Directed Program Analysis: from Suspicion to Witnesses

Kihong Heo
KAIST

Dagstuhl Seminar 25421
October 2025

KAIST

Programming Systems Laboratory

Acknowledgements

Grad Students @ KAIST

Tae Eun Kim **Bongjune Jang** **Jaehoon Jang** **Dongjae Lee**

Geon Park **Yeonhee Ryou** **Sujin Jang**

Collaborators

Sang Kil Cha
KAIST

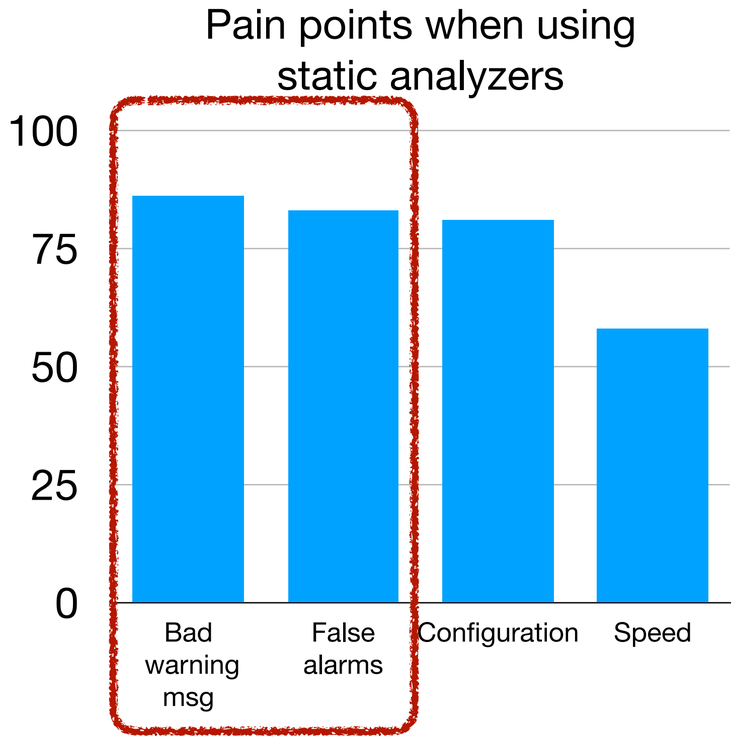
Juneyoung Lee
AWS

Jaeseung Choi
Sogang Univ.

Q: How to explain the presence of bugs found by static analysis?

Needs for Witnesses

- Complexity of SW and bugs ⬆: difficulty of understanding analysis reports ⬆



*survey of 100 Microsoft developers

> *"Although their interviewees felt that using static analysis is **beneficial**, there are certain barriers in their use, like **false positives, poorly presented warnings**, …"*
>
> - Christakis et al., "*What Developers Want and Need from Program Analysis: An Empirical Study*", ASE'16

> *"More than half of the tools have **too poor warning messages**. … Hardly any tools give information about **why they evaluate something as an issue** (giving traces or paths, internal reasoning)."*
>
> - Nachtigall et al., "*A Large-Scale Study of Usability Criteria Addressed by Static Analysis Tools*", ISSTA'22

Why should humans inspect alarms?

Let machines do it!

Directed Program Analysis

- Goal: generating error-triggering inputs at specific suspicious program points
  - E.g., static analysis alarms, recently changed code
- Idea: directed input generation guided by static analysis
  - Different techniques for different input formats (fuzzing or program synthesis)

DAFL [Security'23]

```
0000000    5746 0853 ebde 71f0 0500 0a7f 315b 12c7
0000010    5e7f 8000 2101 00e4 0600 1077 009e 0080
0000020    f000 0071 9b05 530a 6731 7f12 005e 0180
0000030    e421 0000 7206 9e10 8000 0000 a5fb 0314
0000040    0315 1a03 6999 160f 0000 6700 652e 9d00
0000050    0015 0006 0323 140d 1503 0303 031a 2169
0000060    0000 9d67 0015 ffe5 0322 100d 009e 0010
0000070    1500 0600
```

application SW: binary files
(e.g., CoreUtils)

UnitCon [FSE'25]

```
public void test() throws Exception {
  Adapter recv = new Adapter();
  Visitor visitor = new Visitor();
  recv.setVisitor(visitor);
  Select select = new Select();
  recv.visit(select);
}
```

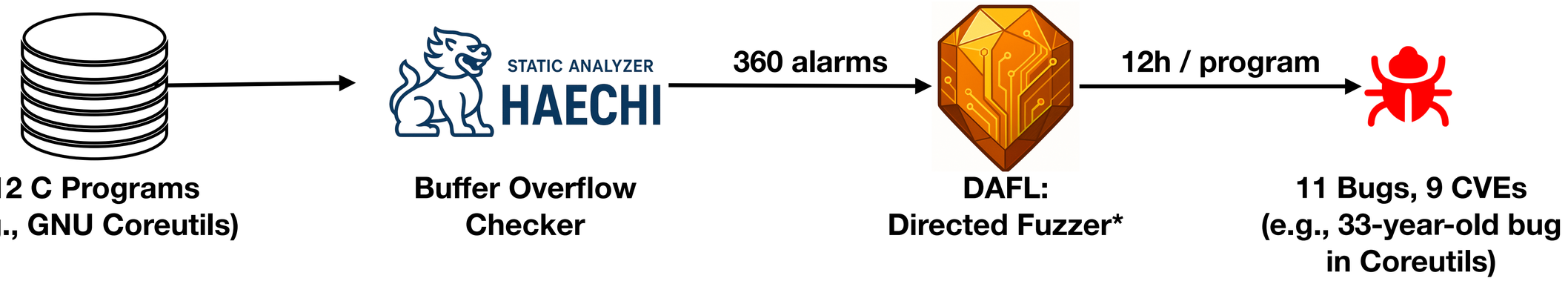
libraries: unit tests
(e.g., JSQLParser)

Optimuzz [PLDI'25]

```
define i32 @src(i32 %conv) {
  %1 = shl i32 %conv, 16
  %conv1 = ashr i32 %1, 16
  %cmp2 = icmp ugt i32 %conv1, 65535
  %cond = select i1 %cmp2, i32 %conv1, i32 0
  ret i32 %cond
}
```

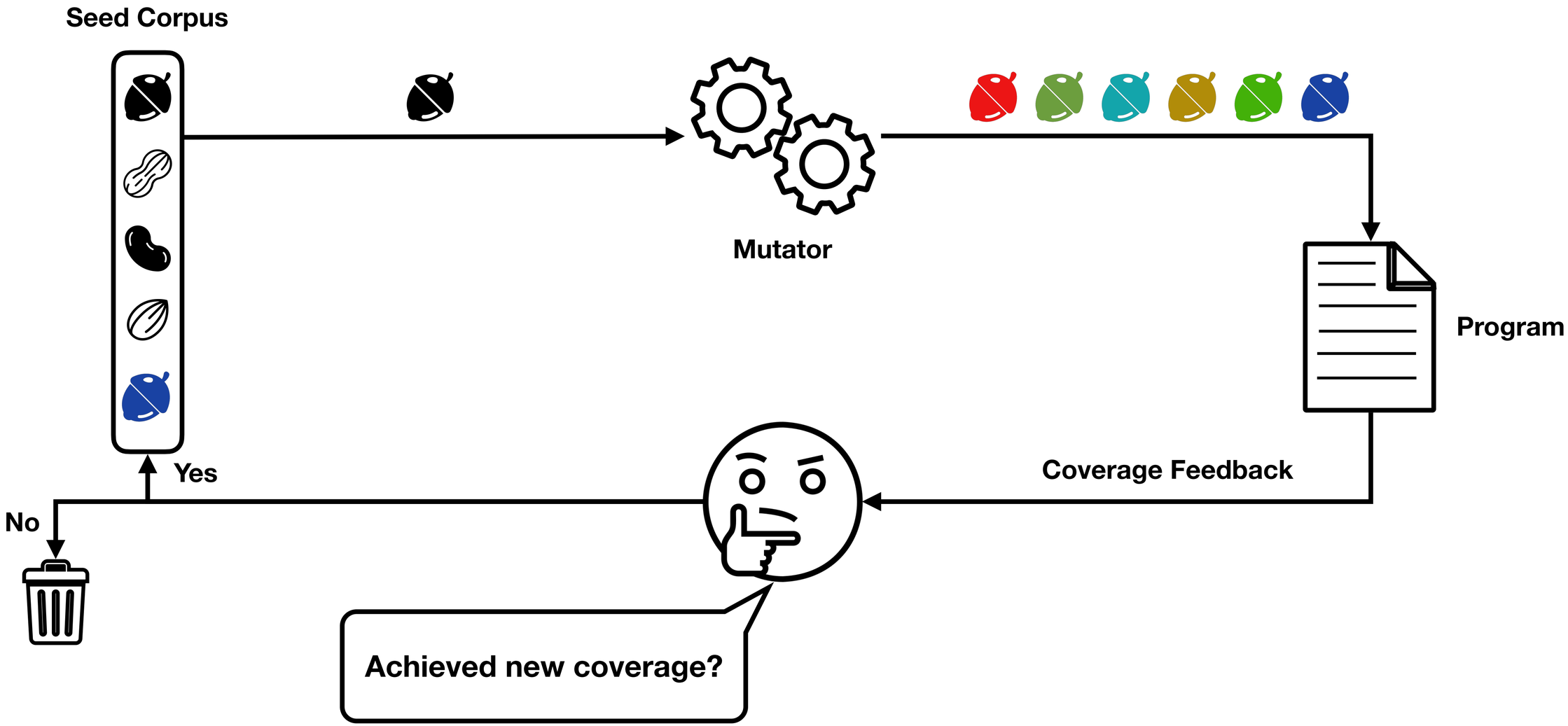
compilers: source programs
(e.g., LLVM)

