



CSI:Rowhammer & PressHammer

Jonas Juffinger Lukas Lamster Sudheendra Raghav Neela Andreas Kogler
Martin Heckel Moritz Lipp Lukas Schwarz Maria Eichlseder Florian
Adamsky Daniel Gruss

2024-07-16

✉ jonas.juffinger@iaik.tugraz.at

🐦 [NotImaginary_](#)

🌐 www.jonasjuffinger.com

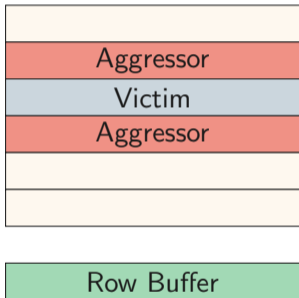
IAIK, Graz University of Technology, Austria





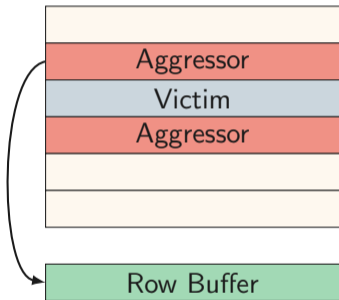
- Jonas Juffinger
- PhD Candidate
- Graz University of Technology, IAIK
- Rowhammer
- Side channels
- Microarchitectural attacks

Introduction



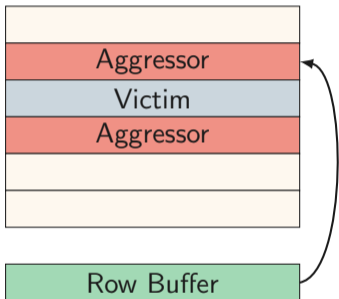
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]



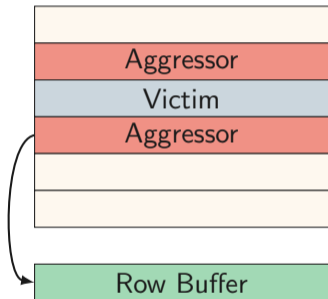
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]



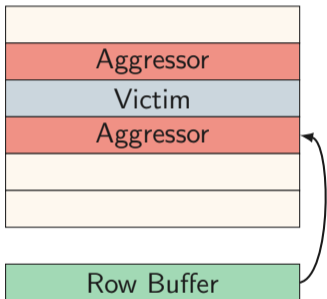
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]



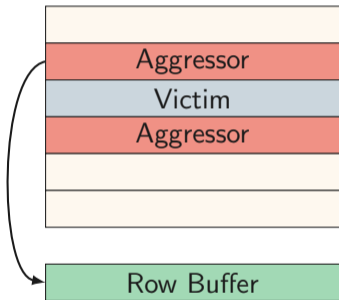
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]



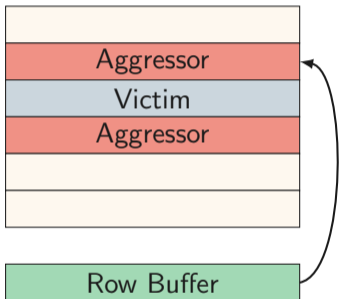
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]



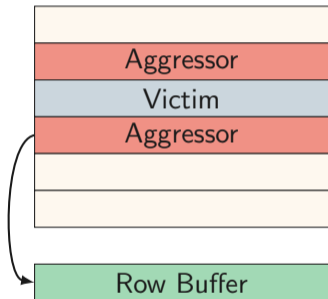
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]



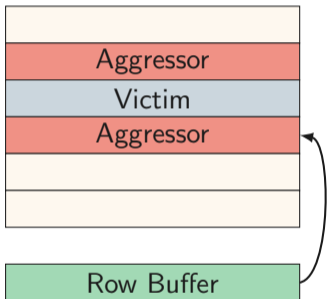
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]



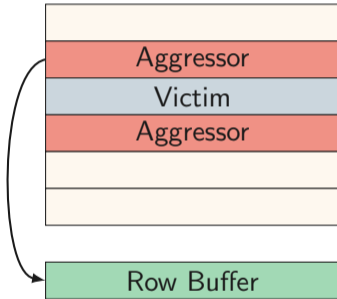
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]



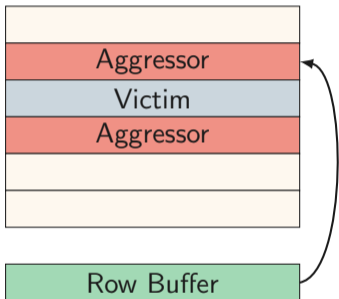
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]



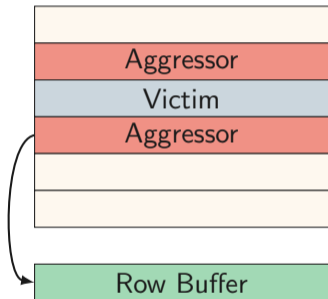
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]



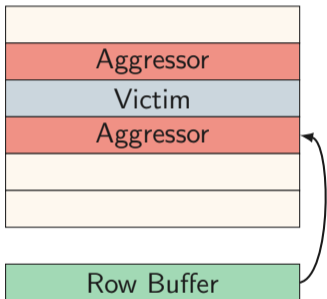
Repeatedly and **quickly** active multiple rows.

Rowhammer [3]

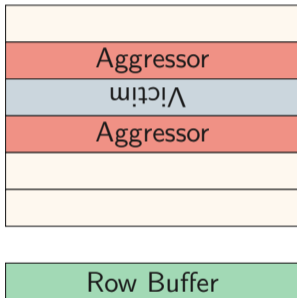


Repeatedly and **quickly** active multiple rows.

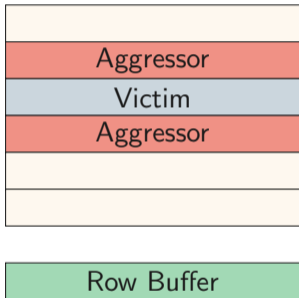
Rowhammer [3]



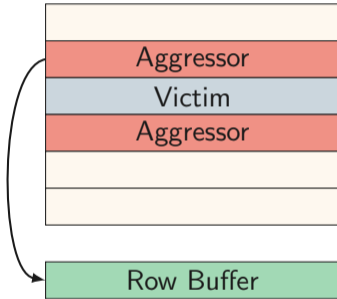
Repeatedly and **quickly** active multiple rows.



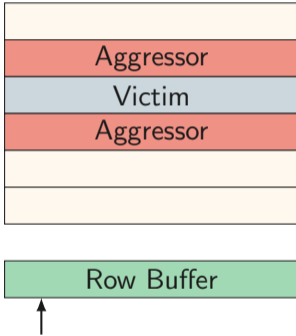
Repeatedly and **quickly** active multiple rows.



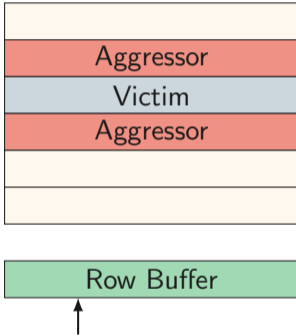
Keep rows open as **long** as possible.



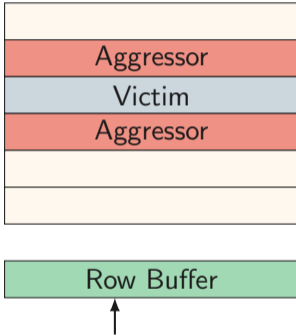
Keep rows open as **long** as possible.



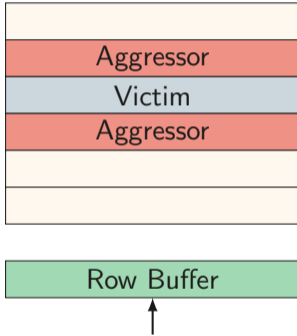
Keep rows open as **long** as possible.



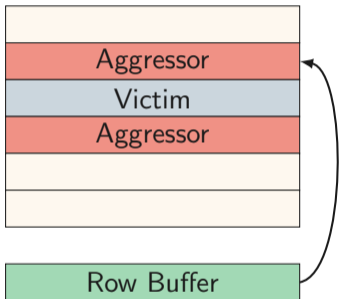
Keep rows open as **long** as possible.



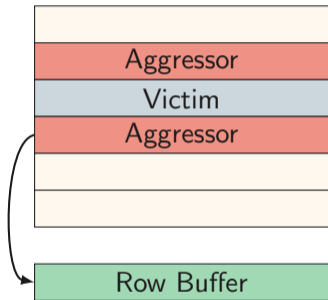
Keep rows open as **long** as possible.



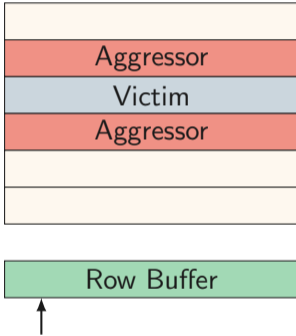
Keep rows open as **long** as possible.



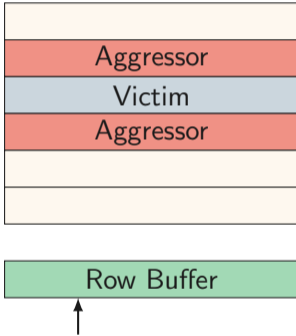
Keep rows open as **long** as possible.



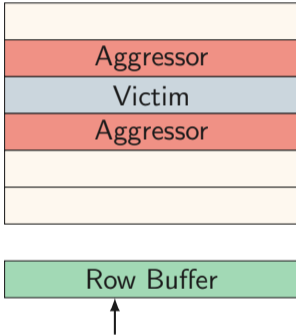
Keep rows open as **long** as possible.



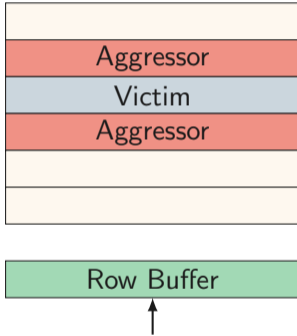
Keep rows open as **long** as possible.



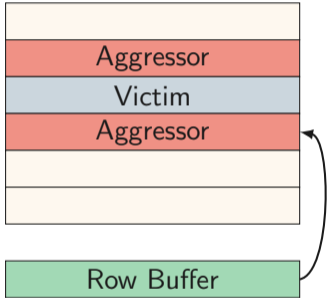
Keep rows open as **long** as possible.



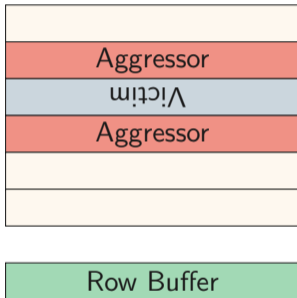
Keep rows open as **long** as possible.



Keep rows open as **long** as possible.

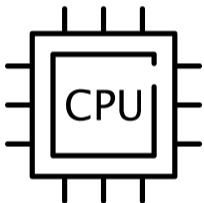


Keep rows open as **long** as possible.

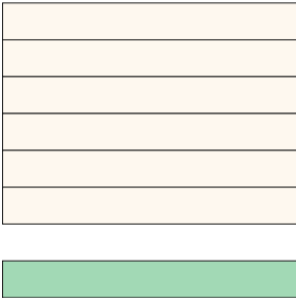


Keep rows open as **long** as possible.

Flips different bits than Rowhammer

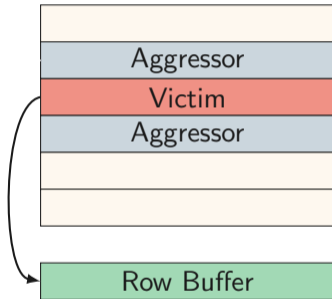


- The CPU's memory controller adheres to the DRAM standard's timing limits.
- A row must not be open longer than $9 \times t_{REFI}$.
- $t_{REFI} = 7.8 \mu\text{s}$



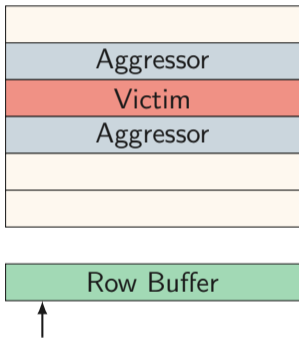
Keep **one** row open as long as possible.

Single-Sided RowPress



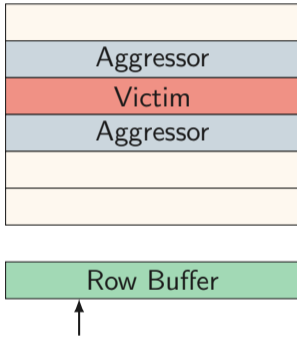
Keep **one** row open as long as possible.

Single-Sided RowPress



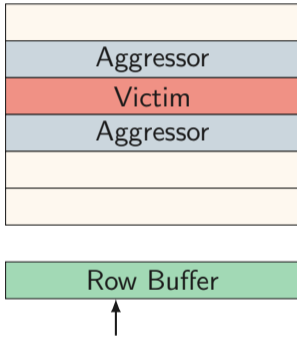
Keep **one** row open as long as possible.

Single-Sided RowPress



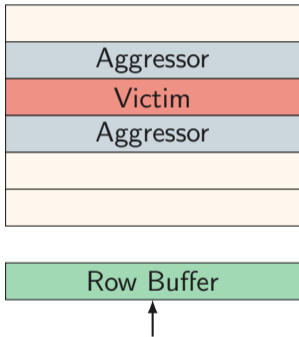
Keep **one** row open as long as possible.

Single-Sided RowPress



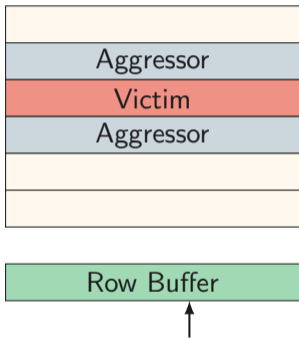
Keep **one** row open as long as possible.

Single-Sided RowPress



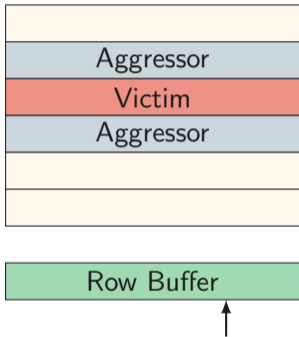
Keep **one** row open as long as possible.

Single-Sided RowPress



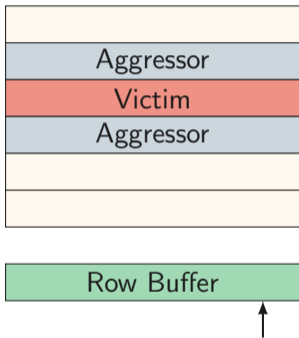
Keep **one** row open as long as possible.

Single-Sided RowPress

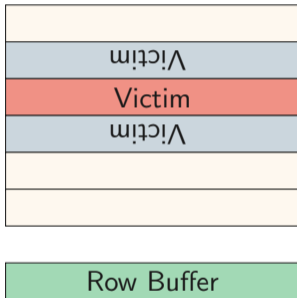


Keep **one** row open as long as possible.

Single-Sided RowPress



Keep **one** row open as long as possible.



Keep **one** row open as long as possible.



But wait...



There is this paper from 2018.

Another Flip in the Wall of Rowhammer Defenses

Daniel Gruss¹, Moritz Lipp¹, Michael Schwarz¹, Daniel Genkin²,
Jonas Juffinger¹, Sioli O'Connell³, Wolfgang Schoechl¹, and Yuval Yarom^{3,4}

¹ Graz University of Technology

² University of Pennsylvania and University of Maryland

³ University of Adelaide

⁴ Data61

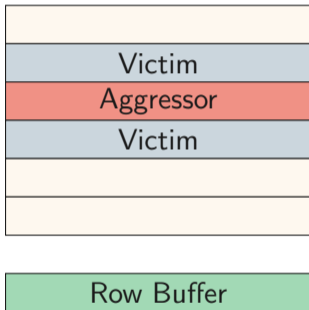
Abstract—The Rowhammer bug allows unauthorized modification of bits in DRAM cells from unprivileged software, enabling powerful privilege-escalation attacks. Sophisticated Rowhammer countermeasures have been presented, aiming at mitigating the Rowhammer bug or its exploitation. However, the state of the art provides insufficient insight on the completeness of these defenses.

In this paper, we present novel Rowhammer attack and exploitation primitives, showing that even a combination of all defenses is ineffective. Our new attack technique, *one-location hammering*, breaks previous assumptions on requirements for triggering the Rowhammer bug, i.e., we do not hammer multiple DRAM rows but only keep one DRAM row constantly open. Our new exploitation technique, *opcode flipping*, bypasses recent isolation mechanisms by flipping bits in a predictable and targeted way in userspace binaries. We replace conspicuous

Software-based mitigations, which can be implemented on commodity systems, have also been proposed. These include ad-hoc defense techniques such as doubling the RAM refresh rates [44], removing unprivileged access to the pagemap interface [45, 62, 65], and prohibiting the `clflush` instruction [65]. However, recent works have already bypassed these countermeasures [6, 24, 68]. Other ad-hoc attempts, such as disabling page deduplication by default [52, 60], only prevent specific Rowhammer attacks exploiting these features [11, 59], but not all Rowhammer attacks.

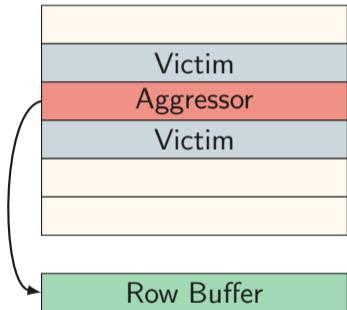
The research community proposed sophisticated defenses which seemingly have solved the Rowhammer problem. Based on the underlying primitives of these defenses we introduce

One-Location Rowhammer [2]



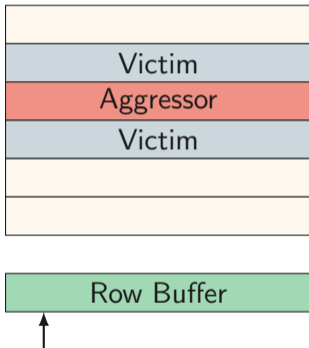
Repeatedly and **quickly** activate **one** row.

One-Location Rowhammer [2]



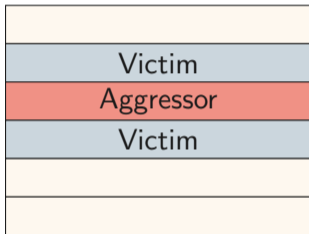
Repeatedly and **quickly** activate **one** row.

One-Location Rowhammer [2]



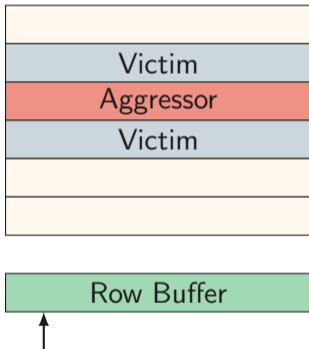
Repeatedly and **quickly** activate **one** row.

One-Location Rowhammer [2]



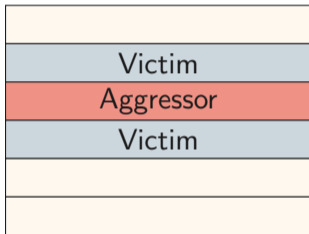
Repeatedly and **quickly** activate **one** row.

One-Location Rowhammer [2]



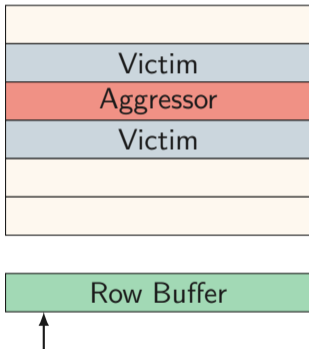
Repeatedly and **quickly** activate **one** row.

One-Location Rowhammer [2]



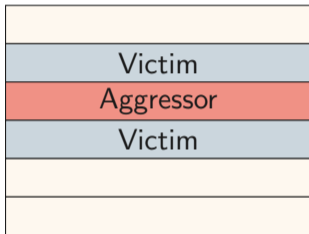
Repeatedly and **quickly** activate **one** row.

One-Location Rowhammer [2]



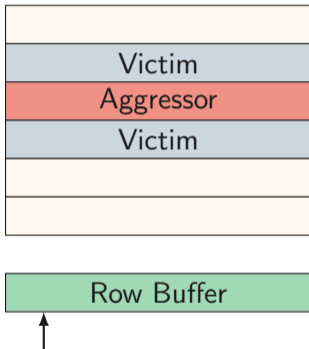
Repeatedly and **quickly** activate **one** row.

One-Location Rowhammer [2]



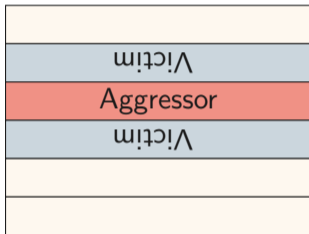
Repeatedly and **quickly** activate **one** row.

One-Location Rowhammer [2]



Repeatedly and **quickly** activate **one** row.

One-Location Rowhammer [2]



Repeatedly and **quickly** activate **one** row.

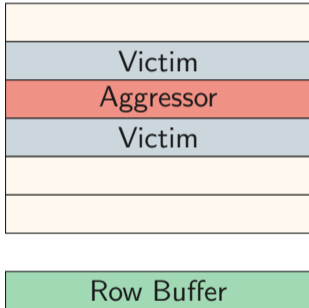


LOOKS A LOT LIKE ROWPRESS



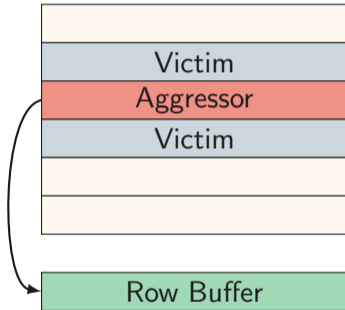
It's the memory controller.

One-Location Rowhammer - Hypothesis



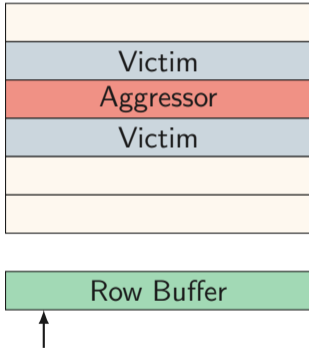
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



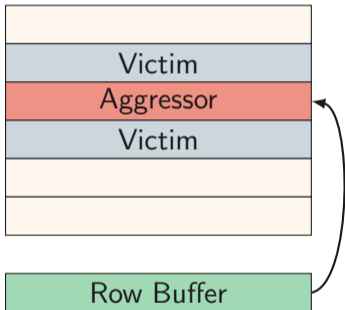
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



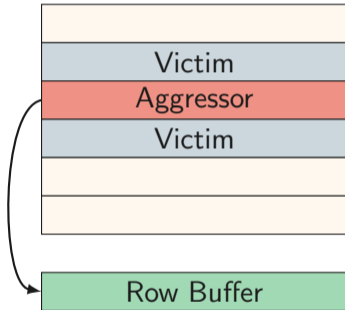
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



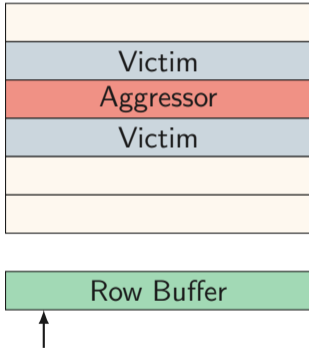
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



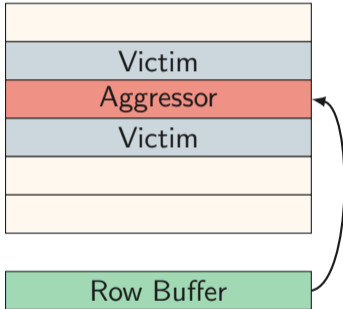
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



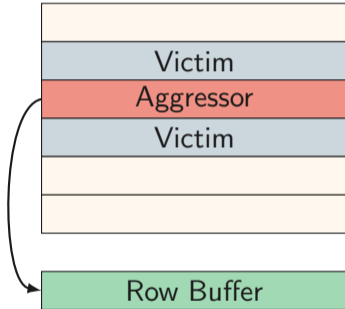
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



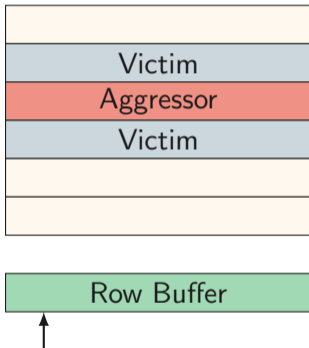
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



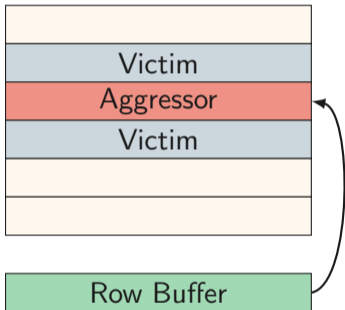
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



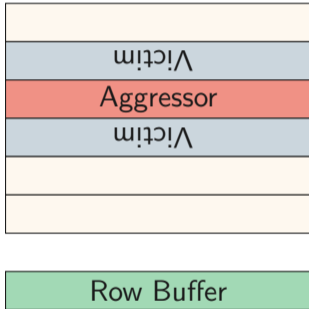
Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



Memory controller employs a *closed-row* policy.

