Low-cost Program-level Detectors for Reducing Silent Data Corruptions

Siva Hari[†], Sarita Adve[†], and Helia Naeimi[‡]

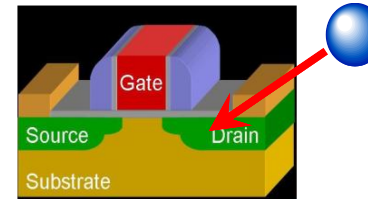
[†]University of Illinois at Urbana-Champaign,

[‡]Intel Corporation

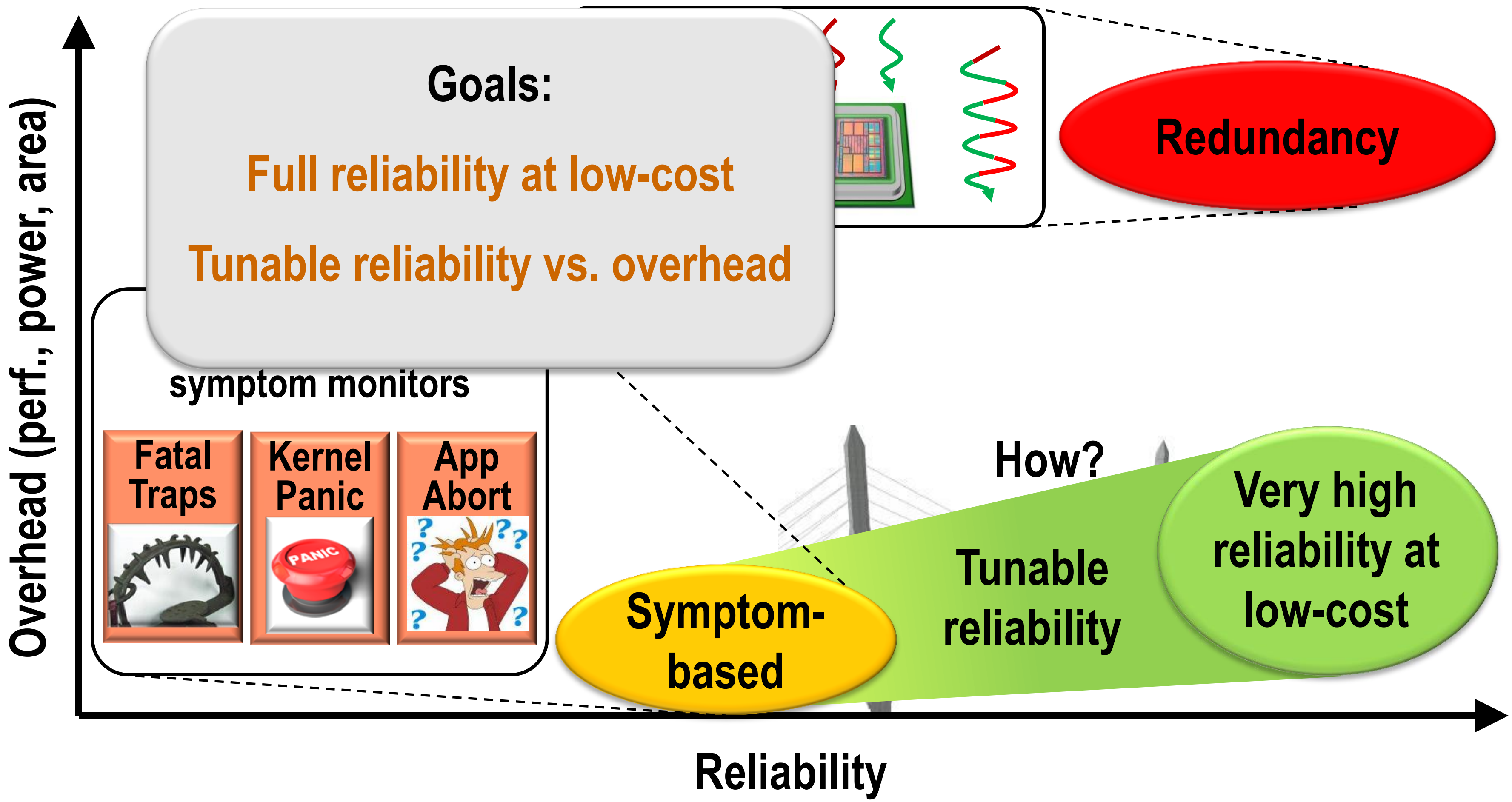
swat@cs.illinois.edu

Motivation

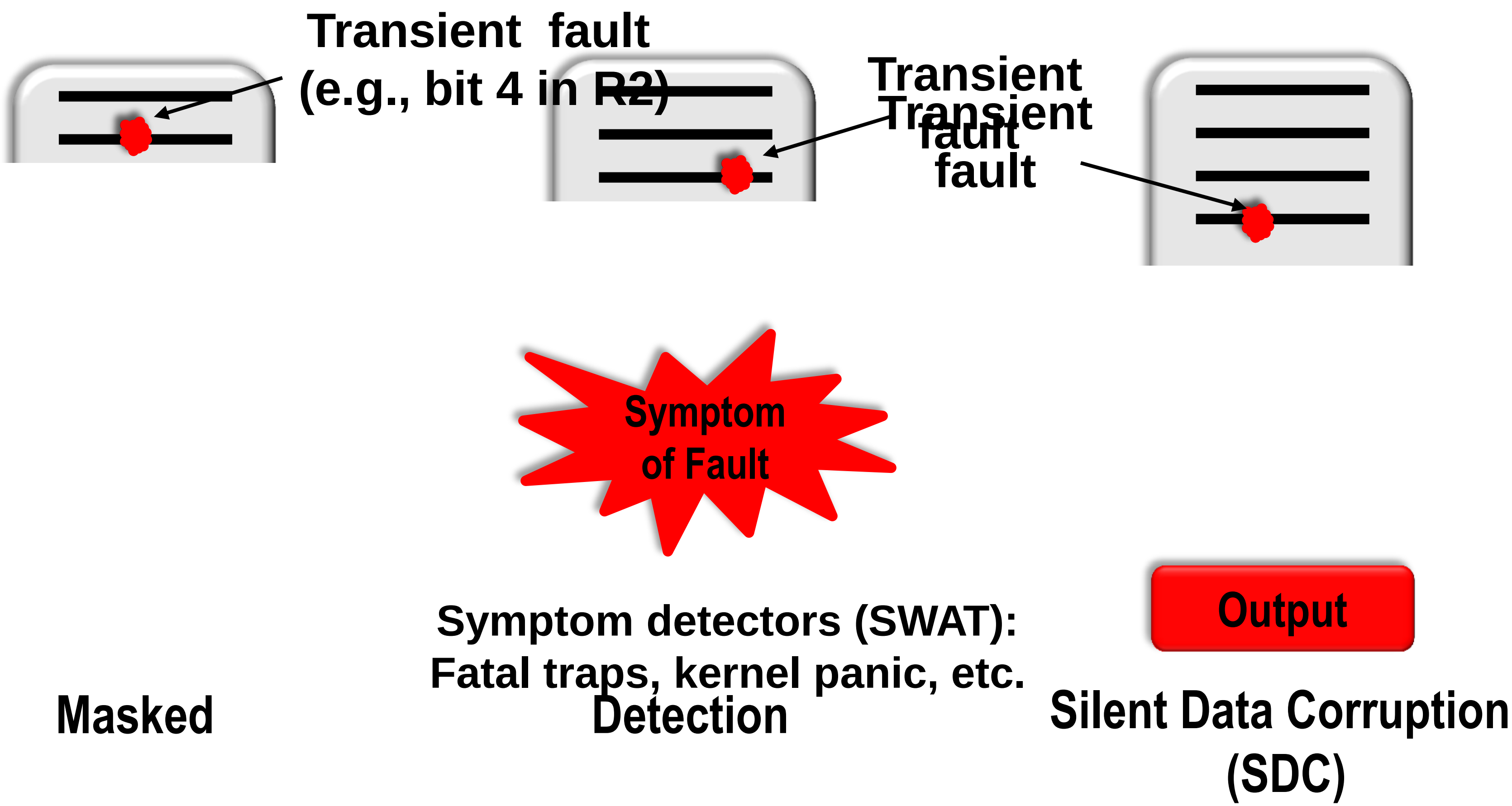
