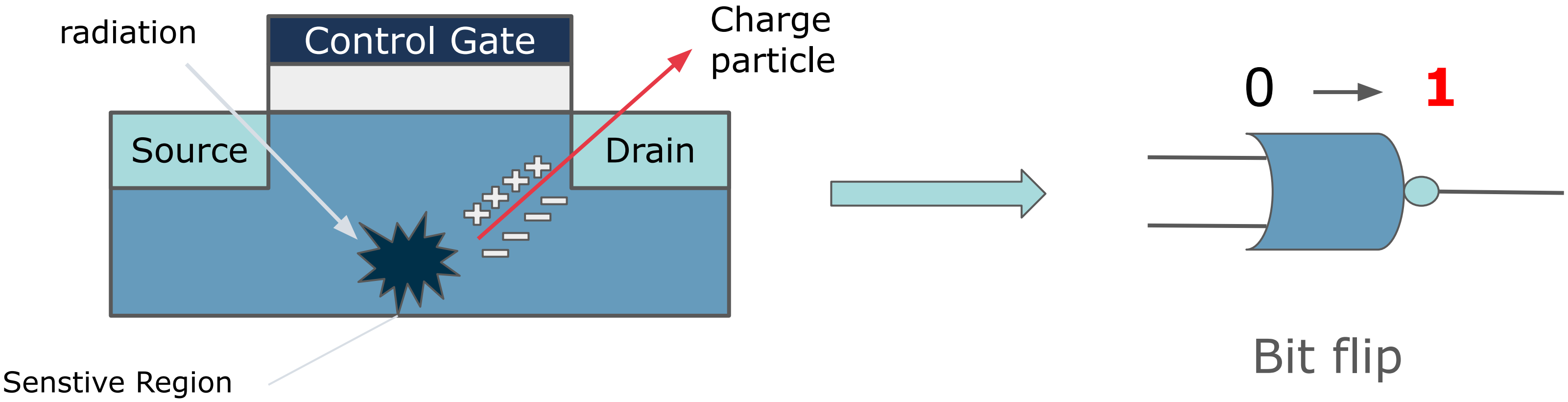
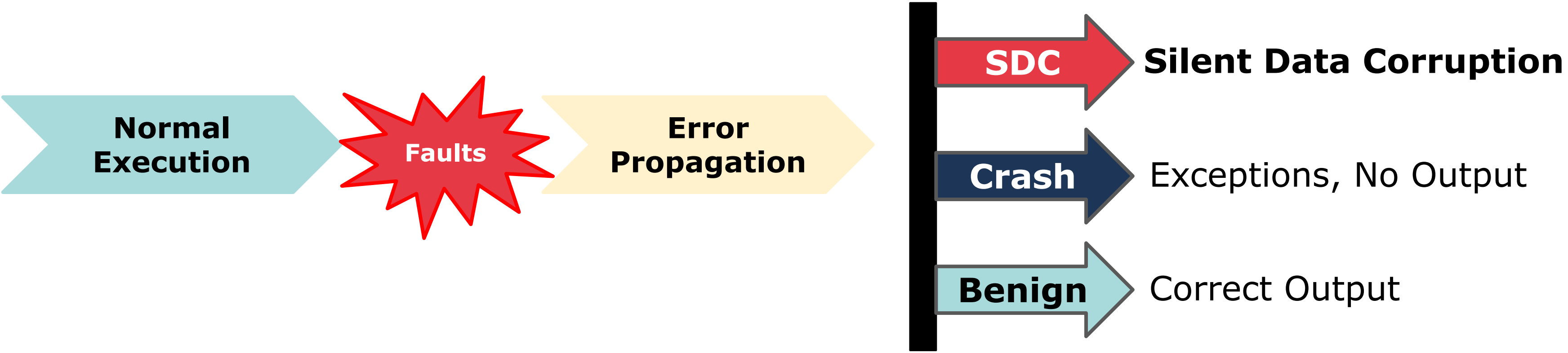
A Fast Low-Level Error Detection Techinque

