

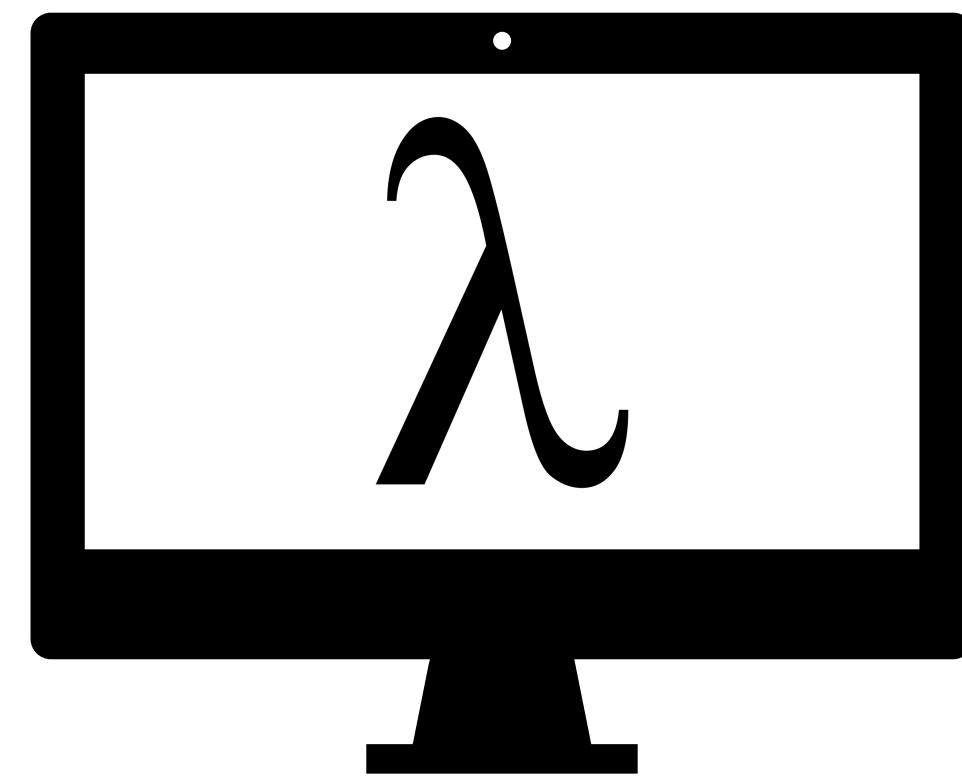
# **IS593: Language-based Security**

## **12. Static Analysis with AI**

Kihong Heo

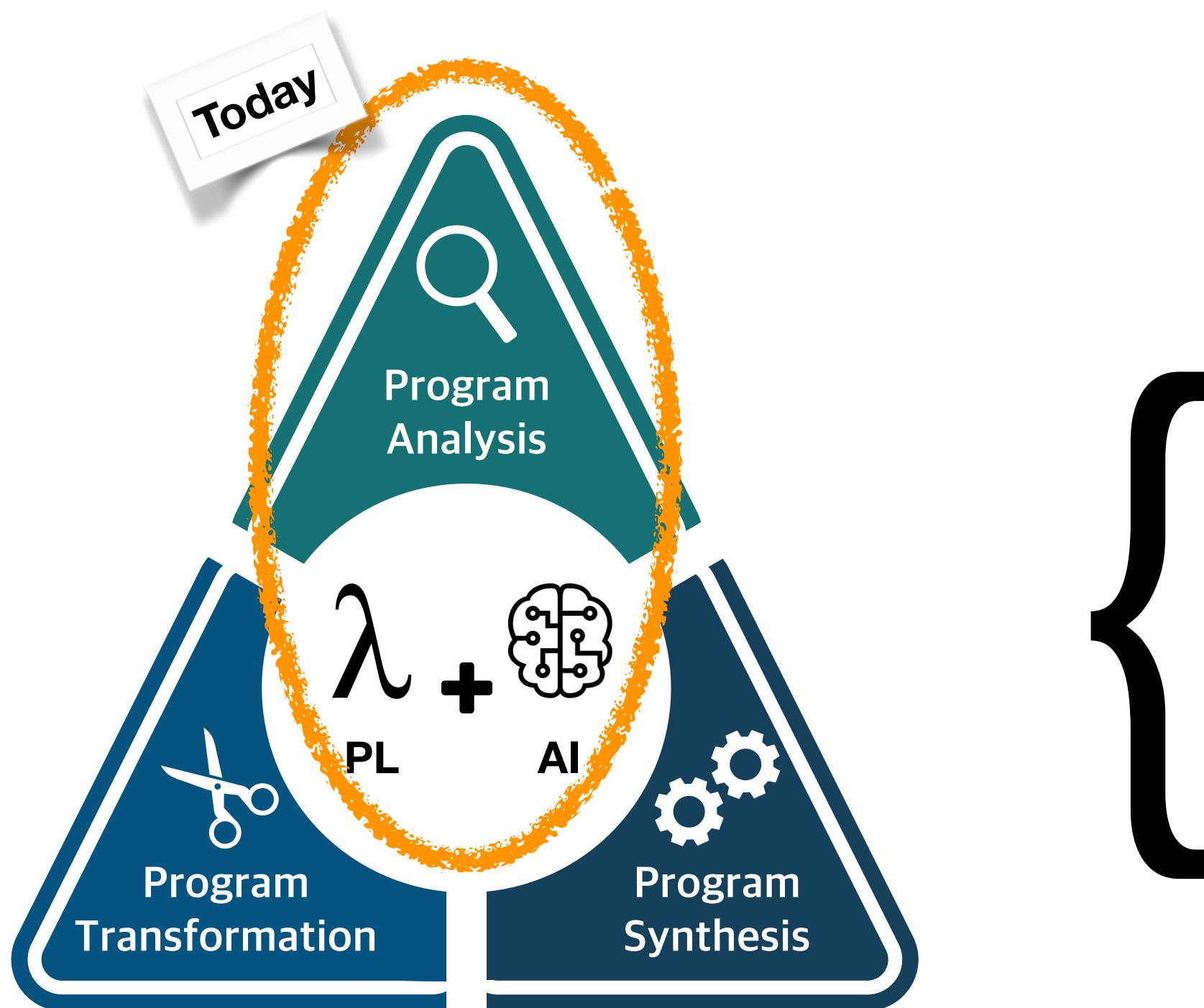


# New Waves in Language-based Security

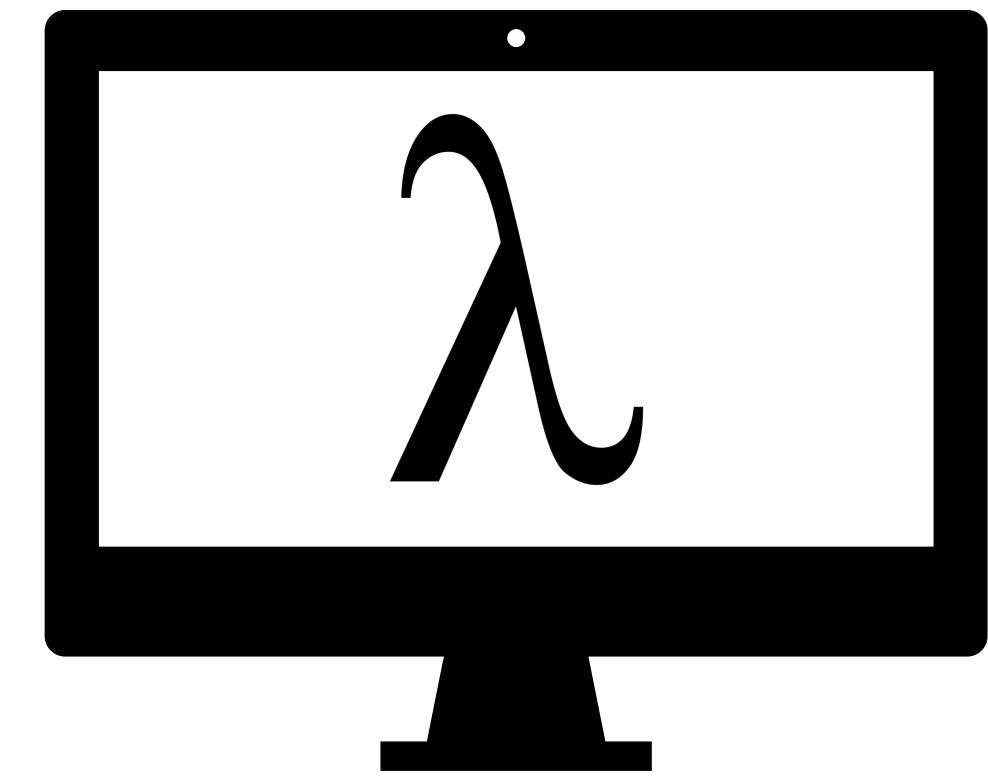


Next-generation  
Programming Systems

# New Waves in Language-based Security



- {
-  Safe
  -  Simple
  -  Smart
- Next-generation  
Programming Systems



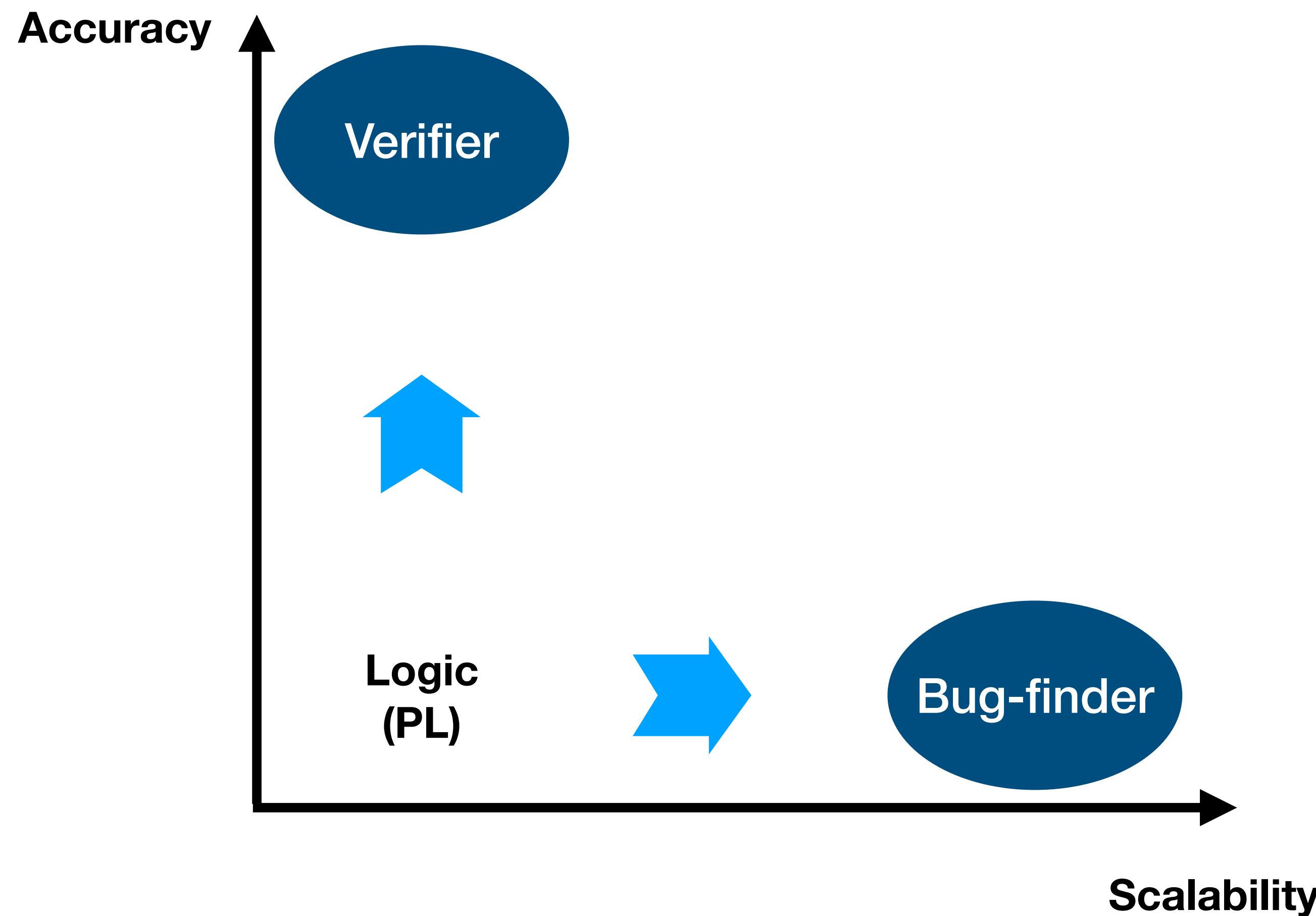
# Challenge in Static Analysis



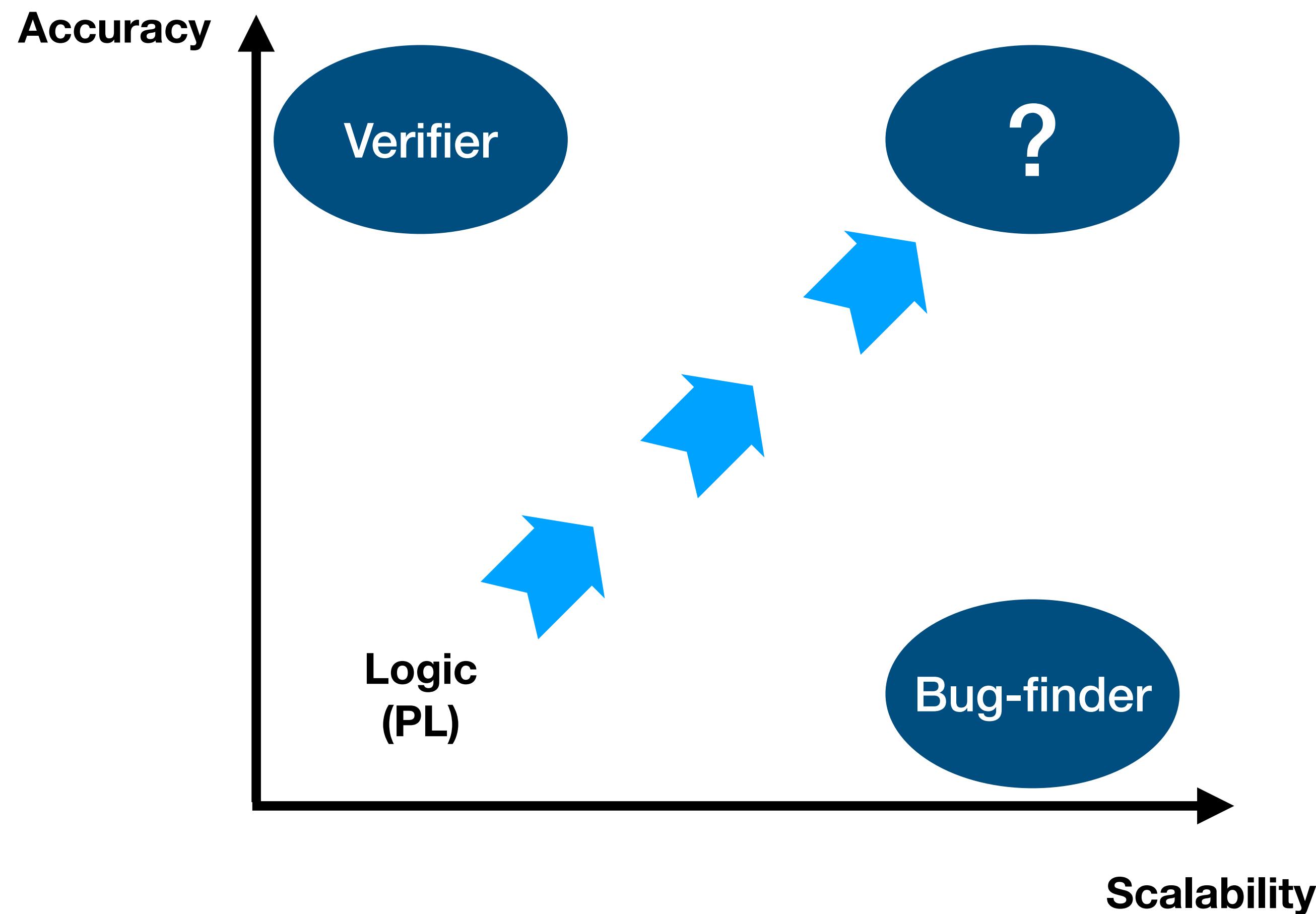
“... can be difficult to do without introducing large numbers of **false positives**, or scaling **performance** exponentially poorly. In this case, **balancing** these ... caused us to **miss the defect.**”

— *On Detecting Heartbleed with Static Analysis, (Coverity, 2014)*

# Conventional Static Analysis



# How to Go Beyond?



# Conventional Static Analysis

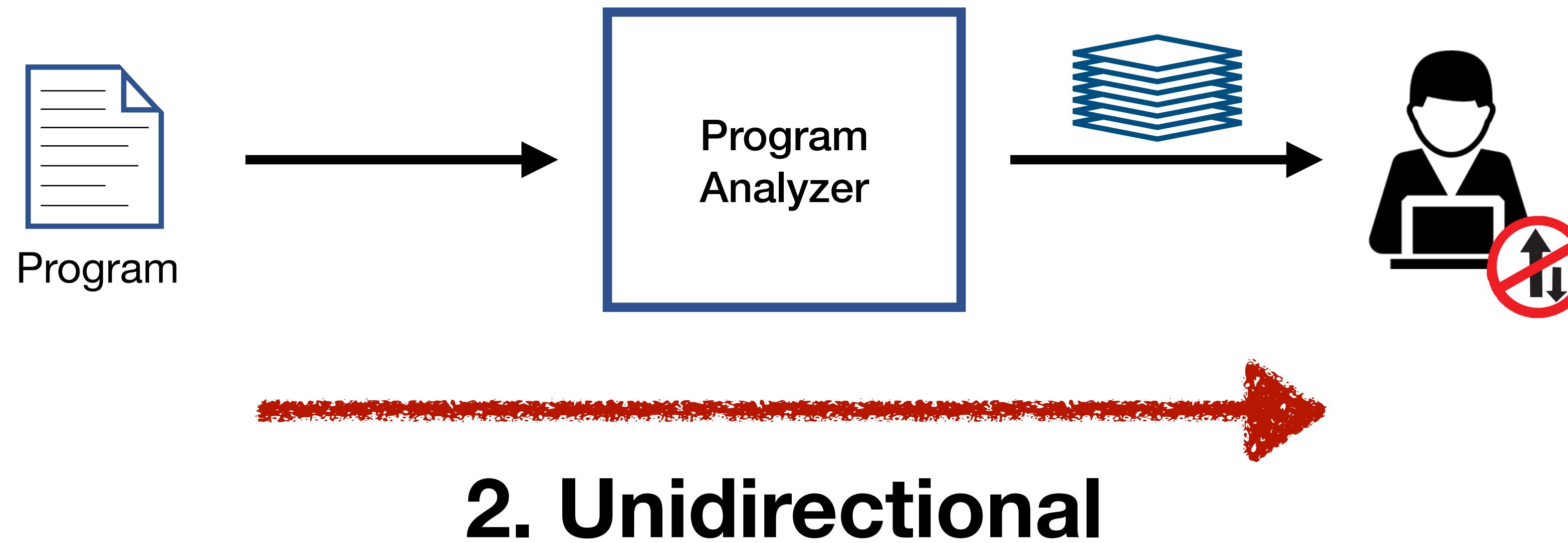


# Conventional Static Analysis

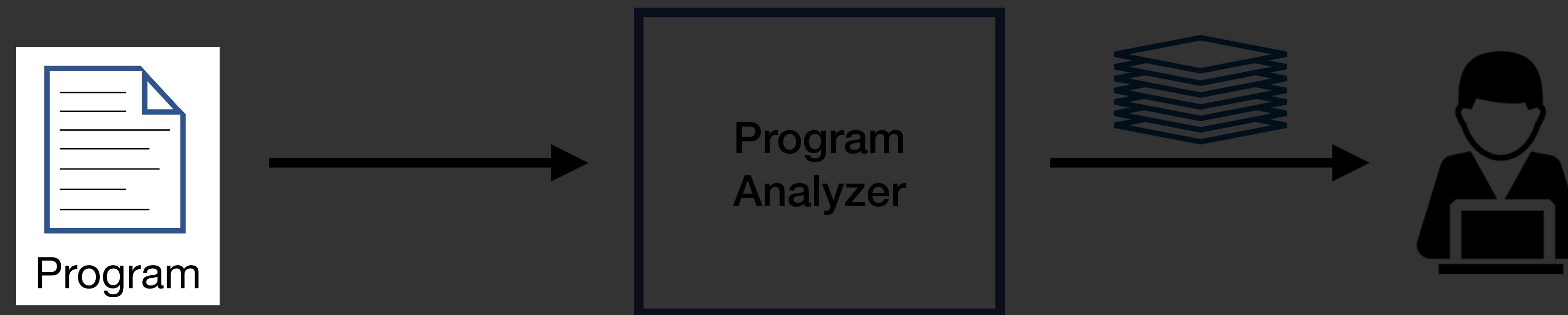


**1. Inflexible**

# Conventional Static Analysis

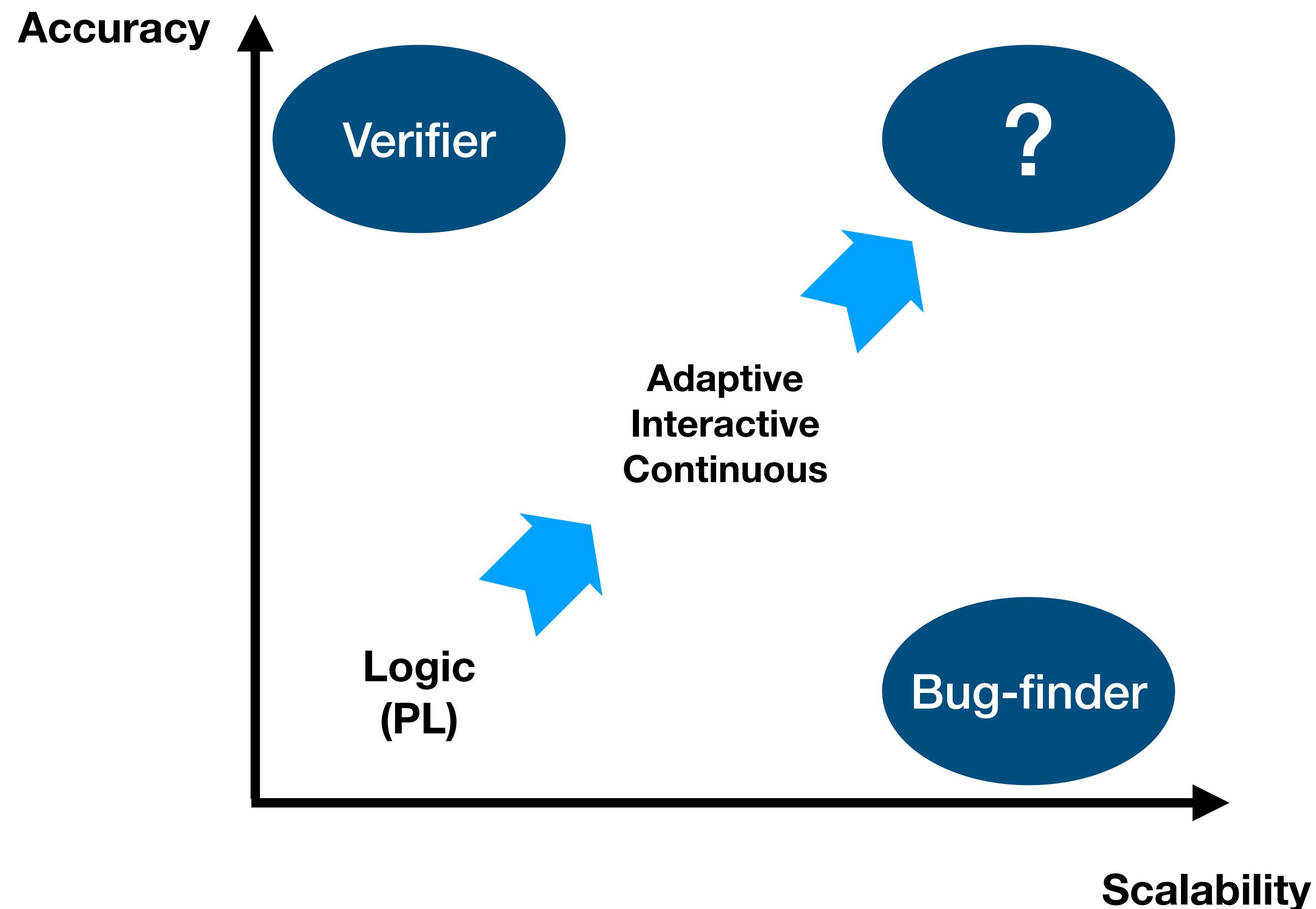


# Conventional Program Analysis

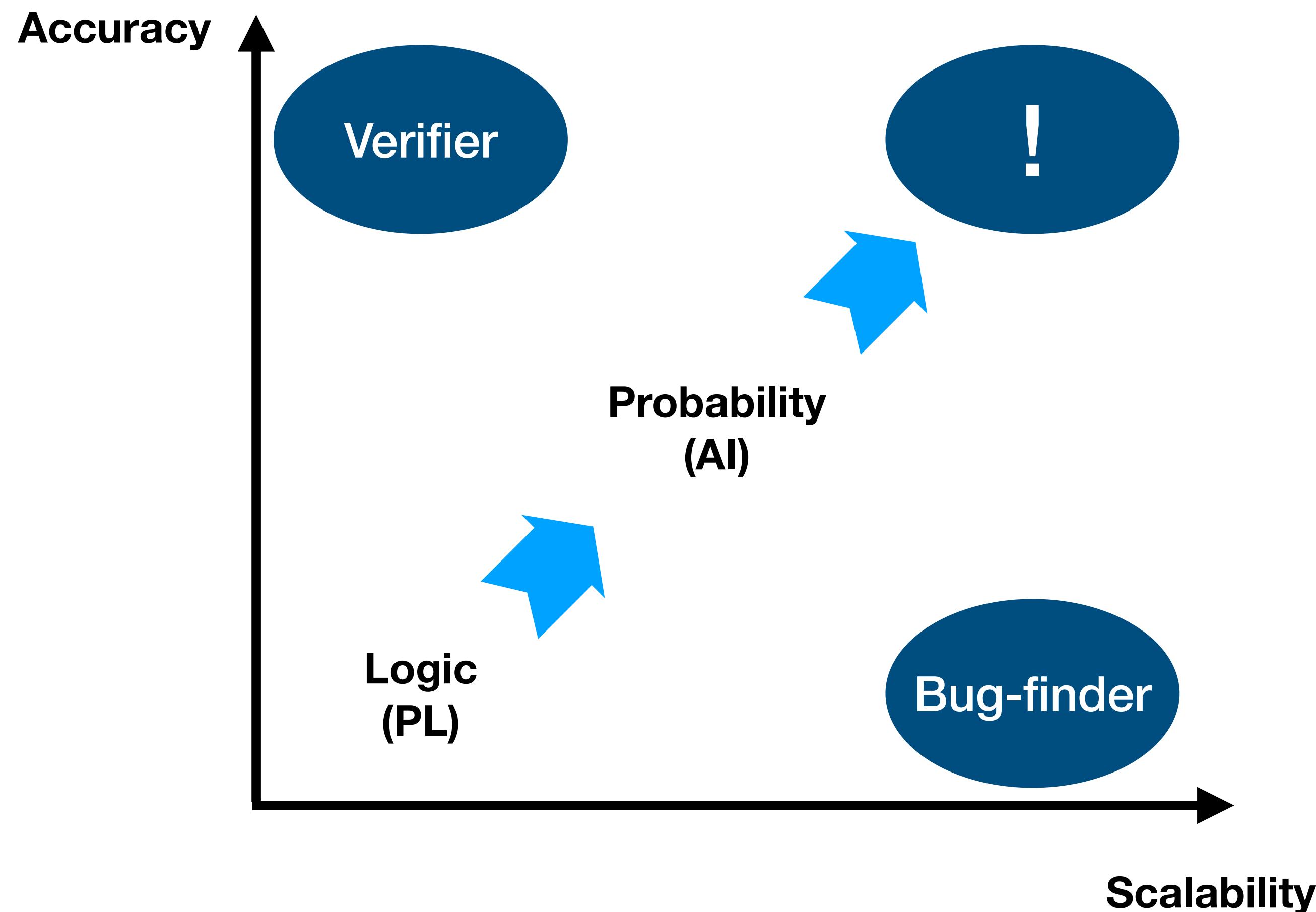


## 3. Narrow-sighted

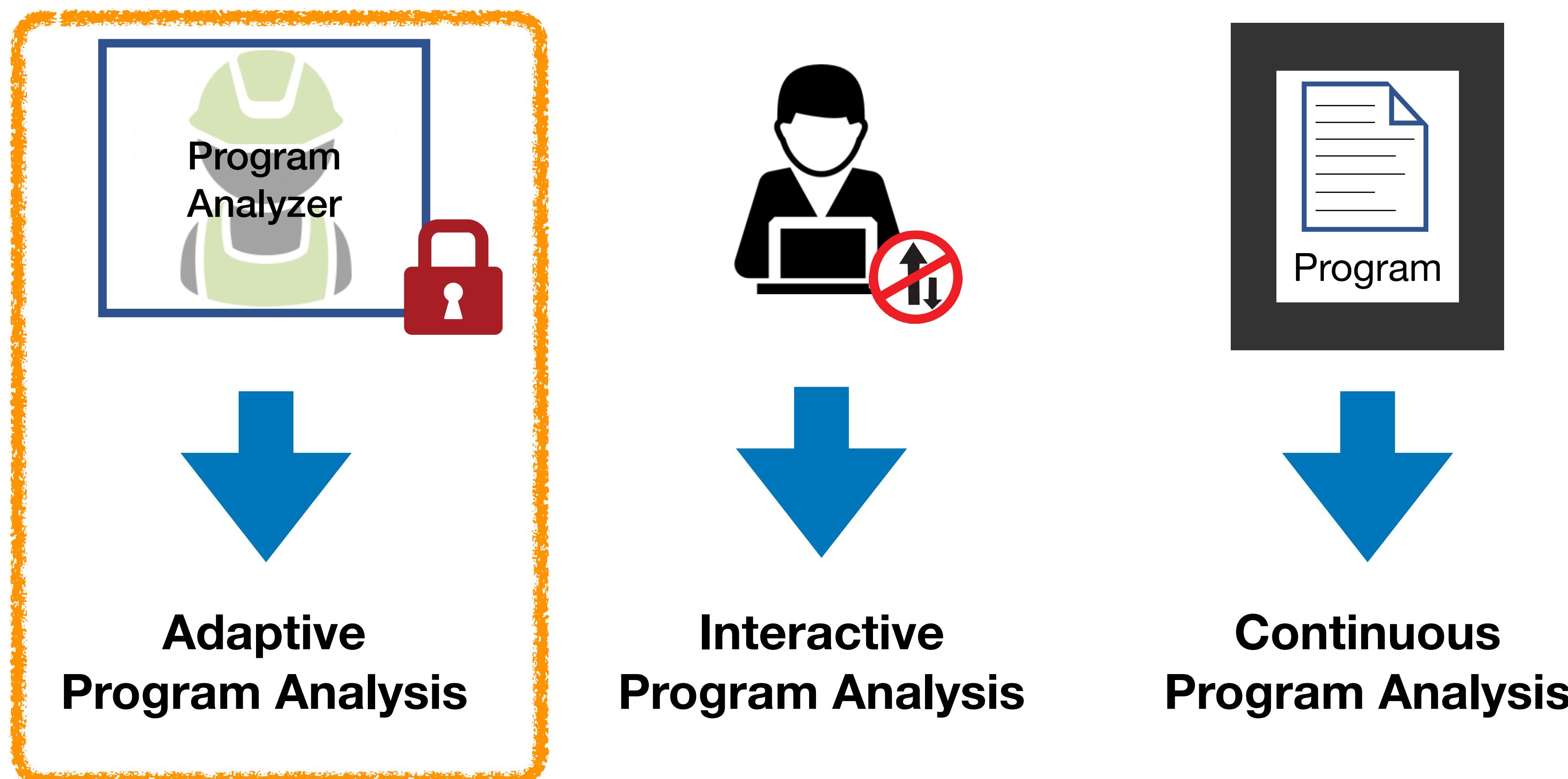
# How to Go Beyond?



