Redundancy in fault tolerant computing

D. P. Siewiorek R.S. Swarz,
Reliable Computer Systems,
Prentice Hall, 1992

Redundancy

Fault tolerance computing is based on redundancy

- **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**

  Fault detection and fault tolerance implemented in sw

HARDWARE REDUNDANCY

Hardware redundancy

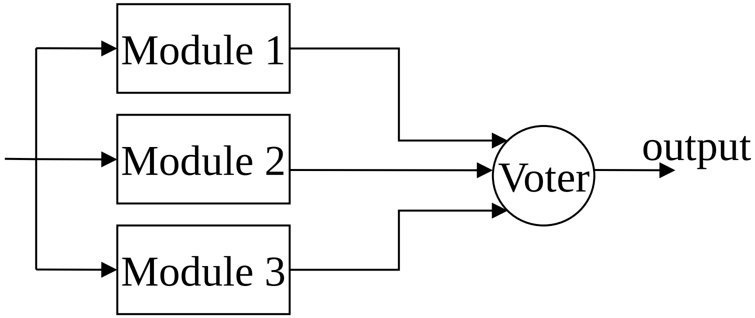
- **Passive fault tolerant techniques**
  - use **fault masking** to hide the occurrence of faults
  - rely upon voting mechanisms to mask the occurrence of faults
  - do not require any action on the part of the system / operator
  - generally do not provide for the detection of faults

- **Active fault tolerance techniques** (dynamic approach)
  - **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

Passive fault tolerance technique

Triple Modular Redundancy (TMR) – fault masking



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

- 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

If we assume that a failed module output is always incorrect tolerates one faulty module

Sometimes some failures in two or more modules may occurr in such a way that a failure is avoided

Example

- stuck-at-1 in a module line; stuck-at-0 in another copy at the same line, correct voted result (**compensating failures)**

- failure at location 127 in a memory; failure at location 10 in another copy, correct voted result (**non overlapping failures)**

Difficulties:

Delay in signal propagation:

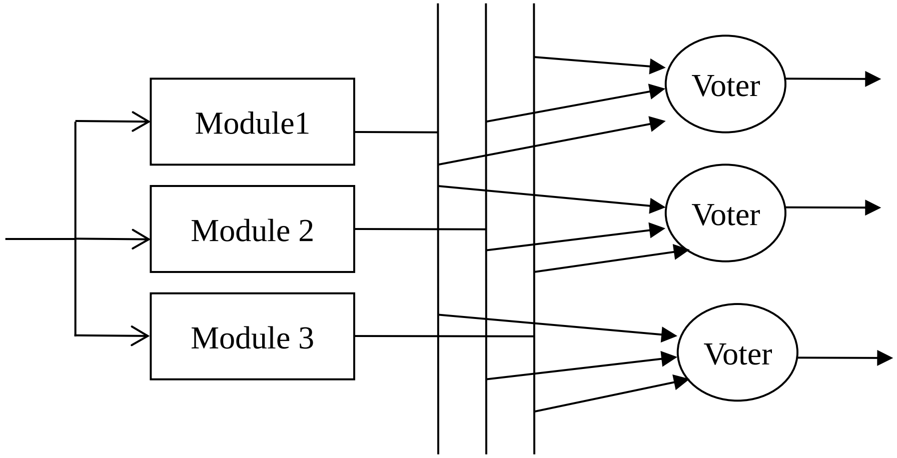
- due to the voter
- due to multiple copies synchronisation

Trade-off : achieved fault tolerance vs hw required

Voter: if the Voter fails, the complete system fails

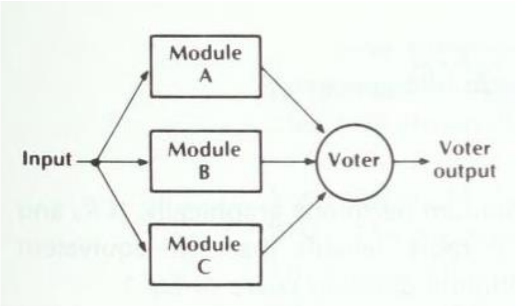
→ Voter is a single point of failure

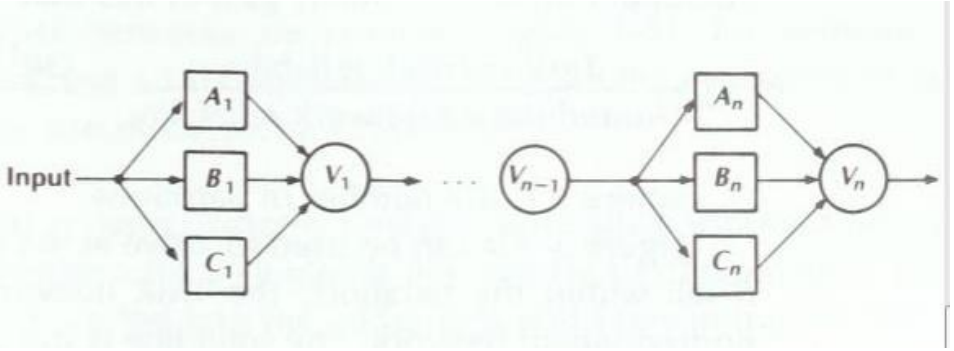
Triplicated Voters in a TMR configuration



If triplicated output is desired, the single point of failure is removed

Cascading TMR

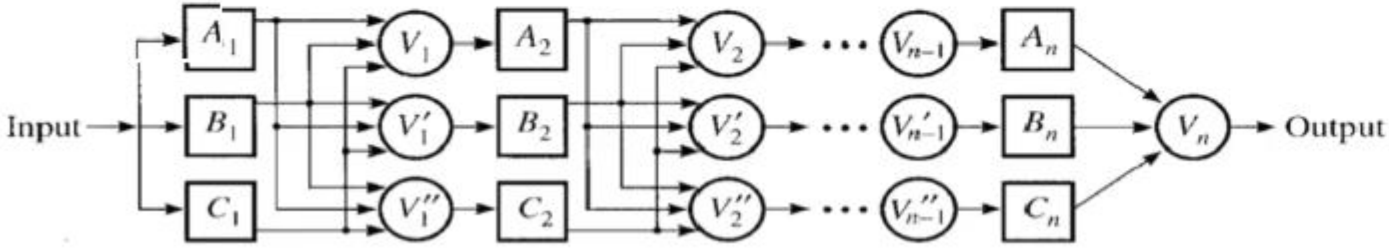




A complex system can be partitioned into smaller subsystems
The effect of partitioning of modules (A, B, C) is that the design can withstand more failures than the solution with only one large triplicated module

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

Cascading TMR with triplicated voters



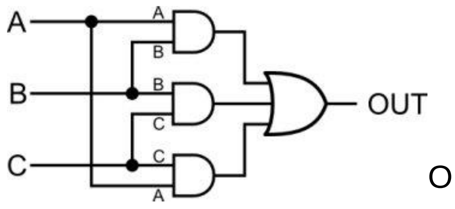
Triplicated voters: voter errors propagates only of one step

Voter:

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

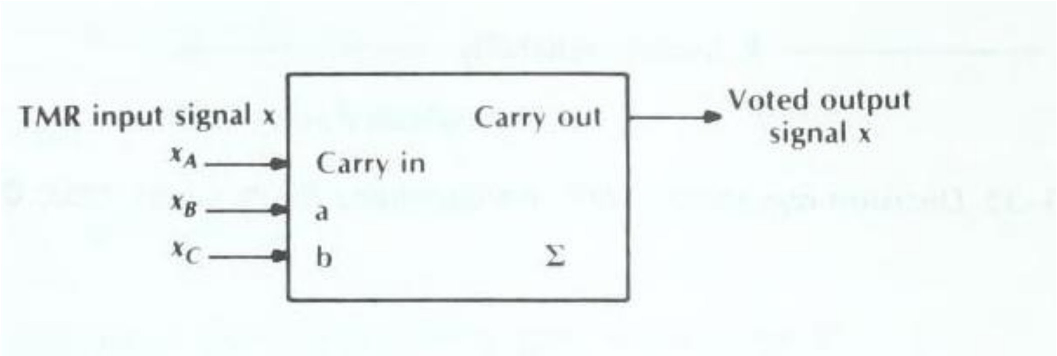
Optimal designs of hardware voters with respect to circuit complexity, number of logic levels, fan-in and fan-out, power dissipation, …, in order to obtain high reliability

1 bit majority voter



OUT = AB + BC + AC

In digital systems majority voting is normally performed by a bit-by-bit basis. For 1 bit line, majority vote can be performed by a 1bit adder.



- **if a module has n output lines, the TMR implementation has n single bit voters**
- **due to the cost of the voting unit, TMR is used at module level**

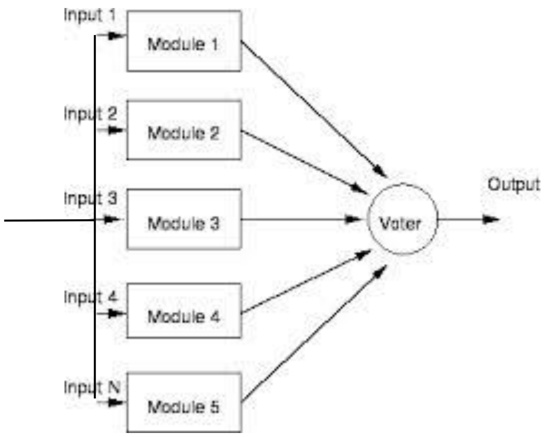
Problems with voting procedure on analog signals:

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)*

N-Modular Redundancy with Voting



- n is made an odd number
- 5MR tolerates 2 faulty modules

Coverage:

m faulty modules, with n = 2m +1

Good for transient faults

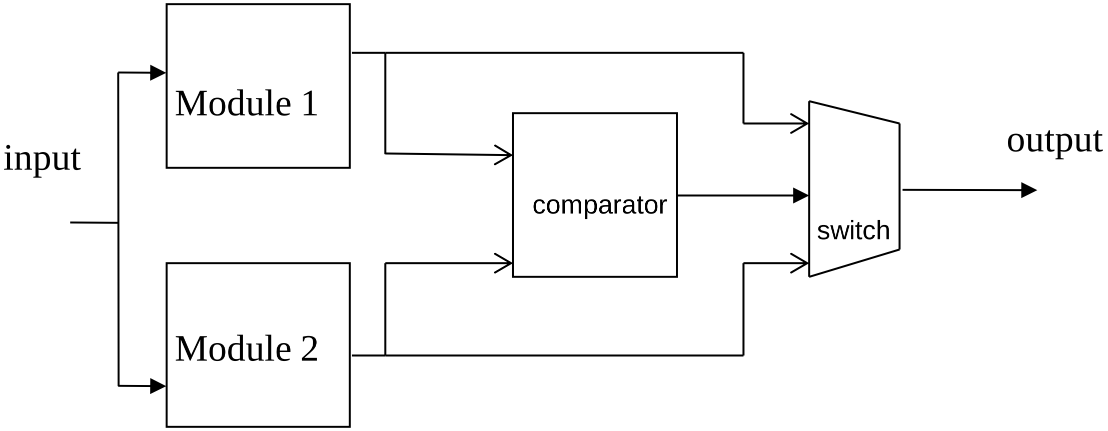
For permanent faults, since the faulty module is not isolated, the protective fault tolerance decreases

