TRRespass: Exploiting the Many Sides of Target Row Refresh

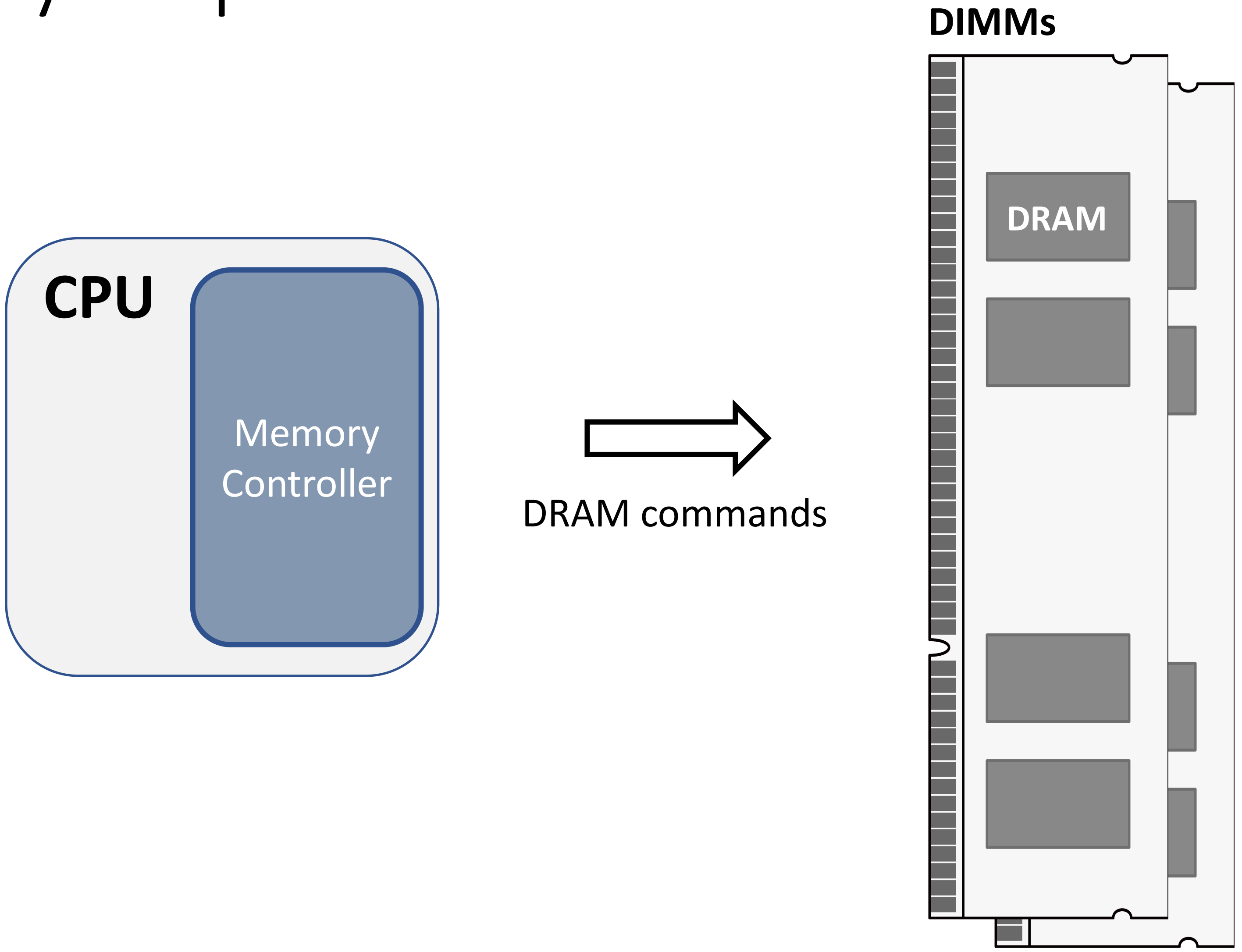
Pietro Frigo[1] Emanuele Vannacci[1] Hasan Hassan[2] Victor Van der Veen[3]
Onur Mutlu[2] Herbert Bos[1] Cristiano Giuffrida[1] Kaveh Razavi[1]

[1]Vrije Universiteit Amsterdam [2]ETH Zürich [3]Qualcomm Technologies Inc.

Teaser

- Memory vendors advertise RowHammer-free devices
- What is Target Row Refresh (TRR)? Not a single mitigation!
- Reverse-engineering of in-DRAM mitigations
  - The Many-sided RowHammer
  - Hammering up to 20 aggressor rows
- 3 major vendors all vulnerable: Samsung, Micron, SK Hynix
  - Currently representing over 95% of the DRAM market

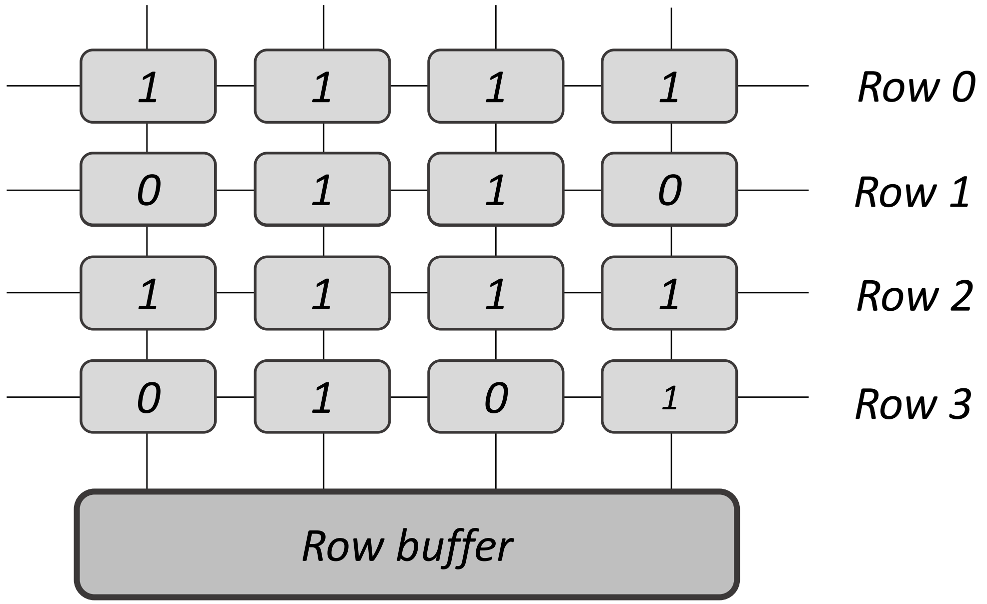
Memory request flow



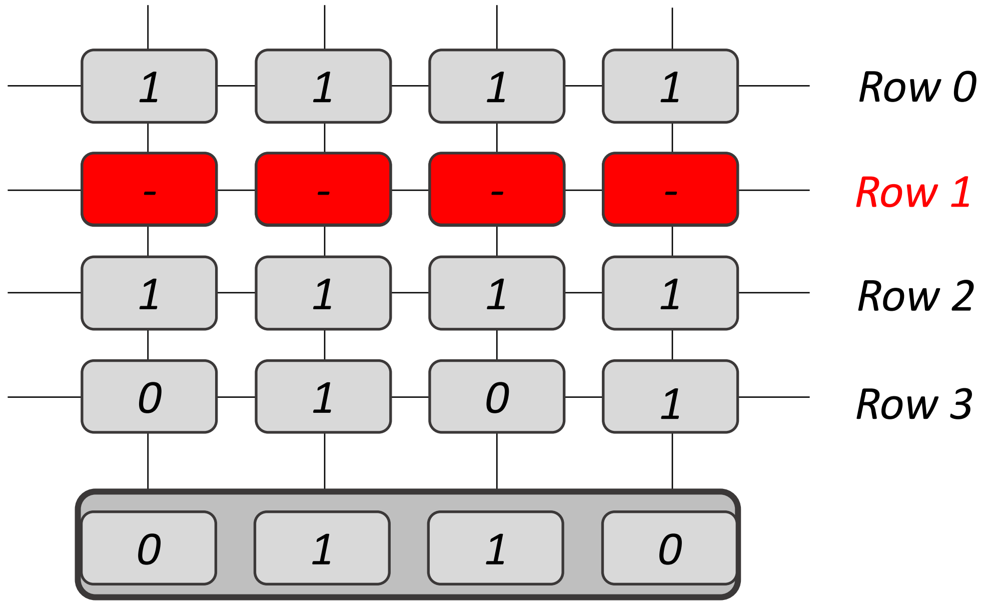
DRAM Refresh

- DRAM is dynamic because data must be refresh periodically
  - Retention time (i.e., 64ms)
- The MC issues a **REFRESH** command every 7.8µs
  - Only a small portion of memory is refreshed with a command
  - 8192 refreshes within a 64ms interval

