# REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations

Michele Marazzi*, Flavien Solt*, Patrick Jattke*, Kubo Takashi†, and Kaveh Razavi*
*Computer Security Group, ETH Zürich †Zentel Japan

***Abstract*—Mitigating Rowhammer requires performing additional refresh operations to recharge DRAM rows before bits start to flip. These refreshes are scarce and can only happen periodically, impeding the design of effective mitigations as newer DRAM substrates become more vulnerable to Rowhammer, and more "victim" rows are affected by a single "aggressor" row.**

**We introduce REGA, the first in-DRAM mechanism that can generate extra refresh operations each time a row is activated. Since row activations are the sole cause of Rowhammer, these extra refreshes become available as soon as the DRAM device faces Rowhammer-inducing activations. Refresh operations are traditionally performed using sense amplifiers. Sense amplifiers, however, are also in charge of handling the read and write operations. Consequently, the sense amplifiers cannot be used for refreshing rows during data transfers. To enable refresh operations in parallel to data transfers, REGA uses additional low-overhead buffering sense amplifiers for the sole purpose of data transfers. REGA can then use the original sense amplifiers for parallel refresh operations of other rows during row activations.**

**The refreshes generated by REGA enable the design of simple and scalable in-DRAM mitigations with strong security guarantees. As an example, we build $\text{REGA}_{\text{M}}$, the first deterministic in-DRAM mitigation that scales to small Rowhammer thresholds while remaining agnostic to the number of victims per aggressor. $\text{REGA}_{\text{M}}$ has a constant 2.1% area overhead, and can protect DDR5 devices with Rowhammer thresholds as small as 261, 517, and 1029 with 23.9%, 11.5%, and 4.7% more power, and 3.7%, 0.8% and 0% performance overhead.**

## I. Introduction

Rowhammer has been a moving target when it comes to mitigations. Worsening Rowhammer thresholds in newer DRAM devices have enabled new access patterns that bypass all deployed in-DRAM mitigations [1]. New Rowhammer effects such as half-double further challenge the design of secure mitigations as they increase the number of potential victim rows for a single aggressor row [2]. The next generation of Rowhammer mitigations will require exceedingly more refresh operations to protect potential victim rows. Unfortunately, the current DRAM architecture provides limited capabilities for additional refreshes.

Since Rowhammer is triggered by DRAM row activations, the only way to scale the number of required refresh operations is during the activations themselves. We present a new in-DRAM mechanism called REfresh-Generating Activation (REGA). REGA time-multiplexes existing DRAM resources using additional lightweight elements, enabling parallel refresh operations while a row is being activated. These parallel refreshes allow for simple, effective, and scalable in-DRAM mitigations against current and future Rowhammer attacks while respecting the respective standards [3], [4].

**Rowhammer attacks.** Rowhammer is part of a broader class of reliability issues known as disturbance errors. Kim et al. [5] showed that Rowhammer affects most DRAM devices in production settings. To trigger Rowhammer, an aggressor row in DRAM must be accessed repeatedly. If this happens frequently enough, bits start to flip in the neighboring victim rows. Follow-up research showed that these reliability errors can compromise systems in a variety of scenarios, most notably, to escalate privileges [2], [6], [7], compromise the browser [8]–[11], phones [12]–[14], clouds [15]–[18], and across the network [19], [20]. Given the severity of these attacks, numerous mitigations have been devised by both academia and industry.

**Mitigations.** Most research in academia proposes to modify the CPU's memory controller (MC) to track aggressor rows, for blocking them before they cause bits to flip in their victims [21], or to send preventive refreshes to their victim rows [5], [22]–[27]. In contrast, in recent years, industry has adopted mitigations that solely operate inside DRAM given the high cost of mitigating Rowhammer inside the CPU [28]. These mitigations track aggressor rows and issue preventive refreshes, also known as Target Row Refresh (TRR), to potential victim rows. Recent academic work provides a formal foundation for designing secure in-DRAM TRR mitigations [29].

**New effects and new patterns.** The additional refreshes that can be generated inside DRAM are scarce, and generating them from the CPU negatively impacts the performance [27]. This forces the mitigations to keep track of aggressor rows to utilize these precious refreshes only when necessary. Tracking aggressors requires assumptions on the behavior of Rowhammer, which is constantly changing with newer technology nodes: the number of required aggressor accesses is rapidly dropping [30] while the number of affected victim rows for a given aggressor is increasing [31]. Researchers were quick to show that these new effects enable new access patterns that evade the mitigations on newer devices [1], [2], [28]. Learning from these experiences, the next generation of Rowhammer mitigations must not tailor their design towards specific (known) behaviors of Rowhammer. The question is how to make this possible given the limited refreshing capability in the current DRAM architecture.

**REGA.** To cleanly decouple the mitigation mechanism (i.e., additional refreshes) from Rowhammer-dependent policies, we should minimize or completely eradicate the state that is necessary for tracking aggressors. Our new in-DRAM mechanism, REGA, makes this possible by allowing parallel

refresh operations whenever DRAM receives an activate command to access a row. Because row activations are the sole cause of triggering Rowhammer, these parallel refreshes are immediately available for mitigating Rowhammer without the need for tracking aggressors. As an example, we have built a simple and scalable in-DRAM mitigation on top of REGA, called $\text{REGA}_\text{M}$, that sequentially refreshes all rows in a DRAM sub-array that receives activate commands.

**Refresh-generating activations.** Modifying DRAM to enable parallel refresh operations is non-trivial due to its highly-optimized architecture and stringent timing requirements dictated by the respective DDR$x$ standards [3], [4]. Without altering the dense DRAM mats (where the data is stored), REGA enables refresh-generating row activations by decoupling row refreshing and data transfer operations performed by the DRAM sense amplifiers. REGA achieves this using a second set of low-overhead buffering sense amplifiers for the sole purpose of supporting data transfers, while the original sense amplifiers are used for parallel refresh operations.

Operating the bitline wires inside the DRAM mats with the additional sense amplifiers complicates the timing of the internal DRAM signals. To ensure the correct operation of REGA, we developed a new accurate DRAM model in collaboration with a DRAM vendor. On top of showing the correctness of REGA in this model, we show that the time between two row activations is sufficient for REGA to perform a single parallel refresh on all 21 DDR4 and 16 DDR5 devices in our test pool. We can further scale REGA to refresh multiple rows during a single activation by increasing the time a row must remain active, defined in the standard as `tRAS`.

Our evaluation using an ASIC implementation of the mitigations on top of REGA, and an analog and cycle-accurate simulation of REGA itself, demonstrates a constant 2.1% area overhead, independent of the degree of Rowhammer vulnerability. The performance and power overhead of REGA depends on the number of refreshes it needs per activation to protect DRAM devices with varying degree of Rowhammer vulnerability. As an example, $\text{REGA}_\text{M}$ protects DDR5 devices with Rowhammer thresholds as small as 261, 517, and 1029 with 23.9%, 11.5%, and 4.7% more power, and 3.7%, 0.8% and 0% performance overhead; regardless of the number of victims per aggressor. These results show that REGA is the first mitigation that can comfortably scale to Rowhammer thresholds that are up to 36× smaller than previously reported (i.e., 9.8 K [30]). Without relying on other mitigations (e.g., ECC), we estimate that REGA can comfortably provide Rowhammer protection for another 10 years. Furthermore, REGA can be scaled to even lower thresholds using the same circuitry by increasing `tRAS` if necessary.

**Contributions.** The following summarizes our contributions:

1. We present REGA, the first in-DRAM mechanism that can scale the number of required refreshes with activations.
2. We develop a new, accurate model of DRAM internals in collaboration with a DRAM vendor. We use this model to ensure the correctness of REGA. To enable future research on DRAM architecture, we release this model as open source to the community: https://comsec.ethz.ch/rega/.
3. We design and implement a new in-DRAM mitigation on top of REGA, called $\text{REGA}_\text{M}$, that can protect against current and future Rowhammer attacks.
4. We evaluate the correctness, performance, area, and power impact of REGA and $\text{REGA}_\text{M}$ using an ASIC implementation, SPICE and cycle-accurate simulations. $\text{REGA}_\text{M}$ has a constant minimal area impact while its performance and power overhead scales to very small Rowhammer thresholds.

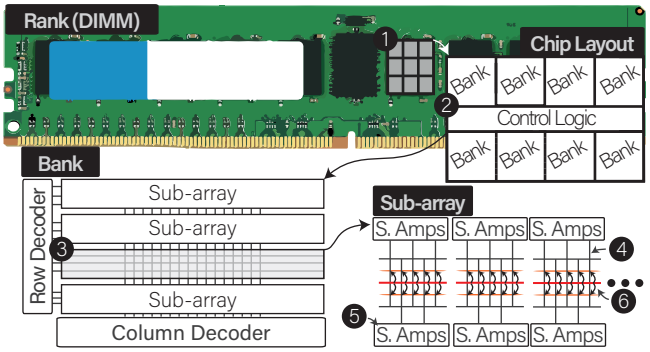


Fig. 1: **DRAM.** Example of multiple DRAM chips on a DIMM (❶). The chip layout is divided into banks (❷) and includes a control logic pad. Each bank is comprised of multiple sub-arrays (❸), in turn composed of mats (❹). Each row spans across an entire sub-array. Each mat is surrounded by bitline sense amplifiers for accessing data inside the mat (❺). Rowhammer affects victim rows adjacent to a repeatedly accessed aggressor row inside a sub-array (❻).

## II. Background and Motivation

We describe DRAM's organization and operation (§II-A), discuss Rowhammer attacks and defenses (§II-B), and the ongoing Rowhammer trends with DRAM technology scaling and what they entail for future defenses (§II-C).

### A. Organization and Operation of DRAM

**Logical Organization.** The DDR$x$ standard [3], [4] describes the logical organization and operation of DRAM devices attached to an external memory controller (typically, in the CPU). Multiple DRAM devices are usually operated simultaneously by the memory controller in a dual in-line memory module (DIMM), as shown in Figure 1. The memory controller can access DRAM by providing an address, which specifies a DRAM bank, a row inside that bank, and a column inside that row that identifies a single byte of data. Internally and abstracted away from the memory controller, each bank is organized into multiple sub-arrays. Each sub-array contains multiple DRAM mats arranged in rows and columns. Each DRAM row consists of capacitors (or cells), each storing one bit of information as an electrical charge. To access a specific column, the entire DRAM row is first selected by a row decoder in the sub-array. The charges in the capacitors of the selected row are then detected by sense amplifiers, collectively known as the row buffer. A column decoder then selects the sense amplifiers associated with the requested column.

**The DDR Protocol.** To access data in DRAM, the memory controller must first activate the row by sending an `ACTIVATE`

(ACT) command to a specific DRAM bank. After sending an ACT, the memory controller must wait tRCD for reliable logical values to become present in the sense amplifiers. Once this happens, the MC can send a READ or WRITE command to a selected column. The data will then be read from or written to the DRAM bus after tCL or tCWL, respectively. DRAM allows one row of a bank to be activated at a time. Thus, if a different row is requested, the memory controller must first deactivate the currently activated row by issuing a PRECHARGE (PRE) command. The PRE is considered complete after waiting for tRP, and the new row of choice can now be activated by sending an ACT for that row. Independently of the DRAM operations, a row can be precharged only after a minimum of tRAS has passed since its activation.

As discussed, DRAM saves data on capacitors that leak charge over time. Therefore, to prevent data corruption, the capacitors' charge is periodically restored via the REFRESH command (REF). REF has to be sent on average every tREFI to keep data integrity. DRAM handles REF as multiple row activations where an incremental index defines a subset of rows that are activated and precharged simultaneously. We will provide more information on the organization and operation of DRAM when describing our accurate DRAM model in §V-C.

### B. Rowhammer

As shown in Figure 1, electrical interferences generated by activating a row (aggressor row) can accelerate charge leakage in physically adjacent rows (victim rows). If the victim rows' charges leak fast enough, i.e., before these rows are refreshed by a REF, DRAM can no longer detect the values that were previously stored in these cells, and bits start to flip. This type of crosstalk was of concern already in 2002 [32] and was demonstrated for the first time in 2014 [5]. Shortly after, security researchers showed many critical attacks based on this effect, now famously known as Rowhammer.

**Attacks.** Rowhammer allows an attacker to compromise the integrity of data stored in adjacent rows that are assumed not be accessible to the attacker. Rowhammer attacks often go through three stages: templating, memory massaging, and exploitation [16]. First, the attacker identifies vulnerable memory locations (templating). The attacker then forces the system to store security-sensitive information in these locations (memory massaging) and retriggers Rowhammer to corrupt the security-sensitive information (exploitation).

Previous research has shown that Rowhammer can be used for local privilege escalation [2], [6], [7], [12], [13], compromising co-located cloud virtual machines [15]–[17], web browsers [8]–[11] and even machines across the network [19], [20]. It has recently even been shown that Rowhammer can increase the strength of Spectre attacks [33].

**Defenses.** Common mechanisms to mitigate Rowhammer include preventive refreshes to victim rows, which are broadly known as Target Row Refresh (TRR) [28], isolation of sensitive data, and error-correcting codes (ECC). Isolation of sensitive data requires software changes, making it challenging to deploy in practice. ECC complicates Rowhammer attacks but does not prevent them [18]. Furthermore, the increasing density of bit flips in newer technology nodes makes secure ECC schemes rather expensive [34]. As a result, TRR is the only specific mitigation that is currently deployed for tackling Rowhammer.

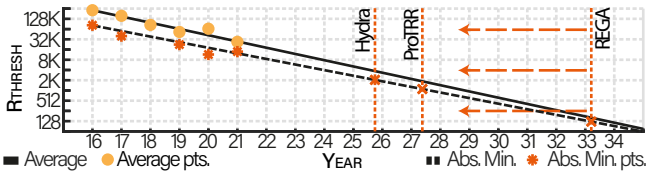


Fig. 2: **Trend of $R_{thresh}$ from 2016 to 2035 (B=4).** The curve represents the minimum number of activations to induce bit flips, both for the absolute minimum values and for the average. Vertical orange lines report the minimum $R_{thresh}$ for ProTRR [29], Hydra [27] and our mitigation, REGA. For REGA, the minimum depends on its configuration, as later discussed in the paper. The fitting is based on previous work [34], [42]. We refer the reader to Appendix A for details about the mitigations' threshold and the estimation methodology.

