THE UNIVERSITY OF UTAH

Testing Static Analyses for Precision and Soundness

Jubi Taneja

Zhengyang Liu, John Regehr

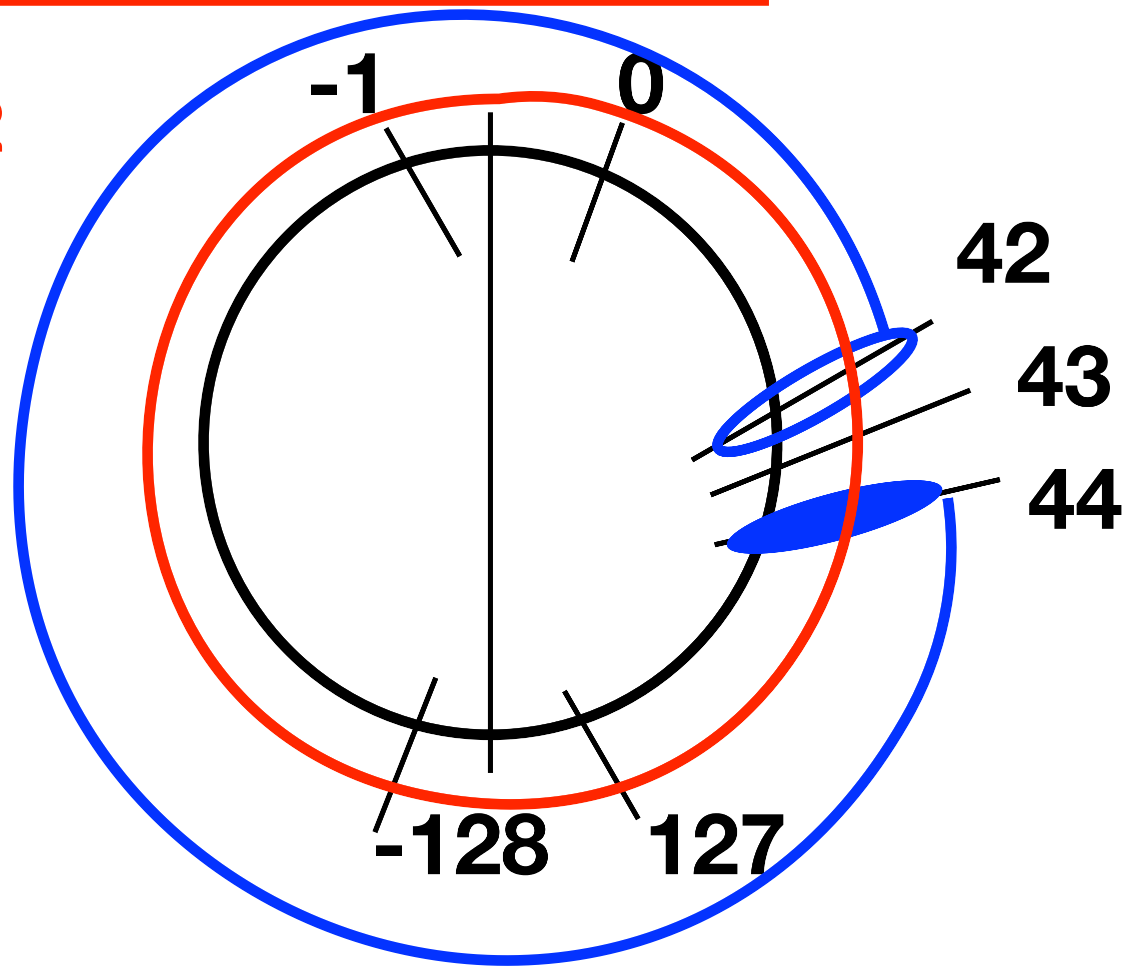
Static Analysis

- **Sound**
- **Precise**
- **Fast**

Motivation: Precision

```
define i1 @foo(i8 %x) {
entry:
  %0 = icmp eq i8 %x, 42
  %1 = icmp eq i8 %x, 43
  %2 = or i1 %0, %1
  %3 = select i1 %2, i8 1, i8 %x
  %4 = icmp eq i8 %3, 42
  ret i1 %4
}
```

