RowHammer: A Retrospective

Onur Mutlu[§‡] Jeremie S. Kim[‡§]
[§]ETH Zürich [‡]Carnegie Mellon University

***Abstract*—This retrospective paper describes the RowHammer problem in Dynamic Random Access Memory (DRAM), which was initially introduced by Kim et al. at the ISCA 2014 conference [133]. RowHammer is a prime (and perhaps the first) example of how a circuit-level failure mechanism can cause a practical and widespread system security vulnerability. It is the phenomenon that repeatedly accessing a row in a modern DRAM chip causes bit flips in physically-adjacent rows at consistently predictable bit locations. RowHammer is caused by a hardware failure mechanism called *DRAM disturbance errors*, which is a manifestation of circuit-level cell-to-cell interference in a scaled memory technology.**

Researchers from Google Project Zero demonstrated in 2015 that this hardware failure mechanism can be effectively exploited by user-level programs to gain kernel privileges on real systems. Many other follow-up works demonstrated other practical attacks exploiting RowHammer. In this article, we comprehensively survey the scientific literature on RowHammer-based attacks as well as mitigation techniques to prevent RowHammer. We also discuss what other related vulnerabilities may be lurking in DRAM and other types of memories, e.g., NAND flash memory or Phase Change Memory, that can potentially threaten the foundations of secure systems, as the memory technologies scale to higher densities. We conclude by describing and advocating a principled approach to memory reliability and security research that can enable us to better anticipate and prevent such vulnerabilities.

***Index Terms*—DRAM, Security, Vulnerability, Technology Scaling, Reliability, Errors, Memory Systems.**

I. Introduction and Outline

Memory is a key component of all modern computing systems, often determining the overall performance, energy efficiency, and reliability characteristics of the entire system. The push for increasing the density of modern memory technologies via technology scaling, which has resulted in higher capacity (i.e., density) memory and storage at lower cost, has enabled large leaps in the performance of modern computers [175]. This positive trend is clearly visible in especially the dominant main memory and solid-state storage technologies of today, i.e., DRAM [57, 58, 125, 146, 151] and NAND flash memory [42, 45, 52], respectively. Unfortunately, the same push has also greatly decreased the reliability of modern memory technologies, due to the increasingly smaller memory cell size and increasingly smaller amount of charge that is maintainable in the cell, which makes the memory cell much more vulnerable to various failure mechanisms and noise and interference sources, both in DRAM [113, 115–118, 133, 161, 176, 196, 203] and NAND flash nemory [42–48, 50–53, 164, 166–168, 176].

As memory scales down to smaller technology nodes, new failure mechanisms emerge that threaten its correct operation. If such failure mechanisms are not anticipated and corrected, they can not only degrade system reliability and availability but also, perhaps even more importantly, open up new security vulnerabilities: a malicious attacker can exploit the exposed failure mechanism to take over the entire system. As such, new failure mechanisms in memory can become practical and significant threats to system security.

In this article, we provide a retrospective of one such example failure mechanism in DRAM, which was initially introduced by Kim et al. at the ISCA 2014 conference [133]. We provide a description of the RowHammer problem and its implications by summarizing our ISCA 2014 paper [133], describe solutions proposed by our original work [133], comprehensively examine the many works that build on our original work in various ways, e.g., by developing new security attacks, proposing solutions, and analyzing RowHammer. What comes next in this section provides a roadmap of the entire article.

In our ISCA 2014 paper [133], we introduce the RowHammer problem in DRAM, which is a prime (and perhaps the first) example of how a circuit-level failure mechanism can cause a practical and widespread system security vulnerability. RowHammer, as it is now popularly referred to, is the phenomenon that repeatedly accessing a row in a modern DRAM chip causes bit flips in physically-adjacent rows at consistently predictable bit locations. It is caused by a hardware failure mechanism called DRAM disturbance errors, which is a manifestation of circuit-level cell-to-cell interference in a scaled memory technology. We describe the RowHammer problem and its root causes in Section II.

Inspired by our ISCA 2014 paper's fundamental findings, researchers from Google Project Zero demonstrated in 2015 that this hardware failure mechanism can be effectively exploited by user-level programs to gain kernel privileges on real systems [215, 216]. Tens of other works since then demonstrated other practical attacks exploiting RowHammer, e.g., [12, 31–33, 37, 54, 66, 73, 76, 89, 90, 108, 158, 198, 199, 201, 207, 235, 236, 241, 251, 260]. These include remote takeover of a server vulnerable to RowHammer, takeover of a victim virtual machine by another virtual machine running on the same system, takeover of a mobile device by a malicious user-level application that requires no permissions, takeover of a mobile system quickly by triggering RowHammer using a mobile GPU, and takeover of a remote system by triggering RowHammer on it through the Remote Direct Memory Access (RDMA) protocol [6]. We describe the works that build on RowHammer to develop new security attacks in Section III-A.

Our ISCA 2014 paper rigorously and experimentally analyzes the RowHammer problem and examines seven different solutions, multiple of which are already employed in practice to prevent the security vulnerabilities (e.g., increasing the

memory refresh rate). We propose a low-cost solution, Probabilistic Adjacent Row Activation, which provides a strong and configurable reliability and security guarantee; a solution whose variants are being adopted by DRAM manufacturers and memory controller designers [5]. We describe this solution and the six other solutions of our original paper in Section II-E. Many other works build on our original paper to propose and evaluate other solutions to RowHammer, and we discuss them comprehensively in Section III-B.

Our ISCA 2014 paper leads to a new mindset that has enabled a renewed interest in hardware security research: general-purpose hardware is fallible, in a very widespread manner, and this causes real security problems. We believe the RowHammer problem will become worse over time since DRAM cells are getting closer to each other with technology scaling. Other similar vulnerabilities may also be lurking in DRAM and other types of memories, e.g., NAND flash memory or Phase Change Memory, that can potentially threaten the foundations of secure systems, as the memory technologies scale to higher densities. Our ISCA 2014 paper advocates a principled system-memory co-design approach to memory reliability and security research that can enable us to better anticipate and prevent such vulnerabilities. We describe promising ongoing and future research directions related to RowHammer (Section IV), including the examination of other potential vulnerabilities in memory (in Section IV-A) and the use of a principled approach to make memory more reliable and more secure (in Section IV-B).

II. The RowHammer Problem: A Summary

Memory isolation is a key property of a reliable and secure computing system. An access to one memory address should not have unintended side effects on data stored in other addresses. However, as process technology scales down to smaller dimensions, memory chips become more vulnerable to *disturbance*, a phenomenon in which different memory cells interfere with each others' operation. We have shown, in our ISCA 2014 paper [133], the existence of *disturbance errors* in commodity DRAM chips that are sold and used in the field. Repeatedly reading from the same address in DRAM could corrupt data in nearby addresses. Specifically, when a DRAM row is opened (i.e., activated) and closed (i.e., precharged) repeatedly (i.e., *hammered*), enough times within a DRAM refresh interval, one or more bits in physically-adjacent DRAM rows can be flipped to the wrong value. This DRAM failure mode is now popularly called *RowHammer* [1, 2, 23, 37, 90, 131, 141, 207, 215, 216, 241, 246]. Using an FPGA-based experimental DRAM testing infrastructure, which we originally developed for testing retention time issues in DRAM [161],[1] we tested 129 DRAM modules manufactured by three major manufacturers (A, B, C) in seven recent years (2008–2014) and found that 110 of them exhibited RowHammer errors, the earliest of which dates back to 2010. This is illustrated in Figure 1, which shows the error rates we found in all 129 modules we tested where modules are categorized based on manufacturing date.[2] In particular, *all* DRAM modules from 2012–2013 were vulnerable to RowHammer, indicating that RowHammer is a recent phenomenon affecting more advanced process technology generations (as also demonstrated repeatedly by various works that come after our ISCA 2014 paper [12, 17, 66, 141, 198, 241]).

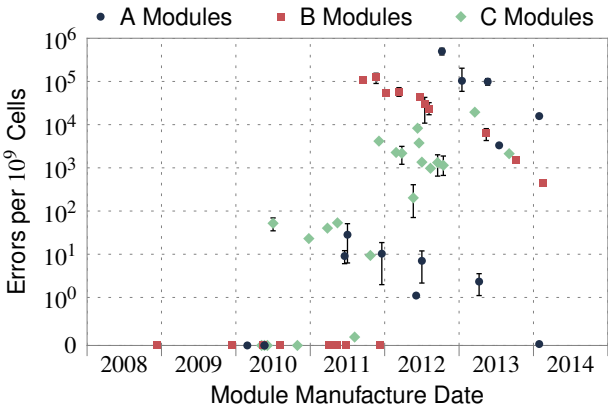


Fig. 1: RowHammer error rate vs. manufacturing dates of 129 DRAM modules we tested (reproduced from [133]).

A. RowHammer Mechanisms

In general, disturbance errors occur whenever there is a strong enough interaction between two circuit components (e.g., capacitors, transistors, wires) that should be isolated from each other. Depending on which component interacts with which other component and also how they interact, many different modes of disturbance are possible.

Among them, our ISCA 2014 paper identifies one particular disturbance mode that affects commodity DRAM chips from all three major manufacturers. *When a wordline's voltage is toggled repeatedly, some cells in nearby rows leak charge at a much faster rate than others.* Such *vulnerable cells*, if disturbed enough times, cannot retain enough charge for even 64*ms*, the time interval at which they are refreshed. Ultimately, this leads to the cells losing data and experiencing disturbance errors.

Without analyzing existing DRAM chips at the device-level, which is an option not available for us, we cannot make definitive claims about how a wordline interacts with nearby cells to increase their leakiness. Our ISCA 2014 paper hypothesizes, based on past studies and findings, that there may be three ways of interaction. At least two major DRAM manufacturers have confirmed all three of these hypotheses as potential causes of disturbance errors. First, changing the voltage of a wordline could inject noise into an adjacent wordline through *electromagnetic coupling* [60, 173, 208]. This

[1]This infrastructure is currently released to the public, and is described in detail in our HPCA 2017 paper [98]. The infrastructure has enabled many studies [57, 58, 79, 98, 115–117, 133, 147, 148, 151, 161, 203] into the failure and performance characteristics of modern DRAM, which were previously not well understood.

[2]Test details and experimental setup, along with a listing of all modules and their characteristics, are reported in our original RowHammer paper [133].

partially enables the adjacent row of access-transistors for a short amount of time and facilitates the leakage of charge from vulnerable cells. Thus, if a row is hammered enough times to disturb such vulnerable cells before they get refreshed, charge in such cells get drained to a point that the original cell values are not recoverable any more. Second, *bridges* are a well-known class of DRAM faults in which conductive channels are formed between unrelated wires and/or capacitors [19, 20]. One study on embedded DRAM (eDRAM) found that toggling a wordline could accelerate the flow of charge between two bridged cells [104]. Third, it has been reported that toggling a wordline for hundreds of hours can permanently damage it by *hot-carrier injection* [64]. If some of the hot-carriers are injected into the neighboring rows, this could modify the amount of charge in their cells or alter the characteristics of their access-transistors to increase their leakiness.

Several recent works have tried to examine and model RowHammer at the circuit level; we survey these works in Section III-C.

B. User-Level RowHammer

Our ISCA 2014 paper also demonstrates that a very simple user-level program [3, 133] can reliably and consistently induce RowHammer errors in three commodity AMD and Intel systems using vulnerable DRAM modules. We released the source code of this program [3], which Google Project Zero later enhanced [4]. Using our user-level RowHammer test program, we showed that RowHammer errors violate two invariants that memory should provide: (i) a read access should not modify data at any address and (ii) a write access should modify data only at the address that it is supposed to write to. As long as a row is repeatedly opened, both read and write accesses can induce RowHammer errors, all of which occur in rows other than the one that is being accessed. Since different DRAM rows are mapped (via mechanisms in the system software and the memory controller) to different software pages, our user-level program could reliably corrupt specific bits in pages belonging to other programs. As a result, RowHammer errors can be exploited by a malicious program to breach memory protection and compromise the system. In fact, we hypothesized, in our ISCA 2014 paper, that our user-level program, with some engineering effort, could be developed into a *disturbance attack* that injects errors into other programs, crashes the system, or hijacks control of the system. We left such research for the future since our primary objective in our ISCA 2014 paper was to understand and prevent RowHammer errors [133].

