

Side-Channel Security

Chapter 5: Software-based Fault Attacks

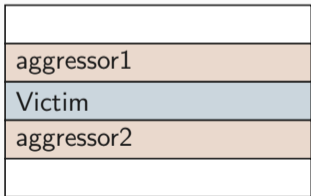
Jonas Juffinger

March 27, 2025

www.isec.tugraz.at

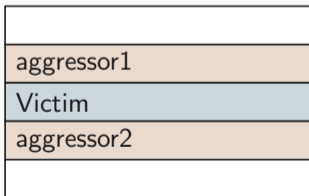
ROWHAMMER

- Hardware fault of the DRAM [17]



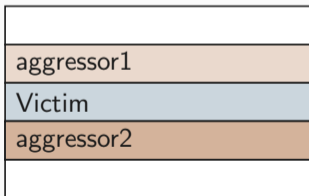
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows



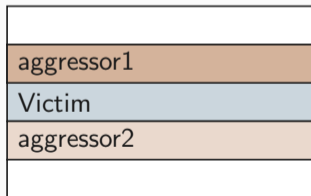
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows



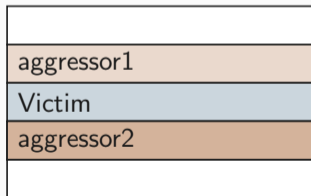
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows



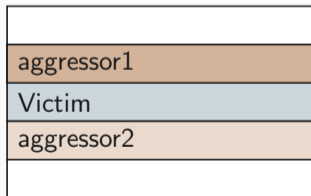
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows



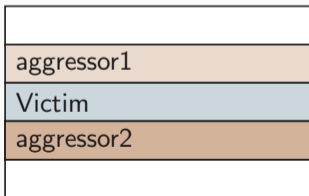
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows



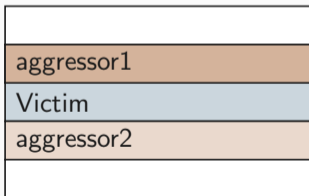
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```


- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows



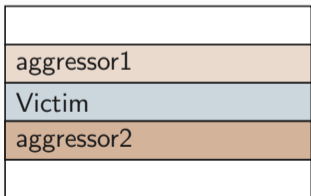
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows



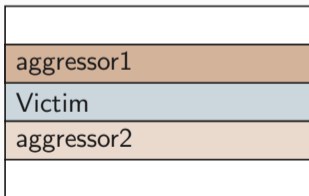
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows



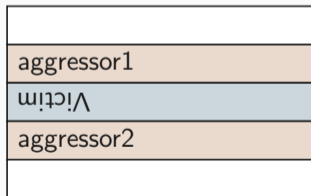
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows



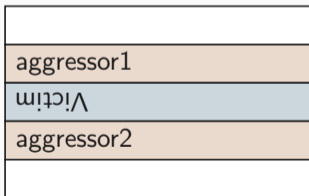
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows



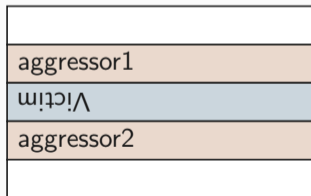
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows
- Worse with every new DRAM generation



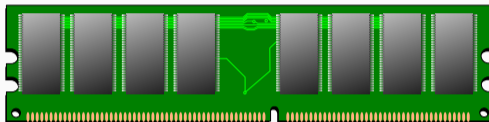
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

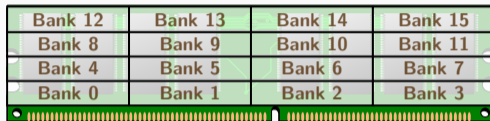
- Hardware fault of the DRAM [17]
- Frequent accesses flip bits in neighboring rows
- Worse with every new DRAM generation
- Enables attacks, many countermeasures proposed

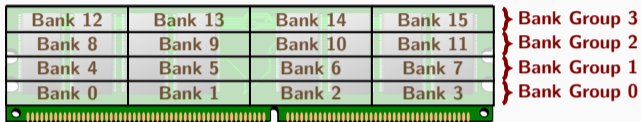


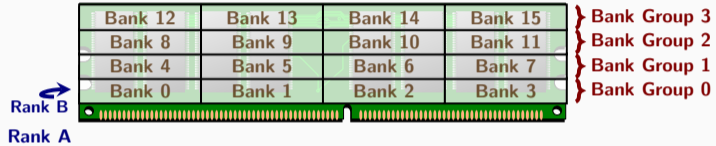
```
for (i = 0; i < N; ++i) {  
    *aggressor1;  
    *aggressor2;  
    flush (aggressor1);  
    flush (aggressor2);  
}
```

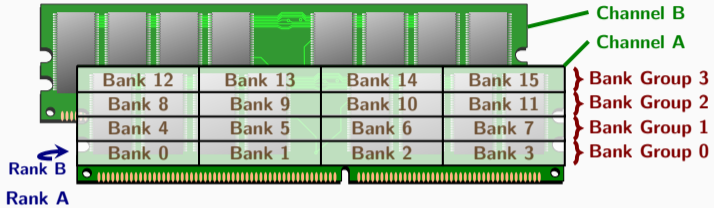
Background

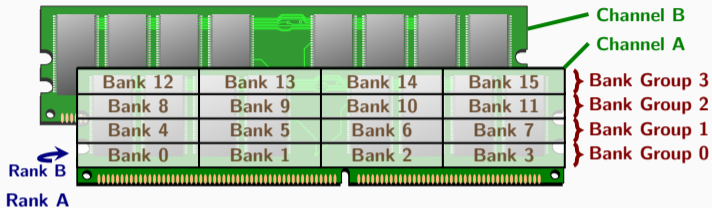




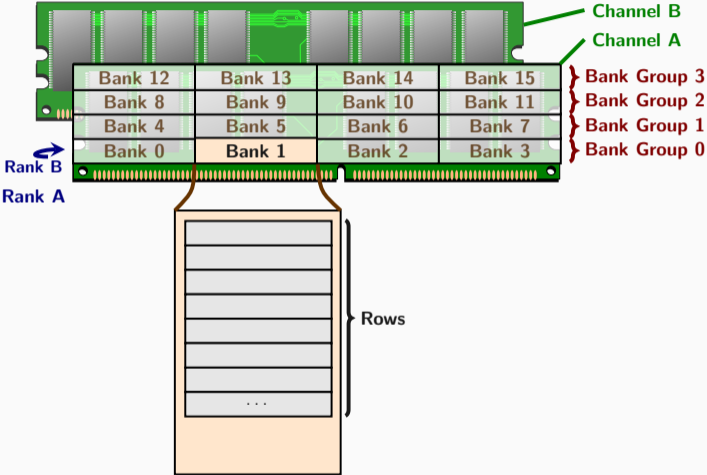


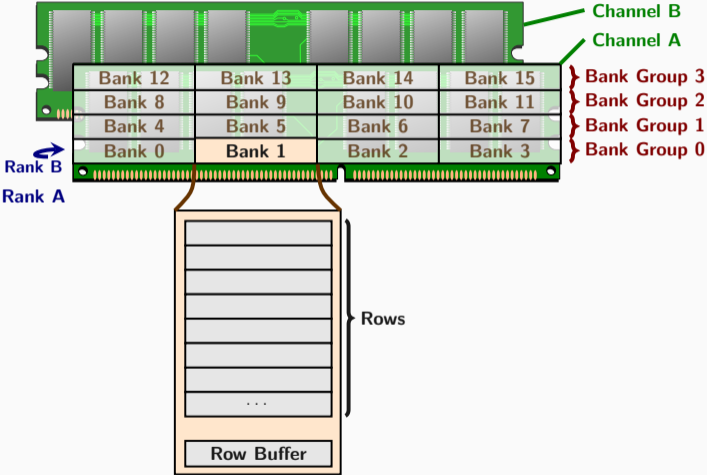


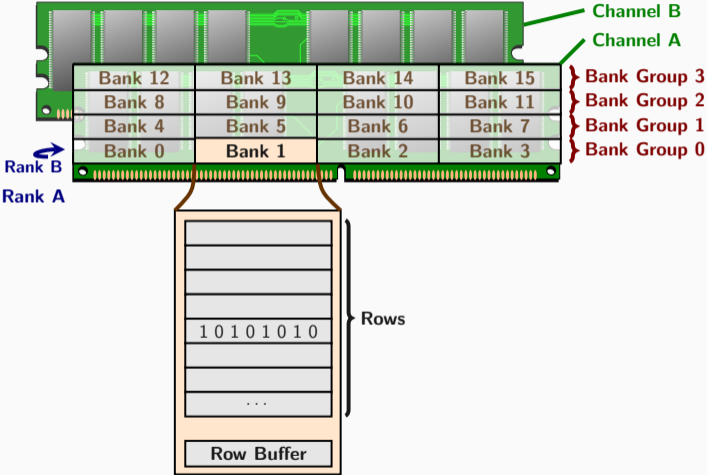


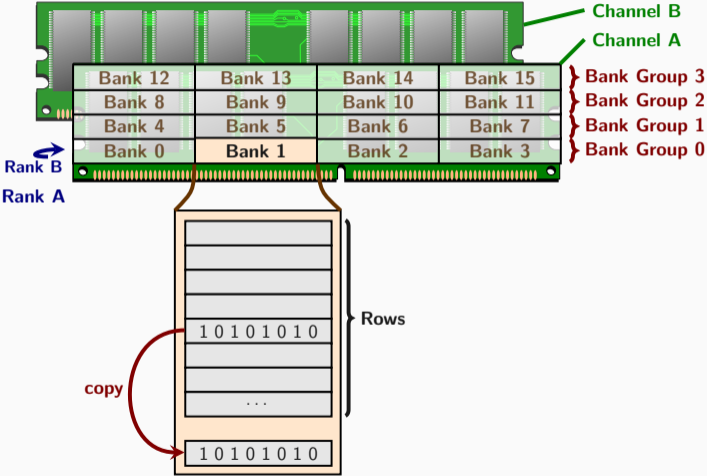


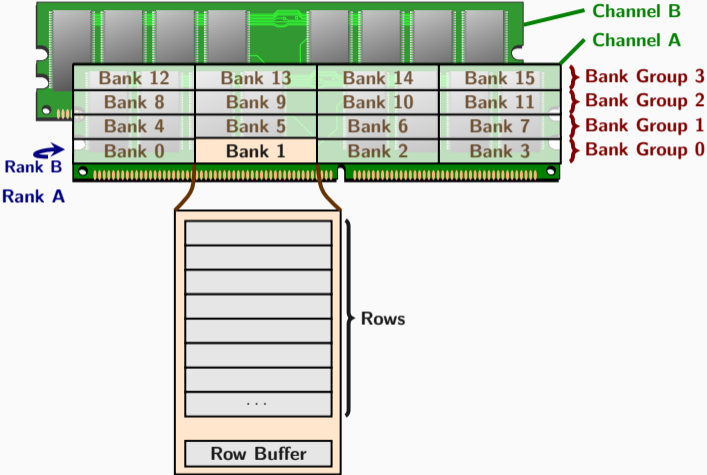
4 banks × 4 bank groups × 2 ranks × 2 channels = 64 banks total

