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

Stefan Nagy Virginia Tech snagy2@vt.edu

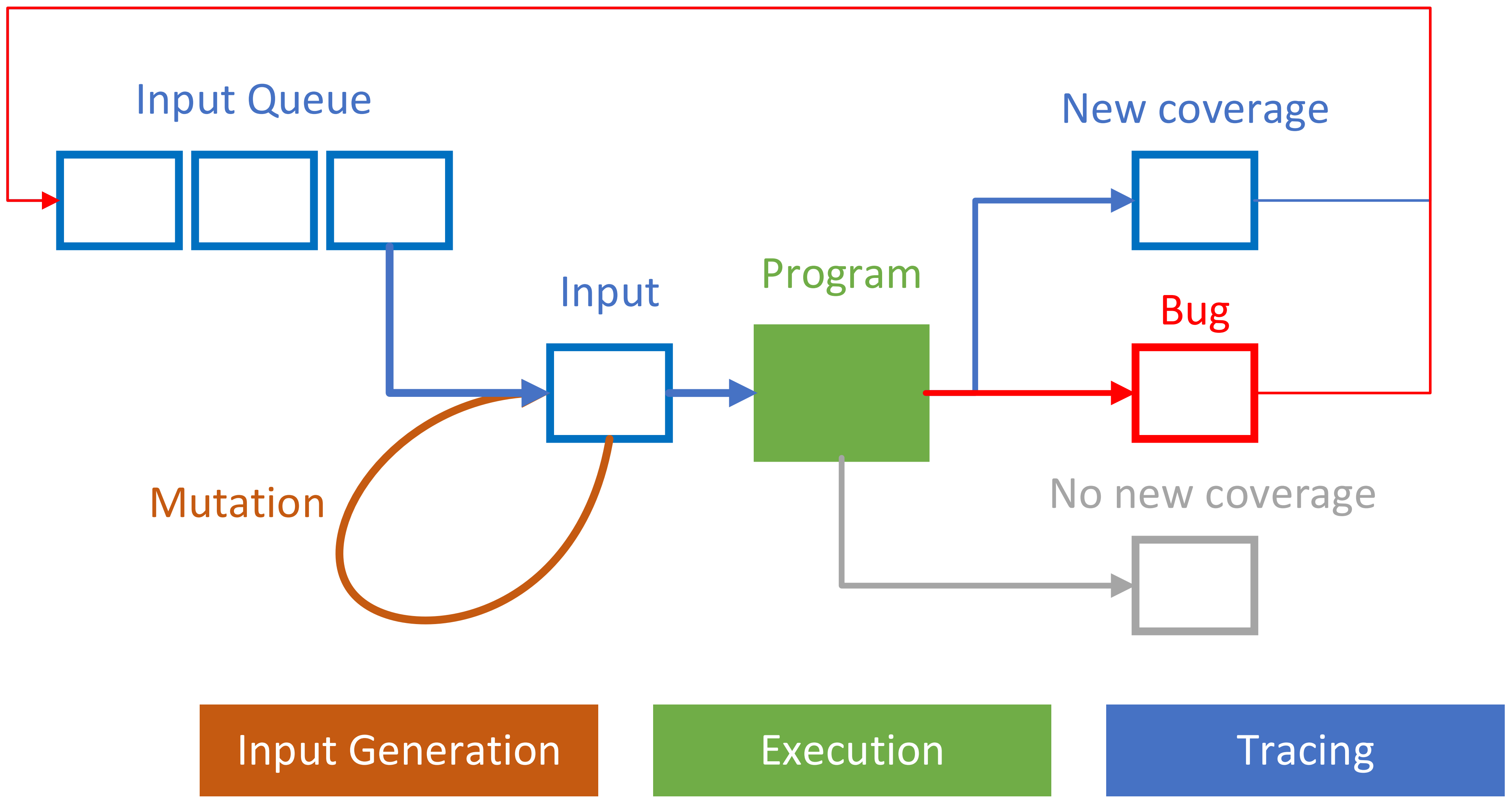
Matthew Hicks Virginia Tech mdhicks2@vt.edu

40th IEEE Symposium on Security and Privacy

MAY 20-22, 2019 AT THE HYATT REGENCY, SAN FRANCISCO, CA

@zjt 2019.11

Re-cap: Algorithm of coverage-guided fuzzing (AFL)



Re-cap: Algorithm of coverage-guided fuzzing

Input generation

- Mutation-based
- Concolic-execution
- …

```
validateXml(xmlStr); // <xml>fuzzable</xml>
```

```
if (str == “magicvalue”) { … }
```