C. Characteristics of RowHammer

Our ISCA 2014 paper [133] provides a detailed experimental analysis of various characteristics of RowHammer, including its prevalence across DRAM chips, access pattern dependence, data pattern dependence, temperature dependence, address correlation between victim and aggressor memory rows, number of bits in a victim row that flip due to RowHammer in an adjacent row, number of rows that get affected due to RowHammer in an adjacent row, relationship of RowHammer-vulnerable cells with leaky cells that need higher refresh rates, repeatability of RowHammer errors, the fact that a memory row is vulnerable to RowHammer on both adjacent wordlines, and real system demonstration of RowHammer. We omit these analyses here in this retrospective and focus on security vulnerabilities and prevention of RowHammer. We refer the reader to [133] for a rigorous treatment of the characteristics of the RowHammer phenomenon.

One of the key takeaways from our characterization is that RowHammer-induced errors are predictably repeatable. In other words, if a cell's value gets corrupted via RowHammer, the same cell's value is very likely to get corrupted again via RowHammer. This repeatability enables the construction of repeatable security attacks in a controlled manner, which we briefly discuss next and cover in detail in Section III-A.

D. RowHammer as a Security Threat

RowHammer exposes a *security threat* since it leads to a breach of memory isolation, where accesses to one row (e.g., a user-level memory page) modifies the data stored in another memory row (e.g., a privileged operating system page). As indicated above, malicious software can be written to take advantage of these disturbance errors. We call these *disturbance attacks* [133], or *RowHammer attacks*. Such attacks can be used to corrupt system memory, crash a system, or take over the entire system. Confirming the predictions of our ISCA paper [133], researchers from Google Project Zero developed a user-level attack that exploits RowHammer to gain kernel privileges and thus take over an entire system [215, 216]. More recently, researchers showed that RowHammer can be exploited in various ways to take over various classes of systems. As such, the RowHammer problem has widespread and profound real implications on system security, threatening the foundations of memory isolation on top of which modern system security principles are built. We survey the works that exploit RowHammer to build many different security attacks in Section III-A.

E. RowHammer Solutions

Our ISCA 2014 paper discusses and analyzes seven different countermeasures to the RowHammer problem. Each solution makes a different trade-off between feasibility, cost, performance, power, and reliability. Among them, we believe our seventh and last solution, called PARA, to be the most efficient with the lowest overhead.

The first six solutions are: 1) manufacturing better DRAM chips that are not vulnerable, 2) using (strong) error correcting codes (ECC) to correct RowHammer-induced errors, 3) increasing the refresh rate for all of memory, 4) statically remapping/retiring RowHammer-prone cells via a one-time post-manufacturing analysis, 5) dynamically remapping/retiring RowHammer-prone cells during system operation, 6) accurately identifying hammered rows during runtime and refresh-

ing their neighbors.[3] We will not go into significant detail in this summary and retrospective, but none of these first six solutions are very desirable as they come at significant power, performance or cost overheads, as we describe in our original work [133]. We will revisit some of these solutions in Section III-B of this article, when we survey related work that builds on RowHammer.

Our ISCA 2014 paper's main proposal to prevent RowHammer is a low-overhead mechanism called *PARA* (*probabilistic adjacent row activation*). The key idea of PARA is simple: every time a row is opened and closed, one or more of its adjacent rows are also opened (i.e., refreshed) with some low probability p (by the memory controller or the DRAM chip). If one particular row happens to be opened and closed repeatedly, then it is statistically certain that the row's adjacent rows will eventually be opened as well, as we show in our original work, assuming p is chosen intelligently and carefully. The main advantages of PARA are that 1) it is *stateless* in the sense that it does *not* require expensive hardware data-structures to count the number of times that rows have been opened or to store the addresses of the aggressor/victim rows, 2) its performance and power consumption overheads are very low due to the infrequent activation of *only adjacent rows* of a closed row. Our ISCA 2014 paper provides a memory-controller-based implementation of PARA, evaluates its reliability guarantee against adversarial access patterns, and empirically examines its performance overhead. We show that by setting the probability of refresh of adjacent rows p to a reasonable yet very low value (e.g., 0.001 or 0.005), PARA provides a strong guarantee against RowHammer and leads to a very small performance overhead of less than 0.75%. More detailed discussion of the implementation and evaluation of PARA can be found in our original work. We will revisit PARA in Section III-B of this article.

III. Survey of Works That Build on RowHammer

RowHammer has spurred a significant amount of research since its publication in 2014. In this section, we provide a categorical survey of the array of works that build off of our original paper that introduces the concept of RowHammer and *disturbance attacks* [133]. We describe seven different types of works: (1) security attacks that exploit the RowHammer vulnerability, (2) defense and mitigation mechanisms against the RowHammer phenomenon and the security attacks, (3) circuit-level studies that aim to understand and model the RowHammer phenomenon, (4) other works that exploit RowHammer for various purposes, (5) works that build platforms to study RowHammer, (6) pop culture references to RowHammer, and (7) works that show that the RowHammer phenomenon continues to exist in future generation DRAM chips younger than the ones we examined in our original ISCA 2014 paper. While we describe the works, we also point out the potential for future research in each topic area.

[3]Several early patent applications propose to maintain an array of counters ("detection logic") in either the memory controller [25, 27, 88] or in the DRAM chips themselves [26, 28, 87]. If the counters are tagged with the addresses of only the most recently activated rows, the number of required counters can be significantly reduced [88].

A. Exploits using RowHammer

Inspired by our ISCA 2014 paper's fundamental findings, researchers from Google Project Zero demonstrated in 2015 that RowHammer can be effectively exploited by user-level programs to gain kernel privileges on real systems [215, 216]. Google Project Zero presented two exploits using RowHammer. The first exploit runs as a Native Client (NaCl) program and escalates privilege to escape from the x86-64 sandbox environment. Since NaCl statically validates code before running it, Google Project Zero simply shows that an attacker can modify safe instructions to become unsafe. The second exploit, which is even more powerful, runs as a normal x86-64 process on Linux and escalates privilege to gain access to all of physical memory and thus take over the entire system. The attacker hammers a page table entry (PTE) such that it changes the PTE to point to a page table owned by the attacking process. This gives the attacking process full read-write access to its own page table and hence to all of physical memory, which enables the attacking process to take over the entire system.

Tens of other works since then demonstrated other attacks exploiting RowHammer and we explain several of them in some detail here. One involves the takeover of a victim virtual machine (VM) by another attacker VM running on the same system [207]. In [207], the attacker VM writes a memory page that it knows exists in the victim VM at a RowHammer-vulnerable memory location. If memory deduplication merges the victim VM's and attacker VM's duplicate pages to the attacker VM page's location, the attacker can then induce RowHammer failures in the deduplicated page's data, which is shared by both the attacker and the victim. Since RowHammer attacks modify memory without writes, the deduplication engine does not detect the modification to memory, and the victim VM continues to use the corrupted page. The authors show two attacks using this method. The first attack compromises OpenSSH [10] by modifying the public keys in a victim VM such that the attacker can easily generate a private key that matches the modified public key. It is easier to generate a private key when a public key becomes easily factorable. The second attack compromises the Linux package installation tool, *apt-get* [8] using two steps. First, the attacker flips a bit in the apt-get domain name of the victim, such that the victim's apt-get requests are redirected to a malicious repository. Second, the attacker flips a bit in the page containing the Ubuntu Archive Signing Keys, which are used to check the validity of packages before installation. Thus, this work exploits the RowHammer vulnerability to break both OpenSSH public key authentication and install malicious software via widely-used installation tools.

The Drammer work [241] demonstrates an attack that exploits RowHammer on a mobile device using a malicious user-level application that requires no permissions. This is the first demonstration of RowHammer attacks on ARM-

based systems. The work takes advantage of the deterministic memory allocation patterns in the Android Linux Operating System. By exploiting these deterministic memory allocation patterns, the authors present a methodology for forcing a victim process to allocate its page table entry in a RowHammer-vulnerable region of memory. To do this, the attacker process must essentially allocate all possible memory regions for a page table allocation and then release the page table allocation that contains the RowHammer-vulnerable DRAM cells at bit offsets that enable exploitation. Because of the use of Buddy Allocation [135] (an allocation scheme that forces allocations to the smallest available contiguous region of memory) in Linux platforms, the attacker does not need to allocate all of memory and risk crashing the system. The researchers found 18 out of 27 phone models to be vulnerable to RowHammer and have since released a mobile application that tests memory for RowHammer-vulnerable cells and aggregates statistics on how widespread the RowHammer phenomenon is on mobile devices. This work shows that existing mobile systems are widely vulnerable to RowHammer attacks.

[90] demonstrates a remote takeover of a server vulnerable to RowHammer via JavaScript code execution. Since JavaScript is present and enabled by default in every modern browser, this work demonstrates the proof-of-concept that the RowHammer attack can be launched by a website to gain root privileges on a system that visits the website.

Other works that have demonstrated attacks exploiting RowHammer include takeover of a mobile system by triggering RowHammer using the WebGL interface on a mobile GPU [76, 89], takeover of a remote system by triggering RowHammer through the Remote Direct Memory Access (RDMA) protocol [158, 235], and various other attacks [12, 31–33, 37, 54, 66, 73, 89, 108, 198, 199, 201, 236, 251, 260]. Thus, RowHammer has widespread and profound real implications on system security, as it breaks memory isolation on top of which modern system security principles are built.

Our work has inspired many researchers to exploit RowHammer to devise new attacks. As mentioned earlier, tens of papers were written in top security venues that demonstrate various practical attacks exploiting RowHammer (e.g., [12, 31–33, 37, 54, 66, 73, 76, 89, 90, 108, 158, 198, 199, 201, 207, 215, 216, 235, 236, 241, 251, 260]). These attacks started with Google Project Zero's first work in 2015 [215, 216] and they continue to this date, with the latest ones that we know of being published in late 2018 [31, 33, 54, 158, 199, 235, 236, 260] and mid 2019 [66]. We believe there is a lot more to come in this direction: as systems security researchers understand more about RowHammer, and as the RowHammer phenomenon continues to fundamentally affect memory chips due to technology scaling problems [176], researchers and practitioners will develop different types of attacks to exploit RowHammer in various contexts and in many more creative ways. Various recent reports suggest that new-generation DDR4 DRAM and other DRAM chips are vulnerable to RowHammer [12, 17, 66, 141, 198], as we examine further in Section III-G, so the fundamental security research on RowHammer is likely to continue into the future.

B. Defenses against RowHammer

Our work also inspired many solution and mitigation techniques for RowHammer from both researchers and industry practitioners. *Apple* publicly mentioned, in their critical security release for RowHammer, that they increased the memory refresh rates due to the "original research by Yoongu Kim et al. (2014)" [21]. The industry-standard *Memtest86* program, which is used to test deployed memory chips for errors, was updated, including a RowHammer test, acknowledging our ISCA 2014 paper [194]. Many academic works developed solutions to RowHammer, working from our original research (e.g., [23, 38, 40, 81, 106, 120, 152, 190, 222, 226, 242]). Additionally, many patents for solutions to RowHammer have been filed [25–28, 30, 88]. We believe such solutions will continue to be generated in both academia and industry, extending RowHammer's impact into the very long term. We cover some of these solutions in this section.

Given that RowHammer is such a critical vulnerability, it is important to find both *immediate* and *long-term* solutions to the RowHammer problem (as well as related problems that might cause similar vulnerabilities). The goal of the immediate solutions is to ensure that existing systems are patched such that the vulnerable DRAM devices that are already in the field cannot be exploited. The goal of the long-term solutions is to ensure that future DRAM devices do not suffer from the RowHammer problem when they are released into the field.

