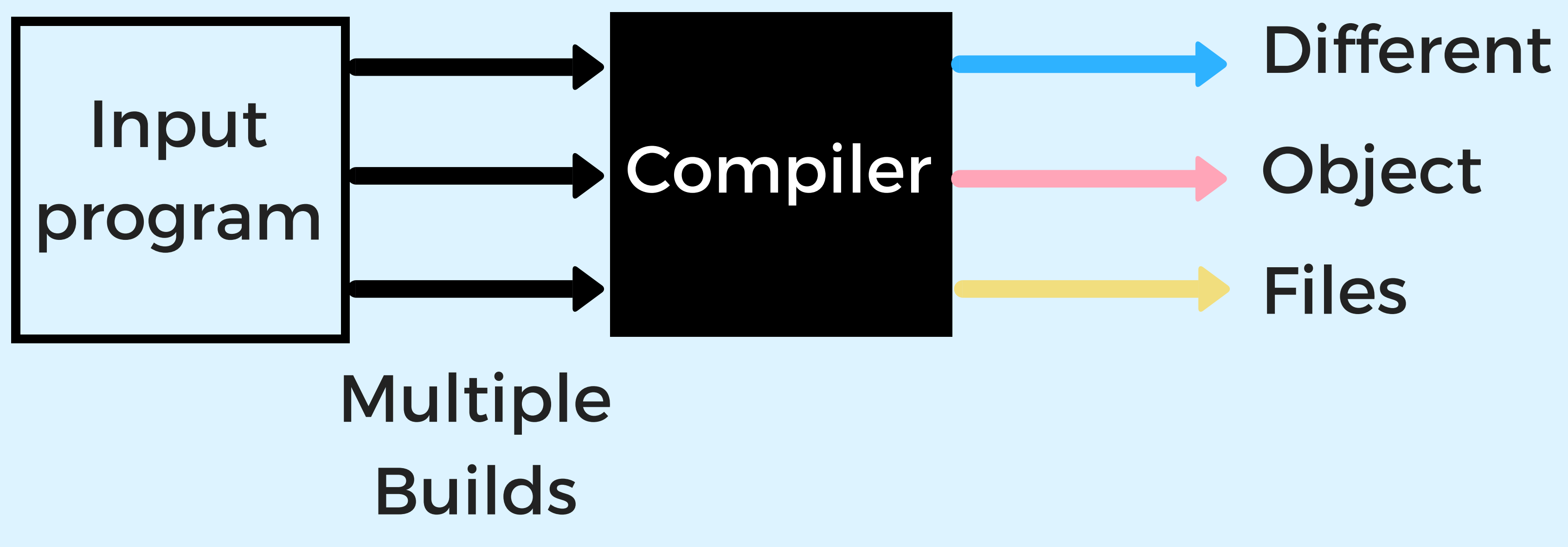
FIGHTING NON-DETERMINISM IN C++ COMPILERS

Mandeep Singh Grang • Qualcomm Innovation Center, Inc., San Diego, CA

A compiler is non-deterministic if its output differs from run to run on the same input

1. THE PROBLEM

C++ compilers may exhibit non-deterministic behavior.



2. WHERE IS IT OBSERVED?

i. Back-to-back runs of the same compiler.

ii. The same compiler hosted on different operating systems.

iii. Asserts vs non-asserts versions of the same compiler.

3. WHY IS IT A PROBLEM?

- Hard-to-reproduce bugs
- Difficult debugging
- Runtime crashes
- Unpredictable performance

"If you can't reproduce a bug, you can't fix it."

– Anonymous Programmer

4. WHAT CAUSES THIS NON-DETERMINISM?

i Iteration of unordered containers

```
std::unordered_set<int> S ( {-1, 0, 1} );

for (auto I : S)
  cout << I;
```

Iteration order of the unordered_set depends on hash values of elements

```
Output on Ubuntu Linux x86: 1 -1 0
Output on Windows x86:      -1 0 1
(compiled with LLVM 6.0)
```

ii Hashing of pointer keys

```
int x = -1, y = 0, z = 1;
std::map<int *, int> M;

M[&x] = x;
M[&y] = y;
M[&z] = z;

for (auto &I : M)
  cout << I.second;
```

Keys of the map are addresses which may change from run to run

```
Output on Ubuntu Linux x86: 1 -1 0
Output on Windows x86:      1 0 -1
(compiled with LLVM 6.0)
```

iii Use of non-stable sort functions

```
using IntPair = std::pair<int, int>;
std::vector<IntPair> V = {{0, 1}, {0, 2}};

std::sort(V.begin(), V.end(),
 [] (IntPair x, IntPair y) {
   return x.first < y.first; });

for (auto &I : V)
  cout << I.second;
```

