Rowhammer

Joseph Hallett

bristol.ac.uk

Abstraction, abstraction…

In Computer Science we like to pretend that its all digital…

- All perfect 1s and 0s
- Hardware can be (largely) ignored
- We forget the lower level details… that's for the electronic engineers!

This doesn't always work out…

DRAM

For example, memory…

- Implemented using a capacitor and transistor per bit
- *Ganged* (arranged) into long *rows* (~8k bits)
- Arranged into *banks*

When we want to access memory, we **activate** the row in the bank by letting the capacitor discharge into an active memory buffer.

Have to actively **refresh** the data stored in the capacitors (every 64ms

Electronic Engineering is messy

Capacitors are leaky!

Current in wires induces current in other nearby wires!

1s and 0s aren't really charged on uncharged capacitors, its whether the capacitor is releasing more/less than a threshold voltage

…but this is fine because electronic components are large!

Electronic Engineering is messy

Capacitors are leaky!

Current in wires induces current in other nearby wires!

1s and 0s aren't really charged on uncharged capacitors, its whether the capacitor is releasing more/less than a threshold voltage

…but this is fine because electronic components are large!
…well they *were* large…

Rowhammer

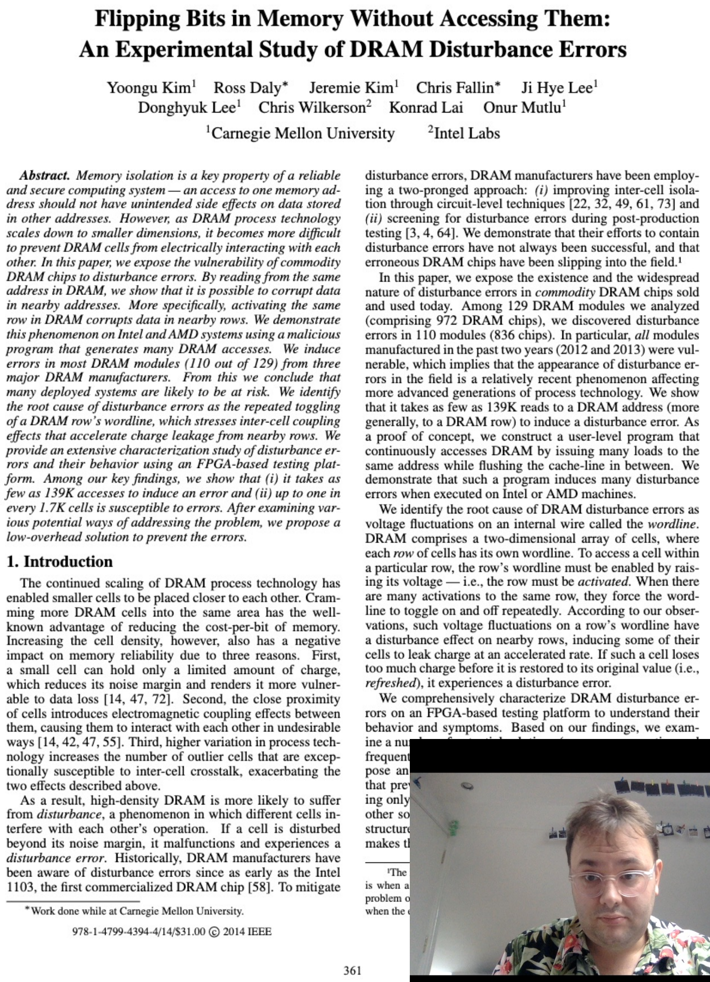
Known bug in DRAM chips since 2010.

If you repeatedly charge and discharge a row in DRAM really quickly it can sometimes cause error in nearby rows.

- Manufacturers all know about it, but its not really documented
- Seen as a *reliability* issue
- Cached memory largely fixes it, and who'd do this anyway…?
- "Not a security issue"

Not the first paper to talk about these issues, but the most detailed one, and the one which made people aware of the issue

- Still sees the problem as a reliability issue.
- More or less every chip is vulnerable to it to some extent
- …but perhaps it could be used to do something malicious maybe?



