Flipping Bits in Memory Without Accessing Them

DRAM Disturbance Errors

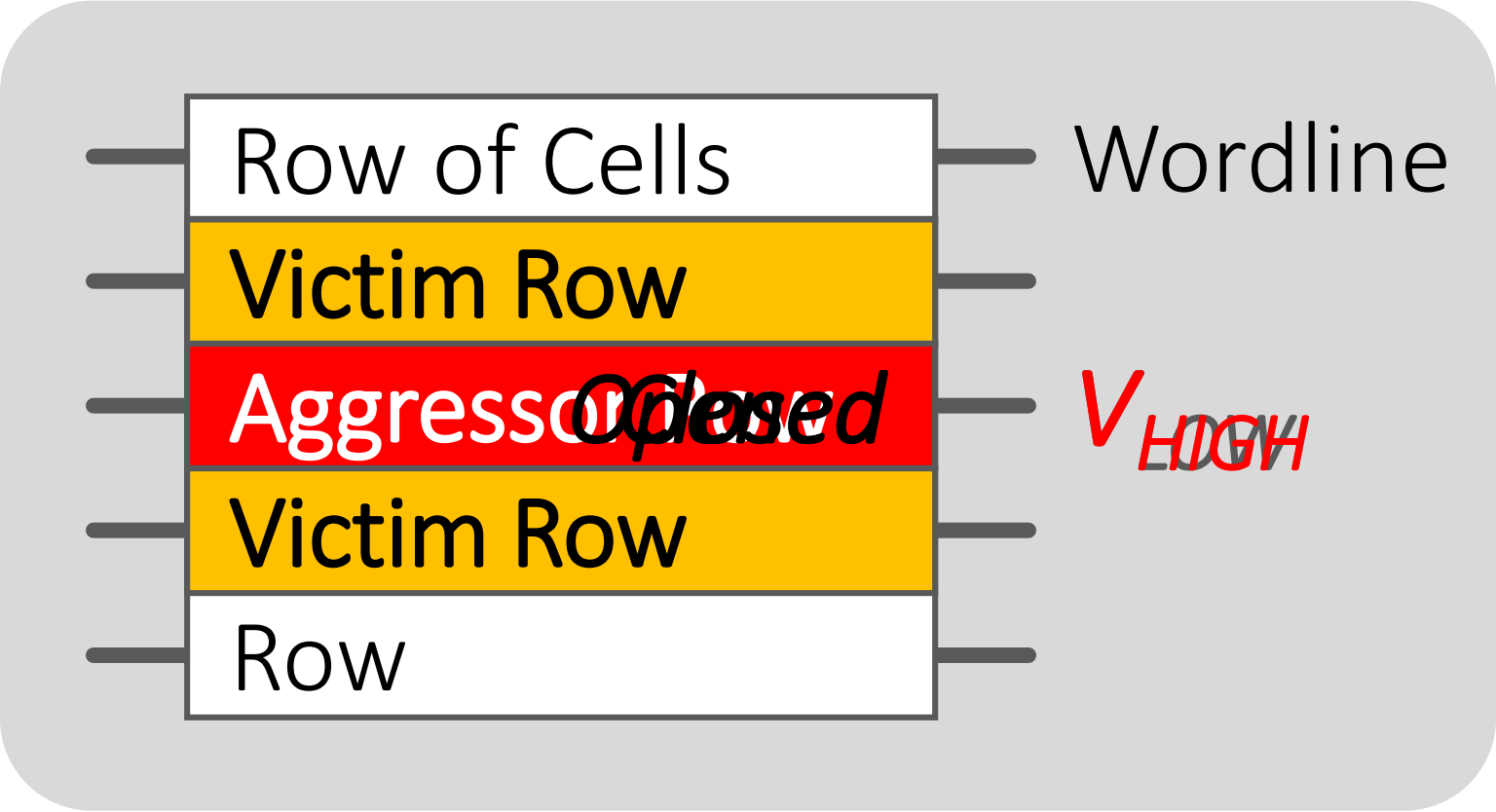
Yoongu Kim

Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee,

Donghyuk Lee, Chris Wilkerson, Konrad Lai, Onur Mutlu

Carnegie Mellon SAFARI intel

DRAM Chip



*Repeatedly opening and closing a row induces **disturbance errors** in adjacent rows*

Quick Summary of Paper

- *We expose the **existence** and **prevalence** of disturbance errors in DRAM chips of today*
  - 110 of 129 modules are vulnerable
  - Affects modules of 2010 vintage or later

- *We characterize the **cause** and **symptoms***
  - Toggling a row accelerates charge leakage in adjacent rows: *row-to-row coupling*

- *We prevent errors using a **system-level** approach*
  - Each time a row is closed, we refresh the charge stored in its adjacent rows with a low probability

1. Historical Context

2. Demonstration (Real System)

3. Characterization (FPGA-Based)

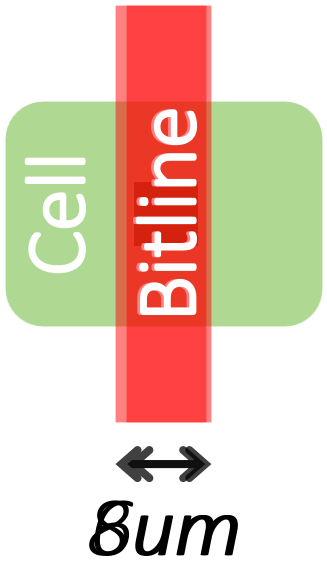
4. Solutions

A Trip Down Memory Lane

1968 IBM's patent on DRAM

1971 Intel commercializes DRAM (Intel 1103)

- Suffered bitline-to-cell coupling



"... this big fat metal line with full level signals running right over the storage node (of cell)."

*– **Joel Karp** (1103 Designer)*
Interview: Comp. History Museum

2013

2014

A Trip Down Memory Lane

1968 IBM's patent on DRAM

1971 Intel commercializes DRAM (Intel 1103)

- Suffered bitline-to-cell coupling

2010 Earliest DRAM with row-to-row coupling

2013 We observe row-to-row coupling

2014 Intel's patents mention *"Row Hammer"*

Lessons from History

- *Coupling in DRAM is not new*
  - Leads to *disturbance errors* if not addressed
  - Remains a major hurdle in DRAM scaling

- *Traditional efforts to contain errors*
  - Design-Time: Improve circuit-level isolation
  - Production-Time: Test for disturbance errors

- *Despite such efforts, disturbance errors have been slipping into the field since 2010*

1. Historical Context

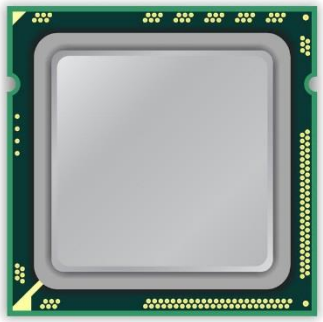
2. Demonstration (Real System)

