RIFF: Reduced Instruction Footprint for Coverage-Guided Fuzzing

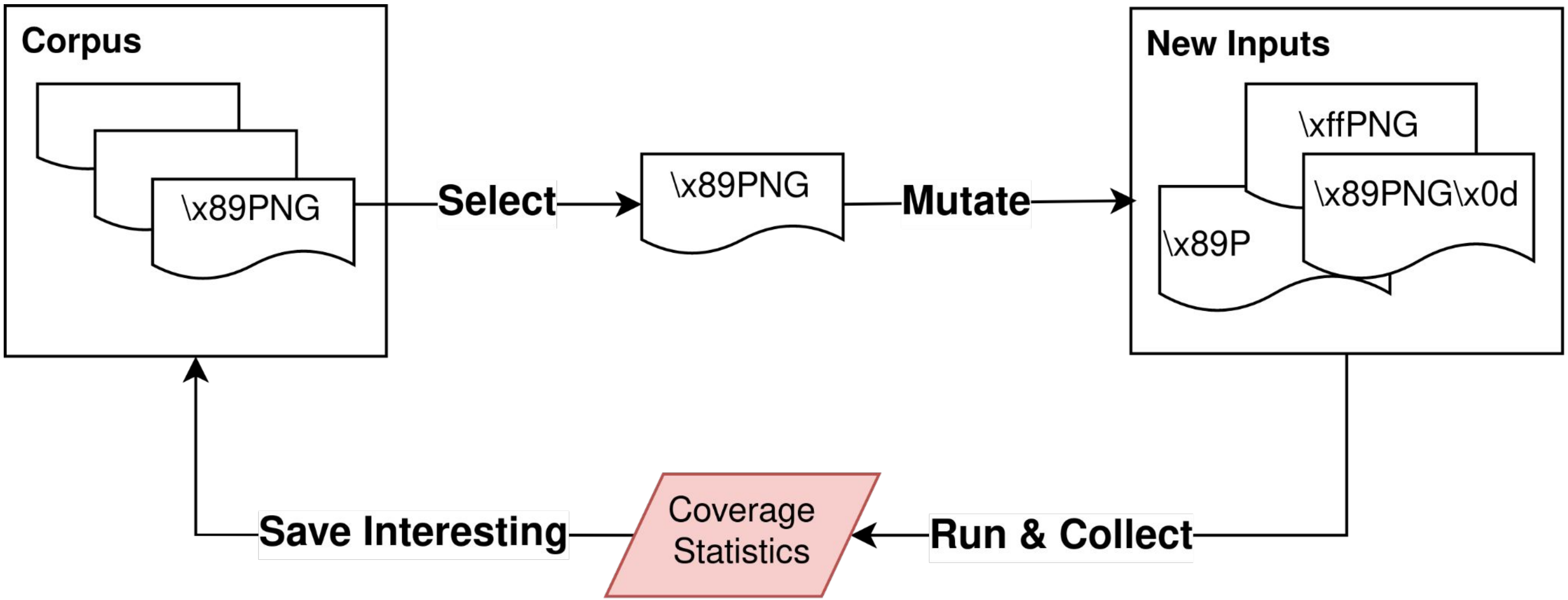
Mingzhe Wang, Jie Liang, Chijin Zhou, Yu Jiang, Rui Wang[2], Chengnian Sun[3], and Jiaguang Sun

