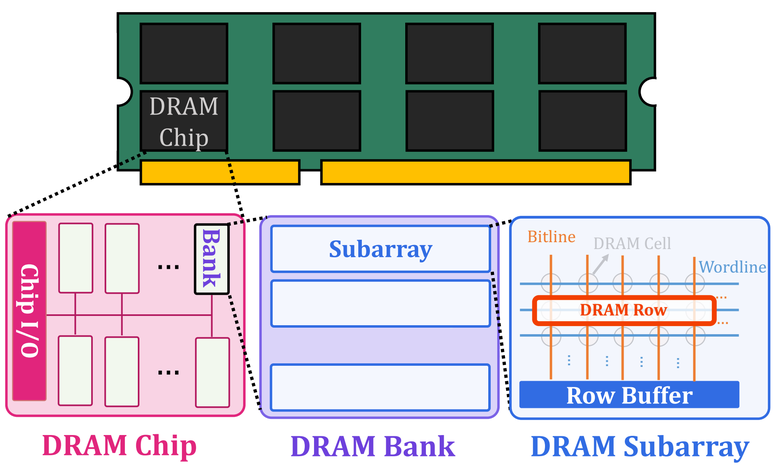
Scalable and Low Overhead DRAM Read Disturbance Mitigation

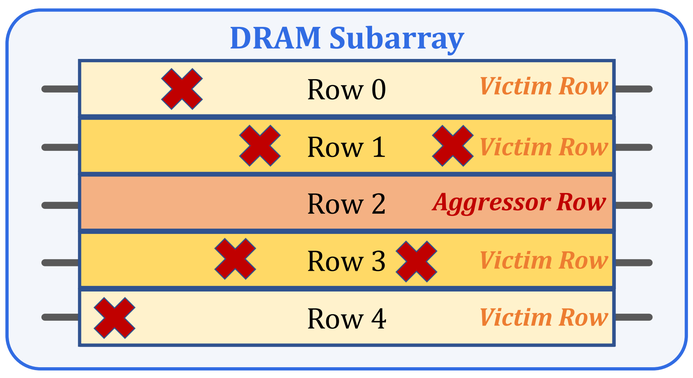
Nisa Bostanci Ataberk Olgun Giray Yaglikci Geraldo de Oliveira Yahya Tugrul
Haocong Luo Ismail Yuksel Konstantinos Kanellopoulos Mohammad Sadrosadati Onur Mutlu

CoMeT ABACuS

1: DRAM Organization



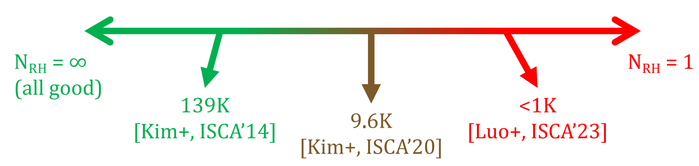
2: Read Disturbance Vulnerabilities



- Repeatedly **activating** and **precharging** a DRAM row causes **RowHammer bitflips** in nearby cells
- The minimum number of activations that causes a bitflip is called **the RowHammer threshold**

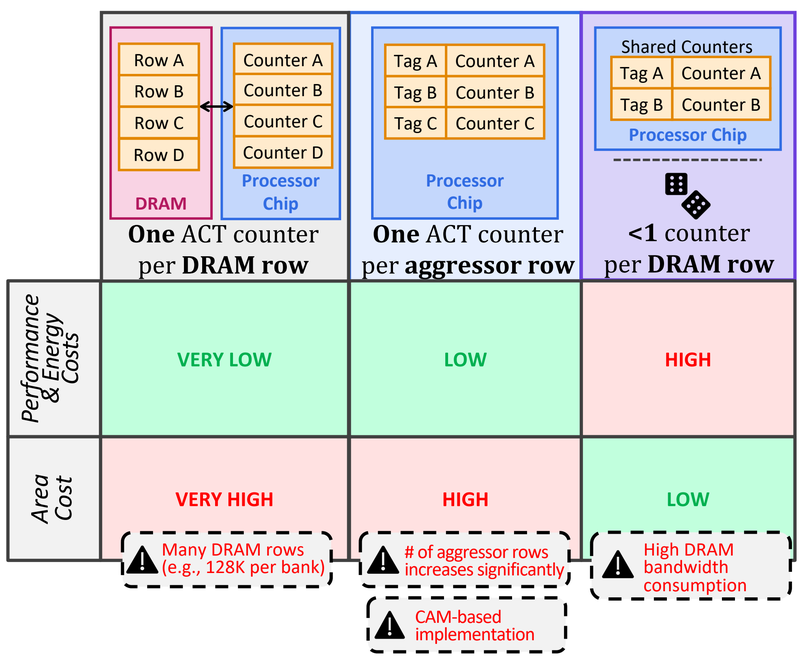
DRAM chips are more vulnerable to read disturbance today

