

# Fuzzing Loop Optimizations in Compilers for C++ and Data-Parallel Languages

Vsevolod Livinskii, University of Utah

Dmitry Babokin, Intel Corporation

John Regehr, University of Utah



June 19<sup>th</sup>, 2023



# Importance of Testing Loop Optimizations

- Advancements in AI and ML fields
- New architectures with vector operations
- Complexity of loop optimizations
  - SCEV and Polly in LLVM

# Summary of Found Bugs

122 new errors in total  
42% are wrong code bugs

- 66 bugs in GCC
  - 32 wrong code, 31 ICE, 3 timeouts
- 28 bugs in LLVM
  - 5 wrong code, 5 ICE
- 12 bugs in ISPC
  - 5 wrong code, 7 ICE
- 16 bugs in Intel<sup>®</sup> oneAPI DPC++ compiler
  - 9 wrong code, 7 ICE
- 2 bugs in Intel<sup>®</sup> SDE
- 2 bugs in Alive2

# Research Contribution and Features

- New static Undefined Behavior avoidance for loops
- Target loop optimizations explicitly
- Support multiple C-family languages

# Research Contribution and Features

- New static Undefined Behavior avoidance for loops
- Target loop optimizations explicitly
- Support multiple C-family languages

# Undefined Behavior (UB)

```
# include <stdio.h>

int main () {
    int x = 1;
    x = x++ + ++x;
    printf ("%d\n", x);
    return 0;
}
```

Who is wrong?

```
>$ icc test.cpp && ./a.out
```

5

```
>$ clang++ test.cpp && ./a.out
```

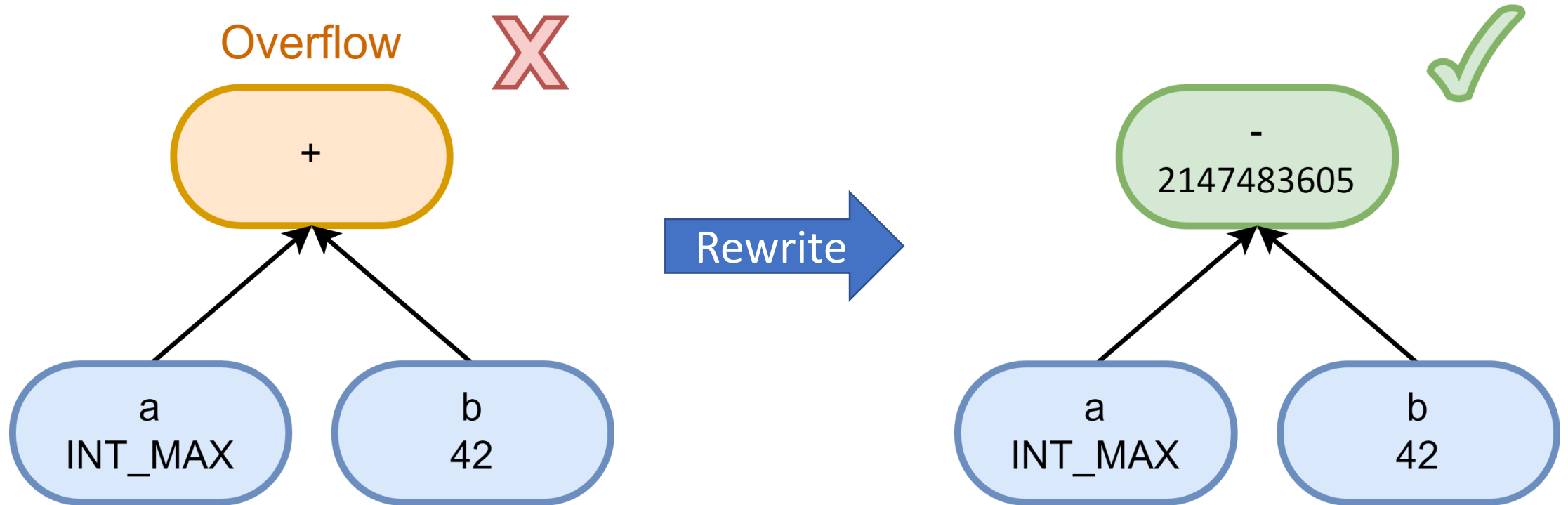
4

*No one!*

*Program contains UB*

# Static Undefined Behavior Avoidance

Based on concrete value tracking and rewrite rules



# UB Avoidance for Loops

```
var_37 = 20;  
var_43 = 99;  
...  
var_10 = (var_37 / 15) - var_43;
```



```
arr_37[20] = {20, 20, 20, ...};  
var_43 = 99;  
...  
arr_10[0] = (arr_37[0] / 15) - var_43;
```



driver.cpp

```
arr_37[20] = {20, 20, 20, ...};  
var_43 = 99;
```

...  
test.cpp

```
for (int i = 0; i < 19; ++i) {  
    arr_10[i] = (arr_37[i] / 15) - var_43;  
}
```



# UB Avoidance for Loops

i	0	1	2	3
a	5	5	5	5
b	7	7	7	7

```
c[i] = a[i] + b[i];
```

*No diversity at runtime!*

i	0	1	2	3
a	5	38	5	38
b	7	15	7	15

```
c[i] = a[i] + b[i];
```

# UB Avoidance for Loops

i	0	1	2	3
a	5	5	5	5
b	7	7	7	7

```
c[i] = a[i] + b[i];
```

*No diversity at runtime!*

i	0	1	2	3
a	5	INT_MAX	5	INT_MAX
b	INT_MIN	15	INT_MIN	15

```
c[i] = (i % 2 == zero) ?  
        (a[i] + b[i]) :  
        (a[i] - b[i]);
```