Execution

- Fork-server
- Parallel

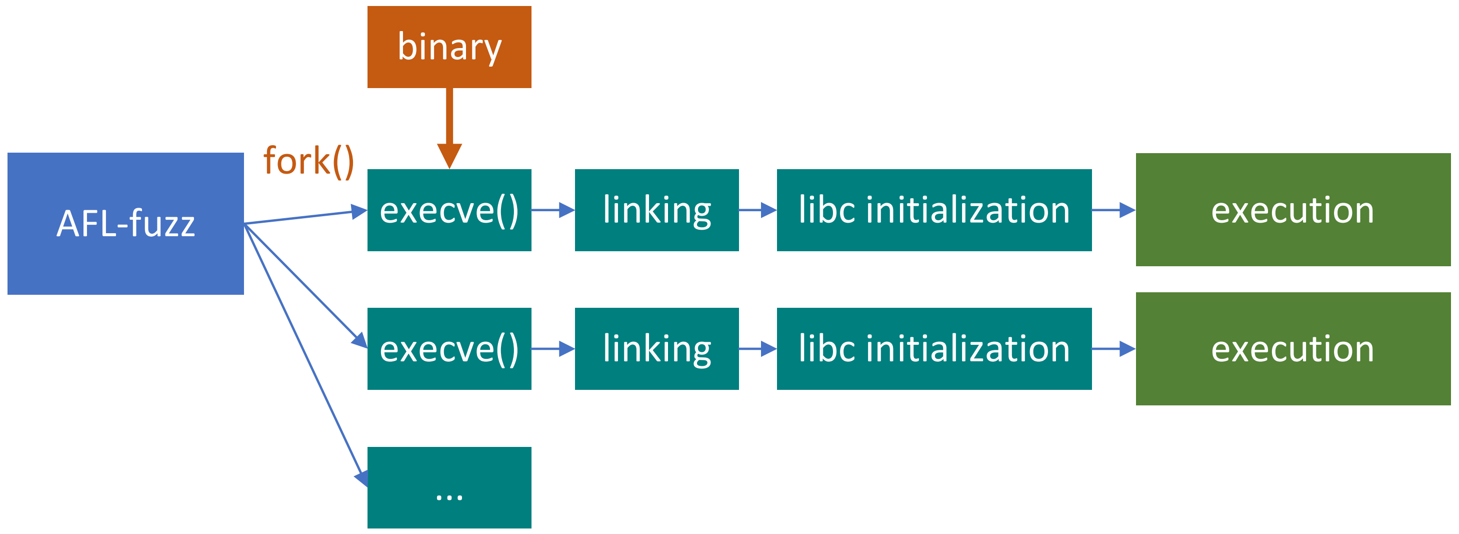
Re-cap: Algorithm of coverage-guided fuzzing