Memory array

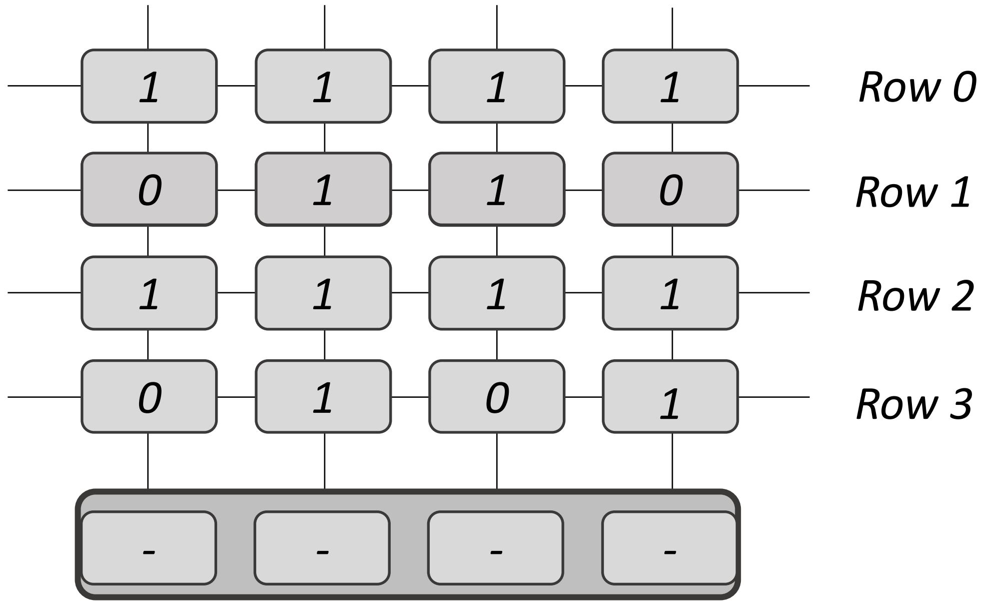


Read operation: Row 1



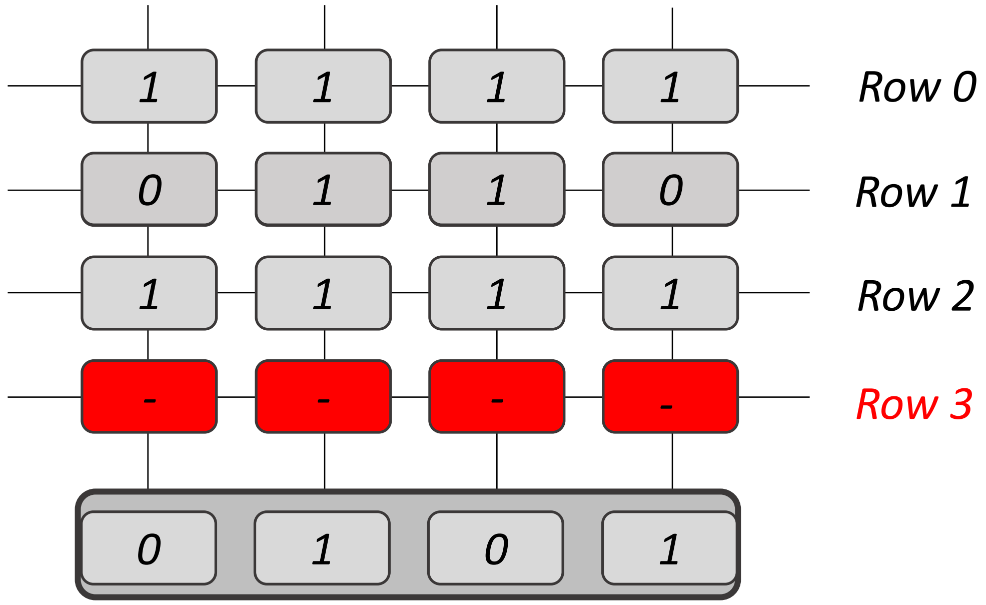
ACTIVATE Row 1

Read operation: Row 3



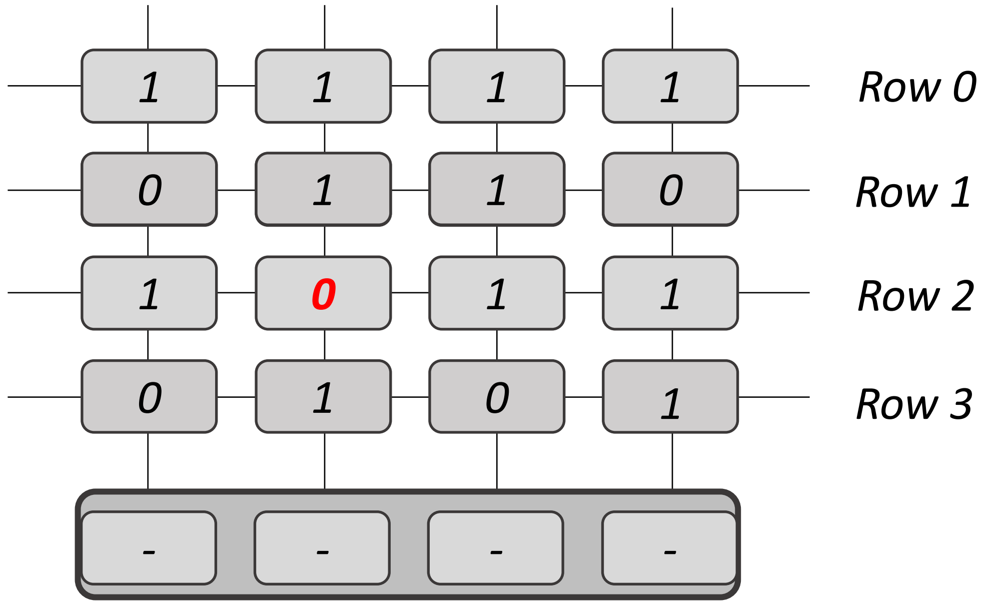
PRECHARGE Row 1

Read operation: Row 3



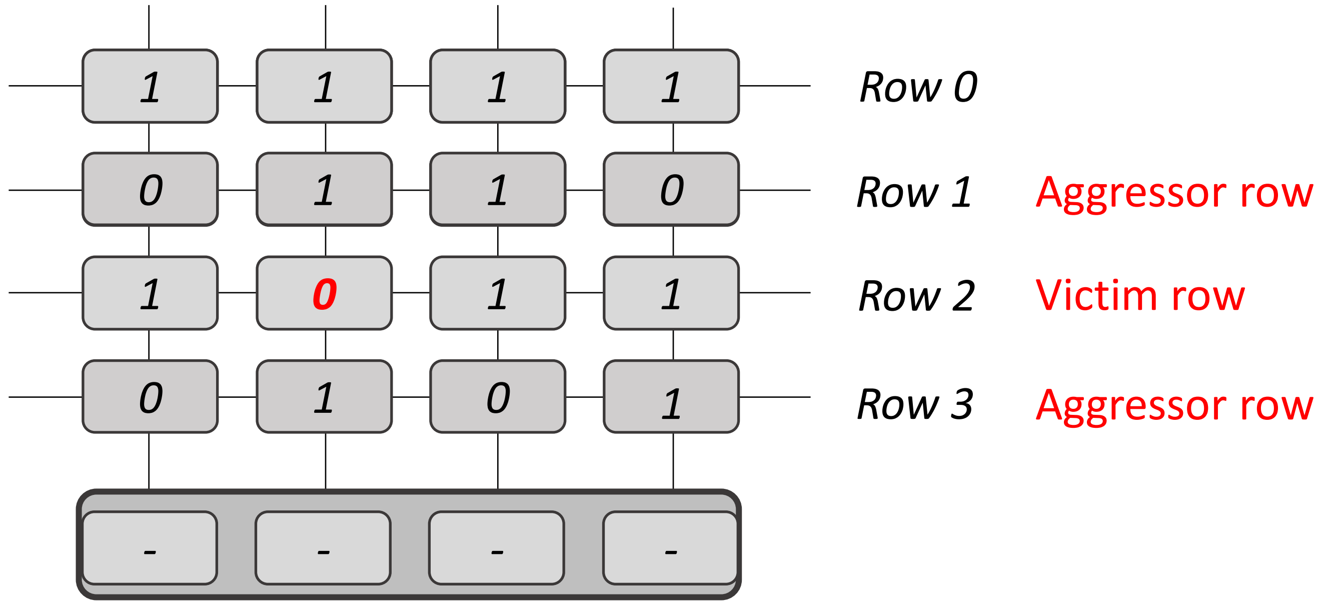
ACTIVATE Row 3

RowHammer



Bit flip!

Double-sided RowHammer



Bit flip!

Hardware mitigations

- Error-correcting code (ECC) [1]

Refreshing a row restores the cells electric charge: it prevents flips.

- Double refresh

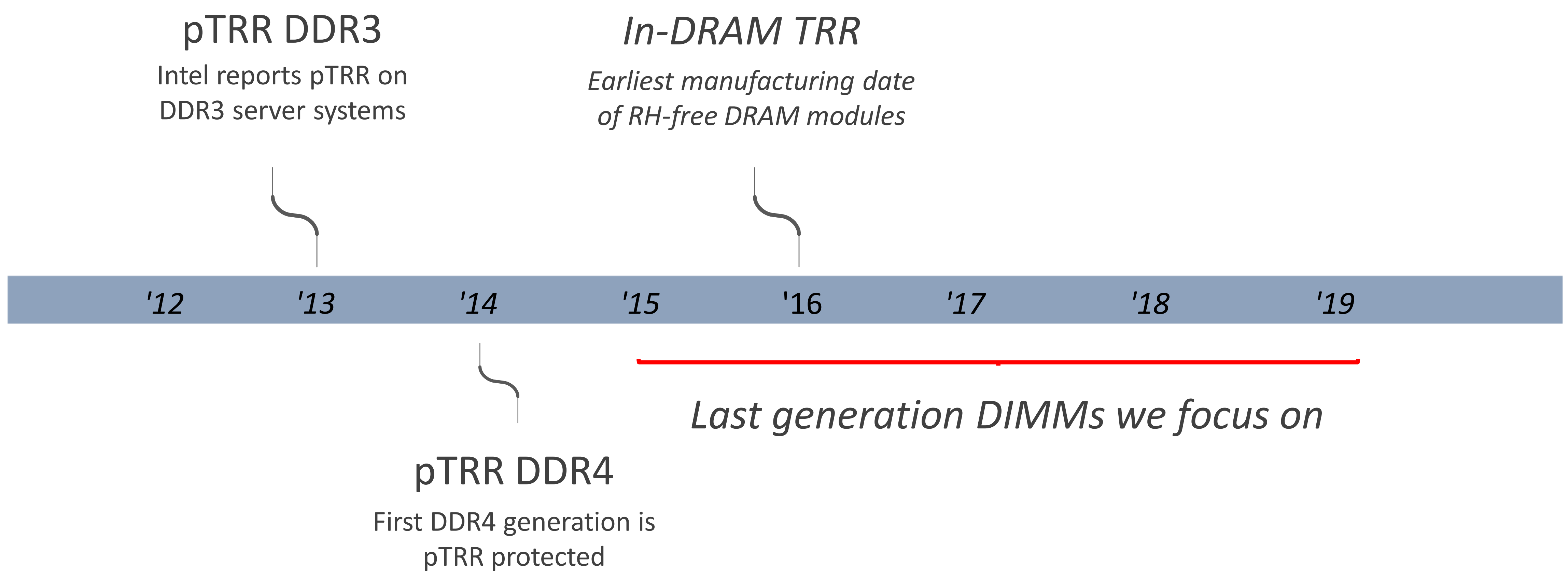
- Target Row Refresh (TRR)

[1*] L. Cojocaret al., “Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks,” in S&P, 2019.*

Target Row Refresh

- TRR-like mitigations track rows activations and refresh victim rows
  - Many possible implementations in practice
  - Security through obscurity
- Pseudo TRR (pTRR)
  - Memory controller implementation
- In-DRAM TRR
  - Embedded in the DRAM circuitry

Timeline


pTRR DDR3
Intel reports pTRR on DDR3 server systems
In-DRAM TRR
Earliest manufacturing date of RH-free DRAM modules
'12
'13
'14
'15
'16
'17
'18
'19
Last generation DIMMs we focus on
pTRR DDR4
First DDR4 generation is pTRR protected

Goals

- Reverse engineer TRR to demystify in-DRAM mitigations
- Memory device assessment
    - A Novel hammering pattern: **The Many-sided RowHammer**
    - Hammering up to **20 aggressor rows** allows to bypass TRR
- Automatically test memory devices: **TRRespass**
    - Automate hammering patterns generation

Challenges

- Analysis from the CPU side not possible
  - No timing side-channels
- FPGA-based memory controller [1,2]

SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies

Hasan Hassan[1,2,3] Nandita Vijaykumar[3] Samira Khan[4,3] Saugata Ghose[3] Kevin Chang[3] Gennady Pekhimenko[5,3] Donghyuk Lee[6,3] Oguz Ergin[2] Onur Mutlu[1,3]
[1]ETH Zürich [2]TOBB University of Economics & Technology [3]Carnegie Mellon University [4]University of Virginia [5]Microsoft Research [6]NVIDIA Research