# Research Contribution and Features

- New static Undefined Behavior avoidance for loops
- Target loop optimizations explicitly
- Support multiple C-family languages

# Loop Generation Policies

- Cannot test optimizations that cannot trigger
- A total of 10 Loop Generation Policies
  - 31 fine-grain control parameters
- Not mutually exclusive; compose gracefully
  - Reduction + stencil: `a += (b[i - 1] + b[i] + b[i + 1]) / 3;`

# Loop Sequence and Loop Fusion

```
for (i=0; i < (d ? e : 10); i++)  
    a[i] = c[i] + b[i];
```

```
for (j=0; j < (d ? e : 10); j++)  
    b[j] = b[j] * c[j];
```



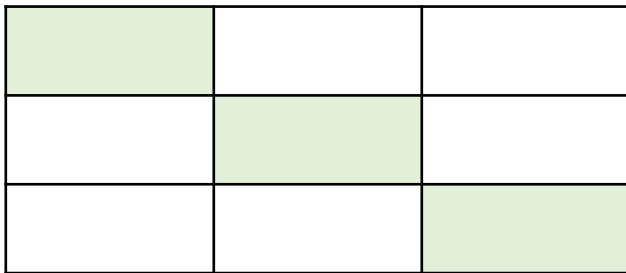
```
for (i=0; i < (d ? e : 10); i++){  
    a[i] = c[i] + b[i];  
    b[i] = b[i] * c[i];  
}
```

- Hard to generate purely at random
- Loop Sequence as first-class IR element for synchronized decisions

# Array Access Patterns

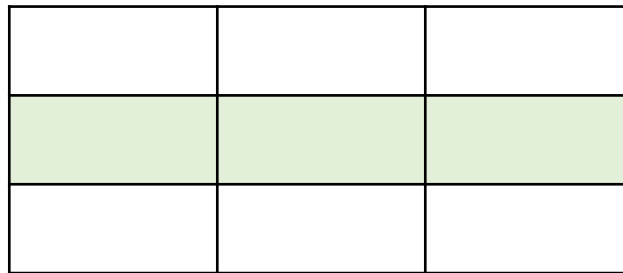
Diagonal

`a[i][i]`



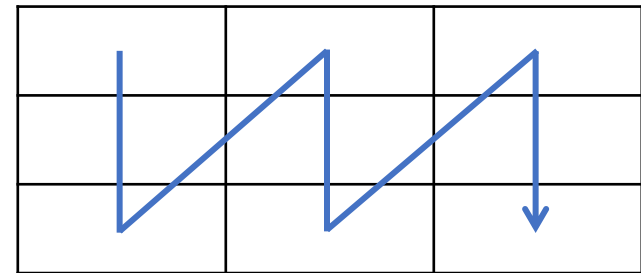
Slice

`a[1][i]`



Column-major

`a[j][i]`



- Relation between loop nest depth and dimensionality
- In-order or not
- Constant, iterator, iterator with offset

# Stencils

```
for (int i = 1; i < n - 1; ++i)
    out[i] = (in[i - 1] +
              in[i] +
              in[i + 1]) / 3;
```

GVN in LLVM forwards values to a subsequent loop iteration

Stencil as a pattern:

- arrays
- stride
- dimensions
- computations

```
.LBB0_2:
    fadd    d1, d0, d1
    fmov    d2, d0
    ldr     d0, [x9], #8
    fmov    d3, x10
    subs    x8, x8, #1
    fadd    d1, d1, d0
    fmul    d3, d1, d3
    fmov    d1, d2
    str     d3, [x1], #8
    b.ne   .LBB0_2
```

# Generation Policies Composition

```
for (int i = 0; i < a + b; i++) {  
    for (int j = 0; j < (c ? d : 10); j++)  
        e[i][j] = f[1][j][i] + f[2][i][i];  
  
    for (int k = 0; k < (c ? d : 10); k++)  
        g += (h[i][k - 1] + h[i][k] + h[i][k + 1]) / 3;  
}
```



# Research Contribution and Features

- New static Undefined Behavior avoidance for loops
- Target loop optimizations explicitly
- Support multiple C-family languages

# Multi-language Support and IR Lowering

## Matrix multiplication

$$c_{ij} = \sum_{k=1}^K a_{ik} b_{kj} ; i = 1, \dots, M ; j = 1, \dots, N$$

# Multi-language Support and IR Lowering

## C++

```
for (int i = 0; i < M; i++)  
  for (int j = 0; j < K; j++)  
    for (int k = 0; k < N; k++)  
      c[i][j] += a[i][k] * b[k][j];
```

## ISPC

```
foreach (m = 0 ... M) {  
  for (k = 0; k < K; k++) {  
    sum = 0.0f;  
    for (n = 0; n < N; n++) {  
      aValue = a[m*N + n];  
      bValue = b[n*K + k];  
      sum += aValue * bValue;  
    }  
    c[m*K + k] = sum;  
  }  
}
```

# Multi-language Support and IR Lowering

Loop #1:  $i$  in  $[0, 10)$ , step 2

If-then (d):

$a[i] = b[i] \wedge d$

Else:

$a[i] = b[i] \& d$

Loop #2:  $j$  in  $[0, 10)$ , step 2

$c[i] = b[j] + 134$

Lowering



```
for (int i = 0; i < 10; i += 2){  
    if (d)  
        a[i] = b[i] ^ d;  
    else  
        a[i] = b[i] & d;  
}  
for (int j = 0; j < 10; j += 2)  
    c[i] = b[j] + 134;
```

- C-family languages have similar UB rules
- High-level IR is (mostly) independent from target languages
  - contains common information

# Limitations

- No floating-point support
- Only stdlib function calls
- Lack of dynamic memory allocation
- ...