# Static Analysis with AI

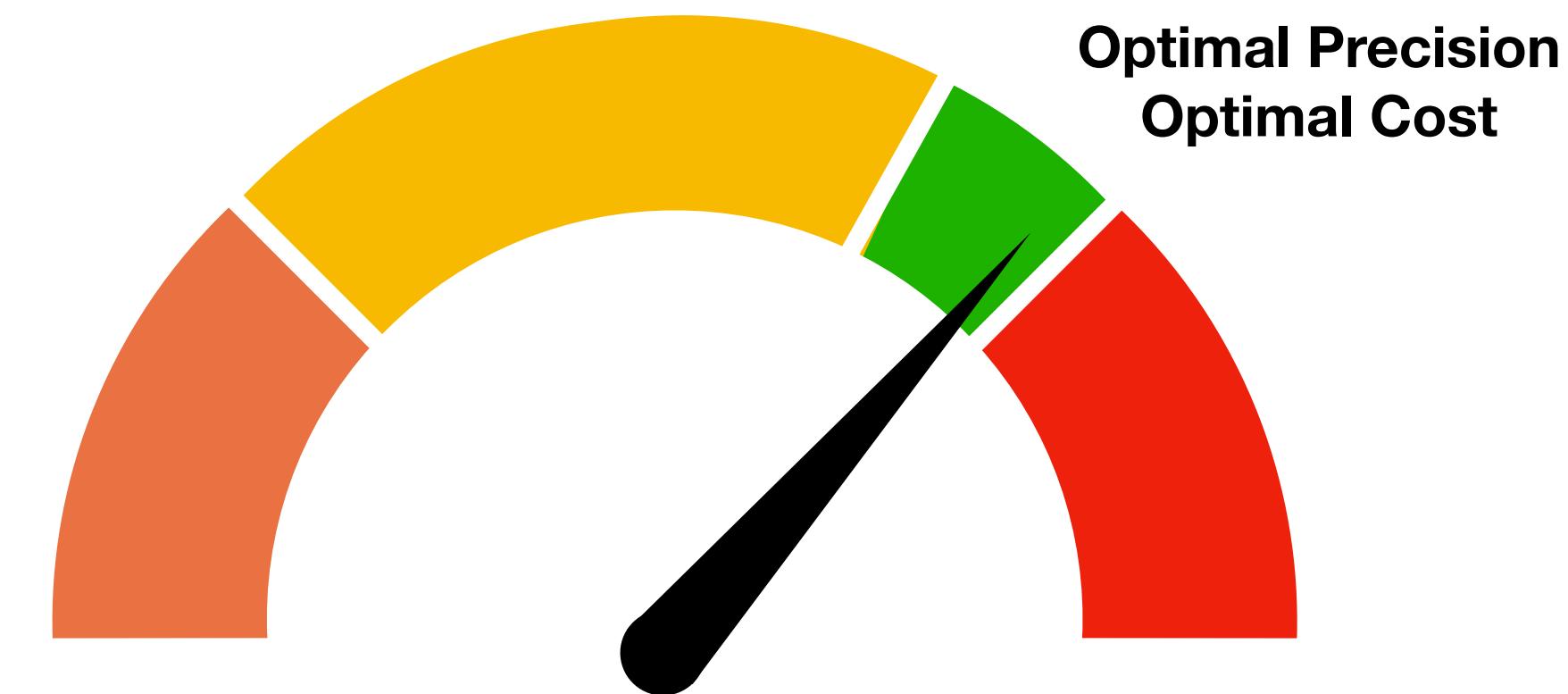


# Outline

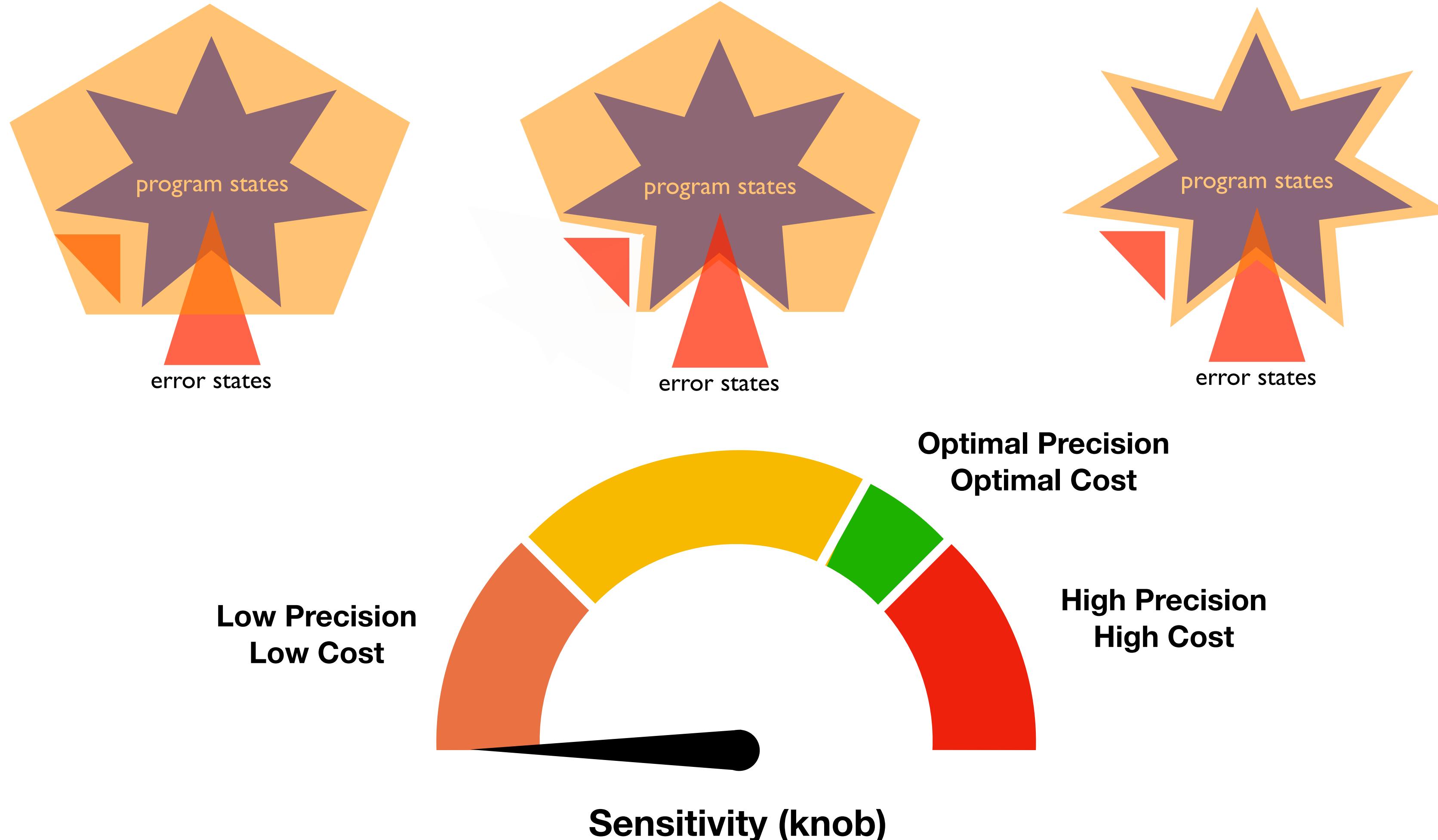


# Adaptive Program Analysis

[SAS'16, OOPSLA'17, ICSE'17, ICSE'19]



# Adaptive Program Analysis

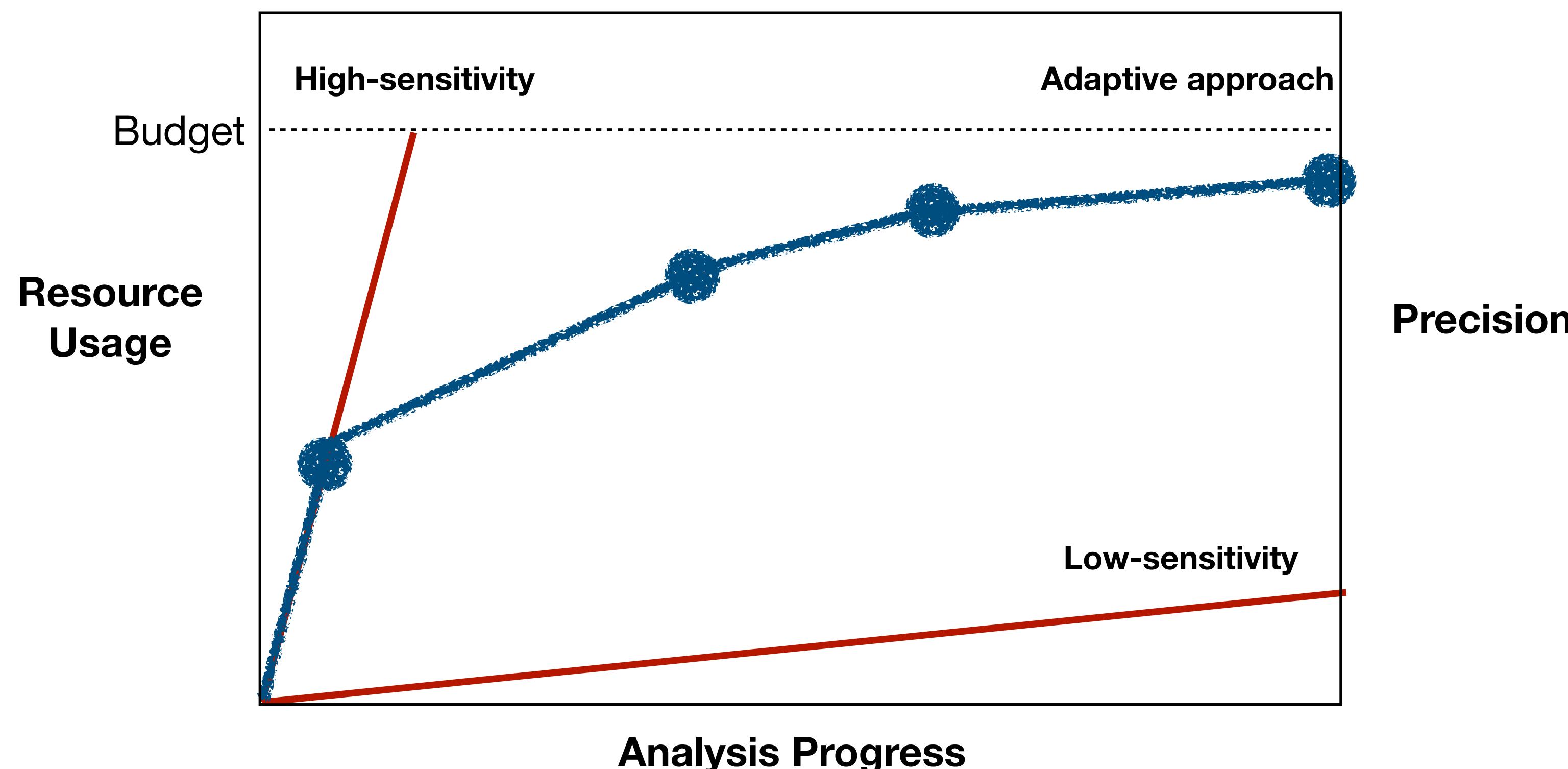


# Applications

Abstraction (Knob)	Cost	Online/Offline	Method	Result
Variable Relationship	Running Time	Offline	Supervised Learning	[SAS'16]
Statement Order Variable Relationship	Running Time	Offline	Supervised Learning	[OOPSLA'17]
Loop Unrolling Library Call Handling	Missed Bugs	Offline	Supervised Learning	[ICSE'17]
Statement Order	Memory Consumption	Online	Reinforcement Learning	[ICSE'19]

# Resource-aware Analysis

- Achieving **maximum precision** within a given **resource budget**
  - e.g., within 128GB of memory



# Knob

- Flow-sensitivity: degree of abstraction of statement order

## Flow-sensitive

```
1: x = 0;  
2: y = 1;  
3: x = 1;  
4: y = 0;
```

## Partially Flow-sensitive

```
1: x = 0;  
3: x = 1;  
2: y = 1;  
4: y = 0;
```

## Flow-insensitive

```
1: x = 0;  
3: x = 1;  
2: y = 1;  
4: y = 0;
```

Line	State
1	{x = [0,0]}
2	{x = [0,0], y = [1,1]}
3	{x = [1,1], y = [1,1]}
4	{x = [1,1], y = [0,0]}

Line	FS	FI
1	{x = [0,0]}	
2	{x = [0,0]}	
3	{x = [1,1]}	{y = [0, 1]}
4	{x = [1,1]}	

Line	State
*	{x = [0, 1], y = [0, 1]}

# Example

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);    // Query 3 (may fail)
```

# Example

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);    // Query 3 (may fail)
```

Line	Flow-Sensitive Abstract State
1	{x = [0,0], y = [0,0], z = [1,1], v = $\top$ , w = $\top$ }

**3 Intervals**

# Example

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);    // Query 3 (may fail)
```

Line	Flow-Sensitive Abstract State
1	{x = [0,0], y = [0,0], z = [1,1], v = ⊤, w = ⊤}
2	{x = [1,1], y = [0,0], z = [1,1], v = ⊤, w = ⊤}

6 Intervals

# Example

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);    // Query 3 (may fail)
```

Line	Flow-Sensitive Abstract State
1	{x = [0,0], y = [0,0], z = [1,1], v = $\top$ , w = $\top$ }
2	{x = [1,1], y = [0,0], z = [1,1], v = $\top$ , w = $\top$ }
3	{x = [1,1], y = [0,0], z = [2,2], v = $\top$ , w = $\top$ }
4	<b>x = [1,1], y = [1,1], z = [2,2], v = <math>\top</math>, w = <math>\top</math></b>

12 Intervals

# Example

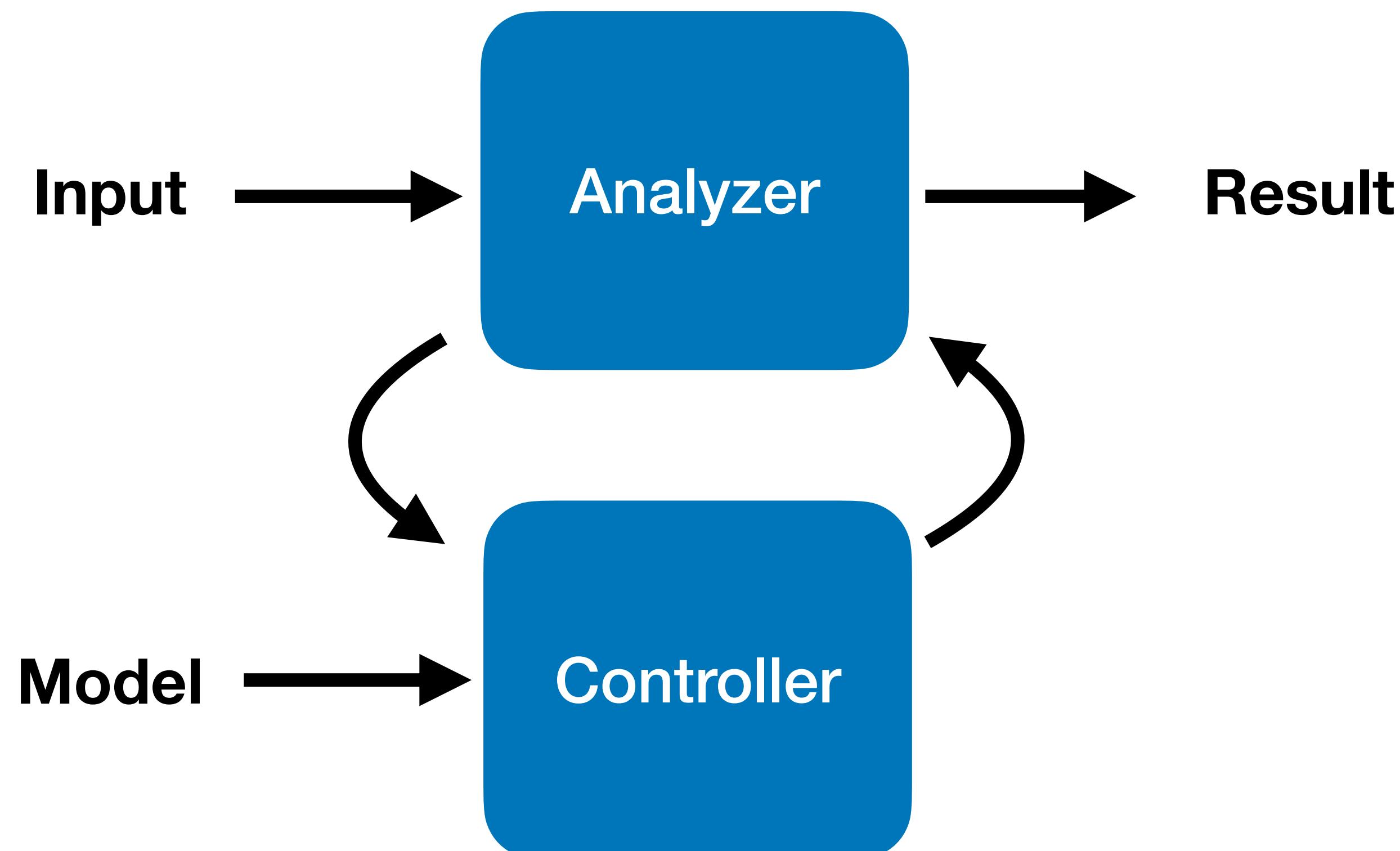
- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);    // Query 1 (hold)
6: assert(z > 0);    // Query 2 (hold)
7: assert(v == w);    // Query 3 (may fail)
```

Line	Flow-Insensitive Abstract State
*	{x = [0,+∞], y = [0,+∞], z = [1,+∞], v = ⊤, w = ⊤}

**3 Intervals**

# Adaptive Analysis



# Model

- Model  $M : \text{Variable} \rightarrow [0, 1]$
- Importance of each variable in terms of flow-sensitivity
- Learned using Bayesian Optimization
  - represent variables as feature vectors and learn weights of features

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);      // Query 1 (hold)
6: assert(z > 0);      // Query 2 (hold)
7: assert(v == w);      // Query 3 (may fail)
```

$$M(x) > M(y) > M(z) > M(v) > M(w)$$

# Controller

- Controller  $\pi : F \rightarrow \text{Pr}(A)$  where  $A = \{0, \dots, 100\}$
- Input: a feature vector describing current status
  - e.g., memory usage, analysis progress, etc
- Output: prob. distribution on % of vars that should be treated flow-insensitively
- Learned using reinforcement learning

# Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);      // Query 1 (hold)
6: assert(z > 0);      // Query 2 (hold)
7: assert(v == w);      // Query 3 (may fail)
```

Model:  $M(x) > M(y) > M(z) > M(v) > M(w)$

# Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);      // Query 1 (hold)
6: assert(z > 0);      // Query 2 (hold)
7: assert(v == w);      // Query 3 (may fail)
```

Model:  $M(x) > M(y) > M(z) > M(v) > M(w)$

Line	Flow-Sensitive Abstract State
1	{x = [0,0], y = [0,0], z = [1,1], v = $\top$ , w = $\top$ }
2	{x = [1,1], y = [0,0], z = [1,1], v = $\top$ , w = $\top$ }

6 Intervals

# Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);      // Query 1 (hold)
6: assert(z > 0);      // Query 2 (hold)
7: assert(v == w);      // Query 3 (may fail)
```

Model:  $M(x) > M(y) > M(z) > M(v) > \cancel{M(w)}$

Line	Flow-Sensitive	Flow-Insensitive
1	{x = [0,0], y = [0,0], z = [1,1], v = $\top$ }	
2	{x = [1,1], y = [0,0], z = [1,1], v = $\top$ }	{w = $\top$ }

6 Intervals

# Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);      // Query 1 (hold)
6: assert(z > 0);      // Query 2 (hold)
7: assert(v == w);      // Query 3 (may fail)
```

Model:  $M(x) > M(y) > M(z) > M(v) > \cancel{M(w)}$

Line	Flow-Sensitive	Flow-Insensitive
1	{x = [0,0], y = [0,0], z = [1,1], v = $\top$ }	
2	{x = [1,1], y = [0,0], z = [1,1], v = $\top$ }	{w = $\top$ }
3	{x = [1,1], y = [0,0], z = [2,2], v = $\top$ }	

**9 Intervals**

# Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);      // Query 1 (hold)
6: assert(z > 0);      // Query 2 (hold)
7: assert(v == w);      // Query 3 (may fail)
```

Model:  $M(x) > M(y) > M(z) > M(v) > M(w)$

Line	Flow-Sensitive	Flow-Insensitive
1	{x = [0,0], y = [0,0]}	
2	{x = [1,+∞], y = [0,0]}	{z = [1,+∞], v = T, w = T}
3	{x = [1,+∞], y = [0,0]}	
6 Intervals		

# Controller

- Partially flow-sensitive interval analysis (budget: 10 intervals)

