DelayAVF

Calculating Architectural Vulnerability Factors for Delay Faults

Peter Deutsch* (MIT), Vincent Ulitzsch* (MIT/TU Berlin)

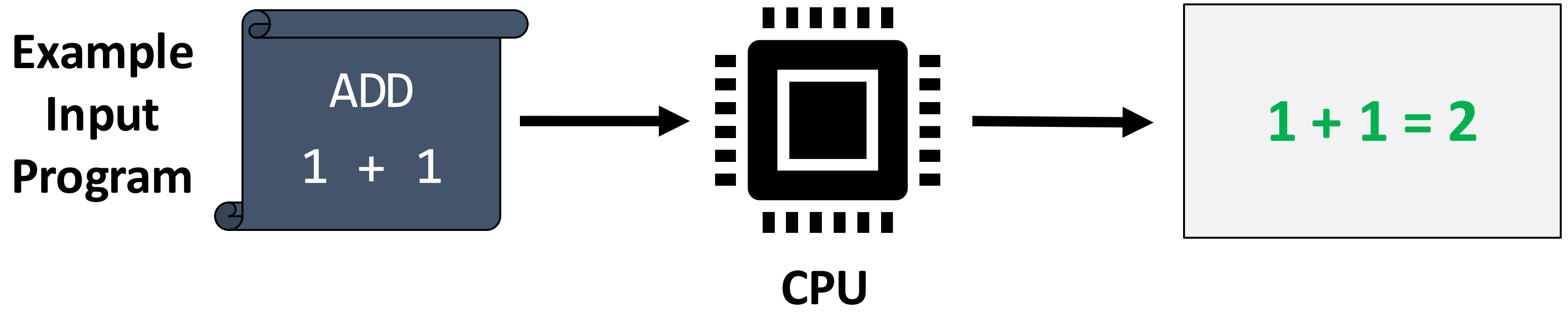
Sudhanva Gurumurthi (AMD), Vilas Sridharan (AMD)

Joel Emer (MIT), Mengjia Yan (MIT)

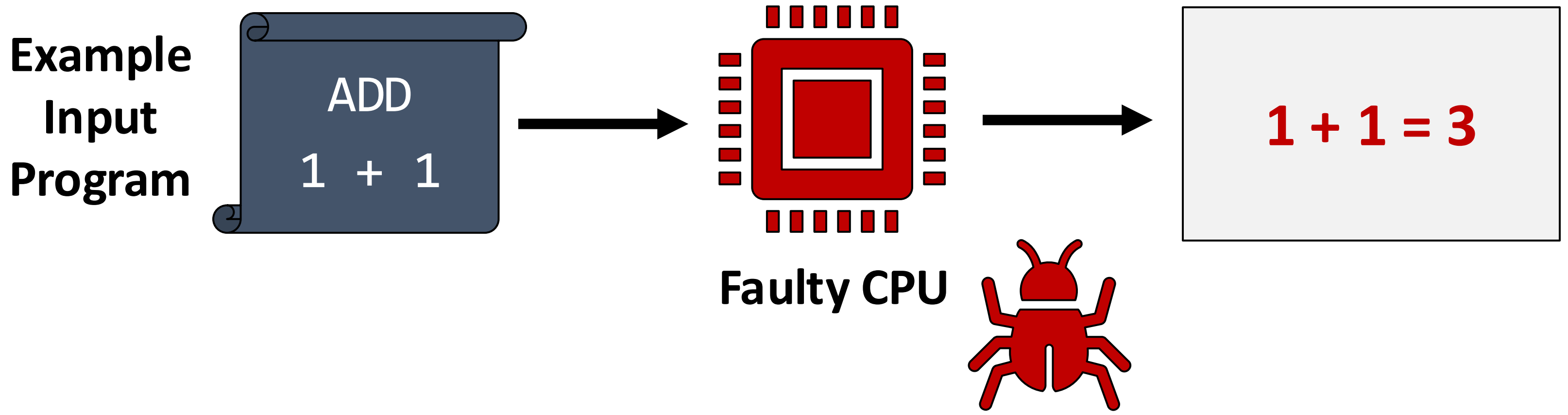
MICRO 2024 – Session 2C

November 4th, 2024

CPUs can have defects, resulting in Silent Data Corruptions (SDCs)!



SDCs can result in silently incorrect outputs, often only realized much later in the execution!



DelayAVF Overview

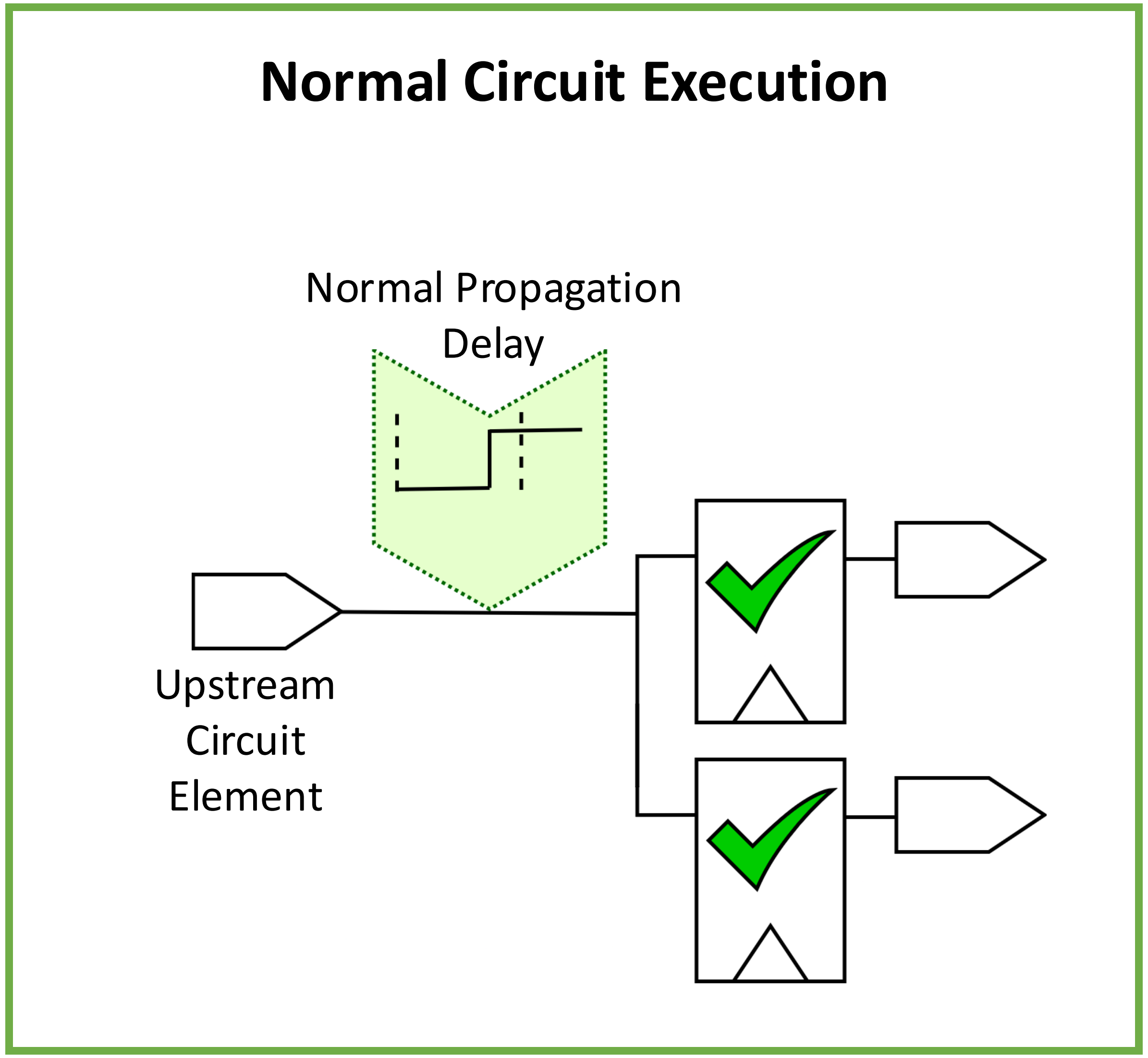
Recent findings point to **small delay faults** caused by **marginal chip defects** as an emergent reason for failures at scale.