Case Study: Directed Fuzzing

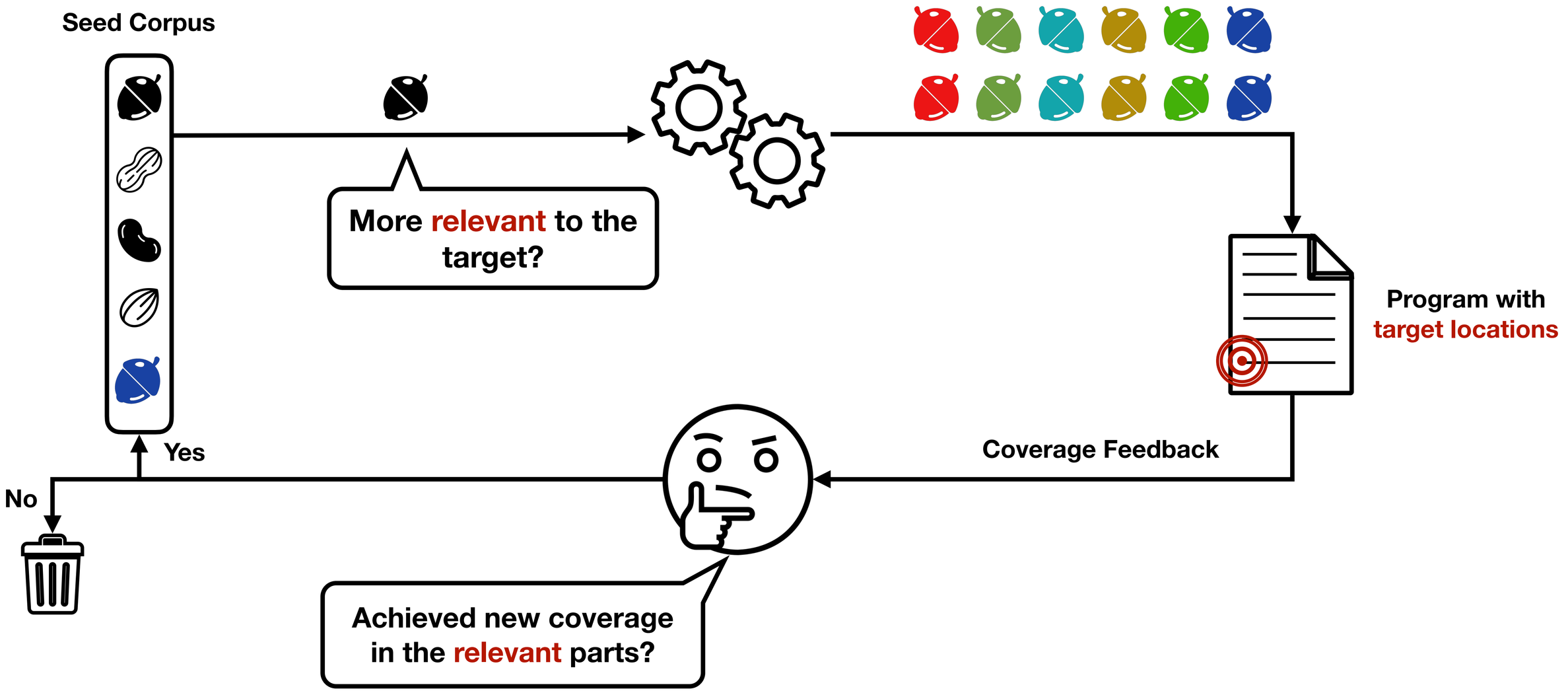


*DAFL: Directed Grey-box Fuzzing Guided by Data Dependency, USENIX Security, 2023