TRR can be implemented in software [13], [19], [35]–[37], in the CPU's MC [5], [21]–[27], [38], and inside DRAM [29], [39]–[41]. Mitigating Rowhammer is costly, making its deployment unattractive for software and CPU vendors. Consequently, TRR is currently deployed inside the component affected by Rowhammer, which is DRAM itself. Unfortunately, recent work shows that all existing implementations of TRR in DDR4 DRAM devices are vulnerable to special access patterns [1], [2]. This is due to the limited availability of additional refresh operations to perform TRR in the current DRAM architecture and the increasing cost of tracking potential aggressor rows for using these refresh operations judiciously. New Rowhammer effects due to technology scaling further accentuate the cost of future mitigations.

### C. The Impact of Technology Scaling on Rowhammer

There are two Rowhammer trends as DRAM moves to smaller technology nodes: the worsening Rowhammer vulnerability and the Rowhammer impact of a single aggressor. As we show, these factors have an heavy impact on existing Rowhammer mitigations.

**The worsening Rowhammer thresholds.** The original Rowhammer study with DDR3 DRAM [5] measured the minimum number of activations required to induce bit flips (i.e., $R_{thresh}$) to be in the order of 139 K to 284 K. A more recent study showed that the $R_{thresh}$ is dropping with each new generation of DRAM devices, reaching as few as 9.6 K activations [30]. Figure 2 shows that $R_{thresh}$ has dropped almost 15 times since Rowhammer was first demonstrated and is posed to drop even further with future generations of DRAM devices. Figure 2 further shows that proposed mitigations can only cope for a few more years with the dropping $R_{thresh}$. This is due to the fact that they need to keep a non-trivial amount of state to use the available row refreshes without significantly sacrifying performance. We show that our proposed mitigation, REGA, provides *better scalability* against Rowhammer in future devices by reducing the cost of refreshes inside DRAM.

**The increasing impact of a single aggressor.** An aggressor row impacts the victim rows directly adjacent to it. This means that the aggressor row has a blast diameter ($B$) of 2 victim rows. Recent work [2] shows that the blast diameter has increased

to 4 in more recent DRAM devices. The blast diameter is posed to further increase, and we are likely to see devices with blast diameters of 6 or even higher based on a recent JEDEC specification [43]. None of the existing mitigations has so far considered blast diameters of 6 or above. We show that the refreshes generated by our new in-DRAM mechanism enable the construction of mitigations that are *independent of the blast diameter*.

## III. Threat Model

We assume that the CPU's MC complies with the respective DRAM standard. We assume the software to be untrusted: the attacker can send ACT commands to target DRAM rows through native code execution [6], [7], [12], [13], [15], [16], a browser tab (e.g., running JavaScript) [9]–[11], or over the network [19], [20] to trigger Rowhammer bit flips that compromise the target system. We assume that DRAM is vulnerable to Rowhammer and that bits start to flip if (aggressor) rows receive a certain number of ACT commands before the standard refresh mechanism in a tREFW refreshes their victims. All recent DRAM devices are reported to be vulnerable to Rowhammer [1], [30]. While some Rowhammer effects are known (e.g., half-double [2]), we assume more will be discovered over time, and our proposed mitigation should be able to cope with that.

## IV. Overview

Our goal is to design a new mechanism that can scale the number of additional refreshes as needed to defend against current and future Rowhammer attacks. We first overview the requirements for this new mechanism, called REGA, before discussing the challenges in designing it.

### A. Requirements

We define our requirements (**R1**–**R3**) around scalability, (forward) security, and practicality.

**Refresh scaling.** The DDR4 protocol only allows issuing extra refreshes during REFs [44]. The DDR5 protocol improves this by introducing the RFM command [4], [29], which can periodically send additional refreshes to help mitigate Rowhammer. However, its periodic nature requires mitigations to keep a state (e.g., counters) to decide which victims to refresh. As discussed before (§II-C), maintaining this state becomes more expensive with smaller technology nodes. Further, shortening the RFM period to allow for more refreshes would negatively impact performance. Instead, an ideal mechanism should allow scaling REFs as needed, up to generating an extra REF, or more, for each ACT received by the DRAM device.

> **Requirement (R1).** REGA should allow scaling of extra REFs as needed.

**Forward security.** Deployed mitigations do not decouple mechanisms from policies. This impedes their adaptability to new Rowhammer effects, such as the rapidly dropping $R_{thresh}$ with smaller technology nodes [30] and the half-double effect [2]. The main reason is the need for careful state management in hardware due to the scarcity of extra REFs. Hence, as a second requirement, we define the possibility of configuring simple (stateless) policies to mitigate current and future Rowhammer effects.

> **Requirement (R2).** REGA should decouple its mitigation mechanism from policies to counter current and future Rowhammer effects.

**Practicality.** An in-DRAM solution must meet two requirements to make its deployment practical. First, the new REF mechanism should have a small impact on current and future technology nodes. Requiring more area with smaller technology nodes will reduce the benefits of using the smaller technology nodes in the first place. Moreover, the DRAM mats should not be changed since they are the most important and highly optimized building block of today's DRAM chips. Second, a new mitigation should operate within the bounds defined by the DDR standards [3], [4] as protocol changes are a multi-year effort requiring consensus among all involved parties.

> **Requirement (R3).** REGA should have a minimal impact on the area and layout of the DRAM while operating inside the bounds of the respective DDR standard.

Next, we discuss how fulfilling these requirements introduces challenges in the design and implementation of REGA, and summarize how we address these challenges in the rest of the paper.

### B. Challenges

REGA requires changing certain DRAM elements to achieve low-cost and scalable refresh generation. To ensure that our changes do not compromise the device's functionality, it is paramount to use an accurate DRAM model. Currently, such a model does not exist, which brings us to our first challenge:

> **Challenge (C1).** Deriving an accurate model of DRAM internals that represents modern DRAM devices.

We address this challenge in §V by explaining the details of the internal DRAM architecture and presenting REGA Model (REM), an accurate DRAM model that we developed in collaboration with Zentel Japan. Furthermore, we show that REM captures timings and internal signal propagations more precisely than the previous state-of-the-art DRAM model. Equipped with REM, we seek to understand and modify the DRAM architecture to fulfill requirements **R1** and **R3**.

So far, extra refreshes have been generated only in response to REF or RFM commands, which are periodic and do not directly scale with a decreasing $R_{thresh}$. Hence, to fulfill **R1**, the DRAM chip must be able to generate these extra refreshes as part of ACTs. To fulfill **R3**, these extra refreshes should not change the DRAM timings, as defined in the respective standards [3], [4]. Consequently, we propose generating these refreshes in parallel with ACT commands.

> **Challenge (C2).** Generating refreshes in parallel with ACTs.

In §VI, we discuss possible placement options for REGA and explain involved timings, leading us to its final design. Because we need direct control of rows, REGA is deployed

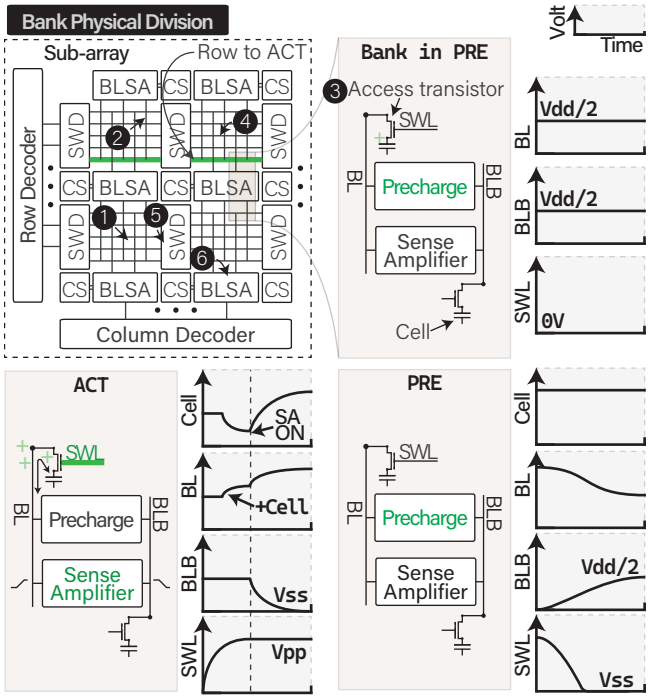


Fig. 3: **DRAM's internal physical organization.** Top left: DRAM bank composition. Top right: voltage levels when a bank is in the precharged state. Bottom left: voltage levels when a row is activated. Bottom right: voltage levels when going from an activated to a precharged state. The voltage levels under these different states are discussed in §V-B.

next to the mats. To perform extra refreshes in parallel with every ACT, REGA must finish before a PRE. We describe the additional circuitry to achieve this capability.

REGA's implementation needs to time-multiplex wires in the mats, which implies that more operations must be performed during tRAS. Through experiments conducted on real DDR4 and DDR5 devices in §VII, we show that this is already feasible in today's existing DRAM designs. To protect devices with ultra-low $R_{thresh}$, REGA requires increasing the tRAS further.

With REGA in place, we address **R2** by building a secure, configurable, and future-proof mitigation against Rowhammer.

**Challenge (C3).** Building a secure and configurable mitigation on top of REGA.

In §VIII, we present REGA$_M$, a new deterministic mitigation on top of REGA that protects against all known and **future** Rowhammer attacks by relying on the sole assumption that Rowhammer is induced by activations. REGA$_M$ is secure against new Rowhammer effects by design, and its power consumption can be configured depending on the $R_{thresh}$.

## V. Accurate Modeling of DRAM

We now describe the details of DRAM's internal physical organization (§V-A) and focus on the sense amplifier's operation (§V-B). Then, we present our novel DRAM model and compare it with the state-of-the-art DRAM model [45] (§V-C). A summary of all abbreviations and symbols introduced in this section can be found in Appendix B.

### A. Physical Organization

In DRAM, data is saved in cells as a full charge (Vdd) or an empty charge (Vss). Depending on the encoding, either one of the two charge levels can correspond to a logical one (1b) [46]. Physically, cells are organized in compact, hierarchical, and matrix-based structures as shown in Figure 3. These structures (*mats*) are composed of $512 \times 1024$ cells (❶), and each cell is connected to a column bitline (❷), via an *access transistor* (❸) [42], [46], [47]. Along a mat's row, all access transistors share the same control line, the *sub-word line* (SWL) (❹), which is raised upon row activation by the *sub-word line driver* (SWD, ❺) [48], [49]. Functionally, the SWL corresponds to the logical row selector. A single row address uses multiple mats. Multiple rows, typically 512, form a sub-array and multiple sub-arrays form a bank [50], [51].

The bitlines are connected to bitline sense amplifiers (BLSAs, ❻) placed on top or on the bottom of the mat in an alternating manner. Each BLSA is connected to two bitlines, one coming from the mat above (BL) and one from the mat below it (BLB). During the sensing operation, one bitline transmits information, and the other is used as a reference voltage. This *open bitline* design reduces crosstalk between bitlines and improves space efficiency. Along a mat's row, each cell has a unique bitline, but along a column, all cells share the same bitline.

**Activation process.** After the memory controller sends an ACT, the associated SWLs are raised by the SWDs. All the cells of the row get connected to the BLSAs, which are subsequently activated. The BLSAs read and amplify the stored data; as they read it, they also recharge the cells. After the sensing operation stabilizes to a logical value, data can be read (or written) by specifying a column. The column's data is obtained by connecting one byte from each mat to the local I/O (LIO). The LIO precedes the global I/O (GIO), and other data manipulation, such as further sensing operation (I/O sense amplifier), serialization, and I/O line drivers [52].

### B. Sense Amplifier

We now describe the bitline sense amplifier and its operation. In DRAM, the cell capacitance is in the order of femtofarads (fF) [53], which allows for high-density memories. However, because of the low capacitance, sense amplifiers are needed to amplify the cell's value to become interpretable. Moreover, sense amplifiers are also used to restore the capacitors' charge. A sense amplifier is a circuit with two inputs: BL and BLB (Figure 4-center). When the sense amplifier is activated, it *senses* the voltage difference between BL and BLB, amplifies it, and obtains its rail-to-rail output. That is, if $V(\text{BL}) > V(\text{BLB})$ for voltage $V$, then BL is raised to Vdd and BLB is brought to Vss (whose voltage levels are given in Appendix C). These values are then held or *latched*. As described before, only BL or BLB will be connected to a cell.

**Charge sharing.** Before the first ACT reaches a bank, the bank is in a precharged state (Fig. 3, "Bank in PRE"). Consequently, all SWLs are low, and all BLSAs are OFF. A precharge circuit keeps the bitlines' voltage at Vdd/2. After an ACT is received (Fig. 3, "ACT"), the precharge is turned off, allowing the

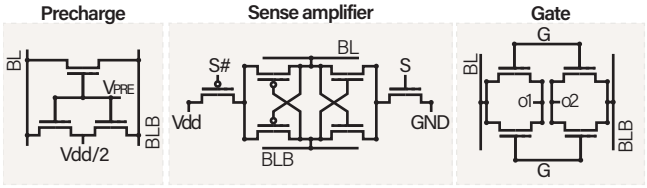


Fig. 4: **Relevant electronic circuits.** Left: the precharge circuit is used to bring the bitline to the reference voltage (Vdd/2). Center: the sense amplifier circuit is used to sense and amplify the charge inside the cell's capacitor and recharge it when needed. Right: the gate circuit is used as a multiplexer in the design of REGA, as discussed in §VI.

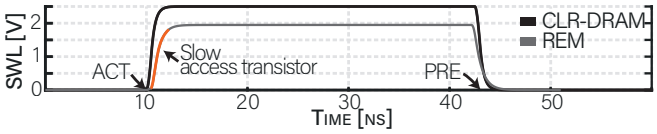


Fig. 5: **DRAM model comparison (SWL).** CLR-DRAM inaccurately assumes an optimistic rise of the SWL. This results in faster (de-)activation of the access transistor.

bitline's voltage to change. Then, the corresponding SWLs are raised to connect the cells to the bitlines.

For a given BLSA, when the access transistor is activated the bitline voltage varies depending on the charge stored in the cell. As the other bitline connected to the BLSA remains at Vdd/2, the difference between BL and BLB can be amplified by activating the sense amplifier (Fig. 4-center, signals S/S#). Because the bitline has a high parasitic capacitance, the cell's value is lost during this operation, which is called *charge sharing*. The sensing operation amplifies the signal to enable retrieving its logical value, and restores the charges in the cells.

