RAM Fault Models & Test Algorithms

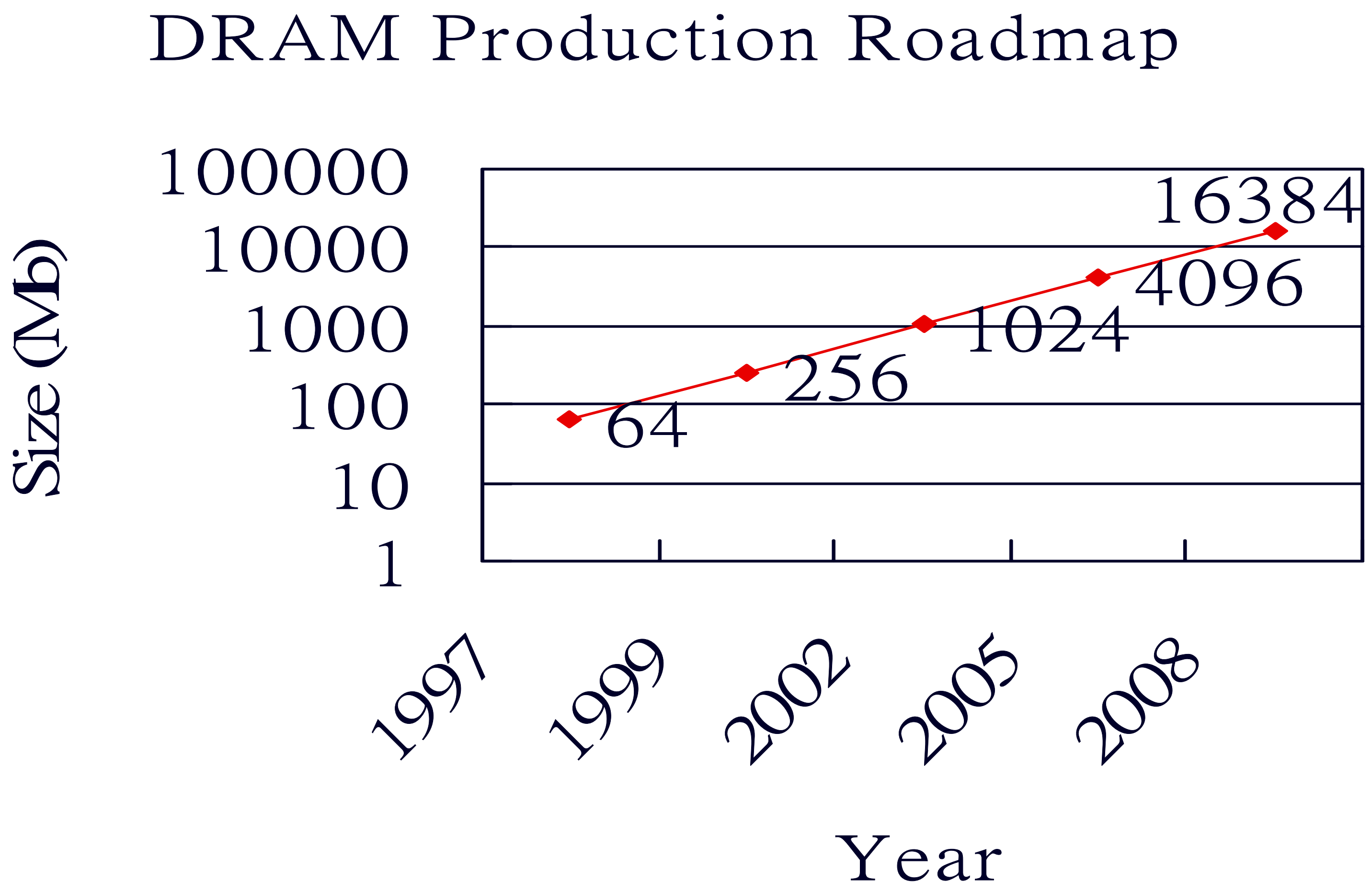
Cheng-Wen Wu 吳誠文

Lab for Reliable Computing
Dept. Electrical Engineering
National Tsing Hua University

Outline

- Introduction
  - Off-line RAM testing
  - RAM functional models
- RAM functional fault models
  - Static faults
  - Dynamic faults
- Classical RAM test algorithms
  - Test time complexity
- March test algorithms

Introduction



- Testing cost also rises rapidly, if DFT is not used

Off-Line RAM Testing

- Parametric Test: DC & AC
- Reliability Screening
  - Long-cycle testing
  - Burn-in: static & dynamic BI
- Functional Test
  1. Device characterization
     - Failure analysis
  2. Fault modeling
     - Simple but effective (accurate & realistic?)
  3. Test algorithm generation
     - Small number of test patterns (data backgrounds)
     - High fault coverage
     - Short test time

DC Parametric Test

- Contact test
- Power consumption test
- Leakage/standby test
- Threshold test
- Output drive current test
- Output short current test
- Etc.

AC Parametric Test

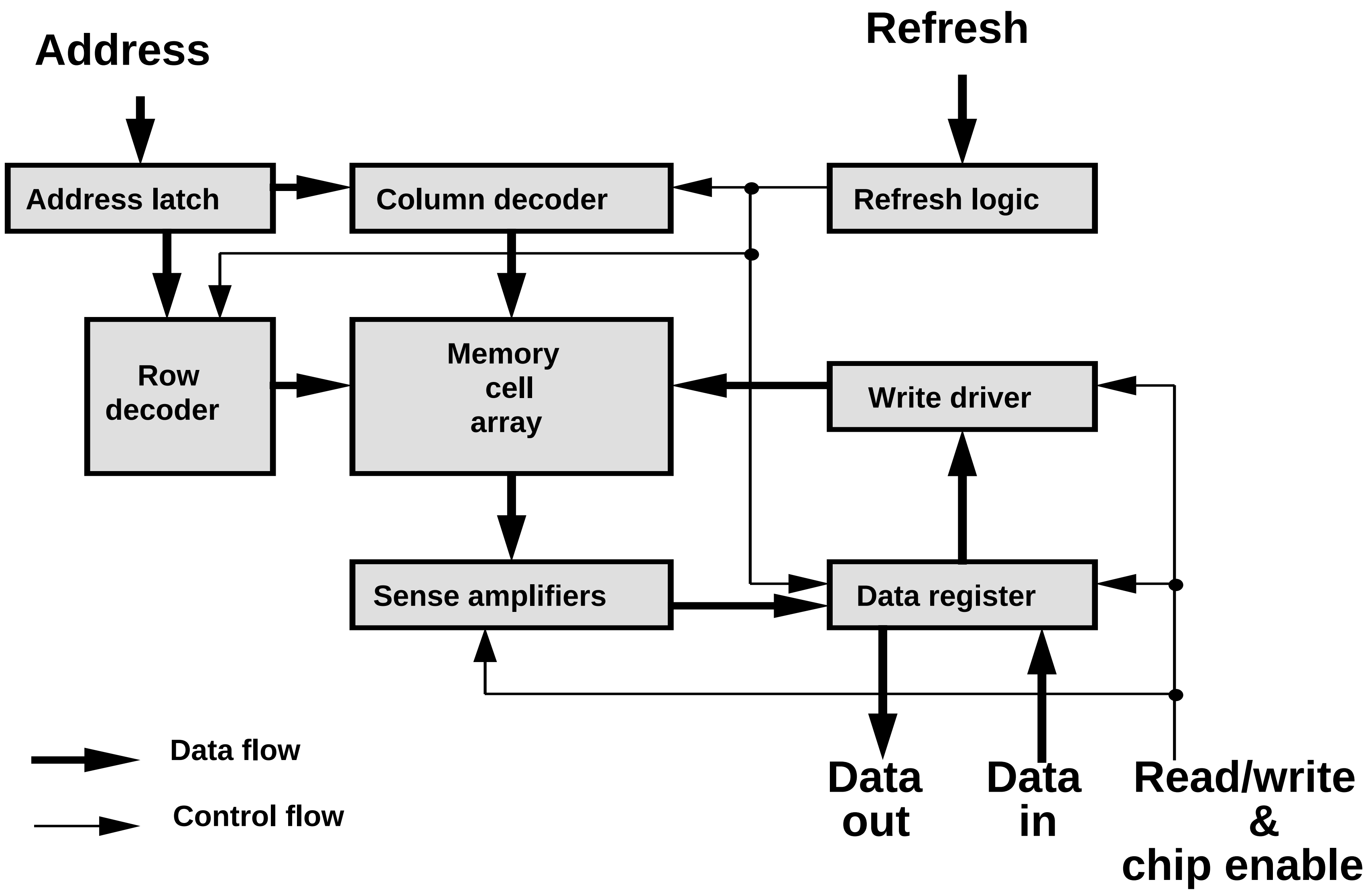
- Output signals (data)
  - Rise & fall times
- Input signals (address/data)
  - Setup & hold times
- Relationship between input and output signals
  - Delay & access times
  - Speed test

