Full-speed Fuzzing: Reducing Fuzzing Overhead through Coverage-guided Tracing

Stefan Nagy
snagy2@vt.edu

Matthew Hicks
mdhicks2@vt.edu

COMPUTER SCIENCE
VIRGINIA TECH

Fuzzing

An Overview of Fuzzing

Time-tested technique
- AFL, honggFuzz, libFuzzer
- **CVE**'s galore

Popular in the industry
- Google, Microsoft

Fuzzing platforms
- MSRD, OSS-Fuzz, FuzzBuzz, FuzzIt

IJG jpeg 1	libjpeg-turbo 1 2
libtiff 1 2 3 4 5	mozjpeg 1
Mozilla Firefox 1 2 3 4	Internet Explorer 1 2 3 4
Adobe Flash / PCRE 1 2 3 4 5 6 7	sqlite 1 2 3 4...
LibreOffice 1 2 3 4	poppler 1 2...
GnuTLS 1	GnuPG 1 2 3 4
PuTTY 1 2	ntpd 1 2
bash (post-Shellshock) 1 2	tcpdump 1 2 3 4 5 6 7 8 9
pdfium 1 2	ffmpeg 1 2 3 4 5

Source: lcamtuf.coredump.cx/afl

Most popular: **coverage-guided fuzzing**

Coverage-guided Fuzzing

- (<< ***N***) ~~(*N*)~~ test cases
- **0.3%** ~~36–612%~~ overhead
- **Orthogonal** to tracing, generation

COMPUTER SCIENCE
VIRGINIA TECH.

How are coverage-increasing test cases found?

By tracing ***every*** test case!

Dynamic translation	Static callbacks	Static inlining

```
addl  $42,
movl  %edi,
popq  %rbp
```

slower → faster

binary-only ("black-box") → **from source** ("white-box")

How do fuzzers spend their time?

AFL – "naïve" fuzzing
Driller – "smart" fuzzing

8 benchmarks, 1hr trials

Fuzzer, tracer	Avg. % time on exec/ trace	Avg. rate cvg.-incr. test cases
AFL-Clang	91.8	6.20E-5
AFL-QEMU	97.3	2.57E-4
Driller-QEMU	95.9	6.53E-5

▼ **O1: > 90%** time on test case tracing, execution

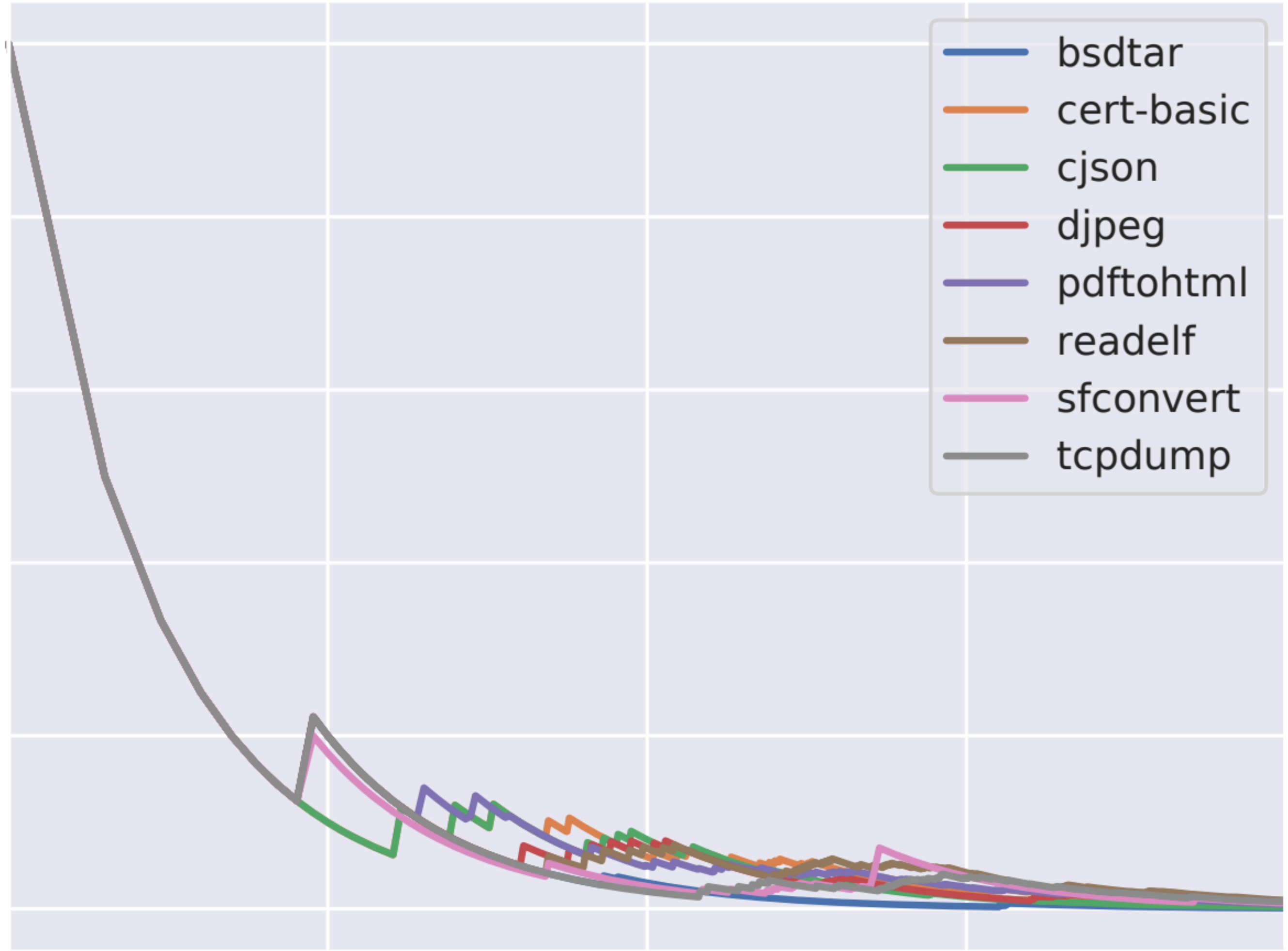
▼ **O2: < 3/10000** test cases increase coverage