<u>Zhengyang He</u> Hui Xu Guanpeng Li

The University of Iowa Fudan University

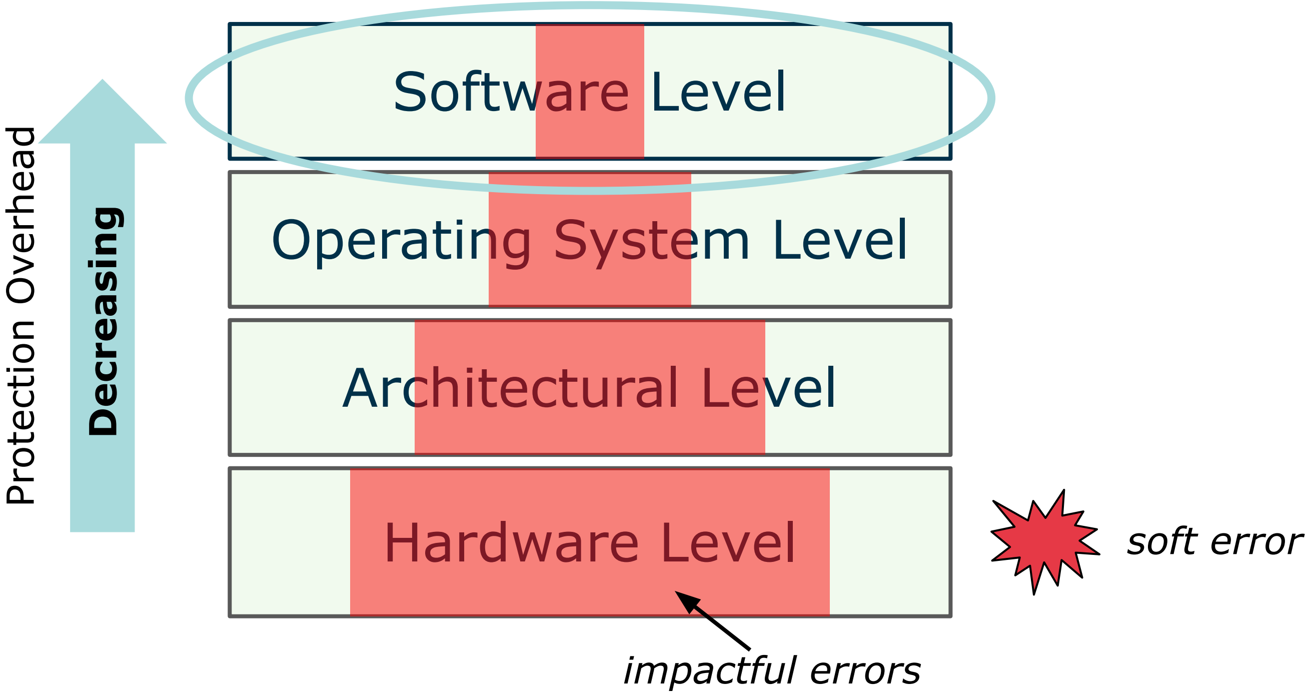
Soft Errors


radiation
Control Gate
Charge particle
Source
Drain
Senstive Region
0
1
Bit flip


Normal Execution
Faults
Error Propagation
SDC
Silent Data Corruption
Crash
Exceptions, No Output
Benign
Correct Output

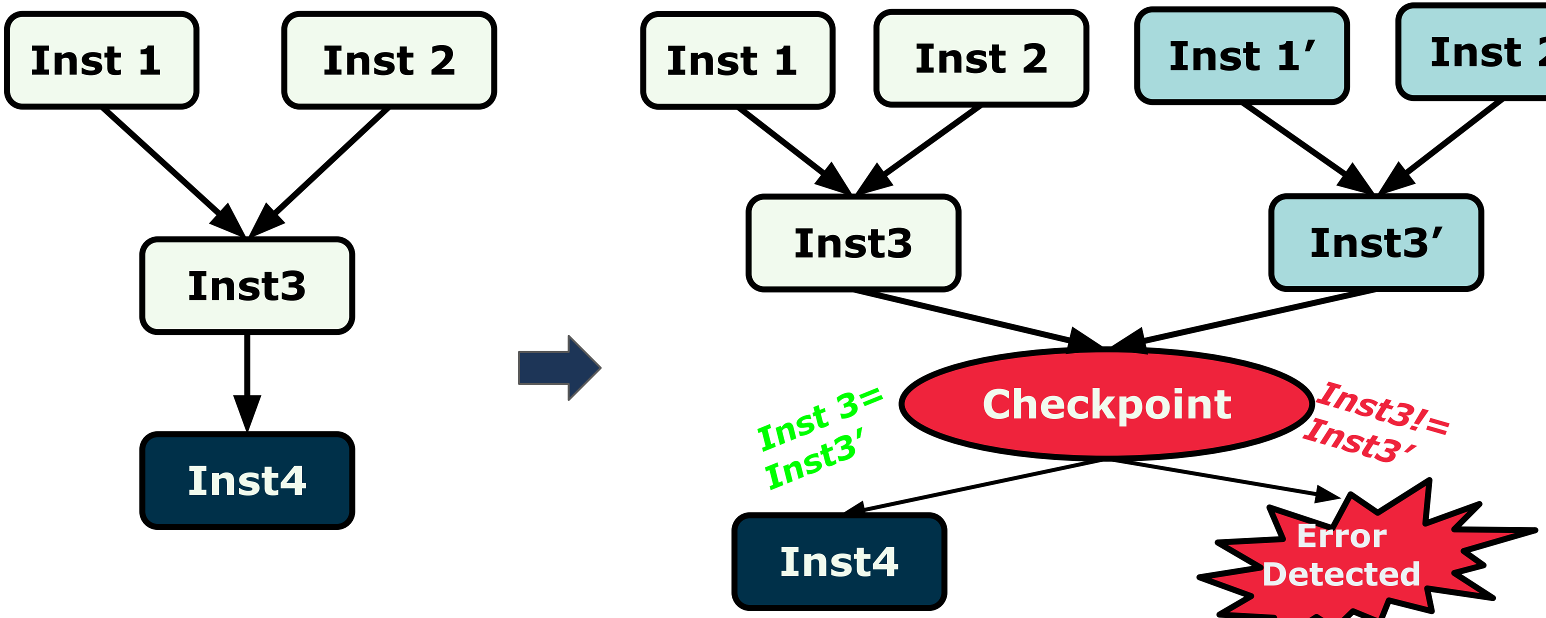
Software Solutions

- **Software solution** is more flexible and cost-effective.



Error Detection by Duplicating Instructions (EDDI)

- **EDDI** duplicates instruction at *compile time* and detects errors at *run time*.
- Compiler-level transformation, hence **program-agnostic**.

