Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Presenters:

Chenhao Zhang
Yanxin Jiang
Gary Su

Memory Reliability is Breaking

Memory Isolation

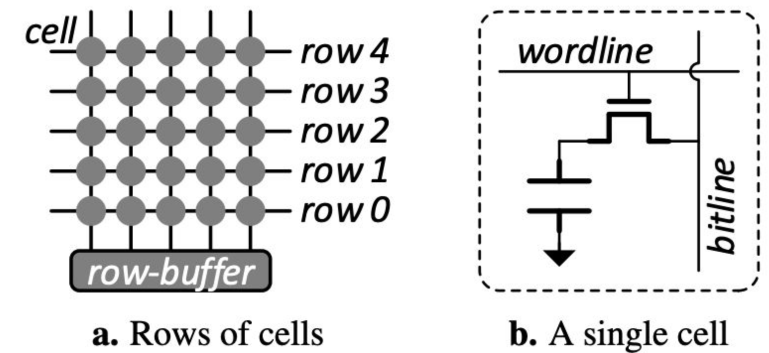
- Key property of a reliable and secure computing system.
- An access to one memory address should not have unintended side effects on data stored in other addresses

DRAM Scaling Impacts

- Enables smaller and closer cells to reduces cost-per-bit of memory.
- Negatively impacts memory reliability → **Increased Disturbance errors**
    - Small cells hold limited charge, reducing noise margin and raising to data loss.
    - Close proximity introduces electromagnetic coupling.
    - Higher process variation increases inter-cell crosstalk in outlier cells.

DRAM Organization

- DRAM is a 2D array of cells:
  - Each cell = capacitor + access transistor
  - Stores data as charged (1) or discharged (0)
- Cells are accessed via:
  - Wordline (row) → activates a row
  - Bitline (column) → reads/writes data
- When a row is activated (wordline high):
  - Cells connect to bitlines via access transistors
  - Sense amplifiers (row buffer) read & restore data
  - Row buffer serves all subsequent accesses
- DRAM hierarchy: Rows → Banks → Ranks
  - Multiple banks enable parallel access



a. Rows of cells **b.** A single cell

Figure 1. DRAM consists of cells

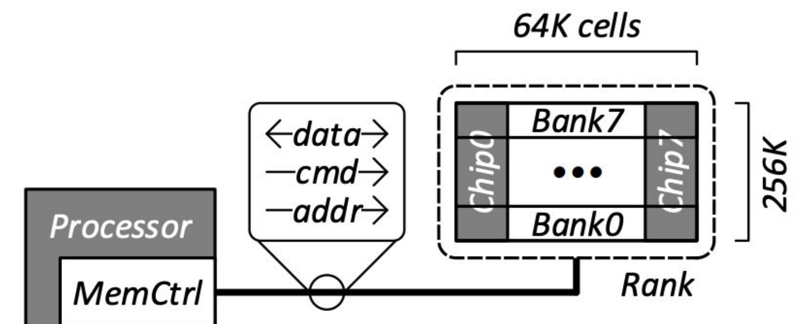


Figure 2. Memory controller, buses, rank, and banks

Carnegie Mellon University

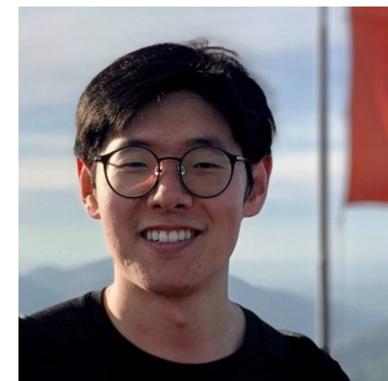
DRAM Access & Refresh

Operation	*Command*	*Address(es)*
1. Open Row	ACTIVATE (ACT)	Bank, Row
2. Read/Write Column	READ/WRITE	Bank, Column
3. Close Row	PRECHARGE (PRE)	Bank
Refresh (Section 2.4)	REFRESH (REF)	—

Table 1. DRAM commands and addresses [34]

- DRAM access follows three steps:
  - Activate (ACT): open a row → load data into row buffer
  - Read/Write: access columns from row buffer
  - Precharge (PRE): close row before accessing another
- Memory controller enforces timing constraints (e.g., tRC)
- DRAM cells leak charge over time → require periodic refresh (~64 ms)