One-Location Rowhammer - Hypothesis



Memory controller employs a *closed-row* policy.

Case closed ✓

Is it?

Is Rowpress = OL Rowhammer?

Is Rowpress = OL Rowhammer?

or where Gruss et al. wrong all along?

Presshammer

Rowhammer and Rowpress without Physical Address Information

Jonas Juffinger Sudheendra Raghav Neela Martin Heckel Lukas Schwarz
Florian Adamsky Daniel Gruss

2024-07-16

IAIK, Graz University of Technology, Austria



RH and RP are caused by different physical effects

RH and RP are caused by different physical effects



Different weak cells i.e. bit flip locations

RH and RP are caused by different physical effects

Different weak cells i.e. bit flip locations

Bit flip locations of OL RH = Rowpress?

RH and RP are caused by different physical effects

Different weak cells i.e. bit flip locations

Bit flip locations of OL RH = Rowpress?

YES

RH and RP are caused by different physical effects

Different weak cells i.e. bit flip locations

Bit flip locations of OL RH = Rowpress?

YES

OL RH = Rowpress

RH and RP are caused by different physical effects

Different weak cells i.e. bit flip locations

Bit flip locations of OL RH = Rowpress?

YES

NO

OL RH = Rowpress

RH and RP are caused by different physical effects

Different weak cells i.e. bit flip locations

Bit flip locations of OL RH = Rowpress?

YES

OL RH = Rowpress

NO

OL RH \neq Rowpress

		RH		RP	
		DS	OL	DS	OL
RH	DS				
	OL				
RP	DS				
	OL				

Results

	DIMM	Size	MT/s	Double Sided		One Location	
				RH*	RP	RH	RP
DDR4	D1	8 GB	2133	✓	✓	✓	✓
	D2	8 GB	2133	✓	✓	✓	✓
	D3	8 GB	2400	✓	✗	~	~
	D4	8 GB	2400	✓	✗	~	~
	D5	8 GB	2133	✗	✗	~	~
	D6	8 GB	2666	✓	✗	~	~
	D7	8 GB	2666	✗	✗	~	~
	D8	8 GB	2666	✗	✗	~	~
	D9	8 GB	2666	✓	✗	~	~
	D10	8 GB	2666	✗	✗	~	~
	D11	8 GB	3000	✗	✗	~	~
	D12	8 GB	2133	✗	✗	~	~

Number of Bit Flips

DIMM	Double Sided			One Location		
	RH*	RP-8	RP-16	RH	RP-8	RP-16
D1 & D2	15 612	2174	4264	14 264	161	0
D3	13 656	0	0	0	0	0
D4	15 786	0	0	0	0	0
D6	157	0	0	0	0	0
D9	3040	0	0	0	0	0



		RH		RP	
		DS	OL	DS	OL
RH	DS	100	56.5	13.7	1.0
	OL	61.8	100	14.5	1.1
RP	DS	98.1	95.3	100	1.0
	OL	96.3	98.8	81.4	100

PressHammer Exploit



Exploit Steps

- Find complete DRAM row



Exploit Steps

- Find complete DRAM row
- Template memory for bit flips



Exploit Steps

- Find complete DRAM row
- Template memory for bit flips
- Spray page tables



Exploit Steps

- Find complete DRAM row
- Template memory for bit flips
- Spray page tables
- Flip page tables



Exploit Steps

- Find complete DRAM row
- Template memory for bit flips
- Spray page tables
- Flip page tables
- → Profit

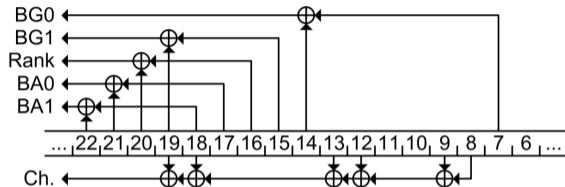
- DRAM row typically 4 kB

Find Complete DRAM Row

- DRAM row typically 4 kB
- Memory Page typically 4 kB

Find Complete DRAM Row

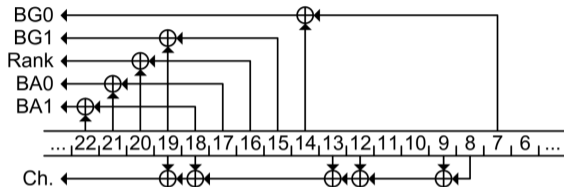
- DRAM row typically 4 kB
- Memory Page typically 4 kB
- No 1:1 mapping



Pessl et al. [6]

Find Complete DRAM Row

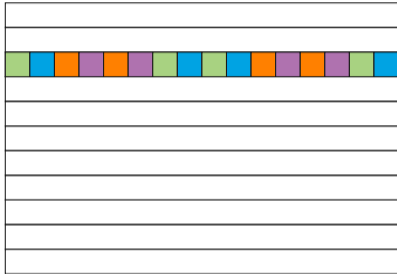
- DRAM row typically 4 kB
- Memory Page typically 4 kB
- No 1:1 mapping



Pessl et al. [6]



DRAM Row Finding



Bank 0

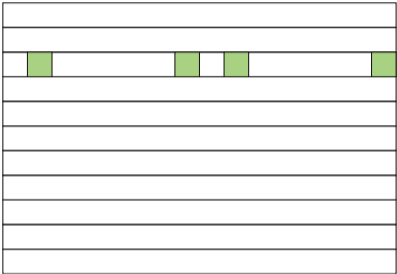


Bank 1

DRAM Row Finding

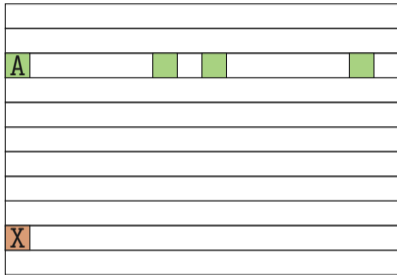


Bank 0

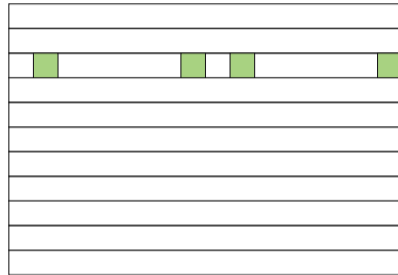


Bank 1

DRAM Row Finding



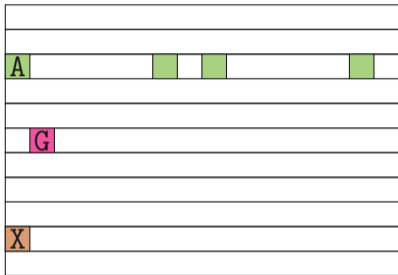
Bank 0



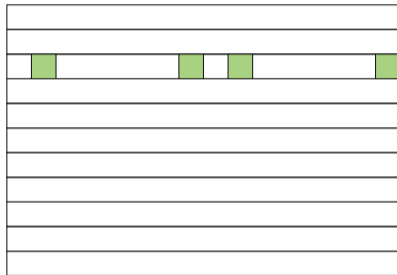
Bank 1

R1 Bank conflict with X

DRAM Row Finding



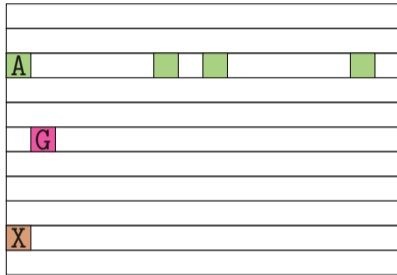
Bank 0



Bank 1

R1 Bank conflict with X

DRAM Row Finding



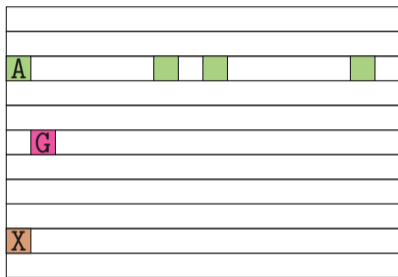
Bank 0



Bank 1

R1 Bank conflict with X ✓

DRAM Row Finding



Bank 0

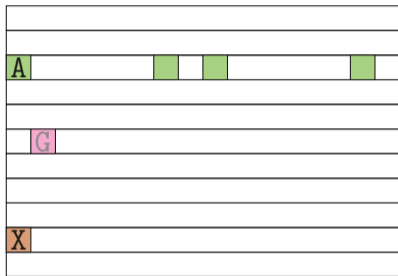


Bank 1

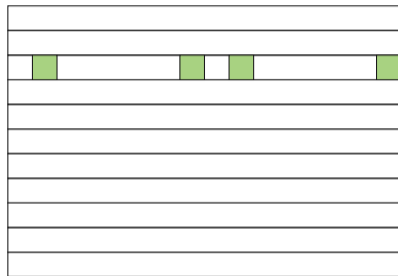
R1 Bank conflict with X ✓

R2 No bank conflict with A

DRAM Row Finding



Bank 0



Bank 1

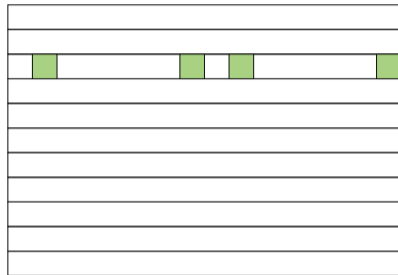
R1 Bank conflict with X ✓

R2 No bank conflict with A ✗

DRAM Row Finding



Bank 0



Bank 1

R1 Bank conflict with X

R2 No bank conflict with A

DRAM Row Finding



Bank 0

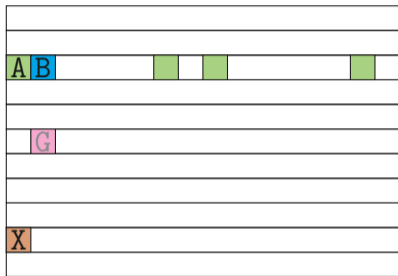


Bank 1

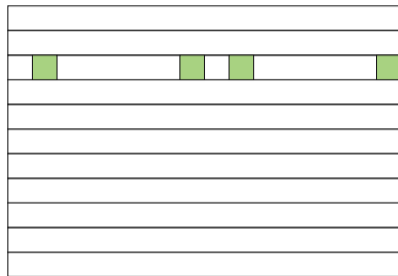
R1 Bank conflict with X ✓

R2 No bank conflict with A

DRAM Row Finding



Bank 0

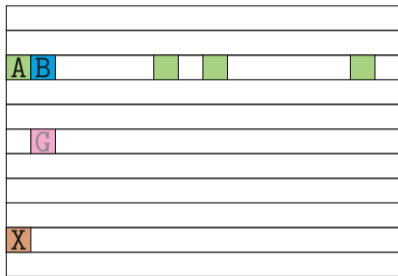


Bank 1

R1 Bank conflict with X ✓

R2 No bank conflict with A ✓

DRAM Row Finding



Bank 0

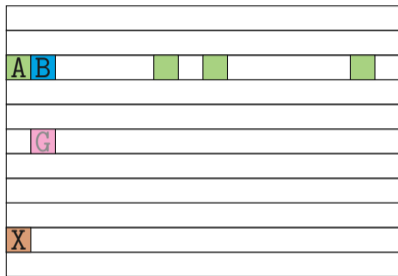


Bank 1

R1 Bank conflict with X

R2 No bank conflict with A

DRAM Row Finding



Bank 0

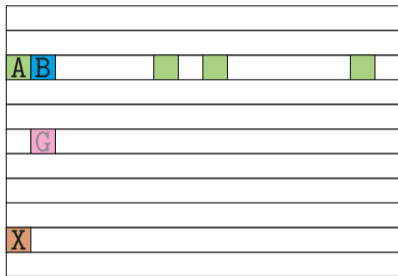


Bank 1

R1 Bank conflict with X ✗

R2 No bank conflict with A

DRAM Row Finding



Bank 0

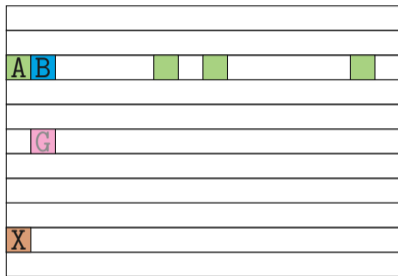


Bank 1

R1 Bank conflict with X ❌

R2 No bank conflict with A

DRAM Row Finding



Bank 0

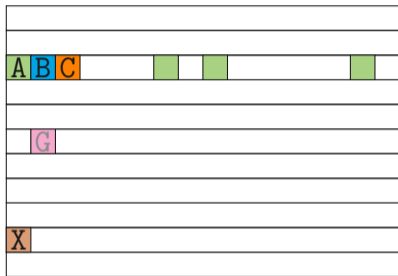


Bank 1

R1 Bank conflict with X ❌

R2 No bank conflict with A ✅

DRAM Row Finding



Bank 0

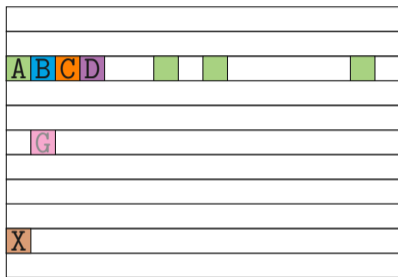


Bank 1

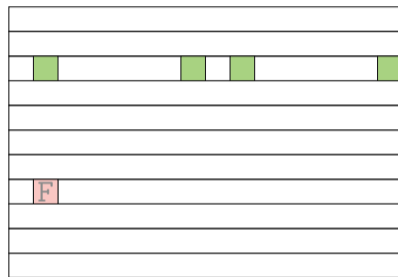
R1 Bank conflict with X ✓

R2 No bank conflict with A ✓

DRAM Row Finding



Bank 0



Bank 1

R1 Bank conflict with X ✓

R2 No bank conflict with A ✓

DRAM Row Finding Accuracy and Speed



# Meas.	Duration per Row	Found CLs	Correct CLs
250	40 ms, $\sigma = 17$	124, $\sigma = 4.9$	119, $\sigma = 14$
500	45 ms, $\sigma = 15$	128, $\sigma = 0.8$	125, $\sigma = 16$
1000	116 ms, $\sigma = 40$	125, $\sigma = 4.2$	123, $\sigma = 4.3$
2500	260 ms, $\sigma = 109$	125, $\sigma = 4.9$	124, $\sigma = 5.2$



Find weak cells

- RowPress bit flips repeatable



Find weak cells

- RowPress bit flips repeatable
- Turn off cache line prefetcher



Find weak cells

- RowPress bit flips repeatable
- ~~Turn off cache line prefetcher~~ X



Find weak cells

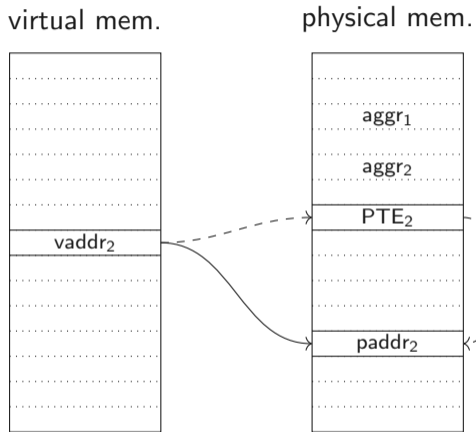
- RowPress bit flips repeatable
 - ~~Turn off cache line prefetcher~~ X
- Shuffle offsets



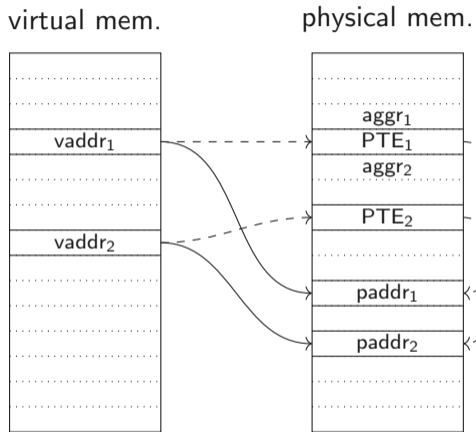
Find weak cells

- RowPress bit flips repeatable
 - ~~Turn off cache line prefetcher~~ X
- Shuffle offsets
- Search 10 exploitable bit flips

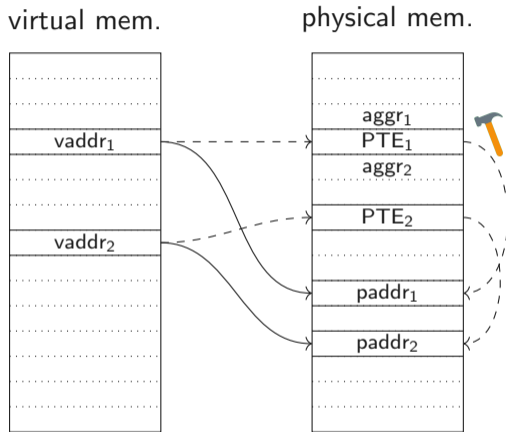
Page Table Entry Flipping



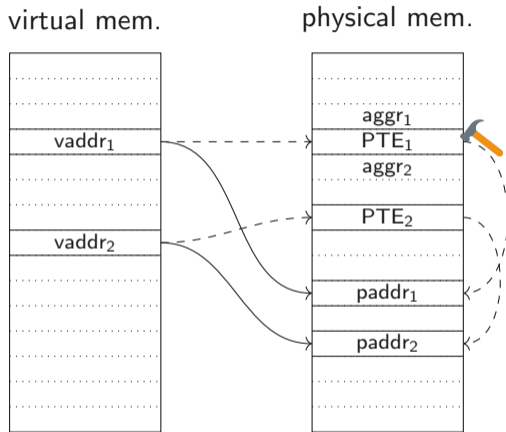
Page Table Entry Flipping



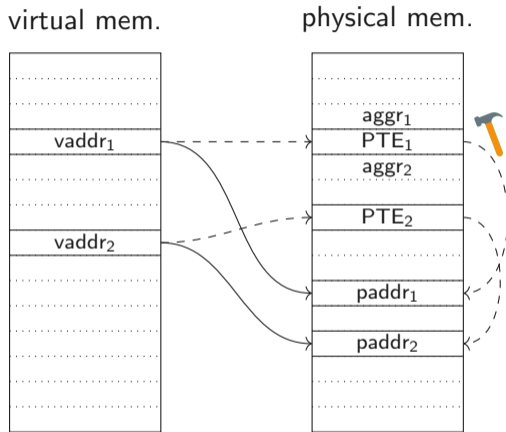
Page Table Entry Flipping



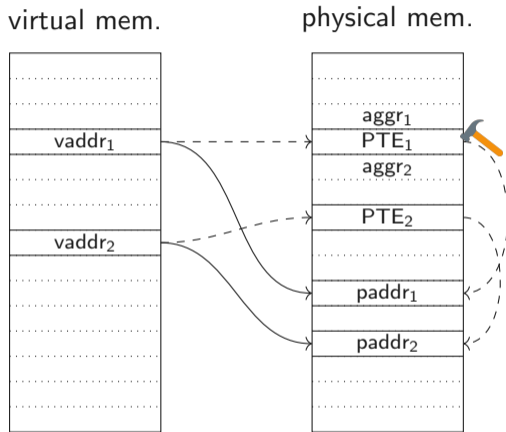
Page Table Entry Flipping



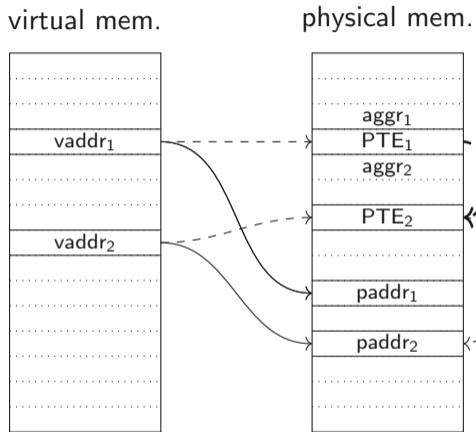
Page Table Entry Flipping



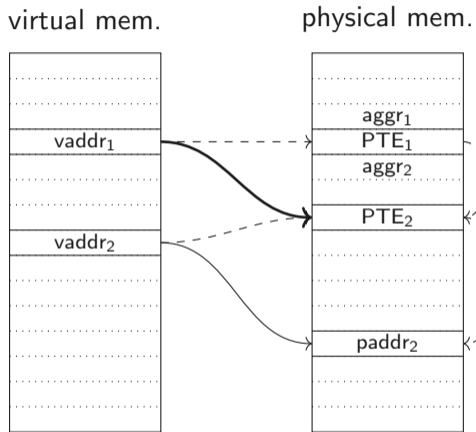
Page Table Entry Flipping



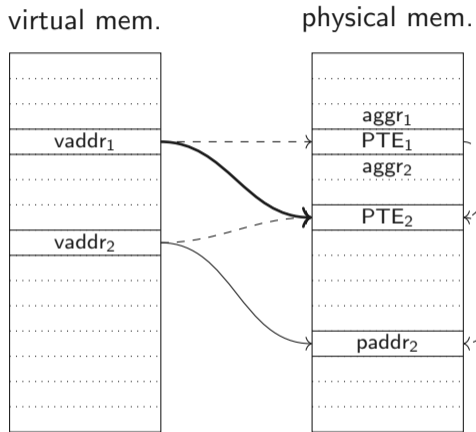
Page Table Entry Flipping



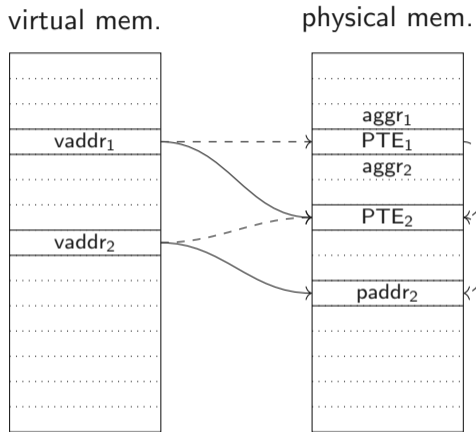
Page Table Entry Flipping



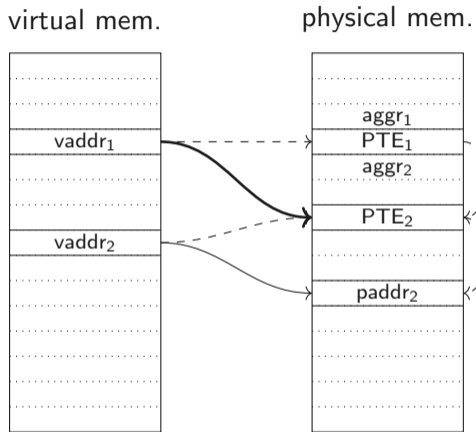
Page Table Entry Flipping



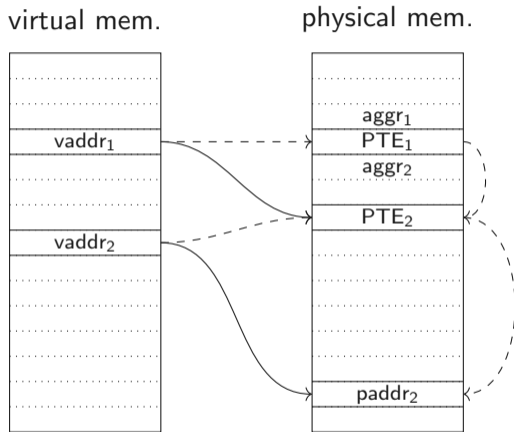
Page Table Entry Flipping



Page Table Entry Flipping



Page Table Entry Flipping







Did not successfully hammer page table.

DNF



585 s

Did not successfully hammer page table.

DNF



585 s



DNF

Did not successfully hammer page table.

Hammered page table, but didn't point to other.

DNF

585 s

DNF

616 s

Did not successfully hammer page table.

Hammered page table, but didn't point to other.

DNF

585 s

DNF

616 s

337 s

Did not successfully hammer page table.

Hammered page table, but didn't point to other.