Relative ordering of keys with same values is non-deterministic

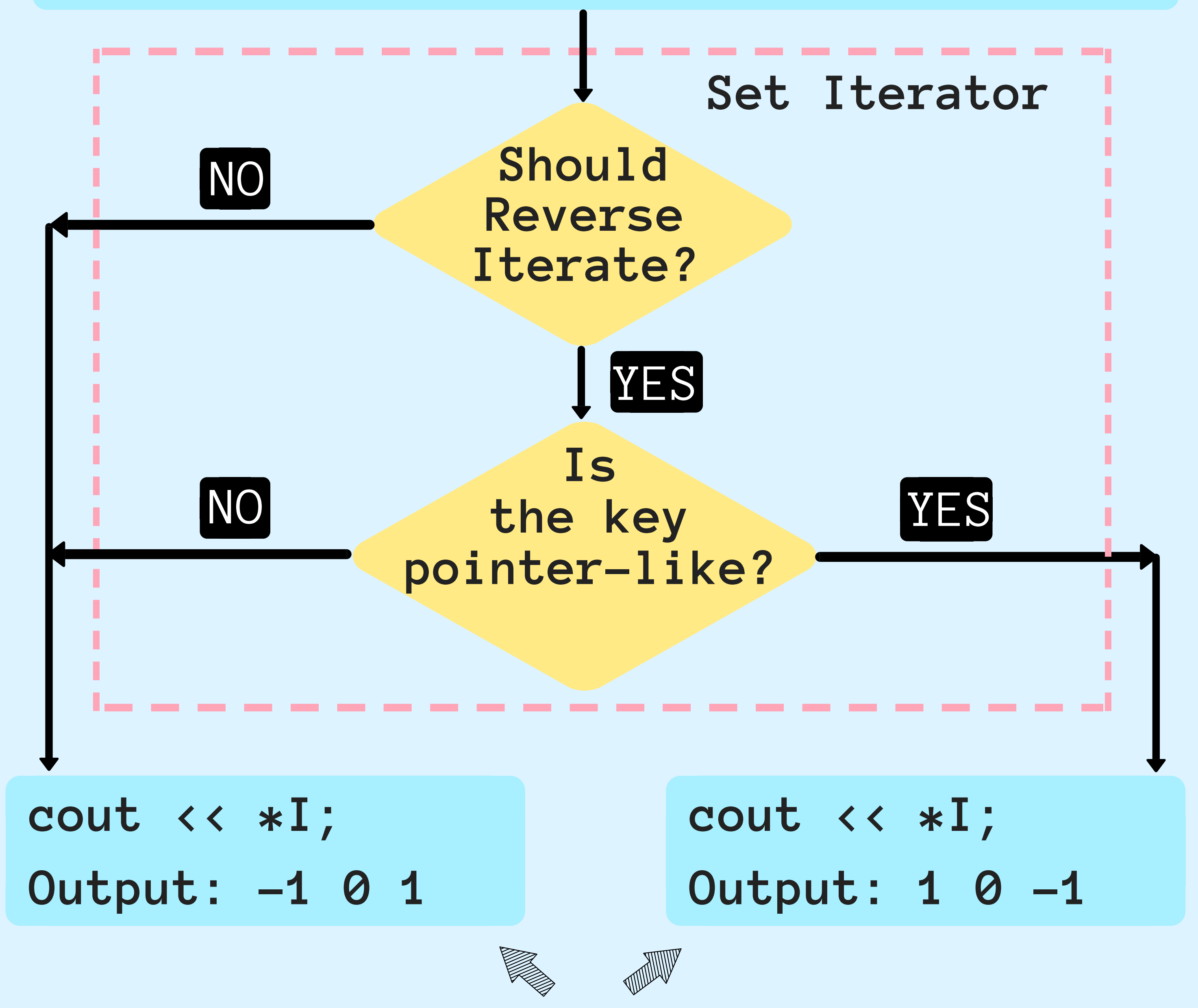
```
Possible outputs: 1 2 or 2 1
```

A Case Study : Non-determinism in the LLVM Compiler

5. HOW DOES LLVM DETECT NON-DETERMINISTIC ITERATION ORDER?

- Added a reverse iteration mode for unordered containers.
- Compare the output of reverse iteration mode with forward iteration mode.
- Bugs uncovered and fixed: 42

```
int x = -1, y = 0, z = 1;
SmallPtrSet<int *, 2> S ({&x, &y, &z});
for (auto &I : S) {
```



```
cout << *I;
Output: -1 0 1
```

