

---

# Cascade: A Universal Programmer-assisted Type Qualifier Inference Tool

Mohsen Vakilian\*

Amarin Phaosawasdi\*

Michael D. Ernst†

Ralph E. Johnson\*

\*University of Illinois at Urbana-Champaign

†University of Washington

# Type qualifiers allow additional static type checks

```
static int oldSubindex(@Nullable MathVector ic, int l) {  
    int i = 0;  
    for (int k = 0; k < MathVector.NDIM; k++) {  
        if (((int) ic.value(k) & 1) != 0)  
            i += Cell.NSUB >> (k + 1);  
    }  
    return i;  
}
```

# Java 8 supports custom type systems\*

- Locking
- Aliasing
- Interning
- Immutability
- Tainting

\*[checkerframework.org](http://checkerframework.org)

# Java 8 supports custom type systems\*

- Locking
- Aliasing
- Interning
- Immutability
- Tainting

+ additional static checks

\*[checkerframework.org](http://checkerframework.org)

# Java 8 supports custom type systems\*

- Locking
- Aliasing
- Interning
- Immutability
- Tainting

+ additional static checks

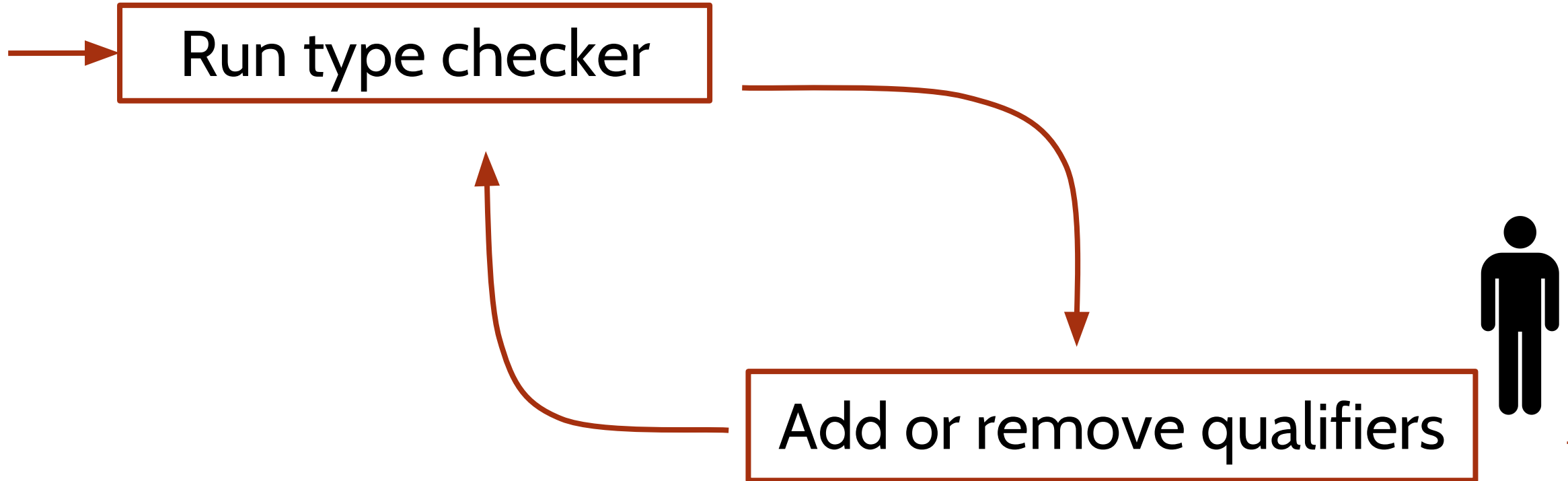
- additional code annotation

\*[checkerframework.org](http://checkerframework.org)

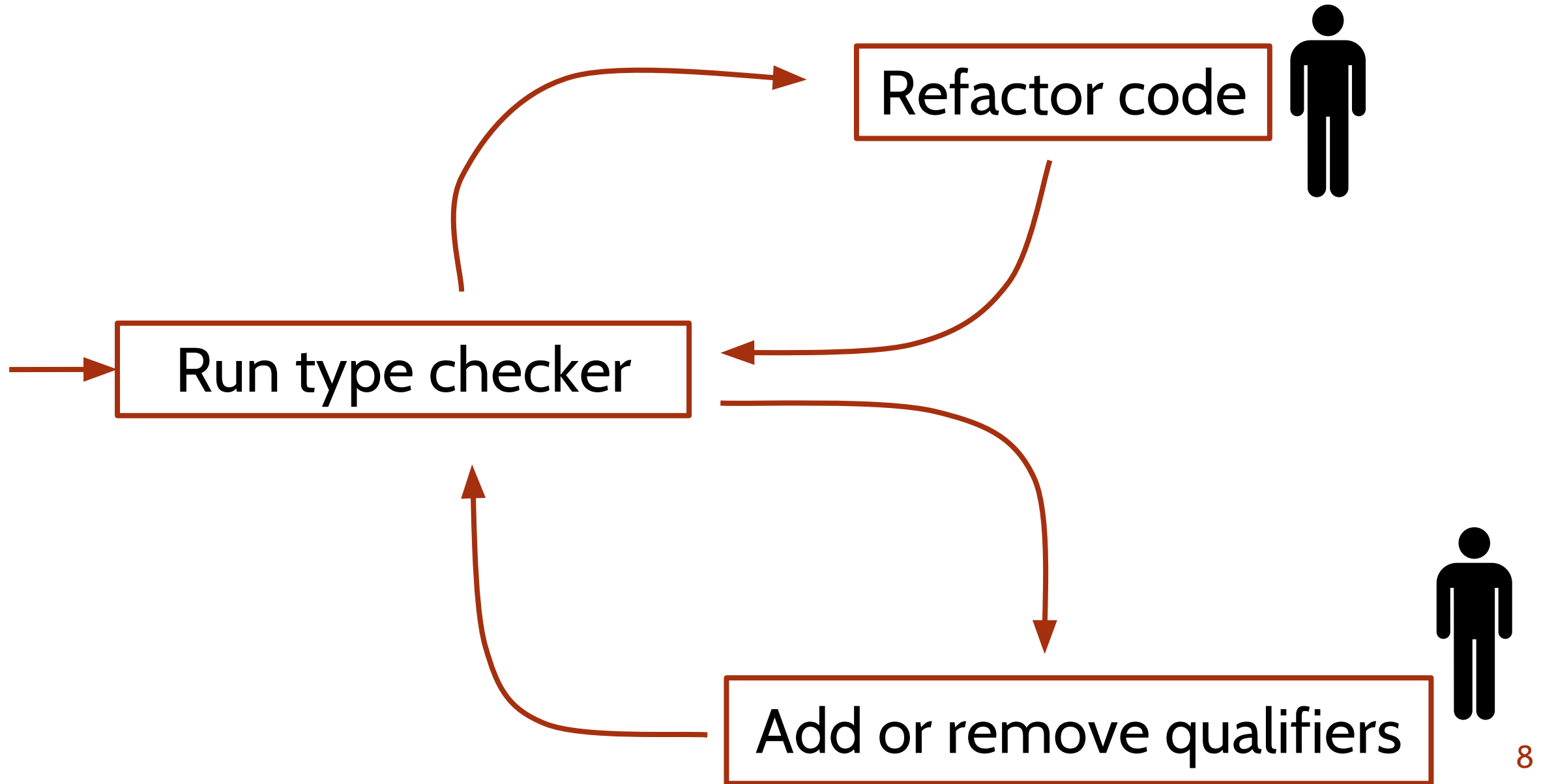
# The manual annotation process is tedious



