Siloz: Leveraging DRAM Isolation Domains to Prevent Inter-VM Rowhammer

Kevin Loughlin
University of Michigan

Jonah Rosenblum
University of Michigan

Stefan Saroiu
Microsoft

Alec Wolman
Microsoft

Dimitrios Skarlatos
Carnegie Mellon University

Baris Kasikci
University of Washington and Google

Abstract

Today's cloud DRAM lacks strong isolation primitives, highlighted by Rowhammer bit flips. Rowhammer poses an increasing threat to cloud security/reliability, given (1) DRAM activation rates in commodity and malicious workloads already exceed Rowhammer thresholds, and (2) thresholds are decreasing in newer DRAM. Deployed hardware mitigations remain vulnerable, turning cloud providers toward software defenses. However, existing defenses incur high performance or memory overhead or contain significant protection gaps.

Accordingly, we introduce *Siloz*, a hypervisor that uses *subarray groups* as DRAM isolation domains to enable efficient protection against inter-VM Rowhammer. *Siloz* exploits the insights that (a) Rowhammer can only flip bits in DRAM rows located in the same subarray—not across subarrays—and (b) VMs can be isolated to groups of subarrays without sacrificing bank-level parallelism, a key component of DRAM performance. *Siloz* thus prevents inter-VM bit flips by placing each VM's and the host's data into private subarray groups. To additionally ensure that a VM cannot escape its provisioned subarray group(s), *Siloz* provides integrity protection for extended page tables (EPTs). We show that *Siloz*'s implementation has negligible effect on average performance across various cloud workloads, SPEC CPU 2017, and PARSEC 3.0 (within ±0.5% of baseline Linux/KVM).

CCS Concepts: • **Security and privacy → Systems security**; **Security in hardware**.

Keywords: DRAM Disturbances, DRAM Isolation Domains, DRAM Subarrays, Rowhammer, Security

ACM Reference Format:
Kevin Loughlin, Jonah Rosenblum, Stefan Saroiu, Alec Wolman, Dimitrios Skarlatos, and Baris Kasikci. 2023. Siloz: Leveraging DRAM Isolation Domains to Prevent Inter-VM Rowhammer. In *ACM SIGOPS 29th Symposium on Operating Systems Principles (SOSP '23), October 23–26, 2023, Koblenz, Germany.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3600006.3613143

SOSP '23, October 23–26, 2023, Koblenz, Germany

ACM ISBN 979-8-4007-0229-7/23/10.
https://doi.org/10.1145/3600006.3613143

1 Introduction

Cloud providers host virtual machines (VMs) from multiple tenants atop the same physical machine, while providing per-VM isolation across various metrics [38, 126]. To provide per-VM performance isolation, providers use a rich set of technologies across hardware resources (e.g., CPU affinity [99], SR-IOV [32], and memory bandwidth allocation [55]). Although providers can also use a growing set of methods to provide per-VM security isolation (e.g., CPU enclaves [27], cache partitioning [81], and memory encryption [3, 54]), providers lack practical means to provide strong isolation in one of the most significant cloud server resources: DRAM.

In particular, today's servers interleave (spread) data from multiple tenants across different DRAM banks, ranks, and channels to maximize the memory-level parallelism afforded by these structures [102, 143]. Unfortunately, sharing these structures without careful consideration exposes co-located VMs to interference [40, 122, 134, 135], including the security and reliability threat of Rowhammer bit flips [79]. To combat interference, we envision a future in which cloud providers can leverage *DRAM isolation domains* that provide isolation capabilities in line with those of other hardware resources.

The goal of this work is to enable cloud providers to take the first step toward practically managing DRAM as a set of isolated domains. To achieve this goal, we propose the use of DRAM *subarrays* for isolation. DRAM consists of many subarrays that are natural isolation boundaries of DRAM cells [80]: cells in one subarray cannot disturb cells in another [24]. While DRAM does not expose subarrays today, software can easily determine subarray boundaries (§4.1). By using subarrays, we show that inter-VM Rowhammer can be prevented on *today's* cloud servers without sacrificing performance.

1.1 This Paper: Mitigating Inter-VM Rowhammer

Inter-VM Rowhammer is a glaring example of today's lack of DRAM isolation; a VM's frequent activations (≈ accesses) of the same DRAM rows—"hammering"—can flip bits in nearby rows used by another VM or the host. Bit flips can cause data loss [79], machine check exceptions [25], denial-of-service [58], side channels [22, 86], and system subversion [133].

Despite deployed hardware mitigations [25, 37, 59, 83], cloud systems remain vulnerable to inter-VM hammering. In fact, recent work [98] shows that malicious and *commodity* cloud workloads already activate rows at rates exceeding today's Rowhammer thresholds. As these thresholds continue to decrease with process scaling [24, 74, 129], Rowhammer poses an *increasing* threat to security and reliability.

State-of-the-art software defenses incur high memory/performance overhead or contain significant gaps in protection. SoftTRR [173] and CTA [161] do not scalably-generalize beyond page table protection, leaving all other data vulnerable. Copy-on-Flip [31] does not protect unmovable memory pages (i.e., a subset of kernel pages) and is susceptible to ECC-corrected disturbances, which can leak data [25]. "Guard row" mitigations [11, 12, 84]—where a set of guard rows are reserved as protection buffers between normal rows—require $\geq 50\%$ extra DRAM per protected region and thus only scale to protect small quantities of data.

Given the limitations of existing software Rowhammer mitigations—coupled with the dearth of hardware DRAM isolation support—cloud providers lack practical means to mitigate inter-VM Rowhammer. Thus, we introduce *Siloz*, a hypervisor that uses *subarray groups* as DRAM isolation domains to prevent inter-VM hammering with negligible performance effect; *Siloz* integrates subarray group isolation, bank-level parallelism, and extended page table (EPT) integrity for efficient protection against inter-VM hammering.

Siloz's key insight is that subarray-based Rowhammer isolation can co-exist with *bank-level parallelism*. Bank-level parallelism is the finest-grained access parallelism exposed by modern DRAM, offering > 18% execution time improvement [143]. As such, *Siloz* enables high performance alongside per-VM Rowhammer isolation by partitioning DRAM into subarray groups of ≈ 1.5 GiB each (depending on DRAM geometry), formed from at least 1 subarray per each bank in a memory pool. Thus, a VM using subarray group(s) can allocate memory across banks, yet isolated to select subarrays.

To conveniently manage subarray groups, *Siloz* builds on existing non-uniform memory access (NUMA) support. *Siloz* abstracts subarray groups as *logical* NUMA nodes, enabling robust memory management, while maintaining compatibility with *physical* NUMA performance optimizations (e.g., *Siloz* can use same-socket subarray groups for lower latency).

Notably, *Siloz*'s ability to enforce subarray group isolation relies on EPT integrity; because EPTs define the host physical addresses that VMs can access, a malicious VM could induce bit flips in even its *own* EPTs to access another domain [133]. While emerging Intel and AMD hardware offer support for EPT integrity checks [3, 54], *Siloz* can also protect against EPT bit flips on legacy systems. Namely, *Siloz* exploits the insight that all EPTs can fit in $< 0.001\%$ of DRAM rows and are thus amenable to supplemental guard row protection without significant cost. By accounting for server DRAM addressing alongside prior guard row techniques [11, 12, 84], *Siloz* limits DRAM overheads for EPT protection to just 32 8 KiB rows per bank ($\approx 0.024\%$ of a 1 GiB bank).

We evaluate *Siloz*'s Linux/KVM [82] implementation on Intel Skylake servers based on a major cloud provider's configuration, demonstrating that *Siloz* prevents inter-VM hammering and EPT bit flips. We find that *Siloz*'s combination of subarray group isolation and EPT protection has negligible effect on average performance (within $\pm 0.5\%$ of baseline Linux/KVM) across various cloud workloads [14, 26, 67, 85, 118], SPEC CPU 2017 [13], and PARSEC 3.0 [10, 170].

In summary, we make the following contributions:

- We present *Siloz*, a hypervisor that uses subarray groups as high-performance *DRAM isolation domains* in the cloud.
- To safeguard isolation metadata, *Siloz* places EPTs in guard-protected rows, using knowledge of DRAM addressing to securely limit reserved DRAM to $\approx 0.024\%$ of each bank.
- We show that *Siloz* offers cloud providers a practical and comprehensive mitigation for inter-VM hammering, providing complete protection with negligible effect on average performance (within $\pm 0.5\%$ of baseline Linux/KVM).

Siloz's Linux/KVM implementation is open-source [95].

2 Background

In this section, we present background on server systems, DRAM, and Rowhammer as needed to understand *Siloz*.

2.1 Hypervisor Memory Management

State-of-the-art hypervisors like Linux/KVM [82] use hardware virtualization extensions (e.g., Intel VT-x [148] or AMD-V [69]) to map host memory pages into a VM's address space. The VM can then access the vast majority of its memory without performance-costly traps/exits into the hypervisor.

While an OS manages mappings between two types of addresses (virtual and physical) for standard processes, a hypervisor manages mappings among *three* types of addresses for VMs: (1) guest virtual addresses (GVAs, equivalent to standard virtual addresses), (2) guest physical addresses (GPAs, the VM's illusion of physical addresses), and (3) host physical addresses (HPAs, equivalent to standard physical addresses). The guest OS's page tables map GVAs to GPAs, while the hypervisor's *extended page tables* (EPTs) map GPAs to HPAs.

2.2 Non-Uniform Memory Access (NUMA)

Cloud providers deploy large quantities of compute and memory per server for cost effectiveness and ease of management. To scale performance amidst large resource quantities, servers are often architected as non-uniform memory access (NUMA). A *NUMA node* conventionally refers to a combination of cores (e.g., a socket) and a local (near) memory pool that is faster to access than remote (far) memory; technically, a node may consist of only cores, only memory, or both.

NUMA's key benefits are its abilities to decrease latencies for workloads using local memory and to reduce interference

(e.g., memory traffic) among independent tasks on different nodes. Additionally, kernel NUMA support offers convenient abstractions to manage compute and/or memory resources.

2.3 Server DRAM (Micro)architecture

A server DRAM module is a hierarchically-organized set of DRAM cells, each storing a bit as high/low charge. Each module is typically attached to a CPU socket, with the socket and its modules forming a conventional NUMA node (§2.2).

Because DRAM cell charges diminish over time, memory controllers and DRAM modules cooperate to periodically *refresh* the charges for data retention. In widely-deployed DDR4 [60] DRAM, cells are refreshed within 64 milliseconds.

As shown in Fig. 1, the DRAM module hierarchy is a set of *ranks*, each encompassing *banks*, each encompassing *subarrays*, which are each a row-column grid of cells. The DDR4 standard specifies that a rank holds up to 16 banks, and a row holds up to 8 KiB of cells. Internal to server DRAM modules, each of a subarray's 8 KiB rows is split into two half-rows across a rank's "A" and "B" sides, with each half-row simultaneously serving half of a given data request [62]. While a row's external representation (i.e., a single 8 KiB structure) is sufficient to understand the majority of *Siloz*'s design, we discuss the relevance of internal half-rows in §6.

Although many other (micro)architectural details are vendor-specific, a common server DDR4 DRAM module is a dual-rank, 32 GiB DIMM (dual in-line memory module). Given 32 GiB split across 2 ranks and 16 banks/rank, a bank is 1 GiB. Each bank is divided among a vendor-specific number of subarrays [18, 24, 80, 100, 155], each consisting of 512–2048 rows [155]. For example, *Siloz*'s evaluation server's subarray size of 1024 8 KiB rows yields 128 subarrays per 1 GiB bank.

2.4 Accessing Data in DRAM

To read/write data in DRAM, a memory controller first translates the data's host physical address to a *media address* that identifies specific DRAM cells. The parallel in the CPU realm is a virtual-to-physical address mapping, which system software typically controls at page-sized granularity (§2.1). However, unlike software-defined virtual-to-physical mappings, physical-to-media mappings are fixed at boot via BIOS settings [49, 56] and applied at cache line granularity.

Given the data's media address, the controller first issues an *activate* (ACT) to the row containing the data. This command connects the row to its encompassing bank's *row buffer*, which can only be occupied by one row per bank at a time. The controller then issues a read or write command to an offset within the row buffer, completing the data access.

