Fault-Tolerant Computing

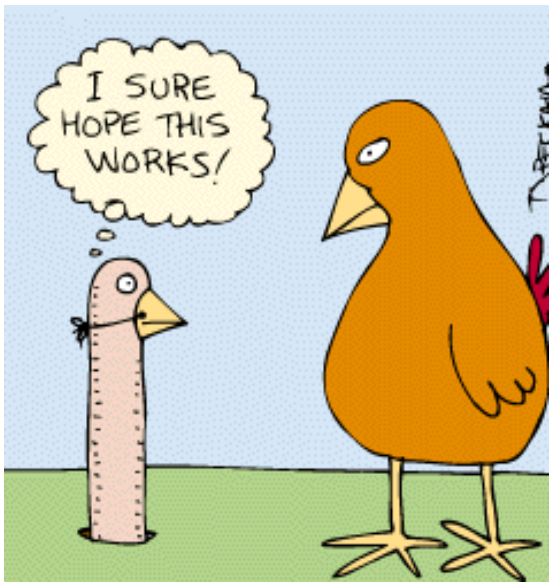
About This Presentation

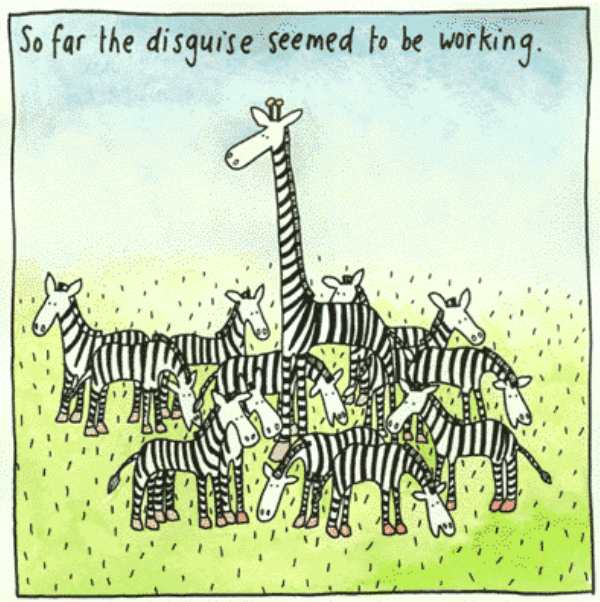
This presentation has been prepared for the graduate course ECE 257A (Fault-Tolerant Computing) by Behrooz Parhami, Professor of Electrical and Computer Engineering at University of California, Santa Barbara. The material contained herein can be used freely in classroom teaching or any other educational setting. Unauthorized uses are prohibited. © Behrooz Parhami

Edition	Released	Revised	Revised
First	Oct. 2006		

Fault Masking

ZERO TOLERANCE ON INTOLERANCE...





