TRRespass: Exploiting the Many Sides of Target Row Refresh

***Proceedings of the <u>41st IEEE Symposium on Security and Privacy</u> (S&P)*, May 2020.**

Authors: Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos and Kaveh Razavi

Presenter: Meryem Banu Cavlak
December 9, 2021

Executive Summary

- **Problem:** RowHammer protection in current DRAM modules relies on **undocumented** in-DRAM TRR mitigations.
- **Goal:** Analyzing the security guarantees of in-DRAM TRR against RowHammer attacks.
- **Key Methodology:**
  - Perform series of hammers and refreshes.
  - Vary the number of hammers and refreshes.
- **Key Observations:**
  - The TRR mitigation acts on refresh command.
  - Sweeping the number of refreshes & aggressor rows reveals the sampler size.
- **Key Mechanism (TRRespass):** Black-box RowHammer test suite that generates effective access patterns to bypass inDRAM TRR solutions by varying the cardinality & location of aggressors.
- **Key Results:**
  - 13/42 DDR4 and LPDDR4 modules are vulnerable to RowHammer.
  - Bit flips can be exploited to create RowHammer attacks.

Executive Summary

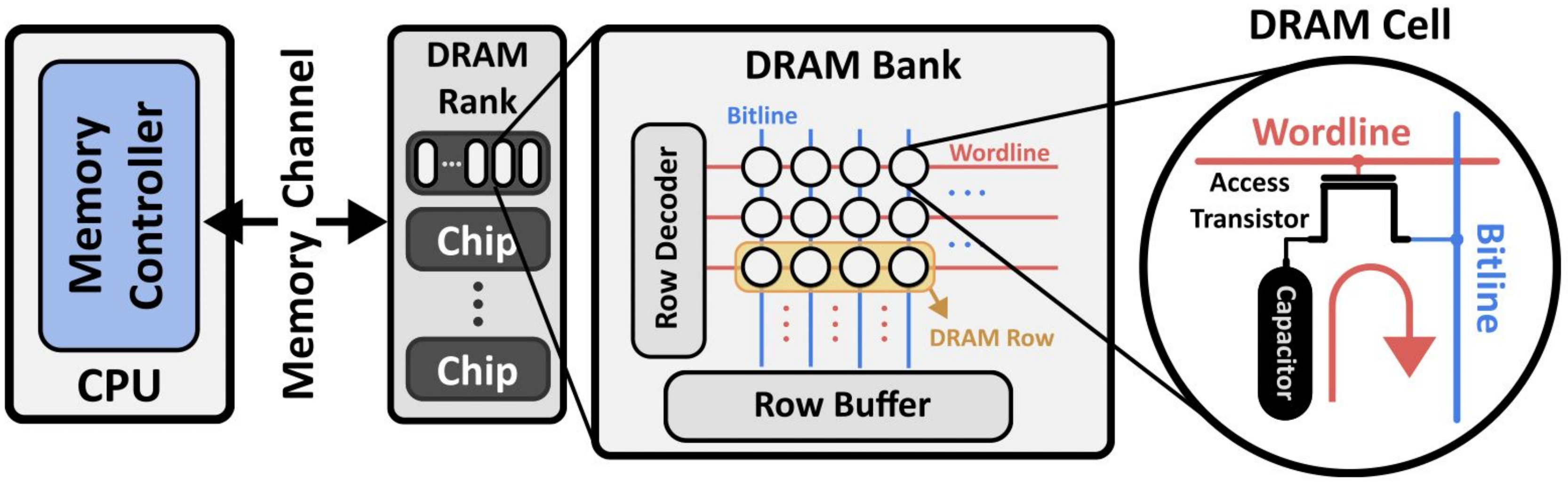
- **Problem:** RowHammer protection in current DRAM modules relies on **undocumented** in-DRAM TRR mitigations.
- **Goal:** Analyzing the security guarantees of in-DRAM TRR against RowHammer attacks.
- **Key Methodology:**
  - Perform series of hammers and refreshes.

RowHammer is still an open problem.

generates effective access patterns to bypass inDRAM TRR solutions by varying the cardinality & location of aggressors.

- **Key Results:**
  - 13/42 DDR4 and LPDDR4 modules are vulnerable to RowHammer.
  - Bit flips can be exploited to create RowHammer attacks.

Background – DRAM Organization

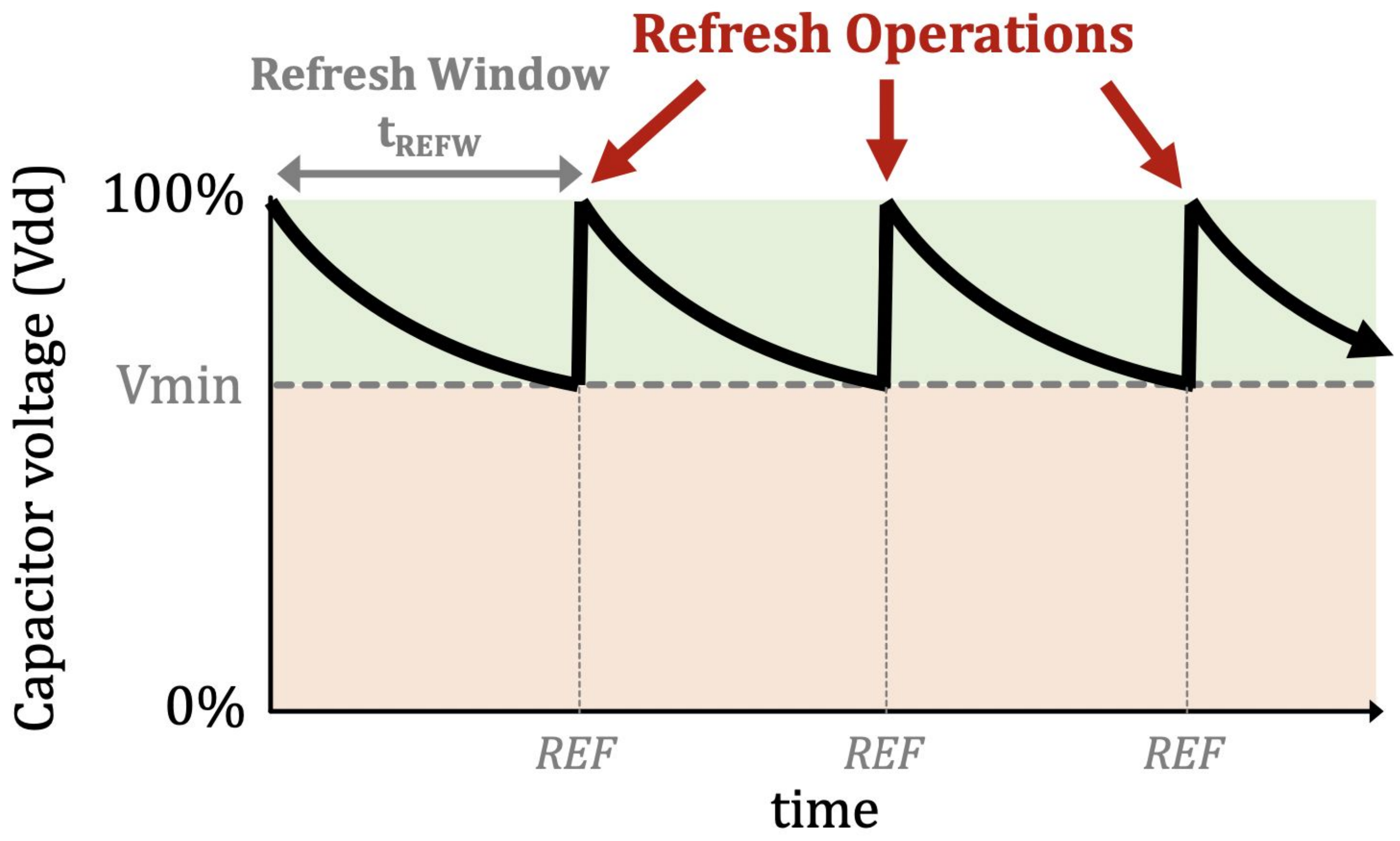


A row needs to be **activated** and fetched to **Row Buffer** before access.

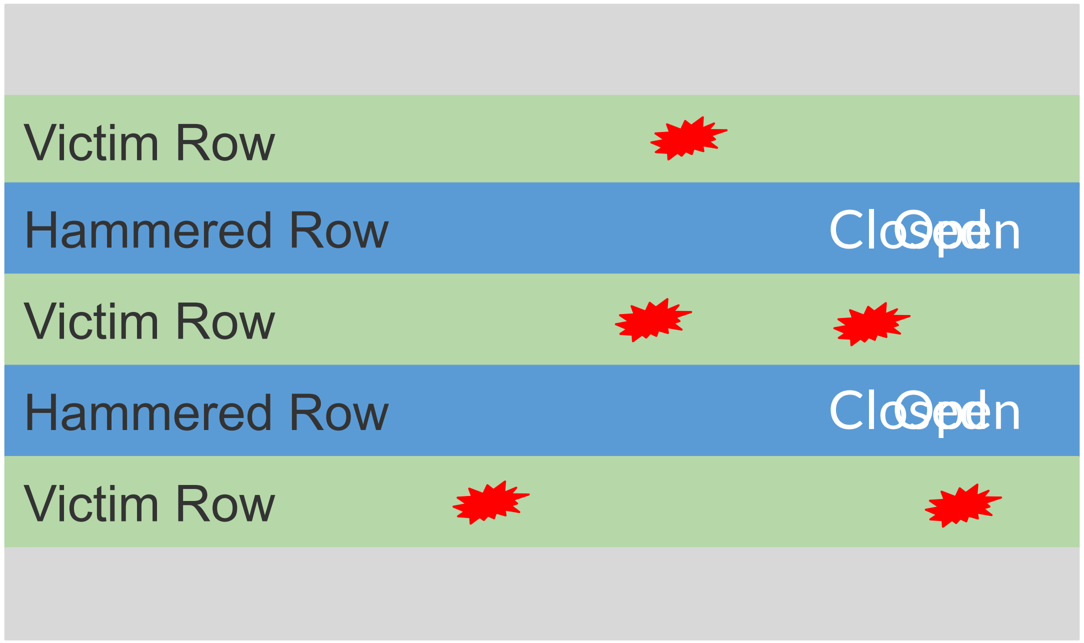
A DRAM cell consists of a **capacitor** to encode the bit and an **access transistor** to access the bit.

Background

Periodic refreshes are required to preserve the stored data as the cells leak charge.



Background

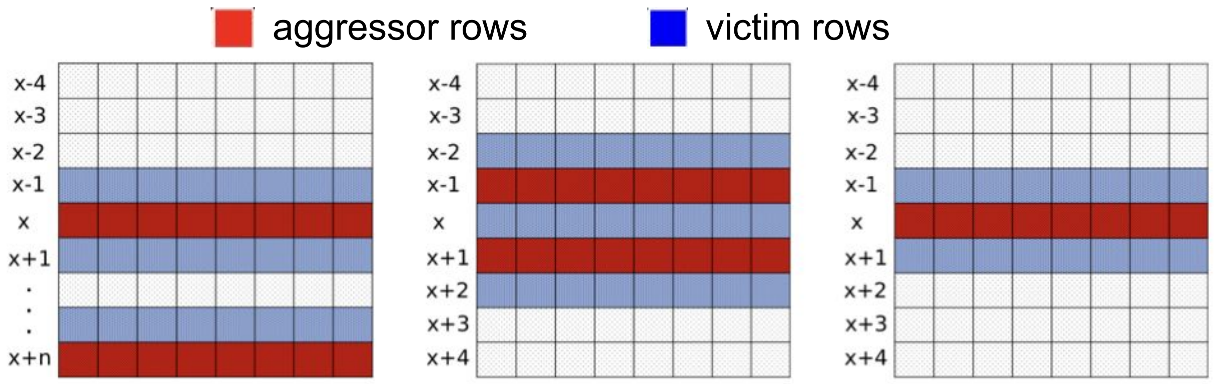


Repeatedly **opening** and **closing** DRAM rows (Hammered Row) cause **RowHammer bit flips** in nearby cells (Victim Rows).

Background

42 recent DDR4 modules are tested against

- single-sided
- double-sided
- one-location hammering patterns.

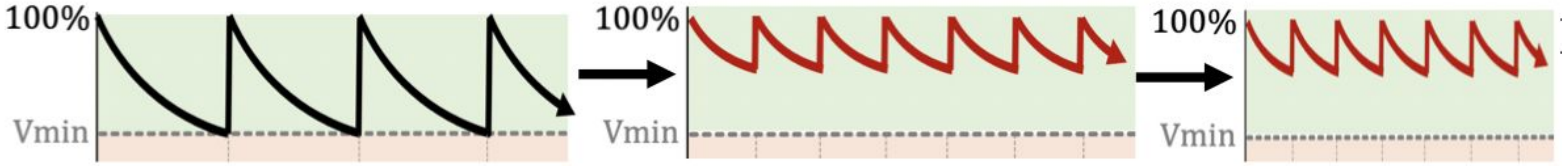


(a) Single-sided (b) Double-sided (c) One-location

No bit flips observed. Existing mitigation mechanisms are **effective** against **the known** hammering patterns.

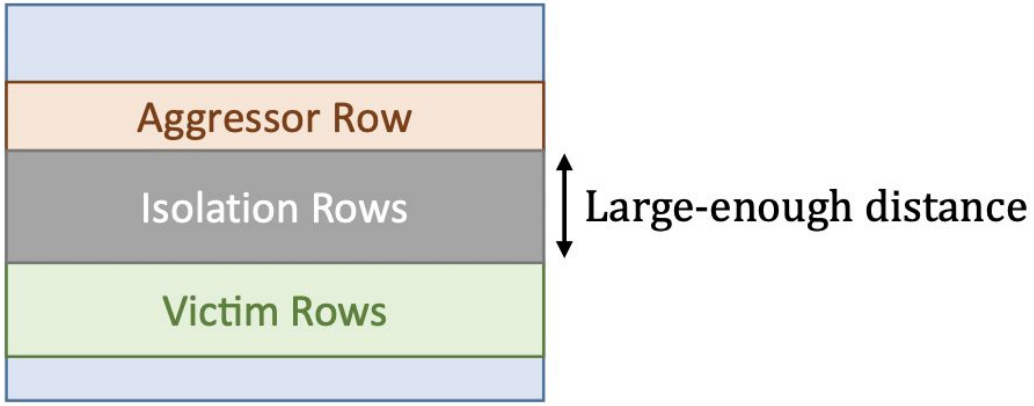
Prior Work

- Increasing DRAM refresh rate **Not scalable**



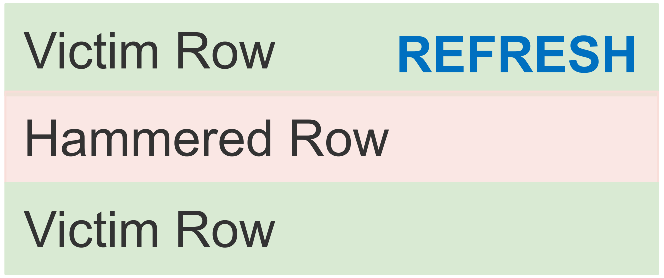
- Physical Isolation

In-DRAM mapping is proprietary information



- Error Correcting Codes **Not effective**

- PARA



Not deterministic

Problem: Is RowHammer Solved?

Target Row Refresh (TRR): *family* of mitigation mechanism that selectively refresh the victim rows.

Can be employed in:

1. The Memory Controller
2. Inside the DRAM Chips

Analyzing the Memory Controller

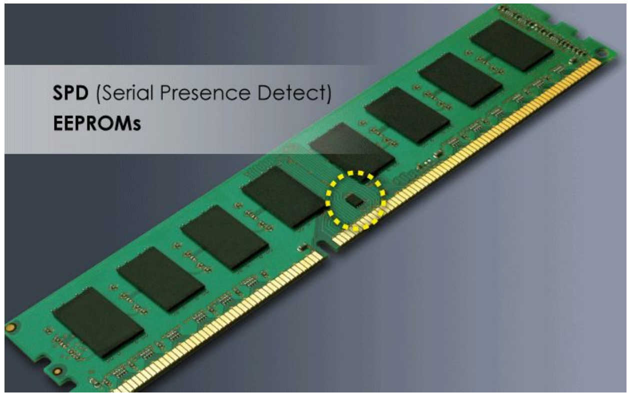
Memory Controller can

- monitor the number of activations to specific DRAM rows
- send additional refreshes to DRAM rows affected

The maximum number of ACTIVATEs (**MAC**) *a row can bear before any bit* in its neighboring rows flips should be known.

