# Fuzzilicon: A Post-Silicon Microcode-Guided x86 CPU Fuzzer

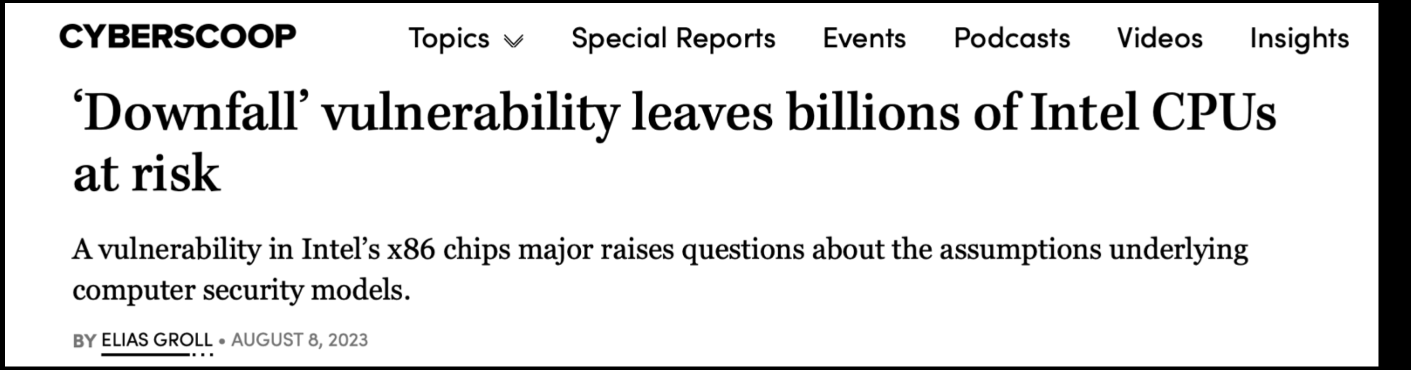
Johannes Lenzen, Mohamadreza Rostami, Lichao Wu, Ahmad-Reza Sadeghi

System Security Lab, Technical University of Darmstadt

SYSTEM SECURITY LAB

TECHNISCHE
UNIVERSITÄT
DARMSTADT

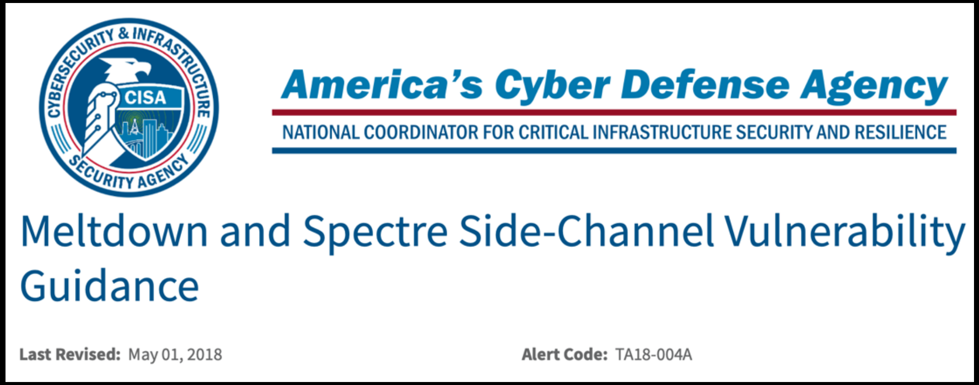
# CPU Vulnerabilities: A Crucial Threat



## ‘Downfall’ vulnerability leaves billions of Intel CPUs at risk

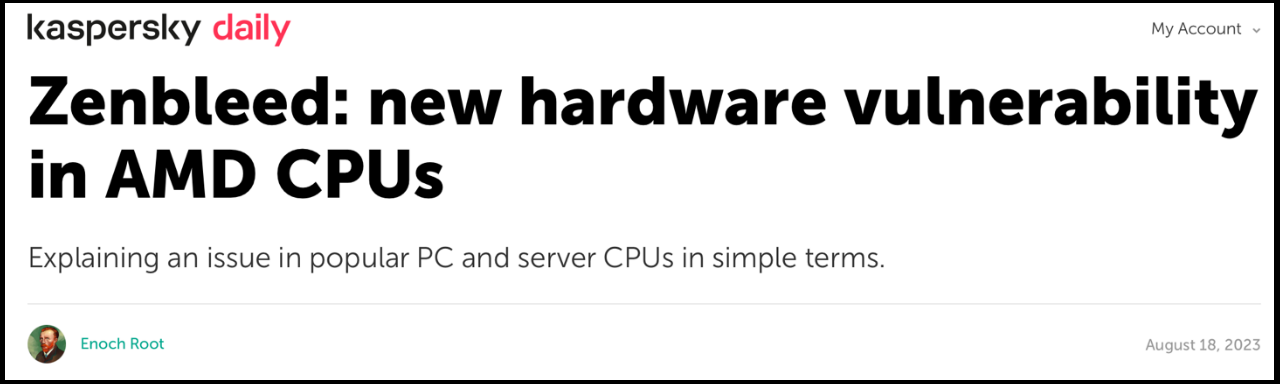
A vulnerability in Intel’s x86 chips major raises questions about the assumptions underlying computer security models.

BY ELIAS GROLL • AUGUST 8, 2023


America’s Cyber Defense Agency
NATIONAL COORDINATOR FOR CRITICAL INFRASTRUCTURE SECURITY AND RESILIENCE

## Meltdown and Spectre Side-Channel Vulnerability Guidance

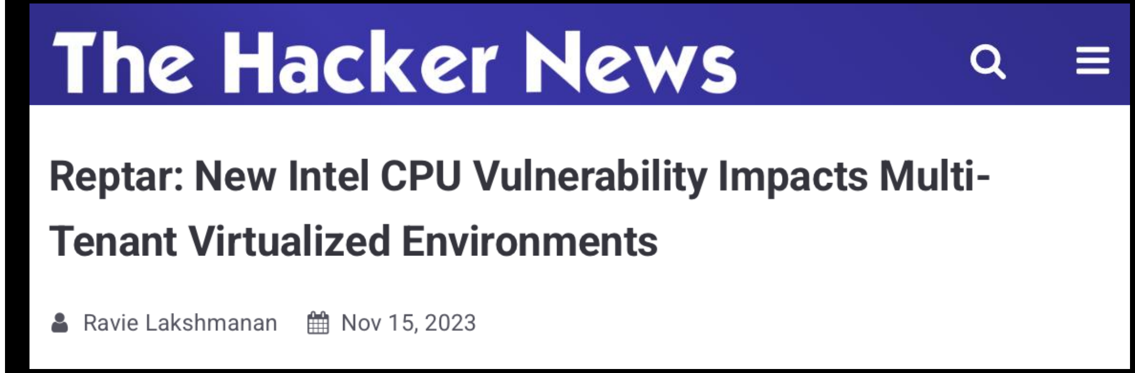
**Last Revised:** May 01, 2018 **Alert Code:** TA18-004A



## Zenbleed: new hardware vulnerability in AMD CPUs

Explaining an issue in popular PC and server CPUs in simple terms.