Tsinghua University

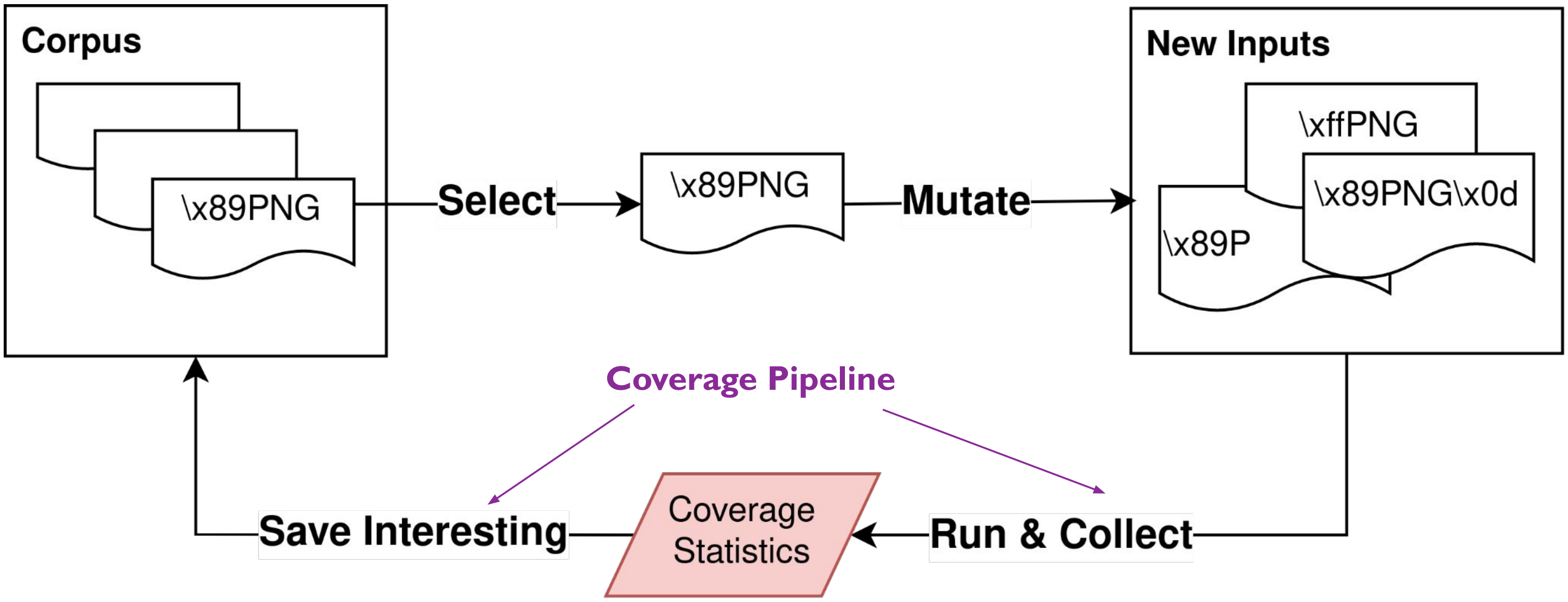
[2] Capital Normal University

[3] Waterloo University

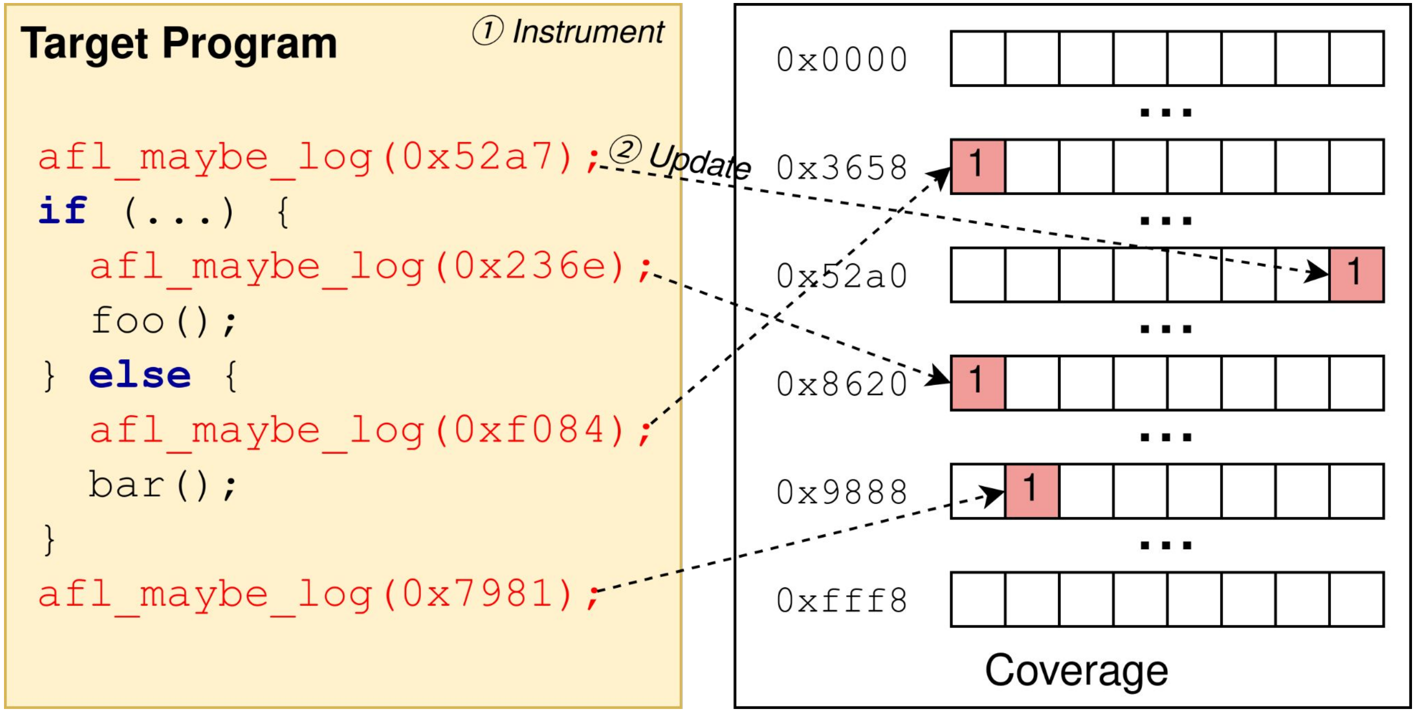
Coverage is Important for Guided Fuzzing



