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

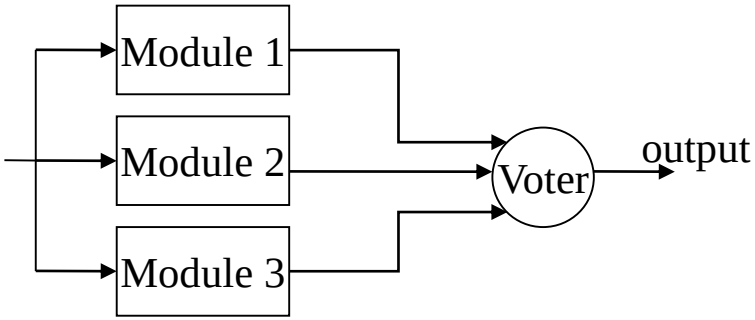
- **Passive fault tolerant techniques**
  - use **fault masking** to hide the occurrence of faults
  - rely upon voting mechanisms to mask the occurrence of faults
  - do not require any action on the part of the system / operator
  - generally do not provide for the detection of faults

- **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

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

Example

- stuck-at-1 in a module line; stuck-at-0 in another copy at the same line, correct voted result
- failure at location 127 in a memory; failure at location 10 in another copy, correct voted result

Difficulties:

Delay in signal propagation:

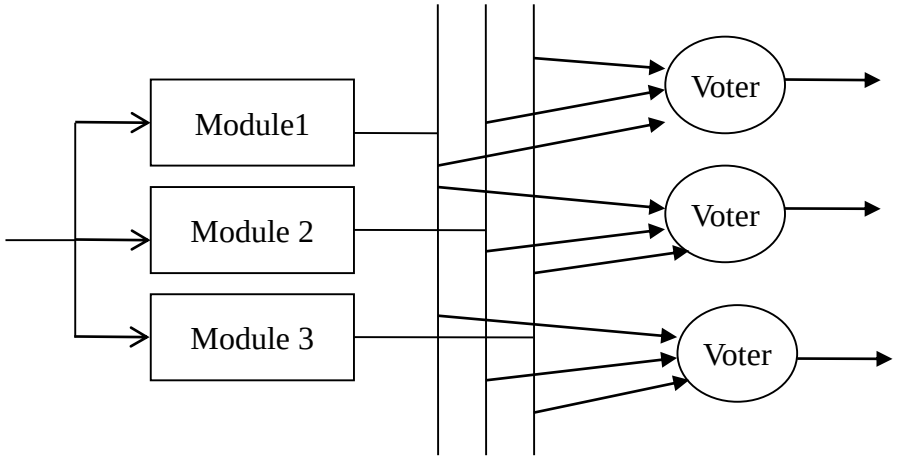
- due to the voter
- due to multiple copies synchronisation

Trade-off : achieved fault tolerance vs hw required

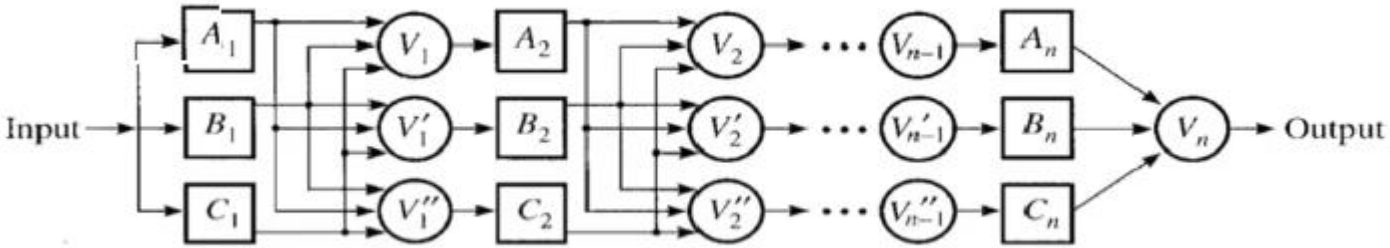
Voter: if the Voter fails, the complete system fails

→ Voter is a single point of failure

Triplicated Voters in a TMR configuration



Cascading TMR with triplicated voters



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

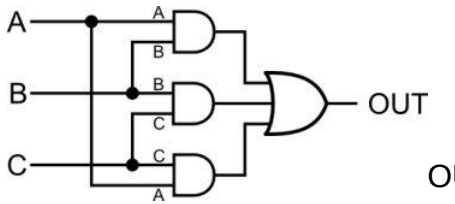
Triplicated voters: voter errors propagates only of one step