While accesses to a single bank are serialized, *different* banks can be accessed in parallel. Thus, commodity physical-to-media address mappings maximize throughput by interleaving (spreading) sequential cache lines across a conventional NUMA node's (e.g., socket's) banks, achieving *bank-level parallelism* for common access patterns [96, 143, 171].

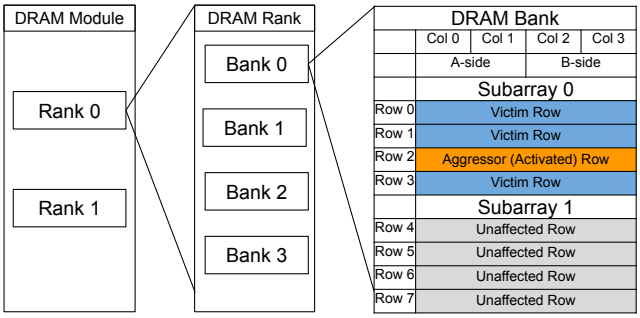

Figure 1. A simplified DRAM module hierarchy (§2.3) in the context of a DRAM row activation (§2.4) and Rowhammer (§2.5). A frequently-activated ("hammering") *aggressor* row may flip bits in *victim* rows in the same subarray.

2.5 Rowhammer and RowPress

Rowhammer [79] is a silicon-level effect in DRAM where frequent ACTs (§2.4) of the same *aggressor* rows can flip bits in nearby *victim* rows due to electric interference. Namely, an ACT's side effects include (a) refreshing charges in the activated row, but (b) potentially disturbing charges in *nearby* rows. When aggressor rows are activated at rates exceeding Rowhammer thresholds (varying across DIMMs), cumulative disturbance effects may flip bits in victim rows that have not been recently-refreshed. Per Fig. 1, rows in the aggressor's subarray are potential victims, while rows in different subarrays are unaffected due to electric isolation [18, 24, 96, 164].

Rowhammer bit flips can cause data loss [79], machine check exceptions [25], denial-of-service [58], side channels [22, 86], and system subversion [133]. Recent work [98] shows that malicious and *commodity* workloads can yield ACT rates surpassing modern Rowhammer thresholds; other work [24, 74, 129] shows that thresholds are decreasing with process node scaling (i.e., susceptibility is increasing).

To mitigate Rowhammer, modern servers rely on error correction codes (ECC [7]) and target row refresh (TRR [37], an in-DRAM mitigation that refreshes a *subset* of victim rows ahead of schedule). While these mitigations have thus far proved effective for commodity workloads, malicious workloads can induce uncorrected bit flips despite ECC [25] and TRR [28, 37, 59, 83]. Even *corrected* bit flips are a security concern, forming side channels that can leak a row's data to attackers with access to other rows in the subarray [86].

RowPress [101] is a similar, recently-discovered phenomenon in which aggressor rows left activated for long time periods of time may flip bits in nearby victim rows. Because subarrays can also form isolation boundaries for such disturbances, this paper treats RowPress similar to Rowhammer.

3 Limitations of Existing Software Defenses

Given gaps in hardware mitigations and the goal of per-VM isolation, cloud providers can use software to supplementally

mitigate *inter-VM hammering*, where a VM's hammering can flip bits in another VM or the host. Unfortunately, existing software mitigations fail to mitigate such hammering without significant performance/memory overheads (if at all), motivating *Siloz*'s design. Broadly, existing software mitigations adopt one of three approaches: selectively-protecting data, detecting attacks in progress, or inserting guard rows.

Mitigation via Selective Data Protection. The first mitigation class protects a subset of data as a security-performance trade-off [161, 173]. For instance, SoftTRR [173] periodically sets reserved bits in page table entries for potential aggressor rows. Thus, accesses to the designated aggressors will trap into system software, which can refresh neighboring victims before the aggressors surpass the Rowhammer threshold.

The key limitation of these defenses is that they only protect a small portion of a VM's data (e.g., page tables) for acceptable performance overheads. As we will show, *Siloz* protects *all* of a VM's data against another VM's hammering.

Mitigation via Attack Detection. The second mitigation class aims to detect Rowhammer and correspondingly stop the attack [5, 17, 31]. For instance, Copy-on-Flip [31] uses ECC-corrected disturbances to identify pages under attack and—if the pages are movable—migrate them in DRAM.

The key limitation of these approaches is that their detection methods do not prevent all disturbances. For example, the ECC-corrected disturbances used for detection in Copy-on-Flip can leak data [90]. In contrast, *Siloz*'s subarray groups isolate each VM from another's hammering, agnostic to Rowhammer thresholds and a VM's hammering method.

Mitigation via Guard Rows. Several proposed software Rowhammer mitigations [11, 12, 84] place *guard rows* between isolation domains (e.g., user-kernel or different processes). These mitigations exploit the fact that Rowhammer only affects data in nearby rows (§2.5), reserving guard rows as protection barriers between "normal" rows. If hammering occurs in the normal rows, it can only flip bits in guard rows, which are unused or contain supplementally-protected data.

These mitigations' key limitation is that they waste DRAM; the guard rows cannot be used as normal rows. Protecting arbitrary data incurs large overheads (i.e., $\geq$ 50% extra DRAM per protected region [84], where DRAM is often the dominant cloud hardware cost [157]). Thus, guard rows only practically scale to protect small quantities of data (§5.4).

Furthermore, because guard rows still share circuitry with normal rows, increasing Rowhammer susceptibility requires increasing quantities of guard rows for mitigation. For instance, ZebRAM's [84] 50% DRAM overhead at 1 guard row per normal row rises to 80% at a modern requirement of 4 guard rows per normal row on server DIMMs [24, 129].

As we will show (§6), *Siloz* use of subarray groups against inter-VM hammering (a) allows $\approx$ 98.5%–100% of DRAM to be used as normal rows (b) offers fundamental, silicon-level Rowhammer isolation, and (c) accounts for server DRAM addressing in both normal rows and potential guard rows.

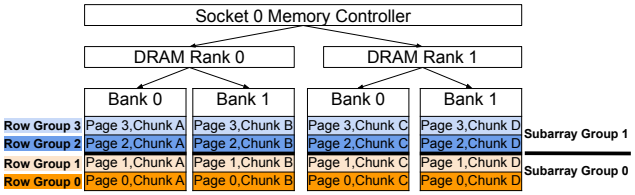

Figure 2. *Subarray groups* in a DRAM hierarchy (§4.1). Ascending physical pages are mapped to ascending *row groups*—and by extension, subarray groups—in a physical node (§4.2). For simplicity, we depict 2 rows per subarray, 1 page per row group, and a monotonically-ascending mapping.

4 Subarray Group Primitive

In this section, we introduce the *subarray group* that *Siloz* uses as a DRAM isolation domain. Recall that each DRAM bank is composed of a set of row-column subarrays, where Rowhammer is ineffective across subarray boundaries (§2.5). Accordingly, the key motivation behind subarray groups is that different VMs occupying disjoint subarray(s) cannot directly hammer each other. We first describe the structure of subarray groups in DRAM (§4.1) before detailing how system-level memory pages map to subarray groups (§4.2).

4.1 Subarray Groups in DRAM

While each individual subarray offers a unit of Rowhammer isolation, *Siloz* opts to provide isolation via *subarray groups*, defining a subarray group as a collection of at least 1 subarray from each bank in a physical[1] NUMA node (e.g., socket). As motivation behind *Siloz*'s use of subarray groups, we consider the challenges of isolation to a single subarray. In particular, allocating a page of memory on a single subarray on modern servers is not practical—if even feasible—due to (a) physical-to-media address mappings that interleave individual pages across a physical node's banks to achieve bank-level parallelism (§2.4), and (b) the performance impact of eliminating such parallelism (e.g., $>$ 18% for some workloads [143]), if such an option is supported in BIOS/firmware.

Overcoming these challenges, *Siloz*' subarray groups' composition from subarrays across every bank in a physical node maintains high throughput and is compatible with physical-to-media mappings. Per the example in Fig. 2, given a fixed subarray size of r rows, subarray group 0 is comprised of rows $[0, r)$ in each of a physical node's banks (i.e., *row groups* $[0, r)$), subarray group 1 of row groups $[r, 2r)$, and so on.

While we do not observe heterogeneously-sized subarrays in *Siloz*'s evaluation platform, subarray group composition can be trivially-adjusted to account for possible heterogeneity [18, 88, 117, 139, 155]. For instance, prior work [117] has observed that a pattern of 3 heterogeneous subarrays repeats every 2048 rows in certain modules. Here, subarray group s

[1]We refer to conventional NUMA nodes (§2.2) as *physical* nodes to distinguish them from *Siloz*'s *logical* nodes (§5.2).

could be composed from the sth *set* of subarrays (e.g., subarrays 0, 1, and 2) in each bank. Composition can be similarly adjusted for modules that activate subarrays in pairs [117].

A subarray group's size is thus the product of a server's banks per physical node, rows per subarray (set), and row size. Banks per physical node and row size are reported to system software. While subarray sizes are *not* reported in DDR4 [60], we have confirmed with a major cloud provider that DRAM vendors can share subarray sizes with them.

Even without (ideal) cooperation from DRAM vendors, one can infer subarray sizes using prior methodologies [24, 166]. We apply the mFIT [24] methodology to *Siloz*'s evaluation platform, observing a pattern of failed Rowhammer attacks at multiples of 1024 rows. Thus, we infer a subarray size of 1024 rows, consistent with prior work [19, 72, 73, 80, 88].

Given the server's 192 banks/physical node and 8 KiB/row, 1024-row subarrays yield a subarray *group* size of 1.5 GiB (192 banks/physical node $*$ 1024 rows/subarray $*$ 8 KiB/row). For subarray sizes in the modern range of 512–2048 rows [155], the group size linearly-increases from 0.75 GiB to 3 GiB. We compare managing different group sizes in §7.4.

4.2 Mapping Pages to Subarray Groups

Subarray group isolation can only work if entire *pages* map to the same subarray group(s). This is because hypervisors—including *Siloz*—provision memory to VMs in pages (§2.1), meaning that a VM is only isolated if its pages reside in the same private subarray group(s). We thus detail how commodity x86-64 physical-to-media addressing maps 2 MiB and 4 KiB pages to single subarray groups, enabling isolation. We then discuss 1 GiB pages, which pose an additional challenge.

2 MiB and 4 KiB Pages. Given 2 MiB alignment in commodity subarray group sizes (with handling of exceptional cases discussed in §6), we exploit the insight that 2 MiB and 4 KiB pages map to a single subarray group on servers that adopt a generally-ascending physical-to-media address mapping. For instance, to a first approximation of Intel's Skylake-based server mappings, row groups are populated in ascending order by ascending page numbers (Fig. 2). Assuming one page per row group for visualization, page 0 maps to row group 0, page 1 to row group 1, and so on.

Considering the finer details of Intel's mapping, increasing page numbers do not *monotonically*-ascend through all row groups, but still result in a layout where 2 MiB and 4 KiB pages map to the same subarray group (maintaining isolation capabilities). Specifically, every n rows groups are populated in *alternating ascending* fashion by two individually-contiguous physical address ranges A and B, where $n = 16$ based on the memory geometry of *Siloz*'s evaluation server (and 16 row groups is 24 MiB of memory: 8 KiB/row $*$ 16 rows/bank $*$ 192 banks/socket). Row groups $[0, n)$ are populated by the first chunk of range A, row groups $[n, 2n)$ by the first chunk of range B, row groups $[2n, 3n)$ by the second chunk of range A, and so on—until repeating with new ranges at a second, 768 MiB-aligned mapping "jump". Crucially, because these chunks align with and encompass entire 2 MiB pages, subarray group isolation remains possible.

1 GiB Pages. The aforementioned address "jump" at 768 MiB-aligned addresses means that 1 GiB (1024 MiB) pages do *not* inherently map to a single subarray group. However, by constructing sets of consecutive subarray groups totaling 3 GiB in size (e.g., 2 sets of 1024-row subarray groups, each 1.5 GiB), we find that at least 1/3 of 1 GiB physical address ranges map to single 3 GiB sets, enabling isolation of associated pages. The remaining 2/3 of memory must be allocated as 2 MiB or smaller pages to preserve isolation.

