Flipping Bits in Memory Without Accessing Them:

An Experimental Study of DRAM Disturbance Errors

Yoongu Kim[1] Ross Daly* Jeremie Kim[1] Chris Fallin* Ji Hye Lee[1]
Donghyuk Lee[1] Chris Wilkerson[2] Konrad Lai Onur Mutlu[1]

[1]Carnegie Mellon University [2]Intel Labs

*Work done while at Carnegie Mellon University

ISCA 2014

Presented by Allan Benelli

ETH Zürich

07 November 2018

Problem

Problem

- The continued scaling of DRAM process technology has enabled smaller cells to be placed closer to each other

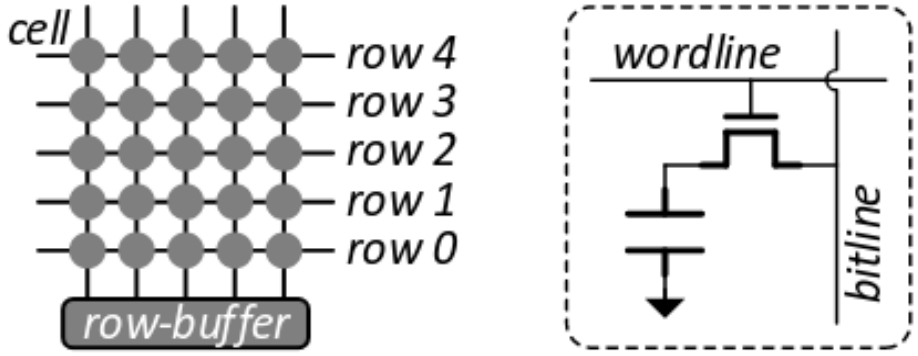
- This gives us:
  - Increase of cells per unit area
  - Decrease of cost per bit memory
- But also:
  - Reduced noise margin, more vulnerable to data loss
  - Electromagnetic coupling effects between cells
  - Higher variation in process technology increases number of outlier cells

Problem

- As a result, high-density DRAM is more likely to suffer from *disturbance*, a phenomenon in which different cells interfere with each other's operation.
- If a cell is disturbed beyond its noise margin, it malfunctions and experiences a *disturbance error*.

Background

DRAM Cell



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

Figure 1. DRAM consists of cells

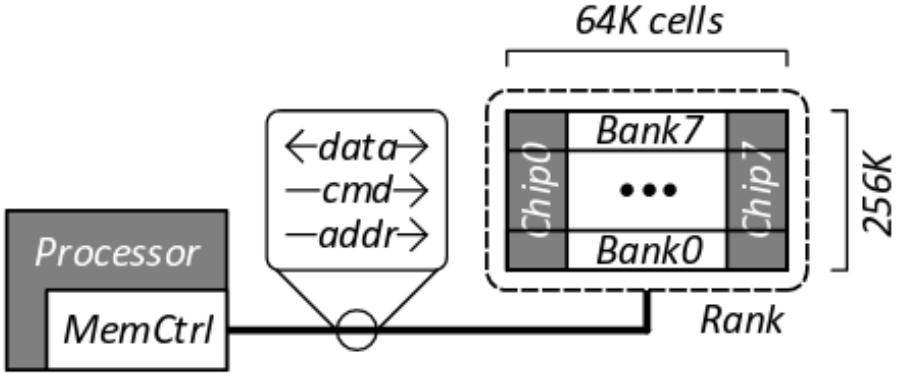
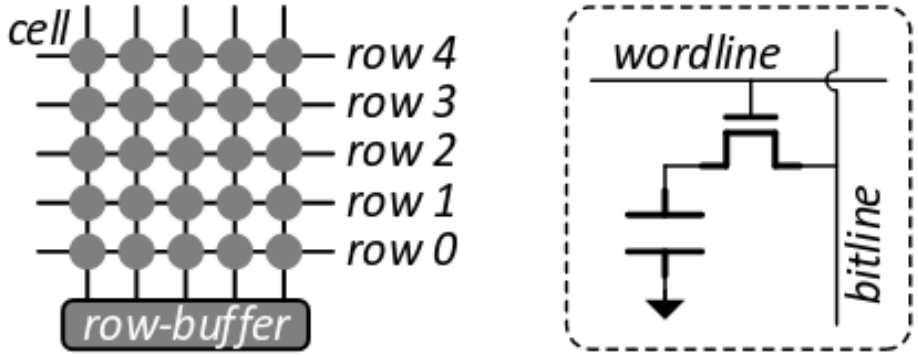


Figure 2. Memory controller, buses, rank, and banks

DRAM Access & Refresh



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

- **Open Row**: raise wordline, transfer data into row-buffer
- **Read/Write**: access row-buffer's data
- **Close Row**: lower wordline, clear row-buffer

- **Refresh**: restore the charge in cells (DDR3 ~ 64ms, can also be achieved by opening a row)

Goal

Goal

- Expose the existence and the widespread nature of disturbance errors in commodity DRAM chips sold and used "today" (2014).