**Precharge.** When PRE is received (Fig. 3, "PRE"), the SWLs' voltage is lowered, thus disconnecting the cells. Now the BLSAs are driving the bitlines to either Vdd or Vss. At this point, if another row were to get activated, the values in this other row would get corrupted [54]. To avoid this, after having turned the sense amplifier to OFF, the precharge circuit (Figure 4-left) brings the bitlines back to Vdd/2.

**Timings and parasitics.** The operations we described are delayed by parasitic elements of the lines (i.e., resistance and capacitance). For example, the SWL's parasitics slow down the (de-)activation of the access transistors. The bitline's parasitics reduce the sensed voltage, slow down the signal propagation, and increase the time required for the precharge operation. These effects are the key elements for a correct and realistic DRAM simulation. Yet, no accurate and up-to-date circuit description based on real devices exists today.

We collaborated with a DRAM vendor to overcome this limitation by designing an accurate model called the REGA Model (REM), using details from a real DRAM design. Next, we provide additional details about REM and show that it captures DRAM details that existing models [45], [48] do not. We open source REM to enable further research on DRAM architecture and its security in the following URL: https://comsec.ethz.ch/rega/.

### C. REM

In the following, we present REM and compare it with the state-of-the-art model described in CLR-DRAM [45]. In CLR-DRAM, the authors propose DRAM architecture modifications to improve performance. To derive our accurate model, REM, the DRAM vendor provided us with real physical values obtained from DDR4 devices and the circuit we use in this work. REM and CLR-DRAM differ substantially.

TABLE I: **Identified issues in CL-DRAM compared to REM.** Summary of differences (DIFF) and inaccuracies (IN) of CL-DRAM compared to REM.

| Property | DIFF | IN | Explanation |
|---|---|---|---|
| Transistors sizes | | ✔ | Inaccurate ratios |
| Voltage sources | ✔ | | Simplified sources |
| Control voltages | ✔ | | Simplified control |
| Line parasitics | | ✔ | Low parasitics |
| SWL drivers | | ✔ | Not modeled |
| LIO & MIO loads | | ✔ | Not modeled |

Overall, accurately describing DRAM requires knowing the (i) transistor dimensions, (ii) source voltages, (iii) control voltages, and (iv) line parasitics. Table I summarizes our findings. We divide our comparison with CLR-DRAM into inaccuracies (**IN**) and circuit differences (**DIFF**). Inaccuracies are fundamental as they affect the simulation's validity. Differences in the circuit may not always be critical as they could emerge from alternative but correct DRAM topologies. Lax timing constraints are easier to comply with, and the resulting power consumption will generally be underestimated. More importantly, an inaccurate model does not have enough fidelity to confirm the feasibility of proposed DRAM modifications. We now describe the differences between REM and CLR-DRAM.

**Sub-word line (IN).** A major inaccuracy of CLR-DRAM is the SWL's characterization. First, this model considers SWL as a single element instead of a transmission line. Second, the model underestimates the line resistance and parasitic capacitance. Lastly, the SWDs are not modeled at all, heavily impacting the reliability. We observed that SWL's rising time, illustrated in Figure 5, is one of the slowest in the circuit when modeled accurately. A slow SWL delays the activation of the access transistor, which further delays the sense amplifier's activation. Evaluating the SWL's *actual* timing is essential to assess the feasibility of performing REGA's parallel refresh within tRAS. Likewise, the SWL timing affects most architectural modifications as it affects the precharging and activation speed.

**Voltages (DIFF).** CLR-DRAM considers only two voltage sources in the circuit: 1.2 V and 2.5 V. Our model describes voltages controlled by 1 V, 1.1 V, 1.4 V, 1.5 V and 2.5 V sources (see Appendix C). Modeling incorrect voltages may affect switching speeds, noise robustness, and power consumption. Our bitlines are referenced to a maximum of 1 V, which makes the sensing operation more difficult yet realistic.

**Sense amplifier's transistor ratios (IN).** Transistors ratios characterize the speed of operation of the sense amplifiers, reflecting in noise sensitivities and timings. For the sense amplifiers, CLR-DRAM uses overly optimistic ratios and optimistic absolute transistor sizes. In other cases, CLR-DRAM uses pessimistic ratios.

**Data path (IN).** We modeled the local and global I/O lines, which all previous models omitted. LIO and GIO lines act as a load to the BLSA during read/write operations.

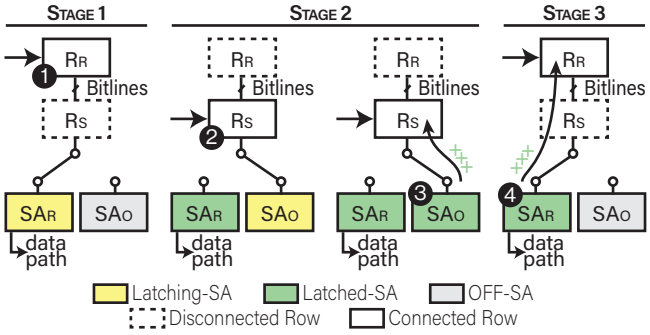


Fig. 6: **REGA's concept.** Row requested ($R_R$) is activated and REGA sense amplifier ($SA_R$) starts latching its charge (❶). While $SA_R$ latches the value, $R_R$ and $SA_R$ are disconnected from the bitline. Row shadow ($R_S$) and the original sense amplifier ($SA_O$) are activated (❷). $R_S$ is recharged by $SA_O$ (❸). $SA_R$ and $R_R$ are connected, and $R_R$ is recharged (❹).

**Sense amplifier circuit (DIFF).** So far, literature has kept adopting the textbook DRAM model [48] where the BLSAs are supplied with a single voltage Vdd. However, in reality modern DRAM implementations may differ, with BLSAs that can be overdriven [51], [55], [56]. The DRAM vendor confirmed that they use overdriven BLSAs, and we designed our DRAM model to implement overdriven BLSAs as described by them. Such BLSA can switch between the common cell high voltage of 1 V and a higher (overdrive) voltage of 1.4 V. The goal of the overdrive voltage is to accelerate the first phase of sense amplification, while the lower 1 V, used in the second phase, saves energy and avoids overcharging the cell.

## VI. REGA

Equipped with REM, we proceed to the design of REGA. Parallelizing an additional DRAM operation next to an ongoing one has (to the best of our knowledge) not yet been explored. We provide a high-level description of how REGA achieves this in §VI-A, before discussing implementation details and internal signal timing control in §VI-B and §VI-C, respectively. We also discuss supporting multiple refreshes per ACT in §VI-D.

### A. High-Level Operation

To parallelize DRAM operations, one obvious direction is doubling the bitlines to make cells in different rows accessible at the same time. Unfortunately, this introduces a significant per-cell area overhead in the otherwise highly-optimized DRAM mats. This means that to remain area-efficient, REGA must multiplex the bitlines during the parallel operations.

**Time-multiplexing the bitline.** The most effective scenario for a Rowhammer attack is one that maximizes the number of activations by simply alternating ACT and PRE on an aggressor row [22], [29]. This means that REGA should generate the necessary refresh either during ACT or PRE. In the same bank, the time between a PRE and an ACT is tRP, typically 13.75 ns. As discussed in §V, DRAM requires this time to bring the bitlines back to the reference voltage. Therefore, we cannot use tRP for our parallel operation.

The time between an ACT and a PRE is defined as tRAS, which is at minimum 32 ns. The row activation at the beginning of tRAS also uses the bitlines. However, since tRAS is relatively long, we can leverage it to multiplex the bitlines for our parallel operation in REGA.

**Reads and writes.** During bitline multiplexing, the memory controller will send either READ or WRITE. As described in §V, these commands can be sent tRCD after an activation, which is less than tRAS. REGA should comply, allowing reads and writes while refreshing an additional row in parallel.

**Shadow refresh.** REGA allows normal DRAM operations on a requested row while simultaneously enabling refreshing another row. We denote the requested row by $R_R$ and the shadow row by $R_S$ (i.e., the row that gets refreshed in parallel). One may propose using the bitline sense amplifiers to first read data from $R_R$ (as part of ACT) and then to refresh $R_S$. However, after $R_R$ has been activated, the memory controller can read from/write to any column at any given time. Therefore, there should be two sets of sense amplifiers: the original sense amplifiers ($SA_O$) to perform refresh operations, and a new set of sense amplifiers, called the REGA sense amplifiers ($SA_R$), to hold the values that could be read (or written). In practice, $SA_R$ acts as a buffer to the original $SA_O$.

Figure 6 shows the high-level operation of REGA. To enable time-multiplexing of the bitlines, we make a key observation that for the sense amplifiers to start their mechanism, **charge sharing** is the only required operation (❶). This step provides the logical values that can be read by the memory controller in time with a tRCD. After the charge sharing phase, the bitlines can be disconnected and connected to a different row and sense amplifiers (❷). When the second set of sense amplifiers is connected, REGA refreshes the shadow row (❸). After the shadow refresh, the $R_R$ is reconnected (❹), allowing any operation conforming to the standard to be executed identically to what is possible after a normal ACT on $R_R$. In other words, REGA does not require changing the DRAM standard, hence satisfying **R3**. The time available to perform both refreshes is tRAS; after this, the memory controller can send a PRE. If tRAS is preceded by a read, the value can be read by the sense amplifier that had latched the logical value. If tRAS is preceded by a write, the new value will replace the one latched by $SA_R$, later used to finish refreshing $R_R$.

### B. Low-Level Operation

REGA requires replicating existing elements in the sub-array's circuit as shown in Figure 8. In particular, additional sense amplifiers and transmission gates (Figure 4-right) are required. No modification to the data path is necessary. We now describe REGA's operation for a single sense amplifier, as the same applies to all of them. We assume that the device receives an ACT for $R_R$, which may or may not be part of an attack, and we assume that the shadow refresh targets $R_S$. REGA's complete design for a single cell is in Figure 7.

**Circuit's basics.** For each bitline, two sets of sense amplifiers are used: the already present, "original" sense amplifier ($SA_O$), and our addition, the REGA sense amplifier ($SA_R$). $SA_R$ latches the requested row's value and serves as an I/O buffer. $SA_O$ refreshes the shadow and requested rows. REGA uses the existing circuit to precharge the bitline.

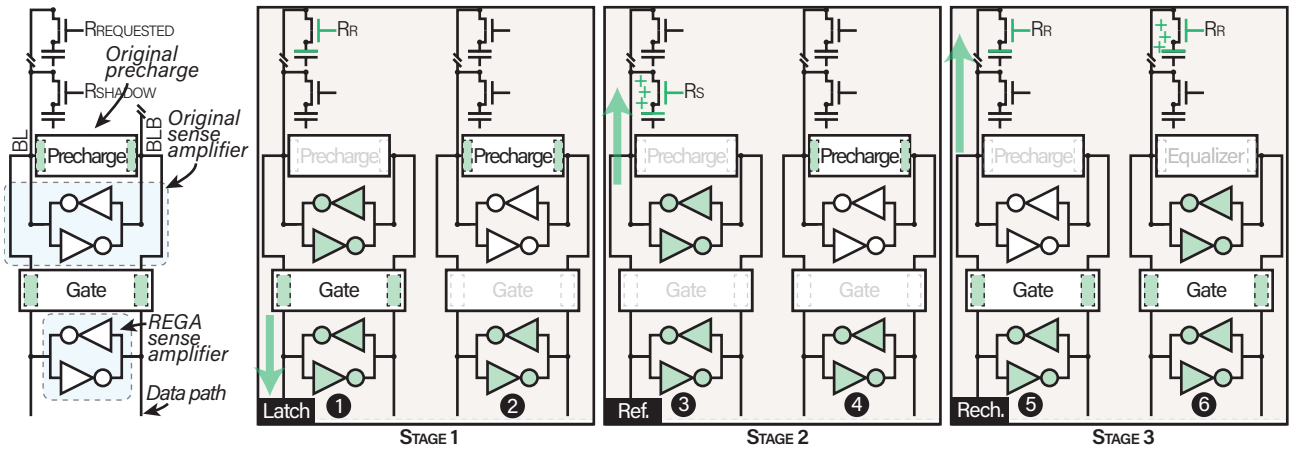


Fig. 7: **REGA's design.** The DRAM receives an ACT for $R_R$. After turning OFF the precharge and connecting $R_R$, $SA_R$ starts latching supported by $SA_O$ (❶). After $SA_R$ has started amplifying the voltage, the gate and $R_R$ are disconnected, the bitline is precharged, and $SA_O$ is turned OFF (❷). $SA_R$ remains active to support reads and writes. When the bitline has been precharged, $R_S$ is connected, and $SA_O$ starts refreshing it (❸). After $R_S$ has been refreshed and $R_S$ is closed, the bitline is precharged (❹). $R_R$ and $SA_R$ are connected to the bitline (❺). Lastly, $SA_O$ is turned ON to support the refresh operation on $R_R$ (❻).

**Stage 1: Logical ACT.** In Stage 1 (Figure 6), REGA latches the logical values from the requested row. The latching mechanism is first started by $SA_O$ and quickly latched by $SA_R$ (Figure 7, Stage 1.1). This combined operation allows $SA_R$ to be smaller than $SA_O$, thereby keeping area overhead and power consumption low. To correctly multiplex the bitline, a transmission gate is required between the two sense amplifiers, which prevents $SA_O$'s refresh mechanism from corrupting $SA_R$'s logical value when the signal $V_G$ is low. Therefore, before activating $R_S$, two events are necessary: turning OFF the original sense amplifier $SA_O$ and precharging the bitline (Figure 7, Stage 1.2).

**Stage 2: Parallel REF.** In Stage 2 (Figure 6), REGA performs a parallel refresh to $R_S$. First $R_S$ is activated, then the standard charge sharing and recharging, as described in §V, happen via $SA_O$ (Figure 7, Stage 2.1). Before Stage 3 can be started, $SA_O$ must be turned OFF, and the bitline must be precharged. This precharge is needed because the $SA_R$ might lose the latched value due to the high-charge parasitics of the bitline.

