Side-Channel Security

Chapter 6: Software-based Fault Attacks

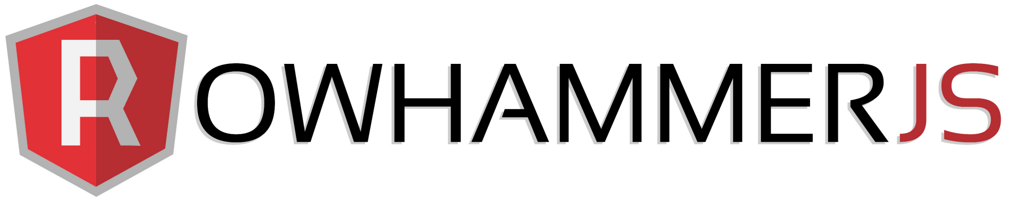
Jonas Juffinger

April 11, 2024

www.iaik.tugraz.at

ROWHAMMER

CTV



ROOT privileges for web apps!

Test - Mozilla Firefox (on lab02)

Test

file:///home/dgruss/rowhammerjs/rowhammer.html

Search

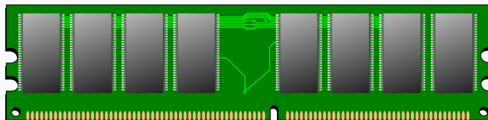
320: 12
330: 9
340: 1
350: 0
360: 1
370: 2
380: 199
390: 76
400: 72
410: 231
420: 572
1250
[!] Found flip (254 != 255) at array index 340021386 when hammering indices 339881984 and 340156416

[!] Found flip (239 != 255) at array index 340022176 when hammering indices 339881984 and 340156416

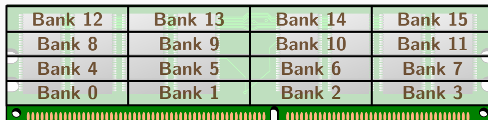
[!] Found flip (191 != 255) at array index 340023138 when hammering indices 339881984 and 340156416

[!] Found flip (254 != 255) at array index 340025146 when hammering indices 339881984 and 340156416

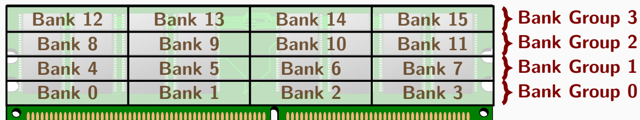
DRAM organization

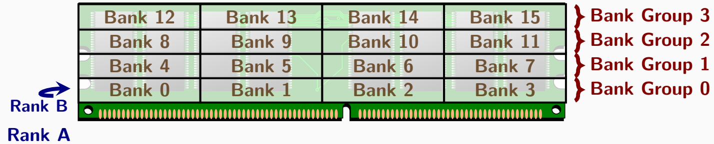


DRAM organization

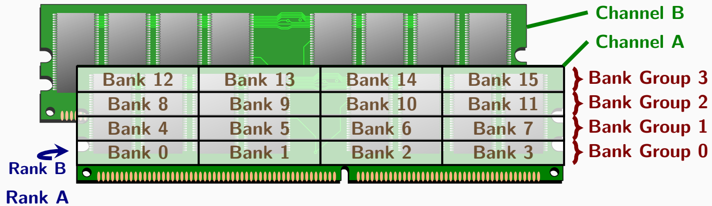


DRAM organization

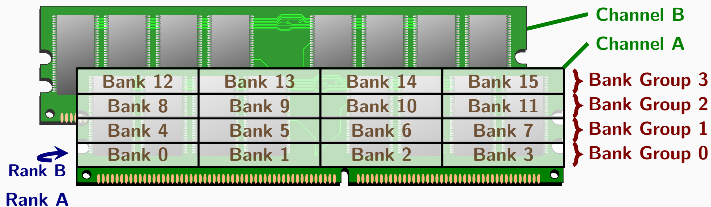



Bank 12
Bank 13
Bank 14
Bank 15
Bank 8
Bank 9
Bank 10
Bank 11
Bank 4
Bank 5
Bank 6
Bank 7
Bank 0
Bank 1
Bank 2
Bank 3
Bank Group 3
Bank Group 2
Bank Group 1
Bank Group 0
Rank B
Rank A

DRAM organization

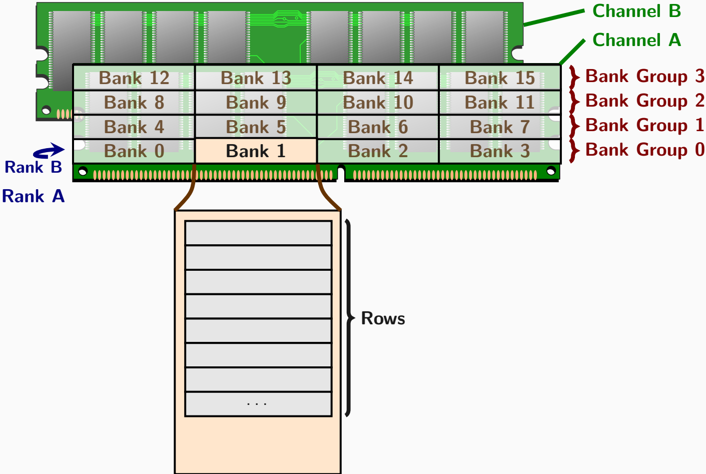


DRAM organization

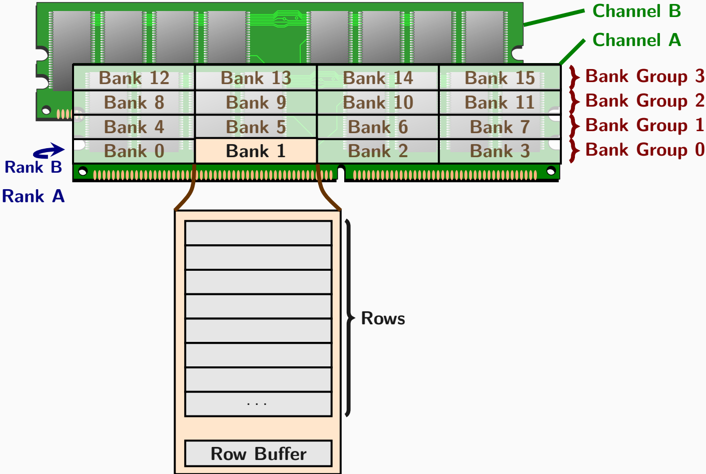


4 banks $\times$ 4 bank groups $\times$ 2 ranks $\times$ 2 channels $=$ 64 banks total

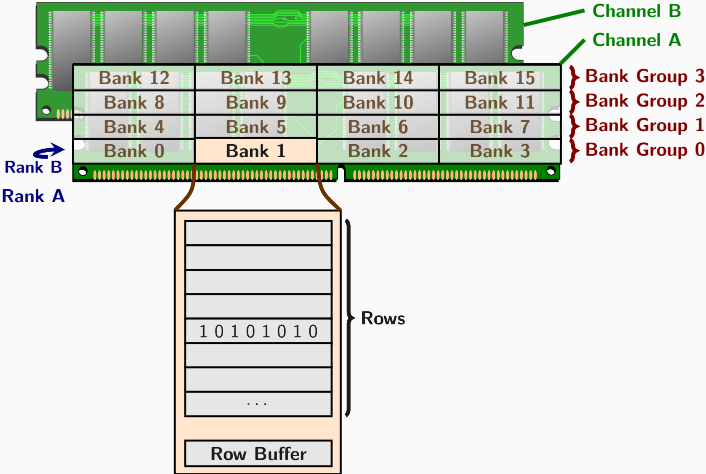
DRAM organization

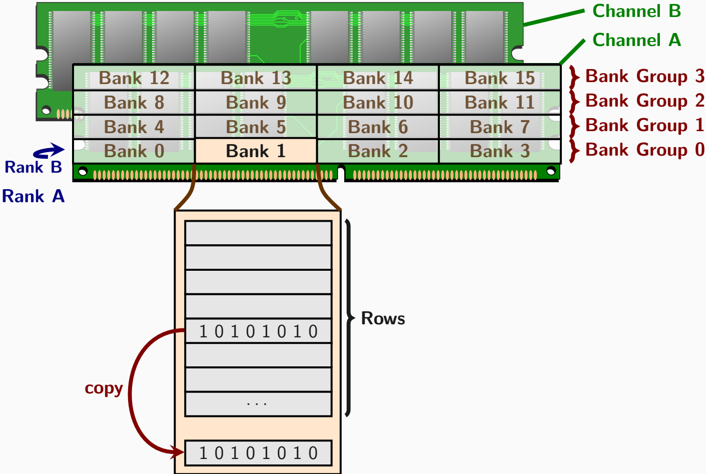


DRAM organization

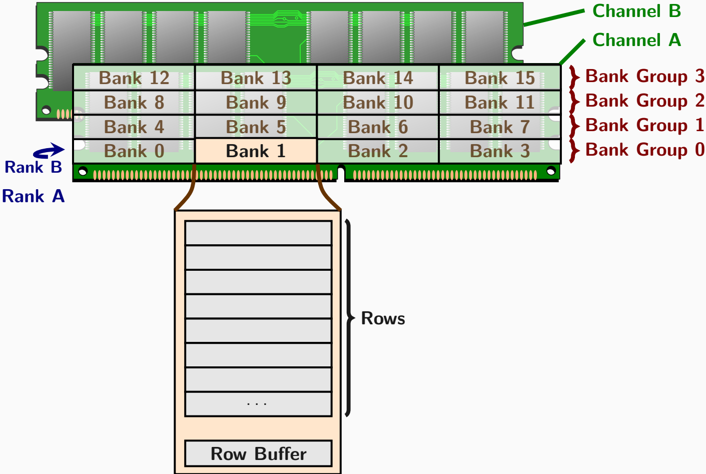


DRAM organization

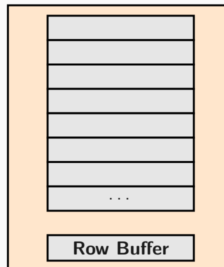



Channel B
Channel A
Bank 12
Bank 13
Bank 14
Bank 15
Bank 8
Bank 9
Bank 10
Bank 11
Bank 4
Bank 5
Bank 6
Bank 7
Bank 0
Bank 1
Bank 2
Bank 3
Bank Group 3
Bank Group 2
Bank Group 1
Bank Group 0
Rank B
Rank A
1 0 1 0 1 0 1 0
...
Rows
copy
1 0 1 0 1 0 1 0

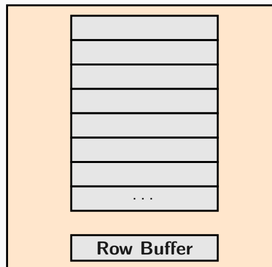
DRAM organization



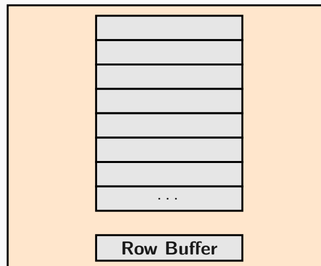
DRAM organization

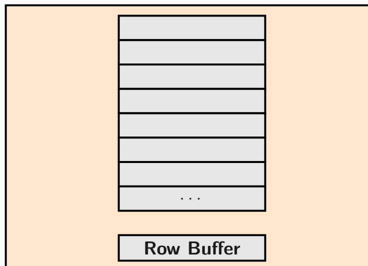


DRAM organization

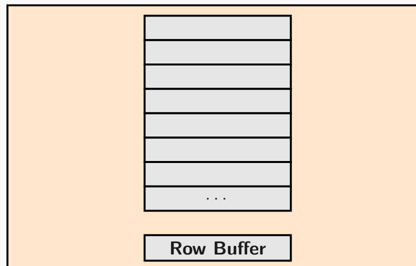


DRAM organization

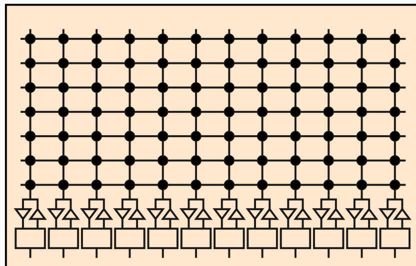



...
Row Buffer

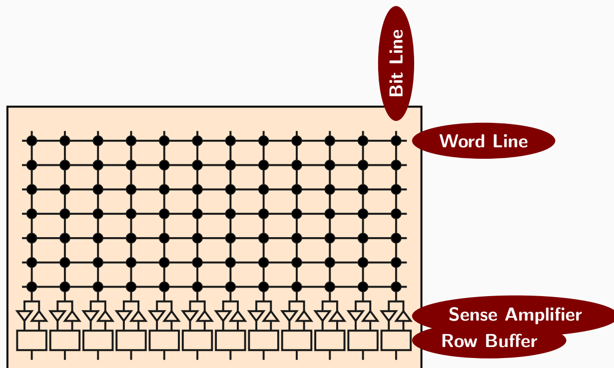
DRAM organization



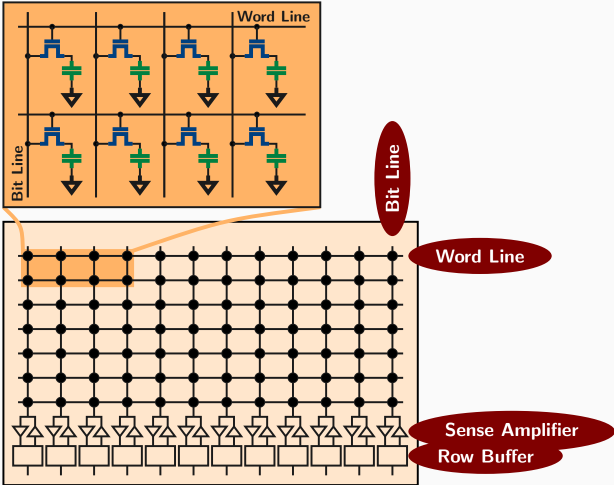
DRAM organization

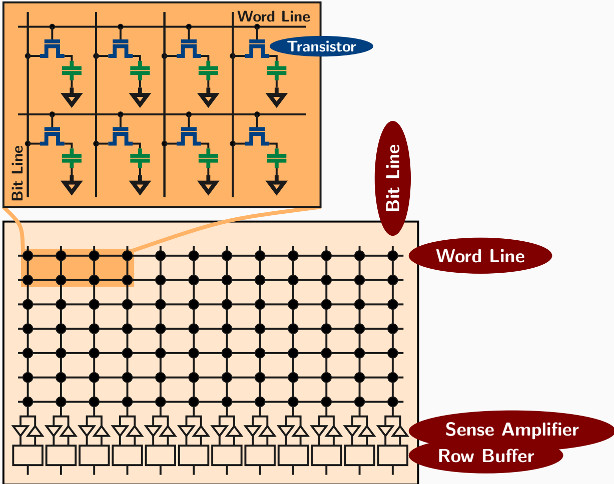


DRAM organization

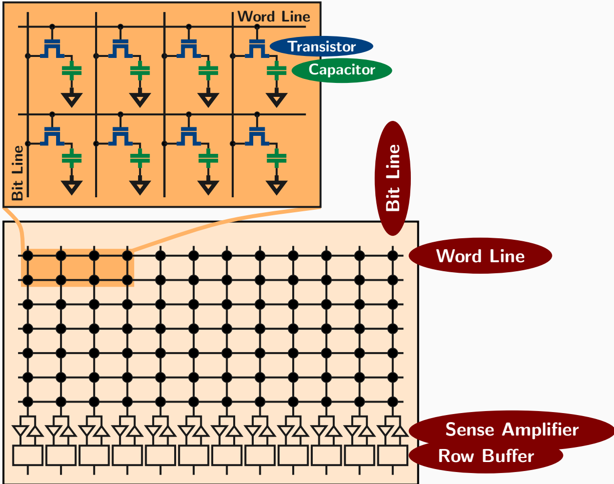


DRAM organization

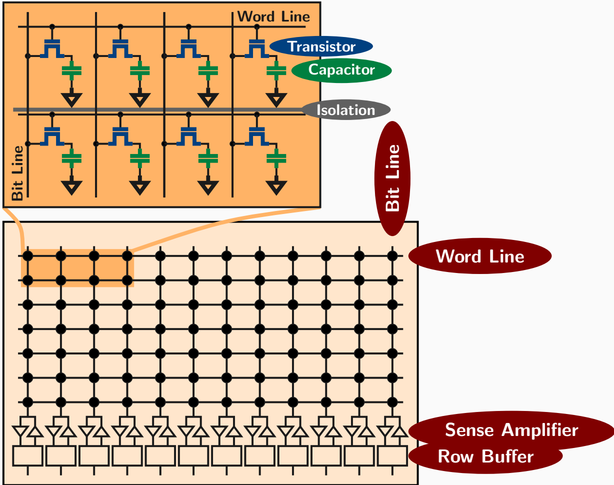



Word Line
Transistor
Bit Line
Bit Line
Word Line
Sense Amplifier
Row Buffer

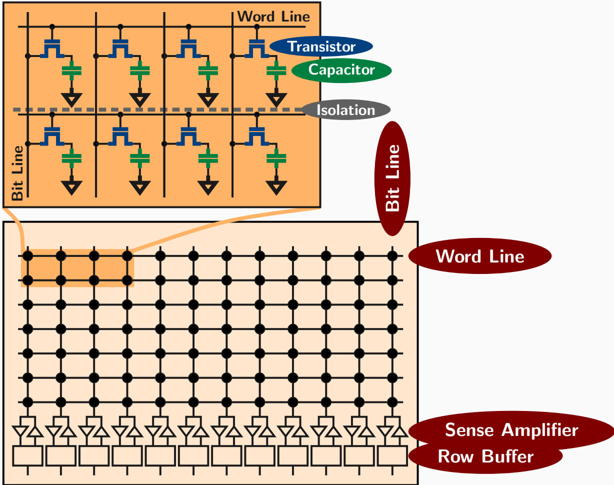
DRAM organization



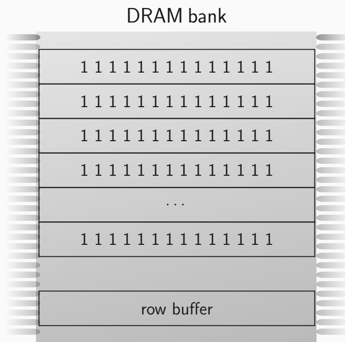
DRAM organization



DRAM organization

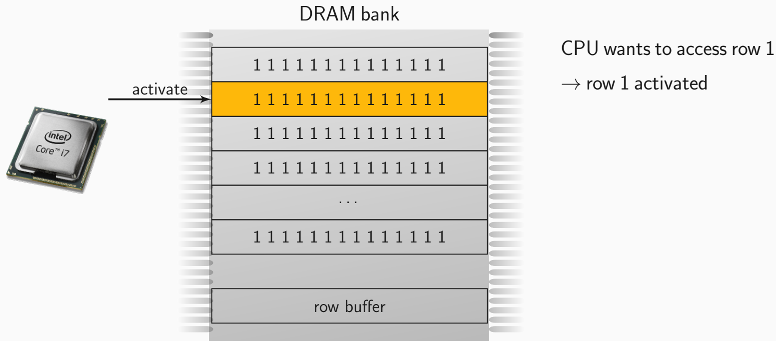


How reading from DRAM works



CPU wants to access row 1

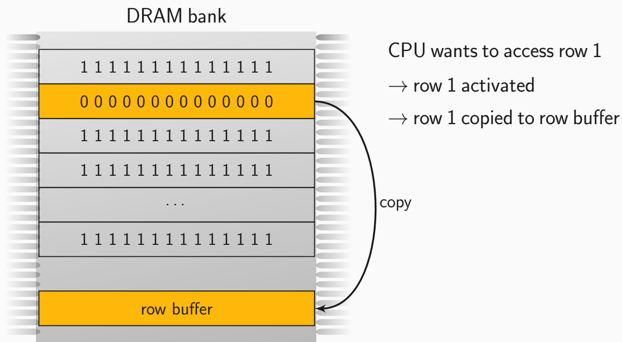
How reading from DRAM works



CPU wants to access row 1

→ row 1 activated

How reading from DRAM works

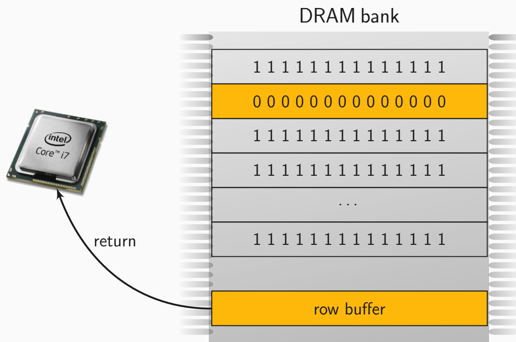


CPU wants to access row 1

→ row 1 activated

→ row 1 copied to row buffer

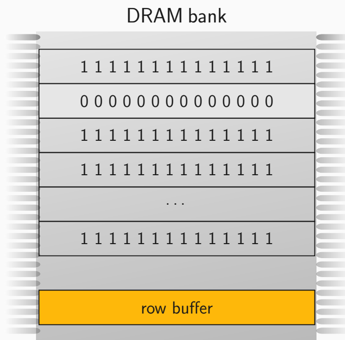
How reading from DRAM works



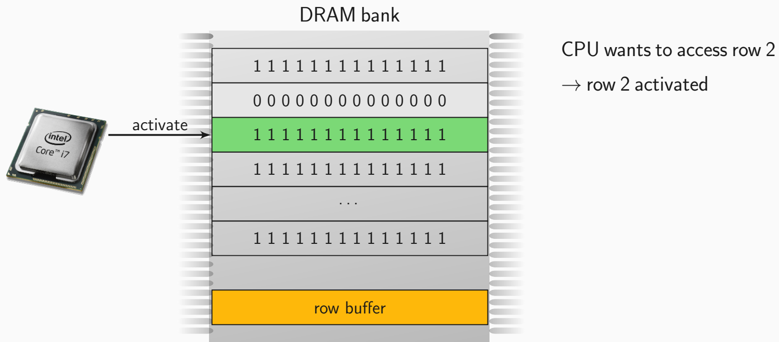
CPU wants to access row 1

→ row 1 activated

→ row 1 copied to row buffer



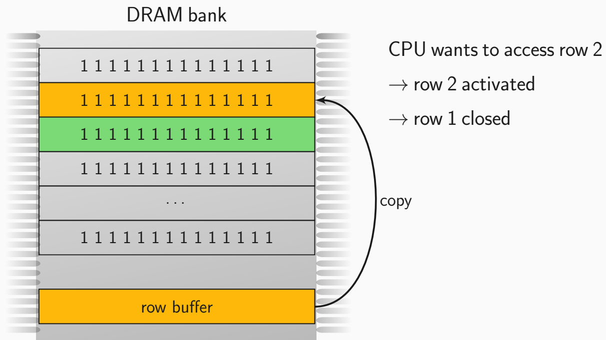
CPU wants to access row 2



CPU wants to access row 2

→ row 2 activated

How reading from DRAM works

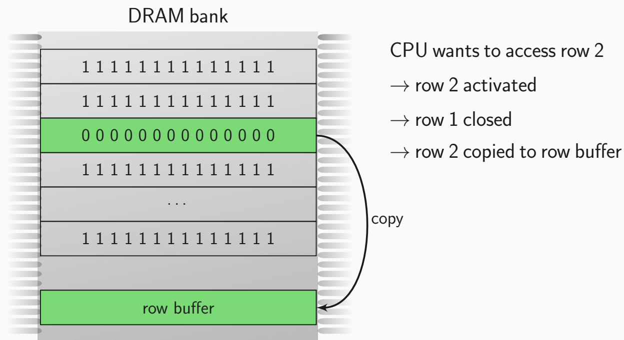


CPU wants to access row 2

→ row 2 activated

→ row 1 closed

How reading from DRAM works



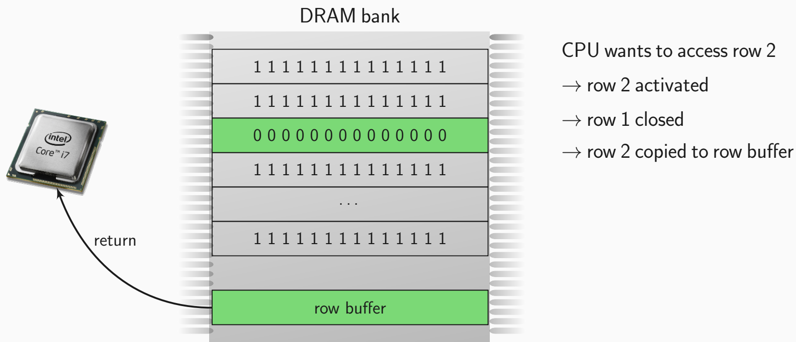
CPU wants to access row 2

→ row 2 activated

→ row 1 closed

→ row 2 copied to row buffer

How reading from DRAM works



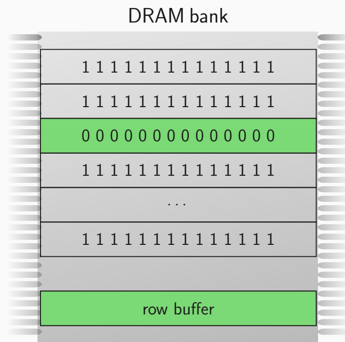
CPU wants to access row 2

→ row 2 activated

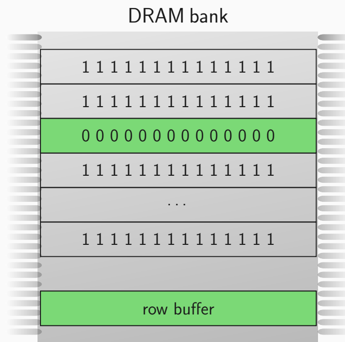
→ row 1 closed

→ row 2 copied to row buffer

How reading from DRAM works

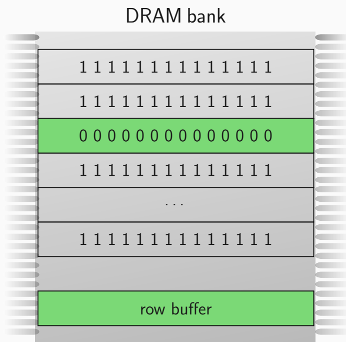


→ slow (row conflict)



