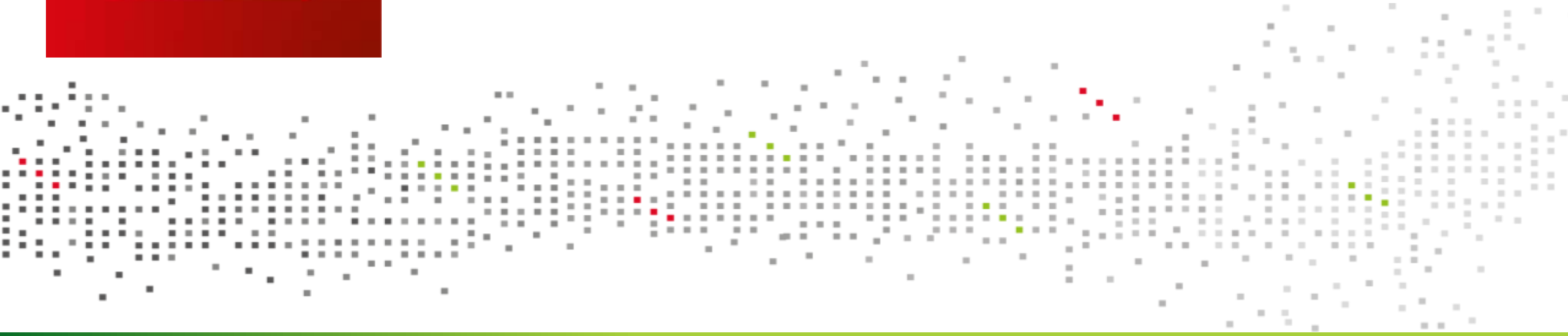


Pointer Integrity on Nanotrust Secure RISC-V processor

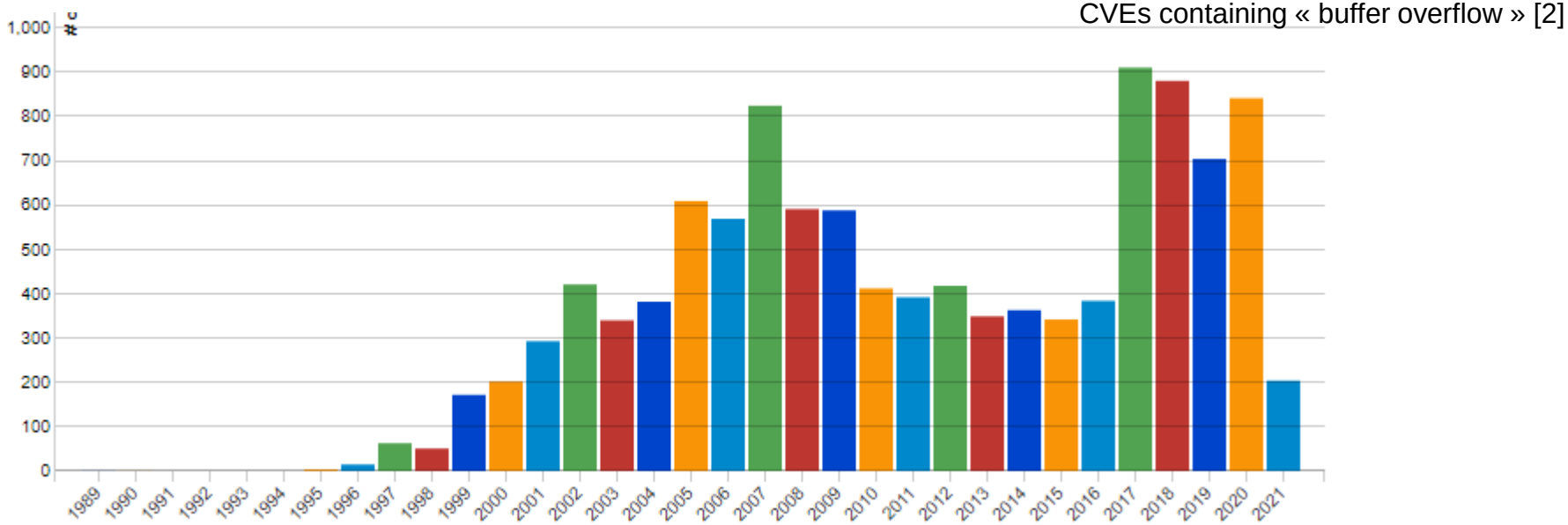
Thomas Hiscock, Olivier Savry, Mustapha El-Majihi

Univ. Grenoble Alpes, CEA, LETI
MINATEC Campus, F-38054 Grenoble, France
Email: firstname.name@cea.fr

Memory Satefy Bugs

- **Among the most exploited vulnerabilities in programs**
  - Remote code execution
  - Data corruption
  - Information disclosure
- **Microsoft: 70% of all exploits rely on memory safety issues [1]**

CVEs containing « buffer overflow » [2]

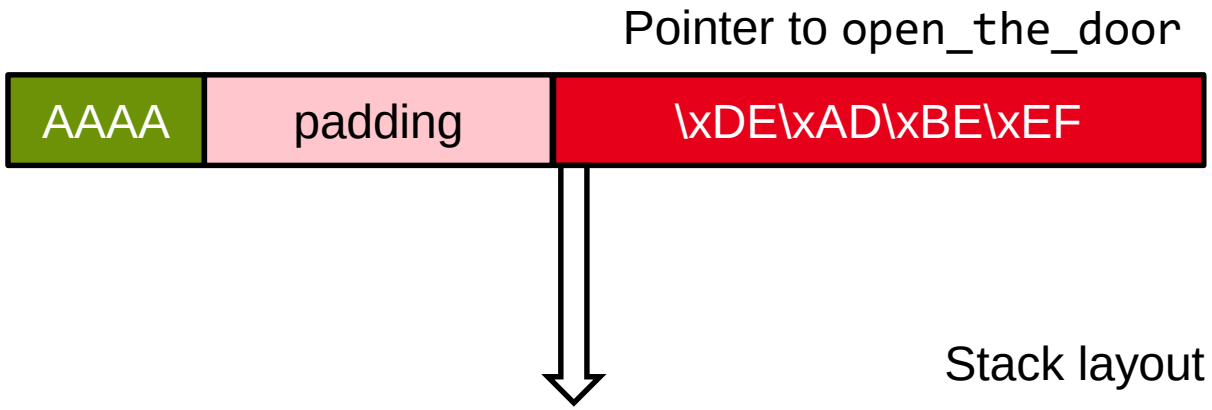


[1] Matt Miller, « Trends, challenges, and strategic shift in the software vulnerability mitigation landscape, Bule Hat 2019

[2] source : https://nvd.nist.gov/vuln/search

Why?