**Stage 3: Logical REF.** In the last stage, REGA recharges $R_R$ with the up-to-date logical value. This is necessary, as the cell had partially lost its value during the charge sharing operation of Stage 1. First, the transmission gate is turned ON, connecting $SA_O$ to the bitlines (Figure 7, Stage 3.1). Then, $SA_R$ is activated to assist in the refresh of $R_R$ (Figure 7, Stage 3.2).

### C. Detailed Signal Timings

We now analyze the detailed signal timings. Our design considers the worst-case scenario, where the $R_R$ and $R_S$ cells have opposite values and with a minimum charge. The timings are obtained using REM simulations as shown in Figure 9. In this accurate description, we use $SA_O$ and $SA_{OD}$ to refer to the activation of the original sense amplifier's supply, either with the common (1 V) or overdriven (1.4 V) voltage.

**Stage 1: Logical ACT.** In the initial state, the bitline is precharged (Figure 9, ⓪). The memory controller sends an ACT to the row $R_R$. This causes the precharge signal to be de-asserted and $R_R$ to be connected to the bitline while $R_R$'s SWL is set high (❶). With this last operation, charge sharing begins, inducing a voltage variation along the bitline. After the bitline has received the capacitor's charge, $SA_{OD}$ is activated ($SA_{OD}$ is set high) to help $SA_R$ latch the value (❷) with the transmission gate active. Shortly after, $SA_R$ is activated and starts latching the logical value of $R_R$. $SA_{OD}$ is turned OFF, and the transmission gate is opened ($V_G$ is set low, ❸).

At this point, the bitline is precharged for the required time (❹). $SA_R$ will latch the logic value before a tRCD. This operation is very fast because $SA_R$ does not have the load of the bitline, which is disconnected via the transmission gate. Before the end of tRAS, read and write operations go to $SA_R$.

**Stage 2: Parallel REF.** This stage follows the standard charge sharing and recharging but targets $R_S$ (❹-❻). After this activation (used as a refresh) of $R_S$ is over, $R_S$ is disconnected from the bitline, and $SA_O$ is turned OFF (❼). Then, the bitline is precharged. As previously noted, this precharge is needed because in the next stage, $SA_R$ will be connected to the bitline.

$SA_R$ now holds a value that must be stored back in $R_R$. However, if the parasitics of bitline held an opposite value, connecting the $SA_R$ might corrupt its value. This depends on $SA_R$'s transistors sizes, on its power supply, and on the bitline parasitics. Considering the technology of our collaborating DRAM vendor, the precharge operation is fast, and the $SA_R$'s transistors are big enough to reliably keep their value.

**Supporting other DRAM technologies.** We provide solutions for cases where the $SA_R$ or the precharge circuit must be very weak. First, we simulated a weaker precharge circuit and determined that it is not required to bring the bitline exactly to Vdd/2. Given our values for the $SA_R$, we found that a margin of at least 60 mV is tolerated. Second, we tested a weaker $SA_R$ with transistor widths halved. This situation can be overcome by controlling the gate circuity separately for the two bitlines (BL and BLB). In particular, in Stage 3 only the gate that connects BL can be connected. This improves the reliability of $SA_R$ to hold its value once connected to charged bitlines. If the assisted refresh is needed in Stage 3, $SA_O$ can initially be kept OFF while only BLB is precharged. Then, BLB's gate is connected, and $SA_O$ finishes the refresh operation.

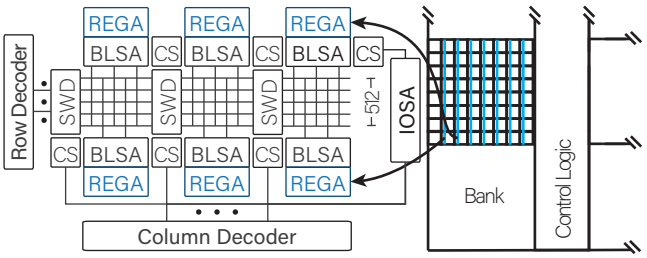


Fig. 8: **REGA's Internal Deployment.** REGA is deployed next to the original bitline sense amplifier (BLSA).

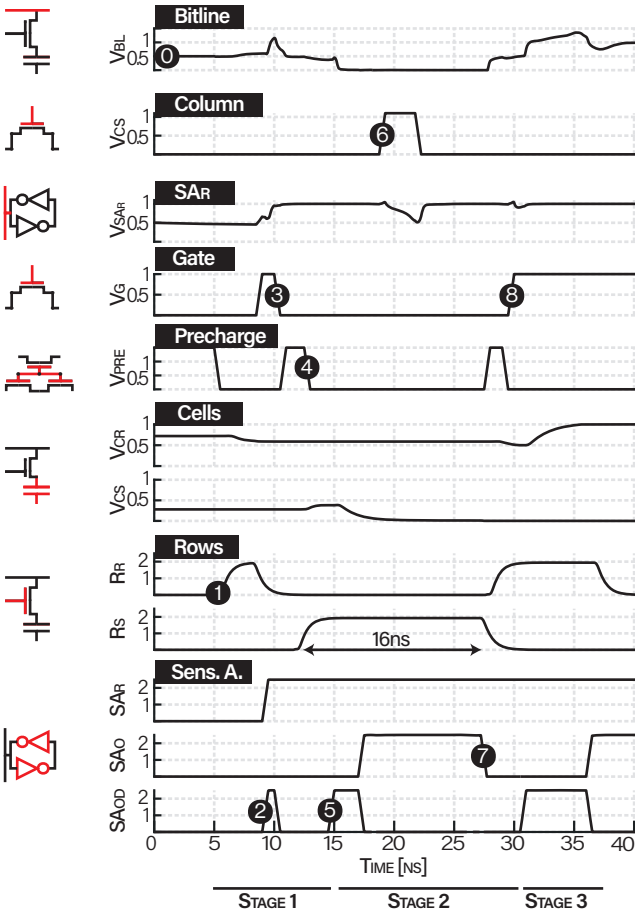


Fig. 9: **Timing of REGA.** Initially, the bitline is precharged (0). An ACT is received for $R_R$, and its word line is risen (1). After the charge sharing operation, $SA_{OD}$ helps $SA_R$ latching the cell's value (2). After $SA_R$ has started the latching operation, the gate is opened to disconnect $SA_R$ from the bitline (3). Now, the bitline is brought back to the reference voltage via the precharge circuit (4). $R_S$ is activated, and $SA_O$ and $SA_{OD}$ perform a normal refresh operation (5,7). Parallel to $R_S$ refresh, a read operation occurs (6) by reading from $SA_R$. Once $R_S$ has been refreshed, $SA_O$ is turned OFF (7). After a brief precharge operation, the gate is closed (8), and $R_R$ is refreshed by the combined operation of $SA_R$ and the original sense amplifier.

**Stage 3: Logical REF.** $R_R$ is activated, and the gate is turned ON (8). After a short time, $SA_{OD}$ is turned ON to assist the row refresh. To compensate for the short refresh time, $SA_{OD}$ is held overdriven longer than usual before switching to $SA_O$.

All the discussed timings need to consider the rising time and the delays due to parasitics. For the sake of simplicity, we did not include them in this description. They are, however, used in our simulations and included in our model. All voltage levels are summarized in Table VI.

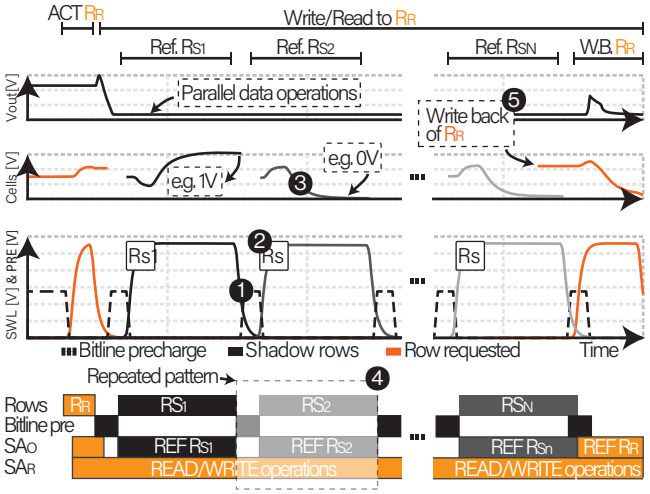


Fig. 10: **REGA Performing Multiple Refreshes.** The bitline is precharged (1), after which the target shadow row is activated (2), and $SA_O$ is used to perform its refresh (3). The same can be repeated for more shadow rows (4), and lastly, the process can be completed by restoring $R_R$ (5).

```
Output : Number of corruptions C for all rows
𝕊 ← PickRandomRows(1000)  // (≈ 2 MiB)
for t ← {8, 16, 24, 32} do // tRAS in ns
    for R ← 𝕊 do
        Fill R with a random data pattern
        Send ACT-PRE to R with tRAS = t
        Read R
        C_R^t ← number of corruptions in R
        if C_R^t == 0 then
            // If it does not corrupt with a low tRAS,
            // neither it will with a higher tRAS
            Remove R from 𝕊
return C
```

Alg. 1: **Experiment for measuring the slack in `tRAS`.**

### D. Parallel Refresh of Multiple Rows

As we discuss in § VIII, multiple parallel refreshes at every activation enable protecting devices with very small Rowhammer thresholds. REGA can perform more than one refresh in parallel to a row activation without changing its circuit. Parallel refreshes involve repeating some key operations, illustrated in Figure 10 for shadow rows $R_{S1}$ to $R_{SV}$. For each shadow row, REGA interleaves bitline precharging with Stage 2 (Parallel REF). First, the bitline is precharged (1). Then, the target shadow row is activated (2), and $SA_O$ is used to perform its refresh (3). Then, these operations can be repeated for a different shadow row (4), or the process can be completed by (re)storing the charge in the requested row (5).

In the next section, we show that on recent DDR4 and DDR5 devices, the minimum tRAS of 32 ns suffices for refreshing a single shadow row ($V = 1$). Performing multiple refreshes requires more time, which a DRAM device can do by extending tRAS from its minimum of 32 ns. To refresh $V$ shadow rows on top of $R_R$, REGA requires a tRAS of $32 + (V-1) \times (17.5)$ ns which we evaluate in §IX.

## VII. Impact of REGA on tRAS

For a single shadow refresh ($V = 1$), REGA needs 16 ns (Figure 9). We introduce two experiments showing that (i) tRAS

has sufficient slack to implement REGA in today's devices, and (ii) reducing tRAS does not negatively affect data retention. We present our experimental platforms in § VII-A, and the experimental methodologies and results in §§ VII-B and VII-C. We discuss how tRAS can be configured for values higher than 32 ns allowing multiple parallel refresh operations in § VII-D.

### A. Experimental Platforms

We measure the slack in tRAS on PCs to obtain minimal tRAS values. To measure the impact of reducing tRAS on data retention, we rely on an FPGA platform which provides us with precise timing for DRAM commands.

**PC configurations.** We use Intel Core i7-8700K (DDR4) and Intel Core i7-12700K (DDR5) machines for PC-based experiments. Our mainboards (Appendix D) allow accurately setting DRAM timings, such as tRAS. We use a SO-DIMM-to-UDIMM extender to connect our SO-DIMMs to these machines, which limits their speed to 2666 MHz. We use MemTest86 Pro (version 9.4) for testing the DIMMs under reduced tRAS.

**FPGA details.** Our platform is based on a Zynq ZCU104 FPGA running AntMicro's LiteX Rowhammer Tester [57]. It supports off-the-shelf DDR4 SO-DIMMs, and allows issuing DRAM commands directly to the memory device. Further, we have a DRAM heating infrastructure with which we can keep the DRAM device's temperature up to 100 °C.

**Test devices.** Our DRAM test pool consists of 21 DDR4 SO-DIMMs and 16 DDR5 UDIMMs, all off-the-shelf DRAM devices, covering all major DRAM manufacturers and varying in size and frequency. For more details about our test devices and the experimental platforms, we kindly refer to Appendix D.

### B. Experiment 1: Slack in tRAS

In this first experiment, we investigate the slack in the default tRAS of existing devices. We achieve this by sending ACT-PRE sequences with a reduced tRAS value and using corruptions as an indicator for failures caused by a too small tRAS value.

We describe our experiment in Algorithm 1. As temperature might affect the DRAM's operation, we perform this experiment under normal room temperature (25 °C) and the maximum temperature specified by the JEDEC standard (85 °C). Because precise timing is essential, we executed this experiment using our FPGA platform on all our SO-DIMMs. The results for the devices where we observed any corruptions are given in Table II. We observe that even with an extremely reduced tRAS of 8 ns, we could only observe corruptions on 3 of 21 tested DDR4 devices. $\mathbb{S}_{16}$ reports fewer corruptions with a higher temperature. This is because higher temperatures can lower the threshold of access transistors, allowing for faster data restoration in some cases (i.e., fewer corruptions). For a tRAS of 16 ns, the value we require for REGA with $V = 1$, we never observed corruptions.

We do not have precise control over DRAM commands on a PC. However, to verify that our results hold for UDIMMs, we reconfigured the tRAS of these devices in the PC's BIOS/UEFI to 16 ns. We then ran a full pass of all 16 MemTest86 [58] tests on all our test devices (including the SO-DIMMs) to check for any corruptions that might be caused by the lower tRAS. We confirm having observed no corruption over all tests in any of our DDR4 and DDR5 devices.

TABLE II: **Result of our free tRAS slack experiment.** We report the no. of corruptions on all SO-DIMMs and UDIMMs in our test pool. We omit devices without any observed corruptions.

| DIMM | Room temp. | | Max. (85 °C) | |
|---|---|---|---|---|
| /tRAS | 8 | 16 | 8 | 16 |
| $\mathbb{S}_6$ | 7 | 0 | 1,747 | 0 |
| $\mathbb{S}_{13}$ | 1,413 | 0 | 2,840 | 0 |
| $\mathbb{S}_{16}$ | 2,937 | 0 | 1,315 | 0 |

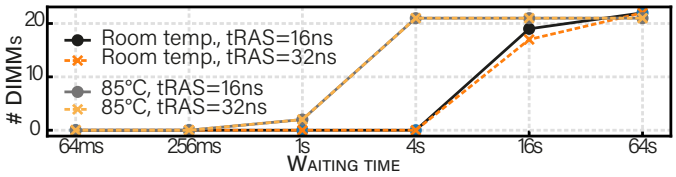


Fig. 11: **Retention time experiment.** Time required for corruptions to occur on the DDR4 SO-DIMMs under different tRAS settings and temperatures.