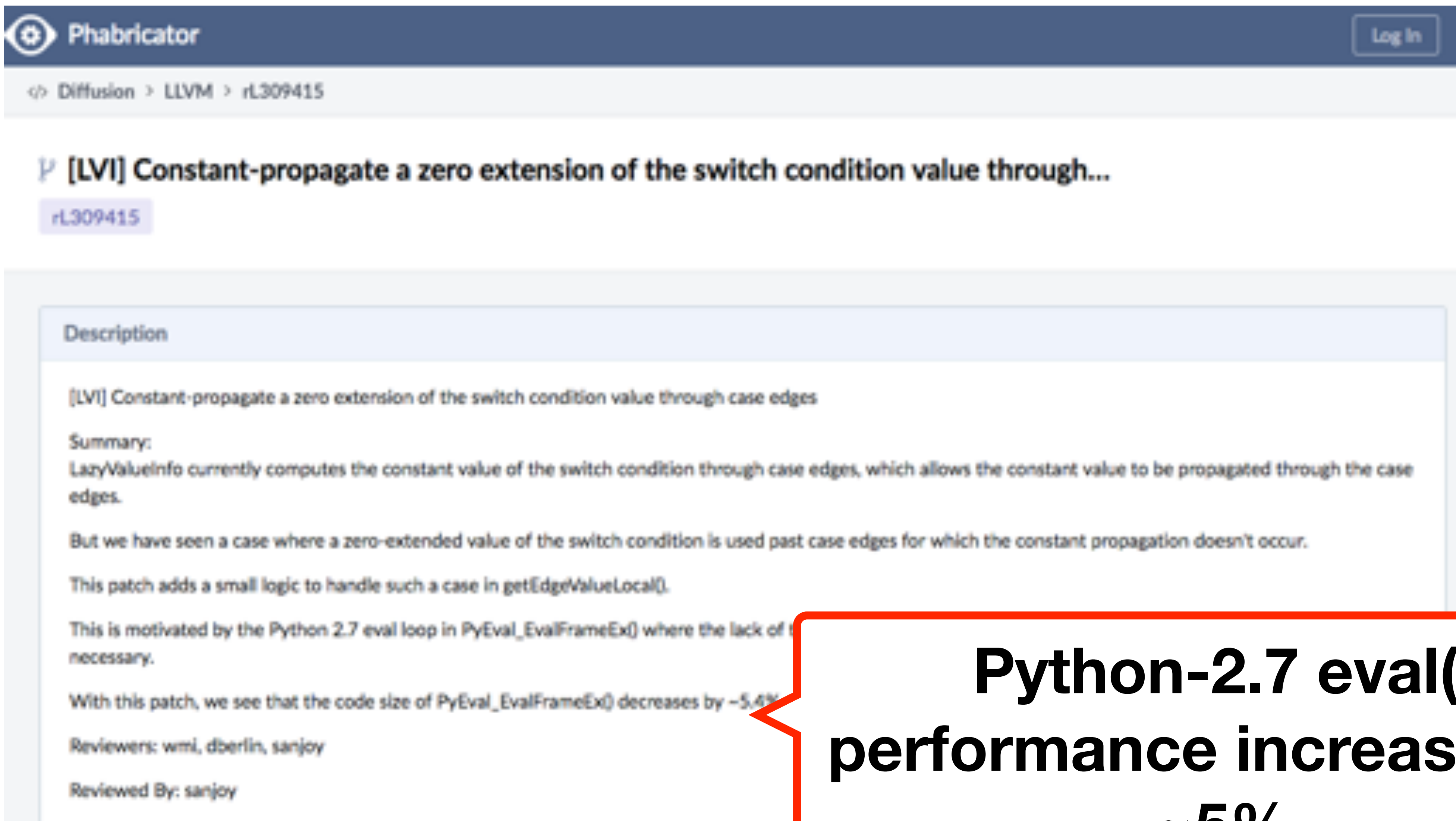
```
define i1 @foo(i8 %x) {
entry:
  %0 = icmp eq i8 %x, 42
  %1 = icmp eq i8 %x, 43
  %2 = or i1 %0, %1

  %3 = select i1 %2, i8 1, i8 %x

  %4 = icmp eq i8 %3, 42
  ret i1 %4
}
```



LLVM Result : No info

Best Result: [44, 42)