DRAM is the primary technology used for main memory in modern systems. Unfortunately, as DRAM scales down to smaller technology nodes, it faces key challenges in both data integrity and latency, which strongly affects overall system reliability and performance. To develop reliable and high-performance DRAM-based main memory in future systems, it is critical to characterize, understand, and analyze various aspects (e.g., reliability, latency) of existing DRAM chips. To enable this, there is a strong need for a publicly-available DRAM testing infrastructure that can flexibly and efficiently test DRAM chips in a manner accessible to both software and hardware developers.

This paper develops the first such infrastructure, SoftMC (Soft Memory Controller), an FPGA-based testing platform that can control and test memory modules designed for the commonly-used DDR (Double Data Rate) interface. SoftMC has two key properties: (i) it provides flexibility to thoroughly control memory behavior or to implement a wide range of mechanisms using DDR commands; and (ii) it is easy to use as it provides a simple and intuitive high-level programming interface for users, completely hiding the low-level details of the FPGA.

We demonstrate the capability, flexibility, and programming ease of SoftMC with two example use cases. First, we implement a test that characterizes the retention time of DRAM cells. Experimental results we obtain using SoftMC are consistent with the findings of prior studies on retention time in modern DRAM, which serves as a validation of our infrastructure. Second, we validate two recently-proposed mechanisms, which rely on accessing recently-refreshed or recently-accessed DRAM cells faster than other DRAM cells. Using our infrastructure, we show that the expected latency reduction effect of these mechanisms is not observable in existing DRAM chips, which demonstrates the usefulness of SoftMC in testing new ideas on existing memory modules. We discuss several other use cases of SoftMC, including the ability to characterize emerging non-volatile memory modules that obey the DDR standard. We hope that our open-source release of SoftMC fills a gap in the space of publicly-available experimental memory testing infrastructures and inspires new studies, ideas, and methodologies in memory system design.