“Flipping Bits in Memory Without Accessing Them”



- **Yoongu Kim** - CMU PhD, now Core Developer at Hudson River Trading
- **Ross Daly** - Stanford PhD, now Self Employed
- **Jeremie Kim** - CMU PhD, now CPU Research Scientist at Apple
- **Chris Fallin** - CMU PhD, Senior Architect at F5
- **Ji Hye Lee** - Researcher at CMU
- **Donghyuk Lee** - CMU PhD, now Research Scientist at NVIDIA Research
- **Chris Wilkerson** - CMU Master, now Senior Principal Engineer at SiMa.ai
- **Konrad Lai** - Senior Principal Engineer at Intel
- **Onur Mutlu** - UT Austin PhD, CMU prof, now ETH

Problem Setup & Key Findings

- Disturbance errors are widespread in modern DRAM
  - Observed in 110 / 129 modules (836 chips)
  - Especially common in recent technology generations
- Errors can be triggered by simple memory accesses
  - As few as 139K repeated row activations can cause bit flips
  - No special privileges required (user-level program)
- Key observations:
  - Disturbable cells exist in most modules
  - Up to 1 in 1.7K cells can be affected

Mechanics of Disturbance Errors

- Disturbance errors arise from unintended interactions between cells
- **Repeated row activation** (wordline toggling):
  - Accelerates charge leakage in nearby rows
  - Cells fail to retain data within refresh interval (~64 ms)
- Possible physical causes:
  - **Electromagnetic coupling** between wordlines
  - Charge leakage through defects (**bridging**)
  - **Hot-carrier injection effects**
- Errors occur when cumulative interference becomes strong enough

Real System Demonstration

- Demonstrated disturbance errors on real Intel/AMD systems (DDR3)
- **Key idea:**
  - Repeatedly access two addresses in the **same bank, different rows**
  - Forces continuous **row activation (ACT) and precharge (PRE)**
- **Implementation:**
  - Read data from DRAM (`mov` instructions)
  - Flush cache (`clflush`) to force DRAM access
  - Use `mfence` + loop to repeat millions of times

```
1 code1a:
2   mov (X), %eax
3   mov (Y), %ebx
4   clflush (X)
5   clflush (Y)
6   mfence
7   jmp code1a
```

a. Induces errors

Carnegie Mellon University

Real System Demonstration

- **Result:**
  - Repeated row activations cause **bit flips in adjacent rows**
  - Observed thousands of errors across multiple platforms
- **Control experiment:**
  - Accessing the same row repeatedly does not cause errors
  - Confirms row toggling is root cause
- **Violates memory invariants:**
  - Reads should not modify data
  - Writes should only affect target address
- **Security implication:**
  - Can corrupt other memory pages
  - Potential for attacks

```
1 code1b:
2   mov (X), %eax
3   clflush (X)
4
5
6   mfence
7   jmp code1b
```

b. Does not induce errors

Bit-Flip	Sandy Bridge	Ivy Bridge	Haswell	Piledriver
‘0’ → ‘1’	7,992	10,273	11,404	47
‘1’ → ‘0’	8,125	10,449	11,467	12

Table 2. Bit-flips induced by disturbance on a 2GB module

Experiment Methodology

Goal: Systematically characterize DRAM disturbance errors

- **FPGA-based testing platform** under precisely controlled access timing, refresh behavior, and data patterns
- **TestBulk** identifies all disturbed cells in a module
  TestEach identifies the victim cells induced by one aggressor row

```
1  TestBulk(AI, RI, DP)
2    setAI(AI)
3    setRI(RI)
4    N ← (2 × RI)/AI
5
6    writeAll(DP)
7    for r ← 0···ROW_MAX
8      for i ← 0···N
9        ACT r^th row
10       READ 0^th col.
11       PRE r^th row
12   readAll()
13   findErrors()
```

a. Test all rows at once

```
1  TestEach(AI, RI, DP)
2    setAI(AI)
3    setRI(RI)
4    N ← (2 × RI)/AI
5
6    for r ← 0···ROW_MAX
7      writeAll(DP)
8      for i ← 0···N
9        ACT r^th row
10       READ 0^th col.
11       PRE r^th row
12     readAll()
13     findErrors()
```

b. Test one row at a time

- **Key parameters:**
  - AI (activation interval) controls how frequently a row is toggled — default 55ns (max row-toggling rate without violating tRC)
  - RI (refresh interval) controls how often the module is refreshed — default 64ms (default DDR3 RI)
  - DP (data pattern) sets the initial data written before testing — Rowstripe & ~Rowstripe & ...
- **Scale of study:** The authors test 129 DDR3 modules / 972 DRAM chips from three major manufacturers.