# The manual annotation process is tedious



# The manual annotation process is tedious





# Type qualifier inference tools run in batch mode



# Type qualifier inference tools run in batch mode



## **Nullness**

Julia, Nit, JastAddJ Nullness, SALSA Nullness, Xylem, Daikon Nullness

## **Immutability**

Javarifier, Pidasai, RelmInfer

## **Ownership**

Universe and Ownership Type Inference System

# Type qualifier inference tools run in batch mode



## Strengths

- Optimal under certain conditions
- Large change without user involvement

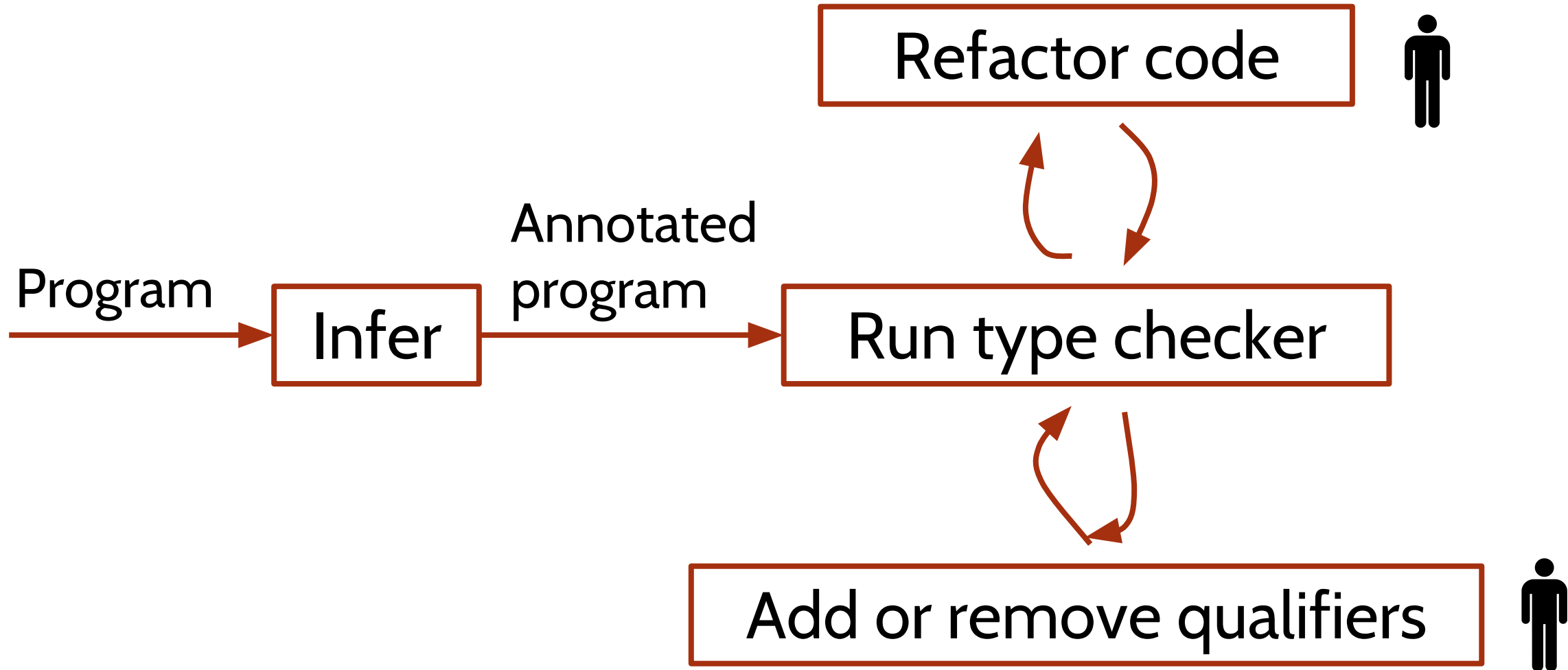
# Type qualifier inference tools run in batch mode