1. Introduction

DRAM (Dynamic Random Access Memory) is the predominant technology used to build main memory systems of modern computers. The continued scaling of DRAM process technology has enabled tremendous growth in DRAM density in the last few decades, leading to higher capacity main memories. Unfortunately, as the process technology node scales down to the sub-20 nm feature size range, DRAM technology faces key challenges that critically impact its reliability and performance.

The fundamental challenge with scaling DRAM cells into smaller technology nodes arises from the way DRAM stores data in cells. A DRAM cell consists of a transistor and a capacitor. Data is stored as charge in the capacitor. A DRAM cell cannot retain its data permanently as this capacitor leaks its charge gradually over time. To maintain correct data in DRAM, each cell is periodically refreshed to replenish the charge in the capacitor. At smaller technology nodes, it is becoming increasingly difficult to store and retain enough charge in a cell, causing various reliability and performance issues [60, 61]. Ensuring reliable operation of the DRAM cells is a key challenge in future technology nodes [38, 45, 60, 61, 65, 68, 71].

The fundamental problem of retaining data with less charge in smaller cells directly impacts the reliability and performance of DRAM cells. First, smaller cells placed in close proximity make cells more susceptible to various types of interference. This potentially disrupts DRAM operation by flipping bits in DRAM, resulting in major reliability issues [46, 66, 74, 83, 84, 90], which can lead to system failure [66, 84] or security breaches [25, 46, 82, 85, 86, 95, 98]. Second, it takes longer time to access a cell with less charge [27, 56], and write latency increases as the access transistor size reduces [38]. Thus, smaller cells directly impact DRAM latency, as DRAM access latency is determined by the worst-case (i.e., slowest) cell in any chip [17, 56]. DRAM access latency has not improved with technology scaling in the past decade [6, 36, 57, 71], and, in fact, some latencies are expected to increase [38], making memory latency an increasingly critical system performance bottleneck.

As such, there is a significant need for new mechanisms that improve the reliability and performance of DRAM-based main memory systems. In order to design, evaluate, and validate many such mechanisms, it is important to accurately characterize, analyze, and understand DRAM (cell) behavior in terms of reliability and latency. For such an understanding to be accurate, it is critical that the characterization and analysis be based on the experimental studies of real DRAM chips, since a large number of factors (e.g., various types of cell-to-cell interference [46, 74, 83], inter- and intra-die process variation [17, 18, 56, 58, 75], random effects [29, 61, 91, 103], operating conditions [59, 61], internal organization [30, 43, 61], stored data patterns [43, 44, 61]) concurrently impact the reliability and latency of cells. Many of these phenomena and their interactions cannot be properly modeled (e.g., in simulation or using analytical methods) without rigorous experimental characterization and analysis of real DRAM chips. The need for such experimental characterization and analysis, with the goal of building the understanding necessary to improve the reliability and performance of future DRAM-based main memories at various levels (both software and hardware), motivates the need for a publicly-available DRAM testing infrastructure that can enable system users and designers to characterize real DRAM chips.

Two key features are desirable from such an experimental memory testing infrastructure. First, the infrastructure should be flexible enough to test any DRAM operation (supported by the commonly-used DRAM interfaces, e.g., the standard Double Data Rate, or DDR, interface) to characterize cell behavior or evaluate the impact of a mechanism (e.g., adopting different refresh rates for different cells [42, 44, 60, 79, 96]) on real DRAM chips. Second, the infrastructure should be easy to use, such

[1] *H. Hassan et al., “SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies,” in HPCA, 2017*
[2] *SAFARI Research Group, “SoftMC — GitHub Repository,” https:// github.com/CMU-SAFARI/SoftMC.*

Building blocks

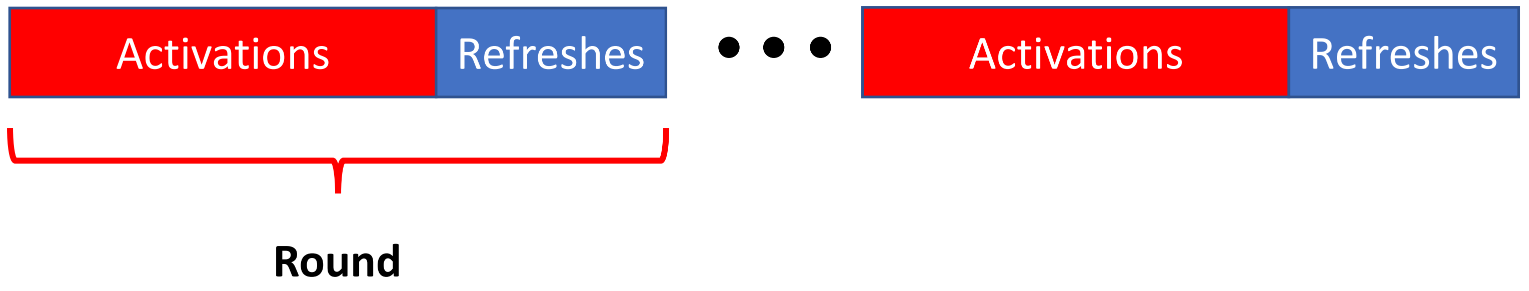
Abstractions:

- **Sampler**
  - Track aggressor rows activations
  - Keep a set of rows
- **Inhibitor**
  - Prevent bit flips
  - Refresh victims