Overview of Grey-box Fuzzing

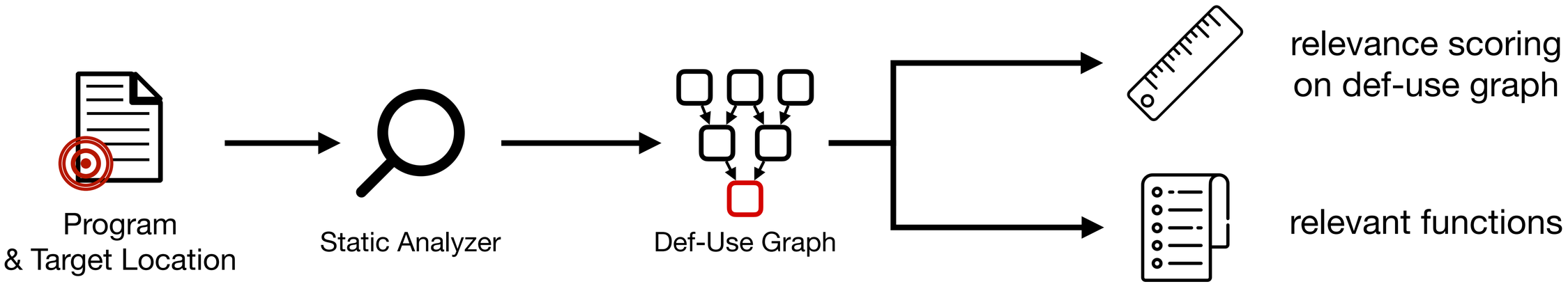


Toward Direct Grey-box Fuzzing

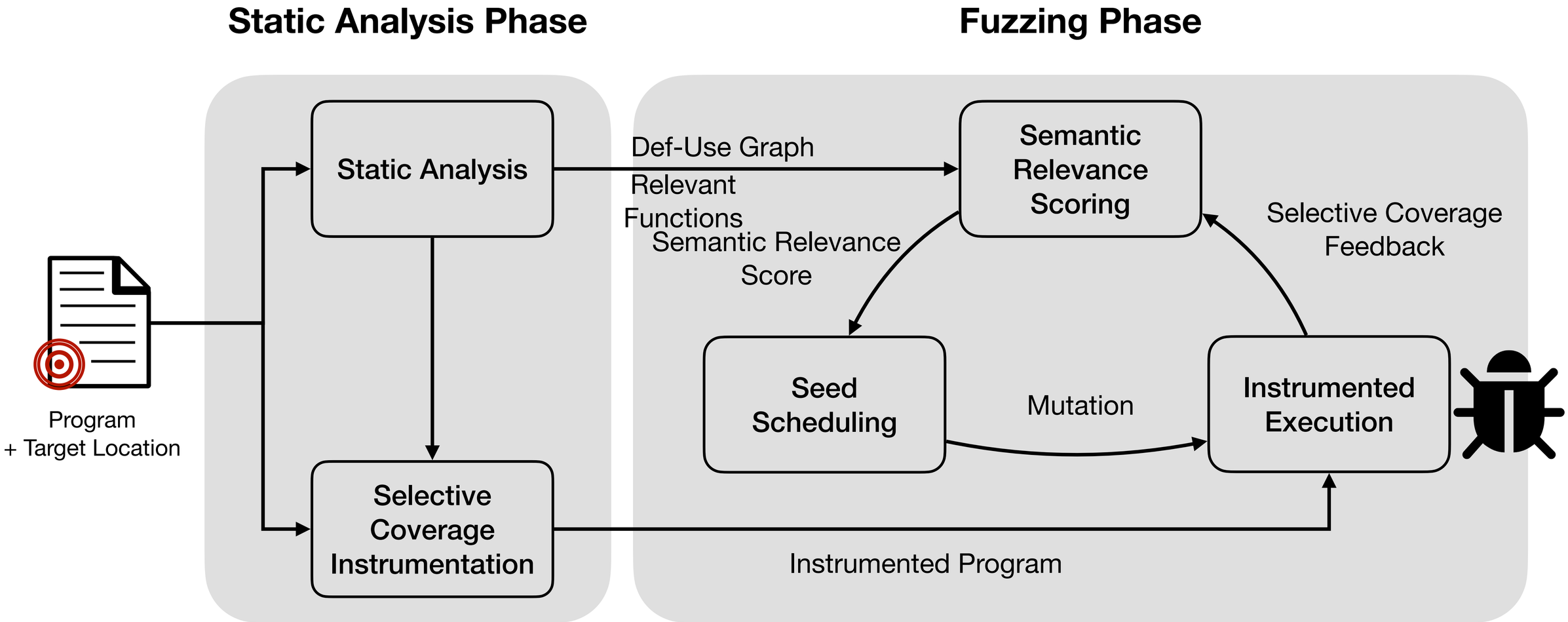


Static Analysis for Directed Fuzzing

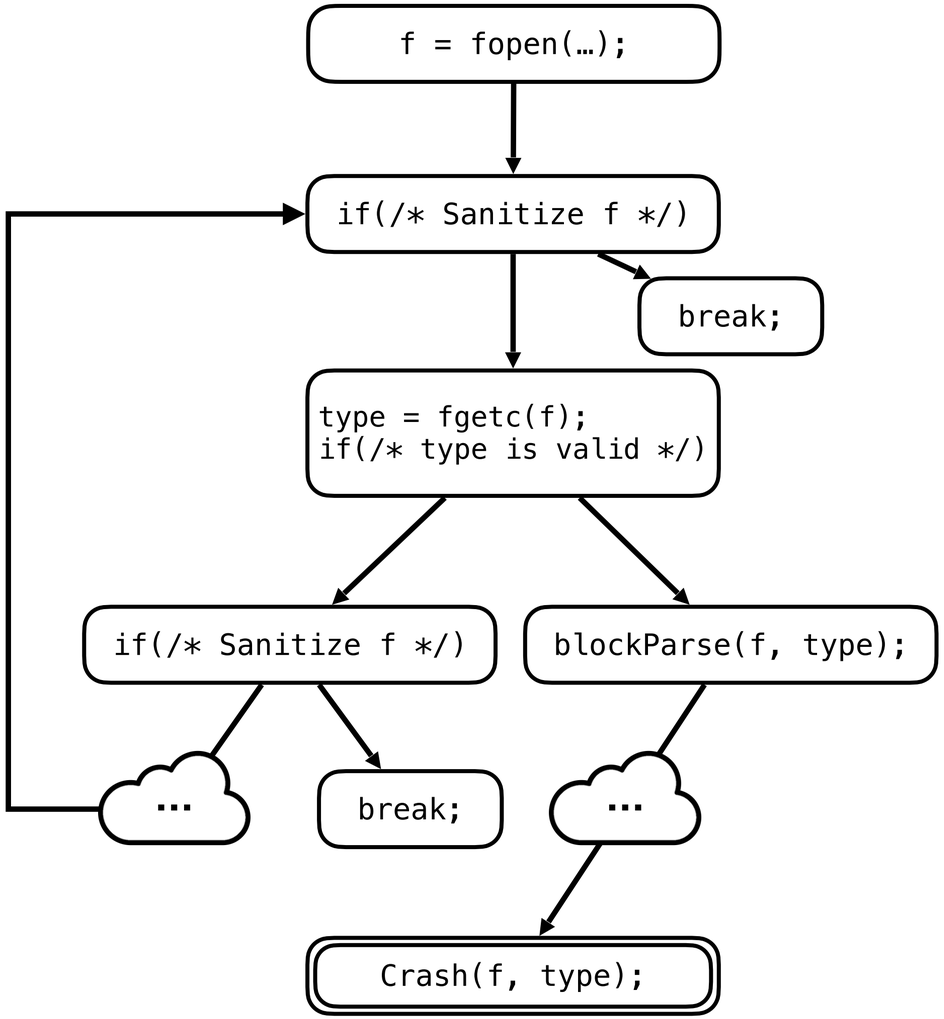
- Two important questions on relevance
  - How to measure the semantic relevance between seed inputs and the target? (mutation)
  - How to estimate semantically relevant parts to the target? (coverage)
- Idea: data dependency estimated by static analysis



