

ColumnDisturb: Understanding Column-based Read Disturbance in Real DRAM Chips and Implications for Future Systems

İsmail Emir Yüksel¹ Ataberk Olgun¹ F. Nisa Bostancı¹
Haocong Luo¹ A. Giray Yağlıkçı^{1,2} Onur Mutlu¹
¹ETH Zürich ²CISPA

We experimentally demonstrate a new widespread read disturbance phenomenon, ColumnDisturb, in real commodity DRAM chips. By repeatedly opening or keeping a DRAM row (aggressor row) open, we show that it is possible to disturb DRAM cells through a DRAM column (i.e., bitline) and induce bitflips in DRAM cells sharing the same columns as the aggressor row (across multiple DRAM subarrays). With ColumnDisturb, the activation of a single row concurrently disturbs DRAM cells across as many as three DRAM subarrays (e.g., up to 3072 DRAM rows in tested DRAM chips) as opposed to RowHammer & RowPress, which affect only a few neighboring rows of the aggressor row in a single subarray. We rigorously and comprehensively characterize ColumnDisturb and its characteristics under various operational conditions (i.e., temperature, data pattern, DRAM timing parameters, average voltage level of the bitline, memory access pattern, and spatial variation) using 216 DDR4 and 4 HBM2 chips from three major DRAM manufacturers. Among our 27 key experimental observations, we highlight two major results and their implications.

First, ColumnDisturb affects chips from all three major DRAM manufacturers and worsens as DRAM technology scales down to smaller node sizes (e.g., the minimum time to induce the first ColumnDisturb bitflip reduces by up to 5.06x and 2.96x on average across all tested DRAM modules). We observe that, even in existing DRAM chips, ColumnDisturb induces bitflips within a standard DDR4 refresh window (e.g., in 63.6 ms) in multiple cells from one module. We predict that, as DRAM technology node size reduces, ColumnDisturb would worsen in future DRAM chips, likely causing many more bitflips in the standard refresh window. Second, beyond the standard refresh window, ColumnDisturb induces bitflips in many (up to 198x) more DRAM rows than retention failures. Therefore, ColumnDisturb has strong implications for existing retention-aware refresh mechanisms that aim to improve system performance and energy efficiency by leveraging the heterogeneity in DRAM cell retention times: our detailed analyses and cycle-level simulations show that ColumnDisturb greatly reduces and can completely diminish the performance and energy benefits of such mechanisms.

Our results have serious implications for the robustness of future DRAM-based computing systems due to continued aggressive DRAM technology scaling. We describe and evaluate two solutions for mitigating ColumnDisturb bitflips and call for more research on the topic.

1. Introduction

Dynamic random access memory (DRAM) [1] is the dominant main memory technology in nearly all modern computing systems due to its latency and cost characteristics. Continuing to increase DRAM capacity requires increasing the density of

DRAM cells by reducing (i.e., scaling) the technology node size of DRAM, which in turn reduces DRAM cell size, cell-to-cell spacing, and cell-to-bitline spacing [2]. Unfortunately, as a result of this aggressive technology node scaling, DRAM suffers from worsening read disturbance problems [3–6].

Read disturbance in modern DRAM chips [4–9] is a widespread phenomenon and is reliably used for breaking memory isolation [5, 7, 9–70], a fundamental building block of robust (i.e., safe, secure, reliable, available) systems. *RowHammer* and *RowPress* are two prominent examples of DRAM read disturbance phenomena where a victim DRAM row experiences bitflips when a nearby aggressor DRAM row is 1) repeatedly activated (i.e., hammered) [5–7] or 2) kept open for a long period (i.e., pressed) [4, 71]. Many prior works [3–5, 7, 9, 41, 42] experimentally demonstrate that read disturbance significantly worsens as DRAM manufacturing technology scales to smaller node sizes in real DRAM chips. For example, chips manufactured in 2018–2020 can experience RowHammer bitflips at an order of magnitude fewer row activations compared to the chips manufactured in 2012–2013 [3]. To ensure the robustness of modern and future DRAM-based systems, it is critical to 1) develop a rigorous understanding of *known* read disturbance effects like RowHammer and RowPress and 2) discover and analyze *previously unknown* read disturbance phenomena.

In this work, for the first time, we experimentally demonstrate a new widespread read disturbance phenomenon, ColumnDisturb, in 216 commercial off-the-shelf (COTS) DDR4 and 4 HBM2 DRAM chips from all three major DRAM manufacturers. We show that it is possible to disturb many DRAM cells that are on the same *DRAM column*.¹ ColumnDisturb manifests as bitflips in DRAM cells across as many as three DRAM subarrays (e.g., up to 3072 DRAM rows in tested DDR4 chips) *concurrently* when we hammer or press an aggressor row.

Fig. 1 highlights the conceptual difference between ColumnDisturb (Fig. 1c) and RowHammer & RowPress (Fig. 1b) in modern DRAM chips (Fig. 1a).

RowHammer & RowPress affect *a few* rows within the same subarray (e.g., two victim rows in Fig. 1b). RowHammer & RowPress happen because rows are too close to each other, affecting cells that are in closeby rows (i.e., closeby rows are victims). In contrast, ColumnDisturb is fundamentally different: ColumnDisturb affects thousands of rows, including distant ones, across both the same subarray and multiple neighboring subarrays (e.g., Fig. 1c). This is because accessing a DRAM row (i.e., DRAM row activation) perturbs many columns, and columns of a single DRAM row span multiple subarrays. Thus, ColumnDisturb affects cells that are vertically

¹In DRAM, each *logical* column is implemented using multiple physical columns, i.e., bitlines (e.g., [72–74]). In this paper, we use the term *DRAM column* to refer to a single physical column/bitline.

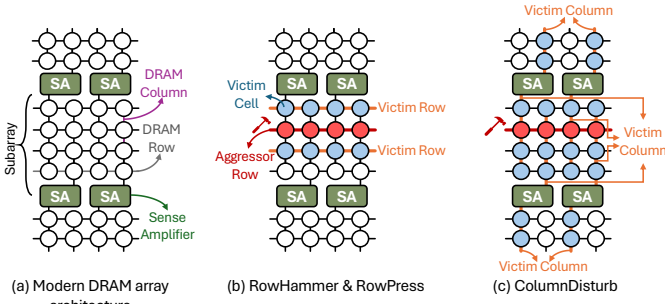


Figure 1: (a) Modern DRAM array architecture, (b) RowHammer & RowPress induce bitflips in a few neighboring rows of the aggressor row, where rows are the victims, (c) ColumnDisturb, a column-based read disturbance phenomenon that affects many rows in multiple subarrays, where columns are victims.

connected to the same bitline (i.e., columns are victims).

Our experimental results (§4 and §5) and hypothetical explanations (§4.6) for ColumnDisturb strongly suggest that ColumnDisturb induces bitflips due to *bitline-voltage-induced disturbance*. In other words, “hammering” (repeatedly raising and lowering) or “pressing” (keeping high or keeping low) a *bitline’s* voltage is what leads to ColumnDisturb bitflips.

To empirically illustrate the ColumnDisturb phenomenon, Fig. 2 shows the bitflips caused by ColumnDisturb, RowHammer, RowPress, and by retention failures in the first 3072 rows of a representative Samsung 16Gb A-die DDR4 DRAM module, spanning the first three consecutive subarrays. The x-axis shows the DRAM row address, and the y-axis shows the number of bitflips in each row.²

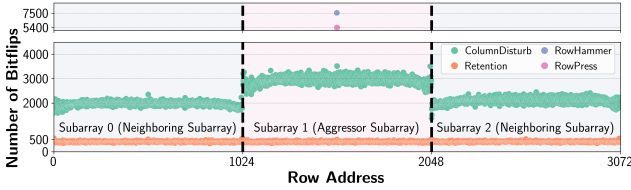


Figure 2: Number of ColumnDisturb, RowHammer, RowPress, and retention failure bitflips in three subarrays.

We observe that ColumnDisturb induces bitflips across three consecutive subarrays, and these bitflips are *not* due to retention failures. Nor are they due to RowHammer & RowPress, which affect only immediate neighboring rows of the aggressor row.

Based on our rigorous experimental characterization of ColumnDisturb and its behavior under numerous parameters (i.e., temperature, data pattern, DRAM timing parameters, average voltage level of the bitline, memory access pattern, and spatial variation), we make 27 empirical observations and share 12 key takeaway lessons. We highlight two of our major new results.

First, newer DRAM chips manufactured with smaller technology nodes are increasingly more vulnerable to ColumnDisturb across all three major manufacturers. For example, we find that the minimum time to induce the first ColumnDisturb bitflip reduces by up to 5.06x and 2.96x on average across all tested modules. We observe that even in *existing* COTS DRAM chips, ColumnDisturb induces bitflips within a nominal DDR4 refresh window (e.g., in 63.6ms) in multiple cells from a single module. Second, significantly more rows experience ColumnDisturb bit-

flips than retention failures, and the number of ColumnDisturb bitflips is much higher than retention failures in all tested refresh intervals (i.e., 64ms, 128ms, 256ms, 512ms, and 1024ms) across all tested DRAM modules. For example, with a 512ms refresh interval and at 65 °C, ColumnDisturb induces 1.64x, 62.49x, and 152.66x more bitflips in 2, 6, and 232 more rows on average for tested SK Hynix, Micron, and Samsung modules, respectively, than retention failures.

Our observations show that ColumnDisturb has serious implications on the robustness of both 1) future systems and 2) existing retention-aware heterogeneous refresh mechanisms [75–88]. First, due to continuously shrinking DRAM node size, more ColumnDisturb bitflips may manifest in the standard refresh window in future DRAM chips, jeopardizing the robustness of future systems. We describe and evaluate two hardware techniques that could mitigate ColumnDisturb bitflips at varying expected performance, energy, and area overheads. A straightforward solution is to increase the DRAM refresh rate to accommodate ColumnDisturb bitflips that could happen in the standard refresh window. However, this straightforward solution reduces system throughput by 42.1% and increases system energy consumption by 67.5%. We instead propose to intelligently and timely refresh *only the victim rows* that are vulnerable to ColumnDisturb. We show that our improved solution reduces the straightforward solution’s 1) system throughput overhead by 70.5% and 2) energy overhead by 73.8%. Second, we evaluate a retention-aware refresh mechanism (RAIDR) [76] and demonstrate that its benefits drastically decrease in the presence of ColumnDisturb (e.g., 53% decrease in performance) compared to a baseline RAIDR that does not suffer from ColumnDisturb.

We call for future research to 1) fundamentally understand ColumnDisturb at the device-level, 2) architect ColumnDisturb-resilient, high-performance retention-aware refresh mechanisms, and 3) other innovative solutions to mitigate ColumnDisturb bitflips to enable ColumnDisturb-resilient, robust future computing systems.

This paper makes the following key contributions:

- This is the first work to experimentally demonstrate a *column-based* (i.e., bitline-based) read disturbance phenomenon in modern DRAM chips, ColumnDisturb, and its widespread existence in real DDR4 chips from all three major DRAM manufacturers and HBM2 chips from Samsung.
- We provide an extensive experimental characterization of ColumnDisturb on 216 real DDR4 and 4 HBM2 DRAM chips. ColumnDisturb induces bitflips across three subarrays (e.g., 3072 rows), greatly more than RowHammer & RowPress that affect only a few rows in a single subarray.
- Our experimental results show that ColumnDisturb 1) gets worse as DRAM technology scales down to smaller cell sizes, 2) already induces bitflips within a standard refresh window in some existing DRAM chips, and 3) induces significantly more bitflips in many more rows than retention failures, for a given refresh interval.
- We describe and evaluate two solutions to ColumnDisturb for future DRAM-based systems.
- We evaluate a retention-aware heterogeneous refresh mechanism (RAIDR [76]) and show that ColumnDisturb can completely diminish the performance and energy benefits of such mechanisms.

²Please see §4.2 for the detailed methodology and observations. We study ColumnDisturb in great detail in §4 and §5.

2. Background & Fundamentals

2.1. Dynamic Random Access Memory (DRAM)

DRAM Organization. Fig. 3 shows the hierarchical organization of modern DRAM-based main memory. A module contains one or multiple DRAM ranks that time-share the memory channel. A rank consists of multiple DRAM chips. Each DRAM chip has multiple DRAM banks, each containing multiple subarrays [89–91]. Within a subarray, DRAM cells form a two-dimensional structure interconnected over *bitlines* and *wordlines*. Each cell encodes one bit of data using the charge level in its capacitor. A true cell encodes data ‘1’ as fully charged (VDD) and data ‘0’ as fully discharged (GND), whereas an anti-cell uses the opposite encoding. The data in the cell is accessed through an access transistor, driven by the wordline to connect the cell capacitor to the bitline. The row decoder decodes the row address and drives one wordline. A row of DRAM cells on the same wordline is referred to as a DRAM *row*. DRAM cells in the same *column* are connected to the sense amplifier via a bitline. To sense an entire row of cells, each subarray has bitlines that connect to two rows of sense amplifiers, one above and one below the subarray, which causes neighboring subarrays to share half of the bitlines [92–100]. As a result, a subarray’s even bitlines (highlighted in red for the middle subarray, Subarray 1 in Fig. 3) are connected to one of its neighboring subarrays’ odd bitlines (highlighted in red for the top subarray, Subarray 0 in Fig. 3), while the odd bitlines of Subarray 1 (highlighted in blue for Subarray 1) are connected to the other neighboring subarray’s even bitlines (highlighted in blue for the bottom subarray, Subarray 2 in Fig. 3) via sense amplifiers. This approach, known as the *open-bitline architecture*, is widely adopted in high-density DRAM [92–100].

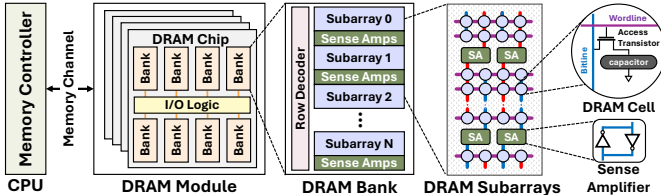


Figure 3: Organization of modern DRAM-based main memory.

DRAM Access. Accessing DRAM consists of two steps. First, the memory controller issues an ACT (activate) command together with a row address to the bank. The row decoder drives the wordline of that row to activate the row (i.e., enables the access transistors). Data is then transferred from the DRAM cells in the row to the row buffer through the bitlines. Second, the memory controller sends a PRE (precharge) command to close the activated row after waiting for t_{RAS} (i.e., the minimum time between opening a row with an ACT command and closing the row with a PRE command). Before accessing another row in the same bank, the memory controller must wait for t_{RP} (i.e., the minimum time between sending a PRE command and opening a row with an ACT command).

2.2. DRAM Read Disturbance

Read disturbance is the phenomenon that reading data from a memory or storage device causes physical disturbance (e.g., voltage deviation, electron injection, electron trapping) on another piece of data that is *not* accessed but located physically nearby the accessed data. Two prime examples of read disturbance in modern DRAM chips are RowHammer [4–7, 9, 101]

and RowPress [4, 102], where repeatedly accessing (hammering) or keeping active (pressing) a row induces bitflips in physically nearby rows, respectively. For read disturbance bitflips to occur, 1) the aggressor row needs to be activated more times than a certain threshold value [7], and/or 2) the aggressor row needs to be open for a long period of time (i.e., $t_{AggOn} > t_{RAS}$) [4].

2.3. DRAM Refresh and Retention Failures

DRAM Refresh. DRAM cell capacitors inherently lose charge over time, potentially resulting in data corruption [76, 85, 86]. A DRAM cell’s retention time defines how long it can reliably store data and typically varies between cells from milliseconds to many hours [76, 84–86, 103–106]. To prevent retention failures from appearing, the memory controller periodically restores each DRAM row’s charge level by sending a REF (refresh) command every t_{REFI} , e.g., [47, 72, 85, 107, 108], which is scheduled according to a timing parameter called the refresh window (t_{REFW}) [72, 85, 107, 108].

DRAM Retention Failures. A modern DRAM cell with Buried Channel Access Transistor (BCAT) [109, 110] has multiple leakage paths [111–113]. Prior works show that the major leakage paths include Gate-Induced Drain Leakage and Junction Leakage [109, 110, 113–116]. These major leakage paths are all from the cell to the substrate of the access transistor, which is biased at a negative voltage to reduce subthreshold leakage through the transistor [113], causing true-cells storing data ‘1’ much more likely to experience retention failures than ‘0’ [85, 86, 117]. As DRAM technology node size reduces, the closer proximity between the capacitor contact and the bitline also creates potential leakage paths [2].

3. Evaluation Infrastructure & Methodology

We describe our commercial off-the-shelf (COTS) DRAM testing infrastructure (§3.1) and our testing methodology (§3.2).

3.1. COTS DRAM Testing Infrastructure

We conduct COTS DRAM chip experiments using DRAM Bender [118, 119] (built upon SoftMC [120, 121]), an FPGA-based DDR4 and HBM2 testing infrastructure that provides precise control of DDR4 and HBM2 commands issued to a DRAM module. Fig. 4 shows one of our experimental setups on DDR4 modules that consists of four main components: 1) a host machine that generates the test program and collects experiment results, 2) an FPGA development board [122], 3) a temperature sensor and heater pads pressed against the DRAM chips to maintain a target temperature level, and 4) a temperature controller [123] to control the temperature. Fig. 5 shows our laboratory comprising many DDR4/HBM2 testing platforms.

