U-TRR

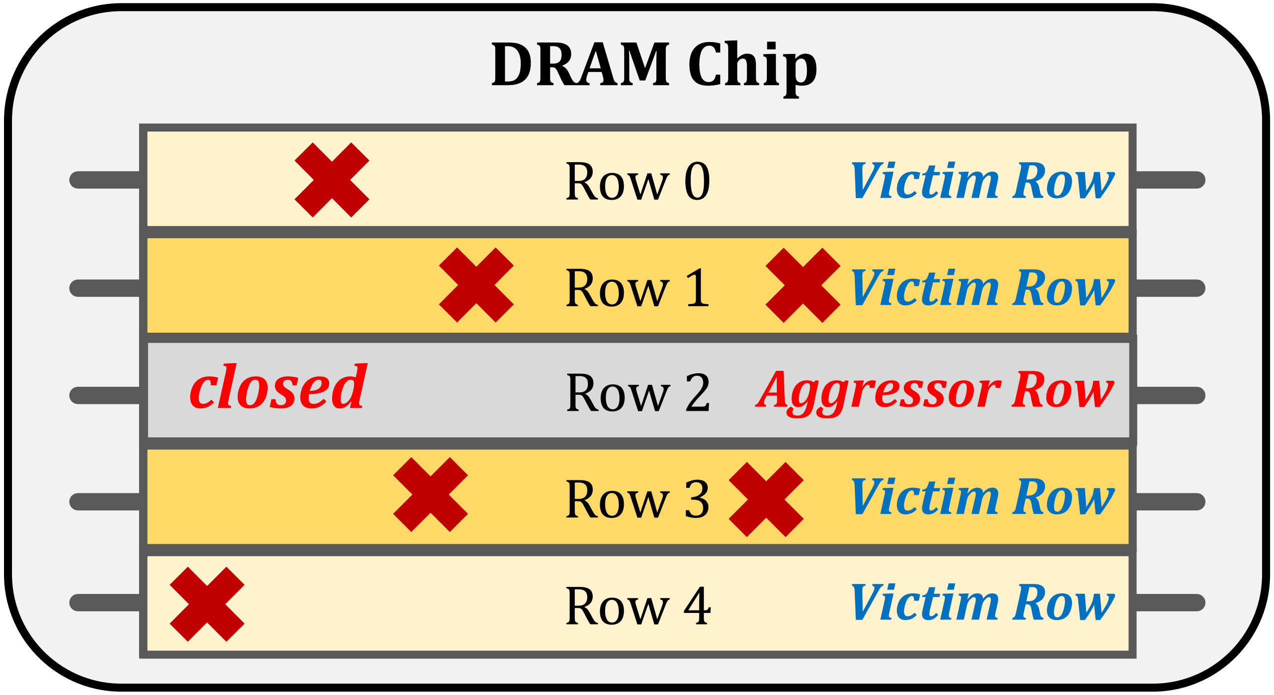
Uncovering in-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications

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

The RowHammer Vulnerability

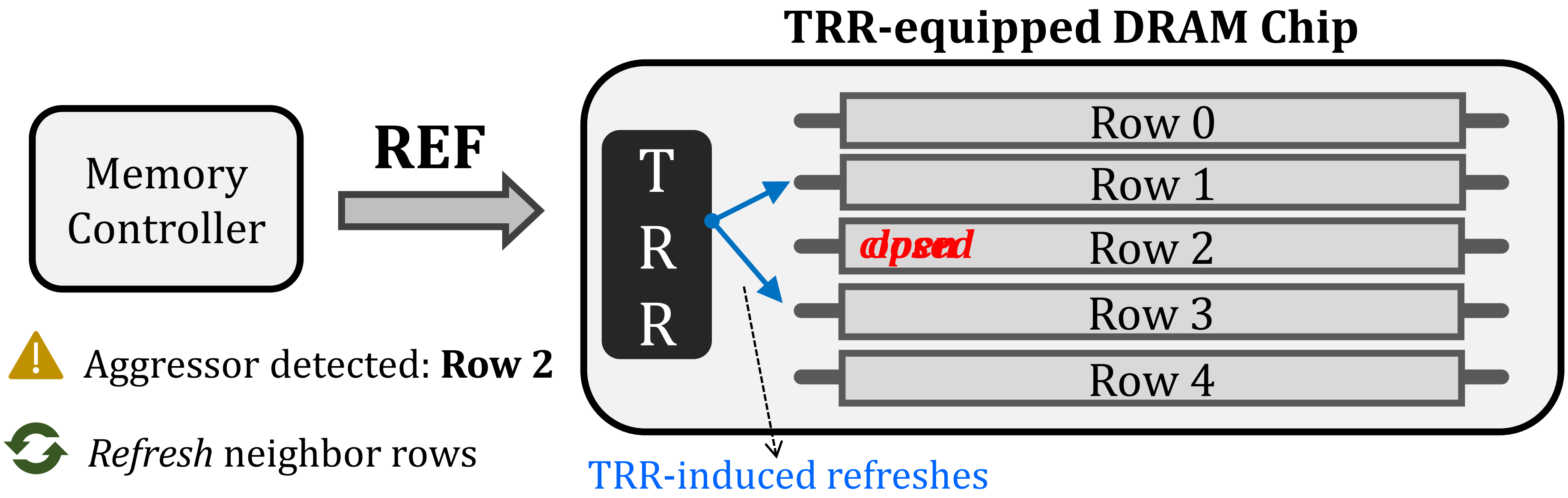


Repeatedly **opening** (activating) and **closing** (precharging) a DRAM row causes **RowHammer bit flips** in nearby cells

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



⚠ Aggressor detected: **Row 2**

Refresh neighbor rows

SAFARI

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

❶ **understand** how they operate

❷ **assess** their security

❸ **secure** DRAM completely against RowHammer

Overview of U-TRR

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

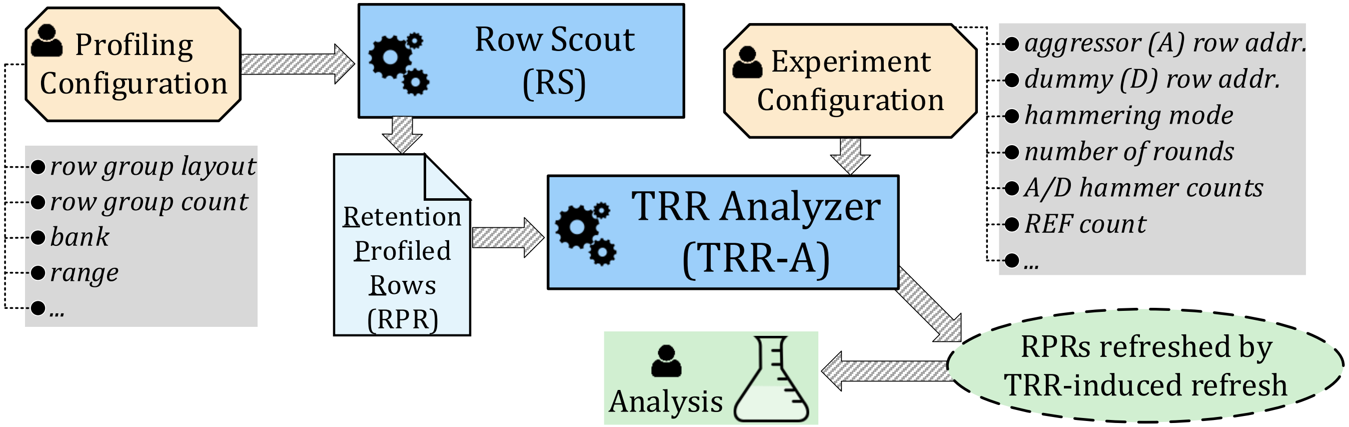
Key idea: Use data retention failures as a side channel to detect when a row is refreshed by TRR