3. Characterization (FPGA-Based)

4. Solutions

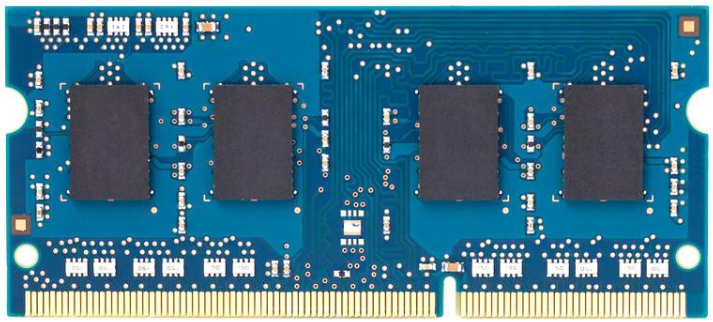
How to Induce Errors

x86 CPU



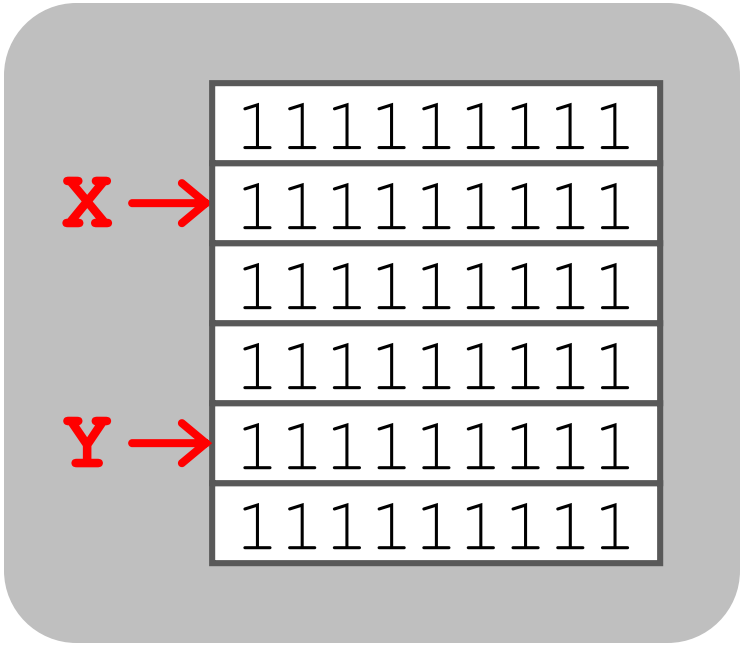
DDR3

DRAM Module



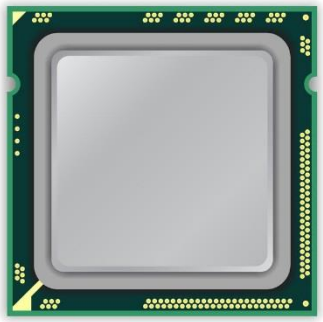
1. Avoid ***cache hits***
   – Flush **X** from cache