Imprecision in LLVM's Integer Range Analysis



Python-2.7 eval() performance increased by ~5%

Miscompilation Bug in LLVM

Bug 23011 - miscompile of % in loop

Status: RESOLVED FIXED

Nick Lewycky 2015-03-24 18:46:31 PDT

```
$ clang++ -v
clang version 3.7.0 (trunk 233044)
Target: x86_64-unknown-linux-gnu

Testcase:
#include <stdio.h>
#include <stdlib.h>
#include <string>

using namespace std;

int main(int argc, char **argv) {
  int r = 2;
  bool ok = true;
  while (ok) {
    string ab;
```

Motivation: Soundness

```
define i32 @foo(i32 %x) {
entry:
  %0 = srem i32 %x, 3
  ret i32 %0
}
```

Q: Number of sign bits for %0?

LLVM-3.7 Result : 31

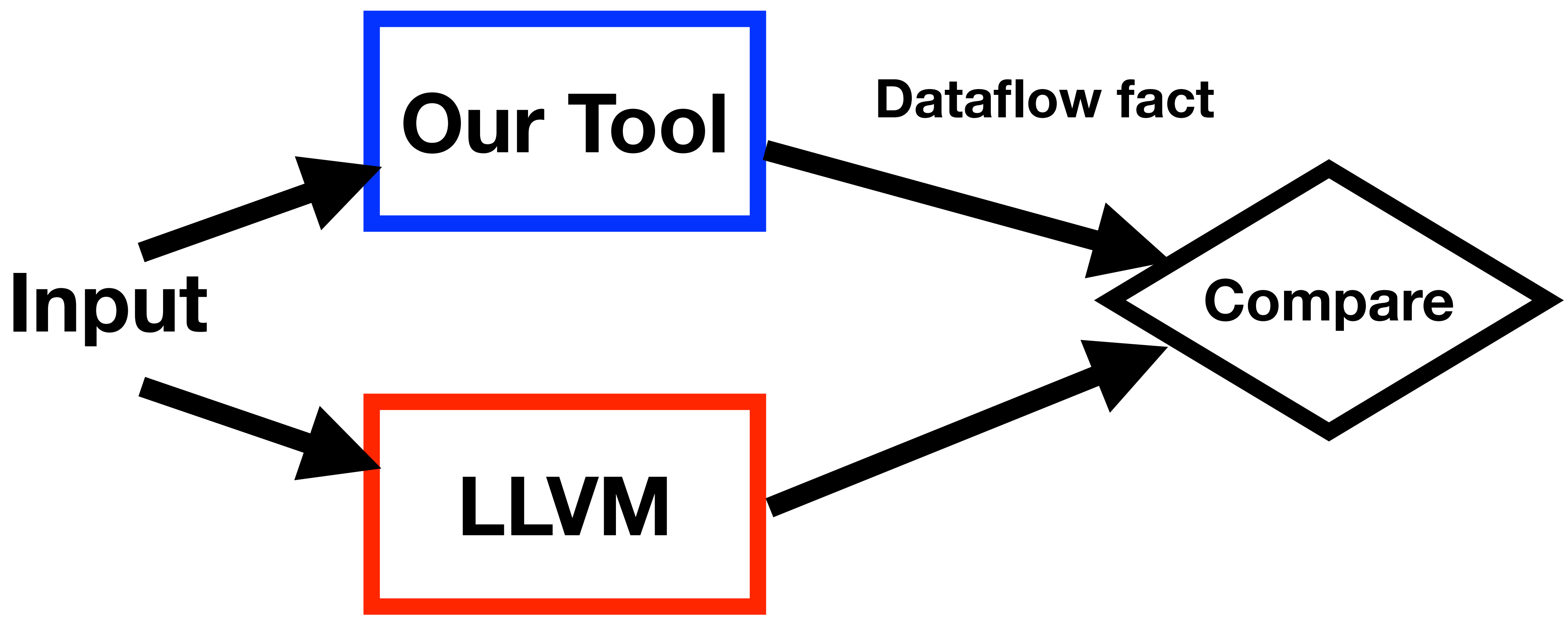
Best Result : 30

Problems

- **Imprecision**
- **Unsoundness**
- **Developers are manually improving the static analysis without any help of formal methods**