CPU wants to access row 2—again

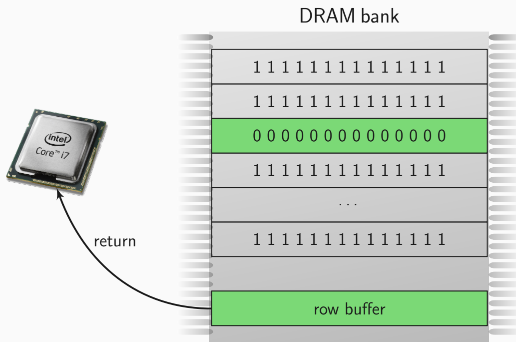
How reading from DRAM works



CPU wants to access row 2—again

→ row 2 already in row buffer

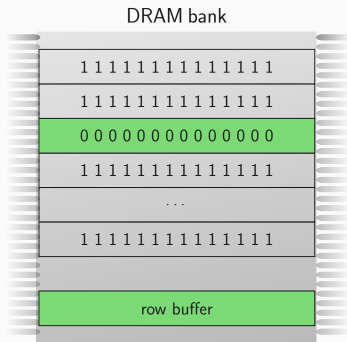
How reading from DRAM works



CPU wants to access row 2—again

→ row 2 already in row buffer

How reading from DRAM works

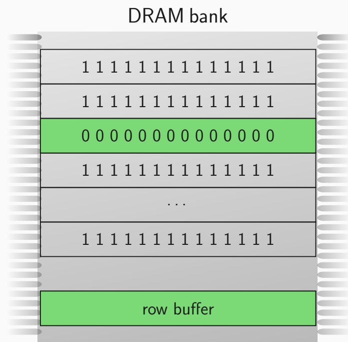


CPU wants to access row 2—again

→ row 2 already in row buffer

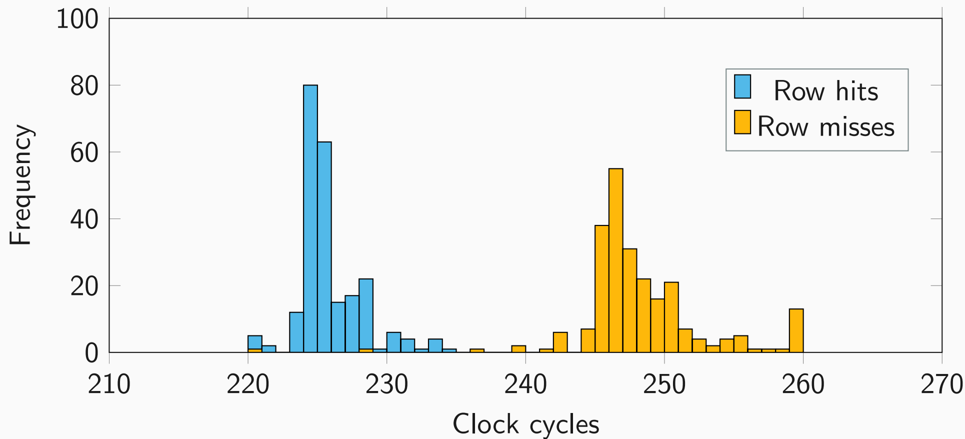
→ fast (row hit)

How reading from DRAM works

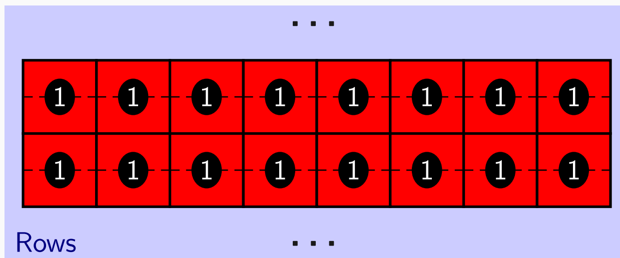


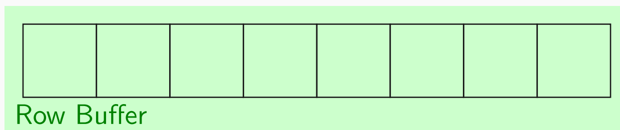
row buffer = cache

Timing difference

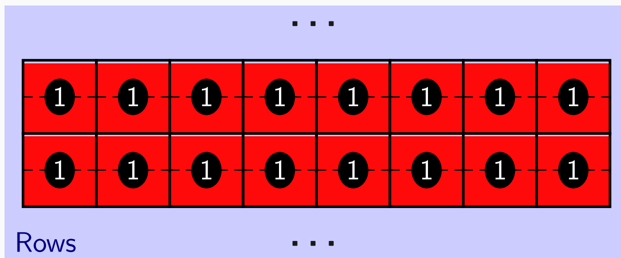


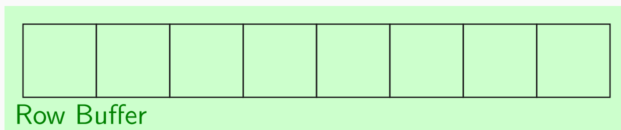
Cell Discharge

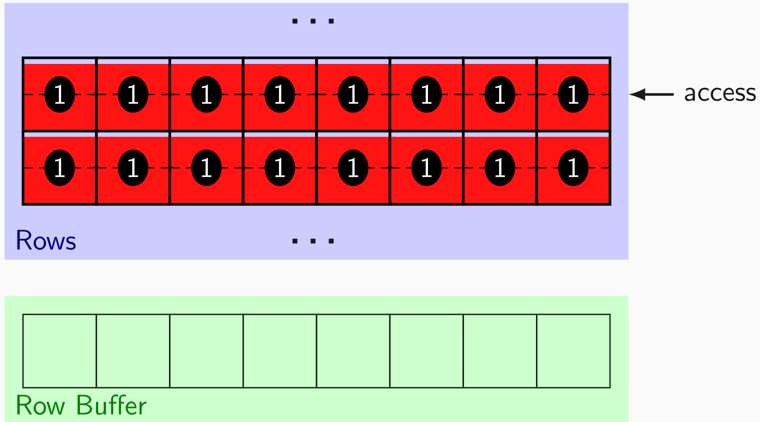




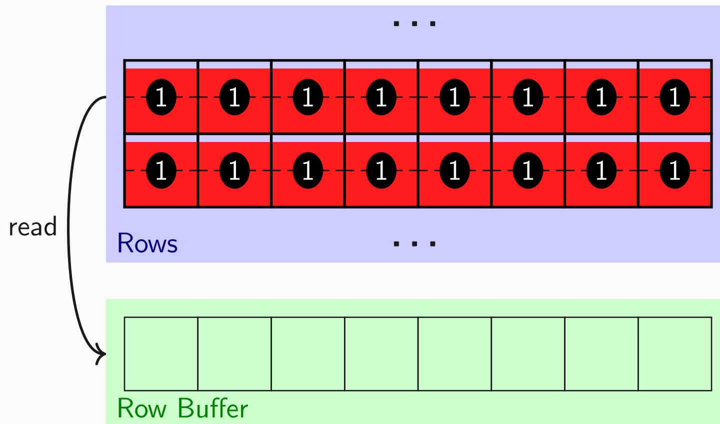
Cell Discharge

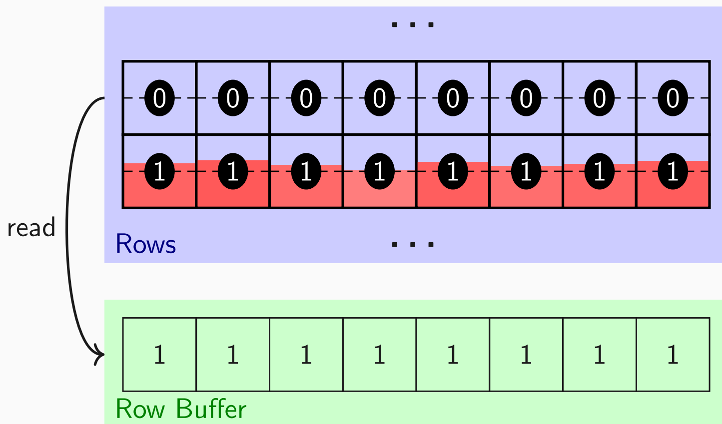




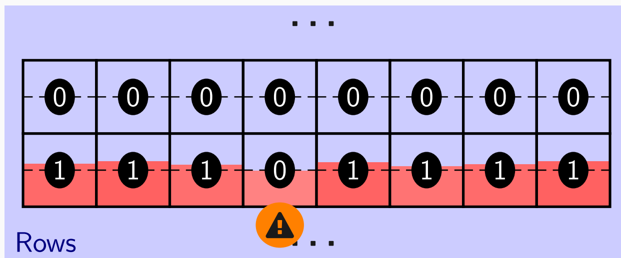

...
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
access
Rows
...
Row Buffer

Cell Discharge

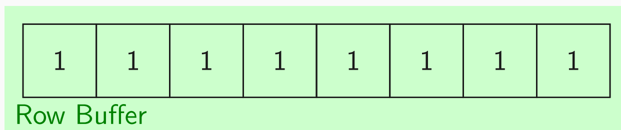




...
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
read
Rows
...
1 1 1 1 1 1 1 1
Row Buffer

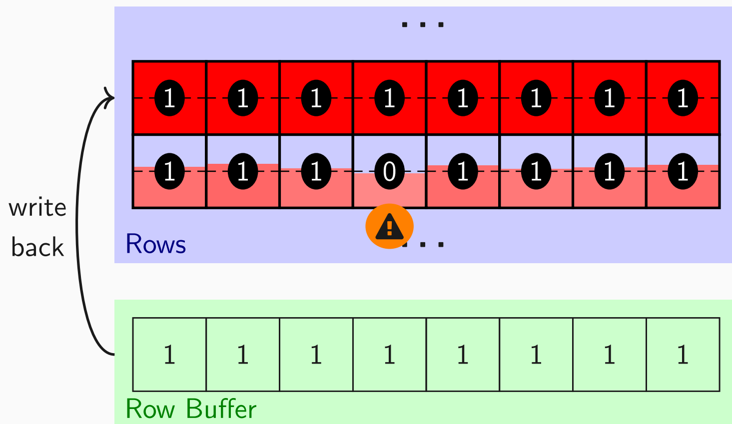


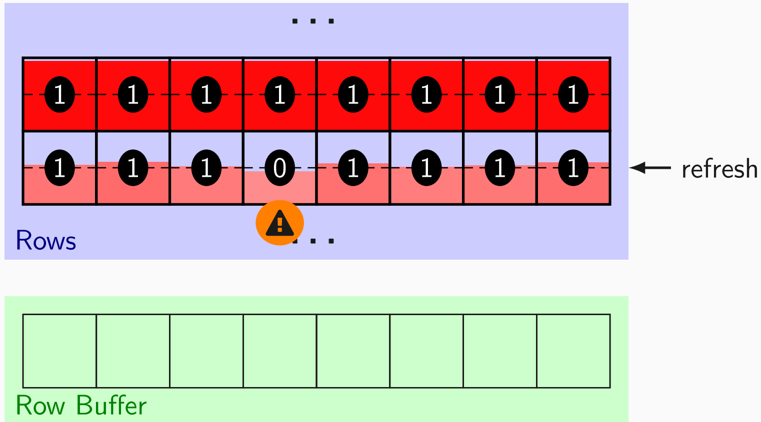
...
0 0 0 0 0 0 0 0
1 1 1 0 1 1 1 1
Rows
...



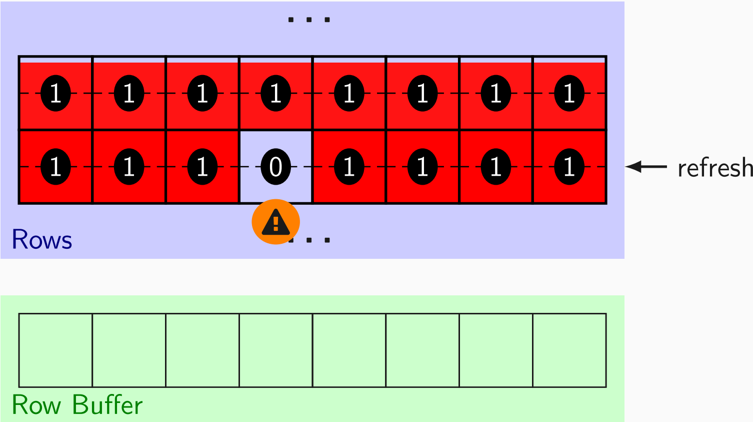
1 1 1 1 1 1 1 1
Row Buffer

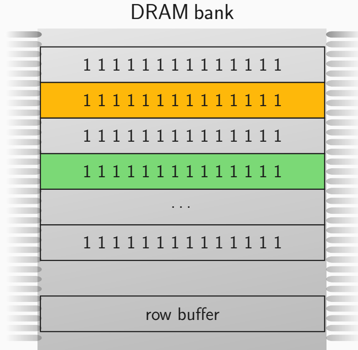
Cell Discharge

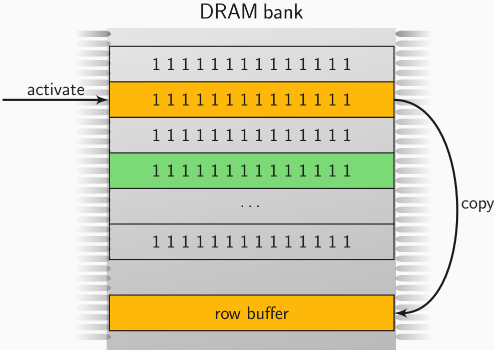


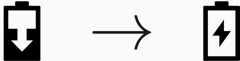

...
1 1 1 1 1 1 1 1
1 1 1 0 1 1 1 1
refresh
Rows
...
Row Buffer

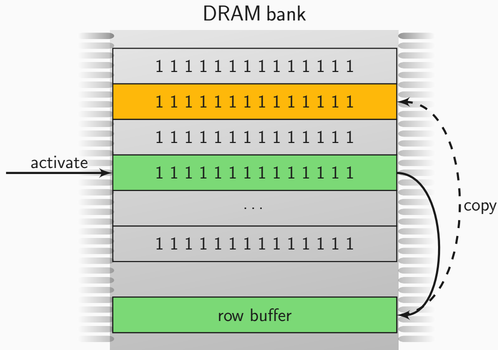
Cell Discharge

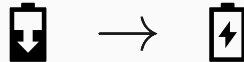



DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
...
1 1 1 1 1 1 1 1 1 1 1 1 1 1
row buffer

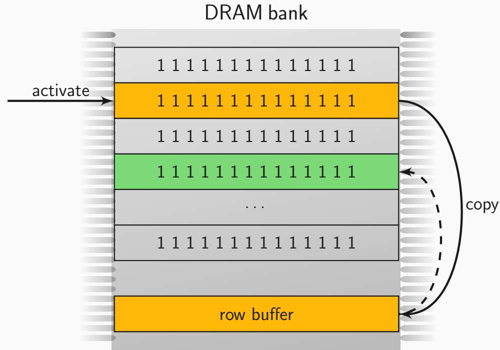

DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . .
1 1 1 1 1 1 1 1 1 1 1 1 1 1
row buffer
copy

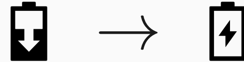




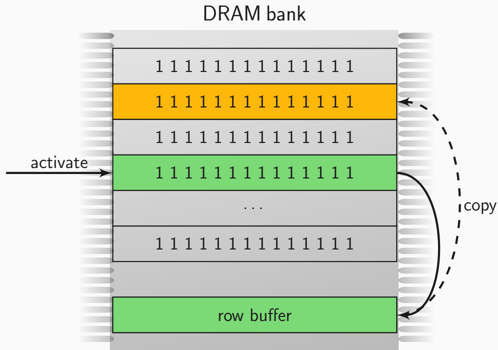


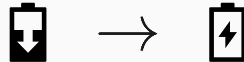
Cells leak faster upon proximate accesses → Rowhammer



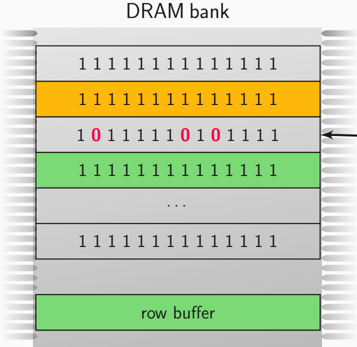


Cells leak faster upon proximate accesses → Rowhammer

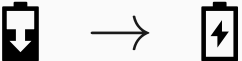




Cells leak faster upon proximate accesses → Rowhammer



bit flips in row 2!



Cells leak faster upon proximate accesses → Rowhammer

How widespread is the issue?

How widespread is the issue?

How widespread is the issue?

- 85% affected [20] (see Figure)

How widespread is the issue?

- 85% affected [20] (see Figure)

- First believed to be safe

How widespread is the issue?

DDR3

- 85% affected [20] (see Figure)

DDR4

- First believed to be safe
- $>$ 90% affected (with active countermeasures) [18]

DDR5

How widespread is the issue?

DDR3

- 85% affected [20] (see Figure)

DDR4

- First believed to be safe
- $>$ 90% affected (with active countermeasures) [18]

DDR5

- Not yet know

How widespread is the issue?

DDR3

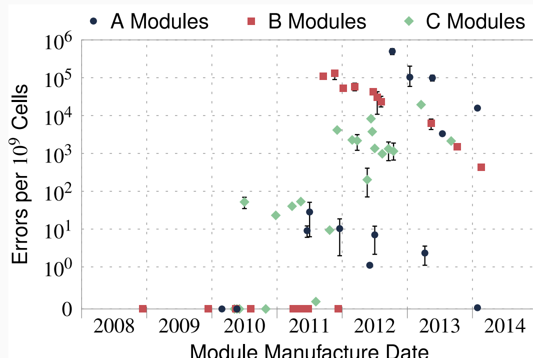
- 85% affected [20] (see Figure)

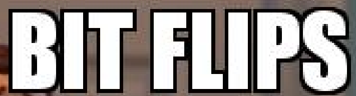
DDR4

- First believed to be safe
- $>$ 90% affected (with active countermeasures) [18]

DDR5

- Not yet know




BIT FLIPS
BIT FLIPS EVERYWHERE

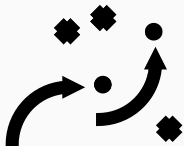
Requirements

Memory accesses must be

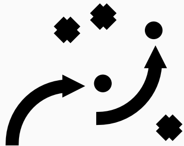
- uncached: reach DRAM
- fast: race against the next row refresh
- targeted: reach specific row

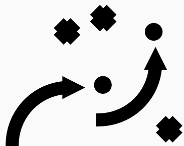
How do we get enough uncached accesses?

Access techniques

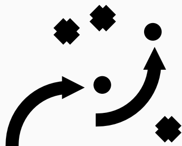


Access techniques



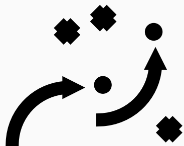


- `clflush` instruction → original paper [20]



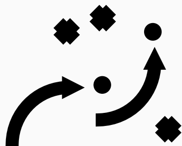
- `clflush` instruction → original paper [20]
- cache eviction [2, 13]

Access techniques



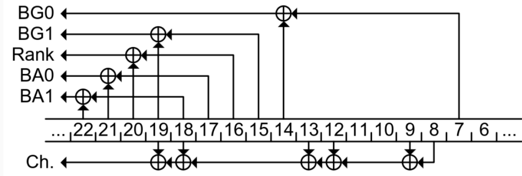
- `clflush` instruction $\rightarrow$ original paper [20]
- cache eviction [2, 13]
- non-temporal accesses [27]

Access techniques

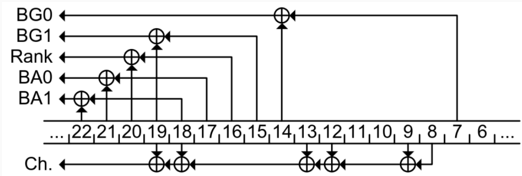


- `clflush` instruction → original paper [20]
- cache eviction [2, 13]
- non-temporal accesses [27]
- uncached memory [33]

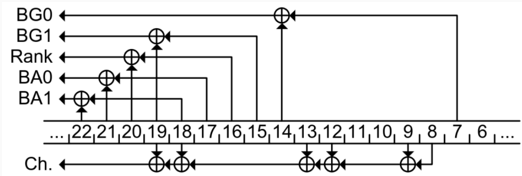
How do we target accesses?



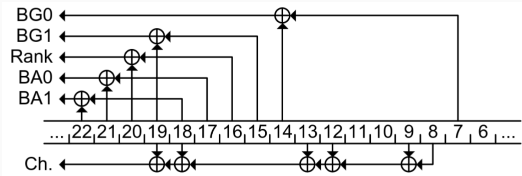
- fixed map: physical addresses → DRAM cells



- fixed map: physical addresses → DRAM cells
- undocumented for Intel



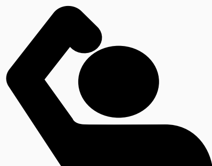
- fixed map: physical addresses → DRAM cells
- undocumented for Intel
- reverse-engineering for Sandy Bridge, Sandy, Ivy, Haswell, Skylake, ... [25, 30]



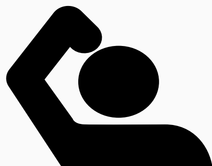
- fixed map: physical addresses → DRAM cells
- undocumented for Intel
- reverse-engineering for Sandy Bridge, Sandy, Ivy, Haswell, Skylake, . . . [25, 30]
- using the timing difference between row hits and row conflicts

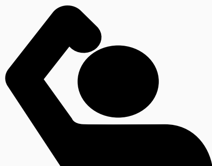
How to exploit random bit flips?

How to exploit random bit flips?



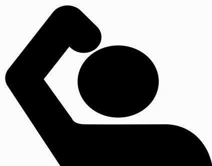
How to exploit random bit flips?





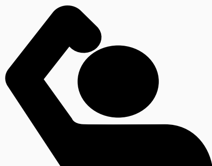
- They are not random → highly reproducible flip pattern!

How to exploit random bit flips?

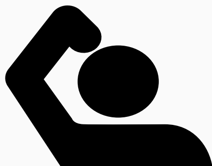


- They are not random → highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations

How to exploit random bit flips?

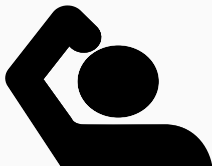


- They are not random → highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations
  2. Scan for "good" flips



- They are not random → highly reproducible flip pattern!
    1. Choose a data structure that you can place at arbitrary memory locations
    2. Scan for “good” flips
    3. Place data structure there

How to exploit random bit flips?