Three possible MAC values in commodity memory modules:

1. unlimited
2. untested
3. a discrete value (e.g., 300K)



Analyzing Intel pTRR

Aim:

- **verify** the existence of pTRR
- analyze different Intel systems to understand the **deployment** and **effectiveness** of pTRR

Motivation:

- p*TRR* is the most prominent in memory controller defense mechanism
- very little is actually **known** about the pTRR mechanism

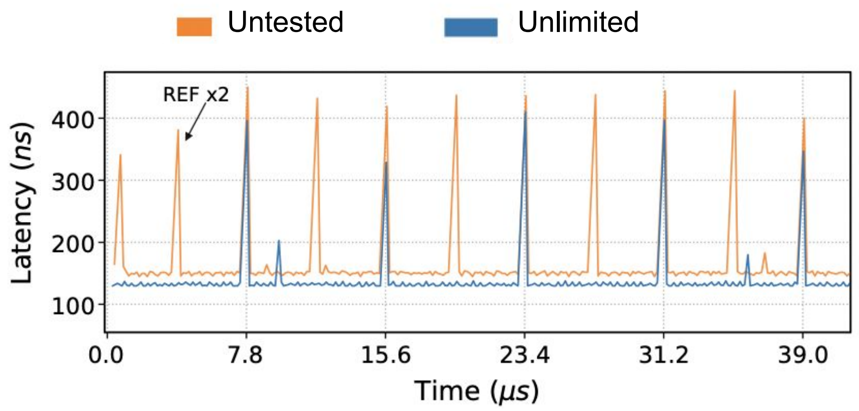
Mechanism:

- refresh the victim row when # number of hammers > MAC
- double refresh for non-TRR-compliant DRAM modules

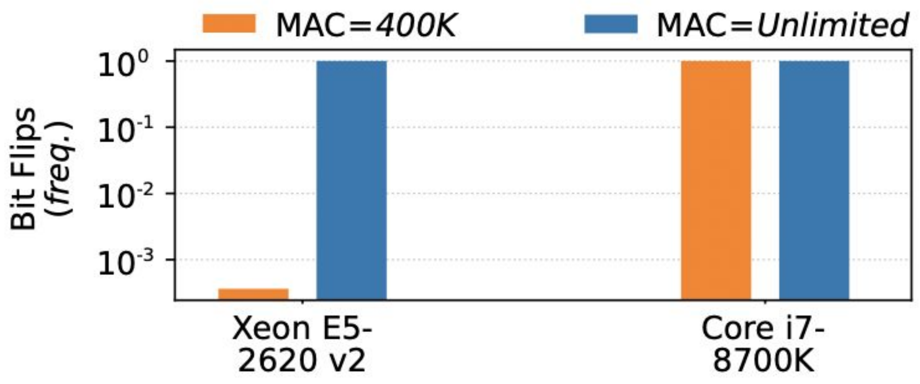
Reverse Engineering pTRR

Experiments (Xeon E5–2620):

1. overwrite the MAC value to untested



2. overwrite the MAC value to different discrete values



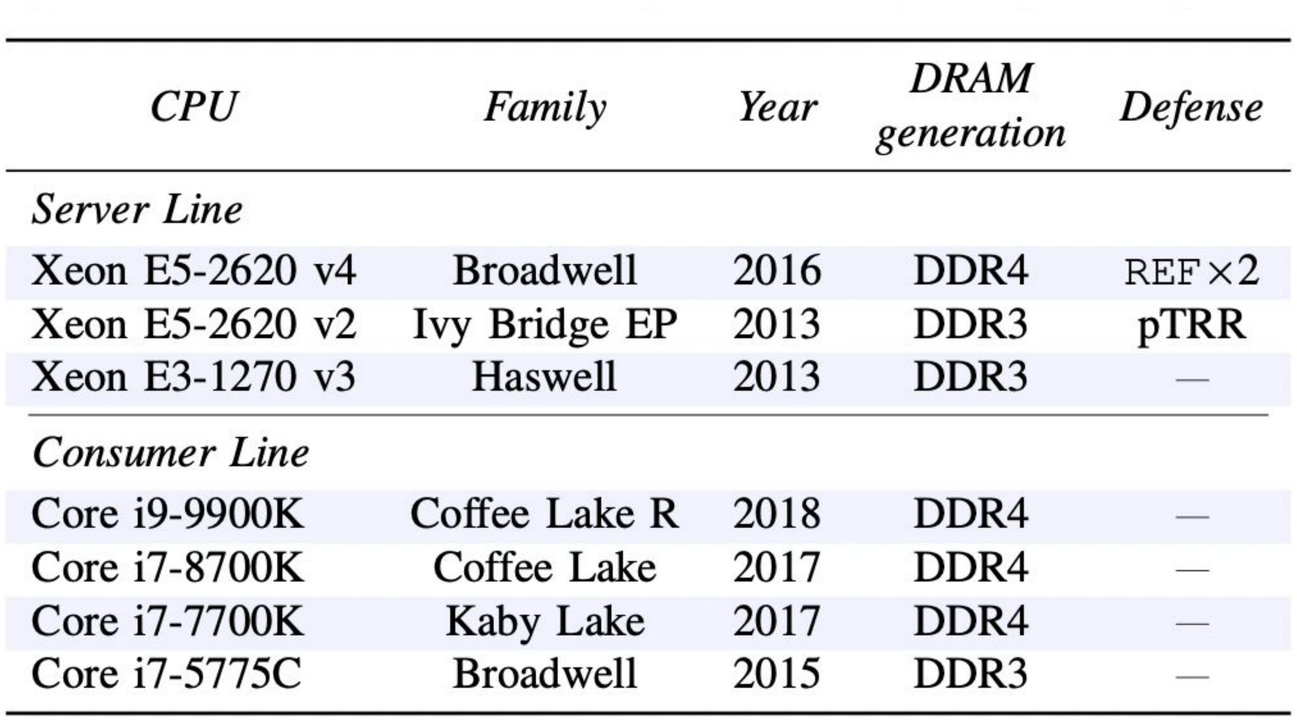
pTRR Deployment

1. overwrite the MAC value to untested

   default refresh rate is observed

2. overwrite the MAC value to different discrete values

   same number of bit flips for all MAC values

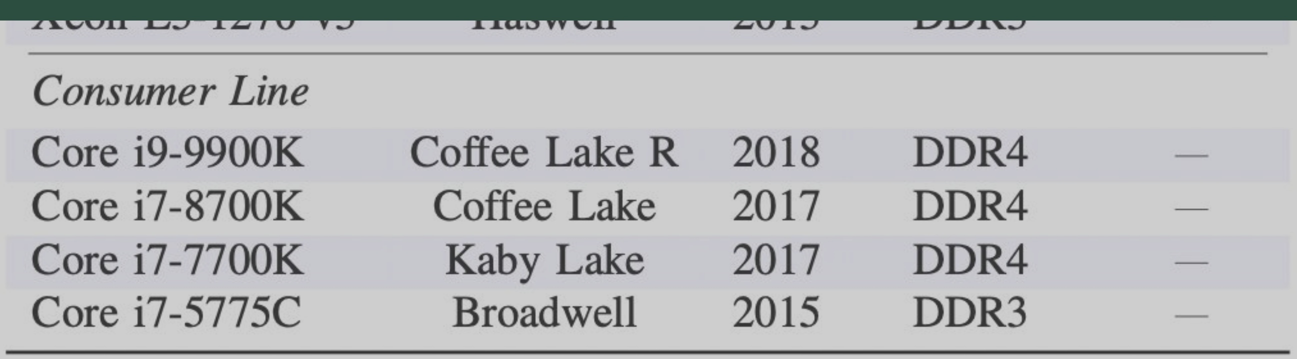


CPU	*Family*	*Year*	*DRAM generation*	*Defense*
Server Line				
Xeon E5-2620 v4	Broadwell	2016	DDR4	REF×2
Xeon E5-2620 v2	Ivy Bridge EP	2013	DDR3	pTRR
Xeon E3-1270 v3	Haswell	2013	DDR3	—
Consumer Line				
Core i9-9900K	Coffee Lake R	2018	DDR4	—
Core i7-8700K	Coffee Lake	2017	DDR4	—
Core i7-7700K	Kaby Lake	2017	DDR4	—
Core i7-5775C	Broadwell	2015	DDR3	—

pTRR Deployment

1. overwrite the MAC value to untested
   default refresh rate is observed
2. overwrite the MAC value to different discrete values
   same number of bit flips for all MAC values

RowHammer mitigation entirely relies on undocumented in-DRAM TRR mitigations.



[illegible]	[illegible]	[illegible]	[illegible]	
Consumer Line				
Core i9-9900K	Coffee Lake R	2018	DDR4	—
Core i7-8700K	Coffee Lake	2017	DDR4	—
Core i7-7700K	Kaby Lake	2017	DDR4	—
Core i7-5775C	Broadwell	2015	DDR3	—

Inside the DRAM Chips

- The exact implementation details of **in-DRAM TRR mechanisms** are **unknown**

- The **goal** is to reverse engineer the implementation details by utilizing SoftMC: an open-source FPGA-based memory controller

Methodology:

1. Create Hypothesis
2. Identify the Goals
3. Perform Case Studies

Inside the DRAM Chips

- The exact implementation details of **in-DRAM TRR mechanisms** are **unknown**
- The **goal** is to reverse engineer the implementation details by utilizing SoftMC: an open-source FPGA-based memory controller

Methodology:

1. **Create Hypothesis**
2. Identify the Goals
3. Perform Case Studies

Building Blocks and Hypotheses

TRR is an **umbrella** term *but* there are two requirements for supporting TRR.

- Sampler:
  - Tracks aggressor row activations
  - **Hypothesis:** The sampler has a *limited size*
  - **Implication:** The TRR mitigation can protect only a *limited number* of victim rows

- Inhibitor:
  - Prevents bit flips by refreshing victim rows
  - **Hypothesis:** The inhibitor acts at refresh time
  - **Implication:** Only a *limited number* of target rows can be refreshed within a refresh command

Inside the DRAM Chips

- The exact implementation details of **in-DRAM TRR mechanisms** are **unknown**
- The **goal** is to reverse engineer the implementation details by utilizing SoftMC: an open-source FPGA-based memory controller

Methodology:

1. Create Hypothesis
2. **Identify the Goals**
3. Perform Case Studies

The Goals

- What is the **size** of the sampler?
- How does the sampler **track** aggressor rows?
- How does the inhibitor **work**? Can it prevent bit flips?

Inside the DRAM Chips

- The exact implementation details of **in-DRAM TRR mechanisms** are **unknown**
- The **goal** is to reverse engineer the implementation details by utilizing SoftMC: an open-source FPGA-based memory controller

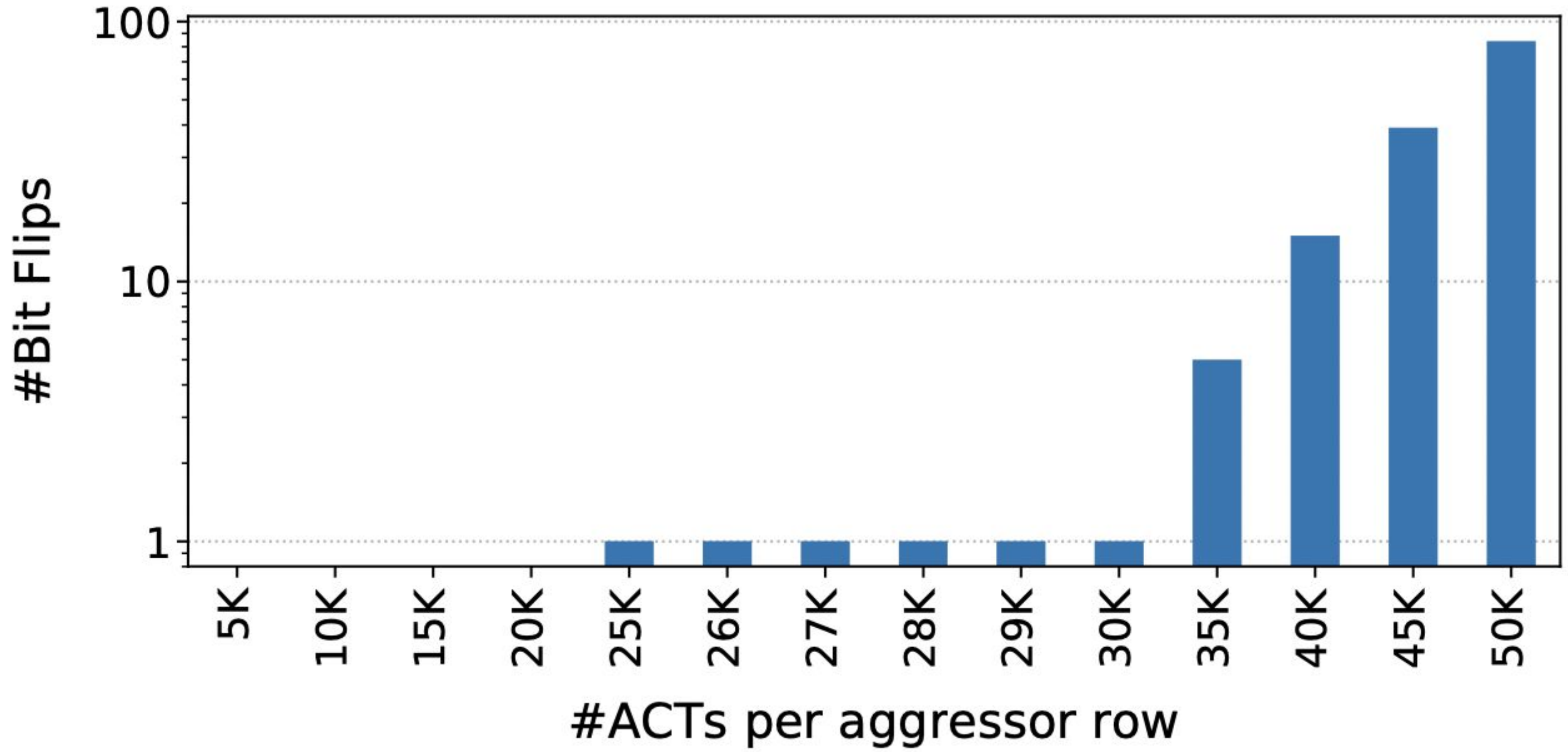
Methodology:

1. Create Hypothesis
2. Identify the Goals
3. **Perform Case Studies**

Case I: Module C12

How many activations we require to actually cause bit flips in DDR4 chips?

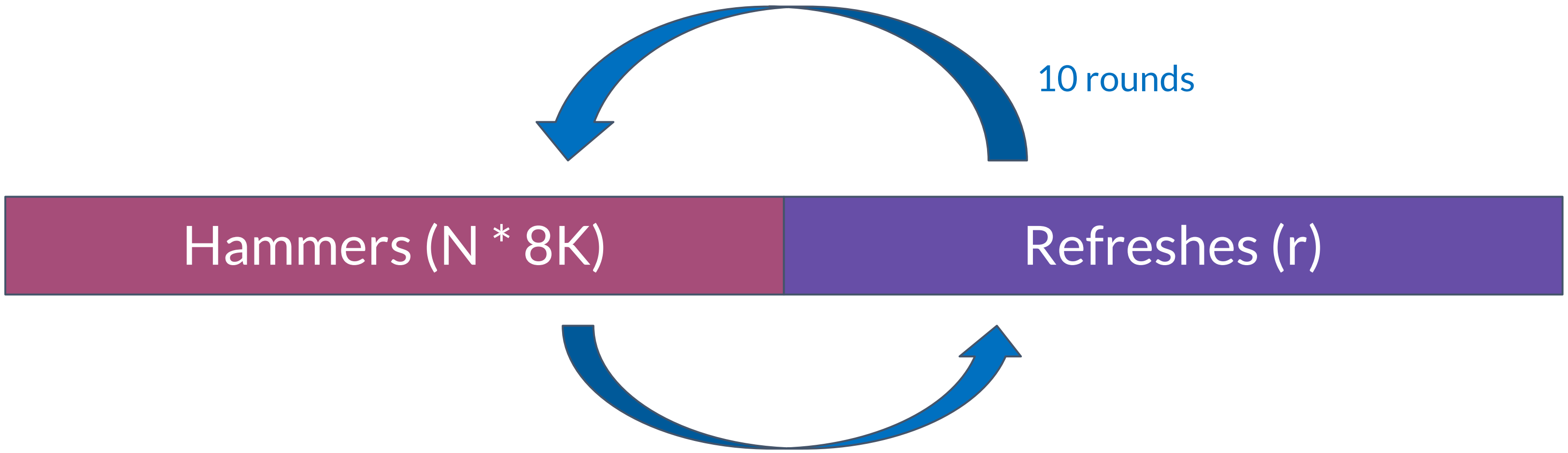
Methodology: Disable refresh command and observe the number of bit flips.



Case I: Module C12 – Mastering Refresh

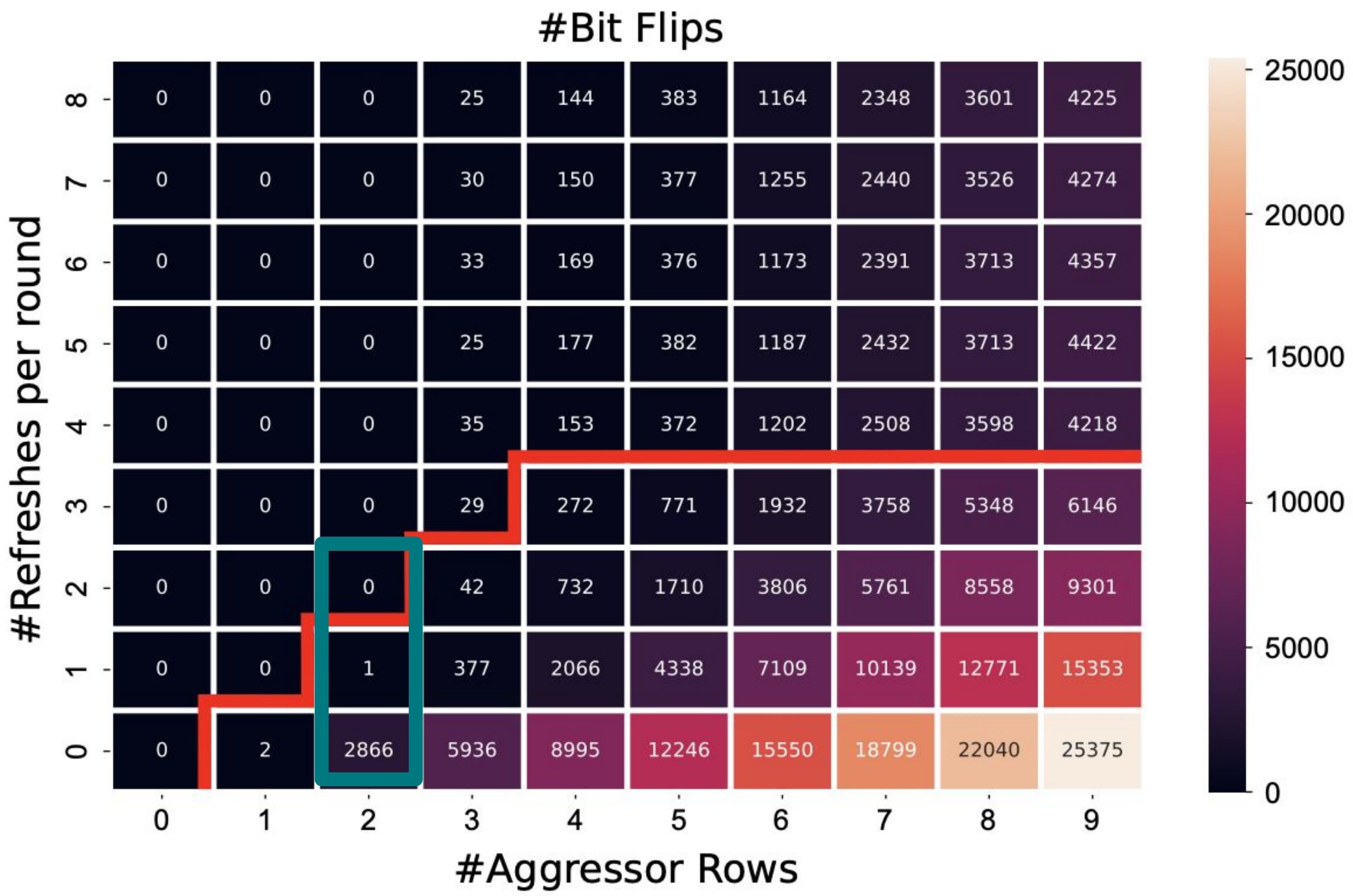
What is the size of the sampler?