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

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

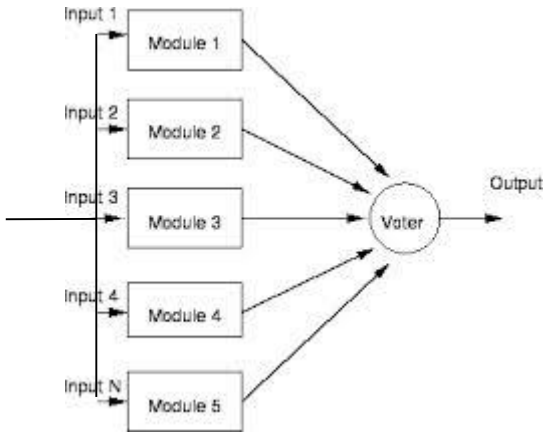
- n is made an odd number
- 5MR tolerates 2 faulty modules

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

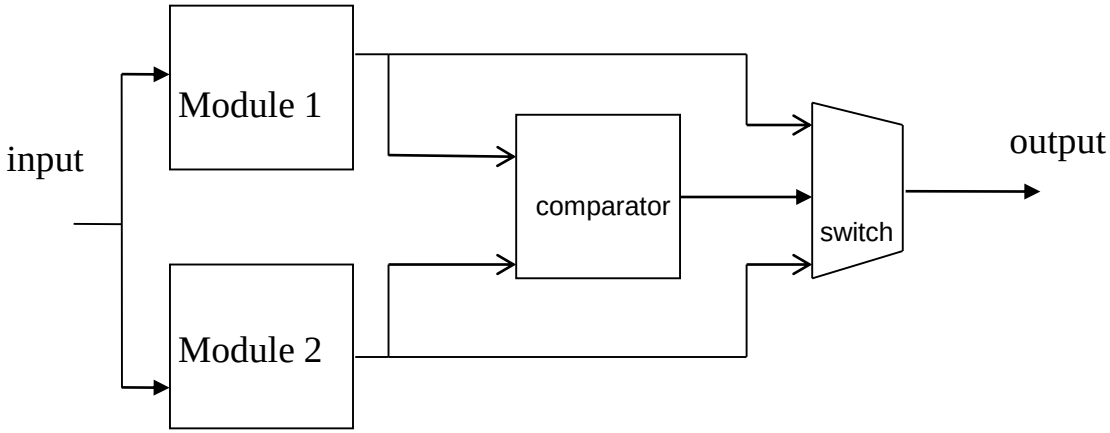


Active hw redundancy

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

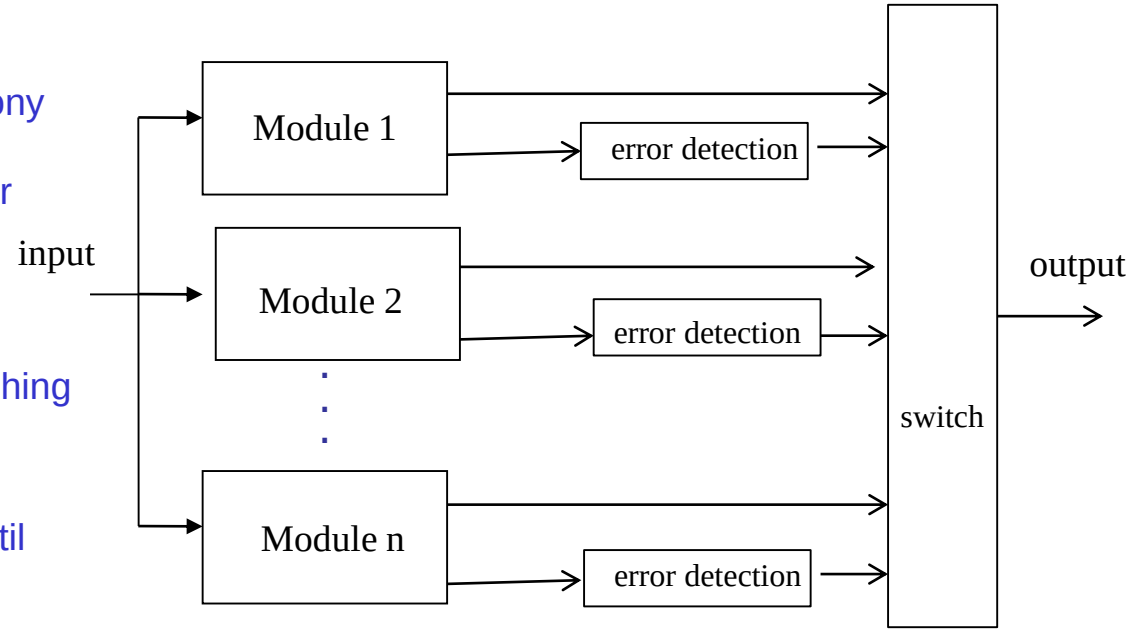
- Part of the modules are operational, part of the modules are spares modules (used as replacement modules)