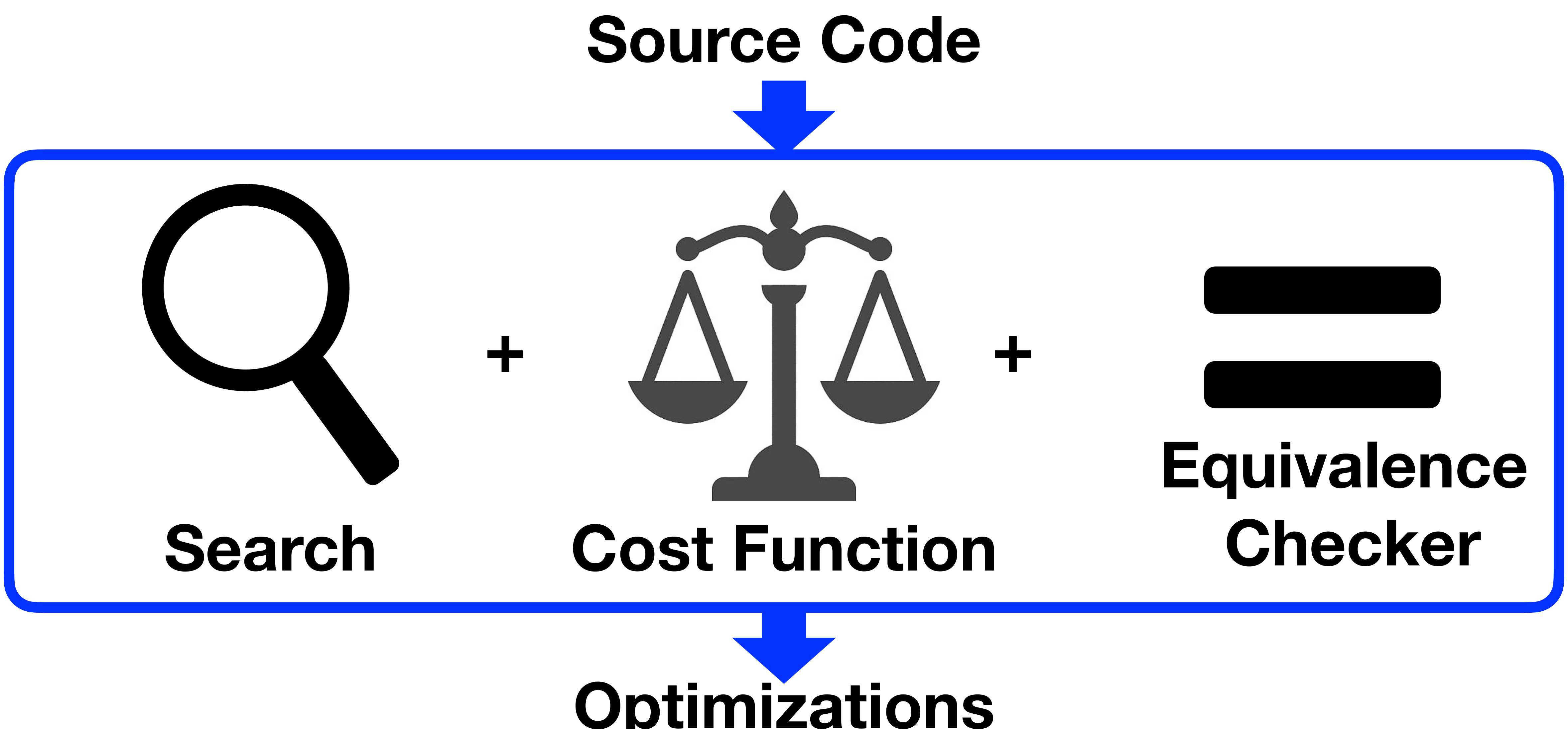
Goal

Automatic testing of LLVM's static analyses using formal methods

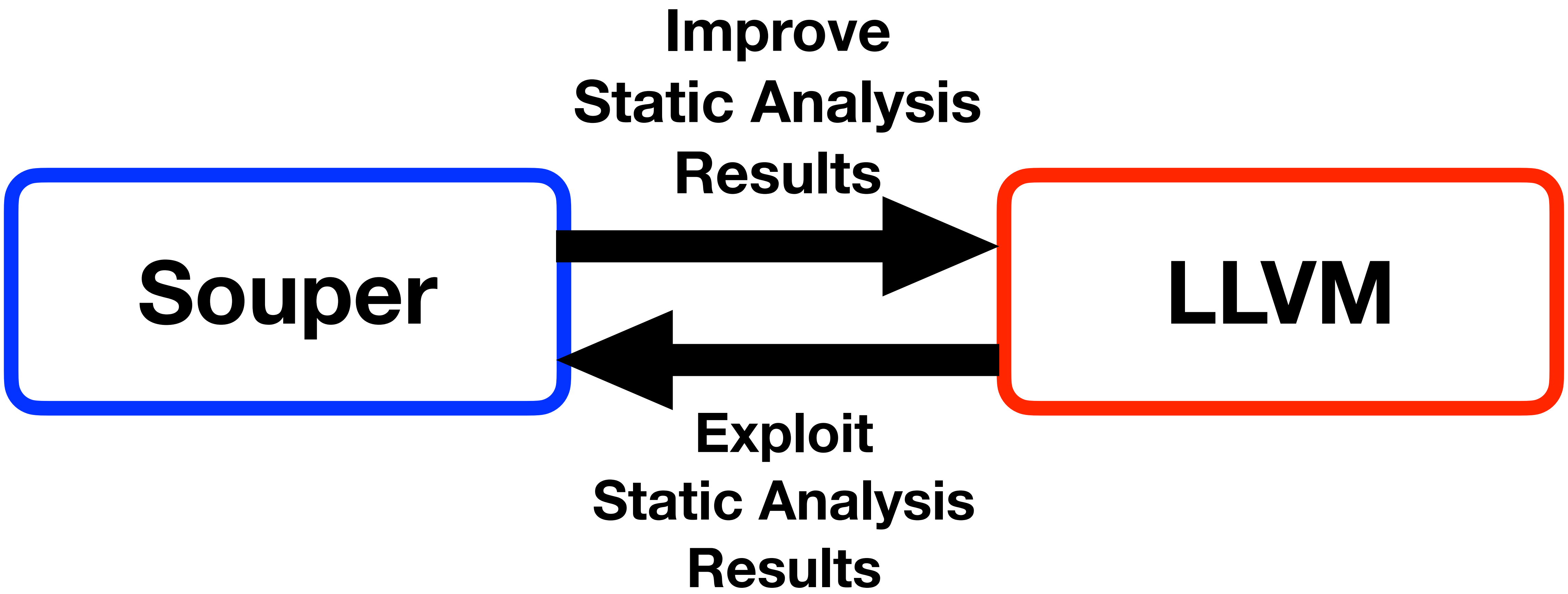


Souper

Automatically discover missing peephole optimizations



Static Analysis and Souper



- Known Bits
- Integer Range
- Number of Sign Bits
- Non-zero
- Non-negative
- Negative
- Power of Two

Forward

- Demanded Bits

Backward

Known Bits

```
%0 = shl i8 %x, 4
```

?	?	?	?	0	0	0	0

```
%0 = shl i8 4, %x
```

?	?	?	?	?	?	?	?

Solver-based Algorithm to Compute Known Bits for 4 << x

							0

						0	0

					?	0	0

more failing guesses …

?	?	?	?	?	?	0	0

```
%0 = shl i8 4, %x
```

LLVM:

?	?	?	?	?	?	?	?

Our Tool:

?	?	?	?	?	?	0	0

- Our algorithm uses at most **2 * BitWidth** solver calls
  - Brute force algorithm uses $3^{BitWidth}$ calls, which is infeasible
- Computes maximally precise known bits as the lattice is separable at bit level
  - Details in the paper

Integer Range

```
define i4 @foo(i4 %y) {
entry:
  %0 = mul i4 %y, %y
  ret i4 %0
}
```