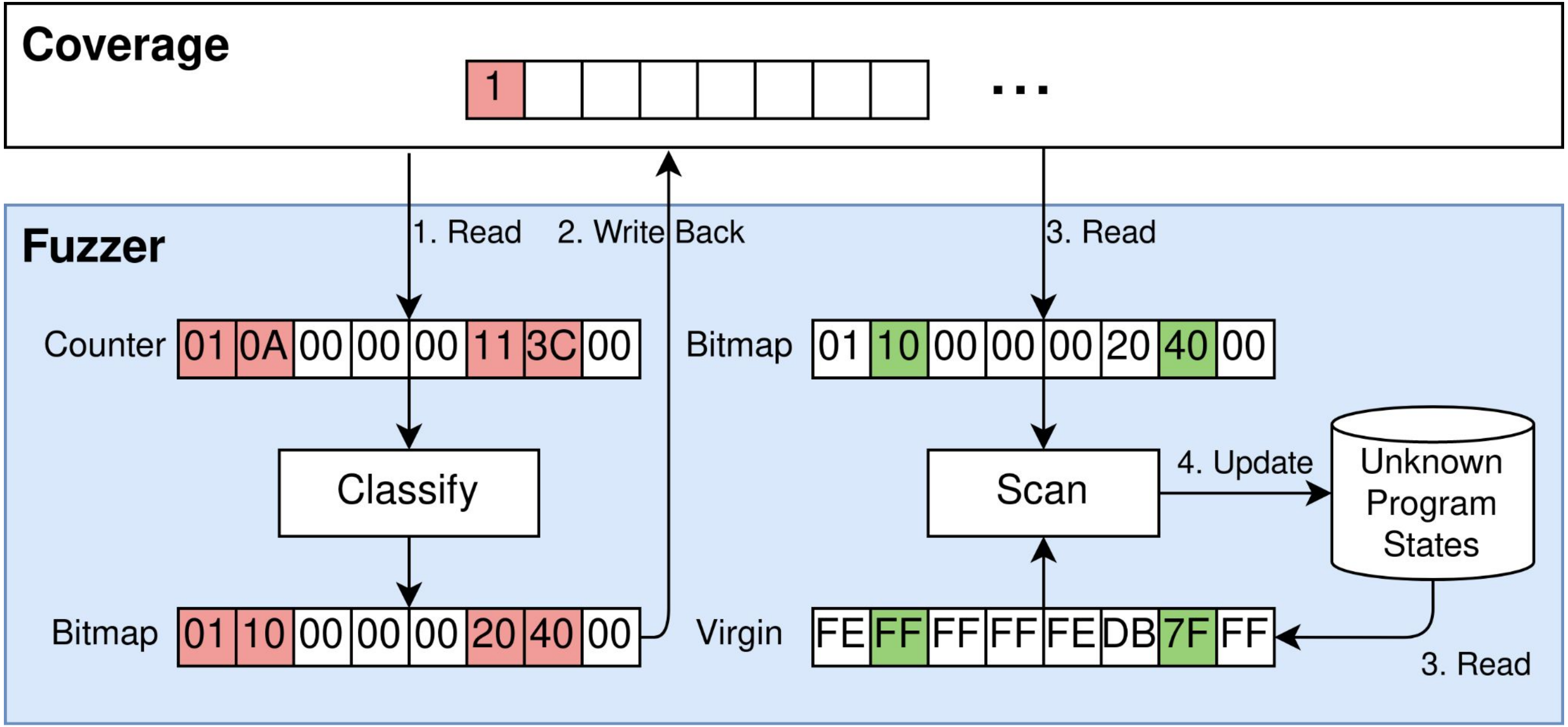
Coverage Pipeline in Fuzzers



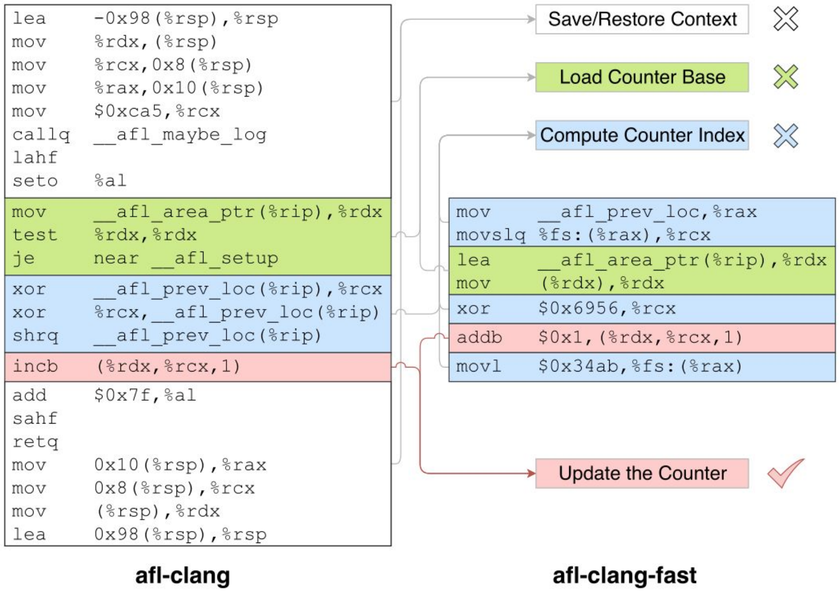
Example: Coverage Collection in AFL



Example: Coverage Analysis in AFL



Overhead in Coverage Collection



Method	Duration	Instructions	L1-I	L1-D	μops
afl-clang	3.50x	4.26x	102.36x	5.16x	4.72x
afl-fuzzbench	2.45x	2.83x	19.88x	2.53x	2.14x
afl-clang-fast	1.69x	1.79x	33.58x	2.88x	2.11x