Some are research questions; others require more engineering resources

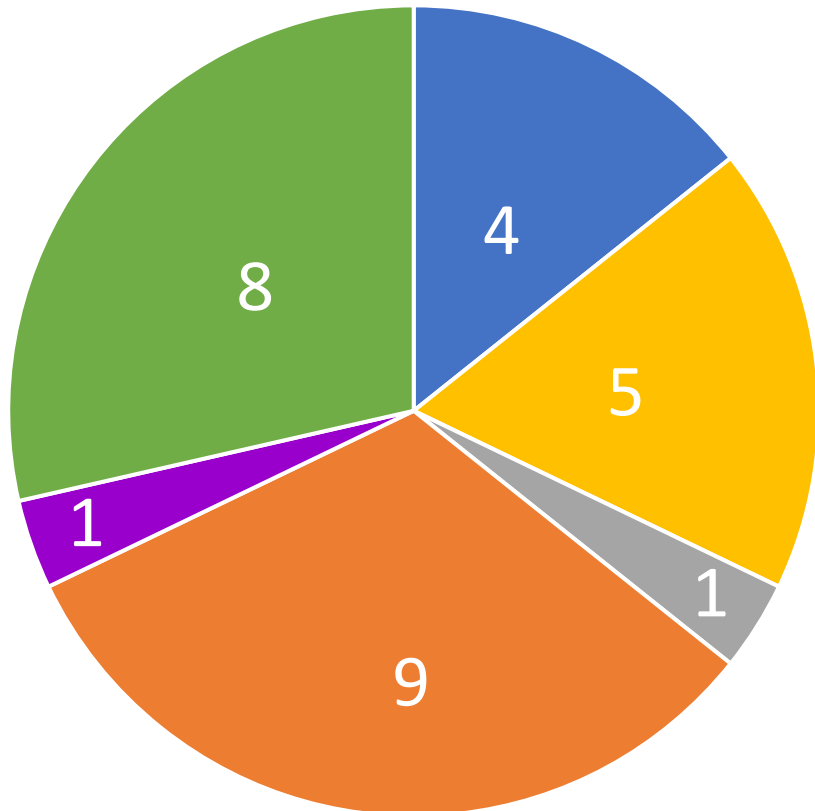
# Summary of Found Bugs

122 new errors in total  
42% are wrong code bugs

- 66 bugs in GCC
  - 32 wrong code, 31 ICE, 3 timeouts
- 28 bugs in LLVM
  - 5 wrong code, 5 ICE
- 12 bugs in ISPC
  - 5 wrong code, 7 ICE
- 16 bugs in Intel<sup>®</sup> oneAPI DPC++ compiler
  - 9 wrong code, 7 ICE
- 2 bugs in Intel<sup>®</sup> SDE
- 2 bugs in Alive2

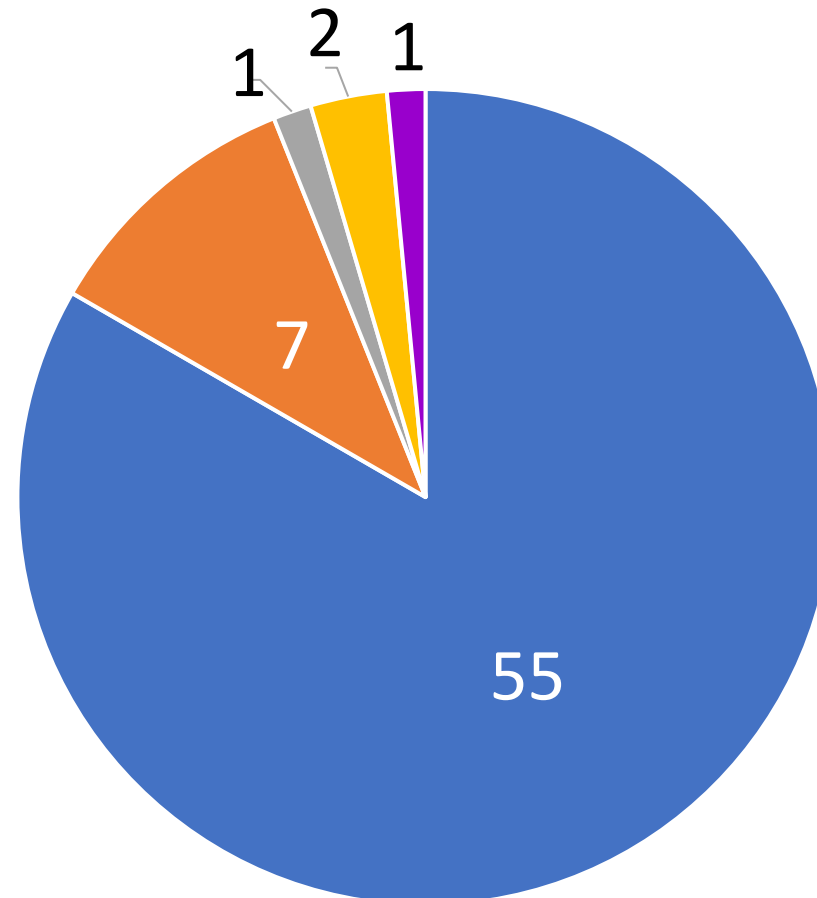
# Bugs Distribution by Components

LLVM (28 bugs)