- Hardware reliability is a challenge
  - Transient (soft) errors are a major problem



Soft Error

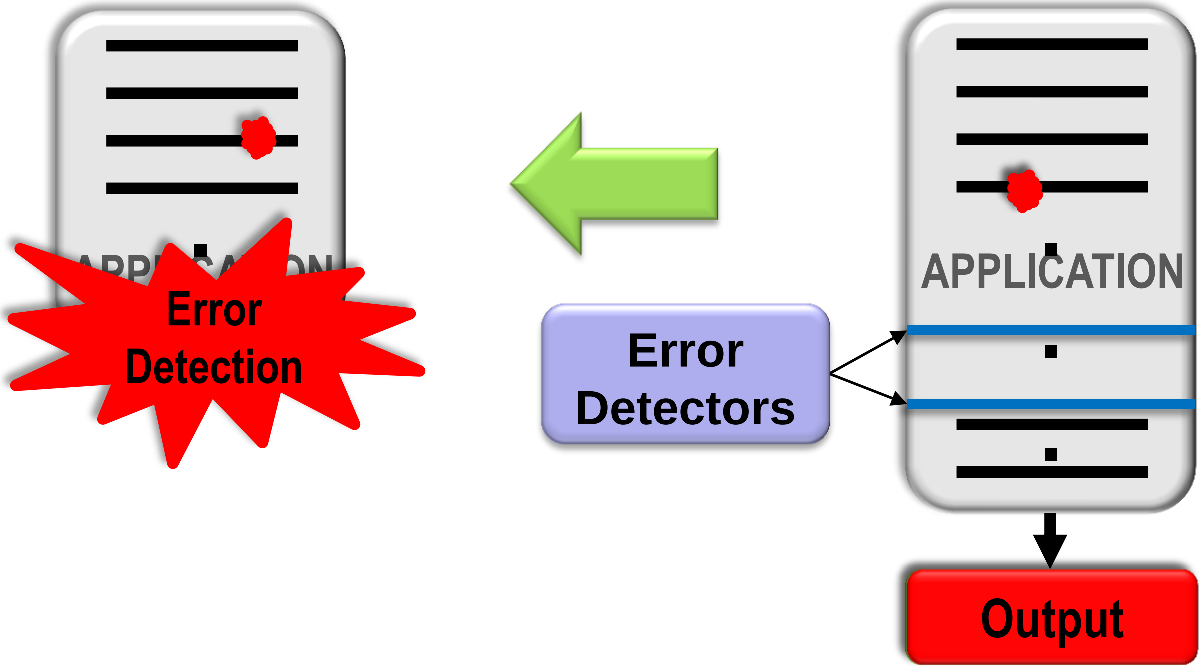


Fault Outcomes



How to convert SDCs to detections?

SDCs to Detections



Silent Data Corruption (SDC)

- Add new detectors in error propagation path?
  - SDC coverage: Fraction of all SDCs converted to detections
- Will it be low-cost?