Current Mitigation against RowPress

	DIMM	Size	MT/s	Double Sided		One Location	
				RH*	RP	RH	RP
DDR4	D1	8 GB	2133	✓	✓	✓	✓
	D2	8 GB	2133	✓	✓	✓	✓
	D3	8 GB	2400	✓	✗	~	~
	D4	8 GB	2400	✓	✗	~	~
	D5	8 GB	2133	✗	✗	~	~
	D6	8 GB	2666	✓	✗	~	~
	D7	8 GB	2666	✗	✗	~	~
	D8	8 GB	2666	✗	✗	~	~
	D9	8 GB	2666	✓	✗	~	~
	D10	8 GB	2666	✗	✗	~	~
	D11	8 GB	3000	✗	✗	~	~
	D12	8 GB	2133	✗	✗	~	~

Current Mitigation against RowPress

	DIMM	Size	MT/s	Double Sided		One Location	
				RH*	RP	RH	RP
DDR4	D1	8 GB	2133	✓	✓	✓	✓
	D2	8 GB	2133	✓	✓	✓	✓
	D3	8 GB	2400	✓	✗	~	~
	D4	8 GB	2400	✓	✗	~	~
	D5	8 GB	2133	✗	✗	~	~
	D6	8 GB	2666	✓	✗	~	~
	D7	8 GB	2666	✗	✗	~	~
	D8	8 GB	2666	✗	✗	~	~
	D9	8 GB	2666	✗	✗	~	~
	D10	8 GB	2666	✗	✗	~	~
	D11	8 GB	3000	✗	✗	~	~
	D12	8 GB	2133	✗	✗	~	~

We Were Not Always So Lucky

Mitigations focus on the **characteristics** of
Rowhammer

Characteristics 1: Flips are Infrequent



- **ECC** Error Correcting Codes



- **ECC** Error Correcting Codes
 - Additional hardware redundancy



- **ECC** Error Correcting Codes
 - Additional hardware redundancy
 - Catch **low** number of flips in HW



- **ECC** Error Correcting Codes
 - Additional hardware redundancy
 - Catch **low** number of flips in HW
- Increase the **refresh rate**

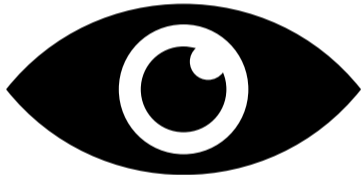


- **ECC** Error Correcting Codes
 - Additional hardware redundancy
 - Catch **low** number of flips in HW
- Increase the **refresh rate**
 - Reduce the attackers time window



- **ECC** Error Correcting Codes
 - Additional hardware redundancy
 - Catch **low** number of flips in HW
- Increase the **refresh rate**
 - Reduce the attackers time window
 - Default *64ms*

Characteristics 2: Attacks are Detectable



- Use Performance Monitoring Counters (PMCs)



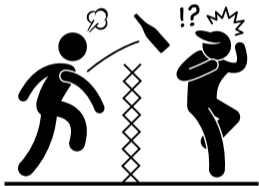
- Use Performance Monitoring Counters (PMCs)
 - Monitor DRAM activity



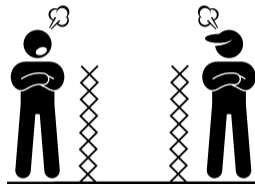
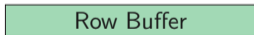
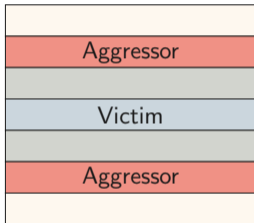
- Use Performance Monitoring Counters (PMCs)
 - Monitor DRAM activity
- Stop **suspicious** applications

Characteristics 3: Flips happen at Distance 1

Characteristics 3: Flips happen at Distance 1

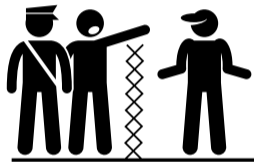
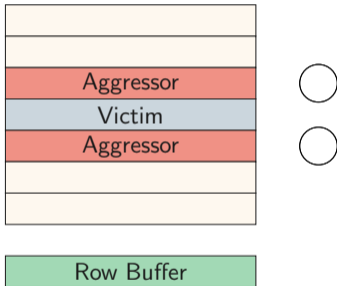


- Flips only happen in **neighboring** rows
- Mitigations based row layout
 - Guard rows
 - Targeted Row Refresh



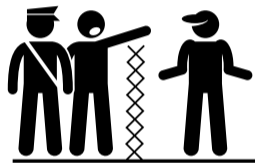
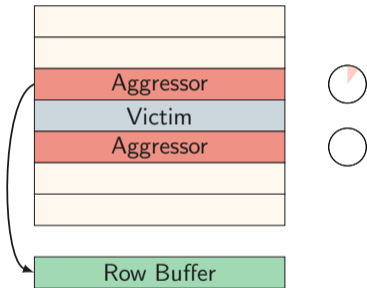
Physically distance potential victim pages.

Target Row Refresh



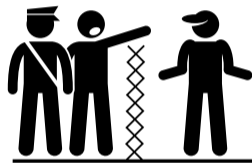
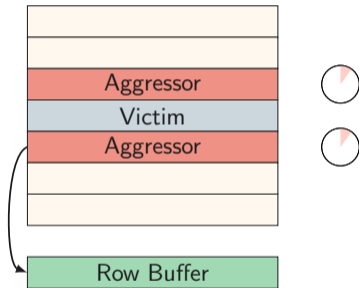
Counts row accesses and refreshes neighbors.

Target Row Refresh



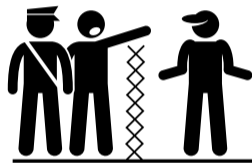
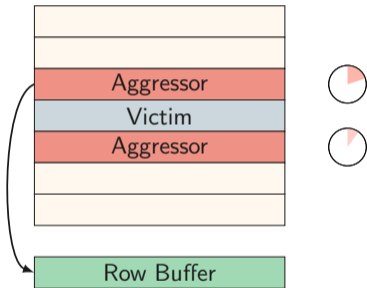
Counts row accesses and refreshes neighbors.

Target Row Refresh



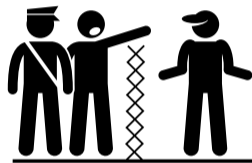
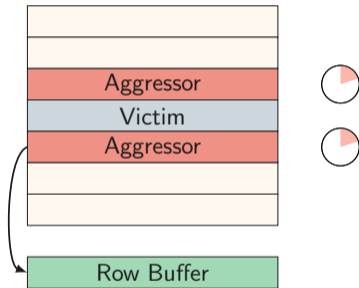
Counts row accesses and refreshes neighbors.

Target Row Refresh



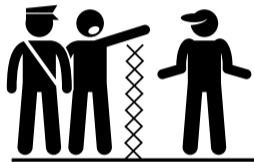
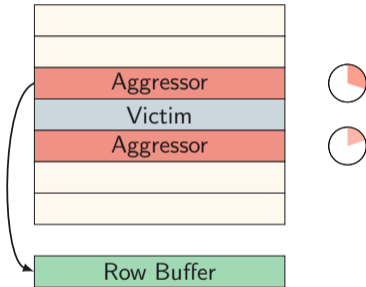
Counts row accesses and refreshes neighbors.

Target Row Refresh



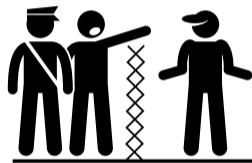
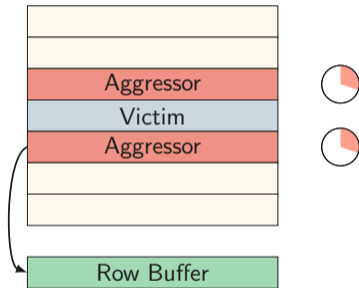
Counts row accesses and refreshes neighbors.

Target Row Refresh



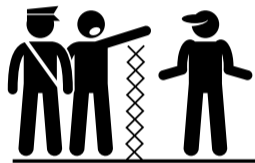
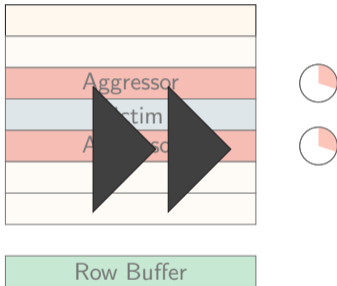
Counts row accesses and refreshes neighbors.

Target Row Refresh



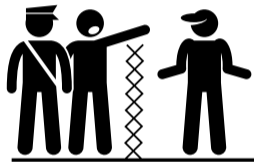
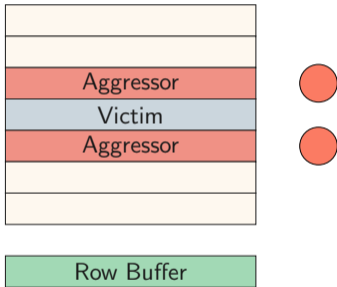
Counts row accesses and refreshes neighbors.

Target Row Refresh



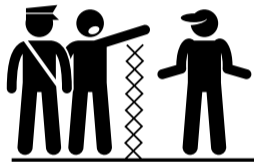
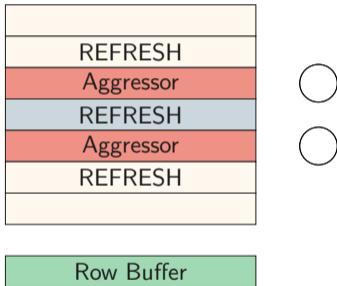
Counts row accesses and refreshes neighbors.

Target Row Refresh



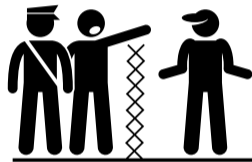
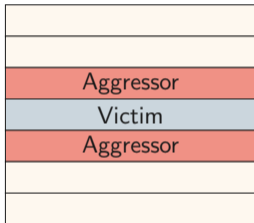
Counts row accesses and refreshes neighbors.

Target Row Refresh



Counts row accesses and refreshes neighbors.

Target Row Refresh



Counts row accesses and refreshes neighbors.

Does TRR fix Rowhammer?

The Half-Double Effect

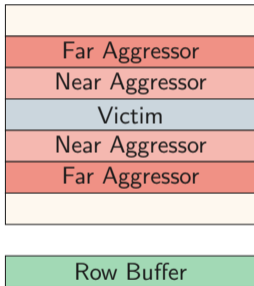
Half-Double

Hammering From the Next Row Over

Andreas Kogler Jonas Juffinger Salman Qazi Yoongu Kim
Moritz Lipp Nicolas Boichat Eric Shiu Mattias Nissler Daniel Gruss

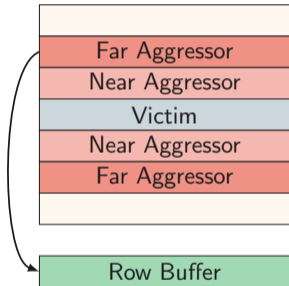
2024-07-16

IAIK, Graz University of Technology, Austria

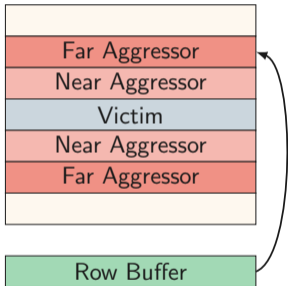


Make TRR a confused debuty that helps with hammering.

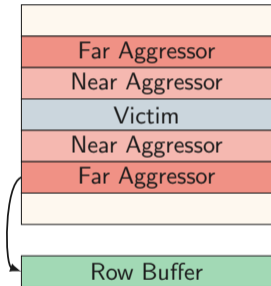
Half-Double Rowhammer



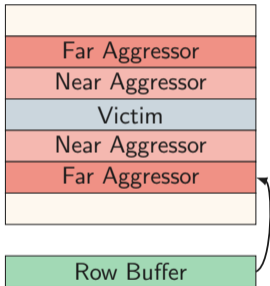
Make TRR a confused deputy that helps with hammering.



Make TRR a confused deputy that helps with hammering.

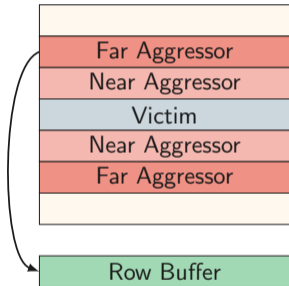


Make TRR a confused deputy that helps with hammering.



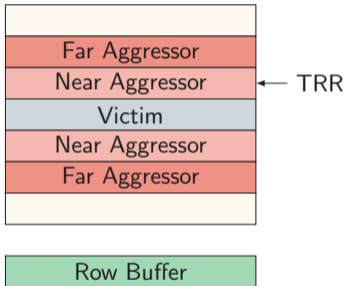
Make TRR a confused debuty that helps with hammering.

Half-Double Rowhammer

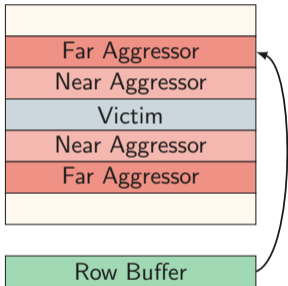


Make TRR a confused debuty that helps with hammering.

Half-Double Rowhammer

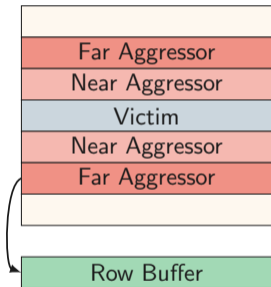


Make TRR a confused debuty that helps with hammering.



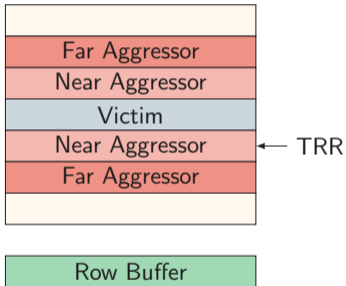
Make TRR a confused deputy that helps with hammering.

Half-Double Rowhammer

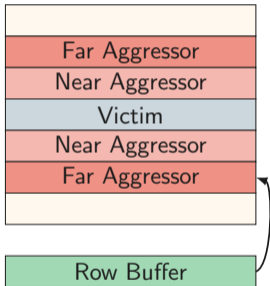


Make TRR a confused deputy that helps with hammering.

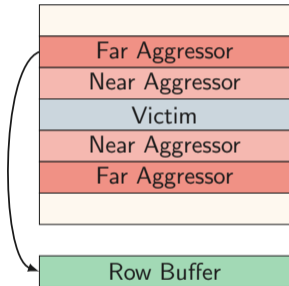
Half-Double Rowhammer



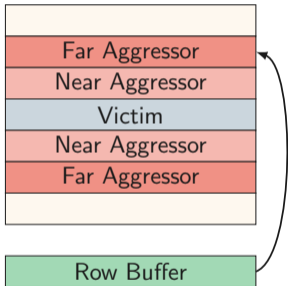
Make TRR a confused deputy that helps with hammering.



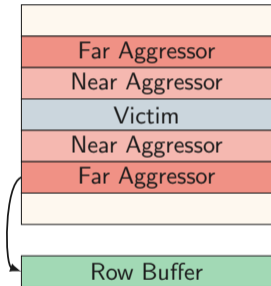
Make TRR a confused debuty that helps with hammering.



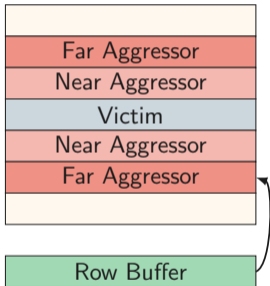
Make TRR a confused deity that helps with hammering.



Make TRR a confused deputy that helps with hammering.

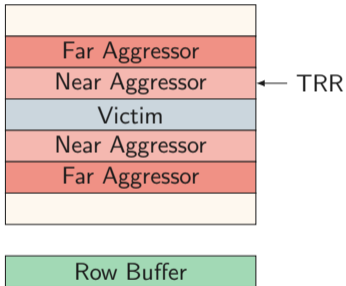


Make TRR a confused deputy that helps with hammering.



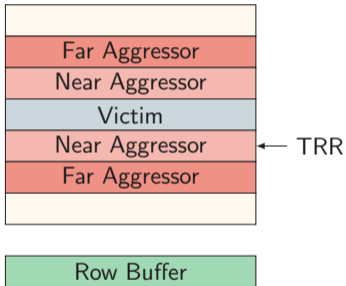
Make TRR a confused debuty that helps with hammering.

Half-Double Rowhammer



Make TRR a confused deputy that helps with hammering.

Half-Double Rowhammer



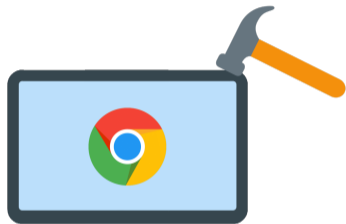
Make TRR a confused debuty that helps with hammering.



Make TRR a confused deity that helps with hammering.

Exploitable in the Wild?

Half-Double End-to-End Exploit



```
0
1 0 1
  1
1 0 1
  0 1
    1
    0
  0
```

- Page table exploit
- Bit-flip verification using speculative execution
- Exploit in less than 40 min

Rethinking Rowhammer Mitigations



Generic approach to data integrity
protection

CSI:Rowhammer – Cryptographic Security and Integrity against Rowhammer



- **Generic** approach to data integrity protection

CSI:Rowhammer – Cryptographic Security and Integrity against Rowhammer



- **Generic** approach to data integrity protection
- Detect **all** data integrity failures with a MAC

CSI:Rowhammer – Cryptographic Security and Integrity against Rowhammer



- **Generic** approach to data integrity protection
- Detect **all** data integrity failures with a MAC
- Best effort correction

CSI:Rowhammer – Cryptographic Security and Integrity against Rowhammer



- **Generic** approach to data integrity protection
- Detect **all** data integrity failures with a MAC
- Best effort correction
- **All** Rowhammer attacks are DoS in the **worst case**

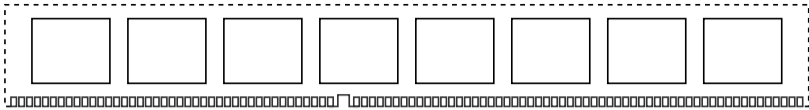
CSI:Rowhammer

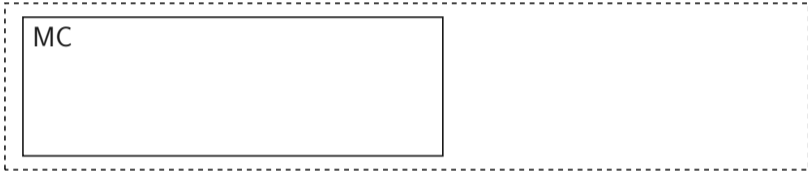
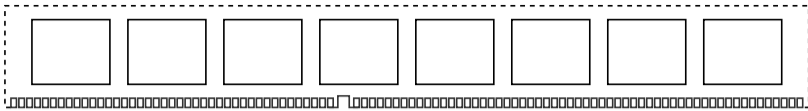
Cryptographic Security and Integrity against Rowhammer

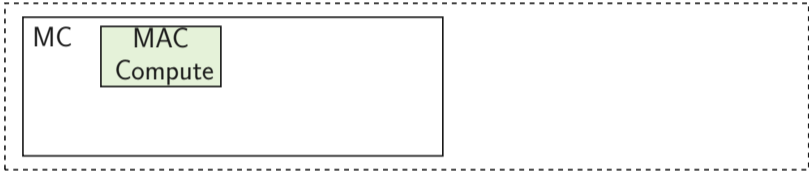
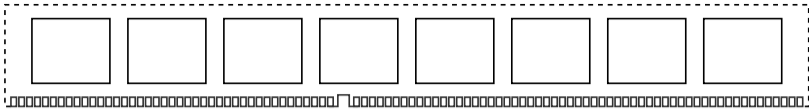
Jonas Juffinger **Lukas Lamster** **Andreas Kogler** **Moritz Lipp**
Maria Eichseder **Daniel Gruss**

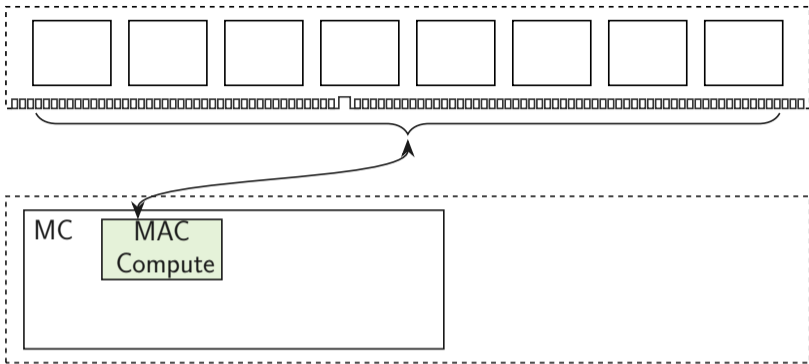
2024-07-16

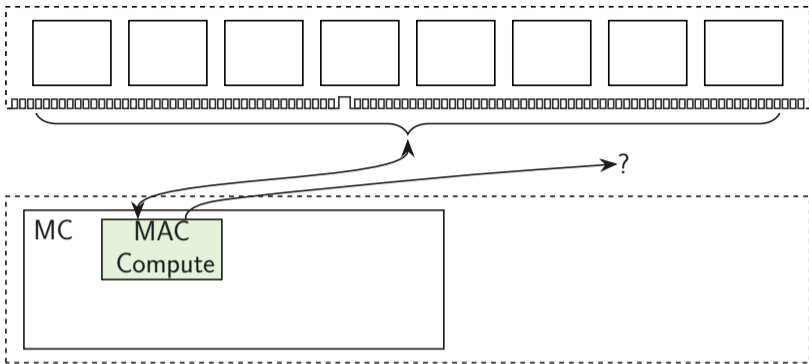
IAIK, Graz University of Technology, Austria

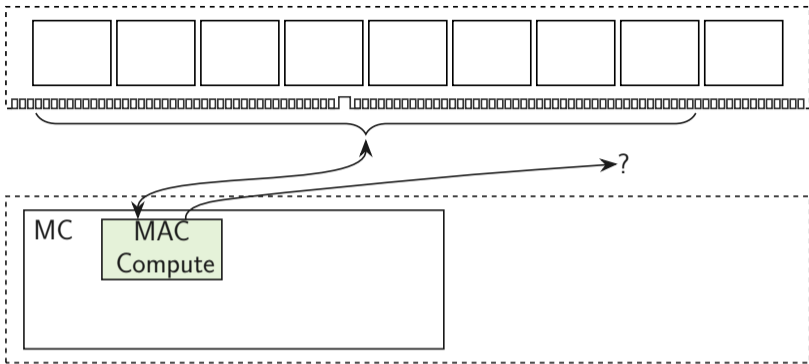








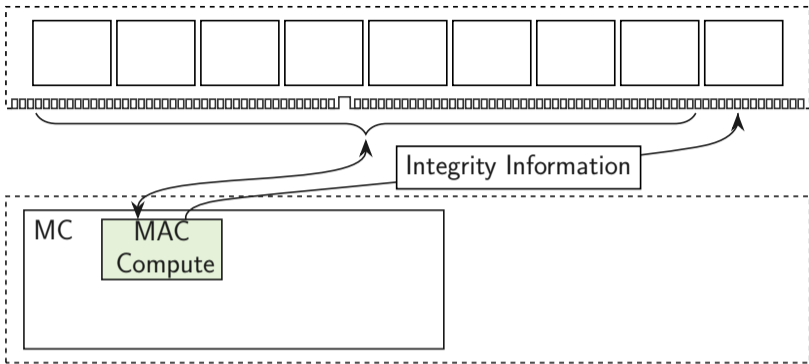


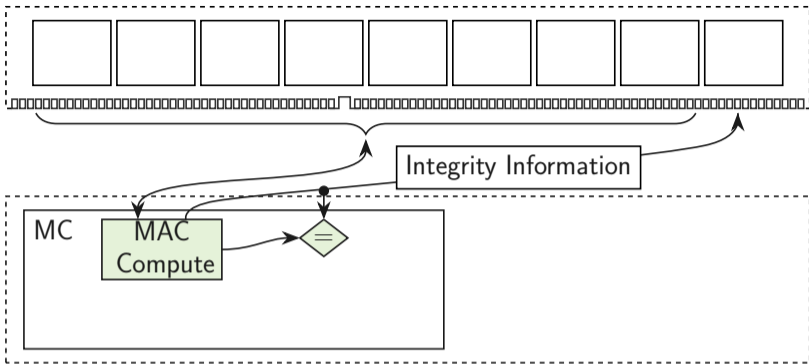


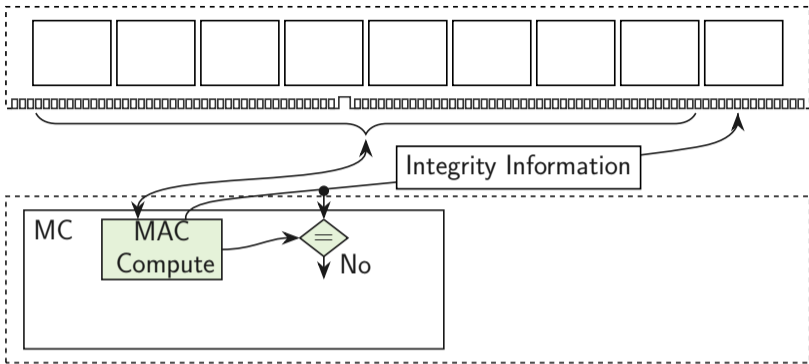
MC

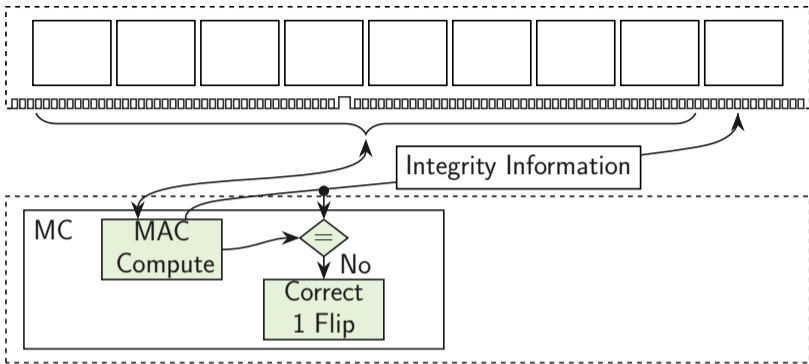
MAC
Compute

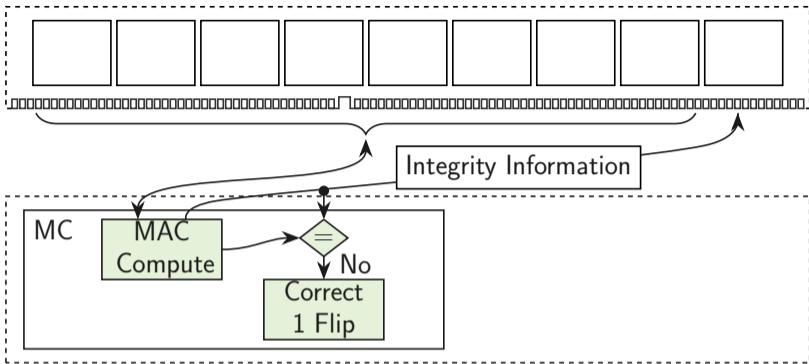
?