- Vary the number of aggressor rows (N) and refreshes (r)
- Perform a series of activations (8K) for each aggressor row
- Perform 10 rounds of series of **hammers** and **refreshes**



Case I: Module C12 – Mastering Refresh

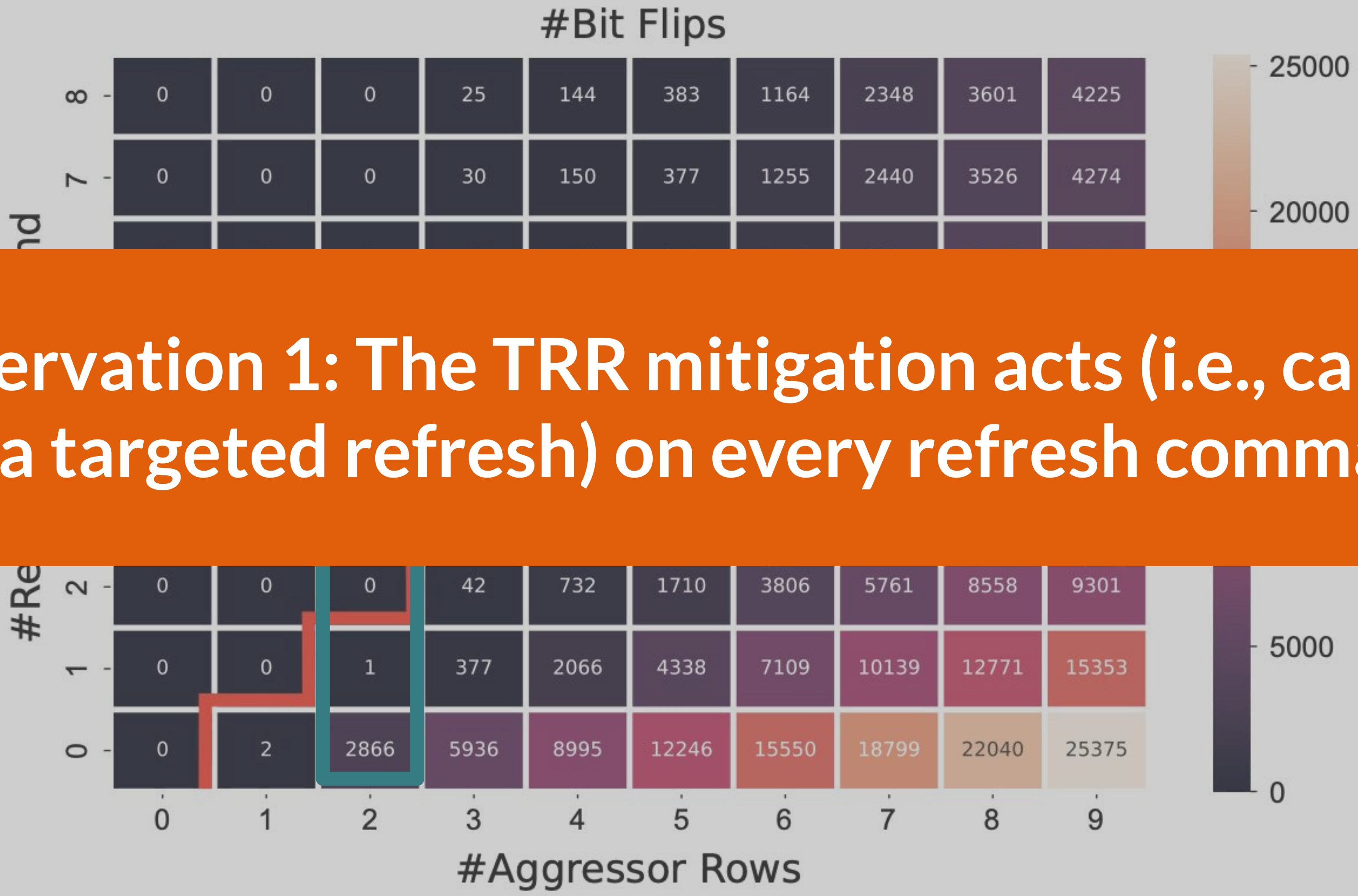
Adding a refresh reduces the number of bit flips



#Refreshes per round	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 I: Module C12 – Mastering Refresh

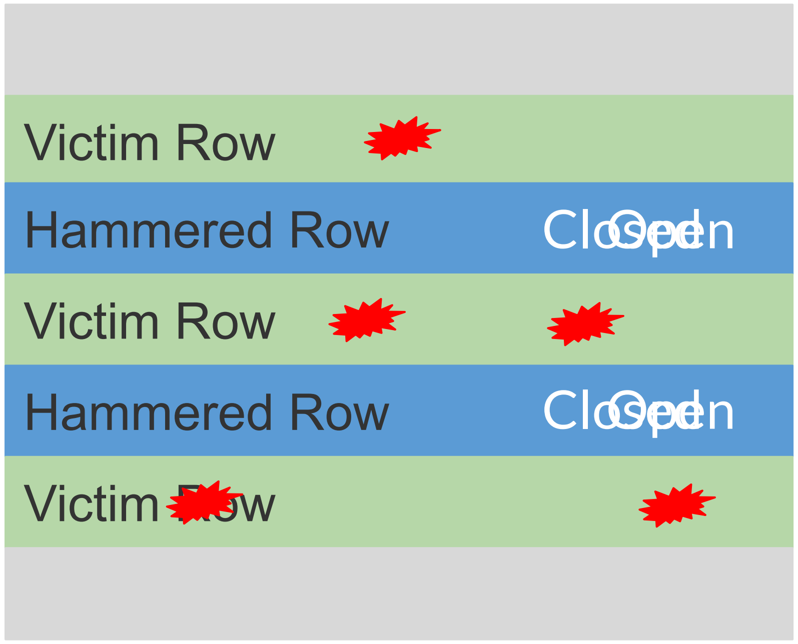
Adding a refresh reduces the number of bit flips

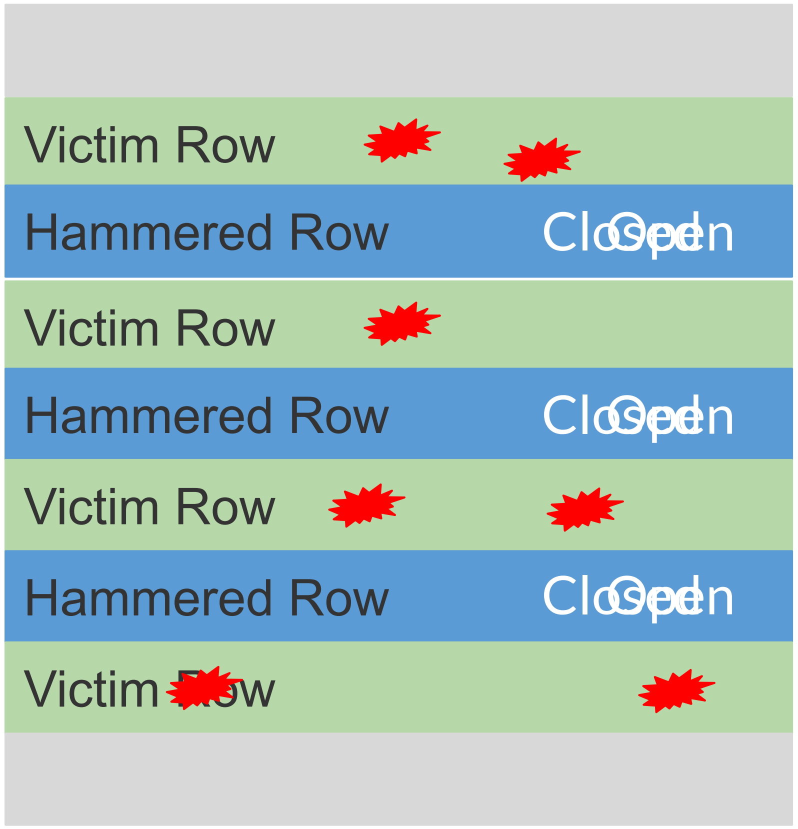


Observation 1: The TRR mitigation acts (i.e., carries out a targeted refresh) on every refresh command.

Increasing the number of Hammered Rows

increasing # of aggressor row –> increasing # of victim row

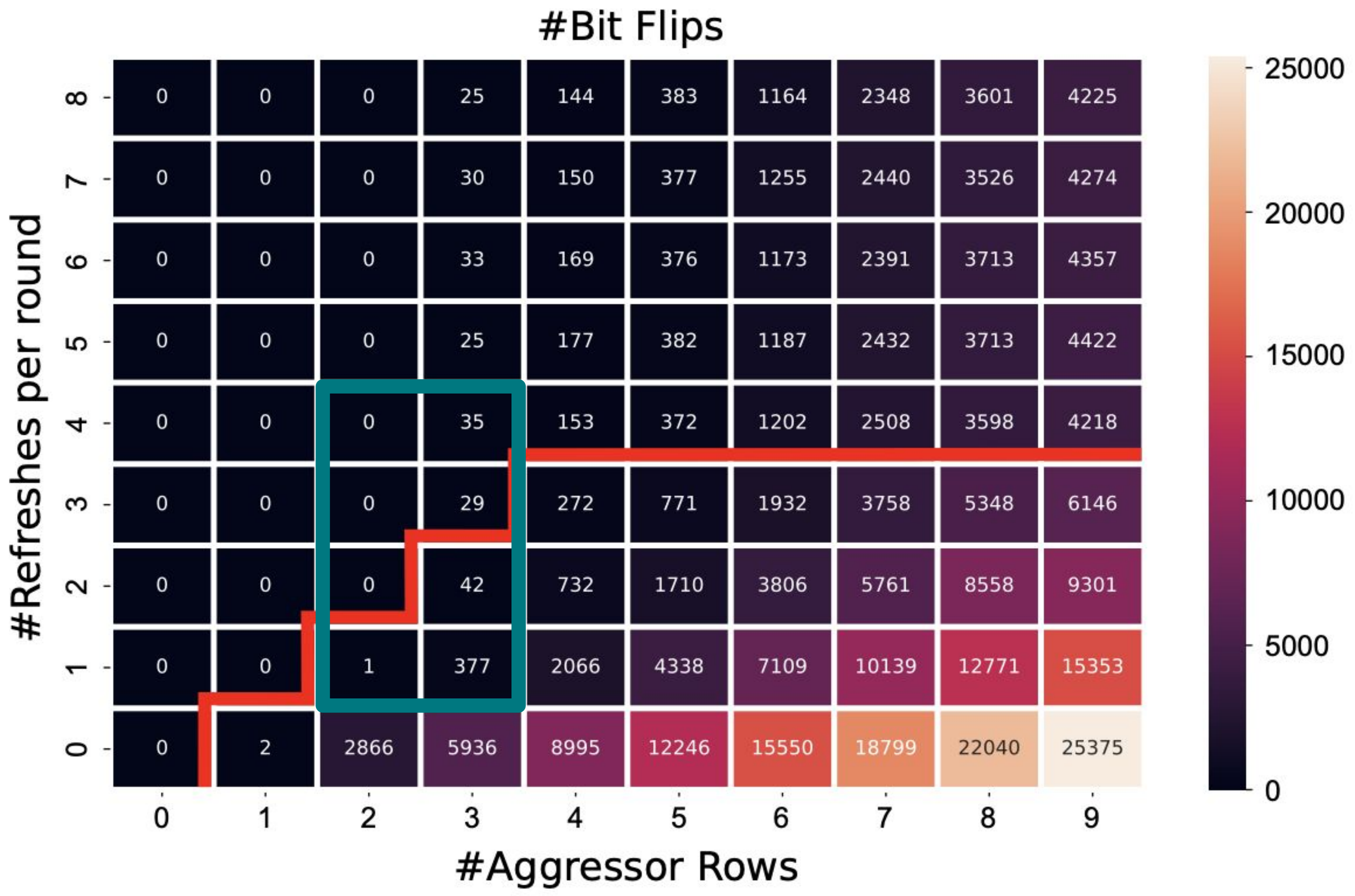




SAFARI

Case I: Module C12 – Mastering Refresh

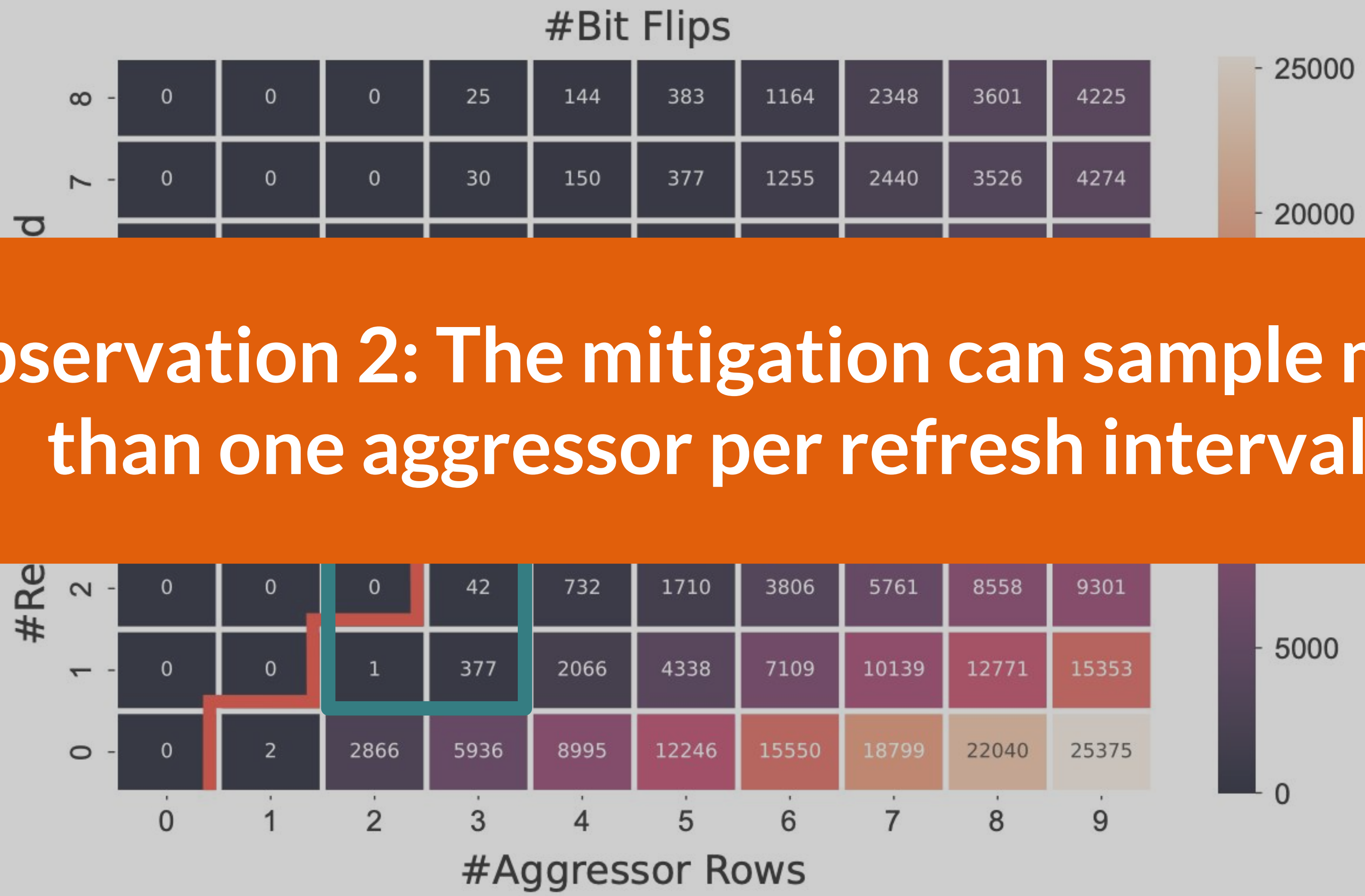
The number of bit flips stabilizes when N = r



#Refreshes 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 I: Module C12 – Mastering Refresh

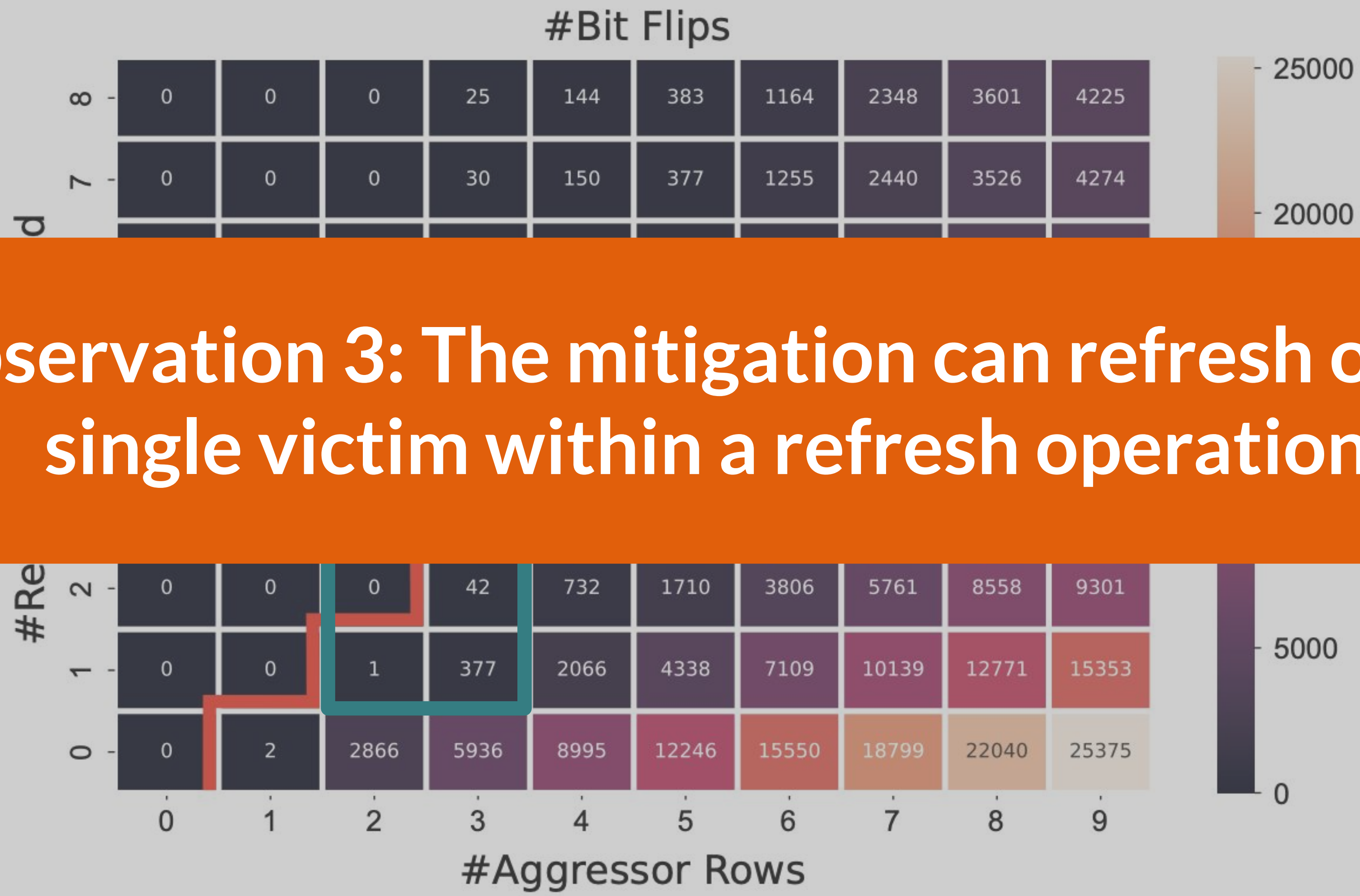
The number of bit flips stabilizes when N = r



