IS593: Language-based Security

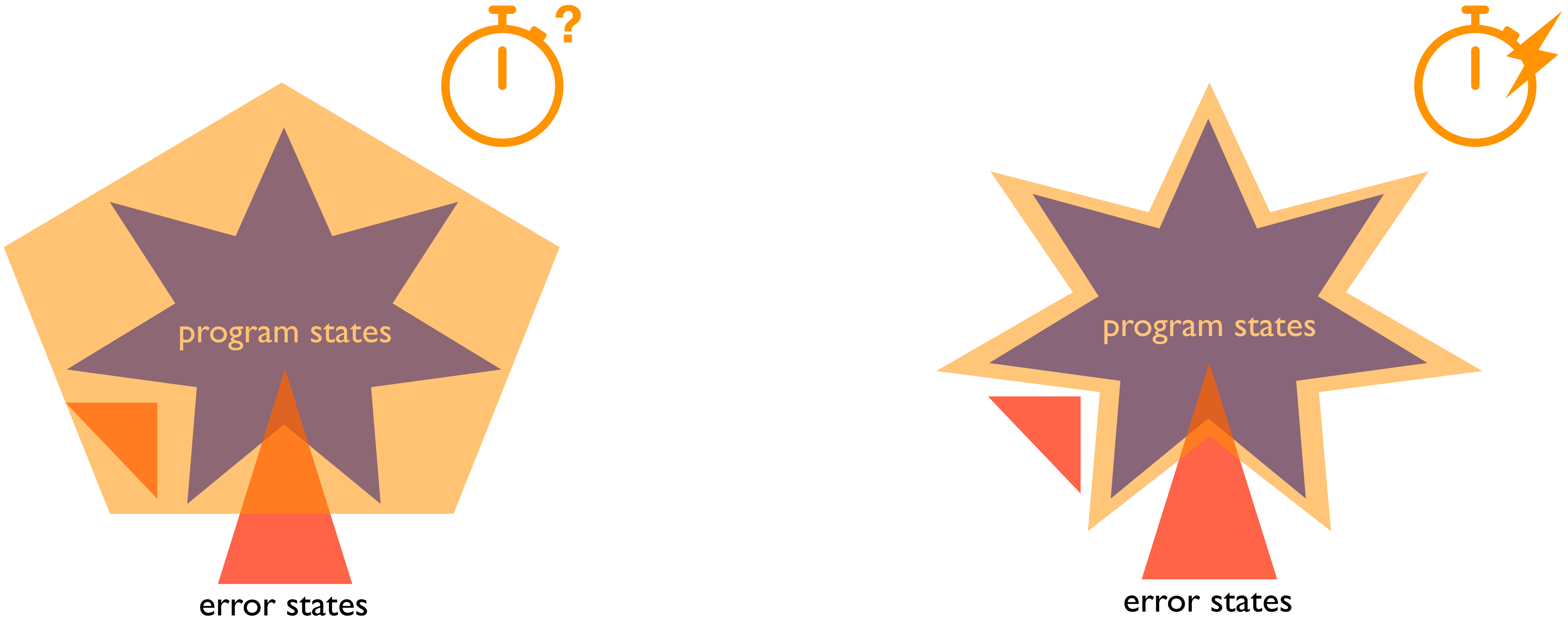
8. Advanced Iteration Techniques

Kihong Heo

KAIST

Advanced Analysis Techniques

- So far, our focus most has been **sound** abstract semantics
- From now on, we will cover several advanced techniques to achieve **efficient** and **accurate** analysis



Iteration Strategies

- **Loop invariant inference**: sequences of abstract iterations
  - Compute **weaker** and **weaker** abstract states until stabilization (via join and widening)
- "***Loop is evil***": a main source of imprecision in static analysis
- Needs for techniques to improve the precision