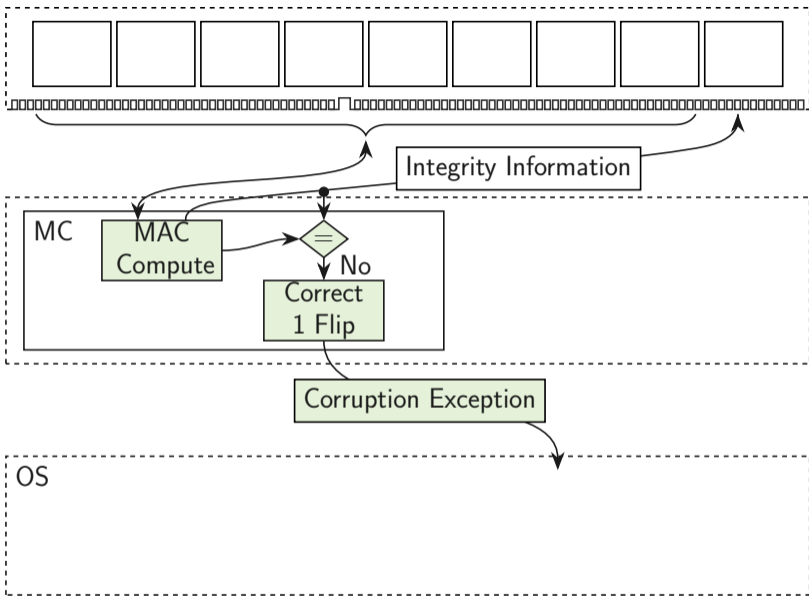


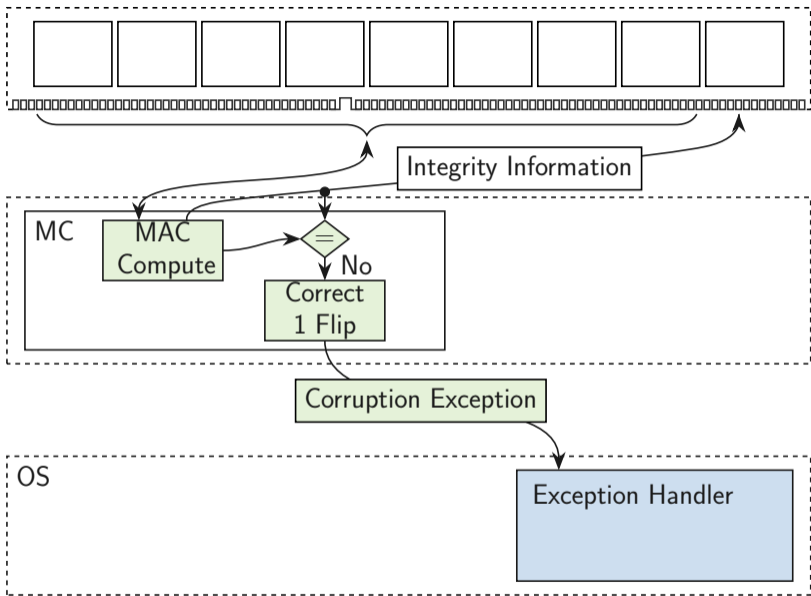


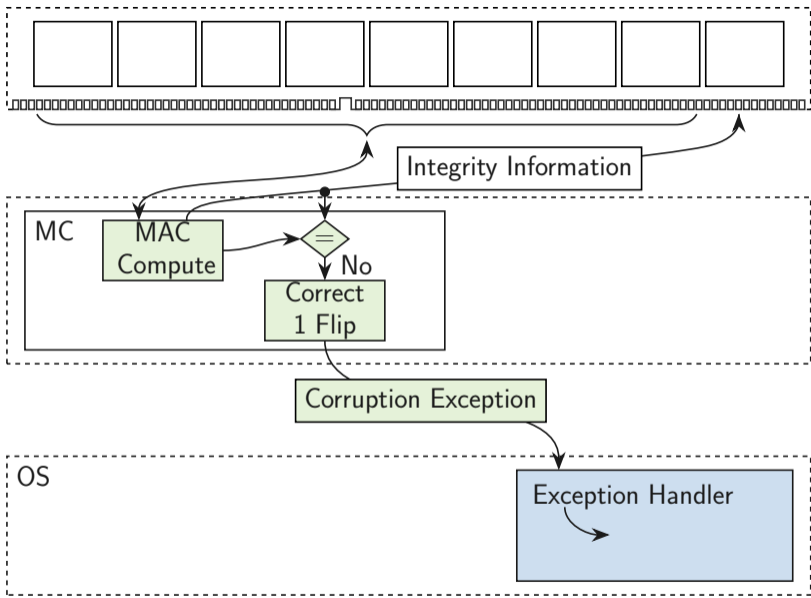


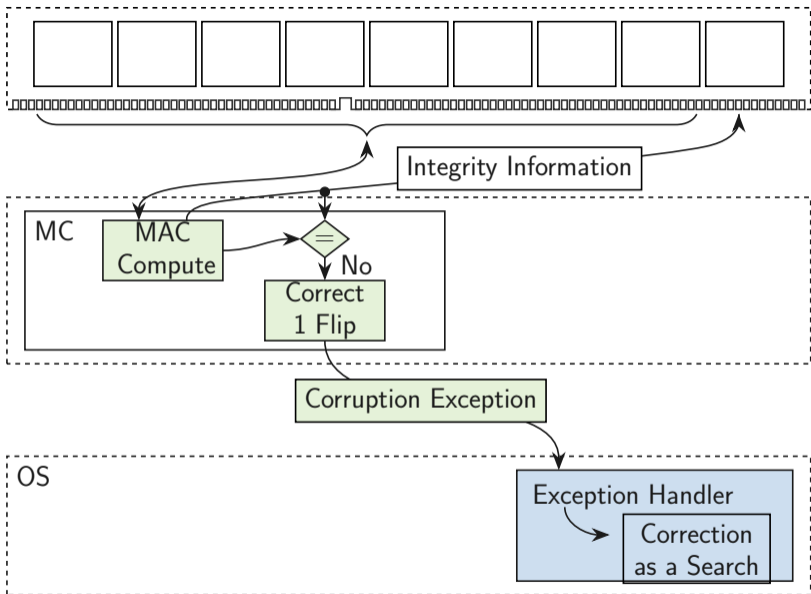


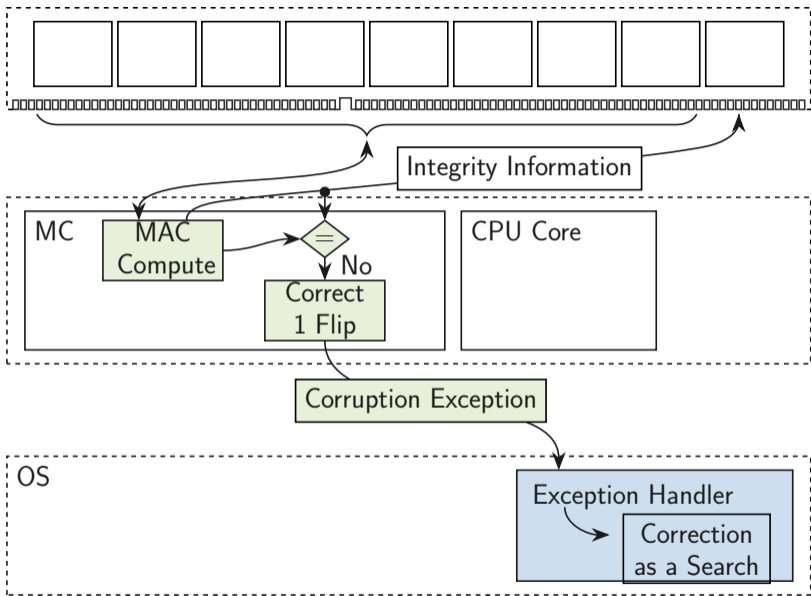
OS

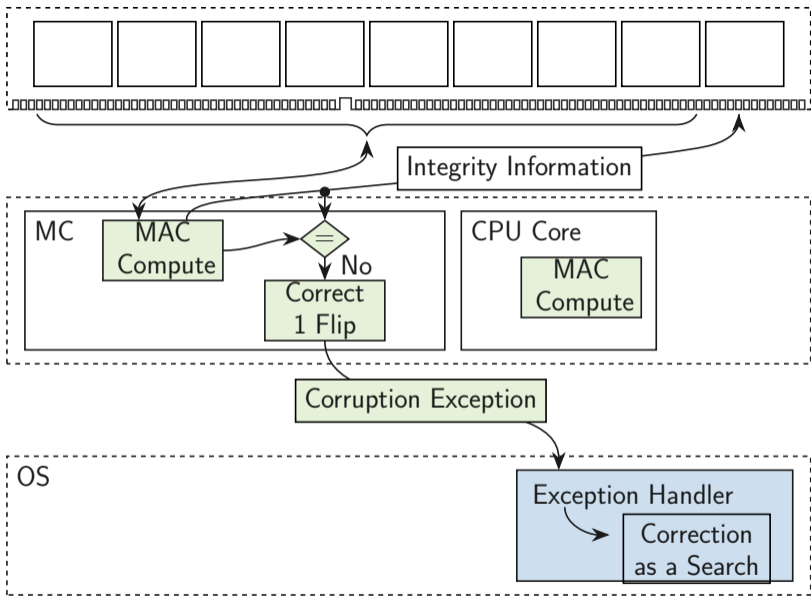


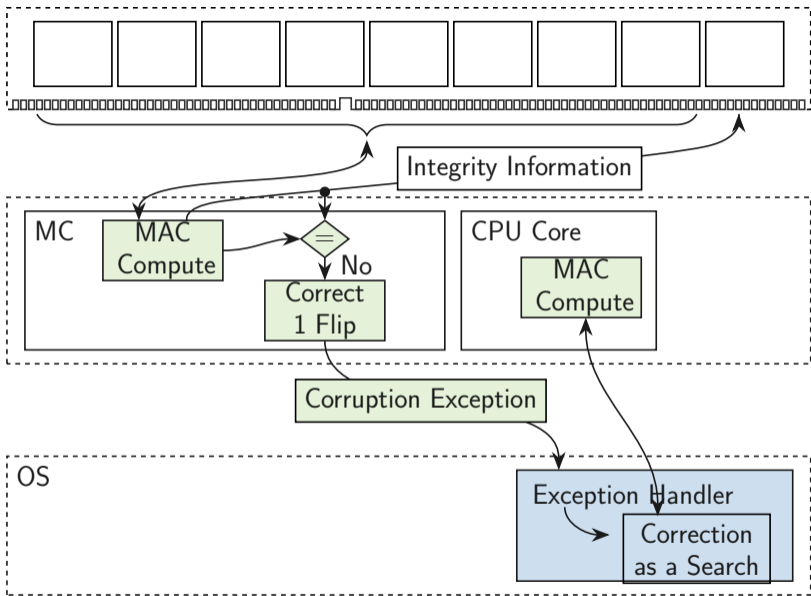


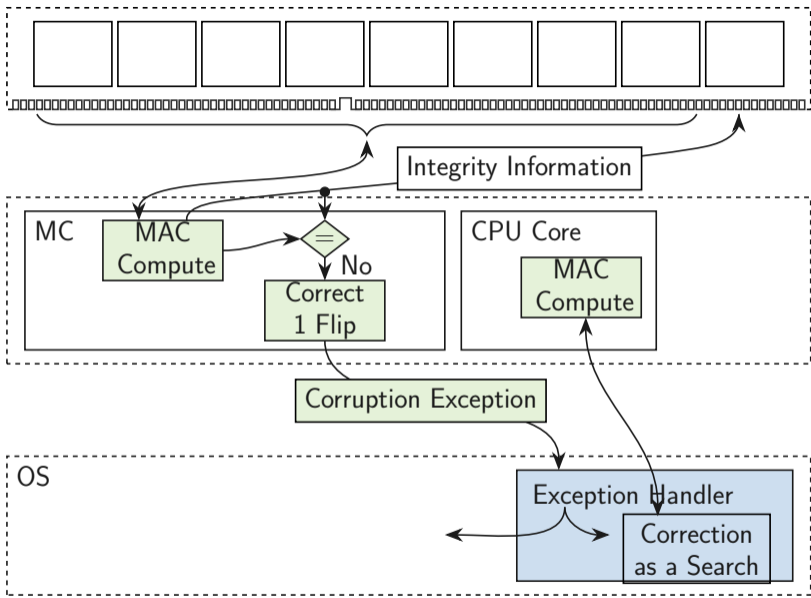


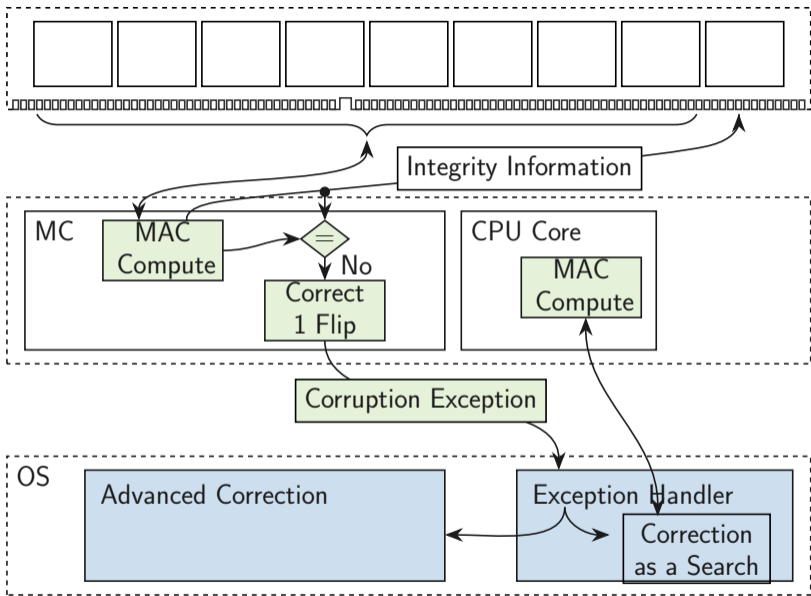


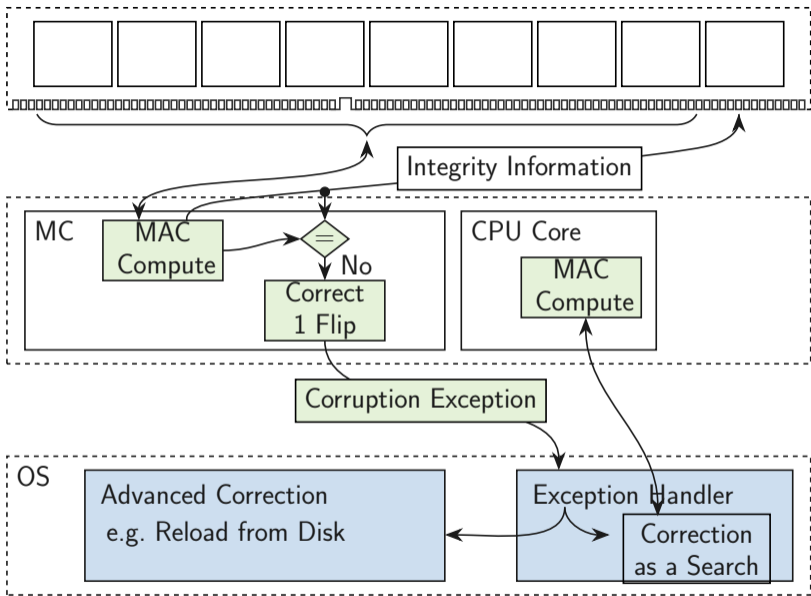


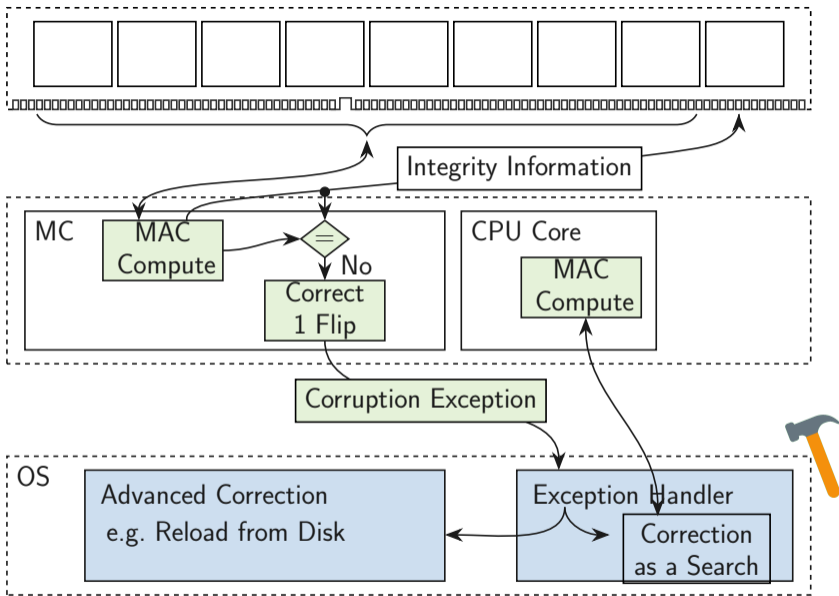


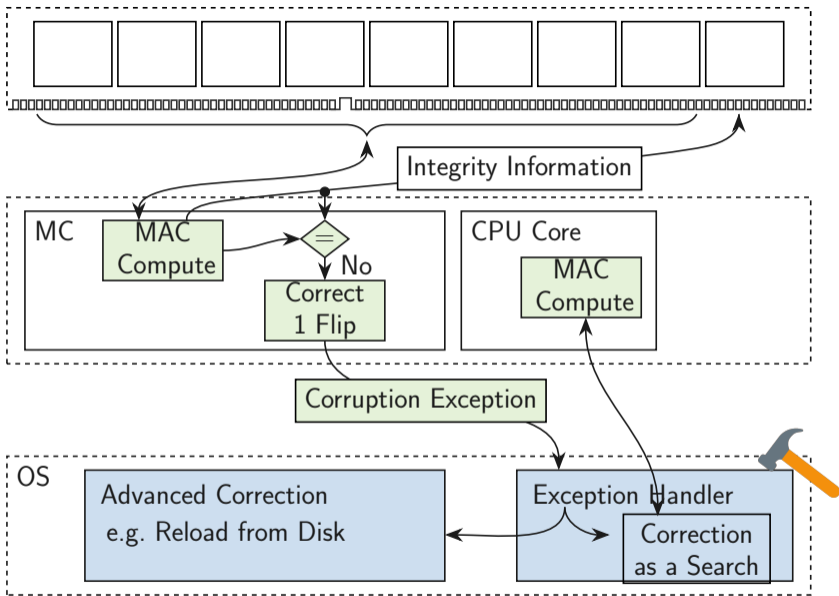


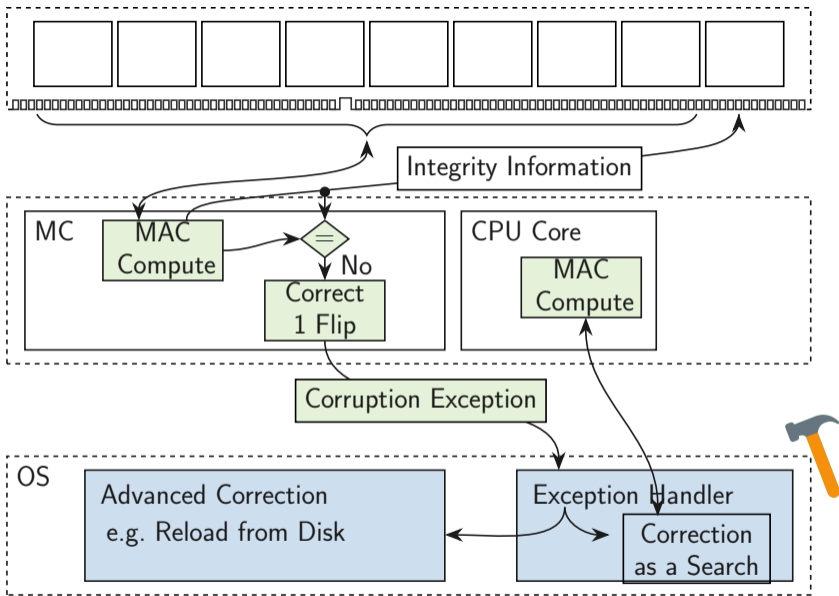


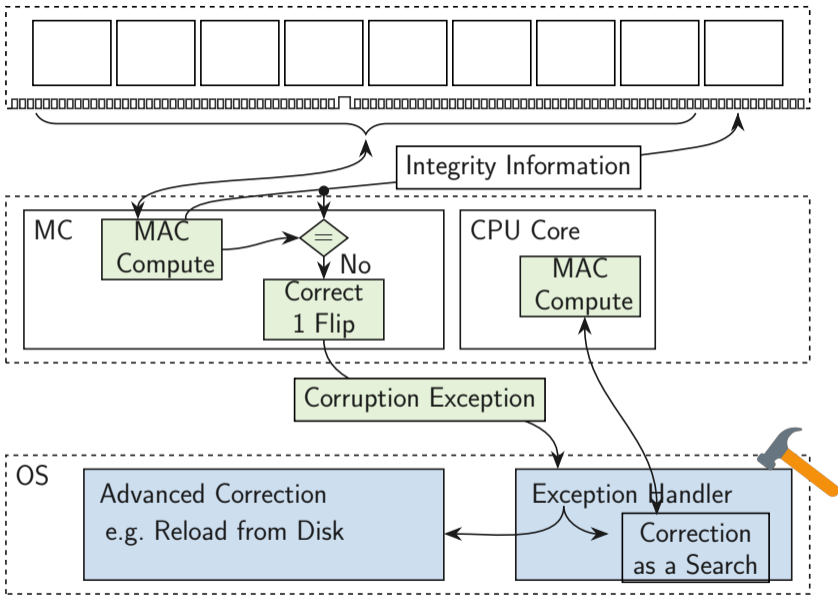


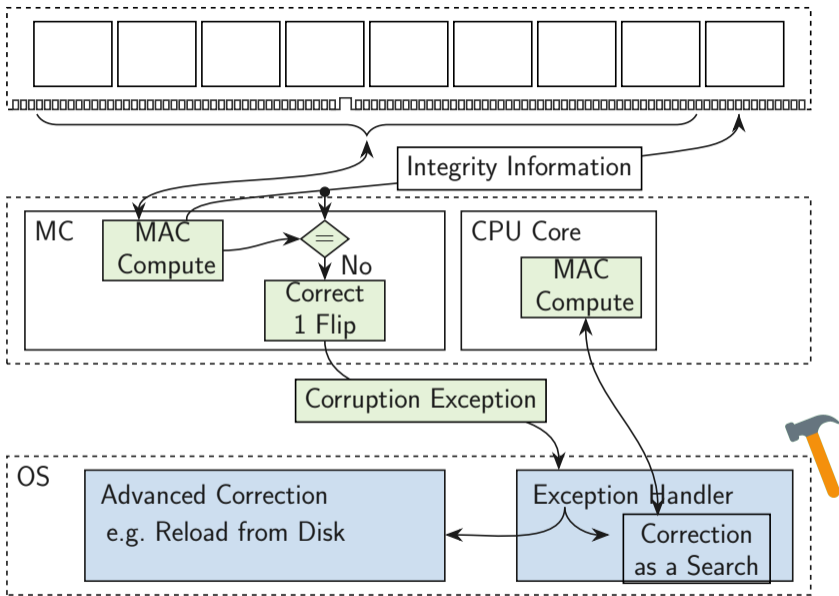


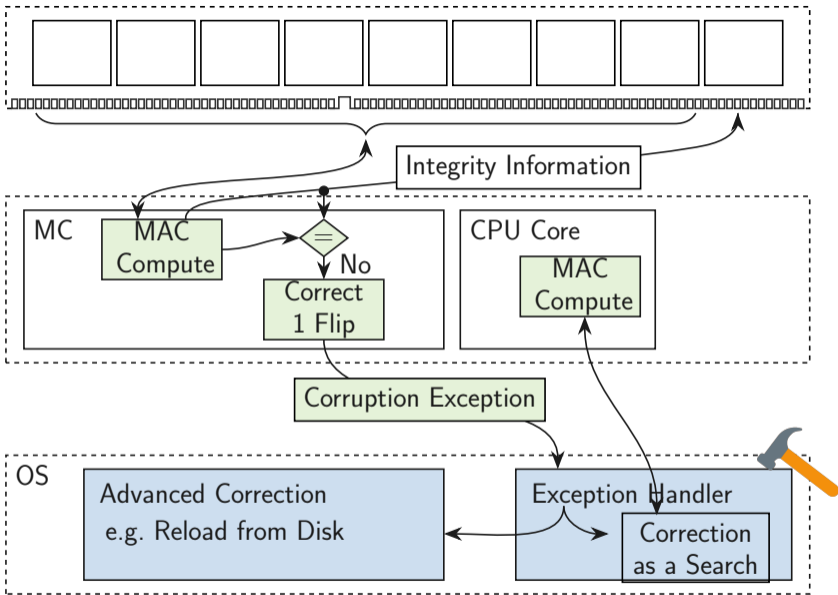


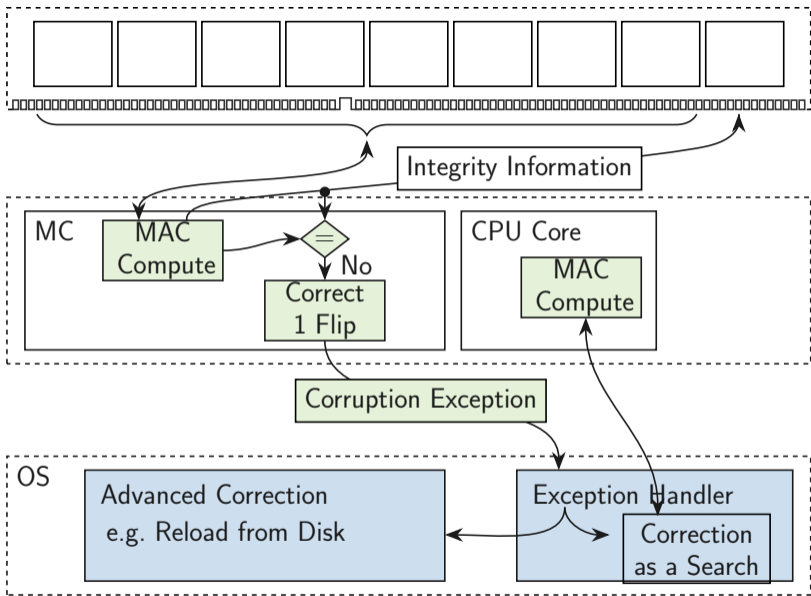


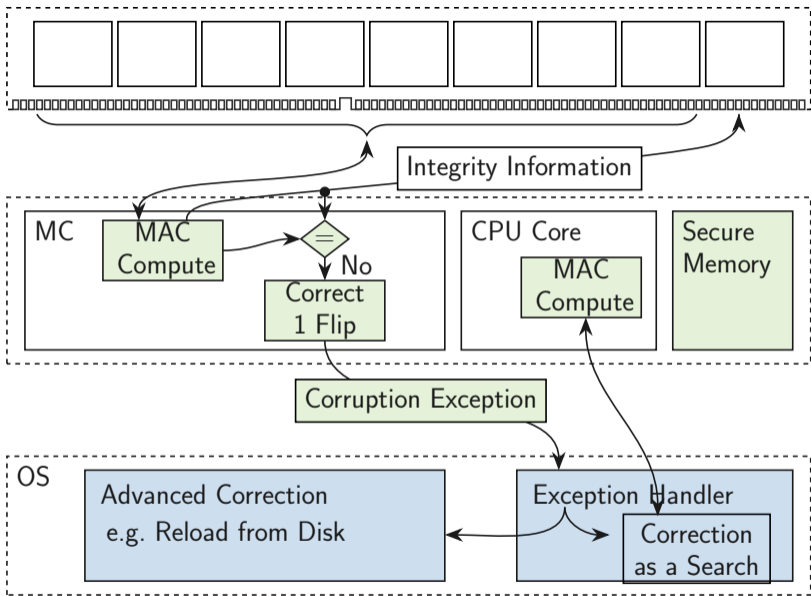


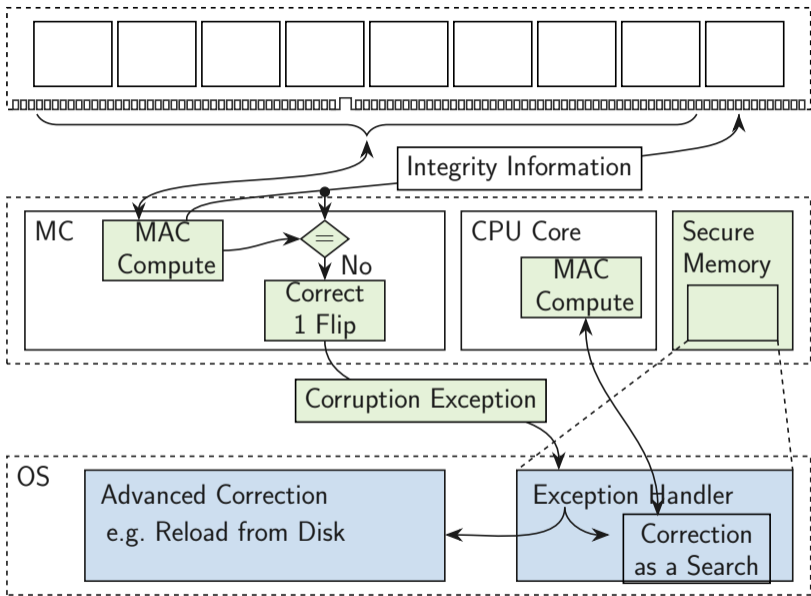




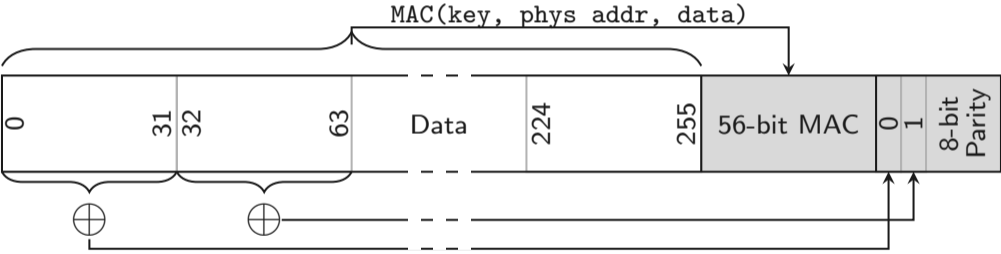


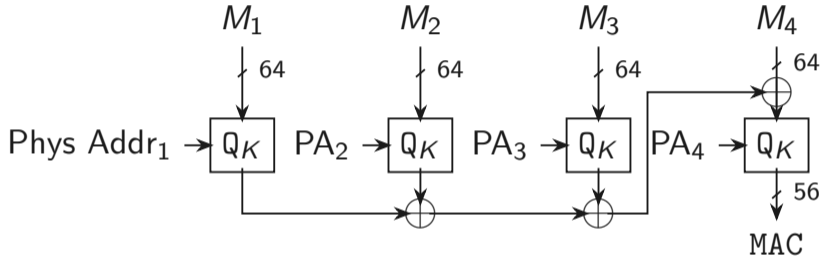


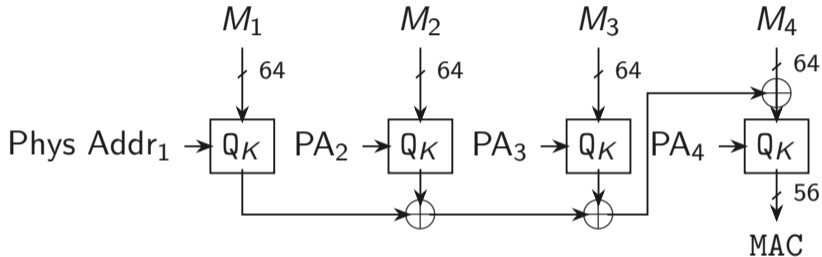




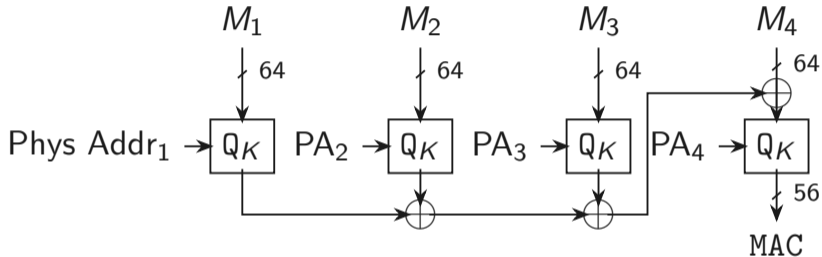
Partitioning



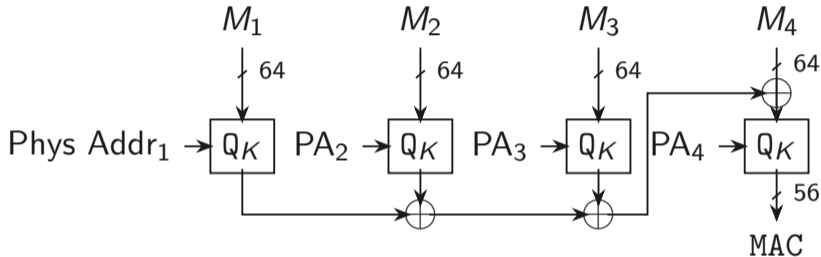




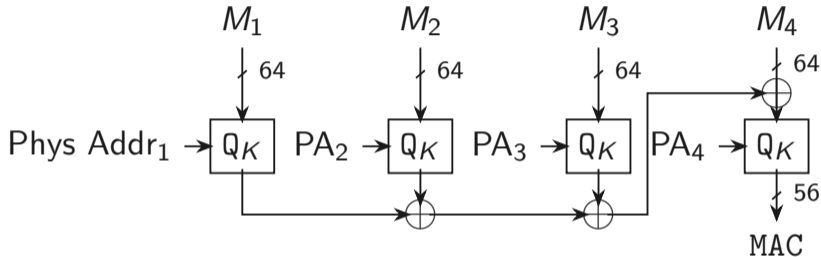