Likelihood of coverage-increasing test cases?

AFL-QEMU

5x 24hr trials
x 8 benchmarks

▼ **O3**: rate decreases over time (< **1/10000**)



Impact of tracing *every* test case?

- **Over 90%** of time is spent **tracing test cases**…
- **Over 99.99%** of which are **discarded**!

Equivalent to checking *every* straw to find the needle!

Why is tracing *every* test case expensive?

Storing coverage

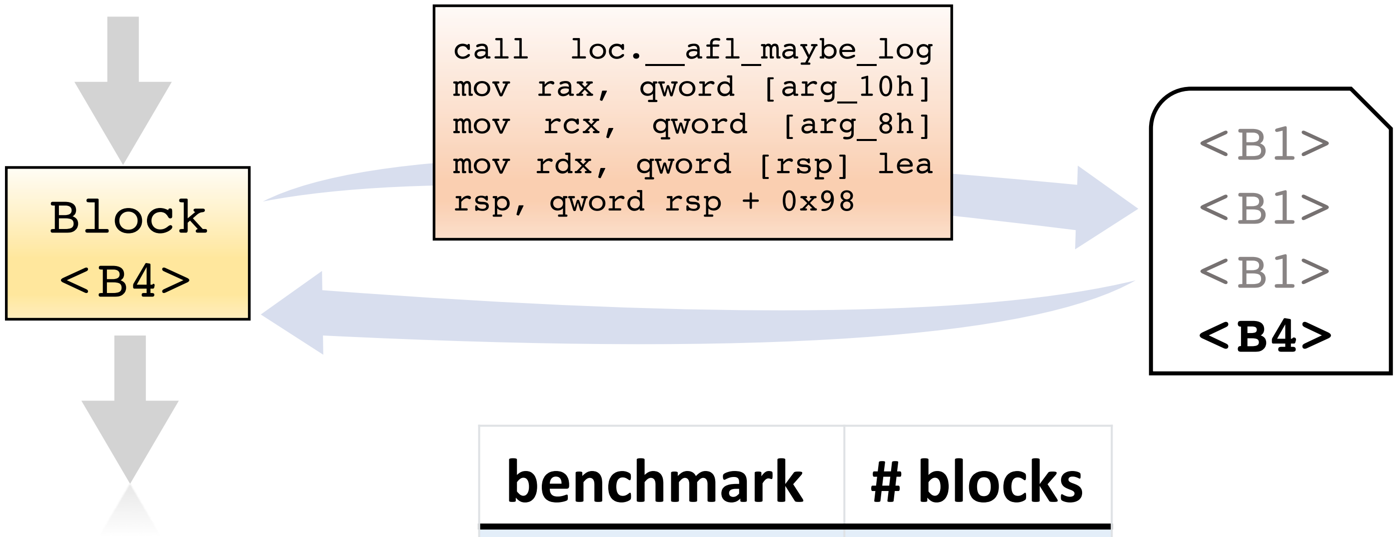
- Bitmaps, arrays

Multiple additional instructions per block

Many blocks, edges

Long exec paths, loops

Overhead quickly adds up



benchmark	# blocks
bsdtar	31379
pdftohtml	54596
readelf	21249
tcpdump	33743

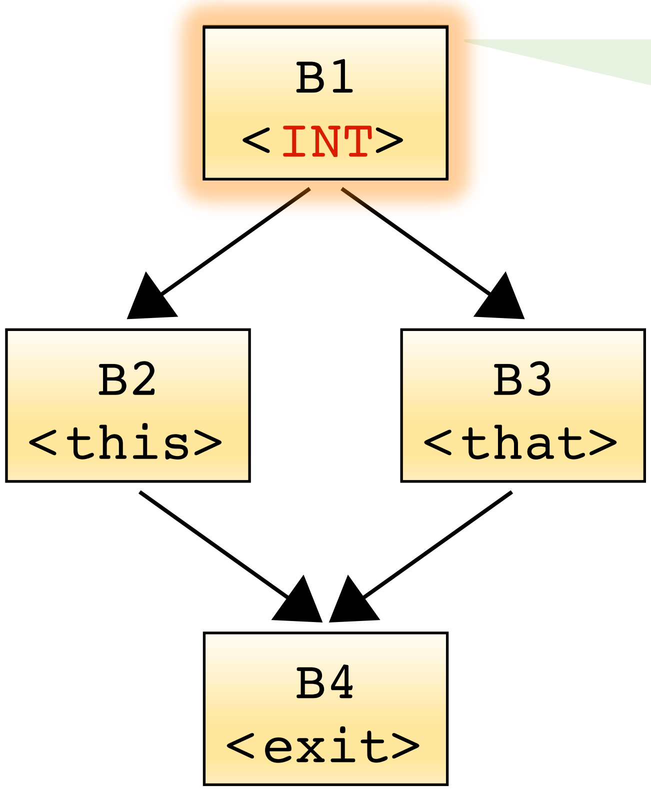
Coverage-guided Tracing

Guiding Principle

Can we identify coverage-increasing test cases
without tracing *every* test case?