Active hw redundancy

1. Duplication with comparison scheme (duplex systems)

- 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 output and reconfigure the switch to select the correct value

The comparator must select the correct value: the comparator uses range checks, assertions, parity checks, ….
executed at each clock period



Sometimes named **dual-modular redundancy**

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 or possible faults in duplicated modules are never detected

Advantages:

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

- Coverage:

  → detects all single faults except those of the comparison element

2. Stand-by sparing

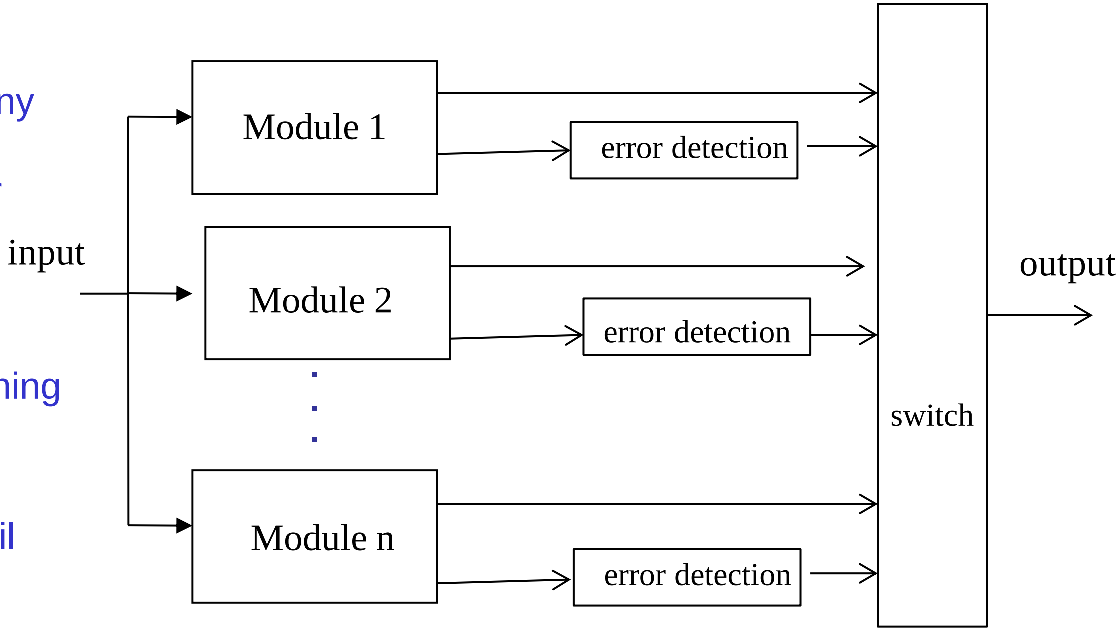
- Part of the modules are operational, part of the modules are spares modules (used as replacement modules)

- 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. **The switch can activate another module.**

- *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



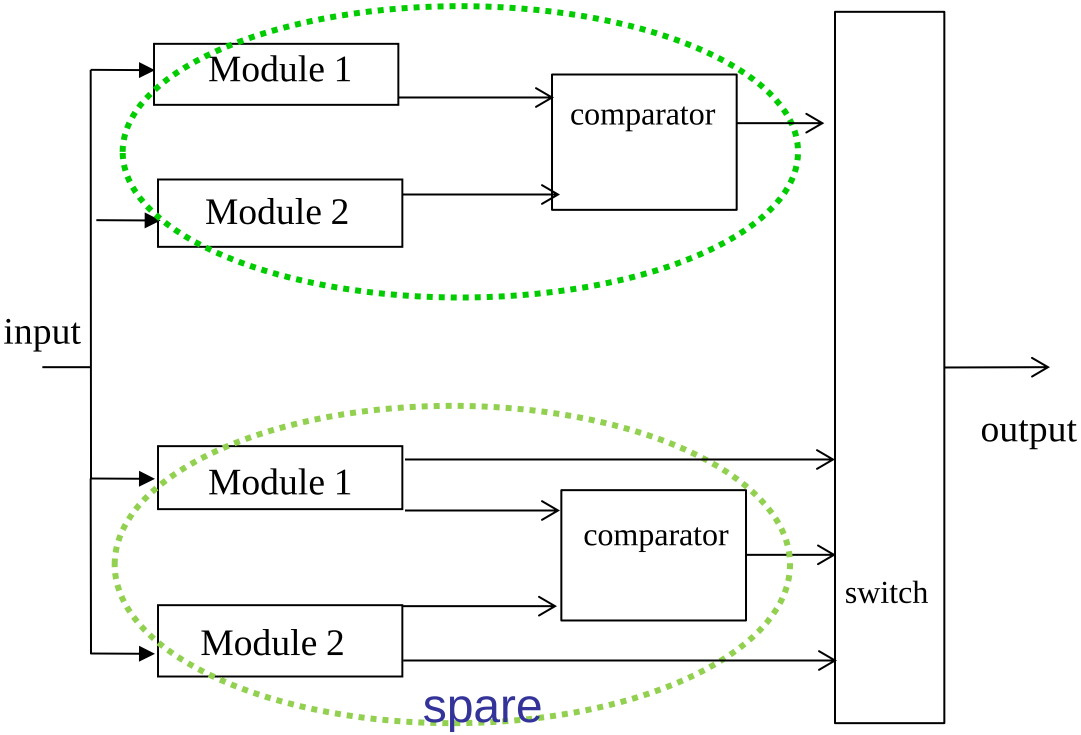
Reconfiguration process can be viewed as a switch that accepts the module's outputs and *error reports* As long as the outputs agree, the spares are not used. When a miscompare occurs, the switch uses the error reports from the modules to identify the faulty module and then select a replacement module.

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 replacemnet using the spare.
- Used in commercial systems, safety critical system (aviation, railways, …)



Pair results are used in a spare arrangment. 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 safety critical applications

NMR with spares (Reconfigurable NMR):

Modules arranged in a voting configuration

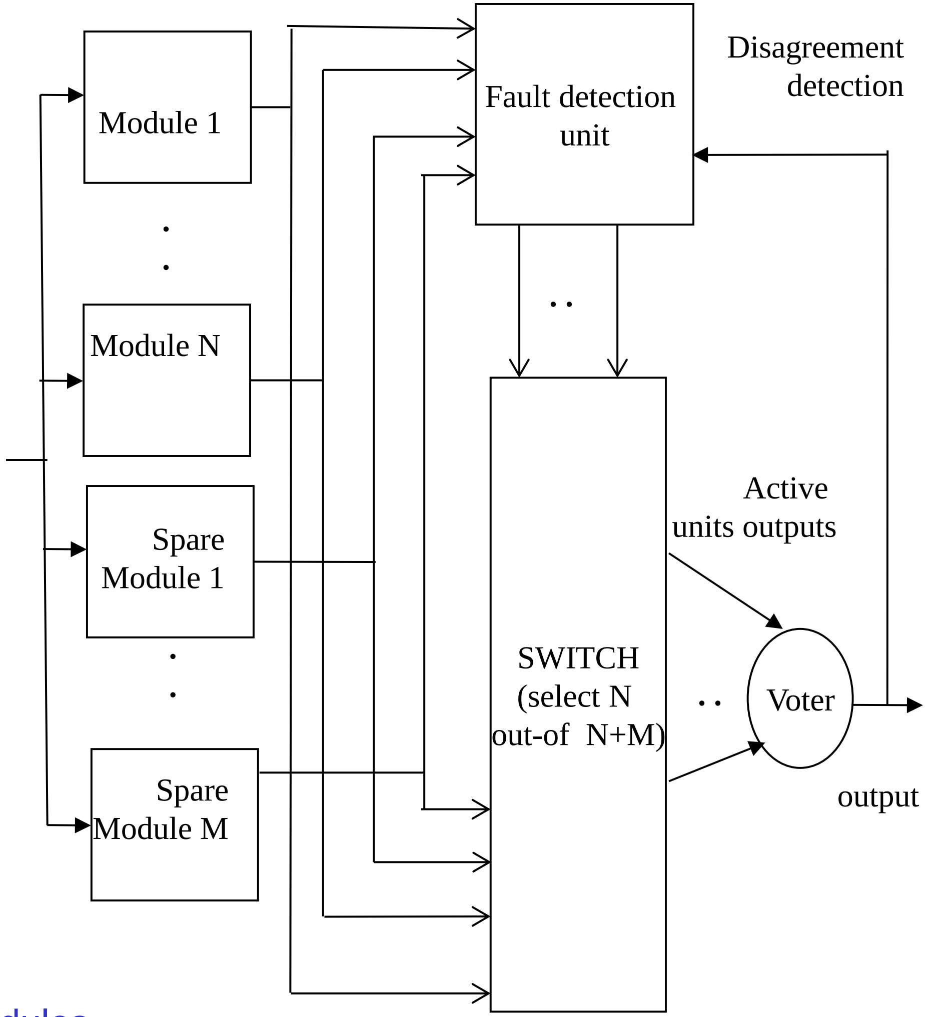
- spares to replace faulty units
- rely on detection of disagreements and determine the module(s) not agreeing with the majority

NMR with spares

- 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)

Hw redundancy techniques

Key differences

Passive: rely on fault masking

Active: rely on error detection, location and recovery

Hybrid: emply 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

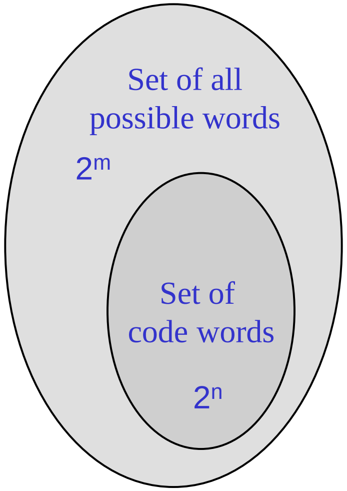
Coding

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

n+c= m 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 …

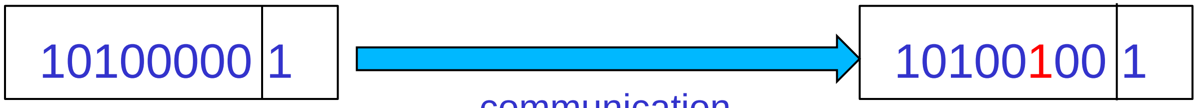


Parity code

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

sender node

receiver node



10100000 1

10100100 1

not a codeword

byte

parity bit

communication channel

one bit flip

Two bit flips are not detected

Coding

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

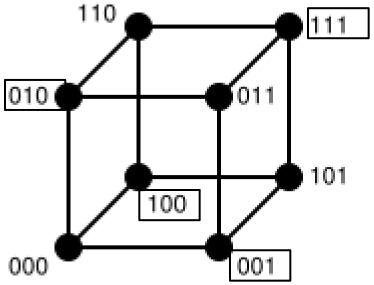
Parity code is a separable code

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

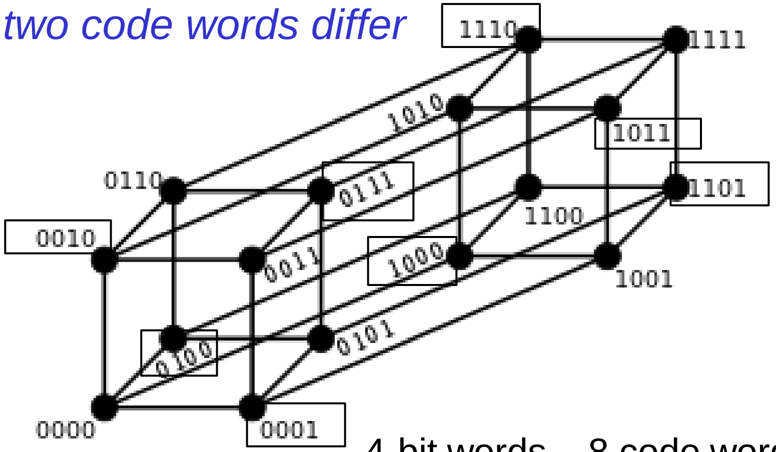
Memories of computer systems. Parity bit added before writing the memory. Parity bit is checked when reading.

