ConcFuzzer: a sanitizer guided hybrid fuzzing framework leveraging greybox fuzzing and concolic execution

Peng Li, Rundong Zhou, Yaohui Chen,

Yulong Zhang, Tao (Lenx) Wei

lipeng28@baidu.com

Baidu X-Lab

Coverage-guided greybox fuzzing

- Fuzzing procedure
  1. Start with sample inputs
  2. Mutate inputs to generate mutants
  3. Run them and collect coverage information
  4. Save and prioritize mutants that contribute new coverage
  5. Repeat the step 2 until the end

- American Fuzzy Lop (AFL)
  - State-of-the-art fuzzer, widely used in both Industry and academia

Coverage-guided greybox fuzzing (cont.)

- Pros:
  - Fast and cheap, i.e., native speed
- Cons:
  - Difficult to access code guarded by complex conditions

```
test_me (int x) {
    if (x == 0xdeadbeef) {
        assert(false && “error!”);
    }
}
```

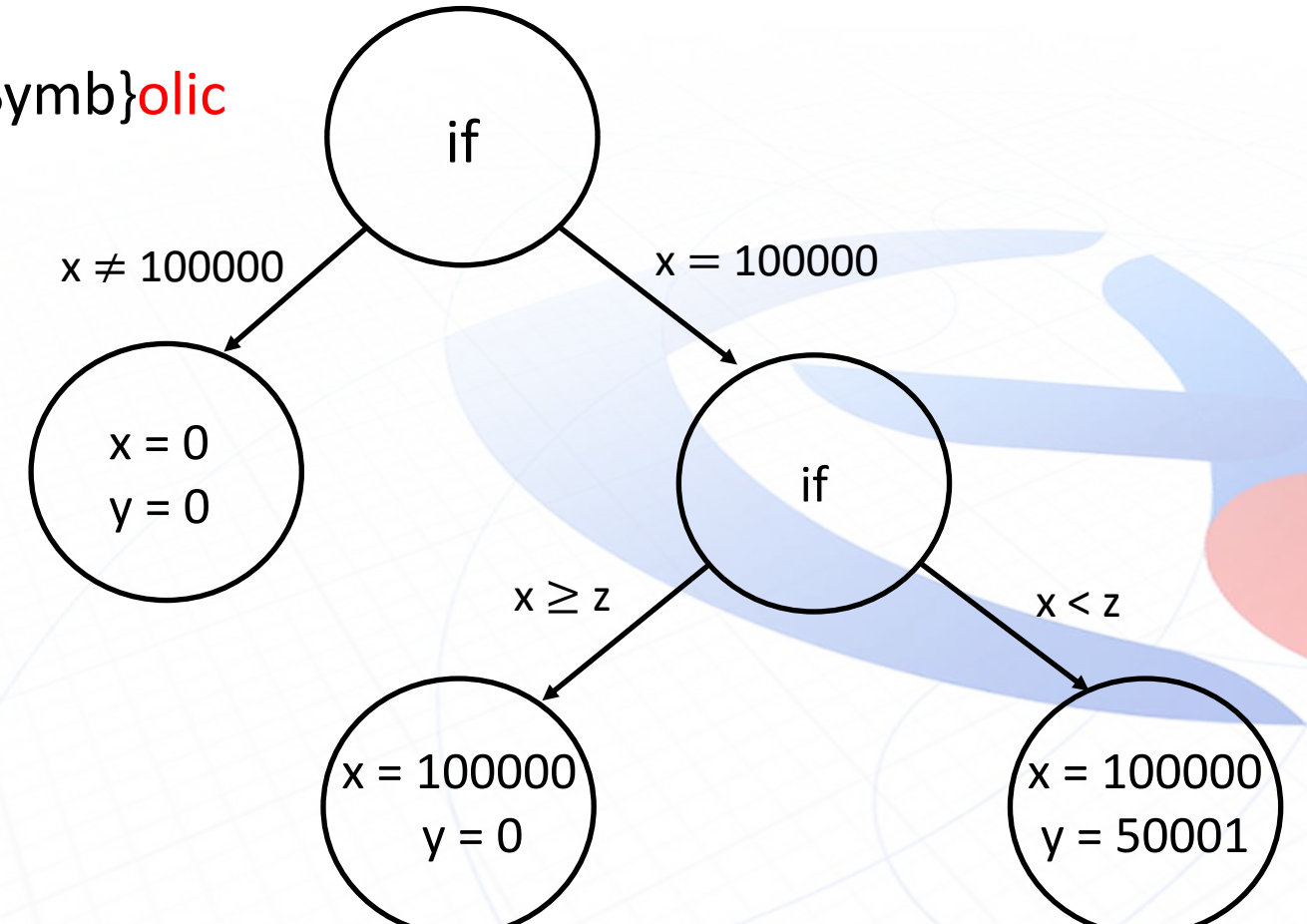
Probability of hitting error = $1 / 2^{32}$

Concolic Execution

- Concolic = Conc{rete + Symb}olic

```
void foo (int x, int y) {
  int z = 2*y;
   if (x == 100000) {
     if (x < z) {
       /* error */
       assert(0);
     }
   }
}
```

