P&S SoftMC

Understanding and Improving Modern DRAM Performance, Reliability, and Security with Hands-On Experiments

Hasan Hassan

Prof. Onur Mutlu

ETH Zürich

Spring 2022

22 March 2022

U-TRR

Uncovering in-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications

Hasan Hassan

Yahya Can Tugrul *Jeremie S. Kim* *Victor van der Veen*
Kaveh Razavi *Onur Mutlu*

ETH zürich TOBB ETÜ University of Economics & Technology Qualcomm

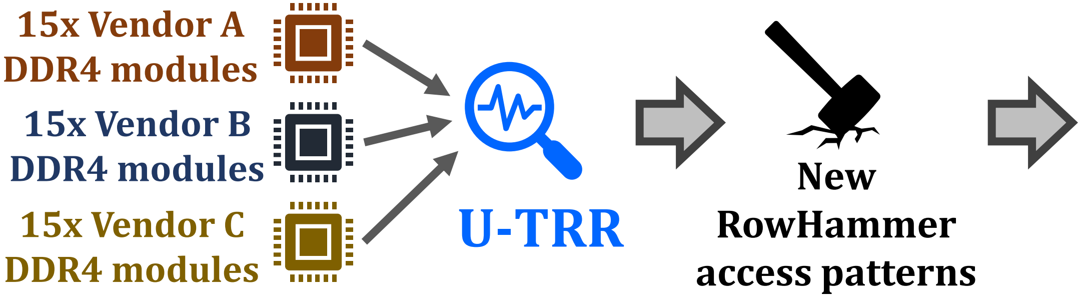
Summary

DRAM **RowHammer** vulnerability leads to critical reliability and security issues

Target Row Refresh (TRR):
a set of obscure, undocumented, and proprietary RowHammer mitigation techniques

Is TRR fully secure? How can we validate its security guarantees?

U-TRR	A new methodology that leverages *data retention failures* to uncover the inner workings of TRR and study its security
High-Level Operation	1) Profile the retention time of a row R 2) Find when TRR refreshes R to understand the underlying TRR mechanism



15x Vendor A DDR4 modules
15x Vendor B DDR4 modules
15x Vendor C DDR4 modules
U-TRR
New RowHammer access patterns

All 45 modules we test are **vulnerable**

99.9% of rows in a DRAM bank experience **at least one RowHammer bit flip**

Up to **7** RowHammer **bit flips** in an 8-byte dataword, **making ECC ineffective**

U-TRR can enable **more secure** RowHammer solutions

Outline

1. DRAM Operation Basics

2. RowHammer & Target Row Refresh

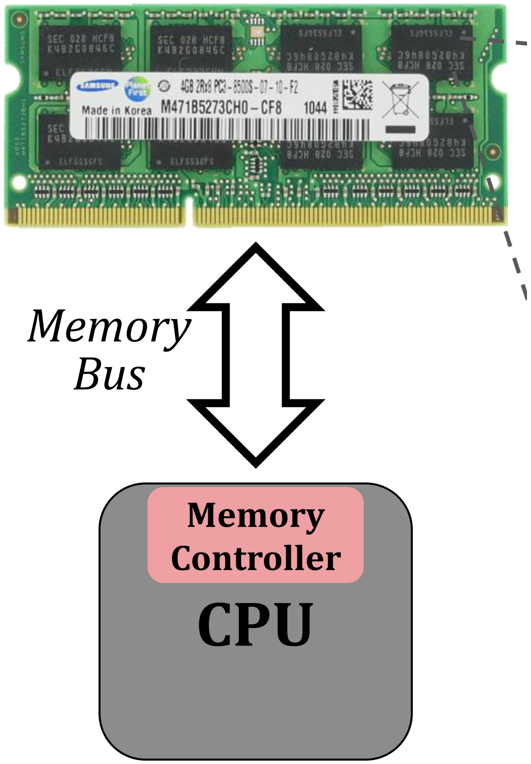
3. The U-TRR Methodology

4. Observations & New RowHammer Access Patterns

5. RowHammer Bit Flip Analysis

6. Takeaways and Conclusion

DRAM Organization

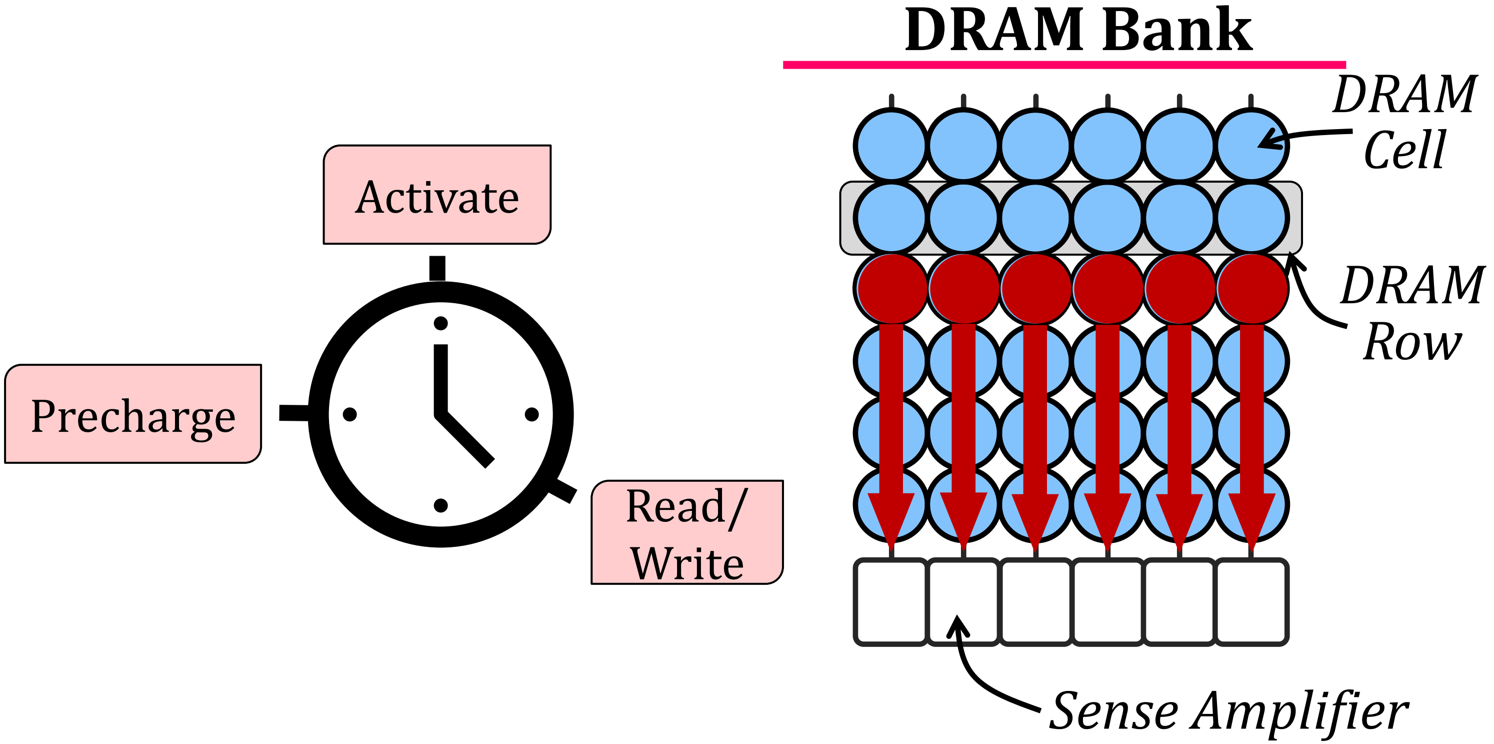


Memory Bus

Memory Controller

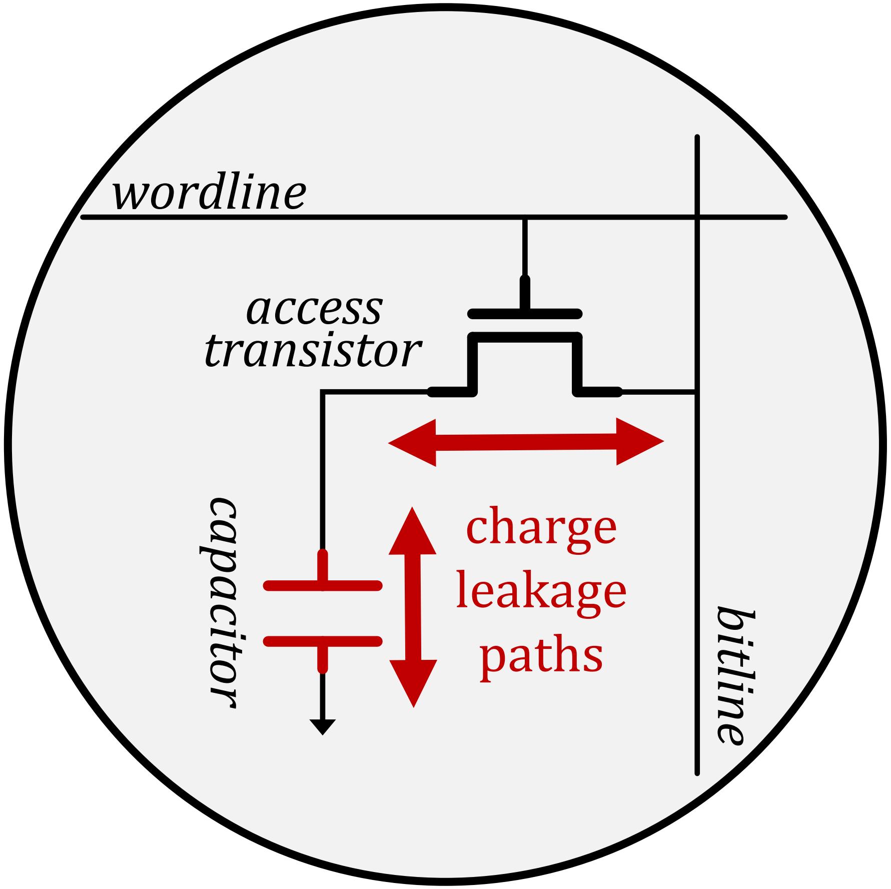
CPU

Accessing DRAM



DRAM Cell Leakage

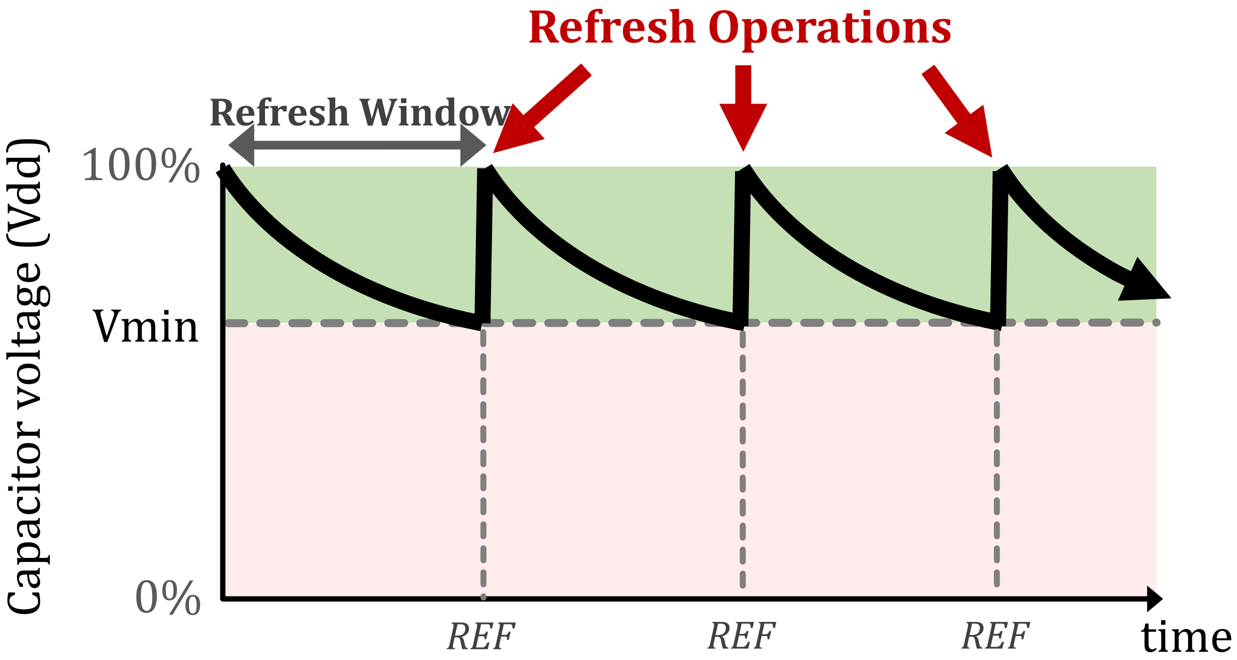
Each cell encodes information in **leaky** capacitors



Stored data is **corrupted** if too much charge leaks (i.e., the capacitor voltage degrades too much)

[Patel+, ISCA'17]