Novelty, Key Approach, and Ideas

Novelty

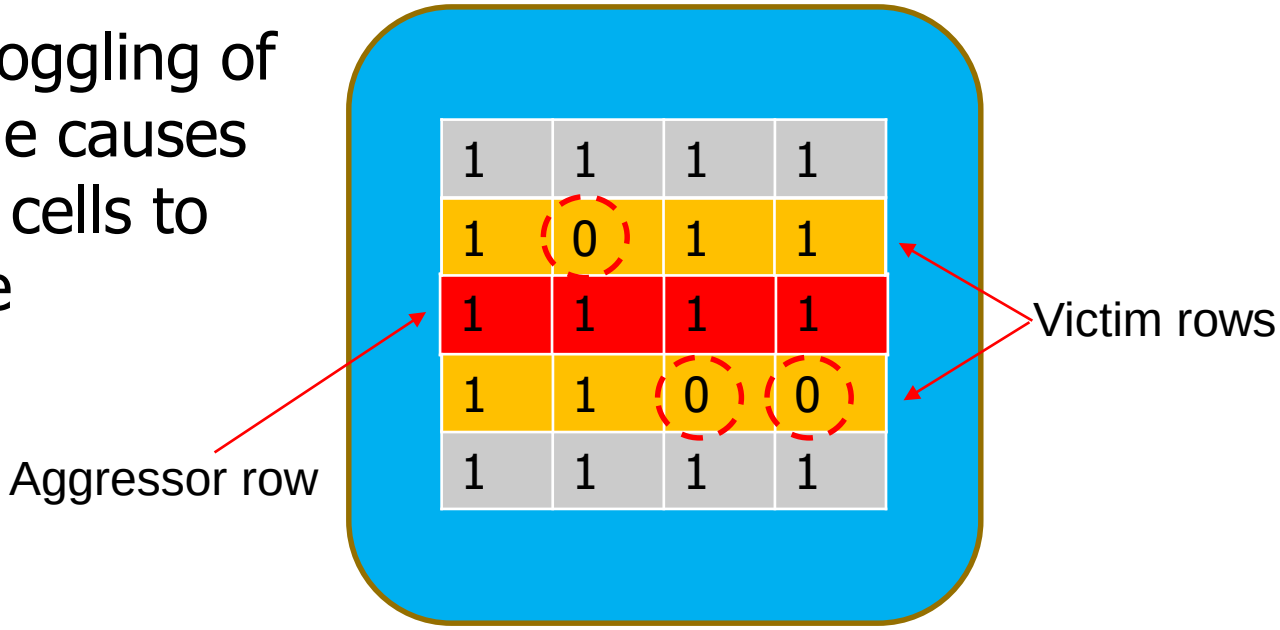
- Demonstrates the existence of DRAM disturbance errors on real systems using DRAM devices
  - Known as „RowHammer“
- Extensively characterizes these errors using FPGA-based testing platform
- Proposes and explores various solutions to prevent DRAM disturbance errors and shows a novel, low-cost system-level approach

Key-Ideas & Approach

- Causes of Disturbance Errors
  - Electromagnetic coupling
    - Toggling the wordline voltage briefly increases the voltage of adjacent wordlines, this slightly opens adjacent rows -> Leakage of charge
  - Conductive bridges
  - Hot-carrier injection

Toggling the wordline

- Repeated toggling of the wordline causes the nearby cells to leak charge



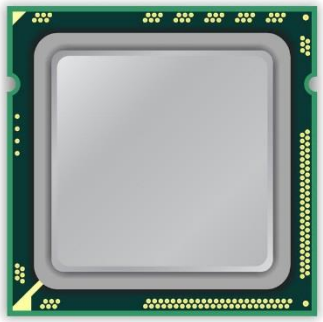
Mechanisms

How to Induce Errors

- Is it that simple?
  - No!
- 1. Avoid cache hits
  - Flush X from cache
- 2. Avoid row hits to X
  - Read Y in another row

