This is the summary of the paper titled "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors" which appeared in ISCA in June 2014 [21].

RowHammer: Reliability Analysis and Security Implications

Yoongu Kim[1], Ross Daly , Jeremie Kim[1], Chris Fallin , Ji Hye Lee[1],
Donghyuk Lee[1], Chris Wilkerson[2], Konrad Lai , and Onur Mutlu[1]
[1]*Carnegie Mellon University* [2]*Intel Labs*

I. RowHammer: A New DRAM Failure Mode

As process technology scales down to smaller dimensions, DRAM chips become more vulnerable to *disturbance*, a phenomenon in which different DRAM cells interfere with each other's operation. For the first time in academic literature, our ISCA paper [21] exposes the existence of *disturbance errors* in commodity DRAM chips that are sold and used today. We show that repeatedly reading from the same address could corrupt data in nearby addresses. More specifically:

> *When a DRAM row is opened (i.e., activated) and closed (i.e., precharged) repeatedly (i.e.,* hammered*), it can induce disturbance errors in adjacent DRAM rows.*

This failure mode is popularly called *RowHammer*. We tested 129 DRAM modules manufactured within the past six years (2008–2014) and found 110 of them to exhibit RowHammer disturbance errors, the earliest of which dates back to 2010. In particular, *all* modules from the past two years (2012–2013) were vulnerable, which implies that the errors are a recent phenomenon affecting more advanced generations of process technology. Importantly, disturbance errors pose an easily-exploitable *security threat* since they are a breach of memory protection, wherein accesses to one page (mapped to one row) modifies the data stored in another page (mapped to an adjacent row).

Our ISCA paper [21] makes the following contributions.

1. We **demonstrate** the existence of DRAM disturbance errors on real DRAM devices from three major manufacturers and real systems using such devices (using a simple piece of user-level assembly code).
2. We **characterize** in detail the characteristics and symptoms of DRAM disturbance errors using an FPGA-based DRAM testing platform.
3. We propose and explore various soluions to **prevent** DRAM disturbance errors. We develop a novel, low-cost system-level approach as a viable solution to the RowHammer problem.

1.1. Demonstration of the RowHammer Problem

Code 1a is a short piece of assembly code that we constructed to induce DRAM disturbance errors on real systems. It is designed to generate a read to DRAM on every data access. First, the two `mov` instructions read from DRAM at address `X` and `Y` and install the data into a register and also the cache. Second, the two `clflush` instructions evict the data that was just installed into the cache. Third, the `mfence` instruction ensures that the data is fully flushed before any subsequent memory instruction is executed. Finally, the code jumps back to the first instruction for another iteration of reading from DRAM.

On processors employing out-of-order execution, Code 1a generates multiple read requests, all of which queue up in the memory controller before they are sent out to DRAM: $req_{\mathtt{X}}$, $req_{\mathtt{Y}}$, $req_{\mathtt{X}}$, $req_{\mathtt{Y}}$, $\cdots$. Importantly, we chose the values of `X` and `Y` so that they map to *different* rows within the *same* bank. This is so that the memory controller is forced to open and close the two rows repeatedly: $\mathtt{ACT}_{\mathtt{X}}$, $\mathtt{RD}_{\mathtt{X}}$, $\mathtt{PRE}_{\mathtt{X}}$, $\mathtt{ACT}_{\mathtt{Y}}$, $\mathtt{RD}_{\mathtt{Y}}$, $\mathtt{PRE}_{\mathtt{Y}}$, $\cdots$. Using the address-pair (`X`, `Y`), we then executed Code 1a for millions of iterations. Subsequently, we repeated this procedure using many different address-pairs until every row in the DRAM module was opened/closed millions of times. In the end, we observed that Code 1a caused many bits to flip. For four different processors, Table 1 reports the total number of bit-flips induced by Code 1a for two different initial states of the module: all '0's or all '1's. Since Code 1a *does not write* any data into DRAM, we conclude that the bit-flips are the manifestation of disturbance errors caused by repeated reading (i.e., hammering) of each memory row. In the next section, we will show that this particular DRAM module yields *millions* of errors in a more controlled environment.

```
code1a:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp code1a
```

a. Induces errors

```
code1b:
  mov (X), %eax
  clflush (X)


  mfence
  jmp code1b
```

b. Does not induce errors

Code 1. Assembly code executed on Intel/AMD machines

Bit-Flip	Intel Sandy Bridge	Intel Ivy Bridge	Intel Haswell	AMD Piledriver
'0' → '1'	7,992	10,273	11,404	47
'1' → '0'	8,125	10,449	11,467	12