```
1: x = 0; y = 0; z = 1; v = input(); w = input();
2: x = z;
3: z = z + 1;
4: y = x;
5: assert(y > 0);      // Query 1 (hold)
6: assert(z > 0);      // Query 2 (hold)
7: assert(v == w);      // Query 3 (may fail)
```

Model:  $M(x) > M(y) > M(z) > M(v) > M(w)$

Line	Flow-Sensitive	Flow-Insensitive
1	{x = [0,0], y = [0,0]}	
2	{x = [1,+∞], y = [0,0]}	$\{z = [1,+∞],$ $v = \top, w = \top\}$
3	{x = [1,+∞], y = [0,0]}	
4	$\{x = [1,+∞], y = [1,+∞]\}$	

**8 Intervals**

# Practical Impact

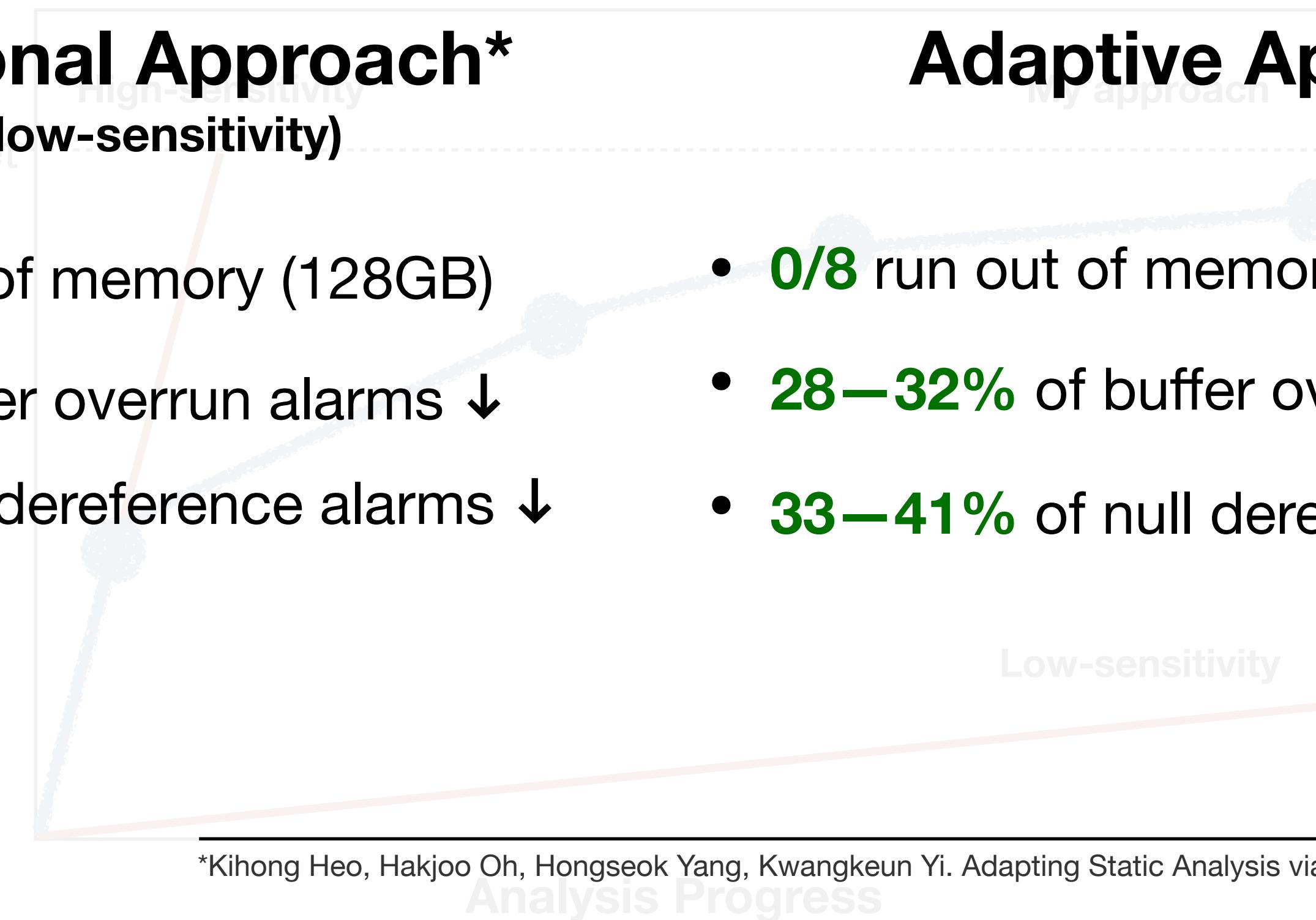
- Achieving **maximum precision** within a given **resource budget**
  - e.g., within 128GB of memory

## Conventional Approach\* (10% flow-sensitivity)

- **3/8** run out of memory (128GB)
- **27%** of buffer overrun alarms ↓
- **30%** of null dereference alarms ↓

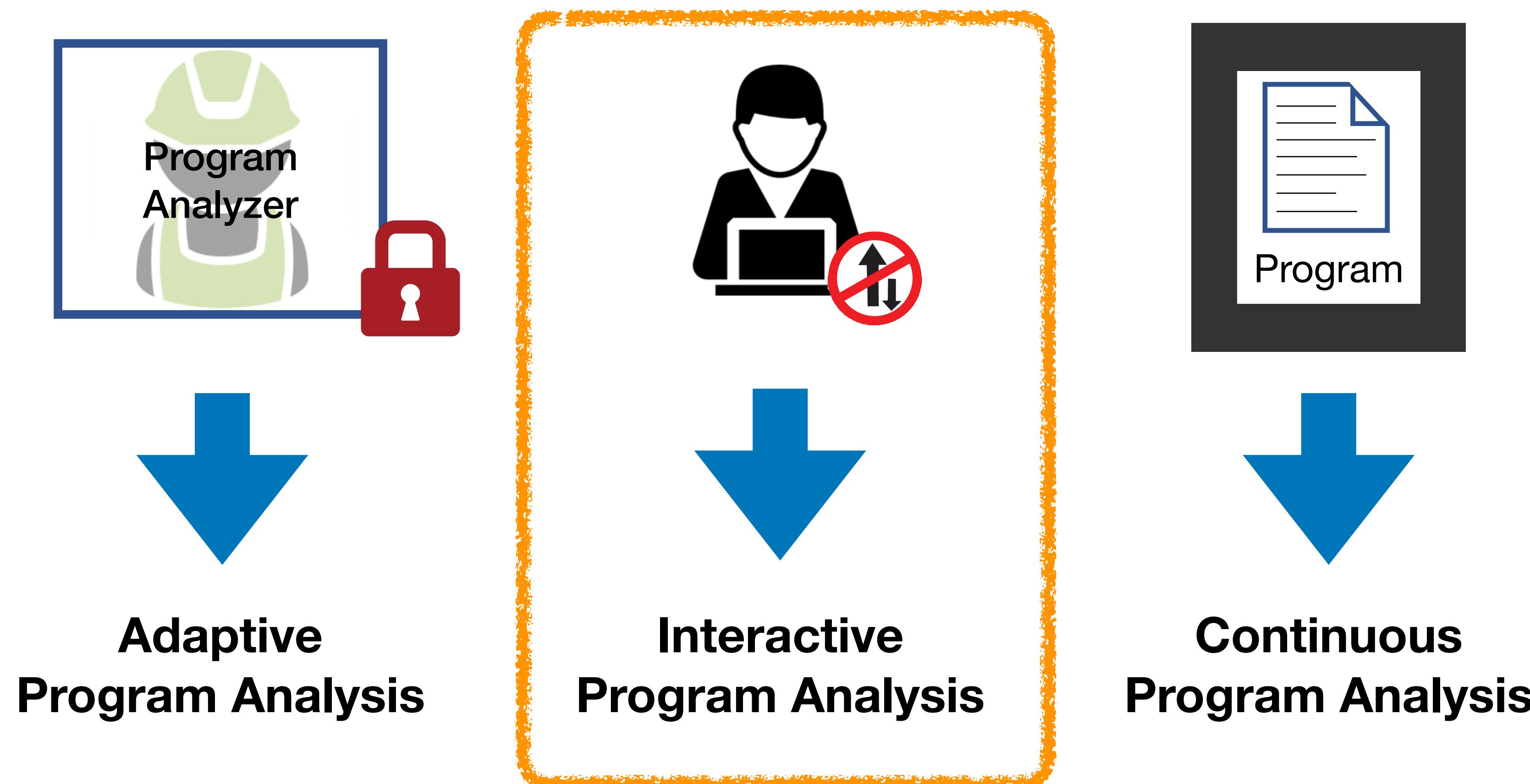
## Adaptive Approach

- **0/8** run out of memory (64 / 128GB)
- **28–32%** of buffer overrun alarms ↓
- **33–41%** of null dereference alarms ↓



\*Kihong Heo, Hakjoo Oh, Hongseok Yang, Kwangkeun Yi. Adapting Static Analysis via Learning with Bayesian Optimization. ACM TOPLAS, 2018

# Outline



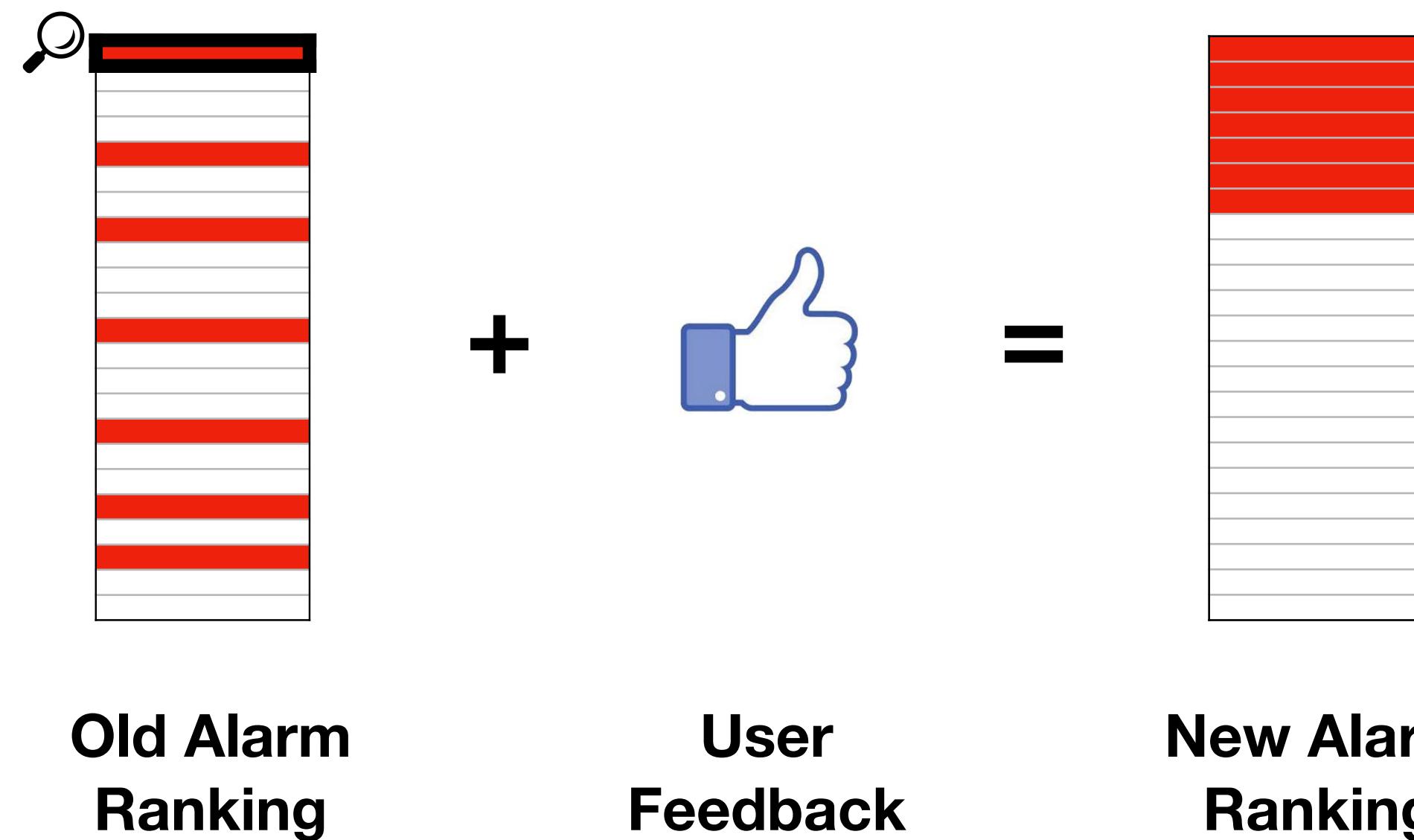
**Adaptive  
Program Analysis**

**Interactive  
Program Analysis**

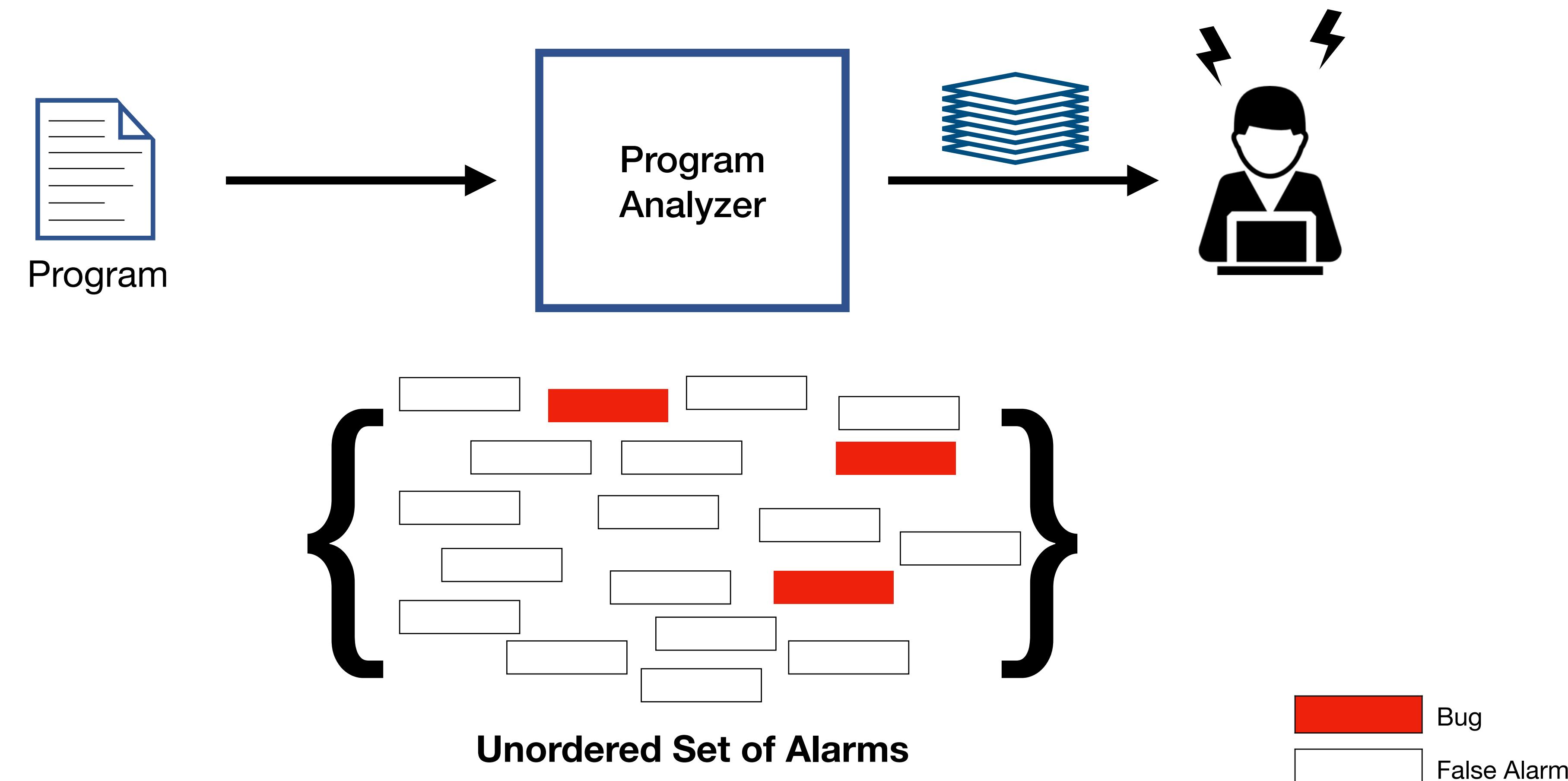
**Continuous  
Program Analysis**

# Outline

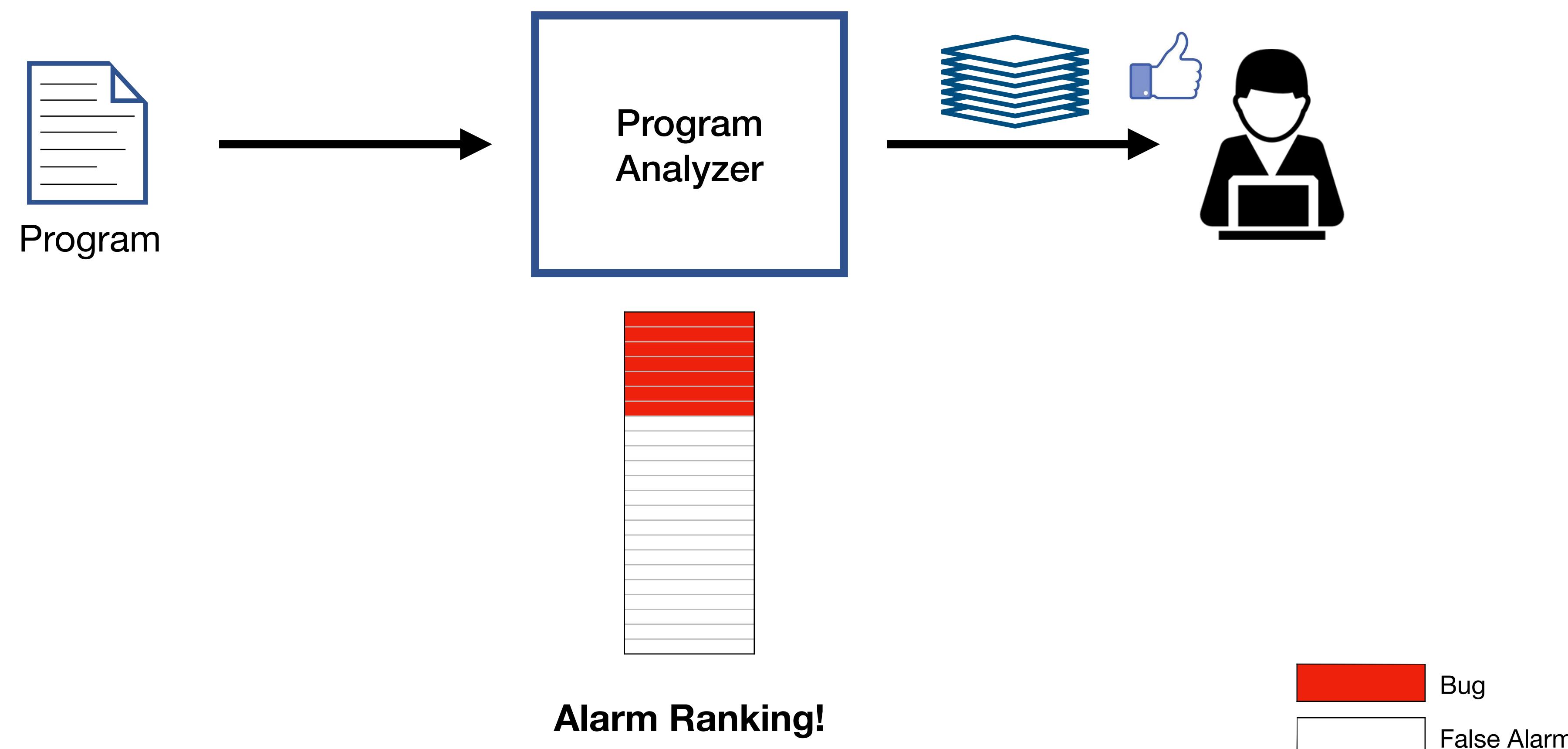
## Bingo: Interactive Alarm Ranking System [PLDI'18]



# Alarm Report

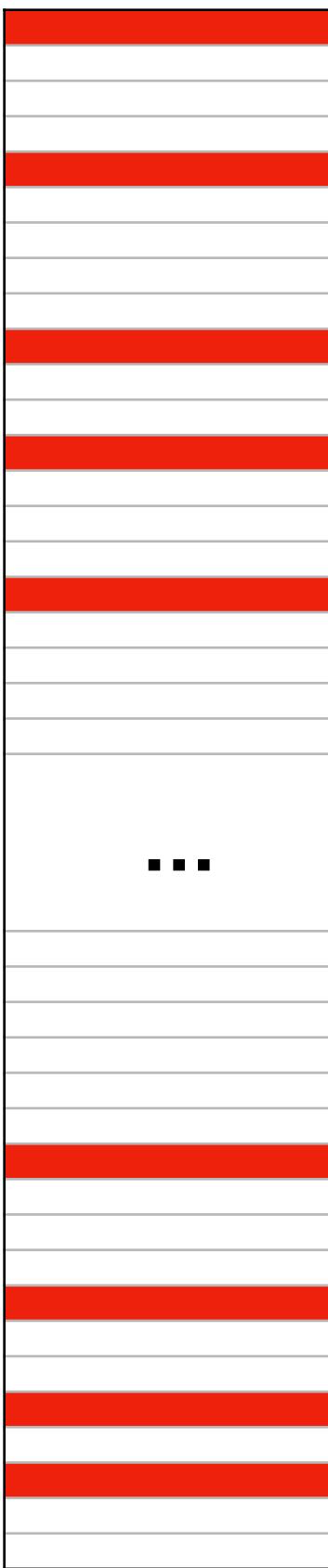


# Goal



# Interactive Alarm Ranker

Rank 1



Rank 522



**152KLOC**  
**75 Datarace Bugs**  
**522 Total Alarms**

Bug  
 False Alarm

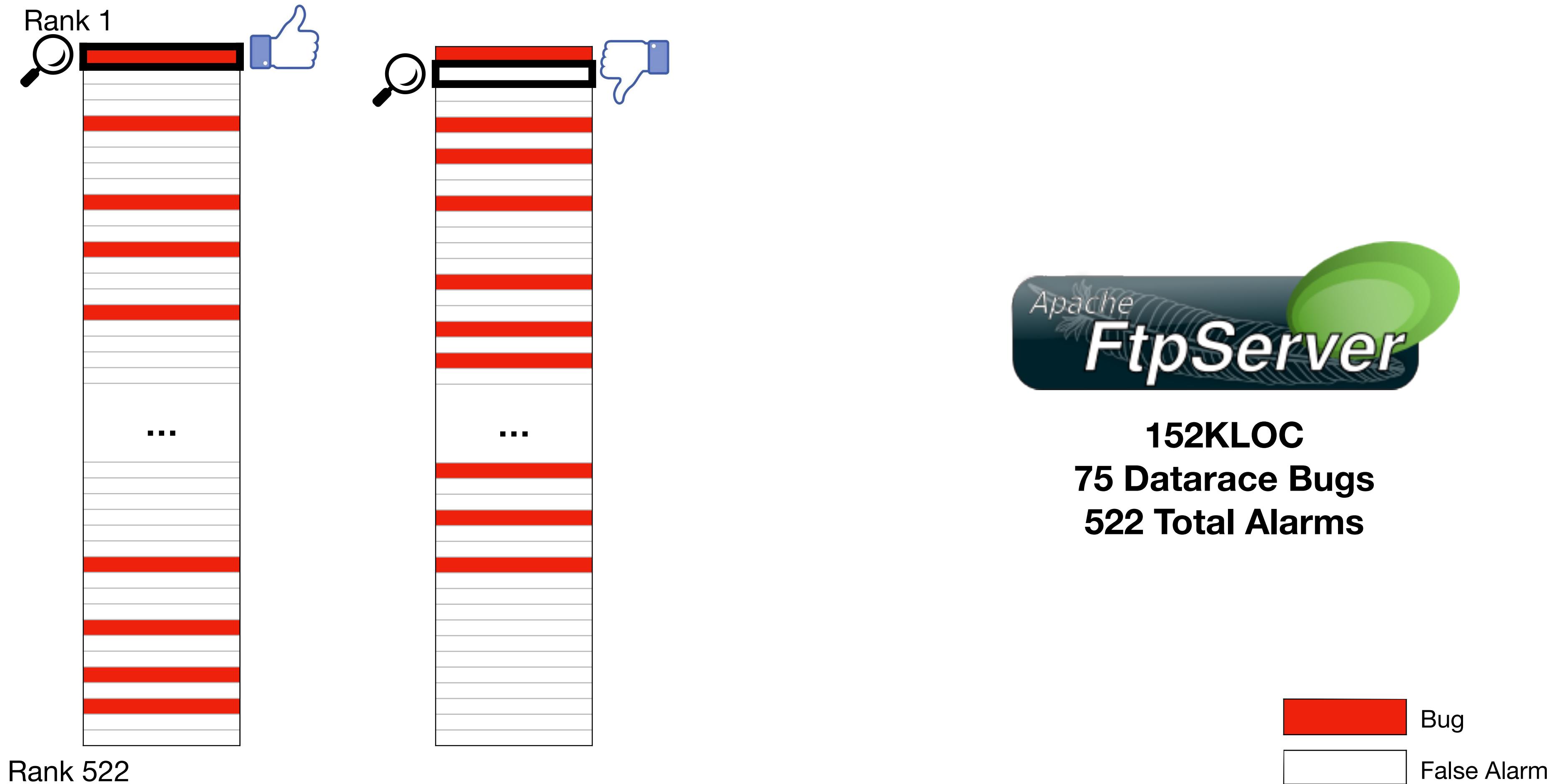
# Interactive Alarm Ranker



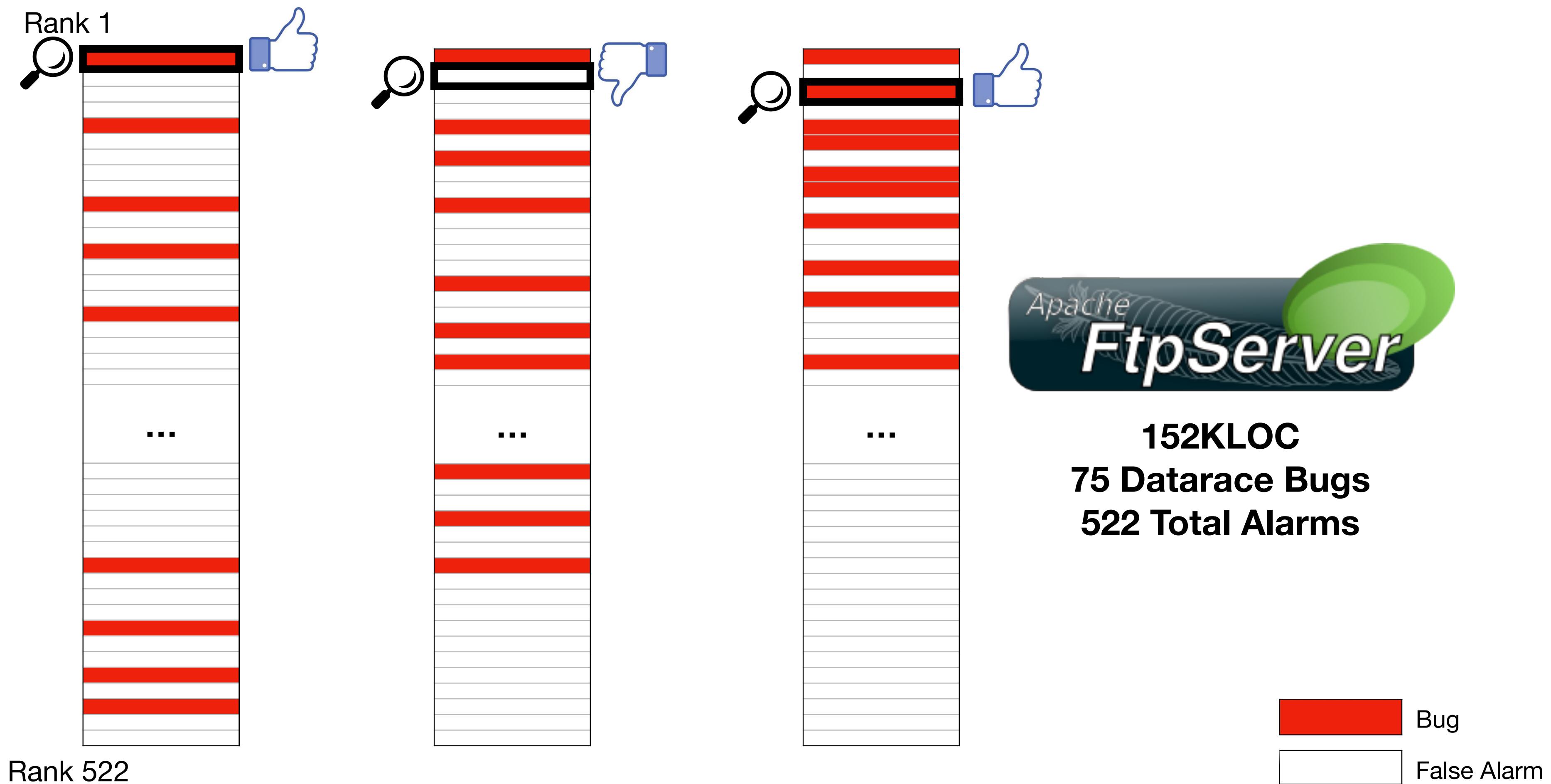
# Interactive Alarm Ranker



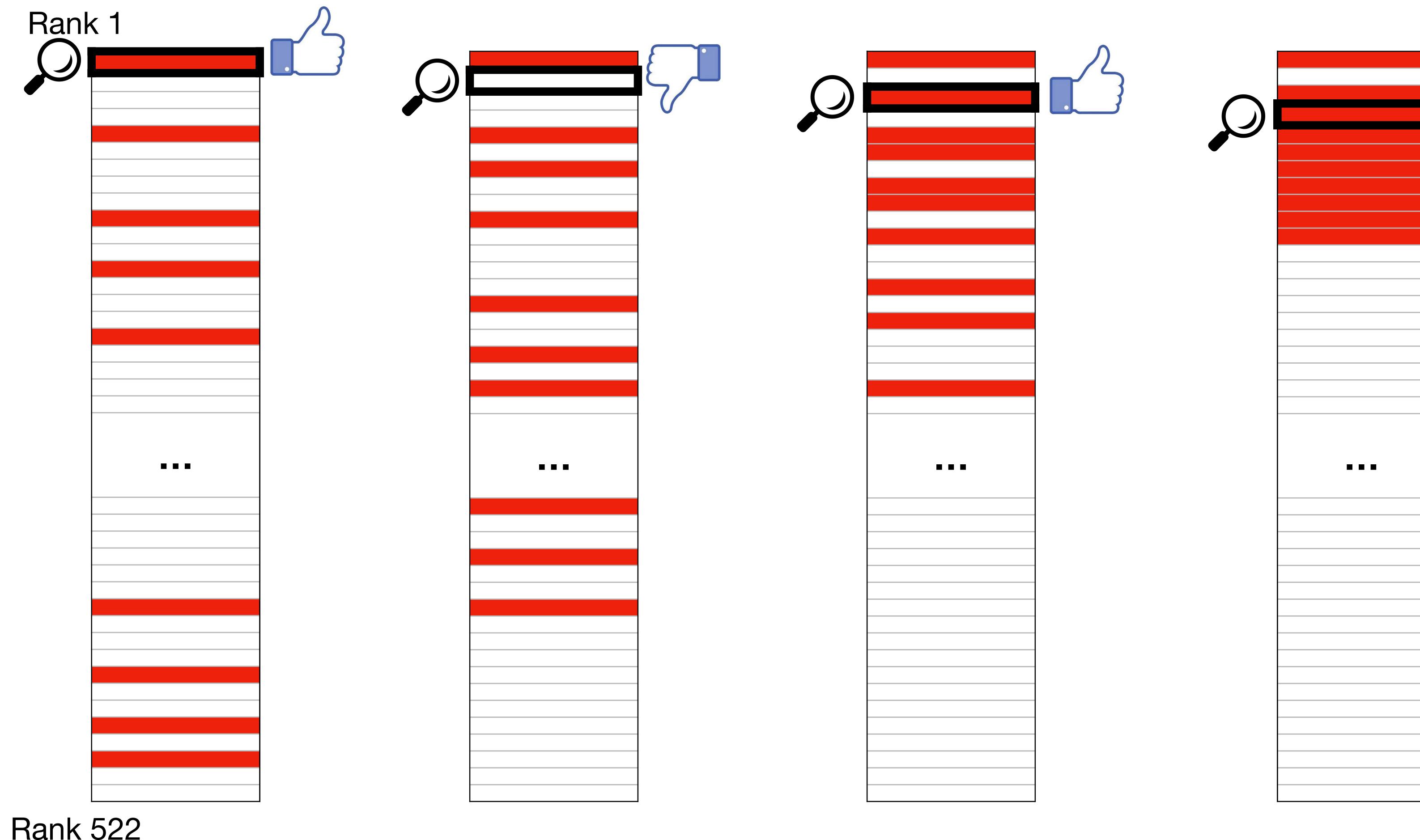
# Interactive Alarm Ranker



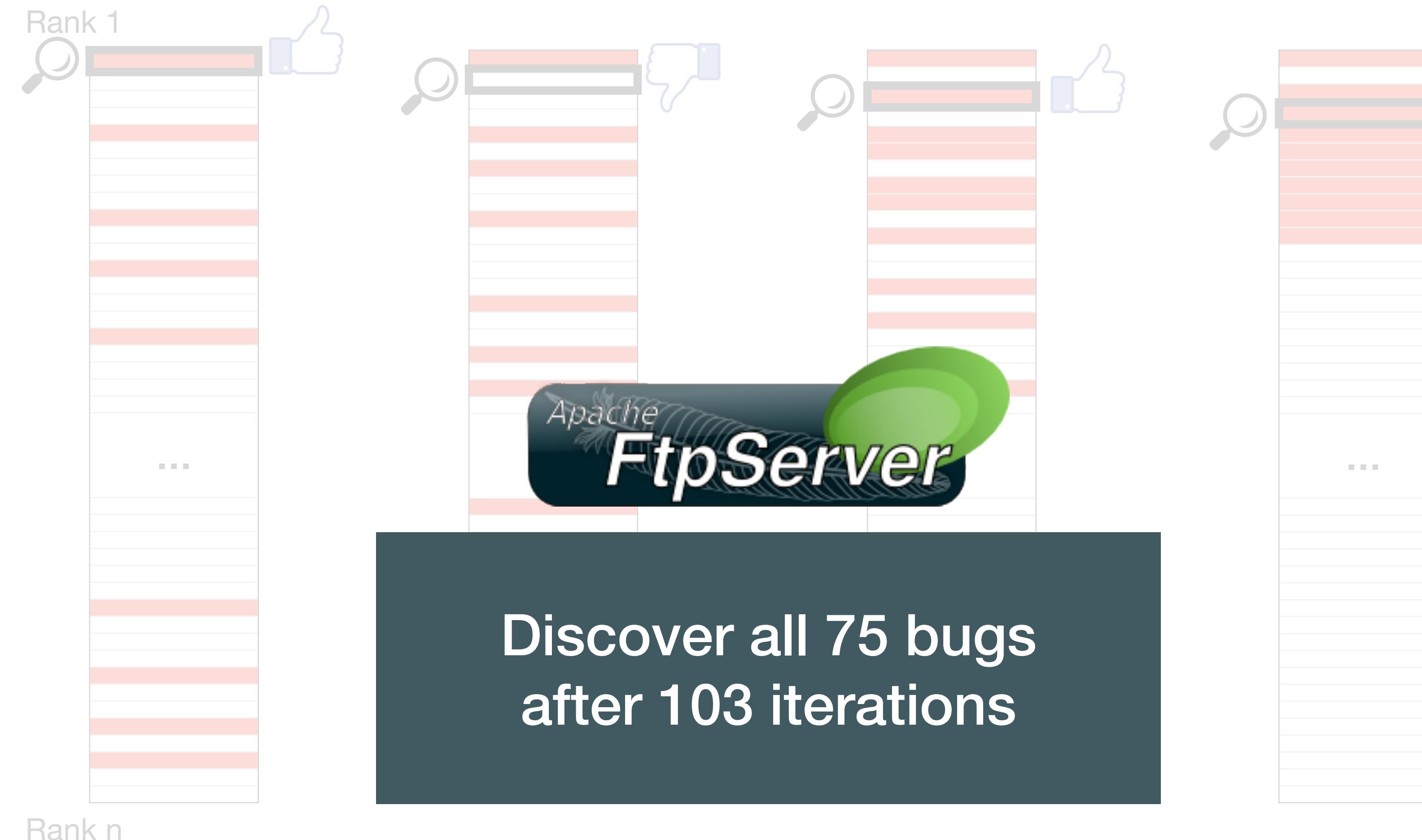
# Interactive Alarm Ranker



# Interactive Alarm Ranker

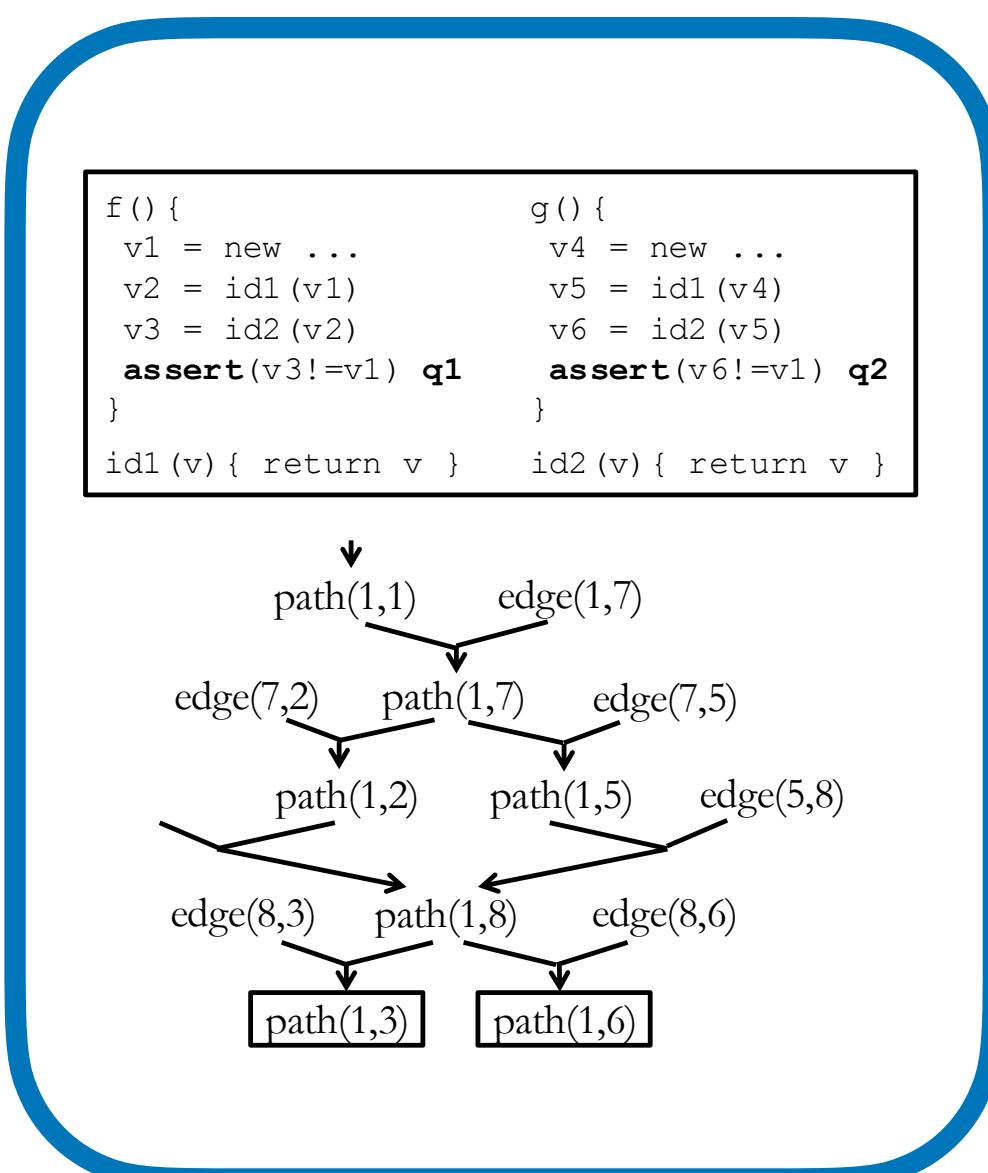


# Interactive Alarm Ranker

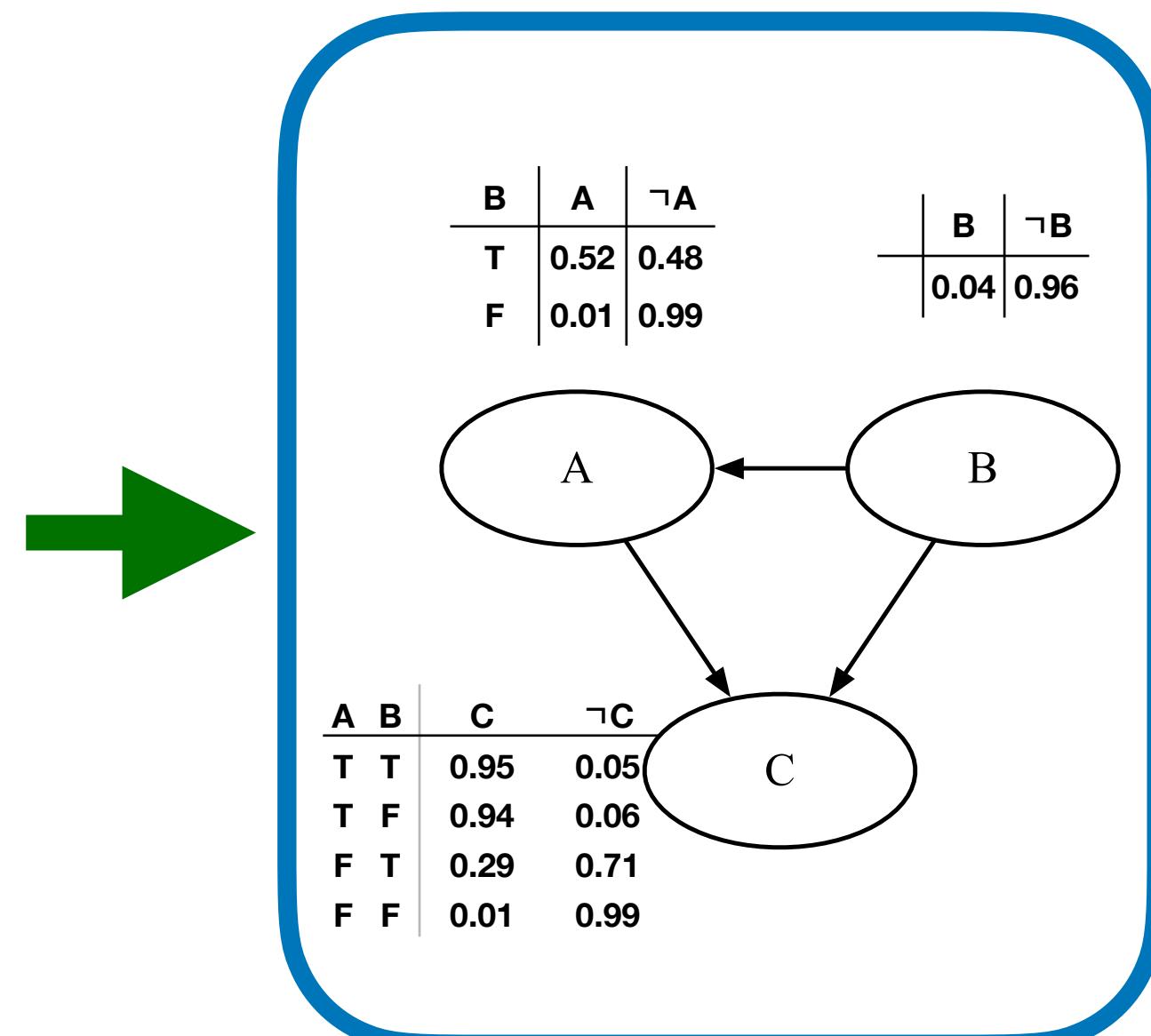


# Key Idea

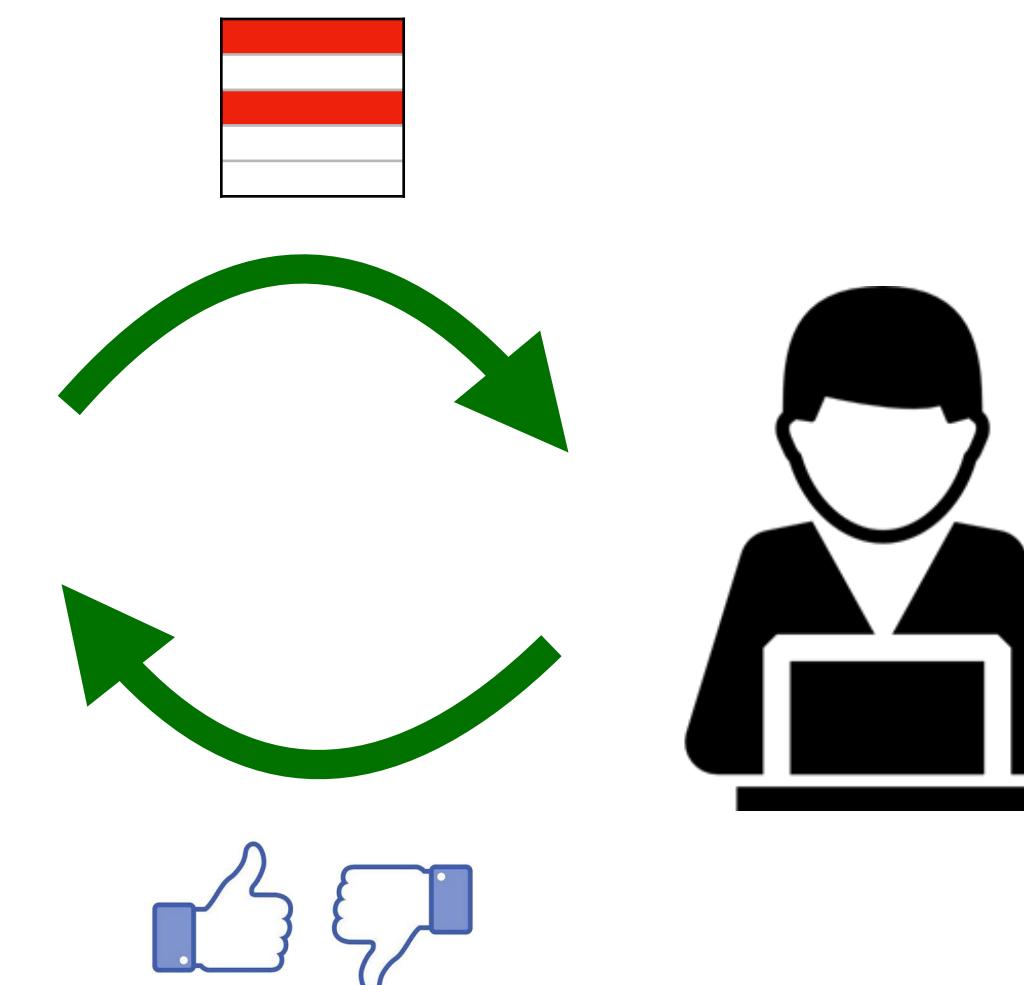
## Human in the loop + Bayesian inference



Program Analysis Result



Probabilistic Model  
(e.g., Bayesian Network)



User

# Datarace Analysis

**Analysis Inputs:**

$\text{Next}(p_1, p_2)$ ,  $\text{Alias}(p_1, p_2)$ ,  $\text{Unguarded}(p_1, p_2)$ .

**Analysis Outputs:**

$\text{Parallel}(p_1, p_2)$ ,  $\text{Race}(p_1, p_2)$

**Analysis Rules:**

**r<sub>1</sub>:**  $\text{Parallel}(p_1, p_3) :- \text{Parallel}(p_1, p_2), \text{Next}(p_2, p_3), \text{Unguarded}(p_1, p_3)$ .

**r<sub>2</sub>:**  $\text{Parallel}(p_1, p_2) :- \text{Parallel}(p_2, p_1)$ .

**r<sub>3</sub>:**  $\text{Race}(p_1, p_2) :- \text{Parallel}(p_1, p_2), \text{Alias}(p_1, p_2)$ .

# Datarace Analysis

$p_i$  is a program point

## Analysis Inputs:

$\text{Next}(p_1, p_2)$ ,  $\text{Alias}(p_1, p_2)$ ,  $\text{Unguarded}(p_1, p_2)$ .

Program point  $p_2$  is  
an immediate successor of  $p_1$

$p_1$  and  $p_2$  may access  
the same memory location

$p_1$  and  $p_2$  are not guarded by  
the same lock

## Analysis Rules:

- r<sub>1</sub>:**  $\text{Parallel}(p_1, p_3) :- \text{Parallel}(p_1, p_2), \text{Next}(p_2, p_3), \text{Unguarded}(p_1, p_3).$
- r<sub>2</sub>:**  $\text{Parallel}(p_1, p_2) :- \text{Parallel}(p_2, p_1).$
- r<sub>3</sub>:**  $\text{Race}(p_1, p_2) :- \text{Parallel}(p_1, p_2), \text{Alias}(p_1, p_2).$

# Datarace Analysis

## Analysis Inputs:

$\text{Next}(p_1, p_2)$ ,  $\text{Alias}(p_1, p_2)$ ,  $\text{Unguarded}(p_1, p_2)$ .

## Analysis Outputs:

$\text{Parallel}(p_1, p_2)$ ,  $\text{Race}(p_1, p_2)$

Program point  $p_1$  and  $p_2$  can be  
executed in parallel

$\text{r}_1: \text{Parallel}(p_1, p_2) :- \text{Next}(p_1, p_2), \text{Next}(p_2, p_1), \text{Unguarded}(p_1, p_2), \text{Unguarded}(p_2, p_1).$

Race condition  
between  $p_1$  and  $p_2$

$\text{r}_2: \text{Parallel}(p_1, p_2) :- \text{Parallel}(p_2, p_1).$

$\text{r}_3: \text{Race}(p_1, p_2) :- \text{Parallel}(p_1, p_2), \text{Alias}(p_1, p_2).$

# Datarace Analysis

## Analysis Inputs:

$\text{Next}(p_1, p_2)$ ,  $\text{Alias}(p_1, p_2)$ ,  $\text{Unguarded}(p_1, p_2)$ .

## Analysis Outputs:

$\text{Parallel}(p_1, p_2)$ ,  $\text{Race}(p_1, p_2)$

## Analysis Rules:

- r<sub>1</sub>:**  $\text{Parallel}(p_1, p_3) :- \text{Parallel}(p_1, p_2), \text{Next}(p_2, p_3), \text{Unguarded}(p_1, p_3)$ .
- r<sub>2</sub>:**  $\text{Parallel}(p_1, p_2) :- \text{Parallel}(p_2, p_1)$ .
- r<sub>3</sub>:**  $\text{Race}(p_1, p_2) :- \text{Parallel}(p_1, p_2), \text{Alias}(p_1, p_2)$ .

Thread 1

