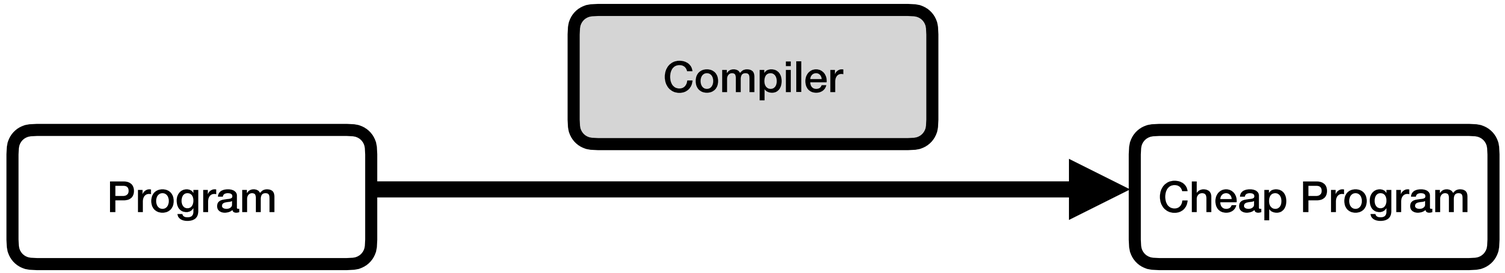
Visualize Optimization-directed Fuzzing for Effective Optimization Testing

Jaeseong Kwon

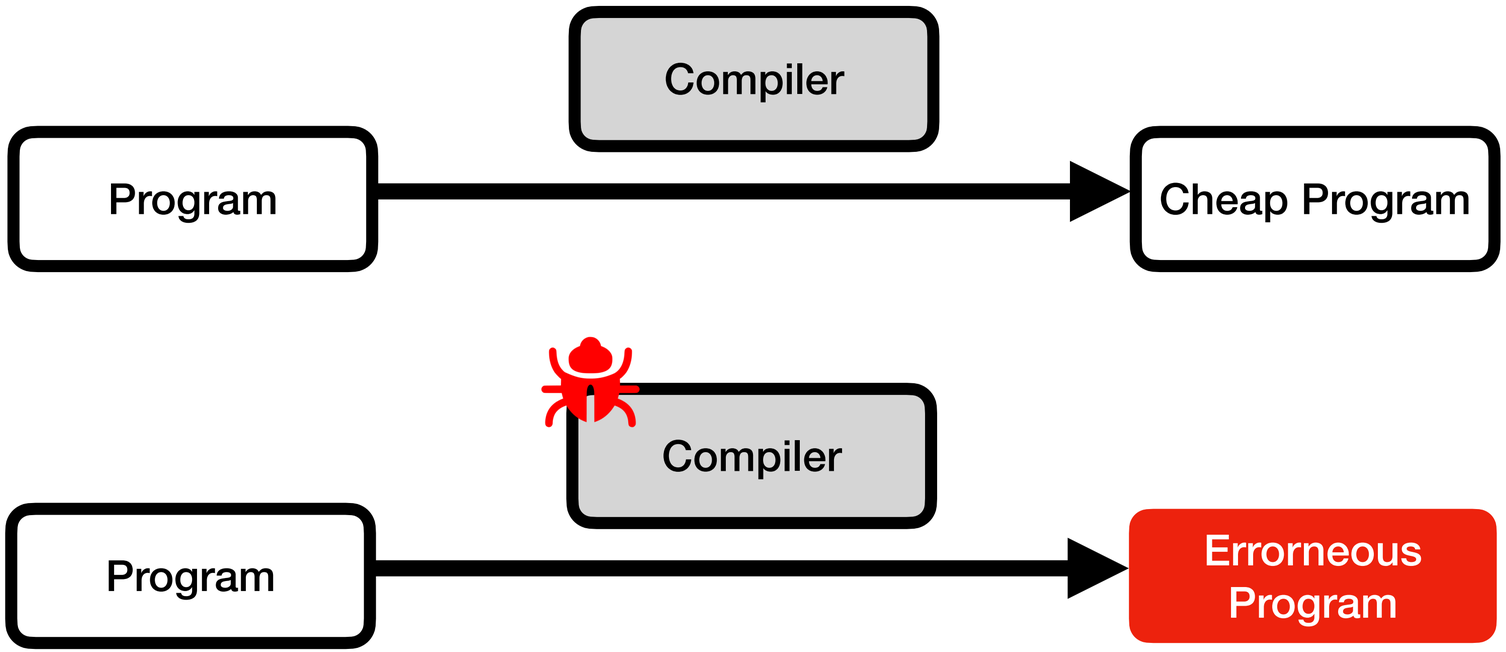
IS661

Compiler’s Optimization Bugs



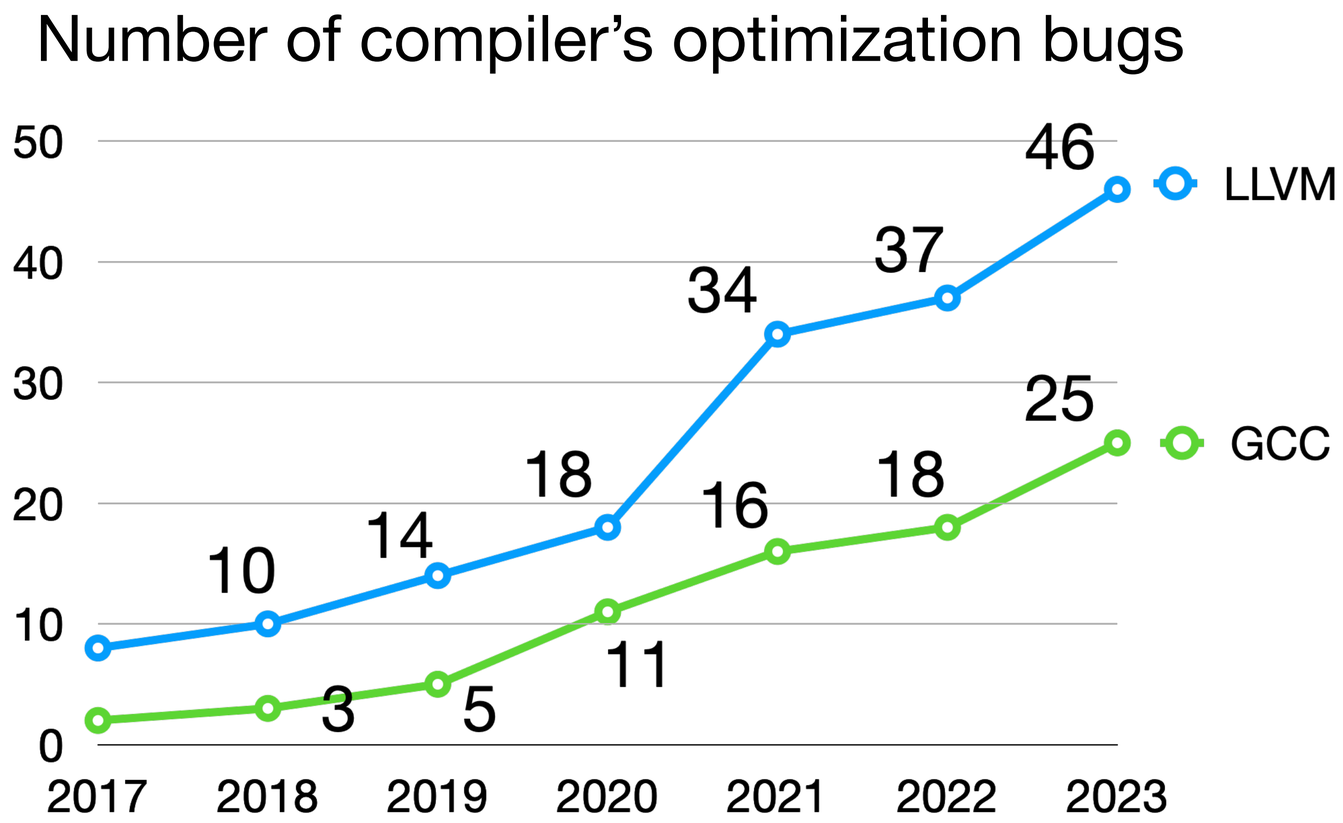
Compiler’s Optimization Bugs

- Incorrect optimization can produce erroneous programs

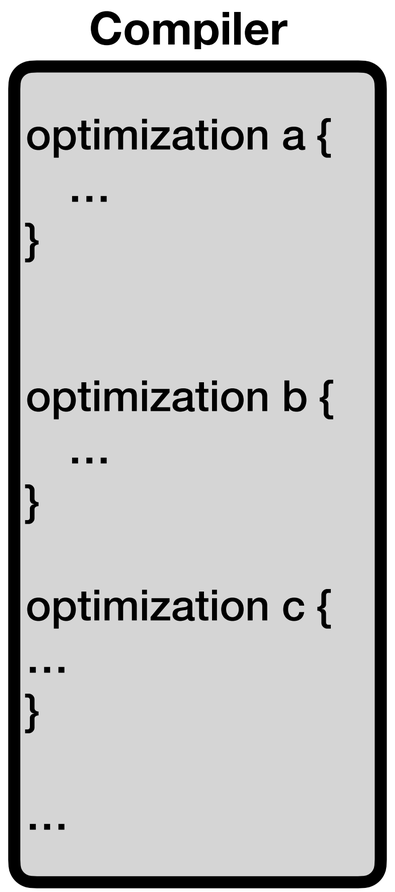


Compiler’s Optimization Bugs

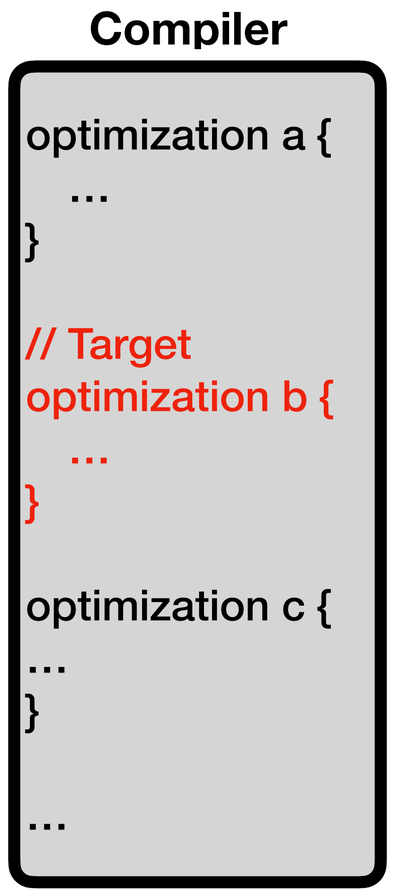
- Compiler optimization bugs are increasing