- They are not random $\rightarrow$ highly reproducible flip pattern!
  1. Choose a data structure that you can place at arbitrary memory locations
  2. Scan for “good” flips
  3. Place data structure there
  4. Trigger bit flip again

Page Table Entries

P	RW	US	WT	UC	R	D	S	G		
										X

Page Table Entries

P	RW	US	WT	UC	R	D	S	G	Ignored	
				Ignored						X

Page Table Entries

P	RW	US	WT	UC	R	D	S	G	Ignored	

Physical Page Number

	Ignored	X

Page Table Entries

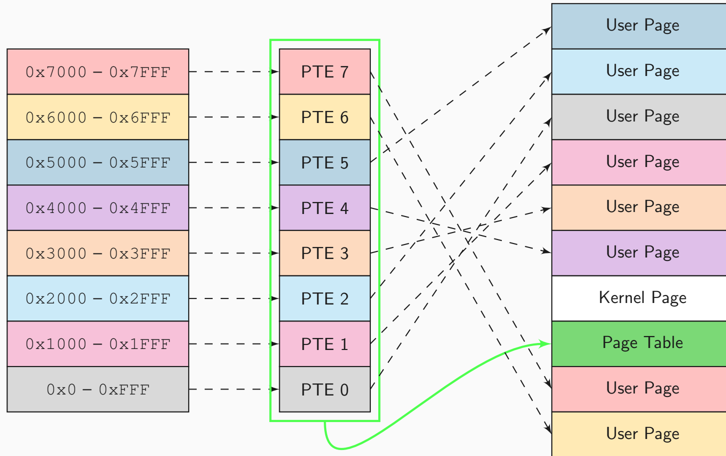
P	RW	US	WT	UC	R	D	S	G	Ignored	

Physical Page Number

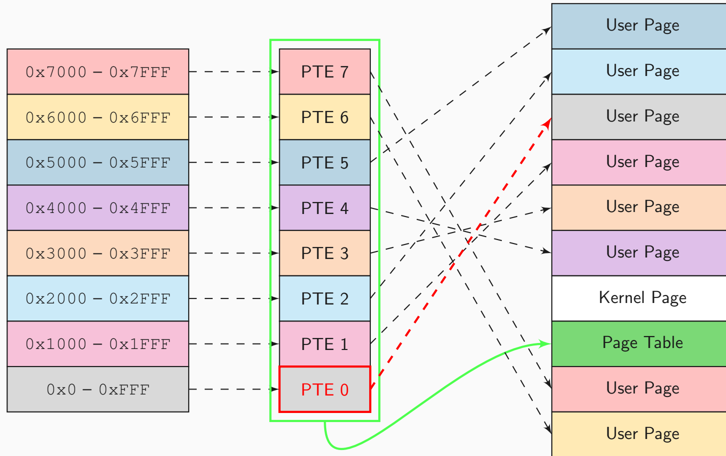
	Ignored	X

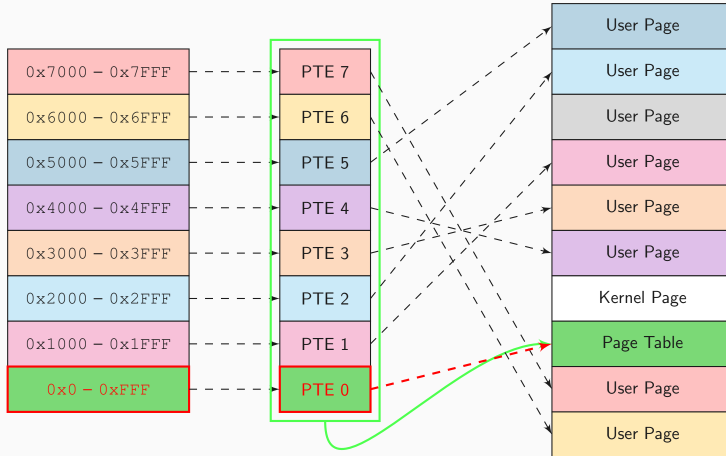
Each 4 KB page table consists of 512 such entries

Page Table Manipulation

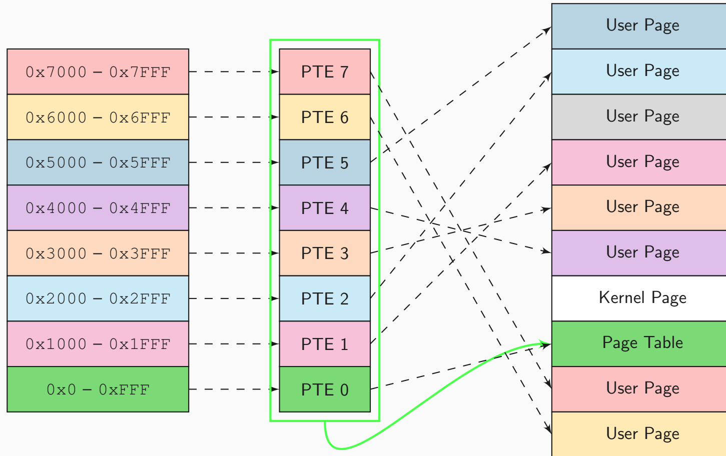


Page Table Manipulation

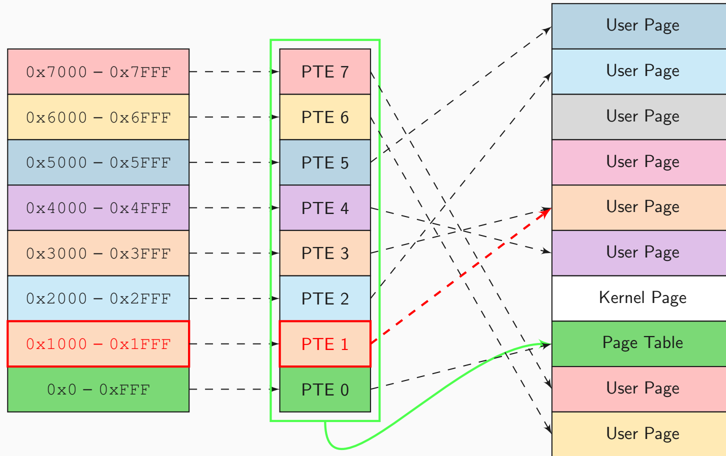


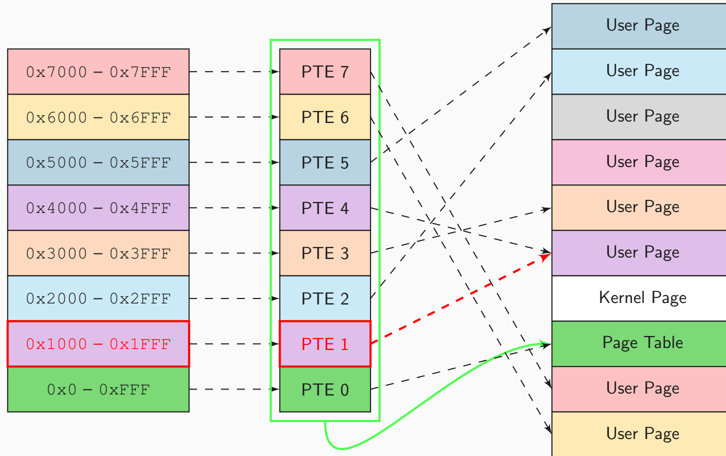

0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF
PTE 7
PTE 6
PTE 5
PTE 4
PTE 3
PTE 2
PTE 1
PTE 0
User Page
User Page
User Page
User Page
User Page
User Page
Kernel Page
Page Table
User Page
User Page

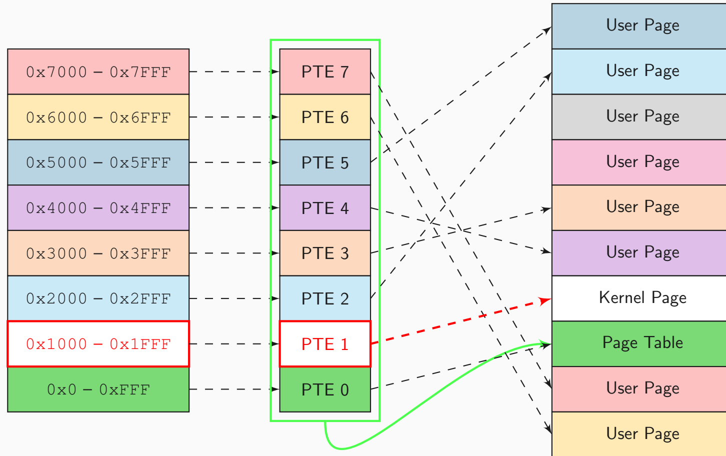
Page Table Manipulation

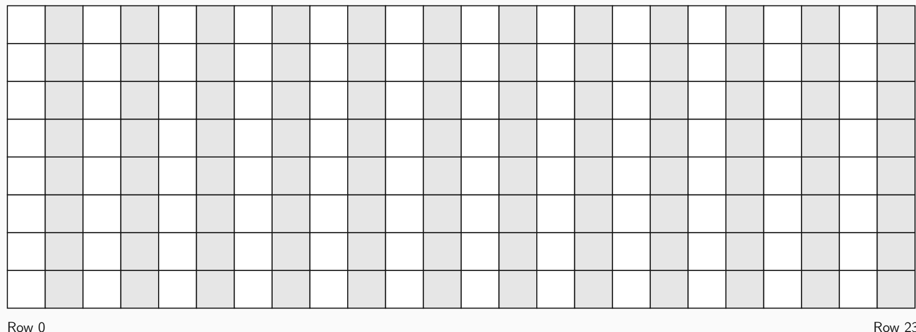


Page Table Manipulation

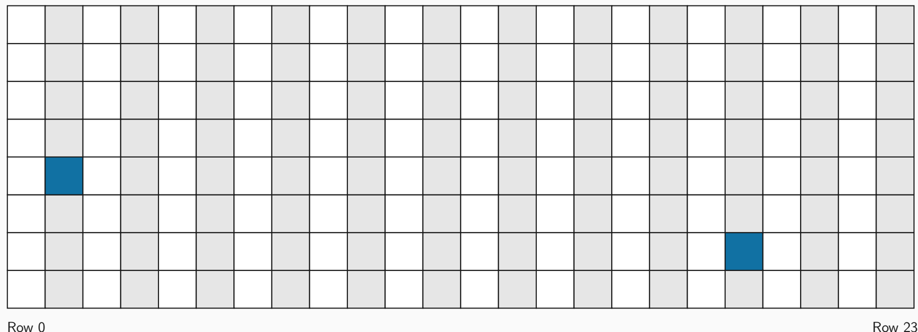



0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF
PTE 7
PTE 6
PTE 5
PTE 4
PTE 3
PTE 2
PTE 1
PTE 0
User Page
User Page
User Page
User Page
User Page
User Page
Kernel Page
Page Table
User Page
User Page


0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF
PTE 7
PTE 6
PTE 5
PTE 4
PTE 3
PTE 2
PTE 1
PTE 0
User Page
User Page
User Page
User Page
User Page
User Page
Kernel Page
Page Table
User Page
User Page

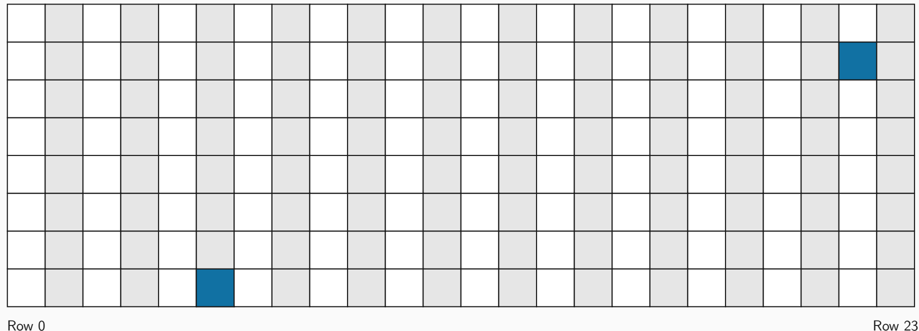


Hammering memory locations in different rows



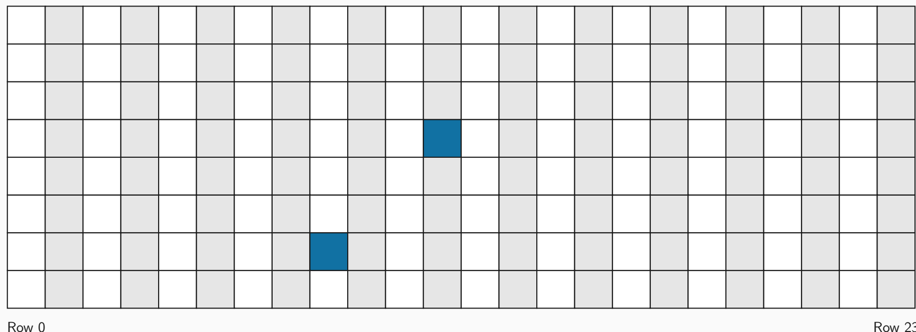
Hammering memory locations in different rows

Search for page with flip



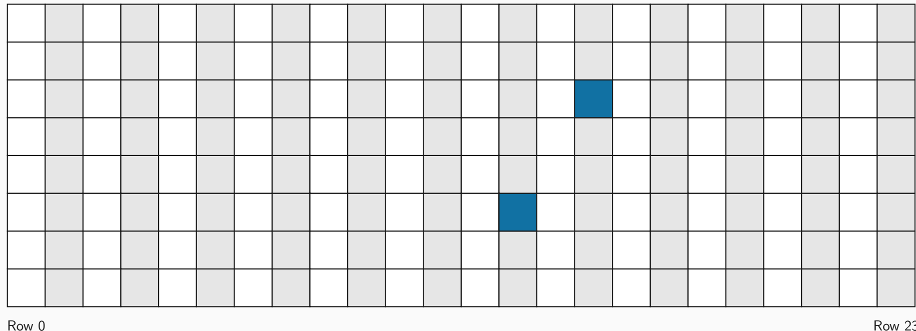
Hammering memory locations in different rows

Search for page with flip

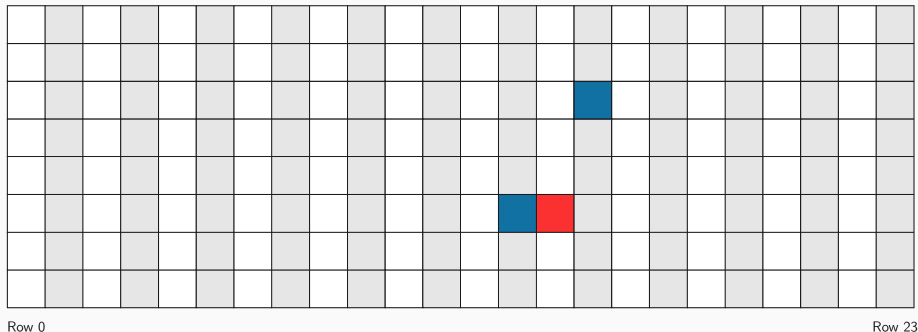


Hammering memory locations in different rows

Search for page with flip

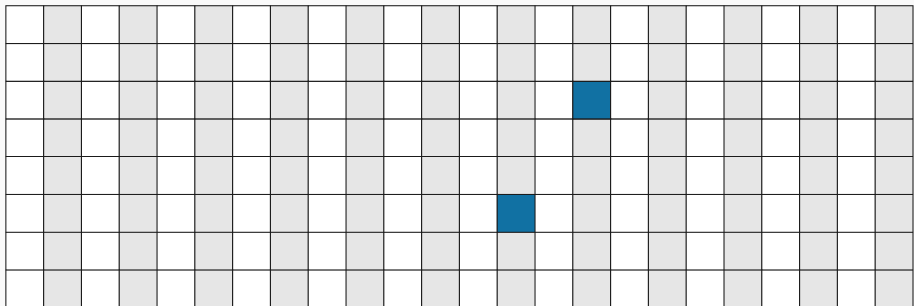


Hammering memory locations in different rows

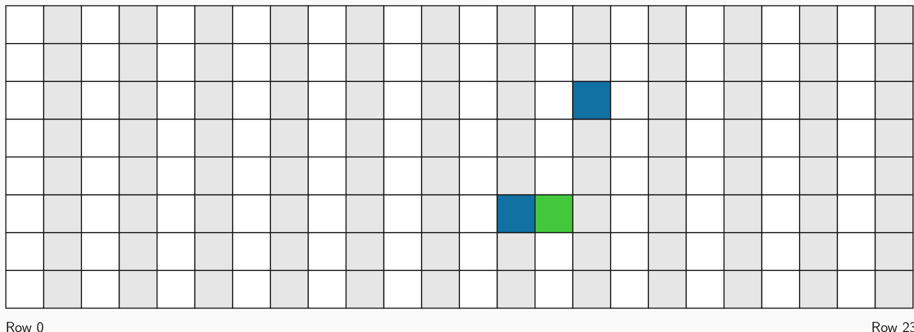


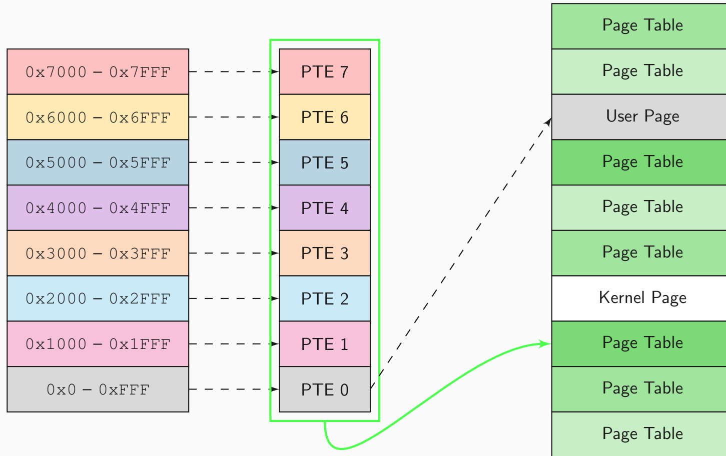
Hammering memory locations in different rows

Fill all remaining memory with page tables



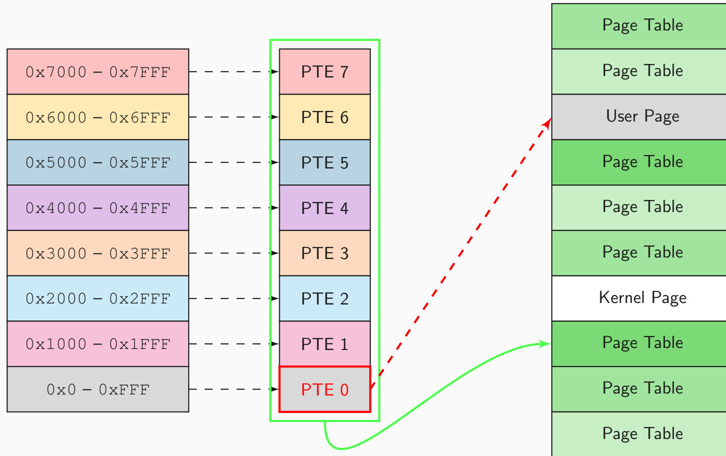
Fill all remaining memory with page tables



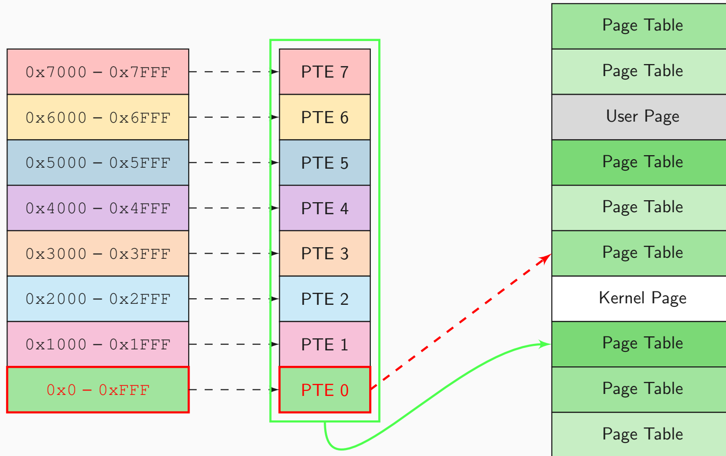


0x7000 – 0x7FFF
0x6000 – 0x6FFF
0x5000 – 0x5FFF
0x4000 – 0x4FFF
0x3000 – 0x3FFF
0x2000 – 0x2FFF
0x1000 – 0x1FFF
0x0 – 0xFFF
PTE 7
PTE 6
PTE 5
PTE 4
PTE 3
PTE 2
PTE 1
PTE 0
Page Table
Page Table
User Page
Page Table
Page Table
Page Table
Kernel Page
Page Table
Page Table
Page Table

Page Table Manipulation

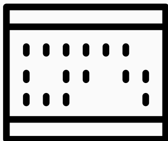


Page Table Manipulation

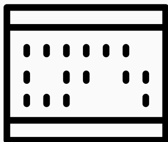


Strategy: Flipping Page Table PPN bits

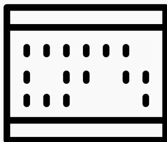
Strategy: Flipping Page Table PPN bits



Strategy: Flipping Page Table PPN bits

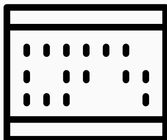


Strategy: Flipping Page Table PPN bits



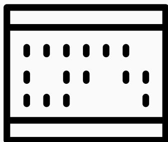
1. Scan for flips

Strategy: Flipping Page Table PPN bits



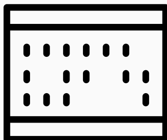
1. Scan for flips
2. Exhaust or massage memory to place a page table at target location

Strategy: Flipping Page Table PPN bits

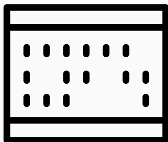


1. Scan for flips
2. Exhaust or massage memory to place a page table at target location
3. Gain access to your own page table → kernel privileges

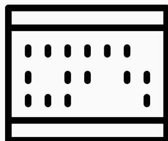
Flipping Page Table PPN bits



Flipping Page Table PPN bits

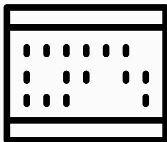


Flipping Page Table PPN bits



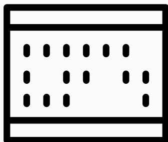
- Idea from [31]

Flipping Page Table PPN bits



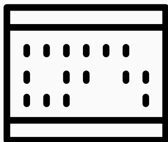
- Idea from [31]
- Same idea applied in several other works:

Flipping Page Table PPN bits



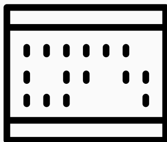
- Idea from [31]
- Same idea applied in several other works:
    - Rowhammer.js [13]

Flipping Page Table PPN bits



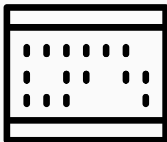
- Idea from [31]
- Same idea applied in several other works:
    - Rowhammer.js [13]
    - One bit flips, one cloud flops [35]

Flipping Page Table PPN bits



- Idea from [31]
- Same idea applied in several other works:
    - Rowhammer.js [13]
    - One bit flips, one cloud flops [35]
    - Drammer [33]

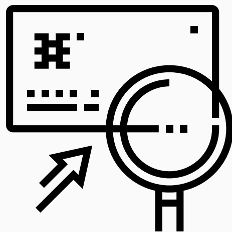
Flipping Page Table PPN bits



- Idea from [31]
- Same idea applied in several other works:
    - Rowhammer.js [13]
    - One bit flips, one cloud flops [35]
    - Drammer [33]
    - Half-Double Rowhammer [21]

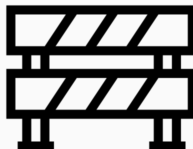
How to mitigate Rowhammer?

Different mitigations have been proposed:



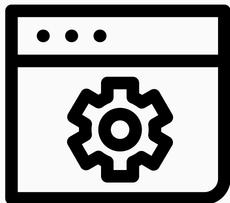
Detection

vs



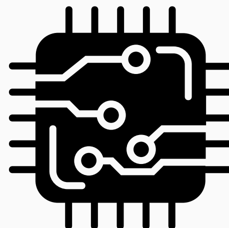
Prevention

Different mitigations have been proposed:



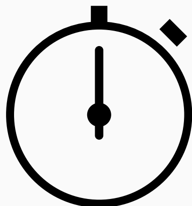
Software

vs



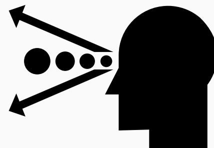
Hardware

Different mitigations have been proposed:



Short Term

vs



Long Term

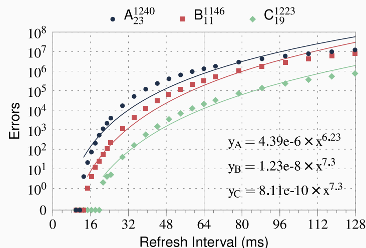
- No `clflush` instruction

- No `clflush` instruction → Rowhammer.js

Quick fixes

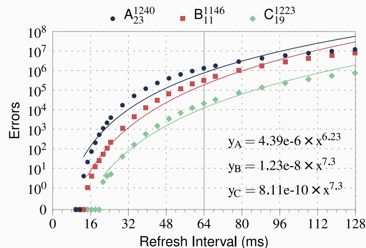
- No `clflush` instruction $\rightarrow$ Rowhammer.js
- Increase the refresh rate

- No `clflush` instruction → Rowhammer.js
- Increase the refresh rate
  - → Would need to be increased by 7× to eliminate all bit flips



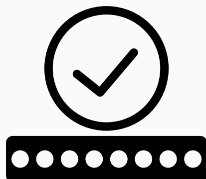
Errors depending on refresh interval [20]

- No `clflush` instruction → Rowhammer.js
- Increase the refresh rate
  - → Would need to be increased by 7× to eliminate all bit flips
  - → Implementation: increased by 2× by BIOS vendors



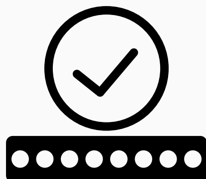
Errors depending on refresh interval [20]

What about ECC?



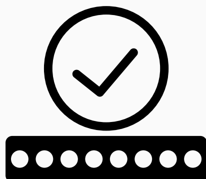
- ECC protection: server can handle or correct single bit errors

What about ECC?



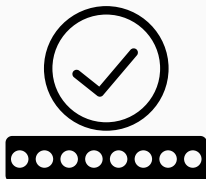
- ECC protection: server can handle or correct single bit errors
- No standard for event reporting

What about ECC?



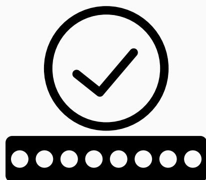
- ECC protection: server can handle or correct single bit errors
- No standard for event reporting
- In practice [22]
  - Common: server counts ECC errors and report only if they reach a threshold (e.g., $>$ 100 bit flips / hour)

What about ECC?



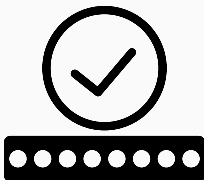
- ECC protection: server can handle or correct single bit errors
- No standard for event reporting
- In practice [22]
  - Common: server counts ECC errors and report only if they reach a threshold (e.g., $>$ 100 bit flips / hour)
  - Some server vendors never report errors to the OS

What about ECC?



