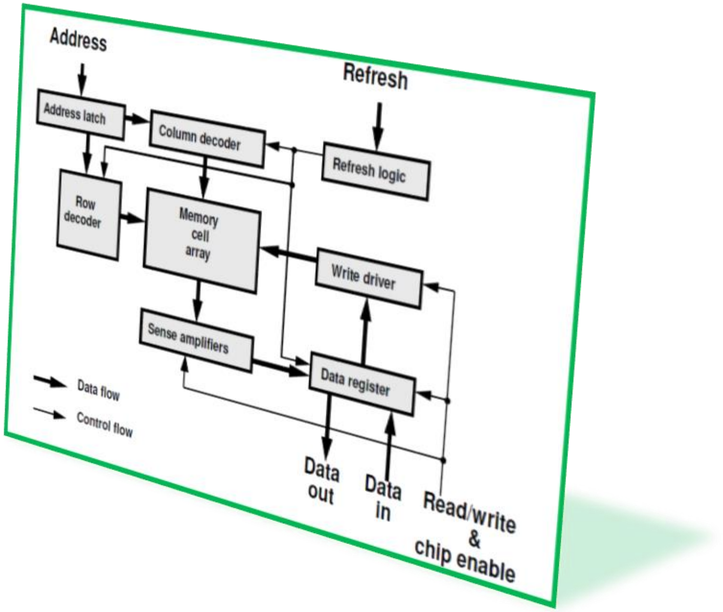
Memory Testing

- Introduction
- Memory Fault Model
- Memory Test Algorithms
  - **Classical algorithms**
  - March algorithms
- Memory Fault Simulation
- Memory Test Generation
- Memory BIST



Memory Test Algorithms

- A *Test algorithm* is a finite sequence of *test elements*
- A *test element* contains a number of
  1. *Memory operations*
     - Read or Write
  2. *Data pattern* (aka. *data background*)
     - Zero or One
  3. *Address sequence*
     - *Ascending* or *Descending*
- *Test (time) Complexity* of test algorithm is expressed in terms of *N*
  - *N* = memory size = number of memory cells
  - Higher complexity means longer test time

Mem. Test Alg. is Different from ATPG Alg.

Memory Test (Time) Complexity

Size	N	$10N$	$N \lg N$	$N^{1.5}$	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	59c
16G	2.8m	28m	1.6h	261d	936c

N = number of memory cells; 100MHz test speed

Linear Complexity Feasible for Production Tests

QUIZ

Q: Which one is correct about memory testing?

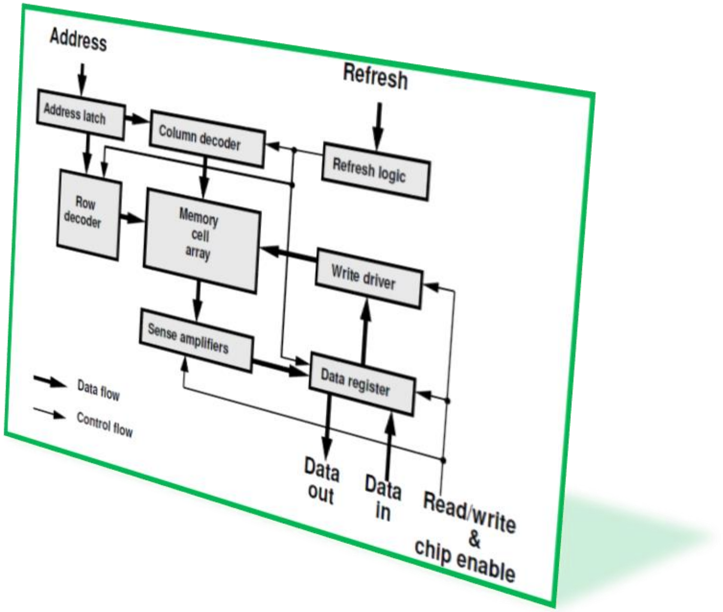
A) Higher complexity means longer CPU time

