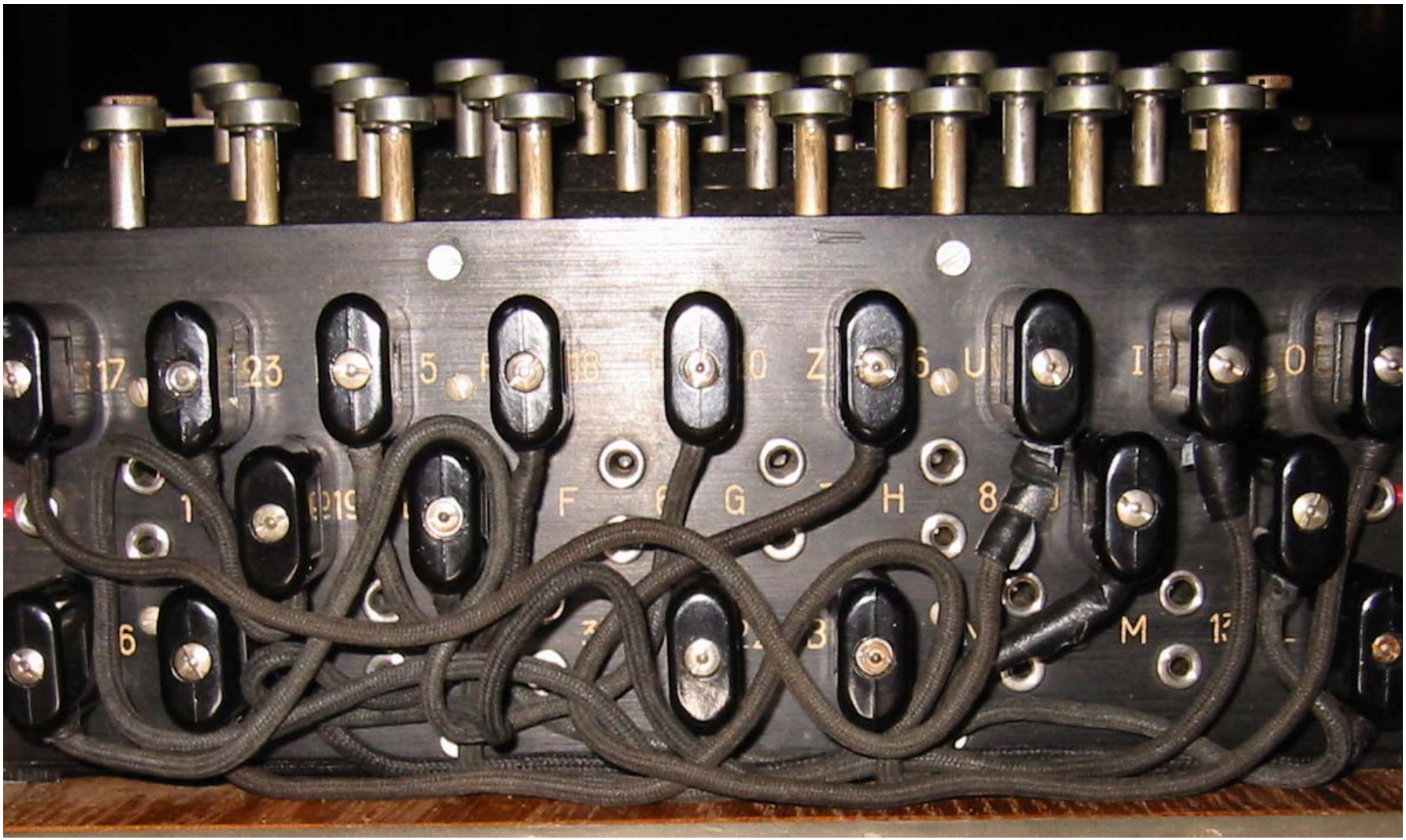
Fault-Tolerant Computing

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	Nov. 2006		

Reconfiguration and Voting



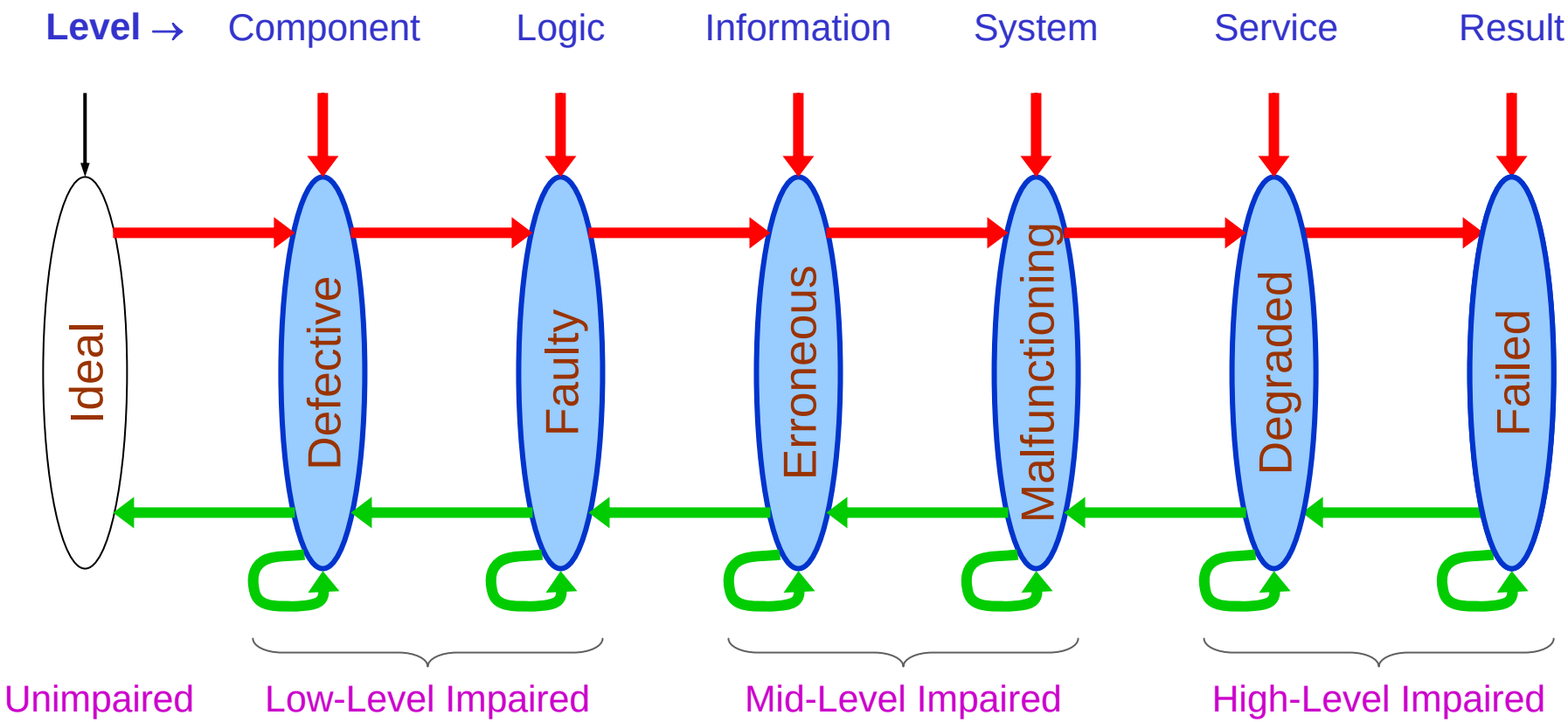
".....and who's been messing with my desktop configuration?"