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

Abstract. *Memory isolation is a 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. However, as DRAM process technology scales down to smaller dimensions, it becomes more difficult to prevent DRAM cells from electrically interacting with each other. In this paper, we expose the vulnerability of commodity DRAM chips to disturbance errors. By reading from the same address in DRAM, we show that it is possible to corrupt data in nearby addresses. More specifically, activating the same row in DRAM corrupts data in nearby rows. We demonstrate this phenomenon on Intel and AMD systems using a malicious program that generates many DRAM accesses. We induce errors in most DRAM modules (110 out of 129) from three major DRAM manufacturers. From this we conclude that many deployed systems are likely to be at risk. We identify the root cause of disturbance errors as the repeated toggling of a DRAM row's wordline, which stresses inter-cell coupling effects that accelerate charge leakage from nearby rows. We provide an extensive characterization study of disturbance errors and their behavior using an FPGA-based testing platform. Among our key findings, we show that (i) it takes as few as 139K accesses to induce an error and (ii) up to one in every 1.7K cells is susceptible to errors. After examining various potential ways of addressing the problem, we propose a low-overhead solution to prevent the errors.*

1. Introduction

The continued scaling of DRAM process technology has enabled smaller cells to be placed closer to each other. Cramming more DRAM cells into the same area has the well-known advantage of reducing the cost-per-bit of memory. Increasing the cell density, however, also has a negative impact on memory reliability due to three reasons. First, a small cell can hold only a limited amount of charge, which reduces its noise margin and renders it more vulnerable to data loss [14, 47, 72]. Second, the close proximity of cells introduces electromagnetic coupling effects between them, causing them to interact with each other in undesirable ways [14, 42, 47, 55]. Third, higher variation in process technology increases the number of outlier cells that are exceptionally susceptible to inter-cell crosstalk, exacerbating the two effects described above.

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*. Historically, DRAM manufacturers have been aware of disturbance errors since as early as the Intel 1103, the first commercialized DRAM chip [58]. To mitigate disturbance errors, DRAM manufacturers have been employing a two-pronged approach: *(i)* improving inter-cell isolation through circuit-level techniques [22, 32, 49, 61, 73] and *(ii)* screening for disturbance errors during post-production testing [3, 4, 64]. We demonstrate that their efforts to contain disturbance errors have not always been successful, and that erroneous DRAM chips have been slipping into the field.[1]

In this paper, we expose the existence and the widespread nature of disturbance errors in *commodity* DRAM chips sold and used today. Among 129 DRAM modules we analyzed (comprising 972 DRAM chips), we discovered disturbance errors in 110 modules (836 chips). In particular, *all* modules manufactured in the past two years (2012 and 2013) were vulnerable, which implies that the appearance of disturbance errors in the field is a relatively recent phenomenon affecting more advanced generations of process technology. We show that it takes as few as 139K reads to a DRAM address (more generally, to a DRAM row) to induce a disturbance error. As a proof of concept, we construct a user-level program that continuously accesses DRAM by issuing many loads to the same address while flushing the cache-line in between. We demonstrate that such a program induces many disturbance errors when executed on Intel or AMD machines.

We identify the root cause of DRAM disturbance errors as voltage fluctuations on an internal wire called the *wordline*. DRAM comprises a two-dimensional array of cells, where each *row* of cells has its own wordline. To access a cell within a particular row, the row's wordline must be enabled by raising its voltage — i.e., the row must be *activated*. When there are many activations to the same row, they force the wordline to toggle on and off repeatedly. According to our observations, such voltage fluctuations on a row's wordline have a disturbance effect on nearby rows, inducing some of their cells to leak charge at an accelerated rate. If such a cell loses too much charge before it is restored to its original value (i.e., *refreshed*), it experiences a disturbance error.

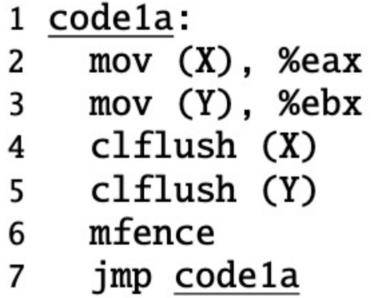
We comprehensively characterize DRAM disturbance errors on an FPGA-based testing platform to understand their behavior and symptoms. Based on our findings, we examine a nu
frequent
pose an
that pre
ing only
other so
structure
makes th

[*] Work done while at Carnegie Mellon University.

