DejaVu

Why You Should Write to Your DRAM Rows Twice, Carefully

Haocong Luo

İsmail Emir Yüksel Ataberk Olgun Nisa Bostanci Orhun Ecemiş

Abdullah Giray Yaglikci Onur Mutlu

SAFARI ETH Zürich

TOBB ETÜ
University of Economics & Technology

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

High-Level Summary

- We demonstrate and analyze **DejaVu**, a new phenomenon in DRAM where **data previously written** to a DRAM row affects

> **Vulnerability to read disturbance**

> **Vulnerability to retention failure**

> **Reliability of Processing-Using-DRAM Operations**

- Tested **112 DDR4 chips** from all 3 major manufacturers
- Analyze **two hypotheses** on root causes
- **Implications** on DRAM testing

Outline

- DRAM Background
- DejaVu
  - Vulnerability to Read Disturbance
  - Vulnerability to Retention Failure
  - Reliability of Processing-Using-DRAM Operations
- Hypotheses on Root Causes
- Implications
- Conclusion

Outline

DRAM Background

DejaVu

- Vulnerability to Read Disturbance
- Vulnerability to Retention Failure
- Reliability of Processing-Using-DRAM Operations

Hypotheses on Root Causes

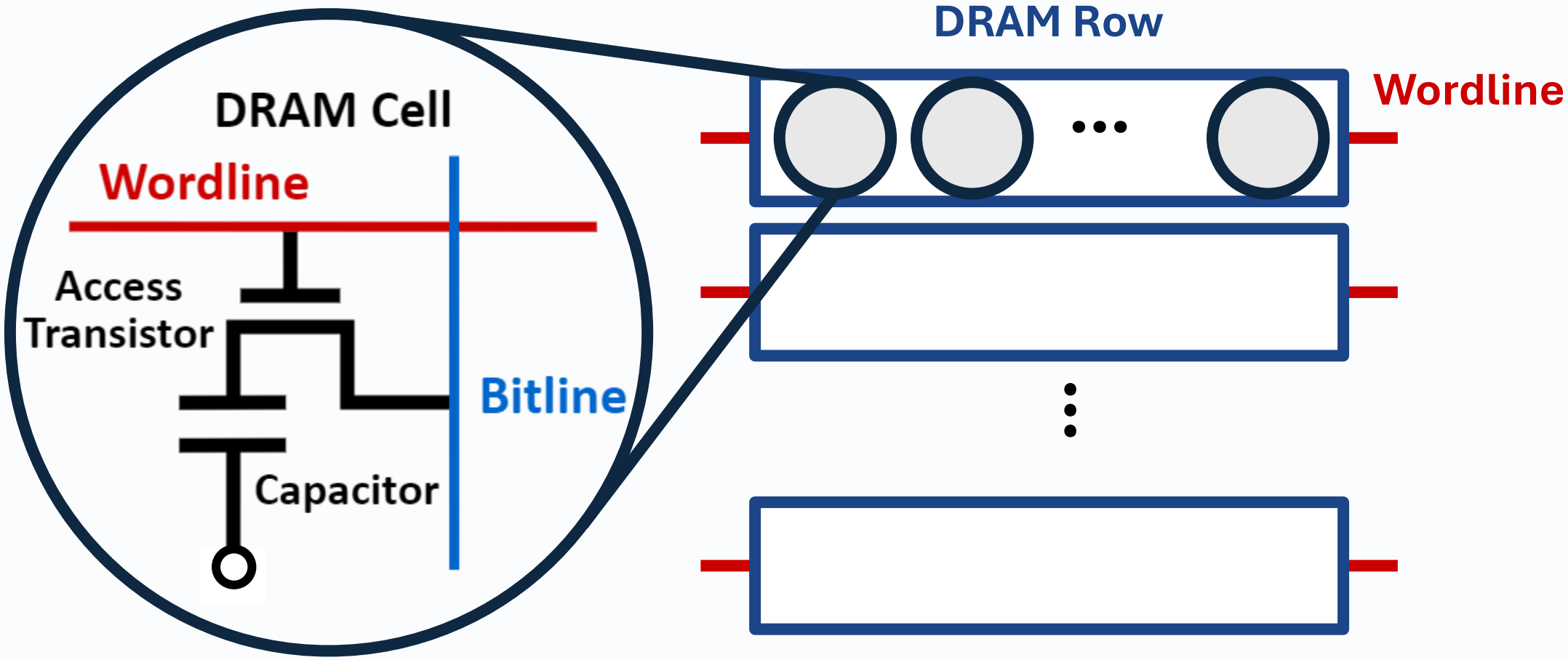
Implications

Conclusion

DRAM Organization

- A **DRAM cell** stores 1 bit of information in a capacitor
- DRAM cells are organized into **DRAM rows**

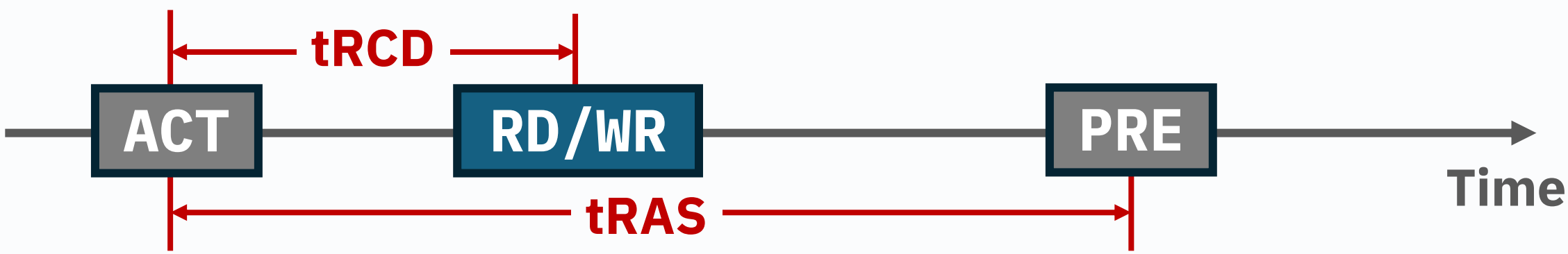
High-Level Illustration of DRAM Cell and DRAM Row



Key DRAM Operation

- DRAM cells are accessed at **row granularity**
- To read/write data in DRAM, the memory controller
  - Sends an **ACT (activate) command** to open the DRAM row
  - Sends **RD/WR commands** to read/write data
  - Sends a **PRE (precharge) command** that closes the row
- Commands need to respect **timing constraints**

Timeline of DRAM Commands



DRAM Robustness Issues (I) – Read Disturbance

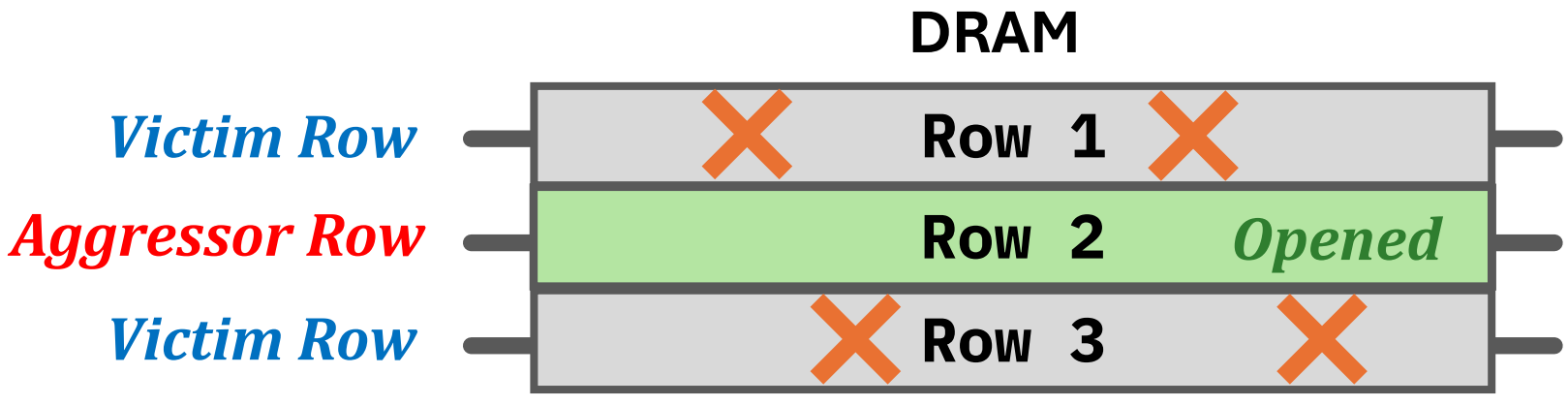
- Modern DRAM is vulnerable to **read disturbance**

RowHammer

Repeatedly **opening (activating)** and **closing** a DRAM row **many times** causes **RowHammer bitflips** in adjacent rows

