Practical Experience Report

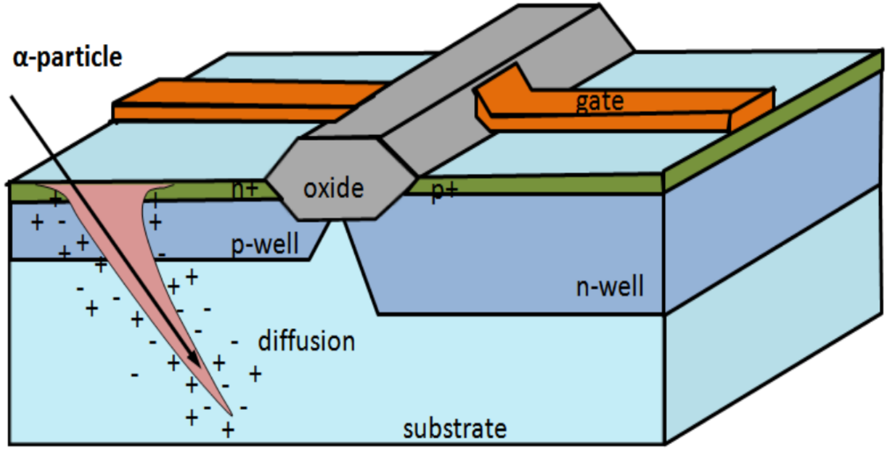
# A TALE OF TWO INJECTORS: END-TO-END COMPARISON OF IR-LEVEL AND ASSEMBLY-LEVEL FAULT INJECTION

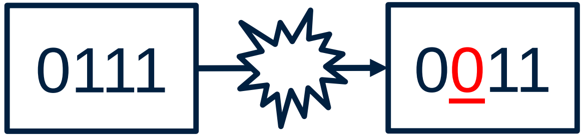
Lucas Palazzi

Co-authors: Guanpeng Li, Bo Fang, and Karthik Pattabiraman

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
THE UNIVERSITY OF BRITISH COLUMBIA

# MOTIVATION: SOFT ERRORS




0111

0011

**Becoming more common in processors**

Photo source: *http://aviral.lab.asu.edu/soft-error-resilience/*

# SOFT ERROR OUTCOMES

1. Benign error
2. Crash
3. Silent data corruption (SDC)

# SOFT ERROR OUTCOMES

1. Benign error

2. Crash

3. Silent data corruption (SDC) ❊ e.g., integer sort program

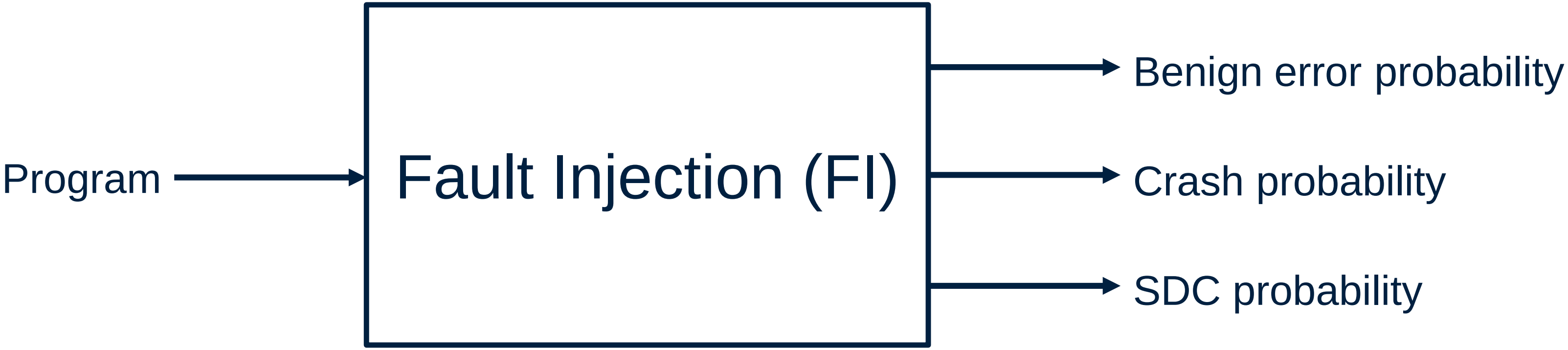
*Error-free program output:*

```
1, 4, 6, 8, 10
```

*SDC program output:*

```
6, 4, 1, 8, 10
```

# FAULT INJECTION



Program

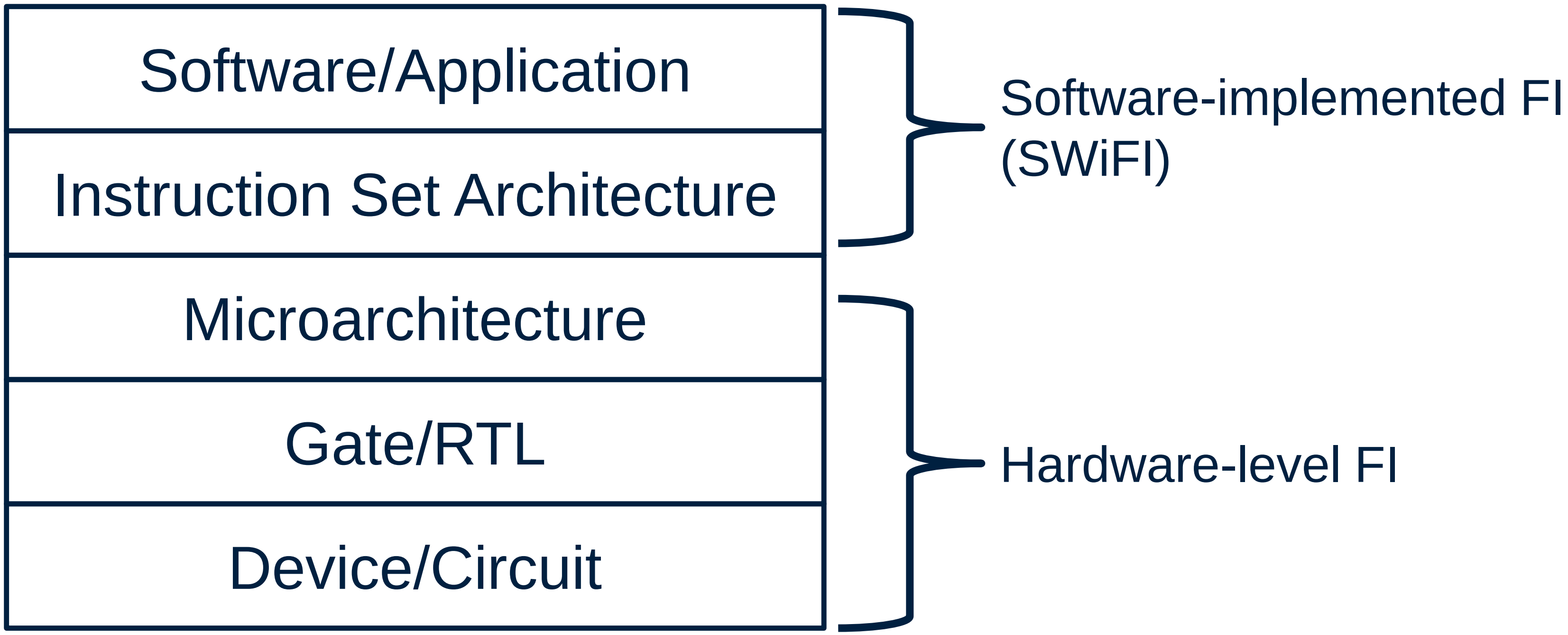
Fault Injection (FI)

Benign error probability

Crash probability

SDC probability

# FI AT DIFFERENT LEVELS OF ABSTRACTION



Software/Application

Instruction Set Architecture

Microarchitecture

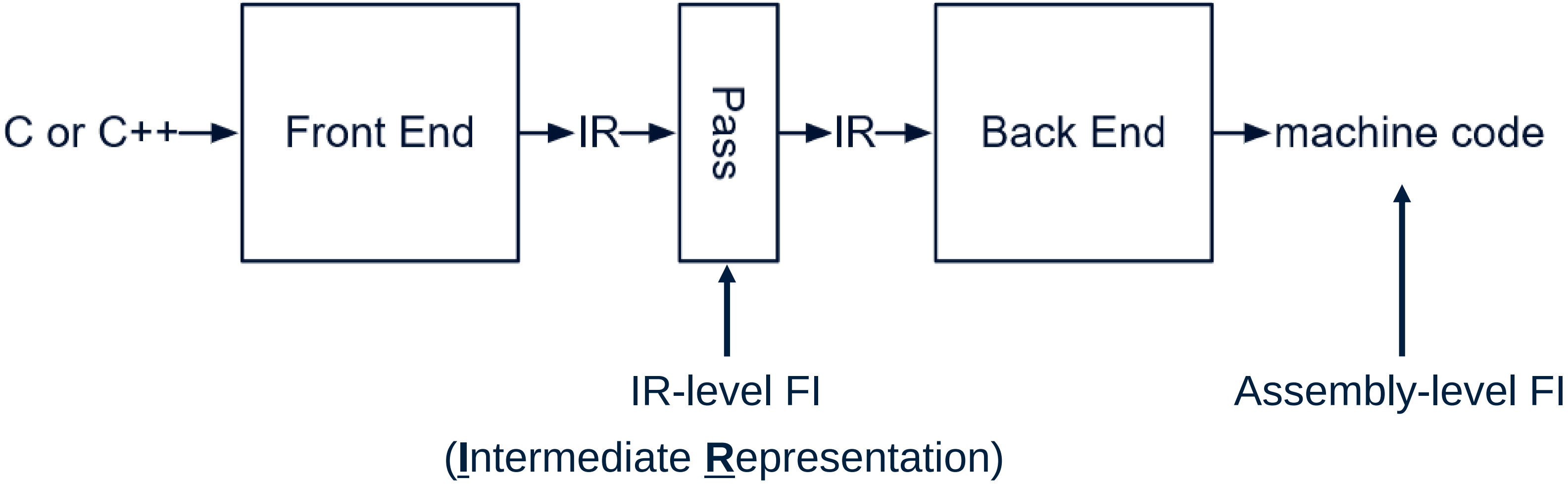
Gate/RTL

Device/Circuit

Software-implemented FI (SWiFI)

Hardware-level FI

# SOFTWARE-IMPLEMENTED FI (SWiFI)



C or C++

Front End

IR

Pass

IR

Back End

machine code

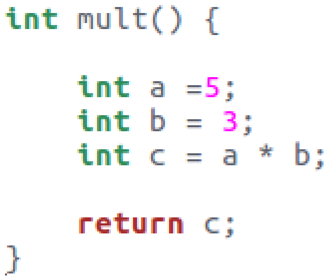
IR-level FI

(**I**ntermediate **R**epresentation)