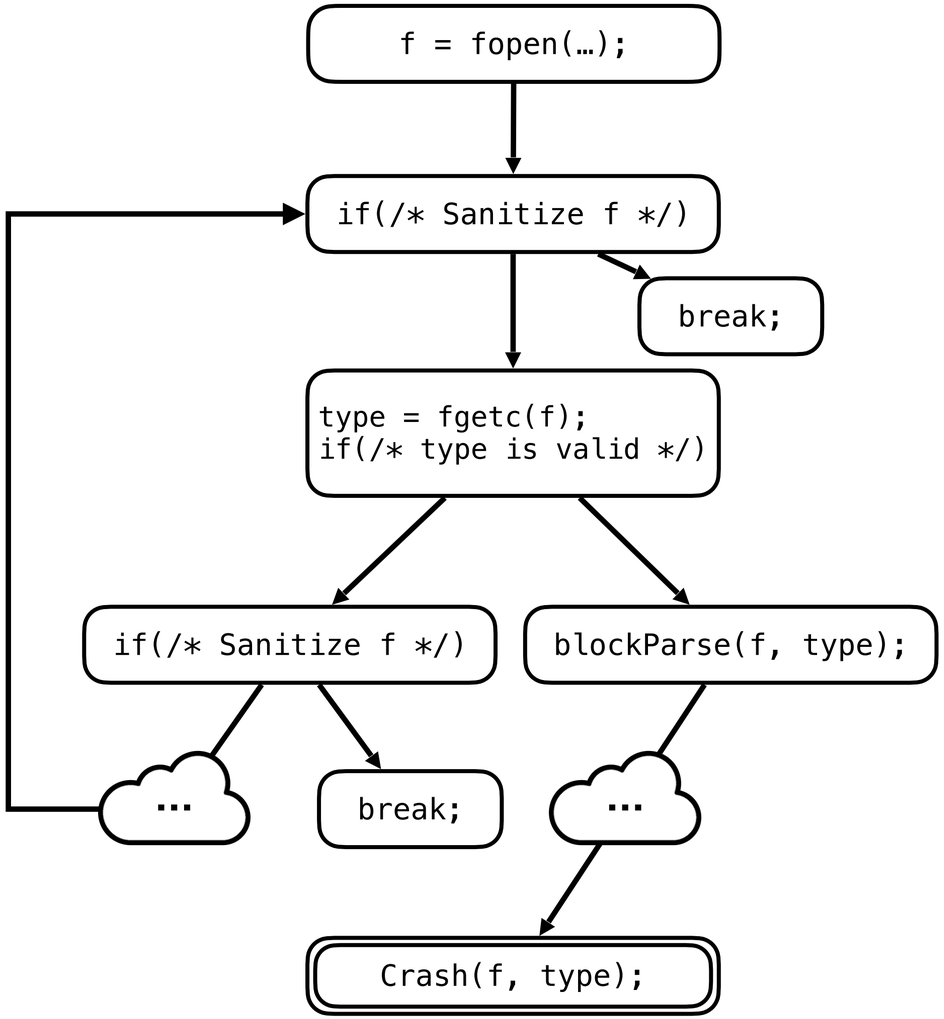
DAFL: Directed Fuzzing System

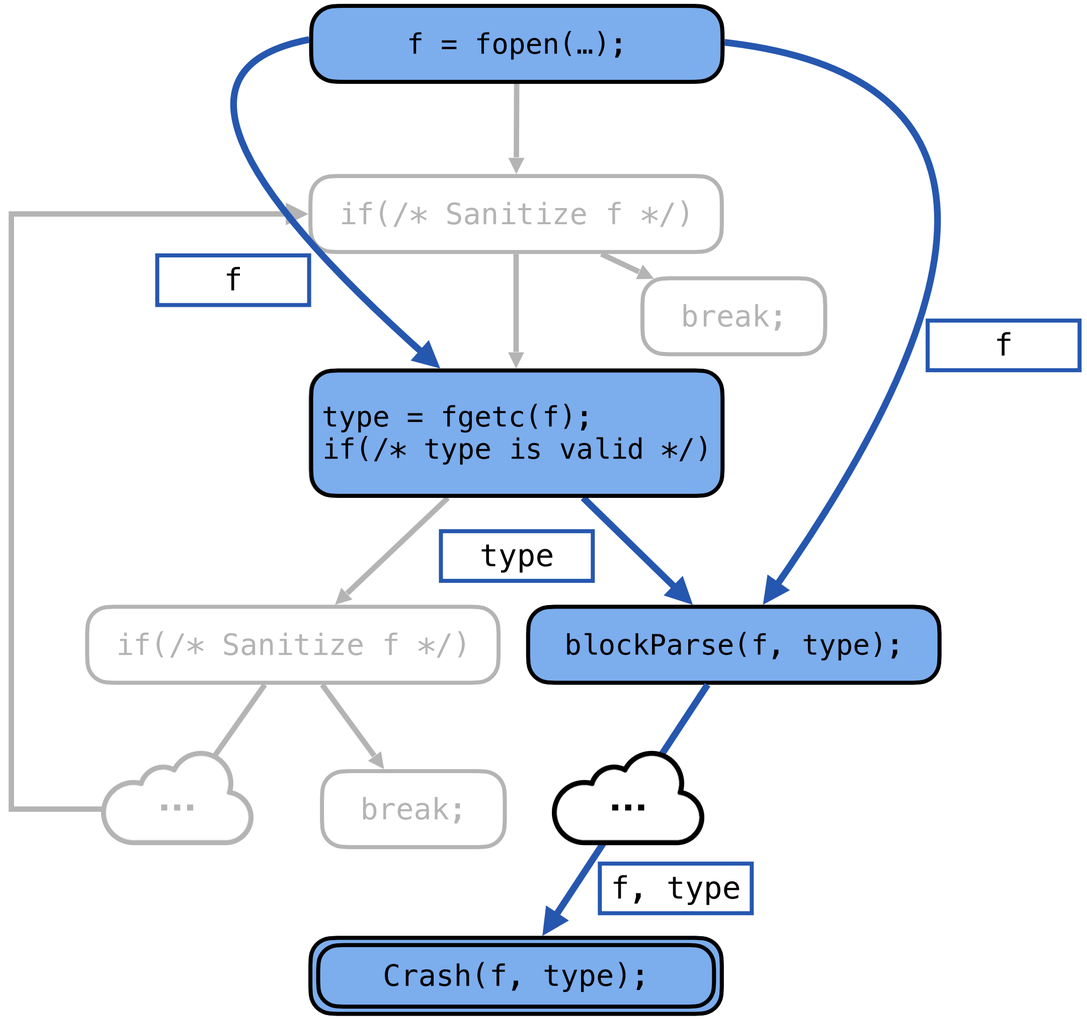


Seed Relevance via DUG

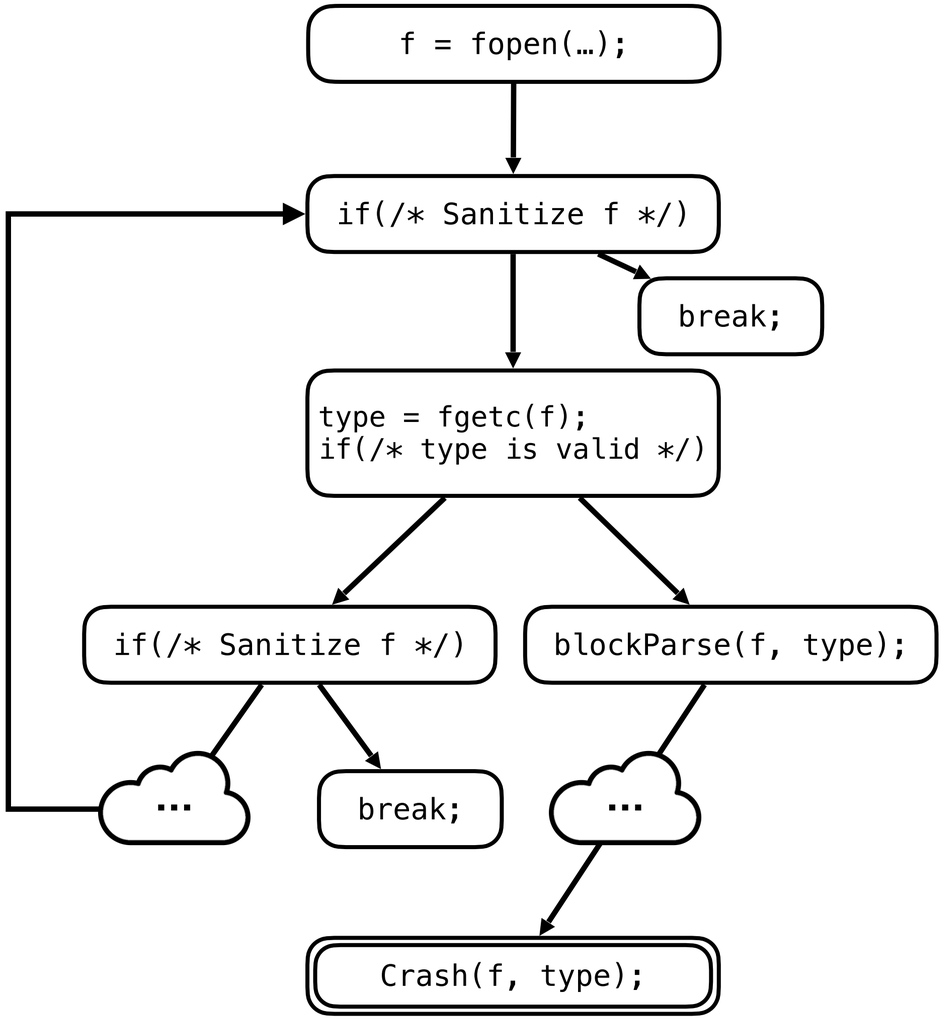


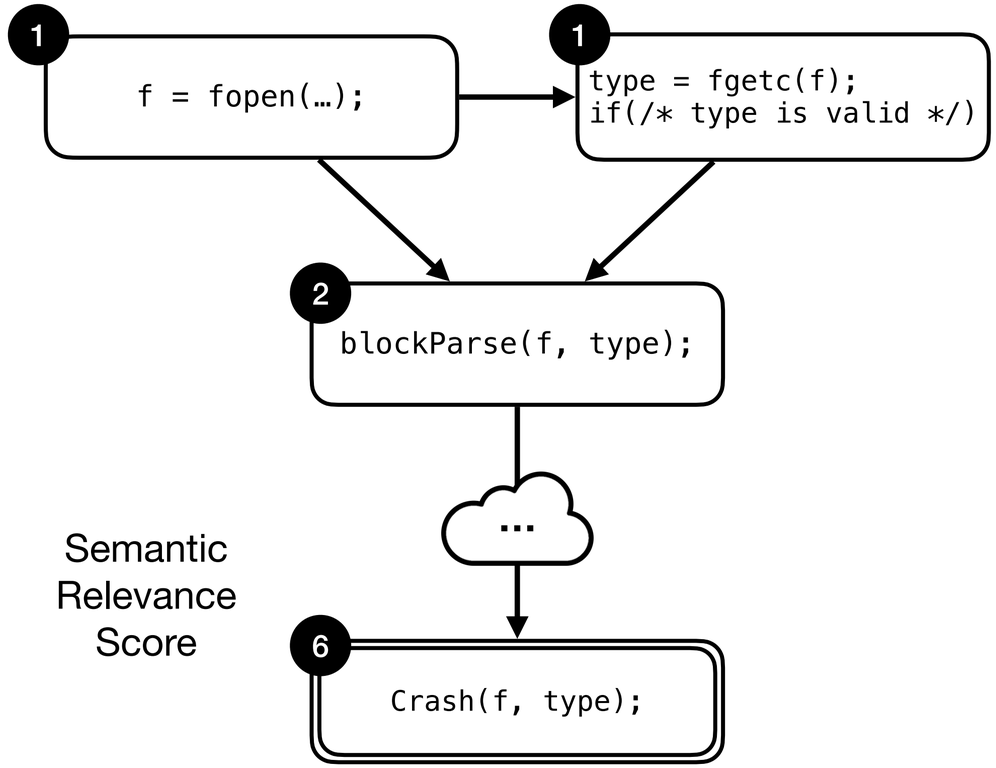
Seed Relevance via DUG





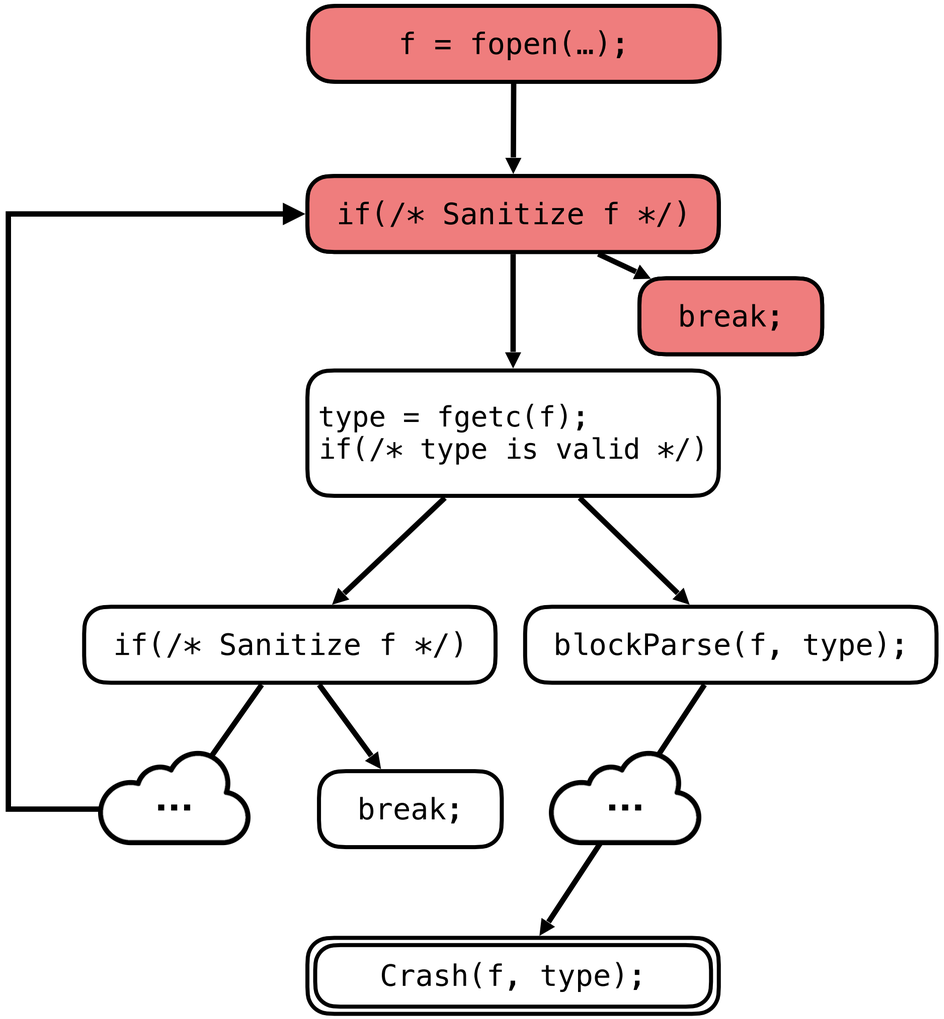
Seed Relevance via DUG

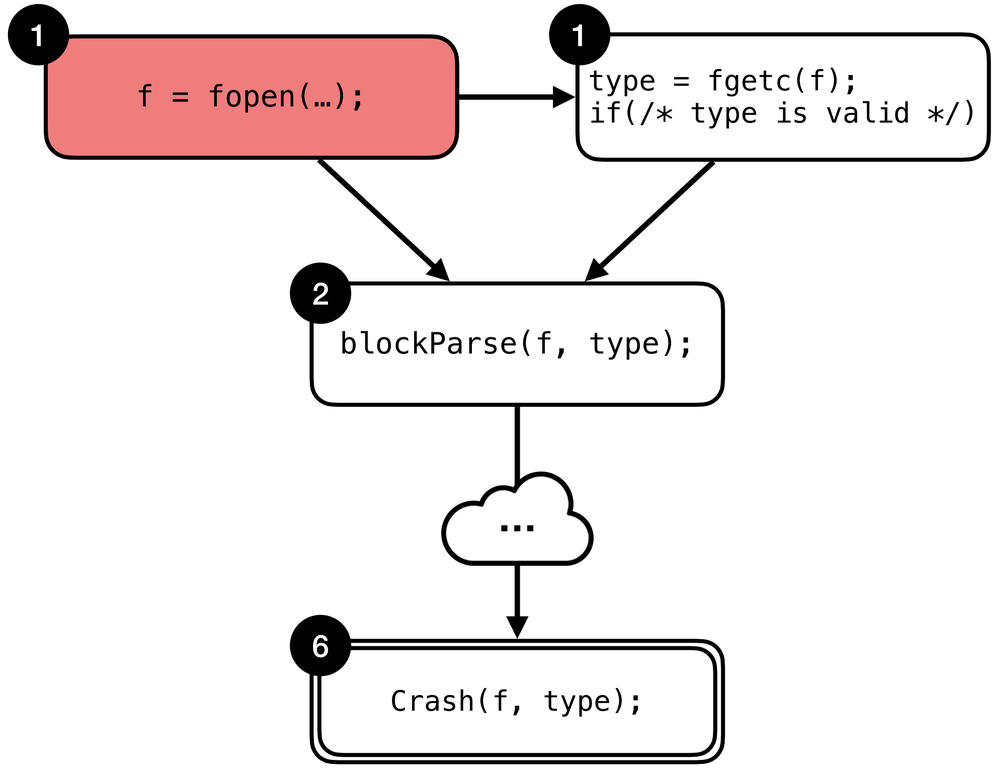