RAM Models

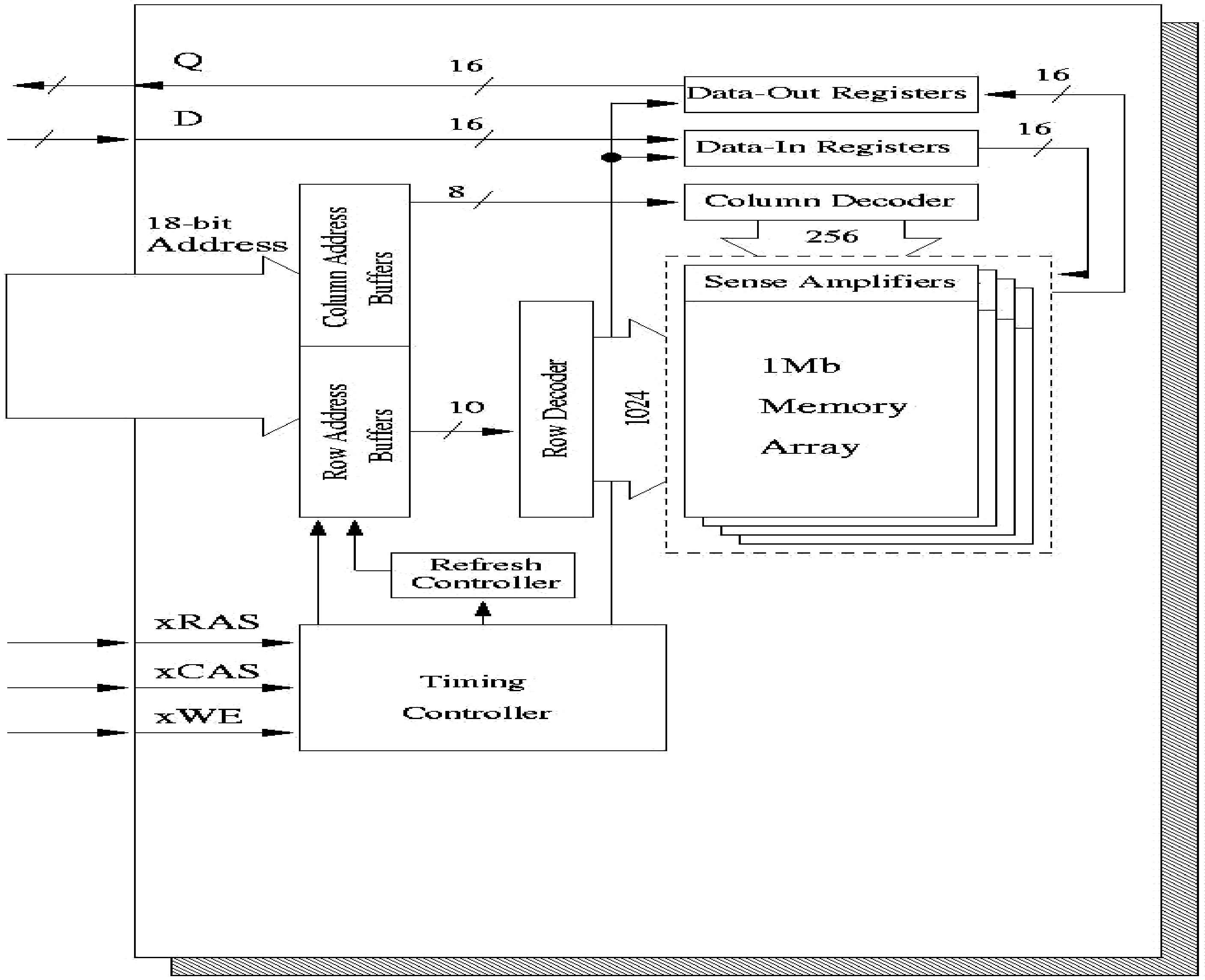
- Behavior Level
  - Verilog/VHDL
- Function Level
  - Verilog/VHDL/Block diagram
  - Normally not synthesizable
- Circuit Level
  - Spice/Schematic
- Layout Level
  - GDS-II/Geometry

☞ Who should provide the models?

DRAM Functional Model



DRAM Functional Model Example



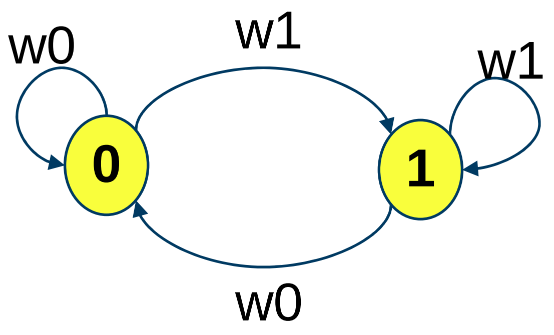
Functional Fault Models

- Classical fault models are not sufficient to represent all important failure modes in RAM
- Sequential ATPG is not possible for RAM
- Functional fault models are commonly used for memories
  - They define functional behavior of faulty memories
- New fault models are being proposed to cover new defects and failures in modern memories
  - New process technology
  - New device
    - Material/cell/circuit/architecture/interface

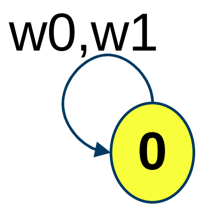
Static RAM Fault Models: SAF/TF

- Stuck-At Fault (SAF)
  - Cell (line) SA0 or SA1
    - A stuck-at fault (SAF) occurs when the value of a cell or line is always 0 (a stuck-at-0 fault) or always 1 (a stuck-at-1 fault)
    - Theorem: A test that detects all SAFs guarantees that from each cell, a 0 and a 1 must be read.
- Transition Fault (TF)
  - Cell fails to transit from 0 to 1 or 1 to 0 in specified time period
    - A cell has a transition fault (TF) if it fails to transit from 0 to 1 (a <$\uparrow$/0> TF) or from 1 to 0 (a <$\downarrow$/1> TF)

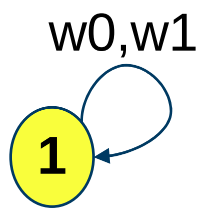
State Diagram Representation



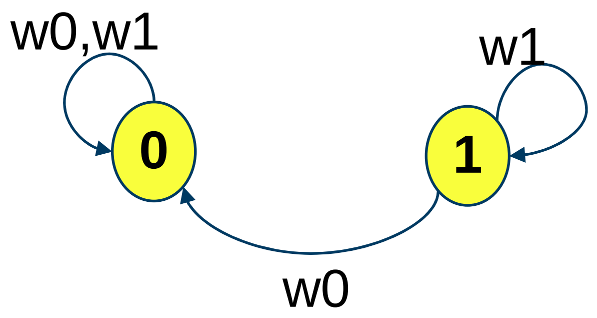
Fault-free RAM cell



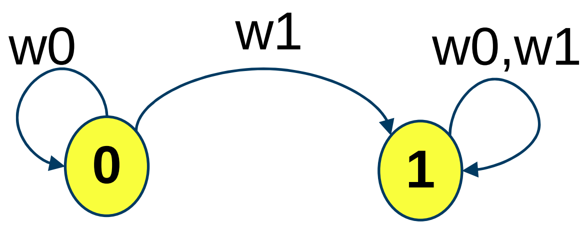
SA0F



SA1F



Rising TF



Falling TF

Static RAM Fault Models: AF

- Address-Decoder Fault (AF)
  - An address decoder fault (AF) is a functional fault in the address decoder that results in one of four kinds of abnormal behavior:
    - Given a certain address, no cell will be accessed
    - A certain cell is never accessed by any address
    - Given a certain address, multiple cells are accessed
    - A certain cell can be accessed by multiple addresses

