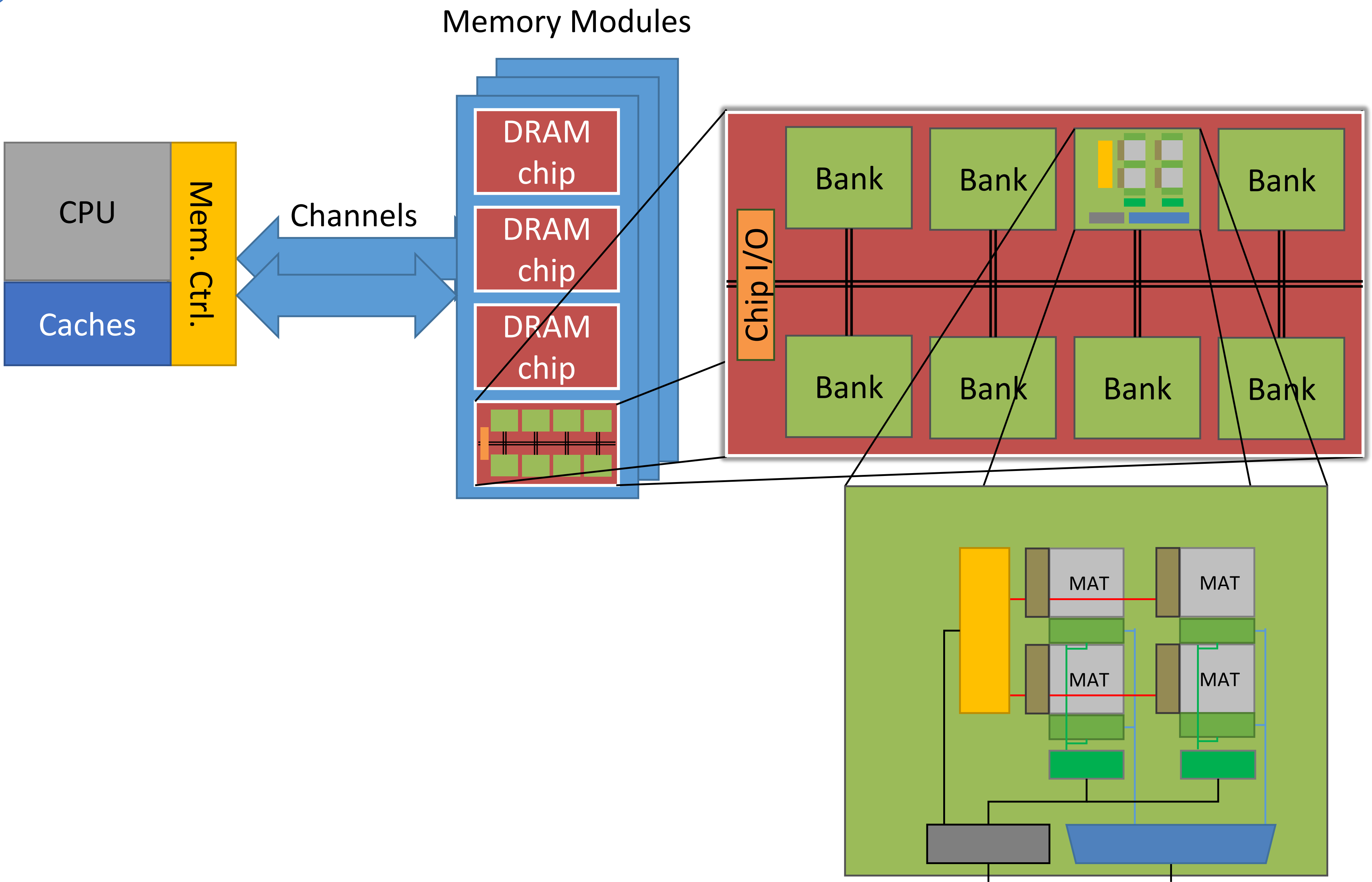
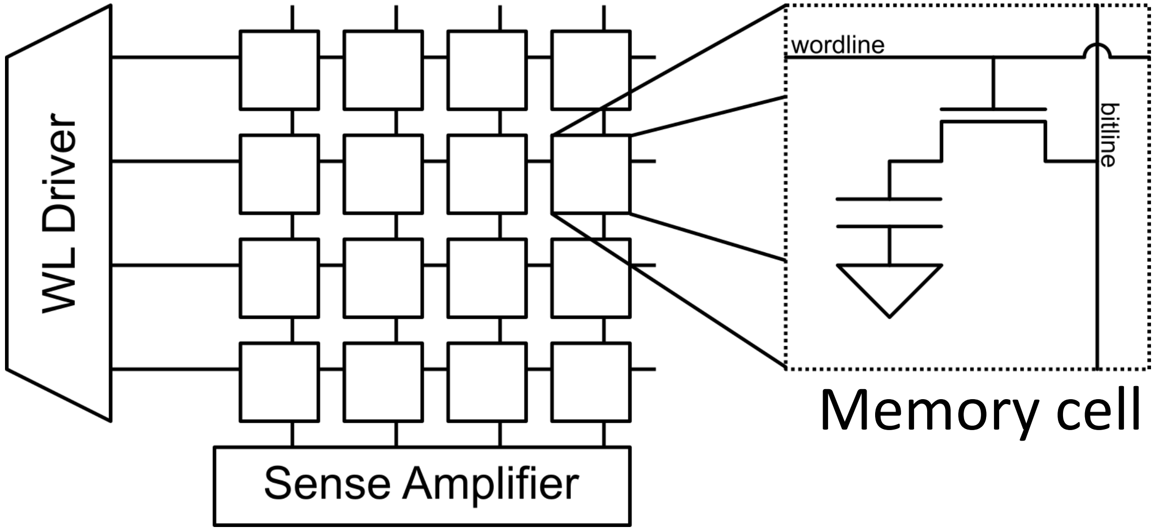
Reducing the Silicon Area Overhead of Counter-Based Rowhammer Mitigations

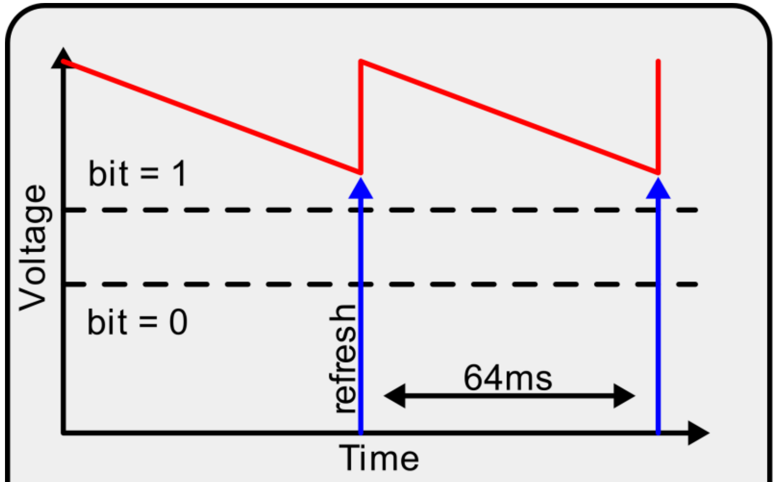
Loïc FRANCE, Florent BRUGUIER, David NOVO, Maria MUSHTAQ and Pascal BENOIT

