Carnegie Mellon University

Scalable and Secure Row-Swap: Efficient and Safe Row Hammer Mitigation in Memory Systems

March. 27. 2026

Presenters: Yi-Ching Chen, Nicolas Keck, Jaehyun Lim

Authors

Jeonghyun Woo
PhD Student at University of British Columbia.

Prashant Nair
Associate Professor at University of British Columbia

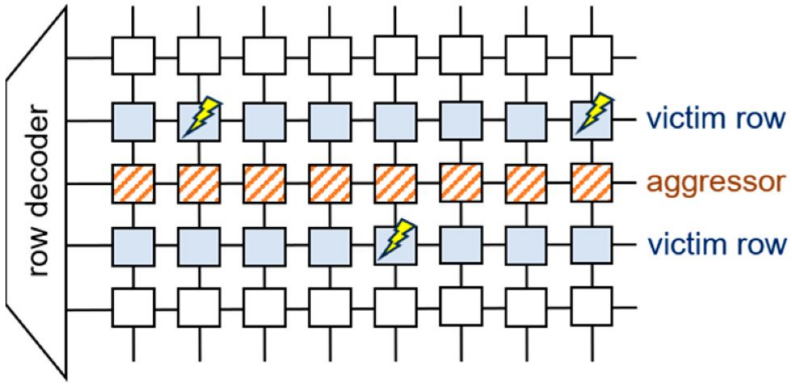
Gururaj Saileshwar

NVIDIA Researcher (2023),

Assistant Professor at University of Toronto.

What is Row Hammer?

- Repeatedly activating a DRAM row can cause bit flips in adjacent rows
- Due to electrical interference and charge leakage
- If activation count exceeds threshold (T_{RH}) → bit flip
- It matters since it causes Data corruption/ security breaches and gets worse with scaling (TRH decreasing over generation)



(a)

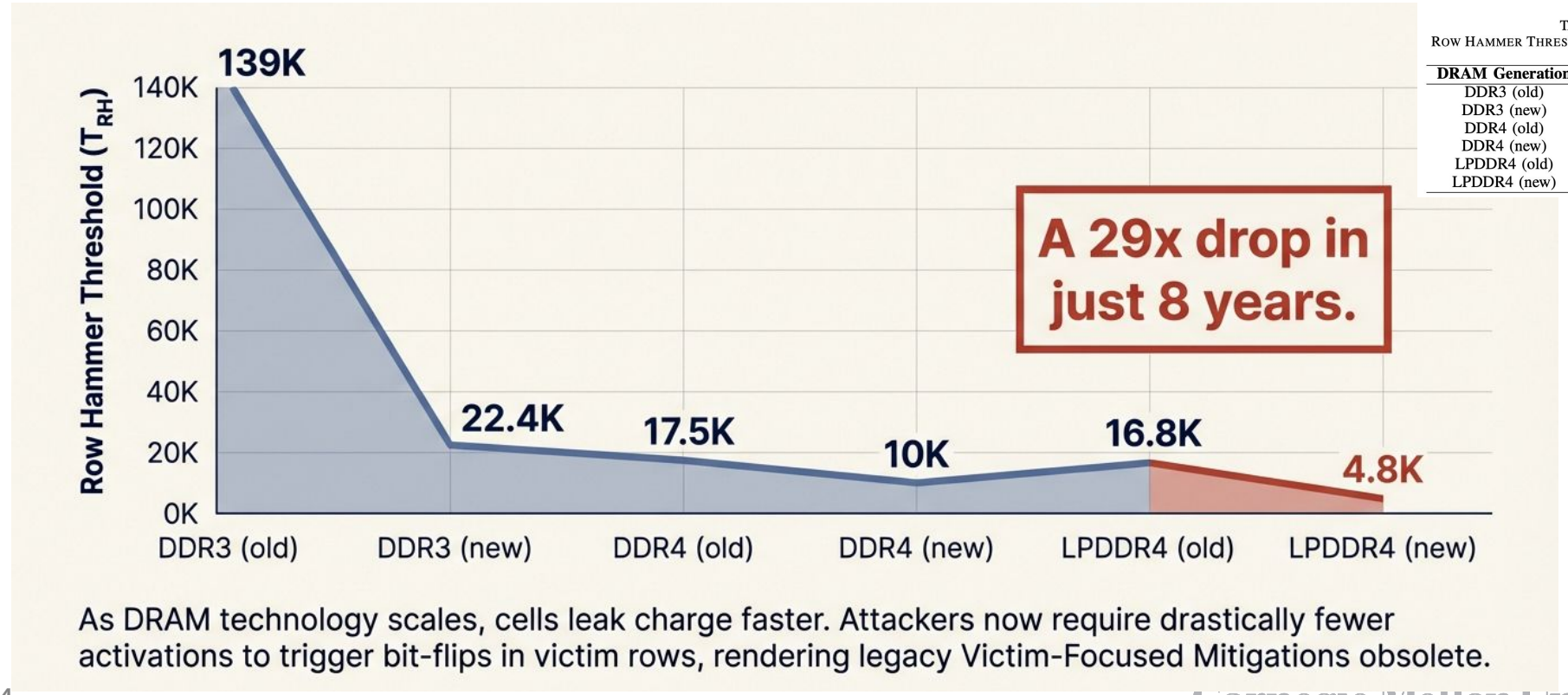
```
hammer:
    mov (X), %eax    // read from address X
    mov (Y), %ebx    // read from address Y
    clflush (X)      // flush cache for address X
    clflush (Y)      // flush cache for address Y
    mfence
    jmp hammer
```

(b)

Figure reference : Rowhammer Attacks in Dynamic Random-Access Memory and Defense Methods

The Row Hammer Threshold is collapsing

- Performance - TRH drops in future generations (it has dropped 29×in 8 years.)



As DRAM technology scales, cells leak charge faster. Attackers now require drastically fewer activations to trigger bit-flips in victim rows, rendering legacy Victim-Focused Mitigations obsolete.

TABLE I
ROW HAMMER THRESHOLD – FROM 2014 TO 2021

DRAM Generation	RH-Threshold
DDR3 (old)	139K [24]
DDR3 (new)	22.4K [21]
DDR4 (old)	17.5K [21]
DDR4 (new)	10K [21]
LPDDR4 (old)	16.8K [21]
LPDDR4 (new)	4.8K [21] - 9K [16]

To prevent from Row Hammer

- Previous method is called **Victim-Focused Mitigation (VFM).**
  - It prevents RowHammer by proactively refreshing neighboring victim rows before the aggressor exceeds the activation threshold.
  - However, it is limited by unknown DRAM row mappings and can be bypassed by attacks like Half-Double.

- Therefore, the limitations led to **Randomized Row-Swap (RRS).**
  - RRS shifts from protecting victims to disrupting attackers by randomizing row locations and breaking spatial correlation between aggressor and victim rows.

RowHammer Mitigation: Security–Performance Trade-off

- Security:
  - Time to break takes >3 years for an untargeted attack.
  - Higher swap rate has even better performance.
- Performance
  - As lower TRH forces more frequent swaps and significantly degrades performance.