- LoopOptimizer
- Polly Optimizer
- Scalar Optimization
- Backend: X86
- isl
- new-bugs

GCC (66 bugs)



- tree-optimization
- target
- rtl-optimization
- ipa
- c++

# Analysis of Reported Bugs

The most common reasons for bugs are:

- Missed corner-cases
- Use of corrupted information
- Too weak preconditions



# Analysis of Reported Bugs

The most common reasons for bugs are:

- Missed corner-cases
  - Examples: INT\_MIN, back-edges
- Use of corrupted information
- Too weak preconditions

# Analysis of Reported Bugs

The most common reasons for bugs are:

- Missed corner-cases
- Use of corrupted information
  - Analyses are computationally expensive
  - Cache invalidation is hard
- Too weak preconditions

# Analysis of Reported Bugs

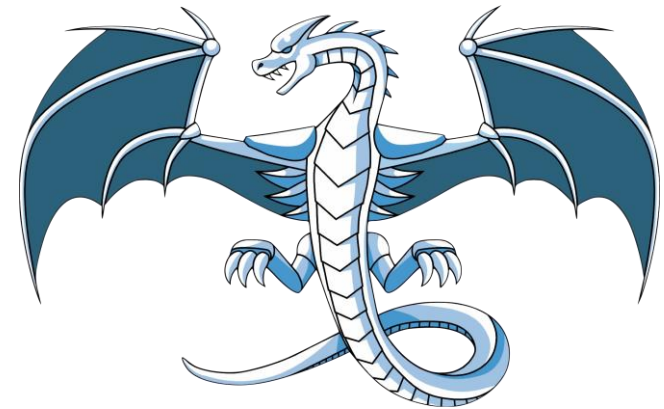
The most common reasons for bugs are:

- Missed corner-cases
- Use of corrupted information
- Too weak preconditions
  - Examples: vector size or type mismatch, bool is special



<https://github.com/intel/yarpgen>

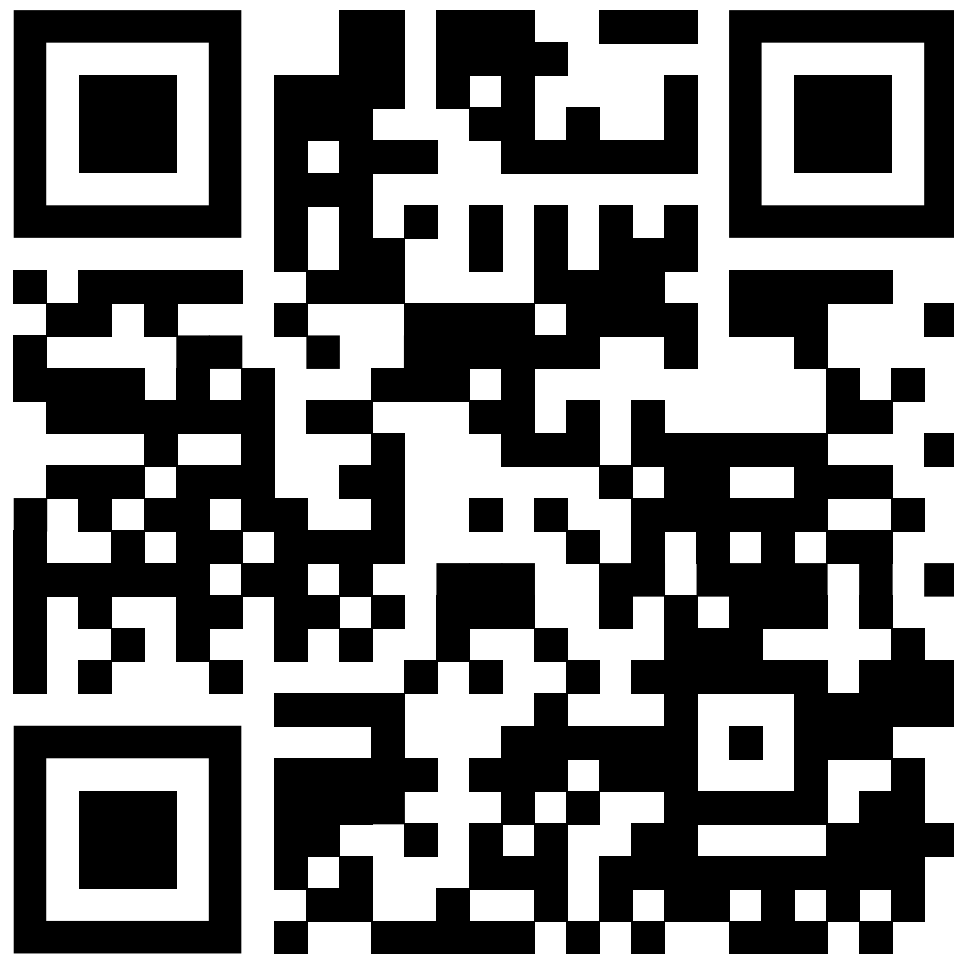
Special thanks to Intel,  
GCC and LLVM developers  
who fix reported bugs!



