CSI:Rowhammer

Cryptographic Security and Integrity against Rowhammer

About me

Jonas Juffinger

PhD Student at IAIK – TU Graz

Hardware Fault Attacks from Software

Microarchitectural Security

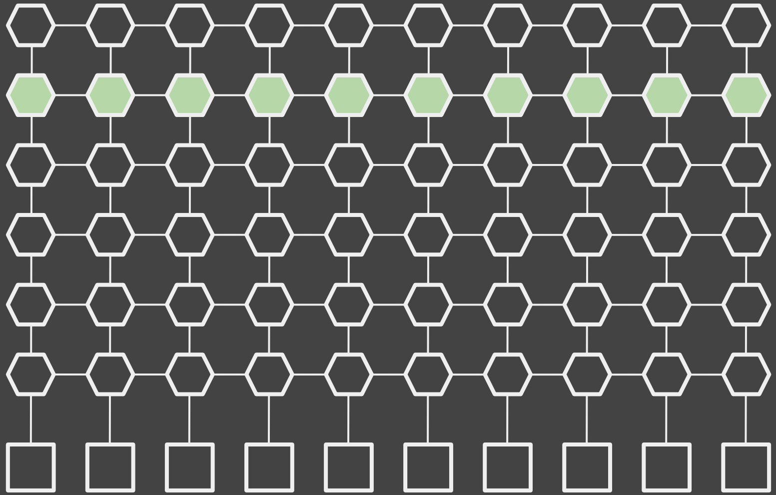
Side Channels

Secure and Energy Efficient Computing


UNSOLVED

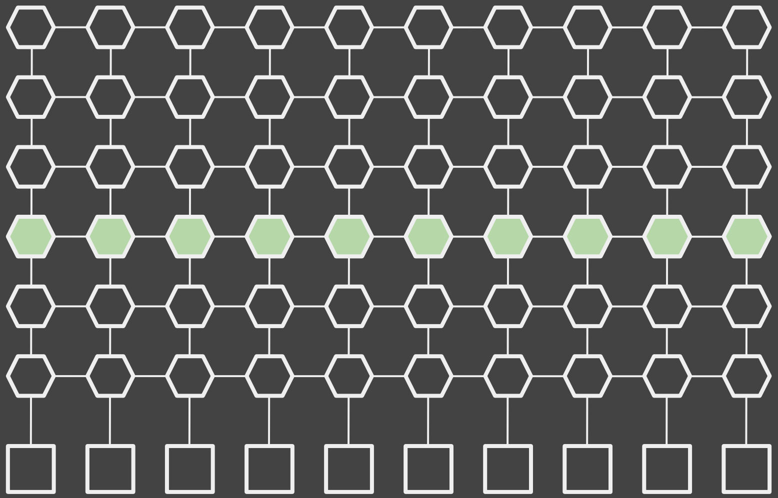
Rowhammer

```
while (1)
{
  *aggressor1;
  *aggressor2;
  clflush(aggressor1);
  clflush(aggressor2);
}
```



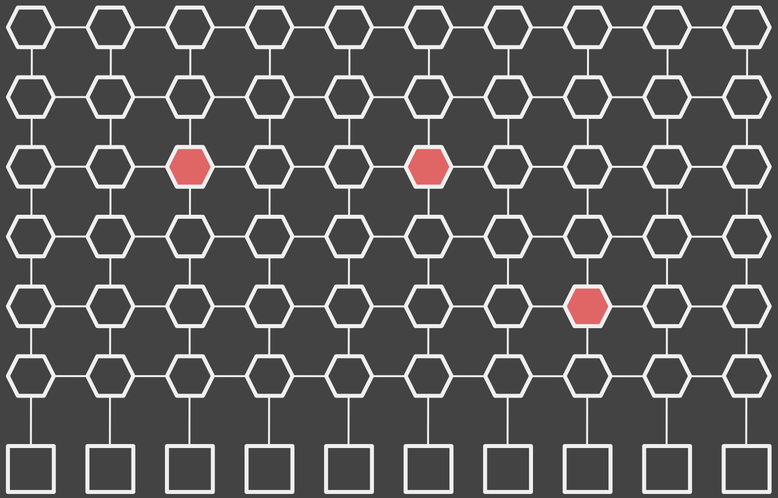
Rowhammer

```
while (1)
{
  *aggressor1;
  *aggressor2;
  clflush(aggressor1);
  clflush(aggressor2);
}
```



Rowhammer

```
while (1)
{
  *aggressor1;
  *aggressor2;
  clflush(aggressor1);
  clflush(aggressor2);
}
```



Demo Video

https://youtu.be/TJJxhcKyM-w?t=844

ATTACKS

Privilege Escalation

Browser Sandbox Escape

Virtual Machine Escape

Over the Network

Read Cryptographic Keys

Build Kernel Spectre Gadgets

...

MITIGATIONS

ECC / Chipkill

Detection in Software

Physical Isolation

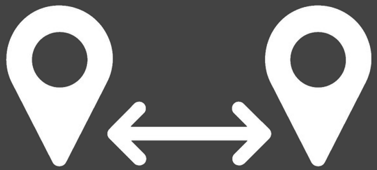
Additional Refreshes

Improved Physical Cell Layout

...

Mitigations Focus on the Characteristics of Rowhammer

Characteristics

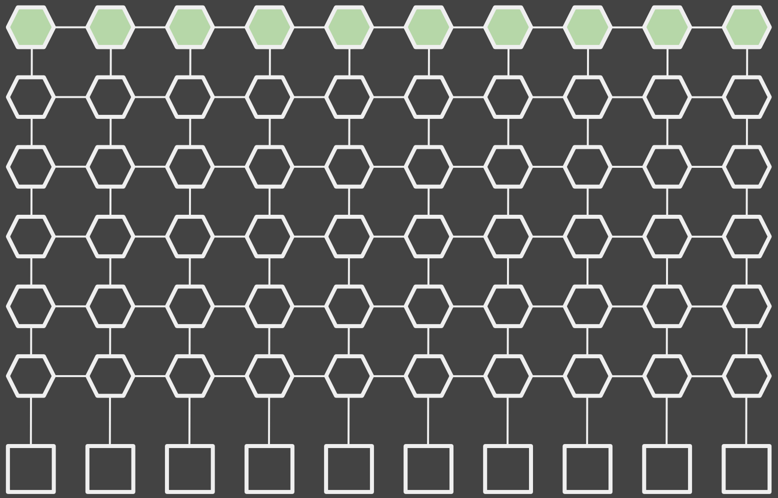


Infrequent Detectable Distance 1

These Characteristics are incomplete

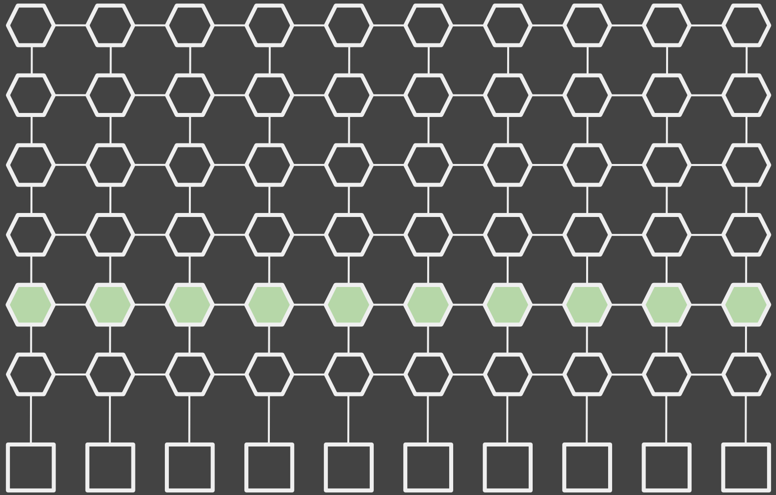
Half-Double Rowhammer

```
while (1)
{
  *aggressor1;
  *aggressor2;
  clflush(aggressor1);
  clflush(aggressor2);
}
```



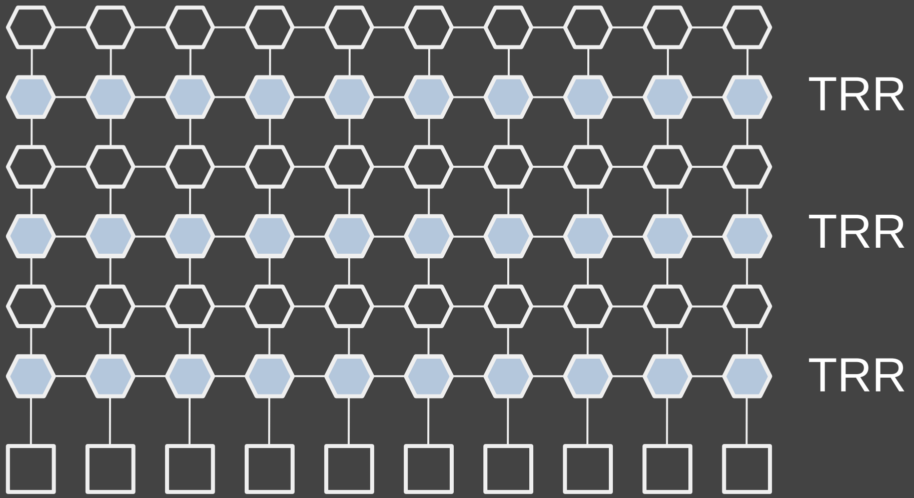
Half-Double Rowhammer

```
while (1)
{
  *aggressor1;
  *aggressor2;
  clflush(aggressor1);
  clflush(aggressor2);
}
```



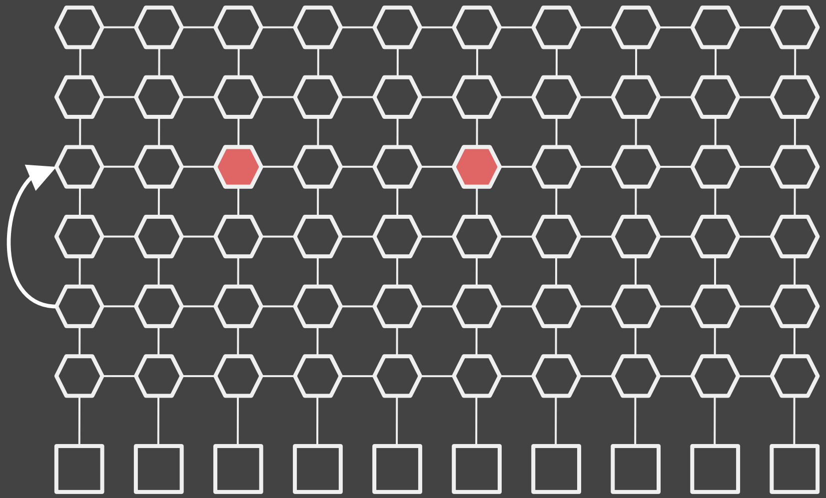
Half-Double Rowhammer

```
while (1)
{
  *aggressor1;
  *aggressor2;
  clflush(aggressor1);
  clflush(aggressor2);
}
```