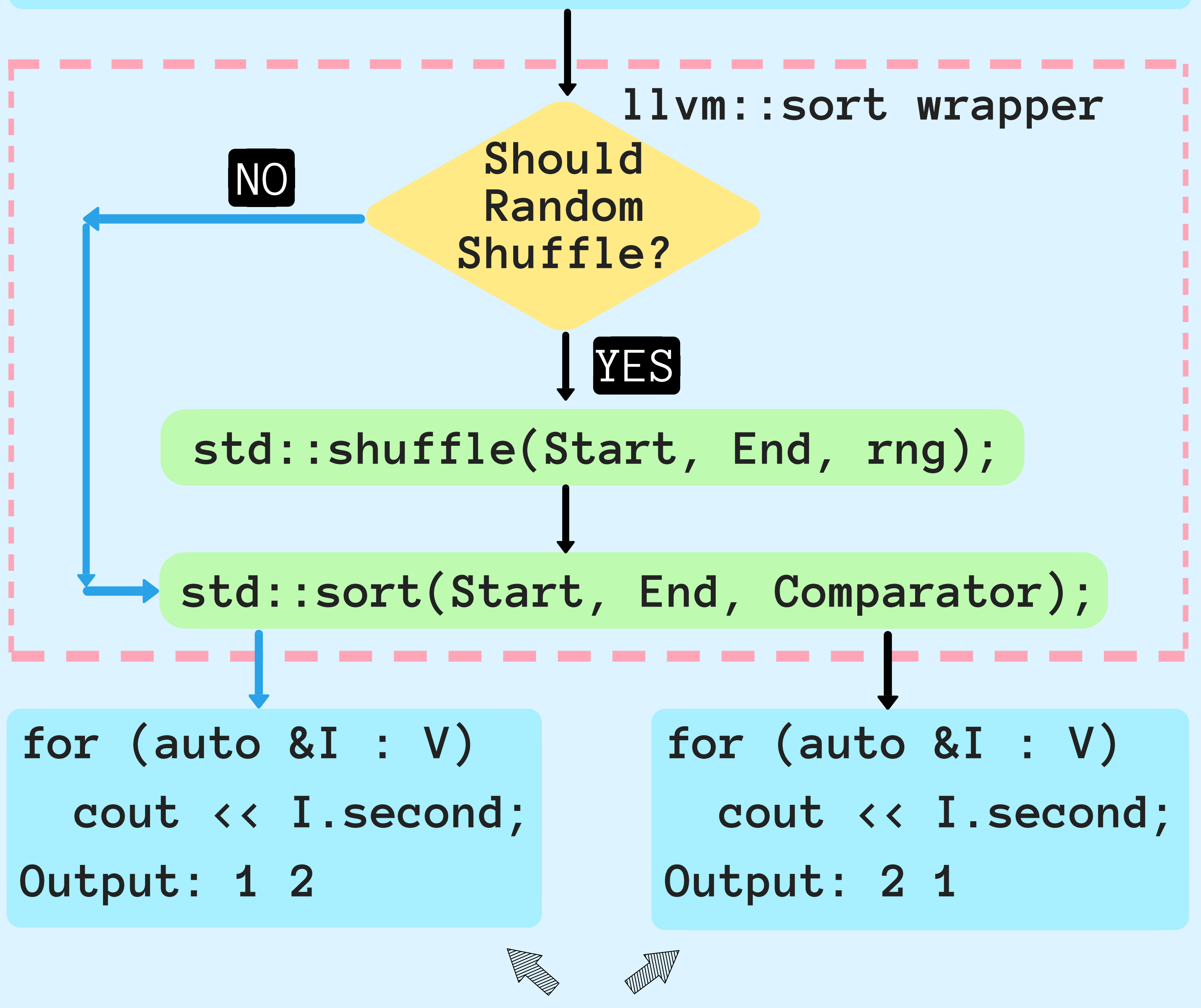
```
cout << *I;
Output: 1 0 -1
```

Output sensitive to iteration order.
Possible non-determinism!

6. HOW DOES LLVM DETECT NON-DETERMINISTIC SORTING ORDER?

- Added a wrapper llvm::sort which randomly shuffles a container before invoking std::sort.
- Uncovers non-deterministic sorting of keys with same values.
- Bugs uncovered and fixed: 44

```
std::vector<IntPair> V = {{0, 1}, {0, 2}};
llvm::sort(V.begin(), V.end());
```



```
std::shuffle(Start, End, rng);
std::sort(Start, End, Comparator);
```

```
for (auto &I : V)
  cout << I.second;
Output: 1 2
```

```
for (auto &I : V)
  cout << I.second;
Output: 2 1
```

Different sorting order.
Possible non-determinism!

7. HOW DO YOU FIX/AVOID NON-DETERMINISM?

i Use an ordered container

```
std::vector<T> V;
for (auto I : V)
```

ii Sort the container before iteration

```
std::sort(V.begin(), V.end());
for (auto I : V)
```

iii Use a stronger sort predicate

```
std::sort(V.begin(), V.end(),
  [] (T a, T b) { return
  a.first  < b.first &&
  a.second < b.second; });
```

iv Use a stable sort function

```
std::stable_sort(V.begin(), V.end());
```

8. WHERE CAN I LEARN MORE?

https://github.com/mgrang/non-determinism

CppCon 2018, Bellevue, WA