Dagstuhl Seminar on
Secure Compilation

Taming Undefined Behavior in LLVM

Seoul National Univ.

Juneyoung Lee
Yoonseung Kim
Youngju Song
Chung-Kil Hur

Azul Systems

Sanjoy Das

Google

David Majnemer

University of Utah

John Regehr

Microsoft Research

Nuno P. Lopes

What this talk is about

- A compiler IR (Intermediate Representation) can be designed to allow more optimizations by supporting "undefined behaviors (UBs)"
- LLVM IR's UB model
  - Complicated
  - Invalidates some textbook optimizations
- Our new UB model
  - Simpler
  - Can validate textbook optimizations (and more)

Undefined Behavior (UB) & Problems

Motivation for UB

Peephole Optimization

```
int* p
int  a
int  b
```

IR

```
output(p + a > p + b)
```

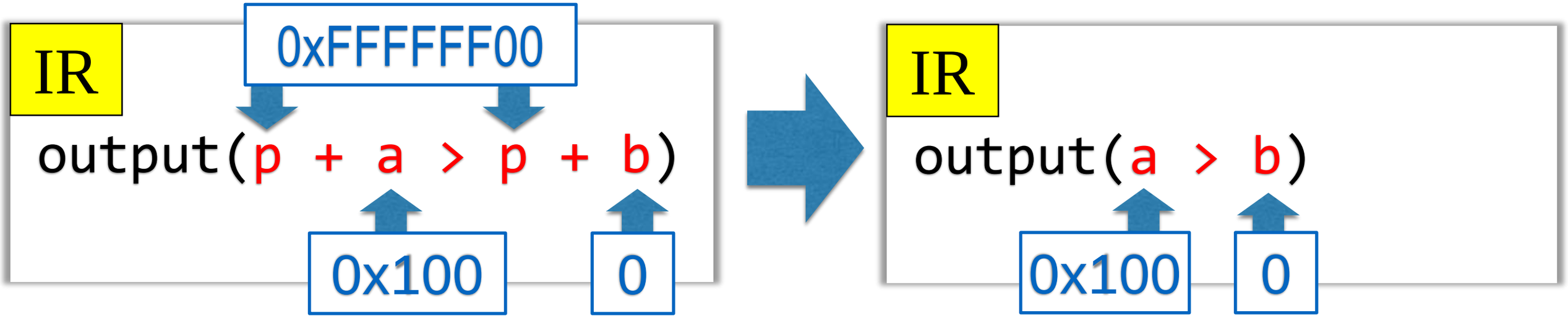
IR

```
output(a > b)
```

Motivation for UB

Peephole Optimization

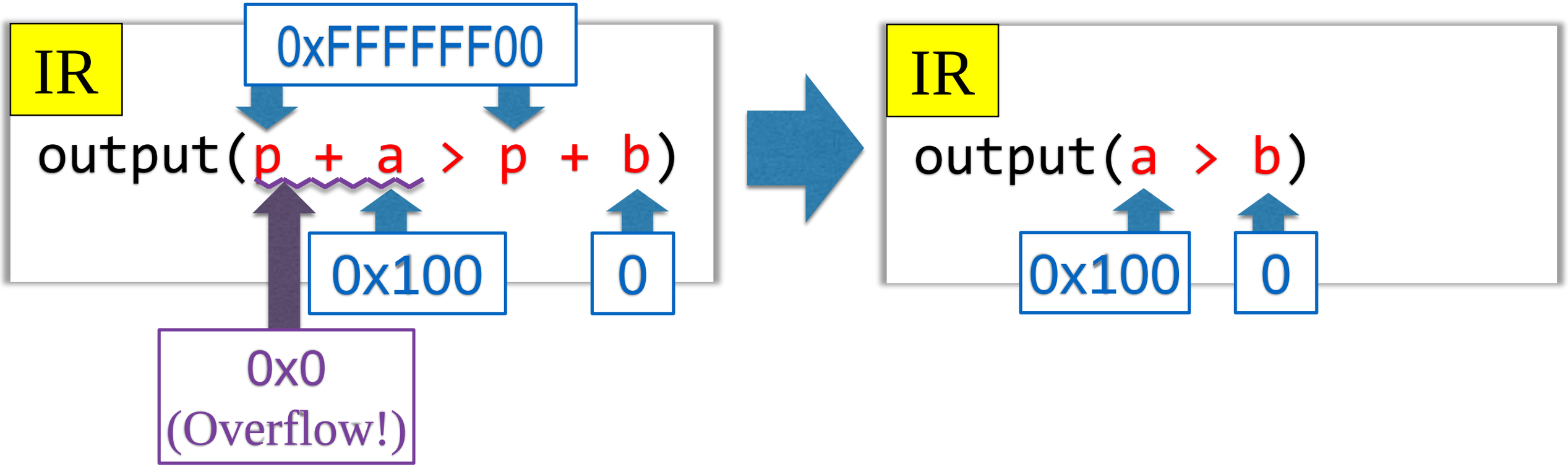
```
int* p
int  a
int  b
```



Motivation for UB

Peephole Optimization

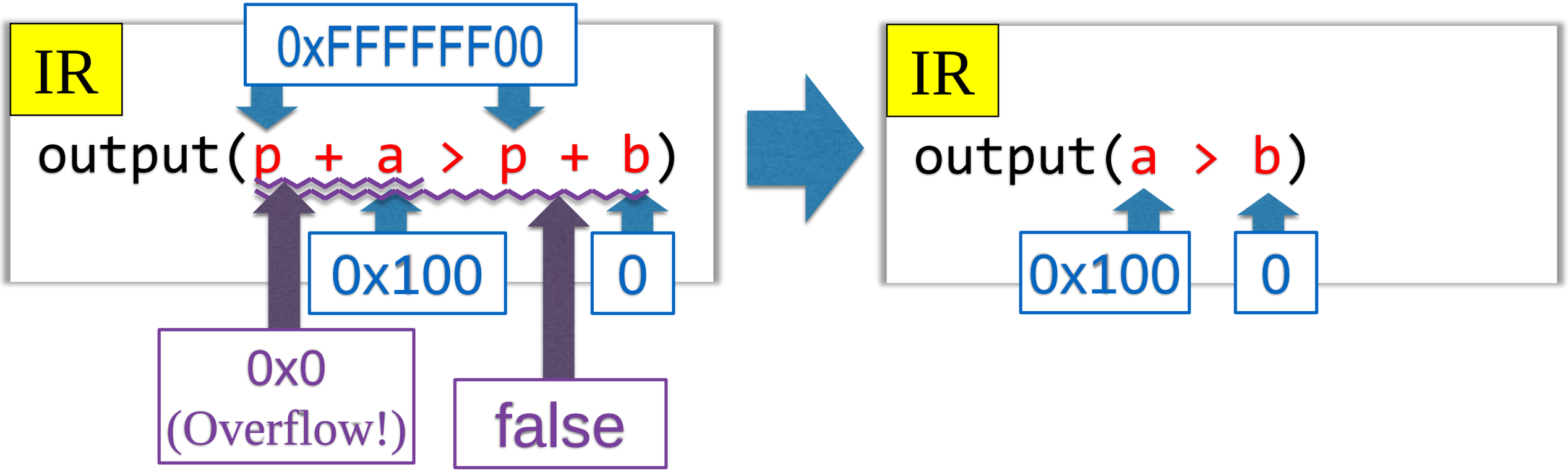
```
int* p
int  a
int  b
```



Motivation for UB

Peephole Optimization

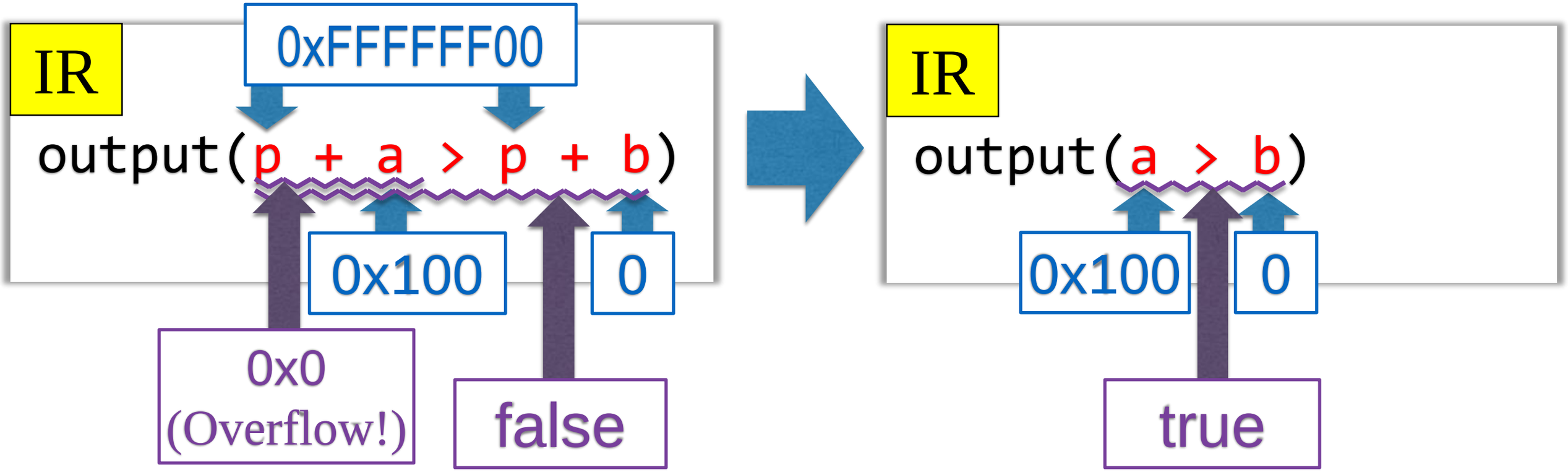
```
int* p
int  a
int  b
```



Motivation for UB

Peephole Optimization

```
int* p
int  a
int  b
```



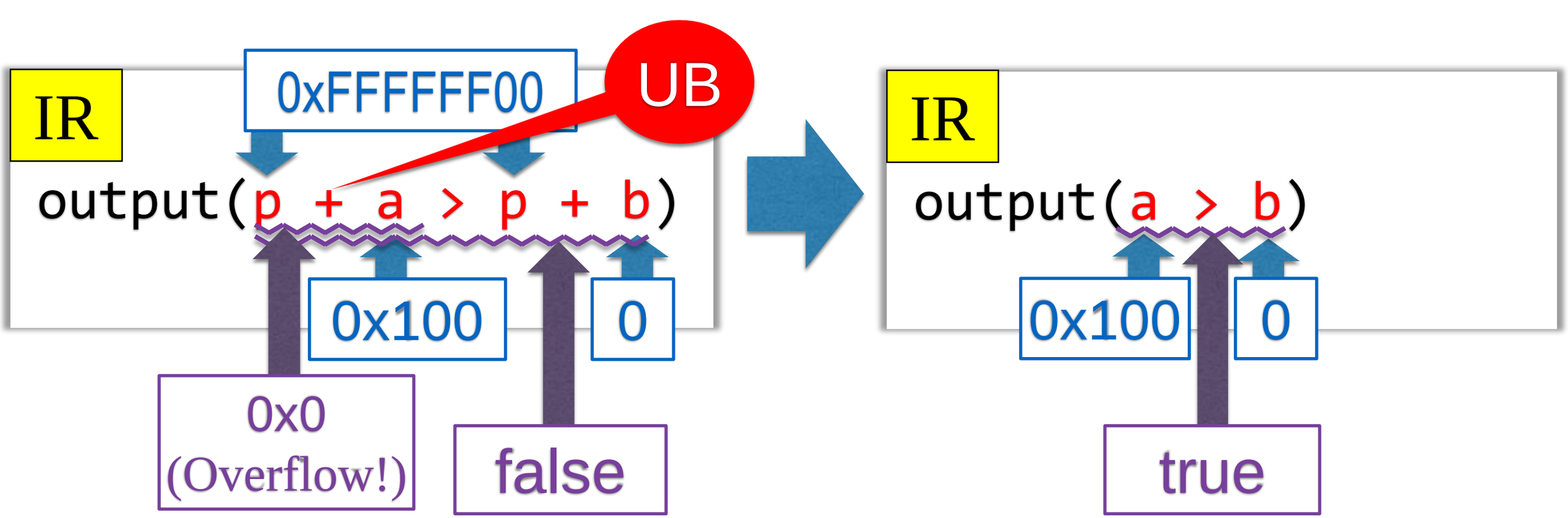
Motivation for UB

Peephole Optimization

Simple UB Model:

Pointer Arithmetic Overflow is

Undefined Behavior



Problems with UB

Loop Invariant Code Motion

Simple UB Model:

Pointer Arithmetic Overflow is

Undefined Behavior

IR

```
...
for(i=0; i<n; ++i)
{
  a[i] = p + 0x100
}
```

IR

```
q = p + 0x100
for(i=0; i<n; ++i)
{
  a[i] = q
}
```

Problems with UB

Loop Invariant Code Motion

Simple UB Model:

Pointer Arithmetic Overflow is

Undefined Behavior

IR

```
...
for(i=0; i<n; ++i)
{
  a[i] = p + 0x100
}
```

0

0xFFFFFF00

IR

```
q = p + 0x100
for(i=0; i<n; ++i)
{
  a[i] = q
}
```

0xFFFFFF00

0

Problems with UB

Loop Invariant Code Motion

Simple UB Model:

Pointer Arithmetic Overflow is

Undefined Behavior

IR

```
...
for(i=0; i<n; ++i)
{
   a[i] = p + 0x100
}
```

0

0xFFFFFF00

IR

```
q = p + 0x100
for(i=0; i<n; ++i)
{
   a[i] = q
}
```

0xFFFFFF00

Overflow!

0

Problems with UB

Loop Invariant Code Motion

Simple UB Model:

Pointer Arithmetic Overflow is

Undefined Behavior

IR

```
...
for(i=0; i<n; ++i)
{
   a[i] = p + 0x100
}
```

0

0xFFFFFF00

0xFFFFFF00

Overflow!