Half-Double Rowhammer

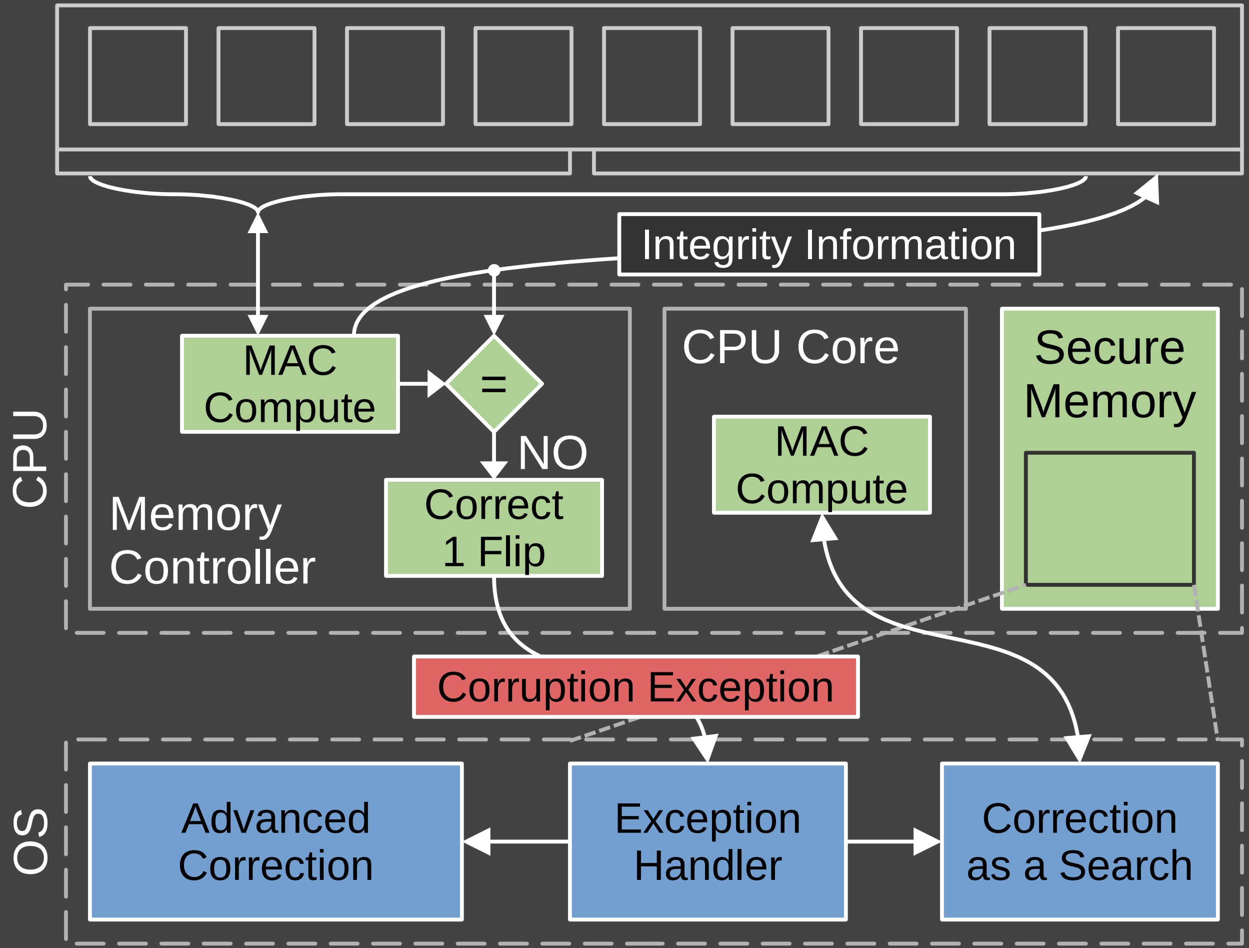
```
while (1)
{
  *aggressor1;
  *aggressor2;
  clflush(aggressor1);
  clflush(aggressor2);
}
```



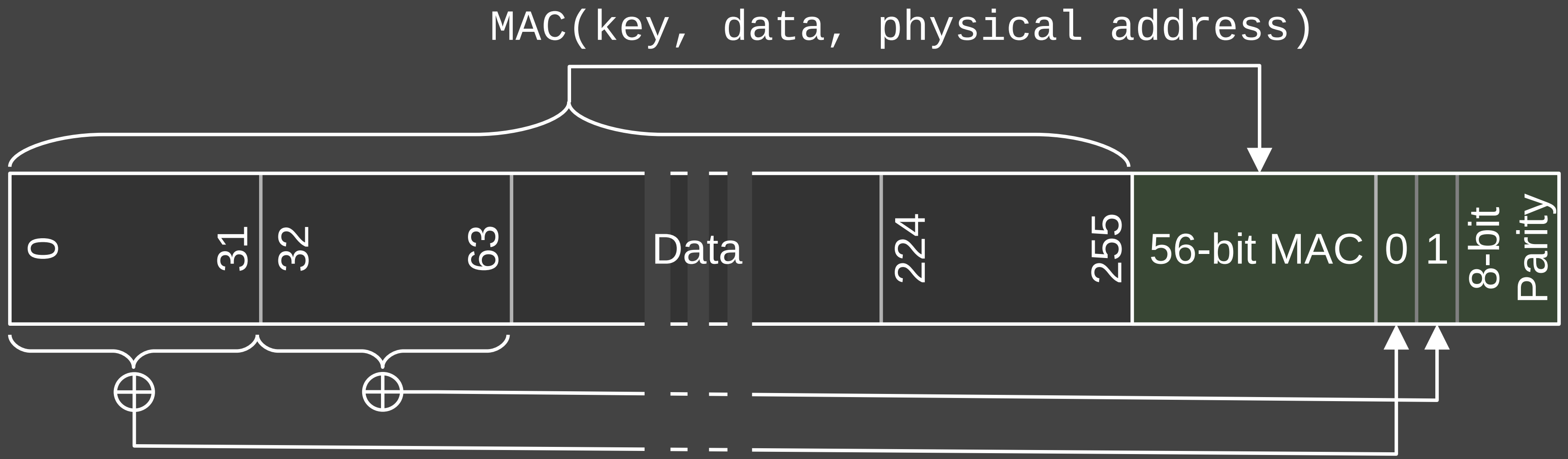
We have to rethink Rowhammer mitigations

Generic approach to data integrity protection

CSI:Rowhammer

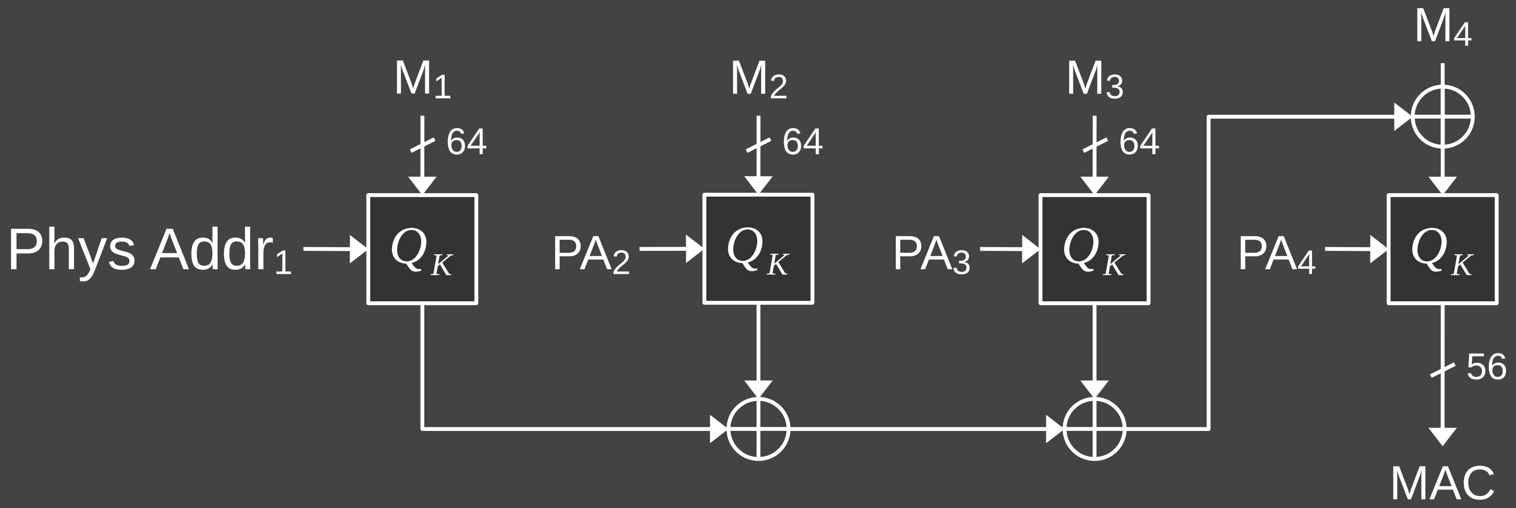

Integrity Information
CPU
MAC
Compute
=
NO
Correct
1 Flip
Memory
Controller
CPU Core
MAC
Compute
Secure
Memory
Corruption Exception
OS
Advanced
Correction
Exception
Handler
Correction
as a Search

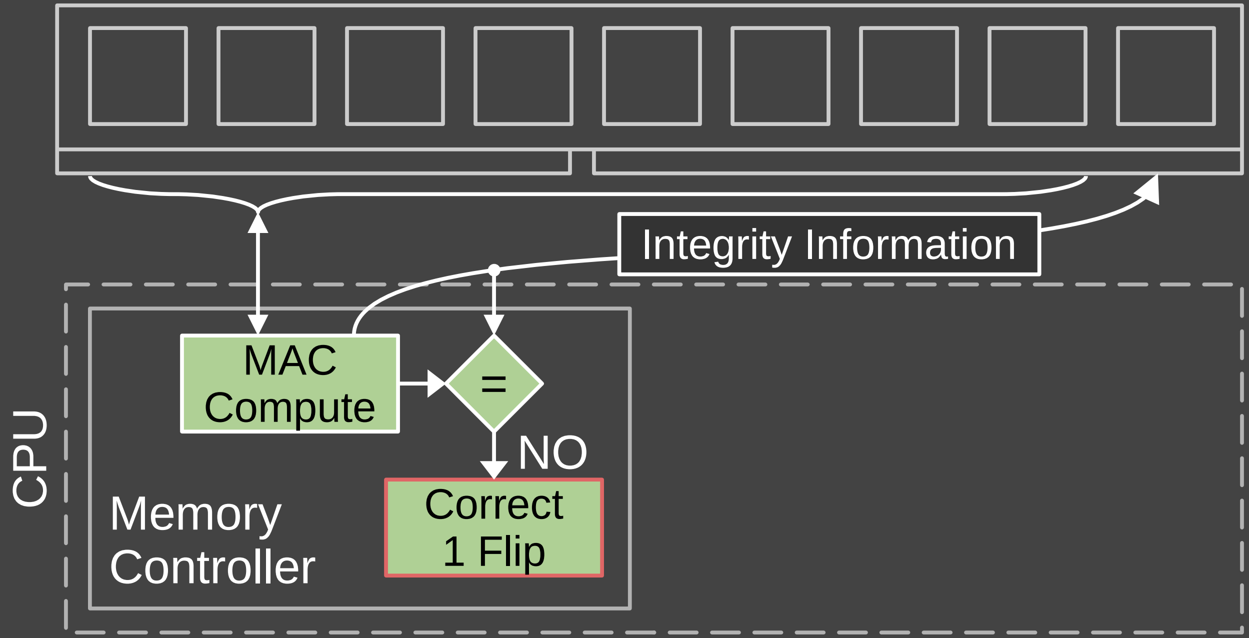
Integrity Information



Compute Message Authentication Code (MAC)