[1] The
is when a
problem o
when the

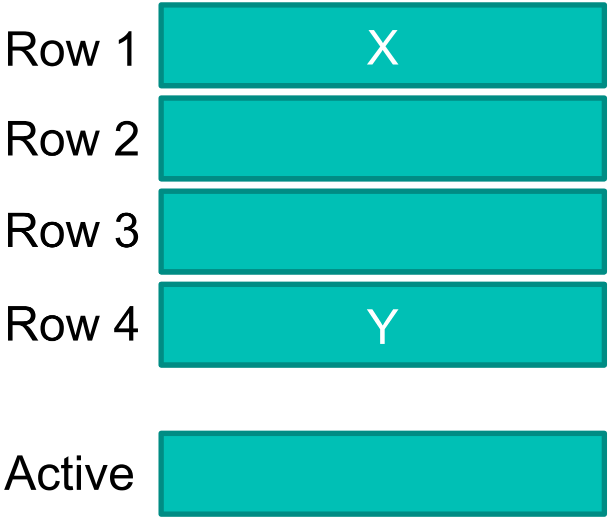
978-1-4799-4394-4/14/$31.00 © 2014 IEEE

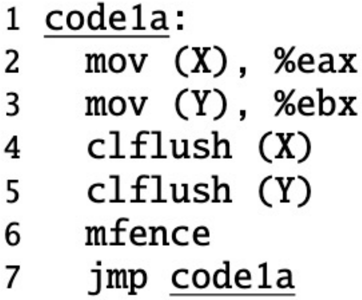
Rowhammer



```
code1a:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp code1a
```

Rowhammer

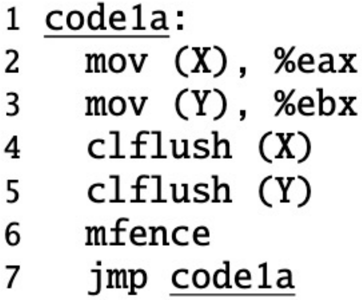




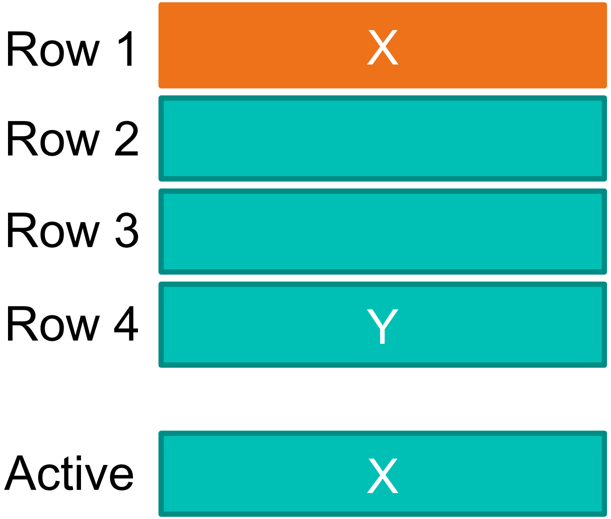
```
code1a:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp code1a
```

Say X is somewhere in Row 1 and Y in Row 4

Rowhammer



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



Row 1 X

Row 2

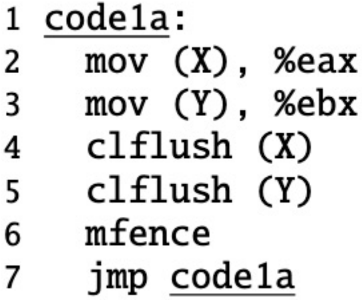
Row 3

Row 4 Y

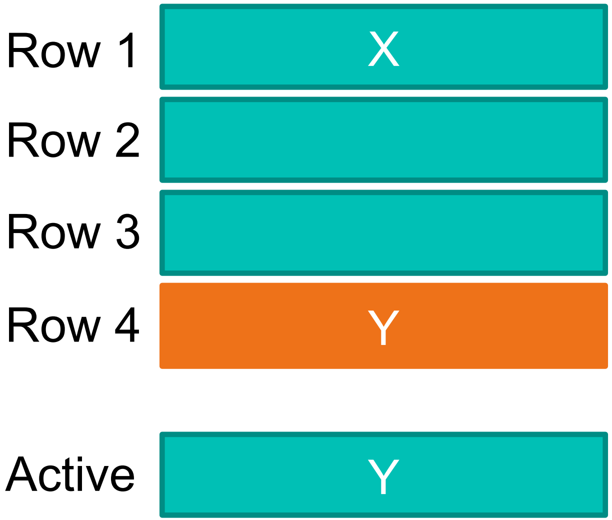
Active X

Load X…

Rowhammer



```
code1a:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp code1a
```