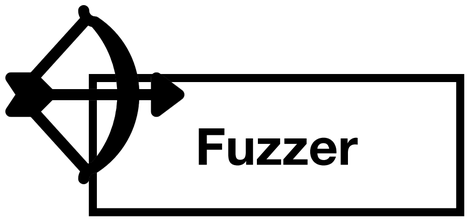
Our Tool: Optimization-directed Fuzzer



Our Tool: Optimization-directed Fuzzer



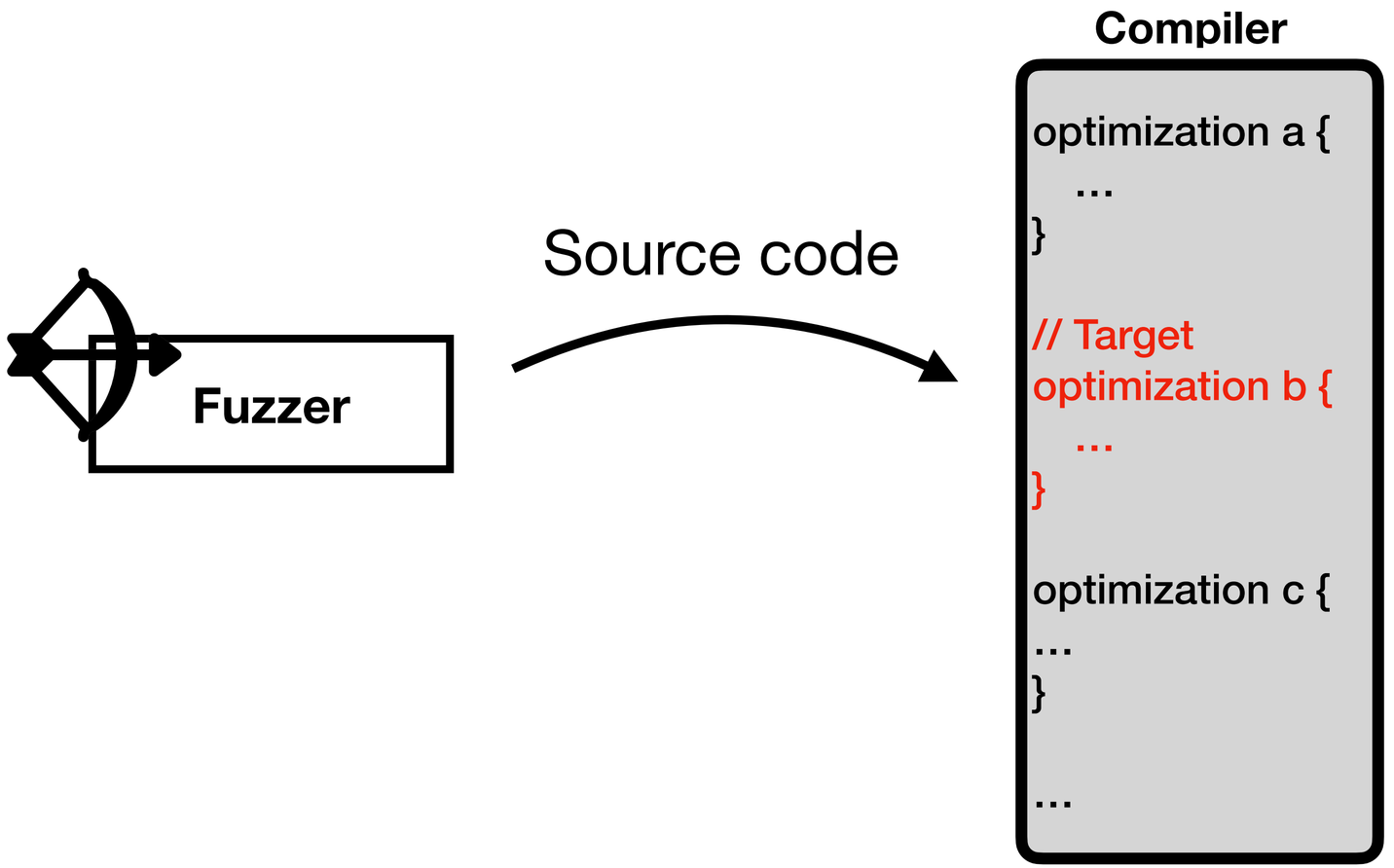
Our Tool: Optimization-directed Fuzzer



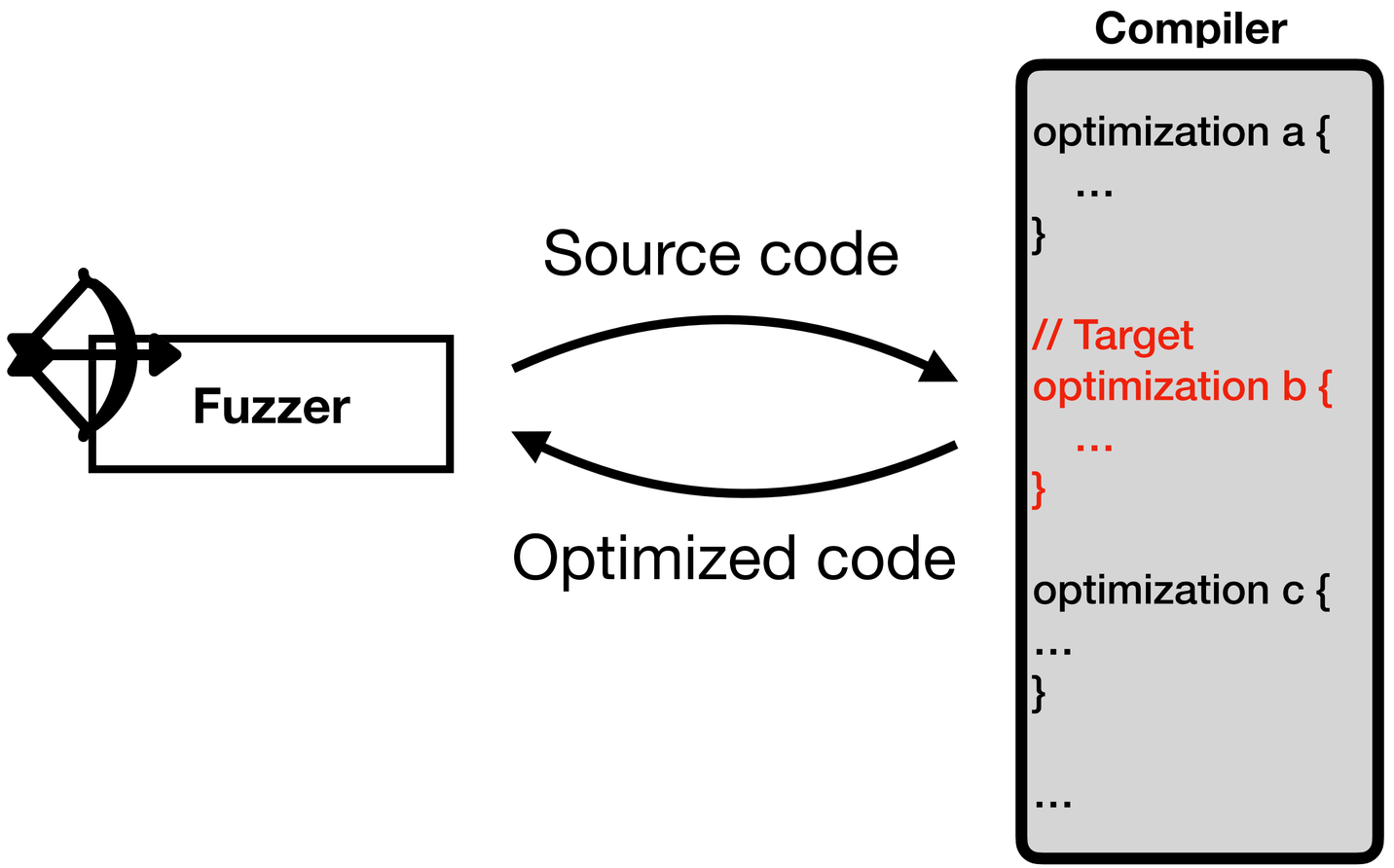
Compiler

```
optimization a {
   …
}

// Target
optimization b {
   …
}

optimization c {
…
}


…
```

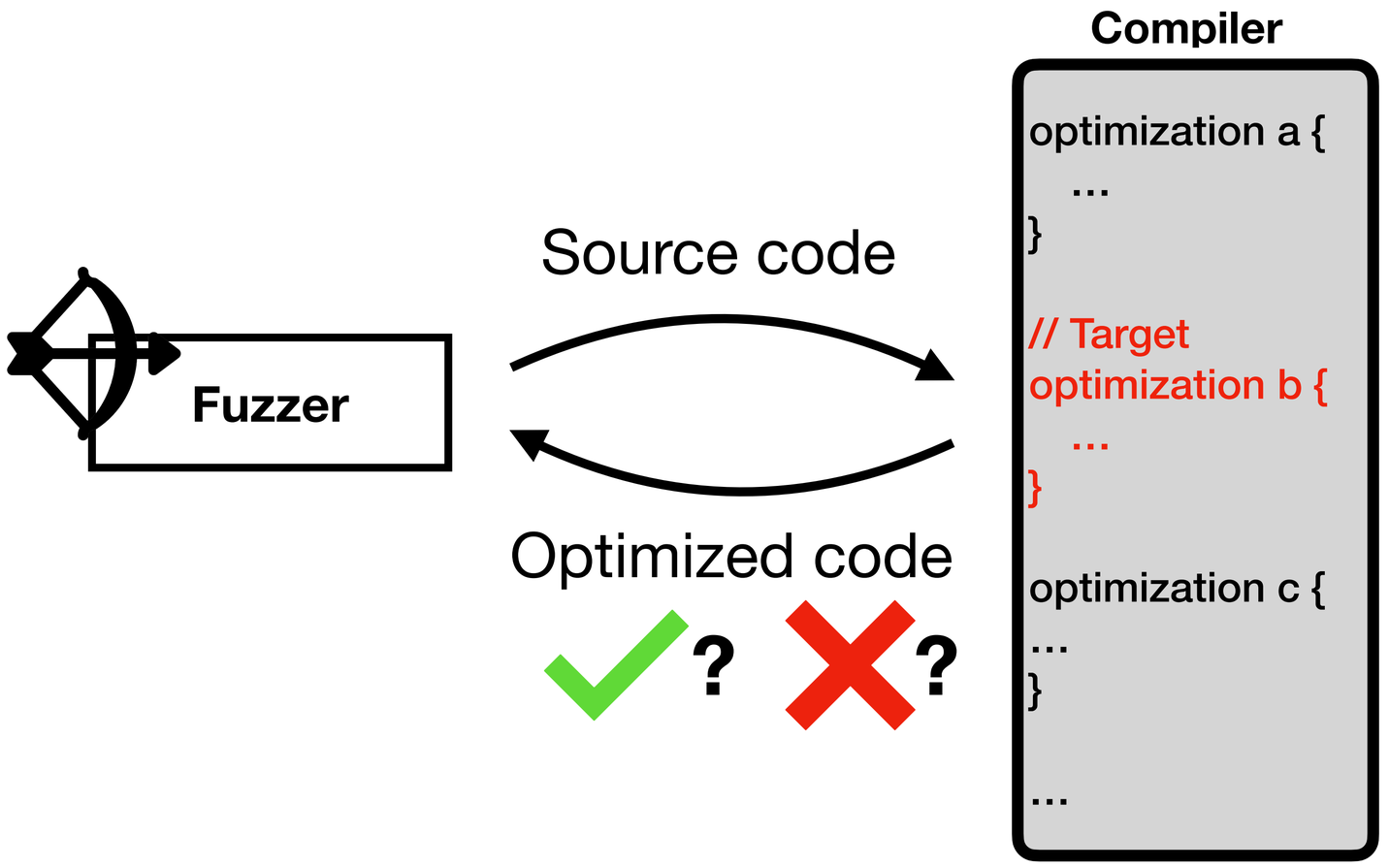
Our Tool: Optimization-directed Fuzzer



Our Tool: Optimization-directed Fuzzer

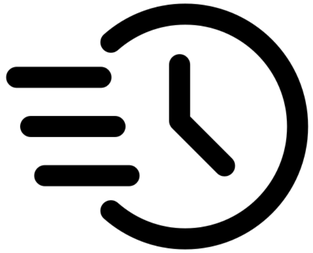


Our Tool: Optimization-directed Fuzzer



Our Tool: Optimization-directed Fuzzer

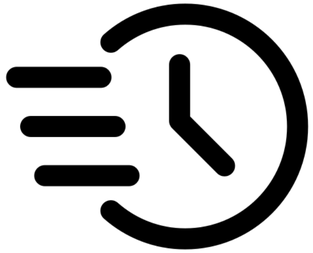
IR Syntax Based Mutation



Create only **valid** compiler’s input

Our Tool: Optimization-directed Fuzzer

IR Syntax Based Mutation



Create only **valid** compiler's input

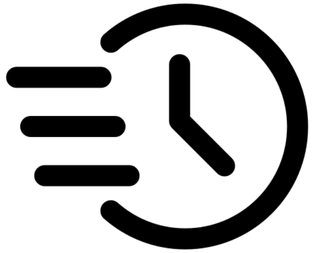
+

Directed Fuzzing

Test the **target** optimization effectively

Our Tool: Optimization-directed Fuzzer

IR Syntax Based Mutation



Create only **valid** compiler's input

+

Directed Fuzzing

Test the **target** optimization effectively

=

Effective Optimization Testing

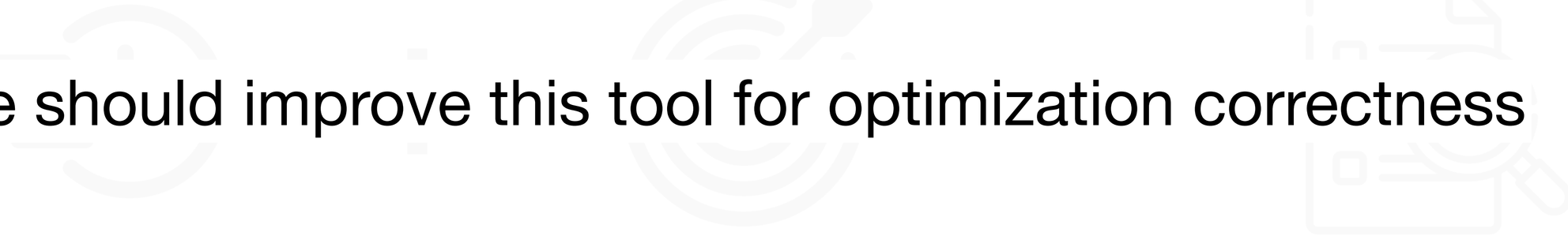
Reproduce 4 bugs within **5 min**

Our Tool: Optimization-directed Fuzzer

IR Syntax Based Mutation

Directed Fuzzing

Effective Optimization Testing



We should improve this tool for optimization correctness

Create only **valid** compiler's input

Test the **target** optimization effectively

Reproduce 4 bugs within **5 min**

How to Improve Our Fuzzing Process?