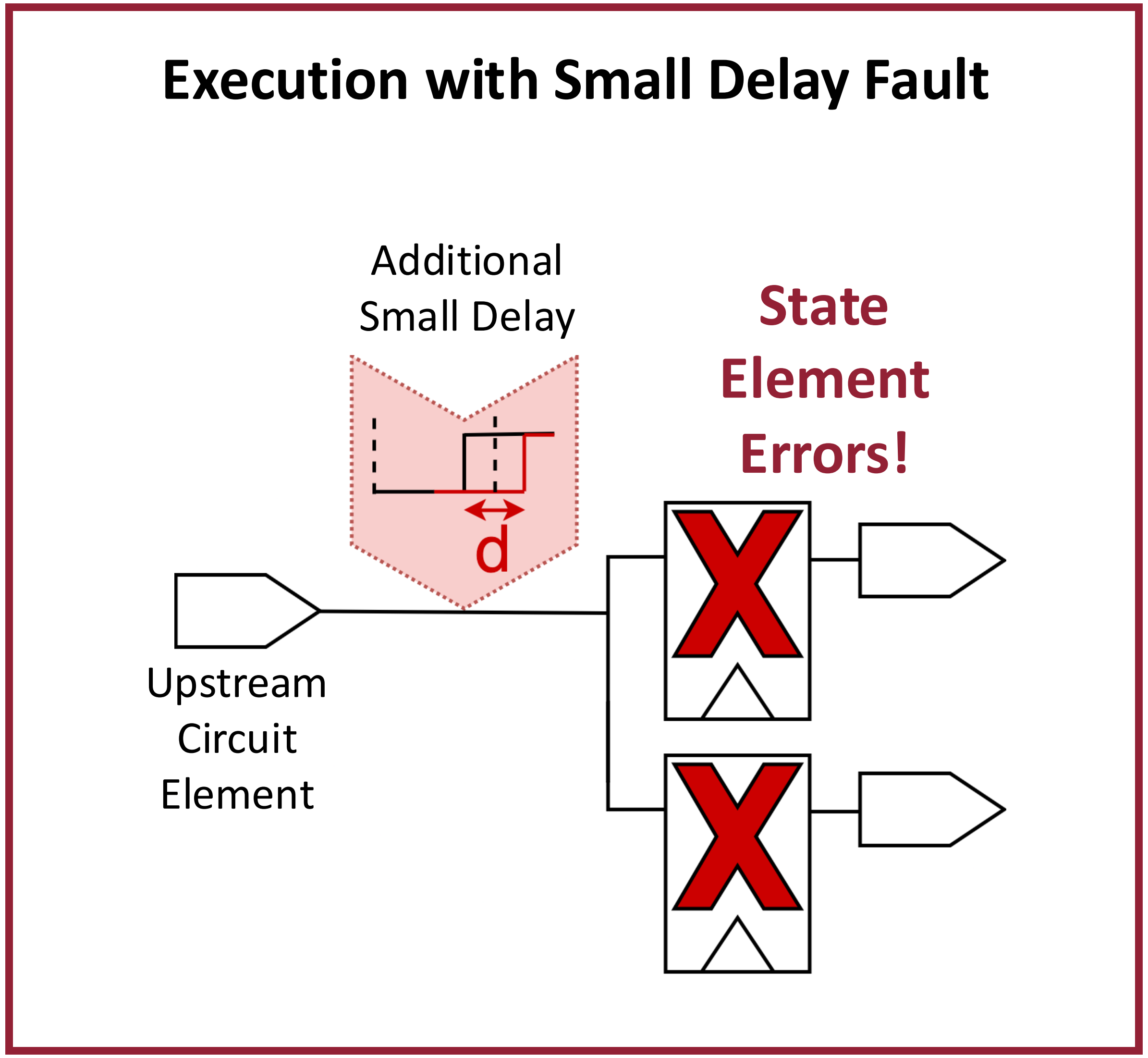
Our current ability to reason about small delay faults is limited as vulnerability estimation work **primarily focuses on particle strikes.**

Our Contribution: **DelayAVF**

A methodology to assess vulnerability to small delay faults, demonstrating actionable architectural insights.

Small delay faults (SDFs) can disturb timing, causing bit-flips!



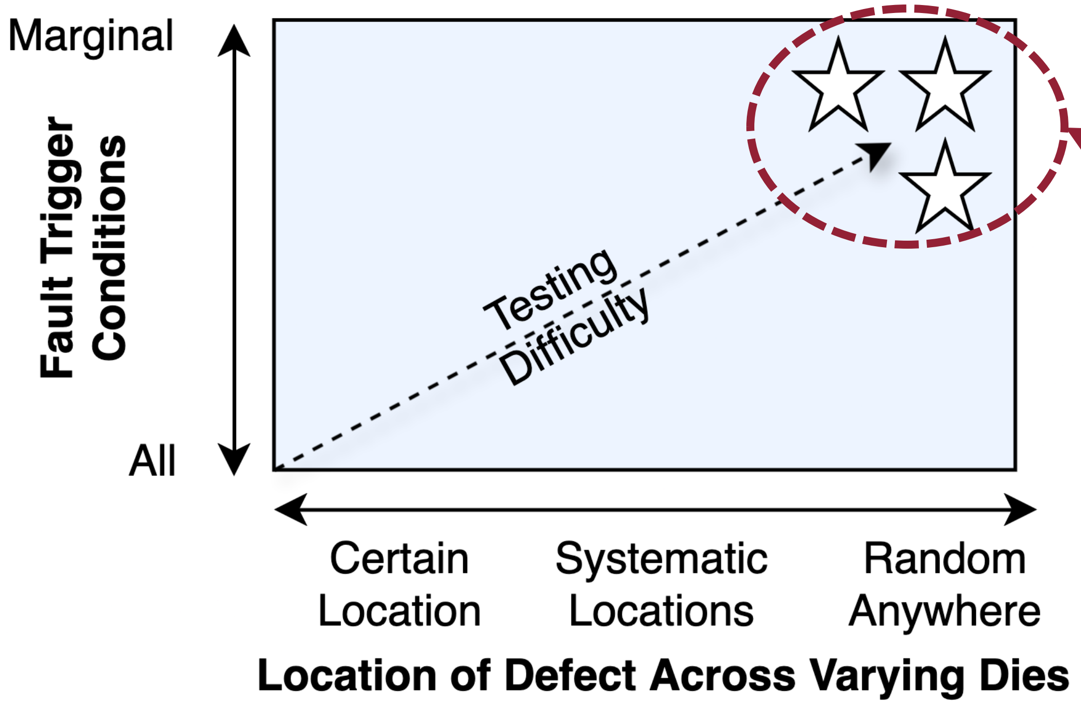


Underlying Defects' Fault Behaviors

We examine the impact of small delay faults caused by *marginal defects* (e.g., high resistance vias, shorts, etc.)

We assume:

1. Defects occur at random locations in the chip, and
2. Result in a sub-cycle delay fault condition that lasts for a single cycle.



Marginal defects are **prohibitively hard to test for** as they appear to manifest at random.

Marginal defects are **prohibitively hard to test for** as they appear to manifest at random.

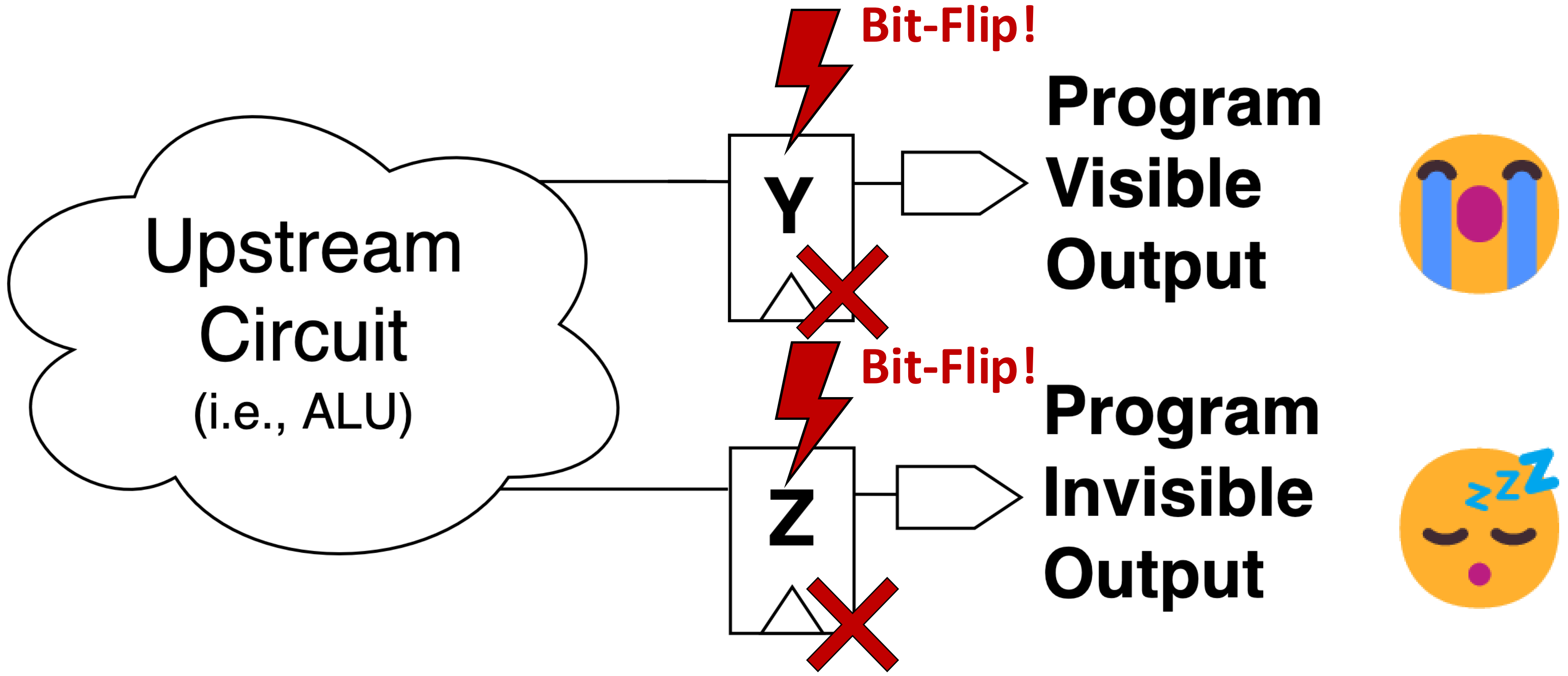
We need to add resilience at the **design stage**, but this can be **expensive**.