- ECC protection: server can handle or correct single bit errors
- No standard for event reporting
- In practice [22]
  - Common: server counts ECC errors and report only if they reach a threshold (e.g., $>$ 100 bit flips / hour)
  - Some server vendors never report errors to the OS
  - One server did not even halt when bit flips were non-correctable

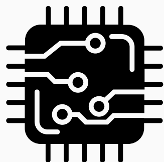
What about ECC?



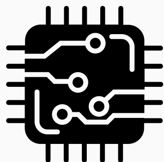
- ECC protection: server can handle or correct single bit errors
- No standard for event reporting
- In practice [22]
  - Common: server counts ECC errors and report only if they reach a threshold (e.g., $>$ 100 bit flips / hour)
  - Some server vendors never report errors to the OS
  - One server did not even halt when bit flips were non-correctable
- ECCploit actively targeting ECC DRAM with Rowhammer [0]

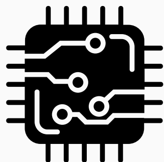
Preventing Rowhammer attacks in hardware

Preventing Rowhammer attacks in hardware



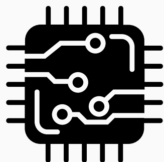
Original ideas from [20]





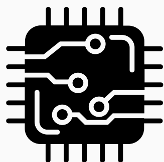
Original ideas from [20]

- Making better DRAM chips that are not vulnerable



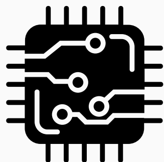
Original ideas from [20]

- Making better DRAM chips that are not vulnerable
- Using error correcting codes (ECC)



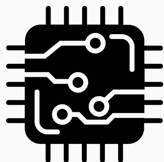
Original ideas from [20]

- Making better DRAM chips that are not vulnerable
- Using error correcting codes (ECC)
- Increasing the refresh rate



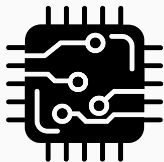
Original ideas from [20]

- Making better DRAM chips that are not vulnerable
- Using error correcting codes (ECC)
- Increasing the refresh rate
- Remapping/retiring faulty cells after manufacturing



Original ideas from [20]

- Making better DRAM chips that are not vulnerable
- Using error correcting codes (ECC)
- Increasing the refresh rate
- Remapping/retiring faulty cells after manufacturing
- Identifying hammered rows at runtime and refreshing neighbors



Original ideas from [20]

- Making better DRAM chips that are not vulnerable
- Using error correcting codes (ECC)
- Increasing the refresh rate
- Remapping/retiring faulty cells after manufacturing
- Identifying hammered rows at runtime and refreshing neighbors

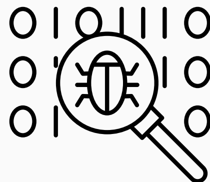
→ Expensive, performance overhead, or increased power consumption

MASCAT - Stopping Microarchitectural Attacks Before Execution [16]

- Static analysis of the binary

MASCAT - Stopping Microarchitectural Attacks Before Execution [16]

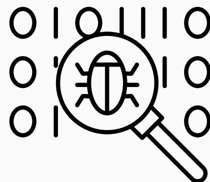
- Static analysis of the binary
- Detect suspicious instruction sequences (`clflush`, `rdtsc`, fences, ...)
- Open problem: false positives



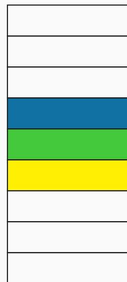
MASCAT - Stopping Microarchitectural Attacks Before Execution [16]

- Static analysis of the binary
- Detect suspicious instruction sequences (`clflush`, `rdtsc`, fences, ...)
- Open problem: false positives

ThrowHammer [32], NetHammer [23].



- B-CATT: disable vulnerable physical memory [5]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [5]



- B-CATT: disable vulnerable physical memory [5]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [5]

B-CATT: Might block 95% of RAM [12, 34]

- B-CATT: disable vulnerable physical memory [5]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [5]

B-CATT: Might block 95% of RAM [12, 34]

G-CATT: What about non-kernel or shared pages? [6, 12]

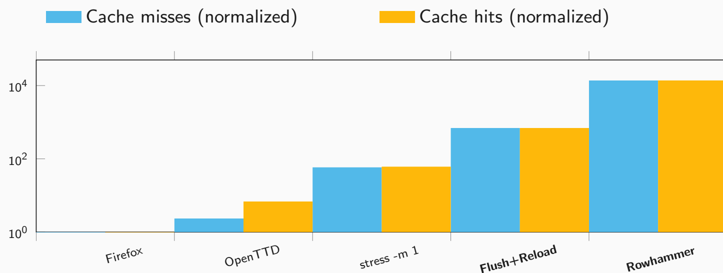
- B-CATT: disable vulnerable physical memory [5]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [5]

B-CATT: Might block 95% of RAM [12, 34]

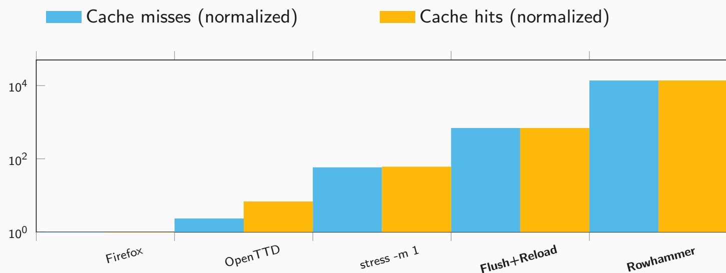
G-CATT: What about non-kernel or shared pages? [6, 12]

G-CATT: Bit flips more than 8 "rows" apart [12, 20]

- Rowhammer: lots of cache misses that can be monitored with hardware performance counters [7, 14, 15, 24]

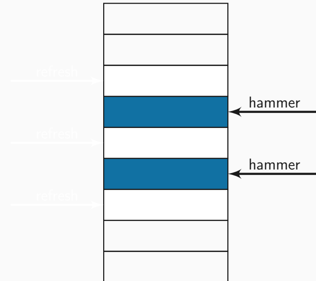


- Rowhammer: lots of cache misses that can be monitored with hardware performance counters [7, 14, 15, 24]

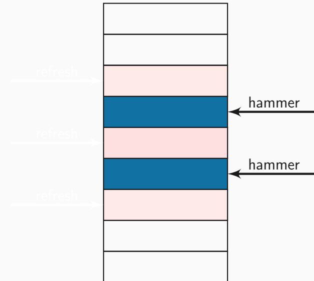


What if performance counters do not work because we run in SGX? [12, 17]

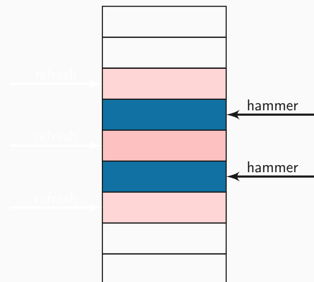
Target Row Refresh (TRR)



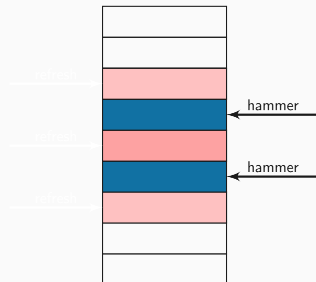
- Counter per row



- Counter per row
- Increment neighbor rows

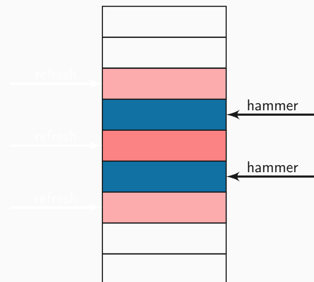


- Counter per row
- Increment neighbor rows



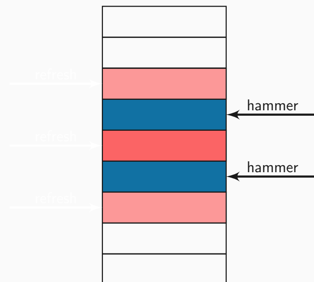
Target Row Refresh (TRR)

- Counter per row
- Increment neighbor rows



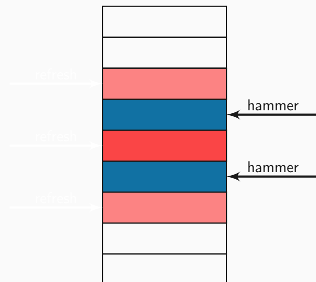
Target Row Refresh (TRR)

- Counter per row
- Increment neighbor rows

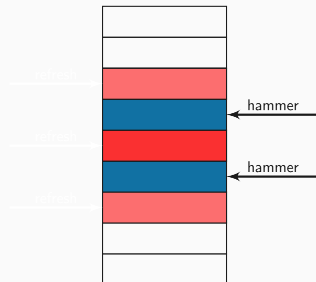


Target Row Refresh (TRR)

- Counter per row
- Increment neighbor rows

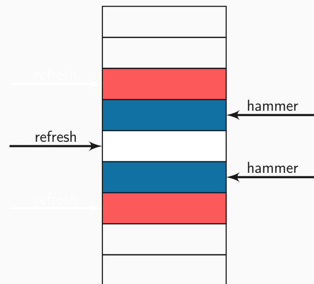


- Counter per row
- Increment neighbor rows

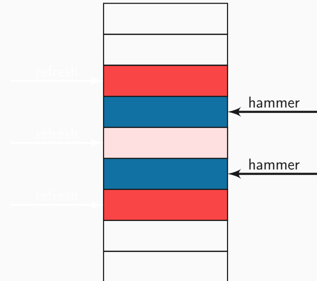


Target Row Refresh (TRR)

- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold

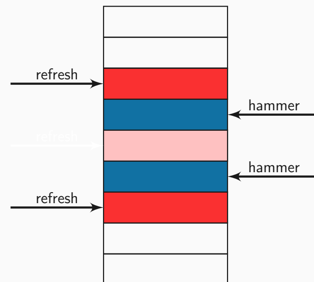


- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold



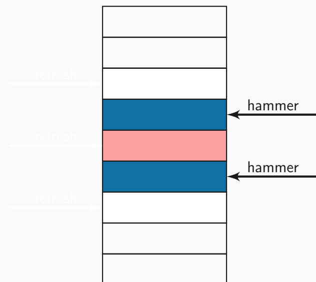
Target Row Refresh (TRR)

- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold



Target Row Refresh (TRR)

- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold



Target Row Refresh (TRR) cont.

- Counter per row too expensive

Target Row Refresh (TRR) cont.

- Counter per row too expensive
- Small number of counters

Target Row Refresh (TRR) cont.

- Counter per row too expensive
- Small number of counters
- Can be tricked to count not hammered rows

Target Row Refresh (TRR) cont.

- Counter per row too expensive
- Small number of counters
- Can be tricked to count not hammered rows
- Complex many sided access patterns

Target Row Refresh (TRR) cont.

- Counter per row too expensive
- Small number of counters
- Can be tricked to count not hammered rows
- Complex many sided access patterns
- TRRespass [11, 29]

Target Row Refresh (TRR) cont.

- Counter per row too expensive
- Small number of counters
- Can be tricked to count not hammered rows
- Complex many sided access patterns
- TRRespass [11, 29]
- Blacksmith [18]

Defenses Overview

Methodology \ Defense	MASCAT	Chiapetta et al. [7]	CloudRadar	[15]	HexPADS	perf	ANVIL	nohammer	No OOM	G-CATT	B-CATT	TRR	MAC	PARA/CRA/PRA	ARMOR	ECC/Chipkill	Refresh Rate
Detection																	
Static Analysis	●	○	○	◐	○	○	○	○	○	○	○	○	○	○	○	○	○
Performance Counters	○	●	●	●	●	●	●	○	○	○	○	○	○	○	○	○	○
Memory Access Pattern	○	○	○	○	○	○	●	●	○	○	○	○	○	○	○	○	○
Neutralization																	
Physical Proximity	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○
Memory Footprint	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○
Elimination																	
Bootloader	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○
Hardware Modification	○	○	○	○	○	○	○	○	○	○	○	●	●	●	●	●	○
BIOS Update	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●

What if you don’t need to hammer two or more rows?

What if you don't need to hammer two or more rows?

One-location hammering

How to hammer?

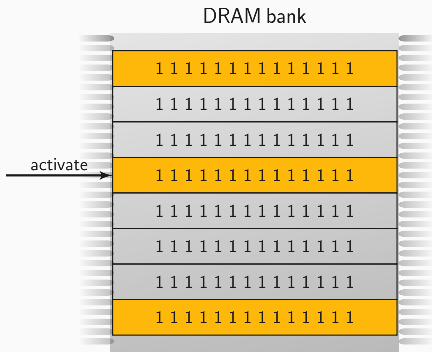
- There are two different hammering techniques

How to hammer?

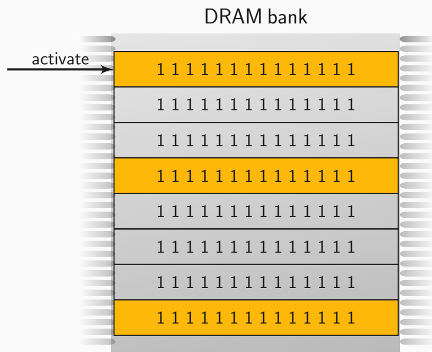
- There are two different hammering techniques
- #1: Hammer one row next to victim row and other random rows

- There are two different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row

#1 - Single-sided hammering

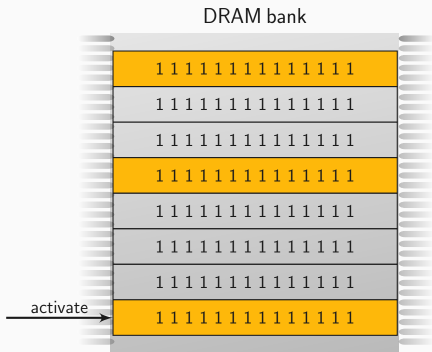


#1 - Single-sided hammering

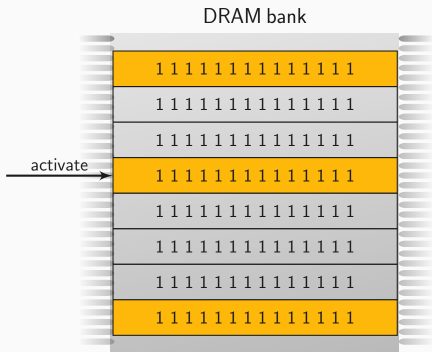


#1 - Single-sided hammering

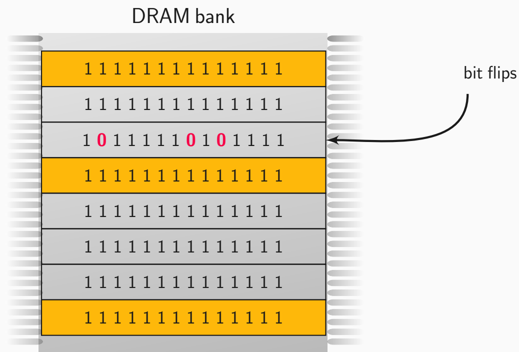
#1 - Single-sided hammering



#1 - Single-sided hammering

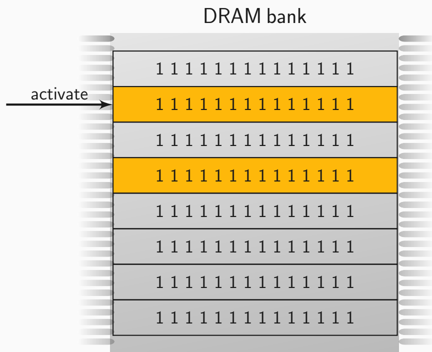


#1 - Single-sided hammering



#2 - Double-sided hammering

#2 - Double-sided hammering

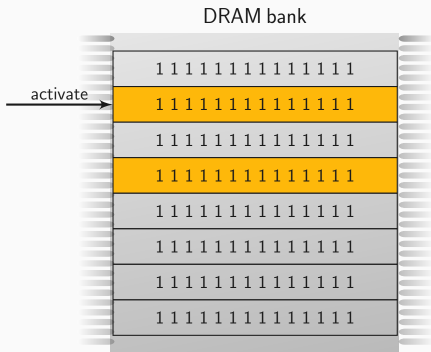


#2 - Double-sided hammering

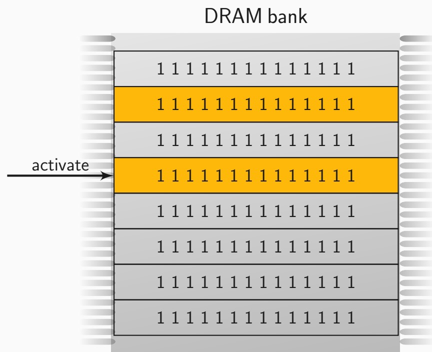
DRAM bank

activate

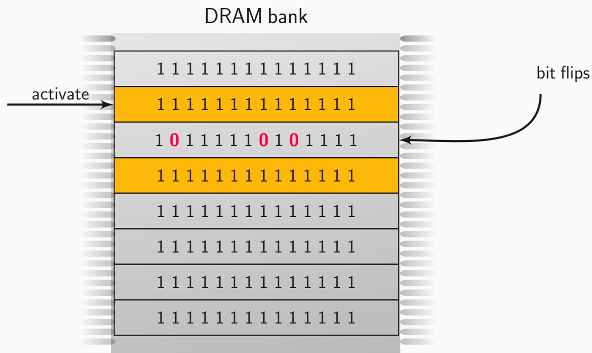
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1


DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1

#2 - Double-sided hammering



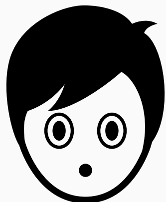
#2 - Double-sided hammering



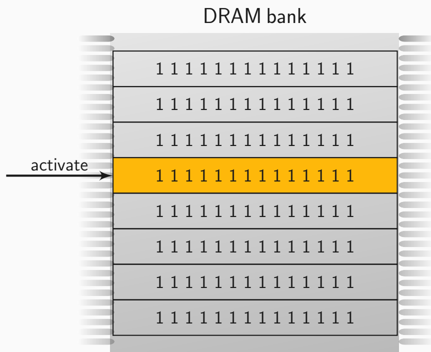
HAMMERING
TWO ROWS

HAMMERING TWO ROWS

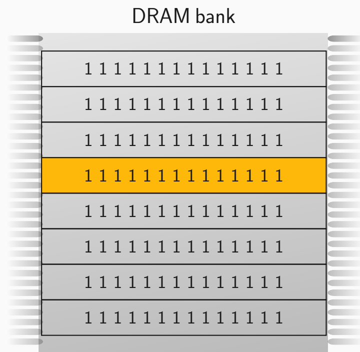
HAMMERING A SINGLE ROW



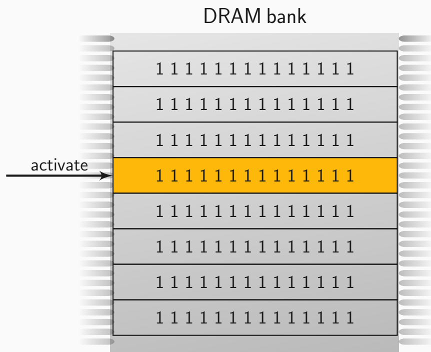
- There are three different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row
- #3: Hammer only one row next to victim row


DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1

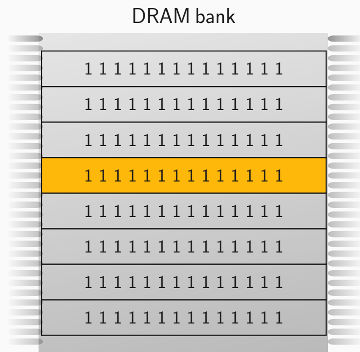
#3 - One-location hammering / Rowpress



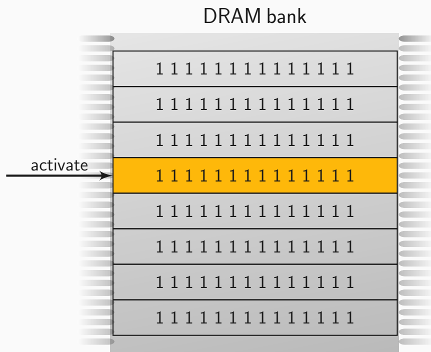
#3 - One-location hammering / Rowpress

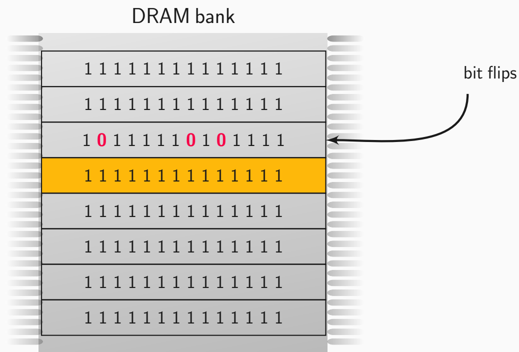


#3 - One-location hammering / Rowpress



#3 - One-location hammering / Rowpress




DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 0 1 0 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
bit flips

dgruss@lab05: ./rowhammer 13

File Edit View Search Terminal Help

```
dgruss@lab05 ~/flipfloyd (git)-[master] % make
g++ -std=c++11 -O3 -o rowhammer rowhammer.cc
dgruss@lab05 ~/flipfloyd (git)-[master] % ./rowhammer 13
Allocating memory... 90%
```

Two underlying reasons:

Two underlying reasons:

Two underlying reasons:

- Memory-controller page policies

Two underlying reasons:

- Memory-controller page policies

Two underlying reasons:

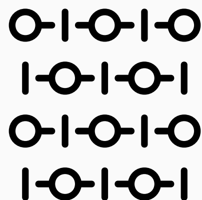
- Memory-controller page policies
- Physical effect called RAS-clobber

Two underlying reasons:

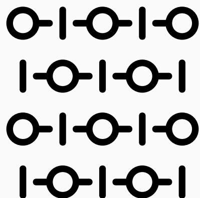
- Memory-controller page policies

- Physical effect called RAS-clobber
- Recently shown in Rowpress paper [0]

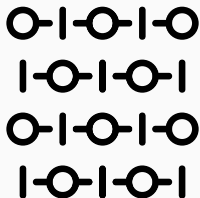
Memory-Controller Policies



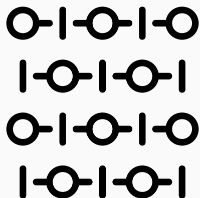
- **Open-page policy**: Keep row opened and buffered



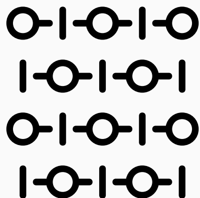
- **Open-page policy**: Keep row opened and buffered
  - Low latency for subsequent accesses to same row
  - High latency for accesses to any other row



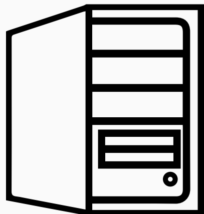
- **Open-page policy**: Keep row opened and buffered
  - Low latency for subsequent accesses to same row
  - High latency for accesses to any other row
- **Closed-page policy**: Immediately close row, ready to open a new row



- **Open-page policy**: Keep row opened and buffered
  - Low latency for subsequent accesses to same row
  - High latency for accesses to any other row
- **Closed-page policy**: Immediately close row, ready to open a new row
  - Medium latency for accesses to any row

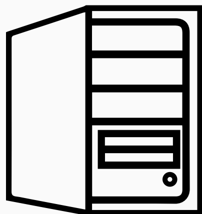


- **Open-page policy**: Keep row opened and buffered
  - Low latency for subsequent accesses to same row
  - High latency for accesses to any other row
- **Closed-page policy**: Immediately close row, ready to open a new row
  - Medium latency for accesses to any row
  - Perform better on multi-core systems [9]

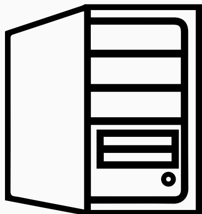


- Policies that preemptively close rows, would allow one-location hammering

Memory-Controller Policies

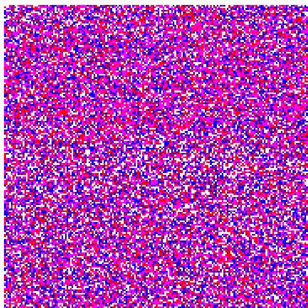


- Policies that preemptively close rows, would allow one-location hammering
- We observed close-page policies on desktop computers



- Policies that preemptively close rows, would allow one-location hammering
- We observed close-page policies on desktop computers
- Mobile devices (e.g., laptops) seem to use mostly open-page policies

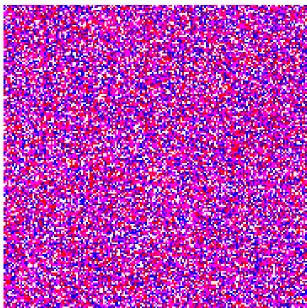
How well does it work?



