Relyzer: Application Resiliency Analyzer for Transient Faults

Siva Kumar Sastry Hari, Illinois

Helia Naeimi, Intel Labs

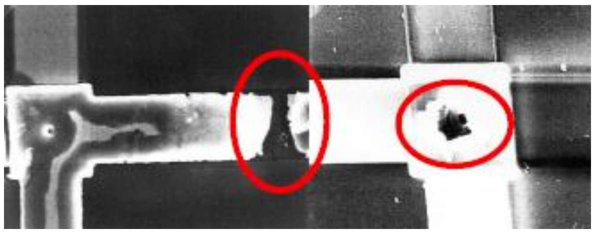
Pradeep Ramachandran, Illinois

Sarita Adve, Illinois

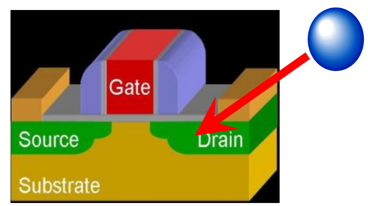
swat@cs.uiuc.edu, helia.naeimi@intel.com

Motivation

- **Reliability is a major challenge**
- **Shipped hardware is likely to fail in the field**



Wear-out Failures



Transient Errors

Design Bugs

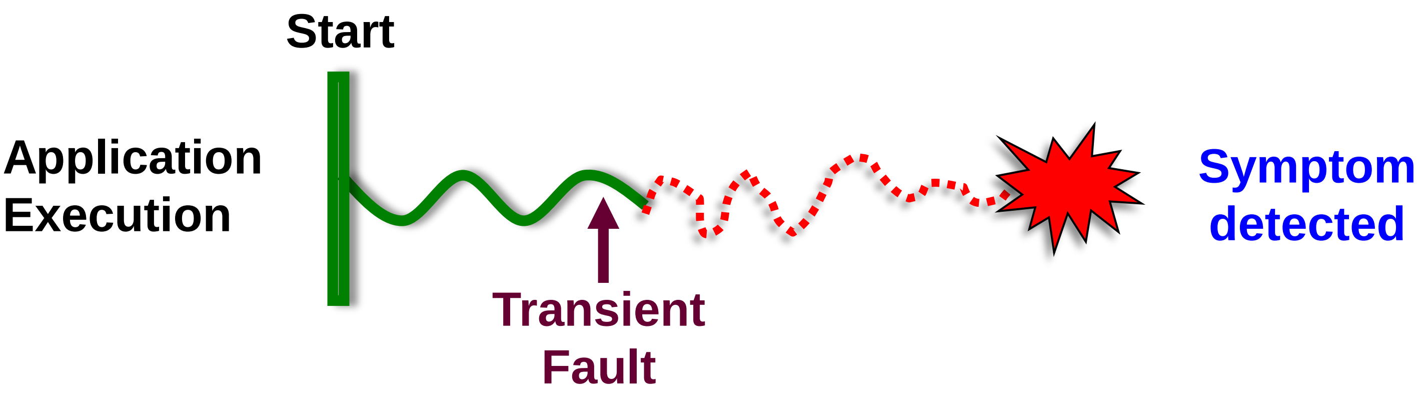
. . .

- **Reliability problem pervasive across many markets**
  - **Traditional redundancy solutions (e.g., nMR) expensive**