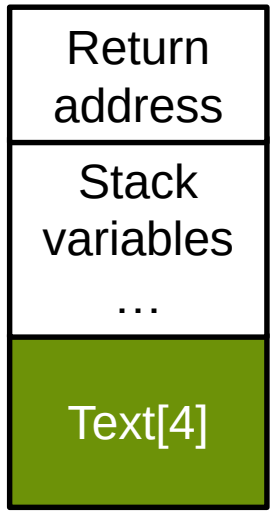
- « Usafe Languages »

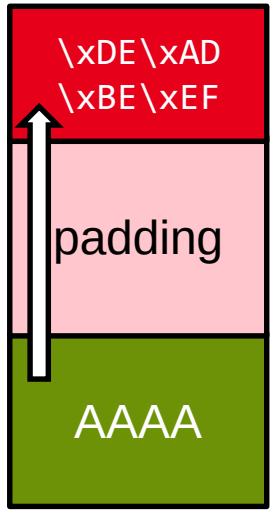



```
void check_input(const char* s) {
    char text[4];
    strcpy(text, s);
    // Other code…
}

void open_the_door() {
        //…
}
```

Stack layout





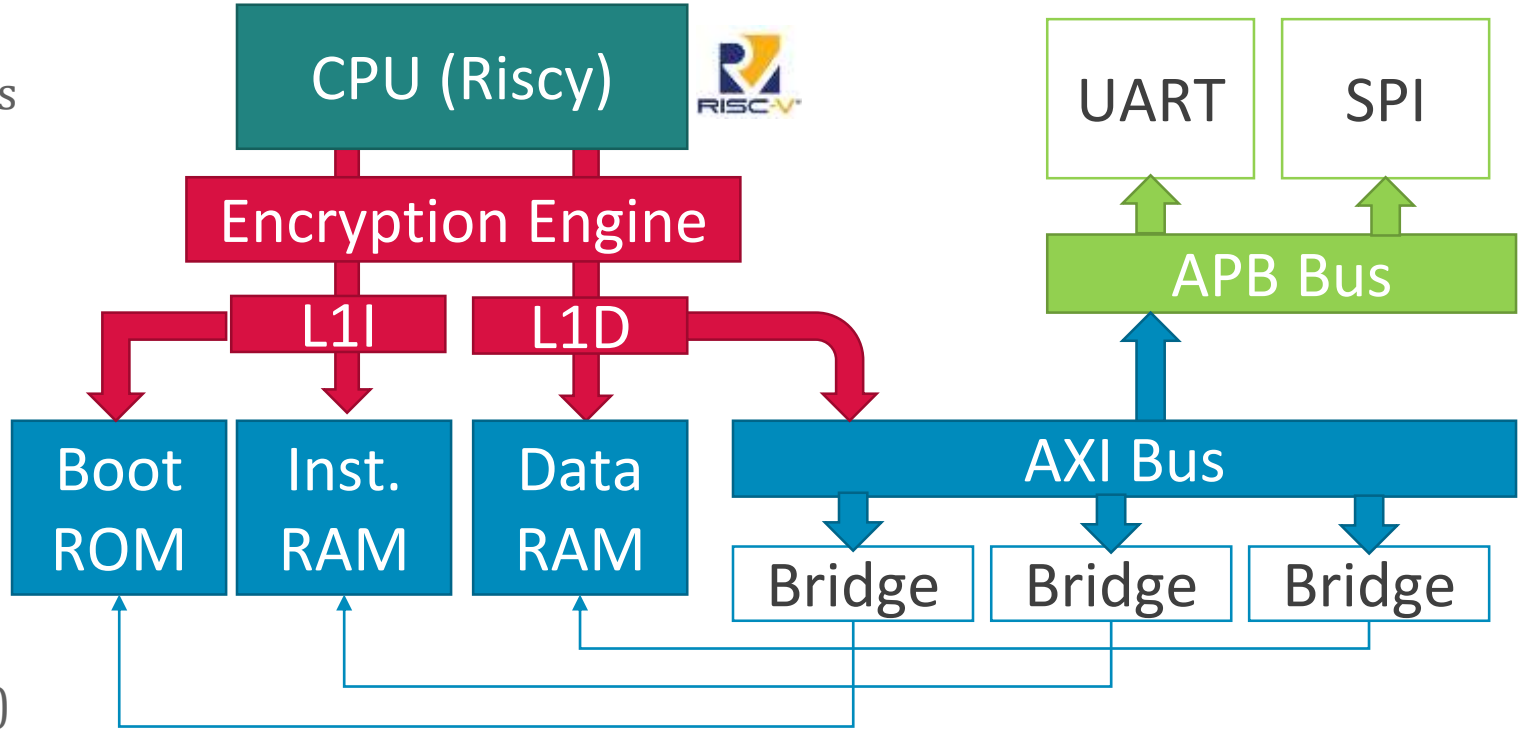
Other vulnerabilities:

- Heap overflow
- Use-after-free
- Type confusion
- Double free
- Uninitialized read

The Nanotrust Project (2018-Now)

- **Philosophy: make processors inherently secure**
  - Strong isolation by default
  - Prevent exploits of future vulnerabilities
  - Reduce scalability of attacks

- **Countermeasures**
  - Memory Encryption and Integrity [1]
  - Control-Flow Integrity [1]
  - Randomized caches [2]
  - Secure pipeline against side-channels and fault injections
  - OS support for strong isolation
  - Proved compiler extensions (CompCert)
  - Pointer Integrity (Today talk!)