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

Case I: Module C12 – Mastering Refresh

The number of bit flips stabilizes when N = r



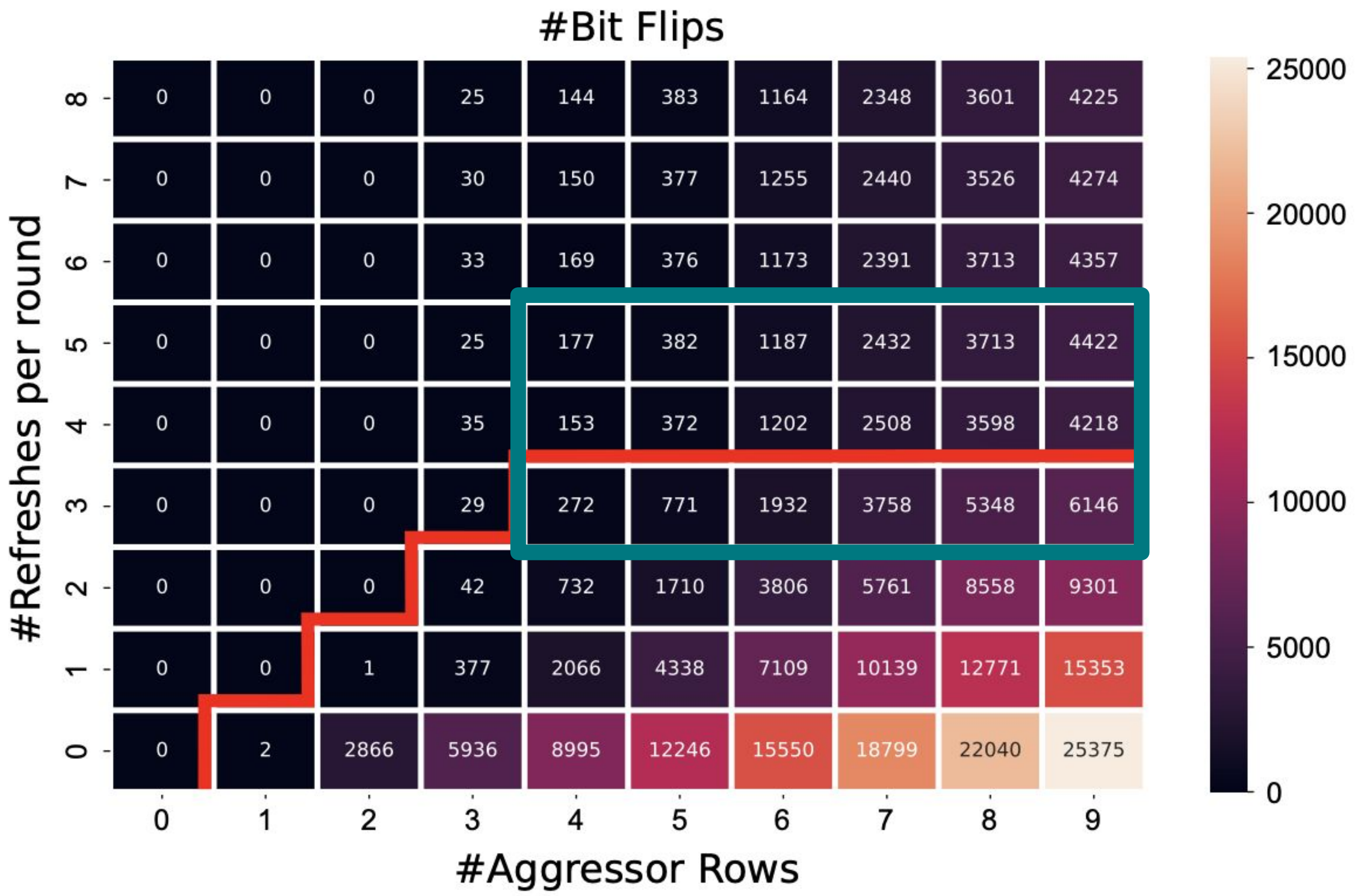
	0	1	2	3	4	5	6	7	8	9
8	0	0	0	25	144	383	1164	2348	3601	4225
7	0	0	0	30	150	377	1255	2440	3526	4274
2	0	0	0	42	732	1710	3806	5761	8558	9301
1	0	0	1	377	2066	4338	7109	10139	12771	15353
0	0	2	2866	5936	8995	12246	15550	18799	22040	25375

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

Case I: Module C12 – Mastering Refresh

RECALL: What is the **size** of the sampler?

The # of bit flips stabilizes for r < N, revealing the sampler size (4).

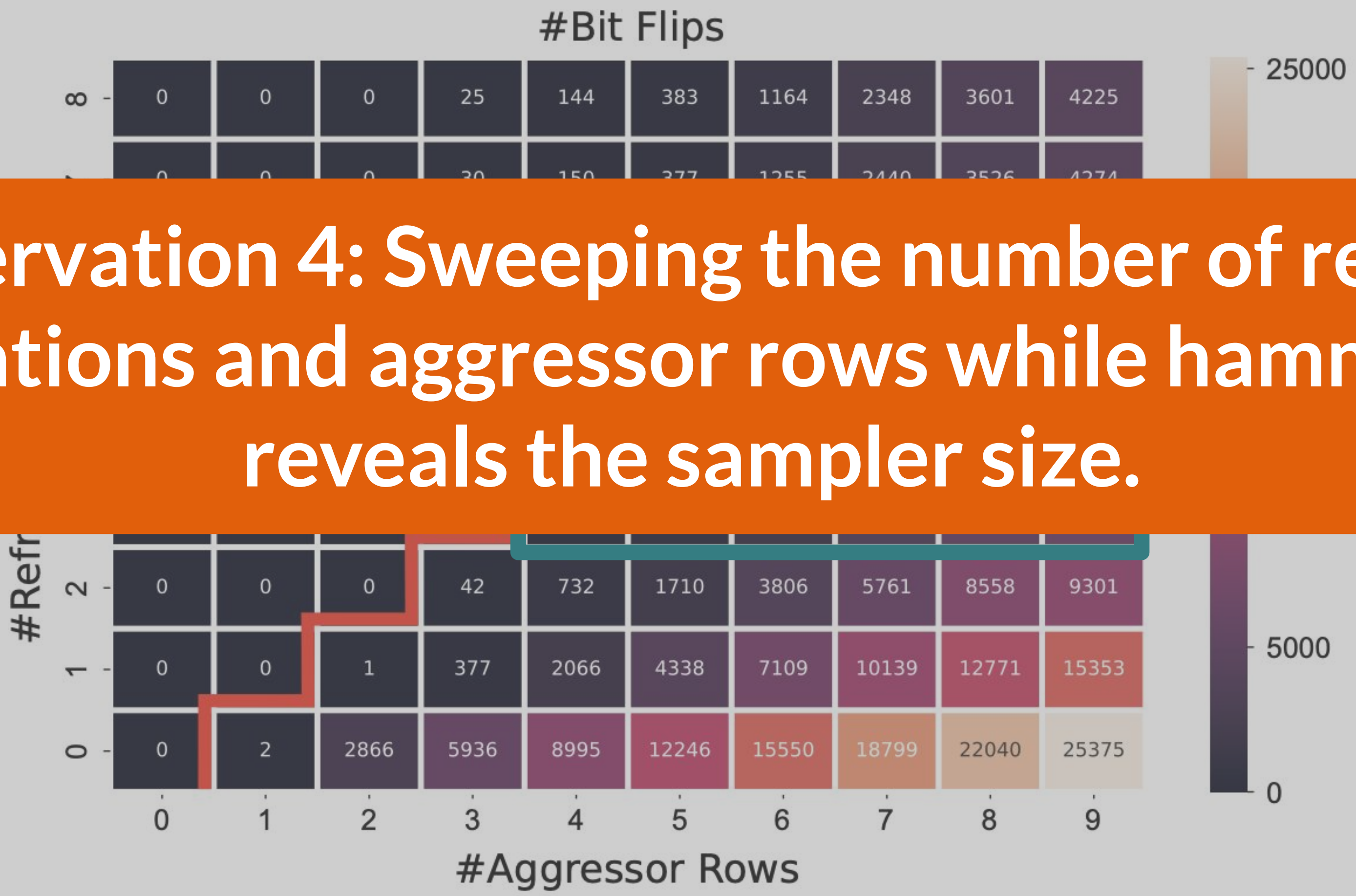


#Refreshes per round \ #Aggressor Rows	0	1	2	3	4	5	6	7	8	9
8	0	0	0	25	144	383	1164	2348	3601	4225
7	0	0	0	30	150	377	1255	2440	3526	4274
6	0	0	0	33	169	376	1173	2391	3713	4357
5	0	0	0	25	177	382	1187	2432	3713	4422
4	0	0	0	35	153	372	1202	2508	3598	4218
3	0	0	0	29	272	771	1932	3758	5348	6146
2	0	0	0	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 I: Module C12 – Mastering Refresh

RECALL: What is the **size** of the sampler?

The # of bit flips stabilizes for r < N, revealing the sampler size (4).

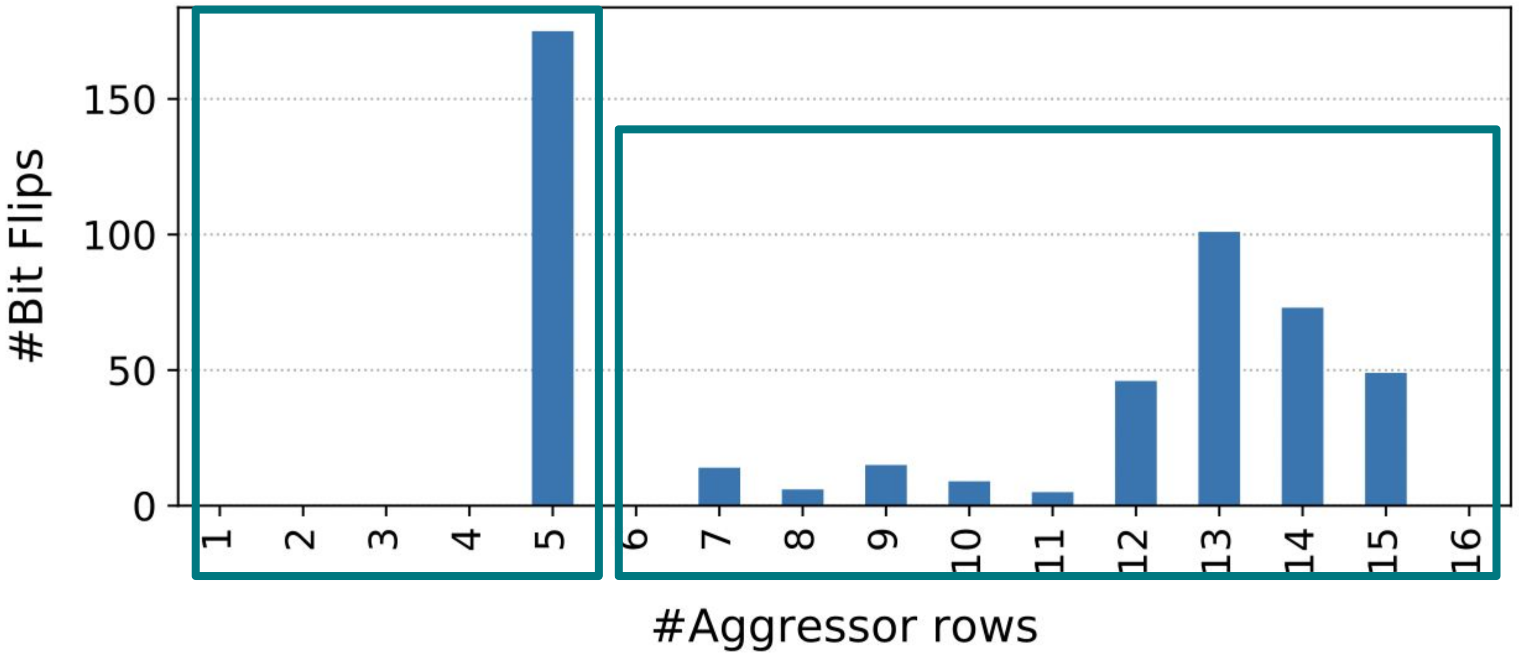


Observation 4: Sweeping the number of refresh operations and aggressor rows while hammering reveals the sampler size.

Case I: Module C12 – Mastering Refresh

Hammering more than 4 rows should circumvent the mitigation.

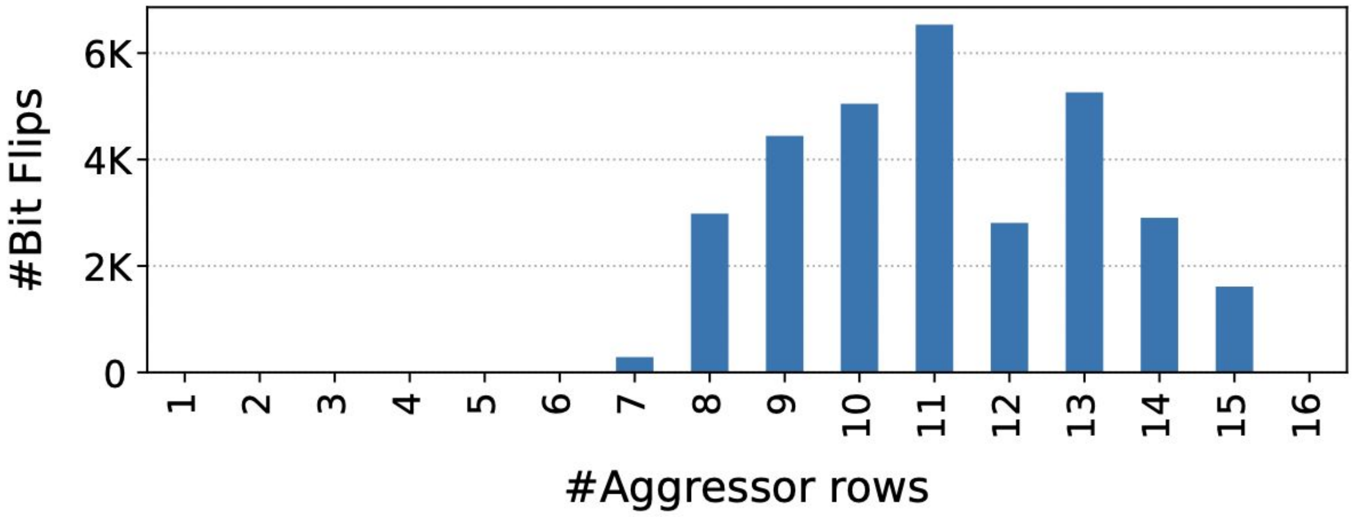
- The mitigation mechanism is overwhelmed by hammering 5 rows
- It is not clear why the number of bit flips is changing drastically after number of aggressor rows > 5



Case II: Module A15

Goal: understand different flavors of in-DRAM TRR

- Prior observations verified
- The sampler size: 6



Is it possible to revive the **more efficient double-sided** RowHammer attack?

Case II: Module A15

Methodology:

1. Find the minimal set of dummy rows
   - to **trick** the mitigation mechanism

2. Select a **specific** victim row
   - Find the threshold of hammers to observe a bit flip

3. Modify
   - the distribution of activations across aggressor and dummy rows
   - the number of dummies starting from sampler size

Case II: Module A15

Methodology:

1. Find the minimal set of dummy rows
   - to **trick** the mitigation mechanism

No bit flips observed

   - the distribution of activations across aggressor and dummy rows
   - the number of dummies starting from sampler size

Case II: Module A15

What might be going wrong?

- ***DRAM command order dependency:***
  - How does sampler act? (on specific DRAM commands?)
  - For A15, the samper records first α activations after the refresh
- ***Address dependency:***
  - The number of bit flips depend on the address of dummy rows
  - This implies the design of the sampler is optimized to reduce storage cost

Case II: Module A15

What might be going wrong?

Observation 5: The sampler records row activations at specific commands and likely at specific ordering of commands (command-order-based sampling).

- *Address dependency:*

Observation 6: The sampling mechanism is affected by the addresses of aggressor rows (row-address-dependent sampling).

Running on the CPU

Challenges:

- The sampling algorithm of TRR is command order and address dependent
- Memory controller optimizes and reorders the requests

Methodology:

- Carry out specific series of activations after Refresh command
- The RowHammer access pattern needs to be synchronized with the Refresh command

much fewer bit flips observed compared to SoftMC

Running on the CPU

Challenges:

- The sampling algorithm of TRR is command order and address dependent
- Memory controller optimizes and reorders the requests

We need a better solution for finding effective access patterns that trigger bit flips on TRR-protected DDR4 chips.