Assembly-level FI

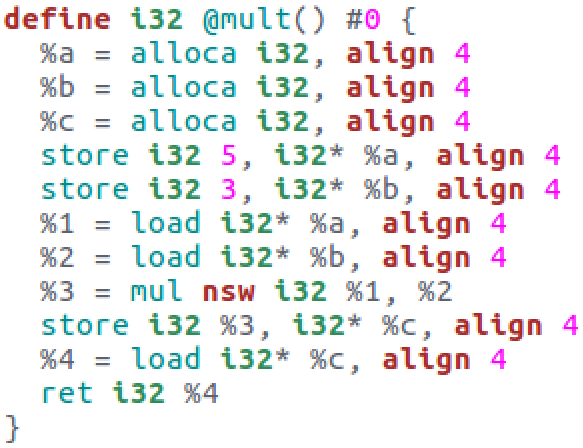
# CODE COMPILATION EXAMPLE

## C Source



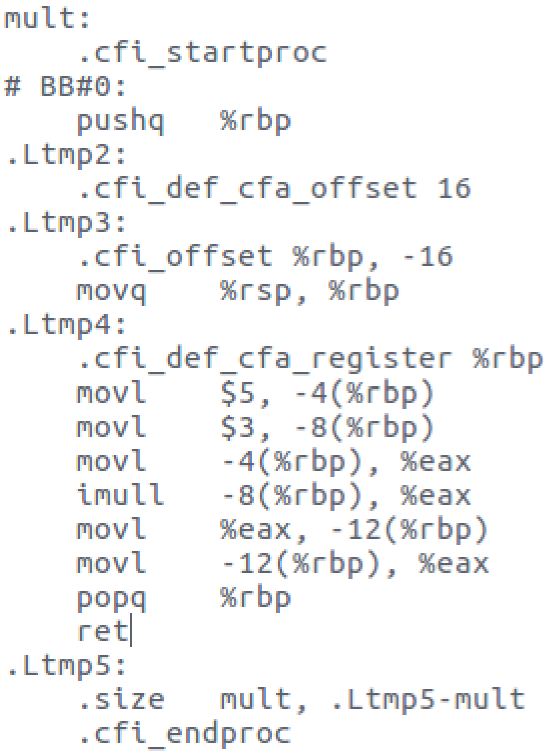
```c
int mult() {

    int a =5;
    int b = 3;
    int c = a * b;

    return c;
}
```

## LLVM IR



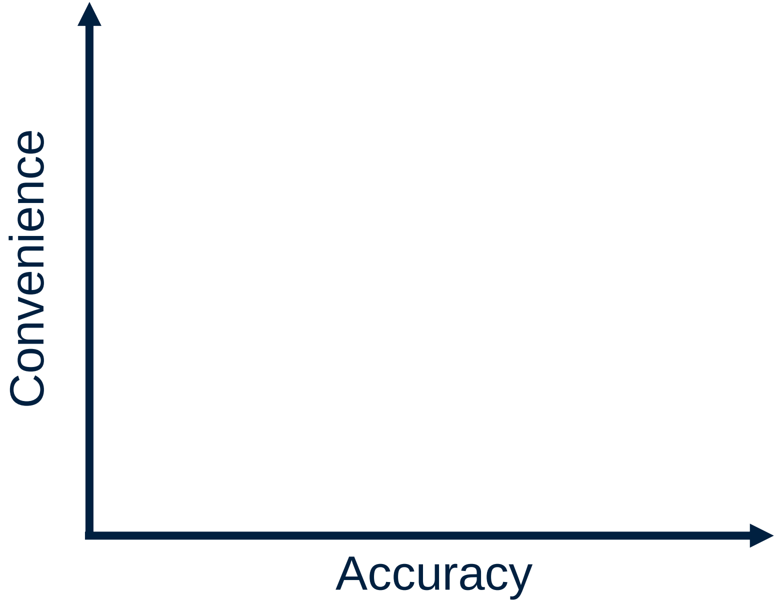
```
define i32 @mult() #0 {
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  store i32 5, i32* %a, align 4
  store i32 3, i32* %b, align 4
  %1 = load i32* %a, align 4
  %2 = load i32* %b, align 4
  %3 = mul nsw i32 %1, %2
  store i32 %3, i32* %c, align 4
  %4 = load i32* %c, align 4
  ret i32 %4
}
```

## x86 Assembly

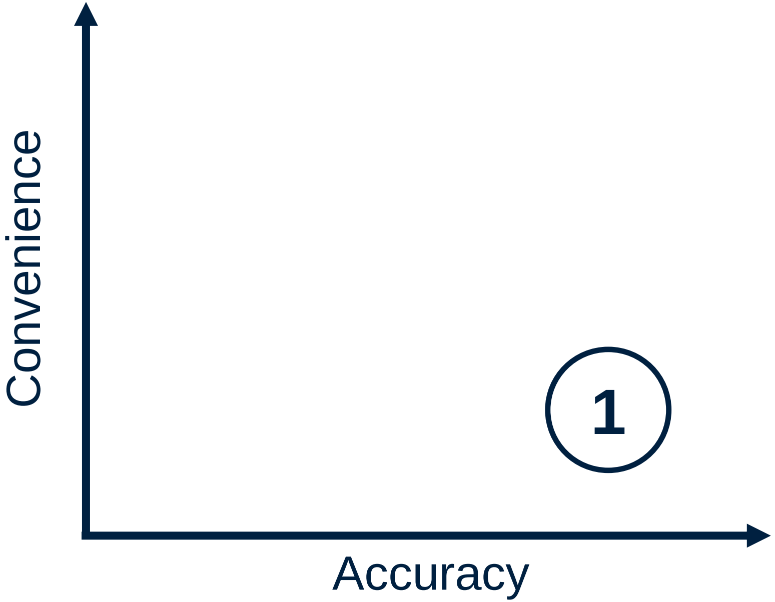


```
mult:
    .cfi_startproc
# BB#0:
    pushq   %rbp
.Ltmp2:
    .cfi_def_cfa_offset 16
.Ltmp3:
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
.Ltmp4:
    .cfi_def_cfa_register %rbp
    movl    $5, -4(%rbp)
    movl    $3, -8(%rbp)
    movl    -4(%rbp), %eax
    imull   -8(%rbp), %eax
    movl    %eax, -12(%rbp)
    movl    -12(%rbp), %eax
    popq    %rbp
    ret
.Ltmp5:
    .size   mult, .Ltmp5-mult
    .cfi_endproc
```

# TRADE-OFFS OF DIFFERENT SWiFI TECHNIQUES

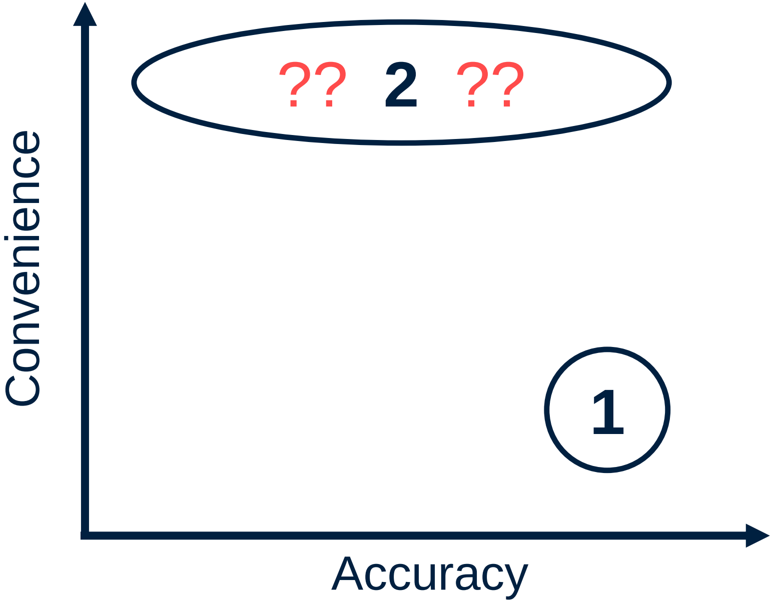


# TRADE-OFFS OF DIFFERENT SWiFI TECHNIQUES



1 Assembly-level FI

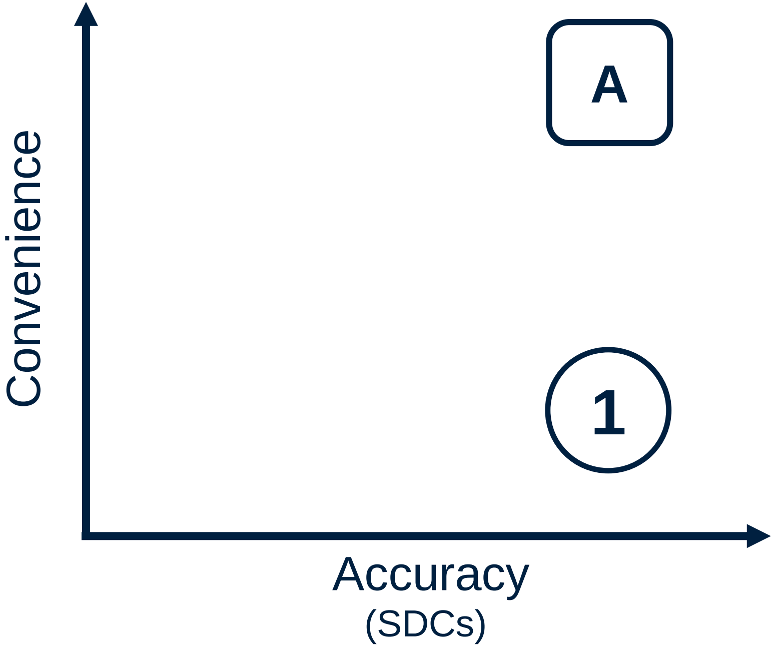
# TRADE-OFFS OF DIFFERENT SWiFI TECHNIQUES