[1] Savry, Olivier, Mustapha El-Majihi, and Thomas Hiscock. "Confidaent: Control FLow protection with Instruction and Data Authenticated Encryption." *2020 23rd Euromicro Conference on Digital System Design (DSD)*.
[2] Amine Jaamoum, Thomas Hiscock, Giorgio Di Natale, « Scramble Cache: an efficient architecture for randomized set permutation", DATE 2021.

Pointer Integrity: definition

Source code

```
int check_input(const char* s) {
    char text[4];
    strcpy(text, s);
    return 0;
}
```

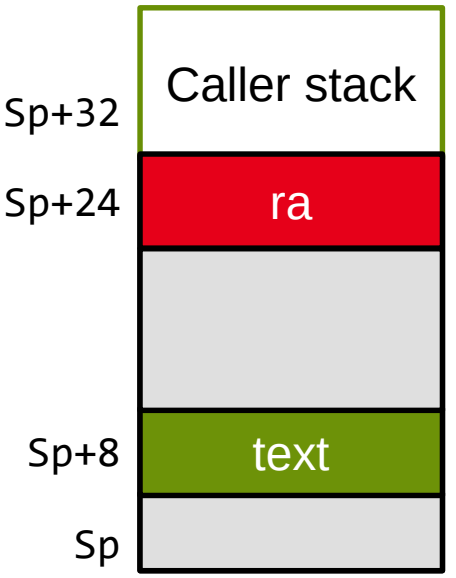
RISC-V Assembly

```
check_input(char const*):
        addi    sp,sp,-32
        mv      a1,a0
        addi    a0,sp,8
        sd      ra,24(sp)
        call    strcpy
        ld      ra,24(sp)
        li      a0,0
        addi    sp,sp,32
        jr      ra
```

Pointer for `text` on the stack.

Pointer for the return address on the stack.

Stack

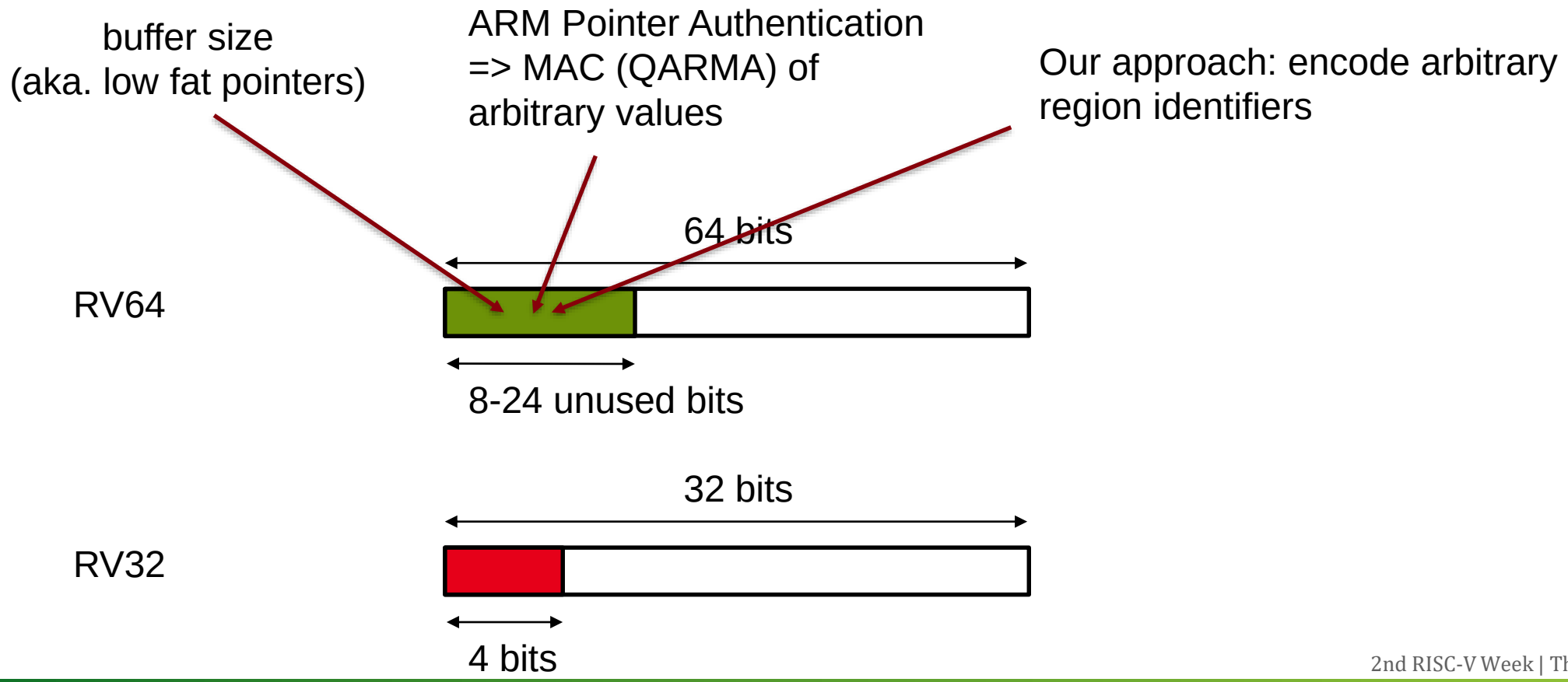


Pointer integrity: each pointer should only allow access to its **region**

Technical challenges: **encoding** and **validation**.