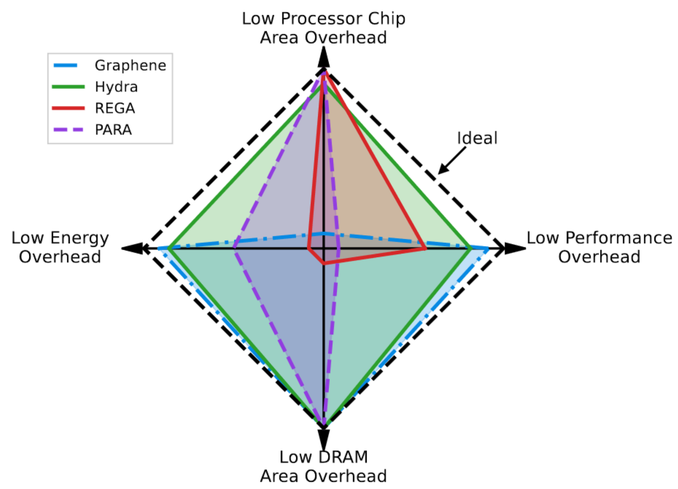
Read disturbance bitflips occur at much lower activation counts (more than two orders of magnitude decrease in less than a decade):



Mitigation techniques against read disturbance attacks need to be **effective** and **efficient** for highly vulnerable systems

3: Limitations of Existing Mitigations





No existing mitigation technique prevents RowHammer bitflips at low area, performance and energy costs

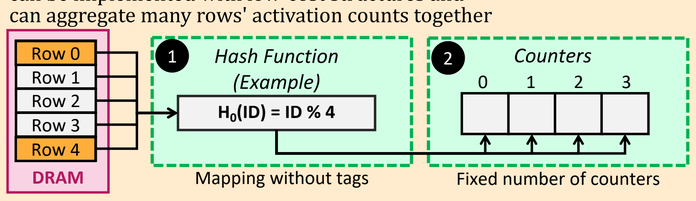
4: Goal

Prevent RowHammer bitflips with low area, performance, and energy overheads in highly RowHammer-vulnerable DRAM-based systems (e.g., a RowHammer threshold of 125)

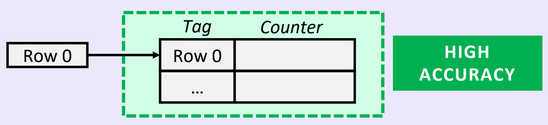
5: CoMeT's Key Observation and Key Idea

Key Observation

Hash-based counters are low-cost:
(i) can be implemented with low-cost structures and
(ii) can aggregate many rows' activation counts together



Tag-based counters are highly accurate:
Each one tracks one row's activation count



Key Idea

1 Track most DRAM rows' activations with low area cost

2 Track a small set of hot rows accurately

Counter Table (CT):
- Maps each DRAM row to a group of low-cost hash-based counters by employing the Count-Min Sketch technique
- Triggers a preventive refreshes when the aggressor's counter group reaches an activation threshold