- Pipelined PMAC construction



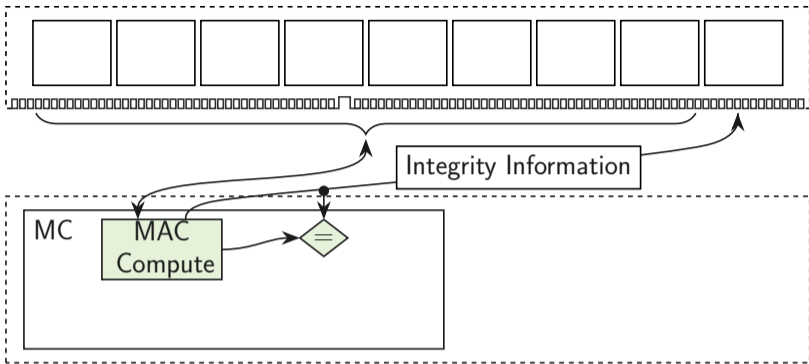
- Pipelined PMAC construction
- QARMA₅-64- σ_0 block cipher
[1]

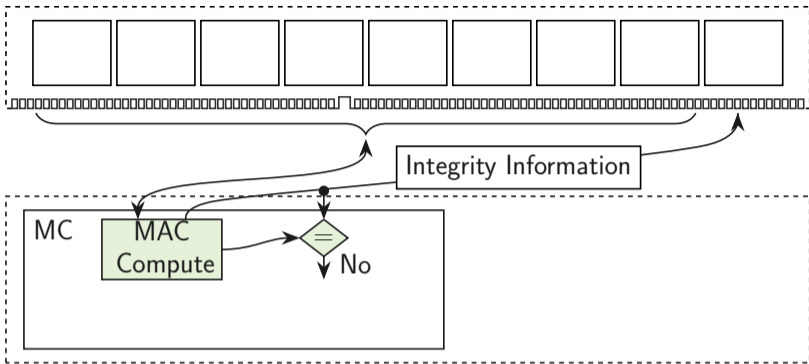


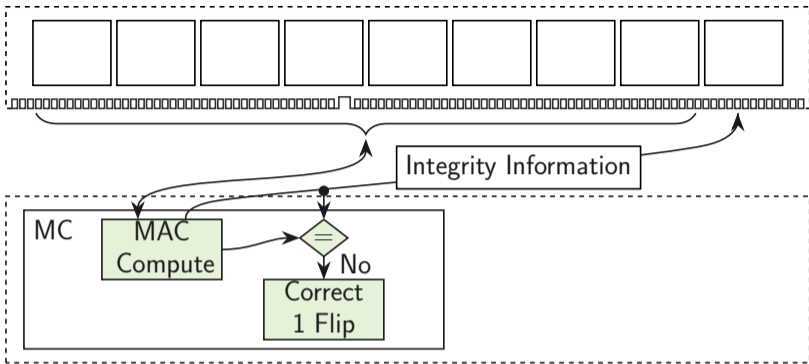
- Pipelined PMAC construction
 - QARMA₅-64- σ_0 block cipher
 - 256-bit data 5.13 ns
- [1]



- Pipelined PMAC construction
- QARMA₅-64- σ_0 block cipher [1]
- 256-bit data 5.13 ns
- 512-bit data 6.60 ns









MACs **cannot** correct bit flips



- MACs **cannot** correct bit flips



- MACs **cannot** correct bit flips
- Brute force search with approximate equality



- MACs **cannot** correct bit flips
- Brute force search with approximate equality
0010110100101101 → 01011010

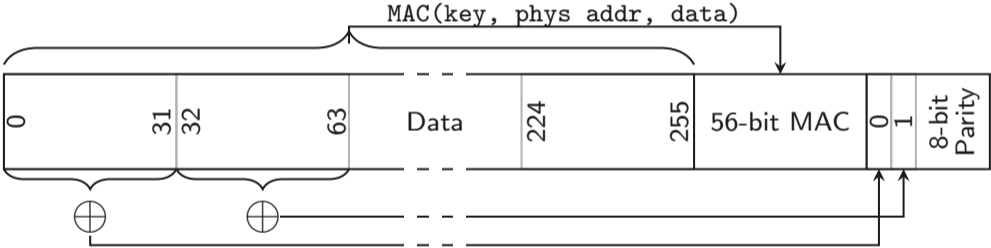


- MACs **cannot** correct bit flips
- Brute force search with approximate equality
0010110100101101 → 01011010
0010110100101101 → 01010010 ✓

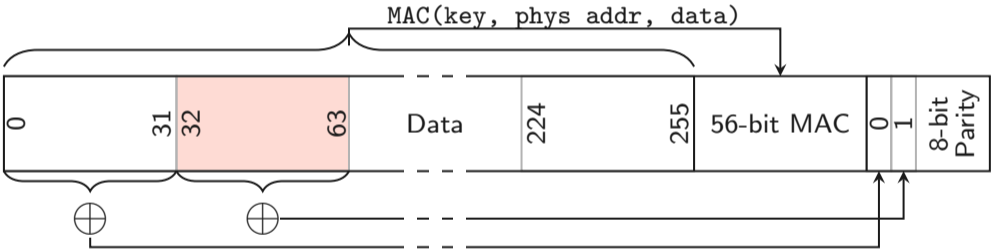


- MACs **cannot** correct bit flips
- Brute force search with approximate equality
0010110100101101 → 01011010
0010110100101101 → 01010010 ✓
- Parity bits to **shrink** search space