Table 1. Bit-flips induced by disturbance on a 2GB module

As a control experiment, we also ran Code 1b which reads from only a single address. Code 1b did *not* induce any disturbance errors as we expected. For Code 1b, all of its reads are to the same row in DRAM: $req_{\mathtt{X}}$, $req_{\mathtt{X}}$, $req_{\mathtt{X}}$, $\cdots$. In this case, the memory controller minimizes the number of DRAM commands [39, 44, 35, 36, 24, 23, 4, 42, 43] by opening and closing the row just *once*, while issuing many column reads in between: $\mathtt{ACT}_{\mathtt{X}}$, $\mathtt{RD}_{\mathtt{X}}$, $\mathtt{RD}_{\mathtt{X}}$, $\mathtt{RD}_{\mathtt{X}}$, $\cdots$, $\mathtt{PRE}_{\mathtt{X}}$. From this we conclude that DRAM disturbance errors are indeed caused by the repeated opening/closing of a row, and *not* by the reads themselves.

1.2. Characterization of the RowHammer Problem

To develop an understanding of disturbance errors, we characterize 129 DRAM modules on an FPGA-based DRAM testing platform [16, 26, 20, 21, 38]. Unlike a general-purpose processor, our testing platform grants us precise and fine-grained control over how and when DRAM is accessed on a cycle-by-cycle basis. To characterize a module, we test each one of its rows one by one. First, we initialize the entire module with a known data-pattern. Second, we activate one particular row as quickly as possible (once every 55*ns*) for the full duration of a refresh interval (64*ms*). Third, we read out the entire module and search for any changes to the data-pattern. We then repeat the three steps for every row in the module.

Key Findings. In the following, we summarize four of the most important findings of our RowHammer characterization study.

1. Errors are widespread. Figure 1 plots the normalized number of errors for each of the 129 modules as a function of their manufacture date. Our modules are sourced from three major DRAM manufacturers whose identities have been anonymized to A, B, and C. From the figure, we see that disturbance errors first started to appear in 2010, and that they afflict all modules from 2012 and 2013. In particular, for each manufacturer, the number of errors per 10^9 cells can reach up to 5.9×10^5, 1.5×10^5, and 1.9×10^4, respectively. To put this into perspective, there can be as many as 10 million errors in a 2GB module.

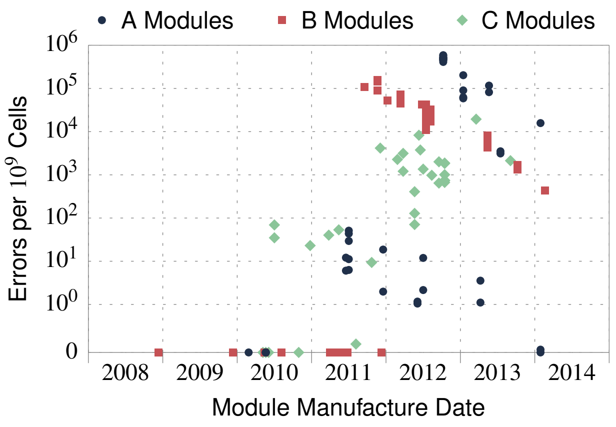

Figure 1. Normalized number of errors vs. manufacture date

2. Errors are symptoms of charge loss. For a given DRAM cell, we observed that it experiences data loss in only a single direction: either '1'→'0' or '0'→'1', but not both. This is due to an intrinsic property of DRAM cells called *orientation*. Depending on the implementation, some cells represent a data value of '1' using the charged state, while other cells do so using the discharged state — these cells are referred to as *true-cells* and *anti-cells*, respectively [28]. We profiled several modules for the orientation of their cells, and discovered that true-cells experience only '1'→'0' errors and that anti-cells experience only '0'→'1' errors. From this, we conclude that disturbance errors occur as a result of charge loss.