Tracks DRAM row activations at low area cost

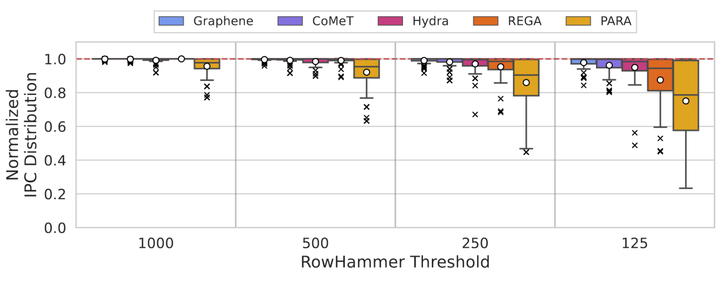
Recent Aggressor Table (RAT):
Allocates highly accurate per-DRAM-row counters for *only* a small set of DRAM rows that are activated many times

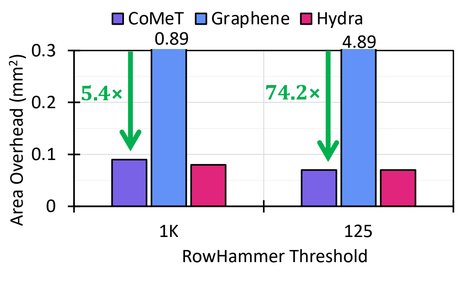
Reduces performance penalties by increasing tracking accuracy

6: CoMeT's Evaluation and Conclusion

Processor	1 or 8 cores, 3.6GHz clock frequency, 4-wide issue, 128-entry instruction window
DRAM	DDR4, 1 channel, 2 rank/channel, 4 bank groups, 4 banks/bank group, 128K rows/bank
Memory Ctrl.	64-entry read and write requests queues, Scheduling policy: FR-FCFS [137, 138] with a column cap of 16 [139]
Last-Level Cache	8 MiB (single-core), 16 MiB (8-core)

Cycle-level simulations using Ramulator
Workloads:
62 1- & 8-core workloads
Four different very low nRH values: 1000, 500, 250, 125
Four state-of-the-art mitigation mechanisms
Graphene, Hydra, PARA, REGA





CoMeT achieves a good trade-off between area, performance and energy costs
- incurs significantly less area overhead (74.2×)
- outperforms the state-of-the-art (by up to 39.1%)

Open-sourced: *https://github.com/CMU-SAFARI/CoMeT*

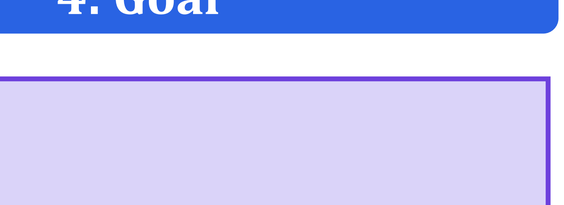
Artifact available, functional, and reproduced

Bostanci et al., **"CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost,"** in HPCA, 2024

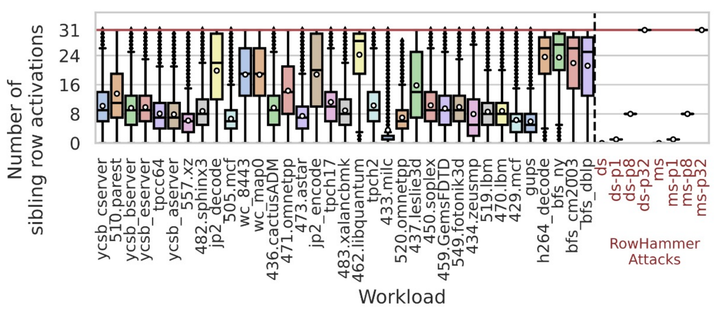
7: ABACuS's Key Observation and Key Idea

1 Address mappings distribute consecutive cache blocks to different banks (but to the same row ID)

2 Workloads tend to access cache blocks in close proximity at around the same time

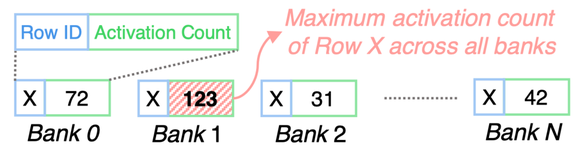


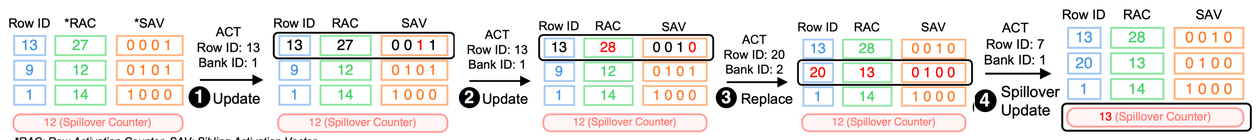
Many workloads access the same row ID in different banks at around the same time
Sibling rows: Rows with the same ID across all banks



