Politecnico di Torino

Row Hammer

The Silent Threat to Computer Security

Operating Systems for Embedded Systems

Prof. Stefano Di Carlo

Lecture prepared by:

Nicolas Abril

Matteo Isoldi

Massimiliano Di Todaro

Diego Porto

Co-funded by the European Union

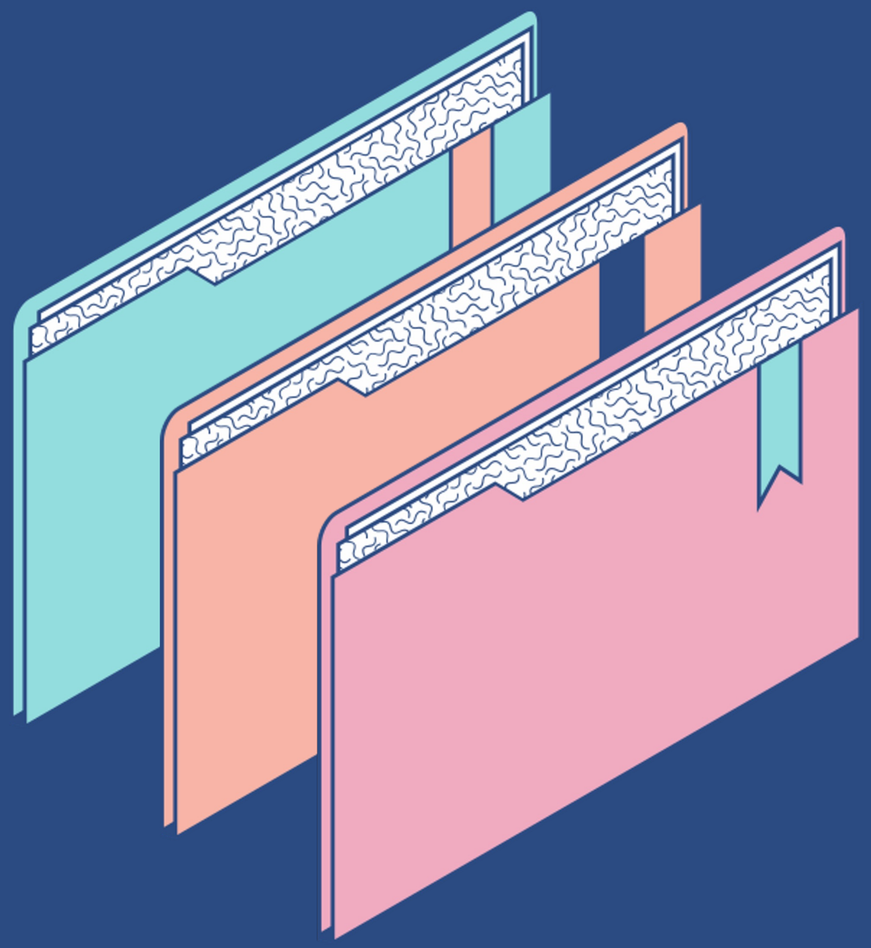
La Région Auvergne-Rhône-Alpes

AGENCE ERASMUS+ FRANCE · EDUCATION FORMATION

Erasmus+

Acknowledgments

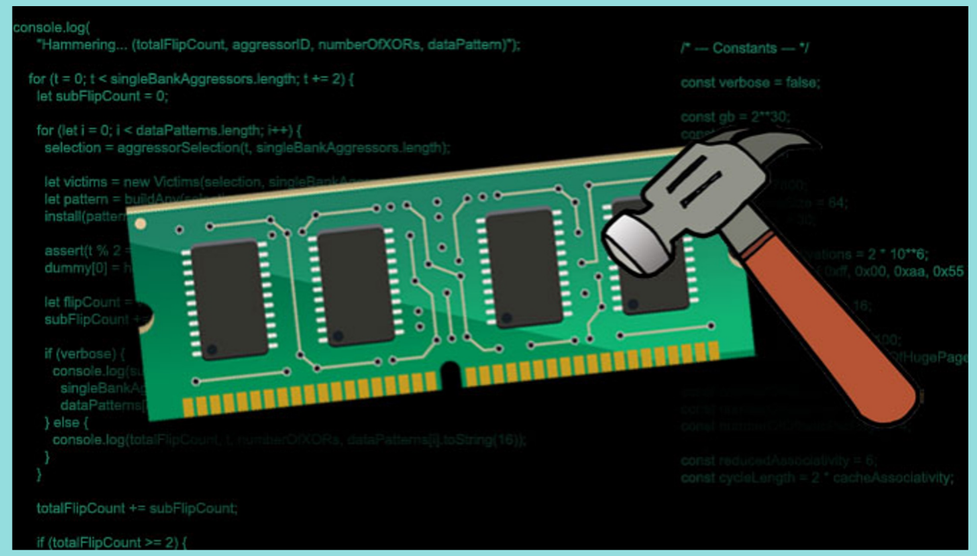
- This material was initially developed as part of an assignment for the Operating Systems for embedded systems course delivered at Politecnico di Torino by Prof. Stefano Di Carlo during the academic year 2022/20023.
- Credits for the preparation of this material go to:
  - ✓ Nicolas Abril (https://github.com/developedby)
  - ✓ Massimiliano Di Todaro (https://github.com/bOhYee)
  - ✓ Matteo Isoldi (https://github.com/mditodaro)
  - ✓ Diego Porto (https://github.com/akhre)



Outline

- Overview of the problem
- DRAM: Description, organization, operations
- Difficulties in performing RowHammer
- Techniques exploiting RowHammer
- Mitigation techniques for RowHammer