# Looking for Job

- Expected graduation: Fall 2023
- CV: [livinskii.com/#cv](https://livinskii.com/#cv)
- Email: [Vsevolod.Livinskii@gmail.com](mailto:Vsevolod.Livinskii@gmail.com)



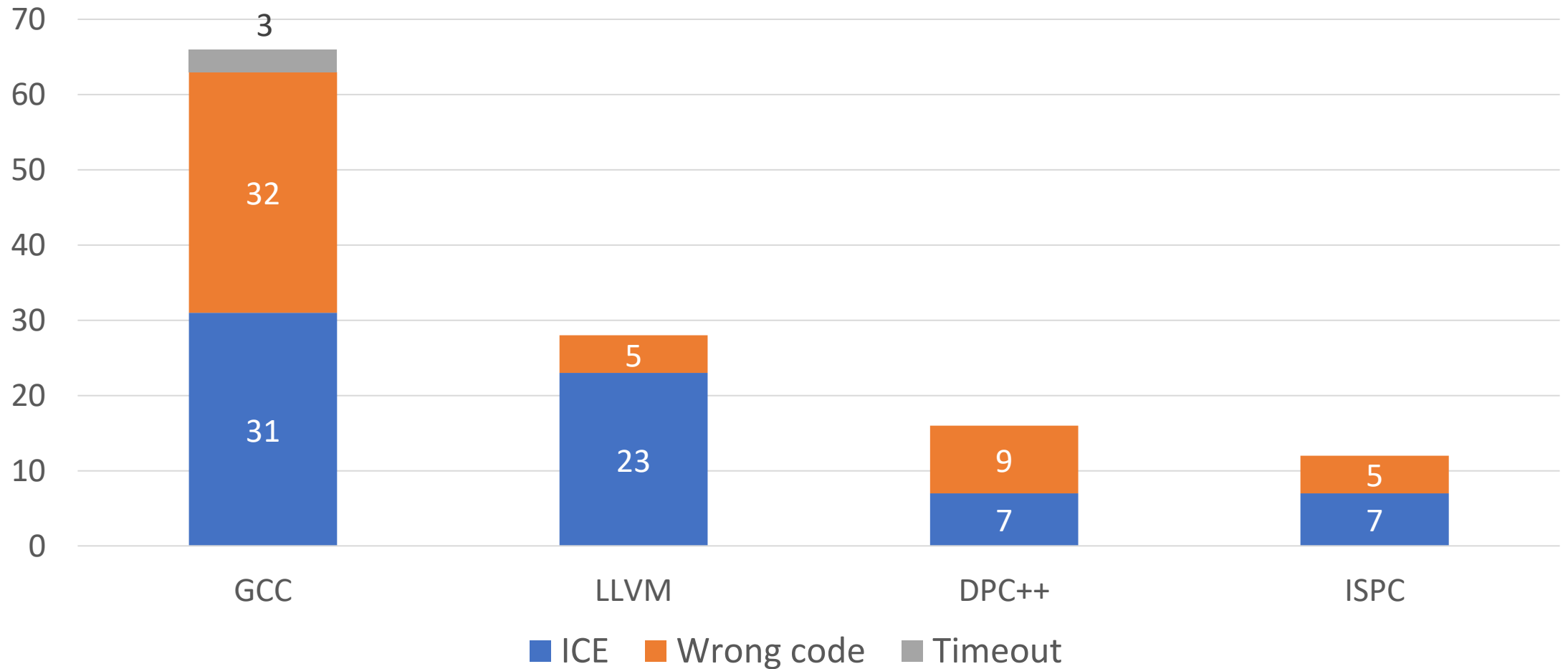


<https://github.com/intel/yarpgen>

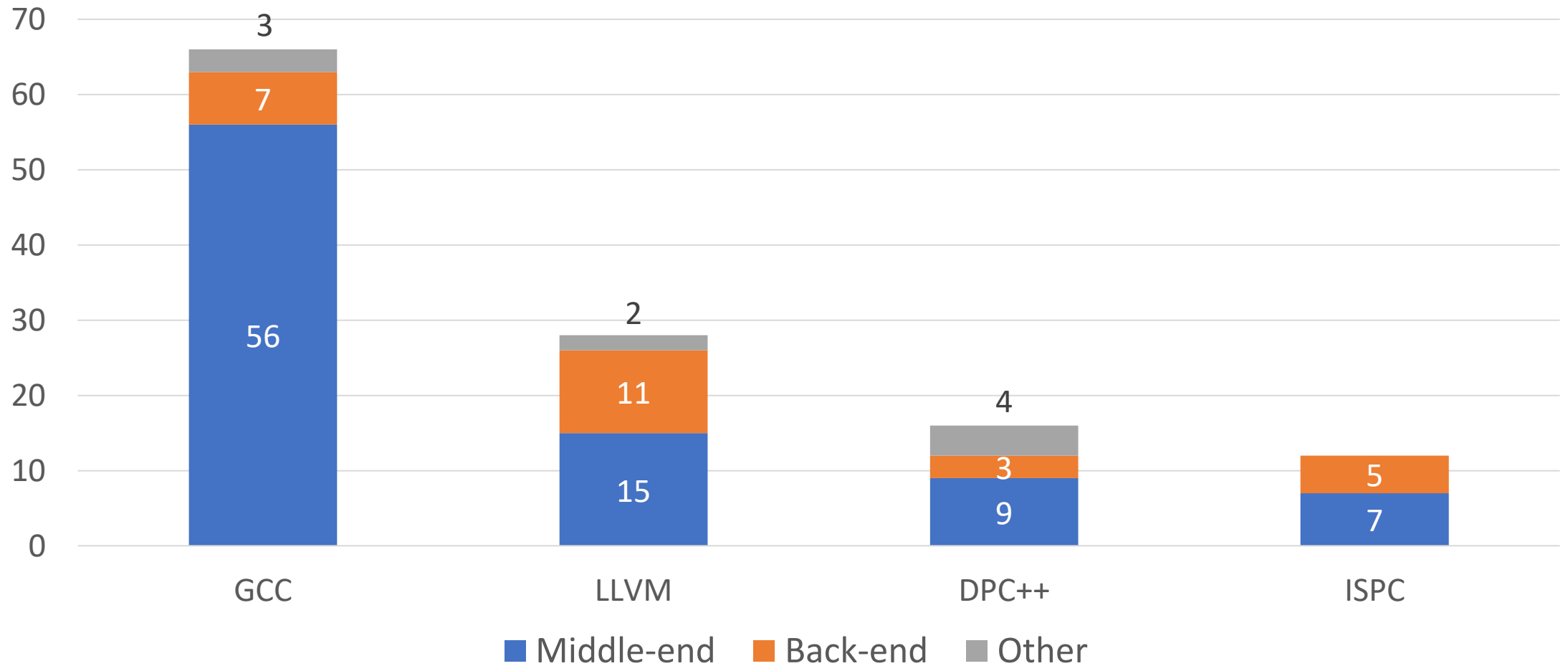
# Backup slides



# Bugs Distribution by Kind



# Bugs Distribution by Component



# Loop Optimizations Coverage

- 238 loop-related optimization counters in LLVM

Test suite + SPEC <sup>®</sup> CPU2017	YARPGen with GP	YARPGen without GP
80	72	71

- With GP 71 is better, 1 is the same, none is worse
  - Geomean ratio is 9.14

# LLVM Bug #[51677](#)

```
void test() {  