Refresh command.

much fewer bit flips observed compared to SoftMC

TRRespass: A TRR-Aware Rowfuzzer

Black-box RowHammer test suite that generates effective access patterns to bypass in-DRAM TRR solutions

It's design consists of three different components:

1. Cardinality
2. Location
3. Fuzzing Strategy

TRRespass: A TRR-Aware Rowfuzzer

1. Cardinality
   - The number of aggressor rows
   - The number of aggressor rows (*typically high*) required to overflow the sampler *varies* across modules
   - The number of activations per refresh interval is limited
2. Location
3. Fuzzing Strategy

TRRespass: A TRR-Aware Rowfuzzer

1. Cardinality
   - The number of aggressor rows
   - The number of aggressor rows (*typically high*) required to overflow the sampler *varies* across modules
   - The number of activations per refresh interval is limited
2. Location
   - sampler may have address dependency: randomize the location of aggressors
3. Fuzzing Strategy

TRRespass: A TRR-Aware Rowfuzzer

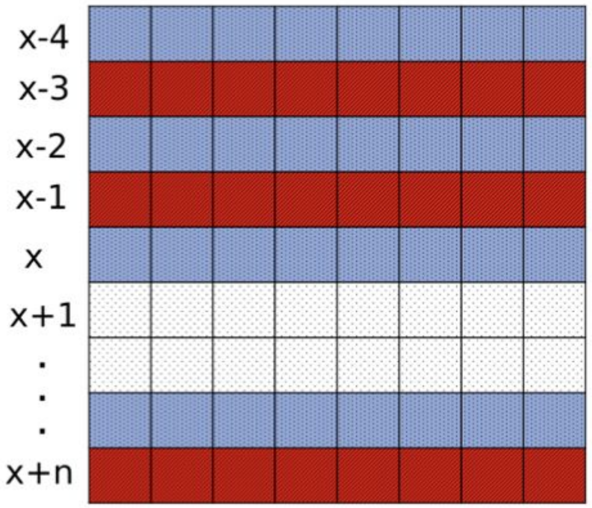
1. Cardinality
   - The number of aggressor rows
   - The number of aggressor rows (*typically high*) required to overflow the sampler *varies* across modules
   - The number of activations per refresh interval is limited
2. Location
   - sampler may have address dependency: randomize the location of aggressors
3. Fuzzing Strategy
   - The access patterns generated are evaluated based on the number of unique bit flips they generate
   - Randomize the cardinality and location parameters
   - Test a chunk of memory for 3 × refresh period

Evaluation Methodology

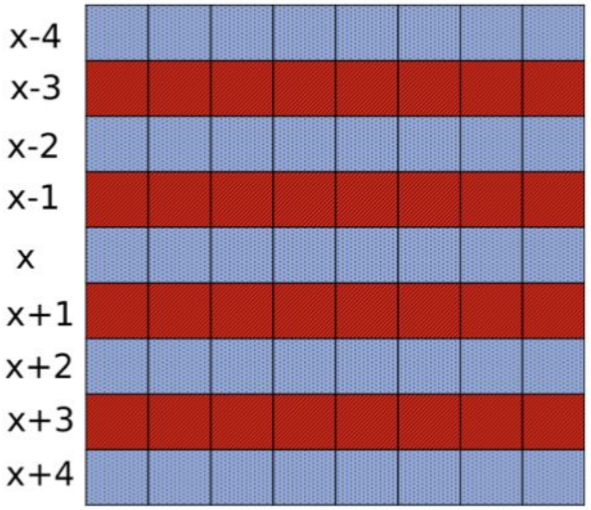
- Device:
  - Intel Core i7-7700K, mounted on an ASUS STRIX Z270G motherboard
- Modules:
  - 42 DDR4 DRAM modules from all **3 major** manufacturers are tested
- Methodology:
  - Perform a sweep over 256MB of contiguous physical memory (128 adjacent rows from each bank)
  - Examine RowHammer bit flips for both true and anti cells

TRRespass-ing Over DDR4

assisted double sided -> many sided



(a) Assisted double-sided



(b) 4-sided

TRRespass-ing Over DDR4

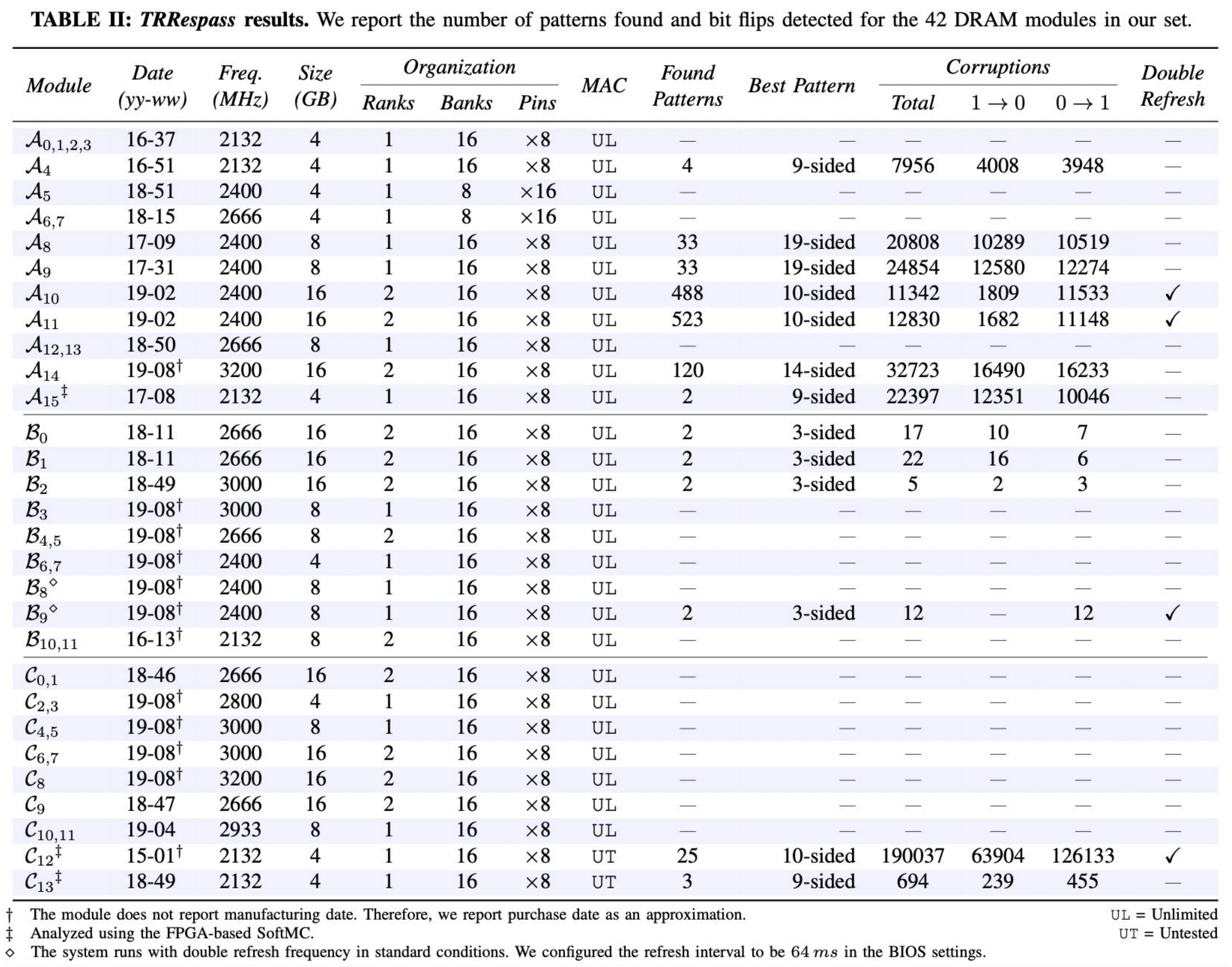


TABLE II: *TRRespass* results. We report the number of patterns found and bit flips detected for the 42 DRAM modules in our set.

Module	*Date (yy-ww)*	*Freq. (MHz)*	*Size (GB)*	*Organization*			*MAC*	*Found Patterns*	*Best Pattern*	*Corruptions*			*Double Refresh*
				Ranks	*Banks*	*Pins*				*Total*	$1 \rightarrow 0$	$0 \rightarrow 1$	
$\mathcal{A}_{0,1,2,3}$	16-37	2132	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_{4}$	16-51	2132	4	1	16	×8	UL	4	9-sided	7956	4008	3948	—
$\mathcal{A}_{5}$	18-51	2400	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_{6,7}$	18-15	2666	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_{8}$	17-09	2400	8	1	16	×8	UL	33	19-sided	20808	10289	10519	—
$\mathcal{A}_{9}$	17-31	2400	8	1	16	×8	UL	33	19-sided	24854	12580	12274	—
$\mathcal{A}_{10}$	19-02	2400	16	2	16	×8	UL	488	10-sided	11342	1809	11533	✓
$\mathcal{A}_{11}$	19-02	2400	16	2	16	×8	UL	523	10-sided	12830	1682	11148	✓
$\mathcal{A}_{12,13}$	18-50	2666	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_{14}$	19-08[†]	3200	16	2	16	×8	UL	120	14-sided	32723	16490	16233	—
$\mathcal{A}_{15}$[‡]	17-08	2132	4	1	16	×8	UL	2	9-sided	22397	12351	10046	—
$\mathcal{B}_{0}$	18-11	2666	16	2	16	×8	UL	2	3-sided	17	10	7	—
$\mathcal{B}_{1}$	18-11	2666	16	2	16	×8	UL	2	3-sided	22	16	6	—
$\mathcal{B}_{2}$	18-49	3000	16	2	16	×8	UL	2	3-sided	5	2	3	—
$\mathcal{B}_{3}$	19-08[†]	3000	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{4,5}$	19-08[†]	2666	8	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{6,7}$	19-08[†]	2400	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{8}$[◇]	19-08[†]	2400	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{9}$[◇]	19-08[†]	2400	8	1	16	×8	UL	2	3-sided	12	—	12	✓
$\mathcal{B}_{10,11}$	16-13[†]	2132	8	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{0,1}$	18-46	2666	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{2,3}$	19-08[†]	2800	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{4,5}$	19-08[†]	3000	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{6,7}$	19-08[†]	3000	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{8}$	19-08[†]	3200	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{9}$	18-47	2666	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{10,11}$	19-04	2933	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{12}$[‡]	15-01[†]	2132	4	1	16	×8	UT	25	10-sided	190037	63904	126133	✓
$\mathcal{C}_{13}$[‡]	18-49	2132	4	1	16	×8	UT	3	9-sided	694	239	455	—

† The module does not report manufacturing date. Therefore, we report purchase date as an approximation.
‡ Analyzed using the FPGA-based SoftMC.
◇ The system runs with double refresh frequency in standard conditions. We configured the refresh interval to be 64 ms in the BIOS settings.

`UL` = Unlimited
`UT` = Untested

TRRespass–ing Over DDR4

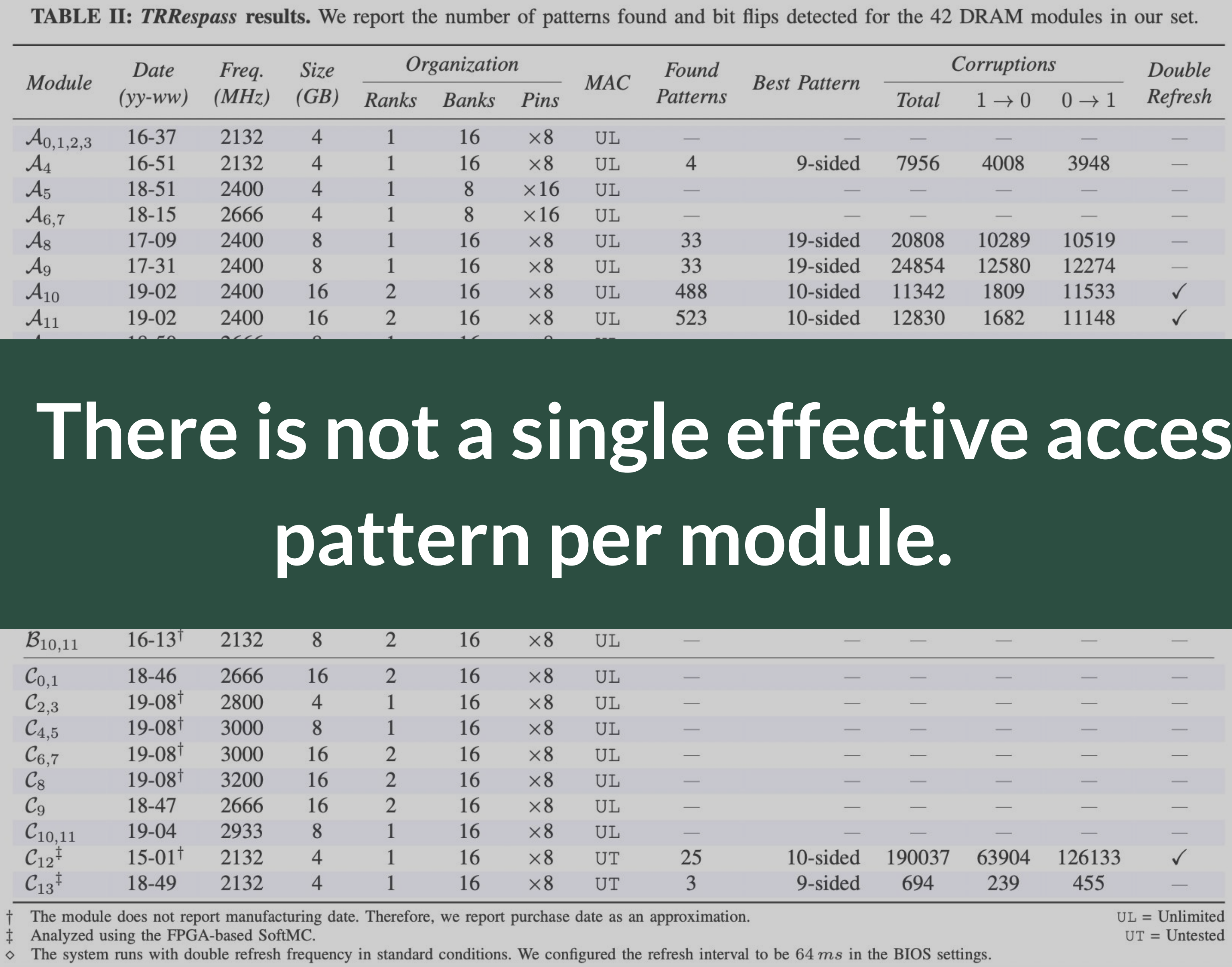


TABLE II: ***TRRespass* results.** We report the number of patterns found and bit flips detected for the 42 DRAM modules in our set.

Module	*Date (yy-ww)*	*Freq. (MHz)*	*Size (GB)*	*Organization*			*MAC*	*Found Patterns*	*Best Pattern*	*Corruptions*			*Double Refresh*
				Ranks	*Banks*	*Pins*				*Total*	$1 \to 0$	$0 \to 1$	
$\mathcal{A}_{0,1,2,3}$	16-37	2132	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_{4}$	16-51	2132	4	1	16	×8	UL	4	9-sided	7956	4008	3948	—
$\mathcal{A}_{5}$	18-51	2400	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_{6,7}$	18-15	2666	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_{8}$	17-09	2400	8	1	16	×8	UL	33	19-sided	20808	10289	10519	—
$\mathcal{A}_{9}$	17-31	2400	8	1	16	×8	UL	33	19-sided	24854	12580	12274	—
$\mathcal{A}_{10}$	19-02	2400	16	2	16	×8	UL	488	10-sided	11342	1809	11533	✓
$\mathcal{A}_{11}$	19-02	2400	16	2	16	×8	UL	523	10-sided	12830	1682	11148	✓
$\mathcal{B}_{10,11}$	16-13†	2132	8	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{0,1}$	18-46	2666	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{2,3}$	19-08†	2800	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{4,5}$	19-08†	3000	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{6,7}$	19-08†	3000	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{8}$	19-08†	3200	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{9}$	18-47	2666	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{10,11}$	19-04	2933	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{12}$‡	15-01†	2132	4	1	16	×8	UT	25	10-sided	190037	63904	126133	✓
$\mathcal{C}_{13}$‡	18-49	2132	4	1	16	×8	UT	3	9-sided	694	239	455	—

† The module does not report manufacturing date. Therefore, we report purchase date as an approximation.
‡ Analyzed using the FPGA-based SoftMC.
◇ The system runs with double refresh frequency in standard conditions. We configured the refresh interval to be 64 ms in the BIOS settings.

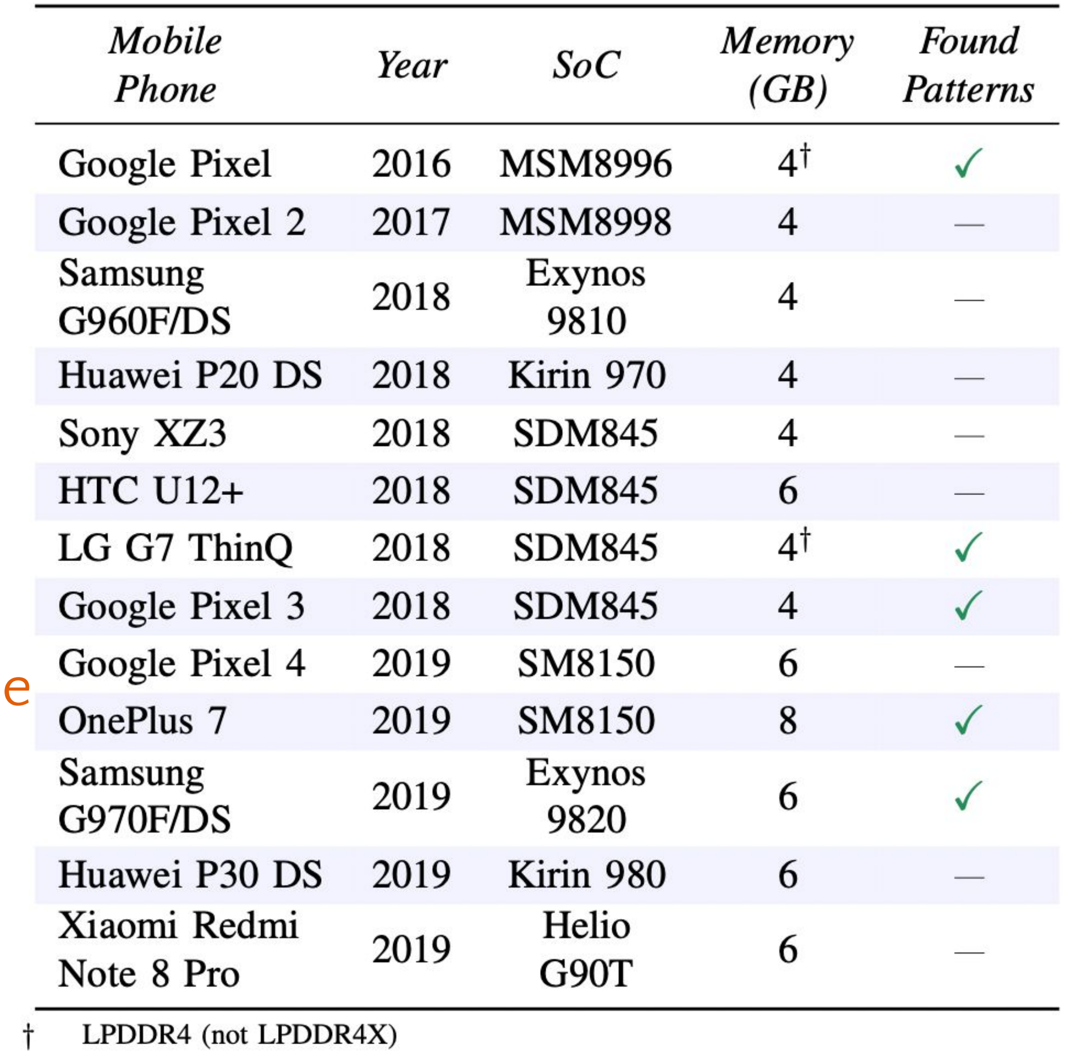
UL = Unlimited
UT = Untested

There is not a single effective access pattern per module.