Multilevel Model

Component

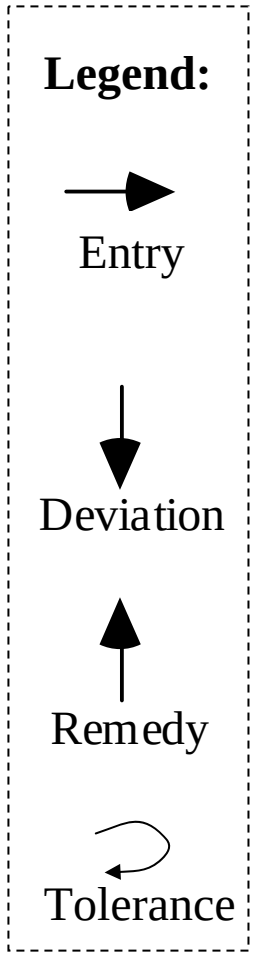
Logic

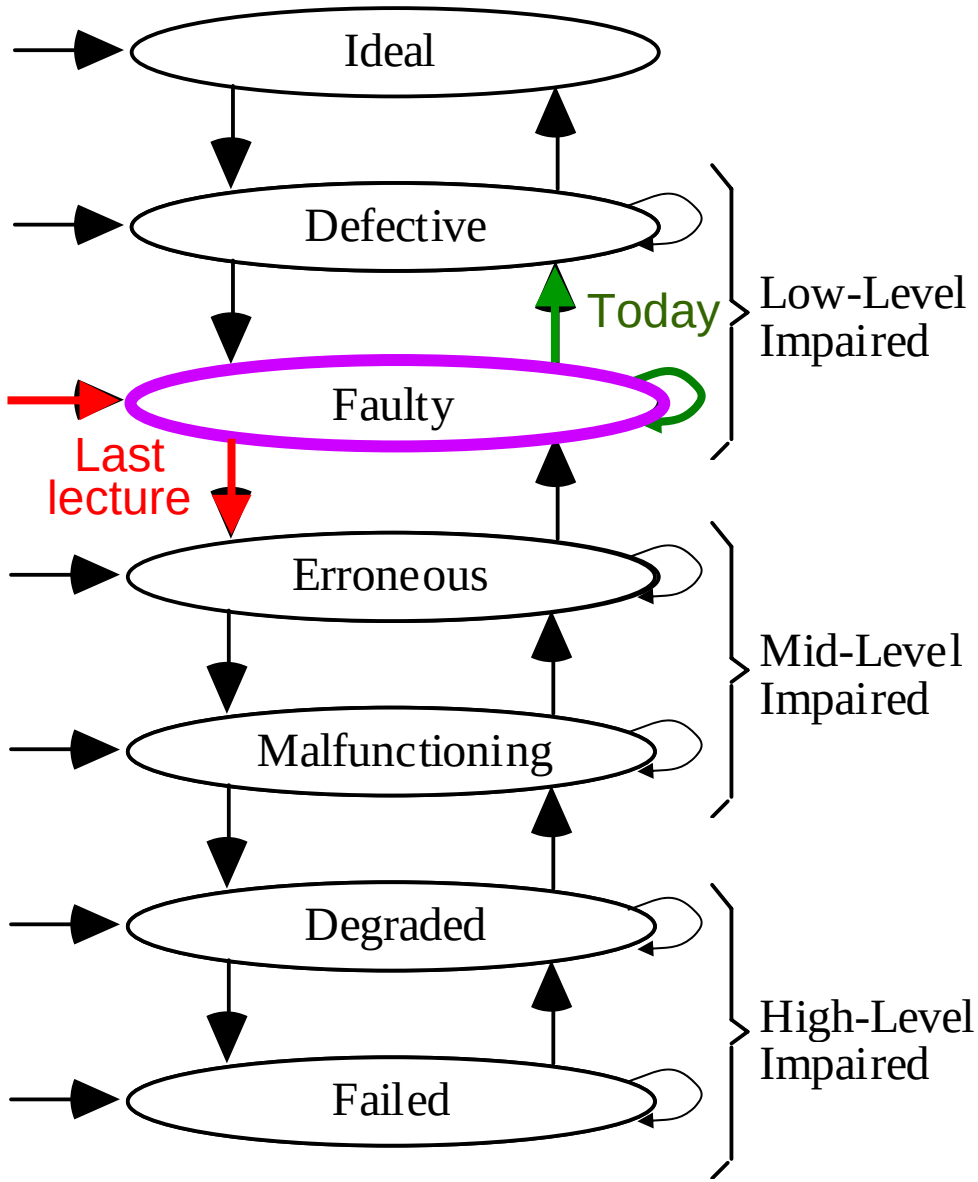
Information

System

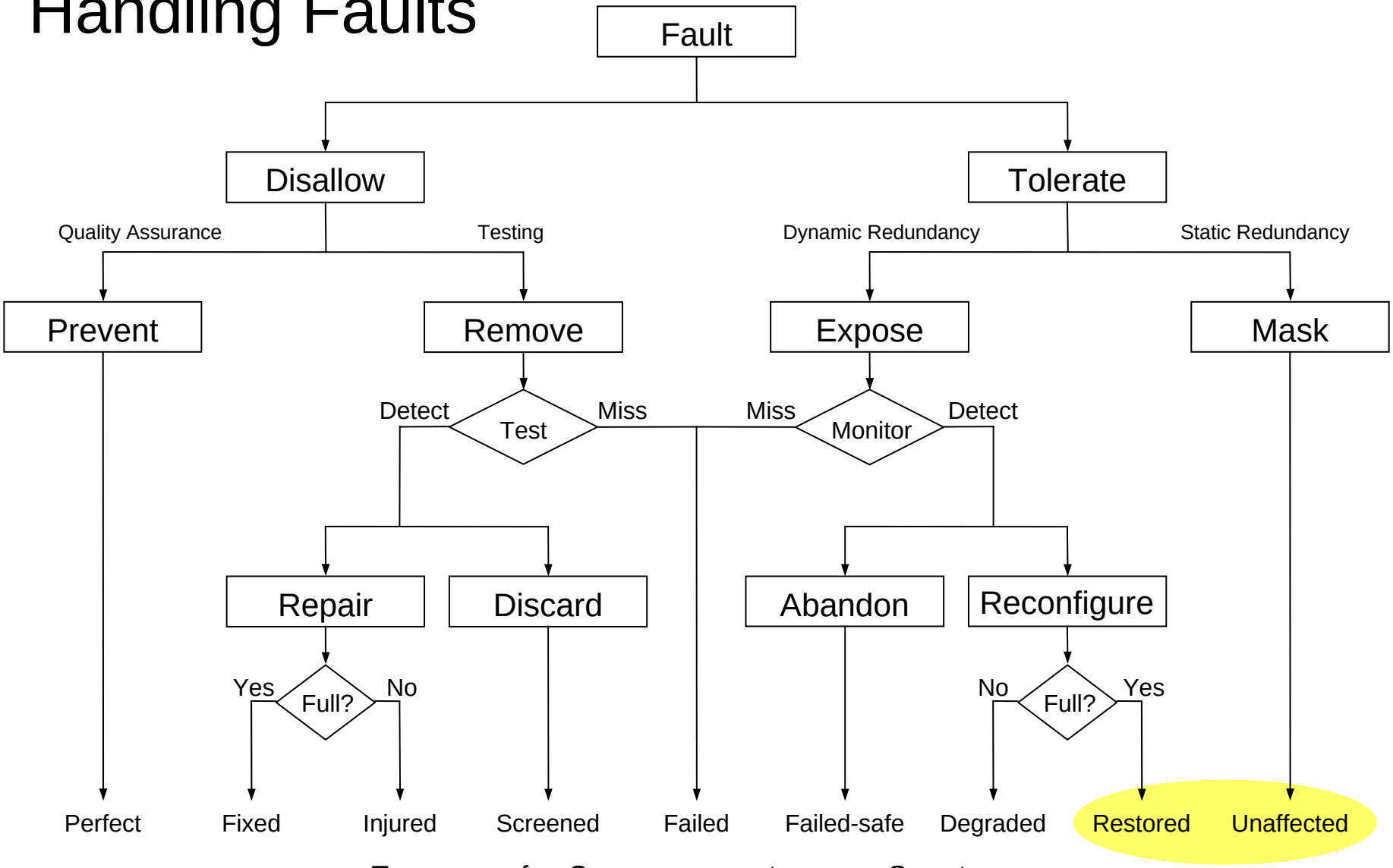
Service

Result





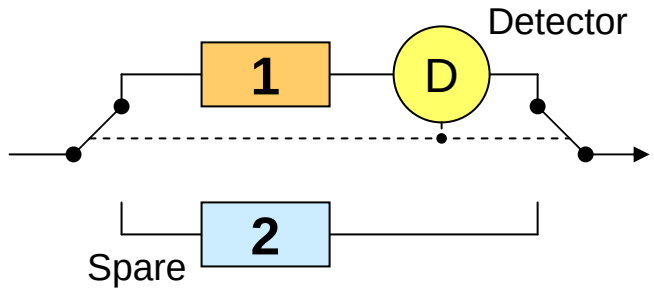
Handling Faults



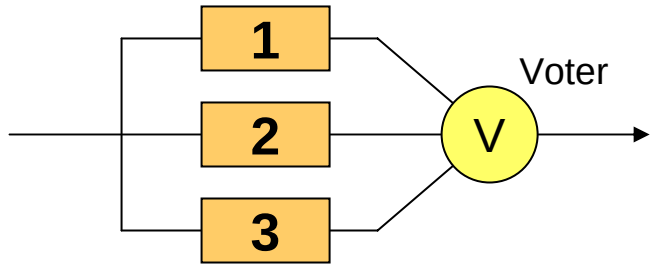
Type of Component or System

Some Options for Fault Tolerance

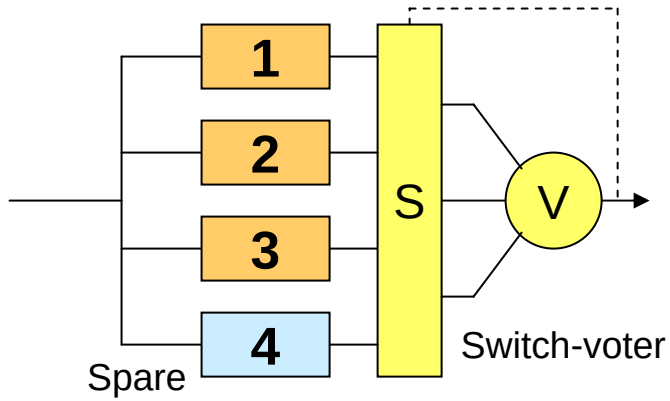
1. **Detect and replace**
   Dynamic redundancy (cold/hot standby)
   Detection via
   -- coding, watchdog timer, self-checking
   -- duplication (pair-and-spares)



2. **Mask**
   Static redundancy
   May revert to simplex instead of duplex
   Design challenges include
   -- synchronization for voting
   -- voting on imprecise results

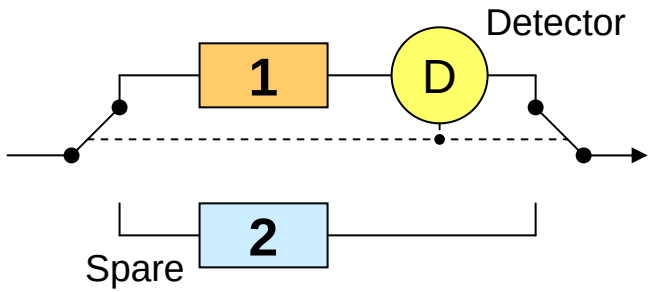


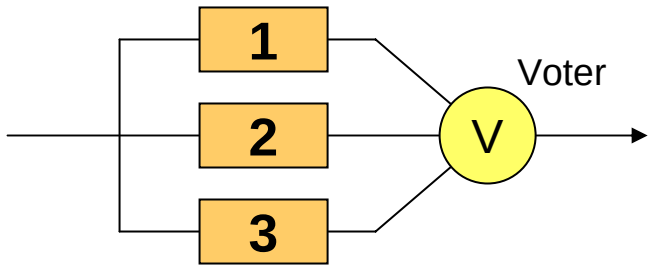