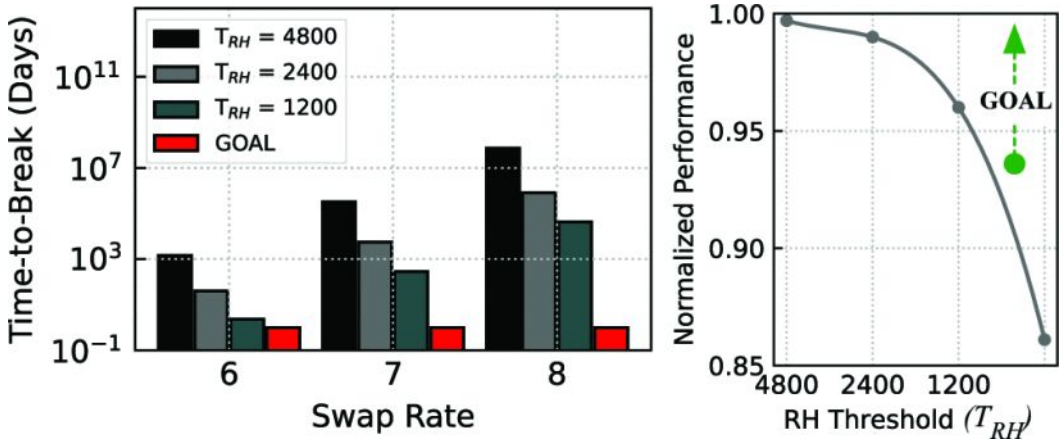


Fig. 1.
(a) Time-to-break (in days) Randomized Row-Swap (RRS) with varying Swap Rate and Row Hammer Thresholds (T_{RH}). Our goal is to break RRS in under 1 day. (b) The normalized performance of RRS as values of T_{RH} vary. Our goal is to minimize the performance overheads of RRS at lower values of $T_{RH}$and enhance security; thereby making it scalable and secure.

Paper target

- **Break RRS faster (Juggernaut Attack)**
  → Can a targeted attack succeed in < 1 day?
- **Stronger security (Secure Row-Swap, SRS)**
  → Defend against Juggernaut + future attacks
- **Scalable low-cost design (Scale-SRS)**
  → Support low swap rate with minimal overhead

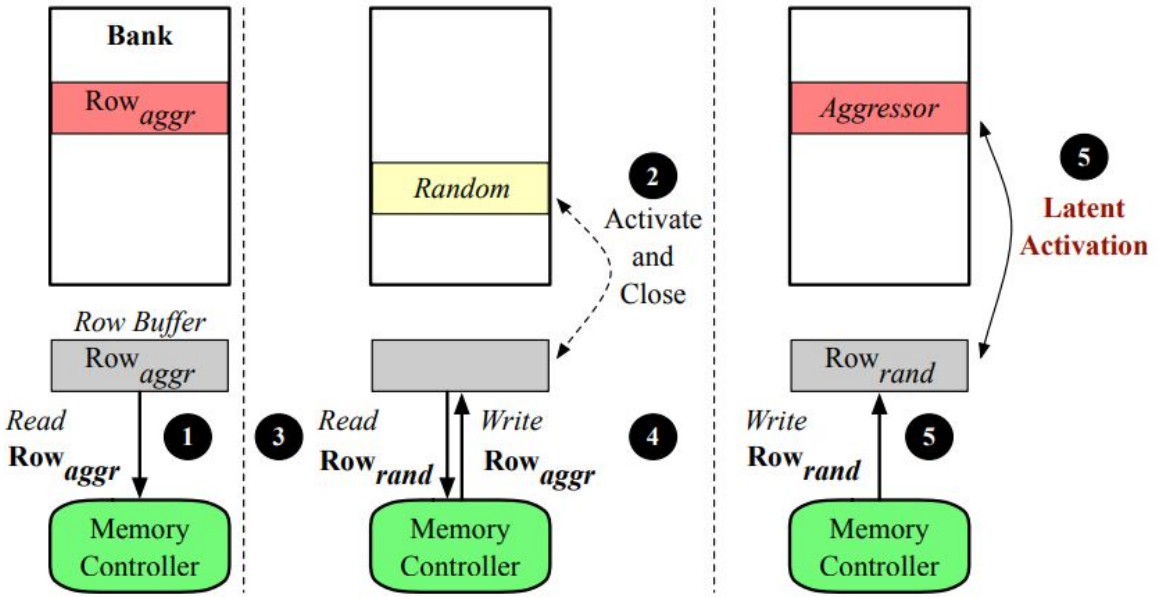
RRS Mechanism - Swap/Unswaps

- RRS swaps affected rows after they receive Ts activations
  - A swap partner is randomly chosen in the memory bank
  - If the row receives another Ts activations, the rows are unswapped, and a new swap partner is chosen

- Why Unswap?
  - Rows must be in their original positions for the memory refresh
  - Chasing long swap chains leads to increased memory latency (worst-case 7% slowdown)

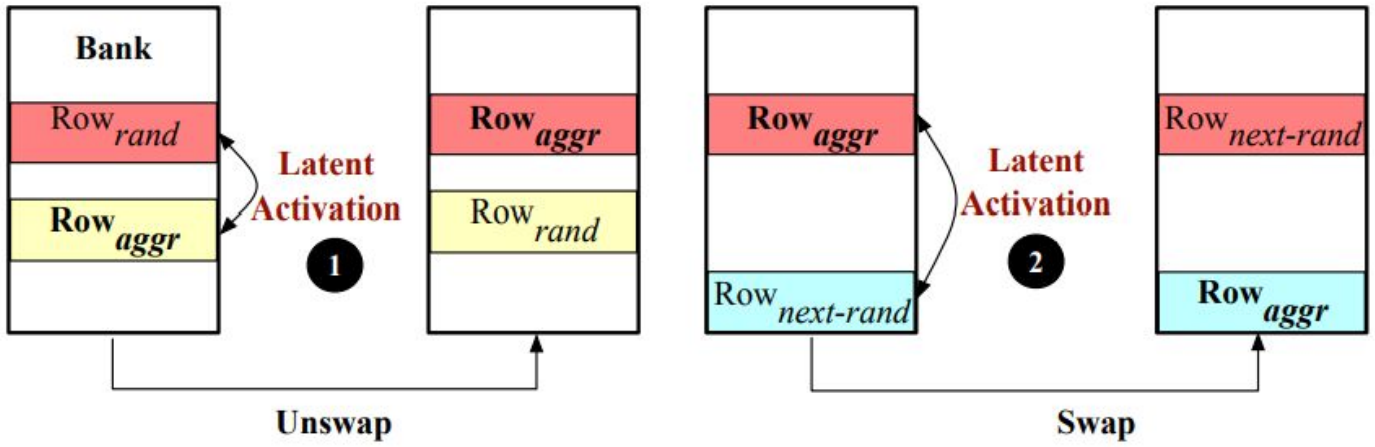
Swapping Cost - Latent Activations

- A swapping operation inherently increases the number of row activations
  - Writing the random row to the attacked row incurs an extra write → Extra activation