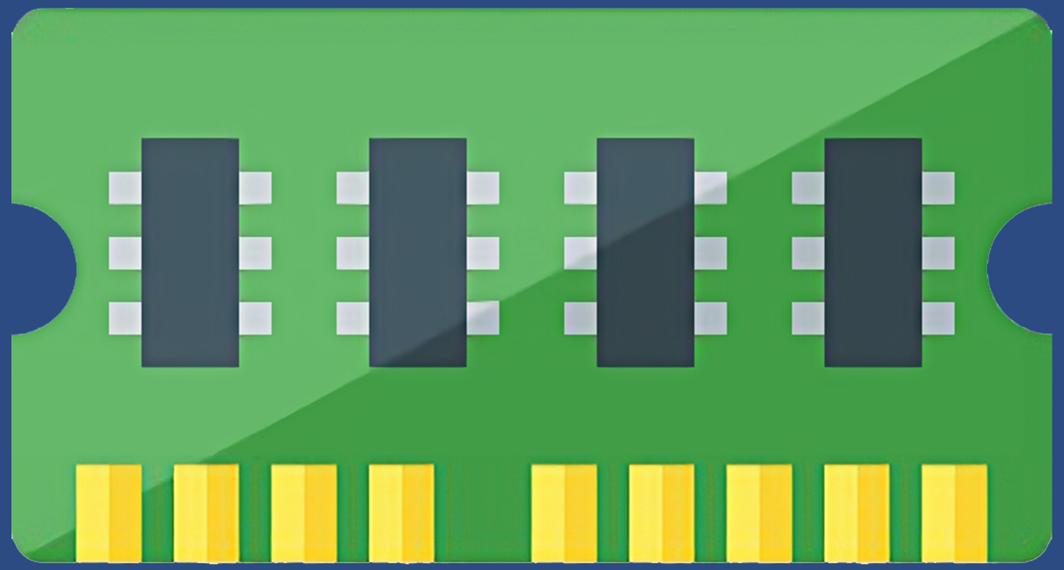
Overview of the problem



- First significant hardware failure that can cause a widespread system security vulnerability.
- A fault in many DRAM modules.
- Caused by a circuit failure mechanism called **disturbance error.**

- Several key moments in the history of RowHammer:
  1. Flipping Bits in Memory Without Accessing them: An Experimental Study of DRAM disturbance Errors (Kim et al. 2014)
  2. Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn and Dullien from Project Zero, 2015)
  3. Drammer: Deterministic RowHammer Attack on Mobile Platforms (van der Veen et al. 2016)

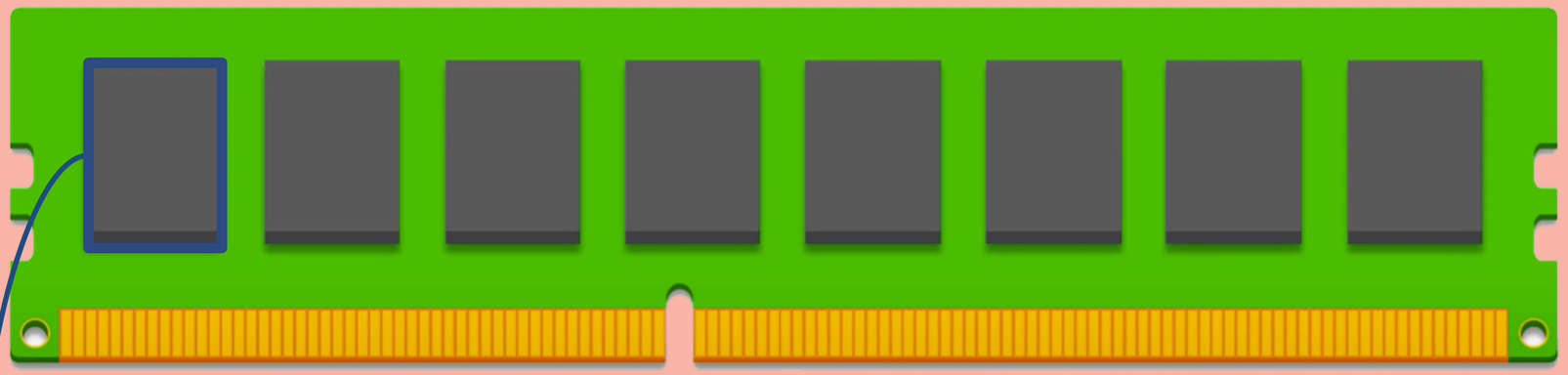
Figure taken from https://whitehat.vn/threads/khai-thac-javascript-qua-ky-thuat-tan-cong-rowhammer-ddr4.14674/



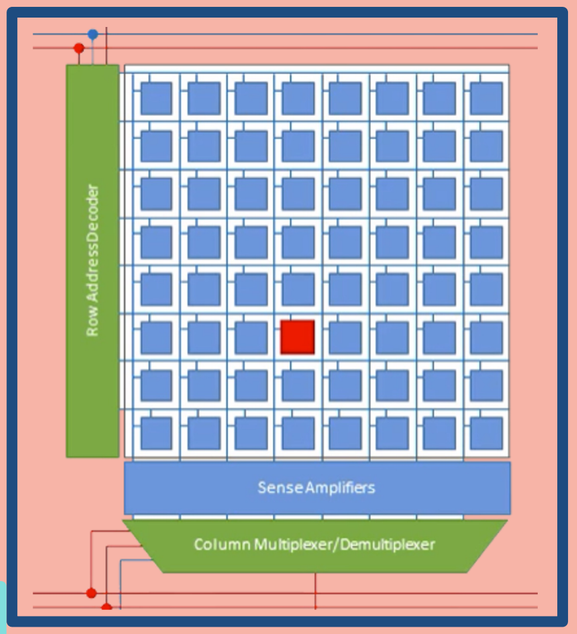
DRAM

Description
Organization
Operations

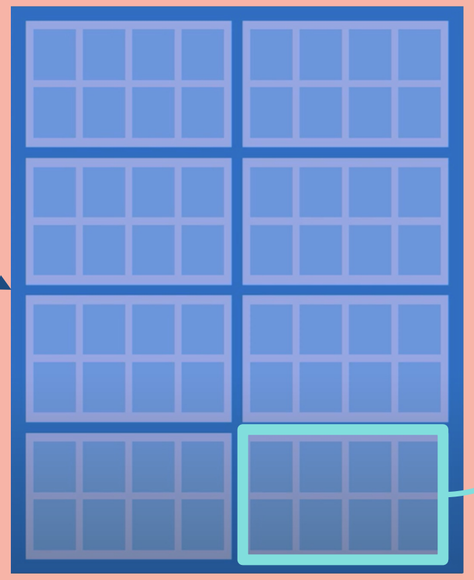
Figures taken from https://www.iconexperience.com/g_collection/icons/?icon=dram