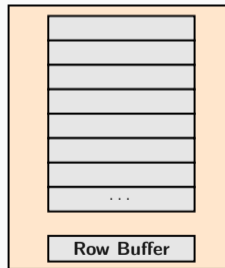


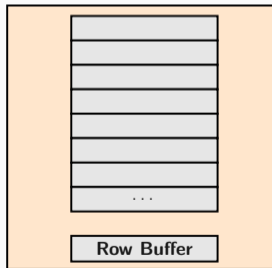


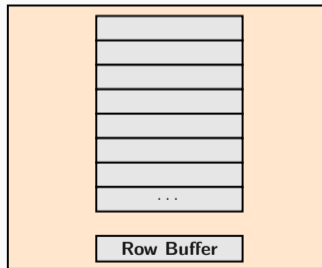


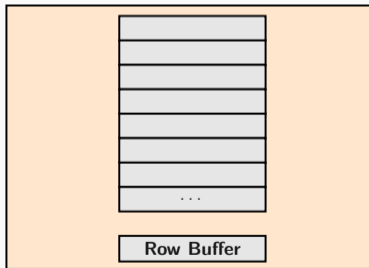


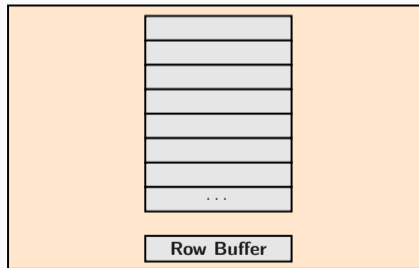


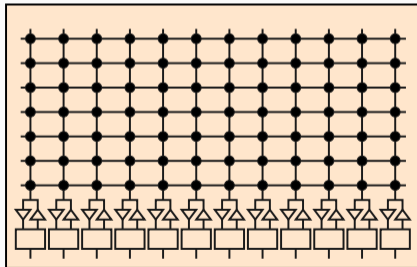


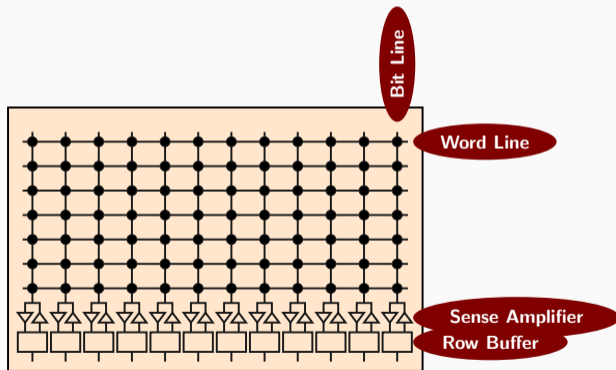


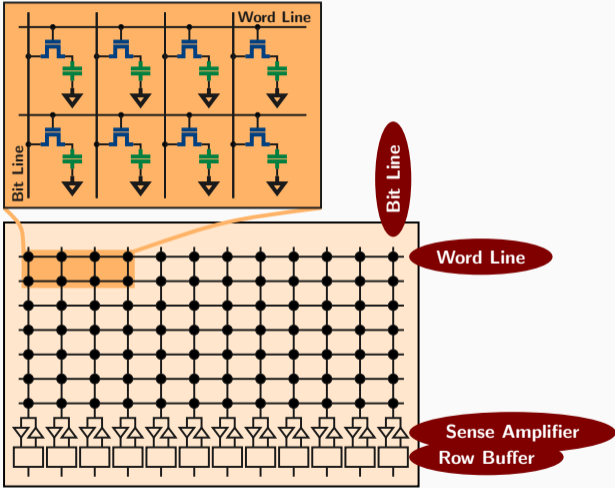


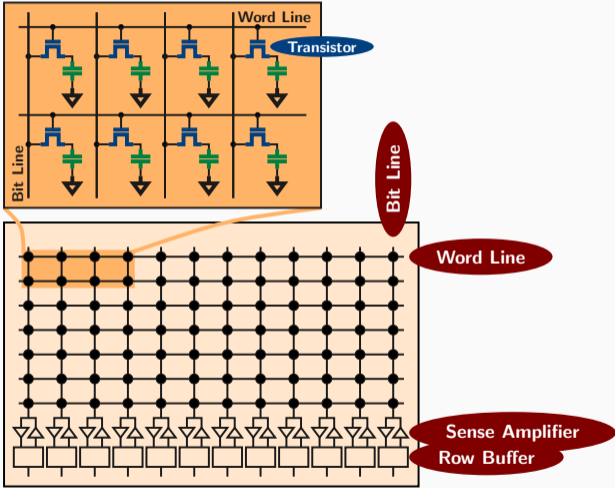


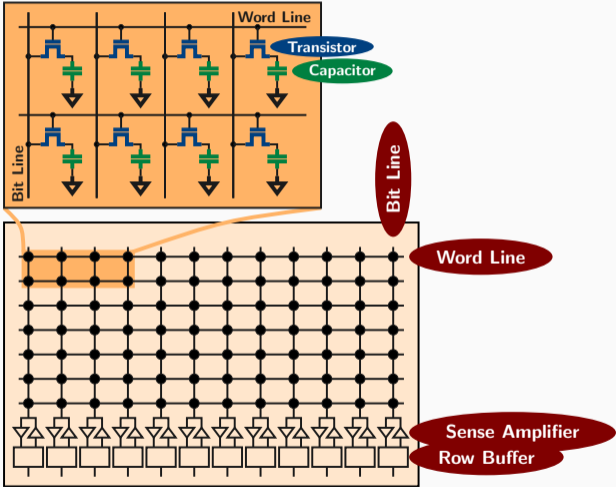


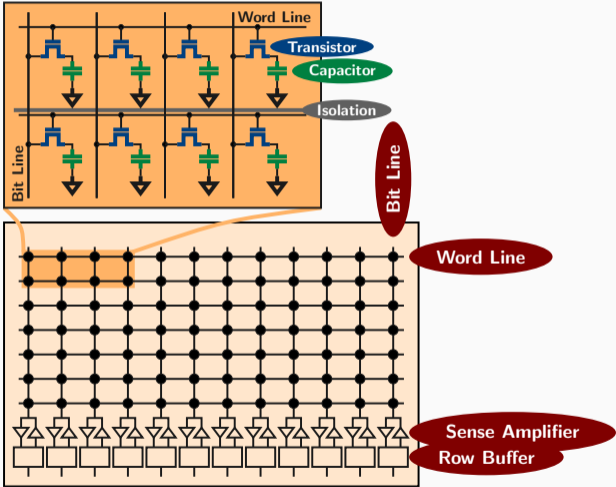


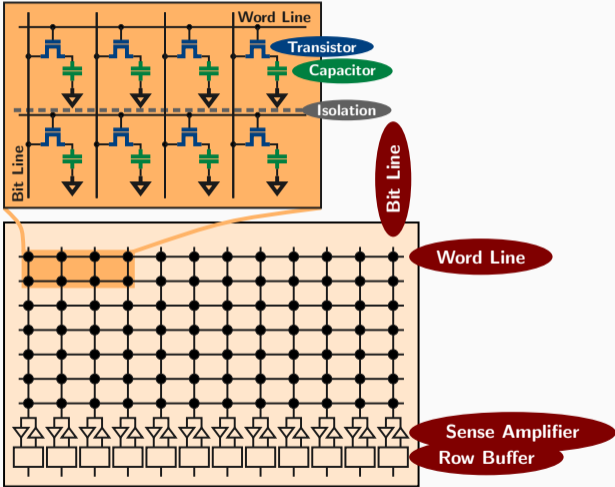


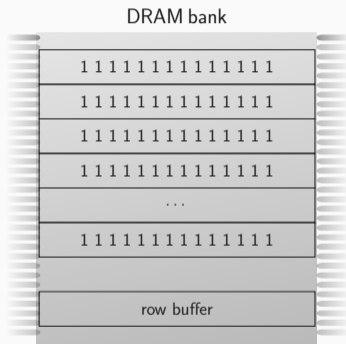




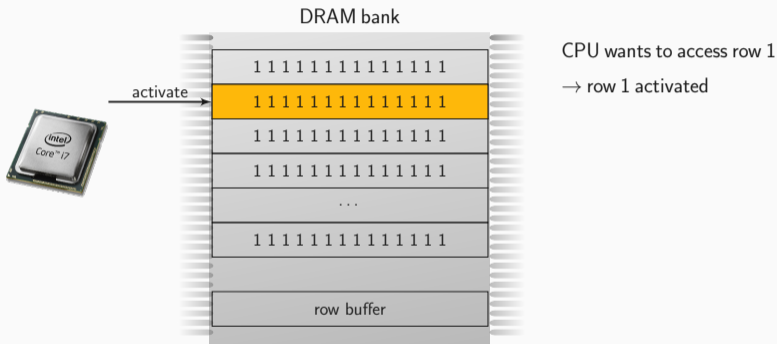


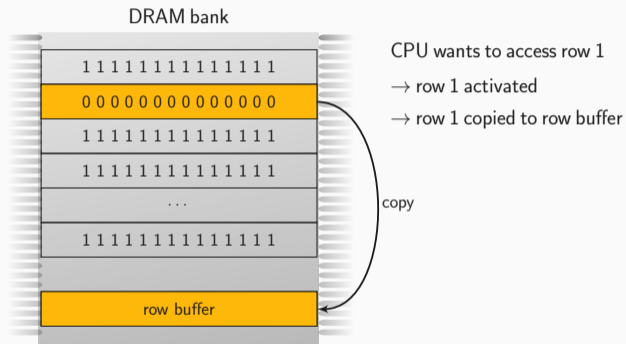


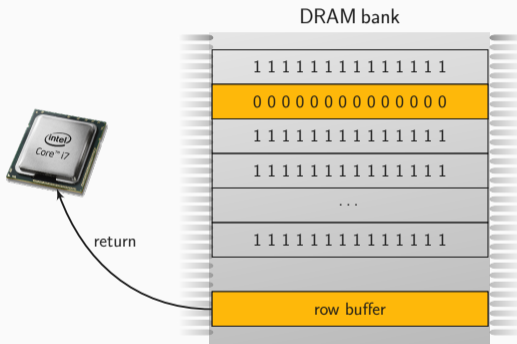




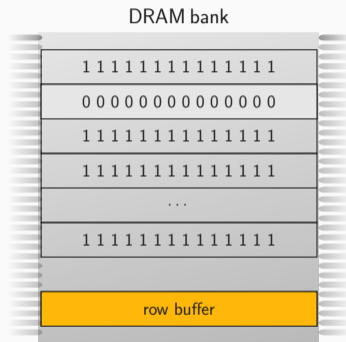
CPU wants to access row 1



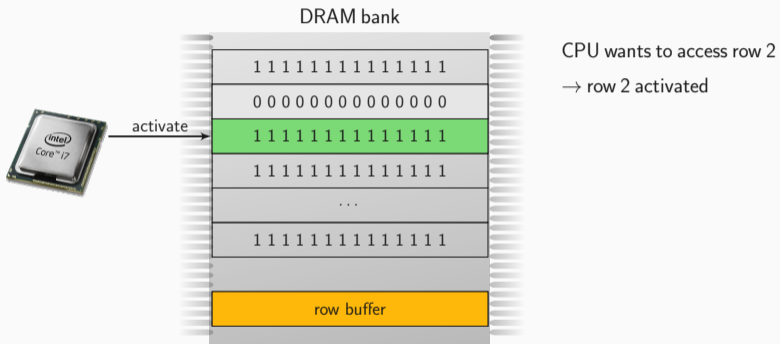


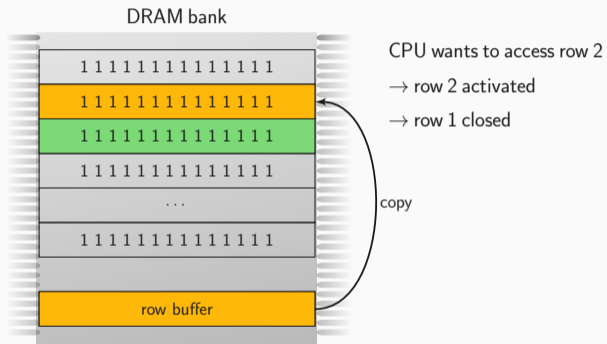


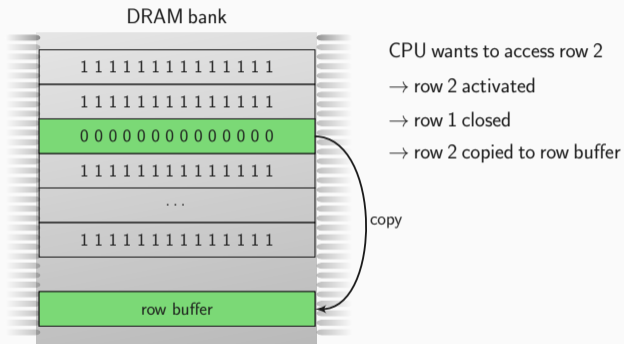
- CPU wants to access row 1
- row 1 activated
- row 1 copied to row buffer

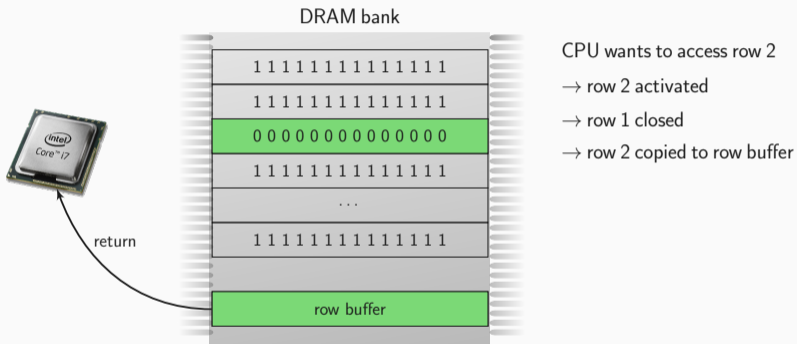


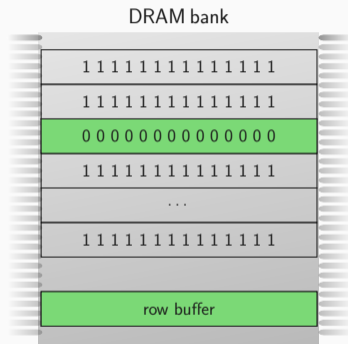
CPU wants to access row 2



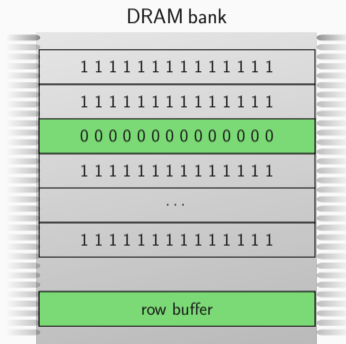




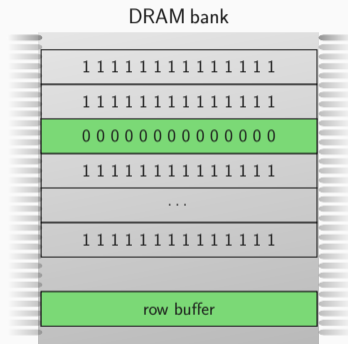




→ slow (row conflict)

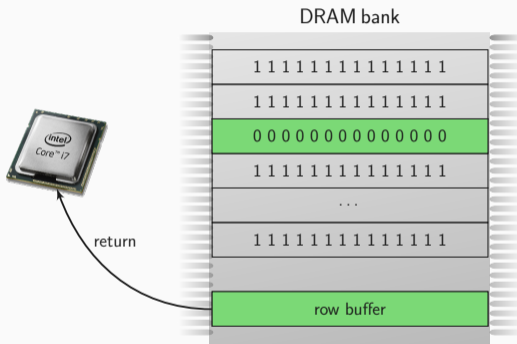


CPU wants to access row 2—again



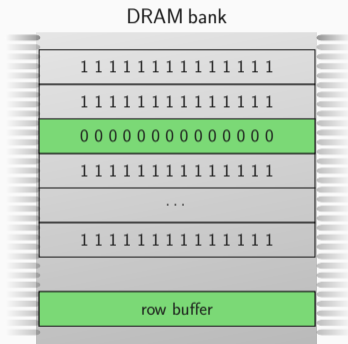
CPU wants to access row 2—again

→ row 2 already in row buffer



CPU wants to access row 2—again

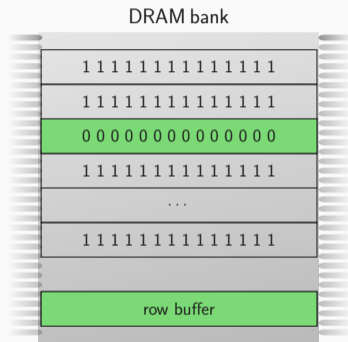
→ row 2 already in row buffer



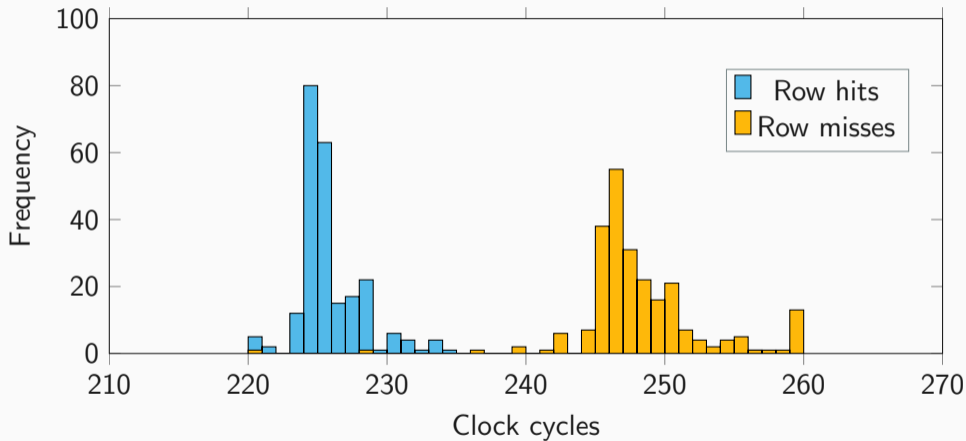
CPU wants to access row 2—again

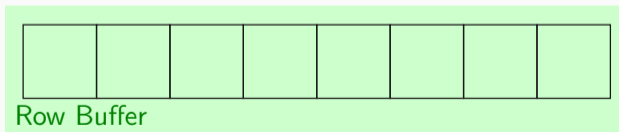
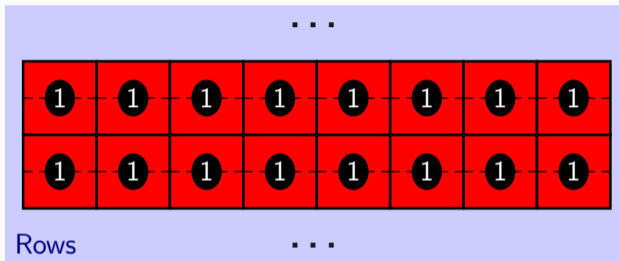
→ row 2 already in row buffer

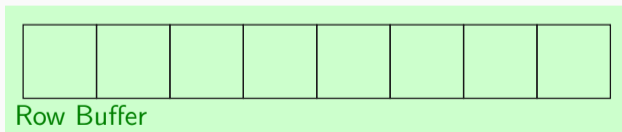
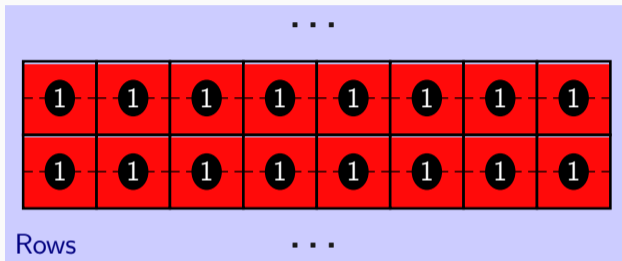
→ **fast** (row hit)

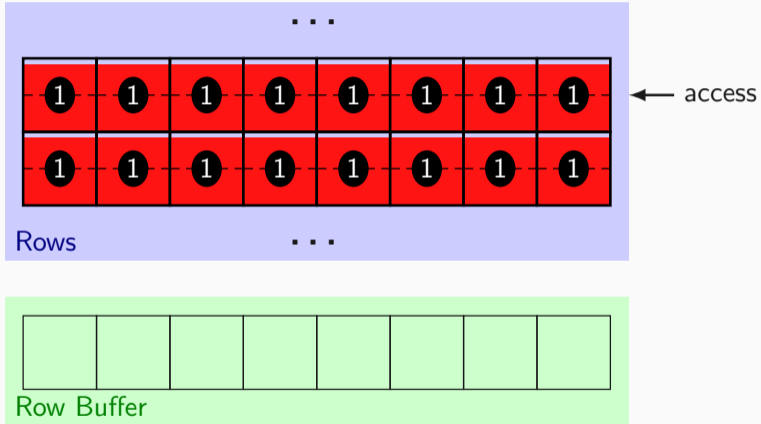


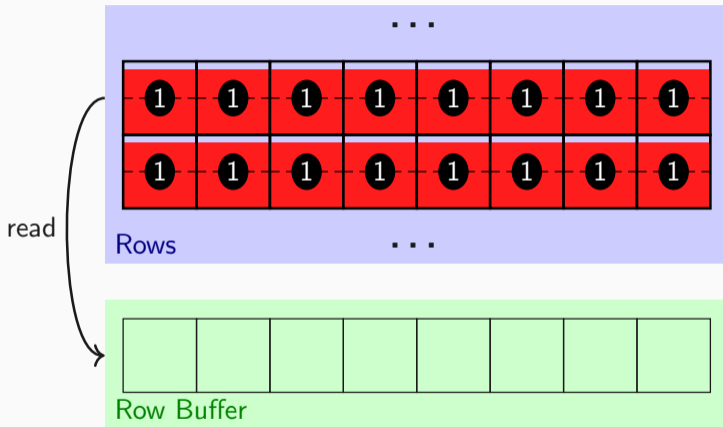
row buffer = cache

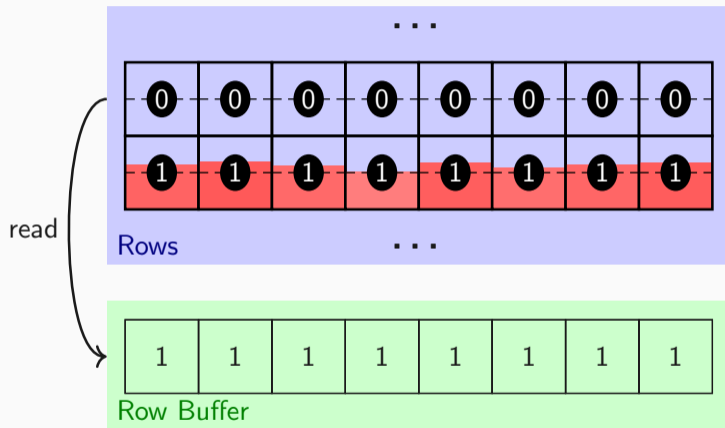


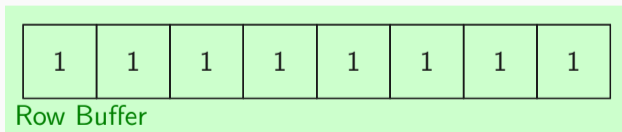
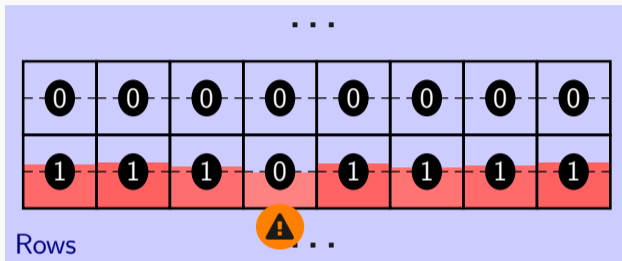


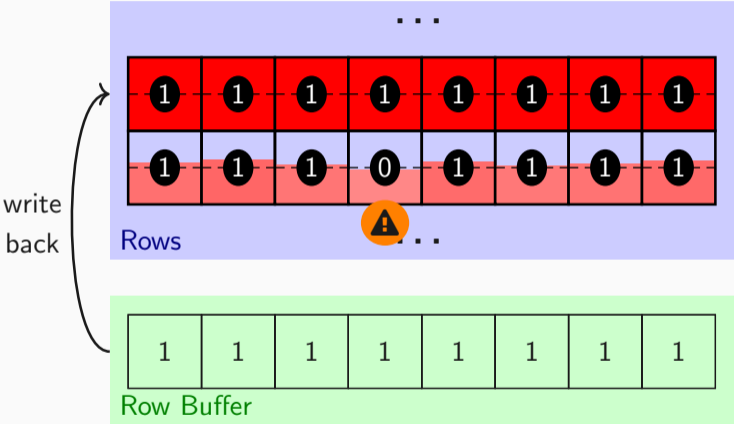


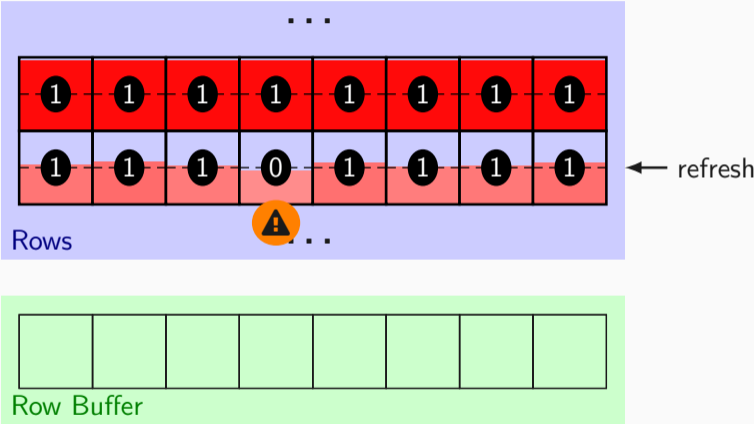


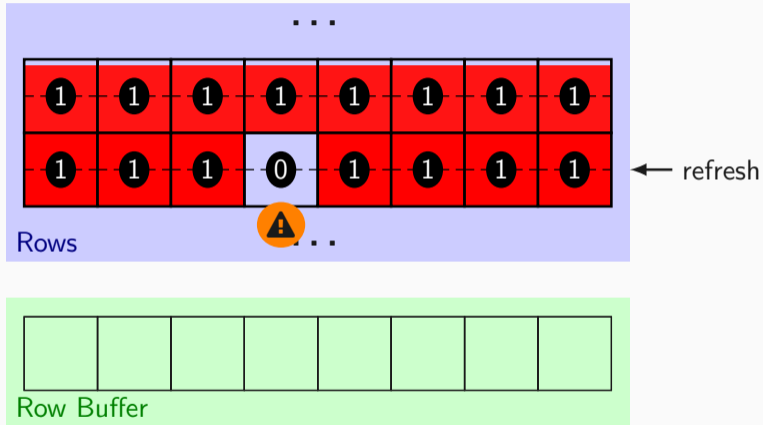




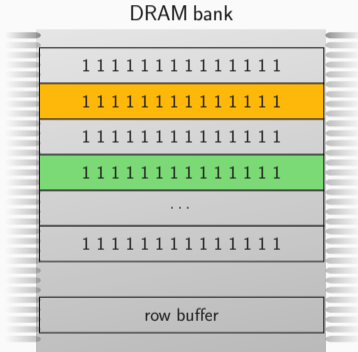


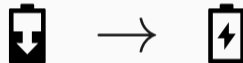
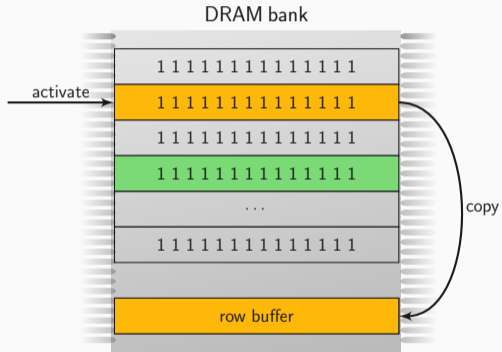