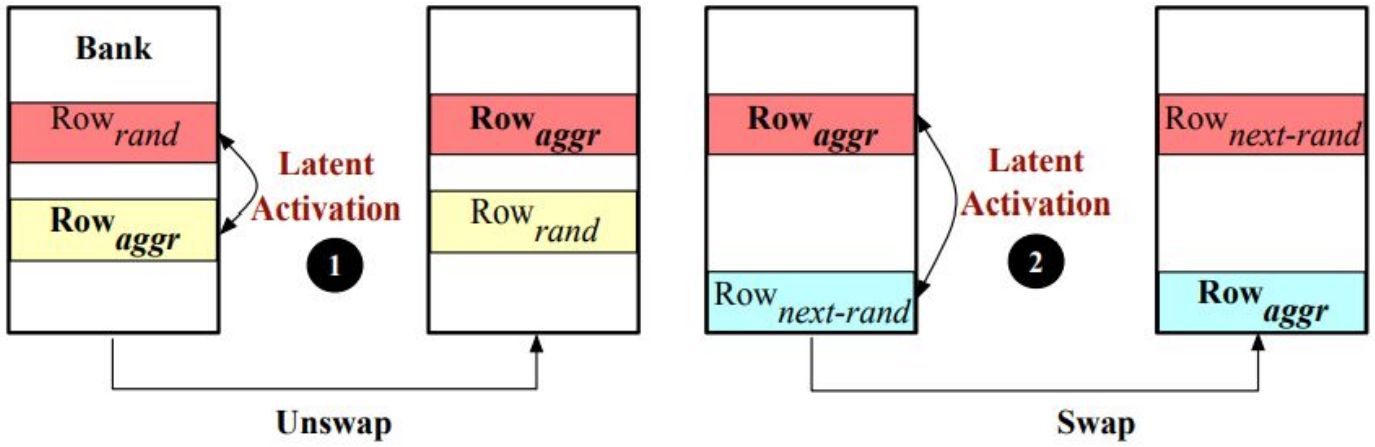
Swapping Cost - Latent Activations

- Each time RRS unswaps and then swaps a row, the row experiences 2 latent activations
  - Generalizing to N sets of Ts activations, the row will experience an additional (2N+1) latent activations
  - Takeaway → Ever 800 (Ts) activations of Row_{aggr} gives you 2 activations on the row's **original physical address**



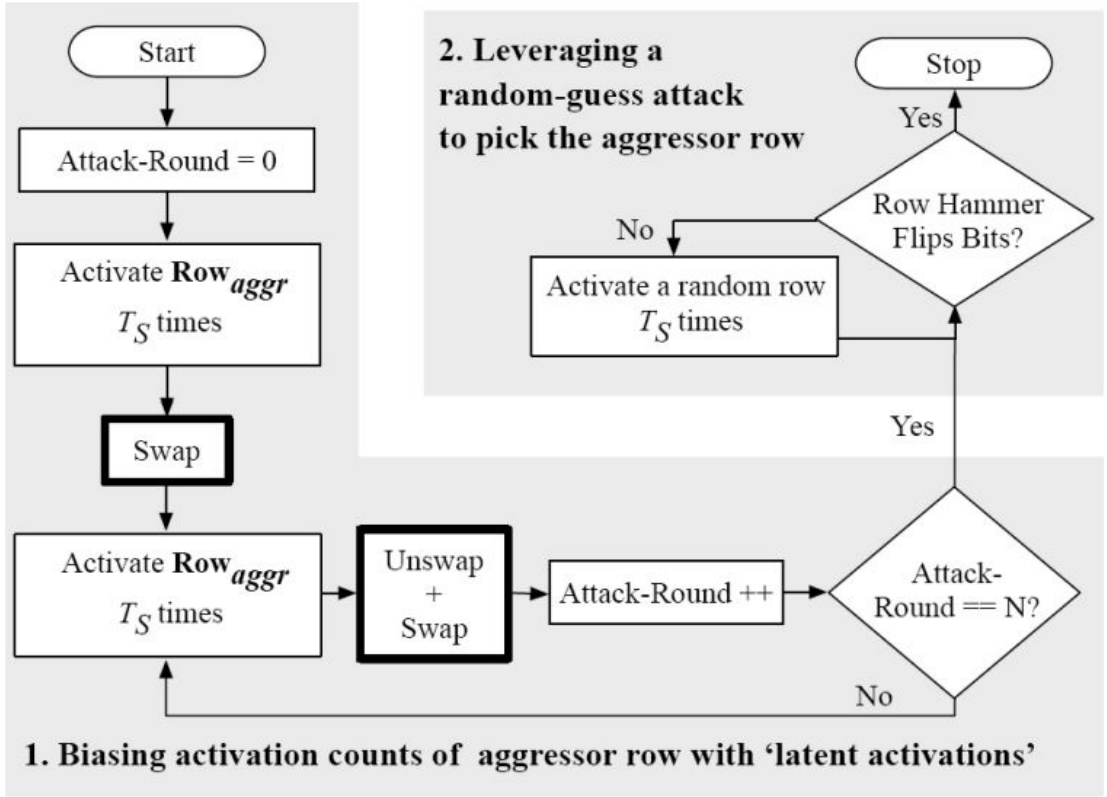
Implications - Latent Activations

- If you could activate the aggressor row's address many many times in a single refresh period, you could probably hit Trh with just latent activations
  - For Trh = 4.8k, Ts = 800, no. naive activations = 1,600,800
  - The good news → Too many activations for a reasonable T_{rh}
  - The bad news → Modern nodes have increasingly lower T_{rh}