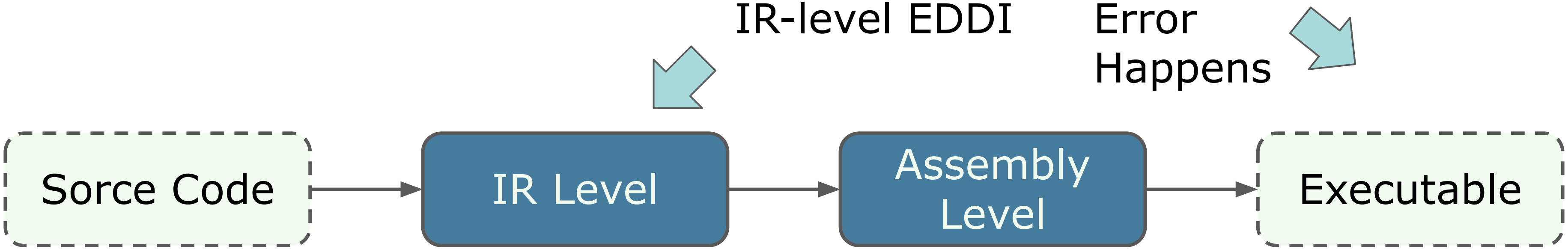


original code

EDDI-protected code

Problems and Motivation

- Existing EDDI suffers from loss of SDC coverage
  - IR-level EDDI fails in realistic fault injection scenarios
- Existing EDDI incurs high runtime overhead



IR-level EDDI

Error Happens

Sorce Code

IR Level

Assembly Level

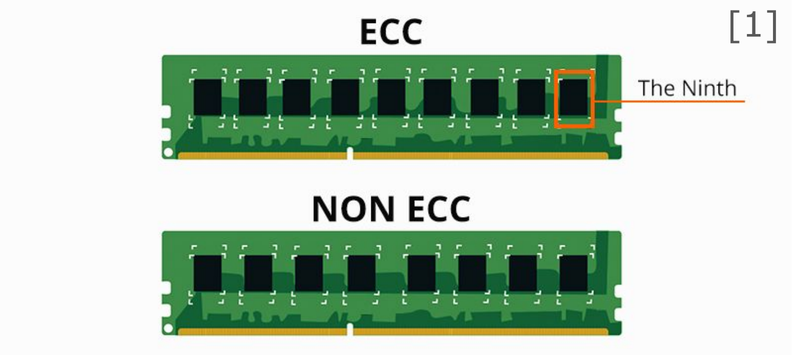
Executable

Transformation

IR-level EDDI may not provide full protection on Assembly-level!

Fault Model

- Single-bit flip, which is accurate enough to evaluate SDC
- Errors in computation units/data path
- One fault per program execution
- Memory errors can be protected by ECC, so do not consider



[1]: https://community.fs.com/article/ecc-vs-non-ecc-memory-which-one-is-better.html

Fault Injection Methodology

🟩 IR level Code

🟦 Assembly Code

- ***IR***: Instructions that contains return value

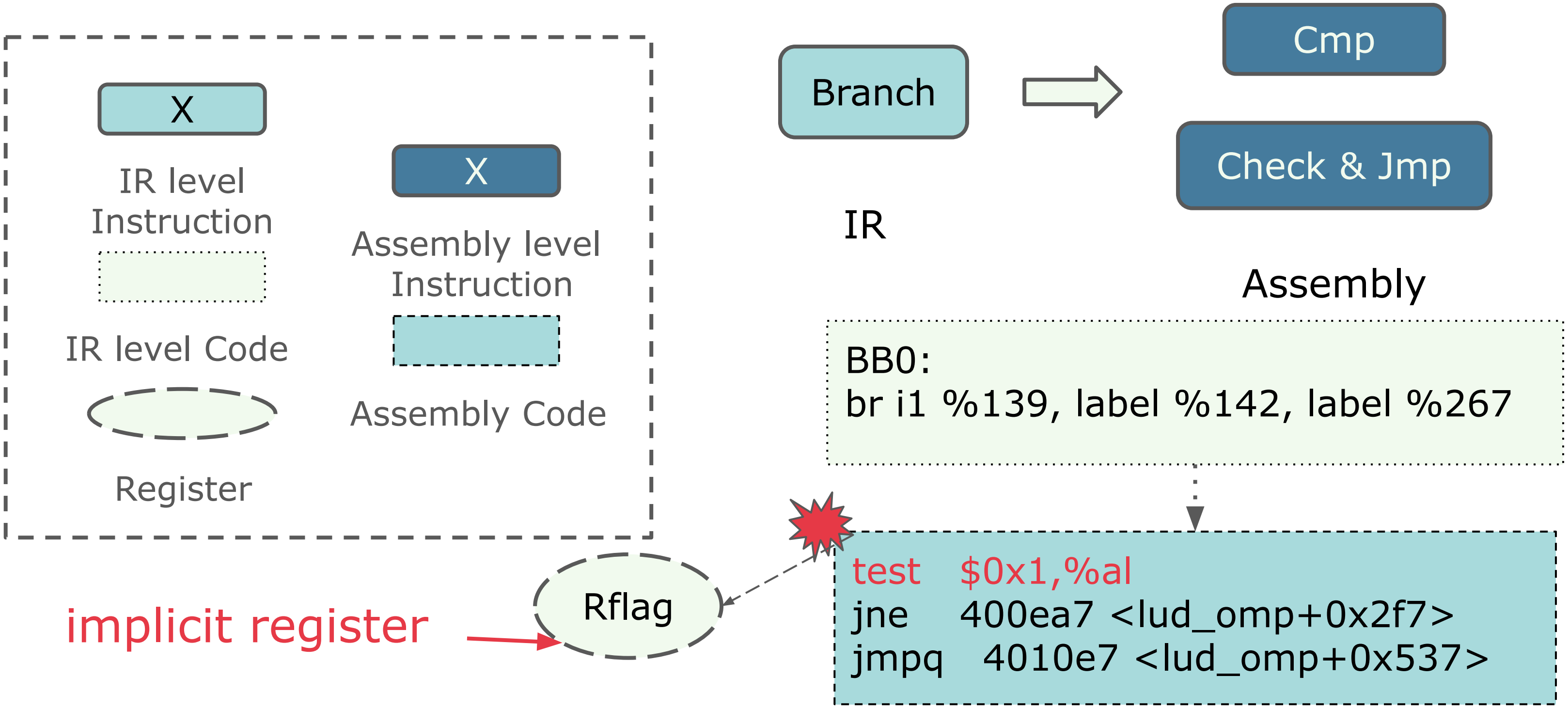
```
%3 = icmp slt i32 %1, %2
%4 = load i32* %1, align 4
%5 = mul i64 1, %4
                    …
```