Overhead in Coverage Analysis

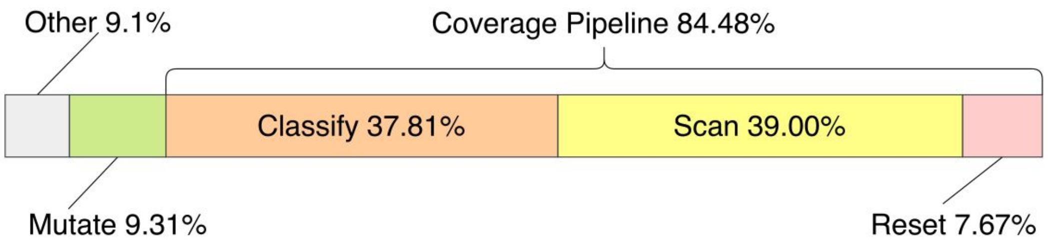
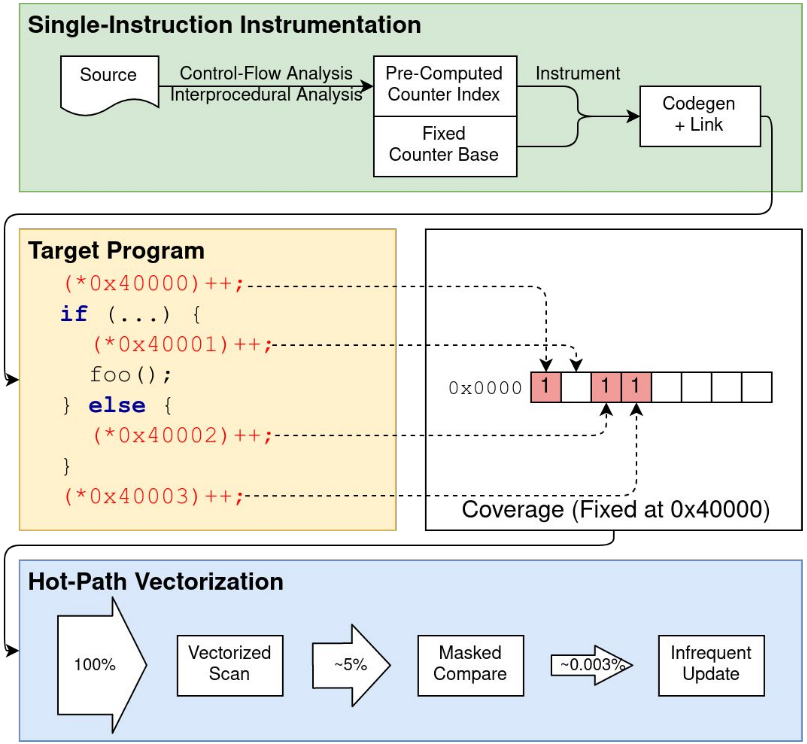


Table 4: Number of Processed Counters and Executions

	Total	Useless	Proportion
Counter	65,536	64,664.37	98.67%
Execution	67,696	67,694	99.997%

RIFF: Overview and Insights



Target Program

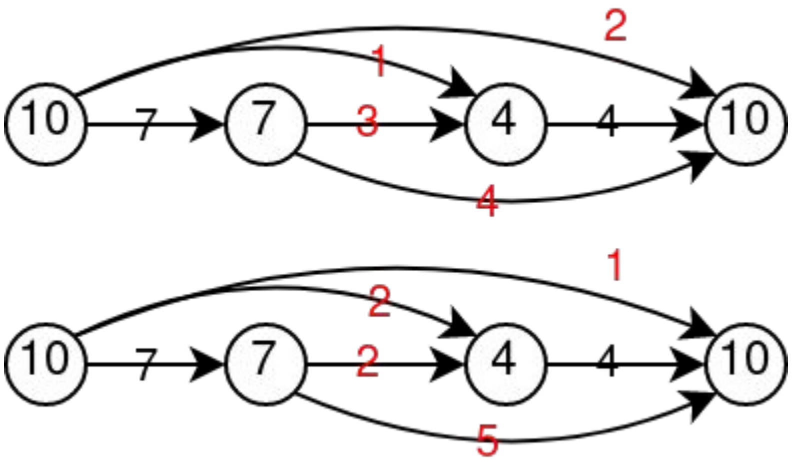
Move run-time computation to compile-time if possible