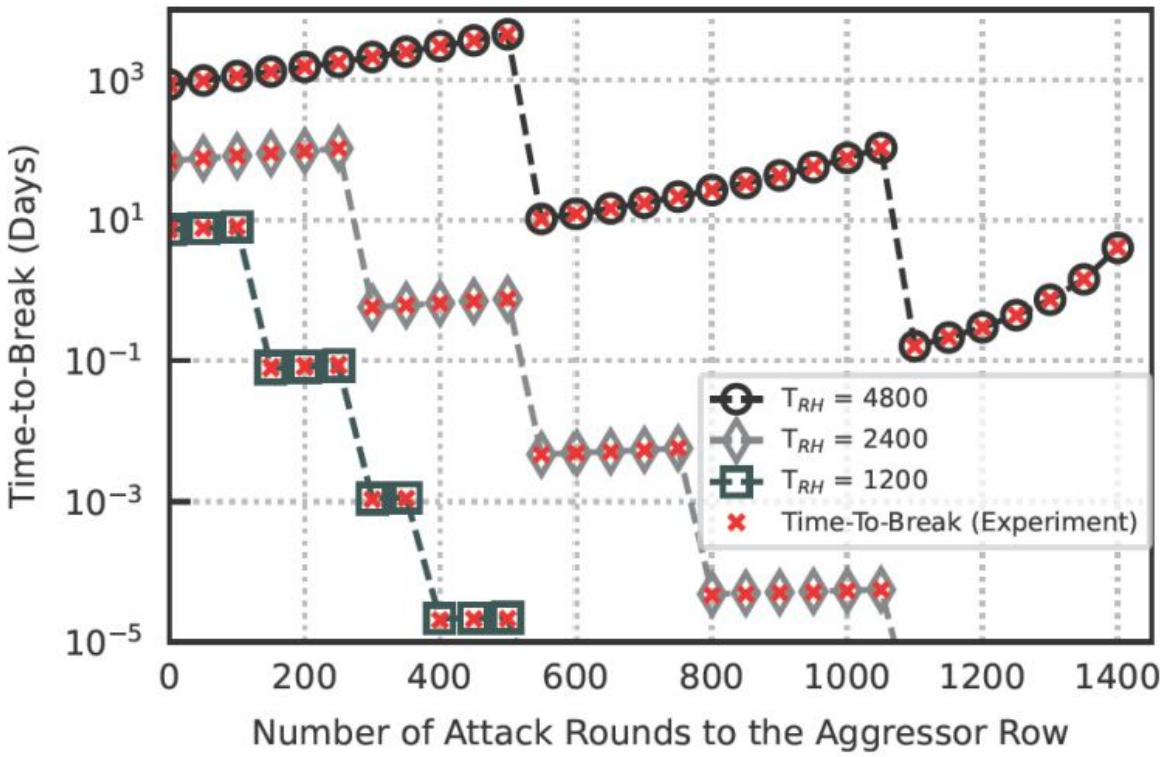
The ugly news - Juggernaut Attack Pattern

- Juggernaut combines latent activations with guessing
  - Use latent activations to reduce the number of successful guesses required
  - Reduces the time to break RRS from years to hours/days



Designing Your Juggernaut

- Details how to pick an optimal number of attack rounds
  - Large drops due to number of latent activations crossing multiples of TS
  - Trade off between time guessing and time naively accessing
  - Juggernaut breaks RRS below 4800 Trh using naive activations

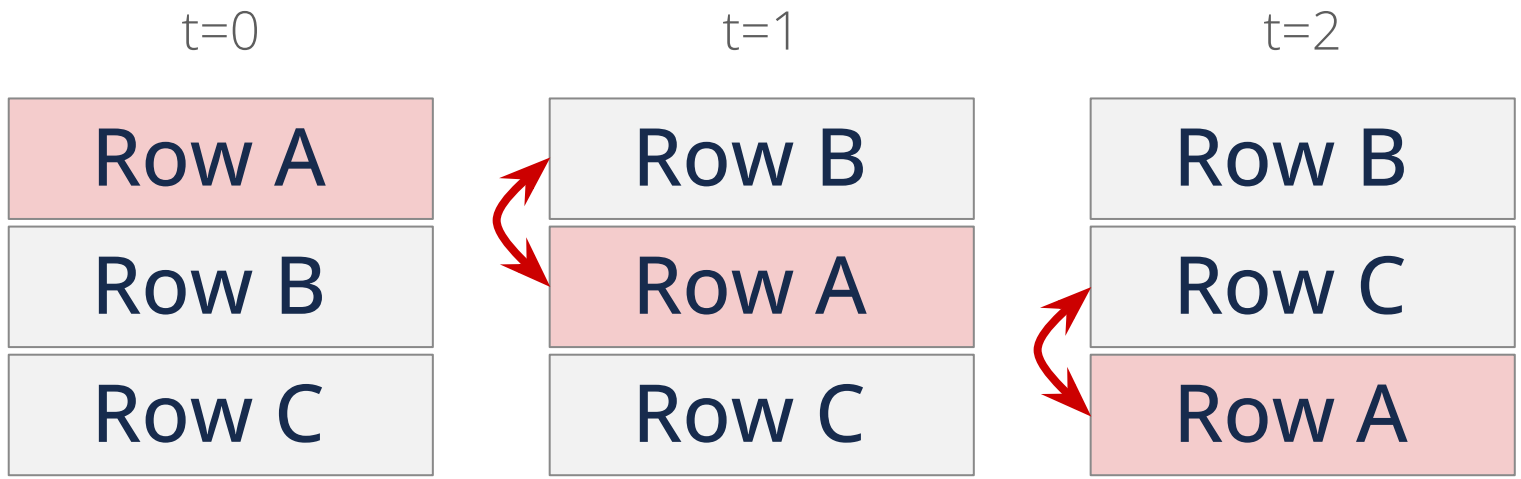


Summary of Flaws in RRS

- **RRS is not secure**
  - The juggernaut attack pattern can break RRS in a day under favorable conditions, rather than the touted years
- **RRS is not scalable**
  - RRS can not adapt to reductions in Trh → no design space for Ts
  - Uses a decent amount of memory bandwidth
- **RRS was not future-proofed**
  - If Juggernaut can crack RRS, what says a new attack can't break our new row hammer solution?
- **Swap/Unswap causes latent activations**
  - The act of unswapping is the largest flaw in the implementation of RRS
  - Yet how can we eschew unswap without performance penalties?

Secure Row Swap to the Rescue!

The big fix: SRS has no unswapping mechanism.



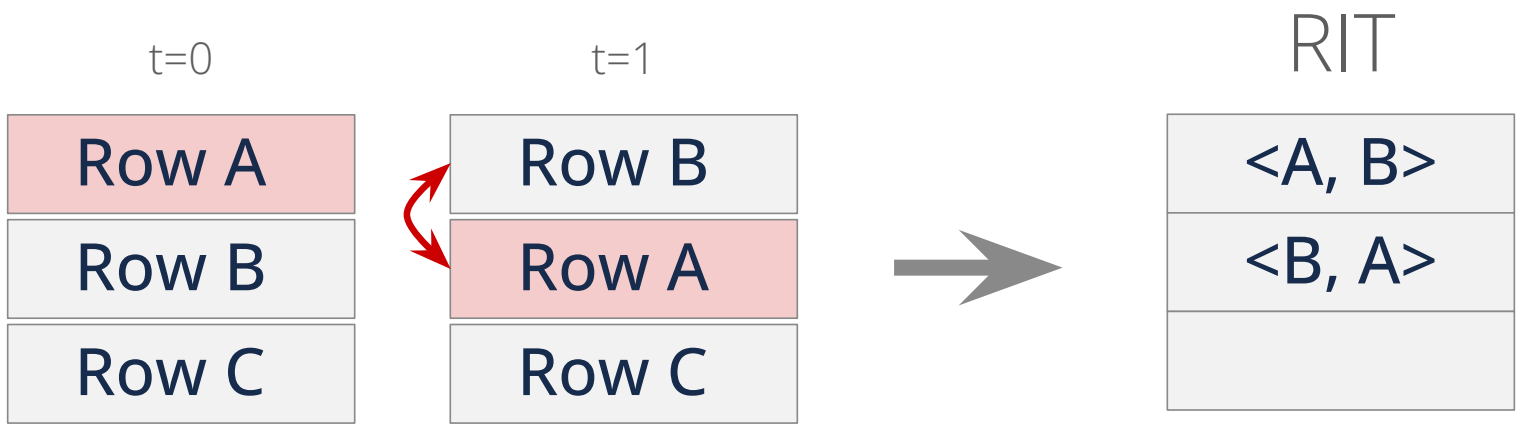
in RRS, in between every swap, A would be unswapped first

SRS Architecture: Mirrored RIT

The Row Indirection Table from RRS is adapted to allow a swap-only SRS.