Wow! It works! My personalized configuration is up and running!

"AFTER MONTHS OF SPEECHES, PROMISES, AND ACCUSATIONS, I'VE DECIDED TO JUST VOTE FOR THE GUY WITH THE COOLEST WEB SITE."

"Let's try voting for the greater of the two evils this time and see what happens."

Multilevel Model of Dependable Computing



Requirements for Reconfiguration

A system consists of modular resources (processors, memory banks, disk storage, . . .) and interconnects

Redundant resources can mitigate the effect of module failures

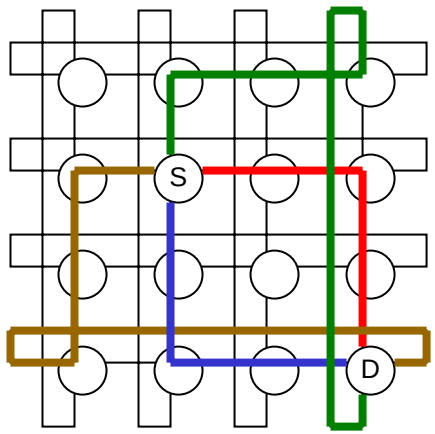
The main challenge of reconfiguration is dealing with interconnects

Assumption: Module and interconnect failures are promptly diagnosed

Overcoming the effect of interconnect failures requires the availability of multiple paths from each source to every possible destination

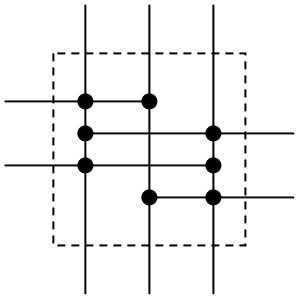
In graph-theoretic terms, we need “edge-disjoint” paths

Existence of *k* edge-disjoint paths between two nodes provides the means for tolerating *k* – 1 link failures



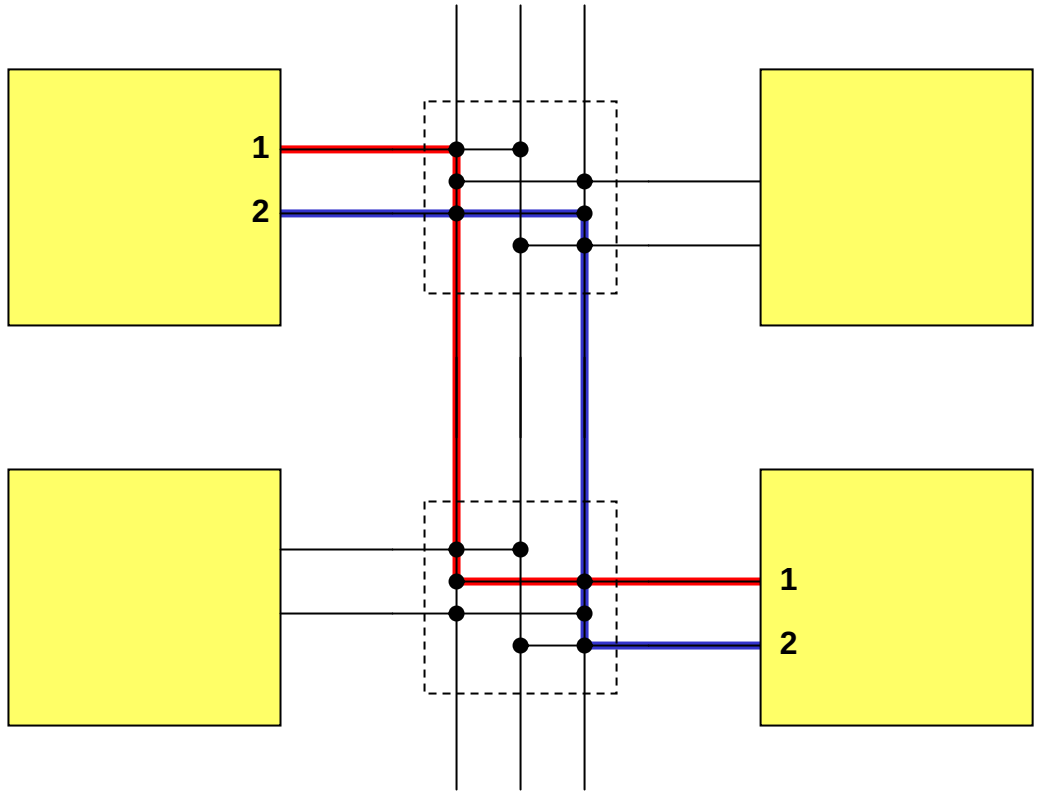
This particular interconnection scheme (torus) is 4-connected and tolerates 3 link/node losses without becoming disconnected

Reconfiguration via Programmable Connections



Interconnection switch with 4 ports (horizontal lines) and 3 channels (vertical lines)

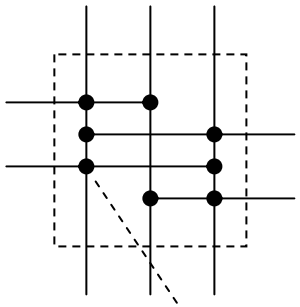
Each port can be connected to 2 of the 3 channels



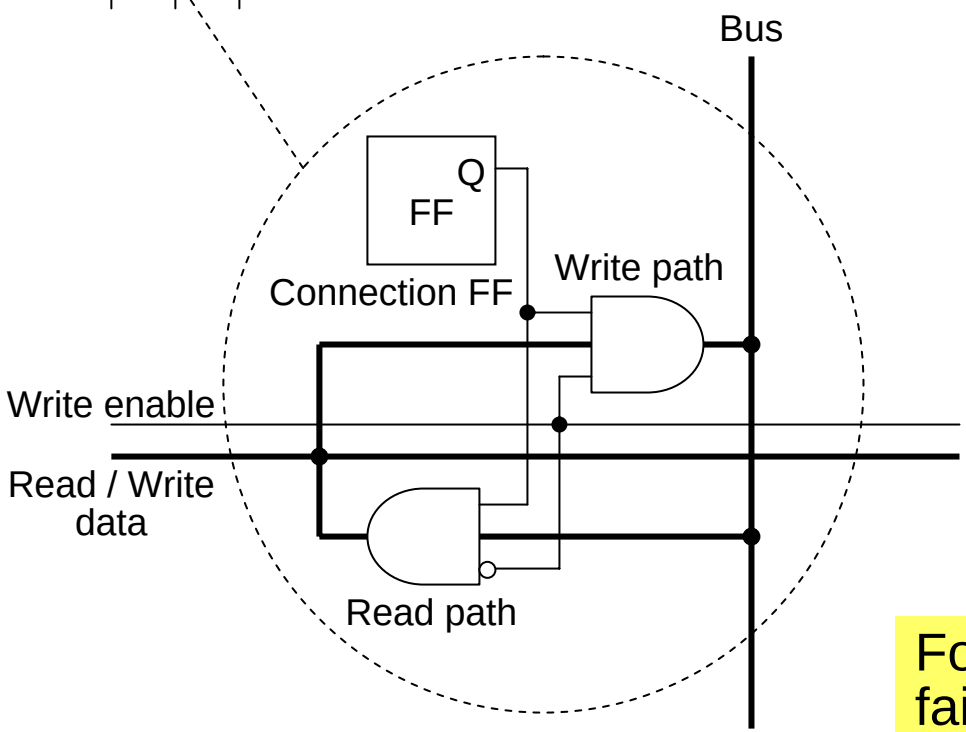
If each module port were connected to every channel, the maximum flexibility would result (leads to complex hardware & control, though)