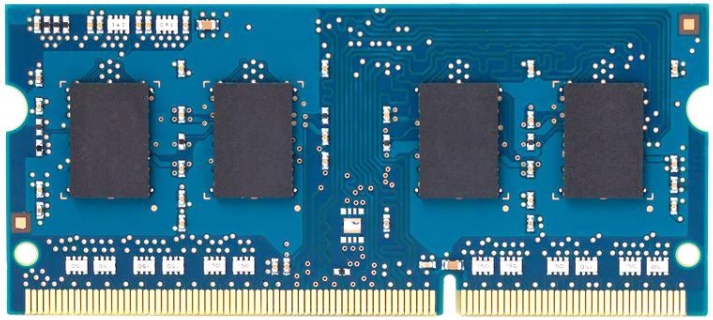
How to Induce Errors

x86 CPU

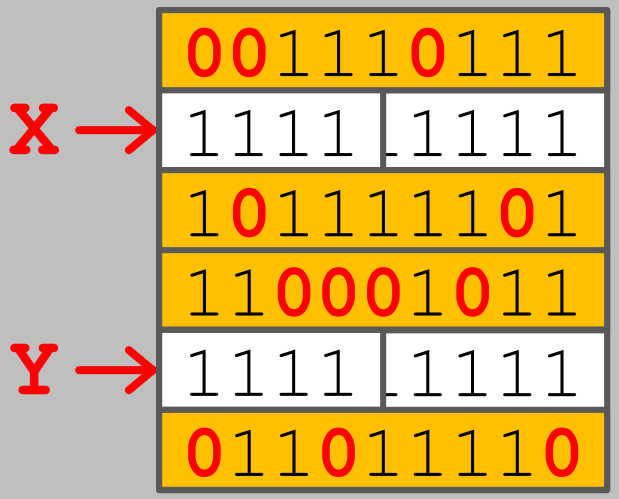


DDR3

DRAM Module



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



Y. Kim's Talk on: "Flipping Bits in Memory Without Accessing Them

Key Results: Methodology and Evaluation

Methodology

- 8 FPGA boards with DDR3 DRAM memory controller
- Tested 129 DRAM modules from manufactures A, B and C, with capacities from 512MB-2GB and production year ’08-14

```
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

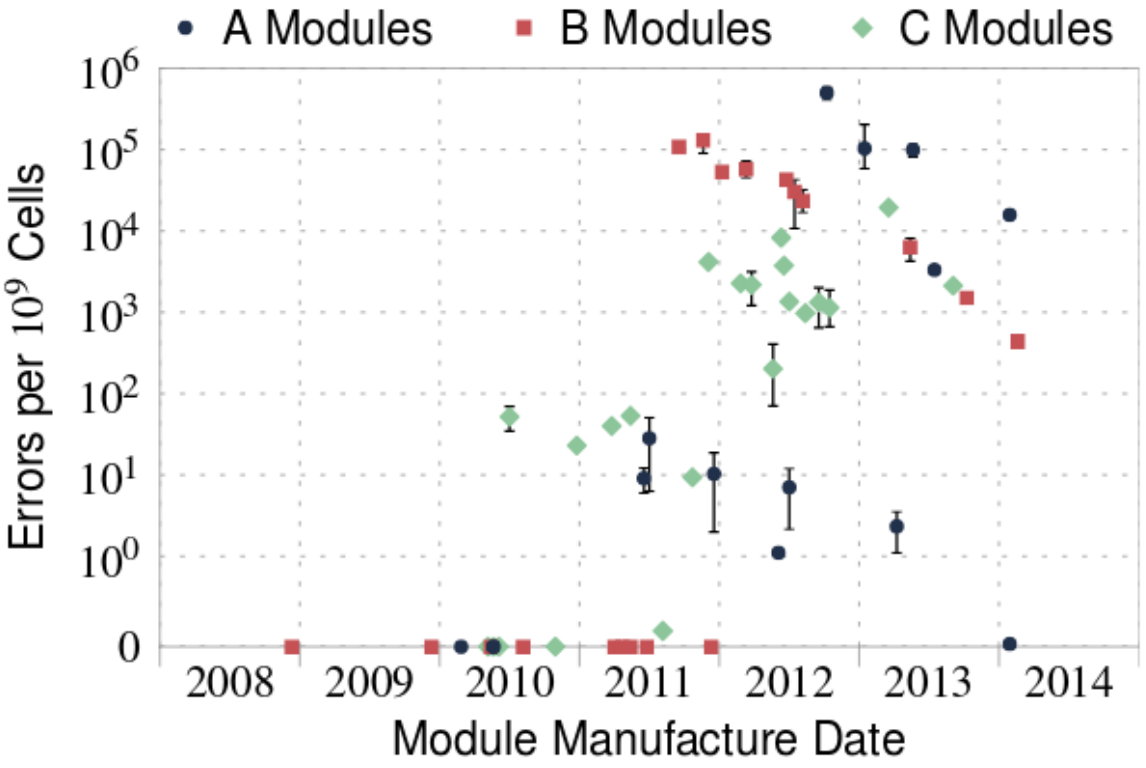
Access Interval (AI)
- Time between two accesses

Refresh Interval (RI)
- Time between two refreshes

Data Pattern (DP)
- Data stored in DRAM
- e.g. RowStripe (~RowStripe) alternate rows 1s and 0s

Disturbance Errors are Widespread

- Most modules are at risk
  - In 110 / 129 tested modules they were able to induce errors
- The modules without errors were built before 2012 (except one)



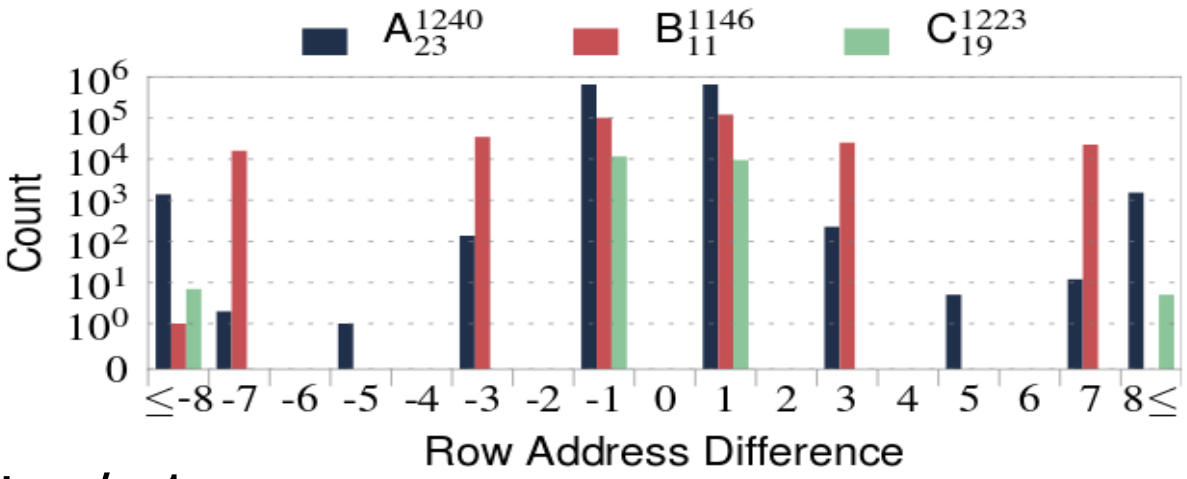
Error = Charge Loss

- Two types of errors
  - - '1' -> '0' and '0' -> '1’
- A given cell suffers only one type
- Two types of cells (chosen by manufacturer)
  - True-cell: Charged = 1 -> only '1' -> '0' errors
  - Anti-cell: Charged = 0 -> only '0' -> '1' errors
- Errors are a loss of charge

- Example module from A:

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

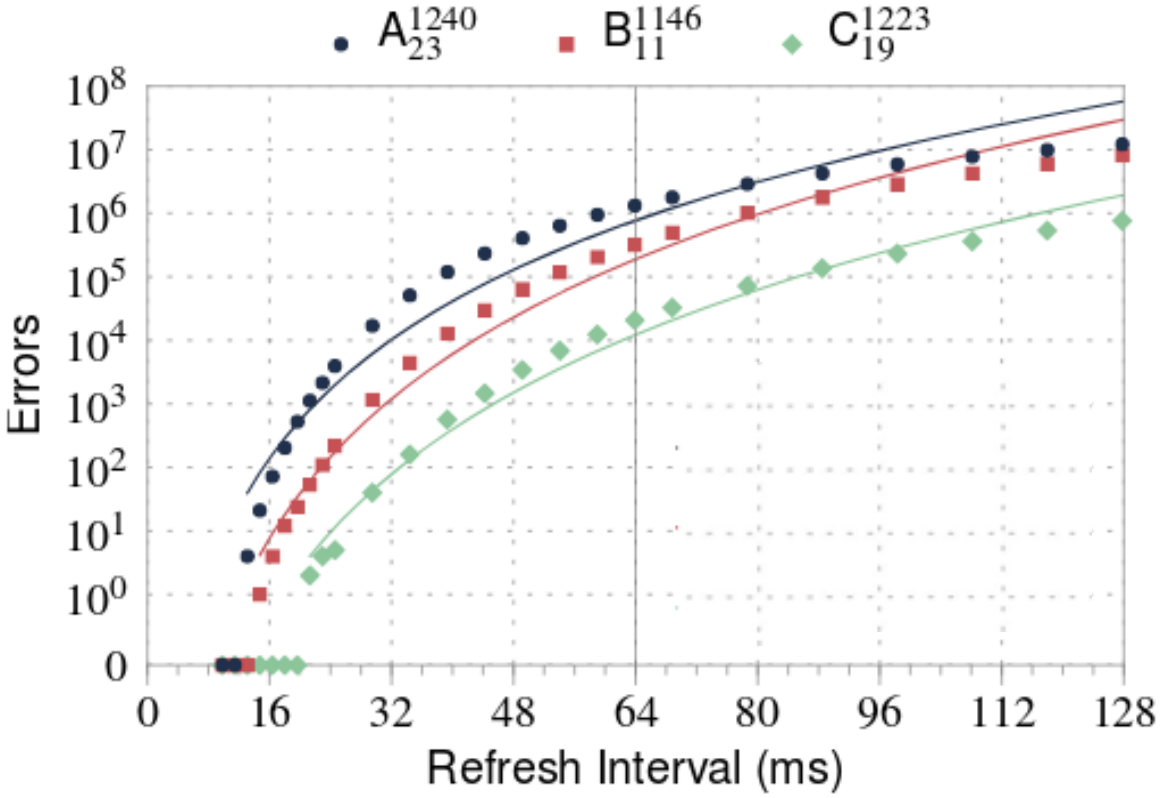
Address Correlation



- Peaks at +/- 1
- But why this distribution?
  - Physical address may differ from logical address
  - Fault rows are often re-mapped to spare rows
  - Aggressor row can affect more than two rows

Sensitivity

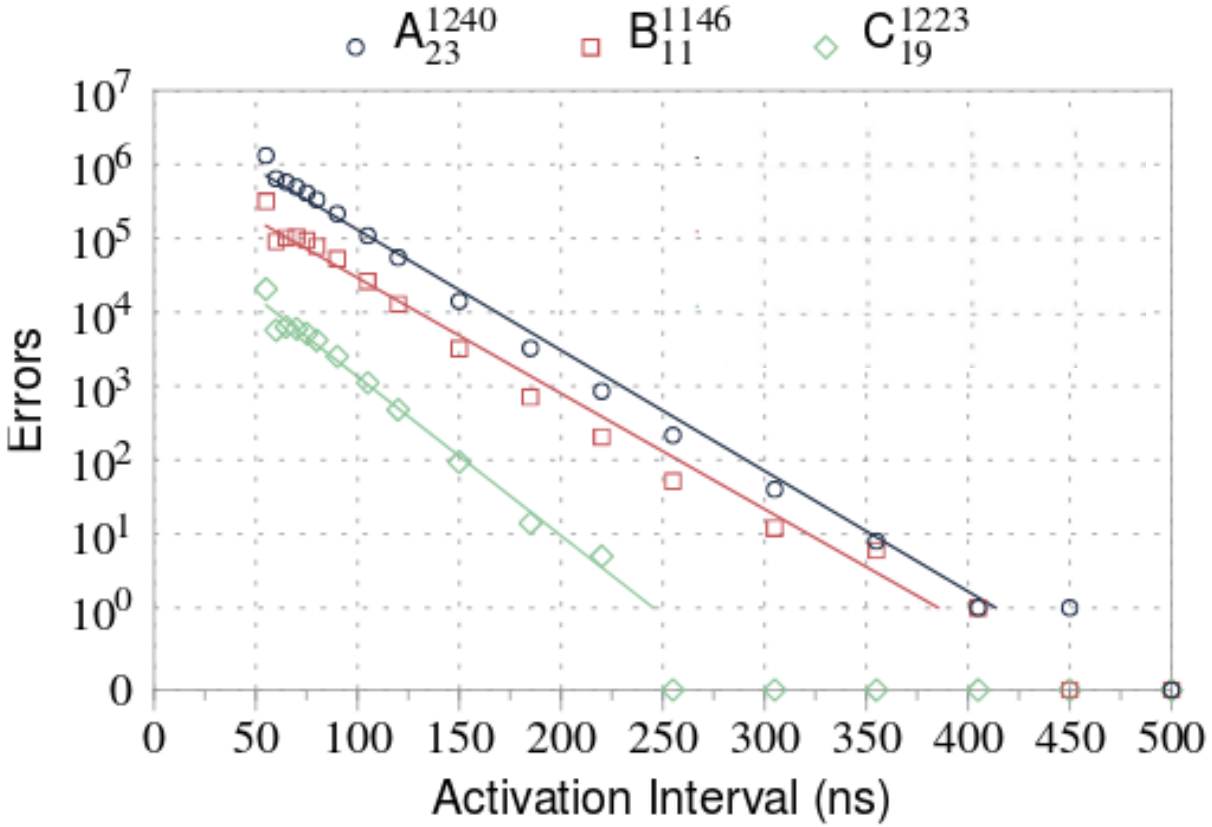
- Shorter RI -> fewer errors



- To eliminate all disturbance errors the refresh interval must be shortened by 7x for the worst module

Sensitivity