Results: Widespread effects

For all 129 modules:

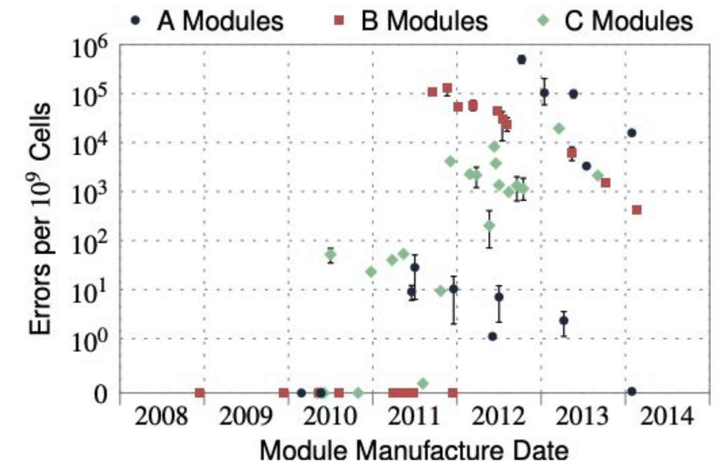
```
TestBulk(55ns, 64ms, RowStripe)
TestBulk(55ns, 64ms, ~RowStripe)
```

110 / 129 DDR3 modules(across 3 manufactures) exhibit disturbance errors

Date Boundary (2010-2011): Vulnerability of modules strongly correlates with newer manufacture dates

From Table 3: these boundaries coincide with die-version upgrades; technology migration (old & reliable → new & unreliable)

→ disturbance errors are a relatively recent but widespread phenomenon



Results: root cause & trigger condition

- **Mini-test 1**: access-pattern dependence test
  - `(open-read-close)^N / (open-write-close)^N → Yes`
    `open-read^N-close / open-write^N-close → No`
  - Only repeated row opening/closing induces errors
  - Root cause: repeated toggling of the same wordline

- **Mini-test 2&3:** RI & AI sweep
  - RI is swept from 10 to 128 ms; AI is swept from 55-500ns
  - Shorter RI → less time to leak charge & row is opened less often → fewer errors
  - Longer AI → row is opened less often → fewer errors

- **Mini-test 4**: Data pattern
  - Errors are not determined by hammering alone but also data pattern
  - RowStripe & Checkered can cause far more errors
  - Disturbance error is not simple two-cell coupling → multi-cell phenomenon

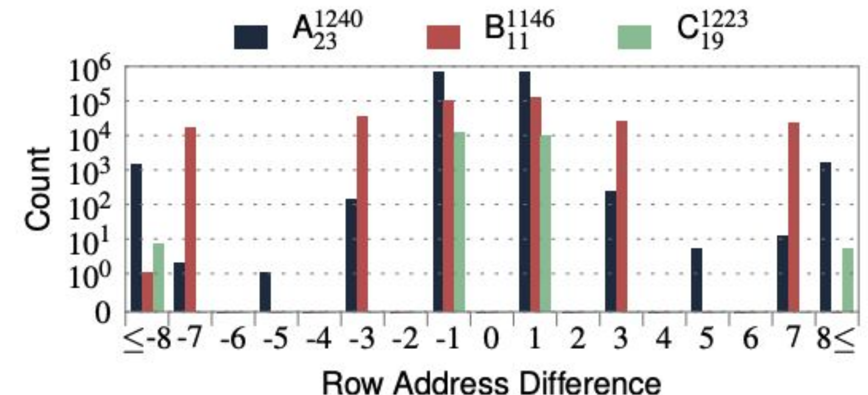
Module	TestBulk(*DP*) + TestBulk(~*DP*)			
	Solid	RowStripe	ColStripe	Checkered
A_{23}	112,123	**1,318,603**	763,763	934,536
B_{11}	12,050	**320,095**	9,610	302,306
C_{19}	57	20,770	130	**29,283**