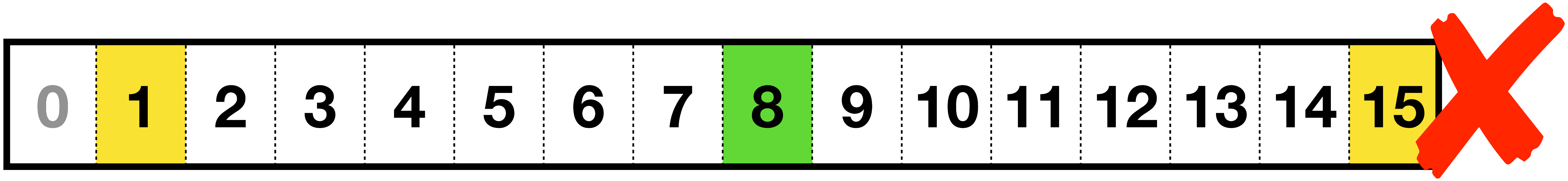
Range of %0?

LLVM: No information

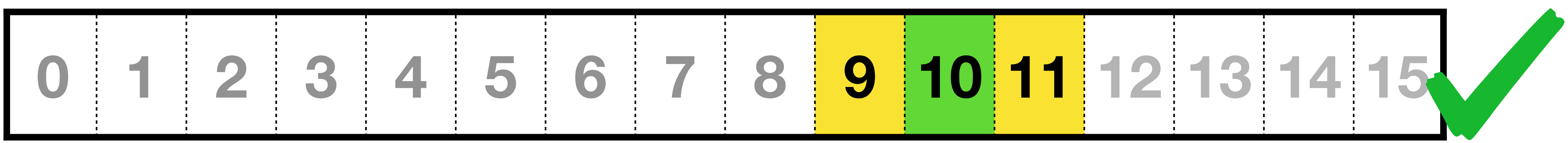
Range [X, X+M)

- **Algorithm for M - Binary Search to find the smallest M**
- **Algorithm for X - Solver-based Constant Synthesis**

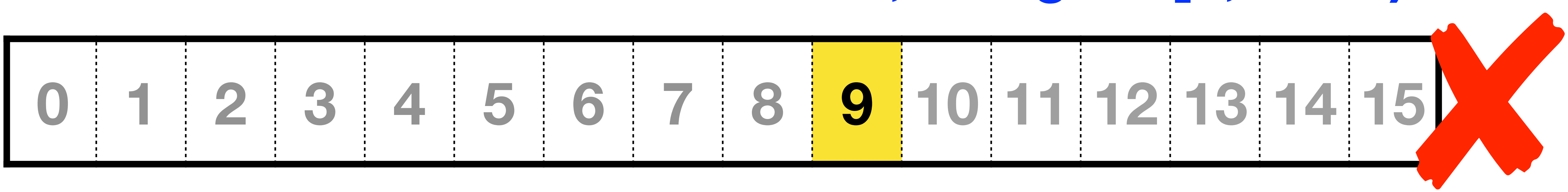
Find X such that (y * y) ∈ [X, X+M)



X = 0, Range = [0, 0+12)



X = 0, Range = [0, 0+10)