- 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



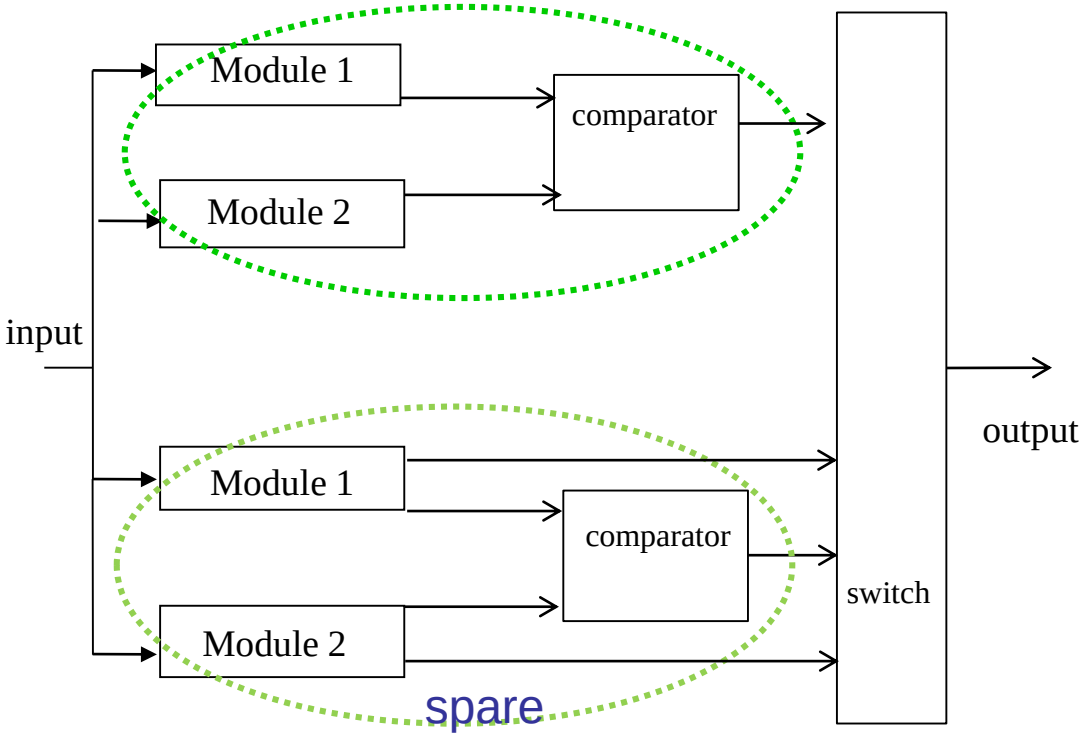
Reconfiguration process can be viewed as a switch that accepts the module's outputs and *error reports*
As long as the outputs agree, the spares are not used. When a miscompare occurs, the switch uses the error reports from the modules to identify the faulty module and then select a replacement module.

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

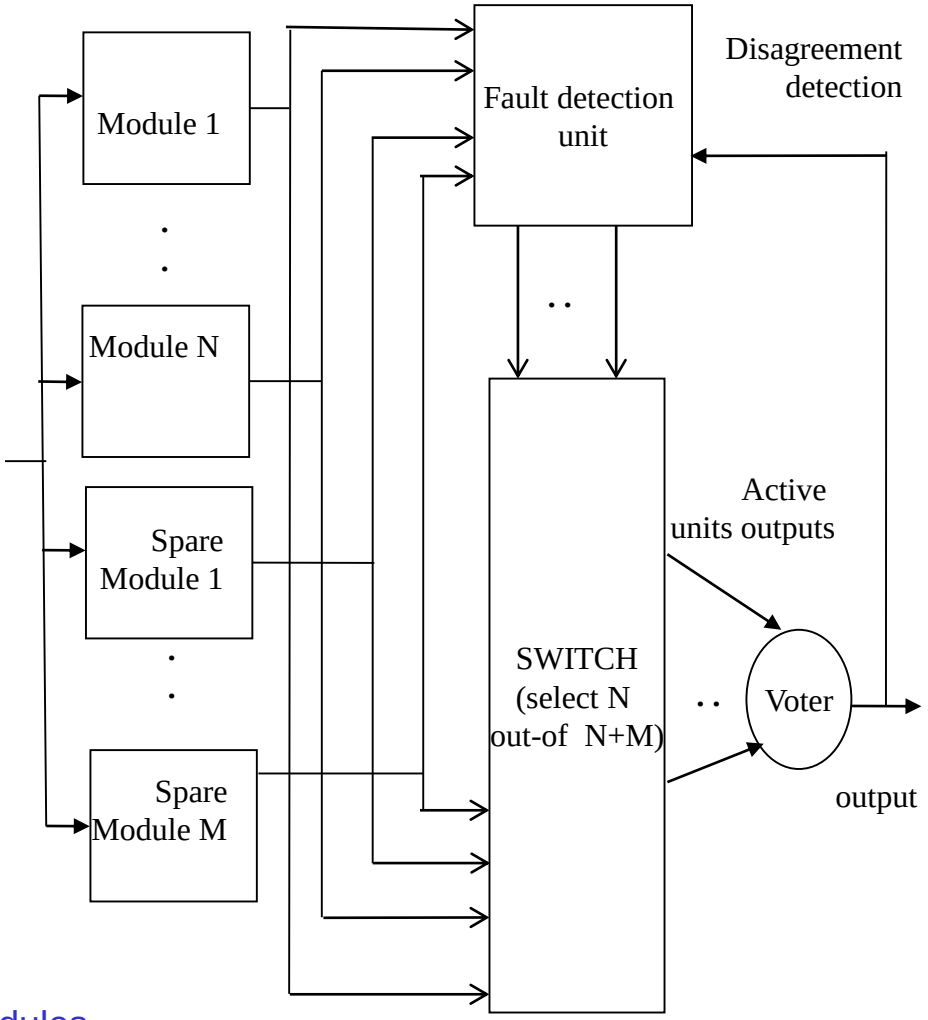
- spares to replace faulty units
- rely on detection of disagreements and determine the module(s) not agreeing with the majority