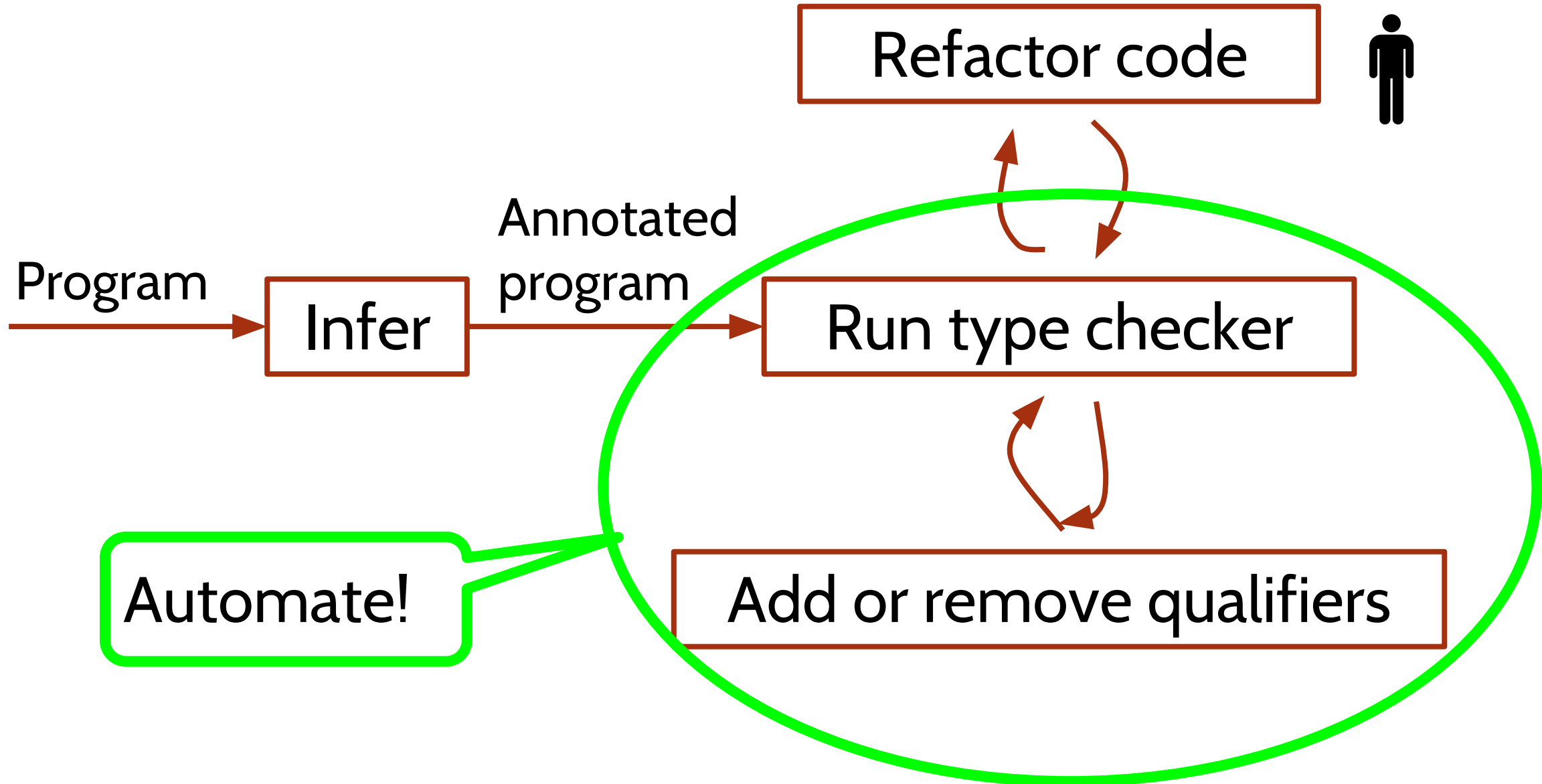
## Weaknesses

- Limited to one set of qualifiers
- Unpredictable
- Rigid
- Inaccurate

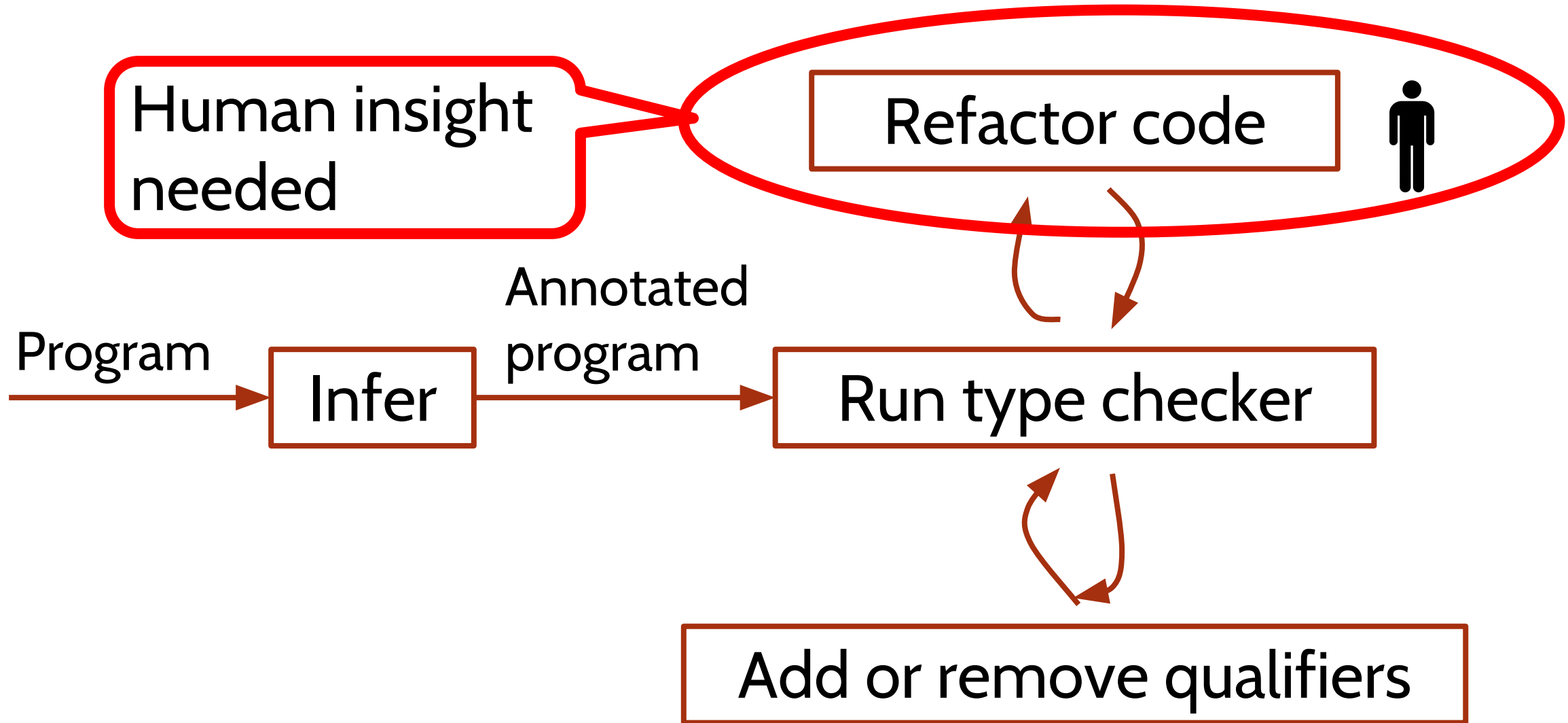
# Type qualifier inference tools run in batch mode



# Type qualifier inference tools run in batch mode



# Type qualifier inference tools run in batch mode



# Batch-mode tools make arbitrary decisions

```
1 class Tree {
2     //...
3     @Nullable Node root;
4
5     void makeTree(int nstep) {
6         for (Enumeration e = bodiesRev(); e.hasMoreElements();) {
7             Body q = (Body) e.nextElement();
8             if (q.mass != 0.0) {
9                 q.expandBox(this, nstep);
10                MathVector xqic = intcoord(q.pos);
11                if (root == null) {
12                    root = q;
13                } else {
14                    root = root.loadTree(q, xqic, Node.IMAX >> 1, this);
15                }
16            }
17        }
18        root.hackcofm();
19    }
20
21    @Nullable MathVector intcoord(MathVector vp) {
22        MathVector xp = new MathVector();
23        //..
24        xsc = (vp.value(2) - rmin.value(2)) / rsize;
25        if (0.0 <= xsc && xsc < 1.0) {
26            xp.value(2, Math.floor(Node.IMAX * xsc));
27        } else {
28            return null;
29        }
30        return xp;
31    }
32 }
```

```
33 class Node {
34     //...
35     Node loadTree(Body p, @Nullable MathVector xp, int l, Tree tree) {
36         int si = oldSubindex(xp, l);
37         //...
38     }
39
40     static int oldSubindex(@Nullable MathVector ic, int l) {
41         int i = 0;
42         for (int k = 0; k < MathVector.NDIM; k++) {
43             if (((int) ic.value(k) & 1) != 0) i += Cell.NSUB >> (k + 1);
44         }
45         return i;
46     }
47 }
```



```
9      q.expandBox(this, nstep);
10     MathVector xqic = intcoord(q.pos);
11     if (root == null) {
12         root = q;
13     } else {
14         root = root.loadTree(q, xqic, Node.IMAX >> 1, this);
15     }
16 }
17 }
18 root.hackcofm();
19 }
```

```
21 @Nullable MathVector intcoord(MathVector vp) {
22     MathVector xp = new MathVector();
23     //..
24     xsc = (vp.value(2) - rmin.value(2)) / rsize;
25     if (0.0 <= xsc && xsc < 1.0) {
26         xp.value(2, Math.floor(Node.IMAX * xsc));
27     } else {
28         return null;
29     }
```

```
33 class Node {
34     //...
35     Node loadTree(Body p, @Nullable MathVector xpic, int l, Tree tree) {
36         int si = oldSubindex(xpic, l);
37         //...
38     }
39
40     static int oldSubindex(@Nullable MathVector ic, int l) {
41         int i = 0;
42         for (int k = 0; k < MathVector.NDIM; k++) {
43             if (((int) ic.value(k) & 1) != 0) i += Cell.NSUB >> (k + 1);
44         }
45         return i;
46     }
47 }
```

**BUG**

# Batch-mode tools make arbitrary decisions

```
1 class Tree {
2   //...
3   @Nullable Node root;
4
5   void makeTree(int nstep) {
6     for (Enumeration e = bodiesRev(); e.hasMoreElements();) {
7       Body q = (Body) e.nextElement();
8       if (q.mass != 0.0) {
9         q.expandBox(this, nstep);
10        MathVector xqic = intcoord(q.pos);
11        if (root == null) {
12          root = q;
13        } else {
14          root = root.loadTree(q, xqic, Node.IMAX >> 1, this);
15        }
16      }
17    }
18    root.hackcofm();
19  }
20
21  @Nullable MathVector intcoord(MathVector vp) {
22    MathVector xp = new MathVector();
23    //...
24    xsc = (vp.value(2) - rmin.value(2)) / rsize;
25    if (0.0 <= xsc && xsc < 1.0) {
26      xp.value(2, Math.floor(Node.IMAX * xsc));
27    } else {
28      return null;
29    }
30    return xp;
31  }
32 }
```

```
33 class Node {
34   //...
35   Node loadTree(Body b, @Nullable MathVector xp, int l, Tree tree) {
36     int si = oldSubindex(xp, l);
37     //...
38   }
39
40   static int oldSubindex(@Nullable MathVector ic, int l) {
41     int i = 0;
42     for (int k = 0; k < MathVector.NDIM; k++) {
43       if (((int) ic.value(k) & 1) != 0) i += Cell.NSUB >> (k + 1);
44     }
45     return i;
46   }
47 }
```

**BUG**

# Type qualifier inference is a refactoring

- Adding type qualifiers preserves the program behavior
- Adding maintainable type qualifiers that match the programmer's intention requires code refactoring

# Cascade: A Universal Programmer-assisted Type Qualifier Inference Tool



# Cascade: A Universal Programmer-assisted Type Qualifier Inference Tool



# Inferring Primitive Changes from Error Messages

## Type Checker Error

## Fix

## Nullness

```
incompatible types in argument.  
  root = root.loadTree(q, xqic);  
                        ^  
found    : @Nullable MathVector  
required: @NonNull MathVector
```

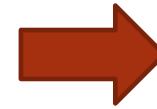
# Inferring Primitive Changes from Error Messages

## Type Checker Error

## Fix

### Nullness

```
incompatible types in argument.  
  root = root.loadTree(q, xpic);  
                        ^  
found    : @Nullable MathVector  
required: @NonNull MathVector
```



```
Change parameter xpic  
of loadTree() to  
@Nullable MathVector
```



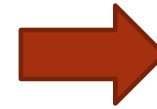
# Inferring Primitive Changes from Error Messages

## Type Checker Error

## Fix

### Nullness

```
incompatible types in argument.  
    root = root.loadTree(q, xpic);  
                          ^  
found    : @Nullable MathVector  
required: @NonNull MathVector
```



```
Change parameter xpic  
of loadTree() to  
@Nullable MathVector
```

### Mutability

```
call to value(int) not allowed  
on the given receiver.  
    ic.value(k);  
          ^  
found    : @ReadOnly MathVector  
required: @Mutable MathVector
```

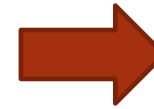
# Inferring Primitive Changes from Error Messages

## Type Checker Error

## Fix

### Nullness

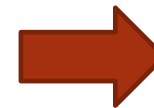
```
incompatible types in argument.  
    root = root.loadTree(q, xpic);  
                          ^  
found    : @Nullable MathVector  
required: @NonNull MathVector
```



```
Change parameter xpic  
of loadTree() to  
@Nullable MathVector
```