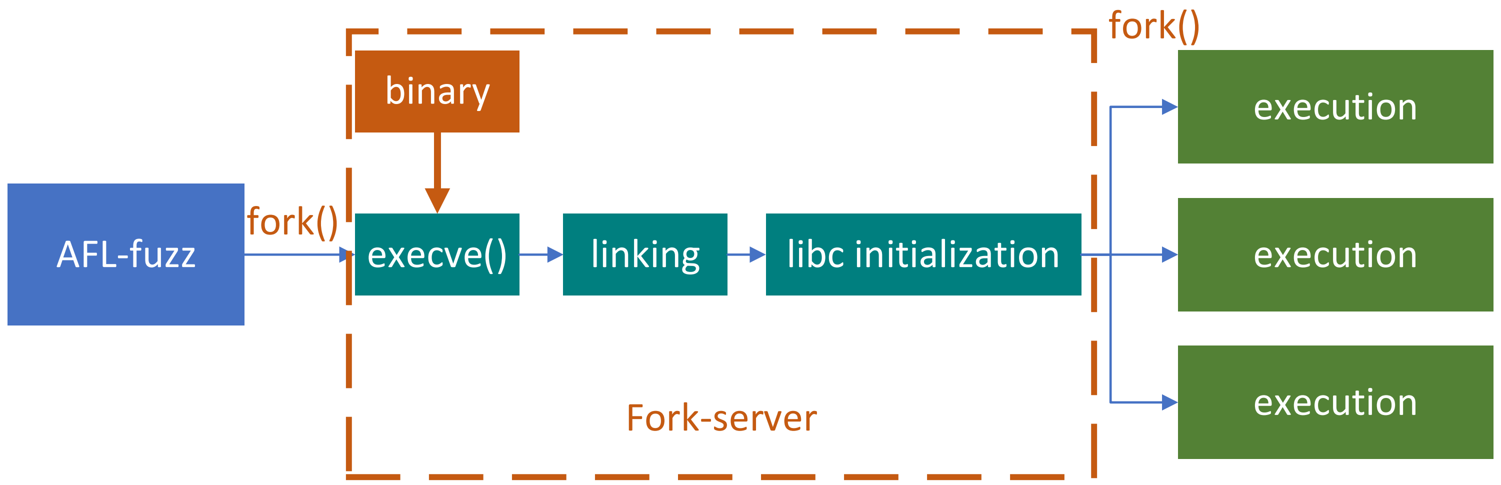
Input generation

- Mutation-based
- Concolic-execution
- …

Execution

- Fork-server
- Parallel

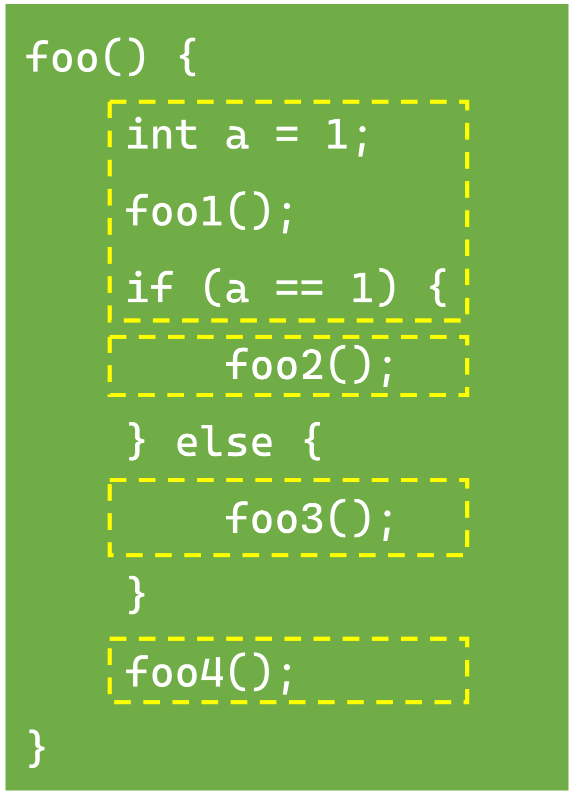




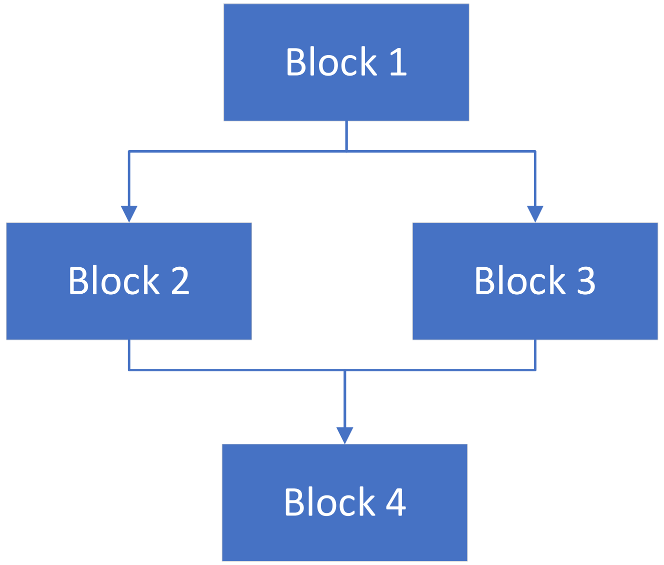
Tracing

- Black-box: binary-only instrumentation (QEMU)
- White-box: compile-time instrumentation

Basic block

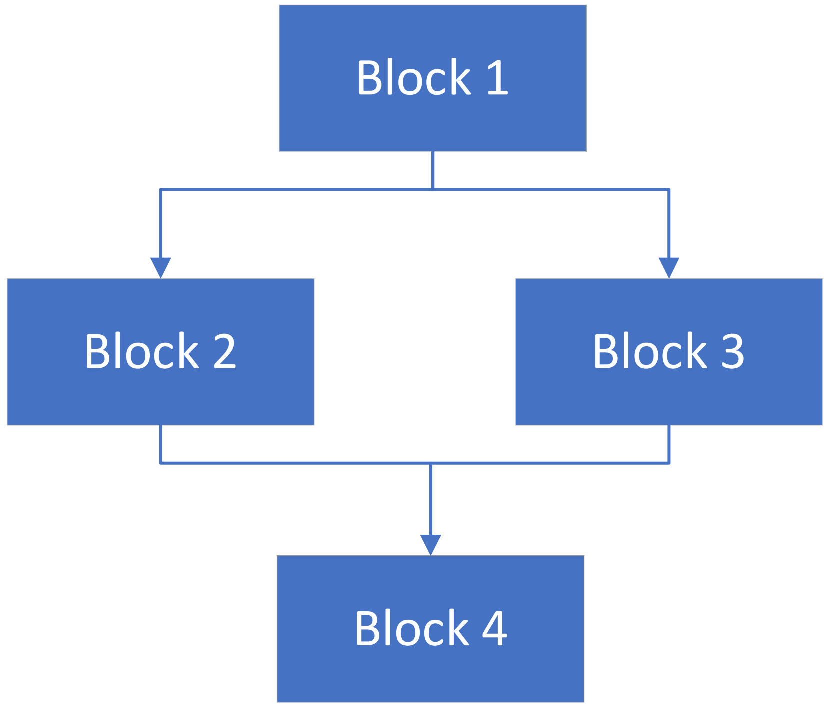


```
foo() {
    int a = 1;
    foo1();
    if (a == 1) {
        foo2();
    } else {
        foo3();
    }
    foo4();
}
```



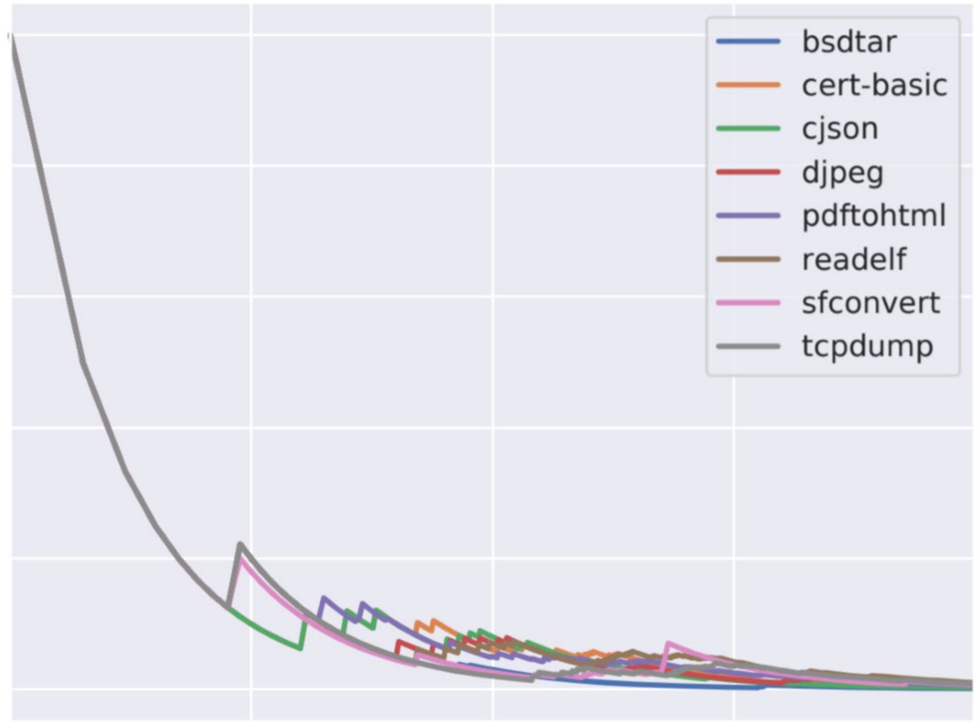
Instrumentation

```
// block start
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
// block content
// …
```