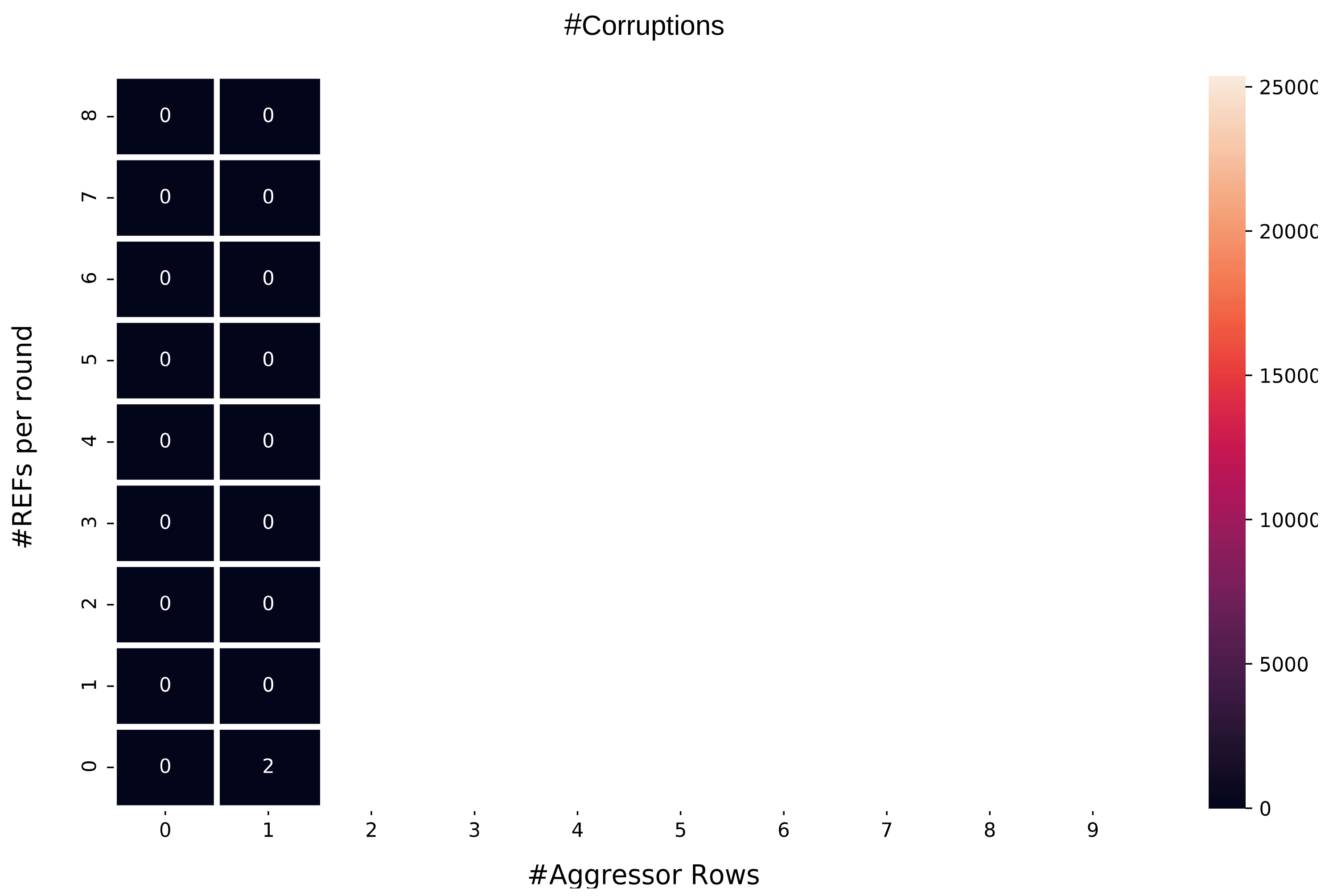
Case study: Vendor C

How big is the sampler?

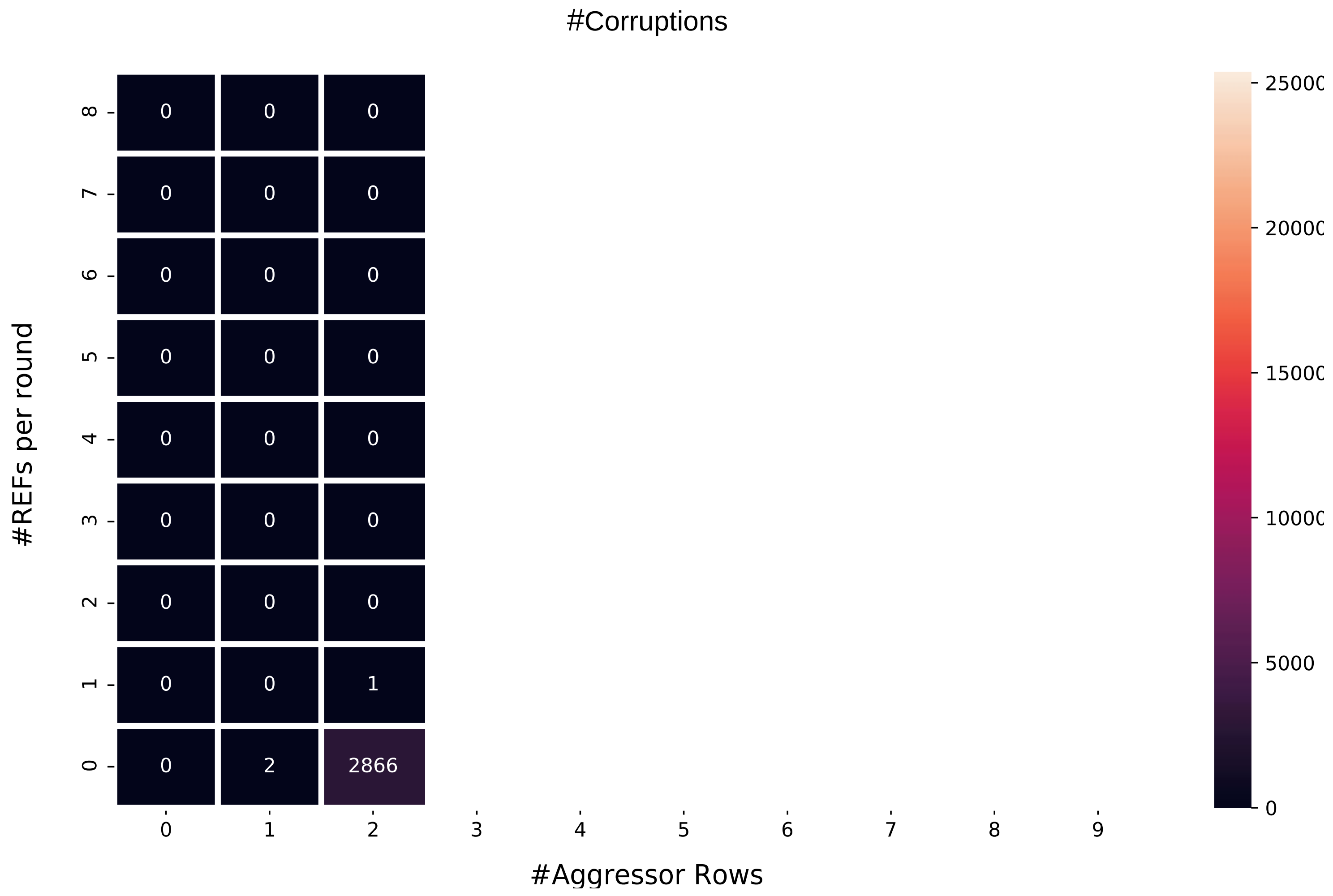
- Pick **N** aggressor rows
- Perform a series of hammers (activations of aggressors)
  - **8K activations**
- After each series of hammers, issue **R refreshes**
- **10 Rounds**