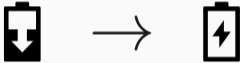
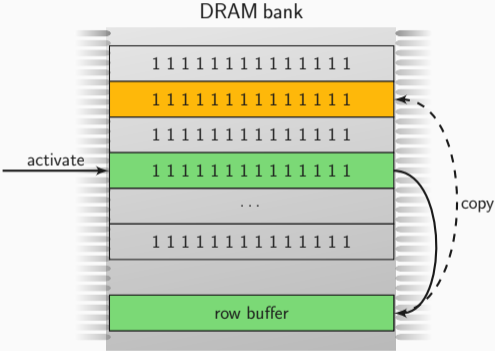




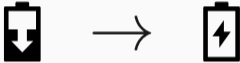
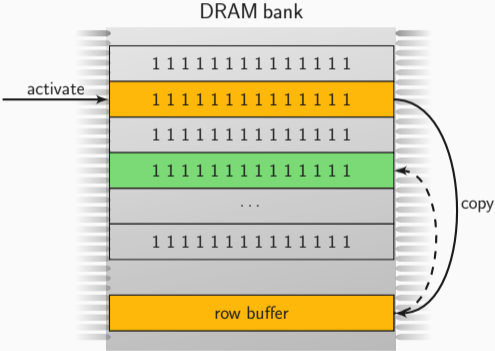
Rowhammer



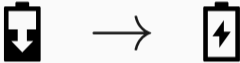
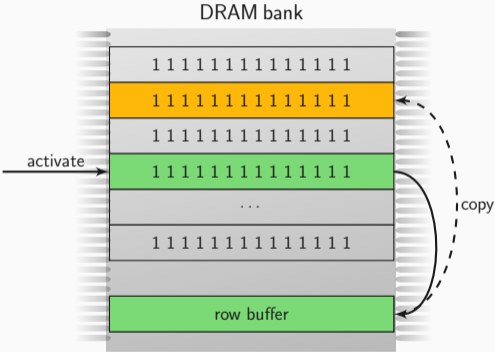




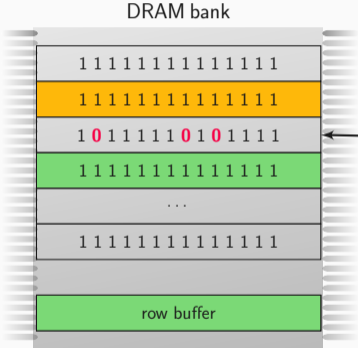
Cells leak faster upon proximate accesses → Rowhammer



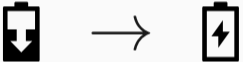
Cells leak faster upon proximate accesses → Rowhammer



Cells leak faster upon proximate accesses → Rowhammer



bit flips in row 2!



Cells leak faster upon proximate accesses → Rowhammer

How widespread is the issue?





- 85% affected [17] (see Figure)





- 85% affected [17] (see Figure)



- First believed to be safe



DDR3

- 85% affected [17] (see Figure)

DDR4

- First believed to be safe
- > 90% affected (with active countermeasures) [14]

DDR5

DDR3

- 85% affected [17] (see Figure)

DDR4

- First believed to be safe
- > 90% affected (with active countermeasures) [14]

DDR5

- One DIMM found

DDR3

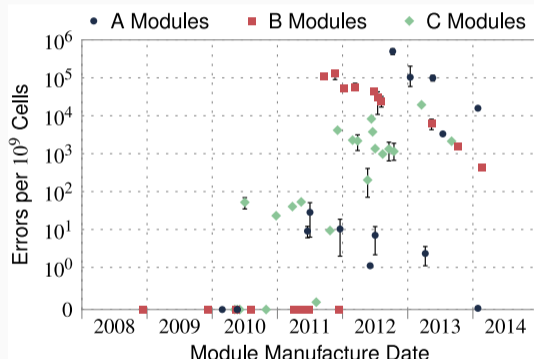
- 85% affected [17] (see Figure)

DDR4

- First believed to be safe
- > 90% affected (with active countermeasures) [14]

DDRS

- One DIMM found



A still from the movie Toy Story showing Woody and Buzz Lightyear. Woody is on the left, looking slightly concerned. Buzz is on the right, wearing his green and white space suit, with his arms raised in a celebratory gesture. The background is a simple room with a door and some toys on the floor.

BIT FLIPS

BIT FLIPS EVERYWHERE



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?







- `clflush` instruction → original paper [17]



- `clflush` instruction → original paper [17]
- cache eviction [1, 9]

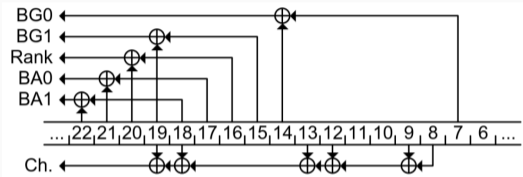


- `clflush` instruction → original paper [17]
- cache eviction [1, 9]
- non-temporal accesses [24]

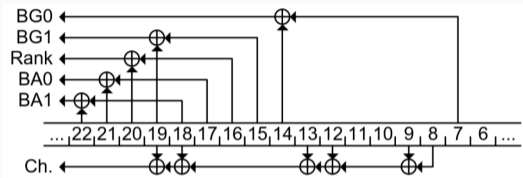


- `clflush` instruction → original paper [17]
- cache eviction [1, 9]
- non-temporal accesses [24]
- uncached memory [29]

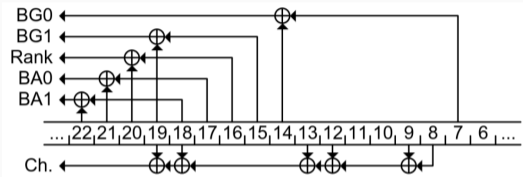
How do we target accesses?



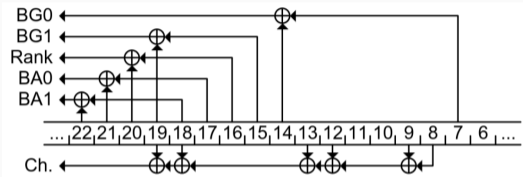
- fixed map: physical addresses → DRAM cells



- 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, ... [23, 26]



- fixed map: physical addresses → DRAM cells
- **undocumented** for Intel
- reverse-engineering for Sandy Bridge, Sandy, Ivy, Haswell, Skylake, ... [23, 26]
- using the timing difference between row hits and row conflicts







- They are not random → highly reproducible flip pattern!



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



- 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



- 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
 4. Trigger bit flip again

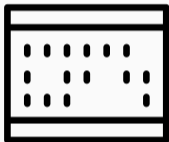
P	RW	US	WT	UC	R	D	S	G		
[Redacted]										
[Redacted]										
[Redacted]				[Green]						X

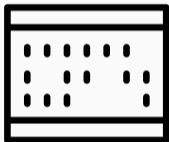
P	RW	US	WT	UC	R	D	S	G	Ignored	
				Ignored						X

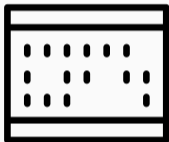
P	RW	US	WT	UC	R	D	S	G	Ignored	
Physical Page Number										
				Ignored						X

P	RW	US	WT	UC	R	D	S	G	Ignored	
Physical Page Number										
				Ignored						X

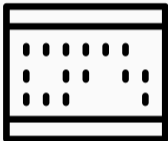
Each 4 KB page table consists of 512 such entries



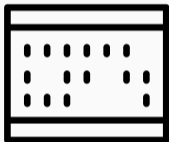




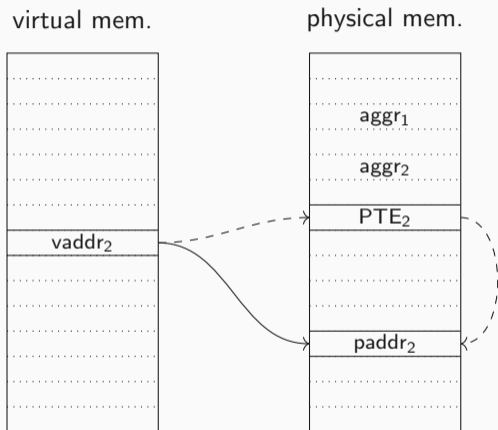
1. Scan for flips

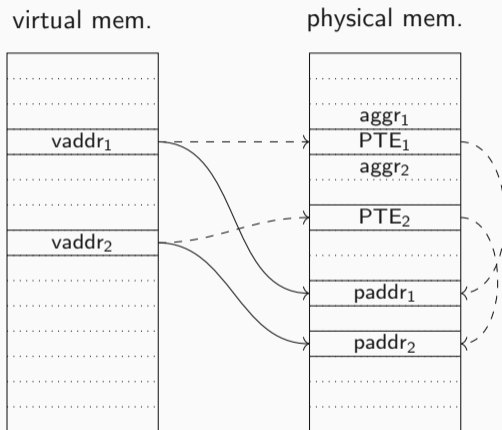


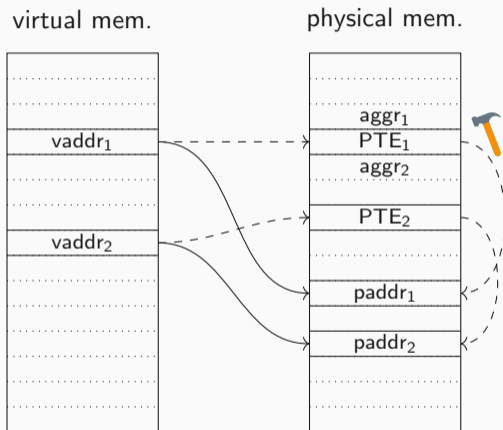
1. Scan for flips
2. Exhaust or massage memory to place a page table at target location

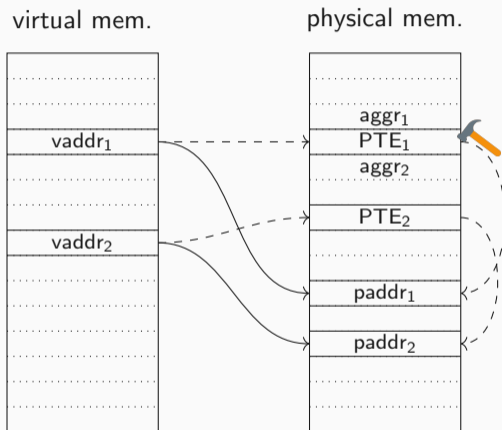


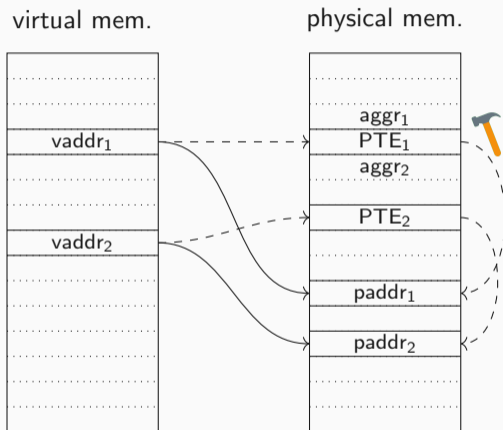
1. Scan for flips
2. Exhaust or massage memory to place a page table at target location
3. Gain access to your own page table → kernel privileges

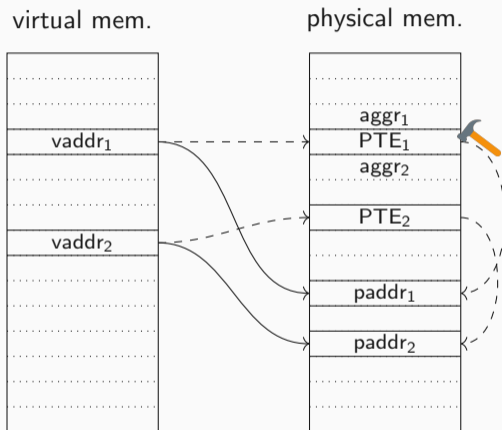


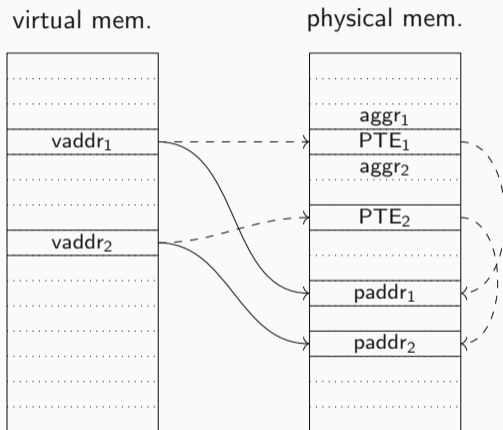


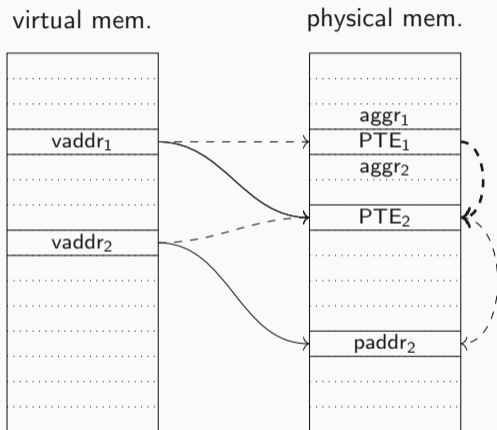


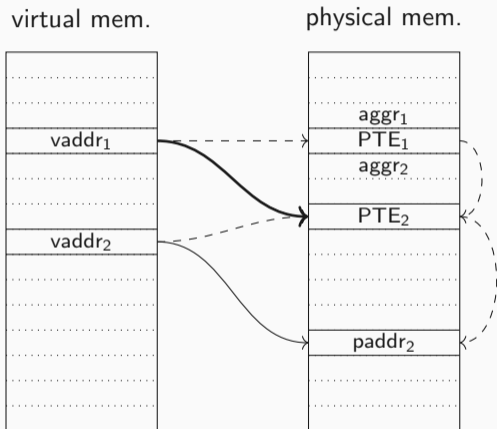


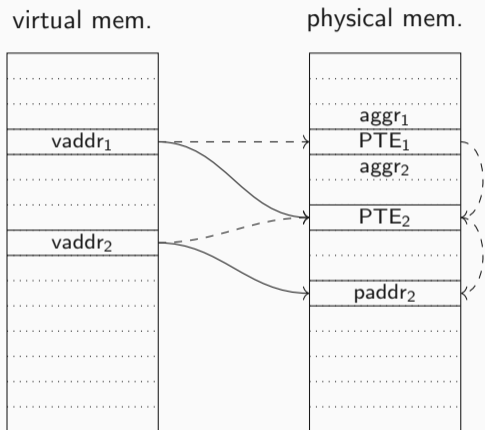


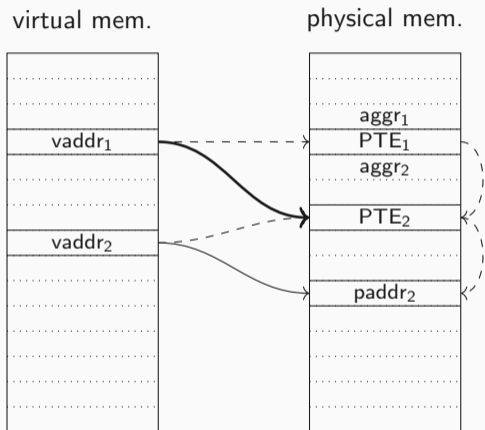


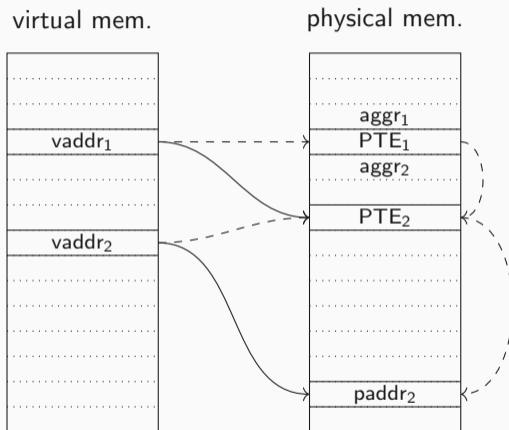


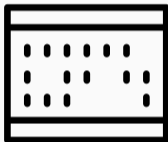


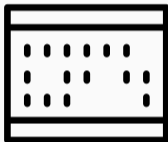




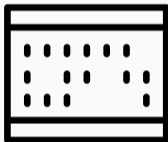


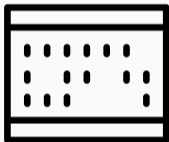




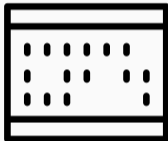


- Idea from [27]

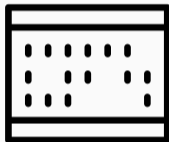




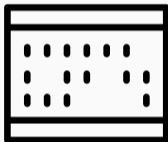
- Idea from [27]
- Same idea applied in several other works:



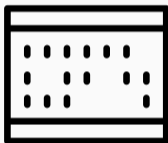
- Idea from [27]
- Same idea applied in several other works:
 - Rowhammer.js [9]



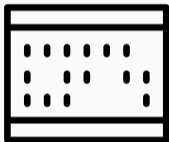
- Idea from [27]
- Same idea applied in several other works:
 - Rowhammer.js [9]
 - One bit flips, one cloud flops [31]



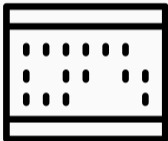
- Idea from [27]
- Same idea applied in several other works:
 - Rowhammer.js [9]
 - One bit flips, one cloud flops [31]
 - Drammer [29]



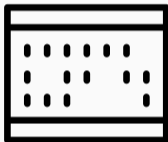
- Idea from [27]
- Same idea applied in several other works:
 - Rowhammer.js [9]
 - One bit flips, one cloud flops [31]
 - Drammer [29]
 - ECCploit [5]



- Idea from [27]
- Same idea applied in several other works:
 - Rowhammer.js [9]
 - One bit flips, one cloud flops [31]
 - Drammer [29]
 - ECCploit [5]
 - Half-Double Rowhammer [18]



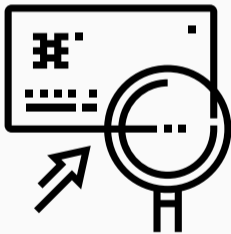
- Idea from [27]
- Same idea applied in several other works:
 - Rowhammer.js [9]
 - One bit flips, one cloud flops [31]
 - Drammer [29]
 - ECCploit [5]
 - Half-Double Rowhammer [18]
 - Presshammer [16]



- Idea from [27]
- Same idea applied in several other works:
 - Rowhammer.js [9]
 - One bit flips, one cloud flops [31]
 - Drammer [29]
 - ECCploit [5]
 - Half-Double Rowhammer [18]
 - Presshammer [16]
 - ...