The challenge lies in using more limited connections effectively

Multiple-Bus Multiprocessors



The vertical channels may be viewed as buses and the heavy dots as controllable bus connections, making this method applicable to fault-tolerant multiprocessors



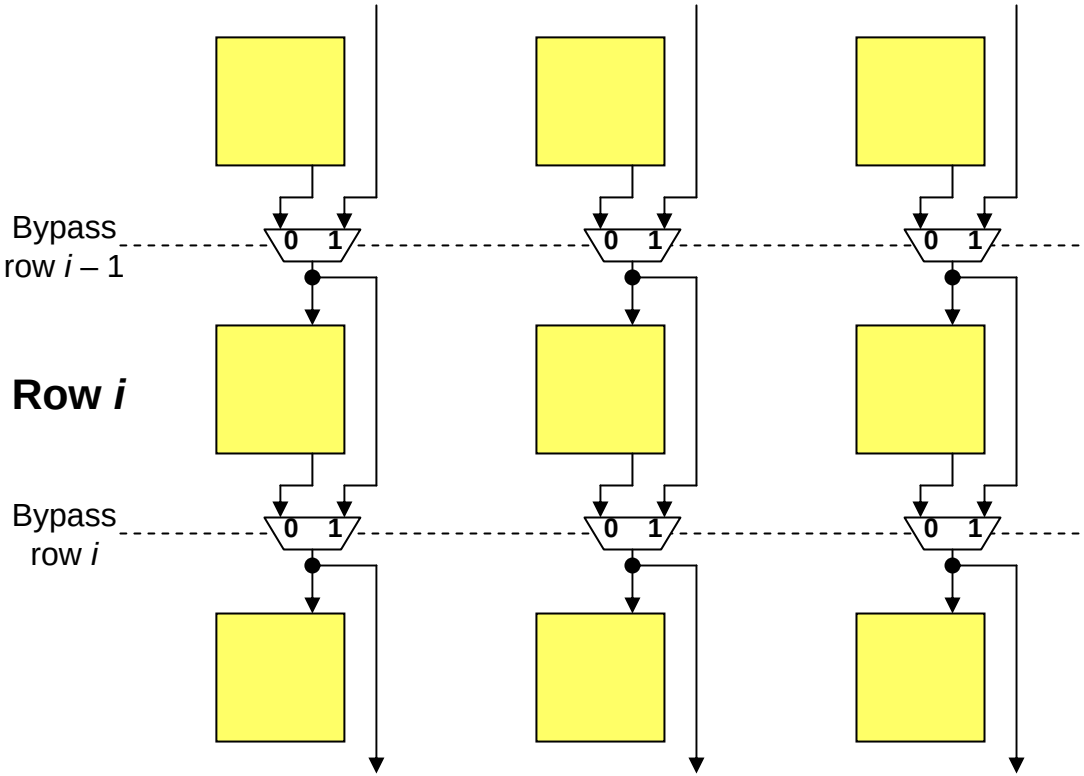
Failed units can be isolated from the buses

No single bus failure can isolate a module from the rest of the system

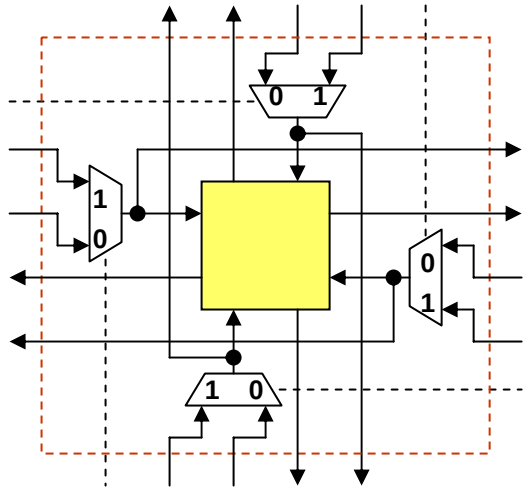
If we have extra buses, then failure in the bus connection logic can be tolerated by avoiding the particular bus

For reliability analysis, lump the failure rate of reconfiguration logic with that of its associated bus

Row/Column Bypassing in 2D Arrays

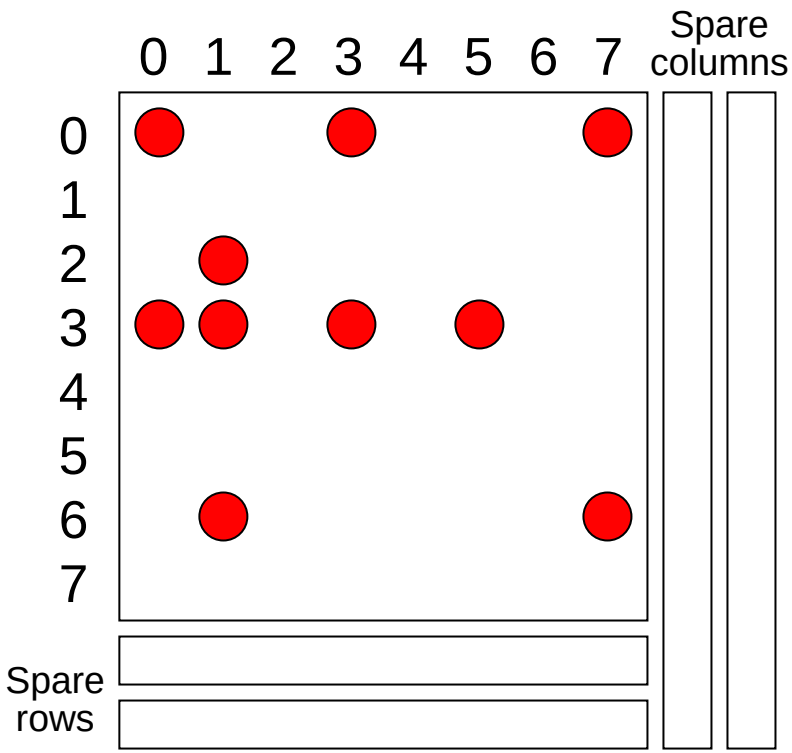


Similar mechanisms needed for northward links in columns and for the eastward and westward links in rows



Question: What types of mechanisms do we need at the edges of this array to allow the row and column edge connections to be directed to the appropriate (nonbypassed) rows and columns?