Enoch Root August 18, 2023



## Reptar: New Intel CPU Vulnerability Impacts Multi-Tenant Virtualized Environments

Ravie Lakshmanan Nov 15, 2023

# The Big Picture

## The Challenge (Context)

- CPU Vulnerabilities
- Proprietary Opacity

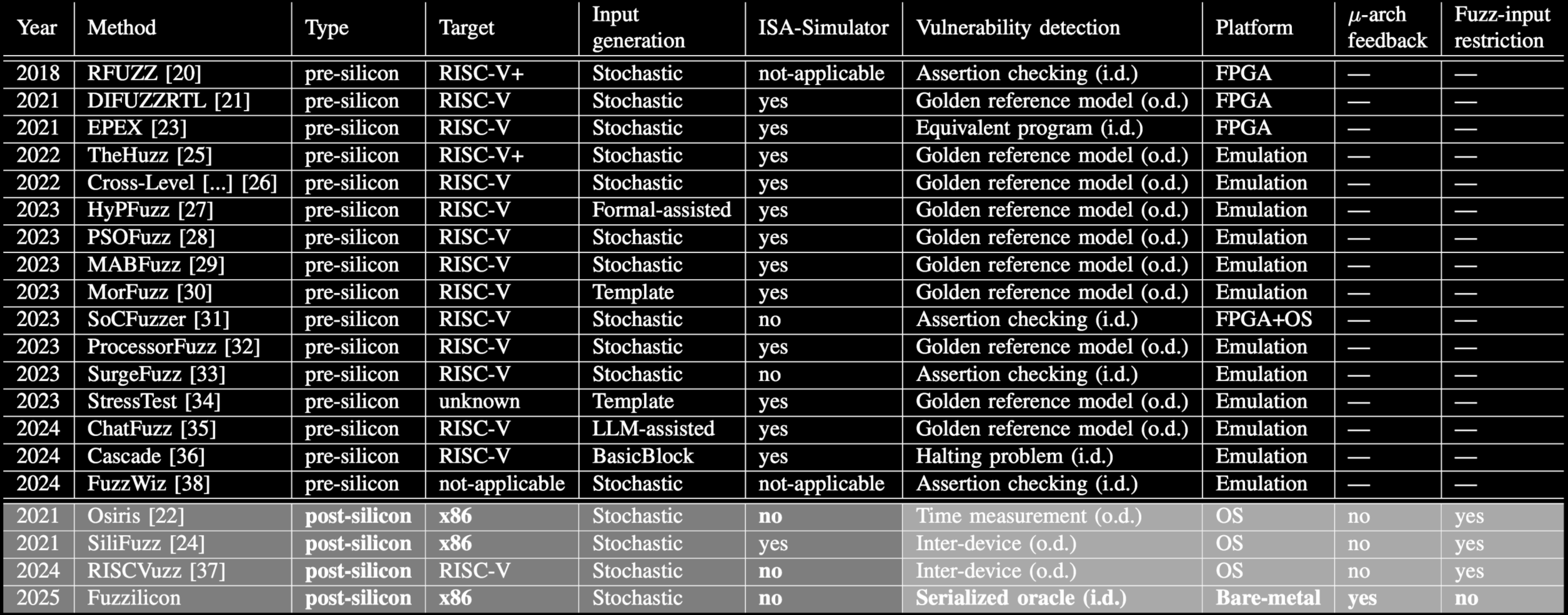
## Limitations of Existing Solutions

- Only Simple ISA
- Limited Feedback
- Absence of Oracles
- Instability & Noise

## Our Solution, Fuzzilicon

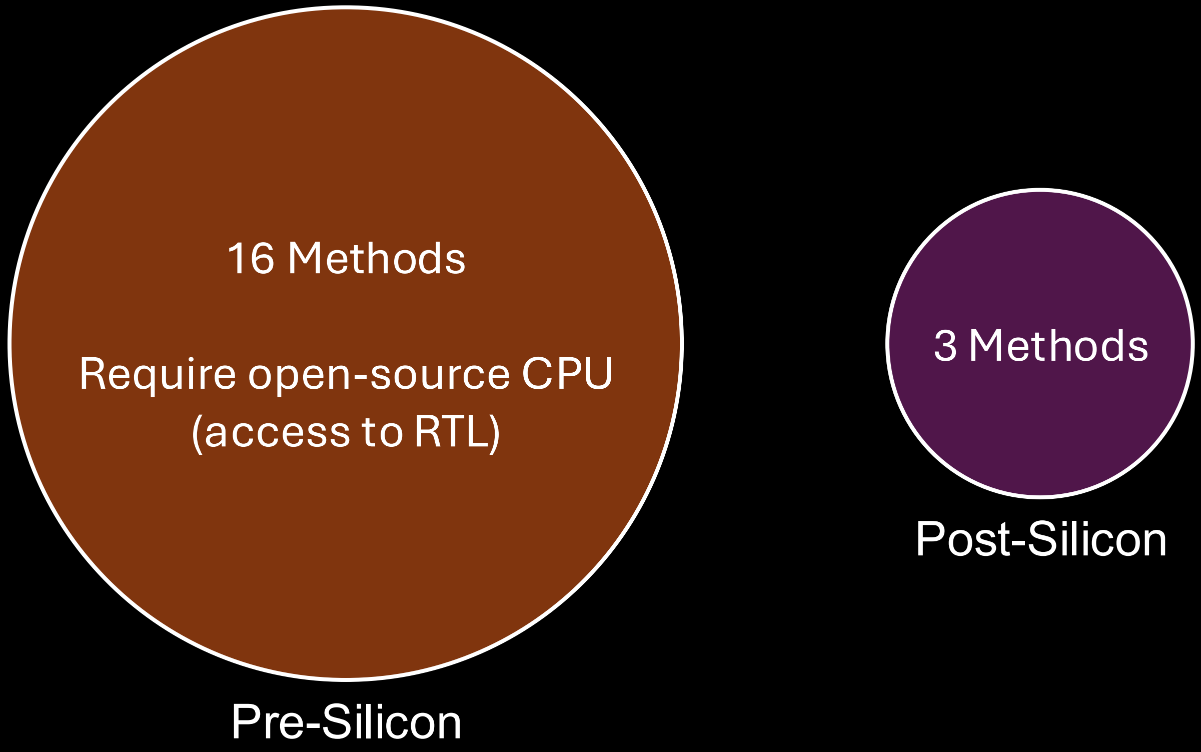
- Microarchitectural Visibility
- Serialization Oracle
- Hypervisor Isolation

# Existing CPU Fuzzing Methods



| Year | Method | Type | Target | Input generation | ISA-Simulator | Vulnerability detection | Platform | $\mu$-arch feedback | Fuzz-input restriction |
|---|---|---|---|---|---|---|---|---|---|
| 2018 | RFUZZ [20] | pre-silicon | RISC-V+ | Stochastic | not-applicable | Assertion checking (i.d.) | FPGA | — | — |
| 2021 | DIFUZZRTL [21] | pre-silicon | RISC-V | Stochastic | yes | Golden reference model (o.d.) | FPGA | — | — |
| 2021 | EPEX [23] | pre-silicon | RISC-V | Stochastic | yes | Equivalent program (i.d.) | FPGA | — | — |
| 2022 | TheHuzz [25] | pre-silicon | RISC-V+ | Stochastic | yes | Golden reference model (o.d.) | Emulation | — | — |
| 2022 | Cross-Level [...] [26] | pre-silicon | RISC-V | Stochastic | yes | Golden reference model (o.d.) | Emulation | — | — |
| 2023 | HyPFuzz [27] | pre-silicon | RISC-V | Formal-assisted | yes | Golden reference model (o.d.) | Emulation | — | — |
| 2023 | PSOFuzz [28] | pre-silicon | RISC-V | Stochastic | yes | Golden reference model (o.d.) | Emulation | — | — |
| 2023 | MABFuzz [29] | pre-silicon | RISC-V | Stochastic | yes | Golden reference model (o.d.) | Emulation | — | — |
| 2023 | MorFuzz [30] | pre-silicon | RISC-V | Template | yes | Golden reference model (o.d.) | Emulation | — | — |
| 2023 | SoCFuzzer [31] | pre-silicon | RISC-V | Stochastic | no | Assertion checking (i.d.) | FPGA+OS | — | — |
| 2023 | ProcessorFuzz [32] | pre-silicon | RISC-V | Stochastic | yes | Golden reference model (o.d.) | Emulation | — | — |
| 2023 | SurgeFuzz [33] | pre-silicon | RISC-V | Stochastic | no | Assertion checking (i.d.) | Emulation | — | — |
| 2023 | StressTest [34] | pre-silicon | unknown | Template | yes | Golden reference model (o.d.) | Emulation | — | — |
| 2024 | ChatFuzz [35] | pre-silicon | RISC-V | LLM-assisted | yes | Golden reference model (o.d.) | Emulation | — | — |
| 2024 | Cascade [36] | pre-silicon | RISC-V | BasicBlock | yes | Halting problem (i.d.) | Emulation | — | — |
| 2024 | FuzzWiz [38] | pre-silicon | not-applicable | Stochastic | not-applicable | Assertion checking (i.d.) | Emulation | — | — |
| 2021 | Osiris [22] | **post-silicon** | **x86** | Stochastic | **no** | Time measurement (o.d.) | OS | no | yes |
| 2021 | SiliFuzz [24] | **post-silicon** | **x86** | Stochastic | yes | Inter-device (o.d.) | OS | no | yes |
| 2024 | RISCVuzz [37] | **post-silicon** | RISC-V | Stochastic | **no** | Inter-device (o.d.) | OS | no | yes |
| 2025 | Fuzzilicon | **post-silicon** | **x86** | Stochastic | **no** | **Serialized oracle (i.d.)** | **Bare-metal** | **yes** | **no** |

# Existing CPU Fuzzing Methods



16 Methods

Require open-source CPU (access to RTL)

Pre-Silicon

3 Methods

Post-Silicon

# Why post-silicon?

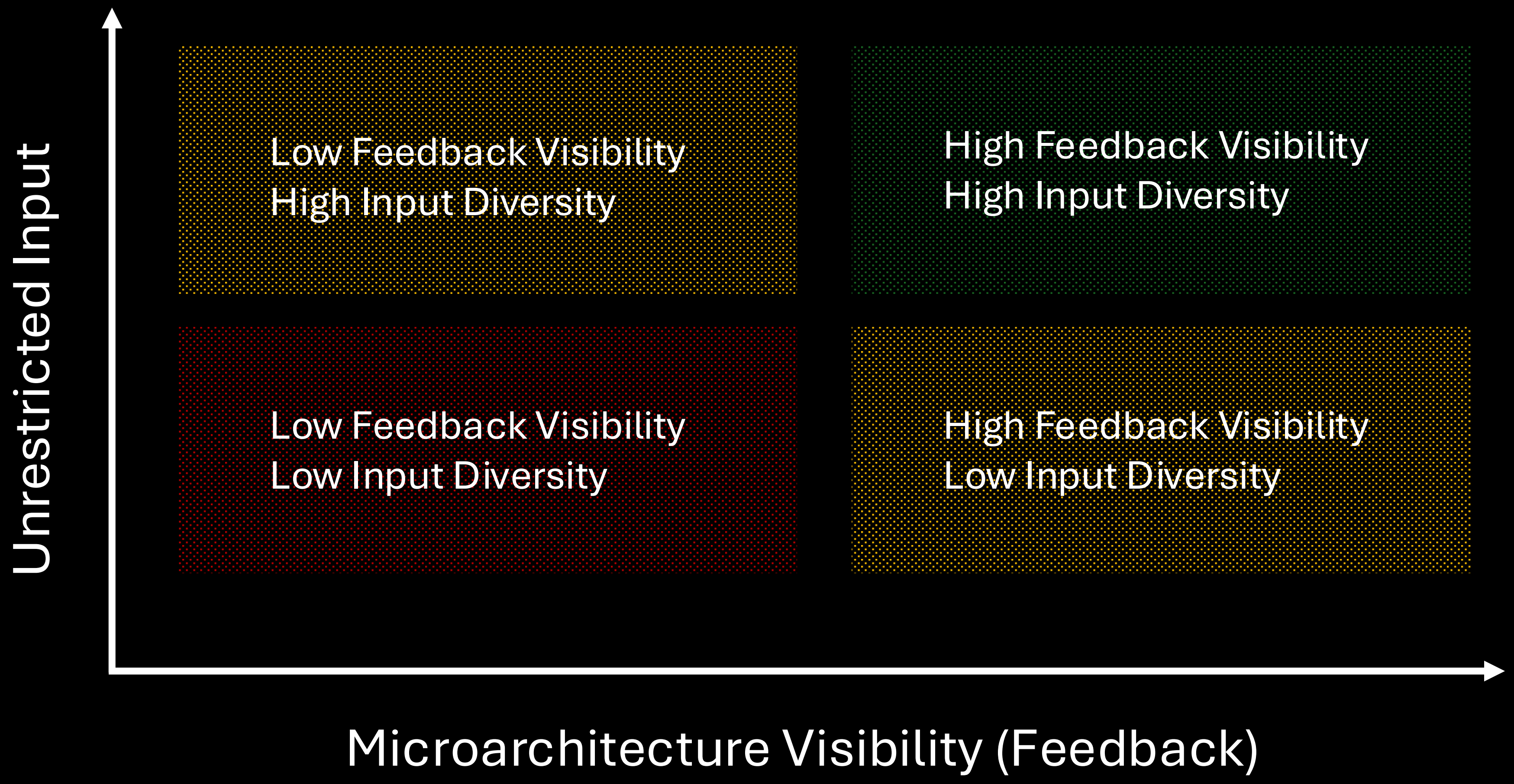
Captures real-world behavior that directly affects end users.

Includes modern CPU features missing in open-source designs.

Targets complex x86 ISA rather than simplified RISC-V models.

Uses real hardware, avoiding slow and inaccurate simulation.