DRAM architecture



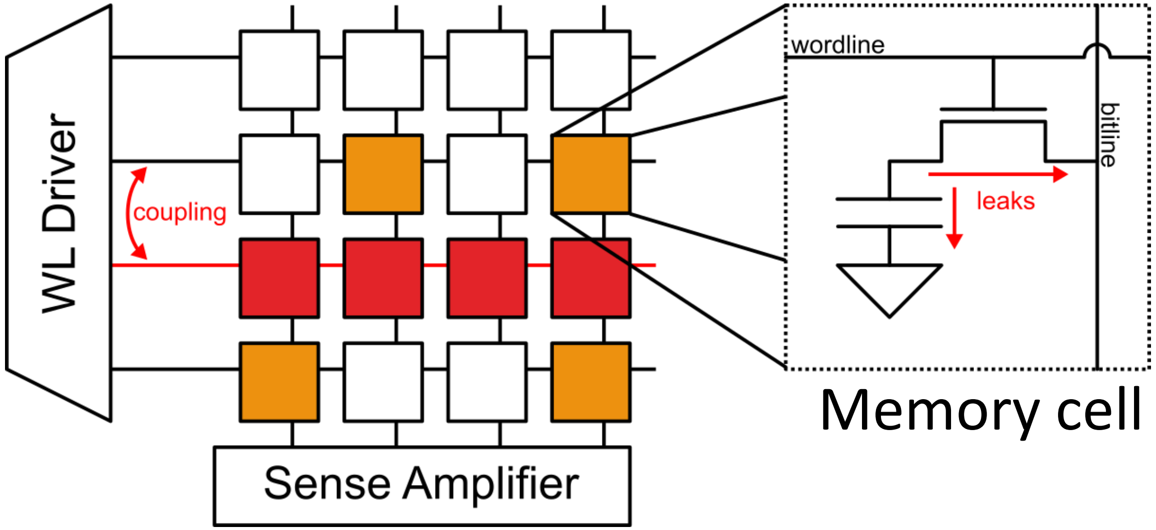
DRAM architecture



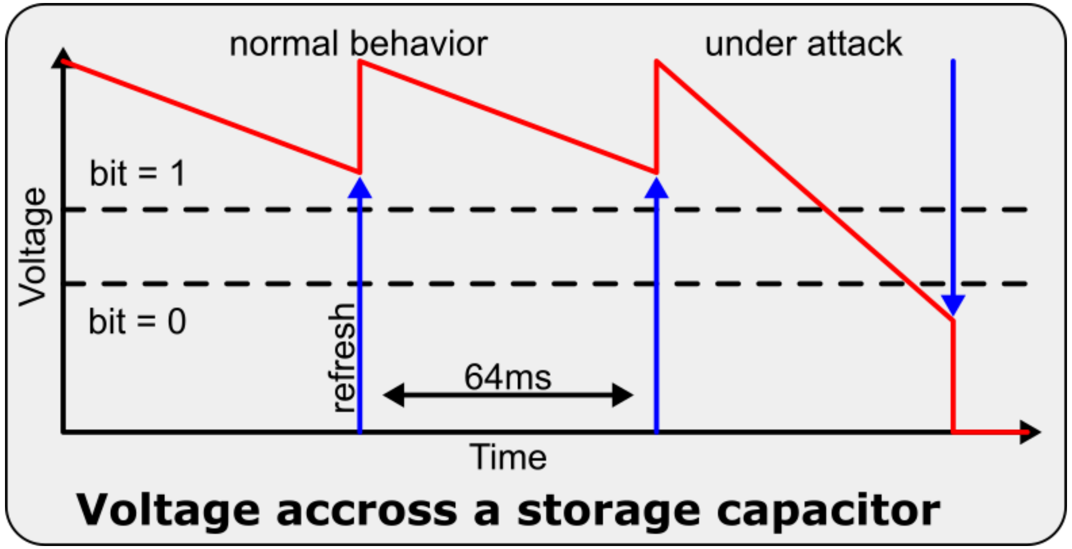


Voltage accross a storage capacitor

Rowhammer

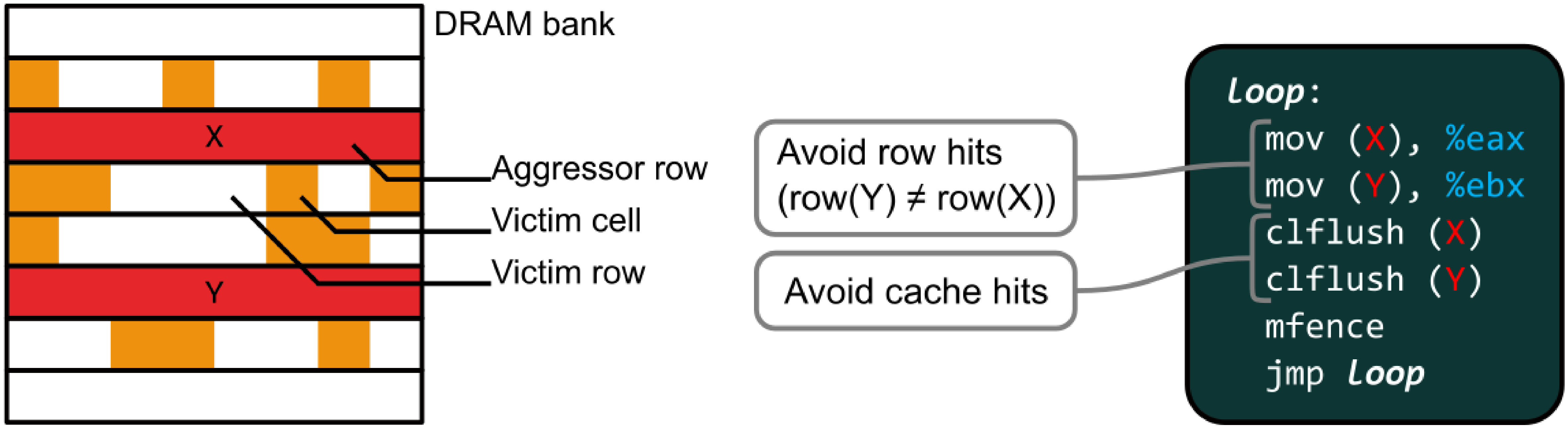


WL Driver
coupling
Sense Amplifier
wordline
bitline
leaks
Memory cell



normal behavior
under attack
bit = 1
bit = 0
Voltage
refresh
64ms
Time
Voltage accross a storage capacitor

Rowhammer



Kim, Yoongu, et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors." *ISCA*, 2014

Rowhammer objectives

Rowhammer

Crash
[1]

Sensitive information retrieval
[2]

Privilege escalation
[3][4][5]

[1] Lipp, Moritz, et al. "Nethammer: Inducing rowhammer faults through network requests." *EuroS&PW,* 2020.
[2] Kwong, Andrew, et al. "Rambleed: Reading bits in memory without accessing them." *SP*, 2020.
[3] Seaborn, Mark, and Thomas Dullien. "Exploiting the DRAM rowhammer bug to gain kernel privileges." *Black Hat*, 2015.
[4] Gruss, Daniel, Clémentine Maurice, and Stefan Mangard. "Rowhammer. js: A remote software-induced fault attack in javascript." *DIMVA*, 2016.
[5] de Ridder, Finn, et al. "SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript." *USENIX Security,* 2021.

Rowhammer objectives

[1] Lipp, Moritz, et al. "Nethammer: Inducing rowhammer faults through network requests." *EuroS&PW,* 2020.
[2] Kwong, Andrew, et al. "Rambleed: Reading bits in memory without accessing them." *SP*, 2020.
[3] Seaborn, Mark, and Thomas Dullien. "Exploiting the DRAM rowhammer bug to gain kernel privileges." *Black Hat*, 2015.
[4] Gruss, Daniel, Clémentine Maurice, and Stefan Mangard. "Rowhammer. js: A remote software-induced fault attack in javascript." *DIMVA*, 2016.
[5] de Ridder, Finn, et al. "SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript." *USENIX Security,* 2021.

Rowhammer mitigation

In software:

- Isolate sensitive data from unsafe programs in the memory [5]
- Read performance counters to detect attack signatures and stop processes [6]

Highly modular,
High performance cost

In hardware:

- Randomly refresh neighbors of activated rows [1]
- Detect most activated rows with counters [2][3][4]

Low performance cost,
Limited modularity,
Silicon area overhead

[1] Kim, Yoongu, et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors." *ISCA*, 2014.
[2] Park, Yeonhong, et al. "Graphene: Strong yet lightweight row hammer protection." *MICRO*, 2020.
[3] Yağlikçi, A. Giray, et al. "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows." *HPCA*, 2021.
[4] Lee, Eojin, et al. "TWiCe: Time window counter based row refresh to prevent row-hammering." *CAL*, 2017.
[5] Konoth, Radhesh Krishnan, et al. "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks.« *OSDI,* 2018.
[6] Alam, Manaar, et al. "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks." *Cryptology ePrint Archive* (2017).