RowPress

Keeping a DRAM row **open for a long period of time** causes **RowPress bitflips** in adjacent rows



DRAM Robustness Issues (I) – Read Disturbance

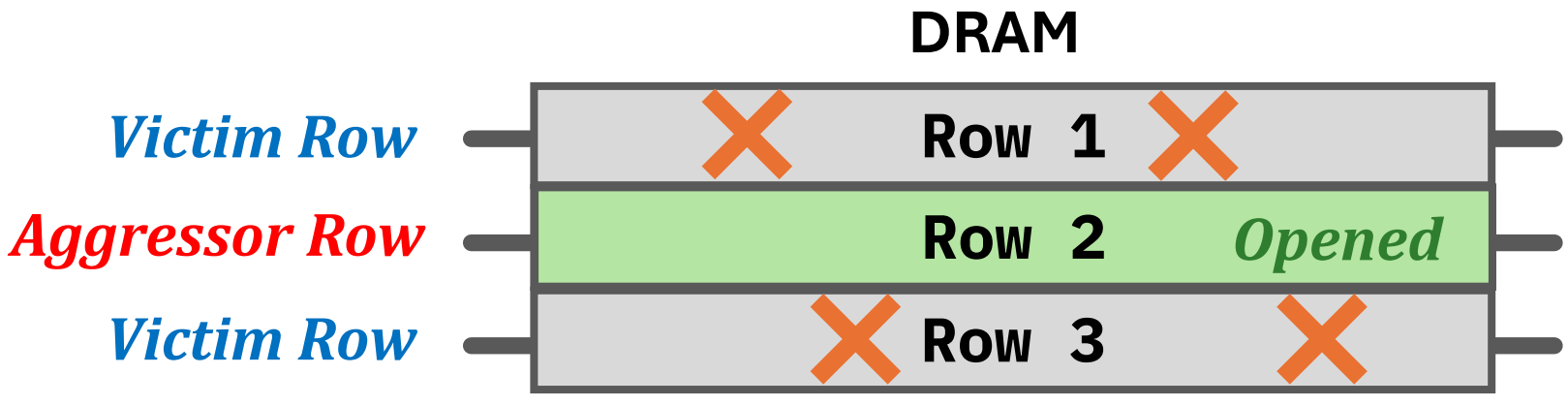
- Modern DRAM is vulnerable to **read disturbance**

RowHammer

Repeatedly **opening (activating)** and **closing** a DRAM row **many times** causes **RowHammer bitflips** in adjacent rows

RowPress

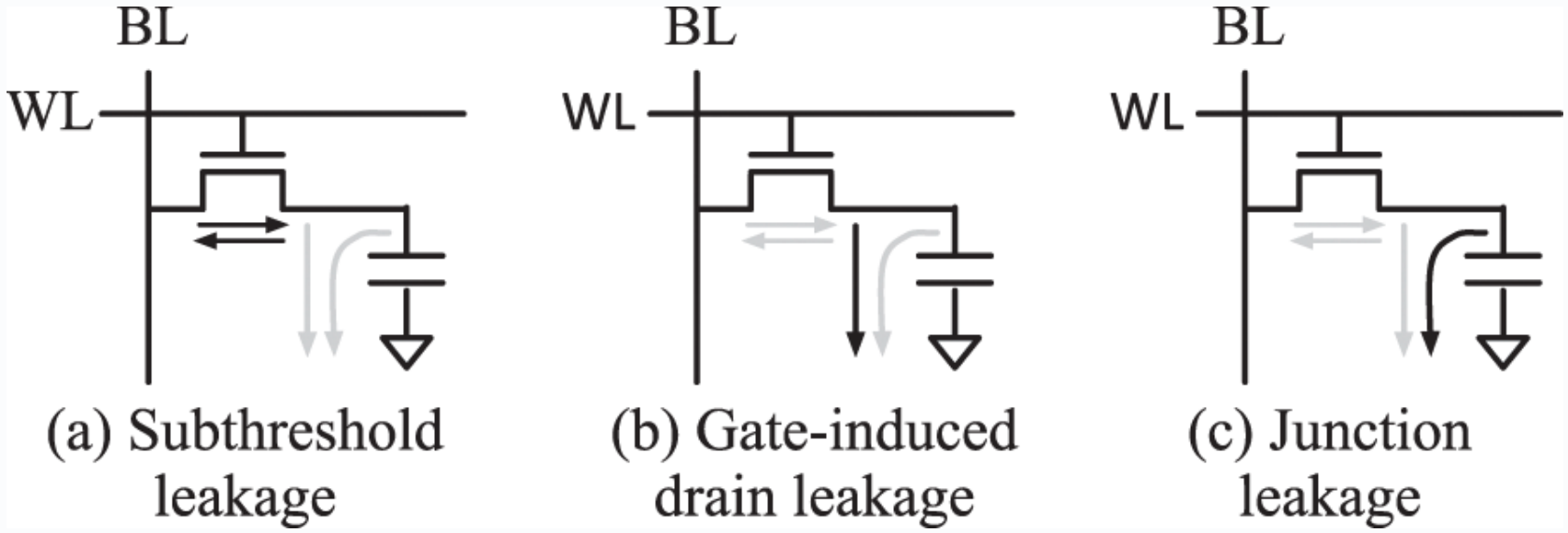
Keeping a DRAM row **open for a long period of time** causes **RowPress bitflips** in adjacent rows



DRAM Robustness Issues (II) – Retention Failure

- The DRAM cell capacitor that stores data is **leaky**

Illustration of DRAM Cell Leakage Paths [Wang+, TC'19]

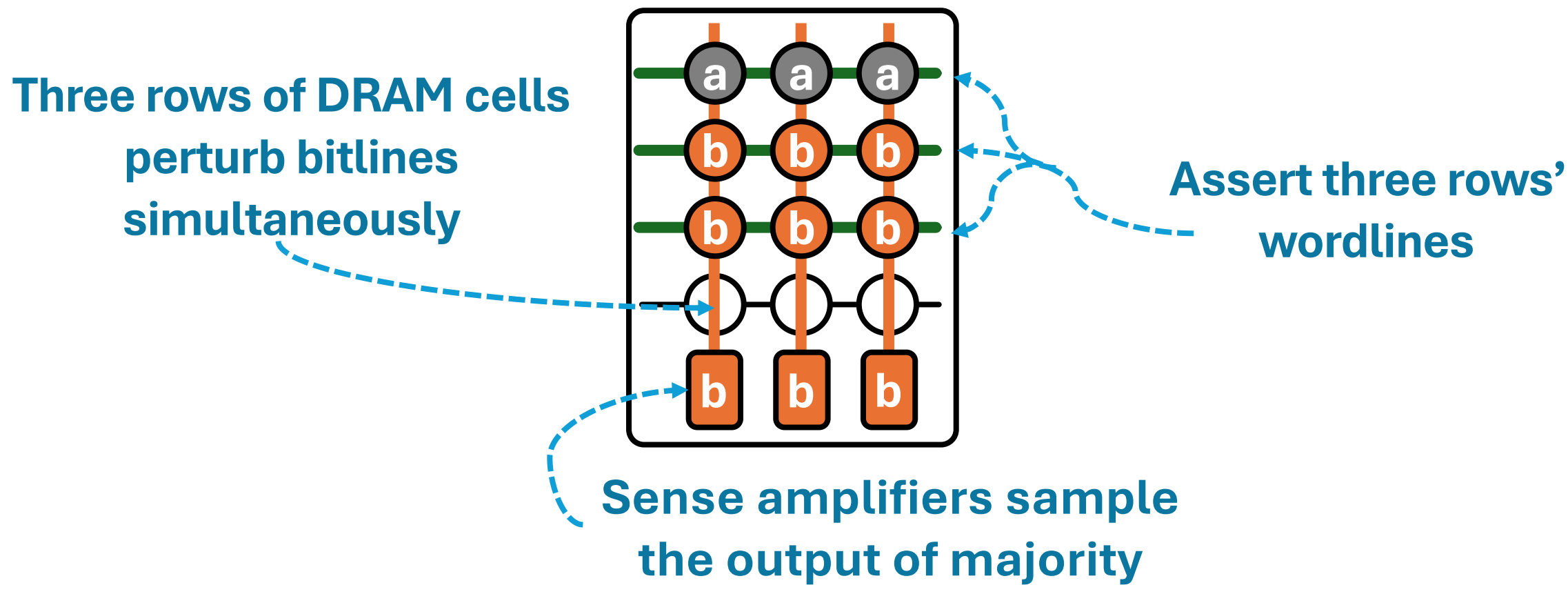


(a) Subthreshold leakage (b) Gate-induced drain leakage (c) Junction leakage

- DRAM needs frequent and periodic refresh to **prevent data retention failure bitflips**
  - One of the main DRAM scaling challenges [Liu+, ISCA'13]