5 *Siloz* Hypervisor Design

In this section, we present the design of a hypervisor, *Siloz*, built to provide efficient inter-VM Rowhammer protection by placing VMs in private subarray group(s). We first detail *Siloz*'s policy for inter-VM isolation via subarray groups (§5.1) and how *Siloz* introduces *logical* NUMA nodes to manage this policy (§5.2). We then describe a subarray group's lifetime from host boot to shutdown (§5.3). Finally, we discuss integrity protection for the extended page tables (EPTs) that *Siloz* uses to enforce subarray group isolation (§5.4).

For convenient concrete examples, we discuss *Siloz* and its subarray groups in the context of a Linux/KVM baseline hypervisor, *Siloz*'s evaluation server—a dual-socket, 192 DRAM banks/socket (i.e., physical node), major cloud provider-based Intel Skylake configuration—and a commodity subarray size of 1024 rows [19, 72, 73, 80, 88] (as found on *Siloz*'s evaluation server, resulting in a subarray group size of 1.5 GiB, §4). However, *Siloz*'s design principles generalize to other hypervisors, memory geometries, subarray sizes, and—given similar physical-to-media address mappings—CPU vendors; we have verified *Siloz*'s functionality on Intel Skylake and Cascade Lake servers with different memory geometries.

Deployment Environment. In line with our major cloud provider partner, *Siloz* does not share memory among tenants for security and backs guest DRAM with reserved huge pages, which cannot be paged, for high performance [103, 175].

5.1 Subarray Group Isolation: Goal and Policy

Siloz places each VM and the host into private subarray groups, such that the effects of hammering are restricted to one's own domain. *Siloz* correspondingly classifies each subarray group as either *host-reserved* (usable by the host) or *guest-reserved* (usable by exactly one VM).

Siloz decides whether to allocate a page from a particular host- or guest-reserved subarray group based on if the page is *unmediated* as henceforth defined. If a VM can directly access the page (e.g., without a VM exit), the page is unmediated and should be allocated from one of the VM's subarray groups. Intuitively, unmediated pages include those mapped into the VM's address space that will not cause a VM exit

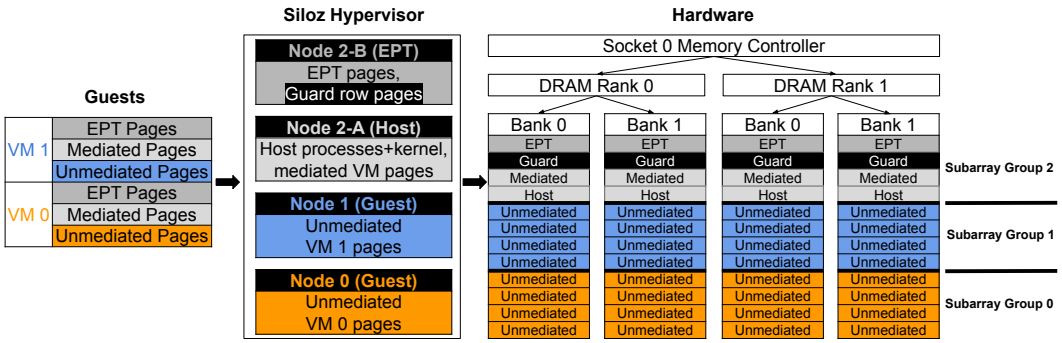

Figure 3. *Siloz* prevents inter-VM hammering by placing specific pages in host- or guest-reserved subarray groups, based on whether a VM has *unmediated* access to the pages (§5.1). *Siloz* abstracts subarray groups as logical NUMA nodes (§5.2) for convenient memory management throughout system lifetime (§5.3). Because extended page tables (EPTs) enforce subarray group isolation, *Siloz* supplementally ensures EPT integrity using emerging hardware extensions [3, 54] or guard rows (§5.4).

for some access type (e.g., guest RAM, guest ROM due to unmediated reads, and select MMIO pages). More specifically, *Siloz* classifies pages based on their existing QEMU memory type [30], indicating which access types trigger exits, if any.

Siloz allocates other pages from host-reserved subarray groups, including mediated, host-only, and EPT pages. We defer discussion of *Siloz*'s protection for EPT pages to §5.4.

The rationale behind *Siloz*'s policy is that a VM can trivially-hammer memory to which it has unmediated access; such memory should therefore be contained to the VM's subarray group(s) to maintain isolation. Conversely, theoretical "confused deputy" hammering by host software (i.e., maliciously-exiting into the hypervisor in a manner that tricks host software into hammering) is a comparatively-difficult—and undemonstrated—attack vector. Thus, host-mediated pages are already relatively-guarded against use for Rowhammer. More importantly, should such confused deputy hammering ever prove feasible, the required VM exit means that the host could easily apply its own mitigation for this hammering (e.g., rate-limiting exit-induced memory accesses).

For guest IO, the current *Siloz* prototype uses paravirtualization (virtio [127]). Thus, the hypervisor manages (i.e., can rate-limit) DMAs on behalf of the guest, meaning the guest cannot issue unmediated DMAs to hammer. To instead support secure passthrough IO (SR-IOV [32]), *Siloz* would need to (1) ensure that the virtual device's IOMMU restricts each guest to perform DMAs within the bounds of the guest subarray groups' address ranges, and (2) protect the corresponding IOMMU page table pages akin to EPT pages.

5.2 Subarray Groups as Logical NUMA Nodes

To manage subarray group isolation, *Siloz* introduces the concept of *logical* NUMA nodes. A logical NUMA node is a memory pool consisting of at least 1 subarray group and is thus a subset of a *physical* NUMA node; logical nodes are hence similar to zNUMA nodes [92].

The key benefit of abstracting subarray groups as (logical) NUMA nodes is that *Siloz* can manage isolation via existing and robust kernel NUMA primitives. Furthermore, similar NUMA support exists across major hypervisors [109, 153, 162], allowing *Siloz*'s design to generalize beyond Linux/KVM.

To preserve physical NUMA semantics alongside logical NUMA extensions, *Siloz* maintains a mapping from each logical node to its physical node. Thus, *Siloz* can support isolation without sacrificing physical NUMA optimizations. For instance, a VM can be comprised of logical nodes from the same physical node to avoid remote NUMA latencies.

Logical nodes consisting of guest-reserved subarray groups are *guest-reserved nodes*. Such nodes comprise all but one logical node per socket and are *memory-only* (i.e., no directly-associated compute resources, §2.2). This design, coupled with a Linux *control group* [33] that limits memory allocations to specific nodes [71], restricts the use of guest-reserved nodes to requests from KVM-privileged processes (§5.3).

The remaining logical nodes correspond to host-reserved subarray groups and are hence classified as host-reserved nodes. Unlike guest-reserved nodes, host-reserved nodes are associated with both subarray group(s) *and* their corresponding socket's cores. Again coupled with a Linux control group, this design restricts *Siloz* to host-reserved nodes by default for both memory allocations and scheduling decisions.

5.3 Lifetime of a Subarray Group

Siloz calculates which physical pages map to which subarray groups during early boot, enabling isolation from boot until shutdown. The number of rows per subarray is passed as a boot parameter. To determine the physical-to-media address mapping (required to map physical addresses to subarray groups), *Siloz* uses its ports of existing drivers [48, 56] for such translations, modified to operate during early boot. Because physical-to-media mappings are fixed based on BIOS settings (§2.4), the calculated subarray group address ranges can be cached across boots in a bootloader or firmware.

Once the subarray group address ranges are loaded, *Siloz* augments existing NUMA topology parsing logic [147] to (a) provision a logical node for each subarray group, and (b) store a mapping from the logical node to its corresponding physical node to preserve physical NUMA semantics (§5.2).

After the required nodes are in place and boot is complete, a privileged user can create a control group with exclusive access to available guest-reserved nodes. A QEMU [8] process, which manages a KVM VM, can then allocate memory on the guest-reserved nodes if the process (a) belongs to the control group, and (b) has KVM privileges. To request this memory, QEMU uses a new UNMEDIATED flag in its mmap() calls for unmediated memory ranges; recall that mediation status is provided by existing QEMU memory types (§5.1). Upon parsing the flag, *Siloz* checks whether the requesting process is permitted to access guest-reserved nodes, and if so, allocates the memory from the appropriate nodes.

During VM execution, *Siloz* avoids potential overheads of managing a large number of nodes by identifying scenarios in which it is unnecessary to iterate over guest-reserved nodes, especially while holding locks. For instance, a guest-reserved node's free memory statistics do not change after VM boot and thus do not require periodic updates [142].

When a VM is shutdown/killed, its backing host memory is freed to the corresponding (logical) nodes' free pools per existing Linux semantics. However, the nodes' reservation remains valid until its encompassing control group is destroyed/modified by a privileged user. We note that there is no modification to host shutdown: the privileged shutdown routine is free to kill any process and its resources, ignoring otherwise active subarray group/logical NUMA constraints.

5.4 Extended Page Table (EPT) Integrity

EPTs pose a distinct challenge to subarray group isolation. Because EPTs define the host physical addresses that a VM may access (§2.1), *Siloz* relies on EPT integrity to enforce subarray group isolation. Thus, unlike other VM data, *Siloz*'s goal of per-VM Rowhammer isolation requires protection of a VM's *own* EPTs, not just inter-VM isolation; EPT bit flips must be prevented or detected-upon-use (integrity-checked).

Hardware-Based Protection. Emerging Intel and AMD servers support *secure EPT* [54], referred to as secure nested paging (SNP) by AMD [3]. With secure EPT, hardware performs mapping integrity checks for EPT entries denoted as "secure", supplementing *Siloz*'s subarray group isolation; the host (e.g., *Siloz*) selects free pages to be used for secure EPTs, while the TDX module/hardware encrypts and integrity checks these pages [20, 53]. While integrity checks only detect—not prevent—EPT corruption, they eliminate the key security threat of EPT bit flips: software cannot use a corrupted EPT to escape subarray group isolation. Nonetheless, depending on how system firmware handles failed TDX integrity checks, a VM may still cause denial-of-service via EPT bit flips; failed SGX checks yield processor lockups [58].

Software-Based Protection. To also provide availability guarantees and support hardware without secure EPT, an alternate solution is needed. Here, *Siloz* exploits the insight that EPTs occupy a small portion of DRAM ($< 0.001\%$ in deployment conditions described shortly), lending themselves to protection via guard rows that *prevent* bit flips (§3).

A basic guard row scheme would reserve an entire subarray group for EPTs, placing n guard rows between each EPT row. However, via insights about server DRAM addressing and VM deployment, *Siloz* can minimize the required DRAM.

In particular, all EPTs can fit into a *single* row group per socket on *Siloz*'s major cloud provider-based server configuration due to several deployment conditions. First, because cloud providers do not typically share pages among tenants for security [92, 140, 151], the number of EPTs is bounded; each host page is mapped in at most one EPT. Second, allocating VMs in contiguous physical memory regions—made feasible by (a) each subarray group's contiguity [175], and (b) static guest memory allocation done for performance [92, 140, 151]—further reduces EPT counts; each last-level EPT can map 512 of the VM's contiguous pages [141]. Third, backing guests with 2 MiB huge pages—again for performance [92, 140, 151]—reduces EPTs by a factor of 512 [141].

In this environment (deployed by multiple major cloud providers [92, 140, 151]), the 512 entries in each last-level EPT page cumulatively map approximately 1 GiB of DRAM, with higher-level EPT pages providing a negligible ($\approx 1/512$) increase in the total number of EPT pages. Since each bank is 1 GiB, and a single 8 KiB row in a bank holds two EPT pages, one row group per socket is sufficient to store all EPTs.

Thus, rather than allocating an entire subarray group for EPTs, *Siloz* reserves a contiguous block of b row groups in a designated subarray group. One row group at offset o in the block serves as the EPT row group, while the other $b-1$ row groups serve as guard rows (roughly split above and below the EPT row group). The host or a VM can accordingly safely use remaining (non-reserved) rows in the subarray group.