Problem 1: Overused Widening

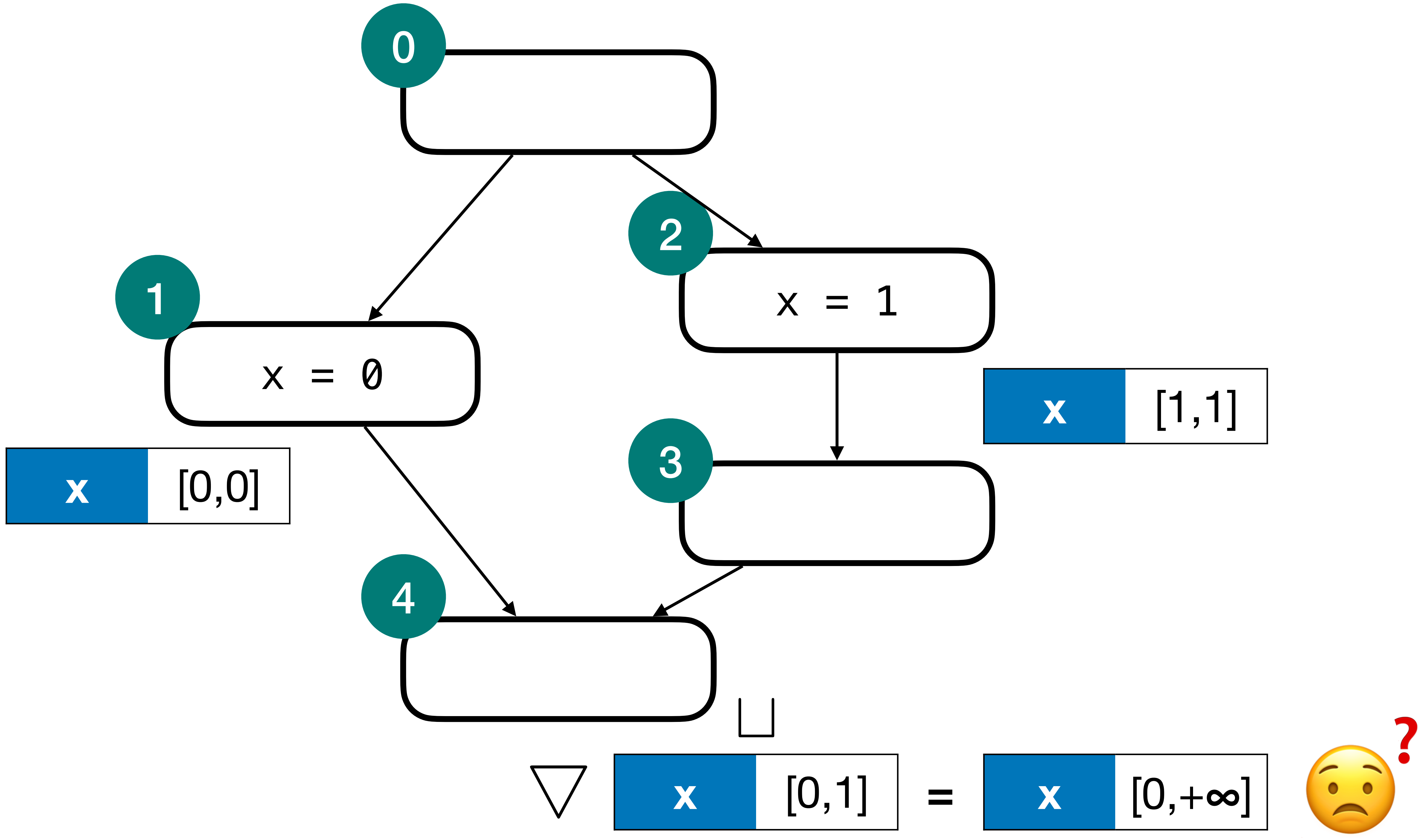
- Recall the worklist algorithm

```
$X, R : \mathbb{L} \to \mathbb{M}^\sharp$
$F^\sharp : (\mathbb{L} \to \mathbb{M}^\sharp) \to (\mathbb{L} \to \mathbb{M}^\sharp)$
$\mathit{Worklist} : \wp(\mathbb{L})$
begin
    $\mathit{Worklist} \leftarrow \mathbb{L}$
    $X \leftarrow \bot$
    repeat
        $R \leftarrow X$
        $X \leftarrow X \triangledown F^\sharp(X|_{\mathit{Worklist}})$
        $\mathit{Worklist} \leftarrow \{l \in \mathbb{L} \mid X(l) \not\sqsubseteq R(l)\}$
    until $\mathit{Worklist} = \emptyset$
    return $R$
end
```