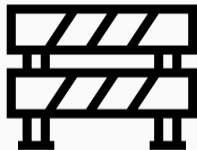
How to mitigate Rowhammer?

Different mitigations have been proposed:



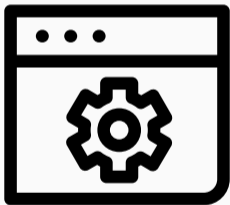
Detection

vs



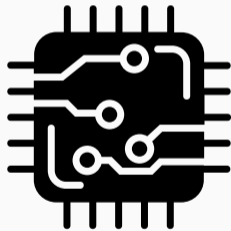
Prevention

Different mitigations have been proposed:



Software

vs



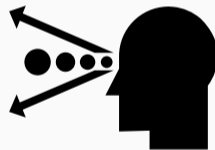
Hardware

Different mitigations have been proposed:

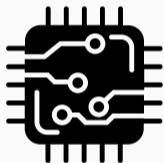


Short Term

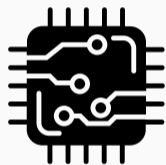
vs

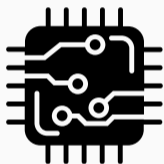


Long Term



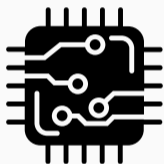
Original ideas from [17]





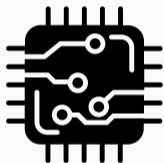
Original ideas from [17]

- Making better DRAM chips that are not vulnerable



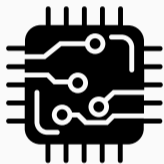
Original ideas from [17]

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



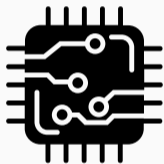
Original ideas from [17]

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



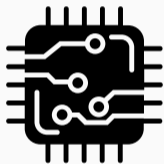
Original ideas from [17]

- 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 [17]

- 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 [17]

- 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

- No `clflush` instruction

× ×



- No `clflush` instruction →
Eviction

✘ ✘

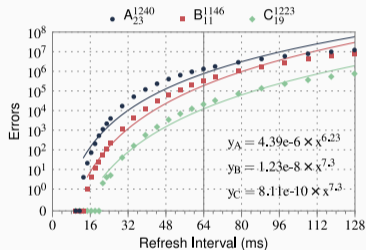


✘ ✘

- No `clflush` instruction → Eviction
- Increase the refresh rate



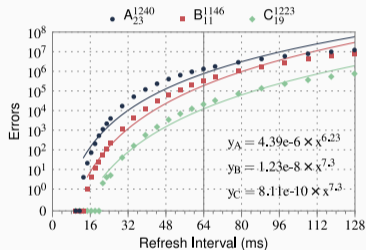
- No `clflush` instruction → Eviction
- Increase the refresh rate
 - Would need to be **increased by 7×** to eliminate all bit flips



Errors depending on refresh interval [17]



- No `clflush` instruction → Eviction
- 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 [17]



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



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



- ECC protection: server can handle or correct single bit errors
- **No standard** for event reporting
- In practice [19]
 - Common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)



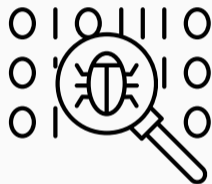
- ECC protection: server can handle or correct single bit errors
- **No standard** for event reporting
- In practice [19]
 - Common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
 - Some server vendors **never report errors** to the OS



- ECC protection: server can handle or correct single bit errors
- **No standard** for event reporting
- In practice [19]
 - 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



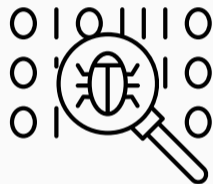
- ECC protection: server can handle or correct single bit errors
- **No standard** for event reporting
- In practice [19]
 - Common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
 - Some server vendors **never report errors** to the OS
 - One server **did not even halt** when bit flips were non-correctable
- ECCploit actively targeting ECC DRAM with Rowhammer [5]



MASCAT - Stopping Microarchitectural Attacks Before Execution

[12]

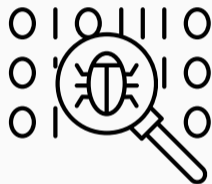
- Static analysis of the binary



MASCAT - Stopping Microarchitectural Attacks Before Execution

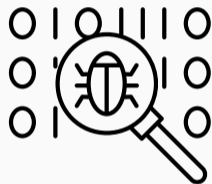
[12]

- Static analysis of the binary
- Detect suspicious instruction sequences
(`clflush`, `rdtsc`, `fences`, ...)



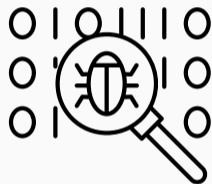
MASCAT - Stopping Microarchitectural Attacks Before Execution [12]

- Static analysis of the binary
- Detect suspicious instruction sequences
(`clflush`, `rdtsc`, `fences`, ...)
- Open problems



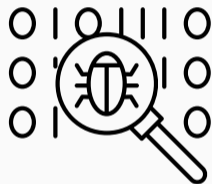
MASCAT - Stopping Microarchitectural Attacks Before Execution [12]

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



MASCAT - Stopping Microarchitectural Attacks Before Execution [12]

- Static analysis of the binary
- Detect suspicious instruction sequences (`clflush`, `rdtsc`, `fences`, ...)
- Open problems
 - False positives
 - Self modifying code

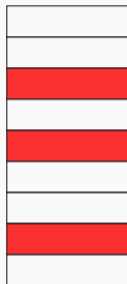


MASCAT - Stopping Microarchitectural Attacks Before Execution [12]

- Static analysis of the binary
- Detect suspicious instruction sequences (`clflush`, `rdtsc`, `fences`, ...)
- Open problems
 - False positives
 - Self modifying code
 - ThrowHammer [28], NetHammer [20]

- B-CATT: disable vulnerable physical memory [2]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [2]

B-CATT

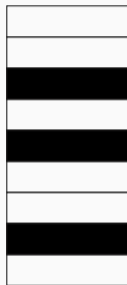


G-CATT



- B-CATT: disable vulnerable physical memory [2]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [2]

B-CATT

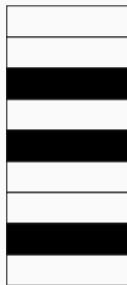


G-CATT



- B-CATT: disable vulnerable physical memory [2]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [2]

B-CATT



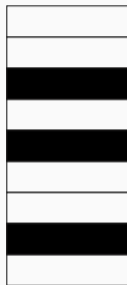
G-CATT



- B-CATT: disable vulnerable physical memory [2]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [2]

B-CATT: Might block 95% of RAM [8, 30]

B-CATT

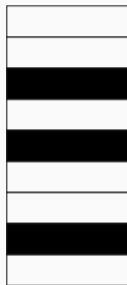


G-CATT



- B-CATT: disable vulnerable physical memory [2]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [2]

B-CATT



G-CATT

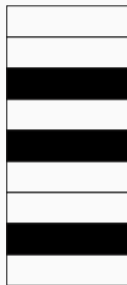


B-CATT: Might block 95% of RAM [8, 30]

G-CATT: What about non-kernel or shared pages? [3, 8]

- B-CATT: disable vulnerable physical memory [2]
- G-CATT: isolate security domains in physical memory based on potential vulnerability [2]

B-CATT



G-CATT

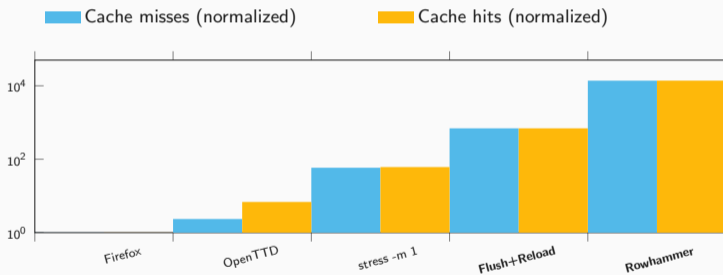


B-CATT: Might block 95% of RAM [8, 30]

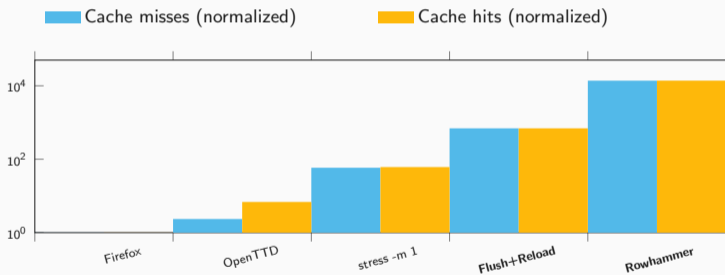
G-CATT: What about non-kernel or shared pages? [3, 8]

G-CATT: Bit flips more than 8 “rows” apart [8, 17], Half-Double Rowhammer [18]

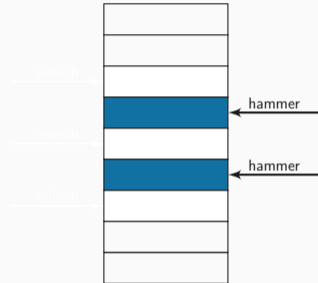
- Rowhammer: lots of **cache misses** that can be monitored with **hardware performance counters** [4, 10, 11, 22]



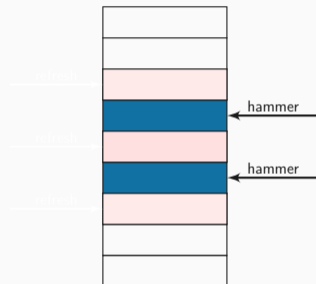
- Rowhammer: lots of **cache misses** that can be monitored with **hardware performance counters** [4, 10, 11, 22]



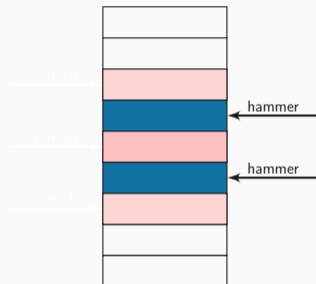
What if performance counters do not work because we run in SGX? [8, 13]



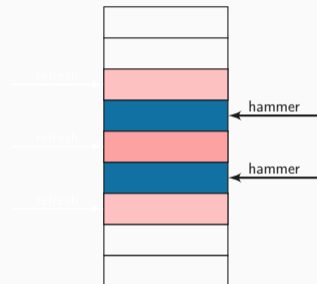
- Counter per row



- Counter per row
- Increment neighbor rows



- Counter per row
- Increment neighbor rows



- Counter per row
- Increment neighbor rows



- Counter per row
- Increment neighbor rows



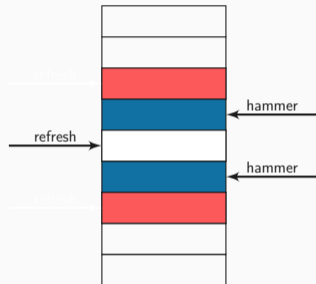
- Counter per row
- Increment neighbor rows



- Counter per row
- Increment neighbor rows



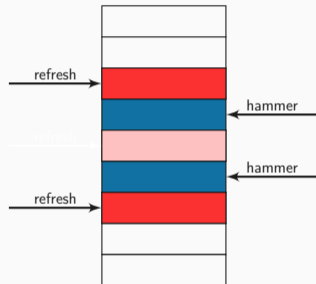
- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold



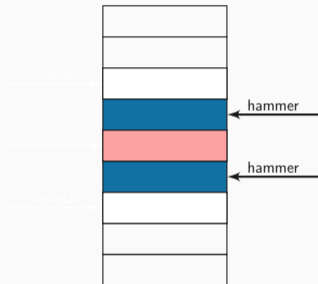
- Counter per row
- Increment neighbor rows
- Refresh when counter reaches a threshold



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



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



- Counter per row too expensive

- Counter per row too expensive
- Small number of counters

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

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

- Counter per row too expensive
- Small number of counters
- Can be tricked to count not hammered rows
- Complex many sided access patterns
- TRRespass [7, 25]

- Counter per row too expensive
- Small number of counters
- Can be tricked to count not hammered rows
- Complex many sided access patterns
- TRRespass [7, 25]
- Blacksmith [14]

- Counter per row too expensive
- Small number of counters
- Can be tricked to count not hammered rows
- Complex many sided access patterns
- TRRespass [7, 25]
- Blacksmith [14]
- Getting pretty good in DDR5 DRAM



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



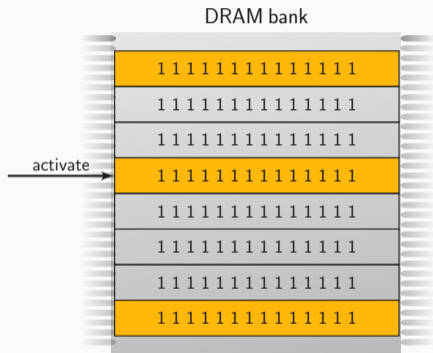
- There are two different hammering techniques

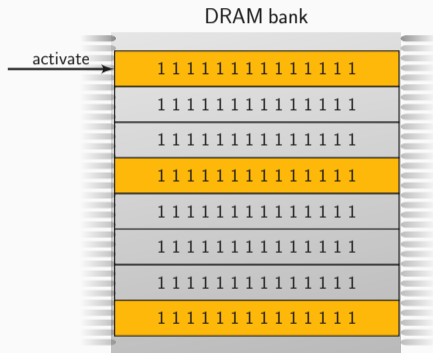


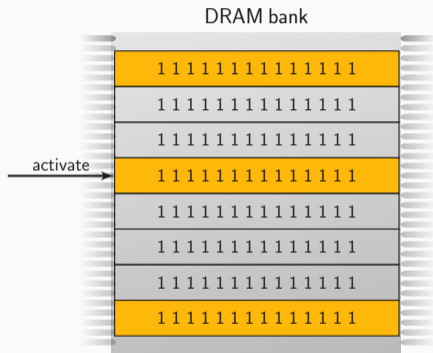
- There are two different hammering techniques
- #1: Hammer one row next to victim row and other random rows

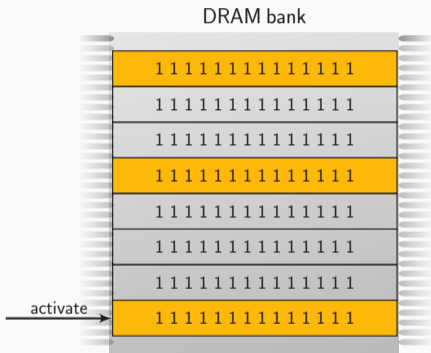


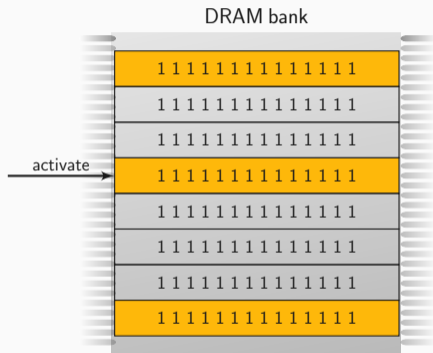
- There are two different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row

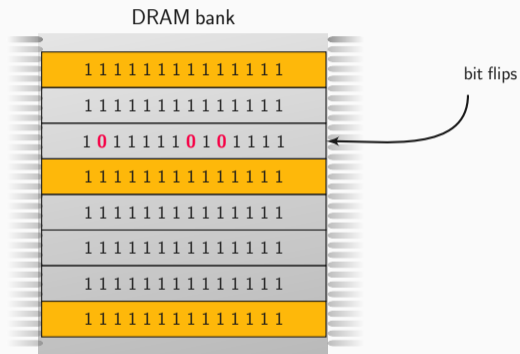


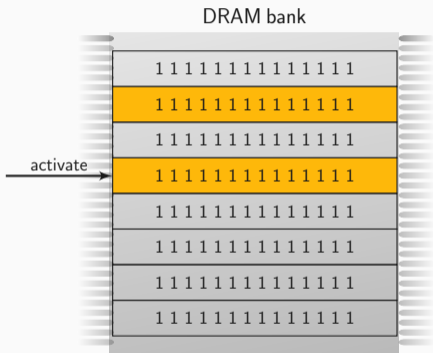


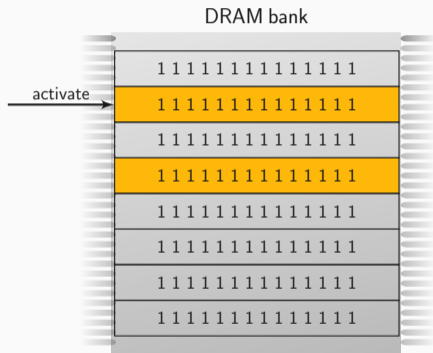


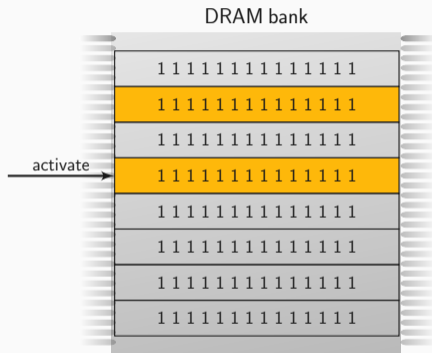


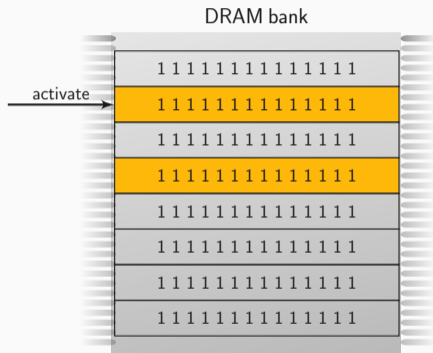


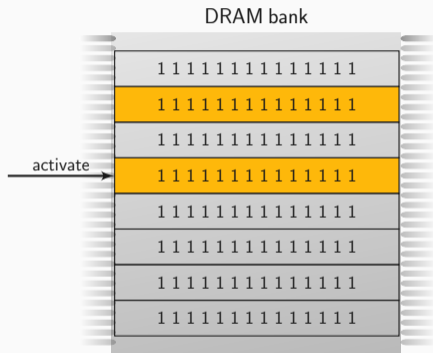


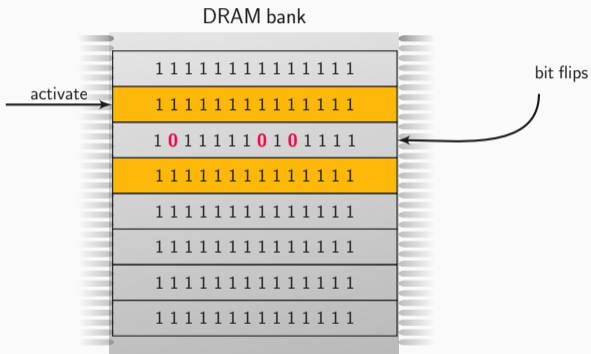














**HAMMERING
TWO ROWS**



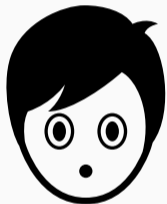
**HAMMERING
TWO ROWS**



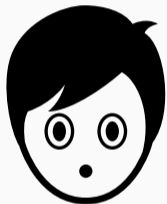
**HAMMERING
A SINGLE ROW**



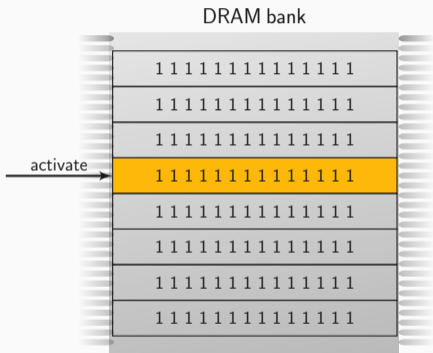
- There are two different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row