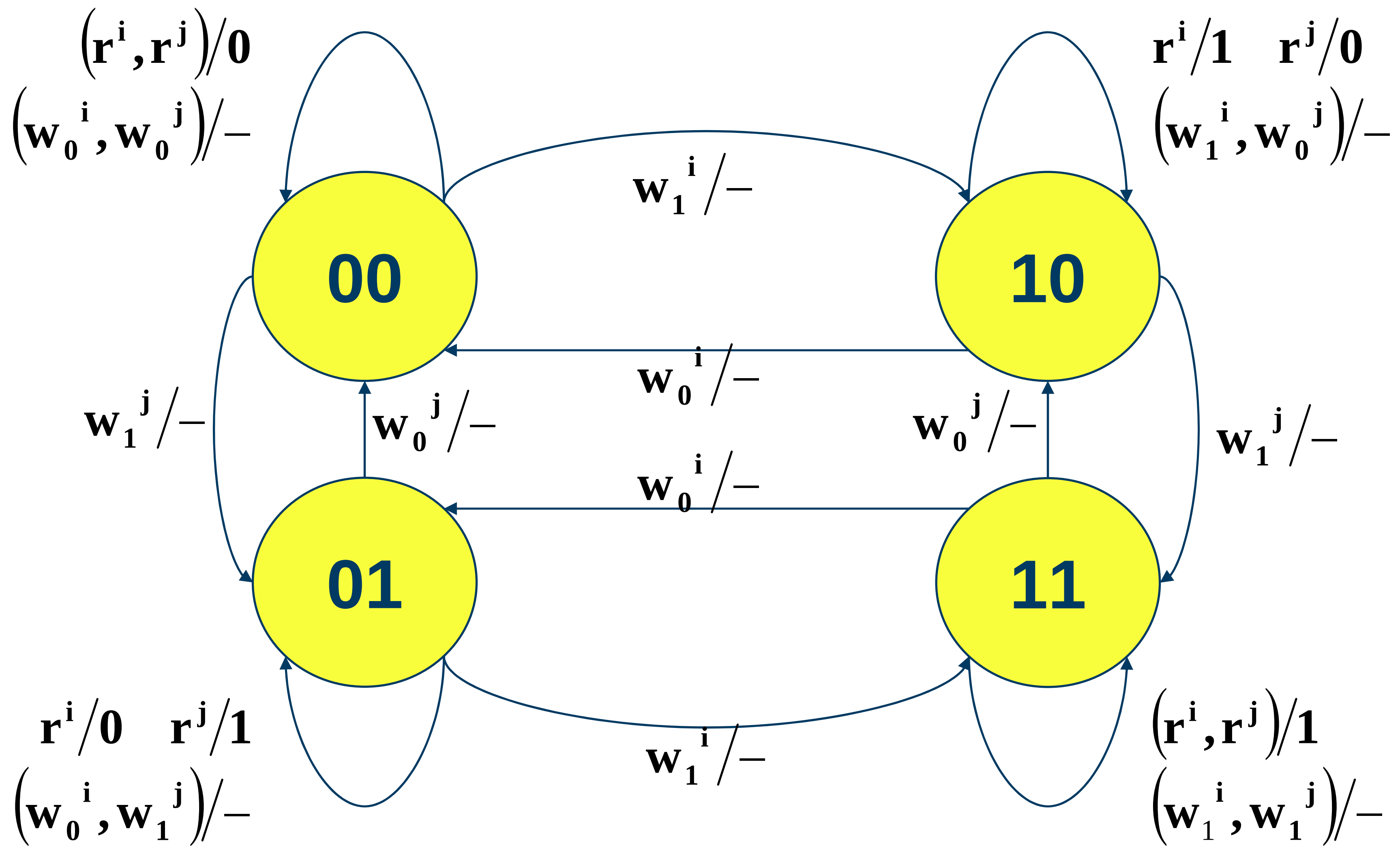
Static RAM Fault Models: BF/SOF

- Bridging Fault (BF)
  - A bridging fault (BF) occurs when there is a short between two cells
    - ∗ AND-type BF
    - ∗ OR-type BF
- Stuck-Open Fault (SOF)
  - A stuck-open fault (SOF) occurs when the cell cannot be accessed due to, e.g., a broken word line
  - A read to this cell will produce the previously read value

RAM Fault Models: CF

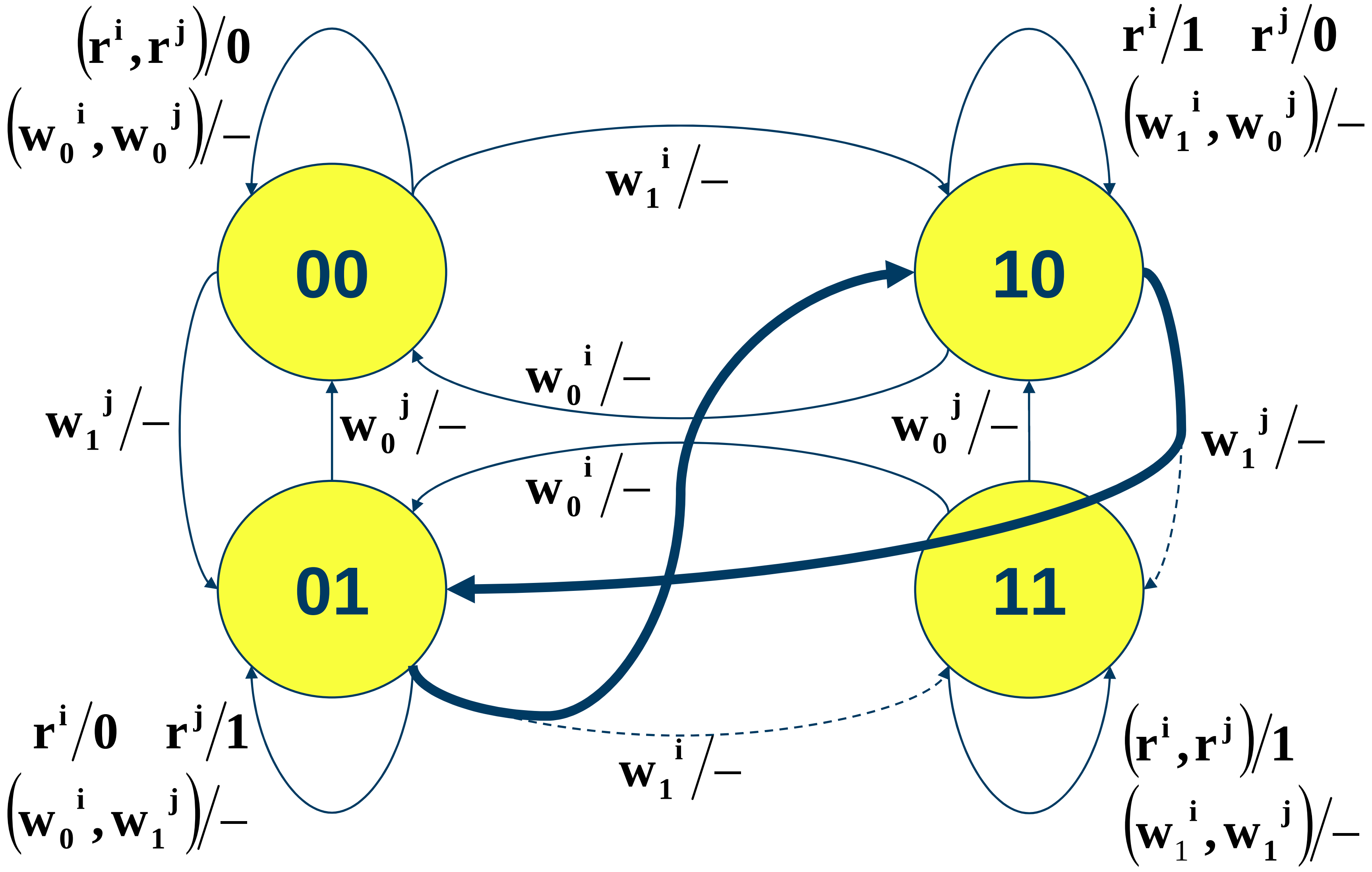
- Coupling Fault (CF)
  - A coupling fault (CF) between two cells occurs when the logic value of a cell is influenced by the content of, or operation on, another cell
  - State Coupling Fault (CFst)
    - Coupled (victim) cell is forced to 0 or 1 if coupling (aggressor) cell is in given state
  - Inversion Coupling Fault (CFin)
    - Transition in coupling cell complements (inverts) coupled cell
  - Idempotent Coupling Fault (CFid)
    - Coupled cell is forced to 0 or 1 if coupling cell transits from 0 to 1 or 1 to 0

State Diagram for 2 Fault-Free Cells

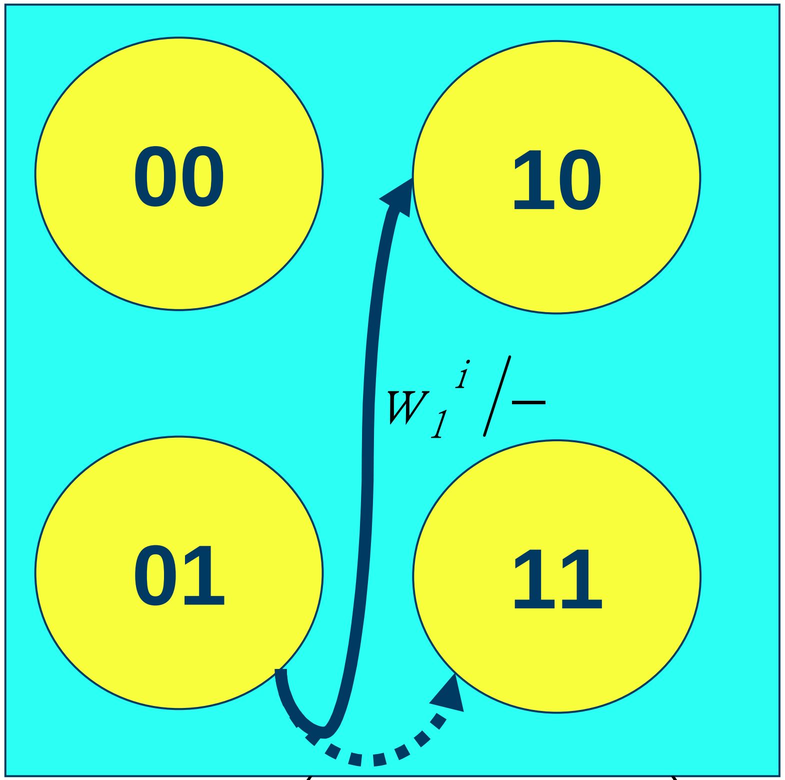


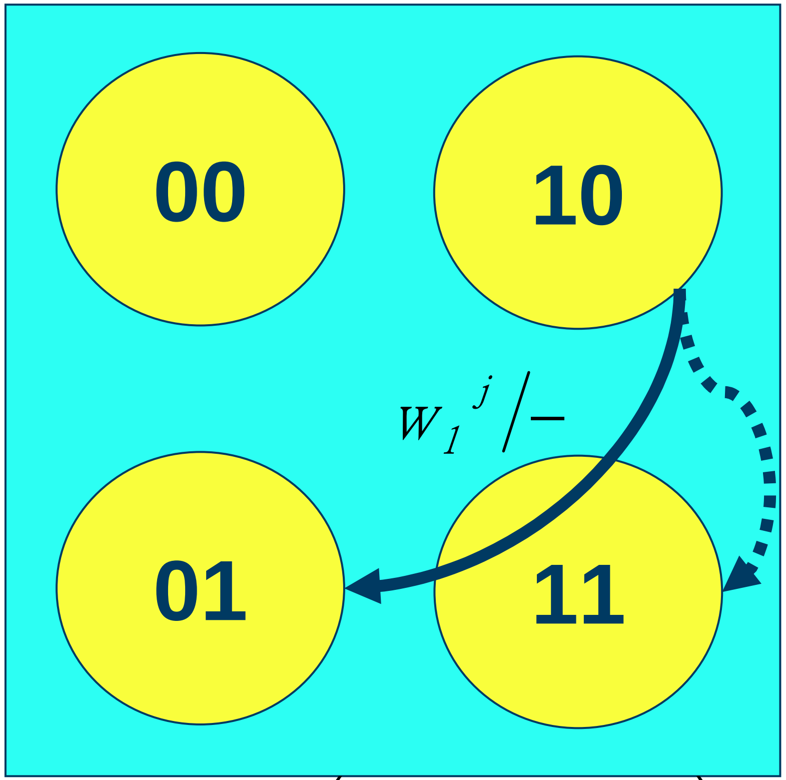
Exercise: draw the state diagrams for the CFs.