In our implementation, *Siloz* specifically uses $b = 32$ and $o = 12$, which reserves just $\approx 0.024\%$ of DRAM for the combination of EPTs and guard rows. At a high level, the specific choices of $b = 32$ and $o = 12$ ensure that an EPT row has a sufficient number on guard rows on both sides to prevent bit flips, in spite of potential DIMM-internal half-row (§2.3) remaps affecting adjacency within 32-aligned blocks. We defer more detailed discussion of such remaps to §6.

To allocate EPTs from appropriate row groups, *Siloz* instruments the host KVM module's kmalloc() calls for EPT pages with a new GFP_EPT flag (get free page EPT). *Siloz* uses this flag in conjunction with the corresponding VM's control group to choose a row group block (implemented, like a subarray group, as a logical NUMA node) for the allocation.

To prevent guard row use, *Siloz* offlines pages mapping to guard rows during system initialization. This behavior extends Linux's system for offlining faulty memory pages [15].

6 Handling Media-to-Internal Mappings

Thus far, we have discussed DRAM row addressing and subarray group isolation in the context of media addresses (via which memory controllers access DRAM, §2.4). However, it is important to consider potential differences in a server DIMM's *internal* mapping of media addresses, such that rows are isolated to expected subarrays. Thus, *Siloz* accounts for various potential sources of row *remaps* in server DIMMs.

Row Repairs. DRAM vendors and cloud vendors can "repair" defective rows by remapping them to spare internal rows that are allocated during manufacturing [4, 21, 51, 52, 61, 63, 75, 75]. Notably, a row's remapped internal address is left up to the DRAM vendor and not exposed to the memory controller, which continues to use the same media address. Such repairs pose a threat to subarray group isolation if they are *inter*-subarray, wherein a defective row could be remapped to a spare row in a *different* subarray group [70].

While our experiments (§7.1) have not yielded evidence of inter-subarray repairs (e.g., many/all repairs may use *intra*-subarray spare rows), *Siloz* can still mitigate the threat of inter-subarray repairs. In the worst-case that a DIMM only uses inter-subarray repairs, the pages mapping to these rows can be identified via address translation drivers and removed from allocatable memory to preserve isolation, as can be done for failing memory pages [15]. We note that only a small portion of rows (e.g., 0.15% [24]) have been observed to be remapped due to row repairs in server DIMMs, meaning little memory capacity would be lost with such a mitigation.

Vendor-Specific Address Scrambling. A subset of major DRAM vendors perform *row address scrambling* [24], transforming bits b_1 and b_2 of the row media address (where b_0 is the least significant bit) by XOR-ing each with b_3. While row scrambling can thus affect the internal *ordering* of a group of 8 rows (bit range $[b_0, b_2]$ encodes 8 rows, where b_1 and b_2 are potentially transformed), it does not affect the internal *contiguity* of these 8 rows; higher-order bits are unchanged.

Thus, for any DIMM whose subarray size is a multiple of 8 rows, there is no impact. In any remaining DIMMs, *Siloz* can remove pages mapping to the 8-row range potentially violating isolation at each subarray boundary from allocatable memory (similar to any inter-subarray repaired rows). For non-multiple-of-8 subarray sizes in the range (512, 2048), this would impact between $\approx 1.56\%$ and $\approx 0.39\%$ of DRAM, respectively (linearly-decreasing with larger subarray sizes).

Standardized Address Mirroring and Inversion. DDR4 specifies [62] two other forms of row address transformations for a specific subset of signals: *mirroring* (for easier signal routing) and *inversion* (for improved signal integrity). Because both have similar ramifications for *Siloz*, we first describe each transformation as depicted in Table 1, before detailing how *Siloz* accounts for them. Again given a modern subarray size of 512–2048 rows [155], we consider transformations of row address bits in the range $[b_0, b_{10}]$ (encoding up to 2048 rows). Notably, transformations of $[b_0, b_{10}]$ and higher-order bits are mutually-independent in DDR4 [62].

Bit	Even Rank		Odd Rank	
	A-side	B-side	A-side	B-side
b_0	b_0	b_0	b_0	b_0
b_1	b_1	b_1	b_1	b_1
b_2	b_2	b_2	b_2	b_2
b_3	b_3	$!\, b_3$	b_4	$!\, b_4$
b_4	b_4	$!\, b_4$	b_3	$!\, b_3$
b_5	b_5	$!\, b_5$	b_6	$!\, b_6$
b_6	b_6	$!\, b_6$	b_5	$!\, b_5$
b_7	b_7	$!\, b_7$	b_8	$!\, b_8$
b_8	b_8	$!\, b_8$	b_7	$!\, b_7$
b_9	b_9	$!\, b_9$	b_9	$!\, b_9$
b_{10}	b_{10}	b_{10}	b_{10}	b_{10}

Table 1. DDR4 address mirroring and inversion [62] of lower-order row media address bits as a function of DIMM rank and "side" (half). Odd-rank addresses are mirrored (red+orange). B-side addresses are inverted (yellow+orange). Lightened colors denote transformed bits. "!" denotes boolean NOT.

In address mirroring, select address bit pairs are mirrored (swapped) in *odd* ranks (red+orange columns in Table 1), while unmodified in even ranks (white+yellow). Specifically, bit pairs $<b_3, b_4>$, $<b_5, b_6>$, and $<b_7, b_8>$ are each mirrored on odd ranks (e.g., 0b**10**000—$b_4 = \mathbf{1}$, $b_3 = \mathbf{0}$—becomes 0b**01**000).

To understand address inversion, recall that each row is internally-split into two half-rows: the A-side and B-side half-rows (§2.3). Bits $[b_3, b_9]$ are inverted in B-side half-rows (yellow+orange), but not in A-side half-rows (white+red).

As with row address scrambling, inversion and mirroring pose a challenge to subarray group isolation only for certain subarray sizes. In particular, if the subarray size is a power-of-2 in the commodity range of 512–2048 rows, inversion and mirroring have no effect on subarray group isolation; for instance, the major vendor's DIMMs in *Siloz*'s evaluation server are unaffected, given their 1024-row subarrays (§4).

For intuition on why power-of-2 sizes work so well, note that the n least-significant bits of a row media address encode the exact number of rows in a subarray of size 2^n. Given sizes of 512 ($n = 9$, $[b_0, b_8]$), 1024 ($n = 10$, $[b_0, b_9]$), and 2048 rows ($n = 11$, $[b_0, b_{10}]$), it is clear from Table 1 that these subarray size-aligned bit ranges are only transformed to different offsets *within* the same subarray, maintaining isolation.

For remaining potential subarray sizes in the commodity range—where inversion and mirroring can cause pages to be split across subarray boundaries—*Siloz* can still provide subarray group isolation by forming "artificial" subarray groups. In particular, *Siloz* can round the subarray size up to the nearest power-of-2, such that the rows in an artificial subarray group maintain the DIMM-internal contiguity property.

Because these artificial boundaries would not always align with true subarray boundaries that provide natural isolation, *Siloz* can instead enforce isolation across artificial bounds by adding n guard rows at the start of each artificial subarray, where $n = 4$ protects against bit flips observed on modern

Parameter	Value
Host Machine	Dual-socket Intel Xeon Gold 6230 CPU @ 2.1 GHz; per-socket: 40 logical cores, 192 GiB DDR4 DRAM (32 GiB 2Rx4 DIMMs @ 2933 MHz, 192 total banks, 1024 8 KiB rows per subarray)
Host OS+Kernel	Ubuntu 22.04+Linux/KVM 5.15 (generic configuration)
Guest OS+Kernel	Ubuntu 22.04+Linux 5.15 (generic configuration)

Table 2. Baseline system configuration. The host kernel is varied among the unmodified Linux/KVM baseline and *Siloz*.

Observed Bit Flips?	DIMM					
	A	B	C	D	E	F
Inside Subarray Group	yes	yes	yes	yes	yes	yes
Outside Subarray Group	NO	NO	NO	NO	NO	NO

Table 3. *Siloz*'s contains bit flips to a hammering domain's subarray group (§7.1), preventing inter-VM hammering.

server DIMMs [24, 87, 129]. Accounting for mappings on different ranks and sides, this would reserve between ≈ 1.56% and ≈ 0.39% of DRAM (again linearly-decreasing with larger subarray sizes). We note that this reservation would be *in place* of any potential reservations for address scrambling, since the artificial subarray size would be a multiple of 8.

Key Takeaways. While commodity power-of-2 subarray sizes integrate most easily, *Siloz* can support other potential subarray sizes by removing the small fraction of pages violating isolation from allocatable memory. Nonetheless, the aforementioned challenges highlight the benefit of hardware-software co-design in DRAM systems [80, 114], especially for subarray group isolation. For instance, these challenges could be overcome by exposing isolation domains such as subarray groups in the DRAM interface, providing architectural guarantees to facilitate *Siloz*'s widespread adoption.

7 Evaluation

We evaluate *Siloz* against a Linux/KVM [82] 5.15 (Ubuntu 22.04 LTS) baseline hypervisor on a major cloud provider-based Intel Skylake server configuration. Unless noted, we use default BIOS settings. Given 192 banks per socket (i.e., physical node) and 1024 8 KiB rows per subarray, *Siloz* manages a subarray group size of $192 * 1024 * 8$ KiB = 1.5 GiB by default. We evaluate the effects of instead managing subarray sizes of 512 and 2048 rows (the lower and upper limits of modern subarray sizes [155]) in §7.4 and distinguish these variants as *Siloz*-512, *Siloz*-1024 (default), and *Siloz*-2048. All *Siloz* variants protect EPTs via guard rows (§5.4). We use the same generic kernel configuration and boot parameters for the baseline and *Siloz*. Table 2 lists our system configuration.

To evaluate *Siloz*'s security, we run an extended version of the Blacksmith [59] Rowhammer fuzzer to flip bits.

To evaluate *Siloz*'s effect on execution time, we run *redis+YCSB* [14, 26] and *terasort* from Hadoop [118] in line with related work [40]. We also run the SPEC CPU 2017 and PARSEC 3.0 benchmark suites used in related work [40, 84, 173].

To evaluate throughput, we run *memcached* [67], SysBench *mySQL* [85], and Intel Memory Latency Checker (MLC) [152], which measures throughput via performance counters.

We run performance benchmarks in an unmodified Ubuntu 22.04 VM using KVM [82] acceleration with QEMU [8] v6.2.0 (i.e., Ubuntu 22.04's version). Select *mmap()* calls are modified to request memory from guest-reserved nodes (§5.3).

VMs are provisioned with all 40 logical cores from socket 0 and 4 GiB of DRAM per logical core (160 GiB total). Multi-threaded workloads are executed with a thread per logical core (40 total), except for PARSEC workloads, which require a power-of-2 number of threads and are thus executed with 32 threads. Guest memory is statically allocated, pinned, not oversubscribed, and backed by 2 MiB host huge pages, as done by multiple major cloud providers [92, 140, 151].

7.1 Security

We assess two aspects of *Siloz*'s security. First, we determine whether *Siloz* can *contain* hammering to a domain's exclusive subarray group(s)—or alternatively put, whether *Siloz* can eliminate inter-VM bit flips. Second, we determine whether *Siloz* can *prevent* bit flips in designated rows (e.g., EPT rows).

We generate bit flips in the baseline system using a modified version of Blacksmith [59] Rowhammer fuzzer (i.e., a fuzzer that attempts to find hammering patterns that induce bit flips despite state-of-the-art hardware mitigations), which we have extended to support server DIMMs. We then compare Blacksmith's effectiveness when running under *Siloz*.

Hammering Containment. We first pin Blacksmith to a *Siloz* subarray group in *Siloz*, where we only detect bit flips in the group, as expected. We left the system running for 24 hours, such that ECC patrol scrubbing would catch any potentially-undetected bit flips. While we observed bit flips in all of the socket's DIMMs (and across ranks and banks in the DIMMs), *none* of the bit flips occurred outside of the subarray group (Table 3). Thus, we confirm *Siloz*'s ability to contain hammering to provisioned subarray groups.

