

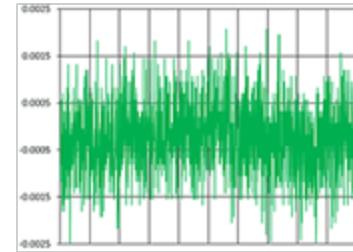
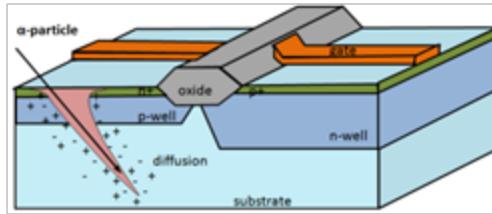
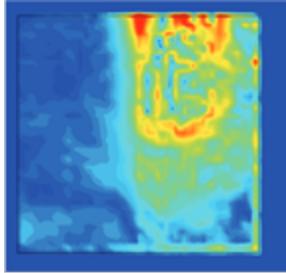
# **LLFI: An Intermediate Code-Level Fault Injection Tool for Hardware Faults**



Qining Lu, Mostafa Farahani,  
Jiesheng Wei, Anna Thomas, and  
**Karthik Pattabiraman**

Department of Electrical and Computer Engineering,  
University of British Columbia

# Motivation



- Hardware faults' frequency is increasing [Borkar'11][Constantinescu'08]
  - Temperature variations
  - Soft errors due to particle strikes
  - Manufacturing defects

# Hardware Errors: Traditional “Solutions”

- **Guard-banding**

Guard-banding wastes power as gap between average and worst-case widens due to variations