What's an RIT?

The RIT in the original RRS stores the **fixed** tuples of swapped rows.



SRS Architecture: Mirrored RIT

Requirement #1: Tuples shouldn't be fixed.

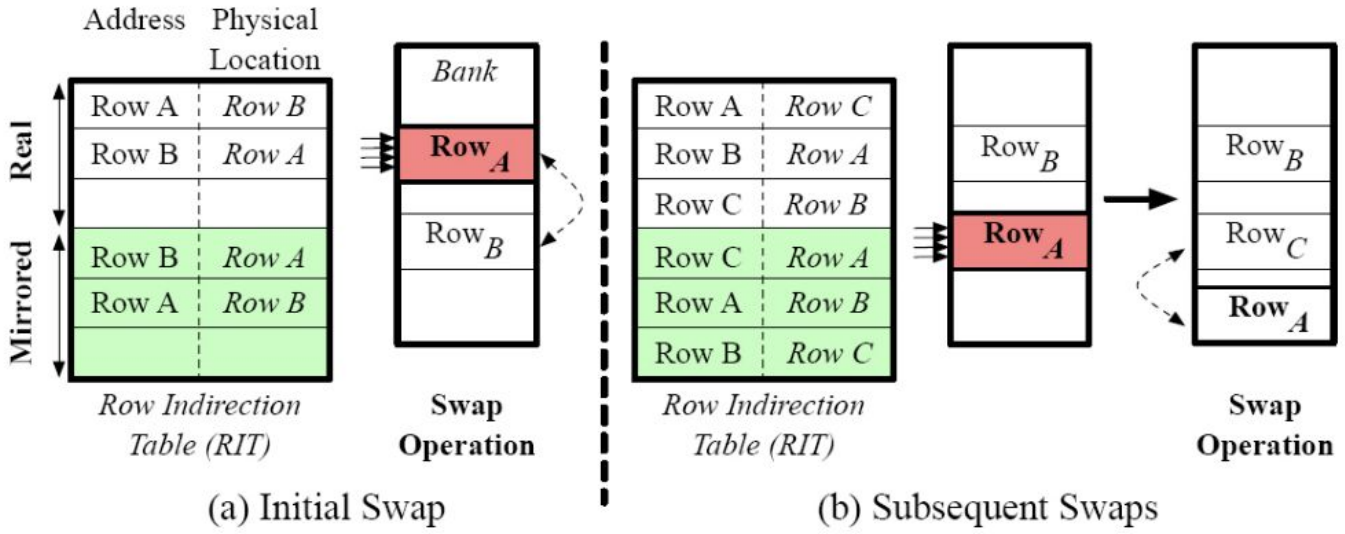


Fig. 9. The RIT (real and mirrored) provides indirections to the rows involved in the *swap* operations in SRS. The tuples in SRS do not have fixed pairs.

SRS Architecture: Mirrored RIT

Requirement #2: The RIT should be reversible

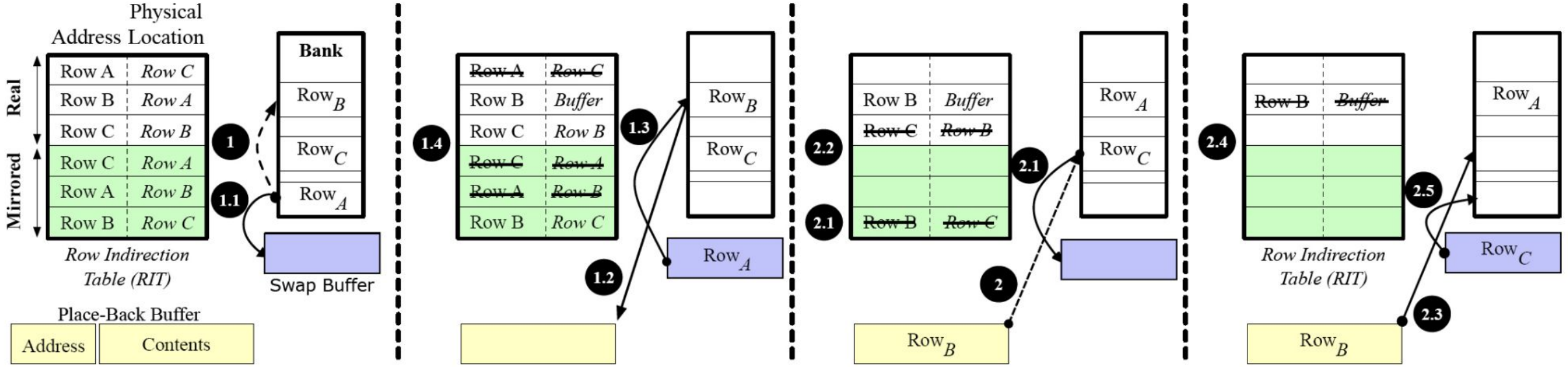
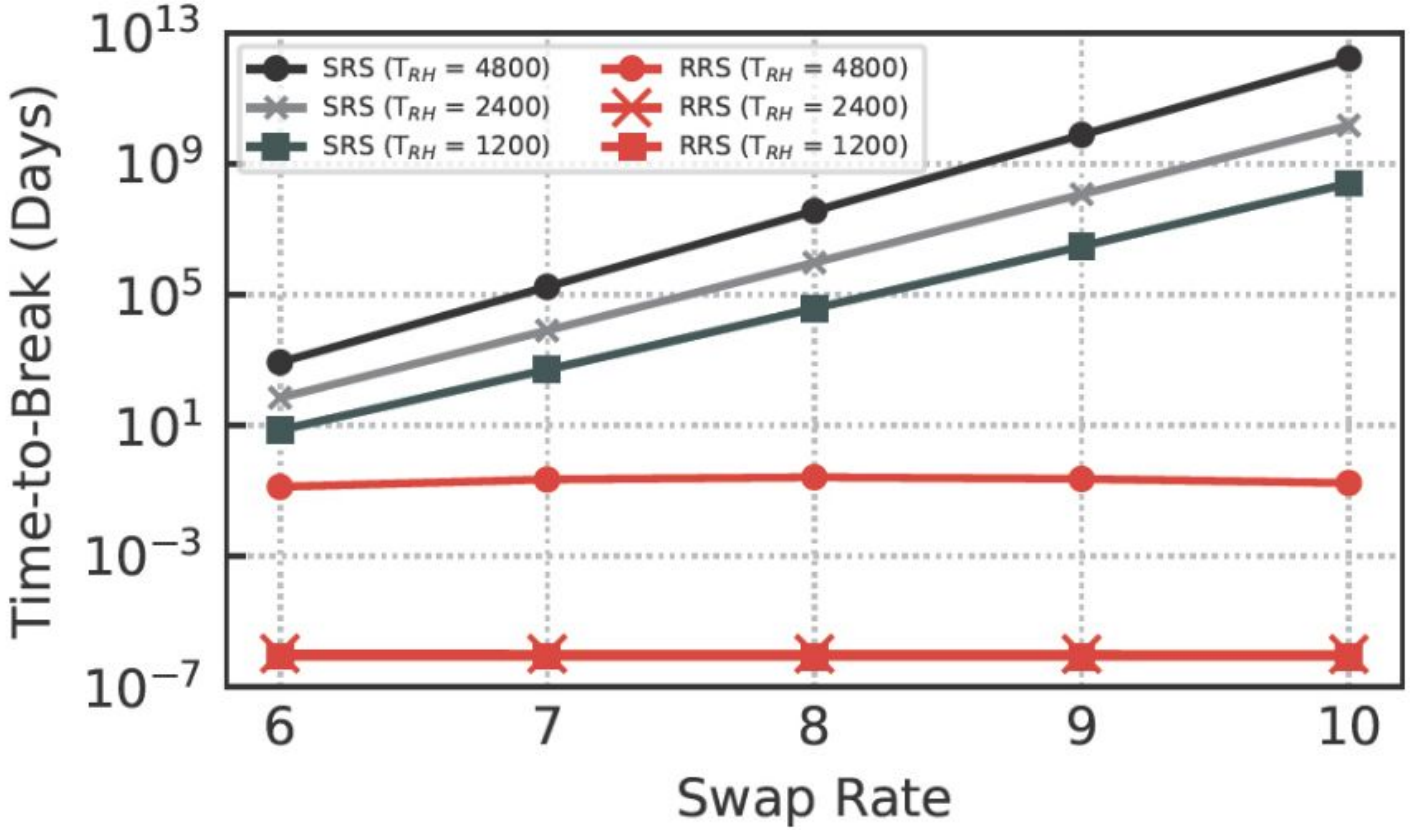