Real DDR4 and HBM2 DRAM Chips Tested. Table 1 provides 216 real DDR4 chips from 28 modules and 4 HBM2 chips that we characterize.^{3,4}

Logical-to-Physical Row Mapping. DRAM manufacturers use mapping schemes to translate logical addresses to physical row addresses [7, 14, 32, 42, 85, 98, 106, 128–134]. To account

³The technology node size of a DRAM chip is usually not publicly available. Prior works [3, 4, 71, 124–126] assume that for a given chip manufacturer and chip density, the alphabetical order of die revision codes may provide an indication of technology node advancement.

⁴We provide much more detail on the tested DRAM chips in the extended version of this paper [127].

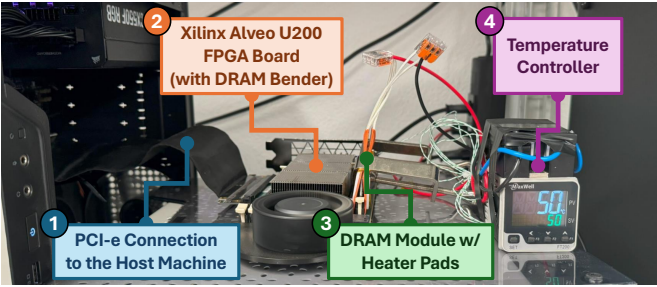


Figure 4: Our DRAM Bender [119] based experimental setup.



Figure 5: Our laboratory for real DRAM chip experiments.

for in-DRAM row address mapping and internal row remapping (which are extensively discussed in [3, 47, 135]), we reverse engineer the physical row address layout, following prior works’ methodology [3, 4, 47, 71, 135–138].

3.2. Experimental Methodology

Characterizing and understanding ColumnDisturb requires 1) reverse engineering the subarray boundaries because the columns of a single DRAM row span three physically consecutive subarrays, and thus, ColumnDisturb bitflips can manifest across three consecutive subarrays (as shown in Fig. 2), 2) filtering out retention failures, since ColumnDisturb induces bitflips within intervals longer than the refresh window (t_{REFW}), and 3) testing many parameters to understand how operational conditions and parameters affect the ColumnDisturb vulnerability.

Metrics. To characterize a DRAM module’s vulnerability to ColumnDisturb, we examine three metrics: 1) minimum time to induce the first ColumnDisturb bitflip in a subarray, 2) the fraction of cells with bitflips in a subarray, and 3) the number of rows that experience at least one ColumnDisturb bitflip in a subarray (i.e., blast radius [3, 4, 7, 47, 50, 71, 125, 126, 136–139]). A shorter time to induce the first bitflip indicates higher vulnerability to ColumnDisturb. Higher values for the fraction of cells with bitflips in a subarray and blast radius indicate higher ColumnDisturb vulnerability.

Access Pattern. To induce ColumnDisturb bitflips in a subarray, we perform the following key DRAM command sequence:

$$\text{ACT } R_{\text{Agg}} \xrightarrow{t_{\text{AggOn}}} \text{PRE} \xrightarrow{t_{\text{RP}}} \text{ACT } R_{\text{Agg}} \xrightarrow{t_{\text{AggOn}}} \dots$$

where R_{Agg} is the aggressor row in the tested subarray, t_{AggOn} is the duration that the aggressor row stays open (i.e., the timing delay between ACT R_{Agg} and PRE) [4], and t_{RP} is the timing delay before issuing the next ACT R_{Agg} command. We conduct all experiments where R_{Agg} is the middle row in the tested subarray unless stated otherwise.

By repeatedly hammering or pressing R_{Agg} with our access pattern (e.g., the aggressor row in Fig. 1-c), we disturb DRAM cells through the columns (e.g., victim columns in Fig. 1c),

Table 1: Summary of DDR4 and HBM2 DRAM chips tested.

Chip Mfr.	Module IDs	#Chips	Die Rev.	Density	Org.
SK Hynix	H0,H1,H2	24	A	8Gb	x8
	H3,H4,H5,H6	32	D	8Gb	x8
	H7	8	A	16Gb	x8
	H8,H9	16	C	16Gb	x8
Micron	M0	8	B	4Gb	x8
	M1,M2,M3	24	R	8Gb	x8
	M4, M5	16	B	16Gb	x8
	M6, M7	8	E	16Gb	x16
	M8,M9,M10,M11	32	F	16Gb	x8
Samsung	S0, S1	16	A	16Gb	x8
	S2, S3	16	B	16Gb	x8
	S4, S5	16	C	16Gb	x16
Samsung	HBM2 Chips	4	N/A	N/A	N/A

^a We report “N/A” if the information is not publicly available.

since a row drives its content to all columns in a subarray after a successful ACT command. As a result, hammering repeatedly raises and lowers the voltage of the columns, while pressing keeps the voltage consistently high or low. We conduct all experiments using the same access pattern and record the bitflips in the aggressor row’s subarray unless stated otherwise.

Algorithm to Find Time to Induce the First Bitflip. For every parameter we evaluate, we determine the minimum number of ACT commands (i.e., hammer count) required to induce the first bitflip per tested subarray, using the bisection-method algorithm used by prior works [4, 71, 102, 124, 126, 136, 138]. We terminate the search when the difference between the current and previous minimum hammer count measurements is smaller than 1% of the previous measurement. During these experiments, we do not issue any REF commands for 512 ms (i.e., the refresh interval is 512 ms). If no ColumnDisturb bitflips are observed within 512 ms in a given subarray, we terminate the search for that subarray and proceed to the next one. For every tested subarray, we repeat the search five times and convert the minimum hammer count value across five iterations to time.

DRAM Subarray Boundaries. Prior works [138, 140–146] demonstrate that real DRAM chips are capable of performing the RowClone operation [90], an in-DRAM data copy operation within a subarray by performing two consecutive row activations, which leads to data in the first activated row (i.e., source row) to be copied to the second activated row (i.e., destination row) via the sense amplifiers connected to both rows. We repeatedly perform the RowClone operation for every possible source and destination row address in each tested bank. When we observe that the destination row gets the same content as the source row after the RowClone operation, we conclude that the source row and the destination row are in the same subarray. Based on this observation, we reverse engineer the subarray boundaries and determine which rows are in the same subarray.

Filtering Out Retention and RowHammer & RowPress Failures. We experimentally identify cells that experience retention failures during our tests and exclude them from the set of bitflips caused by ColumnDisturb in two steps. First, we follow the state-of-the-art retention time test methodology [84–86, 117] for five different data patterns. Second, to cover the worst-case in the presence of the variable retention time phenomenon [85, 86, 115, 116], we repeat each test 50 times and draw our conclusions based on the lowest observed retention time of a cell. To filter out RowHammer and RowPress bitflips, in every experiment, we exclude eight nearest

physical victim rows of the aggressor row when we read from a subarray, since RowHammer & RowPress induce bitflips in nearby neighboring rows of the aggressor row. We exclude eight nearest neighbor rows of the aggressor, even though we experimentally find that RowHammer & RowPress induce bitflips in only +1/-1 neighboring rows. We add this guardband (i.e., six additional rows to exclude) because the state-of-the-art industry solutions to DRAM read disturbance support refreshing RowHammer & RowPress bitflips in up to eight nearest neighboring rows [41, 47, 147–149].⁵

Eliminating Other Interference Sources. We eliminate other potential sources of interference by taking two measures, similar to prior works [3, 4, 47, 71, 136]. First, we disable periodic refresh to prevent refreshing any rows so that we can observe the DRAM inherent device-level behavior. Second, we verify that the tested modules and chips have neither rank-level nor on-die ECC [134, 150, 151].

Test Parameters. We explain the three common parameters we use in testing and elaborate on the specific parameters of each test in §4 and §5. 1) *Data Pattern*: We use five data patterns 0x00, 0xAA, 0x11, 0x33, and 0x77, commonly used in memory reliability testing [152] and by prior work on DRAM characterization (e.g., [3, 4, 7, 71, 138]). We fill aggressor rows with these data patterns while initializing victim rows with the negated data pattern (e.g., if aggressors have 0x00, victim rows are filled with 0xFF). 2) *Temperature*: We perform our experiments at four temperature levels: 45°C, 65°C, 85°C, and 95°C. We conduct all experiments at 85°C unless stated otherwise. 3) *Aggressor Row On Time (t_{AggOn})*: We define t_{AggOn} [4] as the time an aggressor row stays open after each activation during a ColumnDisturb test. We perform our experiments at four t_{AggOn} values: 36ns, 7.8μs, 70.2μs, and 1ms. We conduct all experiments at $t_{\text{AggOn}}=70.2\mu\text{s}$ unless stated otherwise. 4) *Refresh Interval*: We perform our experiments with various different refresh intervals from 64ms to 16s.

4. Foundational Results

We demonstrate a new read disturbance phenomenon, ColumnDisturb, in real DDR4 and HBM2 DRAM chips. ColumnDisturb is a widespread read disturbance phenomenon in real DRAM chips and worsens as DRAM technology scales down to smaller node sizes (§4.1). We observe that ColumnDisturb can induce bitflips in three consecutive subarrays by hammering or pressing an aggressor row (§4.2). We investigate the characteristics of ColumnDisturb by analyzing the direction of bitflips (§4.3), the effect of data pattern on a DRAM column (§4.4), the effect of aggressor row on time (§4.5), the effect of average voltage level on a DRAM column (§4.6), the number of DRAM rows that experience ColumnDisturb bitflips (§4.7), and ColumnDisturb on HBM2 DRAM chips (§4.8).

4.1. Prevalence & Scaling Characteristics

We study the vulnerability of each chip to ColumnDisturb. One critical component of vulnerability is identifying the weakest cell (i.e., the cell that fails earliest). Since ColumnDisturb is effective at subarray granularity (see §1 and §4.2), we check the subarray that the aggressor row (the middle row in the

tested subarray⁶) belongs to and record the time to induce the first ColumnDisturb bitflip in that subarray for each chip (see §3.2 for more details on our complete methodology). In this experiment, we test *all* subarrays in *all* banks across *all* tested DRAM modules from *all* three major manufacturers using the parameters with which a DRAM chip experiences the highest ColumnDisturb vulnerability (determined through extensive experiments in §4 and §5).

Fig. 6 shows a violin plot of the time to observe the first ColumnDisturb bitflip in a subarray across all tested 46,080 subarrays (y-axis) for each different die revision and density pair (x-axis). Each subplot is dedicated to a different manufacturer.

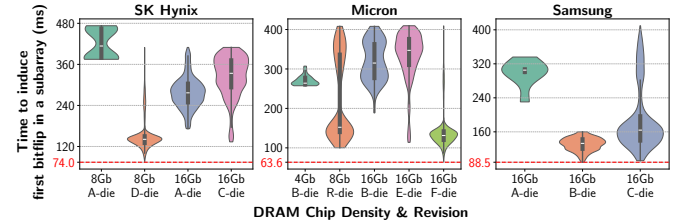


Figure 6: Distribution of the time to induce the first ColumnDisturb bitflip in a subarray for different DRAM chip densities & die revisions across three major manufacturers.

Observation 1. All tested 216 DDR4 chips are vulnerable to ColumnDisturb.

We observe that in every DDR4 chip tested, there is at least one DRAM cell that experiences a ColumnDisturb bitflip.

Observation 2. Newer chips tend to be more vulnerable to ColumnDisturb bitflips.

We observe that in the same die density for each tested manufacturer, more advanced technology nodes experience higher ColumnDisturb vulnerability (i.e., lower time to induce the first ColumnDisturb bitflip).⁷ For SK Hynix, for 8Gb and 16Gb chip density, respectively, the minimum time to induce the first ColumnDisturb bitflip in a subarray reduces by 5.06x and 1.29x as die generation advances (i.e., 8Gb A- to D-die and 16Gb A- to C-die). For Micron, for 16Gb chip density, from B- to F-die, the minimum time to induce the first ColumnDisturb bitflip in a subarray reduces by 2.98x. For Samsung, compared to 16Gb A-die, the minimum time to induce the first ColumnDisturb bitflip in a subarray is 2.50x lower in 16Gb C-die.

We hypothesize that as DRAM technology node advances, the distance between the DRAM cell capacitor contact and the bitline decreases, strengthening tunneling effects between them [2]. Thus, DRAM chips become more vulnerable to ColumnDisturb as DRAM technology node size reduces. §4.6 provides detailed analyses and hypotheses on the device-level mechanisms of ColumnDisturb.

Observation 2 has serious implications for the future as DRAM technology node sizes will continue to reduce and the time to induce the first ColumnDisturb bitflip will likely get smaller. §6.1 discusses the implications of this observation.

⁶We test different aggressor row locations and observe the almost same trend in §5.5.

⁷For a given manufacturer and die density, the later in the alphabetical order the die revision code is, the more likely the chip has a more advanced technology node [3, 4, 71].

Observation 3. There already are some real DRAM chips where ColumnDisturb introduces bitflips within the nominal refresh window under nominal operating conditions.

We observe that in a single 16Gb F-die Micron module, multiple cells (at least four cells) from multiple chips (at least three chips) experience ColumnDisturb bitflips at 63.6ms and 85°C, within the nominal refresh window for DDR4 chips, t_{REFW} . In contrast, retention failures require at least 512ms (at 85°C) to manifest for the same module (see Fig. 11). We also observe that the closest (farthest) victim row that experiences ColumnDisturb bitflips within t_{REFW} is 374 (446) rows away from the aggressor row where both victim row and the aggressor row are in the same subarray.

Takeaway 1. ColumnDisturb is a widespread read disturbance phenomenon in real DRAM chips. ColumnDisturb becomes worse as DRAM technology scales down to smaller node sizes. ColumnDisturb can already induce bitflips within the refresh window in current DRAM chips.

4.2. Disturbing DRAM Columns

Fig. 2 in §1 shows the ColumnDisturb, RowHammer, RowPress, and retention failure bitflips from a single representative module (S0; Samsung 16Gb A-die).⁸ For RowHammer (RowPress), the same aggressor row (row 1536) is hammered (pressed, i.e., with $t_{AggOn}=70\mu s$) for 16 seconds. For the retention failure test, the bank remains idle (i.e., in precharged state) for 16 seconds. Dashed vertical lines indicate reverse-engineered subarray boundaries: row addresses 0-1023 belong to Subarray 0, 1024-2047 to Subarray 1, and 2048-3071 to Subarray 2. The aggressor row (row 1536) is located in Subarray 1 (Aggressor Subarray), and Subarrays 0 and 2 are physically adjacent to the aggressor subarray (Neighboring Subarray).

Observation 4. ColumnDisturb induces bitflips across *three* consecutive subarrays (3072 rows), causing bitflips in many more rows than RowHammer and RowPress.

All 3072 rows in three consecutive subarrays experience ColumnDisturb bitflips. However, *only* the immediate neighbor rows of the aggressor row 1536 (i.e., rows 1535 and 1537) experience RowHammer & RowPress bitflips.⁹

This observation could have serious implications for the fu-

⁸We experiment with all 28 tested modules and observe very similar characteristics to this representative Samsung module, S0.

⁹We verify that bitflips in the immediate neighbors of an aggressor row (± 1 rows of the aggressor row) are caused by RowHammer and RowPress based on three observations. First, prior works on real DRAM chips [4, 135, 153, 154] demonstrate that by *correctly* reverse-engineering logical-to-physical row mapping of DRAM chips, RowHammer & RowPress induce bitflips in adjacent victim rows (± 1 rows of the aggressor row) [4, 135, 153, 154] due to the underlying silicon-level characteristics of RowHammer & RowPress [154–156]. Second, when we hammer different aggressor rows in the same subarray, we observe high bitflip counts only in ± 1 neighboring victim rows (i.e., 7559 RowHammer bitflips and 5406 RowPress bitflips on average across ± 1 and ± 1 neighboring victim rows), while all other victim rows in the aggressor subarray experience similar bitflip counts to each other (between 2353–3505 ColumnDisturb bitflips across all other victim rows, significantly less than RowHammer & RowPress bitflips. Third, when we initialize victim rows with an all-0 pattern, we observe only 0 to 1 bitflips in the ± 1 neighboring rows (not shown). Since ColumnDisturb induces only 1 to 0 bitflips (§4.3), whereas RowHammer & RowPress can induce both 0 to 1 and 1 to 0 bitflips (§4.3), this third observation further indicates that RowHammer & RowPress induce bitflips in only ± 1 rows.

ture, as refresh-based DRAM read disturbance mitigation techniques consider a few neighbor rows (e.g., up to ± 4 in industry solutions [41, 47, 147–149, 157]) of the aggressor row as victim rows, all in the same subarray, whereas ColumnDisturb leads to bitflips in *thousands* of rows (i.e., all rows across *three* consecutive subarrays).

We observe that subarrays that are neither the aggressor subarray nor physically-adjacent to the aggressor subarray do *not* experience any ColumnDisturb bitflips (not shown in the figure). This is due to the open-bitline DRAM architecture [92–100]: two neighboring subarrays share half of their columns. Thus, the aggressor subarray (Subarray 1) shares columns with both the above subarray (Subarray 0) and the below subarray (Subarray 2) (see Fig. 3). As a result, subarrays that do not share columns with the aggressor subarray are not affected (see Fig. 1c). We hypothesize that this observation shows ColumnDisturb is a column-based read disturbance phenomenon, since only the cells sharing the same columns as the aggressor row experience ColumnDisturb bitflips.

Observation 5. ColumnDisturb induces more bitflips in the aggressor subarray’s rows than in those of neighboring subarrays.

On average, ColumnDisturb induces $1.45\times$ more bitflips in the aggressor subarray’s rows than in the neighboring subarrays’ rows. We hypothesize that all columns are affected in the aggressor subarray because an activation causes perturbation in all bitlines in the same subarray. In contrast, only half of the columns in the neighboring subarray are connected to the aggressor subarray, and hence an activation causes a perturbation in *half* of the bitlines in the neighboring subarrays.