- Longer AI -> fewer errors



Sensitivity

- Errors also dependent on data stored in other cells

Solid	RowStripe	ColStripe	Checkered
111111	111111	101010	101010
111111	000000	101010	010101
111111	111111	101010	101010
111111	000000	101010	010101

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**

- RowStripe causes ~10x more errors than Solid

Error Correction Code - ECC

- Couldn't we just use simple Error Correction Codes as SECDED?
  - SECDED (:= Single Error Correction, Double Error-Detection) detects up to two errors and can correct one error
- How many errors per row?

Module	*Number of 64-bit words with X errors*			
	$X = 1$	$X = 2$	$X = 3$	$X = 4$
A_{23}	9,709,721	**181,856**	**2,248**	**18**
B_{11}	2,632,280	**13,638**	**47**	0
C_{19}	141,821	**42**	0	0

- SECDED is not safe!

Other results

- Victim Cells != Weak Cells
  - Weak cells := Cells with the shortest retention times
- Errors are repeatable, but needs a lot of testing time
- Errors are almost independent of temperature change
- Some cells have two aggressors

Possible Solutions

- Make better chips
  - … depends on process technology
- Correct errors
  - … multibit errors and overhead
- Refresh all rows frequently
  - … shorten RI -> overhead and performance
- Retire cells (manufacturer)
  - … exhaustive search, many spare cells required
- Retire cells (end-user)
  - … end-user pays for identifying and remapping
- Identify hot rows, refresh neighbours
  - … counters needed, complex, costs

Proposed Solution

- PARA (Probabilistic Adjacent Row Activation)
  - Idea:
    - When a row is open/closed, an adjacent row is opened with small probability
  - Mechanism:
    - When a row is closed, flip a biased coin ($p << 1$)
    - If head, refresh one of the two adjacent row
  - Problem:
    - Needs to know how logical mapping is done by manufacturer