TRRespass on LPDDR4(x)

- TRRespass discovers hammering patterns on 5 out out 12 devices

TRR-protected mobile platforms are still vulnerable to RowHammer



Mobile Phone	*Year*	*SoC*	*Memory (GB)*	*Found Patterns*
Google Pixel	2016	MSM8996	4[†]	✓
Google Pixel 2	2017	MSM8998	4	—
Samsung G960F/DS	2018	Exynos 9810	4	—
Huawei P20 DS	2018	Kirin 970	4	—
Sony XZ3	2018	SDM845	4	—
HTC U12+	2018	SDM845	6	—
LG G7 ThinQ	2018	SDM845	4[†]	✓
Google Pixel 3	2018	SDM845	4	✓
Google Pixel 4	2019	SM8150	6	—
OnePlus 7	2019	SM8150	8	✓
Samsung G970F/DS	2019	Exynos 9820	6	✓
Huawei P30 DS	2019	Kirin 980	6	—
Xiaomi Redmi Note 8 Pro	2019	Helio G90T	6	—

[†] LPDDR4 (not LPDDR4X)

Evaluation – Results

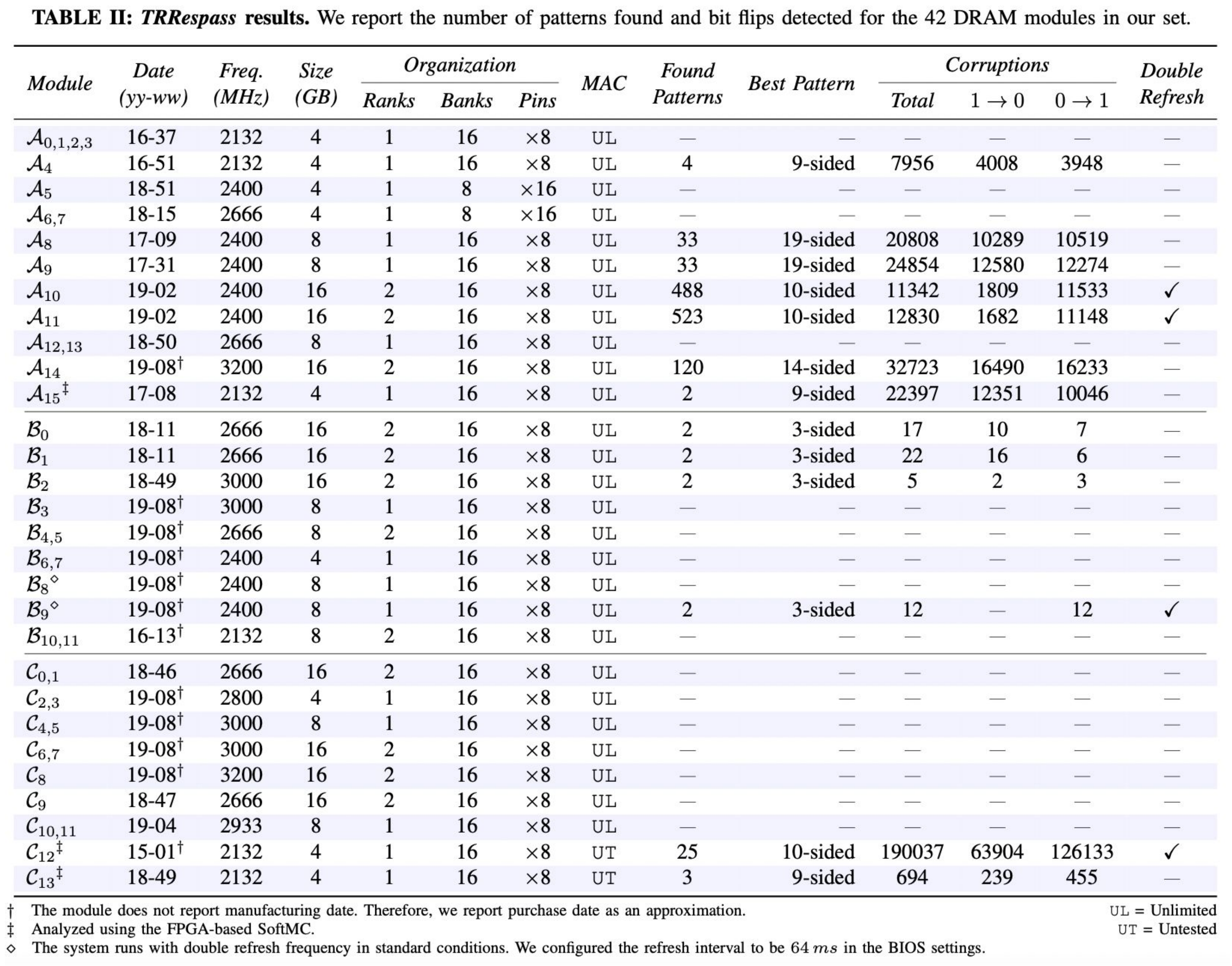


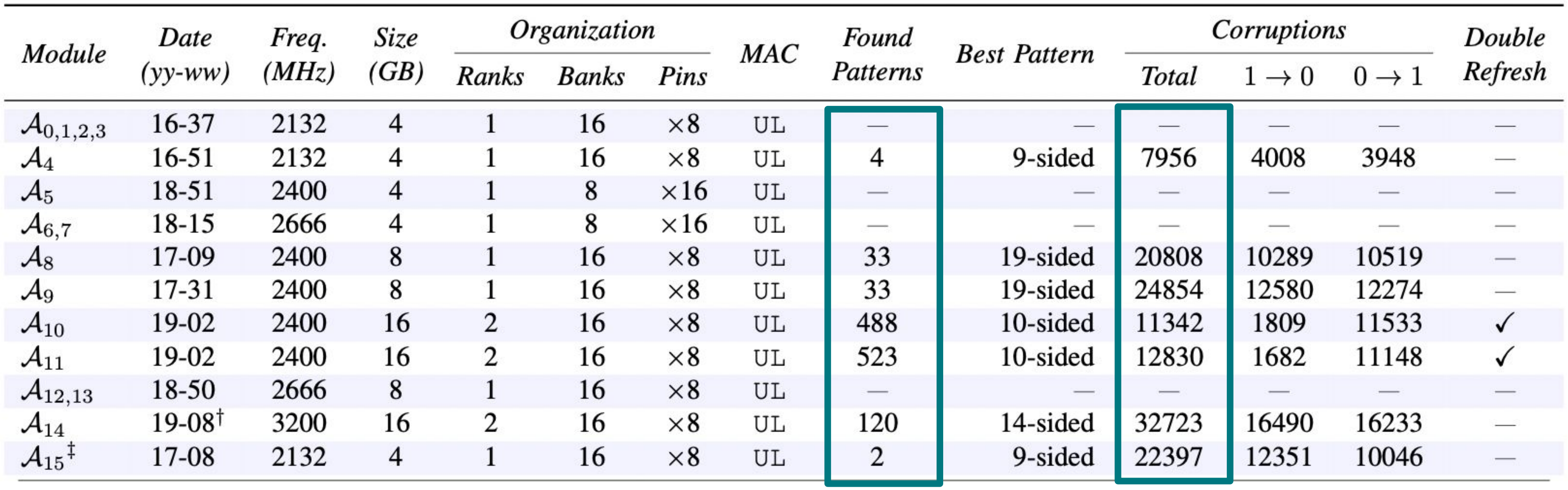
TABLE II: *TRRespass* results. We report the number of patterns found and bit flips detected for the 42 DRAM modules in our set.

Module	*Date (yy-ww)*	*Freq. (MHz)*	*Size (GB)*	*Organization*			*MAC*	*Found Patterns*	*Best Pattern*	*Corruptions*			*Double Refresh*
				Ranks	*Banks*	*Pins*				*Total*	$1 \to 0$	$0 \to 1$	
$\mathcal{A}_{0,1,2,3}$	16-37	2132	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_{4}$	16-51	2132	4	1	16	×8	UL	4	9-sided	7956	4008	3948	—
$\mathcal{A}_{5}$	18-51	2400	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_{6,7}$	18-15	2666	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_{8}$	17-09	2400	8	1	16	×8	UL	33	19-sided	20808	10289	10519	—
$\mathcal{A}_{9}$	17-31	2400	8	1	16	×8	UL	33	19-sided	24854	12580	12274	—
$\mathcal{A}_{10}$	19-02	2400	16	2	16	×8	UL	488	10-sided	11342	1809	11533	✓
$\mathcal{A}_{11}$	19-02	2400	16	2	16	×8	UL	523	10-sided	12830	1682	11148	✓
$\mathcal{A}_{12,13}$	18-50	2666	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_{14}$	19-08†	3200	16	2	16	×8	UL	120	14-sided	32723	16490	16233	—
$\mathcal{A}_{15}$‡	17-08	2132	4	1	16	×8	UL	2	9-sided	22397	12351	10046	—
$\mathcal{B}_{0}$	18-11	2666	16	2	16	×8	UL	2	3-sided	17	10	7	—
$\mathcal{B}_{1}$	18-11	2666	16	2	16	×8	UL	2	3-sided	22	16	6	—
$\mathcal{B}_{2}$	18-49	3000	16	2	16	×8	UL	2	3-sided	5	2	3	—
$\mathcal{B}_{3}$	19-08†	3000	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{4,5}$	19-08†	2666	8	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{6,7}$	19-08†	2400	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{8}$◇	19-08†	2400	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{9}$◇	19-08†	2400	8	1	16	×8	UL	2	3-sided	12	—	12	✓
$\mathcal{B}_{10,11}$	16-13†	2132	8	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{0,1}$	18-46	2666	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{2,3}$	19-08†	2800	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{4,5}$	19-08†	3000	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{6,7}$	19-08†	3000	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{8}$	19-08†	3200	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{9}$	18-47	2666	16	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{10,11}$	19-04	2933	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{C}_{12}$‡	15-01†	2132	4	1	16	×8	UT	25	10-sided	190037	63904	126133	✓
$\mathcal{C}_{13}$‡	18-49	2132	4	1	16	×8	UT	3	9-sided	694	239	455	—

† The module does not report manufacturing date. Therefore, we report purchase date as an approximation.
‡ Analyzed using the FPGA-based SoftMC.
◇ The system runs with double refresh frequency in standard conditions. We configured the refresh interval to be 64 ms in the BIOS settings.

UL = Unlimited
UT = Untested

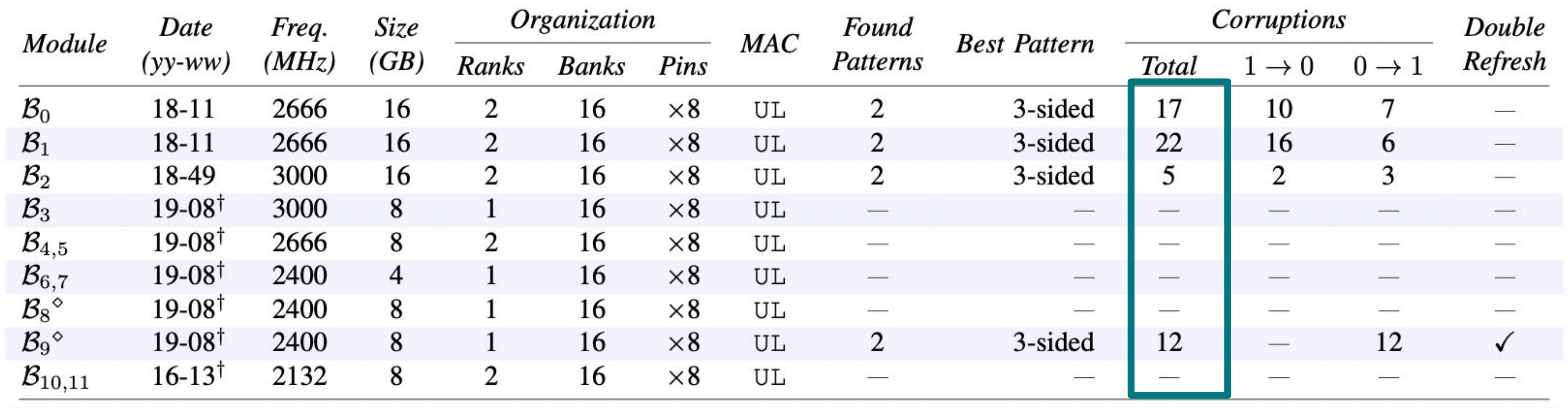
Evaluation – Results



Module	*Date (yy-ww)*	*Freq. (MHz)*	*Size (GB)*	*Organization*			*MAC*	*Found Patterns*	*Best Pattern*	*Corruptions*			*Double Refresh*
				Ranks	*Banks*	*Pins*				*Total*	$1 \rightarrow 0$	$0 \rightarrow 1$	
$\mathcal{A}_{0,1,2,3}$	16-37	2132	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_4$	16-51	2132	4	1	16	×8	UL	4	9-sided	7956	4008	3948	—
$\mathcal{A}_5$	18-51	2400	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_{6,7}$	18-15	2666	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_8$	17-09	2400	8	1	16	×8	UL	33	19-sided	20808	10289	10519	—
$\mathcal{A}_9$	17-31	2400	8	1	16	×8	UL	33	19-sided	24854	12580	12274	—
$\mathcal{A}_{10}$	19-02	2400	16	2	16	×8	UL	488	10-sided	11342	1809	11533	✓
$\mathcal{A}_{11}$	19-02	2400	16	2	16	×8	UL	523	10-sided	12830	1682	11148	✓
$\mathcal{A}_{12,13}$	18-50	2666	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_{14}$	19-08[†]	3200	16	2	16	×8	UL	120	14-sided	32723	16490	16233	—
$\mathcal{A}_{15}$[‡]	17-08	2132	4	1	16	×8	UL	2	9-sided	22397	12351	10046	—

- TRRespass can recover *multiple* effective access patterns for 7 of the 16 modules.
- TRRespass found more than 16K bit flips on average across the 7 vulnerable modules.
- The number of activations required to create bit flips is quite low (~45K row activations).

Evaluation – Results



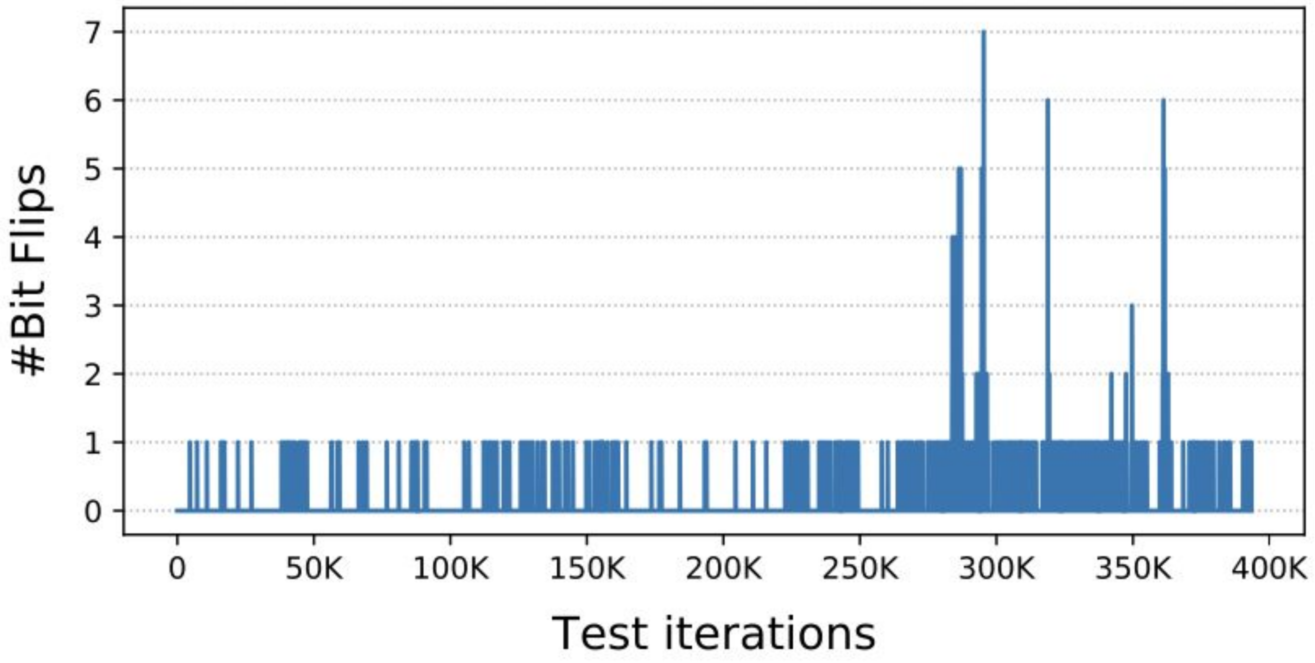
Module	*Date (yy-ww)*	*Freq. (MHz)*	*Size (GB)*	*Organization*			*MAC*	*Found Patterns*	*Best Pattern*	*Corruptions*			*Double Refresh*
				Ranks	*Banks*	*Pins*				*Total*	$1 \rightarrow 0$	$0 \rightarrow 1$	
$\mathcal{B}_0$	18-11	2666	16	2	16	×8	UL	2	3-sided	17	10	7	—
$\mathcal{B}_1$	18-11	2666	16	2	16	×8	UL	2	3-sided	22	16	6	—
$\mathcal{B}_2$	18-49	3000	16	2	16	×8	UL	2	3-sided	5	2	3	—
$\mathcal{B}_3$	19-08†	3000	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{4,5}$	19-08†	2666	8	2	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_{6,7}$	19-08†	2400	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_8$◇	19-08†	2400	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{B}_9$◇	19-08†	2400	8	1	16	×8	UL	2	3-sided	12	—	12	✓
$\mathcal{B}_{10,11}$	16-13†	2132	8	2	16	×8	UL	—	—	—	—	—	—

- The number of bit flips observed is significantly lower compared to vendor A.
- Bypassing the TRR mitigation on these modules is non-trivial.
- Further experiments?