#pragma clang loop vectorize_predicate(enable)  
  for (char a = 4; a < var_3; a++) {  
    arr_13[a] = arr_12[a - 3];  
    var_23 = arr_12[a - 1];  
  }  
}
```

```
>$ clang++ -O0 -march=skx func.cpp driver.cpp && sde -skx -- ./a.out
```

```
1
```

```
>$ clang++ -O1 -march=skx func.cpp driver.cpp && sde -skx -- ./a.out
```

```
0
```

# GCC Bug #[102920](#)

```
void test (unsigned short a, unsigned short b, long long c) {  
    for (char i = 0; i < (char)c; i += 5) {  
        if (!b)  
            var_120 = a;  
        else  
            var_123 = a;  
    }  
}
```

```
>$ g++ -O3 small.cpp && ./a.out
```

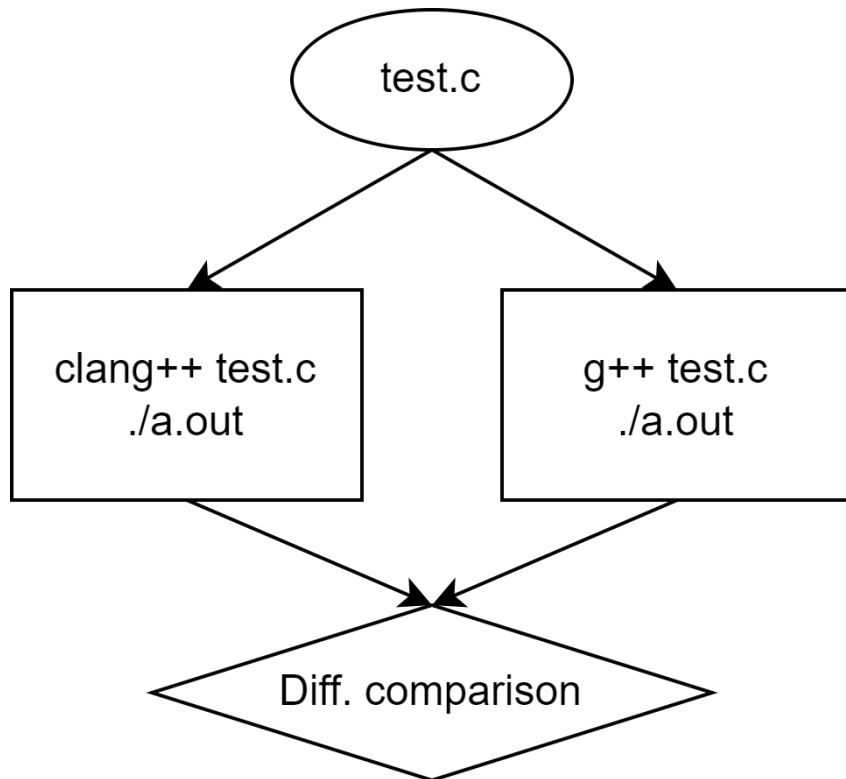
```
0
```

```
>$ g++ -O2 small.cpp && ./a.out
```