### C. Experiment 2: Impact of a Reduced tRAS on Data Retention

This experiment's goal is to show that reducing tRAS does not negatively affect the data retention time even though less time is given to the ACT command. In other words, reducing tRAS should not lead to data corruption due to retention errors. To analyze this, we construct an experiment where we refresh rows varying tRAS values and use wait periods of different lengths to see the impact of tRAS on data retention.

We precisely describe our experiment in Algorithm 2, which we repeat for the two different temperature levels for all DDR4 SO-DIMMs in our test pool. As the JEDEC standard requires a data retention time of 64 ms, we do not expect to see any retention failures for this waiting period. Figure 11 reports the number of DIMMs in which we observed corruptions, for different values of tRAS and after a waiting period. The data clearly shows that for up to 256 ms, there are no data corruptions: the first corruptions start to appear after a waiting time of 1 s. This is significantly more than the 64 ms that JEDEC specifies in the DDR4 standard. Furthermore, there is no clear difference between the retention profiles of devices with different tRAS values. Hence, we conclude that it is safe to reduce the tRAS value to 16 ns, or more precisely, use the available slack to perform REGA operations.

### D. Configuring tRAS

DDR$x$ devices use an SPD chip to inform the memory controller which timings to use [59]–[61]. The content of the SPD chip is standardized by JEDEC [62], [63] and intended to allow departing from the timings described in the DDR standard for future devices.

The SPD chip includes tRAS, therefore devices deploying REGA can set the necessary tRAS value. For DDR4 devices, the tRAS in the SPD can be set up to 512 ns, making REGA with high $V$ (e.g., 8) already deployable. On DDR5 devices, the tRAS on the SPD can be set up to 65.5 ns, making $V$ values higher than 2 not immediately compatible with the current SPD standard. As we discuss in § VIII, $V > 2$ enable protection for devices with $R_{thresh}$ smaller than 517, estimated to occur in around 7 years from now. We hence recommend future revisions to the JEDEC SPD standard to allow the SPD chip to set higher tRAS values as necessary. In § IX we evaluate the

```
Output : Number of corruptions C for all rows
S ← PickConsecutiveRows(1000) // (≈ 2 MiB)
for R ← S do
    Fill R with a random data pattern;
for t ← {8, 16} do // tRAS in ns
    for m ← {64k, 16k, 4k, 1k, 256, 64} do // waiting time in ms
        C^t_R ← 0;
        for r ← 0 to 5 by 1 do // reps. account for VRT
            for R ← S do
                Send PRE-ACT with tRAS = t ns // ≙ refresh
            Wait by sending NOPs for m ms;
            for R ← S do
                Read R;
                C^t_R ← C^t_R + number of corruptions in R;
        if C^t_R == 0 then
            // m causes no corruptions
            // ⟹ any m' < m will not cause corruptions
            break;
        C^t_R ← C^t_R/5 // Average over the 5 repetitions
return C
```

Alg. 2: **Evaluation of the impact of `tRAS` on the data retention time.**

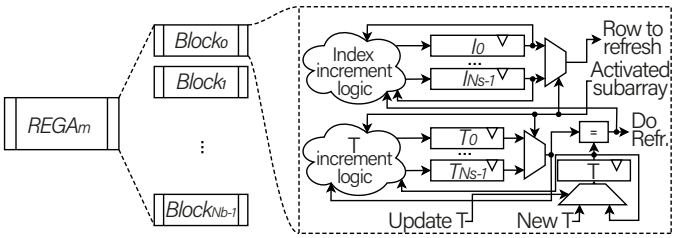


Fig. 12: **Overview of the REGA$_M$ implementation.**

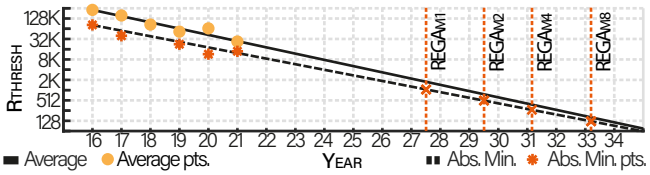


Fig. 13: **Trend of $R_{thresh}$ from 2016 to 2035.** In orange is reported the minimum $R_{thresh}$ for REGA$_{M1}$, REGA$_{M2}$, REGA$_{M4}$ and REGA$_{M8}$.

performance overhead introduced by increasing `tRAS` beyond 32 ns.

## VIII. REGA$_M$

We show how we can leverage refreshes generated by REGA to design new Rowhammer mitigations by demonstrating a blast-independent, fully in-DRAM, stand-alone mitigation called REGA$_M$. REGA$_M$ is simple, deterministic, and configurable based on the device's $R_{thresh}$.

### A. Design

Aggressor rows only affect victims inside the same sub-array since sub-arrays are physically separated from each other by sense amplifiers. Given a sub-array, REGA$_M$ cyclically refreshes all its rows as it receives activations. In particular, REGA$_M$ refreshes $V$ different rows in a sub-array every time it receives $T$ activations. $T$ and $V$ are parameters that can be dynamically configured through a freely available register inside SPD to account for devices with different Rowhammer thresholds and the discovery of new Rowhammer effects. We show in § IX that REGA$_M$ outperforms the state-of-the-art in-DRAM mitigation when it comes to known Rowhammer effects. Furthermore, since REGA$_M$ refreshes all rows in a sub-array that is receiving activations, it also provides strong protection against new (yet unknown) Rowhammer effects. As an example, after this paper was submitted, the latest JEDEC standard extended the maximum supported blast diameter to 6 [43]. REGA$_M$ is immediately capable of protecting devices in these scenarios. To easily integrate normal refresh operations, refreshes targeting a particular sub-array will use REGA$_M$'s row index as a target, and then increment it.

### B. ASIC implementation

We propose to implement REGA$_M$ in the CMOS area of the DRAM chip, similarly to our previous work [29]. Figure 12 provides an overview of REGA$_M$'s ASIC implementation. REGA$_M$ is made of $N_b$ identical blocks, where $N_b$ is the number of banks. Each block is essentially composed of $N_s$ indices $I_i$, where $N_s$ is the number of sub-arrays inside a bank and each $I_i$ is a 9-bit register pointing to the next row to be refreshed in the corresponding sub-array. Additionally, each block contains $N_s$ registers $T_i$, which are duty-cycling the refresh commands, i.e., to ensure that REGA$_M$ sends a refresh every $T$ activations received by a sub-array. Whenever an activation affects the $i$-th sub-array, if $T_i = T - 1$, then $V$ refreshes are sent to $I_i$, $T_i$ is reset and $I_i$ is incremented by $V$, wrapping around the maximum row index.

### C. Security

REGA$_M$ can protect a device for a minimum $R_{thresh}$ depending on its configuration. To obtain $R_{thresh}$, we consider the worst-case scenarios happening before and after a victim row is refreshed. For a typical sub-array size of 512 rows, a victim row is refreshed after a maximum of $\frac{512}{V} \times T$ activations, during which its aggressor rows can repeatedly be activated. Of these activations, $\frac{512}{V}$ will perform extra refreshes. Therefore, because the SWL is raised twice, these will hammer the victim twice. The last extra-refresh operation is an exception, as the victim will be refreshed after SWL is raised once. Instead, $\frac{512}{V} \times (T - 1)$ activations will not perform extra refreshes, hammering the victim only once. Lastly, refreshed rows will also hammer the victim if inside its blast diameter $B$. Depending on the victim's position inside the refreshed group, the refreshed rows will either be a hammering baseline for the next iteration or will hammer before the victim's refresh. In either case, this will result in a total of $B$ hammerings (i.e., row activations). A victim will always have a hammer baseline of 1 due to the SWL raise after its refresh. Therefore, a victim row can be hammered at most $\frac{512}{V} \times (T + 1) + B$ times. We further validated REGA$_M$ using state-of-the-art Rowhammer fuzzers [1], [28] without observing any bit flips.

In certain DRAM chips, rows that are kept active for longer periods could increase the Rowhammer vulnerability [64]. This behavior still requires extensive characterization to conclude whether it can provide an additional benefit to an attacker. If necessary, REGA$_M$ can completely eliminate this effect with a minor variation. Because REGA adds buffering sense amplifiers, $R_R$ does not need to stay active during writes and reads. REGA$_M$ write back operation can then be performed

during a PRE, as it only requires 12 ns.

Figure 13 reports $\text{REGA}_{\text{M}}$ thresholds for $V = 1, 2, 4, 8$, respectively as $\text{REGA}_{\text{M1-8}}$, offering Rowhammer protection for devices up to 10 years from now.

## IX. Evaluation

We now evaluate the key aspects of REGA and the mitigations $\text{REGA}_{\text{M1-8}}$. Results for REGA and the ASIC implementation are shared by all the mitigations. Results that are $V$-specific are indicated as $\text{REGA}_{\text{MX}}$ ($V = x$).

First, we analyze the die's area overhead due to the extra REGA circuitry and the ASIC implementation, and we report the static power consumption (§IX-A). Second, we evaluate the circuit's reliability based on 160 K analog Monte Carlo simulations (§IX-B). Third, we estimate the power, energy and performance overhead of $\text{REGA}_{\text{M1-8}}$ by running cycle-accurate simulations (§IX-C).

We compare the overheads of $\text{REGA}_{\text{M1-8}}$ with ProTRR [29], which is the state-of-the-art in-DRAM Rowhammer mitigation. We synthesized $\text{REGA}_{\text{M}}$'s ASIC and ProTRR using a 12 nm technology with Synopsys Design Compiler 2021. In our evaluations, we consider $B = 2$ for the classical Rowhammer effect [5], $B = 4$ for the recently introduced half-double effect [31], $B = 6$ that has been added in the latest JEDEC standard [43] and $B = 8$ for future DRAM technologies where an aggressor row can cause bit flips in four victim rows on each side. We regard $R_{thresh}$ of interest to be lower than 4 K.

### A. Area Overhead

The area in the DRAM chip is generally limited, and the vendors aim to maximize the area for the mats. Given that, we must ensure that REGA's implementation is practical, i.e., it consumes a reasonably small amount of die area.

**Methodology.** Depending on the DRAM design and technology, the ratio between the sensing circuit and the chip's die can vary between 8% and 15% [42], averaging 11.5%. The sensing circuit includes column selectors, precharge circuits, and BLSAs. According to the sense amplifier layout reported by our collaborating DRAM vendor, the BLSAs occupy 60% of the sensing area. REGA adds small buffering sense amplifiers and transmission gates, and for simplicity, we consider their length to be equivalent to the original BLSAs. More precisely, REGA requires eight additional transistors per each BLSA: four that are $1/6$-th of the BLSA's transistors width and another four that are $1/8$-th of the BLSA's transistors width.

To conservatively evaluate the area overhead of REGA, we assume no available free space to place our extra transistors. Instead, because each BLSA is formed by four transistors, we propose placing them in two groups following the BLSAs. Consequently, REGA's area overhead can be calculated as follows: $60\% \times (\frac{1}{6} + \frac{1}{8}) \times 11.5\%$ resulting in, on average, only 2% area overhead. Our ASIC implementation of $\text{REGA}_{\text{M}}$ incurs as little as 0.06% area overhead.

**Comparison with ProTRR.** In Figure 14, we show the total area overhead of $\text{REGA}_{\text{M}}$ compared to ProTRR for DDR4 and DDR5. As $\text{REGA}_{\text{M}}$ does not use any counters, its area overhead is independent of the Rowhammer threshold.

TABLE III: **Gem5 system configuration.**

| CPU | | Memory Controller | | DRAM | DDR4 | DDR5 |
|---|---|---|---|---|---|---|
| Sched. Type | OoO | #Channels | 2 | Freq (GHz) | 2.9 | 4.8 |
| #Cores | 8 | Page Policy | Open | Ranks | 1 | 1 |
| Freq. (GHz) | 3 | Scheduler | FR-FCFS | Bankgroups | 4 | 8 |
| L1 (KiB) | 2x32 | Queue Type | Per Bank | Banks/group | 4 | 2 |
| L2 (KiB) | 256 | Capacity (GiB) | 16 | Banks/rank | 16 | 16 |
| L3 (MiB) | 8 | | | Rows/bank | 64 K | 64 K |

In DDR4, ProTRR struggles for Rowhammer thresholds lower than 4 K ($B = 2$) or has no protection ($B > 2$). For DDR5 devices, $\text{REGA}_{\text{M}}$'s area is lower for Rowhammer thresholds lower than 1156 ($B = 2$) and any threshold lower than 4 K ($B > 2$). As thresholds approach 1 K, ProTRR can no longer protect these devices ($B = 4, 6, 8$).

**Static power overhead.** The extra sense amplifier circuit uses CMOS technology and incurs a negligible static power overhead. For 32 banks, $\text{REGA}_{\text{M}}$ has a static power consumption of 0.015%, for a baseline of 3 W [29]. This is significantly less than ProTRR, which requires overheads between 4.65% and 2.35% for 16 banks.

### B. Circuit Reliability

**Methodology.** We used LTspice [65] to simulate REGA using REM. We ran 40 K Monte Carlo simulations while introducing random variations of ±5% in the transistor's dimension and the line parasitics. As previously done for DRAM [40], [45], [66]–[69], we modeled the transistors using the 22 nm predictive transistor model (PTM) [70]. Based on indications from our DRAM collaborator, we set the DRAM cells' charge to the minimum that is still considered correct. We repeated the simulations for $V = 1, 2, 4, 8$.

**Results.** The circuit reported 100% reliability on all 160 K Monte Carlo simulations for $\text{REGA}_{\text{M1-8}}$. We consider the circuit reliable if all capacitors are recharged to the correct value. To evaluate the worst-case scenario, we tested the combinations where the sequence of cells to recharge have opposite values.

### C. Performance, energy and power overhead

We benchmark SPEC®2017 [71] with a cycle-accurate simulator, evaluating the energy and power overhead of the extra row refreshes, and the performance overhead due to the corresponding tRAS extension. Before presenting our results, we briefly describe our methodology and simulation setup. Because power presents the main design factor for devices, we refer the reader to Appendix E for the energy overhead results.