### Mutability

```
call to value(int) not allowed  
on the given receiver.  
    ic.value(k);  
           ^  
found    : @ReadOnly MathVector  
required: @Mutable MathVector
```




```
Change receiver parameter  
of value() to  
@ReadOnly MathVector
```

# Speculative analysis suggests a change composition


```
incompatible types in argument.  
this(1, null, null);  
found : null  
required: @Initialized @NonNull TreeNode
```


# Speculative analysis suggests a change composition


incompatible types in argument.  
this(1, null, null);  
found : null  
required: @Initialized @NonNull TreeNode

▼  Change parameter 1 to @Nullable TreeNode (1)

# Speculative analysis suggests a change composition


incompatible types in argument.  
▼  this(1, null, null);  
found : null  
required: @Initialized @NonNull TreeNode

▼  Change parameter l to @Nullable TreeNode (1)


incompatible types in assignment.  
▼  left = l;  
found : @Initialized @Nullable TreeNode  
required: @Initialized @NonNull TreeNode

# Speculative analysis suggests a change composition

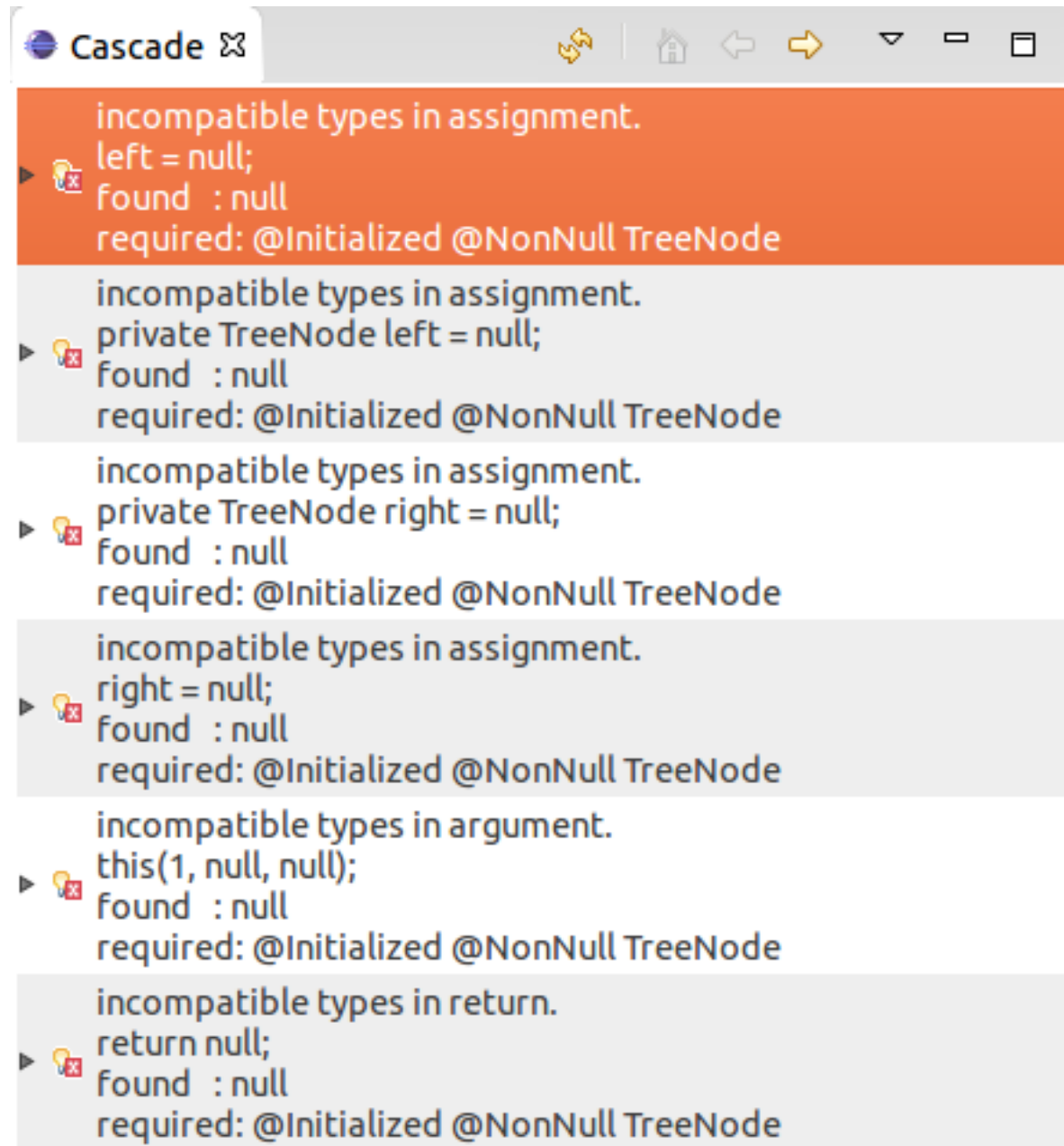
incompatible types in argument.  
this(1, null, null);  
found : null  
required: @Initialized @NonNull TreeNode

▼  Change parameter l to @Nullable TreeNode (1)

incompatible types in assignment.  
left = l;  
found : @Initialized @Nullable TreeNode  
required: @Initialized @NonNull TreeNode

 Change field left to @Nullable TreeNode (1)

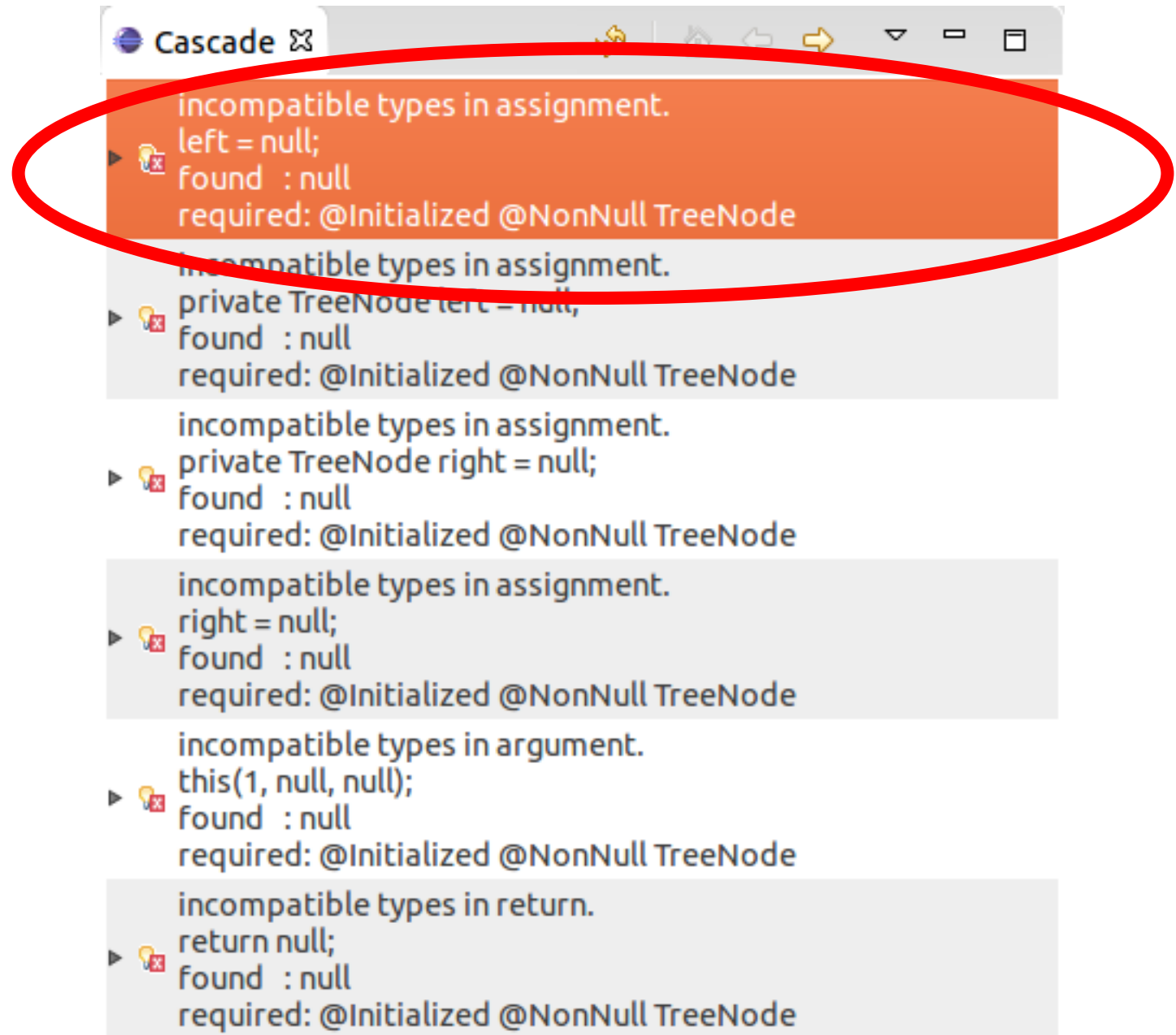
# Cascade Tree



The screenshot shows a window titled "Cascade" with several compilation errors. Each error message is preceded by a lightbulb icon with a red 'x' and a right-pointing triangle. The errors are as follows:

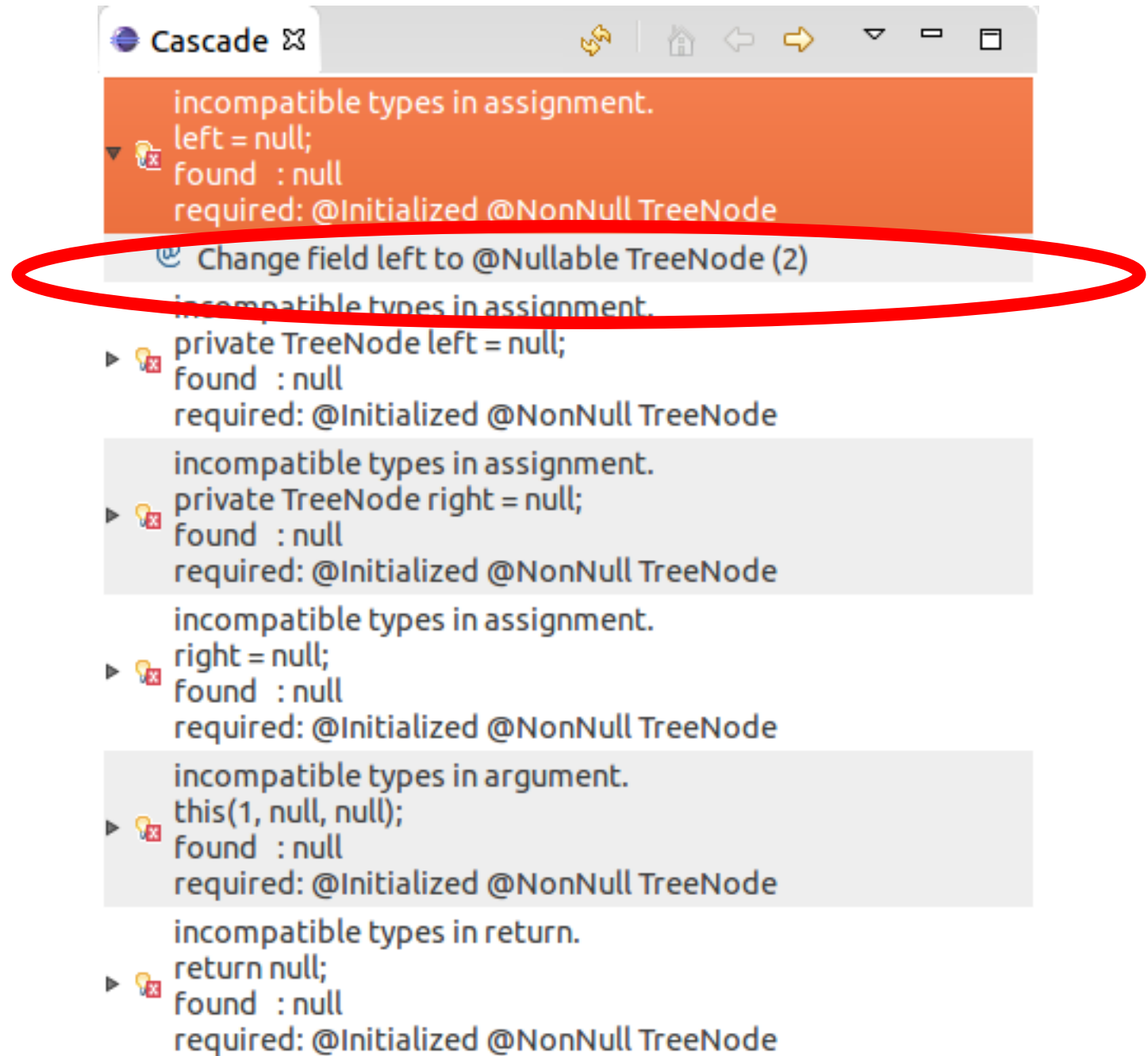
- incompatible types in assignment.**  
left = null;  
found : null  
required: @Initialized @NonNull TreeNode
- incompatible types in assignment.**  
private TreeNode left = null;  
found : null  
required: @Initialized @NonNull TreeNode
- incompatible types in assignment.**  
private TreeNode right = null;  
found : null  
required: @Initialized @NonNull TreeNode
- incompatible types in assignment.**  
right = null;  
found : null  
required: @Initialized @NonNull TreeNode
- incompatible types in argument.**  
this(1, null, null);  
found : null  
required: @Initialized @NonNull TreeNode
- incompatible types in return.**  
return null;  
found : null  
required: @Initialized @NonNull TreeNode

# Cascade Tree





# Cascade Tree



Cascade

incompatible types in assignment.  
left = null;  
found : null  
required: @Initialized @NonNull TreeNode

Change field left to @Nullable TreeNode (2)

incompatible types in assignment.  
private TreeNode left = null;  
found : null  
required: @Initialized @NonNull TreeNode

incompatible types in assignment.  
private TreeNode right = null;  
found : null  
required: @Initialized @NonNull TreeNode

incompatible types in assignment.  
right = null;  
found : null  
required: @Initialized @NonNull TreeNode

incompatible types in argument.  
this(1, null, null);  
found : null  
required: @Initialized @NonNull TreeNode

incompatible types in return.  
return null;  
found : null  
required: @Initialized @NonNull TreeNode

```

    * Create a tree node given the two children. The initial node value is 1.
    **/
    public TreeNode() {
        this(1, null, null);
    }

    /**
     * Construct a subtree with the specified number of levels. We recursively call the c
     * create the tree.
     *
     * @param levels the number of levels in the subtree
     **/
    public TreeNode(int levels) {
        value = 1;
        if (levels <= 1) {
            if (levels <= 0) throw new RuntimeException("Number of levels must be positive no
            left = null;
            right = null;
        } else {
            left = new TreeNode(levels - 1);
            right = new TreeNode(levels - 1);
        }
    }

    /**
     * Set the children of the tree
     *
     * @param l the left child
     * @param r the right child
     **/
    public void setChildren(TreeNode l, TreeNode r) {
        left = l;
        right = r;
    }

```

incompatible types in assignment.

left = null;  
found : null  
required: @Initialized @NonNull TreeNode

incompatible types in assignment.

private TreeNode left = null;  
found : null  
required: @Initialized @NonNull TreeNode

incompatible types in assignment.

private TreeNode right = null;  
found : null  
required: @Initialized @NonNull TreeNode

incompatible types in assignment.

right = null;  
found : null  
required: @Initialized @NonNull TreeNode

incompatible types in argument.

this(1, null, null);  
found : null  
required: @Initialized @NonNull TreeNode

incompatible types in return.

return null;  
found : null  
required: @Initialized @NonNull TreeNode

```

package treeadd;
/**
 * A Tree node data structure.
 */
public class TreeNode {
    private int value = 0;
    private TreeNode left = null;
    private TreeNode right = null;

    /**
     * Create a node in the tree with a given value and two children.
     *
     * @param v the node's value
     * @param l the left child.
     * @param r the right child.
     */
    public TreeNode(int v, TreeNode l, TreeNode r) {
        value = v;
        left = l;
        right = r;
    }

    /**
     * Create a tree node given the two children. The initial node value is 1.
     */
    public TreeNode(TreeNode l, TreeNode r) {
        this(1, l, r);
    }

    /**
     * Create a tree node given the two children. The initial node value is 1.
     */
    public TreeNode() {
        this(1, null, null);
    }
}

```

- incompatible types in assignment.
  - left = null;
    - found : null
    - required: @Initialized @NonNull TreeNode
  - Change field left to @Nullable TreeNode (2)
- incompatible types in assignment.
  - private TreeNode left = null;
    - found : null
    - required: @Initialized @NonNull TreeNode
- incompatible types in assignment.
  - private TreeNode right = null;
    - found : null
    - required: @Initialized @NonNull TreeNode
- incompatible types in assignment.
  - right = null;
    - found : null
    - required: @Initialized @NonNull TreeNode
- incompatible types in argument.
  - this(1, null, null);
    - found : null
    - required: @Initialized @NonNull TreeNode
- incompatible types in return.
  - return null;
    - found : null
    - required: @Initialized @NonNull TreeNode

```
/**
 * A Tree node data structure.
 */
public class TreeNode {
    private int value = 0;
    private @Nullable TreeNode left = null;
    private TreeNode right = null;

    /**
     * Create a node in the tree with a given value and two children.
     *
     * @param v the node's value
     * @param l the left child.
     * @param r the right child.
     */
    public TreeNode(int v, TreeNode l, TreeNode r) {
        value = v;
        left = l;
        right = r;
    }

    /**
     * Create a tree node given the two children. The initial node value is 1.
     */
    public TreeNode(TreeNode l, TreeNode r) {
        this(1, l, r);
    }

    /**
     * Create a tree node given the two children. The initial node value is 1.
     */
    public TreeNode() {
        this(1, null, null);
    }
}
```

incompatible types in assignment.

left = null;

found : null

required: @Initialized @NonNull TreeNode

Change field left to @Nullable TreeNode (2)

incompatible types in assignment.

private TreeNode left = null;

found : null

required: @Initialized @NonNull TreeNode

incompatible types in assignment.

private TreeNode right = null;

found : null

required: @Initialized @NonNull TreeNode

incompatible types in assignment.

right = null;

found : null

required: @Initialized @NonNull TreeNode

incompatible types in argument.

this(1, null, null);

found : null

required: @Initialized @NonNull TreeNode

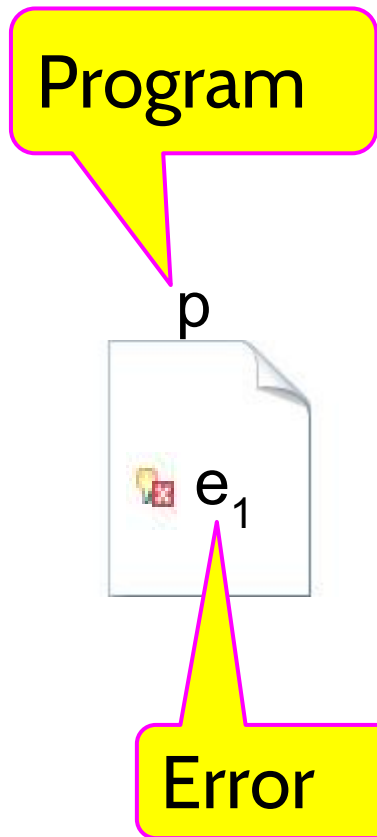
incompatible types in return.

return null;

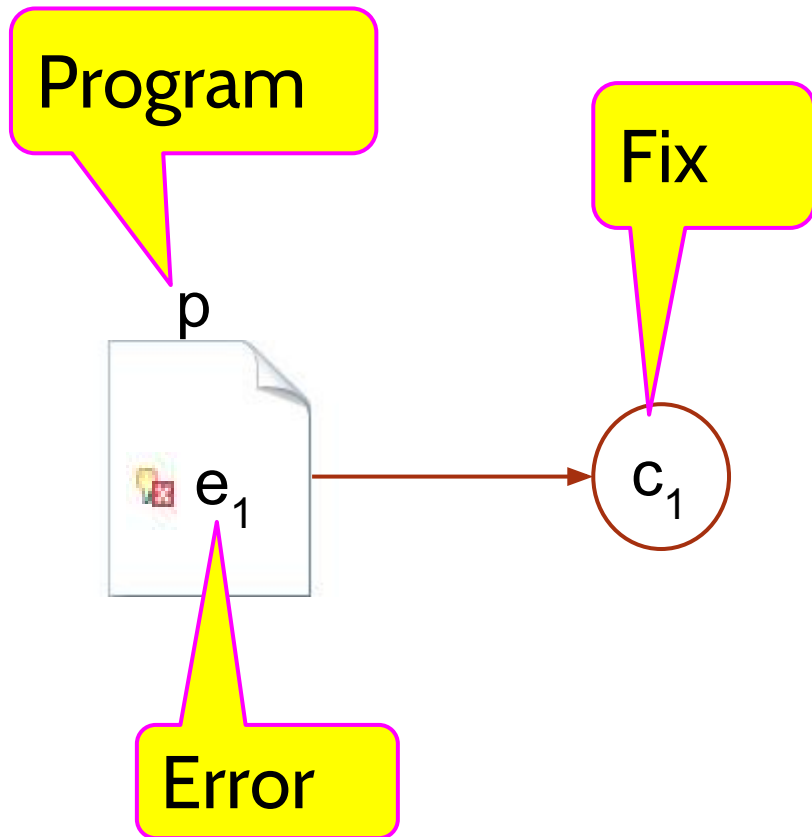
found : null

required: @Initialized @NonNull TreeNode

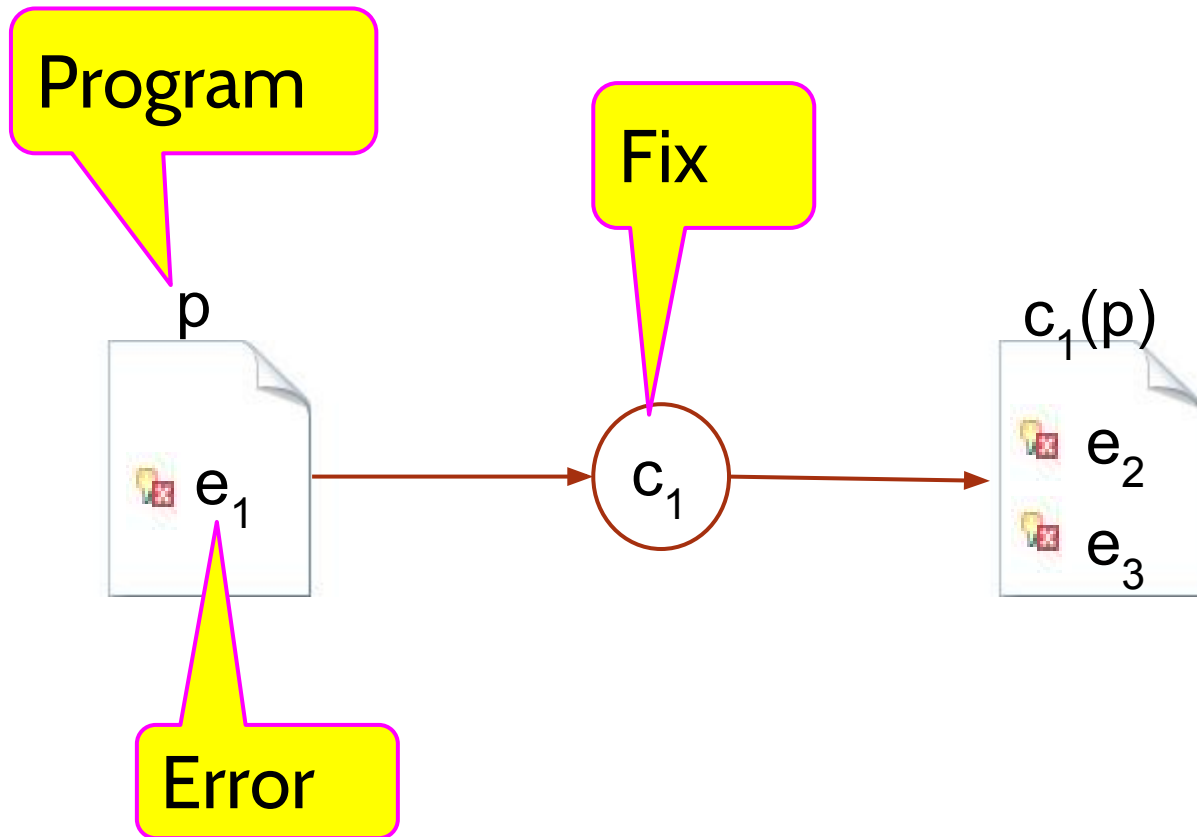
# The speculative analysis computes a tree recursively



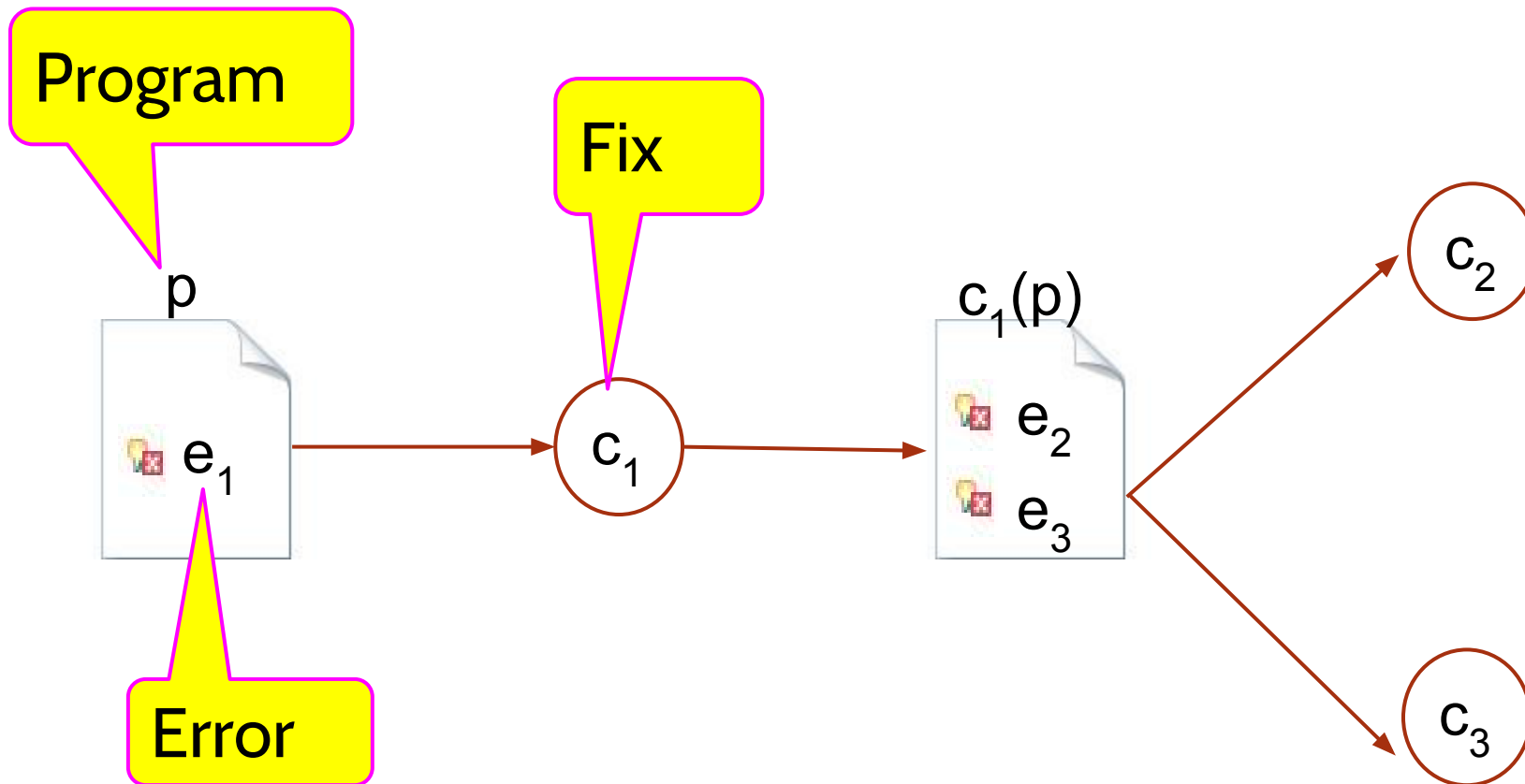
# The speculative analysis computes a tree recursively



# The speculative analysis computes a tree recursively

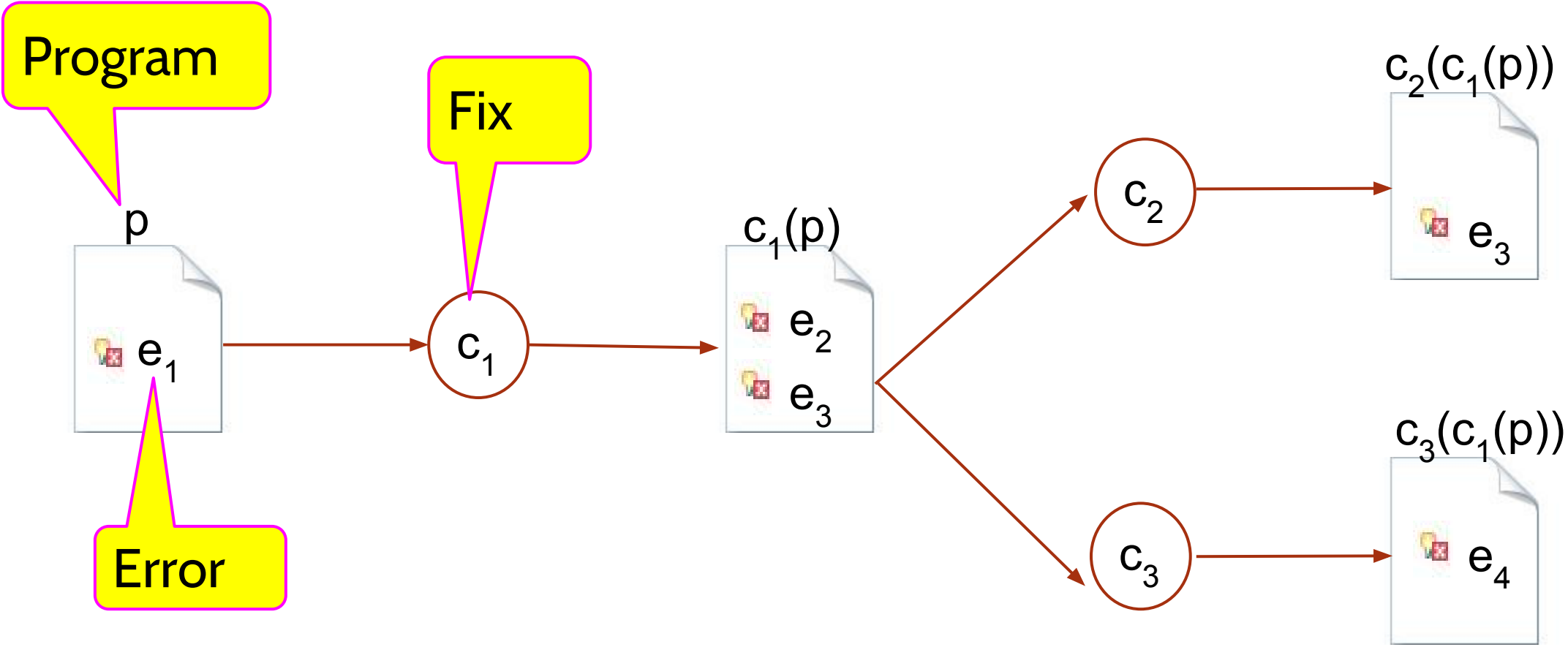


# The speculative analysis computes a tree recursively





# The speculative analysis computes a tree recursively



# A change is represented as an AST path

<b>Primitive Change</b>	<b>Representation</b>
Variable Decl. Fixer	Compilation Unit + Variable Decl. + New Type
Method Return Fixer	Compilation Unit + Method Decl. + New Type
Method Receiver Fixer	Compilation Unit + Method Decl. + New Type

# Research Questions

How does **Cascade** compare with **Julia**, a batch qualifier inference tool?

- Learnability
- Quality of results
- Task completion time
- Control over process
- Willingness to use

# User study

## Subjects:

- 12 computer science graduate students from 9 different research labs
- Familiar with Java and Eclipse
- Average of 10 years of programming experience

# Training

## ■ Nullness Checker

## ■ Julia

## ■ Cascade

[github.com/reprogrammer/tqi-study](https://github.com/reprogrammer/tqi-study)

Nullness Checker Tutorial  
reprogrammer edited this page on May 7 · 30 revisions

### Java 8

Java 8 has added several new annotations, including the possibility to augment the use of a type. As in previous versions, the use of annotations on classes, fields, methods, and constructors is provided by the Nullness Checker.