DRAM Refresh



Periodic **refresh operations** preserve stored data

Outline

1. DRAM Operation Basics

2. RowHammer & Target Row Refresh

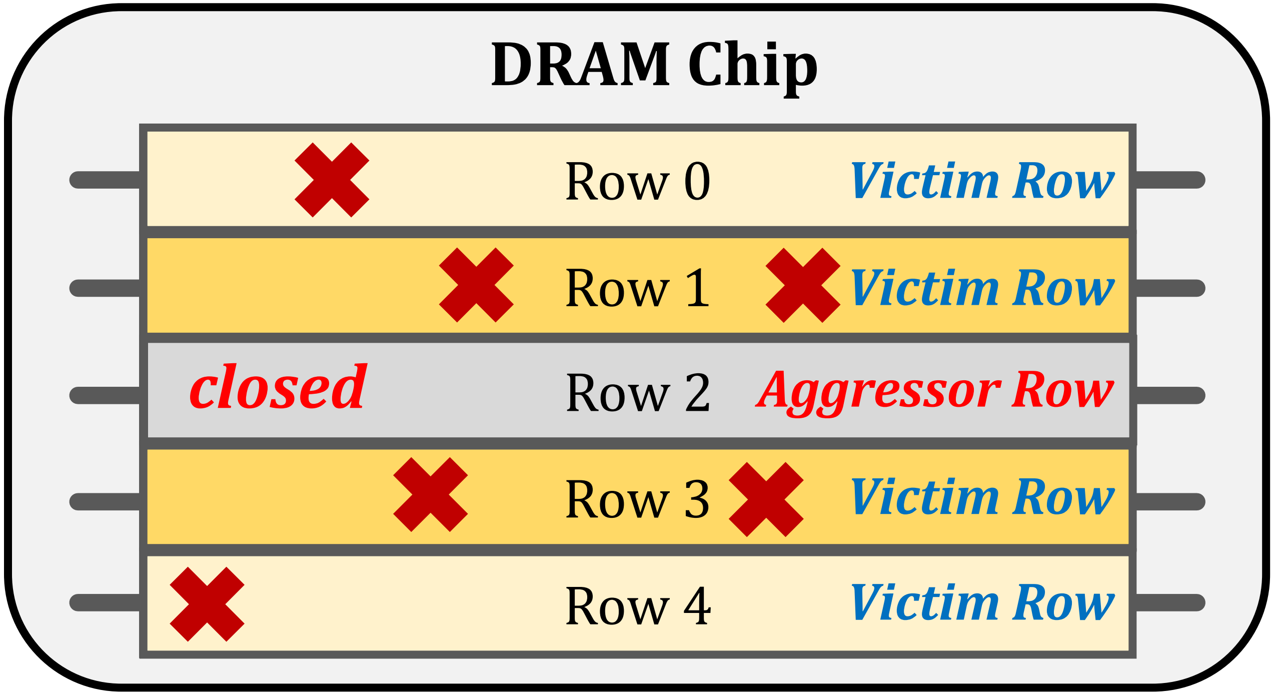
3. The U-TRR Methodology

4. Observations & New RowHammer Access Patterns

5. RowHammer Bit Flip Analysis

6. Takeaways and Conclusion

The RowHammer Vulnerability

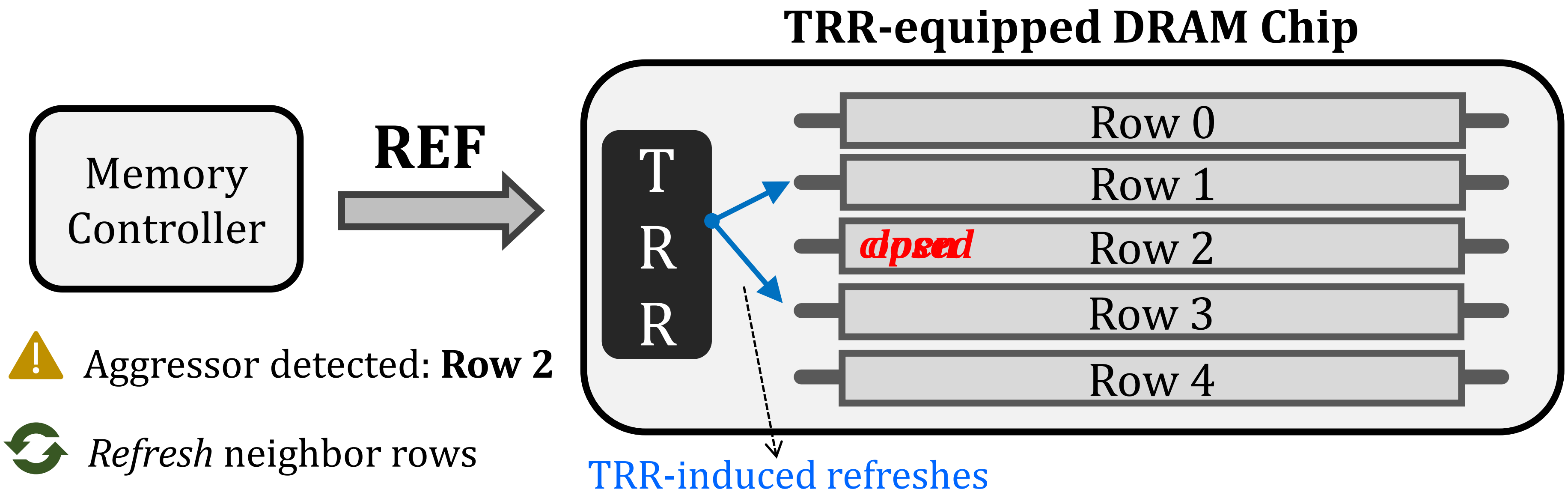


Repeatedly **opening** (activating) and **closing** (precharging) a DRAM row causes **RowHammer bit flips** in nearby cells

Target Row Refresh (TRR)

DRAM vendors equip their DRAM chips with a *proprietary* mitigation mechanisms known as **Target Row Refresh (TRR)**

Key Idea: TRR refreshes nearby rows upon detecting an aggressor row



The Problem with TRR

TRR is obscure, undocumented, and proprietary

We cannot easily study the *security properties* of TRR

Goal

Study in-DRAM TRR mechanisms to

1. **understand** how they operate

2. **assess** their security

3. **secure** DRAM completely against RowHammer

Outline

1. DRAM Operation Basics

2. RowHammer & Target Row Refresh

3. The U-TRR Methodology

4. Observations & New RowHammer Access Patterns

5. RowHammer Bit Flip Analysis

6. Takeaways and Conclusion

Overview of U-TRR

U-TRR: A new methodology to *uncover* the inner workings of TRR

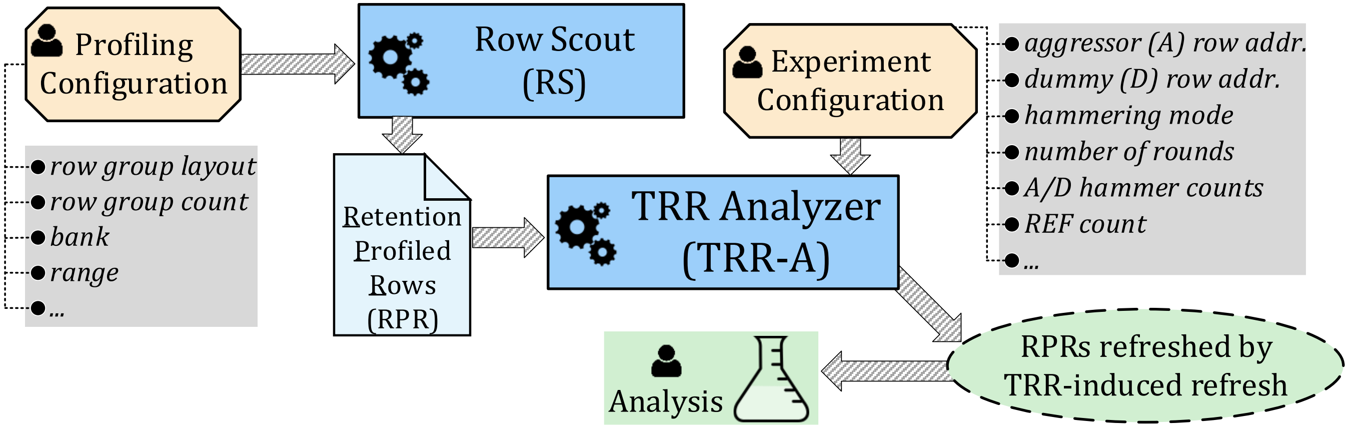
Key idea: Use data retention failures as a side channel to detect when a row is refreshed by TRR

High-Level U-TRR Operation

U-TRR has two main components:
Row Scout (RS) and **TRR Analyzer (TRR-A)**

Row Scout: finds a **set of DRAM rows** that meet certain requirements as needed by TRR-A and **identifies the data retention times** of these rows

TRR Analyzer: uses RS-provided rows to **distinguish between TRR-induced and regular refreshes**, and thus builds an understanding of the underlying TRR mechanism



Row Scout (RS)

Goal: Identify a list of *useful* DRAM rows and their *retention times*

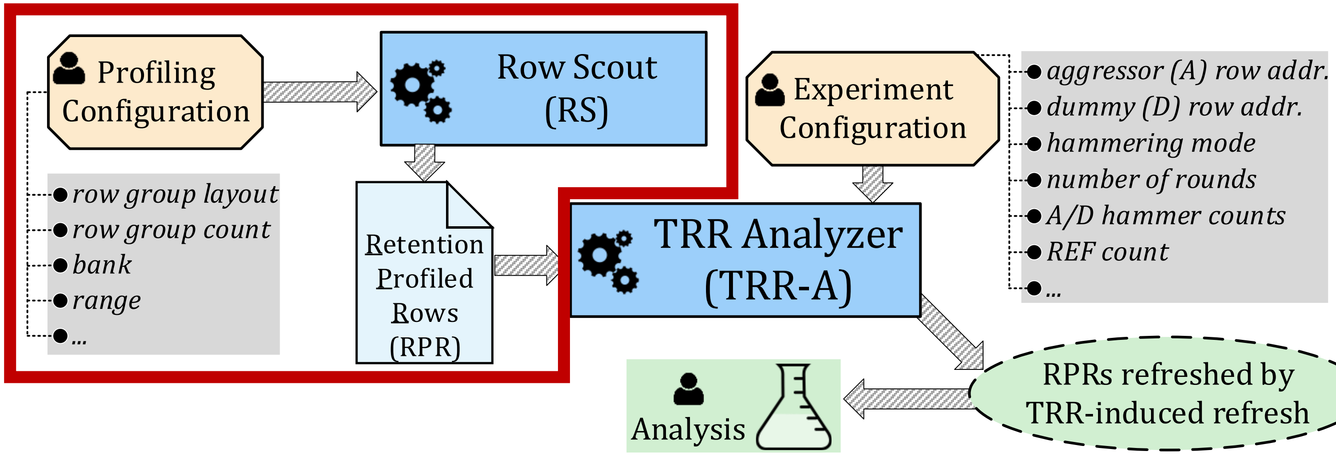
Row Scout **must** find:

✓ Rows with **consistent*** retention times

- ➢ To correctly infer whether a row has been refreshed

✓ **Multiple rows** that are located at *certain configurable distances* and have the *same retention time (i.e., Row Group)*

- ➢ To observe whether TRR can refresh multiple rows at the same time

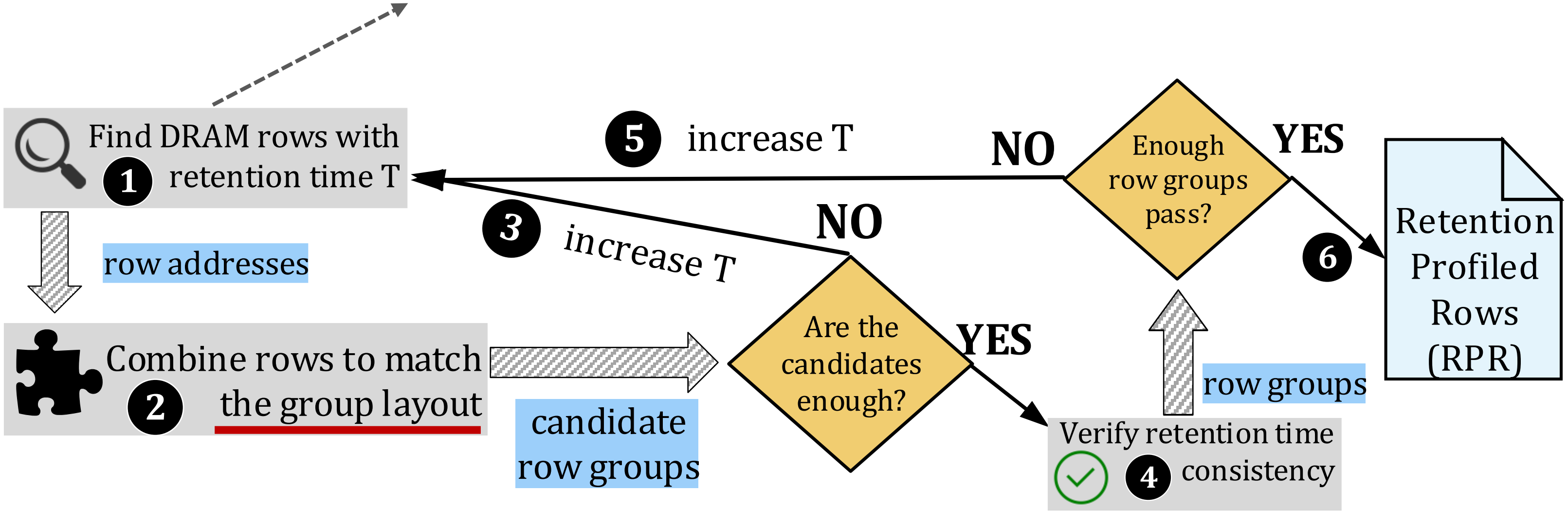


* *The retention time of a DRAM row may change over time due to Variable Retention Time (VRT) effects*

Row Scout (RS) Operation

Profiling the **retention time** of a DRAM row:

1) write data
2) wait for T
3) check for retention bit flips



1 Find DRAM rows with retention time T

row addresses

2 Combine rows to match the group layout

candidate row groups

Are the candidates enough?

NO

3 increase T

YES

4 Verify retention time consistency

row groups

Enough row groups pass?

NO

5 increase T

YES

6

Retention Profiled Rows (RPR)

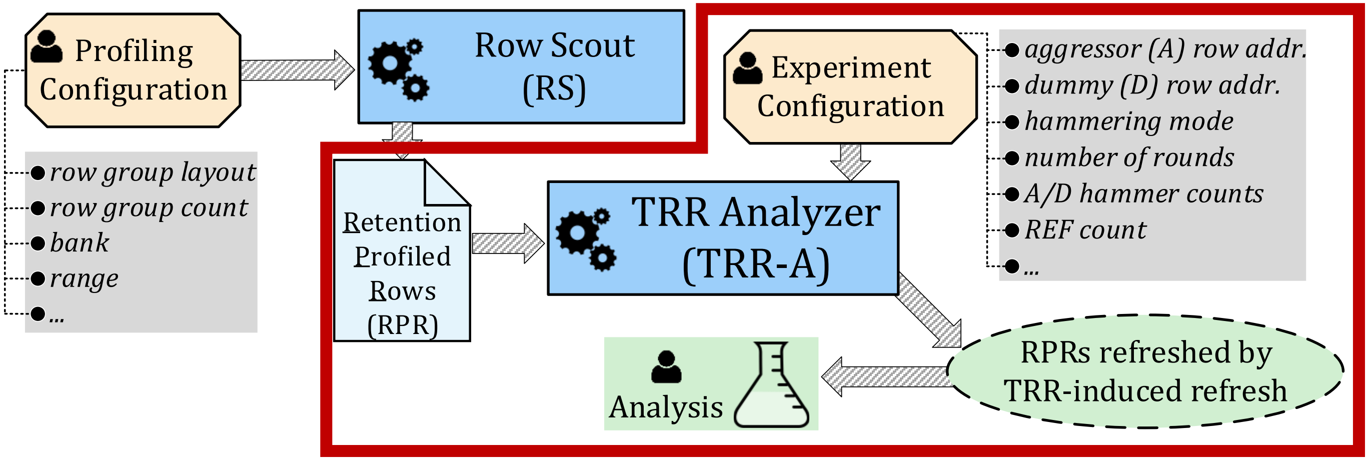
Row Group: V □ V □ V

TRR Analyzer (TRR-A)

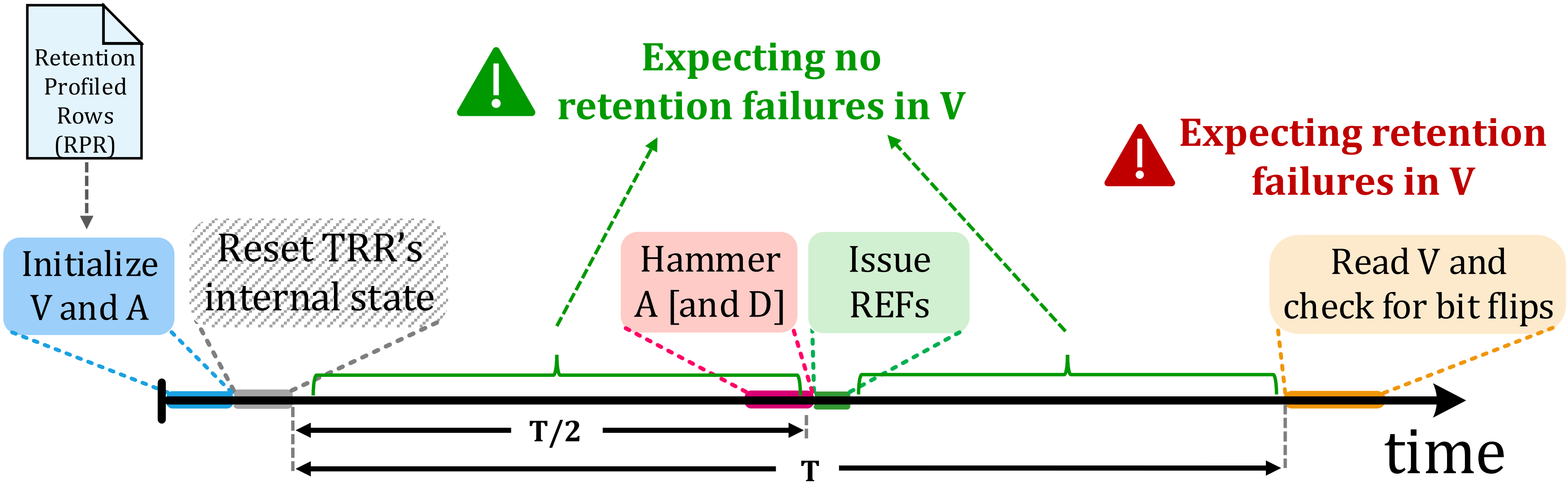
Goal: Use RS-provided rows to determine when TRR refreshes a victim row

