optimuzz

PLDI Seoul 2025

Optimization-Directed Compiler Fuzzing for Continuous Translation Validation

Jaeseong Kwon, Bongjun Jang, Juneyoung Lee, Kihong Heo

aws

Programming Systems Laboratory

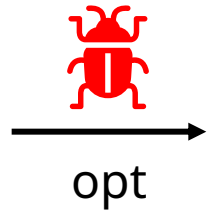
KAIST Korea Advanced Institute of Science and Technology

Correctness of Compiler Optimization

- A wrong compiler optimization can silently change the meaning of your code → Miscompilation
- It is crucial to guarantee the correctness of optimization

```
int src(int a, int b) {
    return (a / b) * b;
}
```

src(4, 3) = 3



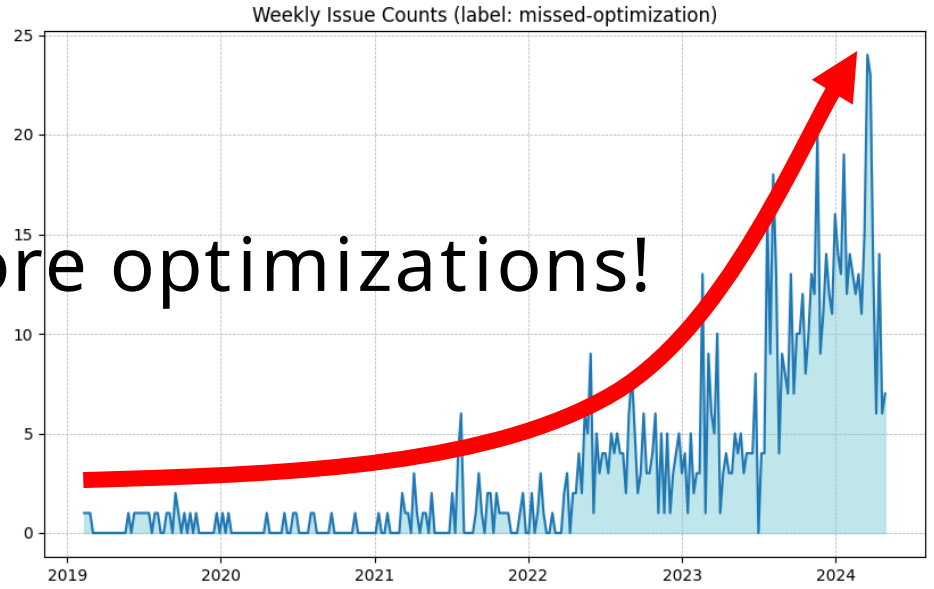

```
int tgt(int a, int b) {
    return a;
}
```

tgt(4, 3) = 4

Challenge: Complexity

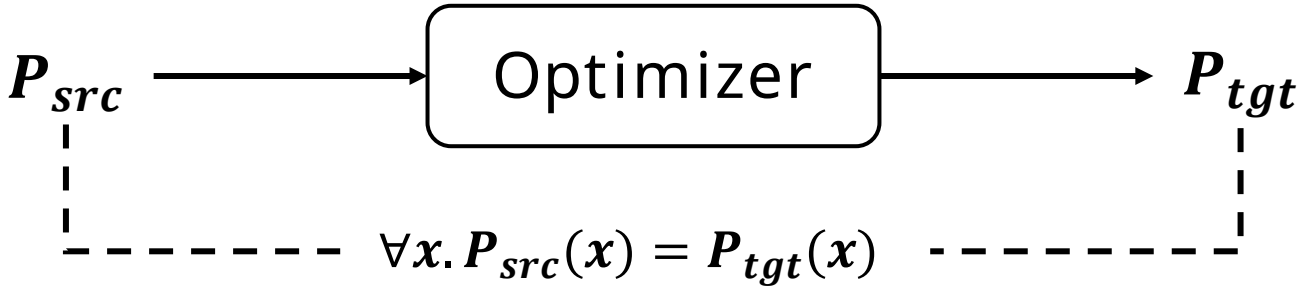
- Real-world Compiler (LLVM)
  - Massive Codebase: > 1.5M LoC
  - Constantly Evolving: > 37,000 commits in 2024

More and more optimizations!



Status Quo: Translation Validation (TV)

- TV checks if the source and target are equivalent (or refinement)
- Common Usage: LLVM optimization updates needs TV checks




```
int src(int a, int b) {
    return (a / b) * b;
}
```