Given that immediate solutions require mechanisms that already exist in systems operating in the field, they are fundamentally more limited. A popular immediate solution, described and analyzed by our ISCA 2014 paper [133], is to increase the refresh rate of memory such that the probability of inducing a RowHammer error before DRAM cells get refreshed is reduced. Several major system manufacturers (including Apple, HP, Cisco, Lenovo, and IBM) have adopted this solution and released security patches that increased DRAM refresh rates (e.g., [21, 75, 100, 154]) in the memory controllers. While this solution might be practical and effective in reducing the vulnerability, it has the significant drawbacks of increasing energy/power consumption, reducing system performance, and degrading quality of service experienced by user programs. Our paper shows that the refresh rate needs to be increased by 7.8X its nominal value today, if we want to eliminate *all* RowHammer-induced errors we saw in our tests of 129 DRAM modules! Figure 2 demonstrates this study: if we examine the most RowHammer-vulnerable module that we test from each manufacturer A, B, C, we find that completely eliminating the RowHammer-induced errors requires us to reduce the refresh interval from the nominal 64ms to 8.2ms, leading to a 7.8X increase in the refresh rate. Since DRAM refresh is already a significant burden [59, 113, 115, 160, 196, 203] on energy consumption, performance, and quality of service, increasing it by any significant amount would only exacerbate the problem. Yet, increased refresh rate is likely the most practical *immediate*

solution to RowHammer that does not require any significant change to the system.

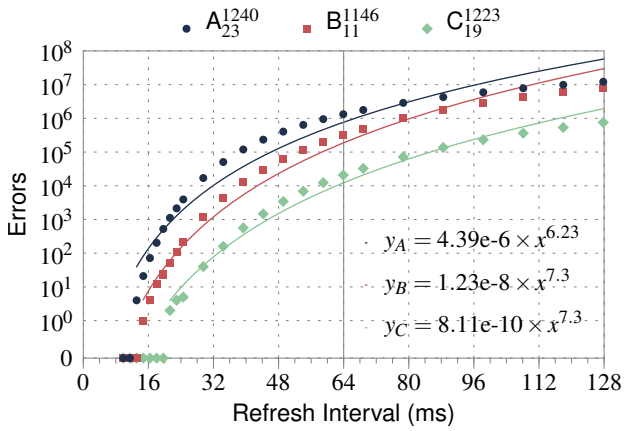


Fig. 2: Number of RowHammer-induced errors observed on the most RowHammer-vulnerable module of each DRAM manufacturer A, B, C, as the refresh interval is varied from 8ms to 128ms (reproduced from [133]).

Other immediate solutions modify the software [38, 106, 107, 137, 190, 215, 242, 250]. For example, ANVIL proposes software-based detection of RowHammer attacks by monitoring via hardware performance counters and selective explicit refreshing of victim rows that are found to be under attack [23]. One short-term approach to mitigating RowHammer attacks is intelligently allocating and physically isolating pages such that RowHammer cannot affect important pages [38, 137, 242]. [38] extends the physical memory allocator of the Operating System to allocate memory in such a way that isolates memory pages of different system entities. [242] prevents DMA-based attacks by isolating DMA buffers with additional buffer rows (i.e., guard rows) that do not store data. This ensures that any DMA-based attack can only induce RowHammer bit flips in the guard rows without affecting rows containing important data. Another approach for mitigating RowHammer statically analyzes code to identify code segments that are probably RowHammer attacks and prevents them prior to execution [106]. ZebRAM [137] reserves odd rows as "safe" rows and even rows as "unsafe" rows such that hammering a safe row should never result in a RowHammer failure in another safe row. The unsafe rows are used as swap space and a portion of safe rows are used as a cache for data in the unsafe-row swap space. Whenever data in unsafe rows is migrated to safe rows, ZebRAM performs software integrity checks and error correction. Unfortunately, such software-based solutions usually 1) require modifications to system software, 2) might be intrusive to system operation, and/or 3) might cause significant performance or memory space overheads (yet they are still promising to research).

As briefly discussed in Section II-E, our ISCA 2014 paper [133] discusses and analyzes seven short-term and long-term countermeasures to the RowHammer problem. The first six solutions are: 1) making better DRAM chips that are not vulnerable, 2) using (strong) error correcting codes (ECC) to correct RowHammer-induced errors, 3) increasing the refresh rate for all of memory, 4) statically remapping/retiring RowHammer-prone cells via a one-time post-manufacturing analysis, 5) dynamically remapping/retiring RowHammer-prone cells during system operation, 6) accurately identifying hammered rows during runtime and refreshing their neighbors. Our work shows that the first six solutions are not very desirable as they come at significant power, performance or cost overheads. We already discussed the overheads of increasing the refresh rates across the board. Similarly, the use of simple SECDED (single-error correcting double-error detecting) error correcting codes (ECC), as employed in many server and datacenter systems, is *not* enough to prevent all RowHammer errors, as some cache blocks experience two or more bit flips, which are not correctable by SECDED ECC, as we have shown in our ISCA 2014 paper [133]. Table I demonstrates this by showing how many 64-bit words in the full address-space (0–2GB) of the most RowHammer-vulnerable DRAM modules of the three major DRAM manufacturers contain 1, 2, 3, or 4 victim cells. While most words have just a single victim, there are also some words with multiple victims. Thus, stronger ECC is very likely required to correct RowHammer errors, which comes at the cost of additional energy, performance, cost, and DRAM capacity overheads.[4] Alternatively, the sixth solution described above, i.e., accurately identifying a row as a hammered row requires keeping track of access counters for a large number of rows in the memory controller [120], potentially leading to very large hardware area and power consumption, and performance, overheads.

Module	*Number of 64-bit words with X errors*			
	$X = 1$	$X = 2$	$X = 3$	$X = 4$
A_{23}	9,709,721	**181856**	**2248**	**18**
B_{11}	2,632,280	**13638**	**47**	0
C_{19}	141,821	**42**	0	0

TABLE I: Uncorrectable multi-bit RowHammer errors (in bold) observed on the most RowHammer-vulnerable module of each DRAM manufacturer A, B, C (reproduced from [133])

There are many other works that propose long-term solutions [28, 30, 40, 68, 81, 82, 88, 111, 128, 134, 152, 213, 222, 226, 243, 245], building on our original work. [226] uses a probabilistic mechanism similar to PARA in the original RowHammer paper in addition to a small stack for maintaining access history information to determine whether adjacent rows need to be refreshed to avoid bit flips. [152, 222, 243, 245] are counter-based defenses that rely on maintaining access counts

[4]Note that protecting *all* memory rows with strong ECC is likely a wasteful solution for RowHammer because RowHammer-induced bit-flips are access-pattern dependent; they are not randomly-occurring bit-flips. Since only a small number of rows can be hammered at a given time, paying the capacity, cost, and energy overheads of extra redundancy required for strong ECC for *all* memory rows, solely to protect them against RowHammer, is likely not an efficient solution to the RowHammer problem.

to DRAM rows and refreshing adjacent rows when the access count of a row exceeds a pre-determined threshold. These works focus on reducing the overhead of counting accesses to DRAM addresses to enable a viable implementation of the sixth soution we described above.

We believe the long-term solution to RowHammer can actually be very simple and low cost: when the memory controller closes a row (after it was activated), it, with a very low probability, refreshes the adjacent rows. The probability value is a parameter determined by the system designer or provided programmatically, if needed, to trade off between performance overhead and vulnerability protection guarantees. We show that this probabilistic solution, called *PARA (Probabilistic Adjacent Row Activation)*, is extremely effective: it eliminates the RowHammer vulnerability, providing much higher reliability guarantees than modern hard disks today, while requiring no storage cost and having negligible performance and energy overheads [133].

PARA is *not* immediately implementable in an existing system because it requires changes to either the memory controllers or the DRAM chips, depending on where it is implemented. If PARA is implemented in the memory controller, the memory controller needs to obtain information on which rows are adjacent to each other in a DRAM bank. This information is currently *unknown* to the memory controller as DRAM manufacturers can internally remap rows to other locations [116, 117, 133, 151, 161] for various reasons, including for tolerating various types of faults. However, this information can be simply provided by the DRAM chip to the memory controller using the serial presence detect (SPD) read-only memory present in modern DRAM modules, as described in our ISCA 2014 paper [133]. It appears that some very recent Intel memory controllers implement a limited variant of PARA, whose adjacent-row activation probability can be chosen by the user via modifications in the BIOS [5]. If PARA is implemented in the DRAM chip, then the hardware interface to the DRAM chip should be such that it allows DRAM-internal refresh operations that are not initiated by an external memory controller. This could be achieved with the addition of a new DRAM command, like the *targeted refresh* command proposed in a patent by Intel [29]. In 3D-stacked memory technologies [130, 150], e.g., HBM (High Bandwidth Memory) [109, 150] or HMC (Hybrid Memory Cube) [7], which combine logic and memory in a tightly integrated fashion, the logic layer can be easily modified to implement PARA.[5] Alternatively, if the memory interface is asynchronous with the processor, a simple controller that is tightly coupled with the memory chip can freely and easily implement PARA internally to the memory chip.

All these implementations of the promising PARA solution are examples of much better cooperation between memory controller and the DRAM chips. Regardless of the exact implementation, we believe RowHammer, and other upcoming reliability vulnerabilities like RowHammer, can be much more easily found, mitigated, and prevented with better cooperation between and co-design of system and memory, i.e., *system-memory co-design* [175]. System-memory co-design is explored by recent works for mitigating various DRAM-based security and DRAM scaling issues, including retention failures and performance problems (e.g., [56, 57, 59, 97, 113–117, 125–127, 129, 133, 146, 147, 149, 151, 160, 161, 165, 175, 183, 203, 205, 217–219, 231–233, 237, 238, 252]). Taking the system-memory co-design approach further, providing more intelligence and configurability/programmability/patchability in the memory controller can greatly ease the tolerance to errors like RowHammer: when a new failure mechanism in memory is discovered, the memory controller can be configured/programmed/patched to execute specialized functions to profile and correct for such mechanisms. We believe this direction is very promising, and several works have explored *online profiling* mechanisms for fixing retention errors [115–118, 196, 203], reducing latency [151], and reducing energy consumption [58]. These works provide examples of how an intelligent memory controller can alleviate the retention failures, and thus the DRAM refresh problem [160, 161], as well as the DRAM latency problem [146, 147].

C. Circuit-level Studies of RowHammer

A very recent work [254] presents evidence via 3D CAD simulations with single charge traps, that the RowHammer effect is governed by the charge pumping process. The RowHammer effect is exacerbated when charge is captured around an aggressor wordline and carriers migrate to victim wordlines. The authors also find that feature size scaling aggravates the RowHammer effect, which could make it more difficult to mitigate in future DRAM generations.

[257] provides a study of the effects of irradiating DRAM on the RowHammer phenomenon, with two major findings. First, the study finds that irradiating DRAM with gamma rays increases the number of DRAM rows that are vulnerable to RowHammer. Second, the authors correlate the cells that are vulnerable to RowHammer with those that have low data retention times and they find almost no correlation, corroborating the results of our ISCA 2014 paper.

[157] also irradiates DRAM with gamma rays, which results in cells with lower data retention times and cells with a higher susceptibility to RowHammer failures. The authors then perform *temperature annealing* (i.e., a method for baking DRAM at a high temperature to "repair" retention-weak cells) on the DRAM devices and find that cells that experience a higher susceptibility to RowHammer after irradiation maintain the

[5] Alternatively, for a solution like PARA to be implemented in the DRAM chip, without modifying the hardware interface to the DRAM chip, one can exploit the timing slack in the DRAM timing parameters that already exist under various conditions. For example, the timing slack in the specified precharge timing parameter or the refresh latency parameter can be exploited by the DRAM chip itself to internally issue refresh operations to targeted rows with some probability. Even though such timing slack exists in DRAM chips, as shown by many recent experimental studies [57, 69, 125, 147, 151], we do not believe this is a robust solution since 1) the timing slack may not exist under all operating conditions or for all chips, 2) many studies would like to reduce the timing slack as much as possible to improve DRAM performance and energy [57, 69, 125, 147, 151].

higher susceptibility to RowHammer even after temperature annealing.