High-Level U-TRR Operation

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

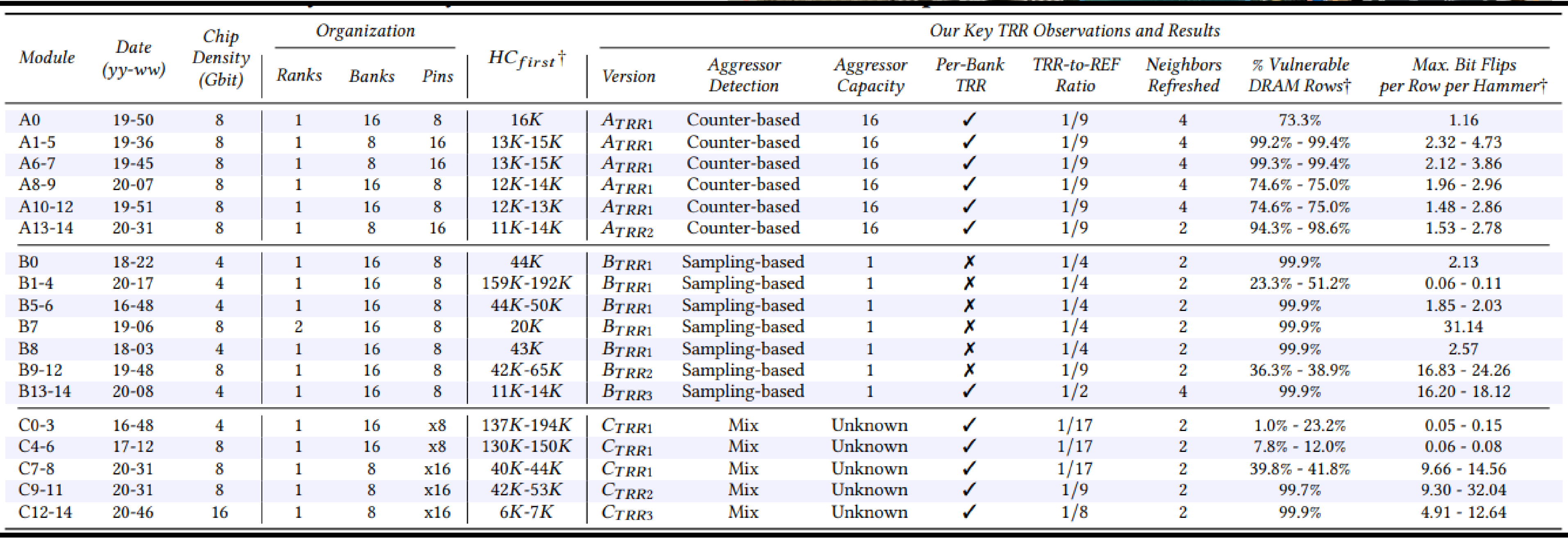
Row Scout: finds a **set of DRAM rows** that meet certain requirements as needed by TRR-A and **identifies the data retention times** of these rows

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



DRAM Testing Infrastructure

We implement U-TRR using FPGA-based *SoftMC* [Hassan+, HPCA'17] *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

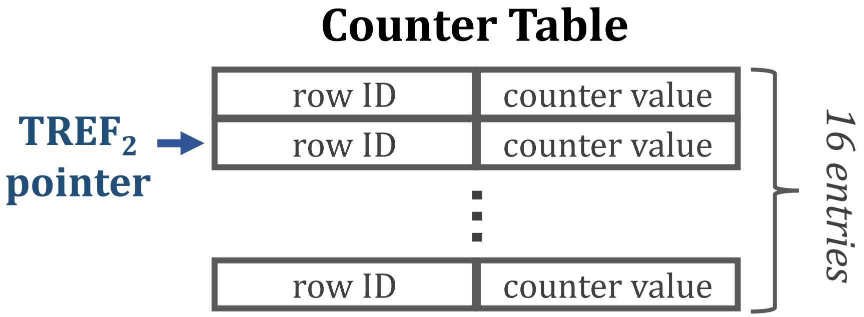
Key Observations: Vendor A

Observation 1: TRR tracks potentially aggressor rows using a Counter Table

Observation 2: The Counter Table has 16 entries

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



Counter Table

TREF$_2$ pointer

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

16 entries

Key Observations: Vendor B

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**

Key Observations: Vendor C

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

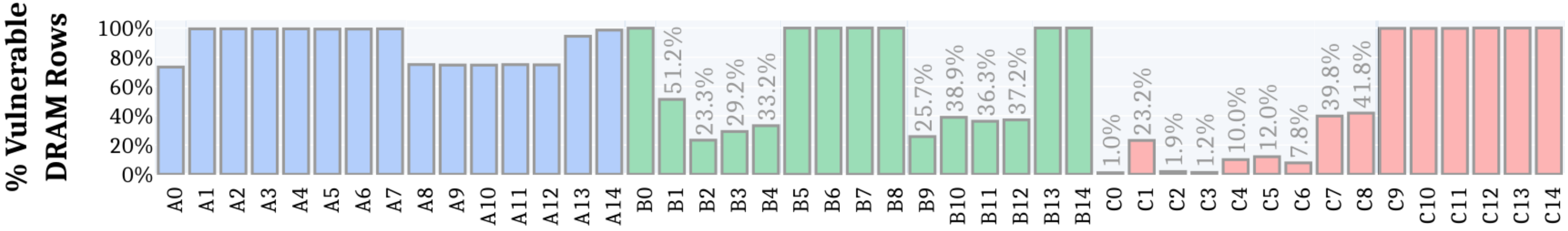
TREF: Detects an aggressor row only among the first 2K ACT commands while favoring the earlier activations more