High-level Operation:

1) Run a certain DRAM access pattern (i.e., RowHammer attack)
2) Monitor retention failures in RS-provided rows to determine when TRR refreshes any of these rows
3) Develop an understanding of the underlying TRR operation



TRR Analyzer (TRR-A) Operation


Retention Profiled Rows (RPR)
Initialize V and A
Reset TRR's internal state
Expecting no retention failures in V
Hammer A [and D]
Issue REFs
Expecting retention failures in V
Read V and check for bit flips
T/2
T
time

Row Group: **V** **A** **V** **A** **V**

V: victim (RS-provided) rows
A: aggressor rows
D: dummy rows

Experiment Configuration

- *aggressor (A) row addr.*
- *dummy (D) row count*
- *hammering mode*
- *number of rounds*
- *A/D hammer counts*
- *REF count*

TRR-A helps to understand how TRR operates based on when Retention Profiled Rows are refreshed by TRR

Outline

1. DRAM Operation Basics

2. RowHammer & Target Row Refresh

3. The U-TRR Methodology

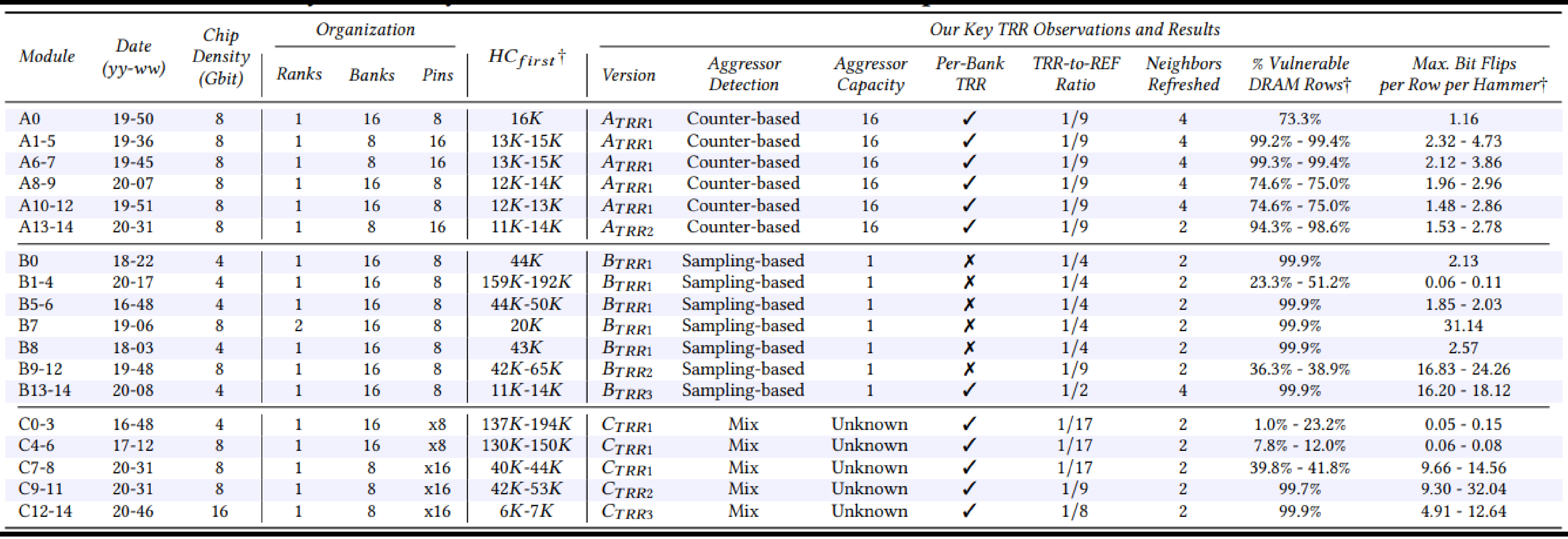
4. Observations & New RowHammer Access Patterns

5. RowHammer Bit Flip Analysis

6. Takeaways and Conclusion

DRAM Testing Infrastructure

We implement U-TRR using FPGA-based *SoftMC* [Hassan+, HPCA'18] *modified to support DDR4 DRAM*



Module	Date (yy-ww)	Chip Density (Gbit)	Organization			HC_{first}†	Our Key TRR Observations and Results							
			Ranks	Banks	Pins		Version	Aggressor Detection	Aggressor Capacity	Per-Bank TRR	TRR-to-REF Ratio	Neighbors Refreshed	% Vulnerable DRAM Rows†	Max. Bit Flips per Row per Hammer†
A0	19-50	8	1	16	8	16K	A_{TRR1}	Counter-based	16	✓	1/9	4	73.3%	1.16
A1-5	19-36	8	1	8	16	13K-15K	A_{TRR1}	Counter-based	16	✓	1/9	4	99.2% - 99.4%	2.32 - 4.73
A6-7	19-45	8	1	8	16	13K-15K	A_{TRR1}	Counter-based	16	✓	1/9	4	99.3% - 99.4%	2.12 - 3.86
A8-9	20-07	8	1	16	8	12K-14K	A_{TRR1}	Counter-based	16	✓	1/9	4	74.6% - 75.0%	1.96 - 2.96
A10-12	19-51	8	1	16	8	12K-13K	A_{TRR1}	Counter-based	16	✓	1/9	4	74.6% - 75.0%	1.48 - 2.86
A13-14	20-31	8	1	8	16	11K-14K	A_{TRR2}	Counter-based	16	✓	1/9	2	94.3% - 98.6%	1.53 - 2.78
B0	18-22	4	1	16	8	44K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	2.13
B1-4	20-17	4	1	16	8	159K-192K	B_{TRR1}	Sampling-based	1	✗	1/4	2	23.3% - 51.2%	0.06 - 0.11
B5-6	16-48	4	1	16	8	44K-50K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	1.85 - 2.03
B7	19-06	8	2	16	8	20K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	31.14
B8	18-03	4	1	16	8	43K	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	2.57
B9-12	19-48	8	1	16	8	42K-65K	B_{TRR2}	Sampling-based	1	✗	1/9	2	36.3% - 38.9%	16.83 - 24.26
B13-14	20-08	4	1	16	8	11K-14K	B_{TRR3}	Sampling-based	1	✓	1/2	4	99.9%	16.20 - 18.12
C0-3	16-48	4	1	16	x8	137K-194K	C_{TRR1}	Mix	Unknown	✓	1/17	2	1.0% - 23.2%	0.05 - 0.15
C4-6	17-12	8	1	16	x8	130K-150K	C_{TRR1}	Mix	Unknown	✓	1/17	2	7.8% - 12.0%	0.06 - 0.08
C7-8	20-31	8	1	8	x16	40K-44K	C_{TRR1}	Mix	Unknown	✓	1/17	2	39.8% - 41.8%	9.66 - 14.56
C9-11	20-31	8	1	8	x16	42K-53K	C_{TRR2}	Mix	Unknown	✓	1/9	2	99.7%	9.30 - 32.04
C12-14	20-46	16	1	8	x16	6K-7K	C_{TRR3}	Mix	Unknown	✓	1/8	2	99.9%	4.91 - 12.64

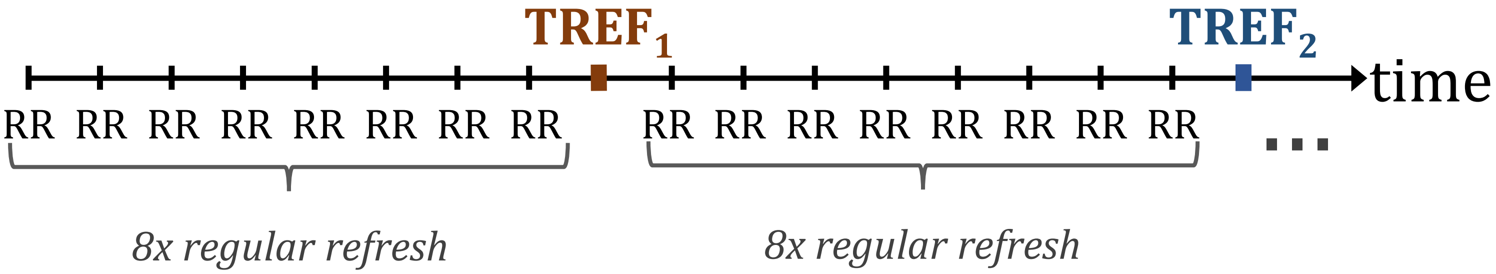
Table 1 in our paper provides more information about the analyzed modules

15x Vendor C DDR4 modules

Key Observations: Vendor A

Refresh Types:

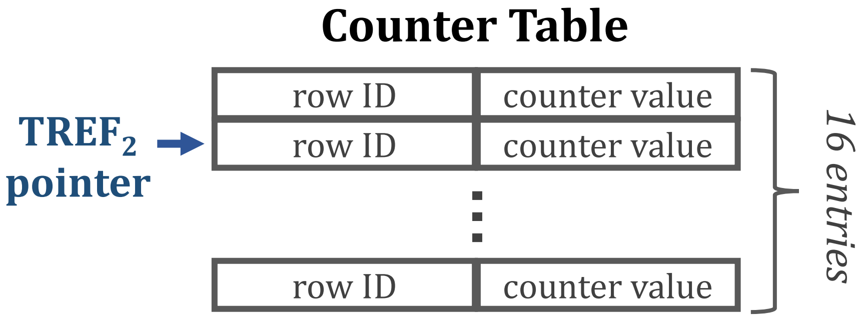
- Regular Refresh (RR)
- TRR-capable Refresh (**$TREF_1$** and **$TREF_2$**)



Observation: TRR tracks potentially aggressor rows using a Counter Table

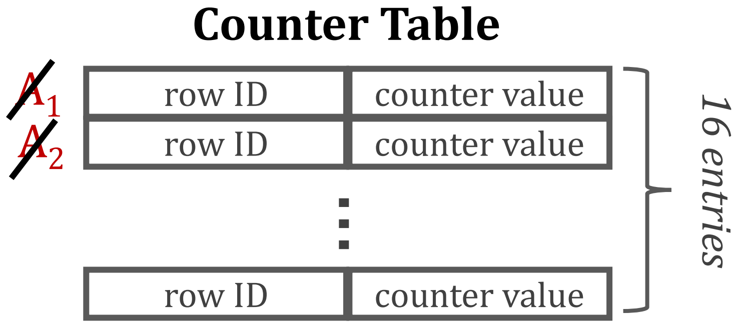
$TREF_1$: Refreshes the victims of **row ID** with the **largest counter value**

$TREF_2$: Refreshes the victims of **row ID** that $TREF_2$ pointer refers to



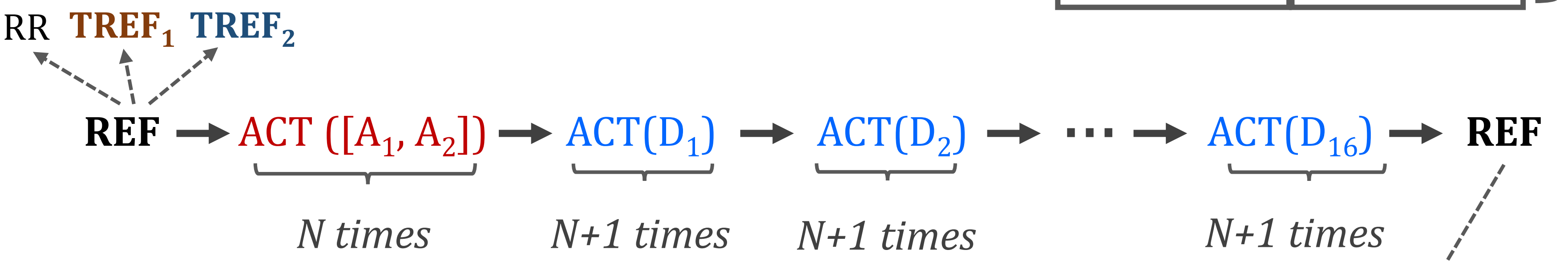
Circumventing Vendor A’s TRR

Approach: Ensure an aggressor row is **discarded** from the *Counter Table* **prior** to a REF command



Counter Table

row ID	counter value
row ID	counter value
⋮	⋮
row ID	counter value



$$\textbf{REF} \rightarrow \underbrace{\text{ACT }([A_1, A_2])}_{N\ times} \rightarrow \underbrace{\text{ACT}(D_1)}_{N+1\ times} \rightarrow \underbrace{\text{ACT}(D_2)}_{N+1\ times} \rightarrow \cdots \rightarrow \underbrace{\text{ACT}(D_{16})}_{N+1\ times} \rightarrow \textbf{REF}$$

A_i: aggressor row
D_i: dummy row

[A1, A2] **not** refreshed by TRR

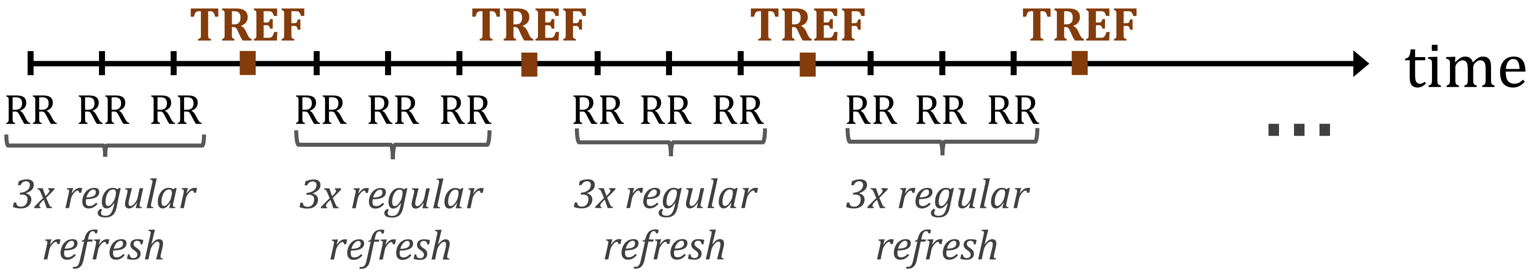
This RowHammer access pattern requires **synchronizing** accesses with REF commands

Circumventing Vendor A’s TRR by discarding the actual aggressor rows from the Counter Table

Key Observations: Vendor B

Refresh Types:

- Regular Refresh (RR)
- TRR-capable Refresh (**TREF**)



Observation 1: TRR *probabilistically* samples the address of an activated row

Observation 2: A newly-sampled row overwrites the previously-sampled one

TREF: Refreshes the victims of the **last sampled row**

Circumventing Vendor B's TRR

Approach: Maximize the **dummy** row hammers **after** hammering the **aggressor** rows and **before** the next **TREF**

$$\text{TREF} \rightarrow \underbrace{\text{ACT}([A_1, A_2])}_{N\ times} \rightarrow \underbrace{\text{ACT}(D_1)}_{M\ times} \rightarrow \text{TREF}$$

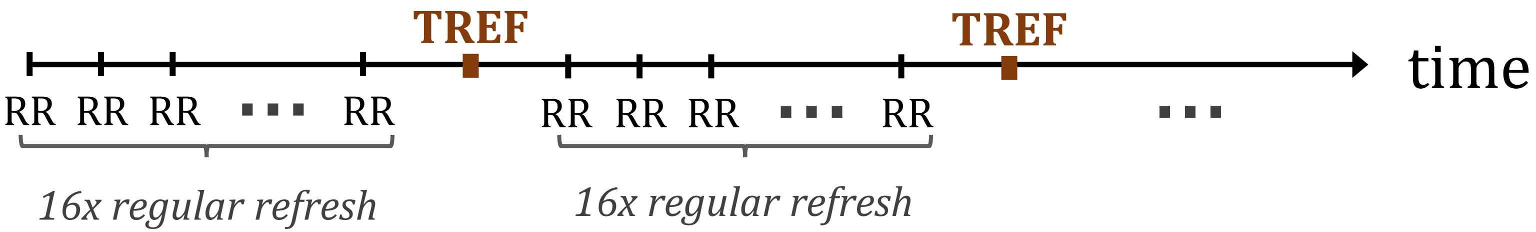
[A1, A2] **not** refreshed by TRR

Circumventing Vendor B's TRR by making it replace a sampled aggressor row by sampling **a dummy row**

Key Observations: Vendor C

Refresh Types:

- Regular Refresh (RR)
- TRR-capable Refresh (**TREF**)



Observation 1: TRR detects an aggressor row only among the first 2K ACT commands issued after a **TREF**

Observation 2: Rows activated earlier within the 2K ACT commands are more likely to be detected by TRR

TREF: Detects an aggressor row only among the first 2K ACT commands while favoring the earlier activations more

Circumventing Vendor C's TRR

Approach: Hammer dummy rows before aggressor rows to **maximize the probability** of TRR **detecting** a dummy row

$$\mathbf{TREF} \rightarrow \underbrace{\mathrm{ACT}(D_1)}_{\textit{N times}} \rightarrow \underbrace{\mathrm{ACT}([A_1, A_2])}_{\textit{M times}} \rightarrow \mathbf{TREF}$$

[A1, A2] **not** refreshed by TRR

Circumventing Vendor C's TRR by first hammering dummy rows to make aggressor rows less likely to be detected

Outline

1. DRAM Operation Basics

2. RowHammer & Target Row Refresh

3. The U-TRR Methodology

4. Observations & New RowHammer Access Patterns

5. RowHammer Bit Flip Analysis

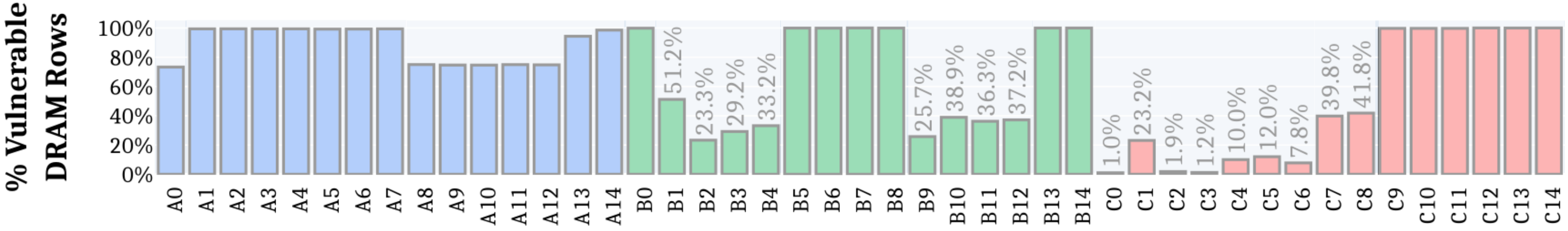
6. Takeaways and Conclusion

Bypassing TRR with New RowHammer Access Patterns

We craft **new RowHammer access patterns** that circumvent TRR of three major DRAM vendors

On the **45** DDR4 modules we test, the new access patterns cause a large number of RowHammer bit flips

Effect on Individual Rows



All 45 modules we tested are vulnerable
to our new RowHammer access patterns

Our RowHammer access patterns
cause bit flips in more than 99.9% of the rows

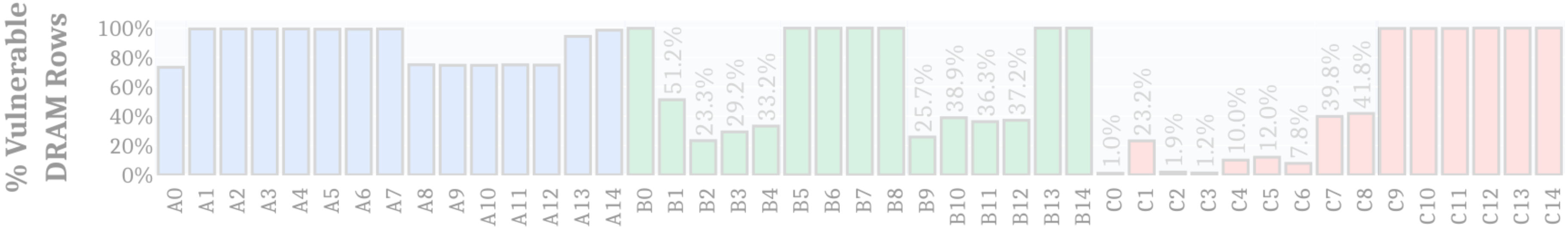
Why are some modules less vulnerable?

1) Fundamentally less vulnerable to RowHammer

2) Different TRR mechanisms

3) Unique row organization

Effect on Individual Rows



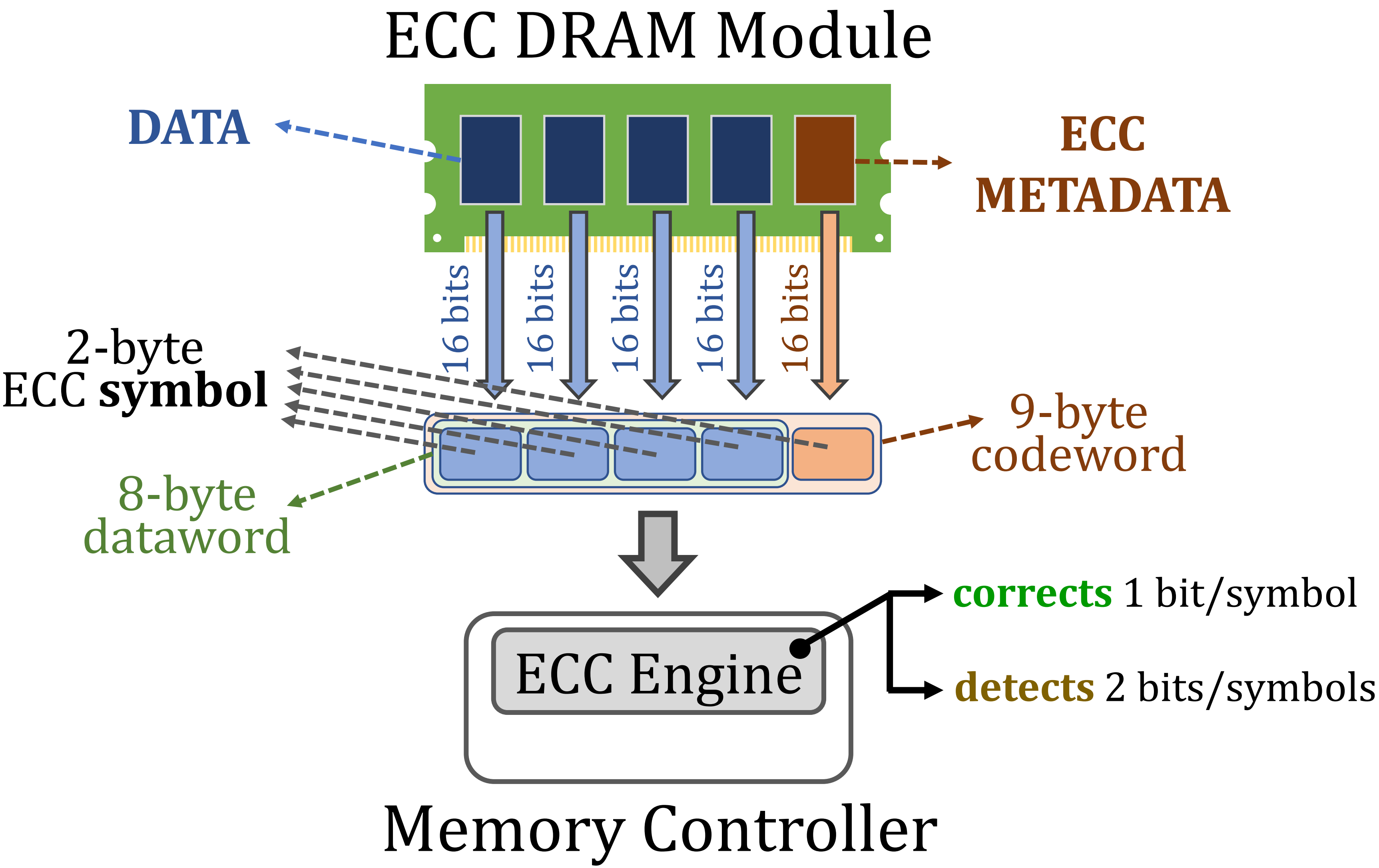
All 45 modules we tested are vulnerable
to our new RowHammer access patterns

Our RowHammer access patterns
cause bit flips in more than 99.9% of the rows

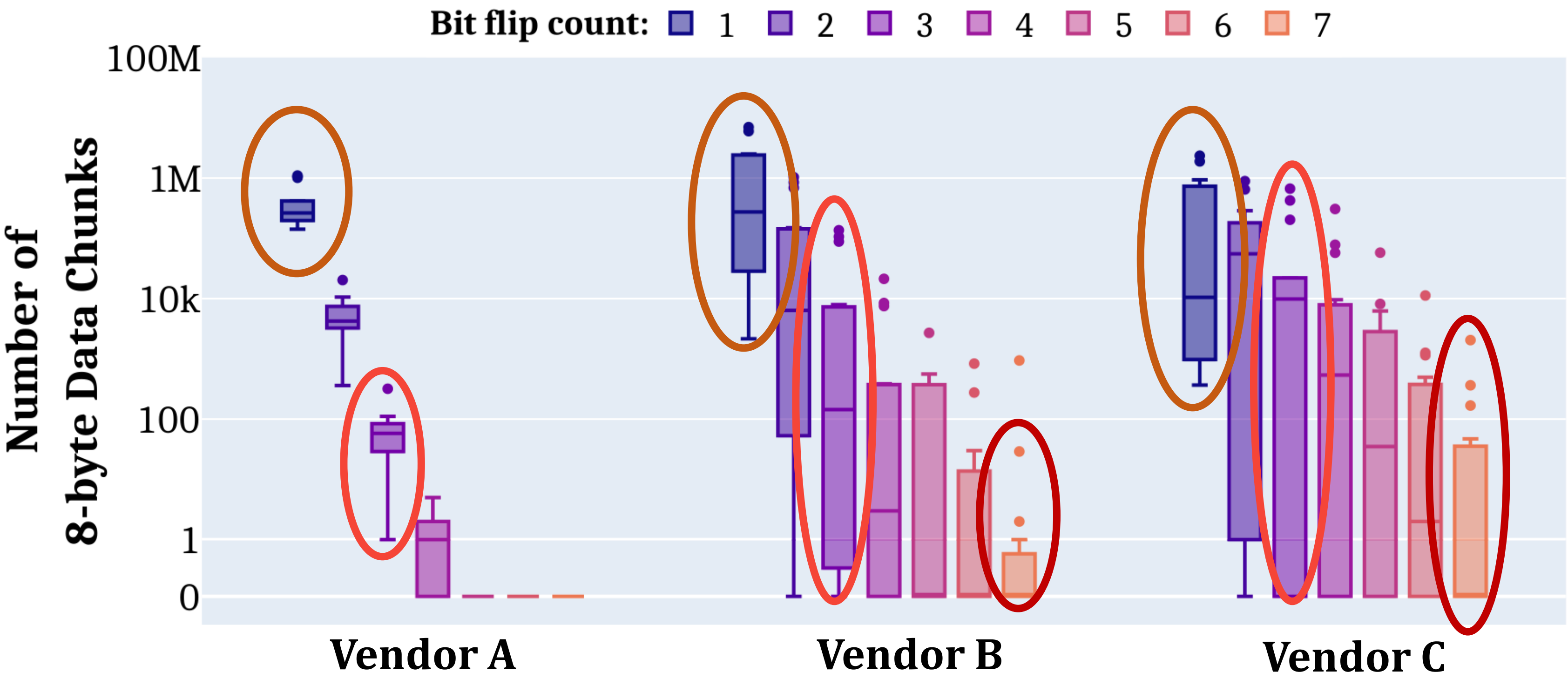
Our access patterns successfully **circumvent** the TRR implementations of **all three major DRAM vendors**

3) Unique row organization

Can ECC Protect Against Our Access Patterns?