Double-sided

77.0 % bit offsets

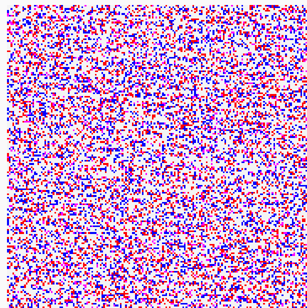
51.7 % 0→1 bit flips



Single-sided

78.5 % bit offsets

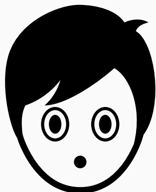
54.1 % 0→1 bit flips



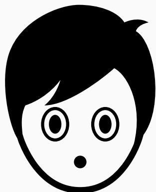
One-location

36.5 % bit offsets

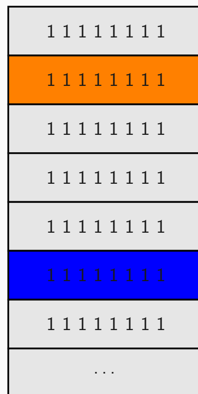
51.6 % 0→1 bit flips

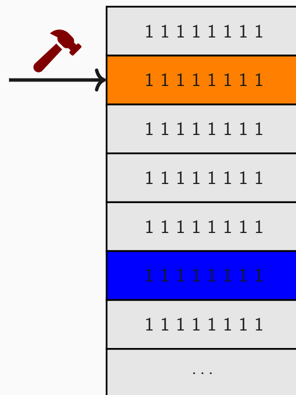


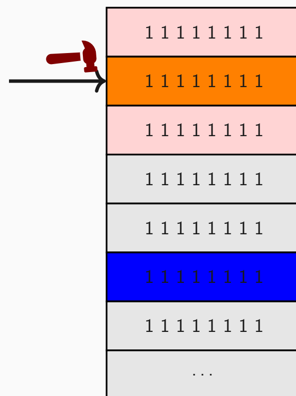
- There are four different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row
- #3: Hammer only one row next to victim row

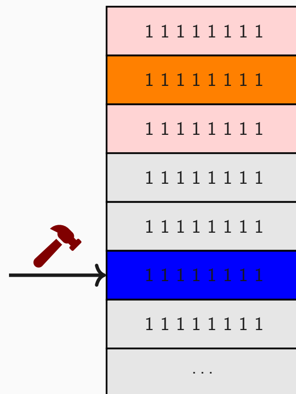


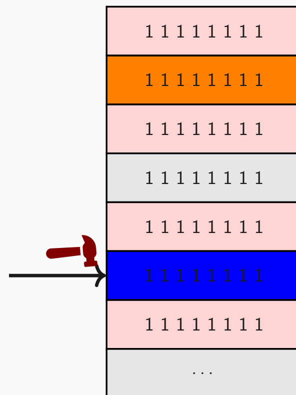
- There are four different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row
- #3: Hammer only one row next to victim row
- #4: Hammer from a larger distance with the **help** of TRR [21]

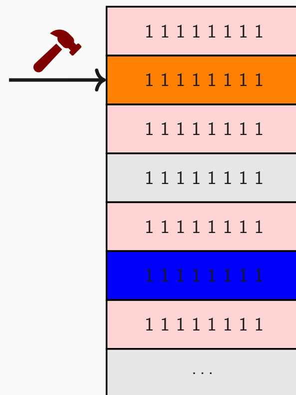

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

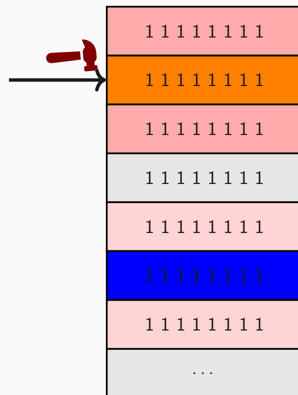

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

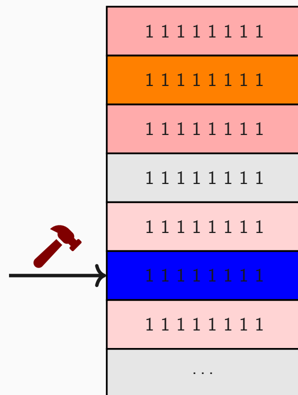

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

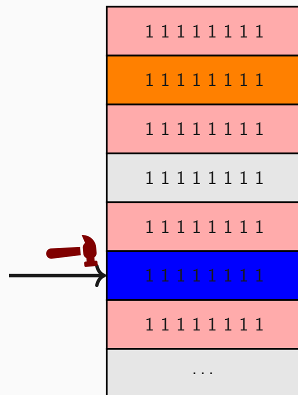

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
. . .

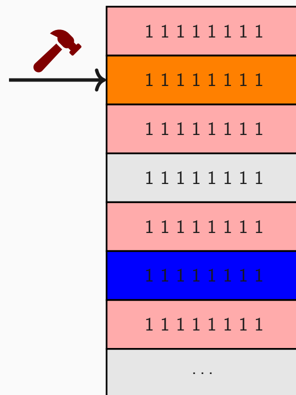

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

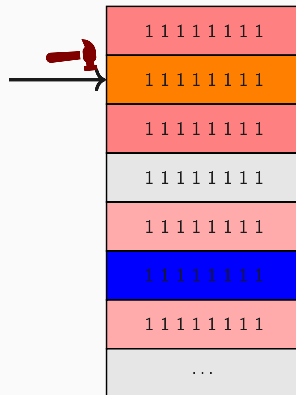

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

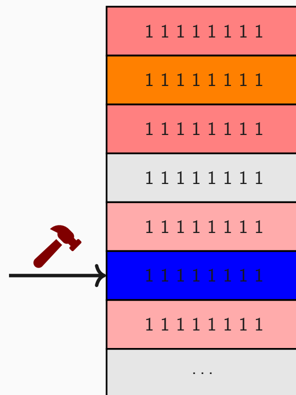

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
. . .


1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

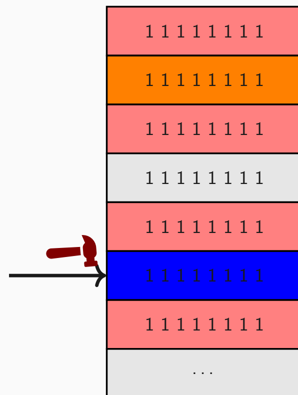

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
. . .

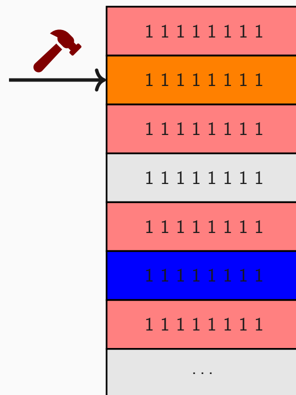

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

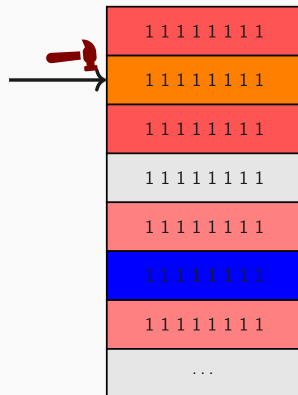

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

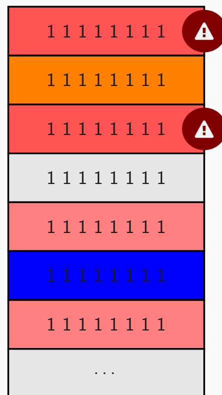

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

Half-Double Rowhammer

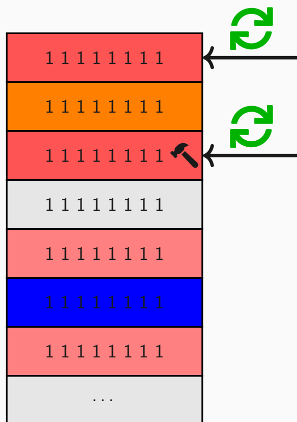


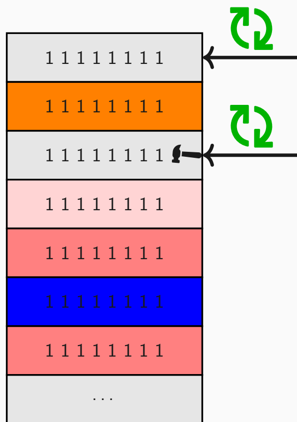

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
. . .

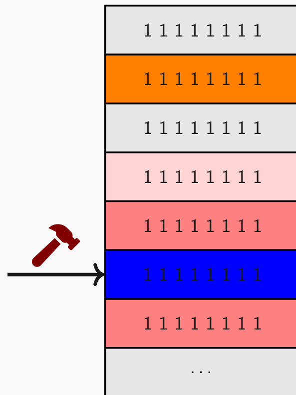

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...


1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

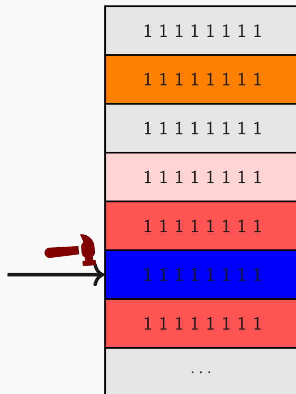
Half-Double Rowhammer

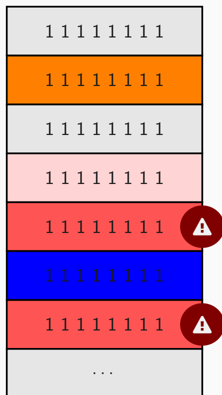



1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

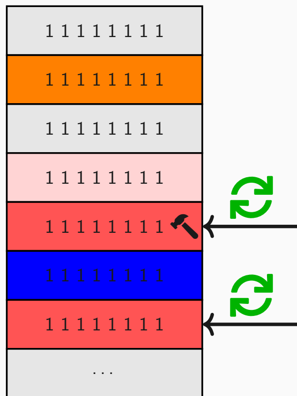

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
. . .

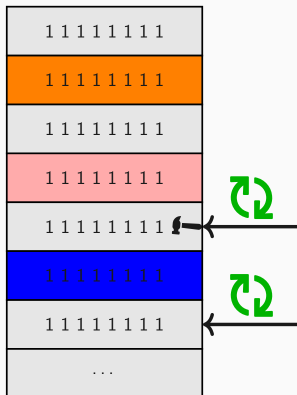
Half-Double Rowhammer

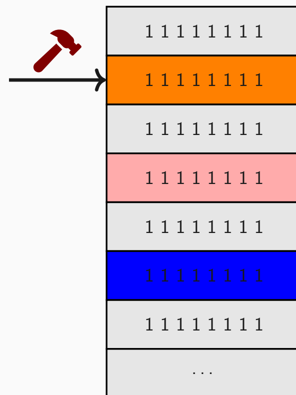




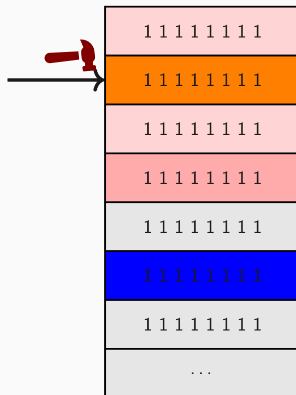
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

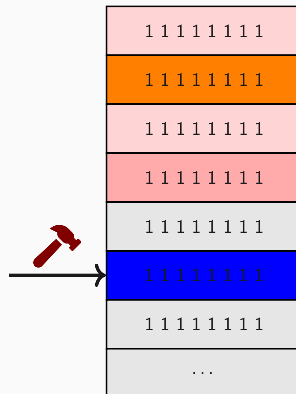

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
. . .


1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
. . .

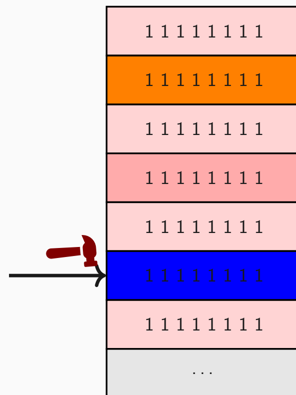

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
. . .

Half-Double Rowhammer

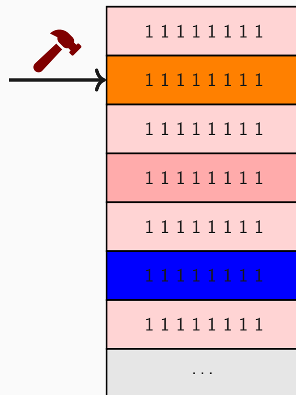


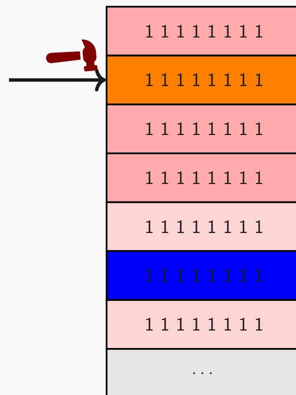

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

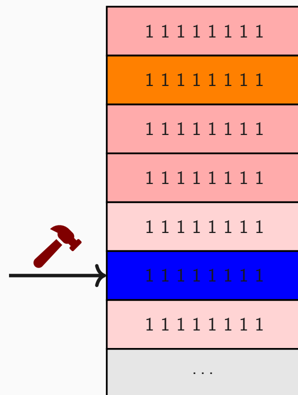
Half-Double Rowhammer

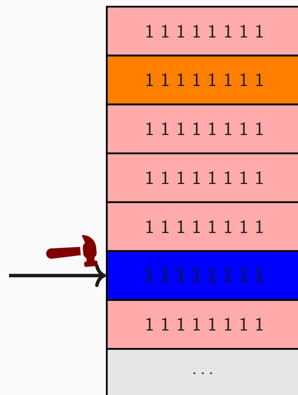


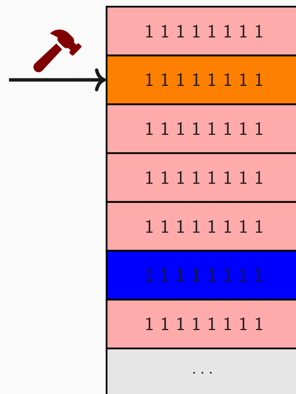
Half-Double Rowhammer

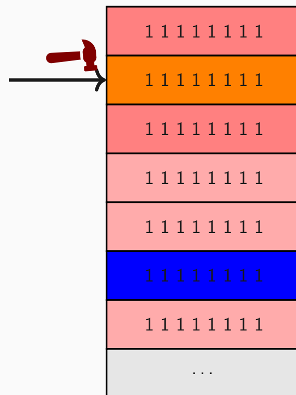


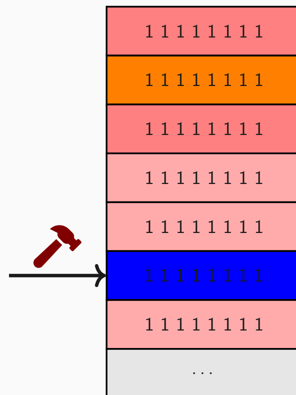

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

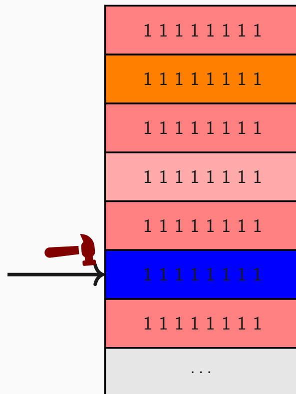

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...


1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

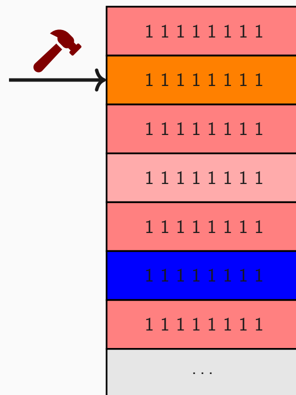

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

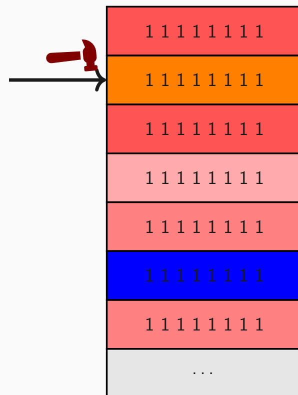

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
. . .

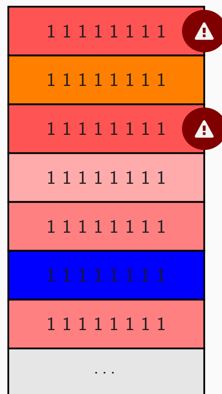

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

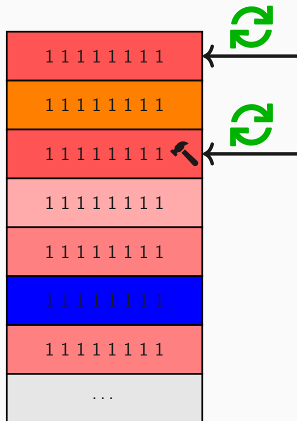

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

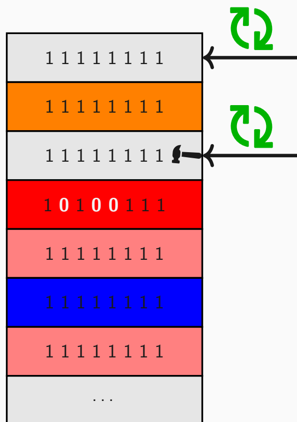
Half-Double Rowhammer

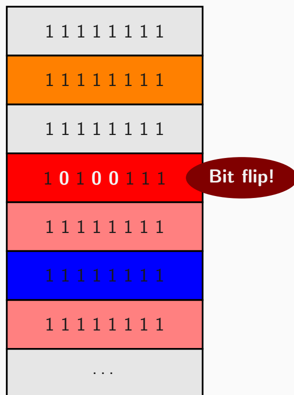



1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...


1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...


1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
. . .


1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 0 1 0 0 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...


1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 0 1 0 0 1 1 1
Bit flip!
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
...

- TRR is a **confused deputy**

- TRR is a **confused deputy**
- it helps an attacker to hammer with greater distance

- TRR is a **confused deputy**
- it helps an attacker to hammer with greater distance
- Rowhammer still changes and is not fully understood

- TRR is a **confused deputy**
- it helps an attacker to hammer with greater distance
- Rowhammer still changes and is not fully understood
- Developing countermeasures for a such a problem is **hard**

What if we cannot target kernel pages?

What if we cannot target kernel pages?

Opcode Flipping

- Many applications perform actions as root

Opcode Flipping

- Many applications perform actions as root
- They can be used by unprivileged users as well

- Many applications perform actions as root
- They can be used by unprivileged users as well
- Implicitly: e.g., `ping` or `mount`

Opcode Flipping

- Many applications perform actions as root
- They can be used by unprivileged users as well
- Implicitly: e.g., `ping` or `mount`
- Explicitly: `sudo`

- Many applications perform actions as root
- They can be used by unprivileged users as well
- Implicitly: e.g., `ping` or `mount`
- Explicitly: `sudo`
- Target `sudo` (easy to exploit)

JE
HLT
0 1 1 1 0 1 0 0
1 1 1 1 0 1 0 0

Opcode Flipping - Conditional Jump

JE
0 1 1 1 0 1 0 0
PUSHQ
0 1 0 1 0 1 0 0

Opcode Flipping - Conditional Jump

Opcode Flipping - Conditional Jump

JE
0 1 1 1 0 1 0 0
JO
0 1 1 1 0 0 0 0

Opcode Flipping - Conditional Jump

Opcode Flipping - Conditional Jump

Opcode Flipping

- Conditional jumps are not the only targets

Opcode Flipping

- Conditional jumps are not the only targets
- Other targets include

Opcode Flipping

- Conditional jumps are not the only targets
- Other targets include
  - Comparisons
  - Addresses of memory loads/stores
  - Address calculations
  - ...

Opcode Flipping

- Conditional jumps are not the only targets
- Other targets include
  - Comparisons
  - Addresses of memory loads/stores
  - Address calculations
  - ...
- Manual analysis of `sudo` revealed 29 possible bit flips

Opcode Flipping

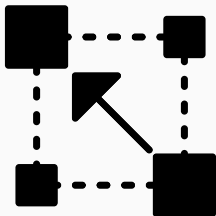
- Conditional jumps are not the only targets
- Other targets include
  - Comparisons
  - Addresses of memory loads/stores
  - Address calculations
  - ...
- Manual analysis of `sudo` revealed 29 possible bit flips
- They all somehow skipped the password check

How to get the target virtual page to the target physical location?

How to get the target virtual page to the target physical location?

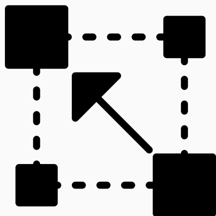
Memory Waylaying

Placing the binary



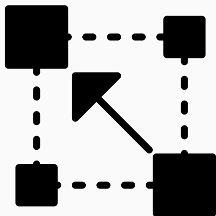
- Not as easy as with page tables

Placing the binary



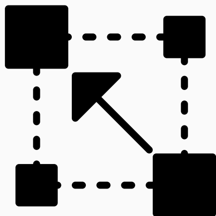
- Not as easy as with page tables
- Binary only once in memory + stays in memory (in the page cache) even after termination

Placing the binary

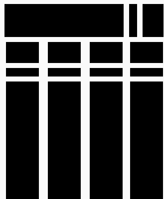


- Not as easy as with page tables
- Binary only once in memory + stays in memory (in the page cache) even after termination
- Only evicted if page cache is full

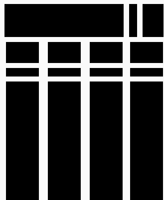
Placing the binary



- Not as easy as with page tables
- Binary only once in memory + stays in memory (in the page cache) even after termination
- Only evicted if page cache is full
- Page cache usually occupies all unused memory

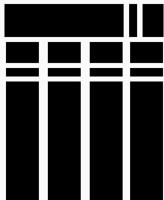


- If a binary is loaded the first time, it is loaded to the memory

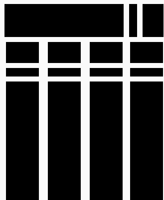


- If a binary is loaded the first time, it is loaded to the memory
- It stays in memory (in the page cache) even after execution

Page Cache



- If a binary is loaded the first time, it is loaded to the memory
- It stays in memory (in the page cache) even after execution
- Only evicted if page cache is full

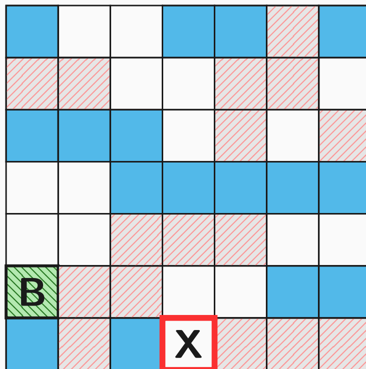


- If a binary is loaded the first time, it is loaded to the memory
- It stays in memory (in the page cache) even after execution
- Only evicted if page cache is full
- Page cache is huge - usually all unused memory

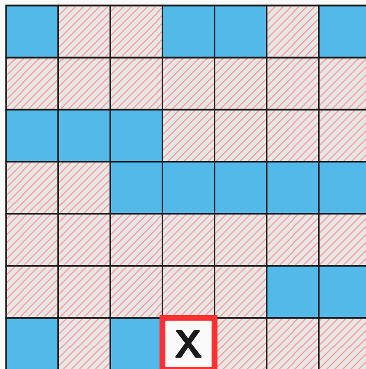
MEMORY WAYLAYING

Wait for the right moment, and then hit it with a bit flip!

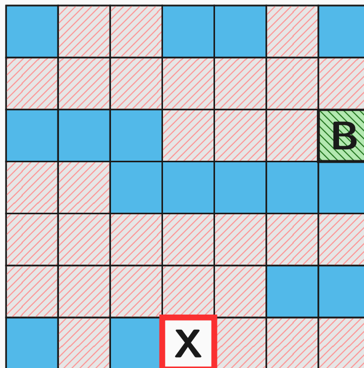
(1) Start



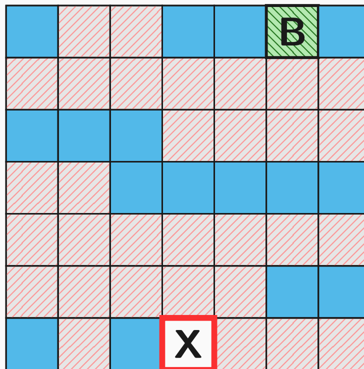
(2) Evict Page Cache



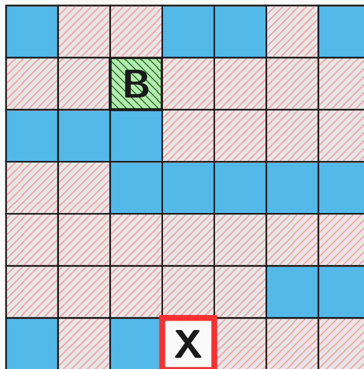
(3) Access Binary



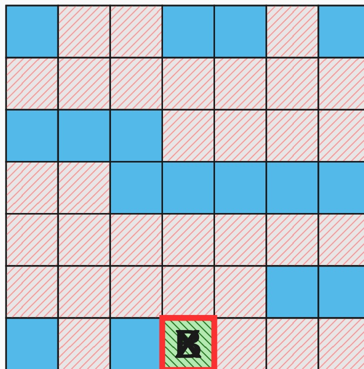
(4) Evict + Access



(5) Evict + Access

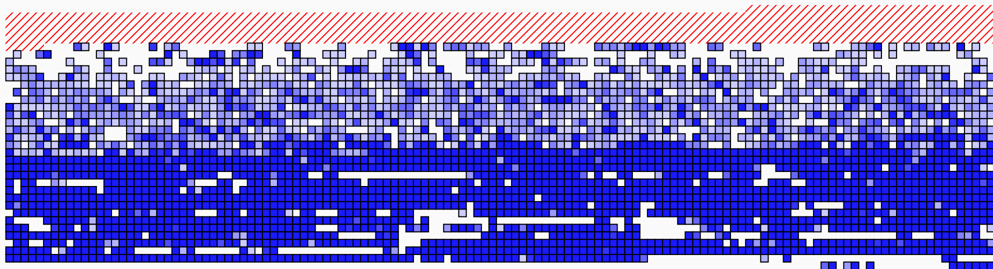


(6) Stop if target reached

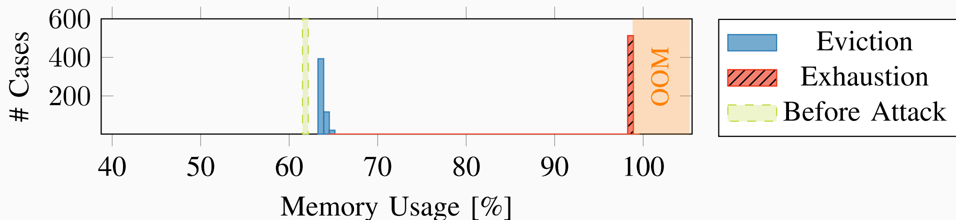


How well does it work?