EPT Bit Flip Prevention. To assess *Siloz*'s ability to prevent bit flips in designated rows (e.g., EPT rows), we run Blacksmith with disjoint (a) groups of 32 consecutive logical rows protected according to *Siloz*'s mitigation, and (b) other groups of 32 rows unprotected in the same subarray group. As expected, we do not observe bit flips in the protected rows, while we *do* observe bit flips in the unprotected rows. We also note that all bit flips observed during our hammering containment tests were in *non*-EPT rows. Thus, we demonstrate *Siloz*'s efficacy in *preventing* EPT bit flips.

7.2 Execution Time

We evaluate *Siloz*'s effect on execution time against redis+YCSB, Hadoop terasort, and the SPEC CPU 2017 (SPECspeed) and PARSEC 3.0 benchmark suites. We include all six YCSB

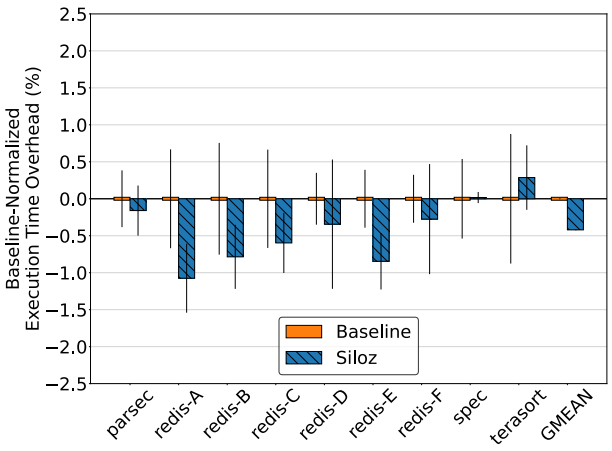

Figure 4. Baseline-normalized execution time (§7.2) for *Siloz*. Error bars depict 95% confidence intervals. Lower is better.

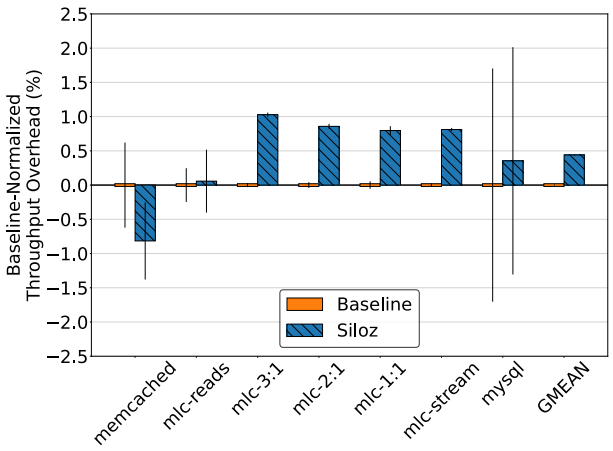

Figure 5. Baseline-normalized throughput (§7.3) for *Siloz*. Error bars depict 95% confidence intervals. Lower is better.

core workloads *A*–*F*. We omit PARSEC's supplemental network benchmarks due to occasional deadlock in the benchmarks on all kernels (i.e., including the unmodified baseline).

As shown in Fig. 4, we find that *Siloz*'s geometric mean timing shows < 0.5% difference from baseline timing, demonstrating *Siloz*'s negligible effect on execution time. Because *Siloz* only affects the location of *boot-time* allocations for each VM, one would not expect significant runtime effects, consistent with our results. Nonetheless, we consider potential sources of varied execution time for completeness.

Beyond the noise present in each benchmark, potential sources of execution time *improvement* for *Siloz* stem from *Siloz*'s relatively-stringent NUMA locality enforcement. For instance, we found that when running *redis-B* in a slightly-modified baseline (i.e., only with *Siloz*'s constraints on EPT allocations), performance slightly improved. We note that better NUMA locality for EPTs is being reviewed in Linux [137].

Potential sources of execution time *worsening* for *Siloz* stem from *Siloz*'s iteration over more (logical) NUMA nodes than the baseline, especially in I/O-bound workloads where the host is more active. However, *Siloz*'s subarray size sensitivity results (§7.4) indicate that noise is a more likely culprit.

7.3 Throughput

We measure *Siloz*'s effects on memory throughput using memcached, Sysbench mySQL, and Intel MLC. MLC workloads are differentiated by all reads (*mlc-reads*), read:write ratios (*mlc-3:1*, *mlc-2:1*, and *mlc-1:1*), and a STREAM triad-like benchmark [108] (*mlc-stream*). As shown in Fig. 5, *Siloz* yields < 0.5% difference from baseline mean throughput.

Factors affecting throughput are similar to those affecting execution time (i.e., *Siloz*'s more stringent NUMA enforcement and management of more nodes). Additionally, both bandwidth and execution time can be affected by address-dependent cache slice/set indexing functions [57, 107, 168]; specific addresses vary between the baseline and *Siloz* due to *Siloz*'s subarray group address range constraints. However, because *Siloz* still manages contiguous regions much larger than those of commonly-optimized access patterns, it is unsurprising that there is no clear performance difference. Given that mean bandwidth and execution time are well-within the confidence intervals of individual benchmarks, we conclude that *Siloz*'s mean differences are insignificant.

7.4 Subarray Size Sensitivity

While we are unaware of modern server DIMMs using the lower bound (512 rows) and upper bound (2048 rows) for conventional modern subarray sizes [155], we can nonetheless measure *Siloz* sensitivity to such sizes by modifying *Siloz*'s *presumed* subarray size (passed as a boot parameter, §5.3).

In particular, because DDR standard access timings do not vary across subarrays [60], and varying subarray sizes does not change the degree of bank-level parallelism available to each subarray group (§4), we can measure *Siloz*'s performance on "artificial" subarray groups without loss of accuracy. We note that such artificial groups do *not* work for evaluating security without additional considerations (§6) because isolation properties *do* change across subarrays.

For clarity, we refer to the "original" *Siloz* variant run on our evaluation's server as *Siloz*-1024 (since the true subarray size is 1024 rows), and compare it to variants *Siloz*-512 and *Siloz*-2048. Notably, *Siloz*-512's smaller subarray group sizes means twice as many logical NUMA nodes as *Siloz*-1024 are needed to represent the correspondingly-larger number of subarray groups. Likewise, *Siloz*-2048's larger subarray group sizes require half as many nodes as *Siloz*-1024.

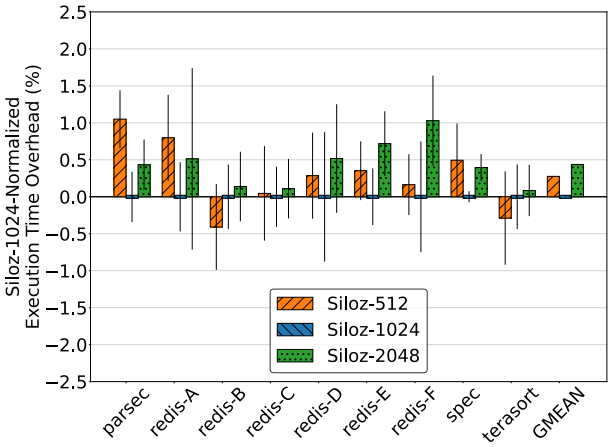

Figure 6. Siloz-1024-normalized execution time when varying from 512 to 2048 row groups per subarray group (§7.4). Error bars depict 95% confidence intervals. Lower is better.

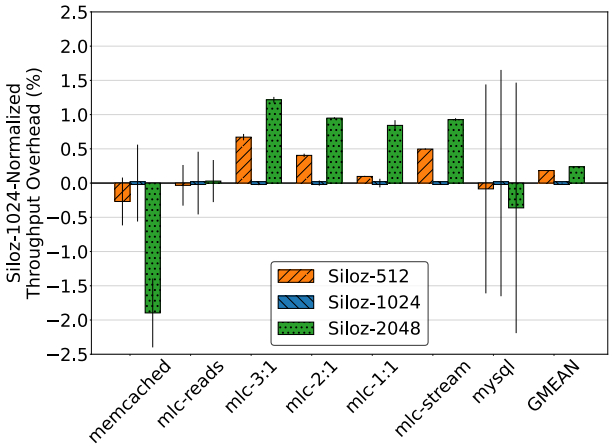

Figure 7. Siloz-1024-normalized throughput time when varying from 512 to 2048 row groups per subarray group (§7.4). Error bars depict 95% confidence intervals. Lower is better.

As shown in Fig. 6 (execution time) and Fig. 7 (throughput), we find $< 0.5\%$ geometric mean overheads for the performance of *Siloz*-512 and *Siloz*-2048 when normalized to that of *Siloz*-1024. The fact that there are no clear trends (nor significant differences) in mean timing and bandwidth as a function of subarray size is expected, given that subarray size does not effect DDR access times nor bank-level parallelism.

Furthermore, the lack of a trend is further indicative that the number of NUMA nodes does not play a significant role in performance, and that the most likely source of performance differences is simply noise. In particular, if NUMA node iterations played a significant role, one would expect the *Siloz* variant with the fewest nodes (*Siloz*-2048) to outperform the one with most nodes (*Siloz*-512), which is *not* the case.

8 Discussion

8.1 Memory Fragmentation

VMs (especially micro-VMs [1, 156]) may have finer-grained DRAM demands than *Siloz*'s subarray group size for a given server configuration. Thus, provisioning an entire subarray group for relatively small needs (e.g., a 1.5 GiB subarray group to a VM needing 512 MiB) risks wasting significant DRAM. The severity of potential fragmentation thus depends on differences between subarray group sizes and granularity of guest DRAM provisioning; multiple major cloud providers offer VM sizing at similar granularity to *Siloz* [110, 140].

More importantly, potential sizing mismatches are not inherent to the subarray group primitive, instead arising from memory controllers' selected DRAM address map. Today's sub-NUMA clustering mapping option [111] can reduce group sizes by 50% to support finer-grained provisioning; the size linearly decreases with the number of banks touched per page (§4.1). In future systems, greater control over physical-to-media address mappings [16, 97] could allow *Siloz* to tailor subarray group sizes to specific VM classes.

8.2 Considerations for Other DRAM Technologies

While today's DDR4 DRAM is widely-deployed, DDR5 and HBM2 DRAM are increasingly-deployed in servers and remain vulnerable to Rowhammer [78, 119]. We thus discuss how differences in DDR5 and HBM2 could impact *Siloz*.

First, memory controllers may use different physical-to-media address mappings for such modules, requiring updated versions of *Siloz*'s drivers for DDR4 DIMMs [48, 56]. Second, DDR5 and HBM2 can provide greater bank-level parallelism than DDR4 by increasing the number of banks per rank (and hence, per physical node). Thus, the upper bound of *Siloz*'s subarray group sizes could proportionally increase in these geometries, yielding coarser-grained memory management (which can be offset using techniques described in §8.1).

In addition to these effects, DDR5 actually eases subarray group isolation for non-power-of-2 subarray sizes. Specifically, DDR5 stipulates that any DIMM-internal address mirroring and inversion (§6) must be *undone* upon arriving at each DRAM device [64], potentially to ease reasoning about DRAM faults/errors. Thus, *Siloz* would not have to create artificial subarray groups for non-power-of-2 subarray sizes in DDR5, as all devices use the same internal addresses.

8.3 Alternate EPT Protection

While *Siloz* provides EPT integrity using guard rows or emerging hardware extensions [3, 54], we emphasize that a variety of EPT bit flip mitigations can help to enforce *Siloz*'s subarray group isolation. For instance, a state-of-the-art SoftTRR-like [173] *software refresh routine* could periodically refresh EPT rows to protect their values against bit flips.

We chose to use guard rows instead of a software refresh routine because of the difficulty of providing real-time guarantees in the Linux kernel [104], especially in many-core, generic production environments. We found that scheduling a software refresh routine to run every 1 ms (as would be required to protect EPT rows via periodic refresh [173]) did *not* consistently meet 1 ms deadlines between refreshes. Rather, we observed a *minimum* of 1 ms between software refreshes due to Linux scheduling semantics [146], even observing a period greater than 32 ms (over 32 times a safe period).