# Existing CPU Fuzzing Methods



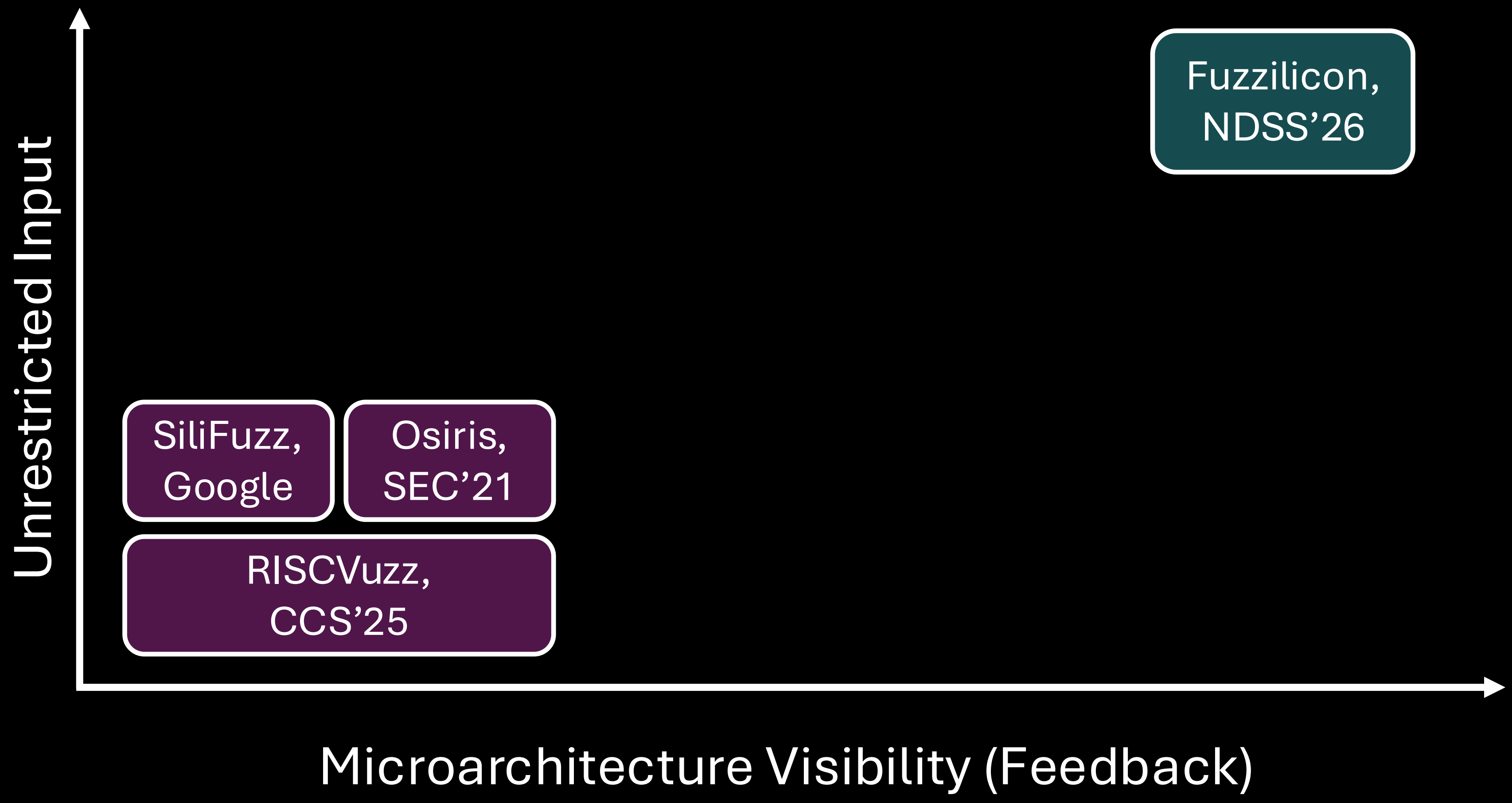
Low Feedback Visibility
High Input Diversity

High Feedback Visibility
High Input Diversity

Low Feedback Visibility
Low Input Diversity

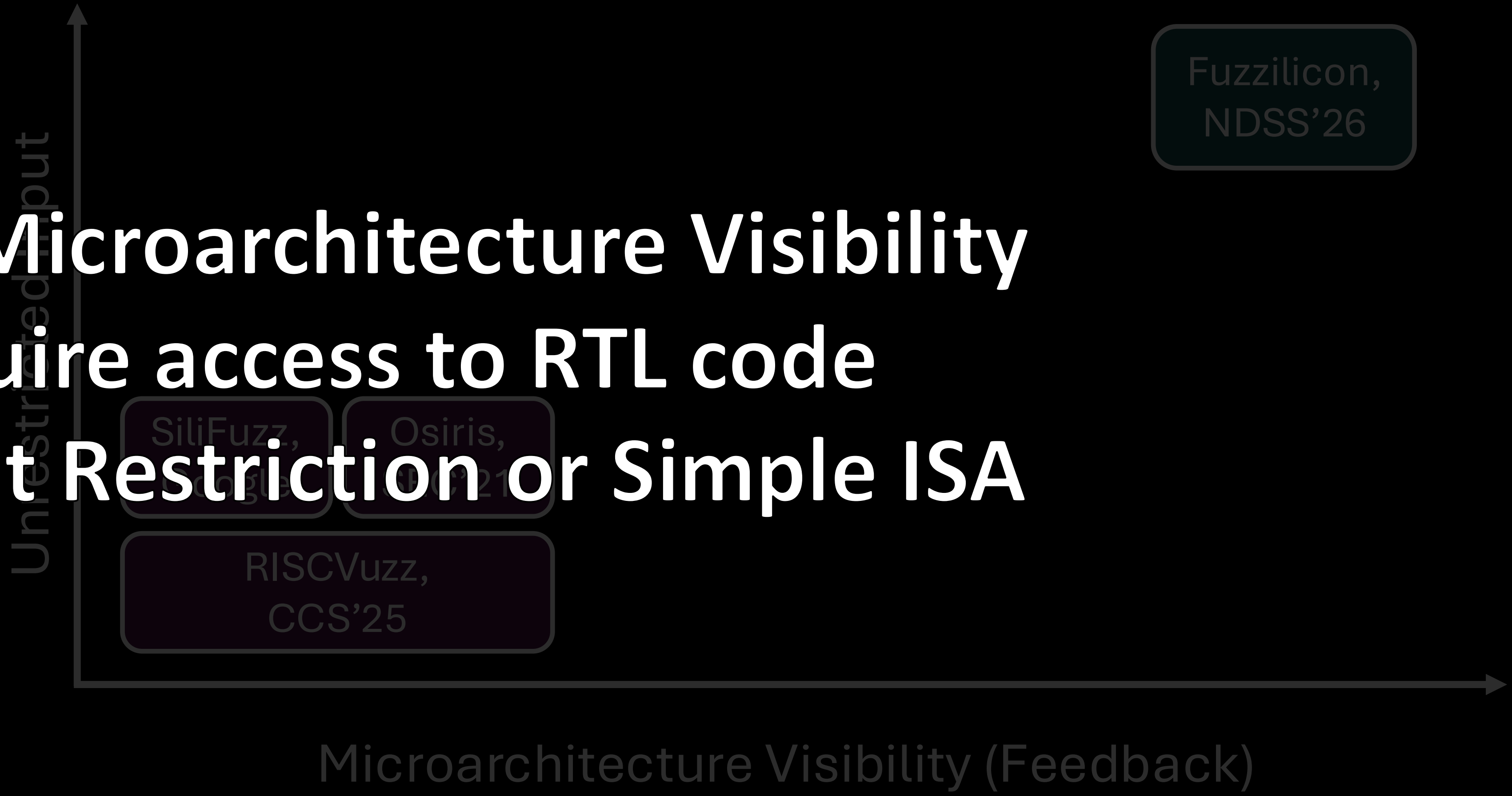
High Feedback Visibility
Low Input Diversity

# Existing CPU Fuzzing Methods



# Existing CPU Fuzzing Methods

- No Microarchitecture Visibility
- Require access to RTL code
- Input Restriction or Simple ISA



# Challenges & Contributions

- First automated post-silicon (x86) fuzzer with µArch Visibility
- Novel $\mu$Code-Level Coverage metric
- Pinpoint bugs via Serialization Oracle
- Novel $\mu$Code-Level Speculative Execution analysis
- 5 new findings on Intel CPU

# Microcode: What? and Why?

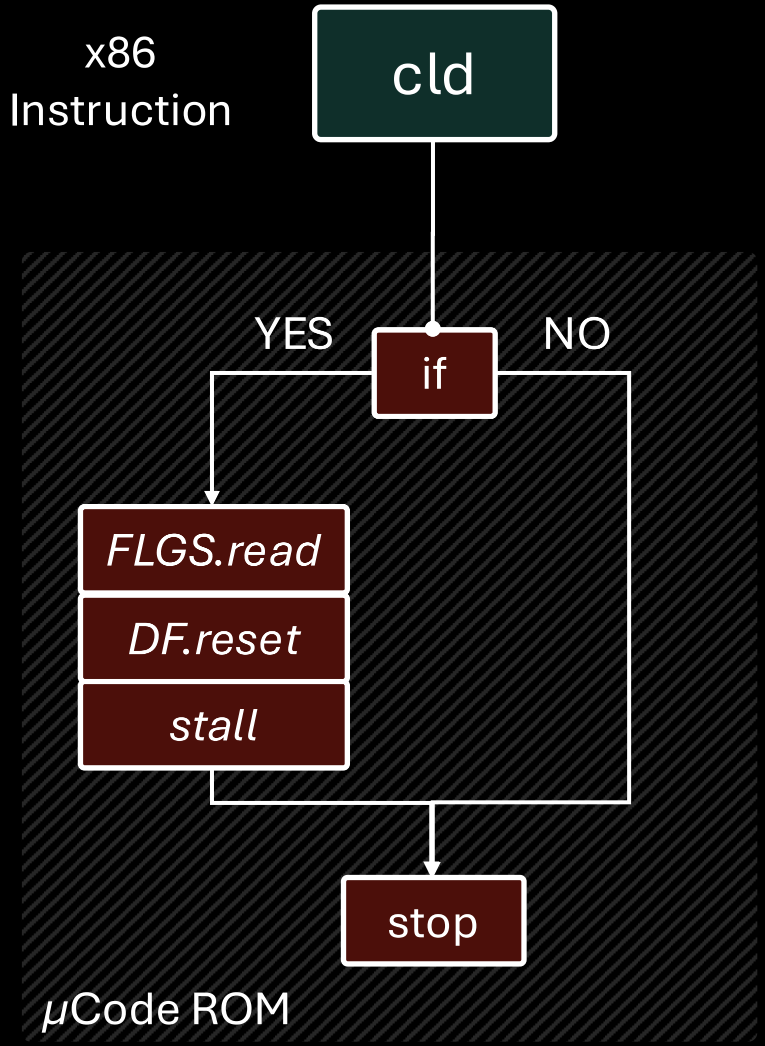
## What?