**Gem5 simulation.** We configured gem5 [72] as described in Table III to do a full-system simulation of Ubuntu (Linux kernel 5.4) based on an 8-core out-of-order CPU. Like previous work [29], we modeled the DRAM subsystem using DRAMsim3 [73] as it allows for a higher accuracy than the DRAM model included in gem5. Per SPEC®2017's recommendation, we ran multiple copies in parallel, equal to the number of cores (i.e., eight in our configuration). This maximizes the workload and, as such, increases the load on the memory subsystem. For each benchmark, we obtained 20 equally-spaced checkpoints (SMARTS methodology [74], [75]) and we ran each checkpoint for 10 M instructions.

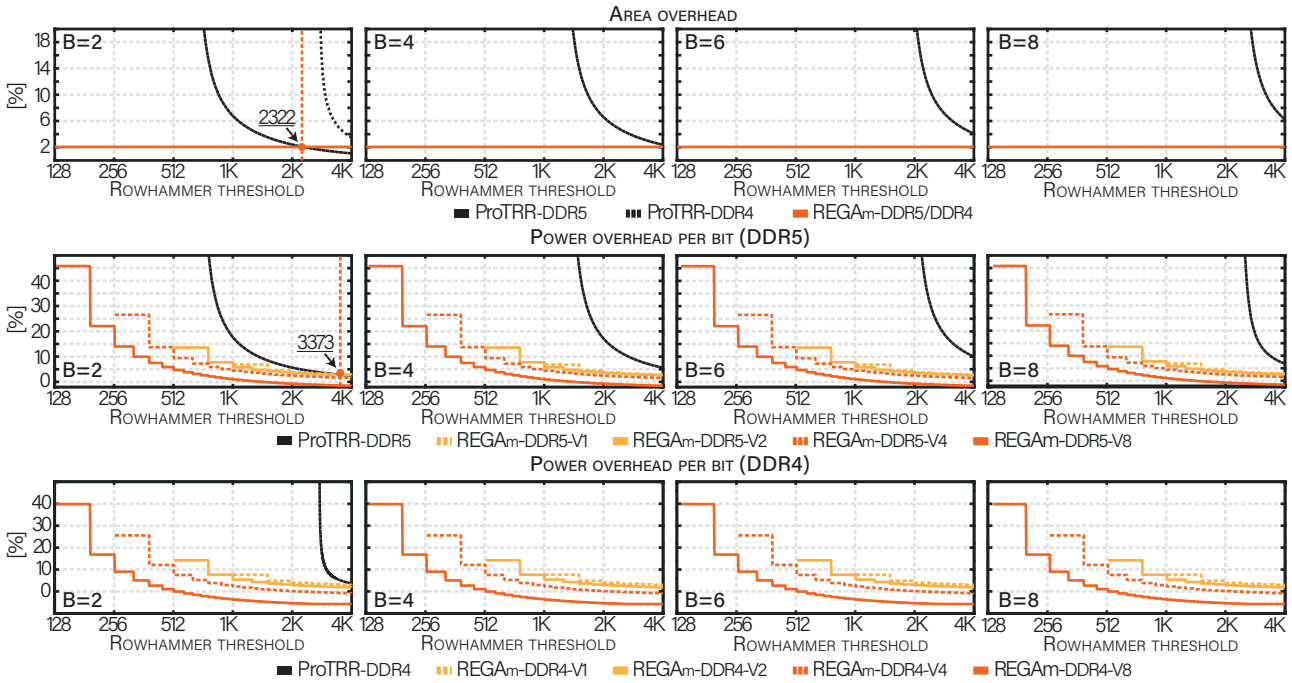


Fig. 14: **Area and power overhead of REGA$_{M1-8}$. (i.) Area overhead.** Overhead for 32 banks and blast diameters of 2, 4, 6, and 8. **(ii.) Power overhead per bit (DDR5/4).** Power overhead related to a bit for a fixed chip area. In orange, the point of crossing between REGA$_M$ and ProTRR is highlighted. Because REGA$_M$ is blast independent, its evaluation for different $Bs$ is identical. The only difference is a $R_{thresh}$ shift of 2 between each panel (see §VIII-C).

**Methodology for energy and power overhead.** We evaluated the energy overhead on top of a regular ACT operation for $V = 1, 2, 4, 8$ extra refreshes. We considered the worst case where a cell needs to be *fully* recharged. Simulating REGA using LTspice [65], averaging over 50 Monte Carlo simulations, showed a per-ACT energy overhead of 38%, 95%, 223% and 479%, respectively for $V = 1, 2, 4, 8$. For each benchmark, we extracted the average total energy consumed and the energy consumed due to activations, repeated for DDR4 and DDR5. Depending on $T$ and $V$, we obtained the overhead with respect to the baseline ($V = 0$). We used this energy to derive power consumption and power overhead, by using the individual CPU time simulated for each benchmark and each checkpoint. We then calculated the energy and power overhead per bit by considering a fixed die size available, as we do not expect manufacturers to increase the die area freely.

**Power overhead.** REGA$_M$ performs refreshes every $T$ activations, which depends on the Rowhammer threshold and on $V$. A high value of $T$ substantially decreases power consumption. For example, if a device has a Rowhammer threshold of 4 K, REGA$_{M1}$ can be activated every 6 activations, incurring 6 times less energy overhead and consequently less power overhead. Figure 14 shows the average power overhead per bit depending on the Rowhammer threshold, compared with ProTRR. In almost all cases of interest, REGA$_M$ has a lower power consumption compared to ProTRR. As an example, to protect DDR5 devices with $R_{thresh}$ of 1027 and $B = 2$, REGA$_M$ requires only 7% extra power per bit, while ProTRR needs 18%. In the case of extended tRAS and high values of $T$, the power is reduced with respect to the baseline (negative overhead). This is due to a reduced amount of activations sent for a given time, and is reflected in a performance overhead (see next). We further provide REGA$_M$'s absolute power overhead (i.e., not relative to the area) in Appendix G.

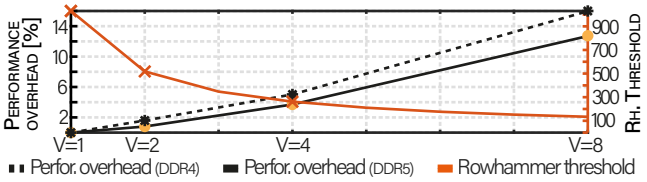


Fig. 15: **Performance overhead.** Average performance overhead for V=1, 2, 4, 8 for SPEC®2017 on DDR4 and DDR5.

**Performance overhead.** REGA does not require timing changes when refreshing a single row ($V = 1$). Therefore, it does not introduce any performance overhead. If REGA refreshes $V > 1$ rows, tRAS must be extended.

We repeat the measurements for $V = 2, 4, 8$ and for the baseline ($V = 0$). In Figure 15 we report the average performance overhead relative to the baseline. A detailed overview can be found in Appendix F. ProTRR has a negligible performance overhead, however, for low thresholds it becomes infeasible due to its area overhead or it cannot provide a sufficient protection. REGA$_M$ offers a protection with 0% performance overhead for the thresholds that are covered by ProTRR, and it extends this protection to thresholds significantly lower by adding a performance overhead due to extending tRAS (e.g., 3.7% for $R_{thresh} = 261$).

## X. Related Work

In the following, we summarize and compare existing Rowhammer mitigations to REGA$_M$. In Table IV, we provide an overview of the existing mitigations, comparing them for their

TABLE IV: **Overview of Rowhammer mitigations.**

| – DRAM | Yr. | Security | | | Comp. | Eval. | Concepts | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Eff. | Vuln. | Det. | DDR 4/5 | DDR5 | Ct. | Pr. | Is. | B.I. |
| **REGA$_{M1-M2}$** | '22 | ● | ○ | ● | ●/● | ● | ○ | ○ | ○ | ● |
| **REGA$_{M4-M8}$** | '22 | ● | ○ | ● | ●/○ | ● | ○ | ○ | ○ | ● |
| Mithril [39] | '22 | ○ | ● | ● | ○/○ | ● | ● | ○ | ○ | ○ |
| ProTRR [29] | '22 | ○ | ○ | ● | ●/● | ● | ● | ○ | ○ | ○ |
| Panopticon [41] | '21 | ○ | ○ | ● | ●/● | ○ | ● | ○ | ○ | ○ |
| ProHIT [40] | '17 | ○ | ● | ○ | ●/● | ○ | ○ | ● | ○ | ○ |
| **– Memory controller** | | | | | | | | | | |
| Hydra [27] | '22 | ○ | ○ | ● | ○/○ | ○ | ● | ○ | ○ | ○ |
| Row-Swap [38] | '22 | ○ | ○ | ○ | ●/● | ○ | ● | ● | ● | ○ |
| BlockH. [21] | '21 | ○ | ○ | ● | ●/● | ○ | ● | ○ | ○ | ○ |
| CAT-TWO [24] | '20 | ○ | ● | ● | ○/○ | ○ | ● | ○ | ○ | ○ |
| Graphene [22] | '20 | ○ | ● | ● | ○/○ | ○ | ● | ○ | ○ | ○ |
| TWiCe [26] | '19 | ○ | ● | ● | ○/○ | ○ | ● | ○ | ○ | ○ |
| MRLoc [23] | '19 | ○ | ● | ○ | ○/○ | ○ | ○ | ● | ○ | ○ |
| CBT [25] | '16 | ○ | ● | ● | ○/○ | ○ | ● | ○ | ○ | ○ |
| PARA [5] | '14 | ○ | ○ | ○ | ●/● | ○ | ○ | ● | ○ | ○ |
| **– Software** | | | | | | | | | | |
| ALIS [19] | '18 | ○ | ○ | ● | –/– | ○ | ○ | ○ | ● | ○ |
| GuardION [12] | '18 | ○ | ● | ● | –/– | ○ | ○ | ○ | ● | ○ |
| ZebRAM [37] | '18 | ○ | ● | ● | –/– | ○ | ○ | ○ | ● | ○ |
| CATT [36] | '17 | ○ | ● | ● | –/– | ○ | ○ | ○ | ● | ○ |
| ANVIL [35] | '16 | ○ | ● | ● | –/– | ○ | ● | ○ | ○ | ○ |

security properties (**Security**), their deployment location, their compatibility to the DDR$x$ standard (**Comp.**), their approach (**Concepts**), and if they were evaluated on the latest DDR standard (**Eval.**). We differentiate between agreement (●), and disagreement (○); for positive (●) and negative properties (●). Not applicable properties are denoted by "–".

**Security.** First, 15 of our 19 considered mitigations are deterministic (**Det.**), which is favorable due to stronger security guarantees. Other mitigations use probabilistic decisions such as MRLoc, ProHIT, and PARA. Second, REGA$_M$ is the first proposed mitigation that can protect against new (unknown) Rowhammer effects without modification (**Eff.**) such as the recently discovered half-double effect or even higher blast diameters [2], [43]. Third, some mitigations suffer from known vulnerabilities (**Vuln.**) as shown by existing work [22], [29] and in Appendix H. We only considered insecurities in the original design, not arising from new effects. As previously exposed [76], in-CPU mitigations that rely on refreshes are either vulnerable or incompatible. For a refresh to be secure, the internal DRAM topology must be known. This way, the mitigation can keep track of the rows that are hammered by extra refreshes. Unfortunately this has never been addressed by the standard, leading to new attacks to surface [2]. We considered a mitigation to be vulnerable if they do not mention this effect, or do not require the topology to be known.

**Location.** The majority of existing mitigations (9 out of 19) need changes in the CPU's memory controller. Software-based mitigations involving the operating system have also been proposed. Notably, the focus has moved since 2019 from software- towards memory controller-based mitigations and, more recently (2021+), to fully in-DRAM mitigations.

**Compatibility.** Most mitigations target the **DDR4** standard, though many require changes to the protocol, for example, by requiring a new (refresh) command or the internal row layout. We consider mitigations to be non-compliant, if in the original publication they required standard modifications for the evaluated protocol. We report Row-Swap to be compatible with the standard, however, as confirmed with the authors, the time delay for the operations should be roughly twice what is used in the paper. Only the more recent mitigations from 2022, namely REGA, ProTRR, and Mithril, are evaluated for **DDR5**. We ignore software-based mitigations, as they are by design independent of the DDR$x$ standard. We considered mitigations that rely on the knowledge of internal row mapping to be incompatible but secure. Currently, REGA$_{M4-M8}$ are non-compliant with the DDR5's SPD specification, which limits the compatible volume to $V = 1, 2$.

**Concepts.** Most mitigations employ counters (**Cnt.**) to keep track of aggressors or victims. Those who do not employ counters use other data structures such as queues (e.g., ProHIT and MRLoc) or are entirely stateless (e.g., PARA). As it is generally difficult to precisely track row activations from software, 4 out of 5 proposed mitigations use isolation (**Isol.**) to protect against Rowhammer. ANVIL is an exception to this trend: it uses performance counters to track row activations. We consider PARA, ProHIT, MRLoc, and also Row-Swap's random swapping of rows to be based on probabilities (**Prob.**).

REGA$_M$ is the only blast-diameter independent (**B.I.**) mitigation. The design of all other mitigations heavily relies on the considered diameter. This gives REGA$_M$ the flexibility to scale with the increasing blast diameter.

## XI. Conclusion

We presented REGA, a new in-DRAM mechanism that can scale the number of refreshes with activations. REGA uses buffering sense amplifiers to time-multiplex DRAM bitlines so that refreshes can occur in parallel to standard DRAM operations. We demonstrated the correctness of REGA's circuit using a new accurate DRAM model that we developed in collaboration with a DRAM vendor. REGA enables simple yet powerful mitigations against current and future Rowhammer attacks. We built a deterministic and scalable mitigation on top of REGA, called REGA$_M$, and evaluated its area, power and performance impact. REGA is the first in-DRAM mitigation that scales to small Rowhammer thresholds with a constant small area overhead (2.1%) and power and performance overhead dependent on the Rowhammer threshold. As an example, REGA scales to $R_{thresh}$ = 517 with 11.5% power and 0.8% performance overhead.


## Acknowledgments

We thank our anonymous reviewers for their valuable feedback. This work was supported by the Swiss National Science Foundation under NCCR Automation, grant agreement 51NF40_180545, the Swiss State Secretariat for Education, Research and Innovation under contract number MB22.00057 (ERC-StG PROMISE), and a Microsoft Swiss JRC grant.

## REFERENCES


[1] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "Blacksmith: Scalable Rowhammering in the Frequency Domain," in *IEEE S&P*, May 2022.