UB

IR

```
q = p + 0x100
for(i=0; i<n; ++i)
{
   a[i] = q
}
```

0

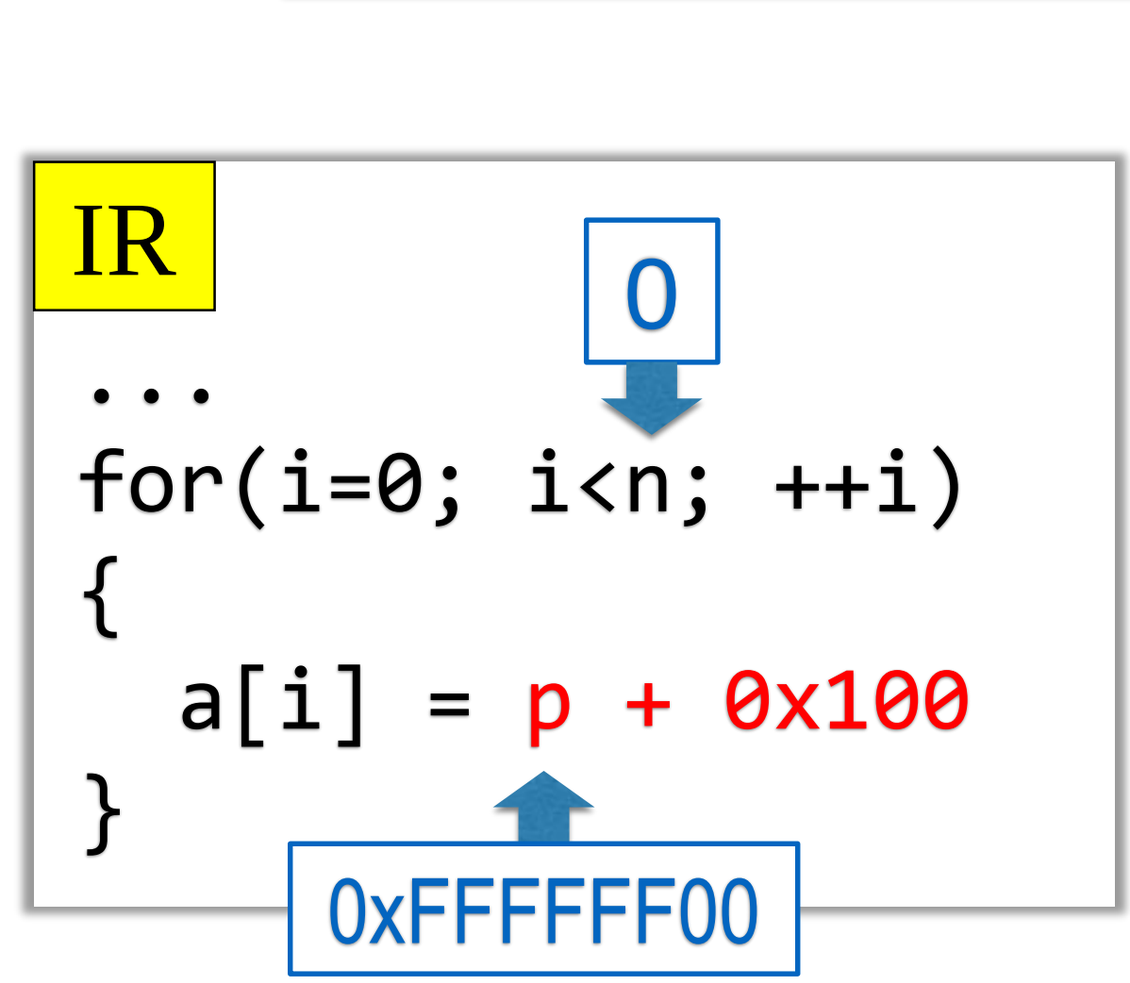
Existing Approaches

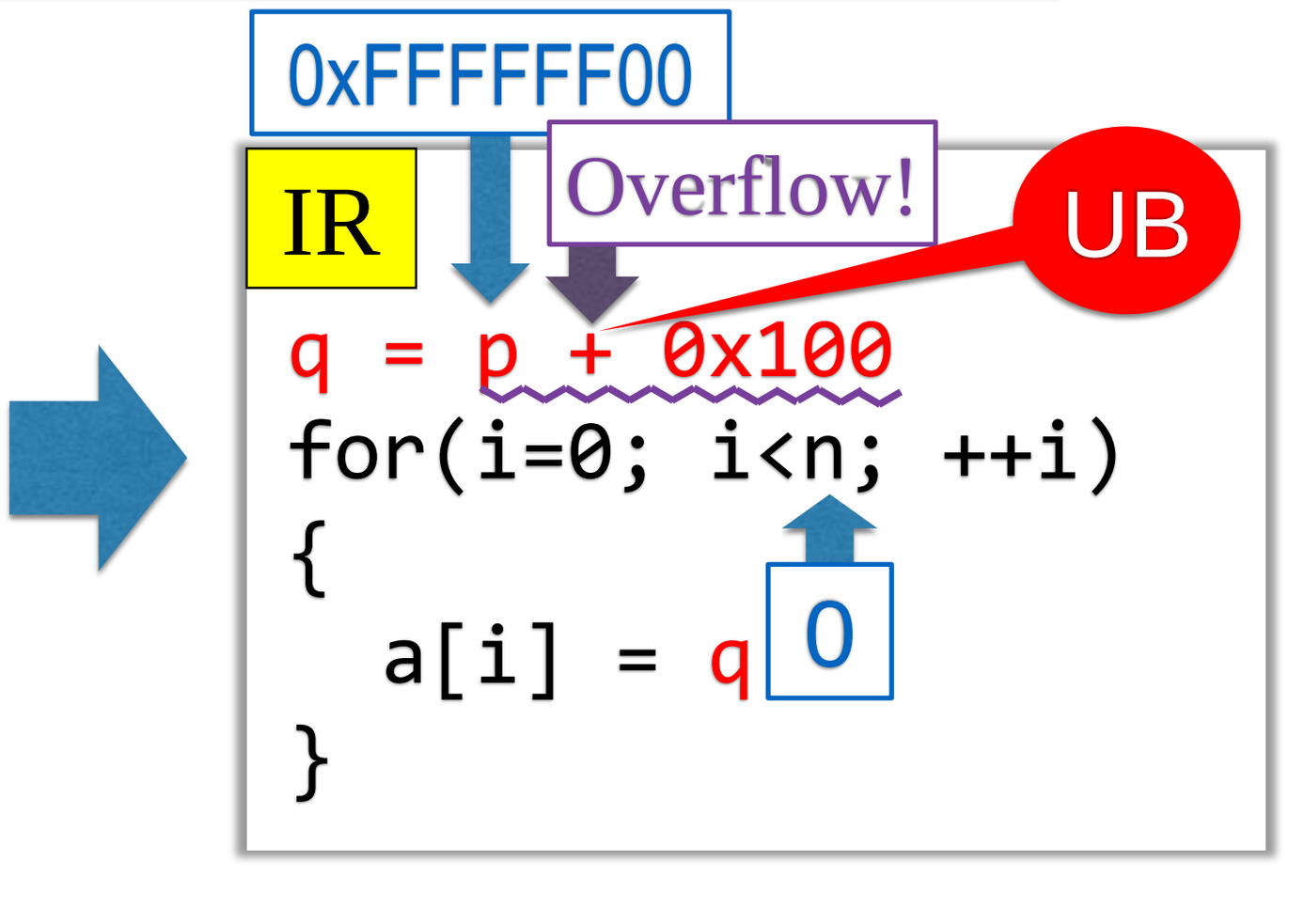
Poison Value: A Deferred UB

Simple UB Model:

Pointer Arithmetic Overflow is

Undefined Behavior



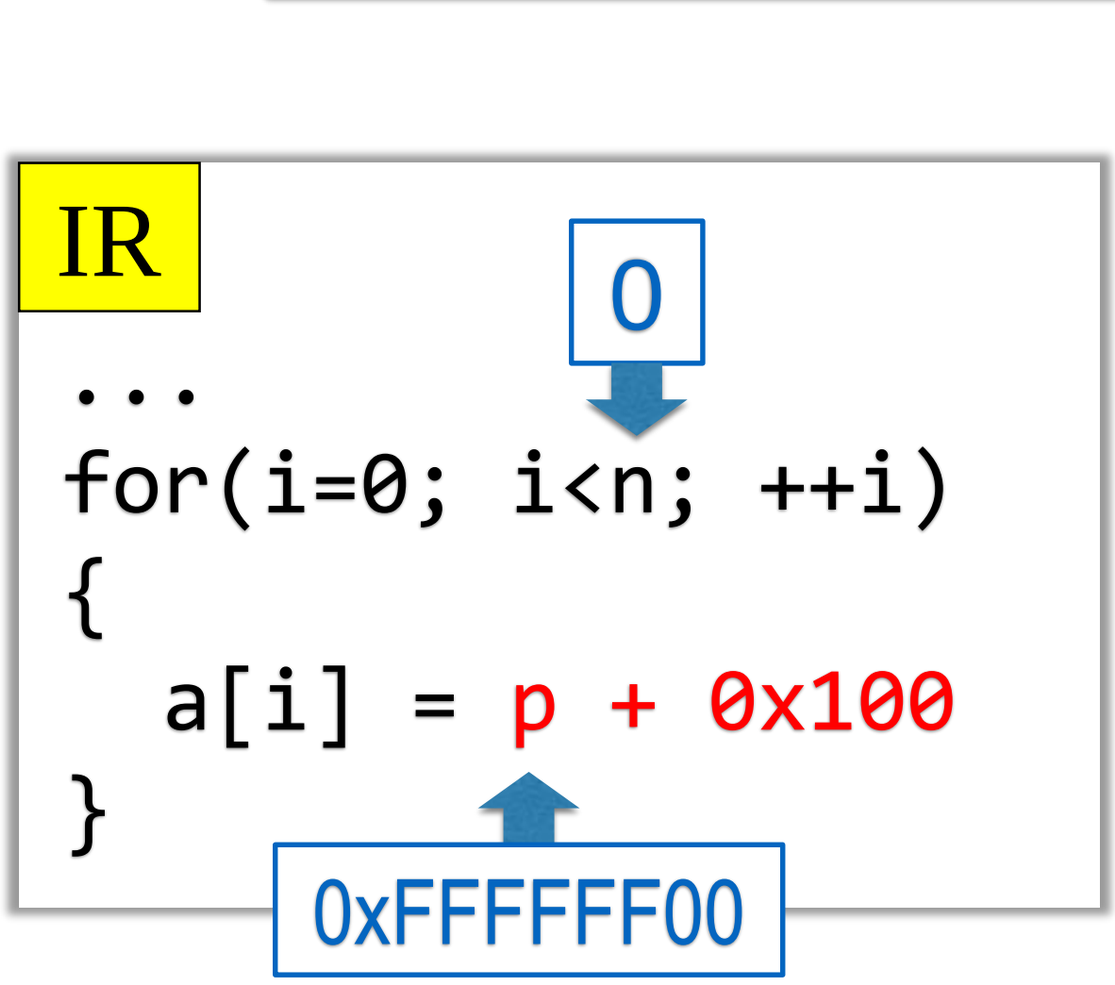


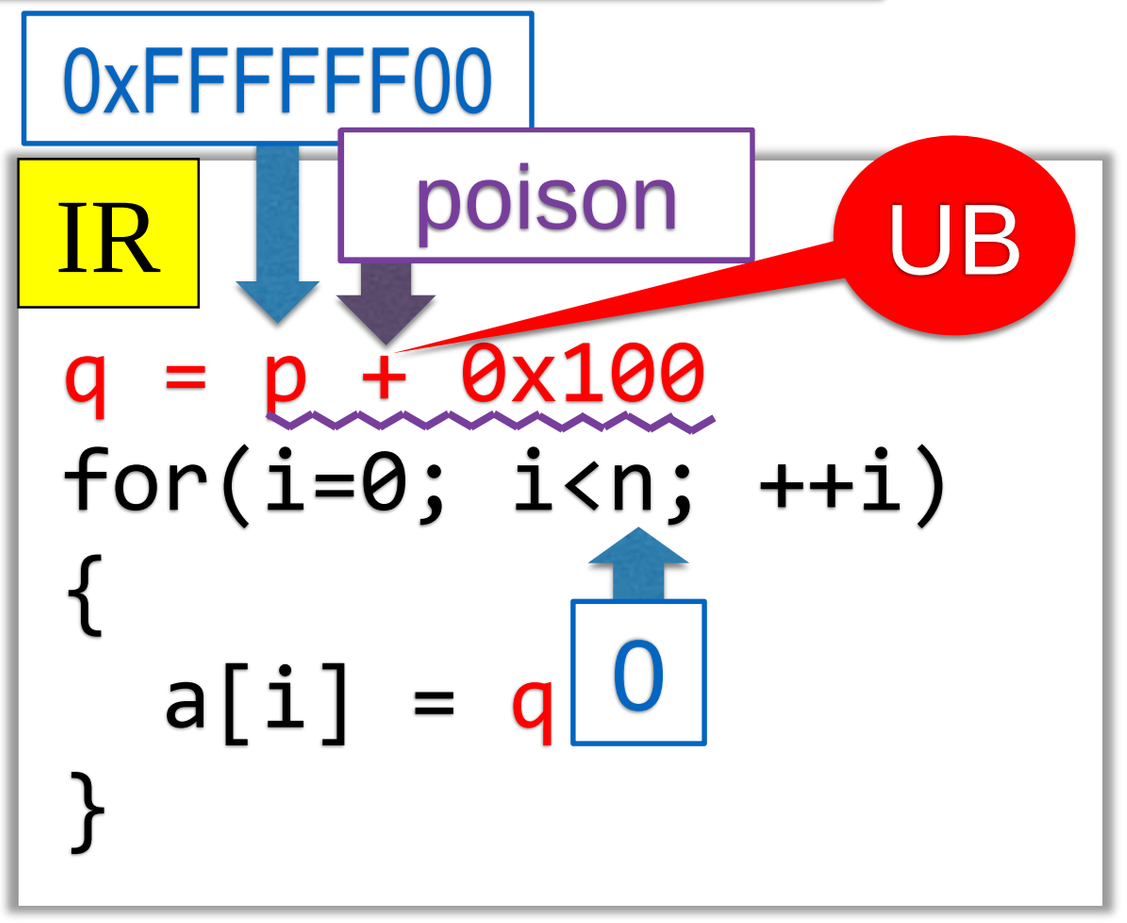
Poison Value: A Deferred UB

LLVM's UB Model:

Pointer Arithmetic Overflow is

A Poison "Value"





Poison Value: A Deferred UB

LLVM's UB Model:

Pointer Arithmetic Overflow is

A Poison "Value"

IR

```
...
for(i=0; i<n; ++i)
{
   a[i] = p + 0x100
}
```

0

0xFFFFFF00

IR

```
q = p + 0x100
for(i=0; i<n; ++i)
{
   a[i] = q
}
```

0xFFFFFF00

poison

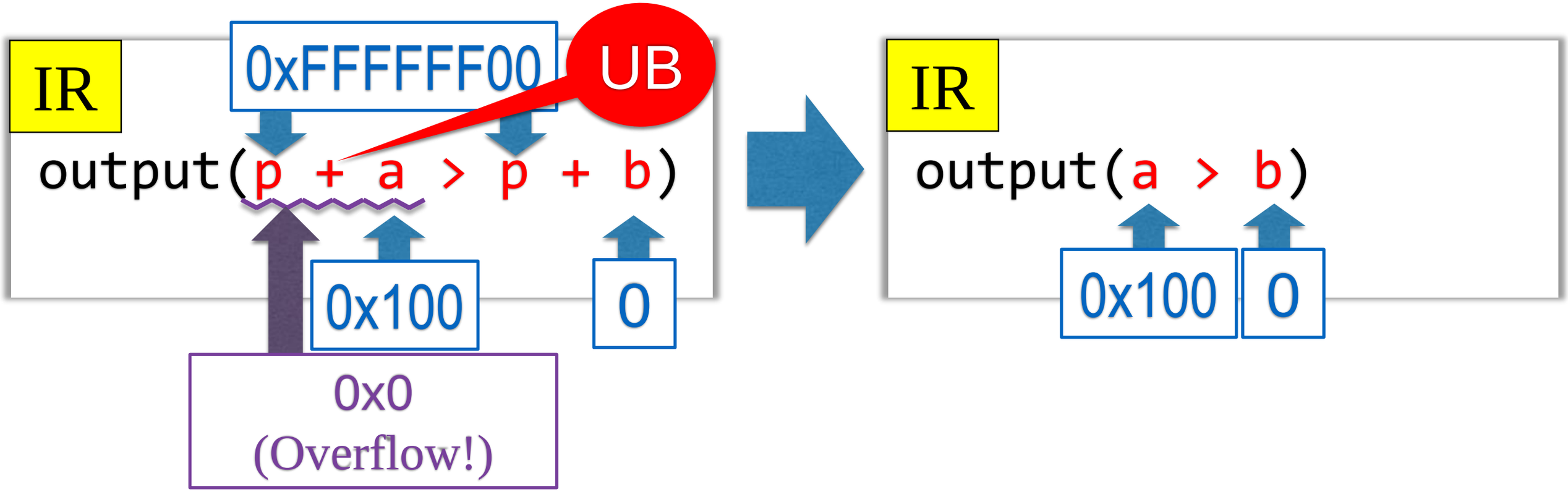
0

Poison Value: A Deferred UB

LLVM's UB Model:

Pointer Arithmetic Overflow is

A Poison "Value"

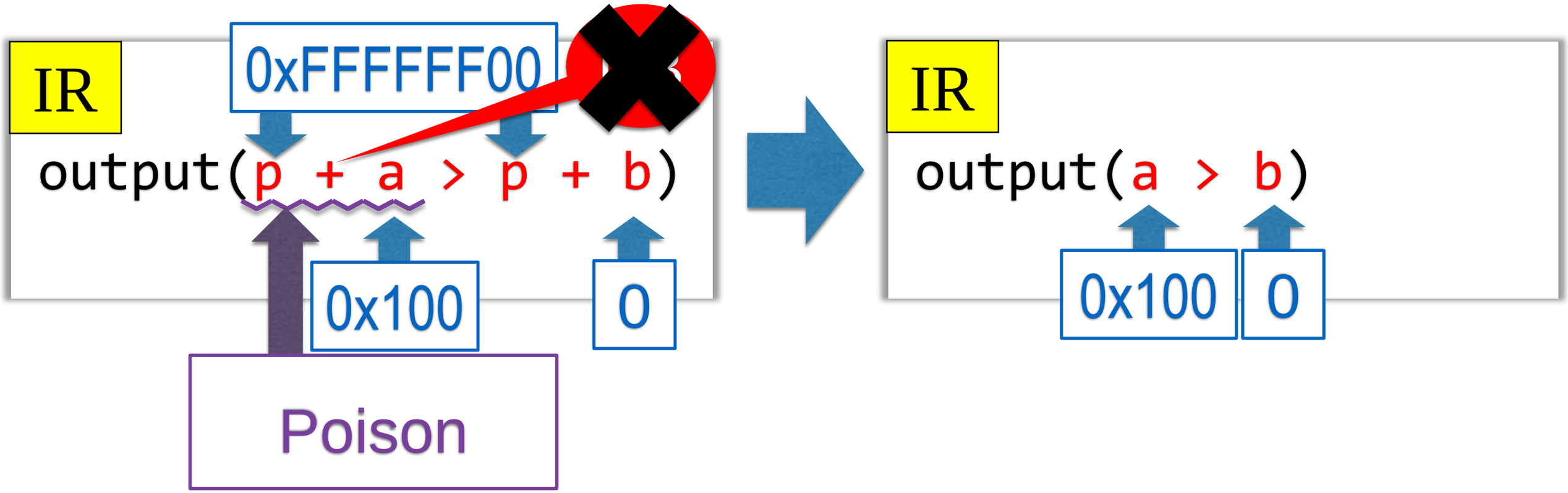


Poison Value: A Deferred UB

LLVM's UB Model:

Pointer Arithmetic Overflow is

A Poison "Value"

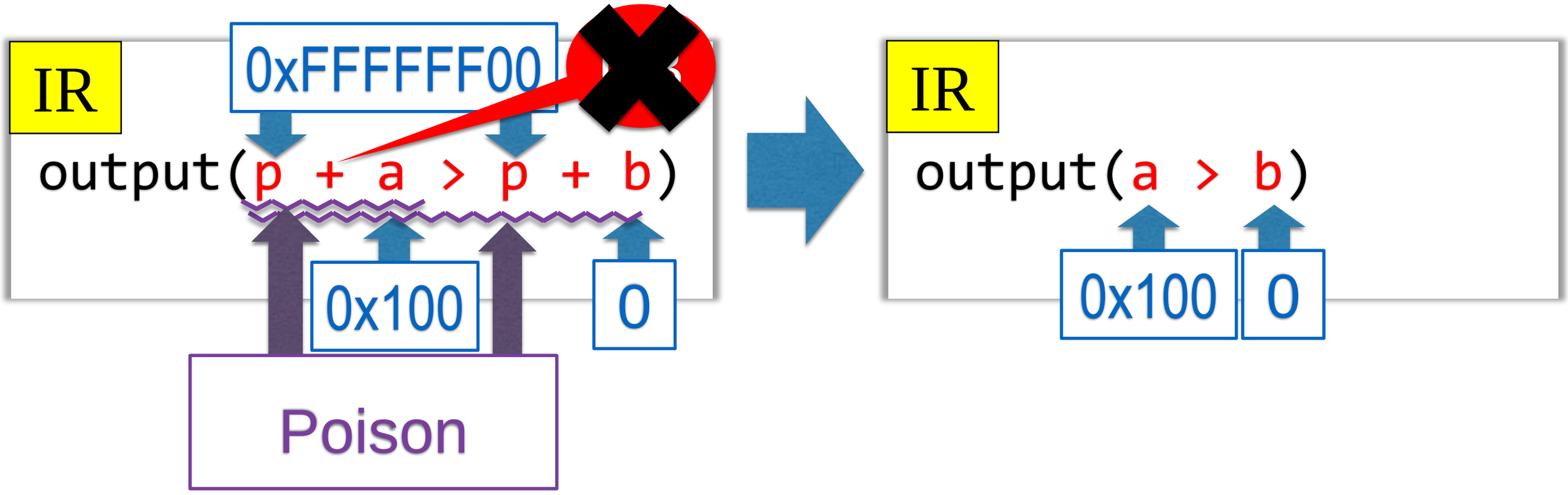


Poison Value: A Deferred UB

LLVM's UB Model:

Pointer Arithmetic Overflow is

A Poison "Value"

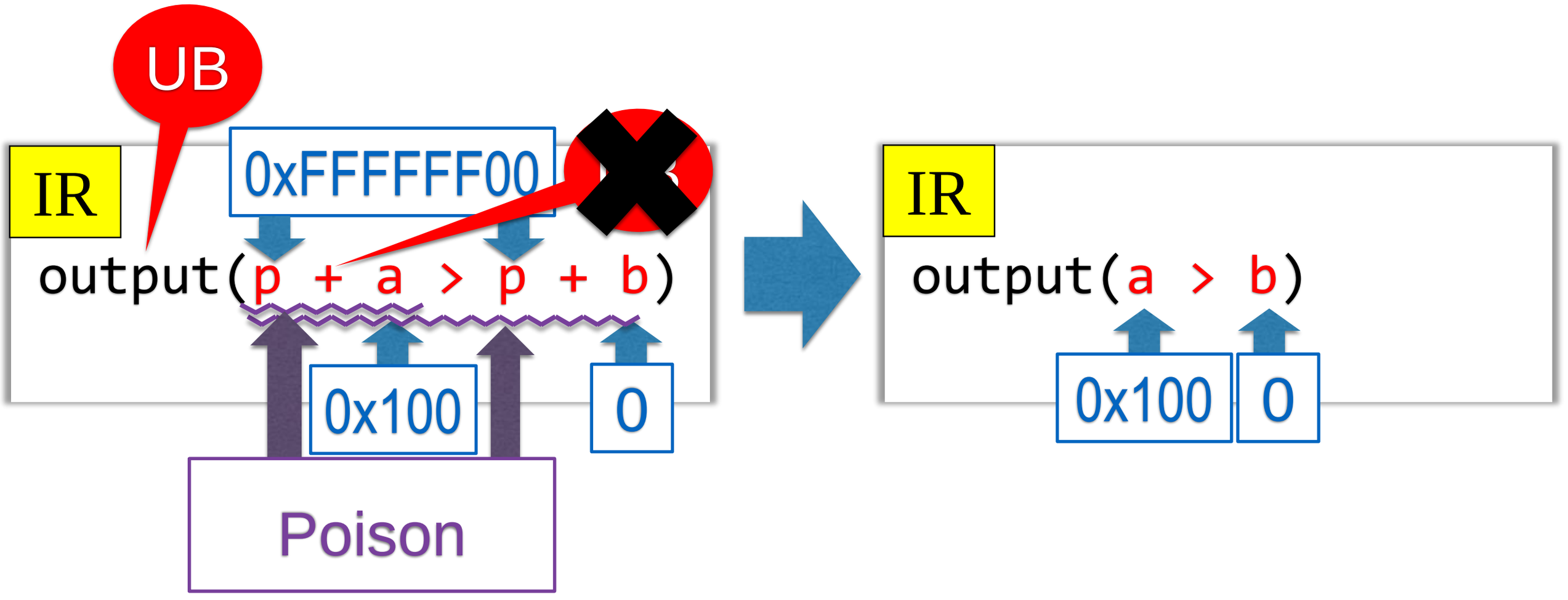


Poison Value: A Deferred UB

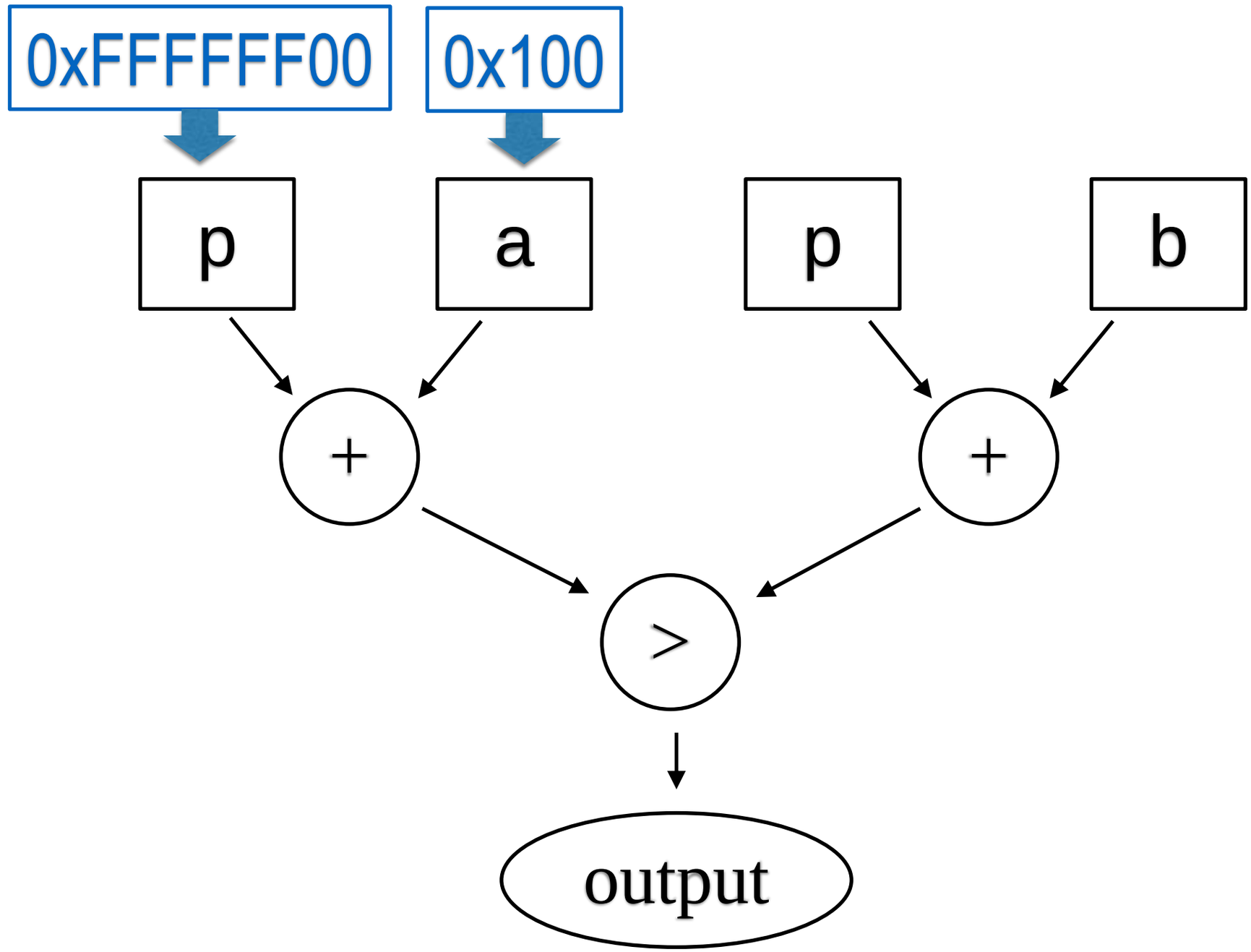
LLVM's UB Model:

Pointer Arithmetic Overflow is

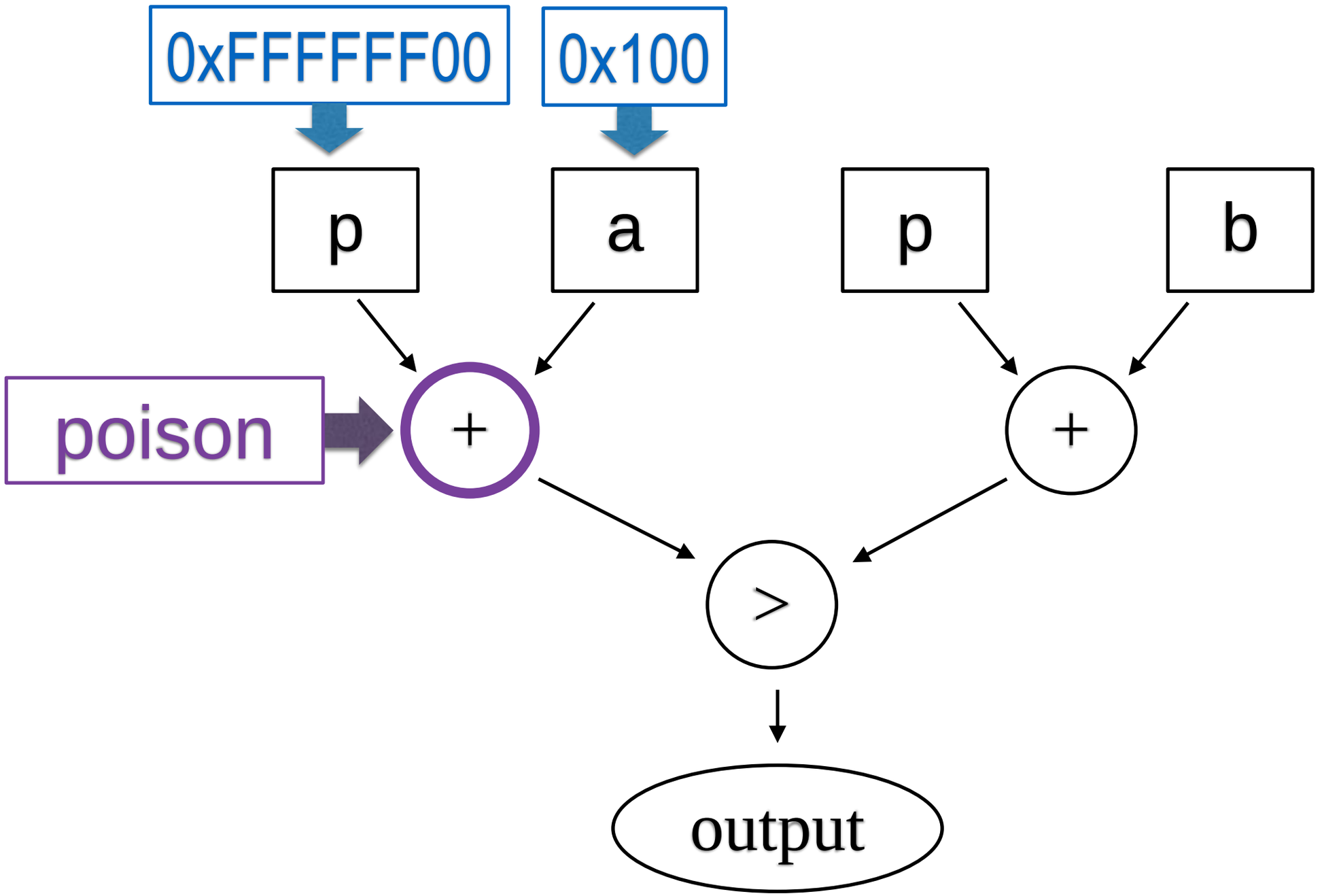
A Poison "Value"