3. **Mask, diagnose, and reconfigure**
   Hybrid redundancy
   Fault masked at output, but diagnosed
   -- e.g., via comparison with voter output
   Faulty circuit is replaced by spare
   Becomes static upon spare exhaustion

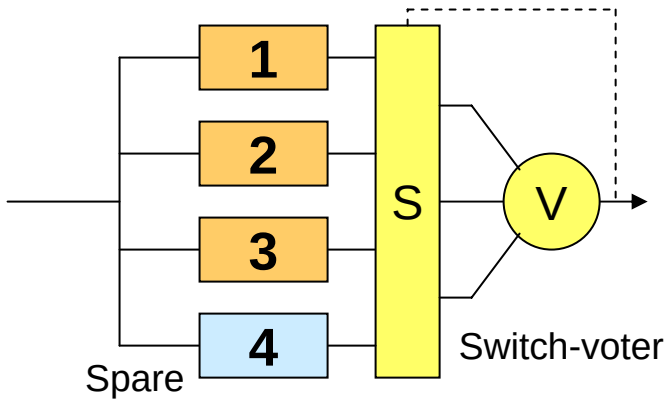


Comparing Fault Tolerance Schemes

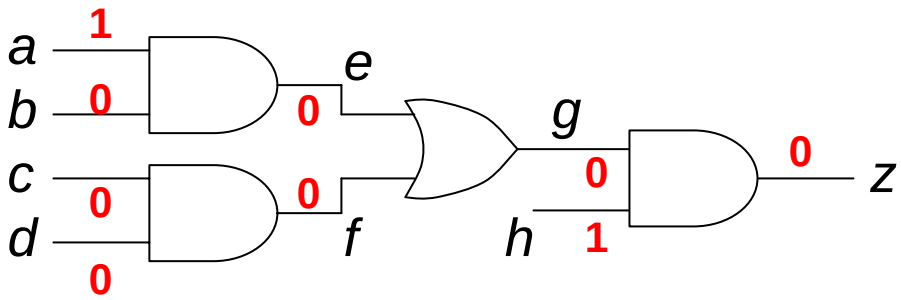
Advantages	**Drawbacks**
Less power (cold standby)	Coverage factor
Long life (just add spares)	Tolerance latency
Immediate masking	Power/area penalty
High safety	Voter critical
Immediate masking	Power/area penalty
Long life and high safety	Switch-voter critical







Inherent Fault Masking in Logic Circuits



0 → 1 fault in *b* is critical

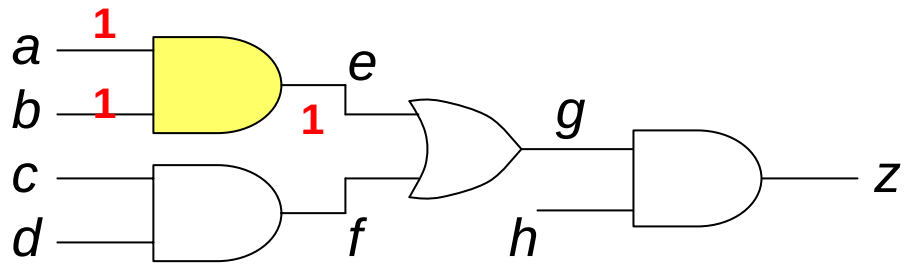
0 → 1 fault in *c* or *d* is not critical (it is masked)

1 → 0 fault in *a* or *h* is not critical (it is masked)