We also observe no overlap in the column indices of the cells that experience *only* ColumnDisturb bitflips in Subarray 0 and Subarray 2 (neighboring subarrays of Subarray 1). We hypothesize that this is because Subarray 1’s even columns are shared with Subarray 0’s odd columns (e.g., shared blue columns in the middle and top subarrays in Fig. 3) and Subarray 1’s odd columns are shared with Subarray 2’s even columns (e.g., red columns in the middle and bottom subarrays in Fig. 3). This observation further supports our hypothesis that ColumnDisturb is a column-based read disturbance phenomenon since *only* cells that share columns with the aggressor row are affected.

Observation 6. For a given refresh interval, ColumnDisturb induces many more bitflips than retention failures.

On average, ColumnDisturb induces 2942.68 bitflips per row in the aggressor subarray and 2025.02 bitflips per row in the neighboring subarrays, which are $7.07\times$ and $4.87\times$ more than retention failure bitflips for a refresh interval of 16 seconds, respectively.

Takeaway 2. ColumnDisturb is a column-based read disturbance phenomenon that disturbs cells via DRAM columns (i.e., bitlines) and induces bitflips in thousands of rows (i.e., as many as all 3072 rows in three DRAM subarrays).

4.3. Bitflip Direction

We analyze the bitflip direction of ColumnDisturb versus retention failures using one representative module (S0; Samsung 16Gb A-die).⁸ Fig. 7 shows the number of 1 to 0 (left) and 0 to 1 (right) bitflips in a subarray (y-axis) due to ColumnDisturb

and retention failure bitflips across five different refresh intervals (x-axis). Error bars represent the range of minimum and maximum number of bitflips across all tested subarrays.

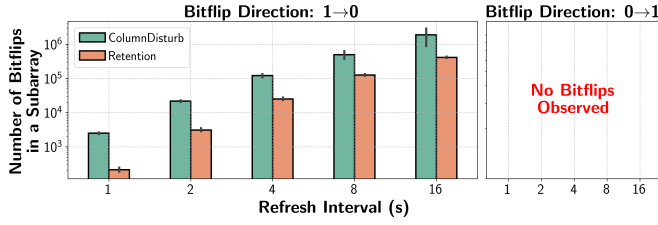


Figure 7: Distribution of the number of 1 to 0 and 0 to 1 bitflips in a subarray due to ColumnDisturb and retention failures.

Observation 7. ColumnDisturb induces *only* 1 to 0 bitflips, which are the same direction as retention failures.

Across all tested refresh intervals and subarrays, the *only* observed bitflip direction for ColumnDisturb is 1 to 0, which is the same as retention failures. We do *not* observe any 0 to 1 ColumnDisturb bitflips. This is in stark contrast to RowHammer and RowPress, which induce bitflips in both 1 to 0 and 0 to 1 directions [3, 4, 7, 135, 154], which we also observe in our tested DRAM chips.

Observation 8. ColumnDisturb induces many more bitflips than retention failures, especially at shorter refresh intervals.

On average, ColumnDisturb induces 11.77x, 7.02x, 4.86x, 3.97x, and 4.58x more bitflips than retention failures at refresh intervals of 1s, 2s, 4s, 8s, and 16s, respectively.

Takeaway 3. DRAM is much more vulnerable to ColumnDisturb than retention failures, and the bitflip directionalities of ColumnDisturb and RowHammer & RowPress are significantly different.

4.4. Data Pattern in Aggressor Row

We investigate how the aggressor data pattern (i.e., data pattern on the columns perturbed by the aggressor row) affects the ColumnDisturb vulnerability using a single representative module from each major DRAM manufacturer (S0, H0, and M6). We test *all* subarrays in a single bank from each tested module. For this experiment, we use all-1 as the data pattern for all victim cells in a subarray since ColumnDisturb induces only 1 to 0 bitflips and $t_{\text{AggOn}} = t_{\text{RAS}}$. We observe that not all subarrays have the same number of rows (the number of rows in a subarray across all tested modules ranges between 512 and 1024), and thus, we use a metric “fraction of cells with bitflips in a subarray.” Fig. 8 shows the fraction of cells that experience bitflips in a subarray across all tested subarrays (y-axis) for five different refresh intervals (x-axis) for three DRAM manufacturers (subplot). Each line represents a different experiment: 1) ColumnDisturb: AggDP=all-0, the aggressor row is initialized with all-0, 2) ColumnDisturb: AggDP=all-1, the aggressor row is initialized with all-1, and 3) RET, retention failures, where all columns are at the precharged voltage (VDD/2) during the entire experiment. The error band of a line shows the minimum and maximum values across all tested subarrays.

Observation 9. ColumnDisturb induces many more bitflips when the aggressor row data pattern is all-0 versus all-1.

For example, at 16s, all-0 aggressor pattern induces 1.15x,

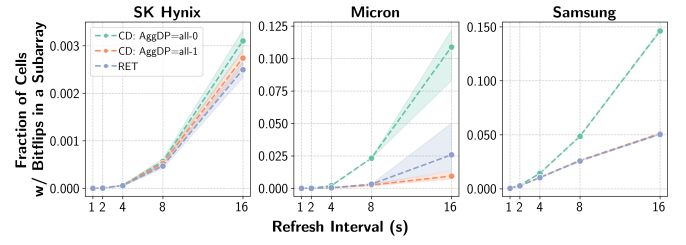


Figure 8: Distribution of fraction of cells with bitflips in a subarray due to ColumnDisturb for two different aggressor data patterns and retention failures.

11.52x, and 2.86x more bitflips than the all-1 aggressor data pattern for ColumnDisturb in SK Hynix, Micron, and Samsung chips, respectively. We hypothesize that this occurs due to a larger voltage level difference between the victim cell and the perturbed columns (i.e., columns connected to the aggressor row). This is because, when the aggressor data pattern is all-0, all perturbed columns in the subarray are at GND (i.e., low voltage) while the victim cells are at VDD, resulting in a VDD difference. In contrast, when the aggressor data pattern is all-1, both victim cells and all perturbed columns in a subarray are at VDD, resulting in no voltage difference. We further analyze the relationship between the perturbed column voltage level and ColumnDisturb in §4.6.

Observation 10. When the aggressor data pattern is all-1 (same as victim data pattern), ColumnDisturb can induce fewer bitflips than retention failures.

For example, in Micron, with a refresh interval of 16s, 2.73x fewer DRAM cells in a subarray on average experience bitflips with ColumnDisturb (when both the aggressor data pattern and the victim data pattern are all-1) versus retention failures.

We hypothesize that this could be due to the reduced voltage level difference between a column and the victim cell. In ColumnDisturb with all-1 aggressor and victim data pattern tests, the voltage difference between the victim cell (VDD) and the column (VDD) is zero. In contrast, in retention failure tests, the column is held at VDD/2 while the cell remains at VDD, resulting in a higher column-cell voltage difference (i.e., VDD/2) than ColumnDisturb with all-1 aggressor and victim data pattern tests. As a result, having a lower voltage difference between the victim cell and the column could lead to a lower ColumnDisturb vulnerability. We provide a detailed analysis and hypotheses for the effect of the voltage difference between victim cells and perturbed columns in §4.6.

Takeaway 4. Data pattern in the aggressor row has a significant effect on the number of observed ColumnDisturb bitflips.

4.5. Aggressor Row on Time

We study the effect of the amount of time an aggressor row is kept open (t_{AggOn}) on ColumnDisturb bitflips.¹⁰ For this analysis, we use all-1 (0xFF) as the victim data pattern and all-0 (0x00) for the aggressor data pattern since these are the worst-case data patterns that induce the most ColumnDisturb bitflips in a DRAM chip. Fig. 9 shows the fraction of cells with bitflips in a subarray observed in three representative modules, one from

¹⁰Keeping an aggressor row open for a long time (i.e., high t_{AggOn} values) perturbs *all* columns connected to the cells in the aggressor row, since the columns remain driven to VDD or GND.

each tested manufacturer. Each line represents a different experiment: 1) ColumnDisturb: $t_{\text{AggOn}}=36\text{ns}$, where we keep the aggressor row open for 36ns, 2) ColumnDisturb: $t_{\text{AggOn}}=70.2\mu\text{s}$, where we keep the aggressor row open for 70.2 μs , and 3) RET, retention failure, where the bank is idle (i.e., in precharged state) during the experiment.

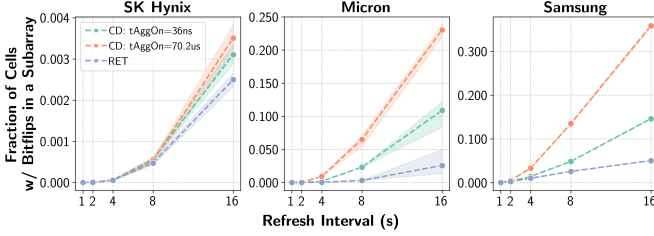


Figure 9: Distribution of fraction of cells with bitflips in a subarray due to ColumnDisturb for two different t_{AggOn} values and retention failures.

Observation 11. Fraction of cells with ColumnDisturb bitflips in a subarray significantly increases as t_{AggOn} increases.

For example, at a refresh interval of 16s, increasing t_{AggOn} from 36ns to 70.2 μs increases the number of ColumnDisturb bitflips by 1.20x, 2.12x, and 2.45x for SK Hynix, Micron, and Samsung modules, respectively. We hypothesize that this occurs because a longer t_{AggOn} keeps the perturbed columns at a low voltage level (GND) for an extended period, which likely results in an increased ColumnDisturb effect.

4.6. Average Voltage Level on Perturbed Columns

We observe that DRAM chips become significantly more vulnerable to ColumnDisturb 1) when the voltage level difference between the victim cell and the perturbed columns (i.e., columns perturbed by the aggressor row) is high (§4.4) and 2) when the perturbed columns are kept active longer (§4.5), i.e., the voltage level difference between the victim cell and the perturbed columns is kept high for an extended period. Based on our observations in §4.4 and §4.5, we hypothesize that the voltage level on the perturbed columns is a key parameter that affects the ColumnDisturb vulnerability in DRAM.

To understand the effect of average column voltage level on ColumnDisturb, we calculate the average voltage level on the perturbed columns $AVG(V_{\text{COL}})$ when we perform ColumnDisturb tests. Our access pattern, for every $t_{\text{AggOn}}+t_{\text{RP}}$ time, (§3.2) 1) keeps the column voltage level at the aggressor data pattern in the aggressor subarray, DP_{COL} , for t_{AggOn} and 2) keeps the column voltage level at precharged voltage ($V_{\text{DD}}/2$) for t_{RP} .

Based on the relationship between $AVG(V_{\text{COL}})$ and DP_{COL} , t_{AggOn} , $V_{\text{DD}}/2$, and t_{RP} parameters, we can calculate $AVG(V_{\text{COL}})$ as follows:

$$AVG(V_{\text{COL}}) = \frac{t_{\text{AggOn}} * DP_{\text{COL}} + V_{\text{DD}}/2 * t_{\text{RP}}}{t_{\text{AggOn}} + t_{\text{RP}}}$$

For example, assume $DP_{\text{COL}}=\text{GND}=0$, $t_{\text{AggOn}}=36\text{ns}$, precharged voltage level= $V_{\text{DD}}/2$, and $t_{\text{RP}}=14\text{ns}$ (i.e., perturbed columns are at GND for 36ns, and at $V_{\text{DD}}/2$ for 14ns in every 50ns), $AVG(V_{\text{COL}})$ becomes $\frac{36\text{ns} * 0 + V_{\text{DD}}/2 * 14\text{ns}}{36\text{ns} + 14\text{ns}} = 0.14 * V_{\text{DD}}$.

Fig. 10 shows the fraction of cells with bitflips (y-axis) as we sweep the average voltage level on the perturbed columns (x-axis) across five refresh intervals (each colored line). In this experiment, we test all subarrays in a single bank from three DRAM modules, one from each tested manufacturer.

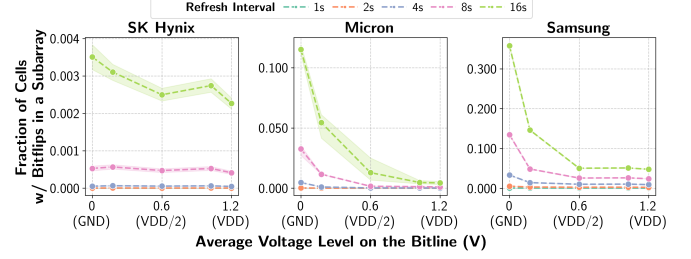


Figure 10: Fraction of cells with ColumnDisturb bitflips for different average column voltage levels.

Observation 12. A DRAM chip becomes significantly *more vulnerable* to ColumnDisturb as the average voltage level on the perturbed columns *decreases*.

We observe that reducing the average column voltage level from VDD to GND increases the fraction of cells that experience bitflips in a subarray by 1.65x, 26.31x, and 7.50x for SK Hynix, Micron, and Samsung, respectively, at a refresh interval of 16 seconds.

We provide two hypotheses to explain our observations. As the perturbed column voltage decreases, the voltage difference across the victim cell and the perturbed column increases, which 1) exacerbates subthreshold leakage of the access transistor [112] and/or 2) exacerbates the dielectric leakage between the victim capacitor and the perturbed column [2].¹¹

Key Hypothesis. ColumnDisturb exacerbates subthreshold leakage of the access transistor [112] and/or the dielectric leakage between the capacitor and the bitline [2].

Takeaway 5. The voltage level on the column plays an important role in ColumnDisturb’s device-level failure mechanisms. Device-level investigation is necessary to develop a better fundamental and first-principles understanding of ColumnDisturb.

4.7. Blast Radius

We study how many rows experience ColumnDisturb bitflips in a subarray for a given refresh interval. To do so, we use a metric: *the number of rows with bitflips in a subarray*, i.e., the number of rows that experience at least one bitflip in a subarray, broadly referred to as *blast radius* [3, 4, 7, 47, 50, 71, 125, 126, 136–139].

Fig. 11 shows the distribution of the number of rows with bitflips in a subarray at 65°C as a boxplot. Each subplot is dedicated to a different manufacturer and shows how many rows experience ColumnDisturb and retention failure bitflips in a subarray (y-axis) for five different refresh intervals.

Observation 13. ColumnDisturb induces bitflips in significantly more DRAM rows than retention failures.

For example, with a refresh interval of 1024ms, ColumnDisturb (Retention) induces bitflips in up to 52 (20), 353 (34), and 1022 (29) rows for SK Hynix, Micron, and Samsung modules, respectively. For SK Hynix, Micron, and Samsung modules, respectively, with a refresh interval of 512ms, ColumnDisturb induces bitflips in 2, 6, and 232 rows *on average*, whereas only two rows experience retention failures *at most*.

¹¹We call for future device-level and silicon-level studies to develop a better understanding of the inner workings of ColumnDisturb, just as device-level studies (e.g., [155, 156]) did for RowPress *after* the RowPress paper [4] demonstrated for the RowPress phenomenon experimentally on real DRAM chips.

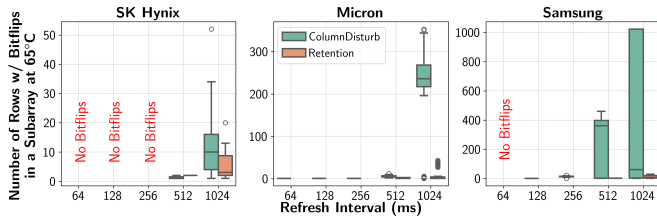


Figure 11: Distribution of the number of rows that experience ColumnDisturb versus retention failure bitflips for different refresh intervals.

Observation 14. Number of rows with ColumnDisturb bitflips significantly increases as the refresh interval increases.

For example, from 512ms to 1024ms, for SK Hynix, Micron, and Samsung, respectively, 9.73, 215.85, and 159.22 more rows in a subarray experience ColumnDisturb bitflips, on average across all tested subarrays. However, for retention failures, the increase is significantly smaller than ColumnDisturb. From 512ms to 1024ms, up to 8.39 (2.76 on average) more rows experience retention failure bitflips on average across all tested modules.

Takeaway 6. Significantly more rows are vulnerable to ColumnDisturb than retention failures.

4.8. ColumnDisturb on COTS HBM Chips

So far, we characterize commodity DDR4 chips to demonstrate and understand ColumnDisturb. In this section, we provide a preliminary study on the vulnerability of real HBM2 chips to ColumnDisturb. We test 4 HBM2 chips and analyze the number of ColumnDisturb bitflips and retention failures across all subarrays in a bank from three HBM2 chips. Fig. 12 shows the number of bitflips in a subarray across all subarrays from a single bank when 1) performing ColumnDisturb and 2) keeping the bank in the idle state without performing any refresh (i.e., retention), for 1, 2, and 4 seconds. Error bars represent the range of minimum and maximum number of bitflips across all tested subarrays.

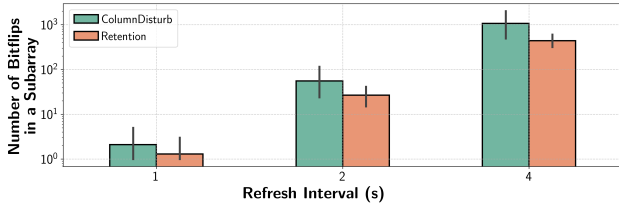


Figure 12: Number of ColumnDisturb and retention failure bitflips for different refresh intervals in HBM2 chips.

Observation 15. HBM2 chips are vulnerable to ColumnDisturb, and ColumnDisturb induces many more bitflips than retention failures.

We observe that across all four HBM2 chips tested, ColumnDisturb induces 1.61x, 2.08x, and 2.43x more bitflips than retention failures in 1s, 2s, and 4s, respectively. We expect our other observations for DDR4 chips (§4 and §5) will hold for HBM2 chips as well, since the DRAM array is the same for both, and ColumnDisturb happens at the array level.