Hamming distance (Code distance)

number of bit positions on which two code words differ



3-bit words
boxed words = code words



4-bit words – 8 code words

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

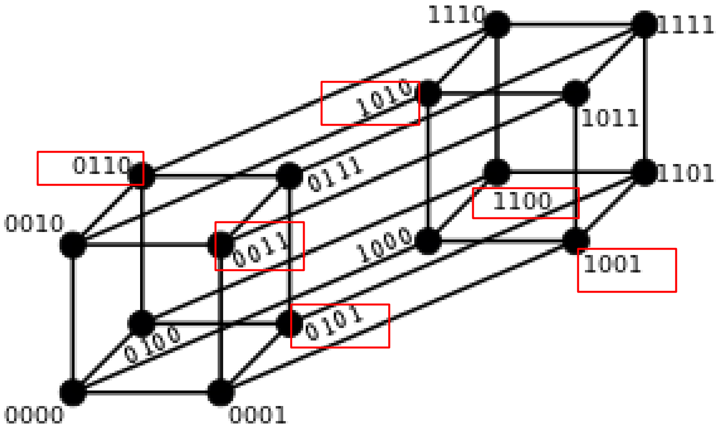
What is the minimum Hamming distance of odd parity? 2
We can detect a 1-bit error
We cannot locate/correct the error
We cannot detect a 2-bit error

2/4 m of n codes

all words with exactly two 1

Hamming distance: 2

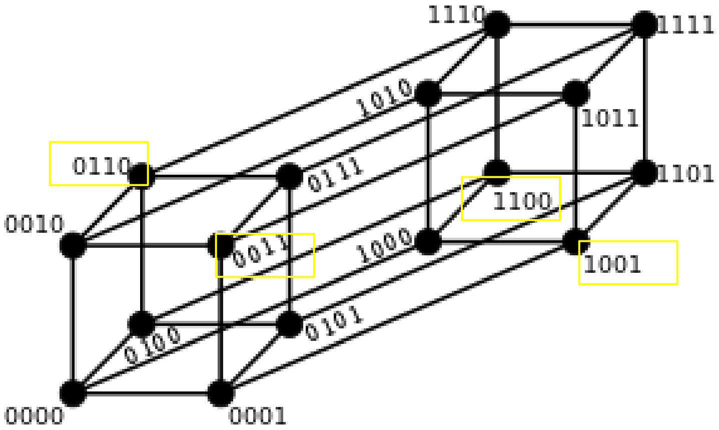
4-bit words – 6 code words

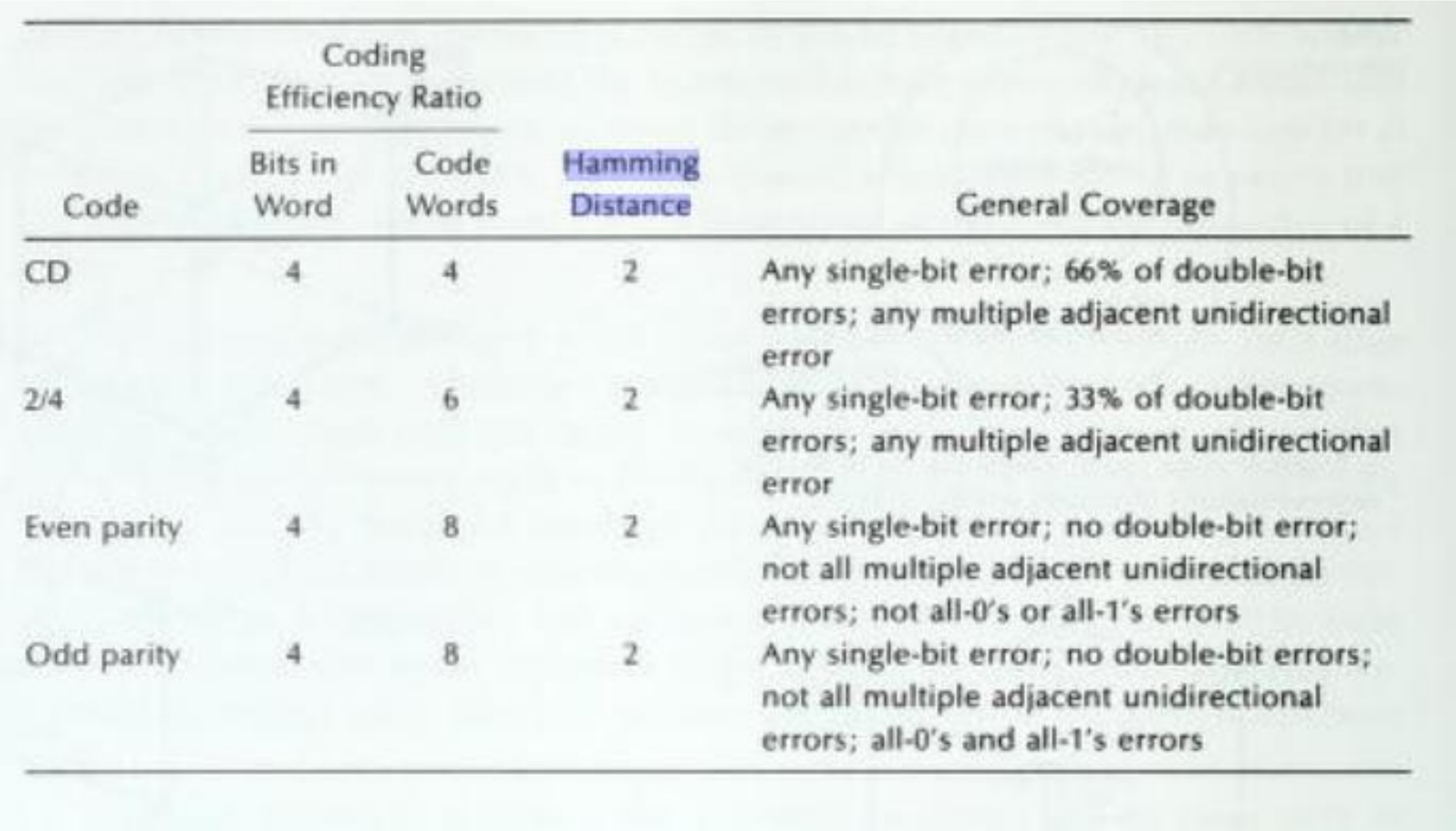


Complemented duplication codes (CD)

Hamming distance: 2

4-bit words – 4 code words





	Coding Efficiency Ratio			
Code	Bits in Word	Code Words	Hamming Distance	General Coverage
CD	4	4	2	Any single-bit error; 66% of double-bit errors; any multiple adjacent unidirectional error
2/4	4	6	2	Any single-bit error; 33% of double-bit errors; any multiple adjacent unidirectional error
Even parity	4	8	2	Any single-bit error; no double-bit error; not all multiple adjacent unidirectional errors; not all-0's or all-1's errors
Odd parity	4	8	2	Any single-bit error; no double-bit errors; not all multiple adjacent unidirectional errors; all-0's and all-1's errors

CD code: 0110 code word

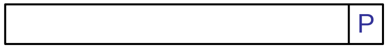
- multiple adjacent unidirectional error 0000 not a code word (detected)
- double bit error 1010 not a code word (detected)
- double bit error 1100 code word (not detected)

2/4 code: 0110 code word

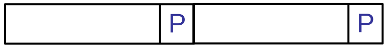
- multiple adjacent unidirectional error 0000 not a code word (detected)
- double bit error 1010 code word (not detected)
- double bit error 1100 code word (not detected)

Parity Code

1. bit-per-word



2. bit-per-byte



3. bit-per-multiple-chip (RAM chips)

when memories are organised using memory chips, *if a chip becomes faulty (multiple bits affected in the same chip), parity code is unable to detect the error.*

Sufficient parity bits are provided to allow each data bit within a chip to be associated with a distinct parity bit

16 bit word 4-bit chips Coverage: single-bit error + chip failure

P0	parity bit for	0,	4,	8,	12
P1	parity bit for	1,	5,	9,	13
P2	parity bit for	2,	6,	10,	14
P3	parity bit for	3,	7,	11,	15
chip0		chip1	chip2	chip3	chip4

Faulty chip: many of P0-P3 affected

Single bit error: one of P0-P3 affected

Linear separable codes: each check bit is calculated as a linear combination of some data bits. Parity codes are linear separable codes: bit calculated as the sum modulo2 of a subset of data bits

Checksumming

applied to large block of data in memories

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)

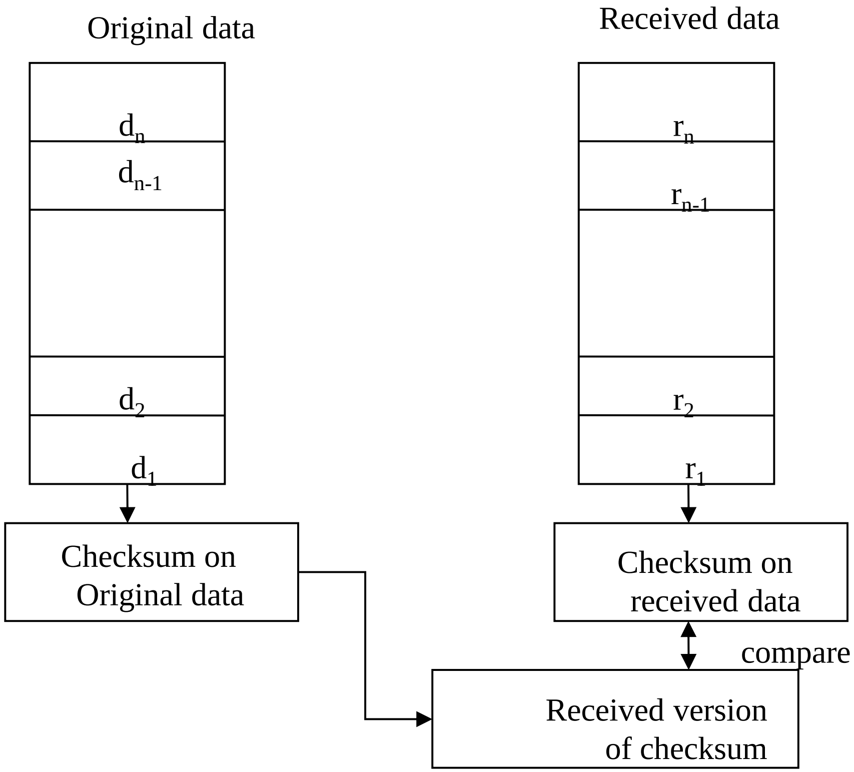
Code word = block + checksum

- 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

Coverage: single fault



Checksummming

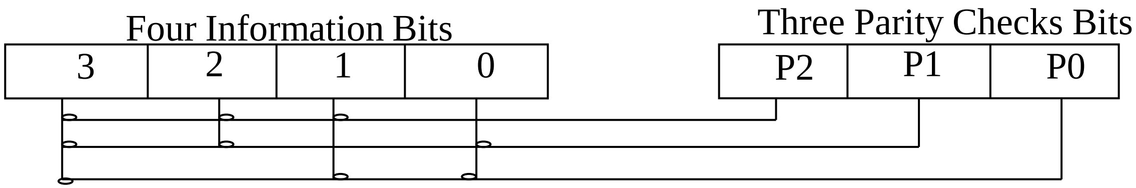
Disadvantages

- 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

ECC – Error Correcting Codes

Parity code be used for location and correction of errors?