DelayAVF identifies which architectural structures are **most vulnerable to small delay faults**, providing an avenue for prioritized protections.

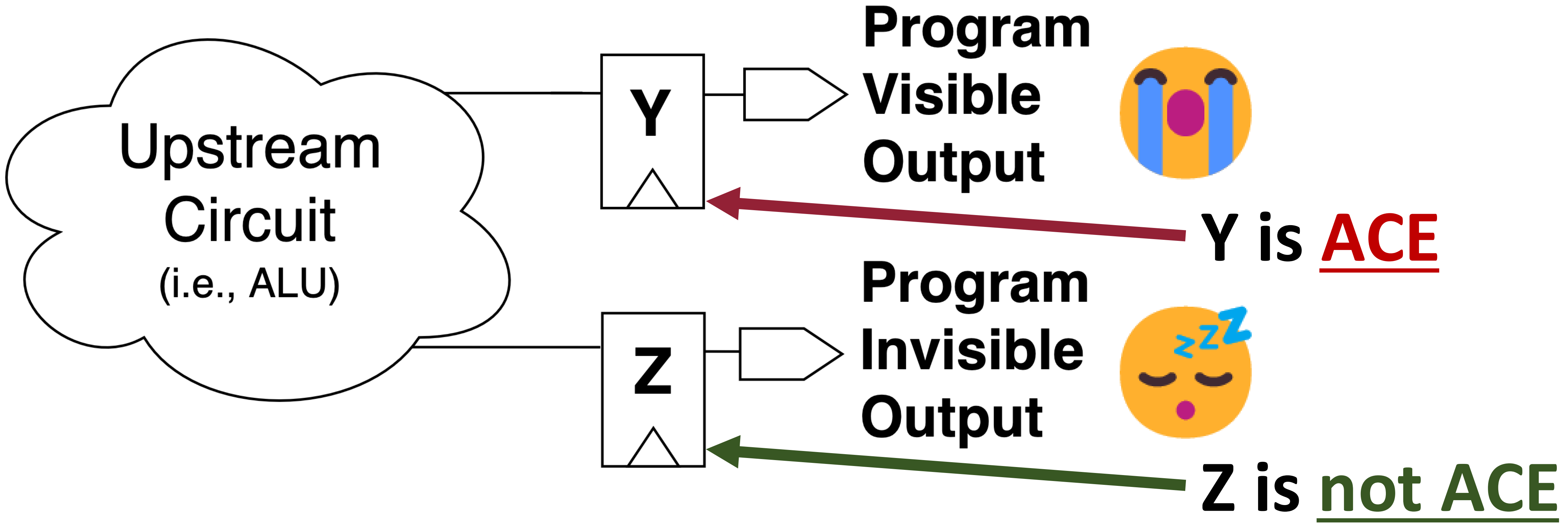
Guiding Question:

How should a computer architect prioritize the placement of protections against faults?

Observation: Not all faults are created equal



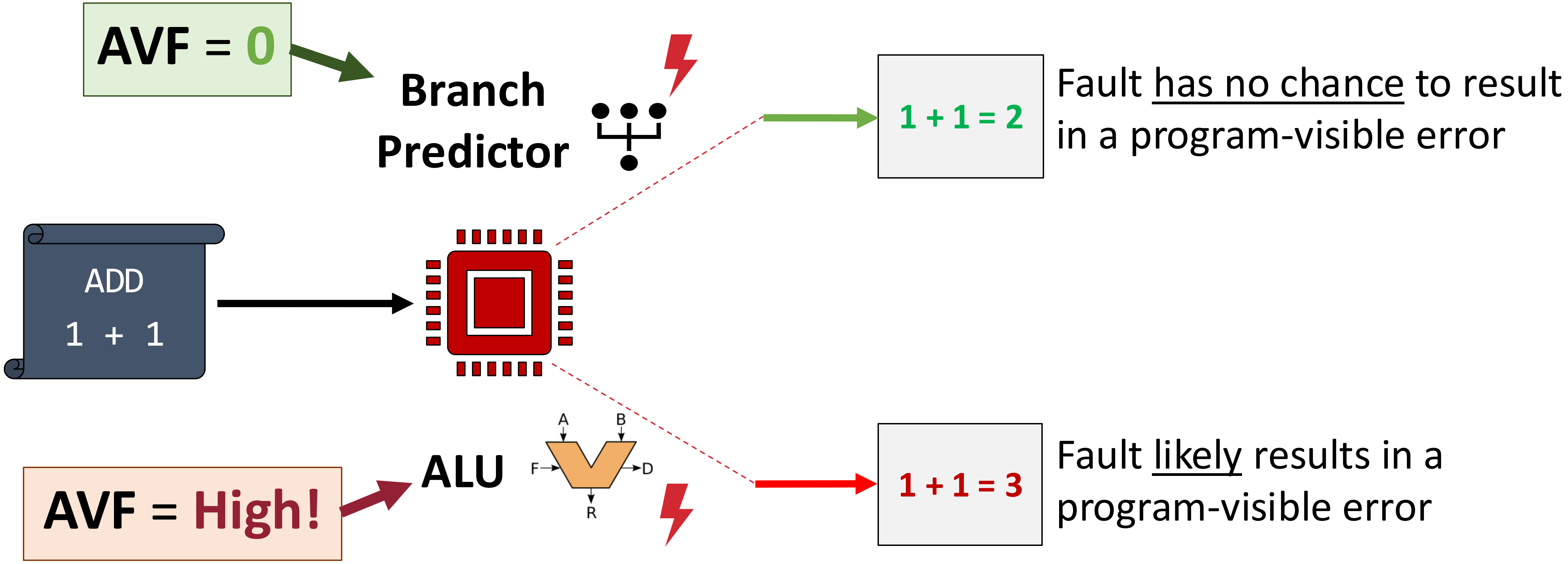
Observation: Not all faults are created equal



Definition
ACE

State element *x* *is* ACE if a bit-flip in cycle i in *x* results in a program-visible error.

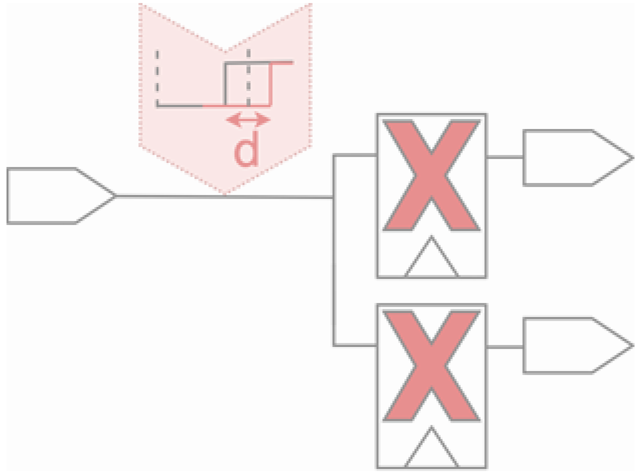
Identifying Vulnerable Structures using Architectural Vulnerability Factor


AVF = 0
Branch Predictor
ADD
1 + 1
1 + 1 = 2
Fault has no chance to result in a program-visible error
AVF = High!
ALU
A
B
F
D
R
1 + 1 = 3
Fault likely results in a program-visible error

Rank structures according to their Architectural Vulnerability Factor (AVF):

$$AVF(S) = \sum_{i=1}^{N} \frac{\#\text{ of } ACE \text{ flops in structure S in cycle } i}{\text{Total number of cycles } N \cdot \#\text{ of Flops in S}}$$

This Work: DelayAVF



We want to estimate a structure's **DelayAVF**:

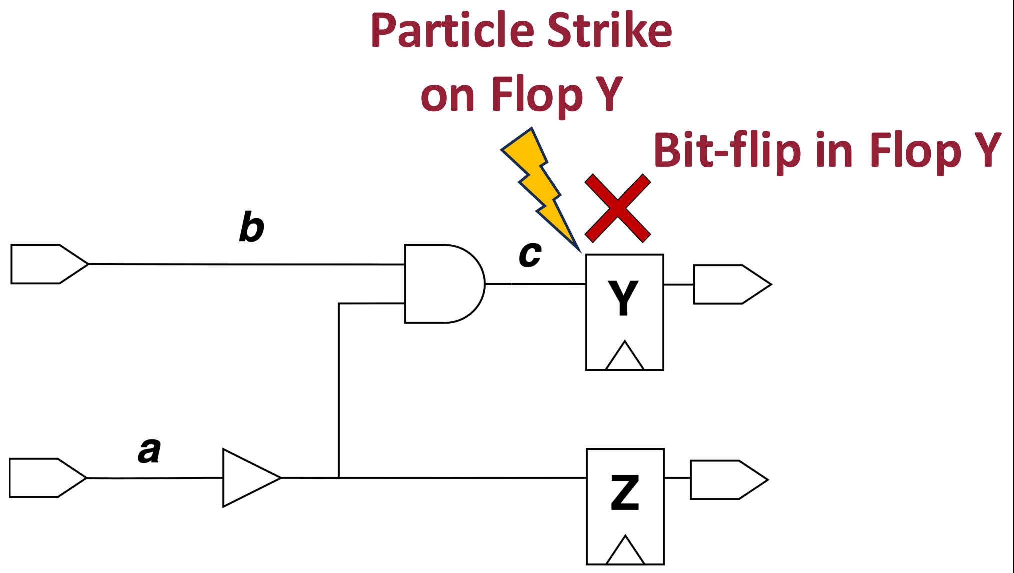
The probability that a <u>small delay fault</u> in a particular architectural structure results in a program-visible error.