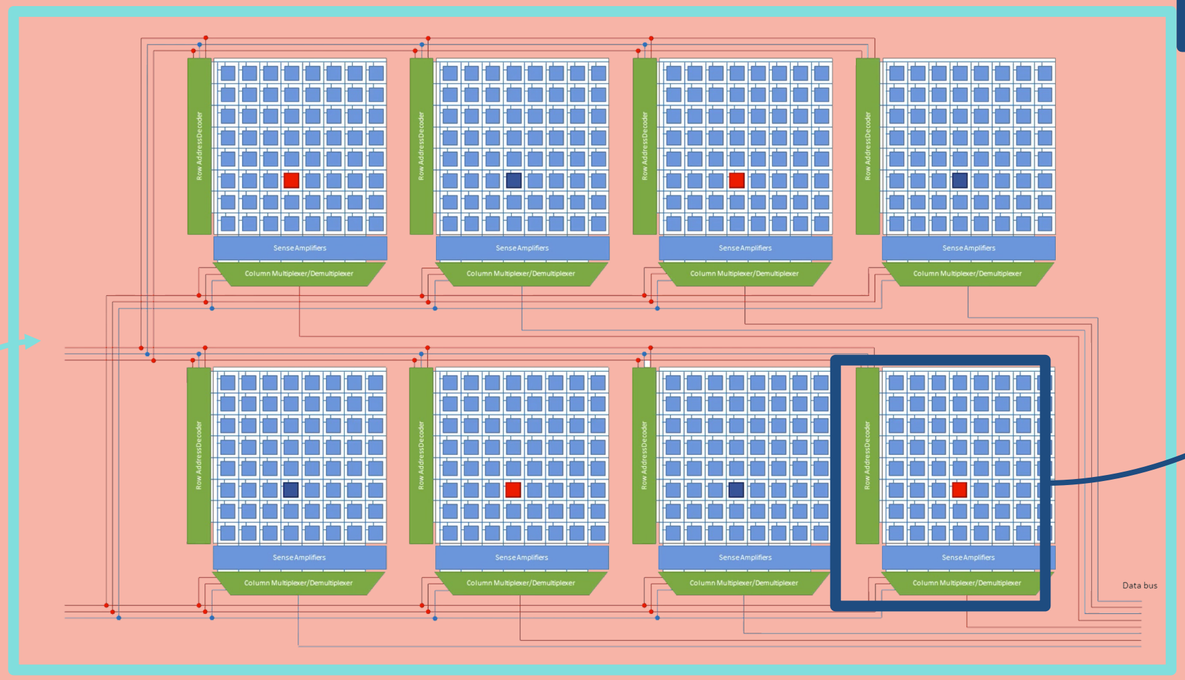
Dual Inline Memory Module



Array of bitcells



Memory modules



Bank

Figures taken from https://youtu.be/Mhqi70OPW0o

Array of bit cells

- **Vertical bit lines** with a parasitic capacitance CL;
- **Horizontal word lines** to select the correct word;
- **Precharging circuit** connects the bit lines to a voltage reference;
- **Sense amplifier** used to detect small voltage differences in order to read the content of the bit cells;
- **Input and output circuits**: that are switches driven by the column decoder and mux.

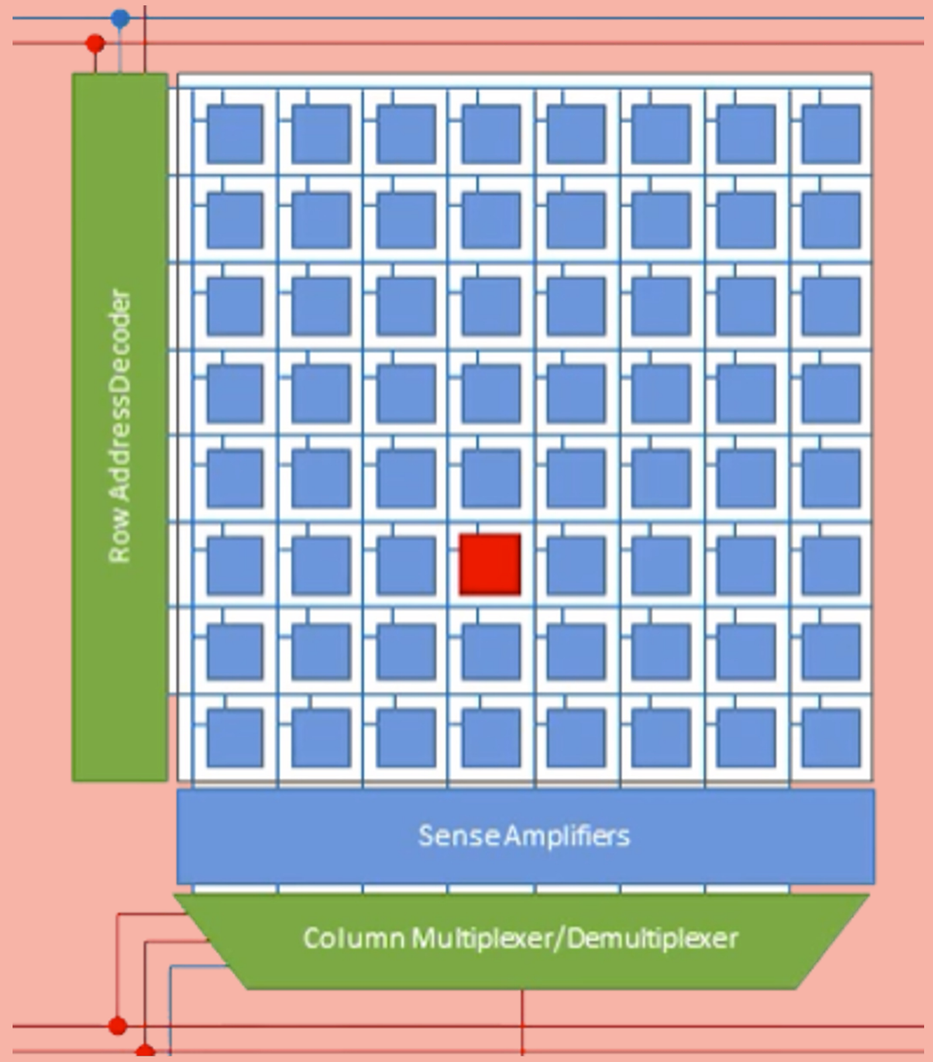


Figure taken from https://youtu.be/Mhqi70OPW0o

Bit cell

- **1T 1C structure bitcell:** More suitable for more densely packed memory chips
- **Trench transistor:** the storage capacitor is obtained by exploiting the parasitic capacitance of this particular transistor that has a very large gate's area