Widening Everywhere?

Problem 1: Overused Widening

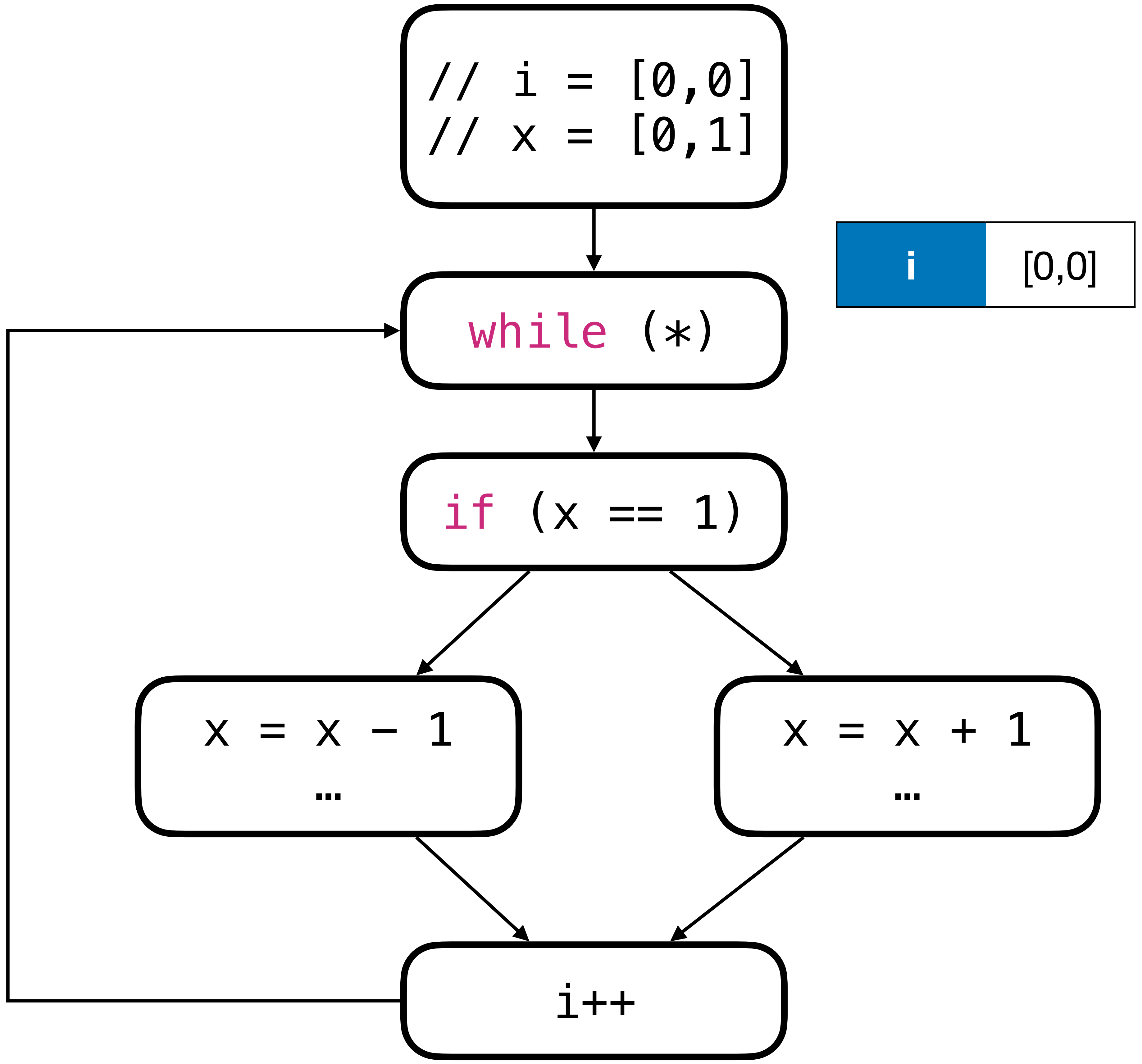
- Consider an analysis with the interval abstract domain