Rowhammer mitigation

In software:

- Isolate sensitive data from unsafe programs in the memory [5]
- Read performance counters to detect attack signatures and stop processes [6]

Highly modular,
High performance cost

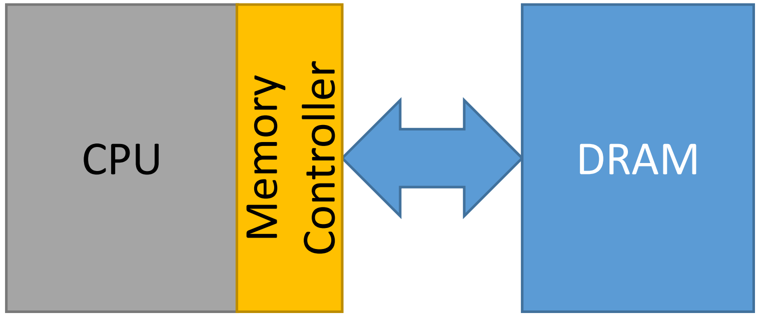
In hardware:

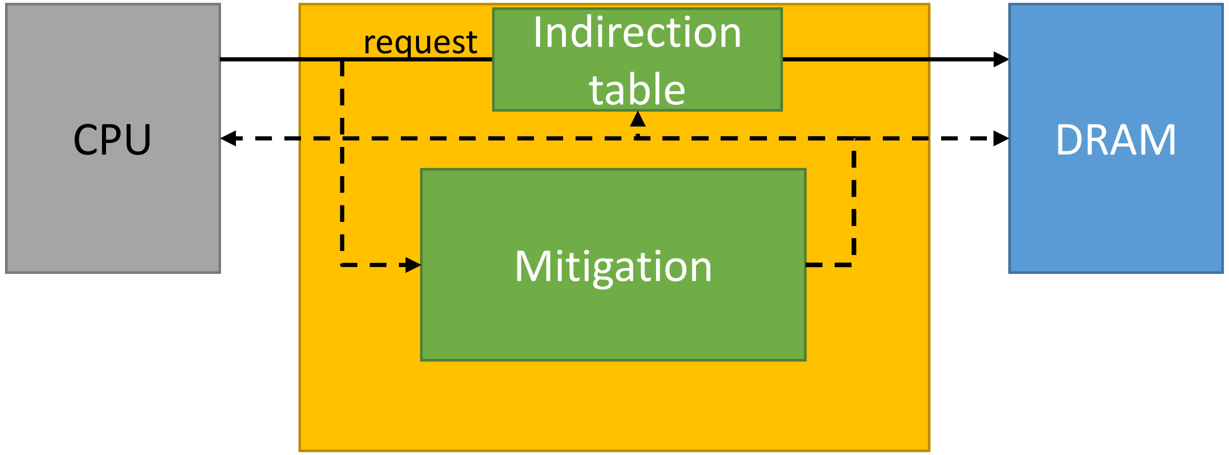
- Randomly refresh neighbors of activated rows [1]
- Detect most activated rows with counters [2][3][4]

Low performance cost,
Limited modularity,
Silicon area overhead

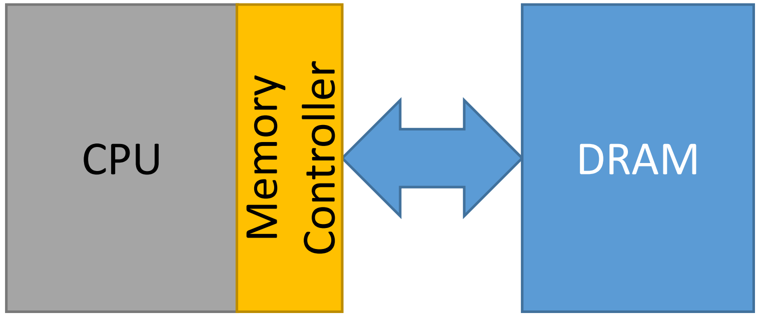
[1] Kim, Yoongu, et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors." *ISCA*, 2014.
[2] Park, Yeonhong, et al. "**Graphene**: Strong yet lightweight row hammer protection." *MICRO*, 2020.
[3] Yağlikçi, A. Giray, et al. "**Blockhammer**: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows." *HPCA*, 2021.
[4] Lee, Eojin, et al. "TWiCe: Time window counter based row refresh to prevent row-hammering." *CAL*, 2017.
[5] Konoth, Radhesh Krishnan, et al. "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks.« *OSDI,* 2018.
[6] Alam, Manaar, et al. "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks." *Cryptology ePrint Archive* (2017).

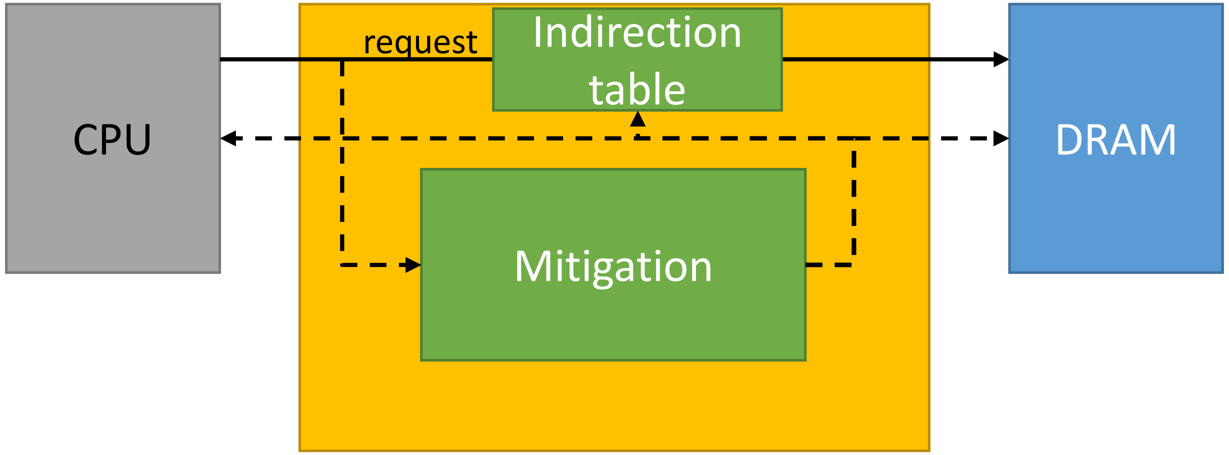
Hardware mitigations principle

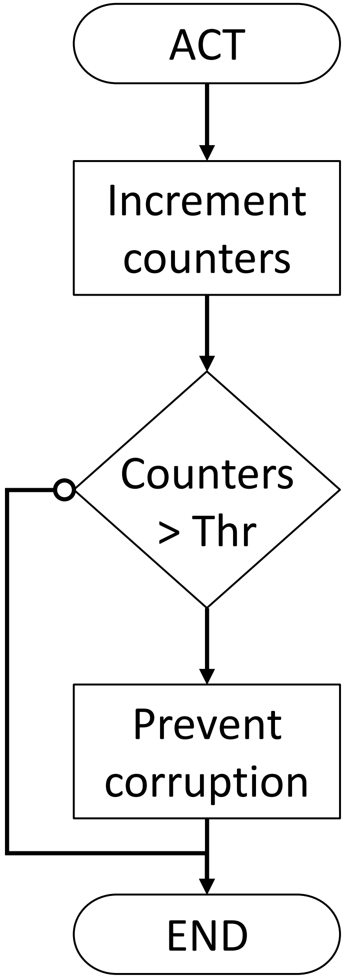




Counter-based hardware mitigations principle







1 counter / row ?