- **Better Mutation**
  - All mutation should create a valid compiler IR
  - Each mutation should be attempted at the intended rate

How to Improve Our Fuzzing Process?

- **Better Mutation**
  - All mutation should create a valid compiler IR
  - Each mutation should be attempted at the intended rate

- **Effective Guide Performance**
  - Fuzzer should efficiently make IR that raise target optimization
  - Fuzzer should guide mutations towards target optimization well

Research Goal: Visualize Our Fuzzer

- **Visualize**
  - "A computer should make both calculations and **graphs**"*

*F. J. Anscombe. 1973. Graphs in Statistical Analysis

Research Goal: Visualize Our Fuzzer

- **Visualize**
  - "A computer should make both calculations and **graphs**"*
- **Mutation Statistics**
  - Graphically represent the attempt rates and success rates of each mutation

*F. J. Anscombe. 1973. Graphs in Statistical Analysis

Research Goal: Visualize Our Fuzzer

- **Visualize**
  - "A computer should make both calculations and **graphs**"*
- **Mutation Statistics**
  - Graphically represent the attempt rates and success rates of each mutation
- **Conditional Statement Tree Heat Map**
  - Measuring guide performance by displaying the coverage in a heatmap

*F. J. Anscombe. 1973. Graphs in Statistical Analysis

Research Goal: Visualize Our Fuzzer

Easy Understanding

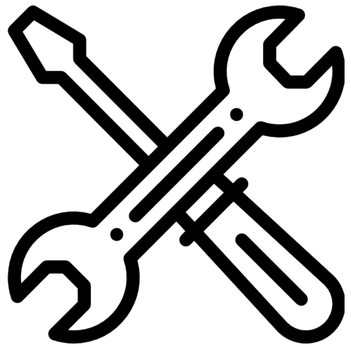
Reduce **30%** time for new developer

Research Goal: Visualize Our Fuzzer

Easy Understanding

Reduce **30%** time for new developer

Maintenance



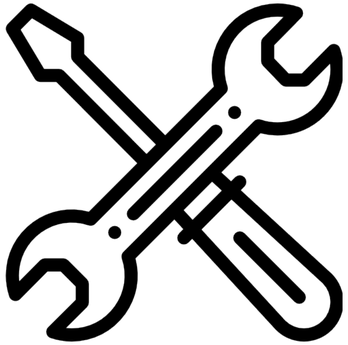
50% faster debugging speed

Research Goal: Visualize Our Fuzzer

Easy Understanding

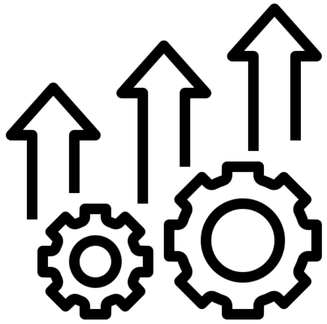
Reduce **30%** time for new developer

Maintenance



50% faster debugging speed

Improvement



Increase the program improvement speed by **20%**

Our Mutation Strategy

- Mutations that change **IR** programs to other **IR** programs
  - **4** mutations: opcode, operand, flag, type

Our Mutation Strategy

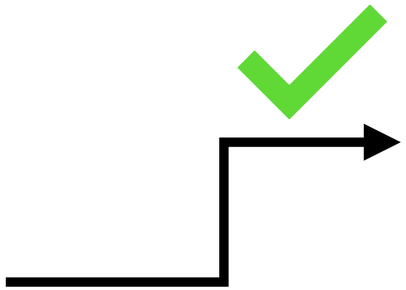
- Mutations that change **IR** programs to other **IR** programs
  - **4** mutations: opcode, operand, flag, type

- All mutations should create valid IR, but it can be failed
  - The failure cases should be revised to improve the mutation efficiency

Our Mutation Strategy

- Opcode Mutation

```
define i32 @src (i32 %x, i32 %y){
entry:
    %add = add i32 %x, %y
    ret i32 %add
}
```

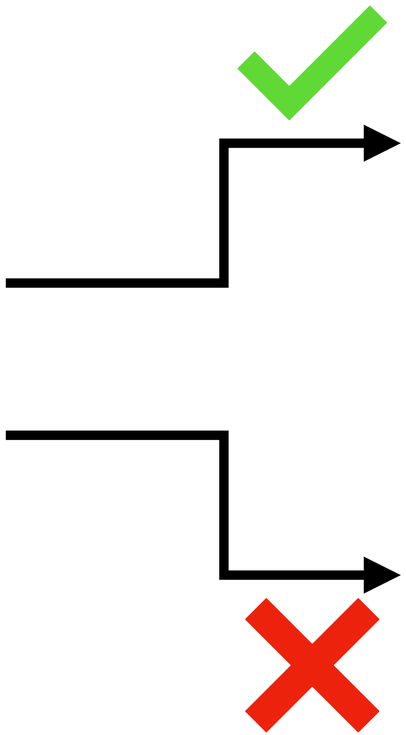


```
define i32 @mutant (i32 %x, i32 %y){
entry:
    %sub = sub i32 %x, %y
    ret i32 %sub
}
```

Our Mutation Strategy

- Opcode Mutation

```
define i32 @src (i32 %x, i32 %y){
entry:
    %add = add i32 %x, %y
    ret i32 %add
}
```



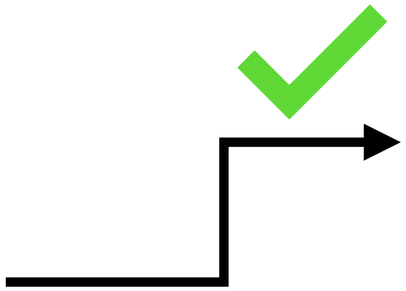
```
define i32 @mutant (i32 %x, i32 %y){
entry:
    %sub = sub i32 %x, %y
    ret i32 %sub
}
```

```
define i32 @invalid(i32 %x, i32 %y){
entry:
    %store = store i32 %x, %y
    ret i32 %store
}
```

Our Mutation Strategy

- Operand Mutation

```
define i32 @src (i32 %x, i32 %y){
entry:
    %add = add i32 %x, %y
    ret i32 %add
}
```

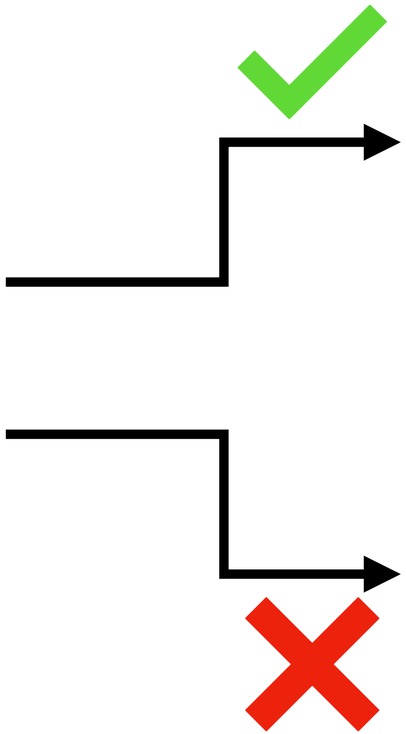


```
define i32 @mutant (i32 %x, i32 %y){
entry:
    %add = add i32 %x, 1
    ret i32 %add
}
```

Our Mutation Strategy

- Operand Mutation

```
define i32 @src (i32 %x, i32 %y){
entry:
    %add = add i32 %x, %y
    ret i32 %add
}
```



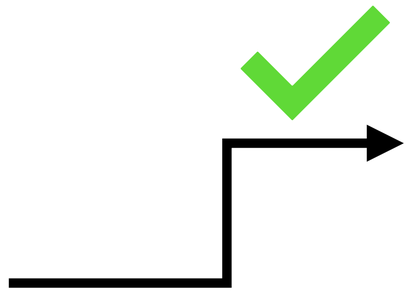
```
define i32 @mutant (i32 %x, i32 %y){
entry:
    %add = add i32 %x, 1
    ret i32 %add
}
```

```
define i32 @invalid(i32 %x,i32 %y,ptr %z){
entry:
    %add = add i32 %x, %z
    ret i32 %add
}
```

Our Mutation Strategy

- Type Mutation

```
define i32 @src (i32 %x, i32 %y){
entry:
    %add = add i32 %x, %y
    ret i32 %add
}
```

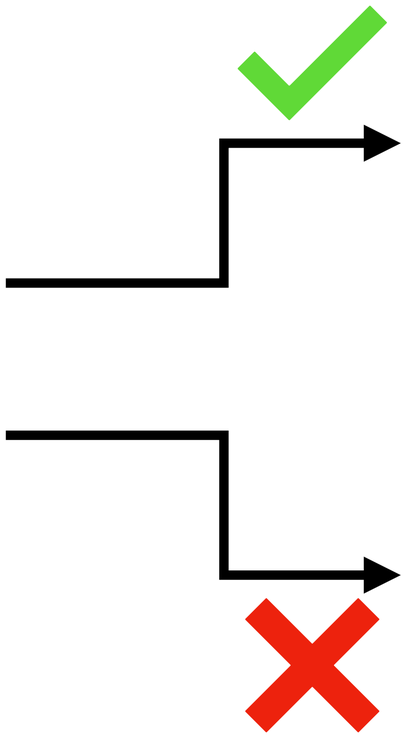


```
define i8 @mutant (i8 %x, i8 %y){
entry:
    %add = add i8 %x, %y
    ret i8 %add
}
```

Our Mutation Strategy

- Type Mutation

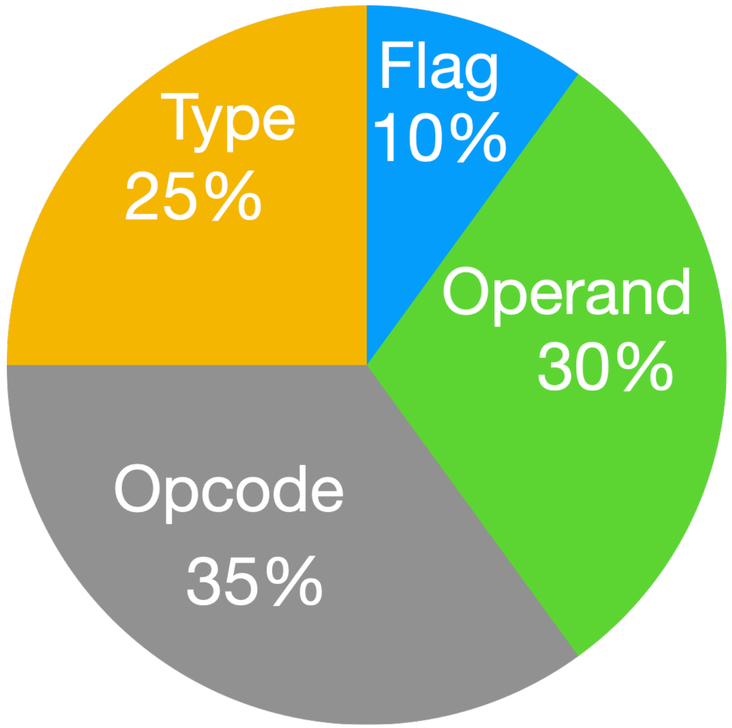
```
define i32 @src (i32 %x, i32 %y){
entry:
    %add = add i32 %x, %y
    ret i32 %add
}
```



```
define i8 @mutant (i8 %x, i8 %y){
entry:
    %add = add i8 %x, %y
    ret i8 %add
}
```