Fig. 8. An overview of the *place-back* operation and *place-back* buffer for enabling Secure Row-Swap (SRS). SRS does not require tuples of row addresses in RIT. The *place-back* buffer helps lazily store the rows that are displaced from the original location.

SRS Security Results

Using the Juggernaut attack, SRS now requires more than 2 years to break even with a high swap rate of 6.



SRS Future Proofing

To be able to detect other patterns of attacks in the future, SRS implements a per-row swap tracker per epoch (twice every refresh period).

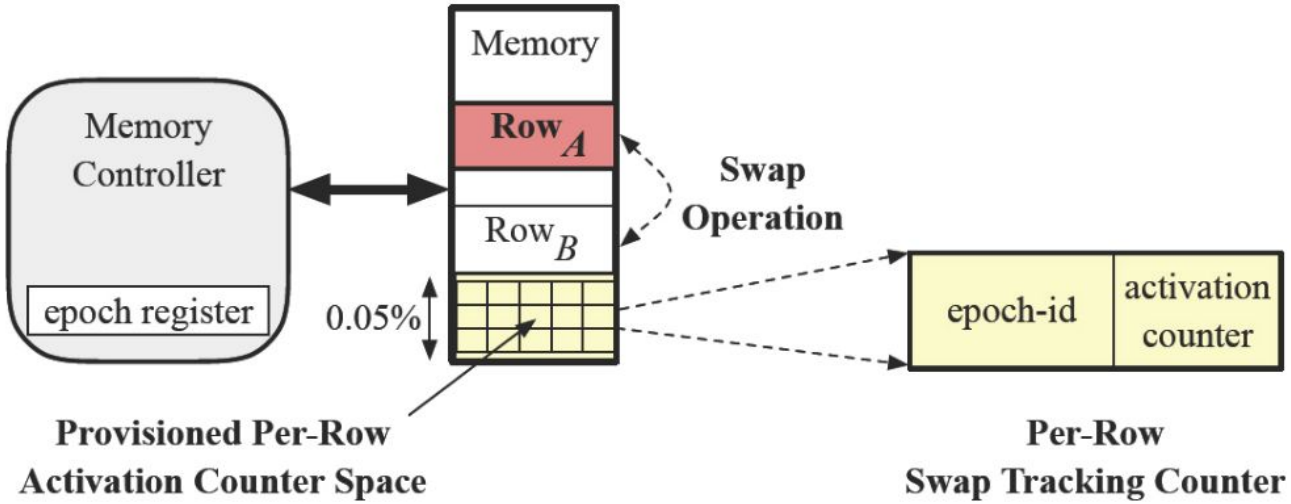
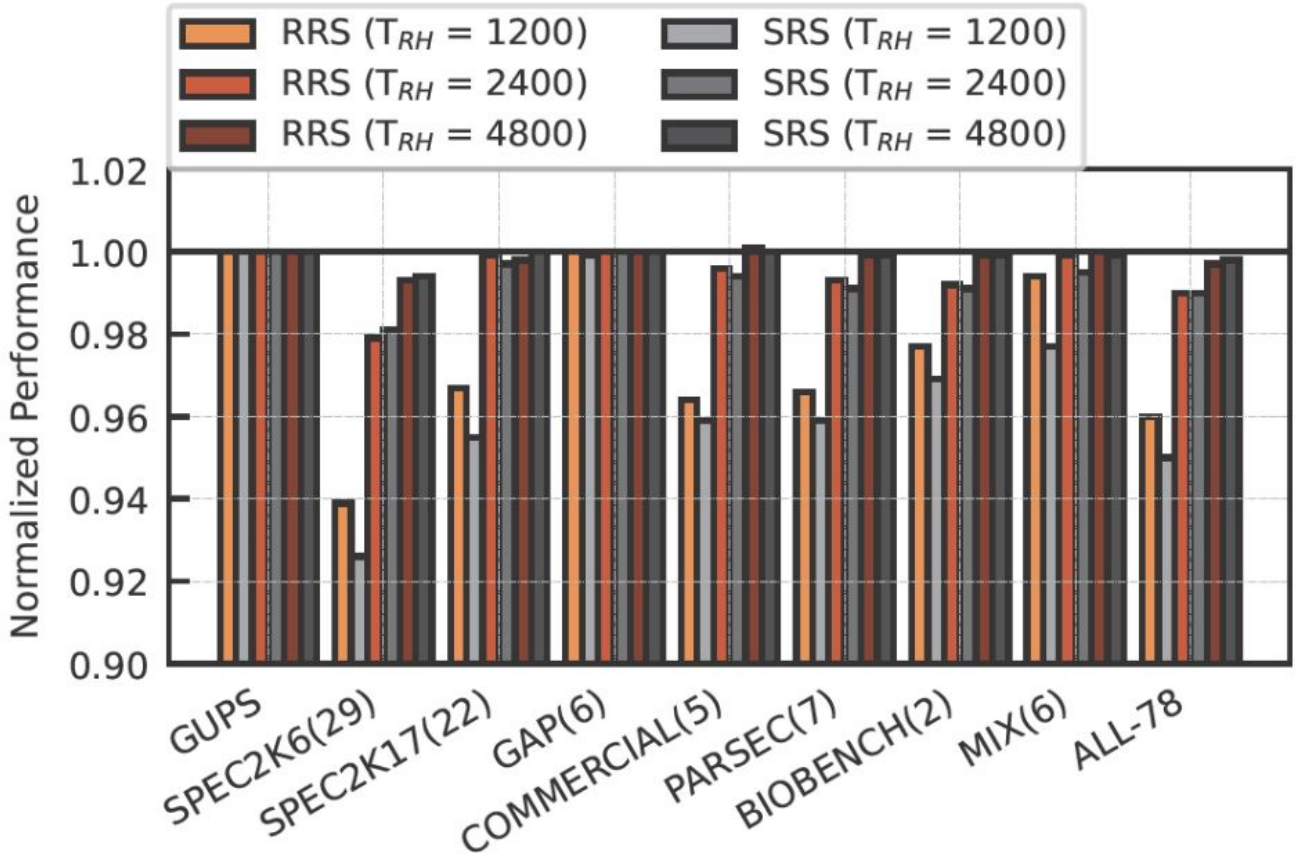