Find New Coverage Without Tracing

Apply and dynamically remove interrupts




```
401a49:  CC        INT 03
401a4a:  48 89 e5  mov %rsp, %rbp
401a4d:  48 81 ec  sub $0x380, %rsp
401a54:  89 bd 8c  mov %edi, -0x374(%rbp)
```

Hit

New coverage!

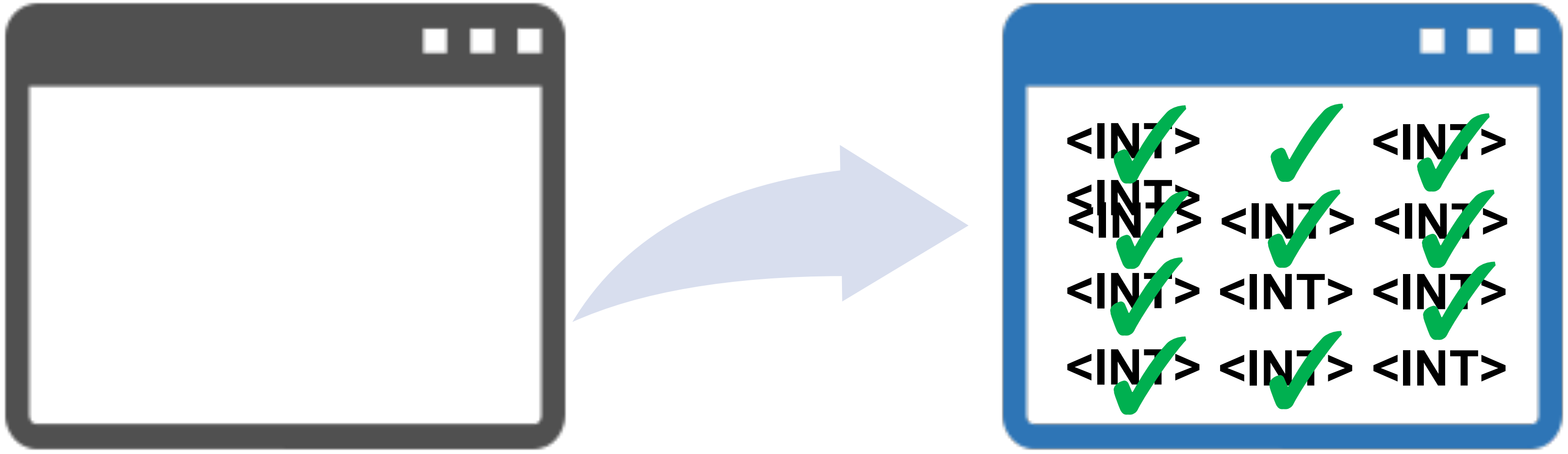
```
401a49:  55        push %rbp
401a4a:  48 89 e5  mov %rsp, %rbp
401a4d:  48 81 ec  sub $0x380, %rsp
401a54:  89 bd 8c  mov %edi, -0x374(%rbp)
```

Reset

Continue!