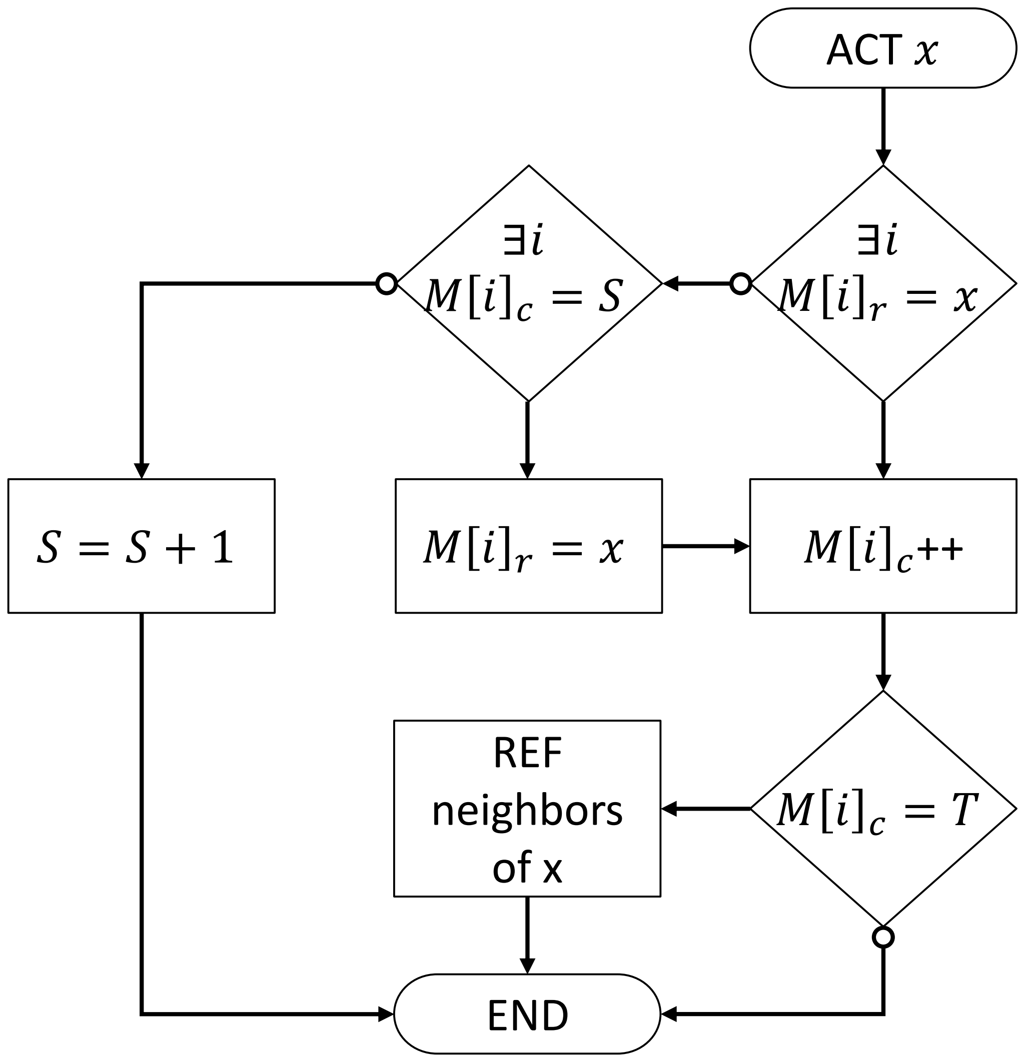
⟹ 64K counters / bank (64K rows / bank)

⟹ 1M counters / rank (16 banks / rank)

Graphene

Misra-Gries Algorithm:

Row	Count
0x1010	5
0x2020	7
0x3030	3
Spillover	**2**



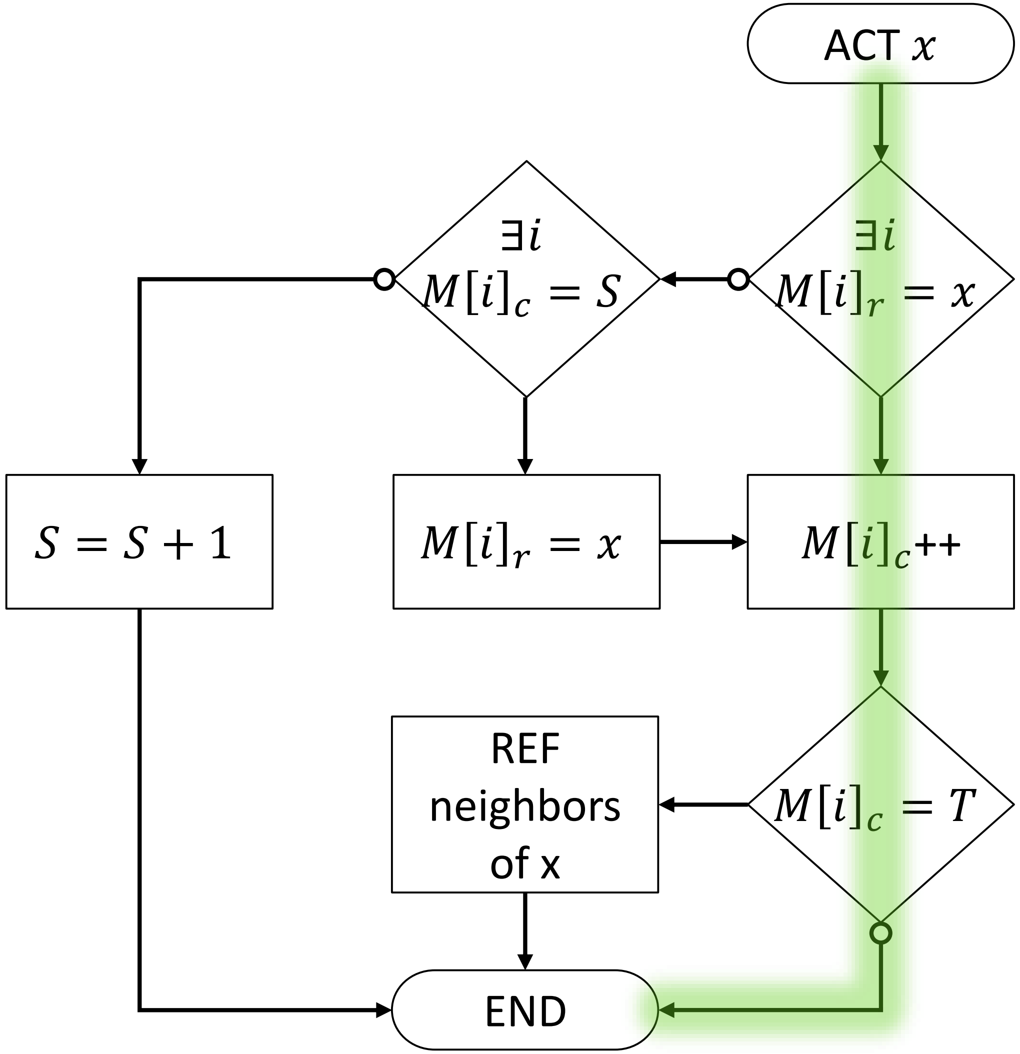
Graphene

Misra-Gries Algorithm:

Row	Count
0x1010	5
0x2020	7
0x3030	3
Spillover	**2**

0x1010

Row	Count
0x1010	6
0x2020	7
0x3030	3
Spillover	**2**



Graphene

Misra-Gries Algorithm:

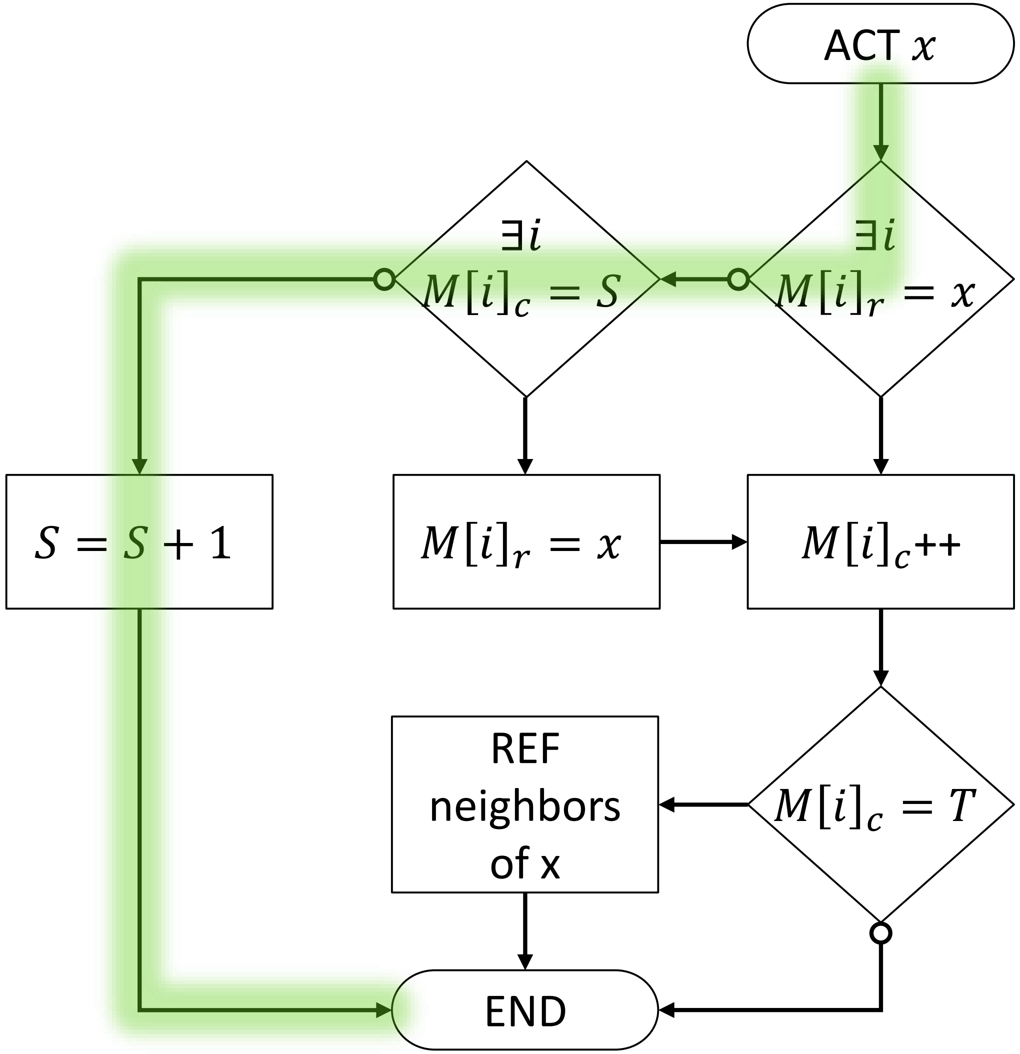
Row	Count
0x1010	5
0x2020	7
0x3030	3
Spillover	**2**

0x1010

Row	Count
0x1010	6
0x2020	7
0x3030	3
Spillover	**2**

0x4040

Row	Count
0x1010	6
0x2020	7
0x3030	3
Spillover	**3**



Graphene

Misra-Gries Algorithm:

Row	Count
0x1010	5
0x2020	7
0x3030	3
Spillover	**2**

0x1010

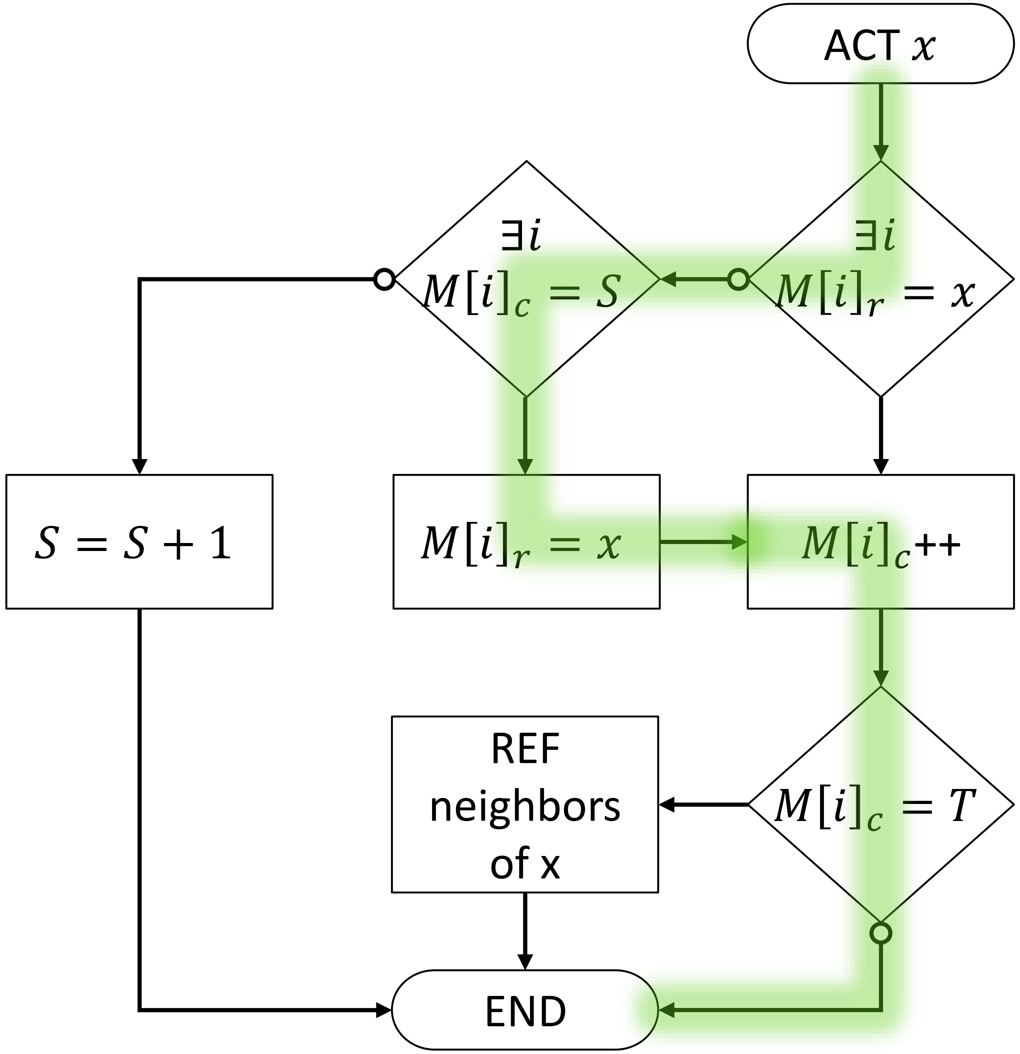
Row	Count
0x1010	6
0x2020	7
0x3030	3
Spillover	**2**

0x4040

Row	Count
0x1010	6
0x2020	7
0x3030	3
Spillover	**3**

0x5050

Row	Count
0x1010	6
0x2020	7
0x5050	4
Spillover	**3**



Graphene

Misra-Gries Algorithm:

Row	Count
0x1010	5
0x2020	7
0x3030	3
Spillover	**2**

N_{entry}

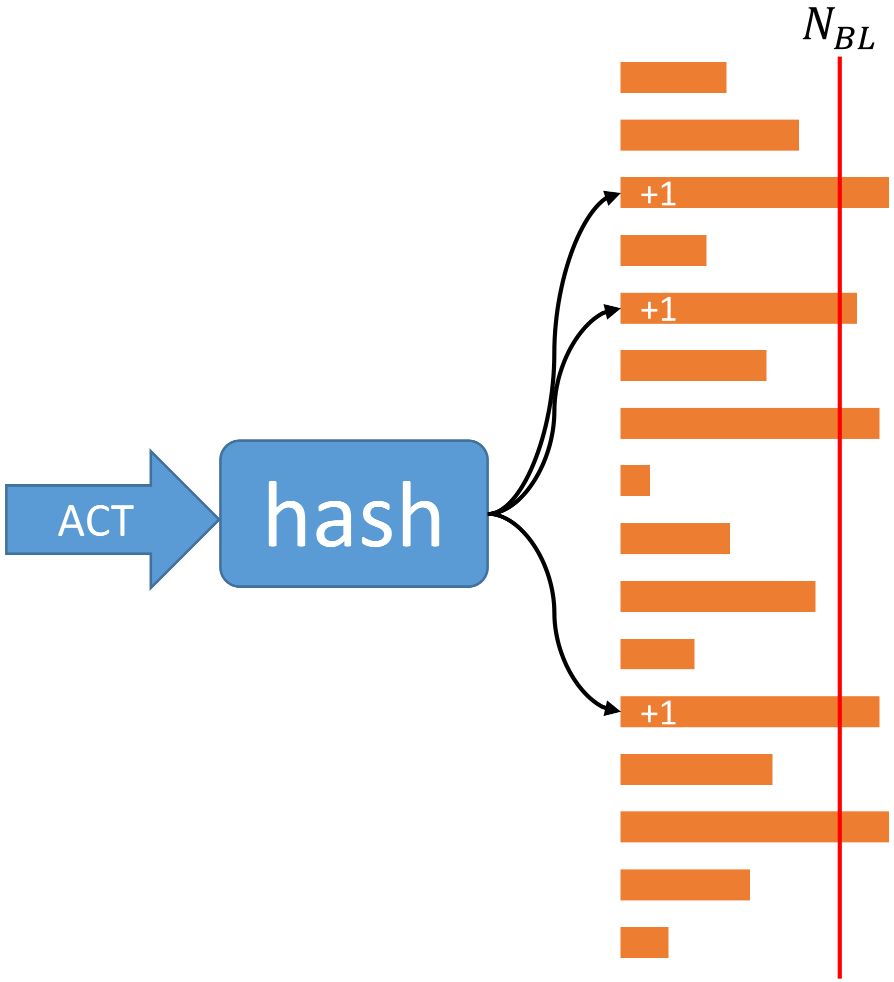
$$N_{entry} = \left\lfloor \frac{W}{T_{RH} \div 4} \right\rfloor$$