Key Challenges

What to protect?	SDC-causing fault sites Identified using Relyzer [ASPLOS'12]	
How to Protect?	Low-cost Detectors	
	Where to place?	Many errors propagate to few program values
	What detectors?	Program-level properties tests
	Uncovered fault-sites?	Selective instruction-level duplication

Contributions

- Discovered common program properties around most SDC-causing sites
- Devised low-cost program-level detectors
  - Average SDC reduction of 84%
  - Average execution overhead 10%
- New detectors + selective duplication = Tunable resiliency at low-cost
  - Found near optimal detectors for any SDC target
  - Lower cost than pure redundancy for all SDC targets
    - E.g., 12% vs. 30% @ 90% SDC reduction

Outline

- Motivation and introduction
- Categorizing and protecting SDC-causing sites
- Tunable resilience vs. overhead
- Methodology
- Results
- Conclusions

Outline

- Motivation and introduction
- Categorizing and protecting SDC-causing sites
  - Loop incrementalization
  - Registers with long life
  - Application-specific behavior
- Tunable resilience vs. overhead
- Methodology
- Results
- Conclusions

Insights

- Identify *where* to place the detectors and *what* detectors to use
- Placement of detectors (*where*)
  - Many errors propagate to few program values
    - End of loops and function calls
- Detectors (*what*)
  - Test program-level properties
    - E.g., comparing similar computations and checking value equality
- Fault model
  - Single bit flips in integer arch. registers

Loop Incrementalization

C Code

```
Array a, b;
For (i=0 to n) {
   . . .
   a[i] = b[i] + a[i]
   . . .
}
```

ASM Code

```
rA = base addr. of a
rB = base addr. of b

L:  load   r1 ← [rA]
       . . .
       load   r2 ← [rB]
       . . .
       store  r3 → [rA]
       . . .
       add      rA = rA + 0x8
       add      rB = rB + 0x8
       add      i = i + 1
       branch (i<n) L
```

Loop Incrementalization

C Code

```
Array a, b;
For (i=0 to n) {
  . . .
  a[i] = b[i] + a[i]
  . . .
}
```

ASM Code

```
rA = base addr. of a
rB = base addr. of b
L:  load   r1 ← [rA]
       . . .
       load   r2 ← [rB]
       . . .
       store  r3 → [rA]
       . . .
       add     rA = rA + 0x8
       add     rB = rB + 0x8
       add     i = i + 1
       branch (i<n) L
```

Collect initial values of rA, rB, and i

SDC-hot app sites

Where: Errors from *all* iterations propagate here in few quantities

What: Property checks on rA, rB, and i

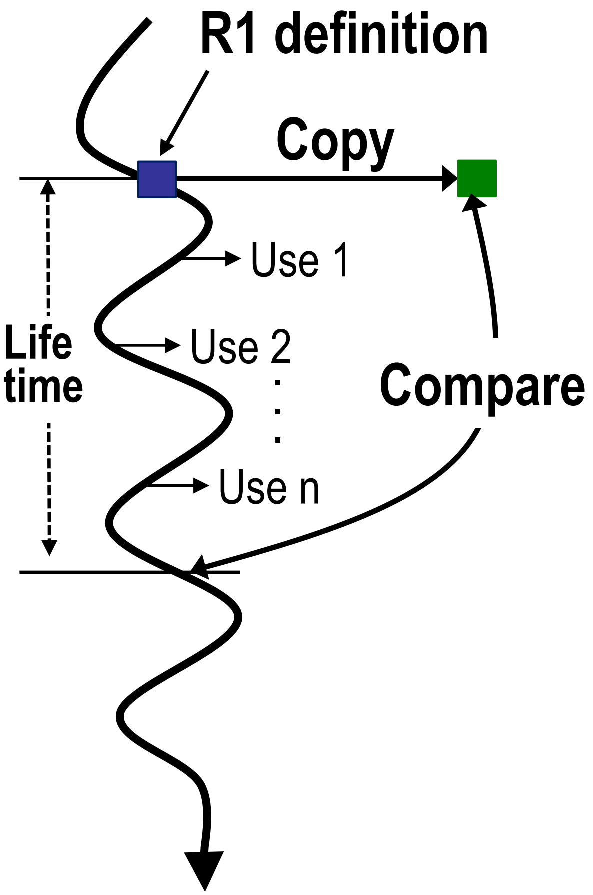
Diff in rA = Diff in rB

Diff in rA = 8 × Diff in i

No loss in coverage - *lossless*

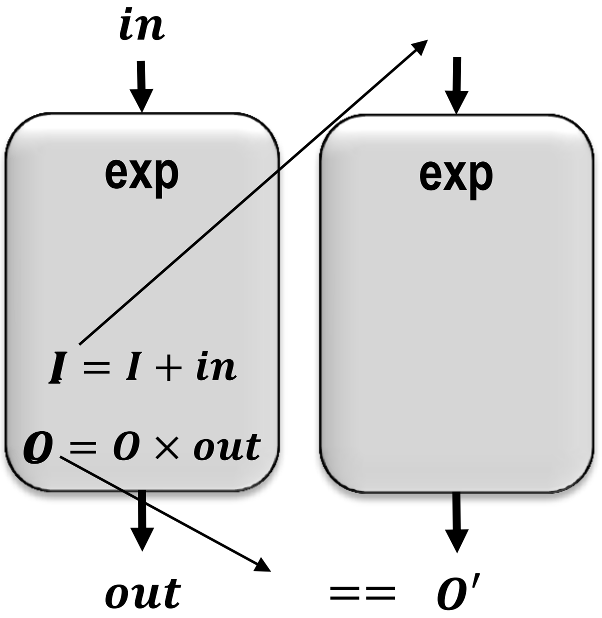
Registers with Long Life

- Some long lived registers are prone to SDCs
- For detection
  - Duplicate the register value at its definition
  - Compare its value at the end of its life
- No loss in coverage - *lossless*