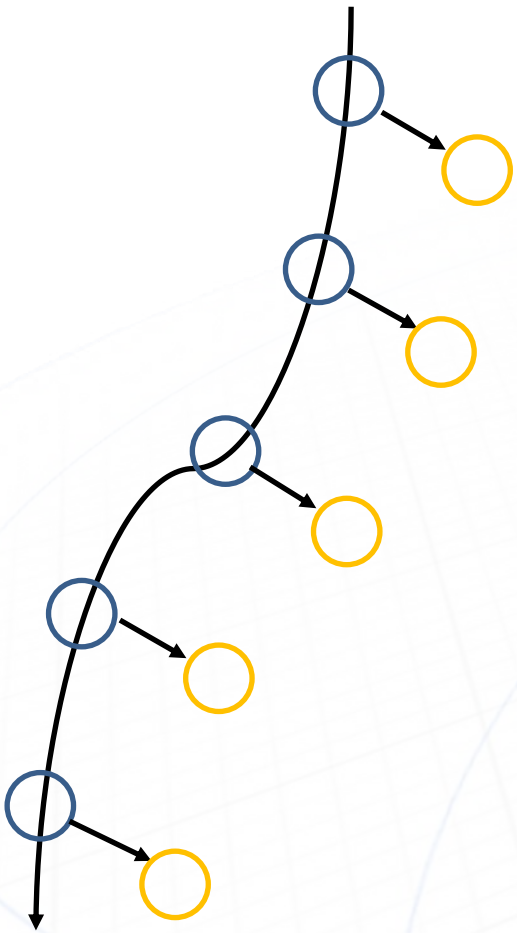
Snippet of C code from wikipedia



Concolic Execution (cont.)

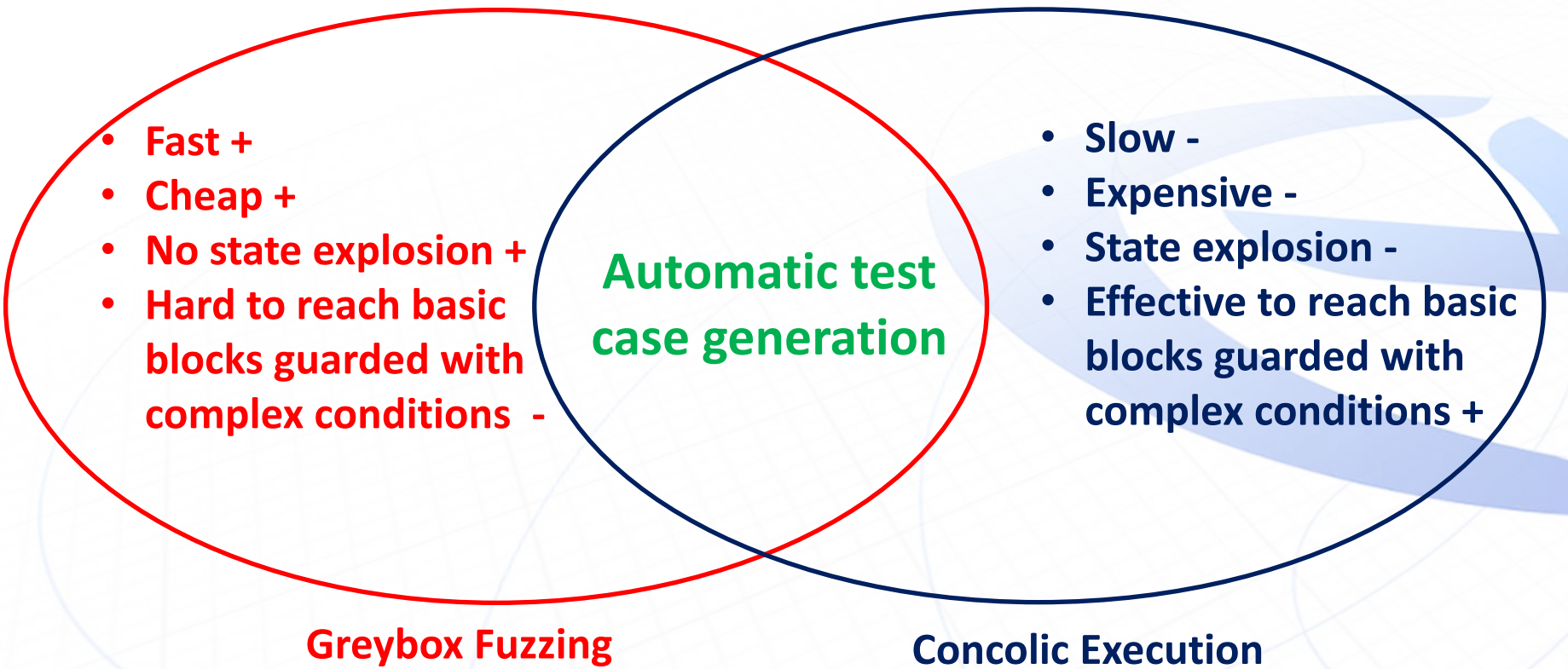
- Pros:
  - Systematic state exploration to reach high coverage
  - Automatic test case generation
  - White box fuzzing
  - Relief the constraint solving burden. i.e., one time solving per path, avoid potential non-linear solving choke
- Cons:
  - Slow compared against native execution
  - State explosion problem