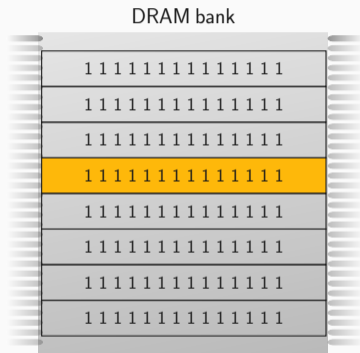


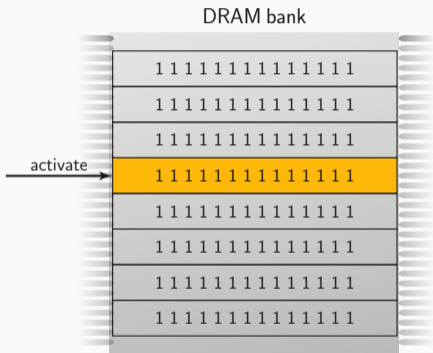
- There are **three** different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row

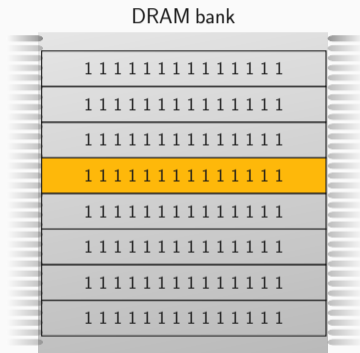


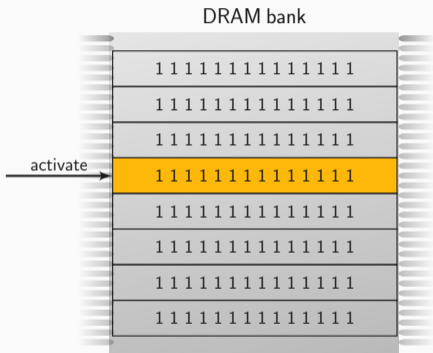
- There are **three** different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row
- **#3: Hammer only one row next to victim row**

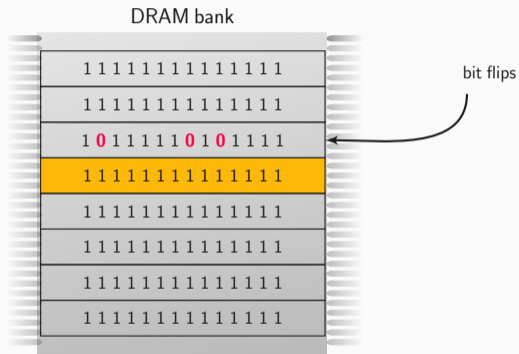












File Edit View Search Terminal Help

```
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:

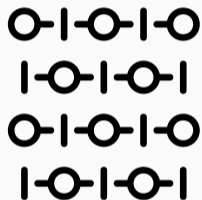
- Memory-controller page policies

Two underlying reasons:

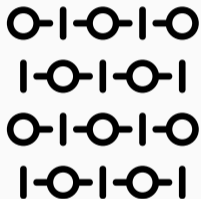
- Memory-controller page policies
- Physical effect called RAS-clobber

Two underlying reasons:

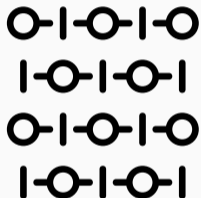
- Memory-controller page policies
- Physical effect called RAS-clobber
- Recently shown in Rowpress paper [21]



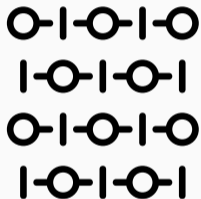
- **Open-page policy:** Keep row opened and buffered



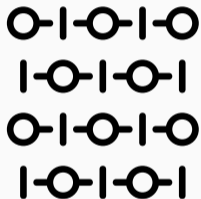
- **Open-page policy:** Keep row opened and buffered
 - Low latency for subsequent accesses to same row
 - High latency for accesses to any other row



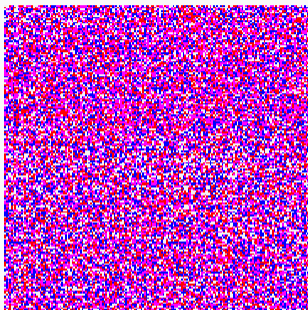
- **Open-page policy:** Keep row opened and buffered
 - Low latency for subsequent accesses to same row
 - High latency for accesses to any other row
- **Closed-page policy:** Immediately close row, ready to open a new row



- **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



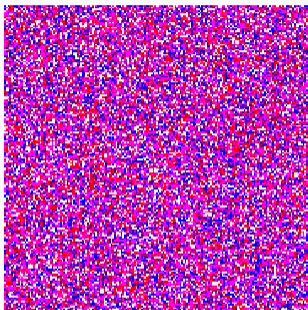
- **Open-page policy:** Keep row opened and buffered
 - Low latency for subsequent accesses to same row
 - High latency for accesses to any other row
- **Closed-page policy:** Immediately close row, ready to open a new row
 - Medium latency for accesses to any row
 - Perform better on multi-core systems [6]



Double-sided

77.0 % bit offsets

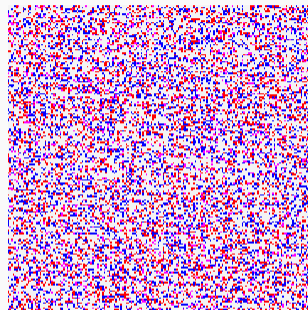
51.7 % 0→1 bit flips



Single-sided

78.5 % bit offsets

54.1 % 0→1 bit flips



One-location

36.5 % bit offsets

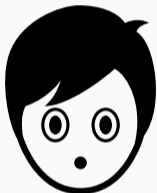
51.6 % 0→1 bit flips



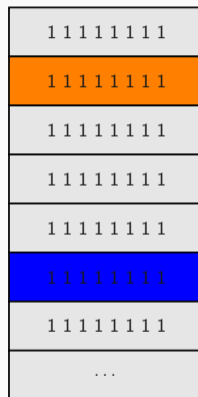
- There are three different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row
- #3: Hammer only one row next to victim row

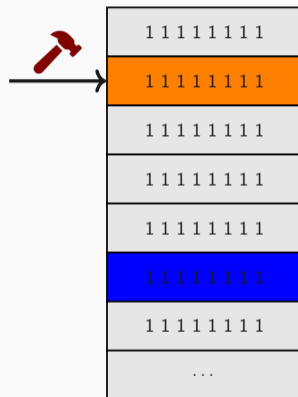


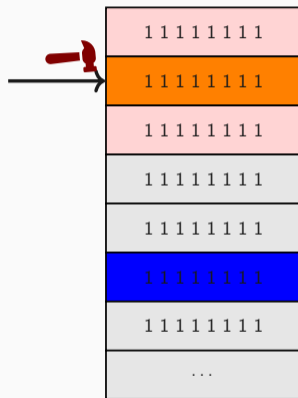
- There are **four** different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row
- #3: Hammer only one row next to victim row

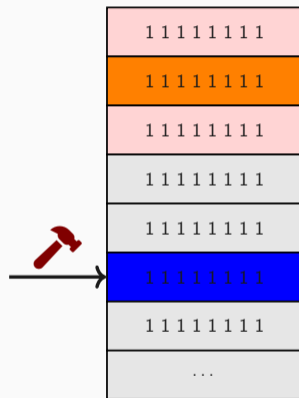


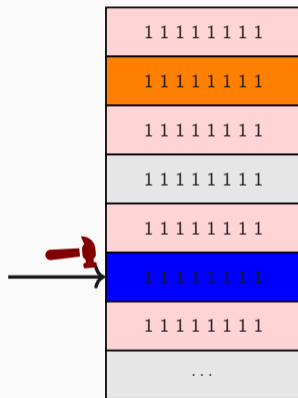
- There are **four** different hammering techniques
- #1: Hammer one row next to victim row and other random rows
- #2: Hammer two rows neighboring victim row
- #3: Hammer only one row next to victim row
- **#4: Hammer from a larger distance with the help of TRR [18]**

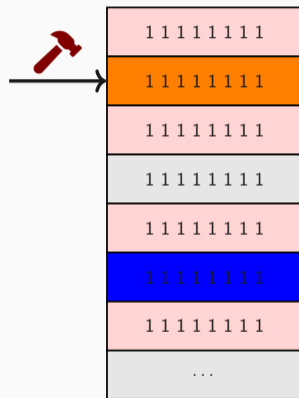


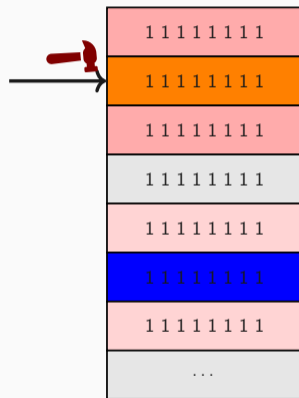


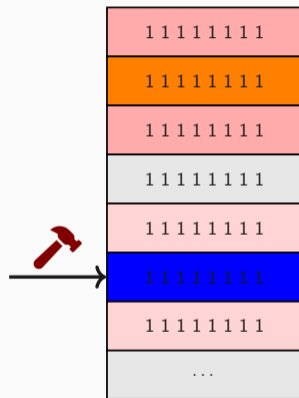


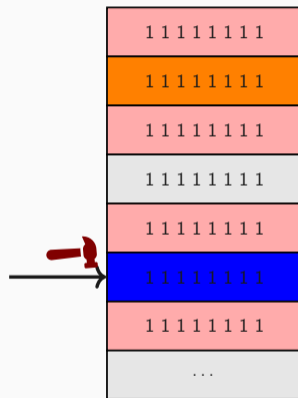


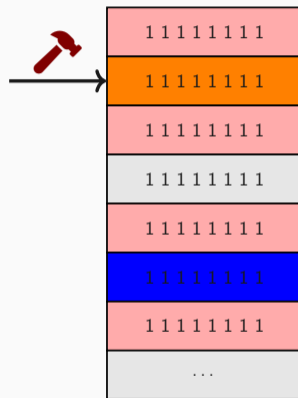


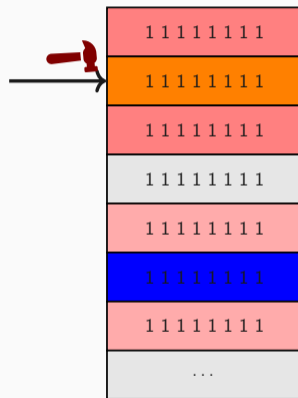


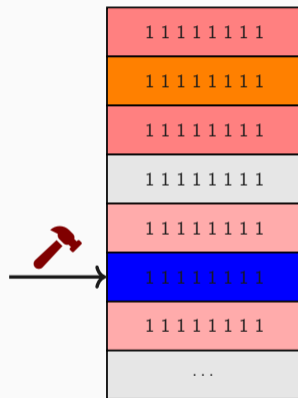


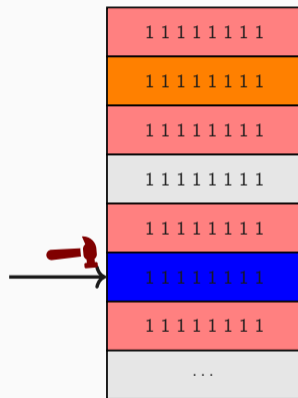


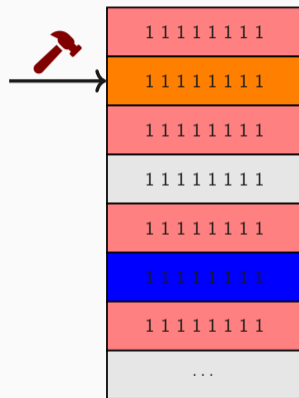


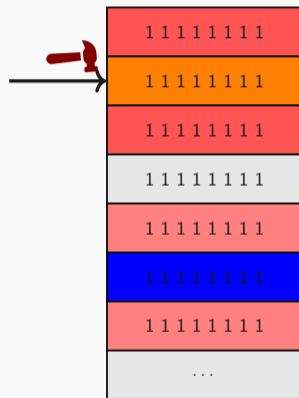


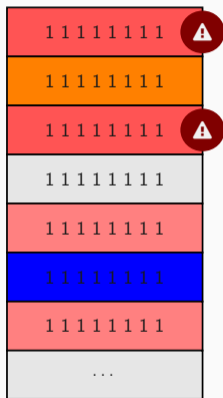


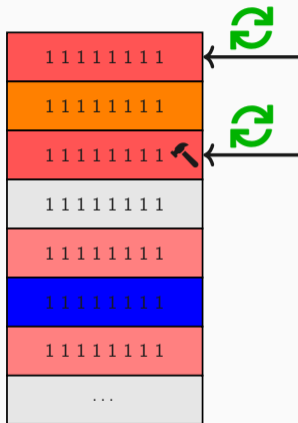


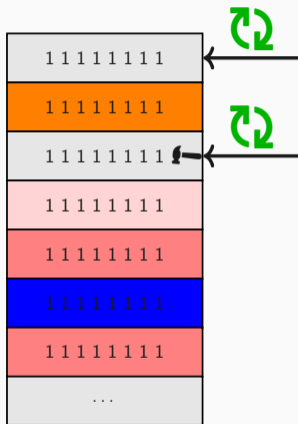


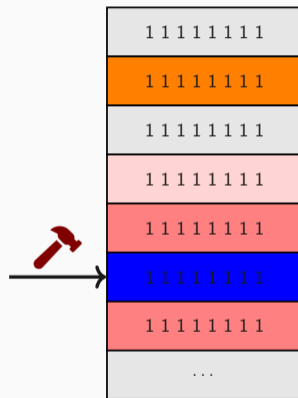


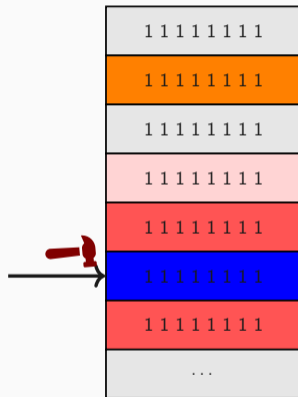


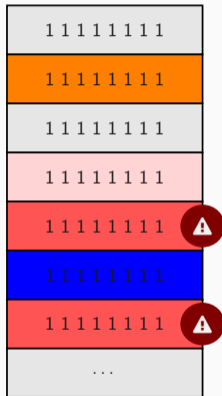


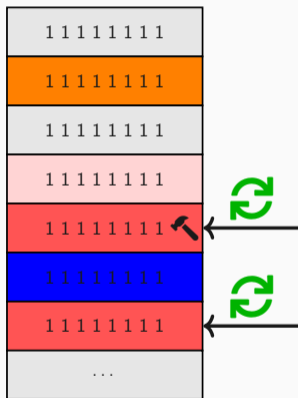


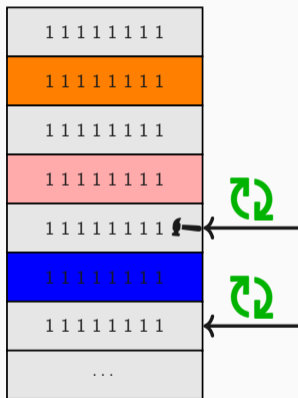


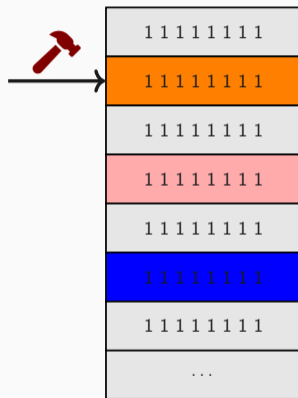


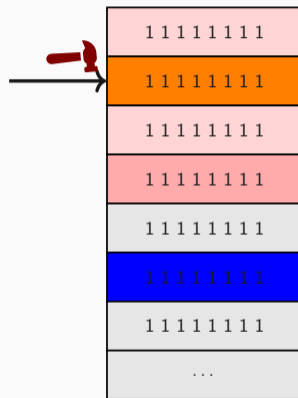


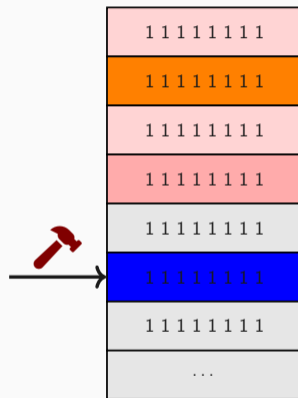


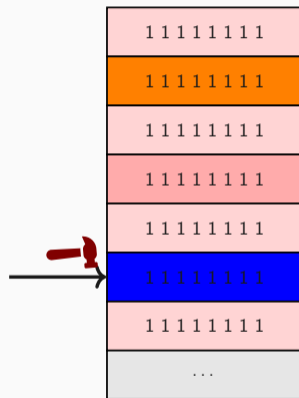


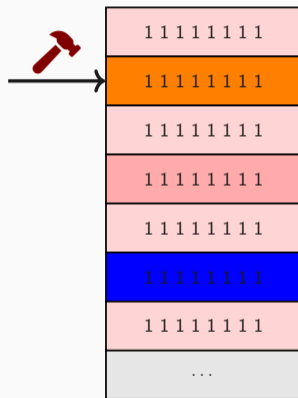


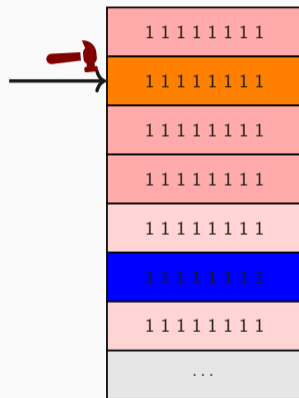


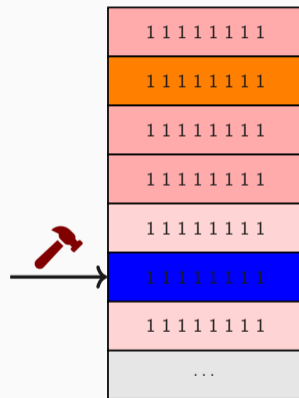


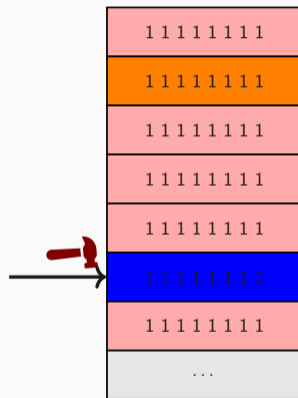


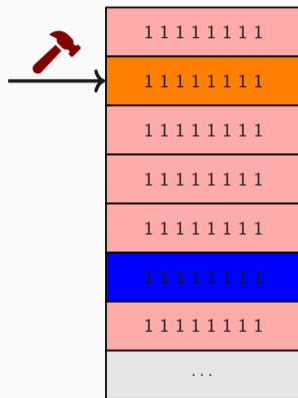


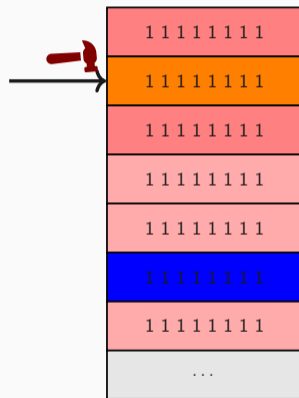


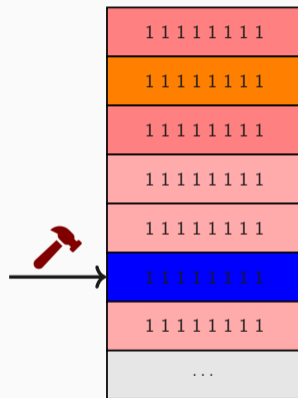


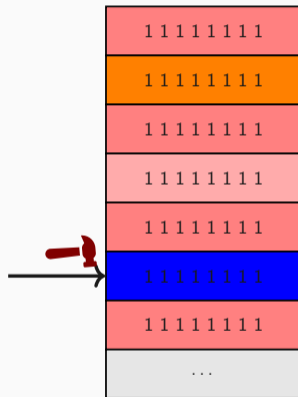


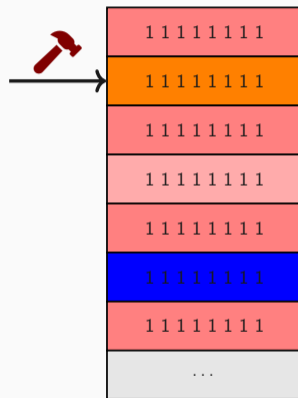


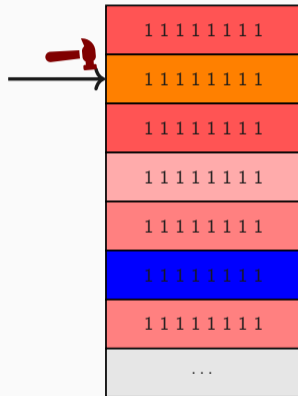


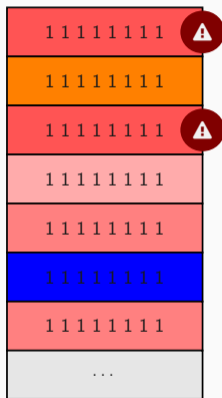


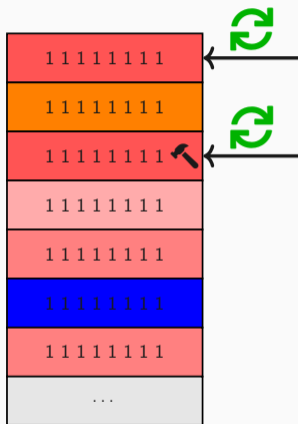


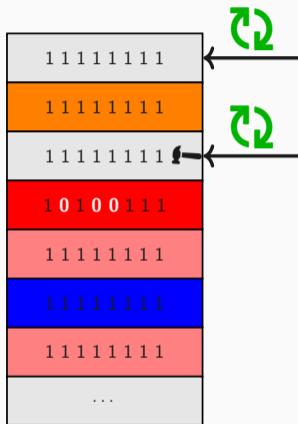


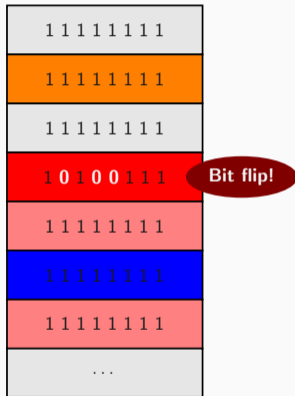














- TRR is a **confused deputy**



- 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



- 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



- 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)



















- Conditional jumps are not the only targets



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



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



- 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

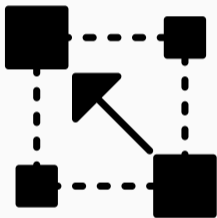


- Conditional jumps are not the only targets
- Other targets include
 - Comparisons
 - Addresses of memory loads/stores
 - Address calculations
 - ...
- Manual analysis of `sudo` revealed 29 possible bit flips
- They all somehow skipped the password check

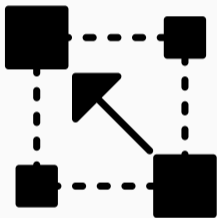
How to get the target virtual page to the target physical location?

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

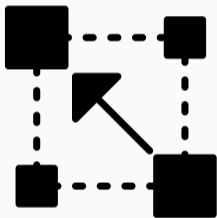
Memory Waylaying



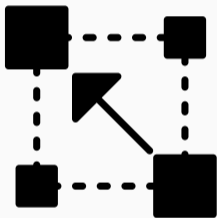
- Not as easy as with page tables



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



- 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



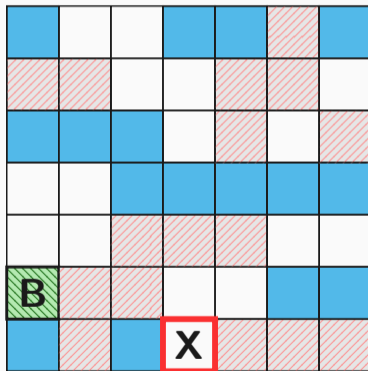
- Not as easy as with page tables
- Binary only once in memory + stays in memory (in the page cache) even after termination
- Only evicted if page cache is full
- Page cache usually occupies all unused memory



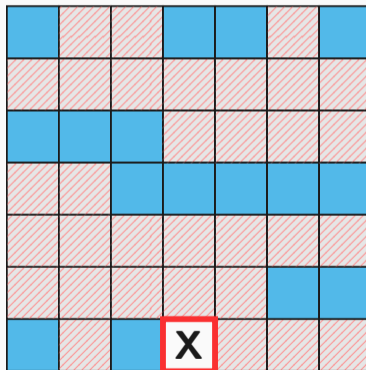
MEMORY WAYLAYING

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

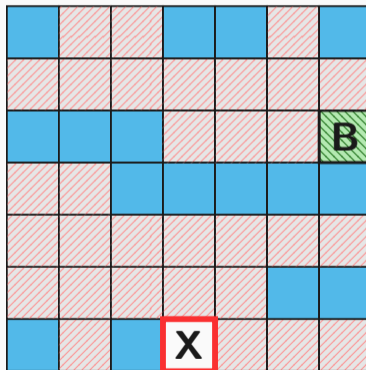
(1) Start



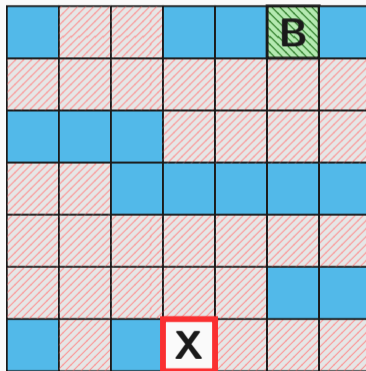
(2) Evict Page Cache



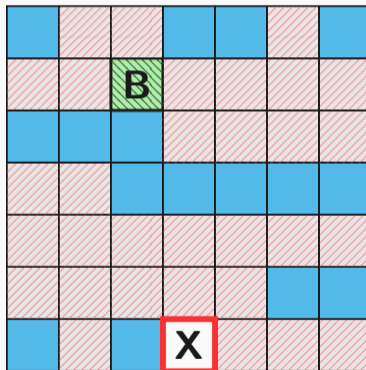
(3) Access Binary



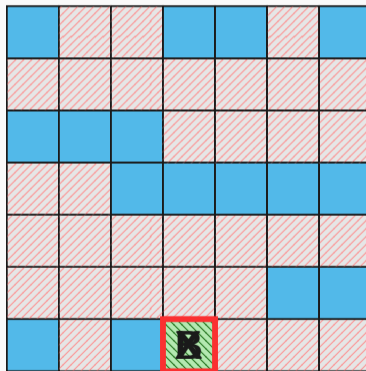
(4) Evict + Access



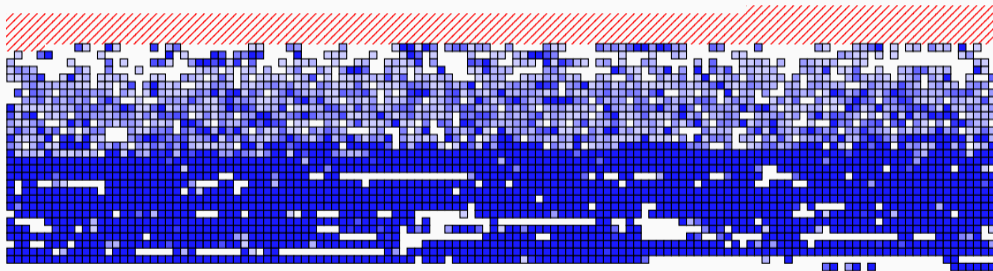
(5) Evict + Access



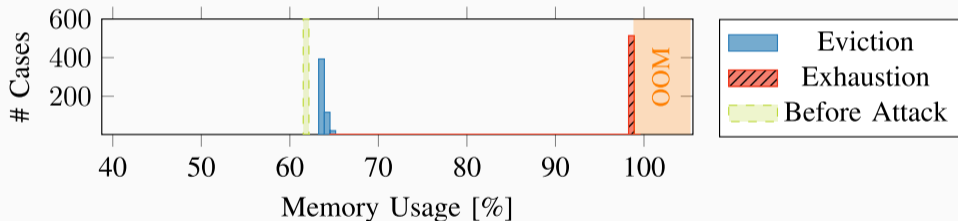
(6) Stop if target reached

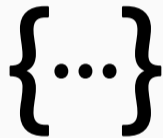


- New pages cover most of the physical memory

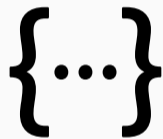


- Great advantage over memory massaging: only negligible memory footprint





- Many (academic) countermeasures were proposed to mitigate Rowhammer



- Many (academic) countermeasures were proposed to mitigate Rowhammer
- All of them can be circumvented [8, 14, 18]



- Many (academic) countermeasures were proposed to mitigate Rowhammer
- All of them can be circumvented [8, 14, 18]
- We cannot design countermeasures focused on specifics of the attack



- Many (academic) countermeasures were proposed to mitigate Rowhammer
- All of them can be circumvented [8, 14, 18]
- We cannot design countermeasures focused on specifics of the attack
- Otherwise we only patch concrete exploits, but do not solve the problem

Apple had a great idea:

- Lower refresh rate → save energy + more flips





Apple had a great idea:

- Lower refresh rate → save energy + more flips
- ECC memory → fewer flips



Apple had a great idea:

- Lower refresh rate → save energy + more flips
- ECC memory → fewer flips
- It's an optimization problem.



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



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?



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?



Apple had a great idea:

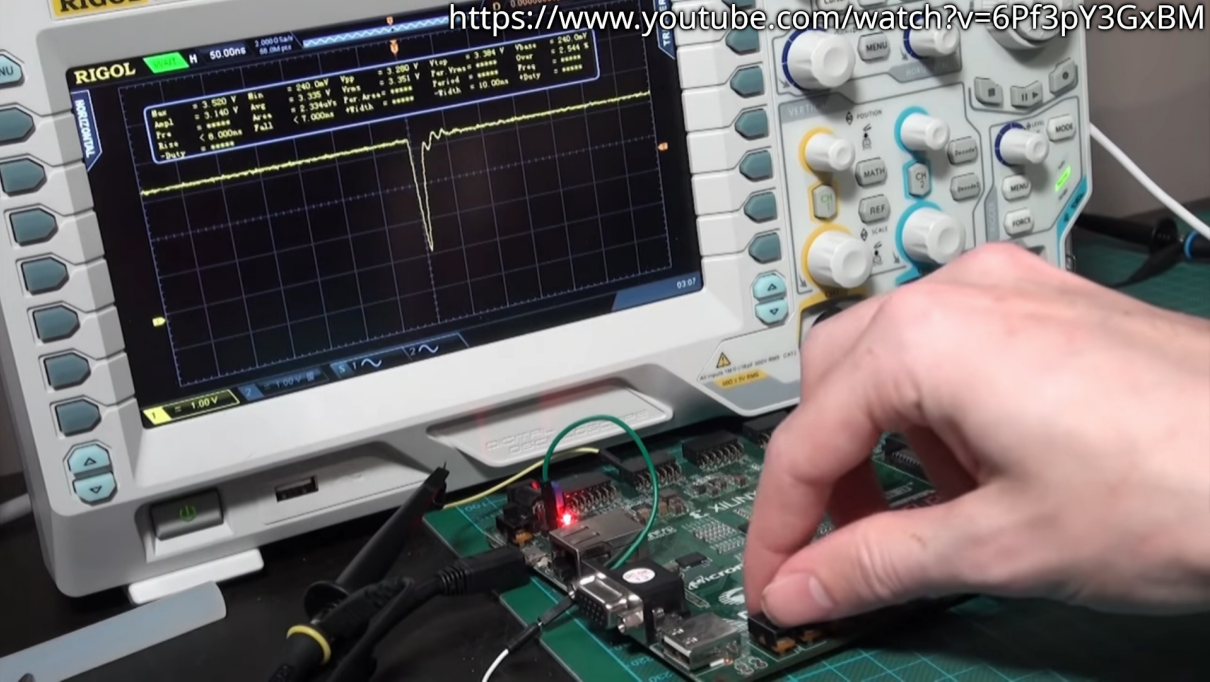
- Lower refresh rate → save energy + more flips
- ECC memory → fewer flips
- It's an optimization problem.
 - Too aggressive? → bit flips
 - Too cautious? → waste of energy
 - What if the “too aggressive” changes over time?
 - What if attackers come up with slightly better attacks?
- Difficult to optimize with an adversary working against you

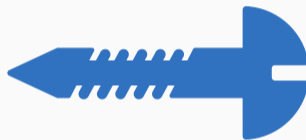


- Data integrity protection for **all data** in the DRAM [15]
- Compute Cryptographic MAC on all data → perfect integrity protection
- **No** focus on Rowhammer
- No new hammer method or exploit can circumvent it
- Becoming reality with integrity protected memory encryption

Undervolting Attacks







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

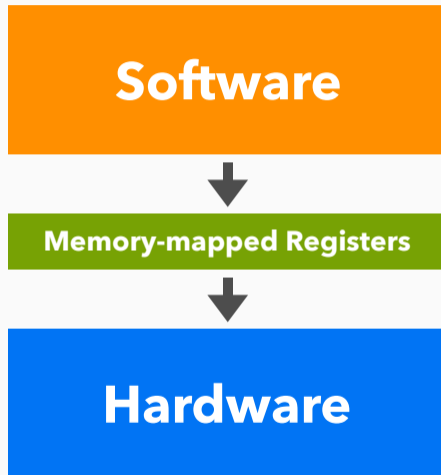
- Gamers want fast responses
- Cloud servers want high-assurance and low running costs

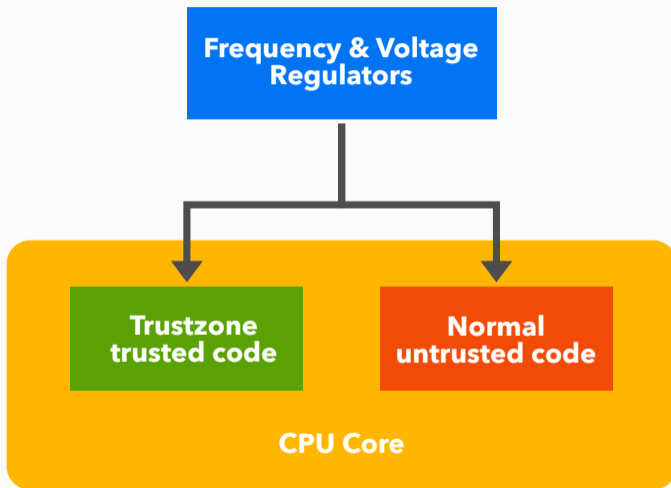
Changing the voltage and frequency of the CPU

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

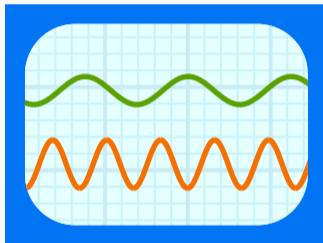
Changing the voltage and frequency of the CPU

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