Application-Specific Behavior

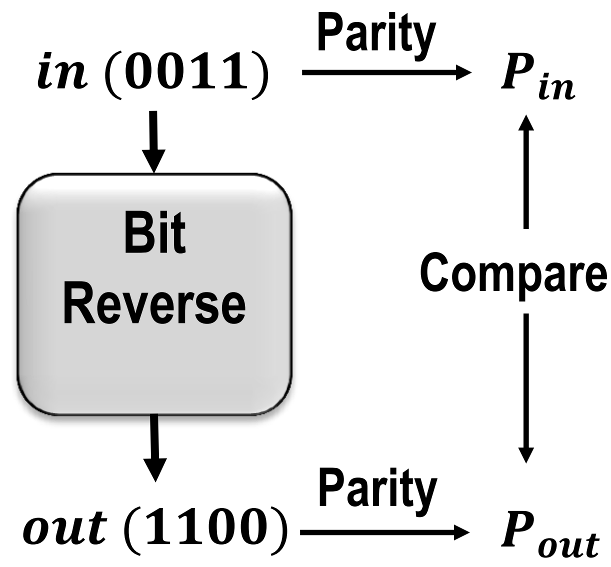
- Exponential function
  - Where: End of ~~every~~ few function invocations
  - What: Re-execution or inverse function (*log*)
  - Periodic test on accumulated quantities
  - Accumulate input and output with $+$ and $\times$
    - $e^{(i1+i2)} = e^{i1} \times e^{i2}$



- Some coverage may be compromised – *lossy*

Application-Specific Behavior (Contd.)

- Bit Reverse function
  - Where: End of function
  - What: Challenge – re-execution?
  - Approach: Parity of *in* & *out* should match



- Other detectors: Range checks
  - $Value \leq Upper\ bound$
  - $Lower\ bound \leq Value \leq Upper\ bound$

- Some coverage may be compromised – *lossy*

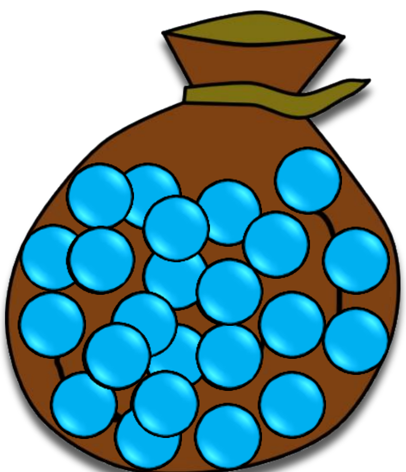
Tunable Resiliency vs. Overhead

- **What if our detectors do not cover all SDC-causing sites?**
  - **Use selective instruction-level redundancy**

- **What if our low-overhead is still not tolerable but lower resiliency is?**
  - **Tunable resiliency vs. overhead**