Coverage-guided Tracing

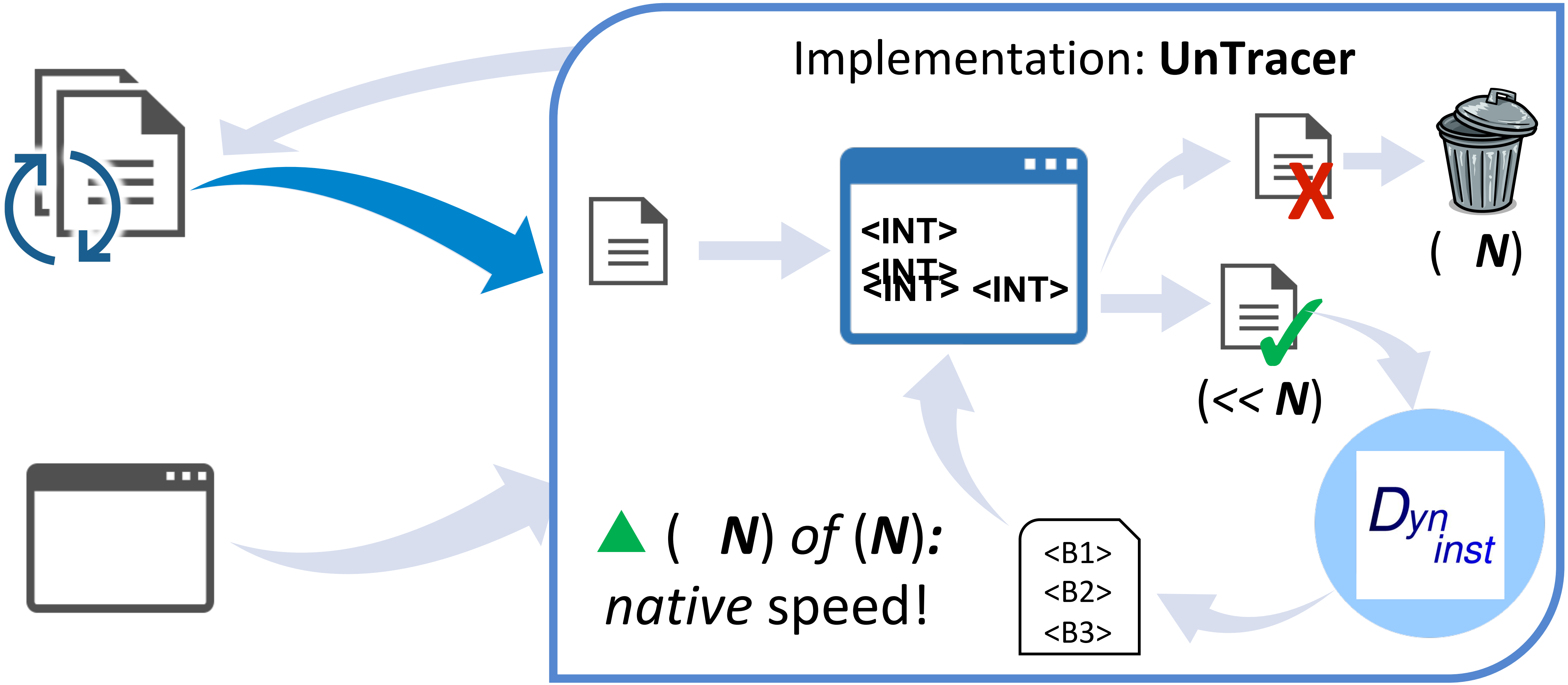
Approach: Trace *only* coverage-increasing test cases
"Filter-out" those that don't hit an interrupt



Hit one
Trace
Reset
Continue

▲ Common case (**99.99%**) *don't hit*—thus aren't traced
▲ Approaches native execution speed (**0% overhead**)

Incorporating CGT into Fuzzing



Evaluation

Performance Evaluation

Goal: isolate tracing overhead

1-core VM's to avoid OS noise

Strip AFL to tracing-only code

8 diverse real-world benchmarks

Compare tracer exec times

- 5 days' test cases per benchmark
- 5x trials per day of test cases

[**BB**] = black-box (binary-only)
[**WB**] = white-box (from source)

Fuzzing Tracer	**Description**
AFL-Dyninst	[**BB**] Static rewriting
AFL-QEMU	[**BB**] Dynamic translation
AFL-Clang	[**WB**] Assembly rewriting
UnTracer (Dyninst)	[**BB**] Coverage-guided Tracing (static rewriting)