Concolic Execution on top of KLEE



1. Delay constraint solving until the state get scheduled
2. If the state's path constraint is satisfiable, compute the input
3. Otherwise, discard the state

Natural complementation between Fuzzing and Concolic execution



Prior arts of hybrid fuzzing

- Hybrid concolic testing [Rupak et al., ICSE'07]
  - Interleaving of random testing and bounded exhaustive concolic execution
- Driller [Nick et al., NDSS'16]
  - Run fuzzing (AFL) and concolic executor (Angr) in parallel
- Hybrid solution outperforms either part w.r.t code coverage

Does high code coverage guarantee finding the bug?

```
int sum(int a, int b) {
        int c = a + b;
        If (c%2 == 0)
          print("c is even \n");
        else
          print("c is odd \n");
        return c;
}
```

- Proper checks must be there
- Inputs violating checks must be present

Motivation

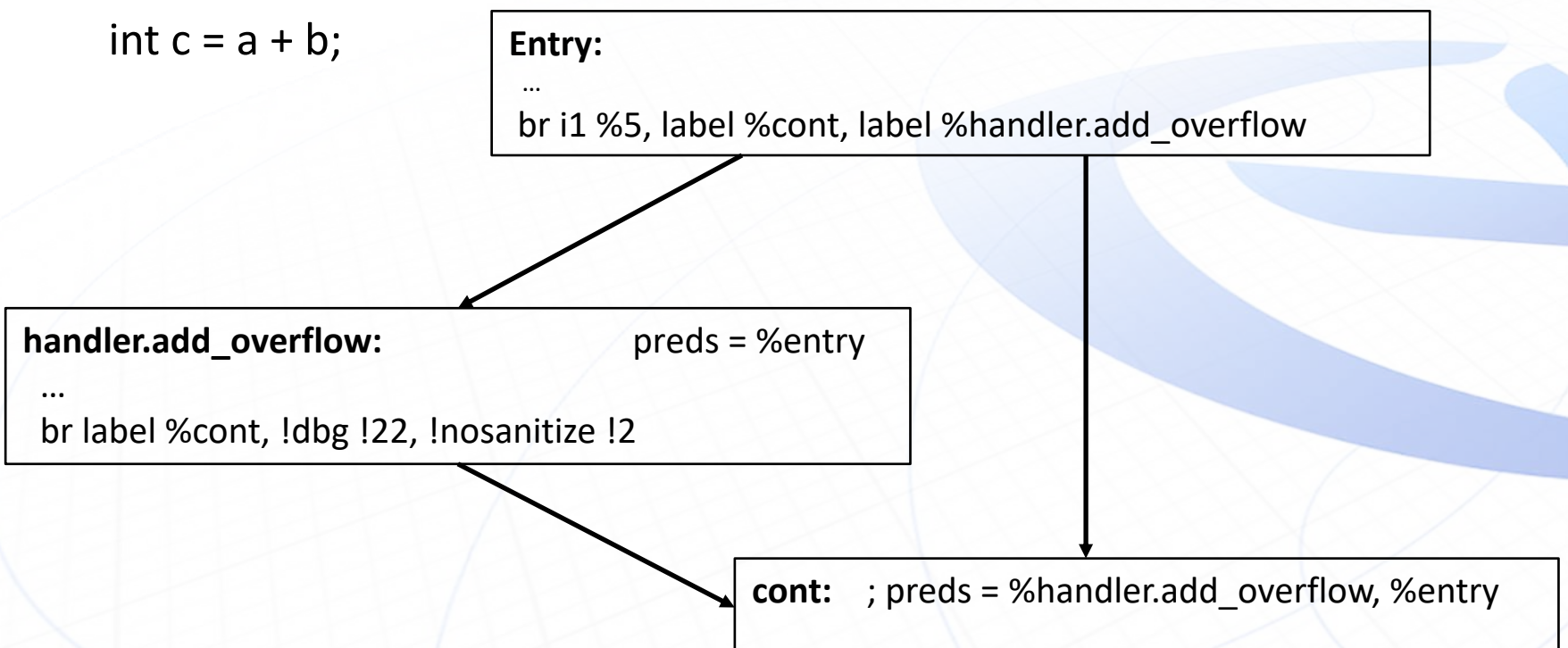
We want to build a system which is able to

- Instrument checks automatically
- Generate inputs with the guidance of checks to catch bugs as quicker as possible

Automatic oracle generation through sanitizers

- Sanitizers
  - Address Sanitizer, Undefined behavior Sanitizer, Thread Sanitizer …