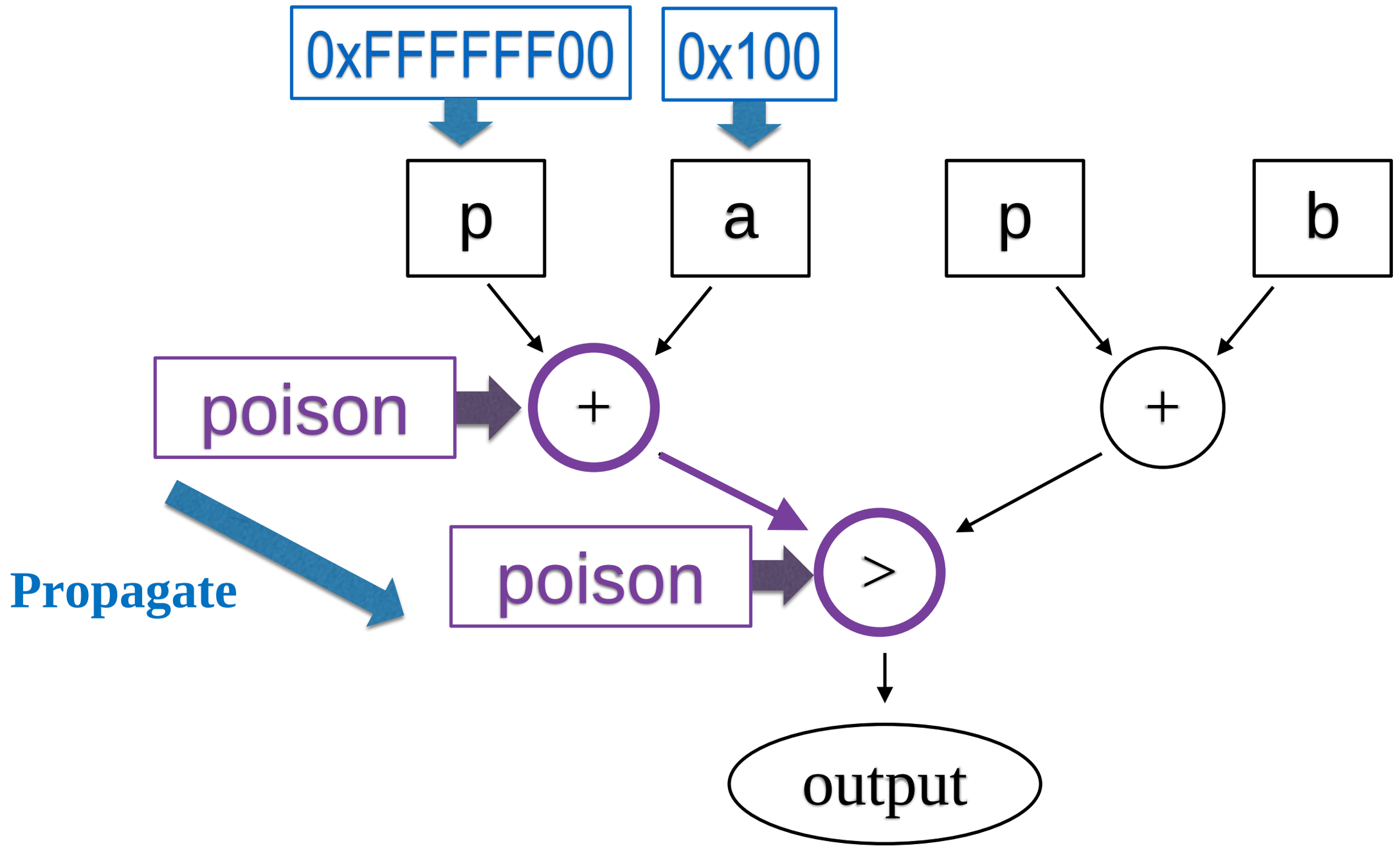
Summary of Poison



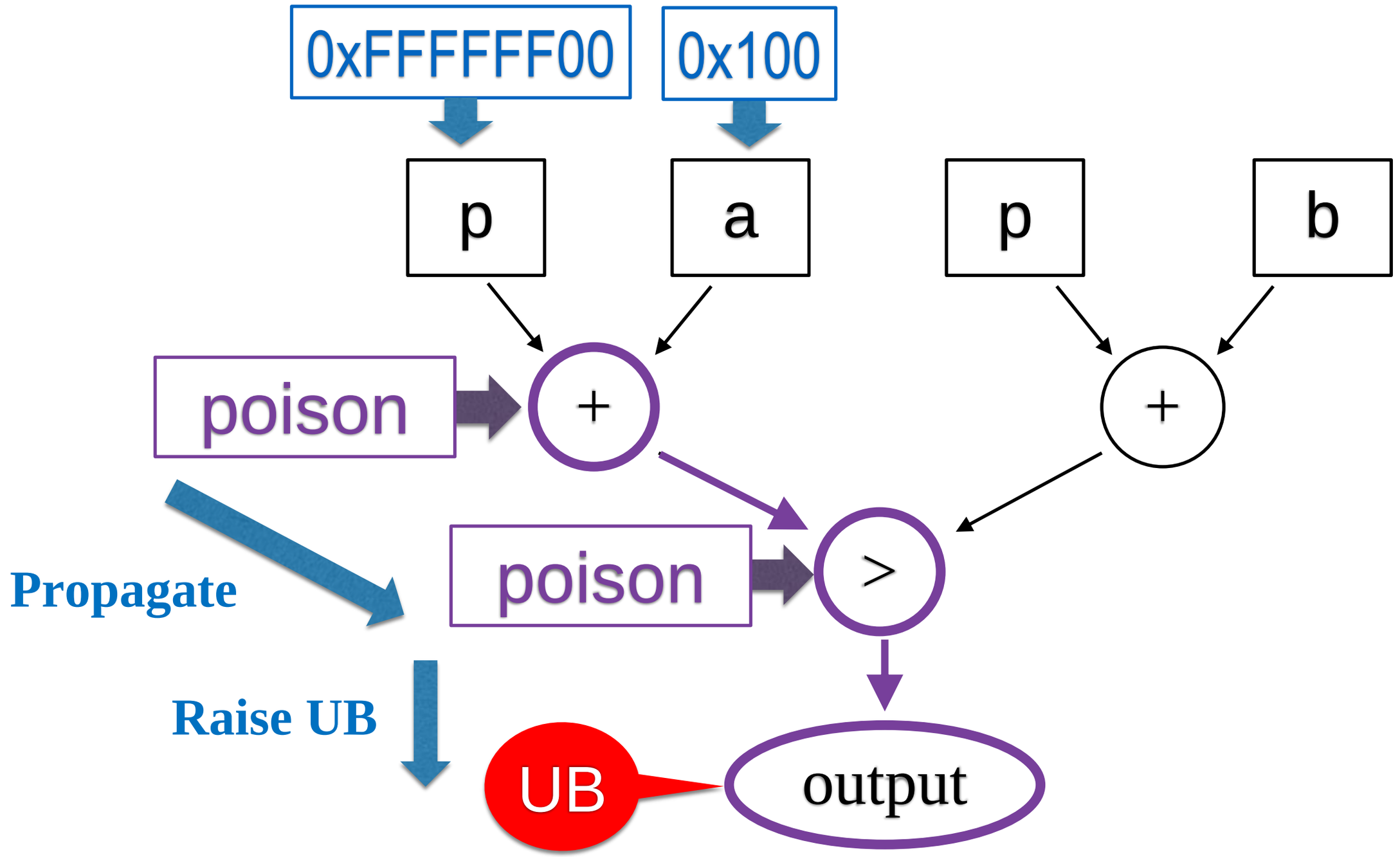
Summary of Poison



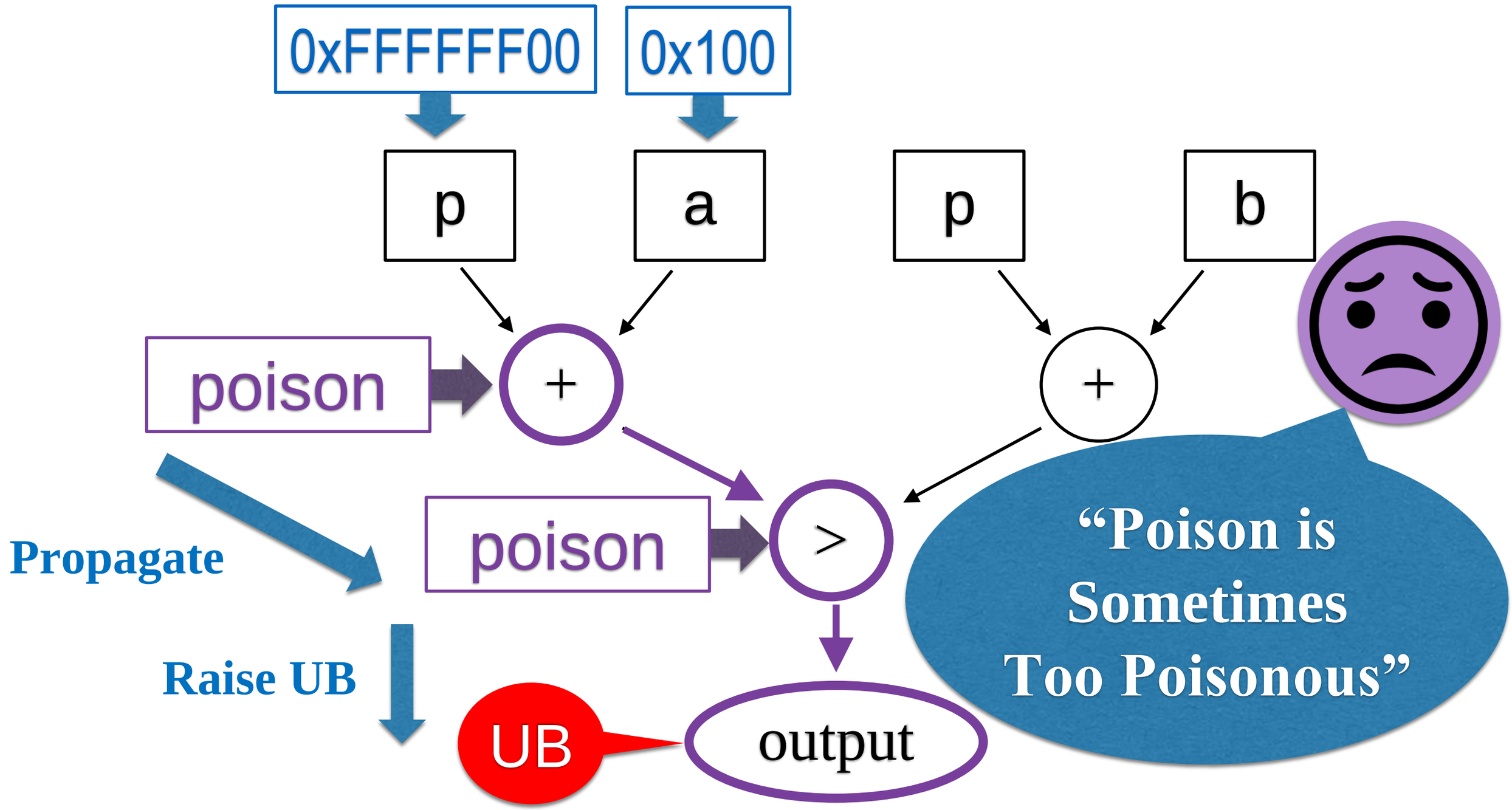
Summary of Poison



Summary of Poison



Summary of Poison



Problem

Problems with LLVM's UB

Global Value Numbering (GVN)

LLVM's UB Model:

Branching on poison is

???

```
if (x == y) {

   .. use x ..

}
```

```
if (x == y) {

   .. use y ..

}
```

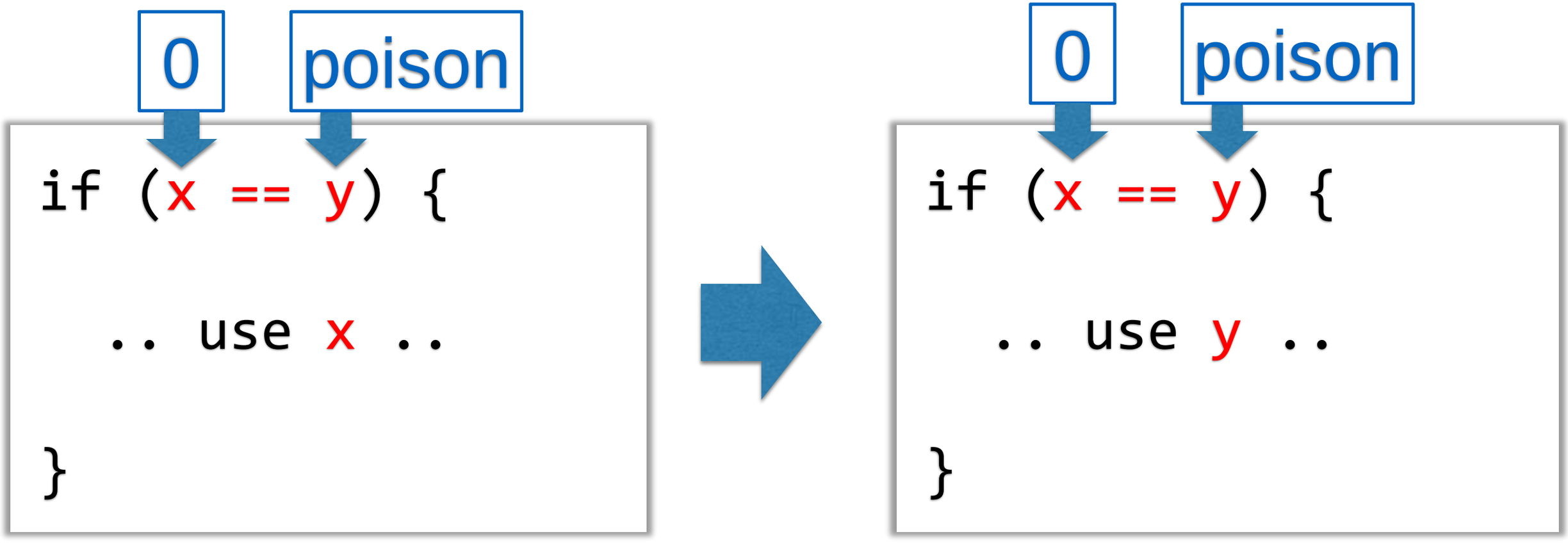
Problems with LLVM's UB

Global Value Numbering (GVN)

LLVM's UB Model:

Branching on poison is

???



Problems with LLVM's UB

Global Value Numbering (GVN)

LLVM's UB Model:

Branching on poison is

???

poison

0 poison

```
if (x == y) {

   .. use x ..

}
```

poison

0 poison

```
if (x == y) {

   .. use y ..

}
```

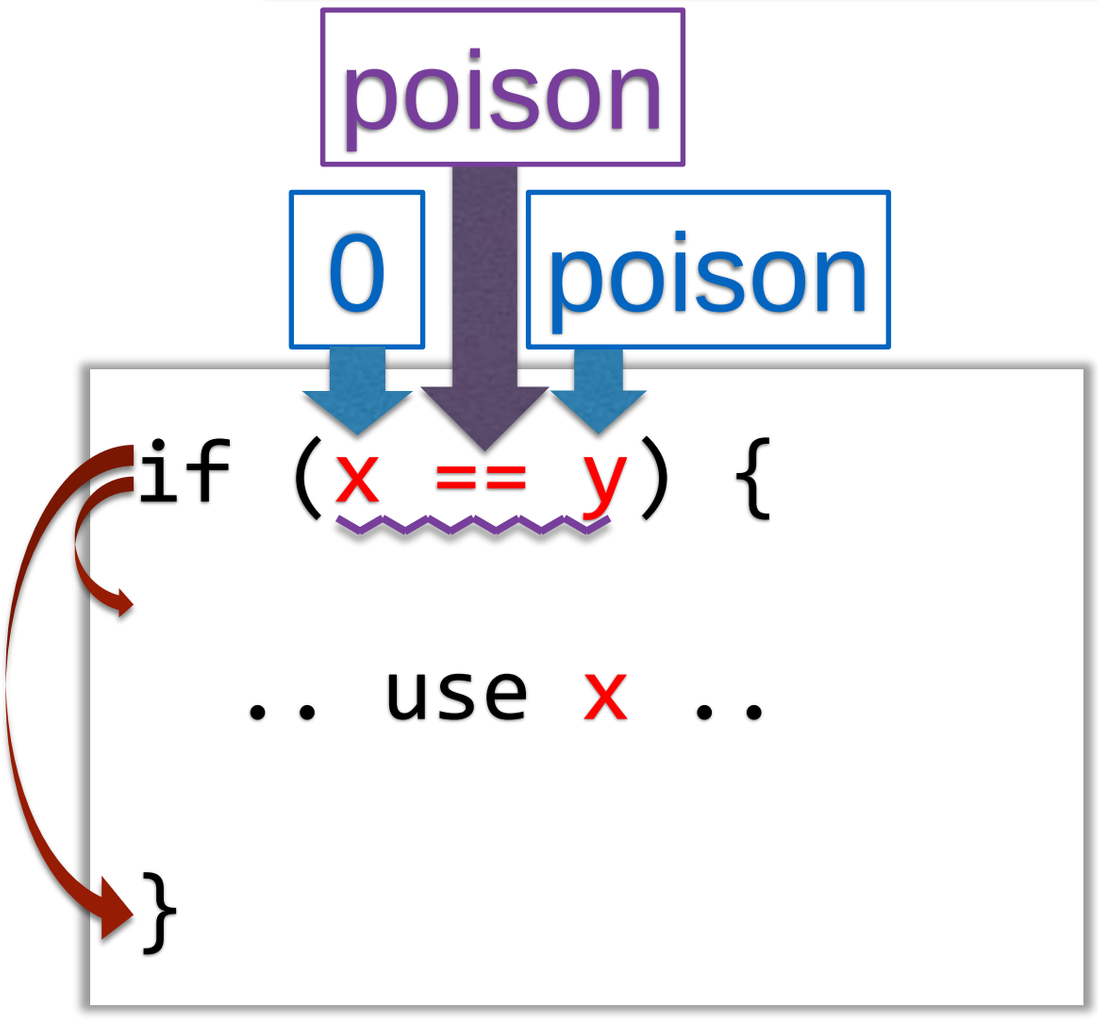
Problems with LLVM's UB

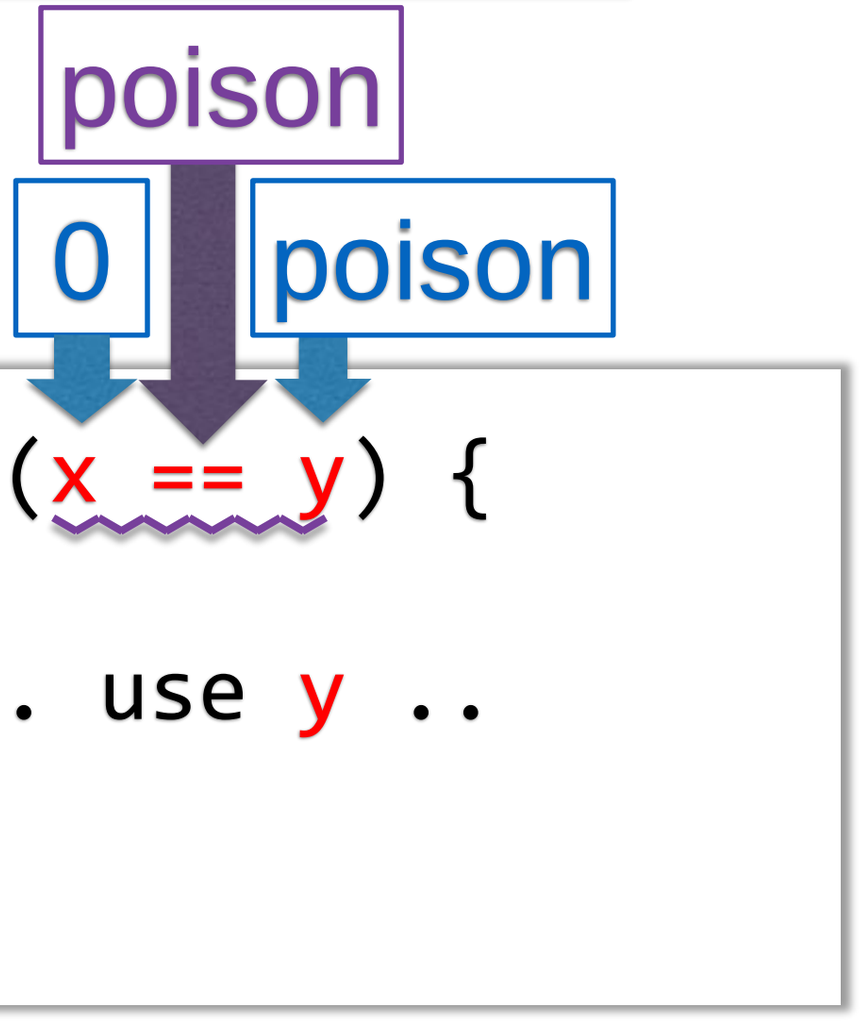
Global Value Numbering (GVN)

LLVM's UB Model:

Branching on poison is

???





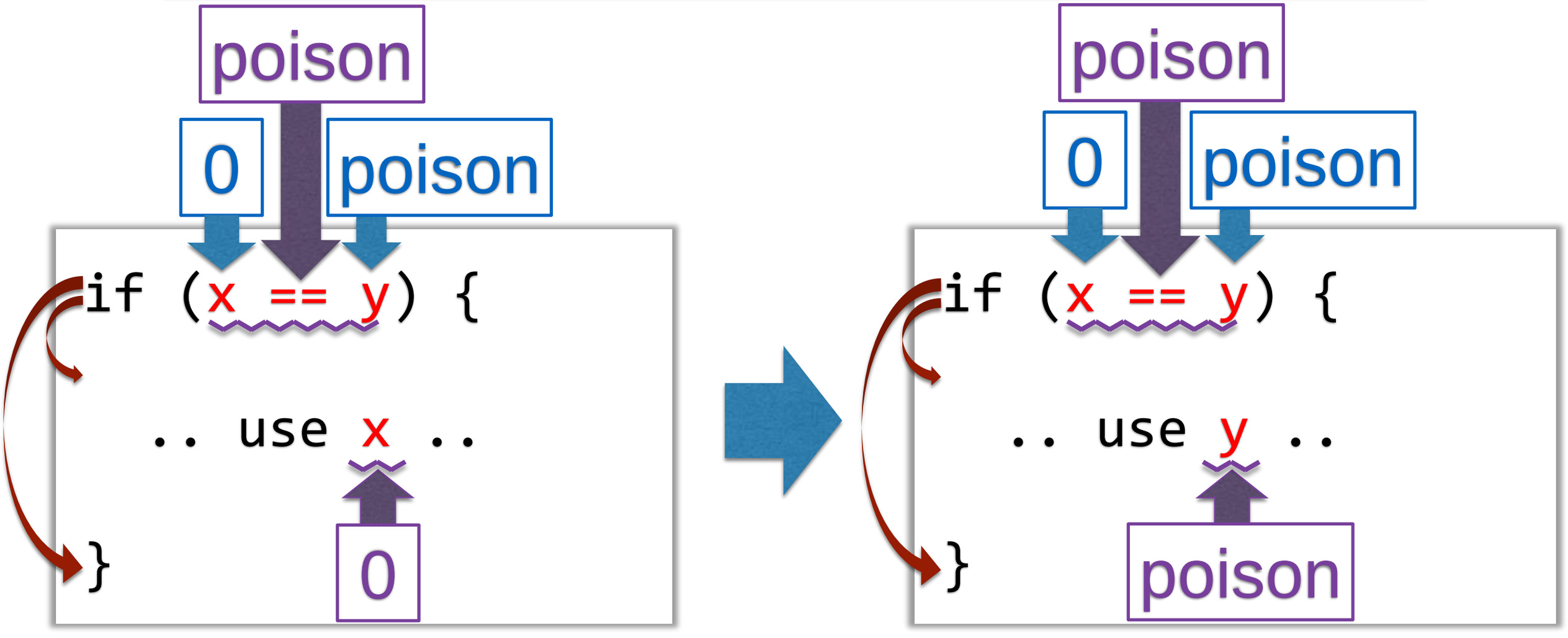
Problems with LLVM's UB

Global Value Numbering (GVN)

LLVM's UB Model:

Branching on poison is

???