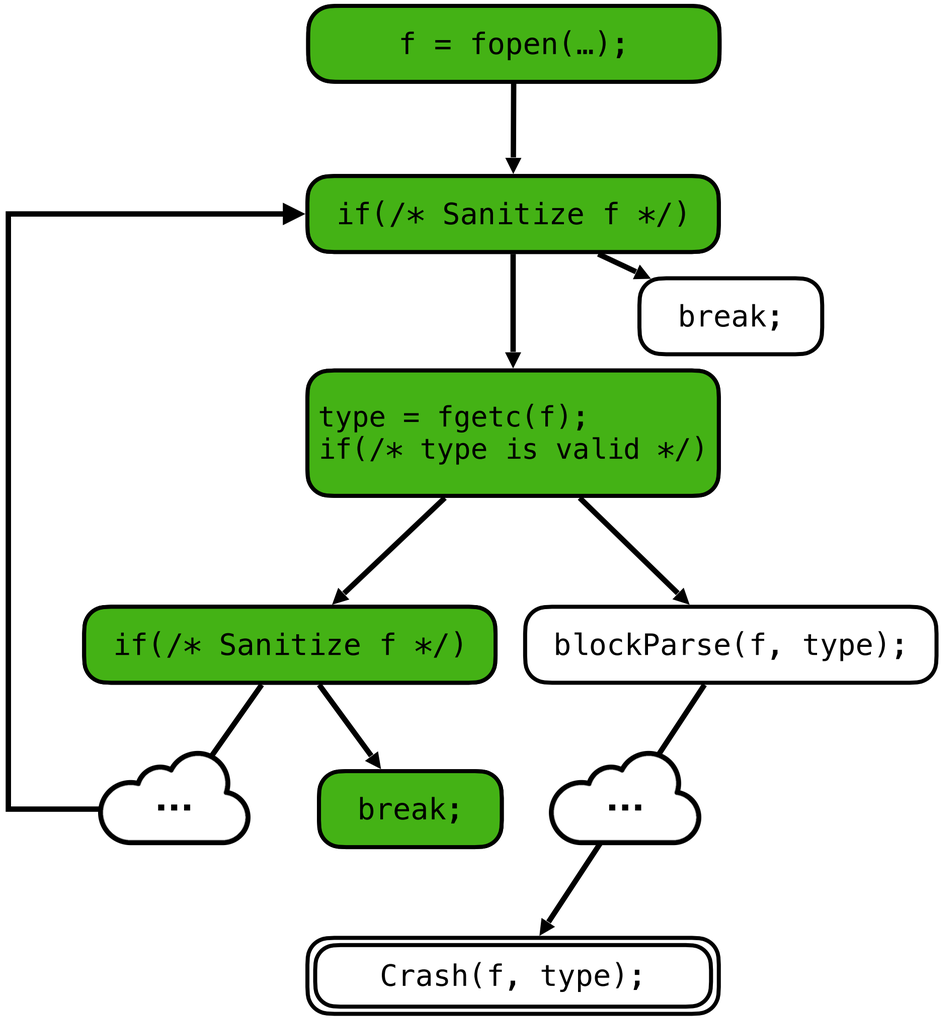
Seed Relevance via DUG

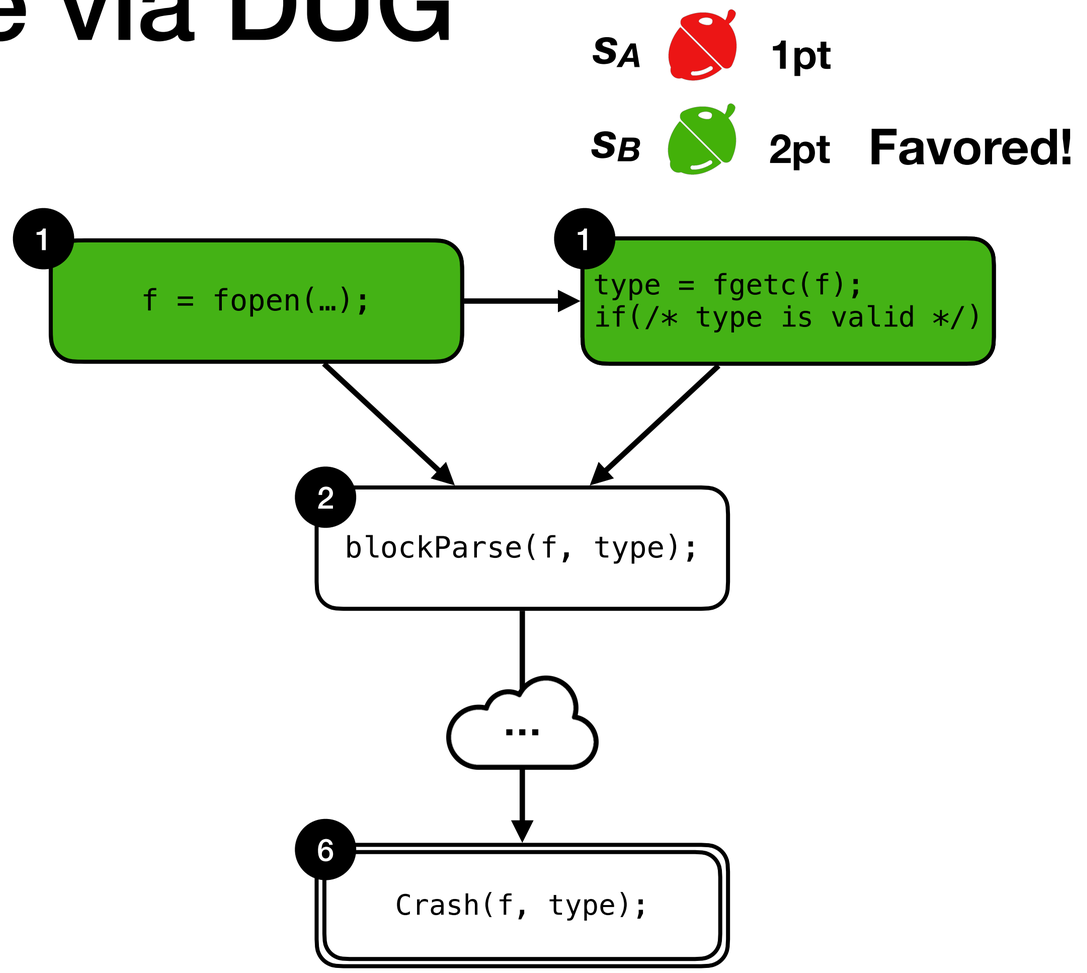
SA 1pt





Seed Relevance via DUG



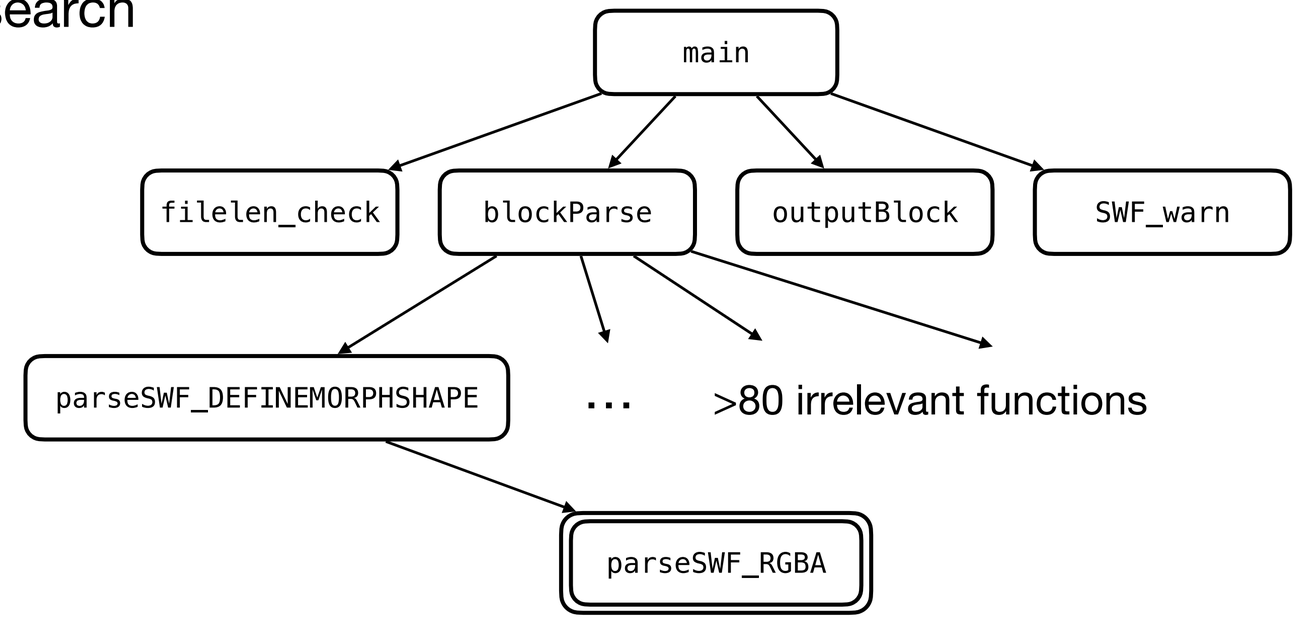


Selective Coverage Feedback via DUG

- Coverage feedback only from relevant functions
  - Functions that the DUG passes through to the target
- Enable target-oriented guided search