2. Avoid ***row hits*** to **X**
   – Read **Y** in another row



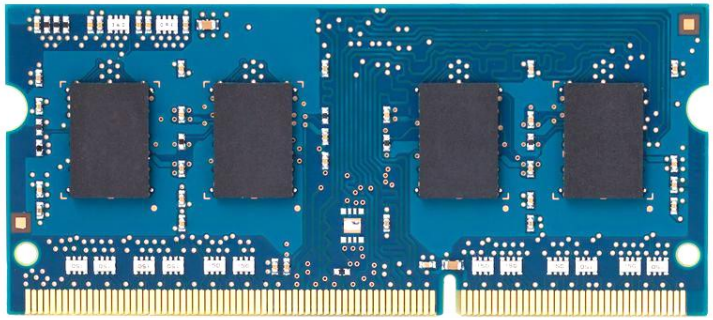
How to Induce Errors

x86 CPU

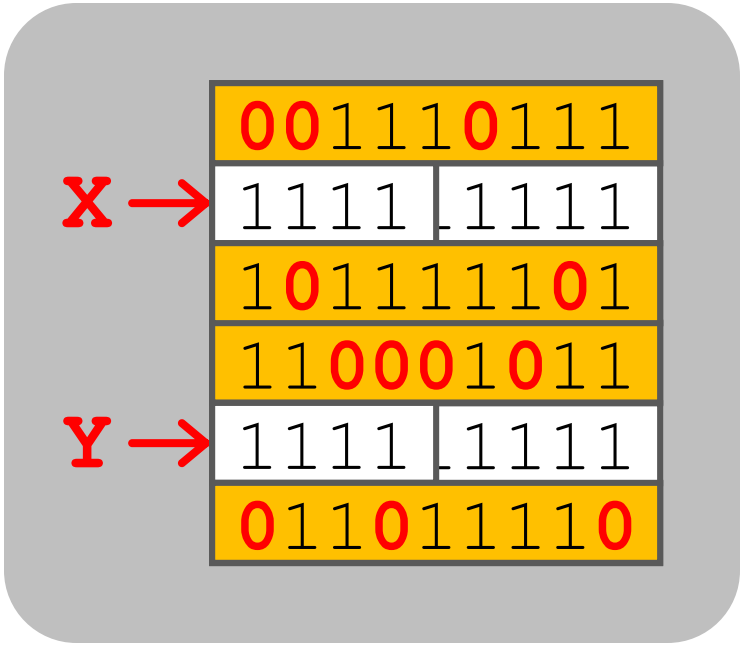


DDR3

DRAM Module



```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```



Number of Disturbance Errors

CPU Architecture	Errors	Access-Rate
Intel Haswell (2013)	22.9K	12.3M/sec
Intel Ivy Bridge (2012)	20.7K	11.7M/sec
Intel Sandy Bridge (2011)	16.1K	11.6M/sec
AMD Piledriver (2012)	59	6.1M/sec

- *In a more controlled environment, we can induce as many as ten million disturbance errors*
- *Disturbance errors are a serious reliability issue*

Security Implications

- *Breach of memory protection*
  - OS page (4KB) fits inside DRAM row (8KB)
  - Adjacent DRAM row → Different OS page
- *Vulnerability: disturbance attack*
  - By accessing its own page, a program could corrupt pages belonging to another program
- *We constructed a proof-of-concept*
  - Using only user-level instructions

Mechanics of Disturbance Errors

- *Cause 1: Electromagnetic coupling*