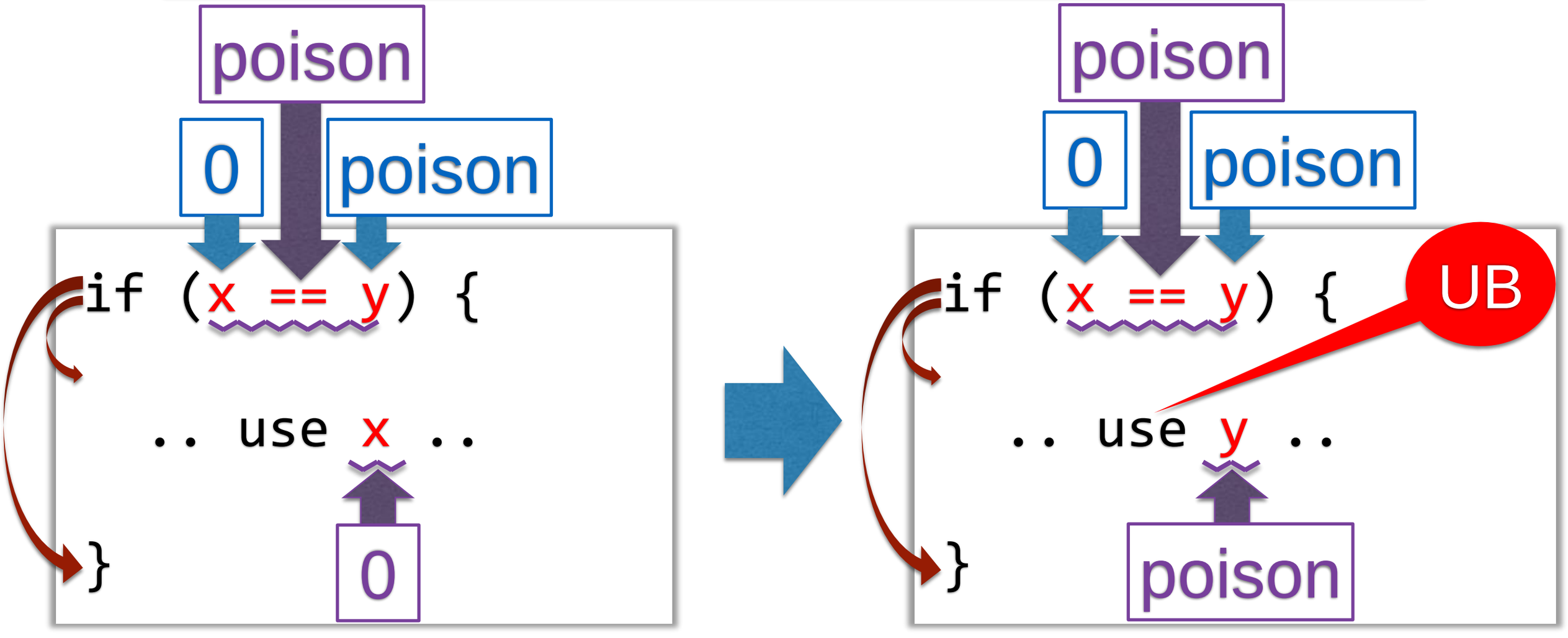
Problems with LLVM's UB

Global Value Numbering (GVN)

LLVM's UB Model:

Branching on poison is

???



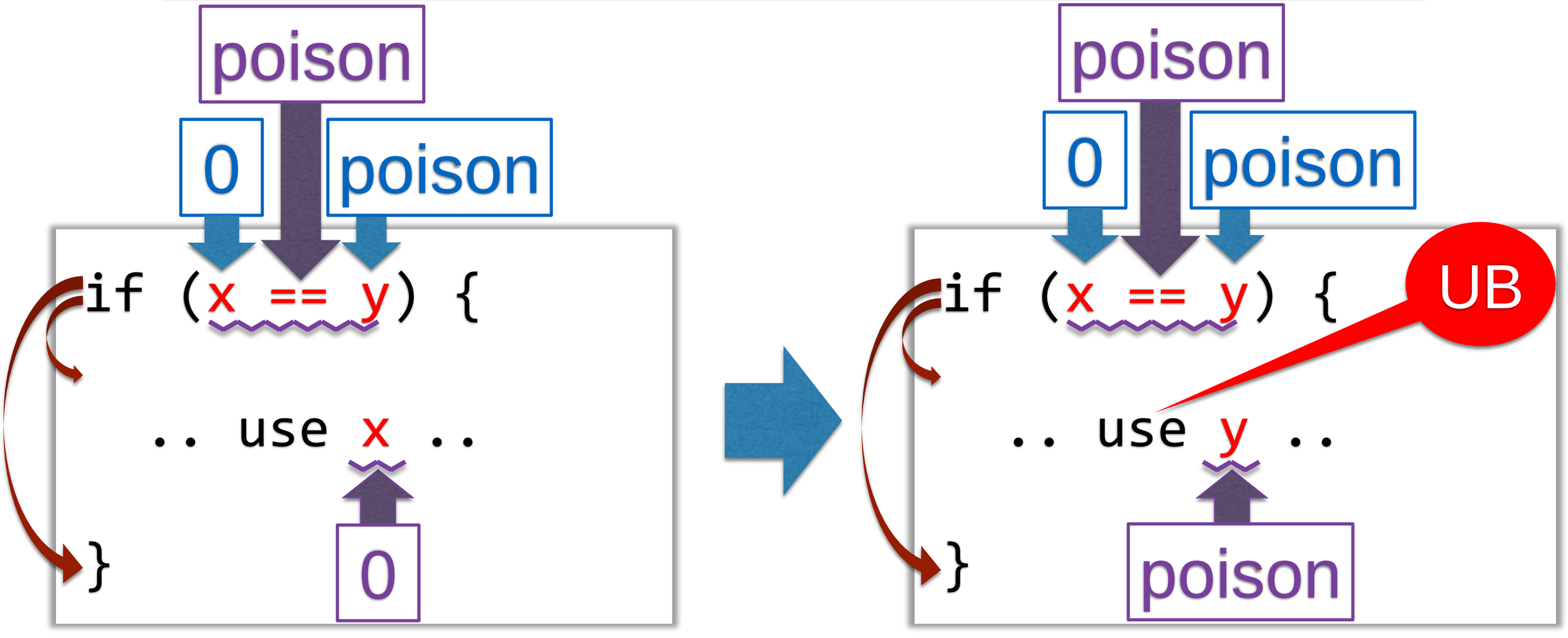
Problems with LLVM's UB

Global Value Numbering (GVN)

LLVM's UB Model:

Branching on poison is

Undefined Behavior



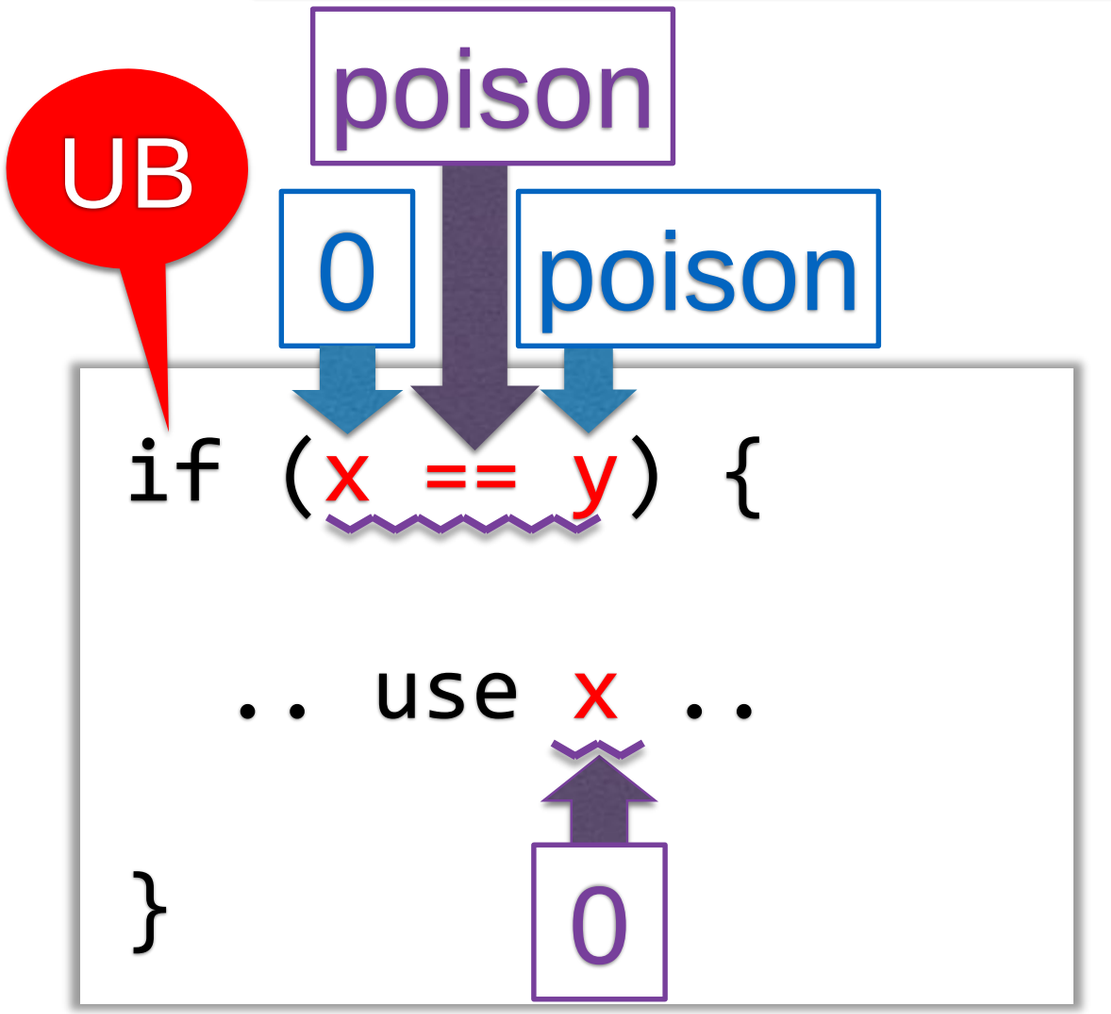
Problems with LLVM's UB

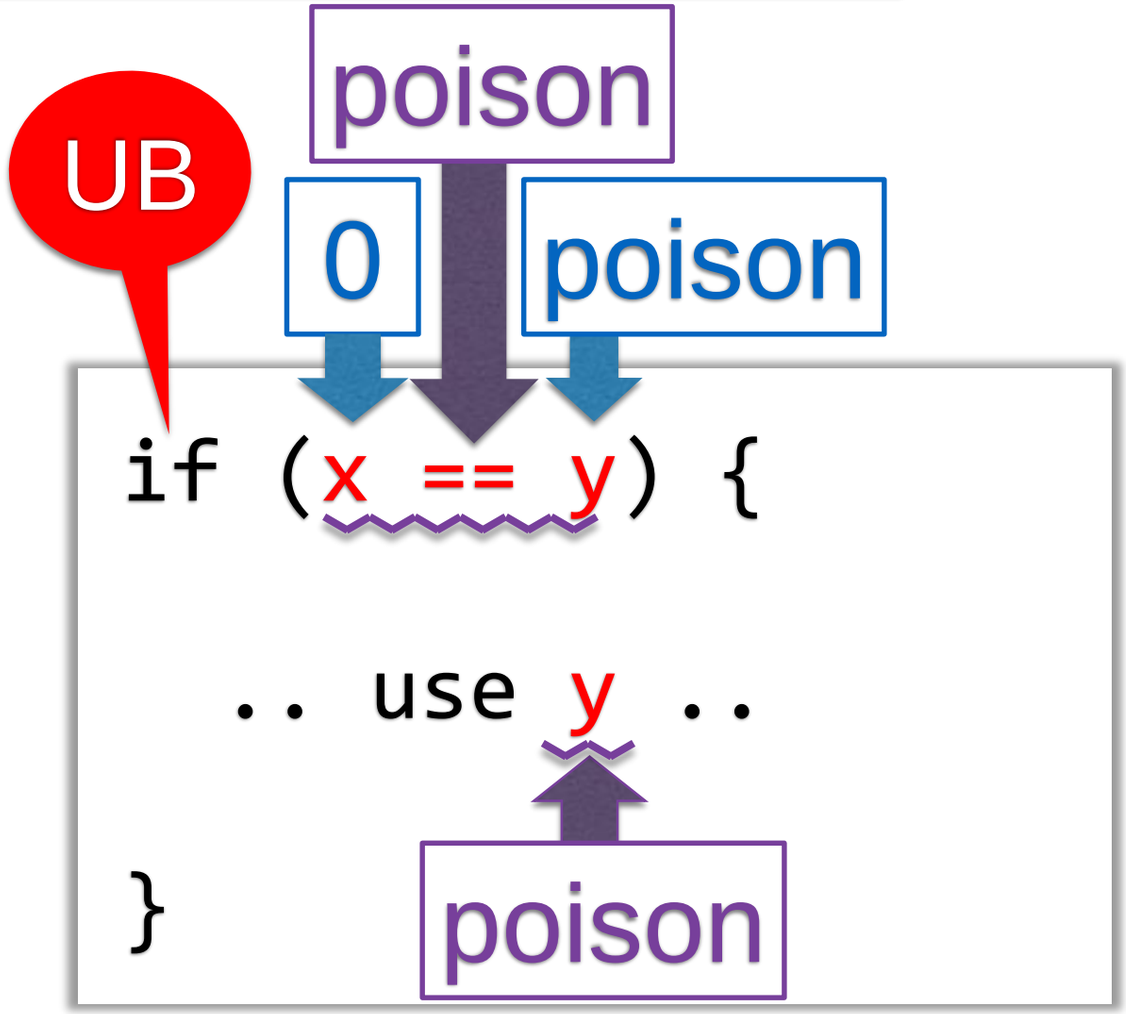
Global Value Numbering (GVN)

LLVM's UB Model:

Branching on poison is

Undefined Behavior





Problems with LLVM's UB

Loop Unswitching (LU)

LLVM's UB Model:

Branching on poison is

Undefined Behavior

```
while (n > 0) {
   if (cond)
      A
   else
      B
}
```

```
if (cond)
   while (n > 0)
   { A }
else
   while (n > 0)
   { B }
```

Problems with LLVM's UB

Loop Unswitching (LU)

LLVM's UB Model:

Branching on poison is

Undefined Behavior

0

```
while (n > 0) {
   if (cond)
      A
   else
      B
}
```

poison

poison

```
if (cond)
   while (n > 0)
   { A }
else
   while (n > 0)
   { B }
```

0

Problems with LLVM's UB

Loop Unswitching (LU)

LLVM's UB Model:

Branching on poison is

Undefined Behavior

0

```
while (n > 0) {
   if (cond)
      A
   else
      B
}
```

poison

UB

poison

```
if (cond)
   while (n > 0)
   { A }
else
   while (n > 0)
   { B }
```

0

Inconsistency in LLVM

- GVN + LU is inconsistent.
- We found a miscompilation bug in LLVM due to the inconsistency.
  - https://bugs.llvm.org/show_bug.cgi?id=31652
  - It is being discussed in the community
  - No solution has been found yet
- 6 months later, LLVM miscompiles itself due to it.
  - https://bugs.llvm.org/show_bug.cgi?id=33652
  - http://lists.llvm.org/pipermail/llvm-dev/2017-July/thread.html#115497