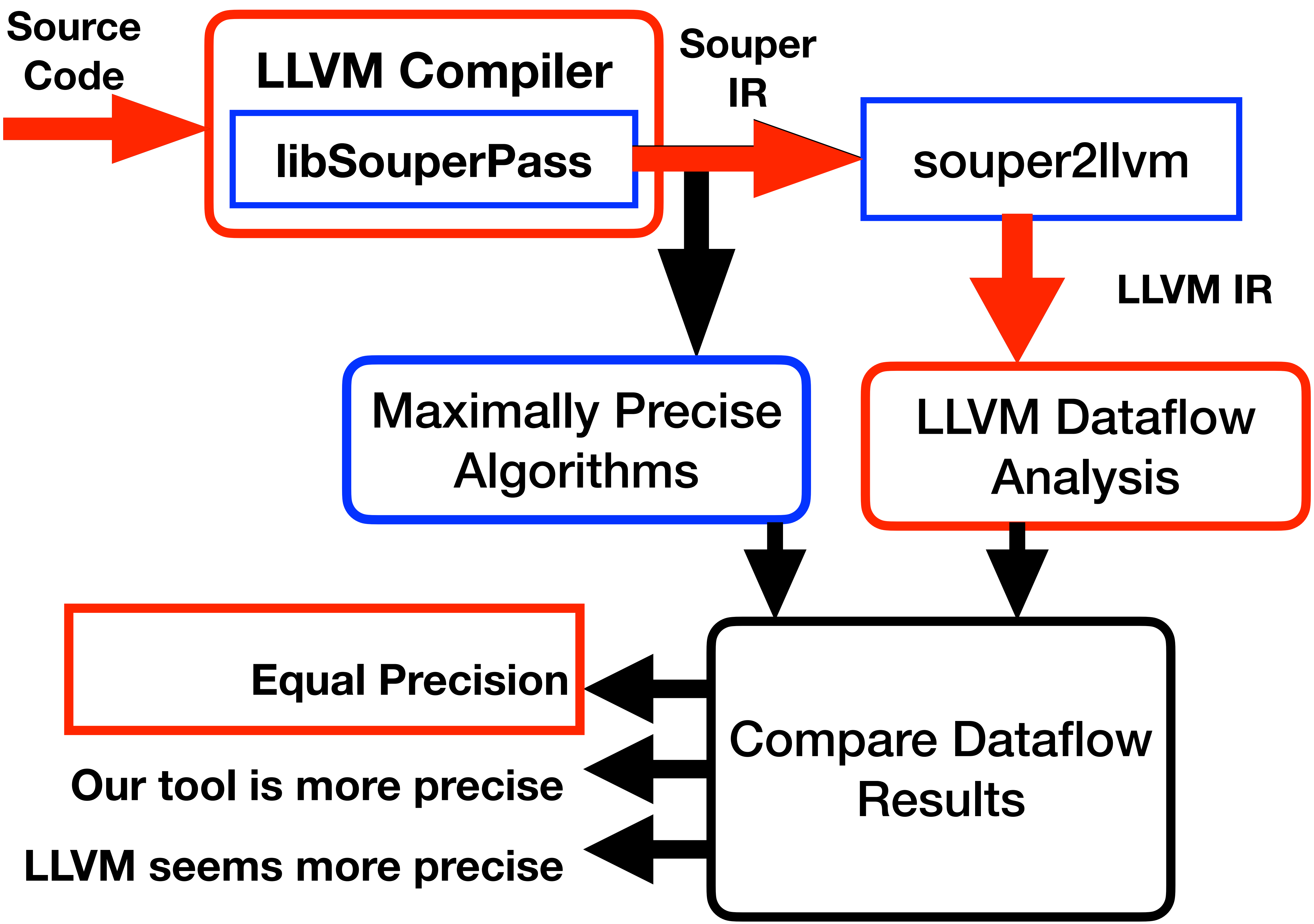
```
%0 = mul i4 %y, %y
```

LLVM: No Information

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Our Tool: [0, 10)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



Source Code
LLVM Compiler
libSouperPass
Souper IR
souper2llvm
LLVM IR
Maximally Precise Algorithms
LLVM Dataflow Analysis
Equal Precision
Our tool is more precise
LLVM seems more precise
Compare Dataflow Results

Impact on LLVM

- **Five concrete precision improvements made in LLVM's static analyses (< version 8) already discussed in the paper.**
- **All integer range imprecisions discussed in the paper have already been fixed in LLVM 10.**
- **More known bits imprecisions have also been fixed in code generation phase.**

What happens if LLVM calls our analyses instead of its own?

Too Slow!

Is LLVM Unsound?

- **No new soundness bugs were found in LLVM+Clang-8.0**
- **Introduced three old soundness bugs from LLVM-2.9+ and our tool detected all of them**

Conclusion

- **Solver-based algorithms to compute maximally precise dataflow results to find imprecisions and unsoundness issues**

- **Encourage compiler developers to use formal methods based techniques to test static analyses**

Backup Slides
Constant Synthesis

- We use SMT Solvers to compute a constant X that satisfies the constraint.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

0 * 0 = 0

1 * 1 = 1

2 * 2 = 4

3*3 = 9

4*4 = 16 = 0

5 * 5 = 25 = 9

6 *6 = 36 = 4

7 * 7 = 49 = 1

8 * 8 = 64 = 0

9 * 9 = 81 = 1

10 * 10 = 100 = 4

11 * 11 = 121 = 9

12 * 12 = 144 = 0

13 * 13 = 169 = 9

14 * 14 = 196 = 4

15 * 15 = 225 = 1

Constant Synthesis

- Use SMT solver to compute a constant C that satisfies the given constraint.