  - Toggling the wordline voltage briefly increases the voltage of adjacent wordlines
  - Slightly opens adjacent rows → Charge leakage
- *Cause 2: Conductive bridges*
- *Cause 3: Hot-carrier injection*

Confirmed by at least one manufacturer

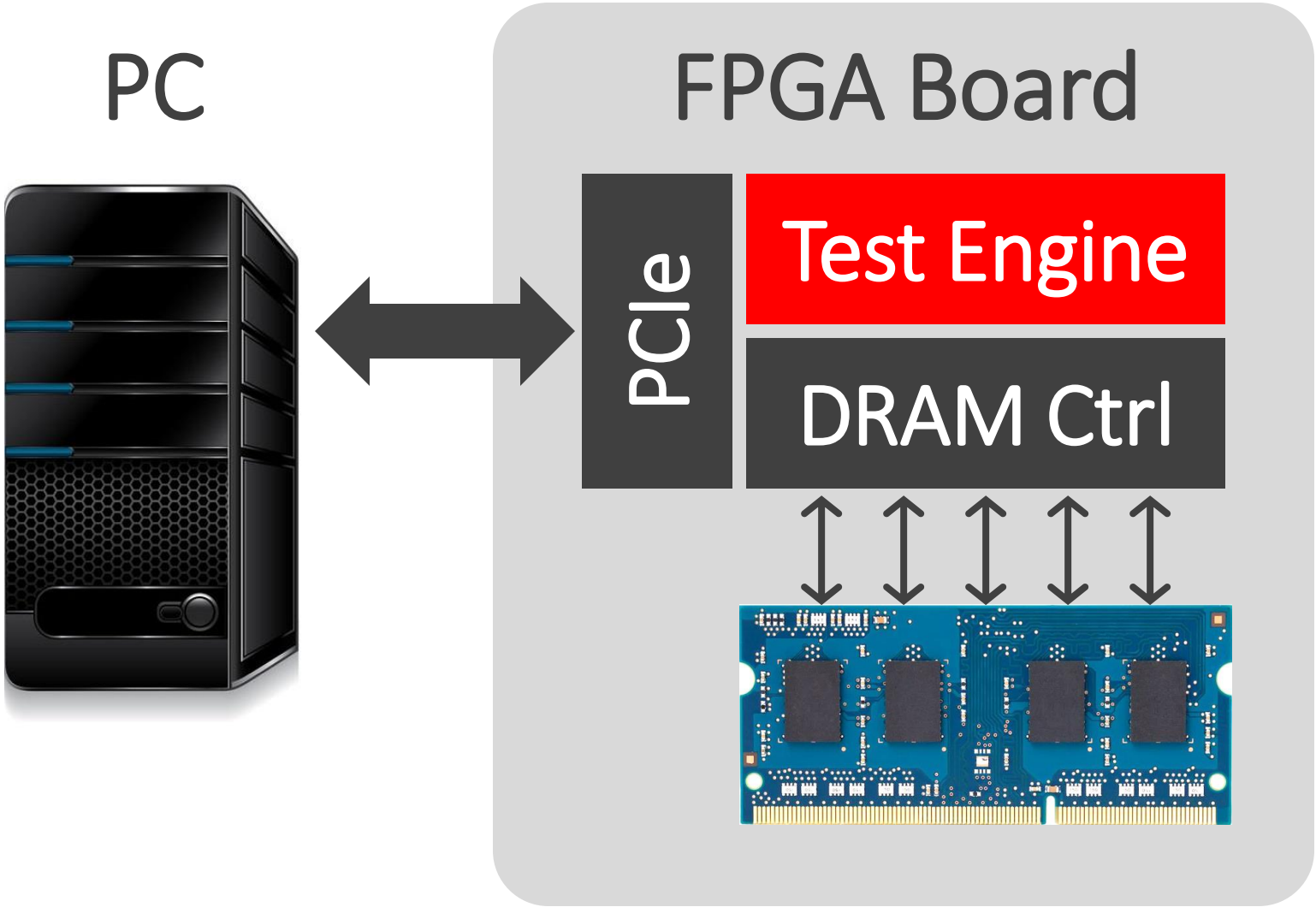
1. Historical Context

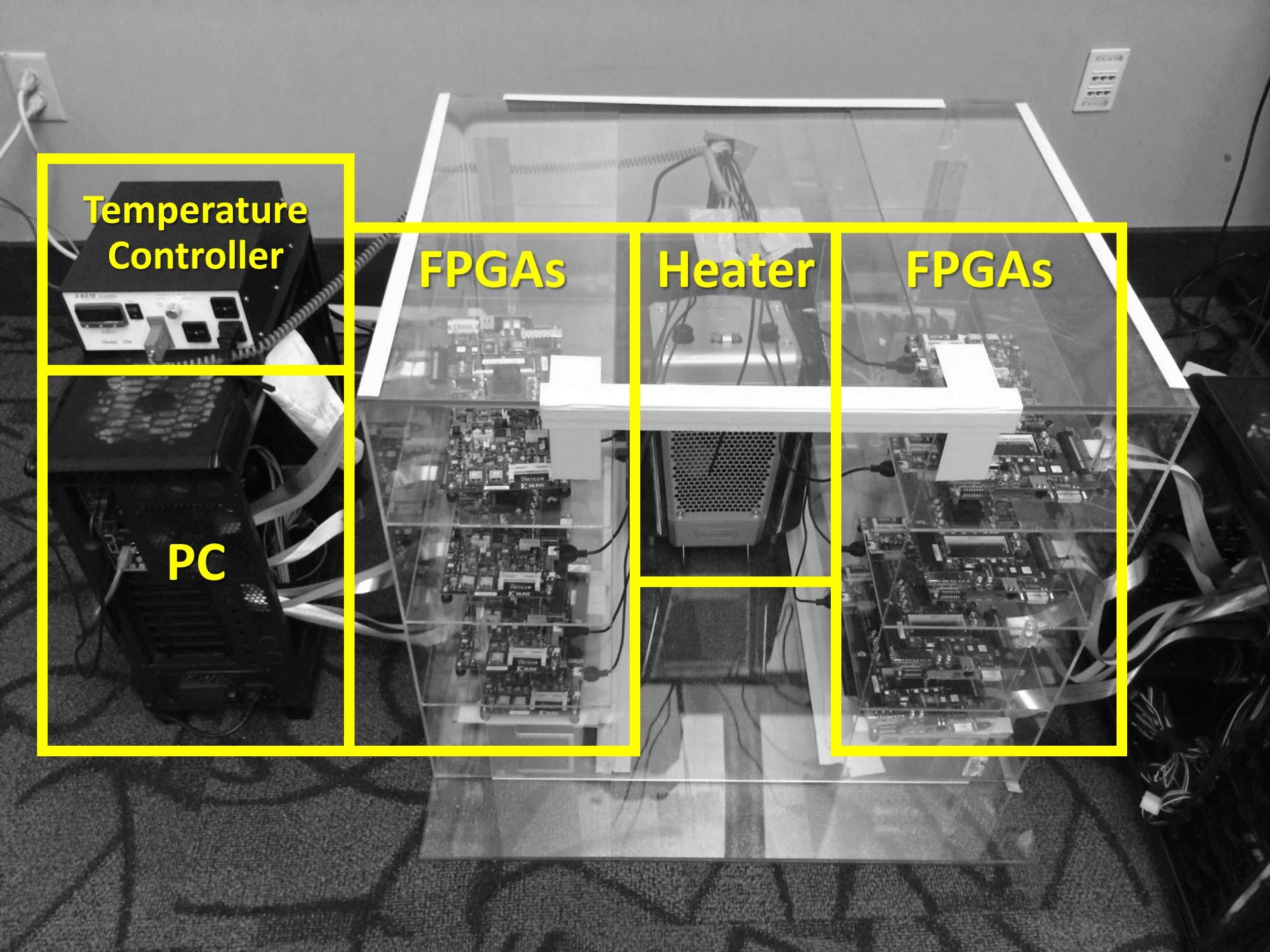
2. Demonstration (Real System)

3. Characterization (FPGA-Based)

4. Solutions

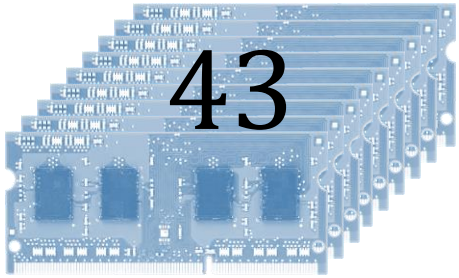
Infrastructure



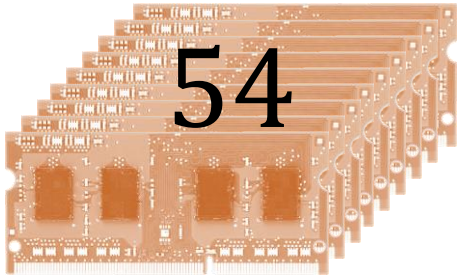

Temperature Controller
PC
FPGAs
Heater
FPGAs

Tested DDR3 DRAM Modules

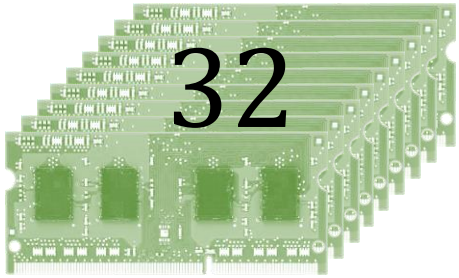
Company A



Company B



Company C

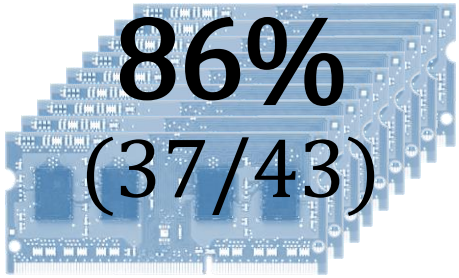
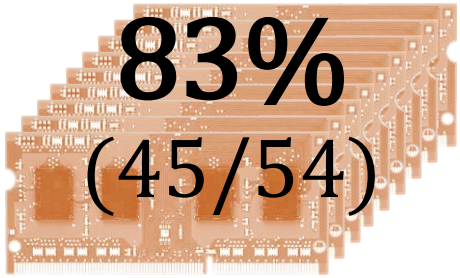
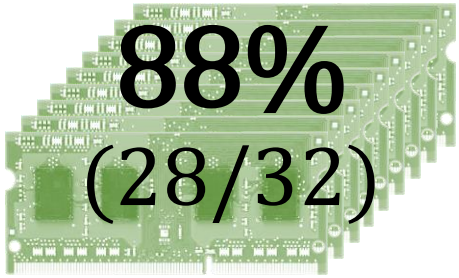


- Total: 129
- Vintage: 2008 – 2014
- Capacity: 512MB – 2GB

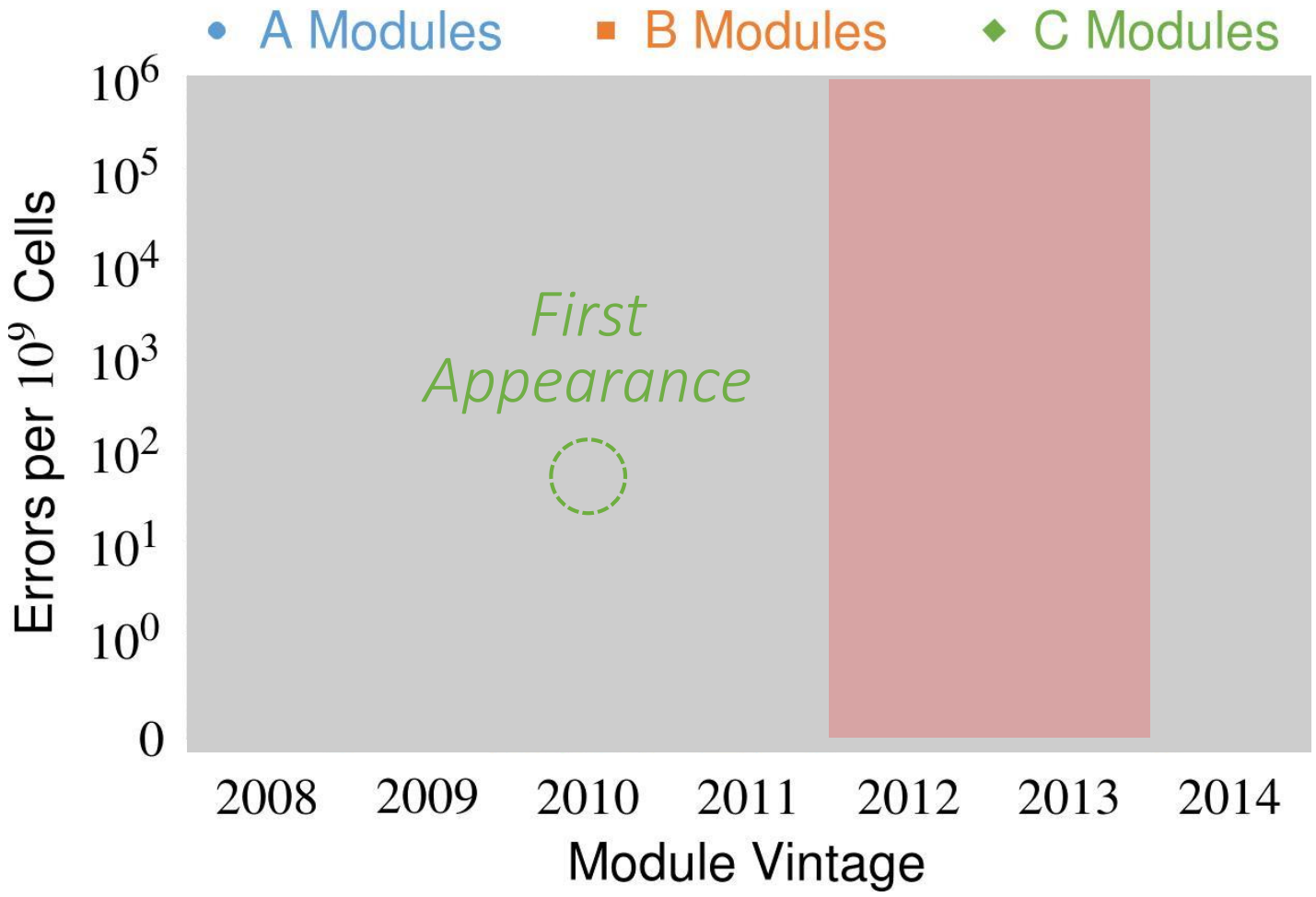
Characterization Results

1. Most Modules Are at Risk
2. Errors vs. Vintage
3. Error = Charge Loss
4. Adjacency: Aggressor & Victim
5. Sensitivity Studies
6. Other Results in Paper