- ***Assembly***: Instructions whose computation destination is a register

```
test   $0x1,%al
sub     %ecx,%eax
mov   -0x40(%rbp),%rax
                    …
```

Motivation

- IR-level EDDI fails in realistic fault injection scenarios



X

IR level Instruction

X

Assembly level Instruction

IR level Code

Assembly Code

Register

Branch

Cmp

Check & Jmp

IR

Assembly

```
BB0:
br i1 %139, label %142, label %267
```

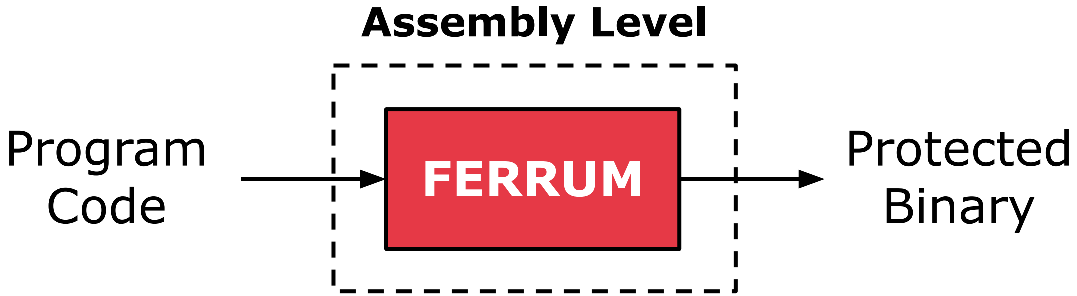
```
test   $0x1,%al
jne    400ea7 <lud_omp+0x2f7>
jmpq   4010e7 <lud_omp+0x537>
```

implicit register

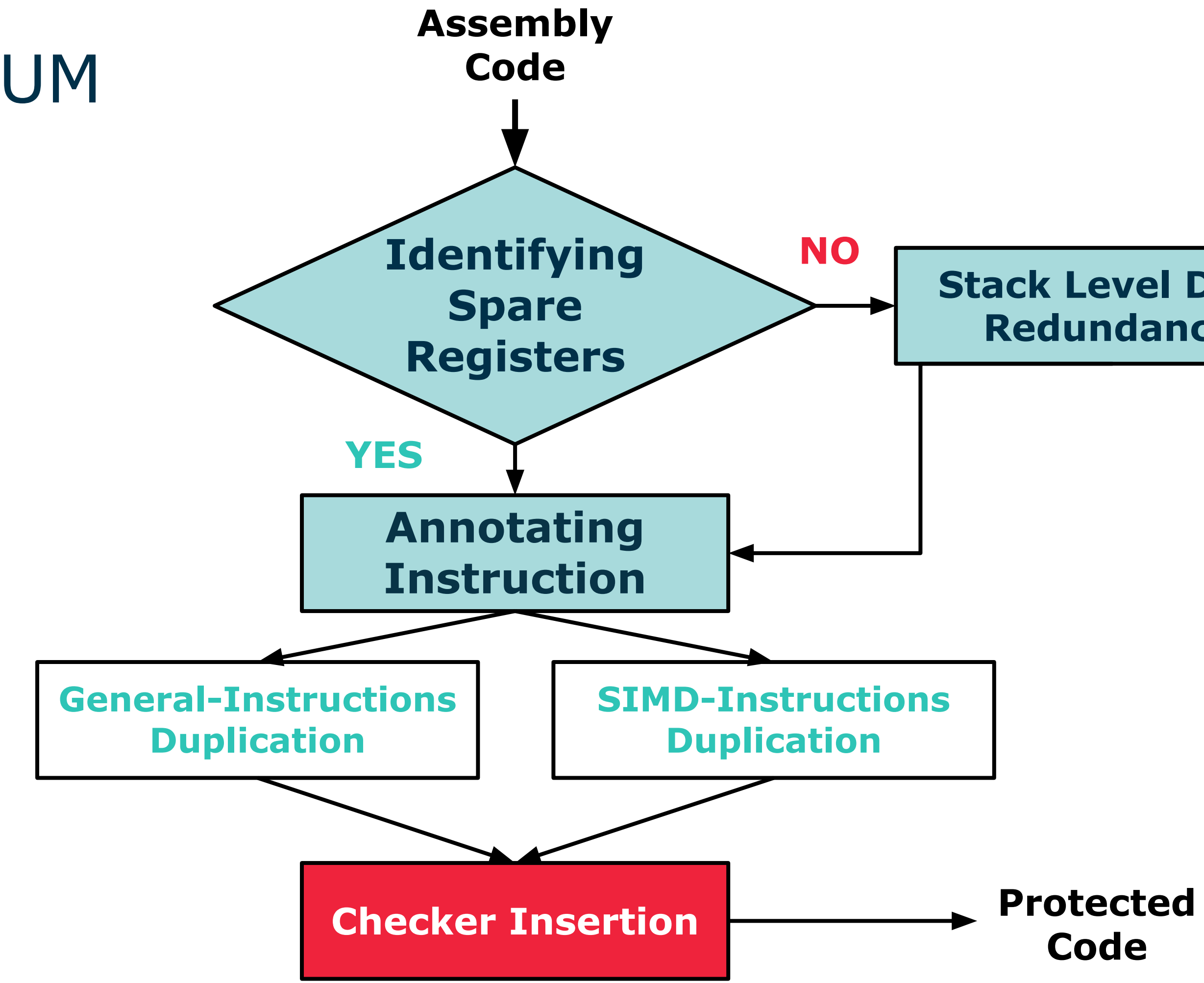
Rflag

FERRUM

- Goal: Protect soft errors at assembly level
- Obsevations:
  - Performance
    - Duplicating and verifying at assembly instructions
    - Leveraging SIMD in modern processors
  - Coverage
    - Cross-layer fault injection analysis