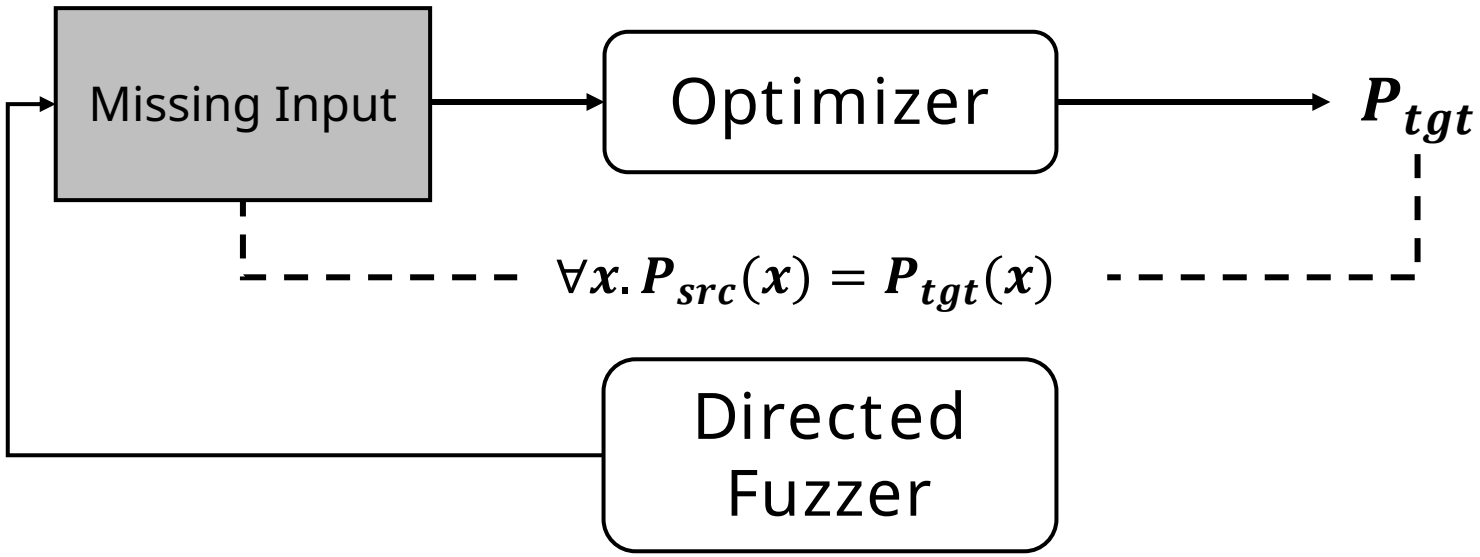
opt

```
int tgt(int a, int b) {
    return a;
}
```

TV: src(4,3) ≠ tgt(4,3)

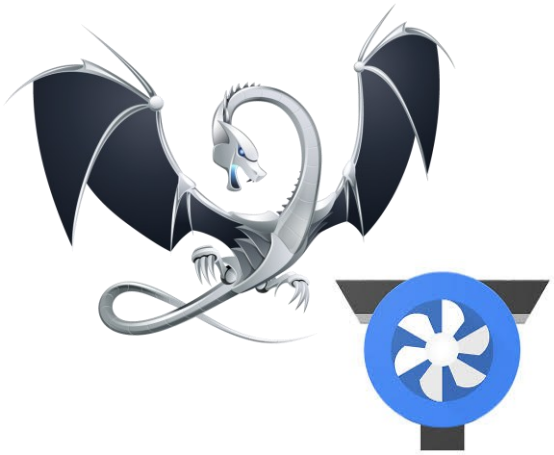
Limitation of TV & Our Solution

- TV needs input programs to detect miscompilation bugs.



Let’s automatically generate input programs!