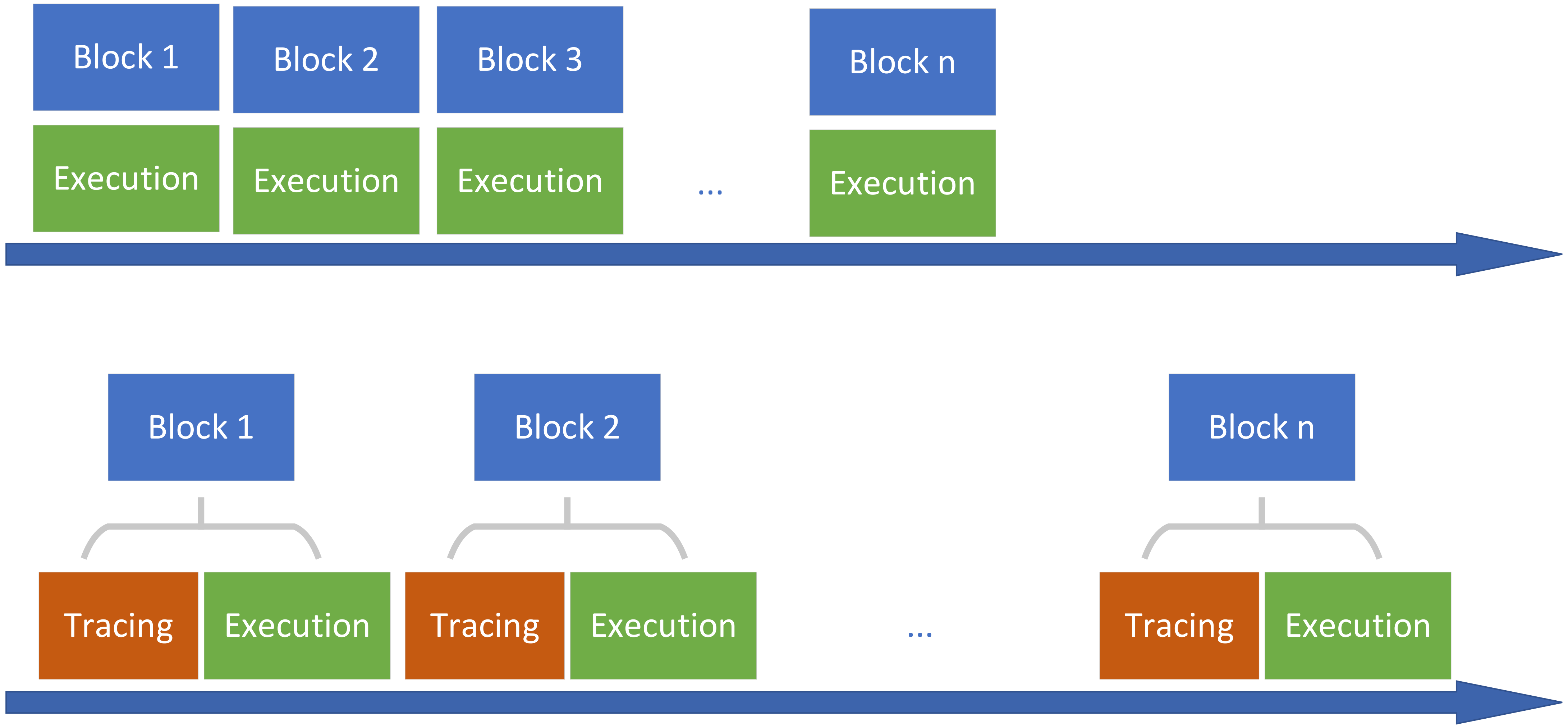


Fuzzing Overhead Evaluation

- Observation 1: > 90% time on execution/tracing
- Observation 2: < 3/10000 test cases increase coverage
- Observation 3: rate decrease overtime