To avoid scheduling delays, we instead ran the refresh routine immediately upon receipt of a periodic timer tick interrupt (i.e., during the interrupt request, rather than as its own task). Notably, this experimental design could interfere with other real-time scheduling, providing a clear drawback.

We also found that we had to deactivate Linux optimizations that disable the timer tick on idle cores for power and performance savings [2, 132, 138, 158]. Despite these changes, we found that the tick interrupts could still be delayed or dropped for various reasons, such as interrupts being disabled. These delayed/dropped tick interrupts resulted in missed refresh deadlines, leaving EPTs vulnerable to bit flips even in the presence of redundant ticks, tick skew [47] across cores, and performance-costly real-time kernel patches.

8.4 Broader Applicability to DRAM Isolation

Siloz uses logical NUMA nodes to manage subarray groups for Rowhammer isolation; however, cloud providers could use logical NUMA nodes to manage other units of DRAM isolation (e.g., banks, ranks, channels, or memory controllers). These units of isolation are attractive in that they could provide VMs with security isolation against additional DRAM timing [122] and power [22] side channels, as well as performance isolation (e.g., memory controller scheduling [112]).

The key challenge to extending *Siloz*'s abstractions beyond subarray groups is the compatibility of physical-to-media address mappings. Given mappings that interleave a memory page across a physical node's banks (§2.4), these forms of isolation are not feasible in default configurations. However, extended addressing control (§8.1) could enable *Siloz*'s application to these units, allowing cloud providers to offer a richer set of isolation options. Alternatively, modifying future DRAM to support subarray-level parallelism [80] could allow subarray groups themselves to offer such properties.

9 Related Work

Rowhammer attacks/analyses. Rowhammer bit flips were publicized in 2014 [79], spurring various attacks and analyses [22–25, 28, 34, 37, 42, 43, 46, 58, 59, 65, 74, 79, 79, 83, 83, 86, 93, 98, 113, 115–117, 119, 120, 123, 125, 129, 133, 144, 145, 149, 150, 163, 165, 172], including the related RowPress phenomenon [101]. These works motivate *Siloz*'s subarray-based approach to DRAM isolation, especially given increasing susceptibility to ACT-induced disturbances inside subarrays.

Rowhammer mitigations. Beyond deployed-but-vulnerable mitigations (§2.5), other hardware and software mitigations offer a range of security-performance trade-offs and are not known to be deployed [5, 6, 9, 11, 12, 17, 31, 35, 36, 39, 41, 44, 45, 50, 66, 68, 74, 76–79, 84, 89, 91, 94, 96, 98, 105, 106, 121, 124, 128, 130, 131, 150, 155, 159–161, 164, 167, 169, 174, 176]. Of these, we analyze mitigations most-related to *Siloz* in §3.

Siloz efficiently prevents inter-VM hammering. However, unlike numerous mitigations, *Siloz only* provides inter-VM protection and not *intra*-VM protection. In fact, *Siloz* can increase intra-VM subarray co-location, potentially simplifying intra-VM Rowhammer. Given inter-VM versus intra-VM exploit severity, we consider this trade-off to be acceptable.

Other DRAM Side Channels. DRAMA [122] shows that DRAM accesses can leak information through timing side channels, such as bank conflicts. HammerScope [22] shows similar leakage through power side channels. Proposed mitigations [29, 136, 154, 177] are largely-based on avoiding simultaneous contention for a DRAM resource (e.g., a bank). Combining these mitigations with *Siloz*'s inter-VM Rowhammer protection is a potential avenue for future work.

Subarrays. mFIT [24] shows that Rowhammer is ineffective across subarrays, as asserted in prior work [96, 164]. X-ray [117] infers subarray structure in select DDR4 and HBM2 modules. *Siloz* builds on these insights to mitigate inter-VM hammering via subarray group isolation. Other work proposes using subarrays for in-DRAM data movement [18, 155] and implementing subarray-level parallelism for DRAM activations [80] and refreshes [166], performance optimizations from which *Siloz*-isolated VMs may benefit.

10 Conclusion

In this work, we have presented *Siloz*, a hypervisor that brings DRAM isolation domains to the cloud via subarray groups. *Siloz* provides VMs with comprehensive protection against inter-VM hammering at negligible performance impact. While today's DRAM standards only allow subarray groups to be used for security, our findings motivate rethinking DRAM addressing's role in both security and performance isolation and considering subarrays for such isolation. We hope that *Siloz*'s effectiveness and practicality spur further development of such DRAM isolation domains.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback and Ishwar Agarwal, Daniel Berger, Tanj Bennett, Lucian Cojocar, Tim Cowles, Brett Dodds, Dan Ernst, Todd Farrell, Adam Grenzebach, Terry Grunzke, Mark Hill, Lisa Hsu, Ingab Kang, Todd Merritt, Onur Mutlu, and Kaiyang Zhao for helpful discussion. This work was supported by an ONR Expeditionary Cyber grant, NSF CNS-2239311, and CCF-2217016. Kevin Loughlin was supported by an NSF Graduate Research Fellowship and a Google PhD Fellowship.

References

[1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight Virtualization for Serverless Applications. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[2] Abdullah Aljuhni, C Edward Chow, Amer Aljaedi, Shaji Yusuf, and Francisco Torres-Reyes. Towards Understanding Application Performance and System Behavior with the Full Dynticks Feature. In *IEEE Computing and Communication Workshop and Conference (CCWC)*, 2018.

[3] AMD. AMD Secure Encrypted Virtualization (SEV), 2020. https://developer.amd.com/sev/.

[4] Rakesh Anigundi, Hongbin Sun, Jian-Qiang Lu, Ken Rose, and Tong Zhang. Architecture Design Exploration of Three-Dimensional (3D) Integrated DRAM. In *International Symposium on Quality Electronic Design*, 2009.

[5] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks. In *ACM SIGARCH Computer Architecture News (CAN)*, 2016.

[6] Kuljit Bains, John Halbert, Christopher Mozak, Theodore Schoenborn, and Zvika Greenfield. Row Hammer Refresh Command, 2015. US Patent 9,117,544.

[7] Majed Valad Beigi, Yi Cao, Sudhanva Gurumurthi, Charles Recchia, Andrew Walton, and Vilas Sridharan. A Systematic Study of DDR4 DRAM Faults in the Field. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2023.

[8] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference (ATC)*, 2005.

[9] Tanj Bennett, Stefan Saroiu, Alec Wolman, and Lucian Cojocar. Panopticon: A Complete In-DRAM Rowhammer Mitigation. In *Workshop on DRAM Security (DRAMSec)*, 2021.

[10] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[11] Carsten Bock, Ferdinand Brasser, David Gens, Christopher Liebchen, and Ahamd-Reza Sadeghi. RIP-RH: Preventing Rowhammer-Based Inter-Process Attacks. In *ACM Asia Conference on Computer and Communications Security (Asia CCS)*, 2019.

[12] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory. In *USENIX Security Symposium (USENIX Security)*, 2017.

[13] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018.

[14] Josiah Carlson. *Redis in Action.* Simon and Schuster, 2013.

[15] Michael Andrew Carlton, Sean P Blanchard, and Nathan A Debardeleben. Improving Memory Error Handling Using Linux. Technical report, Los Alamos National Lab., 2014.

[16] J. Carter, W. Hsieh, L. Stoller, M. Swanson, Lixin Zhang, E. Brunvand, A. Davis, Chen-Chi Kuo, R. Kuramkote, M. Parker, L. Schaelicke, and T. Tateyama. Impulse: Building a Smarter Memory Controller. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 1999.

[17] Anirban Chakraborty, Manaar Alam, and Debdeep Mukhopadhyay. Deep Learning Based Diagnostics for Rowhammer Protection of DRAM Chips. In *IEEE Asian Test Symposium (ATS)*, 2019.

[18] Kevin K Chang, Prashant J Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K Qureshi, and Onur Mutlu. Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[19] Kevin Kai-Wei Chang, Donghyuk Lee, Zeshan Chishti, Alaa R Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu. Improving DRAM Performance by Parallelizing Refreshes with Accesses. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[20] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel TDX Demystified: A Top-Down Approach. *arXiv preprint arXiv:2303.15540*, 2023.

[21] Keewon Cho, Wooheon Kang, Hyungjun Cho, Changwook Lee, and Sungho Kang. A Survey of Repair Analysis Algorithms for Memories. *ACM Computing Surveys (CSUR)*, 2016.

[22] Yaakov Cohen, Kevin Sam Tharayil, Arie Haenel, Daniel Genkin, Angelos D Keromytis, Yossi Oren, and Yuval Yarom. HammerScope: Observing DRAM Power Consumption Using Rowhammer. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[23] Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu. Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[24] Lucian Cojocar, Kevin Loughlin, Stefan Saroiu, Baris Kasikci, and Alec Wolman. mFIT: A Bump-in-the-Wire Tool for Plug-and-Play Analysis of Rowhammer Susceptibility Factors. *Microsoft Tech Report*, 2021.

[25] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In *IEEE Symposium on Security and Privacy (S&P)*, 2019.

[26] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *ACM symposium on Cloud computing*, 2010.

[27] Victor Costan and Srinivas Devadas. Intel SGX Explained. *Cryptology ePrint Archive*, 2016.

[28] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript. In *USENIX Security Symposium (USENIX Security)*, 2021.

[29] Peter W Deutsch, Yuheng Yang, Thomas Bourgeat, Jules Drean, Joel S Emer, and Mengjia Yan. DAGguise: Mitigating Memory Timing Side Channels. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022.

[30] The QEMU Project Developers. The Memory API. qemu.readthedocs.io/en/latest/devel/memory.html, 2022.

[31] Andrea Di Dio, Koen Koning, Herbert Bos, and Cristiano Giuffrida. Copy-on-Flip: Hardening ECC Memory Against Rowhammer Attacks. In *Network and Distributed System Security (NDSS) Symposium*, 2023.

[32] Yaozu Dong, Xiaowei Yang, Jianhui Li, Guangdeng Liao, Kun Tian, and Haibing Guan. High Performance Network Virtualization with SR-IOV. *Journal of Parallel and Distributed Computing*, 2012.

[33] Chris Down. 5 Years of Cgroup v2: The Future of Linux Resource Control. *USENIX Large Installation System Administration Conference*, 2021.

[34] Michael Fahr, Hunter Kippen, Andrew Kwong, Thinh Dang, Jacob Lichtinger, Dana Dachman-Soled, Daniel Genkin, Alexander Nelson, Ray Perlner, Arkady Yerukhimovich, and Daniel Apon. When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer. In *ACM SIGSAC conference on computer and communications security (CCS)*, 2022.

[35] Ali Fakhrzadehgan, Yale N Patt, Prashant J Nair, and Moinuddin K Qureshi. SafeGuard: Reducing the Security Risk from Row-Hammer via Low-Cost Integrity Protection. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2022.

[36] Ali Fakhrzadehgan, Prakash Ramrakhyani, Moinuddin K Qureshi, and Mattan Erez. SecDDR: Enabling Low-Cost Secure Memories by Protecting the DDR Interface. *arXiv preprint arXiv:2209.00685*, 2022.

[37] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[38] Varun Gandhi and James Mickens. Rethinking Isolation Mechanisms for Datacenter Multitenancy. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.

[39] Mohsen Ghasempour, Mikel Lujan, and Jim Garside. Armor: A Run-Time Memory Hot-Row Detector, 2015.

[40] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu. Demystifying Complex Workload-DRAM Interactions: An Experimental Study. In *ACM on Measurement and Analysis of Computing Systems (POMACS)*, 2019.

[41] Hector Gomez, Andres Amaya, and Elkim Roa. DRAM Row-Hammer Attack Reduction Using Dummy Cells. In *IEEE Nordic Circuits and Systems Conference (NORCAS)*, 2016.

[42] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another Flip in the Wall of Rowhammer Defenses. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[43] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in Javascript. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2016.

[44] Hasan Hassan, Ataberk Olgun, Abdullah Giray Yağlıkçı, Haocong Luo, and Onur Mutlu. A Case for Self-Managing DRAM Chips: Improving Performance, Efficiency, Reliability, and Security via Autonomous in-DRAM Maintenance Operations. *arXiv preprint arXiv:2207.13358*, 2022.