- **Need in-field low-cost detection, diagnosis, and repair**

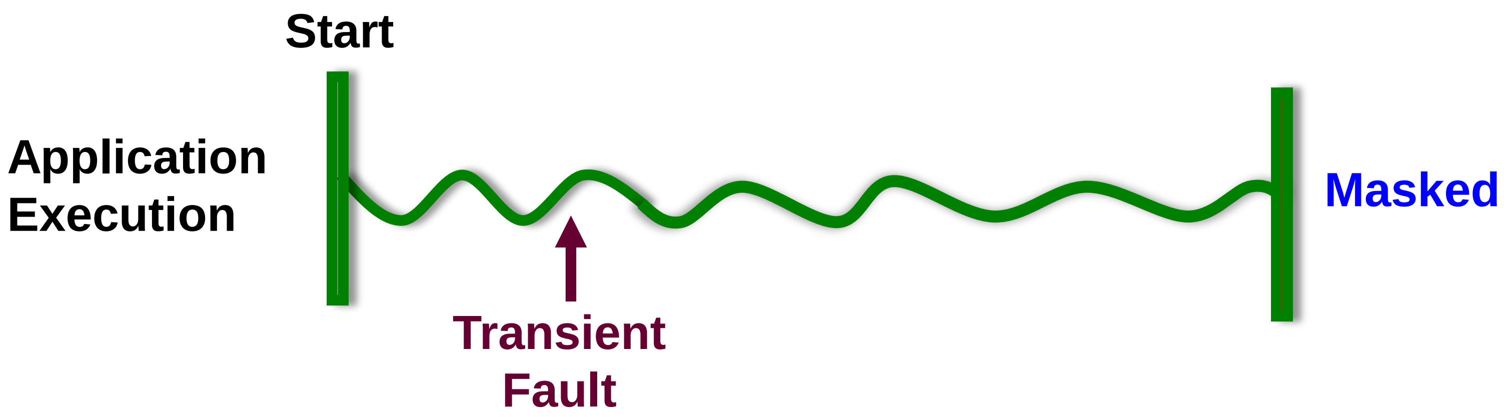
Software-level Resiliency

- **Software-level anomaly detectors are low-cost & effective**
  - **Detect faults by monitoring software misbehavior**
- **Evaluating resiliency solutions through fault injections**



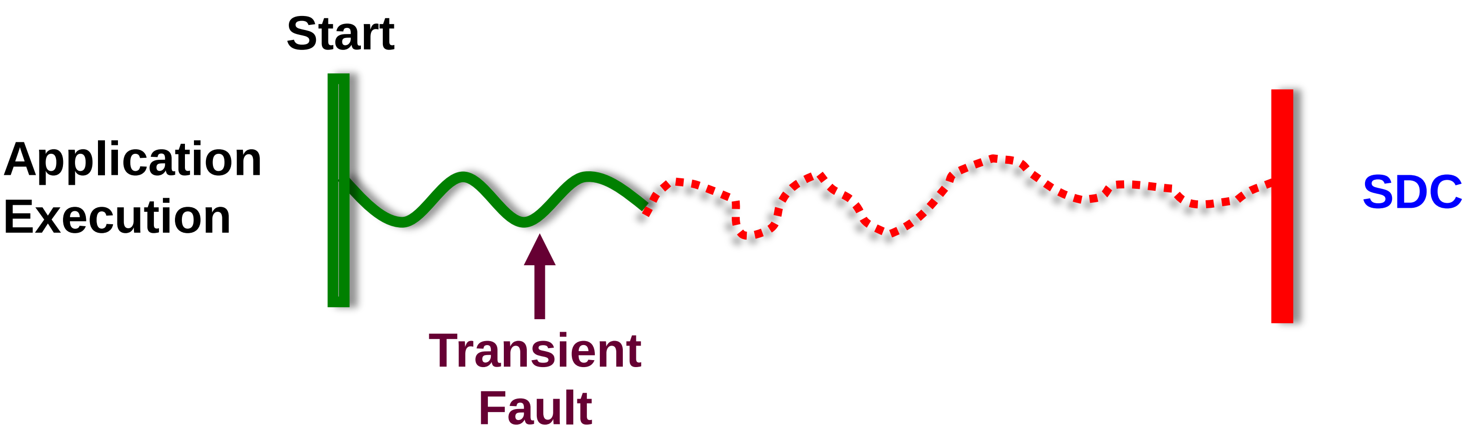
Software-level Resiliency

- **Software-level anomaly detectors are low-cost & effective**
  - **Detect faults by monitoring software misbehavior**
- **Evaluating resiliency solutions through fault injections**



Software-level Resiliency

- **Software-level anomaly detectors are low-cost & effective**
  - **Detect faults by monitoring software misbehavior**
- **Evaluating resiliency solutions through fault injections**



- **Each application has 100s of billions of fault sites**
- **Performing fault injections on all is impossible**
- **Statistical approach: study 10s of thousands of faults**
  - **Cannot provide guarantees**

Challenges

- **Goal: Analyzing all transient faults affecting an application**
  - **Provide guarantees to the detection mechanisms**

- **Do we need fault injections for all the faults?**
- **Can we reduce the # of faults that require detailed analysis?**
- **How to analyze all faults with fewer fault injections?**