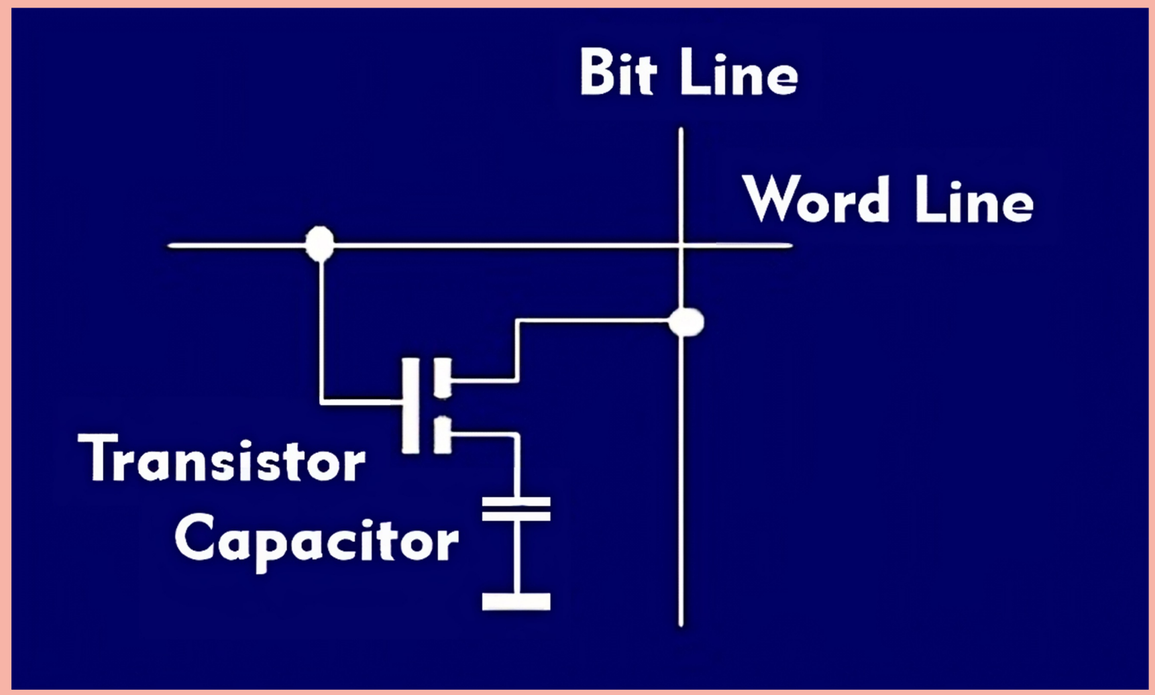


Figure taken from https://www.itwissen.info/lex-images/1T1C-Speicherzelle-ein-Transistor-eine-Kapazitaet_en.png

Operation on DRAM

Read

- The sense amplifier detects small change of the line's voltage
- The information is lost after the reading! It is necessary to write the data back

Write

- The desired value is applied to the bit lines in order to charge (or discharge) the storage capacitor in the bit cell

Refresh

The refresh process reads the data from the sense amplifiers and writes them back into the cells.

- Memory cells are refreshed every 64 milliseconds
- The time needed to refresh a single cell is about 75-120ns.
- Refresh cycles occur while the memory is not used
- The CPU is not aware of it

The DRAM has an excellent write ability but a very low storage permanence because the storage capacitors leak charge over time!

Flipping bit

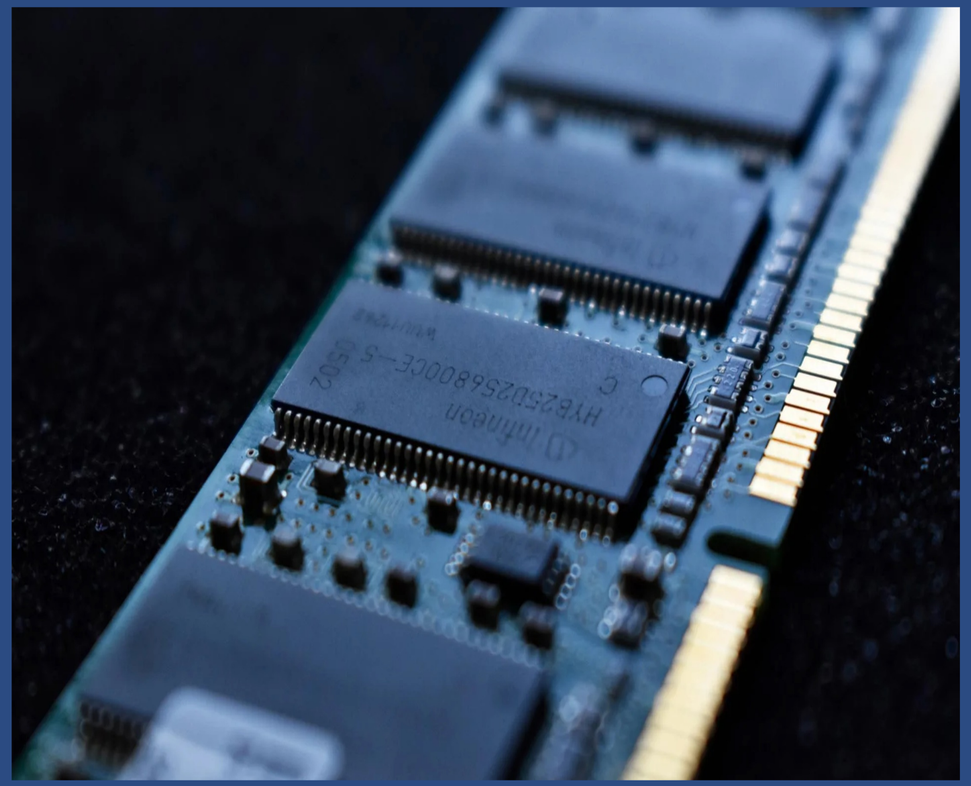
Increasing the cell density has a negative impact on memory reliability:

- Small cell can hold only a limited amount of charge → reduced noise margin.

Noise margin: noise that the bit cell could withstand without compromising the data.

- The close proximity of cells introduces electromagnetic coupling effects between them → undesired interactions

In Row Hammer the attacker, who has access to a DRAM address X, could modify data in a different location Y that is physically near to X in the DRAM.



Difficulties in performing the attack

The problem of the cache and how to access specific DRAM rows.

Figure taken from https://www.pcinvasion.com/ddr4-rowhammer-google-attack-vulnerability/

The problem of the Cache

Repeated row activations can cause bit flips in physically adjacent rows.

- The system needs to access the DRAM with sufficiently high frequency.
- The presence of a **cache** memory could interfere in the flow of the attack by reducing DRAM accesses.
- How to avoid the cache and reach effectively the DRAM?
  - Remove data from the cache.
  - Don't put data in the cache.