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

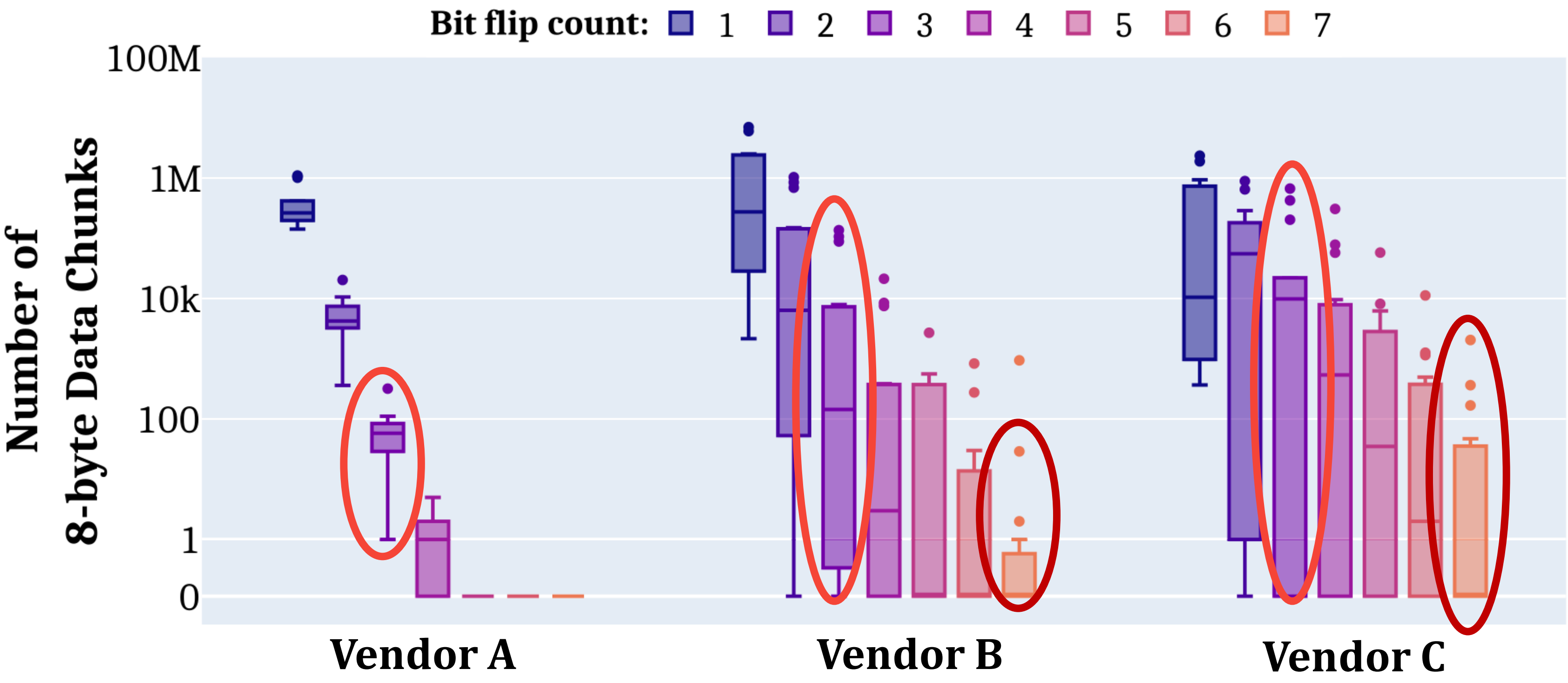
Effect on Individual Rows



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

Bypassing ECC with New RowHammer Patterns

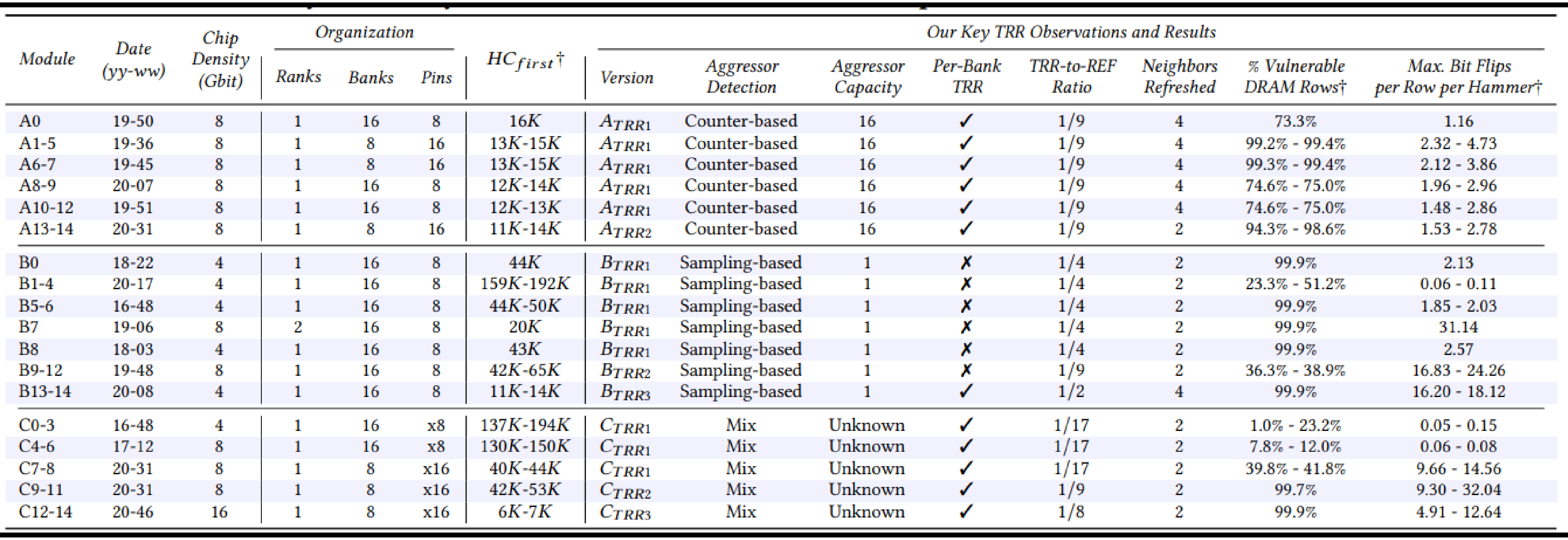


Modules from all three vendors have many **8-byte data chunks** with 3 and more (up to 7) RowHammer bit flips

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

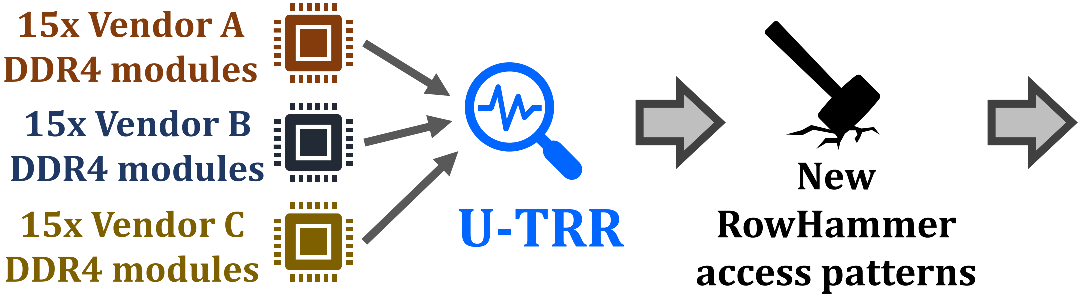
Summary

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



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 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

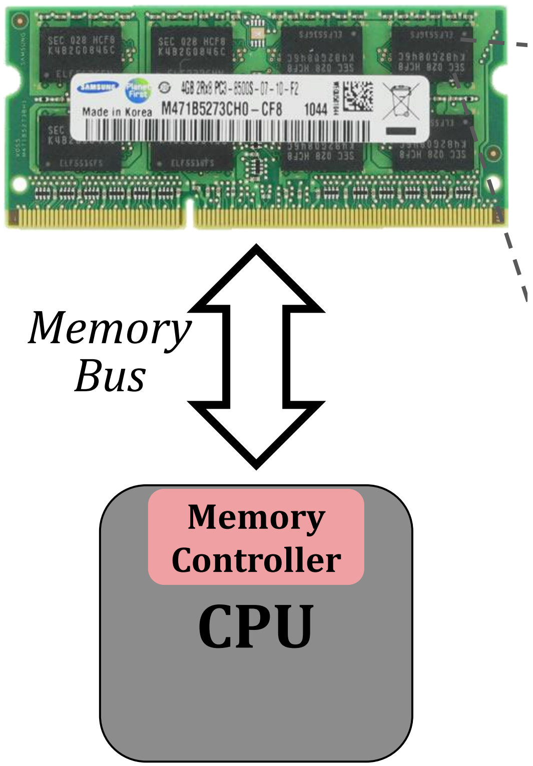
TOBB ETÜ
University of Economics & Technology