  - Advantages:
    - Refreshes row infrequently (low power & performance-overhead)
    - Stateless (low cost & low complexity)

Summary

Summary

- Problem:
  - High-density DRAM is more likely to suffer from disturbance
- Goal:
  - Expose the existence and the widespread nature of disturbance errors in commodity DRAM chips
- Key results:
  - 110 out of 129 modules were vulnerable
  - Root cause: repeated toggling of a wordline
- Conclusion:
  - Disturbance errors are an emerging problem
  - Many deployed systems could be at risk

Strengths

Strengths

- The first paper to expose the widespread existence of disturbance errors in DRAM chips
  - Is the basis for a lot of further work (321 citations)
- Identifies a new reliability problem and a security vulnerability, RowHammer, that affects an entire generation of computing systems being used today
  - RowHammer is still relevant today!
- Real-system approach, not only theoretical
- With PARA a neat solution is provided
- Clear structured paper, worth reading, if you want to understand further papers on RowHammer

Weaknesses

Weaknesses

- Assumes the existence of security exploits, but just touches the topic and doesn't provide a working example.
- Paper is limited to x86-architecture.
- Paper relies on the memory controller flipping a coin. If the outco acker may c in could d be avoid

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

- Difference between # of bitflips with AMD and Intel processors is just explained in a footnote and limited to speed

Thoughts and Ideas

Thoughts and Ideas

- What about RowHammer today?
  - Google Project Zero exploited the DRAM RowHammer bug to gain kernel privileges
  - Recent studies and reports also suggest vulnerability of DDR4 Ram, mobilephones (ARM), GPU of mobilephones and RowHammer Attacks over the Network.