Identifying Near Optimal Detectors: Naïve Approach

Example: Target SDC coverage = 60%



Bag of detectors

Overhead = 10%

Sample 1

SFI

SDC coverage

50%

Overhead = 20%

Sample 2

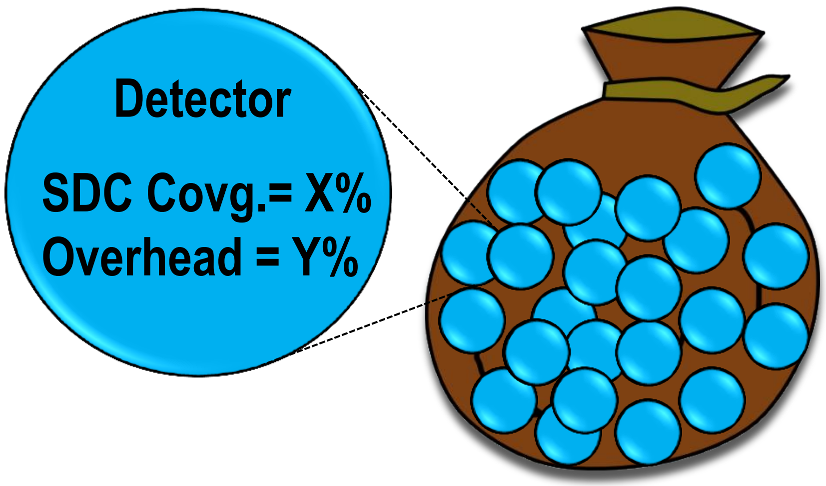
SFI

65%

Tedious and time consuming

Identifying Near Optimal Detectors: Our Approach

1. Set attributes, enabled by Relyzer [ASPLOS’12]



Bag of detectors

2. Dynamic programming

Constraint: Total SDC covg. ≥ 60%

Objective: Minimize overhead

Selected Detectors

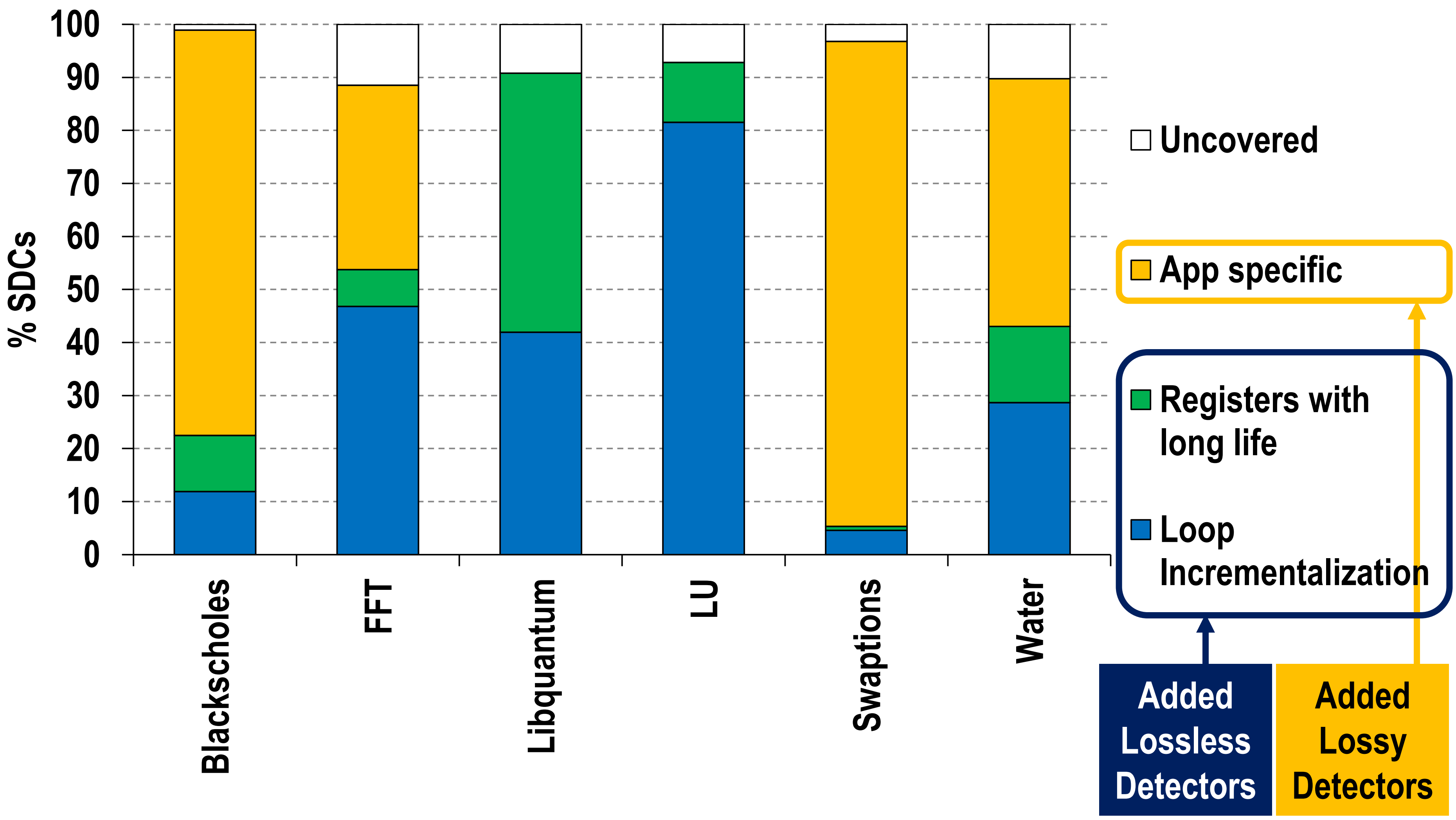
Overhead = 9%

Obtained SDC coverage vs. Performance trade-off curves

Methodology

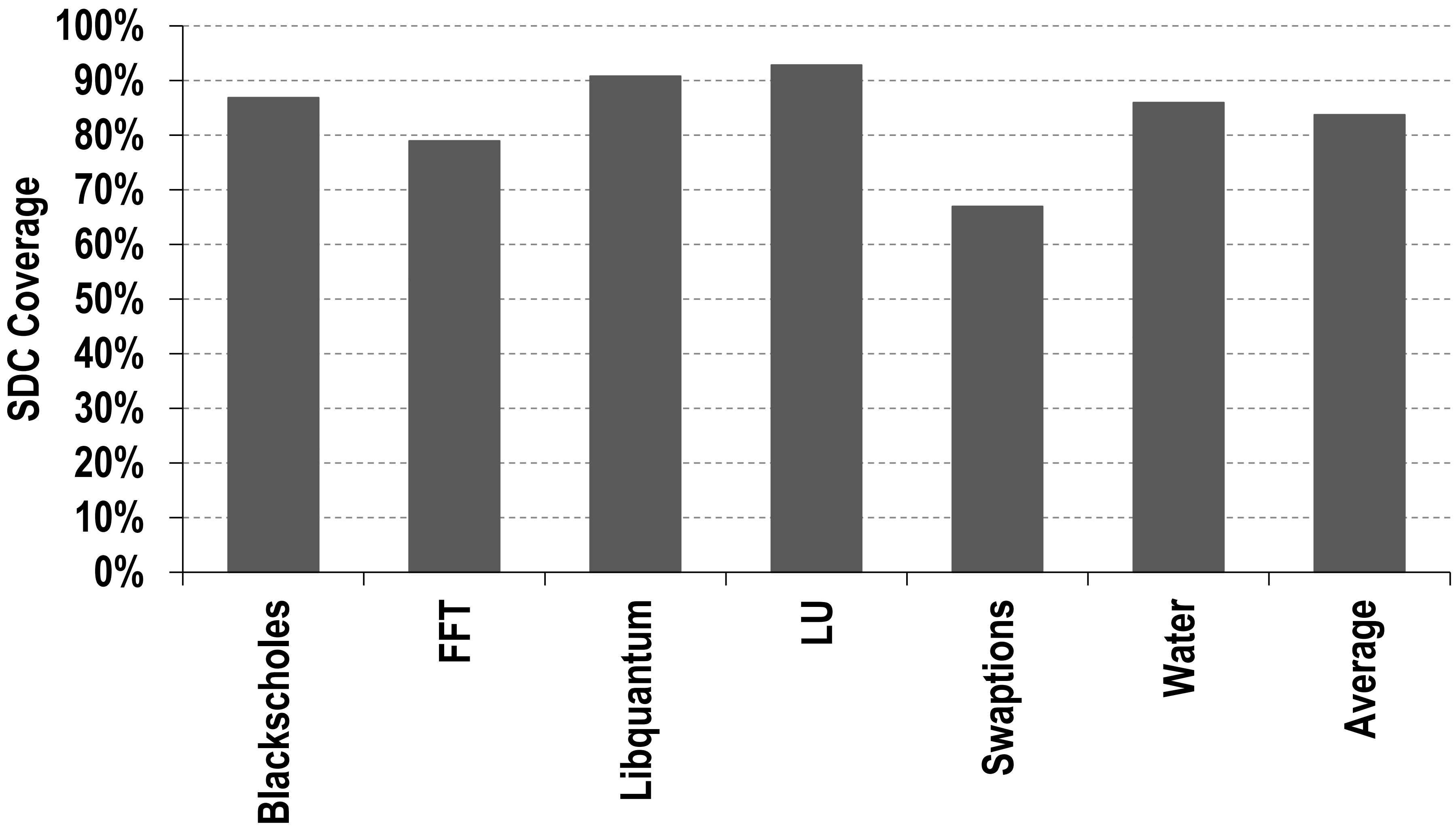
- Six applications from SPEC 2006, Parsec, and SPLASH2
- Fault model: single bit flips in int arch registers at every dynamic instr
- Ran Relyzer, obtained SDC-causing sites, examined them manually
- Our detectors
  - Implemented in architecture simulator
  - Overhead estimation: Num assembly instrns needed
- Selective redundancy
  - Overhead estimation: 1 extra instrn for every uncovered instrn
- Lossy detectors' coverage
  - Statistical fault injections (10,000)