```
x = y + 1; // L1
...
...
```

Thread 2

```
z = y + 1; // L2
x = z + 1; // L3
...
...
```

# Datarace Analysis

## Analysis Inputs:

$\text{Next}(p_1, p_2)$ ,  $\text{Alias}(p_1, p_2)$ ,  $\text{Unguarded}(p_1, p_2)$ .

## Analysis Outputs:

$\text{Parallel}(p_1, p_2)$ ,  $\text{Race}(p_1, p_2)$

## Analysis Rules:

**r<sub>1</sub>:**  $\text{Parallel}(p_1, p_3) :- \text{Parallel}(p_1, p_2), \text{Next}(p_2, p_3), \text{Unguarded}(p_1, p_3)$ .

**r<sub>2</sub>:**  $\text{Parallel}(p_1, p_2) :- \text{Parallel}(p_2, p_1)$ .

**r<sub>3</sub>:**  $\text{Race}(p_1, p_2) :- \text{Parallel}(p_1, p_2), \text{Alias}(p_1, p_2)$ .

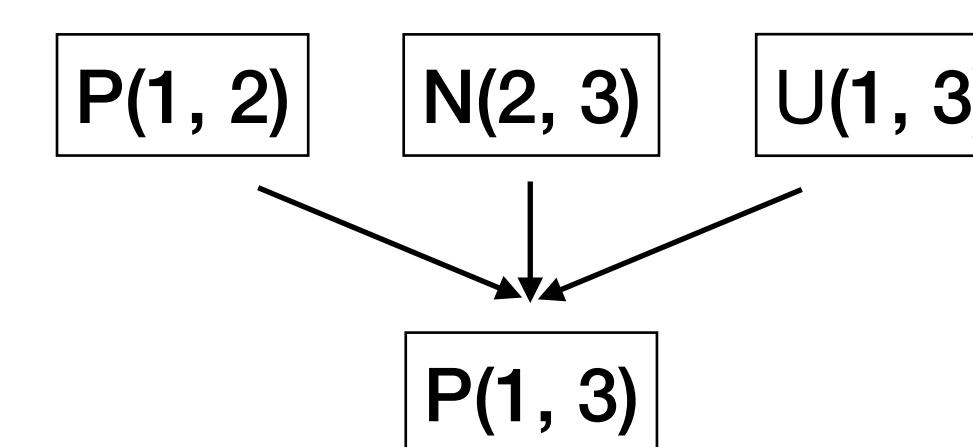
Thread 1

```
x = y + ...  
...  
x = y + 1; // L1  
...
```

Thread 2

```
z = y + ...  
...  
z = y + 1; // L2  
x = z + 1; // L3  
...
```

Derivation



# Datarace Analysis

**Analysis Inputs:**

$\text{Next}(p_1, p_2)$ ,  $\text{Alias}(p_1, p_2)$ ,  $\text{Unguarded}(p_1, p_2)$ .

**Analysis Outputs:**

$\text{Parallel}(p_1, p_2)$ ,  $\text{Race}(p_1, p_2)$

**Analysis Rules:**

$r_1: \text{Parallel}(p_1, p_3) :- \text{Parallel}(p_1, p_2), \text{Next}(p_2, p_3), \text{Unguarded}(p_1, p_3).$

$r_2: \text{Parallel}(p_1, p_2) :- \text{Parallel}(p_2, p_1).$

$r_3: \text{Race}(p_1, p_2) :- \text{Parallel}(p_1, p_2), \text{Alias}(p_1, p_2).$

**Thread 1**

```
x = y + 1; // L1  
...
```

**Thread 2**

```
z = y + 1; // L2  
x = z + 1; // L3  
...
```

**Derivation**

P(1, 2)



P(2, 1)

# Datarace Analysis

## Analysis Inputs:

$\text{Next}(p_1, p_2)$ ,  $\text{Alias}(p_1, p_2)$ ,  $\text{Unguarded}(p_1, p_2)$ .

## Analysis Outputs:

$\text{Parallel}(p_1, p_2)$ ,  $\text{Race}(p_1, p_2)$

## Analysis Rules:

$r_1: \text{Parallel}(p_1, p_3) :- \text{Parallel}(p_1, p_2), \text{Next}(p_2, p_3), \text{Unguarded}(p_1, p_3).$

$r_2: \text{Parallel}(p_1, p_2) :- \text{Parallel}(p_2, p_1).$

$r_3: \text{Race}(p_1, p_2) :- \text{Parallel}(p_1, p_2), \text{Alias}(p_1, p_2).$

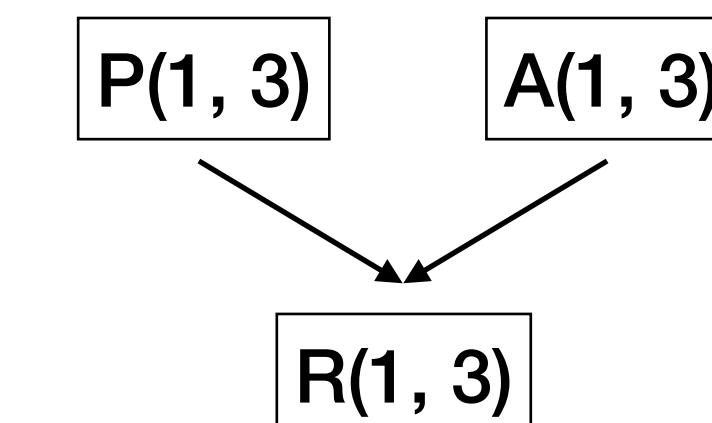
Thread 1

```
x = y + 1; // L1
...
...
```

Thread 2

```
z = y + 1; // L2
x = z + 1; // L3
...
...
```

Derivation



# Example

```
public class RequestHandler {  
    private FtpRequest request;  
  
    public FtpRequest getRequest() {  
        return request; //L0  
    }  
  
    public void close() {  
        synchronized (this) { //L1  
            if (isClosed) return; //L2  
            isClosed = true; //L3  
        }  
        controlSocket.close(); //L4  
        controlSocket = null; //L5  
        request.clear(); //L6  
        request = null; //L7  
    }  
}
```

## Analysis Rules:

- r<sub>1</sub>: P( $p_1, p_3$ ) :- P( $p_1, p_2$ ), N( $p_2, p_3$ ), U( $p_1, p_3$ ).
- r<sub>2</sub>: P( $p_1, p_2$ ) :- P( $p_2, p_1$ ).
- r<sub>3</sub>: R( $p_1, p_2$ ) :- P( $p_1, p_2$ ), A( $p_1, p_2$ ).

\*Apache FTP Server

# Example

```
public class RequestHandler {  
    private FtpRequest request;  
  
    public FtpRequest getRequest() {  
        return request; //L0  
    }  
  
    public void close() {  
        synchronized (this) {  
            if (isClosed) return; //L1  
            isClosed = true; //L2  
        }  
        controlSocket.close(); //L3  
        controlSocket = null; //L4  
        request.clear(); //L5  
        request = null; //L6  
    }  
}
```

## Analysis Rules:

- r<sub>1</sub>: P(p<sub>1</sub>, p<sub>3</sub>) :- P(p<sub>1</sub>, p<sub>2</sub>), N(p<sub>2</sub>, p<sub>3</sub>), U(p<sub>1</sub>, p<sub>3</sub>).
- r<sub>2</sub>: P(p<sub>1</sub>, p<sub>2</sub>) :- P(p<sub>2</sub>, p<sub>1</sub>).
- r<sub>3</sub>: R(p<sub>1</sub>, p<sub>2</sub>) :- P(p<sub>1</sub>, p<sub>2</sub>), A(p<sub>1</sub>, p<sub>2</sub>).

Datarace

\*Apache FTP Server

# Example

```
public class RequestHandler {  
    private FtpRequest request;  
  
    public FtpRequest getRequest() {  
        return request; //L0  
    }  
  
    public void close() {  
        synchronized (this) { //L1  
            if (isClosed) return; //L2  
            isClosed = true; //L3  
        }  
        controlSocket.close(); //L4  
        controlSocket = null; //L5  
        request.clear(); //L6  
        request = null; //L7  
    }  
}
```

## Analysis Rules:

- r<sub>1</sub>: P(p<sub>1</sub>, p<sub>3</sub>) :- P(p<sub>1</sub>, p<sub>2</sub>), N(p<sub>2</sub>, p<sub>3</sub>), U(p<sub>1</sub>, p<sub>3</sub>).
- r<sub>2</sub>: P(p<sub>1</sub>, p<sub>2</sub>) :- P(p<sub>2</sub>, p<sub>1</sub>).
- r<sub>3</sub>: R(p<sub>1</sub>, p<sub>2</sub>) :- P(p<sub>1</sub>, p<sub>2</sub>), A(p<sub>1</sub>, p<sub>2</sub>).

False alarm

False alarm

\*Apache FTP Server

# Applying the Analysis

## Program

```
controlSocket.close(); //L4
controlSocket = null; //L5
request.clear(); //L6
request = null; //L7
```

## Analysis Rules

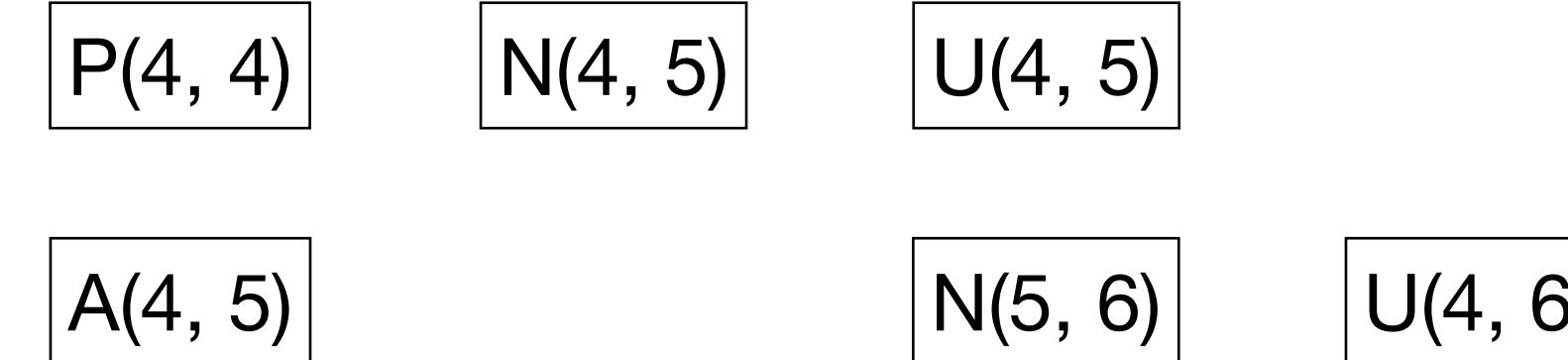
- r<sub>1</sub>:** P( $p_1, p_3$ ) :- P( $p_1, p_2$ ), N( $p_2, p_3$ ), U( $p_1, p_3$ ).
- r<sub>2</sub>:** P( $p_1, p_2$ ) :- P( $p_2, p_1$ ).
- r<sub>3</sub>:** R( $p_1, p_2$ ) :- P( $p_1, p_2$ ), A( $p_1, p_2$ ).

# Applying the Analysis

## Program

```
controlSocket.close(); //L4
controlSocket = null; //L5
request.clear(); //L6
request = null; //L7
```

## Derivation Graph



## Analysis Rules

- r<sub>1</sub>:**  $P(p_1, p_3) :- P(p_1, p_2), N(p_2, p_3), U(p_1, p_3).$
- r<sub>2</sub>:**  $P(p_1, p_2) :- P(p_2, p_1).$
- r<sub>3</sub>:**  $R(p_1, p_2) :- P(p_1, p_2), A(p_1, p_2).$

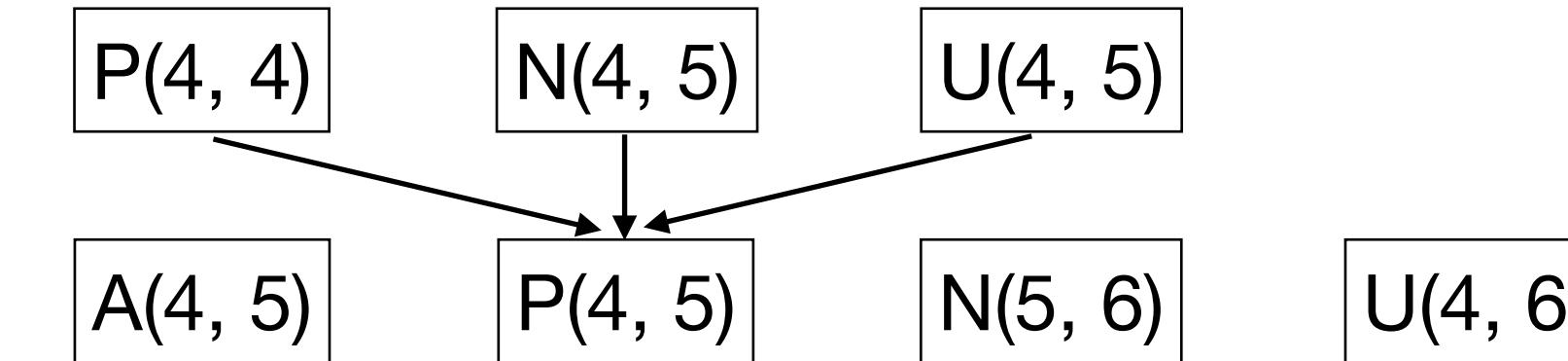
1. Start with inputs

# Applying the Analysis

## Program

```
controlSocket.close(); //L4
controlSocket = null; //L5
request.clear(); //L6
request = null; //L7
```

## Derivation Graph



## Analysis Rules

- r<sub>1</sub>:**  $P(p_1, p_3) :- P(p_1, p_2), N(p_2, p_3), U(p_1, p_3).$
- r<sub>2</sub>:**  $P(p_1, p_2) :- P(p_2, p_1).$
- r<sub>3</sub>:**  $R(p_1, p_2) :- P(p_1, p_2), A(p_1, p_2).$

2. Apply rules to inputs

# Applying the Analysis

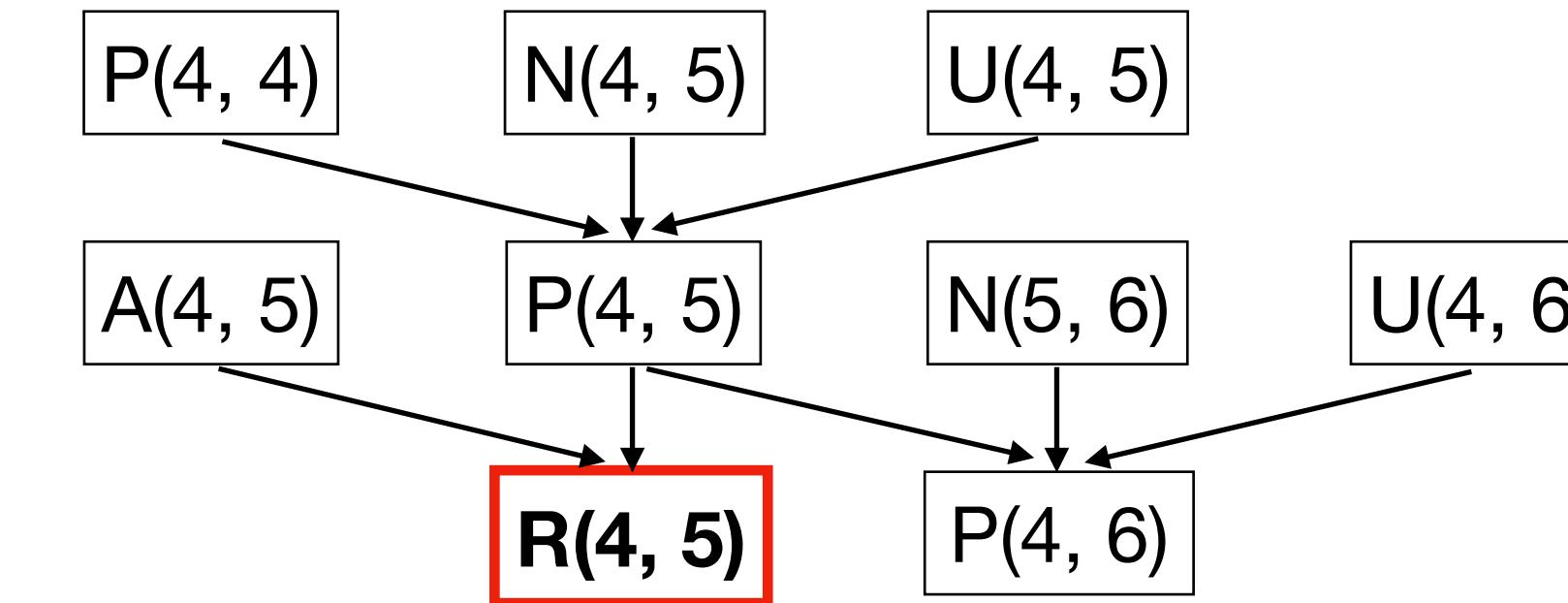
## Program

```
controlSocket.close(); //L4
controlSocket = null; //L5
request.clear(); //L6
request = null; //L7
```

## Analysis Rules

- r<sub>1</sub>:  $P(p_1, p_3) :- P(p_1, p_2), N(p_2, p_3), U(p_1, p_3).$
- r<sub>2</sub>:  $P(p_1, p_2) :- P(p_2, p_1).$
- r<sub>3</sub>:  $R(p_1, p_2) :- P(p_1, p_2), A(p_1, p_2).$

## Derivation Graph



3. Keep applying rules to all intermediate results

# Applying the Analysis

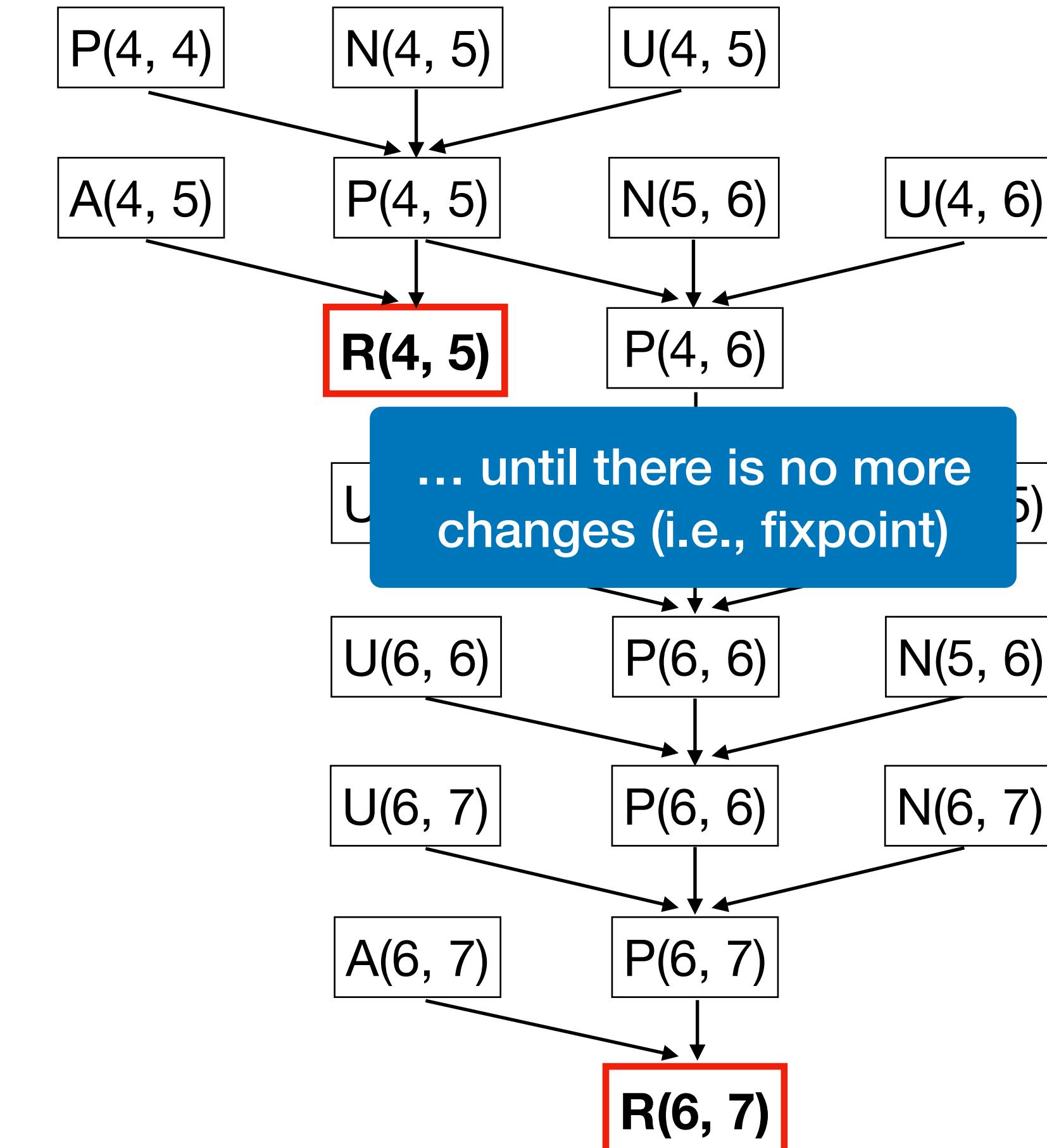
## Program

```
controlSocket.close(); //L4
controlSocket = null; //L5
request.clear(); //L6
request = null; //L7
```

## Analysis Rules

- r<sub>1</sub>:**  $P(p_1, p_3) :- P(p_1, p_2), N(p_2, p_3), U(p_1, p_3).$
- r<sub>2</sub>:**  $P(p_1, p_2) :- P(p_2, p_1).$
- r<sub>3</sub>:**  $R(p_1, p_2) :- P(p_1, p_2), A(p_1, p_2).$

## Derivation Graph

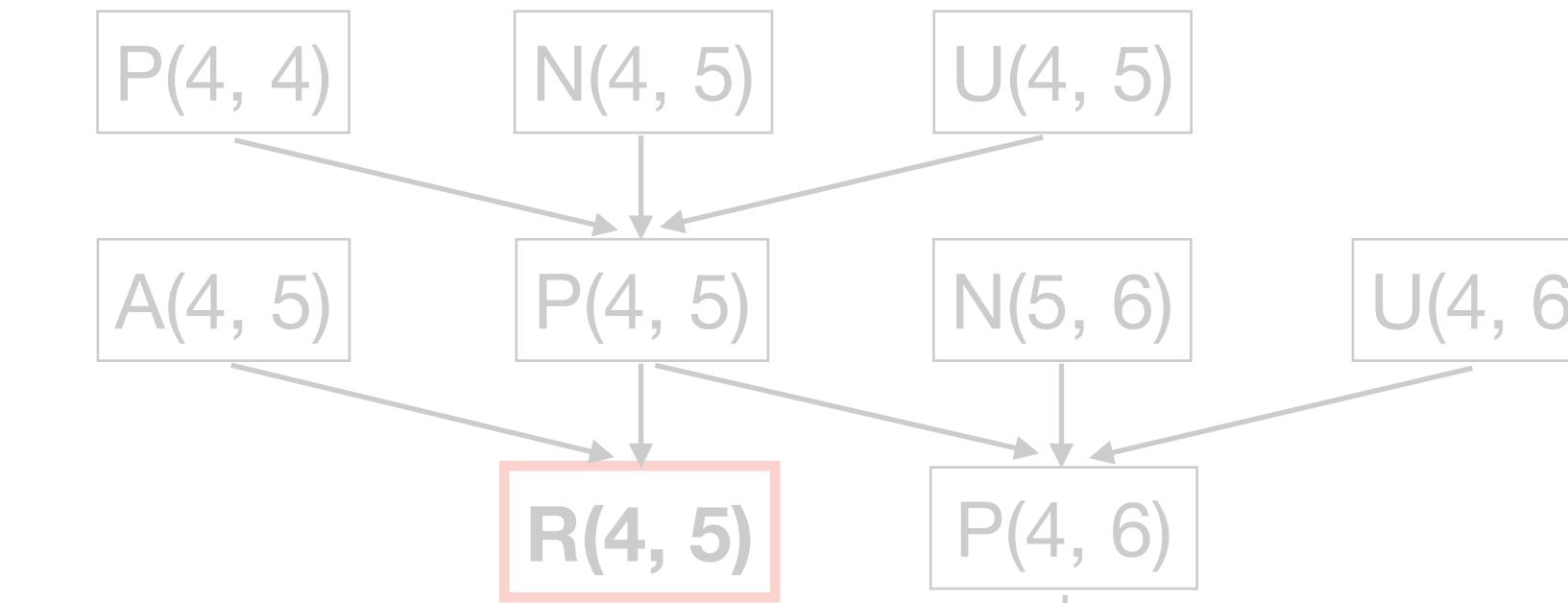


# Applying the Analysis

Program

```
controlSocket.close(); //L4
controlSocket = null; //L5
request.clear(); //L6
request = null; //L7
```

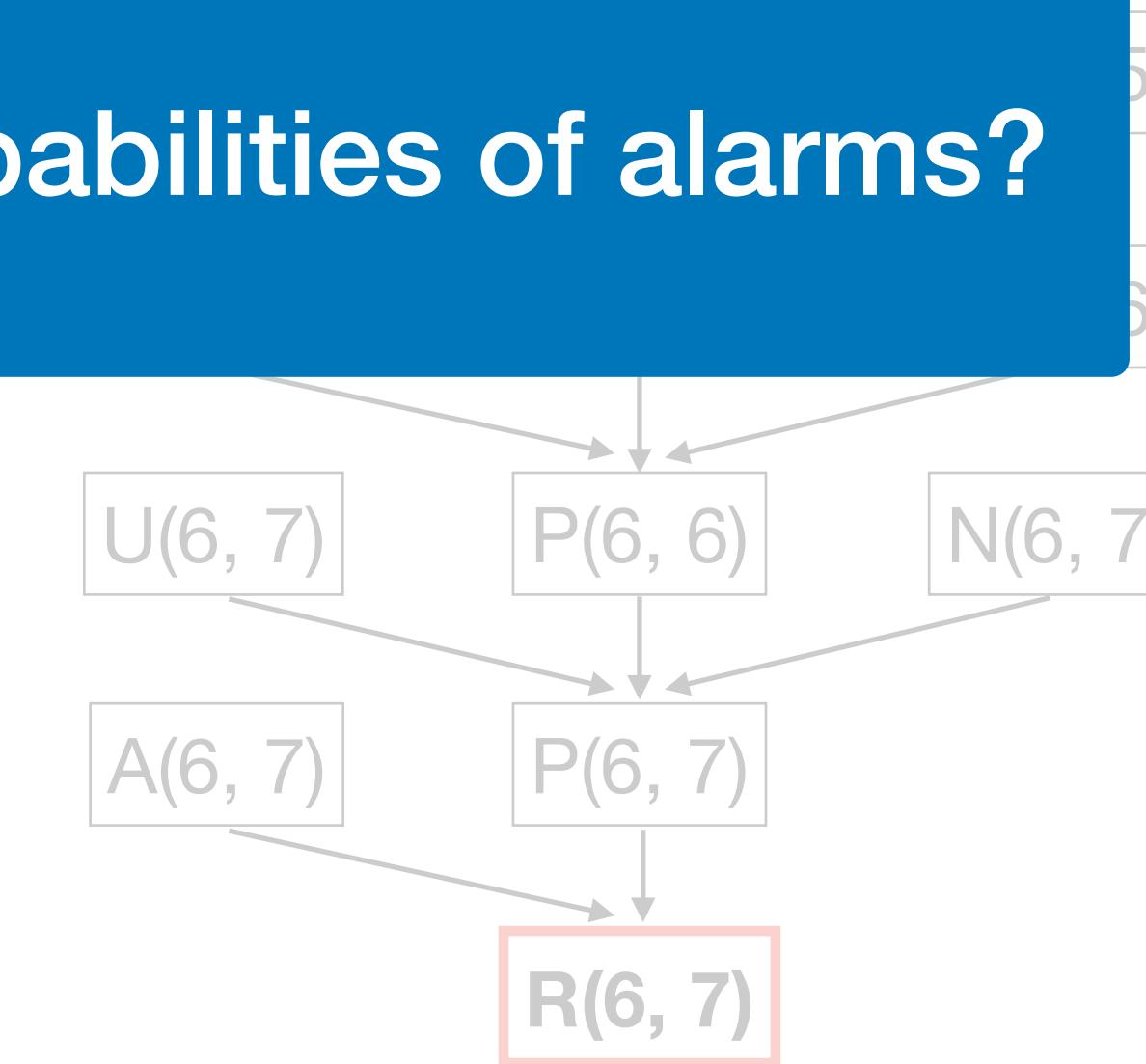
Derivation Graph



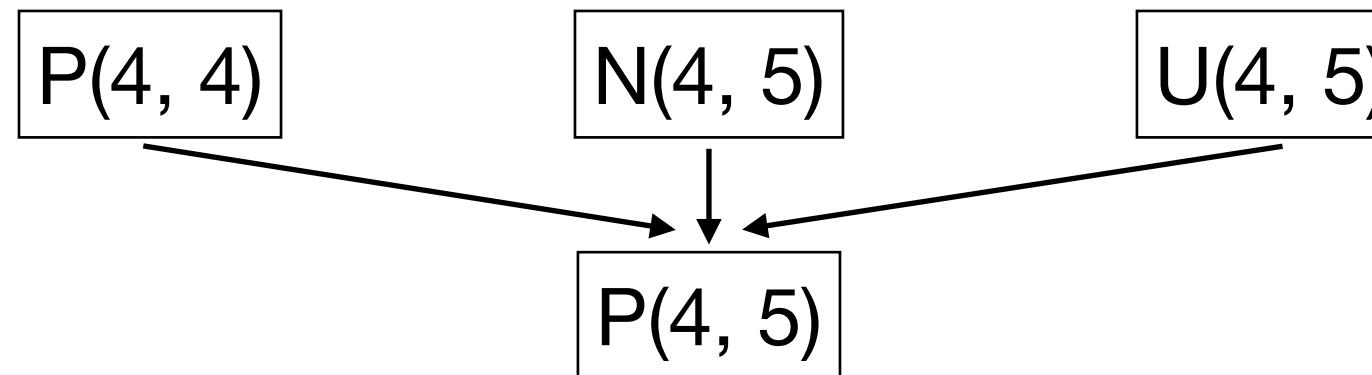
Analysis Rule

**r<sub>1</sub>:**  $P(p_1, p_3)$   
**r<sub>2</sub>:**  $P(p_1, p_2) \leftarrow P(p_2, p_1)$ .  
**r<sub>3</sub>:**  $R(p_1, p_2) \leftarrow P(p_1, p_2), A(p_1, p_2)$ .

**Q: How do we compute probabilities of alarms?**



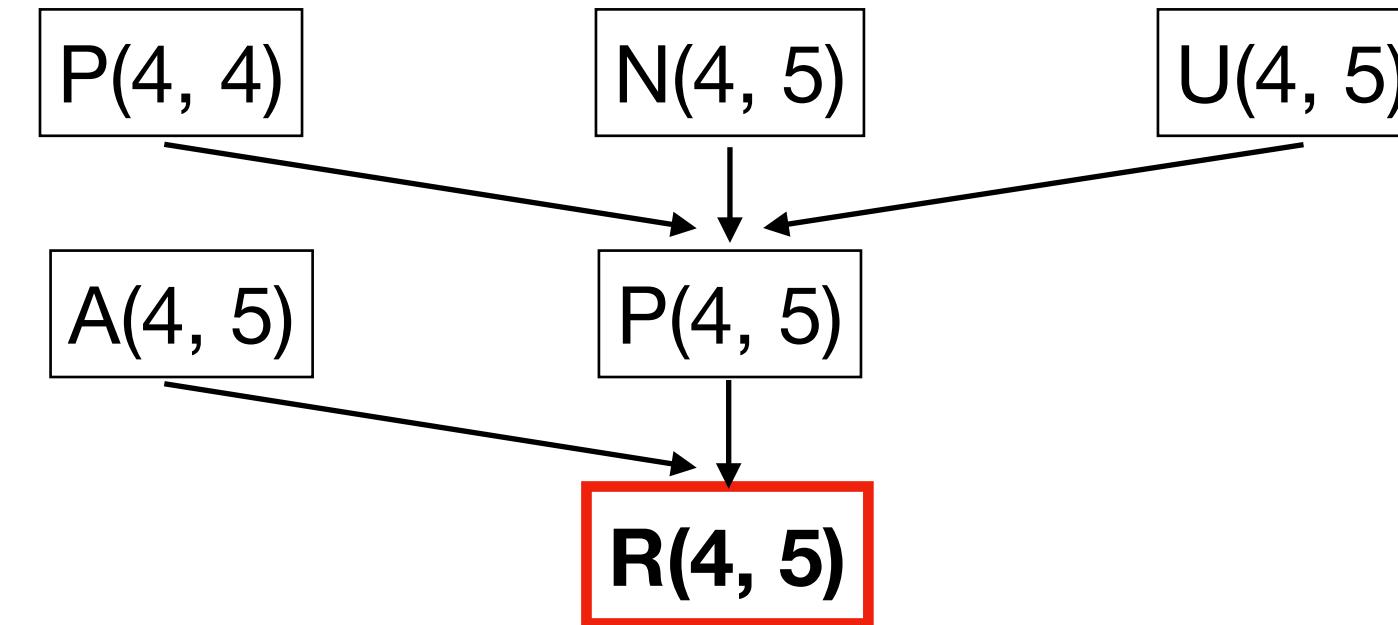
# Bayesian Network



Logical Rule		Probabilistic Rule			
		P(4,4)	N(4,5)	U(4,5)	$Pr(P(4,5)   \dots)$
<b>r<sub>1</sub>:</b>	$P(p_1, p_3) :- P(p_1, p_2), N(p_2, p_3), U(p_1, p_3).$	TRUE	TRUE	TRUE	0.95*
<b>r<sub>2</sub>:</b>	$P(p_1, p_2) :- P(p_2, p_1).$	TRUE	TRUE	FALSE	0
<b>r<sub>3</sub>:</b>	$R(p_1, p_2) :- P(p_1, p_2), A(p_1, p_2).$			...	
		FALSE	FALSE	FALSE	0

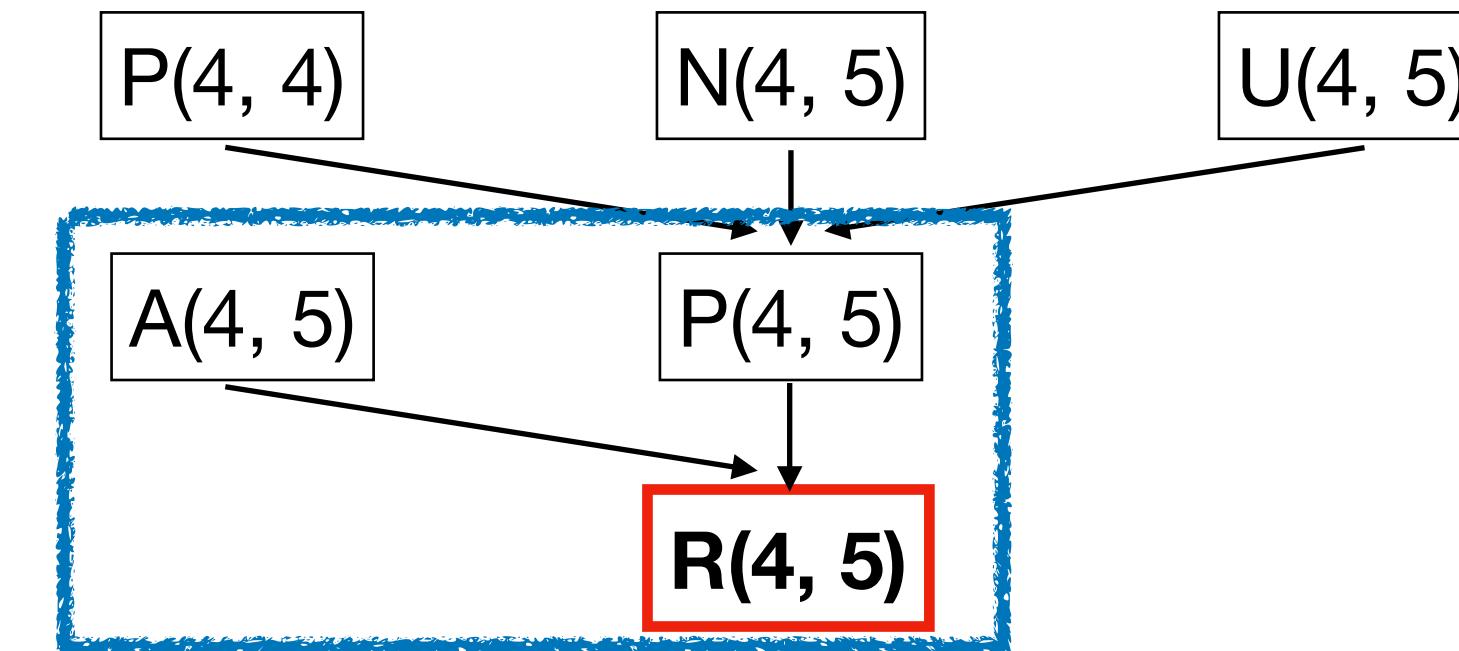
\*computed by an offline learning

# Marginal Inference



$$Pr(R(4,5)) =$$

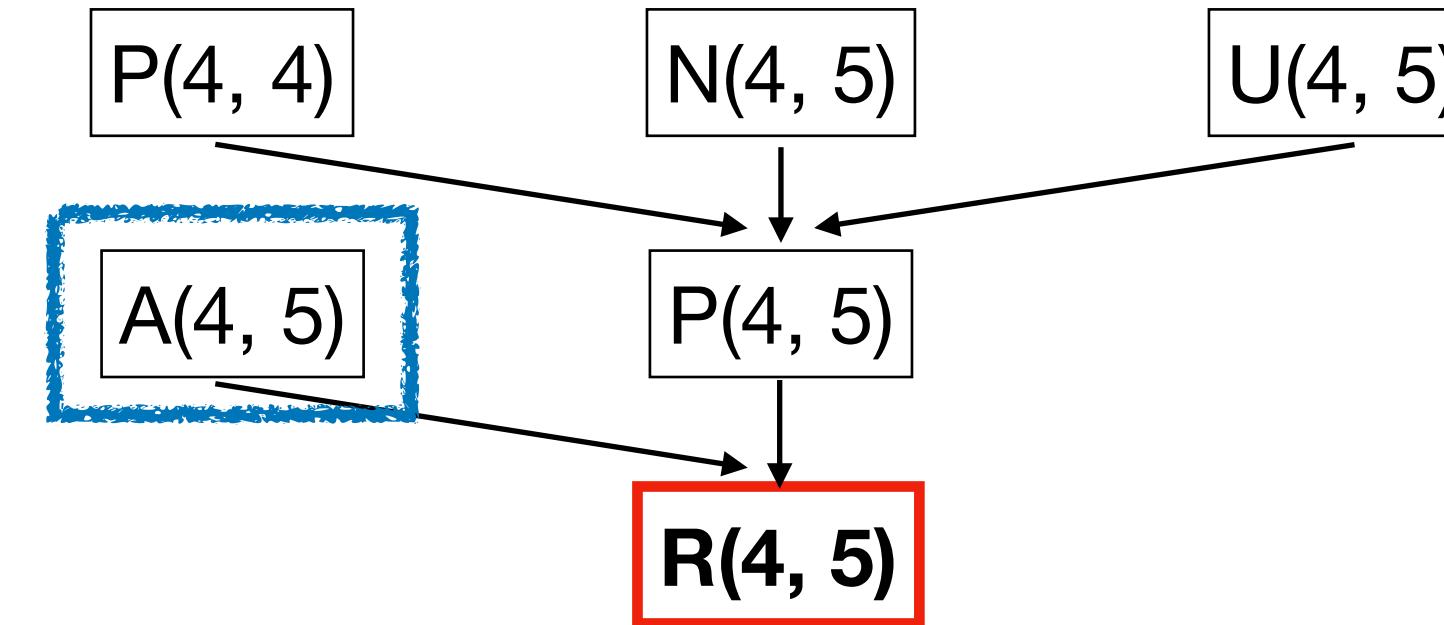
# Marginal Inference



$$Pr(R(4,5)) = Pr(R(4,5) | A(4,5), P(4,5))$$

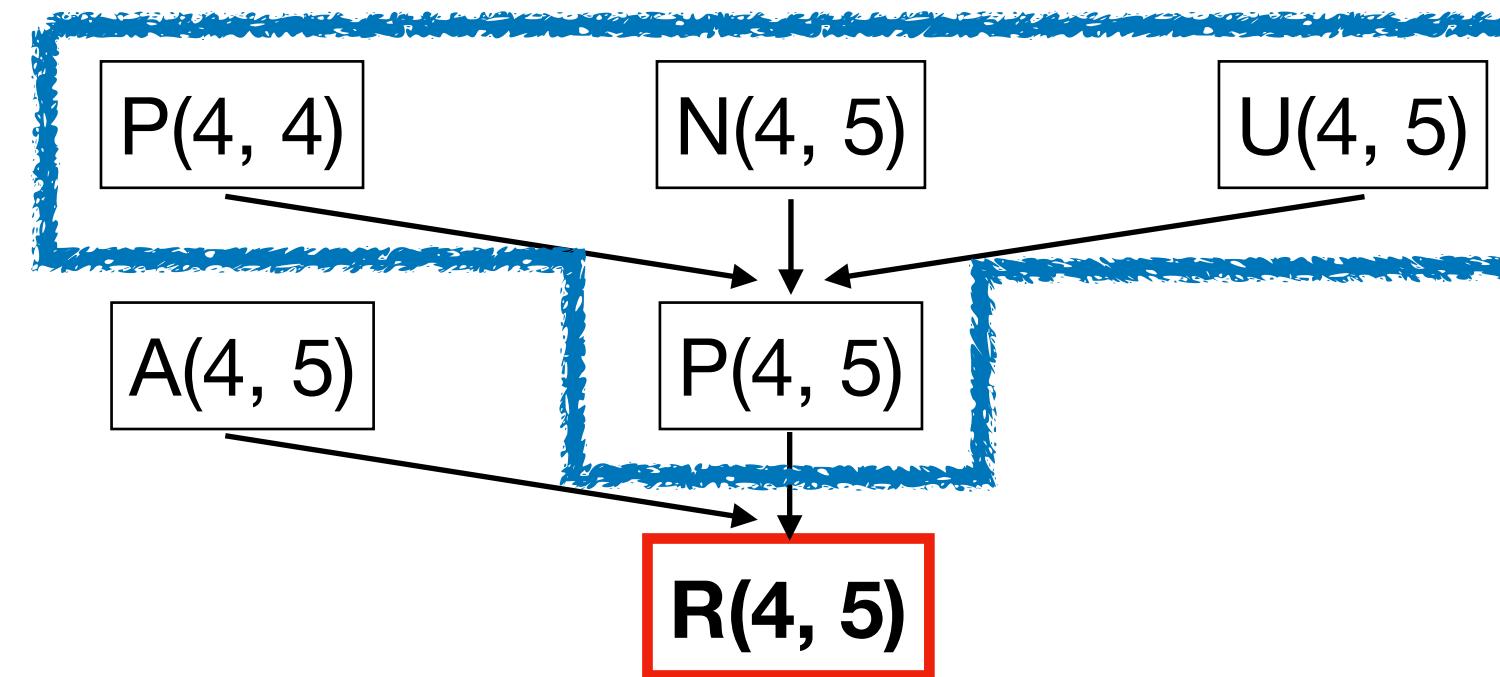
X

# Marginal Inference



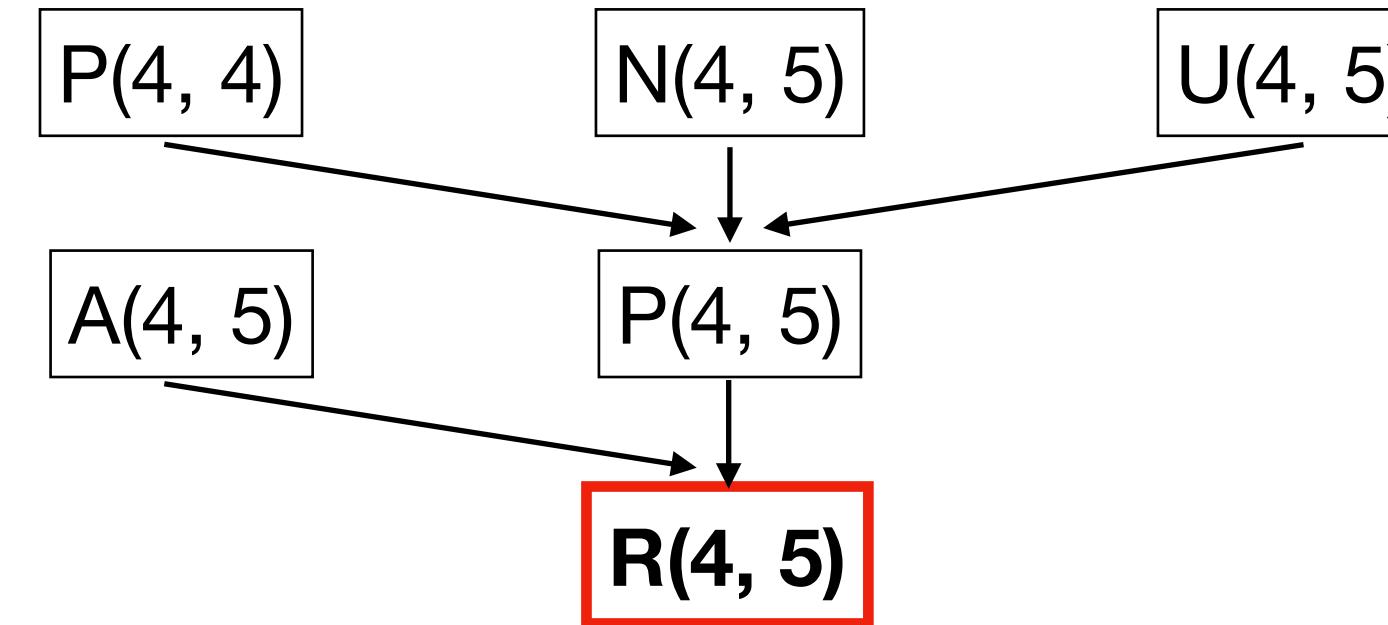
$$\begin{aligned} Pr(R(4,5)) &= Pr(R(4,5) \mid A(4,5), P(4,5)) \\ &\times \boxed{Pr(A(4,5))} \\ &\times \end{aligned}$$

# Marginal Inference



$$\begin{aligned} Pr(R(4,5)) &= Pr(R(4,5) \mid A(4,5), P(4,5)) \\ &\quad \times Pr(A(4,5)) \\ &\quad \times \boxed{Pr(P(4,5) \mid \dots)} \\ &\quad \times \end{aligned}$$

# Marginal Inference



$$\begin{aligned}Pr(R(4,5)) &= Pr(R(4,5) \mid A(4,5), P(4,5)) \\&\quad \times Pr(A(4,5)) \\&\quad \times Pr(P(4,5) \mid \dots) \\&\quad \times \dots \\&= 0.398\end{aligned}$$

by an off-the-shelf marginal  
inference solver

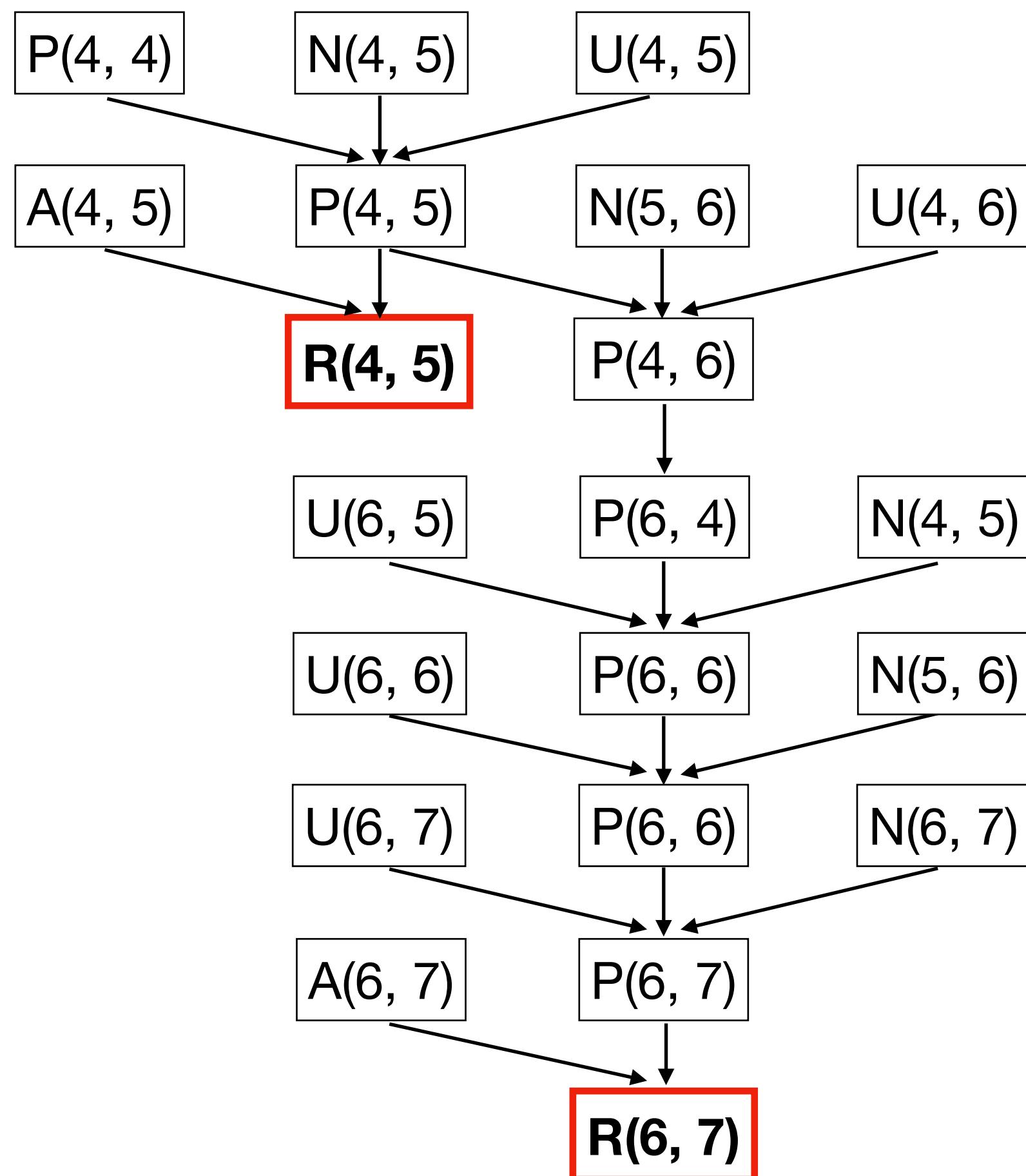
# Alarm Ranking

```
public class RequestHandler {  
    private FtpRequest request;  
  
    public FtpRequest getRequest() {  
        return request; //L0  
    }  
  
    public void close() {  
        synchronized (this) { //L1  
            if (isClosed) return; //L2  
            isClosed = true; //L3  
        }  
        controlSocket.close(); //L4  
        controlSocket = null; //L5  
        request.clear(); //L6  
        request = null; //L7  
    }  
}
```

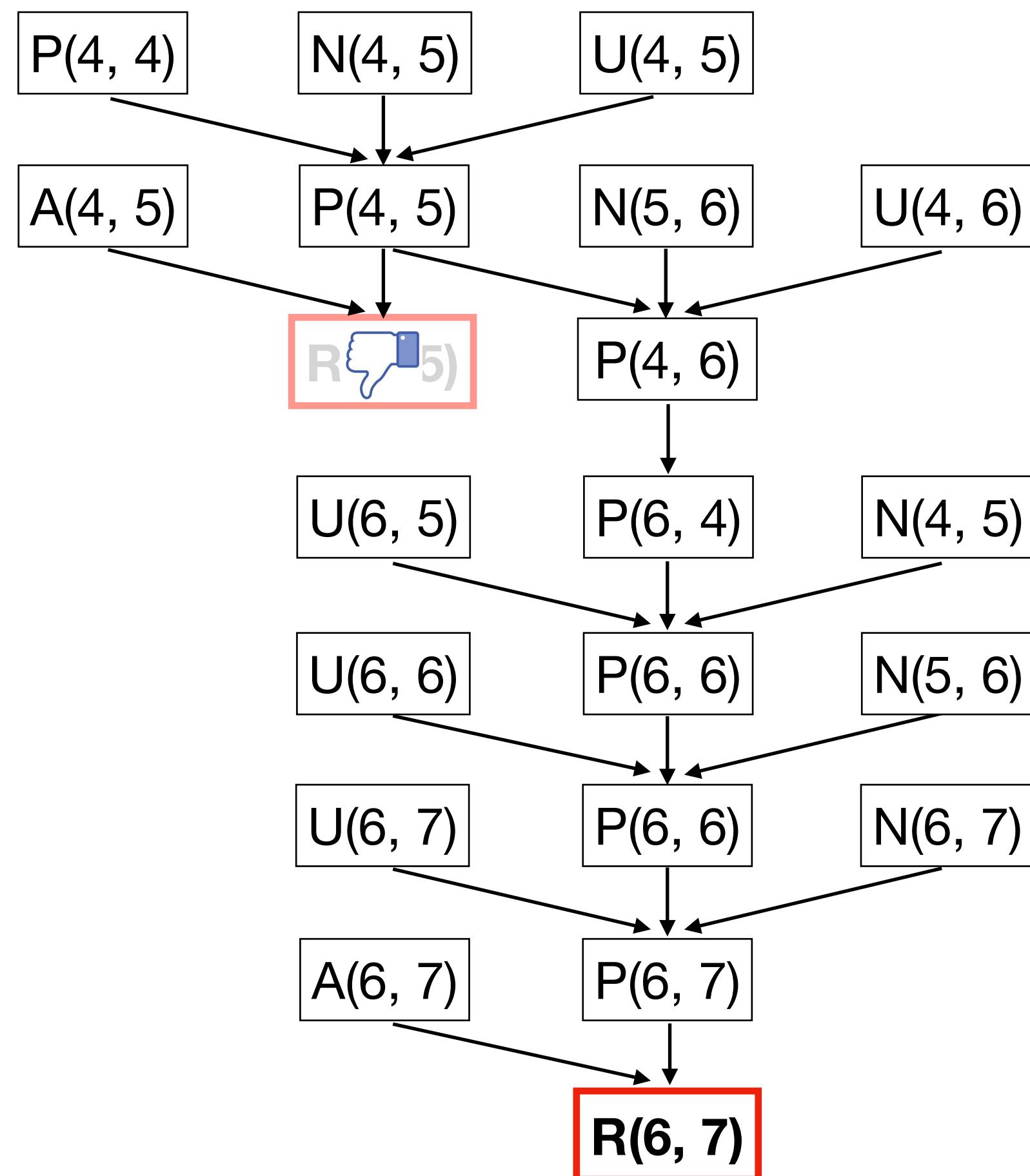
Ranking	Alarm	Confidence	
1	R(4, 5)	0.398	
2	R(5, 5)	0.378	
3	R(6, 7)	0.324	
4	R(7, 7)	0.308	
5	R(0, 7)	0.279	

**Q: What are the probabilities of the other alarms when R(4,5) is false?**

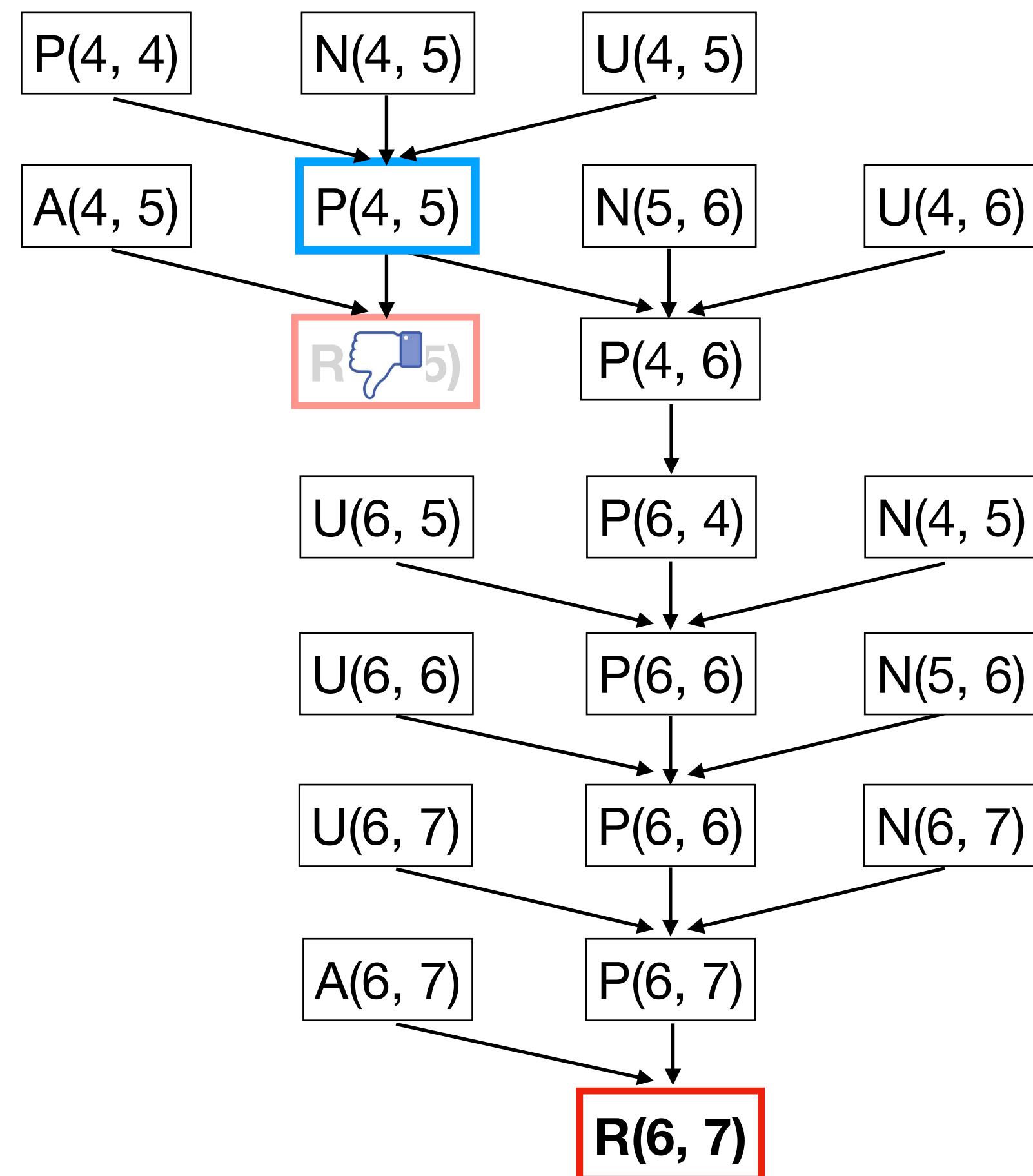
# Probability of Alarms



# Probability of Alarms

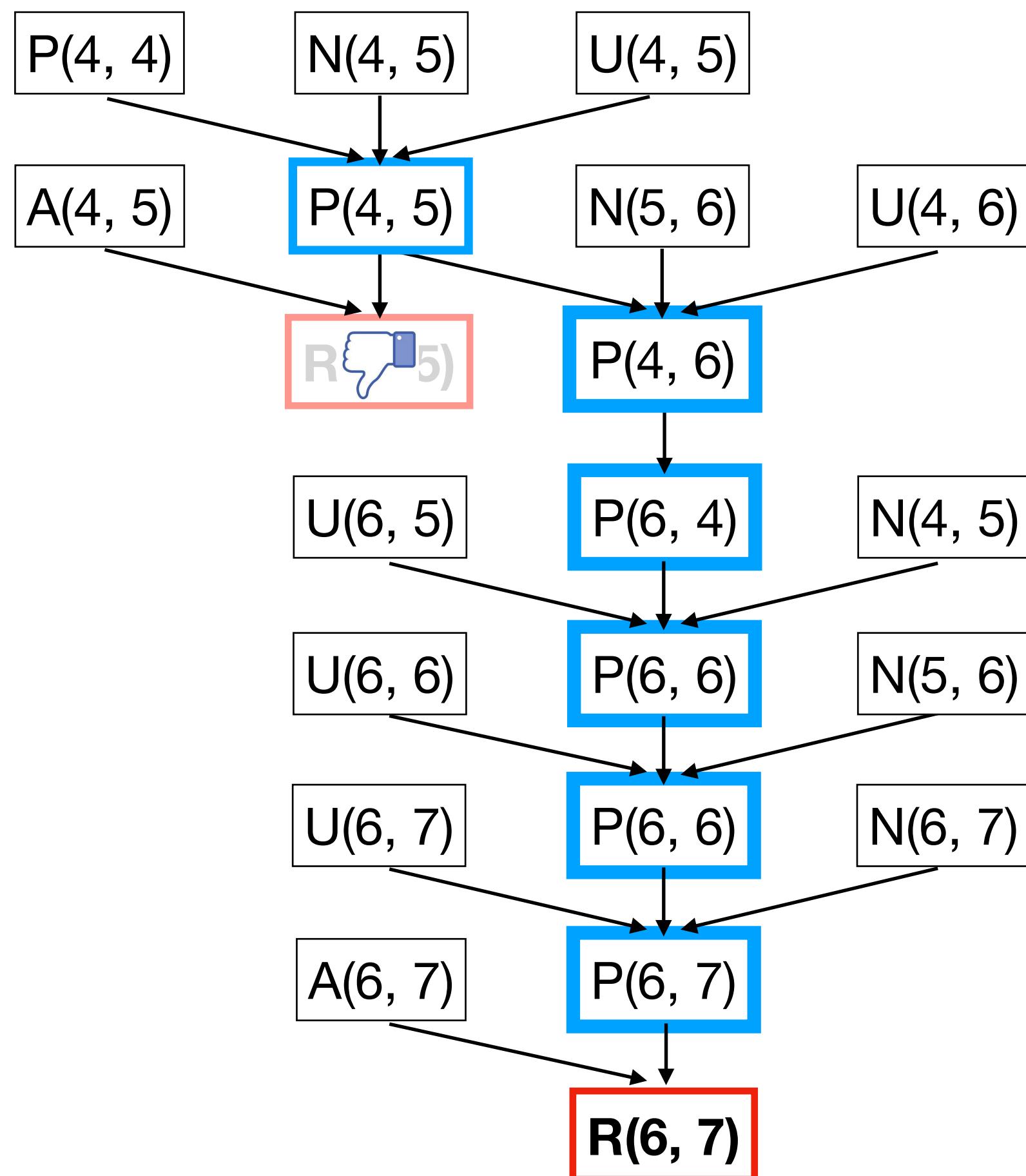


# Probability of Alarms



$$\begin{aligned}Pr(P(4,5) | \neg R(4,5)) \\= Pr(\neg R(4,5) | P(4,5)) * \\Pr(P(4,5)) / Pr(\neg R(4,5)) \\= 0.03\end{aligned}$$

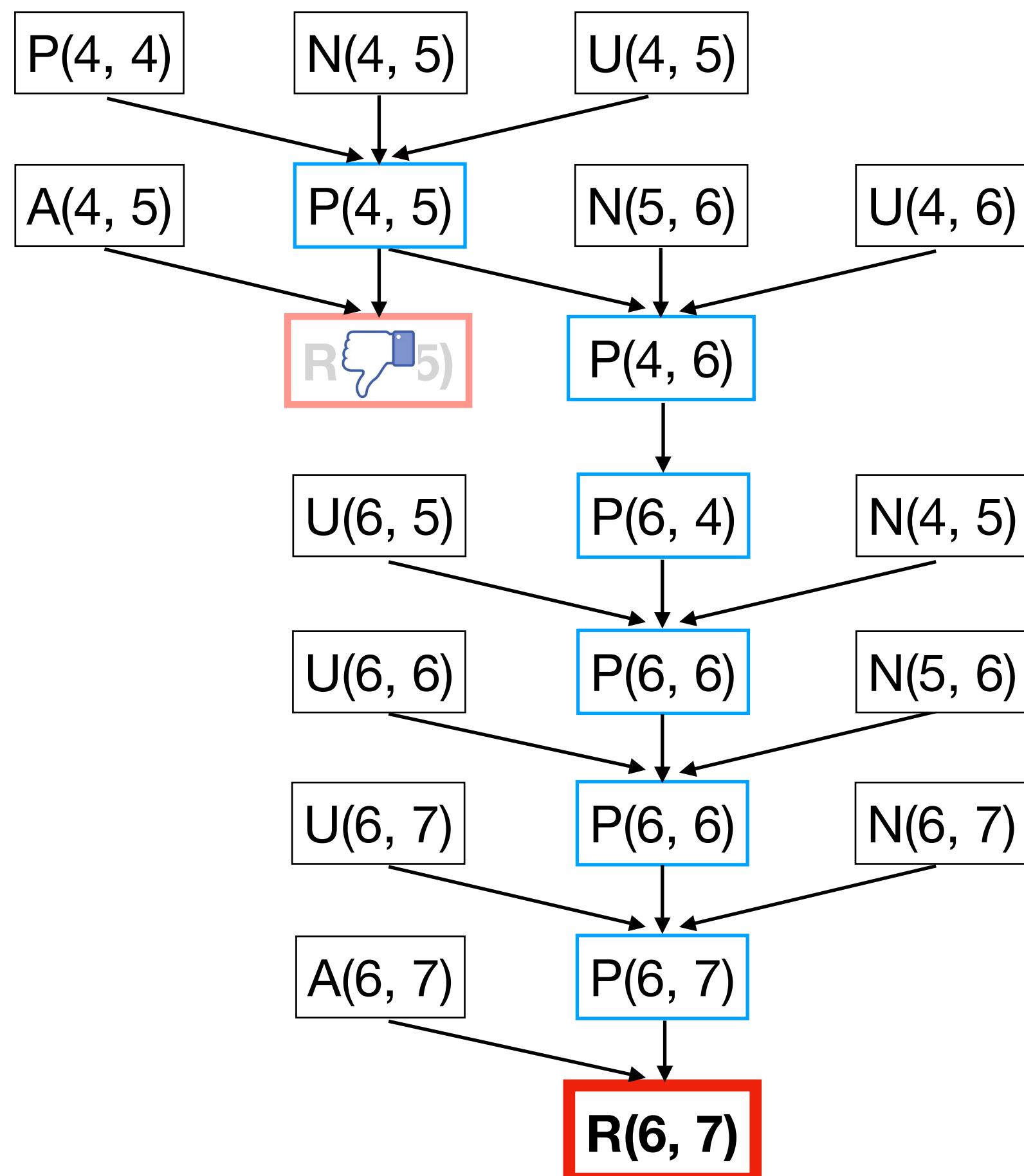
# Probability of Alarms



$$\begin{aligned}Pr(P(4,5) | \neg R(4,5)) \\= Pr(\neg R(4,5) | P(4,5)) * \\Pr(P(4,5)) / Pr(\neg R(4,5)) \\= 0.03\end{aligned}$$

By Bayes's Rule:  
 $Pr(A|B) = Pr(B|A) * Pr(A) / Pr(B)$

# Probability of Alarms



$$\begin{aligned}Pr(P(4,5) | \neg R(4,5)) &= Pr(\neg R(4,5) | P(4,5)) * \\Pr(P(4,5)) / Pr(\neg R(4,5)) &= 0.03\end{aligned}$$

By Bayes's Rule:  
 $Pr(A|B) = Pr(B|A) * Pr(A) / Pr(B)$

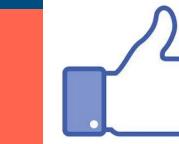
$$\begin{aligned}Pr(R(6,7) | \neg R(4,5)) &= Pr(R(6,7) | P(4,5)) * \\Pr(P(4,5) | \neg R(4,5)) &= 0.03\end{aligned}$$

# New Alarm Ranking

Ranking	Alarm	Confidence	
1	R(4, 5)	0.398	
2	R(5, 5)	0.378	
3	R(6, 7)	0.324	
4	R(7, 7)	0.308	
5	R(0, 7)	0.279	

# New Alarm Ranking

Ranking	Alarm	Confidence
1	R(0, 7)	0.279
2	R(5, 5)	0.035
3	R(6, 7)	0.030
4	R(7, 7)	0.028
5	R(4, 5)	0

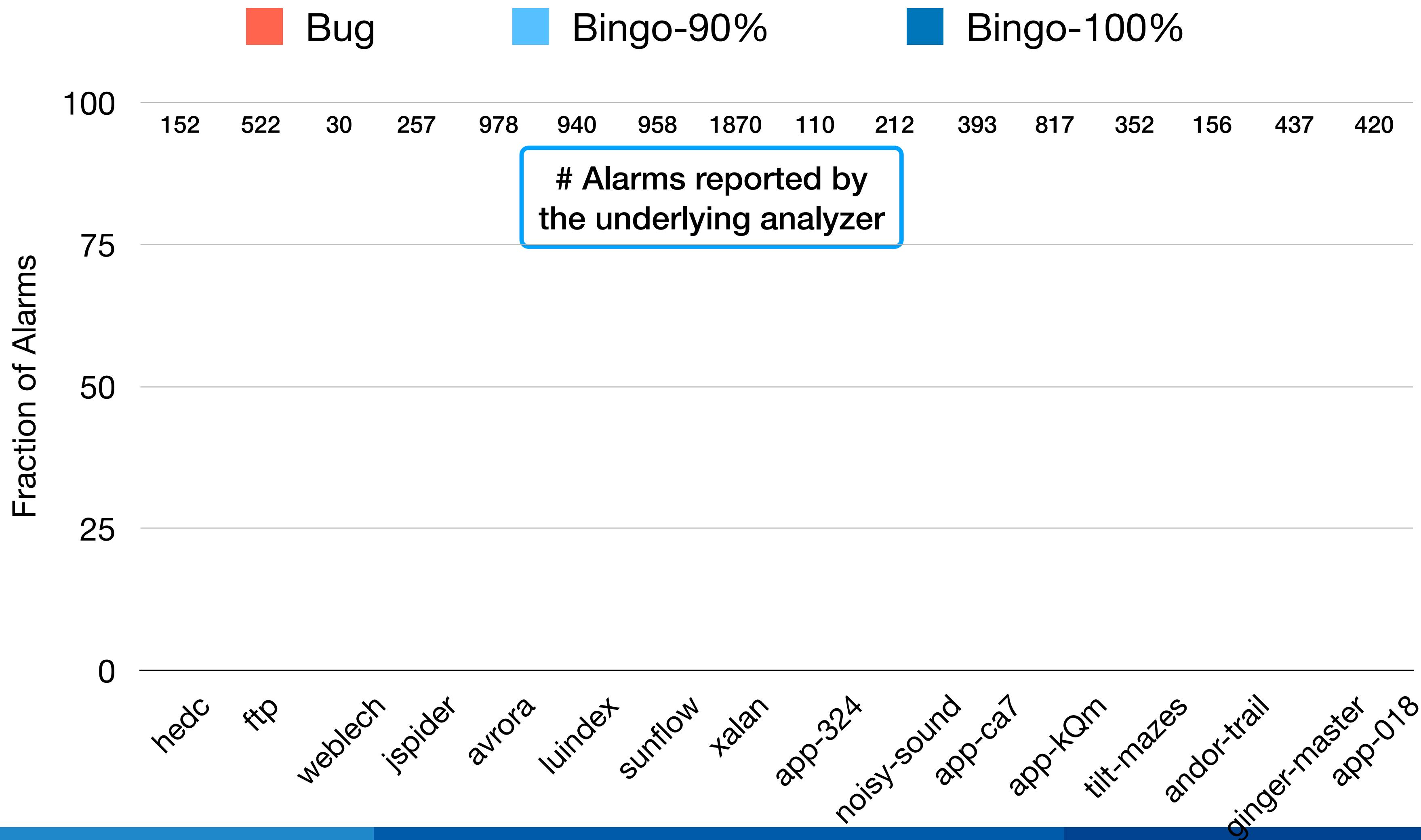


# Effectiveness

40–616KLOC JAVA Programs  
Datarace and Privacy leak analyses

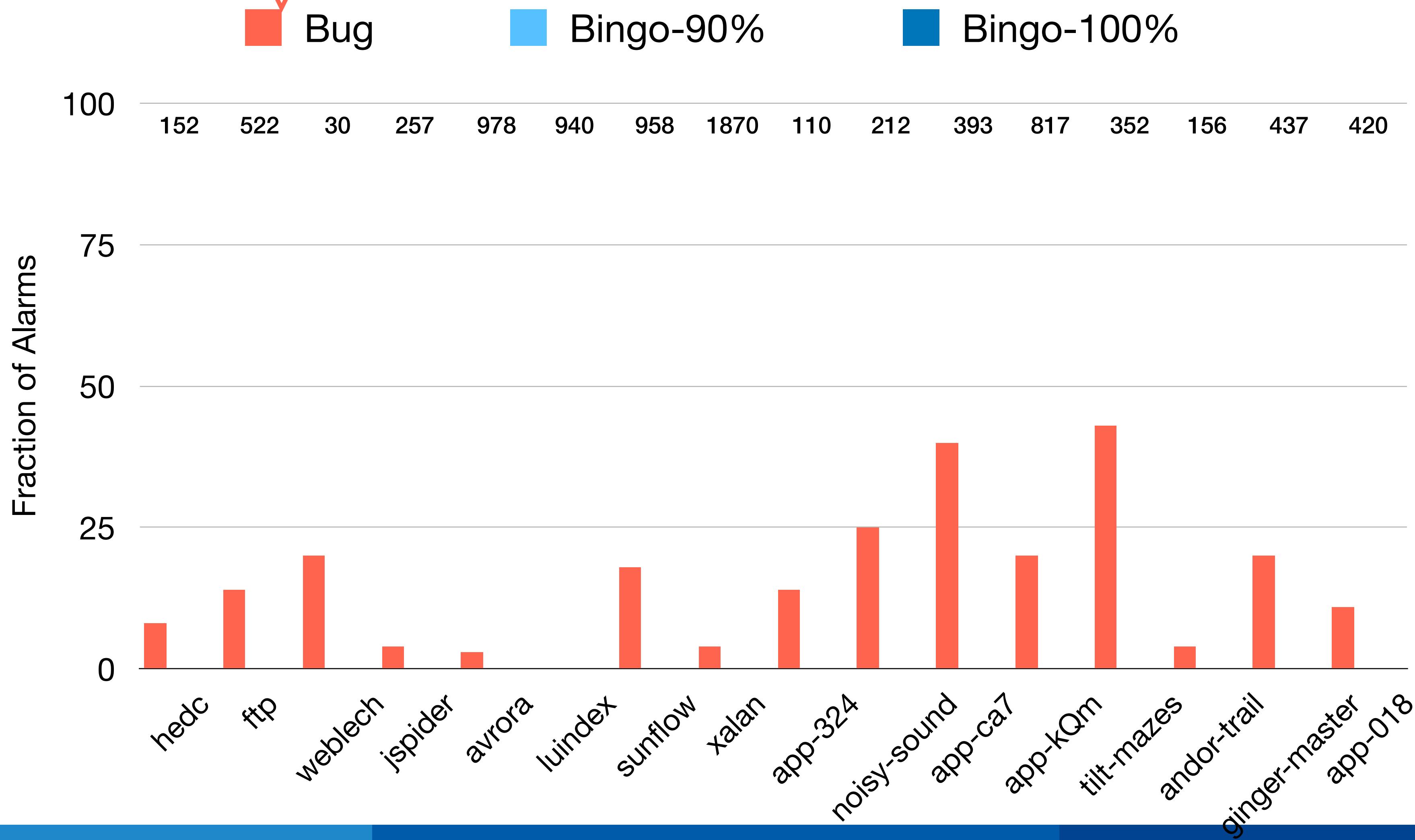
hedc ftp weblech jspider avrora luindex sunflow xalan app-324 noisy-sound app-ca1 app-kQm tilt-mazes andor-trail ginger-master app-018

# Effectiveness

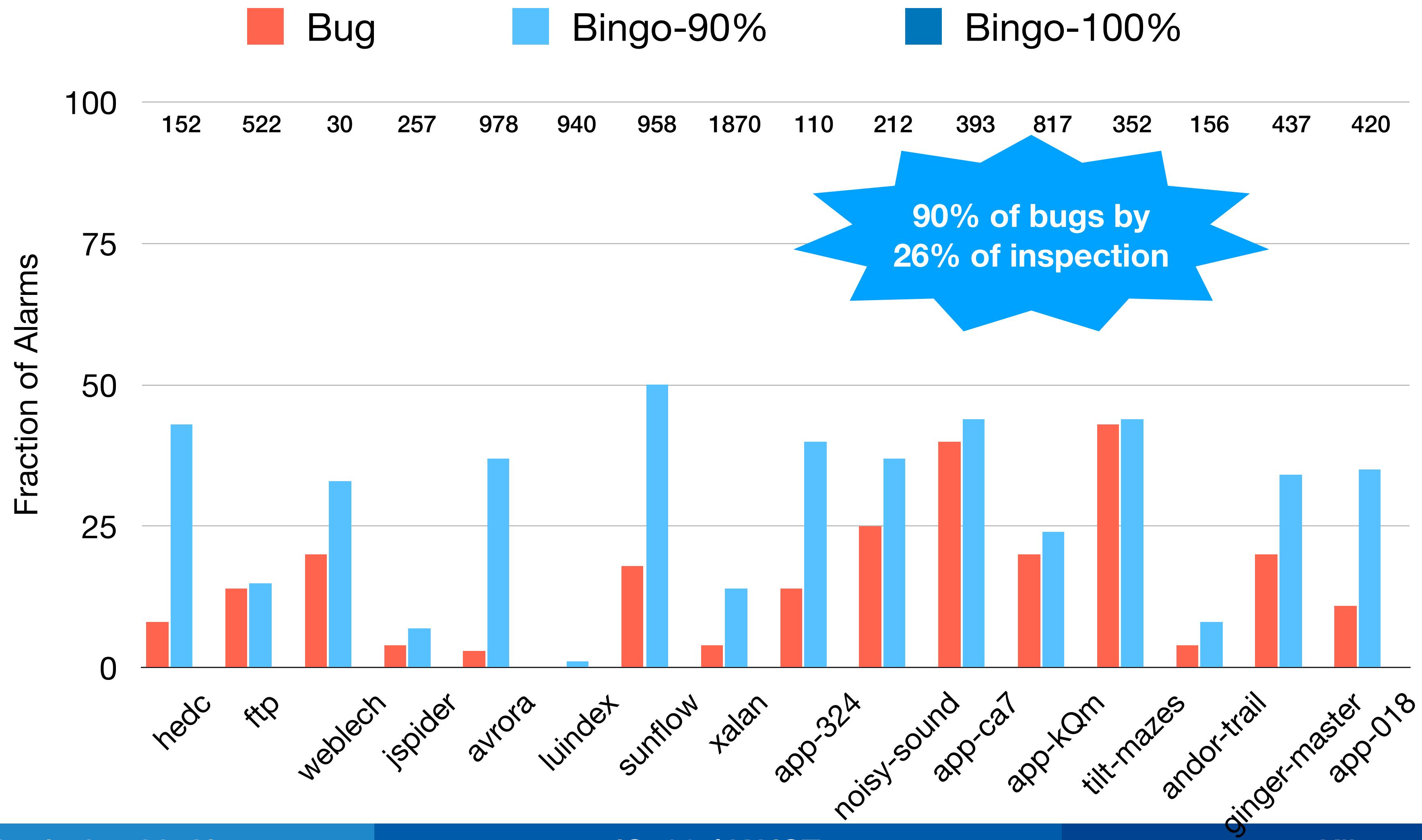


# Effectiveness

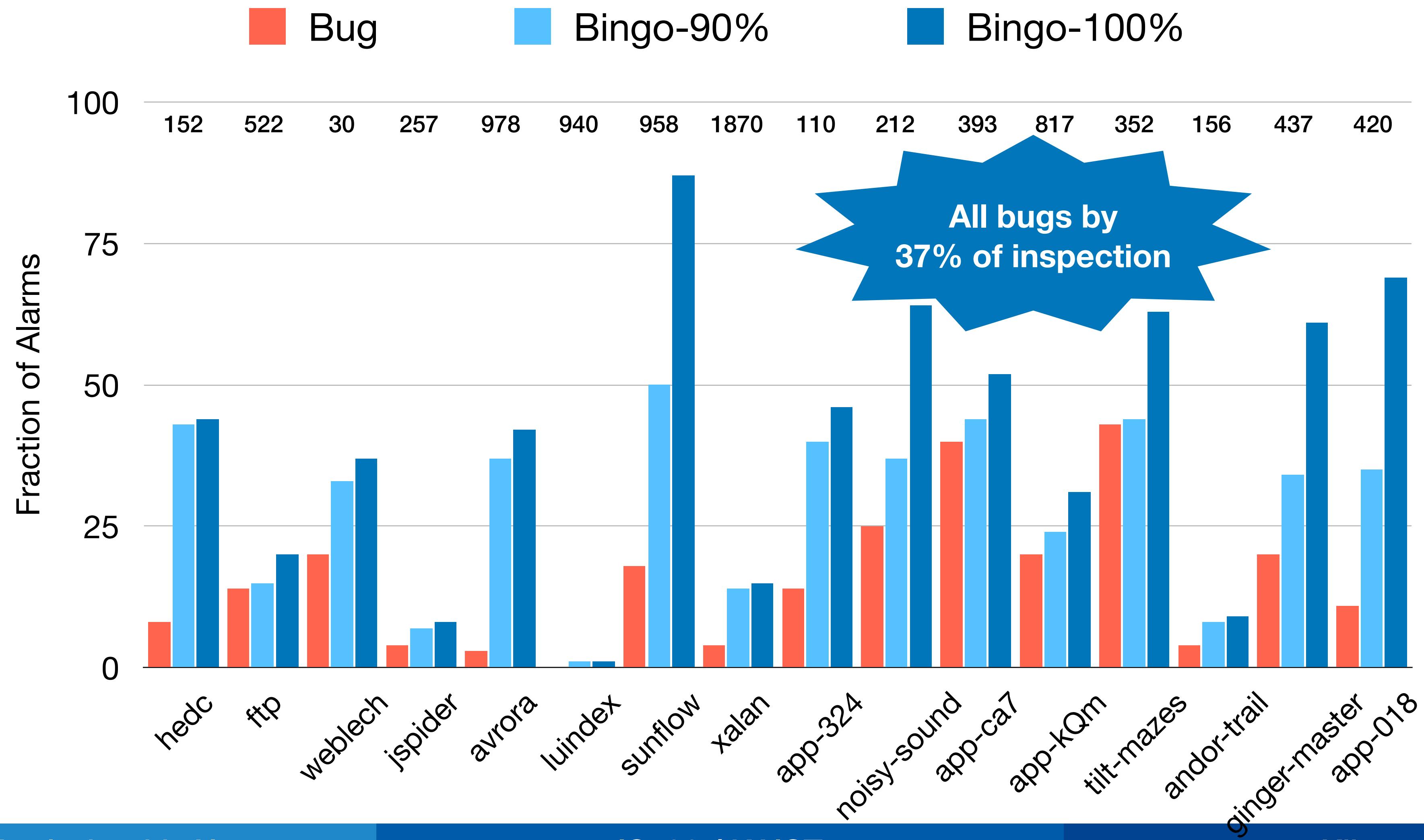
Only a few of them are real bugs (12%)



# Effectiveness



# Effectiveness



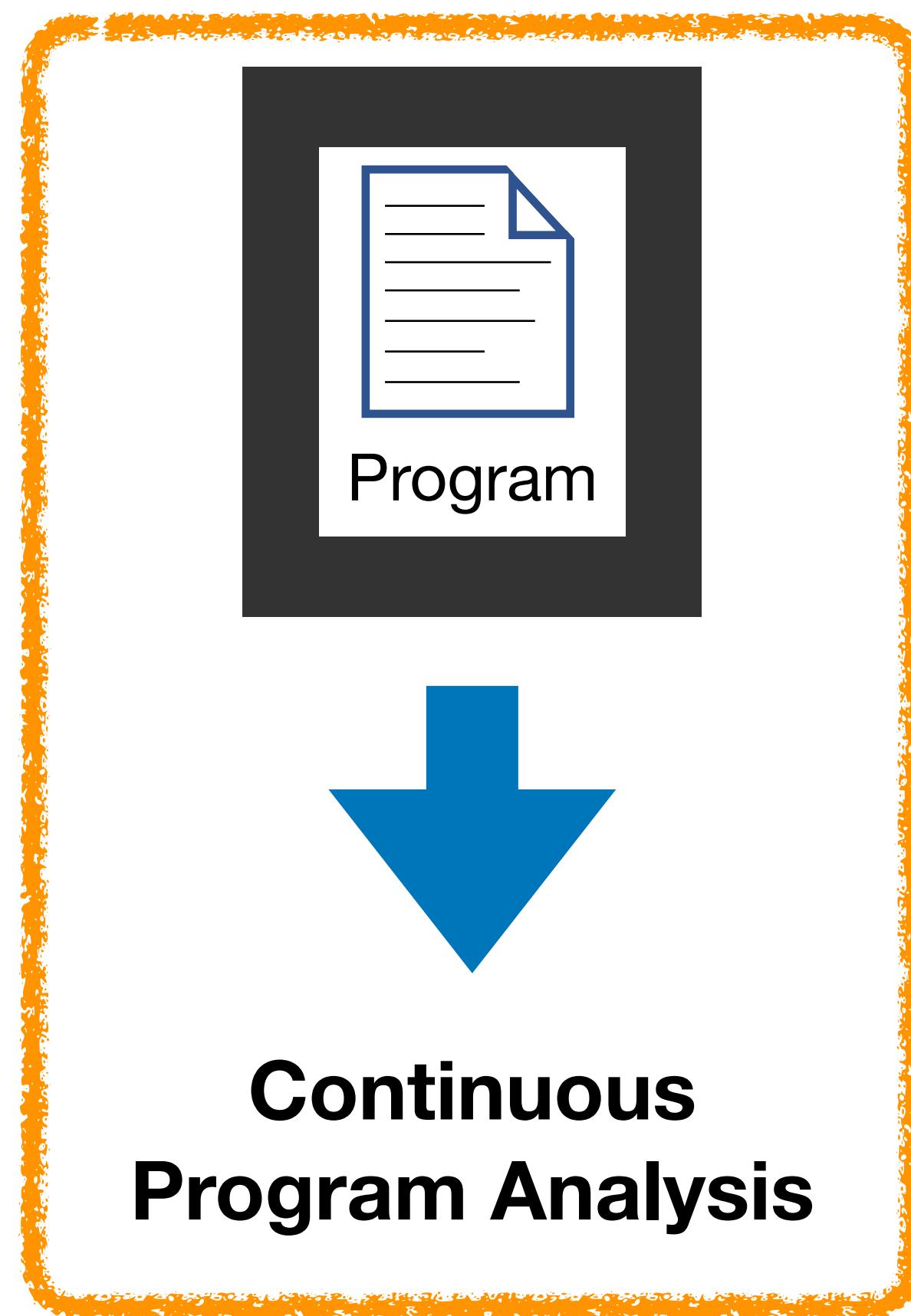
# Outline



**Adaptive  
Program Analysis**



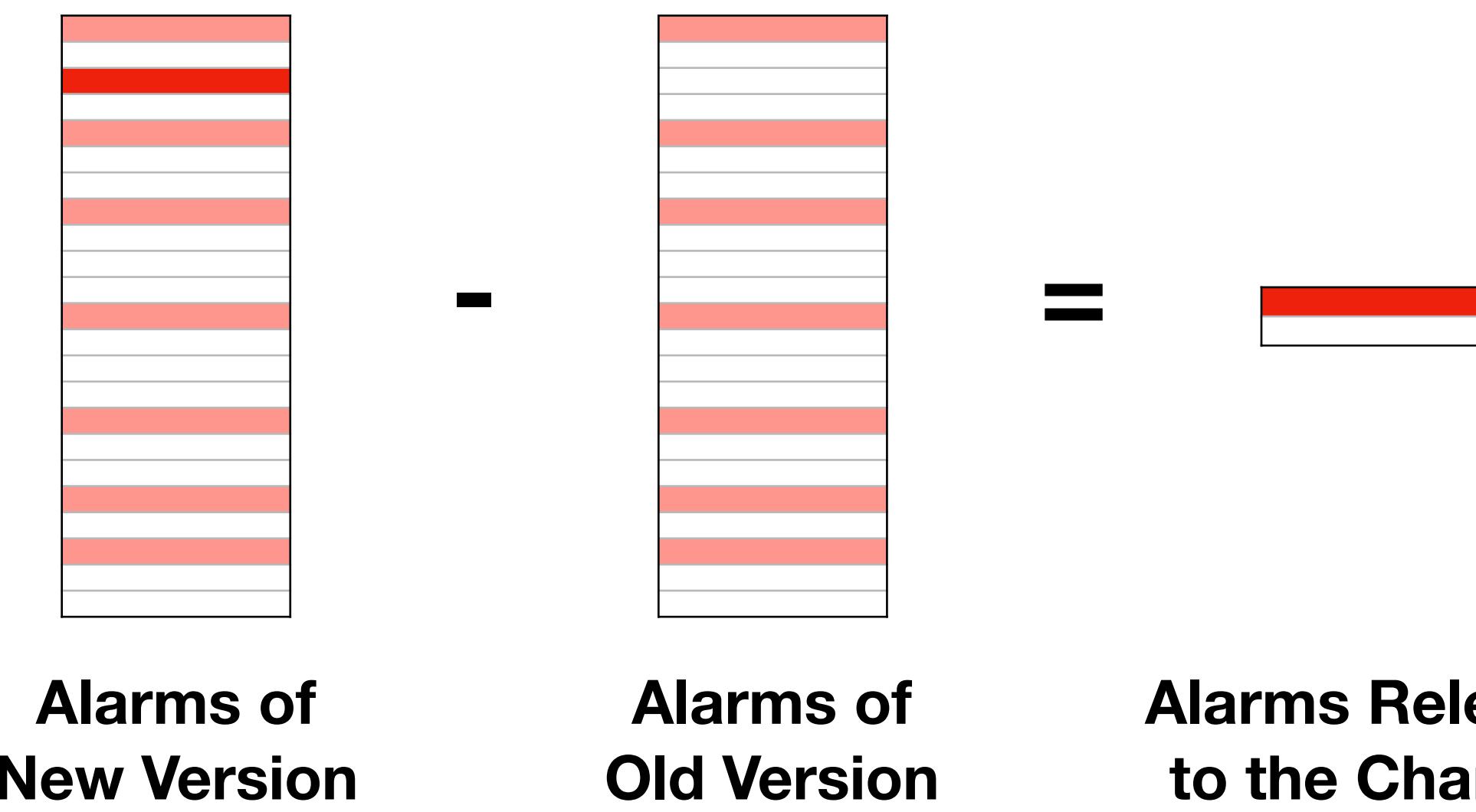
**Interactive  
Program Analysis**



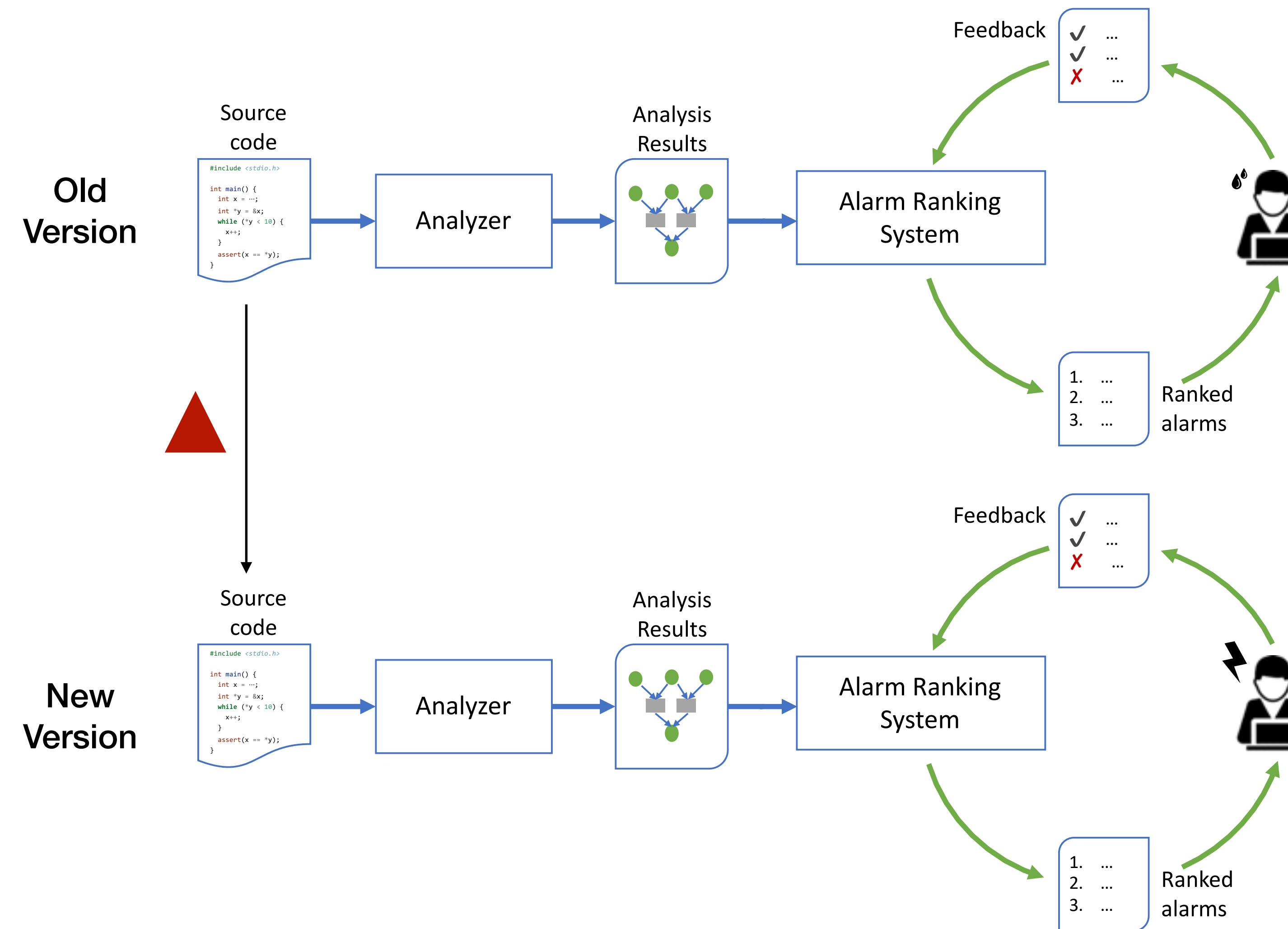
**Continuous  
Program Analysis**

# Drake: Continuous Alarm Masking System

[PLDI'19]



# Batch-mode Reasoning



# Continuous Reasoning

*"We only display results for most analyses on **changed lines** by default; this keeps analysis results*

**relevant** to the code review at hand", - Google, 2015

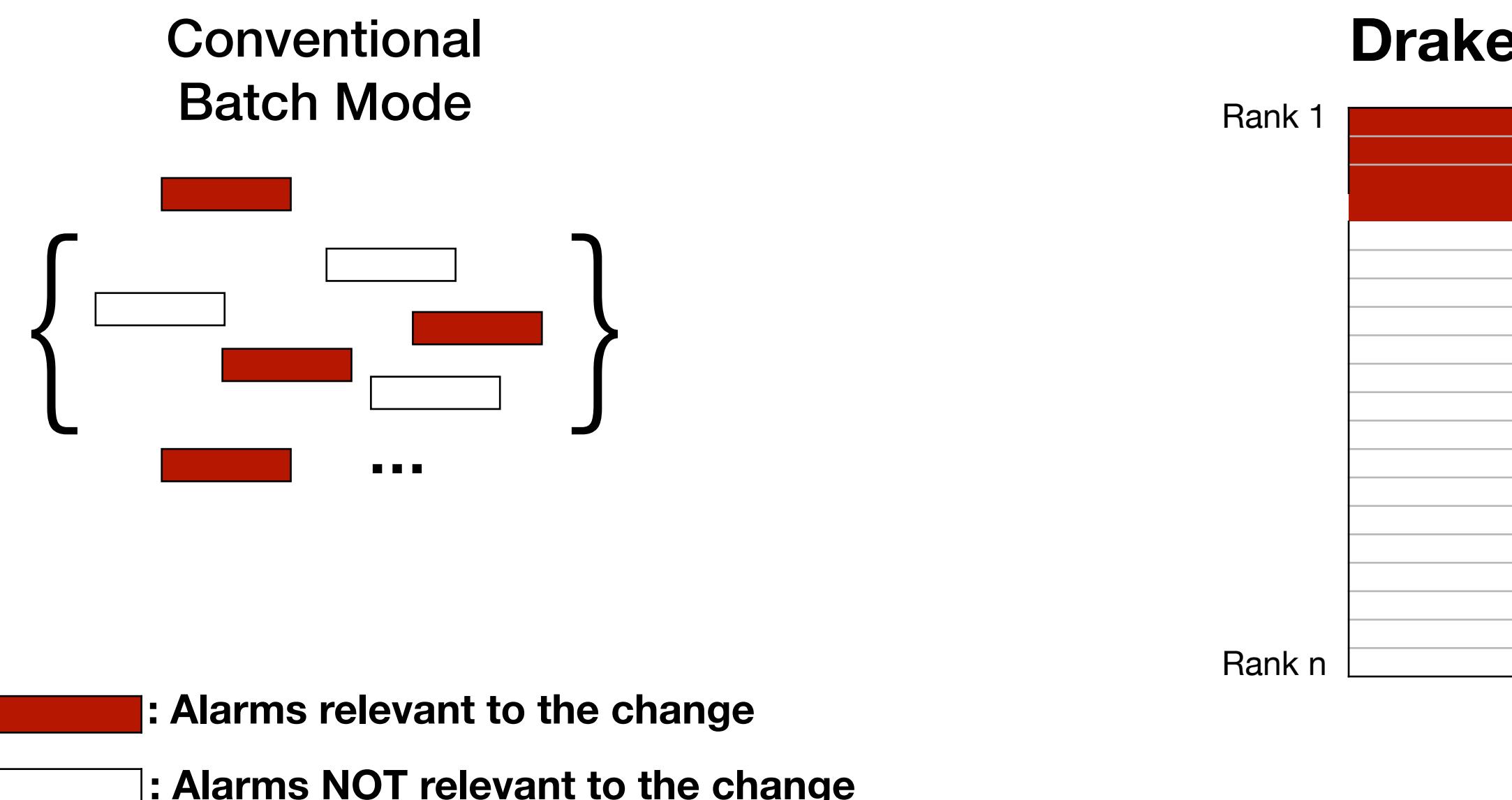
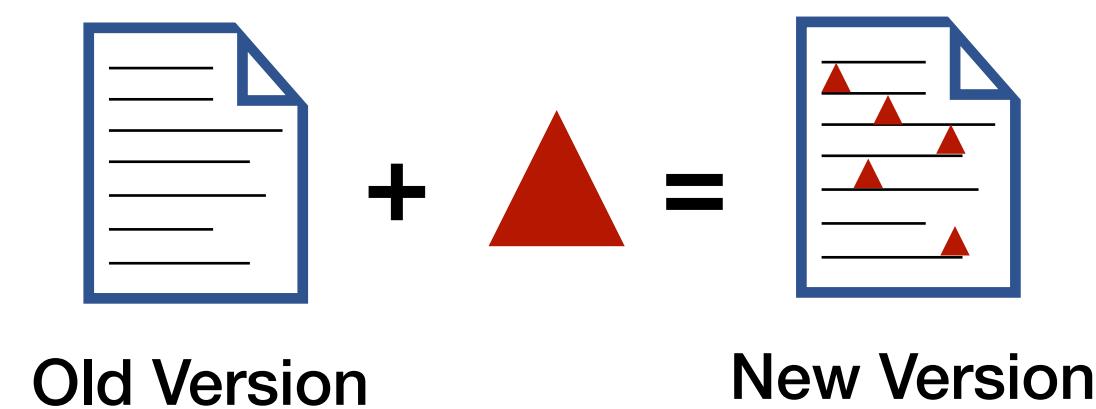
*"The vast majority of Infer's impact to this point is attributable to **continuous reasoning at diff time**", - Facebook, 2016*

*"... is the ability to analyze a **changelist (a.k.a. a commit)** rather than the entire codebase. This functionality can help developers assess the quality and impact of a change ...", - Microsoft, 2016*

*"In order to realize the goal, verification must continue to work with low effort **as developers change the code**. ... Neither of these approaches would work for Amazon as s2n is under **continuous development**.", - Amazon, 2018*

# Goal

- Prioritize alarms by their **relevance to the change**



# Example

## Old Version

```
1: x = input();
2: y = input();
3: x = opaque_dec(x);
4: y = opaque_dec(y);
5: x++; // Alarm ✓
6: y++; // Alarm ✓
```

x and y can be any integers

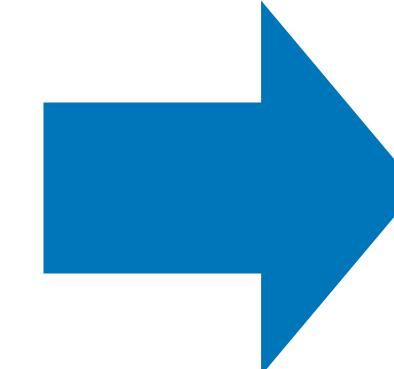
Opaque, but “--” actually

Integer overflow alarms at 5 & 6

# Example

## Old Version

```
1: x = input();
2: y = input();
3: x = opaque_dec(x);
-4: y = opaque_dec(y);
5: x++; // Alarm ✓
6: y++; // Alarm ✓
```



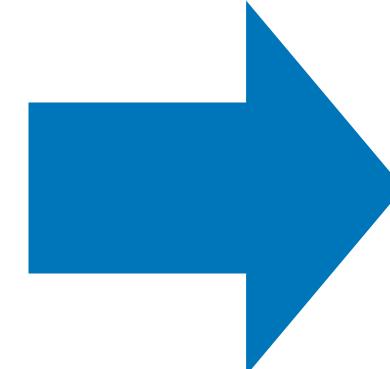
## New Version

```
1: x = input();
2: y = input();
3: x = opaque_dec(x);
+4: y = identity(y);
5: x++; // Alarm ✓
6: y++; // Alarm 🐞
```

# Example

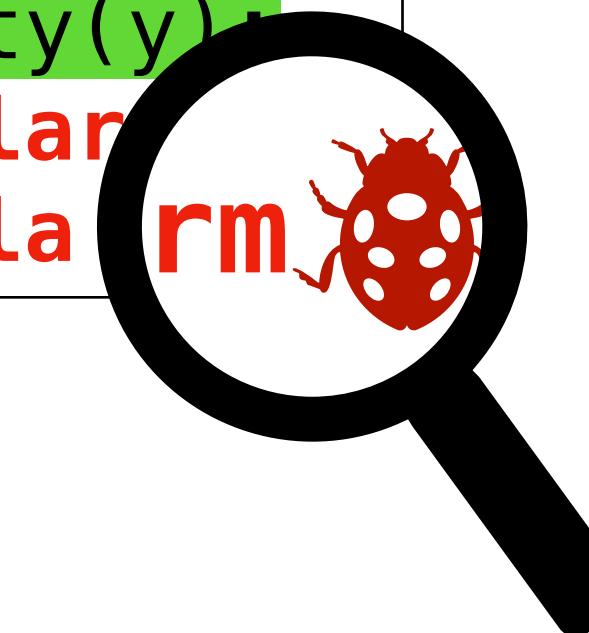
## Old Version

```
1: x = input();
2: y = input();
3: x = opaque_dec(x);
-4: y = opaque_dec(y);
5: x++; // Alarm ✓
6: y++; // Alarm ✓
```



## New Version

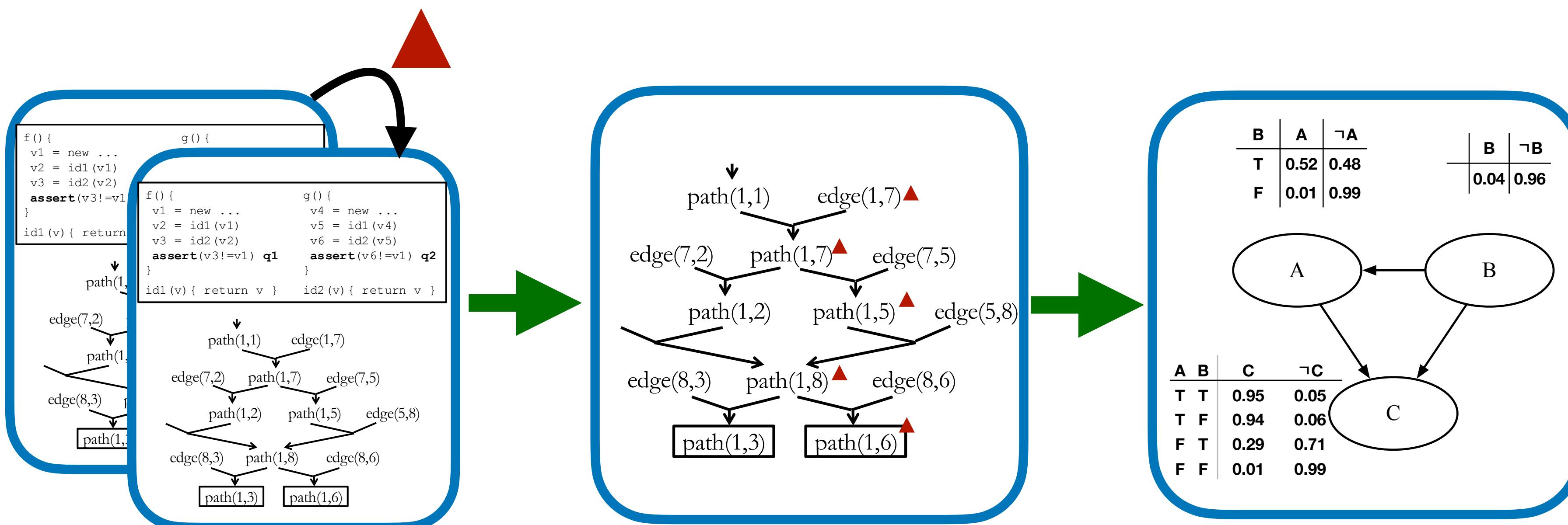
```
1: x = input();
2: y = input();
3: x = opaque_dec(x);
+4: y = identity(y);
5: x++; // Alarm ✓
6: y++; // Alarm ✓
```



Q: How to emphasize the alarm at 6  
that is relevant to this change?

# Key Idea

## Differential derivation + Bayesian inference



Program Analysis Results  
of Two Versions

Differential Derivation

Probabilistic Model  
(Bayesian Network)

# Program Analysis

## Analysis Inputs:

$\text{Input}(p_1)$ ,  $\text{Edge}(p_1, p_2)$ ,  $\text{Inc}(p_1)$

Read an input at  
program point  $p_1$

Immediate data flow  
from  $p_1$  to  $p_2$

Increment expression  
at  $p_1$

## Analysis Rules:

**R<sub>1</sub>:**  $\text{Path}(p_1, p_2) :- \text{Edge}(p_1, p_2).$

**R<sub>2</sub>:**  $\text{Path}(p_1, p_3) :- \text{Path}(p_1, p_2), \text{Edge}(p_2, p_3).$

**R<sub>3</sub>:**  $\text{Overflow}(p_3) :- \text{Input}(p_1), \text{Path}(p_1, p_3), \text{Inc}(p_3).$

# Program Analysis

**Analysis Inputs:**

$\text{Input}(p_1)$ ,  $\text{Edge}(p_1, p_2)$ ,  $\text{Inc}(p_1)$

**Analysis Outputs:**

$\text{Path}(p_1, p_2)$ ,  $\text{Overflow}(p_1)$

Transitive data flow  
from  $p_1$  to  $p_2$

Integer overflow at  $p_1$

$R_2$ :  $\text{Path}(p_1, p_3) :- \text{Path}(p_1, p_2), \text{Edge}(p_2, p_3).$

$R_3$ :  $\text{Overflow}(p_3) :- \text{Input}(p_1), \text{Path}(p_1, p_3), \text{Inc}(p_3).$

# Program Analysis

**Analysis Inputs:**

$\text{Input}(p_1)$ ,  $\text{Edge}(p_1, p_2)$ ,  $\text{Inc}(p_1)$

**Analysis Outputs:**

$\text{Path}(p_1, p_2)$ ,  $\text{Overflow}(p_1)$

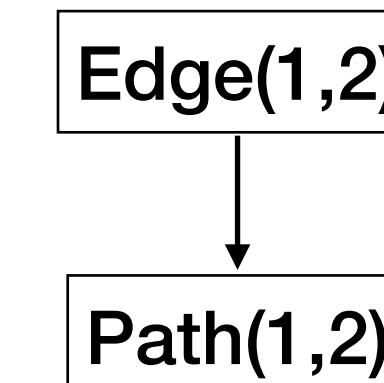
**Analysis Rules:**

**R<sub>1</sub>:**  $\text{Path}(p_1, p_2) :- \text{Edge}(p_1, p_2).$

**R<sub>2</sub>:**  $\text{Path}(p_1, p_3) :- \text{Path}(p_1, p_2), \text{Edge}(p_2, p_3).$

**R<sub>3</sub>:**  $\text{Overflow}(p_3) :- \text{Input}(p_1), \text{Path}(p_1, p_3), \text{Inc}(p_3).$

```
1: x = input();  
2: x = opaque_dec(x);  
3: x++; // Alarm
```

**Derivation**

# Program Analysis

**Analysis Inputs:**

$\text{Input}(p_1)$ ,  $\text{Edge}(p_1, p_2)$ ,  $\text{Inc}(p_1)$

**Analysis Outputs:**

$\text{Path}(p_1, p_2)$ ,  $\text{Overflow}(p_1)$

**Analysis Rules:**

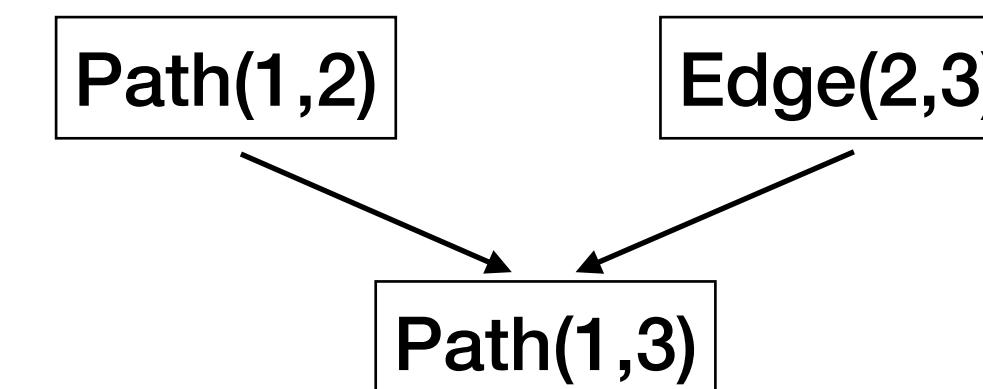
$R_1: \text{Path}(p_1, p_2) :- \text{Edge}(p_1, p_2).$

$R_2: \text{Path}(p_1, p_3) :- \text{Path}(p_1, p_2), \text{Edge}(p_2, p_3).$

$R_3: \text{Overflow}(p_3) :- \text{Input}(p_1), \text{Path}(p_1, p_3), \text{Inc}(p_3).$

**Derivation**

```
1: x = input();  
2: x = opaque_dec(x);  
3: x++; // Alarm
```



# Program Analysis

**Analysis Inputs:**

$\text{Input}(p_1)$ ,  $\text{Edge}(p_1, p_2)$ ,  $\text{Inc}(p_1)$

**Analysis Outputs:**

$\text{Path}(p_1, p_2)$ ,  $\text{Overflow}(p_1)$

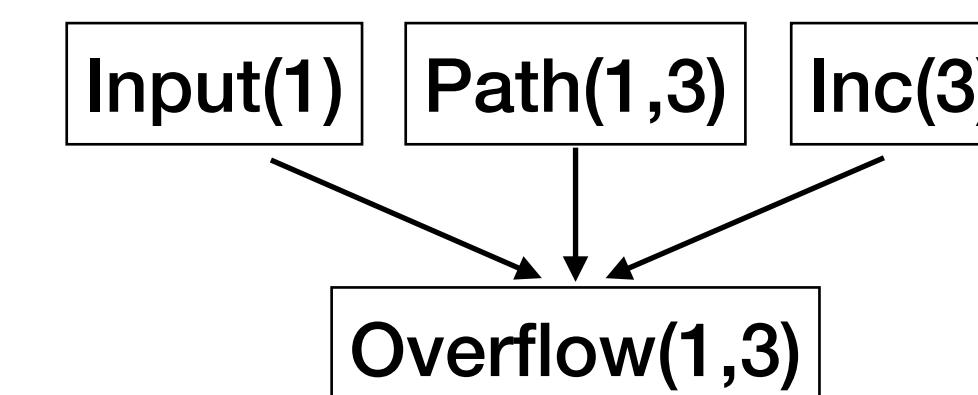
**Analysis Rules:**

**R<sub>1</sub>:**  $\text{Path}(p_1, p_2) :- \text{Edge}(p_1, p_2).$

**R<sub>2</sub>:**  $\text{Path}(p_1, p_3) :- \text{Path}(p_1, p_2), \text{Edge}(p_2, p_3).$

**R<sub>3</sub>:**  $\text{Overflow}(p_3) :- \text{Input}(p_1), \text{Path}(p_1, p_3), \text{Inc}(p_3).$

```
1: x = input();  
2: x = opaque_dec(x);  
3: x++; // Alarm
```

**Derivation**

# Differential Derivation

```
1: x = input();
2: y = input();
3: x = opaque_dec(x);
+4: y = identity(y);
5: x++; // Alarm ✓
6: y++; // Alarm 🐞
```



# Differential Derivation

```
1: x = input();
2: y = input();
3: x = opaque_dec(x);
+4: y = identity(y);
5: x++; // Alarm ✓
6: y++; // Alarm 🐞
```



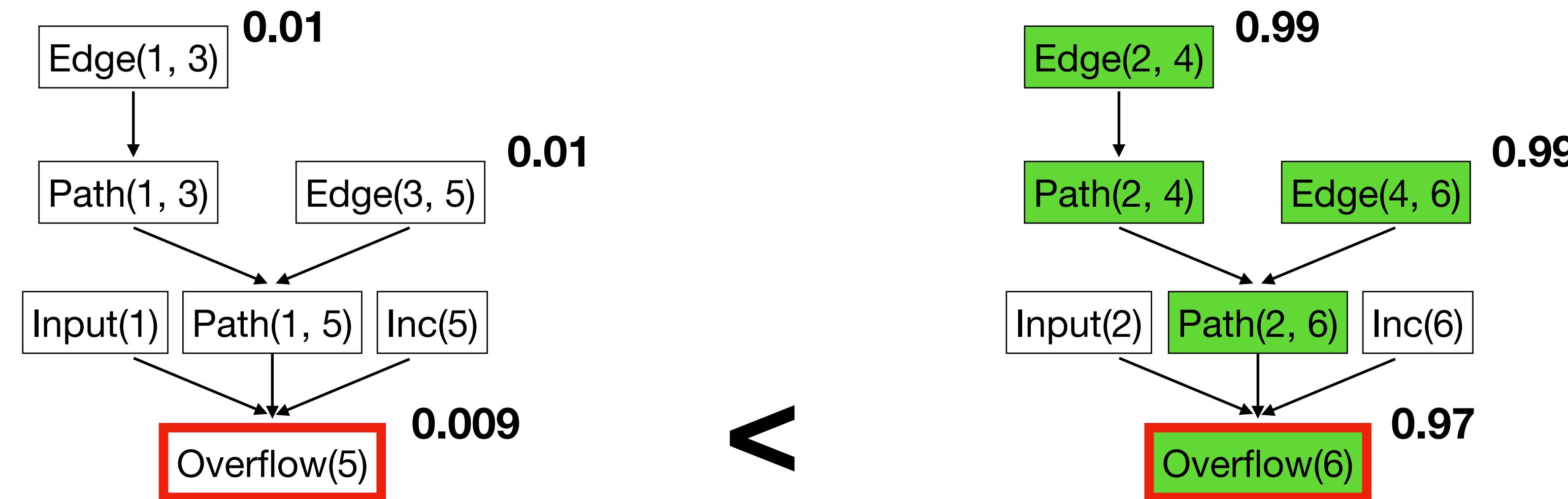
# Relevance Score

```
1: x = input();
2: y = input();
3: x = opaque_dec(x);
+4: y = identity(y);
5: x++; // Alarm ✓
6: y++; // Alarm 🐞
```



# Relevance Score

```
1: x = input();
2: y = input();
3: x = opaque_dec(x);
+4: y = identity(y);
5: x++; // Alarm ✓
6: y++; // Alarm 🐞
```



# Effectiveness

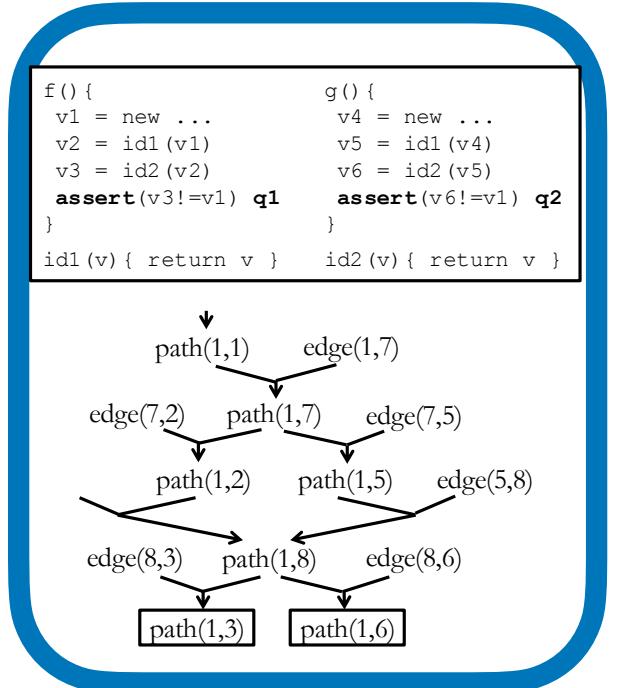
13–112KLOC C Programs (old and new)

3 bugs on average

Buffer overrun and Integer overflow analyses

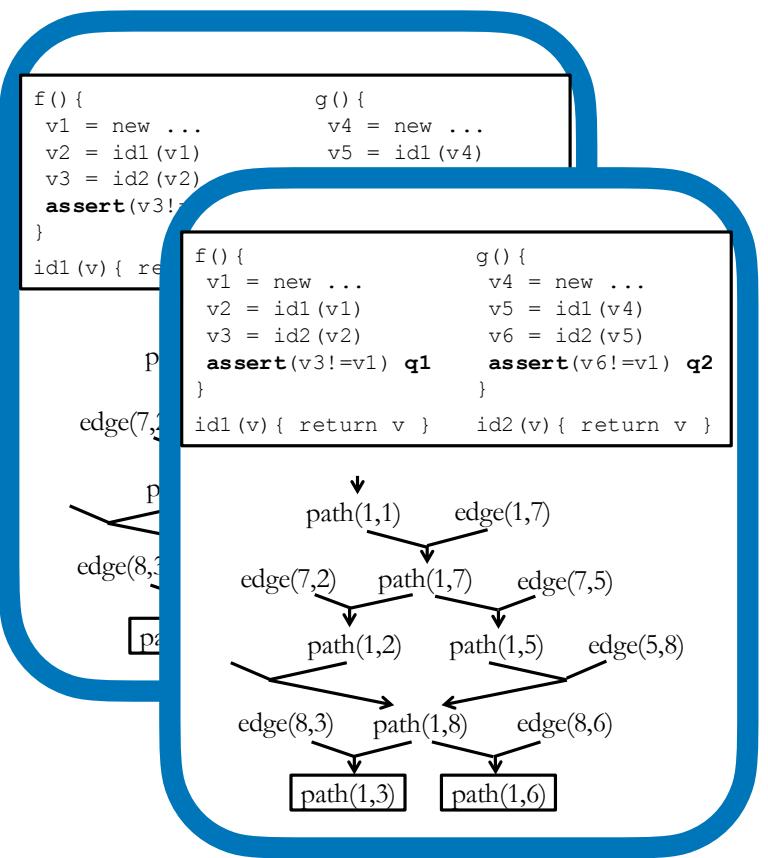
# Effectiveness

Avg. inspection burden  
to discover 3 bugs on average



Batch mode

All bugs are interspersed in  
**563** alarms



Ranking by relevance

All bugs located in  
up to rank **94**



+ Ranking by interaction

All bugs are founded  
up to **30** iterations



# Summary

- Towards next-generation program analysis systems with AI
  - **Adaptive:** automatically find a right abstraction
  - **Interactive:** incorporate user's feedback
  - **Continuous:** keep track of code changes
- Need a lot more research on making AI understand program code
  - E.g., feature engineerings, learning methods, etc