### Exercise 1

If you run the Nullness Checker on the following piece of code, which part of the code will cause an error message and why?

```
public class TreeNode {  
    int value = 0;  
    @Nullable TreeNode left = null;  
    @Nullable TreeNode right = null;  
    //...  
    int addTree() {  
        int total = value;  
        total += left.addTree();  
        total += right.addTree();  
        return total;  
    }  
}
```

### Annotating

Java 7 already provided the `@Beta` annotation for experimental APIs. The `BloomFilter` class is an example of a class that uses annotations for declarations of experimental APIs.

### Exercise 2

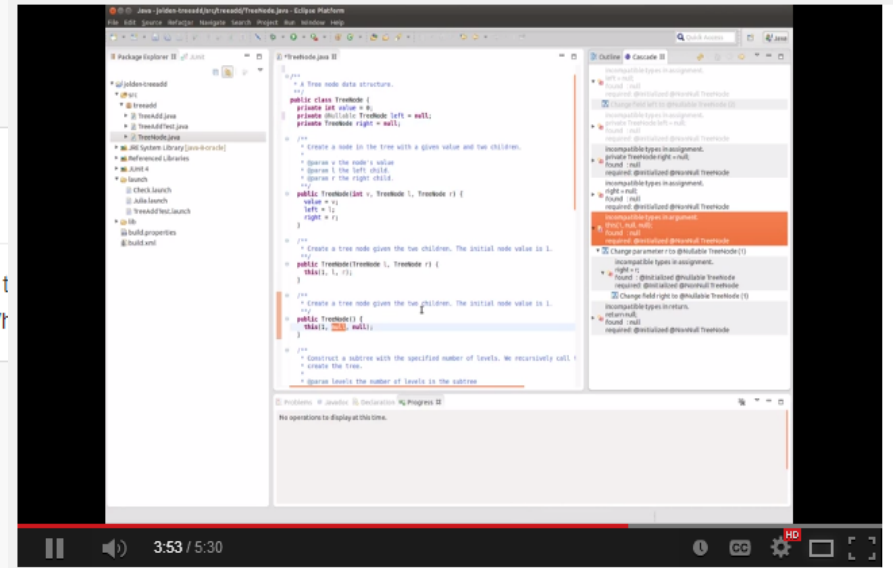
If you run the Nullness Checker on the following piece of code, what warnings are shown below the code. What do they mean?

```
class TreeNode {  
    Object data;  
    TreeNode left;  
    TreeNode right;  
  
    public TreeNode(Object data, int l, int r) {  
        this.data = data;  
        setChildren(l, r);  
    }  
  
    void setChildren(TreeNode l, TreeNode r) {
```

### Julia Tutorial

reprogrammer edited this page on May 3 · 5 revisions  
Julia is a Type Qualifier  
of code and its

Nullness annotations for a given piece of code  
outu.be/XGFJ00SFIQ



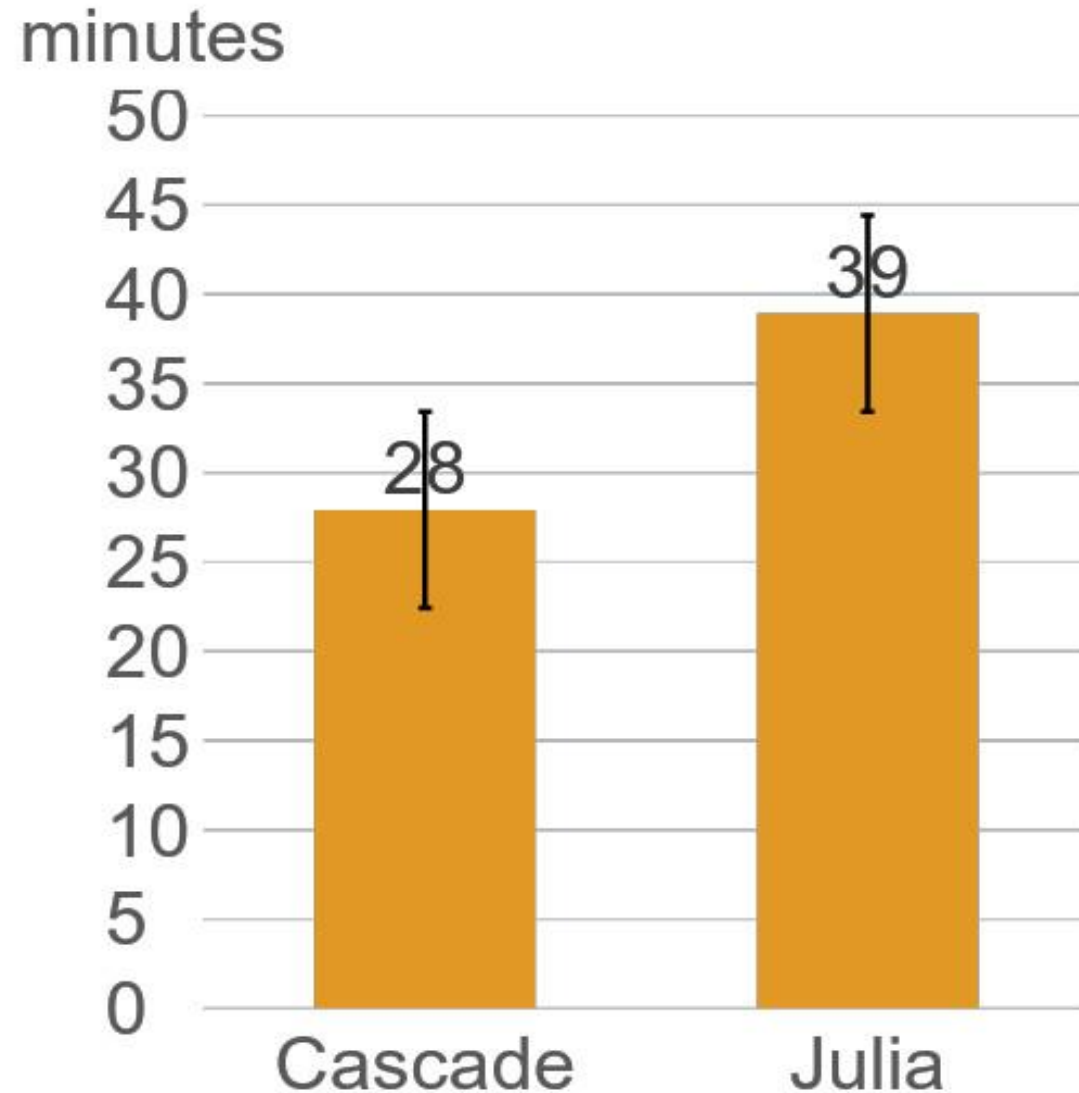
Cascade Video Tutorial

# Task Design

	<b>Julia then Cascade</b>	<b>Cascade then Julia</b>
<b>MST then BH</b>	3 participants	3 participants
<b>BH then MST</b>	3 participants	3 participants

<b>BH</b>	Barnes-Hut, a hierarchical force-calculation algorithm
<b>MST</b>	Bentley's algorithm for finding the minimum spanning tree of a graph

# Users complete tasks faster with Cascade



t test

■  $p = 0.01$

■ Cohen's  $d = 1.13$

# Users added less inaccurate annotations with Cascade

	<b>BH + MST</b>	
	<b>Julia</b>	<b>Cascade</b>
Correct	13.9	9.6
Incorrect	2.8	0.1
Redundant	1.7	0.4
Unnecessary warning suppressions	7.2	0



# Postquestionnaire Results

Questions ( <i>T = Julia or Cascade</i> )	Cascade better	Equal Rating	Julia better
I found T easy to learn.	3	6	3
I know why T inserted each annotation.	8	4	0
Using T, I have control over the process of annotating the code.	9	0	3
I'm willing to use T in the future	11	0	1

# Qualitative Interview Results

The participants believe that:

- Cascade's **speculative analysis** is useful (N = 8).
- Cascade is more **predictable** (N = 7).
- Cascade's tree computation is **slow** (N = 5).
- The **overhead** of fixing Julia's annotations is high (N = 7).

# Future Work

- Improve the **performance** of Cascade.
- Evaluate compositional refactoring and Cascade in the **field**.
- Make Cascade support **bidirectional** speculative analysis.

# Cascade: A Universal Type Qualifier Inference Tool

- Cascade is **easy to use** and helps users complete tasks **fast**.
- **Compositional refactoring** and **speculative analysis**.
- **Less is sometimes more** in the automation of software evolution tasks.
  - More automation is not always better
  - Some tasks need problem-solving and creativity
  - Applicable to other fields