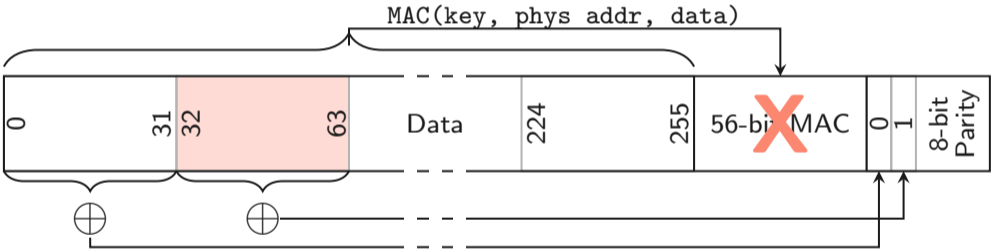
Single Bit Flip Correction



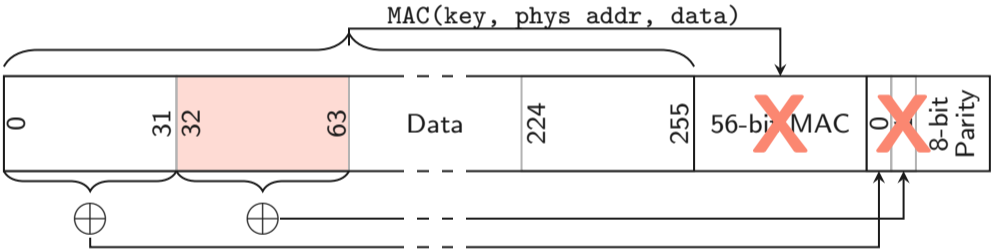
Single Bit Flip Correction



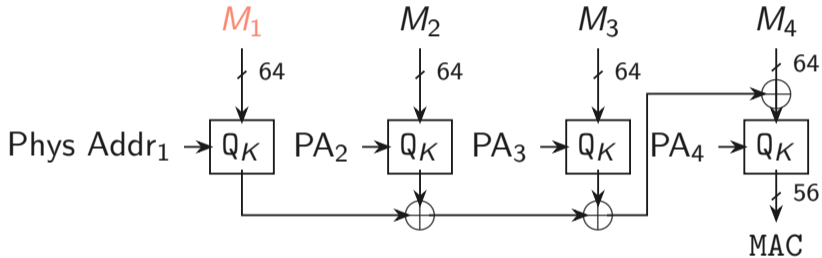
Single Bit Flip Correction



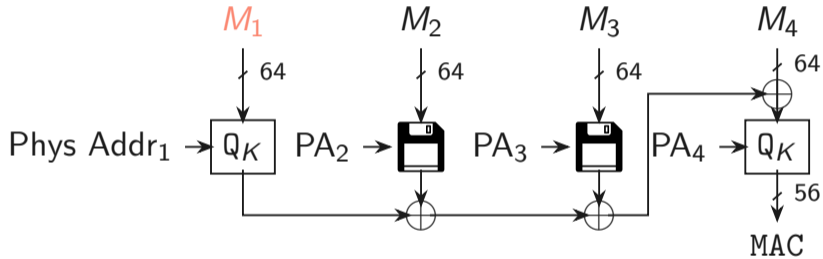
Single Bit Flip Correction



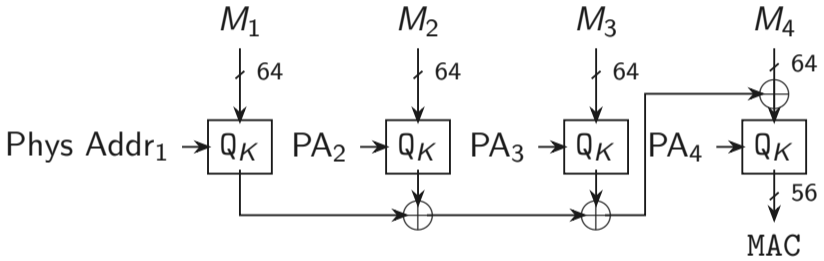
Single Bit Flip Correction



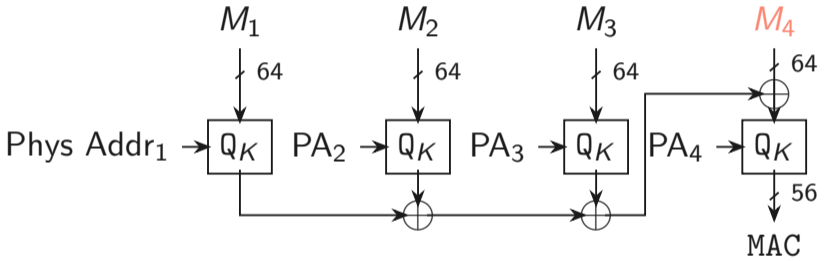
Single Bit Flip Correction



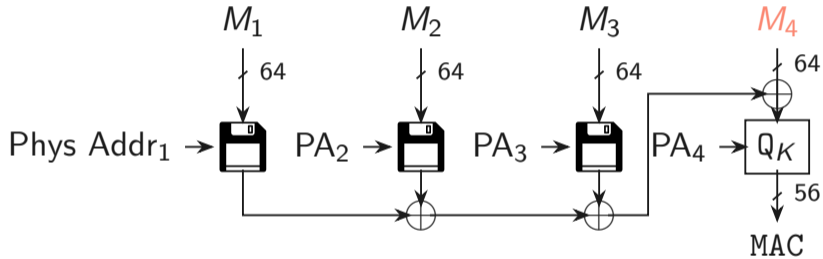
Single Bit Flip Correction



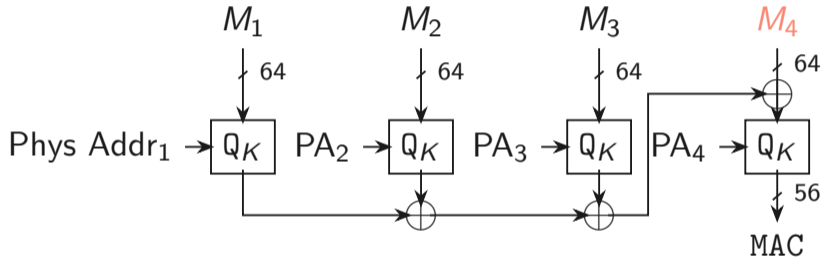
Single Bit Flip Correction



Single Bit Flip Correction

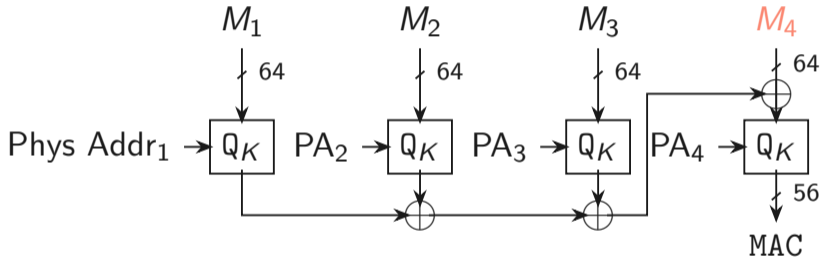


Single Bit Flip Correction



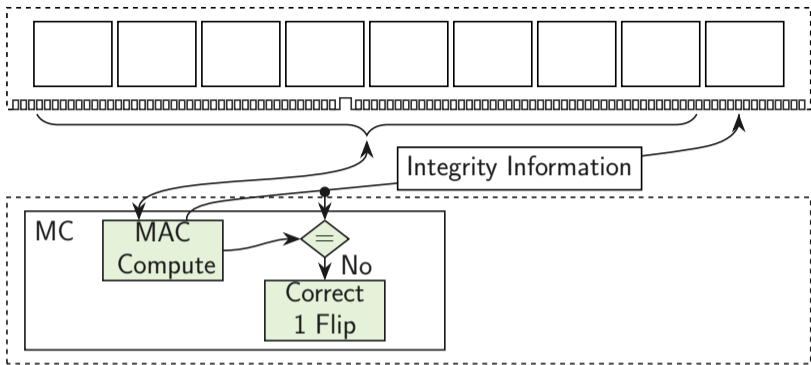
- 256-bit data 11 ns

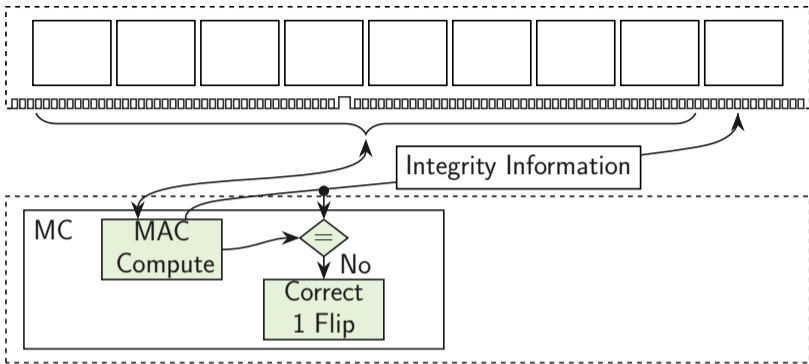
Single Bit Flip Correction



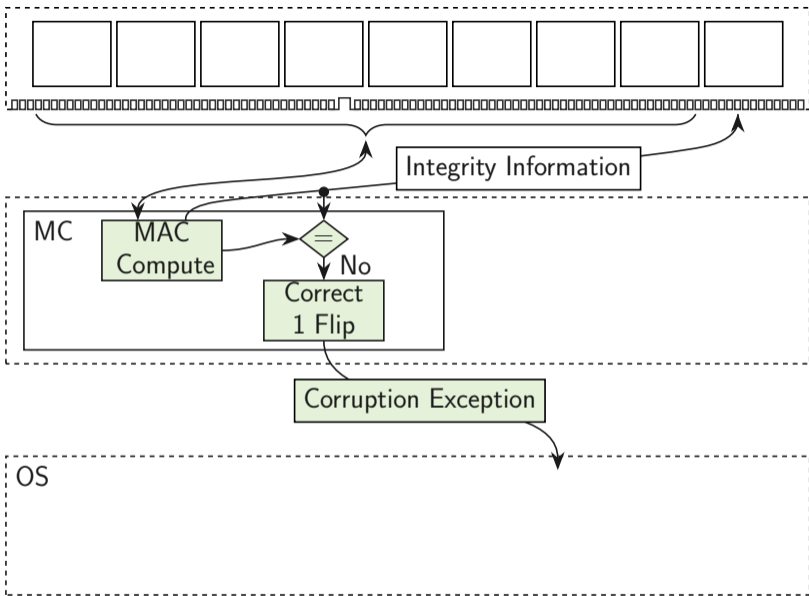
- 256-bit data 11 ns

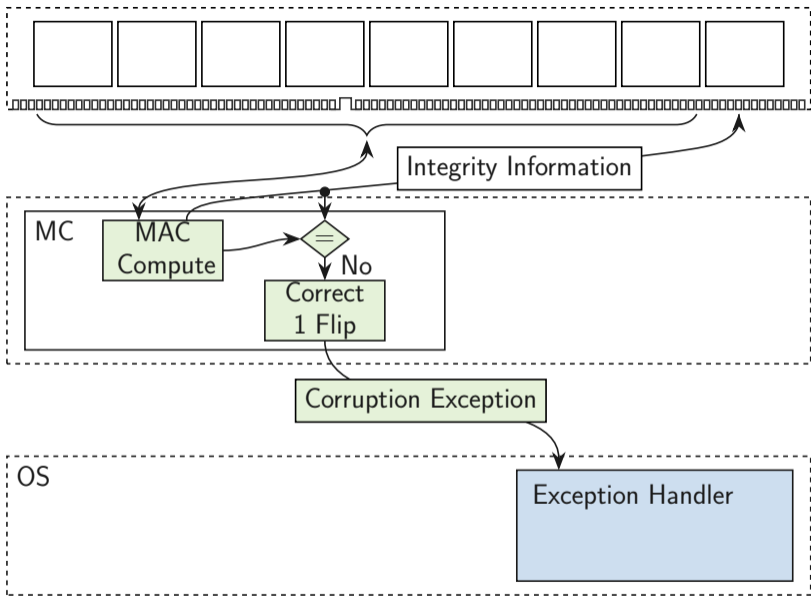
- 512-bit data 18.3 ns

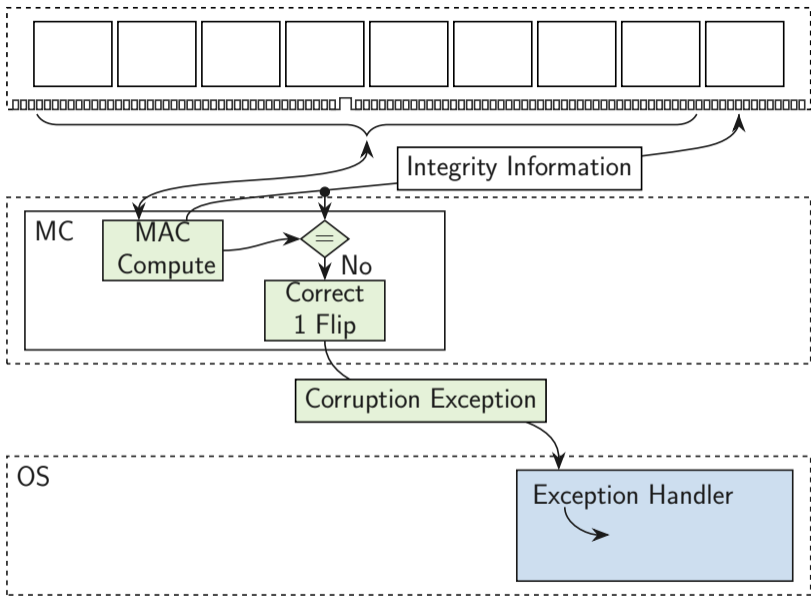


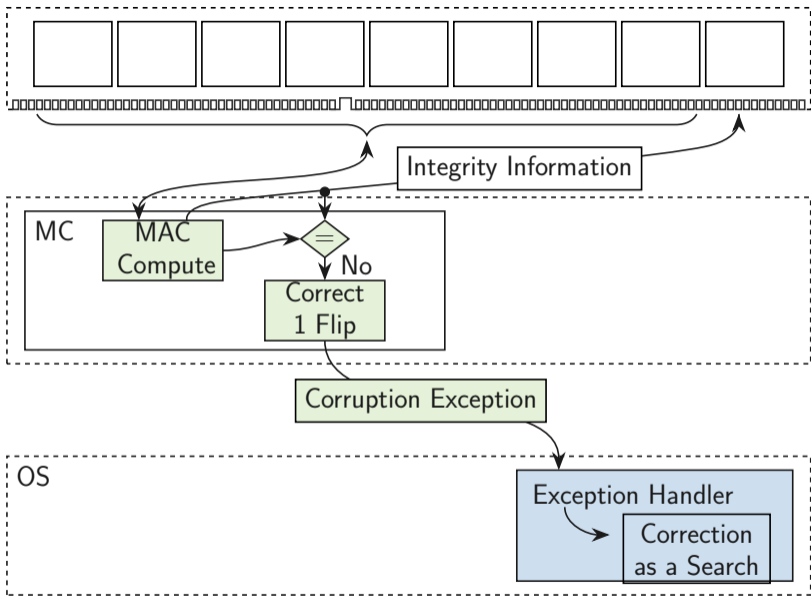


OS









Error Correction in Software



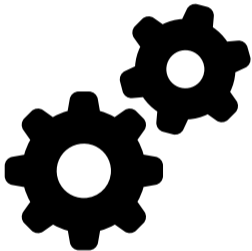
- Similar to page fault



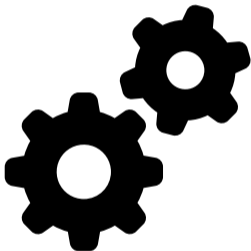
- Similar to page fault
- **Physical address** in CR2



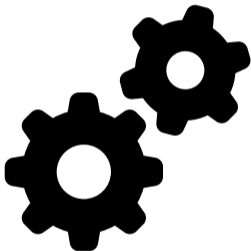
- Similar to page fault
- **Physical address** in CR2
- No dependency on page tables



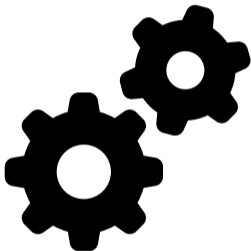
- Brute force **all** possible flips



- Brute force **all** possible flips
- Guided by parity bits



- Brute force **all** possible flips
- Guided by parity bits
- As **fast** as possible



- Brute force **all** possible flips
- Guided by parity bits
- As **fast** as possible
- As **small** as possible


```

goto *LBL_PB[double_flips];
}

// The for outer PB loops are for parity bit input blocks with an even
// number of flips and therefore no change in the parity bits.
LBL_4_PB:
for (pb[3] = 0; pb[3] < CONST_NUM_PARITY_BITS; pb[3]++) {

    LBL_3_PB:
    for (pb[2] = 0; pb[2] <= pb[3]; pb[2]++) {}

        LBL_2_PB:
        for (pb[1] = 0; pb[1] <= pb[2]; pb[1]++) {

            LBL_1_PB:
            for (pb[0] = 0; pb[0] <= pb[1]; pb[0]++) {

                goto *lbl_flip;

                LBL_8_FLIP:
                for (bit[7] = 0, flip_mask[7] = 1, data_parity_block[7] = &data[*ssb[7]];
                    bit[7] < *seb[7]; bit[7]++, flip_mask[7] <<= 1) {
                    *data_parity_block[7] ^= flip_mask[7];

                LBL_7_FLIP:
                for (bit[6] = 0, flip_mask[6] = 1, data_parity_block[6] = &data[*ssb[6]];
                    bit[6] < *seb[6]; bit[6]++, flip_mask[6] <<= 1) {
                    *data_parity_block[6] ^= flip_mask[6];

                LBL_6_FLIP:
                for (bit[5] = 0, flip_mask[5] = 1, data_parity_block[5] = &data[*ssb[5]];
                    bit[5] < *seb[5]; bit[5]++, flip_mask[5] <<= 1) {
                    *data_parity_block[5] ^= flip_mask[5];

                LBL_5_FLIP:
                for (bit[4] = 0, flip_mask[4] = 1, data_parity_block[4] = &data[*ssb[4]];
                    bit[4] < *seb[4]; bit[4]++, flip_mask[4] <<= 1) {
                    *data_parity_block[4] ^= flip_mask[4];

                LBL_4_FLIP:
                for (bit[3] = 0, flip_mask[3] = 1, data_parity_block[3] = &data[*ssb[3]];
                    bit[3] < *seb[3]; bit[3]++, flip_mask[3] <<= 1) {
                    *data_parity_block[3] ^= flip_mask[3];

                LBL_3_FLIP:
                for (bit[2] = 0, flip_mask[2] = 1, data_parity_block[2] = &data[*ssb[2]];
                    bit[2] < *seb[2]; bit[2]++, flip_mask[2] <<= 1) {
                    *data_parity_block[2] ^= flip_mask[2];

                LBL_2_FLIP:
                for (bit[1] = 0, flip_mask[1] = 1, data_parity_block[1] = &data[*ssb[1]];
                    bit[1] < *seb[1]; bit[1]++, flip_mask[1] <<= 1) {
                    *data_parity_block[1] ^= flip_mask[1];

                // We also do the correction of single bit errors in this code
                // to check the correctness of the algorithm.
                LBL_1_FLIP:

```

```

*data_parity_block[2] ^= flip_mask[2];

LBL_2_FLIP:
for (bit[1] = 0, flip_mask[1] = 1, data_parity_block[1] = &data[*ssb[1]];
    bit[1] < *seb[1]; bit[1]++, flip_mask[1] <<= 1) {
    *data_parity_block[1] ^= flip_mask[1];

    // We also do the correction of single bit errors in this code
    // to check the correctness of the algorithm.
    LBL_1_FLIP:
    for (bit[0] = 0, flip_mask[0] = 1, data_parity_block[0] = &data[*ssb[0]];
        bit[0] < *seb[0]; bit[0]++, flip_mask[0] <<= 1) {
        *data_parity_block[0] ^= flip_mask[0];
        mac_computations++;

        if (COMPARE_MACS(data, correct_data)) {
            return mac_computations;
        }

        *data_parity_block[0] ^= flip_mask[0];
    }
    goto *lbl_flip_break[1];
    LBL_1_FLIP_BREAK:

    *data_parity_block[1] ^= flip_mask[1];
}
goto *lbl_flip_break[2];
LBL_2_FLIP_BREAK:
*data_parity_block[2] ^= flip_mask[2];

```

```

*data_parity_block[2] ^= flip_mask[2];

LBL_2_FLIP:
for (bit[1] = 0, flip_mask[1] = 1, data_parity_block[1] = &data[*ssb[1]];
    bit[1] < *seb[1]; bit[1]++, flip_mask[1] <<= 1) {
    *data_parity_block[1] ^= flip_mask[1];

    // We also do the correction of single bit errors in this code
    // to check the correctness of the algorithm.
    LBL_1_FLIP:
    for (bit[0] = 0, flip_mask[0] = 1, data_parity_block[0] = &data[*ssb[0]];
        bit[0] < *seb[0]; bit[0]++, flip_mask[0] <<= 1) {
        *data_parity_block[0] ^= flip_mask[0];
        mac_computations++;

        if (COMPARE_MACS(data, correct_data)) {
            return mac_computations;
        }

        *data_parity_block[0] ^= flip_mask[0];
    }
    goto *lbl_flip_break[1];
    LBL_1_FLIP_BREAK:

    *data_parity_block[1] ^= flip_mask[1];
}
goto *lbl_flip_break[2];
LBL_2_FLIP_BREAK:
*data_parity_block[2] ^= flip_mask[2];

```

```

*data_parity_block[2] ^= flip_mask[2];

LBL_2_FLIP:
for (bit[1] = 0, flip_mask[1] = 1, data_parity_block[1] = &data[*ssb[1]];
    bit[1] < *seb[1]; bit[1]++, flip_mask[1] <<= 1) {
    *data_parity_block[1] ^= flip_mask[1];

    // We also do the correction of single bit errors in this code
    // to check the correctness of the algorithm.
LBL_1_FLIP:
for (bit[0] = 0, flip_mask[0] = 1, data_parity_block[0] = &data[*ssb[0]];
    bit[0] < *seb[0]; bit[0]++, flip_mask[0] <<= 1) {
    *data_parity_block[0] ^= flip_mask[0];
    mac_computations++;

    if (COMPARE_MACS(data, correct_data)) {
        return mac_computations;
    }

    *data_parity_block[0] ^= flip_mask[0];
}
goto *lbl_flip_break[1];
LBL_1_FLIP_BREAK:

*data_parity_block[1] ^= flip_mask[1];
}
goto *lbl_flip_break[2];
LBL_2_FLIP_BREAK:
*data_parity_block[2] ^= flip_mask[2];

```

```

*data_parity_block[2] ^= flip_mask[2];

LBL_2_FLIP:
for (bit[1] = 0, flip_mask[1] = 1, data_parity_block[1] = &data[*ssb[1]];
    bit[1] < *seb[1]; bit[1]++, flip_mask[1] <<= 1) {
    *data_parity_block[1] ^= flip_mask[1];

    // We also do the correction of single bit errors in this code
    // to check the correctness of the algorithm.
LBL_1_FLIP:
for (bit[0] = 0, flip_mask[0] = 1, data_parity_block[0] = &data[*ssb[0]];
    bit[0] < *seb[0]; bit[0]++, flip_mask[0] <<= 1) {
    *data_parity_block[0] ^= flip_mask[0];
    mac_computations++;

    if (COMPARE_MACS(data, correct_data)) {
        return mac_computations;
    }

    *data_parity_block[0] ^= flip_mask[0];
}
goto *lbl_flip_break[1];
LBL_1_FLIP_BREAK:

*data_parity_block[1] ^= flip_mask[1];
}
goto *lbl_flip_break[2];
LBL_2_FLIP_BREAK:
*data_parity_block[2] ^= flip_mask[2];

```

```

*data_parity_block[2] ^= flip_mask[2];

LBL_2_FLIP:
for (bit[1] = 0, flip_mask[1] = 1, data_parity_block[1] = &data[*ssb[1]];
    bit[1] < *seb[1]; bit[1]++, flip_mask[1] <<= 1) {
    *data_parity_block[1] ^= flip_mask[1];

    // We also do the correction of single bit errors in this code
    // to check the correctness of the algorithm.
    LBL_1_FLIP:
    for (bit[0] = 0, flip_mask[0] = 1, data_parity_block[0] = &data[*ssb[0]];
        bit[0] < *seb[0]; bit[0]++, flip_mask[0] <<= 1) {
        *data_parity_block[0] ^= flip_mask[0];
        mac_computations++;

        if (COMPARE_MACS(data, correct_data)) {
            return mac_computations;
        }

        *data_parity_block[0] ^= flip_mask[0];
    }
    goto *lbl_flip_break[1];
    LBL_1_FLIP_BREAK:

    *data_parity_block[1] ^= flip_mask[1];
}
goto *lbl_flip_break[2];
LBL_2_FLIP_BREAK:
*data_parity_block[2] ^= flip_mask[2];

```

```

*data_parity_block[2] ^= flip_mask[2];

LBL_2_FLIP:
for (bit[1] = 0, flip_mask[1] = 1, data_parity_block[1] = &data[*ssb[1]];
    bit[1] < *seb[1]; bit[1]++, flip_mask[1] <<= 1) {
    *data_parity_block[1] ^= flip_mask[1];

    // We also do the correction of single bit errors in this code
    // to check the correctness of the algorithm.
    LBL_1_FLIP:
    for (bit[0] = 0, flip_mask[0] = 1, data_parity_block[0] = &data[*ssb[0]];
        bit[0] < *seb[0]; bit[0]++, flip_mask[0] <<= 1) {
        *data_parity_block[0] ^= flip_mask[0];
        mac_computations++;

        if (COMPARE_MACS(data, correct_data)) {
            return mac_computations;
        }

        *data_parity_block[0] ^= flip_mask[0];
    }
    goto *lbl_flip_break[1];
    LBL_1_FLIP_BREAK:

    *data_parity_block[1] ^= flip_mask[1];
}
goto *lbl_flip_break[2];
LBL_2_FLIP_BREAK:
*data_parity_block[2] ^= flip_mask[2];

```

```

*data_parity_block[2] ^= flip_mask[2];

LBL_2_FLIP:
for (bit[1] = 0, flip_mask[1] = 1, data_parity_block[1] = &data[*ssb[1]];
    bit[1] < *seb[1]; bit[1]++, flip_mask[1] <<= 1) {
    *data_parity_block[1] ^= flip_mask[1];

    // We also do the correction of single bit errors in this code
    // to check the correctness of the algorithm.
LBL_1_FLIP:
for (bit[0] = 0, flip_mask[0] = 1, data_parity_block[0] = &data[*ssb[0]];
    bit[0] < *seb[0]; bit[0]++, flip_mask[0] <<= 1) {
    *data_parity_block[0] ^= flip_mask[0];
    mac_computations++;

    if (COMPARE_MACS(data, correct_data)) {
        return mac_computations;
    }

    *data_parity_block[0] ^= flip_mask[0];
}
goto *lbl_flip_break[1];
LBL_1_FLIP_BREAK:

*data_parity_block[1] ^= flip_mask[1];
}
goto *lbl_flip_break[2];
LBL_2_FLIP_BREAK:
*data_parity_block[2] ^= flip_mask[2];

```



```

// The for outer PB loops are for parity bit input blocks with an even
// number of flips and therefore no change in the parity bits.
LBL_4_PB:
for (pb[3] = 0; pb[3] < CONST_NUM_PARITY_BITS; pb[3]++) {

    LBL_3_PB:
    for (pb[2] = 0; pb[2] <= pb[3]; pb[2]++) {

        LBL_2_PB:
        for (pb[1] = 0; pb[1] <= pb[2]; pb[1]++) {

            LBL_1_PB:
            for (pb[0] = 0; pb[0] <= pb[1]; pb[0]++) {

                goto *lbl_flip;

            LBL_8_FLIP:
            for (bit[7] = 0, flip_mask[7] = 1, data_parity_block[7] = &data[*ssb[7]];
                bit[7] < *seb[7]; bit[7]++, flip_mask[7] <<= 1) {
                *data_parity_block[7] ^= flip_mask[7];

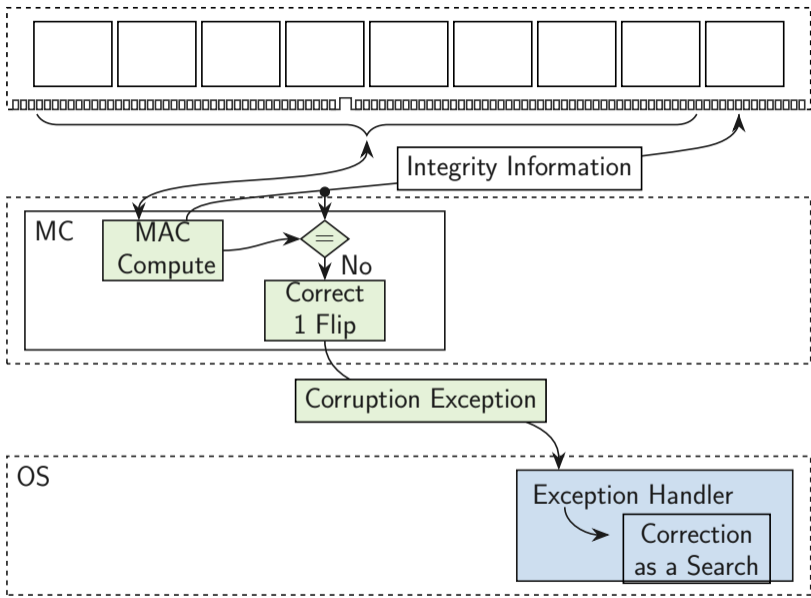
            LBL_7_FLIP:
            for (bit[6] = 0, flip_mask[6] = 1, data_parity_block[6] = &data[*ssb[6]];
                bit[6] < *seb[6]; bit[6]++, flip_mask[6] <<= 1) {
                *data_parity_block[6] ^= flip_mask[6];

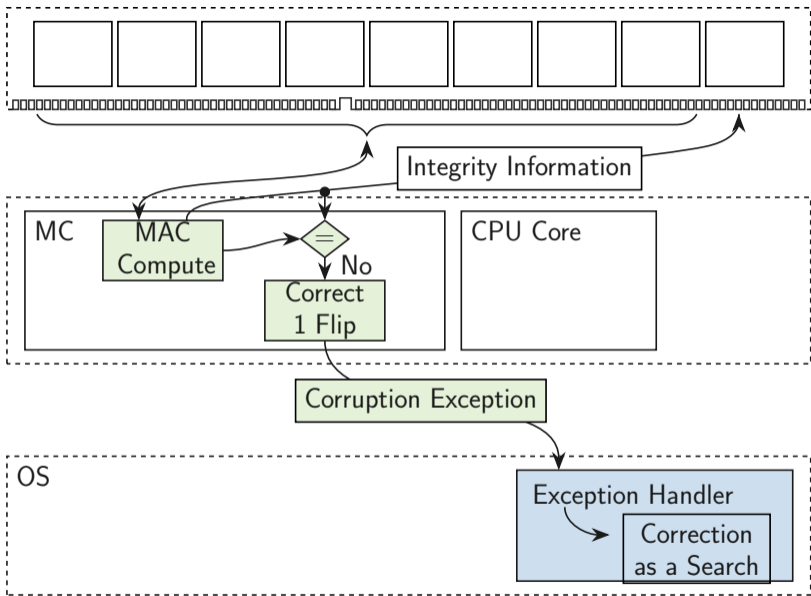
            LBL_6_FLIP:
            for (bit[5] = 0, flip_mask[5] = 1, data_parity_block[5] = &data[*ssb[5]];
                bit[5] < *seb[5]; bit[5]++, flip_mask[5] <<= 1) {

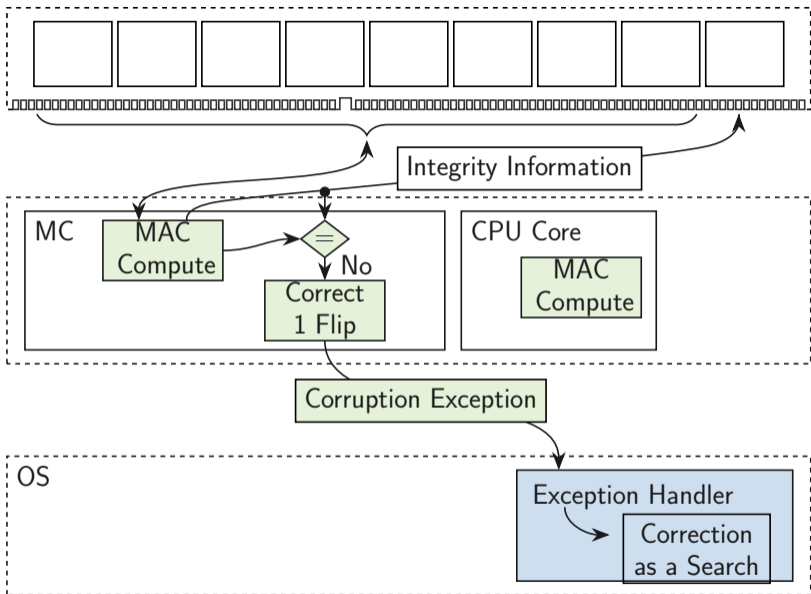
```

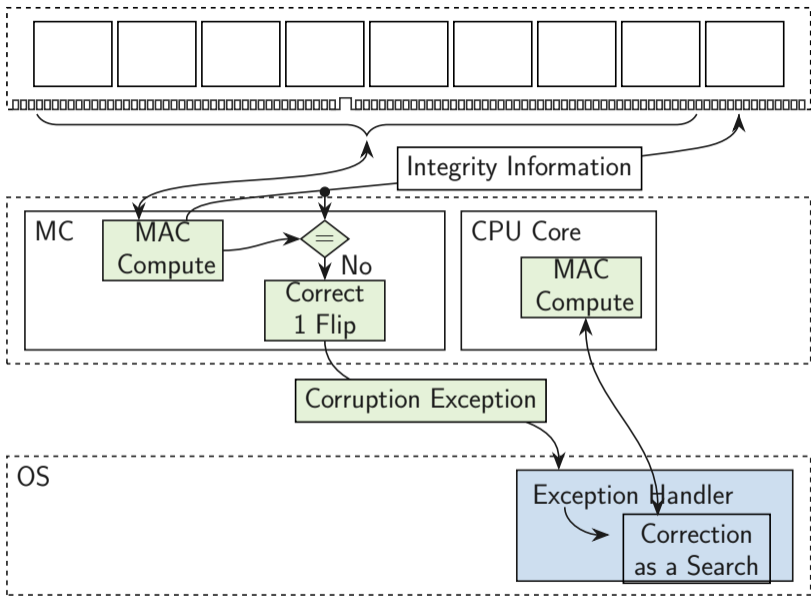
```
// define where to jump to in this loop to correct the number of bit
// flips currently tried.
// This safes a lot of space without loosing much performance.
void *lbl_flip;
void *lbl_flip_break[9], *lbl_pb_break[5];

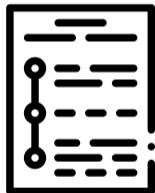
void *LBL_FLIP[9] = {
    NULL,           // let the array start with 1
    &&LBL_1_FLIP,
    &&LBL_2_FLIP,
    &&LBL_3_FLIP,
    &&LBL_4_FLIP,
    &&LBL_5_FLIP,
    &&LBL_6_FLIP,
    &&LBL_7_FLIP,
    &&LBL_8_FLIP
};
void *LBL_PB[5] = {
    NULL,
    &&LBL_1_PB,
    &&LBL_2_PB,
    &&LBL_3_PB,
    &&LBL_4_PB
};
```











Instruction	OPs	Description
<code>csi_mac</code>	ZMM rx rx	Calculates the MAC of the data (OP1) for the physical address (OP2) and writes it into OP3.
<code>csi_load</code>	rx ZMM rx	Reads from physical address (OP1) the raw corrupted data (OP2) and integrity information (OP3).
<code>csi_xchg</code>	rx ZMM rx ZMM	Writes new data (OP4) into the physical address (OP1) if the old data (OP2) and MAC (OP3) are unchanged at physical address.