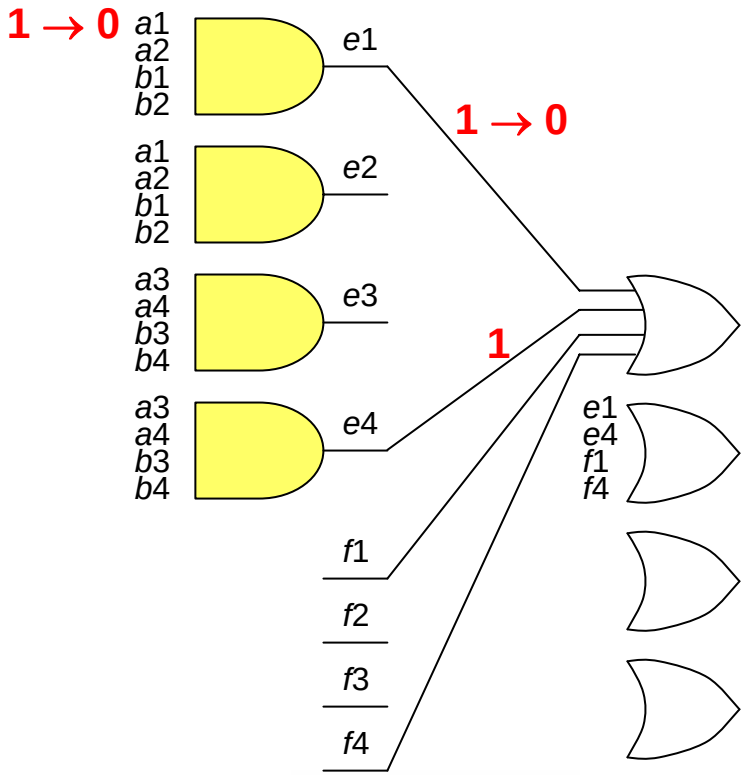
Even nonredundant circuits have some masking capability

Is there a way to exploit the inherent masking capabilities of logic gates to achieve fault tolerance?

Interwoven Redundant Logic



Let $x1$, $x2$, $x3$, and $x4$ be 4 copies of the signal x



1 → 0 change is critical for AND, subcritical for OR

0 → 1 change is critical for OR, subcritical for AND

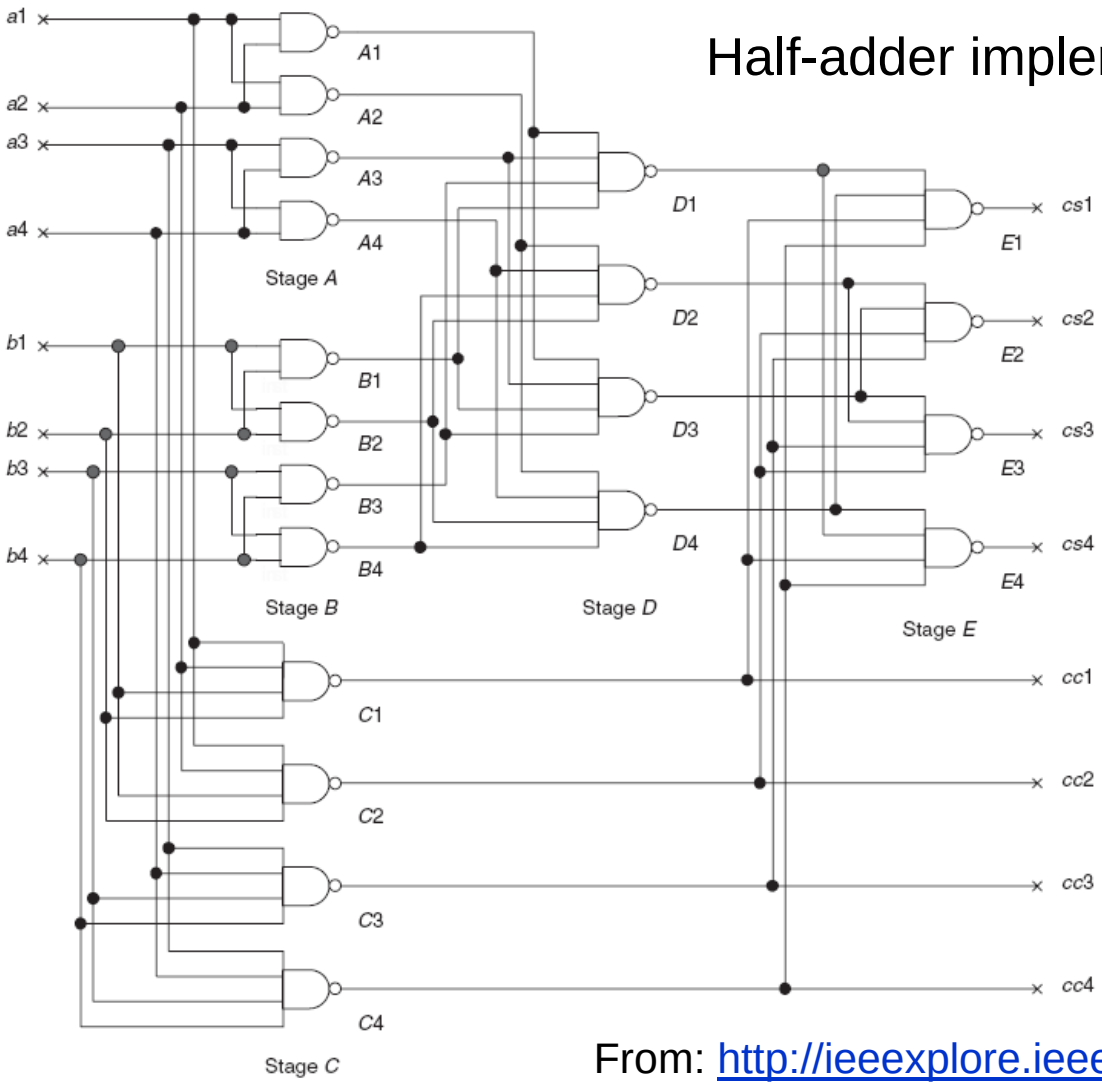
Alternating layers of ANDs and ORs can mask each other's critical faults