Row 1 X

Row 2

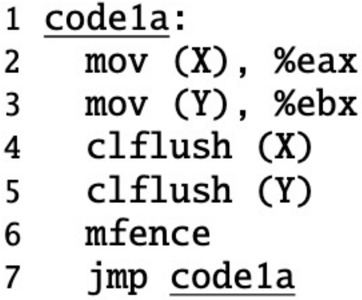
Row 3

Row 4 Y

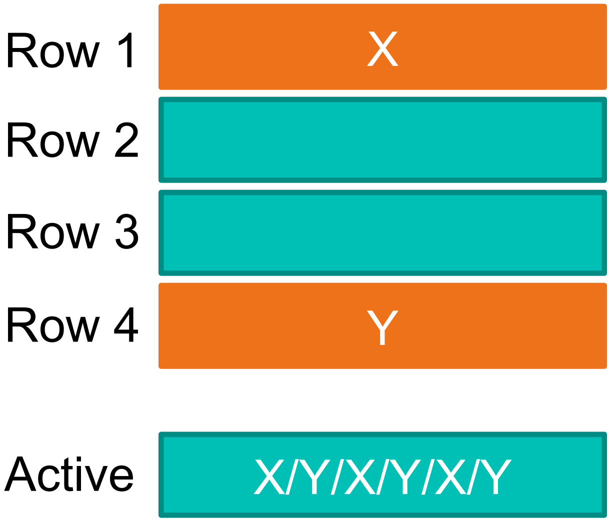
Active Y

Load Y…

Rowhammer

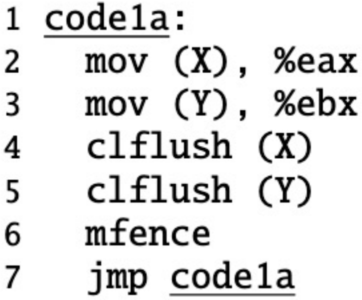


```
codela:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp codela
```

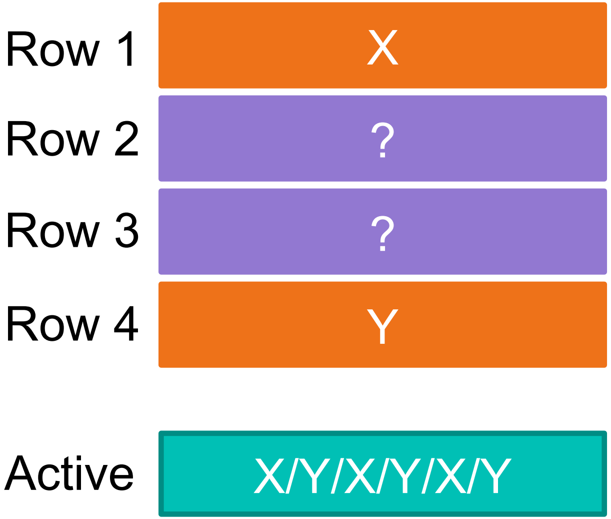


Repeat *really fast…*

Rowhammer



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



Row 1 X

Row 2 ?

Row 3 ?

Row 4 Y

Active X/Y/X/Y/X/Y

You'll start to get errors in adjacent rows

Rowhammer

Mark Seaborn and Halvar Flake (and others) explore further!

- Between 30%-100% of rows of memory for most DDR3 susceptible

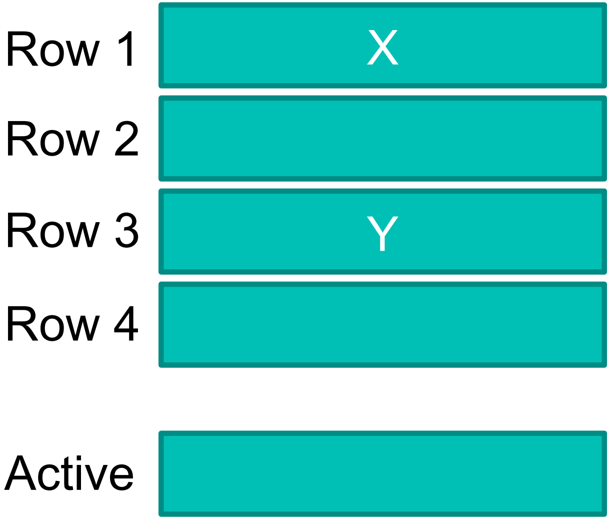
Bit flips are consistent

- You might not know which bit will flip but it will be the same bit.
- Same hardware generally flips the same bits