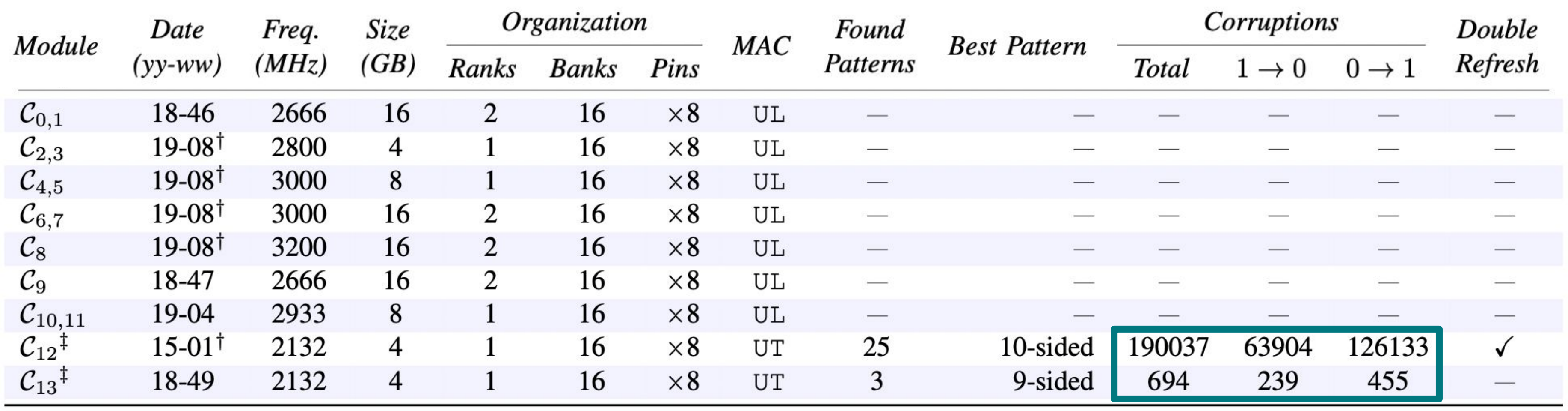
Evaluation – Results

What happens when the same RowHammer experiment using the aggressor rows that are known to be able to cause bit flips is repeated for multiple iterations?

Varying number of bit flips observed



Evaluation – Results



Module	*Date (yy-ww)*	*Freq. (MHz)*	*Size (GB)*	*Organization*			*MAC*	*Found Patterns*	*Best Pattern*	*Corruptions*			*Double Refresh*
				Ranks	*Banks*	*Pins*				*Total*	$1 \rightarrow 0$	$0 \rightarrow 1$	
$\mathcal{C}_{0,1}$	18-46	2666	16	2	16	×8	`UL`	—	—	—	—	—	—
$\mathcal{C}_{2,3}$	19-08[†]	2800	4	1	16	×8	`UL`	—	—	—	—	—	—
$\mathcal{C}_{4,5}$	19-08[†]	3000	8	1	16	×8	`UL`	—	—	—	—	—	—
$\mathcal{C}_{6,7}$	19-08[†]	3000	16	2	16	×8	`UL`	—	—	—	—	—	—
$\mathcal{C}_{8}$	19-08[†]	3200	16	2	16	×8	`UL`	—	—	—	—	—	—
$\mathcal{C}_{9}$	18-47	2666	16	2	16	×8	`UL`	—	—	—	—	—	—
$\mathcal{C}_{10,11}$	19-04	2933	8	1	16	×8	`UL`	—	—	—	—	—	—
$\mathcal{C}_{12}$[‡]	15-01[†]	2132	4	1	16	×8	`UT`	25	10-sided	190037	63904	126133	✓
$\mathcal{C}_{13}$[‡]	18-49	2132	4	1	16	×8	`UT`	3	9-sided	694	239	455	—

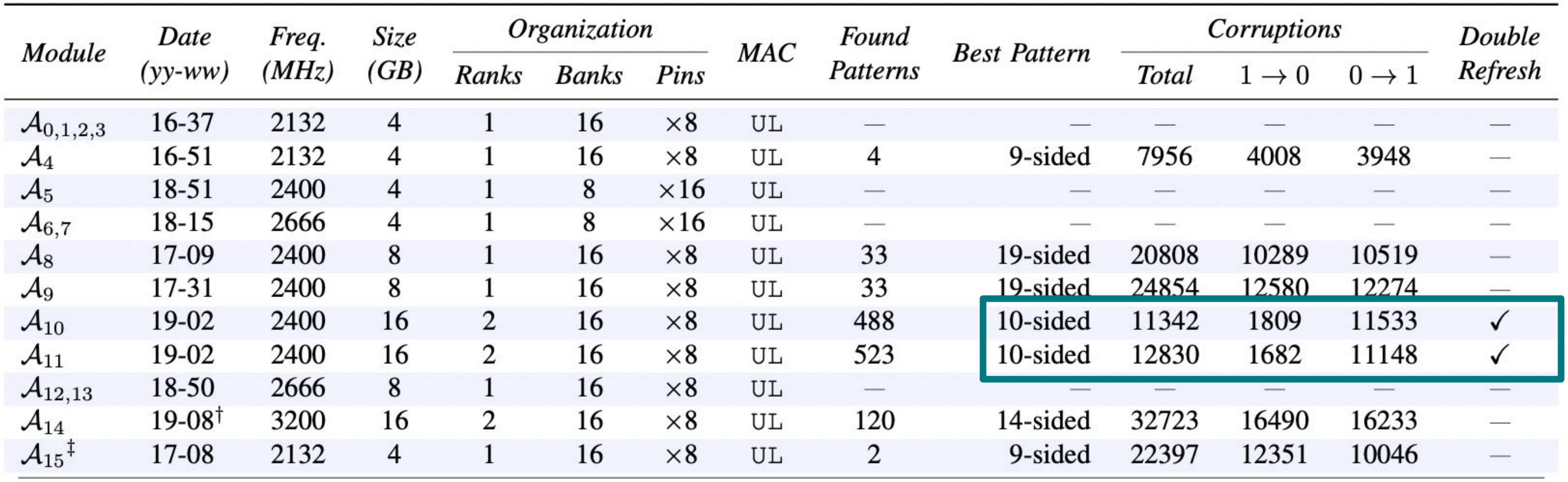
- The number of bit flips observed decreased over years.

in DRAM TRR implementation has improved over time

Evaluation – Increasing Refresh Rate

Doubling/quadrupling refresh rate is one mitigation mechanism.

Increasing refresh rate might also improve the effectiveness of TRR.



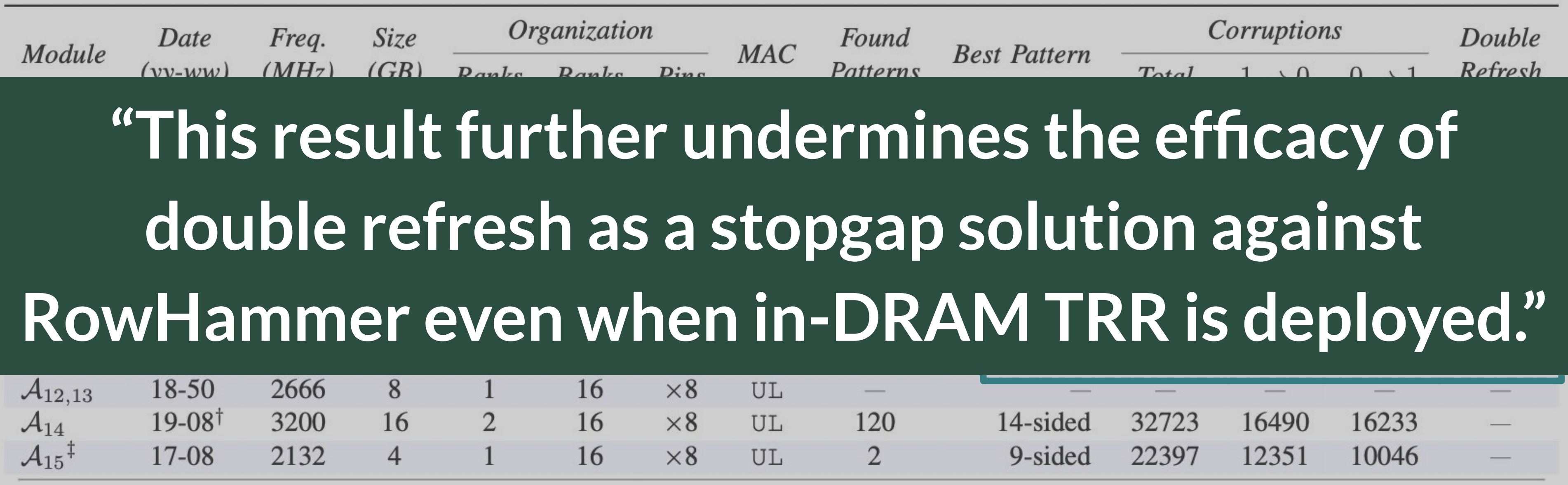
Module	*Date (yy-ww)*	*Freq. (MHz)*	*Size (GB)*	*Organization*			*MAC*	*Found Patterns*	*Best Pattern*	*Corruptions*			*Double Refresh*
				Ranks	*Banks*	*Pins*				*Total*	$1 \to 0$	$0 \to 1$	
$\mathcal{A}_{0,1,2,3}$	16-37	2132	4	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_4$	16-51	2132	4	1	16	×8	UL	4	9-sided	7956	4008	3948	—
$\mathcal{A}_5$	18-51	2400	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_{6,7}$	18-15	2666	4	1	8	×16	UL	—	—	—	—	—	—
$\mathcal{A}_8$	17-09	2400	8	1	16	×8	UL	33	19-sided	20808	10289	10519	—
$\mathcal{A}_9$	17-31	2400	8	1	16	×8	UL	33	19-sided	24854	12580	12274	—
$\mathcal{A}_{10}$	19-02	2400	16	2	16	×8	UL	488	10-sided	11342	1809	11533	✓
$\mathcal{A}_{11}$	19-02	2400	16	2	16	×8	UL	523	10-sided	12830	1682	11148	✓
$\mathcal{A}_{12,13}$	18-50	2666	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_{14}$	19-08[†]	3200	16	2	16	×8	UL	120	14-sided	32723	16490	16233	—
$\mathcal{A}_{15}$[‡]	17-08	2132	4	1	16	×8	UL	2	9-sided	22397	12351	10046	—

TRRespass can trigger bit flips when double refresh is employed.

Evaluation – Increasing Refresh Rate

Doubling/quadrupling refresh rate is one mitigation mechanism.

Increasing refresh rate might also improve the effectiveness of TRR.



Module	Date (yy-ww)	Freq. (MHz)	Size (GB)	Organization			MAC	Found Patterns	Best Pattern	Corruptions			Double Refresh
				Ranks	Banks	Pins				Total	1 → 0	0 → 1	
$\mathcal{A}_{12,13}$	18-50	2666	8	1	16	×8	UL	—	—	—	—	—	—
$\mathcal{A}_{14}$	19-08†	3200	16	2	16	×8	UL	120	14-sided	32723	16490	16233	—
$\mathcal{A}_{15}$‡	17-08	2132	4	1	16	×8	UL	2	9-sided	22397	12351	10046	—

"This result further undermines the efficacy of double refresh as a stopgap solution against RowHammer even when in-DRAM TRR is deployed."

TRRespass can trigger bit flips when double refresh is employed.

Evaluation – Repeatability of the Bit Flips

- Repeatability is a **fundamental factor** in RowHammer exploitation
- The best pattern found by TRRespass is executed on one module per DRAM vendor

- Bit flips are repeatable
  - multiple attempts might be required
  - spurious bit flips might be generated

Exploitation with TRRespass

1. *Memory templating*:
   - find the right RowHammer access pattern
2. *Memory massaging:*
   - *map the target data onto one of the available templates*
3. *Exploitation*:
   - trigger the same RowHammer bit flips on the target data

Exploitation with TRRespass

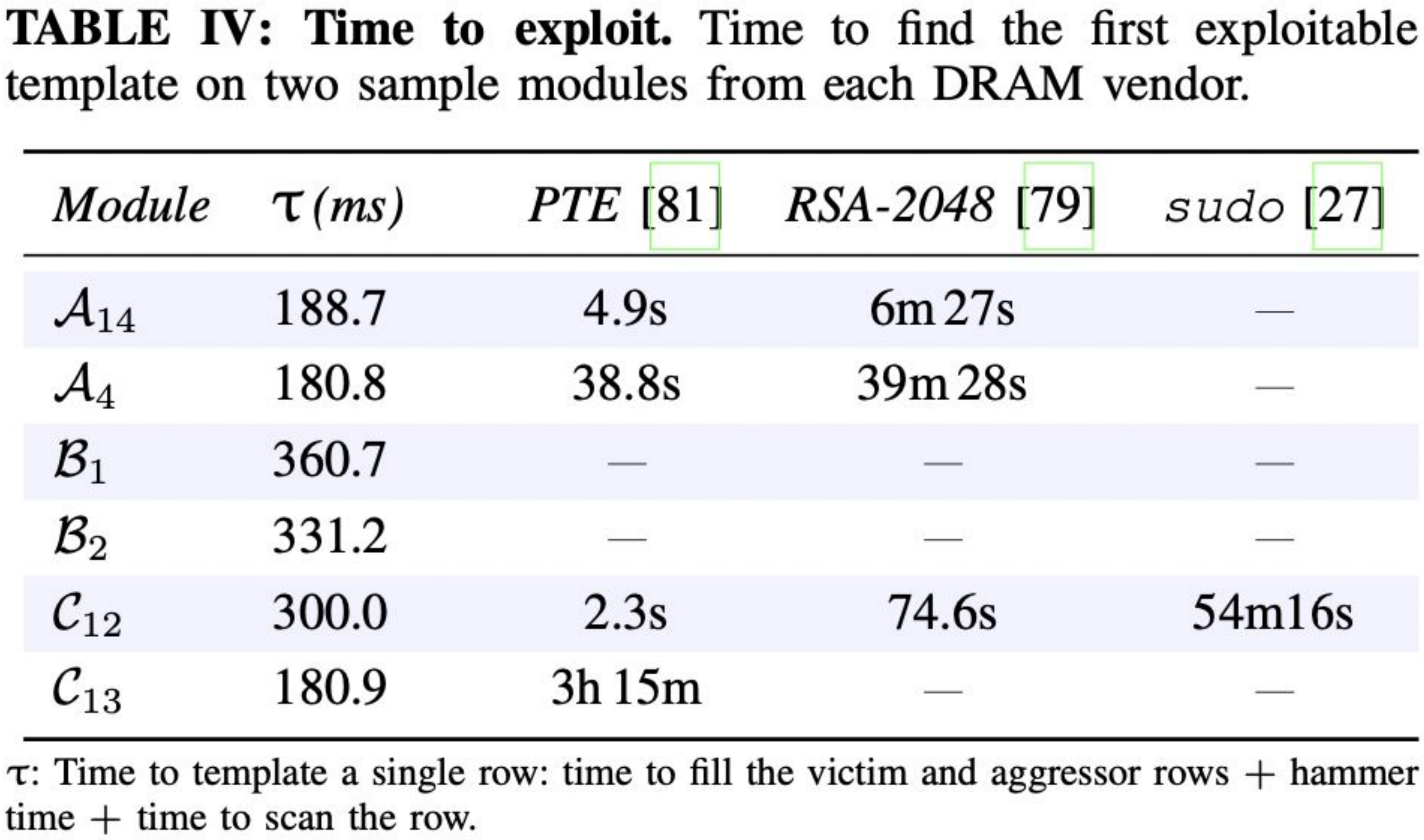


TABLE IV: Time to exploit. Time to find the first exploitable template on two sample modules from each DRAM vendor.

Module	τ *(ms)*	*PTE* [81]	*RSA-2048* [79]	`sudo` [27]
$\mathcal{A}_{14}$	188.7	4.9s	6m 27s	—
$\mathcal{A}_{4}$	180.8	38.8s	39m 28s	—
$\mathcal{B}_{1}$	360.7	—	—	—
$\mathcal{B}_{2}$	331.2	—	—	—
$\mathcal{C}_{12}$	300.0	2.3s	74.6s	54m16s
$\mathcal{C}_{13}$	180.9	3h 15m	—	—

τ: Time to template a single row: time to fill the victim and aggressor rows + hammer time + time to scan the row.

Exploitation with TRRespass

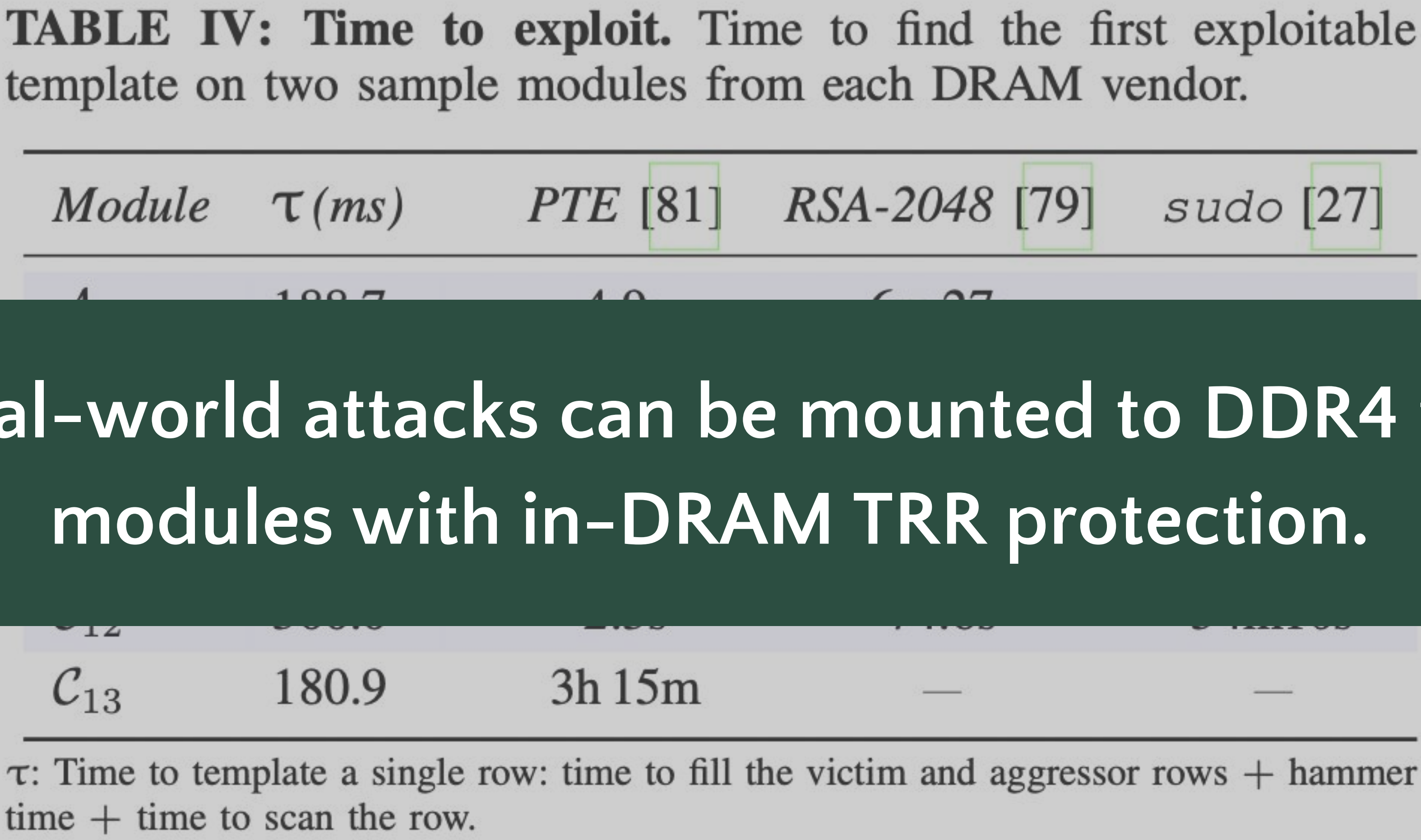


TABLE IV: Time to exploit. Time to find the first exploitable template on two sample modules from each DRAM vendor.