[45] Hasan Hassan, Minesh Patel, Jeremie S Kim, Abdullah Giray Yağlıkçı, Nandita Vijaykumar, Nika Mansouri Ghiasi, Saugata Ghose, and Onur Mutlu. Crow: A Low-Cost Substrate for Improving Dram Performance, Energy Efficiency, and Reliability. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2019.

[46] Hasan Hassan, Yahya Can Tugrul, Jeremie S Kim, Victor Van der Veen, Kaveh Razavi, and Onur Mutlu. Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2021.

[47] Red Hat. 2.14. Reduce CPU Performance Spikes. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/html/tuning_guide/reduce_cpu_performance_spikes.

[48] Marius Hillenbrand. Physical Address Decoding in Intel Xeon v3/v4 CPUs: A Supplemental Datasheet. *Karlsruhe Institute of Technology, Tech. Rep.*, 2017.

[49] Marius Hillenbrand, Mathias Gottschlag, Jens Kehne, and Frank Bellosa. Multiple Physical Mappings: Dynamic DRAM Channel Sharing and Partitioning. In *ACM Asia Pacific Workshop on Systems (APSys)*, 2017.

[50] Seungki Hong, Dongha Kim, Jaehyung Lee, Reum Oh, Changsik Yoo, Sangjoon Hwang, and Jooyoung Lee. DSAC: Low-Cost Rowhammer Mitigation using In-Dram Stochastic and Approximate Counting Algorithm. *arXiv preprint arXiv:2302.03591*, 2023.

[51] Masashi Horiguchi and Kiyoo Itoh. *Nanoscale Memory Repair*. Springer Science & Business Media, 2011.

[52] Chih-Sheng Hou, Yong-Xiao Chen, Jin-Fu Li, Chih-Yen Lo, Ding-Ming Kwai, and Yung-Fa Chou. A Built-In Self-Repair Scheme for DRAMs with Spare Rows, Columns, and Bits. In *2016 IEEE International Test Conference (ITC)*, 2016.

[53] Intel. Architecture Specification: Intel Trust Domain Extensions (Intel TDX) Module, 2020. https://www.intel.com/content/dam/develop/external/us/en/documents/intel-tdx-module-1eas.pdf.

[54] Intel. Intel Trust Domain Extensions (Intel TDX), 2022. https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html.

[55] Intel. Introduction to Memory Bandwidth Allocation . intel.com/content/www/us/en/developer/articles/technical/introduction-to-memory-bandwidth-allocation.html, 2022.

[56] Intel. SKX EDAC Linux Driver. github.com/torvalds/linux/blob/master/drivers/edac/skx_base.c, 2022.

[57] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Systematic Reverse Engineering of Cache Slice Selection in Intel Processors. In *IEEE Euromicro Conference on Digital System Design*, 2015.

[58] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. SGX-Bomb: Locking down the Processor via Rowhammer Attack. In *Workshop on System Software for Trusted Execution (SysTEX)*, 2017.

[59] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable Rowhammering in the Frequency Domain. In *IEEE Symposium on Security and Privacy (S&P)*, 2022.

[60] JEDEC. *Double Data Rate 4 (DDR4) SDRAM Standard*, 2014.

[61] JEDEC. *Low Power Double Data Rate 4 (LPDDR4) SDRAM Standard*, 2017. JESD209-4B.

[62] JEDEC. *DDR4 Registering Clock Driver Definition (DDR4RCD02)*, 2019.

[63] JEDEC. *Double Data Rate 5 (DDR5) SDRAM Standard*, 2020.

[64] JEDEC. *DDR5 Registering Clock Driver Definition (DDR5RCD02)*, 2023.

[65] Sangwoo Ji, Youngjoo Ko, Saeyoung Oh, and Jong Kim. Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks. In *ACM Asia Conference on Computer and Communications Security (Asia CCS)*, 2019.

[66] Biresh Kumar Joardar, Tyler K Bletsch, and Krishnendu Chakrabarty. Machine Learning-Based Rowhammer Mitigation. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[67] Jithin Jose, Hari Subramoni, Miao Luo, Minjia Zhang, Jian Huang, Md. Wasi-ur Rahman, Nusrat S. Islam, Xiangyong Ouyang, Hao Wang, Sayantan Sur, and Dhabaleswar K. Panda. Memcached Design on High Performance RDMA Capable Interconnects. In *IEEE International Conference on Parallel Processing (ICPP)*, 2011.

[68] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Maria Eichlseder, Moritz Lipp, and Daniel Gruss. CSI: Rowhammer–Cryptographic Security and Integrity against Rowhammer. In *IEEE Symposium on Security and Privacy (S&P)*, 2023. To appear.

[69] David Kaplan, Jeremy Powell, and Tom Woller. AMD Memory Encryption. *White paper*, 2016.

[70] Brent Keeth, R Jacob Baker, Brian Johnson, and Feng Lin. *DRAM Circuit Design: Fundamental and High-Speed Topics*, volume 13. John Wiley & Sons, 2007.

[71] The kernel development community. NUMA Memory Policy. kernel.org/doc/html/latest/admin-guide/mm/numa_memory_policy.html, 2022.

[72] Jeremie Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu. Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines. In *IEEE International Conference on Computer Design (ICCD)*, 2018.

[73] Jeremie S Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu. D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[74] Jeremie S Kim, Minesh Patel, Abdullah Giray Yağlıkçı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2020.

[75] Jooyoung Kim, Woosung Lee, Keewon Cho, and Sungho Kang. Hardware-Efficient Built-In Redundancy Analysis for Memory with Various Spares. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2016.

[76] Michael Jaemin Kim, Jaehyun Park, Yeonhong Park, Wanju Doh, Namhoon Kim, Tae Jun Ham, Jae W Lee, and Jung Ho Ahn. Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh. *arXiv preprint arXiv:2108.06703*, 2021.

[77] Moonsoo Kim, Jungwoo Choi, Hyun Kim, and Hyuk-Jae Lee. An Effective DRAM Address Remapping for Mitigating Rowhammer Errors. *IEEE Transactions on Computers*, 2019.

[78] Woongrae Kim, Chulmoon Jung, Seongnyuh Yoo, Duckhwa Hong, Jeongjin Hwang, Jungmin Yoon, Ohyong Jung, Joonwoo Choi, Sanga Hyun, Mankeun Kang, Sangho Lee, Dohong Kim, Sanghyun Ku, Donhyun Choi, Nogeun Joo, Sangwoo Yoon, Junseok Noh, Byeongyong Go, Cheolhoe Kim, Sunil Hwang, Mihyun Hwang, Seol-Min Yi, Hyungmin Kim, Sanghyuk Heo, Yeonsu Jang, Kyoungchul Jang, Shinho Chu, Yoonna Oh, Kwidong Kim, Junghyun Kim, Soohwan Kim, Jeongtae Hwang, Sangil Park, Junphyo Lee, Inchul Jeong, Joohwan Cho, and Jonghwan Kim. A 1.1 V 16Gb DDR5 DRAM with Probabilistic-Aggressor Tracking, Refresh-Management Functionality, Per-Row Hammer Tracking, a Multi-Step Precharge, and Core-Bias Modulation for Security and Reliability Enhancement. In *IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023.

[79] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2014.

[80] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2012.

[81] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2018.

[82] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. KVM: The Linux Virtual Machine Monitor. In *Linux symposium*, 2007.

[83] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering From the Next Row Over. In *USENIX Security Symposium (USENIX Security)*, 2022.

[84] Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[85] Alexey Kopytov. Sysbench Manual. *MySQL AB*, 2012.

[86] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. RAMBleed: Reading Bits in Memory Without Accessing Them. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[87] Zhenrong Lang, Patrick Jattke, Michele Marazzi, and Kaveh Razavi. BLASTER: Characterizing the Blast Radius of Rowhammer. In *Workshop on DRAM Security (DRAMSec)*, 2023.

[88] Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu. Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[89] Eojin Lee, Ingab Kang, Sukhan Lee, G Edward Suh, and Jung Ho Ahn. TWiCe: Preventing Row-Hammering by Exploiting Time Window Counters. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2019.

[90] Seunghak Lee, Nam Sung Kim, and Daehoon Kim. Exploiting OS-Level Memory Offlining for DRAM Power Management. *IEEE Computer Architecture Letters (CAL)*, 2019.

[91] Congmiao Li and Jean-Luc Gaudiot. Detecting Malicious Attacks Exploiting Hardware Vulnerabilities Using Performance Counters. In *IEEE Computer Software and Applications Conference (COMPSAC)*, 2019.

[92] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[93] Moritz Lipp, Michael Schwarz, Lukas Raab, Lukas Lamster, Misiker Tadesse Aga, Clémentine Maurice, and Daniel Gruss. Nethammer: Inducing Rowhammer Faults Through Network Requests. In *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2020.

[94] Qi Liu, Jieming Yin, Wujie Wen, Chengmo Yang, and Shi Sha. NeuroPots: Realtime Proactive Defense against Bit-Flip Attacks in Neural Networks. In *USENIX Security Symposium (USENIX Security)*, 2023.

[95] Kevin Loughlin, Jonah Rosenblum, Stefan Saroiu, Alec Wolman, Dimitrios Skarlatos, and Baris Kasikci. Siloz Source Code. github.com/efeslab/siloz, 2023.

[96] Kevin Loughlin, Stefan Saroiu, Alec Wolman, and Baris Kasikci. Stop! Hammer Time: Rethinking Our Approach to Rowhammer Mitigations. In *Workshop on Hot Topics in Operating Systems (HotOS)*, 2021.

[97] Kevin Loughlin, Stefan Saroiu, Alec Wolman, and Baris Kasikci. Software-Defined Memory Controllers: An Idea Whose Time Has Come. In *Wild and Crazy Ideas (WACI) Session at ASPLOS*, 2022.

[98] Kevin Loughlin, Stefan Saroiu, Alec Wolman, Yatin A. Manerkar, and Baris Kasikci. MOESI-prime: Preventing Coherence-Induced Hammering in Commodity Workloads. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2022.

[99] Robert Love. Kernel Korner: CPU Affinity. *Linux Journal*, 2003.

[100] Shih-Lien Lu, Ying-Chen Lin, and Chia-Lin Yang. Improving DRAM Latency with Dynamic Asymmetric Subarray. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2015.

[101] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2023.

[102] Sangkug Lym, Heonjae Ha, Yongkee Kwon, Chun-kai Chang, Jungrae Kim, and Matta Erez. ERUCA: Efficient DRAM Resource Utilization and Resource Conflict Avoidance for Memory System Parallelism. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[103] Jialun Lyu, Marisa You, Celine Irvene, Mark Jung, Tyler Narmore, Jacob Shapiro, Luke Marshall, Savyasachi Samal, Ioannis Manousakis, Lisa Hsu, Preetha Subbarayalu, Ashish Raniwala, Brijesh Warrier, Ricardo Bianchini, Bianca Schroeder, and Daniel S. Berger. Hyrax: Fail-in-Place Server Operation in Cloud Platforms. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2023.

[104] Michael M Madden. Challenges Using Linux as a Real-Time Operating System. In *AIAA Scitech 2019 Forum*, 2019.

[105] Michele Marazzi, Patrick Jattke, Solt Flavien, and Kaveh Razavi. PROTRR: Principled yet Optimal In-DRAM Target Row Refresh. In *IEEE Symposium on Security and Privacy (S&P)*, 2022.

[106] Michele Marazzi, Flavien Solt, Patrick Jattke, Kubo Takashi, and Kaveh Razavi. REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations. In *IEEE Symposium on Security and Privacy (S&P)*, 2023. To appear.

[107] John McCalpin. Address Hashing in Intel Processors. *UT Faculty/Researcher Works*, 2018.

[108] John D McCalpin. STREAM Benchmark. *Link: www.cs.virginia.edu/stream/ref.html#what*, 1995.

[109] Microsoft. Hyper-V Virtual NUMA Overview. *Microsoft Learn*, 2016.

[110] Microsoft. High Performance Computing VM Sizes, 2023.