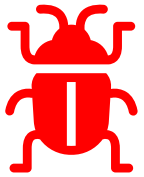
Contributions



First Directed
Compiler Fuzzers
for LLVM and TurboFan

27 vs. 0

27 miscompilation bugs
SOTA compiler fuzzers reproduced 0

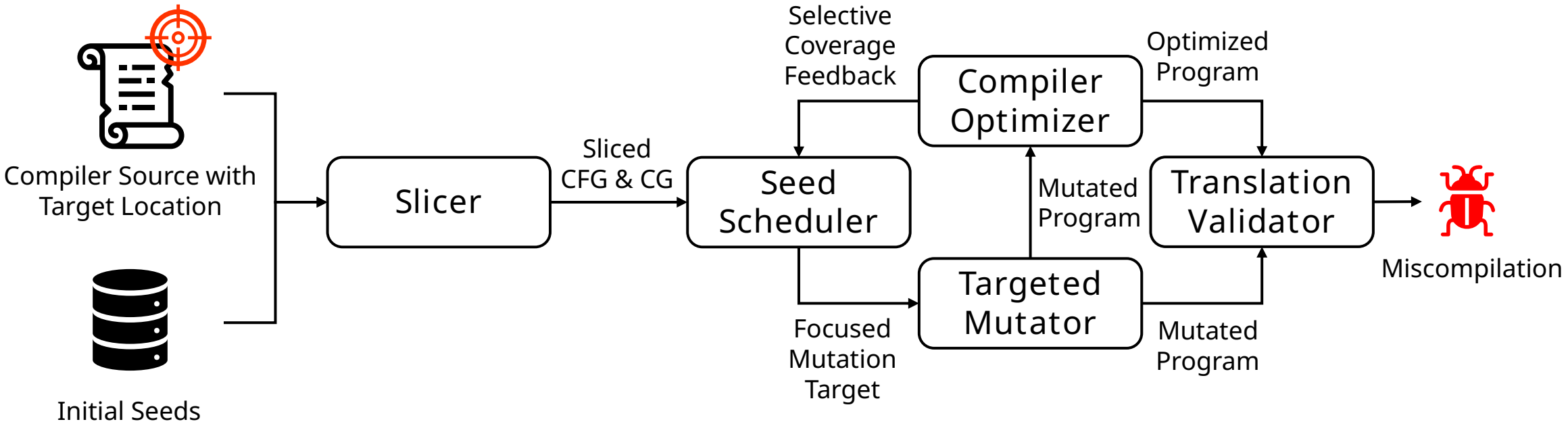


56

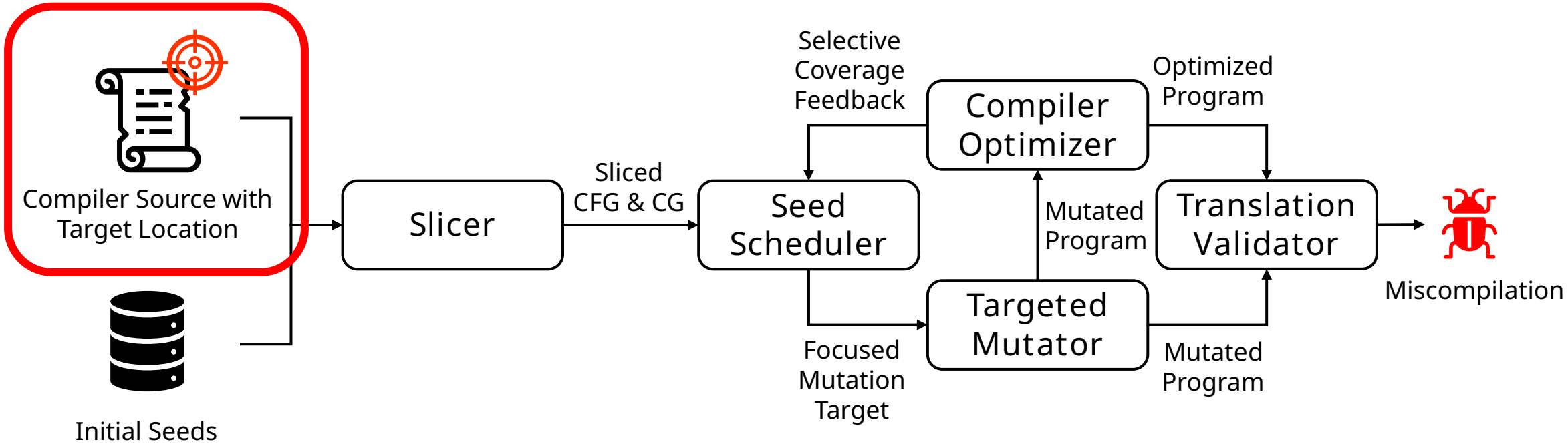
22

56 LLVM bugs reported
22 patched

System Overview



Step 1: Targeting an Optimization



Target Identification

- Optimization: Checks condition + Returns optimized code
- Target Line: Where optimization is triggered and returned

2116921 [InstCombine] Fold select of srem and conditional add

```
+ 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

Automatic Target Line Collection

- Optimization has a simple pattern
  - Simple String Search: lines of "`return NewInstruction()`"
  - LLM: teach the pattern and analyze the source code

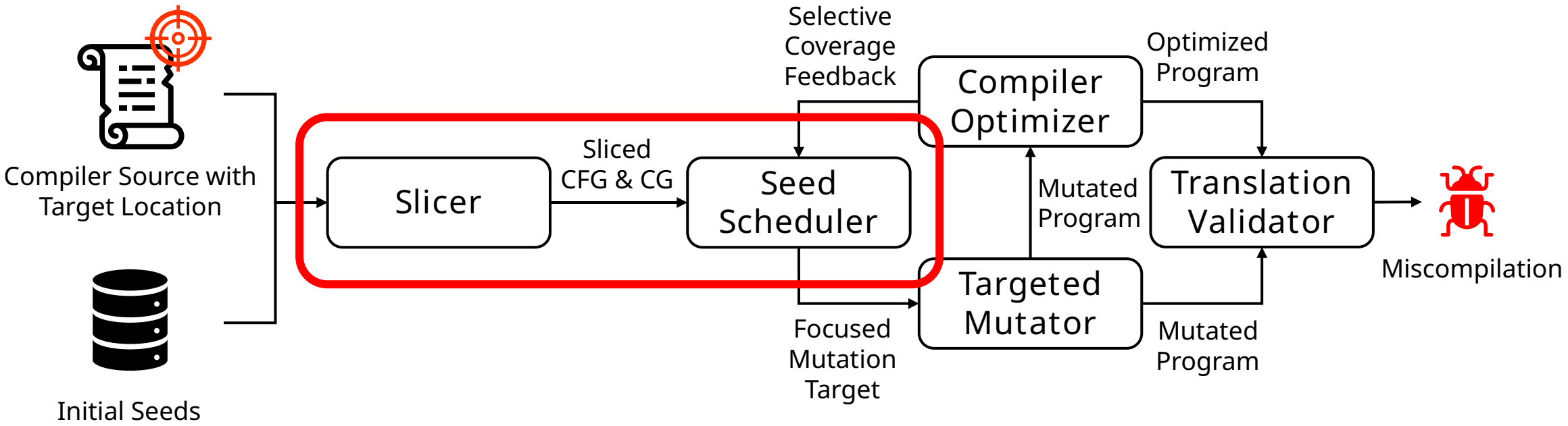
2116921 [InstCombine] Fold select of srem and conditional add