Module	τ *(ms)*	*PTE* [81]	*RSA-2048* [79]	*sudo* [27]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
$\mathcal{C}_{13}$	180.9	3h 15m	—	—

τ: Time to template a single row: time to fill the victim and aggressor rows + hammer time + time to scan the row.

Real-world attacks can be mounted to DDR4 for modules with in-DRAM TRR protection.

Conclusion/Takeaways

- Reverse engineers the pTRR and in-DRAM TRR mechanisms implemented in memory controllers and DRAM chips
- First work to show that DRAM modules with in-DRAM TRR are vulnerable to RowHammer
- Presents TRRespass, Black-box RowHammer test suite that generates effective access patterns to bypass in-DRAM TRR solutions
- Demonstrates that bit flips can be induced in 13/42 DRAM modules tested
- Provides hammering patterns to mount real-world attacks for many of the DDR4 DRAM modules in the market

Conclusion/Takeaways

- Reverse engineers the pTRR and in-DRAM TRR mechanisms implemented in memory controllers and DRAM chips
- First work to show that DRAM modules with in-DRAM TRR are vulnerable to RowHammer

Improves our **understanding** of TRR *significantly*

- Demonstrates that bit flips can be induced in 13/42 DRAM modules tested
- Provides hammering patterns to mount real-world attacks for many of the DDR4 DRAM modules in the market

CRITIQUE

Strengths

- Proves in-DRAM TRR is not effective against RowHammer
- Proposes many-sided RowHammer attacks that can bypass TRR mechanisms
- TRRespass can automatically induce bit flips
- Proves the exploitability of the RowHammer attacks
- Presents the *first* overview on in-DRAM TRR implementations:
  - uncovers some of their underlying mechanisms
  - demonstrates there are a variety of TRR implementations
- Demonstrates that modern memory controllers do not employ TRR

Weaknesses

- TRRespass is **inefficient**
  - might not find a hammering pattern
  - might not find the *best* hammering pattern
- Bit flips observed *only* in 13/42 modules tested
- Cannot *always* create an attack
- The sampling mechanism is not fully understood
- Can create bit flips for only *untested* DRAM modules for vendor C
- The writing is poor
- Vendor based conclusions are not insightful

Discussion Points

What other tests could be developed to exactly figure out the underlying TRR mechanism?

Reverse Engineered:

- ✓ The sampler size
- ✓ Association of TRR with regular refreshes

Still not discovered:

- ❓ How to select which victim row to refresh?
- ❓ What is the sampling mechanism?

Discussion Points

Would TRR be a viable solution if the sampler size increases significantly?

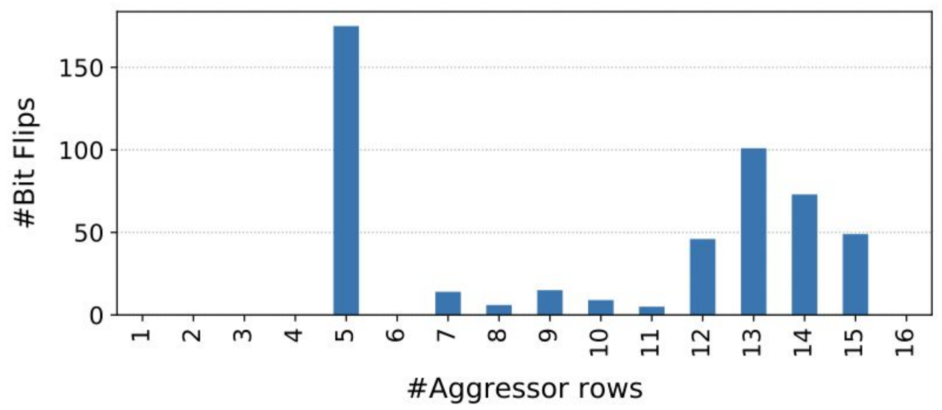


Fig. 10: Bit flips vs. number of aggressor rows. Module $\mathcal{C}_{12}$:

No bit flips for aggressor rows < sampler size

Kim et al., 2020 reports minimum hammering count (HC) as low as 4800 per aggressor row. Which one becomes more dominant?

- The max # of activations per refresh interval / min # HC
- Sampler size

Discussion Points

Assuming that we can increase the sampler size as much as required:

- Can sampler *always* sample the aggressor rows?

Even if we the sampler always samples the aggressor rows

"Observation 6. For a given DRAM manufacturer, chips of newer DRAM technology nodes can exhibit RowHammer bit flips 1) in more rows and 2) farther away from the victim row."

[Kim et al., 2020]

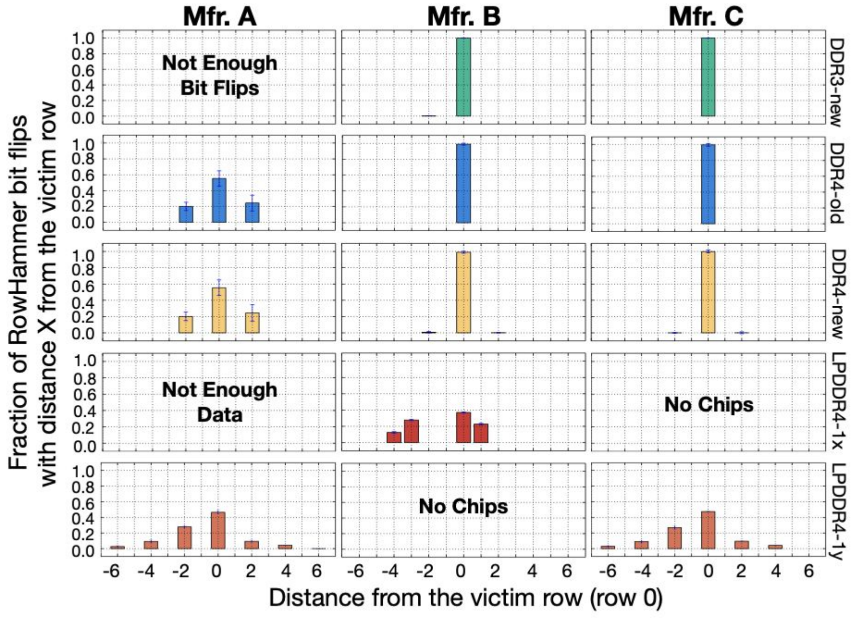


Figure 6: Distribution of RowHammer bit flips across row offsets from the victim row.

- How TRR should specify the victim rows of a given aggressor row?

Discussion Points

One key limitation of in-DRAM TRR is that it relies on the *execution of refresh command.*

- What if the mitigation mechanism was able to act independently of the refresh command?
- Or, does it have to rely on refresh at all **once** the target rows are identified?

Discussion Points

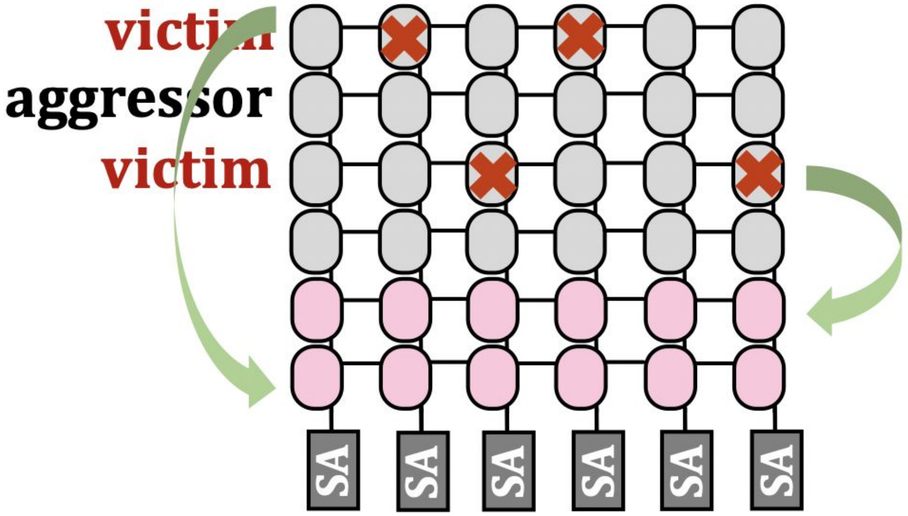
Do you think RowClone can be an *effective* solution against RowHammer?

- Remap the victim rows when the HC exceeds the **threshold**
- Remap the aggressor row when the HC exceeds the **threshold**

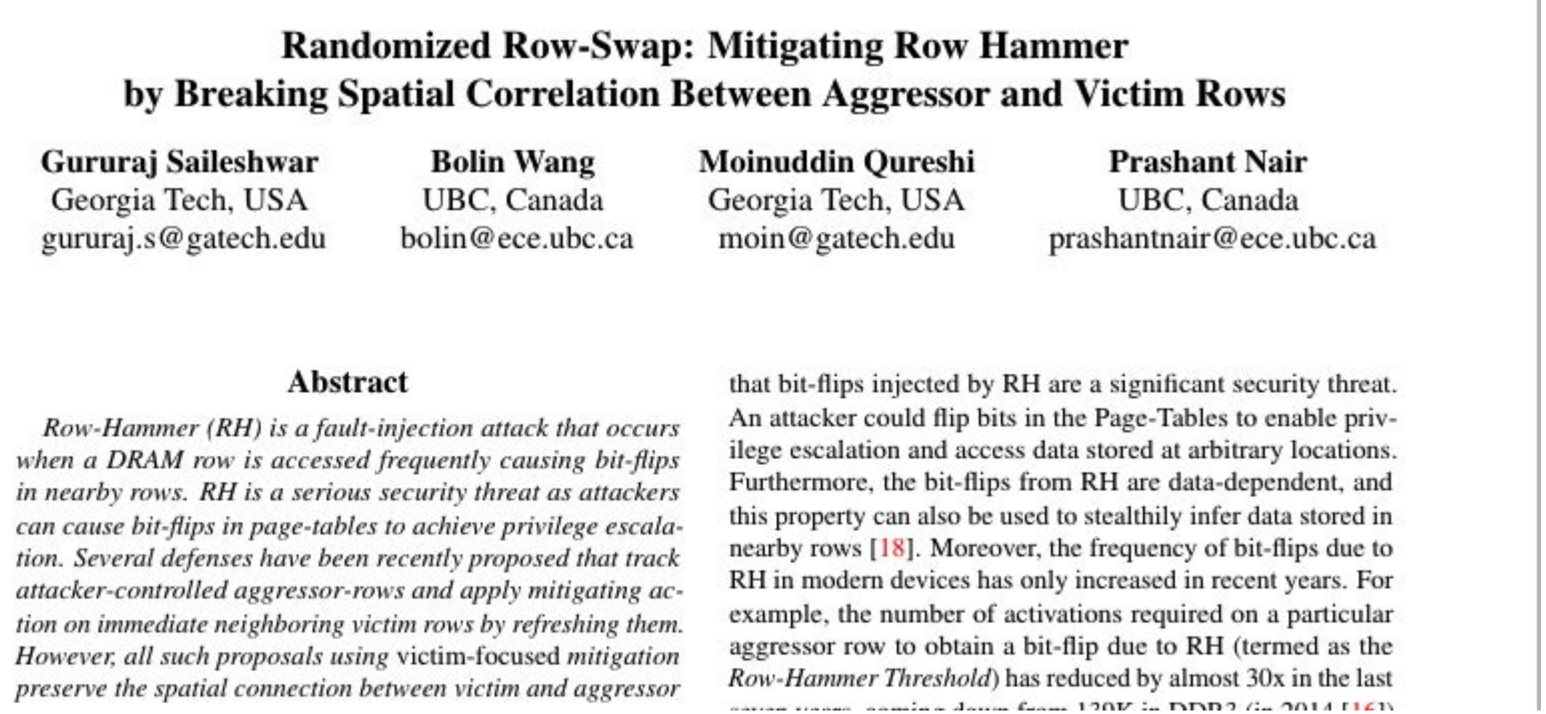
One prior approach (CROW):

- Considers remapping only the victim row
- The number of victim rows that can be remapped is limited by the number of copy rows

[Hassan et al., 2019]



Discussion Points



Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation Between Aggressor and Victim Rows

Gururaj Saileshwar
Georgia Tech, USA
gururaj.s@gatech.edu

Bolin Wang
UBC, Canada
bolin@ece.ubc.ca

Moinuddin Qureshi
Georgia Tech, USA
moin@gatech.edu

Prashant Nair
UBC, Canada
prashantnair@ece.ubc.ca

Abstract

Row-Hammer (RH) is a fault-injection attack that occurs when a DRAM row is accessed frequently causing bit-flips in nearby rows. RH is a serious security threat as attackers can cause bit-flips in page-tables to achieve privilege escalation. Several defenses have been recently proposed that track attacker-controlled aggressor-rows and apply mitigating action on immediate neighboring victim rows by refreshing them. However, all such proposals using victim-focused *mitigation preserve the spatial connection between victim and aggressor*

that bit-flips injected by RH are a significant security threat. An attacker could flip bits in the Page-Tables to enable privilege escalation and access data stored at arbitrary locations. Furthermore, the bit-flips from RH are data-dependent, and this property can also be used to stealthily infer data stored in nearby rows [18]. Moreover, the frequency of bit-flips due to RH in modern devices has only increased in recent years. For example, the number of activations required on a particular aggressor row to obtain a bit-flip due to RH (termed as the *Row-Hammer Threshold*) has reduced by almost 30x in the last

https://twitter.com/gururajs92/status/1460484960016601090?s=21

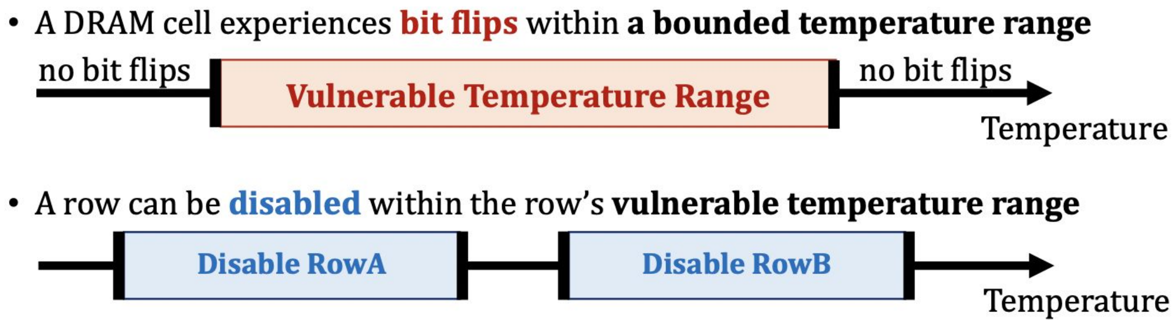
Discussion Points

TRRespass considers the cardinality and the location of aggressor rows while developing the attack.

There are **other** parameters that affect the effectiveness of RowHammer:

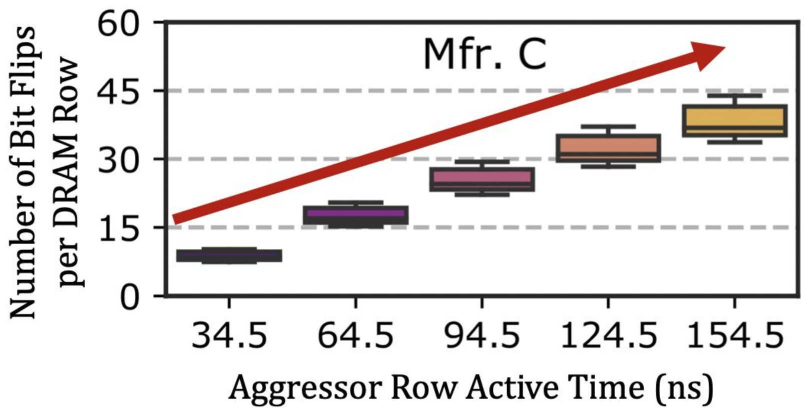
[Orosa et al., 2021]

- Temperature



• A DRAM cell experiences bit flips within a bounded temperature range
no bit flips
Vulnerable Temperature Range
no bit flips
Temperature
• A row can be disabled within the row's vulnerable temperature range
Disable RowA
Disable RowB
Temperature

- Aggressor Row Active Time



Discussion Points

Can we develop attacks that are based on these other parameters?

How would it affect the mitigation mechanisms?

The most common mitigation mechanisms are based on

- Refresh & tracking aggressor rows

Can we design mitigation mechanisms based on these other parameters?

- What other parameters we can think of?

Discussion Points

If we can solve this problem architecturally,

Is it *better* to have deterministic or probabilistic solutions?

Probabilistic

- Does not provide full protection
- In-dram TRR has some probabilistic mechanisms (e.g., address dependency) which makes it difficult to bypass

Deterministic

- Provides full protection
- Would it be easy to bypass it once you know the exact mechanism?

Can it really provide a full solution? *Blockhammer*

TRRespass: Exploiting the Many Sides of Target Row Refresh

Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P), **May 2020.**
Authors: Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos and Kaveh Razavi

Presenter: Meryem Banu Cavlak
December 9, 2021

Discussion Points

"However, all of these solutions merely treat the symptoms of a RowHammer attack (i.e., prevent RowHammer conditions) without solving the core circuit vulnerability." [Kim et al., 2020]

- Any circuit level solution ideas?
- Can we solve this problem architecturally?