Takeaway 7. ColumnDisturb is a widespread read disturbance phenomenon in DRAM chips: not only DDR4 chips, but also HBM2 chips are vulnerable to ColumnDisturb.

5. In-Depth ColumnDisturb Analysis

This section further enhances our analysis of ColumnDisturb by investigating parameters that have been shown to impact both read disturbance and retention failure mechanisms: temperature (§5.1), aggressor row on time (§5.2), access pattern (§5.3), data pattern (§5.4), the location of the aggressor row in a subarray (§5.5), and the effectiveness of Error Correcting Codes (ECC) against ColumnDisturb (§5.6).¹² We conduct our experiments by testing all subarrays in all banks from all tested modules using the parameters with which a DRAM chip experiences the highest ColumnDisturb vulnerability (i.e., aggressor data pattern=all-0, victim data pattern=all-1, $t_{\text{AggOn}}=70.2\mu\text{s}$, temperature= 85°C), unless stated otherwise.

5.1. Temperature

We analyze the relationship between temperature and the ColumnDisturb vulnerability by evaluating three metrics for four different temperature levels: 45°C, 65°C, 85°C, and 95°C.

Time to Induce the First ColumnDisturb Bitflip. Fig. 13 shows how the time to induce the first bitflip changes with temperature using a box and whiskers plot. Each subplot is dedicated to a different manufacturer.

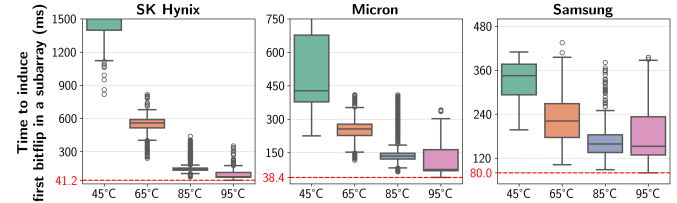


Figure 13: Distribution of time to induce the first ColumnDisturb bitflip at different temperatures.

Observation 16. Time to induce the first ColumnDisturb bitflip significantly reduces as temperature increases.

Across all tested modules, time to induce the first ColumnDisturb bitflip consistently reduces as temperature increases. For example, increasing temperature from 45°C to 95°C reduces the average time to induce the first ColumnDisturb bitflip in a subarray by 9.05x, 5.15x, and 1.96x for SK Hynix, Micron, and Samsung, respectively.

Fraction of Cells with Bitflips in a Subarray. Fig. 14 shows the fraction of cells with bitflips in a subarray (y-axis) due to ColumnDisturb versus retention failures (color-coded) across four temperature levels (x-axis) when the refresh interval is 512ms.

Observation 17. ColumnDisturb is more sensitive to temperature than retention failures.

For example, in SK Hynix chips, when the temperature increases from 85°C to 95°C, ColumnDisturb (retention failure) bitflips increase by 72.96x (3.68x) on average. We also observe that ColumnDisturb induces more bitflips than retention failures in all tested temperature levels across all DRAM modules. For example, at 65°C, ColumnDisturb induces 152.66x more bitflips than retention failures in Samsung modules.

¹²We observe that the overall trends are consistent across the three evaluated metrics: 1) time to induce the first ColumnDisturb bitflip in a subarray, 2) the fraction of cells with ColumnDisturb bitflips in a subarray, and 3) the blast radius. Due to space limitations, we provide the analysis of the first metric in all experiments and provide a detailed analysis of all metrics only for temperature. Our extended version provides comprehensive analyses of all metrics [127].

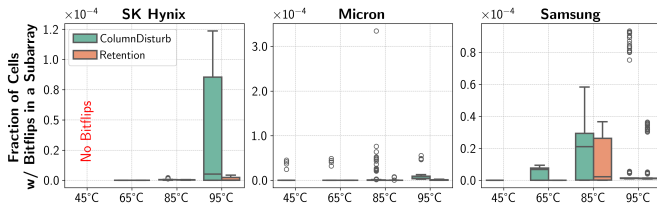


Figure 14: Distribution of fraction of cells with ColumnDisturb and retention failure bitflips in a subarray at different temperatures.

Blast Radius. Fig. 15 shows the blast radius effect (y-axis) of ColumnDisturb and retention failures (i.e., the number of rows in a subarray that experience at least one ColumnDisturb and retention failure bitflip) for three manufacturers (rows of subplots) and four temperature levels (columns of subplots). The x-axis shows the refresh interval in milliseconds (ms).

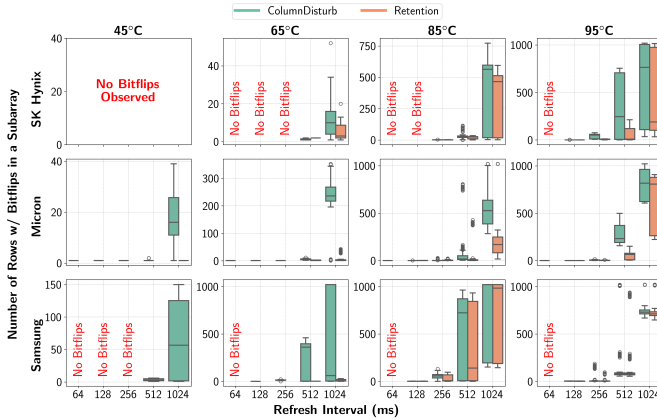


Figure 15: Number of rows that experience at least one bitflip (i.e., blast radius) due to ColumnDisturb and retention failures for different refresh intervals and temperature levels.

Observation 18. ColumnDisturb induces bitflips in many more rows than retention failures in all tested temperature levels.

For example, even at a low temperature level, 45°C, in Micron and Samsung, respectively, ColumnDisturb induces bitflips in up to 39 and 150 rows in a subarray across all tested subarrays. However, at most only a single row exhibits retention failures for Micron, and none for Samsung. We observe that across all tested temperature levels and refresh intervals, ColumnDisturb induces bitflips in up to 198x more DRAM rows than retention failures.

Observation 19. The blast radius of both ColumnDisturb and retention failures increase with higher temperature.

At 95°C, across all tested subarrays, both mechanisms nearly span an entire subarray (i.e., almost all rows of every tested subarray experience bitflips), whereas ColumnDisturb begins exhibiting a wide impact already at 65°C (e.g., at least 5.33% of subarrays in Samsung modules experience a blast radius of 1000).

Takeaway 8. As temperature increases, DRAM chips become more vulnerable to ColumnDisturb.

5.2. Aggressor Row On Time

Fig. 16 shows the distribution of time to induce the first ColumnDisturb bitflip in a subarray across all tested subarrays for

four different t_{AggOn} values: 36ns, 7.8 μ s, 70.2 μ s, and 1ms.

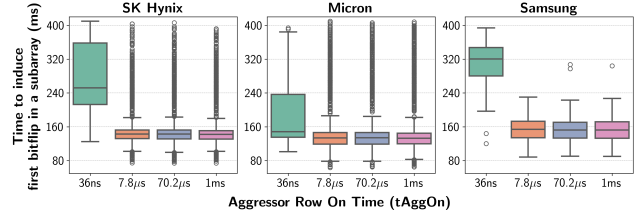


Figure 16: Time to induce first ColumnDisturb bitflip distribution for four different t_{AggOn} values.

Observation 20. Keeping a row open ($t_{\text{AggOn}} \gg t_{\text{RAS}}$) is significantly more effective than immediately closing it ($t_{\text{AggOn}} = t_{\text{RAS}}$) in inducing the first ColumnDisturb bitflip in a subarray.

Increasing t_{AggOn} from 36ns to 7.8 μ s reduces the average time to induce the first ColumnDisturb bitflip by 1.68x, 1.22x, and 2.03x in SK Hynix, Micron, and Samsung, respectively. We observe that when $t_{\text{AggOn}} \gg t_{\text{RAS}}$, the distributions are very similar across all tested t_{AggOn} values. We hypothesize that increasing t_{AggOn} decreases the average voltage level on DRAM columns, resulting in higher ColumnDisturb vulnerability in DRAM chips (i.e., less time to induce the first ColumnDisturb bitflip in a subarray), as discussed in §4.6.

5.3. Access Pattern

Until now, we perform ColumnDisturb with a single aggressor row, causing the column voltage to alternate between VDD/2 and GND or VDD, depending on the aggressor data pattern (described in §3.2). To understand the effect of toggling the column, we introduce a two-aggressor access pattern where two aggressor rows are used with complementary data patterns:

$$\text{ACT } R_{\text{Agg1}} \xrightarrow{t_{\text{AggOn}}} \text{PRE} \xrightarrow{t_{\text{RP}}} \text{ACT } R_{\text{Agg2}} \xrightarrow{t_{\text{AggOn}}} \dots$$

In the single aggressor access pattern (§3.2), we use all-0 as the aggressor data pattern. Thus, the column voltage transition is {GND→VDD/2→GND...}. In the two-aggressor access pattern, we use all-0 in the first aggressor (R_{Agg1}) and all-1 in the second aggressor (R_{Agg2}), and thus, the voltage transition on the column becomes {GND→VDD/2→VDD→VDD/2...}.

Fig. 17 shows the time to induce the first ColumnDisturb bitflip for the two access patterns. In both experiments, victim rows are initialized with all-1 as we observe only 1 to 0 bitflips for ColumnDisturb (§4.3).

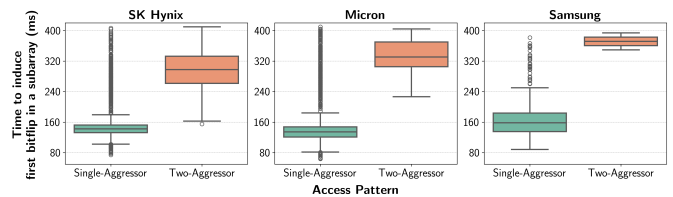


Figure 17: Distribution of time to induce the first ColumnDisturb bitflip for two different access patterns.

Observation 21. Single-aggressor access pattern induces the first ColumnDisturb bitflip in a subarray significantly faster than the two-aggressor access pattern.

For SK Hynix, Micron, and Samsung modules, the single-aggressor access pattern induces the first ColumnDisturb bitflip 1.83x, 1.92x, and 2.16x faster than the two-aggressor access pattern, respectively.

This observation shows that toggling a column through the sequence of GND→VDD/2→VDD (i.e., the two-aggressor access pattern) is less effective than toggling it only between VDD/2 and GND (i.e., the single-aggressor access pattern). We hypothesize that this is because the two-aggressor access pattern results in a higher average column voltage than the single-aggressor access pattern, resulting in lower ColumnDisturb vulnerability in DRAM chips, i.e., longer time to induce the first ColumnDisturb bitflip (§4.6).

Takeaway 9. Access pattern greatly affects a DRAM chip’s vulnerability to ColumnDisturb.

5.4. Data Pattern

We analyze the effect of the aggressor and victim data patterns using five aggressor and victim data pattern pairs. In this experiment, victim cells are initialized with the negated aggressor data pattern (e.g., if the aggressor data pattern is all-0, the victim data pattern is all-1). We omit the inverse of the tested patterns as either we 1) observe almost the same trend or 2) study it before (all-1 in §4.4).

Time to Induce the First ColumnDisturb Bitflip. Fig. 18 shows the distribution of time to induce the first ColumnDisturb bitflip in a subarray for five data patterns.

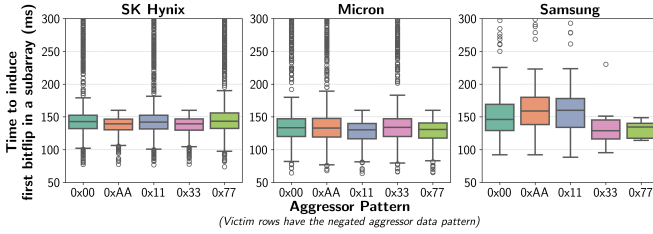


Figure 18: Distribution of time to induce the first ColumnDisturb bitflip for five different aggressor and victim data pattern pairs.

Observation 22. Data pattern has a small effect on the time to induce the first ColumnDisturb bitflip in a subarray.

Across all data patterns, the average time to induce the first ColumnDisturb bitflip varies by at most 1.31x. We hypothesize that this small variation (1.31x) suggests that bitline-to-bitline interference does not strongly affect the time to induce the first ColumnDisturb bitflip. This is because the weakest cell determines the time to induce the first ColumnDisturb bitflip. If the weakest cell is connected to a column written with logic-0, it flips at nearly the same time regardless of the values in neighboring columns/bitlines.

Total Number of Bitflips in a Subarray. Fig. 19 shows the total ColumnDisturb bitflips in a subarray with a refresh interval of 512ms. To maintain a reasonable experiment time, we test three different aggressor and victim data pattern pairs.

Observation 23. Data pattern greatly affects total ColumnDisturb bitflips in a subarray: having more logic-0 values in the perturbed columns generates more bitflips.

For example, in Samsung chips, an aggressor data pattern of 0x00 induces 2.04x more ColumnDisturb bitflips than an aggressor data pattern of 0xAA on average. We hypothesize that this is due to the bitflip direction characteristics of ColumnDisturb: only victim cells that are initialized with logic-1 experience bitflips (see §4.3). Thus, since victim cells are initialized with

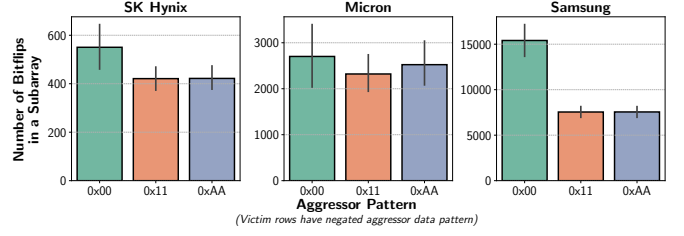


Figure 19: Total number of ColumnDisturb bitflips in a subarray for three different aggressor and victim data pattern pairs.

the negated aggressor data pattern, more logic-0 values in the aggressor pattern (which means more victim cells are initialized with logic-1) lead to more ColumnDisturb bitflips.

5.5. Location of the Aggressor Row in a Subarray

To understand the effects of spatial variation on ColumnDisturb, we analyze how the location of an aggressor row in a subarray affects the ColumnDisturb vulnerability. To do so, we test three aggressor row locations: 1) "Beginning": the first row of the subarray, 2) "Middle": the middle row in the subarray, and 3) "End": the last row of the subarray. Fig. 20 shows the distribution of time to induce the first bitflip in a subarray across DRAM subarrays (y-axis) based on the aggressor row’s location in a subarray (x-axis).

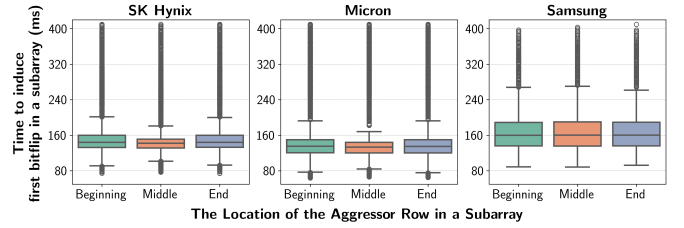


Figure 20: Distribution of time to induce the first ColumnDisturb bitflip based on the aggressor row’s location in a subarray.

Observation 24. The location of the aggressor row only slightly affects the time to induce the first ColumnDisturb bitflip.

We observe that across all tested manufacturers, there is at most 1.08x variation on average when the location of the aggressor row in a subarray changes.

5.6. Effectiveness of System-Level ECC

A system that uses Error-Correcting Codes (ECC) [134, 150, 158–167] can potentially protect against ColumnDisturb bitflips if those bitflips are distributed across the DRAM chip such that no ECC codeword contains more bitflips than ECC can correct. Fig. 21 shows the distribution of ColumnDisturb bitflips across 8-byte data chunks for all tested DRAM modules for 512ms and 1024ms refresh intervals. We use 8-byte data chunks as DRAM ECC typically uses 8-byte or larger datawords [134, 150, 151, 163, 165, 168–172].

Observation 25. ColumnDisturb induces more bitflips than today’s widely-used ECCs can correct or even detect.

We observe that many 8-byte data chunks experience 3 (up to 15) ColumnDisturb bitflips, which the SECDED ECC [163, 168–171] cannot correct or detect, in Micron and Samsung chips.

Observation 26. Relying *only* on ECC to mitigate all ColumnDisturb-induced errors is likely very costly.

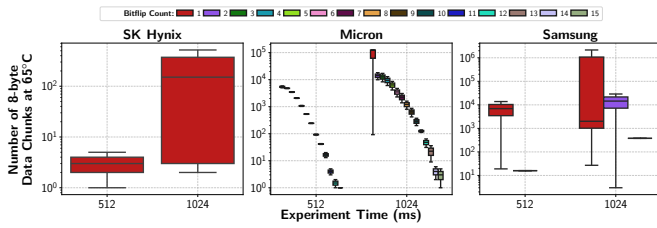


Figure 21: Distribution of 8-byte data chunks with different ColumnDisturb bitflip counts.

Since ColumnDisturb can induce 15 bitflips in multiple 8-byte data chunks, a (7,4) Hamming code [160] could correct such bitflips with very large DRAM storage overheads (75%, i.e., three parity bits for every four data bits). Thus, relying on ECC alone to prevent ColumnDisturb bitflips is likely a very expensive solution.