```
+ 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

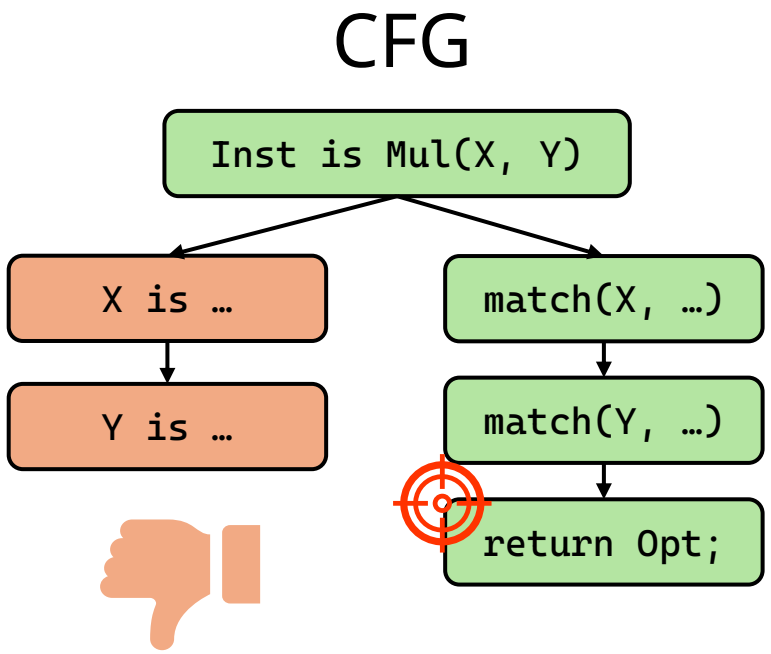
Target: Returns Optimized Code

Step 2: Guided Search Strategy



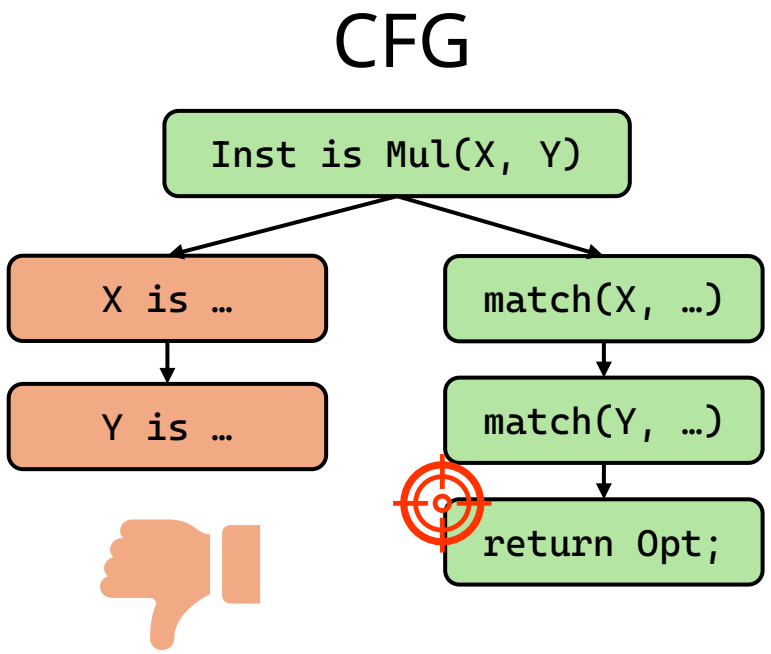
Slicer: Selective Coverage Feedback

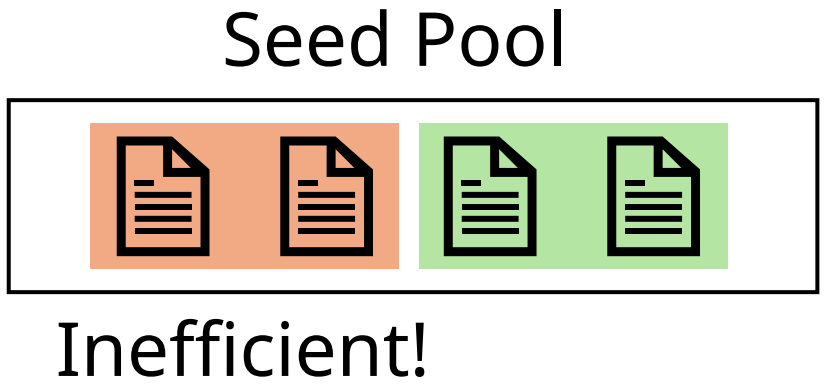
- Coverage feedback guides the fuzzer
- Only relevant program locations → Efficiency