- Different techniques can be used:
  a. Non-temporal accesses
  b. Cache eviction
  c. Flushing the cache

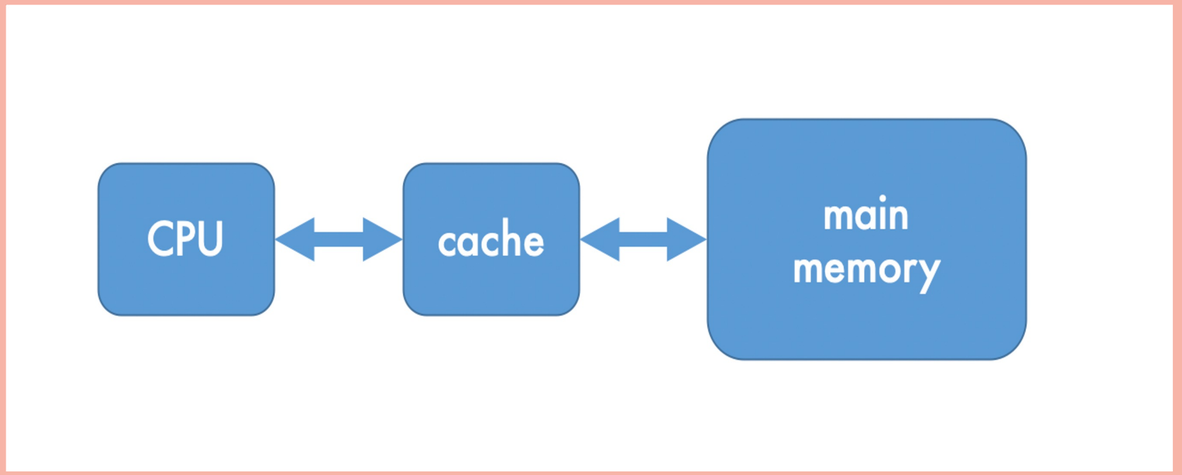


Figure taken from https://blog.feabhas.com/2020/10/introduction-to-the-arm-cortex-m7-cache-part-1-cache-basics/

Avoiding the cache: different techniques

A. Non-Temporal accesses

- Using CPU instructions or APIs that don't follow normal cache-coherency rules.
- The data will not likely be reused, thus does not have to be cached.

A. Cache Eviction

- Repeatedly accessing rows that belongs to the same *cache eviction set.*
- Accesses to a memory address belonging to the same set of a previously-read address will automatically flush the cache while reading.
- Useful if cache maintenance instruction cannot be used.

A. Flushing the cache

- Instruction dependent technique.
- Based on native x86 **clflush** instruction (Kim et al.)
- Since it is based on Flush and Reload loop, the attack is performed both on DRAM and on the cache.

We followed the last approach by using 3 different ArmV8 instructions:

DC CVAC

Clean data cache by address to Point of Coherency (PoC).

DC CIVAC

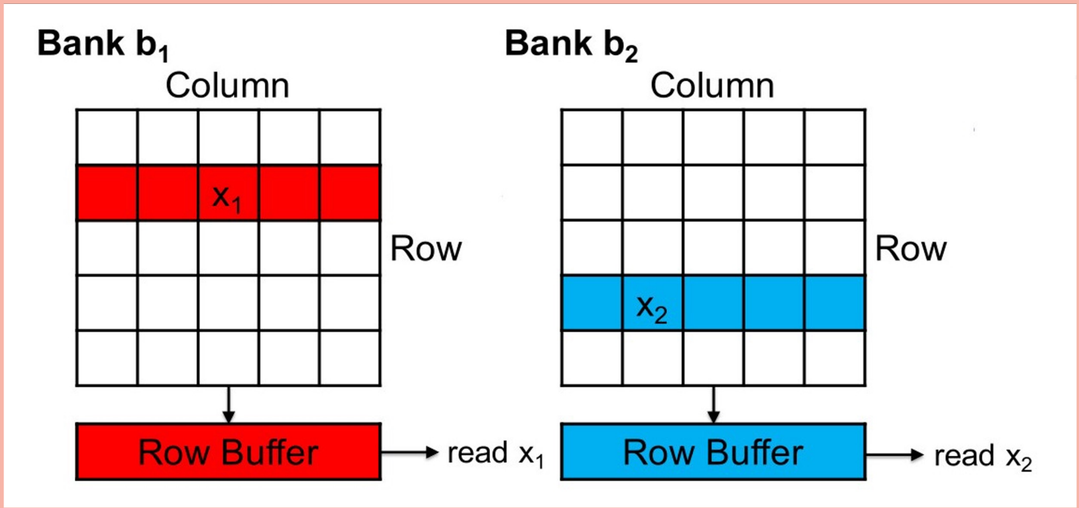
Clean and Invalidate data cache by address to Point of Coherency (PoC).

DC ZVA

Zeroes data cache by virtual addresses.

How to target accesses

- Fixed mapping between physical addresses and DRAM rows (undocumented in most cases).
- One possible strategy is to use reverse engineering.
- Algorithm based on timing latency for accessing different DRAM banks.
- We can exploit this timing difference to identify row bits for the addresses.
- Methods provided by *One Bit Flip, One Cloud Flops: Cross-VM RowHammer Attacks and Privilege Escalation* (Xiao et al.).



- **How to retrieve Physical Addresses?**
  A User-space process with sufficient privileges can retrieve by reading a specific file called **pagemap**, stored in the */proc* directory.
- Physical pages identified by the **Page Frame Number (PFN)**.
- Due to RowHammer vulnerability, since Linux 4.0 only users with admin capabilities can get PFNs.

Figures taken from https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/xiao

Exploiting RowHammer

How RowHammer attacks are used to take control or compromise the integrity of systems

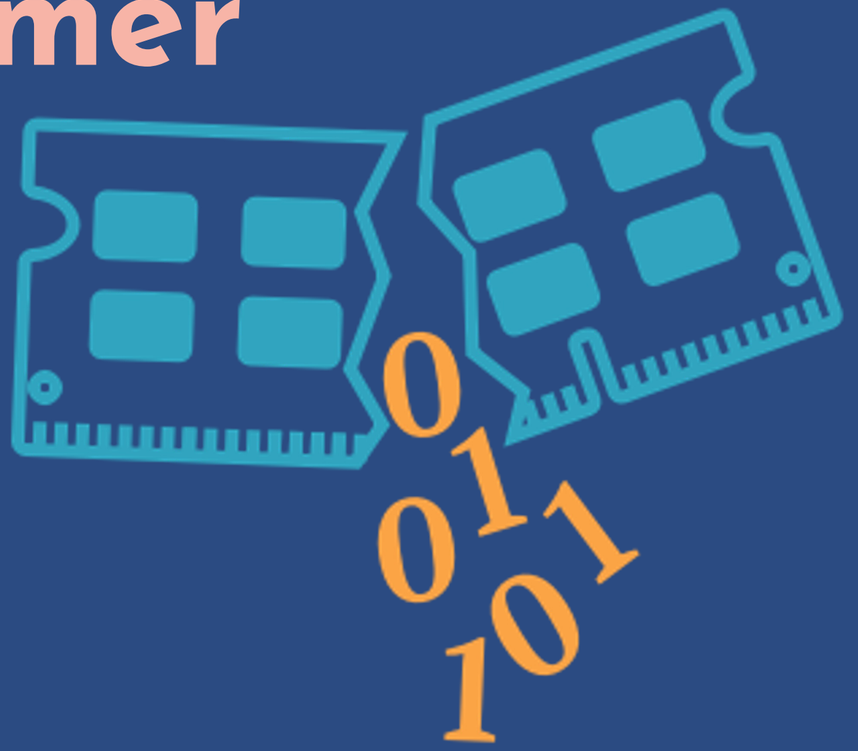
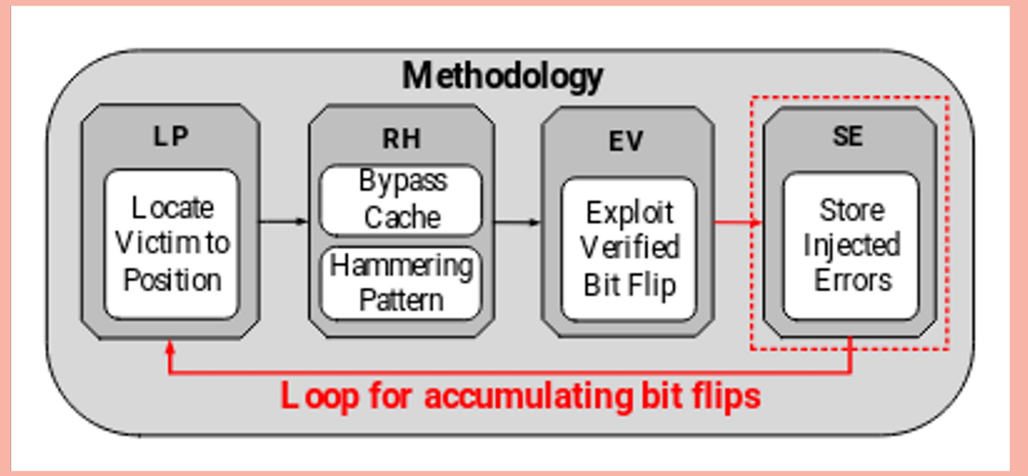