Can we re-leverage the concept of ACE to estimate a structure’s DelayAVF?

Two key challenges emerge as we can no longer reason about small delay faults via individual state elements!

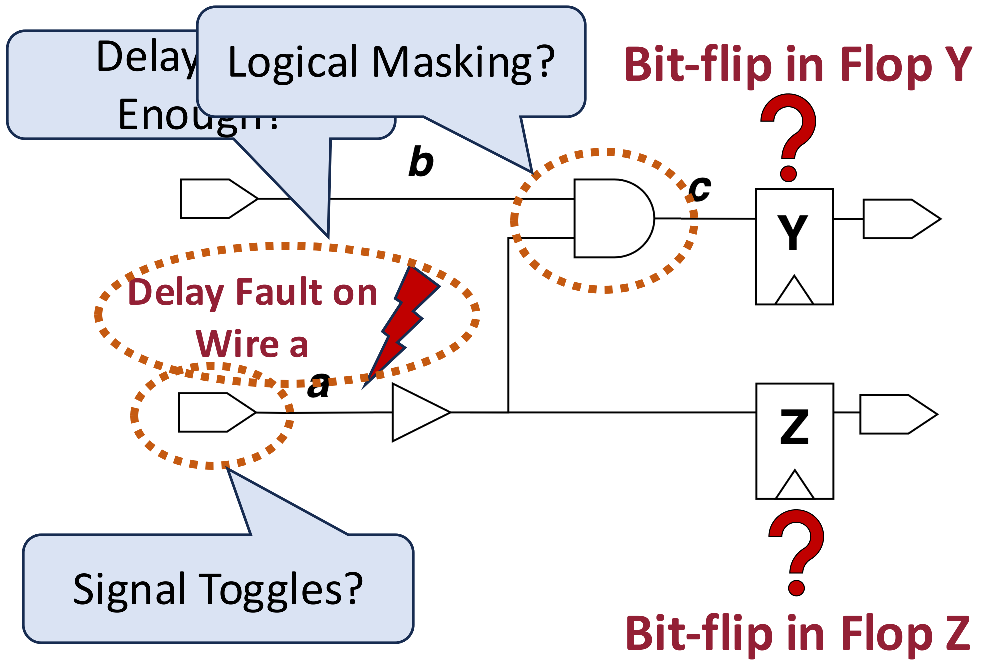
Challenge 1: The point of fault is no longer the point of error

Particle Strike Model



The particle **directly strikes** the state element, which subsequently has an error.

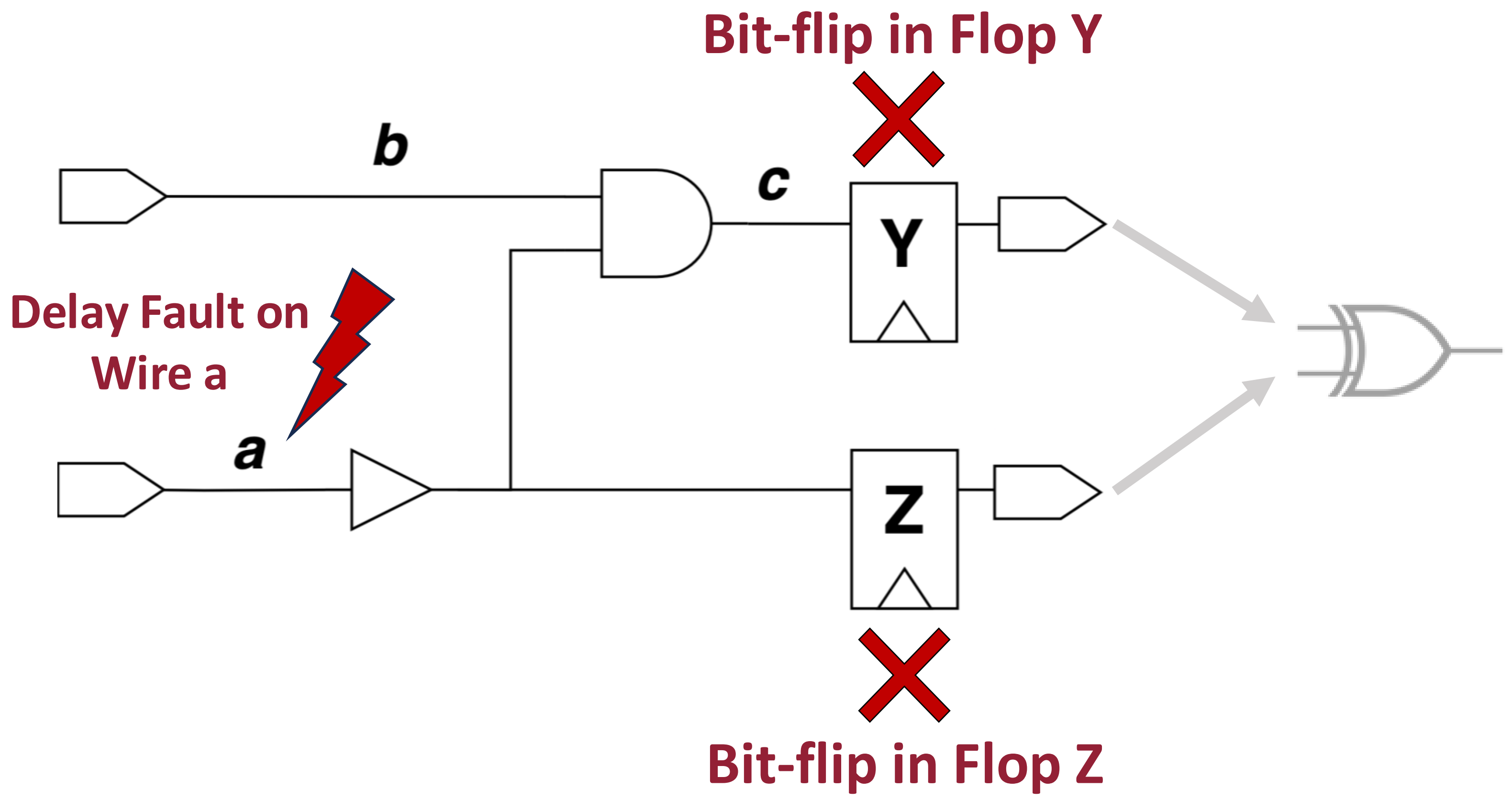
Small Delay Fault Model



The delay affects the **wire**, however the error is only observed at **downstream** state elements!