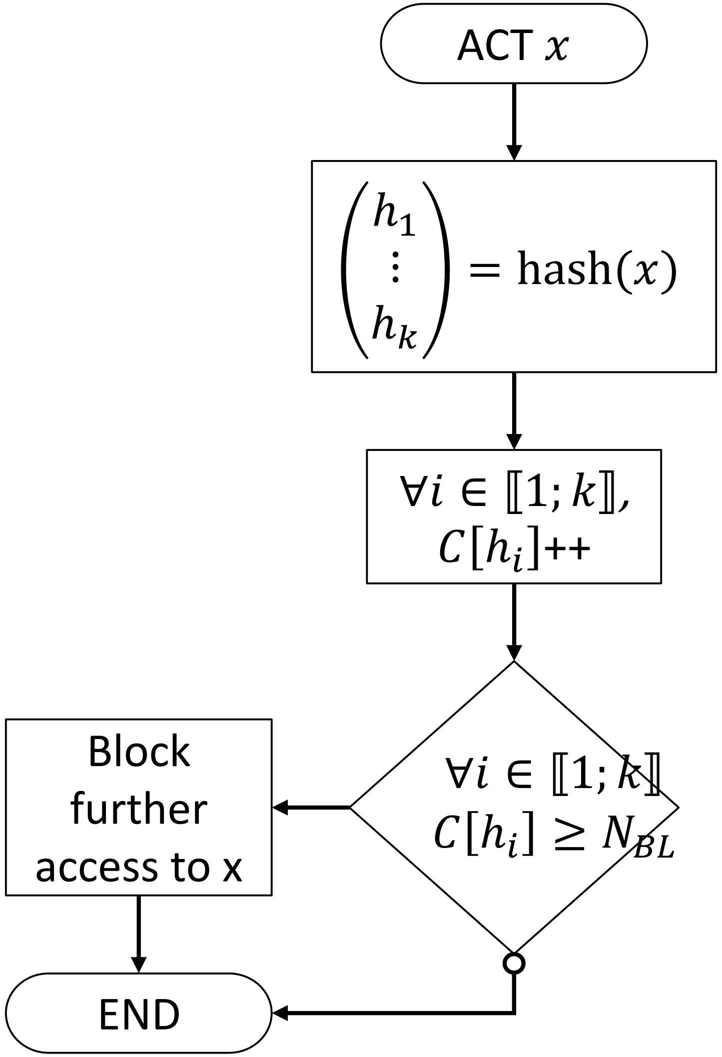
W: maximum number of ACT during t_{REFW};

T_{RH}: Rowhammer corruption threshold.

BlockHammer

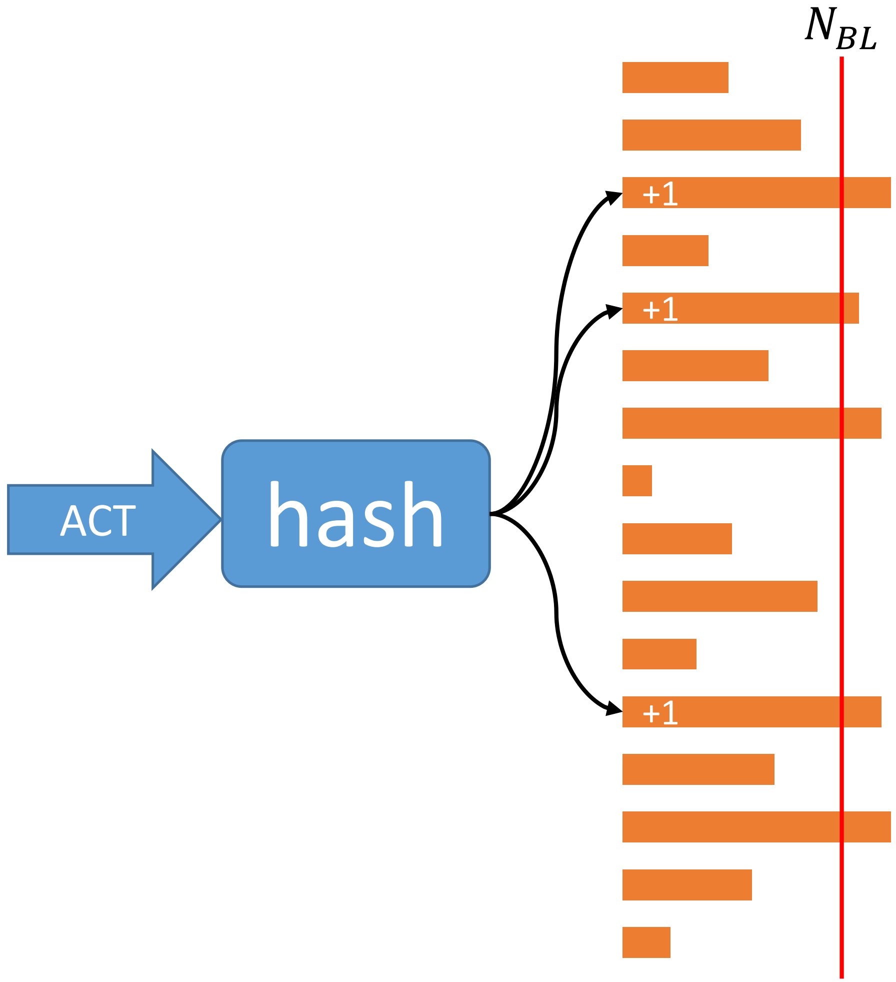
Counting Bloom Filter (CBF):





BlockHammer

Counting Bloom Filter (CBF):



$$P_{FP} = \left(1 - \sum_{l<N_{BL}} \binom{kW}{l} \left(\frac{1}{m}\right)^{l} \left(1 - \frac{1}{m}\right)^{kW-l}\right)^{k}$$

$$P_{FP} \propto \frac{W}{m} \Rightarrow m \propto \frac{W}{P_{FP}} \ (k \text{ const.})$$

W: maximum number of ACT during t_{REFW};
N_{BL}: Rowhammer detection threshold;
k: number of hash functions
m: number of counters;