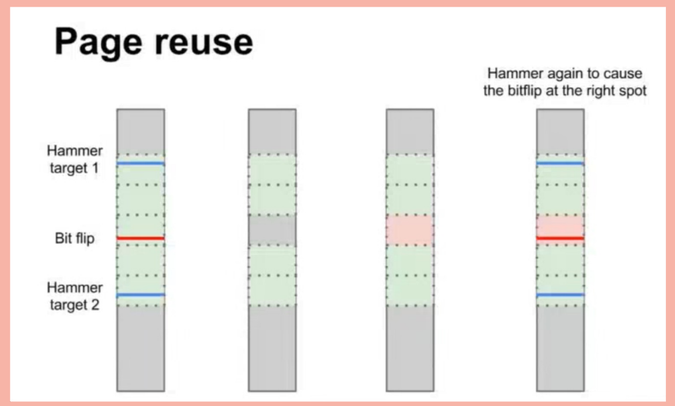
Figure taken from https://rambleed.com/

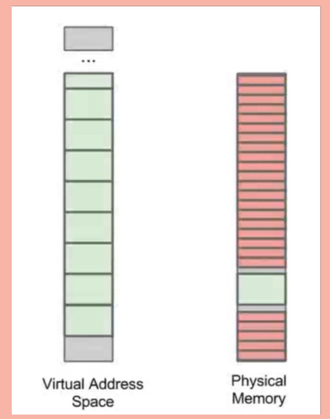
RowHammer flow

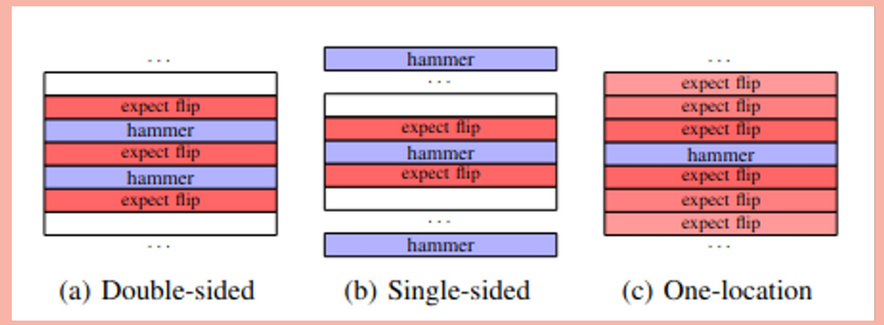
In general, most Rowhammer-based attacks follow the same flow:

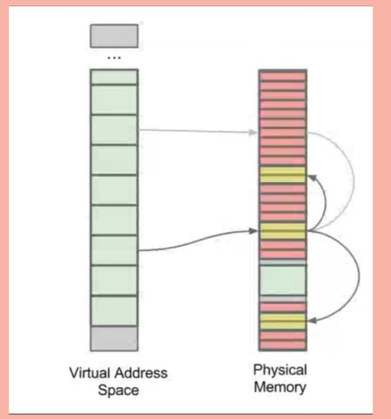
- **Location Preparation**:
  - Find vulnerable bits
  - Place victim structure in desired location
- **Rapid Hammering**: Triggering the RowHammer vulnerability by rapidly activating rows close to victim
  - Avoid cached access
  - Avoid detection by protections
- **Exploit Verification**:
  - Check that the intended bitflips happened
  - Use the error to escalate
- **Store Errors**: Propagate the modified structure back for further bitflips











Figures taken from https://doi.org/10.48550/arXiv.1901.03538, https://doi.org/10.1109/SP40000.2020.00020 and https://www.youtube.com/watch?v=0U7511Fb4to

Attack Classification

RowHammer attacks can be classified on some characteristics

- **Attack Origin**
  Where the trigger for the attack comes from. Could be:
  - **Local**: A normal program, that is able to execute arbitrary code.
  - **Remote**: External data running in a constrained environment.
- **Intended Implication**
  What kind of data is being corrupted by the bitflip
  - Same privilege / Higher privilege
  - Readable / Unreadable
- **Application**
  What is being exploited with the attack, the consequences:
  - Privilege escalation and Sandbox Escape
  - Leaking secrets
  - Denial of Service
  - Data corruption and Program Misbehavior