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

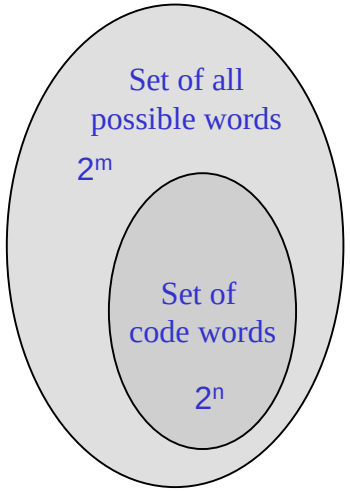
Coding

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

n+c= m bits

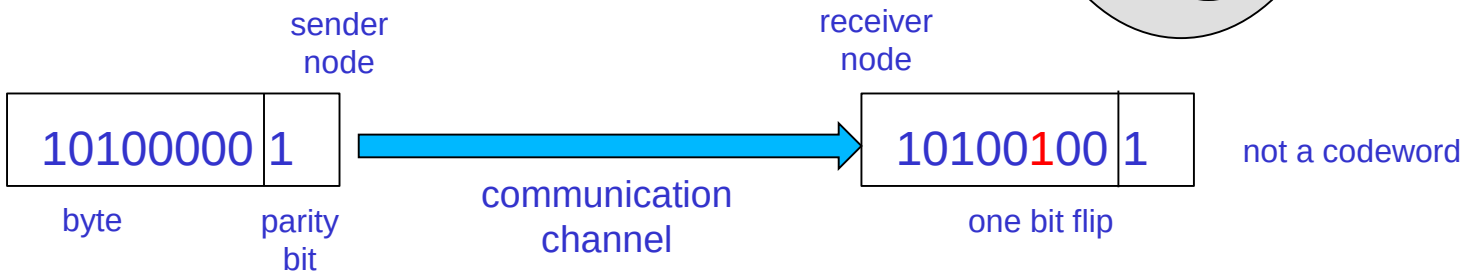
Among all the possible 2^m configurations of the m bits, only 2^n represent acceptable values (code words)

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



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

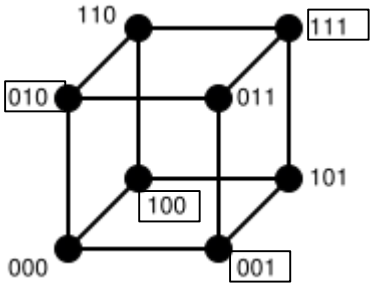
Parity code is a separable code

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

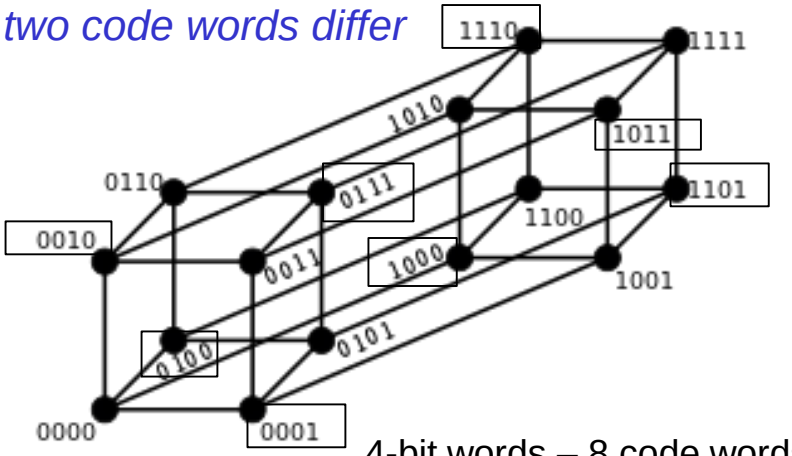
Memories of computer systems. Parity bit added before writing the memory. Parity bit is checked when reading.