Contributions

- **Prune faults with predictable outcomes**
  - **Outcome: Detected, Masked, or SDC**
  - **Such faults do not need fault injections**
- **Prune equivalent faults**
  - **Faults that behave similarly**
- **Only faults with unknown outcomes need fault injections**
  - **Developed pruning techniques to ensure they are small**
  - **Validating pruning techniques is ongoing work**
- **Analyze few faults to estimate behavior of all**

Outline

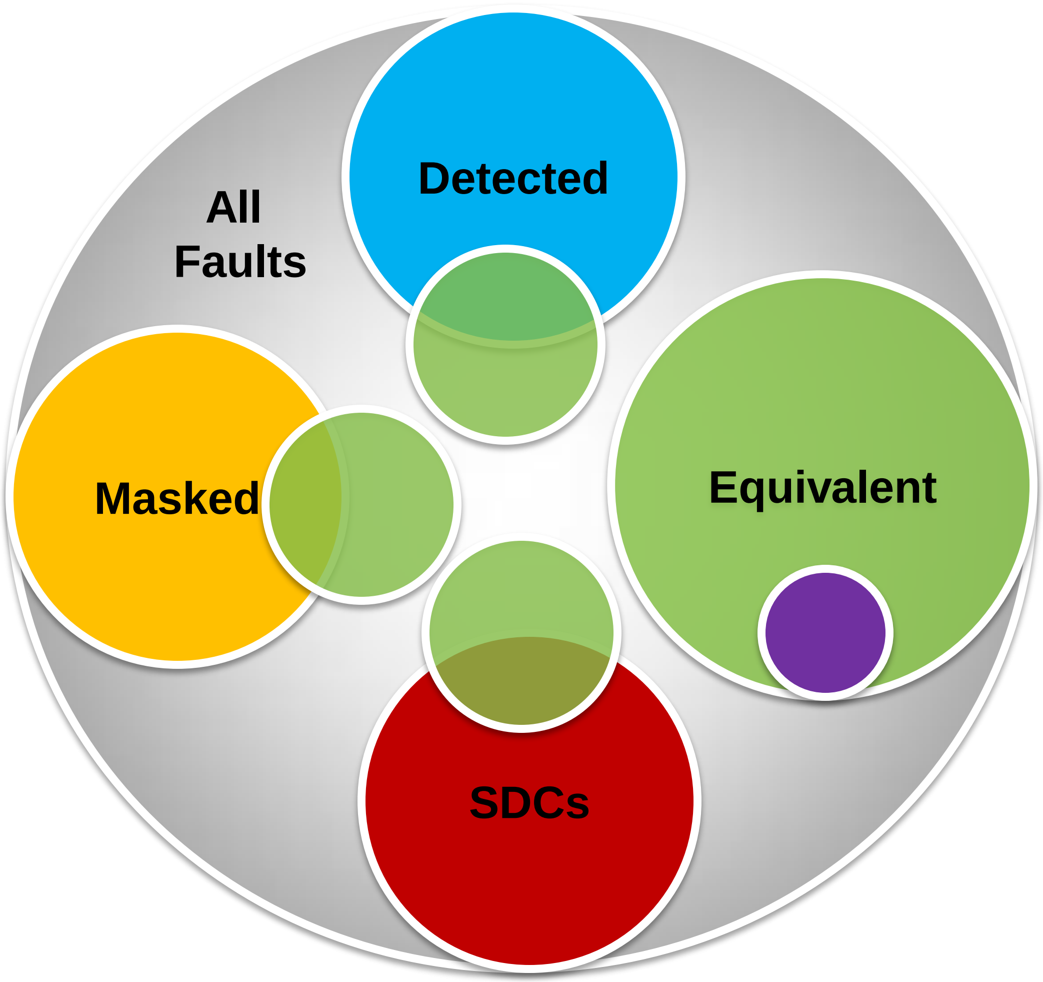
- Motivation
- **Fault Model**
- **Pruning Techniques**
- **Results**
- **Conclusion and Future Work**

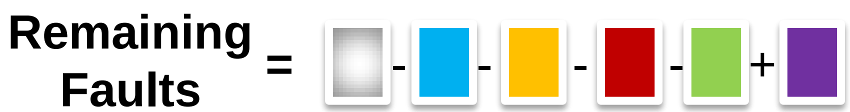
Fault Model

- **Transient faults**
  - **Single bit flips**
- **Faults in ISA-visible instruction-level states**
- **Example:**
  - **Instruction: opcode rd, rs1, rs2**
    - **One bit faults in rd, rs1, and rs2**
  - **Instructions: opcode [addr+imm], rd**
    - **One bit faults in rd, addr, imm, (addr)[addr+imm], (value)[addr+imm]**

Relyzer Overview

- **List all faults**
- **Prune predictable faults**
- **Prune equivalent faults**

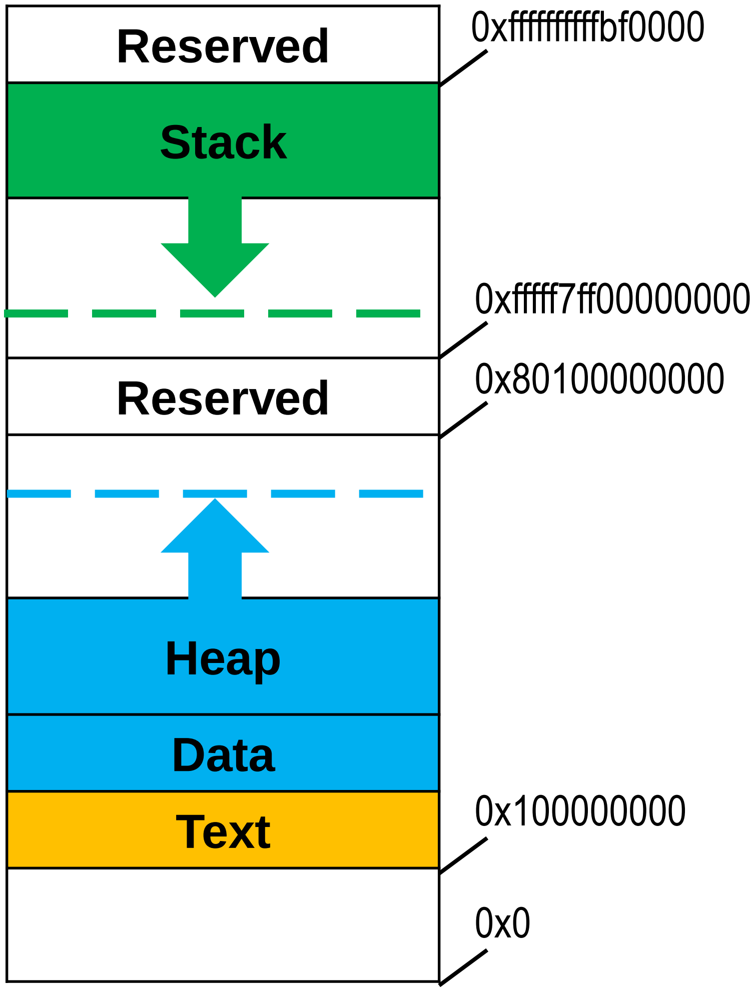




Pruning Predictable Faults

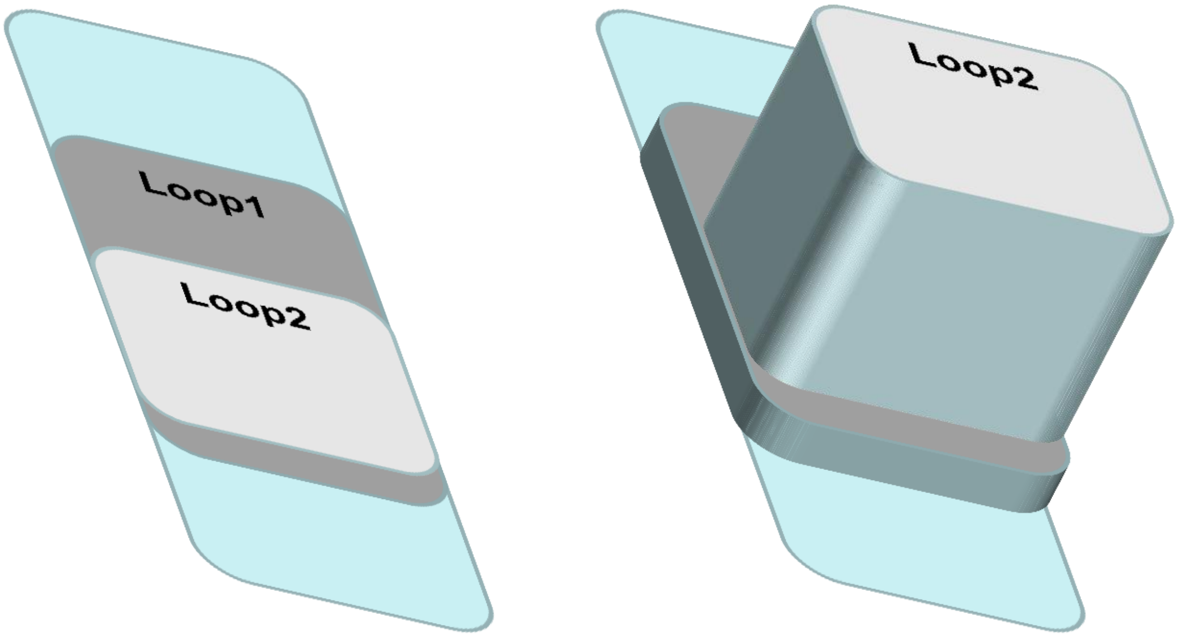
- Watch for out of bounds accesses
  - SWAT detects such accesses
- Prune out of bounds accesses
  - Memory addresses (&)
  - Branch targets ( )
- Boundaries obtained by profiling

SPARC Address Space Layout



Pruning Equivalent Faults

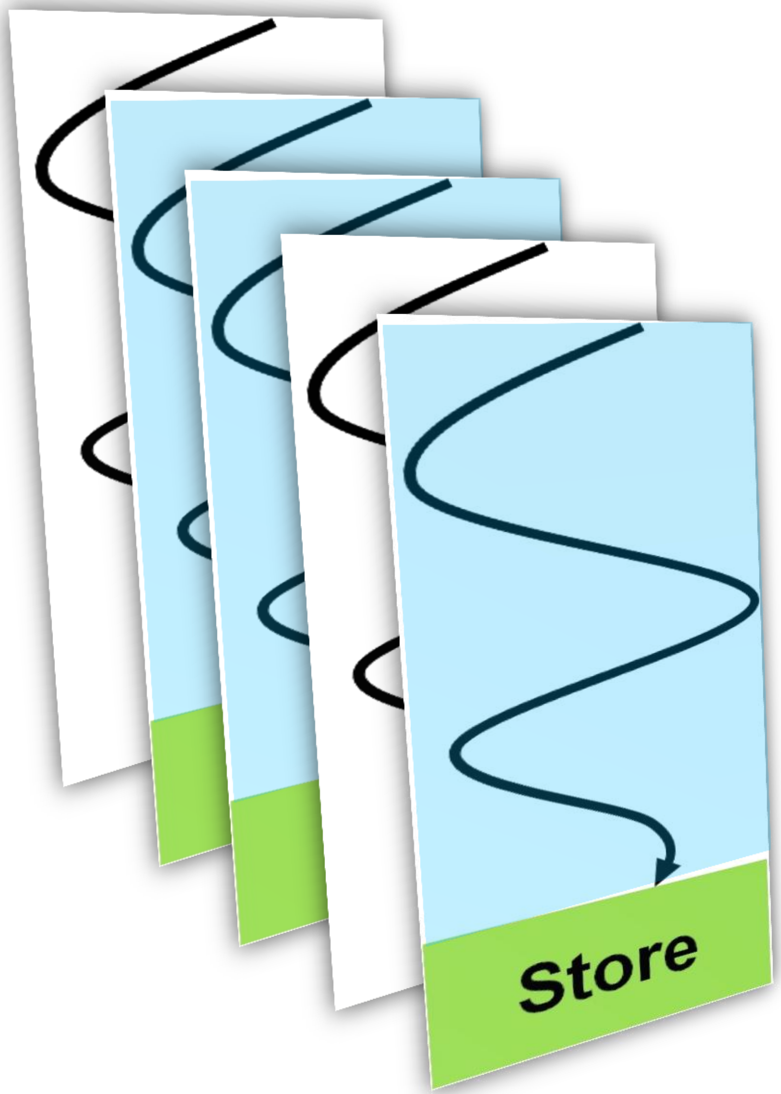
- **Key Insights:**
  - **Faults in different loop iterations may be equivalent**



  - **Stores and Branches have major effects on application**

Faults in Store Instructions

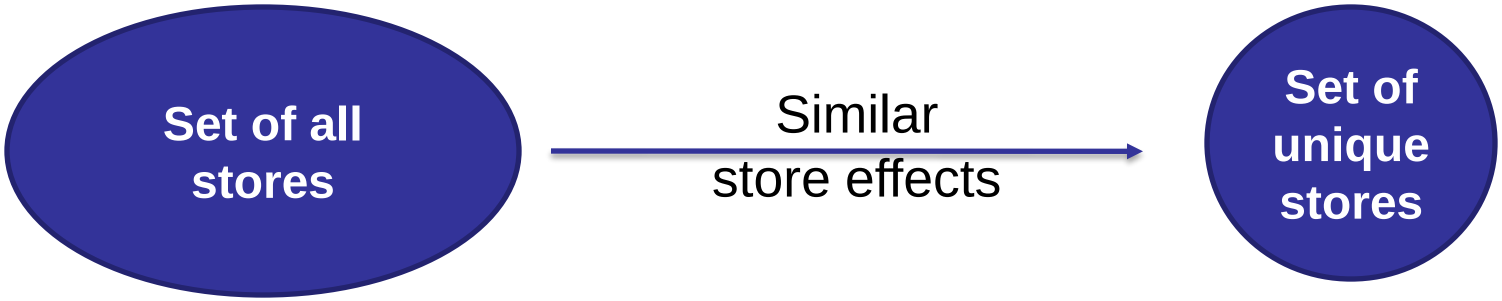
Data flow leading to a store



- **Show stores to be equivalent**
- **Faults in instructions are also equivalent**
- **How to show store equivalence?**

Equalizing store instructions

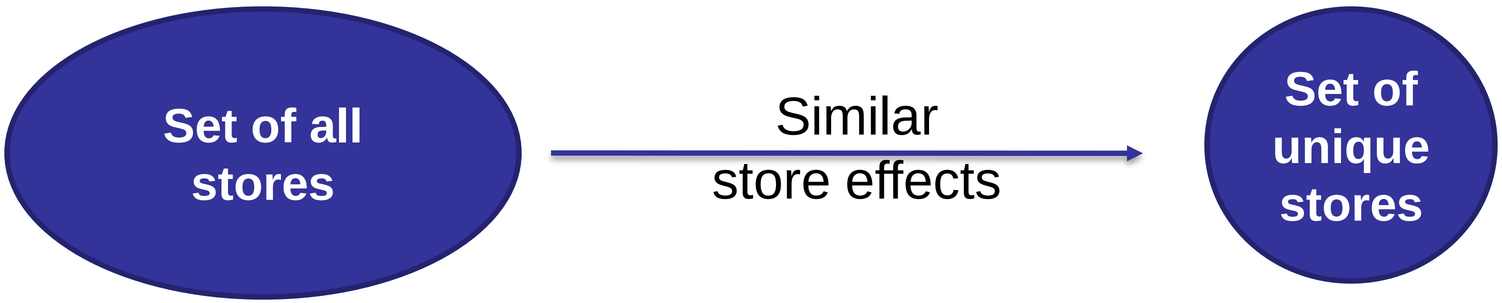
- **Criterion that makes stores equal: effect of faulty store**



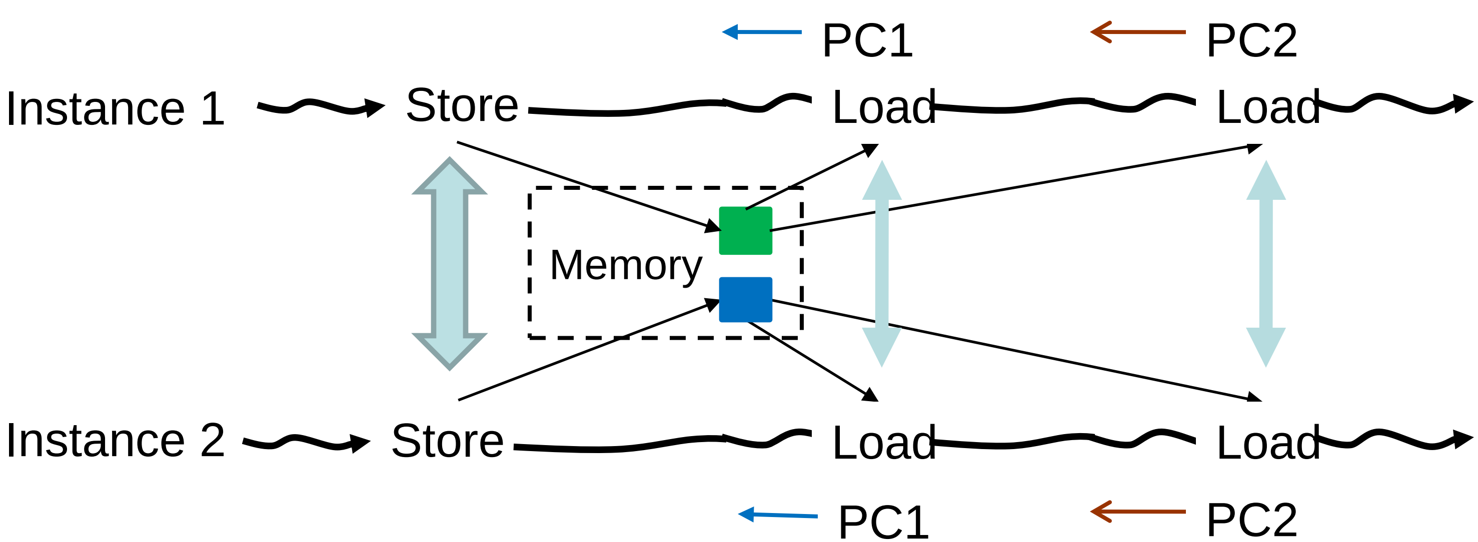
- **Use heuristics to measure store effects**

Equalizing store instructions

- **Criterion that makes stores equal: effect of faulty store**

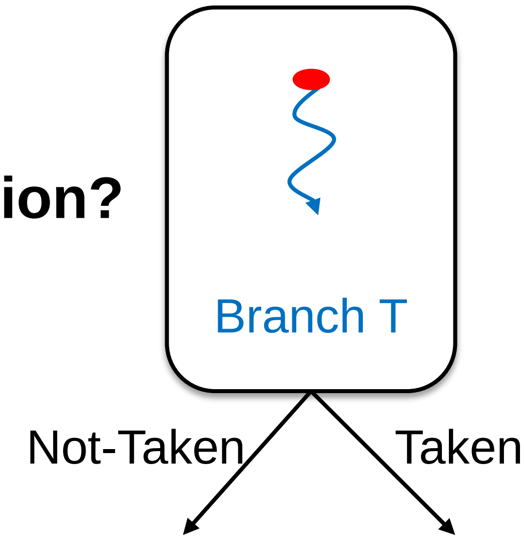


- **Use heuristics to measure store affects**
  - **Location of every use + dynamic instruction flow info**



Faults in Branches

- Faults in section affect only the direction of branch
  - No other side effects
- For section we need to know
  - # of faults resulting in right branch direction?
    - Distribution of faults
  - Outcome of the wrong direction



- For distribution, inject faults exhaustively in one iteration
- For outcome, inject just one fault in the condition code

Other Pruning Techniques

- **Constant Propagation**
  - **Propagate faults through instructions using constants**
  - **Similar to the compiler optimization technique**
- **Def-Use Analysis**

  1. OR r1, 0x1 → r2
  2. ADD r2, r3 → r4

  - **Consider faults only in the uses**
  - **Prune faults from def**
- **Statistical Pruning**
  - **Remaining faults in branch instructions**
    - **In target of dynamic branch**
    - **One direction fault in dynamic conditional branch**

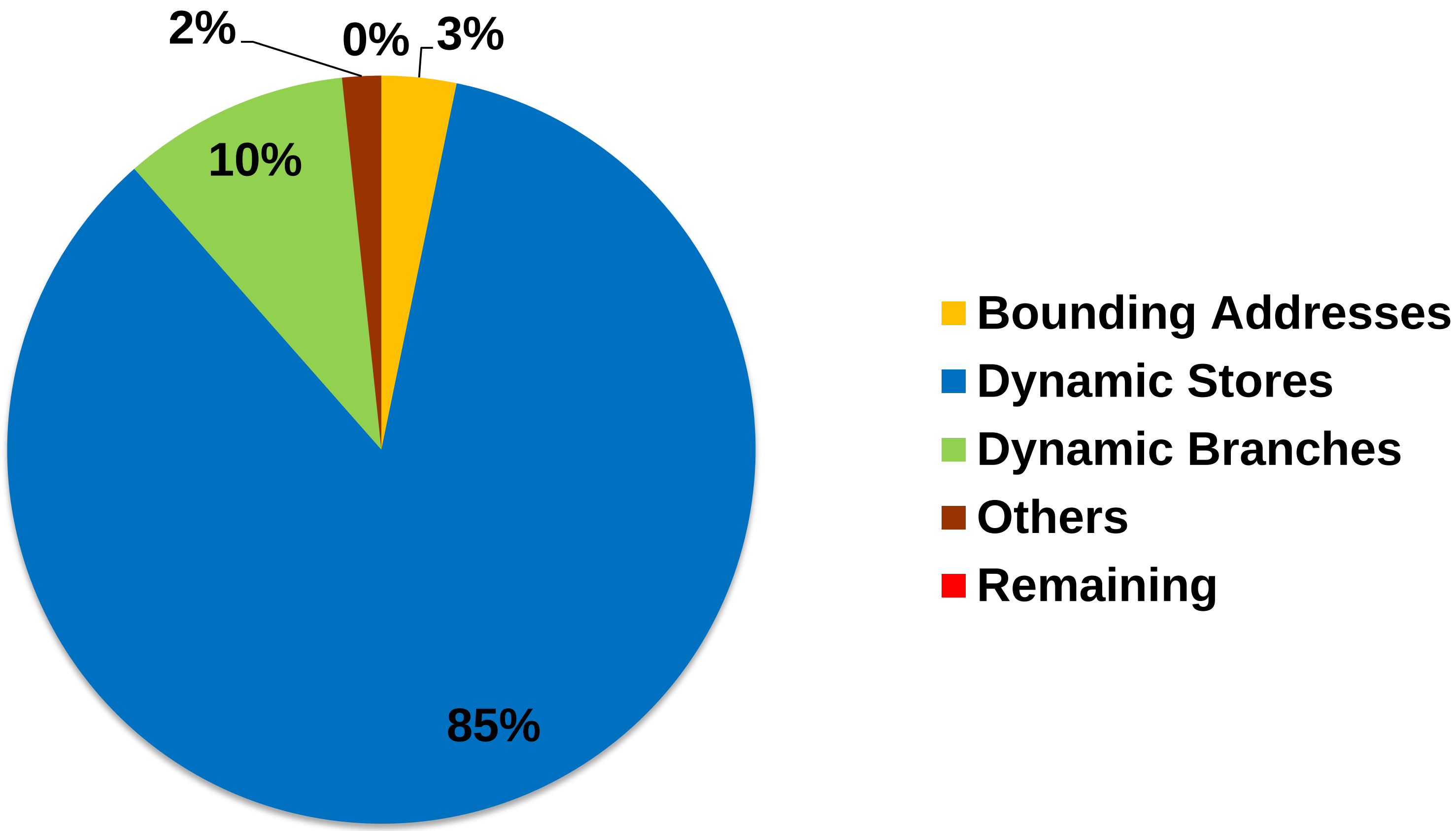
Evaluating Fault Pruning Techniques

Application	Number of Instructions	Number of Faults
LU (SPLASH-2)	2.1 Billion	310 Billion
FFT (SPLASH-2)	7.1 Billion	111 Billion
Blackscholes (PARSEC)	1.7 Billion	214 Billion
Swaptions (PARSEC)	2.7 Billion	534 Billion

Total Faults = 1.17 Trillion

Results

- **Pruned: 99.9979%**
- **Remaining: 0.0021% (25 Million)**



Fault pruning across all applications

Conclusions

- Analyzing all transient faults effecting an application
- Reduced the number of fault injection experiments
  - Developed efficient pruning techniques
    - Predict outcome without fault injection
    - Show equivalence between several faults
- Pruned hardware faults by 5 orders of magnitude
  - Only 0.0021% of faults remaining

Ongoing and Future Work

- **Developed efficient fault injection framework**
- **Evaluating the accuracy of developed pruning techniques**
  - **Current results show an average inaccuracy of only 5%**

- **Compute effective SDC rate**
- **List and analyze SDC causing faults**
- **Identify SDC prone sections of applications**
  - **Devise low-cost detectors (software or hardware)**