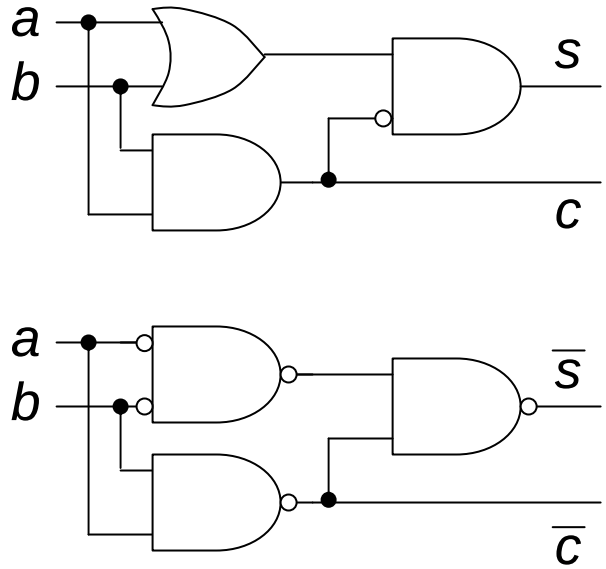
To mask h critical faults:
Number of gates multiplied by $(h + 1)^2$
Gate inputs multiplied by $h + 1$

For $h = 1$, the scheme is known as Quadded logic

Interwoven Logic for Nanoelectronics

Half-adder implemented in quadded logic





IEEE D&T
July-Aug. 2005
pp. 328-339

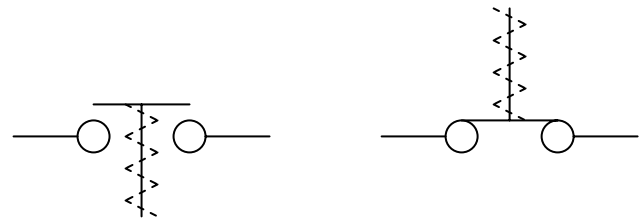
From: http://ieeexplore.ieee.org/iel5/54/32070/01492293.pdf

Highly Reliable Logic with “Crummy” Relays

Moore & Shannon, 1956

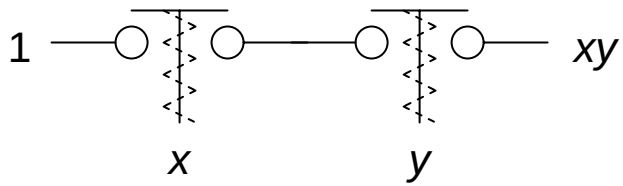
a: prob [contact made | energized]
c: prob [contact made | not energized]

No matter how crummy the relays (i.e., how close the values of *a* and *c*), one can interconnect many of them in a redundant series-parallel structure to achieve arbitrarily high reliability



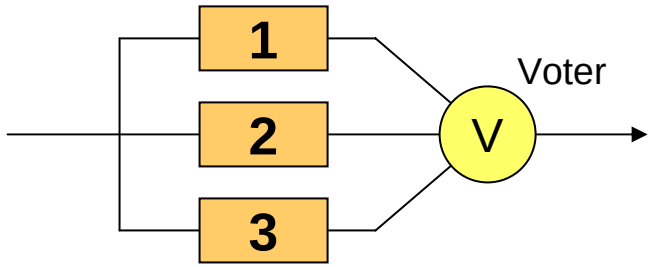
“Make” contact (normally open)
a > c

“Break” contact (normally closed)
a < c



TMR with Perfect Voter

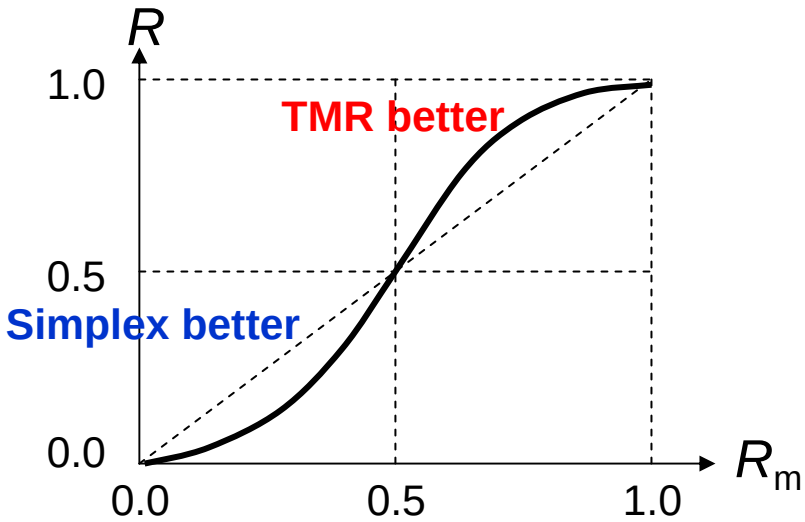
$R = 3R_m^2 - 2R_m^3 \stackrel{?}{>} R_m$

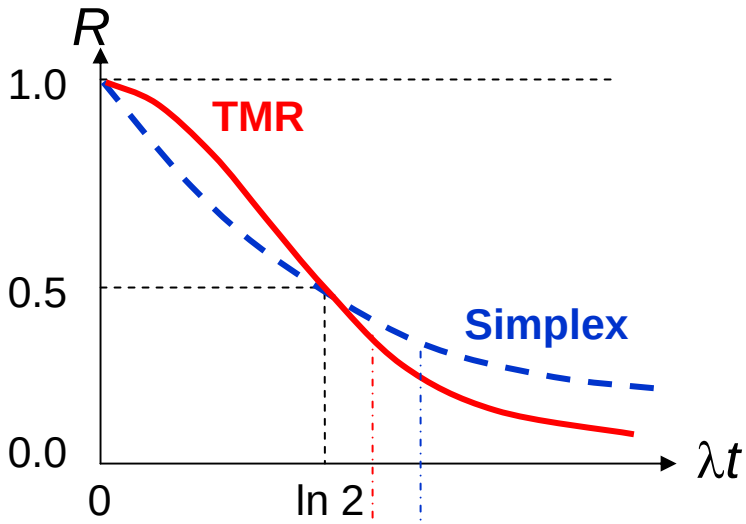


Condition on the module reliability:

$R = R_m [1 + (1 - R_m)(2R_m - 1)]$

$(1 - R_m)(2R_m - 1) > 0 \Rightarrow R_m > 1/2$





$RIF_{TMR/Simplex} = (1 - R_m)/(1 - R)$

$= 1/[1 - R_m(2R_m - 1)]$

MTTF: TMR $5\lambda/6$

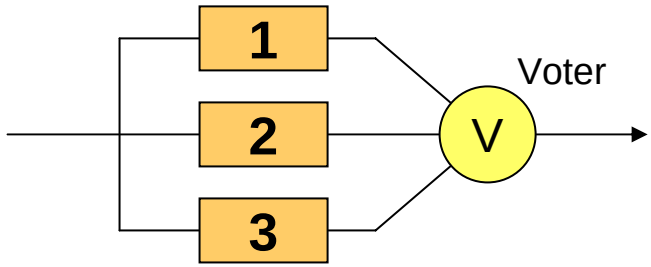
Simplex λ

TMR with Imperfect Voter

$R = R_v(3R_m^2 - 2R_m^3) \stackrel{?}{>} R_m$

Condition on the voter reliability

$R_v > 1 / [3R_m - 2R_m^2]$



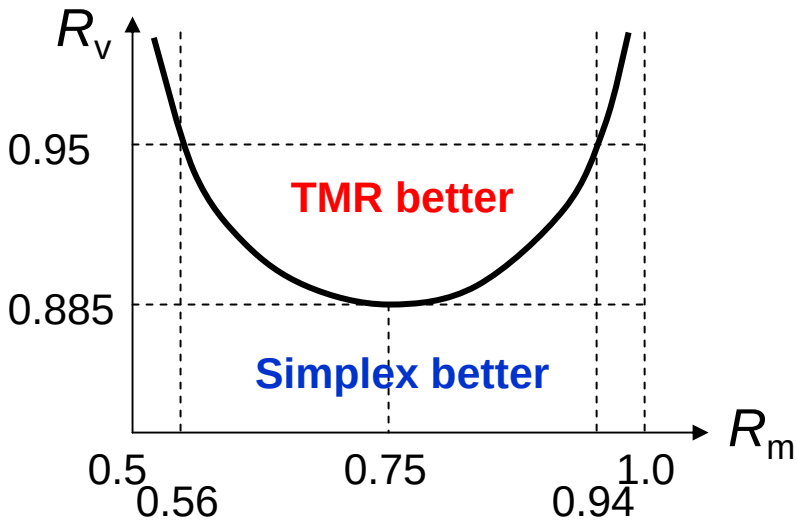
$dR_v^{min} / dR_m = (-3 + 4R_m) / (3R_m - 2R_m^2)^2$

Condition on the module reliability

$$\frac{3 - \sqrt{9 - 8/R_v}}{4} < R_m < \frac{3 + \sqrt{9 - 8/R_v}}{4}$$

Example: $R_v = 0.95$ requires that

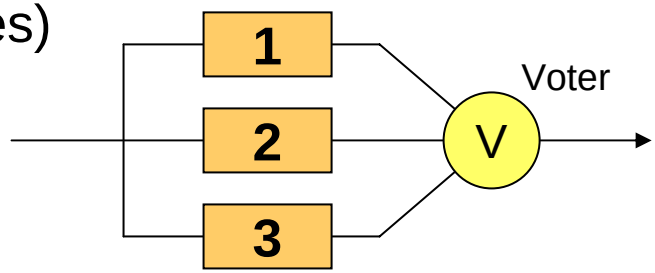
$0.56 < R_m < 0.94$



TMR with Compensating Faults

$R_m = 1 - p_0 - p_1$ (0- and 1-fault probabilities)

$R = (3R_m^2 - 2R_m^3) + 6p_0p_1R_m$



Example: $R_m = 0.998$, $p_0 = p_1 = 0.001$

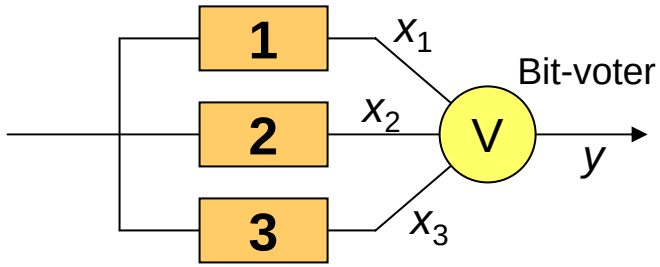
$R = \underset{\text{Basic TMR}}{\underline{0.999{,}984}} + \underset{\text{Compensation}}{\underline{0.000{,}006}} = 0.999{,}990$

$\text{RIF}_{\text{TMR/Simplex}} = 0.002 / 0.000{,}016 = 125$

$\text{RIF}_{\text{Compen/TMR}} = 0.000{,}016 / 0.000{,}010 = 1.6$

Implementing a Bit-Voter

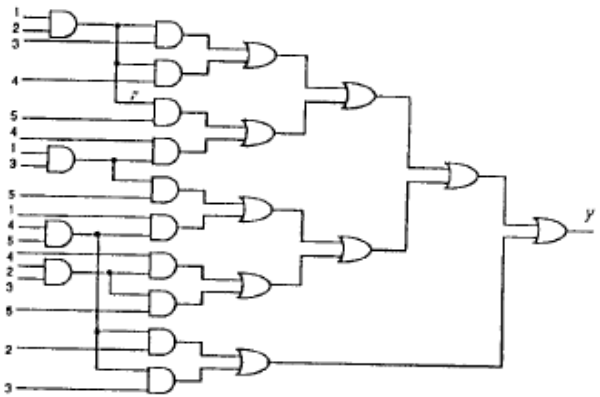
TMR bit-voting: $y = x_1x_2 \vee x_2x_3 \vee x_3x_1$
(carry output of a single-bit full-adder)
What about 5MR, 7MR?