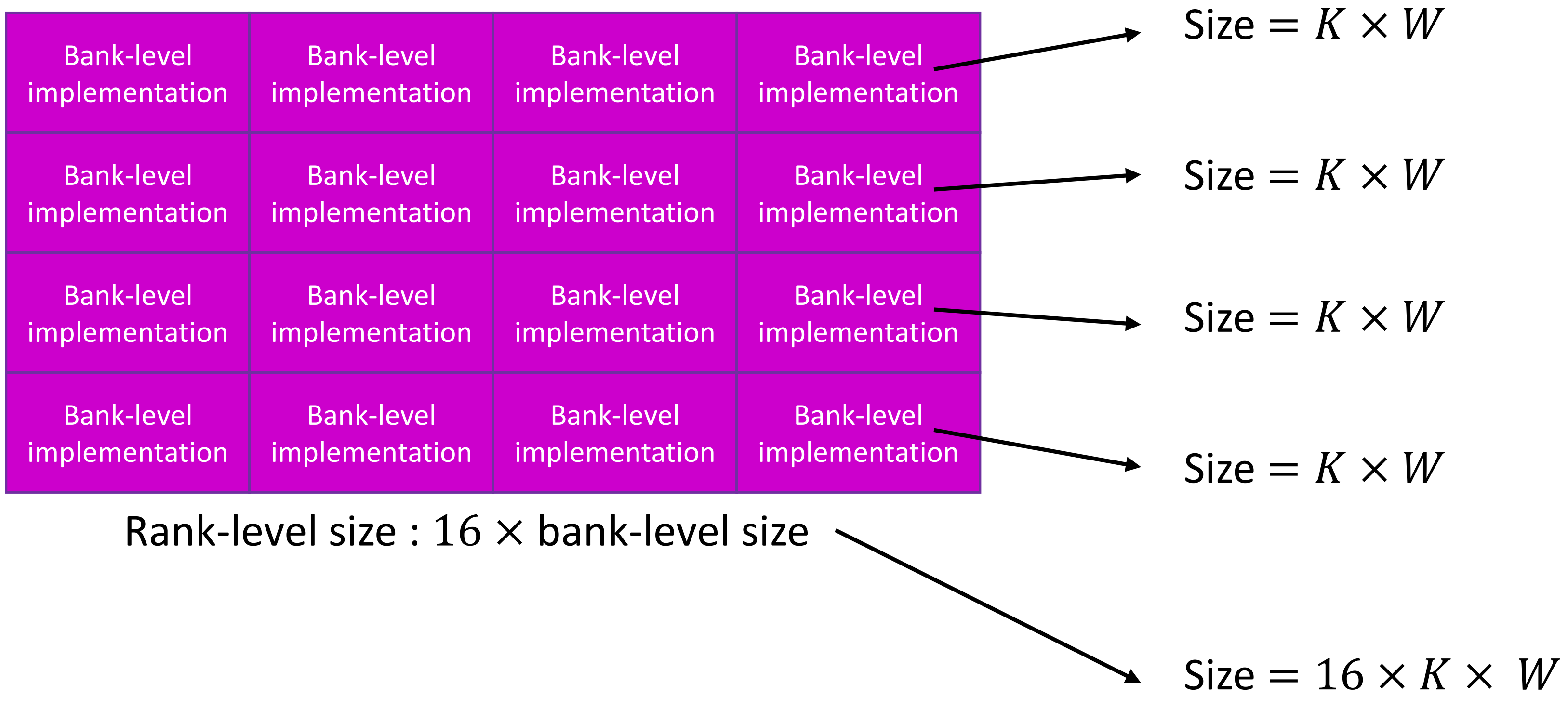
How many counters ?

Graphene and BlockHammer: Bank-level implementation, size $= K \times W$

Bank-level implementation $\rightarrow$ Size $= K \times W$

How many counters ?

Graphene and BlockHammer: Bank-level implementation, size $= K \times W$

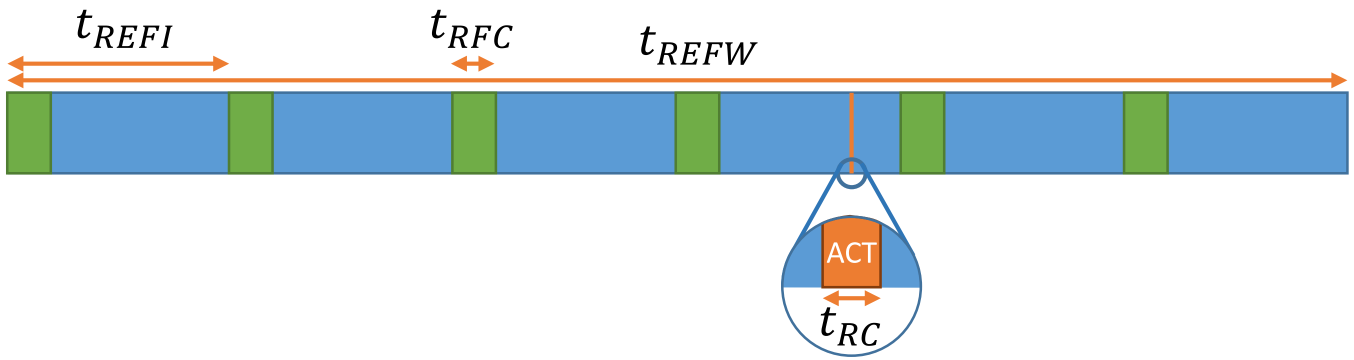


W: maximum number of ACTs during t_{REFW} at bank-level

W_R: maximum number of ACTs during t_{REFW} at rank-level $= 16 \times W$?

Memory bandwidth at different levels

t_{RC}	Same-bank ACT interval	45.8ns
t_{REFW}	Refresh cycle duration	64ms
t_{REFI}	Refresh interval	7.8µs
t_{RFC}	Refresh command duration	350ns

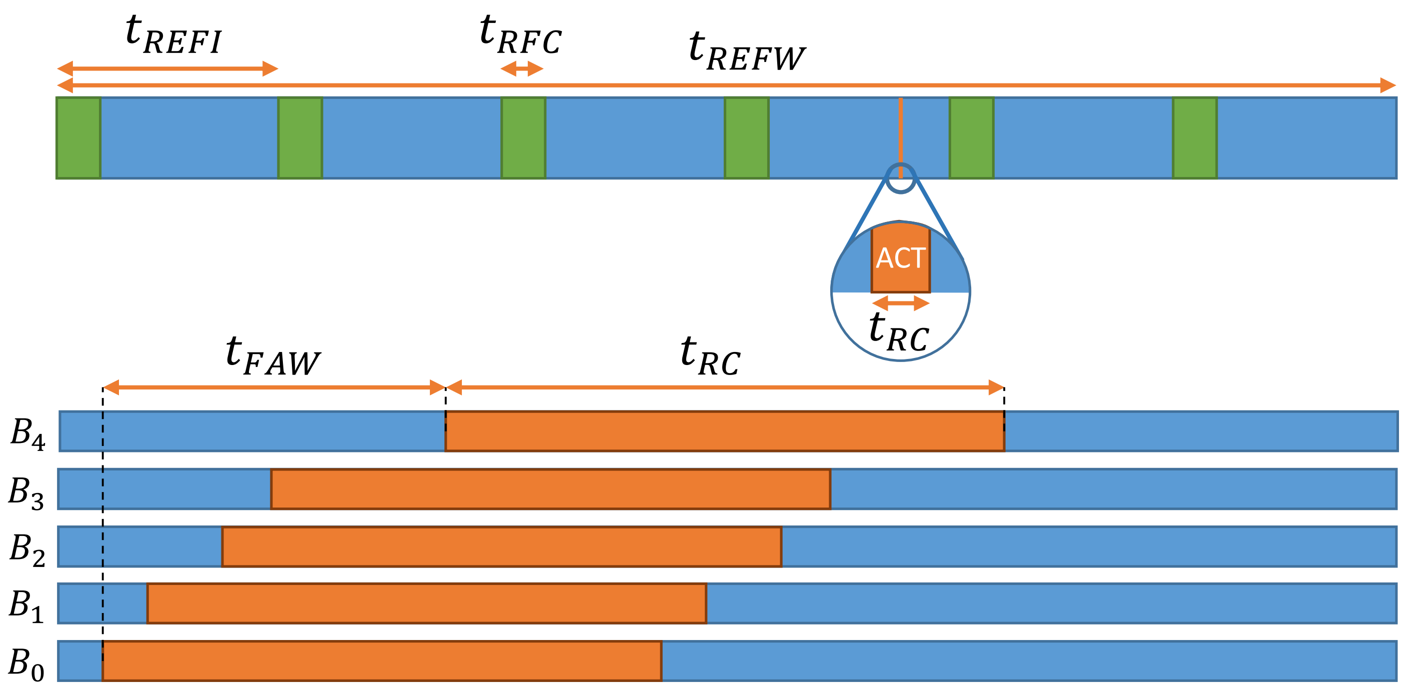


Bank level:

$$W = \left\lfloor \frac{t_{REFW}\left(1 - \frac{t_{RFC}}{t_{REFI}}\right)}{t_{RC}} \right\rfloor \approx 1{,}33\text{M}$$

Memory bandwidth at different levels

t_{RC}	Same-bank ACT interval	45.8ns
t_{REFW}	Refresh cycle duration	64ms
t_{REFI}	Refresh interval	7.8µs
t_{RFC}	Refresh command duration	350ns
t_{FAW}	Four-activate window	21.67ns