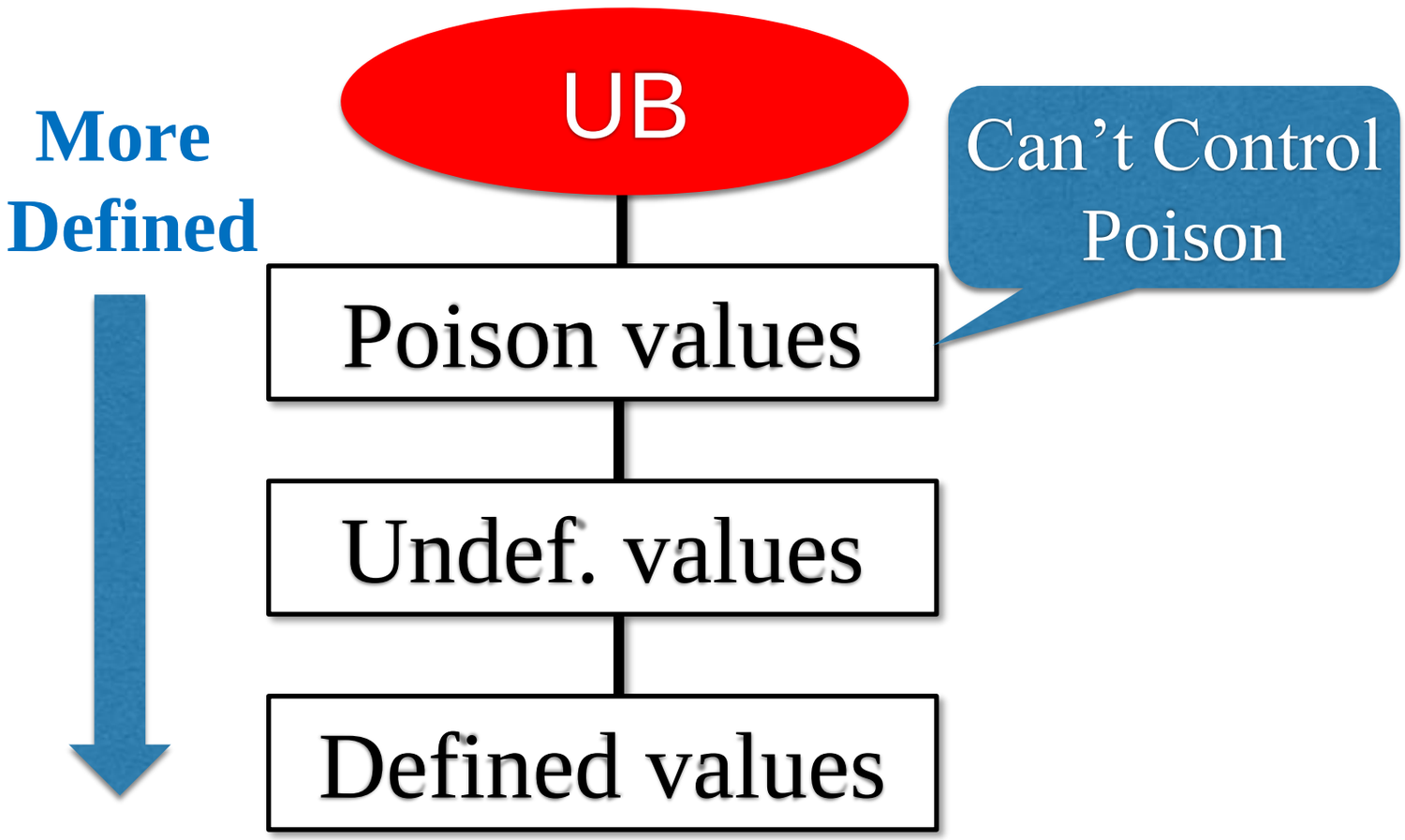
Our Proposal

Overview

Existing Approaches

Complex

Inconsistent  GVN + LU



Overview

Existing Approaches

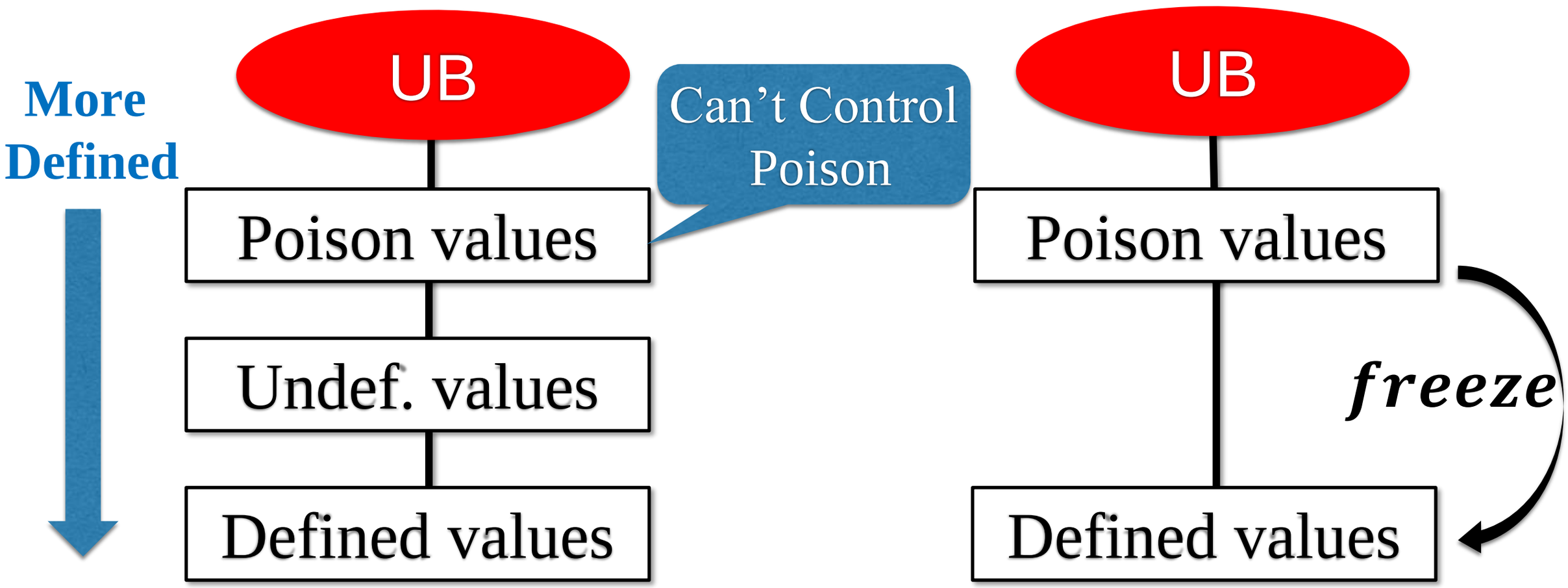
Complex

Inconsistent

GVN + LU

Our Approach

Simpler



Overview

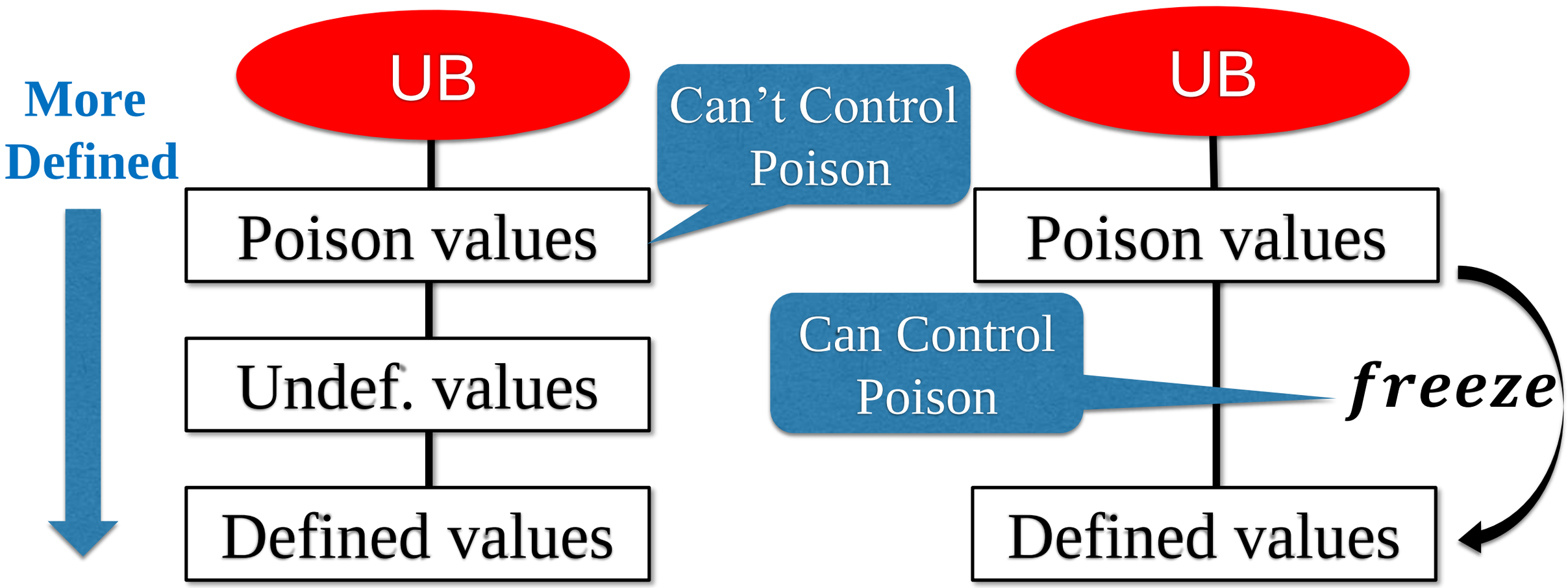
Existing Approaches

Complex

Inconsistent  GVN + LU

Our Approach

Simpler



Overview

Existing Approaches

Complex

Inconsistent

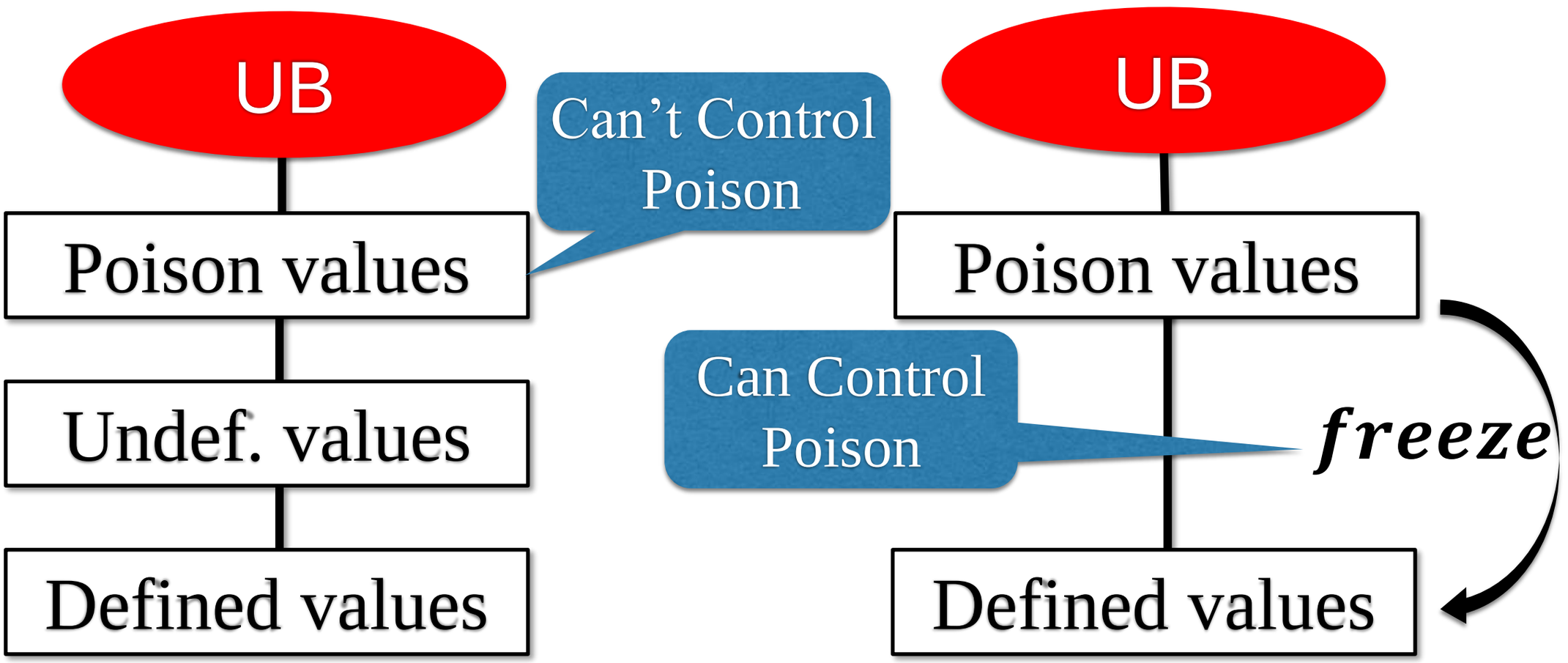
GVN + LU

Our Approach

Simpler

Consistent

More Defined



Key Idea: “Freeze”

- Introduce a new instruction

```
y = freeze x
```

- Semantics:

When `x` is a defined value: `freeze x` ⟶ `x`

When `x` is a poison value: `freeze x` ⟶ `0` `1` `2` …

Nondet. Choice of
A Defined Value

Our Solution

Loop Unswitching

Our UB Model:

Branching on poison is

Undefined Behavior

0

```
while (n > 0) {
  if (cond)
    A
  else
    B
}
```

UB

poison

```
if (cond)
  while (n > 0)
  { A }
else
  while (n > 0)
  { B }
```

Our Solution

Loop Unswitching

Our UB Model:

Branching on poison is

Undefined Behavior

0

```
while (n > 0) {
  if (cond)
    A
  else
    B
}
```

UB

poison

```
if (freeze(cond))
  while (n > 0)
  { A }
else
  while (n > 0)
  { B }
```

Our Solution

Loop Unswitching

Our UB Model:

Branching on poison is

Undefined Behavior

0

```
while (n > 0) {
  if (cond)
    A
  else
    B
}
```

UB

true

false

poison

```
if (freeze(cond))
  while (n > 0)
  { A }
else
  while (n > 0)
  { B }
```

Our Solution

Loop Unswitching

Our UB Model:

Branching on poison is

Undefined Behavior

0

```
while (n > 0) {
   if (cond)
      A
   else
      B
}
```

true | false | poison

```
if (freeze(cond))
   while (n > 0)
   { A }
else
   while (n > 0)
   { B }
```

Summary of Freeze

Compilers can control poison!

- Branching on freeze(poison) => Nondet.
  - Used for Loop Unswitching
- Branching on poison => UB
  - Used for Global Value Numbering

Summary of Freeze

Compilers can control poison!

- Branching on freeze(poison) => Nondet.
  - Used for Loop Unswitching
- Branching on poison => UB
  - Used for Global Value Numbering

Freeze can also fix many other
UB-related problems.

Further Example

Hoisting Division

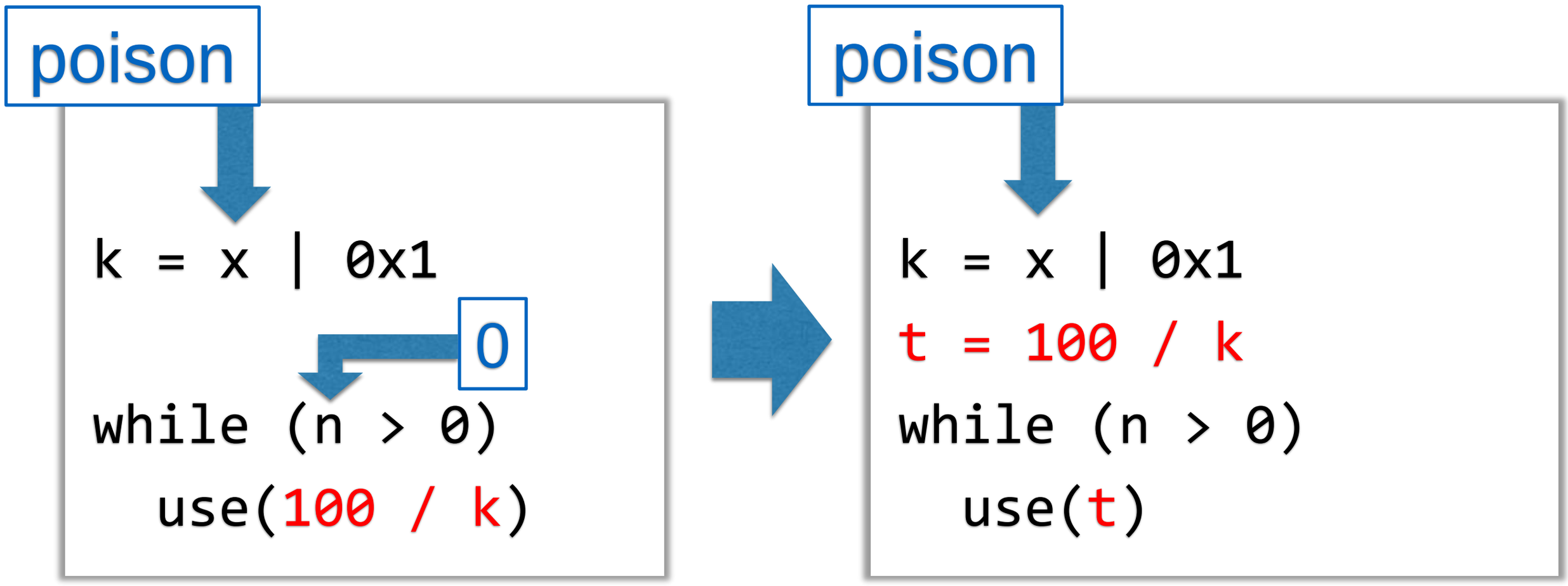
```
// bitwise-or
k = x | 0x1

while (n > 0)
   use(100 / k)
```

```
// bitwise-or
k = x | 0x1
t = 100 / k
while (n > 0)
   use(t)
```