Results: Errors are localized

- Aggressor & Victim
  - Locate with: `TestEach(55ns, 64ms, RowStripe & ~RowStripe)`
  - Aggressor-row fraction: 100% / 99.96% / 47.04% (A23 / B11 / C19)

- Spatial Locality
  - Victim cells from one aggressor row are usually confined to ≤ 2 rows
  - Figure 9 shows strong peaks at ±1: Victims are likely the adjacent rows of the aggressors
  - Non-adjacent cases may come from row remapping or vendor-specific row mapping

- Certain direction is preferred by the error
  - Errors usually occur in one preferred direction (1 → 0)
  - During disturbance: charge is being lost from charged cells



1. **Errors are Mostly Repeatable: (3 modules, 10 iters)**
   - Most victim cells were repeat offenders, meaning that they had an error in every iteration: 78.3%, 74.4%, and 73.2%.
   - some victim cells had an error in just a single iteration: 3.14%, 4.86%, and 4.76%
2. **Victim cells is not weak cells**
   - weak cell: the cells with the shortest retention times
   - 1M weak cells for each module (984K, 993K, and 1.22M), which is on par with the number of victim cells.
   - only few weak cells were also victim cells: 700, 220, and 19.
3. **Not strongly affected by the temperature**
   - under 70±2.0 celsius, +10.2%, -0.553%, and +1.32% errors compared with 50 celsius.
   - under 30±2.0 celsius, -14.5%, +2.71%, and -5.11% errors compared with 50 celsius.

Potential Solutions

1. **Make better chips**
   - improve circuit design
   - may get worse when cells become smaller and more vulnerable in the future.
2. **Correct errors**
   - deploy DRAM ECC, leads to extra 12.5% capacity overhead
3. **Refresh all rows frequently**
   - Disturbance errors can be eliminated for sufficiently short refresh interval
   - frequent refreshes degrade performance and energy-efficiency. 1.4–4.5% of time refreshing → 11.0–35.0% of time refreshing (8.2ms)
4. **Retire cells (manufacturer)**
   - manufacturer could identify victim cells and re-map them to spare cells
   - take several days or more to search victim cells

5. Retire cells (end-user)
- end-users themselves could test the modules and employ system-level techniques for handling DRAM reliability problems
- disable faulty addresses, re-map faulty addresses to reserved addresses or refresh faulty addresses more frequently
- 1st and 2nd not effective when every row in the module is a victim row
- 3rd not effective it always refreshes the victim rows more frequently even when the module is not being accessed

6. Identify "hot" rows and refresh neighbors
- identify frequently opened rows and refresh only their neighbors
- a lot of designs rely heavily on hash functions → hash collision → many rows are falsely flagged as being "hot"

probabilistic adjacent row activation (PARA)

Key Idea: every time a row is opened and closed, one of its adjacent rows is also refreshed with some low probability.
If one particular row happens to be opened and closed repeatedly, then it is statistically certain that the row's adjacent rows will eventually be opened as well.

Advantage: PARA is stateless → no extra hardware data-structures

Implementation: row is closed → controller throw a coin with head probability p (p << 1).
coin is the head → open either of two adjacent rows with equal probability (overall probability p/2)
PARA cannot completely eliminate errors, but it keeps the error rate extremely low

Adjacency Information: controller must know which rows are physically adjacent to each other
Use mapping function to map logical rows to physical rows. This map can be stored in ROM.

Performance: p = 0.005, -0.197% on throughput during the simulated duration of 100ms.
max -0.745% for single benchmark.

Duration	N_{th}=50K	N_{th}=100K	N_{th}=200K
64*ms*	1.4×10^{-11}	1.9×10^{-22}	3.6×10^{-44}
1 *year*	6.8×10^{-3}	9.4×10^{-14}	1.8×10^{-35}

Table 7. Error probabilities for PARA when p=0.001

What Did the Paper Get Right?

State the 3 most important things the paper says.

These could be some combination of the motivations, observations, interesting parts of the design, or clever parts of the implementation.