Processing-Using-DRAM

- The **analog operational principles** of DRAM can be used to **perform computation**
- **Simultaneously activating X rows** that share bitlines enables a **majority-of-X operation (MAJ-X)**



[Seshadri+ MICRO'17, Gao+ MICRO'19, Yuksel+ DSN'24]

Outline

DRAM Background

DejaVu

- Vulnerability to Read Disturbance
- Vulnerability to Retention Failure
- Reliability of Processing-Using-DRAM Operations

Hypotheses on Root Causes

Implications

Conclusion

DejaVu - Summary

We demonstrate and analyze **DejaVu**, a new phenomenon in DRAM where **data previously written** to a DRAM row affects

> **Vulnerability to read disturbance**

> **Vulnerability to retention failure**

> **Reliability of Processing-Using-DRAM Operations**

DejaVu – Row Initialization Patterns

Different ways of initializing (writing to) a DRAM row

- **Baseline**

  DRAM row is written *only once*

  `ACT` `WR X` ×128 (Full row write) `PRE`

DejaVu Patterns

- **OverWrite**

  DRAM row is first written with *opposite data* ~X, then *overwritten* with data X

  `ACT` `WR ~X` ×128 (Full row write) `PRE` ***then*** `ACT` `WR X` ×128 (Full row write) `PRE`

- **SameWrite**

  DRAM row is written with data X *twice*

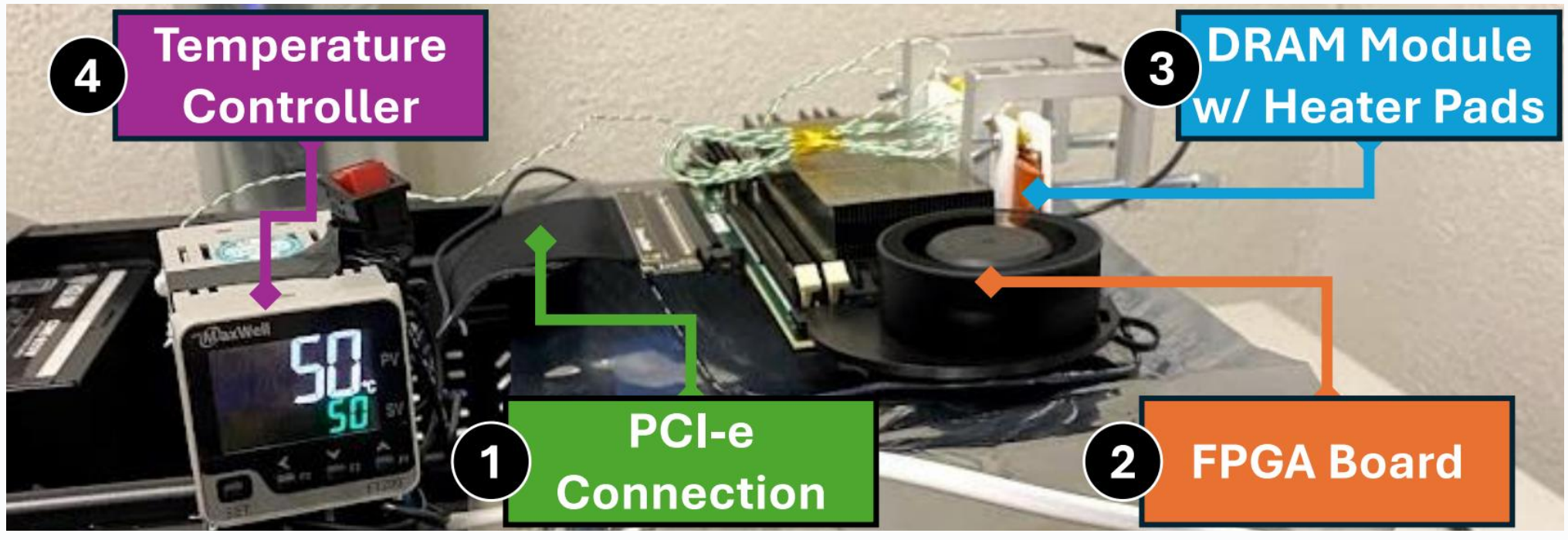
  `ACT` `WR X` ×128 (Full row write) `PRE` ***then*** `ACT` `WR X` ×128 (Full row write) `PRE`

DRAM Characterization Methodology (I)

FPGA-based real DDR4 DRAM testing infrastructure

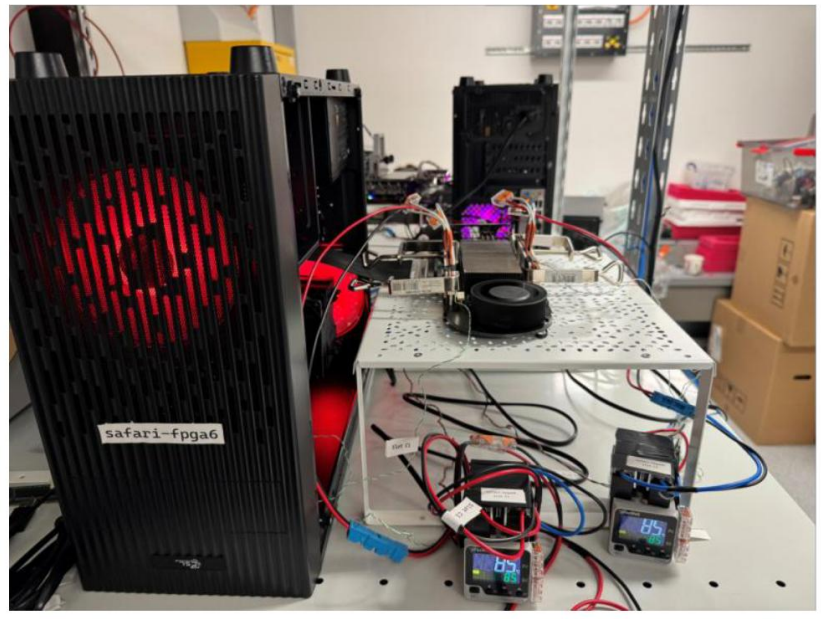
- Developed from **DRAM Bender**
- Tests **Commercial Off-The-Shelf (COTS)** DDR4 modules
- **Fine-grained control** of DRAM commands, timings, and temperature

Photo of an instance of our testing infrastructure



DRAM Characterization Methodology (I)

Photos of our laboratory for understanding DRAM



DRAM Characterization Methodology (II)

112 DDR4 DRAM chips tested (14 modules)

- All **three major DRAM manufacturers**
- Variety of **die density and revisions**

Details of the tested DDR4 DRAM chips

Mfr.	ID	Die Revision	Die Density	DQ	Num. Modules	Num. Chips
S	S0	D	8 Gb	x8	1	8
S	S1	M	16 Gb	x8	1	8
S	S2	A	16 Gb	x8	1	8
S	S3	B	16 Gb	x8	1	8
S	S4	C	16 Gb	x8	1	8
H	H0	A	8 Gb	x8	1	8
H	H1	C	8 Gb	x8	1	8
H	H2	D	8 Gb	x8	1	8
H	H3	A	16 Gb	x8	1	8
M	M0	E	8 Gb	x8	1	8
M	M1	R	8 Gb	x8	1	8
M	M2-M4	F	16 Gb	x8	3	24

Outline

DRAM Background

DejaVu

- **Vulnerability to Read Disturbance**
- Vulnerability to Retention Failure
- Reliability of Processing-Using-DRAM Operations

