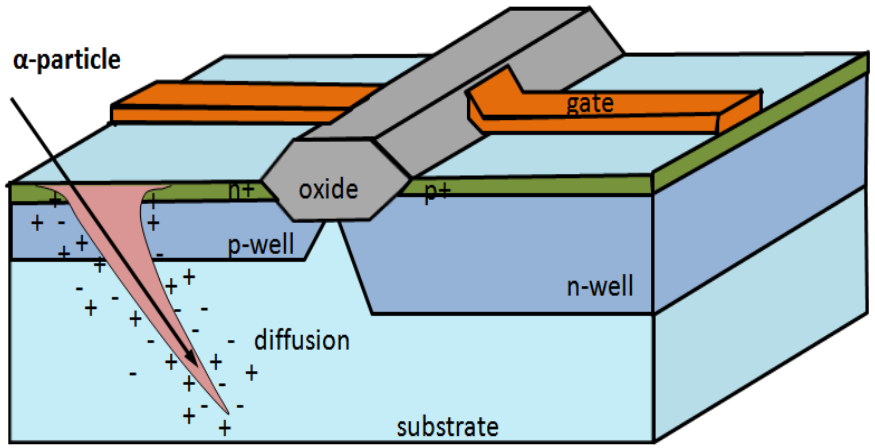
Modeling Input-Dependent Error Propagation in Programs

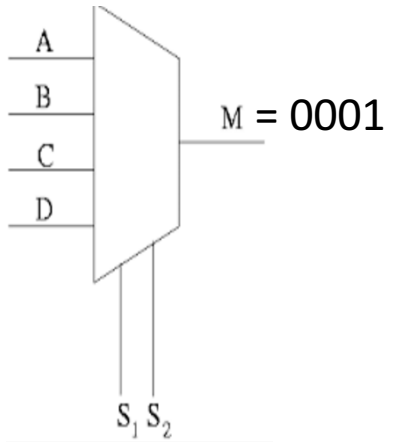
Guanpeng (Justin) Li and Karthik Pattabiraman

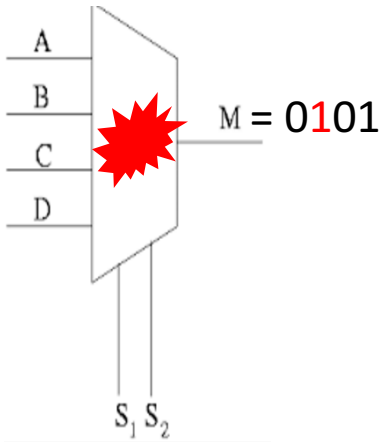
University of British Columbia

Soft Errors

Researchers have expected modern software applications to tolerate hardware errors





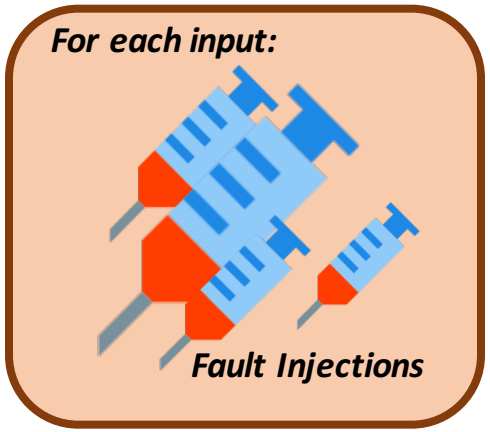


- Error propagation in programs
  - Silent Data Corruption (SDC)
    - Incorrect program output
  - Crash
  - Benign
- Traditional solutions are too expensive
  - Hardware duplication
  - Circuit hardening

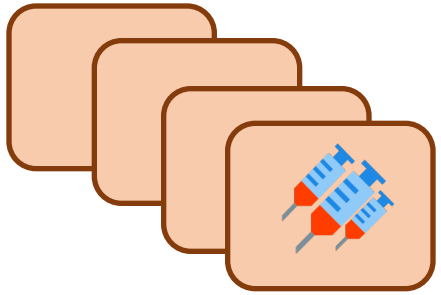
Bounding SDC Rate

+

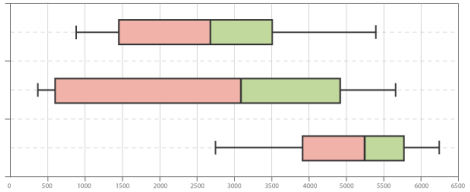
Pool of Representative Inputs



… …

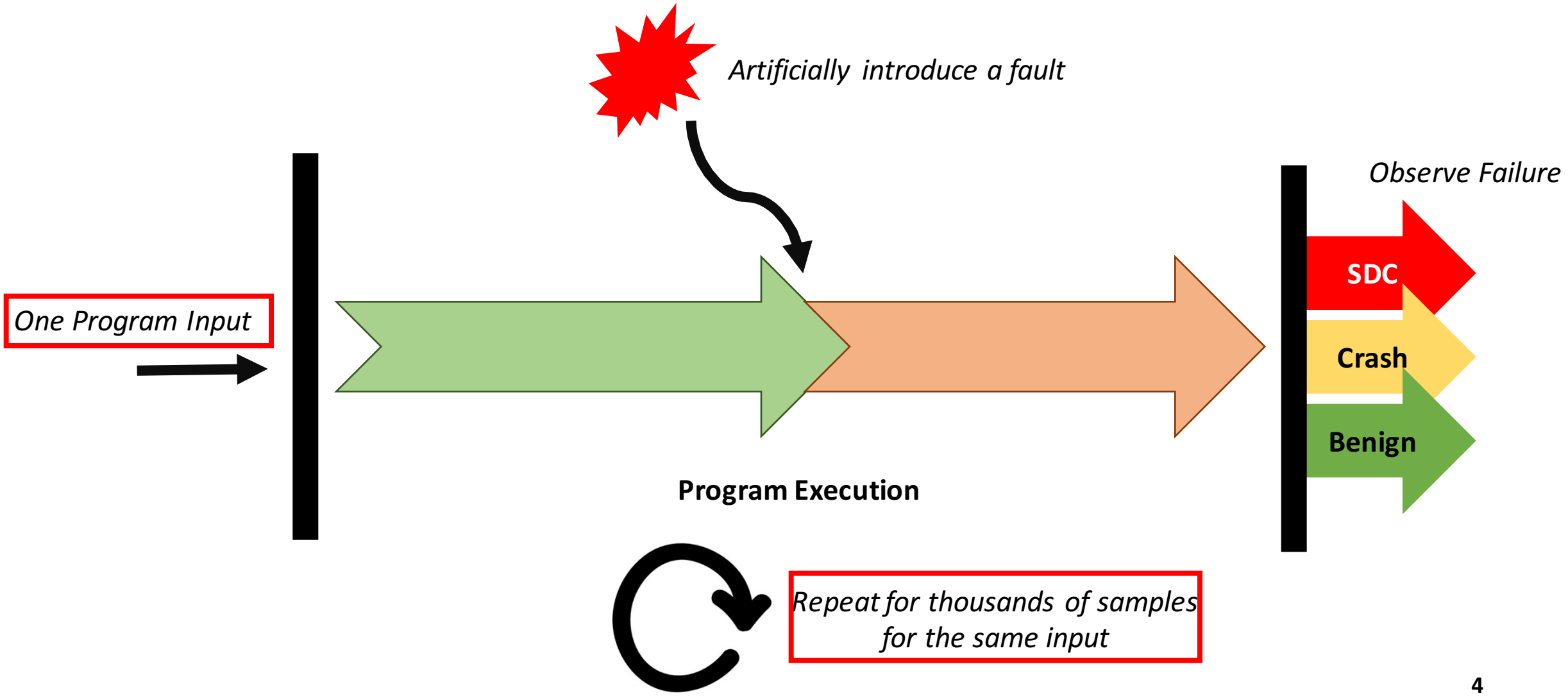


Evaluation of Program SDC Rate



Bound of Program SDC Rate

Fault Injection for Measurement of SDC Rate



Problems

Already time-consuming for only one program input

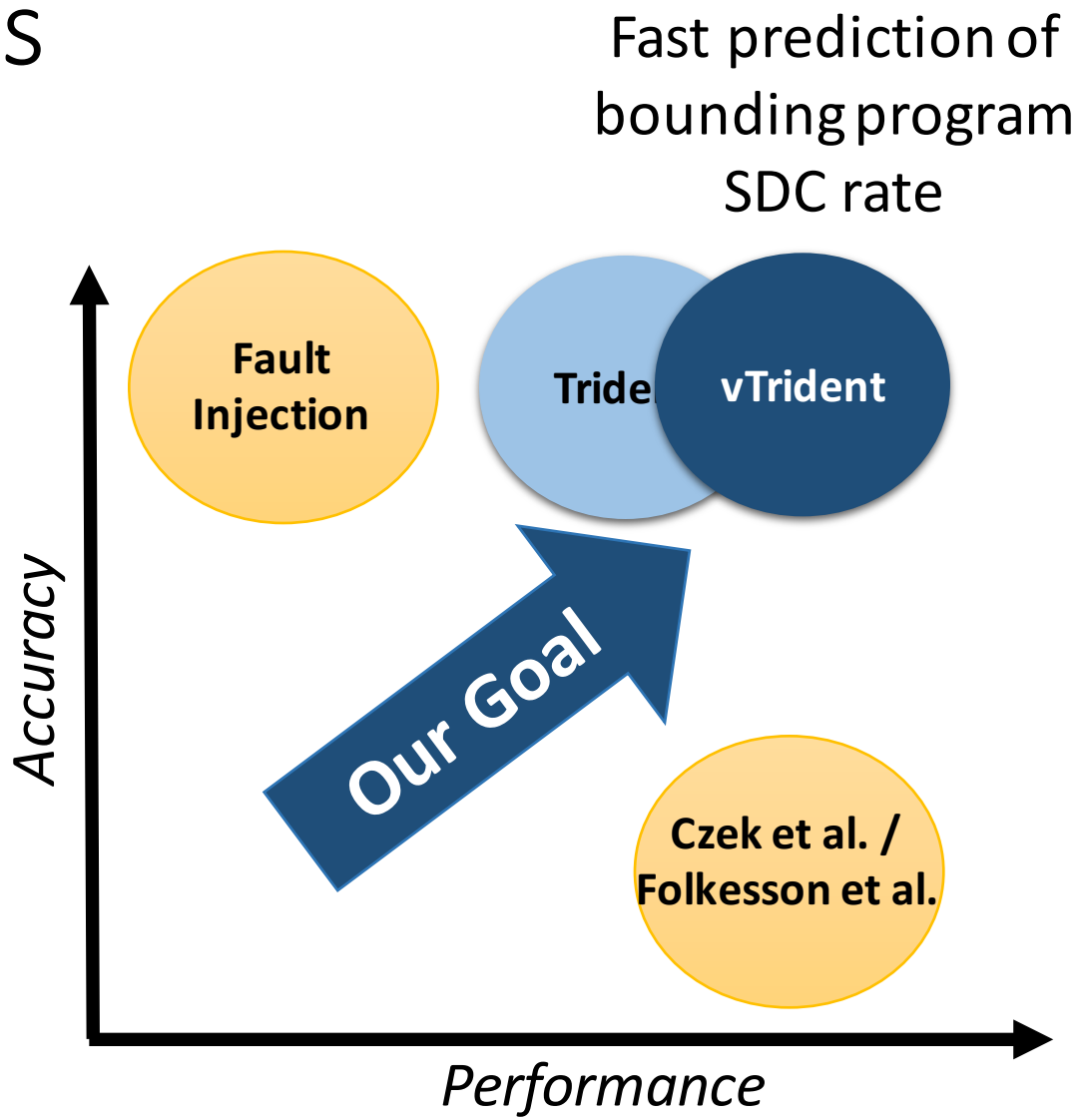
- Fault Injection
  - Even one execution may take hours in a large program
  - Need to repeat for thousands of samples for one input
- SDC is highly Input-Dependent
  - SDC rate changes if program input is changed
  - Repeat the whole fault injections per each input
- Even worse …
  - Need to re-do the whole evaluation every time code is changed

Bounding program SDC rate takes too much time

Impractical to integrate into development cycle

Our Goals & Contributions

1. Understand how different program inputs affect error propagation
2. Develop a fast model to bound the SDC rate of the program given multiple program inputs



Challenges

- Fault injection approach is black-box
- Don’t know what happen during the execution of billions of instructions

Approach

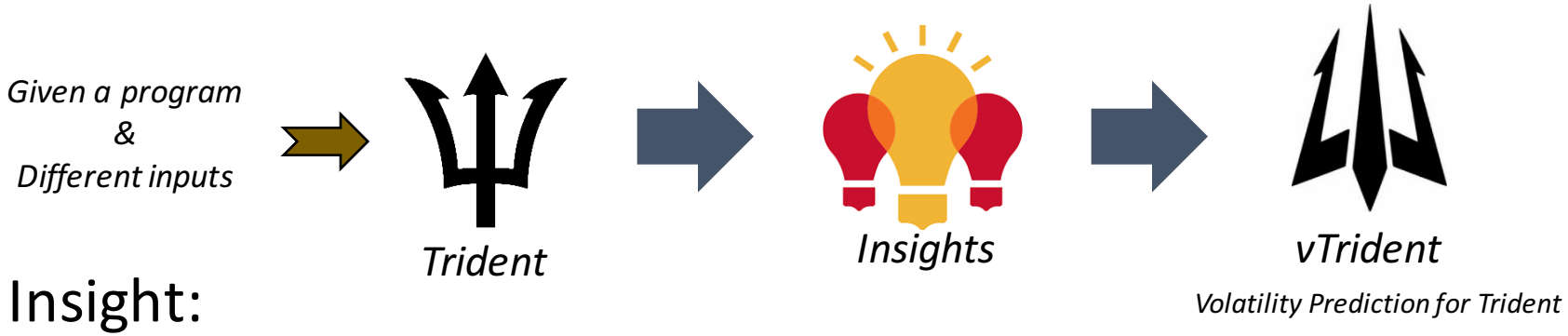
- Understand what to model
  - Our analytical model: Trident
  - Closed formula of error propagation

Trident: Three-level modeling

- Register-communication module
- Control-flow module
- Memory dependency module

$$SDC\ propagation = f_s \bullet f_c \bullet f_m$$

$$= x_1 \bullet x_2 \bullet x_3 \ldots \ldots$$

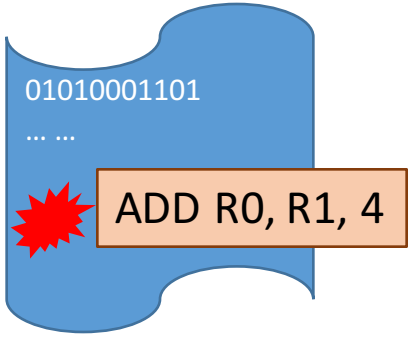


- Key Insight:
  - Only some parts of the entire modeling are critical to bound program SDC rate
  - Remove the parts that are not sensitive to the change of input from Trident

What to model?

Program SDC Rate = f (Inst Exec, Inst SDC Rate)

- Program SDC rate is an aggregation of both …
  - Instruction Execution
  - Instruction SDC rate



	Input A	***Input B***
Variation of Instruction Execution	400 times	100 times
Variation of Instruction SDC Rate	40%	70%
Variation of Program SDC Rate	*Program SDC rate changes!*	

Easy to profile

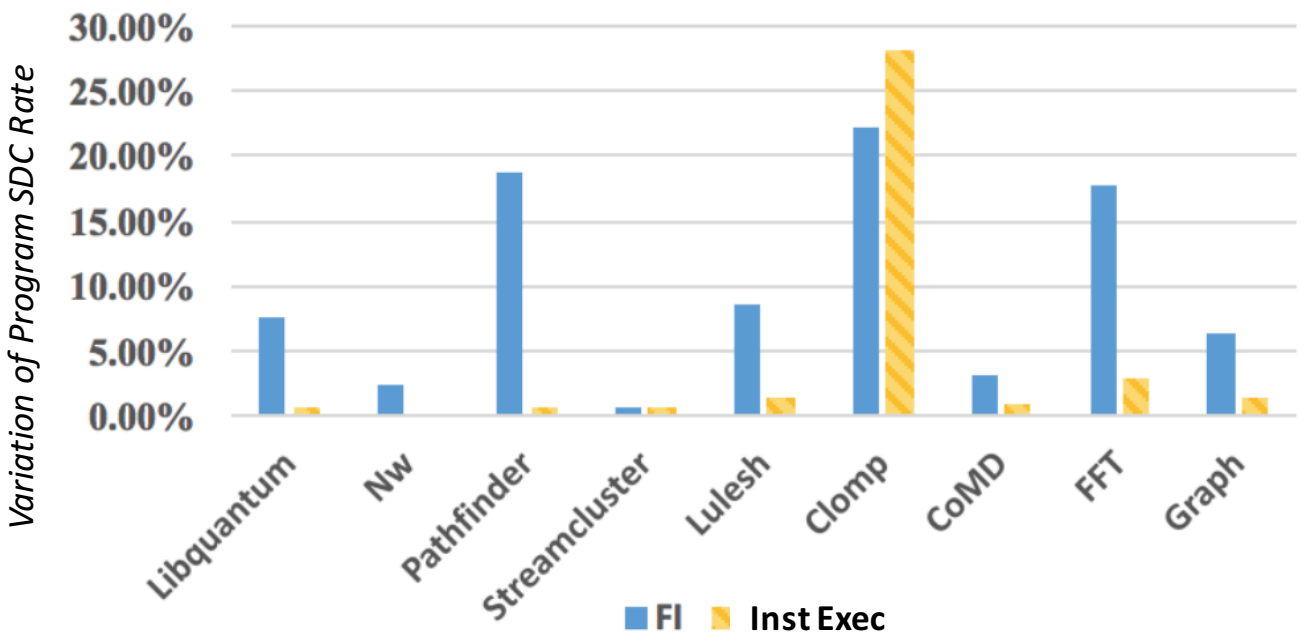
Hard to model

Need to model the variations of both instruction SDC rate and instruction execution!

Prior Work

Program SDC Rate = f (Inst Exec, Inst SDC Rate)

- Only models the variation of instruction execution …
  - Poor accuracy to quantify the variation of program SDC rate

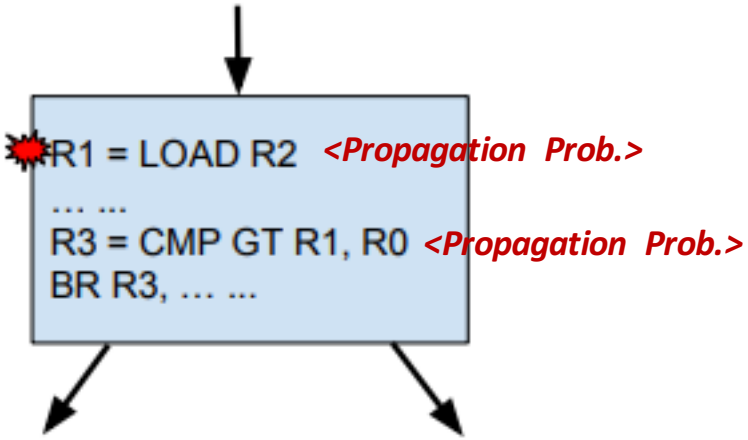


Error Bar: 0.03% - 0.55% at 95% Confidence

Variation of Instruction SDC Rate

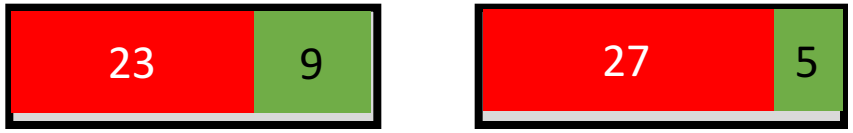
Program SDC Rate = f (Inst Exec, Inst SDC Rate)

One of examples …



CMP GT R1, R0

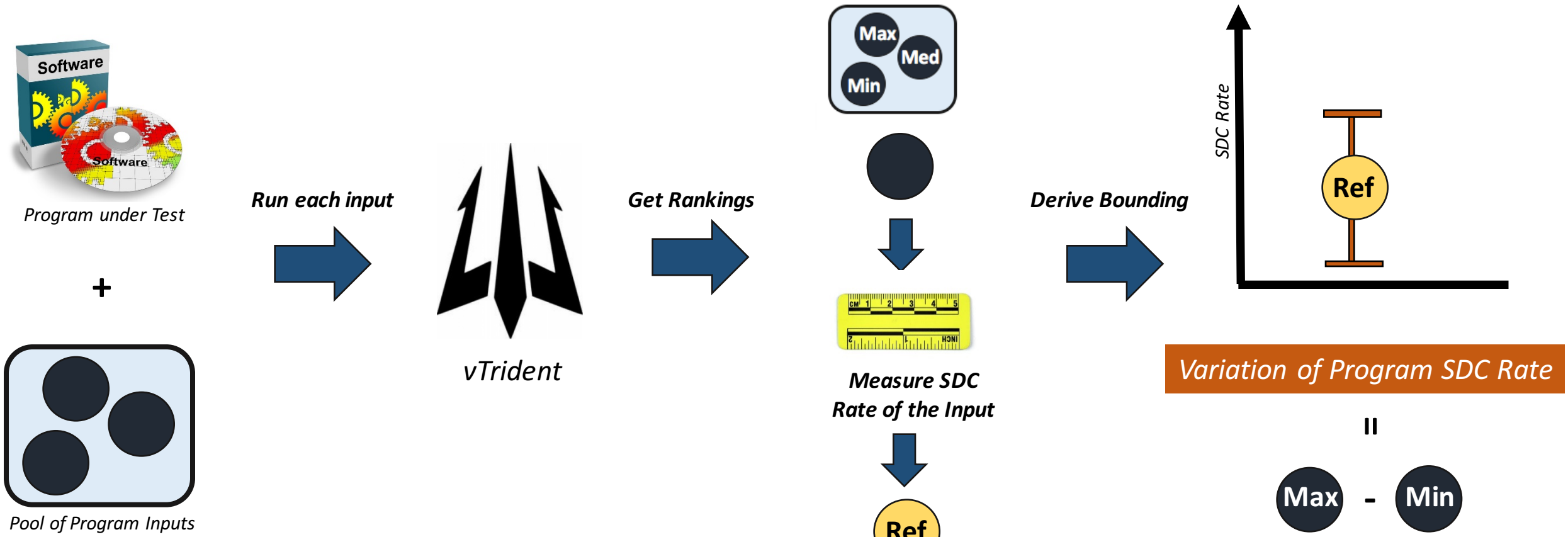
	Input A	*Input B*
Reg. Value	R0 = 512 R1 = 16	R0 = 64 R1 = 16
<Propagation Prob.>	~ 71%	~ 84%



32-bit Data Width

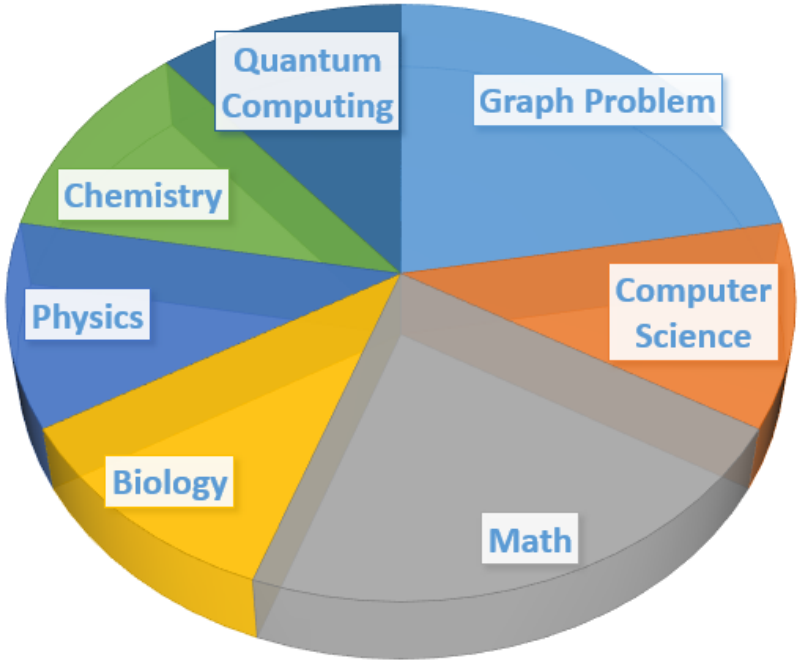
vTrident: Steps

- Bounding program SDC rates for given inputs



Experimental Setup

- Comparison of the Bounding of SDC Rate
  - Accuracy and performance
  - Fault injection results derived by LLFI [1] as baseline
- Fault Model
  - Single bit-flip
  - One fault injected per program execution
- Benchmark
  - 9 open-source benchmarks from various domains taking numerical inputs
  - 10 inputs generated for each benchmark



Benchmark Application Domains

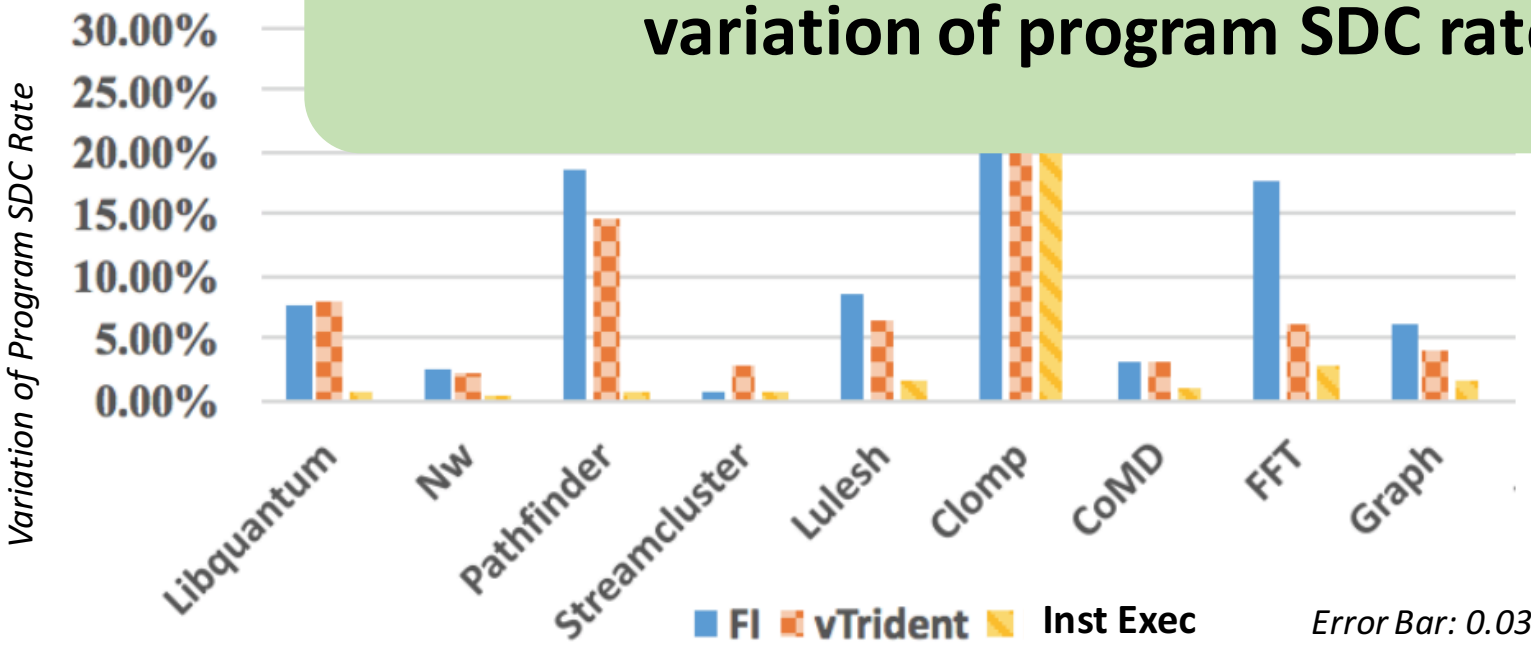
[1] LLVM Fault Injector [DSN'14]

Evaluation: Variation of Program SDC Rate

- Methodology
  - Derived by vTrident, Inst Exec, fault injection
  - The cl

Program SDC Rate = f (Inst Exec, Inst SDC Rate)

vTrident is much better in predicting the variation of program SDC rate



Variation of Program SDC Rate

=

Max - Min

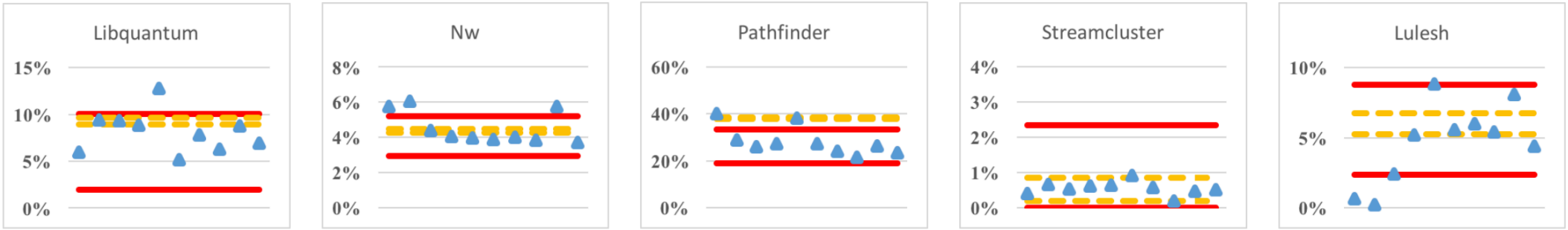
Error Bar: 0.03% - 0.55% at 95% Confidence

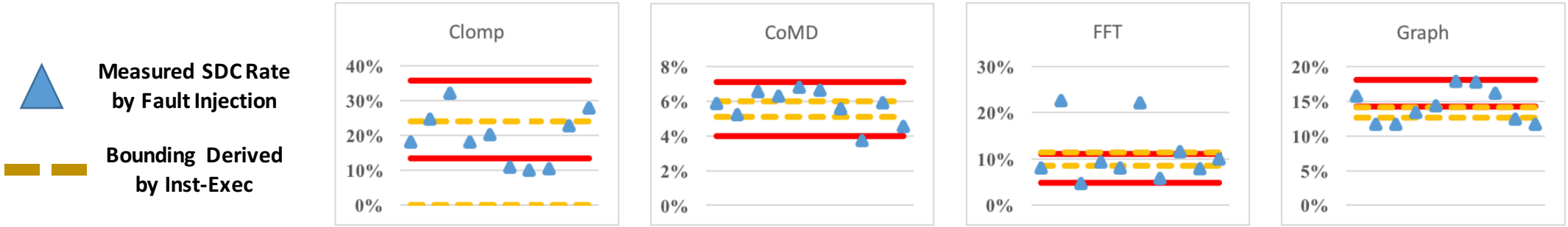
vTrident bounds 79% of SDCs whereas the other model bounds merely 32% of SDCs

Evaluation: Bounding SDC Rate

Program SDC Rate = f (Inst Exec, Inst SDC Rate)

- Methodology:
  - Ranking of SDC rates by Fault Injection and vTrident: Average distance of 2.11
  - Bounding of as much as measured SDC rates with the predicted variation of program SDC rate





Y-axis: SDC Rate; Error Bar: 0.03% - 0.55% at 95% Confidence

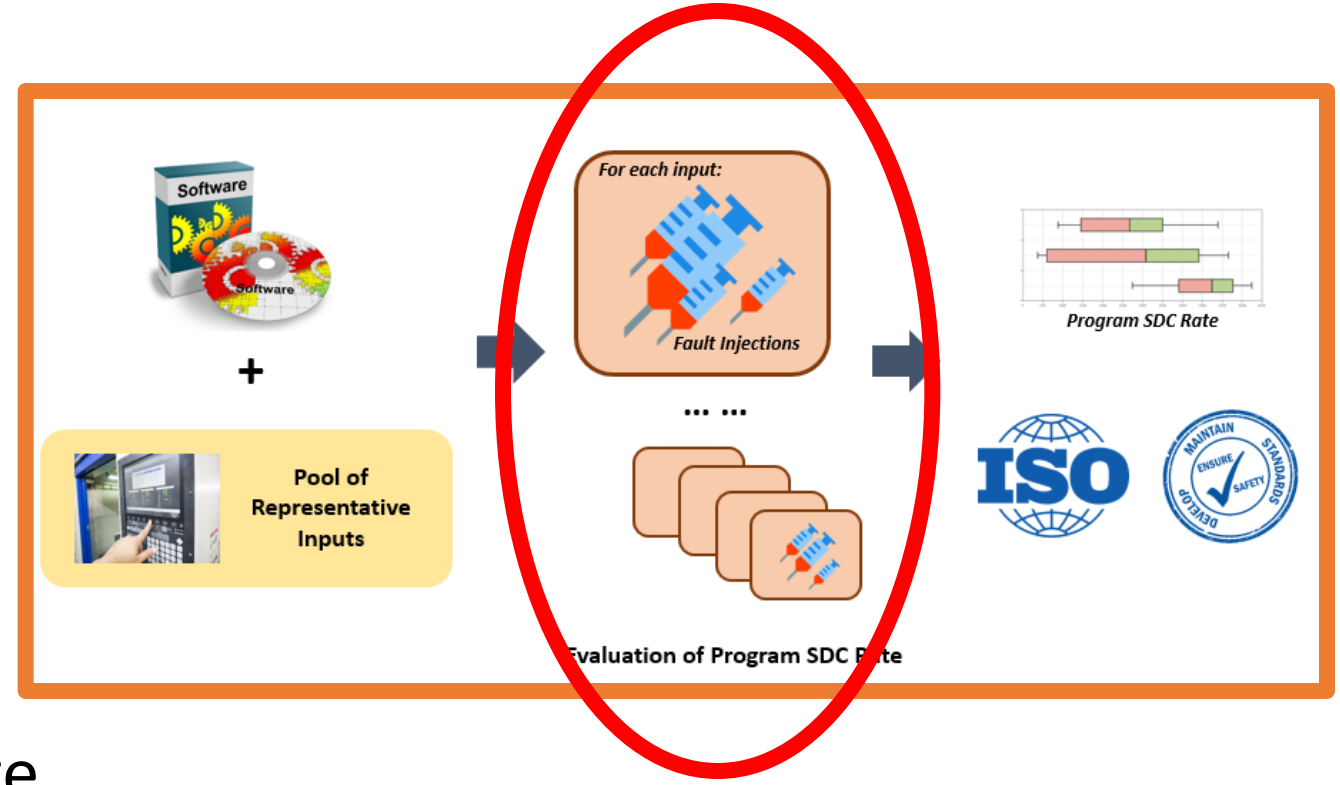
Evaluation: Performance

vTrident is significantly faster than prior techniques, requiring much less hardware resources

- Wall-Clock Time
  - Sample 3,000 faults with each input, totally 10 inputs for each benchmark
    - vTrident takes 2.6 hours, 8x faster than Trident, 37x faster than fault injection
- Memory Required (Peak)
    - vTrident: 14.97 GB
    - Trident: 4 out of 9 benchmarks requires more than 32GB memory

vTrident in Practice

- Built as compiler module
  - Fully automated
- Fast bounding of program SDC rate
  - Intergration into software development process



Now can be replaced by *vTrident*

vTrident

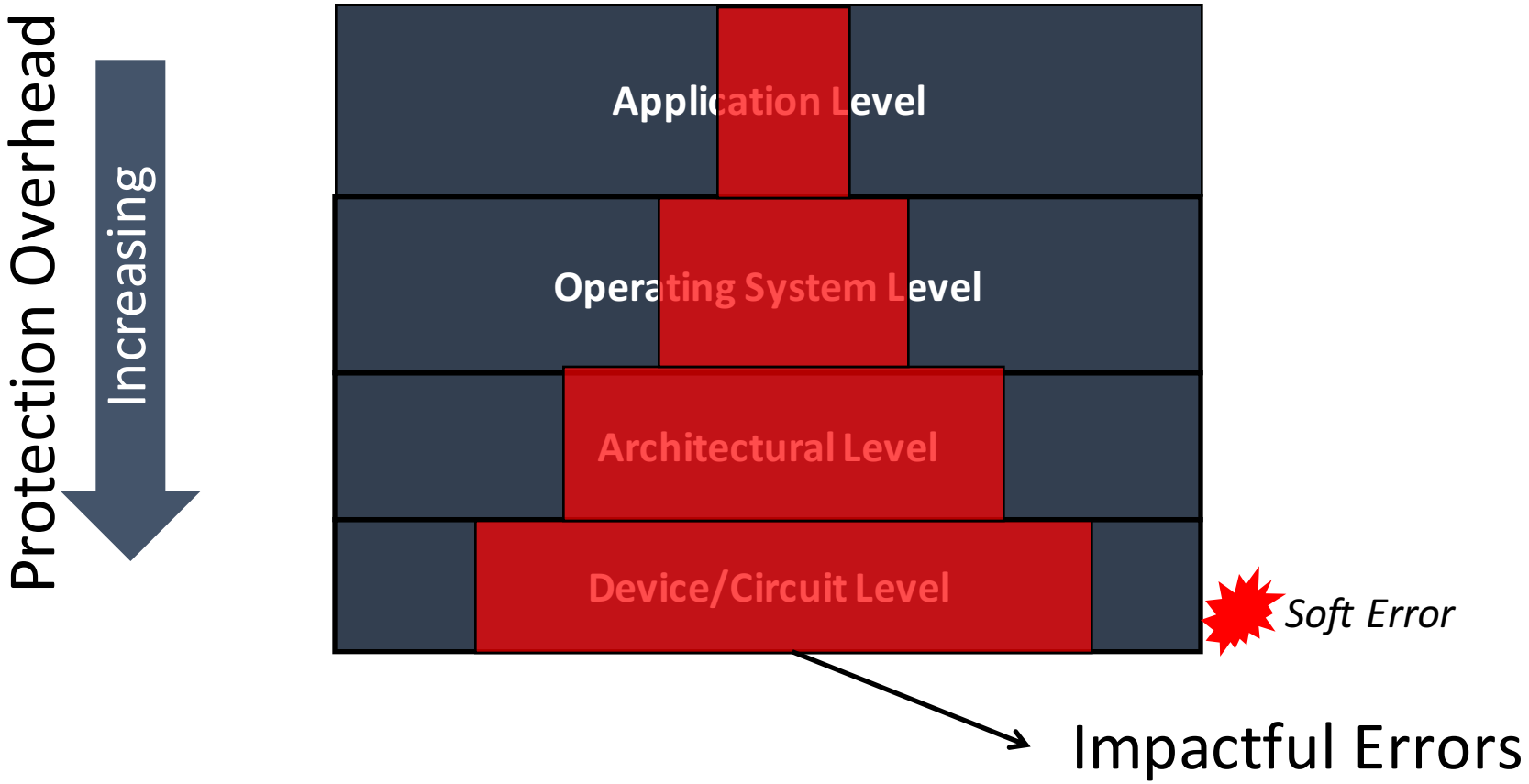
Summary

- Error propagation is highly input-dependent
- Fault injections are too slow to bound program SDC rate given multiple inputs
- Understanding input-dependent error propagation
- vTrident is a fast and accurate model to bound program SDC rate
- Open Source: Code available in the same repo of Trident
  - https://github.com/DependableSystemsLab/Trident

Guanpeng (Justin) Li
University of British Columbia
gpli@ece.ubc.ca

Backup Slides

Software Approach



vTrident: Methodology

- Modifying Trident
  - Simplify memory dependency modeling that is not sensitive to the variation
- Given inputs, vTrident …
  - Predicts the relative ranking of SDC rates
  - Determines the variation of program SDC rates

Evaluation: Performance

- Wall-Clock Time
  - Sample 3,000 faults with each input, totally 10 inputs for each benchmark
    - vTride
- Memory R
  - Average
    - vTrid
    - Trident: 28.13 GB
  - Peak Memory Consumption
    - vTrident: 14.97 GB
    - Trident: 4 out of 9 benchmarks requires more than 32GB memory

vTrident is significantly faster than prior techniques, requiring much less hardware resources