Four Information Bits

3	2	1	0

Three Parity Checks Bits

P2	P1	P0

Bit Error	Parity group affected		
3	P2	P1	P0
2	P2	P1	
1	P2		P0
0		P1	P0
P2	P2		
P1		P1	
P0			P0

m = number of information bits

k = number of parity bits

2^K = number of outcomes of the parity checking process

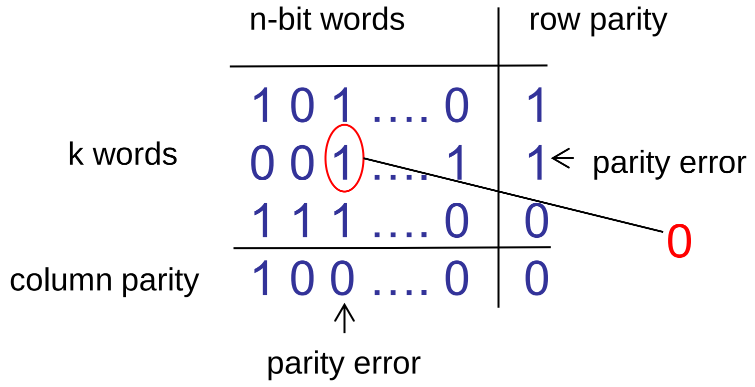
m+k = number of single bit errors

$2^K > m+k$

disadvantage: 75% of redundancy

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	0
column parity	1 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

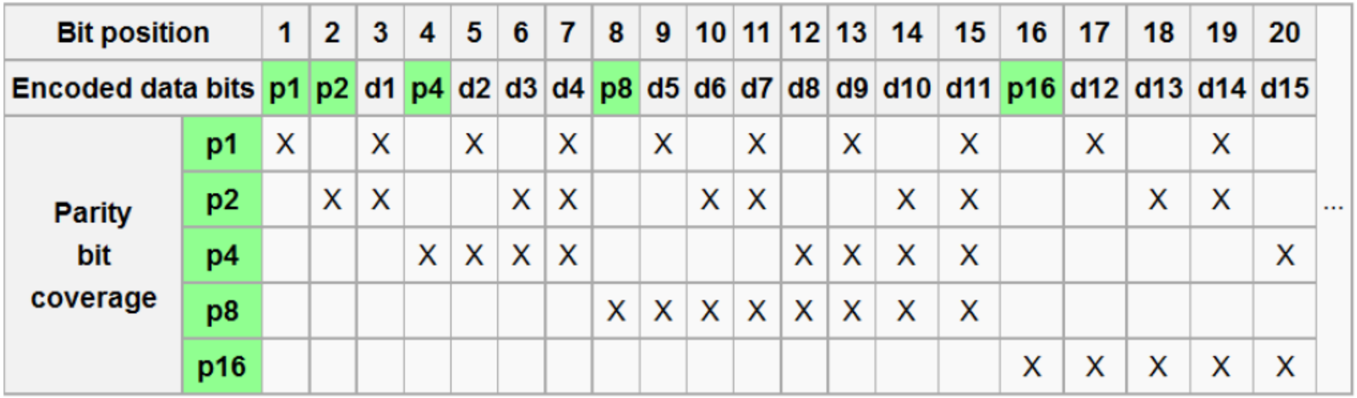
Hamming Codes

Parity bits spread through all the data word

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

Number the bit positions starting from 1: bit 1, 2, 3, 4, 5, etc.

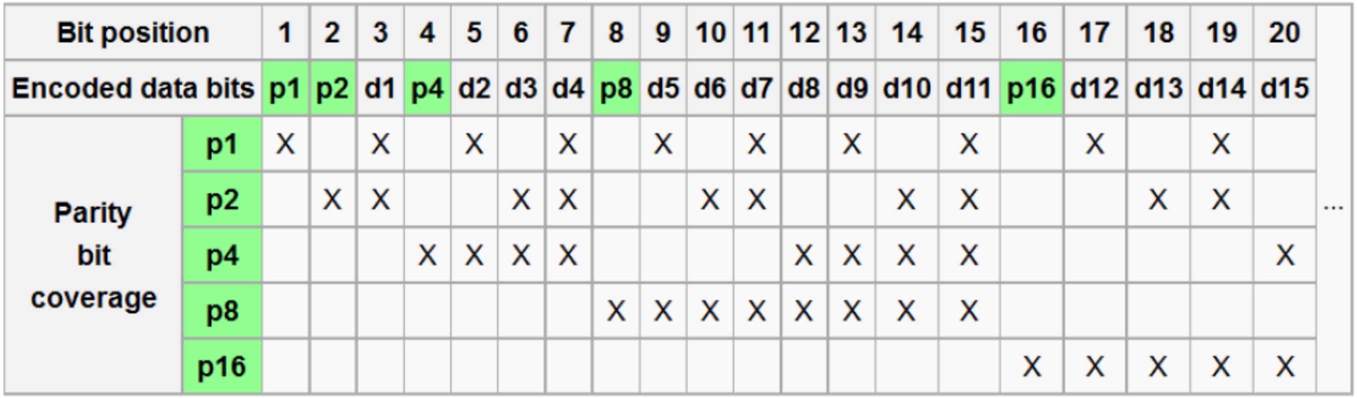
Parity bits
all bit positions that are powers of two : 1, 2, 4, 8, etc.
Data bits
all other bit positions



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	X						
	p16																X	X	X	X	X	

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.

Parity bit pj covers all bits whose position has the j least significant bit set



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	X						
	p16																X	X	X	X	X	

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

Parity bit p2 covers all bit positions which have the second least significant bit set (------1-):
bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.

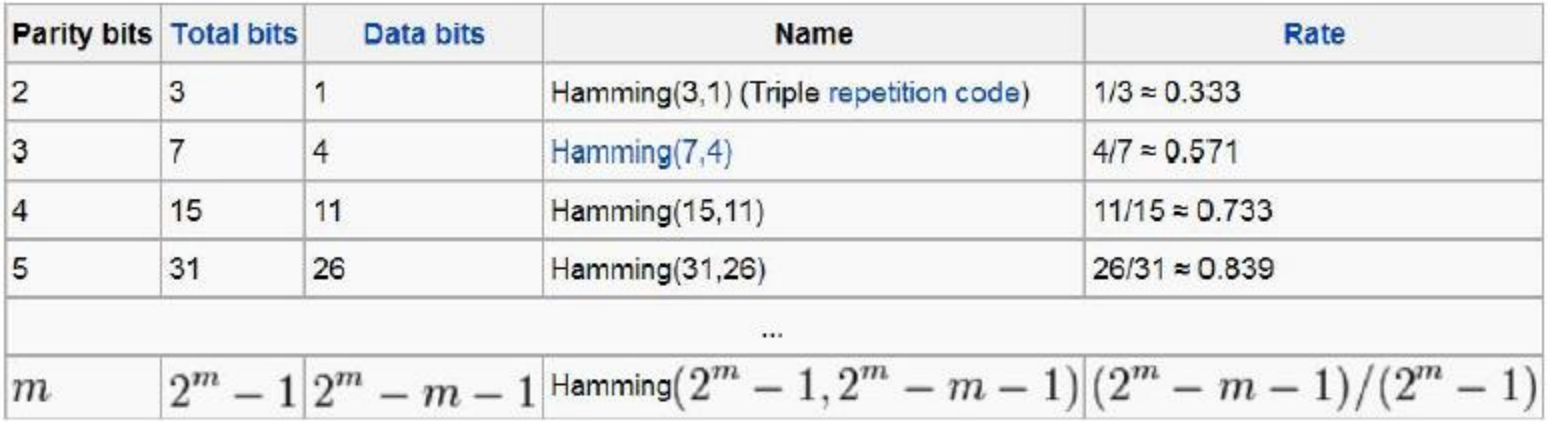
Parity bit p4 covers all bit positions which have the third least significant bit set (-----1--):
bits 4–7, 12–15, 20–23, etc.

Parity bit p8 covers all bit positions which have the fourth least significant bit set (----1---):
bits 8–15, 24–31, 40–47, etc.

Overlap of control bit:

a data bit is controlled by more than one parity bits

Overhead /fault tolerance



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

Minimum Hamming distance: 3

Double-error detection code
Single-error correction code ⇨ SEC-DED code

Self checking circuitry

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

Given a set of faults, **design of comparators and code checkers capable of detecting their own faults (checking the checker)**

Self-checking circuit:

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 → correct code output

fault + code input → (correct code output) or (nocode output)

Self checking circuitry

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 a 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

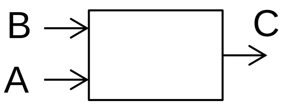
Example:

two signal input comparator
output 0 if inputs are equal; 1 otherwise

input and output coding: 1/2 code
(dual-rail signal: coded signal whose two bits are always complementary)