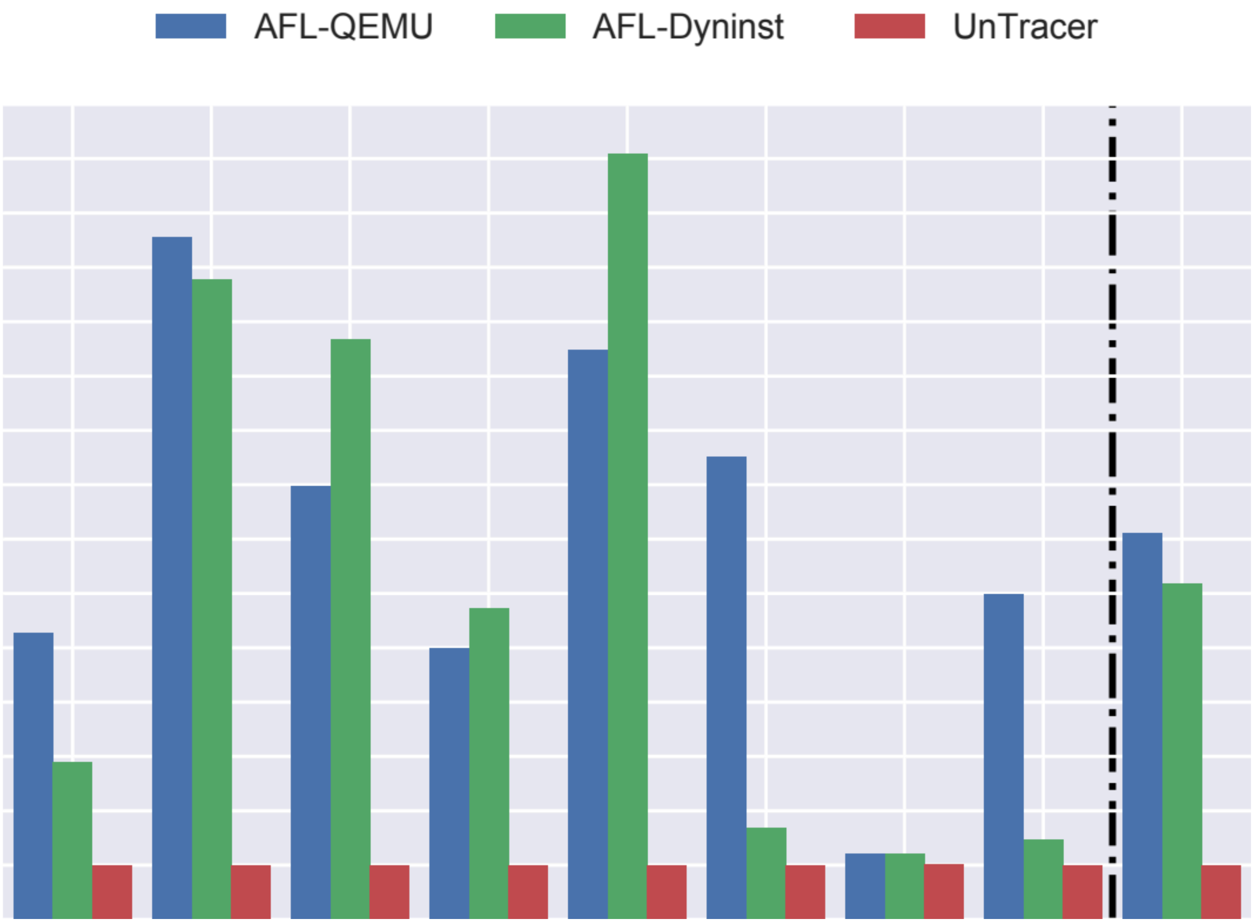
Benchmarks

Benchmark name	Benchmark type
bsdtar (libarchive)	**archiv**ing
cert-basic (libksba)	**crypto**graphy
cjson (cjson)	**web** development
djpeg (libjpeg)	**image** processing
pdftohtml (poppler)	**docu**ment processing
readelf (binutils)	**dev**elopment
sfconvert (audiofile)	**audio** processing
tcpdump (tcpdump)	**net**working

Can CGT beat tracing all with *Black-box*?

AVG. relative overhead:

- ▼ AFL-Dyninst 518%
- ▼ AFL-QEMU 618%
- ▲ **UnTracer** **0.3%**

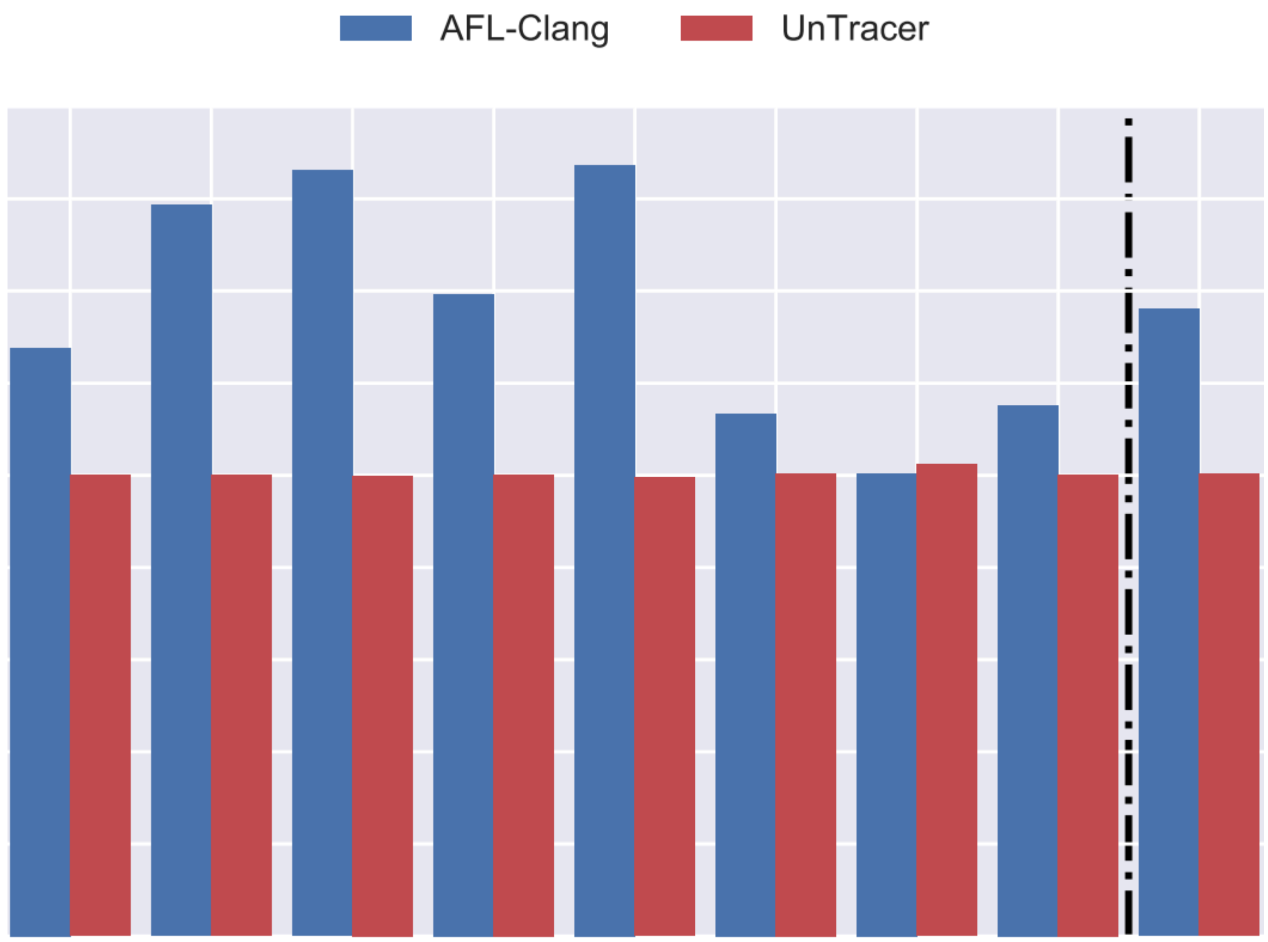


COMPUTER SCIENCE
VIRGINIA TECH.