FSM for CFid< ↑ ;0>



BFE Model for CFid < ↑ ;0>



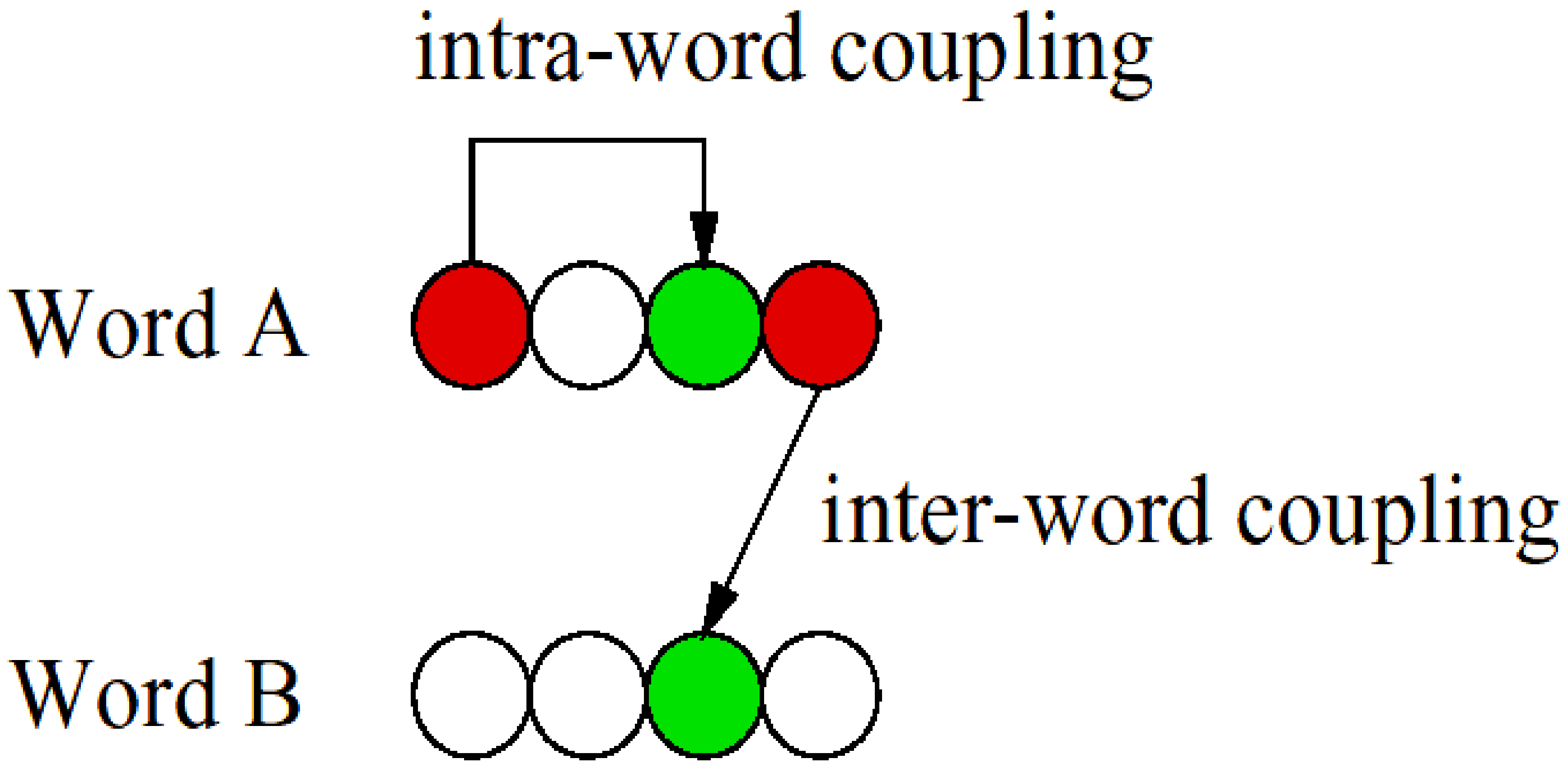


$$TP_1 = \left(01, w_1^i, r_1^j\right) \qquad TP_2 = \left(10, w_1^j, r_1^i\right)$$

BFE (Basic Fault Effect): an FSM with a branch that differs form the fault free model by one transition only

Source: Benso *et al*., DATE02

Intra-Word & Inter-Word CFs



RAM Fault Models: PSF

- Pattern-Sensitive Fault (PSF)
  - The PSF is a general (multi-cell) coupling fault, which causes the content of a memory cell, or the ability to change the content, to be influenced by certain patterns of other cells in the memory
    - In general, the number of aggressor and victim cells may be 4, 5, 9, etc.
- The target PSF is the Neighborhood PSF (NPSF)
  - The aggressor cells are the neighborhood of the victim cell
  - 5-cell neighborhood
  - 9-cell neighborhood

5-Cell NPSF

	N	
W	BC	E
	S	

B	E						
S			N				
		W	B	E			
			S			N	
					W	B	E

Base cell: B

Deleted neighborhood cells: N, E, W, S

Neighborhood cells: B and N, E, W, S

9-Cell NPSF

	NW	N	NE	
	W	B	E	
	SW	S	SE	

NPSF Models: SNPSF

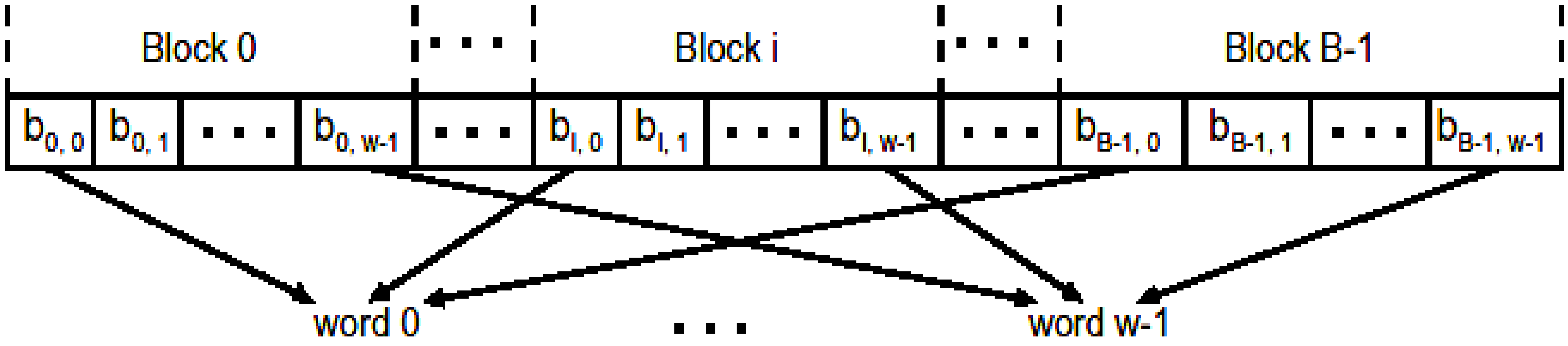
- Static NPSF (SNPSF)
  - BC is forced to a certain state (0 or 1) due to the appearance of a certain pattern in deleted neighborhood (DN) cells
- Assumptions:
  - Single NPSF
  - Address scramble table is available
  - Memory is bit-oriented
    - Word-oriented memory is tested as multiple bit-oriented ones

NPSF Models: PNPSF/ANPSF

- Passive NPSF (PNPSF)
  - BC cannot change its state from 0 to 1 or from 1 to 0 due to the appearance of a certain pattern in the DN cells
- Active NPSF (ANPSF)
  - BC is forced to 0 or 1 when an up transition or down transition occurs in a DN cell, while other DN cells assume a certain pattern
    - Change: a transition in one DN cell, with other DN cells & BC containing a certain pattern

Address & Data Scrambling

- Address scrambling
  - To minimize silicon area and delay, internal address lines (word-lines and bit-lines) are frequently scrambled
    - Physical address is not identical to the logical address
- Data scrambling
  - Each bit of a word is from a different block
    - Physical order of the bits is not the same with the logical order



Block 0
. . .
Block i
. . .
Block B-1
$b_{0,0}$ $b_{0,1}$ . . . $b_{0,w-1}$ . . . $b_{i,0}$ $b_{i,1}$ . . . $b_{i,w-1}$ . . . $b_{B-1,0}$ $b_{B-1,1}$ . . . $b_{B-1,w-1}$
word 0
. . .
word w-1

RAM Fault Models: RF

- Recovery Fault (RF)
  - Sense Amplifier Recovery Fault (SARF)
    - Sense amp saturation after reading/writing long run of 0 or 1
  - Write Recovery Fault (WRF)
    - Write followed by reading/writing at different location resulting in reading/writing at same location
      - ◊ Write-after-write recovery fault
      - ◊ Read-after-write recovery fault
  - Results in functional faults---detected at high speed (e.g., GALROW/GALCOL)