1. Most Modules Are at Risk

A company	B company	C company
 86% (37/43)	 83% (45/54)	 88% (28/32)
Up to 1.0×10^{7} errors	Up to 2.7×10^{6} errors	Up to 3.3×10^{5} errors

2. Errors *vs.* Vintage



All modules from 2012–2013 are vulnerable

3. Error = Charge Loss

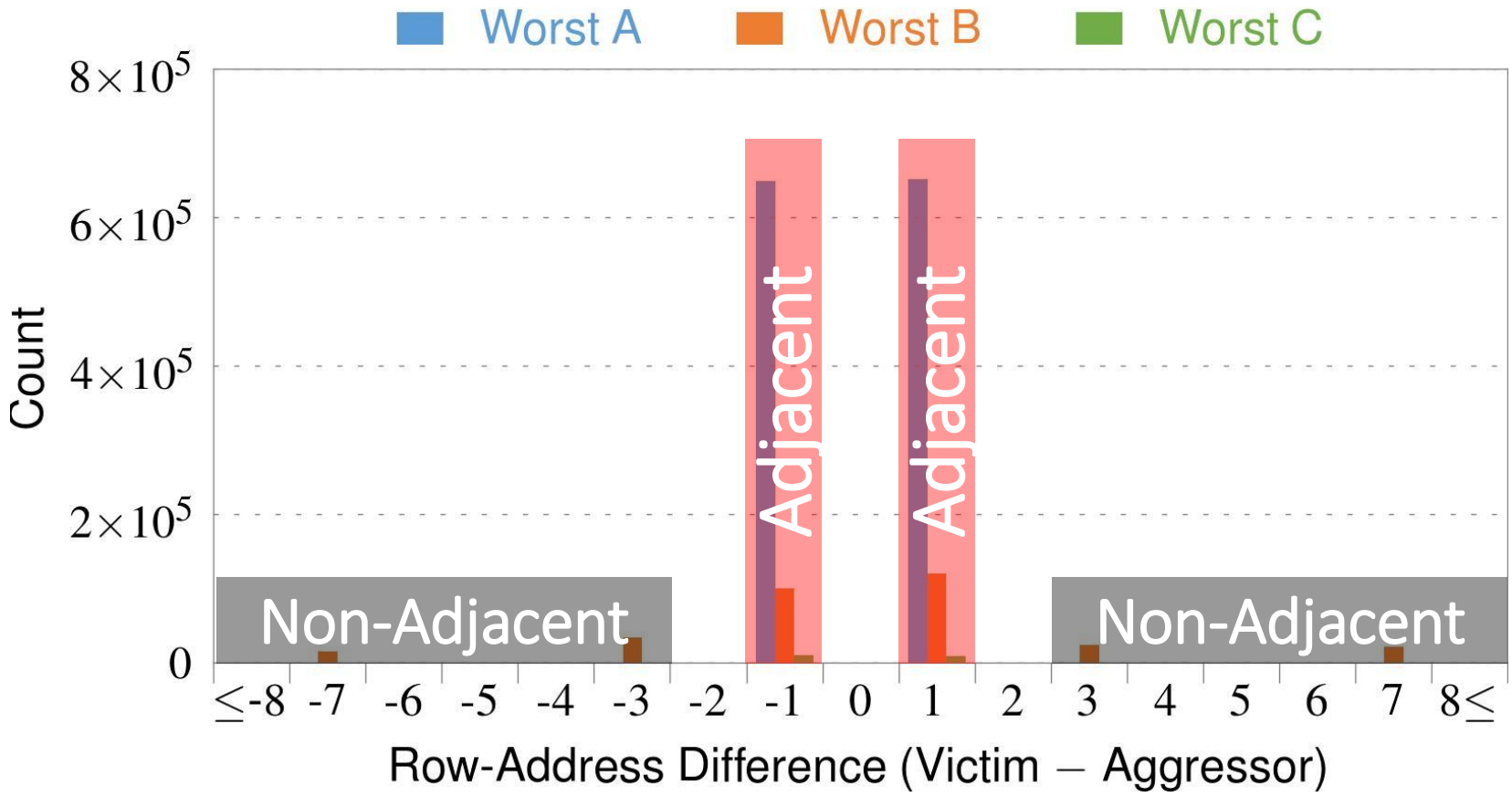
- Two types of errors
  - '1' → '0'
  - '0' → '1'
- A given cell suffers only one type

- Two types of cells
  - True: Charged ('1')
  - Anti: Charged ('0')
- Manufacturer's design choice

- True-cells have only '1' → '0' errors
- Anti-cells have only '0' → '1' errors

Errors are manifestations of charge loss

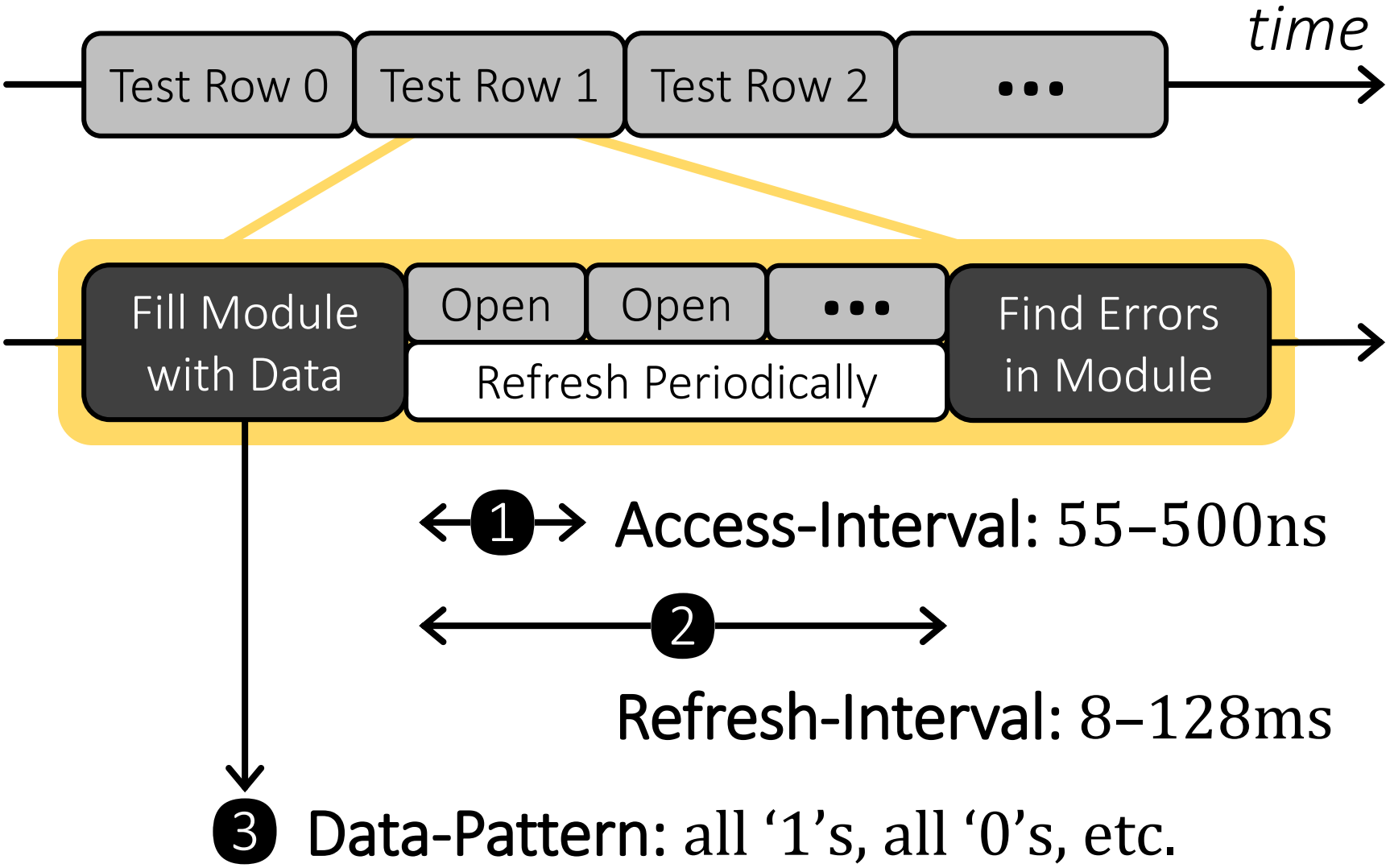
4. Adjacency: Aggressor & Victim



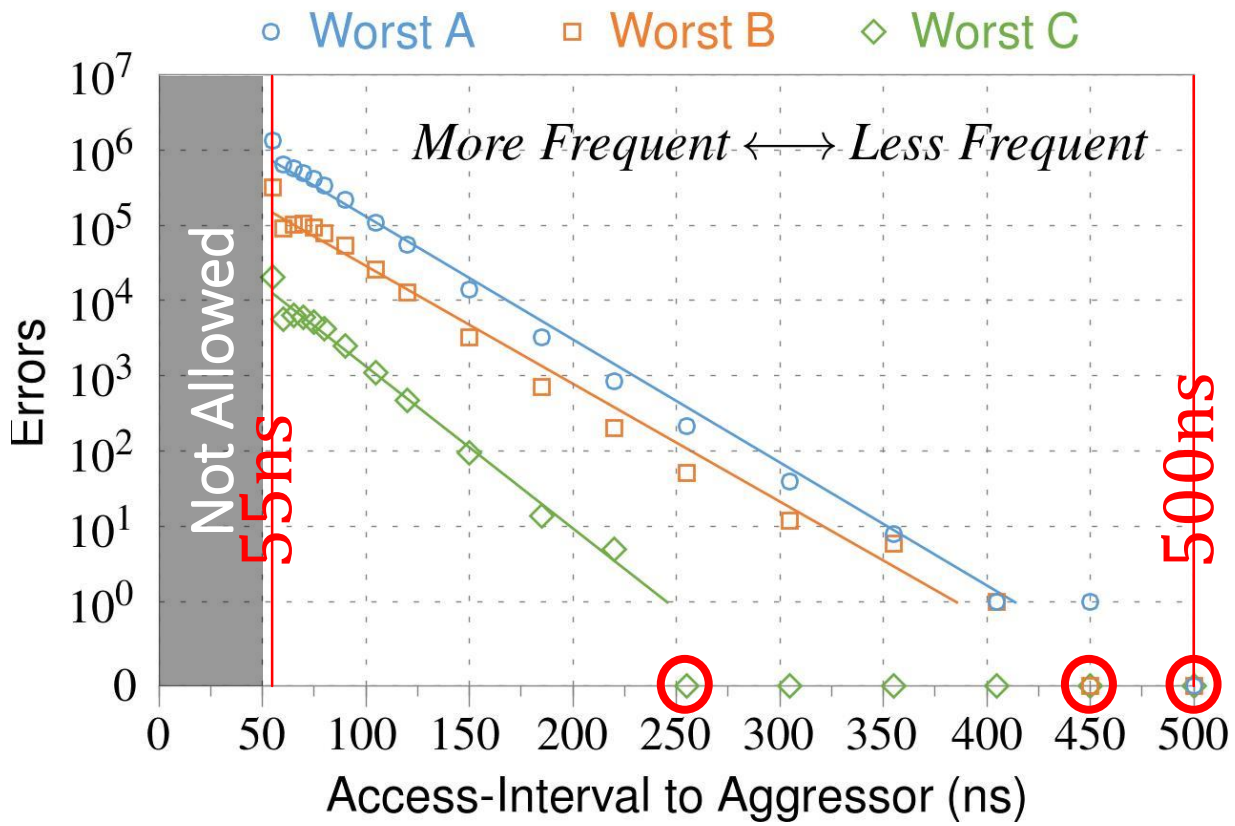
Note: For three modules with the most errors (only first bank)

Most aggressors & victims are adjacent

5. Sensitivity Studies



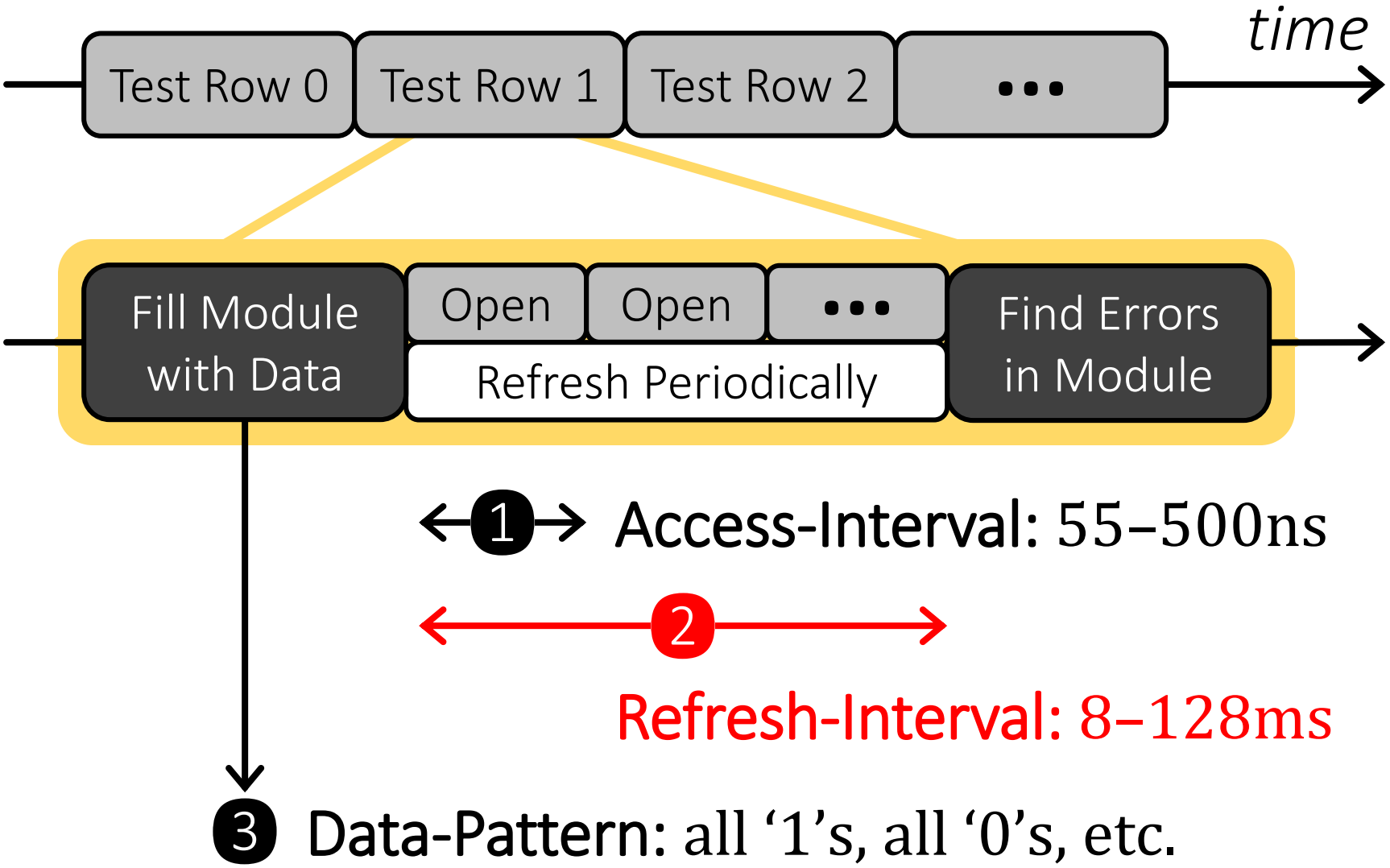
❶ Access-Interval (Aggressor)



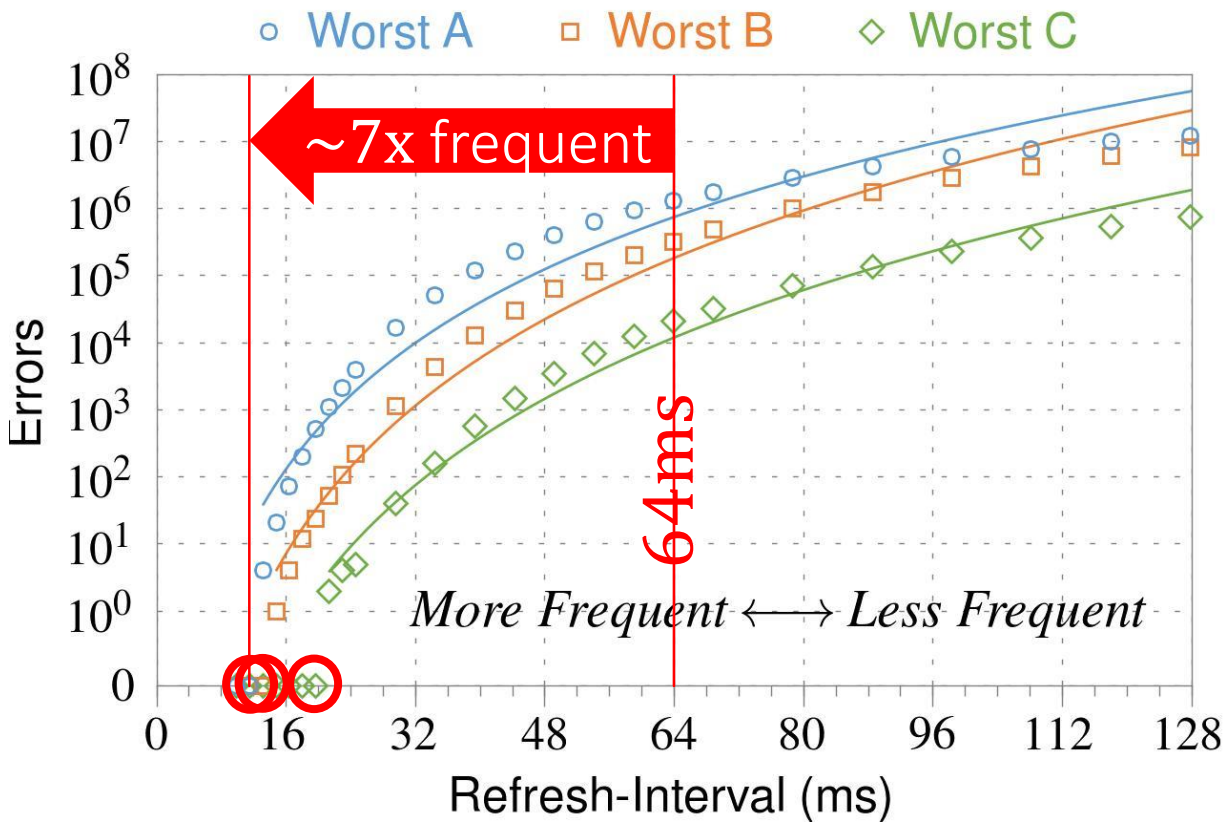
Note: For three modules with the most errors (only first bank)

Less frequent accesses → Fewer errors