Fuzzer

Add hot-path processing logic specially tuned for simple cases

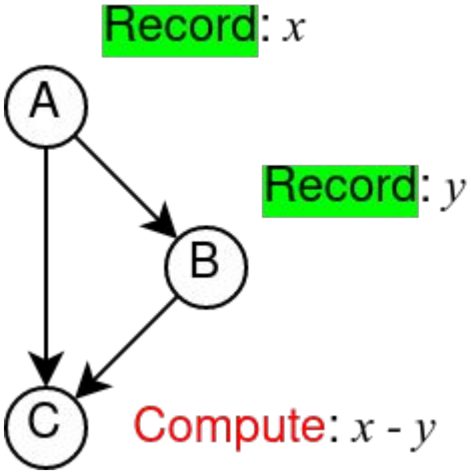
Single-Instruction Instrumentation: Problem of Block Coverage

Block coverage is intuitive but *incomplete*: multiple edge counts map to one block count.



Single-Instruction Instrumentation: Problem of Block Coverage

Block coverage is intuitive but *complex*: requires extra computation at fuzzer's side.



Single-Instruction Instrumentation: Simplified Algorithm

```
for each potential control transfer E in program P:
    if E is direct control transfer:
        if basic block of E.source must transfer to basic block of E.target:
            InstrumentBlock(basic block of E.source)
        else if basic block of E.target must transfer from basic block of E.source:
            InstrumentBlock(basic block of E.target)
        else:
            InstrumentBlock(CreateDummyBlock(E))
    else:
        (Handle indirect control transfer, see the next slide)
```

```
# Single-instruction instrumentation
incb    $INDEX(%rip)        # fe 05 ?? ?? ?? ??
```

Single-Instruction Instrumentation: Simplified Algorithm

`for each` potential control transfer E in program P:

    `if` E is direct control transfer:

        (See the previous slide)

    `else:`

        InstrumentBefore(E.source, SetSourceID)

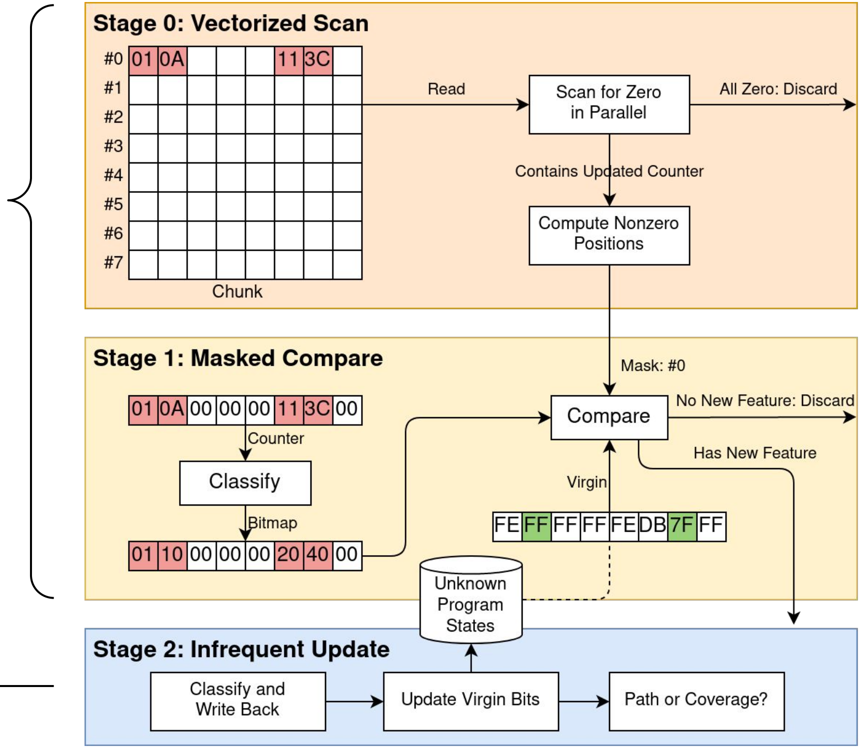
        InstrumentAfter(E.target, LogEdgeTransfer)

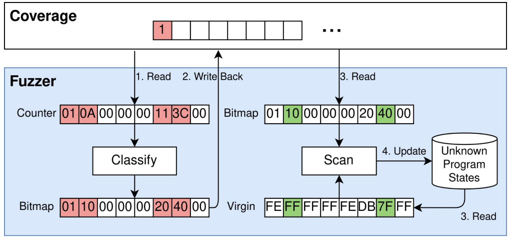
```
# Rare case: indirect transfer (source)
mov     $PREV(%rip),%rcx   # 48 8b 0d ?? ?? ?? ??
movl    $BBID,%fs:(%rcx)   # 64 c7 01 ?? ?? ?? ??
```

```
# Rare case: indirect transfer (destination)
mov     $PREV(%rip),%rcx   # 48 8b 0d ?? ?? ?? ??
movslq  %fs:(%rcx),%rax    # 64 48 63 01
xor     $BBID,%rax         # 48 35 ?? ?? ?? ??
incb    $BASE(%rax)        # fe 80 ?? ?? ?? ??
```