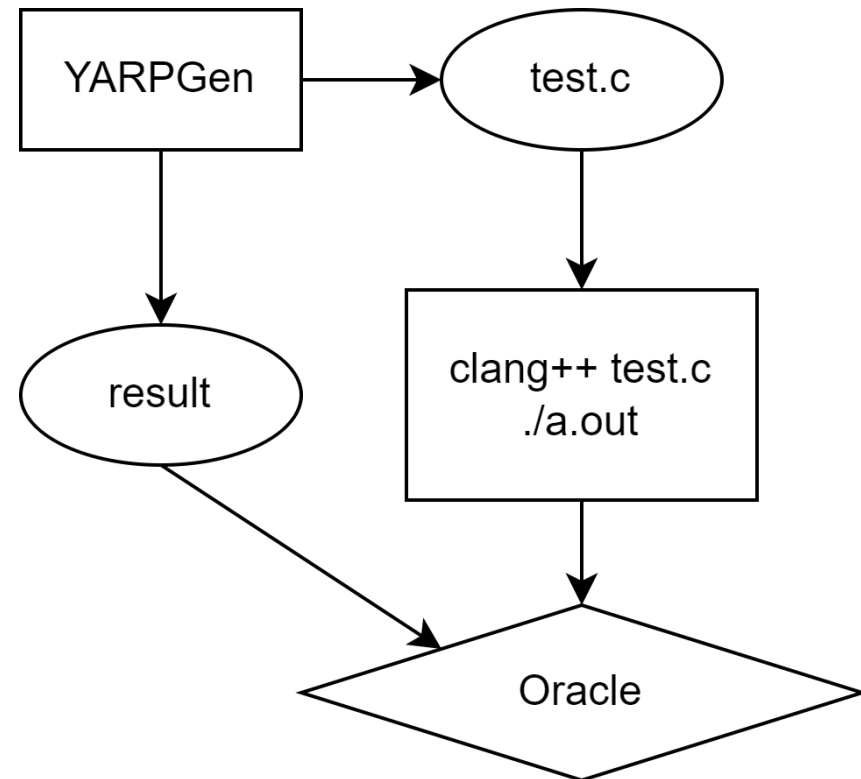
```
42
```

# Test Oracles

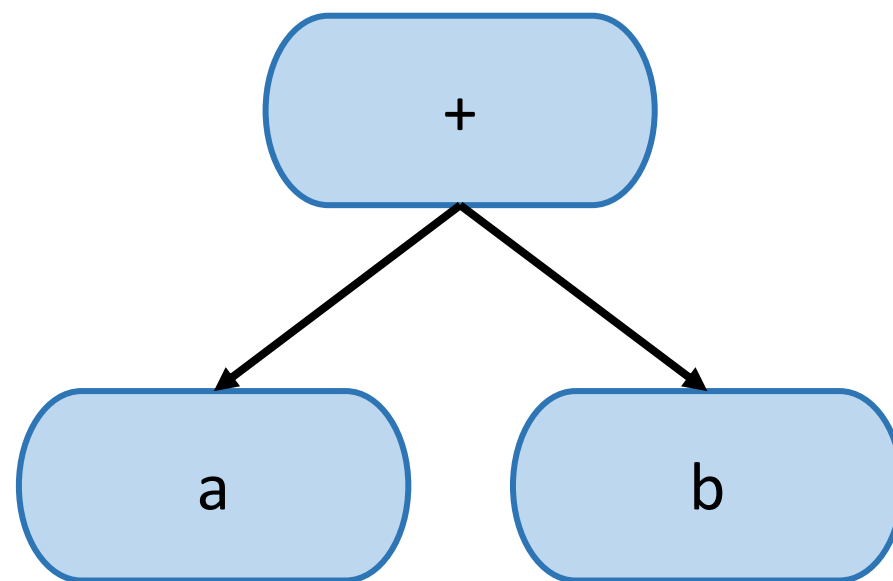
## Differential testing



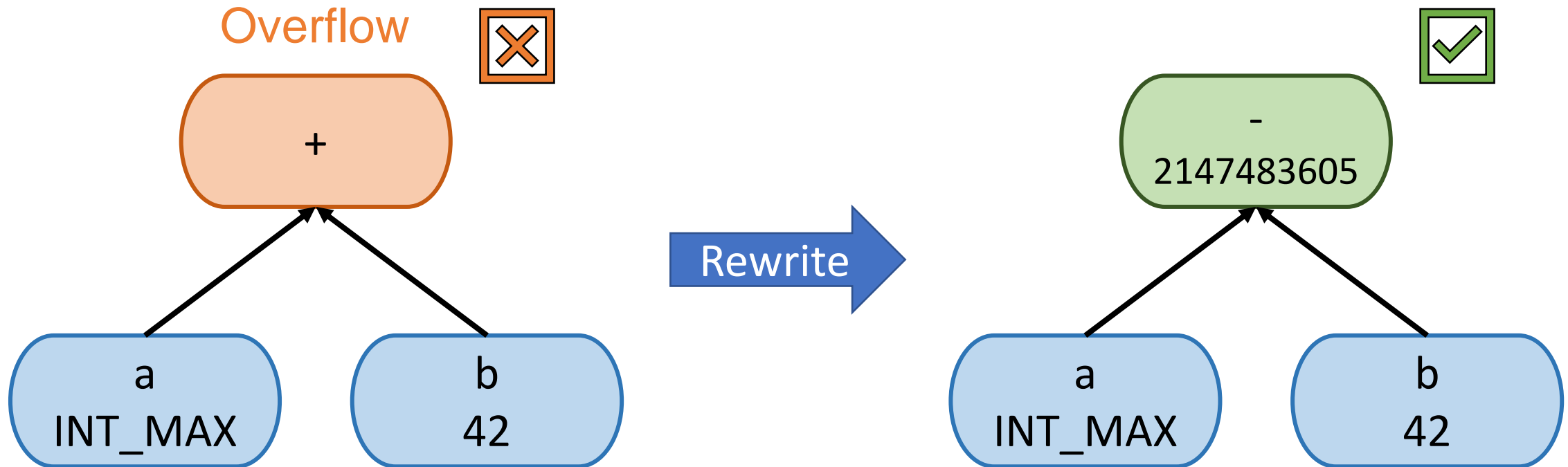
## Ground truth



# Arithmetic Expression Tree

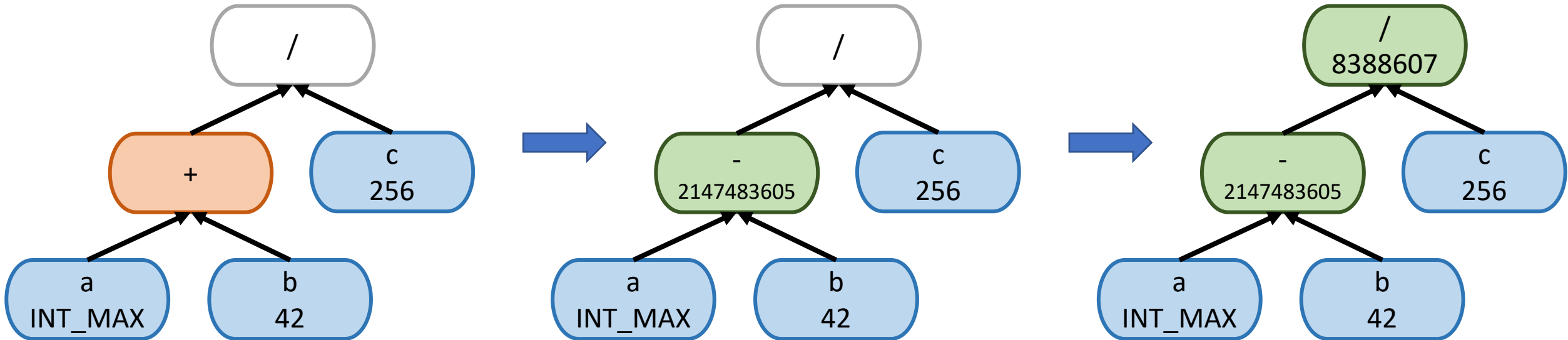


# Undefined Behavior Avoidance





# Undefined Behavior Avoidance



# Rewrite Rules

Operation	Unsafe condition	Signed or unsigned?	Replacement
-a	a == MIN	S	+a
a + b	a + b > MAX    a + b < MIN	S	a - b
a - b	a - b > MAX    a - b < MIN	S	a + b
a * b	a * b > MAX    a * b < MIN, where a != MIN && b != -1	S	a / b
a * b	a == MIN && b == -1	S	a - b
a / b	b == 0	S or U	a * b
a / b	a == MIN && b == -1	S	a - b
a % b	b == 0	S or U	a * b
a % b	a == MIN && b == -1	S	a - b
a << b	MIN < b < 0	a is U && b is S	a << (b + c), where c ∈ [-b; -b + bit_width(a)]
a << b	MIN < b < 0	a is S && b is S	a << (b + c), where c ∈ [-b; -b + bit_width(a) - MSB(a)]
a << b	b == MIN	a is U or S && b is S	a
a << b	b >= bit_width(a)	a is U && b is U or S	a << (b - c), where c ∈ (b - bit_width(a); b]