3. Errors occur in adjacent rows. We verified that the disturbance errors caused by activating a row are localized to two of its immediately adjacent rows. There could be three possible ways in which a row interacts with its neighbors to induce their charge loss: *(i)* electromagnetic coupling, *(ii)* conductive bridges, and *(iii) hot-carrier injection*. We confirmed with at least one major DRAM manufacturer that these three phenomena are potential causes of the errors. Section 3 of our ISCA paper [21] provides more analysis.

4. Errors are access-pattern dependent. For a cell to experience a disturbance error, it must lose enough charge before the next time it is replenished with charge (i.e., *refreshed*). Hence, the more frequently we refresh a module, the more we counteract the effects of disturbance, which decreases the number of errors. On the other hand, the more frequently we activate a row, the more we strengthen the effects of disturbance, which increases the number of errors. We experimentally validated this trend by sweeping the *refresh interval* and the *activation interval* between 10–128*ms* and 55–500*ns*, respectively. In particular, we observed that no errors are induced if the refresh interval is $\leq$8*ms* or if the activation interval is $\geq$500*ns*. Importantly, we found that it takes as few as 139K activations to a row before it induces disturbance errors.

Other Findings. Our ISCA paper provides an extensive set of characterization results, some of which we list below.

- RowHammer errors are repeatable. Across ten iterations of tests, >70% of the erroneous cells had errors in every iteration.
- Errors are not strongly affected by temperature. The numbers of errors at 30°C, 50°C, and 70°C differ by <15%.
- There is almost no overlap between erroneous cells and *weak cells* (i.e., cells that are inherently the leakiest and thus require a higher refresh rate).
- Simple ECC (e.g., SECDED) *cannot* prevent *all* RowHammer-induced errors. There are as many as four errors in a single cache-line.
- Errors are data-pattern dependent. The *Solid* data-pattern (all '0's or all '1's) yields the fewest errors, whereas the *RowStripe* data-pattern (alternating rows of '0's and '1's) yields the most errors.
- A very small fraction of cells experience an error when either one of their adjacent rows is repeatedly activated.

1.3. Prevention of the RowHammer Problem

In our ISCA paper, we examine a total of *seven* solutions to tolerate, prevent, or mitigate DRAM disturbance errors. The first six solutions are: 1) making better DRAM chips, 2) using error correcting codes (ECC), 3) increasing the refresh rate, 4) remapping error-prone cells after manufacturing, 5) remapping/retiring error-prone cells at the user level during operation, 6) identifying hammered rows and refreshing their neighbors. None of these frst six solutions are very desirable as they come at various significant power, performance or cost overheads.

Our main proposal to solve the RowHammer problem is a novel 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 of its adjacent rows is also opened (i.e., refreshed) with some low probability. 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. The main advantage of PARA is that it is *stateless*. PARA 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 error-prone cells. PARA can be implemented either in the memory controller or the DRAM chip (internally).

PARA is implemented as follows. Whenever a row is closed, the PARA control logic flips a biased coin with a probability p of turning up heads, where $p \ll 1$. If the coin turns up heads, the controller opens one of the adjacent rows where either of them is chosen with equal probability ($p/2$). The parameter p can be set so that disturbance errors occur at an extremely low rate — many orders of magnitude lower than the failure rates of other system components (e.g., hard-disk). In fact, even under the most adversarial conditions, PARA's failure rate is only 9.4×10^{-14} errors-per-year when p is set to just 0.001. Due to the extra activations, PARA incurs a small performance overhead (slowdown of 0.20% averaged across 29 benchmarks), which we believe to be justified by the *(i)* strong reliability guarantee and *(ii)* low design complexity resulting from PARA's stateless nature. More detailed analysis can be found in our ISCA 2014 paper [21].