A complex instruction converts to a sequence of $\mu$ops

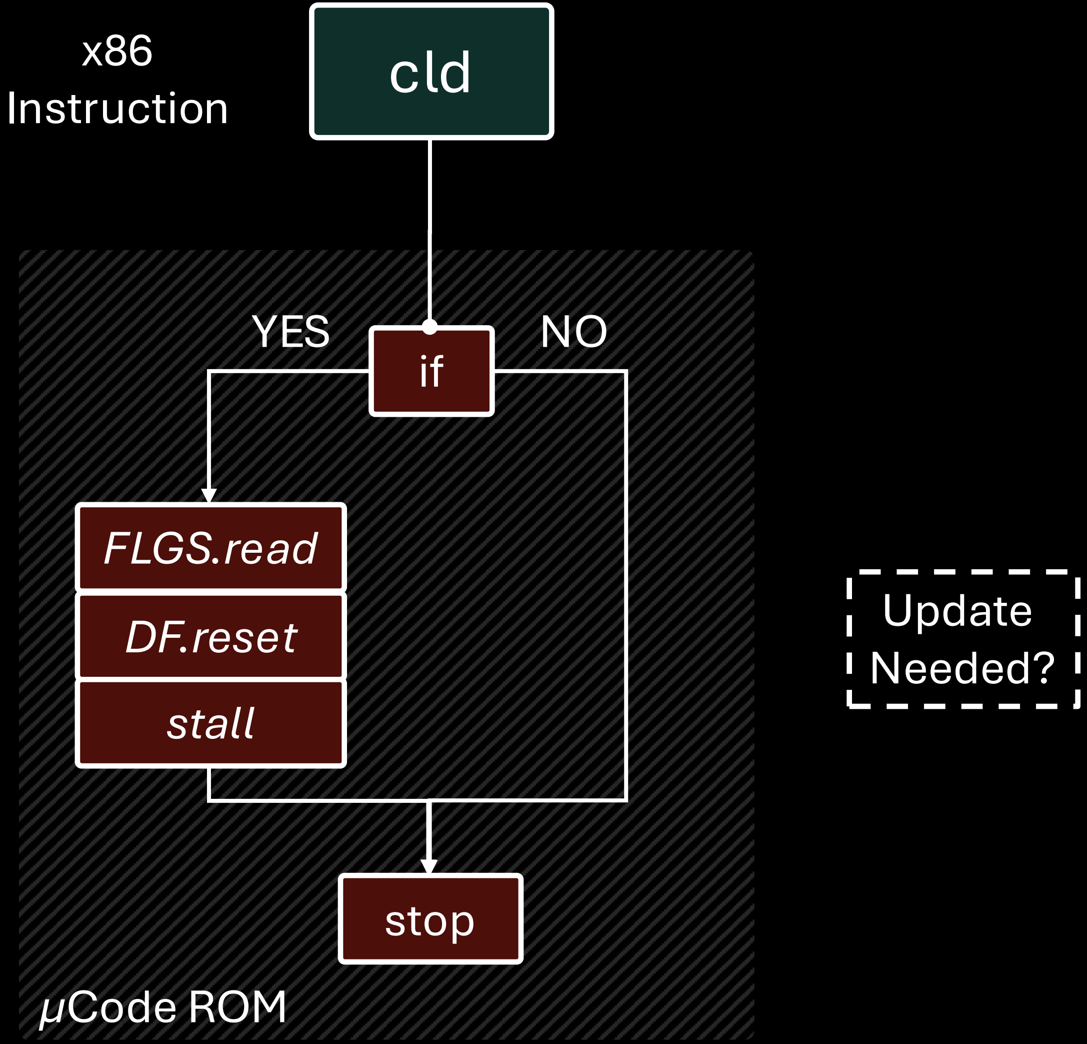
## Why?