Effectiveness of On-Die ECC. We evaluate the effectiveness of a (136,128) single-error-correcting (SEC) Hamming code [160] (possibly implemented in DDR5 chips today [134, 151, 165, 173, 174] for area efficiency and low cost overhead).¹³ This code corrects any one bitflip in a 136-bit codeword. The observed bitflip distributions (Fig. 21) for Micron and Samsung already exceed the error correction capabilities of such codes. Moreover, the SEC code can miscorrect codewords that have more than one bitflip [173, 174, 176]. A miscorrection manifests as additional bitflip(s) in the codeword (i.e., the SEC code induces another bitflip by attempting to correct the erroneous codeword). We randomly generate 10K erroneous codewords with two bitflips (bitflip indices in the codeword are determined using a uniform random number generator). The evaluated SEC code miscorrects 88.5% of the codewords, transforming a codeword with two bitflips into another with three bitflips.

Observation 27. Low-cost on-DRAM-die single-error-correcting codes *cannot* correct all ColumnDisturb bitflips and are likely to induce additional bitflips in an erroneous codeword that has as few as only two bitflips.

Takeaway 10. Conventional DRAM ECC cannot protect against ColumnDisturb bitflips, and an ECC scheme that can protect against ColumnDisturb requires large overheads.

6. Implications

Our findings have strong implications for both 1) the robustness of *future* systems and 2) the system speedup and DRAM energy saving benefits of *existing* retention-aware heterogeneous refresh mechanisms (e.g., [75–88]). ColumnDisturb could jeopardize the safety, security, and reliability of future computing systems as more ColumnDisturb bitflips may manifest in the standard refresh window due to continuously shrinking DRAM technology node size. §6.1 describes and evaluates two techniques that could mitigate ColumnDisturb bitflips at varying performance, energy, and area overheads. In light of our new findings, we evaluate a retention-aware heterogeneous refresh mechanism (RAIDR [76]) and show that its benefits over the baseline DRAM periodic refresh dramatically decrease (e.g., by 53%, see §6.2) because at long refresh windows a large majority of DRAM rows (see §4.7 and §5.1) exhibit ColumnDisturb bitflips.

¹³The exact designs of on-die ECC are kept strictly confidential by DRAM manufacturers [150, 175] and may be different from what we evaluate.

6.1. Implications for System Robustness

Observations 1, 2, 4, 13, 17 and 19 demonstrate that ColumnDisturb induces bitflips in victim rows that are thousands of DRAM rows away from, and even in a different subarray from the aggressor row. For example, the first bitflip we observe in 63.6ms is 374 rows away from the aggressor row. Unfortunately, current RowHammer mitigations only refresh up to 8 neighboring rows [6, 7, 29, 35, 41, 47, 126, 137, 138, 149, 157, 177–237] as RowHammer or RowPress bitflips induce bitflips in immediate neighbors of the aggressor rows [3, 4, 7, 71, 124–126, 135, 136, 138, 139, 153, 154, 238–246]. Simply modifying commercial or many other proposed read disturbance mitigations to preventively refresh all potential victim rows (e.g., 3072 such victims) of ColumnDisturb would induce prohibitive system performance and energy overheads due to the latency and energy cost of performing thousands of preventive refresh operations. For example, according to the latest DDR5 JEDEC Standard [148], refreshing ± 4 physically adjacent neighboring rows incurs 2x more latency ($t_{DRFMAb} = 560\text{ns}$) than refreshing ± 2 ($t_{DRFMAb} = 280\text{ns}$). Refreshing all rows in three consecutive subarrays (e.g., 3072 rows) would take an estimated, prohibitive, $\sim 215\mu\text{s}$ latency.¹⁴ Moreover, as DRAM chip density increases, so does the number of rows in a subarray. Therefore, securely mitigating ColumnDisturb bitflips would likely require refreshing a few thousand rows before an aggressor row is hammered or pressed to the point of failure. However, *reactively* refreshing that many rows incurs prohibitive memory access latencies. We describe and evaluate two solutions for ColumnDisturb that *proactively* refresh potential victim rows of ColumnDisturb without inducing prohibitive memory access latencies.

Increasing DRAM Refresh Rate. A straightforward (and high-overhead) solution for ColumnDisturb is to indiscriminately increase the DRAM refresh rate. DRAM periodic refresh would mitigate ColumnDisturb-induced bitflips if the refresh period is set adequately short. The benefit of this solution is that it can be applied immediately in today’s systems. However, as many prior works [75–88] show, increasing the refresh rate degrades system performance and energy efficiency because more frequent refresh operations 1) increase memory access latency, 2) reduce row buffer hit rate, 3) degrade bank-level parallelism, and 4) cause more activity on the memory bus and in DRAM banks. For example, for a 32 Gb DDR5 chip [147, 247], reducing the default (all bank) refresh period (REFab) from 32 ms to 8 ms (i.e., 4x higher refresh rate) increases the DRAM throughput loss due to periodic refresh operations from 10.5% to 42.1%¹⁵ and relative energy consumed by refresh operations for an otherwise idle DRAM chip increases from 25.1% to 67.5%.¹⁶

We believe that indiscriminately increasing the DRAM re-

¹⁴Assuming 1) refreshing eight rows takes 560 ns [148], which means 70 ns to refresh one row and 2) rows across three subarrays are refreshed sequentially. In this case, refreshing 3072 rows requires $70 \times 3072 = 215\mu\text{s}$.

¹⁵DDR5 all bank refresh command latency (t_{RFC}) is 410 ns for 32 Gb chip density [147]. A DRAM chip *cannot* serve any memory request (read or write) for t_{RFC} after every refresh command, causing throughput loss. The memory controller issues refresh commands more frequently as refresh period reduces: once every 3.9 μs and 975 ns respectively for a refresh period of 32 ms and 8 ms.

¹⁶We estimate refresh and idle energy from manufacturer provided DRAM IDD current values [247].

fresh rate to mitigate ColumnDisturb-induced bitflips is *not* a performance- and energy-efficient solution. We describe a more intelligent mechanism that selectively and timely refreshes *only* the victim rows that are disturbed by ColumnDisturb.

Proactively Refreshing ColumnDisturb Victim Rows (PRVR). We propose proactively refreshing ColumnDisturb victim rows to mitigate ColumnDisturb bitflips. The key idea of PRVR is to refresh *only* the victim DRAM rows in three subarrays: the aggressor DRAM row’s subarray and its two neighboring subarrays. PRVR refreshes *each* of the N victim DRAM rows across three subarrays (e.g., for subarray size = 1024, $N = 3072$) *once* before the aggressor row is hammered or pressed enough times to induce a bitflip. PRVR distributes the refresh operations targeting these N rows over the time it takes to induce the first bitflip in a subarray using ColumnDisturb, similar to how a periodic refresh command refreshes a subset of all DRAM rows in a DRAM chip.

PRVR is a higher-performance and more energy-efficient solution than using a shorter DRAM refresh period. Assuming a default refresh period of 32 ms, a time to induce the first ColumnDisturb bitflip of 8 ms, and $N = 3072$, we analytically estimate PRVR’s DRAM throughput and energy benefits over using a fixed, 8 ms refresh period. PRVR reduces 8 ms refresh period’s throughput loss by 70.5% and refresh energy consumption by 73.8% assuming one DRAM row from each DRAM bank is continuously hammered to induce ColumnDisturb bitflips in a 32 Gb DDR5 chip.¹⁷ We conclude that proactively refreshing ColumnDisturb victim rows could prevent ColumnDisturb bitflips at much smaller performance and energy overheads than simply reducing the DRAM refresh period.

6.2. Implications on Retention-Aware Refresh

Retention-aware heterogeneous refresh mechanisms [75–88] leverage the heterogeneity in DRAM cell data retention time to reduce the number of DRAM row refresh operations while maintaining data integrity. These mechanisms typically classify a DRAM row as *weak* if any of its cells exhibit a retention failure during a relatively short refresh window (e.g., 64 ms). In contrast, a DRAM row that reliably retains data over a much longer time window (e.g., 1024 ms) is considered *strong*. Consequently, the greater the proportion of strong rows in a DRAM chip, the higher the performance and energy reduction provided by a retention-aware heterogeneous refresh mechanism. ColumnDisturb greatly reduces the benefits of retention-aware heterogeneous refresh mechanisms because ColumnDisturb renders many more rows weak than retention failures alone at a given refresh window, as observations 6, 18, and 19 demonstrate.

Fig. 22 shows the number of DRAM row refresh operations (normalized to the number of refresh operations performed by DDR4 64 ms periodic refresh) as the proportion of weak rows in a chip varies. We plot four lines, one each for four strong row retention time values: 128 ms, 256 ms, 512 ms, and 1024 ms. A circle, diamond, and a square on a line show the number of refresh operations needed for empirically observed average proportion of retention-weak rows across all tested modules, average proportion of ColumnDisturb-weak rows across all tested

modules, and the maximum proportion of ColumnDisturb-weak rows across all tested modules, respectively. A smaller y-axis value indicates smaller expected retention-aware heterogeneous refresh mechanism speedup and energy benefits.

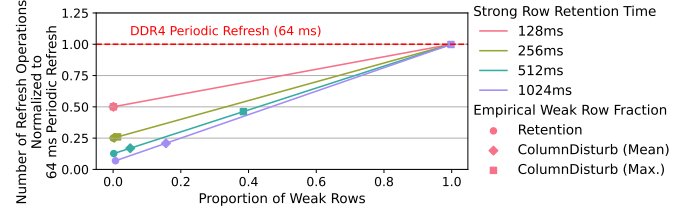


Figure 22: Number of DRAM row refresh operations needed. We plot one line each for four strong row retention times of 128, 256, 512, and 1024 milliseconds.

We make two key observations. First, a relatively large strong row retention time can significantly increase the expected speedup and energy benefits of retention-aware refresh mechanisms. For example, for the empirically observed average retention-weak row proportion (circles), number of refresh operations reduces by 43.1% when a strong row retention time of 1024 ms is used instead of a strong row retention time of 128 ms. Second, ColumnDisturb significantly reduces the expected speedup and energy benefits of retention-aware heterogeneous refresh mechanisms, especially at high strong row retention times where the empirical proportion of ColumnDisturb-weak rows greatly exceeds that of retention-weak rows. For example, in the presence of ColumnDisturb, for a strong row retention time of 1024 ms, the number of refresh operations increases by 3.02× (purple diamond in Fig. 22) on average and by up to 14.43× (purple square in Fig. 22), compared to a baseline case without ColumnDisturb (i.e., only retention failures are present) across all tested modules.

Takeaway 11. ColumnDisturb significantly degrades the expected speedup and energy saving benefits of retention-aware heterogeneous refresh mechanisms.

To quantitatively demonstrate the speedup and energy benefits of retention-aware heterogeneous refresh mechanisms, we evaluate a state-of-the-art retention-aware refresh mechanism, RAIDR [76], building on the Self-Managing DRAM (SMD) framework [248, 249].¹⁸ We evaluate two variants of RAIDR where 1) we store weak row addresses in a space-efficient (low-area-overhead) Bloom Filter [254–256] sized 8Kb with 6 hash functions and 2) we store weak row addresses in a space-inefficient (high-area-overhead) bitmap¹⁹ indexed using the row address (1 bit per DRAM row, 2 Mb for a 16 GiB DDR4 module). We refresh a weak row every 64 ms and a strong row every 1024 ms. We evaluate a hypothetical DRAM system configuration called *No Refresh* where we do *not* issue any refresh operations to show the headroom for system speedup and energy

¹⁸We use Ramulator [250–253] and simulate a modern computing system (configured as the baseline system described in Table 1 of [248]). We use 20 highly-memory-intensive multi-programmed workload mixes, each comprised of four single-core workloads each with last-level cache misses-per-kilo-instructions ≥ 10 . We warm-up the caches by fast-forwarding 100 million (M) instructions. We run each multi-core simulation until each core executes at least 500M instructions.

¹⁹Storing weak row addresses in a bitmap allows the retention-aware heterogeneous refresh mechanism to identify all weak rows accurately [86]. Compared to the Bloom Filter, using a bitmap has higher weak row identification accuracy but incurs higher storage overheads.

¹⁷System integration and rigorous evaluation of PRVR (e.g., using worst-case ColumnDisturb access patterns) is out of this paper’s scope. We expect future work to build on the key idea of PRVR and enable new efficient and effective ColumnDisturb mitigation mechanisms.

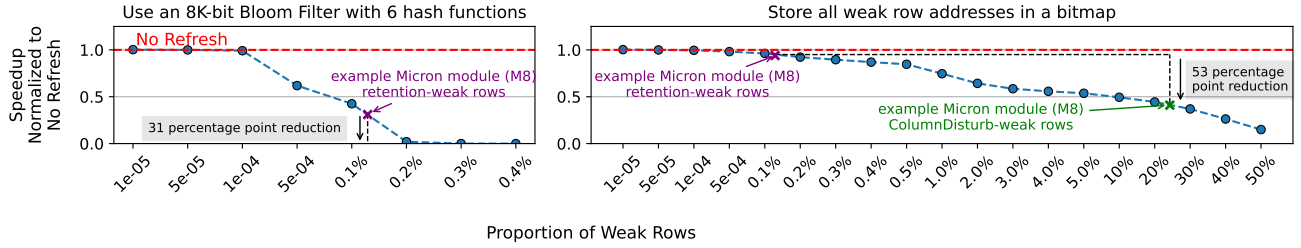


Figure 23: Speedup provided by RAIDR [76] over the proportion of weak rows in a DRAM module. Left and right subplots shows speedup for a low-area-overhead and high-area-overhead implementation, respectively.

consumption improvements of retention-aware heterogeneous refresh mechanisms.

Fig. 23 shows the system speedup provided by retention-aware refresh (in weighted speedup) on average across all 20 four-core workloads normalized to the speedup of No Refresh. The left and right subplots, respectively, show the performance of the RAIDR variant that uses a Bloom Filter and the RAIDR variant that stores each weak row’s address in a bitmap. The x-axis shows the evaluated proportion of weak row values in the simulated DRAM system. We annotate the speedup of retention-aware refresh for empirically observed proportion of 1) retention-weak rows (purple $\times$) and 2) ColumnDisturb-weak rows (green $\times$) in one Micron DDR4 module.

We make two key observations. First, ColumnDisturb can completely negate the performance and energy (not shown in the figure) benefits of space-efficient (using a Bloom Filter) implementations of retention-aware heterogeneous refresh mechanisms. From Fig. 23 (left) we observe that as the proportion of weak rows increases from $1.00\text{e-}04$ to 0.20% (by $20\times$), the speedup and energy benefits reduce to a point where they are almost completely eliminated (≈ 99 percentage point speedup and energy benefit reduction). The $20\times$ increase in the proportion of weak rows is well within our empirical observations: across all tested modules, the proportion of weak rows increased by up to $198\times$ (Fig. 15).

As a concrete example, for the annotated Micron module (purple $\times$), the speedup and energy consumption benefits reduce by 31 percentage points and 32 percentage points (not shown), respectively, as the proportion of weak rows increases to accommodate ColumnDisturb-weak rows. Second, ColumnDisturb can greatly reduce the speedup and energy benefits even of space-inefficient (using a bitmap) implementations of retention-aware heterogeneous refresh mechanisms. From Fig. 23 (right), we observe that for the annotated Micron DDR4 module, the speedup and energy consumption benefits reduce by 53 and 50 percentage points (not shown), respectively.

Takeaway 12. ColumnDisturb can completely negate the performance and energy benefits of low-area-overhead retention-aware heterogeneous refresh mechanisms. ColumnDisturb greatly reduces the benefits of high-area-overhead retention-aware heterogeneous refresh mechanisms.

We conclude that the proportion of strong DRAM rows (that can endure high retention times without a bitflip) for a given refresh window significantly reduces with ColumnDisturb. This reduction greatly hinders (and can completely eliminate) the performance and energy benefits of retention-aware heterogeneous refresh mechanisms.

7. Related Work

To our knowledge, this is the first work to experimentally demonstrate and characterize ColumnDisturb, a column-based read-disturb phenomenon in real DRAM chips.

DRAM Read Disturbance Characterization. Many works [3, 4, 7, 71, 124–126, 135, 136, 138, 139, 153, 154, 238–246] experimentally demonstrate how a real DRAM chip’s read disturbance vulnerability varies with 1) DRAM refresh rate [7, 41, 47], 2) the distance between aggressor and victim rows [3, 7, 139], 3) DRAM generation and technology node [3, 7, 47, 71], 4) temperature [71, 240], 5) time the aggressor row stays active [4, 8, 71, 102, 135, 153, 154, 240, 244, 246], 6) location of the victim cell [8, 71, 138, 244], 7) wordline voltage [136], 8) supply voltage [245], 9) reduced DRAM timing parameters [124, 126], and 10) time [125]. None of these works analyze or demonstrate ColumnDisturb, a column-based read disturbance phenomenon.

DRAM Retention. Many works experimentally study DRAM data retention [84, 85, 104, 117, 132, 159, 162, 166, 224, 257, 258]. Several works [75–88] present various error profiling algorithms to identify data retention failures when reducing the refresh rate and propose retention-aware heterogeneous refresh mechanisms to alleviate refresh overheads. However, we show that ColumnDisturb can significantly amplify the bitflips across subarrays and can cause significant performance and area overheads to retention-aware heterogeneous refresh mechanisms.

Bitline Disturbance in 4F^2 VCT DRAM. Emerging high-density 4F^2 DRAM architectures with Vertical Channel Transistors (VCT) are known to be vulnerable to disturbances from the bitline with a hammering-like access pattern [259–263]. However, since the device architecture of 4F^2 VCT DRAM is completely different from the contemporary COTS 6F^2 DRAM that we characterize in this work, we believe the error mechanism they study (i.e., floating body effect [261, 262]) is different from the one(s) that causes ColumnDisturb (see §4 for our analyses and hypotheses).