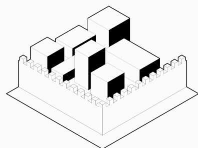
- New pages cover most of the physical memory



- Great advantage over memory massaging: only negligible memory footprint

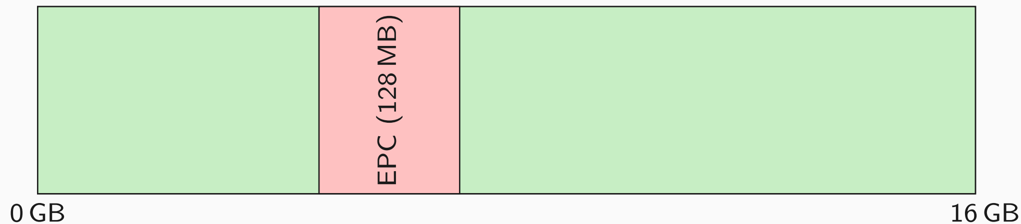


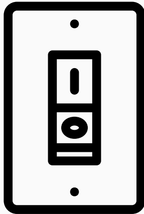
Rowhammer + SGX = Cheap Denial of Service



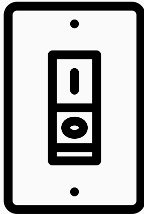
- Instruction-set extension
- Integrity and confidentiality of code and data in untrusted environments
- Run with user privileges and restricted, e.g., no system calls
- Run programs in enclaves using protected areas of memory

0 GB
16 GB

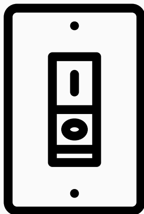

EPC (128 MB)
0 GB
16 GB



- What happens if a bit flips in the EPC?

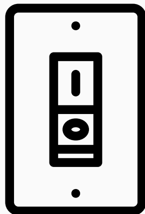


- What happens if a bit flips in the EPC?
- Integrity check will fail!



- What happens if a bit flips in the EPC?
- Integrity check will fail!

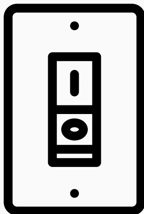
→ Locks up the memory controller



- What happens if a bit flips in the EPC?
- Integrity check will fail!

→ Locks up the memory controller

→ Not a single further memory access!

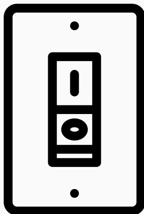


- What happens if a bit flips in the EPC?
- Integrity check will fail!

→ Locks up the memory controller

→ Not a single further memory access!

→ System halts immediately



- What happens if a bit flips in the EPC?
- Integrity check will fail!

→ Locks up the memory controller

→ Not a single further memory access!

→ System halts immediately

SOUNDS UNSAFE?
IT IS UNSAFE!
imgflip.com

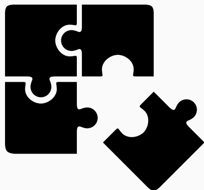
- If a malicious enclave induces a bit flip, ...

- If a malicious enclave induces a bit flip, . . .
- . . . the entire machine halts

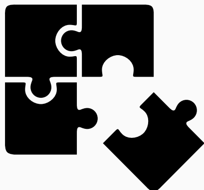
- If a malicious enclave induces a bit flip, ...
- ... the entire machine halts
- ... including co-located tenants

- If a malicious enclave induces a bit flip, ...
- ... the entire machine halts
- ... including co-located tenants
- Denial-of-Service Attacks in the Cloud [12, 17]

SGX + One-location Hammering + Opcode Flipping = Undetectable Exploit

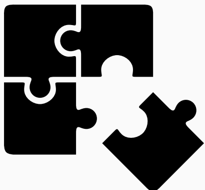


- SGX protects software from malicious environments



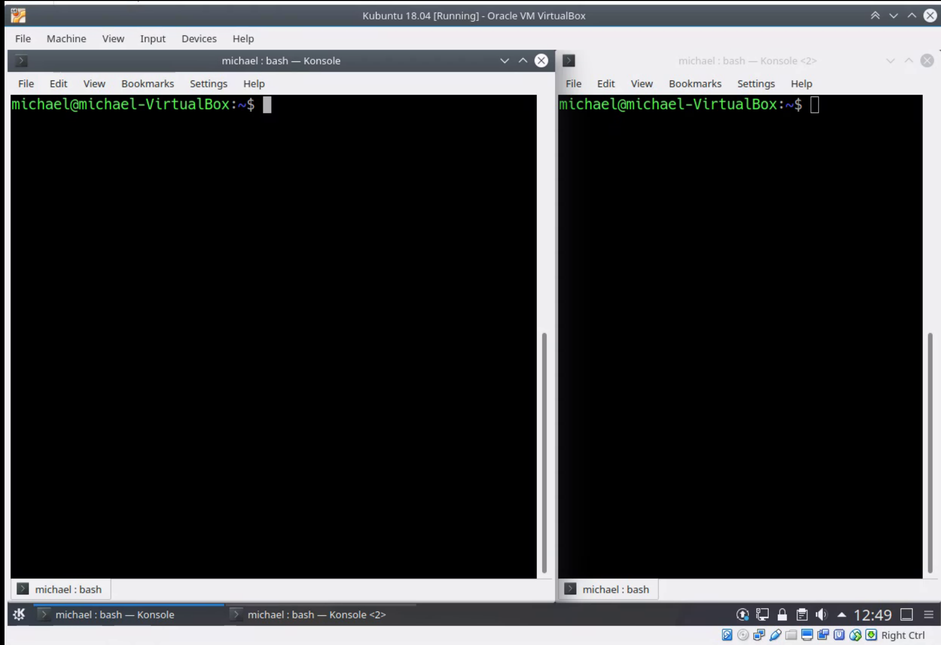
- SGX protects software from malicious environments
- Thwarts static and dynamic (= performance counters) analysis

(Ab)using SGX Protection

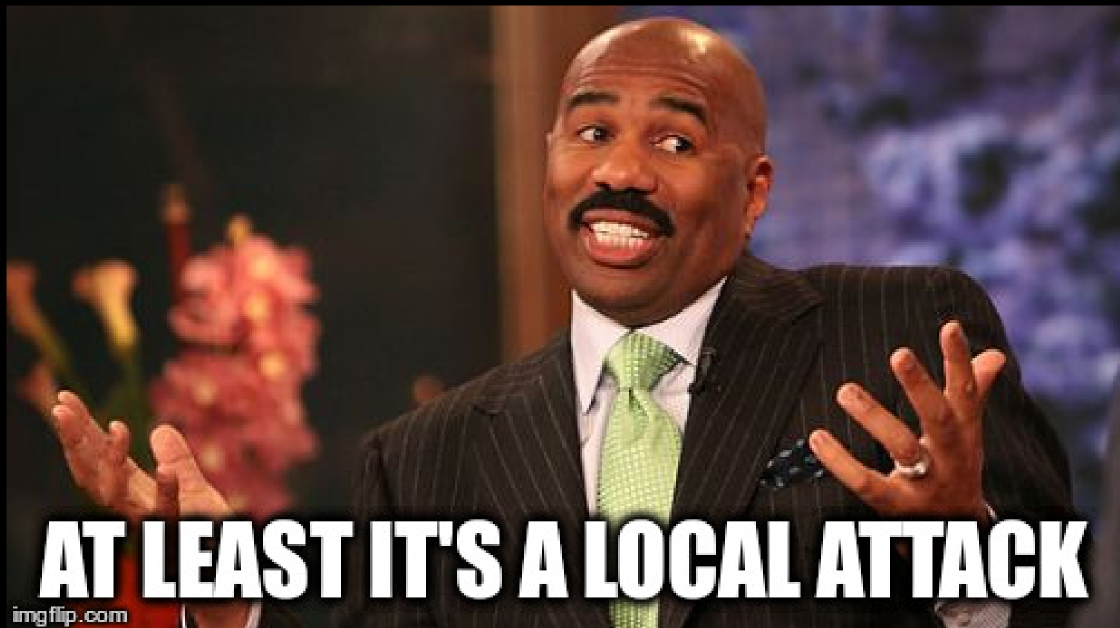


- SGX protects software from malicious environments
- Thwarts static and dynamic (= performance counters) analysis
- Hammering from SGX defeats countermeasures relying on this

STEALTH LEVEL: EXPERT


Kubuntu 18.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
michael : bash — Konsole
File Edit View Bookmarks Settings Help
michael@michael-VirtualBox:~$
michael : bash
michael : bash — Konsole <2>
File Edit View Bookmarks Settings Help
michael@michael-VirtualBox:~$
michael : bash
michael : bash — Konsole
michael : bash — Konsole <2>
12:49
Right Ctrl

Bypass \ Defense Class	Static Analysis	Performance Counters	Memory Access Pattern	Physical Proximity	Memory footprint
Intel SGX	●	●	○	○	○
One-location hammering	○	○	●	○	○
Opcode flipping	○	○	○	●	○
Memory waylaying	○	○	○	○	●
Defense class defeated	●	●	●	●	●


AT LEAST IT'S A LOCAL ATTACK
imgflip.com

Just comparing some performance numbers...

Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [13]

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [13]

$=$ 671 875 accesses per second

Just comparing some performance numbers...

- $\geq 43\,000$ hammering attempts (within 64 ms) for a bit flip [13]

$=$ 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [13]

= 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

→ 111 979 packets per second

Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [13]

= 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

→ 111 979 packets per second

- Network packets are a least 64 B

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [13]

$=$ 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

$\rightarrow$ 111 979 packets per second

- Network packets are a least 64 B

$=$ 7 166 656 B/s $=$ 7 MB/s $=$ 57 Mb/s

Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [13]

= 671 875 accesses per second

- Network packets access memory location up to 6 times (depending on kernel)

→ 111 979 packets per second

- Network packets are a least 64 B

= 7 166 656 B/s = 7 MB/s = 57 Mb/s

→ That sounds doable on "modern" networks

Just comparing some performance numbers...

- $\geq$ 43 000 hammering attempts (within 64 ms) for a bit flip [13]

= 671 875 accesses per second

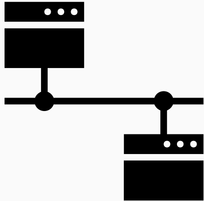
- Network packets access memory location up to 6 times (depending on kernel)

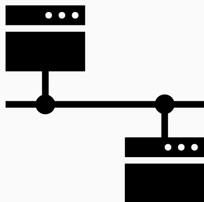
→ 111 979 packets per second

- Network packets are a least 64 B

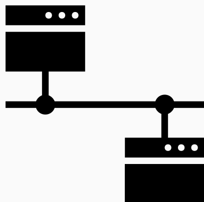
= 7 166 656 B/s = 7 MB/s = 57 Mb/s

→ That sounds doable on "modern" networks



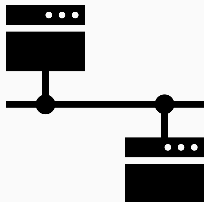


Inducing bit flips:



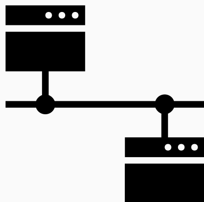
Inducing bit flips:

- Network stacks on ARM often use uncached memory (perfect for hammering)



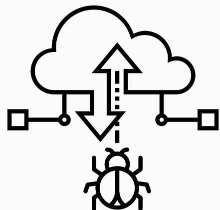
Inducing bit flips:

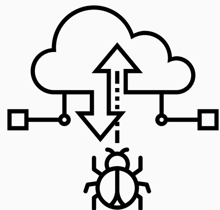
- Network stacks on ARM often use uncached memory (perfect for hammering)
- Intel recommends Intel CAT for QoS (perfect for hammering)



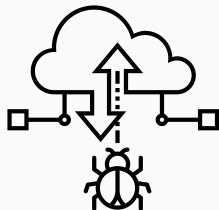
Inducing bit flips:

- Network stacks on ARM often use uncached memory (perfect for hammering)
- Intel recommends Intel CAT for QoS (perfect for hammering)
- Network reachable code might use `clflush` or non-temporal stores
  (both great for hammering)



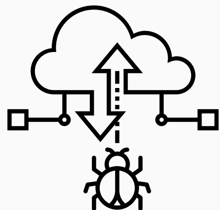


Nethammer on ...



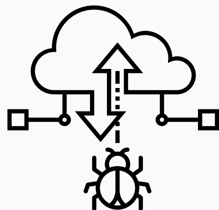
Nethammer on ...

- SGX = powerful DoS



Nethammer on ...

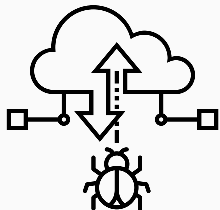
- SGX = powerful DoS
- File system = persistent DoS



Nethammer on ...

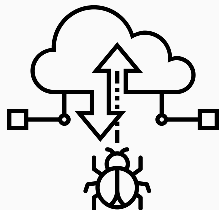
- SGX = powerful DoS
- File system = persistent DoS
- DNS entries on a DNS server = bit-squatting attack

Exploiting Nethammer Bit Flips



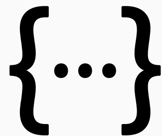
Nethammer on ...

- SGX = powerful DoS
- File system = persistent DoS
- DNS entries on a DNS server = bit-squatting attack
- OCSP servers = make invalid certificates great again



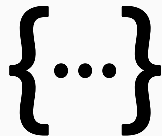
Nethammer on ...

- SGX = powerful DoS
- File system = persistent DoS
- DNS entries on a DNS server = bit-squatting attack
- OCSP servers = make invalid certificates great again
- Crypto = generate private keys for broken public keys



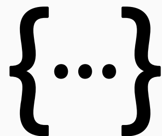
- Many (academic) countermeasures were proposed to mitigate Rowhammer

What do we learn from it?

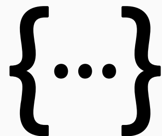


- Many (academic) countermeasures were proposed to mitigate Rowhammer
- We showed that all of them can be circumvented [12]

What do we learn from it?

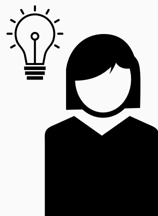


- Many (academic) countermeasures were proposed to mitigate Rowhammer
- We showed that all of them can be circumvented [12]
- We cannot design countermeasures focused on specifics of the attack



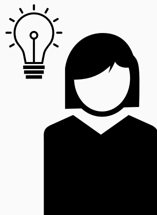
- Many (academic) countermeasures were proposed to mitigate Rowhammer
- We showed that all of them can be circumvented [12]
- We cannot design countermeasures focused on specifics of the attack
- Otherwise we only patch concrete exploits, but do not solve the problem

What do we learn from it?



Apple had a great idea:

- Lower refresh rate $\rightarrow$ save energy $+$ more flips

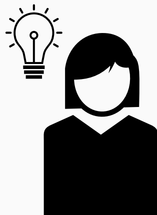


Apple had a great idea:

- Lower refresh rate $\rightarrow$ save energy + more flips

$\rightarrow$ ECC memory $\rightarrow$ fewer flips

What do we learn from it?



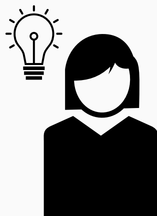
Apple had a great idea:

- Lower refresh rate → save energy + more flips

→ ECC memory → fewer flips

- It's an optimization problem.

What do we learn from it?

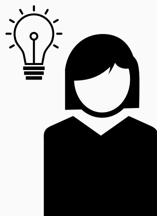


Apple had a great idea:

- Lower refresh rate → save energy + more flips

→ ECC memory → fewer flips

- It's an optimization problem.
  - Too aggressive? → bit flips



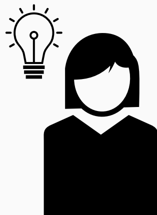
Apple had a great idea:

- Lower refresh rate → save energy + more flips

→ ECC memory → fewer flips

- It's an optimization problem.
  - Too aggressive? → bit flips
  - Too cautious? → waste of energy

What do we learn from it?

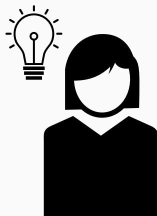


Apple had a great idea:

- Lower refresh rate $\rightarrow$ save energy + more flips

$\rightarrow$ ECC memory $\rightarrow$ fewer flips

- It's an optimization problem.
  - Too aggressive? $\rightarrow$ bit flips
  - Too cautious? $\rightarrow$ waste of energy
  - What if the "too aggressive" changes over time?



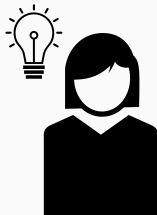
Apple had a great idea:

- Lower refresh rate → save energy + more flips

→ ECC memory → fewer flips

- It's an optimization problem.
  - Too aggressive? → bit flips
  - Too cautious? → waste of energy
  - What if the "too aggressive" changes over time?
  - What if attackers come up with slightly better attacks?

What do we learn from it?



Apple had a great idea:

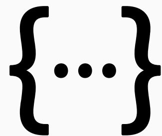
- Lower refresh rate → save energy + more flips

→ ECC memory → fewer flips

- It's an optimization problem.
  - Too aggressive? → bit flips
  - Too cautious? → waste of energy
  - What if the "too aggressive" changes over time?
  - What if attackers come up with slightly better attacks?

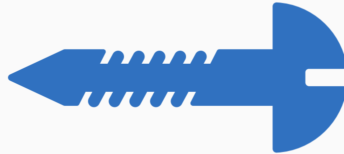
  → Difficult to optimize with an adversary working against you

A new Era of Rowhammer Countermeasures?

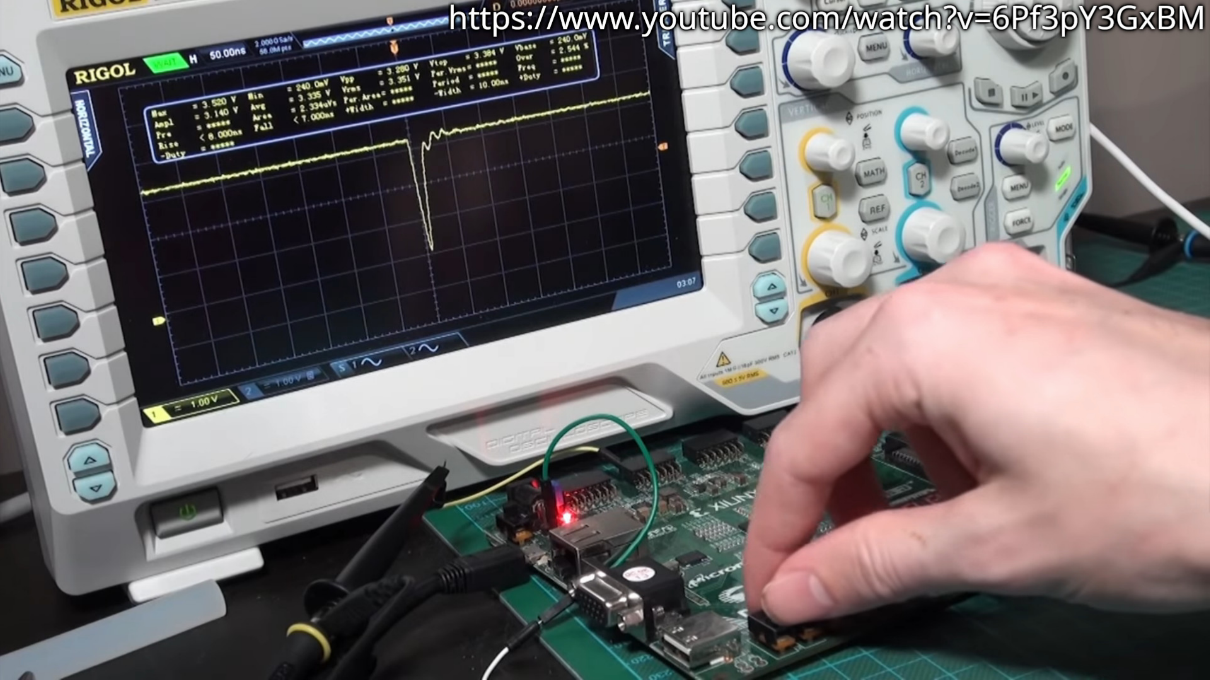


- Data integrity protection for **all data** in the DRAM [19]
- Compute Cryptographic MAC on all data
- No focus on Rowhammer
- No new hammer method or exploit can circumvent it

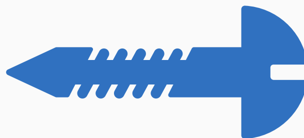
Anything other than Rowhammer?



A NEW CLASS OF FAULT ATTACKS


https://www.youtube.com/watch?v=6Pf3pY3GxBM
RIGOL
50.00ns
MENU
MATH
REF
FORCE
03:07
1.00 V

Our starting point...



Tang2017

DVFS

Changing the voltage and frequency of the CPU

Changing the voltage and frequency of the CPU

- Gamers want fast responses

Changing the voltage and frequency of the CPU

- Gamers want fast responses
- Cloud servers want high-assurance and low running costs

Dynamic Voltage and Frequency Scaling

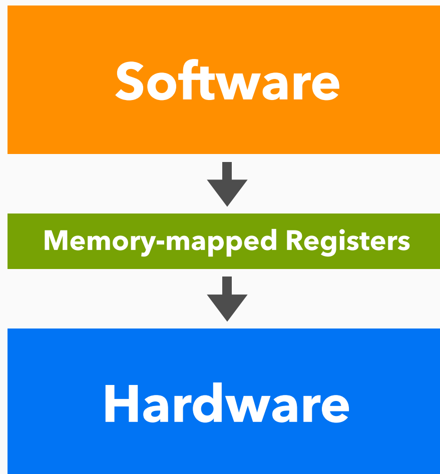
Changing the voltage and frequency of the CPU

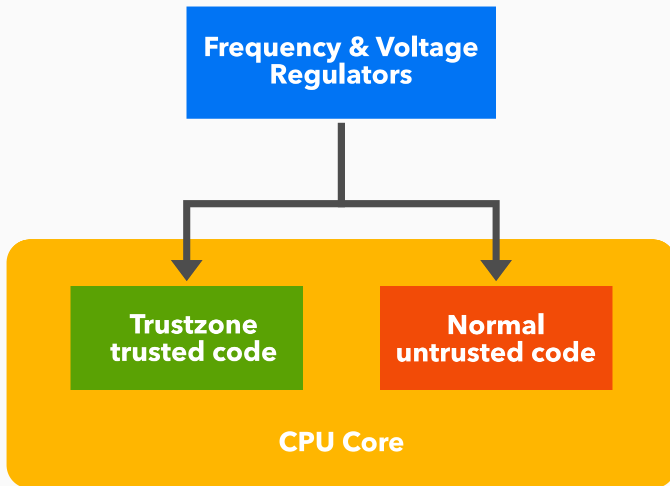
- Gamers want fast responses
- Cloud servers want high-assurance and low running costs
- What if the hardware gets hot?

Dynamic Voltage and Frequency Scaling

Changing the voltage and frequency of the CPU

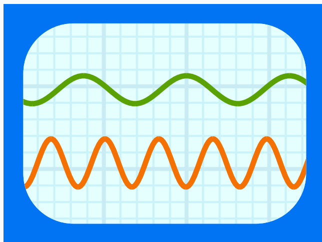
- Gamers want fast responses
- Cloud servers want high-assurance and low running costs
- What if the hardware gets hot?
- Optimal voltage & frequency is difficult!