- Reduce cost of complex hardware
- Ease the post-silicon update

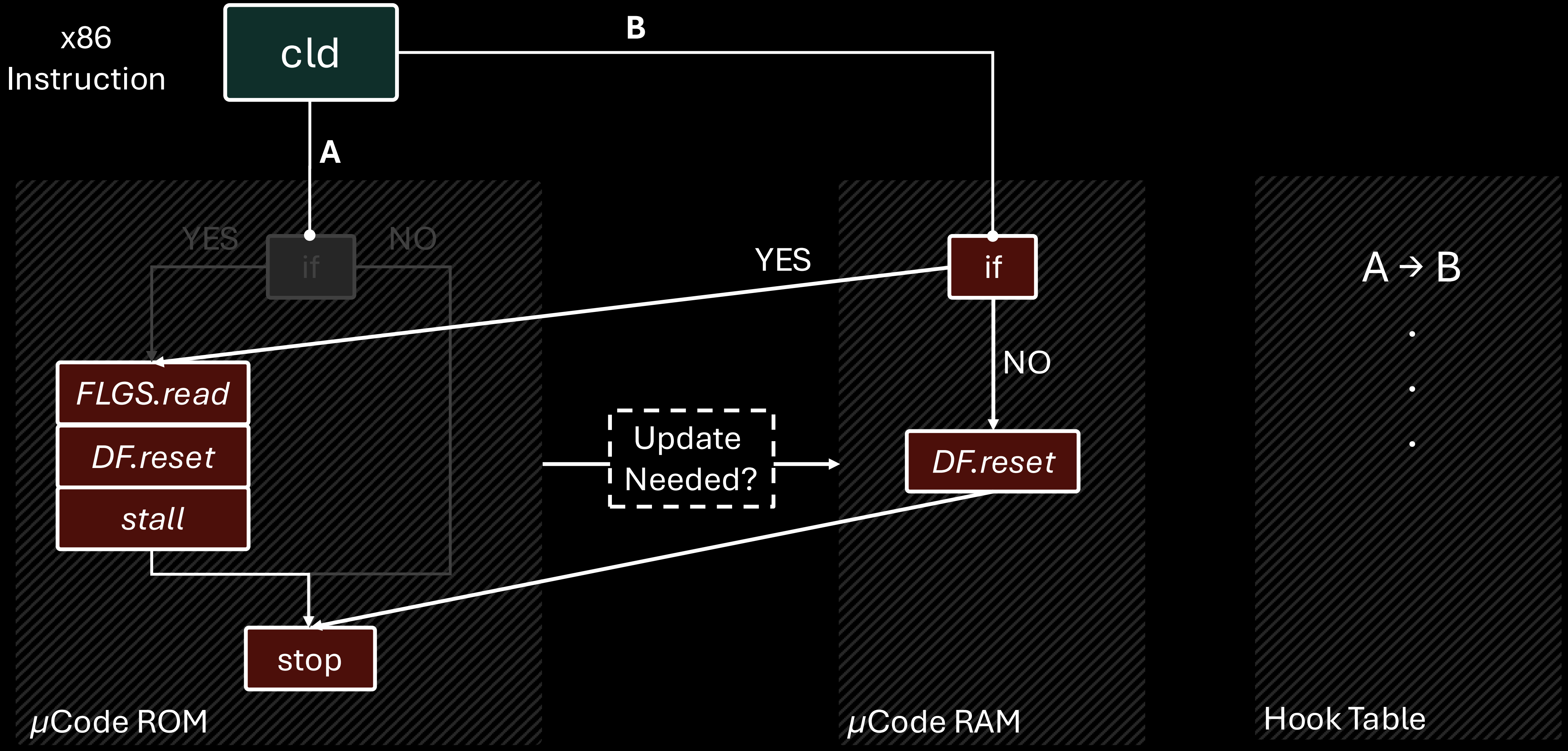
# Microcode: Structure



# Microcode: Structure

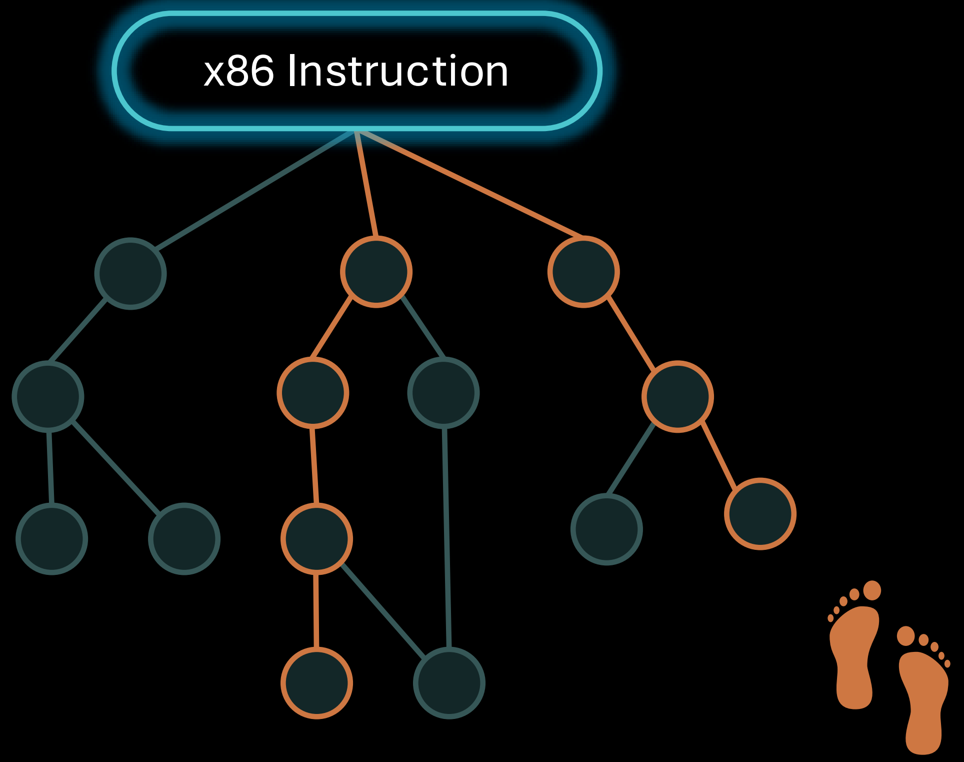


# Microcode: Structure

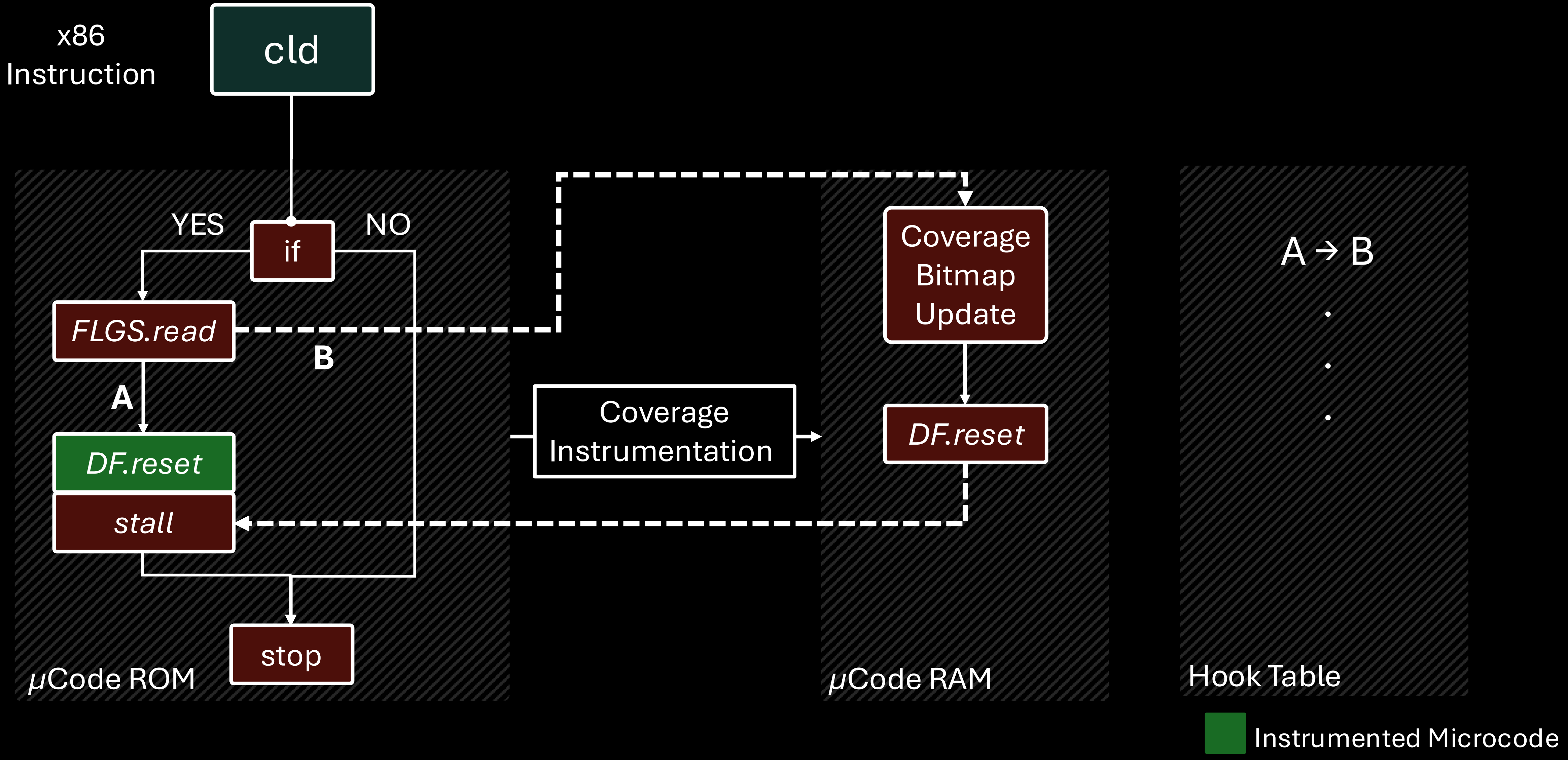


# Fuzzilicon: High-Level Idea

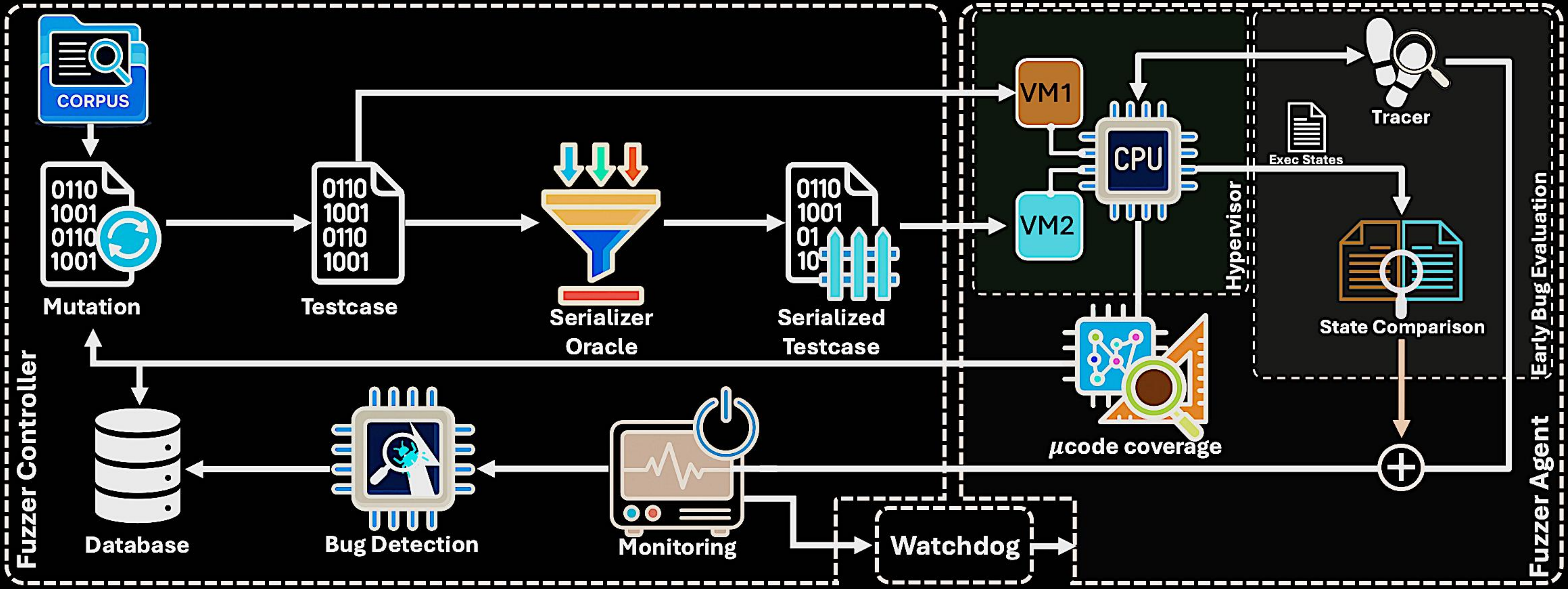
- Microcode is like a software
- Fuzz the CPU with microcode path as coverage
- More microcode coverage = More architectural feature



# Fuzzilicon : Coverage Instrumentation



# Fuzzilicon: Framework



# Fuzzilicon: Challenges

## Instrumentation Overhead

32k microcode address and only 32 hooks

## Vulnerability Detection

CPU will not crash like software, then what is the definition of a vulnerability?

## Unrestricted Input

Arbitrary fuzzing input (instructions) can effect the fuzzing process, e.g., hlt.

# Fuzzilicon: Challenges

NOTE:

Applying custom patches to CPU µcode is normally restricted to manufacturers and not accessible to end users. In our work, we exploited a vulnerability in an Intel CPU to “red-unlock” the processor, allowing us to apply customized µcode patches.

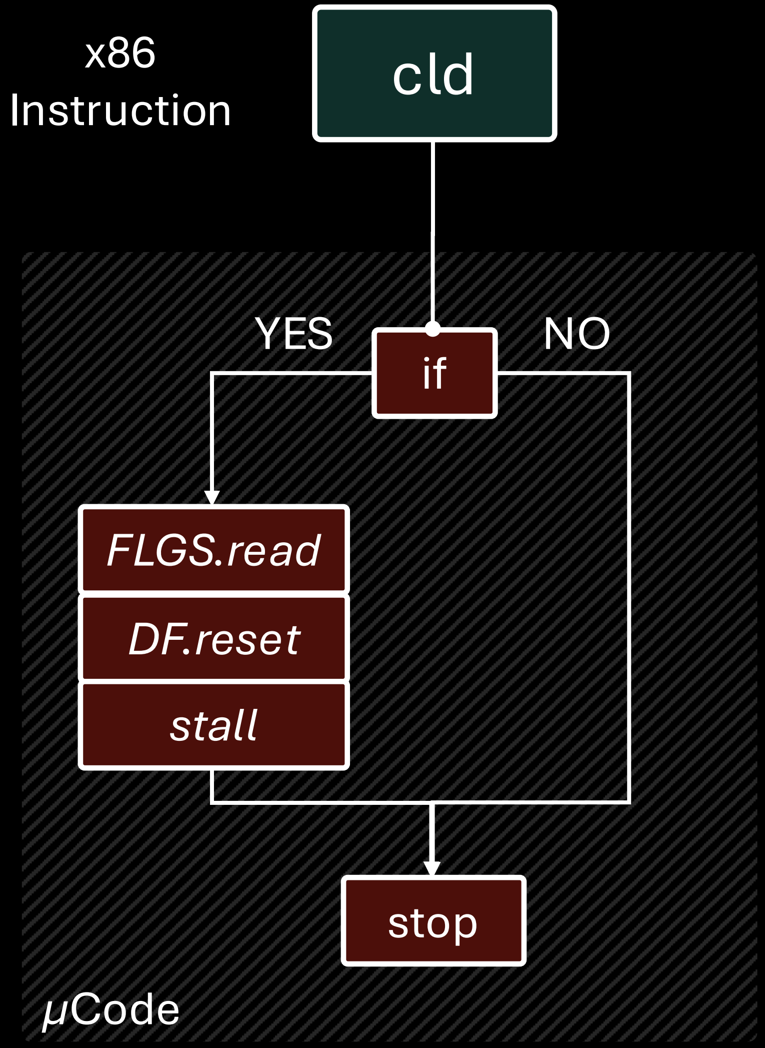
32k microcode address and only 32 hooks

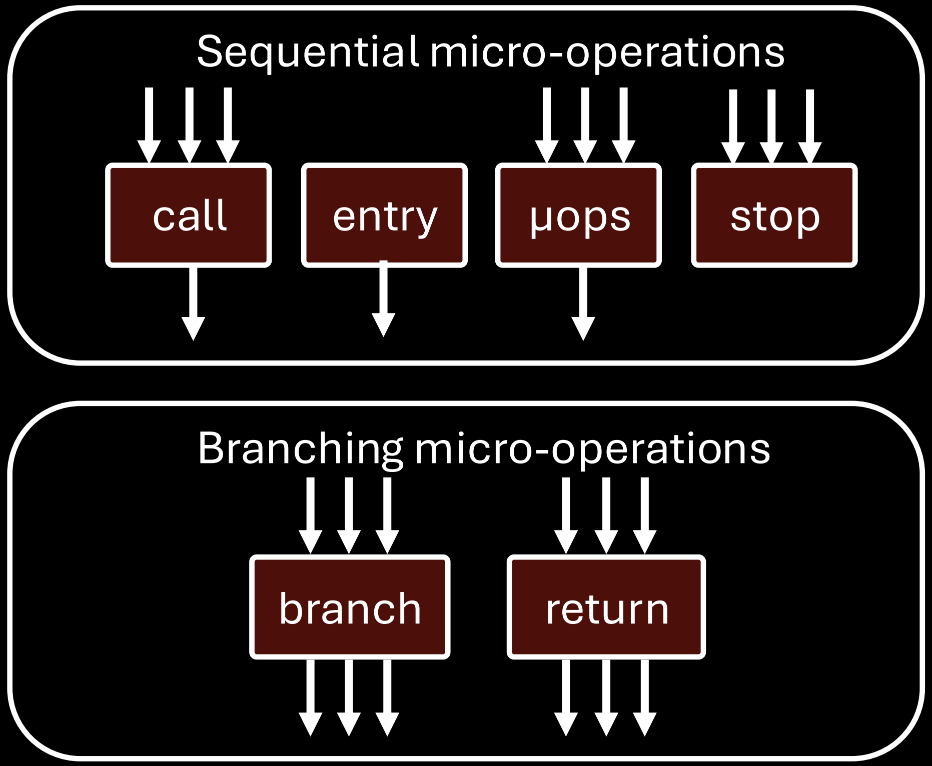
CPU will not crash like [illegible] definition of a vulnerability?