5 ns




Integrity Information
CPU
MAC
Compute
=
NO
Memory
Controller
Correct
1 Flip

Bit Flip Correction

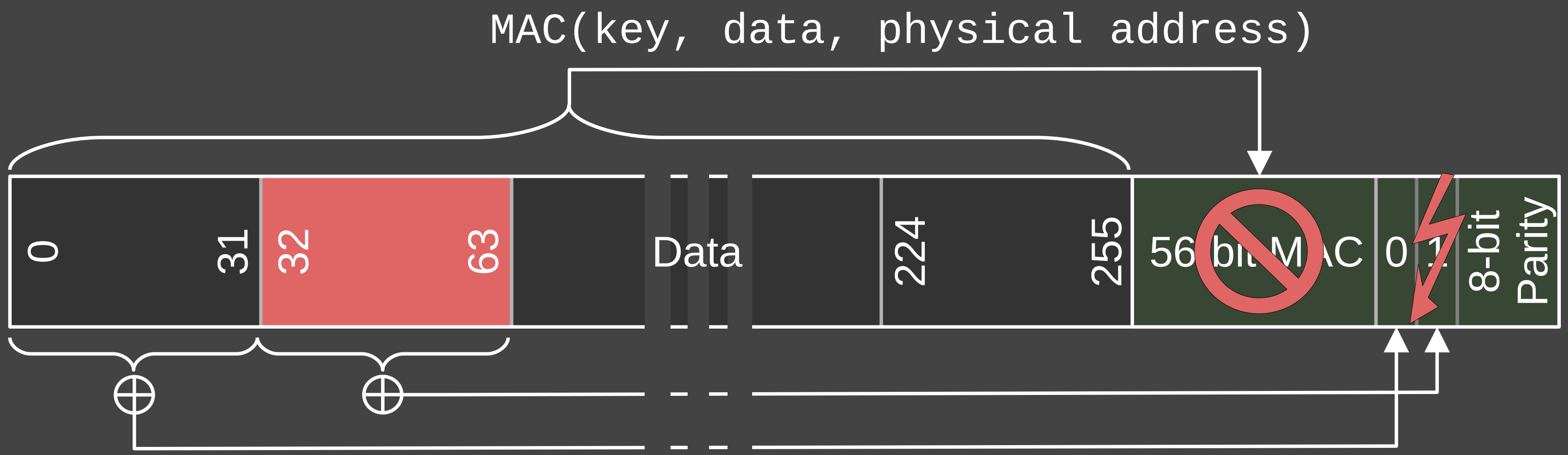
Bit Flip Correction

MACs cannot correct bit flips

Brute force

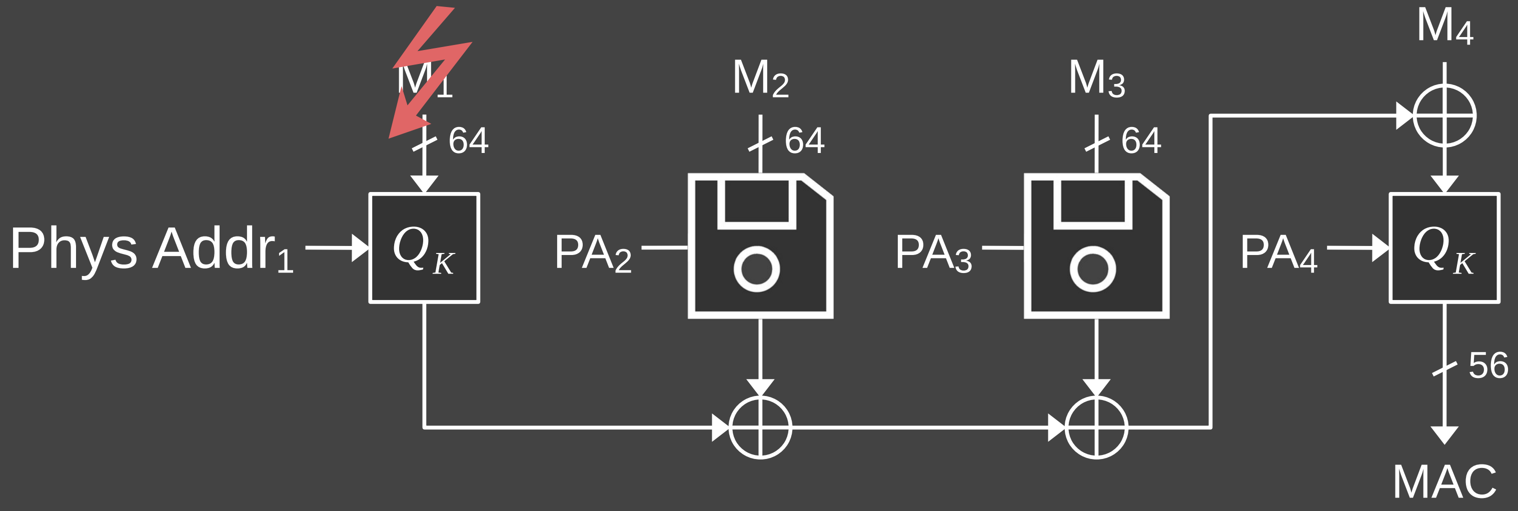
Help from parity bits

Correction in Hardware



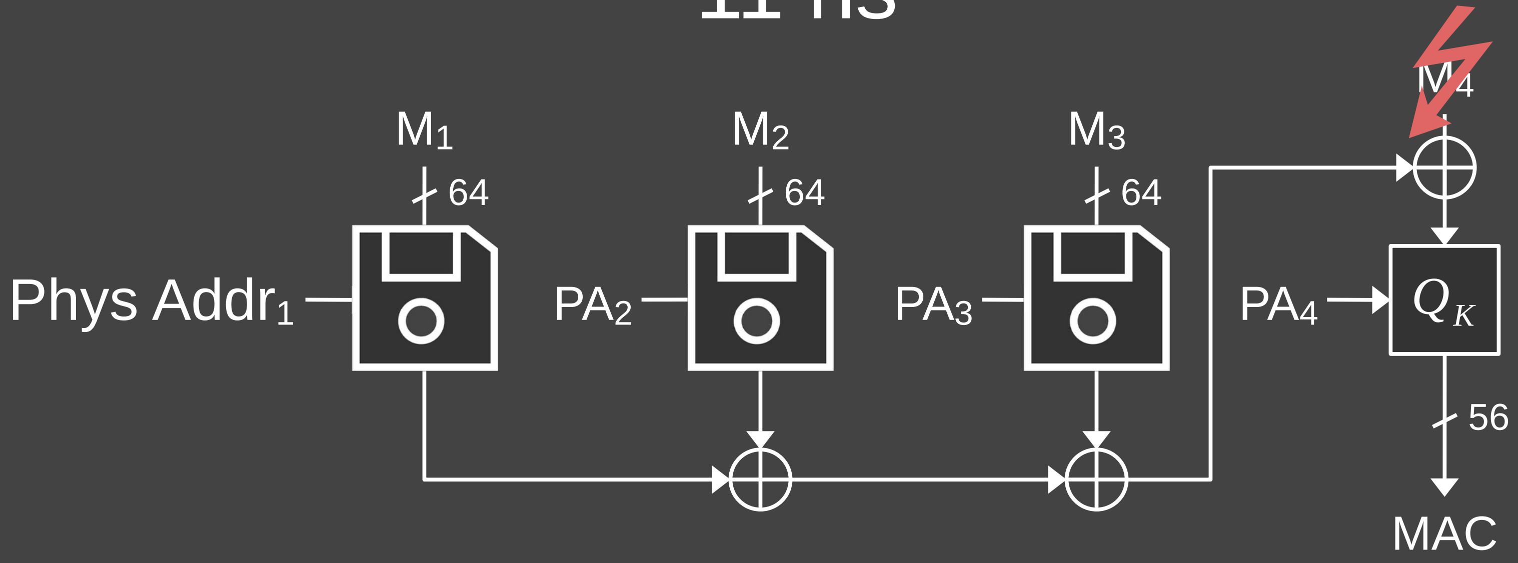
Correction in Hardware

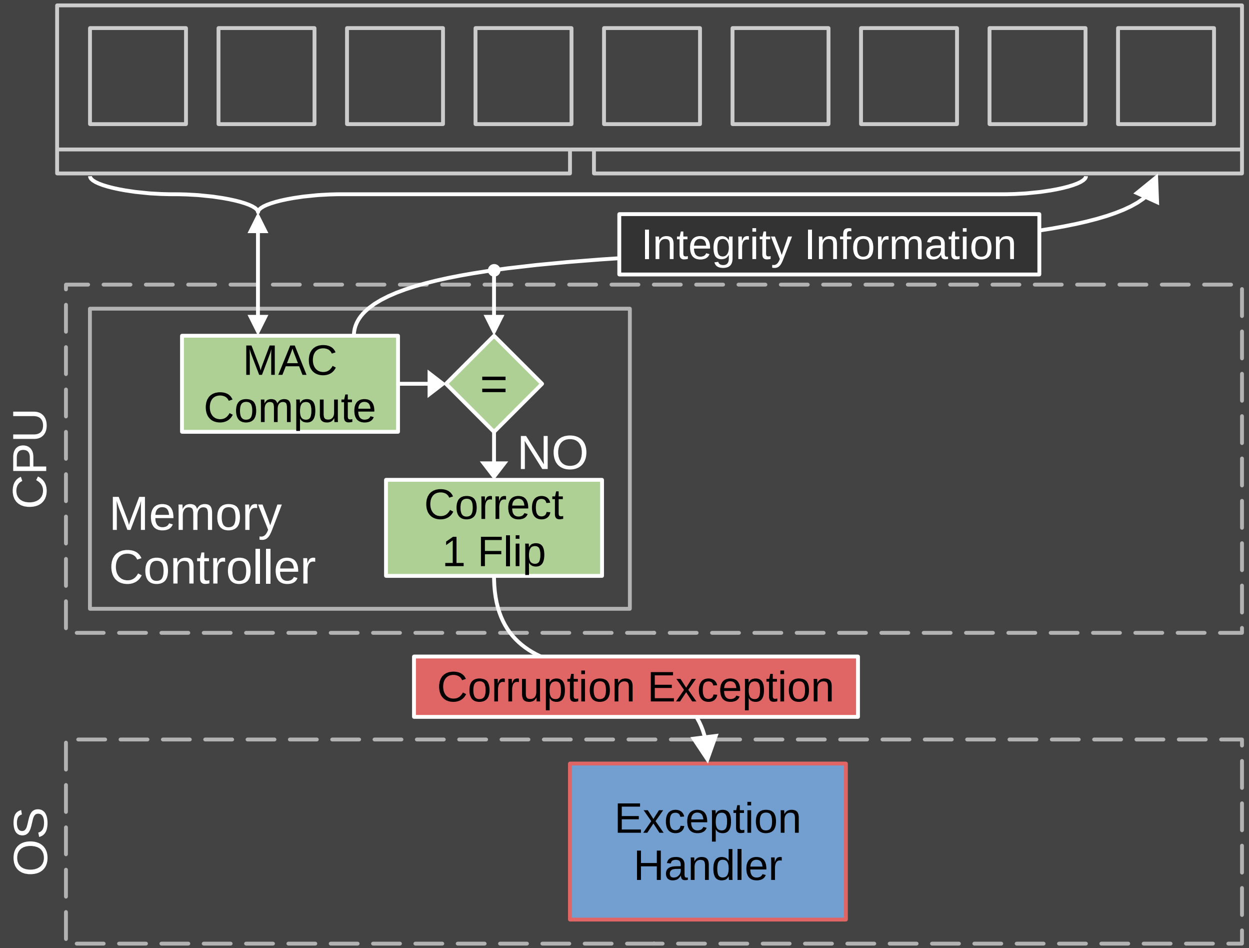
11 ns



Correction in Hardware

11 ns



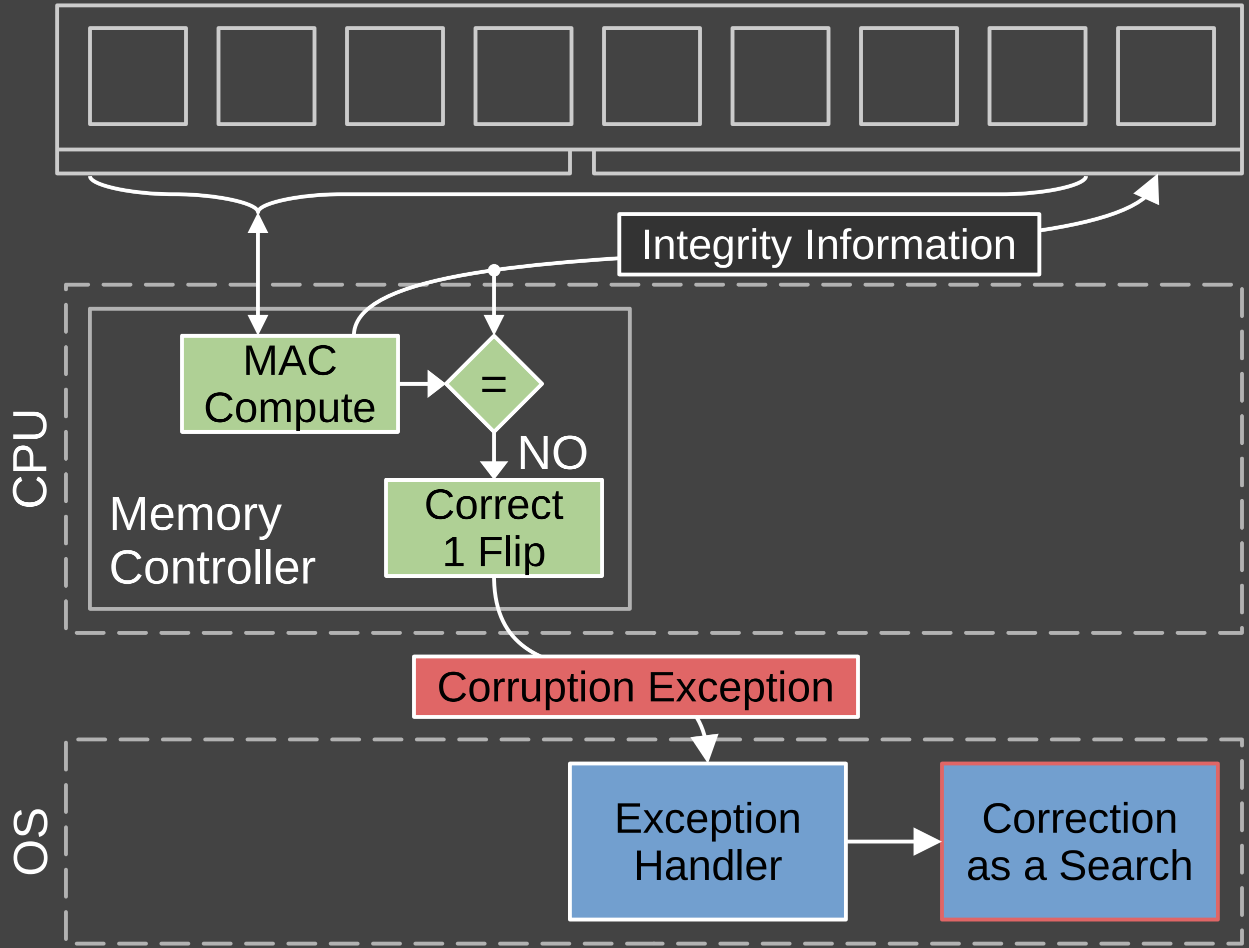

Integrity Information
CPU
MAC
Compute
=
NO
Correct
1 Flip
Memory
Controller
Corruption Exception
OS
Exception
Handler