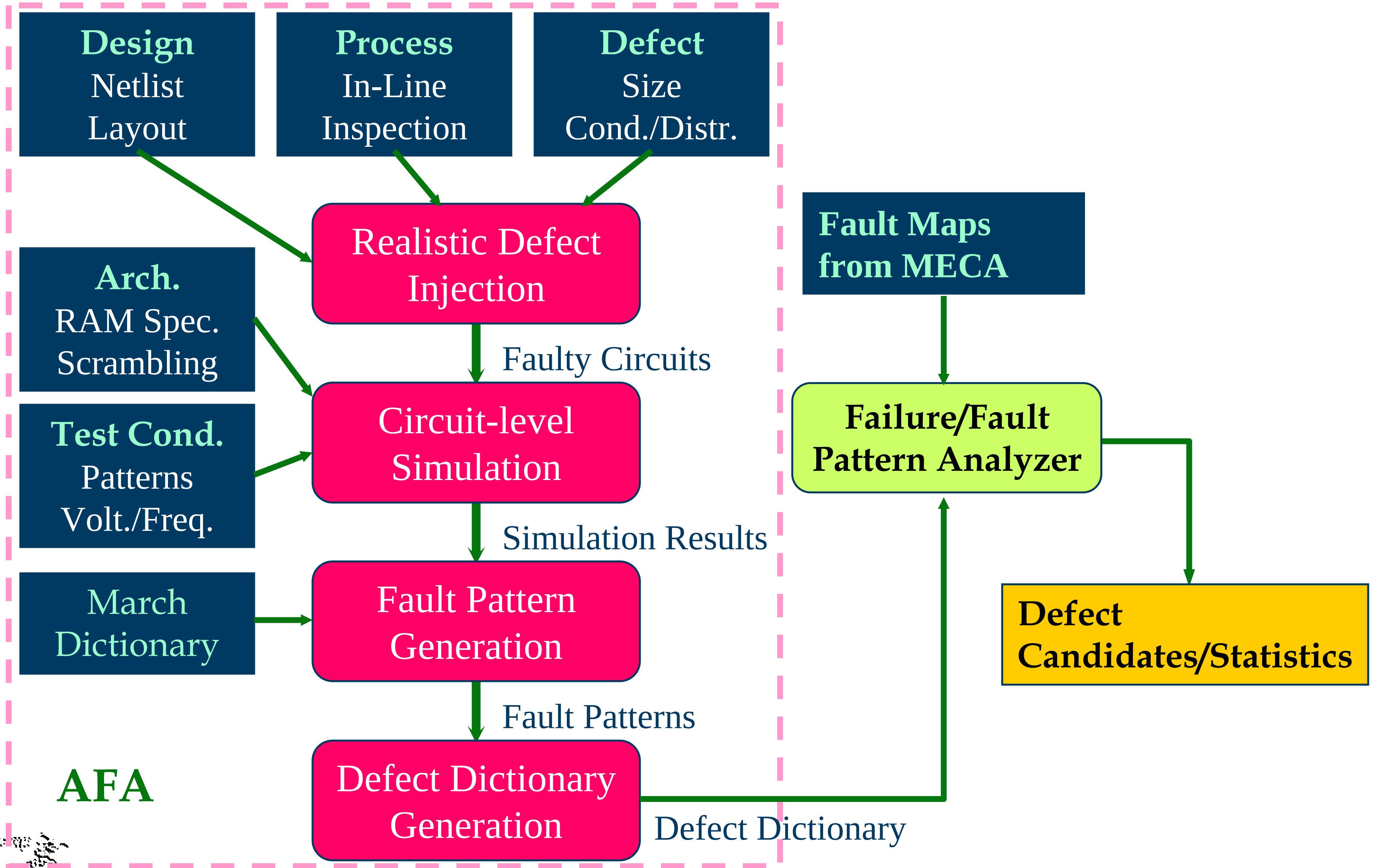
RAM Fault Models: DF

- Disturb Fault (DF)
  - Victim cell forced to 0 or 1 if we (successively) read or write aggressor cell (may be the same cell)
    - Hammer test
  - Read Disturb Fault (RDF)
    - There is a read disturb fault (RDF) if the cell value will flip when being read (successively)

RAM Fault Models: DRF

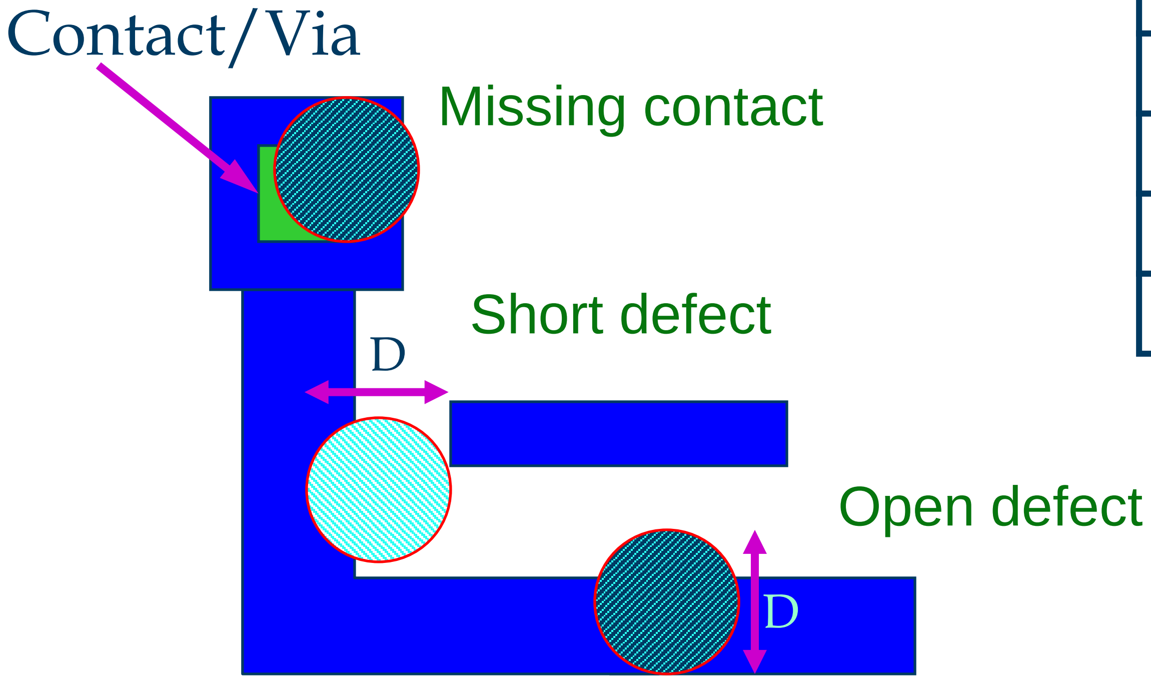
- Data Retention Fault (DRF)
  - DRAM
    - Refresh Fault
      - Refresh-Line Stuck-At Fault
    - Leakage Fault
      - Sleeping Sickness---loose data in less than specified hold time (typically tens of ms)
  - SRAM
    - Leakage Fault
      - Static Data Losses---defective pull-up
  - Checkerboard pattern triggers max leakage
  - BIST good for sync with refresh mechanism