Hot-Path Vectorized Analysis

Newly-Introduced
Hot Path (99.997%)

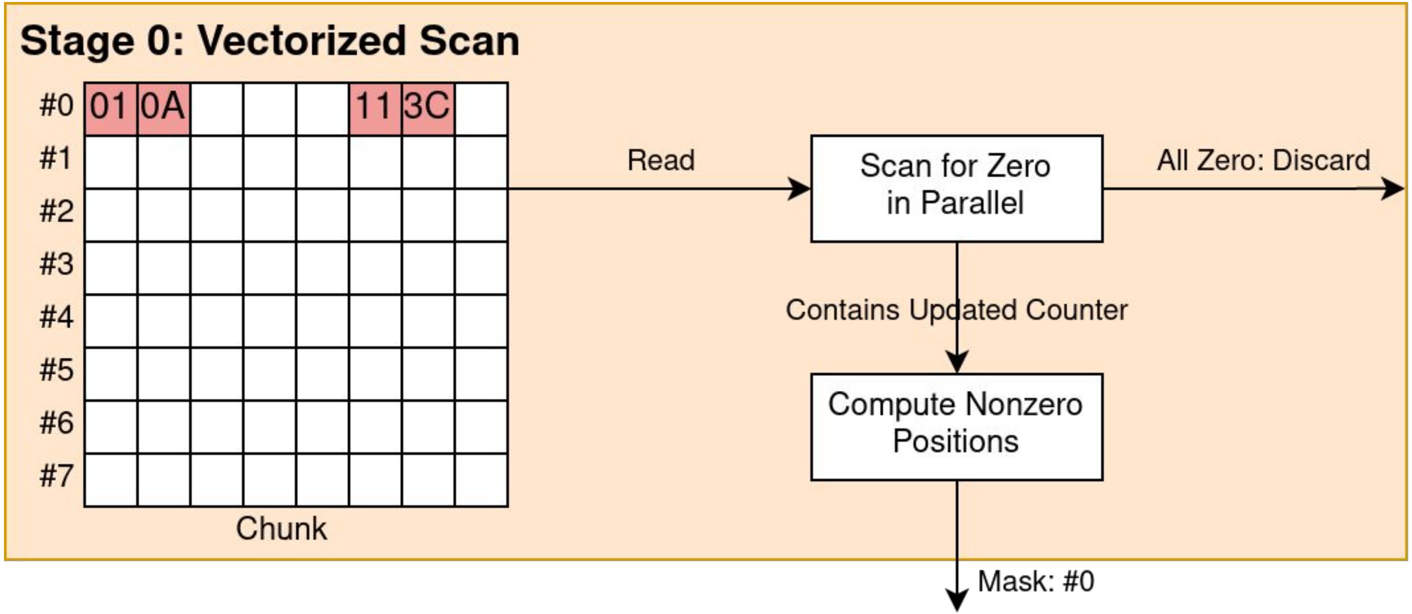




Original Algorithm
Cold Path (0.003%)