Hamming distance

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

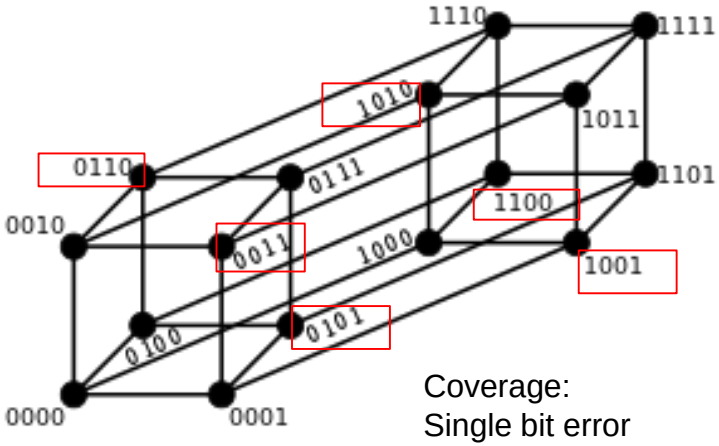
A code such that the minimum Hamming distance is k will correct up to d errors, where k = 2d +1

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

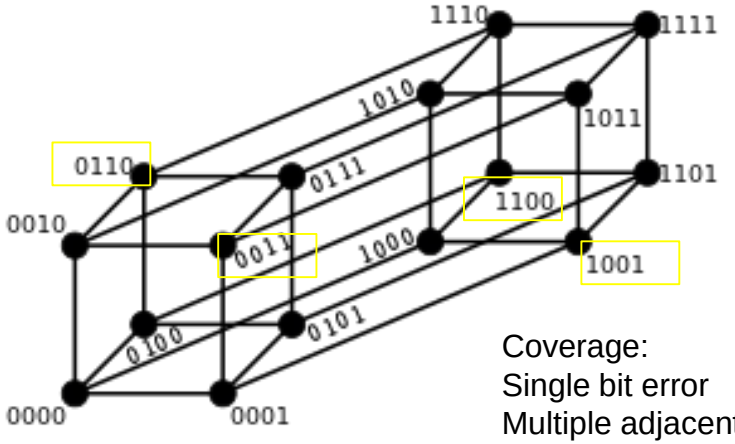


4-bit words – 6 code words

Coverage:
Single bit error
Multiple adjacent unidirectional bit errors
33% double bit errors

Complemented duplication codes (CD)

Hamming distance: 2

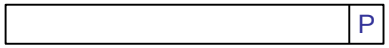


4-bit words – 4 code words

Coverage:
Single bit error
Multiple adjacent unidirectional bit errors
66% double bit errors

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), 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

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

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)

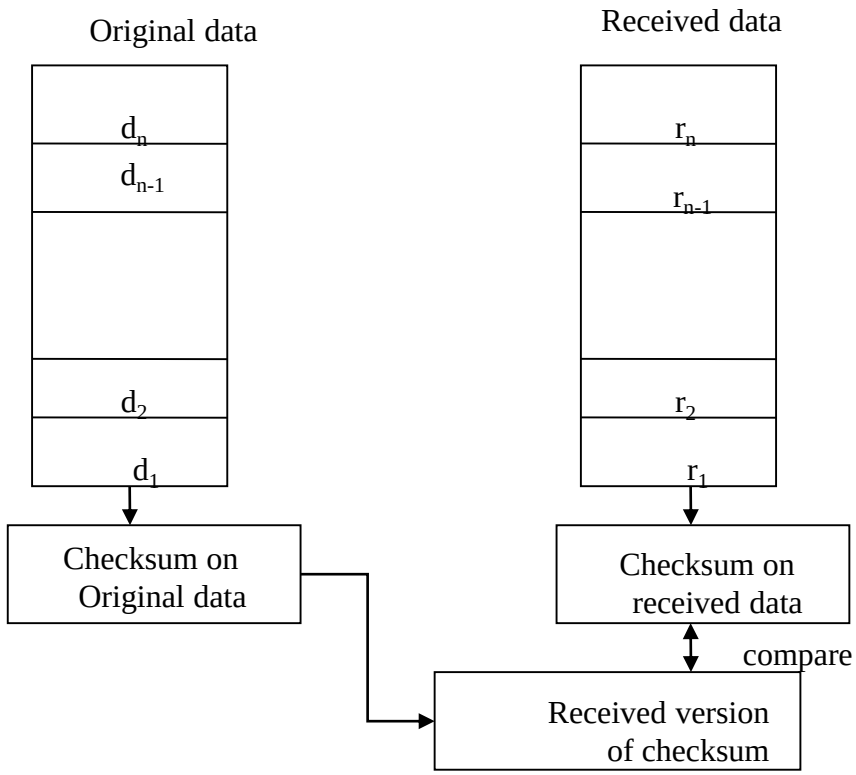
Code word = block + checksum

- 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

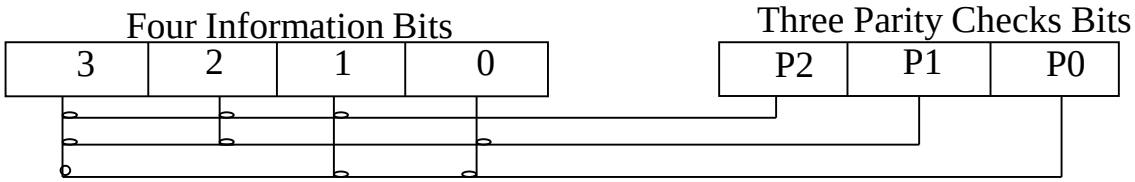
Disadvantages

- 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?