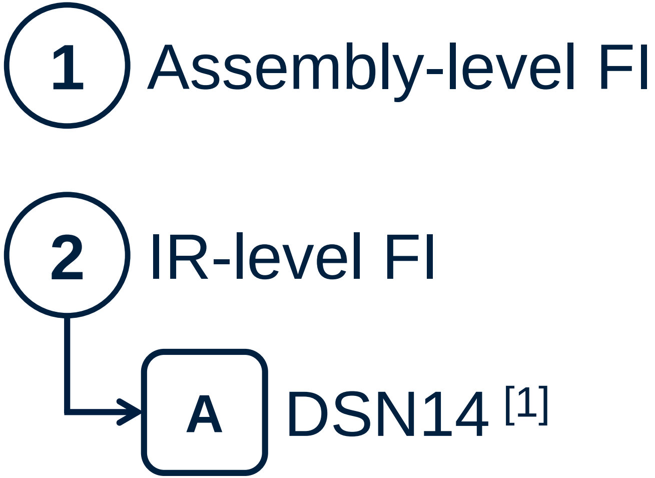


1 Assembly-level FI

2 IR-level FI

# TRADE-OFFS OF DIFFERENT SWiFI TECHNIQUES





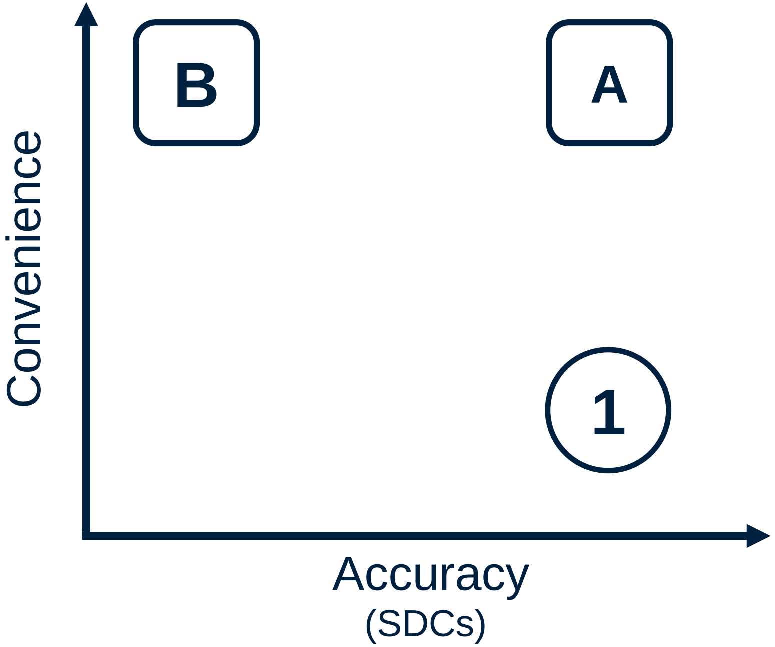
1 Assembly-level FI

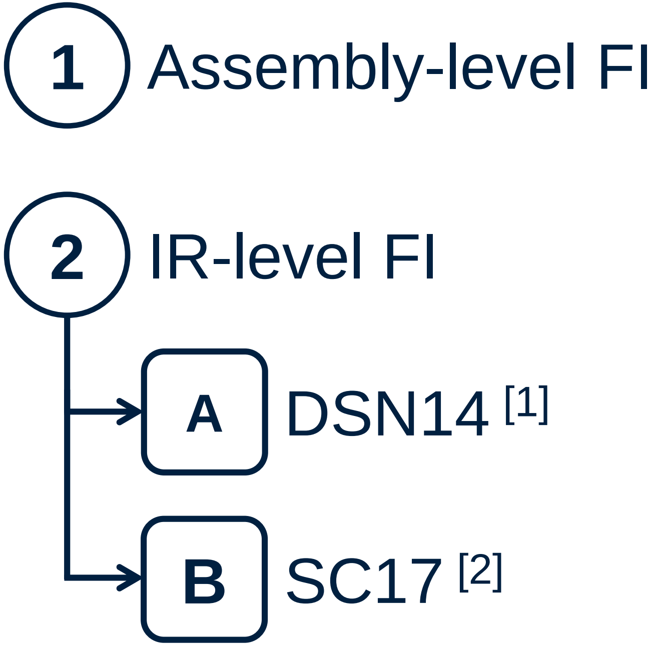
2 IR-level FI

A DSN14 [1]

[1] *Wei et al. DSN’14.*

# TRADE-OFFS OF DIFFERENT SWiFI TECHNIQUES





1 Assembly-level FI

2 IR-level FI

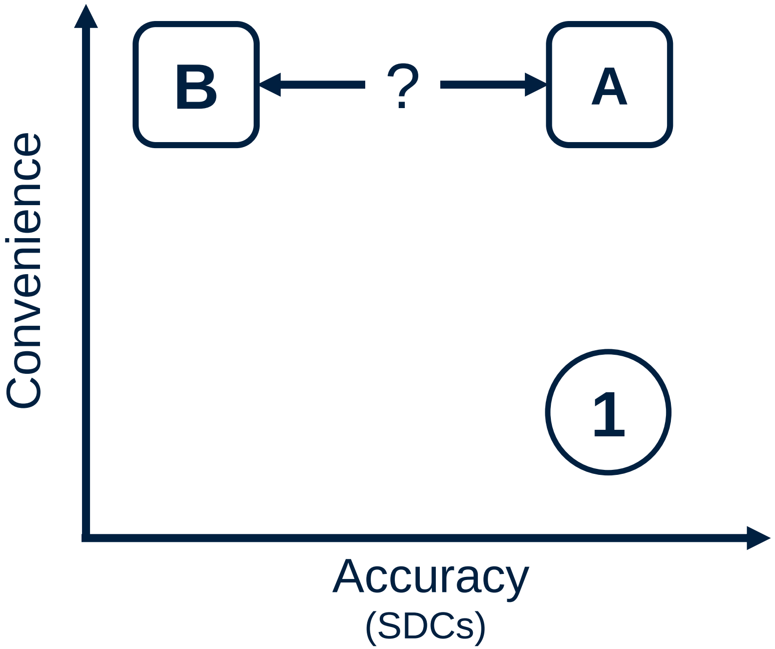
A DSN14 [1]

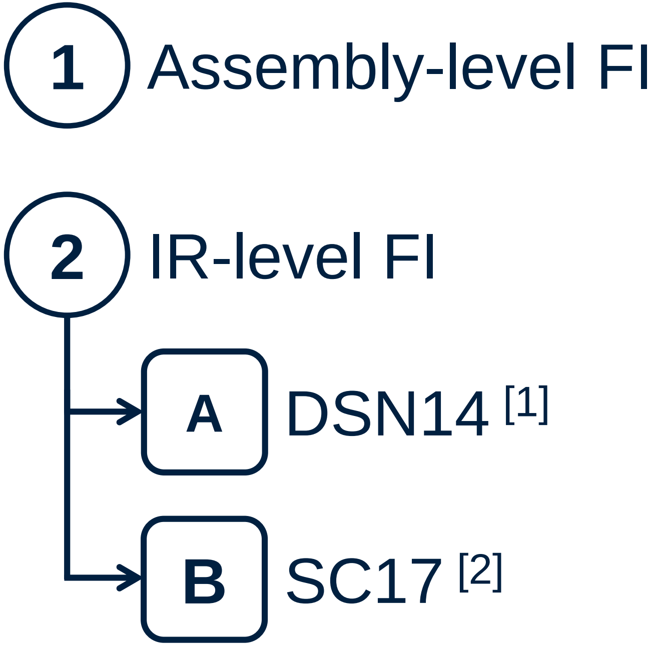
B SC17 [2]

[1] *Wei et al. DSN'14.*

[2] *Georgakoudis et al. SC'17.*

# TRADE-OFFS OF DIFFERENT SWiFI TECHNIQUES





[1] *Wei et al. DSN’14.*

[2] *Georgakoudis et al. SC’17.*

# PRIOR WORK: SUMMARY

[1] https://github.com/DependableSystemsLab/LLFI

[2] https://github.com/DependableSystemsLab/PINFI

# PRIOR WORK: SUMMARY

- Both studies use **LLFI**[1] (IR-level) and **PINFI**[2] (assembly-level)
    - SC17 uses a *modified* version of PINFI

[1] https://github.com/DependableSystemsLab/LLFI

[2] https://github.com/DependableSystemsLab/PINFI

# PRIOR WORK: SUMMARY

- Both studies use **LLFI**[1] (IR-level) and **PINFI**[2] (assembly-level)
    - SC17 uses a *modified* version of PINFI
- **DSN14** (*Wei et al.*)
    - LLFI is as accurate as PINFI for measuring SDC probabilities

[1] https://github.com/DependableSystemsLab/LLFI

[2] https://github.com/DependableSystemsLab/PINFI

# PRIOR WORK: SUMMARY

- Both studies use **LLFI**[1] (IR-level) and **PINFI**[2] (assembly-level)
    - SC17 uses a *modified* version of PINFI
- **DSN14** (*Wei et al.*)
    - LLFI is as accurate as PINFI for measuring SDC probabilities
- **SC17** (*Georgakoudis et al.*)
    - LLFI is ***not*** as accurate as PINFI, even for SDCs
    - Attributed differences to limitations of LLFI (e.g., back-end optimizations)

[1] https://github.com/DependableSystemsLab/LLFI

[2] https://github.com/DependableSystemsLab/PINFI

# RESEARCH QUESTIONS

# RESEARCH QUESTIONS

**1. Why does prior work come to contradictory findings?**

# RESEARCH QUESTIONS

1. Why does prior work come to contradictory findings?

2. What is the accuracy of IR-level FI compared to assembly-level FI?

    2.1 SDCs

    2.2 Crashes

# RESEARCH QUESTIONS

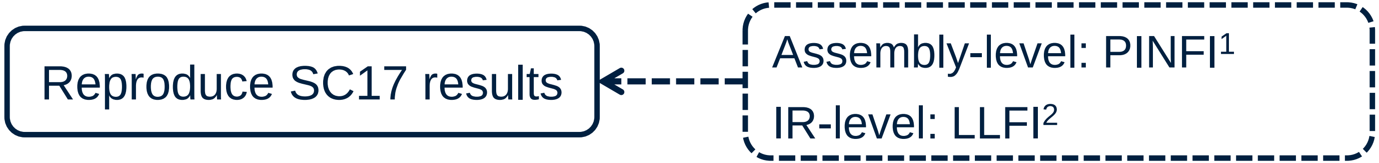
**1. Why does prior work come to contradictory findings?**

**2. What is the accuracy of IR-level FI compared to assembly-level FI?**

- **2.1 SDCs**
- **2.2 Crashes**

# PRIOR WORK ANALYSIS: DSN14 VS. SC17

# PRIOR WORK ANALYSIS: DSN14 VS. SC17



Reproduce SC17 results

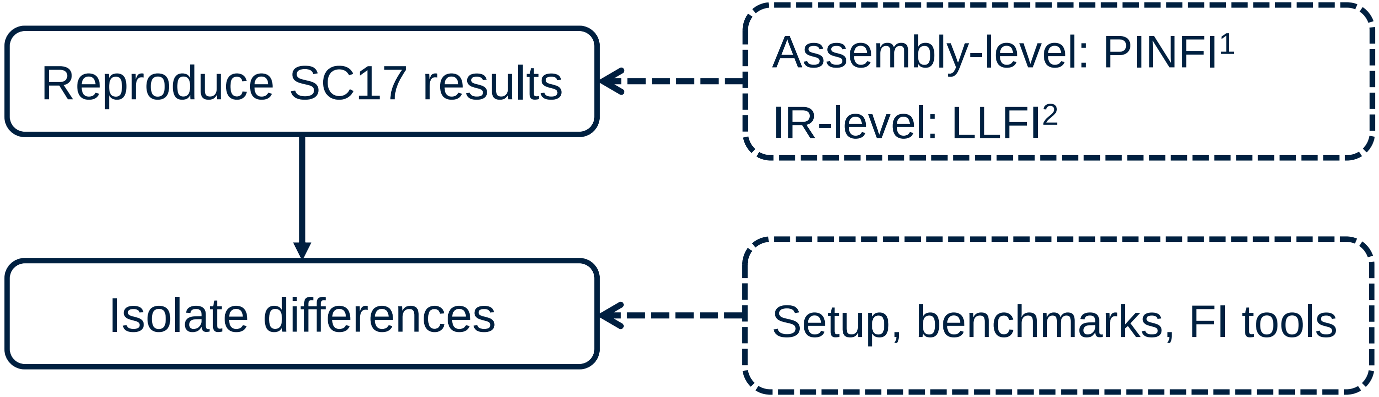
Assembly-level: PINFI[1]