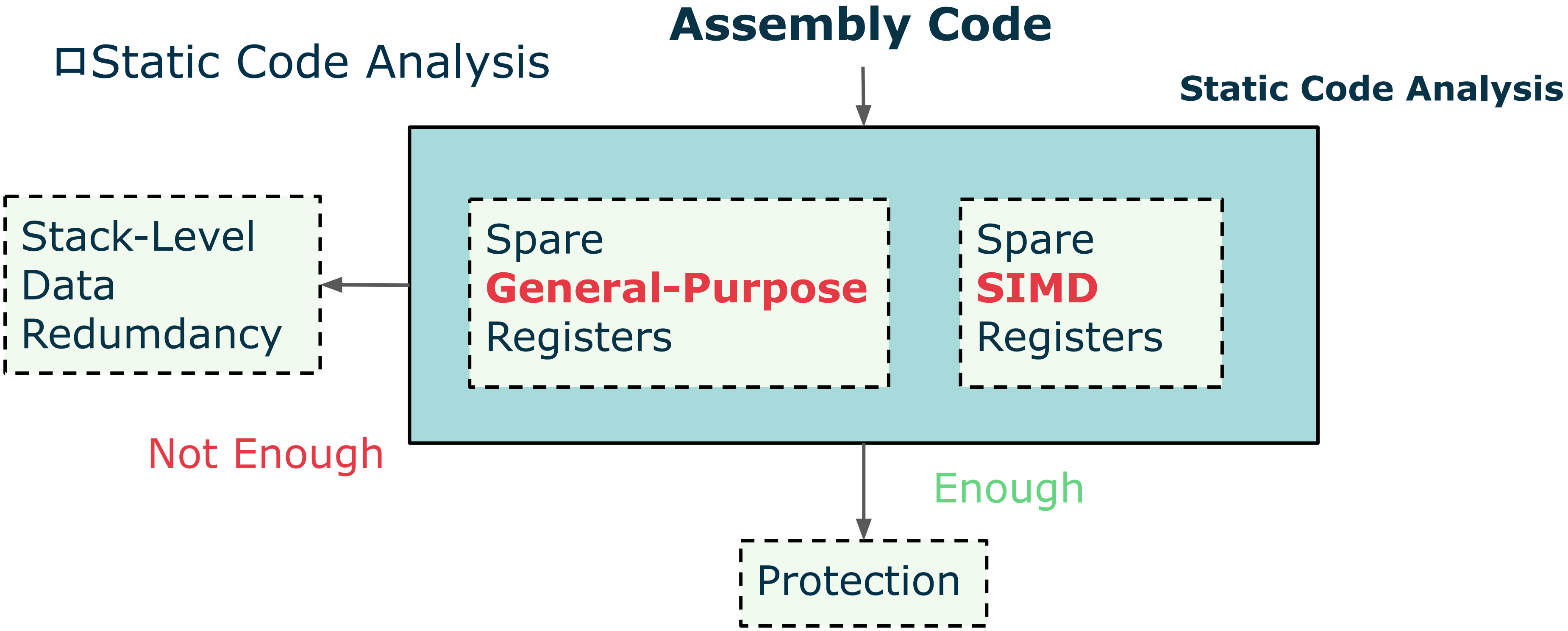


FERRUM


Assembly Code
Identifying Spare Registers
NO
Stack Level Data Redundancy
YES
Annotating Instruction
General-Instructions Duplication
SIMD-Instructions Duplication
Checker Insertion
Protected Code

FERRUM

- □Static Code Analysis



Assembly Code

Static Code Analysis

Spare **General-Purpose** Registers

Spare **SIMD** Registers

Stack-Level Data Redumdancy

Not Enough

Enough

Protection

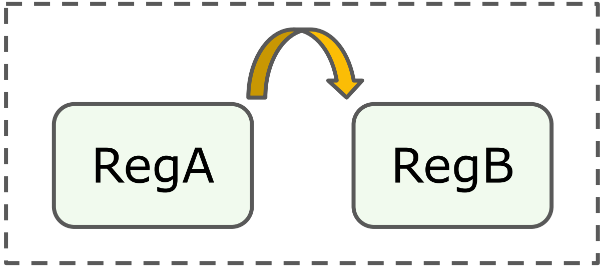
FERRUM

- Duplication for General Instructions

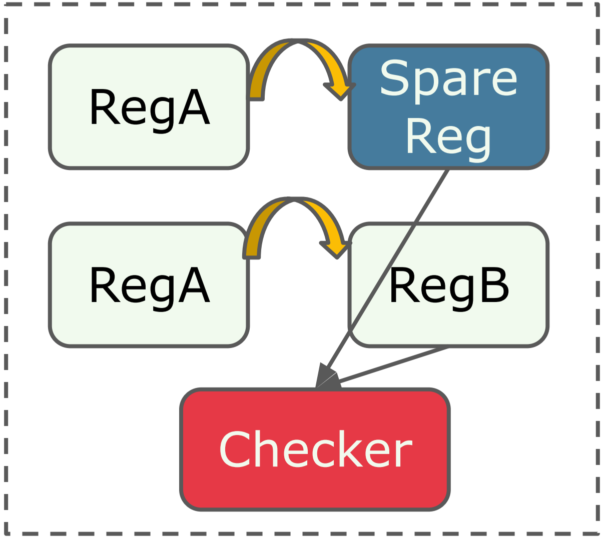
Code Example

```
.LBB0_3:
...
movslq  %ecx, %r10
movslq  %ecx, %rcx
xorq    %rcx, %r10
jne exit_function
...
```