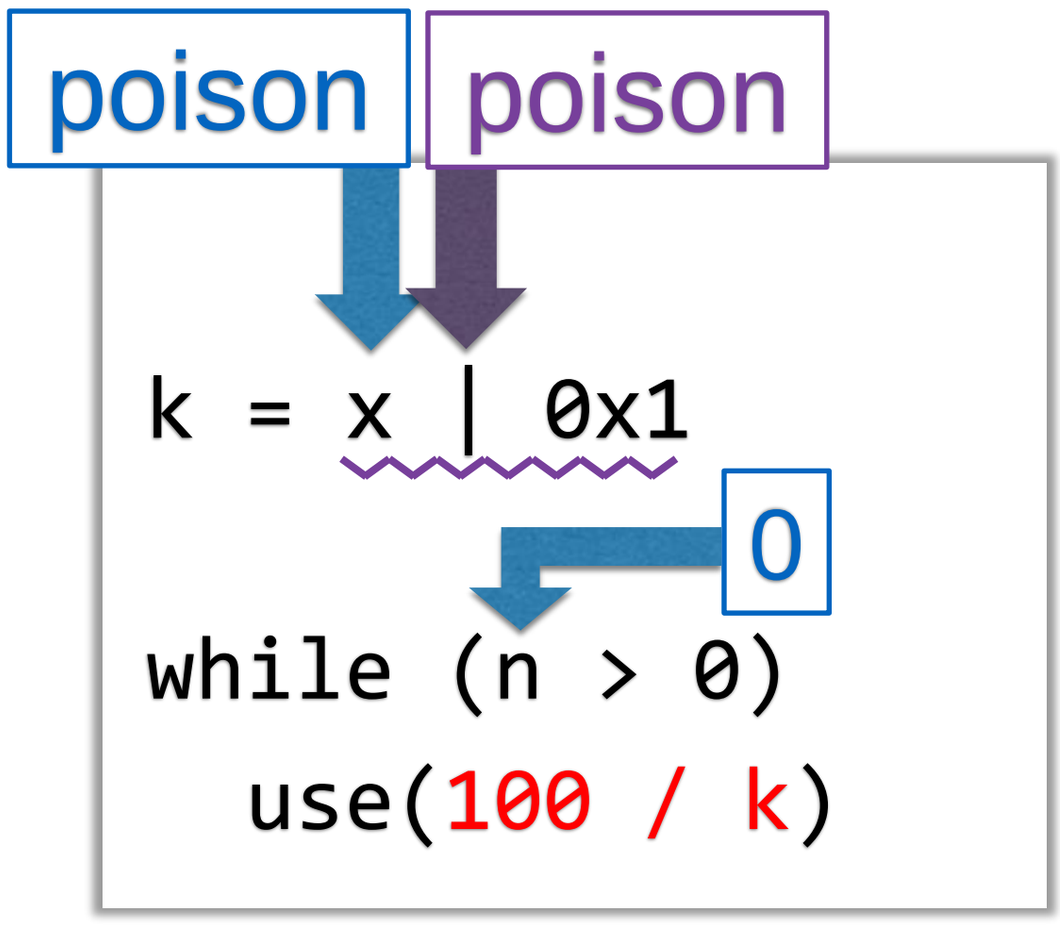
Further Example

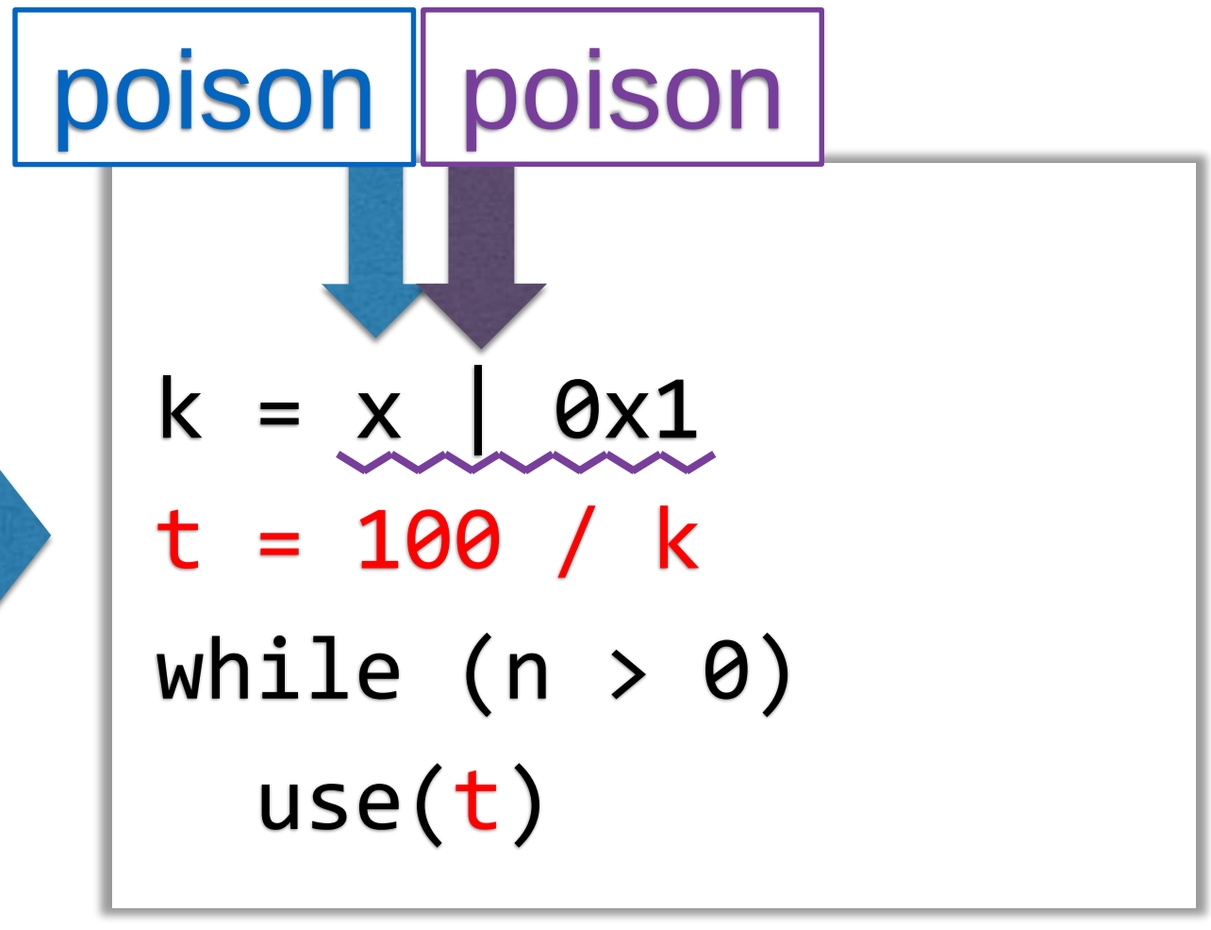
Hoisting Division



Further Example

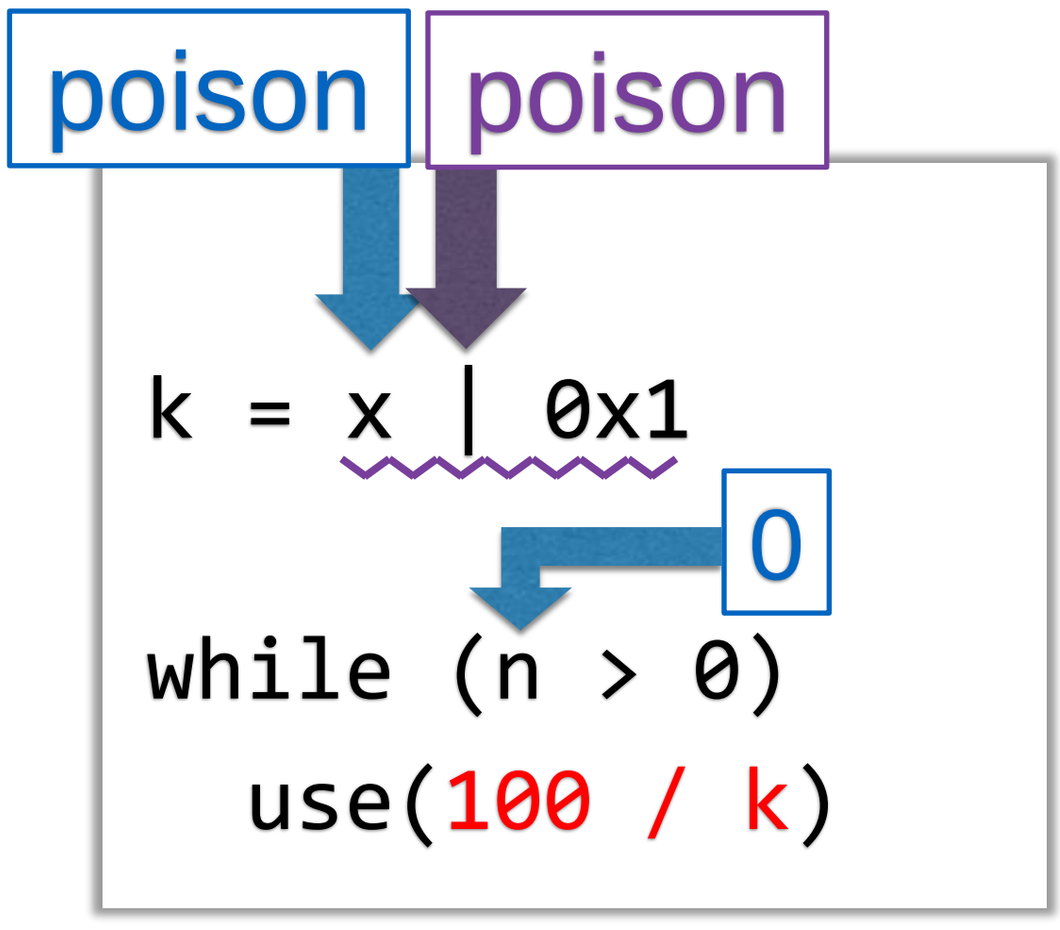
Hoisting Division

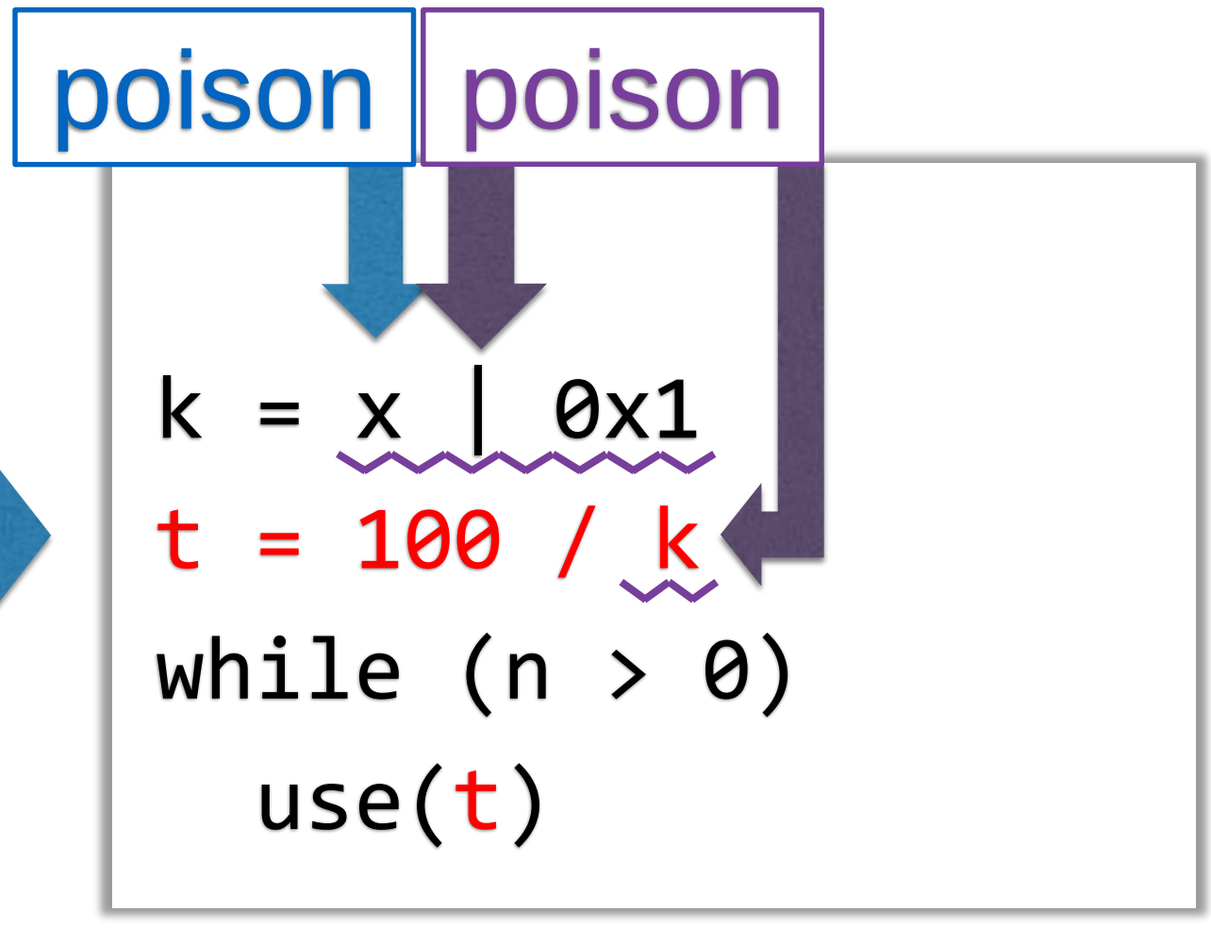




Further Example

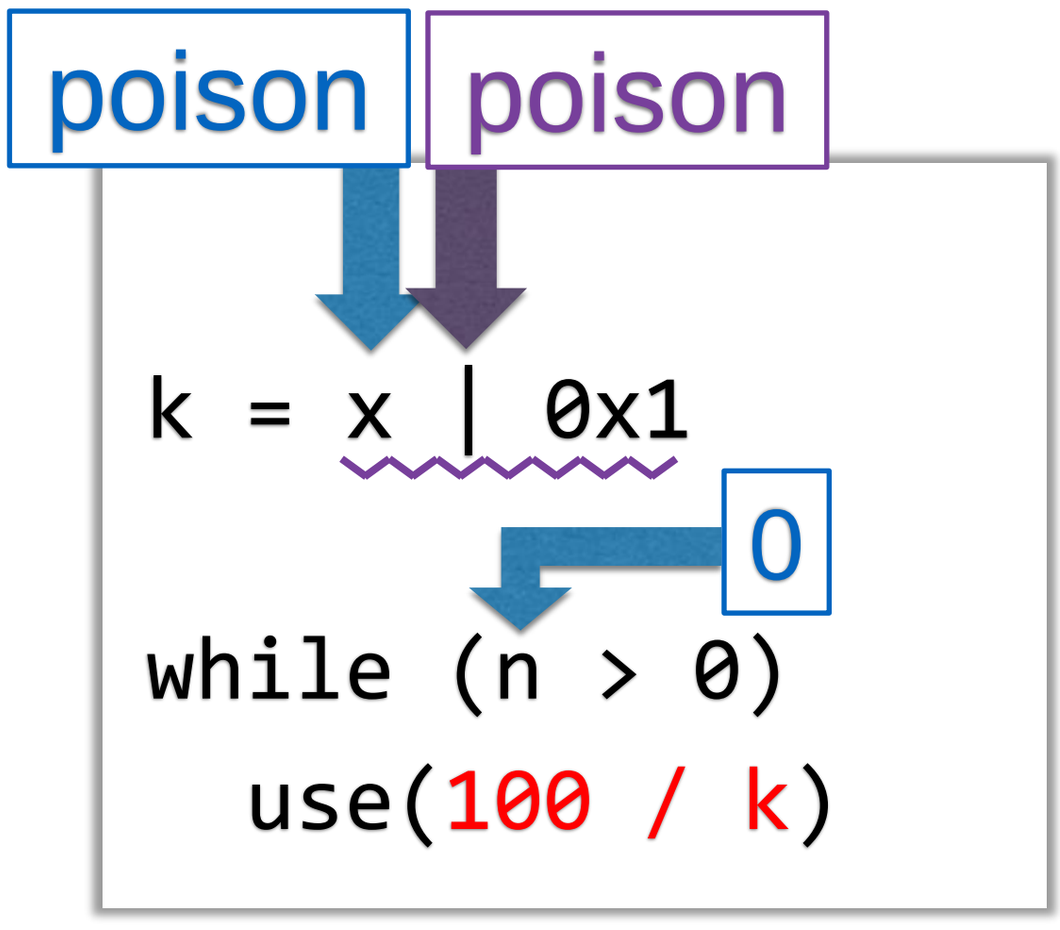
Hoisting Division

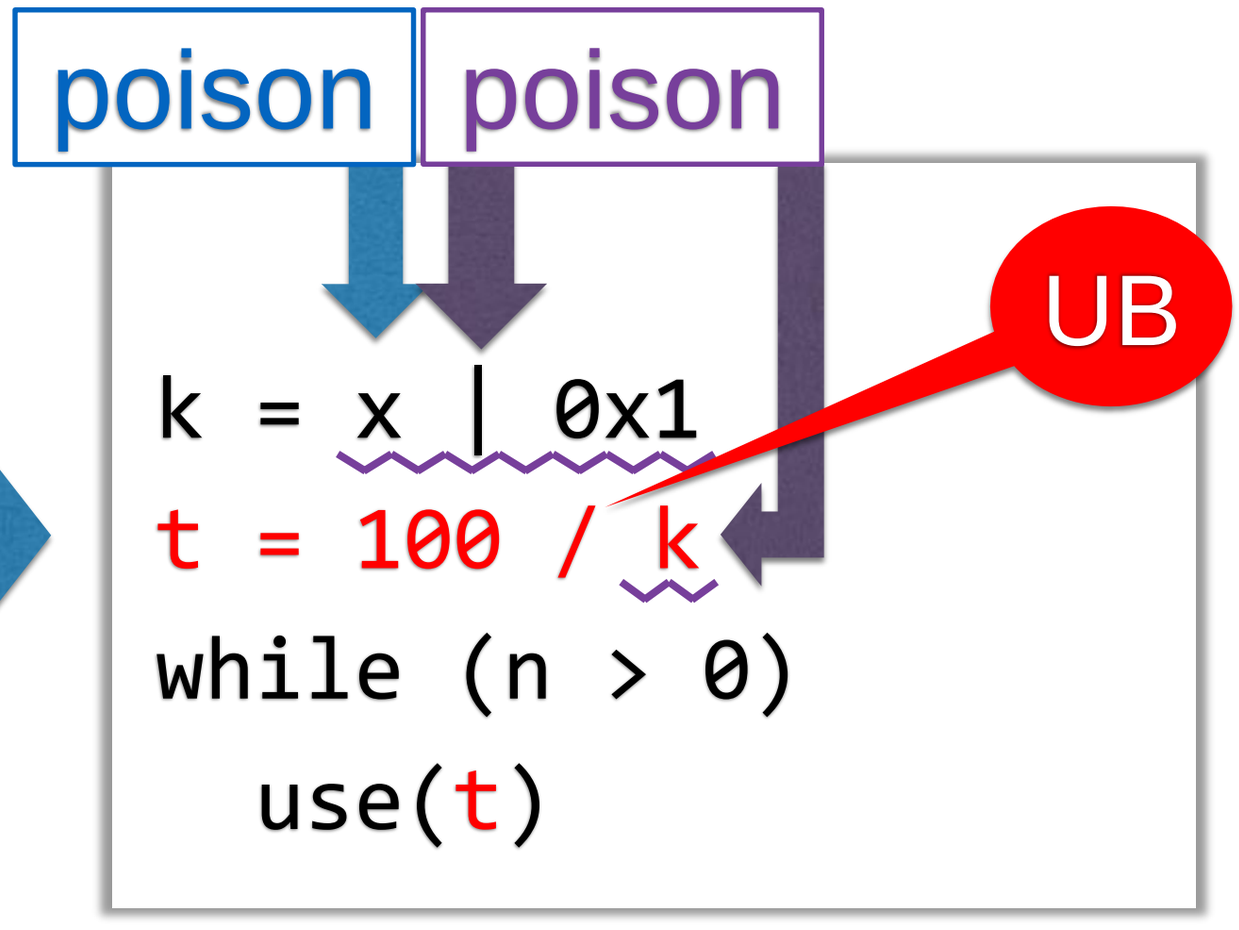




Further Example

Hoisting Division

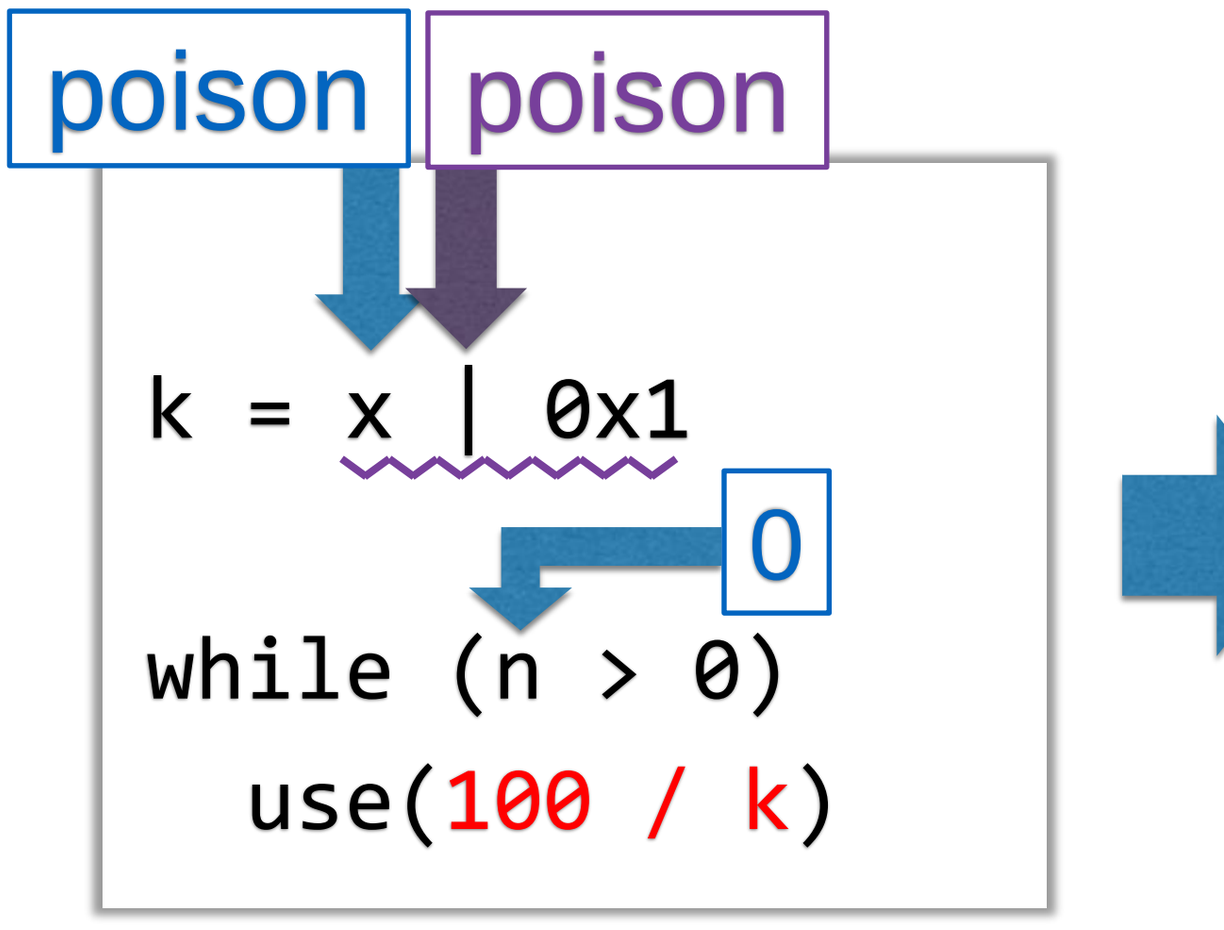


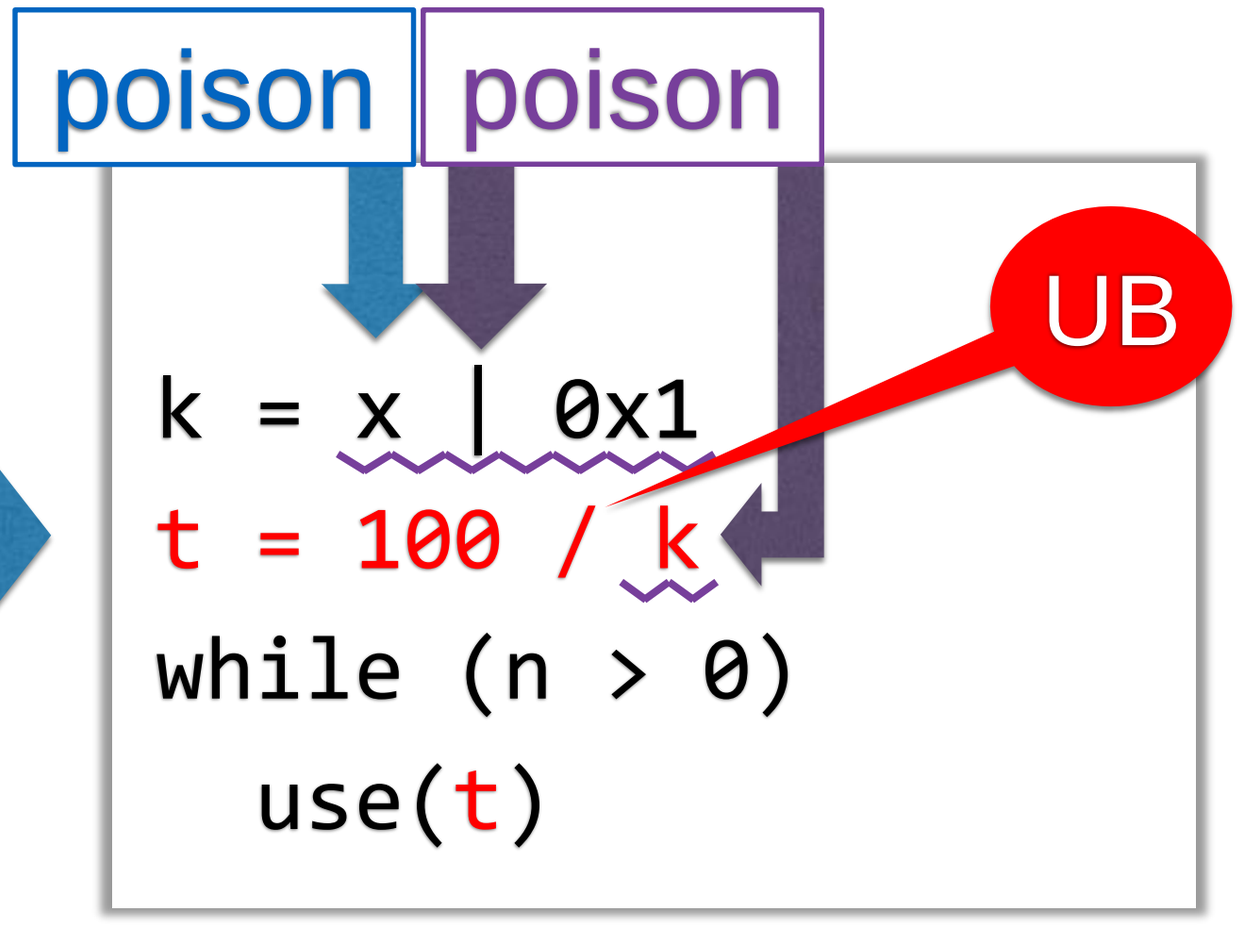


Further Example

Hoisting Division

LLVM does not currently support it.

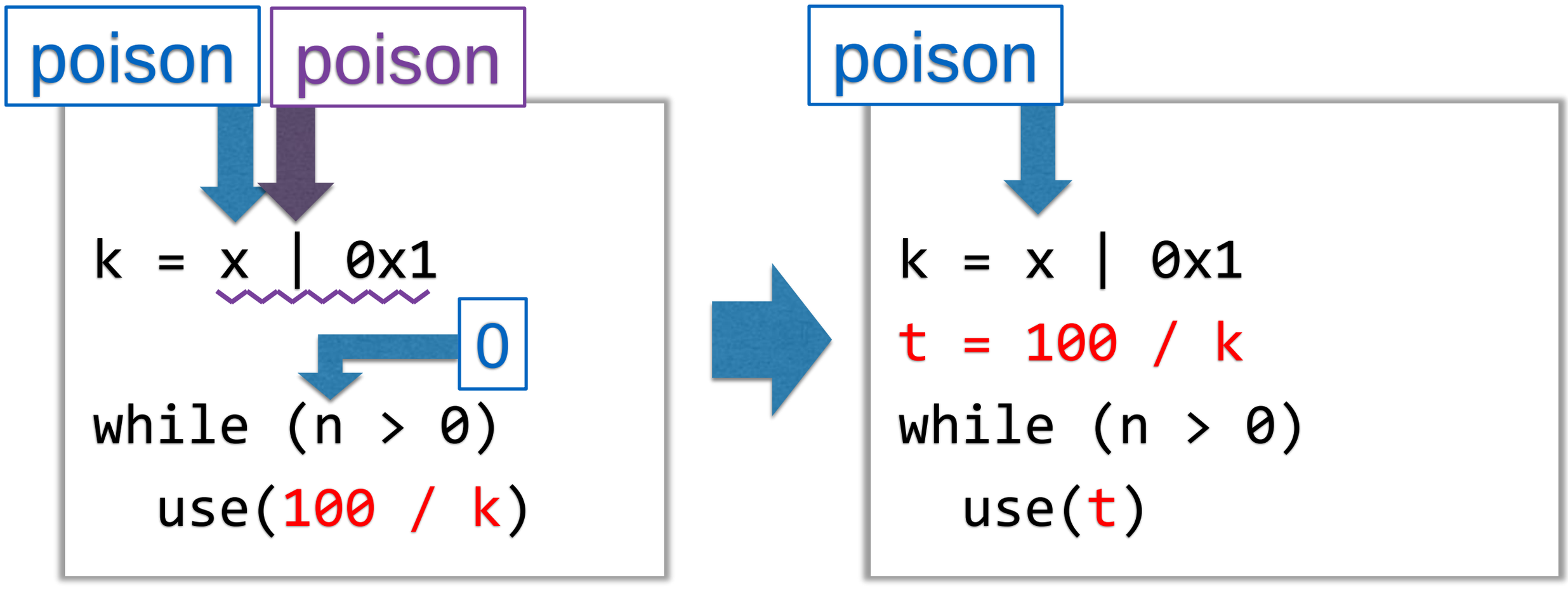




Further Example

Hoisting Division

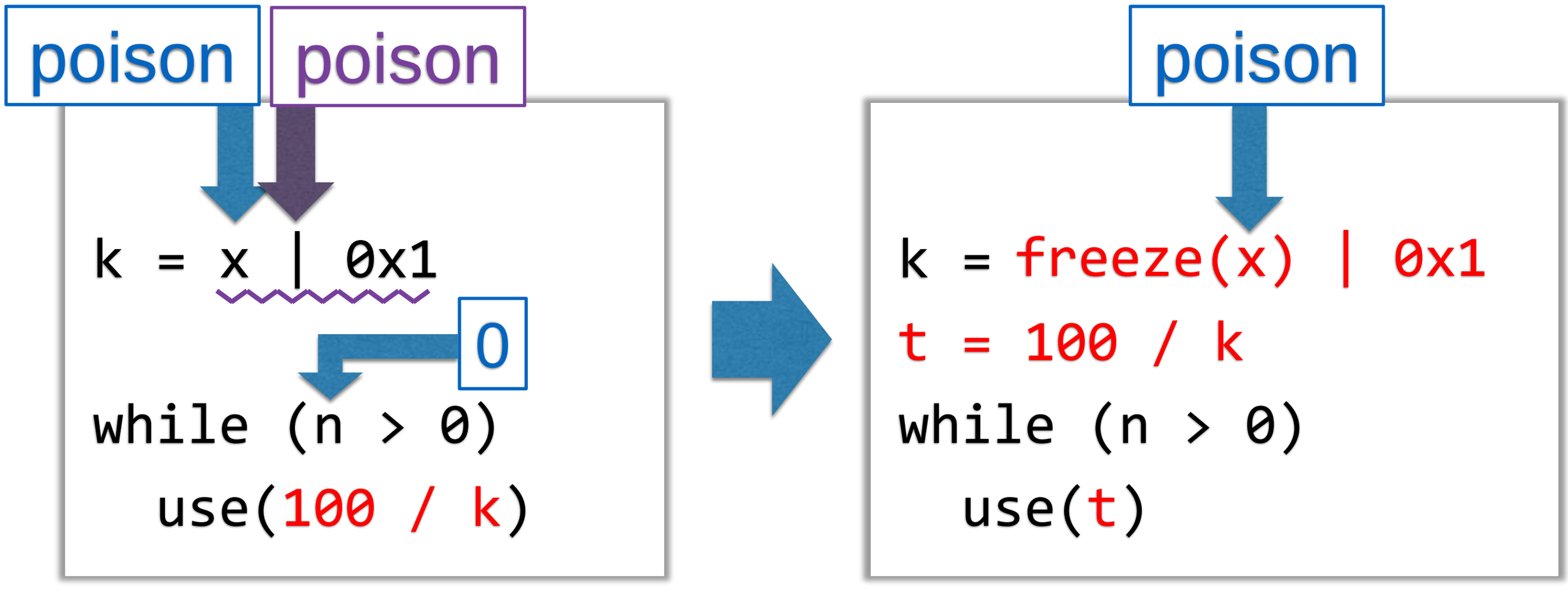
LLVM does not currently support it.



Further Example

Hoisting Division

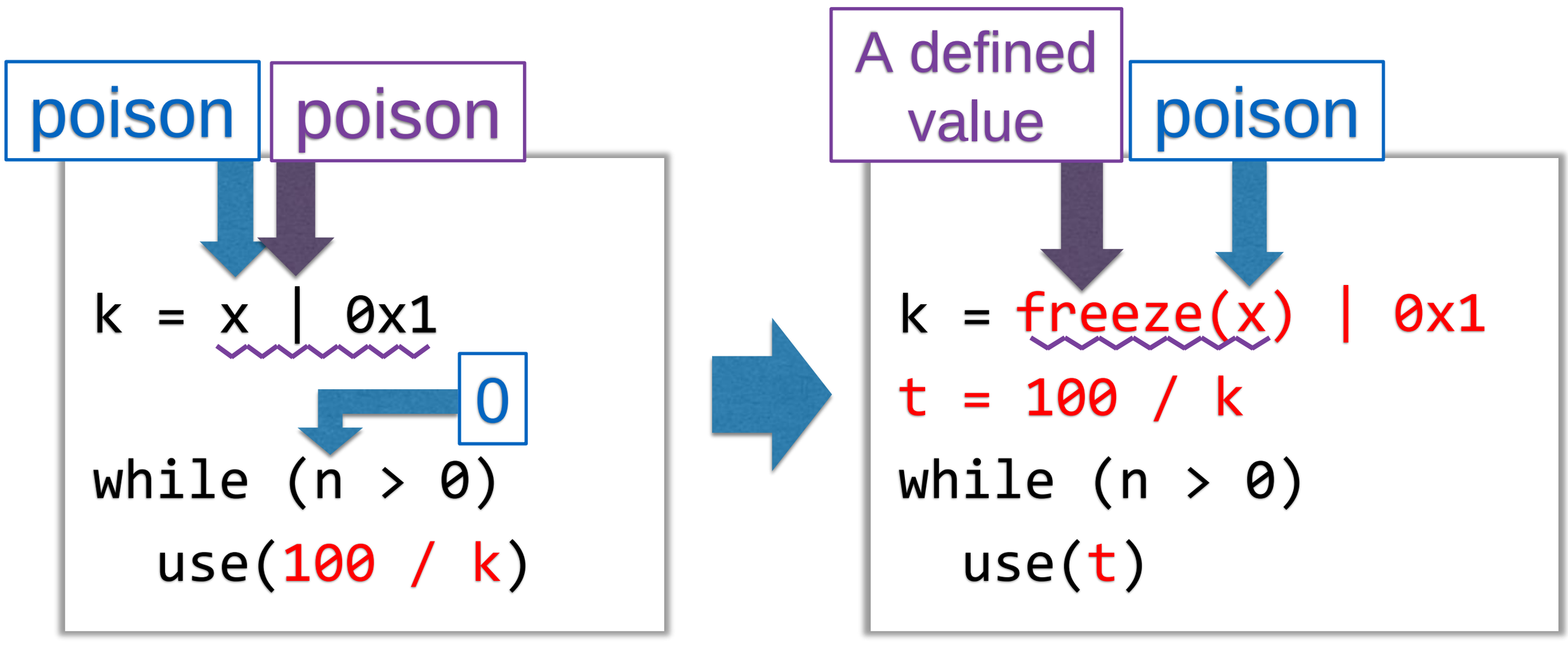
LLVM does not currently support it.



Further Example

Hoisting Division

LLVM does not currently support it.



Further Example

Hoisting Division

LLVM does not currently support it.

poison poison

```
k = x | 0x1
```

0

```
while (n > 0)
   use(100 / k)
```

A defined value poison

```
k = freeze(x) | 0x1
t = 100 / k
while (n > 0)
   use(t)
```

non-zero

Further Example

Hoisting Division

Freeze can make LLVM support it!