Fig. 11. A memory system with per-row swap-tracking counter. The memory controller stores an epoch register. The main memory reserves 0.05% of its space to store a per-row tracking counter. The respective counter for a row is read and updated before each swap operation.

SRS is not Scalable

SRS still incurs the same memory bandwidth overheads of RRS because swaps can still happen frequently.



Solution: Scalable SRS

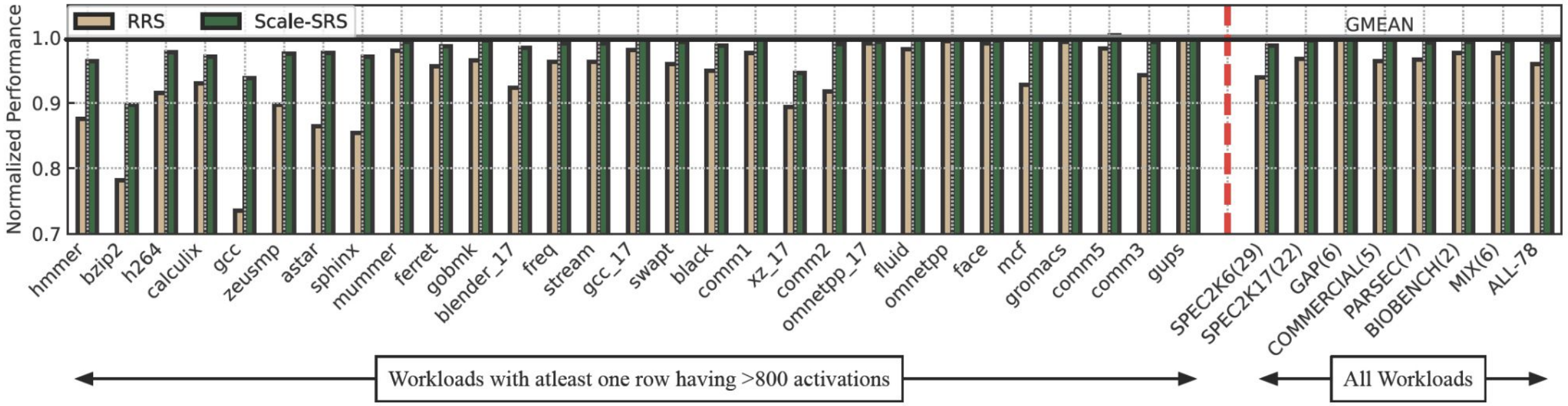
Insight: Only a few aggressor rows receive multiple swaps. To prepare for this worst case, the swap rate needs to be maintained at a high level.

Fix: Pin the frequently accessed aggressor rows in the LLC.

Overhead: At T_{RH}=4800, the LLC needs to store a maximum of 3 DRAM rows in a single bank attack (occurring every 31 days). This is up to 48KB of space, which is 0.05% of an 8MB LLC.

Scalable SRS Evaluation

Performance evaluated across SPEC2006, 2017, GAP, BIOBENCH, PARSEC, and COMMERCIAL benchmarks.



S-SRS: 0.7% average slowdown and RRS: 4% average slowdown

Scalable SRS Evaluation

Even at T_{RH}=512, S-SRS slowdown is 4%, RRS is 14%

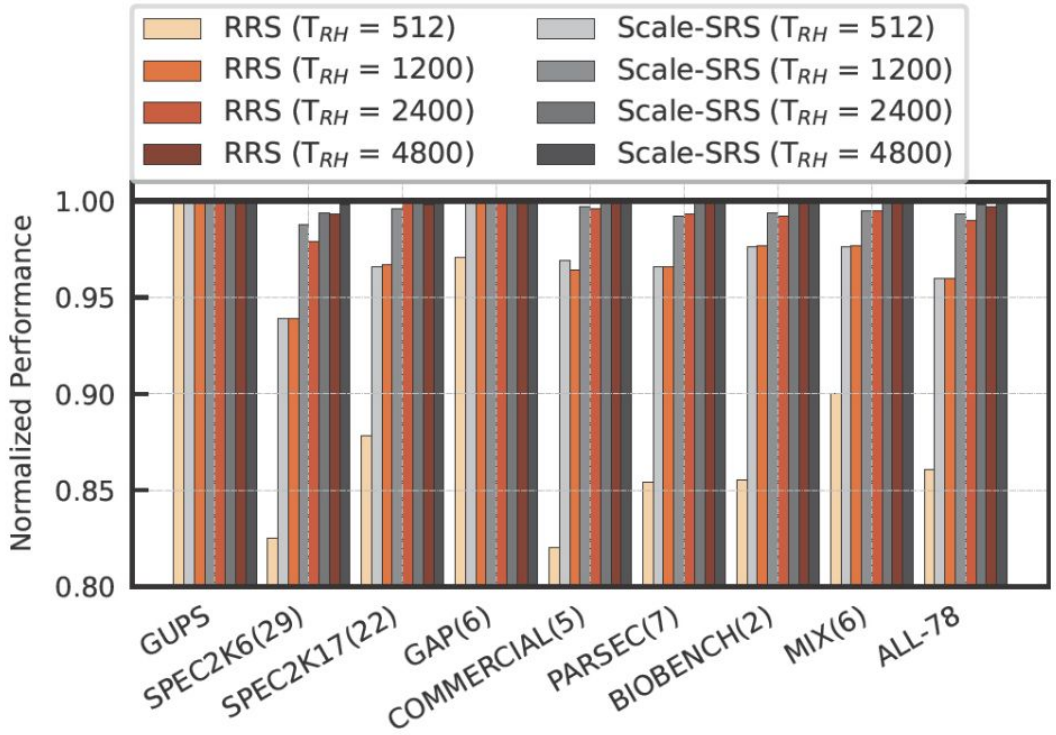


Fig. 15. The normalized performance of SRS and RRS as the value of T_{RH} varies from 4800 to 512. Even at a T_{RH} of 512, Scale-SRS shows an average slowdown of only 4%, whereas RRS shows an average slowdown of 14%.