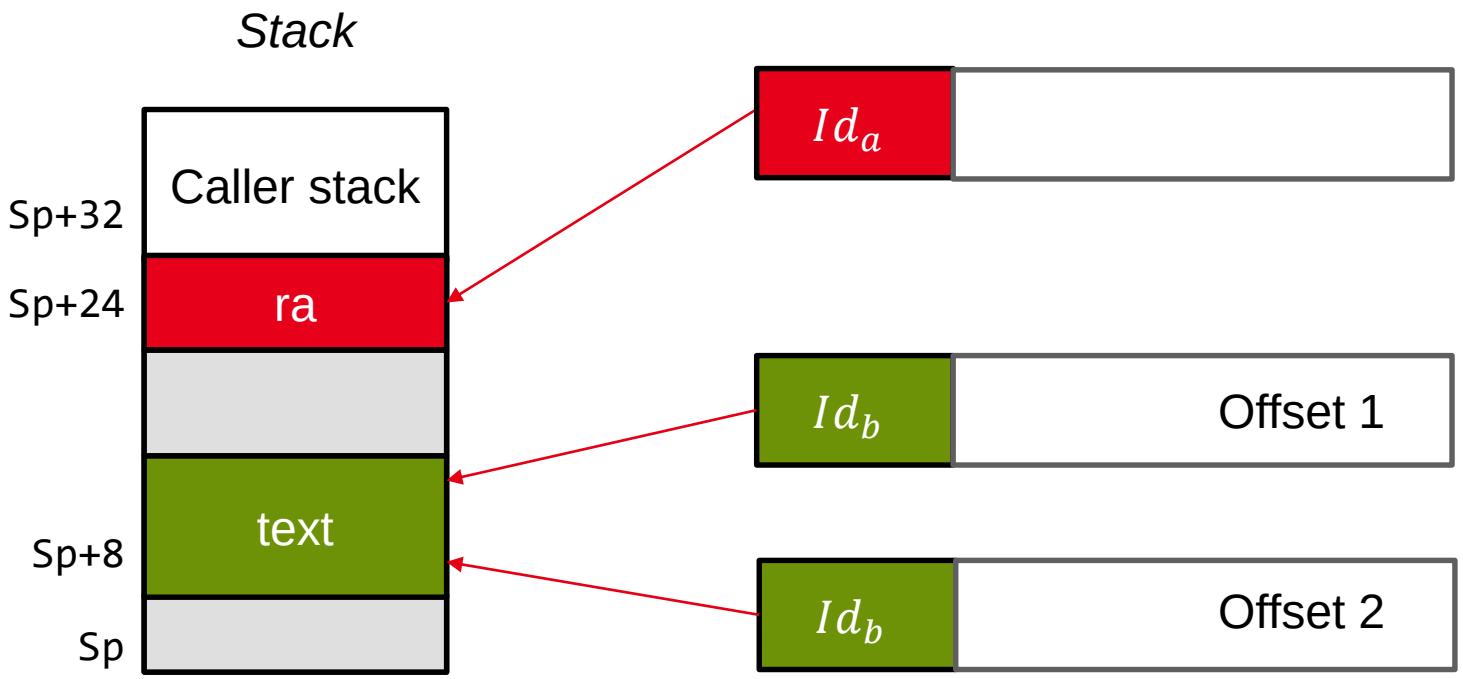
Pointer Integrity, existing techniques

- **Fat Pointers: replace all pointers with data structure (base pointer + identifier + size)**
- **Observation: not all bits of virtual addresses are used**
  - RV64: 39 bits, 48 bits or 64 bits MMU