IR-level: LLFI[2]

[1] https://github.com/DependableSystemsLab/PINFI

[2] https://github.com/DependableSystemsLab/LLFI

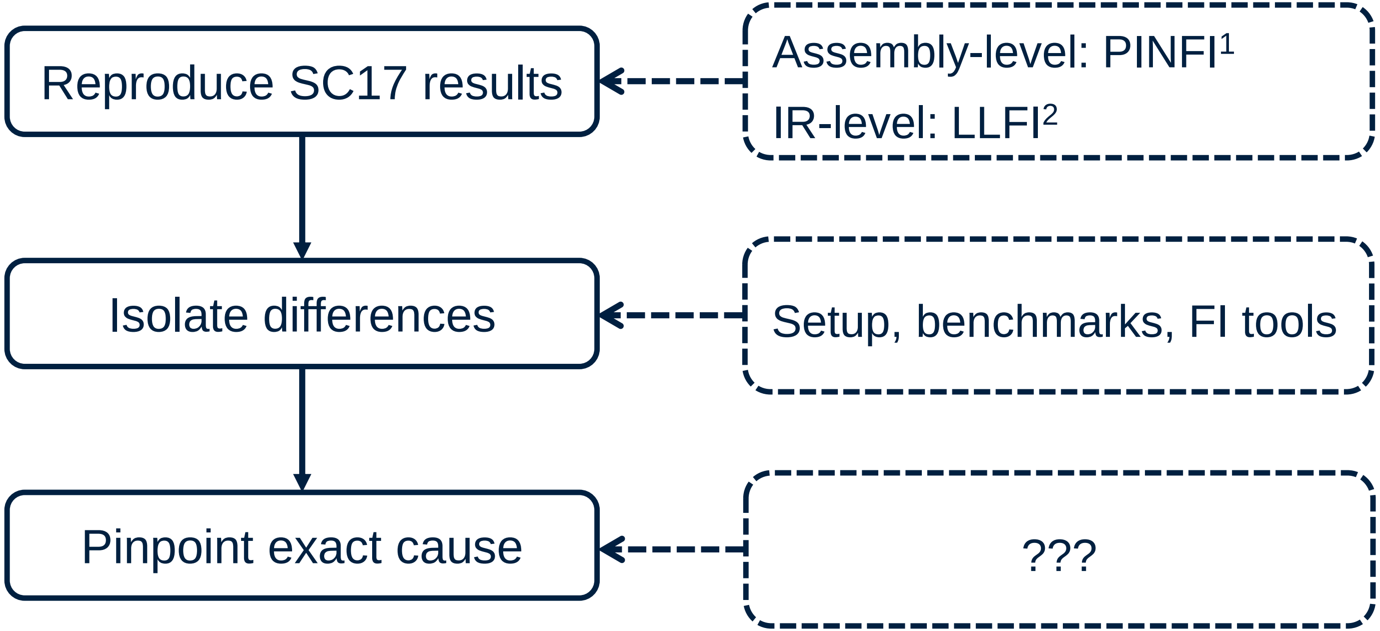
# PRIOR WORK ANALYSIS: DSN14 VS. SC17



[1] https://github.com/DependableSystemsLab/PINFI

[2] https://github.com/DependableSystemsLab/LLFI

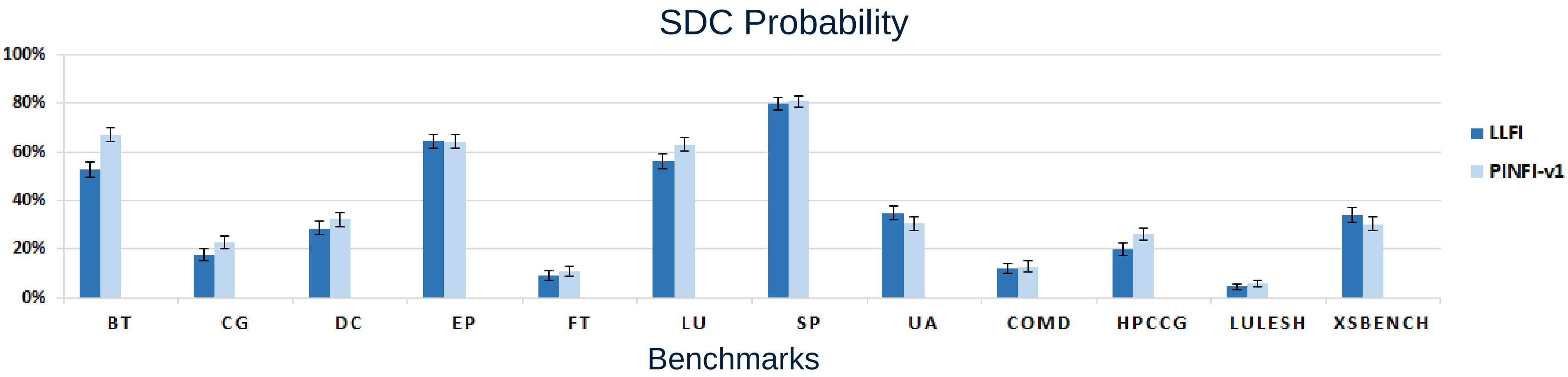
# PRIOR WORK ANALYSIS: DSN14 VS. SC17



[1] https://github.com/DependableSystemsLab/PINFI

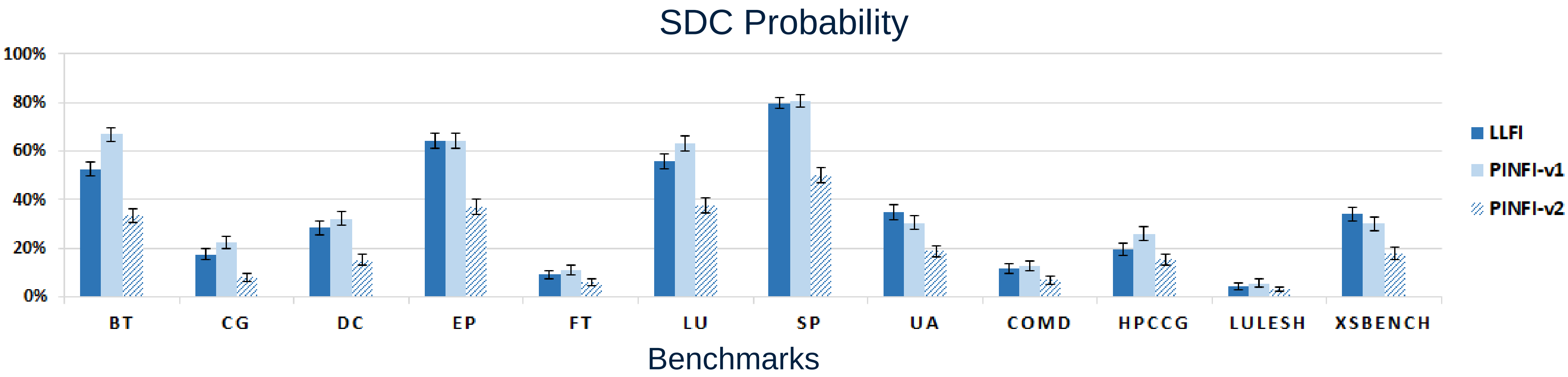
[2] https://github.com/DependableSystemsLab/LLFI

# PRIOR WORK ANALYSIS: DSN14 VS. SC17



| | |
|---|---|
| **LLFI** | Official version used by both DSN14 and SC17 |
| **PINFI** | Official version hosted on GitHub |

# PRIOR WORK ANALYSIS: DSN14 VS. SC17



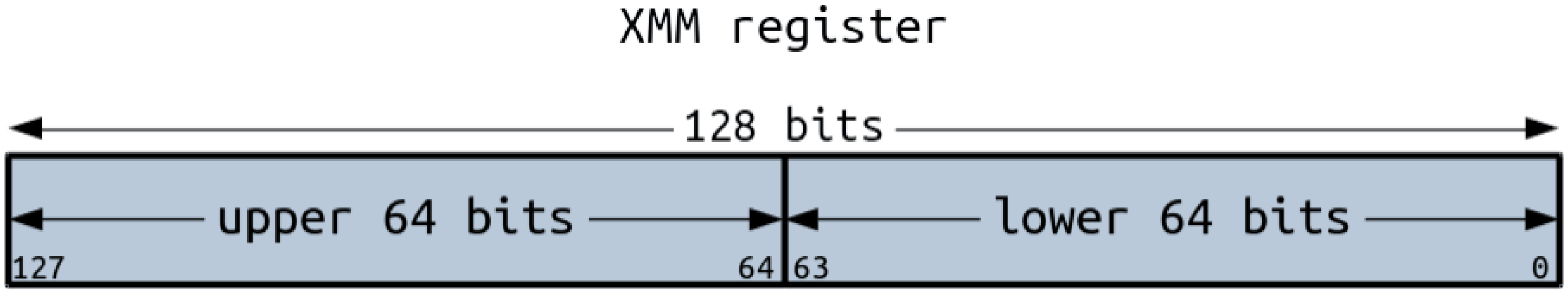
**LLFI** Official version used by both DSN14 and SC17

**PINFI-v1** Official version hosted on GitHub (same as DSN14)

**PINFI-v2** ***Modified*** version used in SC17 (publicly available)

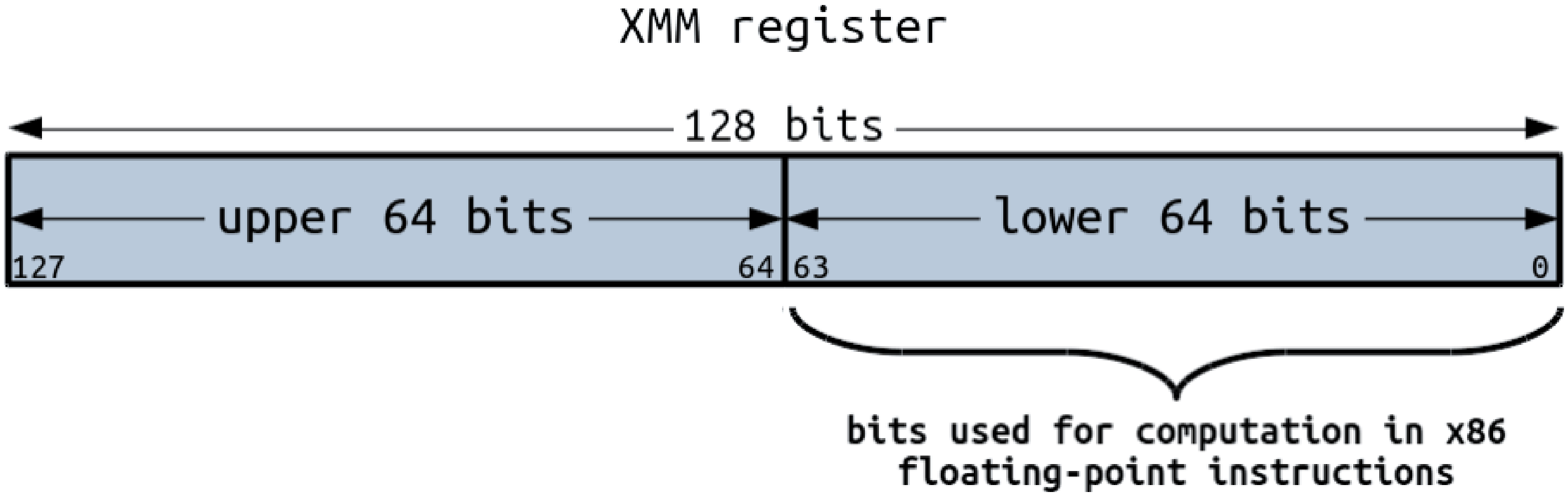
# BIT-SAMPLING METHODOLOGY

e.g., x86 double-precision floating-point instructions (**`addsd`**, **`mulsd`**, etc.)