Qualcomm

U-TRR

BACKUP SLIDES

DRAM Organization

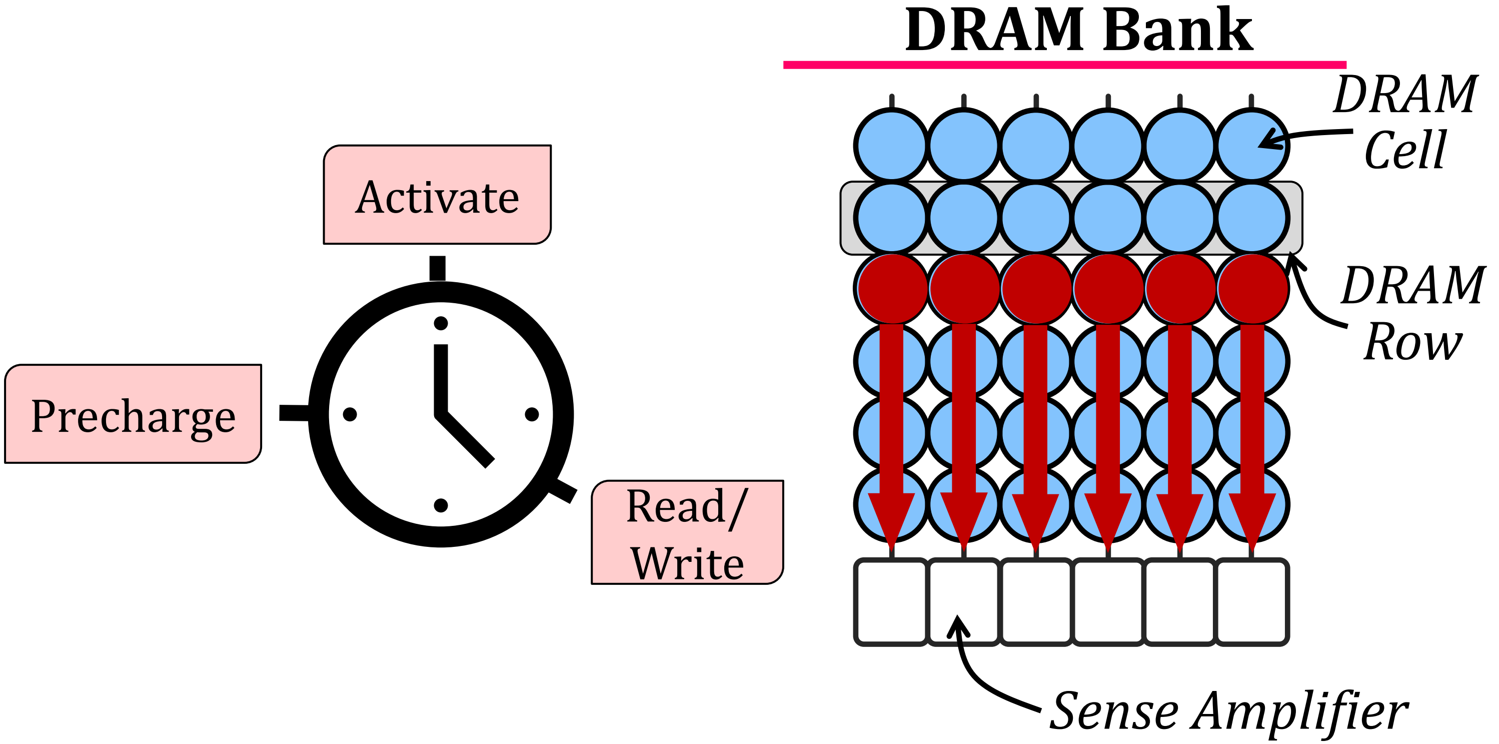


Memory Bus

Memory Controller

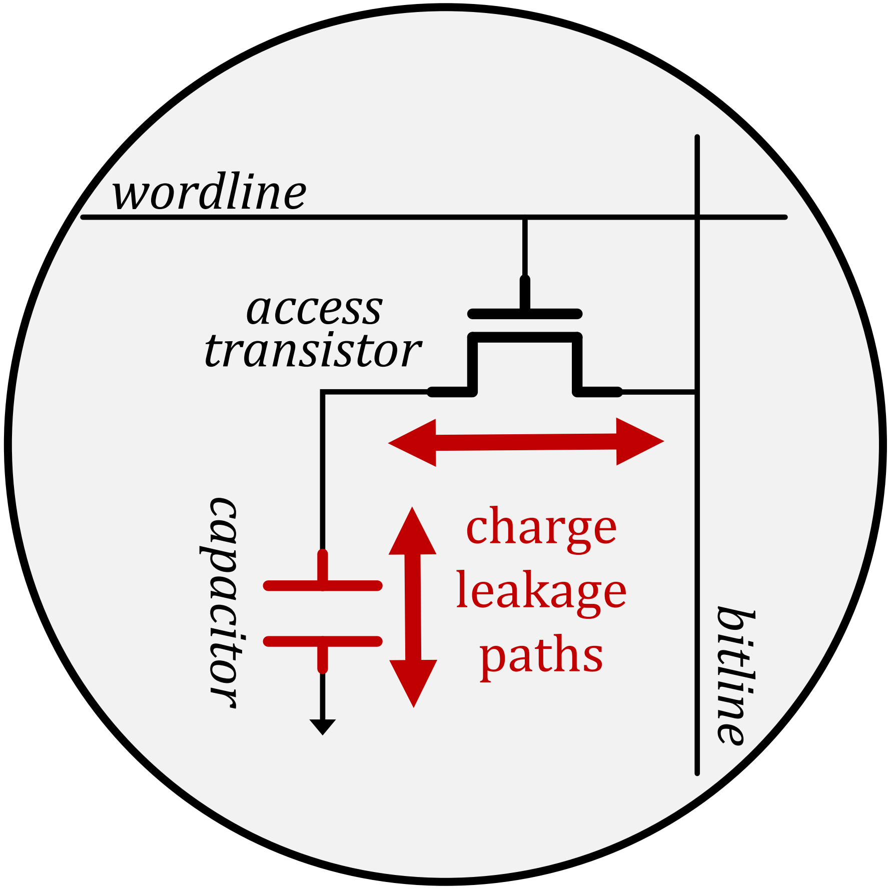
CPU

Accessing DRAM



DRAM Cell Leakage

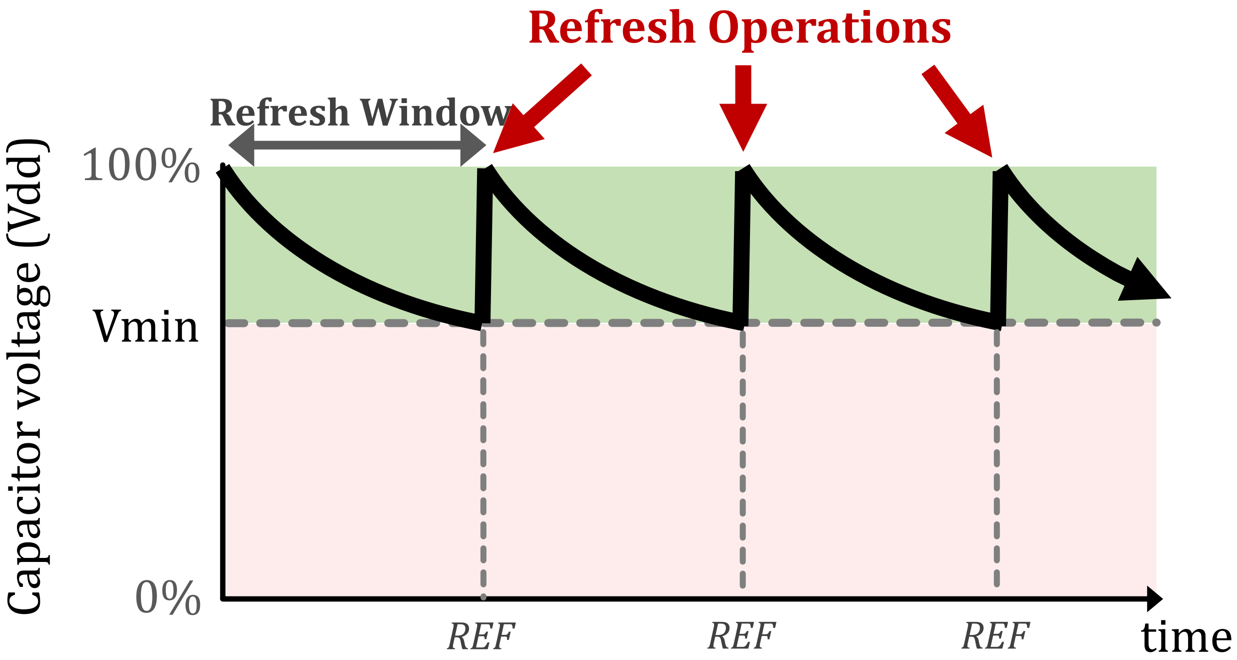
Each cell encodes information in **leaky** capacitors