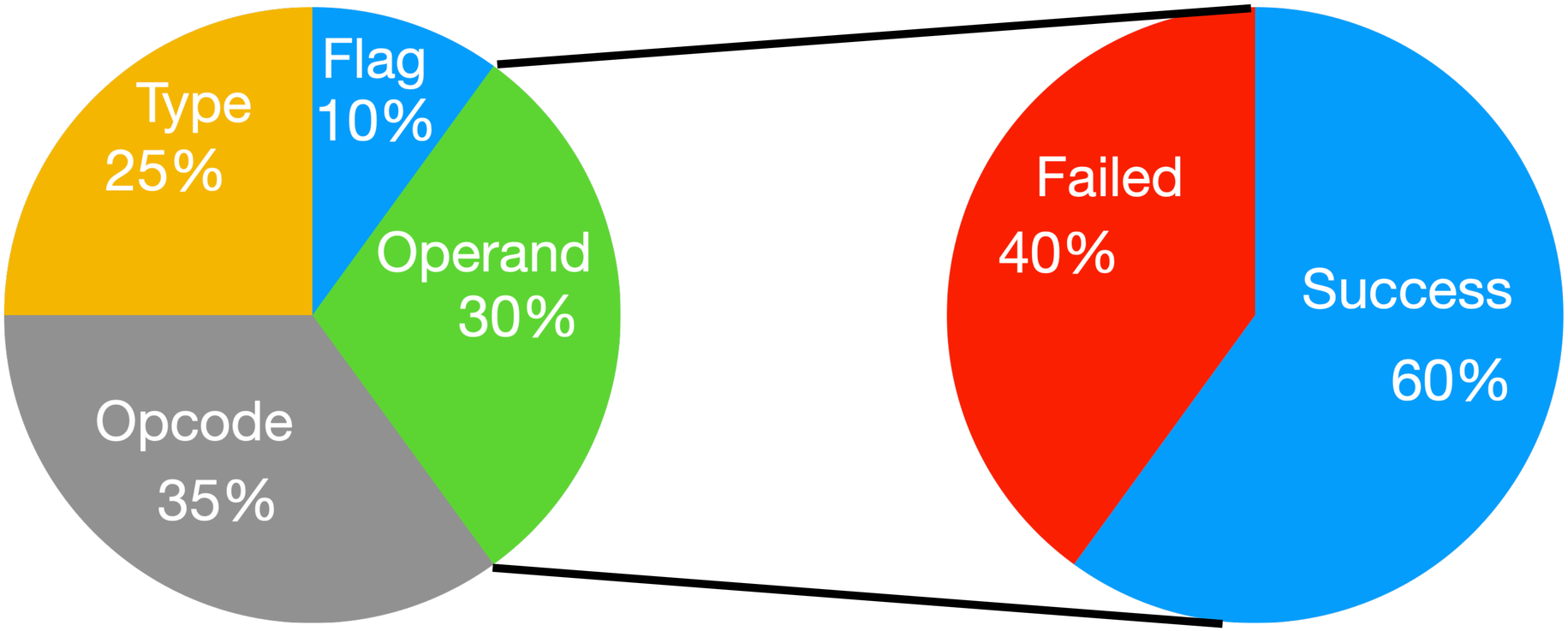
```
define ptr @mutant (ptr %x, ptr %y){
entry:
    %add = add ptr %x, %y
    ret ptr %add
}
```