Choosing the Rows/Columns to Bypass

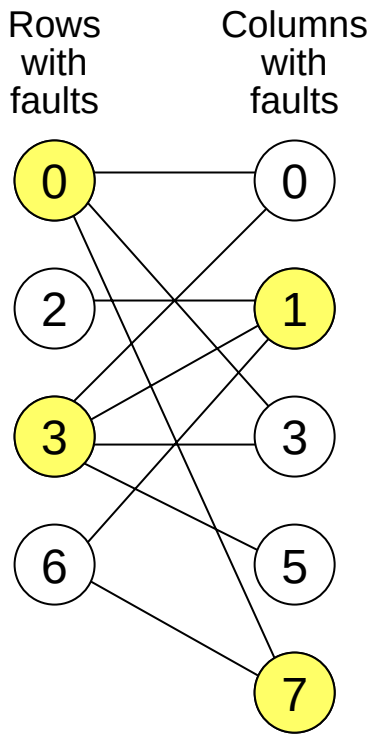


In the adjacent diagram, can we choose up to 2 rows and 2 columns so that they contain all the faults?

Convert to graph problem (Kuo-Fuchs):

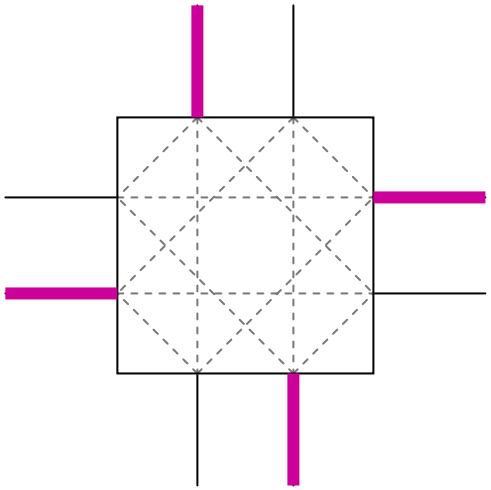
Form bipartite graph, with nodes corresponding to faulty rows and columns

Find a cover for the bipartite graph (set of nodes that touch every edge)



In a large array, with *r* spare rows and *c* spare columns, what is the smallest number of faults that cannot be reconfigured around with row/column bypassing?

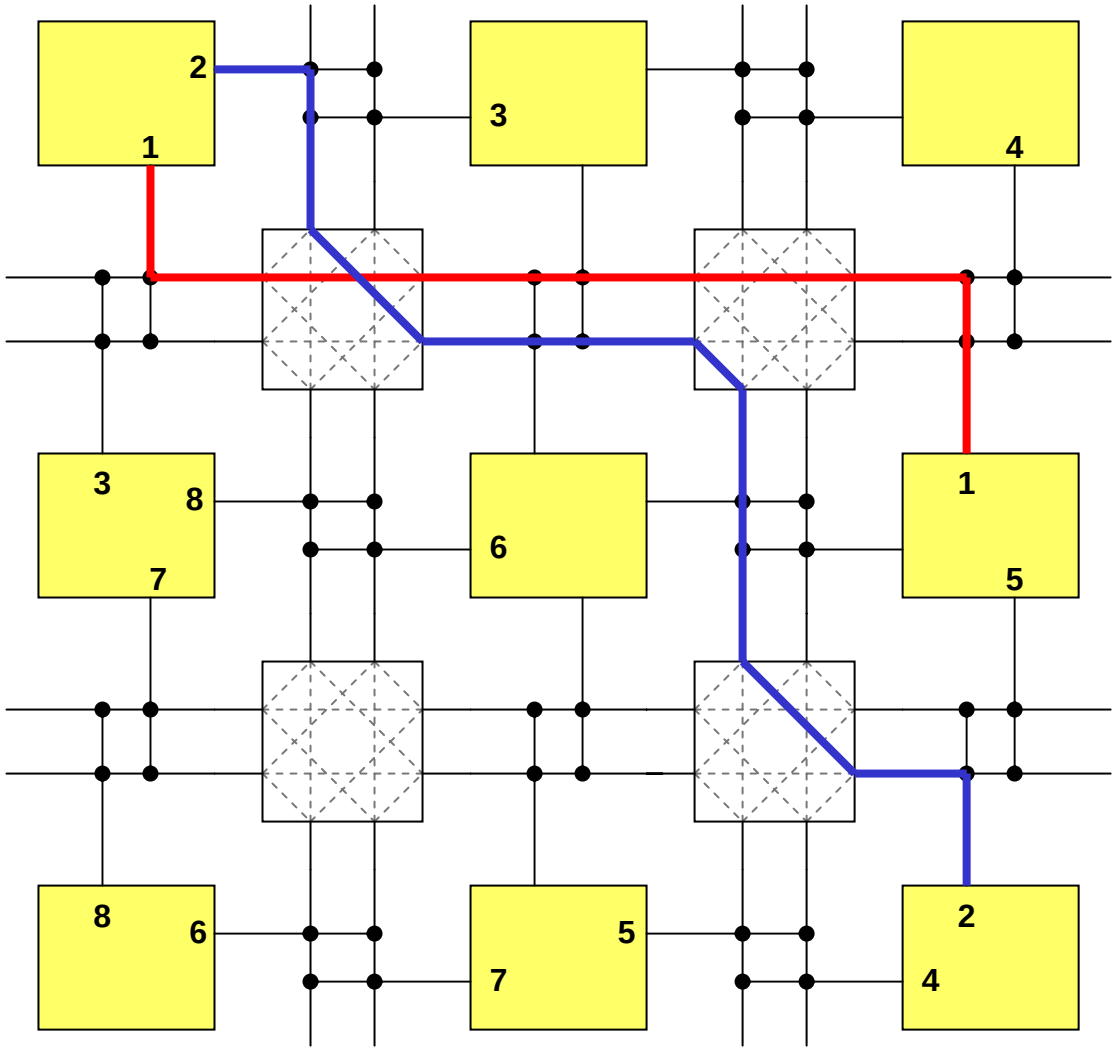
Switch Modules in FPGAs



Interconnection switch with 8 ports and four connection choices for each port:

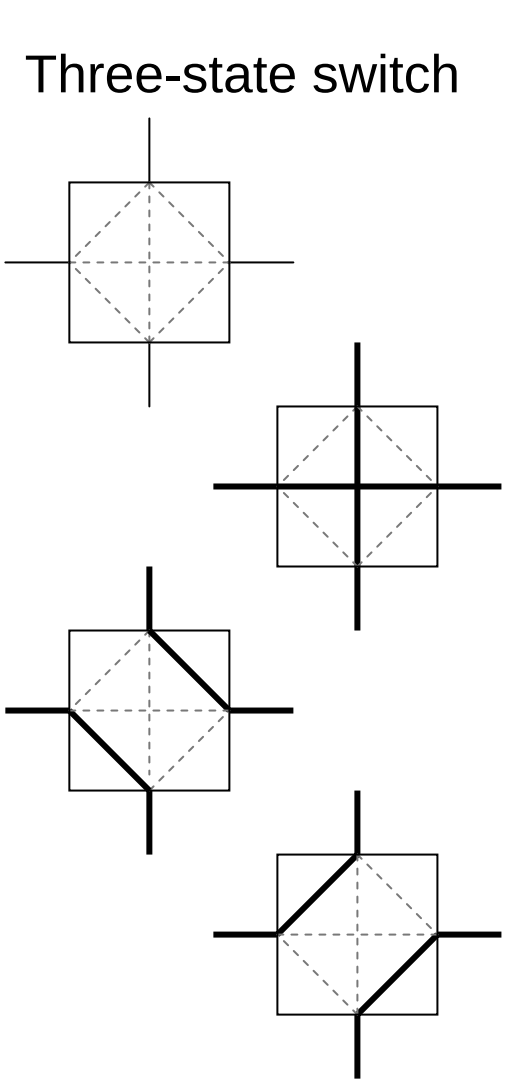
0 – No connection
1 – Straight through
2 – Right turn
3 – Left turn

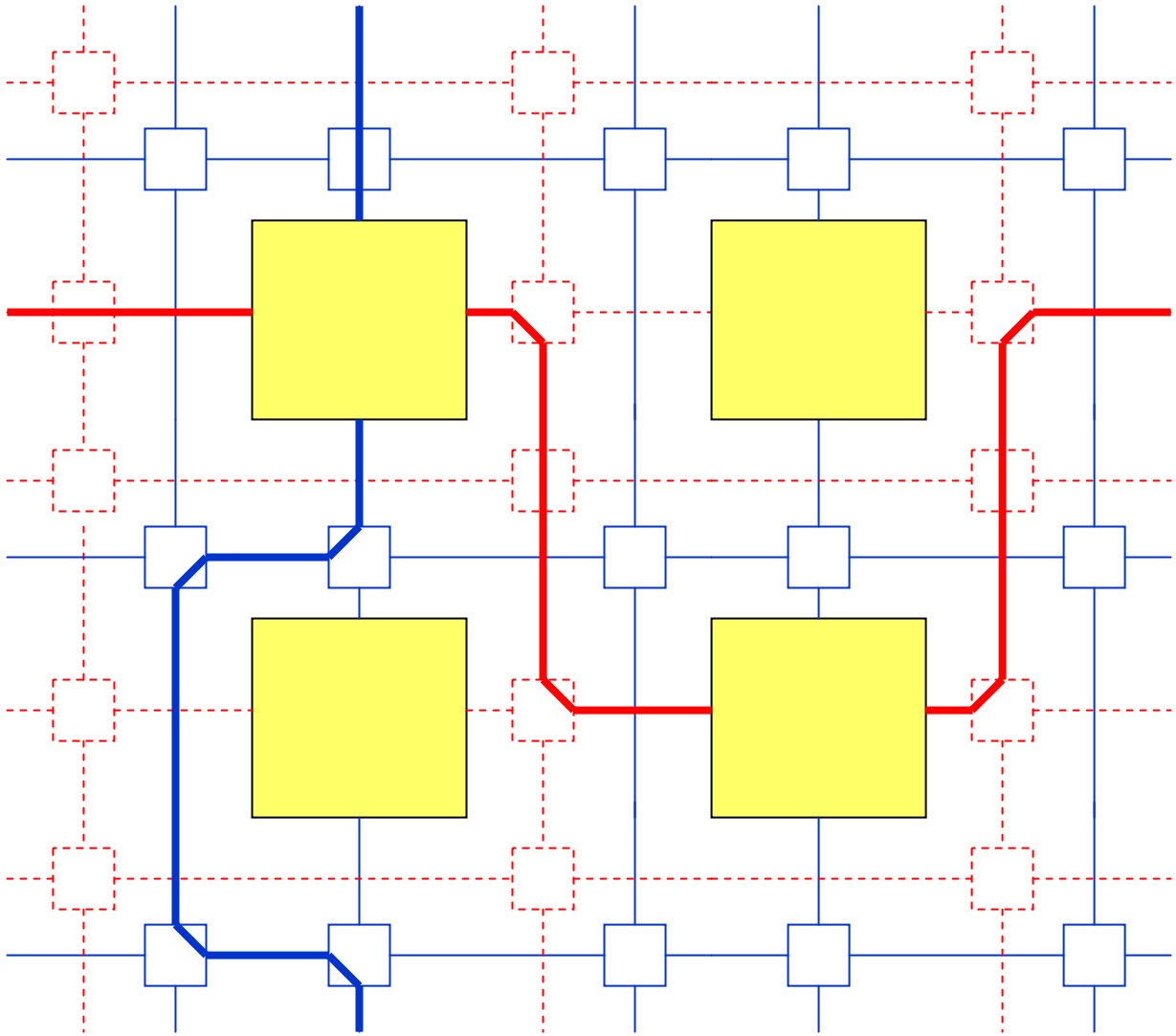
8 control bits (why?)