Bypassing ECC with New RowHammer Patterns

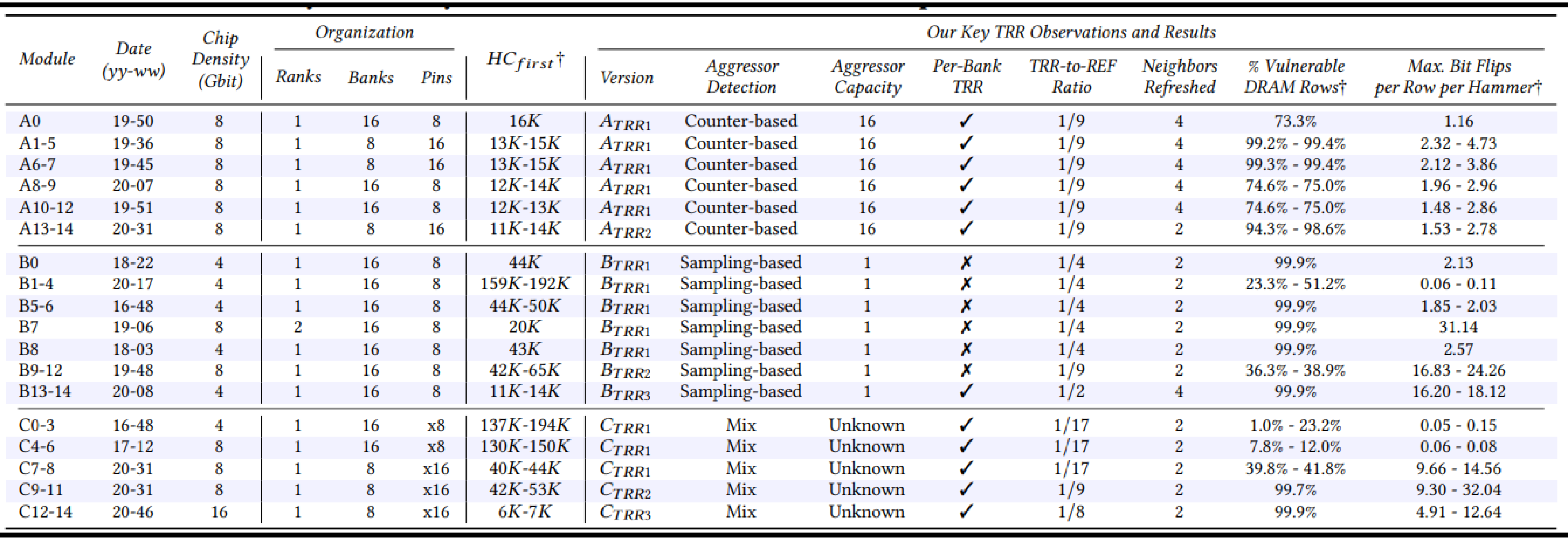


Modules from all three vendors have many **8-byte data chunks** with 3 and more (up to 7) RowHammer bit flips

Conventional DRAM ECC cannot protect against our new RowHammer access patterns

Other Observations and Results in the Paper

- More observations on the TRRs of the three vendors
- Detailed description of the crafted access patterns
- Hammers per aggressor row sensitivity analysis
- Observations and results for individual modules
- …



Module	Date (yy-ww)	Chip Density (Gbit)	Organization			HC_{first}†	Our Key TRR Observations and Results							
			Ranks	Banks	Pins		Version	Aggressor Detection	Aggressor Capacity	Per-Bank TRR	TRR-to-REF Ratio	Neighbors Refreshed	% Vulnerable DRAM Rows†	Max. Bit Flips per Row per Hammer†
A0	19-50	8	1	16	8	$16K$	A_{TRR1}	Counter-based	16	✓	1/9	4	73.3%	1.16
A1-5	19-36	8	1	8	16	$13K$-$15K$	A_{TRR1}	Counter-based	16	✓	1/9	4	99.2% - 99.4%	2.32 - 4.73
A6-7	19-45	8	1	8	16	$13K$-$15K$	A_{TRR1}	Counter-based	16	✓	1/9	4	99.3% - 99.4%	2.12 - 3.86
A8-9	20-07	8	1	16	8	$12K$-$14K$	A_{TRR1}	Counter-based	16	✓	1/9	4	74.6% - 75.0%	1.96 - 2.96
A10-12	19-51	8	1	16	8	$12K$-$13K$	A_{TRR1}	Counter-based	16	✓	1/9	4	74.6% - 75.0%	1.48 - 2.86
A13-14	20-31	8	1	8	16	$11K$-$14K$	A_{TRR2}	Counter-based	16	✓	1/9	2	94.3% - 98.6%	1.53 - 2.78
B0	18-22	4	1	16	8	$44K$	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	2.13
B1-4	20-17	4	1	16	8	$159K$-$192K$	B_{TRR1}	Sampling-based	1	✗	1/4	2	23.3% - 51.2%	0.06 - 0.11
B5-6	16-48	4	1	16	8	$44K$-$50K$	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	1.85 - 2.03
B7	19-06	8	2	16	8	$20K$	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	31.14
B8	18-03	4	1	16	8	$43K$	B_{TRR1}	Sampling-based	1	✗	1/4	2	99.9%	2.57
B9-12	19-48	8	1	16	8	$42K$-$65K$	B_{TRR2}	Sampling-based	1	✗	1/9	2	36.3% - 38.9%	16.83 - 24.26
B13-14	20-08	4	1	16	8	$11K$-$14K$	B_{TRR3}	Sampling-based	1	✓	1/2	4	99.9%	16.20 - 18.12
C0-3	16-48	4	1	16	x8	$137K$-$194K$	C_{TRR1}	Mix	Unknown	✓	1/17	2	1.0% - 23.2%	0.05 - 0.15
C4-6	17-12	8	1	16	x8	$130K$-$150K$	C_{TRR1}	Mix	Unknown	✓	1/17	2	7.8% - 12.0%	0.06 - 0.08
C7-8	20-31	8	1	8	x16	$40K$-$44K$	C_{TRR1}	Mix	Unknown	✓	1/17	2	39.8% - 41.8%	9.66 - 14.56
C9-11	20-31	8	1	8	x16	$42K$-$53K$	C_{TRR2}	Mix	Unknown	✓	1/9	2	99.7%	9.30 - 32.04
C12-14	20-46	16	1	8	x16	$6K$-$7K$	C_{TRR3}	Mix	Unknown	✓	1/8	2	99.9%	4.91 - 12.64

SAFARI

Outline

1. DRAM Operation Basics

2. RowHammer & Target Row Refresh

3. The U-TRR Methodology

4. Observations & New RowHammer Access Patterns

5. RowHammer Bit Flip Analysis

6. Takeaways and Conclusion

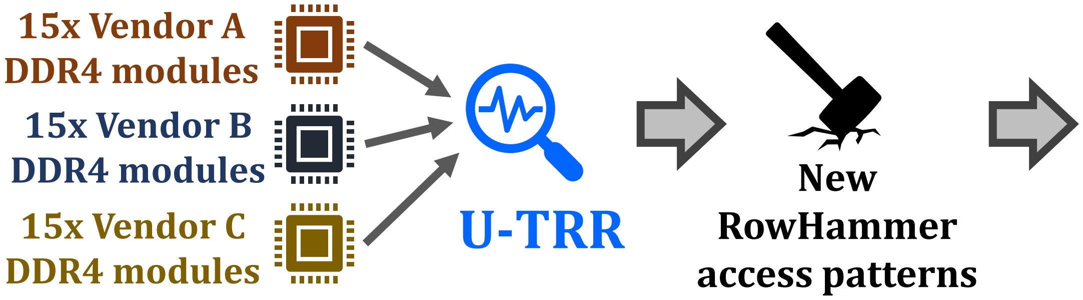
Conclusion

Target Row Refresh (TRR):
a set of obscure, undocumented, and proprietary RowHammer mitigation techniques

We cannot easily study the *security properties* of TRR

Is TRR fully secure? How can we validate its security guarantees?

U-TRR | A new methodology that leverages *data retention failures* to uncover the inner workings of TRR and study its security


15x Vendor A DDR4 modules
15x Vendor B DDR4 modules
15x Vendor C DDR4 modules
U-TRR
New RowHammer access patterns

All 45 modules we test are **vulnerable**

99.9% of rows in a DRAM bank experience **at least one RowHammer bit flip**

Up to **7** RowHammer **bit flips** in an 8-byte dataword, **making ECC ineffective**

TRR does not provide security against RowHammer

U-TRR can facilitate the development of **new RowHammer attacks** and **more secure RowHammer protection** mechanisms

U-TRR

Uncovering in-DRAM RowHammer Protection Mechanisms:
A New Methodology, Custom RowHammer Patterns, and Implications

Hasan Hassan

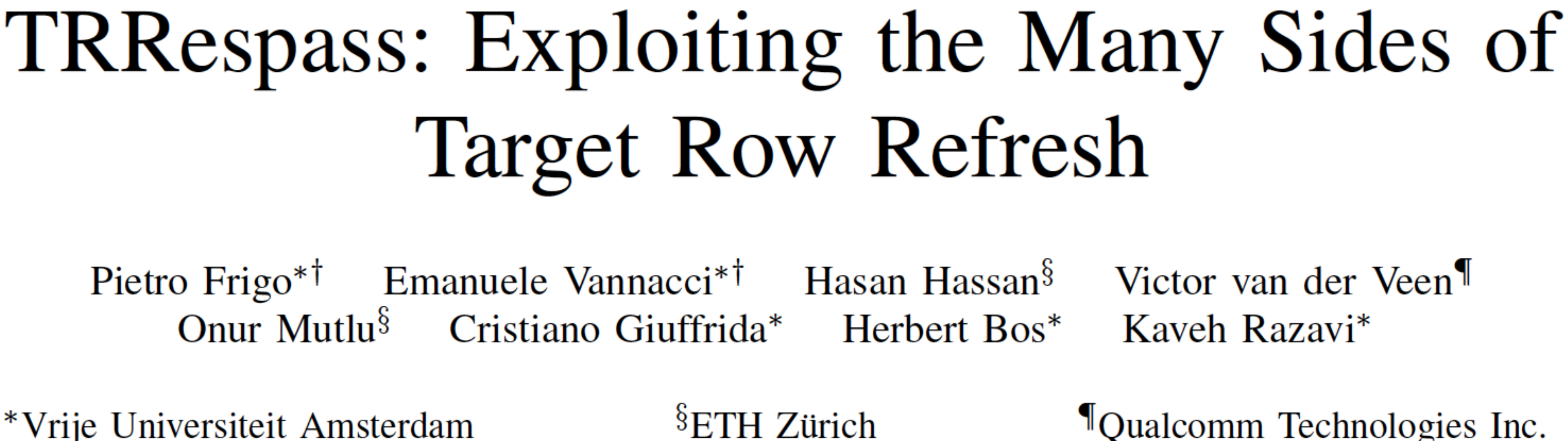
Yahya Can Tugrul *Jeremie S. Kim* *Victor van der Veen*
Kaveh Razavi *Onur Mutlu*

ETH zürich TOBB ETÜ University of Economics & Technology Qualcomm

TRRespass

RowHammer in 2020

- 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"** *Proceedings of the 41st IEEE Symposium on Security and Privacy* (***S&P***), San Francisco, CA, USA, May 2020.
  [Slides (pptx) (pdf)]
  [Talk Video (17 minutes)]
  [Source Code]
  [Web Article]
  Best paper award.



TRRespass: Exploiting the Many Sides of Target Row Refresh

Pietro Frigo[*†] Emanuele Vannacci[*†] Hasan Hassan[§] Victor van der Veen[¶]
Onur Mutlu[§] Cristiano Giuffrida[*] Herbert Bos[*] Kaveh Razavi[*]

[*]Vrije Universiteit Amsterdam [§]ETH Zürich [¶]Qualcomm Technologies Inc.

TRRespass

- First work to show that TRR-protected DRAM chips are vulnerable to RowHammer in the field
  - Mitigations advertised as secure are not secure

- Introduces the Many-sided RowHammer attack
  - Idea: Hammer many rows to bypass TRR mitigations (e.g., by overflowing proprietary TRR tables that detect aggressor rows)

- (Partially) reverse-engineers the TRR and pTRR mitigation mechanisms implemented in DRAM chips and memory controllers

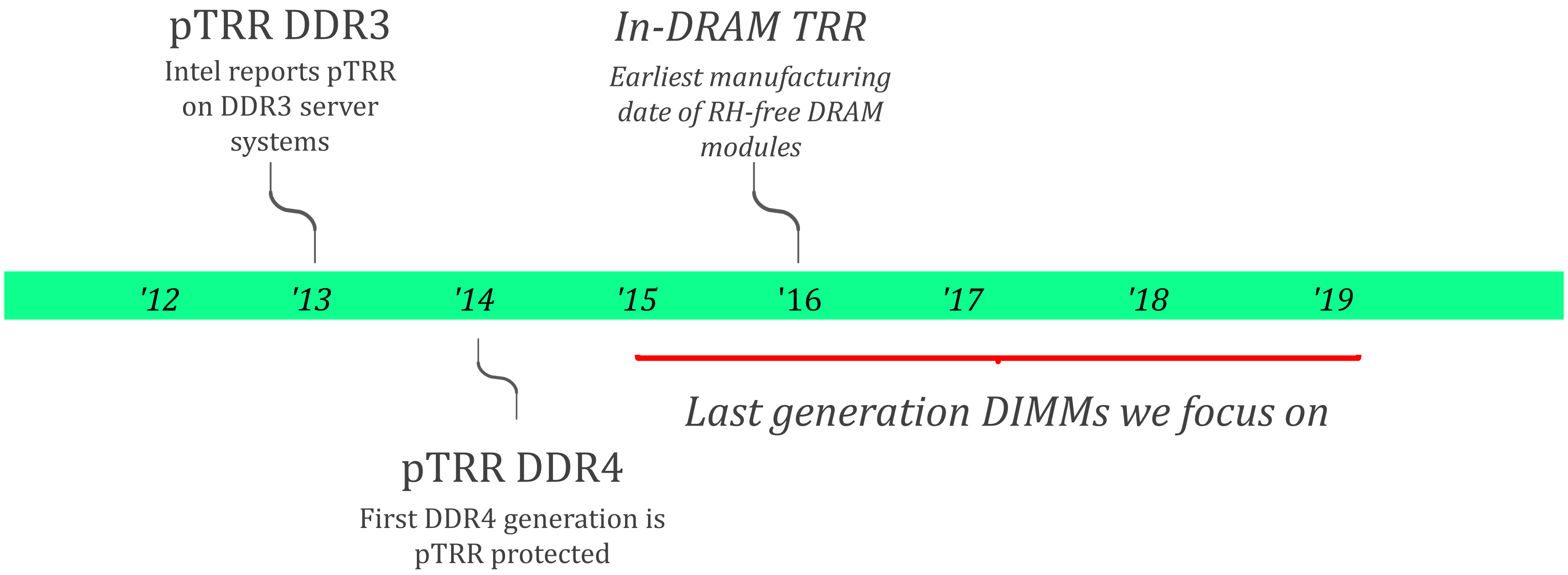
- Provides an automatic tool that can effectively create many-sided RowHammer attacks in DDR4 and LPDDR4(X) chips

Target Row Refresh (TRR)

- How does it work?
  1. *Track activation count of each DRAM row*
  2. *Refresh neighbor rows if row activation count exceeds a threshold*
  - Many possible implementations in practice
  - Security through obscurity

- In-DRAM TRR
  - Embedded in the DRAM circuitry, i.e., not exposed to the memory controller