B) Test algorithm means a finite sequence of test elements, which contain: memory operation, data pattern, and address sequence

C) (*N* log *N*) Complexity is acceptable for large memory

Memory Testing

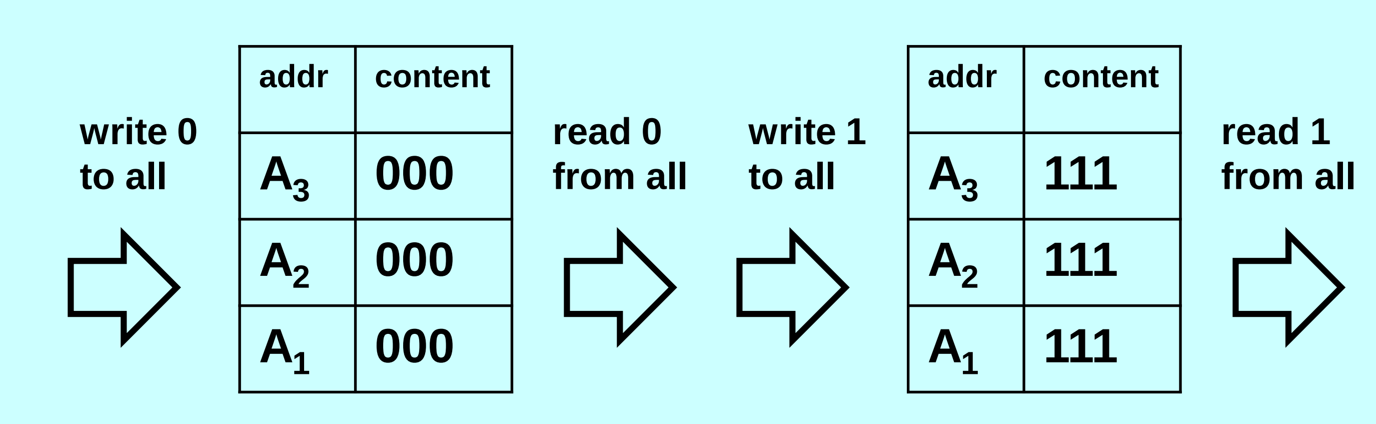
- Introduction
- Memory Fault Model
- **Memory Test Algorithms**
  - **Classical algorithms**
    - **MSCAN**
    - **Checkerboard**
    - **GALPAT**
    - **Butterfly**
  - March algorithms
- Memory Fault Simulation
- Memory Test Generation
- Memory BIST



MSCAN

- aka. *zero-one algorithm*
- Detects all SAF
- Detects <↑/0> TF, not <↓/1>
- Does NOT detect all AF, CF
- Complexity is 4*N*
  - 4 operations each cell

MSCAN
1. Write zero to every cell
2. Read zero from every cell
3. Write one to every cell
4. Read one from every cell