Original Instruction



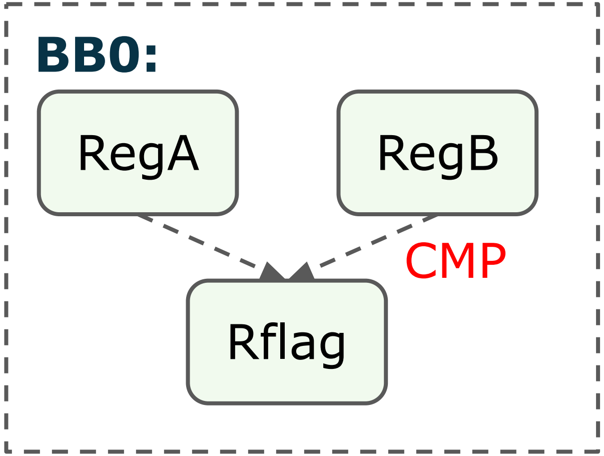
Duplication Process



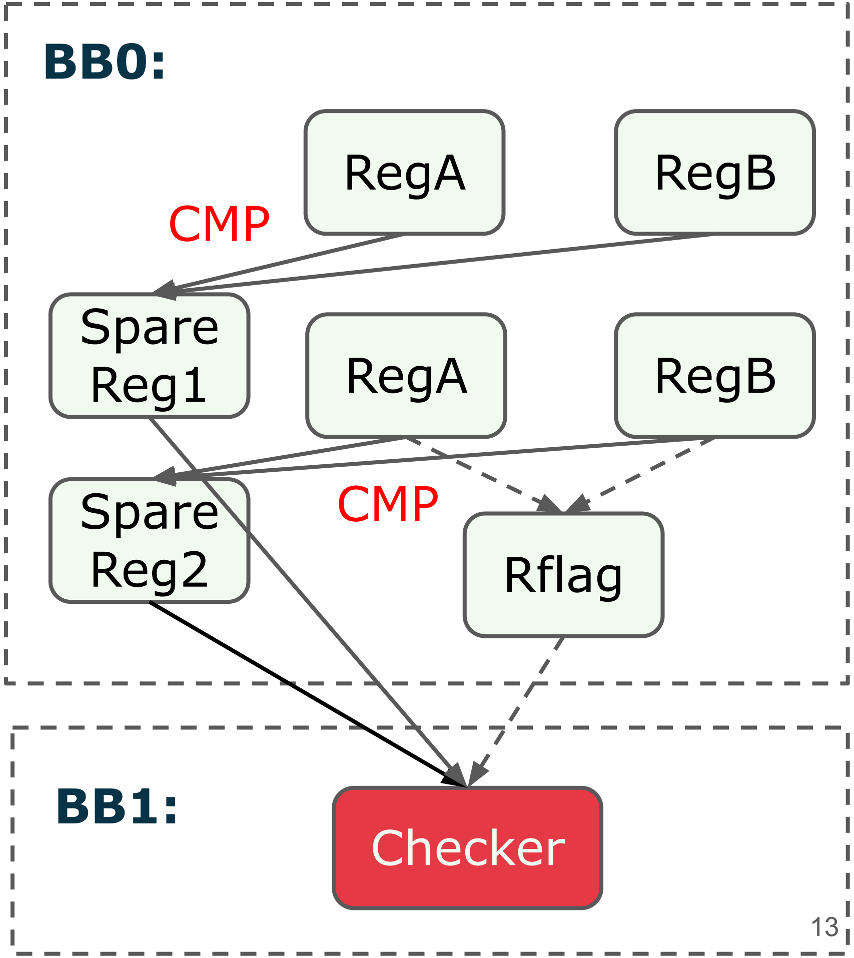
FERRUM

- Duplication for General Instructions

Original Instruction



Duplication Process



FERRUM

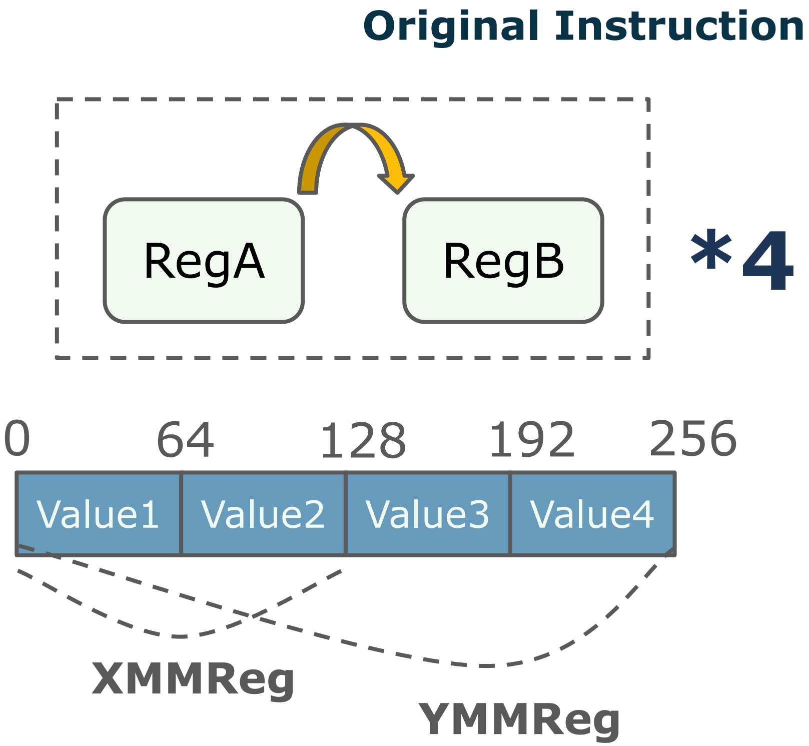
- Duplication for General instructions

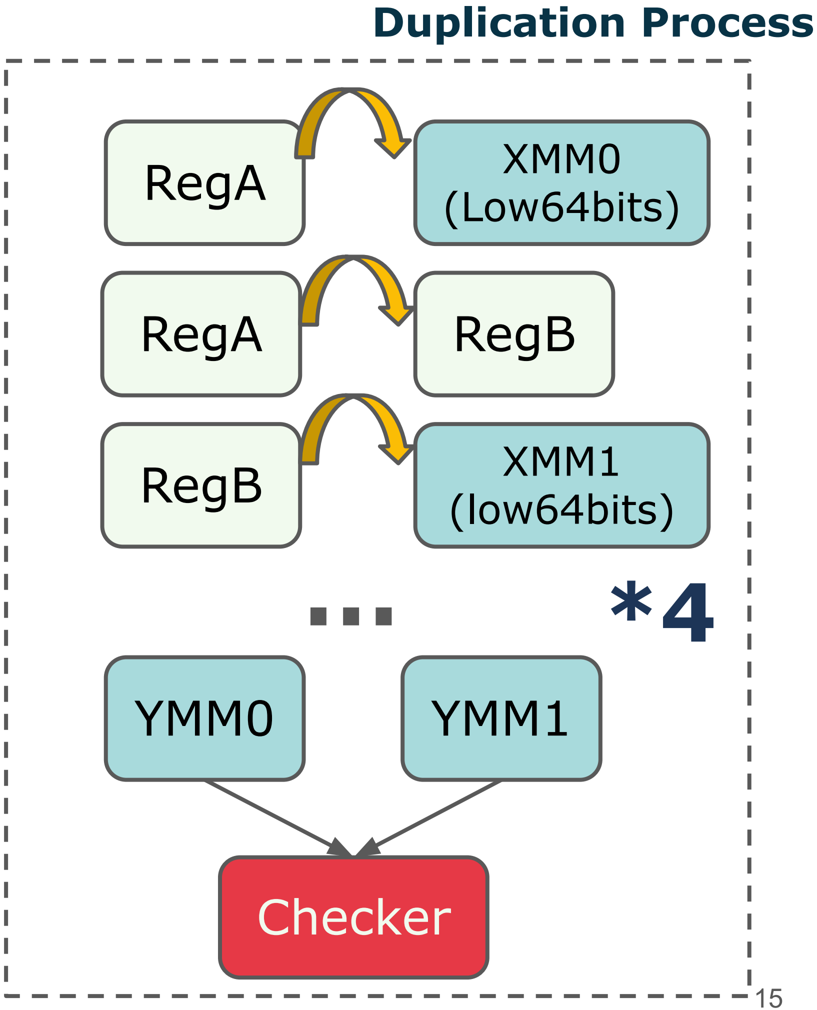
Code Example

```
.LBB7_3:
…
cmpl  -12(%rbp), %eax
sete  %r11b #set original flag
cmpl  -12(%rbp), %eax
sete  %r12b #set duplication flag
jl  .LBB7_4
…
.LBB7_4:
xor  %r11b, %r12b #check flag value