Selective Coverage Feedback via DUG

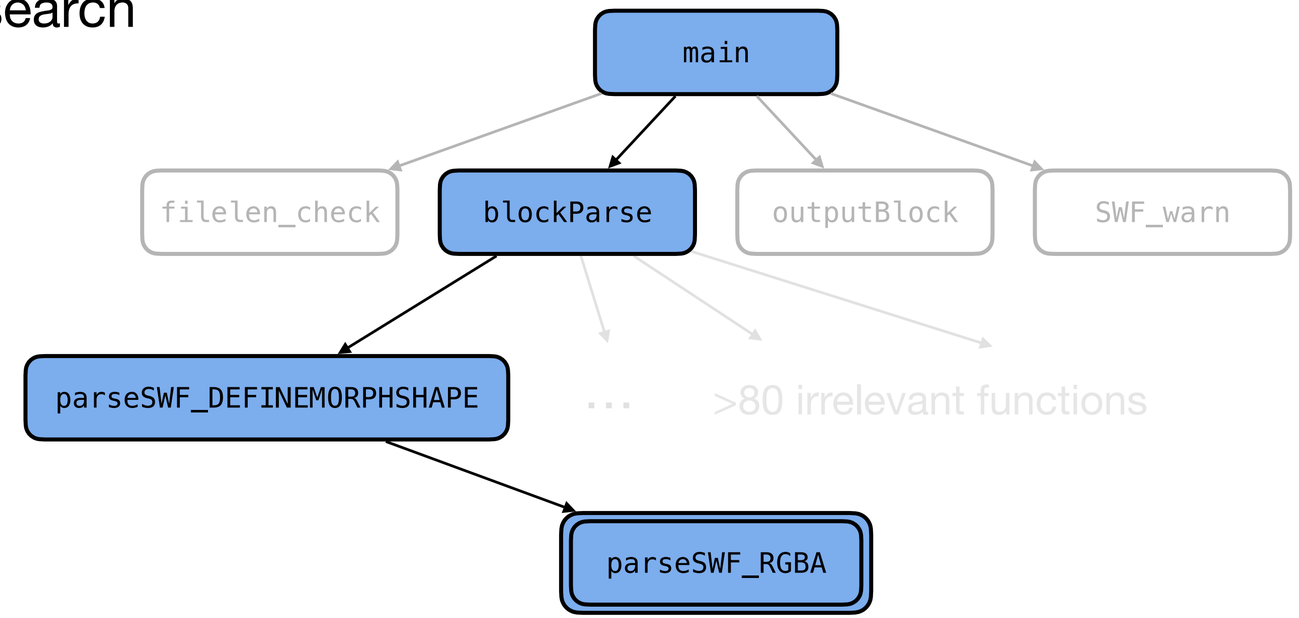
- Coverage feedback only from relevant functions
  - Functions that the DUG passes through to the target
- Enable target-oriented guided search



*swftophp-4.8

Selective Coverage Feedback via DUG

- Coverage feedback only from relevant functions
  - Functions that the DUG passes through to the target
- Enable target-oriented guided search



*swftophp-4.8

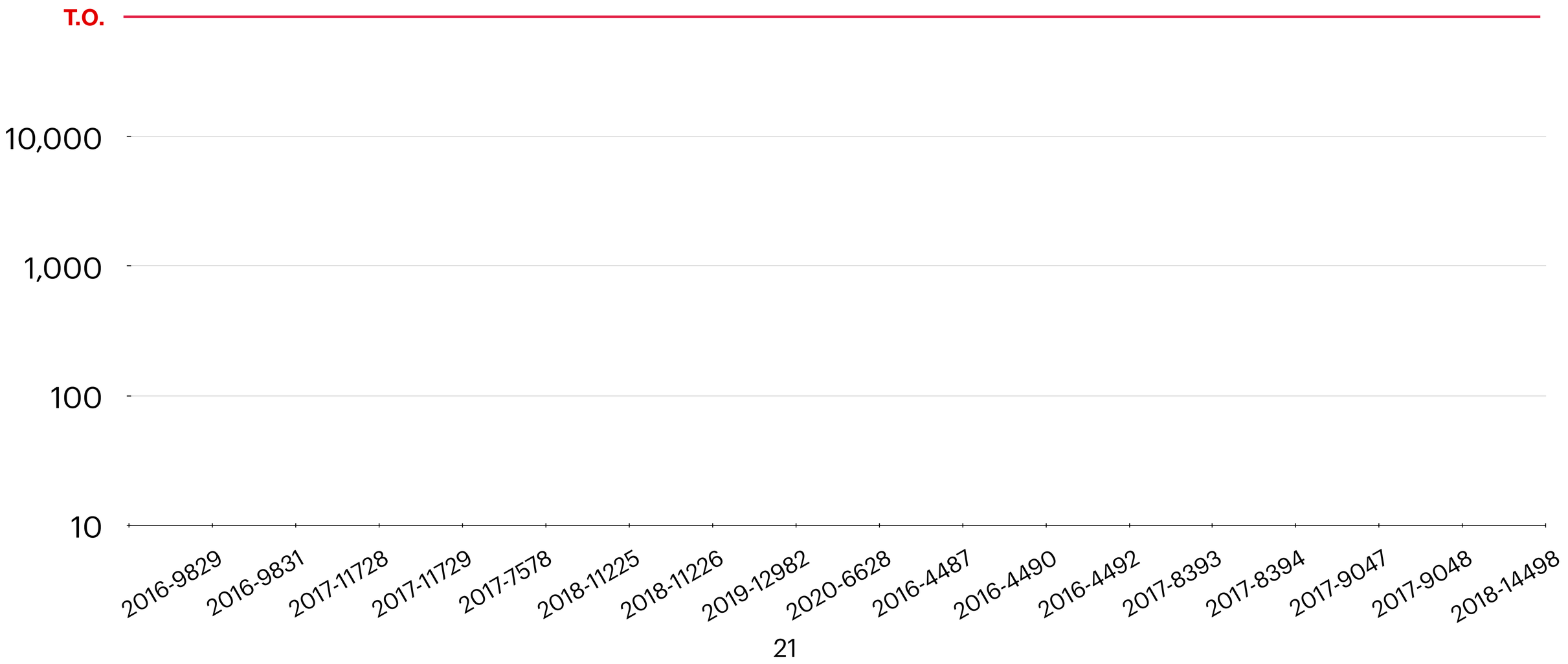
Implementation

- Target: C programs
- Dependency analysis: flow-insensitive, selectively context-sensitive, interval & pointer domains
- Slicing: Thin slicing*
  - Tracking value read/write, excluding pointer manipulation

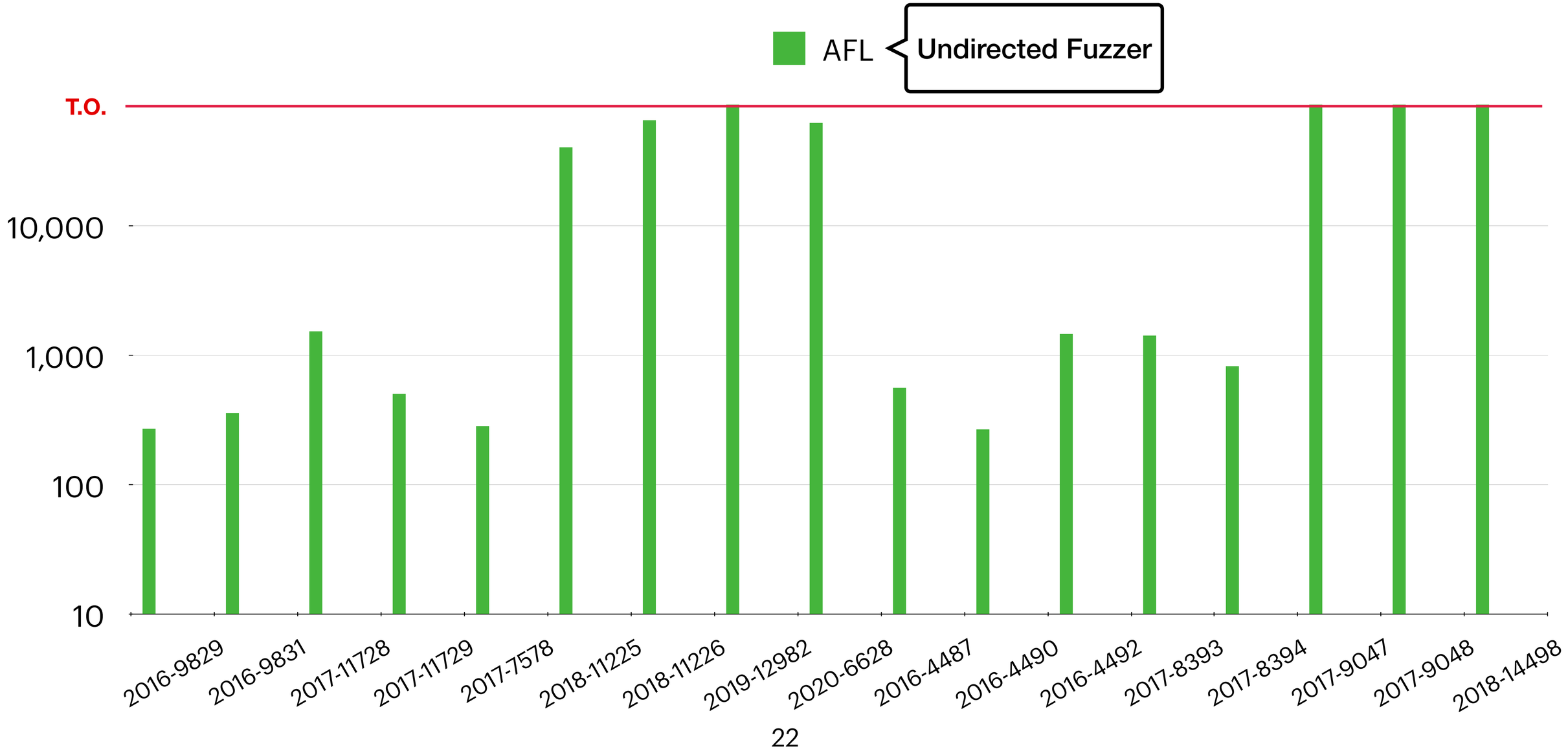
```
x = f();
y = g();
p = &y;
...
x + *p;  // target
```