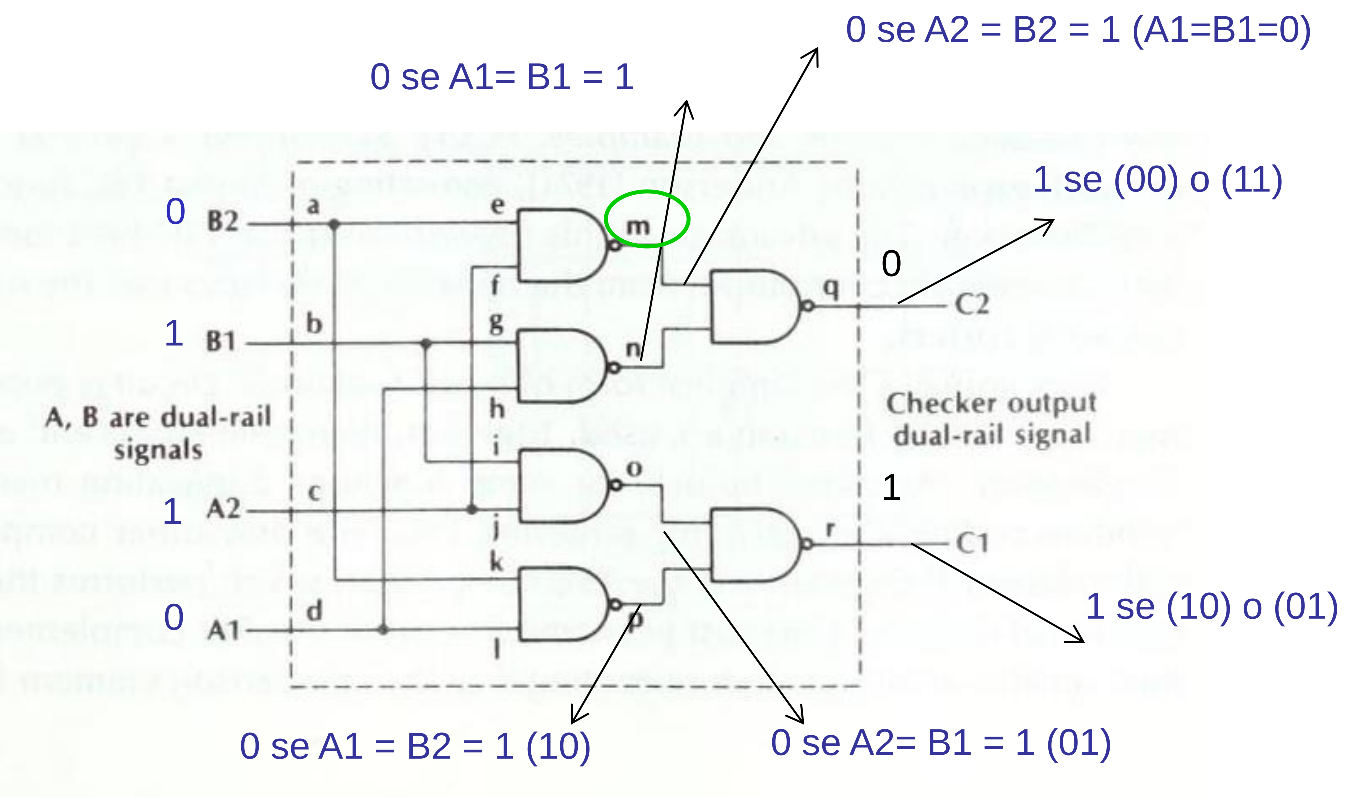
m/n code:
m bit equal to 1

Two input comparator: output 0 if inputs are equal; 1 otherwise



A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

D. P. Siewiorek R.S. Swarz,
Reliable Computer Systems,
Prentice Hall, 1992



0 se A2 = B2 = 1 (A1=B1=0)

0 se A1= B1 = 1

1 se (00) o (11)

0

0

1

1

0

1

1 se (10) o (01)

0 se A1 = B2 = 1 (10)

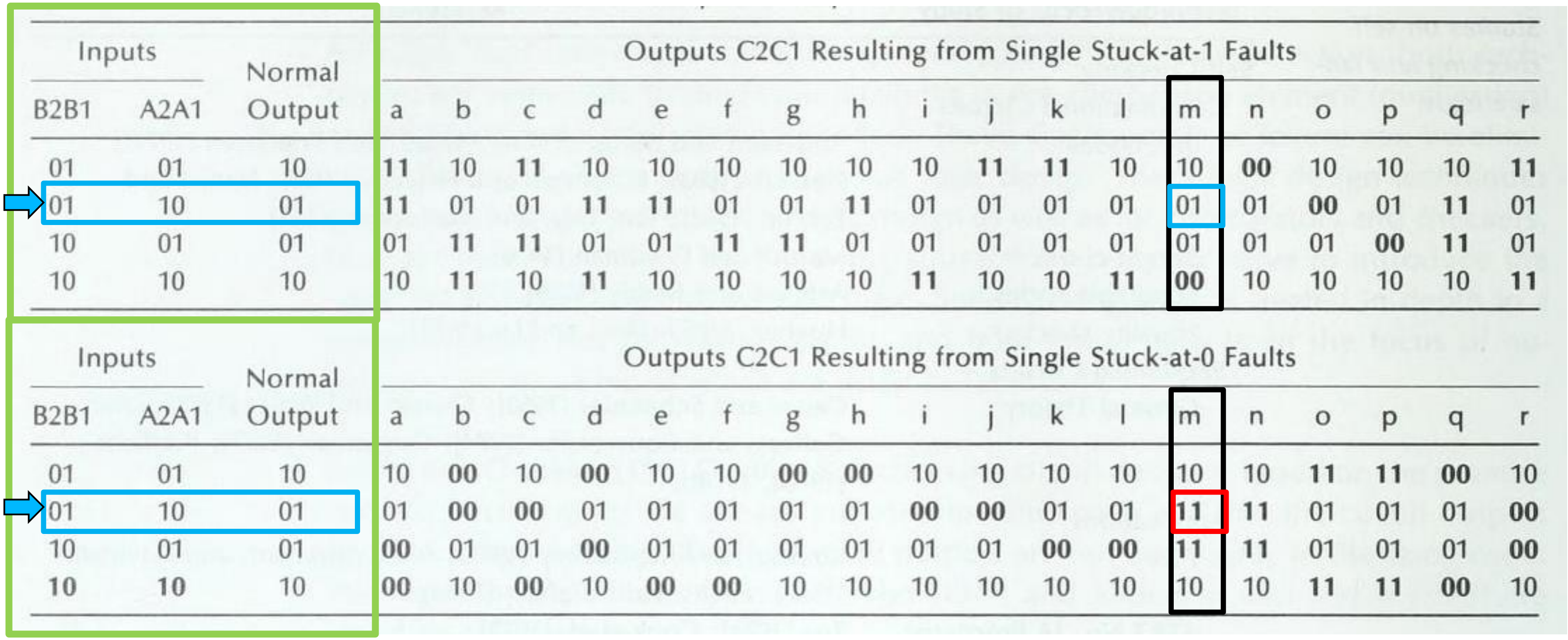
0 se A2= B1 = 1 (01)

Set of faults:
stuck-at-1, stuck-at-0
of each line
(a, b, c, d, e, ……, q, r)

Fault free
A =0, B =1
 m=1, n =1, q=0
o = 0, p=1, r= 1
c2=0
c1=1
code
different input

Faulty:
A=0, B=1
m: stuck-at-0
c2 = 1
c1 = 1
noncode

Faulty:
A=0, B=1
m: stuck-at-1
c2=0
c1=1
code
different input

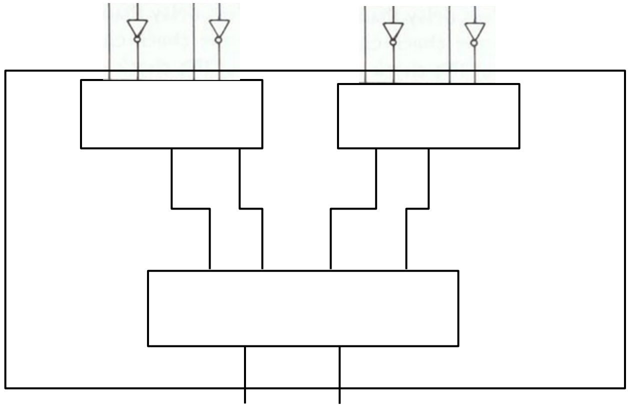


Inputs		Normal	Outputs C2C1 Resulting from Single Stuck-at-1 Faults																	
B2B1	A2A1	Output	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
01	01	10	**11**	10	**11**	10	10	10	10	10	10	**11**	**11**	10	10	**00**	10	10	10	**11**
01	10	01	**11**	01	01	**11**	**11**	01	01	**11**	01	01	01	01	01	01	**00**	01	**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		Normal	Outputs C2C1 Resulting from Single Stuck-at-0 Faults																	
B2B1	A2A1	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

D. P. Siewiorek R.S. Swarz, Reliable Computer Systems, Prentice Hall, 1992

n-input TSC comparator: a tree whose nodes are two-input self checking comparators



RAID

RAID: Redundant Arrays of Independent Disks

disk organization techniques that manage a large numbers of disks, providing a view of a single disk of

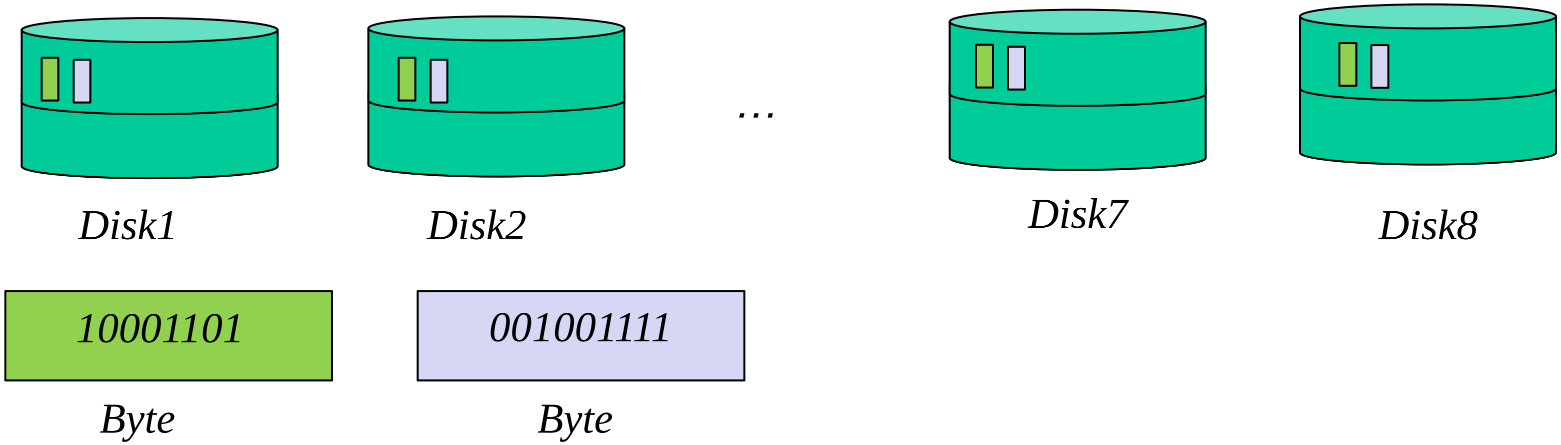
- high capacity and high speed by using multiple disks in parallel, and

- high reliability by storing data redundantly, so that data can be recovered even if a disk fails

Bit-level striping – split the bits of each byte across multiple disks

In an array of eight disks, write bit *i* of each byte to disk *i.*

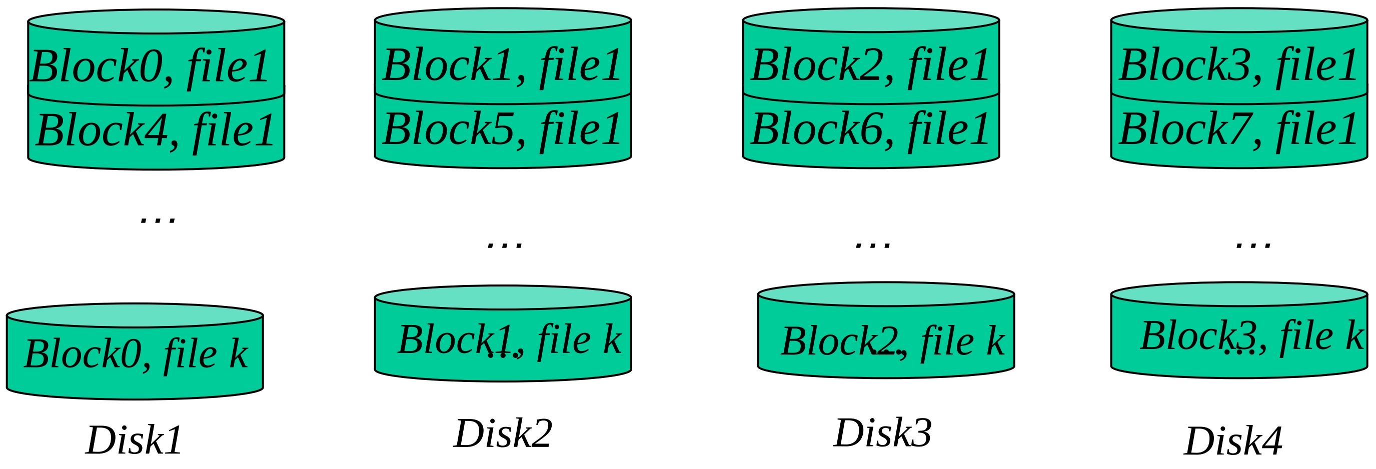
Each access can read data at eight times the rate of a single disk.