Visualize Mutation Statistic



Percentage of Mutations attempted

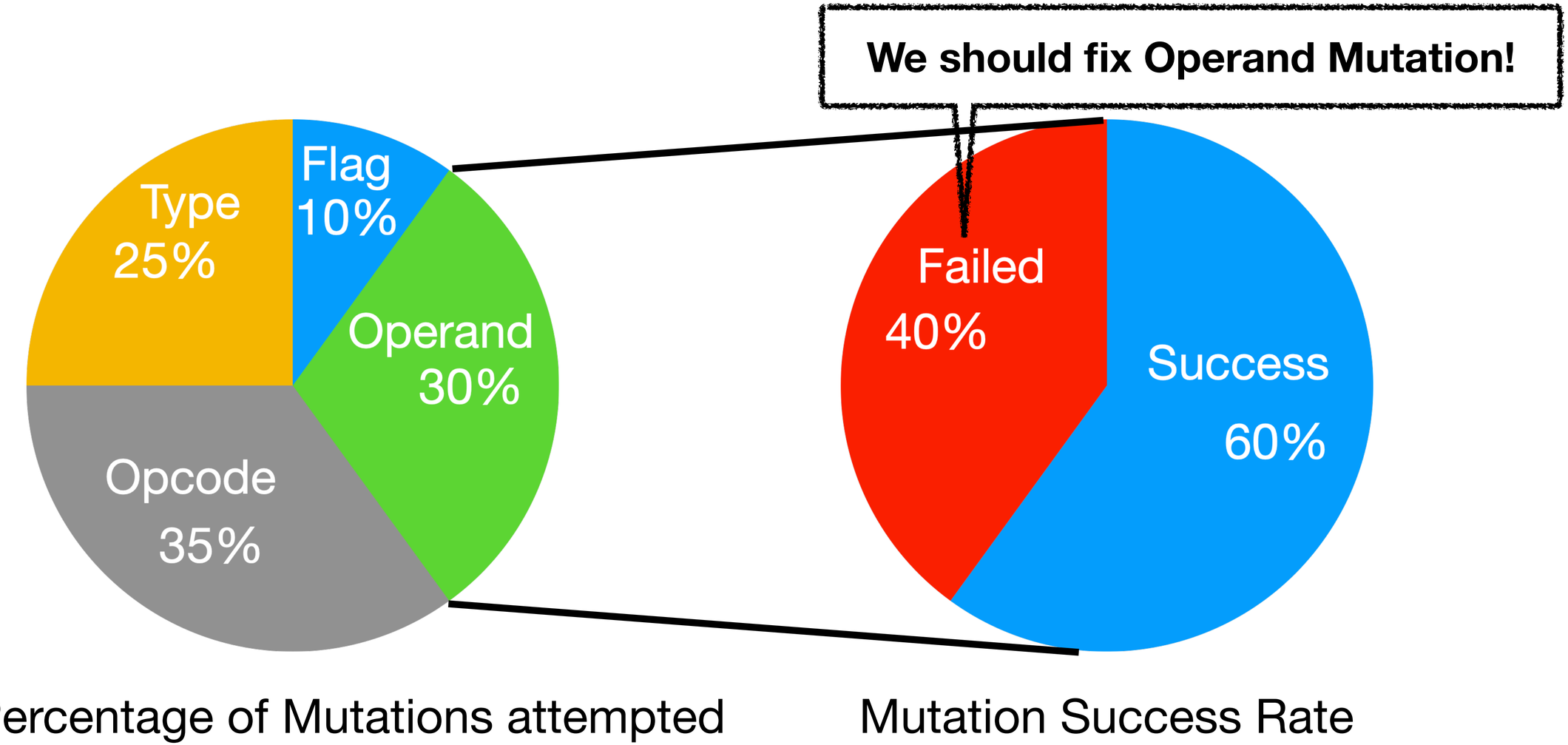
Visualize Mutation Statistic



Percentage of Mutations attempted

Mutation Success Rate

Visualize Mutation Statistic

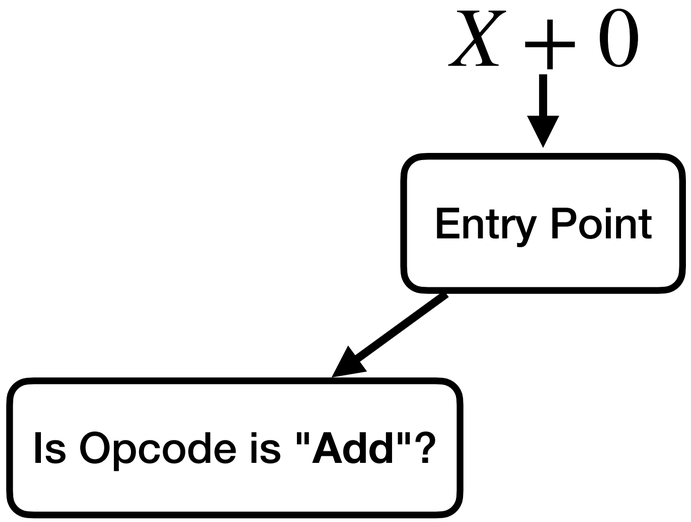


Percentage of Mutations attempted

Mutation Success Rate

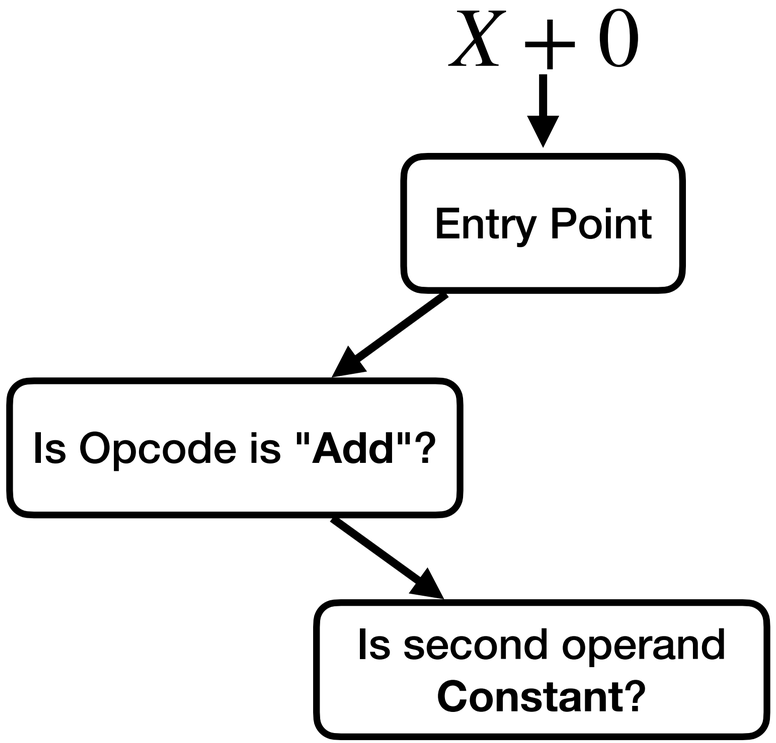
Our Guide Strategy

Target Optimization: $X + 0 \rightarrow X$



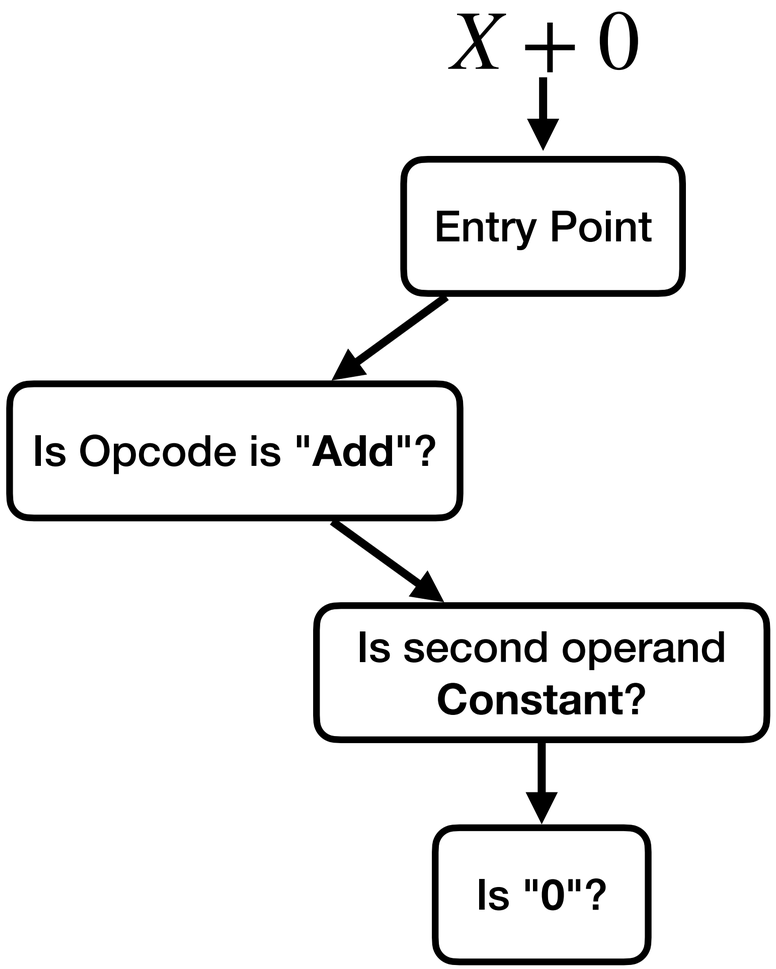
Our Guide Strategy

Target Optimization: $X + 0 \rightarrow X$



Our Guide Strategy

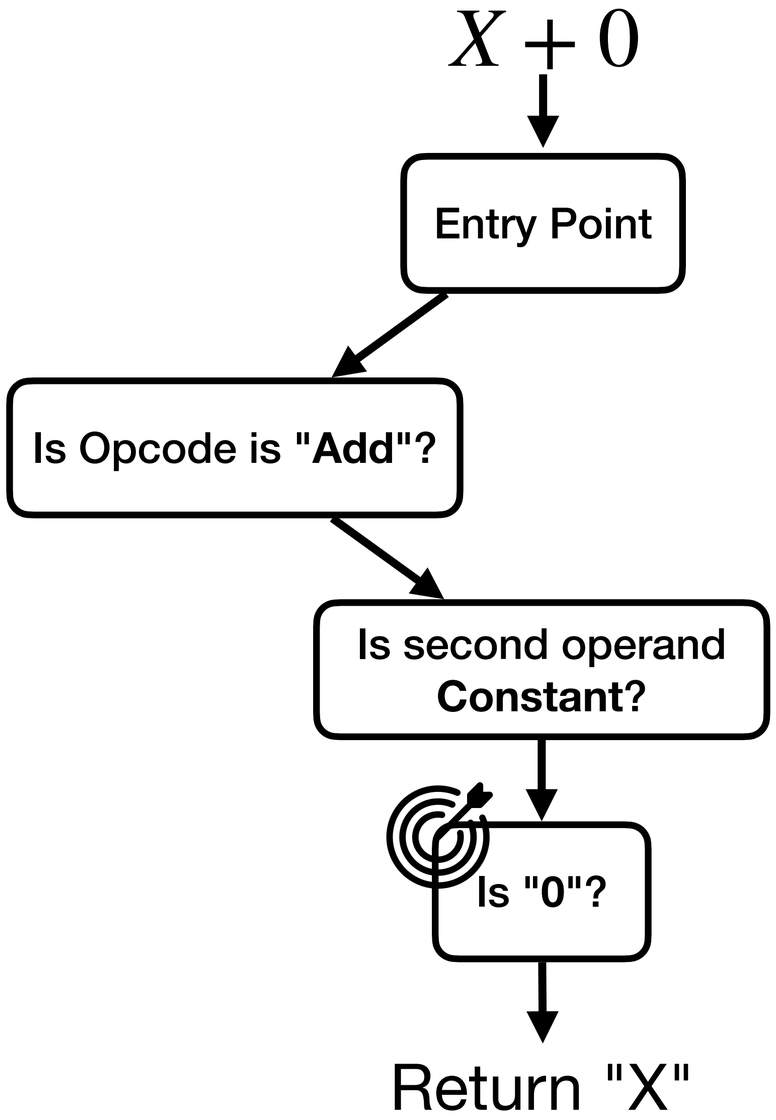
Target Optimization: $X + 0 \rightarrow X$



...

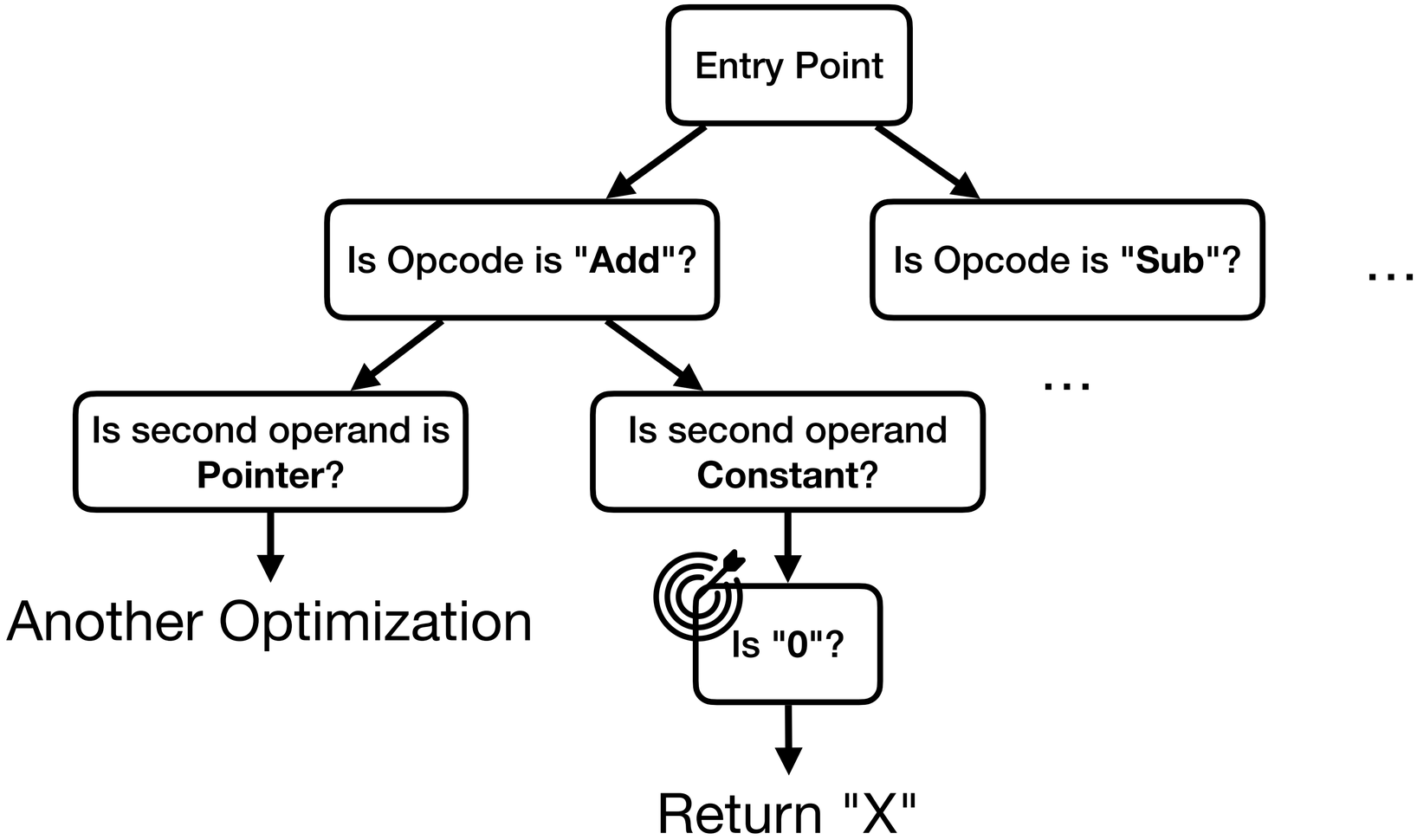
Our Guide Strategy

Target Optimization: $X + 0 \rightarrow X$



Our Guide Strategy

Target Optimization: $X + 0 \rightarrow X$

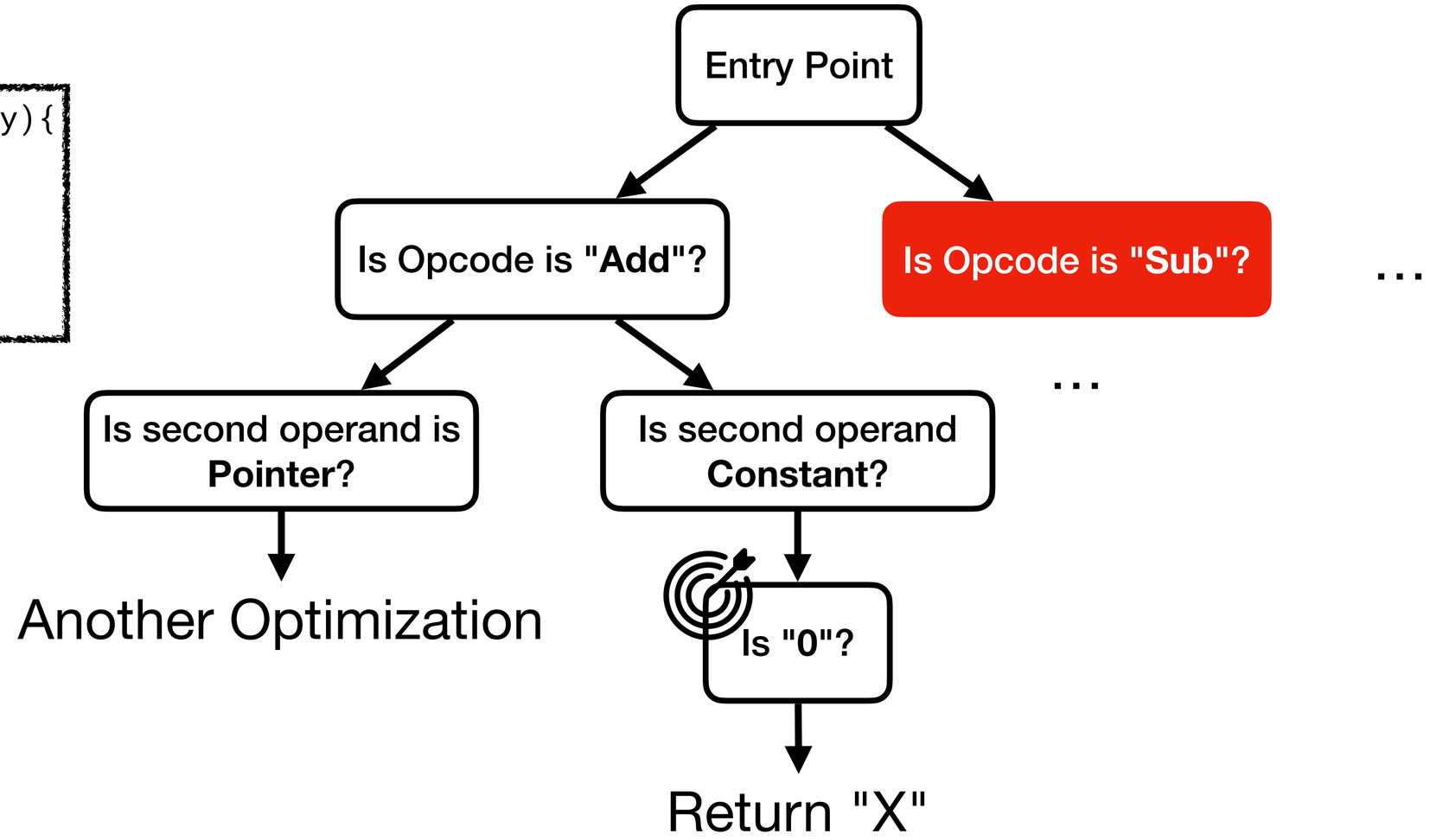


Our Guide Strategy

Target Optimization: $X + 0 \rightarrow X$

```
define i32 @mutation(i32 %x,i32 %y){
entry:
    %1 = Sub i32 %x, %y
    ret i32 %1
}
```

Mutant

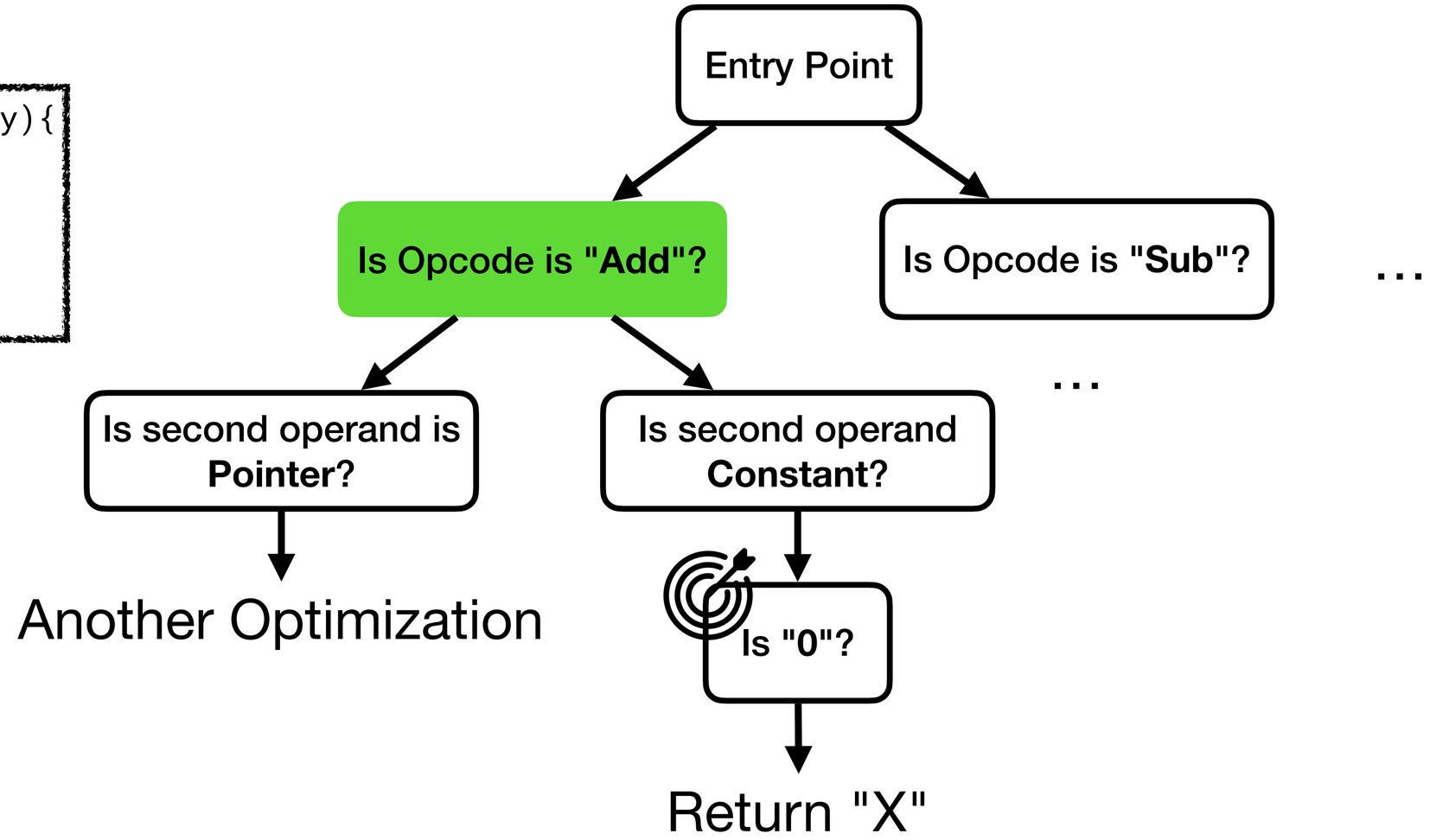


Our Guide Strategy

Target Optimization: $X + 0 \rightarrow X$

```
define i32 @mutation(i32 %x,i32 %y){
entry:
    %1 = Add i32 %x, %y
    ret i32 %1
}
```