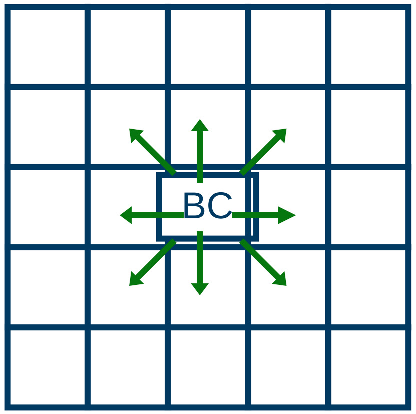
Memory Defect Diagnostics (MDD) System



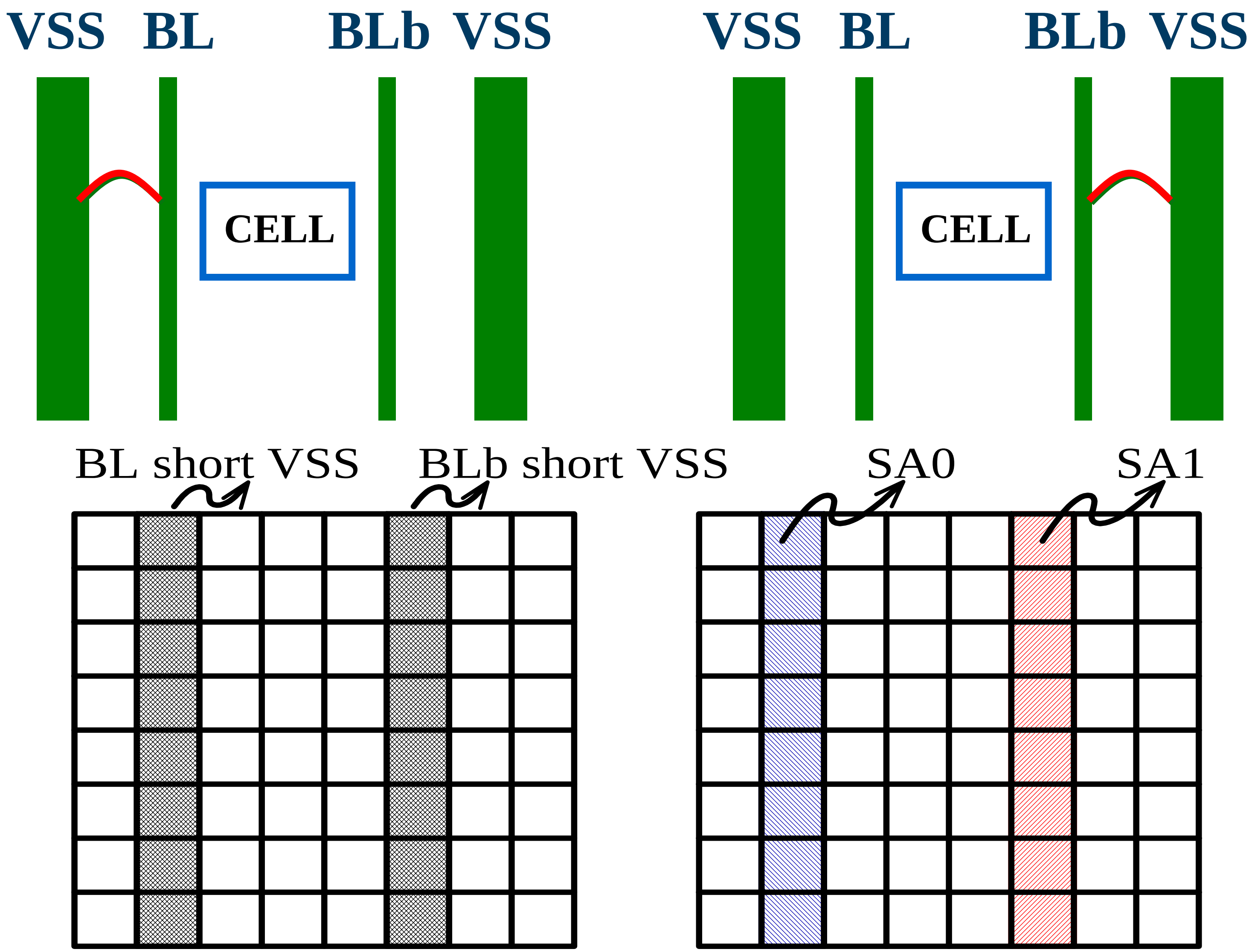
Realistic Defect Injection

- Purpose: to determine from the layout if a circuit is damaged by a certain point defect



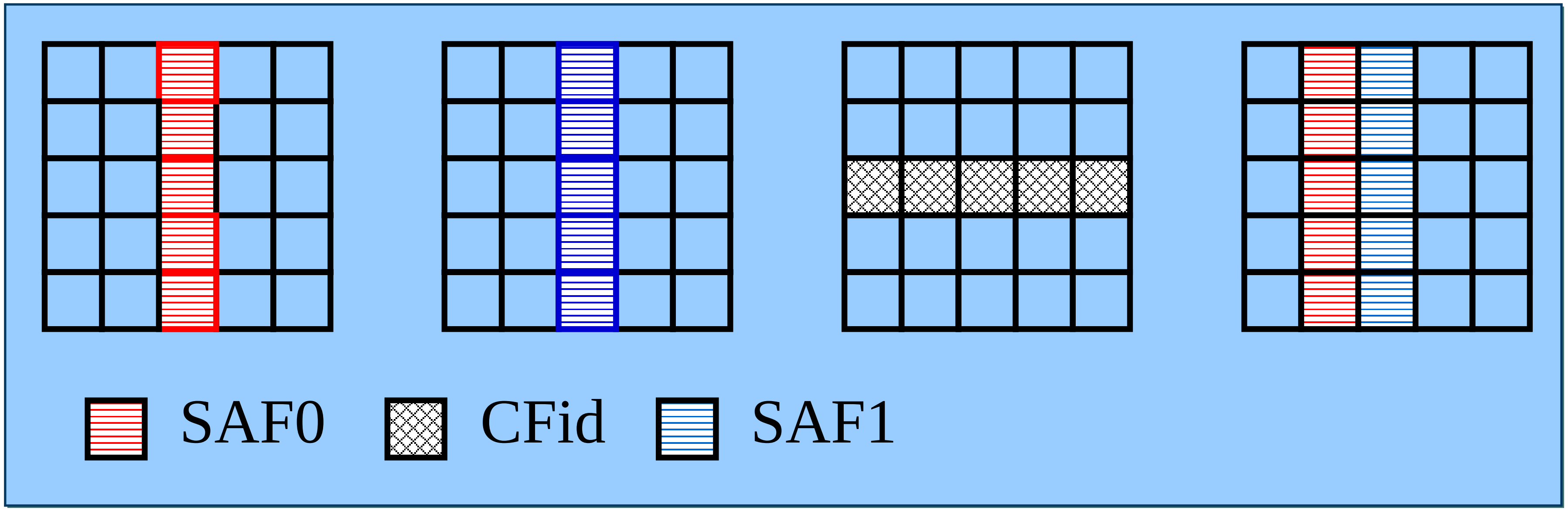


Inaccuracy of Failure Pattern



Fault Pattern

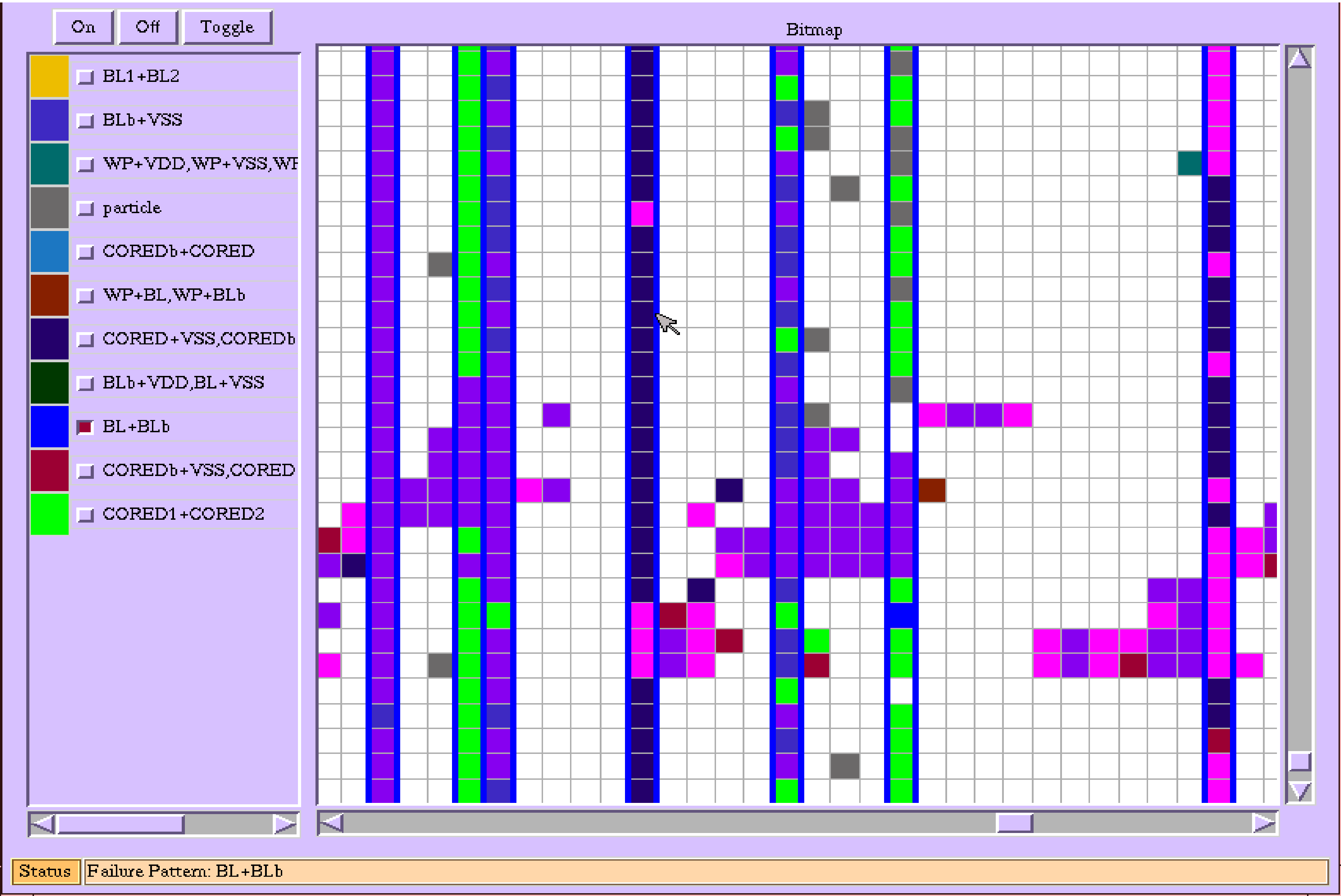
- Combining the failure pattern and fault model



Automatic Fault Analysis

- FA-based approach predicts the signature (faulty behavior) of a given defect
  - Defect model
    - Particle, missing/extra material, missing contacts
    - Resistive short/open
    - layout, defect distribution and characteristics
  - Circuit-level simulation for faulty circuit
- Purposes
  - Fault models
  - Defect dictionary
    - Fault signature and fault pattern

Failure/Fault Patterns



Status Failure Pattern: BL+BLb

Test Time Complexity (100MHz)

Size	N	10N	NlogN	$N^{1.5}$	N^2
1M	0.01s	0.1s	0.2s	11s	3h
16M	0.16s	1.6s	3.9s	11m	33d
64M	0.66s	6.6s	17s	1.5h	1.43y
256M	2.62s	26s	1.23m	12h	23y
1G	10.5s	1.8m	5.3m	4d	366y
4G	42s	7m	22.4m	32d	57c
16G	2.8m	28m	1.6h	255d	915c

RAM Test Algorithm

- A test algorithm (or simply test) is a finite sequence of test elements
  - A test element contains a number of memory operations (access commands)
    - Data pattern (background) specified for the Read and Write operation
    - Address (sequence) specified for the Read and Write operations
- A march test algorithm is a finite sequence of march elements
  - A march element is specified by an address order and a finite number of Read/Write operations

March Test Notation

- ⇑: address sequence is in the ascending order
- ⇓: address changes in the descending order
- ⇳: address sequence is either ⇑ or ⇓
- r: the Read operation
  - Reading an expected 0 from a cell (r0); reading an expected 1 from a cell (r1)
- w: the Write operation
  - Writing a 0 into a cell (w0); writing a 1 into a cell (w1)
- Example (MATS+): $\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)\}$