write 0 to all

addr	content
A_3	000
A_2	000
A_1	000

read 0 from all

write 1 to all

addr	content
A_3	111
A_2	111
A_1	111

read 1 from all

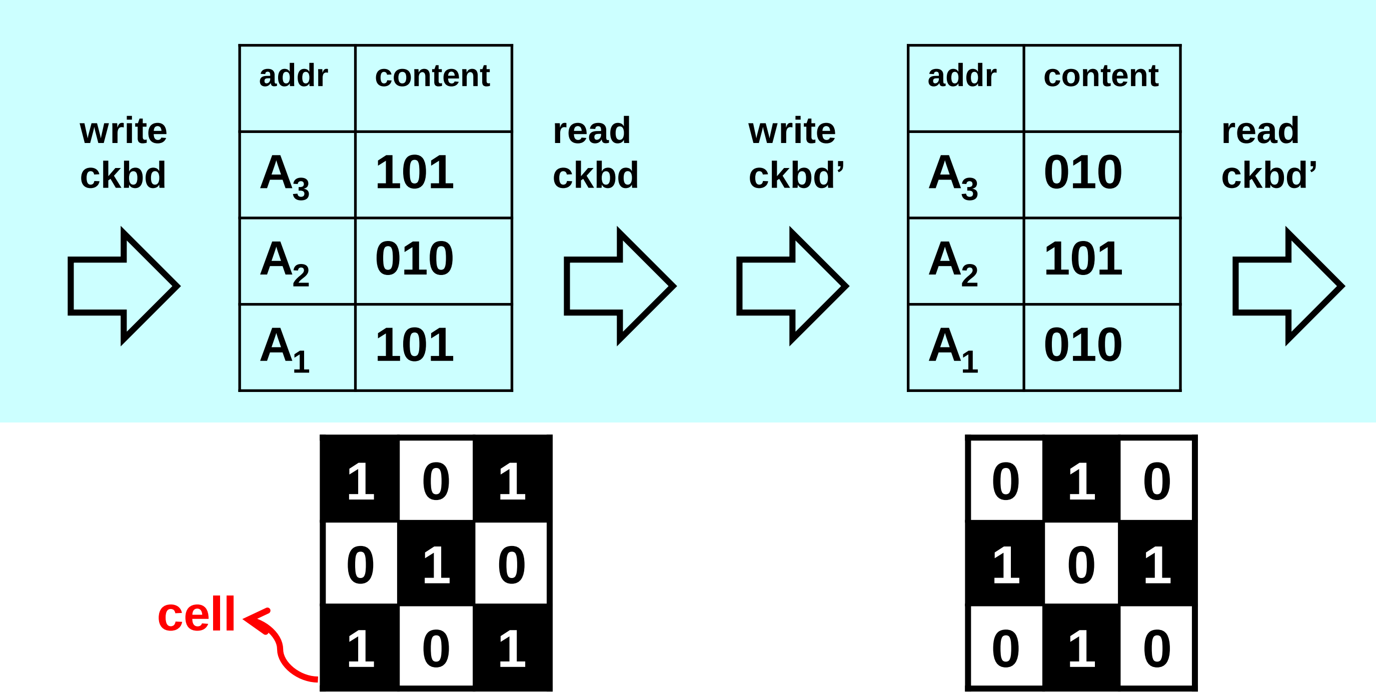
Checkerboard

- Detects all SAF and half TF
- Does NOT detect all AF, CF
- Complexity is *4N*

Same as MSCAN but
Detects *Bridging Fault*

CHECKERBOARD
1. Write chbd pattern to all cells
2. Read ckbd pattern from all cells
3. Write ckbd’ pattern to all cells
4. Read ckbd’ pattern from all cells

ckbd means 01 alternating
ckbd’ is complement of ckbd



write ckbd

addr	content
A_3	101
A_2	010
A_1	101

read ckbd

write ckbd’

addr	content
A_3	010
A_2	101
A_1	010

read ckbd’