Mutant

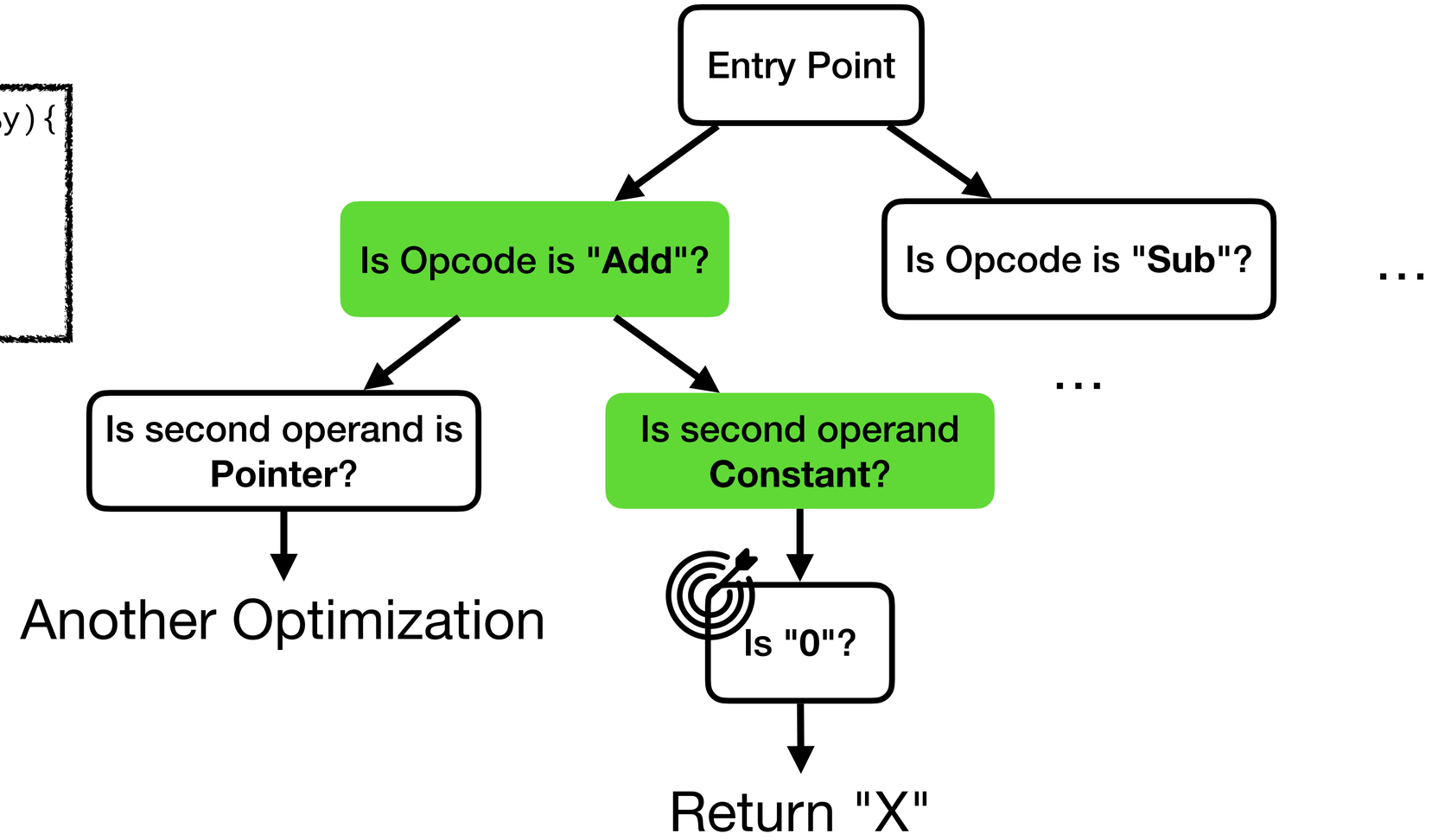


Our Guide Strategy

Target Optimization: $X + 0 \rightarrow X$

```
define i32 @mutation(i32 %x,i32 %y){
entry:
    %1 = Add i32 %x, 3
    ret i32 %1
}
```

Mutant

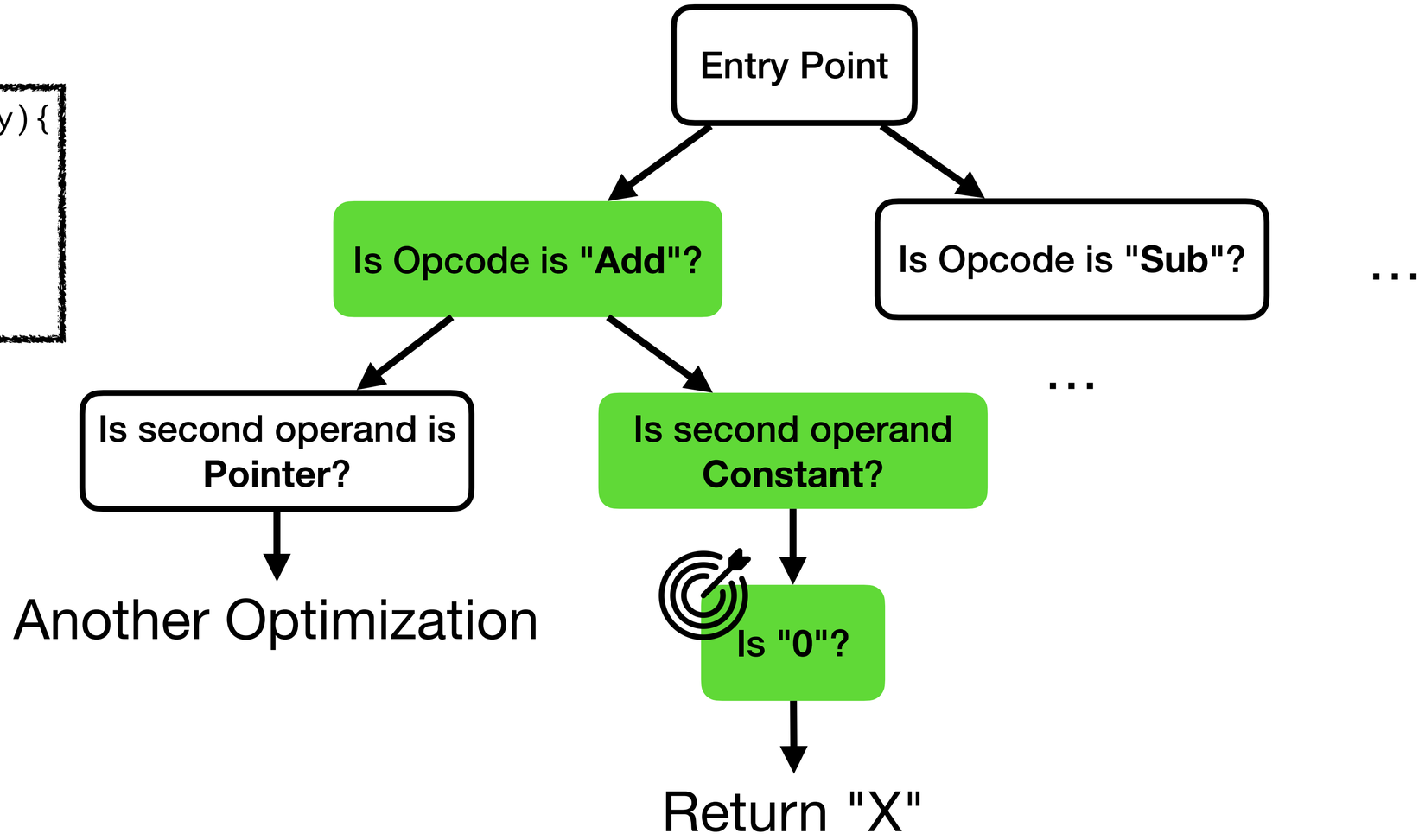


Our Guide Strategy

Target Optimization: $X + 0 \rightarrow X$

```
define i32 @mutation(i32 %x,i32 %y){
entry:
    %1 = Add i32 %x, 0
    ret i32 %1
}
```

Mutant



How can we evaluate the guide performance?

How can we evaluate the guide performance?

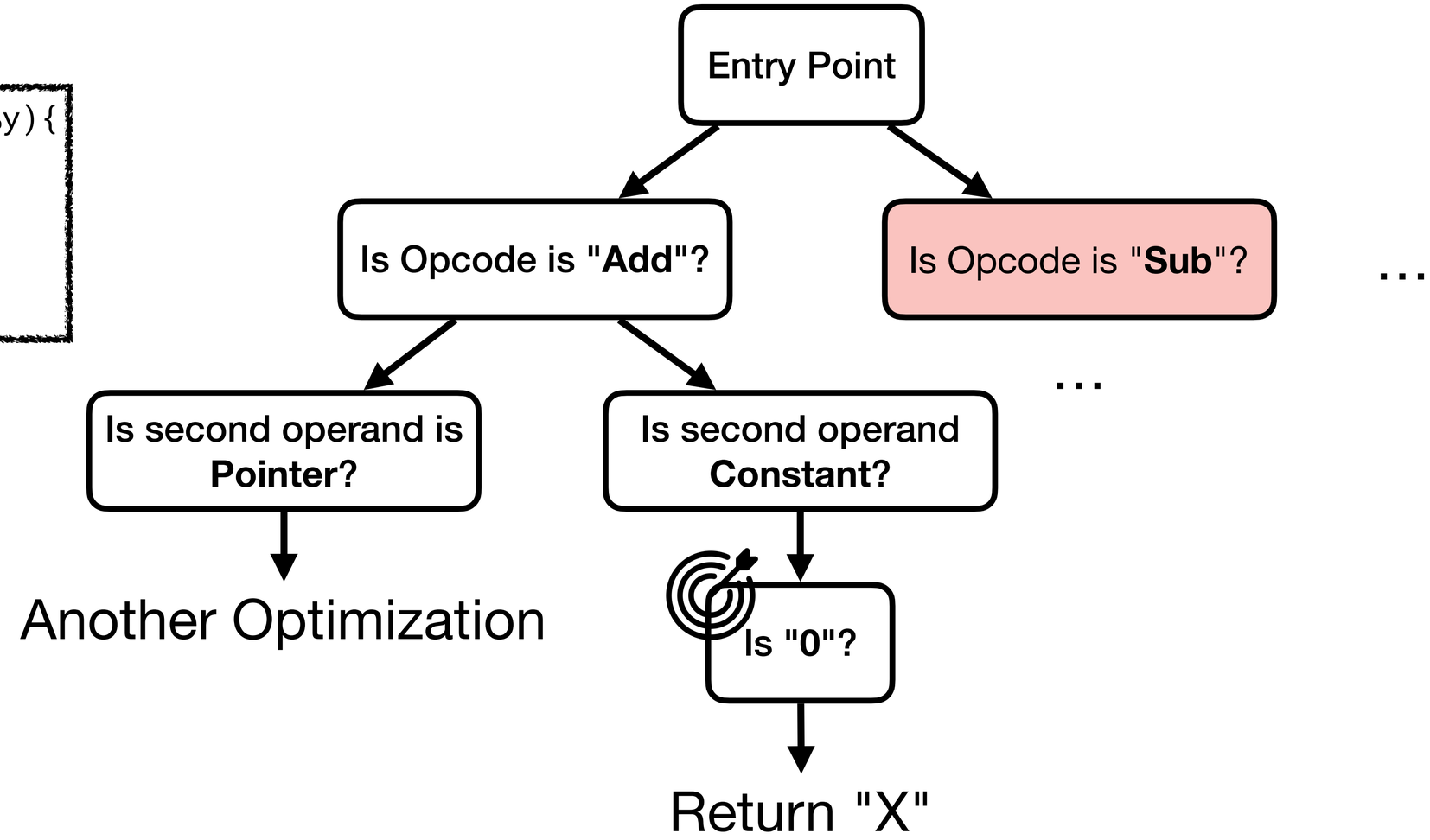
Visualize guide performance using **Heatmap**!