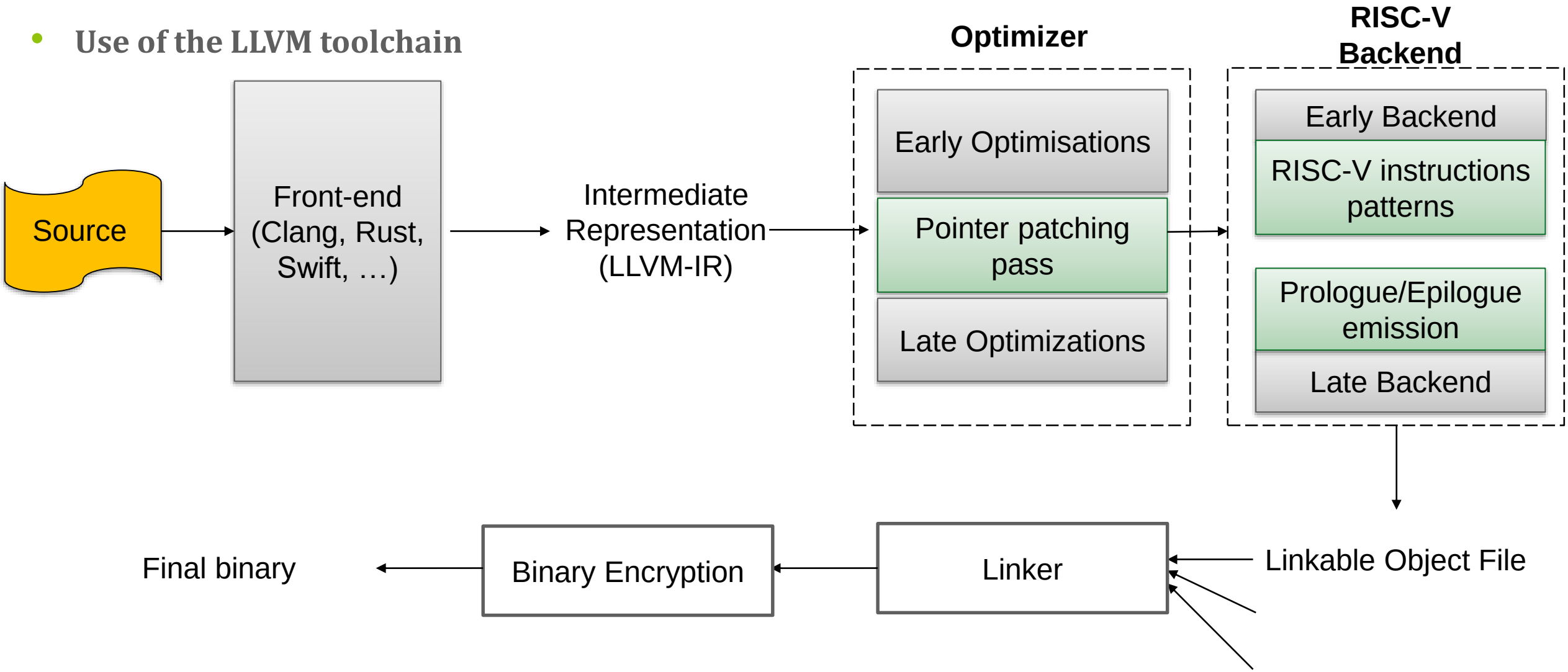
Encoding of memory region

- **Two transformations required**
  1. Setup region identifiers on every pointer: heap, stack, static data.
  2. Check identifiers before access (load/store) => done by hardware (stay tuned!)



Toolchain Support

- Use of the LLVM toolchain



Example

- **(Actually done on LLVM-IR, but simpler to explain on RISC-V)**
- **RISC-V ISA Extension**
  - Load/Store from stack with a specific identifier
  - Setptr: setup pointer identifier in one instruction

```
check_input(char const*):
        addi    sp,sp,-32
        mv      a1,a0
        addi    a0,sp,8
        sd      ra,24(sp)
        call    strcpy
        ld      ra,24(sp)
        li      a0,0
        addi    sp,sp,32
        jr      ra
```

```
# setup pointer ID=42
addi    a0, sp,8
Li      t0, 42
Slli    t0, t0, 48
Or      a0, a0, t0
```

$$a_0 \leftarrow Ptr \oplus (Id \ll 48)$$

```
addi    a0,sp,8
setptr  a0,a0,42
```

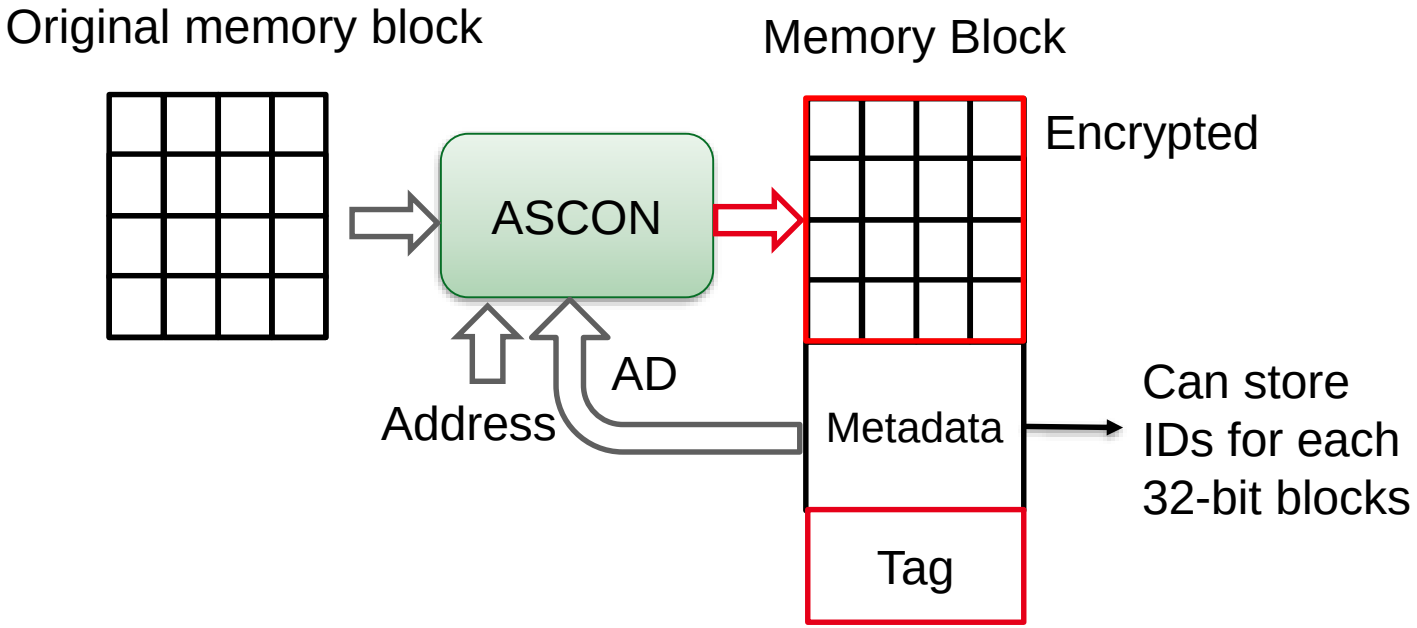
```
# Before load or store
Li      a1, 42
Srli    a0, a0, 48
bne     a0, a1, PTR_FAIL_HANDLER
ld      a0, 0(a0)
```