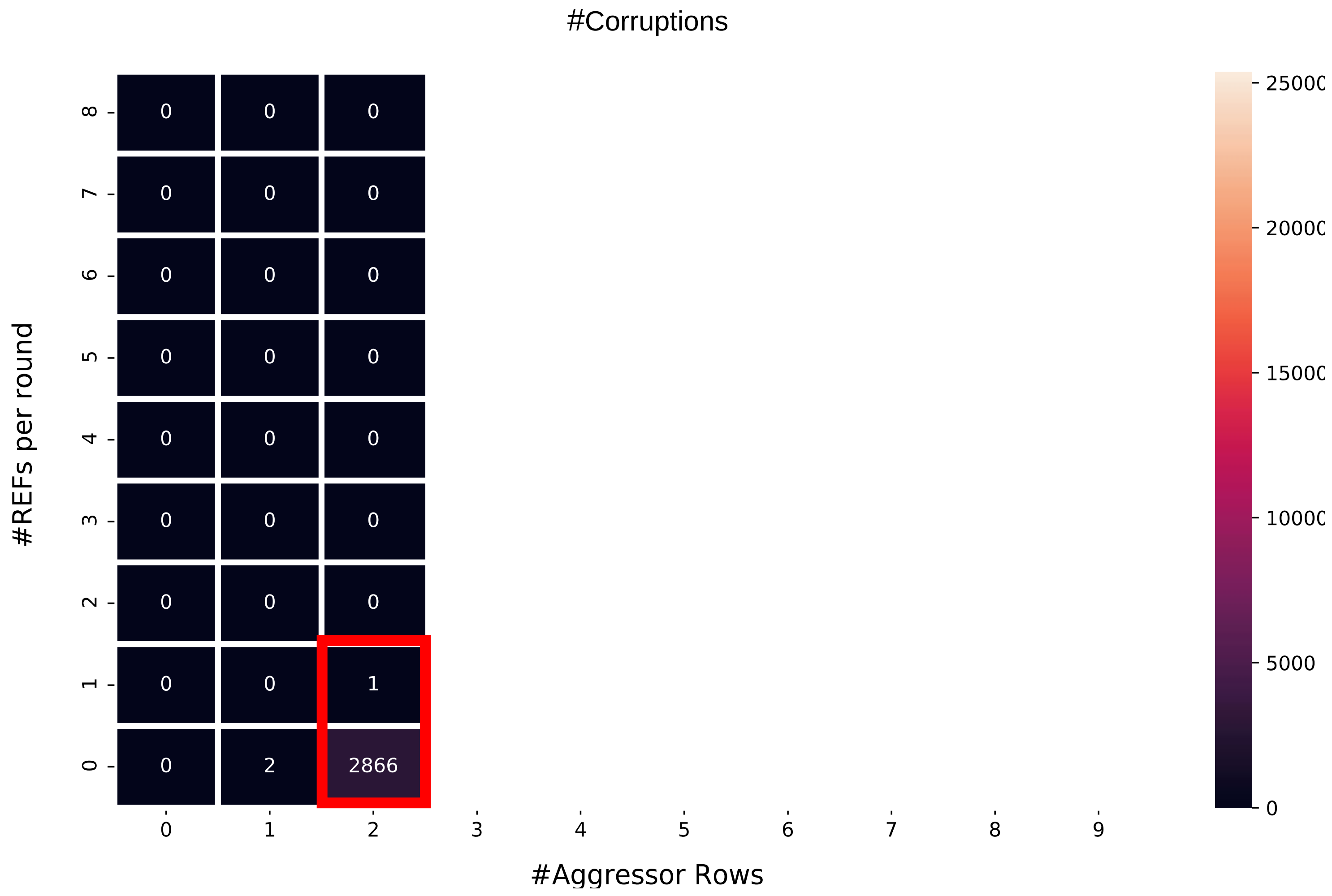
Case study: Vendor C



Case study: Vendor C



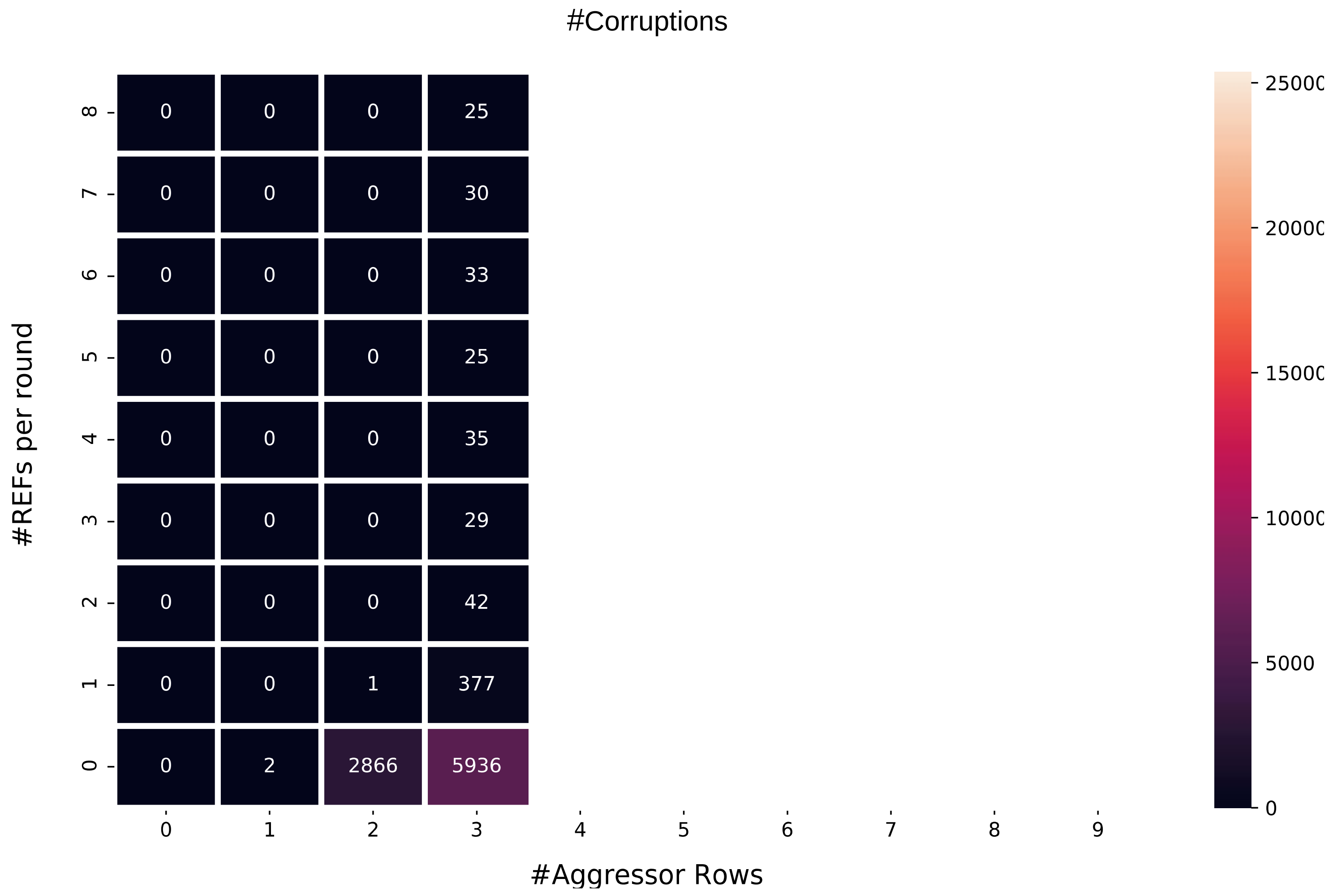
Case study: Vendor C



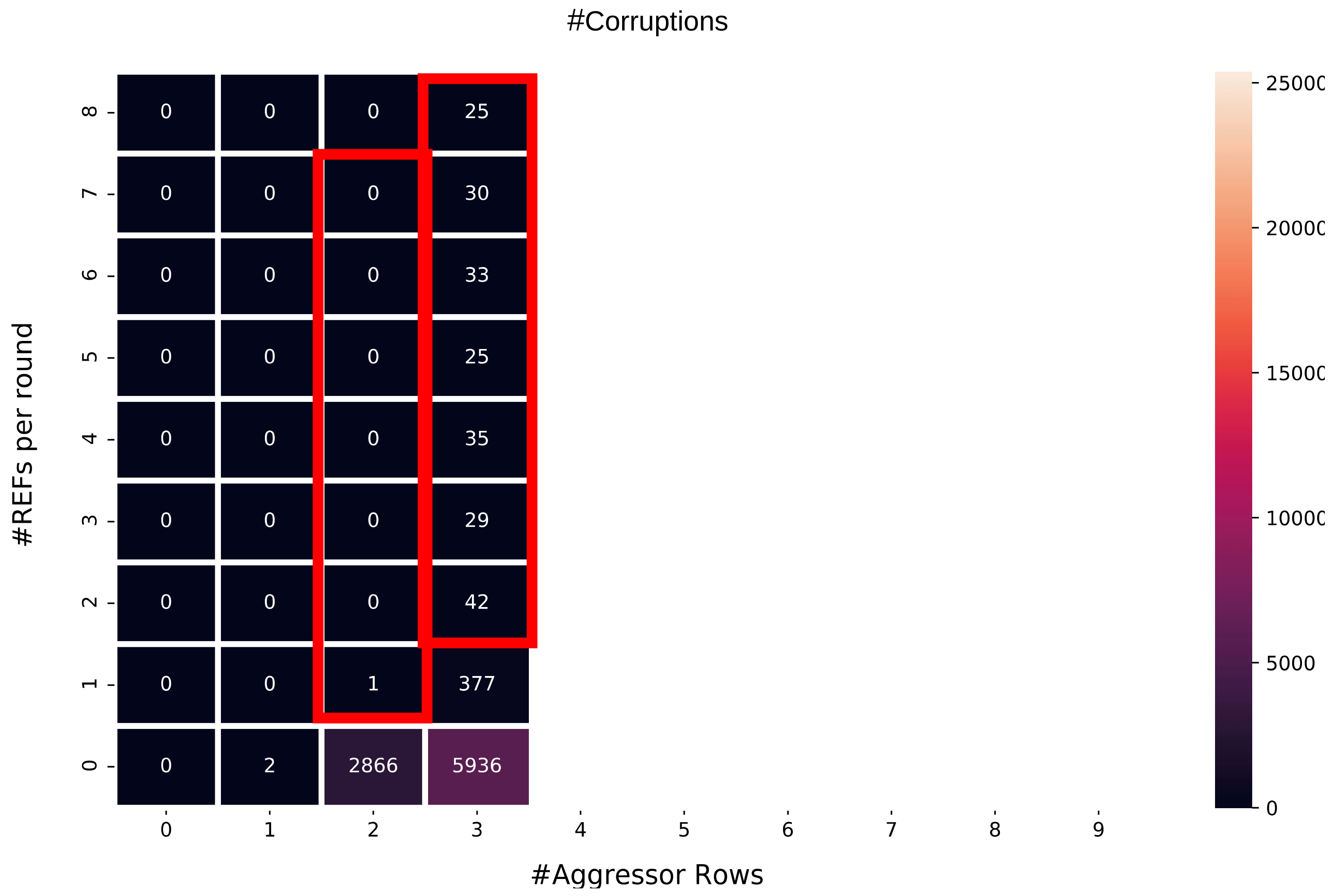
Case study: Observations

- The TRR mitigation acts on every refresh command

Case study: Vendor C



Case study: Vendor C

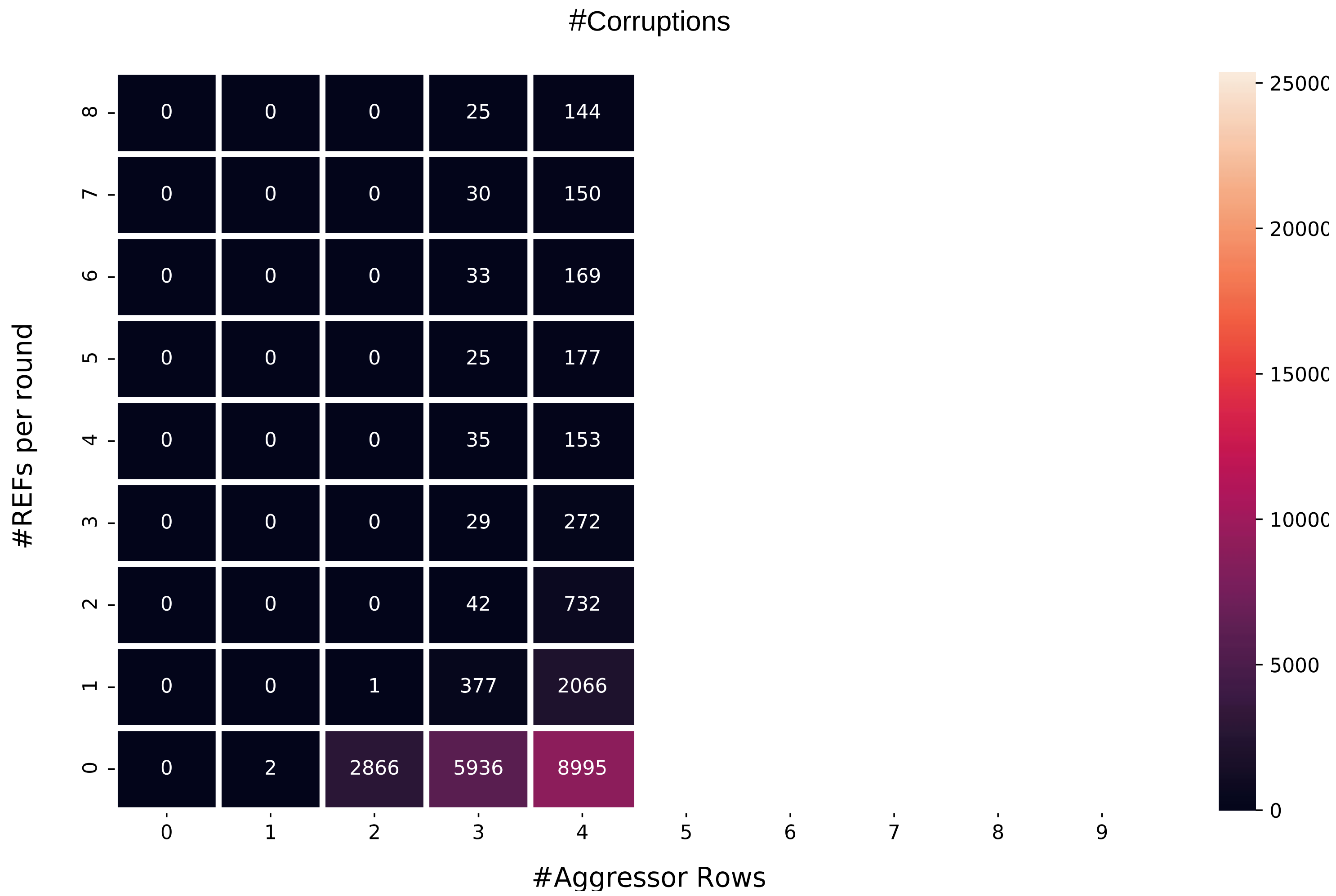


#REFs per round \ #Aggressor Rows	0	1	2	3