Can CGT beat tracing all with *White-box*?

AVG. relative overhead:

- ▼ AFL-Dyninst 518%
- ▼ AFL-QEMU 618%
- ▲ **UnTracer** **0.3%**
- ▼ AFL-Clang 36%



COMPUTER SCIENCE
VIRGINIA TECH.

Can CGT boost *hybrid fuzzing* throughput?

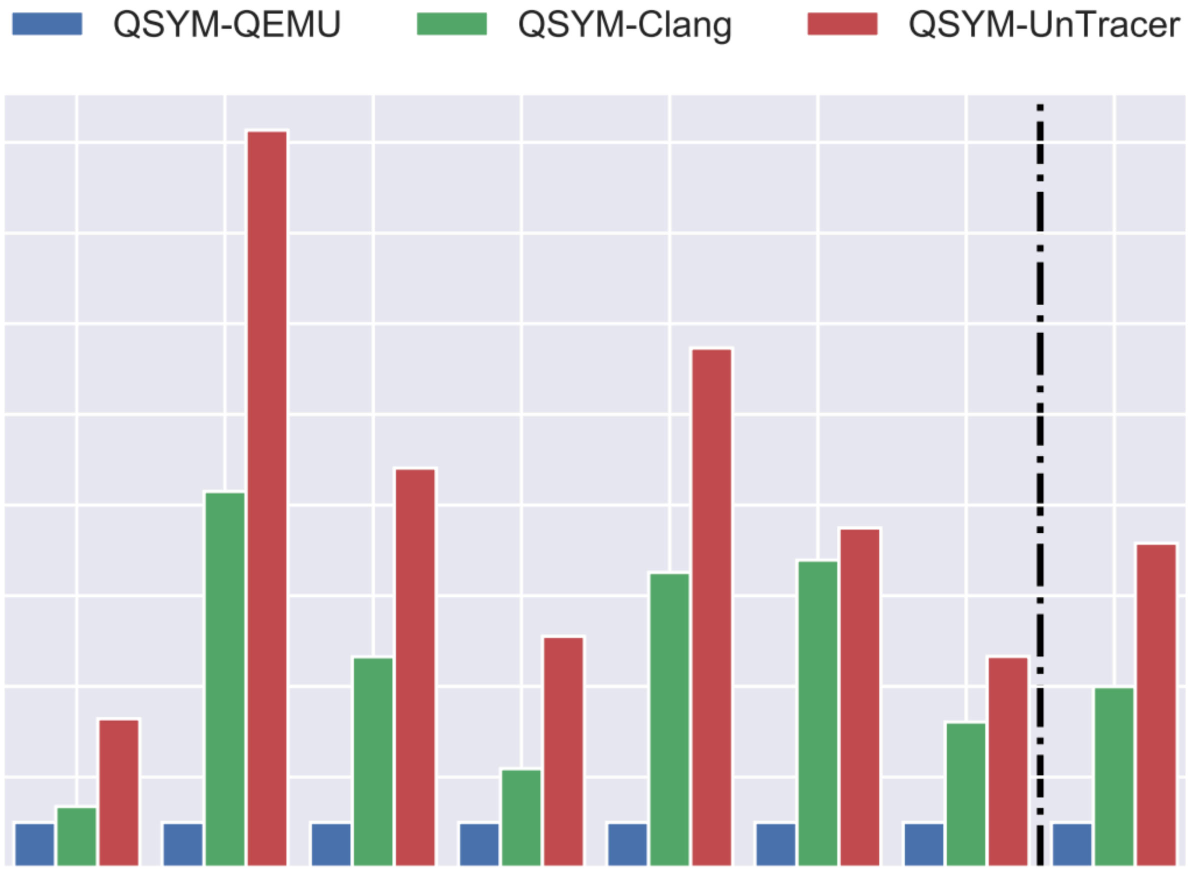
Goal: measure impact on total test case throughput

QSYM (concolic exec + fuzzing)

8 benchmarks, 5x 24-hr trials

QSYM-UnTracer throughput:

- ▲ **616%** >> QSYM-QEMU
- ▲ **79%** >> QSYM-Clang



Conclusions: **Why Coverage-guided Tracing?**

- ▼ Fuzzers find coverage-increasing test cases by tracing *all of them*
- ▼ Costs **over 90% of time** yet **over 99.99%** are **inevitably discarded**

These resources could be better used *to find bugs!*

CGT restricts tracing to the few *guaranteed* to increase coverage

- ▲ Performance: Cuts tracing overhead from **36-618%** to **0.3%**
  Boosts test case throughput by **79-616%**
- ▲ Compatibility: “Filter-out” approach allows plugging-in any tracer
- ▲ Orthogonality: Can combine with other fuzzing improvements
  (e.g., better test case generation, faster tracing)

Thank you!