Corruption Exception

Similar to page fault

Physical address of affected memory location

No dependency on page tables


Integrity Information
CPU
MAC
Compute
=
NO
Correct
1 Flip
Memory
Controller
Corruption Exception
OS
Exception
Handler
Correction
as a Search

Correction as a Search

Brute force all possible flips

As fast as possible

As small as possible

Correction as a Search

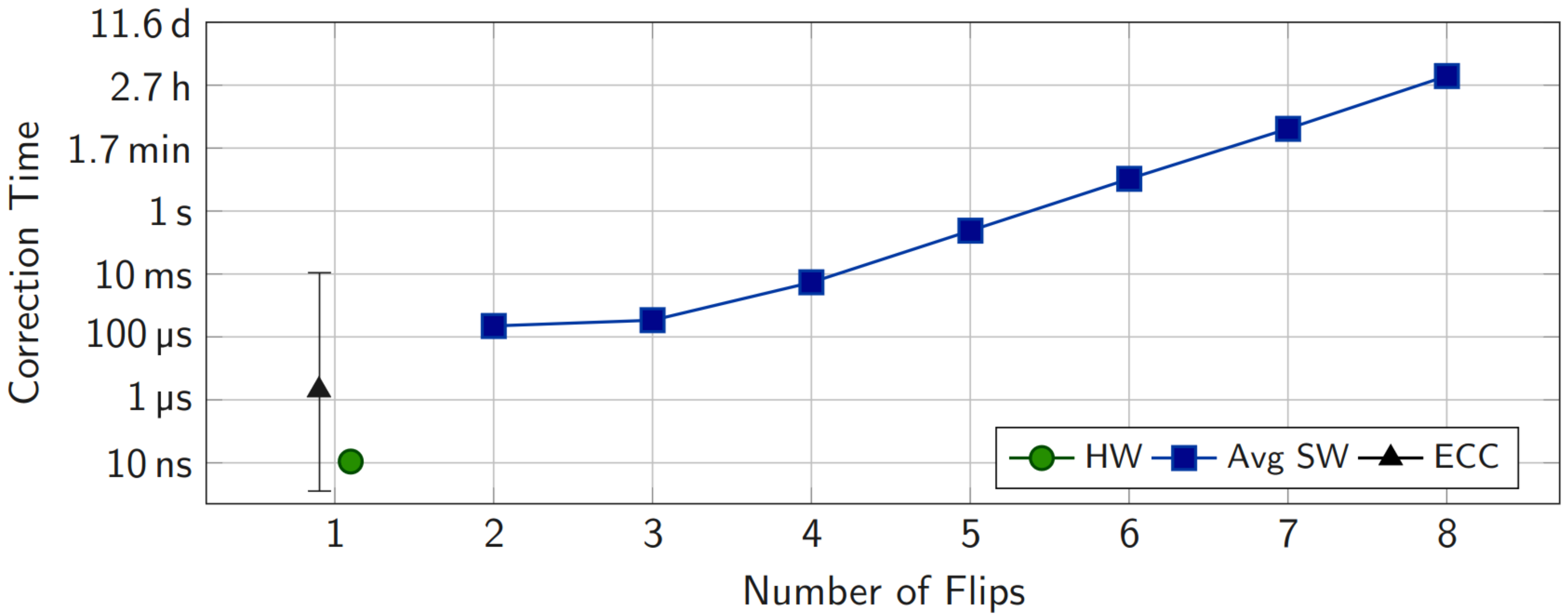
```
LBL_4_PB:
for (pb[3] = 0; pb[3] < CONST_NUM_PARITY_BITS; pb[3]++) {

  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:
                      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_DATA(data, correct_data)) {
                          return mac_computations;
                        }

                        *data_parity_block[0] ^= flip_mask[0];
                      }
```

```c
*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];
```

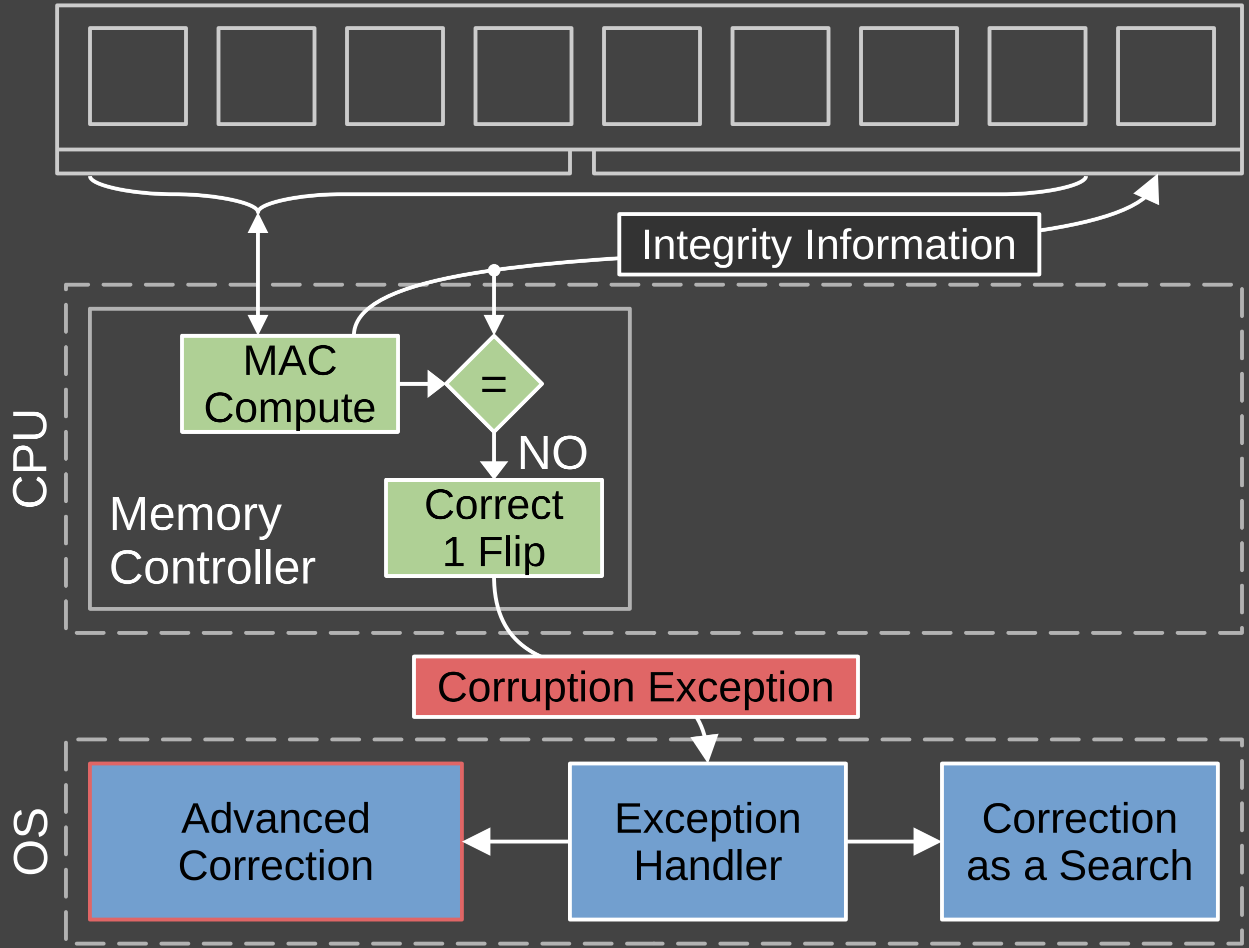
```c
*data_parity_block[2] ^= flip_mask[2];

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];
```

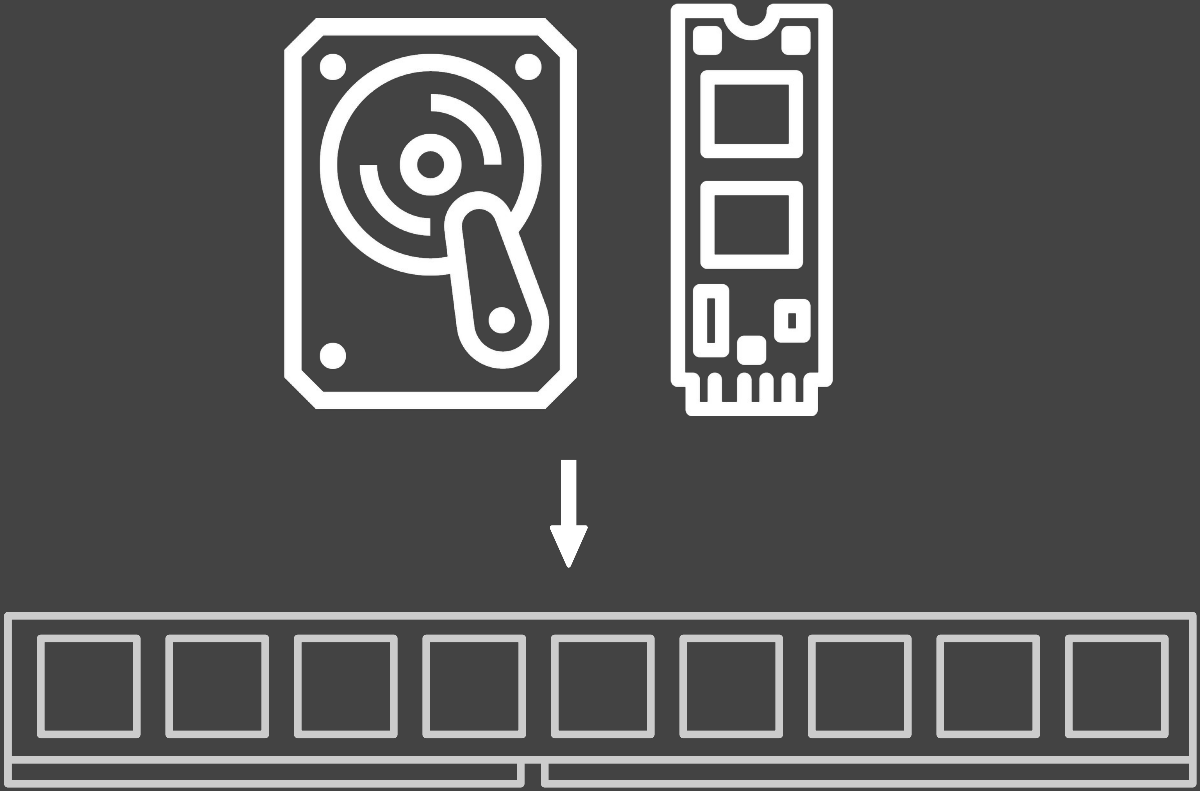
```
// The for outer PB loops are for parity bit input blocks with an even
// number of flips and therefore no change in the parity bits.
LBL_4_PB:
for (pb[3] = 0; pb[3] < CONST_NUM_PARITY_BITS; pb[3]++) {

  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:
```

```
// 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
};
```


11.6 d
2.7 h
1.7 min
1 s
10 ms
100 µs
1 µs
10 ns
Correction Time
1
2
3
4
5
6
7
8
Number of Flips
HW
Avg SW
ECC

Software can be smarter


Integrity Information
CPU
MAC Compute
=
NO
Correct 1 Flip
Memory Controller
Corruption Exception
OS
Advanced Correction
Exception Handler
Correction as a Search