```
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

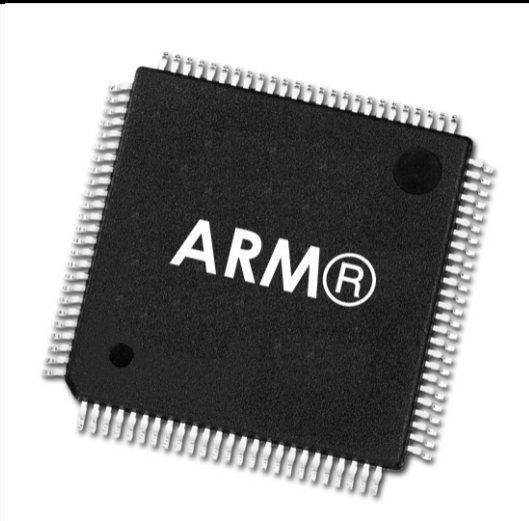


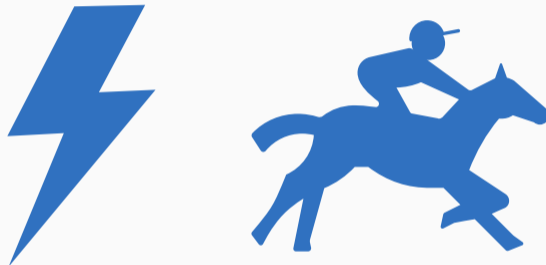


- Infer secret AES key that was stored within Trustzone

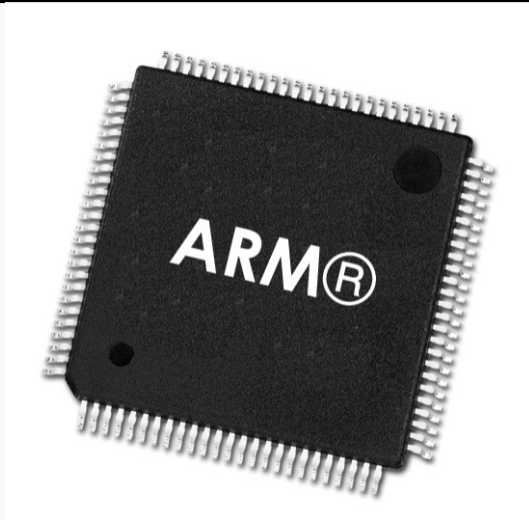


- Infer secret AES key that was stored within Trustzone
- Trick Trustzone into loading a self-signed app

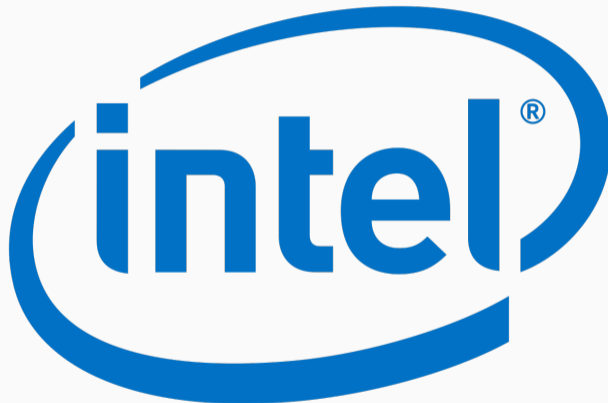




Qiu2019



**ARE WE ALL
THINKING THE
SAME THING?**



System Information Core

Manual Tuning

- All Controls
- Core
- Graphics
- Stress Test
- Profiles

Reference Clock 103.2258 MHz Max Non-Turbo Boost Ratio 34 x

Turbo Boost Short Power Max Enable Turbo Boost Short Power Max 1200,000 W

Disable Enable

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

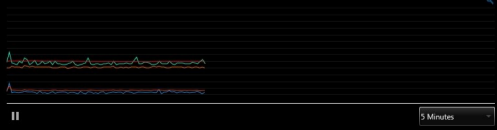
Graphics

Processor Graphics Current Limit Limits the maximum ratio that the processor can use while four cores are active. 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		
Processor Graphics Current L...	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