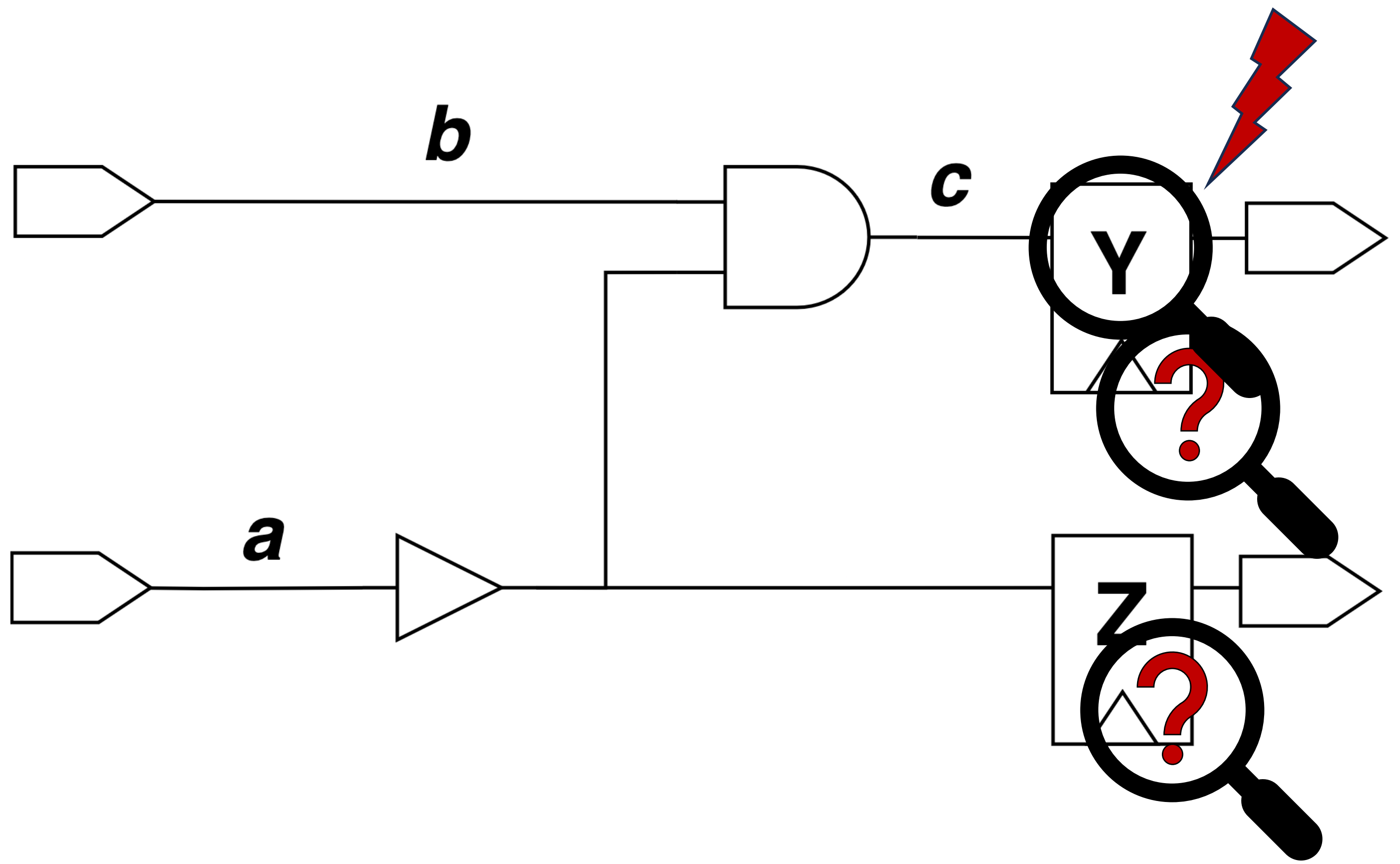
We cannot reason about vulnerability to small delay faults by solely examining state elements!

Challenge 2: Delay faults can result in multiple *simultaneous* bit flips



We need to reason about errors that occur simultaneously, potentially interacting with each other!

DelayAVF's Key Idea: Reason about the vulnerability of the structure's circuit elements (e.g., wires or gates) rather than state elements.

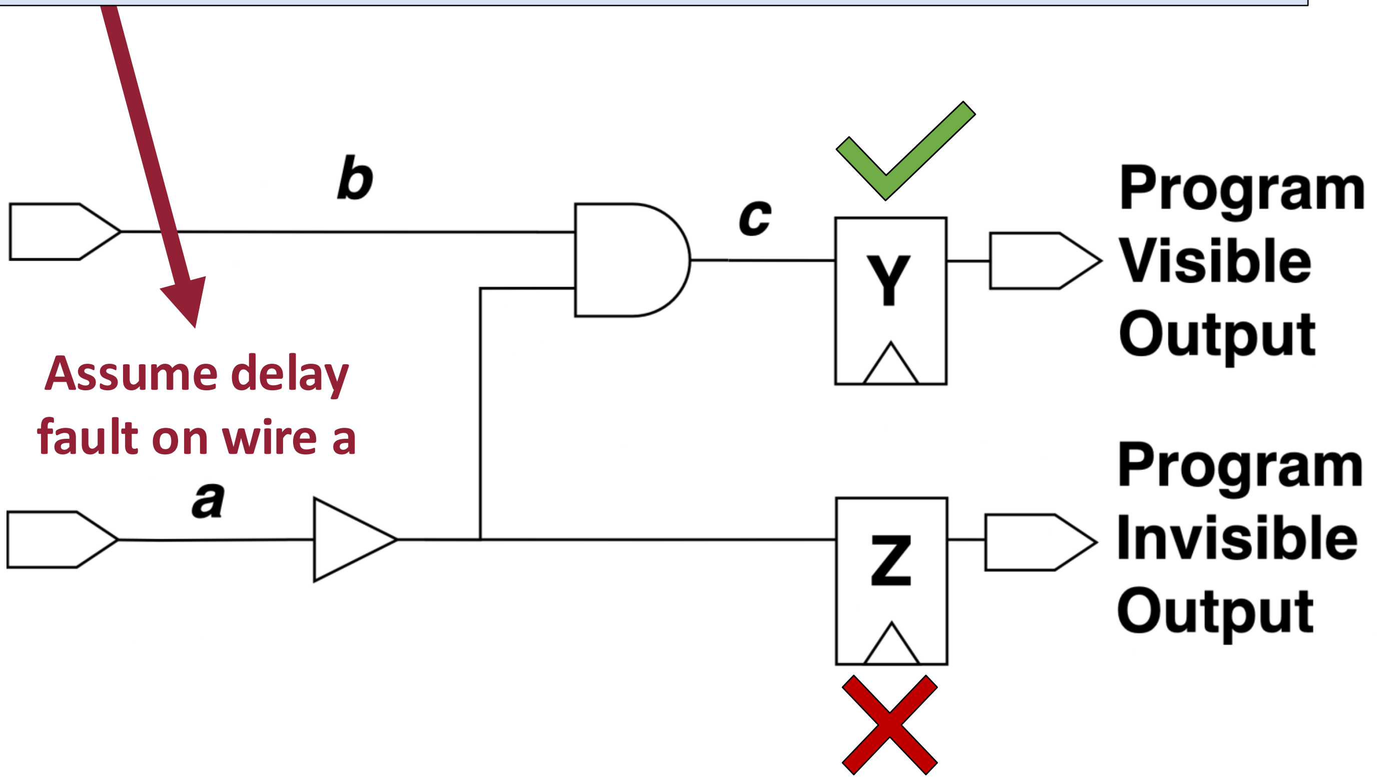


Deriving DelayAVF via the vulnerability of individual circuit elements

Definition
DelayACE

Circuit element *a is* $DelayACE_d(a, i)$ in cycle i if a small delay fault of duration d on *a* results in a program-visible error.

Error in *Z*
↓
***a* is not DelayACE**

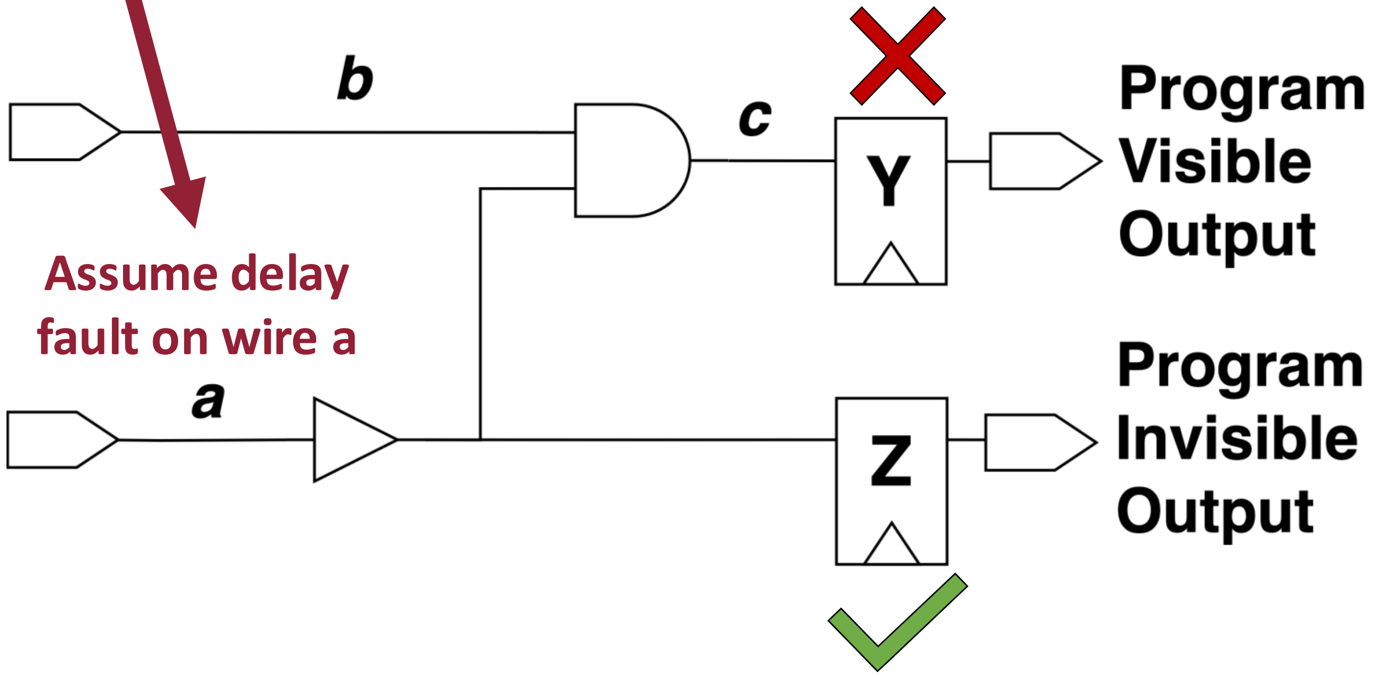


Deriving DelayAVF via the vulnerability of individual circuit elements

Definition
DelayACE

Circuit element *a is* $DelayACE_d(a, i)$ in cycle i if a small delay fault of duration d on *a* results in a program-visible error.