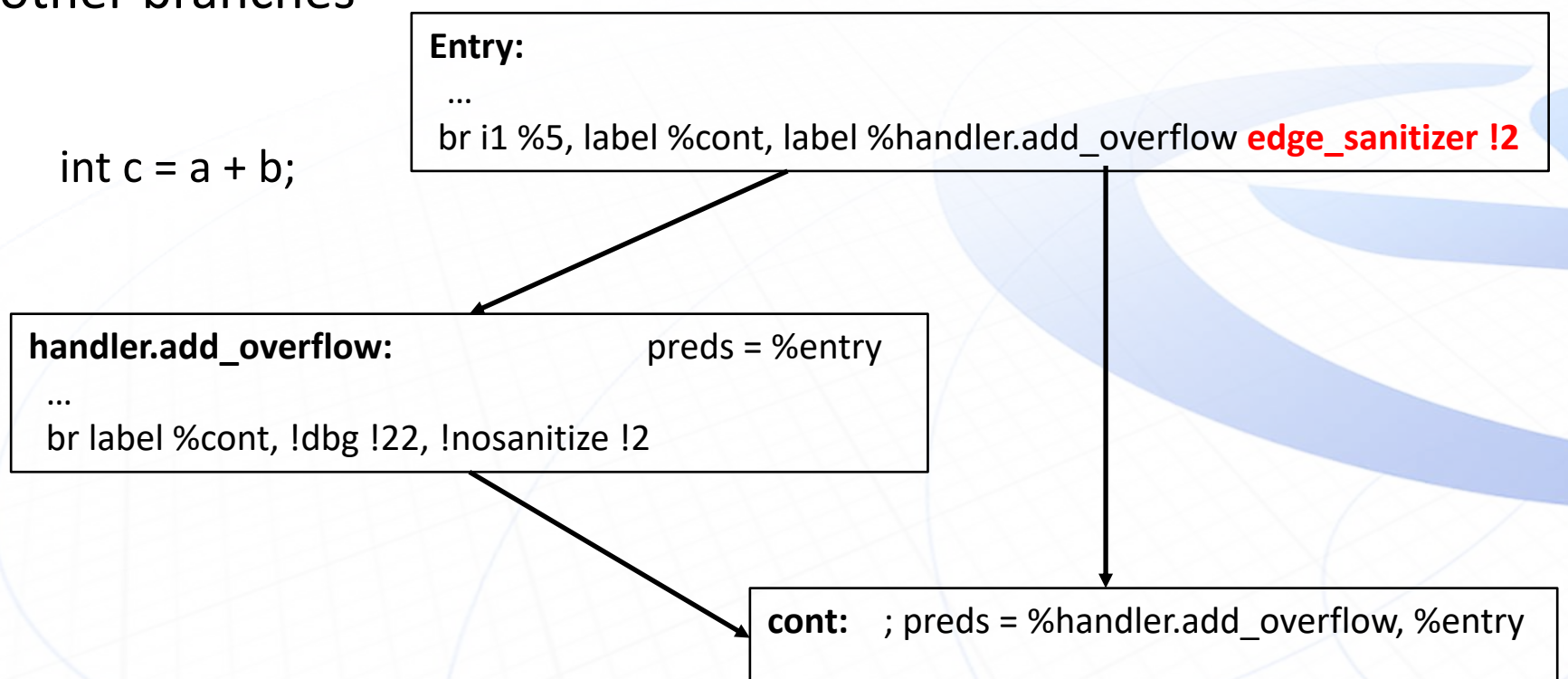
int c = a + b;



A sanitizer guided hybrid fuzzing framework

- Instrument a mark for sanitizer branches to differentiate from the other branches

```
int c = a + b;
```



A sanitizer guided hybrid fuzzing framework (cont.)

- AFL is capable of finding inputs accessing sanitizers fast
  - Most of inputs may access sanitizer branch but did not trigger violations
  - The inputs accessing more sanitizers are prioritized to pass to concolic executor
- Concolic executor determines if ***complementary*** inputs triggering bugs exist through constraint solving
  - Concolic executor prioritizes states leading to sanitizer violation
  - Concolic executor only computes new state's test case and will not continuously explore it in the slow interpretation manner
  - All test cases produced by concolic executor are passed to AFL on the fly