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				

intel Intel® Extreme Tuning Utility

Before

System Information Benchmark your Current Settings

Run XTU Benchmark

Current Score

XTU: 1921 Marks

Maximum Processor Frequency: 4.15 GHz

Highest CPU Temperature: 96 °C

Stress Test

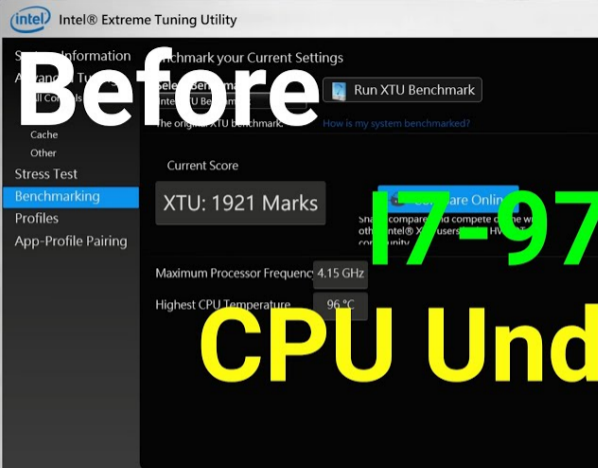
Benchmarking

Profiles

App-Profile Pairing

Compare Online

Share, compare, and compete online with other Intel® XTU users in the HWBOT community.



intel Intel® Extreme Tuning Utility

After

System Information Benchmark your Current Settings

Run XTU Benchmark

Current Score

XTU: 2122 Marks

Maximum Processor Frequency: 4.13 GHz

Highest CPU Temperature: 95 °C

Stress Test

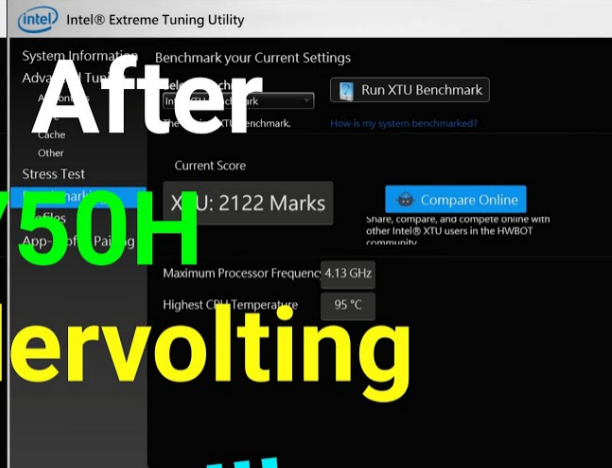
Benchmarking

Profiles

App-Profile Pairing

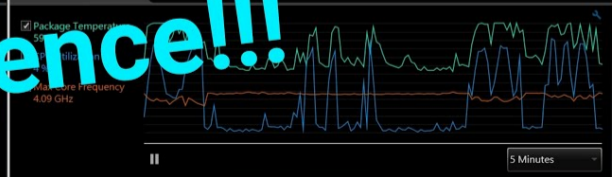
Compare Online

Share, compare, and compete online with other Intel® XTU users in the HWBOT community.



I7-9750H

CPU Undervolting

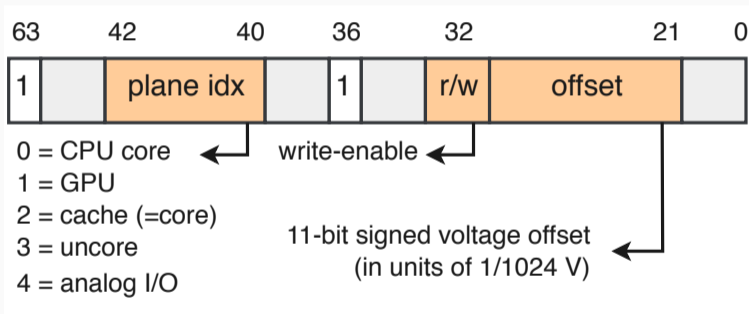


Huge difference!!!





msr 0x150

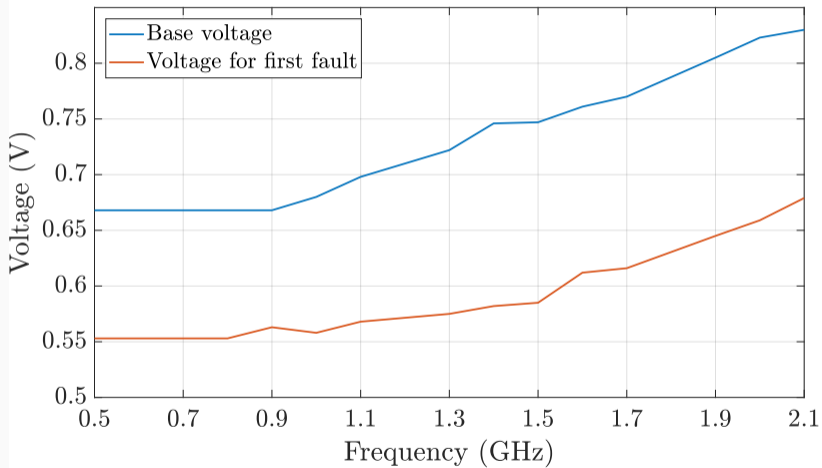


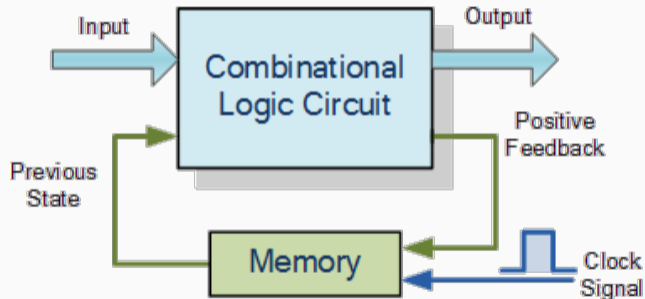
```
uint64_t multiplier = 0x1122334455667788;
uint64_t correct    = 0xdeadbeef * multiplier;
uint64_t var        = 0xdeadbeef * multiplier;

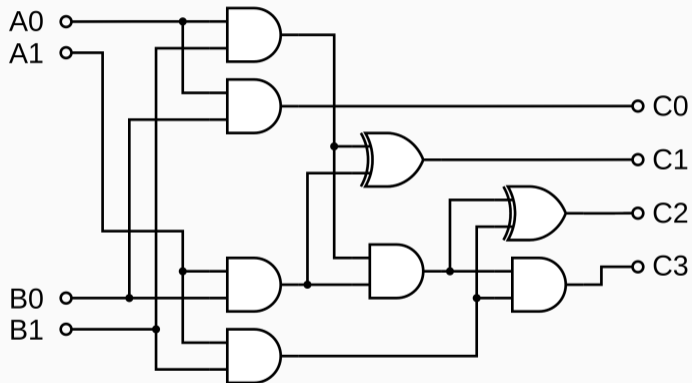
while (var == correct)
{
    var = 0xdeadbeef * multiplier;
}
uint64_t flipped_bits = var ^ correct;
```