Slicer: Selective Coverage Feedback

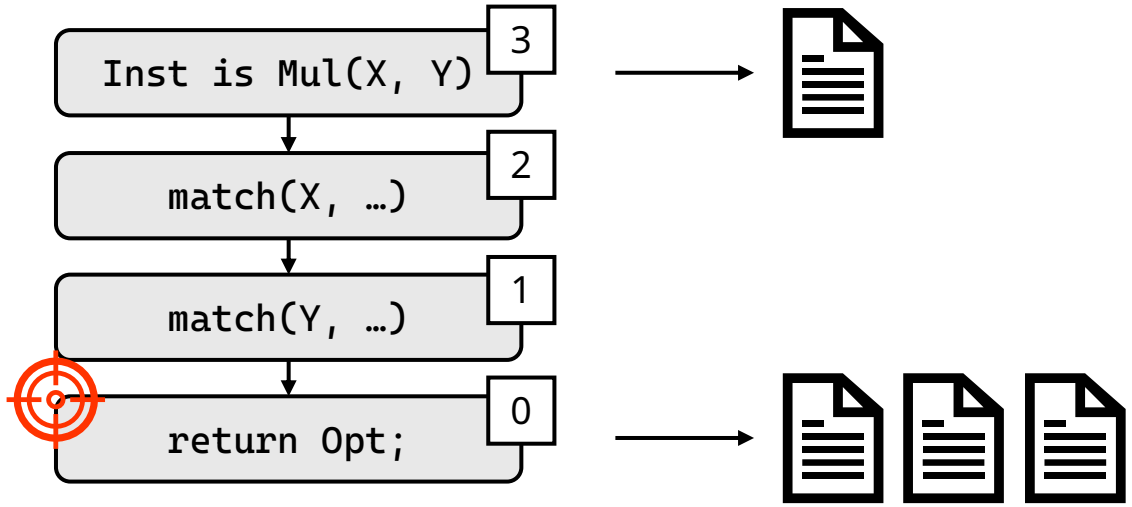
- Coverage feedback guides the fuzzer
- Only relevant program locations → Efficiency



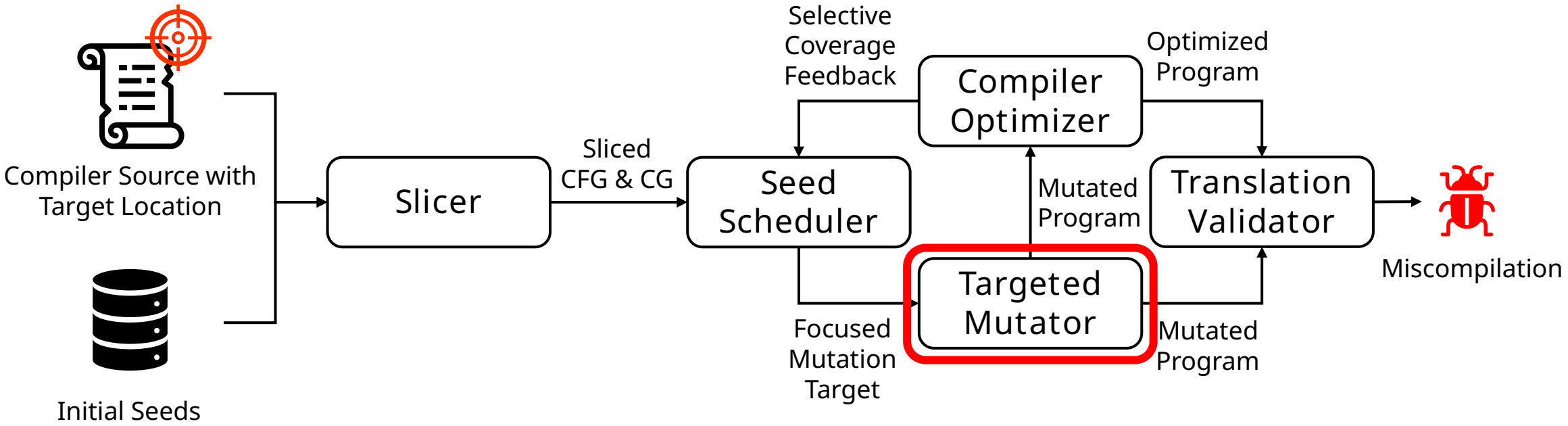


Seed Scheduler: Distance Metric

- Assign “distance-to-target” values to nodes in the sliced CFG
- Closer seed → Generate more mutated programs!

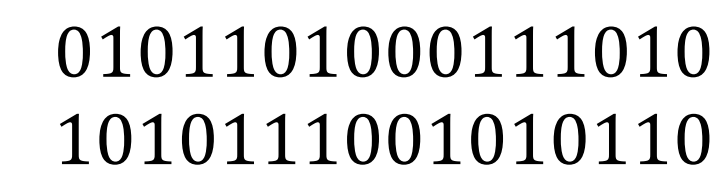


Step 3: Targeted Mutation Strategy



Program as an Input

- Conventional Fuzzing: Bitstream Input
  - Hard to analyze the structure of an input
- Compiler Fuzzing: Program Input
  - Input has useful structures such as CFG, DUG, …



Conventional Fuzzing

```
define i8 @f(i8 %x, i8 %c) {
  %2 = shl 1, %c
  %3 = mul %x, %2
  ret i8 %3
}
```

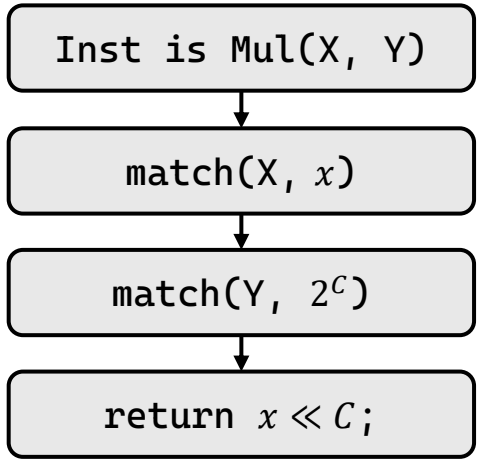
Compiler Fuzzing

Def-Use in Optimization Rules

- Optimization typically checks def-use relation of a program.