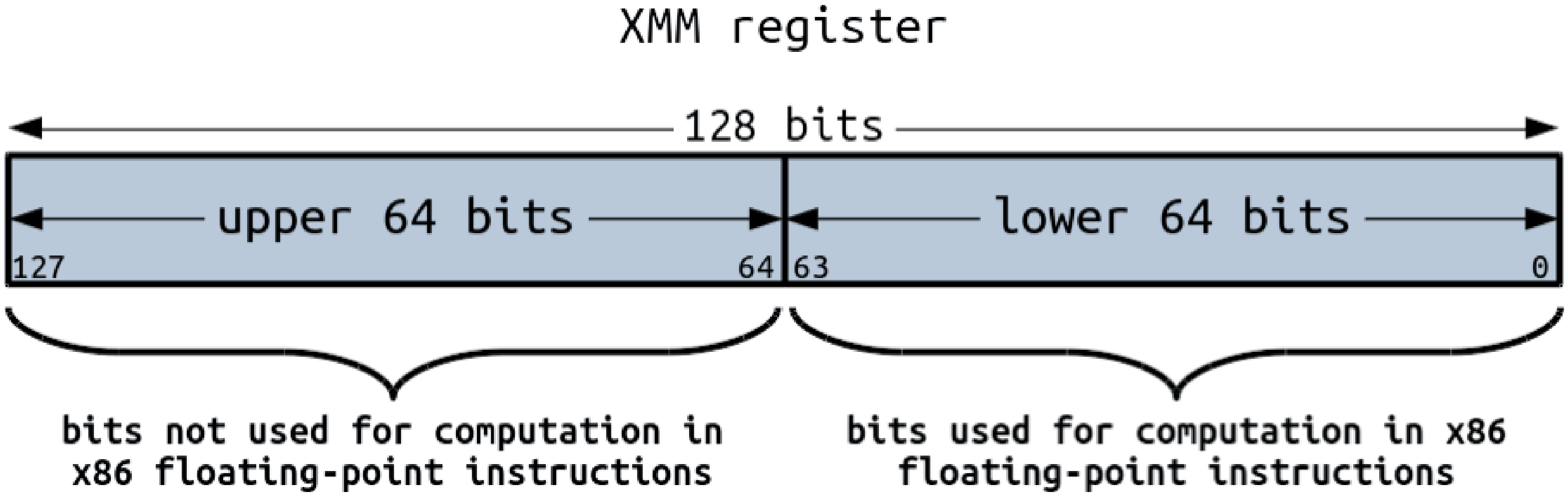
# BIT-SAMPLING METHODOLOGY

e.g., x86 double-precision floating-point instructions (**`addsd`**, **`mulsd`**, etc.)


XMM register
128 bits
upper 64 bits
lower 64 bits
127
64
63
0
bits used for computation in x86 floating-point instructions

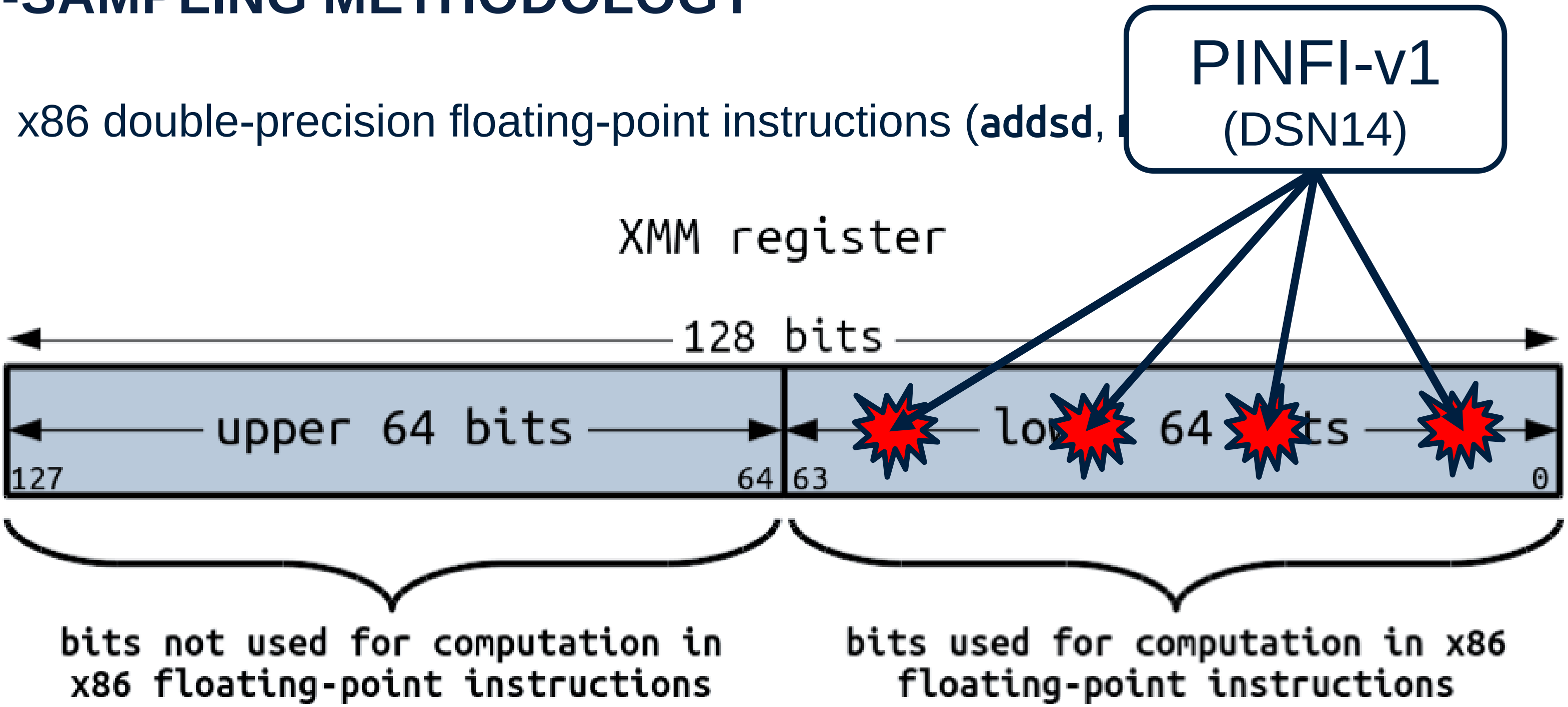
# BIT-SAMPLING METHODOLOGY

e.g., x86 double-precision floating-point instructions (**`addsd`**, **`mulsd`**, etc.)


XMM register
128 bits
upper 64 bits
lower 64 bits
127
64
63
0
bits not used for computation in x86 floating-point instructions
bits used for computation in x86 floating-point instructions

# BIT-SAMPLING METHODOLOGY

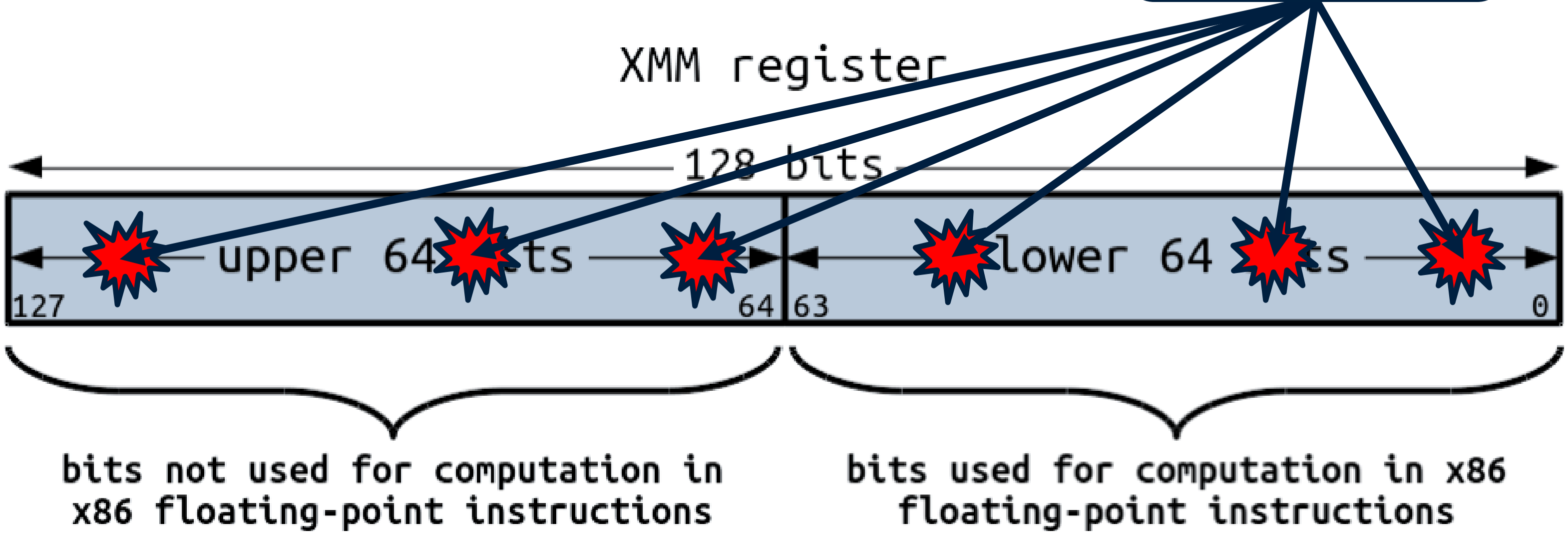
e.g., x86 double-precision floating-point instructions (**addsd**, **


PINFI-v1
(DSN14)
XMM register
128 bits
upper 64 bits
lower 64 bits
127
64
63
0
bits not used for computation in x86 floating-point instructions
bits used for computation in x86 floating-point instructions