Hypotheses on Root Causes

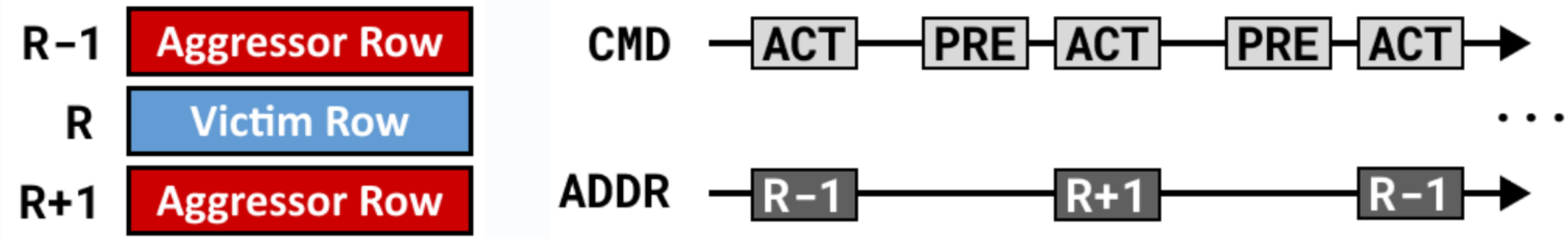
Implications

Conclusion

DejaVu on RowHammer (I) - Methodology

- **Double-sided RowHammer**

Illustrations of the double-sided RowHammer access pattern



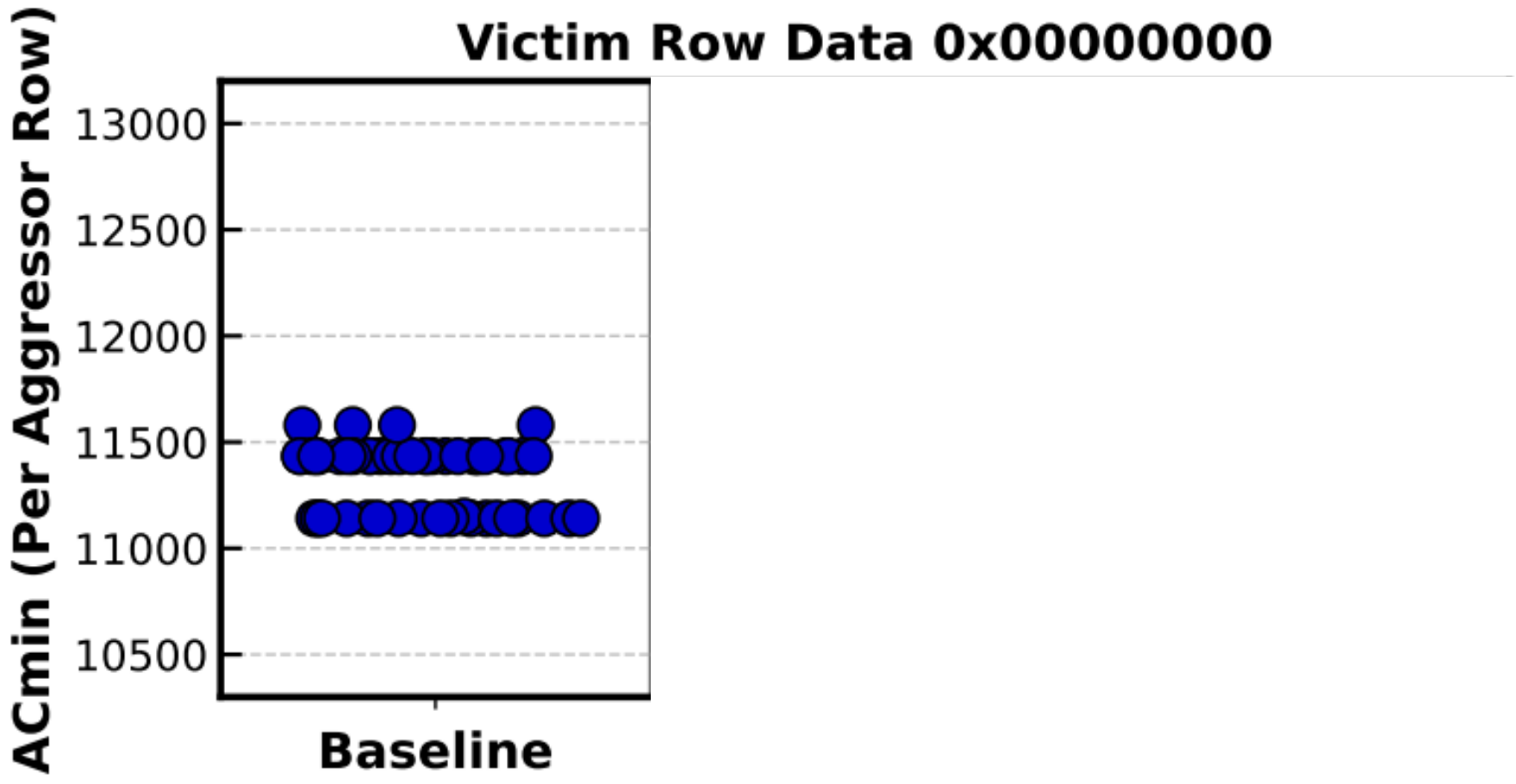
- **Key vulnerability metric: AC_{min}**

  The minimum aggressor row activation count to induce at least one bitflip in the victim row

  - 128 randomly sampled victim rows
  - 50 experiment repetitions

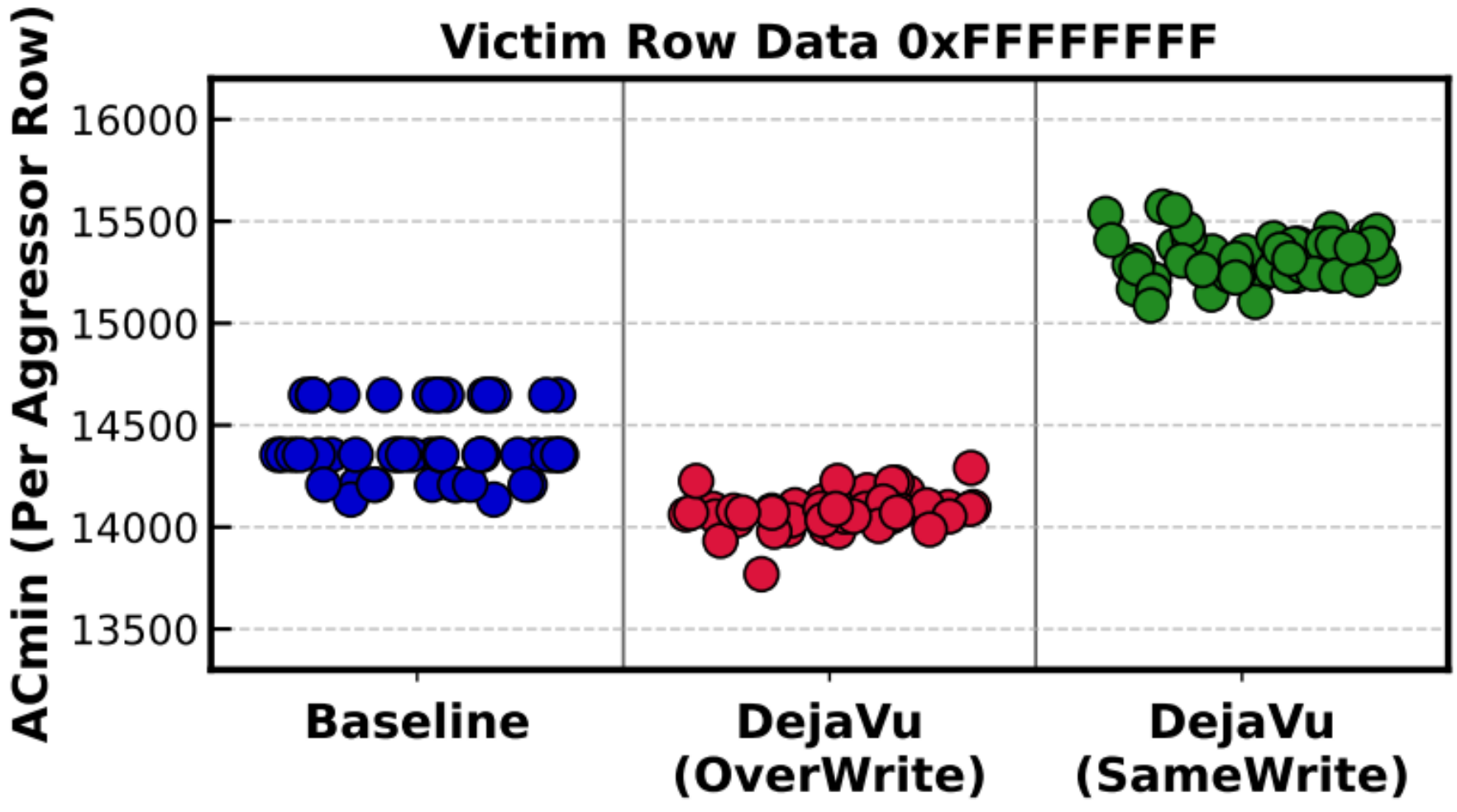
DejaVu on RowHammer (II) – Example

Double-sided RowHammer AC_{min} distribution from one example victim row across 50 test repetitions (0->1 bitflips)