Our open-sourced software:

• UnTracer-AFL	UnTracer integrated with AFL
• afl-fid	AFL suite for fixed input datasets
• FoRTE-FuzzBench	Our 8 real-world benchmarks

All repos are available here! https://github.com/FoRTE-Research

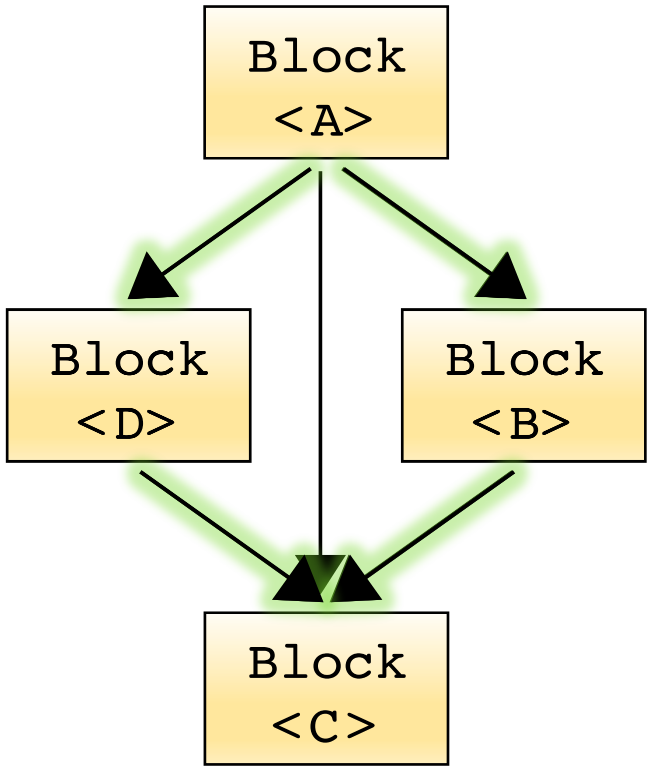
COMPUTER SCIENCE
VIRGINIA TECH.

Expanding Coverage Metrics

Current work: edge coverage, hit counts

Static critical edge handling doable

Hit counts need more complex transforms



Covered Blocks

A, B, C

A, D, C

Implicit Edges

A-B, B-C

A-D, D-C

A-C

CGT versus Hardware-Assisted Tracing

Can approximate Intel-PT overhead:

- AFL-Clang = 36% OH
- AFL-Clang ≅ 10-100% OH rel. to AFL-Clang-fast
- AFL-Clang-fast ≅ 18-32% OH
- Intel-PT ≅ 7% OH rel. to AFL-Clang-fast
- Intel-PT ≅ 19-35% OH