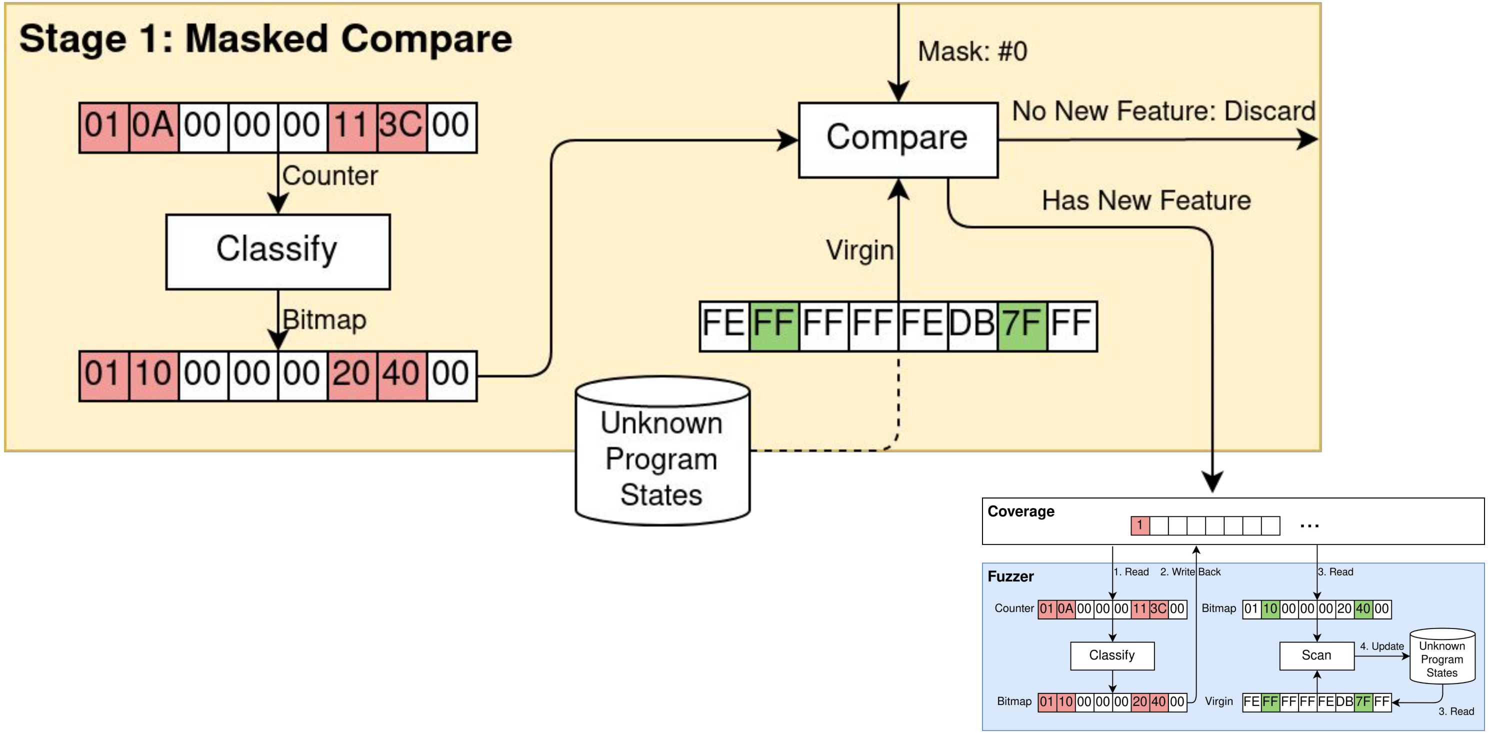
Hot-Path Vectorized Analysis

Stage 0: optimized for useless *counters*



Hot-Path Vectorized Analysis

Stage 1: optimized for useless *executions*



Evaluation: Overall Speedup in Fuzzing

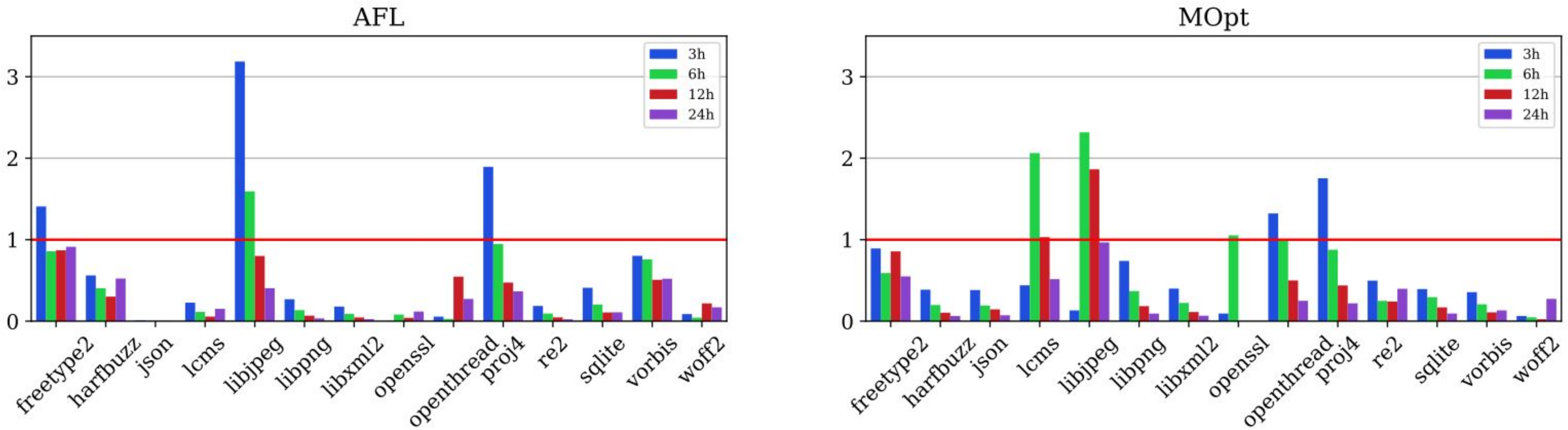


Figure 7: Normalized execution time required by RIFF to reach the same coverage as AFL and MOpt. The X axis is programs, the Y axis is the ratio between the execution times required for reaching the same coverage. A bar below the red line indicates a speed-up.