jne exit_function
…
```

FERRUM

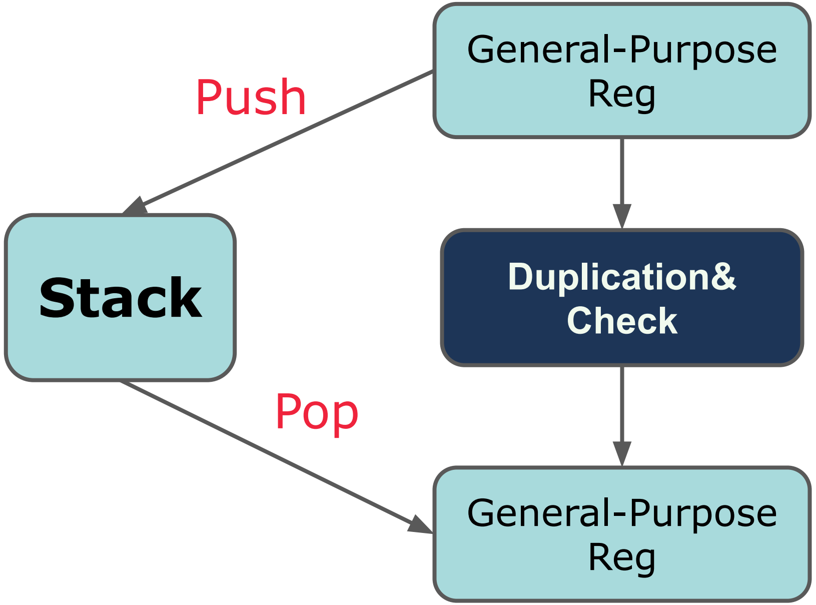
- Duplication for SIMD instructions





FERRUM

- Stack-level data redundancy



Code Example

```
.LBB1_40:
push    %r10    #get temporary use
...
movslq      -68(%rbp), %r10
movslq      -68(%rbp), %rax
cmpq        %rax, %r10
jne exit_function
...
pop     %r10    #reload to previous value
```

Evaluation

- Baseline Technique

	Basic	Store	Branch	Call	Mapping	Comparison
IR-EDDI	IR	\	\	\	\	\
Hybrid-AS-EDDI	AS	AS	IR	AS	AS	IR
FERRUM	AS	AS	AS	AS	AS	AS