March Test Diagram (MTD)

- .: address sequence is in the ascending order
- ‘: address changes in the descending order
- :: address sequence is either . or ‘
- __: the r0 operation
- ‾‾: the r1 operation
- ↓: the w0 operation
- ↑: the w1 operation
- Example (MATS+): . ↓ . __ ↑ ‘ ‾‾ ↓

Classical Test Algorithms: MSCAN

- Zero-One Algorithm [Breuer & Friedman 1976]
  - Also known as MSCAN: $: \downarrow : \underline{\quad} : \uparrow : \overline{\quad}$
  - SAF is detected if the address decoder is correct (not all AFs are covered)
    - Theorem: A test detects all AFs if it contains the march elements ⇑(ra,…,wb) and ⇓(rb,…,wa), and the memory is initialized to the proper value before each march element
  - Solid background (pattern)
  - Complexity is 4N

$$\{\Updownarrow(w0);\Updownarrow(r0);\Updownarrow(w1);\Updownarrow(r1)\}$$

Classical Test Algorithms: Checkerboard

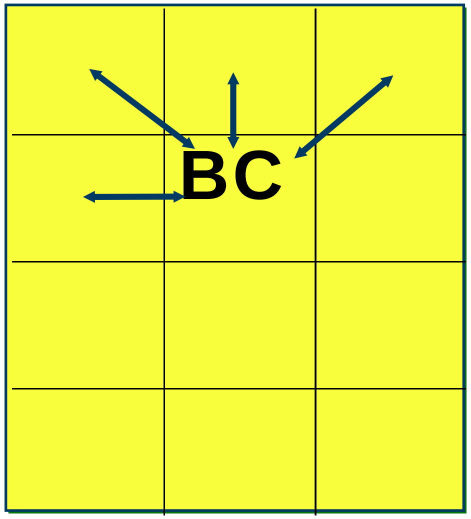
- Checkerboard Algorithm
  - Zero-one algorithm with checkerboard pattern
  - Complexity is 4N
  - Must create true physical checkerboard, not logical checkerboard
  - For SAF, DRF, shorts between cells, and half of the TFs
    - Not good for AFs, and some CFs cannot be detected

1	0	1
0	1	0
1	0	1

Classical Test Algorithms: GALPAT

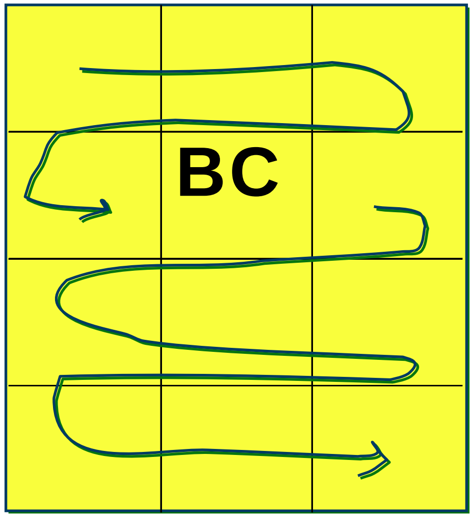
- Galloping Pattern (GALPAT)
  - Complexity is 4N**2—only for characterization
  - A strong test for most faults: all AFs, TFs, CFs, and SAFs are detected and located

```
1. Write background 0;
2. For BC = 0 to N-1
    { Complement BC;
      For OC = 0 to N-1, OC != BC;
        { Read BC;  Read OC; }
      Complement BC; }
3. Write background 1;
4. Repeat Step 2;
```



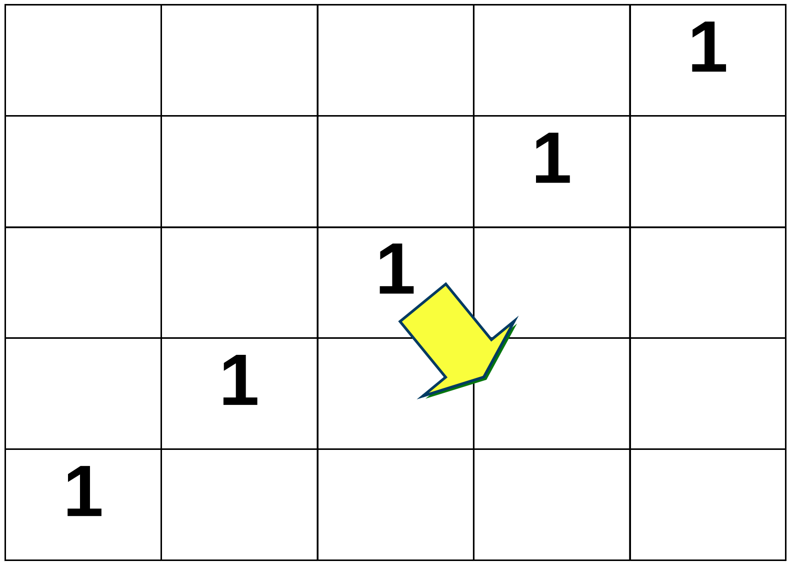
Classical Test Algorithms: WALPAT

- Walking Pattern (WALPAT)
  - Similar to GALPAT, except that BC is read only after all others are read
  - Complexity is 2N**2



Classical Test Algorithms: Sliding

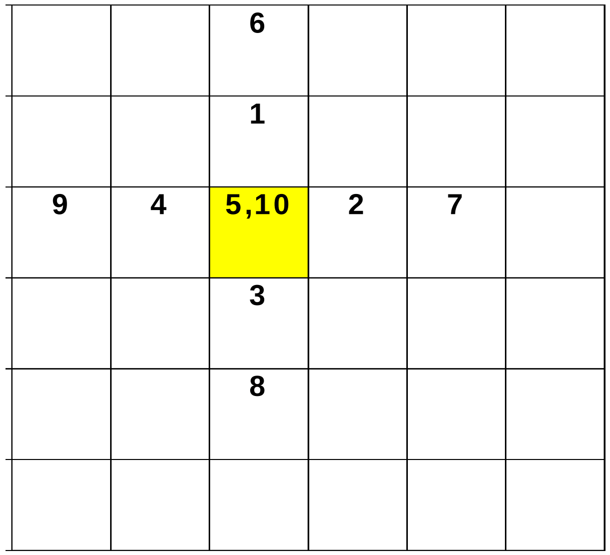
- Sliding (Galloping) Row/Column/Diagonal
  - Based on GALPAT, but instead of shifting a 1 through the memory, a complete diagonal of 1s is shifted
    - The whole memory is read after each shift
  - Detects all faults as GALPAT, except for some CFs
  - Complexity is 4N**1.5



Classical Test Algorithms: Butterfly

- Butterfly Algorithm
  - Complexity is 5NlogN
  - All SAFs and some AFs are detected

```
1. Write background 0;
2. For BC = 0 to N-1
   { Complement BC; dist = 1;
     While dist <= mdist     /* mdist < 0.5 col/row length */
       { Read cell @ dist north from BC;
         Read cell @ dist east from BC;
         Read cell @ dist south from BC;
         Read cell @ dist west from BC;
         Read BC;  dist *= 2; }
     Complement BC; }
3. Write background 1;  repeat Step 2;
```



Classical Test Algorithms: MOVI

- Moving Inversion (MOVI) Algorithm [De Jonge & Smeulders 1976]
  - For functional and AC parametric test
    - ∗ Functional (13N): for AF, SAF, TF, and most CF
    - ∗ ‘ ↓ . __ ↑ ¯¯ . ¯¯ ↓ __ ‘__ ↑ ¯¯ ‘¯¯ ↓ __

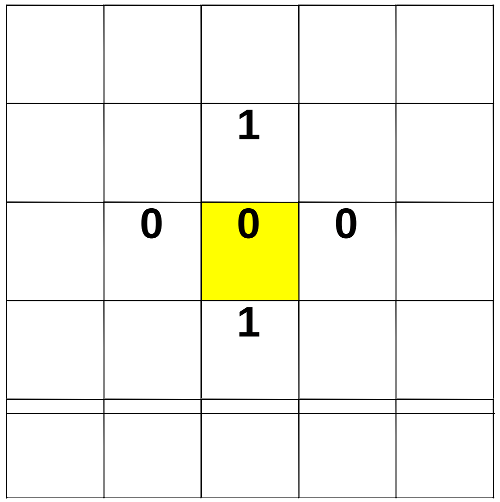
$$\{\Downarrow(w0);\Uparrow(r0,w1,r1);\Uparrow(r1,w0,r0);\Downarrow(r0,w1,r1);\Downarrow(r1,w0,r0)\}$$

    - ∗ Parametric (12NlogN): for Read access time
      - ◊ 2 successive Reads @ 2 different addresses with different data for all 2-address sequences differing in 1 bit
      - ◊ Repeat T2~T5 for each address bit
      - ◊ GALPAT---all 2-address sequences

Classical Test Algorithms: SD

- Surround Disturb Algorithm
  - Examine how the cells in a row are affected when complementary data are written into adjacent cells of neighboring rows
  - Designed on the premise that DRAM cells are most susceptible to interference from their nearest neighbors (eliminates global sensitivity checks)

```
1. For each cell[p,q]   /* row p and column q */
   { Write 0 in cell[p,q-1];
     Write 0 in cell[p,q];
     Write 0 in cell[p,q+1];
     Write 1 in cell[p-1,q];
     Read 0 from cell[p,q+1];
     Write 1 in cell[p+1,q];
     Read 0 from cell[p,q-1];
     Read 0 from cell[p,q]; }
2. Repeat Step 1 with complementary data;
```