- Calculate average MAC comp. per number of bit flips





- Calculate average MAC comp. per number of bit flips
 - e.g. 1.21×10^{11} for 7 bit flips
- 6.87 min



- Calculate average MAC comp. per number of bit flips
 - e.g. 1.21×10^{11} for 7 bit flips
 - 6.87 min
- Run loop n iterations on real hardware



- Calculate average MAC comp. per number of bit flips
 - e.g. 1.21×10^{11} for 7 bit flips
 - 6.87 min
- Run loop n iterations on real hardware
 - Out-of-order execution enables parallelism

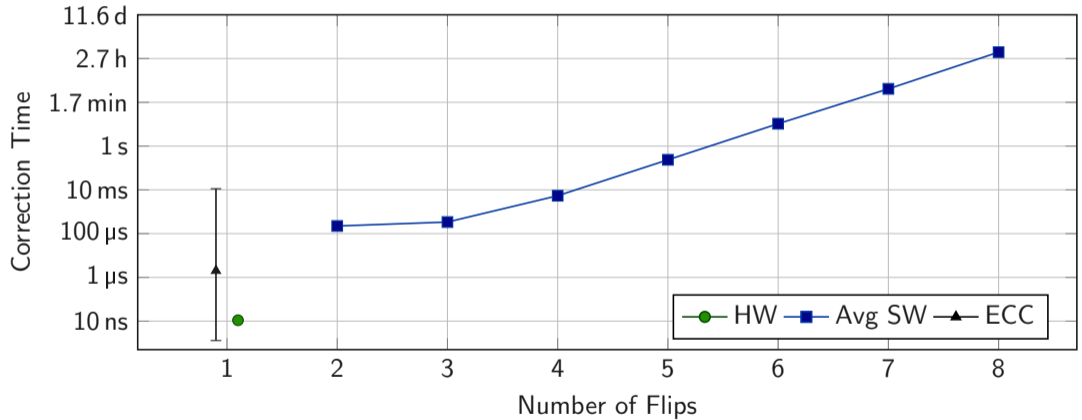


- Calculate average MAC comp. per number of bit flips
 - e.g. 1.21×10^{11} for 7 bit flips
 - 6.87 min
- Run loop n iterations on real hardware
 - Out-of-order execution enables parallelism

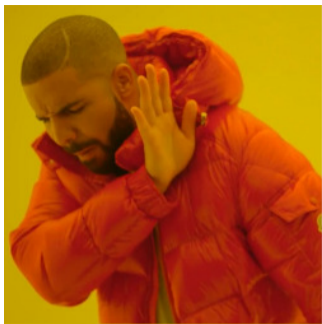
Apple M1	Intel i7-1165G7	AMD 5800X
----------	-----------------	-----------

2.30 min	9.11 min	10.0 min
----------	----------	----------

Correction Time Estimation



Bits in the Mac Can Also Flip



imgflip.com





$$\log_2\left(\sum_{k=0}^d \binom{n}{k}\right)$$



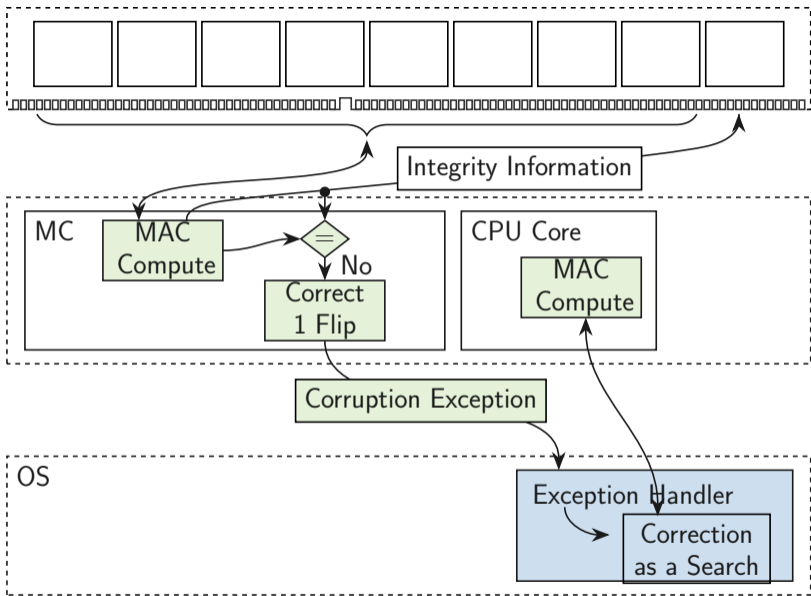
$$56 \text{ bit} - \log_2\left(\sum_{k=0}^3 \binom{56}{k}\right) = 41.2 \text{ bit}$$

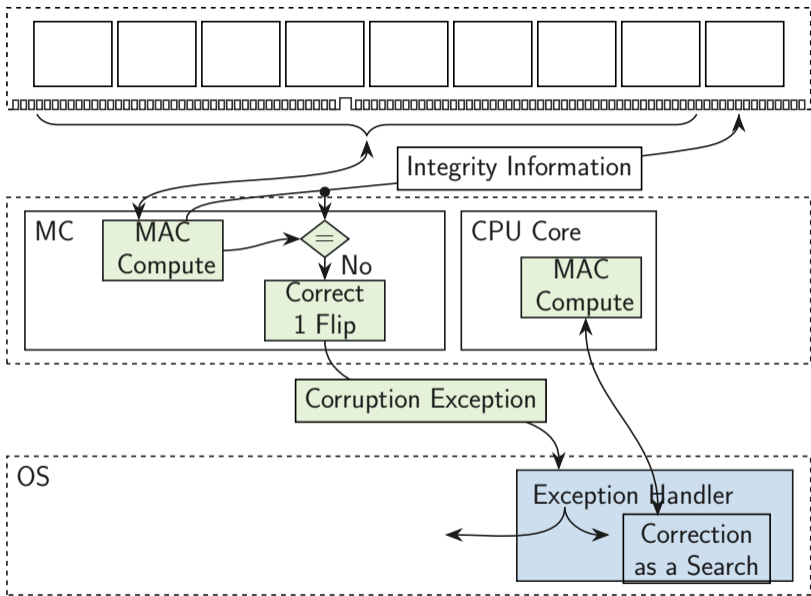


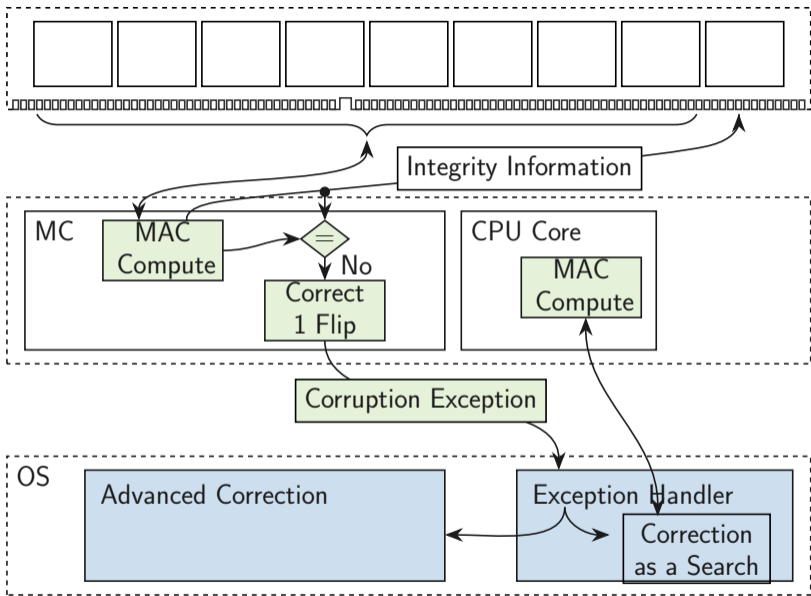
$$56 \text{ bit} - \log_2\left(\sum_{k=0}^3 \binom{56}{k}\right) = 41.2 \text{ bit}$$

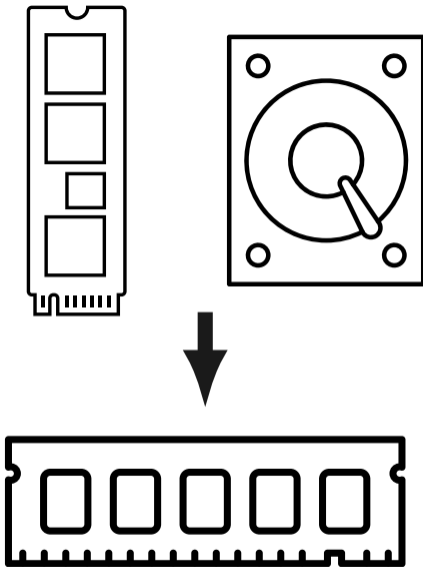
Data Flips	$\log_2(p)$	d	MAC Strength
5	26.0	3	41.2
6	31.5	2	45.4
7	38.8	1	50.2
8	42.4	0	56.0

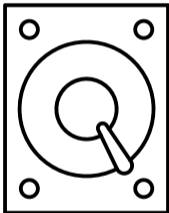
Software Can be Smarter

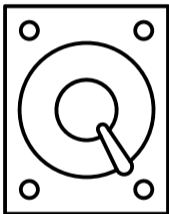




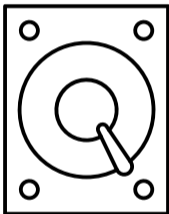




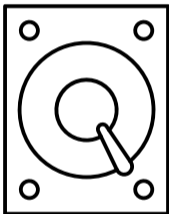




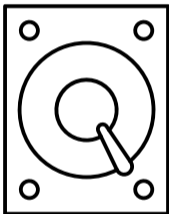
- Application Binaries



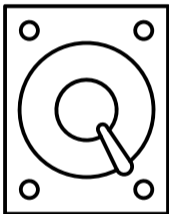
- Application Binaries
- Shared Libraries



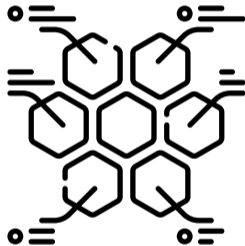
- Application Binaries
- Shared Libraries
- Memory Mapped Files



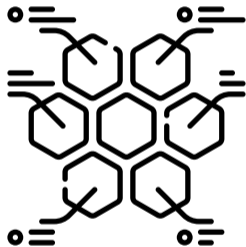
- Application Binaries
- Shared Libraries
- Memory Mapped Files
- Page Cache



- Application Binaries
- Shared Libraries
- Memory Mapped Files
- Page Cache
- Kernel Itself

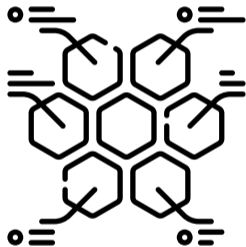


Kernel data with **known** structures



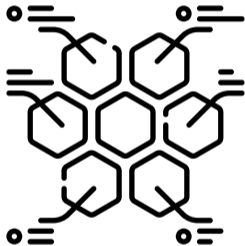
Kernel data with **known** structures

- Kernel code



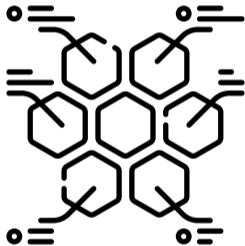
Kernel data with **known** structures

- Kernel code
- Page tables



Kernel data with **known** structures

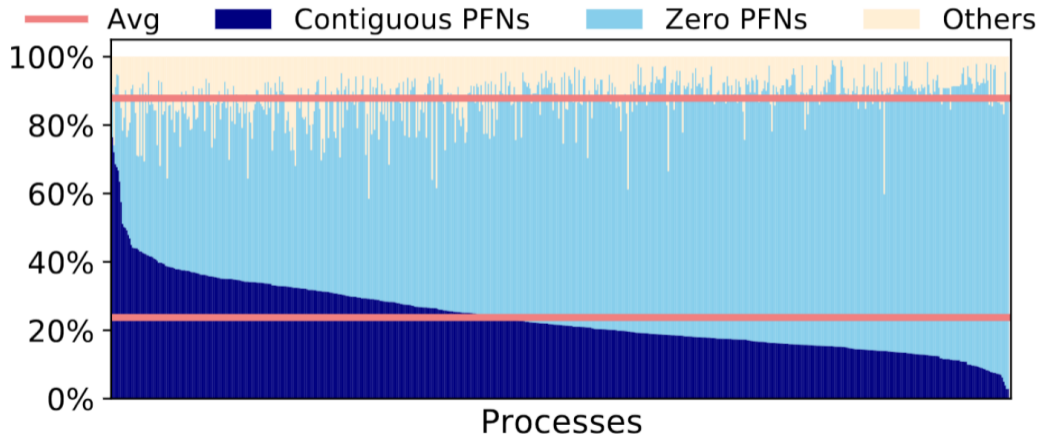
- Kernel code
- Page tables
- Task structs

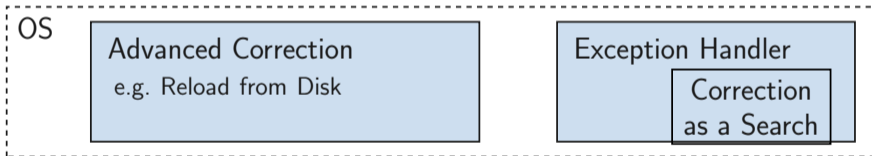


Kernel data with **known** structures

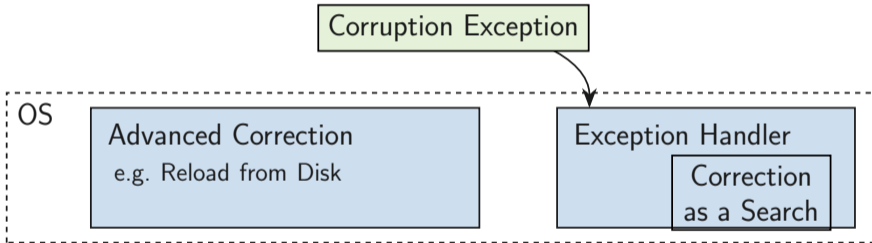
- Kernel code
- **Page Tables**
- Task structs

PT-Guard [7]

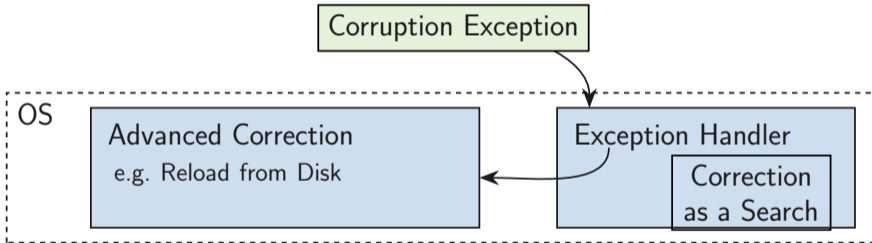




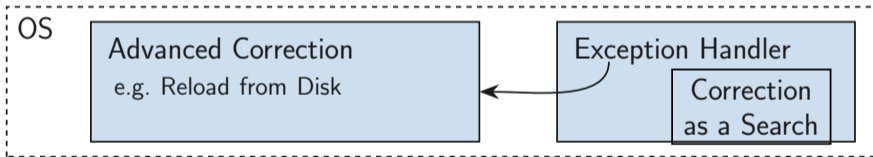
Nesting Detection



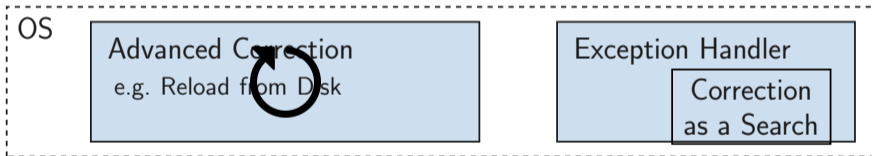
Nesting Detection



Nesting Detection



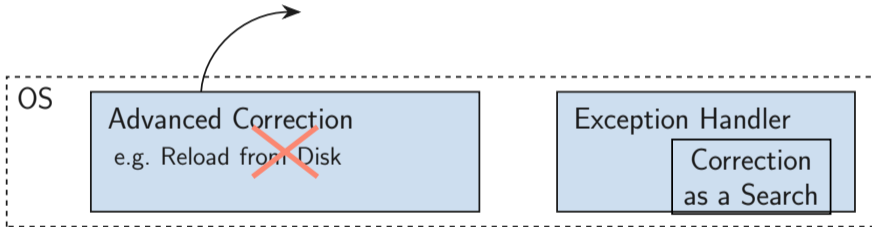
Nesting Detection



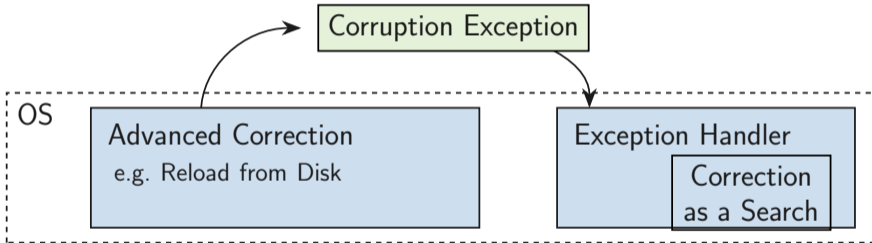
Nesting Detection



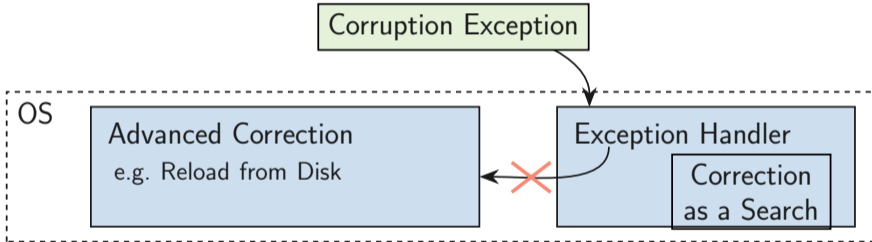
Nesting Detection



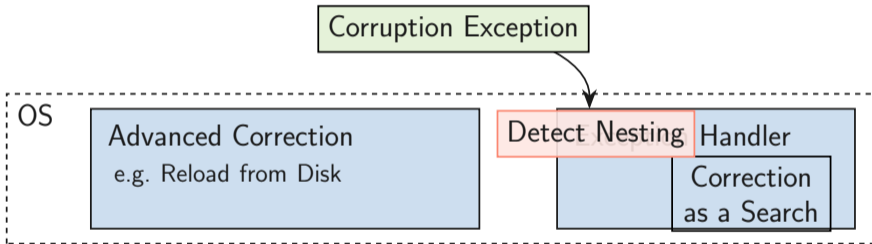
Nesting Detection



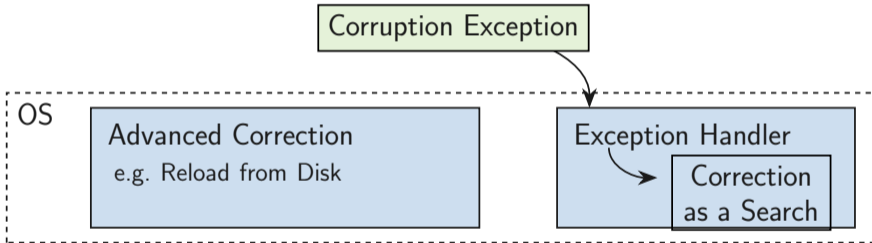
Nesting Detection



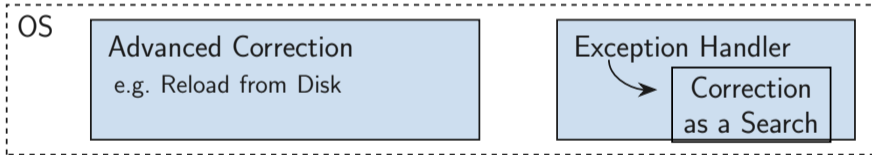
Nesting Detection

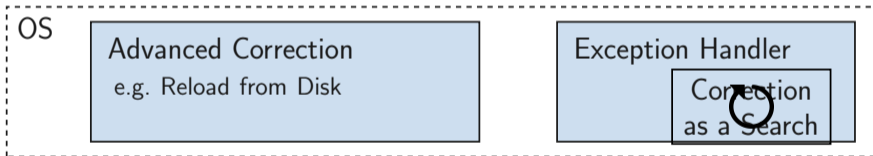


Nesting Detection



Nesting Detection






```
1 corruption_exception_handler ( corruption_address )
2 {
3     if ( nested_bit_set () )
4     {
5         enable_interrupts ();
6         error_correction_as_a_search ( corruption_address );
7     }
8     else
9     {
10        set_nested_bit ();
11        enable_interrupts ();
12        advanced_error_correction ( corruption_address );
13        clear_nested_bit ();
14    }
15 }
```



- Nested Bit



- Nested Bit
 - **Per Process**



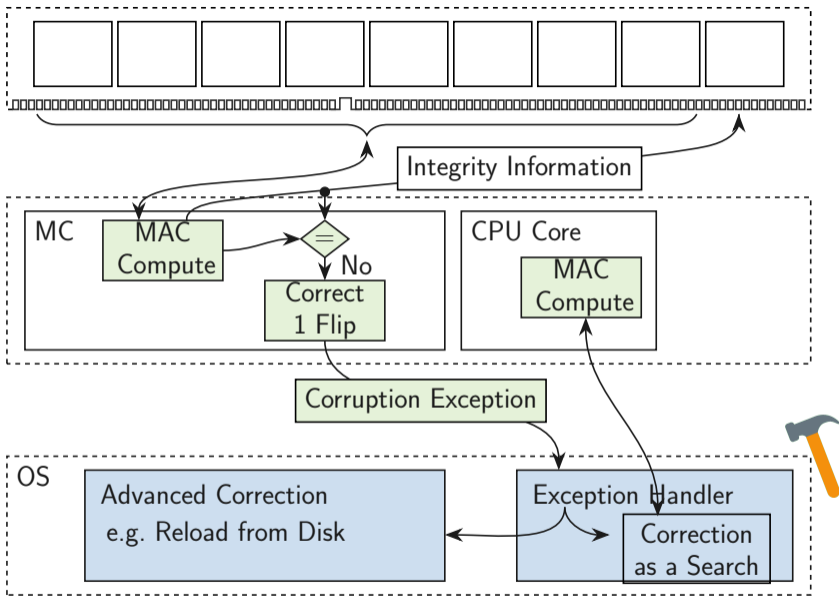
- Nested Bit
 - **Per Process**
 - In **Register**

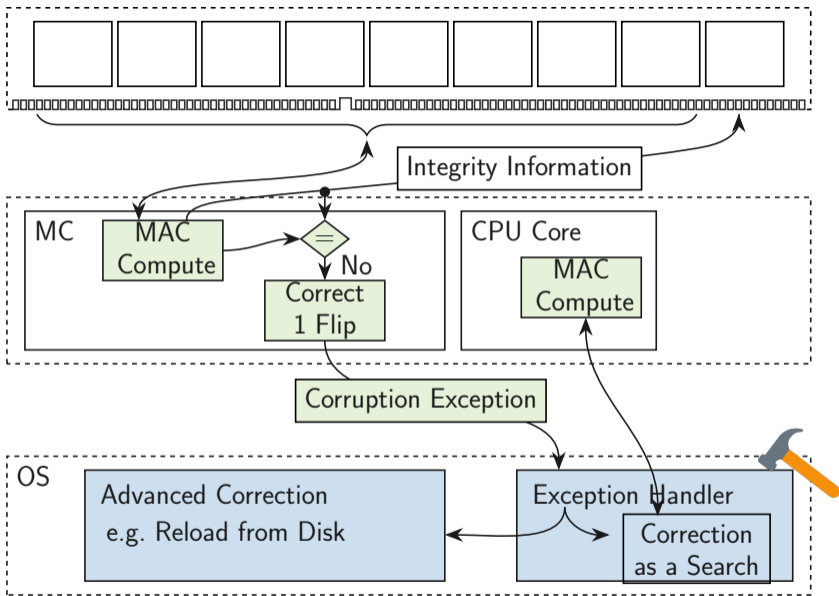


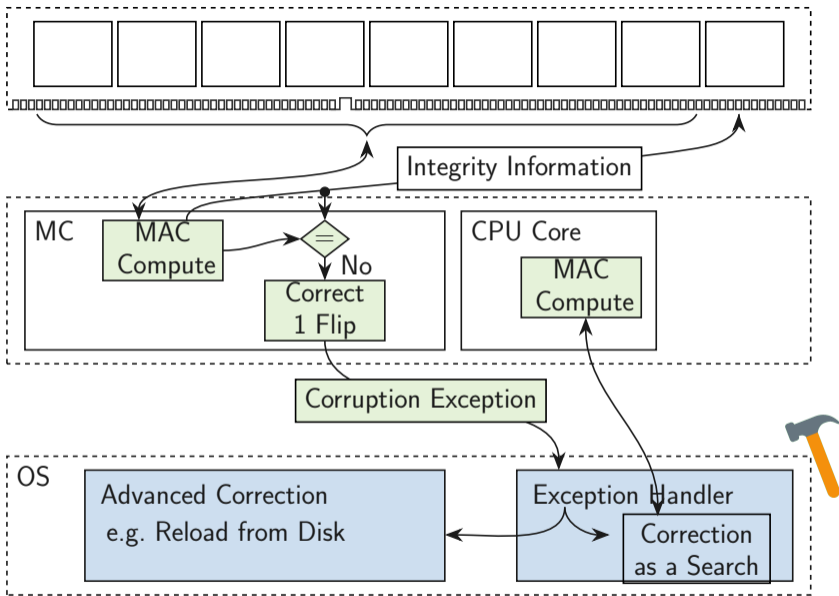
- Nested Bit
 - Per Process
 - In Register