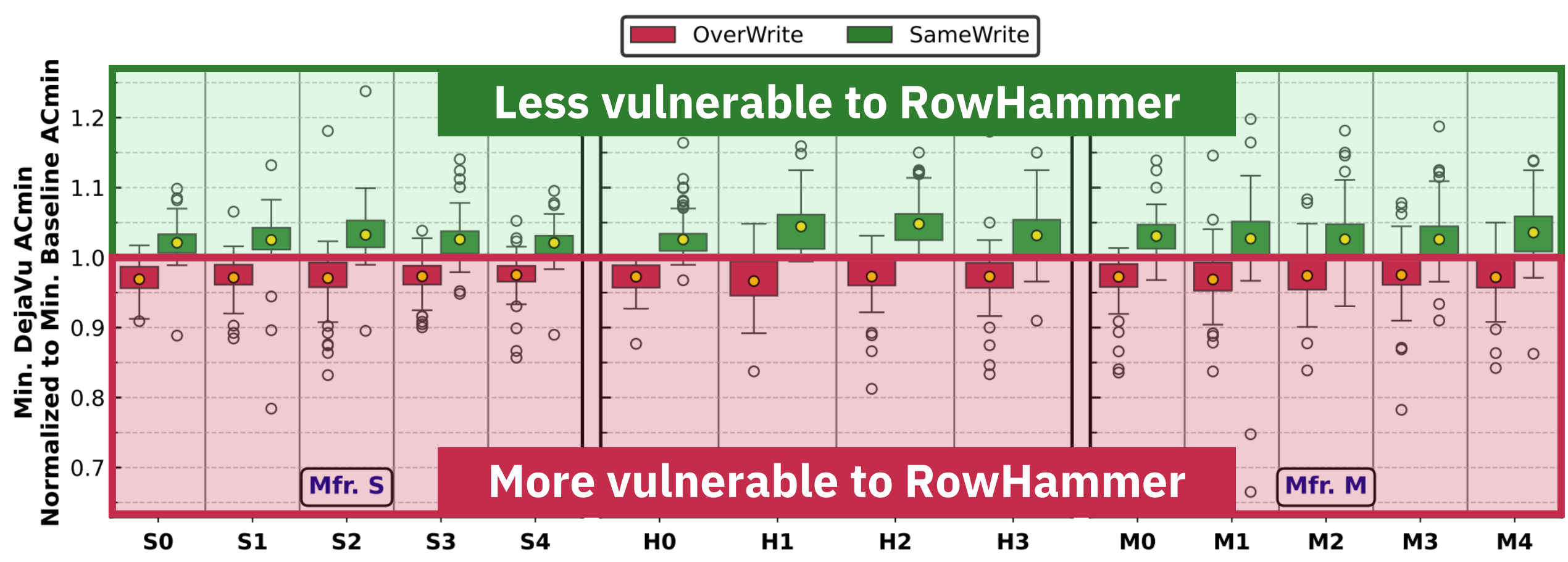
DejaVu on RowHammer (II) – Example

Double-sided RowHammer AC_{min} distribution from one example victim row across 50 test repetitions (1->0 bitflips)



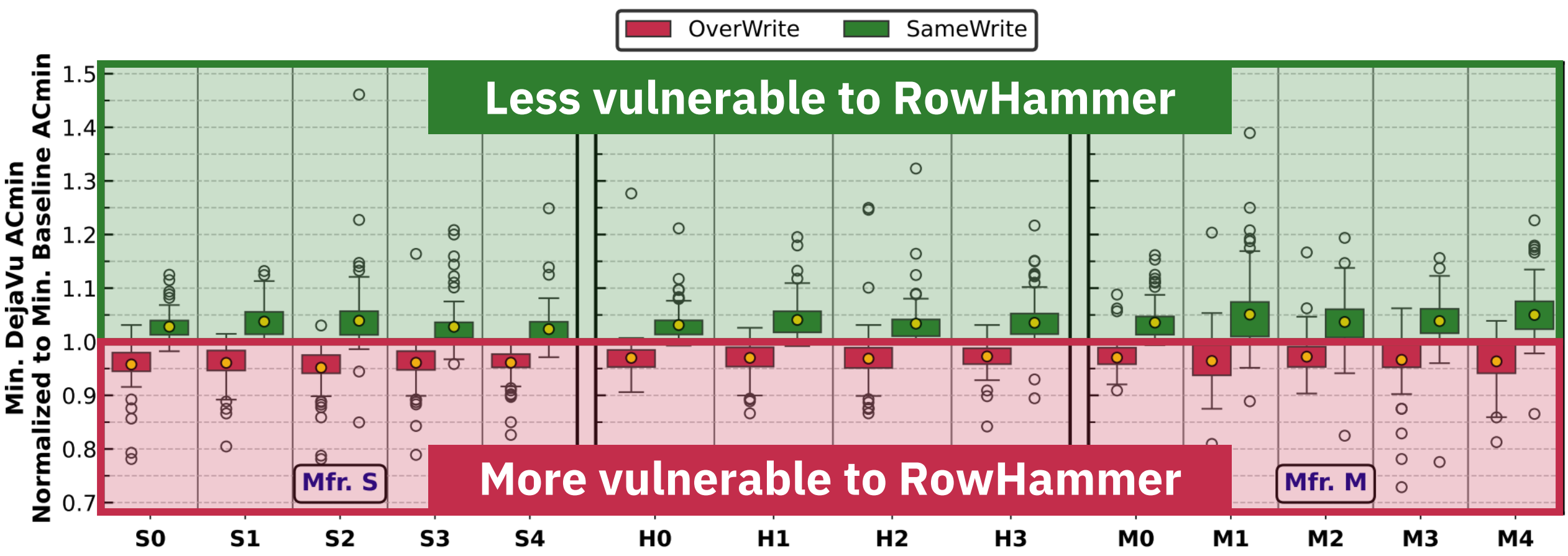
DejaVu on RowHammer (III) – Key Results

- Distribution of **minimum AC_{min}** (y-axis) across all tested rows and repetitions for each module (x-axis)
- DejaVu patterns (**OverWrite** and **SameWrite**) normalized to baseline (0->1 bitflips)



DejaVu on RowHammer (III) – Key Results

- Distribution of **minimum AC_{min}** (y-axis) across all tested rows and repetitions for each module (x-axis)
- DejaVu patterns (**OverWrite** and **SameWrite**) normalized to baseline (1->0 bitflips)



DejaVu on RowHammer (IV) - Summary

Takeaway 1

DejaVu causes **consistent changes** in RowHammer vulnerability on DRAM chips from **all three manufacturers**

Takeaway 2

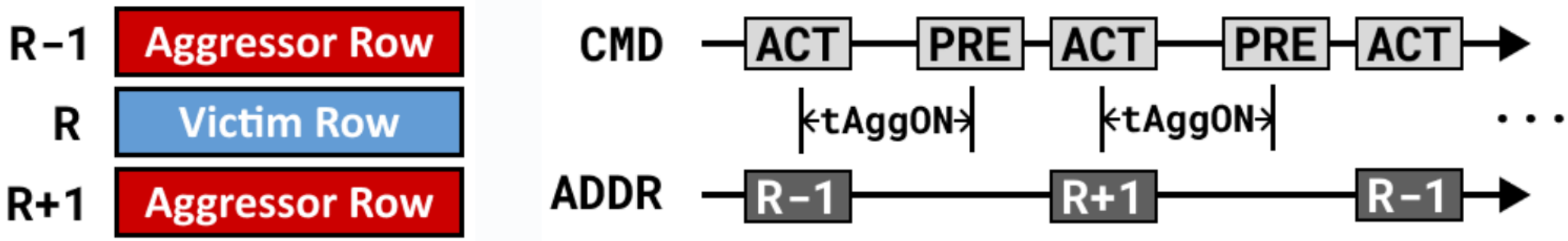
Overwriting the victim row with **opposite data** makes DRAM **more vulnerable** to RowHammer

Writing the **same data** to the victim row **twice** makes DRAM **less vulnerable** to RowHammer