Visualize Guide Performance - Heatmap

Target Optimization: $X + 0 \rightarrow X$

```
define i32 @mutation(i32 %x,i32 %y){
entry:
    %1 = Sub i32 %x, %y
    ret i32 %1
}
```

Mutant

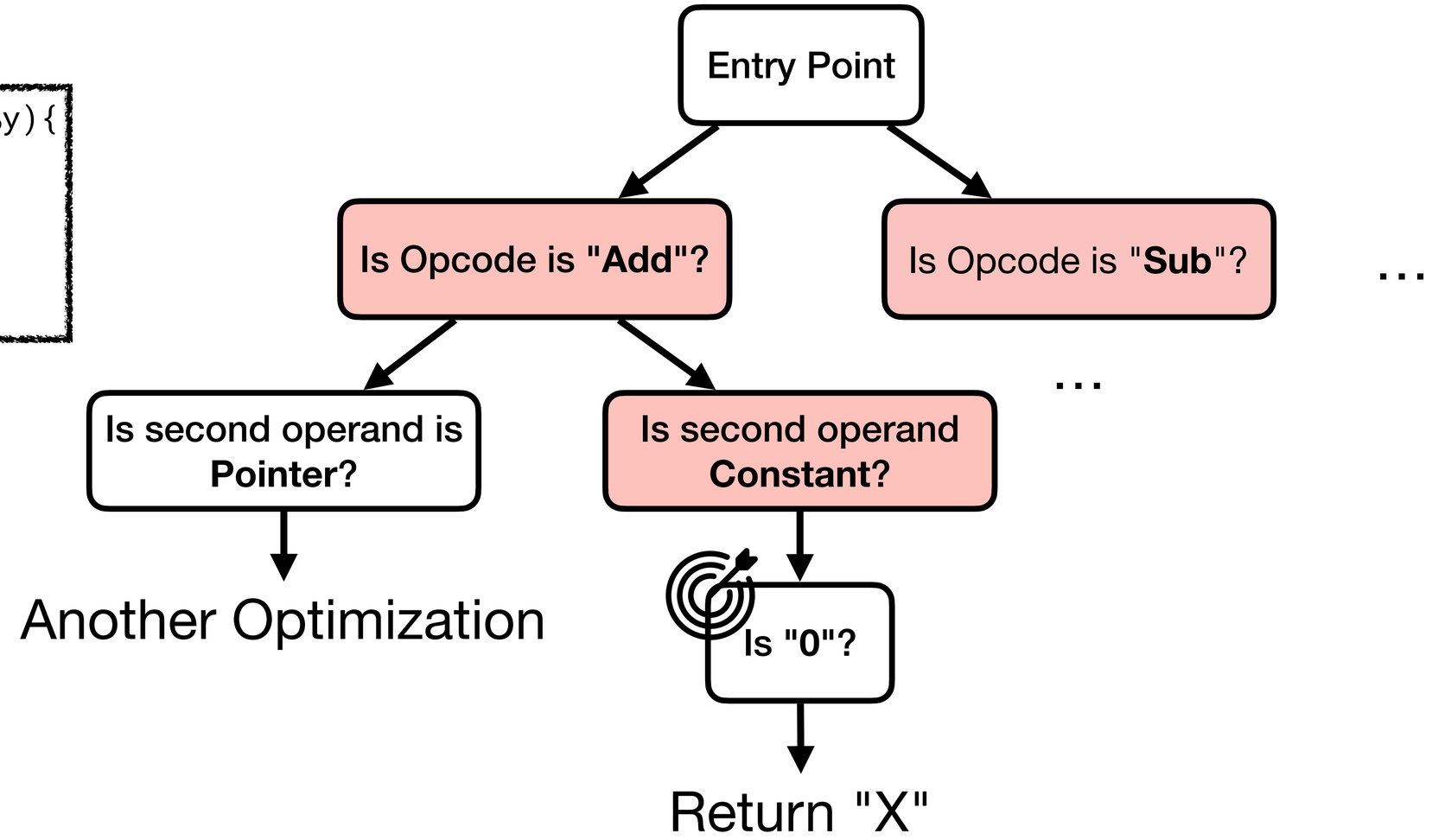


Visualize Guide Performance - Heatmap

Target Optimization: $X + 0 \rightarrow X$

```
define i32 @mutation(i32 %x,i32 %y){
entry:
    %1 = Add i32 %x, %x
    ret i32 %1
}
```

Mutant

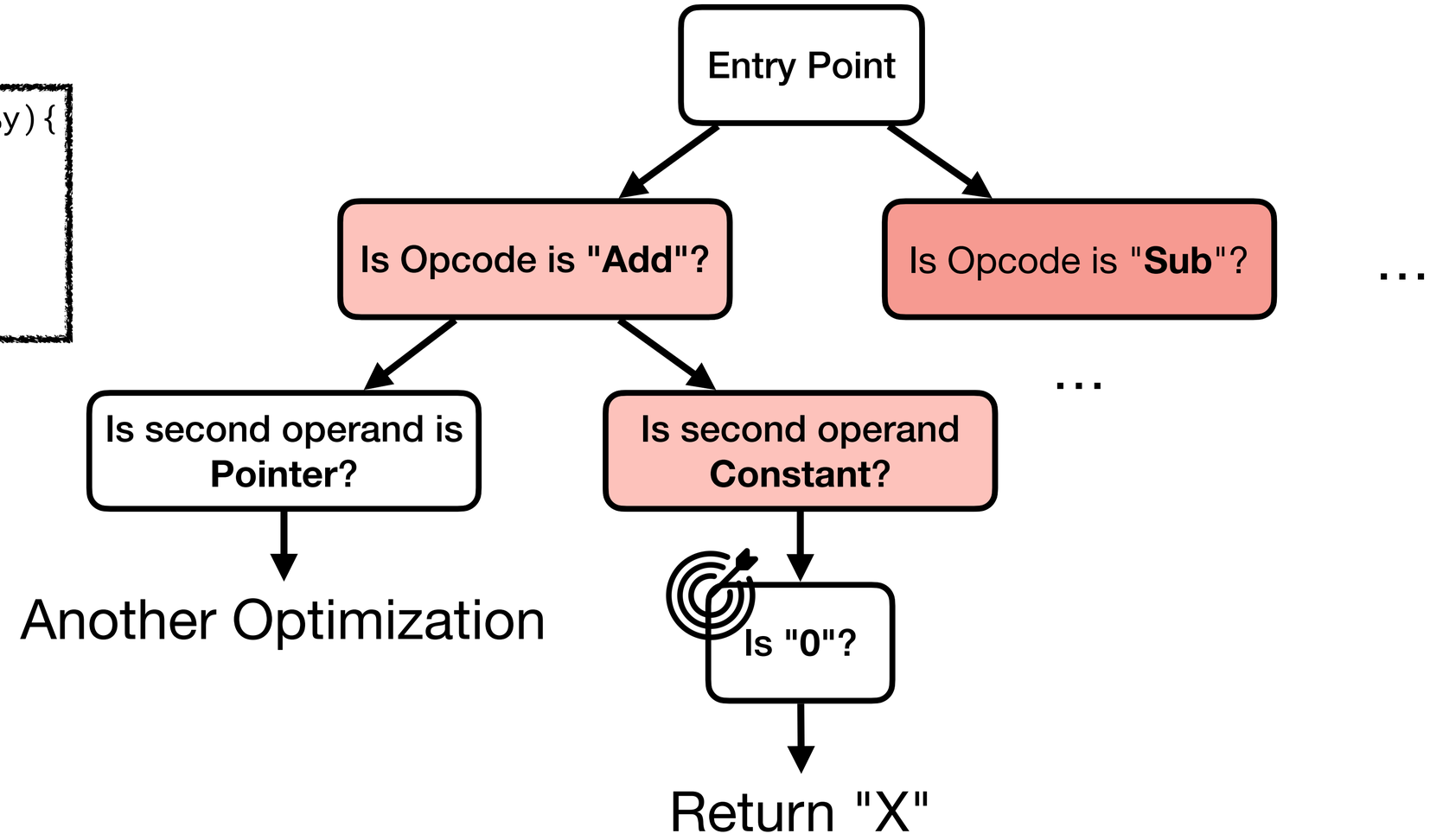


Visualize Guide Performance - Heatmap

Target Optimization: $X + 0 \rightarrow X$

```
define i32 @mutation(i32 %x,i32 %y){
entry:
    %1 = Sub i32 %y, %x
    ret i32 %1
}
```

Mutant

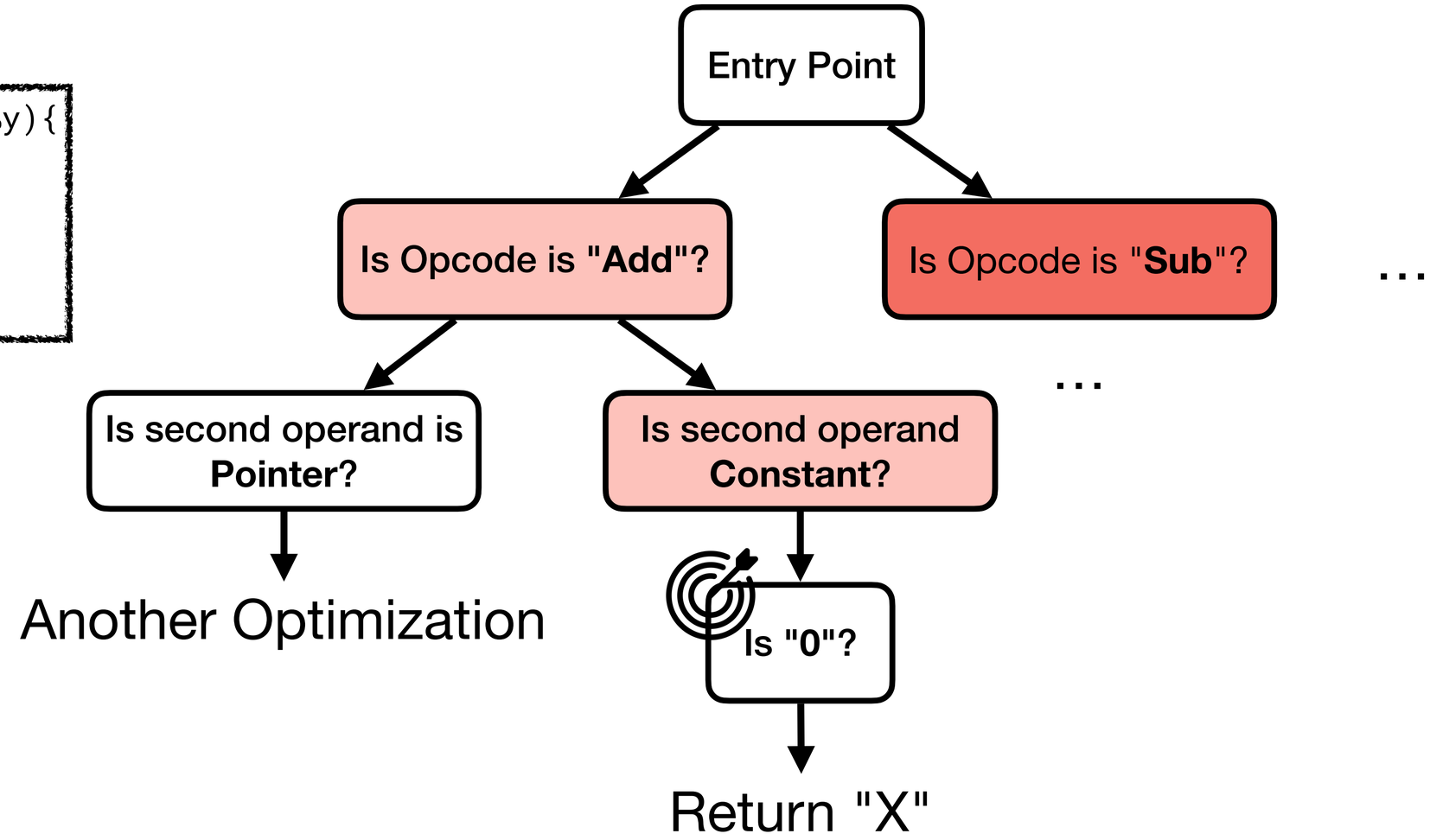


Visualize Guide Performance - Heatmap

Target Optimization: $X + 0 \rightarrow X$

```
define i32 @mutation(i32 %x,i32 %y){
entry:
    %1 = Sub i32 %y, 3
    ret i32 %1
}
```