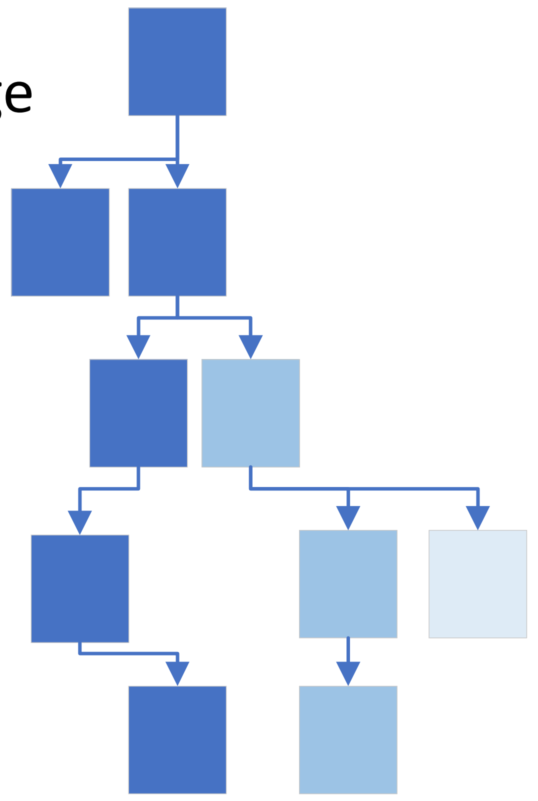


Fuzzing Overhead Evaluation

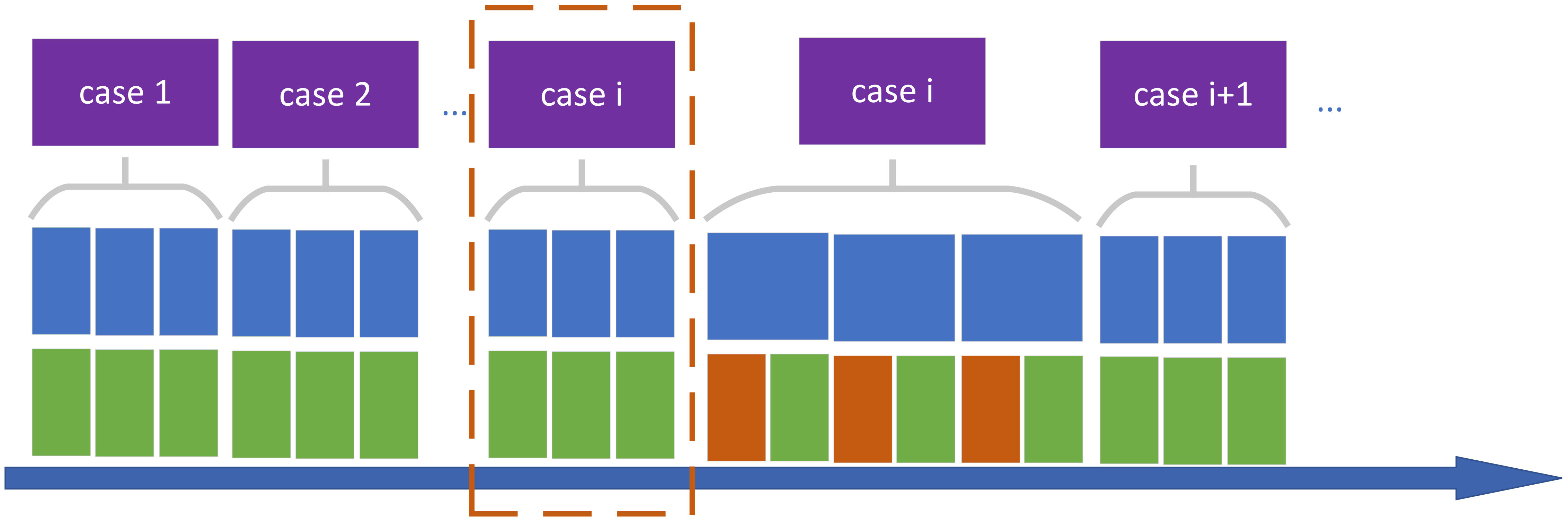


Coverage-Guided Tracing

1. The fuzzer let the program run at full-speed (no-tracing)
2. For a case that triggers new coverage, the program report it to fuzzer
3. The fuzzer trace this case only
4. The fuzzer tell the program not to report about these new coverage



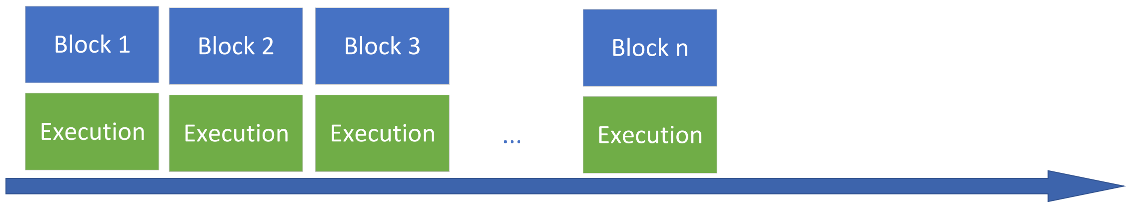
Coverage-Guided Tracing



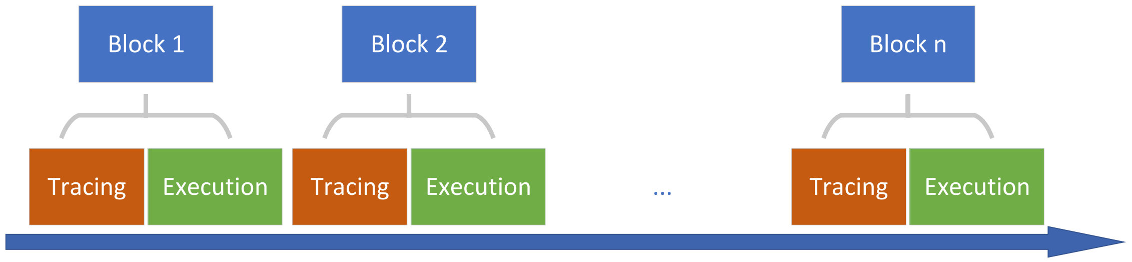
New coverage !!!