Timeline of TRR Implementations



pTRR DDR3

Intel reports pTRR on DDR3 server systems

In-DRAM TRR

Earliest manufacturing date of RH-free DRAM modules

pTRR DDR4

First DDR4 generation is pTRR protected

Last generation DIMMs we focus on

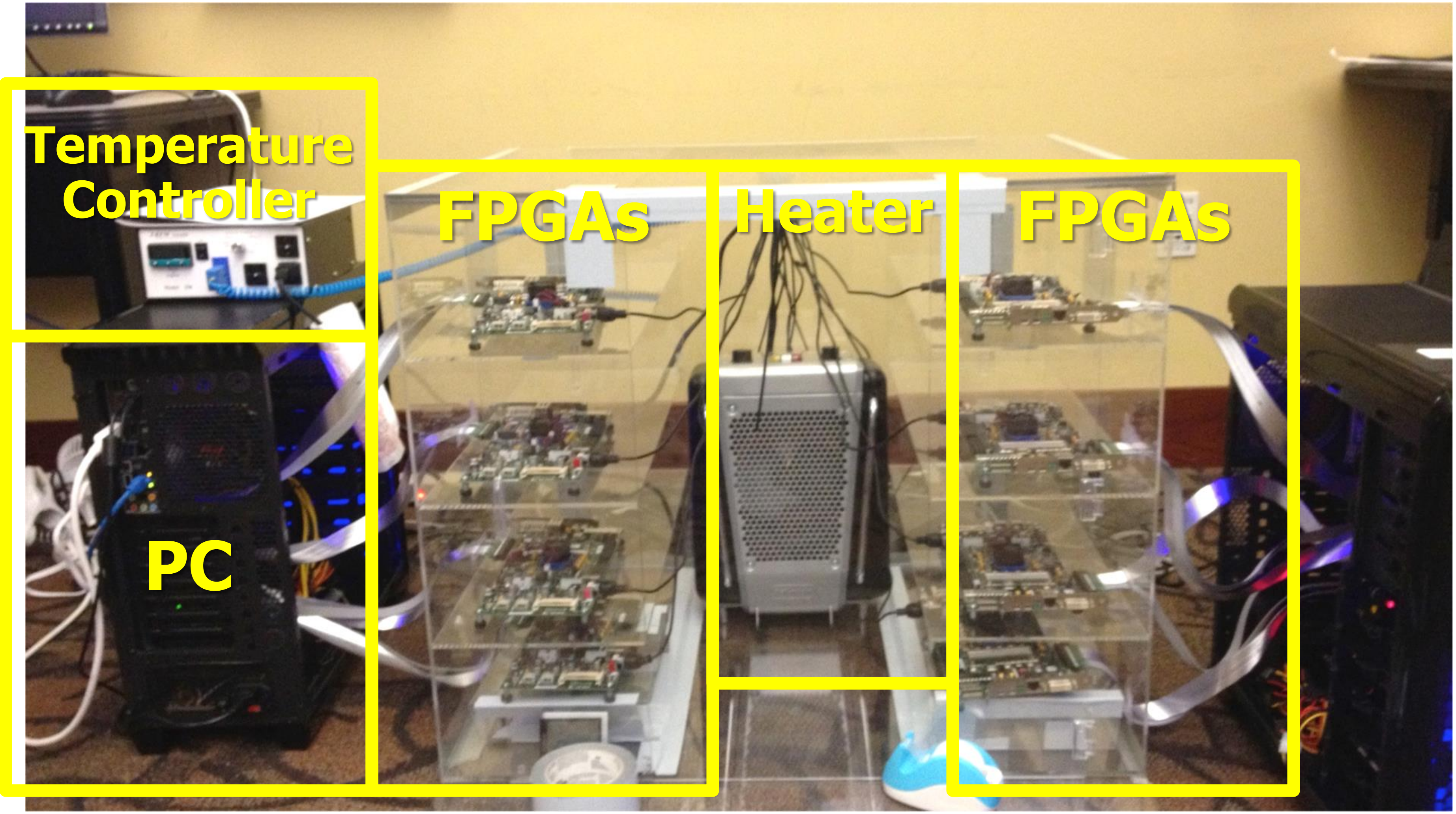
Our Goals

- Reverse engineer in-DRAM TRR to demystify how it works

- Bypass TRR protection
  - A Novel hammering pattern: **The Many-sided RowHammer**
  - Hammering up to **20 aggressor rows** allows bypassing TRR

- Automatically test memory devices: **TRRespass**
  - Automate hammering pattern generation

Infrastructures to Understand Such Issues



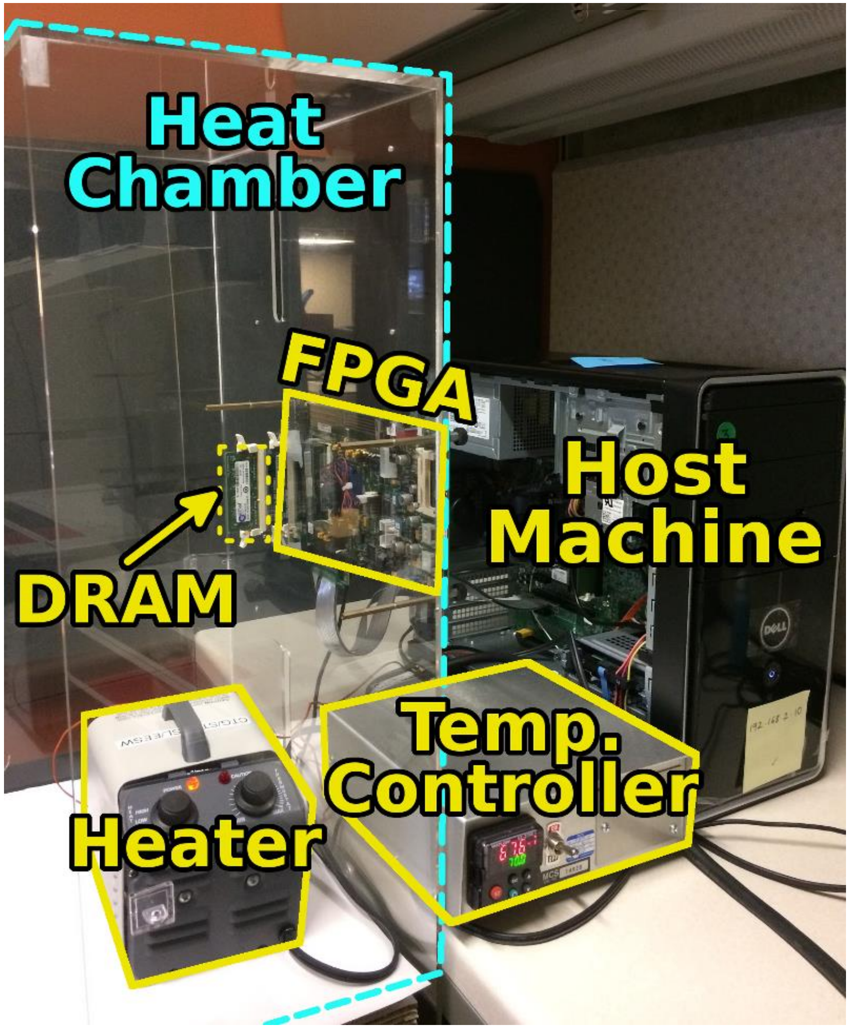
Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

SoftMC: Open Source DRAM Infrastructure

- Hasan Hassan et al., "**SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies**," HPCA 2017.

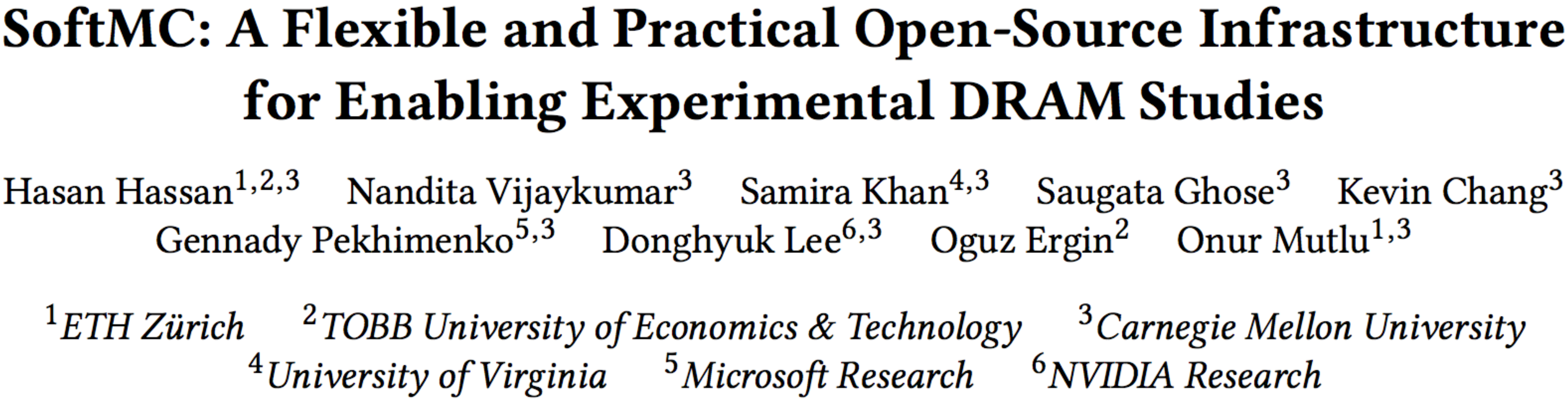
- **Flexible**
- **Easy to Use (C++ API)**
- **Open-source**

github.com/CMU-SAFARI/SoftMC



SoftMC

- https://github.com/CMU-SAFARI/SoftMC



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*

Components of In-DRAM TRR

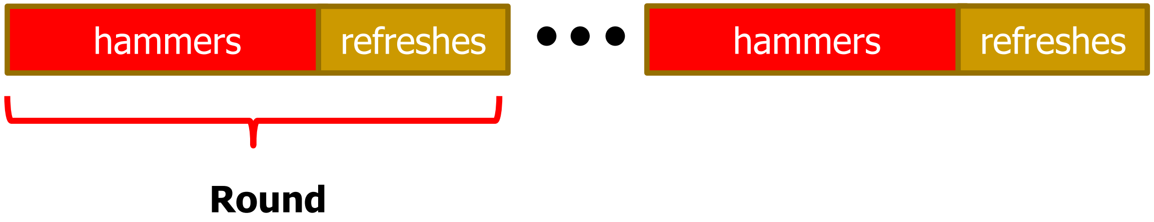
- **Sampler**
  - Tracks aggressor rows activations
  - Design options:
    - Frequency based (record every Nth row activation)
    - Time based (record first N row activations)
    - Random seed (record based on a coin flip)
  - Regardless, the sampler has a limited size

- **Inhibitor**
  - Prevents bit flips by refreshing victim rows
    - The latency of performing victim row refreshes is squeezed into slack time available in *tRFC* (i.e., the latency of regular Refresh command)

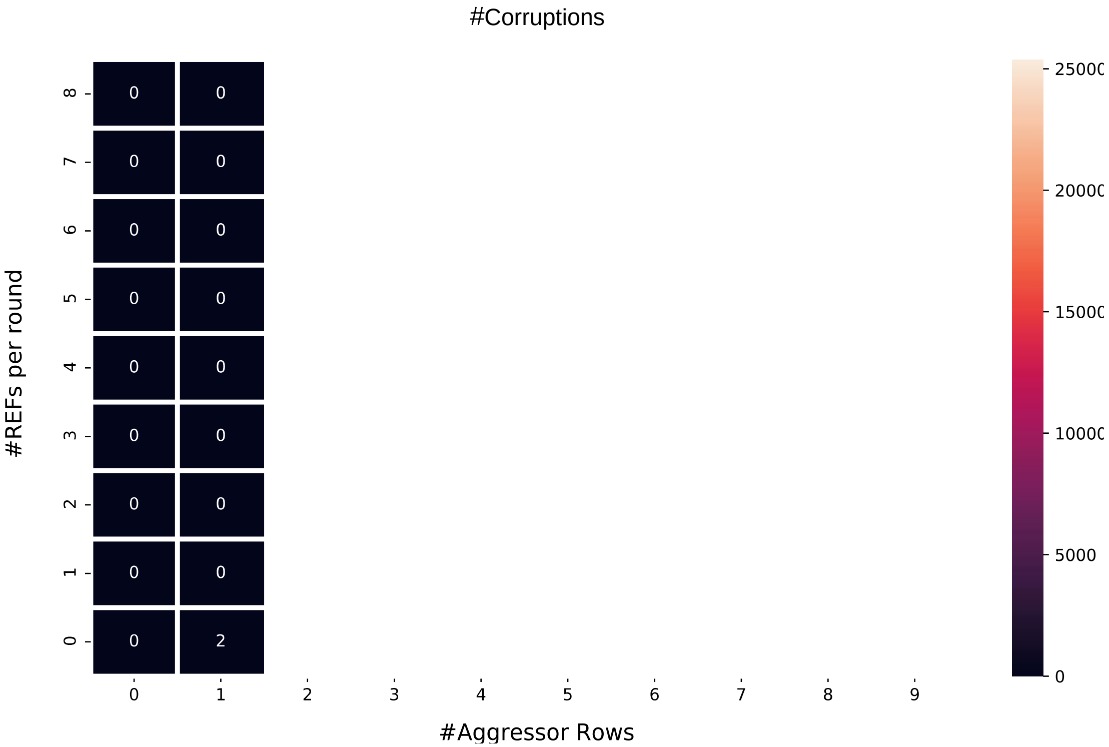
Case Study: Vendor C

How big is the sampler?

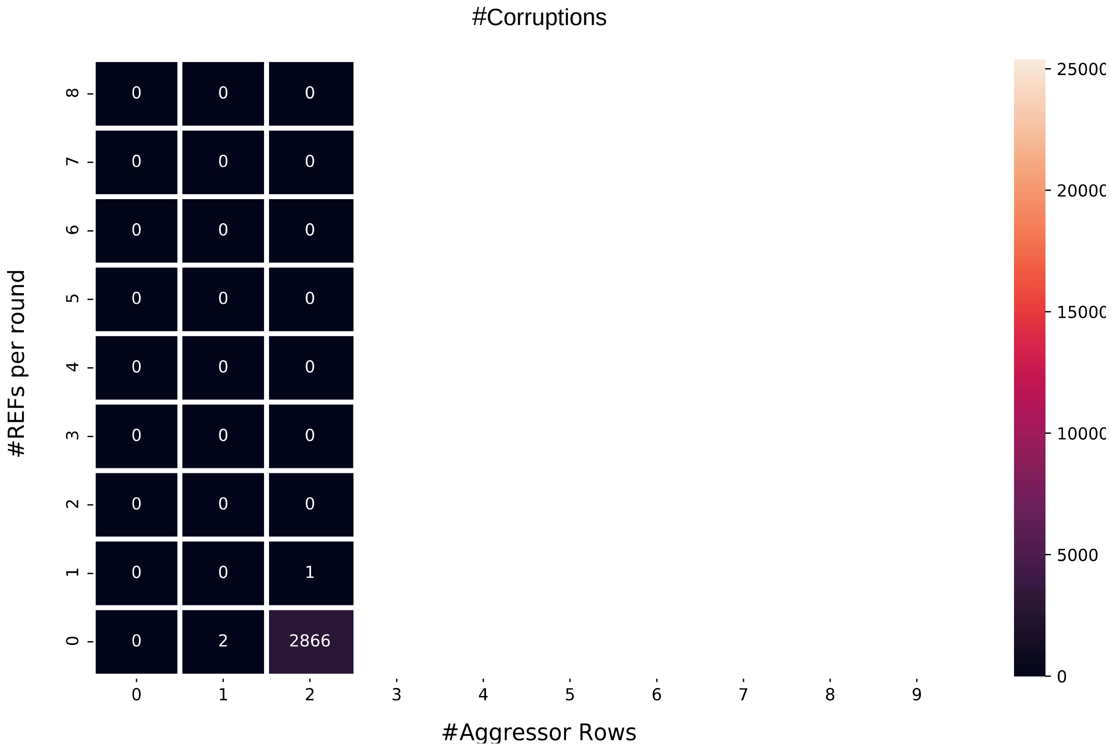
- Pick **N** aggressor rows
- Perform a series of hammers (i.e., activations of aggressors)
  - **8K activations**
- After each series of hammers, issue **R refreshes**
- **10 Rounds**