Compute Integer Range for (y + y) & 1

[x, x+M)

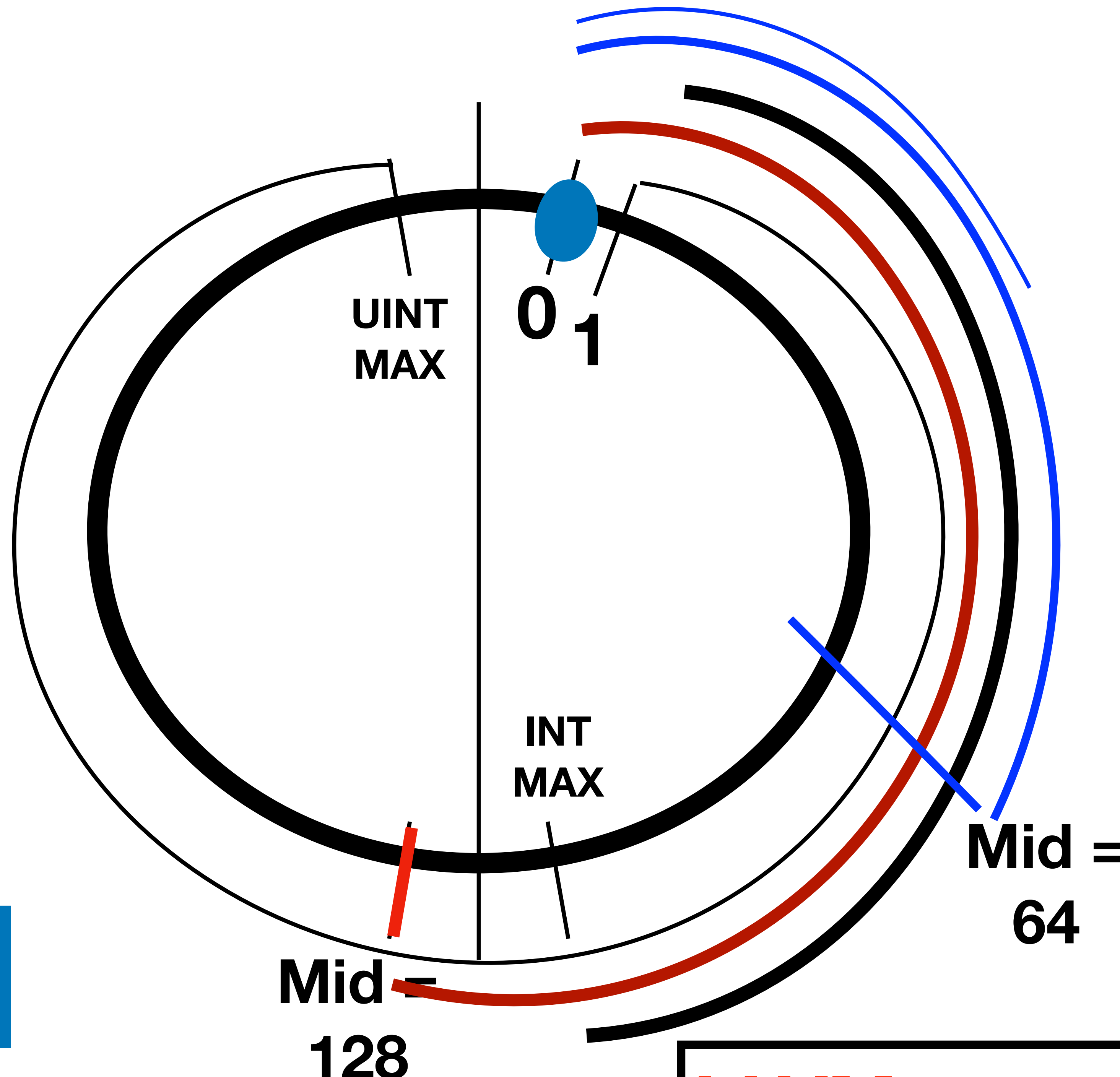
[0, 0+128)

[0, 0+64)

. . .

[0, 0+2)

[0, 0+1)



LLVM: [0, 2)

Precise Tool: [0, 1)