DejaVu on RowPress (I) - Methodology

- **Double-sided RowPress**

Illustrations of the double-sided RowPress access pattern



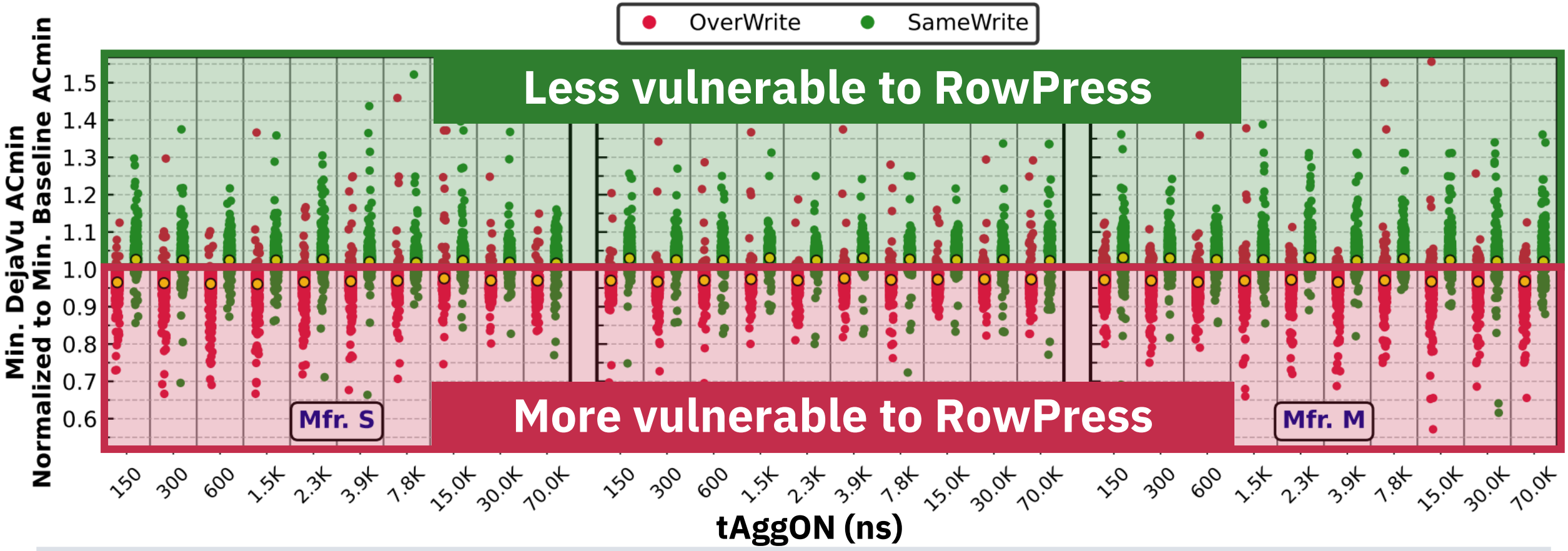
- **Key metric: AC_{min} at different tAggON values**

  The minimum aggressor row activation count to induce at least one bitflip in the victim row

  - 128 randomly sampled victim rows
  - 50 experiment repetitions

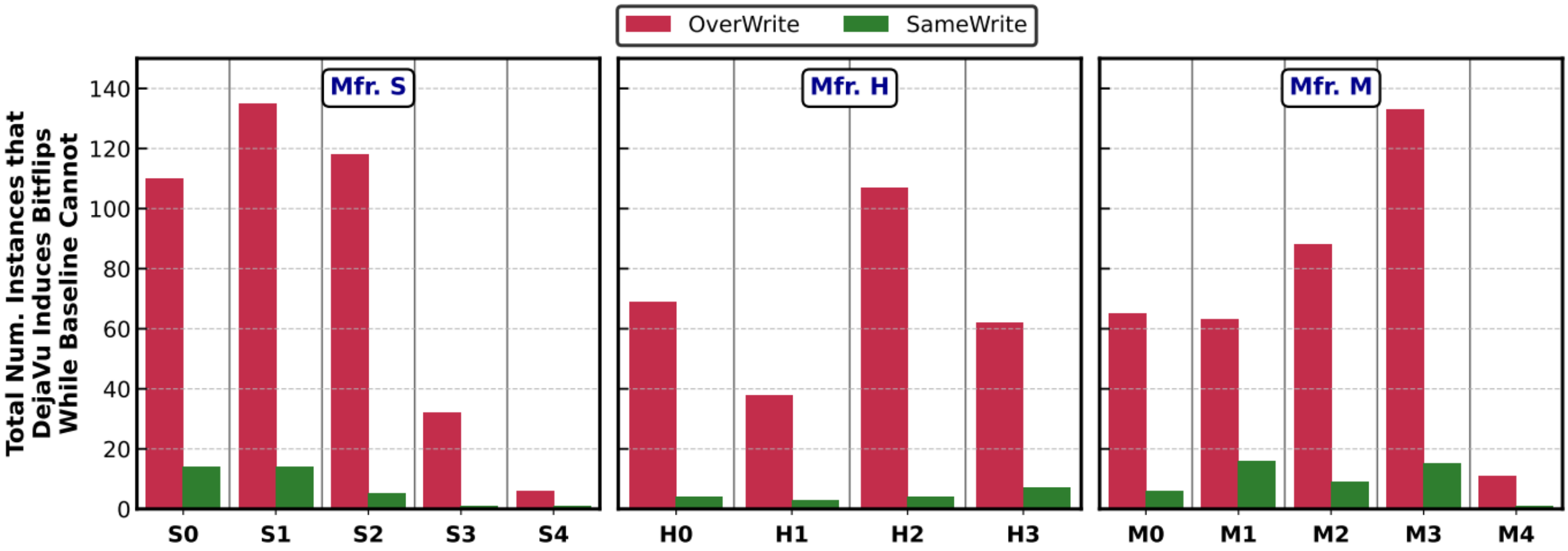
DejaVu on RowPress (II)

- Distribution of **minimum AC_{min}** (y-axis) across all tested rows and repetitions from all modules for different tAggON values (x-axis)
- DejaVu patterns (**OverWrite** and **SameWrite**) normalized to baseline (1->0 bitflips)



DejaVu on RowPress (III)

Number of instances (y-axis) DejaVu patterns (**OverWrite** and **SameWrite**) can **induce RowPress bitflips** when the **baseline cannot** across all tested modules (x-axis)



DejaVu on RowPress (IV) - Summary

Takeaway 3

DejaVu consistently makes DRAM **less** (**SameWrite**) or **more** (**OverWrite**) **vulnerable to RowPress**; similar to RowHammer

Takeaway 4

OverWrite **significantly worsens** DRAM's vulnerability to RowPress by **inducing bitflips the baseline pattern cannot**

Outline

DRAM Background

DejaVu

- Vulnerability to Read Disturbance
- **Vulnerability to Retention Failure**
- Reliability of Processing-Using-DRAM Operations

Hypotheses on Root Causes

Implications

Conclusion

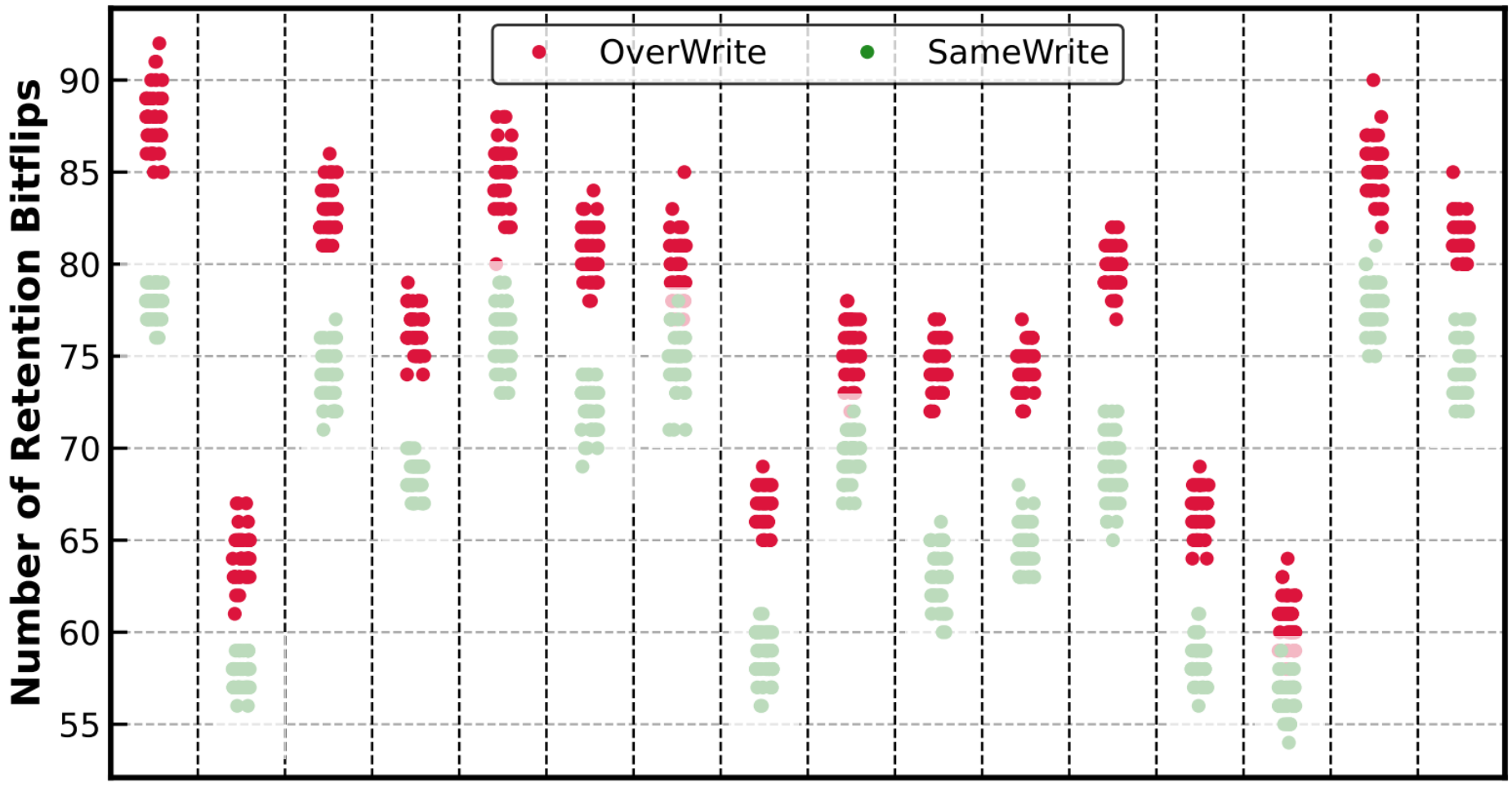
DejaVu on Retention Failure (I) - Methodology