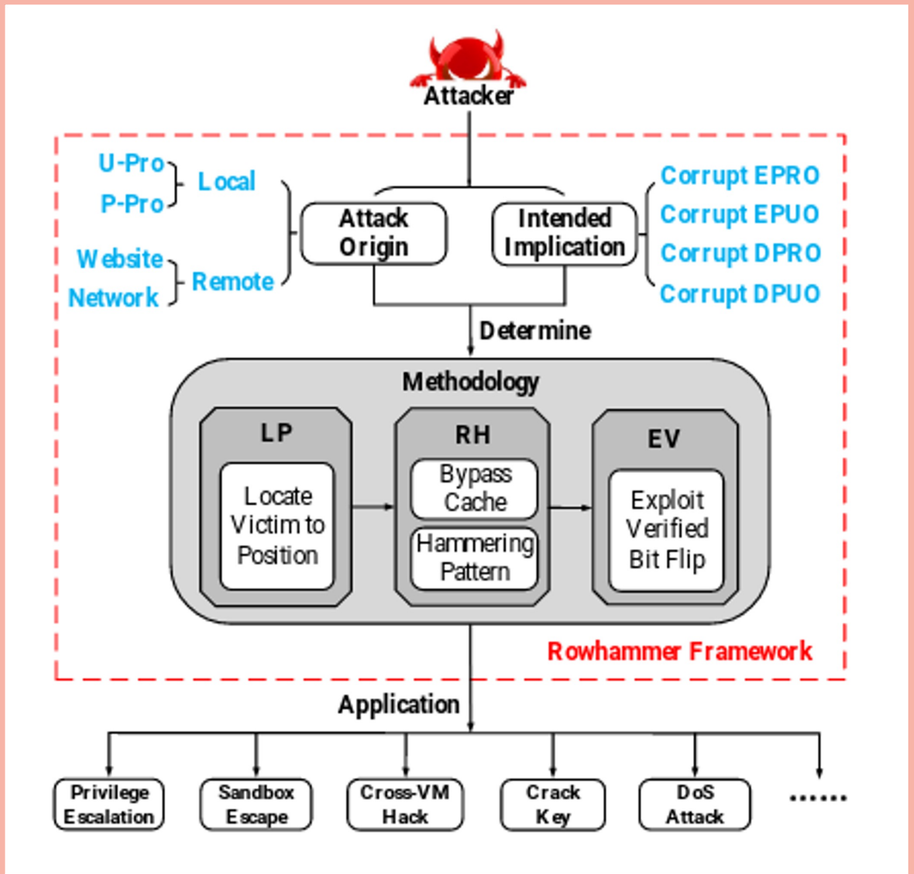


Figure taken from https://doi.org/10.48550/arXiv.1901.03538

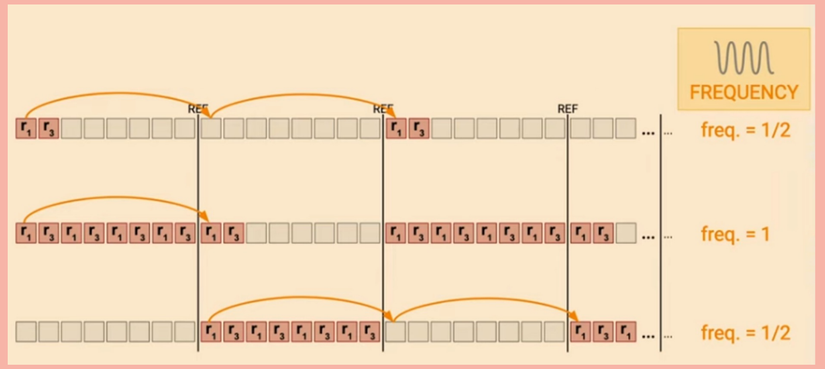
Examples of Attacks

- **Rowhammer.js**:
  - Uses large JS arrays to find and place vulnerable data.
  - Fast cache eviction strategies and double-sided hammering for fast bit-flipping.
  - Triggered by remote code, can escape the browser sandbox, even gain root privilege.

- **RAMBleed**:
  - Memory massage technique to place secret to be leaked next to vulnerable row.
  - Hammer pages on same row as secret to make it leak to the victim row.

- **Blacksmith:**
  - Non-uniform access of aggressor and unrelated rows over long patterns evades many Rowhammer mitigations.
  - Fuzzer for finding optimal hammer pattern.
  - Supports different location preparation schemes.

- **Rowhammer-like attacks on MLC NAND Flash:**
  - Same principle as Rowhammer, used to corrupt data in NAND Flash.
  - Not as precise and targeted, still enough for privilege escalation.
  - Showcases the same problem on other devices storing data as charge.



Figures taken from https://www.youtube.com/watch?v=7bGedQygqkA and https://rambleed.com/docs/20190603-rambleed-web.pdf

Where are we headed?

What are the research community and industry doing to prevent memory failures from happening?

Figure taken from https://lespritalenvers.org/wp-content/uploads/2016/12/lavoro.jpg

Different attacks, Different solutions

How the academic world is pushing new ideas to target the threat

Various solutions have been proposed by researchers and engineers in the past years.
They differ between each other, but they can be classified in four main categories:

1. Tracking approaches
2. Sampling approaches
3. Partitioning approaches
4. Clean-slate approaches

Different attacks, Different solutions

How the academic world is pushing new ideas to target the threat

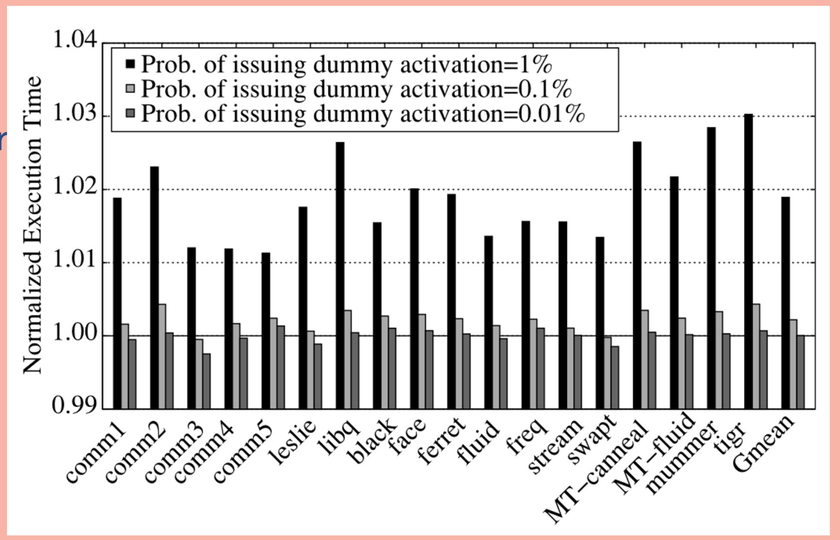
	Per-Bank (bits)	Per-Channel (kilo-bytes)	Per-CPU (kilo-bytes)
Graphene	2,511	39.23	156.9
BlockHammer	13,312	208.00	832.0
TWiCe	22,200	346.88	1,387.5

1. Tracking approaches ⟶ They track microarchitectural events through the use of a small, fast memory in order to intervene when these happen at an high rate
   - Pros: highly reactive towards high-accessed rows
   - Cons: performance drop, circuit area reduced, memory modifications required