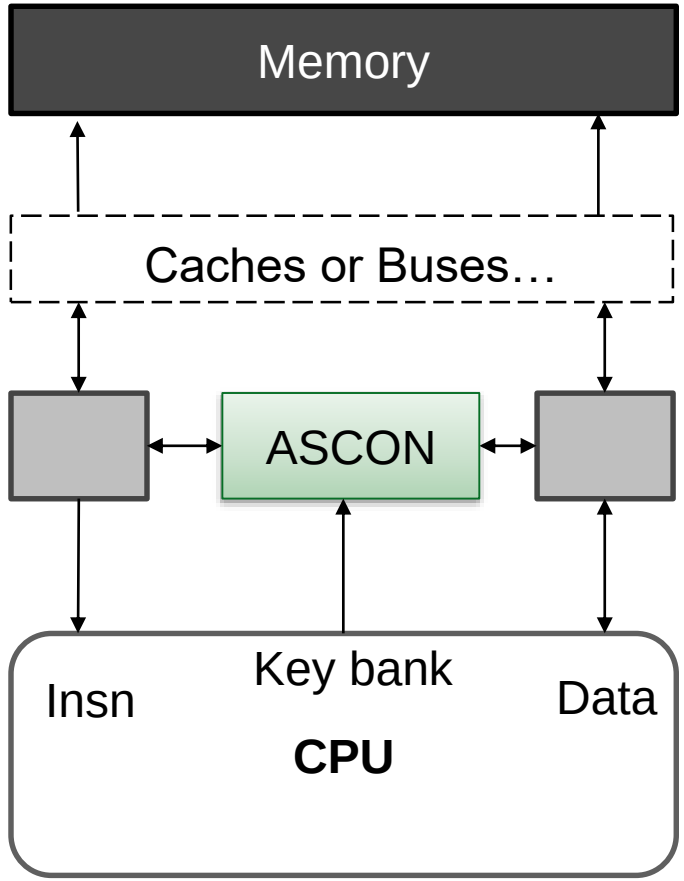
```
ld  a0, 0(a0)
```

Checked by hardware!

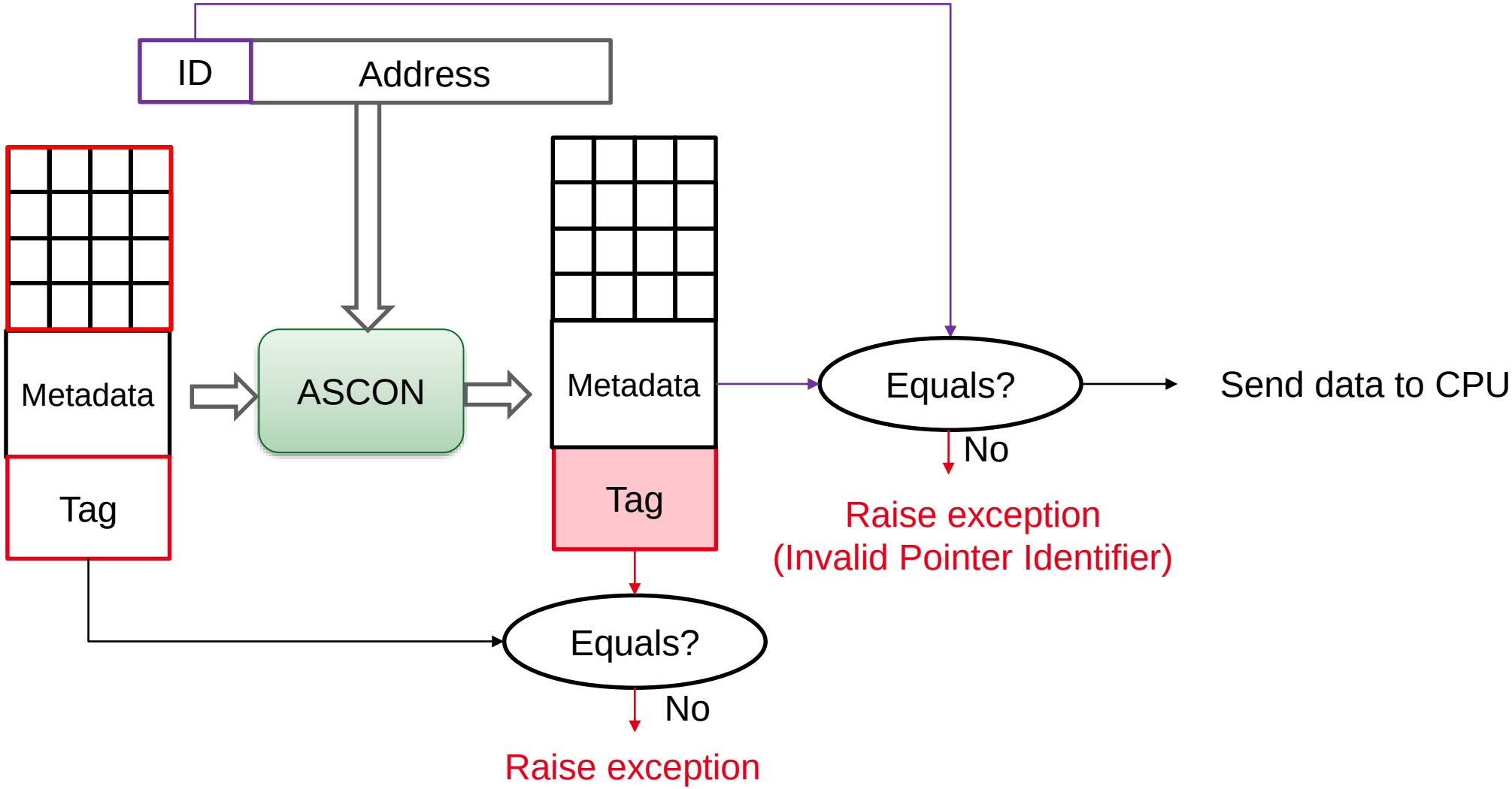
Hardware Pointer Checking

- CPU memory has encryption & integrity checking
- Memory is divided into blocks of 16 bytes