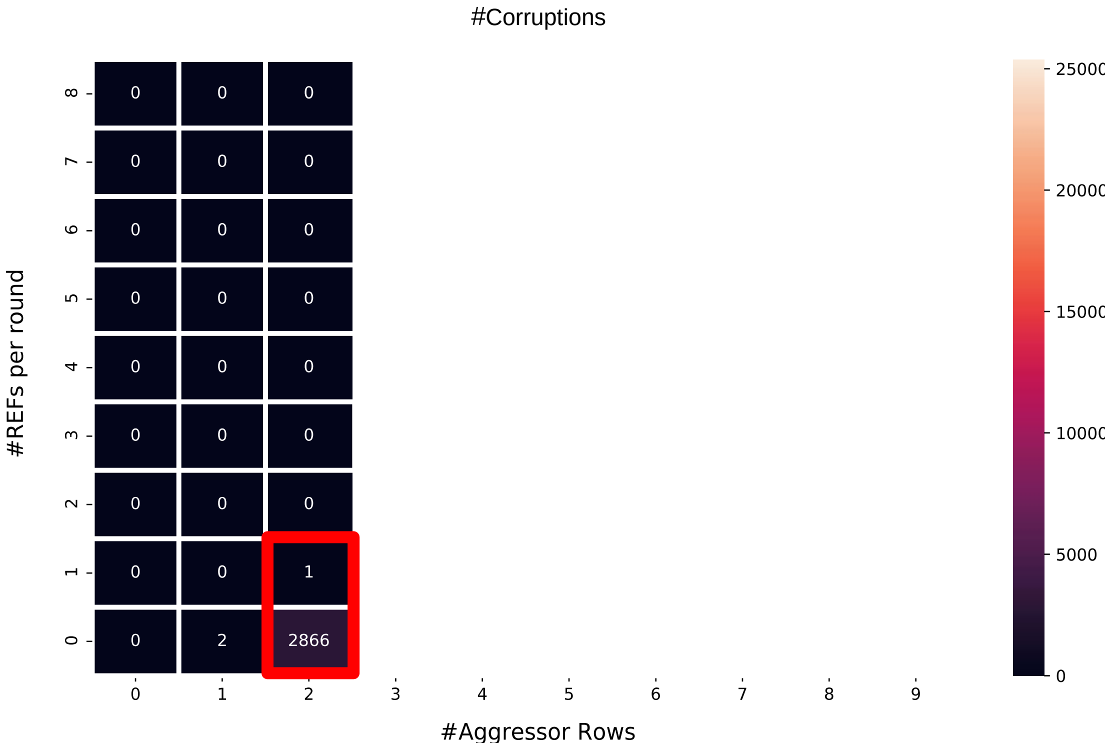
Case Study: Vendor C



Case Study: Vendor C

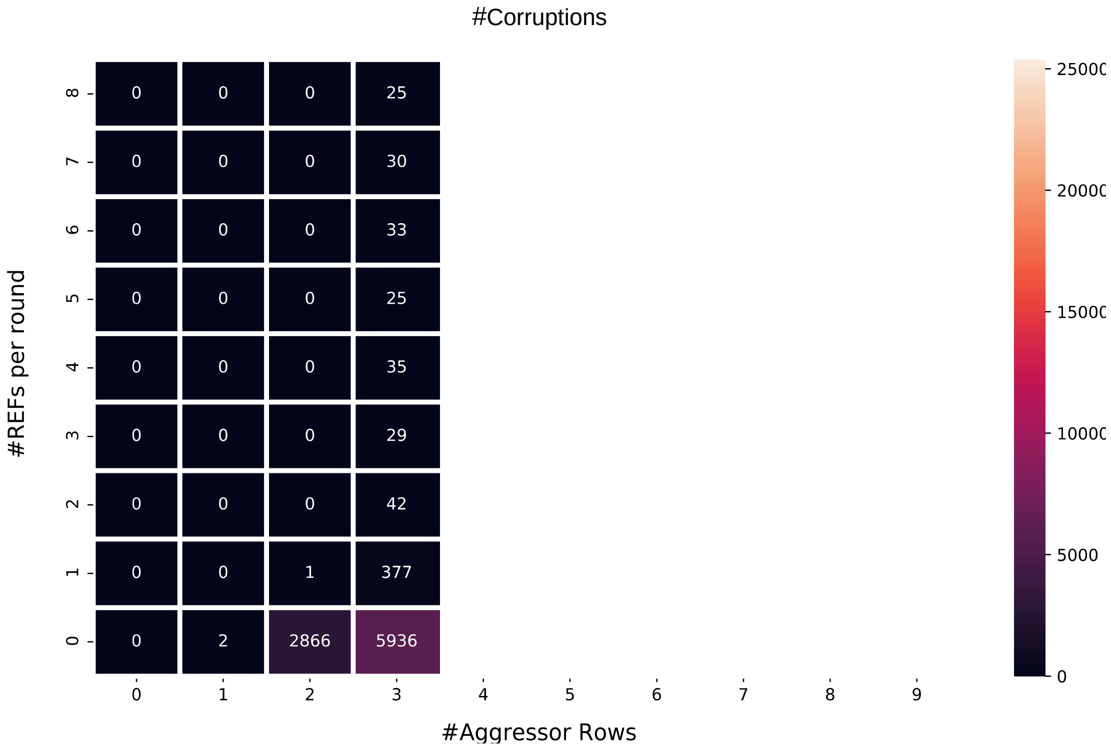


Case Study: Vendor C

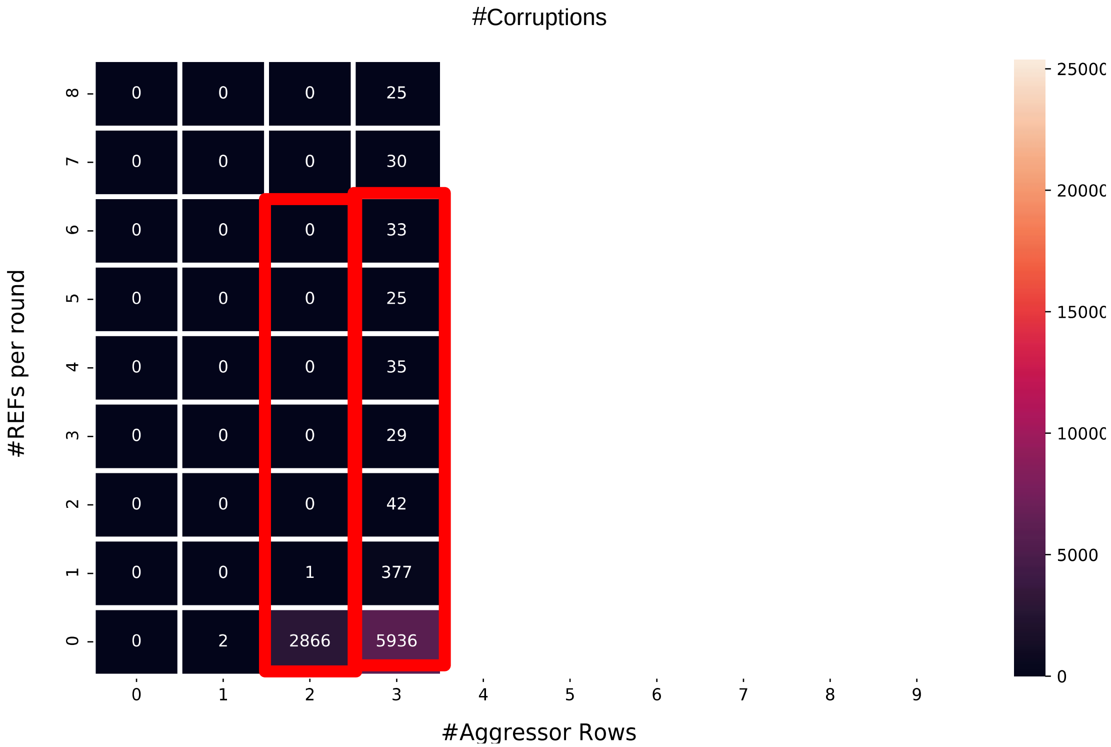


1. The TRR mitigation **acts on a refresh command**

Case Study: Vendor C



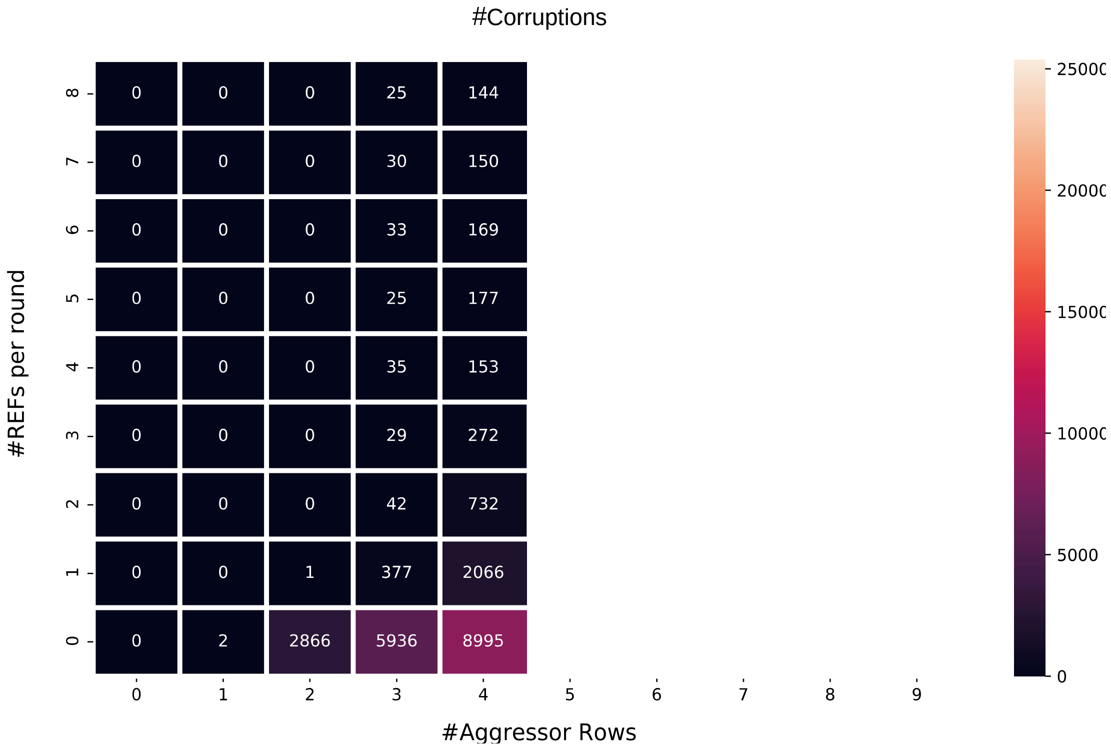
Case Study: Vendor C



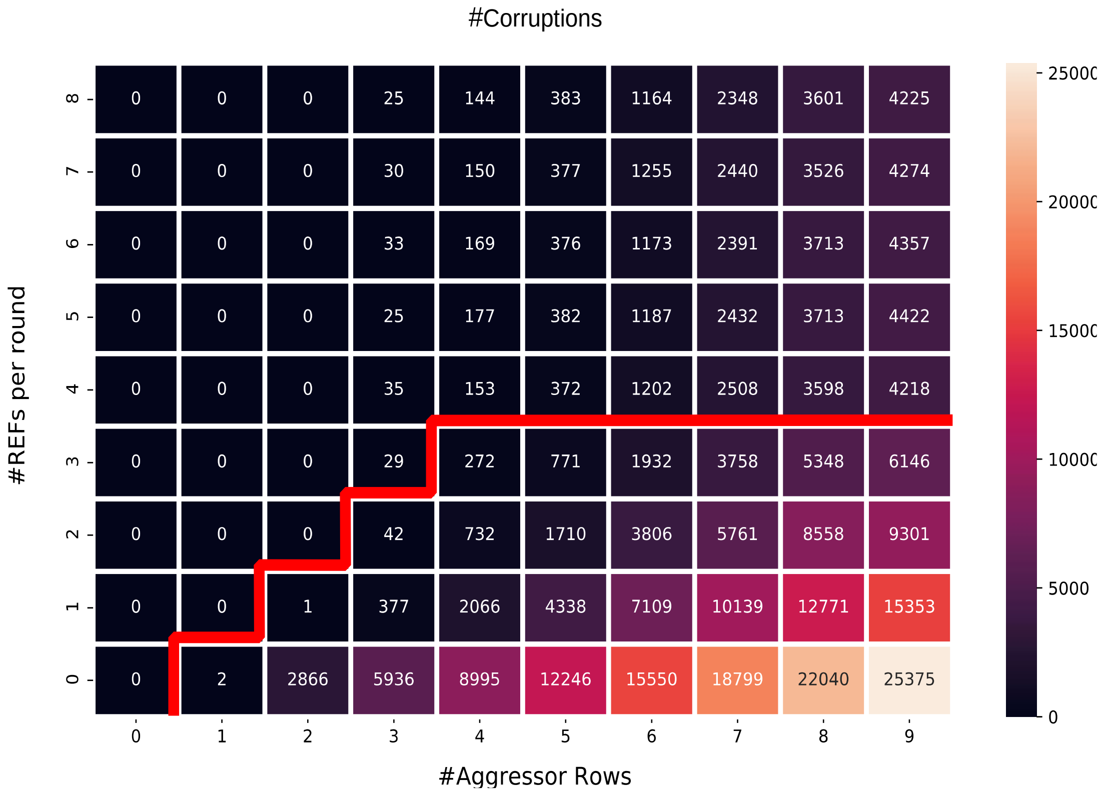
2. The mitigation **can sample more than one aggressor** per refresh interval