[2] A. Kogler, J. Juffinger, S. Qazi, Y. Kim, M. Lipp, N. Boichat, E. Shiu, M. Nissler, and D. Gruss, "Half-double: Hammering from the next row over," in *USENIX Security*, 2022.

[3] JEDEC Solid State Technology Association, "JESD79-4B, DDR4 Specification," 2017.

[4] ——, "JESD79-5, DDR5 Specification," 2020.

[5] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *ACM/IEEE ISCA*, 2014.

[6] M. Seaborn and T. Dullien, "Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges," in *Black Hat USA*, 2015.

[7] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another Flip in the Wall of Rowhammer Defenses," in *IEEE S&P*, 2018.

[8] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU," in *IEEE S&P*, 2018.

[9] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector," in *IEEE S&P*, 2016.

[10] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.Js: A Remote Software-Induced Fault Attack in JavaScript," in *DIMVA*, 2016.

[11] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: Synchronized Many-Sided Rowhammer Attacks from JavaScript," in *USENIX Security*, 2021.

[12] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *ACM SIGSAC*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1675–1689.

[13] V. van der Veen, M. Lindorfer, Y. Fratantonio, H. P. Pillai, G. Vigna, C. Kruegel, H. Bos, and K. Razavi, "Guardion: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM," in *DIMVA*, 2018.

[14] Z. Zhang, Z. Zhan, D. Balasubramanian, X. Koutsoukos, and G. Karsai, "Triggering Rowhammer Hardware Faults on ARM: A Revisit," in *ACM ASHES*, 2018.

[15] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading Bits in Memory Without Accessing Them," in *IEEE S&P*, 2020.

[16] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip Feng Shui: Hammering a Needle in the Software Stack," in *USENIX Security*, 2016.

[17] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation." in *USENIX Security*, 2016.

[18] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks," in *IEEE S&P*, 2019.

[19] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer Attacks over the Network and Defenses," in *USENIX ATC*, 2018.

[20] M. Lipp, M. Schwarz, L. Raab, L. Lamster, M. T. Aga, C. Maurice, and D. Gruss, "Nethammer: Inducing Rowhammer Faults Through Network Requests," in *EuroS&PW*, 2020, pp. 710–719.

[21] A. G. Yağlikçi, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi, S. Ghose, and O. Mutlu, "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows," in *IEEE HPCA*, 2021, pp. 345–358.

[22] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. H. Ahn, and J. W. Lee, "Graphene: Strong yet Lightweight Row Hammer Protection," in *IEEE/ACM MICRO*. IEEE, 2020, pp. 1–13.

[23] J. M. You and J.-S. Yang, "MRLoc: Mitigating Row-Hammering based on Memory Locality," in *ACM/IEEE DAC*. IEEE, 2019, pp. 1–6.

[24] I. Kang, E. Lee, and J. H. Ahn, "CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention," *IEEE Access*, vol. 8, pp. 17 366–17 377, 2020.

[25] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Counter-Based Tree Structure for Row Hammering Mitigation in DRAM," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 18–21, 2016.

[26] E. Lee, I. Kang, S. Lee, G. Edward Suh, and J. Ho Ahn, "TWiCe: Preventing Row-Hammering by Exploiting Time Window Counters," in *ACM/IEEE ISCA*, 2019.

[27] M. Qureshi, A. Rohan, G. Saileshwar, and P. J. Nair, "Hydra: Enabling low-overhead mitigation of row-hammer at ultra-low thresholds via hybrid tracking," in *ISCA*. New York New York: ACM, Jun. 2022, pp. 699–710.

[28] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *IEEE S&P*, 2020.

[29] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "PROTRR: Principled yet optimal in-DRAM target row refresh," in *IEEE S&P*, 2022.

[30] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," in *ACM/IEEE ISCA*, 2020, pp. 638–651.

[31] G. LLC, "Half-Double: Next-Row-Over Assisted Rowhammer," Tech. Rep., May 2021.

[32] M. Redeker, B. F. Cockburn, and D. G. Elliott, "An investigation into crosstalk noise in DRAM structures," in *IEEE MTDT*. IEEE, 2002, pp. 123–129.

[33] Y. Tobah, A. Kwong, I. Kang, D. Genkin, and K. G. Shin, "SpecHammer: Combining spectre and rowhammer for new speculative attacks," in *IEEE S&P*, 2022.

[34] H. Hassan, Y. C. Tugrul, J. S. Kim, V. Van der Veen, K. Razavi, and O. Mutlu, "Uncovering in-DRAM RowHammer protection mechanisms: A new methodology, custom RowHammer patterns, and implications," in *IEEE/ACM MICRO*, 2021, pp. 1198–1213.

[35] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks," in *ACM ASPLOS*, 2016.

[36] F. Brasser, L. Davi, D. Gens, C. Liebchen, and A.-R. Sadeghi, "CAn't Touch This: Software-Only Mitigation against Rowhammer Attacks targeting Kernel Memory," in *USENIX Security*, 2017.

[37] R. K. Konoth, M. Oliverio, A. Tatar, D. Andriesse, H. Bos, C. Giuffrida, and K. Razavi, "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks," in *USENIX OSDI*, 2018.

[38] G. Saileshwar, B. Wang, M. Qureshi, and P. J. Nair, "Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation between Aggressor and Victim Rows," p. 14, 2022.

[39] M. J. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. J. Ham, J. W. Lee, and J. H. Ahn, "Mithril: Cooperative row hammer protection on commodity DRAM leveraging managed refresh," in *IEEE HPCA*. IEEE, 2022, pp. 1156–1169.

[40] M. Son, H. Park, J. Ahn, and S. Yoo, "Making DRAM Stronger Against Row Hammering," in *ACM/IEEE DAC*, 2017, pp. 1–6.

[41] T. Bennett, S. Saroiu, A. Wolman, and L. Cojocar, "Panopticon: A Complete In-DRAM Rowhammer Mitigation," in *DRAMSec*, 2020.

[42] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *IEEE/ACM MICRO*. IEEE, 2010, pp. 363–374.

[43] JEDEC Solid State Technology Association, "JESD795B," 2022.

[44] ——, "JEP300-1, NEAR-TERM DRAM LEVEL ROWHAMMER MITIGATION," 2021.

[45] H. Luo, T. Shahroodi, H. Hassan, M. Patel, A. G. Yağlıkçı, L. Orosa, J. Park, and O. Mutlu, "CLR-DRAM: A low-cost DRAM architecture enabling dynamic capacity-latency trade-off," in *ACM/IEEE ISCA*. IEEE, 2020, pp. 666–679.

[46] K. Kraft, C. Sudarshan, D. M. Mathew, C. Weis, N. Wehn, and M. Jung, "Improving the error behavior of DRAM by exploiting its Z-channel property," in *DATE*. IEEE, 2018, pp. 1492–1495.

[47] M. Jung, C. C. Rheinländer, C. Weis, and N. Wehn, "Reverse engineering of DRAMs: Row hammer with crosshair," in *ACM MEMSYS*, 2016, pp. 471–476.

[48] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM Circuit Design: Fundamental and High-Speed Topics*. John Wiley & Sons, 2007, vol. 13.

[49] A. Kotabe, "Low-power DRAM," in *Green Computing with Emerging Memory*, T. Kawahara and H. Mizuno, Eds. New York, NY: Springer New York, 2013, pp. 87–110.

[50] K. Koo, S. Ok, Y. Kang, S. Kim, C. Song, H. Lee, H. Kim, Y. Kim, J. Lee, S. Oak, Y. Lee, J. Lee, J. Lee, H. Lee, J. Jang, J. Jung, B. Choi, Y. Kim, Y. Hur, Y. Kim, B. Chung, and Y. Kim, "A 1.2V 38nm 2.4Gb/s/Pin 2Gb

DDR4 SDRAM with bank group and ×4 half-page architecture," in *IEEE ISSCC*, Feb. 2012, pp. 40–41.
[51] M. Nakamura, T. Takahashi, T. Akiba, G. Kitsukawa, M. Morino, T. Sekiguchi, I. Asano, K. Komatsuzaki, Y. Tadaki, S. Cho *et al.*, "A 29-Ns 64-Mb DRAM with hierarchical array architecture," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1302–1307, 1996.
[52] T.-Y. Oh, H. Chung, J.-Y. Park, K.-W. Lee, S. Oh, S.-Y. Doo, H.-J. Kim, C. Lee, H.-R. Kim, J.-H. Lee *et al.*, "A 3.2 Gbps/Pin 8 gbit 1.0 V LPDDR4 SDRAM with integrated ECC engine for sub-1 V DRAM core operation," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 178–190, 2014.
[53] TechInsights Inc., "SK hynix 21 nm DRAM Cell Technology: Comparison of 1st and 2nd generation," Jun. 2017.
[54] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch *et al.*, "RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization," in *IEEE/ACM MICRO*, 2013, pp. 185–197.
[55] T. Takahashi, T. Sekiguchi, R. Takemura, S. Narui, H. Fujisawa, S. Miyatake, M. Morino, K. Arai, S. Yamada, S. Shukuri *et al.*, "A multigigabit DRAM technology with 6F/sup 2/open-bitline cell, distributed overdriven sensing, and stacked-flash fuse," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1721–1727, 2001.
[56] J.-T. Lin and C.-C. Hsu, "An initial overdriven sense amplifier for low voltage DRAMS," in *IEEE ICCDCS*. IEEE, 2000, pp. C31–1.
[57] AntMicro, "LiteX Rowhammer Tester," 2022.
[58] "MemTest86 - Official Site of the X86 Memory Testing Tool."
[59] JEDEC, "DDR4 SDRAM UDIMM Design Specification," 2019.
[60] ——, "DDR4 SDRAM SODIMM Design Specification," 2019.
[61] ——, "DDR3 SDRAM Unbuffered DIMM Design Specification," 2021.
[62] ——, "SPD Annex L: Serial Presence Detect (SPD) for DDR4 SDRAM Modules, Release 6," 2022.
[63] ——, "JESD4005A," 2022.
[64] L. Orosa, A. G. Yaglikci, H. Luo, A. Olgun, J. Park, H. Hassan, M. Patel, J. S. Kim, and O. Mutlu, "A deeper look into RowHammer's sensitivities: Experimental analysis of real DRAM chips and implications on future attacks and defenses," in *IEEE/ACM MICRO*, 2021, pp. 1182–1197.
[65] Analog Devices Inc., "LTspice Simulator," 2022.
[66] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "ChargeCache: Reducing DRAM latency by exploiting row access locality," in *IEEE HPCA*. IEEE, 2016, pp. 581–593.
[67] C. Lin, W. He, Y. Sun, Z. Mao, and M. Seok, "CDAR-DRAM: An in-Situ charge detection and adaptive data restoration DRAM architecture for performance and energy efficiency improvement," in *ACM/IEEE DAC*. IEEE, 2021, pp. 1093–1098.
[68] Y. Wang, L. Orosa, X. Peng, Y. Guo, S. Ghose, M. Patel, J. S. Kim, J. G. Luna, M. Sadrosadati, N. M. Ghiasi *et al.*, "Figaro: Improving system performance via fine-grained in-Dram data relocation and caching," in *IEEE/ACM MICRO*. IEEE, 2020, pp. 313–328.
[69] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *IEEE/ACM MICRO*. IEEE, 2017, pp. 273–287.
[70] N. Integration and A. Modeling (NIMO) Group, "Predictive Technology Model (PTM)," 2022.
[71] J. Bucek, K.-D. Lange, and J. v. Kistowski, "SPEC CPU2017: Next-Generation Compute Benchmark," in *ICPE*, 2018, pp. 41–42.
[72] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH*, vol. 39, no. 2, pp. 1–7, Aug. 2011.
[73] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *IEEE Computer Architecture Letters*, 2020.
[74] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," in *ACM/IEEE ISCA*, 2003, pp. 84–97.
[75] K. Loughlin, I. Neal, J. Ma, E. Tsai, O. Weisse, S. Narayanasamy, and B. Kasikci, "{DOLMA}: Securing speculation with the principle of transient {non-Observability}," in *USENIX Security*, 2021, pp. 1397–1414.
[76] S. Saroiu and A. Wolman, "How to Configure Row-Sampling-Based Rowhammer Defenses," in *DRAMSec*, 2022.

## APPENDIX A
## ROWHAMMER TREND

The fitting of the curves is based on the minimum Rowhammer thresholds reported in previous works [34], [42]. We now briefly report the methodology and fitting results.

**Average curve.** For each DRAM vendor, we calculated the average $R_{thresh}$ in each year. Then, we averaged across vendors for the same year. The resulting points are reported in Figure 2, which also includes a fitted curve using an exponential function (i.e., $a \times e^{b \times x}$) obtained using MATLAB® 2020 automatic fitting tool (R-square=0.96).

**Absolute minimum curve.** For each year, we considered the absolute minimum across all vendors. In the dataset, the year 2018 included only a single point, which we considered to be an overly optimistic outlier. For this reason, we removed it from the computation, as it would have skewed the minimum curve to be overly optimistic. Like above, the points used for the fitting are reported in Figure 2, and fitted with an exponential function (i.e., $a \times e^{b \times x}$) obtained using MATLAB® 2020 automatic fitting tool (R-square=0.98).

The figure reports the minimum thresholds supported by the mitigations. However, the Rowhammer threshold is defined differently depending on the publication. In this work and others [22], [29], [39], $R_{thresh}$ is the minimum number of activations to have bit flips. A $R_{thresh}$ of 1024 with $B = 4$ could be reached by 4 different aggressors, each activated $1024/4 = 256$ times. Other mitigations [27], [38] refer to $R_{thresh}$ relatively to aggressors. In those cases, the threshold is the number of times *every* aggressor in the blast diameter needs to be activated to induce bit flips. For example, Hydra [27] targets a threshold of 500 with $B = 4$. Because each row can be activated 500 times, and each victim has 4 different aggressors, this corresponds to a $R_{thresh}$ equal to 2000.

## APPENDIX B
## ABBREVIATIONS

In Table V, we summarize the abbreviations and symbols we used throughout this work.

## APPENDIX C
## VOLTAGES USED IN REM

The voltage levels of our REM are listed in Table VI.

## APPENDIX D
## EXPERIMENT PLATFORM & DEVICES

In the following, we provide more details on our PC-based test platform and the test devices of our DRAM test pool.