Marked sanitizers in UBSan

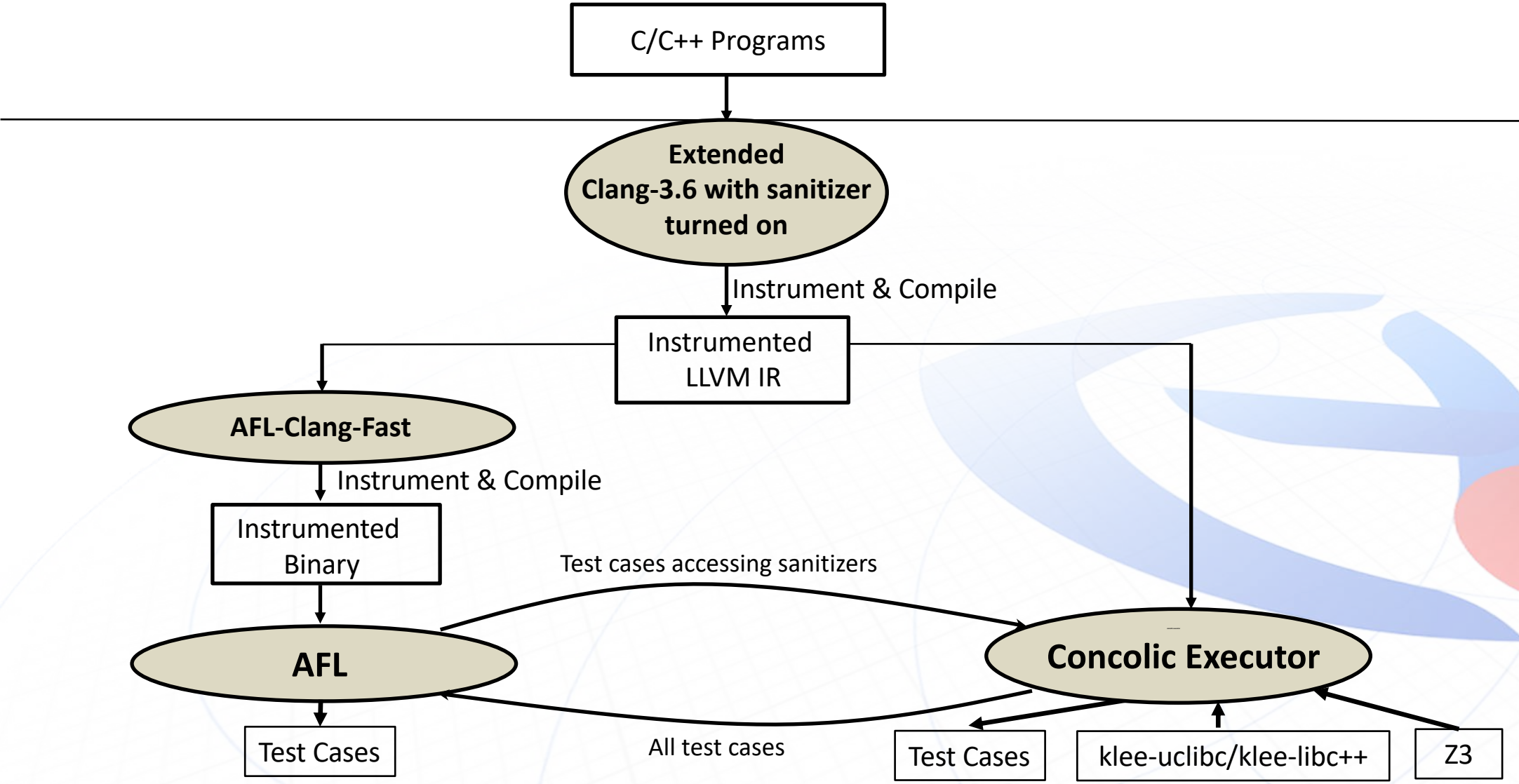
- Addition
- Subtraction
- Multiplication
- Left/Right shift
- Division/Modulus
- Load invalid Enum/Bool types
- Out of bound array index

Beyond C semantics

- Apollo (Baidu's open autonomous driving platform)
  - https://github.com/ApolloAuto/apollo
  - ~ 145,000 LOC pure in C++
  - ~ 90 global partners
- Most of C++ features are handled by Clang front-end
- uclibc++

Beyond C semantics

- Apollo (Baidu's open autonomous driving platform)
  - https://github.com/ApolloAuto/apollo
  - ~ 145,000 LOC pure in C++
  - ~ 90 global partners
- Most of C++ features are supported by Clang front-end
- ~~uclibc++~~
  - Obsolete
  - Does not support C++11
- klee-libc++
- Atomic C++ operations


C/C++ Programs
Extended
Clang-3.6 with sanitizer
turned on
Instrument & Compile
Instrumented
LLVM IR
AFL-Clang-Fast
Instrument & Compile
Instrumented
Binary
Test cases accessing sanitizers
AFL
Concolic Executor
Test Cases
All test cases
Test Cases
klee-uclibc/klee-libc++
Z3

How well does it perform on real world applications?

djpeg (Libjpeg-9b)