$$x \times 2^C \Rightarrow x \ll C$$

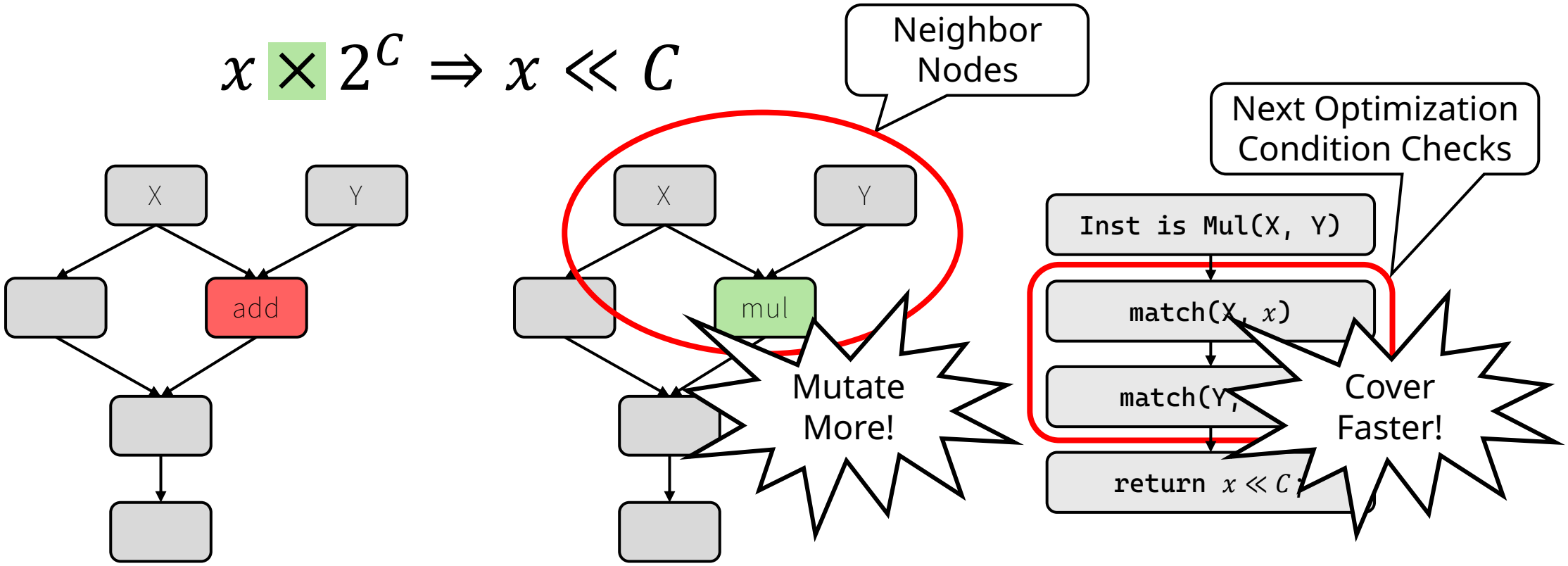
Optimization Rule



CFG

Finding Next Mutation Targets

$$x \times 2^C \Rightarrow x \ll C$$



Input Program

Mutated Program

CFG

Effectiveness Evaluation

- Optimuzz Framework: Fuzzer + Translation Validator (TV)

Compiler	LLVM
Fuzzer	Our Own
TV	Alive2 (PLDI ’17)

Effectiveness Evaluation

- Optimuzz Framework: Fuzzer + Translation Validator (TV)

Compiler	LLVM
Fuzzer	Our Own
TV	Alive2 (PLDI ’17)
Benchmark	24 Known Bugs
Baselines	FLUX (ASE’ 23) Alive-Mutate (CGO, 24)

Effectiveness Evaluation

- Optimuzz Framework: Fuzzer + Translation Validator (TV)

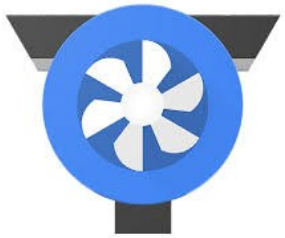
Compiler	LLVM	TurboFan
Fuzzer	Our Own	Fuzzilli + Our Strategies
TV	Alive2 (PLDI ’17)	TurboTV (ICSE ’24)
Benchmark	24 Known Bugs	6 Known Bugs
Baselines	FLUX (ASE’ 23) Alive-Mutate (CGO, 24)	Fuzzilli (As Is)

Effectivess of Optimuzz

- Optimuzz performs significantly better than SOTA fuzzers.