Bit Error	Parity group affected			
3	P2	P1	P0	(7)
2	P2	P1		(6)
1	P2		P0	(5)
0		P1	P0	(3)
P2	P2			(4)
P1		P1		(2)
P0			P0	(1)
no error				(0)

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

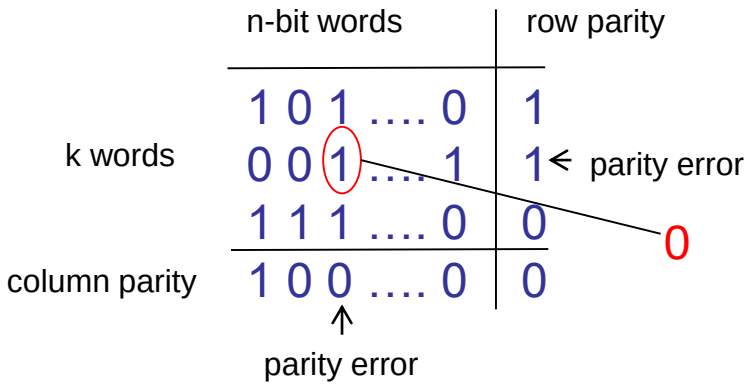
$2^K > m+k$

disadvantage: 75% of redundancy

ECC – Error Correcting Codes

Two-dimensional parity

Odd parity



Error location is possible for single-bit error:
one error in the row parity vector, one error in the column parity vector

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

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

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

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

Parity bit 8 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 (non-code output)

Self checking circuitry

Let be given a set of faults.

Self-testing circuit: if, for every fault from the set, the circuit produces a non-code output for at least one correct code input (each single fault is detectable)

Fault-secure circuit: if, for every fault from the set, the circuit never produces a not correct code output for a code input (i.e. correct code output or non-code 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

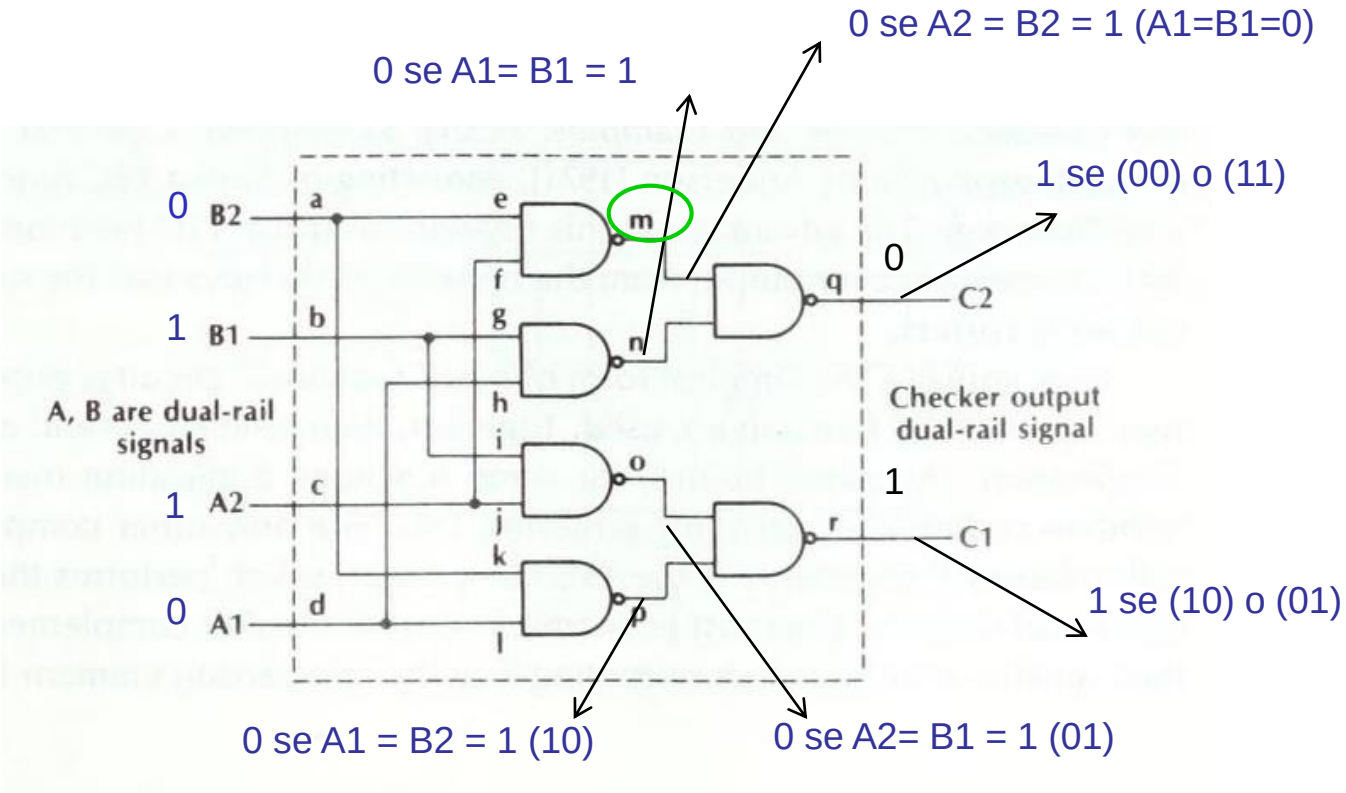
input and output coding: 1/2 code
(dual-rail signal: coded signal whose two bits are always complementary)