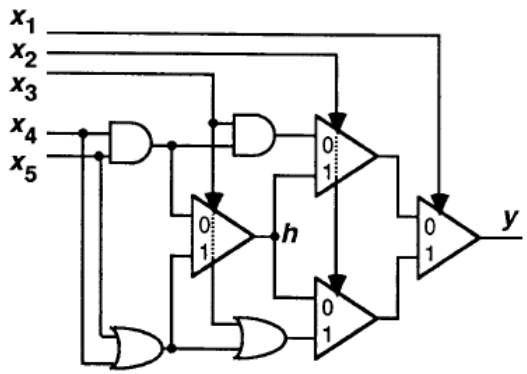
Gate-level design quickly explodes in size

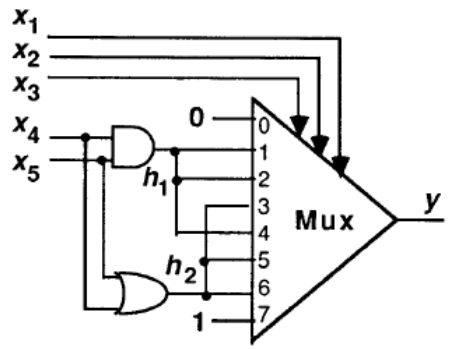
Other designs are also possible
- Arithmetic: add the bits, compare to threshold
- Mux-based
- Selection-based (majority of bit values is their median)

3-out-of-5 voter built of 2-input gates

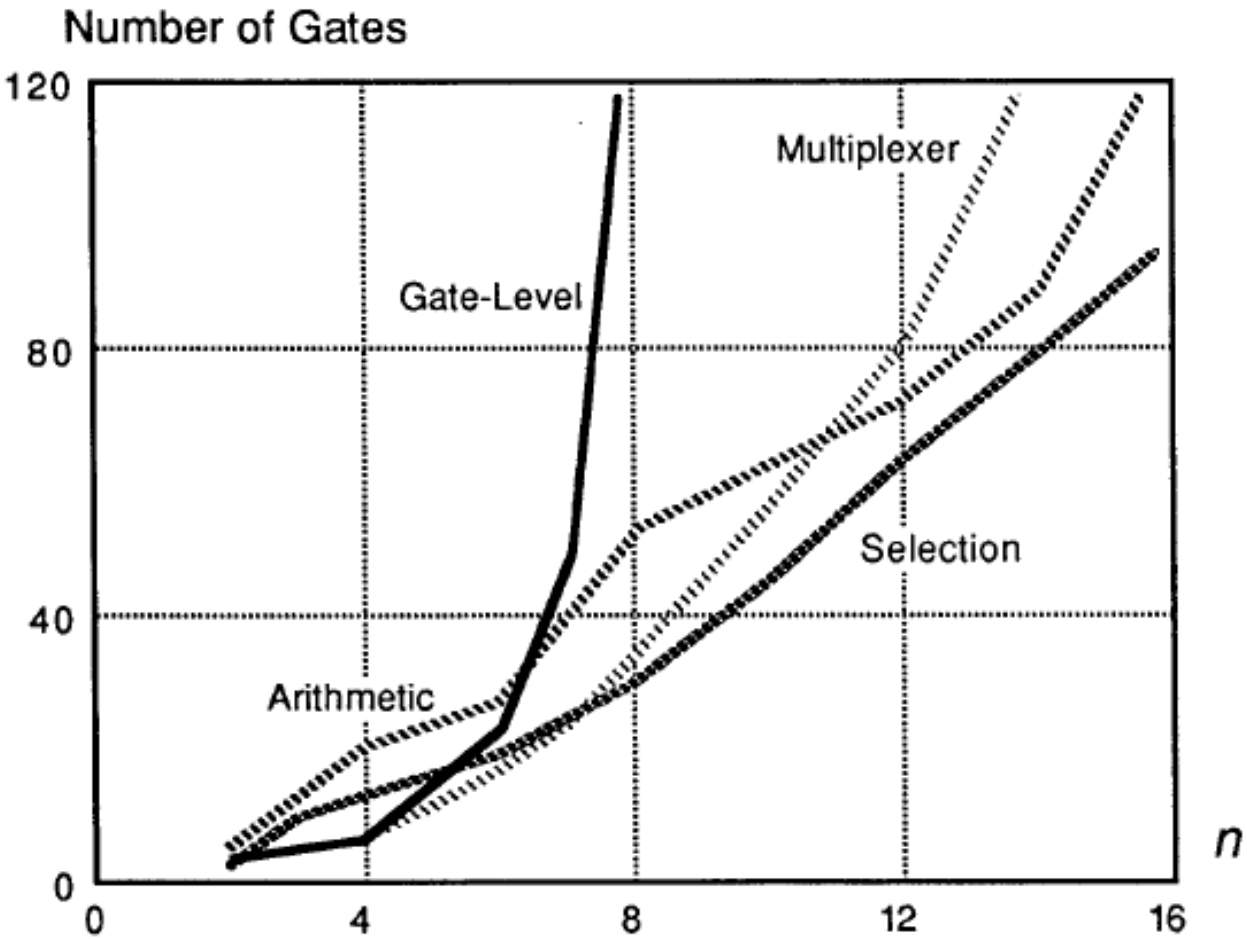


Two mux-based designs for a 3-out-of-5 bit-voter





Complexity of Different Bit-Voter Designs



Cost of majority bit-voters as a function of the number *n* of inputs

Voting at the Word Level

Using bit-by-bit voting may be dangerous

One might think that in this example, any of the module outputs could be correct, so that producing 1 0 at the output isn't all that wrong

However, with bit-by-bit voting, the output may be different from all inputs

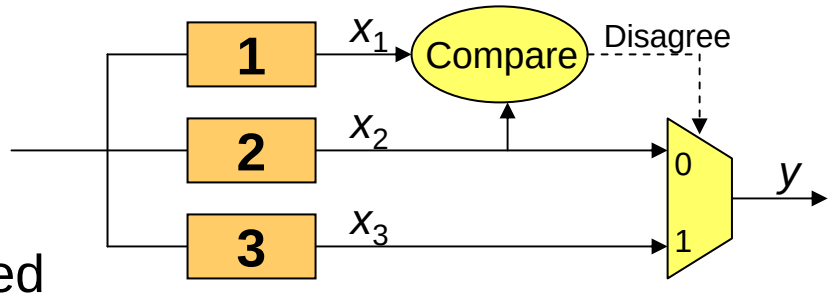
$$\begin{array}{l} x_1 = 0\ \ 0 \\ x_2 = 1\ \ 0 \\ \underline{x_3 = 1\ \ 1} \\ y\ \ = 1\ \ 0 \end{array}$$

$$\begin{array}{l} x_1 = 0\ \ 0\ \ 0 \\ x_2 = 1\ \ 0\ \ 1 \\ \underline{x_3 = 1\ \ 1\ \ 0} \\ y\ \ = 1\ \ 0\ \ 0 \end{array}$$