  - "Solutions": Shorten RI to 32ms, ECC, TRR and restrict clflush

- What about ARM / Mobile platform? What about SRAM, flash and harddisk?
  - ARM --> Drammer: Deterministic Rowhammer Attacks on Mobile Platforms [V. van der Veen et al., 2016]
  - NAND Flash --> Read Disturb Errors in MLC NAND Flash Memory: … [Y. Cai, O.Mutlu, et al. 2015]

Takeaways

Key Takeaways

- "*It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after*" (Slides of O.Mutlu)
- RowHammer is a real issue - Disturbance errors are widespread!
- The fact that computer parts are getting smaller and smaller and the associated problems including RowHammer should receive much more attention than it currently enjoys.
- Technological progress in manufacturing technology and the scale down to smaller dimensions can produce unexpected errors that one wouldn't think of.

Questions/Open Discussion

Discussion

- Is shortening the refresh interval (and or lengthen the activation interval) a practical approach?
- Is it very likely for a normal application to hammer a row accidentally?
- Is PARA enough? Do you have other solutions in mind?
- How would you implement such a coin flip used in PARA?
- Was this paper a roadmap for hackers?

Additional Slides

Additional papers and webpages

- Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript [D. Gruss et al. 2015]
- Throwhammer: Rowhammer Attacks over the Network and Defenses [A. Tatar et al. 2018]
- DDR4: http://www.thirdio.com/rowhammer.pdf
- Exploiting the DRAM rowhammer bug to gain kernel privileges [Mark Seaborn, et al.2015]
- Read Disturb Errors in MLC NAND Flash Mermory: … [Y. Cai, O.Mutlu, et al. 2015]
- ANVIL: Software-Based Protection Agains Next-Generation Rowhammer Attacks [Z. Aweke et al., 2016]
- Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU [P. Frigo et al. 2018]