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



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

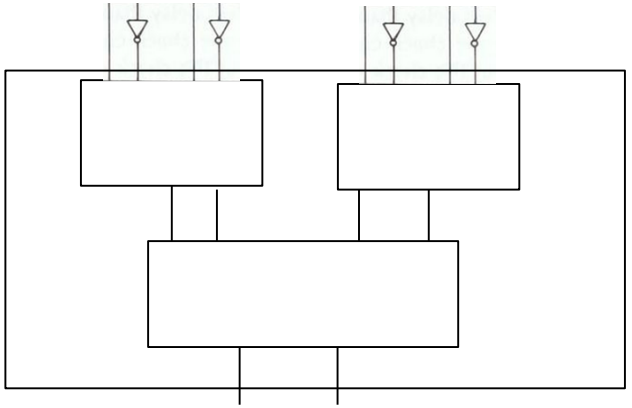
Faulty:
A=0, B=1
m: stuck-at-1
c2=0
c1=1
code
different input

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:
tree of two input
self checking comparators



TIME REDUNDANCY

Time redundancy techniques

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

1. Repetition of computations

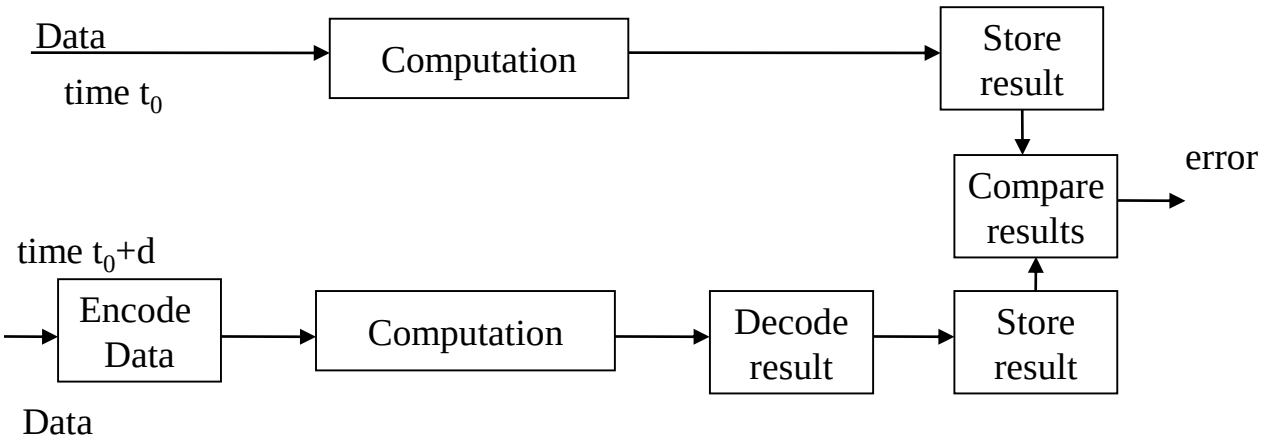
- compare the results to detect faults
- re-execute computations (disagreement disappears or remains)

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

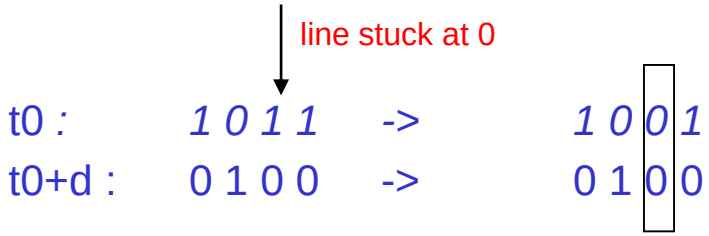
Example

- 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



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

Multi-version approaches

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

Single-version approaches

one code with error detection and fault tolerant capabilities inside

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**.