Design of bit- and word-voting networks discussed in:
Parhami, B., "Voting Networks," *IEEE TR*, Aug. 1991

Some Simple Voter Designs

If in the case of 3-way disagreement any of the inputs can be chosen, then a simple design is possible



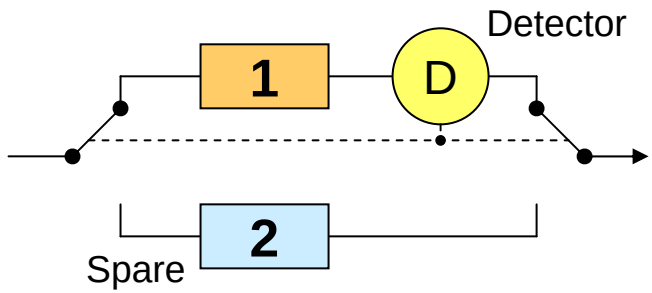
This design can be readily generalized to a larger number of inputs

One can perform pseudo voting that yields the median of 3 analog signals (Dennis, N.G., *Microelectronics and Reliability*, Aug. 1974)

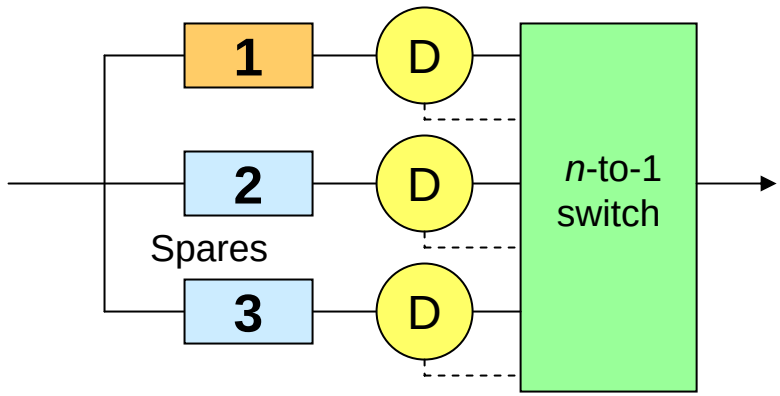
Median and mean voting are also possible with digital signals

Switch for Standby Redundancy

Standby redundancy requires an *n*-to-1 switch to select the output of the currently active module



The detectors use various info to deduce fault conditions
-- Error coding
-- Reasonableness checks
-- Watchdog timer

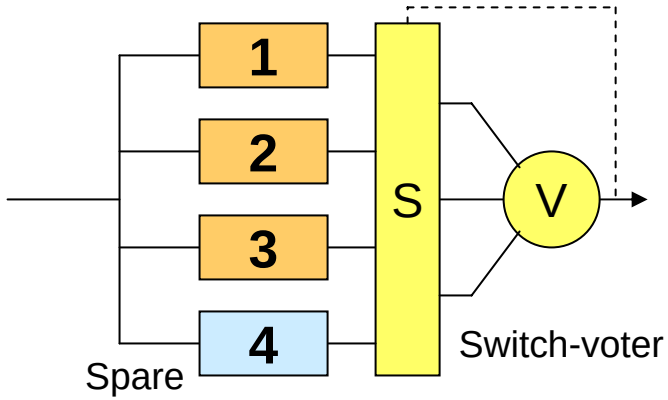


Once a fault has been detected, the switch reconfigures the system by flagging the faulty unit and activating next spare in sequence

If we use an *n*-to-2 switch and compare the two selected outputs, the configuration is known as “pair-and-spares”

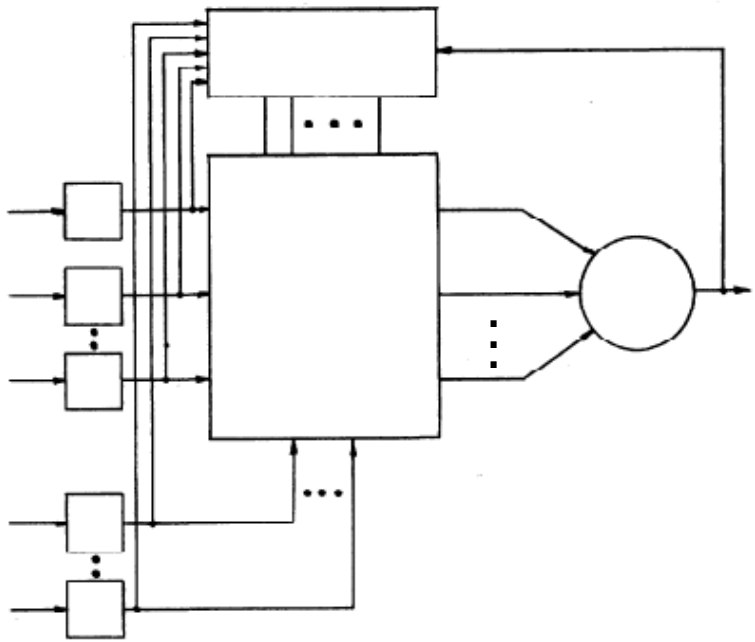
Switch for Hybrid Redundancy

Hybrid redundancy with *n* active and *s* spare modules requires an (*n* + *s*)-to-*n* switch to select the outputs of the active modules



Self-purging redundancy is a variant of hybrid redundancy in which all modules are active at the outset, but they are purged as they disagree with the majority output

Voter in self-purging redundancy is a threshold voter that considers the inputs with weights of 1 (active) or 0 (purged)



Applications of *n*MR and Hybrid Redundancy

The Space Shuttle:

Uses 5-way redundancy in hardware

- Originally, 3 operational units and 2 spares (one warm, one cold)
- More recently, 4 operational units and 1 spare

Additionally, uses two independently developed software systems

Japanese Shinkansen "Bullet" Train

Triple-duplex system (6-fold redundancy)

⋮