Unrestricted [illegible]

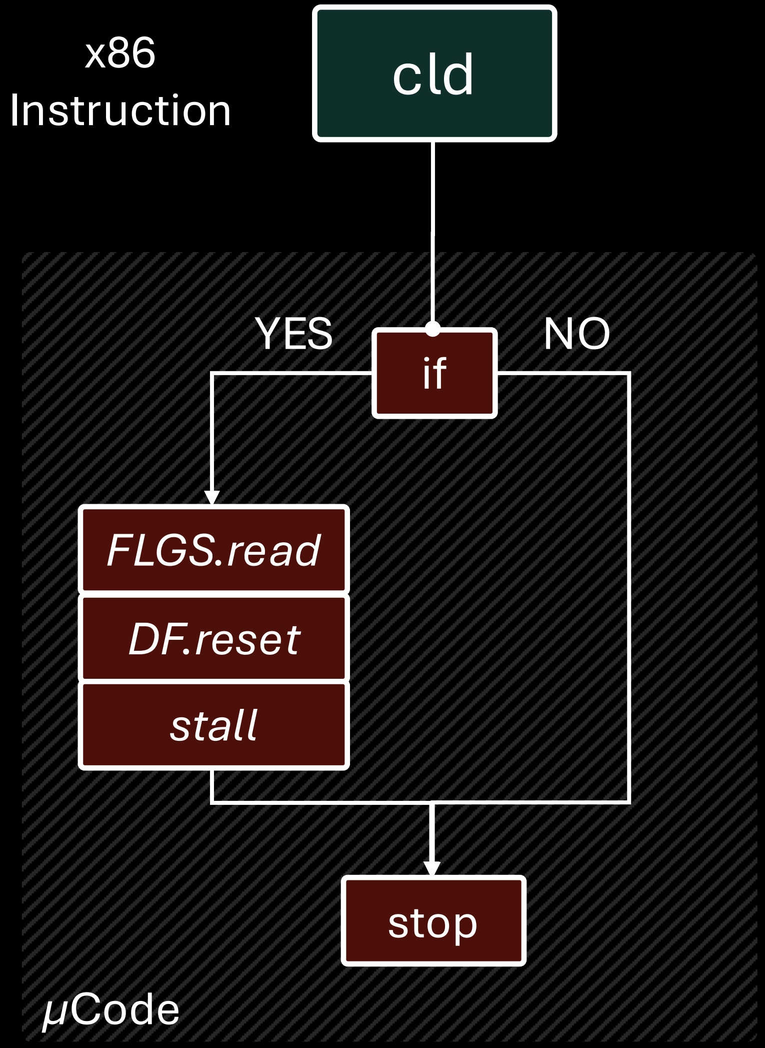
Arbitrary fuzzing input [illegible] fuzzing process, e.g., hlt.

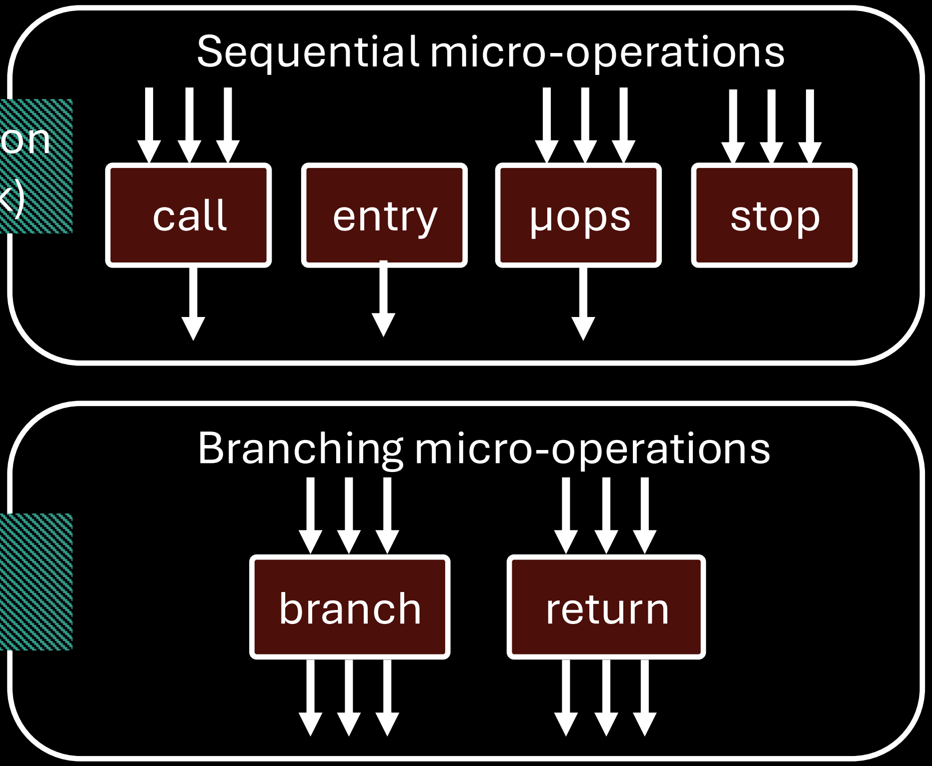
# Fuzzilicon: Instrumentation Overhead





# Fuzzilicon: Instrumentation Overhead



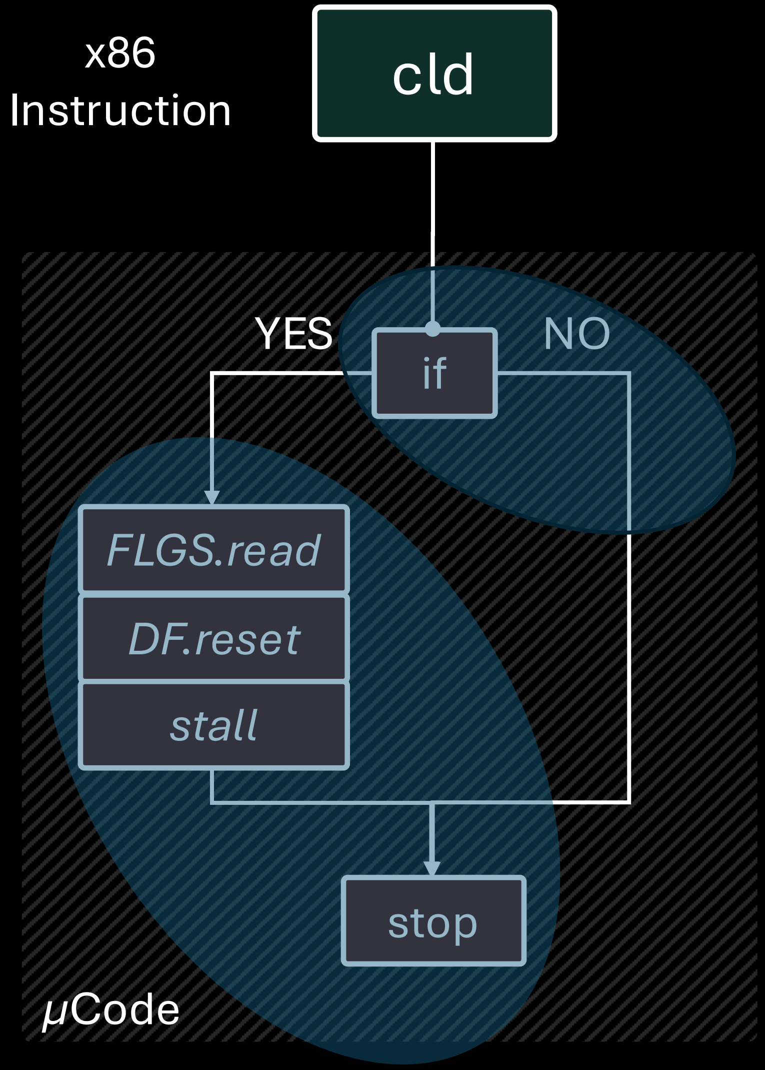


Only one instrumentation is enough (Basic Block)

Each path should be instrumented (Edge)

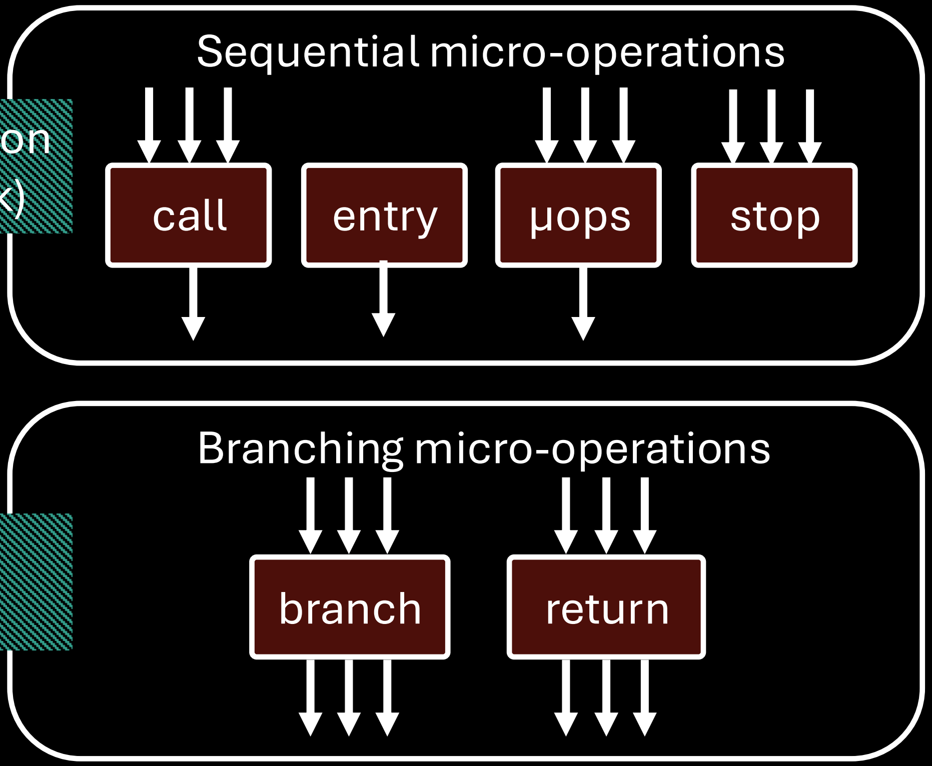
**Solution:** One time static analysis of all whole µCode ROM and extract Edges and Basic Blocks.

# Fuzzilicon: Instrumentation Overhead



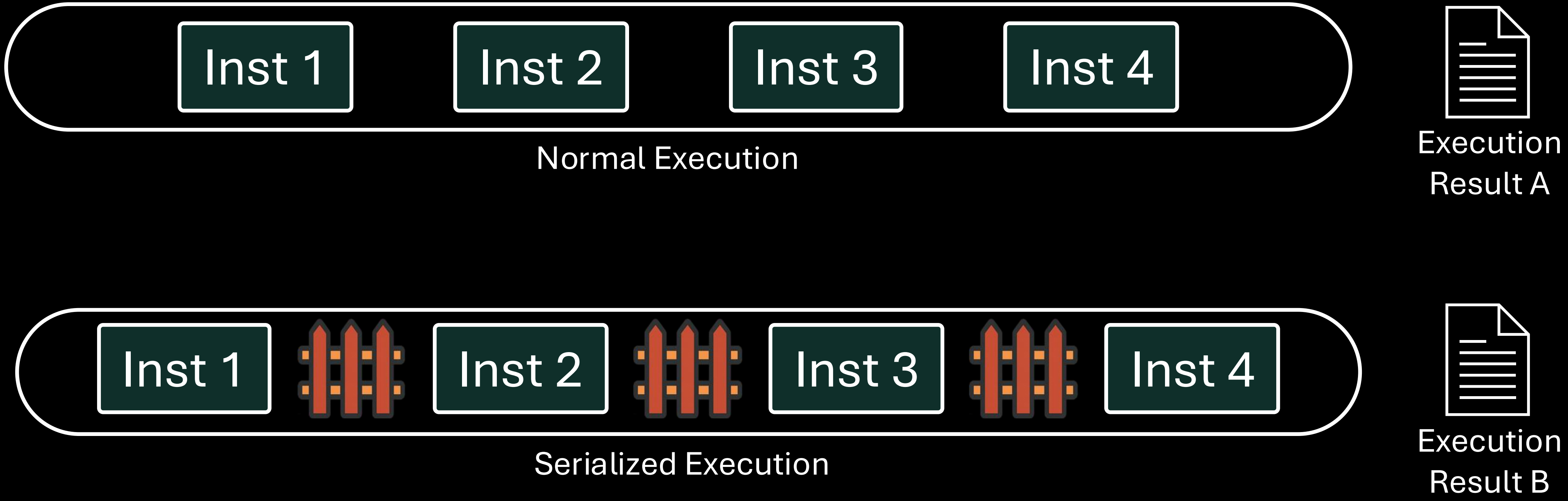
Only one instrumentation is enough (Basic Block)