Two-version systems N = 2

Upon disagreement among the versions?

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

N-version programming

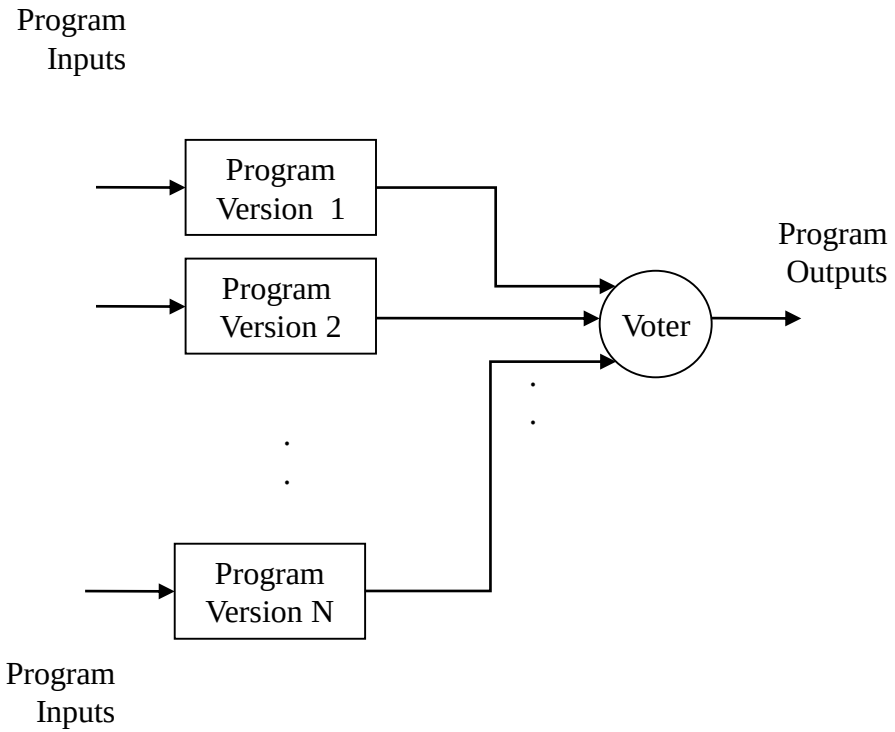
N-self-checking programming

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

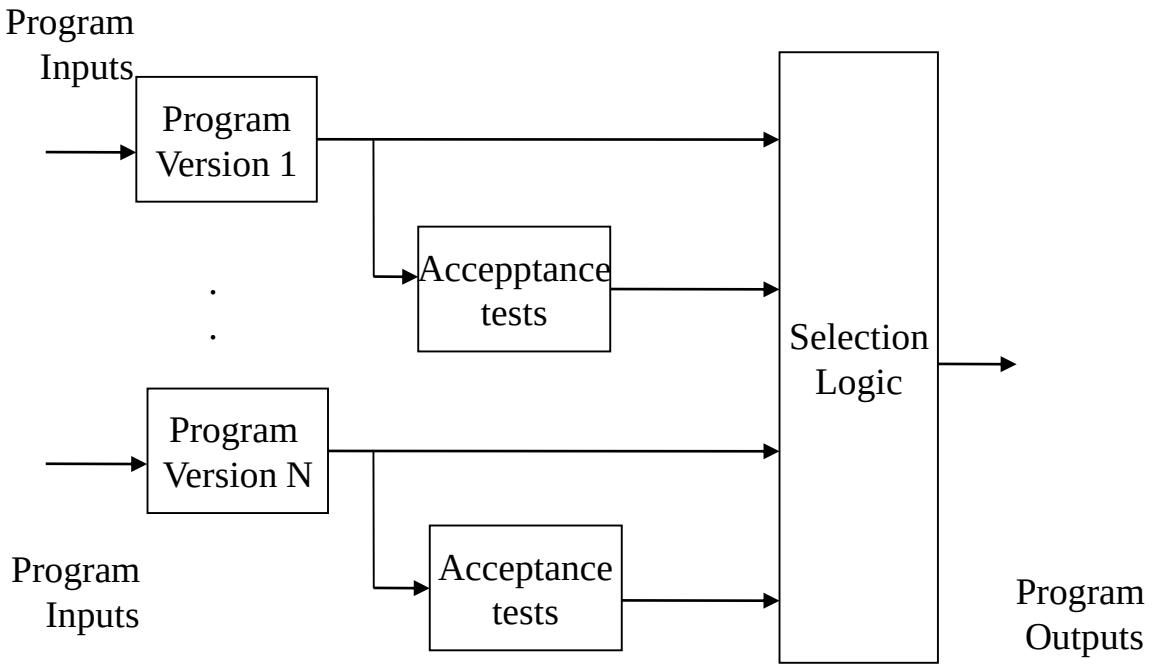
- must assure that the input data vector to each of the versions is identical

- 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.