8	0	0	0	25
7	0	0	0	30
6	0	0	0	33
5	0	0	0	25
4	0	0	0	35
3	0	0	0	29
2	0	0	0	42
1	0	0	1	377
0	0	2	2866	5936

Case study: Observations

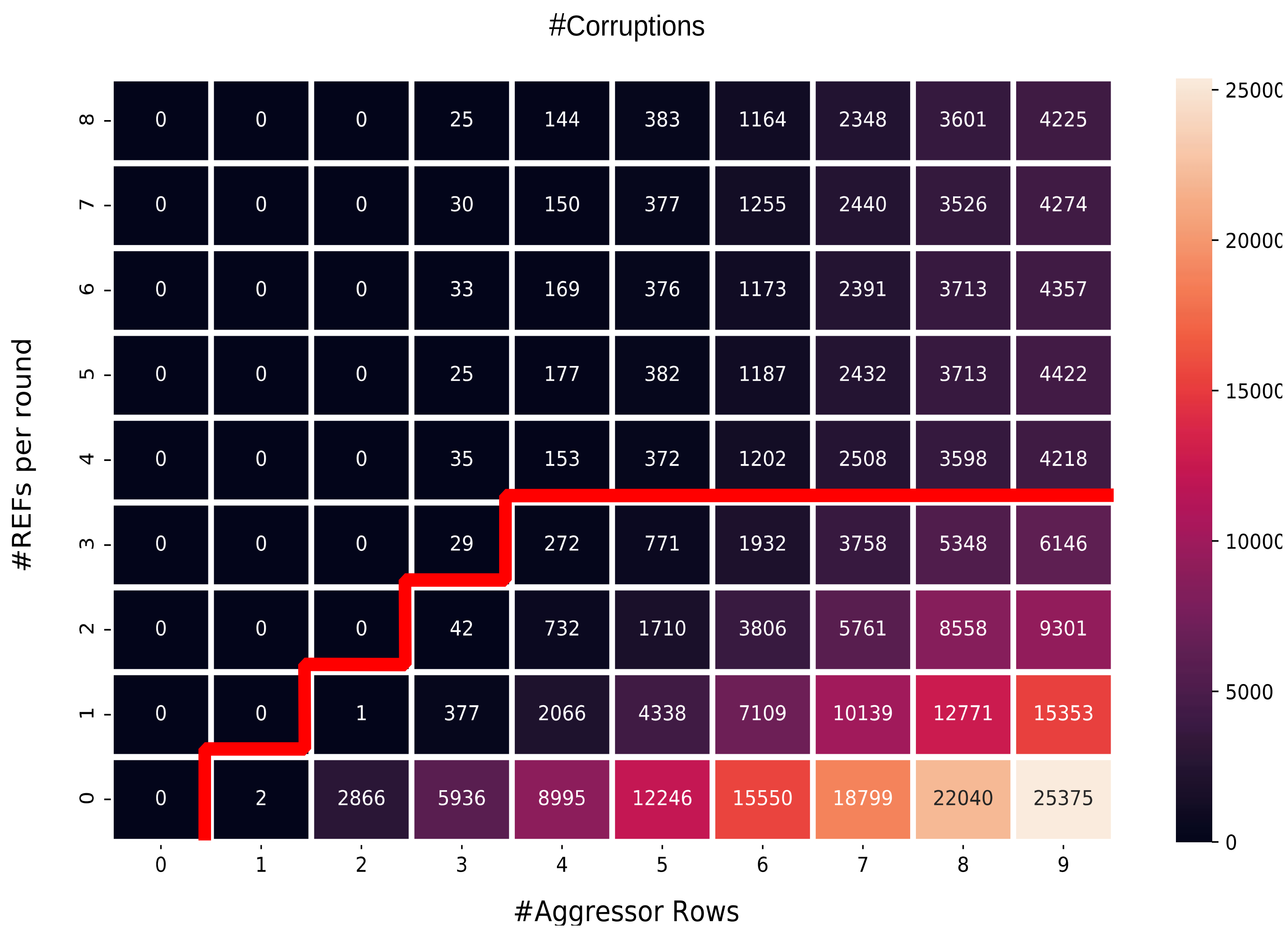
- The TRR mitigation acts on every refresh command
- The mitigation can sample more than one aggressor per refresh interval
- The mitigation can refresh only a single victim within a refresh operation