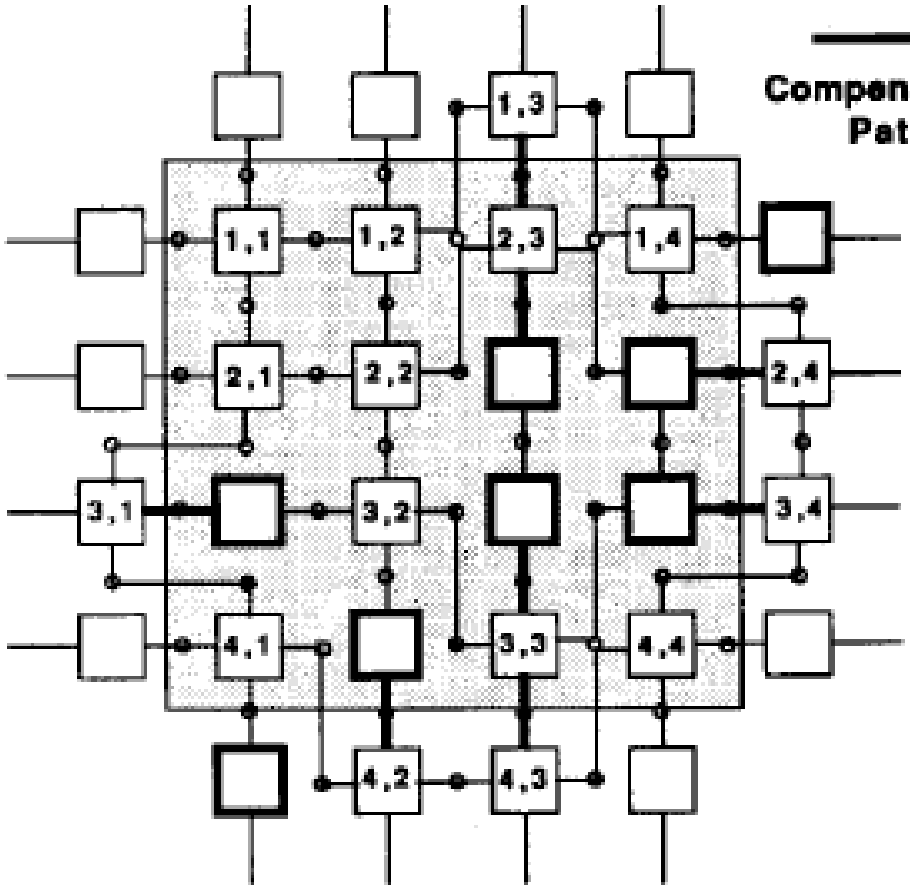
An Array Reconfiguration Scheme

Three-state switch

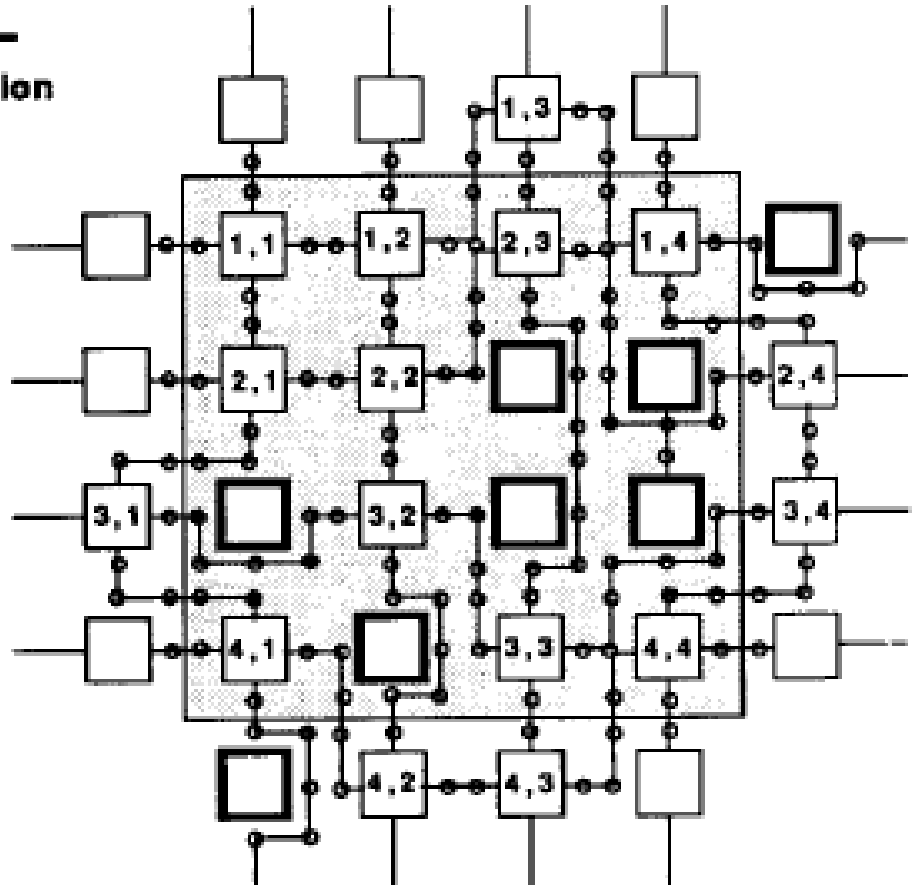




One-Track and Two-Track Switching Schemes



One-track switching model



Two-track switching model

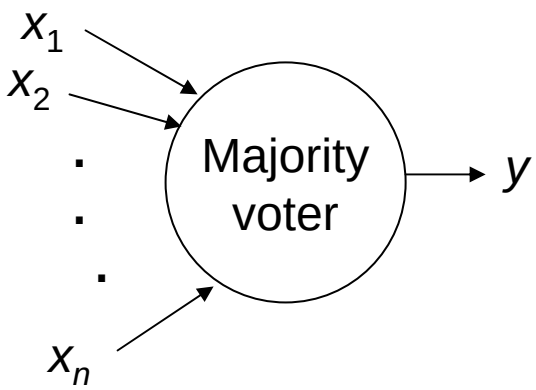
Source: S.-Y. Kung, S.-N. Jean, C.-W. Chang, *IEEE TC*, Vol. 38, pp. 501-514, April 1989

What We Already Know About Voting

Majority voting: Select the value that appears on at least $\lfloor n/2 \rfloor + 1$ of the n inputs

Number n of inputs is usually odd, but does not have to be

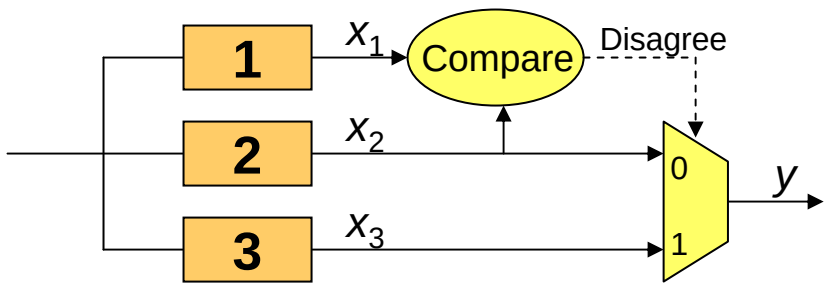
Example: vote(1, 2, 3, 2, 2) = 2



Majority voters can be realized by means of comparators and muxes

This design assumes that in the case of 3-way disagreement any one of the inputs can be chosen

Can add logic to this voter so that it signals three-way disagreement

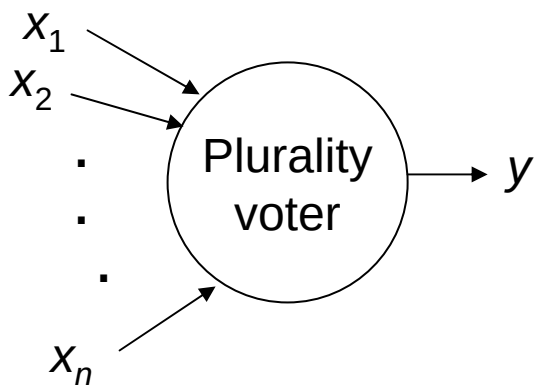


There is More to Voting than Simple Majority

Plurality voting: Select the value that appears on the largest number of inputs

Example: vote(1, 3, 2, 3, 4) = 3

What should we take as the result of vote(1.00, 3.00, 0.99, 3.00, 1.01)?