RAID

Block-level striping – with *n* disks, block *i* of a file goes to disk (*i* mod *n*) + 1

Requests for different blocks can run in parallel if the blocks reside on different disks. A request for a long sequence of blocks can utilize all disks in parallel

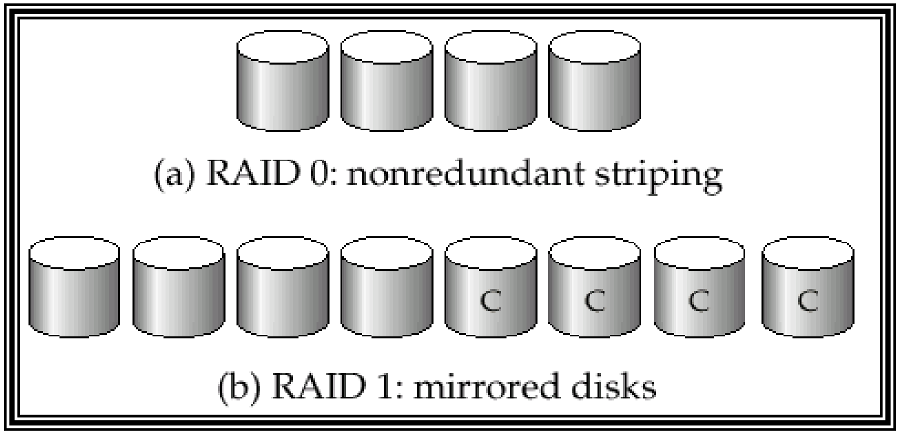


RAID Levels

Schemes to provide redundancy at lower cost by using disk striping combined with parity bits

Different RAID organizations, or RAID levels, have differing cost, performance and reliability characteristics

- **RAID Level 0**: Block striping; non-redundant.
  - Used in high-performance applications where data lose is not critical.
- **RAID Level 1**: Mirrored disks with block striping
  - Popular for applications such as storing log files in a database system.



(a) RAID 0: nonredundant striping

(b) RAID 1: mirrored disks

RAID Levels (Cont.)

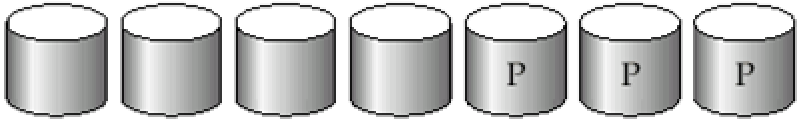
RAID Level 2: **Memory-Style Error-Correcting-Codes** (ECC) with bit striping

Each byte is assigned a parity bit: the bit records whether the number of bits in the byte that are set to 1 is even or odd

If one bit in the byte gets damaged the parity of the byte changes and will not match the computed parity
ALL 1-BIT ERRORS ARE DETECTED

Error correcting codes store extra bits to reconstruct the data if a single bit gets damaged

Disks labelled P store the ECC



(c) RAID 2: memory-style error-correcting codes

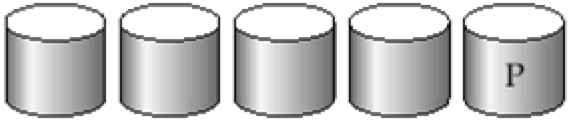
4 data bits – 3 parity bits SEC code

RAID Levels (Cont.)

RAID Level 3: Bit-Interleaved Parity

exploit the fact that **disk controllers, unlike memory systems, can detect whether a sector has been read correctly**

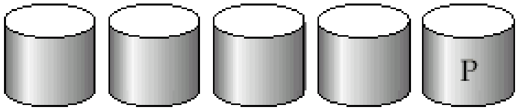
a single parity bit is enough for error correction, not just detection, since we know which disk has failed



(d) RAID 3: bit-interleaved parity

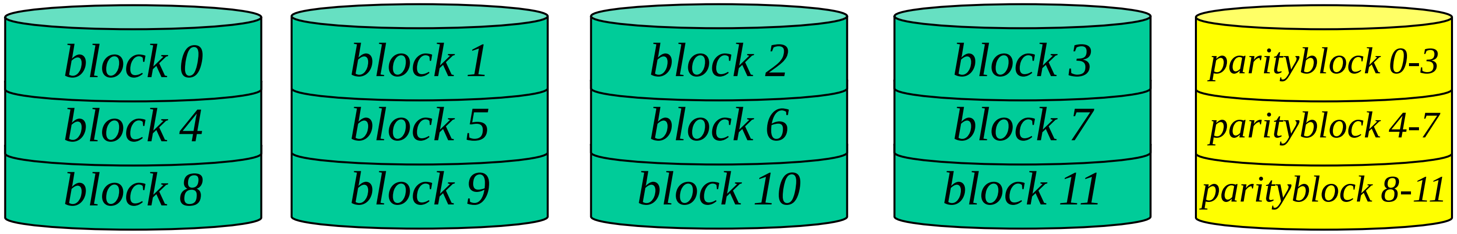
RAID Levels (Cont.)

RAID Level 4: Block-Interleaved Parity; uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from *N* other disks.



(e) RAID 4: block-interleaved parity

Example

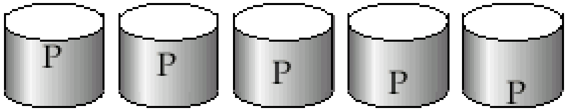


block 0	*block 1*	*block 2*	*block 3*	*parityblock 0-3*
block 4	*block 5*	*block 6*	*block 7*	*parityblock 4-7*
block 8	*block 9*	*block 10*	*block 11*	*parityblock 8-11*

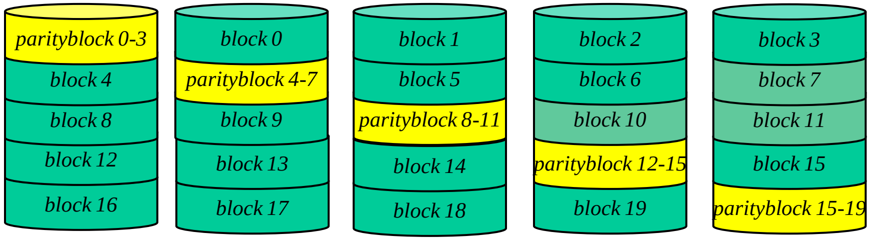
RAID Levels (Cont.)

RAID Level 5: Block-Interleaved Distributed Parity; partitions data and parity among all *N* + 1 disks, rather than storing data in *N* disks and parity in 1 disk.

E.g., with 5 disks, parity block for *n*th set of blocks is stored on disk (*n mod* 5) + 1, with the data blocks stored on the other 4 disks.



(f) RAID 5: block-interleaved distributed parity



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

RAID Levels (Cont.)

RAID Level 5 (Cont.)

For each set of N logical blocks, one of the disks store the parity and the other N disks store the blocks

The P’s are distributed across all the disks

A parity block can not store parity for bocks of the same disk, since then a disk failure would result in loss of data as well as of parity(failure not recoverable)

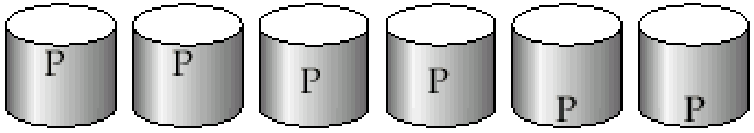
RAID Levels (Cont.)

RAID Level 6: P+Q Redundancy scheme;
similar to Level 5, but stores extra redundant information to guard against multiple disk failures.

Better reliability than Level 5 at a higher cost; not used as widely.

Level 6, instead of using parity, uses ECC.

In the figure 2 bits of redundant data are stored for every 4 bits of data and the system can tolerate two disk failures



(g) RAID 6: P + Q redundancy

TIME REDUNDANCY

Time redundancy techniques

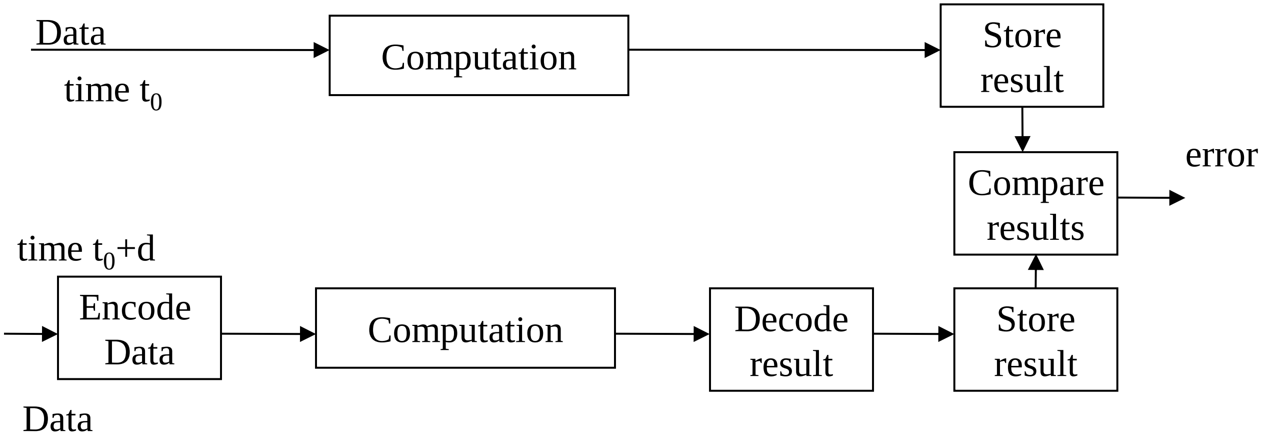
Attempt to reduce the amount of extra hw at the expense of using additional time

1. 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
(after a transient fault system data can be completely corrupted)