5. Sensitivity Studies



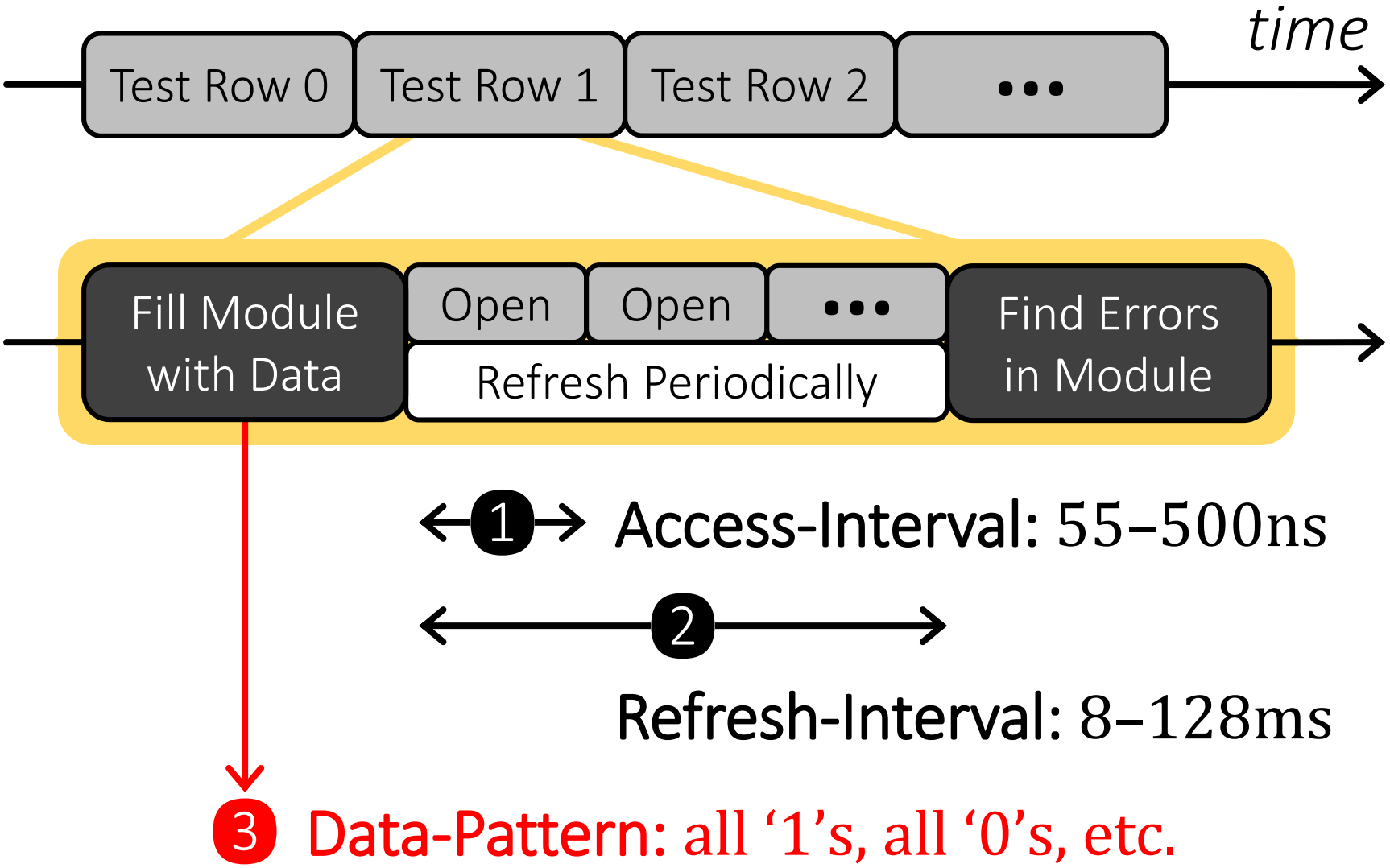
❷ Refresh-Interval



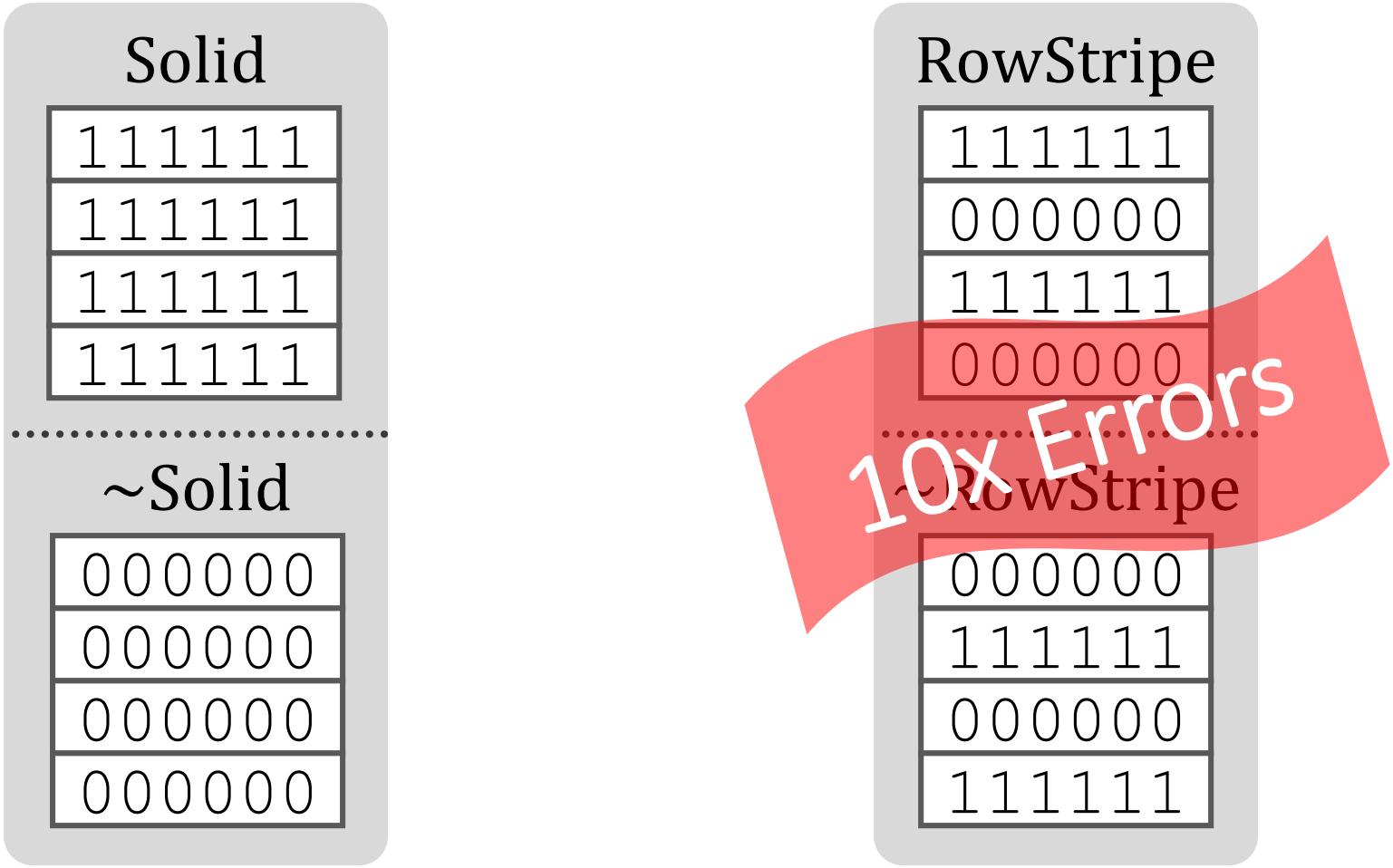
Note: Using three modules with the most errors (only first bank)

More frequent refreshes → Fewer errors

5. Sensitivity Studies



❸ Data-Pattern



Errors affected by data stored in other cells

Naive Solutions

1. *Throttle accesses to same row*
   - Limit access-interval: $\geq 500\text{ns}$
   - Limit number of accesses: $\leq 128\text{K}$ $(=64\text{ms}/500\text{ns})$

2. *Refresh more frequently*
   - Shorten refresh-interval by $\sim 7\text{x}$

Both naive solutions introduce significant overhead in performance and power

Characterization Results

1. Most Modules Are at Risk
2. Errors vs. Vintage
3. Error = Charge Loss
4. Adjacency: Aggressor & Victim
5. Sensitivity Studies
6. Other Results in Paper

6. Other Results in Paper

- *Victim Cells ≠ Weak Cells (i.e., leaky cells)*
  - Almost no overlap between them

- *Errors not strongly affected by temperature*
  - Default temperature: $50°C$