# BIT-SAMPLING METHODOLOGY

e.g., x86 double-precision floating-point instructions (**addsd**,

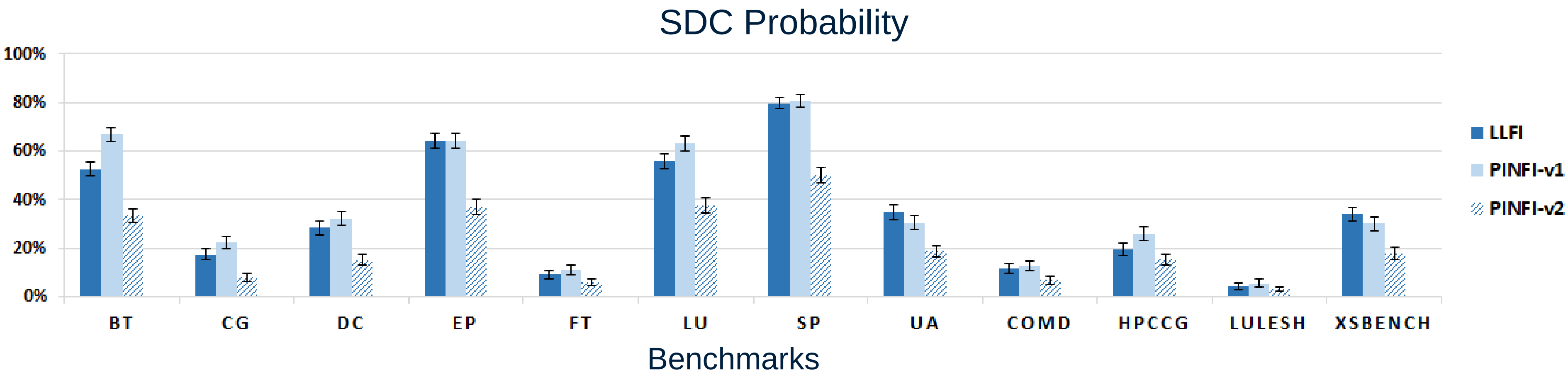
PINFI-v2
(SC17)



bits not used for computation in x86 floating-point instructions

bits used for computation in x86 floating-point instructions

# PRIOR WORK ANALYSIS: DSN14 VS. SC17

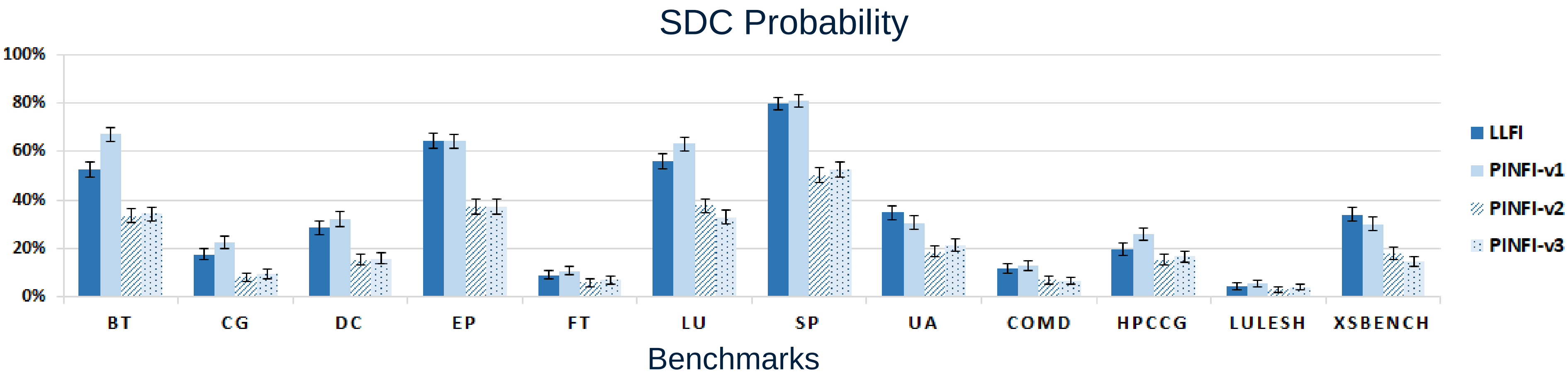


**LLFI** Official version used by both DSN14 and SC17

**PINFI-v1** Official version hosted on GitHub (same as DSN14)

**PINFI-v2** Version used in SC17 (publicly available)

# PRIOR WORK ANALYSIS: DSN14 VS. SC17



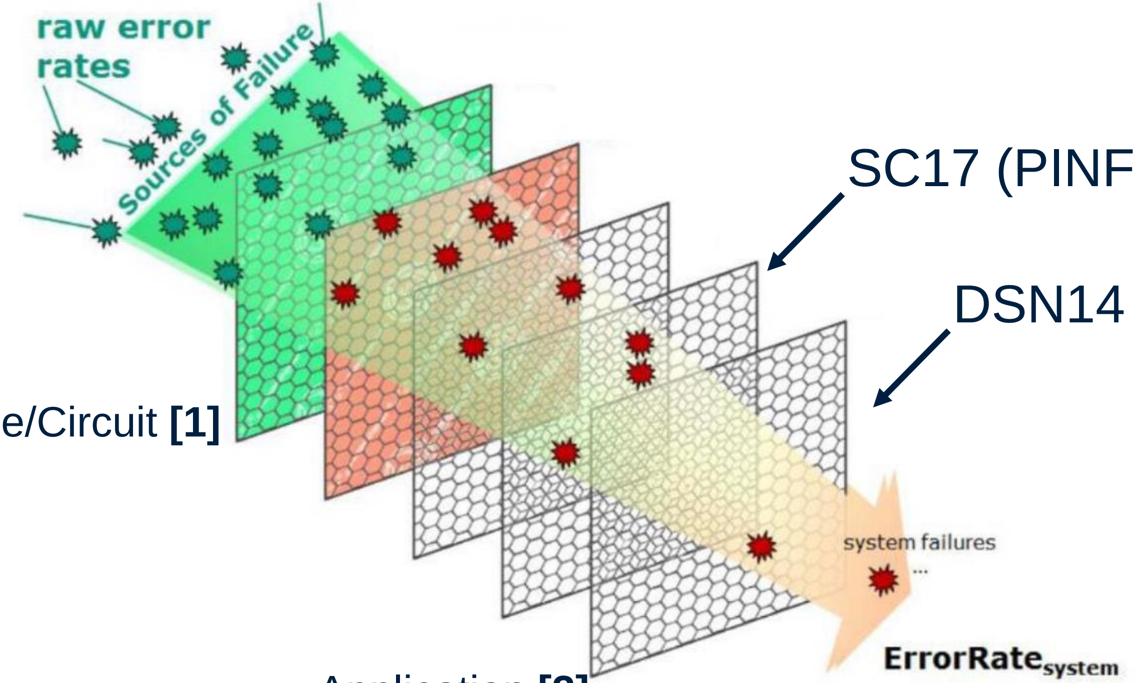
| | |
|---|---|
| **LLFI** | Official version used by both DSN14 and SC17 |
| **PINFI-v1** | Official version hosted on GitHub (same as DSN14) |
| **PINFI-v2** | Version used in SC17 (publicly available) |
| **PINFI-v3** | PINFI-v1, modified to match bit-sampling methodology of PINFI-v2 |

# WHY DOES THIS MATTER?

- Affects results significantly
- Depends on desired fault model

***Important to stay consistent in comparison studies!***

# “*fault sensitivity*” **[1]** vs “*error sensitivity*” **[2]**



SC17 (PINFI-v2)

DSN14 (PINFI-v1, LLFI)

Device/Circuit **[1]**

Application **[2]**

Photo source: *https://pdfs.semanticscholar.org/c052/8c02f566d211f9bd90b7c1d3703256fad053.pdf*

# RESEARCH QUESTIONS

**1. Why does prior work come to contradictory findings?**

An invalid comparison in SC17 due to an inconsistent bit-sampling model

**2. What is the accuracy of IR-level FI compared to assembly-level FI?**

**2.1 SDCs:**

**2.2 Crashes:**

# RESEARCH QUESTIONS

**1. Why does prior work come to contradictory findings?**

An invalid comparison in SC17 due to an inconsistent bit-sampling model

**2. What is the accuracy of IR-level FI compared to assembly-level FI?**

**2.1 SDCs:**

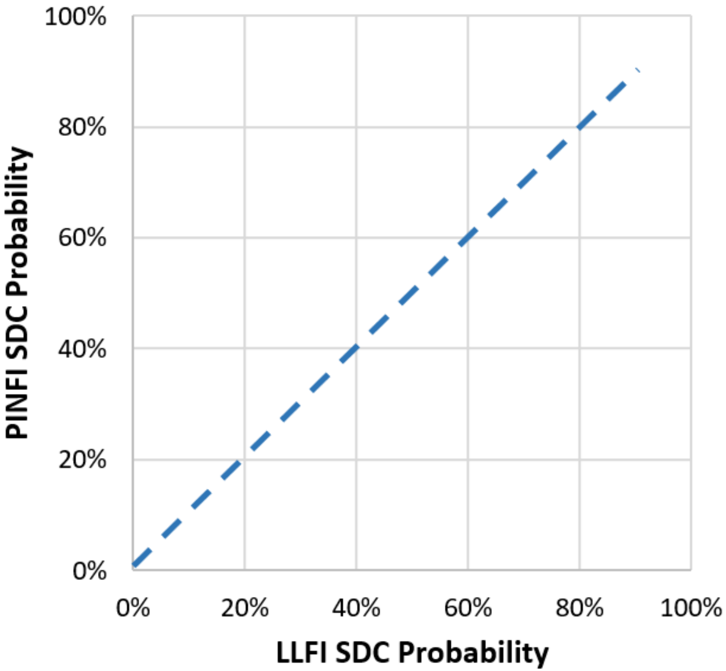
**2.2 Crashes:**

# END-TO-END EVALUATION

- **Extensive FI comparison study** (LLFI vs. PINFI)
- **25 benchmarks** (incl. most from DSN14 and SC17)
- **4 LLVM optimization levels** (*-O0*, *-O1*, *-O2*, *-O3*)
- **Three statistical tests** (linear reg., t-test, Spearman’s rank)

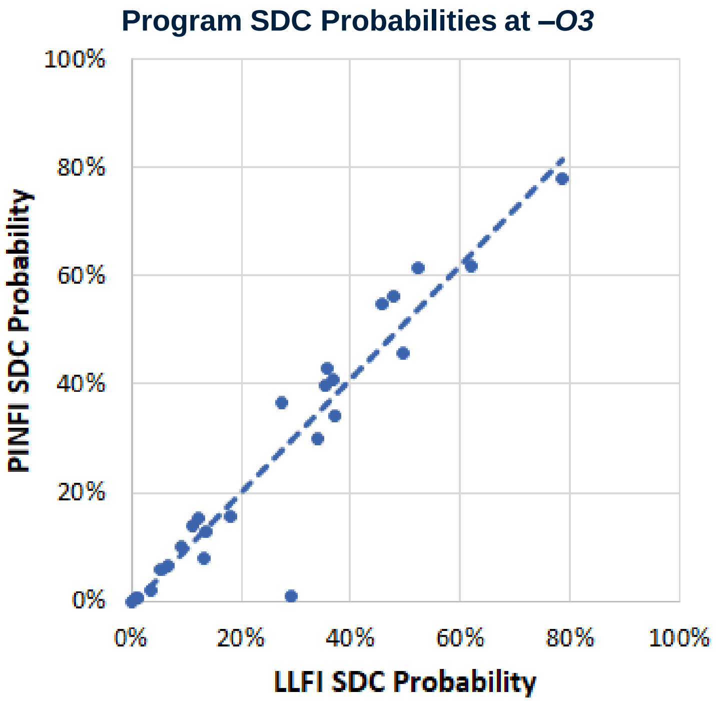
***Are IR-level SDC/crash probability measurements accurate?***