3. The mitigation **can refresh only a single victim** within a refresh operation

Case Study: Vendor C



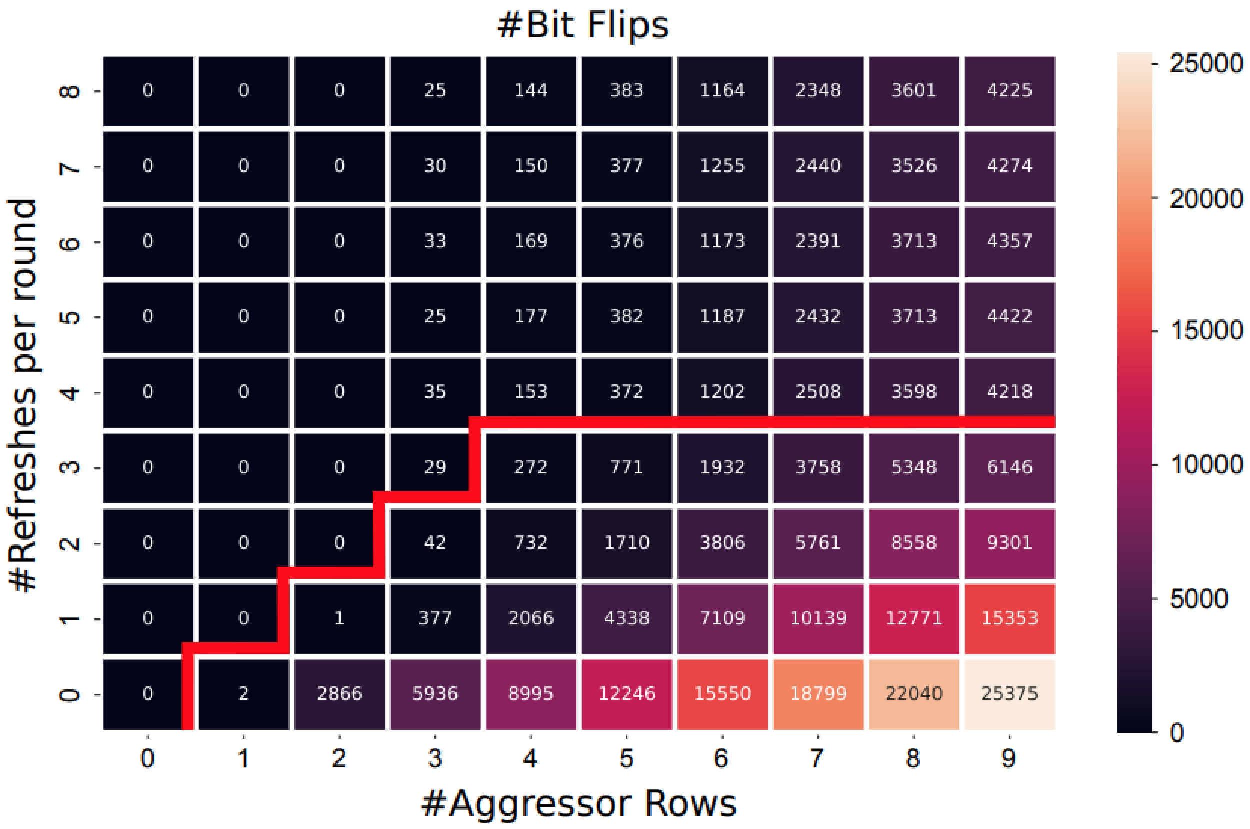
Case Study: Vendor C



	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

4. Sweeping the number of refresh operations and aggressor rows while hammering reveals the sampler size

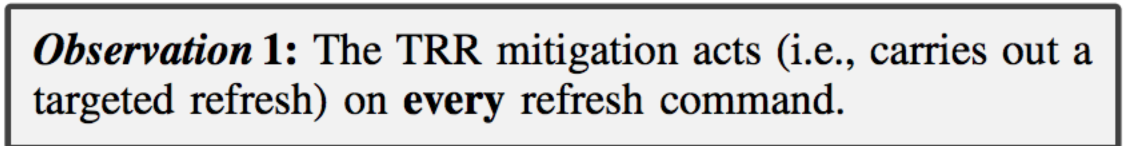
Many-Sided Hammering



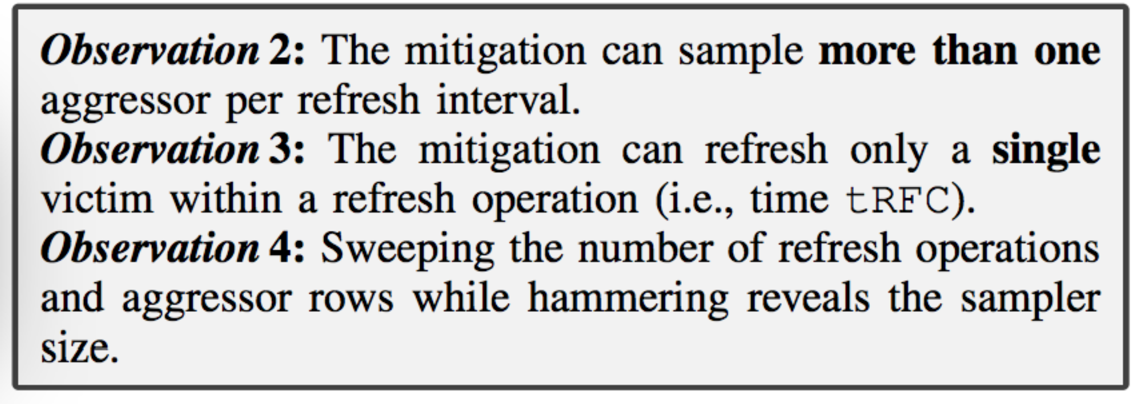
	0	1	2	3	4	5	6	7	8	9
8	0	0	0	25	144	383	1164	2348	3601	4225
7	0	0	0	30	150	377	1255	2440	3526	4274
6	0	0	0	33	169	376	1173	2391	3713	4357
5	0	0	0	25	177	382	1187	2432	3713	4422
4	0	0	0	35	153	372	1202	2508	3598	4218
3	0	0	0	29	272	771	1932	3758	5348	6146
2	0	0	0	42	732	1710	3806	5761	8558	9301
1	0	0	1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

Fig. 9: Refreshes vs. Bit Flips. Module $\mathcal{C}_{12}$: Number of bit flips detected when sending r refresh commands to the module. We report this for different number of aggressor rows (n). For example, when hammering 5 rows, followed by sending 2 refreshes, we find 1,710 bit flips. This figure shows that the number of bit flips stabilizes for $r \geq 4$, implying that the size of the sampler may be 4.

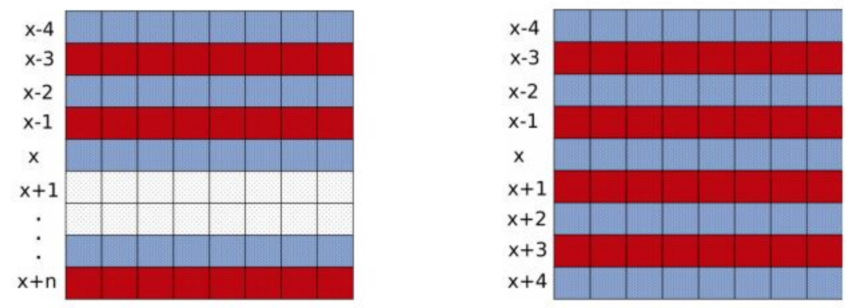
Some Observations



***Observation* 1:** The TRR mitigation acts (i.e., carries out a targeted refresh) on **every** refresh command.



***Observation* 2:** The mitigation can sample **more than one** aggressor per refresh interval.
***Observation* 3:** The mitigation can refresh only a **single** victim within a refresh operation (i.e., time `tRFC`).
***Observation* 4:** Sweeping the number of refresh operations and aggressor rows while hammering reveals the sampler size.

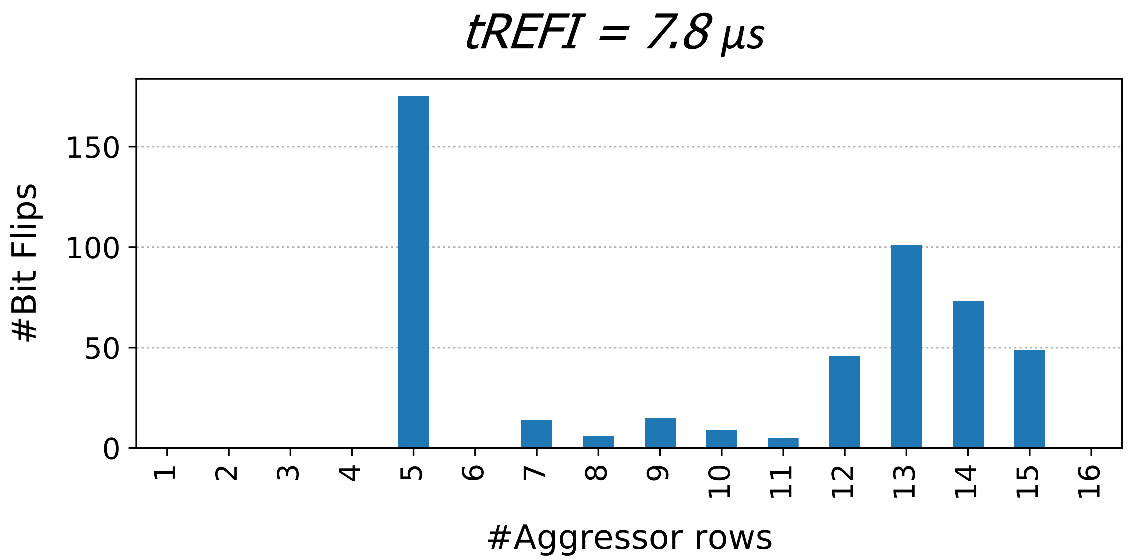


(a) Assisted double-sided **(b)** 4-sided

Fig. 12: Hammering patterns discovered by *TRRespass*. Aggressor rows are in red (🟥) and victim rows are in blue (🟦).

Case Study: Vendor C

Hammering using the default refresh rate



BitFlips vs. Number of Aggressor Rows

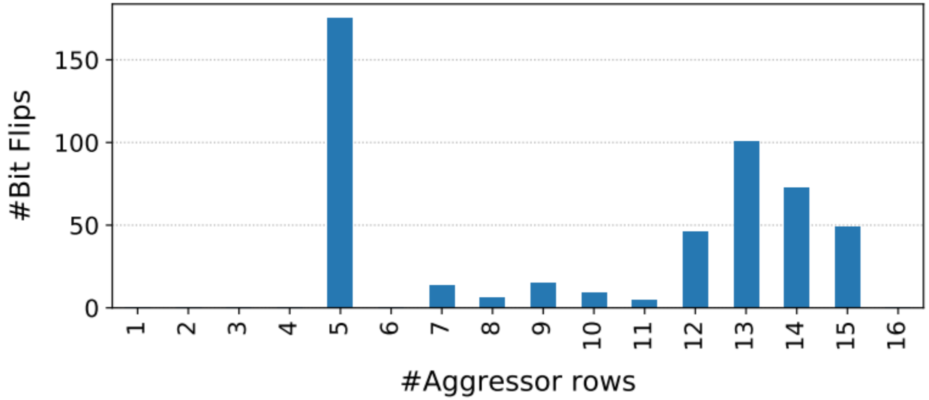


Fig. 10: Bit flips vs. number of aggressor rows. Module $\mathcal{C}_{12}$: Number of bit flips in bank 0 as we vary the number of aggressor rows. Using SoftMC, we refresh DRAM with standard `tREFI` and run the tests until each aggressor rows is hammered 500K times.

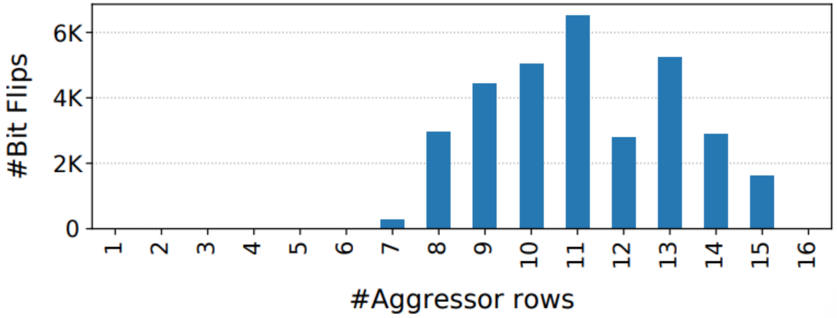


Fig. 11: Bit flips vs. number of aggressor rows. Module $\mathcal{A}_{15}$: Number of bit flips in bank 0 as we vary the number of aggressor rows. Using SoftMC, we refresh DRAM with standard `tREFI` and run the tests until each aggressor rows is hammered 500K times.

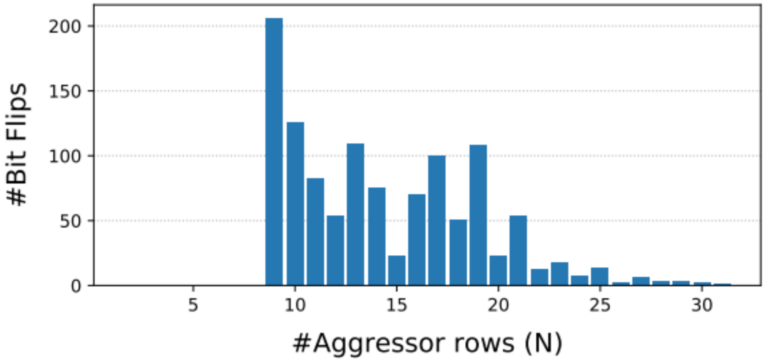


Fig. 13: Bit flips vs. number of aggressor rows. Module $\mathcal{A}_{10}$: Number of bit flips triggered with *N-sided* RowHammer for varying number of *N* on Intel Core i7-7700K. Each aggressor row is one row away from the closest aggressor row (i.e., `VAVAVA...` configuration) and aggressor rows are hammered in a round-robin fashion.

TRRespass Key Results

- 13 out of 42 tested DDR4 DRAM modules are vulnerable
  - From all 3 major manufacturers
  - 3-, 9-, 10-, 14-, 19-sided attacks needed

- 5 out of 13 mobile phones tested vulnerable
  - From 4 major manufacturers
  - With LPDDR4(X) DRAM chips

- These results are scratching the surface
  - TRRespass tool is not exhaustive
  - There is a lot of room for uncovering more vulnerable chips and phones

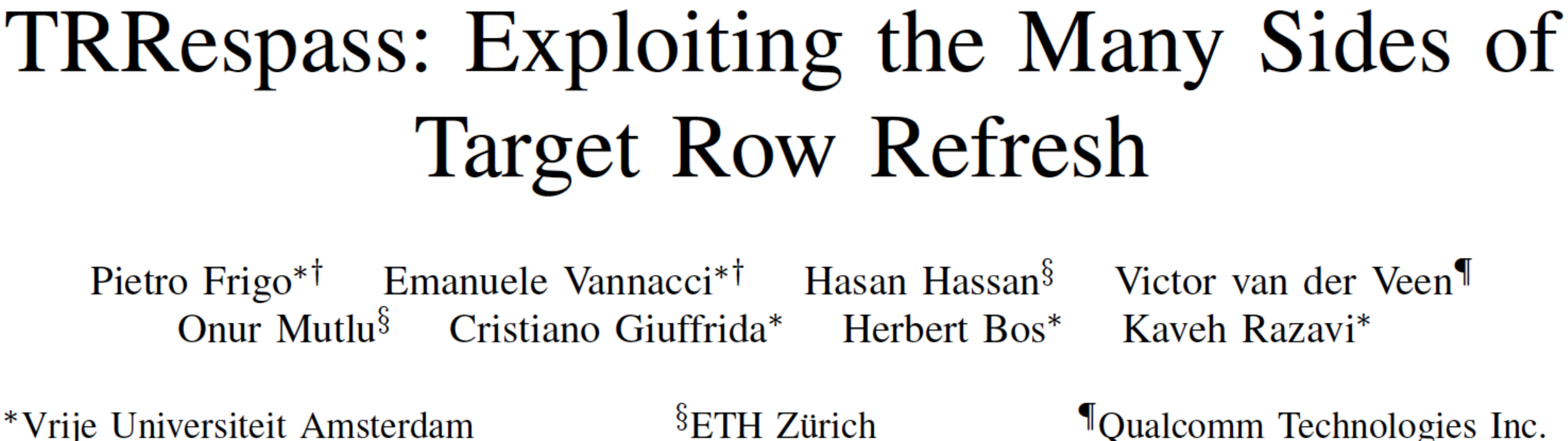
TRRespass Key Takeaways

RowHammer is still
an open problem

Security by obscurity
is likely not a good solution

More on TRRespass

- 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"** *Proceedings of the 41st IEEE Symposium on Security and Privacy* (***S&P***), San Francisco, CA, USA, May 2020.
  [Slides (pptx) (pdf)]
  [Talk Video (17 minutes)]
  [Source Code]
  [Web Article]
  Best paper award.



TRRespass: Exploiting the Many Sides of Target Row Refresh

Pietro Frigo[*†] Emanuele Vannacci[*†] Hasan Hassan[§] Victor van der Veen[¶]
Onur Mutlu[§] Cristiano Giuffrida[*] Herbert Bos[*] Kaveh Razavi[*]

[*] Vrije Universiteit Amsterdam [§] ETH Zürich [¶] Qualcomm Technologies Inc.

P&S SoftMC

Understanding and Improving Modern DRAM Performance, Reliability, and Security with Hands-On Experiments

Hasan Hassan

Prof. Onur Mutlu

ETH Zürich

Spring 2022

22 March 2022