II. Significance

Our ISCA paper [21] identifies a new reliability problem and a security vulnerability, *RowHammer*, that affects an entire generation of computing systems being used today. We build a comprehensive understanding of the problem based on a wealth of empirical data we obtain from more than 120 DRAM memory modules. After examining various ways of addressing the problem, we propose a low-overhead solution that provides a strong reliability (and, hopefully, security) guarantee.

Exposition. For the first time in academic literature, we expose the widespread vulnerability of commodity DRAM chips to disturbance (aka, RowHammer) errors. After testing a large sample population of DRAM modules (the oldest of which dates back to 2008), we determine that the problem first arose in 2010 and that it still persists to this day. We found modules from *all* three major manufacturers, as well as in *all* modules assembled between 2012–2013, are vulnerable to the RowHammer problem.

Demonstration. We demonstrate that disturbance errors are an actual hardware vulnerability affecting real systems. We construct a user-level kernel which induces many errors on general-purpose processors from Intel (Sandy Bridge, Ivy Bridge, Haswell) and AMD (Piledriver). With its ability to bypass memory protection (OS/VMM), the kernel can be deployed as a *disturbance attack* to corrupt the memory state of a system and its software. We discuss the possibility, for the first time, that this *RowHammer* problem can be developed into a malicious "disturbance attack" (Section 4 of our ISCA paper [21]).[1]

Characterization. We characterize the cause and symptoms of disturbance errors based on a large-scale study, involving 129 DRAM modules (972 DRAM chips) sampled from a time span of six years. We extensively test the modules using a custom-built FPGA infrastructure [16, 26, 28, 21, 38] to determine the specific conditions under which the errors occur, as well as the specific manner in which they occur. From this, we build a comprehensive understanding of disturbance errors.

Solution. We propose seven categories of solutions that could potentially be employed to prevent disturbance errors. Among them, our main proposal is called *PARA* (*probabilistic adjacent row activation*), whose major advantage lies in its stateless nature. PARA eliminates the need for hardware counters to track the number of activations to different rows (proposed in other potential solutions [5, 6, 7, 8, 11, 12, 18]), while still being able to refresh the at-risk rows in a timely manner. Based on our analysis, we establish that PARA provides a strong reliability guarantee even under the worst-case conditions, proving it to be an effective and efficient solution against the RowHammer problem.

2.1. Long-Term Impact

We believe our ISCA paper will affect industrial and academic research and development for the following four reasons. First, it exposes a real and pressing manifestation of the difficulties in DRAM scaling — a critical problem which is expected to become only worse in the future [33, 37, 34]. Second, it breaks the conventional wisdom of memory protection, demonstrating that the system-software (OS/VMM) — just by itself — cannot isolate the address space of one process (or virtual machine) from that of another, thereby exposing the vulnerability of systems, for the first time, to what we call *DRAM disturbance attacks*, or *RowHammer attacks*. Third, it builds the necessary experimental infrastructure to seize full control over DRAM chips, thereby unlocking a whole new class of characterization studies and quantitative data. Fourth, it emphasizes the important role of computer architects to examine holistic approaches for improving system-level reliability and security mechanisms for modern memory devices — even when the underlying hardware is unreliable.

Empirical Evidence of Challenges in DRAM Technology Scaling. DRAM process scaling is becoming more difficult due to increased cost and complexity, as well as degraded reliability [9, 14, 15, 33, 37, 34]. This explains why disturbance errors are found in all three DRAM manufacturers, in addition to why their first appearances coincide with each other during the same timeframe (2010–2011). As process scaling continues onward, we could be faced with a new and diverse array of DRAM failures, some of which may be diagnosed only after they have been released into the wild — as was the case with disturbance errors of today. In this context, our paper raises awareness about the next generation of DRAM failures that could undermine system integrity. We expect, based on our experimental evidence of DRAM errors, that future DRAM chips could suffer from similar or other vulnerabilities.