Categorization of SDC-causing Sites



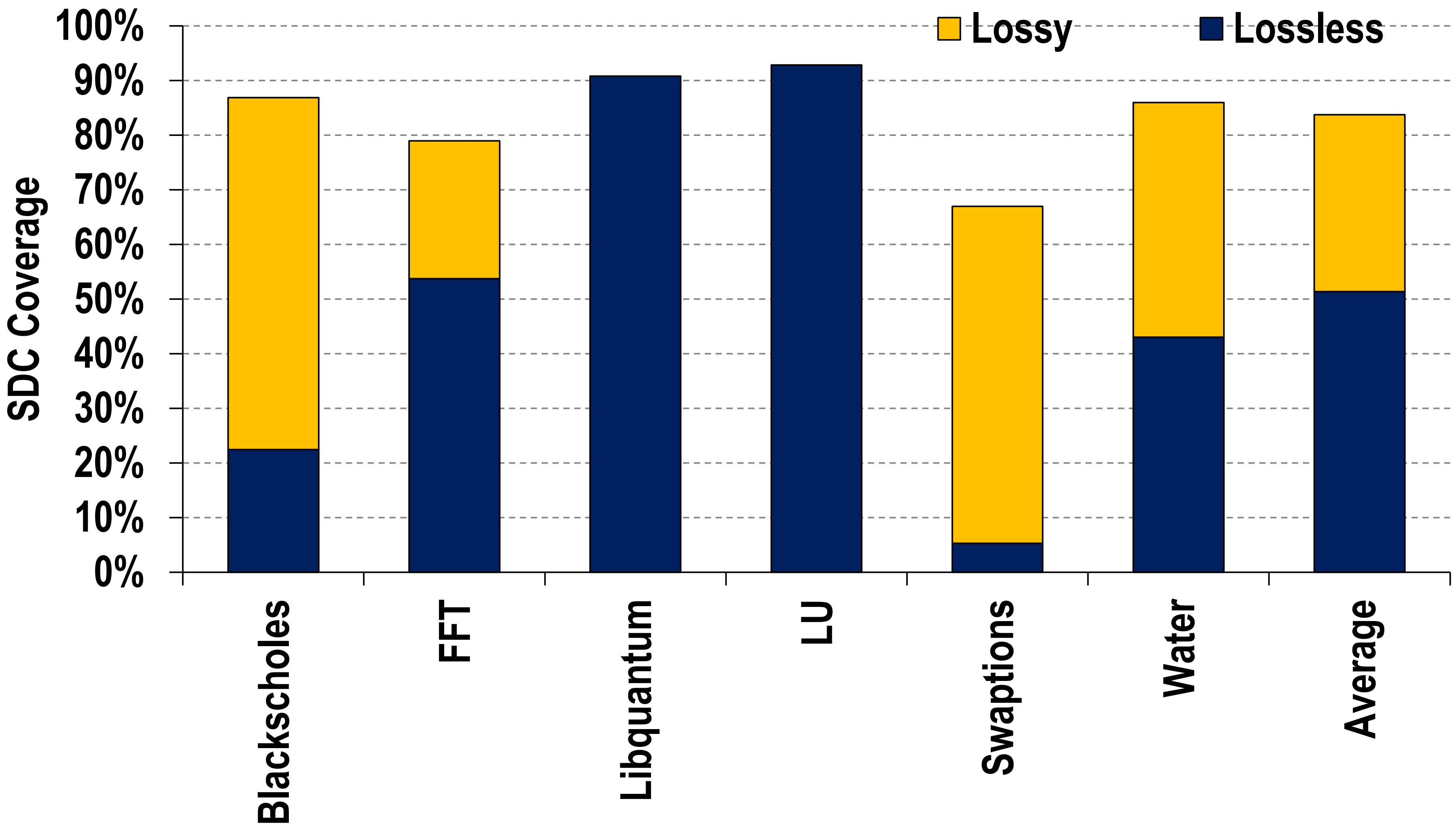
- Categorized >88% SDC-causing sites

SDC coverage



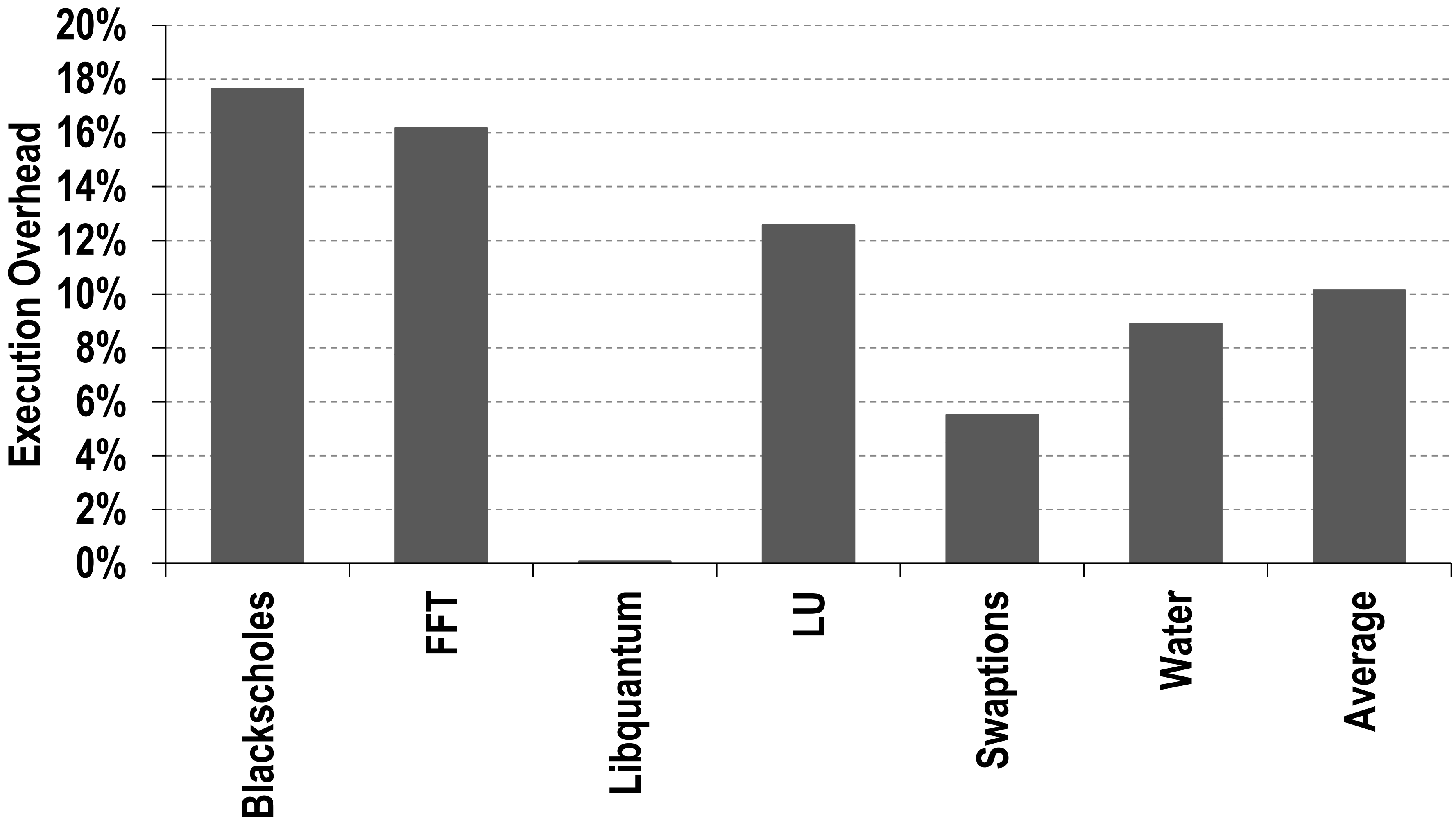