Application Binaries

Shared Libraries

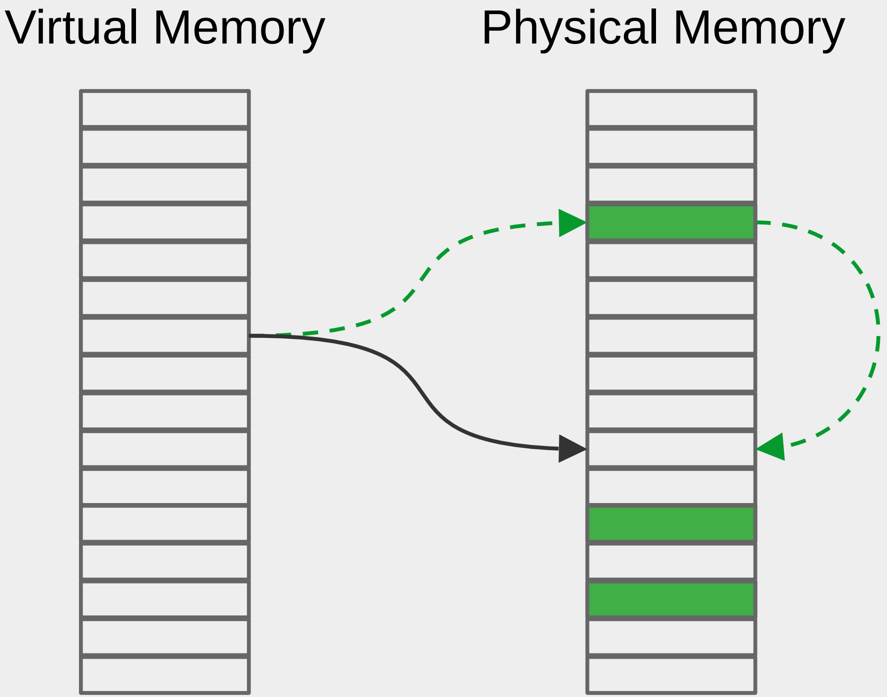
Memory Mapped Files

Page Cache

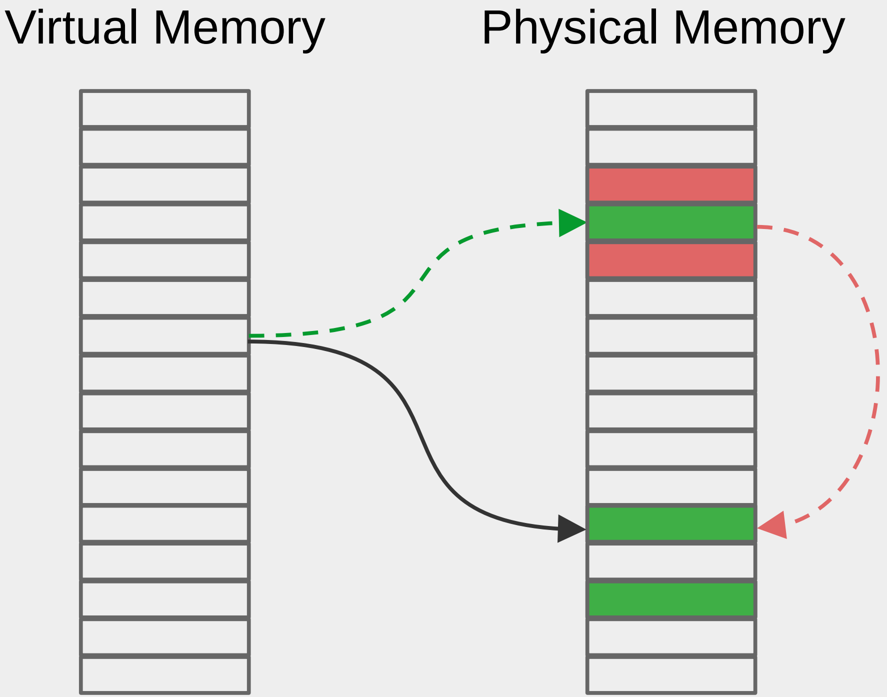
Kernel Itself


Hacker-Pschorr
Himmel
der Bayern

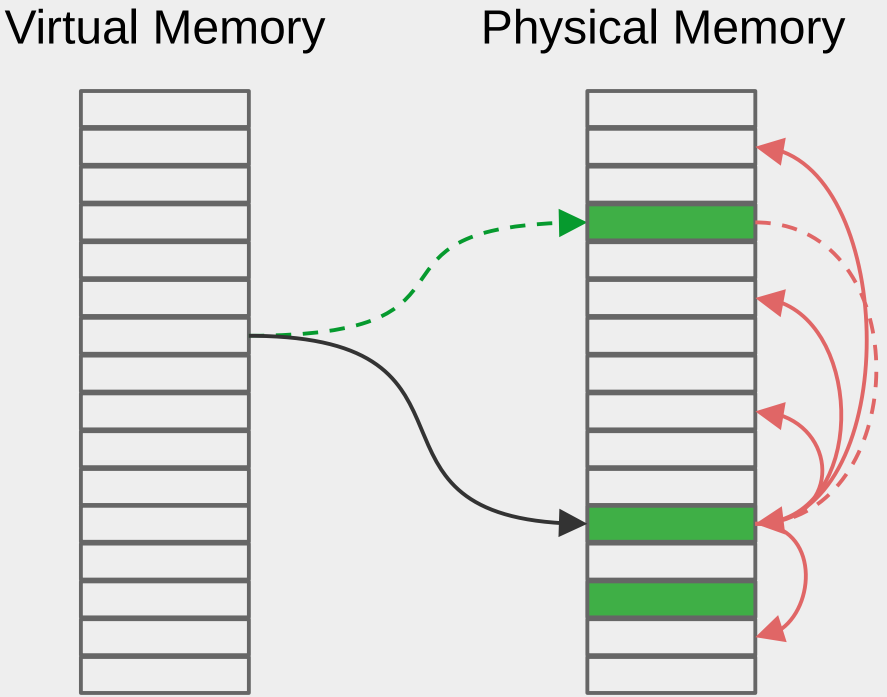
Page Tables



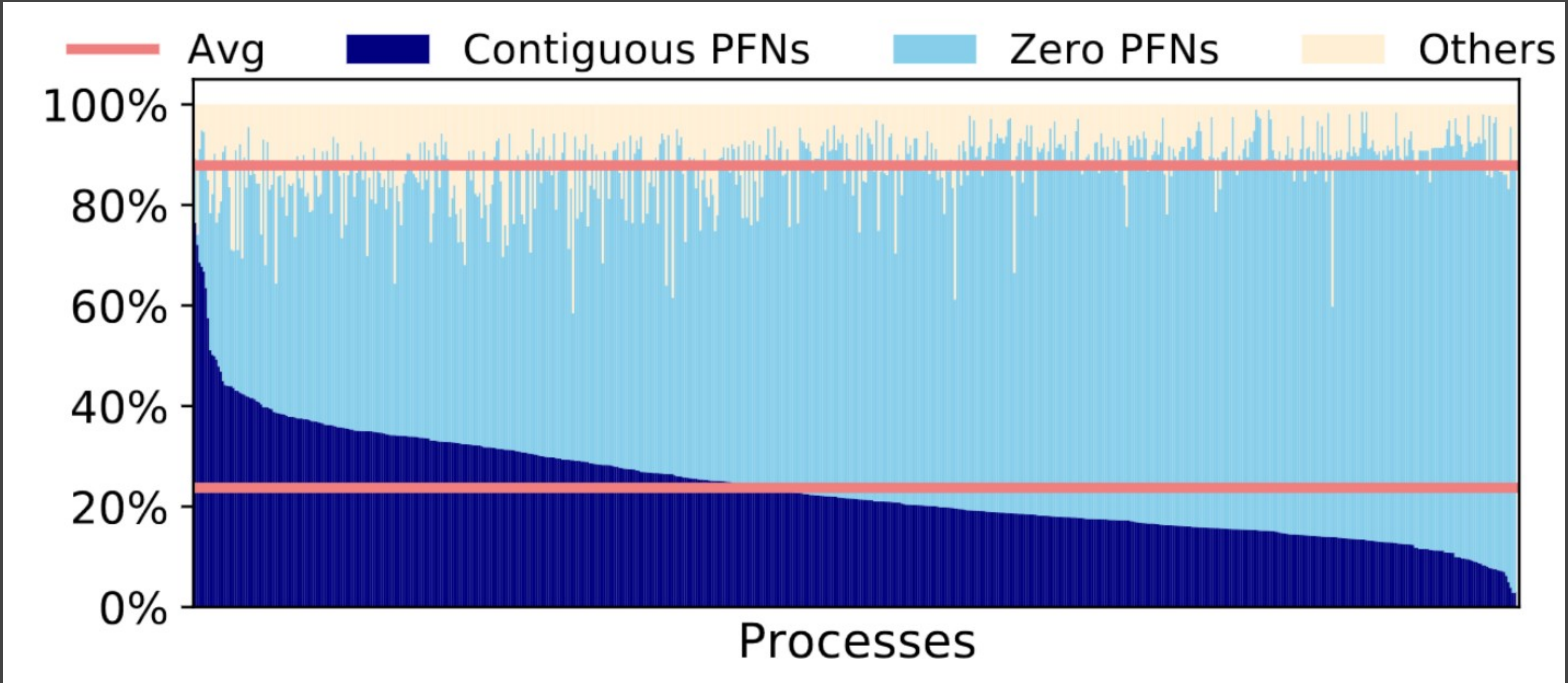
Page Tables



Page Tables



PT-Guard


Avg
Contiguous PFNs
Zero PFNs
Others
100%
80%
60%
40%
20%
0%
Processes

The MAC can also flip

Th ip

MAC Corruption

Approximate Equality:

Comp. MAC: 0101001...

DRAM MAC: 0111001...

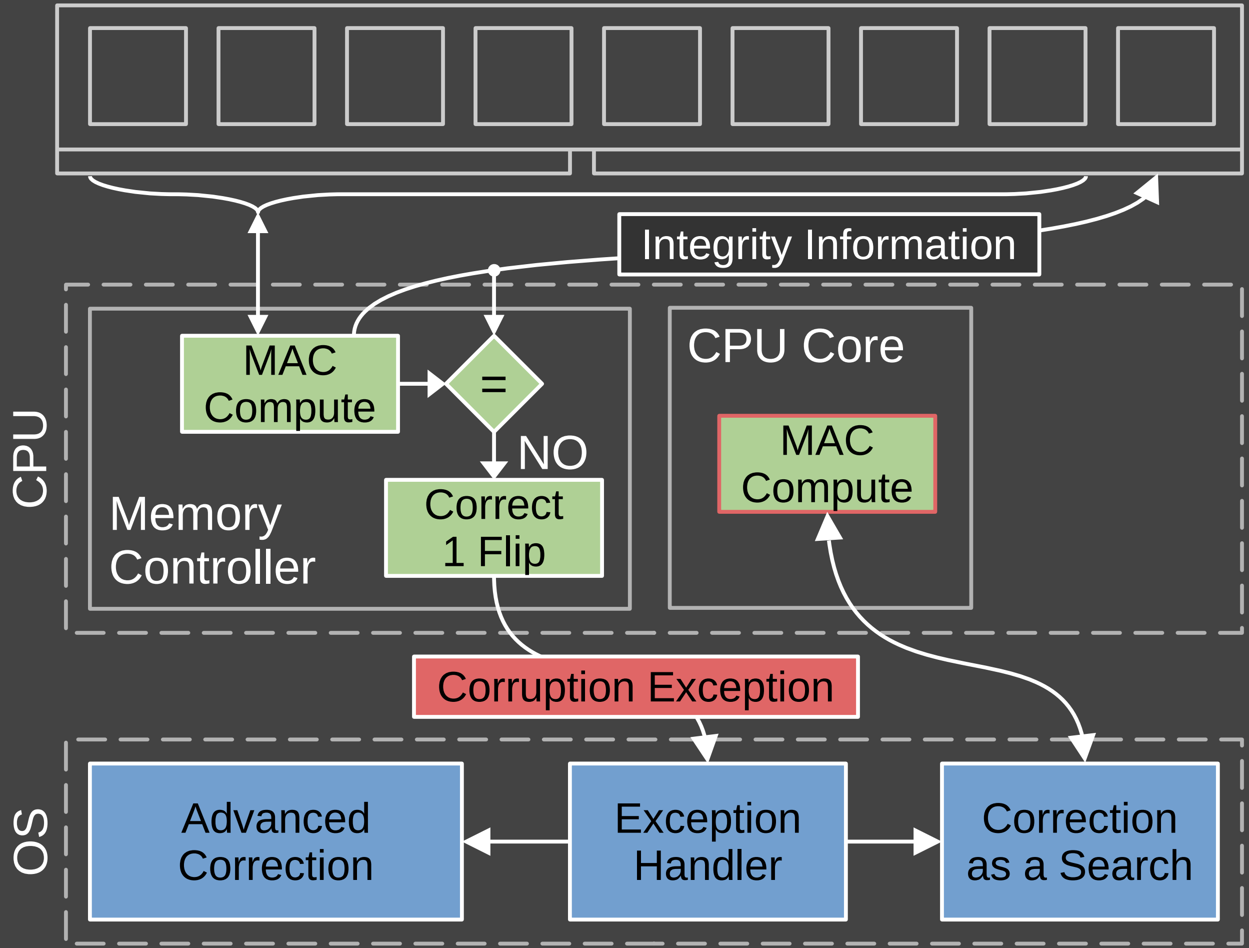
Approximate Equality

$$\log_2 \sum_{k=0}^{3} \binom{56}{k}$$

$$MAC\ strength: 56\,bit \rightarrow 41.2\,bit$$

SDC once per 10^9 billion years

Rowhammer second preimage after one year: 10^{-4} %


Integrity Information
CPU
MAC Compute
=
NO
Correct 1 Flip
Memory Controller
CPU Core
MAC Compute
Corruption Exception
OS
Advanced Correction
Exception Handler
Correction as a Search