Coverage-Guided Tracing

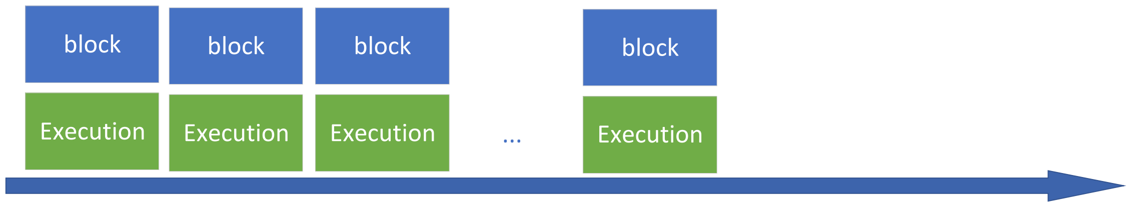
Normal exec



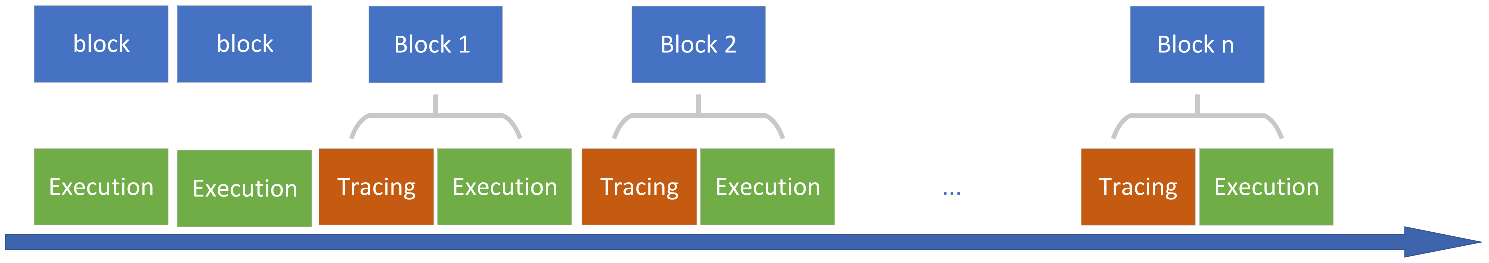
Tracing every block



Coverage-Guide Tracing
(non-coverage-increase)
99.97%



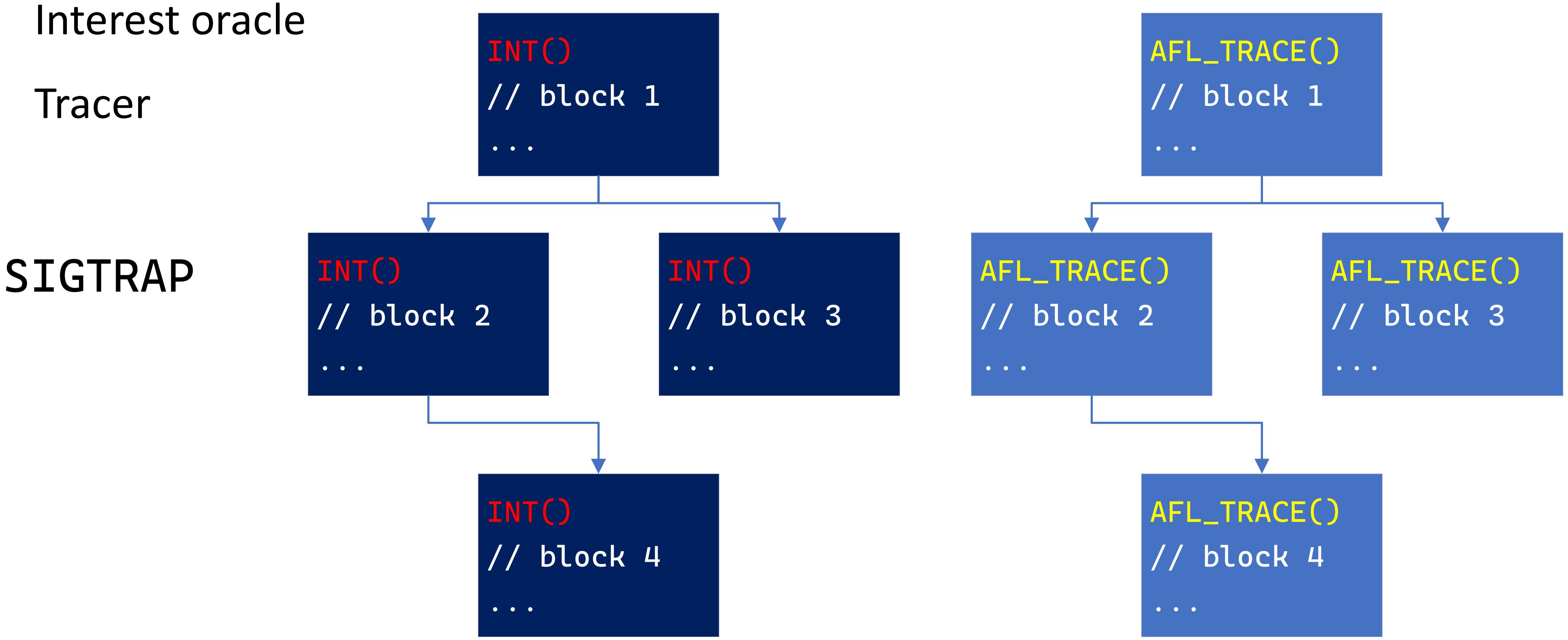
Coverage-Guide Tracing
(coverage-increase)
0.03%