8. Conclusion

We present the first experimental demonstration and analysis of a new read disturbance phenomenon, ColumnDisturb, in modern DRAM chips: hammering or pressing an aggressor row disturbs DRAM cells through a DRAM column and induces bitflips in DRAM cells sharing the same columns as the aggressor row (across as many as three DRAM subarrays). Our experimental characterization of 216 real DDR4 and 4 HBM2 DRAM chips reveals that 1) ColumnDisturb is fundamentally different from RowHammer & RowPress, 2) ColumnDisturb worsens with DRAM technology scaling, 3) introduces bitflips within the nominal refresh window under nominal operating conditions in some real DRAM chips, and 4) DRAM is significantly

more vulnerable to ColumnDisturb than to retention failures. We describe and evaluate two ColumnDisturb mitigation techniques. We hope and believe that our detailed demonstration and analyses will inspire future works on better understanding ColumnDisturb at the device-level and mitigating it before it becomes a bigger vulnerability in future DRAM chips.

Acknowledgments

We thank the anonymous reviewers of MICRO 2025 for feedback. We thank the SAFARI Research Group members (especially Konstantinos Sgouras and Harsh Songara) for constructive feedback and the stimulating intellectual environment. We acknowledge the generous gift funding provided by our industrial partners (especially Google, Huawei, Intel, Microsoft), which has been instrumental in enabling the research we have been conducting on read disturbance in DRAM in particular and memory systems in general [101]. This work was in part supported by a Google Security and Privacy Research Award and the Microsoft Swiss Joint Research Center.

References

- [1] Robert H. Dennard, Field-Effect Transistor Memory, 1968. U.S. Patent 3,387,286.
- [2] Yexiao Yu, Zhongming Liu, Jingsi Cui, Zhong Kong, GuoBao Xiong, and Hong Ma. The Study of Bit Line to Storage Node Contact Leakage in Advanced DRAM. In *ICET*, 2022.
- [3] Jeremie S. Kim, Minesh Patel, Abdullah Giray Yağlıkçı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques. In *ISCA*, 2020.
- [4] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In *ISCA*, 2023.
- [5] Onur Mutlu and Jeremie S Kim. RowHammer: A Retrospective. *TCAD*, 2019.
- [6] Onur Mutlu, Ataberk Olgun, and A. Giray Yağlıkçı. Fundamentally Understanding and Solving RowHammer. In *ASP-DAC*, 2023.
- [7] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In *ISCA*, 2014.
- [8] Ataberk Olgun, Majd Osseiran, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. Read Disturbance in High Bandwidth Memory: A Detailed Experimental Study on HBM2 DRAM Chips. In *DSN*, 2024.
- [9] Onur Mutlu. The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser. In *DATE*, 2017.
- [10] Apostolos P Fournaris, Lidia Pocero Fraile, and Odysseas Koufopavlou. Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: A Survey of Potent Microarchitectural Attacks. *Electronics*, 2017.
- [11] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. Attacking Deterministic Signature Schemes using Fault Attacks. In *EuroS&P*, 2018.
- [12] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer Attacks Over the Network and Defenses. In *USENIX ATC*, 2018.
- [13] Sebastien Carre, Matthieu Desjardins, Adrien Facon, and Sylvain Guille. OpenSSL Bellcore's Protection Helps Fault Attack. In *DSN*, 2018.
- [14] Alessandro Barenghi, Luca Breveglieri, Niccolò Izzo, and Gerardo Pelosi. Software-Only Reverse Engineering of Physical DRAM Mappings for Rowhammer Attacks. In *IVSW*, 2018.
- [15] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon Koutsoukos, and Gabor Karsai. Triggering Rowhammer Hardware Faults on ARM: A Revisit. In *ASHES*, 2018.
- [16] Sarani Bhattacharya and Debdeep Mukhopadhyay. Advanced Fault Attacks in Software: Exploiting the Rowhammer Bug. In *Fault Tolerant Architectures for Cryptography and Hardware Security*, 2018.
- [17] Mark Seaborn and Thomas Dullien. Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. <http://googleprojectzero.blogspot.com.tr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html>, 2015.
- [18] SAFARI Research Group. RowHammer — GitHub Repository. <https://github.com/CMU-SAFARI/rowhammer>, 2014.
- [19] Mark Seaborn and Thomas Dullien. Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. *Black Hat*, 2015.
- [20] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *CCS*, 2016.
- [21] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in Javascript. arXiv:1507.06955 [cs.CR], 2016.
- [22] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In *USENIX Security*, 2016.
- [23] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *USENIX Security*, 2016.
- [24] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In *USENIX Security*, 2016.
- [25] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup Est Machina: Memory Deduplication as An Advanced Exploitation Vector. In *S&P*, 2016.
- [26] Sarani Bhattacharya and Debdeep Mukhopadhyay. Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis. In *CHES*, 2016.
- [27] Wayne Burleson, Onur Mutlu, and Mohit Tiwari. Invited: Who is the Major Threat to Tomorrow's Security? You, the Hardware Designer. In *DAC*, 2016.
- [28] Rui Qiao and Mark Seaborn. A New Approach for RowHammer Attacks. In *HOST*, 2016.
- [29] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. Can't Touch This: Software-Only Mitigation Against Rowhammer Attacks Targeting Kernel Memory. In *USENIX Security*, 2017.
- [30] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *SOSP*, 2017.
- [31] Misiker Tadesse Aga, Zelalem Birhanu Aweke, and Todd Austin. When Good Protections Go Bad: Exploiting Anti-DoS Measures to Accelerate Rowhammer Attacks. In *HOST*, 2017.
- [32] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer. In *RAID*, 2018.
- [33] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoecl, and Yuval Yarom. Another Flip in the Wall of Rowhammer Defenses. In *S&P*, 2018.
- [34] Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, Lukas Raab, and Lukas Lamster. Nethammer: Inducing Rowhammer Faults Through Network Requests. arXiv:1805.04956 [cs.CR], 2018.
- [35] Victor van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM. In *DIMVA*, 2018.
- [36] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU. In *S&P*, 2018.
- [37] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In *S&P*, 2019.
- [38] Sangwoo Ji, Youngjoo Ko, Saeyoung Oh, and Jong Kim. Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks. In *ASIACCS*, 2019.
- [39] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks. In *USENIX Security*, 2019.
- [40] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. RAMBleed: Reading Bits in Memory Without Accessing Them. In *S&P*, 2020.
- [41] 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. In *S&P*, 2020.
- [42] Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu. Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers. In *S&P*, 2020.
- [43] Zane Weissman, Thore Tiemann, Daniel Moghimi, Evan Custodio, Thomas Eisenbarth, and Berk Sunar. JackHammer: Efficient Rowhammer on Heterogeneous FPGA-CPU Platforms. arXiv:1912.11523 [cs.CR], 2020.
- [44] Zhi Zhang, Yueqiang Cheng, Dongxi Liu, Surya Nepal, Zhi Wang, and Yuval Yarom. PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses. In *MICRO*, 2020.
- [45] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. Deephammer: Depleting the Intelligence of Deep Neural Networks Through Targeted Chain of Bit Flips. In *USENIX Security*, 2020.
- [46] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized Many-Sided Rowhammer Attacks from JavaScript. In *USENIX Security*, 2021.
- [47] Hasan Hassan, Yahya Can Tuğrul, Jeremie S. Kim, Victor van der Veen, Kaveh Razavi, and Onur Mutlu. Uncovering in-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications. In *MICRO*, 2021.
- [48] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable Rowhammering in the Frequency Domain. In *S&P*, 2022.
- [49] M Caner Tol, Saad Islam, Berk Sunar, and Ziming Zhang. Toward Realistic Backdoor Injection Attacks on DNNs using RowHammer. arXiv:2110.07683, 2022.
- [50] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering From the Next Row Over. In *USENIX Security*, 2022.
- [51] Lois Orosa, Ulrich Rührmair, A. Giray Yağlıkçı, Haocong Luo, Ataberk Olgun, Patrick Jattke, Minesh Patel, Jeremie Kim, Kaveh Razavi, and Onur Mutlu. Spy-Hammer: Using RowHammer to Remotely Spy on Temperature. arXiv:2210.04084, 2022.
- [52] Zhi Zhang, Wei He, Yueqiang Cheng, Wenhao Wang, Yansong Gao, Dongxi Liu, Kang Li, Surya Nepal, Anmin Fu, and Yi Zou. Implicit Hammer: Cross-Privilege-Boundary Rowhammer through Implicit Accesses. *TDSC*, 2022.
- [53] Liang Liu, Yanan Guo, Yueqiang Cheng, Youtao Zhang, and Jun Yang. Generating Robust DNN with Resistance to Bit-Flip based Adversarial Weight Attack. *IEEE TC*, 2022.
- [54] Yaakov Cohen, Kevin Sam Tharayil, Arie Haenel, Daniel Genkin, Angelos D Keromytis, Yossi Oren, and Yuval Yarom. HammerScope: Observing DRAM Power Consumption Using Rowhammer. In *CCS*, 2022.
- [55] Mengxin Zheng, Qian Lou, and Lei Jiang. TrojViT: Trojan Insertion in Vision Transformers. arXiv:2208.13049, 2022.

- [56] Michael Fahr Jr, Hunter Kippen, Andrew Kwong, Thinh Dang, Jacob Lichtinger, Dana Dachman-Soled, Daniel Genkin, Alexander Nelson, Ray Perlner, Arkady Yerukhimovich, et al. When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer. *CCS*, 2022.
- [57] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G. Shin. SpeechHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In *S&P*, 2022.
- [58] Adnan Siraj Rakin, Md Hafizul Islam Chowdhury, Fan Yao, and Deliang Fan. DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories. In *S&P*, 2022.
- [59] Hakan Aydin and Ahmet Sertbaş. Cyber Security in Industrial Control Systems (ICS): A Survey of RowHammer Vulnerability. *Applied Computer Science*, 2023.
- [60] Koksai Mus, Yarkin Doröz, M Caner Tol, Kristi Rahman, and Berk Sunar. Jolt: Recovering TLS Signing Keys via Rowhammer Faults. *Cryptology ePrint Archive*, 2022.
- [61] Jianxin Wang, Hongke Xu, Chaoen Xiao, Lei Zhang, and Yuzheng Zheng. Research and Implementation of Rowhammer Attack Method based on Domestic NeoKylín Operating System. In *ICFTIC*, 2022.
- [62] Sam Lefforge. Reverse Engineering Post-Quantum Cryptography Schemes to Find Rowhammer Exploits. Master's thesis, University of Arkansas, 2023.
- [63] Michael J Fahr. *The Effects of Side-Channel Attacks on Post-Quantum Cryptography: Influencing FrodoKEM Key Generation Using the Rowhammer Exploit*. PhD thesis, University of Arkansas, 2022.
- [64] Anandpreet Kaur, Pravin Srivastav, and Bibhas Ghoshal. Work-in-Progress: DRAM-MAUT: DRAM Address Mapping Unveiling Tool for ARM Devices. In *CASES*, 2022.
- [65] Kunbei Cai, Zhenkai Zhang, and Fan Yao. On the Feasibility of Training-time Trojan Attacks through Hardware-based Faults in Memory. In *HOST*, 2022.
- [66] Dawei Li, Di Liu, Yangkun Ren, Ziyi Wang, Yu Sun, Zhenyu Guan, Qianhong Wu, and Jianwei Liu. CyberRadar: A PUF-based Detecting and Mapping Framework for Physical Devices. arXiv:2201.07597, 2022.
- [67] Arman Roohi and Shaahin Angizi. Efficient Targeted Bit-Flip Attack Against the Local Binary Pattern Network. In *HOST*, 2022.
- [68] Felix Staudigl, Hazem Al Indari, Daniel Schön, Dominik Sisejkovic, Farhad Merchant, Jan Moritz Joseph, Vikas Rana, Stephan Menzel, and Rainer Leupers. NeuroHammer: Inducing Bit-Flips in Memristive Crossbar Memories. In *DATE*, 2022.
- [69] Li-Hsing Yang, Shin-Shan Huang, Tsai-Ling Cheng, Yi-Ching Kuo, and Jian-Jhih Kuo. Socially-Aware Collaborative Defense System against Bit-Flip Attack in Social Internet of Things and Its Online Assignment Optimization. In *ICCCN*, 2022.
- [70] Saad Islam, Koksai Mus, Richa Singh, Patrick Schaumont, and Berk Sunar. Signature Correction Attack on Dilithium Signature Scheme. In *Euro S&P*, 2022.
- [71] Lois Orosa, A Giray Yağlıkçı, Haocong Luo, Ataberk Olgun, Jisung Park, Hasan Hassan, Minesh Patel, Jeremie S. Kim, and Onur Mutlu. A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses. In *MICRO*, 2021.
- [72] Micron Technology. SDRAM, 4Gb: x4, x8, x16 DDR4 SDRAM Features, 2014.
- [73] SK Hynix. 16GB DDR4-2400 RDIMM PC4-19200T-R Dual Rank x4 HMA42GR7AFR4N-UH Module – Datasheet. <https://pdf1.alldatasheet.com/datasheet-pdf/view/1179068/HYNIX/HMA42GR7AFR4N-UH.html>, 2020.
- [74] Samsung Electronics Co., Ltd. 288Pin Unbuffered DIMM based on 8Gb C-die. https://download.semiconductor.samsung.com/resources/data-sheet/DDR4_8Gb_C-die_Unbuffered_DIMM_Rev1.4_Apr.18.pdf, 2018.
- [75] Chung-Hsiang Lin, De-Yu Shen, Yi-Jung Chen, Chia-Lin Yang, and Michael Wang. SECRET: Selective Error Correction for Refresh Energy Reduction in DRAMs. In *ICCD*, 2012.
- [76] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *ISCA*, 2012.
- [77] Prashant J Nair, Dae-Hyun Kim, and Moinuddin K Qureshi. ArchShield: Architectural Framework for Assisting DRAM Scaling by Tolerating High Error Rates. In *ISCA*, 2013.
- [78] Taku Ohsawa, Koji Kai, and Kazuaki Murakami. Optimizing the DRAM Refresh Count for Merged DRAM/Logic LSIs. In *ISLPED*, 1998.
- [79] R.K. Venkatesan, S. Herr, and E. Rotenberg. Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM. In *HPCA*, 2006.
- [80] Jue Wang, Xiangyu Dong, and Yuan Xie. ProactiveDRAM: A DRAM-Initiated Retention Management Scheme. In *ICCD*, 2014.
- [81] Seungjae Baek, Sangyeun Cho, and Rami Melhem. Refresh Now and Then. *TC*, 2014.
- [82] Ishwar Bhati, Mu-Tien Chang, Zeshan Chishty, Shih-Lien Lu, and Bruce Jacob. DRAM Refresh Mechanisms, Penalties, and Trade-Offs. *IEEE TC*, 2015.
- [83] Zehan Cui, Sally A McKee, Zhongbin Zha, Yungang Bao, and Mingyu Chen. DTail: A Flexible Approach to DRAM Refresh Management. In *SC*, 2014.
- [84] Samira Khan, Donghyuk Lee, Yoongu Kim, Alaa R Alameldeen, Chris Wilkerson, and Onur Mutlu. The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study. In *SIGMETRICS*, 2014.
- [85] Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, Onur Mutlu, J Liu, B Jaiyen, Y Kim, C Wilkerson, and O Mutlu. An Experimental Study of Data Retention Behavior in Modern DRAM Devices. In *ISCA*, 2013.
- [86] M.K. Qureshi, Dae-Hyun Kim, S. Khan, P.J. Nair, and O. Mutlu. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. In *DSN*, 2015.
- [87] Ying Wang, Yinhe Han, Cheng Wang, Huawei Li, and Xiaowei Li. RADAR: A Case for Retention-Aware DRAM Assembly and Repair in Future FGR DRAM Memory. In *DAC*, 2015.
- [88] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G Zorn. Flikker: Saving DRAM Refresh-Power Through Critical Data Partitioning. In *ASPLOS*, 2011.
- [89] Kevin K Chang, Prashant J Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K Qureshi, and Onur Mutlu. Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM. In *HPCA*, 2016.
- [90] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd Mowry. RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization. In *MICRO*, 2013.
- [91] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, Onur Mutlu, Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM. In *ISCA*, 2012.
- [92] Brent Keeth et al. *DRAM Circuit Design. Fundamental and High-Speed Topics*. Wiley-IEEE Press, 2007.
- [93] T Schloesser, F Jakubowski, J v Kluge, A Graham, S Slesazek, M Popp, P Baars, K Muemmler, P Moll, K Wilson, et al. 6F2 Buried Wordline DRAM Cell for 40nm and Beyond. In *IEDM*, 2008.
- [94] Tomonori Sekiguchi, Kiyoo Itoh, Tsugio Takahashi, Masahiro Sugaya, Hiroki Fujisawa, Masayuki Nakamura, Kazuhiko Kajigaya, and Katsutaka Kimura. A Low-Impedance Open-Bitline Array for Multigigabit DRAM. *JSSC*, 2002.
- [95] Haocong Luo, Taha Shahroodi, Hasan Hassan, Minesh Patel, Abdullah Giray Yağlıkçı, Lois Orosa, Jisung Park, and Onur Mutlu. CLR-DRAM: A Low-Cost DRAM Architecture Enabling Dynamic Capacity-Latency Trade-Off. In *ISCA*, 2020.
- [96] Kevin K Chang, Prashant J Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K Qureshi, and Onur Mutlu. Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM. In *HPCA*, 2016.
- [97] Bruce Jacob, Spencer Ng, and David Wang. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2008.
- [98] Kiyoo Itoh. *VLSI Memory Chip Design*. Springer, 2001.
- [99] Kiyoo Itoh. Semiconductor Memory. US Patent 4,044,340, April 23, 1977.
- [100] Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu. Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture. In *HPCA*, 2013.
- [101] Onur Mutlu. Retrospective: Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors. *arXiv*, 2023.
- [102] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. RowPress Vulnerability in Modern DRAM Chips. *IEEE Micro*, 2024.
- [103] Minesh Patel, Jeremie S Kim, and Onur Mutlu. The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions. *ISCA*, 2017.
- [104] Christian Weis, Matthias Jung, Peter Ehse, Cristiano Santos, Pascal Vivet, Sven Goossens, Martijn Koedam, and Norbert Wehn. Retention Time Measurements and Modelling of Bit Error Rates of Wide I/O DRAM in MPSoCs. In *DATE*, 2015.
- [105] Matthias Jung, Deepak M Mathew, Carl C Rheinländer, Christian Weis, and Norbert Wehn. A Platform to Analyze DDR3 DRAM's Power and Retention Time. *IEEE Design & Test*, 2017.
- [106] Samira Khan, Chris Wilkerson, Zhe Wang, Alaa R Alameldeen, Donghyuk Lee, and Onur Mutlu. Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content. In *MICRO*, 2017.
- [107] JEDEC Solid State Technology Association et al. DDR3 SDRAM Standard. *JEDEC Standard*, (79-3F):226, 2012.
- [108] JEDEC. *JESD79-4C: DDR4 SDRAM Standard*, 2020.
- [109] Chia-Ming Yang, Jer-Chyi Wang, Wei-Ping Lee, Chien-Chi Lee, Chih-Hung Lin, Chung Yuan Lee, Jo-Hui Lin, Hsin-Huei Chen, Chih-Yuan Hsiao, Ruey-Dar Chang, and Chao-Sung Lai. Superior Improvements in GIDL and Retention by Fluorine Implantation in Saddle-Fin Array Devices for Sub-40-nm DRAM Technology. *IEEE Electron Device Letters*, 2013.
- [110] Sung-Kye Park. Technology Scaling Challenge and Future Prospects of DRAM and NAND Flash Memory. In *IMW*, 2015.
- [111] Yong Liu, Da Wang, Pengpeng Ren, Jie Li, Zheng Qiao, Maokun Wu, Yichen Wen, Longda Zhou, Zixuan Sun, Zirui Wang, Qinghua Han, Blacksmith Wu, Kanyu Cao, Runsheng Wang, Zhigang Ji, and Ru Huang. Understanding Retention Time Distribution in Buried-Channel-Array-Transistors (BCAT) Under Sub-20-nm DRAM Node—Part I: Defect-Based Statistical Compact Model. *IEEE TED*, 2024.
- [112] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits. *Proceedings of the IEEE*, 2003.
- [113] Dong-Su Lee, Young-Hyun Jun, and Bai-Sun Kong. Simultaneous Reverse Body and Negative Word-Line Biasing Control Scheme for Leakage Reduction of DRAM. *IEEE JSSC*, 2011.
- [114] K. Saino et al. Impact of Gate-Induced Drain Leakage Current on the Tail Distribution of DRAM Data Retention Time. In *IEDM*, 2000.
- [115] D. Yaney et al. A Meta-stable Leakage Phenomenon in DRAM Charge Storage - Variable Hold Time. In *IEDM*, 1987.
- [116] P. J. Restle et al. DRAM Variable Retention Time. In *IEDM*, 1992.
- [117] Minesh Patel, Jeremie S Kim, and Onur Mutlu. The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions. In *ISCA*, 2017.
- [118] SAFARI Research Group. DRAM Bender — GitHub Repository. <https://github.com/CMU-SAFARI/DRAM-Bender>, 2022.
- [119] Ataberk Olgun, Hasan Hassan, A Giray Yağlıkçı, Yahya Can Tuğrul, Lois Orosa, Haocong Luo, Minesh Patel, Oguz Ergin, and Onur Mutlu. DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips. *TCAD*, 2023.
- [120] Hasan Hassan, Nandita Vijaykumar, Samira Khan, Saugata Ghose, Kevin Chang, Gennady Pekhimenko, Donghyuk Lee, Oguz Ergin, and Onur Mutlu. SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies. In *HPCA*, 2017.
- [121] SAFARI Research Group. SoftMC — GitHub Repository. <https://github.com/CMU-SAFARI/softmc>, 2017.
- [122] Xilinx Inc. Xilinx Alveo U200 FPGA Board. <https://www.xilinx.com/products/boards-and-kits/alveo/u200.html>.
- [123] Maxwell. FT20X User Manual. <https://www.maxwell-fa.com/upload/files/base/8/m/311.pdf>.
- [124] Ismail Emir Yuksel, Akash Sood, Ataberk Olgun, Oguzhan Canpolat, Haocong