CPU Instructions

`csi_mac`

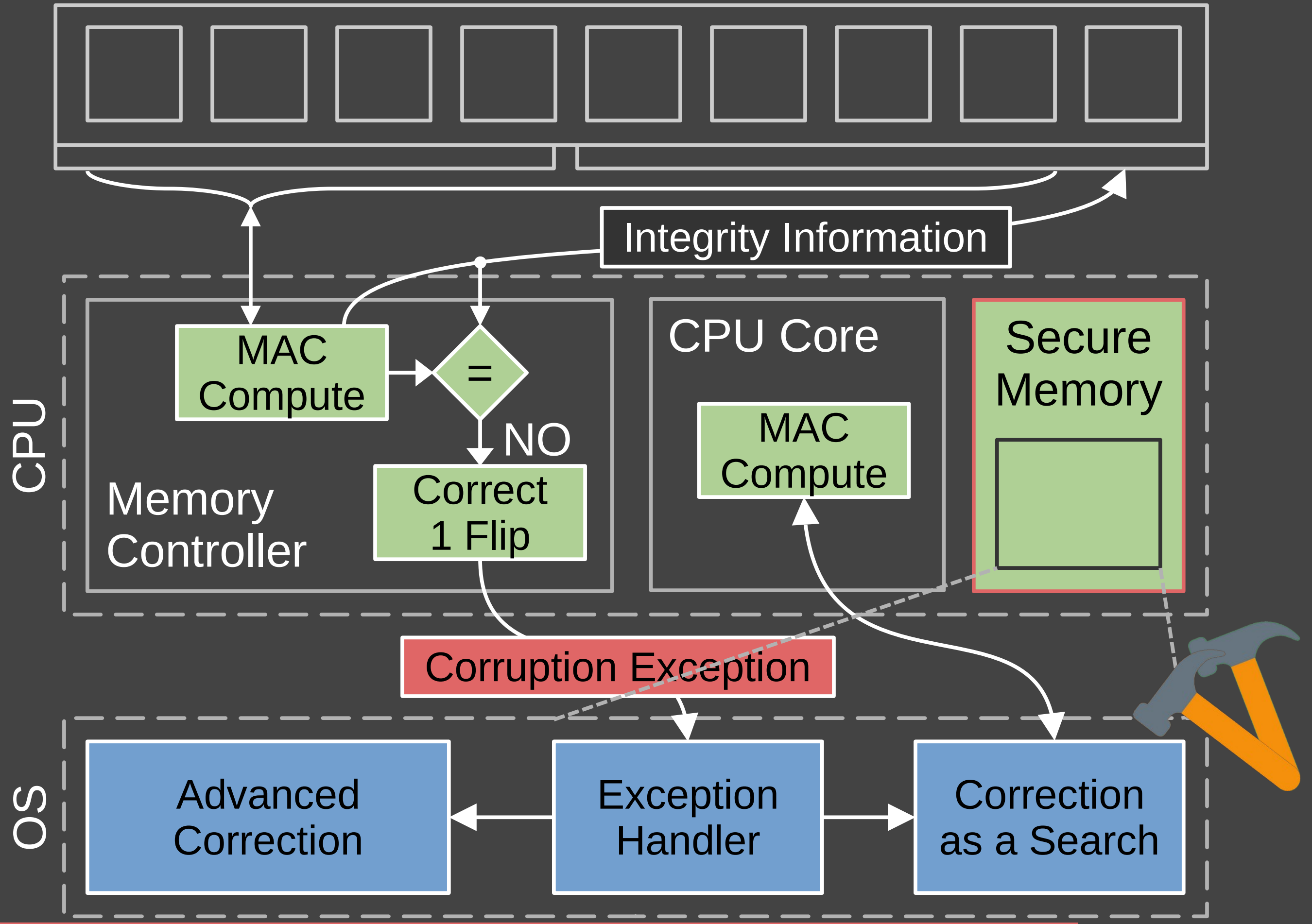
Compute MAC

`csi_load`

Load based on physical address

`csi_xchg`

Write if unchanged in DRAM


Integrity Information
CPU
MAC Compute
=
NO
Correct 1 Flip
Memory Controller
CPU Core
MAC Compute
Secure Memory
Corruption Exception
OS
Advanced Correction
Exception Handler
Correction as a Search

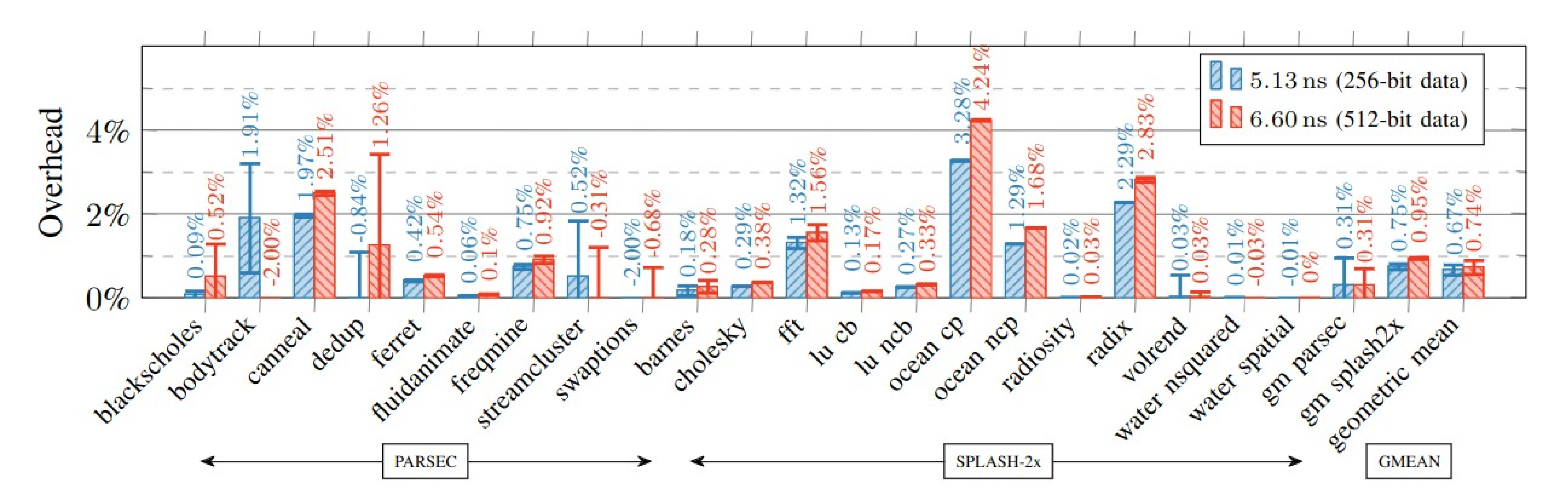
Secure Memory

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

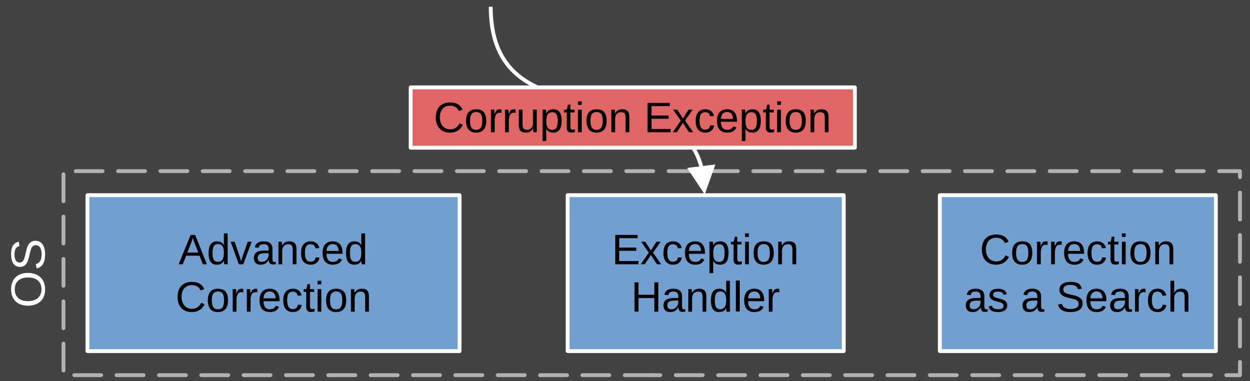
Performance Evaluation

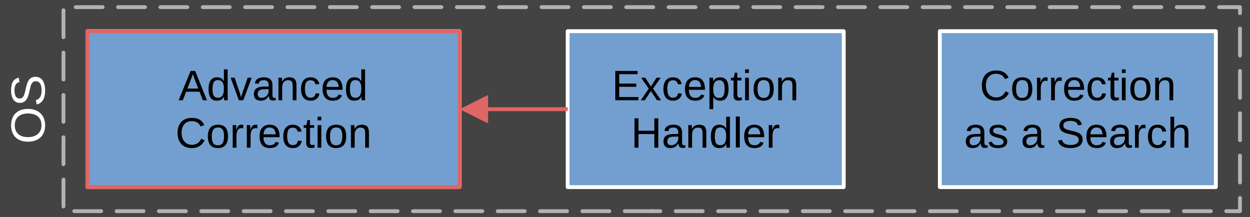
Implemented in gem5 and Linux kernel

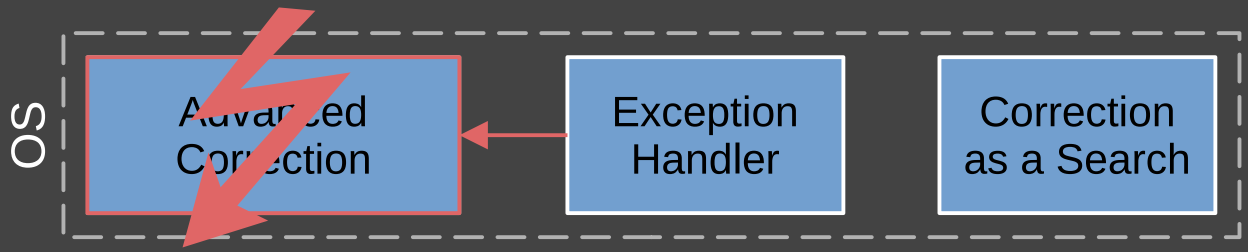
System not under attack

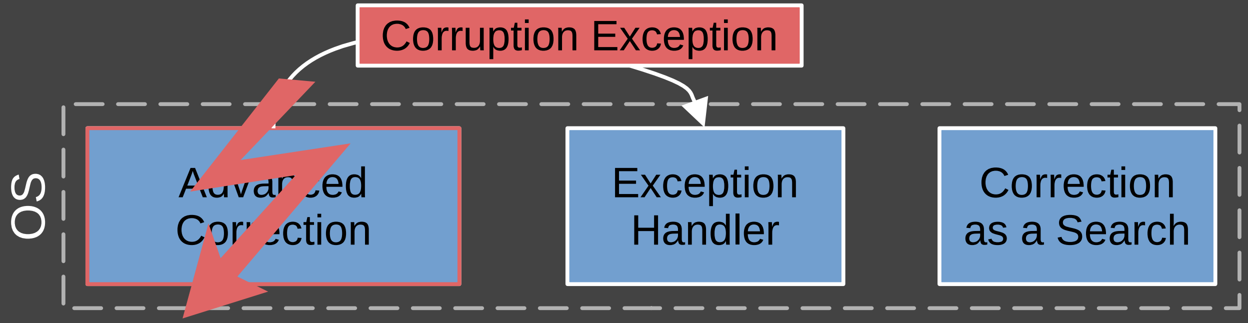

Overhead
0%
2%
4%
5.13 ns (256-bit data)
6.60 ns (512-bit data)
blackscholes
bodytrack
canneal
dedup
ferret
fluidanimate
freqmine
streamcluster
swaptions
barnes
cholesky
fft
lu cb
lu ncb
ocean cp
ocean ncp
radiosity
radix
volrend
water nsquared
water spatial
gm parsec
gm splash2x
geometric mean
PARSEC
SPLASH-2x
GMEAN