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

[Patel+, ISCA'17]

Row Scout (RS)

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

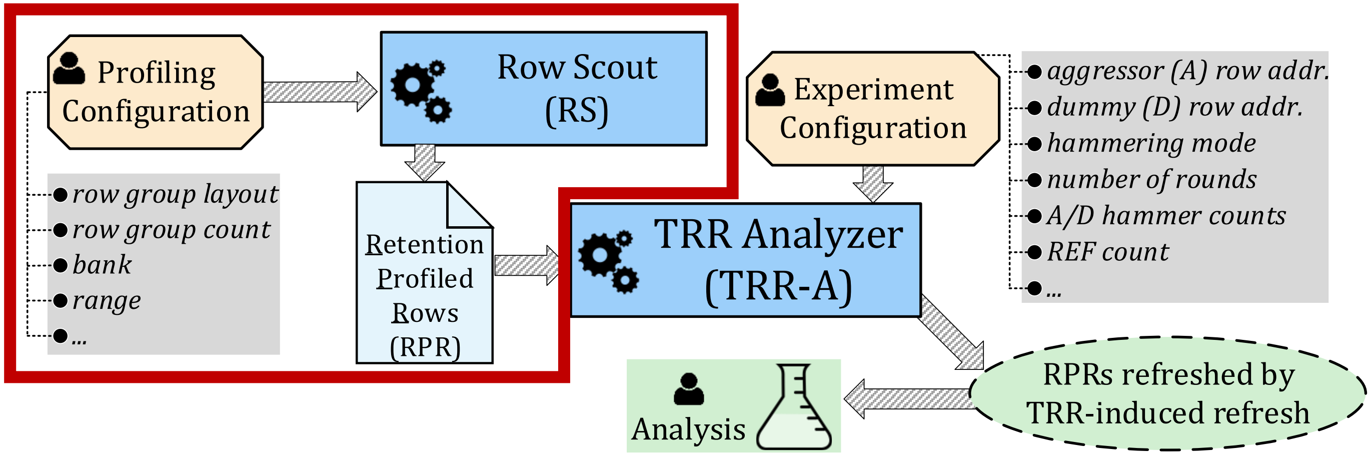
Row Scout **must** find:

✓ 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

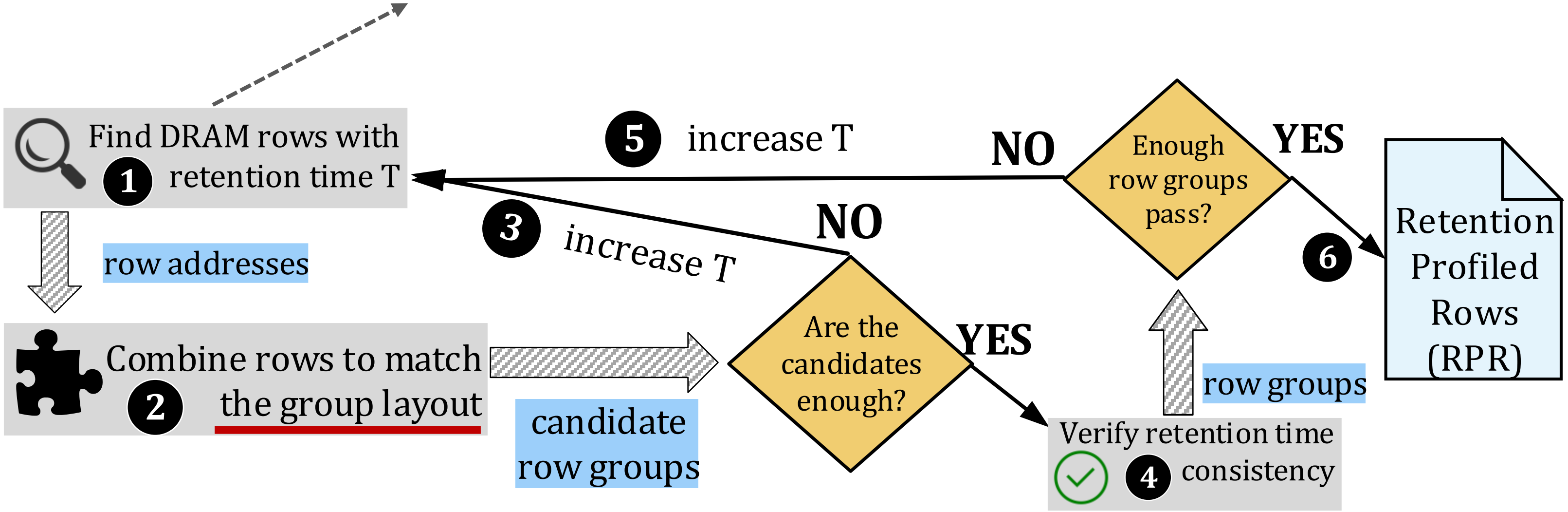


* The retention time of a DRAM row may change over time due to Variable Retention Time (VRT) effects

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)

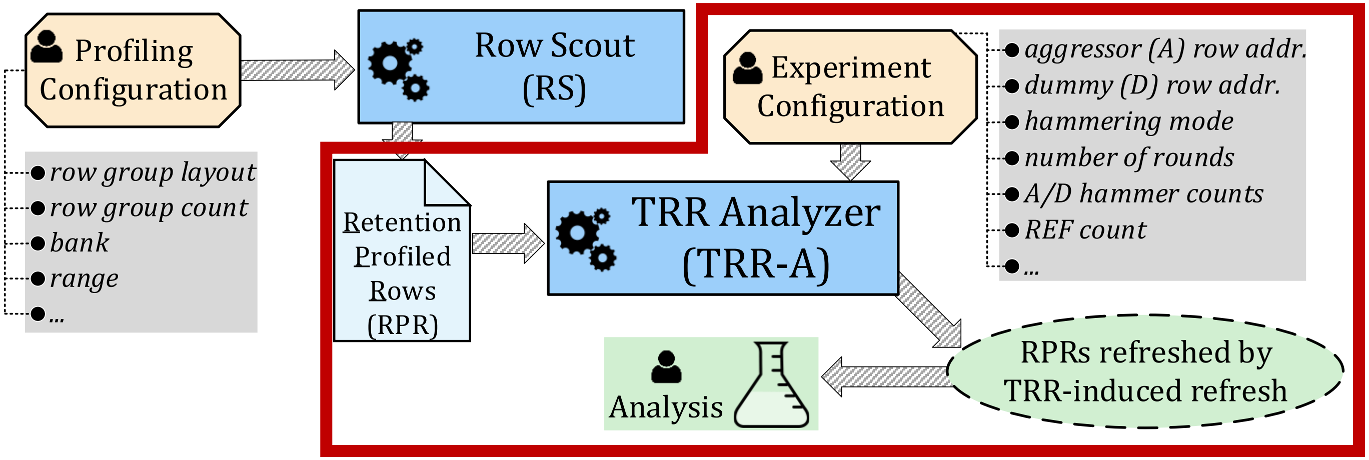
Row Group: **V** □ **V** □ **V**

TRR Analyzer (TRR-A)