Software
Memory-mapped Registers
Hardware

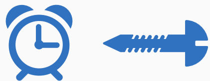


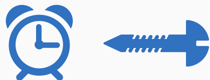
Frequency & Voltage Regulators
Trustzone trusted code
Normal untrusted code
CPU Core

```
add.w    (a0)+,d1
cmp.l    a0,d0
bcc.s    loop
movea.l  #$18E,a1
cmp.w    (a1),d1
bne.w    WrongChecksum
```

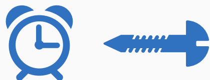


2 + 2 = 5

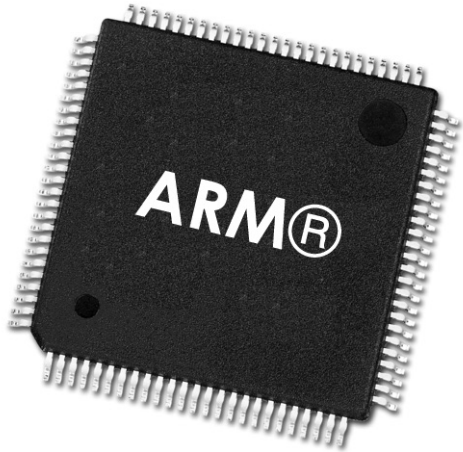




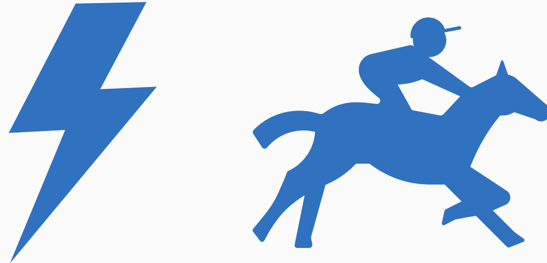
- Infer secret AES key that was stored within Trustzone



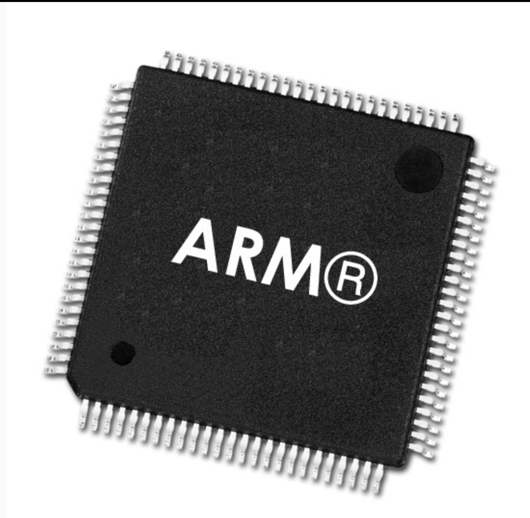
- Infer secret AES key that was stored within Trustzone
- Trick Trustzone into loading a self-signed app


ARM®

More are joining the revolution...

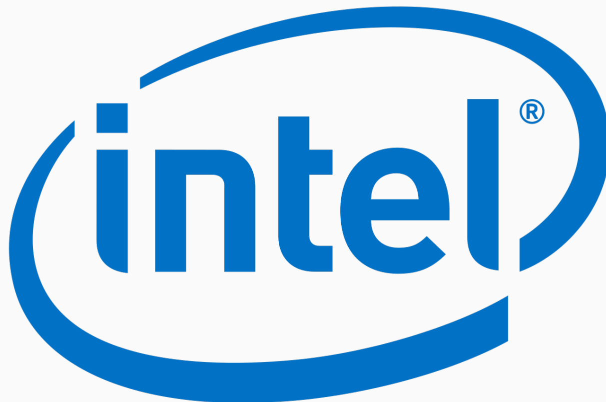


Qiu2019


ARM®

ARE WE ALL THINKING THE SAME THING?

What about Intel?



Intel® Extreme Tuning Utility

Monitoring Settings Help

System Information
Manual Tuning
• All Controls
Core
Graphics
Stress Test
Profiles

Core

Reference Clock 103,2258 MHz
Max Non Turbo Boost Ratio 34 x
Turbo Boost Short Power Max Enable: Disable / Enable
Turbo Boost Short Power Max 1200,000 W
Turbo Boost Power Max 1050,000 W
Turbo Boost Power Time Window 0,00097656 Seconds
Core Current Limit 300,000 A
Additional Turbo Voltage 0,00000 mV

Multipliers

1 Active Core	42 x
2 Active Cores	42 x
3 Active Cores	42 x
4 Active Cores	42 x

4 Active Cores
Default 38 x
Active 38 x
Proposed 42 x
Limits the maximum ratio that the processor can use while four cores are active.

Graphics

Processor Graphics Current Limit 300,000 A

Core	Default	Proposed
Reference Clock	101,0526 MHz	103,2258 MHz
Max Non Turbo Boost Ratio	34 x	34 x
Max Non-Turbo Boost CPU Sp...	3,436 GHz	3,510 GHz
Max Turbo Boost CPU Speed	4,042 GHz	4,335 GHz
1 Active Core	40 x	42 x
2 Active Cores	40 x	42 x
3 Active Cores	39 x	42 x
4 Active Cores	38 x	42 x
Turbo Boost Power Max	1000,000 W	1050,000 W
Turbo Boost Short Power Max	1200,000 W	1200,000 W
Turbo Boost Short Power Max...	Enable	Enable
Turbo Boost Power Time Wind...	0,00097656 S...	0,00097656 S...
Core Current Limit	300,000 A	300,000 A
Additional Turbo Voltage	0,00000 mV	0,00000 mV

Graphics	Default	Proposed
Processor Graphics Current Li...	300,000 A	300,000 A

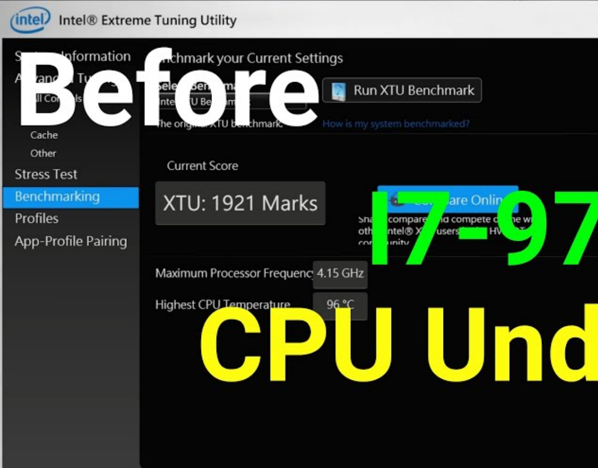
Apply Discard Save to Profile
Force Reboot

CPU Core Temperature 37 °C
CPU Utilization 3 %
Processor Frequency 3,54 GHz
Memory Utilization 2708 MB
CPU Total TDP 15 W

5 Minutes

CPU Utilization 3 %	Memory Utilization 2708 MB	CPU Core Temperature 36 °C	CPU Throttling 0%	Processor Frequency 3,54 GHz
Graphics Frequency 354 MHz	Active Core Count 1	CPU Total TDP 16 W	IACore TDP 10 W	Graphics TDP 0 W
Reference Clock Frequency 101,0 MHz	CPU Core Temperature 1 36 °C	CPU Core Temperature 2 36 °C	CPU Core Temperature 3 36 °C	CPU Core Temperature 4 36 °C
Memory Frequency 1617 MHz				

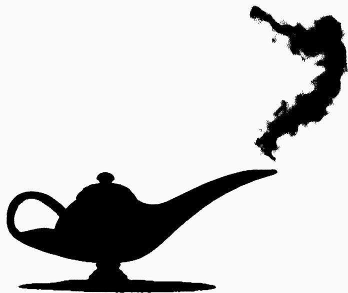
ComputerBase


Intel® Extreme Tuning Utility
Before
Stress Test
Benchmarking
Profiles
App-Profile Pairing
Run XTU Benchmark
How is my system benchmarked?
Current Score
XTU: 1921 Marks
Maximum Processor Frequency 4.15 GHz
Highest CPU Temperature 96 °C
Package Temperature 65 °C
CPU Utilization 5 %
4.14 GHz
5 Minutes

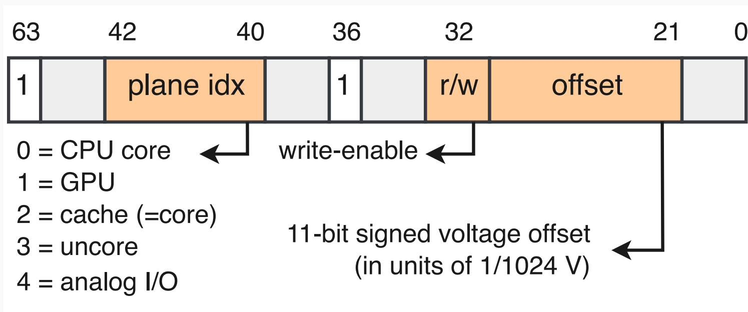
Intel® Extreme Tuning Utility
After
Benchmark your Current Settings
Run XTU Benchmark
How is my system benchmarked?
Current Score
XTU: 2122 Marks
Compare Online
Share, compare, and compete online with other Intel® XTU users in the HWBOT community
Maximum Processor Frequency 4.13 GHz
Highest CPU Temperature 95 °C
Max Core Frequency 4.09 GHz
5 Minutes
I7-9750H
CPU Undervolting
Huge difference!!!

Intel Power Management

Intel Power Management



`msr 0x150`

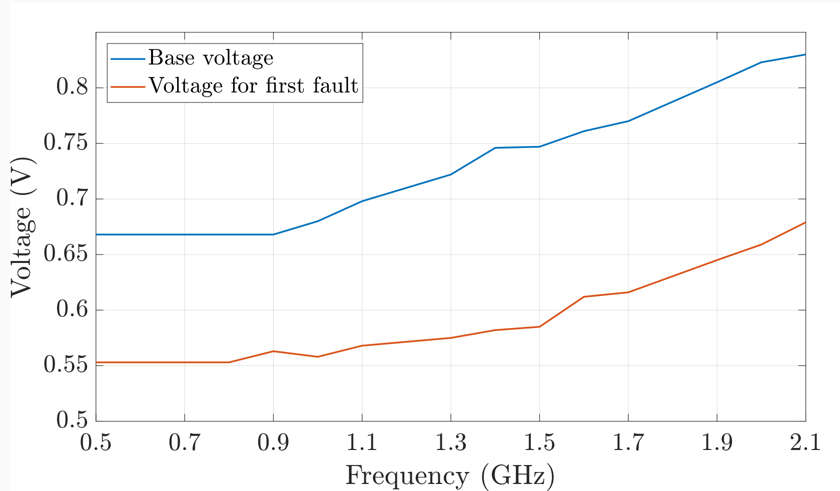

63
42
40
36
32
21
0
1
plane idx
1
r/w
offset
0 = CPU core
1 = GPU
2 = cache (=core)
3 = uncore
4 = analog I/O
write-enable
11-bit signed voltage offset
(in units of 1/1024 V)

Will it fault?

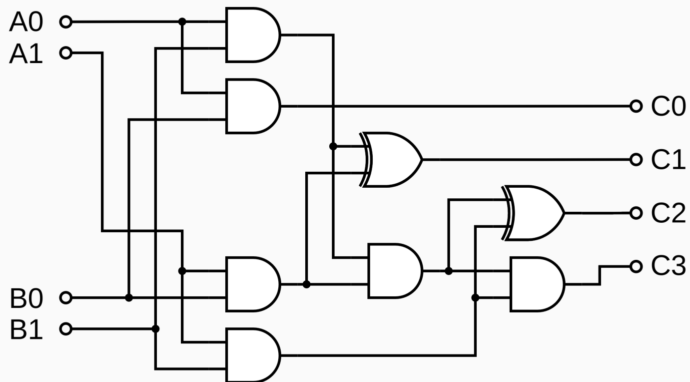
```
uint64_t multiplier = 0x1122334455667788;
uint64_t correct    = 0xdeadbeef * multiplier;
uint64_t var        = 0xdeadbeef * multiplier;

while (var == correct)
{
    var = 0xdeadbeef * multiplier;
}
uint64_t flipped_bits = var ^ correct;
```