*Sridharan et al, Thin Slicing, PLDI 2007

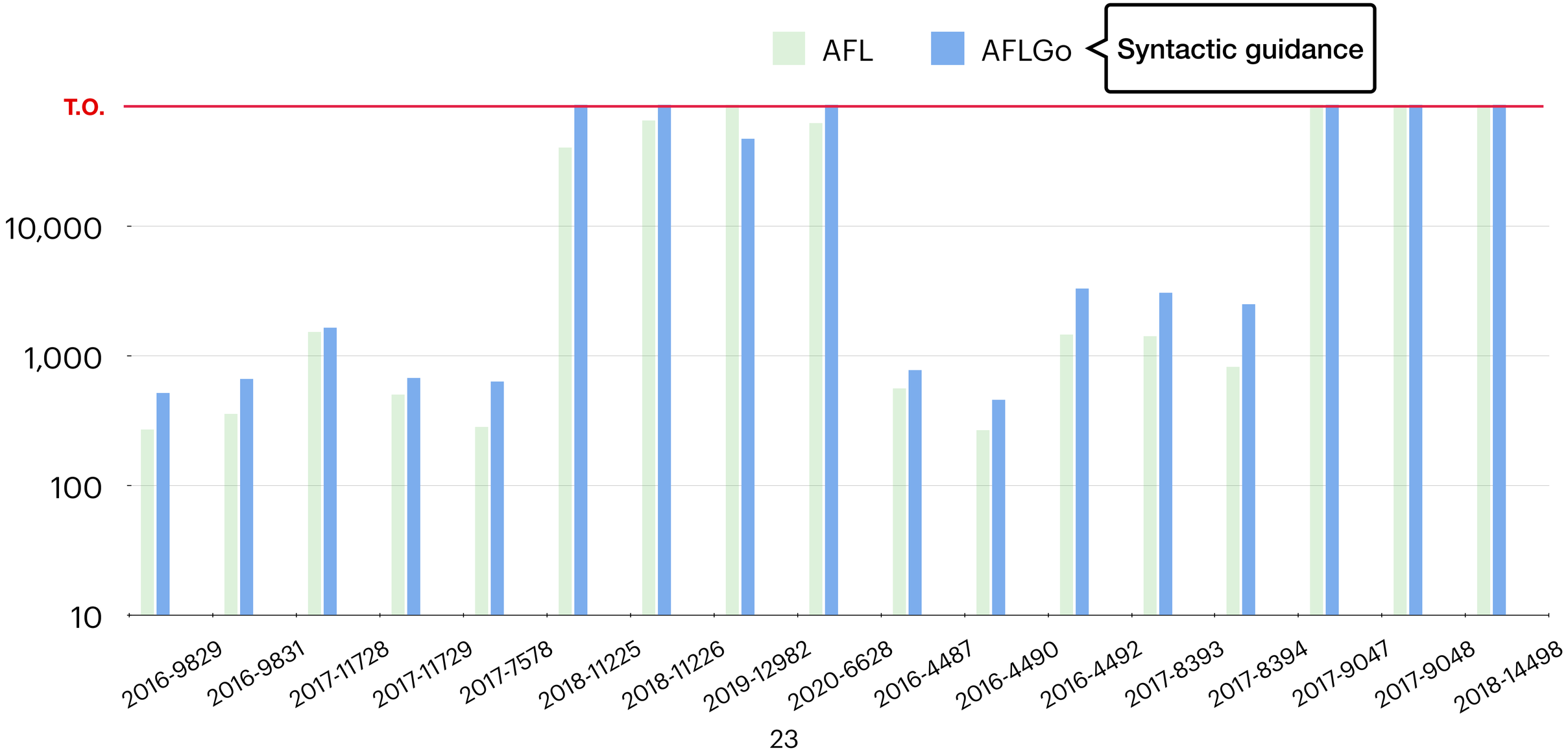
Crash Reproduction



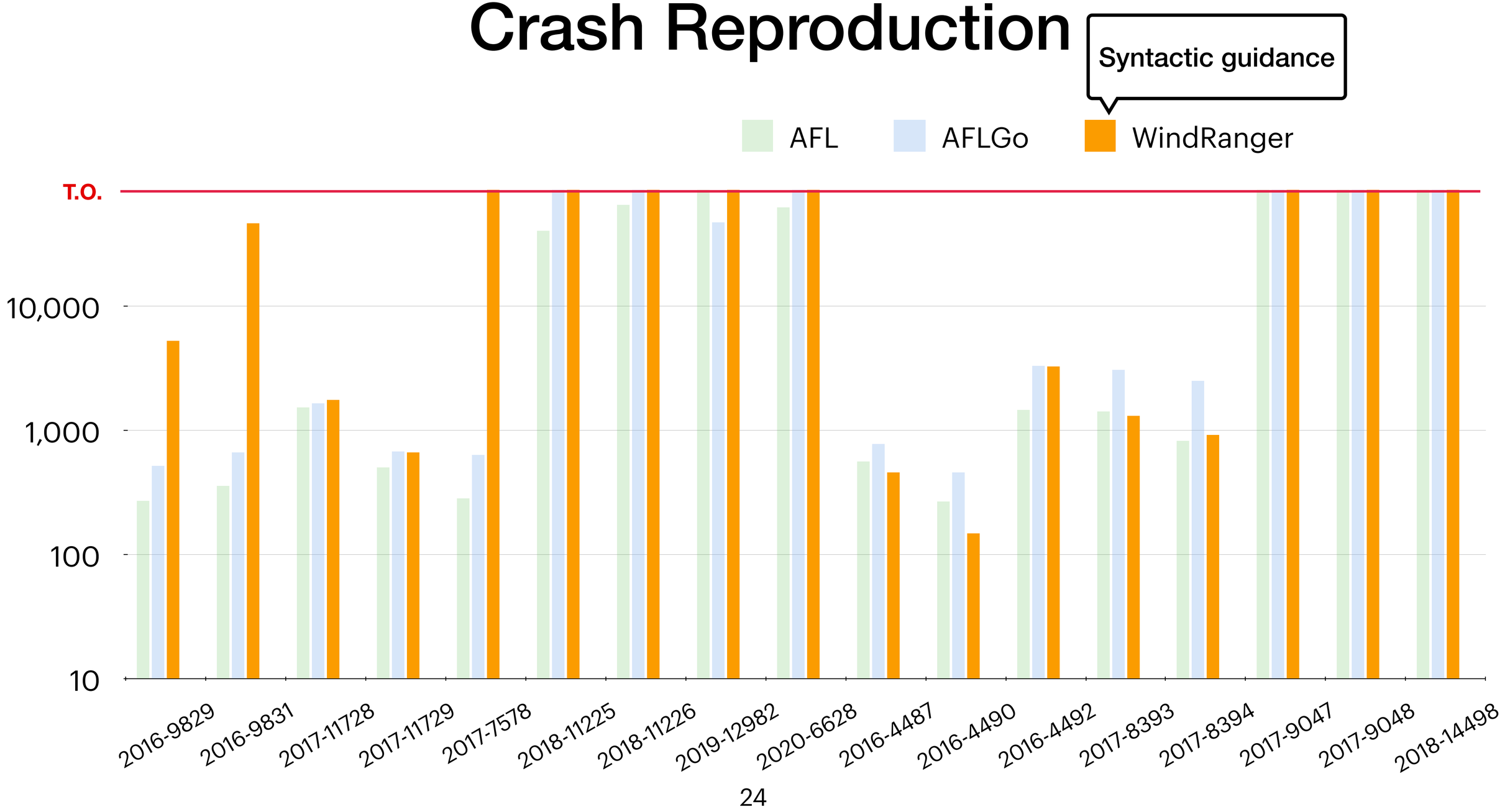
Crash Reproduction



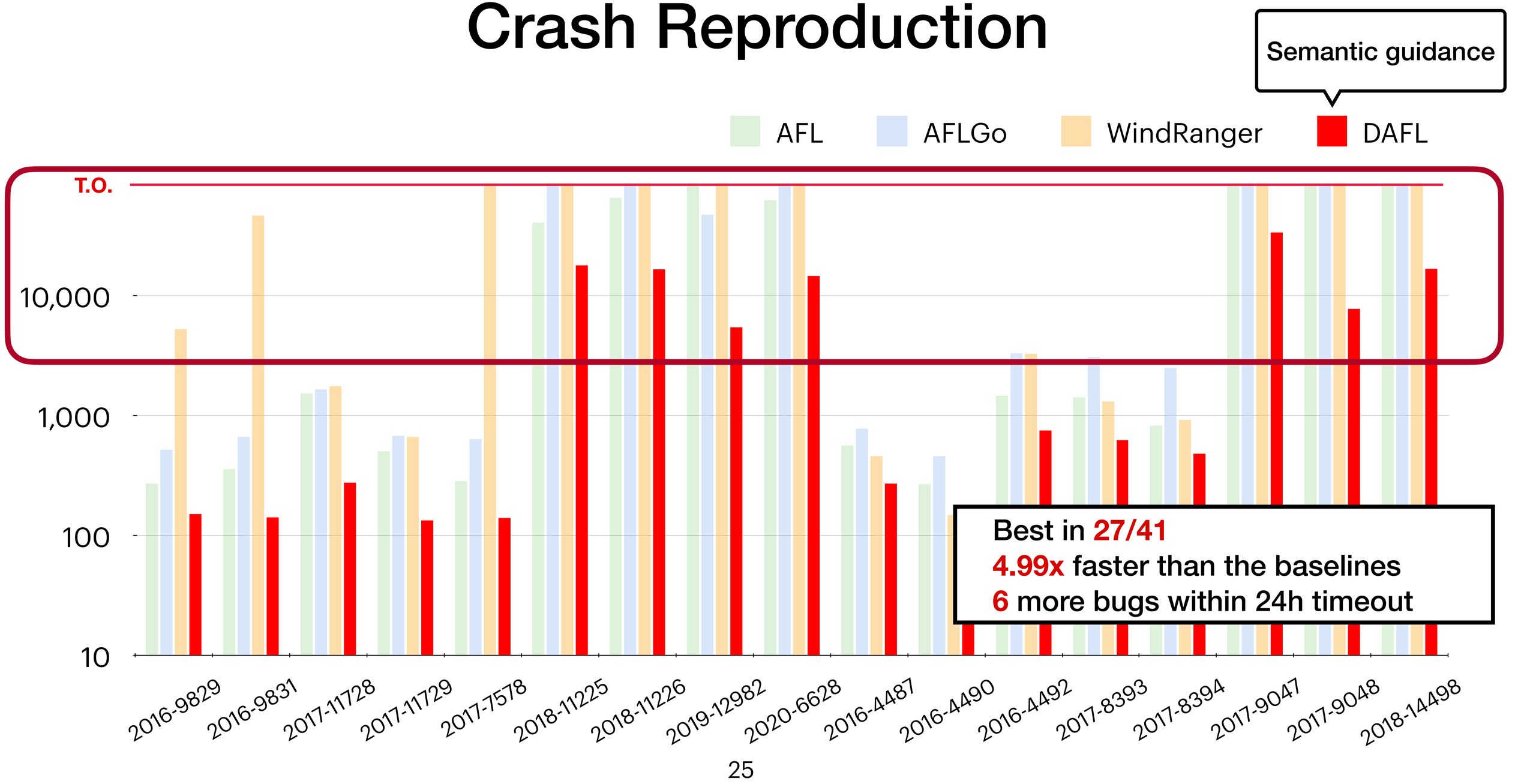
Crash Reproduction



Crash Reproduction



Crash Reproduction



Directed Program Analysis

- Goal: generating error-triggering inputs at specific suspicious program points
  - E.g., static analysis alarms, recently changed code
- Idea: directed input generation guided by static analysis
  - Different techniques for different input formats (fuzzing or program synthesis)

DAFL [Security'23]

```
0000000    5746 0853 ebde 71f0 0500 0a7f 315b 12c7
0000010    5e7f 8000 2101 00e4 0600 1077 009e 0080
0000020    f000 0071 9b05 530a 6731 7f12 005e 0180
0000030    e421 0000 7206 9e10 8000 0000 a5fb 0314
0000040    0315 1a03 6999 160f 0000 6700 652e 9d00
0000050    0015 0006 0323 140d 1503 0303 031a 2169
0000060    0000 9d67 0015 ffe5 0322 100d 009e 0010
0000070    1500 0600
```

application SW: binary files
(e.g., CoreUtils)

UnitCon [FSE'25]

```
public void test() throws Exception {
  Adapter recv = new Adapter();
  Visitor visitor = new Visitor();
  recv.setVisitor(visitor);
  Select select = new Select();
  recv.visit(select);
}
```

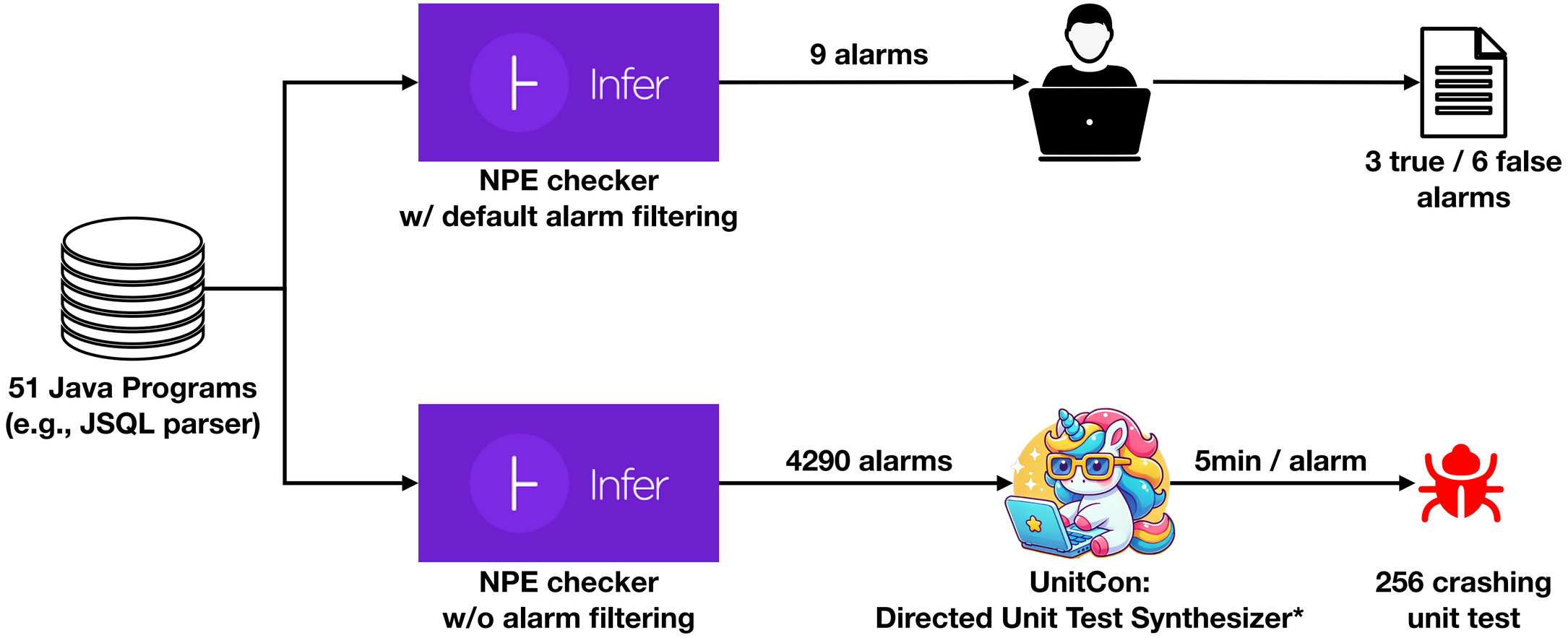
libraries: unit tests
(e.g., JSQLParser)

Optimuzz [PLDI'25]

```
define i32 @src(i32 %conv) {
  %1 = shl i32 %conv, 16
  %conv1 = ashr i32 %1, 16