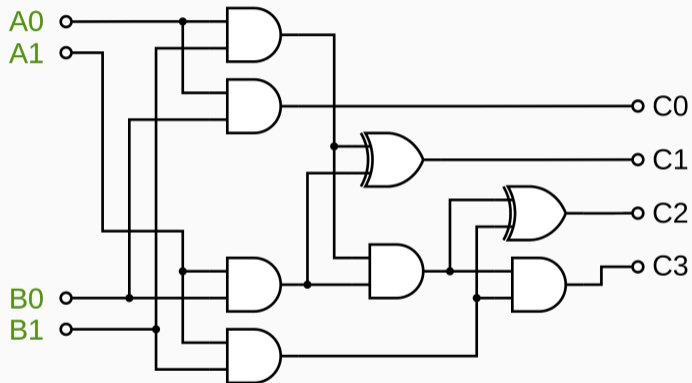
bagger> █

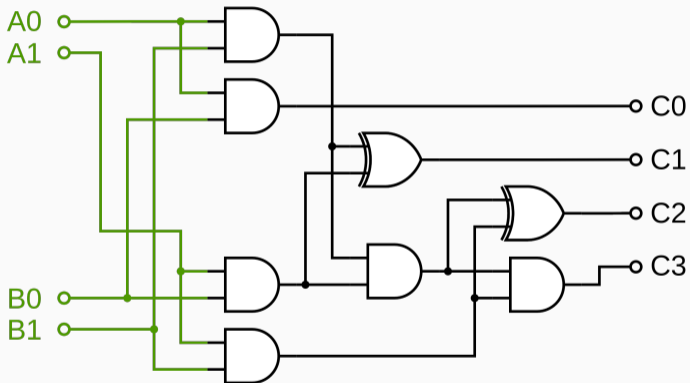
I

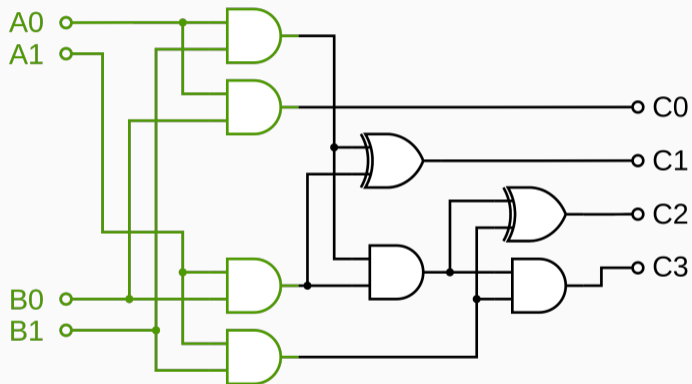


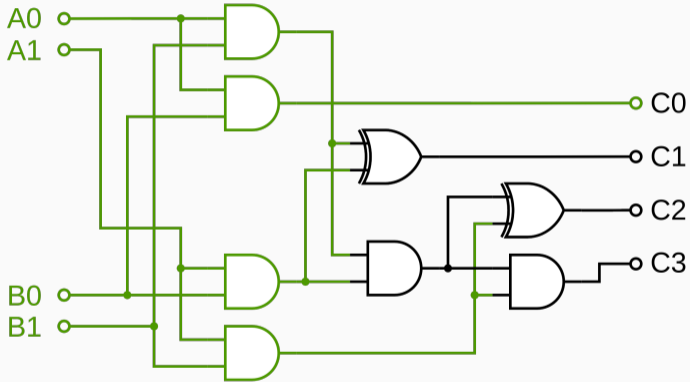


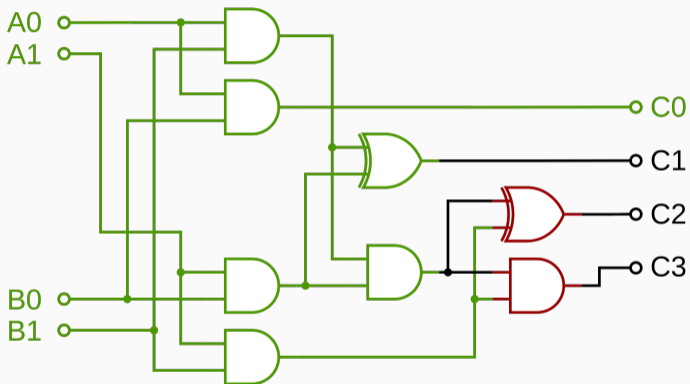


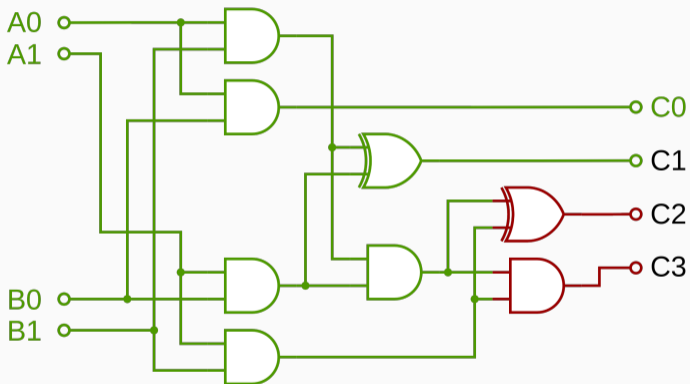


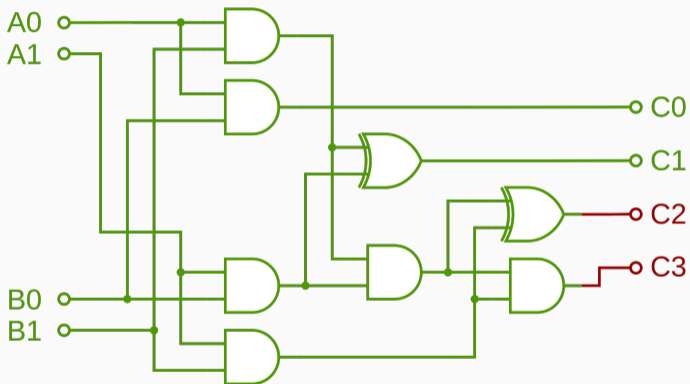


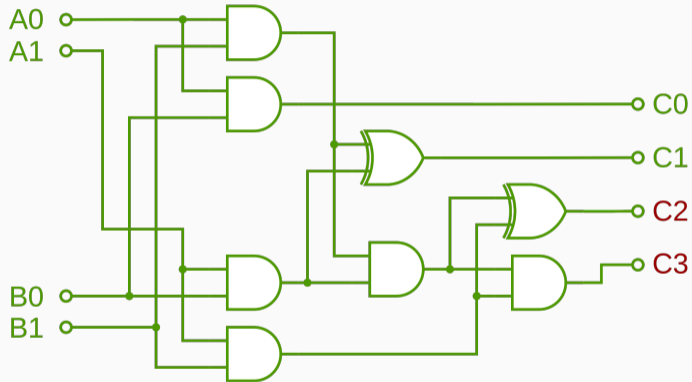


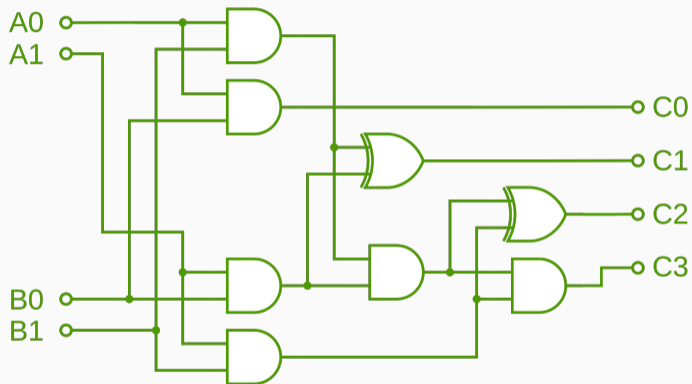


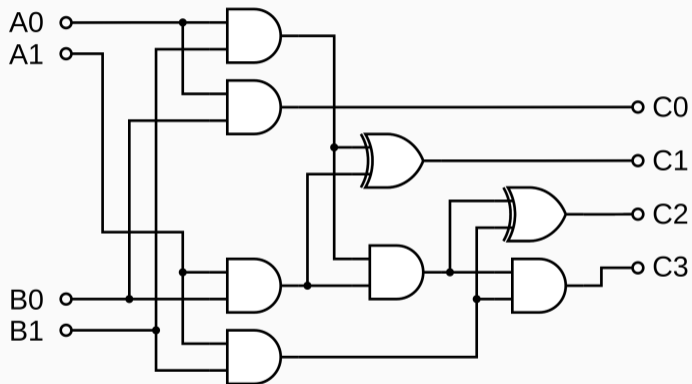


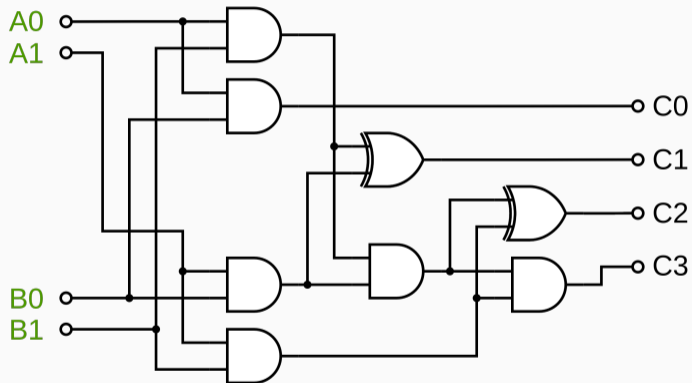


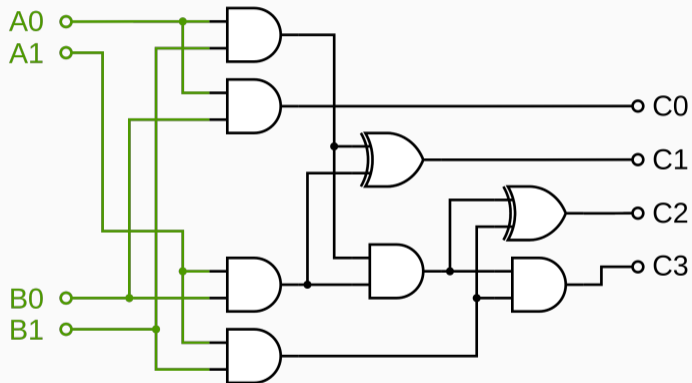


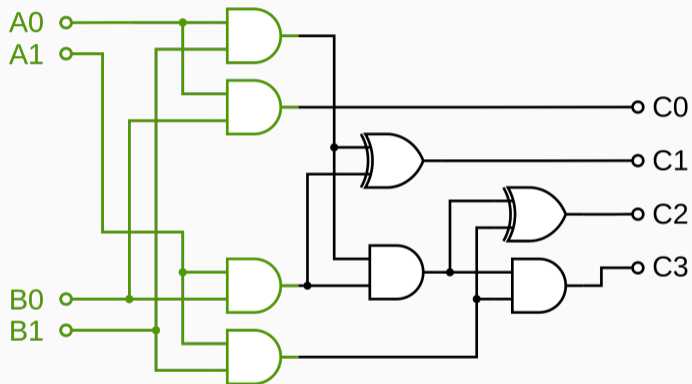


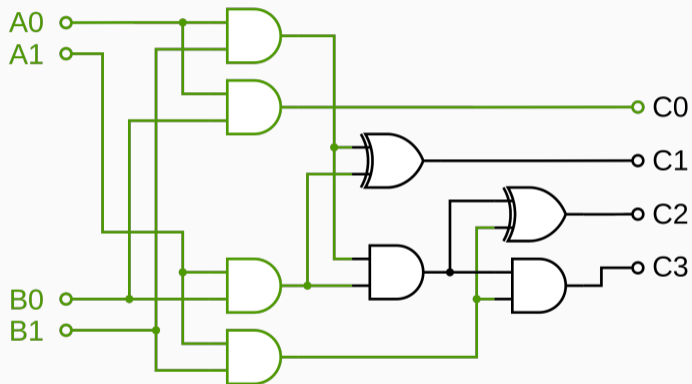


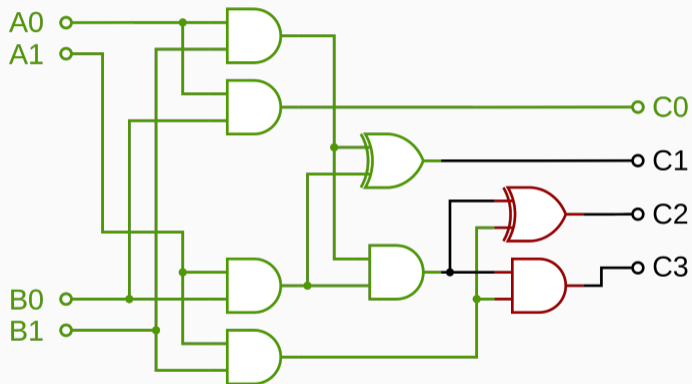


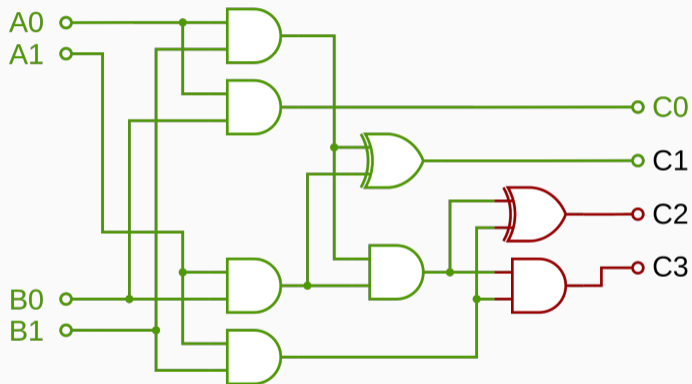


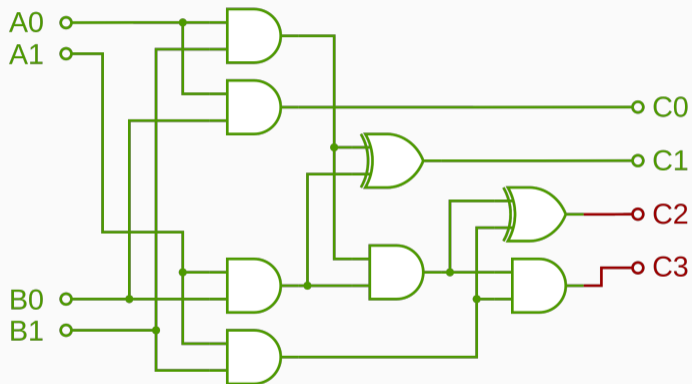


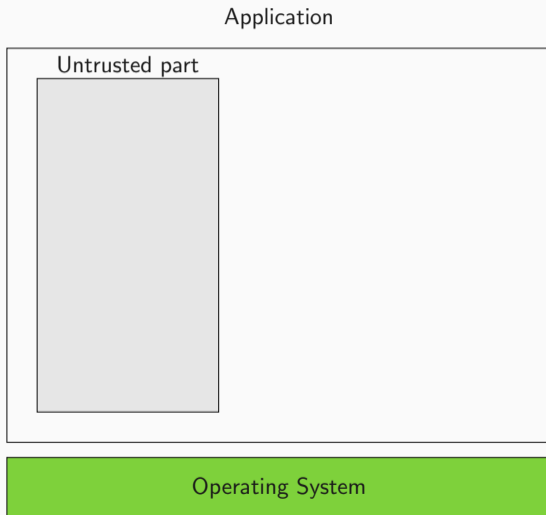


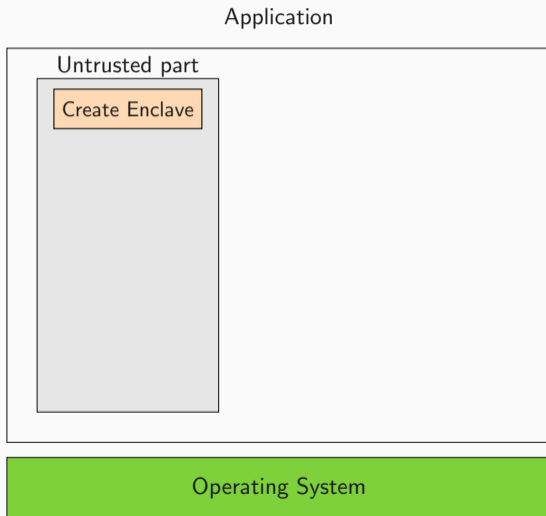


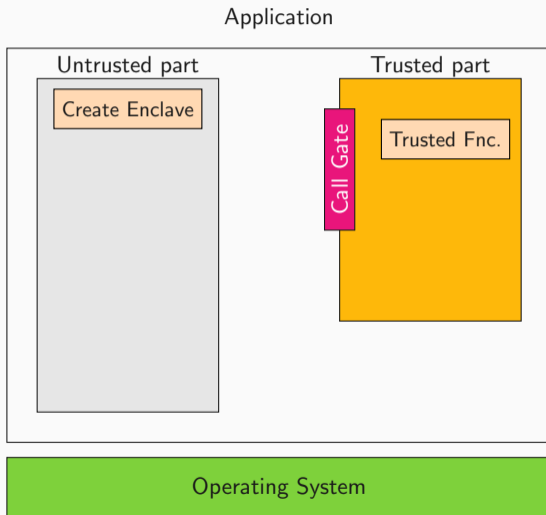


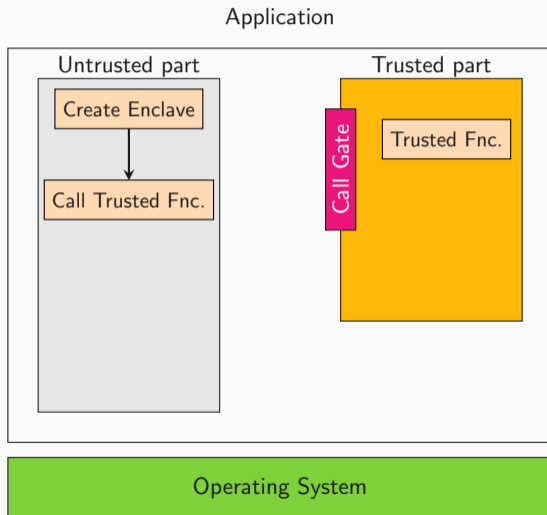


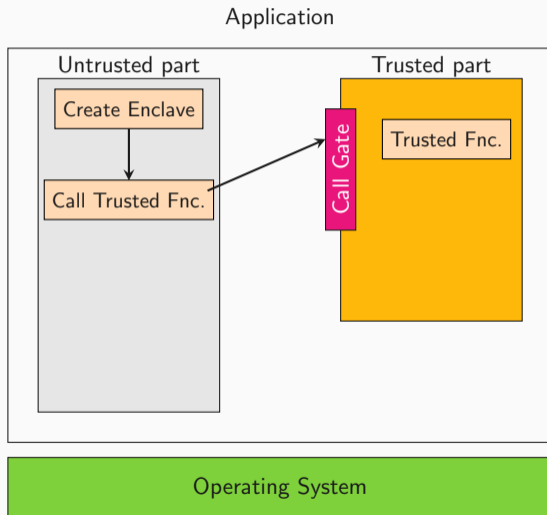


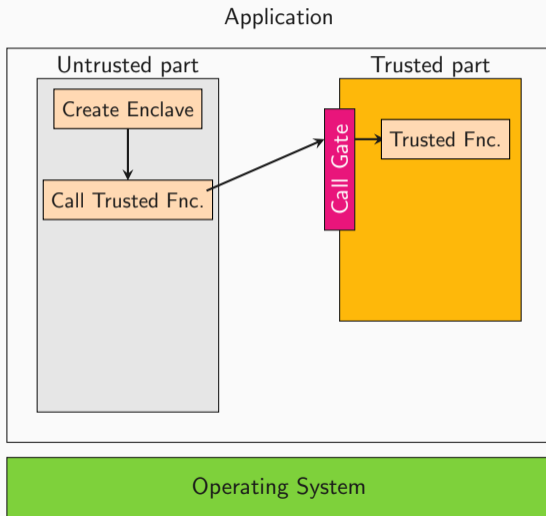


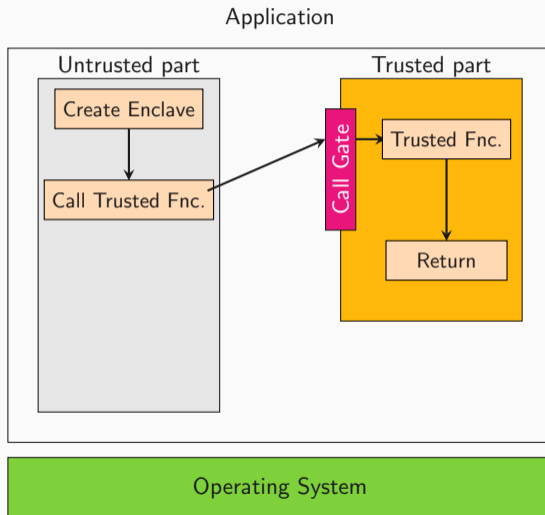


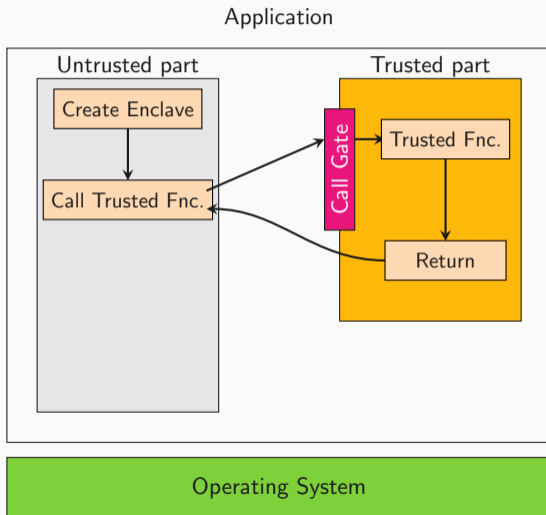


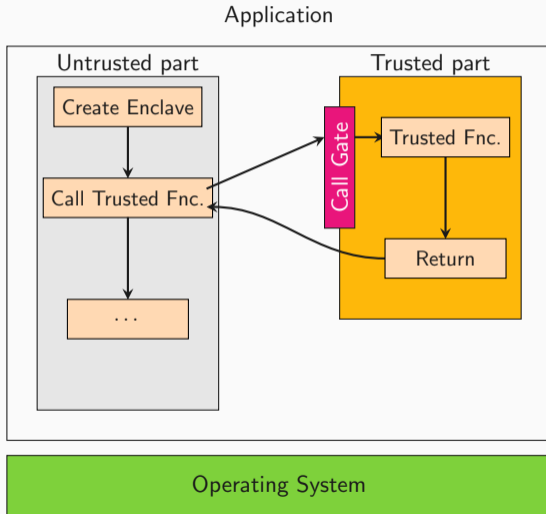


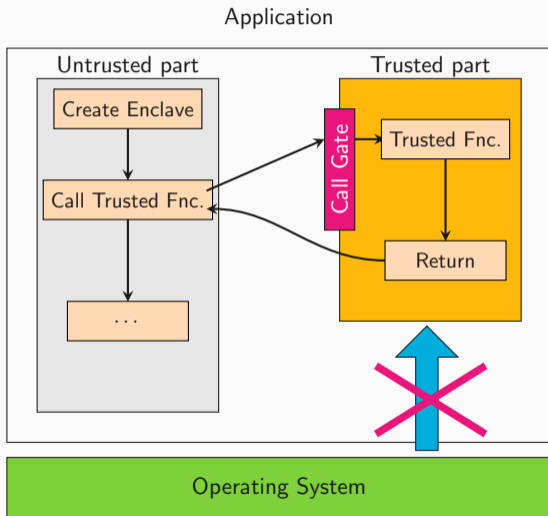






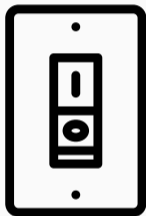




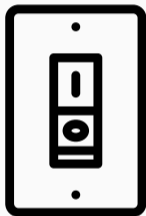




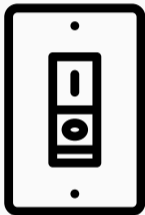




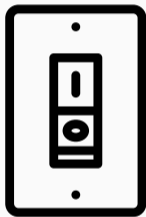
- Bit flips in the EPC?



- 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

$$n = p \times q$$

$$S = H^d \text{ mod } n$$

$$n = p \times q$$

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

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

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

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

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

$$n = p \times q$$

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

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

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

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

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

$$n = p \times q$$

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

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

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

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

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

$$n = p \times q$$

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

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

$$n = p \times q$$

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

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

$$n = p \times q$$

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

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

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

$$n = p \times q$$

$$S = \left(\overbrace{\hspace{10em}}^{\text{something}} \right) \times q + s_2$$

$$S' = \left(\overbrace{\hspace{10em}}^{\text{something else}} \right) \times q + s_2$$

$$S - S' = \hspace{10em} \text{something} \hspace{10em} \times q$$

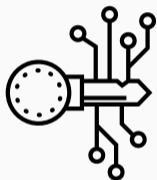
$$q = \gcd(S - S', n)$$

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

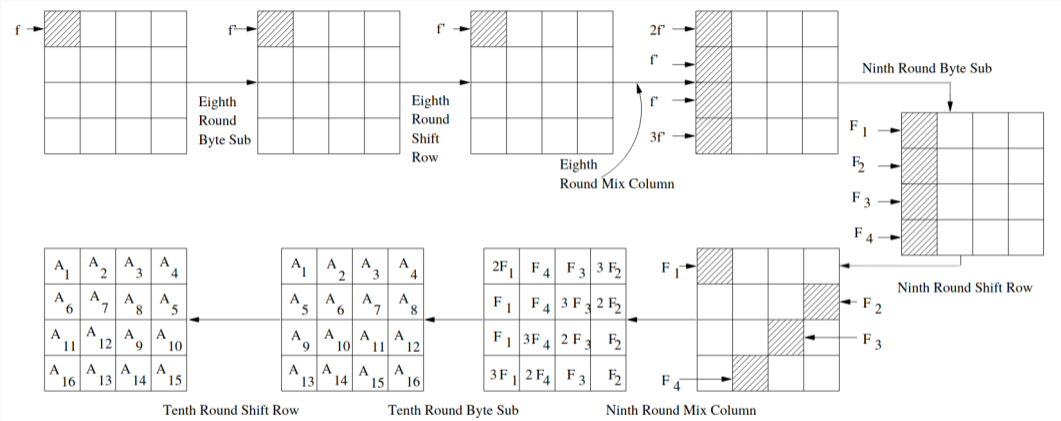
    // Actual decryption
    ippsRSA_Decrypt(ct, dec, pPrv, scratchBuffer);
}
```

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


**WHAT ELSE
CAN WE
BREAK?!**



- 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

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

```
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**

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



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



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

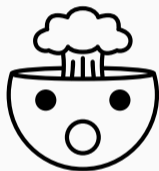
```
foo = arr + offset * 0x24
```

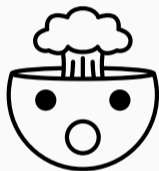


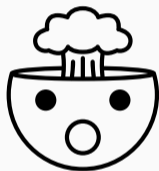
```
Creating enclave...
==== Victim Enclave ====
[pt.c] /dev/sgx-step opened!
Enclave Base: 0x7f001a000000 ←
Enclave Limit: 0x7f001c000000 ←
EDBGRD: debug
█
```

Voltage
0.584V

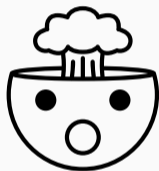
Undervolting
-235mV



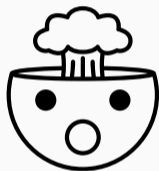




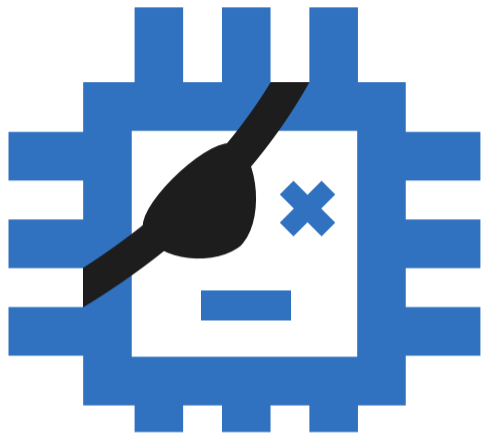
- RSA keys



- RSA keys
- AES keys



- RSA keys
- AES keys
- Memory corruption



PLUNDER
VOLT



- A new type of attack



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



- 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



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



- 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



- 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

- Intel simply removed the undervolting MSR

- Intel simply removed the undervolting MSR
- Cannot be used for legitimate undervolting

- Intel simply removed the undervolting MSR
- Cannot be used for legitimate undervolting
- also...

- Intel simply removed the undervolting MSR
- Cannot be used for legitimate undervolting
- also...
- There is no SGX on consumer CPUs anymore

Side-Channel Security

Chapter 5: Software-based Fault Attacks

Jonas Juffinger

March 27, 2025

www.isec.tugraz.at

References

- [1] 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* 51 (2016), pp. 743–755.
- [2] 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.
- [3] 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).

- [4] Marco Chiappetta, Erkey 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).
- [5] 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.
- [6] Howard David, Chris Fallin, Eugene Gorbatoov, Ulf R Hanebutte, and Onur Mutlu. “Memory power management via dynamic voltage/frequency scaling”. In: *ACM International Conference on Autonomic Computing*. 2011.
- [7] 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.

- [8] 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.
- [9] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA*. 2016.
- [10] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. “Flush+Flush: A Fast and Stealthy Cache Attack”. In: *DIMVA*. 2016.
- [11] 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.
- [12] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. “MASCAT: Stopping Microarchitectural Attacks Before Execution”. In: *Cryptology ePrint Archive, Report 2016/1196* (2017).

- [13] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. “SGX-Bomb: Locking Down the Processor via Rowhammer Attack”. In: *SysTEX*. 2017.
- [14] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. “BLACKSMITH: Rowhammering in the Frequency Domain”. In: *S&P*. 2021.
- [15] 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.
- [16] Jonas Juffinger, Sudheendra Raghav Neela, Martin Heckel, Lukas Schwarz, Florian Adamsky, and Daniel Gruss. “Presshammer: Rowhammer and Rowpress without Physical Address Information”. In: *DIMVA*. 2024.

- [17] 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.
- [18] 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.
- [19] Mark Lanteigne. *How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware*. 2016. URL:
<http://www.thirdio.com/rowhammer.pdf>.
- [20] 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.

- [21] 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.
- [22] Matthias Payer. “HexPADS: a platform to detect “stealth” attacks”. In: *ESSoS*. 2016.
- [23] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: *USENIX Security*. 2016.
- [24] Rui Qiao and Mark Seaborn. “A New Approach for Rowhammer Attacks”. In: *HOST*. 2016.

- [25] 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.
- [26] 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>.
- [27] Mark Seaborn and Thomas Dullien. “Exploiting the DRAM Rowhammer bug to gain kernel privileges”. In: *Black Hat USA*. 2015.
- [28] 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.

- [29] 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.
- [30] 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.
- [31] 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.