poison poison

```
k = x | 0x1
```

0

```
while (n > 0)
   use(100 / k)
```

A defined value poison

```
k = freeze(x) | 0x1
t = 100 / k
```

non-zero

```
while (n > 0)
   use(t)
```

Implementation

- Target: LLVM 4.0 RC 4 (Mar. 2017)
- Add Freeze instruction to LLVM IR
- Bug Fixes Using Freeze
  - Loop Unswitching Optimization
  - C Bitfield Translation to LLVM IR
  - InstCombine Optimizations

* More details are given in the paper

Experiment Results

- Benchmarks (4.6M LOC):
  - SPEC CPU2006
  - LLVM Nightly Test
  - Large Single File Benchmarks
    - Compilation Time: ± 1%
    - Compilation Memory Usage: Max + 2%
    - Generated Code Size: ± 0.5%
    - Execution Time: ± 3%

* More details are given in the paper

“Freeze” Can Fix UB Semantics Without Significant Performance Penalty

- Benchmarks (4.6M LOC):
  - SPEC CPU2006
  - LLVM Nightly Test
  - Large Single File Benchmarks
- Compilation Time: ± 1%
- Compilation Memory Usage: Max + 2%
- Generated Code Size: ± 0.5%
- Execution Time: ± 3%

* More details are given in the paper

Conclusion

- Modern compilers' UB models cannot support some textbook optimizations.

- We propose "freeze" to fix such problems.

- Freeze has little impact on performance.