- 84% average SDC coverage (67% - 92%)

SDC coverage



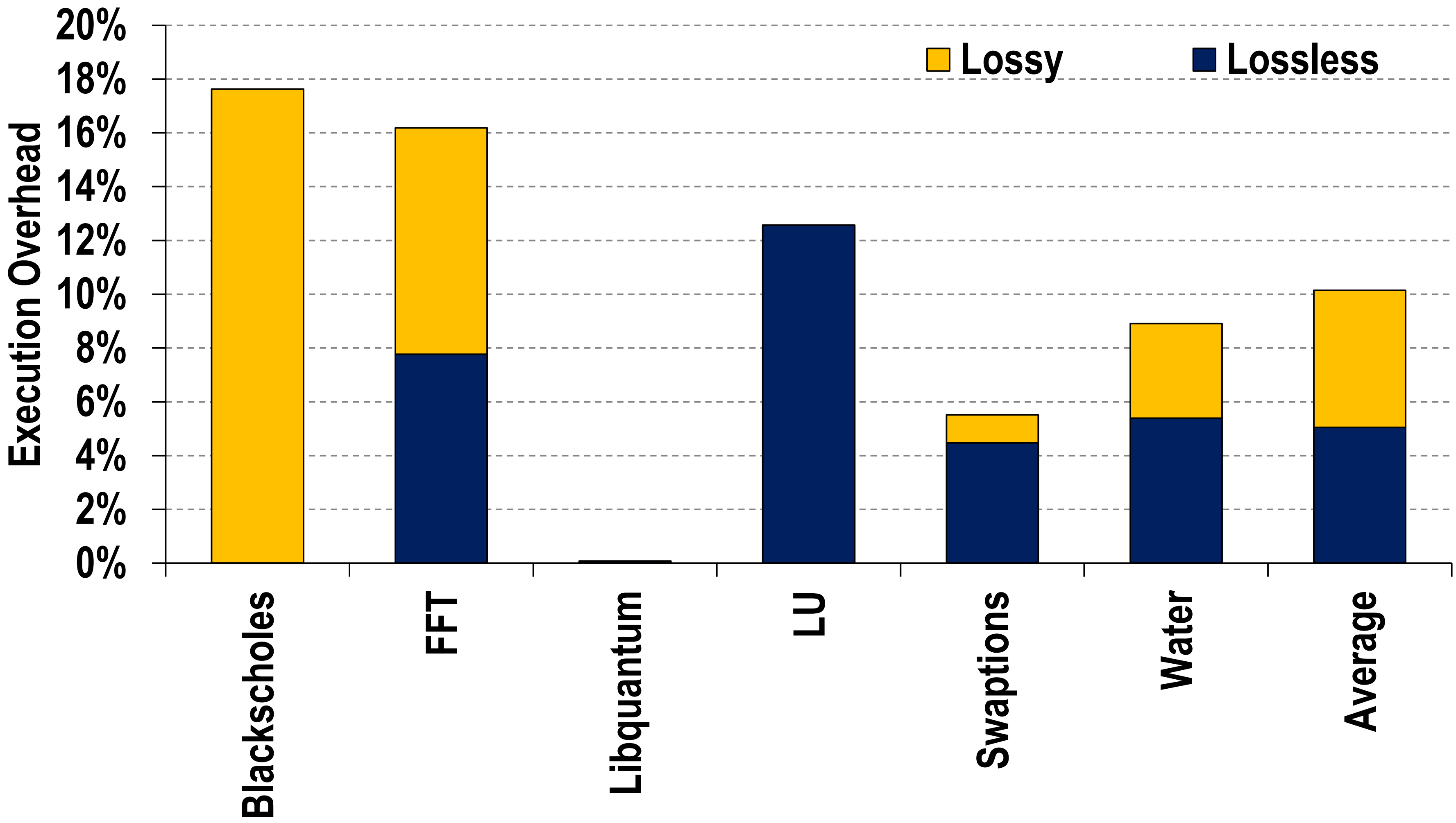
- 84% average SDC coverage (67% - 92%)