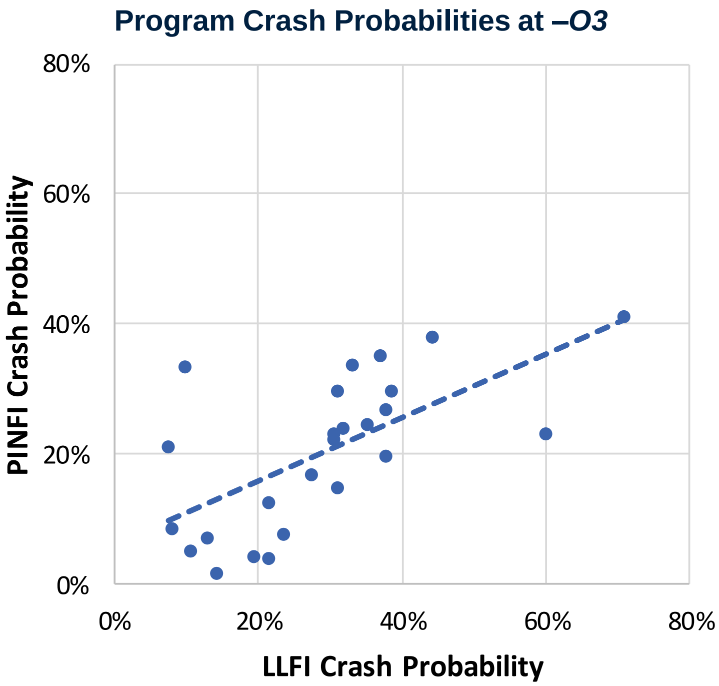
# LINEAR REGRESSION ANALYSIS



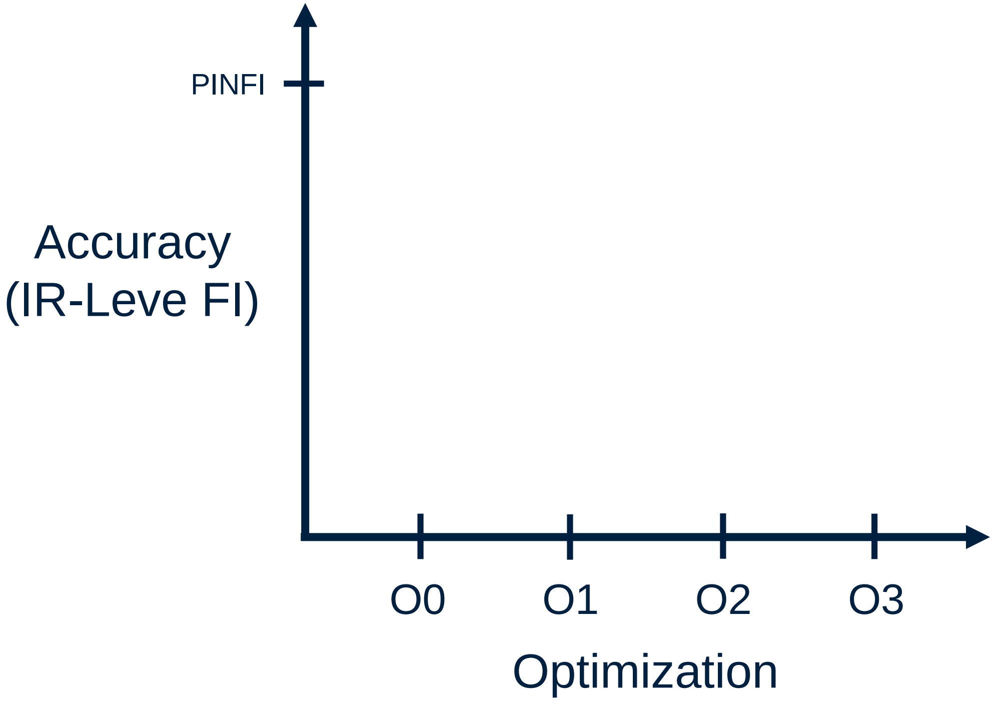
**Ideal case:**
**Linear equation $y = x$**

# LINEAR REGRESSION ANALYSIS

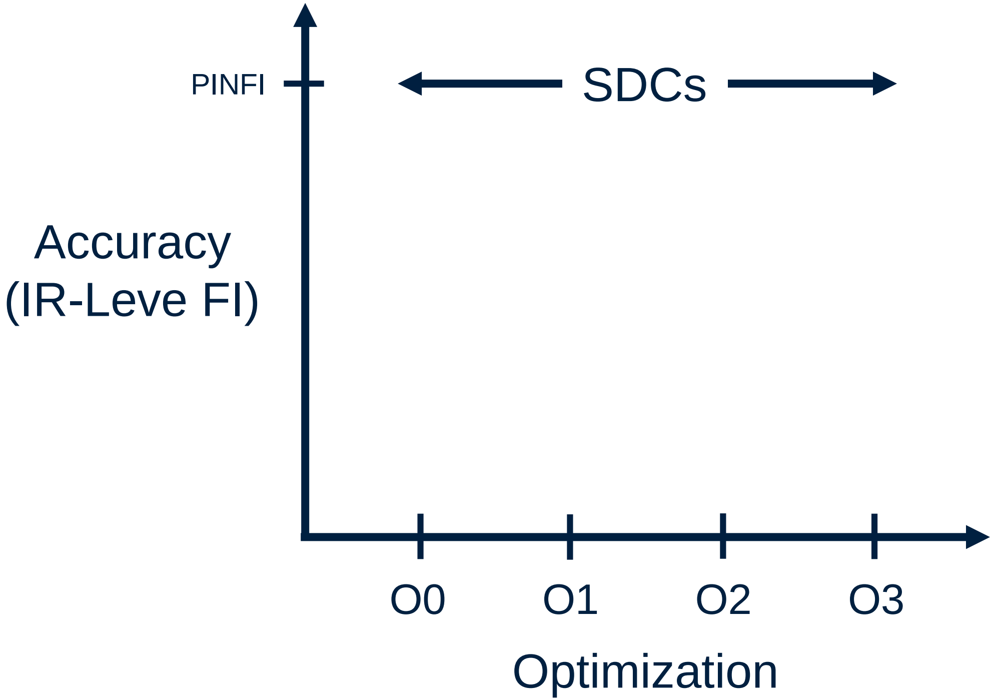




# OVERALL FINDINGS

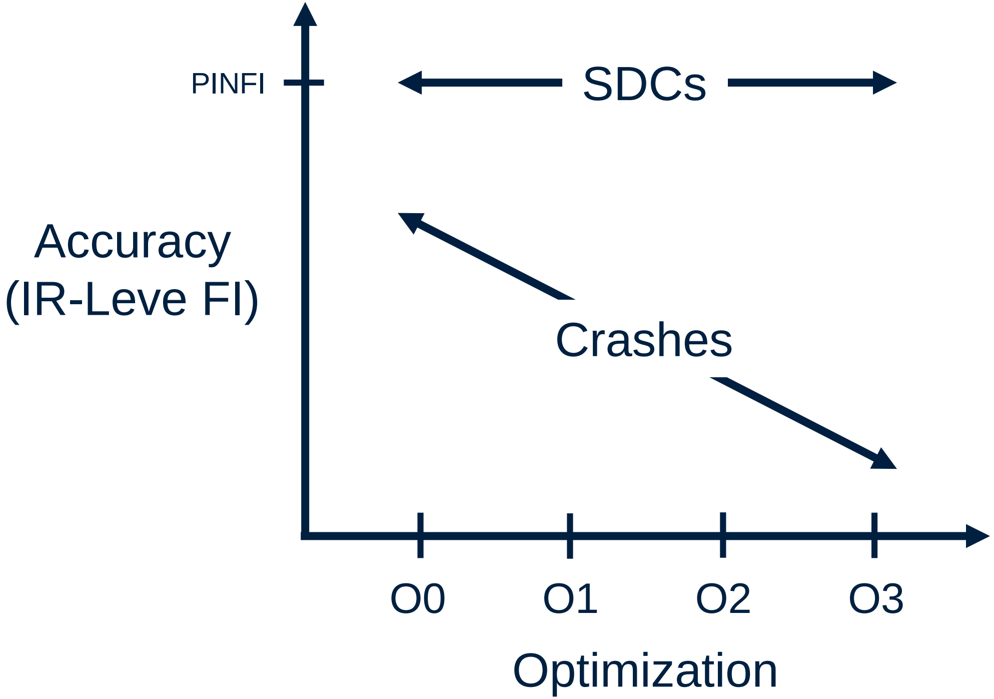


# OVERALL FINDINGS



**Findings are consistent with DSN14 results**

# OVERALL FINDINGS



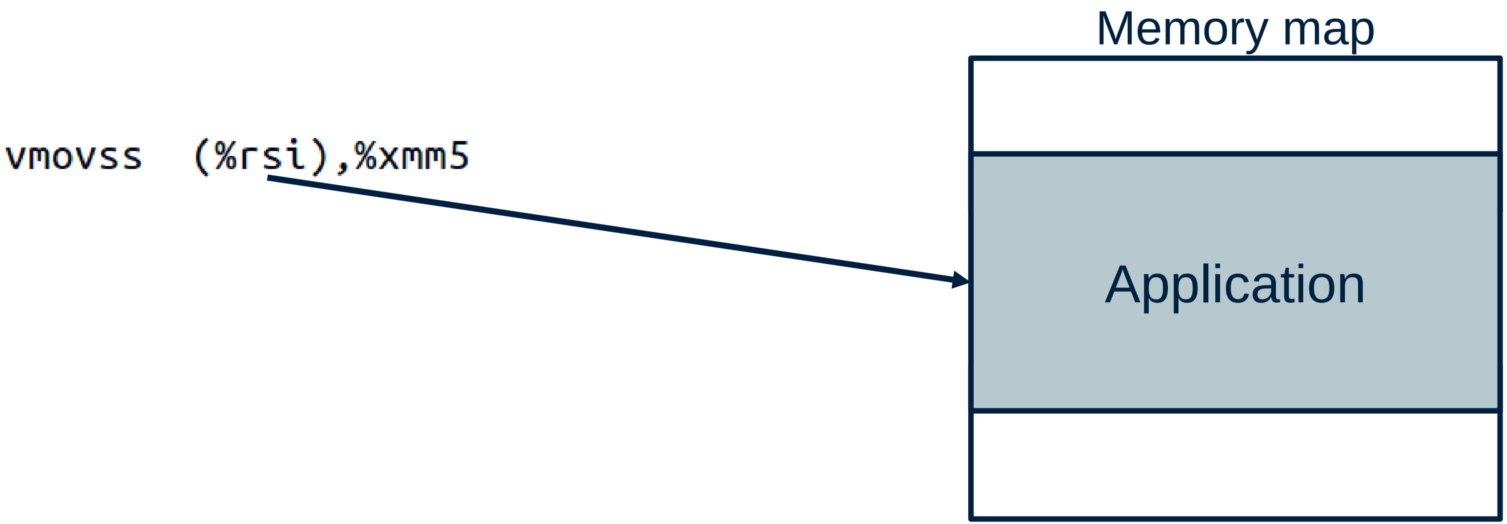
**Findings are consistent with DSN14 results**