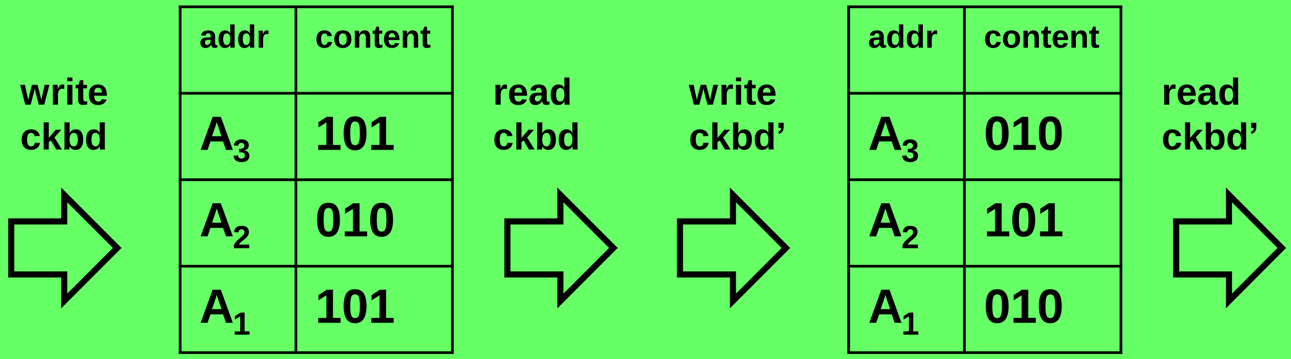
1	0	1
0	1	0
1	0	1

0	1	0
1	0	1
0	1	0

QUIZ

Q1: Can checkerboard detect OR-type AF between A_1 and A_2?
ANS:

Q2: Can checkerboard detect OR-type AF between A_1 and A_3?
ANS:



write ckbd

addr	content
A_3	101
A_2	010
A_1	101

read ckbd

write ckbd’

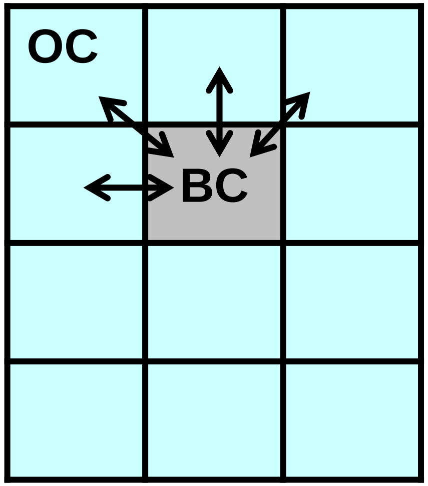
addr	content
A_3	010
A_2	101
A_1	010

read ckbd’

Galloping Test (GALPAT)

- Aka. *Ping-pong test*
- Detects all SAF, TF
- Detects CF, and AF
- Complexity is $4N^2$
  - Line 2: $2N^2$
  - Line 4: $2N^2$

```
GALPAT
1. Write background 0 to all cells;
2. For BaseCell = 0 to N-1
       Complement BC;
       For OtherCell = 0 to N-1, OC != BC;
          Read BC; Read OC;
       Complement BC;
3. Write background 1 to all cells;
4. Repeat Step 2;
```



memory cell array

Too Long! Only for Characterization

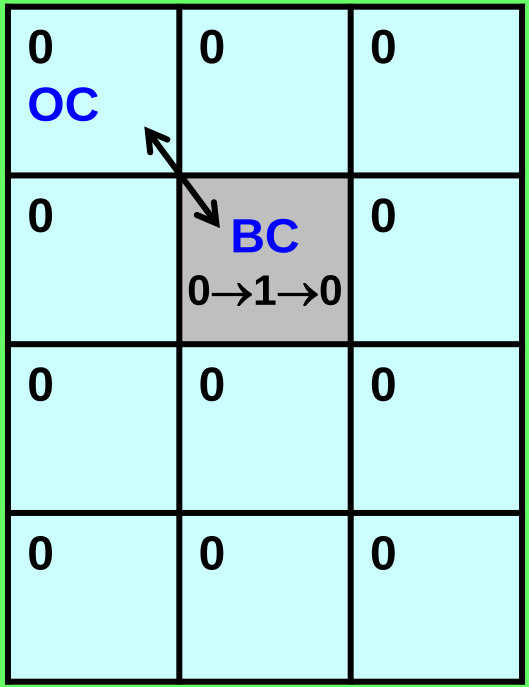
QUIZ

Suppose BC is aggressor and OC is victim.

Q1: Can we detect CF_{st} <1; 0/1>?

Q2: Can we detect CF_{id} <↑; 0/1> ?

Q3: Can we detect CF_{in} <↑ ; ∀/↨ >?