[210] presents evidence that hydrogen (H2) annealing of cell-transistors during the dry etch process shows a reduction in interface trap density. Since the RowHammer failure is mainly caused by the traps in the interface (according to the authors' hypotheses), the authors show that this technique can help to improve DRAM reliability against crosstalk and thus alleviate RowHammer attacks.

[192] and [193] experimentally test DDR3 devices for RowHammer susceptibility, show statistical distributions of RowHammer failures across many devices, and present evidence that the root cause of the RowHammer phenomenon is charge recombination of the victim cell with electrons from the current channels between neighboring cells and their corresponding bitlines.

D. Other Works Exploiting RowHammer

There are other papers that build upon RowHammer but do not necessarily show a RowHammer attack or defense. One work shows that the RowHammer phenomenon can be used as a security primitive. [212] shows that RowHammer can be used as an effective Physical Unclonable Function (PUF), a function that generates unique identifiers (i.e., fingerprints) of a device based on the unique properties of the device due to manufacturing variation. The authors experimentally show that by reserving a region of memory and inducing RowHammer failures in each of the rows of the region, they can generate bit flips in the region whose locations are unique to the device and can be used to identify the device. A more recent work [258] presents an attack on the RowHammer-based PUF [212] by effectively showing that hammering on rows surrounding the region reserved by the RowHammer-based PUF causes the rows at the edges of the reserved DRAM region to have an increased number of bit flips. This results in a modification of the resulting fingerprint, which then results in an unidentifiable device.

E. Platforms for Studying RowHammer

Many prior works present ways to make studying RowHammer easier. [74] describes their Raspberry Pi Operating System for exploring memory concepts simply due to a direct linear mapping between virtual addressses to physical addresses. This mitigates the difficulty of determining which DRAM rows are physically adjacent. SoftMC [98, 225] is an FPGA-based memory controller implementation that enables testing custom DRAM timing parameter values with direct access to DRAM physical addresses. Drammer [70, 71] is an open-source Android application that tests mobile devices for vulnerability to the RowHammer exploit and gathers data from users to determine how widespread the RowHammer vulnerability is across many generations of mobile devices. MemTest86 [194] is software that tests DRAM for many types of reliability issues. As described above, after our original ISCA 2014 paper, MemTest86 developers added RowHammer testing to their suite, which enables users to test their system for the RowHammer vulnerability. [31, 107, 190] provide methods for reverse engineering DRAM address mapping such that attackers can determine the two rows that surround a victim row and hammer the victim row more effectively for causing RowHammer failures. [244] provides an algorithm for determining the eviction set of cache lines in linear time such that an attacker can maximize accesses to DRAM even when caching is unavoidable. [17] repurposes a DDR protocol analyzer with a DIMM interposer to count the activations to each row within a 64 ms interval to detect whether RowHammer occurs in any application.

F. Media References to RowHammer

Our ISCA 2014 work also turned RowHammer into a popular phenomenon (e.g., [1, 2, 16, 39, 65, 83–85, 94, 105, 119, 139, 141, 187–189, 194, 211, 216, 239, 240, 246]), which, in turn, has helped make hardware security even more "mainstream" in popular media and the broader security community. It showed that hardware reliability problems can be very serious security threats that have to be defended against. A well-read article from the Wired magazine, all about RowHammer, is entitled "Forget Software – Now Hackers are Exploiting Physics!" [86], indicating the shift of mindset towards very low-level hardware security vulnerabilities in the popular mainstream security community. Many other popular articles in press have been written about RowHammer, many of which pointing to our ISCA 2014 work [133] as the first demonstration and scientific analysis of the RowHammer problem. Showing that hardware reliability problems can be serious security threats and pulling them to the popular discussion space, and thus influencing the mainstream discourse, creates a very long term impact for the RowHammer problem and thus our original ISCA 2014 paper.

G. Persistence of RowHammer Failures in Modern DRAM

Unfortunately, despite the many proposals in industry and academia to fix the RowHammer issue, RowHammer failures still seem to be observable in state-of-the-art DRAM devices in a variety of generations and standards (e.g., DDR4 [12, 17, 141, 198], ECC DRAM [66], LPDDR3 and LPDDR2 DRAM [241]). This persisting phenomenon suggests that the security vulnerabilities might continue in the current generation of DRAM chips as well. As such, it is critical to continue to investigate solutions to the RowHammer vulnerability.

H. RowHammer in a Broader Context

Springing off from the stir created by RowHammer, we take a step back and argue that there is little that is surprising about the fact that we are seeing disturbance errors in the heavily-scaled DRAM chips of today. Disturbance errors are a general class of reliability problems that is present in not only DRAM, but also other memory and storage technologies. All scaled memory technologies, including SRAM [62, 93, 121], flash [42, 45, 46, 50–53, 67, 167, 168, 171, 214], and hard disk drives [110, 234, 249], exhibit such disturbance problems. In

fact, two of our works experimentally examine read disturb errors in flash memory: 1) our original work in DATE 2012 [42] that provides a rigorous experimental study of error patterns in modern MLC NAND flash memory chips demonstrates the importance of read disturb error patterns, 2) our recent work at DSN 2015 [51] experimentally characterizes the read disturb errors in flash memory, shows that the problem is widespread in recent flash memory chips, and develops mechanisms to correct such errors in the flash memory controller. Even though the mechanisms that cause the bit flips are different in different technologies, the high-level root cause of the problem, *cell-to-cell interference*, due to the fact that the memory cells are too close to each other, is a fundamental issue that appears and will likely continue to appear in any technology that scales down to small enough technology nodes [53, 254]. Thus, we should expect such problems to continue as we scale any memory technology, including emerging ones, to higher densities.

What sets DRAM disturbance errors apart from other technologies' disturbance errors is that 1) DRAM is exposed to the user-level programs and manipulated directly by a program's load and store instructions (which we do not anticipate to change any time soon, since direct data manipulation in main memory is a fundamental component of programming languages and systems), and 2) in modern DRAM, as opposed to other technologies, strong error correction mechanisms are *not* commonly employed (either in the memory controller or the memory chip). The success of DRAM scaling until recently has *not* relied on a memory controller that corrects errors (other than performing periodic refresh and more recently employing very simple single-error correcting codes [9, 113, 185, 186, 191, 195]). Instead, DRAM chips were implicitly assumed to be error-free and did *not* require the help of the controller to operate correctly. Thus, such errors were perhaps not as easily anticipated and corrected within the context of DRAM. In contrast, the success of other technologies, e.g., flash memory and hard disks, has heavily relied on the existence of an intelligent controller that plays a key role in correcting errors and making up for reliability problems of the memory chips themselves [52]. This has not only enabled the correct operation of assumed-faulty memory chips but also enabled a mindset where the controllers are co-designed with the chips themselves, covering up the memory technology's deficiencies and hence perhaps enabling better anticipation of errors with technology scaling. This approach is very prominent in modern SSDs (solid state drives), for example, where the flash memory controller employs a wide variety of error mitigation and correction mechanisms [42–48, 50–53, 166], including not only sophisticated strong ECC mechanisms but also targeted voltage optimization, retention mitigation and disturbance mitigation techniques. We believe changing the mindset in modern DRAM to a similar mindset of *assumed-faulty memory chip and an intelligent memory controller that makes it operate correctly* can not only enable better anticipation and correction of future issues like RowHammer but also better scaling of DRAM into future technology nodes [175].

IV. Ongoing and Future Work

We believe there is a lot more research to come that will build on RowHammer, from at least three perspectives: 1) the security attack perspective, 2) the defense/mitigation perspective, 3) a broader understanding, modeling, and prevention perspective.

As systems security researchers understand more about RowHammer, and as the RowHammer phenomenon continues to fundamentally affect memory chips due to technology scaling problems [176], researchers and practitioners will develop different types of attacks to exploit RowHammer in various contexts and in many more creative ways. RowHammer is a critical problem that manifests in the difficulties in DRAM scaling and is expected to only become worse in the future [177, 182, 183]. As we discussed, some recent reports suggest that new-generation DRAM chips are vulnerable to RowHammer (e.g., DDR4 [12, 17, 141, 198], ECC [66, 133], LPDDR3 and LPDDR2 [241]). This indicates that effectively mitigating the RowHammer problem with low overhead is difficult and becomes more difficult as process technology scales further. Even with the wide array of works that build on top of RowHammer, we believe that these papers have yet to scratch the surface of this field of reliability and security, especially as manufacturing technology scaling continues in all technologies. It is critical to deeply understand the underlying factors of the RowHammer problem (and more generally the crosstalk problem) such that we can effectively prevent these issues across all technologies with minimal overhead. As DRAM cells become even smaller and less reliable, it is likely for them to become even more vulnerable to complicated and different modes of failure that are sensitized only under specific access-patterns and/or data-patterns. As a scalable solution for the future, our ISCA 2014 paper argues for adopting a system-level approach [175] to DRAM reliability and security, in which the DRAM chips, the memory controller, and perhaps the operating system collaborate together to diagnose/treat emerging DRAM failure modes.

We believe that more and more researchers will focus on providing security in all aspects of computing so that such hardware faults that are exposed to the software (and thus the public) are minimized. RowHammer enabled a shift of mindset among mainstream security researchers: general-purpose hardware is fallible (in a very widespread manner) and its problems are actually exploitable. This shift of mindset enabled many systems security researchers to examine hardware in more depth and understand its inner workings and vulnerabilities better. We believe it is no coincidence that two of the groups that concurrently discovered the heavily-publicized Meltdown [159] and Spectre [136] vulnerabilities (Google Project Zero and TU Graz InfoSec) have heavily worked on RowHammer attacks before. We believe this shift in mindset, enabled in good part by the existence and prevalence of RowHammer, will continue to be very be important for discovering and solving other potential vulnerabilities that may rise as a result of both technology scaling and hardware design.

A. Other Potential Vulnerabilities

We believe that, as memory technologies scale to higher densities, other problems may start appearing (or may already be going unnoticed) that can potentially threaten the foundations of secure systems. There have been recent large-scale field studies of memory errors showing that both DRAM and NAND flash memory technologies are becoming less reliable [42, 50, 52, 53, 167, 168, 171, 172, 175, 176, 183, 196, 214, 227–229]. As detailed experimental analyses of real DRAM and NAND flash chips show, both technologies are becoming much more vulnerable to cell-to-cell interference effects [42, 45–48, 51–53, 133, 166, 175, 176, 178, 183], data retention is becoming significantly more difficult in both technologies [42–44, 46, 50, 52, 53, 59, 113, 115–117, 160, 161, 164, 167–169, 175, 178, 183, 203], and error variation within and across chip, and across operating conditions, is increasingly prominent [42, 46, 55, 57, 125–127, 147, 151, 161]. Emerging memory technologies [170, 175], such as Phase-Change Memory [142–144, 202, 204, 206, 247, 255, 256, 261], STT-MRAM [61, 138], and RRAM/ReRAM/memristors [248] are likely to exhibit similar and perhaps even more exacerbated reliability issues. We believe, if not carefully accounted for and corrected, these reliability problems may surface as security problems as well, as in the case of RowHammer, especially if the technology is employed as part of the main memory system that is directly exposed to user-level programs.

We briefly examine two example potential vulnerabilities. We believe future work examining these vulnerabilities, among others, are promising for both fixing the vulnerabilities and enabling the effective scaling of memory technology.