- Luo, Nisa Bostanci, Mohammad Sadrosadati, Giray Yaglikci, and Onur Mutlu. PuDHammer: Experimental Analysis of Read Disturbance Effects of Processing-using-DRAM in Real DRAM Chips. In *ISCA*, 2025.
- [125] Ataberk Olgun, F. Nisa Bostanci, Ismail Emir Yuksel, Oguzhan Canpolat, Haocong Luo, Geraldo F. Oliveira, A. Giray Yaglikci, Minesh Patel, and Onur Mutlu. Variable Read Disturbance: An Experimental Analysis of Temporal Variation in DRAM Read Disturbance. In *HPCA*, 2025.
- [126] Yahya Can Tugrul, A. Giray Yaglikci, Ismail Emir Yuksel, Ataberk Olgun, Oguzhan Canpolat, Nisa Bostanci, Mohammad Sadrosadati, Oguz Ergin, and Onur Mutlu. Understanding RowHammer Under Reduced Refresh Latency: Experimental Analysis of Real DRAM Chips and Implications for Future Solutions. In *HPCA*, 2025.
- [127] Ismail Emir Yuksel, Ataberk Olgun, Nisa Bostanci, Haocong Luo, Giray Yaglikci, and Onur Mutlu. ColumnDisturb: Understanding Column-based Read Disturbance in Real DRAM Chips and Implications for Future Systems. In *arXiv*, 2025.
- [128] Robert T Smith, James D Chlipala, JOHN FM Bindels, Roy G Nelson, Frederick H Fischer, and Thomas F Mantz. Laser Programmable Redundancy and Yield Improvement in a 64K DRAM. *JSSC*, 1981.
- [129] Masashi Horiguchi. Redundancy Techniques for High-Density DRAMs. In *ISIS*, 1997.
- [130] B. Keeth and R.J. Baker. *DRAM Circuit Design: A Tutorial*. John Wiley & Sons, 2001.
- [131] Vivek Seshadri, Thomas Mullins, Amirali Boroumand, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-Unit Strided Accesses. In *MICRO*, 2015.
- [132] Samira Khan, Donghyuk Lee, and Onur Mutlu. PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM. In *DSN*, 2016.
- [133] Donghyuk Lee, Samira Khan, Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungrun, Gennady Pekhimenko, Vivek Seshadri, and Onur Mutlu. Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms. In *SIGMETRICS*, 2017.
- [134] Minesh Patel, Jeremie Kim, Taha Shahroodi, Hasan Hassan, and Onur Mutlu. Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics. In *MICRO*, 2020.
- [135] Hwayong Nam, Seungmin Baek, Minbok Wi, Michael Jaemin Kim, Jaehyun Park, Chihun Song, Nam Sung Kim, and Jung Ho Ahn. Dramscope: Uncovering DRAM Microarchitecture and Characteristics by Issuing Memory Commands. *ISCA*, 2024.
- [136] A. Giray Yaglikci, Haocong Luo, Geraldo F De Oliveira, Ataberk Olgun, Minesh Patel, Jisung Park, Hasan Hassan, Jeremie S Kim, Lois Orosa, and Onur Mutlu. Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices. In *DSN*, 2022.
- [137] A Giray Yaglikci, Ataberk Olgun, Minesh Patel, Haocong Luo, Hasan Hassan, Lois Orosa, Oguz Ergin, and Onur Mutlu. HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips. In *MICRO*, 2022.
- [138] A. Giray Yaglikci, Geraldo Francisco Oliveira, Yahya Can Tugrul, Ismail Emir Yuksel, Ataberk Olgun, Haocong Luo, and Onur Mutlu. Spatial Variation-Aware Read Disturbance Defenses: Experimental Analysis of Real DRAM Chips and Implications on Future Solutions. In *HPCA*, 2024.
- [139] Zhenrong Lang, Patrick Jattke, Michele Marazzi, and Kaveh Razavi. Blaster: Characterizing the blast radius of rowhammer. In *DRAMSec*, 2023.
- [140] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs. In *MICRO*, 2019.
- [141] Ataberk Olgun, Juan Gómez Luna, Konstantinos Kanellopoulos, Behzad Salami, Hasan Hassan, Oguz Ergin, and Onur Mutlu. PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM. *TACO*, 2022.
- [142] Ataberk Olgun, Minesh Patel, A Giray Yaglikci, Haocong Luo, Jeremie S Kim, Nisa Bostanci, Nandita Vijaykumar, Oguz Ergin, and Onur Mutlu. QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips. In *ISCA*, 2021.
- [143] Ismail Emir Yuksel, Yahya Can Tugrul, Ataberk Olgun, F. Nisa Bostanci, A. Giray Yaglikci, Geraldo F. de Oliveira, Haocong Luo, Juan Gomez Luna, Mohammad Sadrosadati, and Onur Mutlu. Functionally-Complete Boolean Logic in Real DRAM Chips: Experimental Characterization and Analysis. In *HPCA*, 2024.
- [144] Ismail Emir Yuksel, Yahya Can Tugrul, F. Nisa Bostanci, Geraldo F. de Oliveira, A. Giray Yaglikci, Ataberk Olgun, Melina Soysal, Haocong Luo, Juan Gomez Luna, Mohammad Sadrosadati, and Onur Mutlu. Simultaneous Many-Row Activation in Off-the-Shelf DRAM Chips: Experimental Characterization and Analysis. In *DSN*, 2024.
- [145] Onur Mutlu, Ataberk Olgun, Geraldo F Oliveira, and Ismail E Yuksel. Memory-Centric Computing: Recent Advances in Processing-in-DRAM. In *IEDM*, 2024.
- [146] Onur Mutlu, Ataberk Olgun, and Ismail Emir Yuksel. Memory-Centric Computing: Solving Computing's Memory Problem. In *IMW*, 2025.
- [147] JEDEC. *JESD79-5: DDR5 SDRAM Standard*, 2020.
- [148] JEDEC. *JESD79-5c: DDR5 SDRAM Standard*, 2024.
- [149] Oguzhan Canpolat, A Giray Yaglikci, Geraldo F Oliveira, Ataberk Olgun, Oguz Ergin, and Onur Mutlu. Understanding the Security Benefits and Overheads of Emerging Industry Solutions to DRAM Read Disturbance. *DRAMSec*, 2024.
- [150] Minesh Patel, Geraldo Francisco de Oliveira Jr., and Onur Mutlu. HARP: Practically and Effectively Identifying Uncorrectable Errors in Main Memory Chips That Use On-Die ECC. In *MICRO*, 2021.
- [151] Minesh Patel. *Enabling Effective Error Mitigation in Modern Memory Chips that Use On-Die Error-Correcting Codes*. PhD thesis, 2021.
- [152] A.J. van de Goor and I. Schanstra. Address and Data Scrambling: Causes and Impact on Memory Tests. In *DELTA*, 2002.
- [153] Hwayong Nam, Seungmin Baek, Minbok Wi, Michael Jaemin Kim, Jaehyun Park, Chihun Song, Nam Sung Kim, and Jung Ho Ahn. X-ray: Discovering DRAM Internal Structure and Error Characteristics by Issuing Memory Commands. *IEEE CAL*, 2023.
- [154] Haocong Luo, Ismail Emir Yuksel, Ataberk Olgun, A Giray Yaglikci, and Onur Mutlu. Revisiting DRAM Read Disturbance: Identifying Inconsistencies Between Experimental Characterization and Device-Level Studies. In *VTS*, 2025.
- [155] Longda Zhou, Sheng Ye, Runsheng Wang, and Zhigang Ji. Unveiling RowPress in Sub-20 nm DRAM Through Comparative Analysis With Row Hammer: From Leakage Mechanisms to Key Features. In *IEEE TED*, 2024.
- [156] Longda Zhou, Jie Li, Pengpeng Ren, Sheng Ye, Da Wang, Zheng Qiao, and Zhigang Ji. Understanding the Physical Mechanism of RowPress at the Device-Level in Sub-20 nm DRAM. In *IRPS*, 2024.
- [157] Oguzhan Canpolat, A Giray Yaglikci, Geraldo F Oliveira, Ataberk Olgun, Nisa Bostanci, Ismail Emir Yuksel, Haocong Luo, Oguz Ergin, and Onur Mutlu. Chronus: Understanding and Securing the Cutting-Edge Industry Solutions to DRAM Read Disturbance. In *HPCA*, 2025.
- [158] Timothy J Dell. A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory. *IBM Microelectronics Division*, 1997.
- [159] Seong-Lyong Gong. *Memory Protection Techniques for DRAM Scaling-induced Errors*. PhD thesis, 2018.
- [160] Richard W Hamming. Error Detecting and Error Correcting Codes. *The Bell system technical journal*, 1950.
- [161] Uksong Kang, Hak-Soo Yu, Churoo Park, Hongzhong Zheng, John Halbert, Kuljit Bains, S Jang, and Joo Sun Choi. Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling. In *The Memory Forum*, 2014.
- [162] Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu. Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field. In *DSN*, 2015.
- [163] ECC Brings Reliability and Power Efficiency to Mobile Devices. Technical report, Micron Technology inc., 2017.
- [164] Prashant J Nair, Vilas Sridharan, and Moinuddin K Qureshi. XED: Exposing on-die error detection information for strong memory reliability. In *ISCA*, 2016.
- [165] Minesh Patel, Jeremie S Kim, Hasan Hassan, and Onur Mutlu. Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices. In *DSN*, 2019.
- [166] B. Schroder et al. DRAM Errors in the Wild: A Large-Scale Field Study. In *SIGMETRICS*, 2009.
- [167] Vilas Sridharan and Dean Liberty. A Study of DRAM Failures in the Field. In *SC*, 2012.
- [168] Intelligent Memory. IM ECC DRAM with Integrated Error Correcting Code. https://www.intelligentmemory.com/fileadmin/Editors/pdf/PB_I_M_ECC_DRAM.pdf, 2016.
- [169] Nohhyung Kwak, Saeng-Hwan Kim, Kyong Ha Lee, Chang-Ki Baek, Mun Seon Jang, Yongsuk Joo, Seung-Hun Lee, Woo Young Lee, Eunyeong Lee, and Donghee Han. 23.3 A 4.8 Gb/s/pin 2Gb LPDDR4 SDRAM with Sub-100μA Self-Refresh Current for IoT Applications. In *ISSCC*, 2017.
- [170] Tae-Young Oh, Hoeju Chung, Jun-Young Park, Ki-Won Lee, Seunghoon Oh, Su-yeon Doo, Hyoung-Joo Kim, ChangYong Lee, Hye-Ran Kim, Jong-Ho Lee, et al. A 3.2Gbps/pin 8Gb 1.0V LPDDR4 SDRAM with Integrated ECC Engine for sub-1V DRAM Core Operation. In *ISSCC*, 2014.
- [171] Young Hoon Son, Sukhan Lee, O Seongil, Sanghyuk Kwon, Nam Sung Kim, and Jung Ho Ahn. CiDRA: A Cache-Inspired DRAM Resilience Architecture. In *HPCA*, 2015.
- [172] Minesh Patel, Taha Shahroodi, Aditya Manglik, A. Giray Yaglikci, Ataberk Olgun, Haocong Luo, and Onur Mutlu. A Case for Transparent Reliability in DRAM Systems. arXiv:2204.10378, 2022.
- [173] Irina Alam and Puneet Gupta. COMET: On-die and In-controller Collaborative Memory ECC Technique for Safer and Stronger Correction of DRAM Errors. In *DSN*, 2022.
- [174] Dongwee Kim, Jaeyoon Lee, Wonyeong Jung, Michael Sullivan, and Jungrae Kim. Unity ECC: Unified Memory Protection Against Bit and Chip Errors. In *SC*, 2023.
- [175] Minesh Patel, Taha Shahroodi, Aditya Manglik, A Giray Yaglikci, Ataberk Olgun, Haocong Luo, and Onur Mutlu. Rethinking the producer-consumer relationship in modern dram-based systems. *IEEE Access*, 2024.
- [176] Sangunh Cha, O. Seongil, Hyunsung Shin, Sangjoon Hwang, Kwangil Park, Seong Jin Jang, Joo Sun Choi, Gyo Young Jin, Young Hoon Son, Hyunyeon Cho, Jung Ho Ahn, and Nam Sung Kim. Defect Analysis and Cost-Effective Resilience Architecture for Future DRAM Devices. In *HPCA*, 2017.
- [177] Apple Inc. About the Security Content of Mac EFI Security Update 2015-001. <https://support.apple.com/en-us/HT204934>, 2015. June 2015.
- [178] Hewlett-Packard Enterprise. HP Moonshot Component Pack Version 2015.05.0. <http://h17007.www1.hp.com/us/en/enterprise/servers/products/moonshot/component-pack/index.aspx>, 2015.
- [179] Lenovo. Row Hammer Privilege Escalation. https://support.lenovo.com/us/en/product_security/row_hammer, 2015.
- [180] Zvika Greenfield and Tomer Levy. Throttling Support for Row-Hammer Counters, 2016. U.S. Patent 9,251,885.
- [181] Dae-Hyun Kim, Prashant J Nair, and Moinuddin K Qureshi. Architectural Support for Mitigating Row Hammering in DRAM Memories. *CAL*, 2014.
- [182] K.S. Bains and J.B. Halbert. Distributed Row Hammer Tracking. US Patent App. 13/631,781, April 3 2014.
- [183] K.S. Bains et al. Method, Apparatus and System for Providing a Memory Refresh. US Patent: 9,030,903, 2015.
- [184] K.S. Bains et al. Row Hammer Refresh Command. US Patent App. 13/539,415, 2014.
- [185] K. Bains et al. Row Hammer Refresh Command. US Patent App. 14/068,677, 2014.
- [186] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks. In *ASPLOS*, 2016.
- [187] Kuljit Bains, John Halbert, Christopher Mozak, Theodore Schoenborn, and Zvika Greenfield. Row Hammer Refresh Command, 2015. U.S. Patent 9,117,544.
- [188] Kuljit S Bains and John B Halbert. Row Hammer Monitoring Based on Stored Row Hammer Threshold Value. US Patent: 10,083,737, 2016. U.S. Patent 9,384,821.
- [189] Kuljit S Bains and John B Halbert. Distributed Row Hammer Tracking, 2016. U.S. Patent 9,299,400.
- [190] Mungyu Son, Hyunsun Park, Junwhan Ahn, and Sungjoo Yoo. Making DRAM