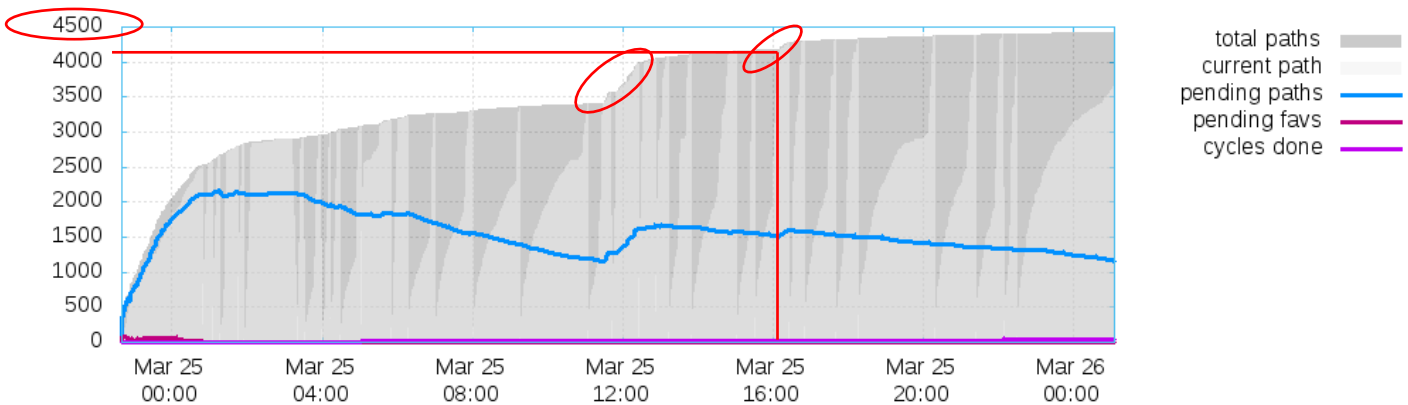


Fig 1. ConcFuzzer's 24 hours running results for djpeg

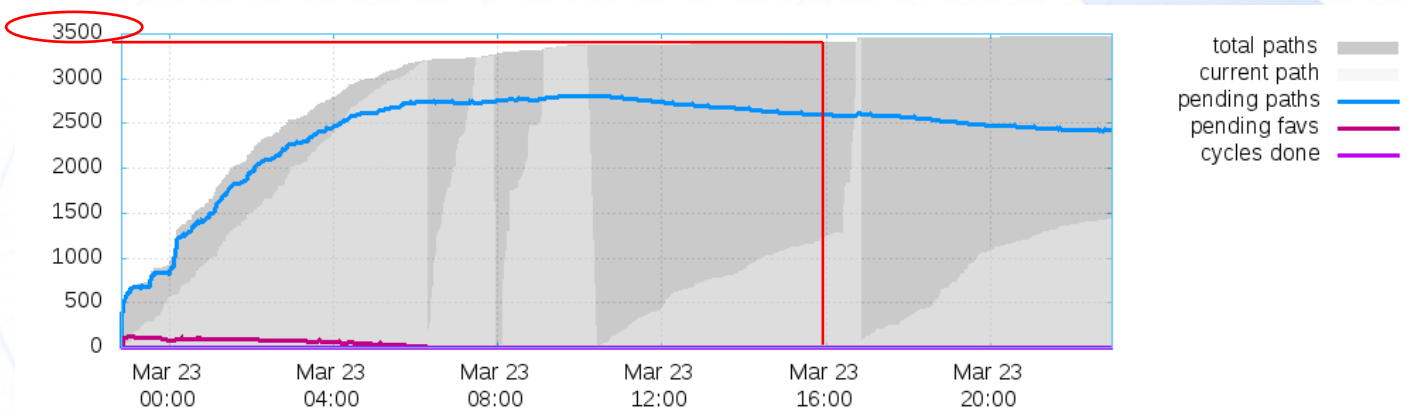


Fig 2. AFL's 24 hours running results for djpeg

djpeg (Libjpeg-9b)

	Covered Paths #	Imported inputs	# of bugs found
AFL	3,476	N.A.	114 Left shift overflow 6 buffer overflow

Table 1. AFL vs ConcFuzzer vs KLEE results after 24 hours execution. N.A. means AFL and KLEE do not have the feature of importing inputs from concolic executor to AFL. # of generated ktests is used to denote the covered paths # in KLEE

djpeg (Libjpeg-9b)

	Covered Paths #	Imported inputs	# of bugs found
AFL	3,476	N.A.	114 Left shift overflow 6 buffer overflow
ConcFuzzer	4,421	89	128 Left shift overflow 7 buffer overflow

Table 1. AFL vs ConcFuzzer vs KLEE results after 24 hours execution. N.A. means AFL and KLEE do not have the feature of importing inputs from concolic executor to AFL. # of generated ktests is used to denote the covered paths # in KLEE

djpeg (Libjpeg-9b)

	Covered Paths #	Imported inputs	# of bugs found
AFL	3,476	N.A.	114 Left shift overflow 6 buffer overflow
ConcFuzzer	4,421	89	128 Left shift overflow 7 buffer overflow
KLEE*	61	N.A.	0

Table 1. AFL vs ConcFuzzer vs KLEE results after 24 hours execution. N.A. means AFL and KLEE do not have the feature of importing inputs from concolic executor to AFL. # of generated ktests is used to denote the covered paths # in KLEE

Tcpdump (libpcap)