1) Data Retention Failures: Data retention is a fundamental reliability problem, and hence a potential vulnerability, in especially charge-based memories like DRAM and flash memory. This is because charge leaks out of the charge storage unit (e.g., the DRAM capacitor or the NAND flash floating gate) over time. As such memories become denser, three major trends make data retention more difficult [50, 113, 160, 161]. First, the number of memory cells increases, leading to the need for more refresh operations to maintain data correctly. Second, the charge storage unit (e.g., the DRAM capacitor) becomes smaller and/or morphs in structure, leading to potentially lower retention times. Third, the voltage margins that separate one data value from another become smaller (e.g., the same voltage window gets divided into more "states" in NAND flash memory, to store more bits per cell), and, as a result, the same amount of charge loss is more likely to cause a bit error in a smaller technology node than in a larger one.

DRAM Data Retention Issues

Data retention issues in DRAM are a fundamental scaling limiter of the DRAM technology [113, 161, 175]. We have shown, in recent works based on rigorous experimental analyses of modern DRAM chips [115, 117, 118, 161, 196, 203], that determining the minimum retention time of a DRAM cell is getting significantly more difficult. Thus, determining the correct rate at which to refresh DRAM cells has become more difficult, as also indicated by industry [113]. This is due to two major phenomena, both of which get worse (i.e., become more prominent) with technology scaling. First, Data Pattern Dependence (DPD): the retention time of a DRAM cell is heavily dependent on the data pattern stored in itself and in the neighboring cells [161]. Second, Variable Retention Time (VRT): the retention time of some DRAM cells can change drastically over time, due to a memoryless random process that results in very fast charge loss via a phenomenon called trap-assisted gate-induced drain leakage [161, 209, 253]. These phenomena greatly complicate the accurate determination of minimum data retention time of DRAM cells. In fact, VRT, as far as we know, is very difficult to test for because there seems to be no way of determining that a cell exhibits VRT until that cell is observed to exhibit VRT and the time scale of a cell exhibiting VRT does not seem to be bounded, given the current experimental data [115, 161, 196, 203]. As a result, some retention errors can easily slip into the field because of the difficulty of the retention time testing. Therefore, data retention in DRAM is a vulnerability that can greatly affect both reliability and security of current and future DRAM generations. We encourage future work to investigate this area further, from both reliability and security, *as well as* performance and energy efficiency perspectives. Various works in this area provide insights about the retention time properties of modern DRAM devices based on experimental data [98, 115, 117, 118, 161, 196, 203], develop infrastructures to obtain valuable experimental data [98], and provide potential solutions to the DRAM retention time problem [59, 115–118, 160, 161, 196, 203], all of which the future works can build on.

Note that data retention failures in DRAM are likely to be investigated heavily to ensure good performance and energy efficiency. And, in fact they already are being investigated for this purpose (see, for example, [59, 115–118, 160, 196, 203]). We believe it is important for such investigations to ensure no new vulnerabilities (e.g., side channels) open up due to the solutions developed.

NAND Flash Data Retention Issues

Experimental analysis of modern flash memory devices show that the dominant source of errors in flash memory are data retention errors [42, 52]. As a flash cell wears out, its charge retention capability degrades [42, 50, 52, 53, 167, 168, 171, 214] and the cell becomes leakier. As a result, to maintain the original data stored in the cell, the cell needs to be refreshed [43, 44]. The frequency of refresh increases as wearout of the cell increases. We have shown that performing refresh in an adaptive manner greatly improves the lifetime of modern MLC (multi-level cell) NAND flash memory while causing little energy and performance overheads [43, 44]. Most high-end SSDs today employ such adaptive refresh mechanisms.

As flash memory scales to smaller manufacturing technology nodes and even more bits per cell, data retention becomes a bigger problem. As such, it is critical to understand

the issues with data retention in flash memory. Our recent work provides detailed experimental analysis of data retention behavior of planar and 3D MLC NAND flash memory [50, 52, 53, 167, 168]. We show, among other things, that there is a wide variation in the leakiness of different flash cells: some cells leak very fast, some cells leak very slowly. This variation leads to new opportunities for correctly recovering data from a flash device that has experienced an uncorrectable error: by identifying which cells are fast-leaking and which cells are slow-leaking, one can probabilistically estimate the original values of the cells before the uncorrectable error occurred. This mechanism, called *Retention Failure Recovery*, leads to significant reductions in bit error rate in modern MLC NAND flash memory [50, 52, 53] and is thus very promising. Unfortunately, it also points to a potential security and privacy vulnerability: by analyzing data and cell properties of a failed device, one can potentially recover the original data. We believe such vulnerabilities can become more common in the future and therefore they need to be anticipated, investigated, and understood.

2) Other Vulnerabilities in NAND Flash Memory: We believe other sources of error (e.g., cell-to-cell interference) and cell-to-cell variation in flash memory can also lead various vulnerabilities. For example, another type of variation (that is similar to the variation in cell leakiness that we described above) exists in the vulnerability of flash memory cells to read disturbance [51]: some cells are much more prone to read disturb effects than others. This wide variation among cells enables one to probabilistically estimate the original values of cells in flash memory after an uncorrectable error has occurred. Similarly, one can probabilistically correct the values of cells in a page by knowing the values of cells in the neighboring page [47]. These mechanisms [47, 51] are devised to improve flash memory reliability and lifetime, but the same phenomena that make them effective in doing so can also lead to potential vulnerabilities, which we believe are worthy of investigation to ensure security and privacy of data in flash memories.

As an example, we have recently shown [48] that it is theoretically possible to exploit vulnerabilities in flash memory programming operations on existing solid-state drives (SSDs) to cause (malicious) data corruption. This particular vulnerability is caused by the *two-step programming* method employed in dense flash memory devices, e.g., MLC NAND flash memory. An MLC device partitions the threshold voltage range of a flash cell into four distributions. In order to reduce the number of errors introduced during programming of a cell, flash manufacturers adopt a two-step programming method, where the least significant bit of the cell is partially programmed first to some intermediate threshold voltage, and the most significant bit is programmed later to bring the cell up to its full threshold voltage. We find that two-step programming exposes new vulnerabilities, as both cell-to-cell program interference and read disturbance can disrupt the intermediate value stored within a multi-level cell before the second programming step completes. We show that it is possible to exploit these vulnerabilities on existing solid-state drives (SSDs) to alter the partially-programmed data, causing (malicious) data corruption. We experimentally characterize the extent of these vulnerabilities using contemporary 1X-nm (i.e., 15-19nm) flash chips [48]. Building on our experimental observations, we propose several new mechanisms for MLC NAND flash that eliminate or mitigate disruptions to intermediate values, removing or reducing the extent of the vulnerabilities, mitigating potential exploits, and increasing flash lifetime by 16% [48]. We believe investigation of such vulnerabilities in flash memory will lead to more robust flash memory devices in terms of both reliability and security, as well as performance. In fact, a recent work from IBM builds on our work [48] to devise a security attack at the file system level [140].

B. Prevention

Various reliability problems experienced by scaled memory technologies, if not carefully anticipated, accounted for, and corrected, may surface as security problems as well, as in the case of RowHammer. We believe it is critical to develop principled methods to understand, anticipate, and prevent such vulnerabilities. In particular, principled methods are required for three major steps in the design process.

First, it is critical to understand the potential failure mechanisms and anticipate them beforehand. To this end, developing solid methodologies for failure modeling and prediction is critical. To develop such methodologies, it is essential to have real experimental data from past and present devices. Data available both at the small scale (i.e., data obtained via controlled testing of individual devices, as in, e.g., [42–53, 57, 115, 125–127, 147, 161, 166–168, 195, 196]) as well as at the large scale (i.e., data obtained during in-the-field operation of the devices, under likely-uncontrolled conditions, as in, e.g., [171, 172]) can enable accurate models for failures, which could aid many purposes, including the development of better reliability mechanisms and prediction of problems before they occur.

Second, it is critical to develop principled architectural methods that can avoid, tolerate, or prevent such failure mechanisms that can lead to vulnerabilities. For this, we advocate co-architecting of the system and the memory together, as we described earlier. Designing intelligent, flexible, configurable, programmable, patch-able memory controllers that can understand and correct existing and potential failure mechanisms can greatly alleviate the impact of failure mechanisms on reliability, security, performance, and energy efficiency. A *system-memory co-design* approach can also enable new opportunities, like performing effective processing near or in the memory device (e.g., [13–15, 18, 22, 24, 34–36, 56, 63, 72, 77, 78, 80, 91, 92, 95, 96, 99, 101–103, 112, 122–124, 153, 155, 156, 162, 163, 174, 180, 181, 184, 197, 200, 217–221, 223, 224, 230, 259, 262]). In addition to designing the memory device together with the controller, we believe it is important to investigate mechanisms for good partitioning of duties across the various levels of transformation in computing,

including system software, compilers, and application software.

Third, it is critical to develop principled methods for electronic design, automation and testing, which are in harmony with the failure modeling/prediction and system reliability methods, which we mentioned in the above two paragraphs. Design, automation and testing methods need to provide high and predictable coverage of failures and work in conjunction with architectural and across-stack mechanisms. For example, enabling effective and low-cost *online profiling of DRAM* [115–117, 151, 161, 196, 203] in a principled manner requires cooperation of failure modeling mechanisms, architectural methods, and design, automation and testing methods.

V. Conclusion

We provided a retrospective on the RowHammer problem and our original ISCA 2014 paper [133] that introduced the problem, and a survey of many flourishing works that have built on RowHammer. It is clear that the reliability of memory technologies we greatly depend on is reducing, as these technologies continue to scale to ever smaller technology nodes in pursuit of higher densities. These reliability problems, if not anticipated and corrected, can also open up serious security vulnerabilities, which can be very difficult to defend against, if they are discovered in the field. RowHammer is an example, likely the first one, of a hardware failure mechanism that causes a practical and widespread system security vulnerability. As such, its implications on system security research are tremendous and exciting. We hope the summary, retrospective, and commentary we provide in this paper on the RowHammer phenomenon are useful for understanding the RowHammer problem, its context, mitigation mechanisms, and the large body of work that has built on it in the past five years.

We believe that the need to prevent such reliability and security vulnerabilities at heavily-scaled memory technologies opens up new avenues for principled approaches to 1) understanding, modeling, and prediction of failures and vulnerabilities, and 2) architectural as well as design, automation and testing methods for ensuring reliable and secure operation. We believe the future is very bright for research in reliable and secure memory systems, and many discoveries abound in the exciting yet complex intersection of reliability and security issues in such systems.

Acknowledgments

This paper is based on two previous papers we have written on RowHammer, one that first scientifically introduced and analyzed the phenomenon in ISCA 2014 [133] and the other that provided an analysis and future outlook on RowHammer [176]. The presented work is a result of the research done together with many students and collaborators over the course of the past eight years. In particular, three PhD theses have shaped the understanding that led to this work. These are Yoongu Kim's thesis entitled "Architectural Techniques to Enhance DRAM Scaling" [132], Yu Cai's thesis entitled "NAND Flash Memory: Characterization, Analysis, Modeling and Mechanisms" [49] and his continued follow-on work after his thesis, summarized in [52, 53], and Donghyuk Lee's thesis entitled "Reducing DRAM Latency at Low Cost by Exploiting Heterogeneity" [145]. We also acknowledge various funding agencies (NSF, SRC, ISTC, CyLab) and industrial partners (AliBaba, AMD, Google, Facebook, HP Labs, Huawei, IBM, Intel, Microsoft, Nvidia, Oracle, Qualcomm, Rambus, Samsung, Seagate, VMware) who have supported the presented and other related work in our group generously over the years.

The first version of the talk associated with this paper was delivered at a CMU CyLab Partners Conference in September 2015. Other versions of the talk were delivered as part of an Invited Session at DAC 2016, with a collaborative accompanying paper entitled "Who Is the Major Threat to Tomorrow's Security? You, the Hardware Designer" [41], at DATE 2017 [176], and at the Top Picks in Hardware and Embedded Security workshop, co-located with ICCAD 2018 [11], where RowHammer was selected as a Top Pick among hardware and embedded security papers published between 2012-2017. The most recent version of the associated talk was delivered at COSADE 2019 [179].