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



Time redundancy techniques

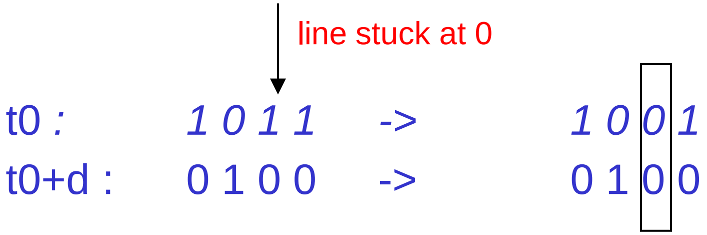
Example

- errors in data transmitted over a parallel bus
- stuck at 0 of a line of the bus

t0: transmit original data

t0+d : transmit complement data

When the 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 should alternate between a logic 1 and a logic 0 (*alternating logic*)

SOFTWARE REDUNDANCY

Software redundancy techniques

Due to the large cost of developing software, most of the software dependability effort has focused on

fault prevention techniques and testing strategies

Fault tolerant software

Single-version approaches

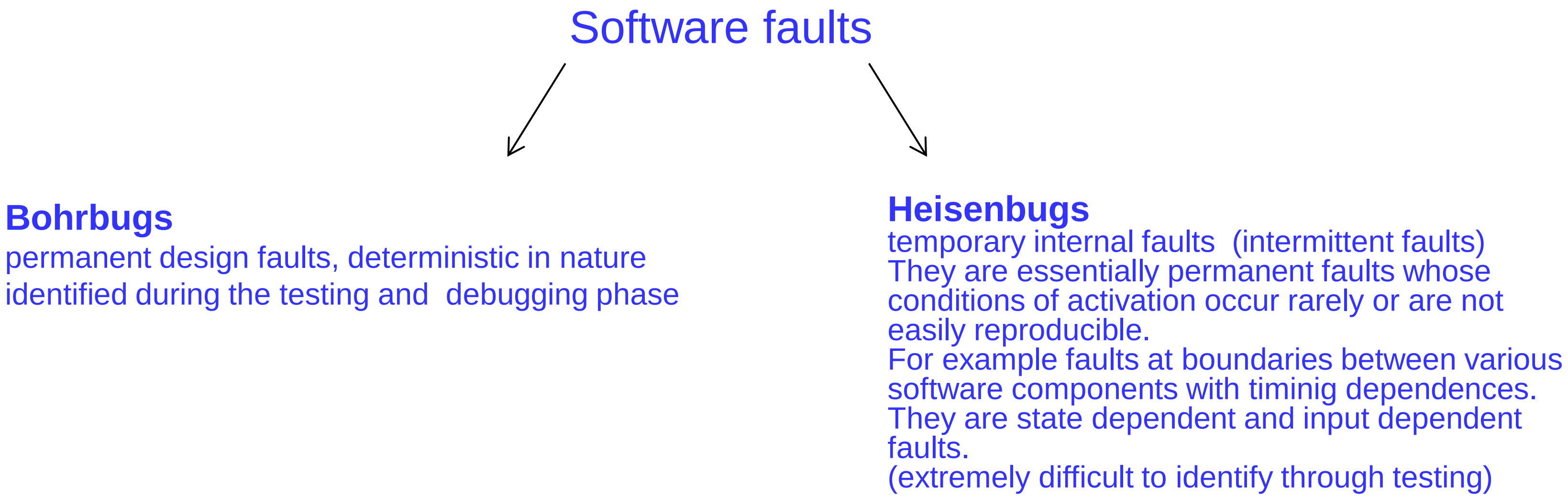
one code with error detection and fault tolerant capabilities inside

Multi-version approaches

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

Single-version software fault tolerance techniques

(redundancy applied to a single version of software to detect errors and recover)



Software faults

Bohrbugs
permanent design faults, deterministic in nature
identified during the testing and debugging phase

Heisenbugs
temporary internal faults (intermittent faults)
They are essentially permanent faults whose conditions of activation occur rarely or are not easily reproducible.
For example faults at boundaries between various software components with timinig dependences.
They are state dependent and input dependent faults.
(extremely difficult to identify through testing)

Basis to implement fault tolerance

- software architecture (modularization)
- system closure principle
- self-checking and self-protection principle

Software architecture
(basis to implement fault tolerance)

1) Modularization and Partitioning
functional independent modules + control modules (that coordinate the execution)
provide isolation between functionally independent modules

2) Hierarchy and connectivity of components
used to analyse error propagation

3) Temporal structuring of the activity between interacting components

atomic action: activity in which the components interact with each other and there is no interaction with the rest of the system for the duration of the activity
provide a framework for error confinement and recovery
(if a failure is detected during an atomic action, only the participating components can be affetcted)

Error confinement areas, with boundary at interfaces between components

System closure fault tolerance principle

no action is permissible unless explicitly authorized (mutual suspicion)

1. Each component is only granted the capabilities needed to execute its function

2. Each component examines each request or data item from other components before acting on it
For example, each software module checks legality and reasonableness of each request received

3. A capability disabled by an error disables a valid action
(it does not result in an undesirable action)

Error detection and confinement
Added overhead, need for providing:
signalling back to requestor and own strategy for dealing with erroneous requests

Self-protection and self-checking principles

Software system: a set of communicating components

Component (self-protection): protect itself by detecting errors in the information received by other interacting components

Component (self-checking): able to detect internal errors and take appropriate actions to prevent the propagation to other components

- Exceptions
- Checkpointing
- Redundancy at code level
- Control flow errors
- …..

Exception mechanism: error detection and recovery

Exception handling

exceptions are signalled by the error detection mechanism

catch() clauses implement the appropriate error recovery

Three classes of exceptions

interface exceptions
(invalid service request, triggered by the self-protection mechanism, handled by the module that requested the serice)

internal local exceptions
(an error in the internal operations of the module, triggered by the error detection mechanism of the module, handled by the module)

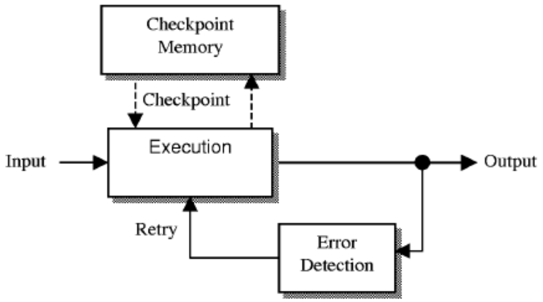
failure exceptions
(detected error, not handled by the fault processing mechanism. Tell the module requesting the service that the service had a failure)

Error confinement is essential to design effective exception handlers

Checkpointing and restart recovery mechanism

Most of the faults at this stage are *Heisenbugs,* hence these faults result in transient failures, i.e., failures which may not recur if the software is restarted.

Restart is usually enough to successful completion of the execution of the module

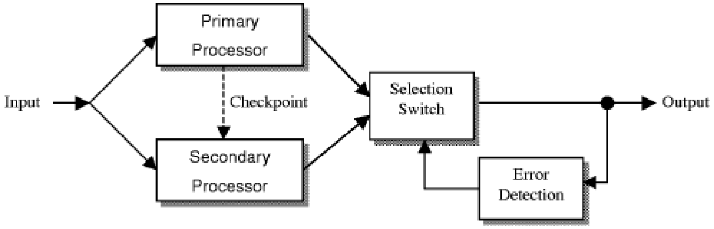


W. Torres-Pomales
Software fault tolerance: A tutorial
NASA,/TM-2000-210616, 2000

Checkpointing and restart recovery mechanism

- **Static**
  restart from predetermined states
  (initial state or intermediate state, ..)
- **Dynamic**
  restart from checkpoints created during
  the execution of the module (backword error recovery)

Process pair



Process pair:

two processors
uses the same version of the software
the primary processor sends checkpoints to the other
error detection:
the secondary processor takes the role of primary and starts from the checkpoint

C/C++ language: checkpoint libraries

Redundancy at code level

1. Duplication implemented in a compiler

   RECCO: a REliable c/c++ Code COmpiler for dependable applications

   - duplicate variables (code analysis to find important variables - read variables, variables keeping a value for a long time, lifetime of variables)

   - duplicate instructions - selective instruction duplication (e.g., instruction that are executed more frequently)

   Covered faults: data errors, memory instruction in memory errors

2. Add information to the Control Flow Graph, and check conditions at run-time

   Covered faults: control flow errors

3. ……………………………

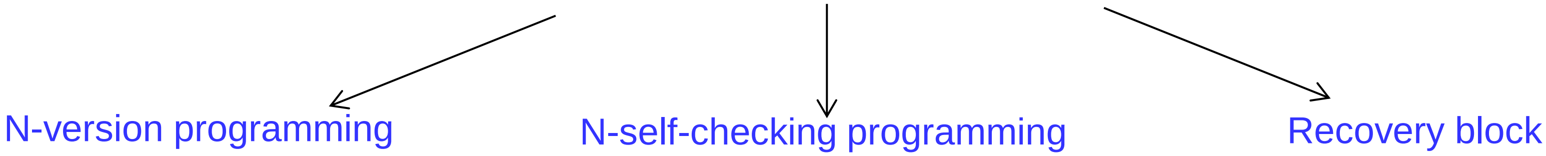
Multi-version approaches

replicate the complete program

Software diversity

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

Independent generation of N >= 2 functionally equivalent programs, called *versions*, **from the same initial specification**.



N-version programming

N-self-checking programming

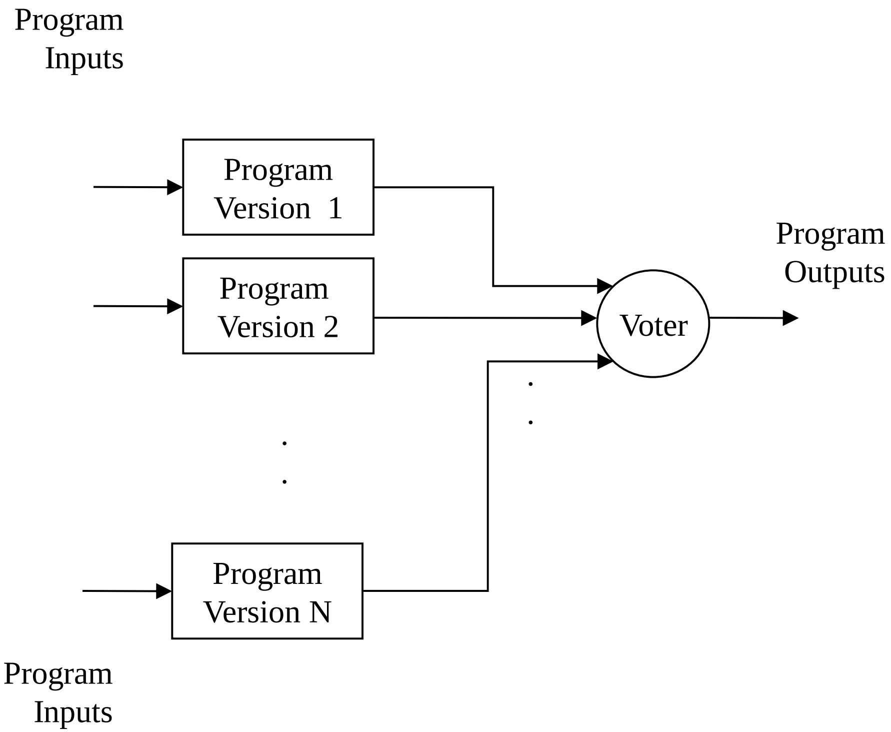
Recovery block

N-version programming

- 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 (fault avoidance)