$B_0 - B_4$ are 5 banks of a single rank

Bank level:

$$W = \left\lfloor \frac{t_{REFW}\left(1 - \frac{t_{RFC}}{t_{REFI}}\right)}{t_{RC}} \right\rfloor \approx 1{,}33\mathrm{M}$$

Rank level:

$$W_R = \left\lfloor \frac{t_{REFW}\left(1 - \frac{t_{RFC}}{t_{REFI}}\right)}{t_{FAW} \div 4} \right\rfloor \approx 11{,}3\mathrm{M} \neq 16 \times W$$

Reduction in considered ACTs

Bank-level implementation	Bank-level implementation	Bank-level implementation	Bank-level implementation
Bank-level implementation	Bank-level implementation	Bank-level implementation	Bank-level implementation
Bank-level implementation	Bank-level implementation	Bank-level implementation	Bank-level implementation
Bank-level implementation	Bank-level implementation	Bank-level implementation	Bank-level implementation

Total considered ACTs : $16 \times W = 21.28\text{M}$

Rank-level implementation

Total considered ACTs : $W_R = 11.3\text{M}$

-47% ACTs

Consequences for Graphene & BlockHammer

	Bank-level implementation	Rank-level implementation	reduction
Graphene	162 entries entry size: 30 bits*. Total size: $16 \times 162 \times 30 \text{ bits} = \mathbf{9.61KiB}$	1377 entries entry size: 34 bits** Total size: $1377 \times 162 \times 34 \text{ bits} = \mathbf{5.79KiB}$	$\mathbf{-40\%}$
BlockHammer	2048 counters 13 bits / counter Total size: $16 \times 2048 \times 13\ bits = \mathbf{52}\boldsymbol{KiB}$	16384 counters*** 13 bits / counter Total size: $16384 \times 13\ bits = \mathbf{26}\boldsymbol{KiB}$	$\mathbf{-50\%}$

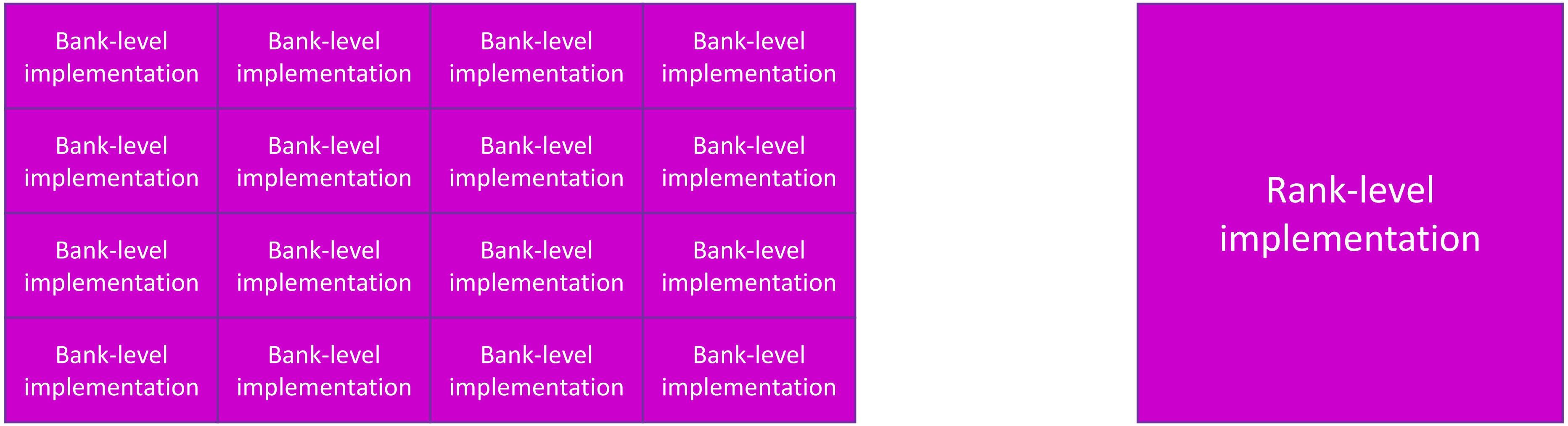
*: row address – 16 bits, counter – 13 bits, overflow bit – 1 bit

**: row address – 20 bits, counter – 13 bits, overflow bit – 1 bit

***: keeps the same P_{FP} as for bank-level implementation

Reduction in storage requirements

	Bank level	Rank level	reduction
$16 \times W \;/\; W_R$	21.28M	11.3M	-47%
Graphene	9.61KiB	5.79KiB	-40%
BlockHammer	52KiB	26KiB	-50%



Total considered ACTs : $16 \times W = 21.28\text{M}$

Total considered ACTs : $W_R = 11.3\text{M}$

$-40 - 50\%$ storage

Reduction in storage requirements – DDR5

	Bank level	Rank level	reduction
$32 \times W$ / W_R	$21.15\mathrm{M}$	$8\mathrm{M}$	-62%
Graphene	$9.38\mathrm{KiB}$	$4.05\mathrm{KiB}$	-57%
BlockHammer	$52\mathrm{KiB}$	$19.5\mathrm{KiB}$	-62.5%

(with same P_{FP} as with DDR4)

Bank-level implementation	Bank-level implementation	Bank-level implementation	Bank-level implementation
Bank-level implementation	Bank-level implementation	Bank-level implementation	Bank-level implementation
Bank-level implementation	Bank-level implementation	Bank-level implementation	Bank-level implementation
Bank-level implementation	Bank-level implementation	Bank-level implementation	Bank-level implementation

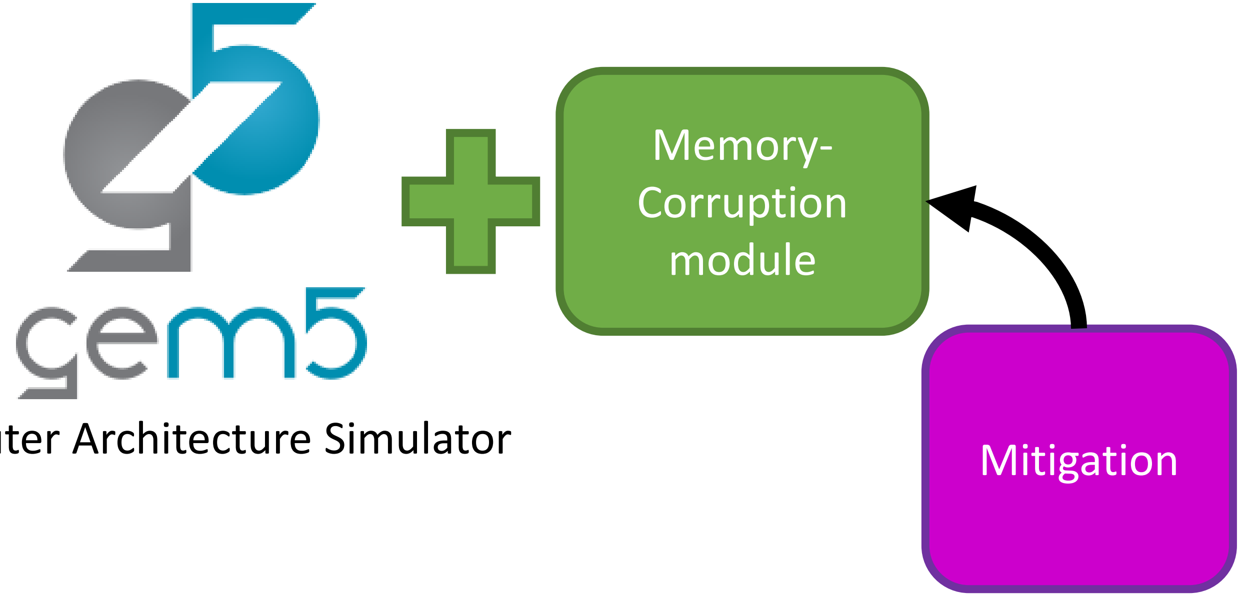
Total considered ACTs : $32 \times W = 21.15\mathrm{M}$

Rank-level implementation

Total considered ACTs : $W_R = 8\mathrm{M}$

$-57 - 62.5\%$ storage

Does it still work as it should ?



Computer Architecture Simulator

France, Loïc, et al. "Implementing Rowhammer Memory Corruption in the gem5 Simulator." 32nd International Workshop on Rapid System Prototyping (RSP). IEEE, 2021.

Thank you for your attention