References

[1] "RowHammer Discussion Group," https://groups.google.com/forum/#!forum/rowhammer-discuss.
[2] "RowHammer on Twitter," https://twitter.com/search?q=rowhammer.
[3] "Rowhammer: Source Code for Testing the Row Hammer Error Mechanism in DRAM Devices." https://github.com/CMU-SAFARI/rowhammer.
[4] "Test DRAM for Bit Flips Caused by the RowHammer Problem," https://github.com/google/rowhammer-test.
[5] "Tweet about RowHammer Mitigation on x210," https://twitter.com/isislovecruft/status/1021939922754723841.
[6] "RDMA Consortium," http://www.rdmaconsortium.org, 2009.
[7] *Hybrid Memory Consortium*, 2012, http://www.hybridmemorycube.org.
[8] "apt-get Linux man page," https://linux.die.net/man/8/apt-get, 2017.
[9] "ECC Brings Reliability and Power Efficiency to Mobile Devices," Micron Technology inc., Tech. Rep., 2017.
[10] "OpenSSH," https://www.openssh.com/, 2017.
[11] "Top Picks in Hardware and Embedded Security - Workshop Collocated with ICCAD 2018," https://wp.nyu.edu/toppicksinhardwaresecurity/, 2017.
[12] M. T. Aga *et al.*, "When Good Protections go Bad: Exploiting anti-DoS Measures to Accelerate Rowhammer Attacks," in *HOST*, 2017.
[13] S. Aga *et al.*, "Compute caches," in *HPCA*, 2017.
[14] J. Ahn *et al.*, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[15] J. Ahn *et al.*, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.
[16] B. Aichinger, "The Known Failure Mechanism in DDR3 Memory referred to as Row Hammer," http://ddrdetective.com/files/6414/1036/5710/The_Known_Failure_Mechanism_in_DDR3_memory_referred_to_as_Row_Hammer.pdf, September 2014.
[17] B. Aichinger, "DDR Memory Errors Caused by Row Hammer," in *HPEC*, 2015.
[18] B. Akin *et al.*, "Data Reorganization in Memory using 3D-stacked DRAM," in *ISCA*, 2015.
[19] Z. Al-Ars *et al.*, "DRAM-Specific Space of Memory Tests," in *ITC*, 2006.
[20] Z. Al-Ars, "DRAM Fault Analaysis and Test Generation," Ph.D. dissertation, TU Delft, 2005.
[21] Apple Inc., "About the security content of Mac EFI Security Update 2015-001," https://support.apple.com/en-us/HT204934, June 2015.
[22] H. Asghari-Moghaddam *et al.*, "Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems," in *MICRO*, 2016.
[23] Z. B. Aweke *et al.*, "Anvil: Software-based protection against next-generation rowhammer attacks," in *ASPLOS*, 2016.
[24] O. O. Babarinsa and S. Idreos, "JAFAR: Near-data Processing for Databases," in *SIGMOD*, 2015.
[25] K. Bains *et al.*, "Row Hammer Refresh Command," US Patent App. 14/068,677, Feb. 27 2014.
[26] K. Bains *et al.*, "Method, Apparatus and System for Providing a Memory Refresh," US Patent App. 13/625,741, Mar. 27 2014.
[27] K. Bains *et al.*, "Row Hammer Refresh Command," US Patent App. 13/539,415, Jan. 2 2014.
[28] K. Bains and J. Halbert, "Distributed Row Hammer Tracking," US Patent App. 13/631,781, Apr. 3 2014.

[29] K. Bains *et al.*, "Row hammer refresh command," U.S. Patent Number 9117544 B2, 2015.

[30] K. S. Bains and J. B. Halbert, "Row Hammer Monitoring Based on Stored Row Hammer Threshold Value," uS Patent 9,032,141. May 12 2015.

[31] A. Barenghi *et al.*, "Software-only Reverse Engineering of Physical DRAM Mappings for Rowhammer Attacks," in *IVSW*, 2018.

[32] S. Bhattacharya and D. Mukhopadhyay, "Curious Case of RowHammer: Flipping Secret Exponent Bits using Timing Analysis," in *CHES*, 2016.

[33] S. Bhattacharya and D. Mukhopadhyay, "Advanced Fault Attacks in Software: Exploiting the Rowhammer Bug," in *Fault Tolerant Architectures for Cryptography and Hardware Security*, 2018.

[34] A. Boroumand *et al.*, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *IEEE CAL*, 2016.

[35] A. Boroumand *et al.*, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.

[36] A. Boroumand *et al.*, "CoNDA: Enabling Efficient Near-Data Accelerator Communication by Optimizing Data Movement," *ISCA*, 2019.

[37] E. Bosman *et al.*, "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector," *S&P*, 2016.

[38] F. Brasser *et al.*, "Can't Touch This: Practical and Generic Software-only Defenses Against RowHammer Attacks," *USENIX Sec.*, 2017.

[39] S. Brown, "Rowhammer: The Evolution of a New Generation of Attacks," https://cyware.com/news/rowhammer-the-evolution-of-a-new-generation-of-attacks-7baa0a3c, December 2018.

[40] L. Bu *et al.*, "SRASA: a Generalized Theoretical Framework for Security and Reliability Analysis in Computing Systems," *Journal of Hardware and Systems Security*, 2018.

[41] W. Burleson *et al.*, "Who Is the Major Threat to Tomorrow's Security? You, the Hardware Designer," *DAC*, 2016.

[42] Y. Cai *et al.*, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *DATE*, 2012.

[43] Y. Cai *et al.*, "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime," in *ICCD*, 2012.

[44] Y. Cai *et al.*, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," *ITJ*, 2013.

[45] Y. Cai *et al.*, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," in *ICCD*, 2013.

[46] Y. Cai *et al.*, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis and Modeling," in *DATE*, 2013.

[47] Y. Cai *et al.*, "Neighbor-Cell Assisted Error Correction for MLC NAND Flash Memories," in *SIGMETRICS*, 2014.

[48] Y. Cai *et al.*, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *HPCA*, 2017.

[49] Y. Cai, "NAND flash memory: Characterization, Analysis, Modeling and Mechanisms," Ph.D. dissertation, Carnegie Mellon University, 2012.

[50] Y. Cai *et al.*, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization and Recovery," in *HPCA*, 2015.

[51] Y. Cai *et al.*, "Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery," in *DSN*, 2015.

[52] Y. Cai *et al.*, "Error Characterization, Mitigation, and Recovery in Flash-memory-based Solid-state Drives," *Proceedings of the IEEE*, 2017.

[53] Y. Cai *et al.*, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," *arXiv preprint arXiv:1711.11427*, 2017.

[54] S. Carre *et al.*, "OpenSSL Bellcore's Protection Helps Fault Attack," in *DSD*, 2018.

[55] K. Chandrasekar *et al.*, "Exploiting Expendable Process-margins in DRAMs for Run-time Performance Optimization," in *DATE*, 2014.

[56] K. Chang *et al.*, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.

[57] K. Chang *et al.*, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," *SIGMETRICS*, 2016.

[58] K. Chang *et al.*, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," in *SIGMETRICS*, 2017.

[59] K. Chang *et al.*, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," in *HPCA*, 2014.

[60] M.-T. Chao *et al.*, "Fault Models for Embedded-DRAM Macros," in *DAC*, 2009.

[61] E. Chen *et al.*, "Advances and Future Prospects of Spin-Transfer Torque Random Access Memory," *IEEE Transactions on Magnetics*, 2010.

[62] Q. Chen *et al.*, "Modeling and Testing of SRAM for New Failure Mechanisms Due to Process Variations in Nanoscale CMOS," in *VTS*, 2005.

[63] P. Chi *et al.*, "Prime: A Novel Processing-in-memory Architecture for Neural Network Computation in Reram-based Main Memory," in *ISCA*, 2016.

[64] P.-F. Chia *et al.*, "New DRAM HCI Qualification Method Emphasizing on Repeated Memory Access," in *Integrated Reliability Workshop*, 2010.

[65] C. Cimpanu, "Rowhammer Attacks can now Bypass ECC Memory Protections," https://www.zdnet.com/article/rowhammer-attacks-can-now-bypass-ecc-memory-protections/, November 2018.

[66] L. Cojocar *et al.*, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against RowHammer Attacks," in *S&P*, 2019.

[67] J. Cooke, "The Inconvenient Truths of NAND Flash Memory," in *Flash Memory Summit*, 2007.

[68] J.-L. Danger *et al.*, "CCFI-Cache: A Transparent and Flexible Hardware Protection for Code and Control-Flow Integrity," in *DSD*, 2018.

[69] A. Das *et al.*, "VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency," in *DAC*, 2018.

[70] Drammer App Source Code, https://github.com/vusec/drammer-app.

[71] Drammer Source Code, https://github.com/vusec/drammer.

[72] A. Farmahini-Farahani *et al.*, "NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules," in *HPCA*, 2015.

[73] A. P. Fournaris *et al.*, "Exploiting Hardware Vulnerabilities to Attack Embedded System Devices: A Survey of Potent Microarchitectural Attacks," *Electronics*, 2017.

[74] P. Francis-Mezger and V. M. Weaver, "A Raspberry Pi Operating System for Exploring Advanced Memory System Concepts," in *Memsys*, 2018.

[75] T. Fridley and O. Santos, "Mitigations Available for the DRAM Row Hammer Vulnerability," http://blogs.cisco.com/security/mitigations-available-for-the-dram-row-hammer-vulnerability, March 2015.

[76] P. Frigo *et al.*, "Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU," *IEEE S&P*, 2018.

[77] M. Gao *et al.*, "Practical Near-data Processing for in-memory Analytics Frameworks," in *PACT*, 2015.

[78] M. Gao and C. Kozyrakis, "HRL: Efficient and Flexible Reconfigurable Logic for Near-data Processing," in *HPCA*, 2016.

[79] S. Ghose *et al.*, "What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study," in *SIGMETRICS*, 2018.

[80] S. Ghose *et al.*, "The Processing-in-Memory Paradigm: Mechanisms to Enable Adoption," *Beyond-CMOS Technologies for Next Generation Computer Design*, 2019.

[81] H. Gomez *et al.*, "DRAM Row-hammer Attack Reduction using Dummy Cells," in *NORCAS*, 2016.

[82] S.-L. Gong, "Memory Protection Techniques for DRAM Scaling-induced Errors," Ph.D. dissertation, 2018.

[83] D. Goodin, "Cutting-edge hack gives super user status by exploiting DRAM weakness," https://arstechnica.com/information-technology/2015/03/cutting-edge-hack-gives-super-user-status-by-exploiting-dram-weakness/, 2016.

[84] D. Goodin, "Once thought safe, DDR4 memory shown to be vulnerable to Rowhammer," https://arstechnica.com/information-technology/2016/03/once-thought-safe-ddr4-memory-shown-to-be-vulnerable-to-rowhammer/, 2016.

[85] D. Goodin, "Using Rowhammer bitflips to root Android phones is now a thing," https://arstechnica.com/information-technology/2016/10/using-rowhammer-bitflips-to-root-android-phones-is-now-a-thing/, 2016.

[86] A. Greenberg, "Forget Software – Now Hackers are Exploiting Physics," https://www.wired.com/2016/08/new-form-hacking-breaks-ideas-computers-work/, 2016.

[87] Z. Greenfield *et al.*, "Method, Apparatus and System for Determining a Count of Accesses to a Row of Memory," US Patent App. 13/626,479, Mar. 27 2014.

[88] Z. Greenfield *et al.*, "Row Hammer Condition Monitoring," US Patent App. 13/539,417, Jan. 2, 2014.

[89] D. Gruss *et al.*, "Another Flip in the Wall of Rowhammer Defenses," *IEEE S&P*, 2018.