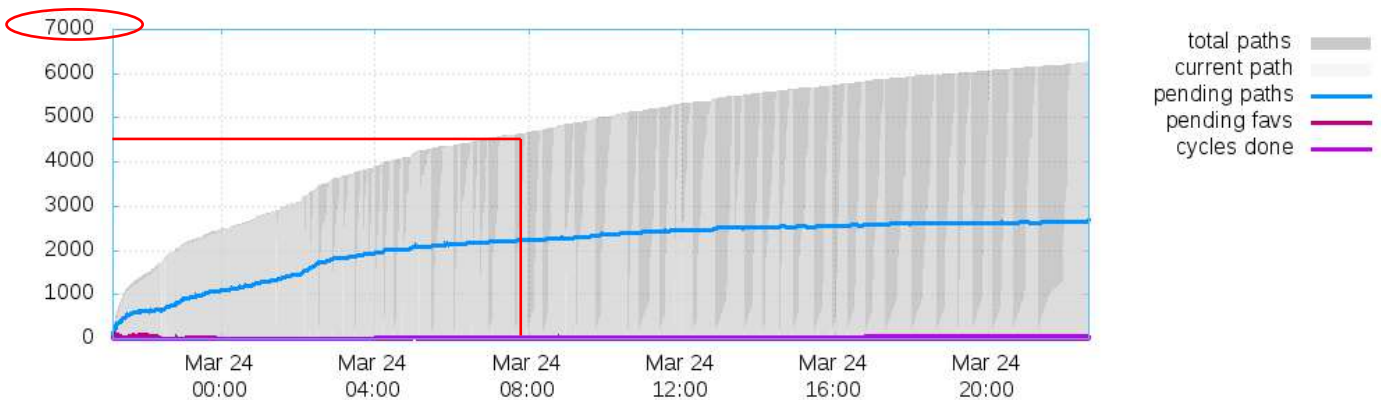


Fig 3. concfuzzer's 24 hours running results for Tcpdump

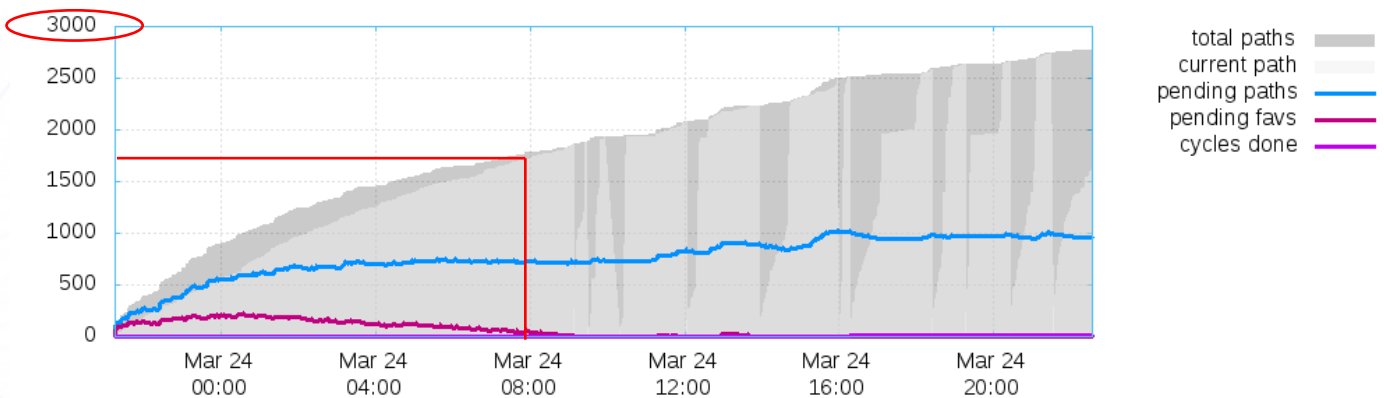


Fig 4. AFL's 24 hours running results for Tcpdump.

Tcpdump (libpcap)

	Covered Paths #	Imported inputs	# of bugs found
AFL	2,773	N.A.	2 left shift overflow

Table 2. AFL vs ConcFuzzer vs KLEE results after 24 hours execution. N.A. means AFL and KLEE do not have the feature of importing inputs from concolic executor to AFL. # of generated ktests is used to denote the covered paths # in KLEE

Tcpdump (libpcap)

	Covered Paths #	Imported inputs	# of bugs found
AFL	2,773	N.A.	2 left shift overflow
ConcFuzzer	6,300	101	11 left shift overflow

Table 2. AFL vs ConcFuzzer vs KLEE results after 24 hours execution. N.A. means AFL and KLEE do not have the feature of importing inputs from concolic executor to AFL. # of generated ktests is used to denote the covered paths # in KLEE

Tcpdump (libpcap)

	Covered Paths #	Imported inputs	# of bugs found
AFL	2,773	N.A.	2 left shift overflow
ConcFuzzer	6,300	101	11 left shift overflow
KLEE*	57	N.A.	0

Table 2. AFL vs ConcFuzzer vs KLEE results after 24 hours execution. N.A. means AFL and KLEE do not have the feature of importing inputs from concolic executor to AFL. # of generated ktests is used to denote the covered paths # in KLEE