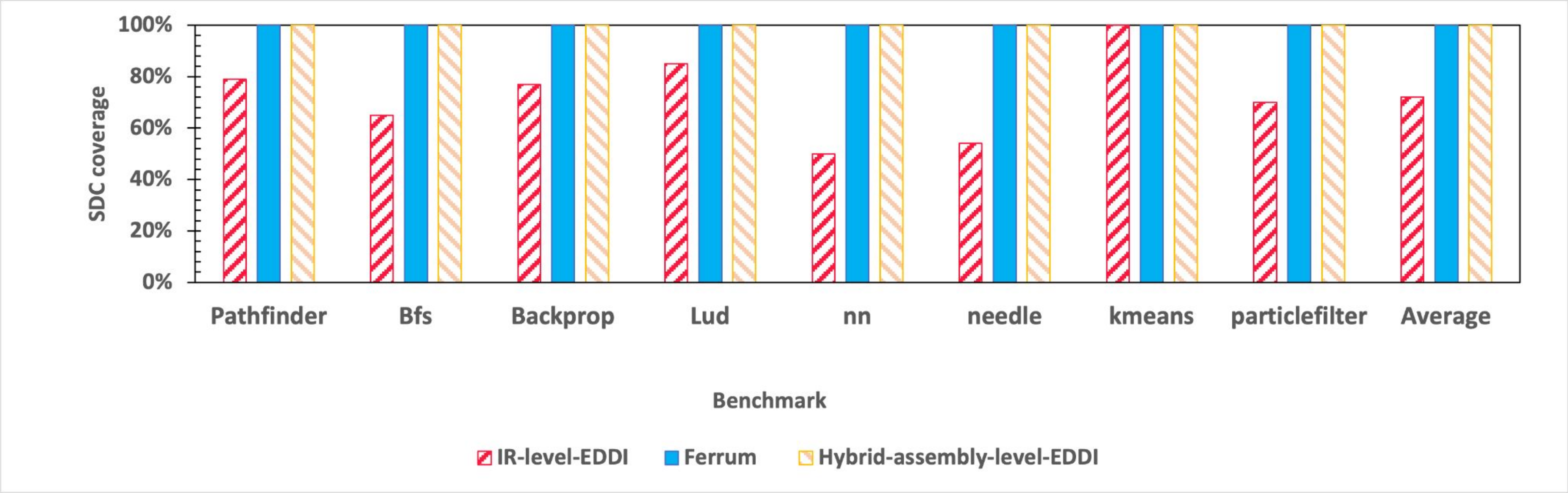
IR: The protection is implemented at IR level

AS: The protection is implemented at Assembly level **without SIMD**

AS: The protection is implemented at Assembly level **with SIMD**

Evaluation

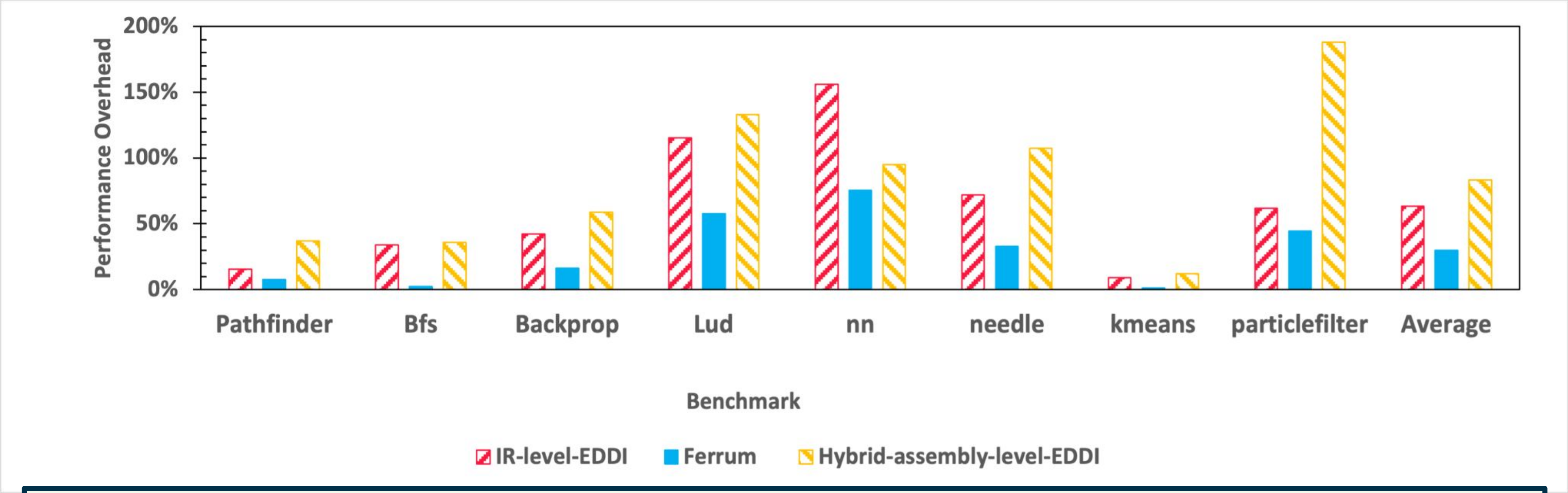
- SDC Coverage



Soft Error Protection efficiency is significantly enhanced

Evaluation

- Performance Overhead



Performance is significantly accelarated

Conclusion

□ We propose **FERRUM**, an assembly level error detection techinque.

□ FERRUM can achieve **100%** protection efficincy at assembly level.

□ FERRUM utilizes **SIMD** to optimize and enhance performance.

□ FERRUM can run **less performance overhead** than baseline techinques.

IOWA

復旦大學 FUDAN UNIVERSITY