[90] D. Gruss *et al.*, "Rowhammer.js: A remote software-induced fault attack in javascript," *CoRR*, vol. abs/1507.06955, 2015.

[91] B. Gu *et al.*, "Biscuit: A Framework for Near-data Processing of Big Data Workloads," in *ISCA*, 2016.

[92] Q. Guo *et al.*, "3D-stacked Memory-side Acceleration: Accelerator and System Design," in *WoNDP*, 2014.

[93] Z. Guo *et al.*, "Large-Scale SRAM Variability Characterization in 45 nm CMOS," *JSSC*, 2009.

[94] R. Harris, "Flipping DRAM Bits - Maliciously," http://www.zdnet.com/article/flipping-dram-bits-maliciously/, December 2014.

[95] M. Hashemi *et al.*, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller," in *ISCA*, 2016.

[96] M. Hashemi *et al.*, "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads," in *MICRO*, 2016.

[97] H. Hassan *et al.*, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," in *HPCA*, 2016.

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

[99] S. M. Hassan *et al.*, "Near Data Processing: Impact and Optimization of 3D Memory System Architecture on the Uncore," in *Memsys*, 2015.

[100] 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.

[101] K. Hsieh *et al.*, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," *ICCD*, 2016.

[102] K. Hsieh *et al.*, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," *ISCA*, 2016.

[103] K. Hsieh *et al.*, "Accelerating Pointer Chasing in 3D-stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.

[104] R.-F. Huang *et al.*, "Alternate Hammering Test for Application-Specific DRAMs and an Industrial Case Study," in *DAC*, 2012.

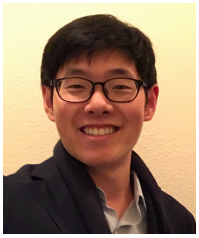
[105] I. Ilascu, "ECC Memory Vulnerable to Rowhammer Attack," https://www.bleepingcomputer.com/news/security/ecc-memory-vulnerable-to-rowhammer-attack/, November 2018.

[106] G. Irazoqui *et al.*, "MASCAT: Stopping Microarchitectural Attacks Before Execution," *IACR Cryptology ePrint Archive*, 2016.
[107] N. Izzo, "Reliably Achieving and Efficiently Preventing Rowhammer Attacks," Ph.D. dissertation, Politecnico Milano, 2017.
[108] Y. Jang *et al.*, "SGX-Bomb: Locking Down the Processor via Rowhammer Attack," in *SysTEX*, 2017.
[109] JEDEC, "JESD235 High Bandwidth Memory (HBM) DRAM," 2013.
[110] W. Jiang *et al.*, "Cross-Track Noise Profile Measurement for Adjacent-Track Interference Study and Write-Current Optimization in Perpendicular Recording," *Journal of Applied Physics*, 2003.
[111] A. K. Jones *et al.*, "Holistic Energy Efficient Crosstalk Mitigation in DRAM," in *IGSC*, 2017.
[112] M. Kang *et al.*, "An Energy-efficient VLSI Architecture for Pattern Recognition via Deep Embedding of Computation in SRAM," in *ICASSP*, 2014.
[113] U. Kang *et al.*, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," in *The Memory Forum*, 2014.
[114] C. Keller *et al.*, "Dynamic Memory-based Physically Unclonable Function for the Generation of Unique Identifiers and True Random Numbers," in *ISCAS*, 2014.
[115] S. Khan *et al.*, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," *SIGMETRICS*, 2014.
[116] S. Khan *et al.*, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," *CAL*, 2016.
[117] S. Khan *et al.*, "PARBOR: An Efficient System-Level Technique to Detect Data-Dependent Failures in DRAM," in *DSN*, 2016.
[118] S. Khan *et al.*, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," *MICRO*, 2017.
[119] S. Khandelwal, "NethammerExploiting DRAM Rowhammer Bug Through Network Requests," https://thehackernews.com/2018/05/remote-rowhammer-attack.html, May 2018.
[120] D.-H. Kim *et al.*, "Architectural Support for Mitigating Row Hammering in DRAM Memories," *IEEE CAL*, 2015.
[121] D. Kim *et al.*, "Variation-Aware Static and Dynamic Writability Analysis for Voltage-Scaled Bit-Interleaved 8-T SRAMs," in *ISLPED*, 2011.
[122] D. Kim *et al.*, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-density 3D Memory," in *ISCA*, 2016.
[123] G. Kim *et al.*, "Toward Standardized Near-data Processing with Unrestricted Data Placement for GPUs," in *SC*, 2017.
[124] J. S. Kim *et al.*, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping using Processing-in-memory Technologies," *BMC Genomics*, 2018.
[125] J. S. Kim *et al.*, "Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines," in *ICCD*, 2018.
[126] J. S. Kim *et al.*, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices," in *HPCA*, 2018.
[127] J. S. Kim *et al.*, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput," in *HPCA*, 2019.
[128] M. Kim *et al.*, "An Effective DRAM Address Remapping for Mitigating Rowhammer Errors," *TC*, 2019.
[129] Y. Kim *et al.*, "A case for subarray-level parallelism (SALP) in DRAM," in *ISCA*, 2012.
[130] Y. Kim *et al.*, "Ramulator: A Fast and Extensible DRAM Simulator," *IEEE CAL*, 2015.
[131] Y. Kim *et al.*, "RowHammer: Reliability Analysis and Security Implications," *ArXiV*, 2016.
[132] Y. Kim, "Architectural Techniques to Enhance DRAM Scaling," Ph.D. dissertation, Carnegie Mellon University, 2015.
[133] Y. Kim *et al.*, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.
[134] D. Kline *et al.*, "Sustainable Fault Management and Error Correction for Next-generation Main Memories," in *IGSC*, 2017.
[135] K. C. Knowlton, "A Fast Storage Allocator," *Communications of the ACM*, 1965.
[136] P. Kocher *et al.*, "Spectre Attacks: Exploiting Speculative Execution," *S&P*, 2018.
[137] R. K. Konoth *et al.*, "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks," in *OSDI*, 2018.
[138] E. Kultursay *et al.*, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *ISPASS*, 2013.
[139] M. Kumar, "New Rowhammer Attack Can Hijack Computers Remotely Over the Network," https://thehackernews.com/2018/05/rowhammer-attack-exploit.html, May 2018.
[140] A. Kurmus *et al.*, "From Random Block Corruption to Privilege Escalation: A Filesystem Attack Vector for RowHammer-like Attacks," in *WOOT*, 2017.
[141] M. Lanteigne, "How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware," http://www.thirdio.com/rowhammer.pdf, March 2016.
[142] B. C. Lee *et al.*, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.
[143] B. C. Lee *et al.*, "Phase Change Memory Architecture and the Quest for Scalability," *CACM*, 2010.
[144] B. C. Lee *et al.*, "Phase Change Technology and the Future of Main Memory," *MICRO*, 2010.
[145] D. Lee, "Reducing DRAM Latency by Exploiting Heterogeneity," *ArXiV*, 2016.
[146] D. Lee *et al.*, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.
[147] D. Lee *et al.*, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," in *HPCA*, 2015.
[148] D. Lee *et al.*, "Reducing DRAM Latency by Exploiting Design-Induced Latency Variation in Modern DRAM Chips," *ArXiV*, 2016.
[149] D. Lee *et al.*, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.
[150] D. Lee *et al.*, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," *TACO*, 2016.
[151] D. Lee *et al.*, "Design-induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," *POMACS*, 2017.
[152] E. Lee *et al.*, "TWiCe: Time Window Counter Based Row Refresh to Prevent Row-Hammering," *CAL*, 2018.
[153] J. H. Lee *et al.*, "BSSync: Processing Near Memory for Machine Learning Workloads with Bounded Staleness Consistency Models," in *PACT*, 2015.
[154] Lenovo, "Row Hammer Privilege Escalation," https://support.lenovo.com/us/en/product_security/row_hammer, March 2015.
[155] S. Li *et al.*, "Drisa: A DRAM-based Reconfigurable in-situ Accelerator," in *MICRO*, 2017.
[156] S. Li *et al.*, "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories," in *DAC*, 2016.
[157] C. Lim *et al.*, "Active Precharge Hammering to Monitor Displacement Damage Using High-Energy Protons in 3x-nm SDRAM," *IEEE Transactions on Nuclear Science*, 2017.
[158] M. Lipp *et al.*, "Nethammer: Inducing Rowhammer Faults through Network Requests," *arxiv.org*, 2018.
[159] M. Lipp *et al.*, "Meltdown: Reading Kernel Memory from User Space," in *USENIX Security*, 2018.
[160] J. Liu *et al.*, "RAIDR: Retention-aware intelligent DRAM refresh," *ISCA*, 2012.
[161] J. Liu *et al.*, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," *ISCA*, 2013.
[162] Z. Liu *et al.*, "Concurrent Data Structures for Near-memory Computing," in *SPAA*, 2017.
[163] G. H. Loh *et al.*, "A Processing in Memory Taxonomy and a Case for Studying Fixed-function PIM," in *WoNDP*, 2013.
[164] Y. Luo *et al.*, "WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management," *MSST*, 2015.
[165] Y. Luo *et al.*, "Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost via Heterogeneous-Reliability Memory," *DSN*, 2014.
[166] Y. Luo *et al.*, "Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory," *JSAC*, 2016.
[167] Y. Luo *et al.*, "HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness," in *HPCA*, 2018.
[168] Y. Luo *et al.*, "Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation," *POMACS*, 2018.
[169] J. Mandelman *et al.*, "Challenges and Future Directions for the Scaling of Dynamic Random-Access Memory (DRAM)," *IBM Journal of Research and Development*, 2002.
[170] J. Meza *et al.*, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," in *WEED*, 2013.
[171] J. Meza *et al.*, "A Large-Scale Study of Flash Memory Errors in the Field," in *SIGMETRICS*, 2015.
[172] J. Meza *et al.*, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," *DSN*, 2015.
[173] D.-S. Min *et al.*, "Wordline Coupling Noise Reduction Techniques for Scaled DRAMs," in *Symposium on VLSI Circuits*, 1990.
[174] A. Morad *et al.*, "GP-SIMD Processing-in-memory," *TACO*, 2015.
[175] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," *IMW*, 2013.
[176] O. Mutlu, "The RowHammer Problem and Other Issues we may Face as Memory Becomes Denser," *DATE*, 2017.
[177] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," in *MemCon*, 2013.
[178] O. Mutlu, "Error Analysis and Management for MLC NAND Flash Memory," in *Flash Memory Summit*, 2014.
[179] O. Mutlu, "RowHammer and Beyond," in *COSADE*, 2019.
[180] O. Mutlu *et al.*, "Enabling Practical Processing in and near Memory for Data-Intensive Computing," *DAC*, 2019.
[181] O. Mutlu *et al.*, "Processing Data Where It Makes Sense: Enabling In-Memory Computation," *MICPRO*, 2019.
[182] O. Mutlu *et al.*, "The Main Memory System: Challenges and Opportunities," *Communications of the KIISE*, 2015.
[183] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *SUPERFRI*, 2014.
[184] L. Nai *et al.*, "GraphPIM: Enabling Instruction-level Pim Offloading in Graph Computing Frameworks," in *HPCA*, 2017.
[185] P. Nair *et al.*, "A Case for Refresh Pausing in DRAM Memory Systems," in *HPCA*, 2013.
[186] P. J. Nair *et al.*, "XED: Exposing On-Die Error Detection Information for Strong Memory Reliability," in *ISCA*, 2016.
[187] E. Nashilov, "Scientists have made the Rowhammer more Dangerous," https://threatpost.ru/dutch-researchers-made-rowhammer-even-more-dangerous/29378/?es_p=8358650, November 2018.
[188] L. H. Newman, "An Ingenious Data Hack is more Dangerous than Anyone Feared," https://www.wired.com/story/rowhammer-ecc-memory-data-hack/,