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

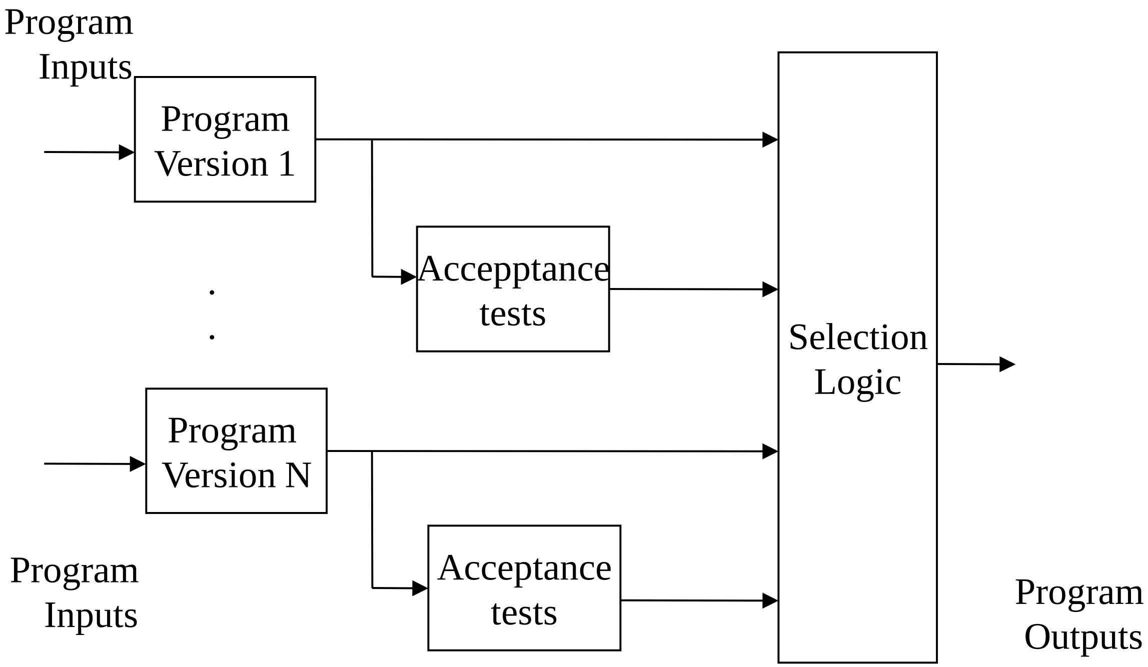
- must assure that the input data vector to each of the versions is identical

- 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

- Cannot adopt the hardware analogy and assume versions fail independently
- Empirical evidence that there will be common faults
- There is evidence that diversity delivers some improvement over single versions

related faults may result from dependencies in the separate designs and implementations

(example: specification mistakes)

Functional diversity

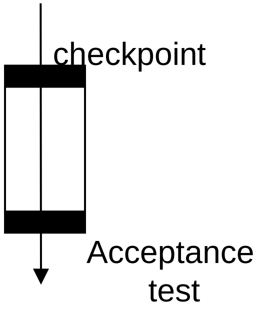
assign to independent software versions diverse functions that compute the same task

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

- Accettability of the result is decided by an acceptance test **T**
- Primary alternate, secondary alternates



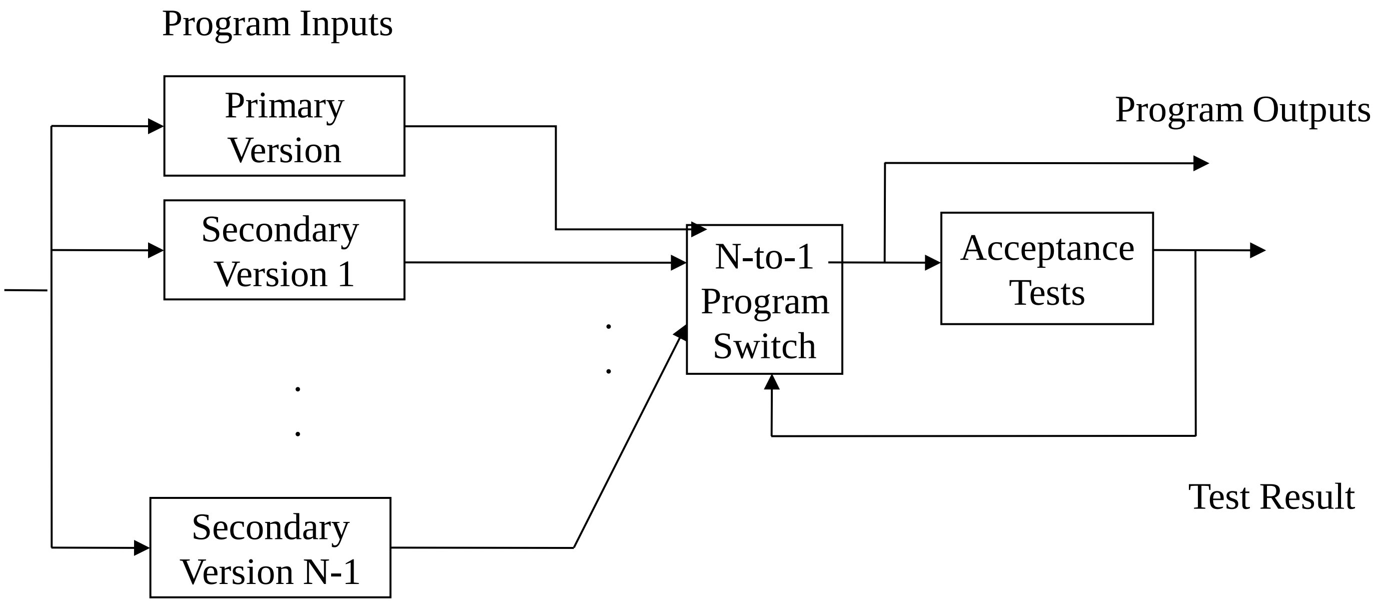
Basic structure:

Ensure T
By P
else by Q
Else error

1. Each recovery block contains variables global to the block that will be automatically checkpointed if they are altered within the block.

2. Upon entry to a recovery block, 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.
   If the test is failed or the alternative fails to execute, the content of the recovery cache pertinent to the block is reinstated, and the second alternate is executed. This cycle is executed until either an alternative is successful or no more alternatives exist.
   In this case an error is reported.

Recovery block software fault tolerant technique



- A single acceptance test
- Only one single implementation of the program is run at a time
- Combines elements of checkpointing and backup
- Minimizes the information to be backed up
- 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.

Recovery block in concurrent systems

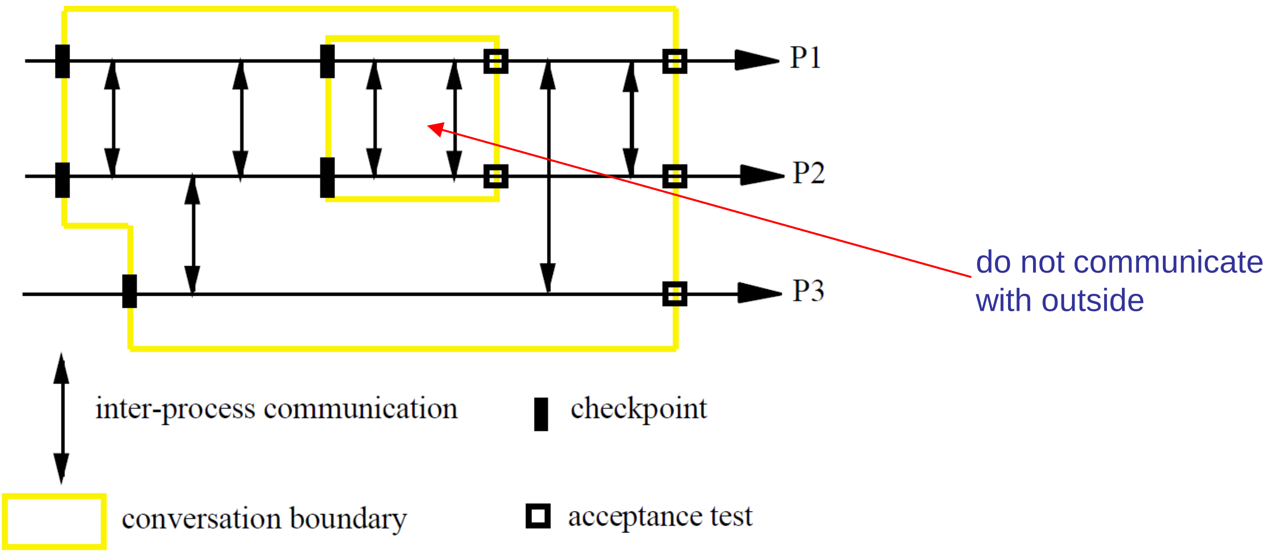
When a system of cooperating processes employs recovery blocks, each process will be continually establishing and discarding checkpoints, and may also need to restore to a previously established checkpoint.

However, if recovery and communication operations are not performed in a coordinated fashion, then the rollback of a process can result in a cascade of rollbacks that could push all the processes back to their beginnings — the domino

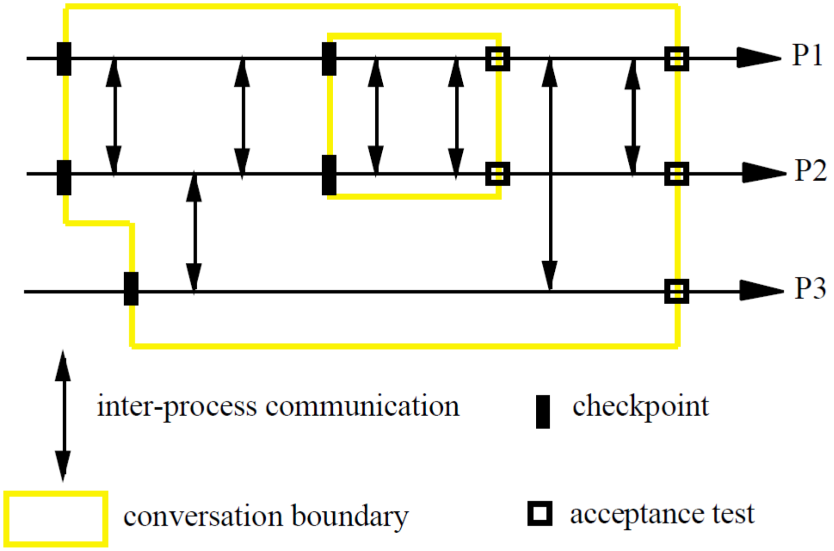
➡ the notion of *conversation*

Conversion scheme

- one of the fundamental approaches to structured design of fault-tolerant concurrent programs
- provides a means of coordinating the recovery blocks of interacting processes to avoid the “domino effect”



Example where three processes communicate within a conversation and the processes P1 and P2 communicate within a nested conversation



The operation of a conversation is: (i) on entry to a conversation a process establishes a checkpoint; (ii) if an error is detected by any process then all the participating processes must restore their checkpoints; (iii) after restoration all processes use their next alternates; and (iv) all processes leave the conversation together.

Deserters in a conversation: real-time applications may suffer from the possibility of desertes in a conversation— if a deadline is to be met then a process that fails to reach its acceptance test could cause all the processes in the conversation to miss that deadline

Observations

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

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**)

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.