23 vs. 0

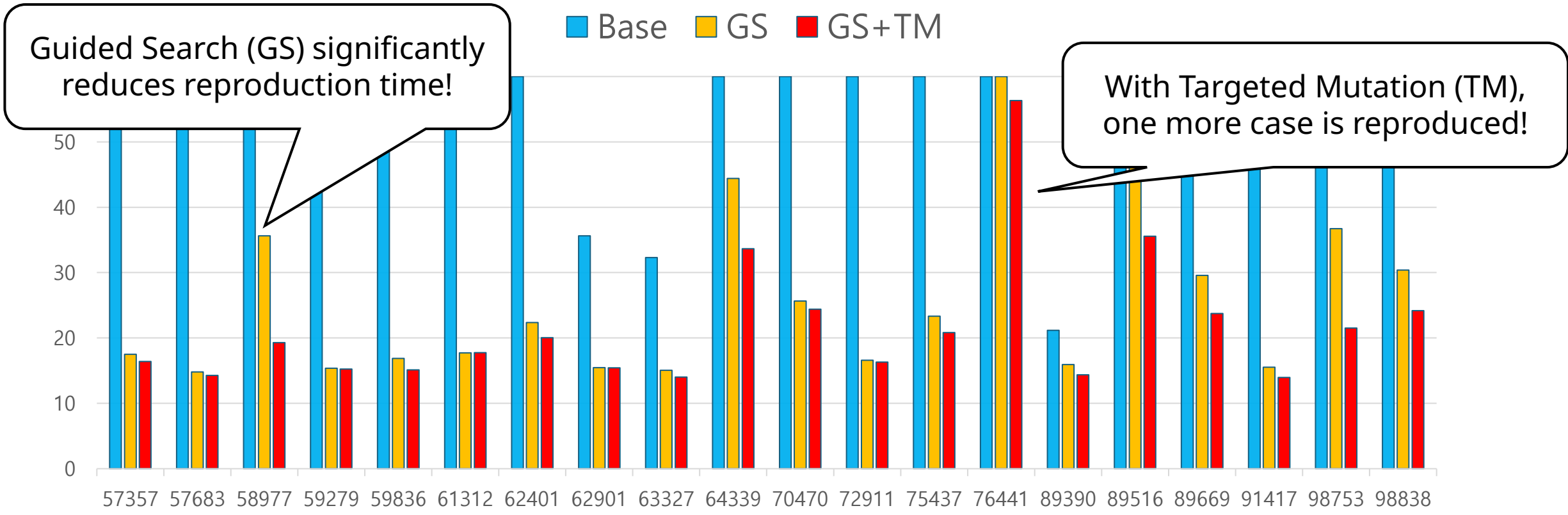
⌛ ~ 1 Hour



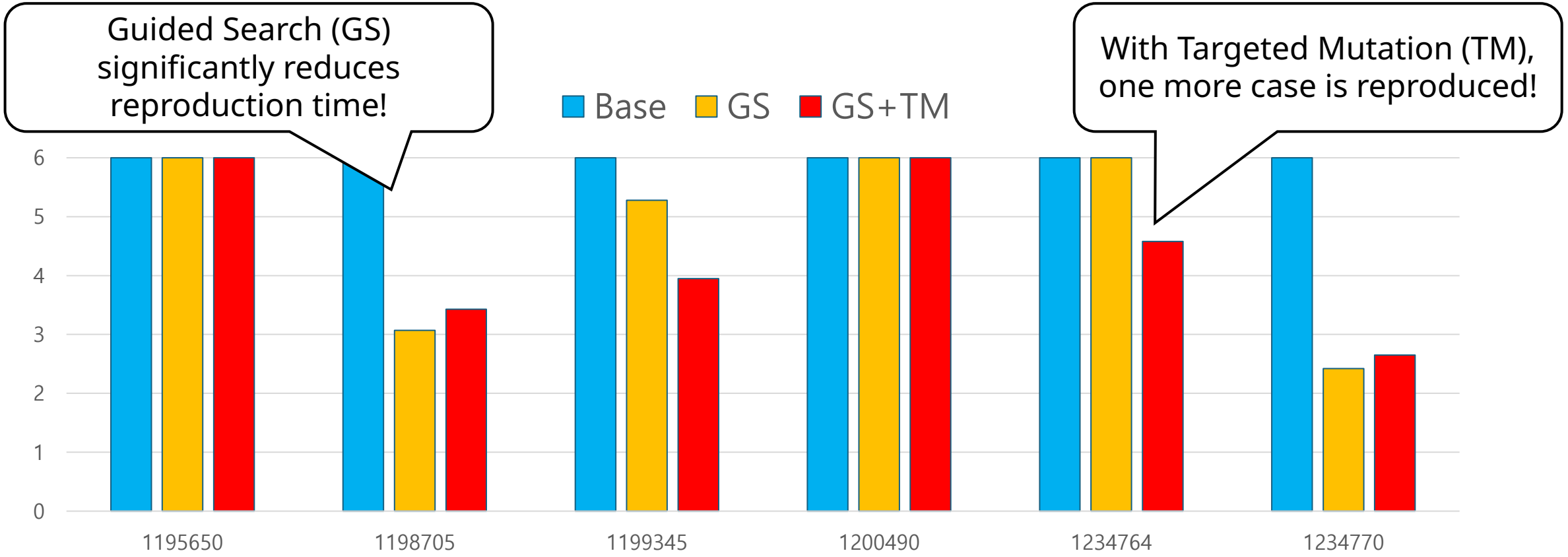
4 vs. 0

⌛ ~ 6 Hour

Each Strategy's Impact on LLVM



Each Strategy's Impact on TurboFan



LLVM Bug Finding

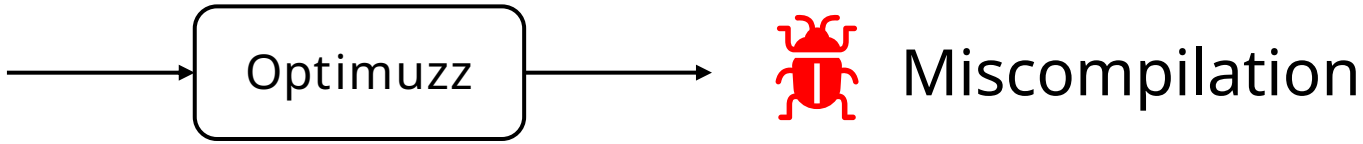
- Optimuzz is effective to find unknown miscompilations
- Continuous Mode: for targets in each LLVM update
- Batch Mode: for all targets in the latest version of LLVM

LLVM Bug Finding: Continuous Mode

- 8 bugs from LLVM GitHub commits

2116921 [InstCombine] Fold select of srem and conditional add

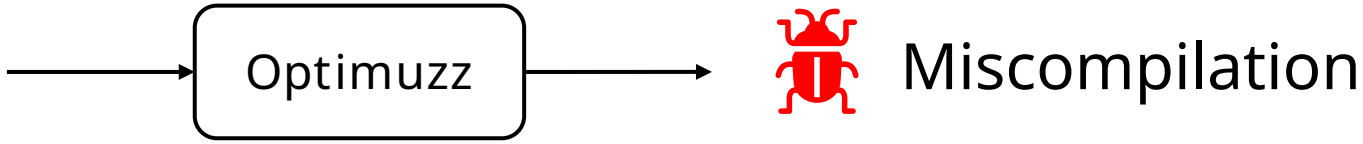
```
+ if ( ... )
+   return Opt;
```



Miscompilation

e710a5 [InstCombine] Fold fneg/fabs patterns

```
+ if ( ... )
+   return Opt;
```

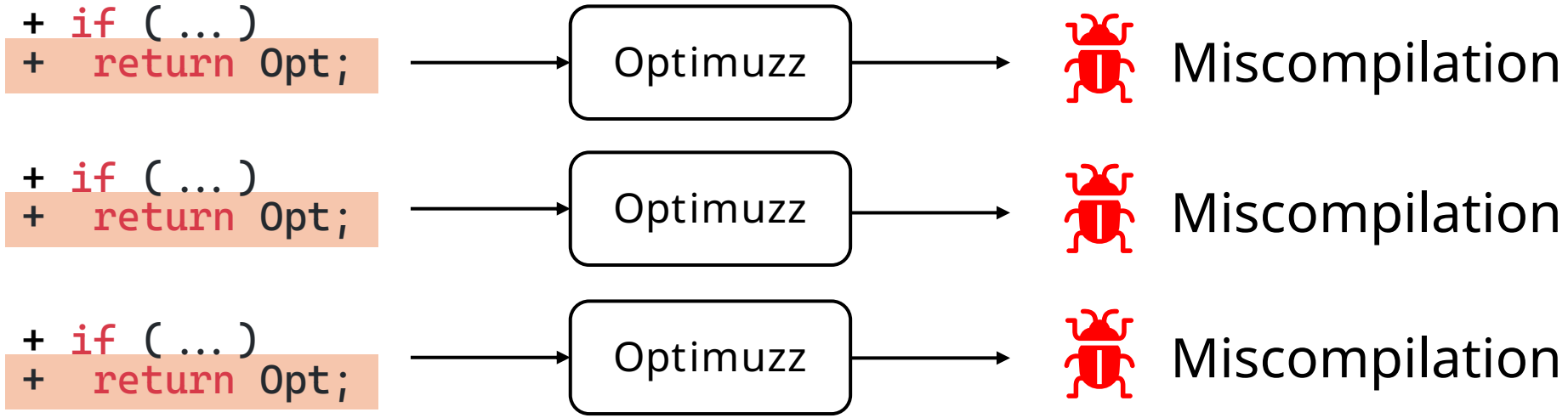


Miscompilation

LLVM Bug Finding: Batch Mode

- 48 bugs from the latest version of LLVM codebase

llvm/lib/Transforms/InstCombine/InstCombineSelect.cpp



Summary

- First Directed Compiler Fuzzing for Continuous TV
- The Guided Search Strategy
  - Selective Coverage Feedback
  - Distance Metric
- The Targeted Mutation Strategy
  - Effectively Finding Next Mutation Targets
- Real World Impacts
  - 56 bugs reported to LLVM, 22 patched

Visit Our Website!

Appendix

Other than InstCombine

- Optimuzz works for optimizations other than InstCombine
- It leverages the condition check and return structure. The structure is observed generally in compiler source codes.

Mode	Optimization Pass					
	InstCombine	InstSimplify	Correlated Propagation	SLP Vectorization	GVN	Vector Combine
Cont	4	0	3	0	0	1
Batch	39	2	0	4	2	1
Total	56					

Target-Hit Ratio (LLVM, %)