It would be reasonable to take 1.00 as the result, because 3 inputs agree or approximately agree with 1.00, while only 2 agree with 3.00

Will discuss approximate voting and a number of other sophisticated voting schemes under software design topics

Median voting: one way to deal with approximate values
median(1.00, 3.00, 0.99, 3.00, 1.01) = 1.01

Median value is equal to the majority value when we have majority

Threshold Voting and Its Generalizations

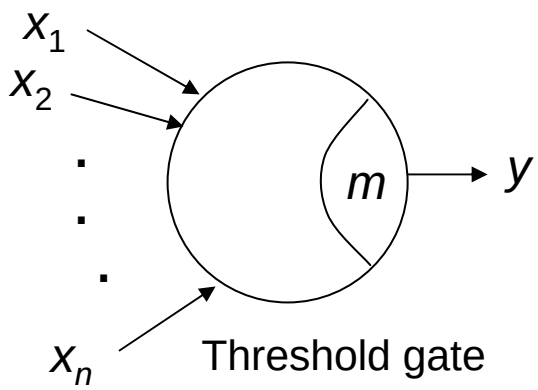
Simple threshold (m-out-of-n) voting:
Output is 1 if at least m of the n inputs are 1s

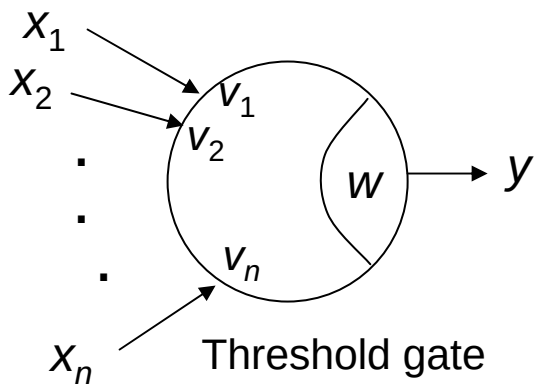
Majority voting is a special case of threshold voting: $(\lfloor n/2 \rfloor + 1)$-out-of-$n$ voting

Weighted threshold (w-out-of-Σv_i) voting:
Output is 1 if at least $\Sigma v_i x_i$ is w or more

Agreement or quorum sets
$\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_1\}$ – same as 2-out-of-3
$\{x_1, x_2\}, \{x_1, x_3, x_4\}, \{x_2, x_3, x_4\}$

The 2nd example above is weighted voting with $v_1 = v_2 = 2$, $v_3 = v_4 = 1$, and threshold $w = 4$



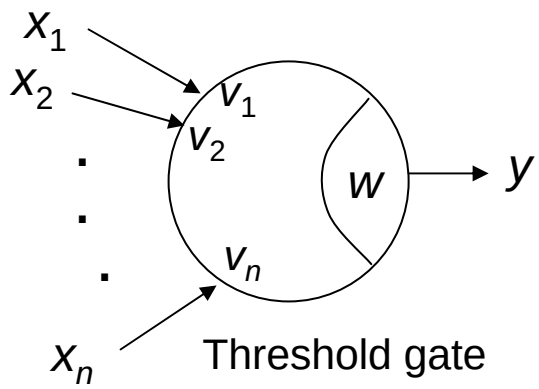


Agreement sets are more general than weighted voting in the sense of some agreement sets not being realizable as weighted voting

Usefulness of Weighted Threshold Voting

Unequal weights allow us to take different levels of input reliabilities into account

Zero weights can be used to disable or purge some of the inputs (combined switch-voter)



Maximum-likelihood voting

prob{x_1 correct} = 0.9
prob{x_2 correct} = 0.9
prob{x_3 correct} = 0.8
prob{x_4 correct} = 0.7
prob{x_5 correct} = 0.6

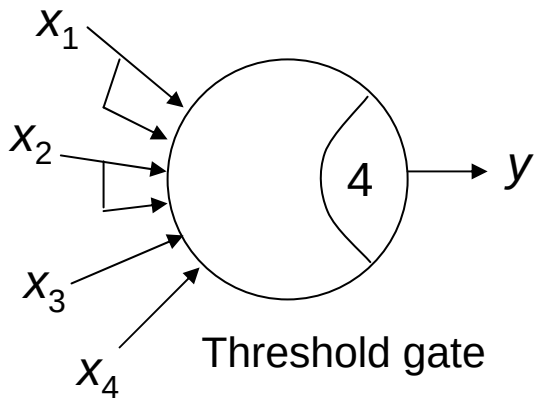
Max-likelihood voting can be implemented in hardware via table lookup or approximated by weighted threshold voting (otherwise, we need software)

Assume $x_1 = x_3 = a$, $x_2 = x_4 = x_5 = b$

prob{a correct} = $0.9 \times 0.1 \times 0.8 \times 0.3 \times 0.4 = 0.00864$
prob{b correct} = $0.1 \times 0.9 \times 0.2 \times 0.7 \times 0.6 = 0.00756$

Implementing Weighted Threshold Voters

Example: Implement a 4-input threshold voter with $v_1 = v_2 = 2$, $v_3 = v_4 = 1$, and threshold $w = 4$



Strategy 1: If weights are small integers, fan-out each input an appropriate number of times and use a simple threshold voter

Strategy 2: Use table lookup based on comparison results

$x_1=x_2$	$x_1=x_3$	$x_2=x_3$	$x_3=x_4$	Result
1	x	x	x	x_1
0	0	0	1	Error
0	1	0	1	x_1
0	0	1	1	x_2
0	x	x	0	Error

Is this table complete? Why, or why not?

Strategy 3: Convert the problem to agreement sets (discussed next)

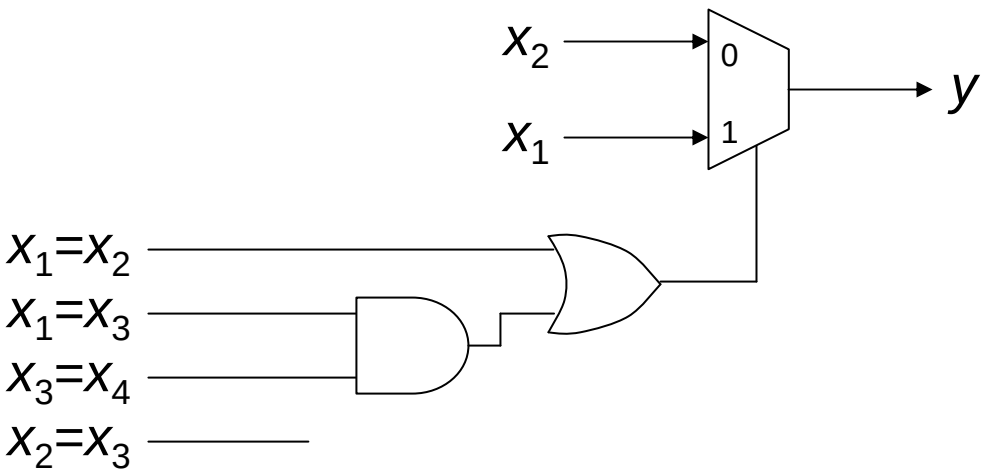
Implementing Agreement-Set Voters

Example: Implement a voter corresponding to the agreement sets $\{x_1, x_2\}$, $\{x_1, x_3, x_4\}$, $\{x_2, x_3, x_4\}$