**PC.** We use Intel Core i7-8700K ("Coffee Lake") machines for DDR4 experiments, equipped with ASUS ROG STRIX Z930-E mainboards. We use Intel Core i7-12700K ("Alder Lake") machines for DDR5 experiments, equipped with Gigabyte Z690 AORUS PRO mainboards.

**DDR4/5 Test Devices.** In Table VII, we list all the DDR4 and DDR5 DIMMs in our DRAM test pool.

TABLE V: **Abbreviations & Symbols.** A summary of abbreviations and symbols we used in our work with a brief description and reference to the section where it has been introduced.

| Abbrv./Symb. | Description | Ref. (§) |
|---|---|---|
| BLSA | Bitline Sense Amplifier | V-A |
| DIMM | Dual-Inline Memory Module | II-A |
| ECC | Error-Correcting Codes | II-B |
| GIO | Global I/O | V-A |
| LIO | Local I/O | V-A |
| MC | Memory Controller | X |
| MC | Monte Carlo | IX |
| REM | REGA (DRAM) Model | V-C |
| REGA | Refresh-Generating Activations | IV |
| RFM | Refresh Management (DDR5 Extension) | IV |
| SPD | Serial Presence Detect | IV |
| SWD | Sub-Word Line Driver | V-A |
| SWL | Sub-Word Line | V-A |
| TRR | Target Row Refresh | II-B |
| B | Blast Diameter | II-C |
| T | Period of $\text{REGA}_\text{M}$ parallel refreshes | VIII |
| V | No. of shadows rows refreshed in parallel | VI-B |
| ACT | A DRAM activation command | II-A |
| PRE | A DRAM precharge command | II-A |
| tRAS | Min. Row Address Strobe: ACT-to-PRE delay | II-A |
| $R_{thresh}$ | #ACTs req. to trigger bit flips | II-C |

TABLE VI: **Voltage levels used by REM.**

| Parameter | Voltage (V) | Description |
|---|---|---|
| Vpp | 2.5 | SWD power supply and overdrive control |
| Vss | 0.0 | Ground |
| Vperi | 1.1 | Peripherical circuitry voltage |
| Vdd | 1.0 | DRAM core voltage and cell's high value |
| Vpre | 1.5 | Control voltage of the precharge circuit |
| Vod | 1.4 | Overdrive voltage |

## Appendix E
## Energy Overhead

**Results energy overhead.** Figure 16 shows the average energy overhead per bit depending on the Rowhammer threshold, compared with ProTRR. Given the recent half-double effect, $\text{REGA}_\text{M}$ is always convenient for DDR4 and DDR5 for the threshold in scope. For $B = 2$, $\text{REGA}_\text{M}$ is comparable to ProTRR (DDR5) and convenient in current DDR4 technologies.

## Appendix F
## Performance Overhead

Figure 18 and Figure 17 show the performance overhead for $V = 2, 4, 8$, for the individual benchmarks of SPEC®2017 on DDR4 and DDR5. As discussed in the paper, $V = 1$ does not incur any performance overhead.

## Appendix G
## Power Overhead

Figure 20 and Figure 19 show the average power overhead without considering the area overhead. The value is relative to the baseline power consumption, which depends on the device. Average power consumptions around 3 W are common [29].

## Appendix H
## Mithril

In this appendix, we point out vulnerabilities we discovered in the state-of-the-art in-DRAM mitigation Mithril [39]. Our analysis is focused on its security and standard compliance.

TABLE VII: **Specifications of the DRAM devices in our test pool.** Upper half ($\mathbb{S}_x$): DDR4 SO-DIMMs, lower half ($\mathbb{U}_x$): DDR5 UDIMMs. We report for each device its DRAM manufacturer (**DRAM Manuf.**) or "*n/a*" if there is no information reported by the DIMM's SPD chip; its manufacturing date (**Mf. Date**), or the date of purchase (†) in case it is not reported by the SPD chip; the frequency (**Freq.**); the device's size (**Size**); the geometry (**Geom.**) as number of ranks/banks; and its default **tRAS** value. DDR5: Same devices are same-kit modules.

| DIMM | DRAM Manuf. | Mf. Date (yy-ww) | Freq. (MHz) | Size (GiB) | Geom. (#R, #B) | tRAS (ns) |
|---|---|---|---|---|---|---|
| $\mathbb{S}_0$ | SK Hynix | 22-31 † | 2133 | 8 | 1, 16 | 33.000 |
| $\mathbb{S}_1$ | Micron | 20-41 | 2400 | 16 | 1, 16 | 29.125 |
| $\mathbb{S}_2$ | Micron | 20-48 | 3200 | 8 | 1, 16 | 26.250 |
| $\mathbb{S}_3$ | Samsung | 20-47 | 2666 | 8 | 1, 16 | 32.000 |
| $\mathbb{S}_4$ | Samsung | 20-52 | 2666 | 4 | 1, 8 | 32.000 |
| $\mathbb{S}_5$ | Micron | 22-31 † | 3200 | 16 | 1, 16 | 32.000 |
| $\mathbb{S}_6$ | Samsung | 20-44 | 2133 | 4 | 1, 16 | 33.000 |
| $\mathbb{S}_7$ | Micron | 20-45 | 2400 | 8 | 1, 16 | 32.000 |
| $\mathbb{S}_8$ | Nanya | 20-43 | 2400 | 8 | 1, 16 | 32.000 |
| $\mathbb{S}_9$ | SK Hynix | 22-31 † | 2400 | 16 | 2, 16 | 32.000 |
| $\mathbb{S}_{10}$ | Samsung | 19-34 | 2666 | 8 | 1, 16 | 32.000 |
| $\mathbb{S}_{11}$ | *n/a* | 22-31 † | 2666 | 8 | 1, 16 | 32.000 |
| $\mathbb{S}_{13}$ | Samsung | 22-31 † | 2666 | 16 | 2, 16 | 29.250 |
| $\mathbb{S}_{14}$ | Micron | 22-22 | 2666 | 8 | 2, 16 | 32.000 |
| $\mathbb{S}_{15}$ | Micron | 22-21 | 2666 | 16 | 2, 16 | 32.000 |
| $\mathbb{S}_{16}$ | Samsung | 22-31 † | 3200 | 8 | 1, 16 | 32.000 |
| $\mathbb{S}_{17}$ | Micron | 22-31 † | 3200 | 32 | 2, 16 | 32.000 |
| $\mathbb{S}_{18}$ | SK Hynix | 21-28 | 2666 | 16 | 2, 16 | 32.000 |
| $\mathbb{S}_{19}$ | SK Hynix | 16-25 | 2133 | 16 | 2, 16 | 33.000 |
| $\mathbb{S}_{20}$ | Zentel | 22-31 † | 2400 | 4 | 1, 16 | 32.000 |
| $\mathbb{S}_{21}$ | *n/a* | 22-15 | 2666 | 16 | 2, 16 | 32.000 |
| $\mathbb{U}_0$ | Micron | 22-04 | 4800 | 16 | 1, 32 | 32.000 |
| $\mathbb{U}_1$ | Micron | 21-41 | 4800 | 16 | 1, 32 | 32.000 |
| $\mathbb{U}_2$ | SK Hynix | 22-05 | 4800 | 8 | 1, 16 | 32.000 |
| $\mathbb{U}_3$ | Micron | 22-07 | 4800 | 16 | 1, 32 | 32.000 |
| $\mathbb{U}_4$ | Samsung | 21-52 | 4800 | 8 | 1, 16 | 32.000 |
| $\mathbb{U}_{5,6}$ | Micron | 21-42 | 4800 | 16 | 1, 32 | 32.000 |
| $\mathbb{U}_7$ | Micron | 21-49 | 4800 | 16 | 1, 32 | 32.000 |
| $\mathbb{U}_8$ | Samsung | 22-01 | 4800 | 16 | 1, 32 | 32.000 |
| $\mathbb{U}_9$ | Samsung | 21-42 | 4800 | 16 | 1, 32 | 32.000 |
| $\mathbb{U}_{10,11}$ | Micron | 22-02 † | 4800 | 16 | 1, 32 | 32.000 |
| $\mathbb{U}_{12,13}$ | Samsung | 21-44 † | 5600 | 16 | 1, 32 | 32.000 |
| $\mathbb{U}_{14,15}$ | SpecTek | 21-48 | 4800 | 16 | 1, 32 | 32.000 |

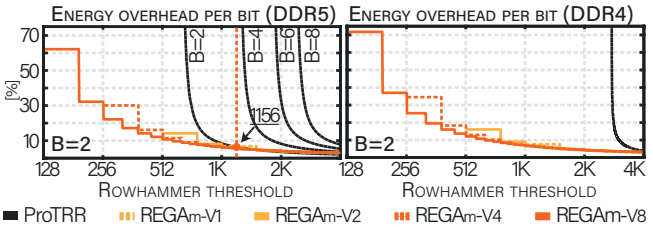


Fig. 16: **Energy overhead per bit of $\text{REGA}_{\text{M1-8}}$.** Energy overhead related to a bit for a fixed chip area. In orange, the point of crossing between $\text{REGA}_\text{M}$ and ProTRR is highlighted. Because $\text{REGA}_\text{M}$ is blast independent, its evaluation for different Bs is identical and in this figure grouped in a single plot. The only difference is a shift in $R_{thresh}$ as described before (§VIII-C).

**Missing standard compliance.** Mithril is not compliant with the current DDR5 standard. It proposes using the new RFM command but fails to adhere to the correct parameters in the respective JEDEC specification [4]. In particular, the authors assume and evaluate an RFM command targeting a specific bank. However, the RFM can only either target all banks (RFMab) or banks that have the same ID in all bank groups (RFMsb). Moreover, the standard specifies RFM periods between 32 and 80 with steps of 8. Mithril, however, assumes an arbitrary RFM period. The standard also requires the bank activation counter to be decreased at every REF, which is another aspect not

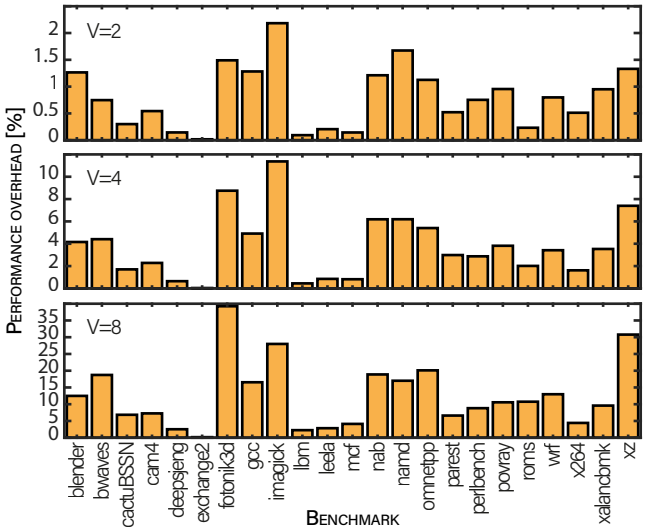


Fig. 17: **Individual performance overhead (DDR5).** Performance overhead for V=2,4,8 for the different benchmarks of SPEC®2017. Geometric mean = 0.8%, 3.7%, 12.7% respectively for V of 2, 4 and 8.

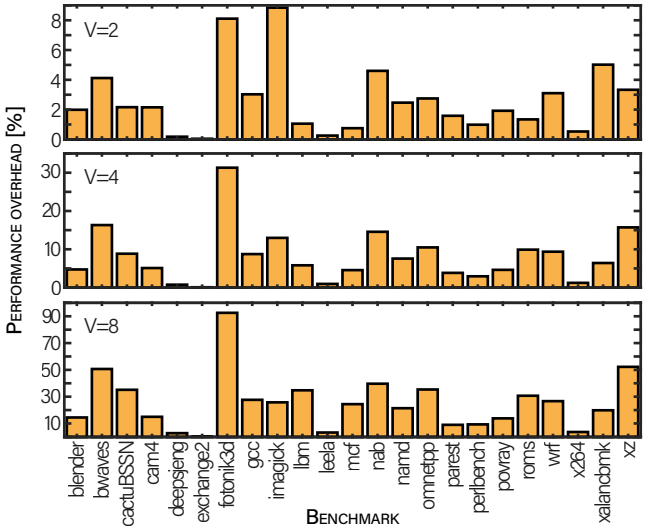


Fig. 18: **Individual performance overhead (DDR4).** Performance overhead for V=2,4,8 for the different benchmarks of SPEC®2017. Geometric mean = 1.6%, 5%, 16% respectively for V of 2, 4 and 8.

considered by Mithril.

**Vulnerabilities.** Mithril uses wrapping counters that expose the design to security issues. The counters can be manipulated to reach *MAX-1* for a row $a_0$, where *MAX* is the maximum number the counter can hold before wrapping around. Now, hammering a different row $a_1$ would lead to it being replaced and subsequently refreshed. At this point, any new hammered row will replace the new minimum (0), and our victim row $a_0$ will never be picked for a refresh until all the others have been refreshed before. Lastly, the effect of the refresh itself is not considered in the paper. It has been shown that refreshes can be used as a vector for exploitation on real-world devices [2], making the mitigation act as a confused deputy.

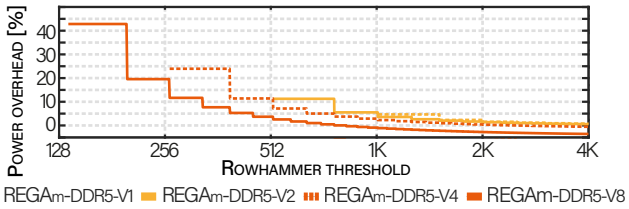


Fig. 19: **Power overhead (DDR5).** Average power overhead for $\text{REGA}_\text{M}$.

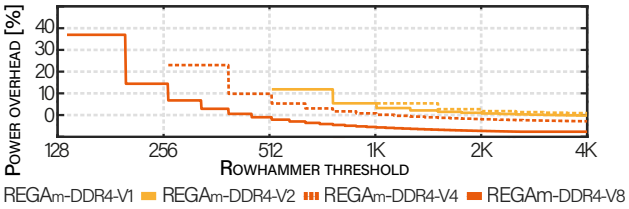


Fig. 20: **Power overhead (DDR4).** Average power overhead for $\text{REGA}_\text{M}$.