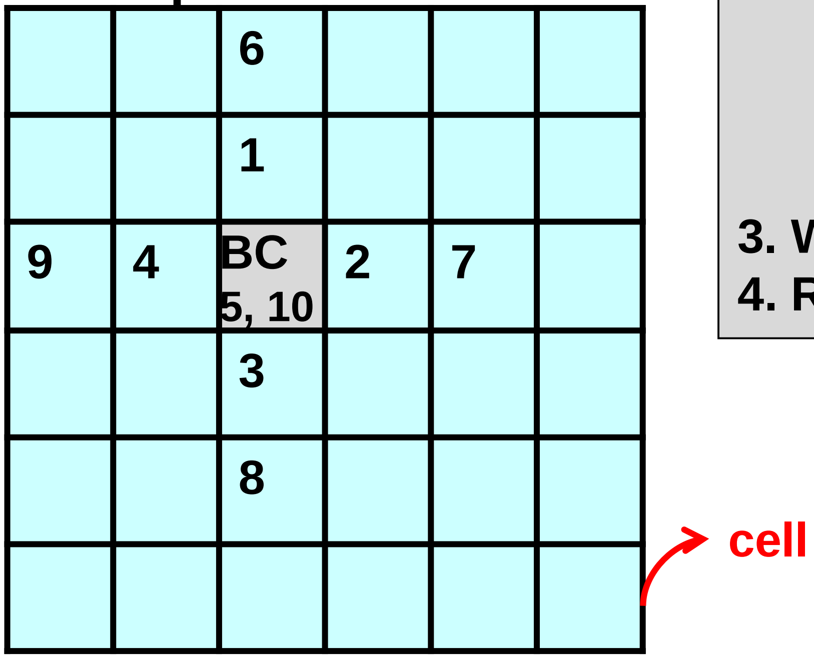


```
GALPAT
1. Write background 0 to all cells;
2. For BaseCell = 0 to N-1
       Complement BC;
       For OtherCell = 0 to N-1, OC != BC;
           Read BC; Read OC;
       Complement BC;
3. Write background 1 to all cells;
4. Repeat Step 2;
```

Butterfly Algorithm

- Detects SAF and TF
- Does not detect all CF, AF
- Complexity is 10 *N* log *N*
  - 5 reads for each *dist*
  - *dist* doubles each loop
  - Repeated in line 4

```
BUTTERFLY     // given MAXDIST < 0.5  col/row size
1. Write background 0;
2. For BaseCell = 0 to N-1
       Complement BC; dist = 1;
       While dist ≤ MAXDIST
           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 = dist * 2;
       Complement BC;
3. Write background 1;
4. Repeat Step 2;
```



memory cell array

Summary

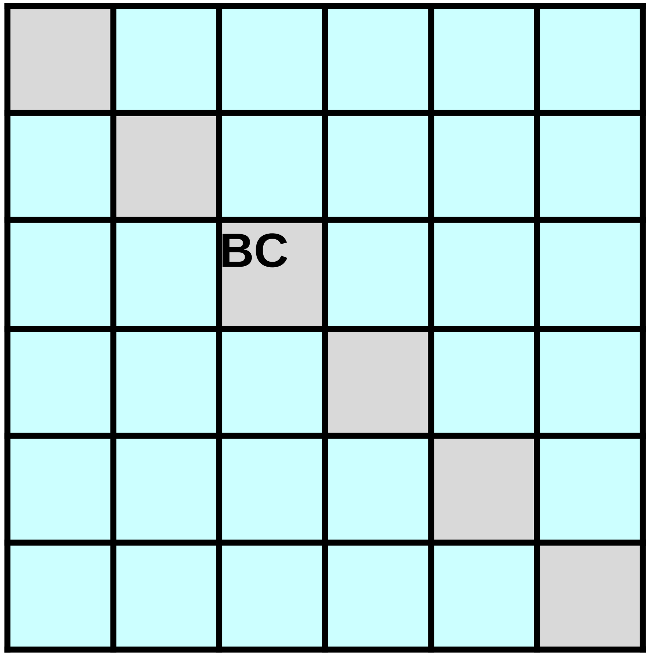
- Test algorithm is a finite sequence of test elements
  - which are: Memory operation, Data pattern, Address sequence
  - Test complexity means test time (not CPU time)
- Four classical test algorithms
  - MSCAN and Checkerboard's fault coverage NOT good
  - GALPAT and Butterfly are too slow
- Need linear-time test algorithms with good fault coverage
  - *March test algorithms* (see 16.3)