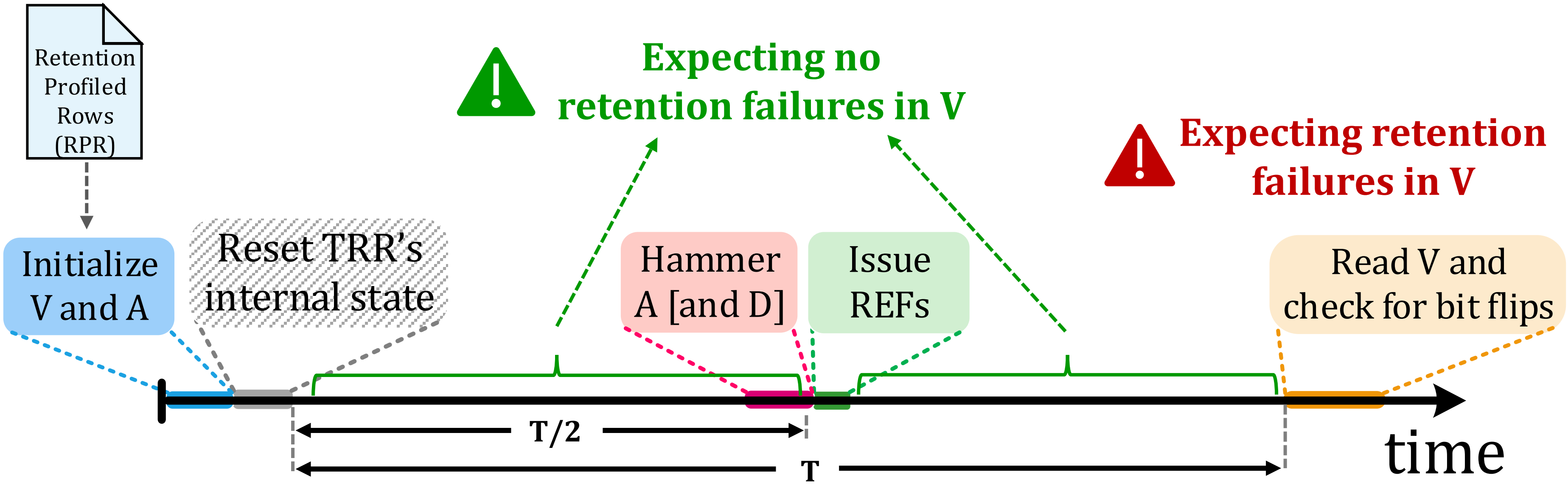
Goal: Use RS-provided rows to determine when TRR refreshes a victim row

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

Hammer A [and D]

Issue REFs

Read V and check for bit flips

Expecting no retention failures in V

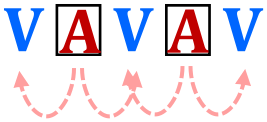
Expecting retention failures in V

T/2

T

time

Row Group:



V A V A V

V: victim (RS-provided) rows

A: aggressor rows

D: dummy rows

Experiment Configuration

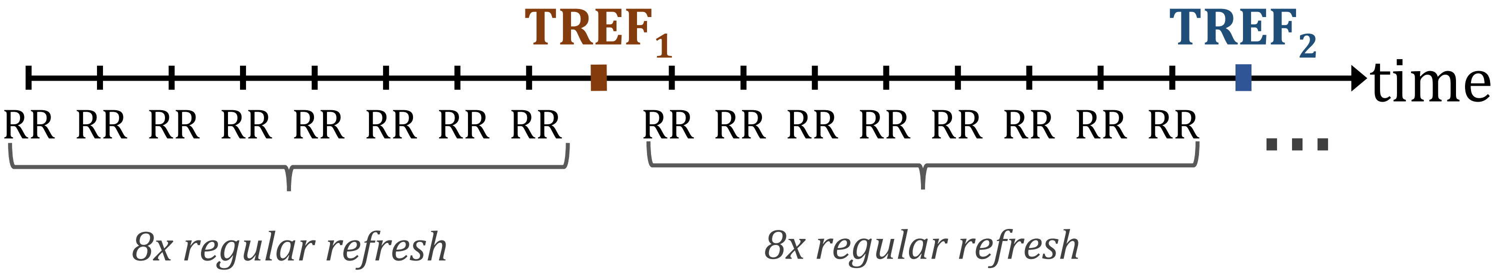
- *aggressor (A) row addr.*
- *dummy (D) row count*
- *hammering mode*
- *number of rounds*
- *A/D hammer counts*
- *REF count*

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

Key Observations: Vendor A

Refresh Types:

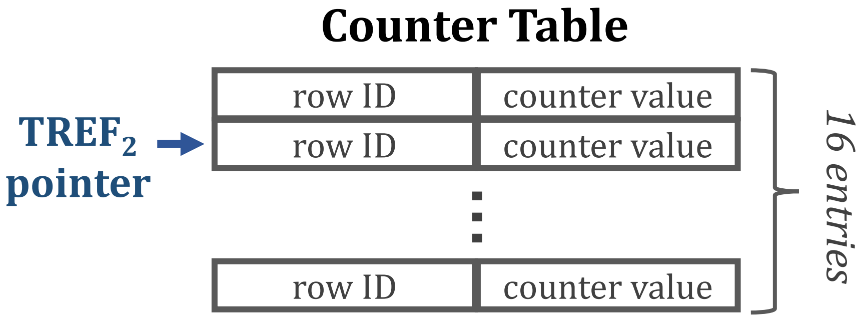
- Regular Refresh (RR)
- TRR-capable Refresh (**$\mathbf{TREF_1}$** and **$\mathbf{TREF_2}$**)



Observation: TRR tracks potentially aggressor rows using a Counter Table

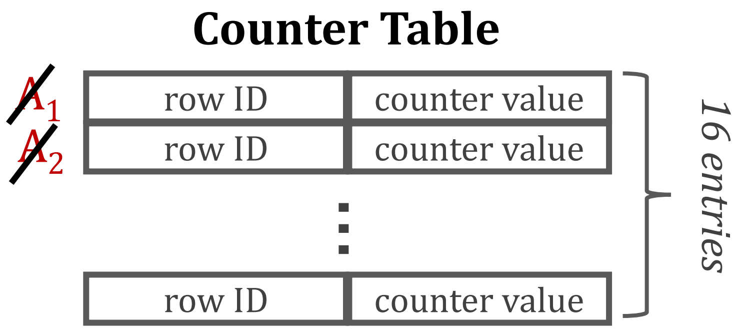
$\mathbf{TREF_1}$: Refreshes the victims of **row ID** with the **largest counter value**

$\mathbf{TREF_2}$: Refreshes the victims of **row ID** that $TREF_2$ pointer refers to