# WHAT ABOUT CRASHES?

- Back-end optimizations
- Memory operations (e.g., register allocation)
- Predominant source of crashes: **segmentation faults** [Fang et al., DSN16]

# WHAT ABOUT CRASHES?

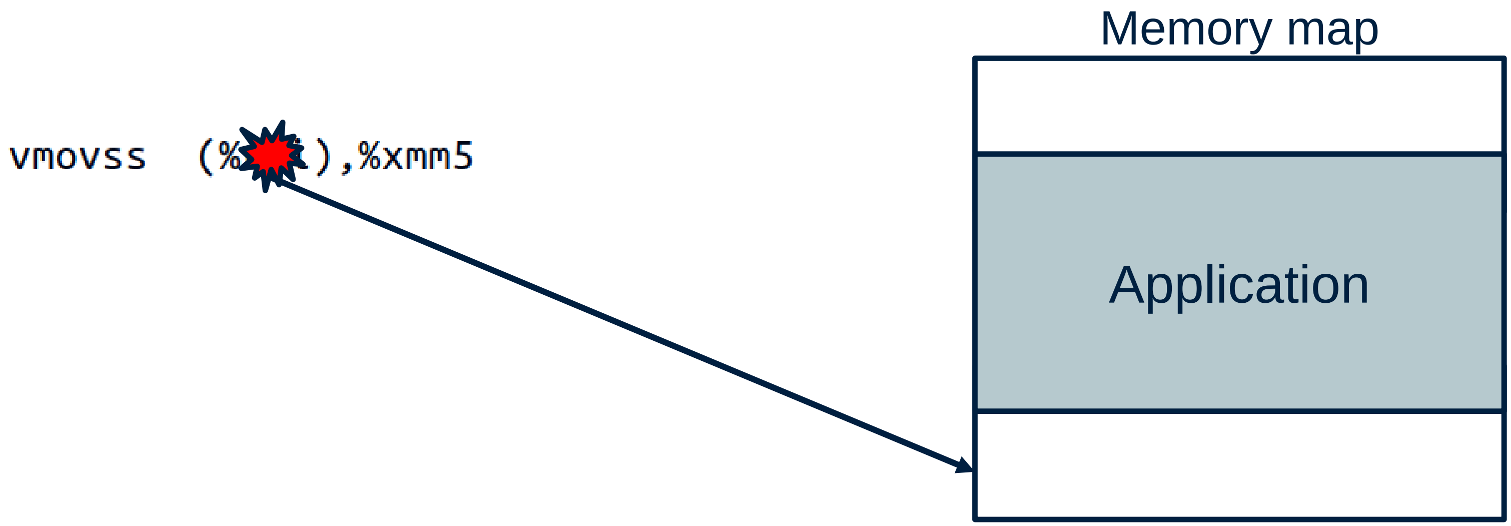
- Back-end optimizations
- Memory operations (e.g., register allocation)
- Predominant source of crashes: **segmentation faults** [Fang et al., DSN16]



```
vmovss  (%rsi),%xmm5
```

# WHAT ABOUT CRASHES?

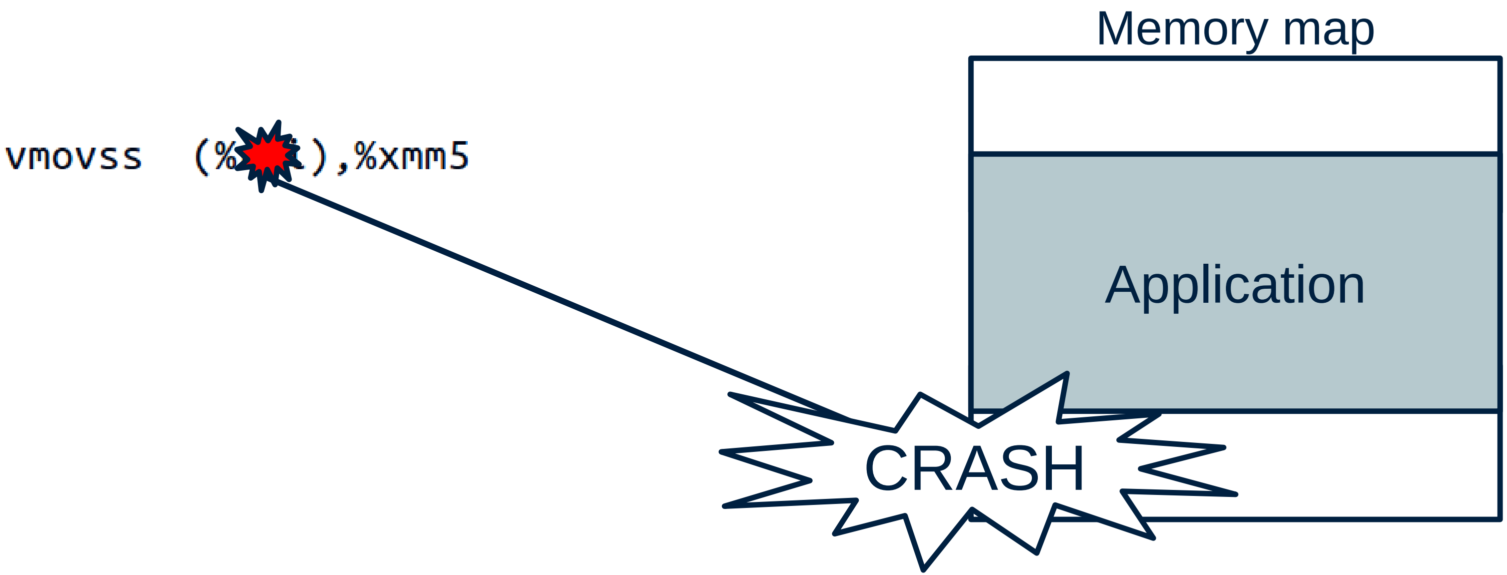
- Back-end optimizations
- Memory operations (e.g., register allocation)
- Predominant source of crashes: **segmentation faults** [Fang et al., DSN16]



```
vmovss   (%[illegible]i),%xmm5
```

# WHAT ABOUT CRASHES?

- Back-end optimizations
- Memory operations (e.g., register allocation)
- Predominant source of crashes: **segmentation faults** [Fang et al., DSN16]



```
vmovss   (%[illegible]i),%xmm5
```

# RESEARCH QUESTIONS

**1. Why does prior work come to contradictory findings?**

An invalid comparison in SC17 due to an inconsistent bit-sampling model

**2. What is the accuracy of IR-level FI compared to assembly-level FI?**

- **2.1 SDCs:** IR-level FI is accurate across all optimization levels
- **2.2 Crashes:** IR-level FI is *not* accurate; accuracy gets worse with optimizations

# RESEARCH QUESTIONS

**1. Why does prior work come to contradictory findings?**

An invalid comparison in SC17 due to an inconsistent bit-sampling model

**2. What is the accuracy of IR-level FI compared to assembly-level FI?**

**2.1 SDCs:** IR-level FI is accurate across all optimization levels

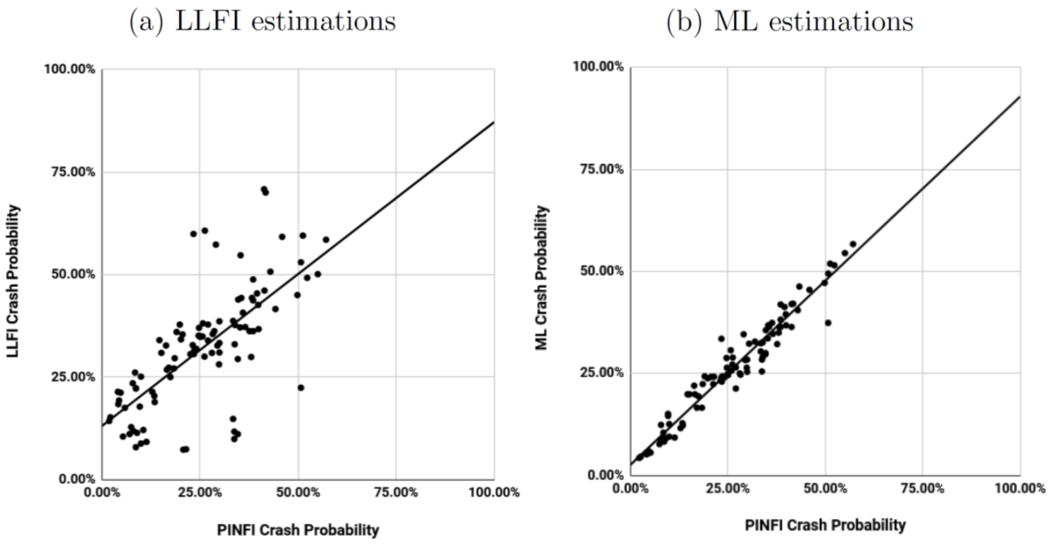
**2.2 Crashes:** IR-level FI is *not* accurate; accuracy gets worse with optimizations

# FUTURE WORK

- Extending comparison to other platforms (e.g., ARM)
- Evaluate accuracy across individual optimizations
- Improve accuracy of IR-level FI for crashes

# FUTURE WORK

- Extending comparison to other platforms (e.g., ARM)
- Evaluate accuracy across individual optimizations
- Improve accuracy of IR-level FI for crashes


(a) LLFI estimations

(b) ML estimations

# SUMMARY OF CONTRIBUTIONS

- Settled confusion on accuracy of IR-level FI (DSN14 vs SC17)
- Highlight the importance of clearly defining FI parameters
- Re-establish confidence in IR-level FI for SDCs
- Quantify accuracy (SDCs/crashes) across optimization levels

# SUMMARY OF CONTRIBUTIONS

- Settled confusion on accuracy of IR-level FI (DSN14 vs SC17)
- Highlight the importance of clearly defining FI parameters
- Re-establish confidence in IR-level FI for SDCs
- Quantify accuracy (SDCs/crashes) across optimization levels

***Thank you!***

lpalazzi@ece.ubc.ca

*Data and tools are publicly available: https://github.com/DependableSystemsLab/ISSRE19/*

THE UNIVERSITY OF BRITISH COLUMBIA