	SAF	AF	TF	CF	Complexity
MSCAN	D	-	-	-	$4N$
Checkerboard	D	-	-	-	$4N$
GALPAT	D	D	D	D	$4N^2$
Butterfly	D	-	D	-	$10 N \log N$

D: all detected;
- : not all detected

FFT

- Q: In GALPAT, choosing all cells as base cell is slow. Can we choose only cells on the diagonal line as BC? What is complexity?
- Q: Why do we choose cells in the same column and row of BC in Butterfly test?



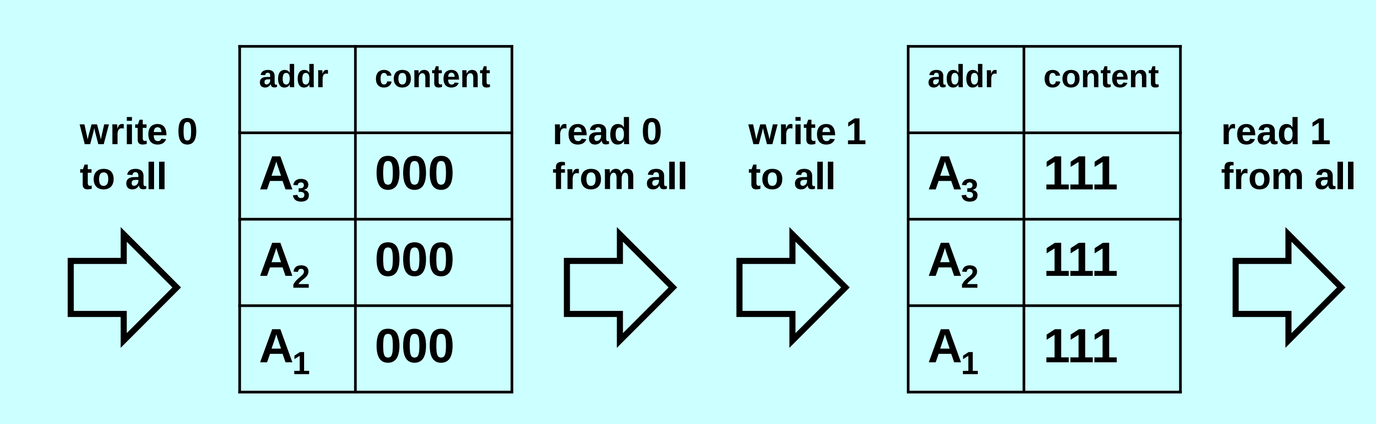
GALPAT-DIAGONAL

```
1. Write background 0 to all cells;
2. For BaseCell = 0 to N-1
       // BC must be on diagonal line
       Complement BC;
       For OtherCell = 0 to N-1, OC != BC;
          Read BC; Read OC;
       Complement BC;
3. Write background 1 to all cells;
4. Repeat Step 2;
```

MSCAN [Breuer & Friedman 1976]

- aka. *zero-one algorithm*
- Detects all SAF
- Detects <↑/0> TF, not <↓/1>
- Does NOT detect all AF, CF
- Complexity is 4*N*

MSCAN
1. Write zero to every cell
2. Read zero from every cell
3. Write one to every cell
4. Read one from every cell



addr	content
A_3	000
A_2	000
A_1	000

addr	content
A_3	111
A_2	111
A_1	111

	SAF	AF	TF	CF	Complexity
MSCAN	D	-	1/2	-	4N

D: all detected
- : not all detected