Execution Overhead



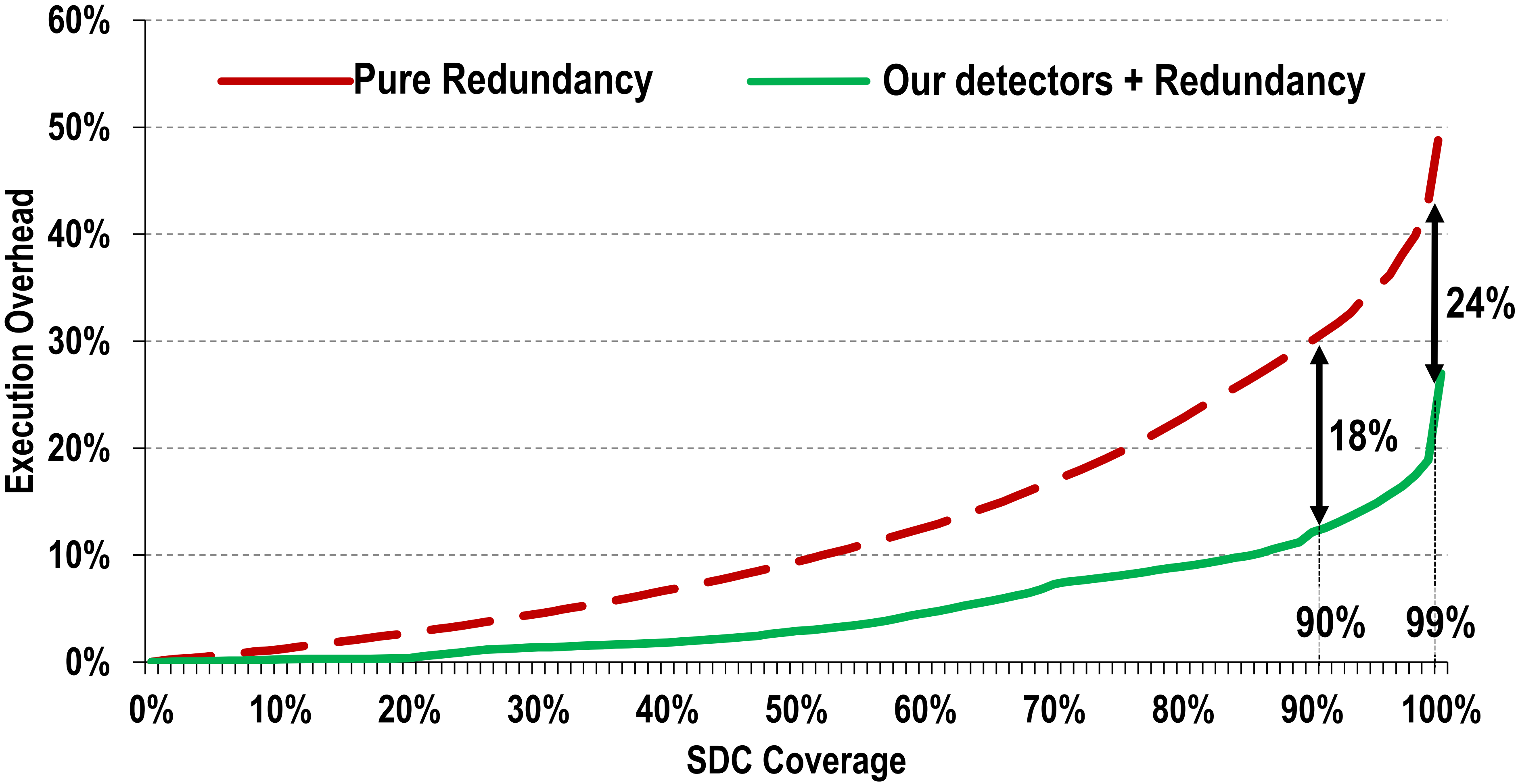
- 10% average overhead (0.1% - 18%)

Execution Overhead



- **10% average overhead (0.1% - 18%)**

SDC Coverage vs. Overhead Curve



- Consistently better over pure (selective) instruction-level duplication

Conclusions

- Reduction in SDCs is crucial for low-cost reliability
- Discovered common program properties around most SDC-causing sites
- Devised low-cost program-level detectors
  - 84% avg. SDC coverage at 10% avg. cost
- New detectors + selective duplication = Tunable resiliency at low-cost
  - Found near optimal detectors for any SDC target
  - Lower cost than pure redundancy for all SDC targets
- Future directions
  - More applications and fault models
  - Automating detectors' placement and derivation