[111] Srikanta Kumar Mohapatra, Sankararao Majji, Prathipati Ratna Kumar, Ravula Arun Kumar, and Santoshachandra Rao Karanam. Authentication of Sub-NUMA Clustering Effect on Intel Skylake for Memory Latency and Bandwidth. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 2021.

[112] Thomas Moscibroda and Onur Mutlu. Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems. In *USENIX Security Symposium (USENIX Security)*, 2007.

[113] Koksal Mus, Yarkın Doröz, M Caner Tol, Kristi Rahman, and Berk Sunar. Jolt: Recovering TLS Signing Keys via Rowhammer Faults. *Cryptology ePrint Archive*, 2022.

[114] Onur Mutlu. Memory Scaling: A Systems Architecture Perspective. In *IEEE International Memory Workshop*, 2013.

[115] Onur Mutlu. The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017.

[116] Onur Mutlu, Ataberk Olgun, and Abdullah Giray Yağlıkçı. Fundamentally Understanding and Solving RowHammer. *arXiv preprint arXiv:2211.07613*, 2022.

[117] Hwayong Nam, Seungmin Baek, Minbok Wi, Michael Jaemin Kim, Jaehyun Park, Chihun Song, Nam Sung Kim, and Jung Ho Ahn. X-ray: Discovering DRAM Internal Structure and Error Characteristics by Issuing Memory Commands. *IEEE Computer Architecture Letters (CAL)*, 2023.

[118] Jack Norris. Package org.apache.hadoop.examples.terasort, 2013.

[119] Ataberk Olgun, Majd Osseiran, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. An Experimental Analysis of RowHammer in HBM2 DRAM Chips. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2023.

[120] Lois Orosa, Abdullah Giray Yağlıkçı, Haocong Luo, Ataberk Olgun, Jisung Park, Hasan Hassan, Minesh Patel, Jeremie S Kim, and Onur Mutlu. A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chipsand Implications on Future Attacks and Defenses. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2021.

[121] Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, and Jae W Lee. Graphene: Strong yet Lightweight Row Hammer Protection. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2020.

[122] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *USENIX Security Symposium (USENIX Security)*, 2016.

[123] Rui Qiao and Mark Seaborn. A New Approach for Rowhammer Attacks. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016.

[124] Moinuddin Qureshi, Aditya Rohan, Gururaj Saileshwar, and Prashant J Nair. Hydra: Enabling Low-Overhead Mitigation of Row-Hammer at Ultra-Low Thresholds via Hybrid Tracking. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2022.

[125] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In *USENIX Security Symposium (USENIX Security)*, 2016.

[126] Kaveh Razavi and Animesh Trivedi. Stratus: Clouds with Microarchitectural Resource Management. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2020.

[127] Rusty Russell. Virtio: Towards a De-Facto Standard for Virtual I/O Devices. *ACM SIGOPS Operating Systems Review*, 42(5), 2008.

[128] Gururaj Saileshwar, Bolin Wang, Moinuddin Qureshi, and Prashant J Nair. Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation Between Aggressor and Victim Rows. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.

[129] Stefan Saroiu, Alec Wolman, and Lucian Cojocar. The Price of Secrecy: How Hiding Internal DRAM Topologies Hurts Rowhammer Defenses. In *International Reliability Physics Symposium (IRPS)*, 2022.

[130] Anish Saxena, Gururaj Saileshwar, Jonas Juffinger, Andreas Kogler, Daniel Gruss, and Moinuddin Qureshi. PT-Guard: Integrity-Protected Page Tables to Defend Against Breakthrough Rowhammer Attacks. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2023.

[131] Anish Saxena, Gururaj Saileshwar, Prashant J Nair, and Moinuddin Qureshi. AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2022.

[132] Stijn Schildermans, Kris Aerts, Jianchen Shan, and Xiaoning Ding. Paratick: Reducing Timer Overhead in Virtual Machines. In *International Conference on Parallel Processing*, 2021.

[133] Mark Seaborn and Thomas Dullien. Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. *Black Hat*, 2015. See also http://googleprojectzero.blogspot.co/2015/03/exploiting-dram-rowhammer-bug-to-gain.html.

[134] Seyed Mohammad Seyedzadeh, Alex K Jones, and Rami Melhem. Mitigating Wordline Crosstalk Using Adaptive Trees of Counters. In *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2018.

[135] Seyed Mohammad Seyedzadeh, Donald Kline Jr, Alex K Jones, and Rami Melhem. Mitigating Bitline Crosstalk Noise in DRAM Memories. In *International Symposium on Memory Systems*, 2017.

[136] Ali Shafiee, Akhila Gundu, Manjunath Shevgoor, Rajeev Balasubramonian, and Mohit Tiwari. Avoiding Information Leakage in the Memory Controller with Fixed Service Policies. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2015.

[137] Vipin Sharma. NUMA Aware Page Table's Page Allocation. *LWN*, 2022.

[138] Suresh Siddha, Venkatesh Pallipadi, and AVD Ven. Getting Maximum Mileage out of Tickless. In *Linux Symposium*. Citeseer, 2007.

[139] Young Hoon Son, O Seongil, Yuhwan Ro, Jae W Lee, and Jung Ho Ahn. Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations. In *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2013.

[140] Androski Spicer. Deep Dive on Amazon EC2, 2017.

[141] Jovan Stojkovic, Dimitrios Skarlatos, Apostolos Kokolis, Tianyin Xu, and Josep Torrellas. Parallel Virtualized Memory Translation with Nested Elastic Cuckoo Page Tables. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.

[142] Brian K Tanaka. Monitoring Virtual Memory with vmstat. *Linux Journal*, 2005.

[143] Xulong Tang, Mahmut Kandemir, Praveen Yedlapalli, and Jagadish Kotra. Improving Bank-Level Parallelism for Irregular Applications. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2016.

[144] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer. In *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, 2018.

[145] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G Shin. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In *IEEE Symposium on Security and Privacy (S&P)*, 2022.

[146] Linus Torvalds et al. Linux Source Code. https://github.com/torvalds/linux, 2023.

[147] Unified Extensible Firmware Interface UEFI. Advanced Configuration and Power Interface Specification. *ACPI. INFO, Roseville*, 2013.

[148] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L Santoni, Fernando CM Martins, Andrew V Anderson, Steven M Bennett, Alain Kagi, Felix H Leung, and Larry Smith. Intel Virtualization Technology. *Computer*, 2005.

[149] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In *ACM SIGSAC conference on computer and communications security (CCS)*, 2016.

[150] Victor van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. GuardION: Practical Mitigation of DMA-based Rowhammer Attacks on ARM. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2018.

[151] Kirtana Venkatraman. Virtual Machine Memory Allocation and Placement on Azure Stack. *Microsoft Azure*, 2019.

[152] Vish Viswanathan, Karthik Kumar, Thomas Willhalm, Patrick Lu, Blazej Filipiak, and Sri Sakthivelu. Intel Memory Latency Checker v3.9a. *Intel*, 2021.

[153] VMware. Using NUMA Systems with ESXi. *VMware Docs*, 2022.

[154] Yao Wang, Andrew Ferraiuolo, and G Edward Suh. Timing Channel Protection for a Shared Memory Controller. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014.

[155] Yaohua Wang, Lois Orosa, Xiangjun Peng, Yang Guo, Saugata Ghose, Minesh Patel, Jeremie S. Kim, Juan Gómez-Luna, Mohammad Sadrosadati, Nika Mansouri-Ghiasi, and Onur Mutlu. FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2020.

[156] Zicheng Wang. Can "Micro VM" Become the Next Generation Computing Platform?: Performance Comparison Between Light Weight Virtual Machine, Container, and Traditional Virtual Machine. In *IEEE International Conference on Computer Science, Artificial Intelligence and Electronic Engineering (CSAIEE)*, 2021.

[157] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. TMO: Transparent Memory Offloading in Datacenters. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.

[158] Frederic Weisbecker. Status of Linux Dynticks. In *Workshop on Operating Systems Platforms for Embedded Real-Time applications-OSPERT13*. Citeseer, 2013.

[159] Minbok Wi, Jaehyun Park, Seoyoung Ko, Michael Jaemin Kim, Nam Sung Kim, Eojin Lee, and Jung Ho Ahn. SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2023.

[160] Jeonghyun Woo, Gururaj Saileshwar, and Prashant J Nair. Scalable and Secure Row-Swap: Efficient and Safe Row Hammer Mitigation in Memory Systems. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2023.

[161] Xin-Chuan Wu, Timothy Sherwood, Frederic T Chong, and Yanjing Li. Protecting Page Tables from Rowhammer Attacks Using Monotonic Pointers in DRAM True-Cells. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[162] Xen. Xen on NUMA Machines. *Xen Project Wiki*, 2015.

[163] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In *USENIX Security Symposium (USENIX Security)*, 2016.

[164] Abdullah Giray Yağlıkçı, Jeremie S Kim, Fabrice Devaux, and Onur Mutlu. Security Analysis of the Silver Bullet Technique for RowHammer Prevention. *arXiv preprint arXiv:2106.07084*, 2021.

[165] Abdullah Giray Yağlıkçı, Haocong Luo, Geraldo F De Oliviera, Ataberk Olgun, Minesh Patel, Jisung Park, Hasan Hassan, Jeremie S Kim, Lois Orosa, and Onur Mutlu. Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022.

[166] Abdullah Giray Yağlıkçı, Ataberk Olgun, Minesh Patel, Haocong Luo, Hasan Hassan, Lois Orosa, Oğuz Ergin, and Onur Mutlu. HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2022.

[167] Abdullah Giray Yağlıkçı, Minesh Patel, Jeremie S. Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, Saugata Ghose, and Onur Mutlu. BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2021.

[168] Yuval Yarom, Qian Ge, Fangfei Liu, Ruby B Lee, and Gernot Heiser. Mapping the Intel Last-Level Cache. *Cryptology ePrint Archive*, 2015.

[169] Jung Min You and Joon-Sung Yang. MRLoc: Mitigating Rowhammering Based on Memory Locality. In *ACM/IEEE Design Automation Conference (DAC)*, 2019.

[170] Xusheng Zhan, Yungang Bao, Christian Bienia, and Kai Li. PARSEC 3.0: A Multicore Benchmark Suite with Network Stacks and SPLASH-2X. *ACM SIGARCH Computer Architecture News (CAN)*, 2017.

[171] Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang. A Permutation-Based Page Interleaving Scheme to Reduce Row-Buffer Conflicts and Exploit Data Locality. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2000.

[172] Zhi Zhang, Yueqiang Cheng, Dongxi Liu, Surya Nepal, Zhi Wang, and Yuval Yarom. PThammer: Cross-User-Kernel-Boundary Rowhammer Through Implicit Accesses. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2020.

[173] Zhi Zhang, Yueqiang Cheng, Minghua Wang, Wei He, Wenhao Wang, Surya Nepal, Yansong Gao, Kang Li, Zhe Wang, and Chenggang Wu. SoftTRR: Protect Page Tables against Rowhammer Attacks using Software-only Target Row Refresh. In *USENIX Annual Technical Conference (ATC)*, 2022.

[174] Ziyuan Zhang, Meiqi Wang, Wencheng Chen, Han Qiu, and Meikang Qiu. Mitigating Targeted Bit-Flip Attacks via Data Augmentation: An Empirical Study. In *International Conference on Knowledge Science, Engineering and Management*, 2022.

[175] Kaiyang Zhao, Kaiwen Xue, Ziqi Wang, Dan Schatzberg, Leon Yang, Antonis Manousis, Johannes Weiner, Rik Van Riel, Bikash Sharma, Chunqiang Tang, and Dimitrios Skarlatos. Contiguitas: The Pursuit of Physical Memory Contiguity in Datacenters. In *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2023.

[176] Ranyang Zhou, Sabbir Ahmed, Adnan Siraj Rakin, and Shaahin Angizi. DNN-Defender: An in-DRAM Deep Neural Network Defense Mechanism for Adversarial Weight Attack. *arXiv preprint arXiv:2305.08034*, 2023.

[177] Yanqi Zhou, Sameer Wagh, Prateek Mittal, and David Wentzlaff. Camouflage: Memory Traffic Shaping to Mitigate Timing Attacks. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.