Each path should be instrumented (Edge)



**Example:**
2 vs 5 (2.5x less instrumentation)

# Fuzzilicon: Vulnerability Detection



**Differential Testing:**
**If (Result A != Result B), the hardware has leaked data via speculation.**

# Fuzzilicon: Unrestricted Input

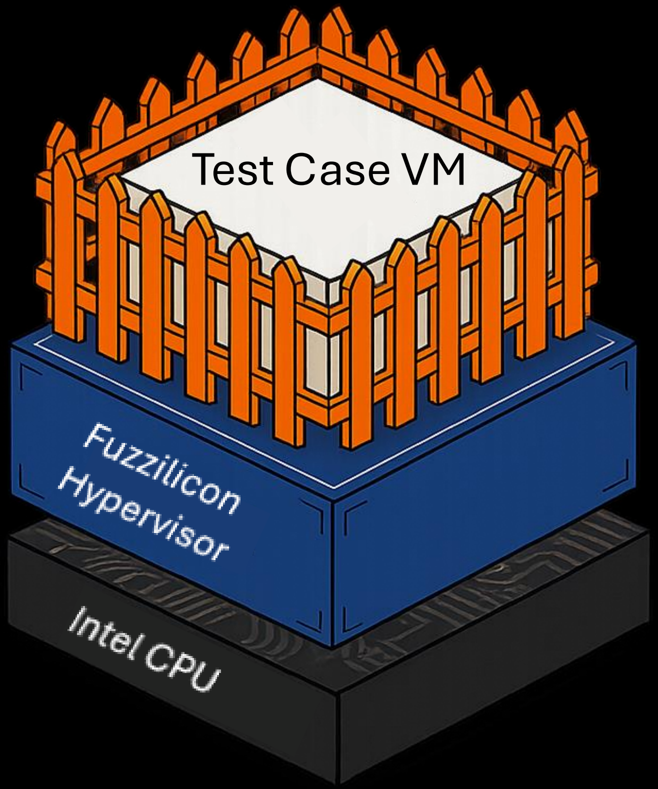
**Solution:** customized bare-metal hardware isolation between each fuzzing testcase.

- Utilization of the Intel VMX (Virtual Machine Extensions)

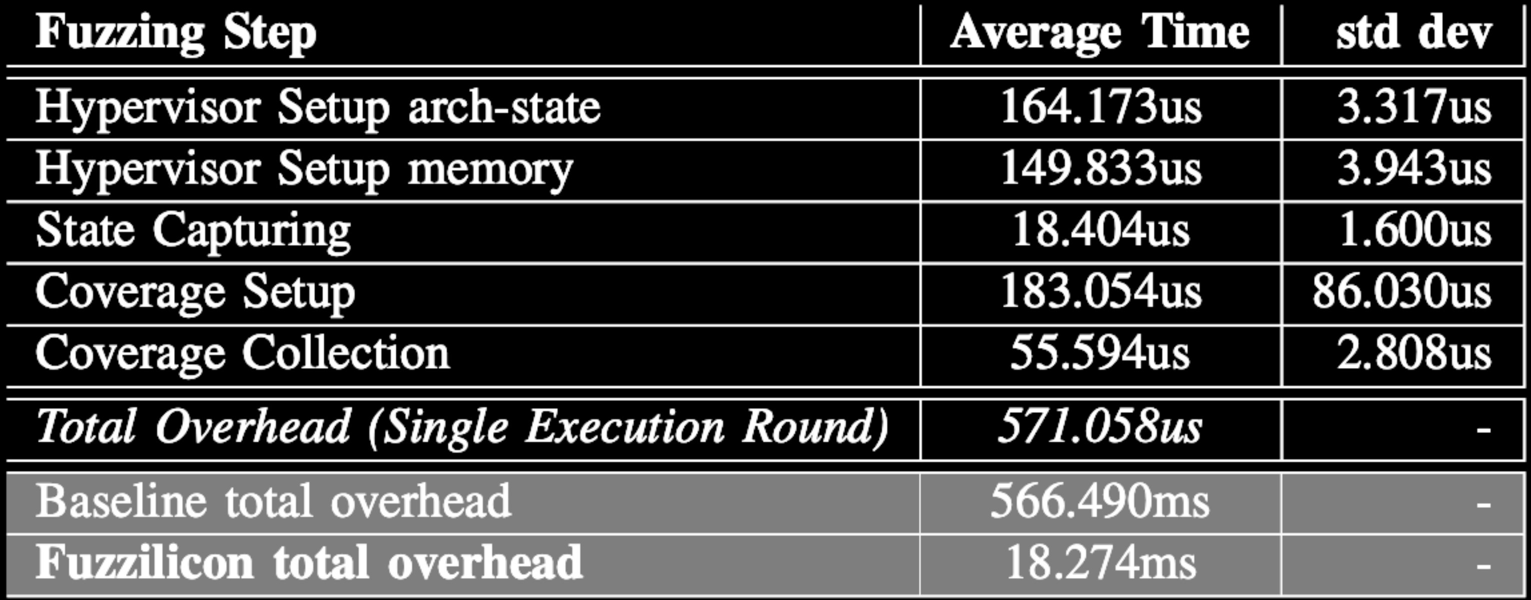
- Customized, low-overhead, and bare-metal hypervisor (per fuzzing testcase)

- State guarding: Isolation (memory, configuration, termination)

- Reproducibility: Initial state, no persistent side-effects, determinism



# Fuzzilicon: Overhead Evaluation



| **Fuzzing Step** | **Average Time** | **std dev** |
|---|---|---|
| Hypervisor Setup arch-state | 164.173us | 3.317us |
| Hypervisor Setup memory | 149.833us | 3.943us |
| State Capturing | 18.404us | 1.600us |
| Coverage Setup | 183.054us | 86.030us |
| Coverage Collection | 55.594us | 2.808us |
| *Total Overhead (Single Execution Round)* | *571.058us* | - |
| Baseline total overhead | 566.490ms | - |
| **Fuzzilicon total overhead** | 18.274ms | - |

**Result:** On average 31 times faster than baseline instrumentation method.

# Fuzzilicon: Coverage Evaluation

RQ1: How effective is µCode coverage at guiding the fuzzer to explore CPU µarch states?

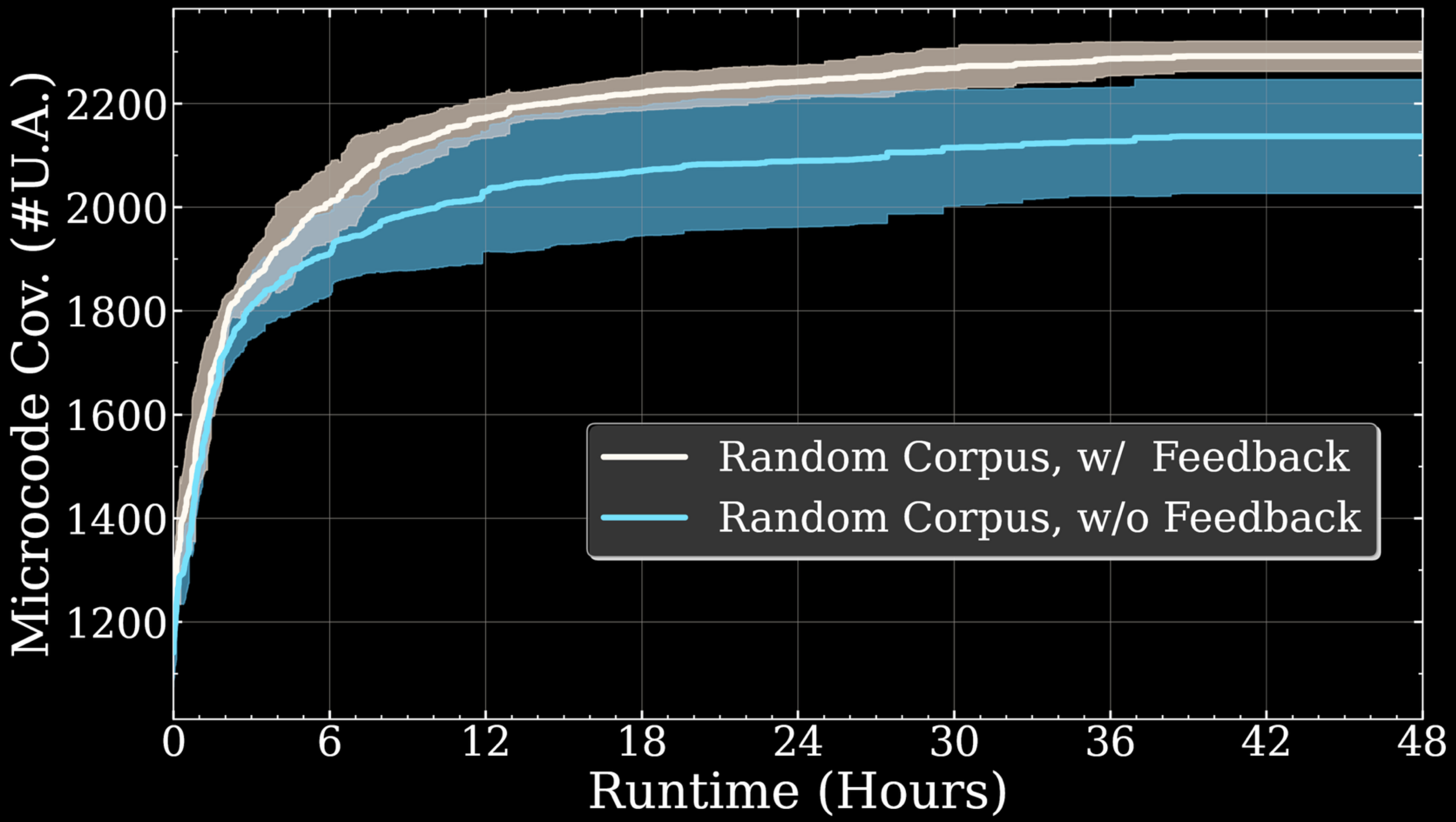
RQ2: How does initial corpus quality and randomness impact CPU exploration?

RQ3: To what extent can µCode coverage function as an independent coverage metric?

# Fuzzilicon: Coverage Evaluation

RQ1: How effective is µCode coverage at guiding the fuzzer to explore CPU µarch states?