Error in *Y*
↓
a is DelayACE



How can we calculate DelayACE in practice?

We can compute DelayACE into two steps:

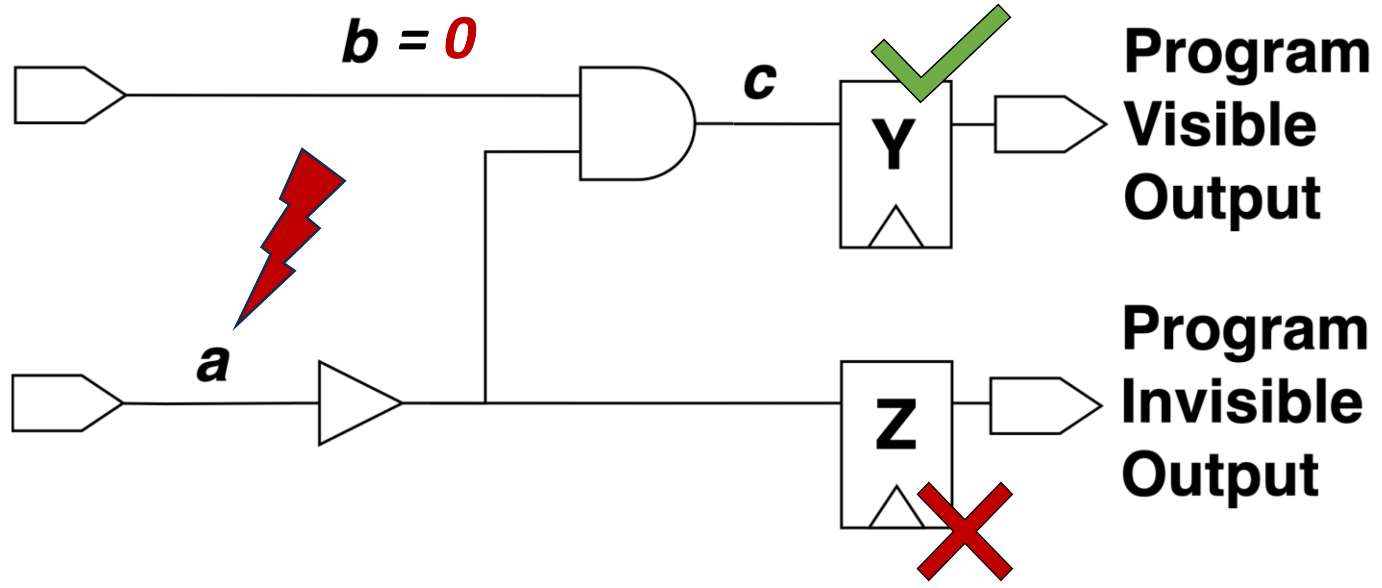
1. What is the set of state elements experiencing an error?
2. Will simultaneous errors in these state elements cause a program-visible error?

Two-step approach to determining whether a circuit element is DelayACE

$$DynamicReachable_d(a, i)$$

What is the set of *state elements* experiencing an error?

$$DynamicReachable_d(\boldsymbol{a}, \boldsymbol{i}) = \{Z\}$$

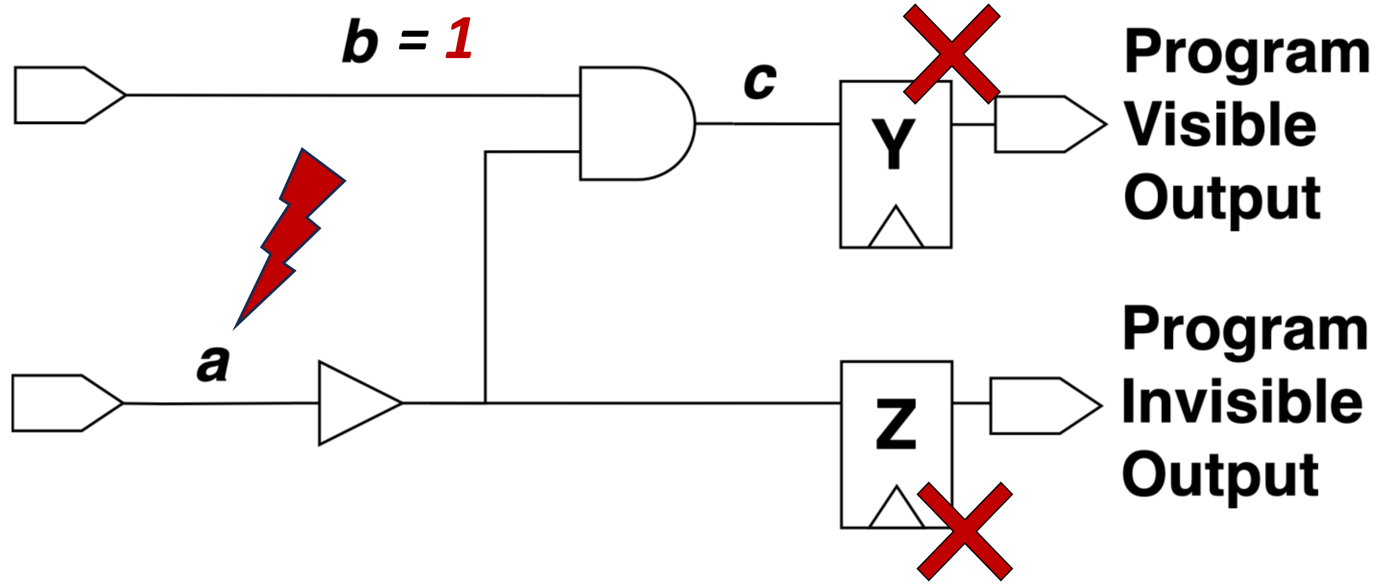


Two-step approach to determining whether a circuit element is DelayACE

$$DynamicReachable_d(a, i)$$

What is the set of *state elements* experiencing an error?

$$DynamicReachable_d(\boldsymbol{a}, \boldsymbol{i}) = \{Y, Z\}$$



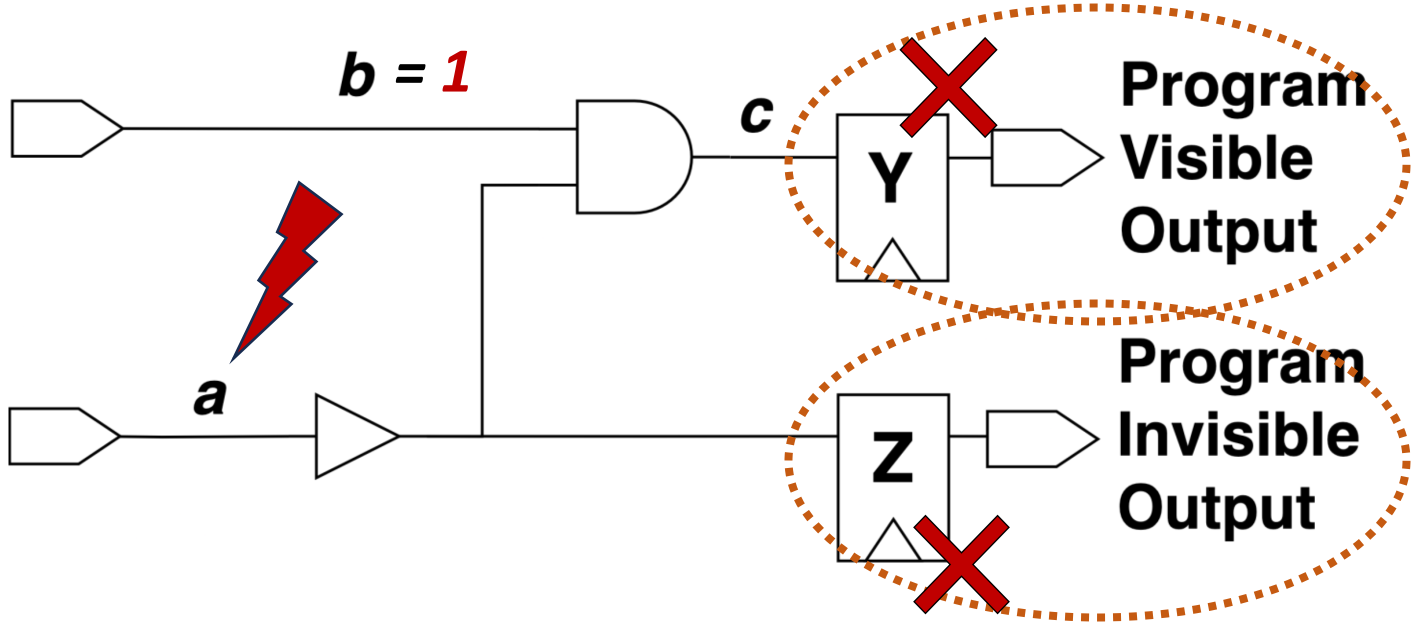
Two-step approach to determining whether a circuit element is DelayACE

$$DelayACE_d(a,i) = GroupACE(DynamicReachable_d(a,i), i+1)$$

Will simultaneous errors in these state elements cause a program-visible error?