a << b	b >= bit_width(a)	a is S && b is U or S	a << (b - c), where c ∈ (b - bit_width(a) + MSB(a); b]
a >> b	MIN < b < 0	a is U or S && b is S	a >> (b + c), where c ∈ [-b; -b + bit_width(a)]
a >> b	b == MIN	a is U or S && b is S	a
a >> b	b >= bit_width(a)	a is U or S && b is U or S	a >> (b - c) c ∈ (b - bit_width(a); b]
a >> b †	MIN < a < 0	a is S && b is U or S	(a + MAX) >> b
a >> b †	a == MIN	a is S && b is U or S	b

† implementation-defined behavior

# Generative Fuzzers for C

	Csmith	Orange	Quest
UB avoidance mechanism	Static analysis + wrapper functions	Static analysis	Limited subset of C
Specialization	Universal	Arithmetic expressions	Calling conventions
Oracle	Differential testing	Build-in assertions	Ground truth

# Example of a Missed Bug (GCC [#105189](#))

- Triggered with `-O1`
- Survived for almost 4 years
  - Introduced on July 23rd 2018
  - Detected on April 6th 2022

```
int foo() {  
    return -1;  
}
```

```
int main() {  
    int c = foo() >= 0U && 1;  
    if (c != 1)  
        abort ();  
}
```

# Coverage-Guided Fuzzing