Case study: Vendor C



#REFs per round \ #Aggressor Rows	0	1	2	3	4
8	0	0	0	25	144
7	0	0	0	30	150
6	0	0	0	33	169
5	0	0	0	25	177
4	0	0	0	35	153
3	0	0	0	29	272
2	0	0	0	42	732
1	0	0	1	377	2066
0	0	2	2866	5936	8995

Case study: Vendor C



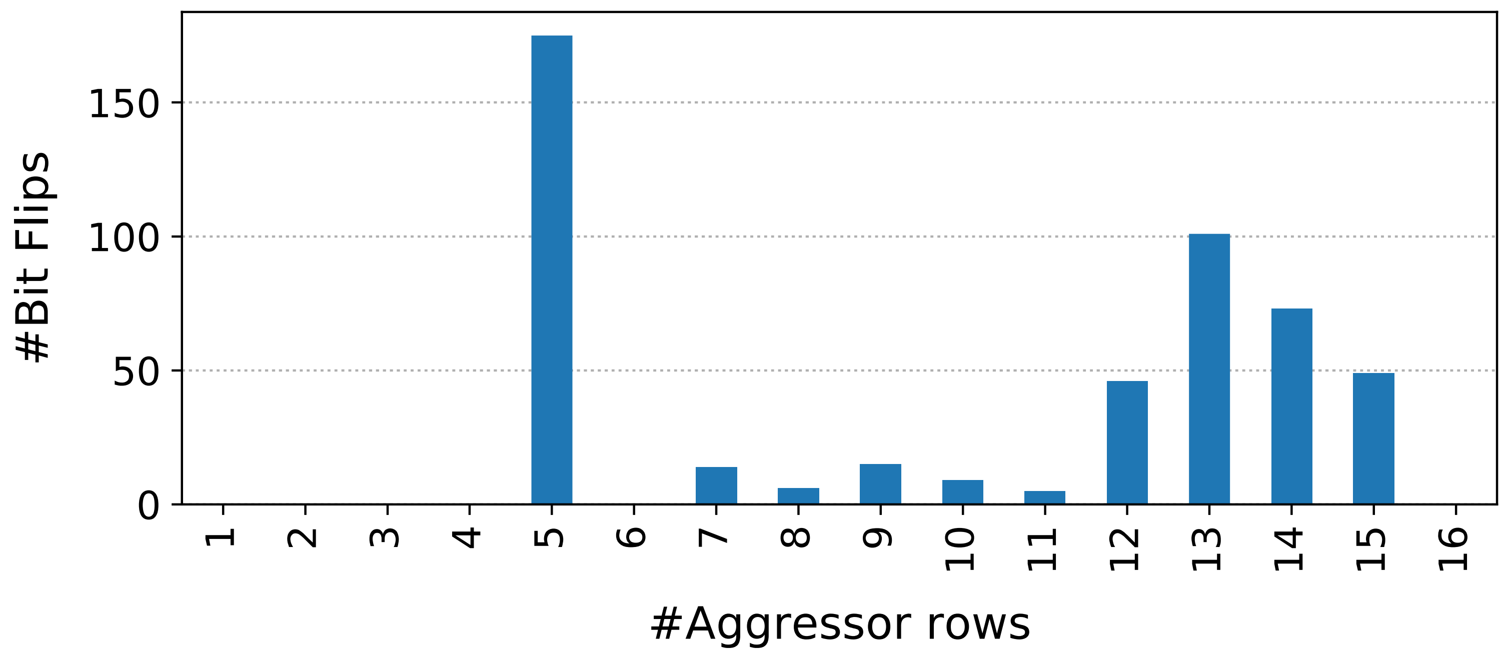
#REFs per round \ #Aggressor Rows	0	1	2	3	4	5	6	7	8	9
8	0	0	0	25	144	383	1164	2348	3601	4225
7	0	0	0	30	150	377	1255	2440	3526	4274
6	0	0	0	33	169	376	1173	2391	3713	4357
5	0	0	0	25	177	382	1187	2432	3713	4422
4	0	0	0	35	153	372	1202	2508	3598	4218
3	0	0	0	29	272	771	1932	3758	5348	6146
2	0	0	0	42	732	1710	3806	5761	8558	9301
1	0	0	1	377	2066	4338	7109	10139	12771	15353
0	0	2	2866	5936	8995	12246	15550	18799	22040	25375

Case study: Observations

- The TRR mitigation acts on every refresh command
- The mitigation can sample more than one aggressor per refresh interval
- The mitigation can refresh only a single victim within a refresh operation
- **Sweeping the number of refresh operations and aggressor rows while hammering reveals the sampler size**

Case study: Vendor C

with tREFi == 7.8µs



Case study: Observations

- The TRR mitigation acts on every refresh command
- The mitigation can sample more than one aggressor per refresh interval
- The mitigation can refresh only a single victim within a refresh operation
- **Sweeping the number of refresh operations and aggressor rows while hammering reveals the sampler size**
- The sampling mechanism is affected by the addresses of aggressor rows

TRRespass: The RowFuzzer

- Black-box fuzzing for RowHammer
  - Ignore the MC optimizations
  - Scalable approach for testing

- The sampler can track a limited number of aggressor rows
  - # Aggressors
- The sampler design may be row address dependent
  - Aggressor Location

TRRespass: Results

- **42** DIMMS from **3 of the major vendors**: Samsung, Micron, SK Hynix
  - 95% of the market
- Testing 256MB of contiguous memory against the best pattern
- **13** DIMMs with bit flips
  - Multiple effective patterns for each of them
- Bit flips with **double refresh**
- Fuzzing is effective.
  - How to Improve? Parameter selection.

Exploitation

- Memory templating
  - Find the right hammering pattern
  - Locations of aggressors not always fundamental
- Bit flips are repeatable
  - Spurious flips
- We demonstrate the feasibility of 3 example attacks:
  - Privilege escalation [1]
  - Access to co-hosted VM via RSA key corruption [2]
  - Sudo exploit: opcode flipping [3]

[1] *M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," in Black Hat USA, 2015*
[2] *K. Razavi et al., "Flip Feng Shui: Hammering a Needle in the Software Stack," in USENIX Sec., 2016*
[3] *D. Gruss et al., "Another Flip in the Wall of Rowhammer Defenses," in S&P, 2018.*

Conclusion

- Bit flips with more than 20 aggressor rows!
  - DDR4 devices are much more vulnerable than DDR3
  - Bit flips with less than 50K activations
- Fuzzing can help in memory testing
  - Reverse engineering to find meaningful parameters
- RowHammer is still a serious problem
- No prompt mitigations available

Questions!