   Most notable example of this approach are Graphene and BlockHammer

1. Sampling approaches ⟶ They track every row access to the memory ar based on probability, a refreshing operation is performed on hypothetical victim rows
   - Pros: highly reactive towards row accesses
   - Cons: considerable performance drop, memory modifications required

   Most notable example is PRA (Probabilistic Row Activation)



Figures taken from https://stefan.t8k2.com/publications/dramsec/2021/panopticon.pdf, https://doi.org/10.1109/LCA.2014.2332177

Different attacks, Different solutions

How the academic world is pushing new ideas to target the threat

3. Partitioning approaches ⟶ Memory rows are delimited by “guard rows”, rows with no meaningful data inside, in order to avoid having bit flips in adjacent rows

- Pros: intrinsic prevention, no memory involved
- Cons: storage area reduced, memory modifications required, high complexity in managing va-to-pa mappings

Most notable example of this approach is GuardION

4. Clean-slate approaches ⟶ They design new possible structure for the memory to reduce cell disturbances

TRR: the vendors solution

How DRAM vendors moved without a standard

Without a direction from JEDEC, vendors thought of a way to prevent RowHammer from happening: TRR. Target Row Refresh is a mitigation specific to the vendor, but it is generally based on a tracking approach and consists of important building blocks:

- Sampler ⟶ Track hammered rows
- Inhibitor ⟶ Applies the mitigation mechanism

TRR has been present since before 2020, but researchers have proved that this mechanism fails at doing its job efficiently due to the sampler's memory limited size.

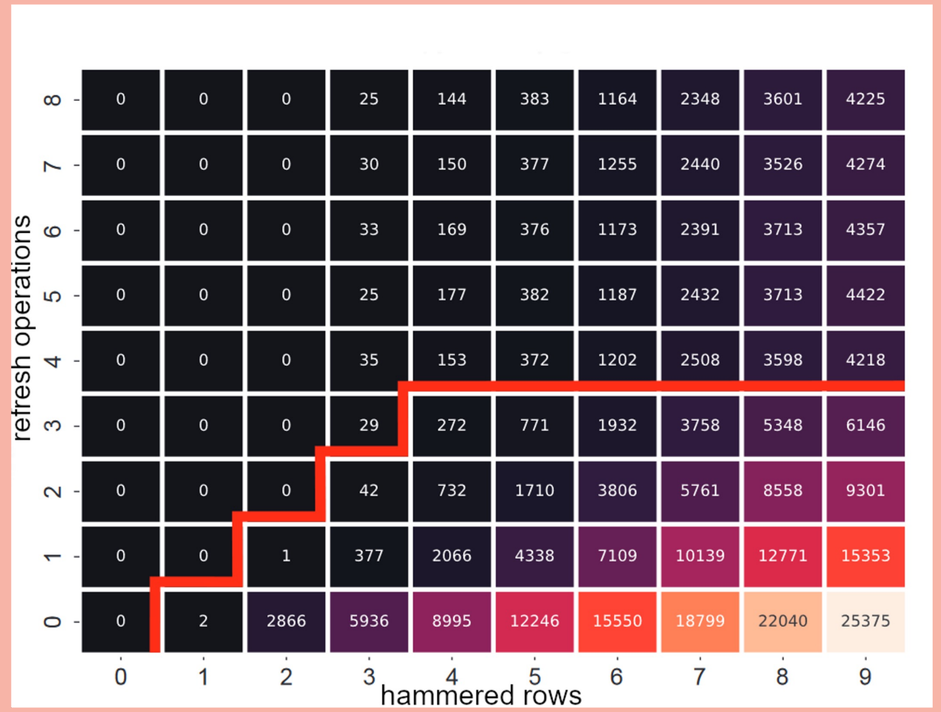


Figure taken from https://doi.org/10.1109/SP40000.2020.00090

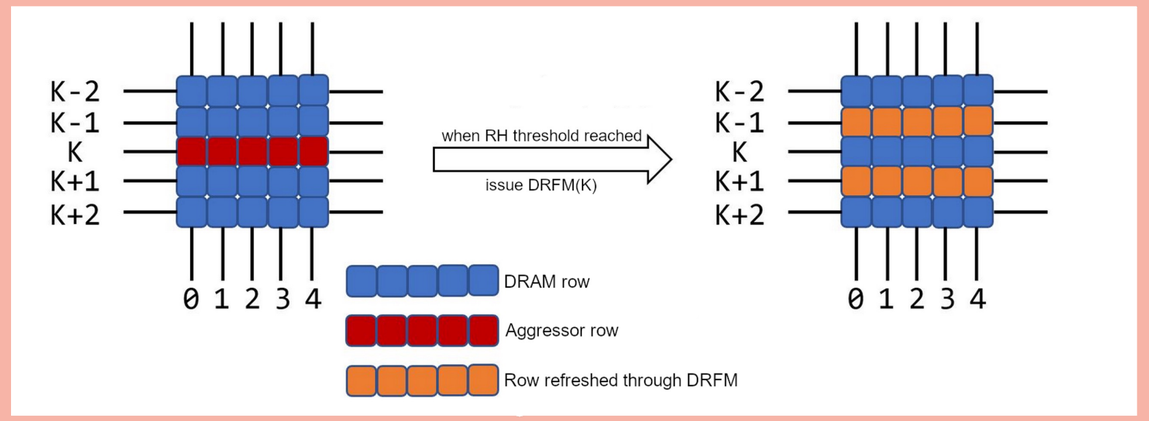
A possible path: DRFM

With the new standard for DDR5 a new path has been opened for mitigations

Last year, JEDEC added with the new standard, for DDR5, a command for MMU to request a refreshing operation for all nearby rows of an aggressor row: DRFM or **Directed Refresh Management**.

This allows a new, broadened range of possible mitigation used to hinder row hammering on memory, since prevention mechanism can be designed outside the memory itself.

Even if the standard details this new command, it doesn't go too deeply in its usage (so, when and how to use it). This permits flexibility for MMU but for researchers mean bad application and coverage for possible attacks.



First figure taken from https://upload.wikimedia.org/wikipedia/commons/4/41/JEDEC_Logo.svg

Second figure modified starting from https://stefan.t8k2.com/rh/DRFM/images/transitive-rh.jpg

Conclusion

Although a new step was taken with the recent standard, its effect will be seen in the years to come. Even so, RowHammer remains a threat and its influence is rapidly expanding towards other types of memories (i.e., NAND Flash). It is crucial that manufacturers and researchers continue to explore new mitigation strategies and work to develop more secure and resilient memory architectures,in order to prevent further complications in the future, when scaling technology advances will increase cell interferences.

Lecture credits to

Nicolas Abril
Massimiliano Di Todaro
Matteo Isoldi
Diego Porto

Enrolled in the Operating Systems for Embedded Systems Course in 2023