Key Idea: Sibling rows can share one hardware counter
Reduce the number of counters by a factor of the number of banks in the chip

Key Mechanism: ABACuS
Track the maximum (worst) activation count of sibling rows using one counter



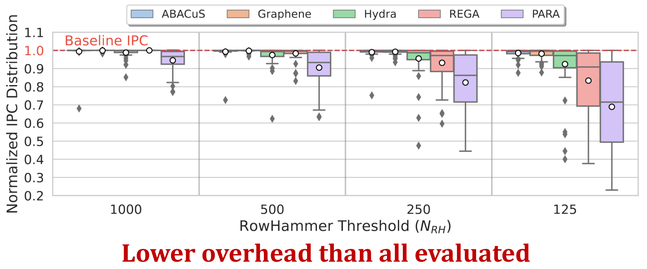


*RAC: Row Activation Counter, SAV: Sibling Activation Vector

8: ABACuS's Evaluation and Conclusion

Processor	1 or 8 cores, 3.6GHz clock frequency, 4-wide issue, 128-entry instruction window
DRAM	DDR4, 1 channel, 2 rank/channel, 4 bank groups, 4 banks/bank group, 128K rows/bank, 3200 MT/s
Memory Ctrl.	64-entry read and write requests queues, Scheduling policy: FR-FCFS [181, 182] with a column cap of 16 [183], Address mapping: MOP [166, 168] 45 ns tRC, 7.9 µs $tREFI$, 64 ms $tREFW$ 64 ms ABACuS reset period
Last-Level Cache	2 MiB per core

Cycle-level simulations using Ramulator
Workloads:
62 1- & 8-core workloads
Four different very low nRH values: 1000, 500, 250, 125
Four state-of-the-art mitigation mechanisms
Graphene, Hydra, PARA, REGA



Lower overhead than all evaluated state-of-the-art mechanisms

Mitigation Mechanism	SRAM KB	CAM KB	Area mm²	Area % CPU	Area % DRAM	Access Energy (pJ)	Static Power (mW)
N_{RH} = 1000							
ABACuS	**10.63**	**8.30**	**0.04**	**0.02**	-	**25.98**	**12.22**
Row ID Table	-	5.64	0.01	<0.01	-	12.85	6.61
Row Activation Counter Table	-	2.66	0.02	<0.01	-	11.13	4.66
Sibling Activation Vector	10.63	-	0.01	<0.01	-	2.00	0.95
PARA [1]	-	-	-	<0.01	-	-	-
Graphene [102]	-	**286.51**	**0.81**	**0.35**	-	**873.38**	**187.98**
Hydra [106]	**61.56**	-	**0.10**	**0.04**	-	**43.07**	**24.17**
REGA [177]	-	-	-	-	**2.06**	-	-

Mitigation Mechanism	SRAM KB	CAM KB	Area mm²	Area % CPU	Area % DRAM	Access Energy (pJ)	Static Power (mW)
N_{RH} = 125							
ABACuS	**85.00**	**66.41**	**0.25**	**0.11**	-	**36.87**	**50.54**
Row ID Table	-	45.16	0.12	0.05	-	20.64	27.56
Row Activation Counter Table	-	21.25	0.06	0.03	-	11.66	15.53
Sibling Activation Vector	85.00	-	0.07	0.03	-	4.57	7.44
PARA [1]	-	-	-	<0.01	-	-	-
Graphene [102]	-	**2037.09**	**5.68**	**2.43**	-	**1042.49**	**1385.52**
Hydra [106]	**56.5**	-	**0.07**	**0.03**	-	**40.26**	**23.21**
REGA [177]	-	-	-	-	**2.06**	-	-

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Open-sourced: *https://github.com/CMU-SAFARI/ABACuS*

Olgun et al., **"ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation,"** in USENIX Security, 2024