Improved Performance Brought by Speedup

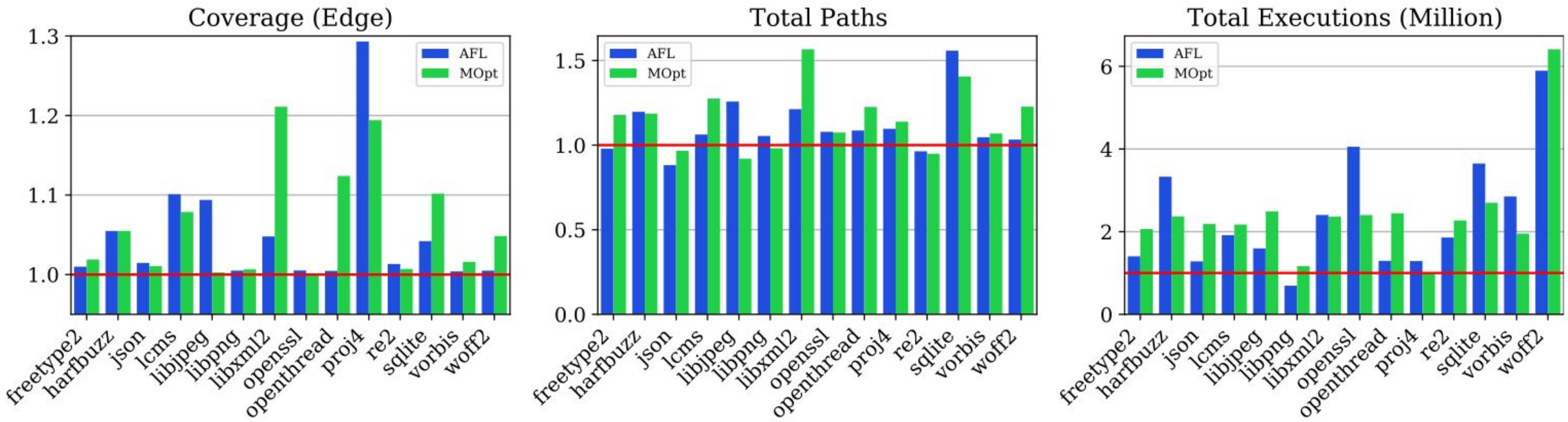


Figure 8: Normalized performance metrics for RIFF-based fuzzers after 24 hours of fuzzing. X axis is programs, Y axis is the normalized performance metric (ratio between RIFF and standard fuzzer). Bars higher than 1 (red line) indicate better performance.

Speedup in Coverage Collection and Analysis

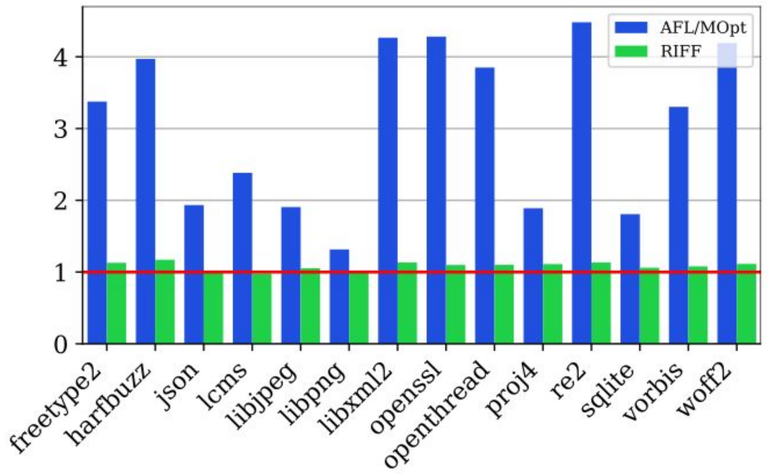


Figure 10: Normalized execution duration of fuzzed programs: time to execute 1000 on fixed inputs normalized to the time of uninstrumented programs. Lower bars indicate better performance.

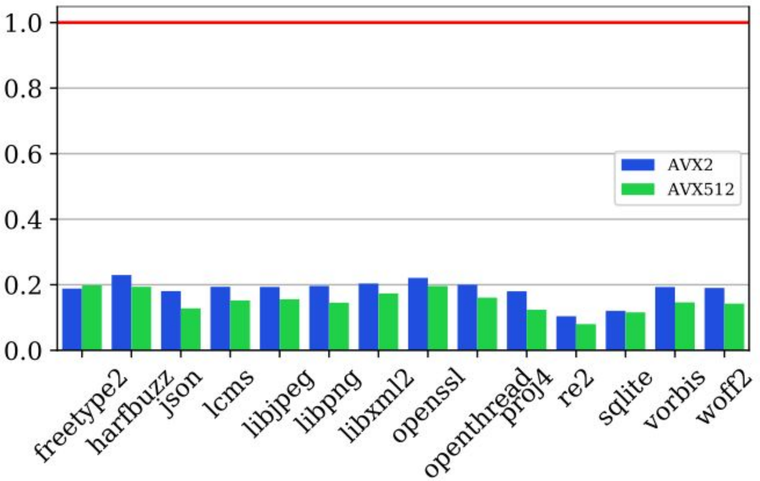


Figure 11: Coverage processing time (normalized against the baseline algorithm). Lower bars indicate better performance.

Summary

Observation	Design	Implementation
1. Coverage collection and analysis significantly affect the speed of fuzzing.	1. Accelerate coverage collection with single instruction instrumentation.	1. Adapt RIFF to popular fuzzers, including AFL and MOpt.
2. We break down the cost of instrumentation and analysis code.	2. Accelerate coverage analysis with hot-path vectorization.	2. Integrated into AFL++.

Thank You

Q & A