November 2018.
[189] S. Nichols, "3 is the Magic Number (of Bits): Flip 'em at Once and Your ECC Protection can be Rowhammer'd," https://www.theregister.co.uk/2018/11/21/rowhammer_ecc_server_protection/, November 2018.
[190] S. Oh and J. Kim, "Reliable Rowhammer Attack and Mitigation Based on Reverse Engineering Memory Address Mapping Algorithms," in *WISA*, 2018.
[191] T.-Y. Oh *et al.*, "A 3.2Gbps/pin 8Gb 1.0V LPDDR4 SDRAM with Integrated ECC Engine for sub-1V DRAM Core Operation," in *ISSCC*, 2014.
[192] K. Park *et al.*, "Experiments and Root Cause Analysis for Active-precharge Hammering Fault in DDR3 SDRAM under 3× nm Technology," *Microelectronics Reliability*, 2016.
[193] K. Park *et al.*, "Statistical Distributions of Row-hammering Induced Failures in DDR3 Components," *Microelectronics Reliability*, 2016.
[194] PassMark Software, "MemTest86: The Original Industry Standard Memory Diagnostic Utility," http://www.memtest86.com/troubleshooting.htm, 2015.
[195] M. Patel *et al.*, "Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices," in *DSN*, 2019.
[196] M. Patel *et al.*, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," *ISCA*, 2017.
[197] A. Pattnaik *et al.*, "Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities," *PACT*, 2016.
[198] P. Pessl *et al.*, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in *USENIX Security*, 2016.
[199] D. Poddebniak *et al.*, "Attacking Deterministic Signature Schemes using Fault Attacks," in *EuroS&P*, 2018.
[200] S. H. Pugsley *et al.*, "NDC: Analyzing the Impact of 3D-stacked Memory+ Logic Devices on MapReduce Workloads," in *ISPASS*, 2014.
[201] R. Qiao and M. Seaborn, "A New Approach for Rowhammer Attacks," in *HOST*, 2016.
[202] M. K. Qureshi *et al.*, "Scalable high performance main memory system using phase-change memory technology," in *ISCA*, 2009.
[203] M. K. Qureshi *et al.*, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *DSN*, 2015.
[204] M. K. Qureshi *et al.*, "Enhancing Lifetime and Security of Phase Change Memories via Start-Gap Wear Leveling," in *MICRO*, 2009.
[205] A. Rahmati *et al.*, "Probable Cause: The Deanonymizing Effects of Approximate DRAM," in *ISCA*, 2016.
[206] S. Raoux *et al.*, "Phase-change Random Access Memory: A Scalable Technology," *IBM Journal of Research and Development*, 2008.
[207] K. Razavi *et al.*, "Flip Feng Shui: Hammering a Needle in the Software Stack," *USENIX Security*, 2016.
[208] M. Redeker *et al.*, "An Investigation into Crosstalk Noise in DRAM Structures," in *MTDT*, 2002.
[209] P. J. Restle *et al.*, "DRAM Variable Retention Time," ser. IEDM, 1992.
[210] S.-W. Ryu *et al.*, "Overcoming the Reliability Limitation in the Ultimately Scaled DRAM using Silicon Migration Technique by Hydrogen Annealing," in *IEDM*, 2017.
[211] J. Sanders, "Every Android Device from the Last 6 Years may be at Risk to RAMPage Vulnerability," https://www.techrepublic.com/article/every-android-device-from-the-last-6-years-may-be-at-risk-to-rampage-vulnerability/, June 2018.
[212] A. Schaller *et al.*, "Intrinsic Rowhammer PUFs: Leveraging the Rowhammer Effect for Improved Security," in *HOST*, 2017.
[213] R. Schilling *et al.*, "Pointing in the Right Direction-Securing Memory Accesses in a Faulty World," in *ACSAC*, 2018.
[214] B. Schroeder *et al.*, "Flash Reliability in Production: The Expected and the Unexpected," in *USENIX FAST*, 2016.
[215] M. Seaborn and T. 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.
[216] M. Seaborn and T. Dullien, "Exploiting the DRAM RowHammer Bug to Gain Kernel Privileges," *BlackHat*, 2016.
[217] V. Seshadri *et al.*, "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," in *MICRO*, 2013.
[218] V. Seshadri *et al.*, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses," in *MICRO*, 2015.
[219] V. Seshadri *et al.*, "Fast Bulk Bitwise AND and OR in DRAM," *CAL*, 2015.
[220] V. Seshadri *et al.*, "Ambit: In-memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," in *MICRO*, 2017.
[221] V. Seshadri and O. Mutlu, "Simple Operations in Memory to Reduce Data Movement," in *Advances in Computers*, 2017.
[222] S. M. Seyedzadeh *et al.*, "Counter-based Tree Structure for Row Hammering Mitigation in DRAM," *CAL*, 2017.
[223] A. Shafiee *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with in-situ Analog Arithmetic in Crossbars," *ISCA*, 2016.
[224] G. Singh *et al.*, "NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning," *DAC*, 2019.
[225] SoftMC Source Code, https://github.com/CMU-SAFARI/SoftMC.
[226] M. Son *et al.*, "Making DRAM Stronger Against Row Hammering," in *DAC*, 2017.
[227] V. Sridharan *et al.*, "Memory Errors in Modern Systems: The Good, The Bad, and The Ugly," in *ASPLOS*, 2015.
[228] V. Sridharan and D. Liberty, "A Study of DRAM Failures in the Field," in *SC*, 2012.
[229] V. Sridharan *et al.*, "Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults," in *SC*, 2013.
[230] Z. Sura *et al.*, "Data Access Optimization in a Processing-in-memory System," in *CF*, 2015.
[231] S. Sutar *et al.*, "D-PUF: An Intrinsically Reconfigurable DRAM PUF for Device Authentication and Random Number Generation," in *TECS*, 2018.
[232] S. Sutar *et al.*, "D-PUF: An Intrinsically Reconfigurable DRAM PUF for Device Authentication in Embedded Systems," in *CASES*, 2016.
[233] Q. Tang *et al.*, "A DRAM Based Physical Unclonable Function Capable of Generating $> 10^{32}$ Challenge Response Pairs per 1Kbit Array for Secure Chip Authentication," in *CICC*, 2017.
[234] Y. Tang *et al.*, "Understanding Adjacent Track Erasure in Discrete Track Media," *Transactions on Magnetics*, 2008.
[235] A. Tatar *et al.*, "Throwhammer: Rowhammer Attacks over the Network and Defenses," *USENIX ATC*, 2018.
[236] A. Tatar *et al.*, "Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer," in *RAID*, 2018.
[237] F. Tehranipoor *et al.*, "DRAM Based Intrinsic Physical Unclonable Functions for System Level Security," in *GLVLSI*, 2015.
[238] F. Tehranipoor *et al.*, "Investigation of DRAM PUFs Reliability Under Device Accelerated Aging Effects," in *ISCAS*, 2017.
[239] L. Tung, "'Rowhammer' DRAM Flaw could be Widespread, says Google," https://www.zdnet.com/article/rowhammer-dram-flaw-could-be-widespread-says-google/, March 2015.
[240] L. Tung, "Android Alert: This New Type of Rowhammer GPU Attack can Hijack your Phone Remotely," https://www.zdnet.com/article/android-alert-this-new-type-of-rowhammer-gpu-attack-can-hijack-your-phone-remotely/, May 2018.
[241] V. van der Veen *et al.*, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," *CCS*, 2016.
[242] V. van der Veen *et al.*, "GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM," in *DIMVA*, 2018.
[243] S. Vig *et al.*, "Rapid Detection of Rowhammer Attacks using Dynamic Skewed Hash Tree," in *HASP*, 2018.
[244] P. Vila *et al.*, "Theory and Practice of Finding Eviction Sets," *S&P*, 2019.
[245] Y. Wang *et al.*, "Detect DRAM Disturbance Error by Using Disturbance Bin Counters," *CAL*, 2019.
[246] Wikipedia, "Row hammer," https://en.wikipedia.org/wiki/Row_hammer.
[247] H.-S. P. Wong *et al.*, "Phase Change Memory," *Proceedings of the IEEE*, 2010.
[248] H.-S. P. Wong *et al.*, "Metal-Oxide RRAM," in *Proceedings of the IEEE*, 2012.
[249] R. Wood *et al.*, "The Feasibility of Magnetic Recording at 10 Terabits Per Square Inch on Conventional Media," *Transactions on Magnetics*, 2009.
[250] X.-C. Wu *et al.*, "Protecting Page Tables from RowHammer Attacks using Monotonic Pointers in DRAM True-Cells," *ASPLOS*, 2019.
[251] Y. Xiao *et al.*, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation," *USENIX Sec.*, 2016.
[252] W. Xiong *et al.*, "Run-time Accessible DRAM PUFs in Commodity Devices," in *CHES*, 2016.
[253] D. Yaney *et al.*, "A Meta-stable Leakage Phenomenon in DRAM Charge Storage - Variable Hold Time," ser. IEDM, 1987.
[254] T. Yang and X.-W. Lin, "Trap-assisted DRAM Row Hammer Effect," *EDL*, 2019.
[255] H. Yoon *et al.*, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," in *ICCD*, 2012.
[256] H. Yoon *et al.*, "Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories," *TACO*, 2014.
[257] D. Yun *et al.*, "Study of TID Effects on One Row Hammering using Gamma in DDR4 SDRAMs," in *IRPS*, 2018.
[258] S. Zeitouni *et al.*, "It's Hammer Time: How to Attack (Rowhammer-based) DRAM-PUFs," in *DAC*, 2018.
[259] D. Zhang *et al.*, "TOP-PIM: Throughput-oriented Programmable Processing in Memory," in *HPDC*, 2014.
[260] Z. Zhang *et al.*, "Triggering Rowhammer Hardware Faults on ARM: A Revisit," in *ASHES*, 2018.
[261] P. Zhou *et al.*, "A Durable and Energy Efficient Main Memory using Phase Change Memory Technology," in *ISCA*, 2009.
[262] Q. Zhu *et al.*, "Accelerating Sparse Matrix-matrix Multiplication with 3D-stacked Logic-in-memory Hardware," in *HPEC*, 2013.

Onur Mutlu is a Professor of Computer Science at ETH Zurich. He is also a faculty member at Carnegie Mellon University, where he previously held the Strecker Early Career Professorship. His current broader research interests are in computer architecture, systems, hardware security, and bioinformatics. A variety of techniques he, along with his group and collaborators, has invented over the years have influenced industry and have been employed in

commercial microprocessors and memory/storage systems. He obtained his PhD and MS in ECE from the University of Texas at Austin and BS degrees in Computer Engineering and Psychology from the University of Michigan, Ann Arbor. He started the Computer Architecture Group at Microsoft Research (2006-2009), and held various product and research positions at Intel Corporation, Advanced Micro Devices, VMware, and Google. He received the inaugural IEEE Computer Society Young Computer Architect Award, the inaugural Intel Early Career Faculty Award, US National Science Foundation CAREER Award, Carnegie Mellon University Ladd Research Award, faculty partnership awards from various companies, and a healthy number of best paper or "Top Pick" paper recognitions at various computer systems, architecture, and hardware security venues. He is an ACM Fellow "for contributions to computer architecture research, especially in memory systems", IEEE Fellow for "contributions to computer architecture research and practice", and an elected member of the Academy of Europe (Academia Europaea). His computer architecture and digital circuit design course lectures and materials are freely available on YouTube, and his research group makes a wide variety of software and hardware artifacts freely available online. For more information, please see his webpage at https://people.inf.ethz.ch/omutlu/.



Jeremie S. Kim received the BS and MS degrees in Electrical and Computer Engineering from Carnegie Mellon University in Pittsburgh, Pennsylvania, in 2015. He is currently working on his PhD with Onur Mutlu at Carnegie Mellon University and ETH Zurich. His current research interests are in computer architecture, memory latency/power/reliability, hardware security, and bioinformatics, and he has several publications on these topics.