  - At $30°C$ and $70°C$, number of errors changes $<15\%$

- *Errors are repeatable*
  - Across ten iterations of testing, $>70\%$ of victim cells had errors in every iteration

6. Other Results in Paper (cont'd)

- *As many as 4 errors per cache-line*
  - Simple ECC (e.g., SECDED) cannot prevent all errors

- *Number of cells & rows affected by aggressor*
  - Victims cells per aggressor: ≤ 110
  - Victims rows per aggressor: ≤ 9

- *Cells affected by two aggressors on either side*
  - Very small fraction of victim cells (< 100) have an error when either one of the aggressors is toggled

1. Historical Context

2. Demonstration (Real System)

3. Characterization (FPGA-Based)

4. Solutions

Several Potential Solutions

- Make better DRAM chips — Cost
- Refresh frequently — Power, Performance
- Sophisticated ECC — Cost, Power
- Access counters — Cost, Power, Complexity

Our Solution

- **PARA:** *Probabilistic Adjacent Row Activation*
- **Key Idea**
  - After closing a row, we activate (i.e., refresh) one of its neighbors with a low probability: $\mathrm{p} = 0.005$
- **Reliability Guarantee**
  - When $\mathrm{p}=0.005$, errors in one year: 9.4×10^{-14}
  - By adjusting the value of p, we can provide an arbitrarily strong protection against errors

Advantages of PARA

- *PARA refreshes rows infrequently*
  - Low power
  - Low performance-overhead
    - Average slowdown: 0.20% (for 29 benchmarks)
    - Maximum slowdown: 0.75%
- *PARA is stateless*
  - Low cost
  - Low complexity

- *PARA is an effective and low-overhead solution to prevent disturbance errors*

Conclusion

- *Disturbance errors are **widespread** in DRAM chips sold and used today*
- *When a row is opened repeatedly, **adjacent rows leak charge** at an accelerated rate*
- *We propose a **stateless solution** that prevents disturbance errors with low overhead*
- *Due to difficulties in DRAM scaling, **new and unexpected types of failures** may appear*

Flipping Bits in Memory Without Accessing Them

DRAM Disturbance Errors

Yoongu Kim

Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee,

Donghyuk Lee, Chris Wilkerson, Konrad Lai, Onur Mutlu

Carnegie Mellon SAFARI intel