MbedTLS X509 Certificate Parser

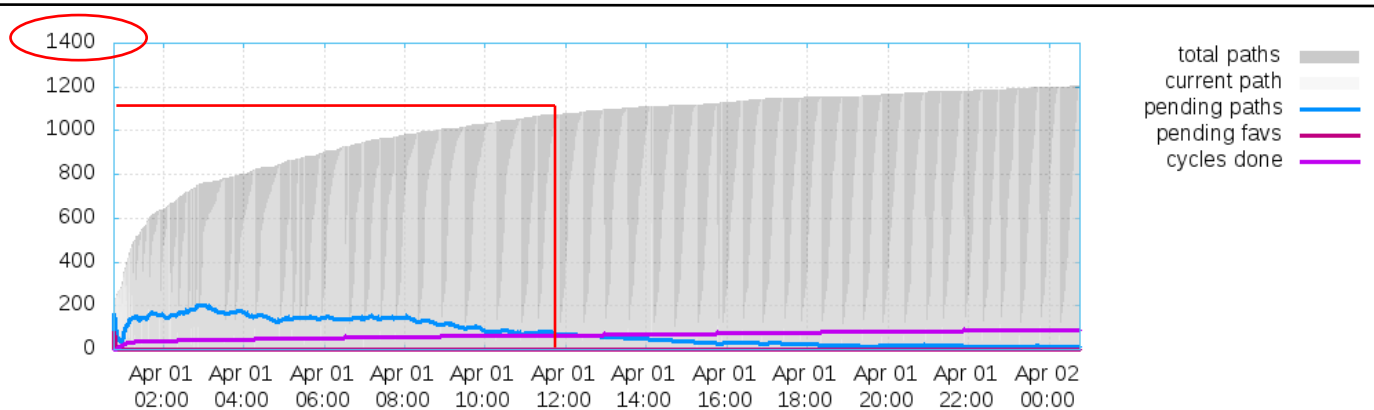


Fig 5. concfuzzer's 24 hours running results for mbedtls x509 certificate parser

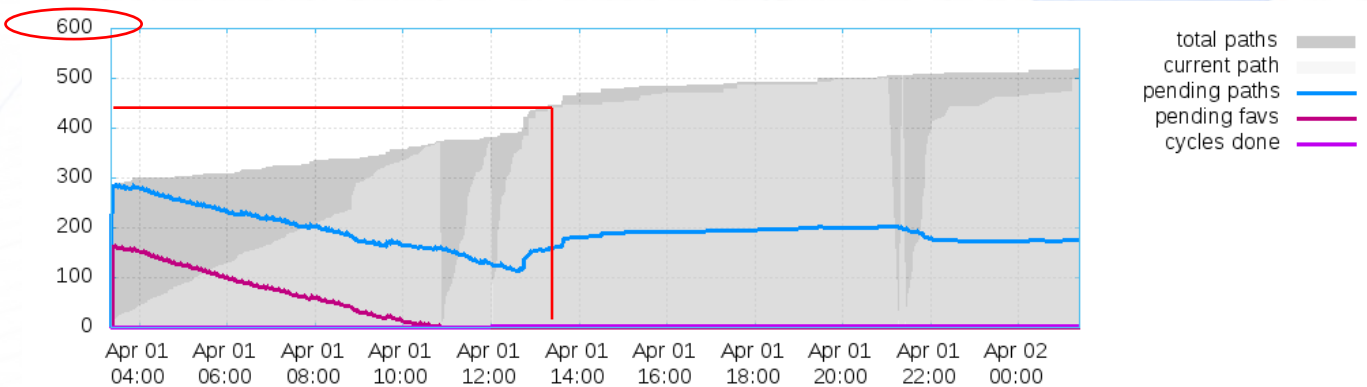


Fig 6. AFL's 24 hours running results for mbedtls x509 certificate parser

MbedTLS X509 Certificate Parser

	Covered Paths #	Imported inputs	# of bugs found
AFL	518	N.A.	0

Table 3. AFL vs ConcFuzzer vs KLEE results after 24 hours execution. N.A. means AFL do not have the feature of importing inputs from concolic executor to AFL. # of generated ktests is used to denote the covered paths # in KLEE

MbedTLS X509 Certificate Parser

	Covered Paths #	Imported inputs	# of bugs found
AFL	518	N.A.	0
ConcFuzzer	1,204	1	0

Table 3. AFL vs ConcFuzzer vs KLEE results after 24 hours execution. N.A. means AFL do not have the feature of importing inputs from concolic executor to AFL. # of generated ktests is used to denote the covered paths # in KLEE

MbedTLS X509 Certificate Parser

	Covered Paths #	Imported inputs	# of bugs found
AFL	518	N.A.	0
ConcFuzzer	1,204	1	0
KLEE*	6,193	N.A.	0

Table 3. AFL vs ConcFuzzer vs KLEE results after 24 hours execution. N.A. means AFL do not have the feature of importing inputs from concolic executor to AFL. # of generated ktests is used to denote the covered paths # in KLEE

Conclusion

- Standing on the shoulders of giants:
  - Clang / LLVM
  - AFL / KLEE
  - Z3 / STP

- Invent a sanitizer guided hybrid fuzzing solution to improve the bug finding capability while guaranteeing the high coverage

- Demonstrate an effective sanitizer guided hybrid fuzzing system named ConcFuzzer to analyze real world C/C++ applications and catch bugs

THANKS