Implementation

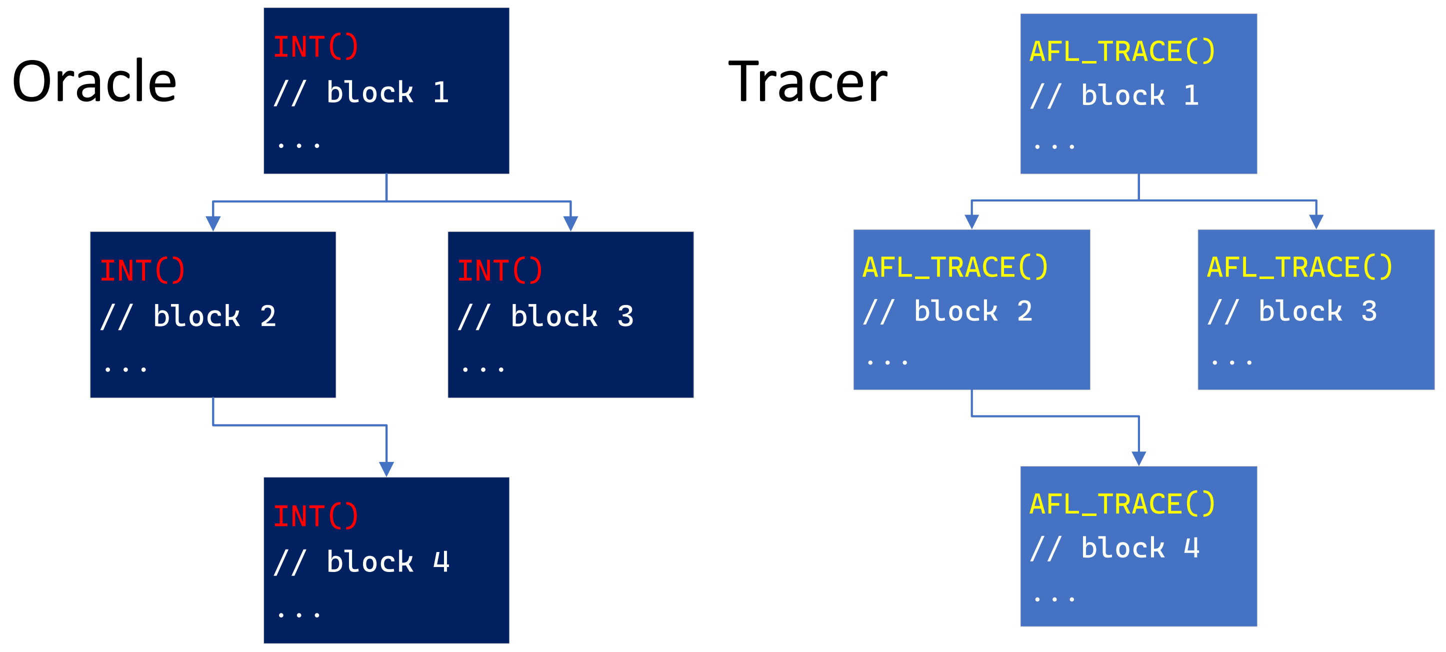
Two versions of binary:

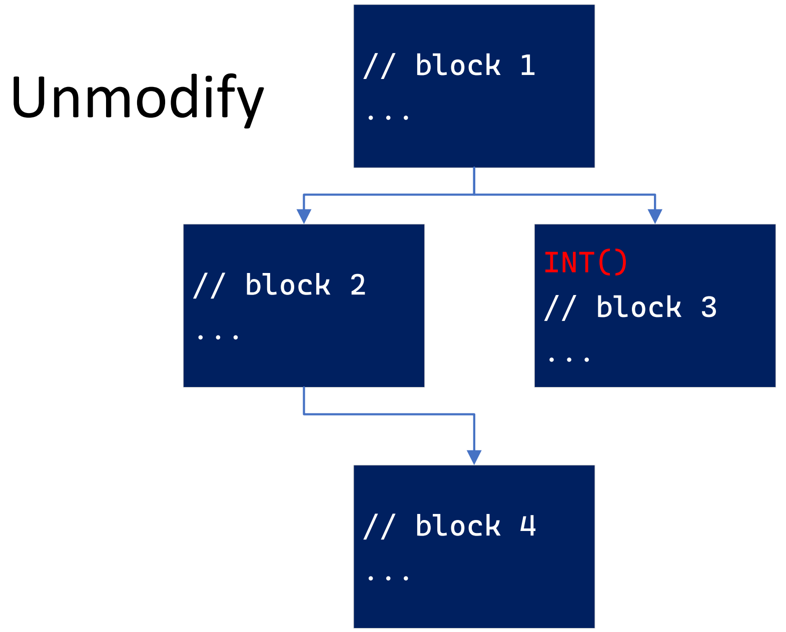
- Interest oracle
- Tracer



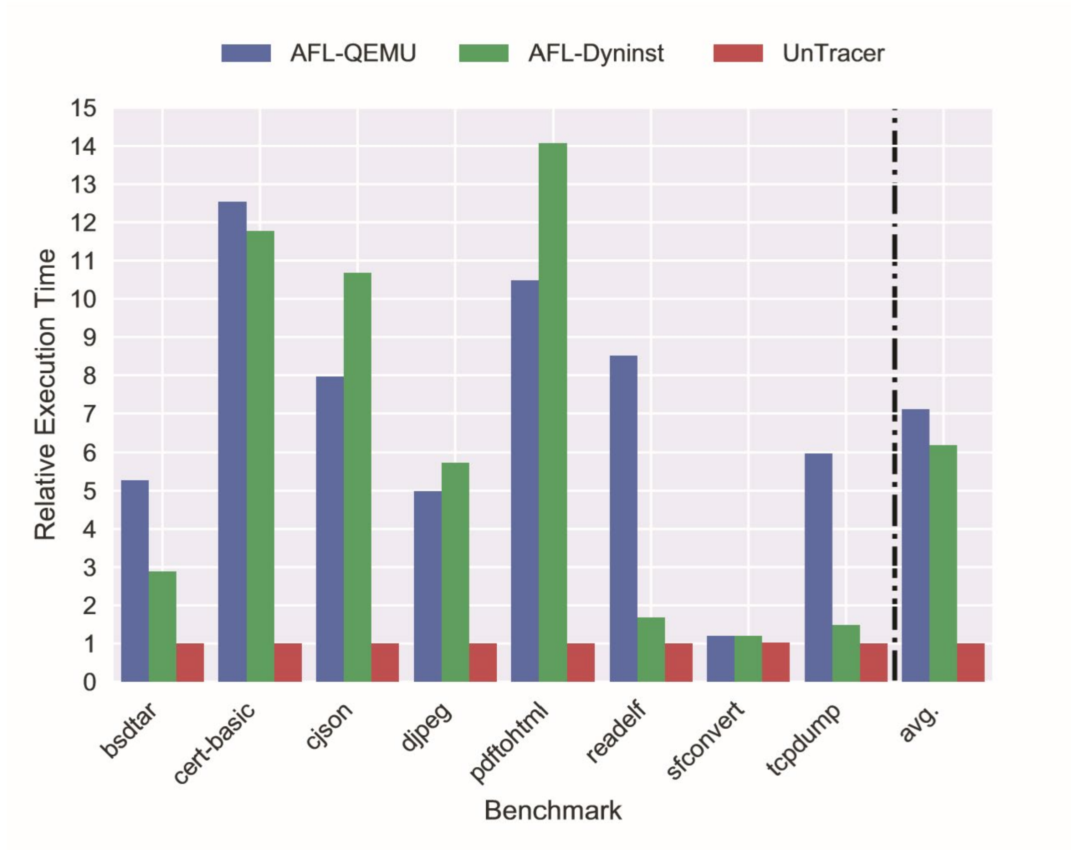
Implementation

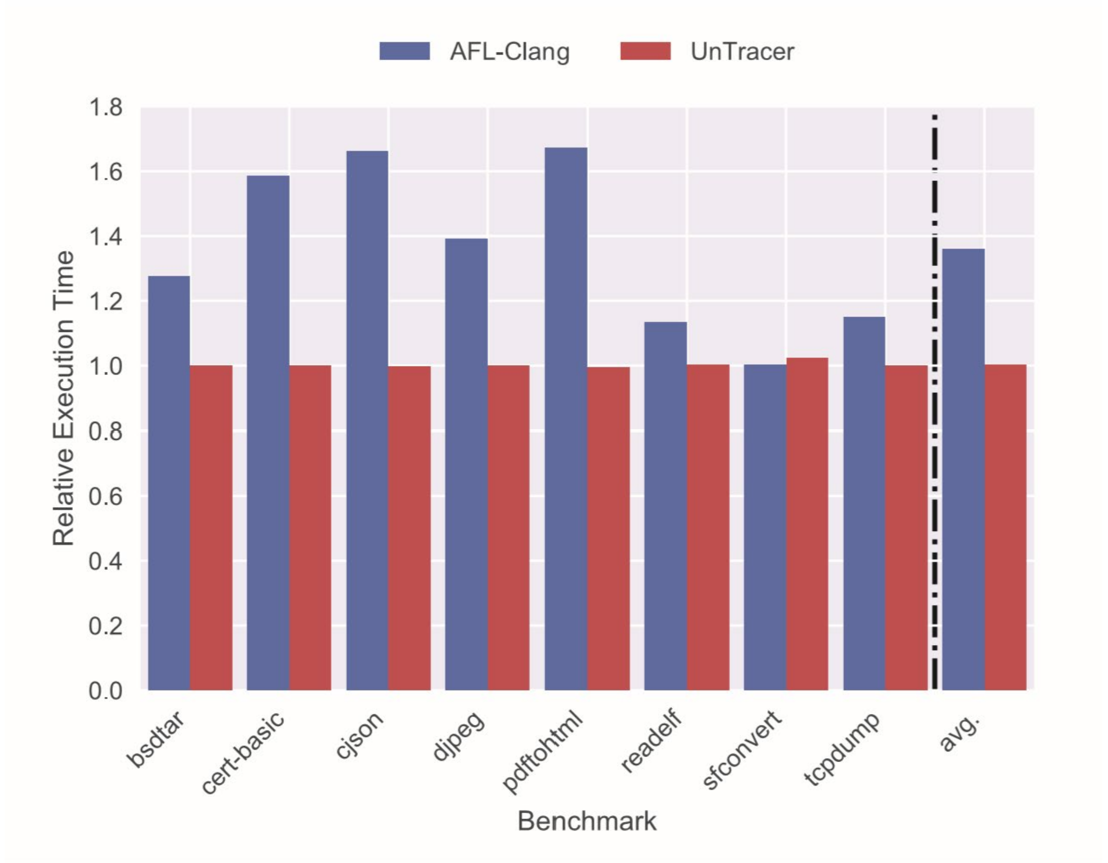
1. Start two fork-servers
2. Execute on interest oracle
3. Trace interesting test cases
4. Stop oracle fork-server
5. Unmodify (remove interrupts)
6. Restart fork-server





Evaluation





612% / 518% / 0.3%

36% / 0.3%

Conclusion

- Fuzzers find coverage-increasing test cases by tracing all of them
- Costs over 90% of time yet over 99.99% are inevitably discarded
- The resource could be used to find bugs

- Cut tracing overhead from 36%-618% to 0.3%

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

Thanks.