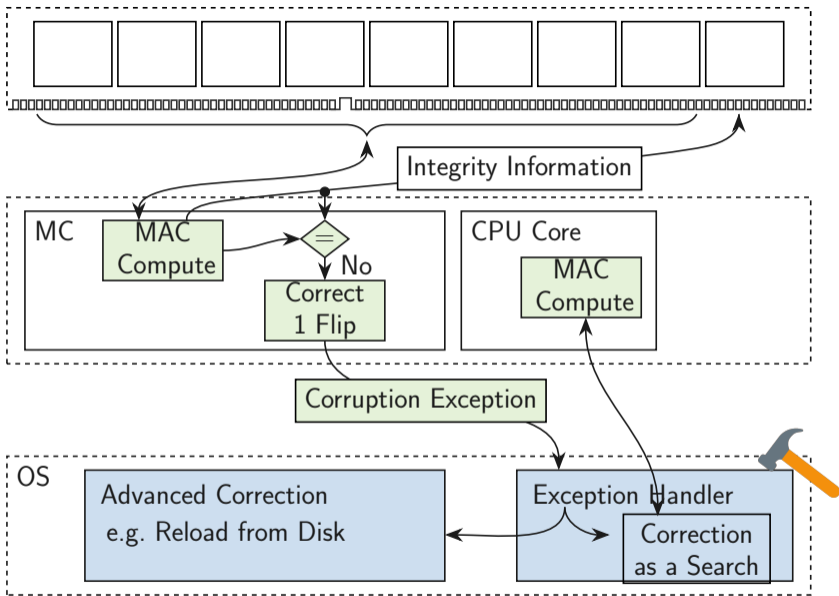
→ High bit in CR3 register

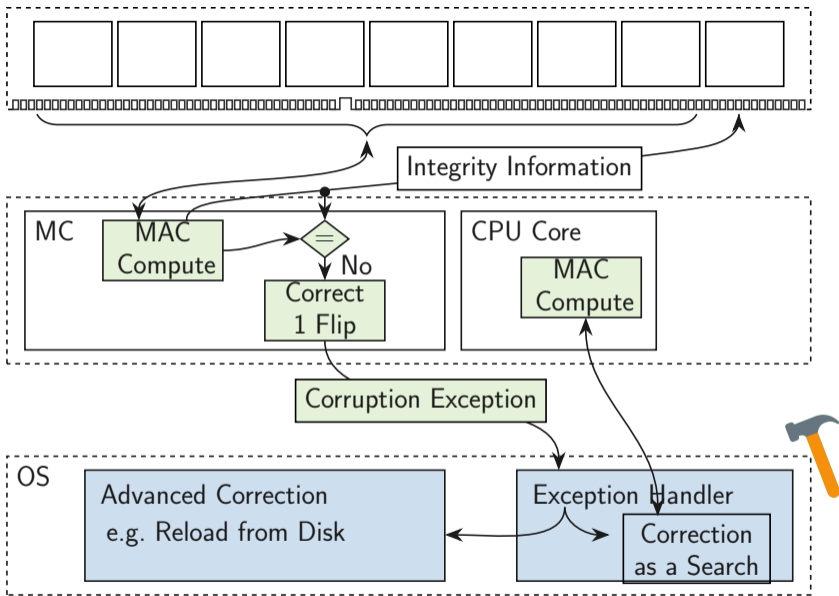


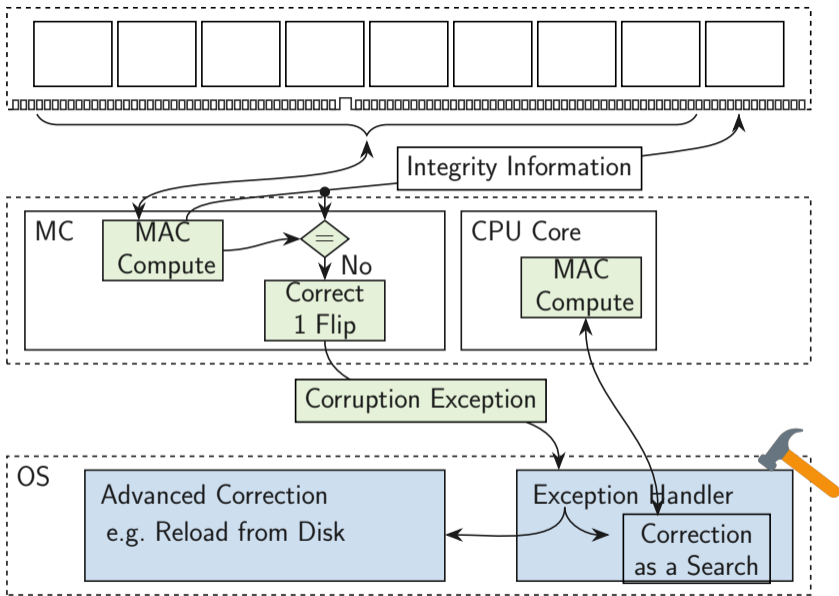


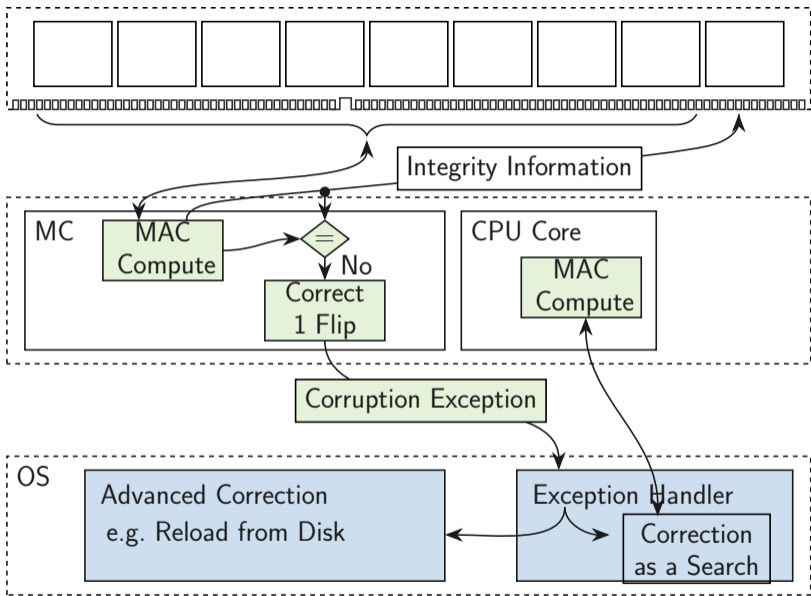


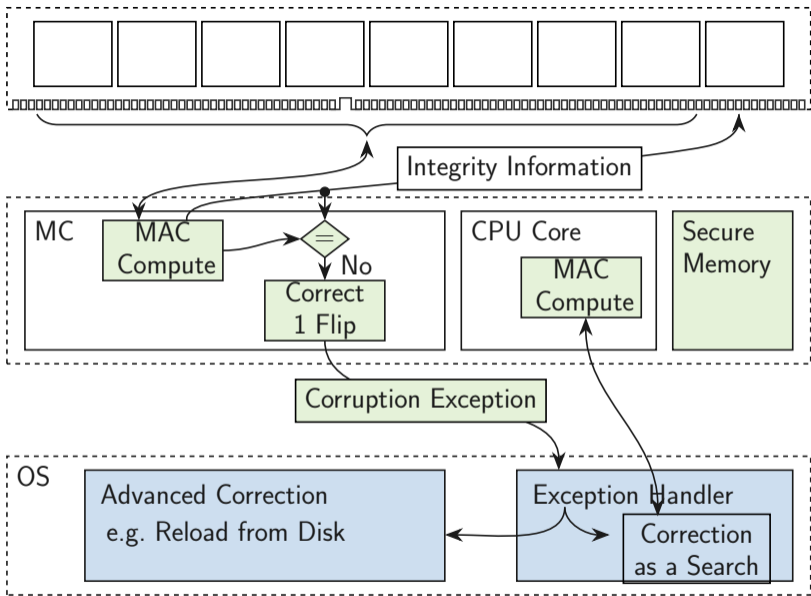


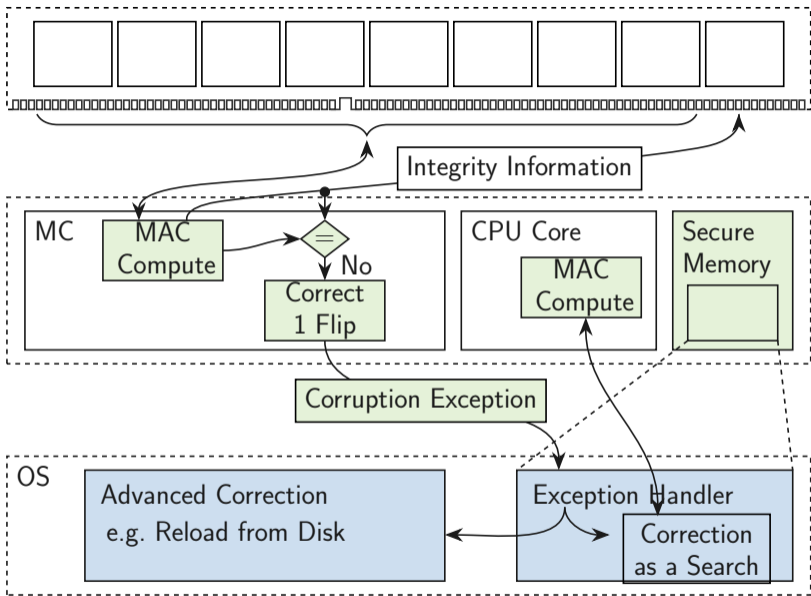
















- On-die SRAM like CPU caches



- On-die SRAM like CPU caches
- **0x4000** bytes large



- On-die SRAM like CPU caches
- **0x4000** bytes large
 - IDT, GDT



- On-die SRAM like CPU caches
- **0x4000** bytes large
 - IDT, GDT
 - Exception handler



- On-die SRAM like CPU caches
- **0x4000** bytes large
 - IDT, GDT
 - Exception handler
 - Correction as a search code





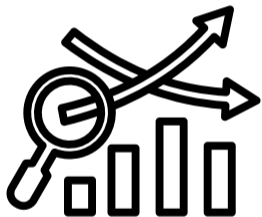
- Stack could also be corrupted



- Stack could also be corrupted
- Increase stack pointer by 256 B (512 B)

Evaluation





- Implemented CSI:Rowhammer in **gem5**



- Implemented CSI:Rowhammer in **gem5**
- Modified **Linux** kernel

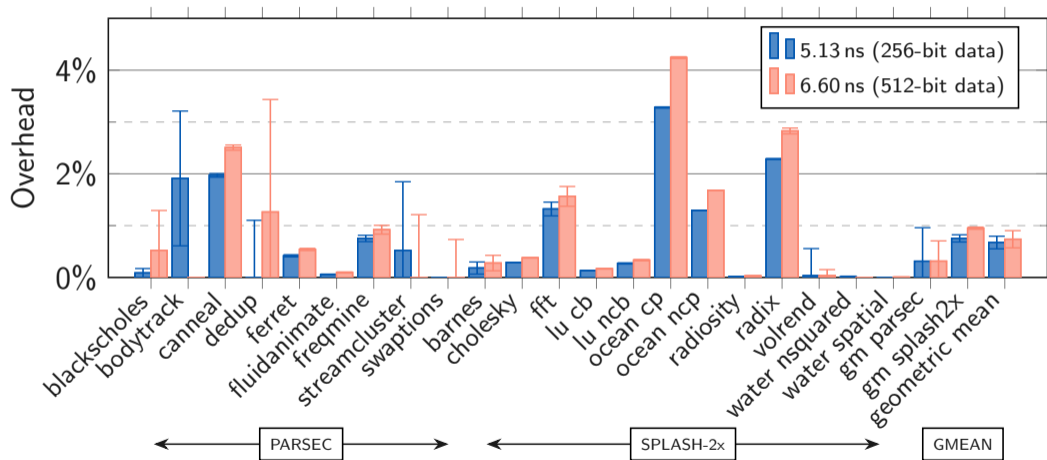


- Implemented CSI:Rowhammer in **gem5**
- Modified **Linux** kernel
- Evaluated **correct functionality**



- Implemented CSI:Rowhammer in **gem5**
- Modified **Linux** kernel
- Evaluated **correct functionality**
- Evaluated **performance overhead**

Performance Overhead







- **Approximate Equality**

- **Approximate Equality**

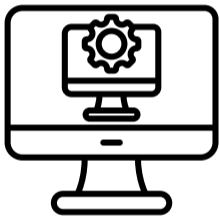
Data Flips	$\log_2(p)$	d	MAC Strength
5	26.0	3	41.2
6	31.5	2	45.4
7	38.8	1	50.2
8	42.4	0	56.0

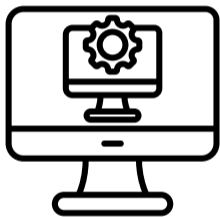
- **Approximate Equality**
- Silent Data Corruption rate less than once per 10^9 billion years.

Data Flips	$\log_2(p)$	d	MAC Strength
5	26.0	3	41.2
6	31.5	2	45.4
7	38.8	1	50.2
8	42.4	0	56.0

- **Approximate Equality**
- Silent Data Corruption rate less than once per 10^9 billion years.
- Rowhammer second preimage after one year: $9.75 \cdot 10^{-5}\%$

Data Flips	$\log_2(p)$	d	MAC Strength
5	26.0	3	41.2
6	31.5	2	45.4
7	38.8	1	50.2
8	42.4	0	56.0

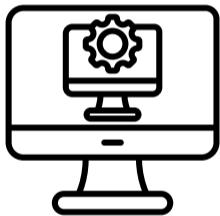




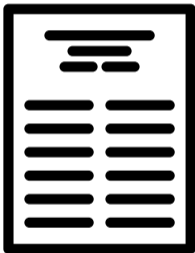
- Only **host support** required



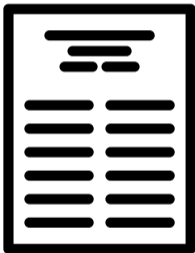
- Only **host support** required
- Guest support optional



- Only **host support** required
- Guest support optional
- Makes upgrading easy



- **Correct** errors from a faulty DIMM connection



- **Correct** errors from a faulty DIMM connection
- Less than **0.01%** CPU die area overhead

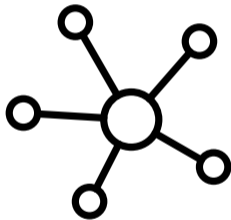


- **Correct** errors from a faulty DIMM connection
- Less than **0.01%** CPU die area overhead
- **Compatible** with TRR

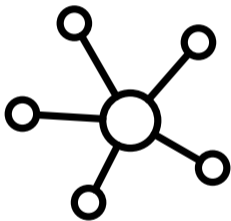


- **Correct** errors from a faulty DIMM connection
- Less than **0.01%** CPU die area overhead
- **Compatible** with TRR
- **Compatible** with hardware memory encryption

The Possibilities

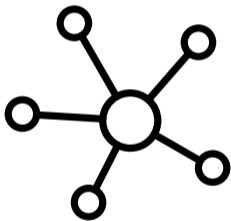


Correction in **software** opens up many new **possibilities**



Correction in **software** opens up many new **possibilities**

- Data correction as a service (DCaaS)



Correction in **software** opens up many new **possibilities**

- Data correction as a service (DCaaS)
- Correction by ChatGPT™

CSI:Rowhammer – Correction by ChatGPT

MA

Find the error in "Testmessoge" so that is has the following SHA512 hash
"9167bf29600fea2cb1ba4e21e2f311bf5c2641f0cd5cb5bec948177264ff
78e2bc346216427b75e75fe67e842ef3085ac303b8111a37adeaf977b2b
269ada636"



CSI:Rowhammer – Correction by ChatGPT



It is not possible for the string "Testmessoge" to have the SHA512 hash "9167bf29600fea2cb1ba4e21e2f311bf5c2641f0cd5cb5bec948177264ff78e2bc346216427b75e75fe67e842ef3085ac303b8111a37adeaf977b2b269ada636". This is because the SHA512 hash function is a one-way function, meaning that it is not possible to determine the input that produced a given hash. Therefore, it is not possible to determine the error in the input string "Testmessoge" based on the given hash value.



CSI:Rowhammer – Correction by ChatGPT

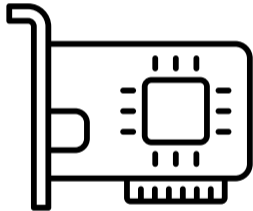
MA



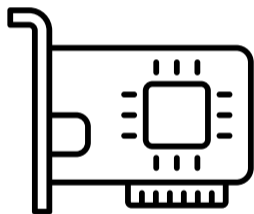
Can just brute force the correct answer for me?



This content may violate our [content policy](#). If you believe this to be in error, please [submit your feedback](#) — your input will aid our research in this area.

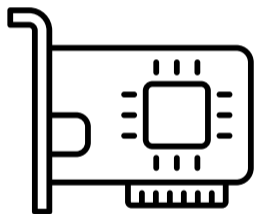


Hardware acceleration



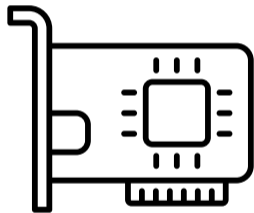
Hardware acceleration

- Ice Lake SP Xeon Server die size: 628 mm²



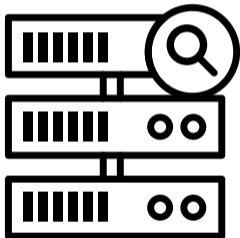
Hardware acceleration

- Ice Lake SP Xeon Server die size: 628 mm²
- Fits roughly 500 000 QARMA blocks

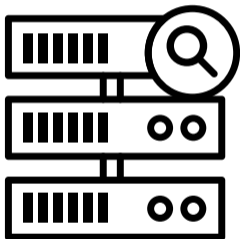


Hardware acceleration

- Ice Lake SP Xeon Server die size: 628 mm²
- Fits roughly 500 000 QARMA blocks
- 8 bit flips correction: 5.44 h → 40 ms



- Multi-bit flips not happening often



- Multi-bit flips not happening often
- **Specialized** data correction nodes



Performance impact caused by MAC computation



Performance impact caused by MAC computation

- Transient execution with unverified data



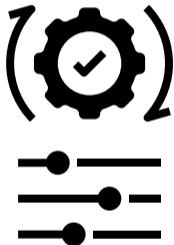
Performance impact caused by MAC computation

- Transient execution with unverified data
- Can be made **secure** [8, 4]

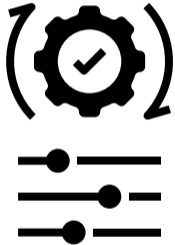


Performance impact caused by MAC computation

- Transient execution with unverified data
 - Can be made **secure** [8, 4]
- **No** performance impact

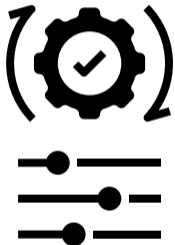


Idea:



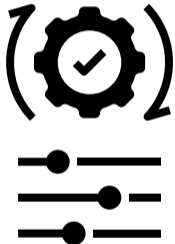
Idea:

- DRAM comes with ECC



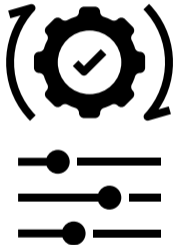
Idea:

- DRAM comes with ECC
- Reduce voltage and correct potential flips



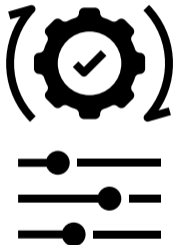
Idea:

- DRAM comes with ECC
- Reduce voltage and correct potential flips
- ✗ No security benefit from ECC

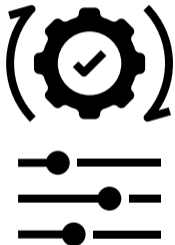


Idea:

- DRAM comes with ECC
- Reduce voltage and correct potential flips
- ✗ No security benefit from ECC
 - ✗ Double bit flips uncorrectable

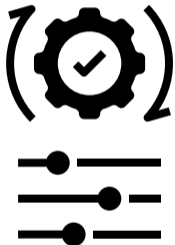


Better Idea:



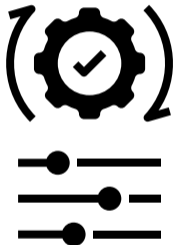
Better Idea:

- DRAM comes with CSI:Rowhammer



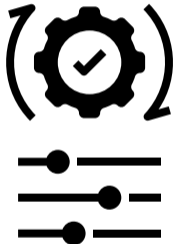
Better Idea:

- DRAM comes with CSI:Rowhammer
- Reduce voltage and correct potential flips



Better Idea:

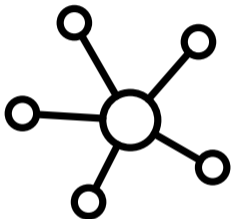
- DRAM comes with CSI:Rowhammer
- Reduce voltage and correct potential flips
- ✓ Security **always** guaranteed

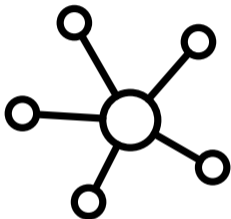


Better Idea:

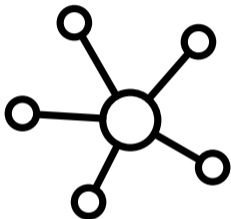
- DRAM comes with CSI:Rowhammer
- Reduce voltage and correct potential flips
- ✓ Security **always** guaranteed
- ✓ Too aggressive optimization hurts performance

Conclusion

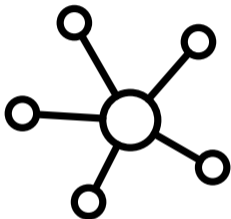




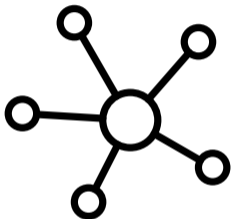
- RowPress and Rowhammer effects can **overlap**



- RowPress and Rowhammer effects can **overlap**
- Rowhammer mitigations focusing on characteristics can **never** guarantee security



- RowPress and Rowhammer effects can **overlap**
- Rowhammer mitigations focusing on characteristics can **never** guarantee security
- Using a MAC to ensure **data integrity** is viable



- RowPress and Rowhammer effects can **overlap**
- Rowhammer mitigations focusing on characteristics can **never** guarantee security
- Using a MAC to ensure **data integrity** is viable
- Software based correction opens up many **possibilities**



CSI:Rowhammer & PressHammer

Jonas Juffinger Lukas Lamster Sudheendra Raghav Neela Andreas Kogler
Martin Heckel Moritz Lipp Lukas Schwarz Maria Eichlseder Florian
Adamsky Daniel Gruss

2024-07-16

✉ jonas.juffinger@iaik.tugraz.at

🐦 [NotImaginary_](#)

🌐 www.jonasjuffinger.com

IAIK, Graz University of Technology, Austria



References

- [1] Roberto Avanzi. The QARMA Block Cipher Family: Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes. In: IACR Transactions on Symmetric Cryptology 2017.1 (2017), pp. 4–44.
- [2] Daniel Gruss et al. Another Flip in the Wall of Rowhammer Defenses. In: S&P. 2018.

- [3] Yoongu Kim et al. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In: ISCA. 2014.
- [4] Tamara Silbergleit Lehman et al. PoisonIvy: Safe speculation for secure memory. In: MICRO. 2016.
- [5] Haocong Luo et al. RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In: ISCA. 2023.
- [6] Peter Pessl et al. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: USENIX Security. 2016.
- [7] Anish Saxena et al. PT-Guard: Integrity-Protected Page Tables to Defend Against Breakthrough Rowhammer Attacks. In: DSN. 2023.

- [8] Weidong Shi et al. Authentication Control Point and Its Implications For Secure Processor Design. In: MICRO. 2006.