Trace decoding adds way more

Fully Black-box (binary-only) Implementation

Oracle forkserver uses assembly-time instrumentation

Theoretically doable via binary rewriting

- Dyninst’s performance infeasible

Binary hooking an alternative

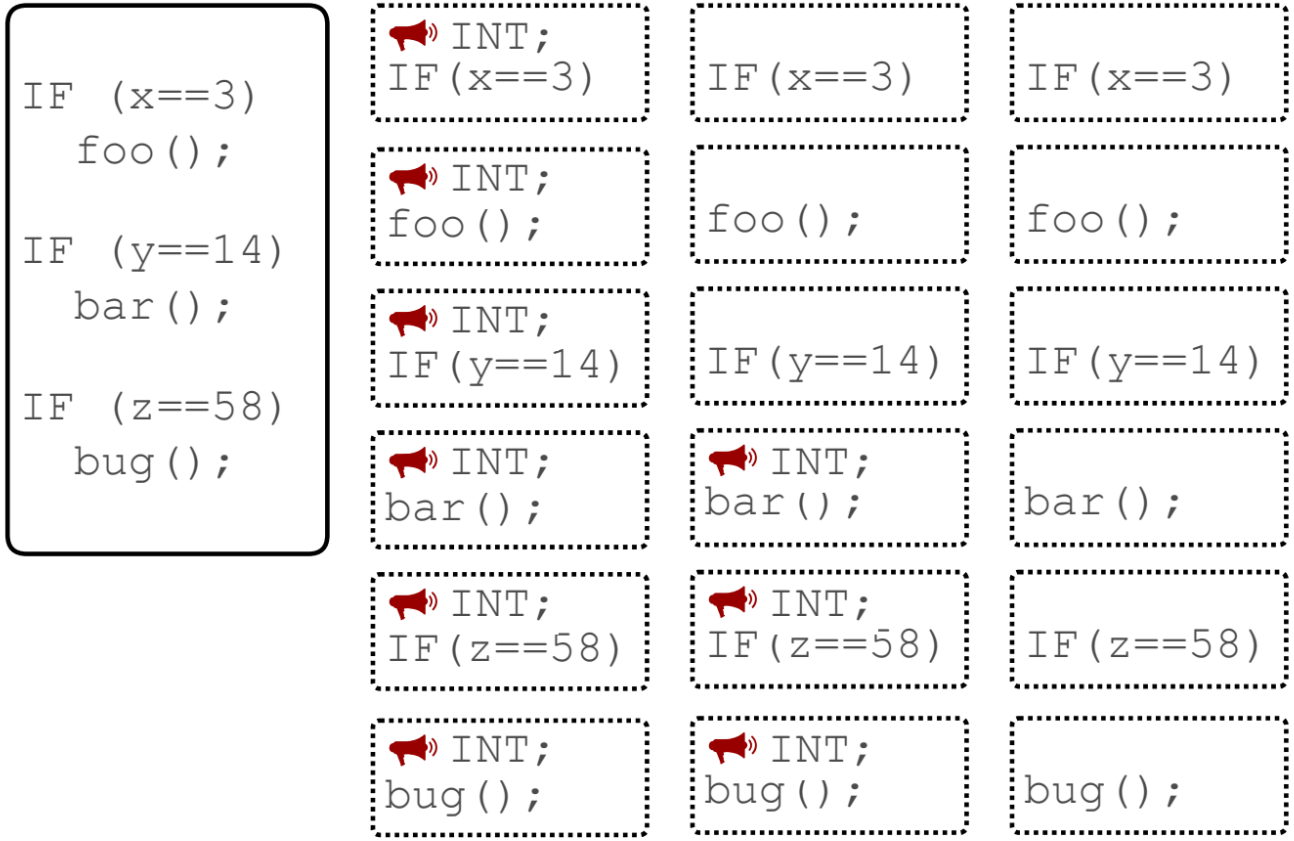
e.g., via LD_PRELOAD

Appendix -- CGT step-by-step

Intuition: restrict tracing to coverage-increasing test cases

1. Statically overwrite start of each block with an interrupt
   - The “Interest Oracle”
2. Get a new test case and run it on the oracle
3. If an interrupt is triggered:
   - Trace the test case’s code coverage
   - Unmodify (reset) all *newly*-covered blocks
4. Return to step 2

Appendix -- CGT step-by-step

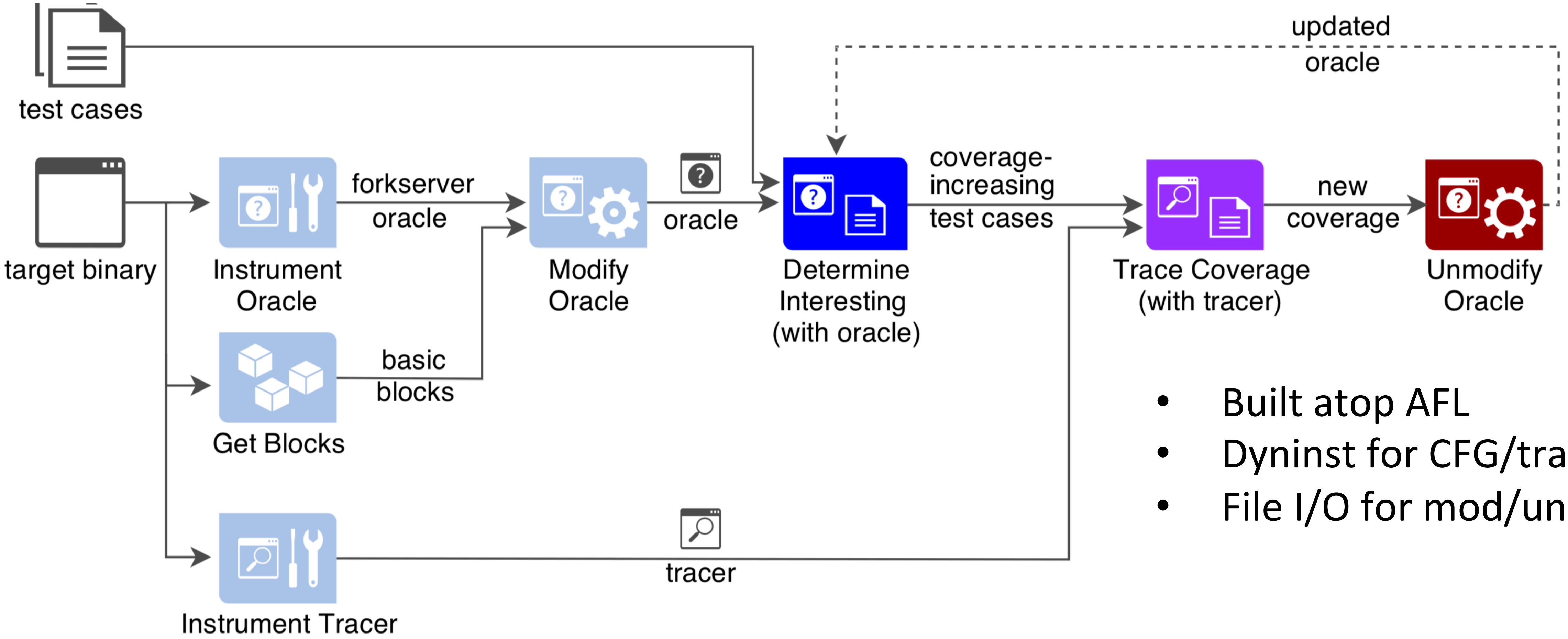


Round	01	02	03	100
Test case values	x=3 y=0 z=0	x=3 y=14 z=0	x=3 y=14 z=58	x=3 y=14 z=58

As more blocks unmodified over time, binary starts to mirror the original

Thus, most testcases are run at **native execution speed**!

Appendix -- Implementation: UnTracer



- Built atop AFL
- Dyninst for CFG/tracing
- File I/O for mod/unmod