Solution: Selective Widening

- Apply widening only when the label is the target of a **cycling** control flow
  - e.g., while-loop heads, targets of cycling gotos, (spurious) call-cycle
- For other labels, apply the **join** operation instead

Case 1: Loop Heads



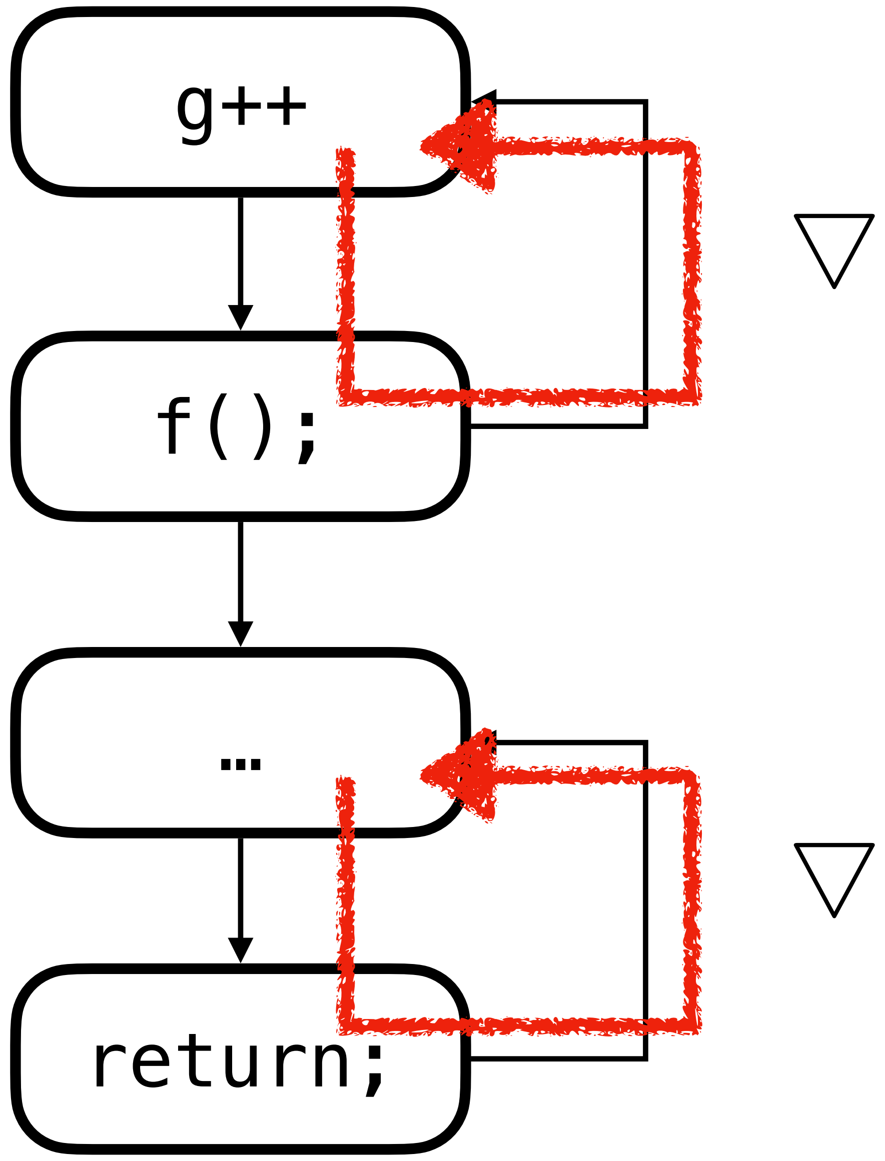
i [0,0] ▽ i [0,1] = i [0,+∞]