Circumventing Vendor A's TRR

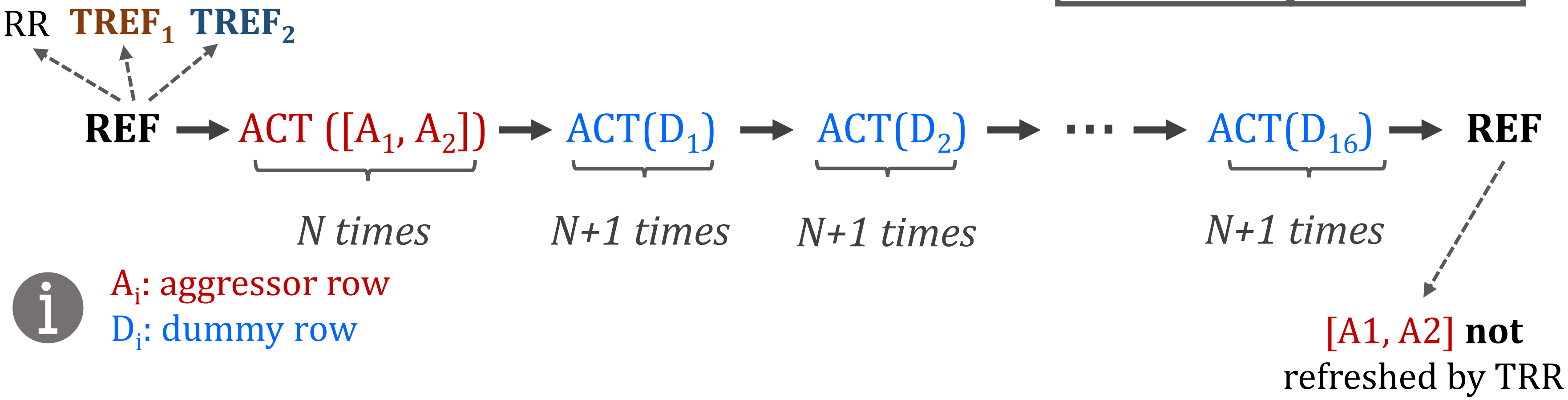
Approach: **Ensure** an aggressor row is **discarded** from the *Counter Table* **prior** to a REF command



Counter Table

~~A_1~~	row ID	counter value
~~A_2~~	row ID	counter value
	⋮	
	row ID	counter value

16 entries



REF → ACT ([A_1, A_2]) → ACT(D_1) → ACT(D_2) → ··· → ACT(D_{16}) → REF

N times · *N+1 times* · *N+1 times* · *N+1 times*

A_i: aggressor row
D_i: dummy row

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

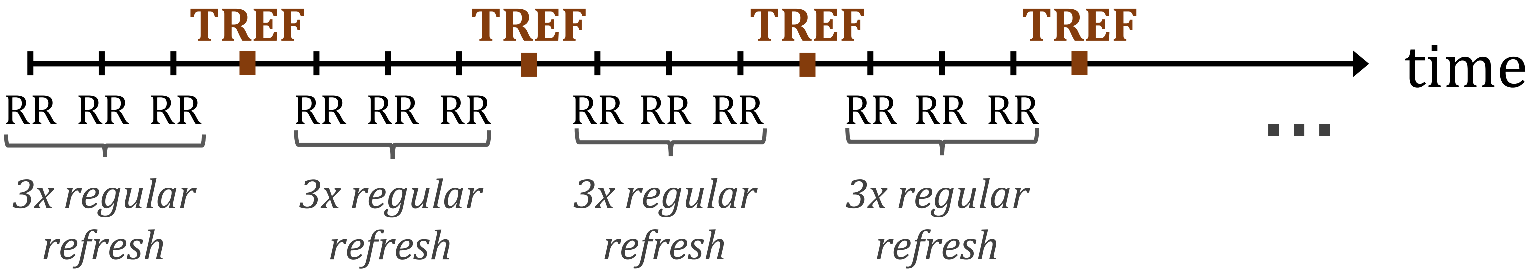
This RowHammer access pattern requires **synchronizing** accesses with REF commands

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**

SAFARI

Circumventing Vendor B's TRR

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

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

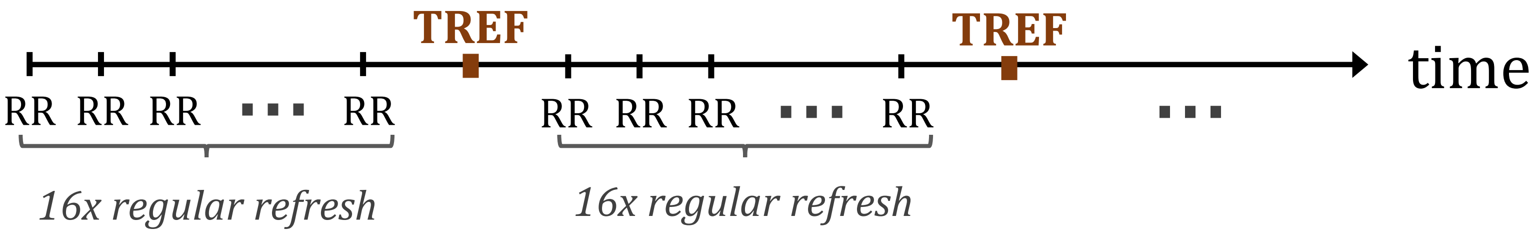
[A1, A2] **not** refreshed by TRR

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

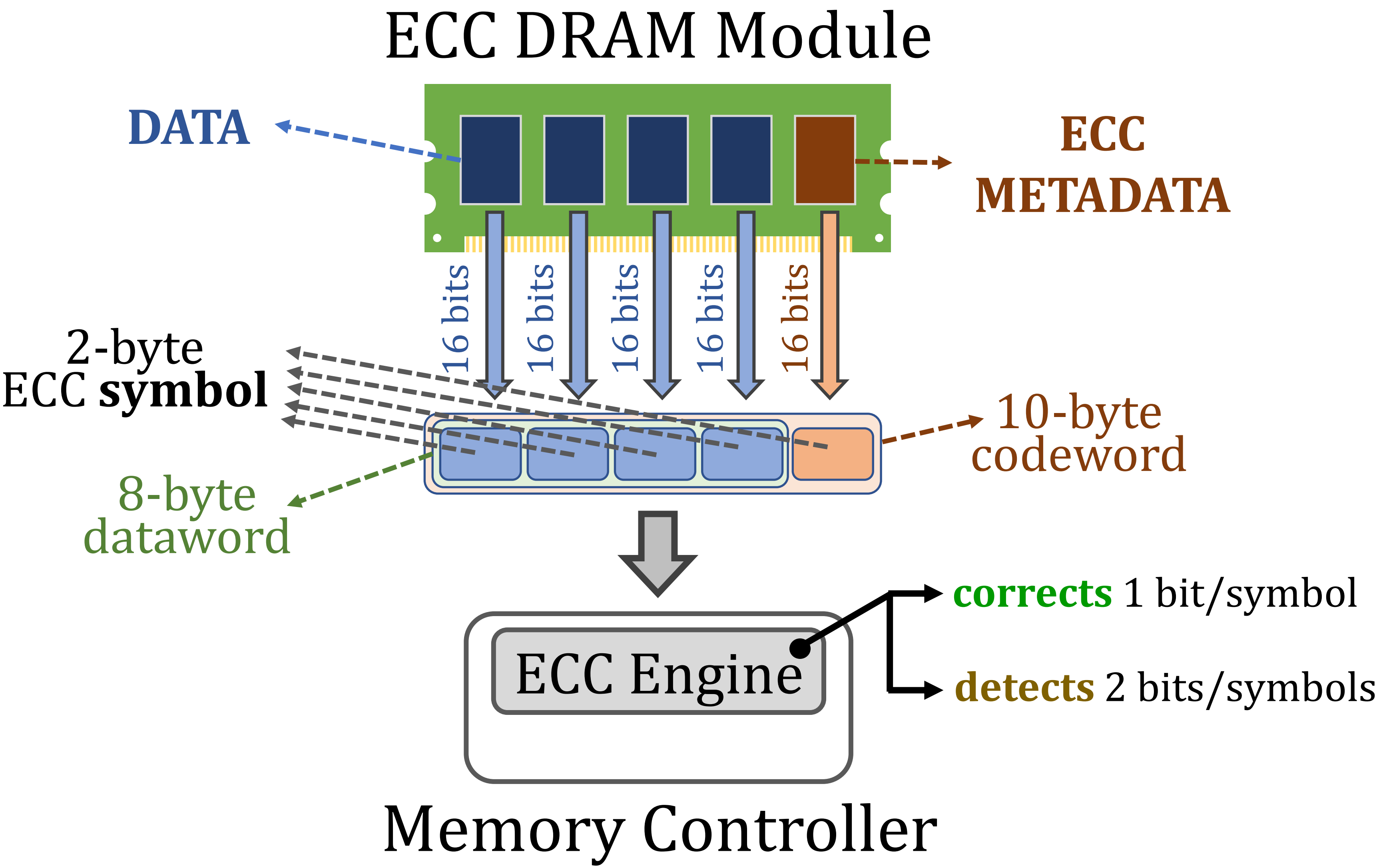
Approach: Hammer dummy rows before aggressor rows to **maximize the probability** of TRR **detecting** a dummy row

$$\textbf{TREF} \rightarrow \underbrace{\mathrm{ACT}(\mathrm{D}_1)}_{\textit{N times}} \rightarrow \underbrace{\mathrm{ACT}([\mathrm{A}_1, \mathrm{A}_2])}_{\textit{M times}} \rightarrow \textbf{TREF}$$

[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

Can ECC Protect Against Our Access Patterns?