  %cmp2 = icmp ugt i32 %conv1, 65535
  %cond = select i1 %cmp2, i32 %conv1, i32 0
  ret i32 %cond
}
```

compilers: source programs
(e.g., LLVM)

Case Study: Directed Unit Test Synthesis



*UnitCon: Synthesizing Targeted Unit Tests for Java Runtime Exceptions, FSE 2025

Case Study: Opt.-Directed Compiler Fuzzing

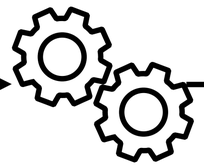
```
+ if (!(match(TrueVal, m_Add(m_Value(RemRes), m_Value(Remainder))) &&
+       match(RemRes, m_SRem(m_Value(Op), m_Specific(Remainder))) &&
+       IC.isKnownToBeAPowerOfTwo(Remainder, /*OrZero*/ true) &&
+       FalseVal == RemRes))
+   return nullptr;
+
+ Value *Add = Builder.CreateAdd(Remainder,
+                                Constant::getAllOnesValue(RemRes->getType()));
+ return BinaryOperator::CreateAnd(Op, Add);
```

Checks Optimization Condition

Target: Returns Optimized Code

LLVM

Peephole opt code



Target Selector

~4500 targets

optimuzz

Optimization-directed Fuzzer*

1hr/target

Alive2: Translation Validator

55 miscompilation bugs

*Optimization-Directed Compiler Fuzzing for Continuous Translation Validation, PLDI 2025

Summary

- Directed program analysis: from suspicion to witnesses
  - Applications: directed fuzzing and directed unit test synthesis
  - Enabler: semantic-based static analysis (abstract interpretation)
- “Test-time scaling” for program analysis
  - Complementarity of static and dynamic (search-based) approaches

<The Blind Man and the Lame Man>
Korean folktale