x [0,0] ⊔ x [0,1] = x [0,1] 👍

Case 2: Call-cycle

- Widening when a **recursive call-cycle** exists

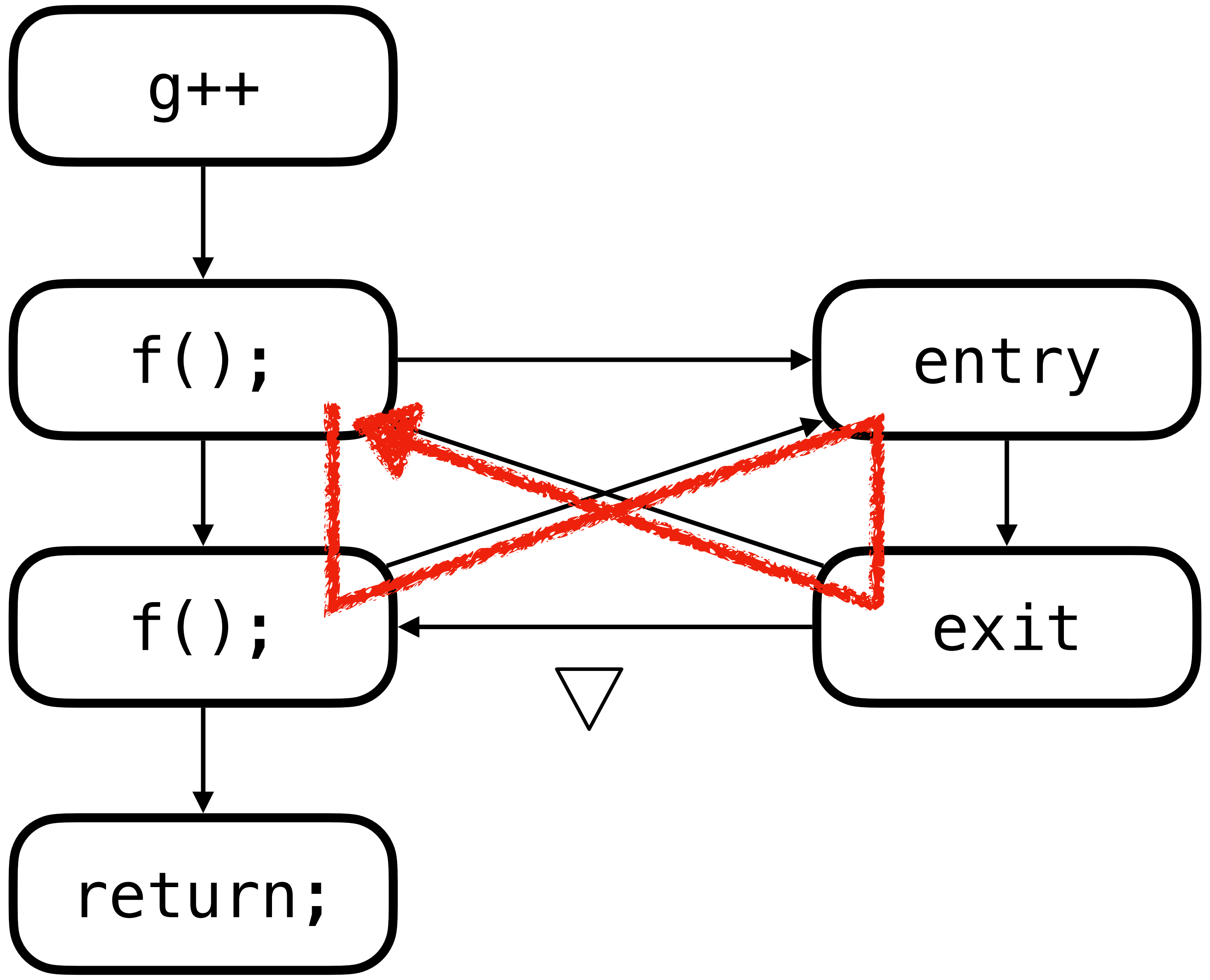
```
void f() {
  g++;
  f();
  …
  return;
}
```