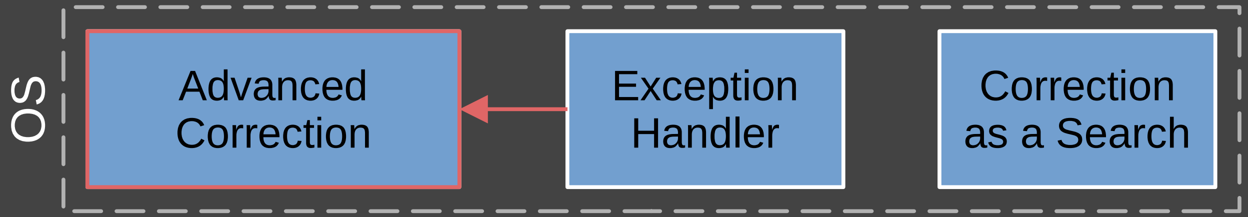
Nesting Detection

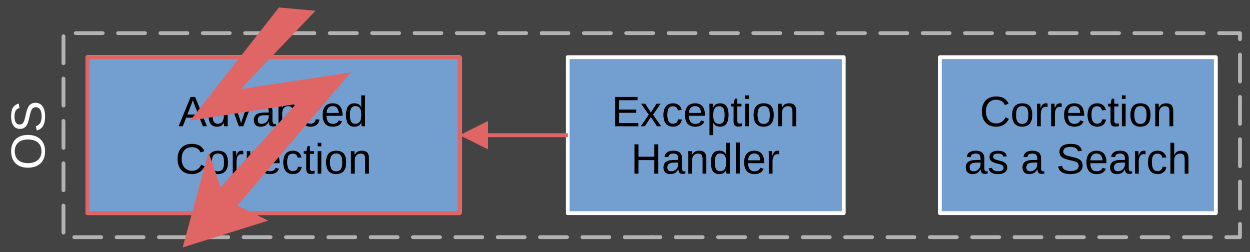

Corruption Exception
OS
Advanced Correction
Exception Handler
Correction as a Search

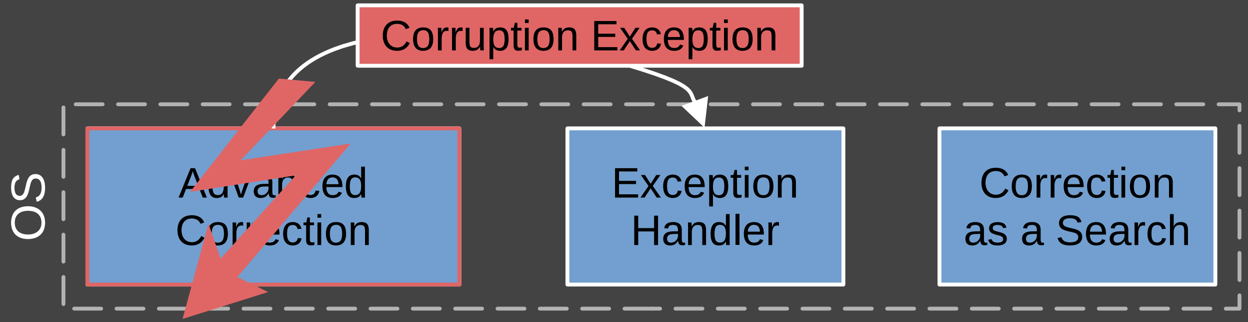

OS
Advanced Correction
Exception Handler
Correction as a Search

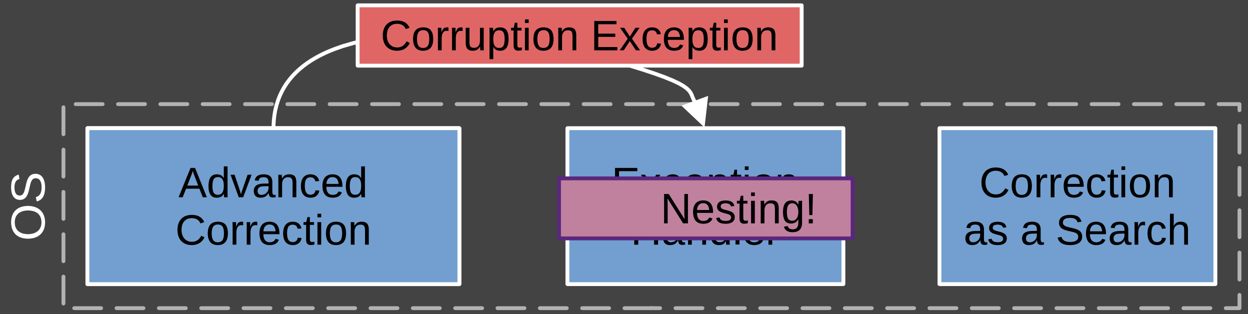

OS
Advanced Correction
Exception Handler
Correction as a Search

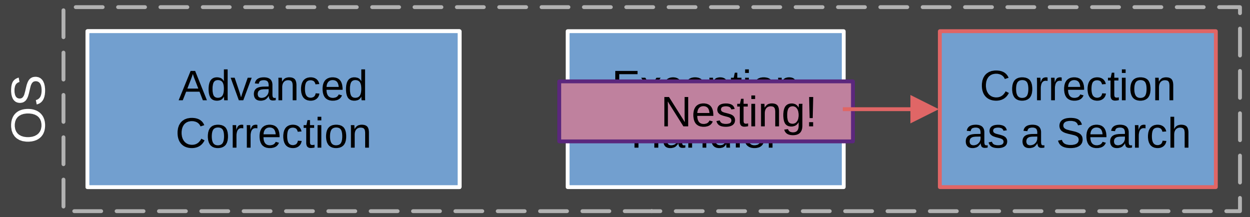

Corruption Exception
OS
Advanced Correction
Exception Handler
Correction as a Search


OS
Advanced Correction
Exception Handler
Correction as a Search


OS
Advanced Correction
Exception Handler
Correction as a Search


Corruption Exception
OS
Advanced Correction
Exception Handler
Correction as a Search


Corruption Exception
OS
Advanced Correction
Nesting!
Correction as a Search


OS
Advanced Correction
Nesting!
Correction as a Search

```
corruption_exception_handler(corruption_address)
{
  if (nested_bit_set())
  {
    enable_interrupts();
    error_correction_as_a_search(corruption_address);
  }
  else
  {
    set_nested_bit();
    enable_interrupts();
    advanced_error_correction(corruption_address);
    clear_nested_bit();
  }
}
```

Nesting Detection

Nested Bit

- Per Process
- In Register

High bit in CR3 register

Unused	Base of PML4T	PCID

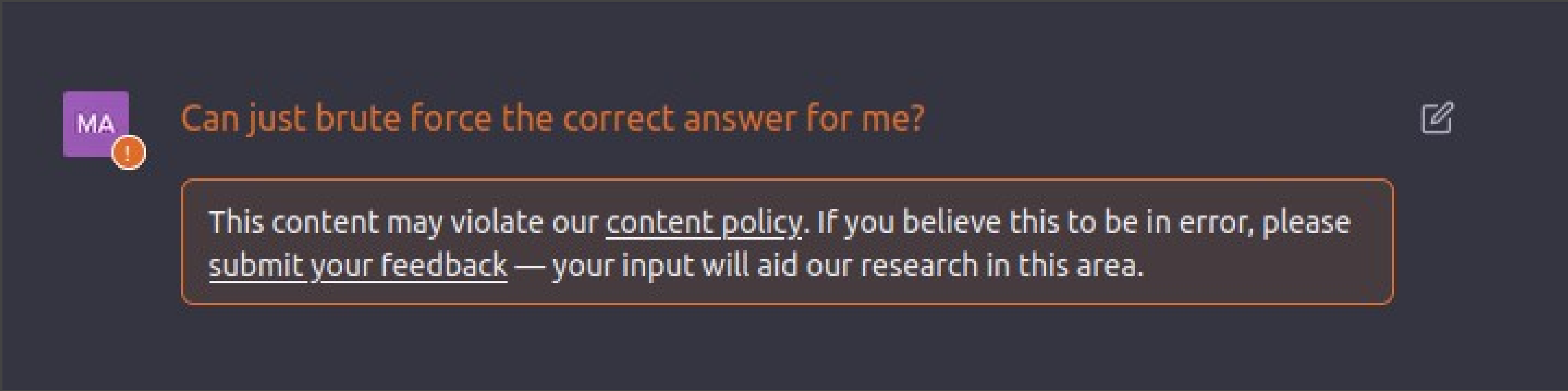
Very Advanced Correction

They say AI can do everything

MA

Find the error in "Testmessoge" so that is has the following SHA512 hash "9167bf29600fea2cb1ba4e21e2f311bf5c2641f0cd5cb5bec948177264ff78e2bc346216427b75e75fe67e842ef3085ac303b8111a37adeaf977b2b269ada636"

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.


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

Ice Lake SP Xeon Server die size: 628 mm^2

Fits roughly 500 000 QARMA blocks

8 flips correction: 5.44 h → 40 ms

Single correction node in datacenter

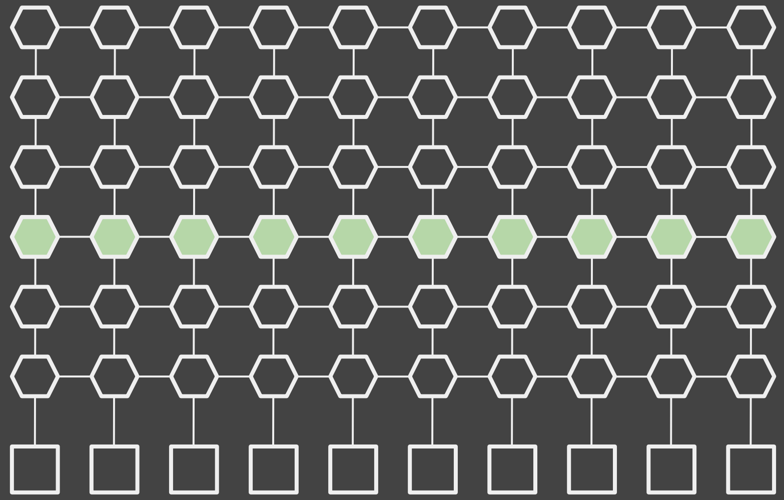
There's more

- VM support
  - With and without guest
- Memory encryption
- Hardware correction of stuck bits
- Race conditions resistance