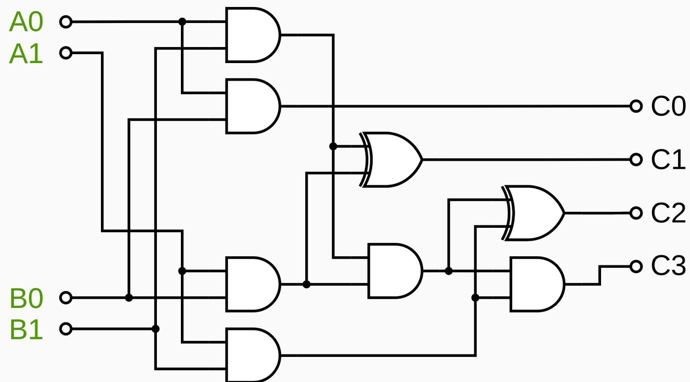
```
bagger>
```

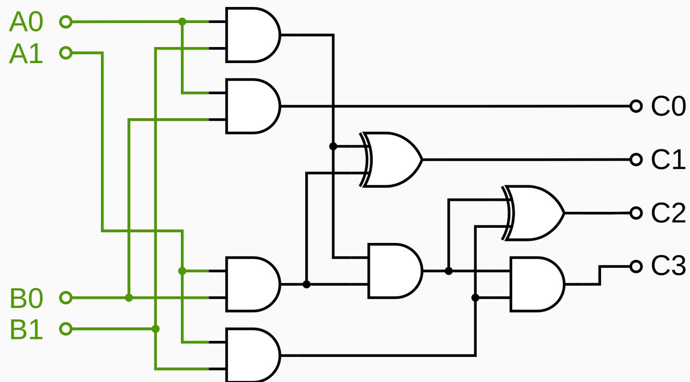
Safe voltages

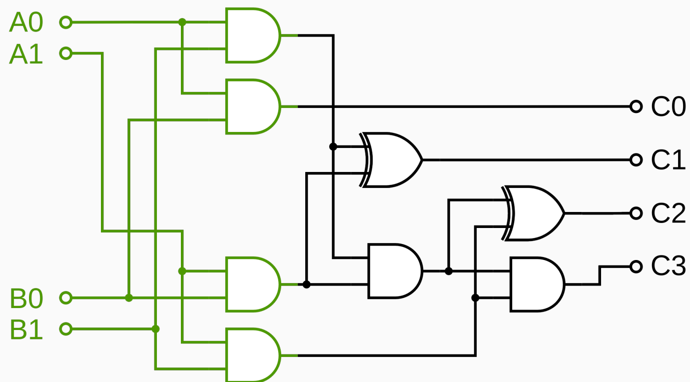


Correct Multiplication

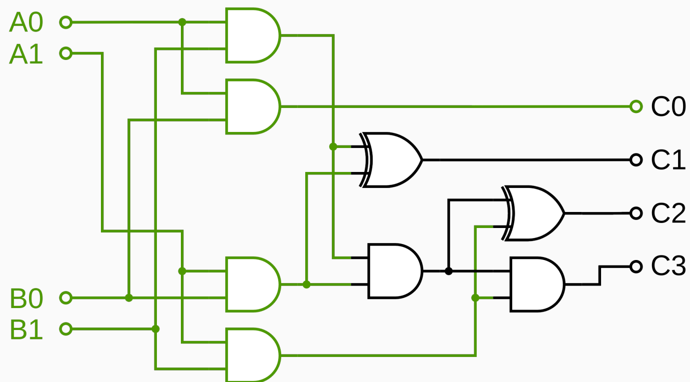



A0
A1
B0
B1
C0
C1
C2
C3

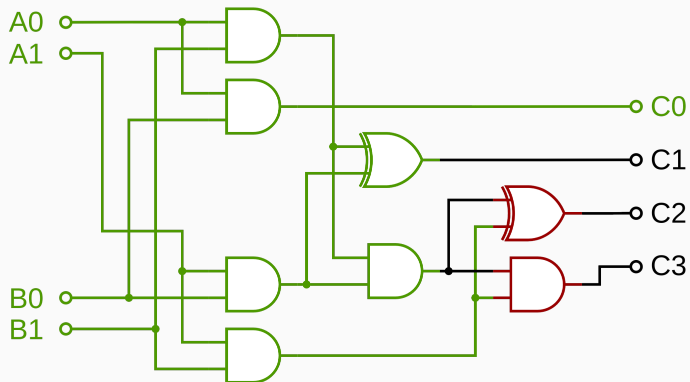

A0
A1
B0
B1
C0
C1
C2
C3


A0
A1
B0
B1
C0
C1
C2
C3

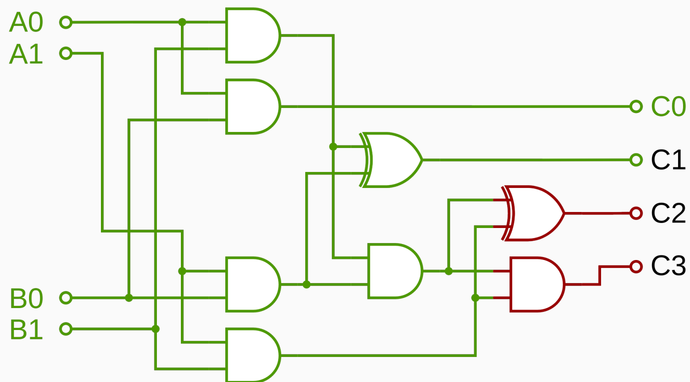
Correct Multiplication

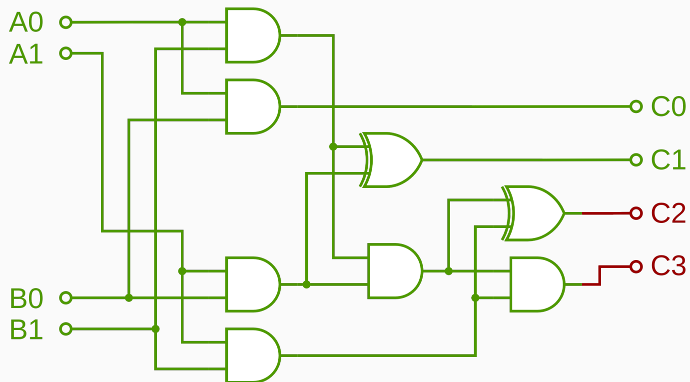


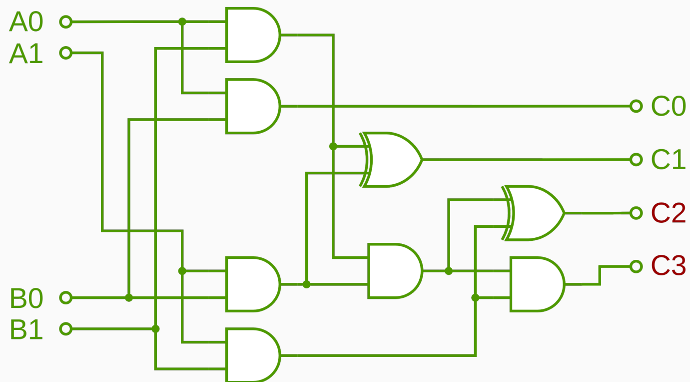
Correct Multiplication

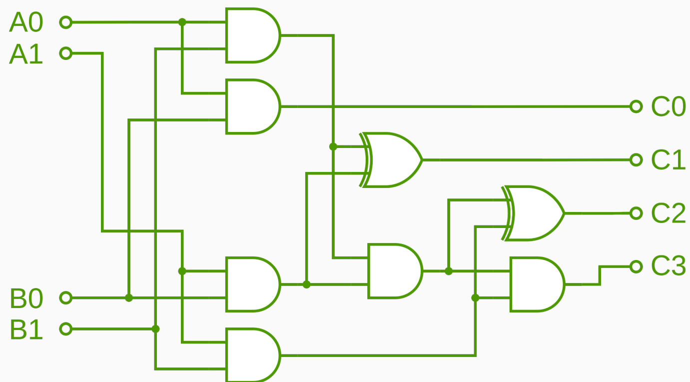


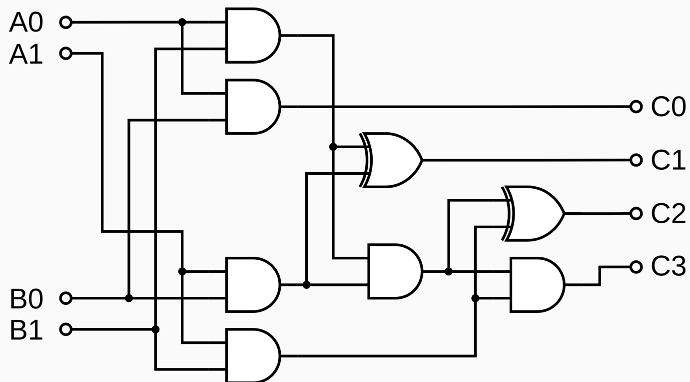
Correct Multiplication

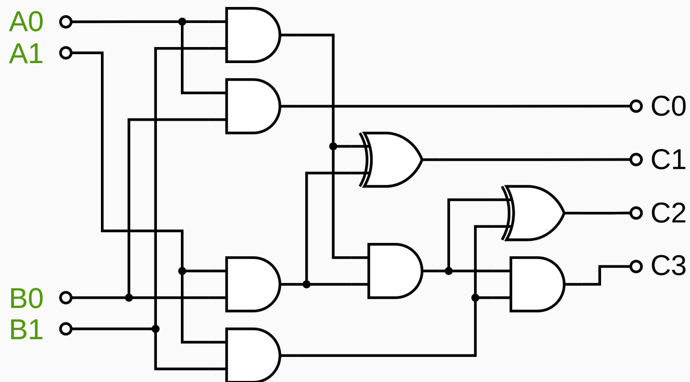


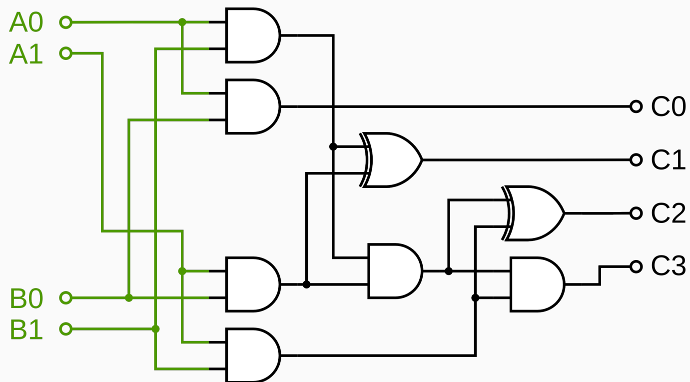

A0
A1
B0
B1
C0
C1
C2
C3

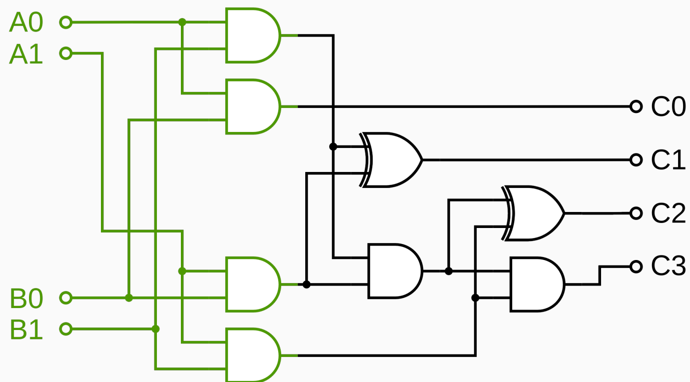

A0
A1
B0
B1
C0
C1
C2
C3

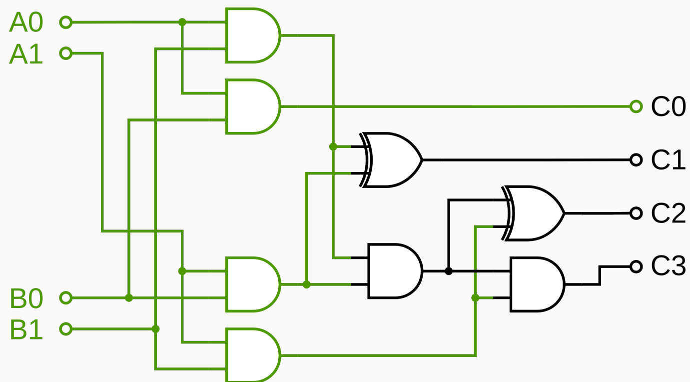

A0
A1
B0
B1
C0
C1
C2
C3

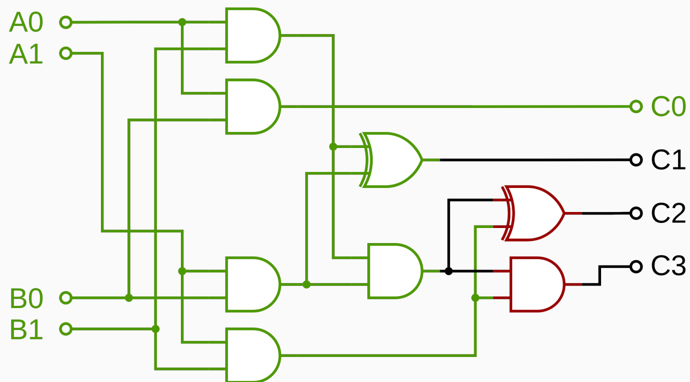

A0
A1
B0
B1
C0
C1
C2
C3

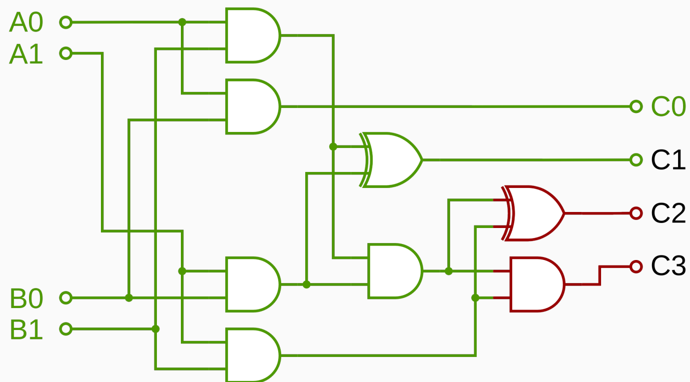

A0
A1
B0
B1
C0
C1
C2
C3


A0
A1
B0
B1
C0
C1
C2
C3

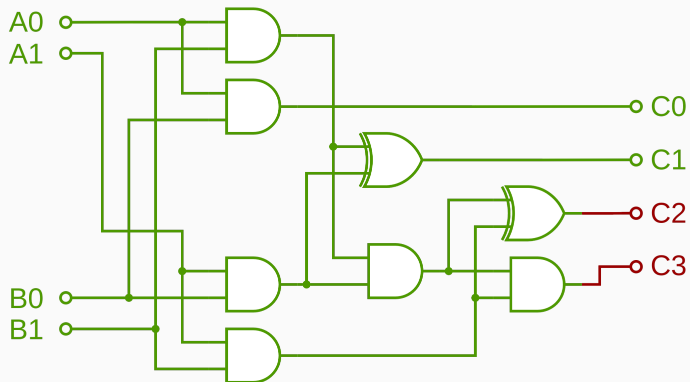

A0
A1
B0
B1
C0
C1
C2
C3

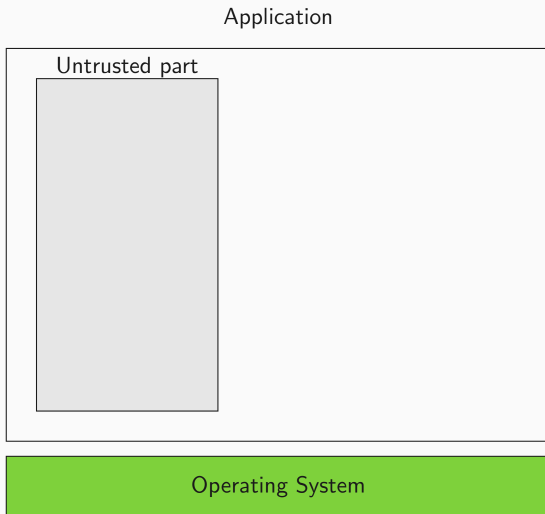

A0
A1
B0
B1
C0
C1
C2
C3

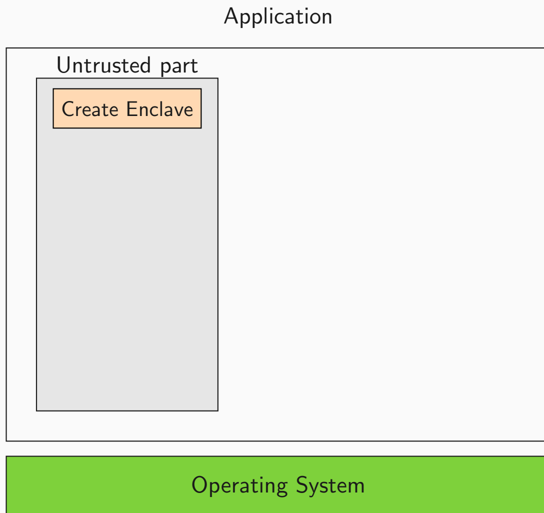

A0
A1
B0
B1
C0
C1
C2
C3

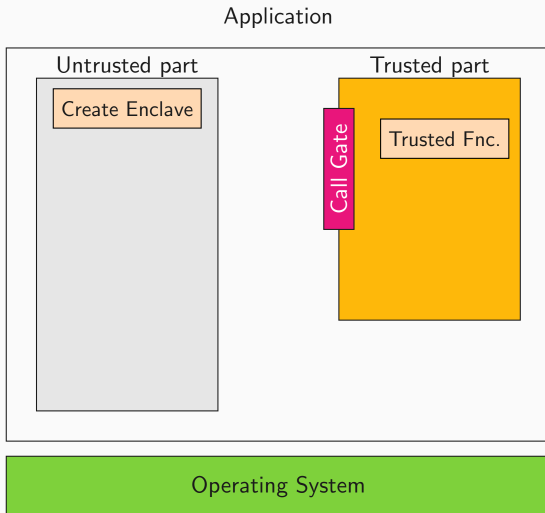

A0
A1
B0
B1
C0
C1
C2
C3

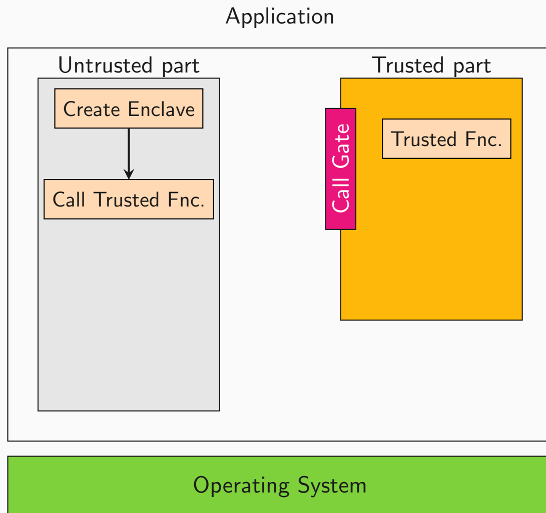
Faulty Multiplication

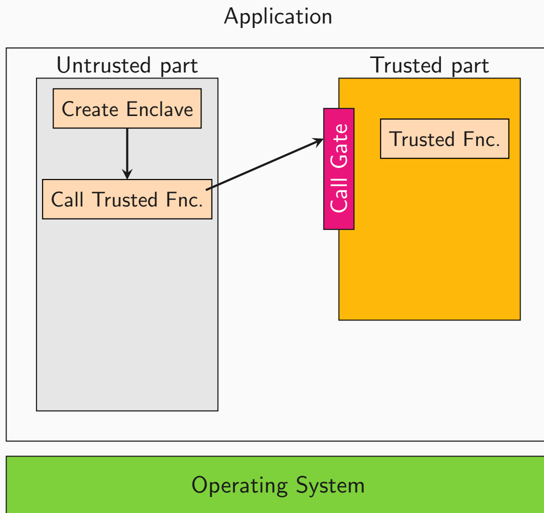


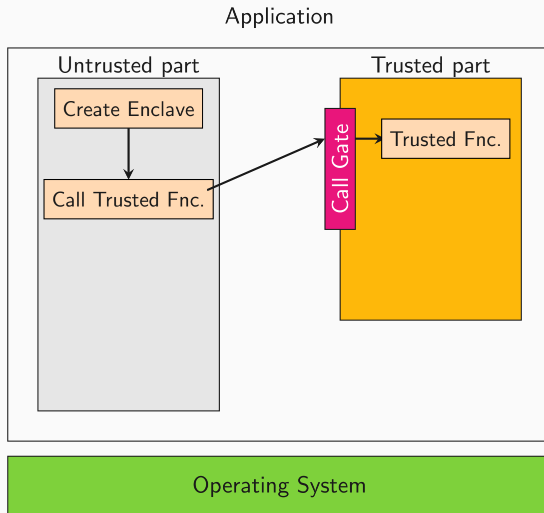

Application
Untrusted part
Operating System

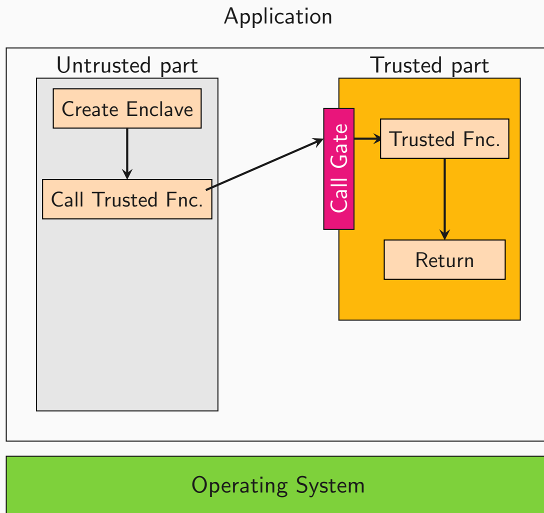

Application
Untrusted part
Create Enclave
Operating System

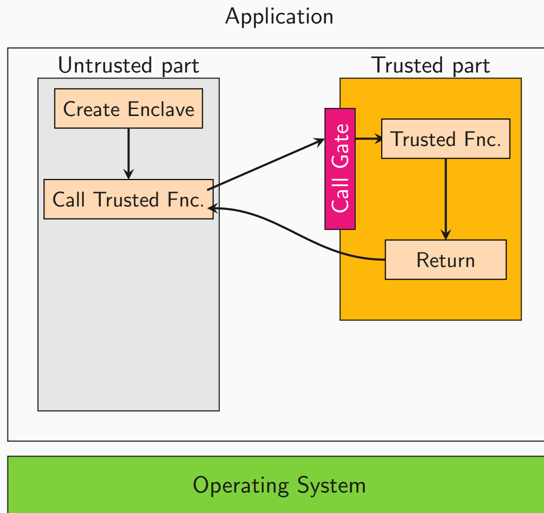

Application
Untrusted part
Create Enclave
Trusted part
Call Gate
Trusted Fnc.
Operating System

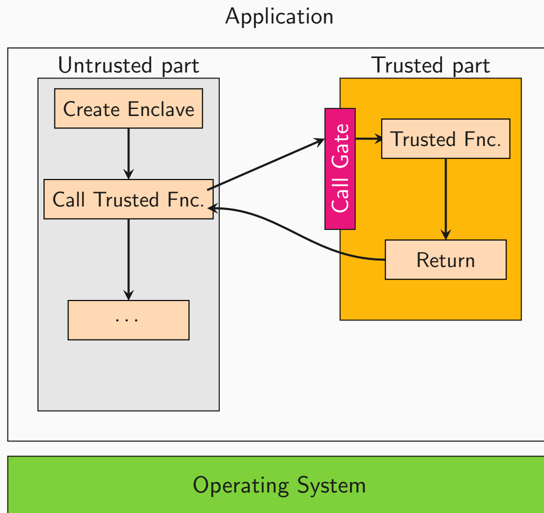

Application
Untrusted part
Create Enclave
Call Trusted Fnc.
Trusted part
Call Gate
Trusted Fnc.
Operating System

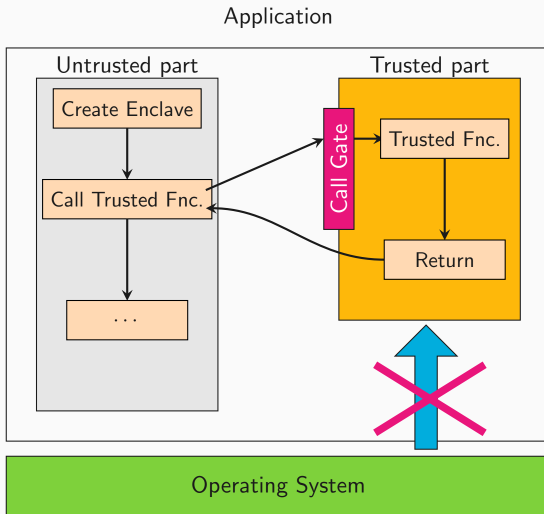

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Operating System


Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Operating System


Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Return
Operating System

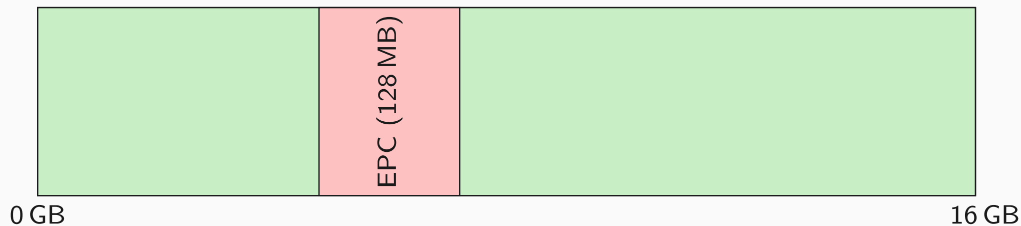

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Return
Operating System


Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
...
Call Gate
Trusted Fnc.
Return
Operating System

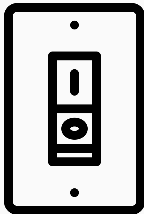

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
...
Call Gate
Trusted Fnc.
Return
Operating System

0 GB
16 GB

SGX Encrypted Memory

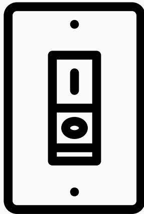


Bit Flips in the EPC

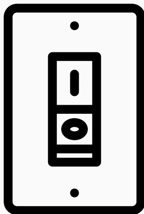


- Bit flips in the EPC?

Bit Flips in the EPC

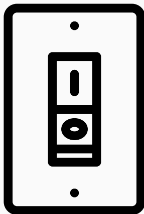


- Bit flips in the EPC?
- Integrity check fails!



- Bit flips in the EPC?
- Integrity check fails!

→ Lock up memory controller



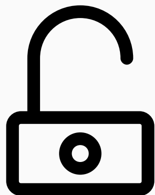
- Bit flips in the EPC?
- Integrity check fails!

→ Lock up memory controller

→ System halts immediately (no exploit, but DoS!)

Will SGX save us?

```
                                              [Userspace] Voltage: -261 mV
[Enclave] Multiplying 0xdeadbeef 0x1122334455667788
[Enclave] Multiply_it. result: 0.
```



- Public Key Crypto
- Encrypt/Sign messages
- Untrusted channel
- Encrypt/Verify messages with public key
- Decrypt/Sign messages with private key

RSA Signature with Chinese Remainder Theorem

$$n = \mathrm{p} \times q$$

$$S = H^{d} \bmod n$$

RSA Signature with Chinese Remainder Theorem

$$n = p \times q$$

$$S = H^{d} \bmod n$$

$$d_p = d \bmod p - 1$$

$$d_q = d \bmod q - 1$$

$$s_1 = H^{d_p} \bmod p$$

$$s_2 = H^{d_q} \bmod q$$

$$n = \mathrm{p} \times q$$

$$S = ((q^{-1} \bmod p)(s_1 - s_2) \bmod p) \times q + s_2$$

$$d_p = d \bmod p - 1$$

$$d_q = d \bmod q - 1$$

$$s_1 = H^{d_p} \mod p$$

$$s_2 = H^{d_q} \mod q$$

$$n = \mathrm{p} \times q$$

$$S = ((q^{-1} \bmod p)(s_1 - s_2) \bmod p) \times q + s_2$$

$$d_p = d \bmod p - 1$$

$$d_q = d \bmod q - 1$$

$$s_1 = H^{d_p} \bmod p$$

$$s_2 = H^{d_q} \bmod q$$

$$n = \mathrm{p} \times q$$

$$S = ((q^{-1} \bmod p)(s_1 - s_2) \bmod p) \times q + s_2$$

$$S' = ((q^{-1} \bmod p)(s_1 - s_2) \bmod p) \times q + s_2$$

$$n = p \times q$$

$$S = (\text{something}) \times q + s_2$$

$$S' = (\text{something else}) \times q + s_2$$

RSA Signature with Chinese Remainder Theorem

$$n = p \times q$$

$$S = (\text{something}) \times q + s_2$$

$$S' = (\text{something else}) \times q + s_2$$

$$S - S' = \text{something} \times q$$

$$n = p \times q$$

$$S = (\text{something}) \times q + s_2$$

$$S' = (\text{something else}) \times q + s_2$$

$$S - S' = \text{something} \times q$$

$$q = \gcd(S - S', n)$$

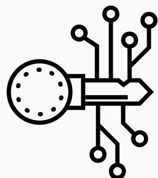
RSA decryption


```c
uint8_t rsa_dec_ecall(int iterations)
{
    // Wait for first fault
    trigger_fault(iterations);

    // Actual decryption
    ippsRSA_Decrypt(ct, dec, pPrv, scratchBuffer);
}
```

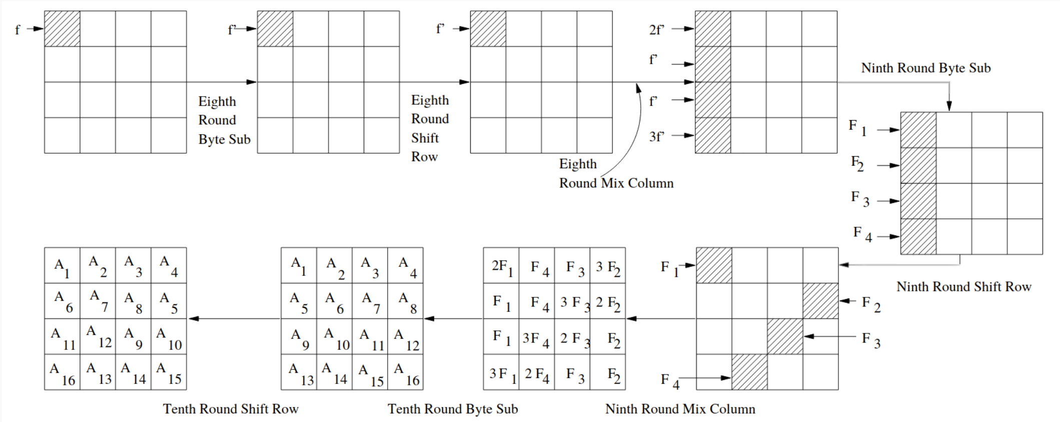
```
bagger> dog Enclave/encl
```

WHAT ELSE CAN WE BREAK?!

AES Basics



- Symmetric Key Crypto
- Encrypt messages for transfer over public channel
- Encrypt data for (untrusted) storage
- 4×4 byte state
- 10 rounds: S-Box, ShiftRows, MixColumns, AddRoundKey



Tunstall2011

AES-NI (New Instructions)

Instruction	Description
`AESENC`	Perform one round of an AES encryption flow
`AESENCLAST`	Perform the last round of an AES encryption flow
`AESDEC`	Perform one round of an AES decryption flow
`AESDECLAST`	Perform the last round of an AES decryption flow
`AESKEYGENASSIST`	Assist in AES round key generation
`AESIMC`	Assist in AES Inverse Mix Columns
`PCLMULQDQ`	Carryless multiply (CLMUL)

AES New Instructions - inside SGX


```
do
{
        i++;
        plaintext = <randomly generated>

        result1 = aes128_enc(plaintext);
        result2 = aes128_enc(plaintext);
} while (vec_equal_128(result1,result2) && i<iterations);
```

```
bagger> sudo ./aes-encrypt 100000 -262
```

THIS IS MORE THAN JUST CRYPTO

But what about memory corruption?


```
struct_foo_t *foo = &arr[offset];
foo->foo = enclave_secret;
```

But what about memory corruption?

```
struct_foo_t *foo = &arr[offset];
foo->foo = enclave_secret;
```

But what about memory corruption?

```
foo = arr + offset * 0x24
```

But what about memory corruption?

```
foo = arr + offset * 0x24
```

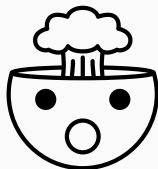
```
Creating enclave...
==== Victim Enclave ====
[pt.c] /dev/sgx-step opened!
Enclave Base:    0x7f001a000000
Enclave Limit:   0x7f001c000000
EDBGRD: debug
```

Voltage
0.584V

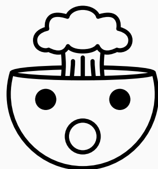
Undervolting
-235mV

More than just crypto?

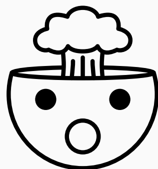
More than just crypto?



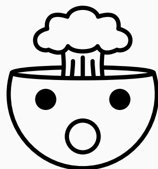
More than just crypto?



More than just crypto?

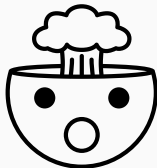


- RSA keys

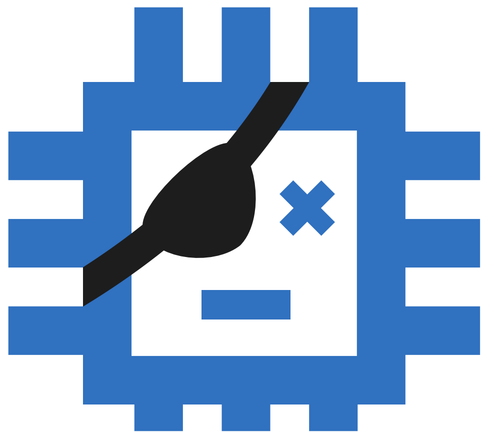


- RSA keys
- AES keys

More than just crypto?



- RSA keys
- AES keys
- Memory corruption



PLUNDER
VOLT

- A new type of attack

Plundervolt TLDR

- A new type of attack
- Breaks the integrity of SGX

Plundervolt TLDR

- A new type of attack
- Breaks the integrity of SGX
- Within SGX we:

- A new type of attack
- Breaks the integrity of SGX
- Within SGX we:
  - Retrieve keys from AES-NI

Plundervolt TLDR

- A new type of attack
- Breaks the integrity of SGX
- Within SGX we:
    - Retrieve keys from AES-NI
    - Retrieve RSA signature key

Plundervolt TLDR

- A new type of attack
- Breaks the integrity of SGX
- Within SGX we:
  - Retrieve keys from AES-NI
  - Retrieve RSA signature key
  - Induce memory corruption in bug free code

Plundervolt TLDR

- A new type of attack
- Breaks the integrity of SGX
- Within SGX we:
  - Retrieve keys from AES-NI
  - Retrieve RSA signature key
  - Induce memory corruption in bug free code
  - Make enclave write secrets to untrusted memory

Side-Channel Security

Chapter 6: Software-based Fault Attacks

Jonas Juffinger

April 11, 2024

www.iaik.tugraz.at

References

[2] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. "ANVIL: Software-based protection against next-generation Rowhammer attacks". In: *ACM SIGPLAN Notices* (2016).

[5] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. "CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory". In: *USENIX Security*. 2017.

[6] Yueqiang Cheng, Zhi Zhang, and Surya Nepal. "Still Hammerable and Exploitable: on the Effectiveness of Software-only Physical Kernel Isolation". In: *arXiv:1802.07060* (2018).

[7] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. "Real time detection of cache-based side-channel attacks using Hardware Performance Counters". In: *Cryptology ePrint Archive, Report 2015/1034* (2015).

[0] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks". In: *S&P*. 2019.

[9] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R Hanebutte, and Onur Mutlu. "Memory power management via dynamic voltage/frequency scaling". In: *ACM International Conference on Autonomic Computing*. 2011.

[11] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. "TRRespass: Exploiting the Many Sides of Target Row Refresh". In: *S&P*. 2020.

[12] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. "Another Flip in the Wall of Rowhammer Defenses". In: *S&P*. 2018.

[13] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: *DIMVA*. 2016.

[14] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. "Flush+Flush: A Fast and Stealthy Cache Attack". In: *DIMVA*. 2016.

[15] Nishad Herath and Anders Fogh. "These are Not Your Grand Daddys CPU Performance Counters – CPU Hardware Performance Counters for Security". In: *Black Hat USA*. 2015.

[16] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. "MASCAT: Stopping Microarchitectural Attacks Before Execution". In: *Cryptology ePrint Archive, Report 2016/1196* (2017).

[17] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. "SGX-Bomb: Locking Down the Processor via Rowhammer Attack". In: *SysTEX*. 2017.

[18] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. "BLACKSMITH: Rowhammering in the Frequency Domain". In: *S&P*. Nov. 2021.

[19] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Maria Eichlseder, Moritz Lipp, and Daniel Gruss. "CSI: Rowhammer - Cryptographic Security and Integrity against Rowhammer". In: *S&P*. 2023.

[20] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors". In: *ISCA*. 2014.

[21] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. "Half-Double: Hammering From the Next Row Over". In: *USENIX Security*. 2022.

[22] Mark Lanteigne. *How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware*. 2016. URL: http://www.thirdio.com/rowhammer.pdf.

[23] Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, Lukas Raab, and Lukas Lamster. "Nethammer: Inducing Rowhammer Faults through Network Requests". In: *SILM Workshop*. 2020.

[0] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. "RowPress: Amplifying Read Disturbance in Modern DRAM Chips". In: *ISCA*. 2023.

[24] Matthias Payer. “HexPADS: a platform to detect “stealth” attacks”. In: *ESSoS*. 2016.

[25] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: *USENIX Security*. 2016.

[27] Rui Qiao and Mark Seaborn. “A New Approach for Rowhammer Attacks”. In: *HOST*. 2016.

[29] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. “SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript”. In: *USENIX Security*. 2021.

[30] Mark Seaborn. *How physical addresses map to rows and banks in DRAM*. 2015. URL: http://lackingrhoticity.blogspot.com/2015/05/how-physical-addresses-map-to-rows-and-banks.html.

[31] Mark Seaborn and Thomas Dullien. “Exploiting the DRAM Rowhammer bug to gain kernel privileges”. In: *Black Hat USA*. 2015.

[32] Andrei Tatar, Radhesh Krishnan, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. “Throwhammer: Rowhammer Attacks over the Network and Defenses”. In: *USENIX ATC*. 2018.

[33] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: *CCS*. 2016.

[34] Victor van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. “GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM”. In: *DIMVA*. 2018.

[35] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. “One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation”. In: *USENIX Security*. 2016.