Case 2: Call-cycle (Cont'd)

- Widening when even **spurious-cycle** happens
  - For example, context-insensitive analysis

```
int main() {
  g++;
  f();  // non-recursive
  f();
  return;
}
```



Caveat

- In general, cycle detection cannot be done before analysis
  - control-flow is **dynamic** (e.g., higher-order functions, exceptions, etc)
- Possible solutions:
  - online cycle-detection (during analysis): precise but costly
  - offline cycle-detection with pre-analysis (before analysis): imprecise but lightweight

Problem 2: Hasty Join

```
x = ?; // any value
i = 1;
while (i > 0) {
  if(x < 0 || x > 1000) {
    x = 0;
  } else {
    x = x + 1;
  }
  input(i);
}
// actually, x is in [0, 1001]
```

Initialization step

- The abstract value for x with a naive approach would be [-∞, +∞]
- Idea: **detach** the first iteration from the rest

Solution: Loop Unrolling

```
x = ?; // any value
i = 1;
while (i > 0) {
  if(x < 0 || x > 1000) {
    x = 0;
  } else {
    x = 1 + x;
  }
  input(i);
}
// actually, x is in [0, 1001]
```

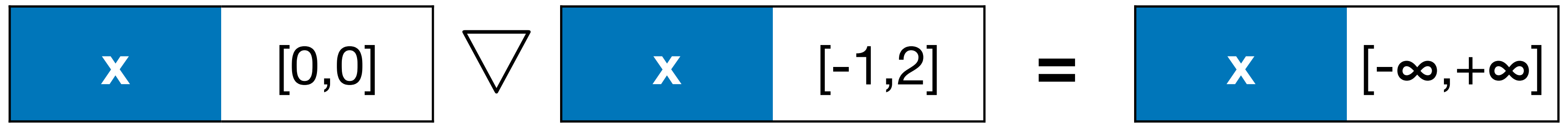
```
x = ?; // any value
i = 1;
if(x < 0 || x > 1000) {
  x = 0;
} else {
  x = 1 + x;
}
input(i);
// x is in [0, 1001]
while (i > 0) {
  if(x < 0 || x > 1000) {
    x = 0;
  } else {
    x = 1 + x;
  }
  input(i);
}
// x is in [0, 1001]
```

first iter.

rest

Problem 3: Hasty Widening

```
x = 0;
while (rand()) {
  if(rand()) {
    x = -1;
  } else {
    x = x + 2;
  }
}
// x >= -1
```