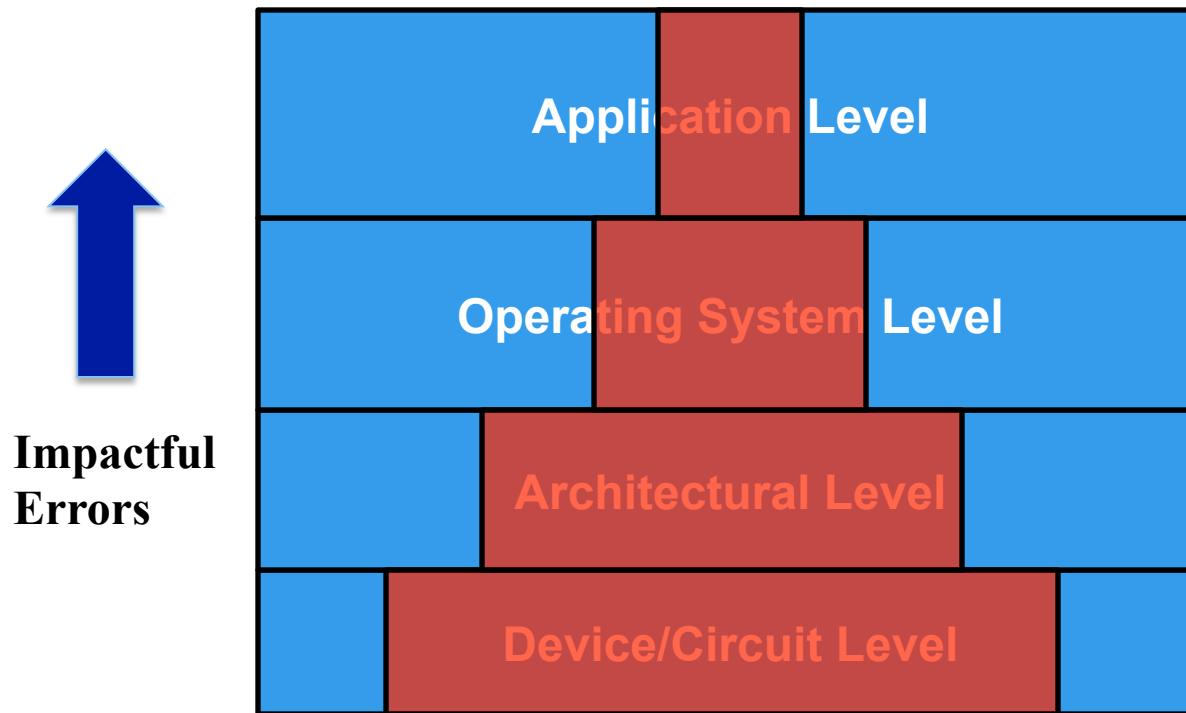


- **Duplication**

Hardware duplication (DMR) can result in 2X slowdown and/or energy consumption

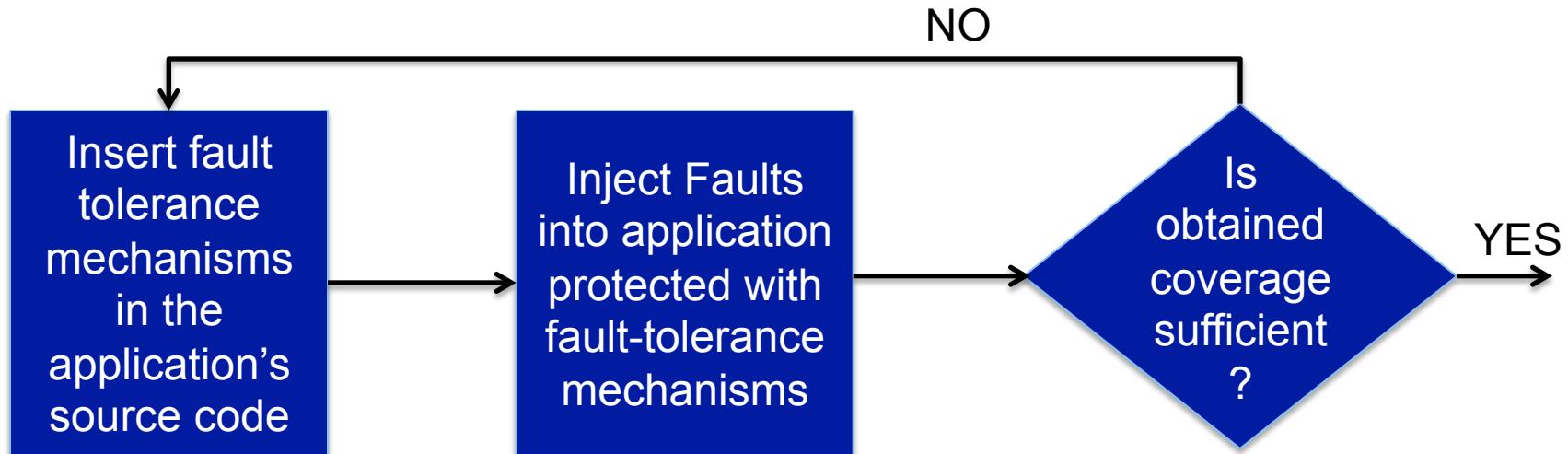


# Application-level techniques

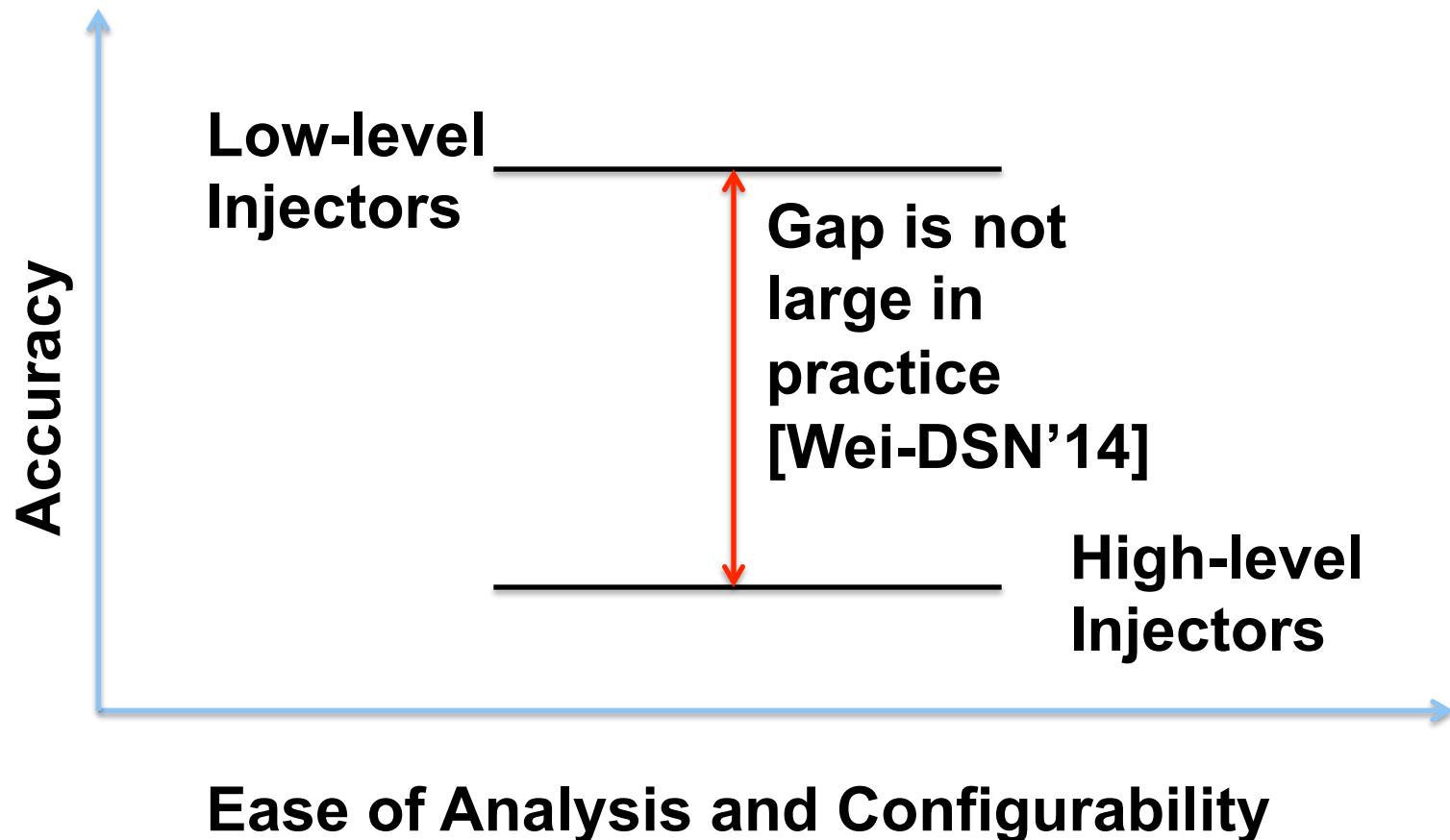


# Fault Injection

- To obtain coverage estimates for applications
- Iteratively improve coverage based on the errors missed by fault tolerance mechanisms
  - Analyze the errors that are missed by the FTMs



# High-Level Vs. Low-Level Injectors



# Outline

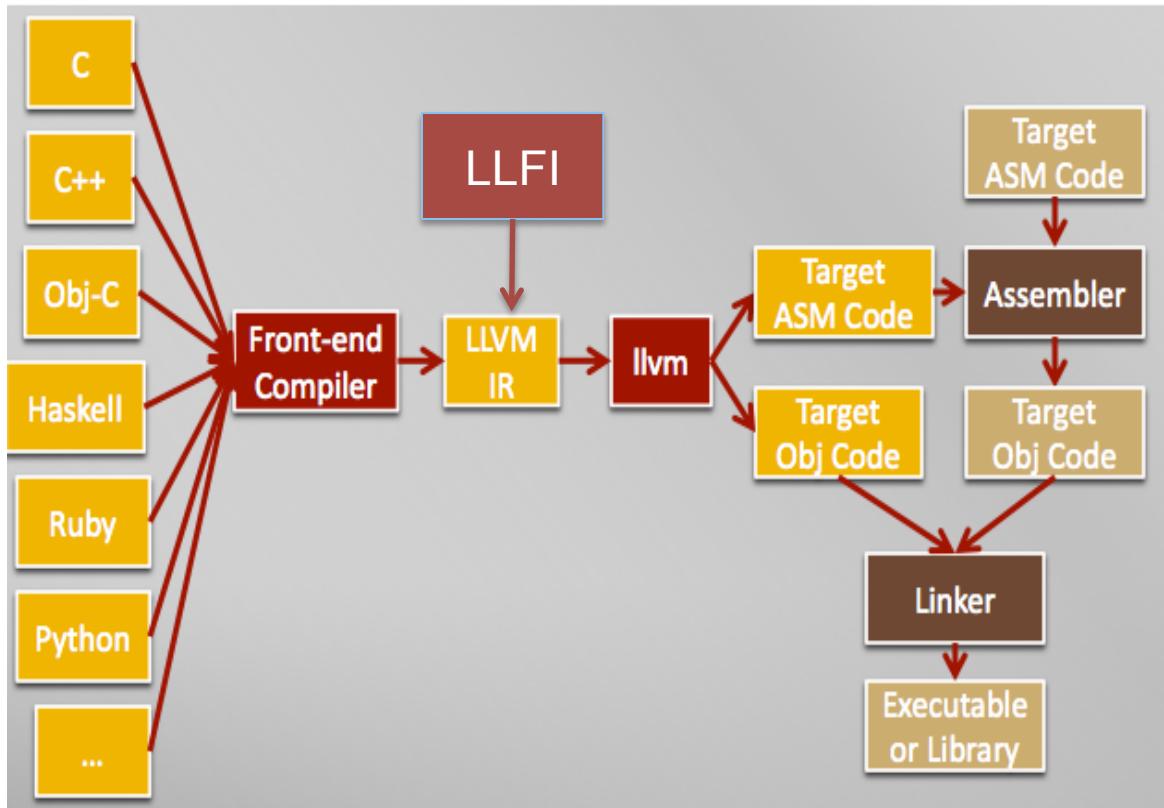
- Motivation
- LLFI: High Level Fault Injector
- Experimental Setup and Results
- Conclusion

# LLFI: High-Level Fault Injector

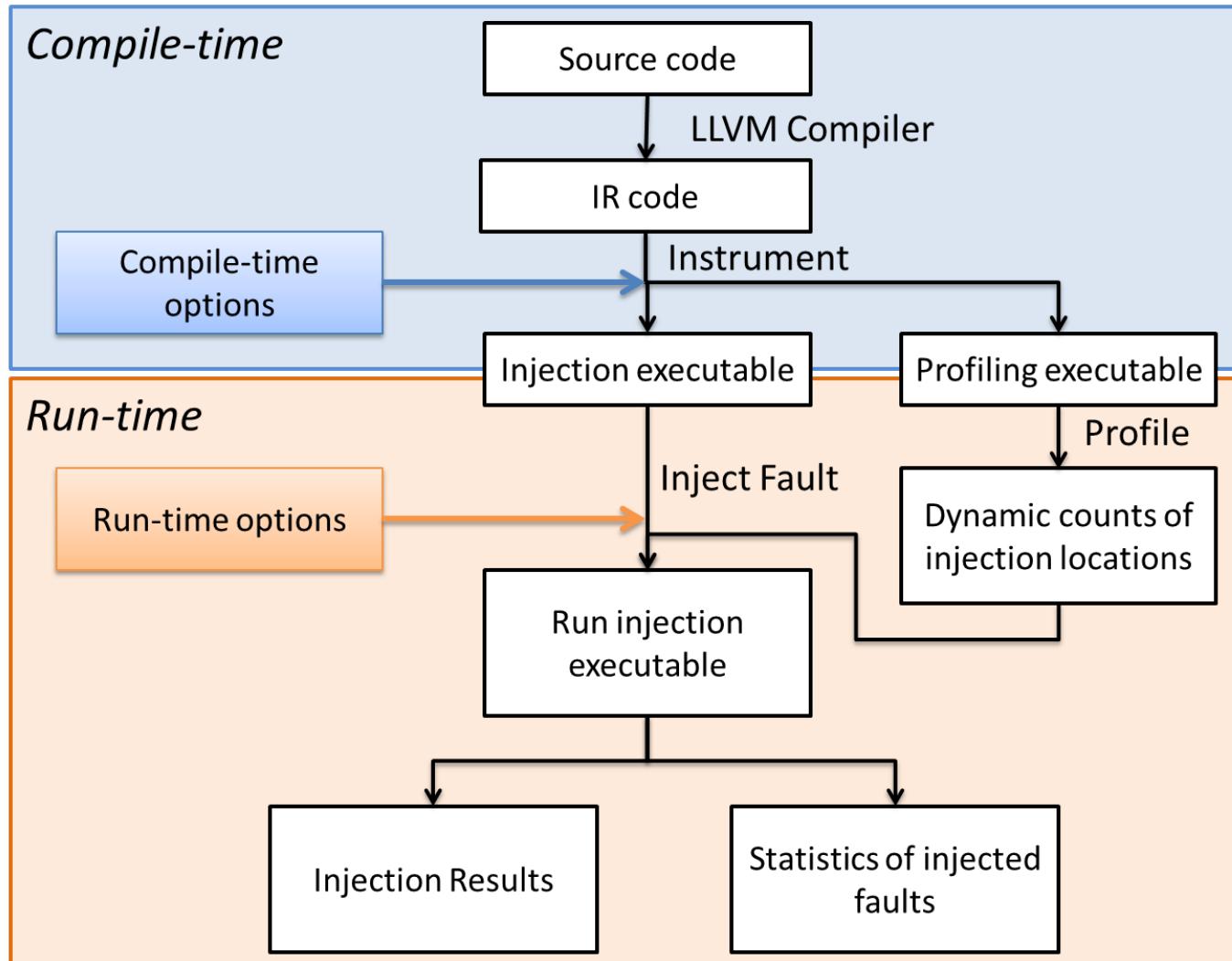
- **A fault injector based on LLVM**
  - Intermediate representation (IR) level injection
- **Features**
  - Easy to customize the fault injection
  - Easy to analyze fault propagation
  - Easy to use by developers

# LLFI: Structure

Works at LLVM compiler's intermediate (IR) code level [Lattner'05] –widely used in industry



# LLFI: Workflow

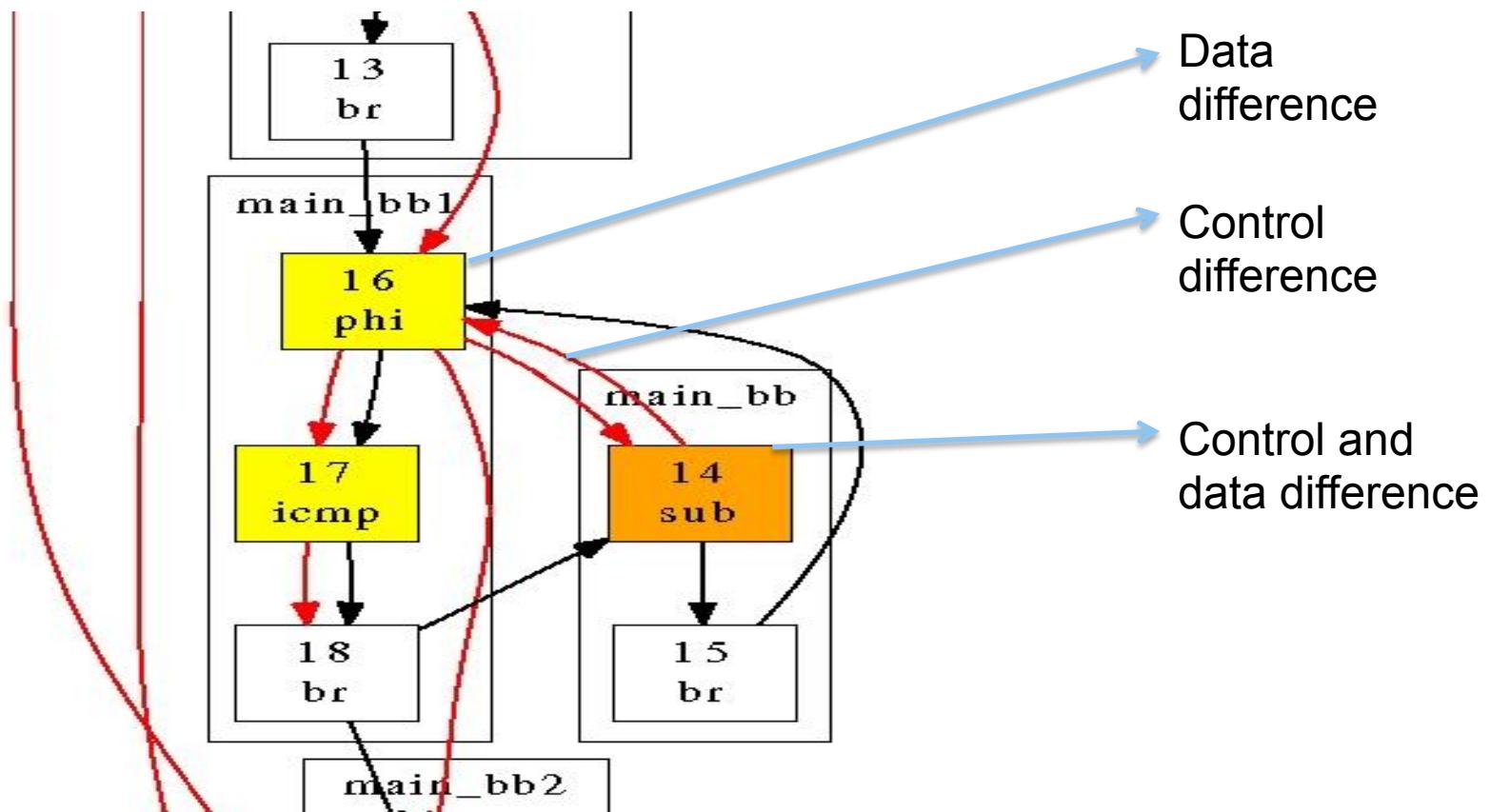


# LLFI: Easy to Customize

- **Fault injection instruction selector**
  - Based on instruction type (LLVM IR)
    - Include: add + cmp
    - Exclude: load
  - Based on register target (source/destination)
  - Can choose type of fault to inject (single/ double bit flip, stuck at fault etc.)

# LLFI: Easy to Analyze

- Graphical output of trace differences



# LLFI: Easy to Use

The screenshot shows the LLFI application window. At the top, there's a menu bar with Standard, Advanced, File, Edit, Help, and several tabs: Compile To IR, Instrument, Profiling, Runtime Options, Inject Fault, Trace Graph, and Show graph output (unchecked). A 'List Of Files' sidebar on the left lists factorial.c and factorial.ll. The main area has tabs for Index and FileContent, with the Index tab active. The code editor displays LLVM assembly-like code:

```
; Function Attrs: nounwind readnone
declare void @llvm.dbg.declare(metadata, metadata) #1

declare i32 @atoi(...) #2

declare i32 @printf(i8*, ...) #3

attributes #0 = { nounwind ssp uwtable "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-p
attributes #1 = { nounwind readnone }
attributes #2 = { "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf"
!llvm.debug.cu = !{!0}
!llvm.module.flags = !{!12, !13}
!llvm.ident = !{!14}
```

To the right of the code editor is a 'Tutorial' section with the title 'Tutorial for Running LLFI :'. It contains four numbered steps:

1. Please upload a valid C or C++ file by choosing File ->Open File ->Select the file -> Open  
Note : The program could not be edited here.
2. Click the "Compile To IR" button to compile your program to IR form.
3. Click the "Instrument" button and select the Compile options to configure LLFI.
4. Please enter the command line input if there exists any and then Click the "Profiling" button.

Below the code editor is a table titled 'Profiling Status' with columns: Run Option, Fault Injection Type, Index, Line, Cycle, Bit, SDC Occurance, Status, Result, Trace, and Select All. The table contains 10 rows of data:

Run Option	Fault Injection Type	Index	Line	Cycle	Bit	SDC Occurance	Status	Result	Trace	Select All ...
1	bitflip	31	9	16	20	Occured	Injected	Nil	<input checked="" type="checkbox"/>	
2	bitflip	26	11	13	5	Occured	Injected	Nil	<input checked="" type="checkbox"/>	
3	bitflip	5	N/A	5	21	Not Occured	Injected	Program crashed, terminate...	<input type="checkbox"/>	
4	bitflip	5	N/A	5	42	Not Occured	Injected	Program crashed, terminate...	<input type="checkbox"/>	
5	bitflip	27	11	14	28	Occured	Injected	Nil	<input type="checkbox"/>	
6	bitflip	28	11	31	7	Occured	Injected	Nil	<input type="checkbox"/>	
7	bitflip	28	11	23	14	Occured	Injected	Nil	<input type="checkbox"/>	
8	bitflip	31	9	32	30	Occured	Injected	Nil	<input type="checkbox"/>	
9	bitflip	2	11	2	51	Not Occured	Injected	Program crashed, terminate...	<input type="checkbox"/>	
10	bitflip	31	9	48	0	Occured	Injected	Nil	<input type="checkbox"/>	

A 'Command Line Input' section at the bottom left contains the number 5.

# Outline

- Motivation
- LLFI: High Level Fault Injector
- **Experimental Setup and Results**
- Conclusion

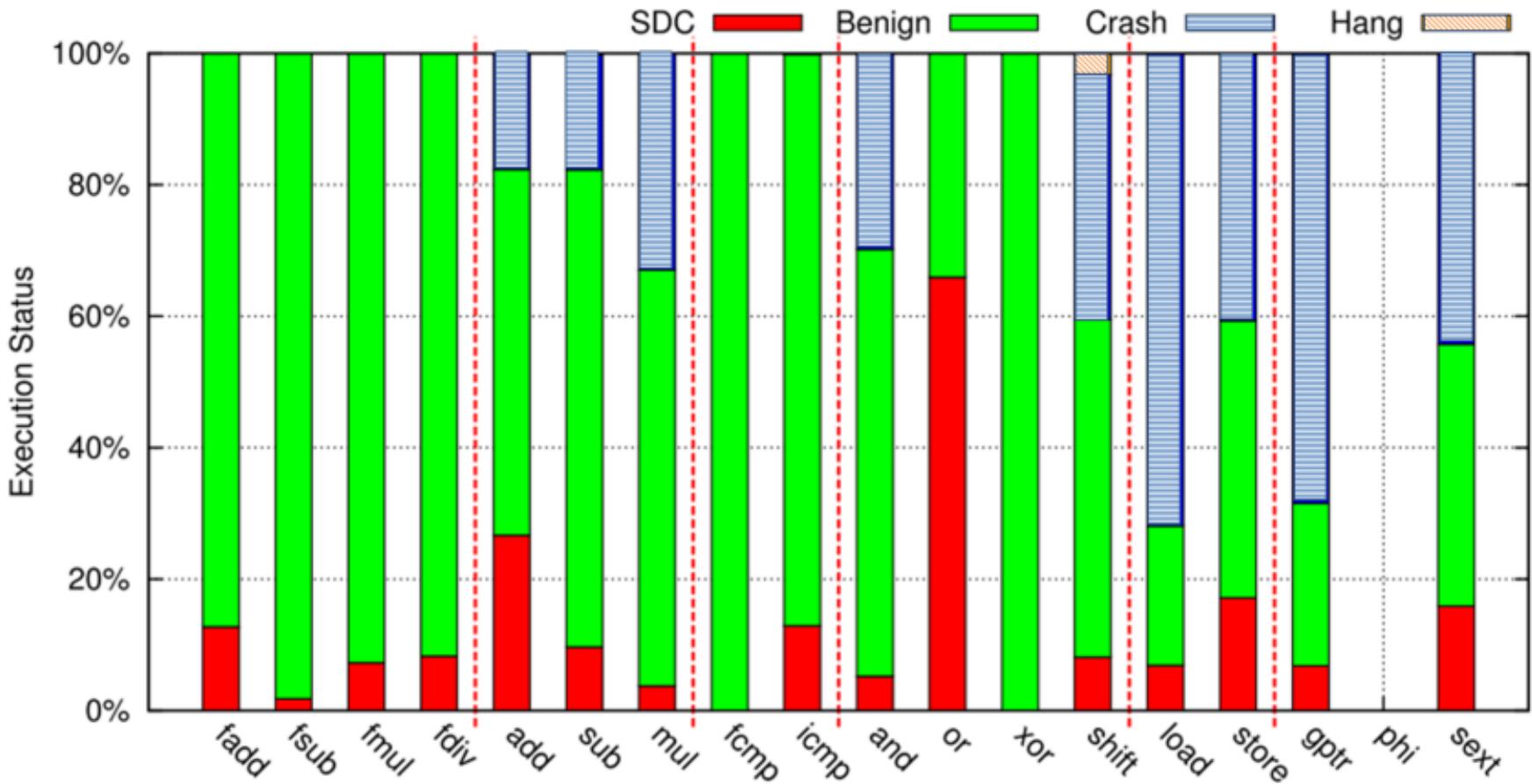
# Research Questions

- RQ1: Effect of injecting faults into different types of instructions in the program
- RQ2: Effect of injecting faults into different kinds of registers (source or destination)
- RQ3: Effect of single bit flip vs. double bit flip fault type, and ‘distance’ between them

# Experimental Setup

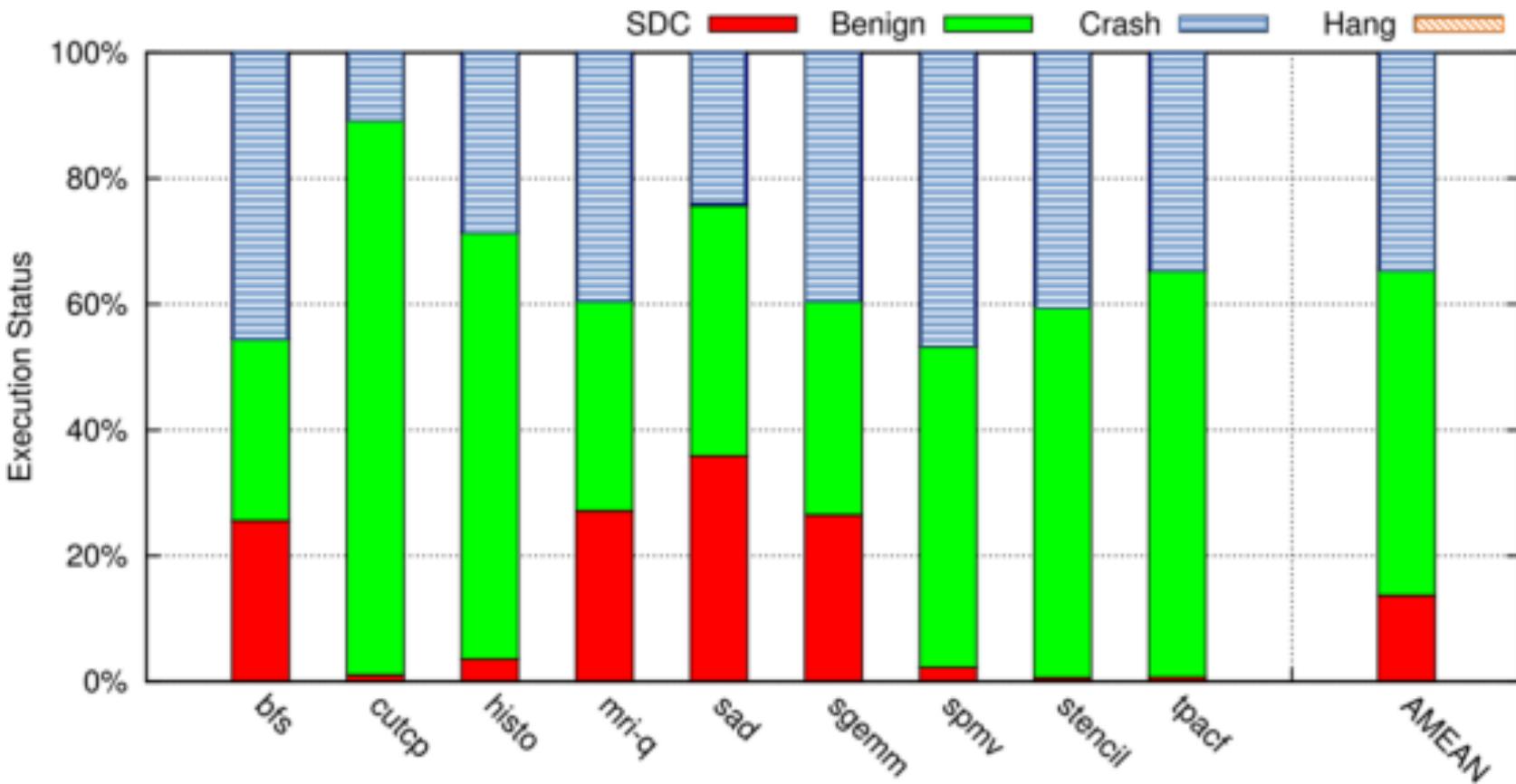
- **Benchmarks:**
  - *PARBOIL* benchmark suite for CPUs
- **Method:**
  - 1000 fault injections per benchmark per experiment
  - One fault per run (single bit flip)
- **Outcomes:**
  - Crash, Hang, Silent Data Corruption (SDCs) or Benign

# Results: RQ1 (Instruction Types)



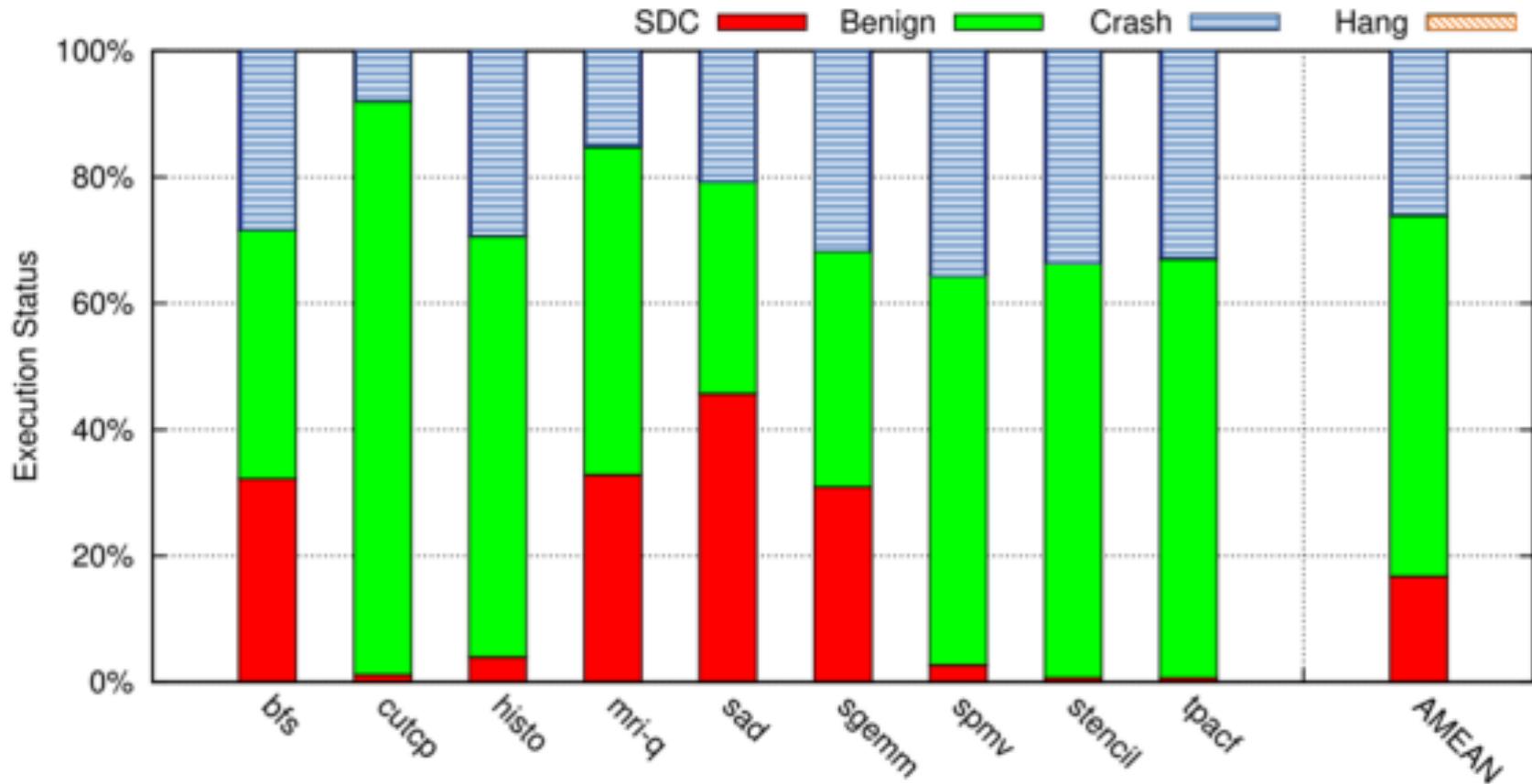
Failure distribution of injecting different instruction types

# Results: RQ2 (Register Types)



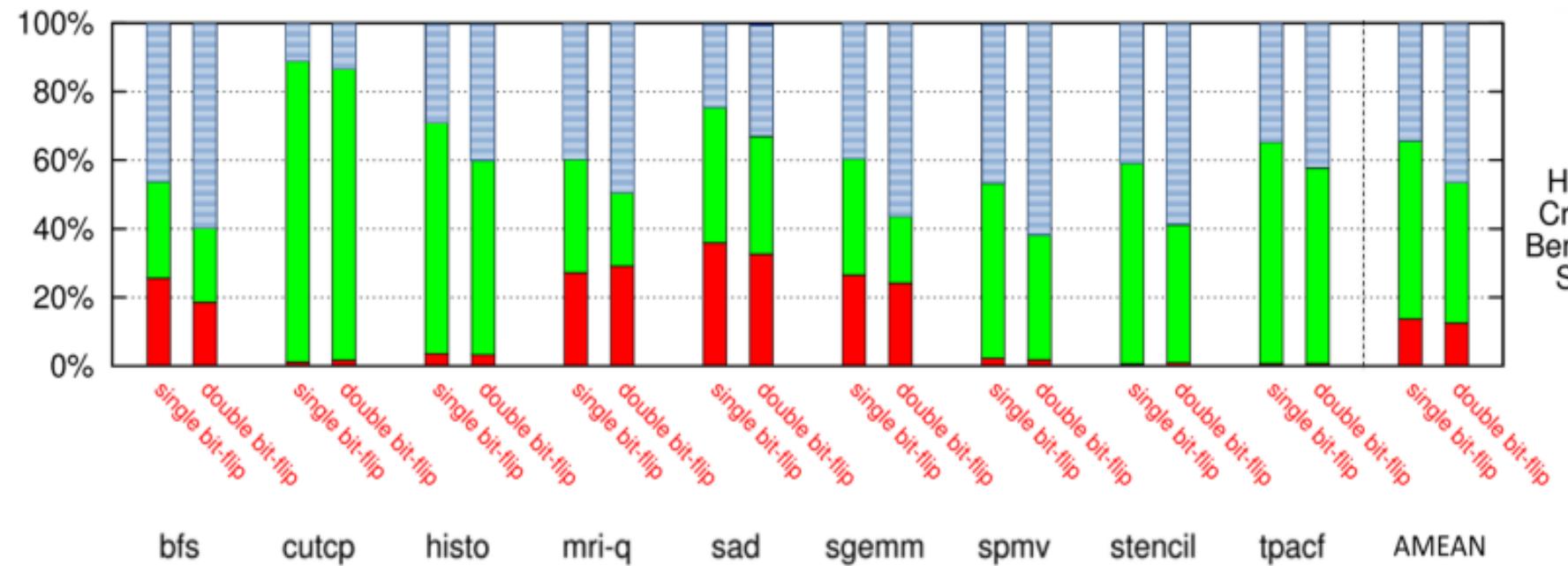
Failure distribution of injecting into source register

# Results: RQ2 (Register Types)



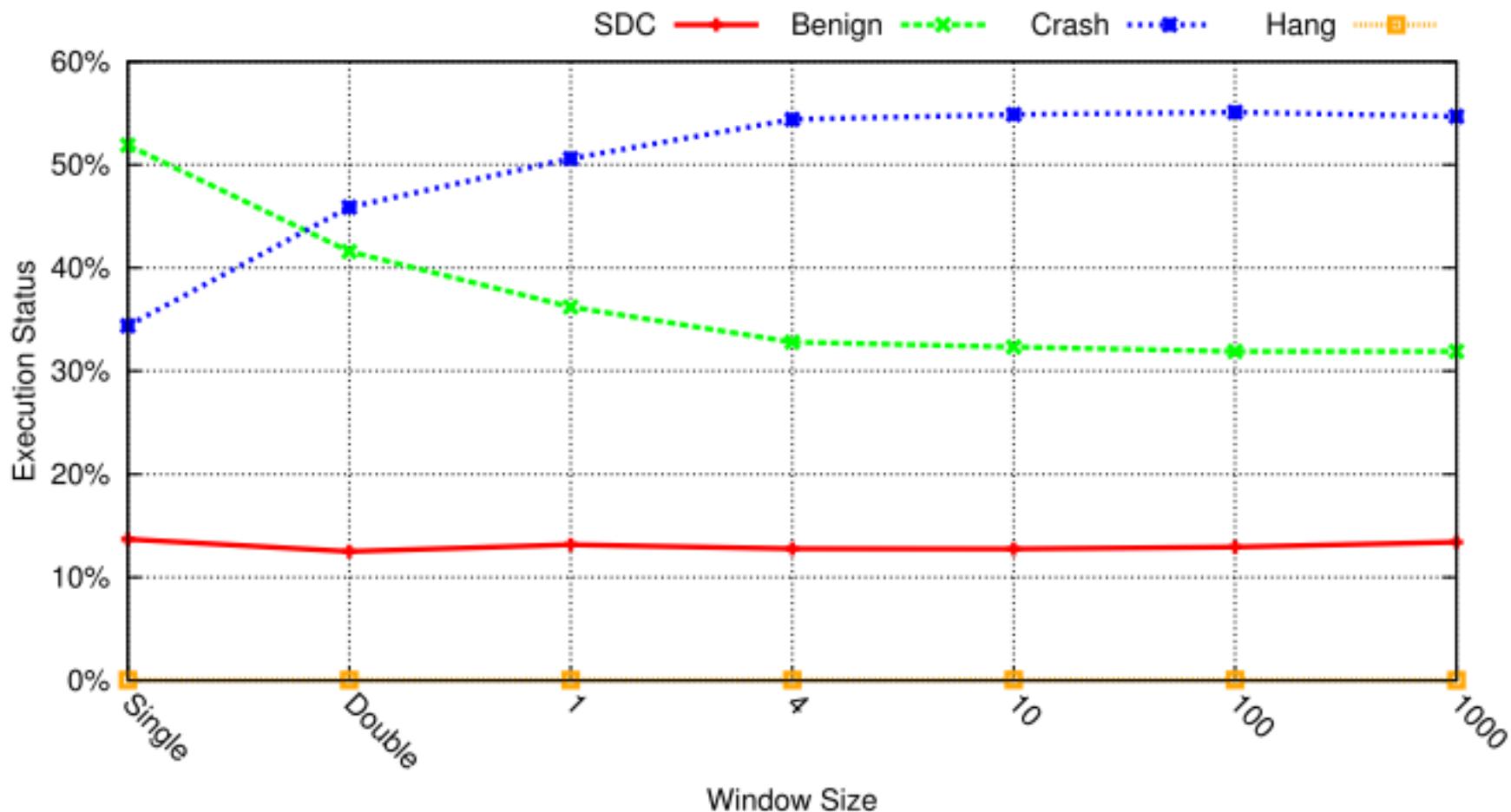
Failure distribution of injecting into destination register

# Results: RQ3 (Single vs. Double Bit Flip in same Register)



Failure distribution of single bit-flip vs. double bit-flip

# Results: RQ3 (Window Size)



Effect of window size for double bit flip

# Outline

- Motivation
- LLFI: High Level Fault Injector
- Experimental Setup and Results
- Conclusion

# Conclusion

- LLFI is a highly configurable fault injection tool that works at the LLVM IR level
- Both Instruction type and register target are highly correlated with failure outcome
- Double bit fault increases the crash rate but not the SDC rates, and that too only within a short window of <5 instructions

<https://github.com/DependableSystemsLab/LLFI>