What is the set of *state elements* experiencing an error?

$$GroupACE(\{Y,Z\}) = Yes, \mathbf{DelayACE_d(a,i) = Yes}$$



Concretely determining DelayAVF

$$\boldsymbol{DelayACE_d(a, i) = GroupACE(DynamicReachable_d(a, i), i + 1)}$$

This two-step approach enables tractable computation of $DelayACE$!

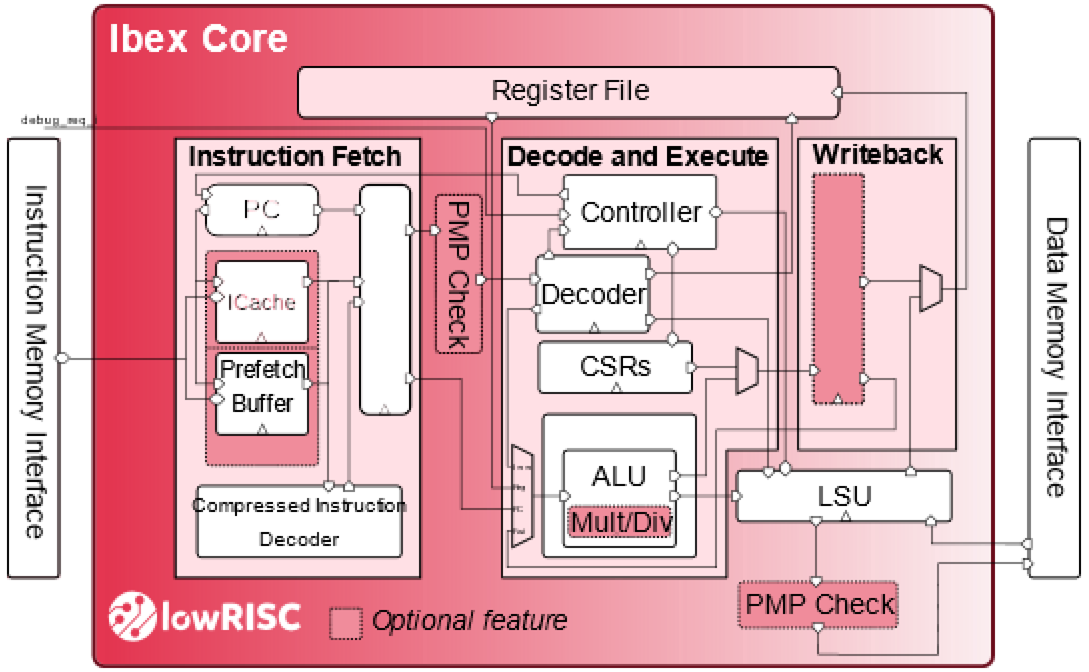
DelayAVF Definition

The fraction of circuit elements in a structure S that are DelayACE, averaged over all cycles of a reference program.

$$DelayAVF_d(S) = \sum_{i=1}^{N} \frac{\text{\# Number of } DelayACE_d \text{ elements in structure } S \text{ in cycle } i}{\text{\# Number of Cycles } N \cdot \text{\# Total Number Of Elements in } S}$$

Case Study: IBEX RISC-V Core

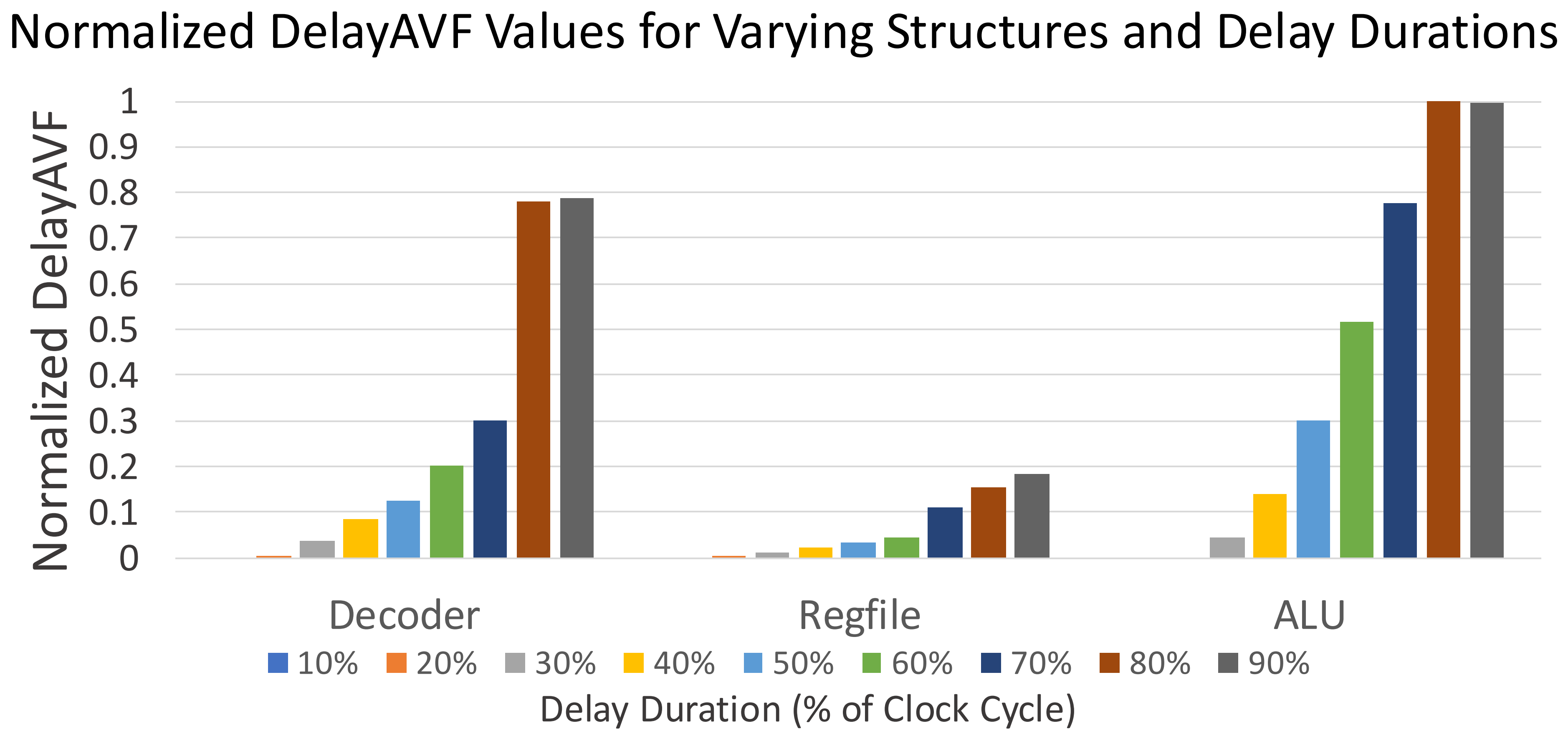
- We evaluate DelayAVF for several structures in IBEX, an in-order open-source RISC-V core.
- We compute DelayAVF with reference to the Beebs benchmark suite using a 45nm technology library.