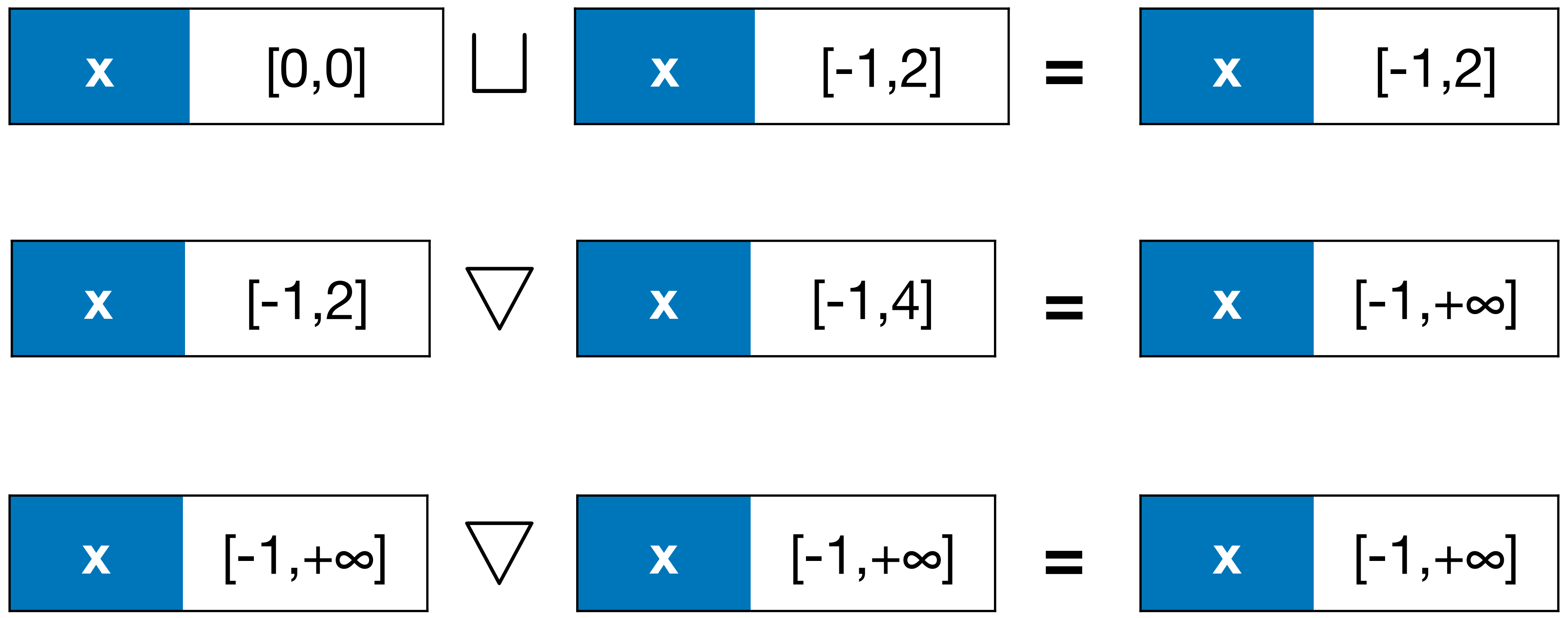
- The abstract value of x with a naive approach would be [-∞, +∞]

- Idea: **delay** the application of widening for the first N iterations

Solution: Delayed Widening

```
x = 0;
while (rand()) {
  if(rand()) {
    x = -1;
  } else {
    x = x + 2;
  }
}
// x >= -1
```

Delayed widening where N = 1



x [0,0] ⊔ x [-1,2] = x [-1,2]

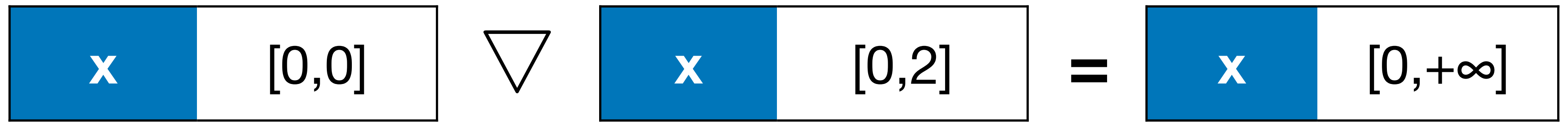
x [-1,2] ▽ x [-1,4] = x [-1,+∞]

x [-1,+∞] ▽ x [-1,+∞] = x [-1,+∞]

Fixed Point!