Keeping the Promise

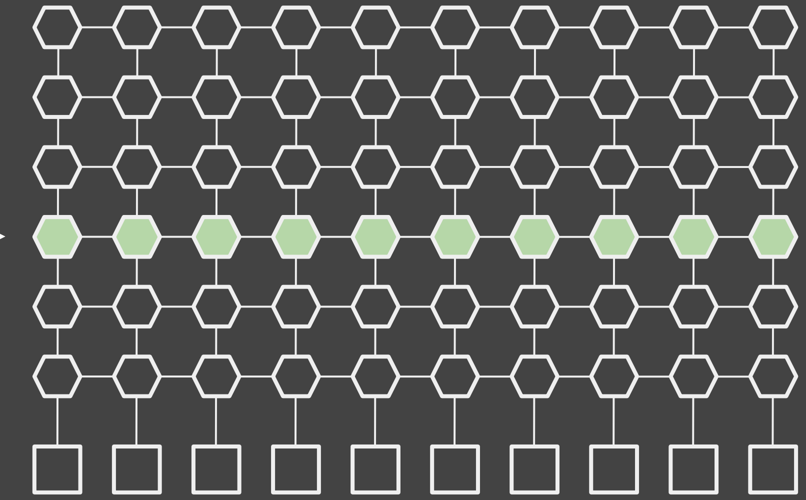
Rowpress Luo et al.

```
while (1)
{
  *aggressor[0];
  *aggressor[1];
  *aggressor[2];
  ...
  clflush(aggressor[0]);
  clflush(aggressor[1]);
  clflush(aggressor[2]);
  ...
}
```



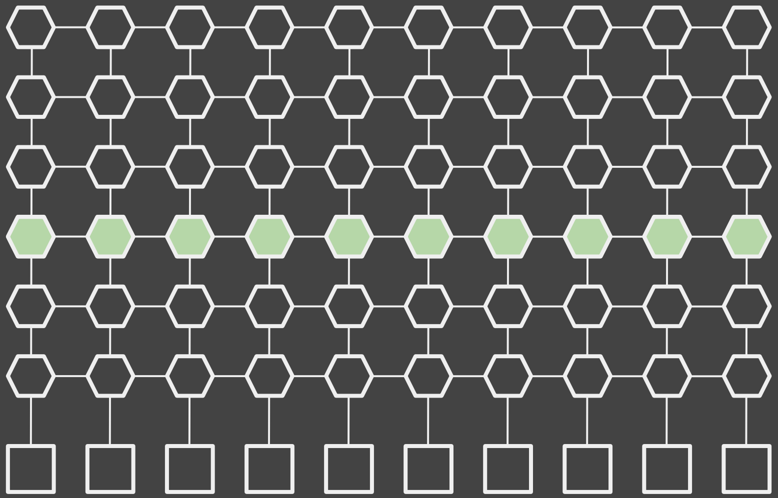
Rowpress Luo et al.

```
while (1)
{
  *aggressor[0];
  *aggressor[1];
  *aggressor[2];
  ...
  clflush(aggressor[0]);
  clflush(aggressor[1]);
  clflush(aggressor[2]);
  ...
}
```



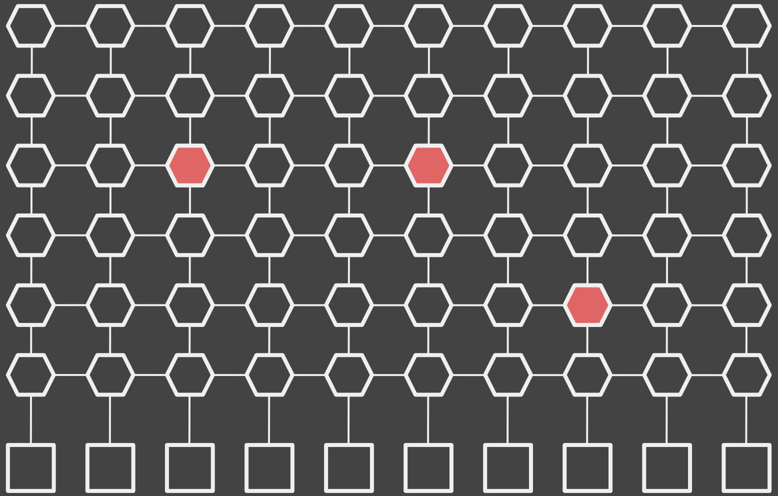
Rowpress Luo et al.

```
while (1)
{
  *aggressor[0];
  *aggressor[1];
  *aggressor[2];
  ...
  clflush(aggressor[0]);
  clflush(aggressor[1]);
  clflush(aggressor[2]);
  ...
}
```



Rowpress Luo et al.

```
while (1)
{
  *aggressor[0];
  *aggressor[1];
  *aggressor[2];
  ...
  clflush(aggressor[0]);
  clflush(aggressor[1]);
  clflush(aggressor[2]);
  ...
}
```



Rowpress

Not Rowhammer

Defeats many Mitigations

Detected by CSI:Rowhammer

CSI:Rowhammer

Cryptographic Security and Integrity against Rowhammer

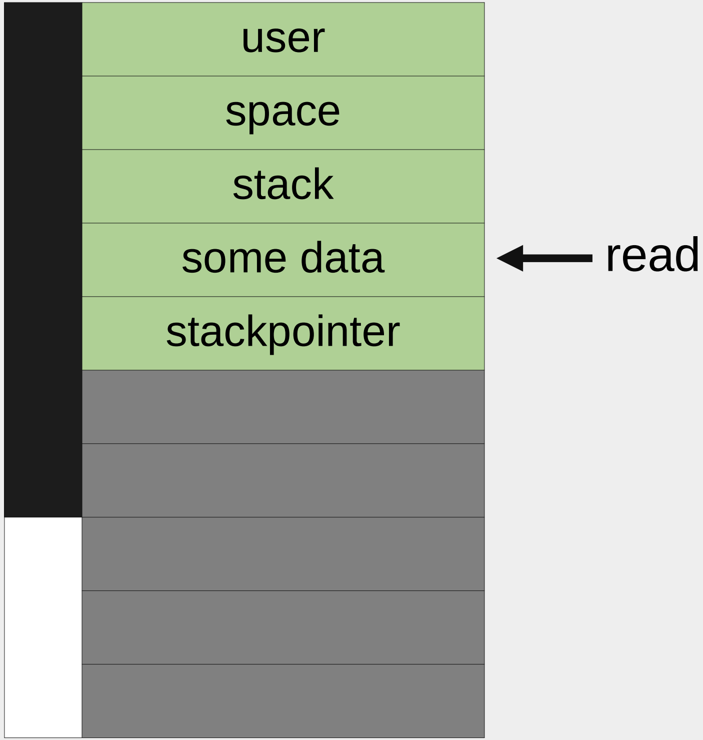
Jonas Juffinger, Lukas Lamster, Andreas Kogler, Moritz Lipp, Maria Eichlseder, Daniel Gruss

jonas.juffinger@iaik.tugraz.at – @notimaginary_ – www.jonasjuffinger.com

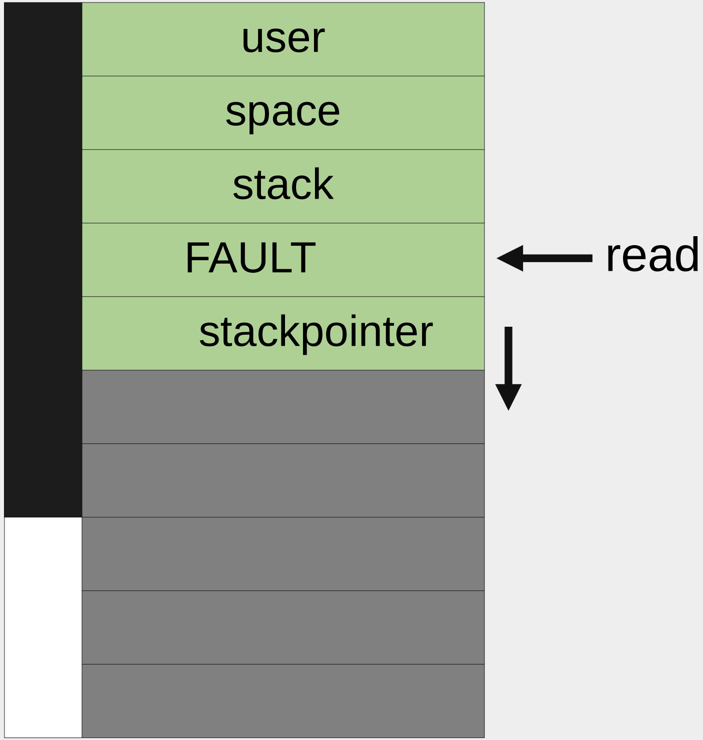
PoC: https://github.com/IAIK/CSIRowhammer

Additional Slides

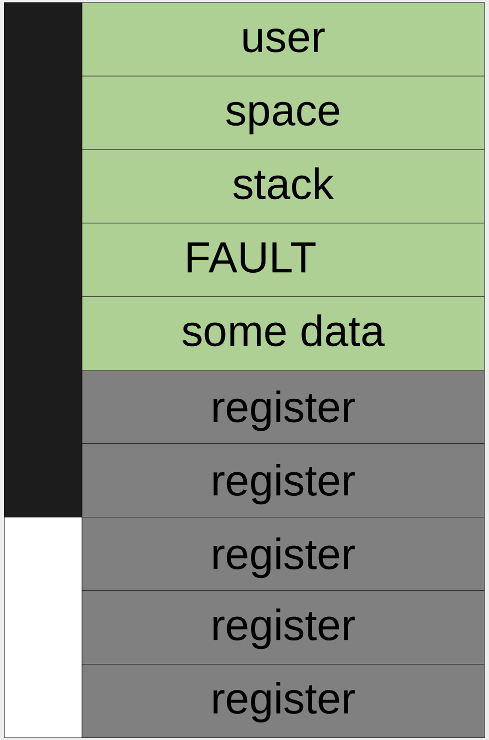
Stack Protection



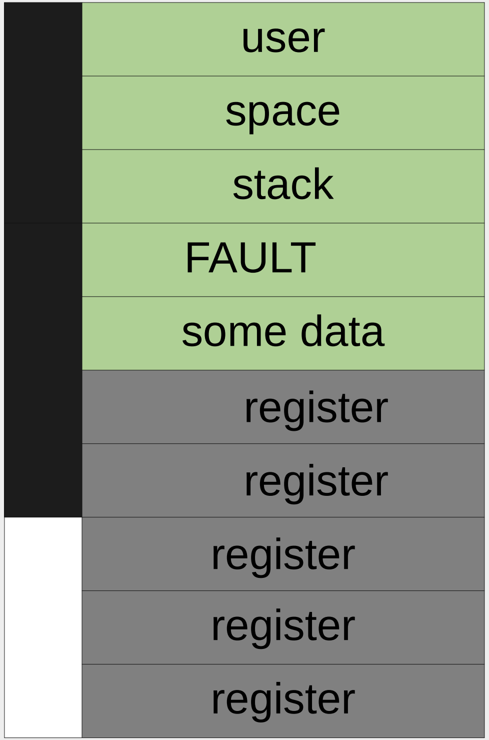
Stack Protection



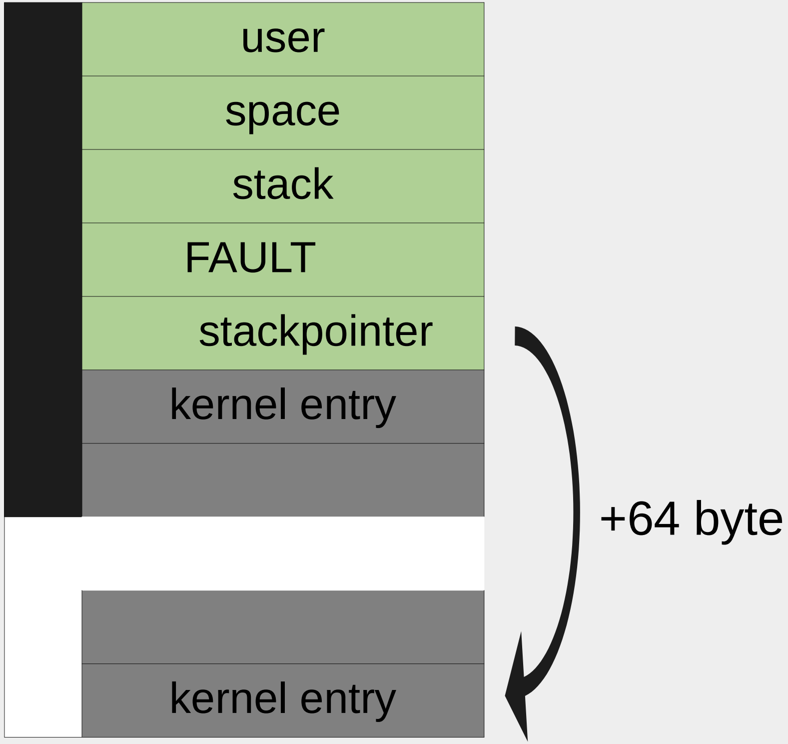
Stack Protection



Stack Protection



Stack Protection



Stack Protection