- **Retention failure characterization workflow**
  1. Set DRAM temperature to **95°C**
  2. Write “1” to the DRAM row under test using either **SameWrite** or **OverWrite**
  3. Pause DRAM auto-refresh for **4096 ms**
  4. Read back data and check for bitflips

- **Key metric:** Number of retention failure bitflips
  - The same 128 tested rows as in read disturbance tests
  - 50 experiment repetitions

DejaVu on Retention Failure (II)

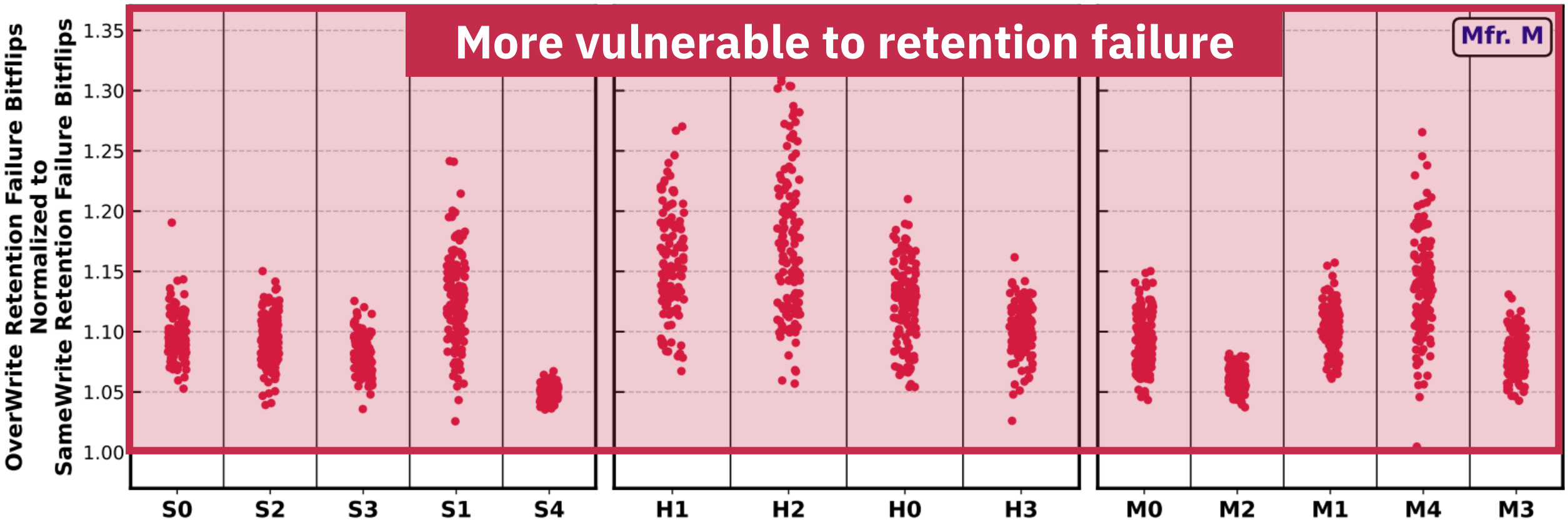
- Distribution of **retention failure bitflip counts** per row (y-axis) from 16 example rows (x-axis)
- DejaVu patterns (**OverWrite** and **SameWrite**)



DejaVu on Retention Failure (III)

- Distribution of the **average retention failure bitflip counts** (y-axis) from all tested modules (x-axis)
- **OverWrite** normalized to **SameWrite**

Avg: +10.4%, Max: +36.7%



DejaVu on Retention Failure (IV) - Takeaway

Takeaway 5

OverWrite significantly **worsens DRAM's vulnerability to retention failures**

Outline

DRAM Background

DejaVu

- Vulnerability to Read Disturbance
- Vulnerability to Retention Failure
- **Reliability of Processing-Using-DRAM Operations**

Hypotheses on Root Causes

Implications

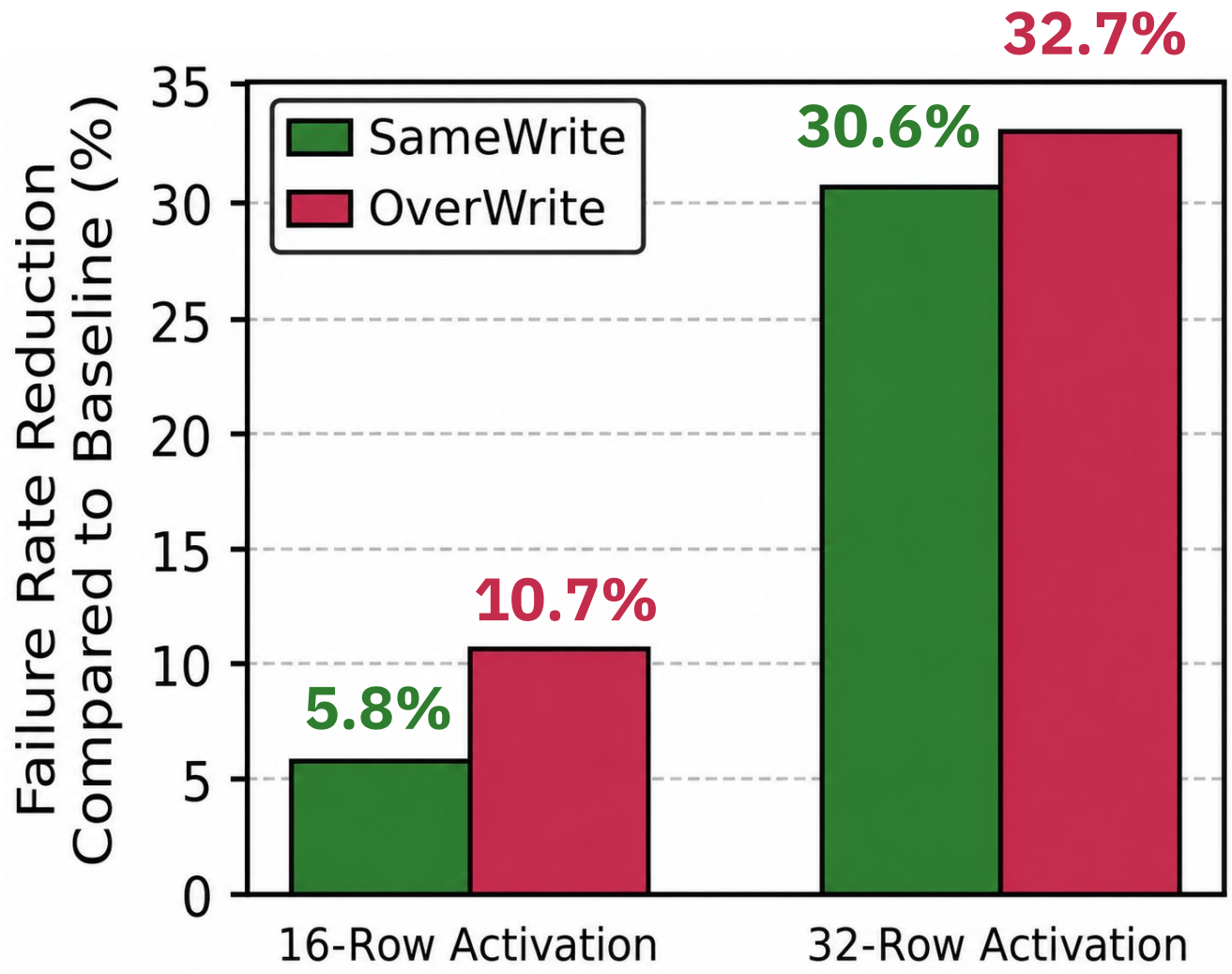
Conclusion

DejaVu on PuD Reliability (I) - Methodology

- **PuD Operations: 16- and 32-row MAJ-3 operations**
- **Input Data: Random pattern**
  - Maximize inter-bitline interference during simultaneous multi-row activation
- **Reliability Metric: Bitline Failure Rate**
  - Fraction of DRAM cells (bitlines) that produce an incorrect output in at least **1 of 1000 repetitions**
- **Tested Input Rows**
  - 3 different subarrays
  - **100 row groups** from each subarray that can perform simultaneous multi-row activation