Problem 4: Excessive Widening

```
x = 0;
while (x <= 100) {
  if(x >= 50) {
    x = 10;
  } else {
    x = x + 2;
  }
}
// actually, x is in [0, 50]
```



x [0,0] ▽ x [0,2] = x [0,+∞]

- The abstract value of x with a naive approach is [0, +∞]

- Idea: use a **slower and more precise** widening

Solution: Widening with Thresholds

- Take several small steps and stops at pre-defined threshold values
- For example, consider only one threshold *B*:

A naive widening operator

$$[n,p] \ \triangledown \ [n,q] = \begin{cases} [n,p] & \text{if } p \geq q \\ [n,+\infty] & \text{if } p < q \end{cases}$$

A widening with thresholds

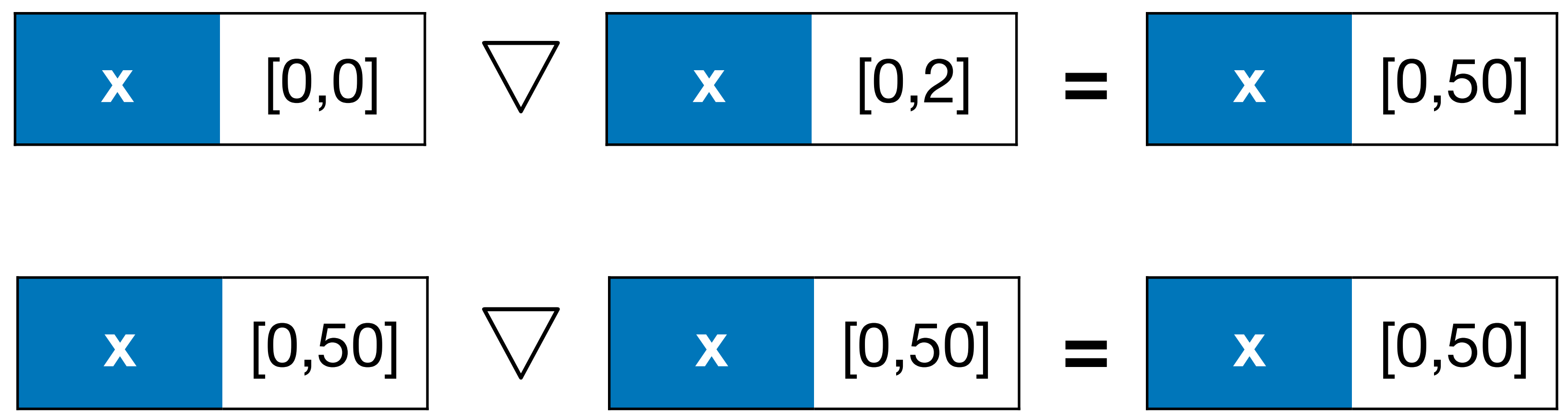
$$[n,p] \ \triangledown \ [n,q] = \begin{cases} [n,p] & \text{if } p \geq q \\ [n,B] & \text{if } p < q \leq B \\ [n,+\infty] & \text{if } B < q \end{cases}$$

*only the right bounds, for brevity

Widening with Thresholds

```
x = 0;
while (x <= 100) {
  if(x >= 50) {
    x = 10;
  } else {
    x = x + 2;
  }
}
```

Thresholds = {50}



x [0,0] ▽ **x** [0,2] **=** **x** [0,50]

x [0,50] ▽ **x** [0,50] **=** **x** [0,50]

Fixed Point!

Summary

- “***Loop is evil***”: one of the main source of imprecision
- Important to design effective iteration techniques
  - no universal solutions
  - depending on the target program’s characteristics
- Need for domain knowledge (human experts or learning techniques)