Strategy 1: Implement as weighted threshold voter, if possible

Strategy 2: Implement directly

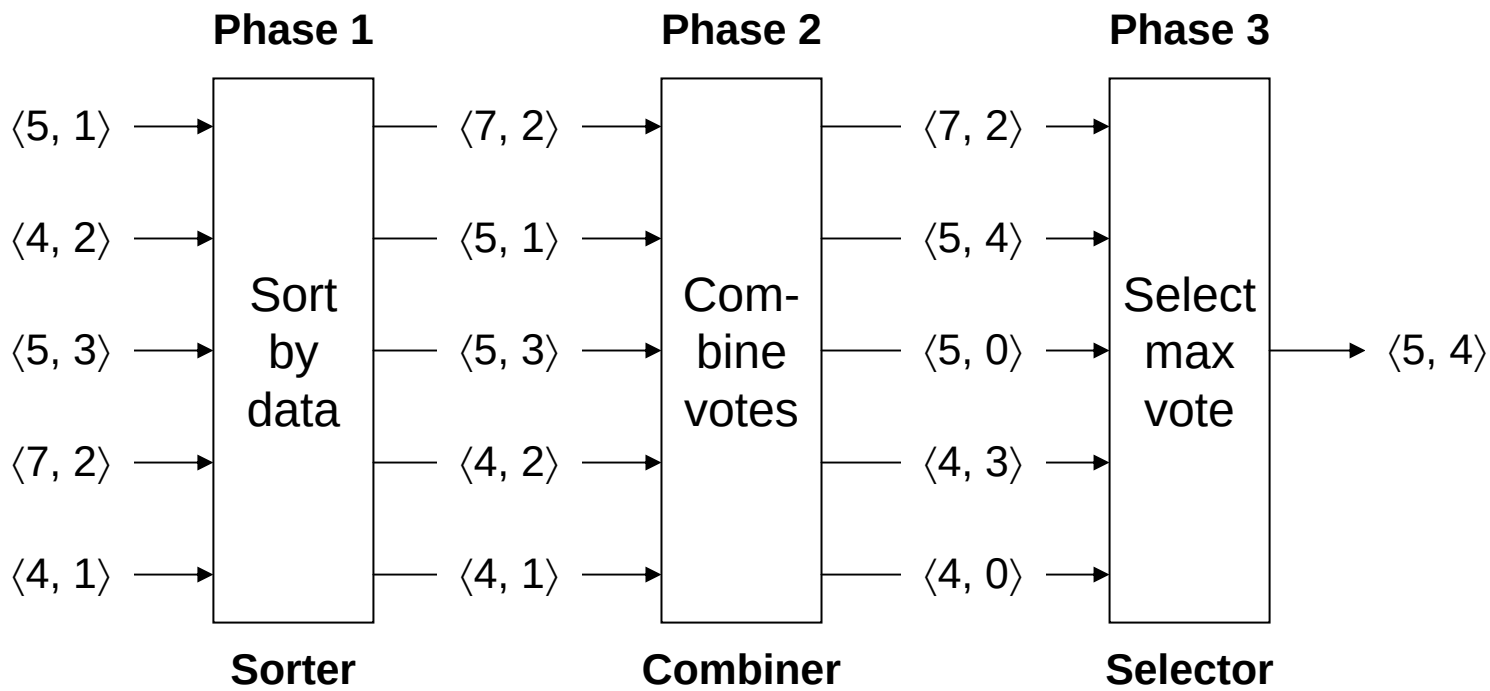
Find a minimal set of comparators that determine the agreement set



Complete this design by producing the "no agreement" signal

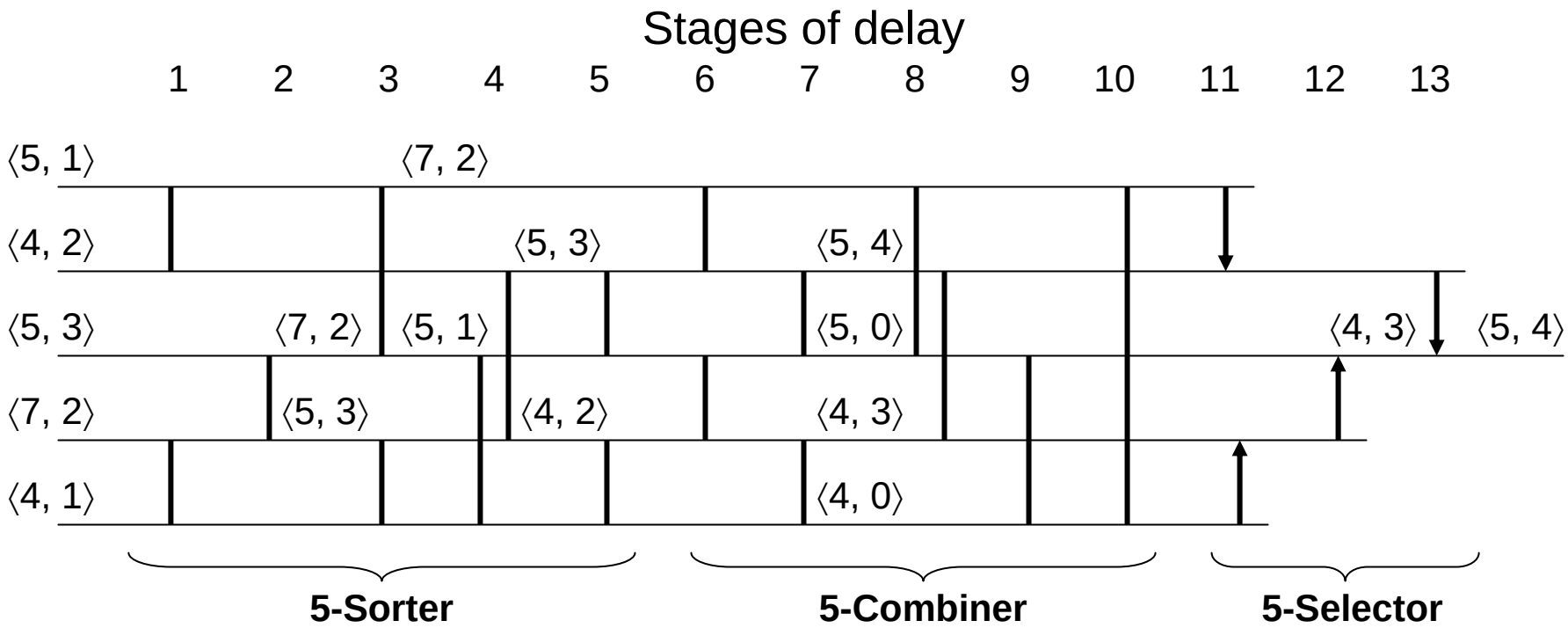
Implementing Weighted Plurality Voters

Inputs: Data, vote-weight pairs
Output: Data with maximal vote and its associated vote tally



Source: B. Parhami, *IEEE Trans. Reliability*, Vol. 40, No. 3, pp. 380-394, August 1991

Details of a Sorting-Based Plurality Voter



The first two phases (sorting and combining) can be merged, producing a 2-phase design – fewer, more complex cells lead to tradeoff

Source: B. Parhami, *IEEE Trans. Reliability*, Vol. 40, No. 3, pp. 380-394, August 1991