Double sided row hammering makes flipping bits *really likely*

- Even in chips thought to be resistant

Double Sided Rowhammer



Row 1 | X

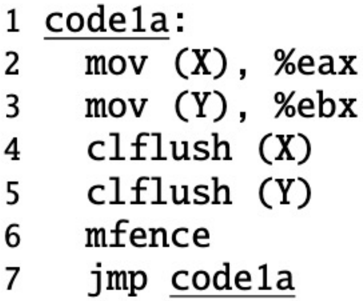
Row 2

Row 3 | Y

Row 4

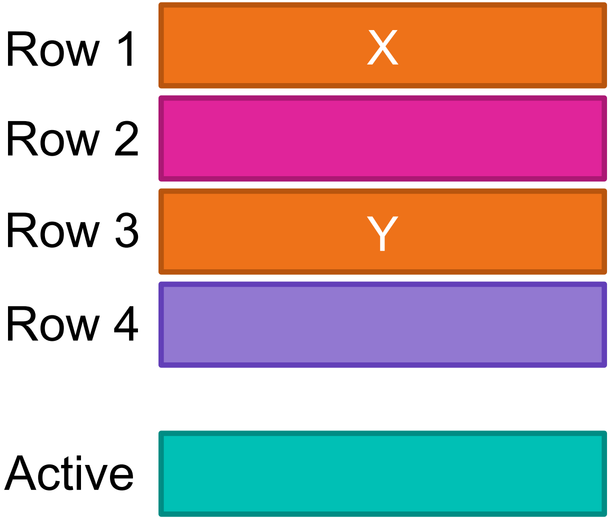
Active

Say X is somewhere in Row 1 and Y in Row 3



```
code1a:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp code1a
```

Double Sided Rowhammer



Row 1 X

Row 2

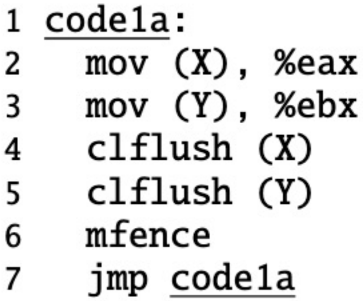
Row 3 Y

Row 4

Active

If you row hammer 1 and 3... you'll still get bit-flips in adjacent rows 2 and 4...

But 2 is adjacent to both—much more likely to induce errors quicker!



```
code1a:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp code1a
```

Lets write an exploit!

If we can get a variable held in two different rows in one bank we can induce random (but consistent) single bit errors in surrounding rows

…but its hard to know how virtual memory translates to *real* memory locations.

…so there will be an element of getting lucky with memory locations

…but we only need to get lucky once

…and can increase our luck by running many things at once!

NaCL

Sandbox for C/C++ code that aims to make things safe!

Sandbox runs with privileges

- Checks whether a chunk of code is safe to run…
- If so, loads it into memory aligned on 32B boundaries

```
andl $~31, %eax // Truncate address to 32 bits and mask to be 32-byte-aligned.
addq %r15, %rax // Add %r15, the sandbox base address.

jmp *%rax       // Indirect jump.
```

If we can corrupt this, maybe we can run unsafe hidden code...

Variadic Instruction Sets

No requirement for aligning instructions in x86-architecture.

Different instructions have different lengths...

- Some have multiple lengths!

```
20ea0: 48 b8 0f 05 eb 0c f4 f4 f4 f4     movabs rax,0xf4f4f4f40ceb050f

20ea2:       0f 05                       syscall
20ea4:             eb 0c                 jmp 0xe
```

NaCL Exploit

Code section is readable by loaded processes!

- So we can spot when the Rowhammering has been successful

Attack:

- Load a sequence of safe code that have unsafe instruction-sequences at 1-bit different offsets
- Rowhammer NaCL's code loading code (load NaCL into memory a lot and hope we get lucky)
- Either we get an invalid instruction sequence...
  - We crash
- Or we Rowhammer the kernels memory accidentally
  - Whoops... we crash.
- Or we get lucky…

Sooner or later we get lucky and then we can start to escape the sandbox

NaCL Exploit

There are links in the course webpages to Mark Seaborn and Thomas Dullien's papers, and the CMU ICSA paper…

- Do read them/watch the talks...
- They're good and this stuff is fiddly!

Preventing Rowhammer

Buy better RAM?

- But how do you tell?

With ECC?

- Can fix single bit errors, and reboot on 2 bit errors (and probably be exploitable on 3-bit errors)
- Also slower and more expensive

  …for a server it is worth it, for a laptop 🤷‍♂️

Which you refresh faster?

- Slows it down more

And which refreshes neighboring rows more often?

- More recent DRAM standards do this... but more expensive