Storage and Power Overhead

TABLE IV
STORAGE OVERHEAD PER BANK

Structure	$T_{RH} = 4800$		$T_{RH} = 2400$		$T_{RH} = 1200$	
	RRS	**Scale-SRS**	**RRS**	**Scale-SRS**	**RRS**	**Scale-SRS**
RIT	35 KB	9.4KB	130KB	35KB	250KB	67.5KB
Swap-Buffer	1 KB	1 KB	1KB	1KB	1KB	1KB
Place-Back Buffer	-	8KB	-	8KB	-	8KB
Epoch Register	-	19 bits	-	19 bits	-	19 bits
Pin Buffer	-	289 bytes	-	420 bytes	-	420 bytes
Total	36 KB	18.7KB	131KB	44.4KB	251KB	76.9KB

TABLE V
EXTRA POWER CONSUMPTION PER CHANNEL ($T_{RH} = 4800$)

Type of Power Overhead	**RRS**	**Scale SRS**
DRAM Power Overhead (Row-Swap)	0.5%	0.2%
SRAM Power Overhead	903 mW	703 mW

Discussion and work limitation

1. **Internal Chip Address vs Physical Address:**
   Scale-SRS can operate on physical addresses, but can also be adapted to internal DRAM row addresses without changing the design or security guarantees.

2. **Near-Memory / In-Memory Implementation:**
   Scale-SRS can be implemented in the memory controller or moved closer to DRAM (near-memory/in-memory), enabling compatibility with emerging interfaces like CXL.

3. **Effect of Open-Page Policy**
   Open-page policy can reduce Juggernaut attack effectiveness by lowering activation frequency, but becomes ineffective as TRH decreases in future DRAM.

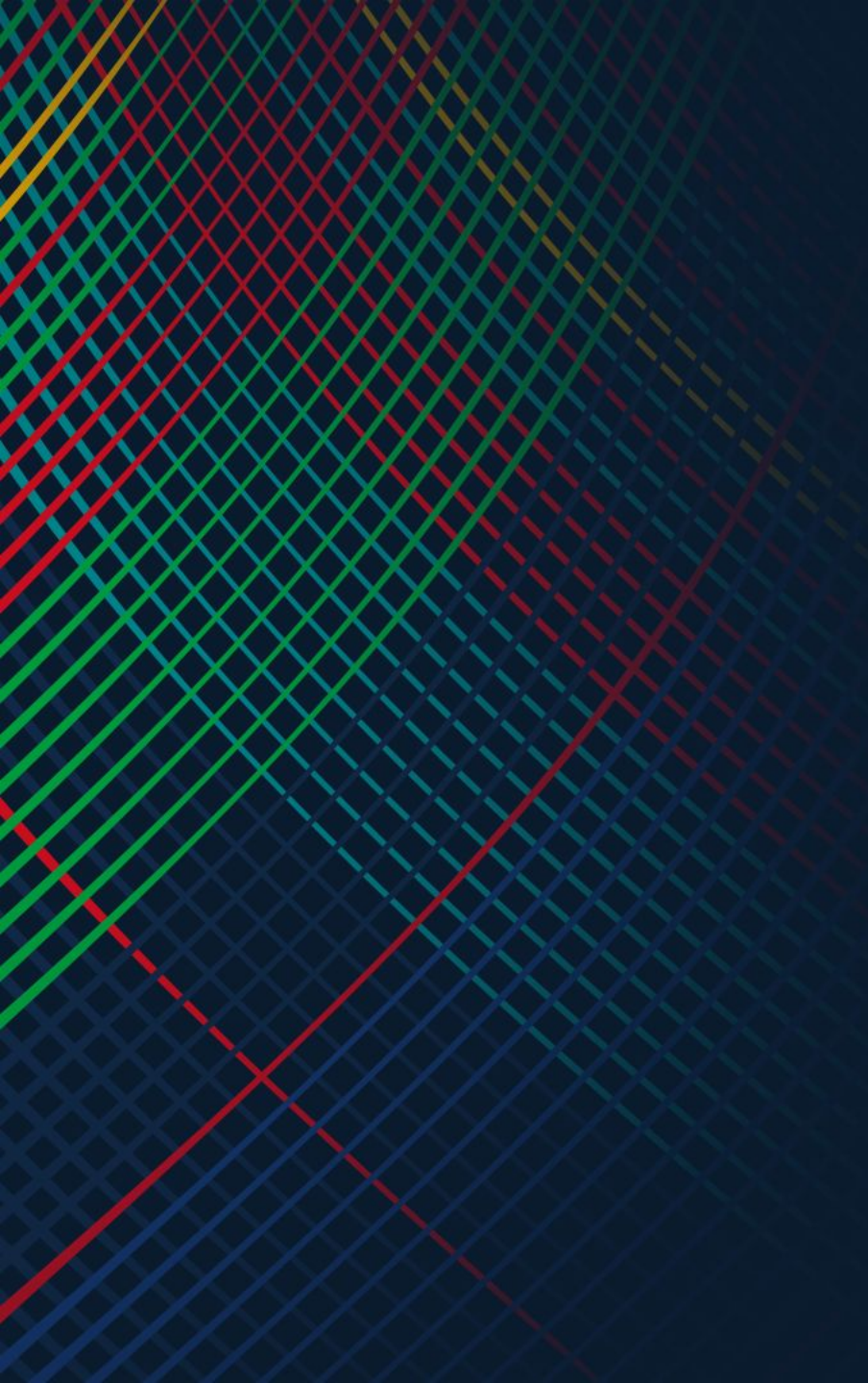
Discussion and work limitation

4. **Storage Optimization Opportunity**
   The storage overhead of Scale-SRS can be further reduced by optimizing the RIT design, such as eliminating mirrored entries.

5. **Future DRAM and Scalability**
   As TRH continues to decrease in future DRAM, attacks like Juggernaut remain effective, highlighting the need for scalable defenses like Scale-SRS.

Conclusion

- **RRS is not secure nor scalable**
  → Vulnerable to **Juggernaut (breaks in <1 day)**
  - **Key issue: latent activations**
    → Defense itself introduces new vulnerability
- **Proposed: Scale-SRS**
  → Eliminates latent activation
  → Robust against current & future attacks
- **Better scalability** → Works with **lower swap rate**
- **Low overhead** → **0.7% slowdown, 3.3× less storage** (vs RRS)



Questions?