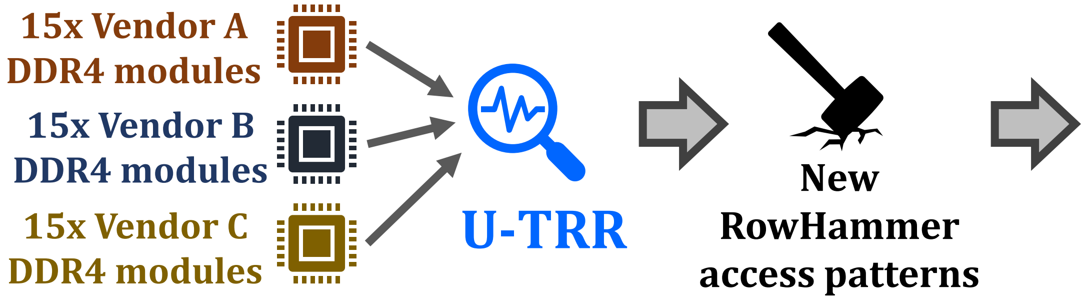
Conclusion

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**

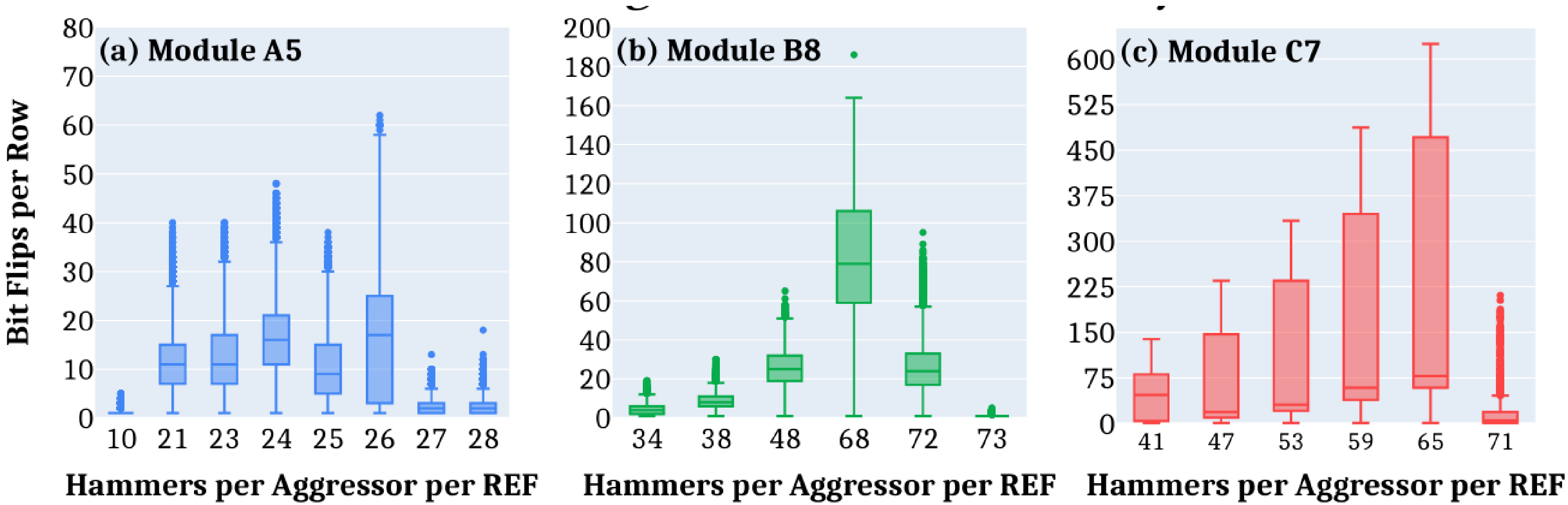
99.9% of rows in a DRAM bank experience **at least one RowHammer bit flip**

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

TRR does not provide security against RowHammer

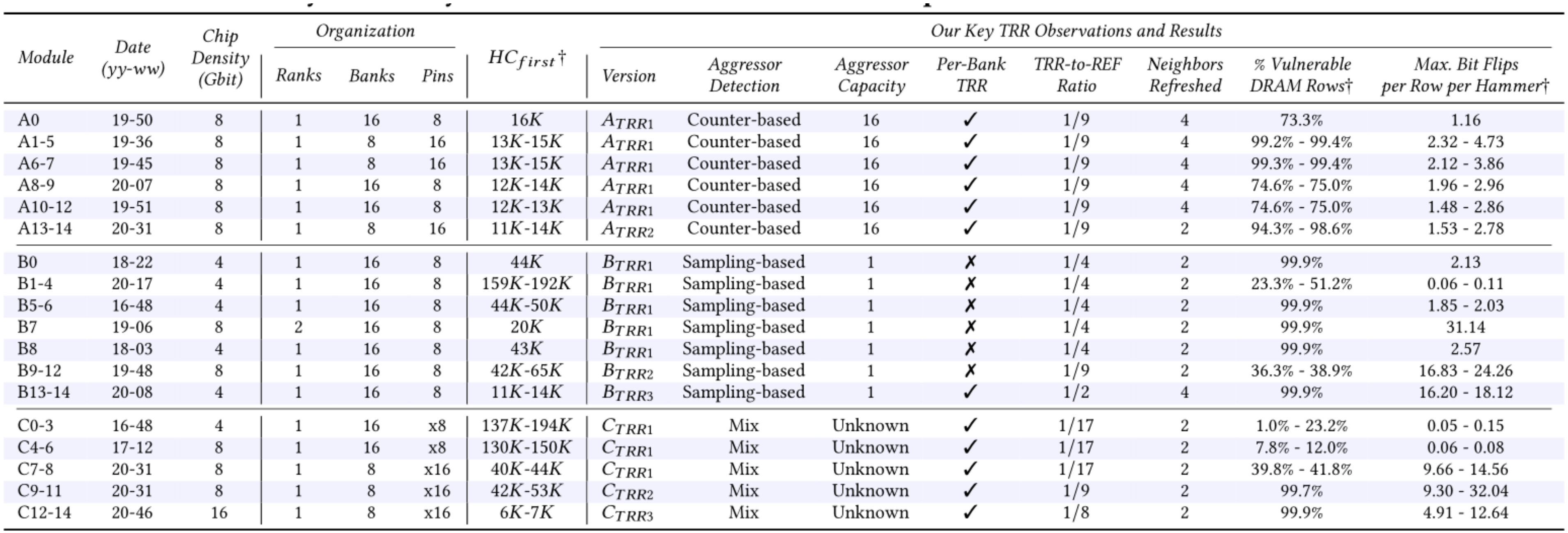
U-TRR can facilitate the development of **new RowHammer attacks** and **more secure RowHammer protection** mechanisms

Hammer Count Sweep



SAFARI

Summary of 45 DDR4 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