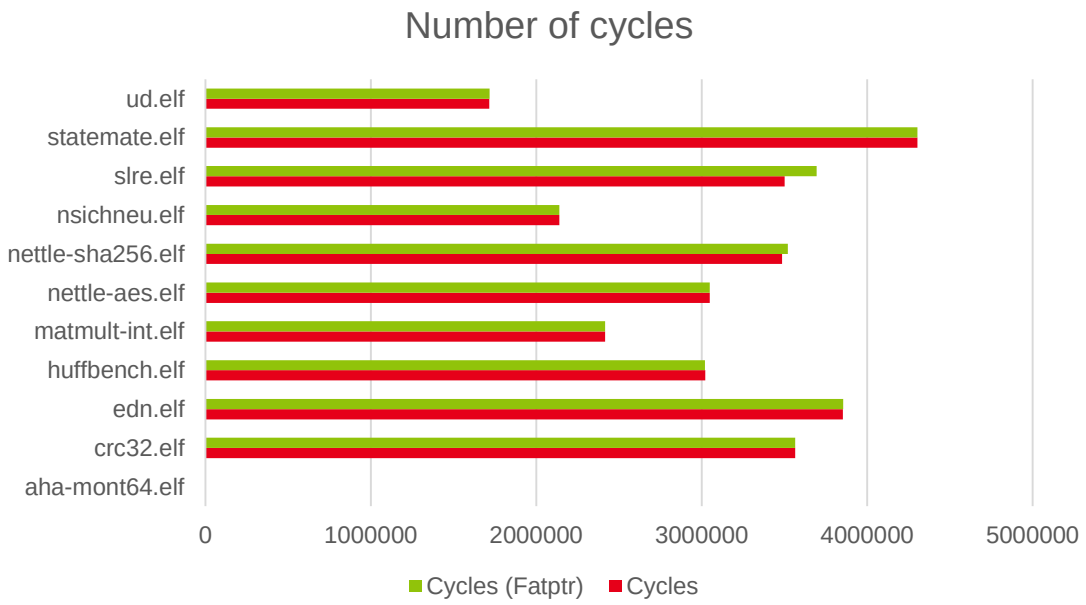


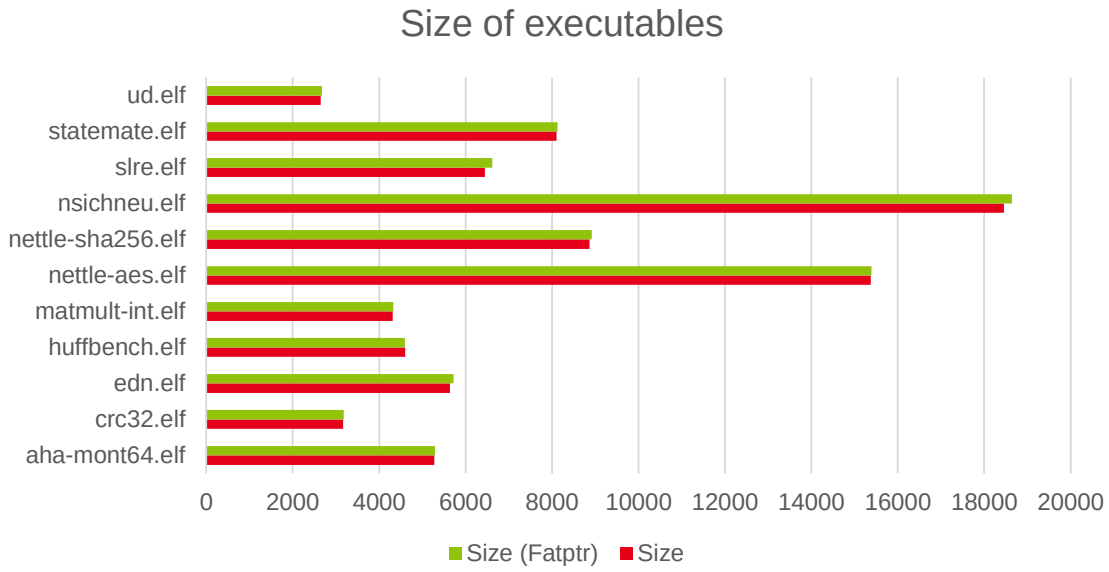
Hardware Checking (2)



Benchmarks

- Implemented & benchmark on an ISS
- Use of the Embench benchmarks
- Almost no overhead of Pointer Integrity => instructions are just replaced
- Encryption slowdown x2





Conclusion

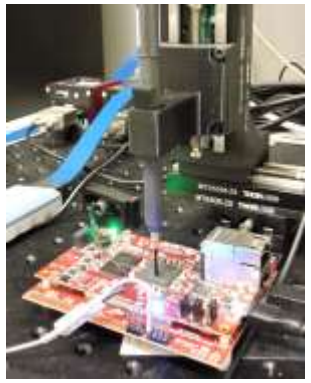
- **What our pointer integrity mechanism supports**
  - Heap pointers
  - Stack pointers
  - Static data (i.e., global variables)
  - Isolation of structure fields
  - Binary Compatibility

- **Current PoC on RISC-V 32**
  - Only 4 bits for encoding regions (16 max)
  - Working on a 64-bit implementation

- **Furtherwork**
  - Better integration with compiler optimizations
  - Formal Verification of countermeasures
  - Optimization of hardware-based checking

Thomas Hiscock (Thomas.hiscock@cea.fr)
Olivier Savry
Mustapha El-Majihi

LSOSP (Laboratoire Sécurité des Objets et des Systèmes Physiques), CEA-LETI, @Grenoble



https://www.emploi.cea.fr/

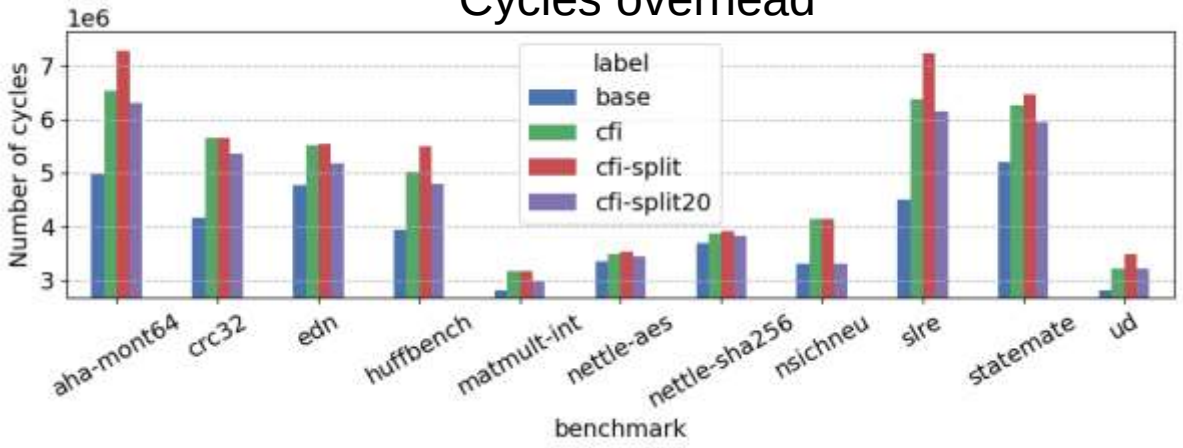
Questions?