Discussion Summary Question #2

What Did the Paper Get Wrong?

Describe the paper's single most glaring deficiency.

Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

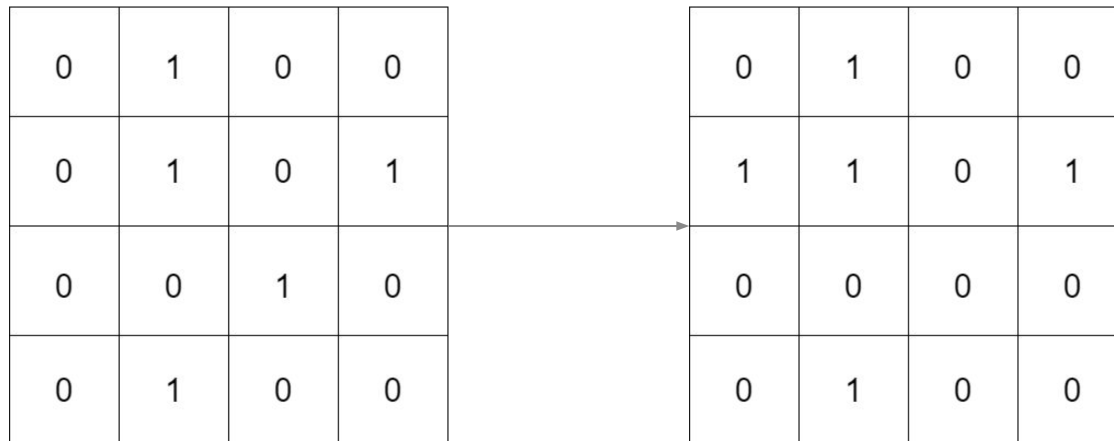
The PARA is not adopted within industrial because it functions as a probabilistic rather than a deterministic system. While PARA effectively minimizes error rates, industrial standards demand deterministic reliability to ensure data integrity and predictable outcomes."

PARA made refreshes happen at random intervals, it could lead to unpredictable performance spikes,

PARA made valuable observations regarding DRAM disturbance errors, providing strong inspiration for future DRAM design.

Extra content: ECC DRAM (a.k.a. SECDED/Hamming Code)

easy to detect 1 bit flip and 2 bit flip. cannot detect more



Extra content: ECC DRAM

Q: how do we detect this?

A: Parity check!

bit 0 represent the whole parity of these 16 bit example

bit 1 represent column 1 and 3 (starts from 0)

bit 2 represent column 2 and 3

bit 4 represent the row 1 and 3

bit 8 represent row 2 and 3

0	1	0	0
1	1	0	1
0	0	0	0
0	1	0	0

Extra content: ECC DRAM

0	1	0	0
1	1	0	1
0	0	0	0
0	1	0	0

0	1	0	0
1	1	0	1
0	0	0	0
0	1	0	0

0	1	0	0
1	1	0	1
0	0	0	0
0	1	0	0

Then we know, error is at column 2

Extra content: ECC DRAM

0	1	0	0
1	1	0	1
0	0	0	0
0	1	0	0

0	1	0	0
1	1	0	1
0	0	0	0
0	1	0	0

0	1	0	0
1	1	0	1
0	0	0	0
0	1	0	0

Then we know, error is at row 2
then we figure out that error! row 2 and column 2!

Extra content: ECC DRAM

Why it is good for DRAM?

ok now we start with parity check from bit 1, 2, 4 and 8 **in reverse order**

if it passes the check, then it means that it is 0, else, it will represent a 1.

bit 8	bit 4	bit 2	bit 1
1	0	1	0

1010 = 10(in base 10) and error is exactly on it!

0	1	0	0
1	1	0	1
0	0	0	0
0	1	0	0

Extra content: ECC DRAM

Now some magic on math

we extract all of the 1's position

1st 1: 0001
2nd 1: 0100
3rd 1: 0101
4th 1: 0111
5th 1: 1101

we xor all of them
$0001 \oplus 0100 \oplus 0101 \oplus 0111 \oplus 1101 = 1010$!

0	1	0	0
1	1	0	1
0	0	0	0
0	1	0	0

0000	0001	0010	0011
0100	0101	0110	0111
1000	1001	1010	1011
1100	1101	1110	1111