RowHammer Security Implications: Threats of Unreliable Memory. Virtual machines and virtual memory protection mechanisms exist to provide an isolated execution environment that is safeguarded from external tampering or snooping. However, in the presence of disturbance errors (or other hardware faults), it is possible for one virtual machine (or application) to corrupt the memory of another virtual machine (or application) that is housed in the same physical machine. This has serious implications for modern systems (from mobile systems to data centers), where multiprogramming is common and many virtual machines (or applications) potentially from different users are consolidated onto the same physical machine. As long as there is sharing between any two pieces of software, our paper shows that

[1]Recent works [40, 13] that build upon our ISCA 2014 paper actually demonstrate that a user can take over an entire system and gain kernel privilges by intelligently exploiting the RowHammer problem.

strong isolation guarantees between them *cannot* be provided unless all levels of the system stack are secured. As a result, malicious software can be written to take advantage of these disturbance errors. We call these *disturbance attacks*, 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 [21], researchers from Google Project Zero recently developed a user-level attack that exploits disturbance errors to take over an entire system [40]. More recently, researchers have shown that the RowHammer can be exploited remotely via the use of JavaScript [13]. As such, the new problem exposed by our ISCA 2014 paper, the DRAM RowHammer problem, has widespread and profound real implications on system security, threatening the foundations of memory security on top of which modern systems are built.

DRAM Testing Infrastructure. Our paper builds a powerful infrastructure for testing DRAM chips. It was designed to grant the user with software-based control over the precise timing and the exact data with which DRAM is accessed. This creates new opportunities for DRAM research in ways that were not possible before. For example, we have already leveraged the infrastructure for *(i)* characterizing different modes of retention failures [17, 28] and *(ii)* characterizing the safety margin in timing parameters to operate DRAM at lower latencies than what is recommended [26]. In the future, we plan to open-source the infrastructure for the benefit of other researchers.

System-Level Approach to Enable DRAM Scaling. Unlike most other known DRAM failures, which are relatively easily caught by the manufacturers, disturbance errors require an extremely large number of accesses before they are sensitized — a full-coverage test to reveal all disturbance errors could take days or weeks. 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 which are sensitized only under specific access-patterns and/or data-patterns. As a scalable solution for the future, our paper argues for adopting a system-level approach [33] to DRAM reliability and security, in which the DRAM chips, the memory controller, and the operating system collaborate together to diagnose/treat emerging DRAM failure modes.

III. Conclusion

Our ISCA 2014 paper [21] is the first work that exposes, demonstrates, characterizes, and prevents a new type of DRAM failure, the DRAM RowHammer problem, which can cause serious security and reliability problems. As DRAM process technology scales down to even smaller feature sizes, we hope that our findings will inspire the research community to develop innovative approaches to enhance system-level reliability and security by focusing on such new forms of memory errors and their implications for modern computing systems.

IV. More Information

For more information, we point the reader to the following resources:

1. We have released source code to induce DRAM RowHammer errors as open source software [3].

2. We have released presentations on the RowHammer problem [32, 19].

3. We have written various papers describing challenges in DRAM scaling and memory systems in general [33, 37, 34].

4. Building upon our observations, others have exploited the RowHammer problem to take over modern systems and have released their source code [40, 13].

5. Mark Seaborn maintains a discussion group [1] that discusses RowHammer issues.

6. More detailed background information and discussion on DRAM can be found in our video lectures [30, 31] or recent works that explains the operation and architecture of modern DRAM chips [29, 25, 27, 26, 16, 28, 10, 41, 22].

7. Twitter has a record of popular discussions on the RowHammer problem [2].

References

[1] RowHammer Discussion Group. `https://groups.google.com/forum/#!forum/rowhammer-discuss`.

[2] RowHammer on Twitter. `https://twitter.com/search?q=rowhammer&src=typd`.

[3] SAFARI RowHammer Code. `https://github.com/CMU-SAFARI/rowhammer`.

[4] R. Ausavarungnirun, K. Chang, L. Subramanian, G. Loh, and O. Mutlu. Staged Memory Scheduling: Achieving high performance and scalability in heterogeneous systems. In *ISCA*, 2012.