Limitations of Classical Tests

- Zero-one and checkerboard algorithms do not have sufficient coverage
- Other algorithms are too time-consuming for large RAMs
  - Test time is the key factor of test cost
  - Complexity ranges from N^2 to NlogN
- Need linear-time test algorithms with small constants
  - March test algorithms
- RAM test time = $\Omega(N)$
  - Cannot do better than linear-time

Simple March Tests

- Zero-One (MSCAN): : ↓ : ___ : ↑ : ‾‾‾
- Modified Algorithmic Test Sequence (MATS) [Nair, Thatte & Abraham 1979]
  - OR-type address decoder fault
    - ∗ : ↓ : ___ ↑ : ‾‾‾ $\{\Updownarrow(w0);\Updownarrow(r0,w1);\Updownarrow(r1)\}$
  - AND-type address decoder fault
    - ∗ : ↑ : ‾‾‾ ↓ : ___ $\{\Updownarrow(w1);\Updownarrow(r1,w0);\Updownarrow(r0)\}$
- MATS+ [Abadir & Reghbati 1983]: : ↓ . ___ ↑ ‘ ‾‾‾ ↓
  - For both OR- & AND-type AFs and SAFs
  - The suggested test for unlinked SAFs

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)\}$$

March Tests: Marching-1/0

- Marching-1/0 [Breuer & Friedman 1976]
  - . ↓ . __ ↑ ‾‾ ’ ‾‾ ↓ __ . ↑ . ‾‾ ↓ __ ’ __ ↑ ‾‾
  - Marching-1: begins by writing a background of 0s, then read and write back complement values (and read again to verify) for all cells (from cell 0 to n-1, and then from cell n-1 to 0), in 7N time
  - Marching-0: follows exactly the same pattern, with the data reversed
  - For AF, SAF, and TF (but only part of the CFs)
  - It is a *complete test*, i.e., all faults that should be detected are covered
  - It however is a *redundant test*, because only the first three march elements are necessary

$$\{\Uparrow(w0); \Uparrow(r0, w1, r1); \Downarrow(r1, w0, r0);$$

$$\Uparrow(w1); \Uparrow(r1, w0, r0); \Downarrow(r0, w1, r1)\}$$

March Tests: MATS++

- MATS++ [Goor 1991]
  - : ↓ . __ ↑ ' ‾‾ ↓ __
  - Also for AF, SAF, and TF
  - Optimized marching-1/0 scheme—complete and irredundant
  - Similar to MATS+, but allow for the coverage of TFs
  - The suggested test for unlinked SAFs & TFs

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0,r0)\}$$

March Tests: March X/C

- March X: : ↓ . __ ↑ ' ‾ ↓ : __
  - Called March X because the test has been used without being published
  - For AF, SAF, TF, & CFin

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$$

- March C [Marinescu 1982]
  - : ↓ . __ ↑ . ‾ ↓ : __ ' __ ↑ ' ‾ ↓ : __
  - For AF, SAF, TF, & all CFs, but semi-optimal (redundant)

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);$$

$$\Updownarrow(r0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$$

March Tests: March C-

- March C- [Goor 1991]
  - : ↓ . __ ↑ . ¯ ↓ '__ ↑ ' ¯ ↓ : __
  - Remove the redundancy in March C
  - Also for AF, SAF, TF, & all CFs
  - Optimal (irredundant)

$$\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$$

- Extended March C-
  - : ↓ . __ ↑ ¯ . ¯ ↓ '__ ↑ ' ¯ ↓ : __
  - Covers SOF in addition to the above faults

$$\{\Updownarrow(w0);\Uparrow(r0,w1,r1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)\}$$

Limitations of March Tests

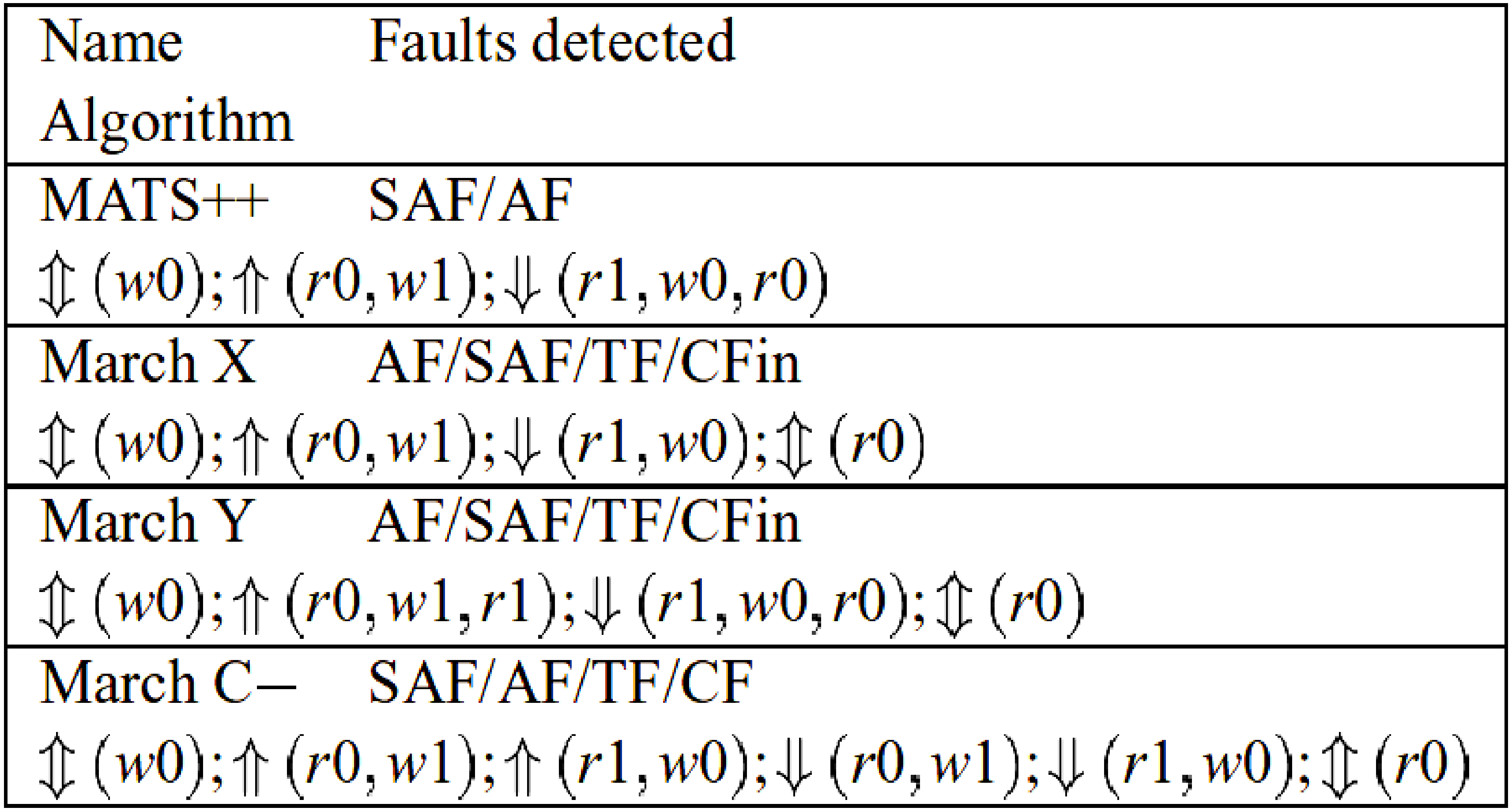
- Limitations
  - Sequential faults in address decoders
  - RF
  - NPSF
    - $\Omega(9N-2)$ for 2-CF [Marinescu 1982]
    - $\Omega(2N\log N+11N)$ for 3-CF [Cockburn 1994]
- Solutions
  - Address sequence variation
    - Hopping
    - Pseudorandom

Exercise

1. Prove that Marching 1/0 is a redundant test for AFs, SAFs, and TFs.
2. Prove that MATS++ is complete and irredundant for AFs, SAFs, and TFs.
3. Determine the march element type in the following procedure. What faults can it detect?

```
Procedure My-March
{ for(i=0; i<n; i++) write 0 in cell[i];
  pause; /* detects retention of 0---typically 100 ms */
  for(i=0; i<n; i++) read cell[i];
  for(i=0 && j=n-1; i<n/2 && j>(n/2-1); i++ && j--)
        { write 1 in cell[i];
          read cell[i];
          write 1 in cell[j];
          read cell[j]; }
  pause; /* detects retention of 1---typically 100 ms */
  for(i=0; i<n; i++) read cell[i];
  for(i=(n/2-1) && j=n/2; i>=0 && j<n; i-- && j++)
        { write 0 in cell[i];
          read cell[i];
          write 0 in cell[j];
          read cell[j]; } }
```

Fault Detection Summary



Name Algorithm	Faults detected
MATS++	SAF/AF
$\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0,r0)$	
March X	AF/SAF/TF/CFin
$\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$	
March Y	AF/SAF/TF/CFin
$\Updownarrow(w0);\Uparrow(r0,w1,r1);\Downarrow(r1,w0,r0);\Updownarrow(r0)$	
March C–	SAF/AF/TF/CF
$\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$	

Comparison of March Tests

	MATS++	March X	March Y	March C-
SAF	√	√	√	√
TF	√	√	√	√
AF	√	√	√	√
SOF	√		√	
CFin		√	√	√
CFid				√
CFst				√

Conclusions

- Functional fault models are commonly used for memories
- New fault models are being proposed to cover new defects and failures in modern memories
- Most classical test algorithms are too time-consuming for large RAMs
- Need linear-time test algorithms with small constants
  - March test algorithms have a complexity of O(N)
  - RAM test time = Ω(N)