- Drammer: Deterministic Rowhammer Attacks on Mobile Platforms [V. van der Veen et al., 2016]
- A New Approach for Rowhammer Attacks [R. Qiao, M.Seaborn]

Additional slides

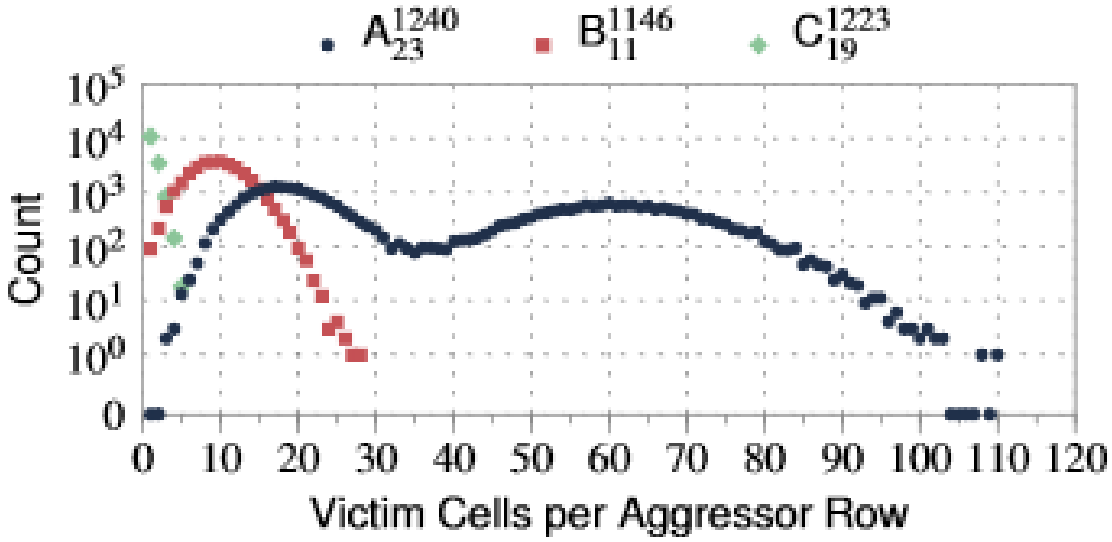
Manufacturer	Module	Date*	Timing†		Organization		Chip			Victims-per-Module			RI_{th} (ms)
		(yy-ww)	Freq (MT/s)	t_{RC} (ns)	Size (GB)	Chips	Size (Gb)‡	Pins	DieVersion§	Average	Minimum	Maximum	Min
C Total of 32 Modules	C_1	10-18	1333	49.125	2	8	2	×8	$\mathcal{A}$	0	0	0	–
	C_2	10-20	1066	50.625	2	8	2	×8	$\mathcal{A}$	0	0	0	–
	C_3	10-22	1066	50.625	2	8	2	×8	$\mathcal{A}$	0	0	0	–
	C_{4-5}	10-26	1333	49.125	2	8	2	×8	$\mathcal{B}$	8.9×10^2	6.0×10^2	1.2×10^3	29.5
	C_6	10-43	1333	49.125	1	8	1	×8	$\mathcal{T}$	0	0	0	–
	C_7	10-51	1333	49.125	2	8	2	×8	$\mathcal{B}$	4.0×10^2	4.0×10^2	4.0×10^2	29.5
	C_8	11-12	1333	46.25	2	8	2	×8	$\mathcal{B}$	6.9×10^2	6.9×10^2	6.9×10^2	21.3
	C_9	11-19	1333	46.25	2	8	2	×8	$\mathcal{B}$	9.2×10^2	9.2×10^2	9.2×10^2	27.9
	C_{10}	11-31	1333	49.125	2	8	2	×8	$\mathcal{B}$	3	3	3	39.3
	C_{11}	11-42	1333	49.125	2	8	2	×8	$\mathcal{B}$	1.6×10^2	1.6×10^2	1.6×10^2	39.3
	C_{12}	11-48	1600	48.125	2	8	2	×8	$\mathcal{C}$	7.1×10^4	7.1×10^4	7.1×10^4	19.7
	C_{13}	12-08	1333	49.125	2	8	2	×8	$\mathcal{C}$	3.9×10^4	3.9×10^4	3.9×10^4	21.3
	C_{14-15}	12-12	1333	49.125	2	8	2	×8	$\mathcal{C}$	3.7×10^4	2.1×10^4	5.4×10^4	21.3
	C_{16-18}	12-20	1600	48.125	2	8	2	×8	$\mathcal{C}$	3.5×10^3	1.2×10^3	7.0×10^3	27.9
	C_{19}	12-23	1600	48.125	2	8	2	×8	$\mathcal{E}$	1.4×10^5	1.4×10^5	1.4×10^5	18.0
	C_{20}	12-24	1600	48.125	2	8	2	×8	$\mathcal{C}$	6.5×10^4	6.5×10^4	6.5×10^4	21.3
	C_{21}	12-26	1600	48.125	2	8	2	×8	$\mathcal{C}$	2.3×10^4	2.3×10^4	2.3×10^4	24.6
	C_{22}	12-32	1600	48.125	2	8	2	×8	$\mathcal{C}$	1.7×10^4	1.7×10^4	1.7×10^4	22.9
	C_{23-24}	12-37	1600	48.125	2	8	2	×8	$\mathcal{C}$	2.3×10^4	1.1×10^4	3.4×10^4	18.0