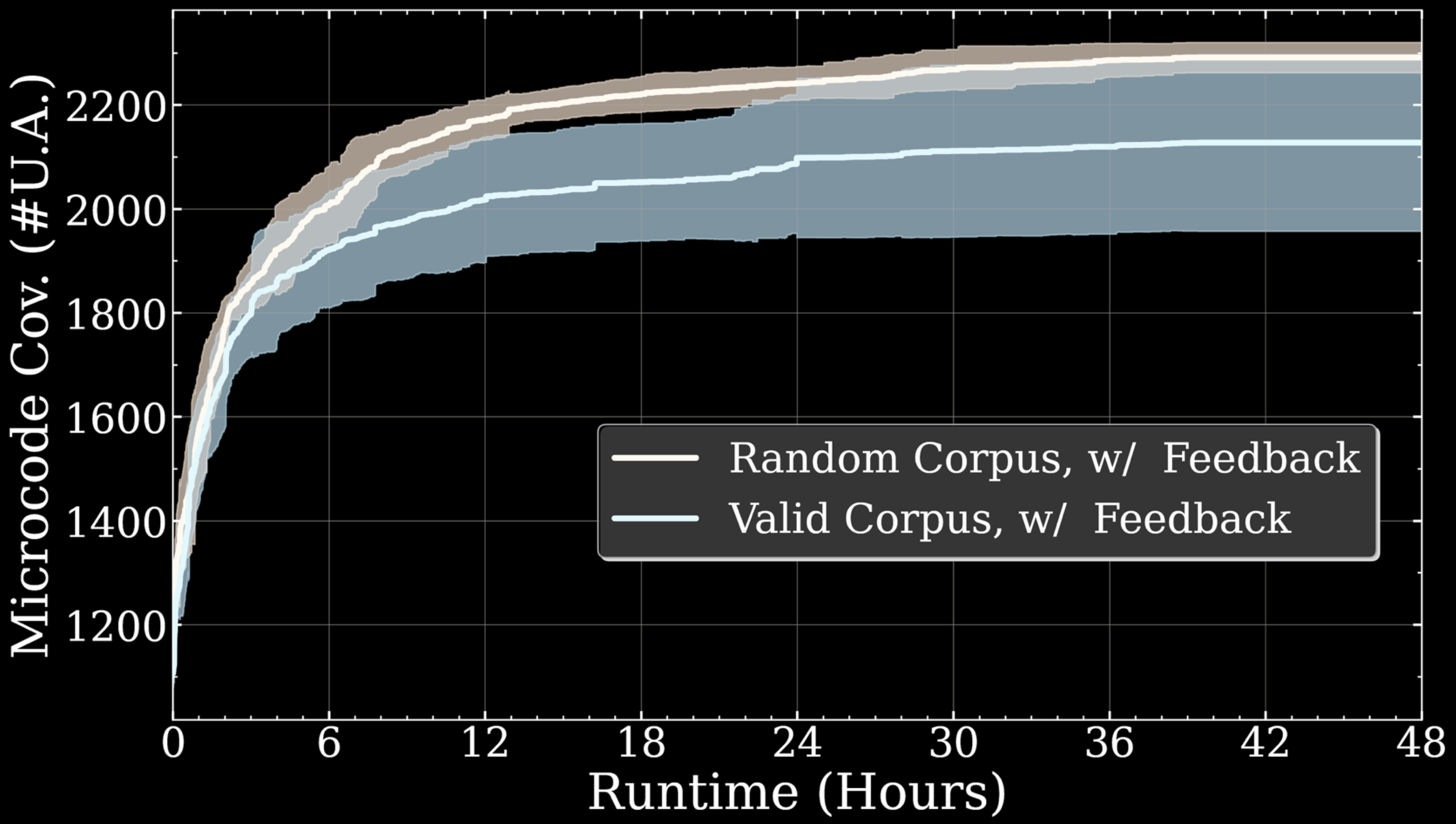
Fuzzilicon 8 times faster compare to SOTA coverage metrics.



# Fuzzilicon: Coverage Evaluation

RQ2: How does initial corpus quality and randomness impact CPU exploration?

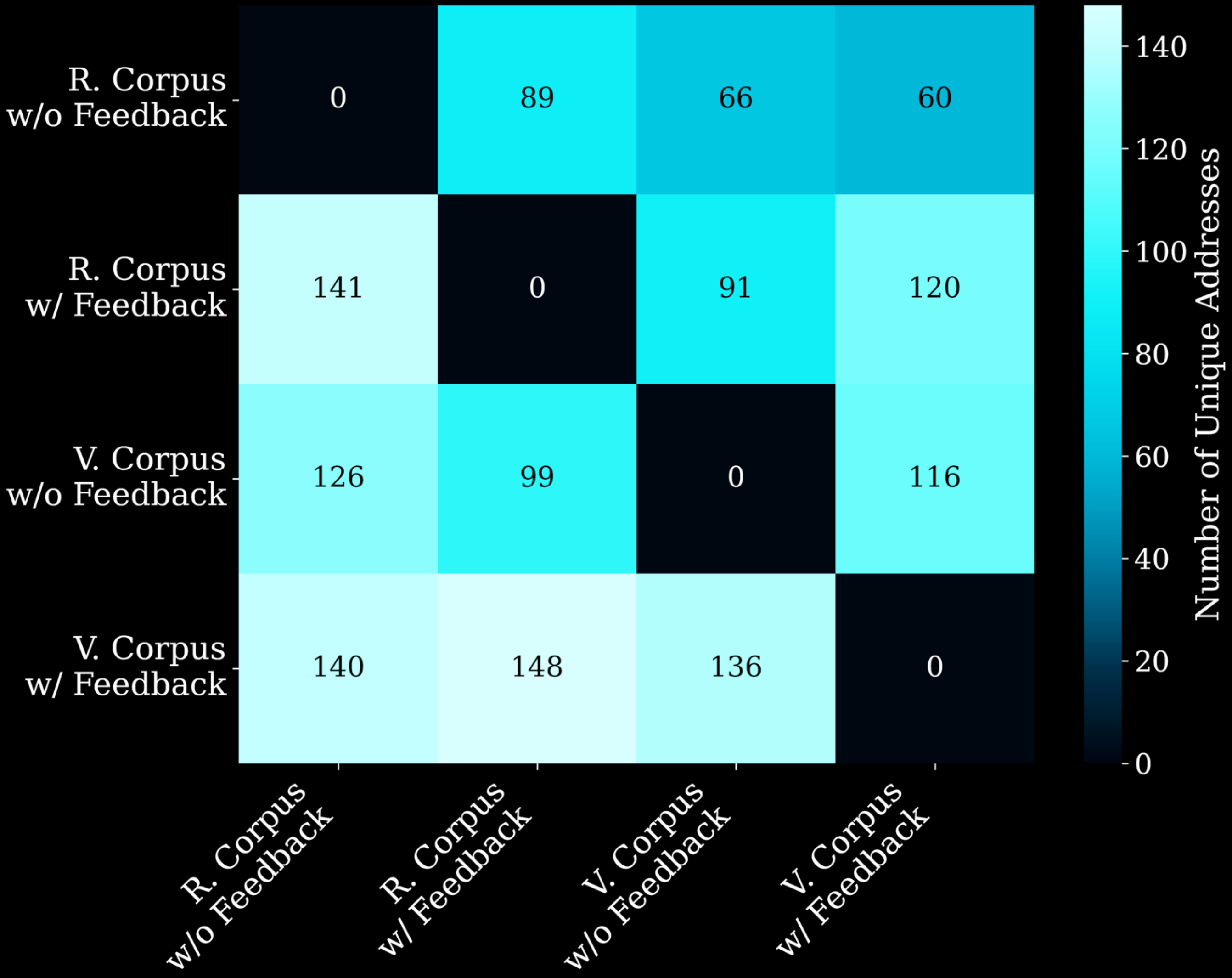
Starting with random corpora, fuzzilicon shows better performance than with valid corpora since valid corpora contain structured instructions that may exhibit redundancy across different corpus samples.



# Fuzzilicon: Coverage Evaluation

RQ3: To what extent can µCode coverage function as an independent coverage metric?

Setups using µCode-coverage feedback consistently discover more unique µCode addresses than those without feedback.



# Fuzzilicon: Findings

Automated
Rediscovery

Novel
Finding (F2)

Novel
Finding (F3)

Novel
Finding (F4)

Novel
Finding (F5)

# Fuzzilicon: Findings

Automated Rediscovery

Novel Finding (F2)

Novel Finding (F3)

Novel Finding (F4)

Novel Finding (F5)

- Fuzzilicon automatically discovered the µSpectre vulnerability class during fuzzing
- Detection required no prior knowledge and no changes to the test programs

# Fuzzilicon: Findings

Automated Rediscovery

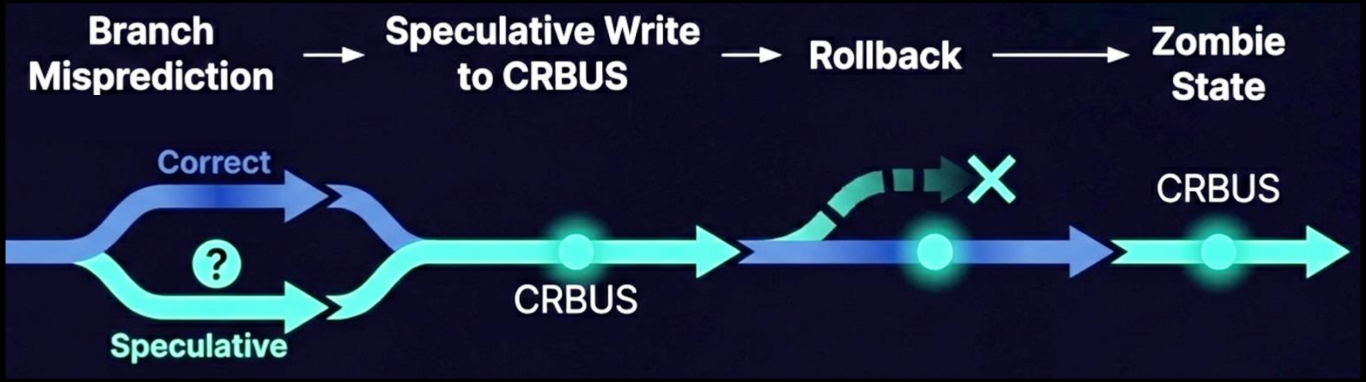
Novel Finding (F2)

Novel Finding (F3)

Novel Finding (F4)

Novel Finding (F5)

- Speculative writes to the CRBUS persist after rollback
- Attacker can disable security patches by speculatively executing 'MOVETOCREG' µOp



# Fuzzilicon: Findings

Automated Rediscovery

Novel Finding (F2)

Novel Finding (F3)

Novel Finding (F4)

Novel Finding (F5)

- Fuzzilicon discovered a flaw where speculative µcode writes to segment selector caches persist after misprediction (F3).
- This can lead to crashes or privilege escalation.

# Fuzzilicon: Findings

Automated Rediscovery

Novel Finding (F2)

Novel Finding (F3)

Novel Finding (F4)

Novel Finding (F5)

- Fuzzilicon found that µcode-implemented instructions stop further speculation (F4).
- Dispatch to the µcode sequencer acts as an implicit speculation barrier.
- This creates timing differences that can leak execution-path information.

# Fuzzilicon: Findings

Automated Rediscovery

Novel Finding (F2)

Novel Finding (F3)

Novel Finding (F4)

Novel Finding (F5)

- Fuzzilicon observed that speculative μops leave measurable traces in performance counters (F5).
- Counters can change even when the speculative path is later discarded.

# Q&A

Thank you for your attention

Download the code

Download the paper