IBEX Block Diagram

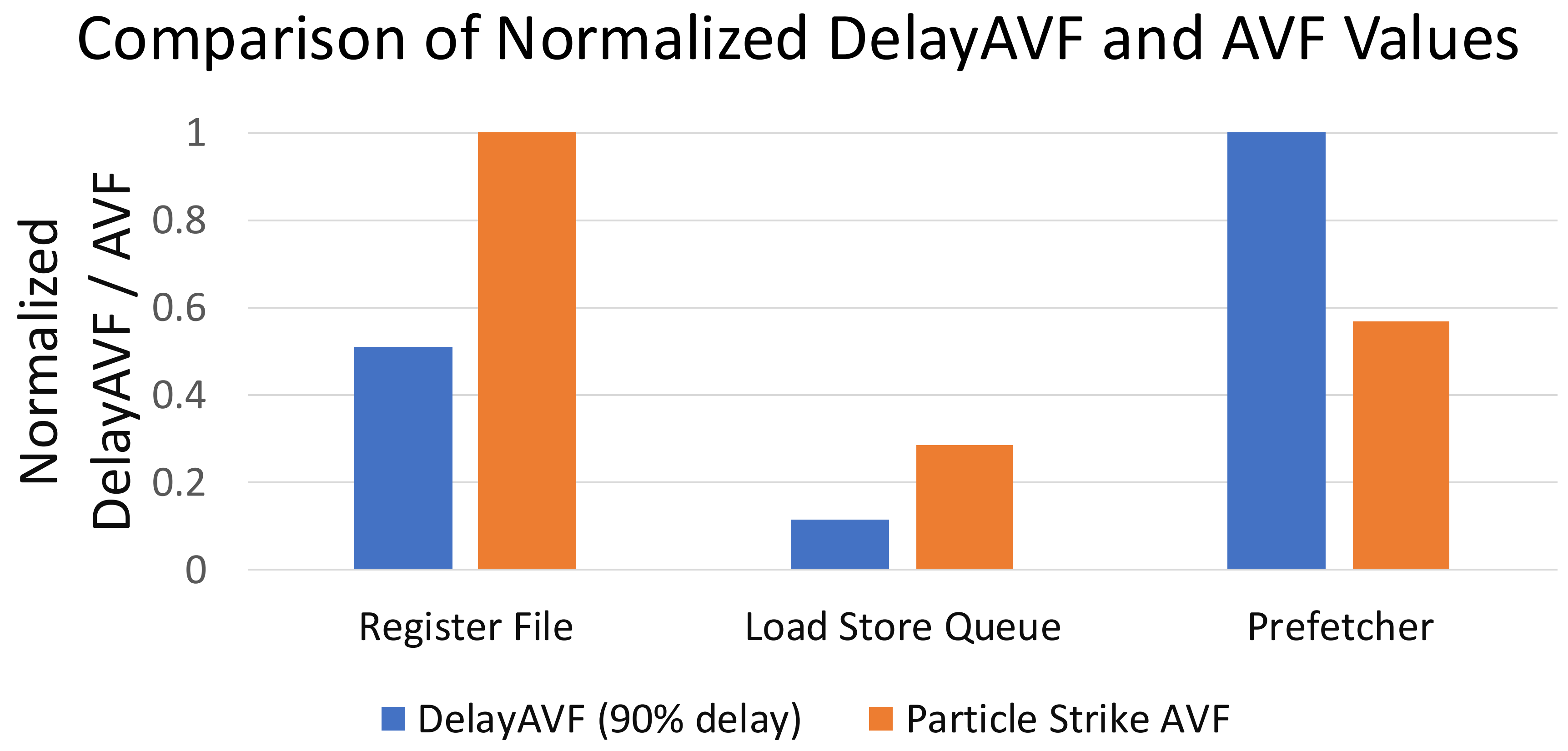
DelayAVF’s Insights

Q1: Is DelayAVF useful in guiding placement of mitigations? Yes!



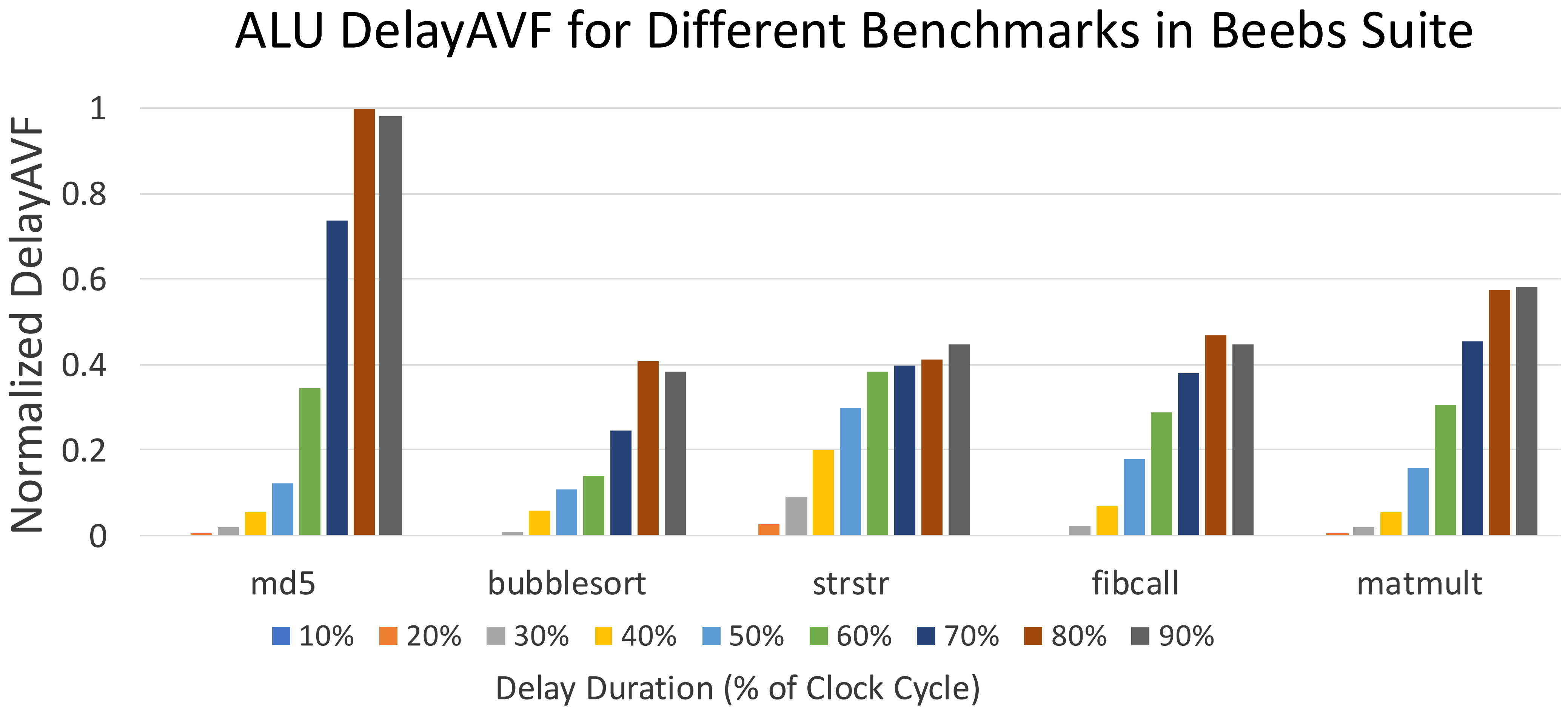
DelayAVF reveals that different microarchitectural structures can have significantly different vulnerabilities to small delay faults!

Q2: Could we just use particle-strike AVF? No, it leads to different rankings!



High vulnerability to particle strikes does not imply a high vulnerability to small delay faults (and vice-versa).

Q3: Is Static Timing Analysis Sufficient to Reason About Delay Vulnerability? No!



Both program and architectural-level effects can influence vulnerability to delay faults.

Much more in the paper!

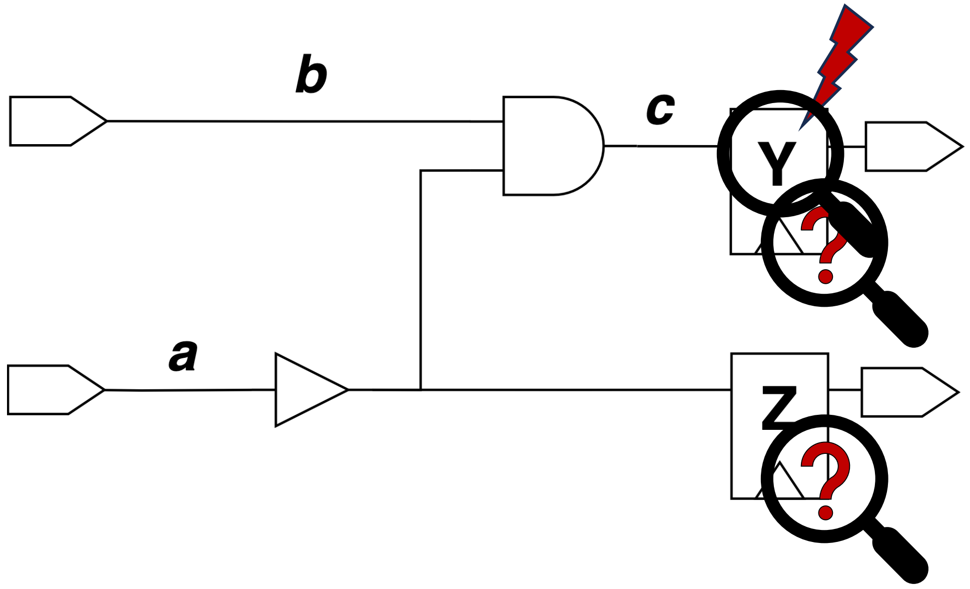
➡ How we model circuit timing, when small delay faults occur, and their impact.

➡ Analysis of interactions between multiple simultaneous errors (ACE Compounding & Interference).

➡ A method to heuristically approximate GroupACE via particle-strike ACEness.

Summary: A methodology to target mitigations against delay faults

- **Prior work:** Estimate AVF through the ACEness of state elements.
- **This work:** A metric to quantify the vulnerability to small delay faults.
- **Key Insights**: We can estimate DelayAVF through DelayACEness, shifting the focus from state elements to circuit elements.
- **Future Work:** We hope that DelayAVF will inspire future work examining delay faults.



$$DelayACE_d(a,i) = GroupACE(DynamicReachable_d(a,i), i+1)$$

Questions/Comments?
pwd@mit.edu, viniul@mit.edu

https://github.com/viniul/delayAVF