	C_{25-30}	12-41	1600	48.125	2	8	2	×8	$\mathcal{C}$	2.0×10^4	1.1×10^4	3.2×10^4	19.7
	C_{31}	13-11	1600	48.125	2	8	2	×8	$\mathcal{C}$	3.3×10^5	3.3×10^5	$\mathbf{3.3 \times 10^5}$	**14.7**
	C_{32}	13-35	1600	48.125	2	8	2	×8	$\mathcal{C}$	3.7×10^4	3.7×10^4	3.7×10^4	21.3
B Total of 54 Modules	B_8	11-25	1333	49.125	2	8	2	×8	$\mathcal{C}$	0	0	0	–
	B_9	11-37	1333	49.125	2	8	2	×8	$\mathcal{D}$	1.9×10^6	1.9×10^6	1.9×10^6	11.5
	B_{10-12}	11-46	1333	49.125	2	8	2	×8	$\mathcal{D}$	2.2×10^6	1.5×10^6	$\mathbf{2.7 \times 10^6}$	11.5
	B_{13}	11-49	1333	49.125	2	8	2	×8	$\mathcal{C}$	0	0	0	–
	B_{14}	12-01	1866	47.125	2	8	2	×8	$\mathcal{D}$	9.1×10^5	9.1×10^5	9.1×10^5	**9.8**
	B_{15-31}	12-10	1866	47.125	2	8	2	×8	$\mathcal{D}$	9.8×10^5	7.8×10^5	1.2×10^6	11.5
	B_{32}	12-25	1600	48.125	2	8	2	×8	$\mathcal{E}$	7.4×10^5	7.4×10^5	7.4×10^5	11.5
	B_{33-42}	12-28	1600	48.125	2	8	2	×8	$\mathcal{E}$	5.2×10^5	1.9×10^5	7.3×10^5	11.5
	B_{43-47}	12-31	1600	48.125	2	8	2	×8	$\mathcal{E}$	4.0×10^5	2.9×10^5	5.5×10^5	13.1
	B_{48-51}	13-19	1600	48.125	2	8	2	×8	$\mathcal{E}$	1.1×10^5	7.4×10^4	1.4×10^5	14.7
	B_{52-53}	13-40	1333	49.125	2	8	2	×8	$\mathcal{D}$	2.6×10^4	2.3×10^4	2.9×10^4	21.3
	B_{54}	14-07	1333	49.125	2	8	2	×8	$\mathcal{D}$	7.5×10^3	7.5×10^3	7.5×10^3	26.2

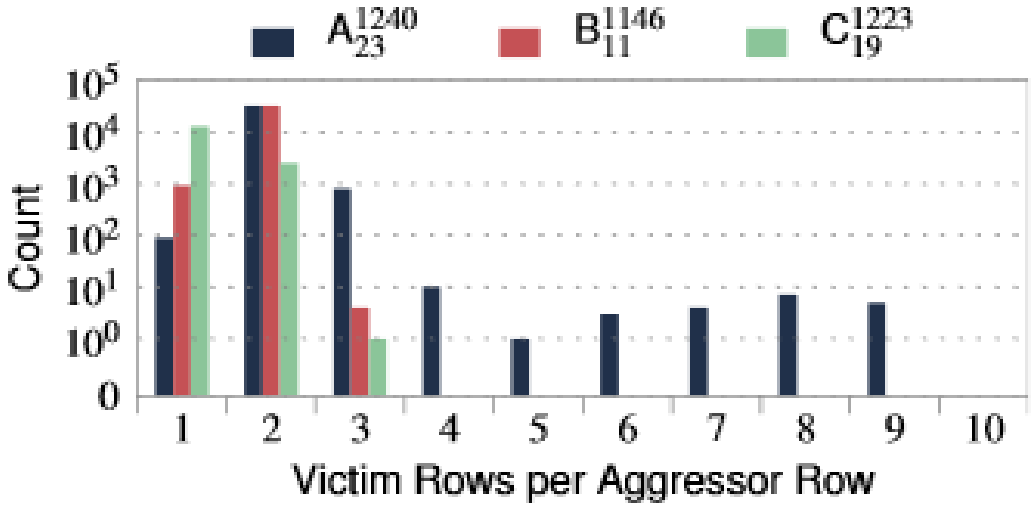
Additional slides

Access Pattern	Disturbance Errors?
1. $(open–read–close)^N$	**Yes**
2. $(open–write–close)^N$	**Yes**
3. $open–read^N–close$	No
4. $open–write^N–close$	No

Table 4. Access patterns that induce disturbance errors

Additional slides





Additional slides

Duration	N_{th}=50K	N_{th}=100K	N_{th}=200K
64ms	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

- Nth = open and close during a refresh interval
- Independent coin flips –> p_coinflip = (1-p/2)^Nth

Additional slides

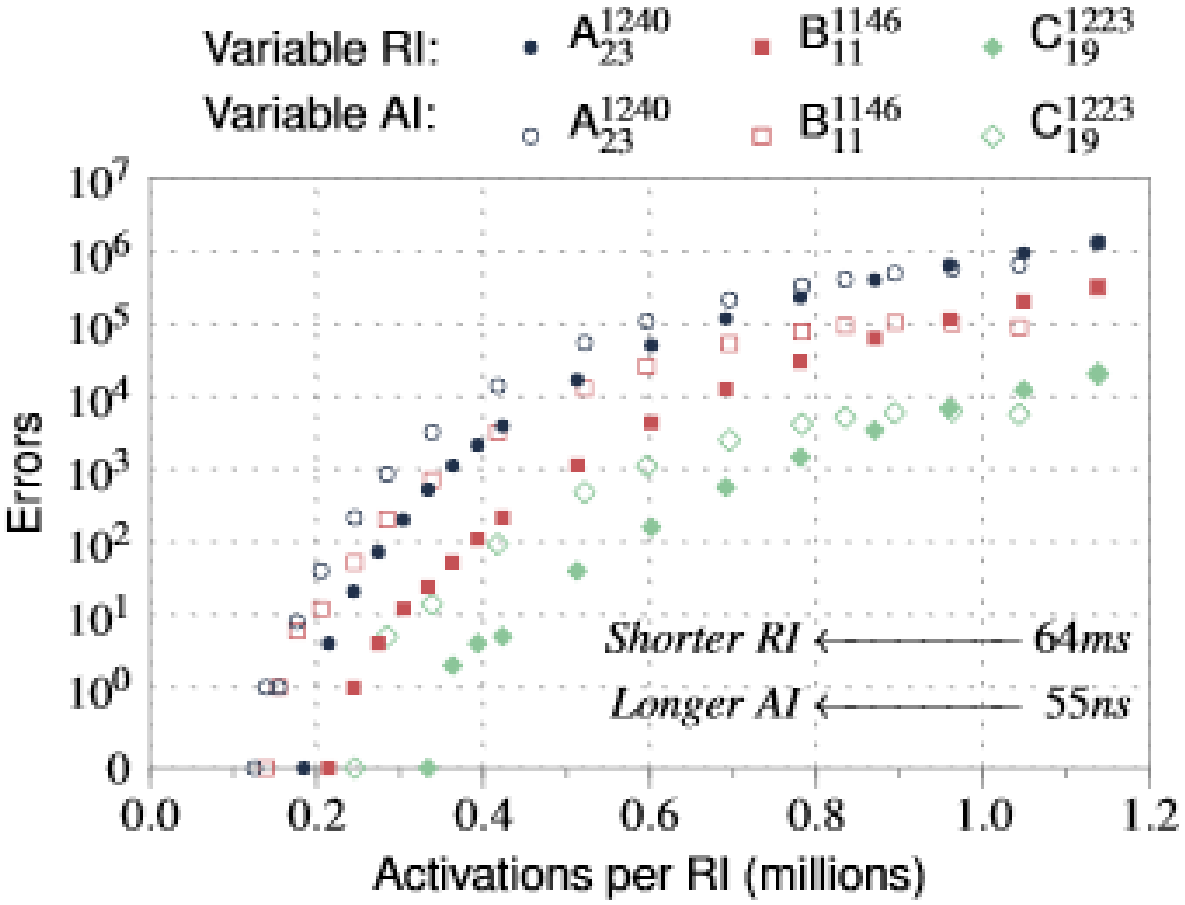


Figure 6. Number of errors vs. number of activations