Mutant

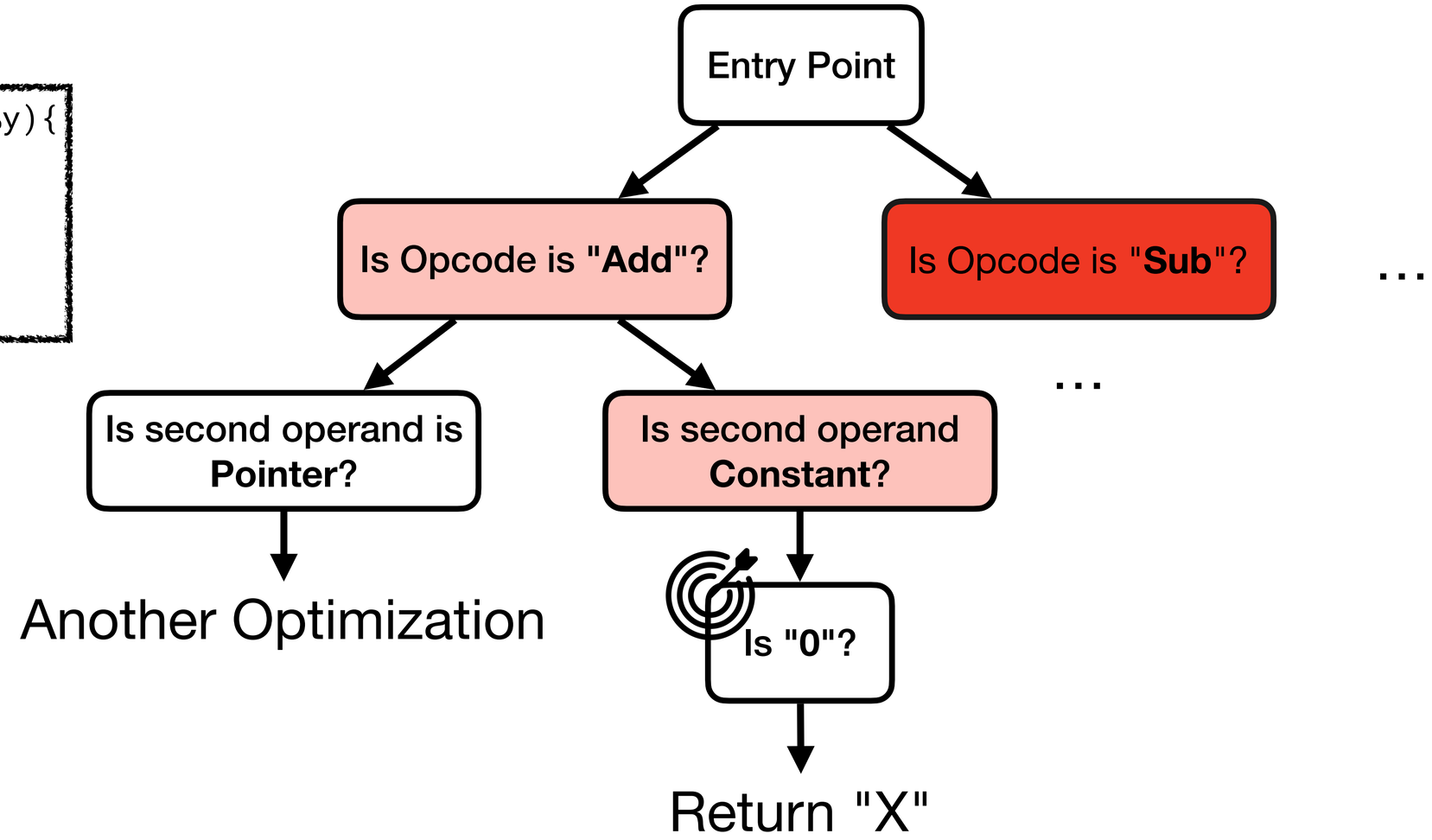


Visualize Guide Performance - Heatmap

Target Optimization: $X + 0 \rightarrow X$

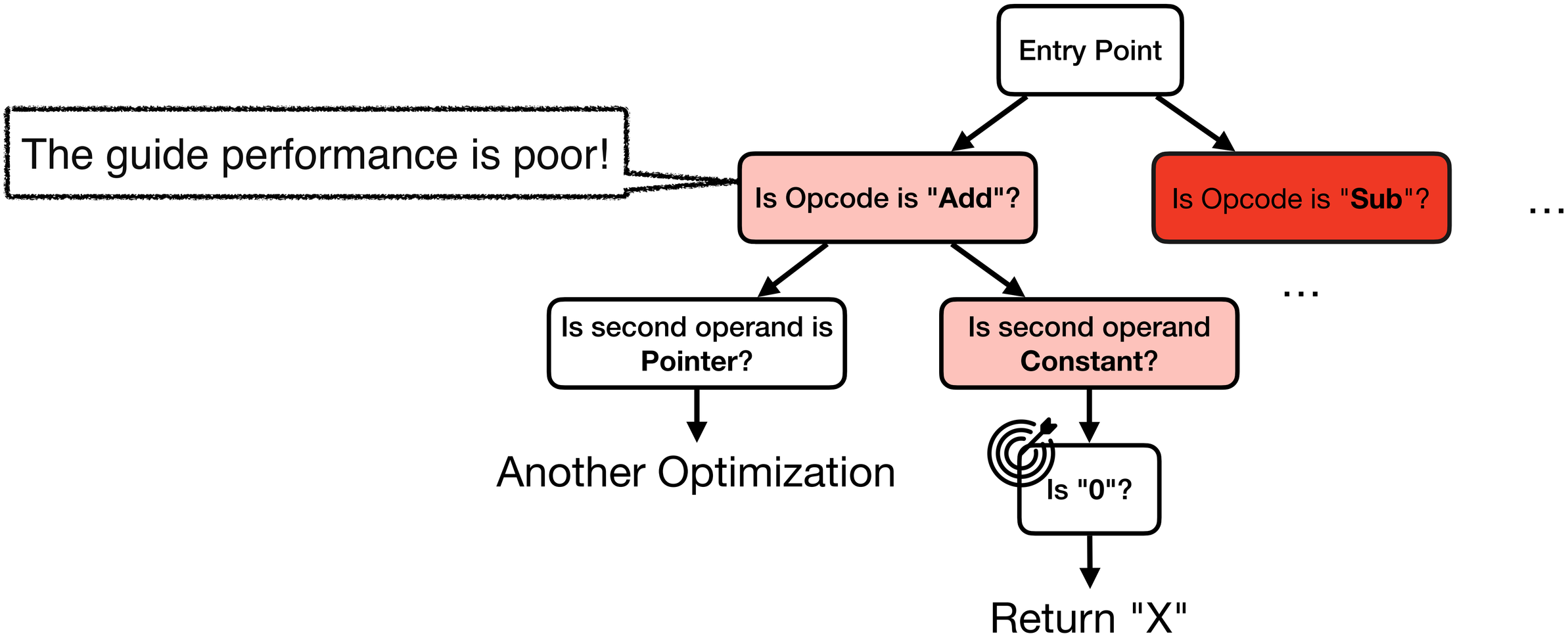
```
define i32 @mutation(i32 %x,i32 %y){
entry:
    %1 = Sub i32 5, 3
    ret i32 %1
}
```

Mutant



Visualize Guide Performance - Heatmap

Target Optimization: $X + 0 \rightarrow X$

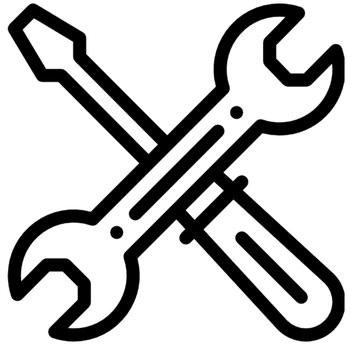


Expected Results & How to Evaluate

Easy Understanding

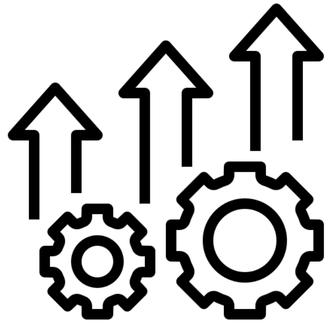
Reduce **30%** time for new developer

Maintenance



50% faster debugging speed

Improvement



Increase the program improvement speed by **20%**

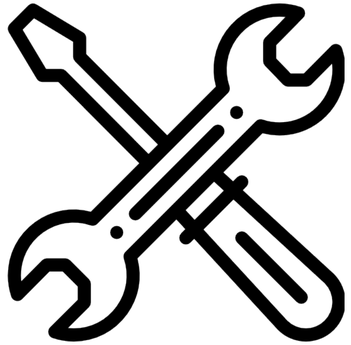
Expected Results & How to Evaluate

Easy Understanding

Reduce **30%** time for new developer

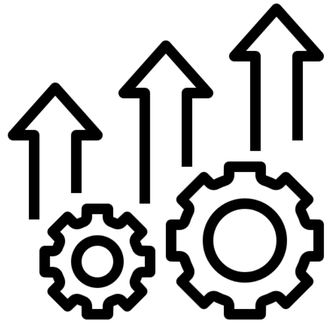
Measure the **actual time** required
for a new undergraduate researcher

Maintenance



50% faster debugging speed

Improvement



Increase the program improvement speed by **20%**

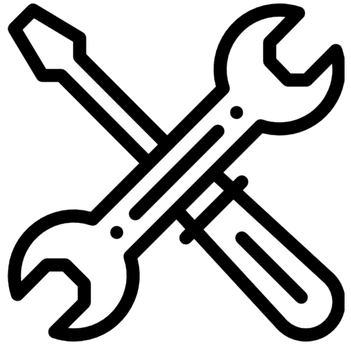
Expected Results & How to Evaluate

Easy Understanding

Reduce **30%** time for new developer

Measure the **actual time** required for a new undergraduate researcher

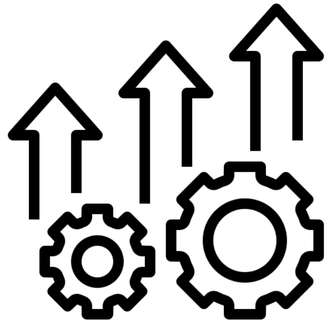
Maintenance



50% faster debugging speed

Measure the **number of issues** resolved within the same time

Improvement



Increase the program improvement speed by **20%**

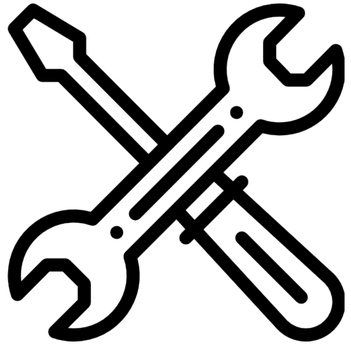
Expected Results & How to Evaluate

Easy Understanding

Reduce **30%** time for new developer

Measure the **actual time** required for a new undergraduate researcher

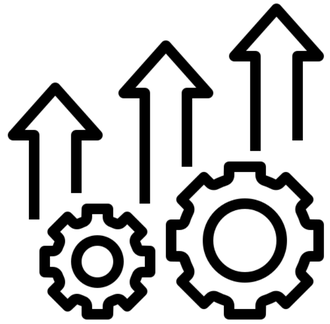
Maintenance



50% faster debugging speed

Measure the **number of issues** resolved within the same time

Improvement



Increase the program improvement speed by **20%**

Compare the time taken for adding new features

Out Team & Our Plan

- Two master’s students & One undergraduate student
  - Coding expert, available for full-time research engagement

Out Team & Our Plan

- Two master's students & One undergraduate student
  - Coding expert, available for full-time research engagement

- It is possible to finish the visualization research within **3** months!

Conclusion

- We should test the correctness of compiler optimizations.

Conclusion

- We should test the correctness of compiler optimizations.
- Our tool, Optimization-directed Fuzzer, is the hope for this problem

Conclusion

- We should test the correctness of compiler optimizations.
- Our tool, Optimization-directed Fuzzer, is the hope for this problem
- To improve our tool, visualization is necessary
  - Visualize Mutation Statistic
  - Conditional Statement Tree Heat Map

Conclusion

- We should test the correctness of compiler optimizations.
- Our tool, Optimization-directed Fuzzer, is the hope for this problem
- To improve our tool, visualization is necessary
  - Visualize Mutation Statistic
  - Conditional Statement Tree Heat Map
- We are world-class research team, we will finish within **3** months