[5] K. Bains et al. Method, Apparatus and System for Providing a Memory Refresh. US Patent App. 13/625,741, Mar. 27 2014.

[6] K. Bains et al. Row Hammer Refresh Command. US Patent App. 13/539,415, Jan. 2 2014.

[7] K. Bains et al. Row Hammer Refresh Command. US Patent App. 14/068,677, Feb. 27 2014.

[8] K. Bains and J. Halbert. Distributed Row Hammer Tracking. US Patent App. 13/631,781, Apr. 3 2014.

[9] S. Y. Cha. DRAM and Future Commodity Memories. In *VLSI Technology Short Course*, 2011.

[10] K. Chang, D. Lee, Z. Chishti, C. Wilkerson, A. Alameldeen, Y. Kim, and O. Mutlu. Improving DRAM Performance by Parallelizing Refreshes with Accesses. In *HPCA*, 2014.

[11] 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.

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

[13] D. Gruss et al. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. *arXiv.org*, 2015.

[14] S. Hong. Memory Technology Trend and Future Challenges. In *IEDM*, 2010.

[15] U. Kang et al. Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling. In *The Memory Forum (ISCA)*, 2014.

[16] S. Khan et al. The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study. In *SIGMETRICS*, 2014.

[17] S. Khan et al. The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study. In *SIGMETRICS*, 2014.

[18] D. Kim et al. Architectural Support for Mitigating Row Hammering in DRAM Memories. *Computer Architecture Letters*, 2014.

[19] Y. Kim. Flipping Bits in Memory Without Accessing Them. `http://users.ece.cmu.edu/~omutlu/pub/dram-row-hammer_kim_talk_isca14.pptx`.

[20] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In *ISCA*, 2014.

[21] Y. Kim et al. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In *ISCA*, 2014.

[22] Y. Kim et al. Ramulator: A Fast and Extensible DRAM Simulator". *IEEE CAL*, 2015.

[23] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers. In *HPCA*, 2010.

[24] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior. In *MICRO*, 2010.

[25] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu. A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM. In *ISCA*, 2012.

[26] D. Lee et al. Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case. In *HPCA*, 2015.

[27] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu. Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture. In *HPCA*, 2013.

[28] J. Liu et al. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. In *ISCA*, 2013.

[29] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *ISCA*, 2012.

[30] O. Mutlu. Main Memory and DRAM Basics. `https://www.youtube.com/watch?v=ZLCy3pG7Rc0`.

[31] O. Mutlu. Main Memory and the DRAM System. `https://www.youtube.com/watch?v=IUk9o9wvX1Y`.

[32] O. Mutlu. The DRAM RowHammer Problem (and Its Reliability and Security Implications). `http://users.ece.cmu.edu/~omutlu/pub/rowhammer.pptx`.

[33] O. Mutlu. Memory Scaling: A Systems Architecture Perspective. In *MemCon*, 2013.

[34] O. Mutlu, J. Meza, and L. Subramanian. The main memory system: Challenges and opportunities. *Communications of the KIISE*, 2015.

[35] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *MICRO*, 2007.

[36] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *ISCA*, 2008.

[37] O. Mutlu and L. Subramanian. Research problems and opportunities in memory systems. *Superfri*, 2015.

[38] M. Qureshi et al. AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems. In *DSN*, 2015.

[39] S. Rixner et al. Memory Access Scheduling. In *ISCA*, 2000.

[40] M. Seaborn. Exploiting the DRAM rowhammer bug to gain kernel privileges. `http://googleprojectzero.blogspot.com.tr/2015/03/exploiting-dram-rowhammer-bug-to-gain.html`.

[41] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data. In *MICRO*, 2013.

[42] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu. The blacklisting memory scheduler: Achieving high performance and fairness at low cost. In *ICCD*, 2014.

[43] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu. MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems. In *HPCA*, 2013.

[44] W. Zuravleff and T. Robinson. Controller for a Synchronous DRAM that Maximizes Throughput by Allowing Memory Requests and Commands to be Issued Out of Order. US Patent 5,630,096, May 13, 1997.