DejaVu on PuD Reliability (II) - Results

Failure rate reduction (y-axis) of DejaVu patterns (**OverWrite** and **SameWrite**) for 16- and 32-row MAJ-3 operations



DejaVu on PuD Reliability (III) - Takeaway

Takeaway 6

Both **OverWrite** and **SameWrite** significantly **improves the reliability of Processing-Using-DRAM operations**

Outline

DRAM Background

DejaVu

- Vulnerability to Read Disturbance
- Vulnerability to Retention Failure
- Reliability of Processing-Using-DRAM Operations

Hypotheses on Root Causes

Implications

Conclusion

Hypotheses (I)

Hypothesis 1

DejaVu is caused by the **under-restoration of charge** in the capacitors of DRAM cells

Hypothesis 2

DejaVu is caused by the **change in charge-trap occupancy** in the DRAM cell access transistor

Hypotheses (II)

More investigations on the hypotheses in the paper

- Experiments to (partially) test the charge-under restoration hypothesis
- Analyses of the relationship between the trap occupancy hypothesis and PuD reliability results

Call for Contribution

We call for future works to **fundamentally understand** DRAM read disturbance and other robustness issues **across the stack: from device to circuit and to chip** to enable **high-performance, safe, secure, and robust DRAM**

Outline

DRAM Background

DejaVu

- Vulnerability to Read Disturbance
- Vulnerability to Retention Failure
- Reliability of Processing-Using-DRAM Operations

Hypotheses on Root Causes

Implications

Conclusion

Implications (I)

Characterizing read disturbance threshold

- Critical for configuring mitigation techniques
- **Overwriting data is legal and not uncommon (e.g., page zeroing)**
- Our results show OverWrite consistently reduces AC_{min} of RowHammer and RowPress

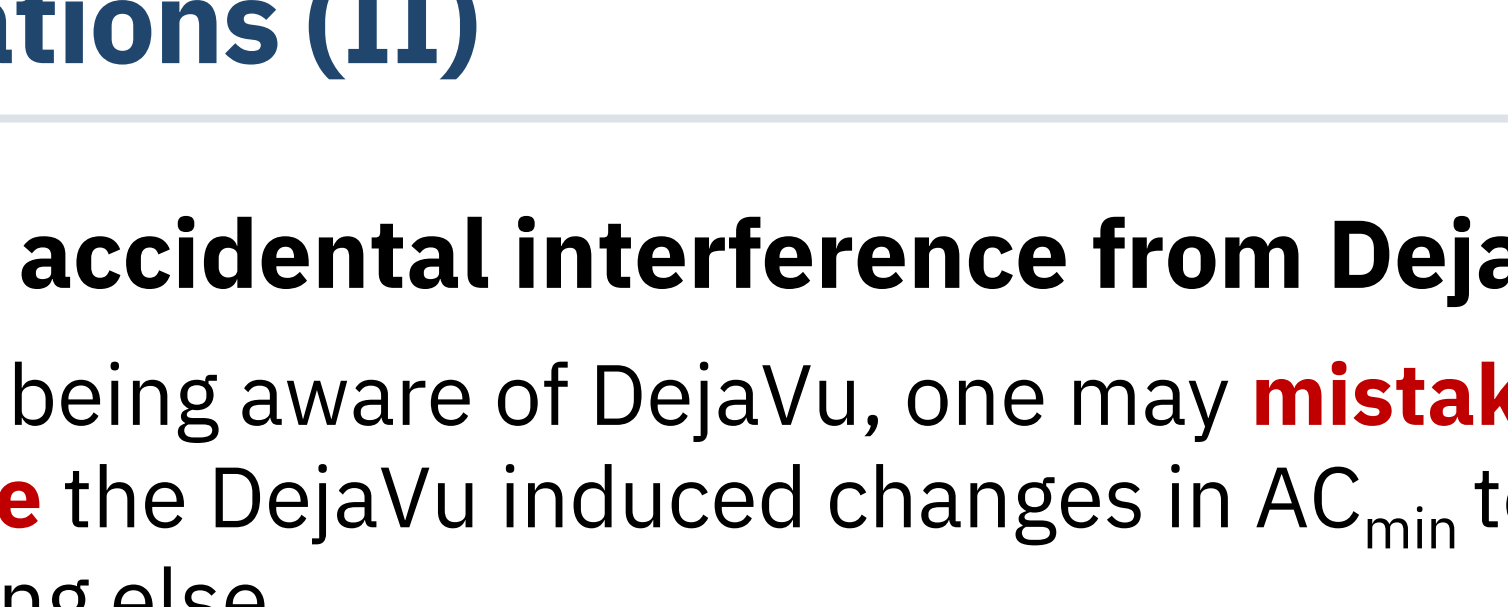
Implication 1

When characterizing DRAM to **find the minimum AC_{min}** to configure read disturbance mitigation techniques, the victim row should **always be initialized with OverWrite**

Implications (II)

Avoiding accidental interference from DejaVu

- Without being aware of DejaVu, one may **mistakenly attribute** the DejaVu induced changes in AC_{min} to something else
- **Example case study:** Does the data in rows beyond the two immediate neighboring rows of the victim row affects AC_{min} of RowHammer?



Implications (II) – Cont'd

Avoiding accidental interference from DejaVu

- It is natural to use a range-based loop to bulk-initialize many contiguous DRAM rows (i.e., from R-k to R+k), and *then initialize* row R to test different data patterns

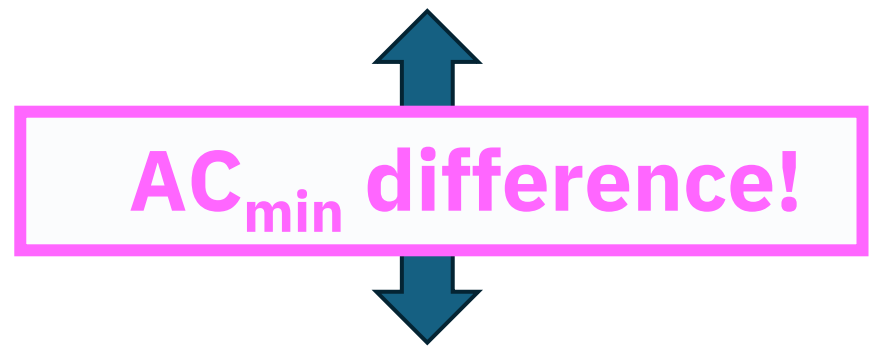
Test Pattern A – Same Data

```
for i in range(R-k, R+k):
  write_row(i, data)
write_row(R, data)
```

Test Pattern B – Opposite Data

```
for i in range(R-k, R+k):
  write_row(i, data)
write_row(R, ~data)
```

Accidental SameWrite



Accidental OverWrite

Implications (III) – Cont'd

Implication 2

Avoid accidentally comparing SameWrite to OverWrite by always writing the same data to the tested DRAM row twice (i.e., **always do SameWrite**)

More in Our Paper

DejaVu: Why You Should Write to Your DRAM Rows Twice, Carefully

Haocong Luo[1] İsmail Emir Yüksel[1] Ataberk Olgun[1]
Nisa Bostanci[1] Orhun Ecemiş[2] Abdullah Giray Yaglikci[3] Onur Mutlu[1]
[1]*ETH Zurich* [2]*TOBB ETÜ* [3]*CISPA*

Paper (arXiv)

Open-source Artifact

DejaVu Public

Edit Pins | Watch 0 | Fork 0 | Star 0

m... | Go to file | Code

RichardLuo79 Markdown formatting of ... aa7a723 · 5 days ago

characterization	Init	5 days ago
dct	Init	5 days ago

About

Artifact for the ISCA'26 paper "DejaVu: Why You Should Write to Your DRAM Rows Twice, Carefully." Described in https://arxiv.org/abs/2606.22297

Artifact (Github)

Outline

DRAM Background

DejaVu

- Vulnerability to Read Disturbance
- Vulnerability to Retention Failure
- Reliability of Processing-Using-DRAM Operations

Hypotheses on Root Causes

Implications

Conclusion

Conclusion

- We demonstrate and analyze **DejaVu**, a new phenomenon in DRAM where **data previously written** to a DRAM row affects

> **Vulnerability to read disturbance**

> **Vulnerability to retention failure**

> **Reliability of Processing-Using-DRAM Operations**

Call for Contribution

We call for future works to **fundamentally understand** DRAM read disturbance and other robustness issues **across the stack: from device to circuit and to chip** to enable **high-performance, safe, secure, and robust DRAM**

DejaVu

Why You Should Write to Your DRAM Rows Twice, Carefully

Haocong Luo

İsmail Emir Yüksel Ataberk Olgun Nisa Bostanci Orhun Ecemiş

Abdullah Giray Yaglikci Onur Mutlu

SAFARI ETH Zürich

TOBB ETÜ
University of Economics & Technology

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY