Relyzer: Exploiting Application-level Fault Equivalence to Analyze Application Resiliency to Transient Faults

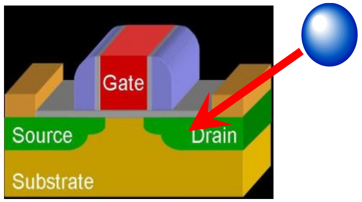
Siva Hari[1], Sarita Adve[1], Helia Naeimi[2], Pradeep Ramachandran[2]

[1] University of Illinois at Urbana-Champaign,

[2] Intel Corporation

swat@cs.illinois.edu

Motivation

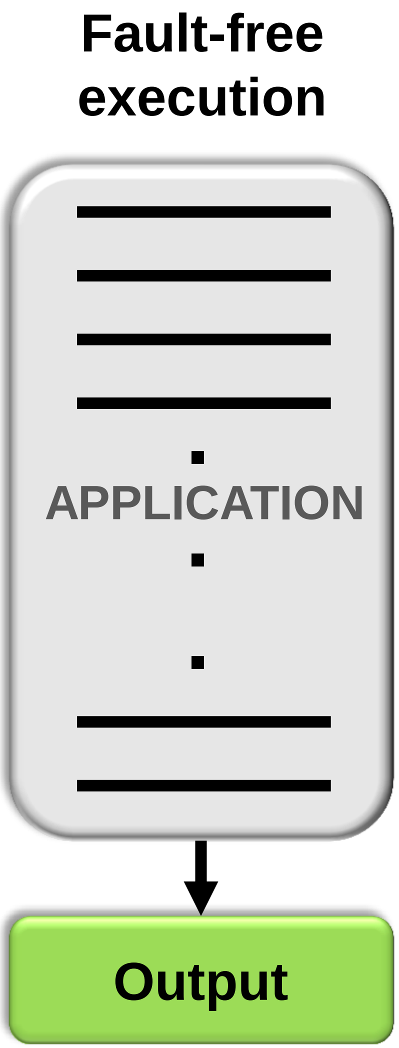


Soft Error

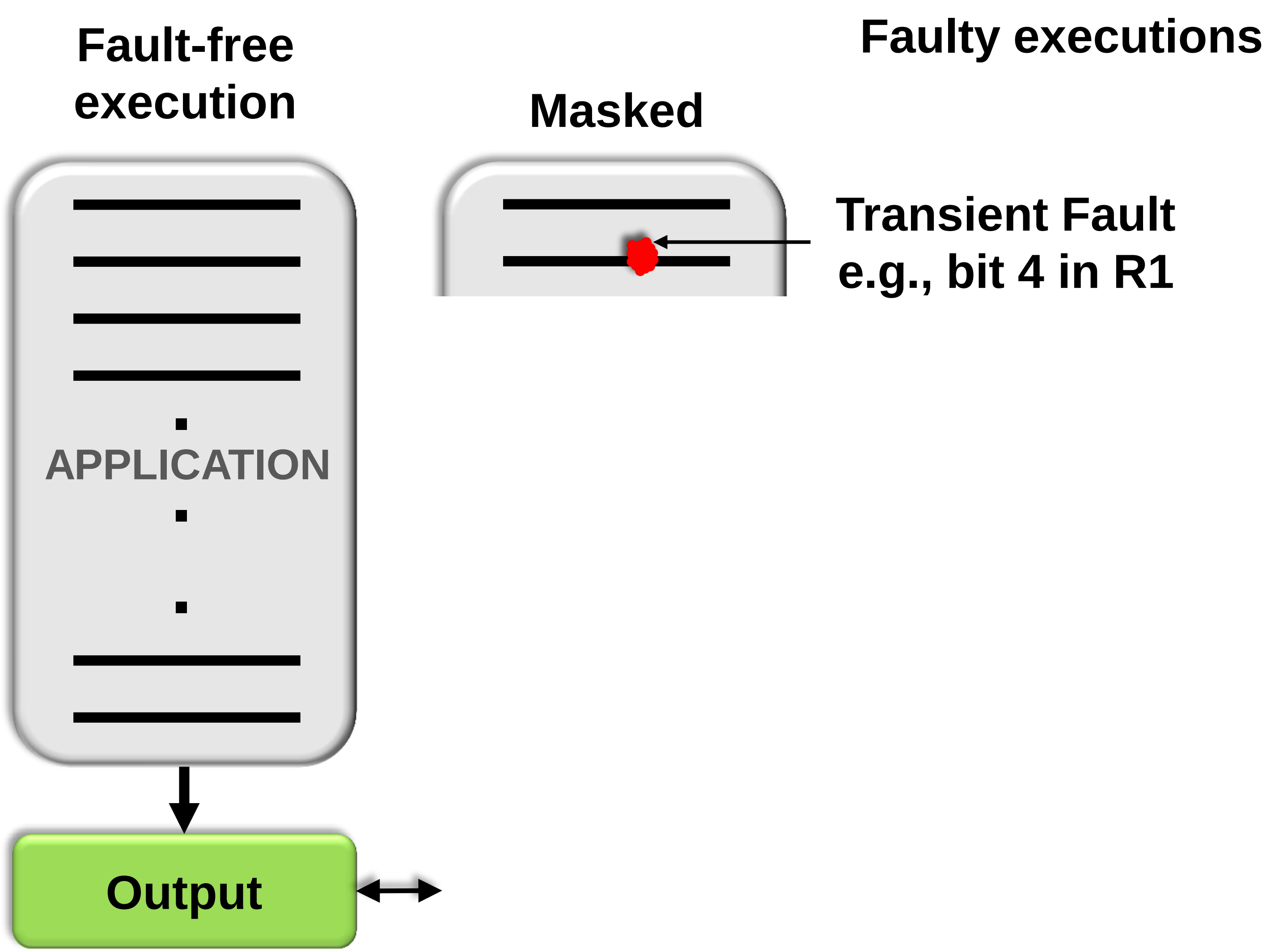
- **Hardware reliability is a major challenge**
  - **Transient (soft) errors are a major problem**
  - **Need in-field low-cost reliability solution**
- **Traditional redundancy based solutions are expensive**
- **Alternative: Treat s/w anomalies as symptoms of h/w faults**
  - **Detect faults using low-cost software symptom monitors**
  - **Diagnosis, recovery more complex, but infrequent**
- **Efficacy depends heavily on application**

How to evaluate application-level resiliency?

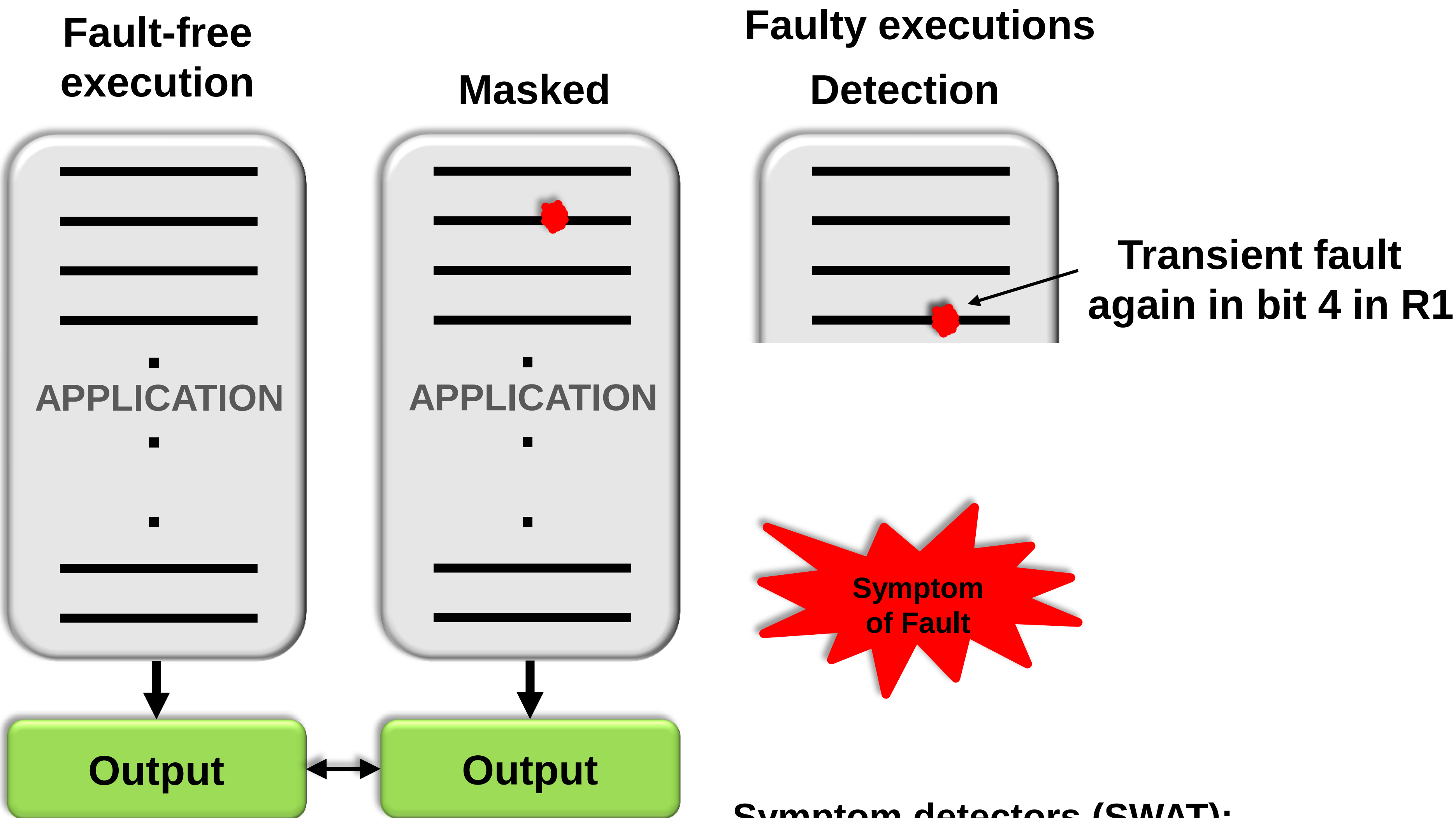
Fault Outcomes



Fault Outcomes

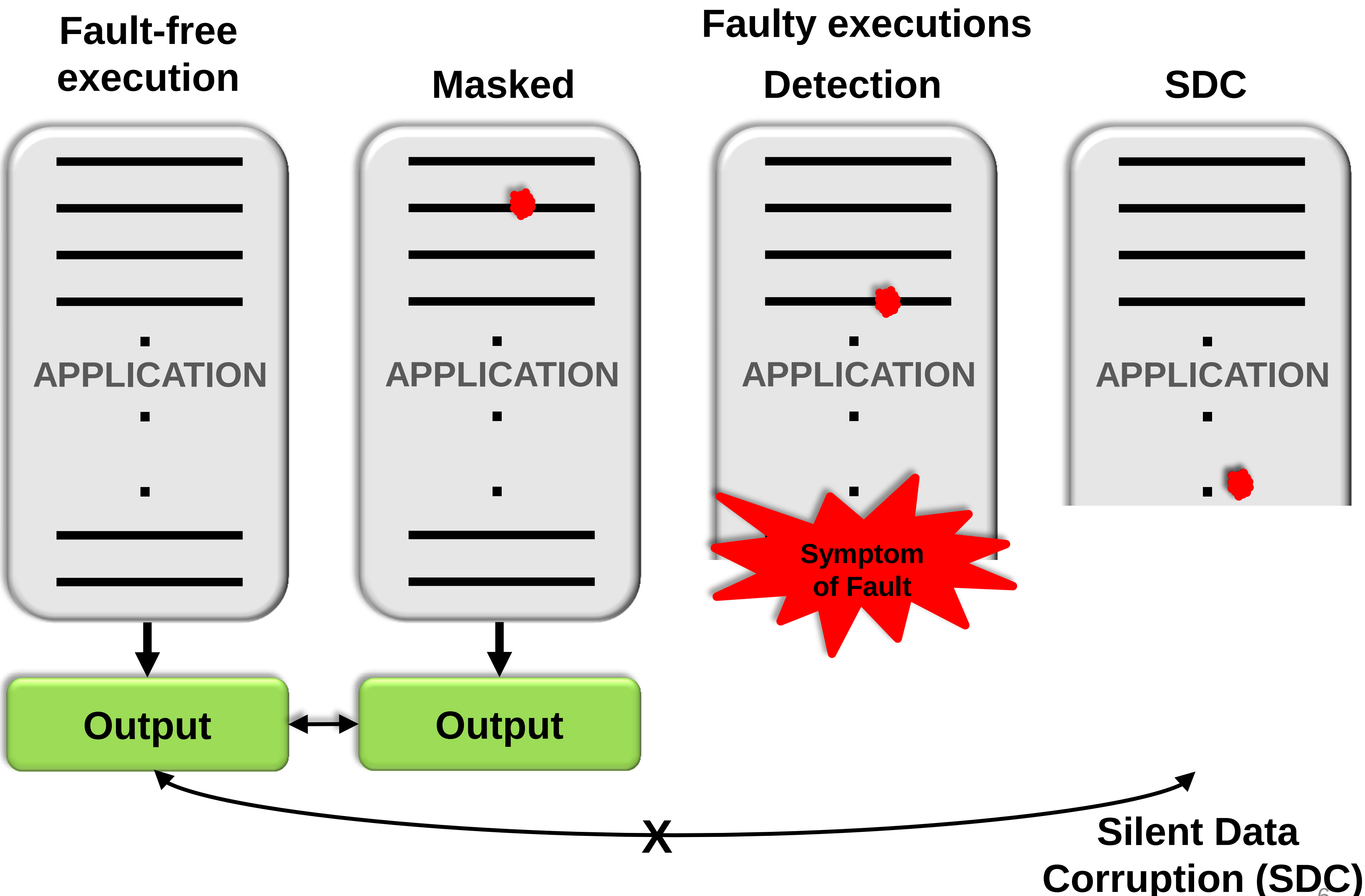


Fault Outcomes

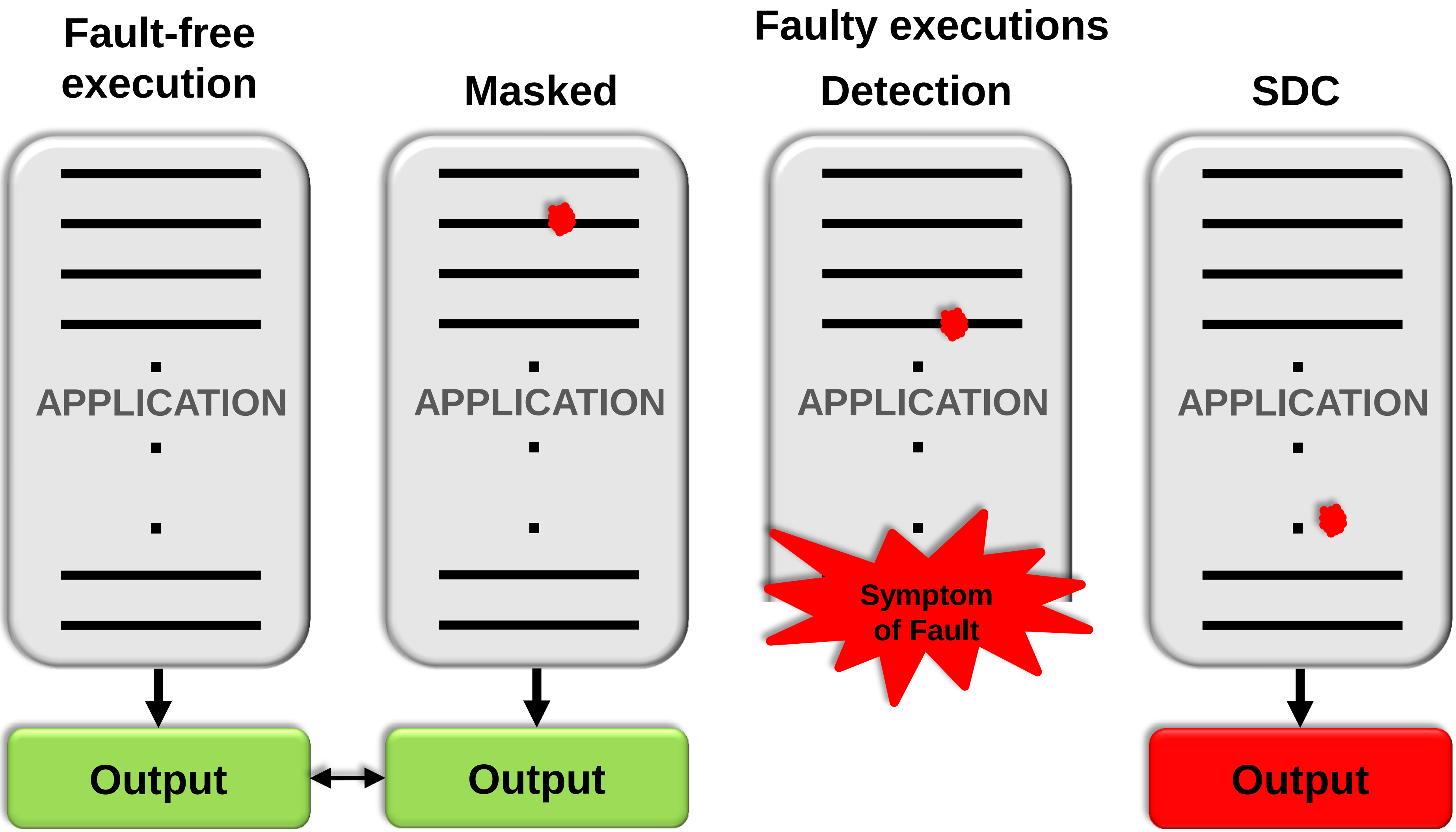


Symptom detectors (SWAT):
Fatal traps, assertion violations, etc.

Fault Outcomes



Fault Outcomes

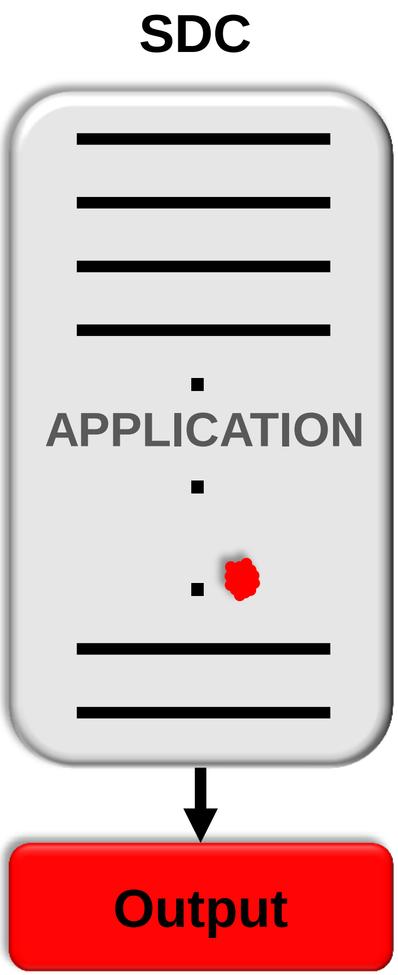


Goal: Lower SDC rate to zero

Silent Data Corruptions

- Symptom detectors are effective, BUT
  - SDC rate is still >0%

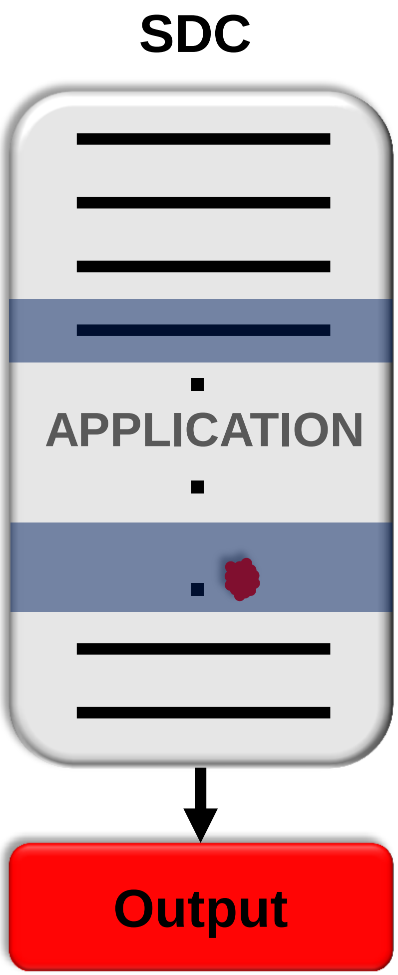
- Two key challenges
  - Which application fault sites cause SDCs?
  - How to convert SDCs to detections?



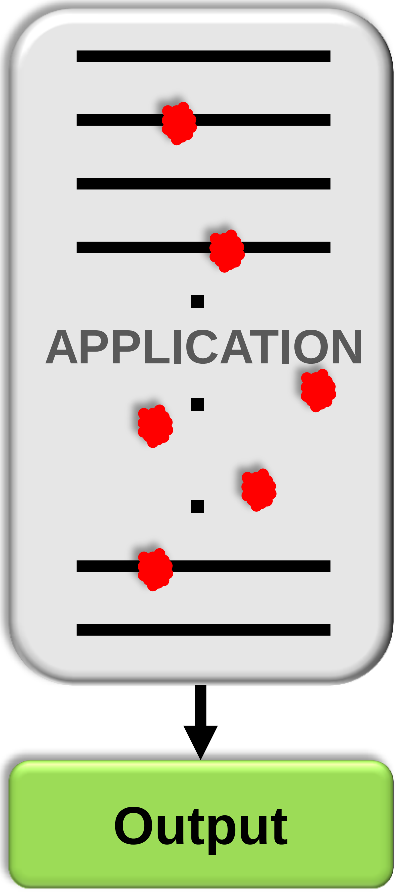
Silent Data Corruptions

- Symptom detectors are effective, BUT
  - SDC rate is still >0%

- Two key challenges
  - Which application fault sites cause SDCs?
    - ⇒ Relyzer lists SDC sites
  - How to convert SDCs to detections?
    - ⇒ Relyzer guides detectors [DSN'12]

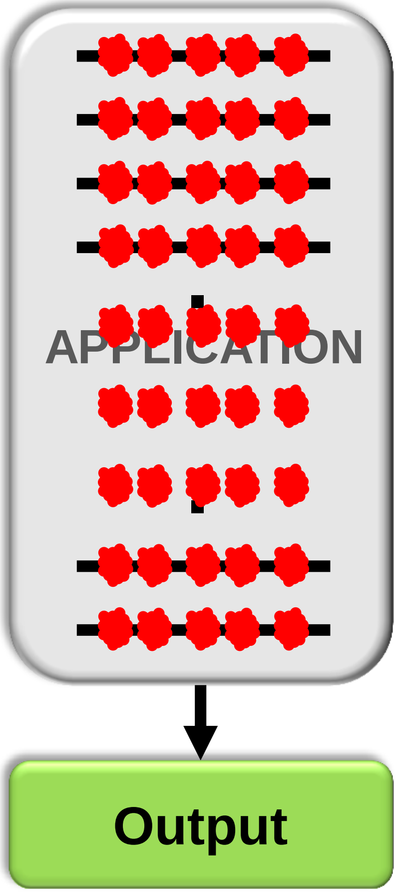


Evaluating Application-Level Resiliency



Statistical Fault Injection
Injections in few sites
Cannot find all SDC sites

Evaluating Application-Level Resiliency

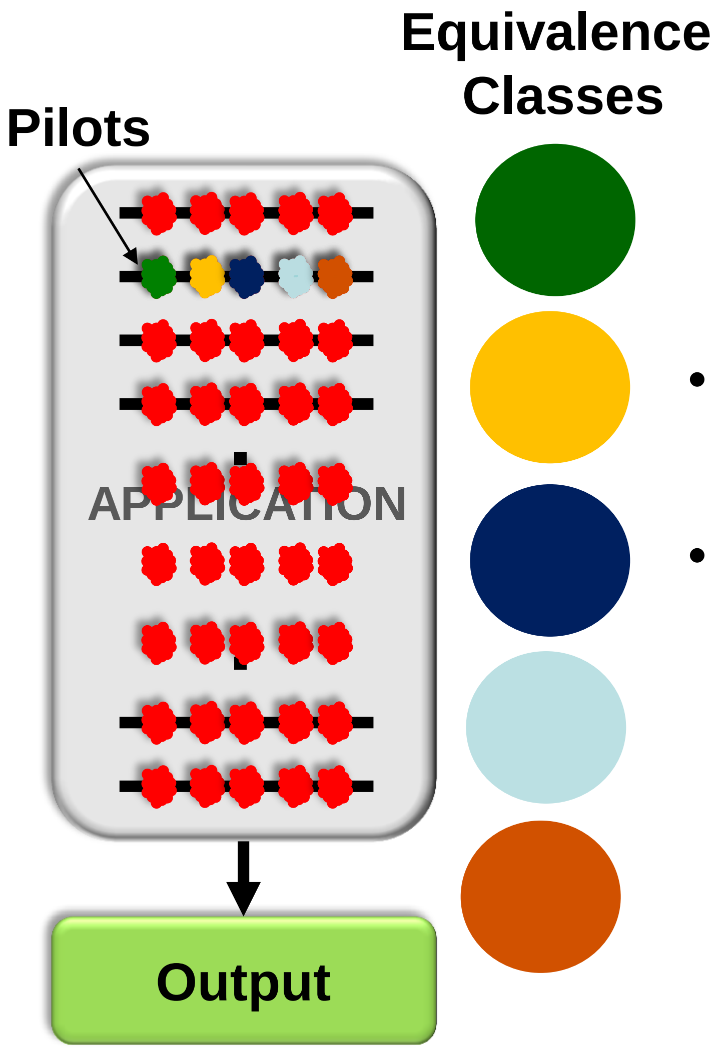


Statistical Fault Injection	Ideal Injection
Injections in few sites	Injections in ALL sites
Cannot find all SDC sites	Find ALL SDC sites

Goal: with

Relyzer: Analyze all app fault sites with few injections

Relyzer Approach



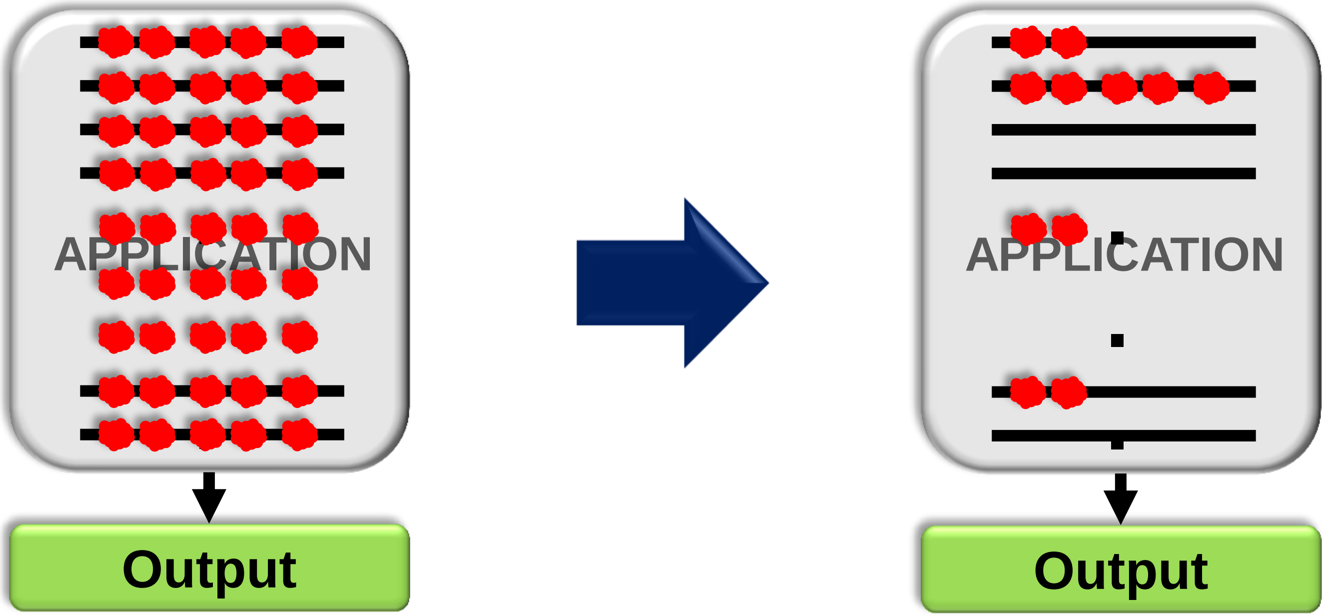
Prune fault sites

- Show application-level fault equivalence
- Predict fault outcomes without injections

Detailed injections for remaining faults

Contributions

- Relyzer: A tool for complete application resiliency analysis
- Developed novel fault pruning techniques
  - 3 to 6 orders of magnitude fewer injections for most apps
  - 99.78% app fault sites pruned
    - Only 0.04% represent 99% of all fault sites



- Can identify all potential SDC causing fault sites

Outline

- Motivation
- Pruning Techniques
- Methodology and Results
- Conclusions and Ongoing Work

Outline

- Motivation
- Pruning Techniques
  - Application-level fault equivalence
  - Predictable faults
- Methodology and Results
- Conclusions and Ongoing Work

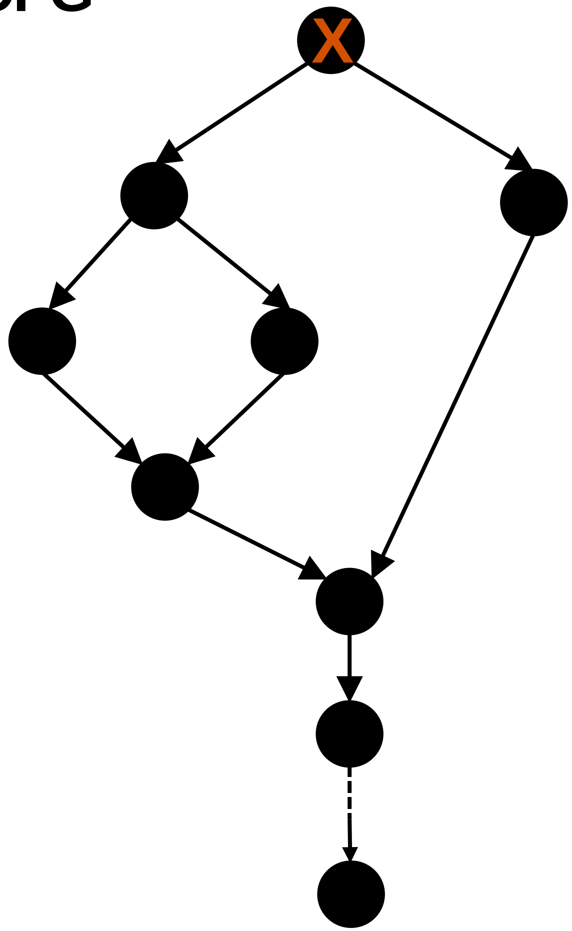
Outline

- Motivation
- Pruning Techniques
  - Application-level fault equivalence
    - Control flow equivalence
    - Store equivalence
    - Definition to first use equivalence
  - Predictable faults
- Methodology and Results
- Conclusions and Ongoing Work

Control Flow Equivalence

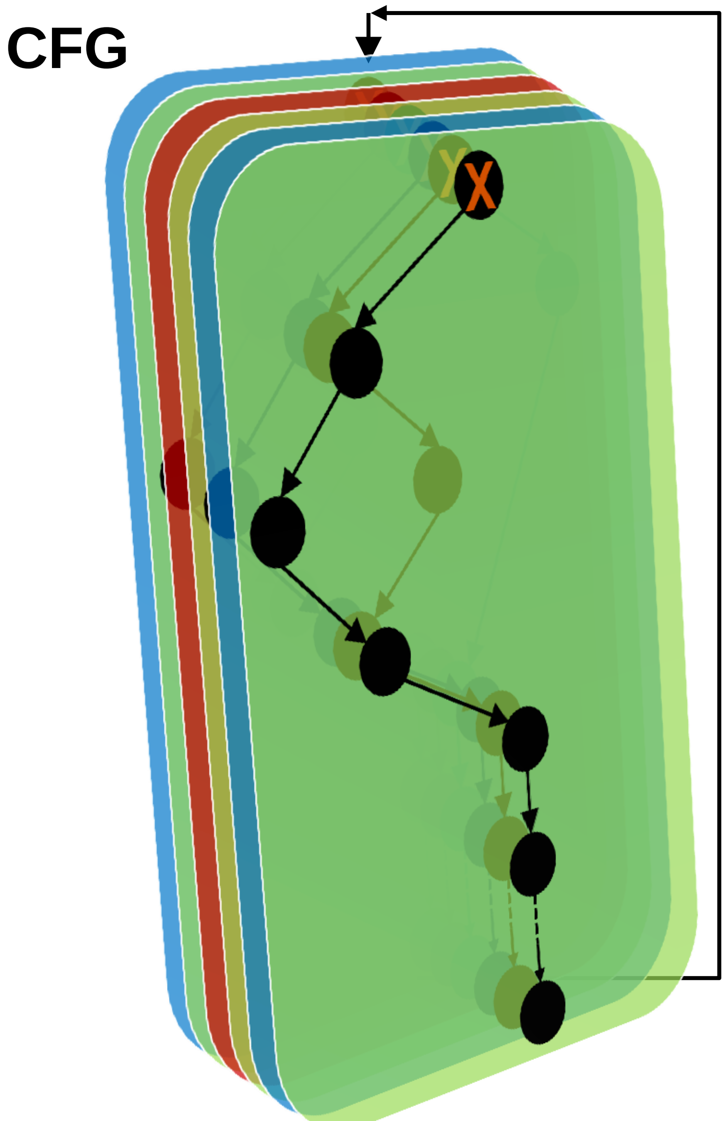
Insight: Faults flowing through similar control paths may behave similarly

CFG



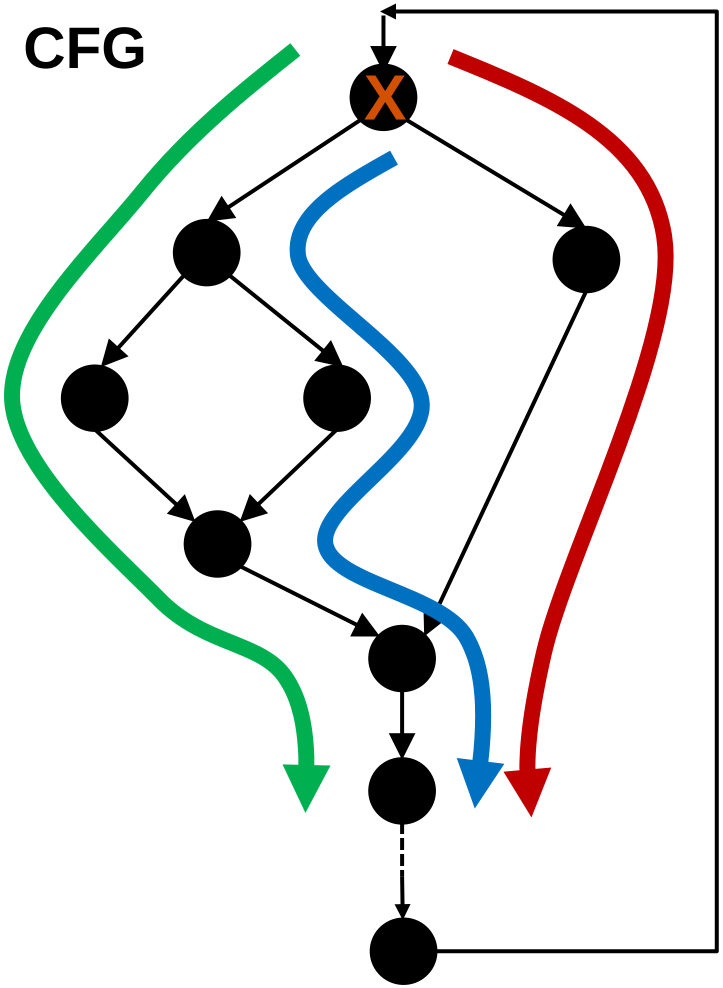
Control Flow Equivalence

Insight: Faults flowing through similar control paths may behave similarly



Control Flow Equivalence

Insight: Faults flowing through similar control paths may behave similarly

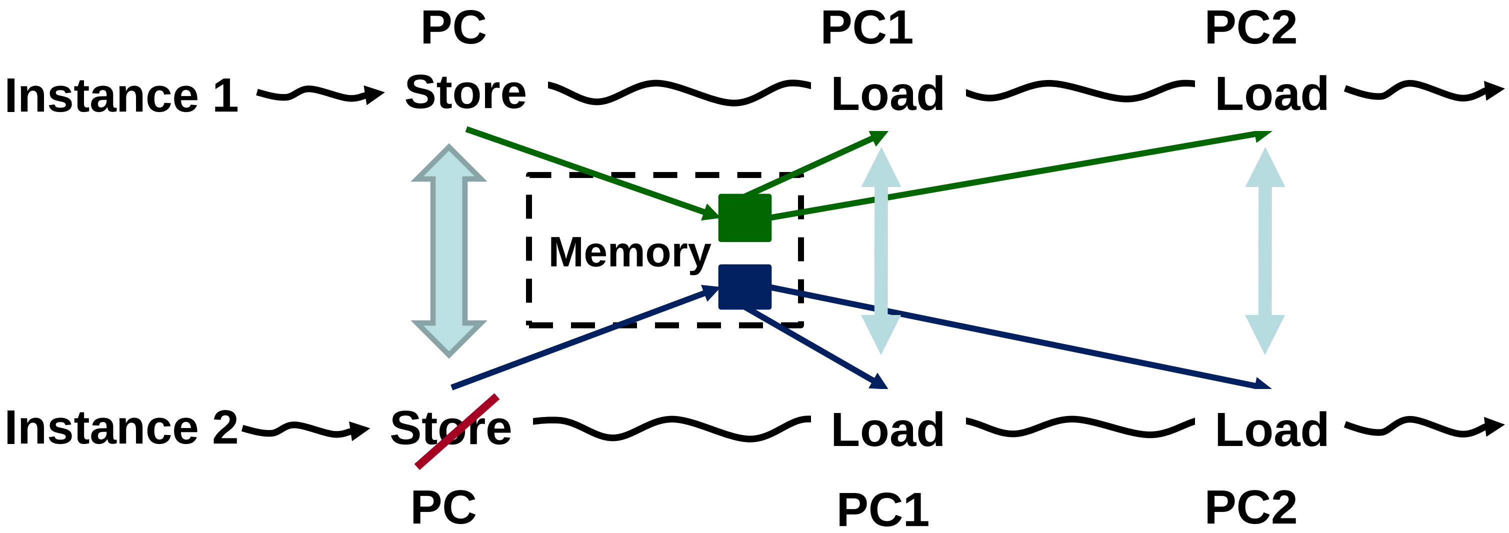


Faults in X that take ▮ paths behave similarly

Heuristic: Use direction of next 5 branches

Store Equivalence

- Insight: Faults in stores may be similar if stored values are used similarly
- Heuristic to determine similar use of values:
  - Same number of loads use the value
  - Loads are from same PCs



Def to First-Use Equivalence

- Fault in first use is equivalent to fault in def $\Rightarrow$ prune def

Def → ~~r1~~ = r2 + r3

r4 = r1 + r5

↑ First use

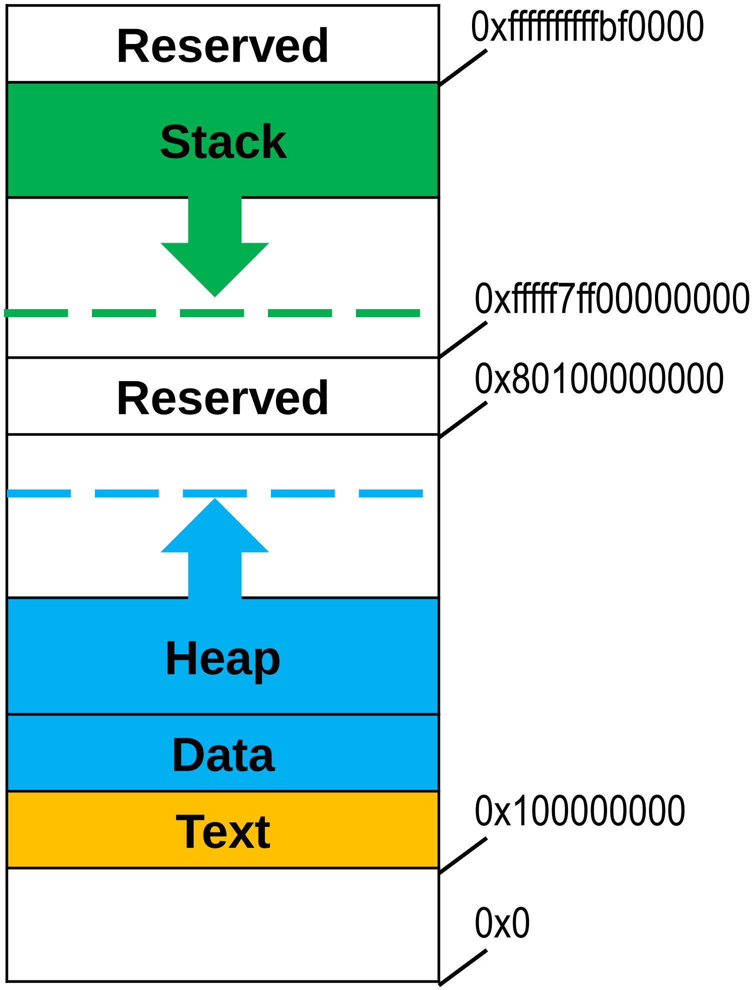
…

- If there is no first use, then def is dead $\Rightarrow$ prune def

Pruning Predictable Faults

- Prune out-of-bounds accesses
  - Detected by symptom detectors
  - Memory addresses not in [blue] & [green]
- Boundaries obtained by profiling

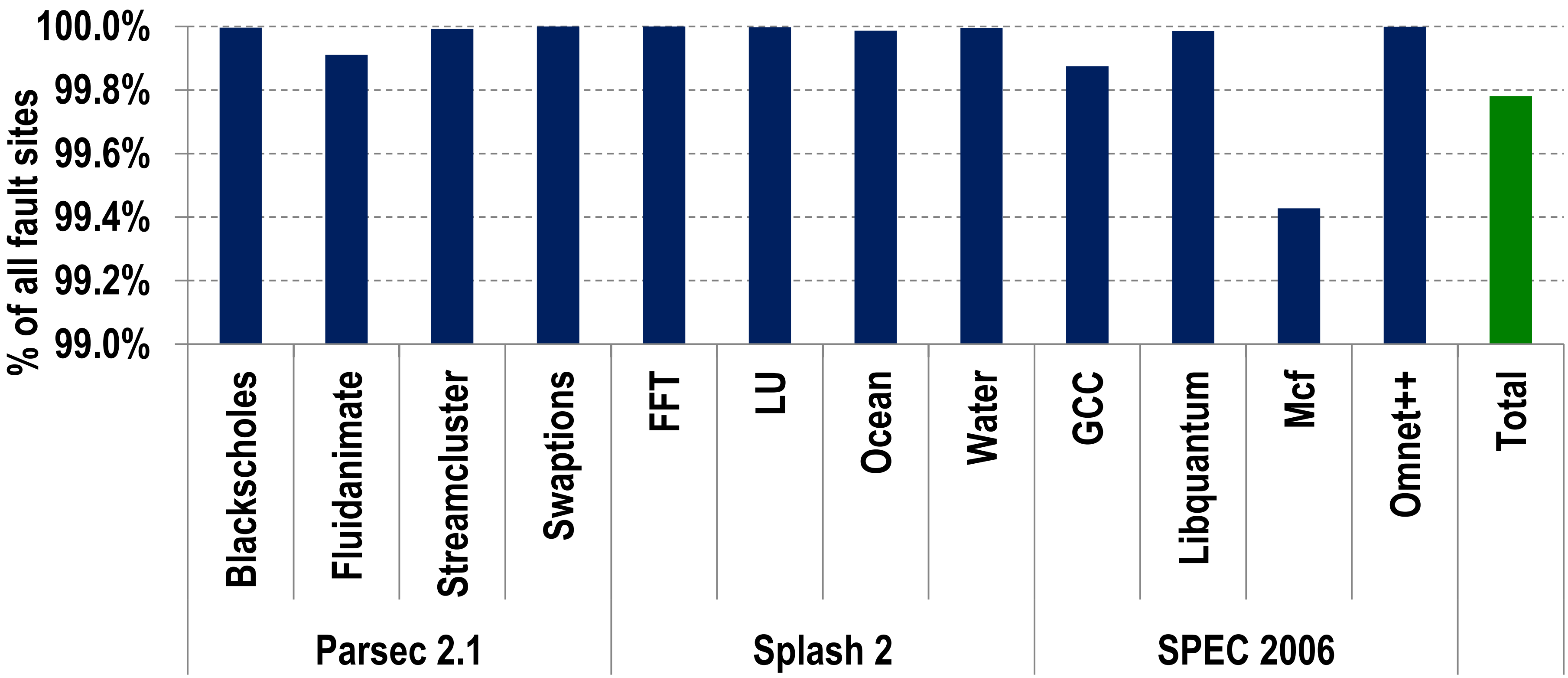
SPARC Address Space Layout



Methodology

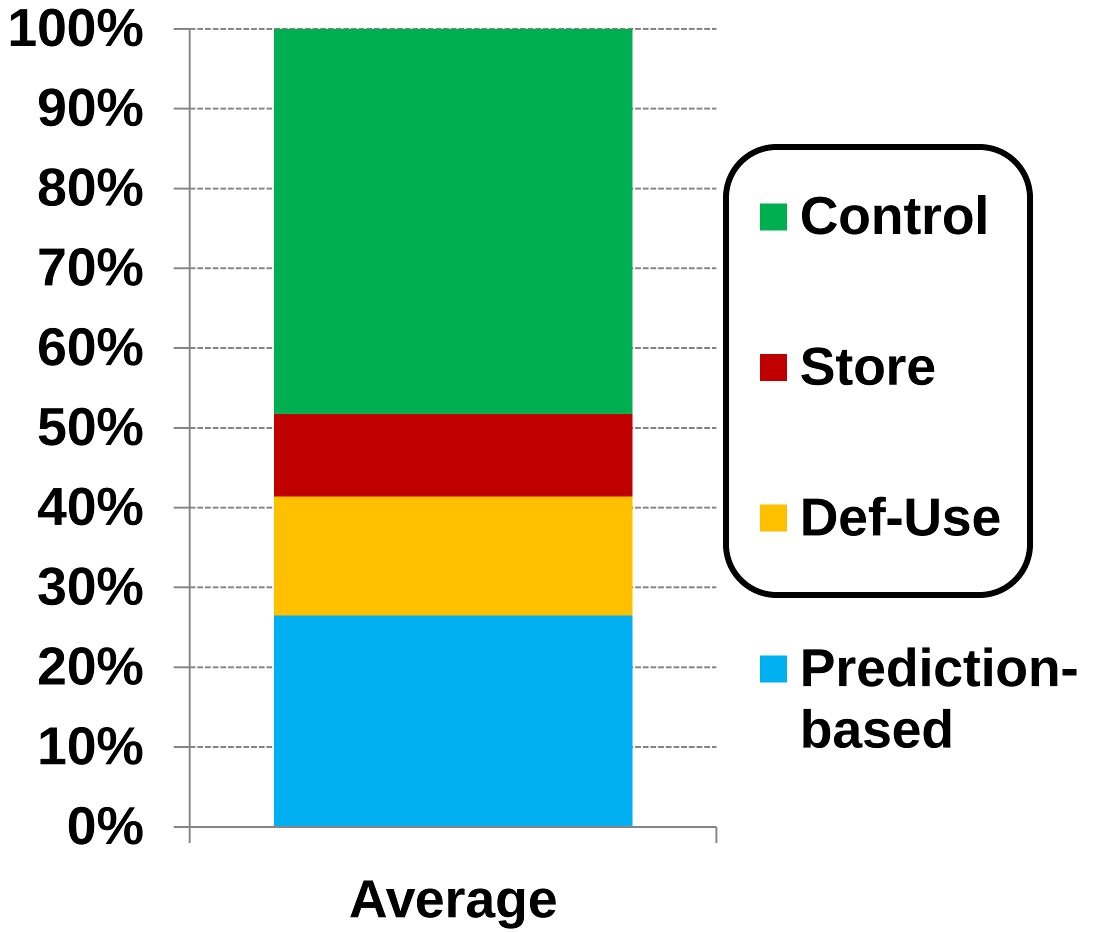
- **Pruning**
  - **12 applications (from SPEC 2006, Parsec, and Splash 2)**
- **Fault model**
  - **Where (hardware) and when (application) to inject transient faults**
  - **Where: Hardware fault sites**
    - **Faults in integer arch registers**
    - **Faults in output latch of address generation unit**
  - **When: Every dynamic instruction that uses these units**

Pruning Results



- **99.78% of fault sites are pruned**
- **3 to 6 orders of magnitude pruning for most applications**
  - **For mcf, two store instructions observed low pruning (of 20%)**
- **Overall 0.004% fault sites represent 99% of total fault sites**

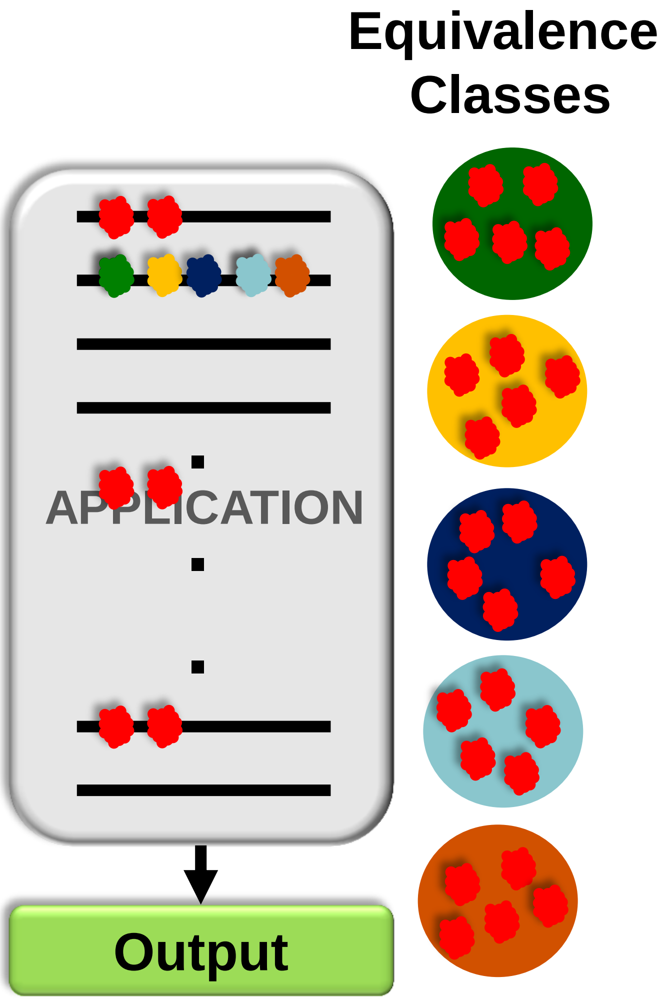
Contribution of Pruning Techniques

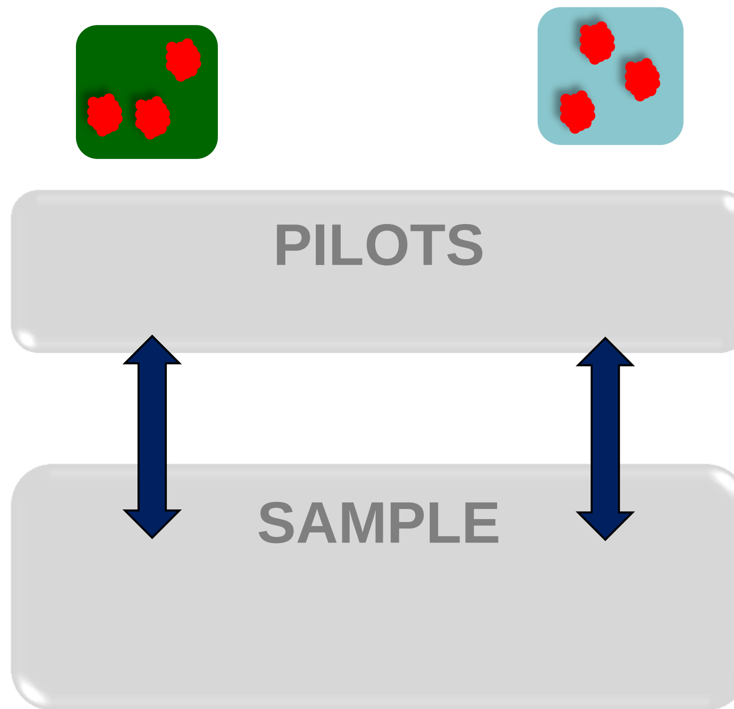


Both equivalence and prediction based techniques are effective

Methodology: Validating Pruning Techniques

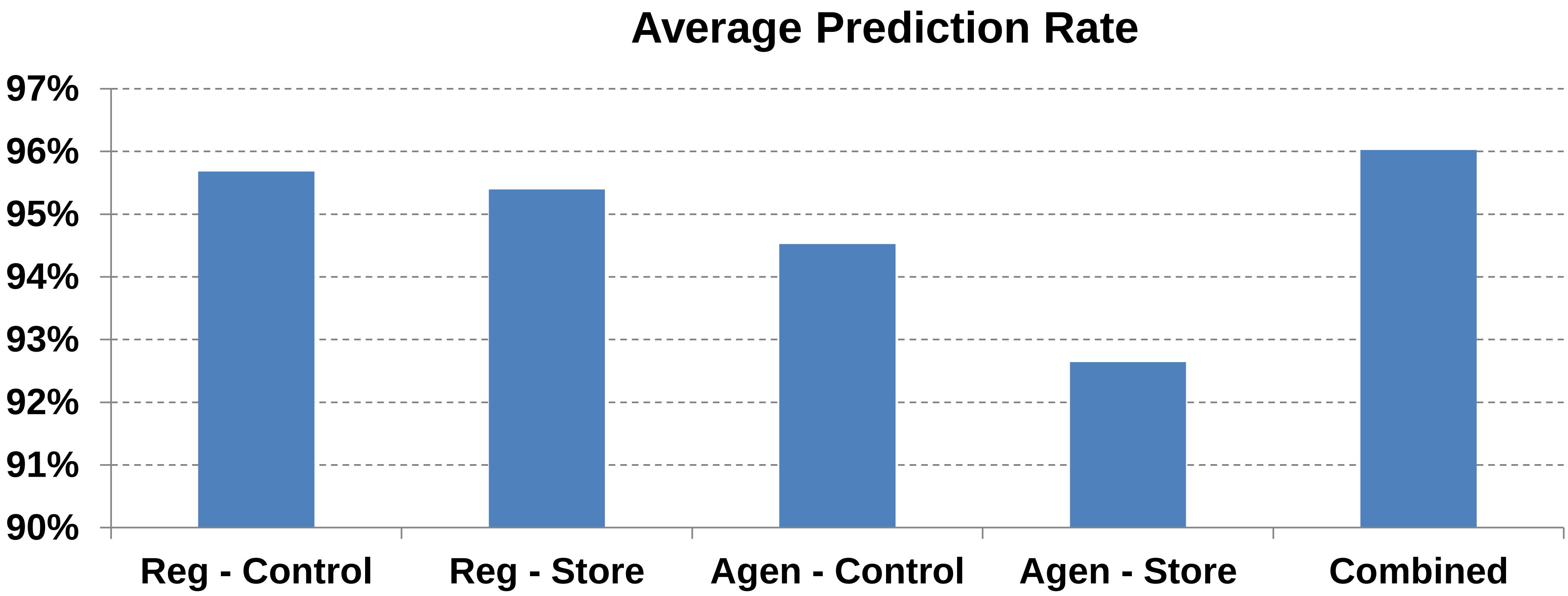
- **Validation for Control and Store equivalence pruning**





Compute Prediction Rate

Validating Pruning Techniques



- Validated control and store equivalence
  - >2M injections for randomly selected pilots, samples from equivalent set
- 96% combined accuracy (including fully accurate prediction-based pruning)
- 99% confidence interval with <5% error

Conclusions and Ongoing Work

- Relyzer: Novel fault pruning for application resiliency analysis
  - 3 to 6 orders of magnitude fewer injections for most apps
    - 99.78% of fault sites pruned
      - Only 0.004% represent 99% of all fault sites
    - Average 96% validation
- Can list all SDC prone instructions and fault propagation path
  - Guides low-cost detectors
  - Ongoing work (to appear in DSN'12)
    - Understand application properties responsible for SDCs
    - Devise (automate) low-cost app-level detectors
    - Quantifiable resilience vs. performance

Relyzer: Exploiting Application-level Fault Equivalence to Analyze Application Resiliency to Transient Faults

Siva Hari[1], Sarita Adve[1], Helia Naeimi[2], Pradeep Ramachandran[2]

[1] University of Illinois at Urbana-Champaign,

[2] Intel Corporation

swat@cs.illinois.edu