- Stronger Against Row Hammering. In *DAC*, 2017.
- [191] S. M. Seyedzadeh, A. K. Jones, and R. Melhem. Mitigating Wordline Crosstalk Using Adaptive Trees of Counters. In *ISCA*, 2018.
- [192] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. MASCAT: Stopping Microarchitectural Attacks Before Execution. *IACR Cryptology*, 2016.
- [193] Jung Min You and Joon-Sung Yang. MRLoc: Mitigating Row-Hammering Based on Memory Locality. In *DAC*, 2019.
- [194] Eojin Lee, Ingab Kang, Sukhan Lee, G Edward Suh, and Jung Ho Ahn. TWiCe: Preventing Row-Hammering by Exploiting Time Window Counters. In *ISCA*, 2019.
- [195] Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, and Jae W Lee. Graphene: Strong yet Lightweight Row Hammer Protection. In *MICRO*, 2020.
- [196] A. Giray Yağlıkcı, Jeremie S. Kim, Fabrice Devaux, and Onur Mutlu. Security Analysis of the Silver Bullet Technique for RowHammer Prevention. arXiv:2106.07084, 2021.
- [197] A Giray Yağlıkcı, Minesh Patel, Jeremie S. Kim, Roknoddin Azizibarzoki, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, Saugata Ghose, and Onur Mutlu. BlockHammer: Preventing Row-Hammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows. In *HPCA*, 2021.
- [198] Ingab Kang, Eojin Lee, and Jung Ho Ahn. CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention. *IEEE Access*, 2020.
- [199] Moinuddin Qureshi, Aditya Rohan, Gururaj Saileshwar, and Prashant J Nair. Hydra: Enabling Low-Overhead Mitigation of Row-Hammer at Ultra-Low Thresholds via Hybrid Tracking. In *ISCA*, 2022.
- [200] Gururaj Saileshwar, Bolin Wang, Moinuddin Qureshi, and Prashant J Nair. Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation Between Aggressor and Victim Rows. In *ASPLOS*, 2022.
- [201] Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks. In *OSDI*, 2018.
- [202] Saru Vig, Sarani Bhattacharya, Debdeep Mukhopadhyay, and Siew-Kei Lam. Rapid Detection of Rowhammer Attacks Using Dynamic Skewed Hash Tree. In *HASP*, 2018.
- [203] Michael Jaemin Kim, Jaehyun Park, Yeonhong Park, Wanju Doh, Namhoon Kim, Tae Jun Ham, Jae W Lee, and Jung Ho Ahn. Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh. In *HPCA*, 2022.
- [204] Gyu-Hyeon Lee, Seongmin Na, Ilkwon Byun, Dongmoon Min, and Jangwoo Kim. CryoGuard: A Near Refresh-Free Robust DRAM Design for Cryogenic Computing. In *ISCA*, 2021.
- [205] Michele Marazzi, Patrick Jattke, Flavien Solt, and Kaveh Razavi. ProTRR: Principled yet Optimal In-DRAM Target Row Refresh. In *S&P*, 2022.
- [206] Zhi Zhang, Yueqiang Cheng, Minghua Wang, Wei He, Wenhao Wang, Surya Nepal, Yansong Gao, Kang Li, Zhe Wang, and Chenggang Wu. SoftTRR: Protect Page Tables against Rowhammer Attacks using Software-only Target Row Refresh. In *USENIX ATC*, 2022.
- [207] Bireesh Kumar Joardar, Tyler K Bletsch, and Krishnendu Chakrabarty. Learning to Mitigate RowHammer Attacks. In *DATE*, 2022.
- [208] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Maria Eichlseder, Moritz Lipp, and Daniel Gruss. CSI: Rowhammer—Cryptographic Security and Integrity against Rowhammer. In *S&P*, 2023.
- [209] Anish Saxena, Gururaj Saileshwar, Prashant J. Nair, and Moinuddin Qureshi. AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime. In *MICRO*, 2022.
- [210] Shuhei Enomoto, Hiroki Kuzuno, and Hiroshi Yamada. Efficient Protection Mechanism for CPU Cache Flush Instruction Based Attacks. *IEICE TIS*, 2022.
- [211] Evgeny Manzhosov, Adam Hastings, Meghna Pancholi, Ryan Piersma, Mohamed Tarek Ibn Ziad, and Simha Sethumadhavan. Revisiting Residue Codes for Modern Memories. In *MICRO*, 2022.
- [212] Samira Mirbagher Ajorpaz, Daniel Moghimi, Jeffrey Neal Collins, Gilles Pokam, Nael Abu-Ghazaleh, and Dean Tullsen. EVAX: Towards a Practical, Pro-active & Adaptive Architecture for High Performance & Security. In *MICRO*, 2022.
- [213] Amir Naseredini, Martin Berger, Matteo Sammartino, and Shale Xiong. ALARM: Active Learning of Rowhammer Mitigations. <https://users.sussex.ac.uk/~mfb21/rh-draft.pdf>, 2022.
- [214] Bireesh Kumar Joardar, Tyler K. Bletsch, and Krishnendu Chakrabarty. Machine Learning-based Rowhammer Mitigation. *TCAD*, 2022.
- [215] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Bo Li, Peter Volgyesi, and Xenofon Koutsoukos. Leveraging EM Side-Channel Information to Detect Rowhammer Attacks. In *S&P*, 2020.
- [216] Kevin Loughlin, Stefan Saroiu, Alec Wolman, and Baris Kasikci. Stop! Hammer Time: Rethinking Our Approach to Rowhammer Mitigations. In *HotOS*, 2021.
- [217] Fabrice Devaux and Renaud Ayrignac. Method and Circuit for Protecting a DRAM Memory Device from the Row Hammer Effect. US Patent: 10,885,966, 2021. 10,885,966.
- [218] Ali Fakhrzadehgan, Yale N. Patt, Prashant J. Nair, and Moinuddin K. Qureshi. SafeGuard: Reducing the Security Risk from Row-Hammer via Low-Cost Integrity Protection. In *HPCA*, 2022.
- [219] Stefan Saroiu, Alec Wolman, and Lucian Cojocar. The Price of Secrecy: How Hiding Internal DRAM Topologies Hurts Rowhammer Defenses. In *IRPS*, 2022.
- [220] Kevin Loughlin, Stefan Saroiu, Alec Wolman, Yatin A. Manerker, and Baris Kasikci. MOESI-Prime: Preventing Coherence-Induced Hammering in Commodity Workloads. In *ISCA*, 2022.
- [221] Jin Han, Jungsik Kim, Dafna Beery, K Deniz Bozdog, Peter Cuevas, Amitay Levi, Irwin Tain, Khai Tran, Andrew J Walker, Senthil Vadakupudhu Palayam, et al. Surround Gate Transistor With Epitaxially Grown Si Pillar and Simulation Study on Soft Error and Rowhammer Tolerance for DRAM. *TED*, 2021.
- [222] Jeonghyun Woo, Gururaj Saileshwar, and Prashant J Nair. Scalable and Secure Row-Swap: Efficient and Safe Row Hammer Mitigation in Memory Systems. In *HPCA*, 2023.
- [223] Carsten Bock, Ferdinand Brasser, David Gens, Christopher Liebchen, and Ahamd Reza Sadeghi. RIP-RH: Preventing Rowhammer-Based Inter-Process Attacks. In *ACCS*, 2019.
- [224] Dae-Hyun Kim, Prashant J Nair, and Moinuddin K Qureshi. Architectural Support for Mitigating Row Hammering in DRAM Memories. *CAL*, 2015.
- [225] Yicheng Wang, Yang Liu, Peiyun Wu, and Zhao Zhang. Discreet-PARA: Rowhammer Defense with Low Cost and High Efficiency. In *ICCD*, 2021.
- [226] Tanj Bennett, Stefan Saroiu, Alec Wolman, and Lucian Cojocar. Panopticon: A Complete In-DRAM Rowhammer Mitigation. In *DRAMSec*, 2021.
- [227] Ataberk Olgun, Yahya Can Tuğrul, Nisa Bostanci, Ismail Emir Yüksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yağlıkcı, Geraldo F. Oliveira, and Onur Mutlu. ABA-CuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation. In *USENIX Security*, 2024.
- [228] F Nisa Bostanci, Ismail Emir Yüksel, Ataberk Olgun, Konstantinos Kanellopoulos, Yahya Can Tuğrul, A Giray Yağlıcı, Mohammad Sadrosadati, and Onur Mutlu. CoMeT: Count-Min-Sketch-Based Row Tracking to Mitigate RowHammer at Low Cost. In *HPCA*, 2024.
- [229] Hritvik Taneja and Moin Qureshi. DREAM: Enabling Low-Overhead Rowhammer Mitigation via Directed Refresh Management. In *ISCA*, 2025.
- [230] Suhas Vittal, Salman Qazi, Poulami Das, and Moinuddin Qureshi. MoPAC: Efficiently Mitigating Rowhammer with Probabilistic Activation Counting. In *ISCA*, 2025.
- [231] Chris S Lin, Jeonghyun Woo, Prashant J Nair, and Gururaj Saileshwar. CnC-PRAC: Coalesce, not Cache, Per Row Activation Counts for an Efficient in-DRAM Rowhammer Mitigation. *DRAMSec*, 2025.
- [232] Salman Qazi and Moinuddin Qureshi. DRFM and the Art of Rowhammer Sampling. *DRAMSec*, 2025.
- [233] Shih-Lien Lu, Jeonghyun Woo, and Prashant J Nair. Counterpoint: One-Hot Counting for PRAC-Based RowHammer Mitigation. *DRAMSec*, 2025.
- [234] Moinuddin Qureshi. AutoRFM: Scaling Low-Cost in-DRAM Trackers to Ultra-Low Rowhammer Thresholds. In *HPCA*, 2025.
- [235] Jeonghyun Woo and Prashant J Nair. DAPPER: A Performance-Attack-Resilient Tracker for RowHammer Defense. In *HPCA*, 2025.
- [236] Jeonghyun Woo, Shaopeng Chris Lin, Prashant J Nair, Aamer Jaleel, and Gururaj Saileshwar. QPRAC: Towards Secure and Practical PRAC-based Rowhammer Mitigation using Priority Queues. In *HPCA*, 2025.
- [237] F. Nisa Bostanci, Oğuzhan Canpolat, Ataberk Olgun, İsmail Emir Yüksel, Konstantinos Kanellopoulos, Mohammad Sadrosadati, A Giray Yağlıkcı, and Onur Mutlu. Understanding and Mitigating Covert Channel and Side Channel Vulnerabilities Introduced by RowHammer Defenses. In *MICRO*, 2025.
- [238] Chulseung Lim, Kyungbae Park, and Sanghyeon Baeg. Active Precharge Hammering to Monitor Displacement Damage Using High-Energy Protons in 3x-nm SDRAM. *TNS*, 2017.
- [239] Kyungbae Park, Donghyuk Yun, and Sanghyeon Baeg. Statistical Distributions of Row-Hammering Induced Failures in DDR3 Components. *Microelectronics Reliability*, 2016.
- [240] Kyungbae Park, Chulseung Lim, Donghyuk Yun, and Sanghyeon Baeg. Experiments and Root Cause Analysis for Active-Precharge Hammering Fault in DDR3 SDRAM under 3xnm Technology. *Microelectronics Reliability*, 2016.
- [241] Seong-Wan Ryu, Kyungkyu Min, Jungho Shin, Heimi Kwon, Donghoon Nam, Taekyung Oh, Tae-Su Jang, Minsoo Yoo, Yongtaik Kim, and Sungjoo Hong. Overcoming the Reliability Limitation in the Ultimately Scaled DRAM using Silicon Migration Technique by Hydrogen Annealing. In *IEDM*, 2017.
- [242] Donghyuk Yun, Myungsang Park, Chulseung Lim, and Sanghyeon Baeg. Study of TID Effects on One Row Hammering using Gamma in DDR4 SDRAMs. In *IRPS*, 2018.
- [243] Chulseung Lim, Kyungbae Park, Geunyoung Bak, Donghyuk Yun, Myungsang Park, Sanghyeon Baeg, Shi-Jie Wen, and Richard Wong. Study of Proton Radiation Effect to Row Hammer Fault in DDR4 SDRAMs. *Microelectronics Reliability*, 2018.
- [244] Ataberk Olgun, Majd Osseiran, Abdullah Giray Yağlıkcı, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. An Experimental Analysis of RowHammer in HBM2 DRAM Chips. In *DSN Disrupt*, 2023.
- [245] Wei He, Zhi Zhang, Yueqiang Cheng, Wenhao Wang, Wei Song, Yansong Gao, Qifei Zhang, Kang Li, Dongxi Liu, and Surya Nepal. WhistleBlower: A System-level Empirical Study on RowHammer. *TC*, 2023.
- [246] Haocong Luo, Ismail Emir Yüksel, Ataberk Olgun, A Giray Yağlıkcı, Mohammad Sadrosadati, and Onur Mutlu. An Experimental Characterization of Combined RowHammer and RowPress Read Disturbance in Modern DRAM Chips. In *DSN Disrupt*, 2024.
- [247] Micron Technology. 16Gb DDR5 SDRAM Addendum, 2021.
- [248] Hasan Hassan, Ataberk Olgun, A Giray Yağlıkcı, Haocong Luo, Onur Mutlu, and ETH Zurich. Self-Managing DRAM: A Low-Cost Framework for Enabling Autonomous and Efficient DRAM Maintenance Operations. In *MICRO*, 2024.
- [249] SAFARI Research Group. Self-Managing DRAM (SMD) — GitHub Repository. <https://github.com/CMU-SAFARI/SelfManagingDRAM>, 2024.
- [250] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A Fast and Extensible DRAM Simulator. *CAL*, 2016.
- [251] SAFARI Research Group. Ramulator — GitHub Repository. <https://github.com/CMU-SAFARI/ramulator>.
- [252] Haocong Luo, Yahya Can Tuğrul, F. Nisa Bostanci, Ataberk Olgun, A. Giray Yağlıkcı, and Onur Mutlu. Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator. *CAL*, 2023.
- [253] SAFARI Research Group. Ramulator V2.0. <https://github.com/CMU-SAFARI/ramulator2>.
- [254] B Bloom. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *CACM*, 1970.
- [255] Saar Cohen and Yossi Matias. Spectral Bloom Filters. In *SIGMOD*, 2003.
- [256] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An Improved Construction for Counting Bloom Filters. In *ESA*, 2006.
- [257] Matthias Jung, Éder Zúlian, Deepak M. Mathew, Matthias Herrmann, Christian

- Brugger, Christian Weis, and Norbert Wehn. Omitting Refresh: A Case Study for Commodity and Wide I/O DRAMs. In *MEMSYS*, 2015.
- [258] Samira Khan, Chris Wilkerson, Donghyuk Lee, Alaa R Alameldeen, and Onur Mutlu. A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM. *CAL*, 2016.
- [259] Hyunwoo Chung, Huijung Kim, Hyungi Kim, Kanguk Kim, Sua Kim, Ki-Wan Song, Jiyoung Kim, Yong Chul Oh, Yoosang Hwang, Hyeongsun Hong, Gyo-Young Jin, and Chilhee Chung. Novel 4F2 DRAM cell with Vertical Pillar Transistor(VPT). In *ESSDERC*, 2011.
- [260] Kyung Kyu Min, Sungmin Hwang, Jong-Ho Lee, and Byung-Gook Park. Vertical Inner Gate Transistors for 4F2 DRAM Cell. *IEEE TED*, 2020.
- [261] Youngseung Cho, Pyeongho Choi, Younghwan Hyeon, Junhwa Song, Yoosang Hwang, and Byoungdeog Choi. Novel Band-to-Band Tunneling Body Contact (BTBC) Structure to Suppress the Floating- Body Effect in a Vertical-Cell DRAM. *Electron Device Letters*, 2018.
- [262] Youngseung Cho, Huijung Kim, Kyungho Jung, Bongsoo Kim, Yoosang Hwang, Hyeongsun Hong, and Byoungdeog Choi. Suppression of the Floating-Body Effect of Vertical-Cell DRAM With the Buried Body Engineering Method. *TED*, 2018.
- [263] Yong Liu, Da Wang, Pengpeng Ren, Runsheng Wang, Zhigang Ji, and Ru Huang. Bit Line Hammering in Si-Based VCT DRAM: A New Security Challenge and Its Mitigation. *Electron Device Letters*, 2025.