda t}-2e^{-3\lambda t}$



TMR with Triplicate Voters



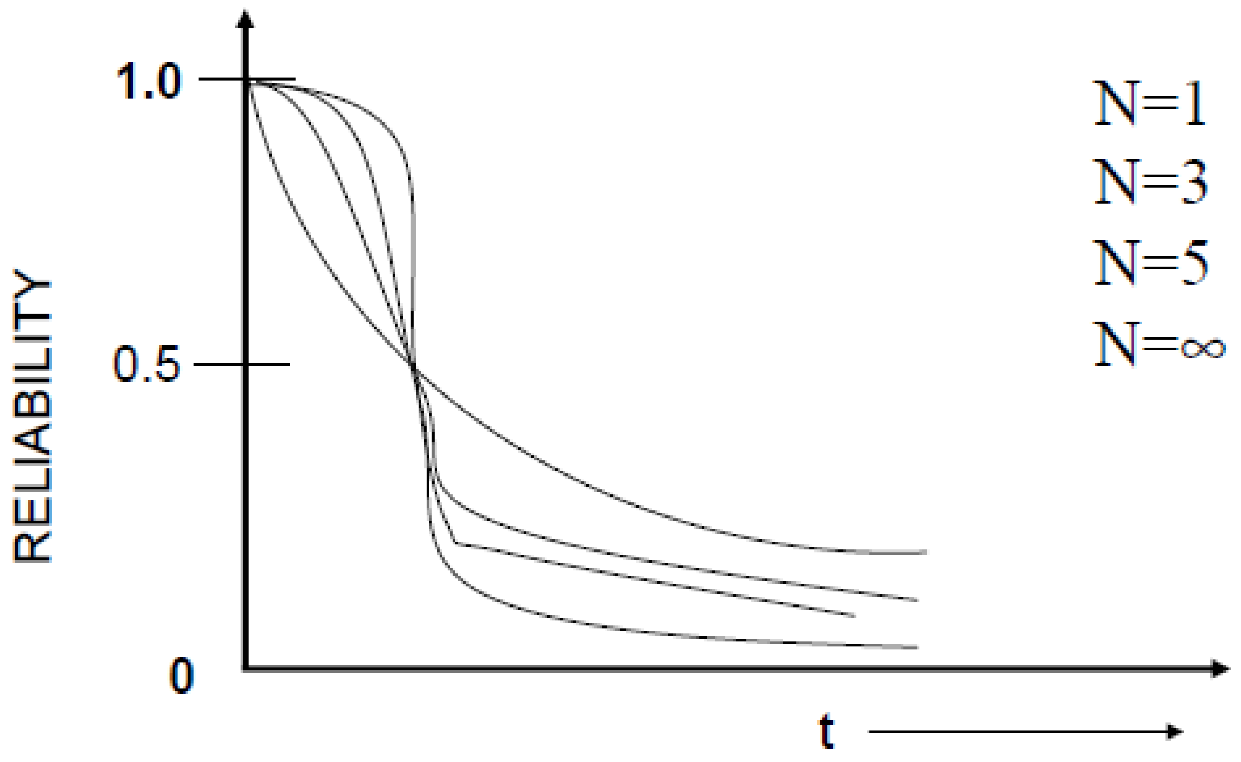
Multistage TMR System



N-Modular Redundancy (NMR)

- Generalization of TMR employing N modules rather than 3.
- PRO:
  - If N>2f, up to f faults can be tolerated:
    - e.g. 5MR allows tolerating the failures of two modules
- CON:
  - Higher cost wrt TMR

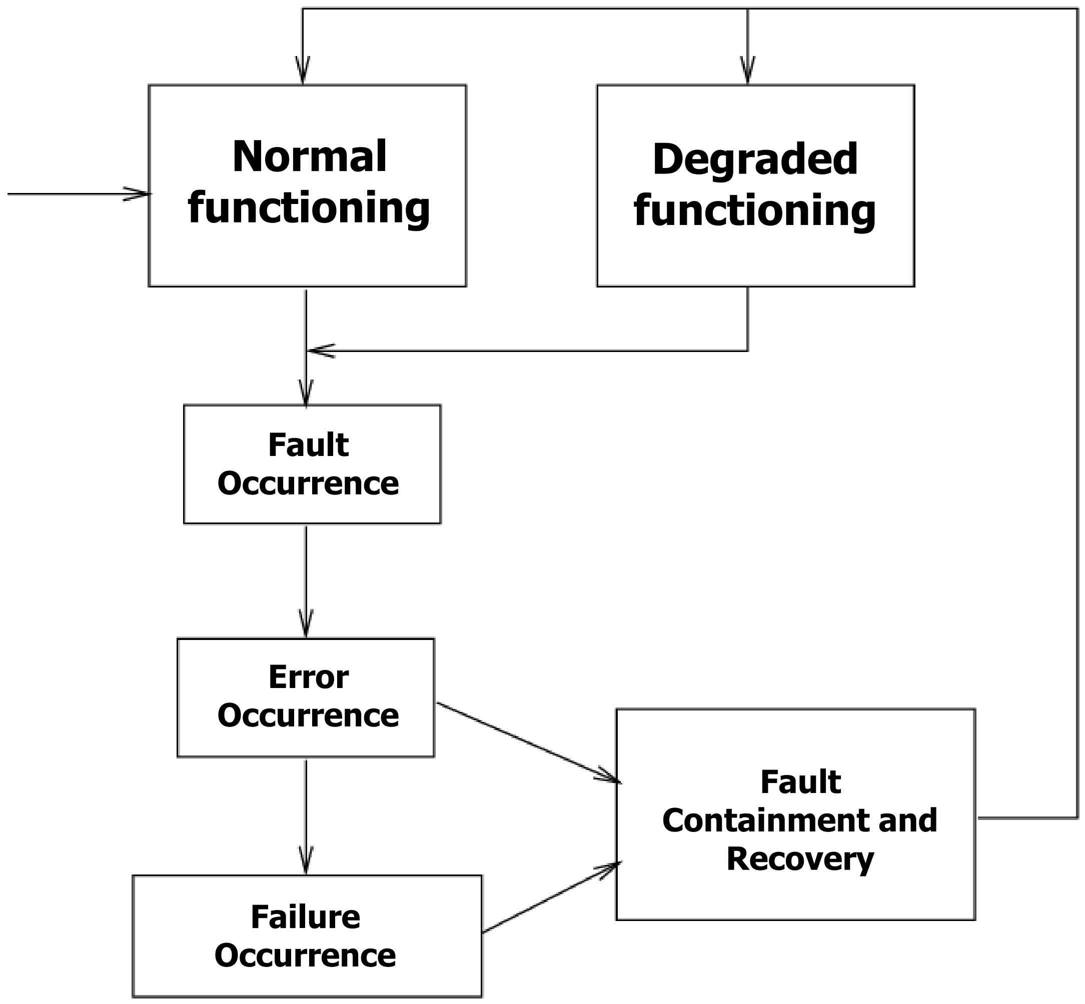
Reliability Plot



Hardware vs Software Voters

- The decision to use hardware voting or software voting depends on:
  - The availability of processor to perform voting.
  - The speed at which voting must be performed.
  - The criticality of space, power, and weight limitations.
  - The flexibility required of the voter with respect to future changes in the system.
- Hardware voting is faster, but at the cost of more hardware.
- Software voting is usually slow, but no additional hardware cost.

Dynamic (or active) redundancy



Standby Sparing

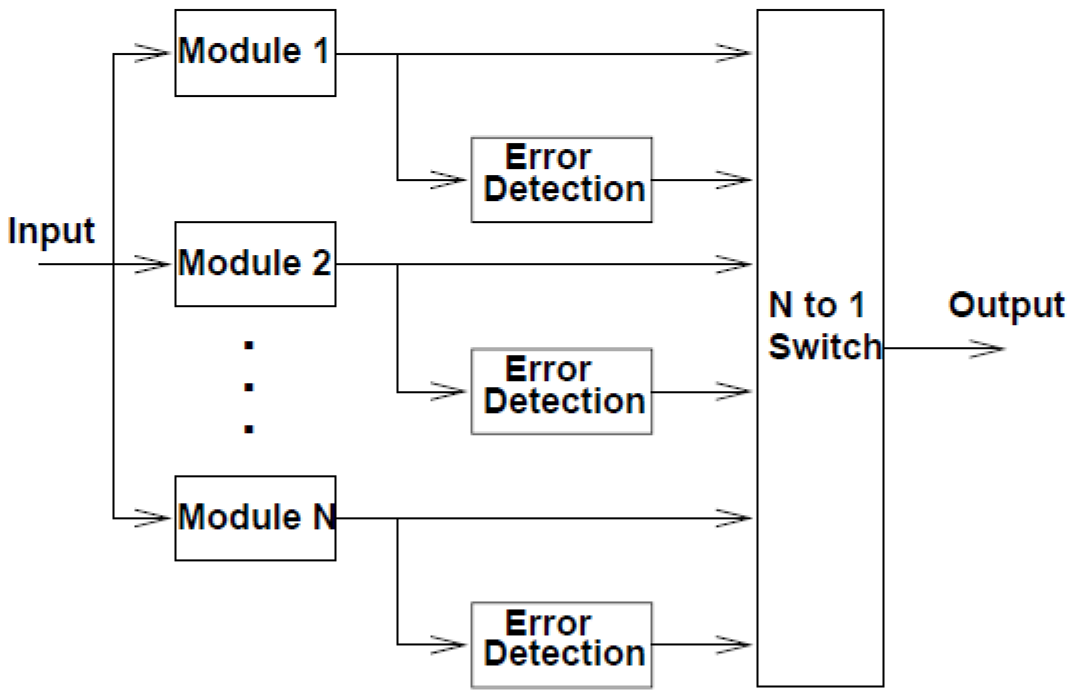
- In standby sparing, one module is operational and one or more modules serve as standbys or spares.
- If a fault is detected and located, the faulty module is removed from the operation and replaced with a spare.
- **Hot standby sparing:** the standby modules operate in synchrony with the online modules and are prepared to take over any time.
- **Cold standby sparing**: the standby modules are unpowered until needed to replace a faulty module. This involves momentary disturbance in the service.

Standby Sparing (Cont’d)

- Hot standby is used in applications such as process control where the reconfiguration time needs to be minimized.
- Cold standby is used in applications where power consumption is extremely important.
- The key advantage of standby sparing is that a system containing n identical modules can often provide fault tolerance capabilities with significantly fewer power consumption than n redundant/parallel modules.

Standby Sparing (Cont'd)

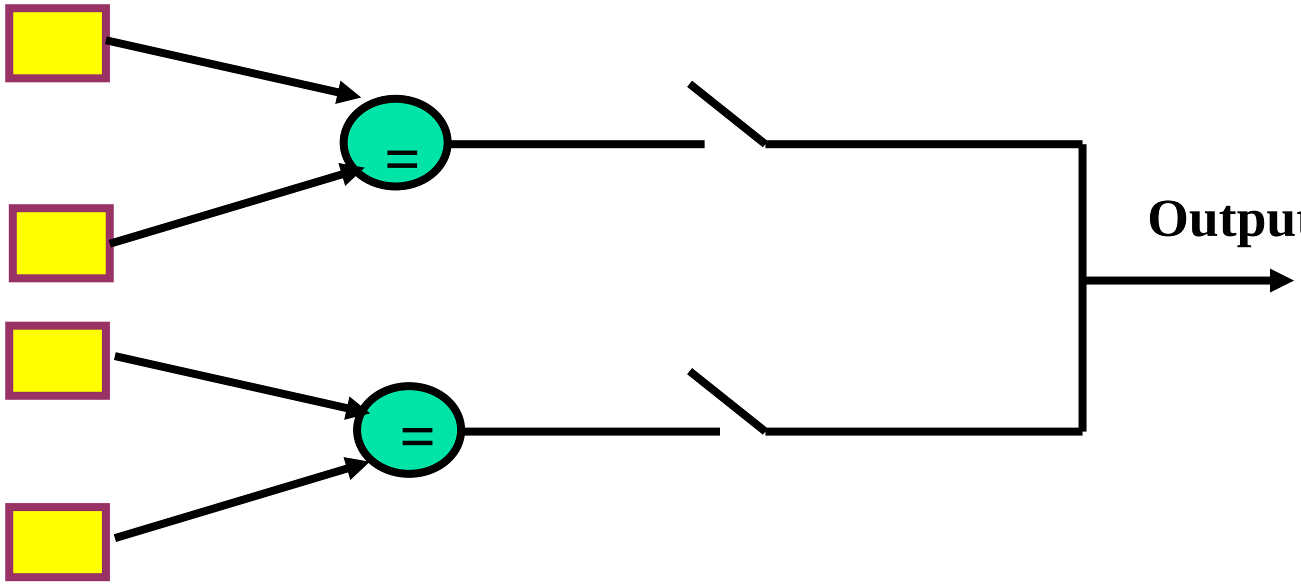
- Here, one of the N modules is used to provide system's output and the remaining (N-1) modules serve as spares.



Pair-and-a-Spare Technique

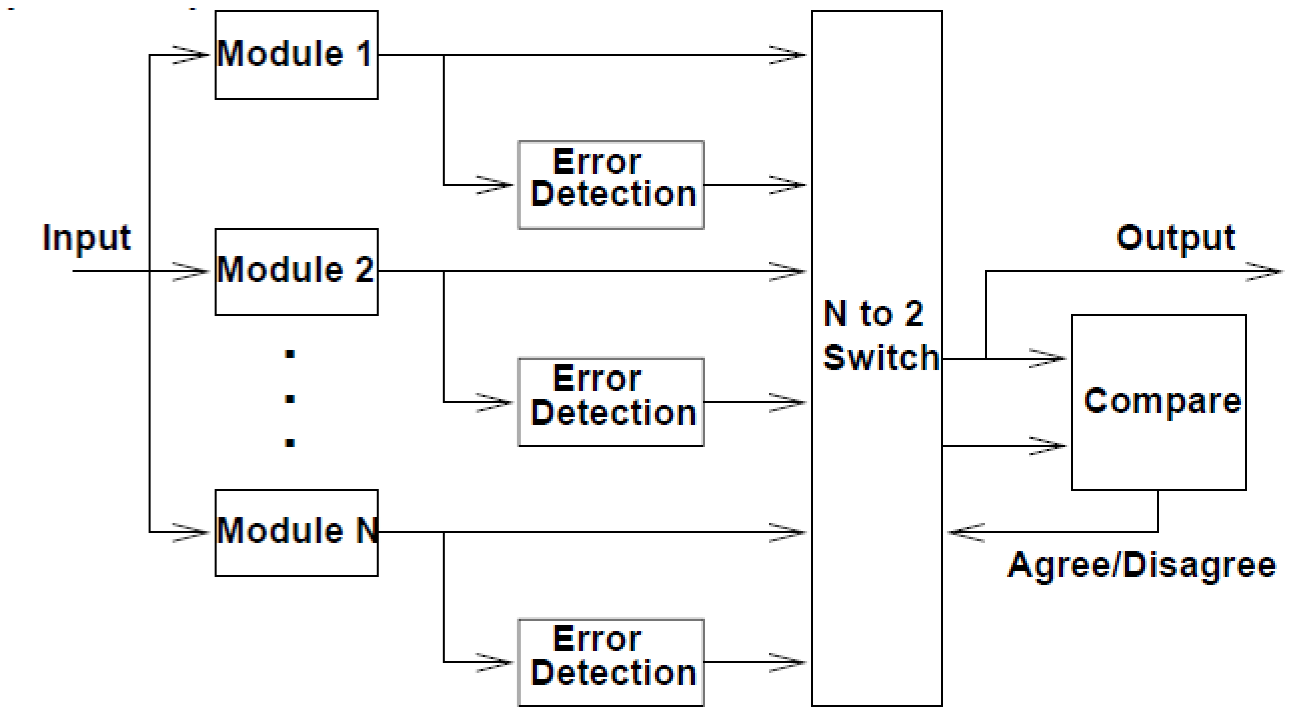
- Pair-and-a-Spare technique combines the features present in both standby sparing and duplication with comparison.
- Two modules are operated in parallel at all times and their results are compared to <u>provide the error detection capability required in the standby sparing approach</u>.
- second duplicate (pair, and possibly more in case of pair and k-spare) is used to take over in case the working duplicate (pair) detects an error
- a pair is always operational

Pair-and-a-Spare Technique (Cont’d)



Pair-and-a-Spare Technique (Cont'd)

- Two modules are always online and compared, and any spare replace either of the online modules.



Mettere figura di impianto…….

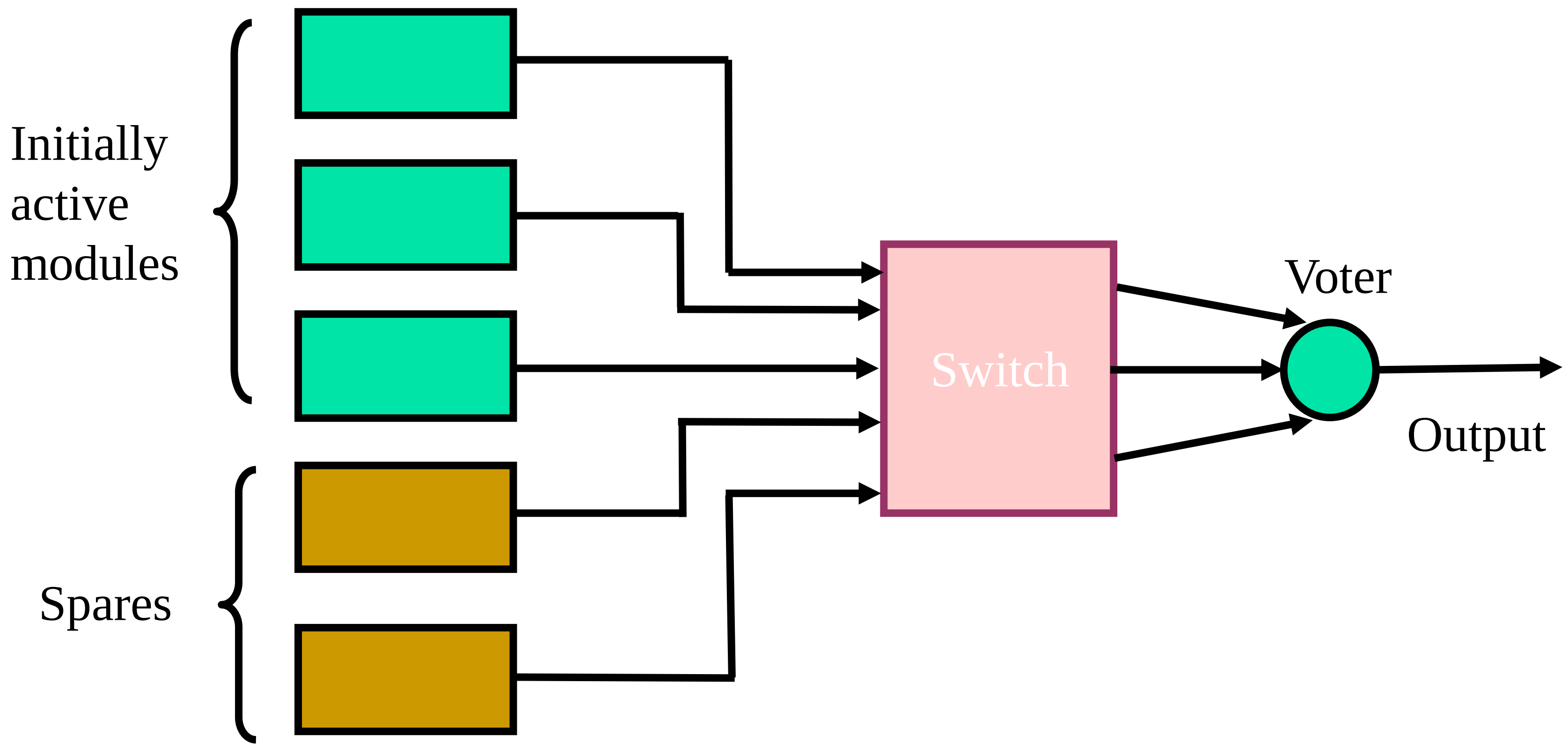
Watchdog Timers

- The concept of a watchdog timer is that the lack of an action is an indicative of fault.
- A watchdog timer is a timer that must be reset on a repetitive basis.
- The fundamental assumption is that the system is fault free if it possesses the capability to repetitively perform a function such as setting a timer.
- The frequency at which the timer must be reset is application dependent.
- A watchdog timer can be used to detect faults in both the hardware and the software of a system.

Hybrid redundancy

- Hybrid hardware redundancy
  - Key - combine passive and active redundancy schemes
  - NMR with spares
    - example - 5 units
      - 3 in TMR mode
      - 2 spares
      - all 5 connected to a switch that can be reconfigured
    - comparison with 5MR
      - 5MR can tolerate only two faults where as hybrid scheme can tolerate three faults that occur sequentially
      - cost of the extra fault-tolerance: switch

Hybrid redundancy

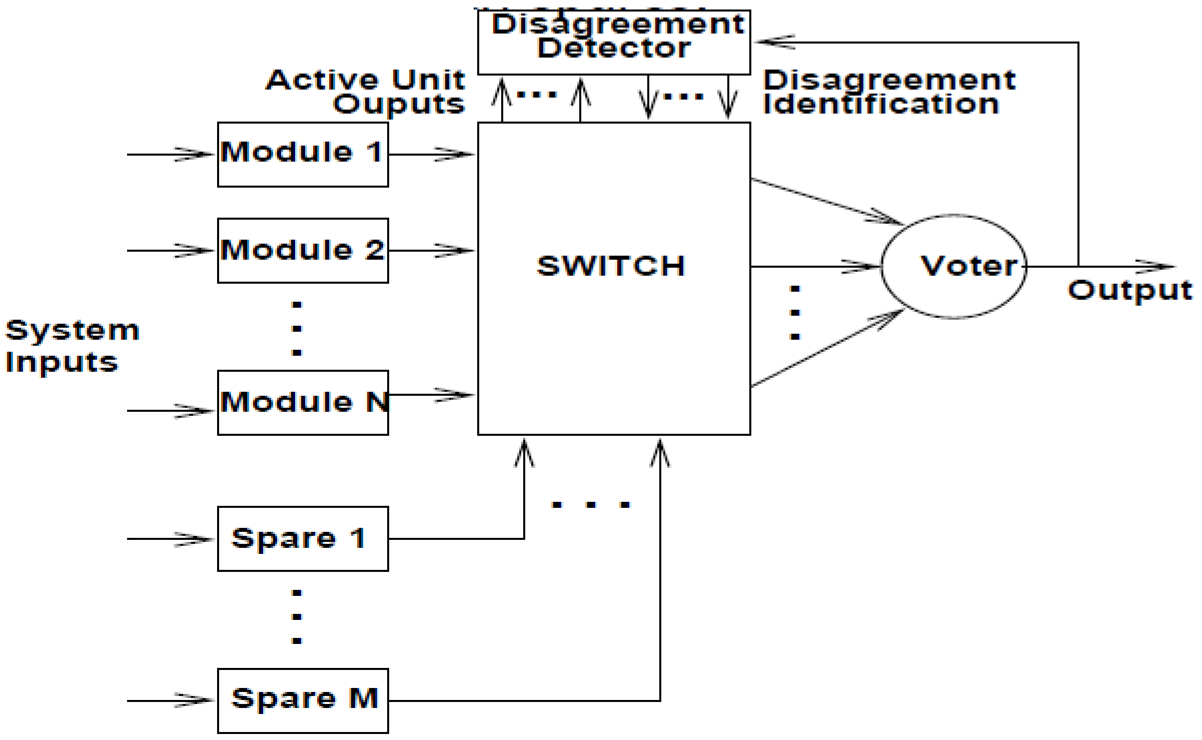


NMR with spares

- The idea here is to provide a basic core of N modules arranged in a form of voting configuration and spares are provided to replace failed units in the NMR core.
- The benefit of NMR with spares is that a voting configuration can be restored after a fault has occurred.

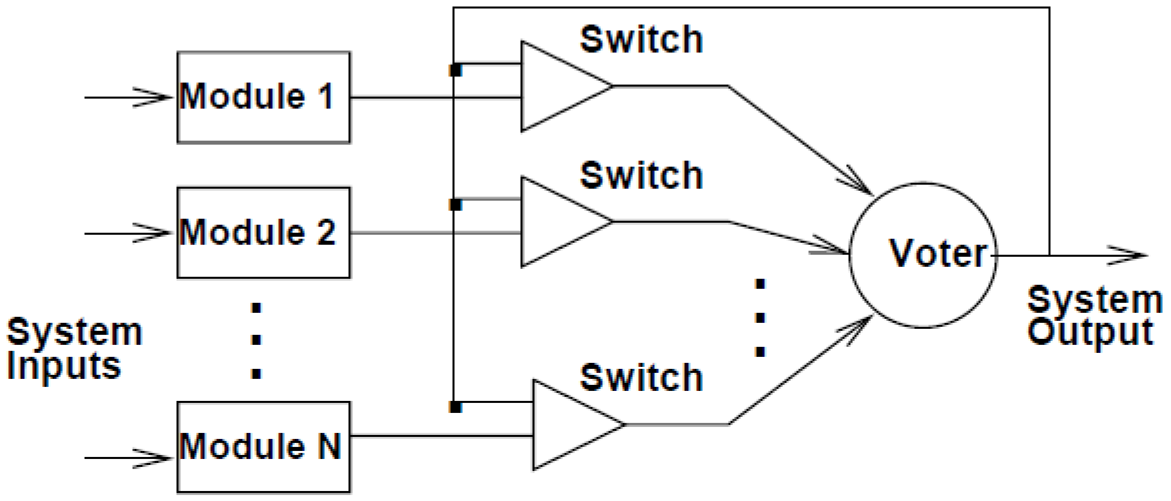
NMR with Spares (Cont'd)

- The voted output is used to identify faulty modules, which are then replaced with spares.



Self-Purging Redundancy

- This is similar to NMR with spares except that all the modules are active, whereas some modules are not active (i.e., the spares) in the NMR with spares.



Sift-Out Modular Redundancy

- It uses N identical modules that are configured into a system using special circuits called comparators, detectors, and collectors.

- The function of the comparator is used to compare each module's output with remaining modules' outputs.

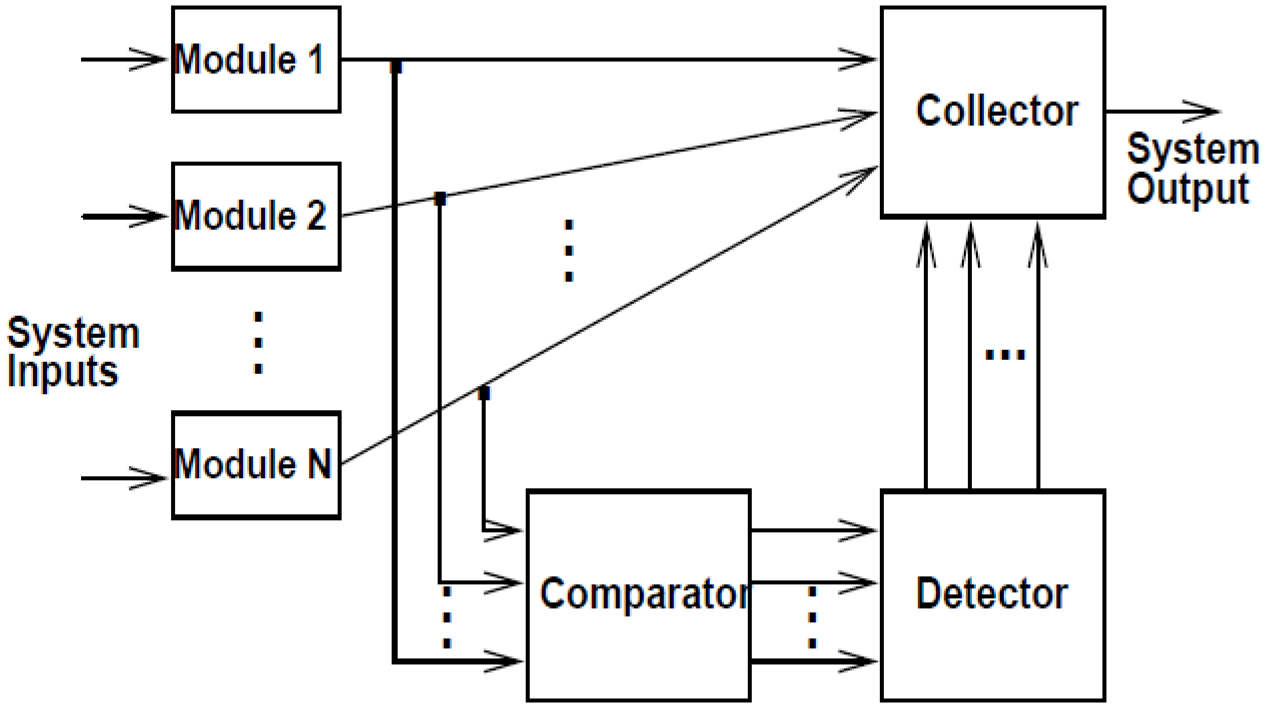
- The function of the detector is to determine which disagreements are reported by the comparator and to disable a unit that disagrees with a majority of the remaining modules.

Sift-Out Modular Redundancy (Cont'd)

- The detector produces one signal value for each module. This value is 1, if the module disagrees with the majority of the remaining modules, 0 otherwise.
- The function of the collector is to produce system's output, given the outputs of the individual modules and the signals from the detector that indicate which modules are faulty.

Sift-Out Modular Redundancy (Cont'd)

- All modules are compared to detect faulty modules.



Hardware Redundancy - Summary

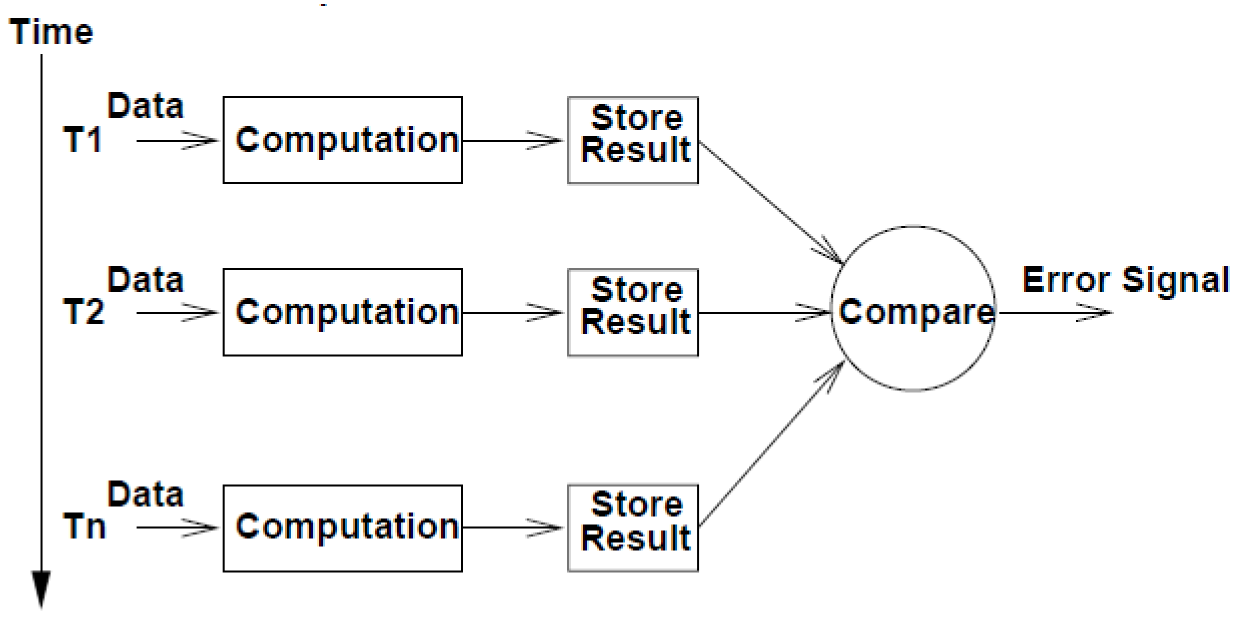
- *Static techniques* rely strictly on fault masking.
- *Dynamic techniques* do not use fault masking but instead employ detection, location, and recovery techniques (reconfiguration).
- *Hybrid techniques* employ both fault masking and reconfiguration.

- In terms of hardware cost, dynamic technique is the least expensive, static technique in the middle, and the hybrid technique is the most expensive.

TIME REDUNDANCY

Time Redundancy - Transient Fault Detection

- In time redundancy, computations are repeated at different points in time and then compared. No extra hardware is required.

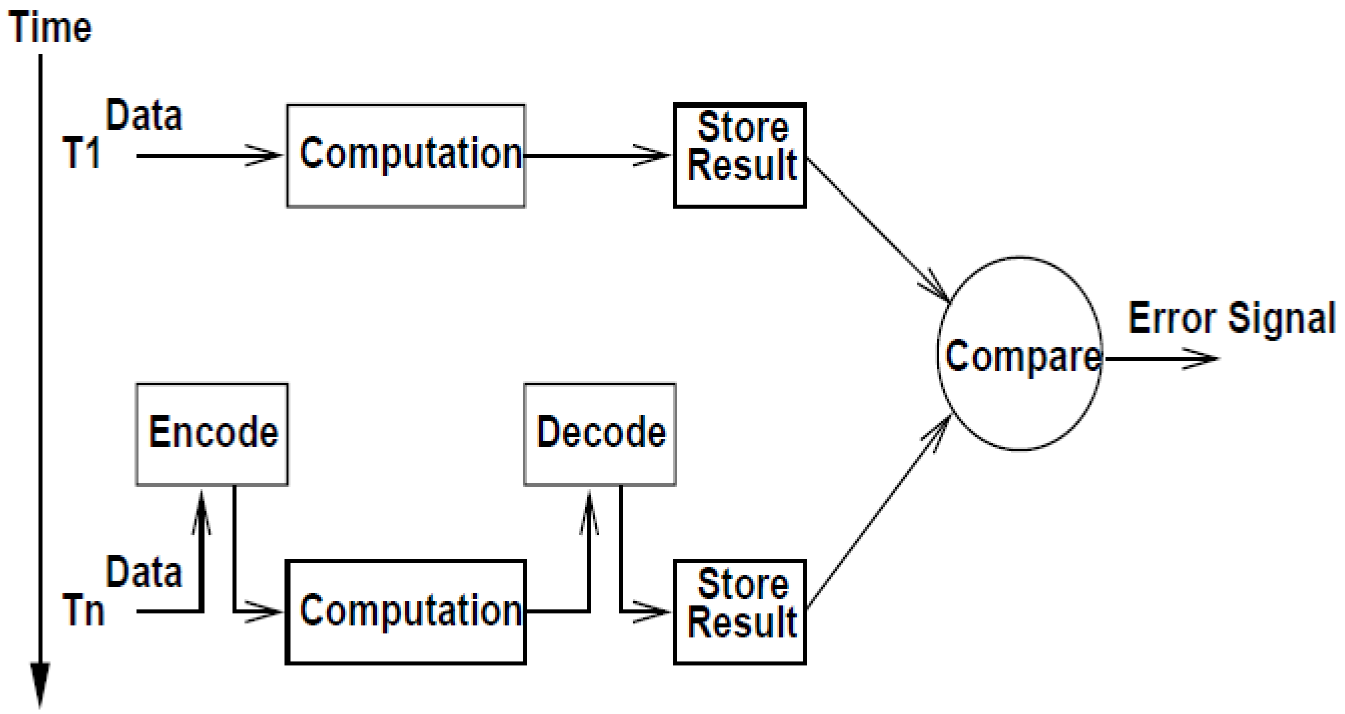


Time Redundancy - Permanent Fault Detection

- During first computation, the operands are used as presented.
- During second computation, the operands are encoded in some fashion.
- The selection of encoding function is made so as to allow faults in the hardware to be detected.

- **Used approaches, e.g., in ALUs:**
  - Recomputing with shifted operands
  - Recomputing with swapped operands
  - ...

Time Redundancy - Permanent Fault Detection (Cont’d)



SOFTWARE REDUNDANCY

Software Redundancy – to Detect Hardware Faults

- **Consistency checks** use a priori knowledge about the characteristics of the information to verify the correctness of that information.
  Example: Range checks, overflow and underflow checks.

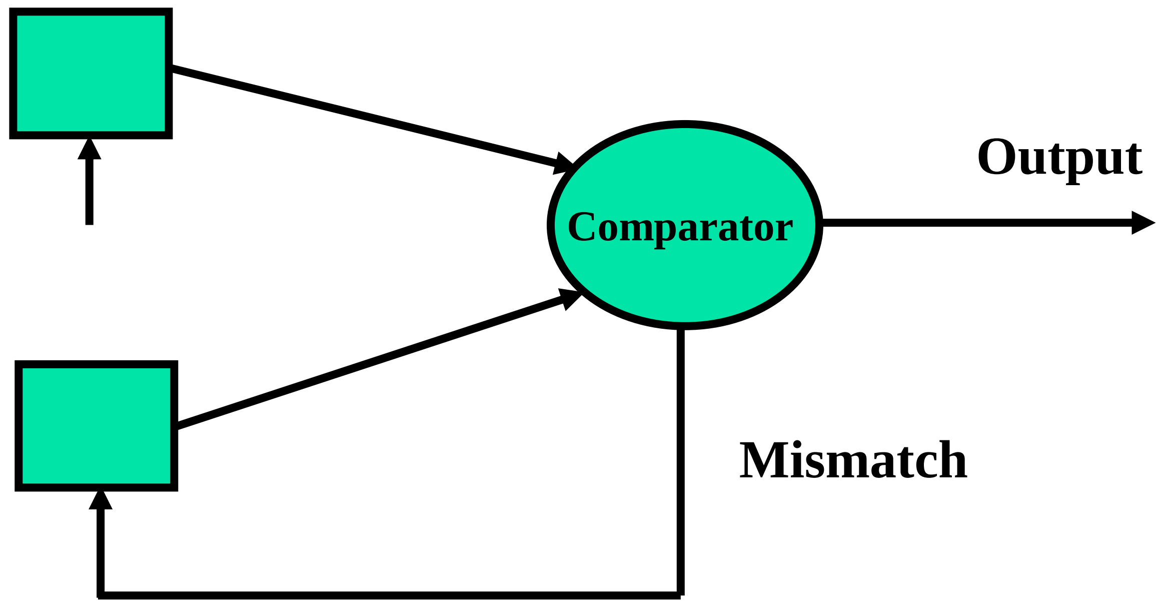
- **Capability checks** are performed to verify that a system possesses the expected capabilities.
  Examples: Memory test - a processor can simply write specific patterns to certain memory locations and read those locations to verify that the data was stored and retrieved properly.

Software Redundancy - to Detect Hardware Faults (Cont’d)

- **ALU tests:** Periodically, a processor can execute specific instructions on specific data and compare the results to known results stored in ROM.

- **Testing of communication** among processors, in a multiprocessor, is achieved by periodically sending specific messages from one processor to another or writing into a specific location of a shared memory.

Fault Tolerance Software Implemented Against Hardware Faults. An example.



- Disagreement triggers interrupts to both processors.
- Both run self diagnostic programs
- The processor that find itself failure free within a specified time continues operation
- The other is tagged for repair

Software Redundancy - to Detect Hardware Faults. One more example.

- All modern day microprocessors use instruction retry
- Any transient fault that causes an exception such as parity violation is retried
- Very cost effective and is now a standard technique

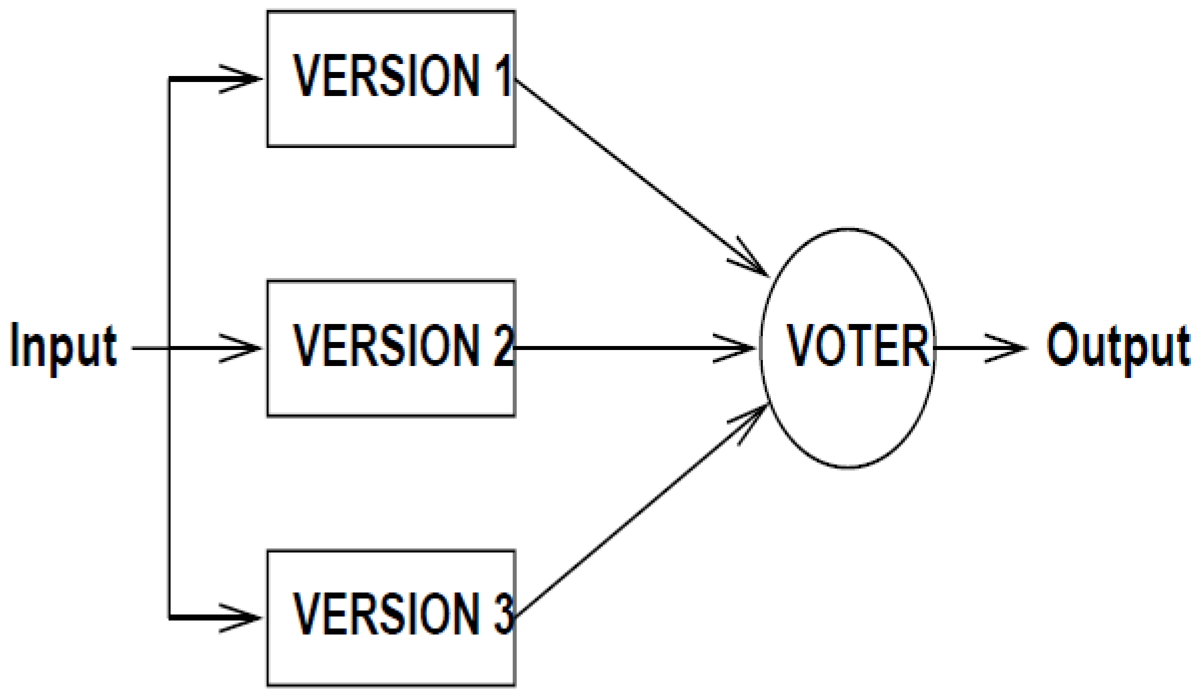
Software Redundancy – to Detect Software Faults

- There are two popular approaches: **N-Version Programming** (NVP) and **Recovery Blocks (RB)**.
- NVP masks faults.
- RB is a backward error recovery scheme.
- In NVP, multiple versions of the same task is executed concurrently, whereas in RB scheme, the versions of a task are executed serially.
- NVP relies on *voting.*
- RB relies on *acceptance test.*

N-Version Programming (NVP)

- NVP is based on the principle of **design diversity**, that is coding a software module by different teams of programmers, to have multiple versions.
- The diversity can also be introduced by employing different algorithms for obtaining the same solution or by choosing different programming languages.
- NVP can tolerate both hardware and software faults.
- Correlated faults are not tolerated by the NVP.
- In NVP, deciding the number of versions required to ensure acceptable levels of software reliability is an important design consideration.

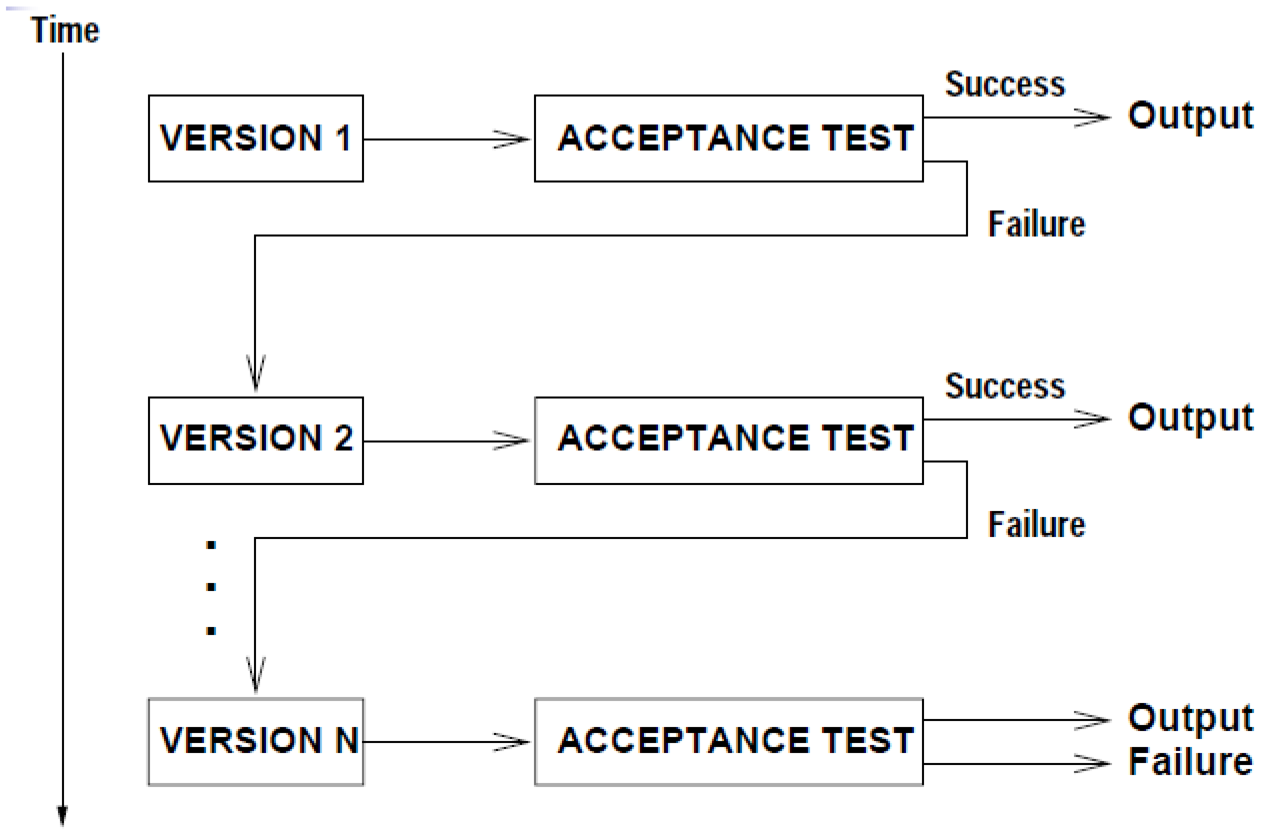
N-Version Programming (Cont’d)



Recovery Blocks (RB)

- RB uses multiple alternates (backups) to perform the same function; one module (task) is primary and the others are secondary.
- The primary task executes first. When the primary task completes execution, its outcome is checked by an **acceptance test.**
- If the output is not acceptable, another task is executed after undoing the effects of the previous one (i.e., rolling back to the state at which primary was invoked) until either an acceptable output is obtained or the alternatives are exhausted.

Recovery Blocks (Cont'd)



Recovery Blocks (Cont'd)

- The acceptance tests are usually sanity checks; these consist of making sure that the output is within a certain acceptable range or that the output does not change at more than the allowed maximum rate.

- Selecting the range for the acceptance test is crucial. If the allowed ranges are too small, the acceptance tests may label correct outputs as bad (false positives). If they are too large, the probability that incorrect outputs will be accepted (false negatives) will be increase.

- RB can tolerate software faults because the alternatives are usually implemented with different approaches; RB is also known as **Primary-Backup approach.**

Single Version Fault Tolerance: Software Rejuvenation

- Example: Rebooting a PC
- As a process executes
  - it acquires memory and file-locks without properly releasing them
  - memory space tends to become increasingly fragmented
  - the process can become faulty and stop executing
- To head this off, proactively halt the process, clean up its internal state, and then restart it
- Rejuvenation can be time-based or prediction-based
- Time-Based Rejuvenation - periodically
- Rejuvenation period - balance benefits against cost

INFORMATION REDUNDANCY

Information Redundancy

- Guarantee data consistency by exploiting additional information to achieve a redundant encoding.

- Redundant codes permit to detect or correct corrupted bits because of one or more faults:
  - Error Detection Codes (EDC)
  - Error Correction Codes (ECC)

Functional Classes of Codes

- Single error correcting codes
  - any one bit can be detected and corrected
- Burst error correcting codes
  - any set of consecutive b bits can be corrected
- Independent error correcting codes
  - up to t errors can be detected and corrected
- Multiple character correcting codes
  - n-characters, t of them are wrong, can be recovered
- Coding complexity goes up with number of errors
  - Sometimes partial correction is sufficient

Redundant Codes

Let:

b: be the code's alphabet size (the base in case of numerical codes);

n: the (constant) block size;

N: the number of elements to be coded;

m: the minimum value of n which allows to encode all the elements in the source code, i.e. minimum **m** such that $b^m \geq N$

A code is said

Not redudant if	$n = m$
Redundant if	$n > m$
Ambiguous if	$n < m$

Binary Codes: Hamming distance

The **Hamming distance d(x,y)** between two words (x,y) of a code (C) is the number of different bits in the same position between x and y

d(10010 , 01001) = 4

d(11010 , 11001) = 2

The **minimum distance** of a code is

$d_{min} = \min(d(x,y))$ for all $x \neq y$ in C

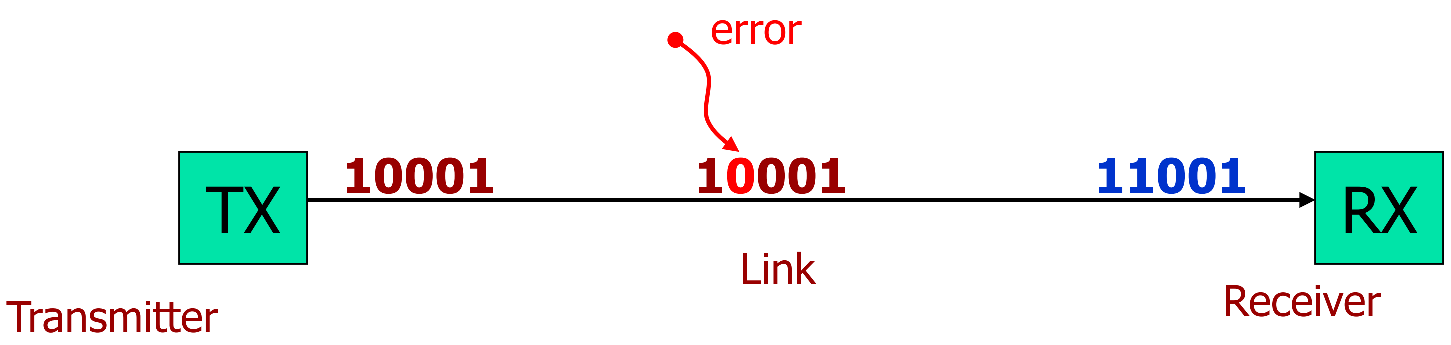
Ambiguity and redundancy

Not redundant codes	$h = 1$ (and $n = m$)
Redundant codes	$h >= 1$ (and $n > m$)
Ambiguous codes	$h = 0$

Hamming Distance: Examples

Words of C	First Code	Second Code	Third Code	Fourth code	Fifth code
alfa	000	0000	00	0000	110000
beta	001	0001	01	0011	100011
gamma	010	0010	11	0101	001101
delta	011	0011	10	0110	010110
mu	100	0100	00	1001	011011
	h = 1	h = 1	h = 0	h = 2	h = 3
	Not Red.	Red.	Amb.	Red. *(EDC)*	Red. *(ECC)*

Error Detecting Codes (EDC)



To"detect" transmission erroes the transmittin system
introduces redundancy in the transmitted information.

In an *error detecting code* the occurrence of an error
on a word of the code generates a *word not belonging to the code*

The **error weight** is the number (and distribution) of corrupted bits tolerated by the code.

In binary systems there are only two error possibilities
Trasmit 0 Receive 1
Trasmit 1 Receive 0

Error Detection Codes

The **Hamming distance d(x,y)** between two words (x,y) of a code (C)

is the number of different positions (bits) between x and y

d(10010 , 01001) = 4

d(11010 , 11001) = 2

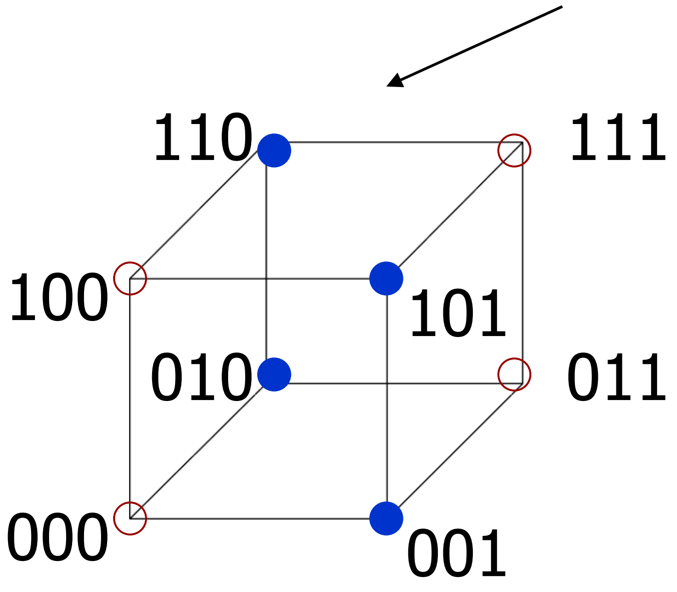
The **minimum distance** of a code is

$d_{min} = \min(d(x,y))$ for all $x \neq y$ in C

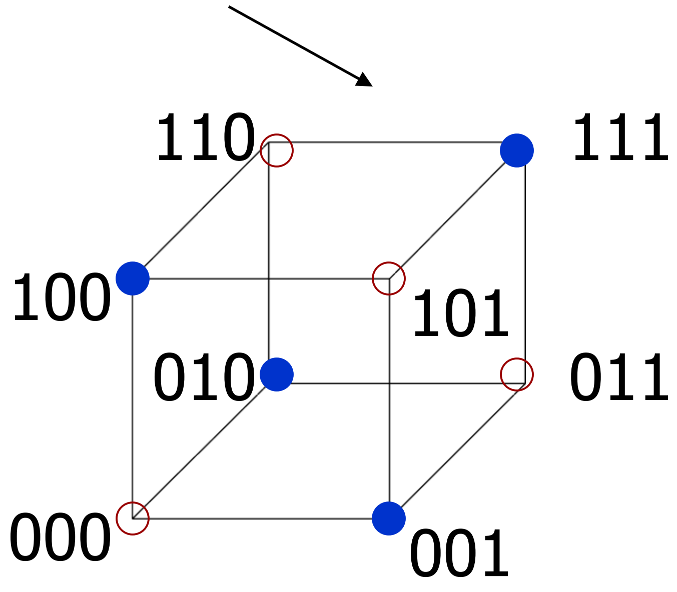
A code having **minimum distance d** is
able to **detect errors with weight $\leq$ d-1**

Error Detecting Codes

		Code 1	Code 2
A	=>	000	000
B	=>	100	011
C	=>	011	101
D	=>	111	110



$d_{min}=1$



$d_{min}=2$

○ Legal code words
● Illegal code words

Parity Code (minimum distance 2)

A code having d_{min} =2 can be obtained by using the following expressions:

$d_1 + d_2 + d_3 + \ldots.. + d_n + p = 0$ parity (even number of "1") or

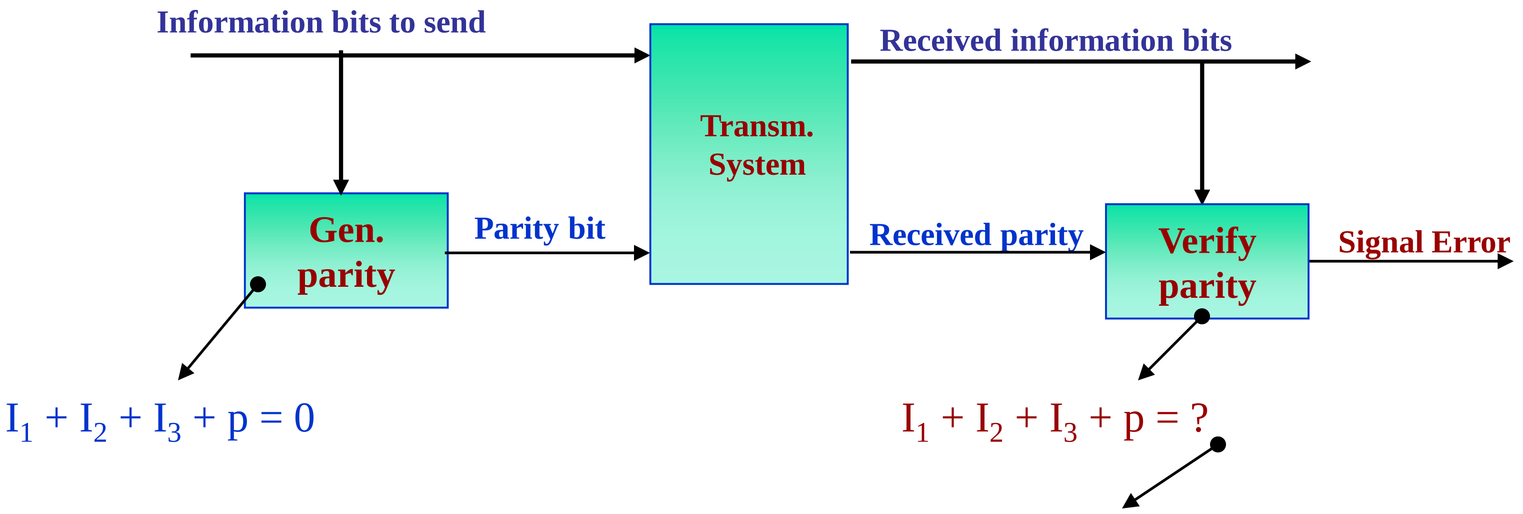
$d_1 + d_2 + d_3 + \ldots.. + d_n + p = 1$ odd number of "1"

Being n the number of bits of the original block code and + is modulo 2 sum operator and p is the parity bit to add to the original word to obtain an EDC code

Information	**Parity (even)**	**Parity (odd)**
000	000 0	000 1
001	001 1	001 0
010	010 1	010 0
011	011 0	011 1
100	100 1	100 0
101	101 0	101 1
110	110 0	110 1
111	111 1	111 0

A code with minimum distance equal to 2
can detect errors having weight 1 (single error)

Parity Code



$I_1 + I_2 + I_3 + p = 0$

$I_1 + I_2 + I_3 + p = ?$

- If equal to 0 there has been ***no*** single error
- If equal to 1 there has been a single error

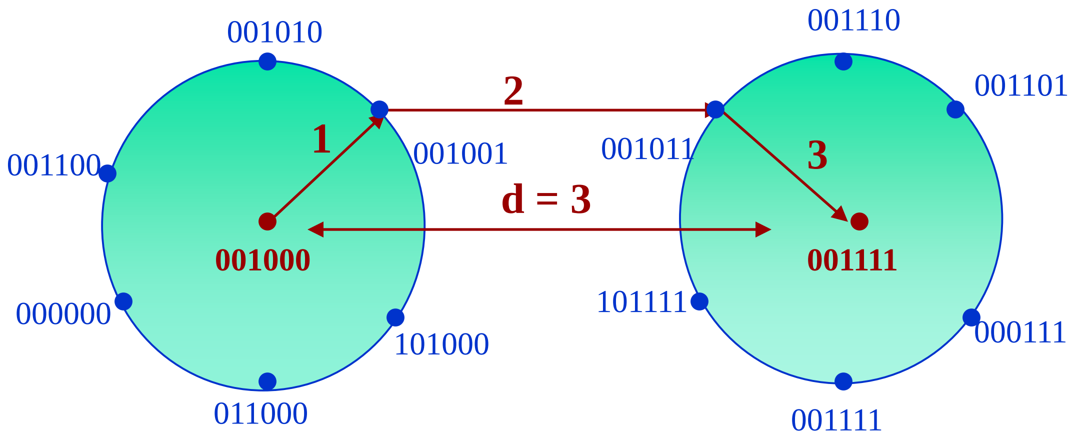
Ex. I trasmit 101
Parity generator computes the parity bit $1 + 0 + 1 + p = 0$, namely $p = 0$ and 1010 is transmitted

If 1110 is received, the parity check detects an error $1 + 1 + 1 + 0 = 1 \neq 0$
If 1111 is received: $1+1+1+1 = 0$ all right??, no single mistakes!!
(double/even weight errors are unnoticeable)

Error Correcting Codes

A code having minimum distance **d** can **correct errors** with weight $\leq \lfloor (d-1)/2 \rfloor$

When a code has minimum distance 3 can correct errors having weight = 1



Codici Hamming(1)

- Metodo per la costruzione di codici a distanza minima 3

- per ogni *i* e' possibile costruire un codice a 2^i -1 bit con *i* bit di parità (check bit) e 2^i -1-i bit di informazione.

- I bit in posizione corrispondente ad una **potenza di 2** (1,2,4,8,...) sono **bit di parità** i rimanenti sono bits di informazione

- Ogni bit di parità controlla la correttezza dei bit di informazione la cui posizione, espressa in binario, ha un 1 nella potenza di 2 corrispondente al bit di parità

$$(3)_{10} = (0 \quad 1 \quad 1)_2$$
$$(5)_{10} = (1 \quad 0 \quad 1)_2$$
$$(6)_{10} = (1 \quad 1 \quad 0)_2$$
$$(7)_{10} = (1 \quad 1 \quad 1)_2$$
$$2^2 \; 2^1 \; 2^0$$

$$I_7 + I_6 + I_5 + p_4 = 0$$

$$I_7 + I_6 + I_3 + p_2 = 0$$

$$I_7 + I_5 + I_3 + p_1 = 0$$

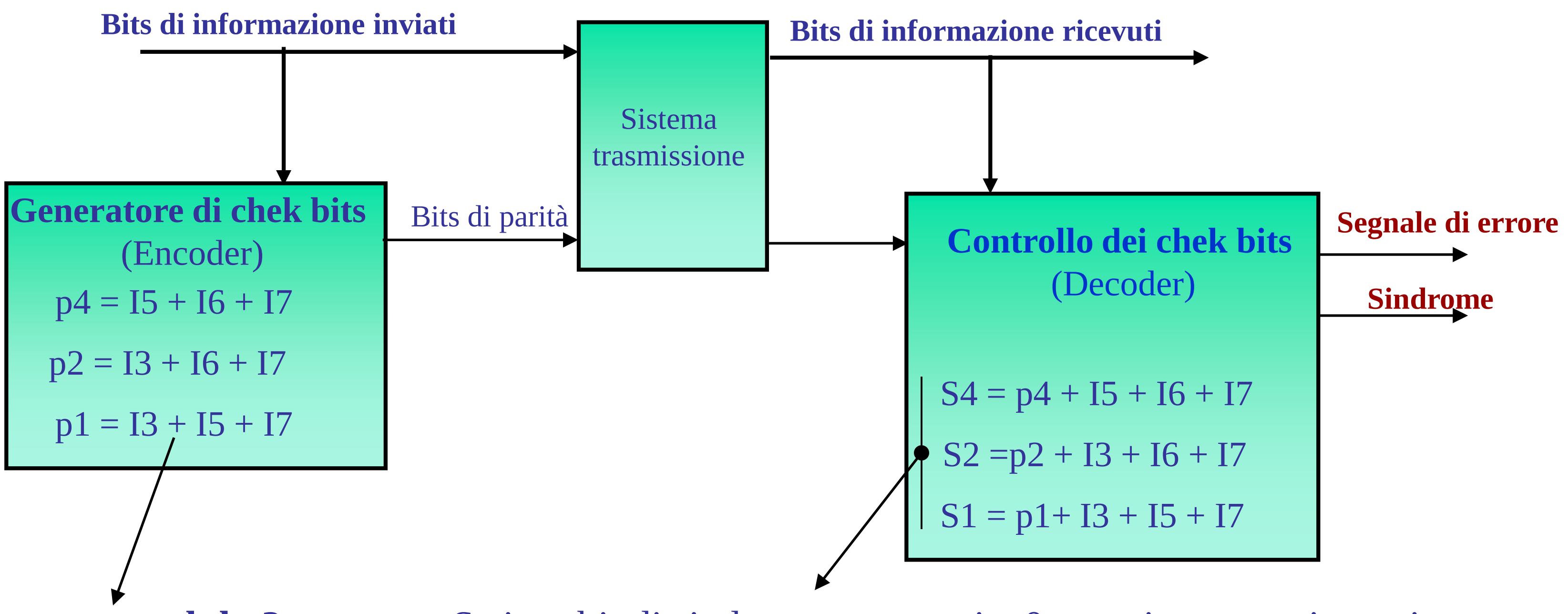
Codici Hamming(2)

	p_1	p_2	I_3	p_4	I_5	I_6	I_7	
	1	2	3	4	5	6	7	$\leftarrow$ posizione
Gruppi				X	X	X	X	$p_4 + I_5 + I_6 + I_7 = 0$
		X	X			X	X	$p_2 + I_3 + I_6 + I_7 = 0$
	X		X		X		X	$p_1 + I_3 + I_5 + I_7 = 0$

p_i: bit di parità

I_i: bit di informazione

Circuito di EDAC (Error Detection And Correction)


Bits di informazione inviati
Sistema trasmissione
Bits di informazione ricevuti
Generatore di chek bits (Encoder)
Bits di parità
Controllo dei chek bits (Decoder)
Segnale di errore
Sindrome

$p4 = I5 + I6 + I7$

$p2 = I3 + I6 + I7$

$p1 = I3 + I5 + I7$

$S4 = p4 + I5 + I6 + I7$

$S2 = p2 + I3 + I6 + I7$

$S1 = p1 + I3 + I5 + I7$

Somma modulo 2

Se i tre bit di sindrome sono pari a 0 non ci sono stati errori altrimenti il loro valore da’ la posizione del bit errato

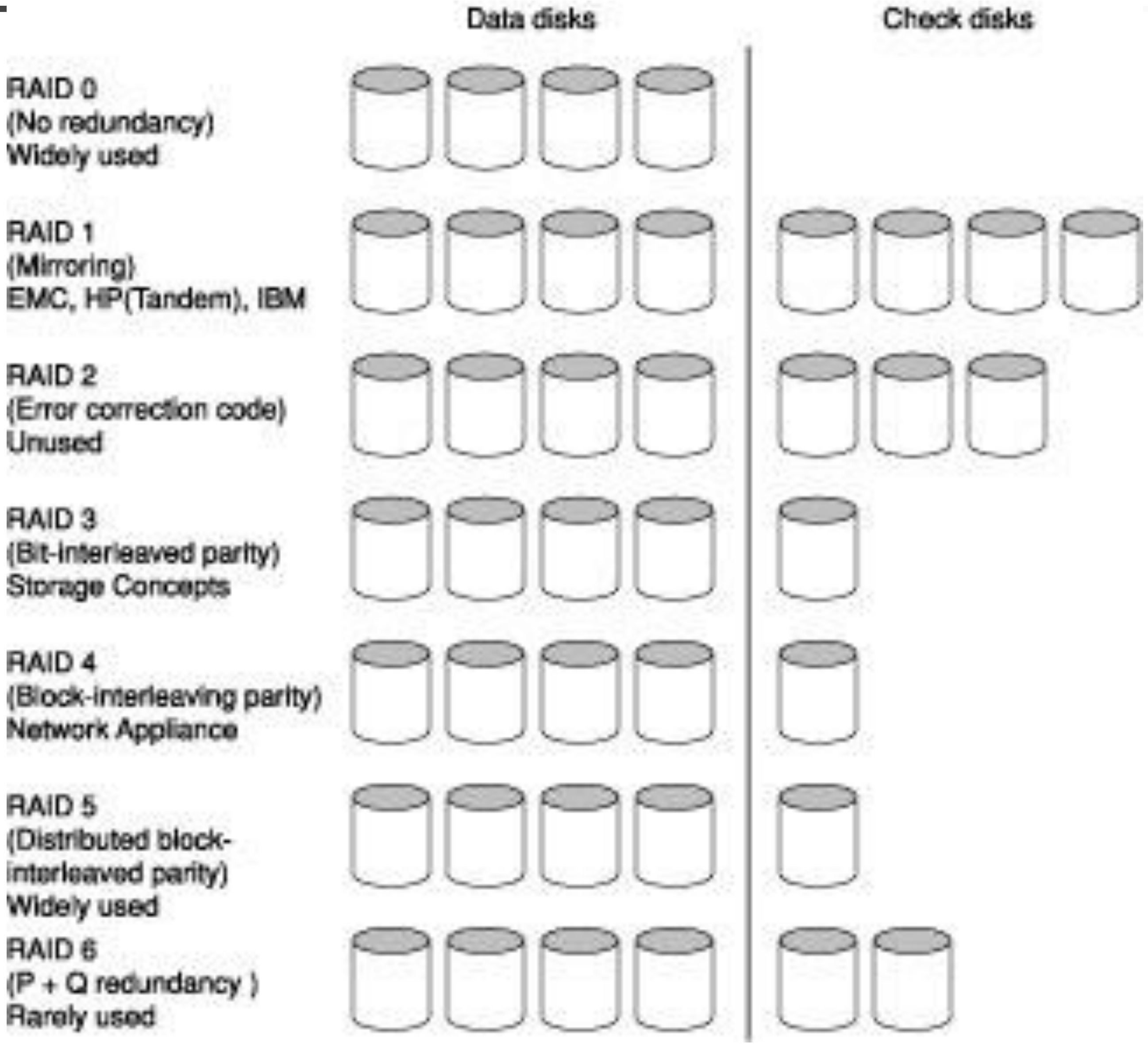
Redundant Array of Inexpensive Disks
RAID

RAID Architecture

- RAID: Redundant Array of Inexpensive Disks
  - Combine multiple small, inexpensive disk drives into a group to yield performance exceeding that of one large, more expensive drive
  - Appear to the computer as a single virtual drive
  - Support fault-tolerance by redundantly storing information in various ways
  - Uses Data Striping to achieve better performance

Basic Issues

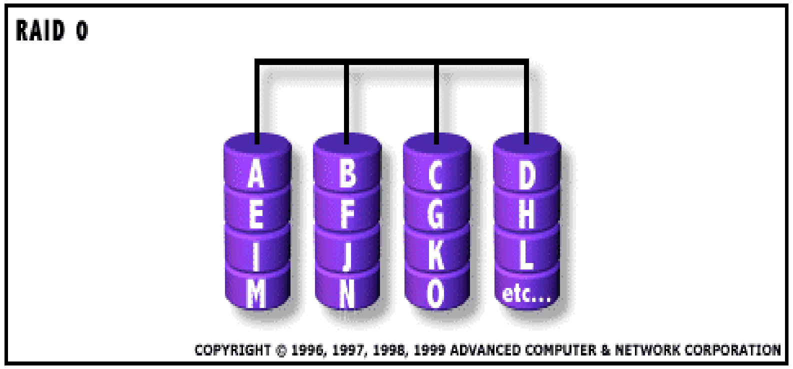
- Two operations performed on a disk
  - Read() : small or large.
  - Write(): small or large.
- Access Concurrency is the number of simultaneous requests the can be serviced by the disk system
- Throughput is the number of bytes that can be read or written per unit time as seen by one request
- Data Striping: spreading out blocks of each file across multiple disk drives.



Data disks
Check disks
RAID 0
(No redundancy)
Widely used
RAID 1
(Mirroring)
EMC, HP(Tandem), IBM
RAID 2
(Error correction code)
Unused
RAID 3
(Bit-interleaved parity)
Storage Concepts
RAID 4
(Block-interleaving parity)
Network Appliance
RAID 5
(Distributed block-interleaved parity)
Widely used
RAID 6
(P + Q redundancy)
Rarely used

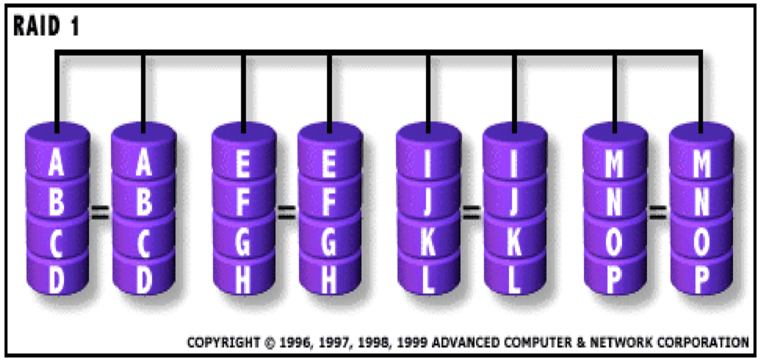
RAID Levels: RAID-0

- No Redundancy
  - No Fault Tolerance, If one drive fails then all data in the array is lost.
- High I/O performance
  - Parallel I/O
- Best Storage efficiency


RAID 0
A
E
I
M
B
F
J
N
C
G
K
O
D
H
L
etc...
COPYRIGHT © 1996, 1997, 1998, 1999 ADVANCED COMPUTER & NETWORK CORPORATION

RAID-1

- Disk Mirroring
  - Poor Storage efficiency.
- Best Read Performance: double of the RAID 0.
- Poor write Performance: two disks to be written.
- Good fault tolerance: as long as one disk of a pair is working then we can perform R/W operations.


RAID 1
A B C D = A B C D
E F G H = E F G H
I J K L = I J K L
M N O P = M N O P
COPYRIGHT © 1996, 1997, 1998, 1999 ADVANCED COMPUTER & NETWORK CORPORATION

RAID-2

- Bit Level Striping.
- Uses Hamming Codes, a form of Error Correction Code (ECC).

- Can Tolerate the Failure of one disk
- # Redundant Disks = O (log (total disks)).

- Better Storage efficiency than mirroring.

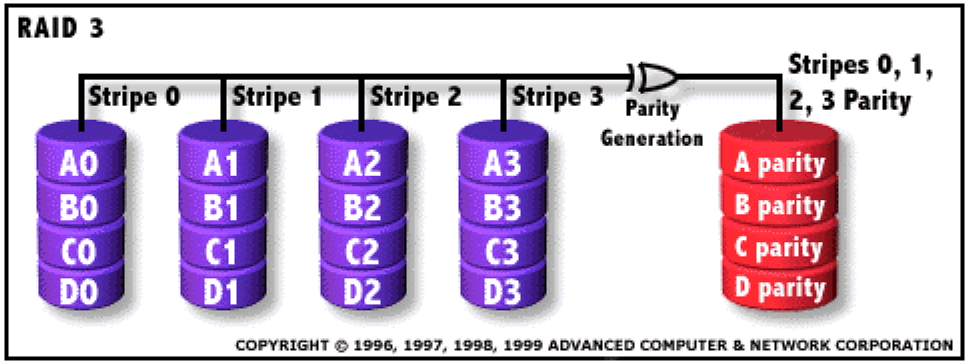
- High throughput but no access concurrency:
  - disks need to be ALWAYS simulatenously accessed
    - Synchronized rotation

- Expensive write.

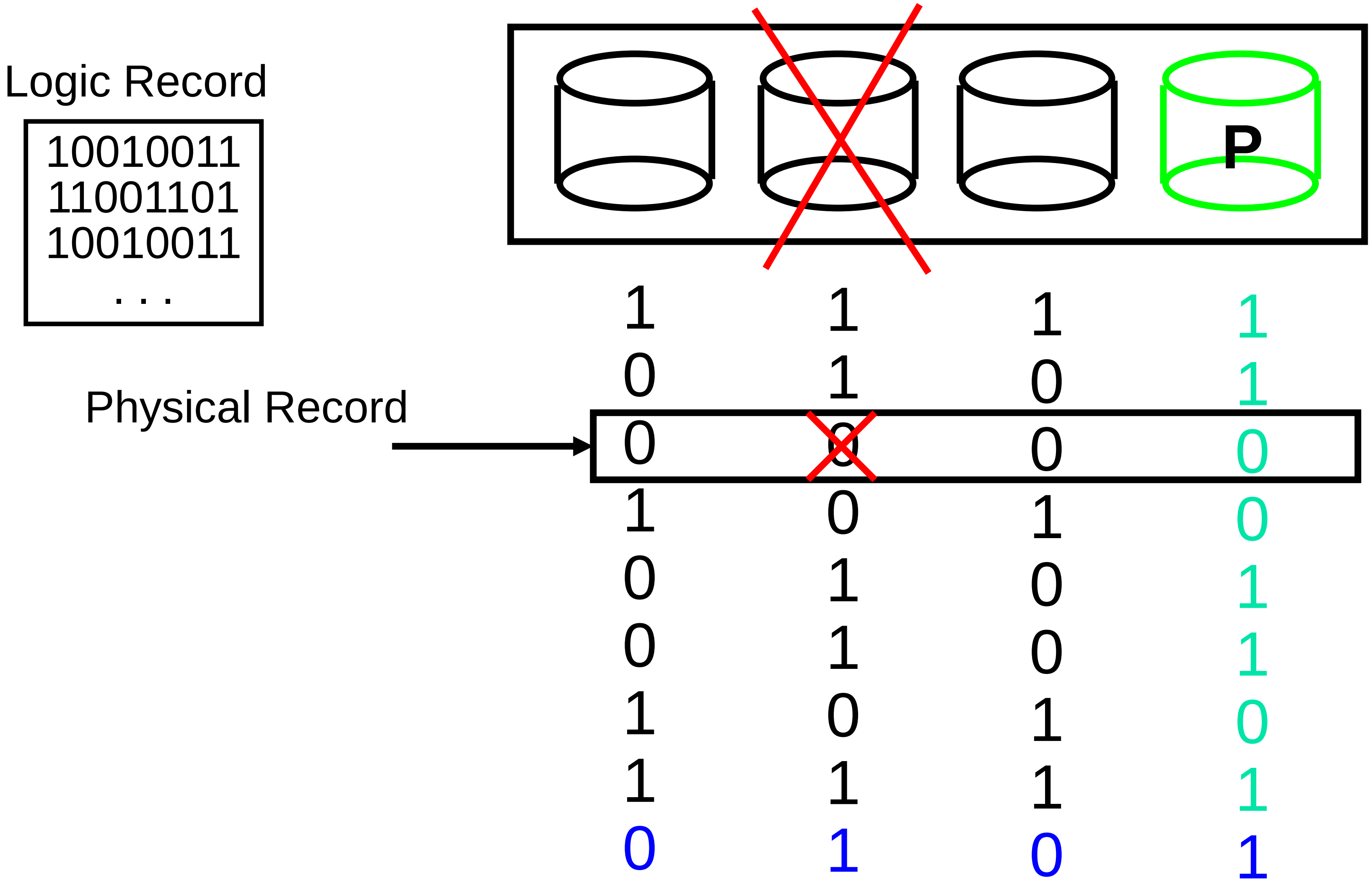
- Example, for 4 disks 3 redundant disks to tolerate one disk failure

RAID-3

- Byte Level Striping with parity.
- No need for ECC since the controller knows which disk is in error. So parity is enough to tolerate one disk failure.
- Best Throughput, but no concurrency.
- Only one Redundant disk is needed.



RAID-3 (example in which there is only a byte for disk)



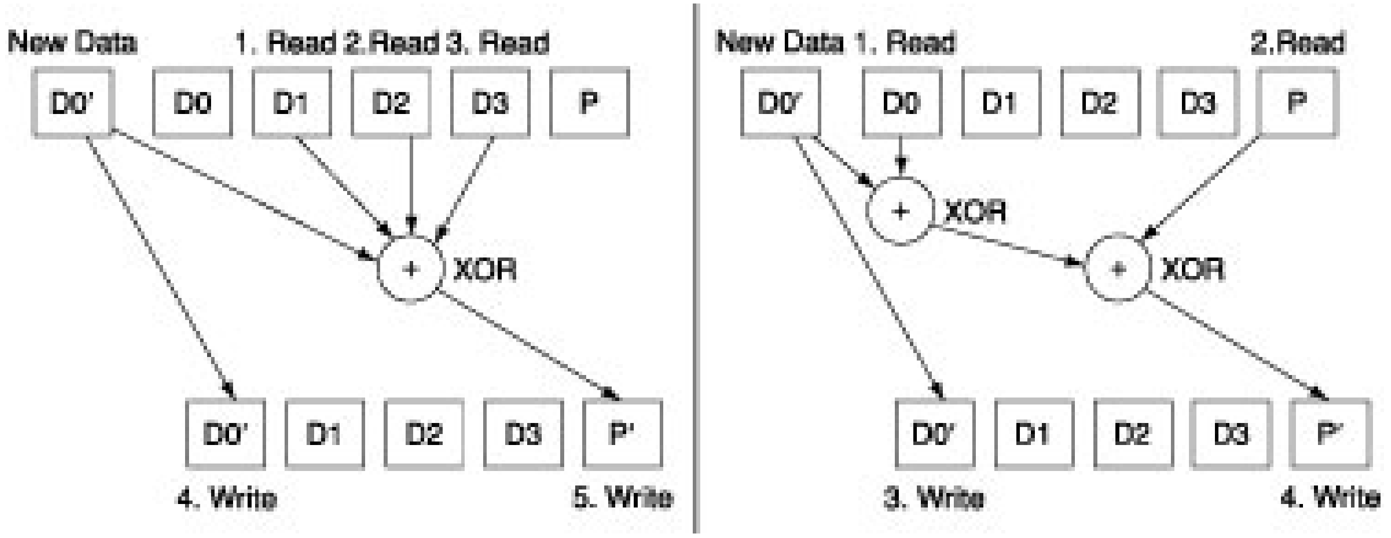
RAID-4

- Block Level Striping.
- Stripe size introduces the tradeoff between access concurrency versus throughput.

- Parity disk is a bottleneck in the case of a small write where we have multiple writes at the same time.
- No problems for small or large reads.

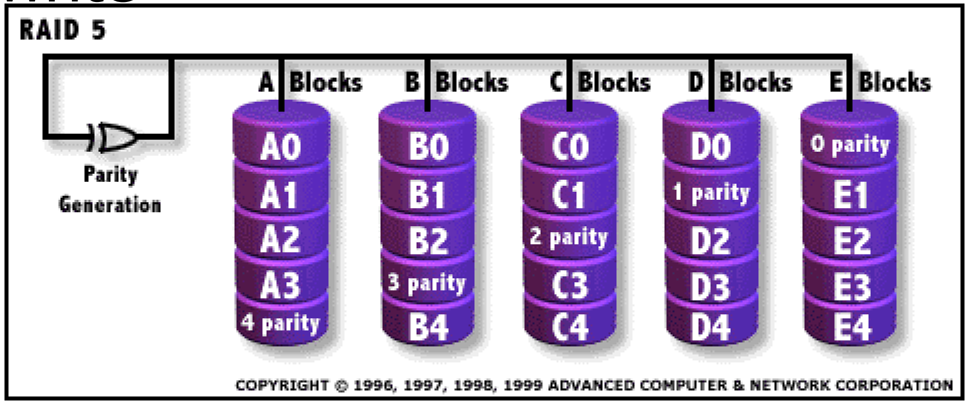
Writes in RAID-3 and RAID-4.

- In general writes are very expensive.
  - *Option 1*: read data on all other disks, compute new parity P' and write it back
    - Ex.: 1 logical write= 3 physical reads + 2 physical writes
  - *Option 2*: check old data D0 with the new one D0', add the difference to P, and write back P'
    - Ex.: 1 logical write= 2 physical reads + 2 physical writes

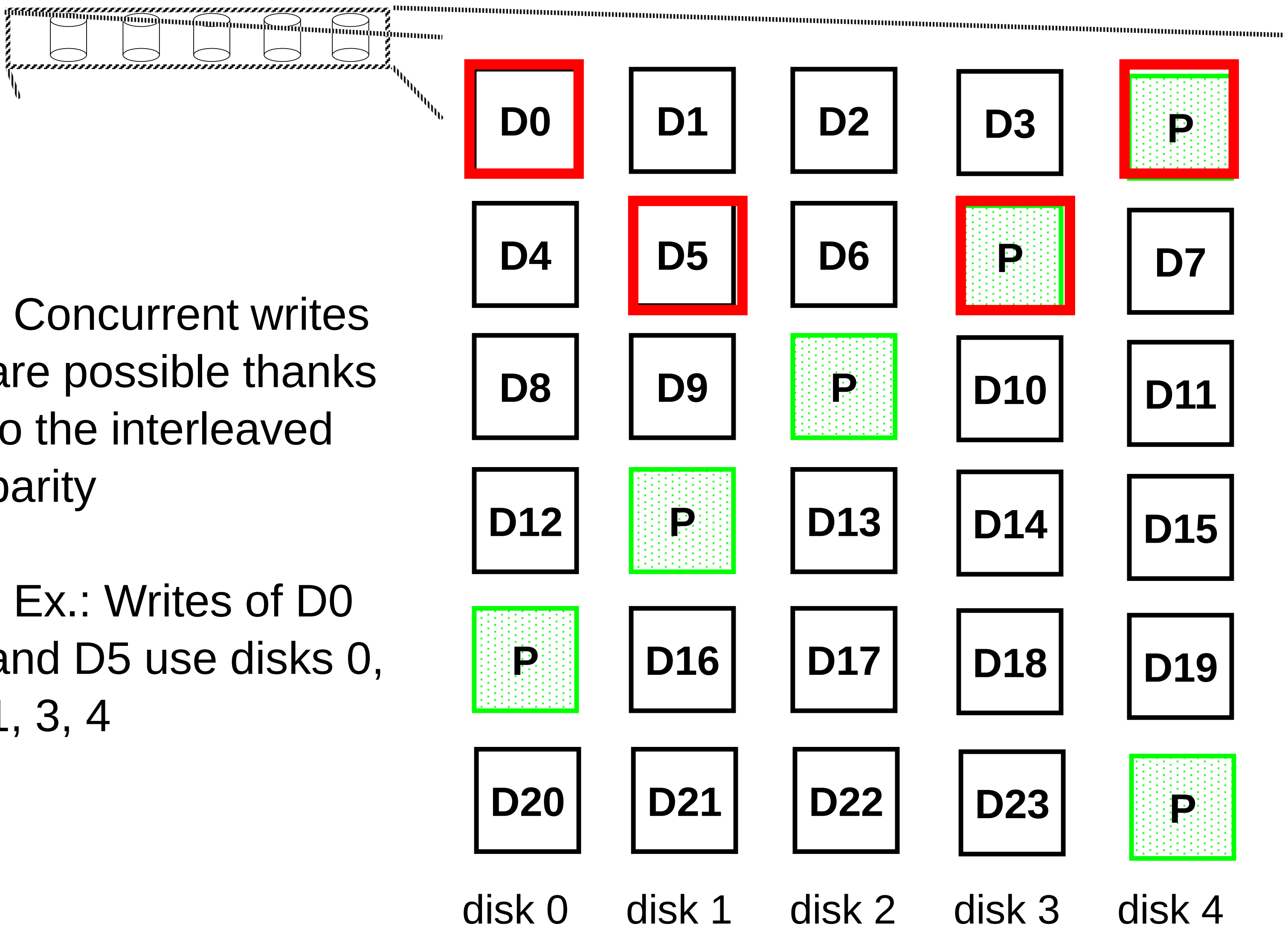


RAID-5

- Block-Level Striping with *Distributed* parity.
- Parity is uniformly distributed across disks.
- Reduces the parity Bottleneck.
- Best small and large read (same as 4).
- Best Large write.
- Still costly for small write



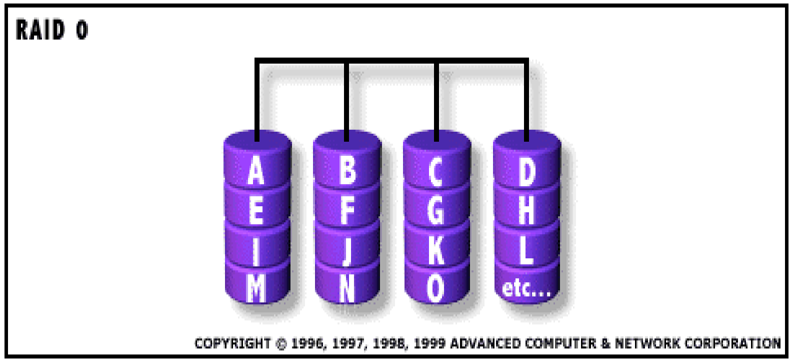
Writes in Raid 5



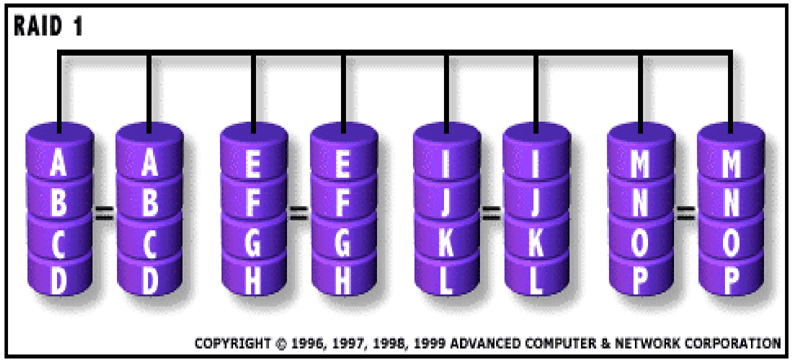
- Concurrent writes are possible thanks to the interleaved parity

- Ex.: Writes of D0 and D5 use disks 0, 1, 3, 4

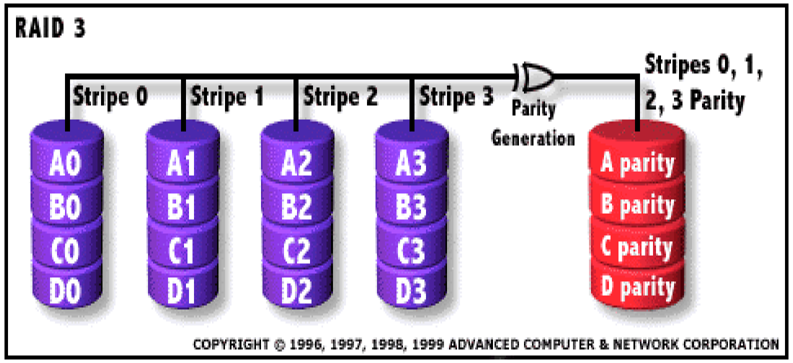
Summary of RAID Levels

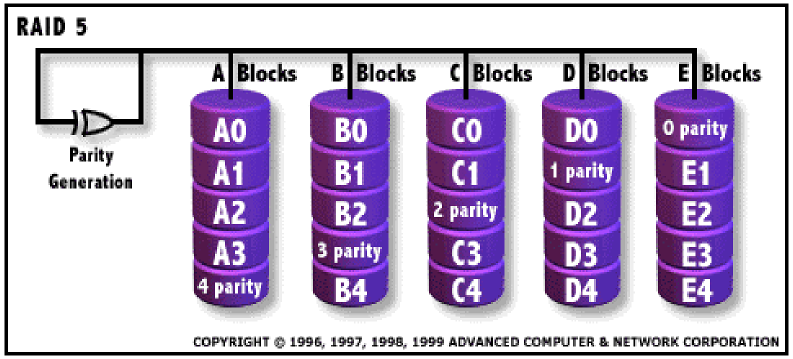


RAID 0
A E I M
B F J N
C G K O
D H L etc...
COPYRIGHT © 1996, 1997, 1998, 1999 ADVANCED COMPUTER & NETWORK CORPORATION



RAID 1
A B C D = A B C D
E F G H = E F G H
I J K L = I J K L
M N O P = M N O P
COPYRIGHT © 1996, 1997, 1998, 1999 ADVANCED COMPUTER & NETWORK CORPORATION



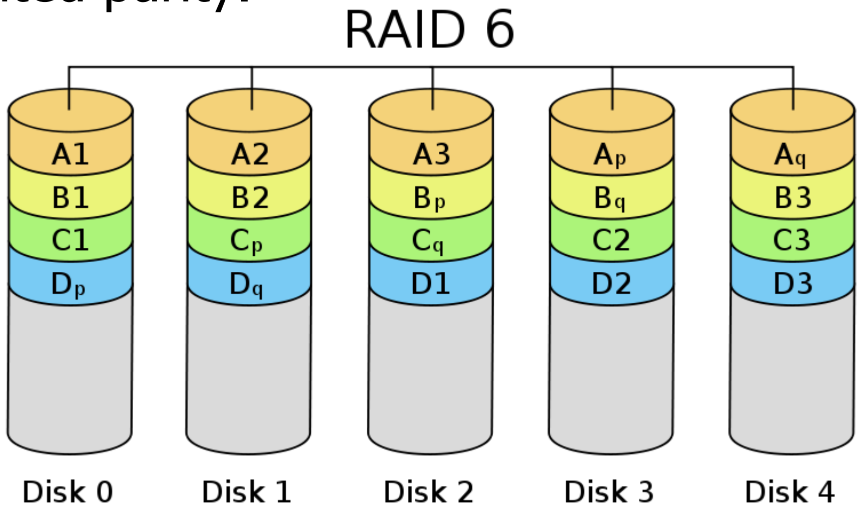


Limits of RAID-5

- RAID-5 is probably the most employed scheme
- The larger the number of disks in a RAID-5, the better performances we may get...
- ...but the larger gets the probability of double disk failure:
  - After a disk crash, the RAID system needs to reconstruct the failed crash:
    - detect, replace and recreate a failed disk
    - this can take hours if the system is busy
  - The probability that one disk out N-1 to crashes within this vulnerability window can be high if N is large:
    - especially considering that disks in an array have typically the same age => correlated faults
    - rebuilding a disk may cause reading a HUGE number of data
    - may become even higher than the probability of a single disk's failure

RAID-6

- Block-level striping with *dual* distributed parity.



- *Two* sets of parity are calculated.
- Better fault tolerance
  Data reconstruction is faster than
  RAID5, so the probability of a second
  Fault during data reconstruction is less.
- Writes are slightly worse than 5 due to the added overhead of more parity calculations.
- May get better read performance than 5 because data and parity are spread into more disks.

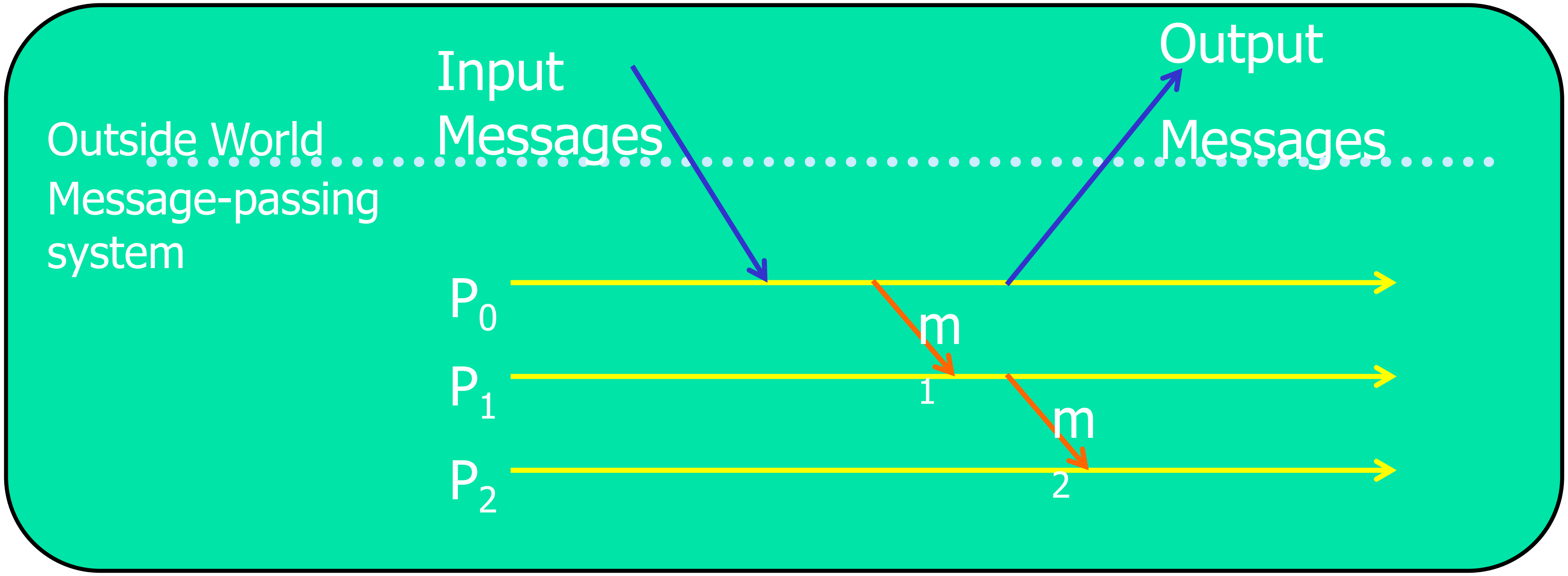
Error Propagation in Distributed Systems

and

Rollback Error Recovery Techniques

System Model

- System consists of a fixed number (N) of processes which communicate only through messages.
- Processes cooperate to execute a distributed application program and interact with outside world by receiving and sending input and output messages, respectively.



Rollback Recovery in a Distributed System

- Rollback recovery treats a distributed system as a collection of processes that communicate through a network
- Fault tolerance is achieved by periodically using stable storage to save the processes' states during the failure-free execution.
- Upon a failure, a failed process restarts from one of its saved states, thereby reducing the amount of lost computation.
- Each of the saved states is called a <u>checkpoint</u>

Checkpoint based Recovery: Overview

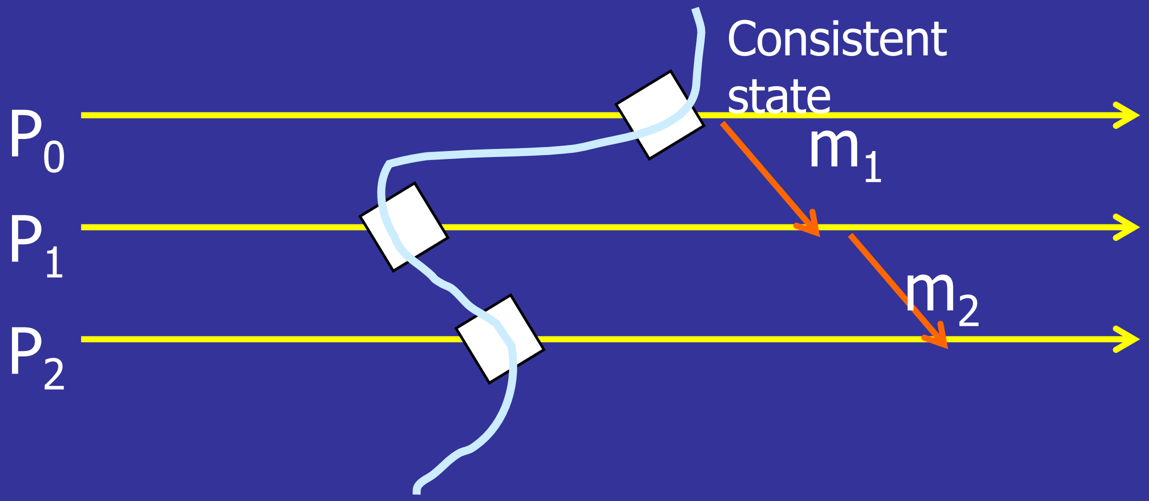
- **Uncoordinated checkpointing**: Each process takes its checkpoints independently
- **Coordinated checkpointing**: Processes coordinate their checkpoints in order to save a system-wide consistent state. This consistent set of checkpoints can be used to bound the rollback
- **Communication-induced checkpointing**: It forces each process to take checkpoints based on information piggybacked on the application messages it receives from other processes.

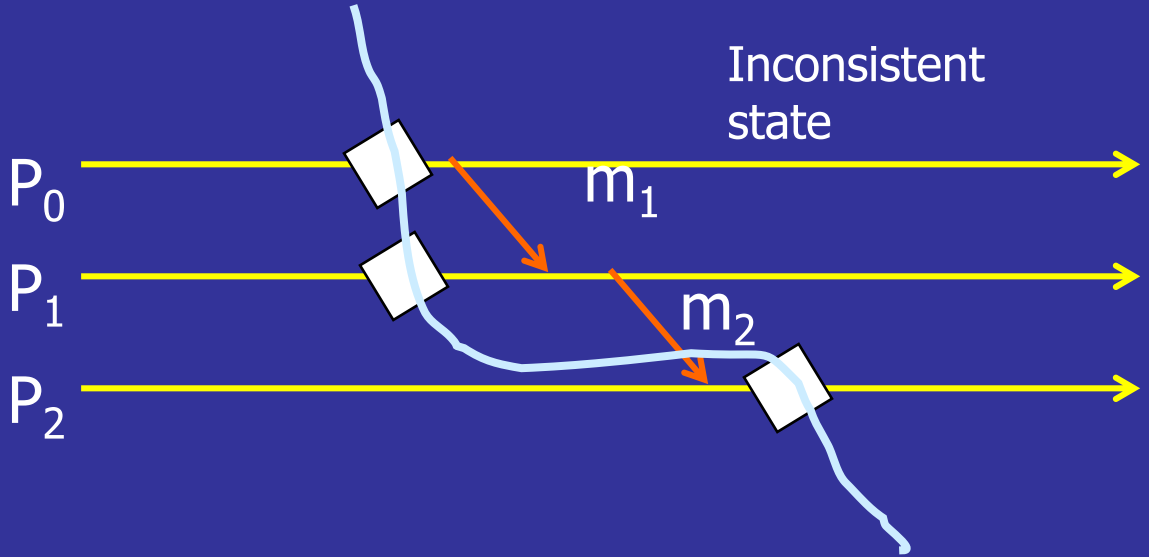
Consistent System State

- A consistent system state is one in which if a process's state reflects a message receipt, then the state of the corresponding sender reflects sending that message.

- A fundamental goal of any rollback-recovery protocol is to bring the system into a consistent state when inconsistencies occur because of a fault.

Example



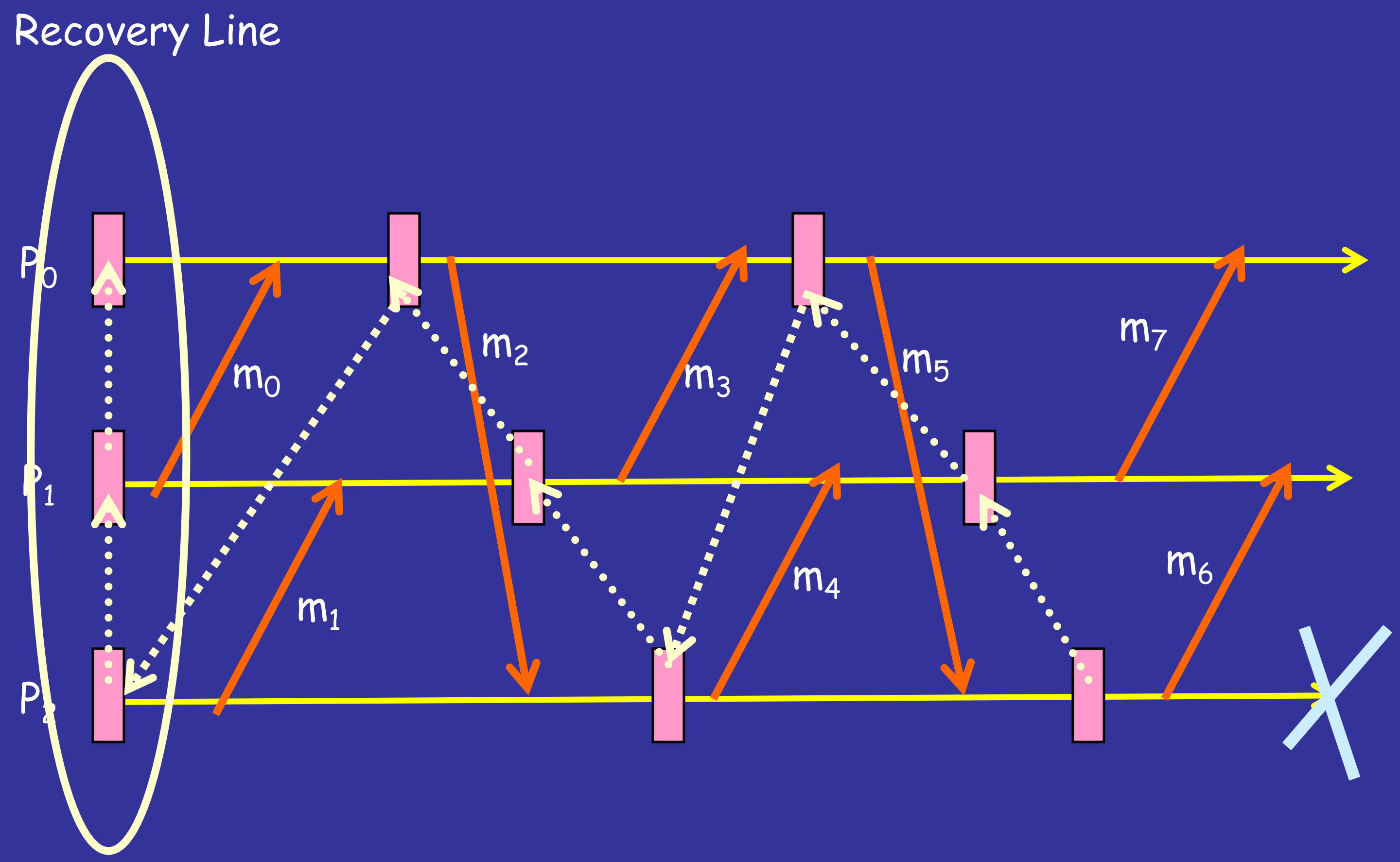


"m_2" becomes the orphan message

Checkpointing protocols

- Each process “periodically / not periodically” saves its state on stable storage.
- The saved state contains sufficient information to restart process execution.
- A consistent global checkpoint is a set of N local checkpoints, one from each process, forming a consistent system state.
- Any consistent global checkpoint can be used to restart process execution upon a failure.
- The most recent consistent global checkpoint is termed as the recovery line.
- In the uncoordinated checkpointing paradigm, the search for a consistent state might lead to **domino effect**.

Domino effect: example



<u>Domino Effect:</u> Cascaded rollback which causes the system to roll back to too far in the computation (even to the beginning), in spite of all the checkpoints

Interactions with outside world

- A message passing system often interacts with the outside world to receive input data or show the outcome of a computation. If a failure occurs the outside world cannot be relied on to rollback.
- For example, a printer cannot rollback the effects of printing a character, and an automatic teller machine cannot recover the money that it dispensed to a customer.
- It is therefore necessary that the outside world perceive a consistent behavior of the system despite failures.

Interactions with outside world (contd.)

- Thus, before sending output to the outside world, the system must ensure that the state from which the output is sent will be recovered despite of any future failure
- Similarly, input messages from the outside world may not be regenerated, thus the recovery protocols must arrange to save these input messages so that they can be retrieved when needed.

Garbage Collection

- Checkpoints and event logs consume storage resources.
- As the application progresses and more recovery information collected, a subset of the stored information may become useless for recovery.
- Garbage collection is the deletion of such useless recovery information.
- A common approach to garbage collection is to identify the recovery line and discard all information relating to events that occurred before that line.

Checkpoint-Based Protocols

- Uncoordinated Check pointing
  - Allows each process maximum autonomy in deciding when to take checkpoints
  - Advantage: each process may take a checkpoint when it is most convenient
  - Disadvantages:
    - Domino effect
    - Possible useless checkpoints
    - Need to maintain multiple checkpoints
    - Garbage collection is needed
    - Not suitable for applications with outside world interaction (output commit)

Coordinated Checkpointing

- Coordinated checkpointing requires processes to orchestrate their checkpoints in order to form a consistent global state.
- It simplifies recovery and is not susceptible to the domino effect, since every process always restarts from its most recent checkpoint.
- Only one checkpoint needs to be maintained and hence less storage overhead.
- No need for garbage collection.
- Disadvantage is that a large latency is involved in committing output, since a global checkpoint is needed before output can be committed to the outside world.

Blocking Coordinated Checkpointing

- Phase 1: A coordinator takes a checkpoint and broadcasts a request message to all processes, asking them to take a checkpoint.
- When a process receives this message, it stops its execution and flushes all the communication channels, takes a tentative checkpoint, and sends an acknowledgement back to the coordinator.
- Phase 2: After the coordinator receives all the acknowledgements from all processes, it broadcasts a commit message that completes the two-phase checkpointing protocol.
- After receiving the commit message, all the processes remove their old permanent checkpoint and make the tentative checkpoint permanent.
- Disadvantage: Large Overhead due to large block time

Communication-induced checkpointing

- Avoids the domino effect while allowing processes to take some of their checkpoints independently.
- However, process independence is constrained to guarantee the eventual progress of the recovery line, and therefore processes may be forced to take additional checkpoints.
- The checkpoints that a process takes independently are local checkpoints while those that a process is forced to take are called forced checkpoints.

Communication-induced checkpoint (contd.)

- Protocol related information is piggybacked to the application messages:
  - receiver uses the piggybacked information to determine if it has to force a checkpoint to advance the global recovery line.
- The forced checkpoint must be taken before the application may process the contents of the message, possibly incurring high latency and overhead:
  - Simplest communication-induced checkpointing:
    - force a checkpoint whenever a message is received, before processing it
  - reducing the number of forced checkpoints is important.
- No special coordination messages are exchanged.