Leti, technology research institute
Commissariat à l'énergie atomique et aux énergies alternatives
Minatec Campus | 17 avenue des Martyrs | 38054 Grenoble Cedex | France
www.leti-cea.com

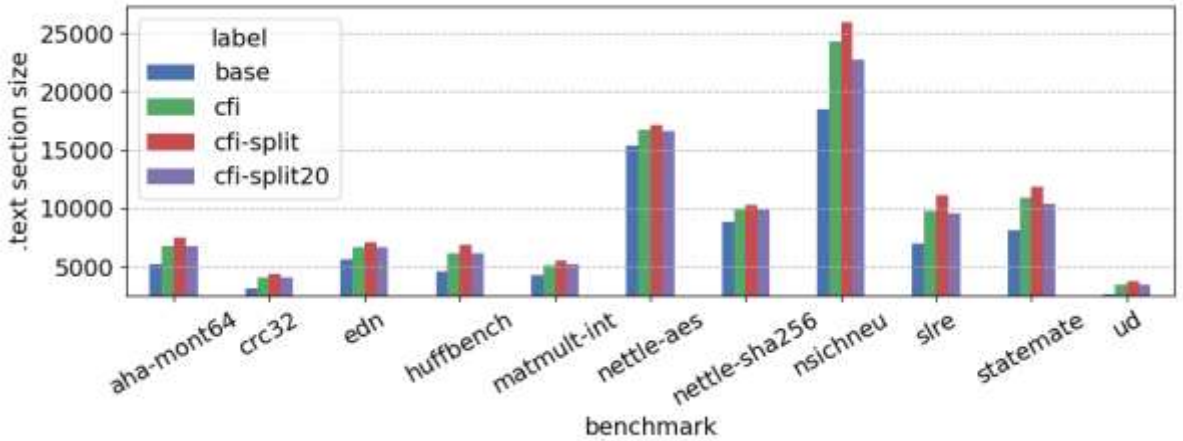
Performance results

- Use of the Embench [1] benchmark suite, compiled with –O2
- Exact cycle count obtained by a RTL simulation

Cycles overhead



Size overhead



- CFI Overhead
  - Directly linked to the number of branches
  - 20-bit mask variant is the most interesting
  - Cycles: +3% - %36%; Size: +7% - +36%
- CFI + Memory integrity
  - Memory usage: x2
  - Cycles: x2.32 – x3.27

[1] https://github.com/embench/embench-iot

Raw Results

Cycles overhead

benchmark	base	cfi	cfi-split	cfi-split20
aha-mont64	4,983,621	6,536,843 (+31.17%)	7,272,503 (+45.93%)	6,311,849 (+26.65%)
crc32	4,159,221	5,645,549 (+35.74%)	5,646,133 (+35.75%)	5,348,729 (+28.60%)
edn	4,778,450	5,510,499 (+15.32%)	5,545,619 (+16.05%)	5,171,863 (+8.23%)
huffbench	3,931,101	5,028,320 (+27.91%)	5,491,008 (+39.68%)	4,786,819 (+21.77%)
matmult-int	2,808,107	3,161,533 (+12.59%)	3,166,417 (+12.76%)	2,990,772 (+6.50%)
nettle-aes	3,342,817	3,478,669 (+4.06%)	3,527,257 (+5.52%)	3,448,319 (+3.16%)
nettle-sha256	3,692,382	3,880,247 (+5.09%)	3,907,251 (+5.82%)	3,813,947 (+3.29%)
nsichneu	3,304,837	4,131,404 (+25.01%)	4,131,404 (+25.01%)	3,307,563 (+0.08%)
slre	4,510,608	6,376,640 (+41.37%)	7,225,528 (+60.19%)	6,144,376 (+36.22%)
statemate	5,212,495	6,261,268 (+20.12%)	6,458,060 (+23.90%)	5,939,445 (+13.95%)
ud	2,803,095	3,224,836 (+15.05%)	3,487,836 (+24.43%)	3,214,037 (+14.66%)

Size overhead

benchmark	base	cfi	cfi-split	cfi-split20
aha-mont64	5,264	6,756 (+28.34%)	7,548 (+43.39%)	6,776 (+28.72%)
crc32	3,152	4,120 (+30.71%)	4,384 (+39.09%)	4,096 (+29.95%)
edn	5,624	6,652 (+18.28%)	7,060 (+25.53%)	6,680 (+18.78%)
huffbench	4,584	6,216 (+35.60%)	6,900 (+50.52%)	6,156 (+34.29%)
matmult-int	4,296	5,180 (+20.58%)	5,504 (+28.12%)	5,204 (+21.14%)
nettle-aes	15,360	16,708 (+8.78%)	17,092 (+11.28%)	16,580 (+7.94%)
nettle-sha256	8,855	9,955 (+12.42%)	10,351 (+16.89%)	9,935 (+12.20%)
nsichneu	18,440	24,312 (+31.84%)	25,980 (+40.89%)	22,696 (+23.08%)
slre	7,002	9,834 (+40.45%)	11,130 (+58.95%)	9,542 (+36.28%)
statemate	8,092	10,976 (+35.64%)	11,900 (+47.06%)	10,420 (+28.77%)
ud	2,632	3,504 (+33.13%)	3,804 (+44.53%)	3,484 (+32.37%)