# Learning a Strategy for Adapting a Program Analysis via Bayesian Optimization


Hakjoo Oh

Korea University

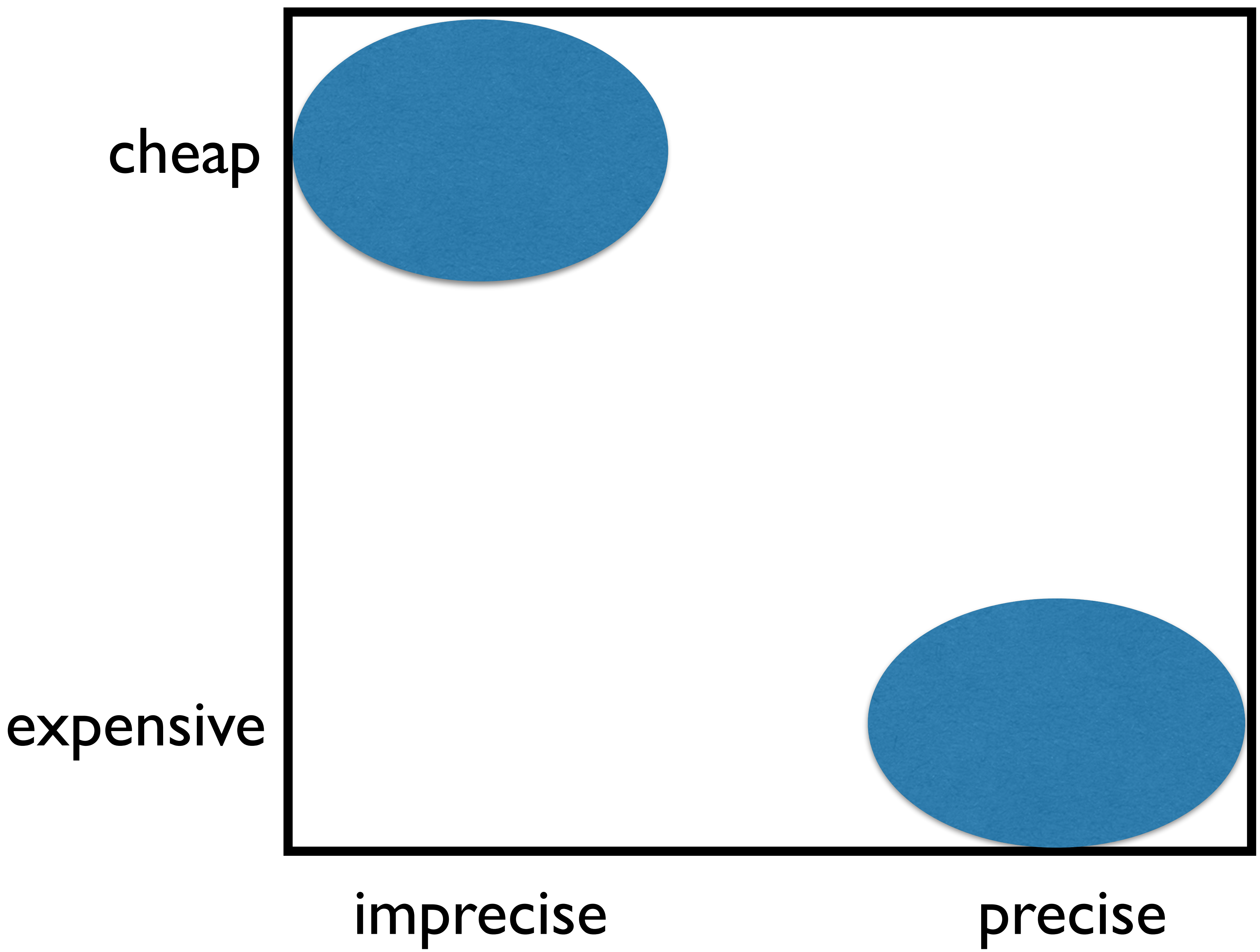
Hongseok Yang

Oxford University

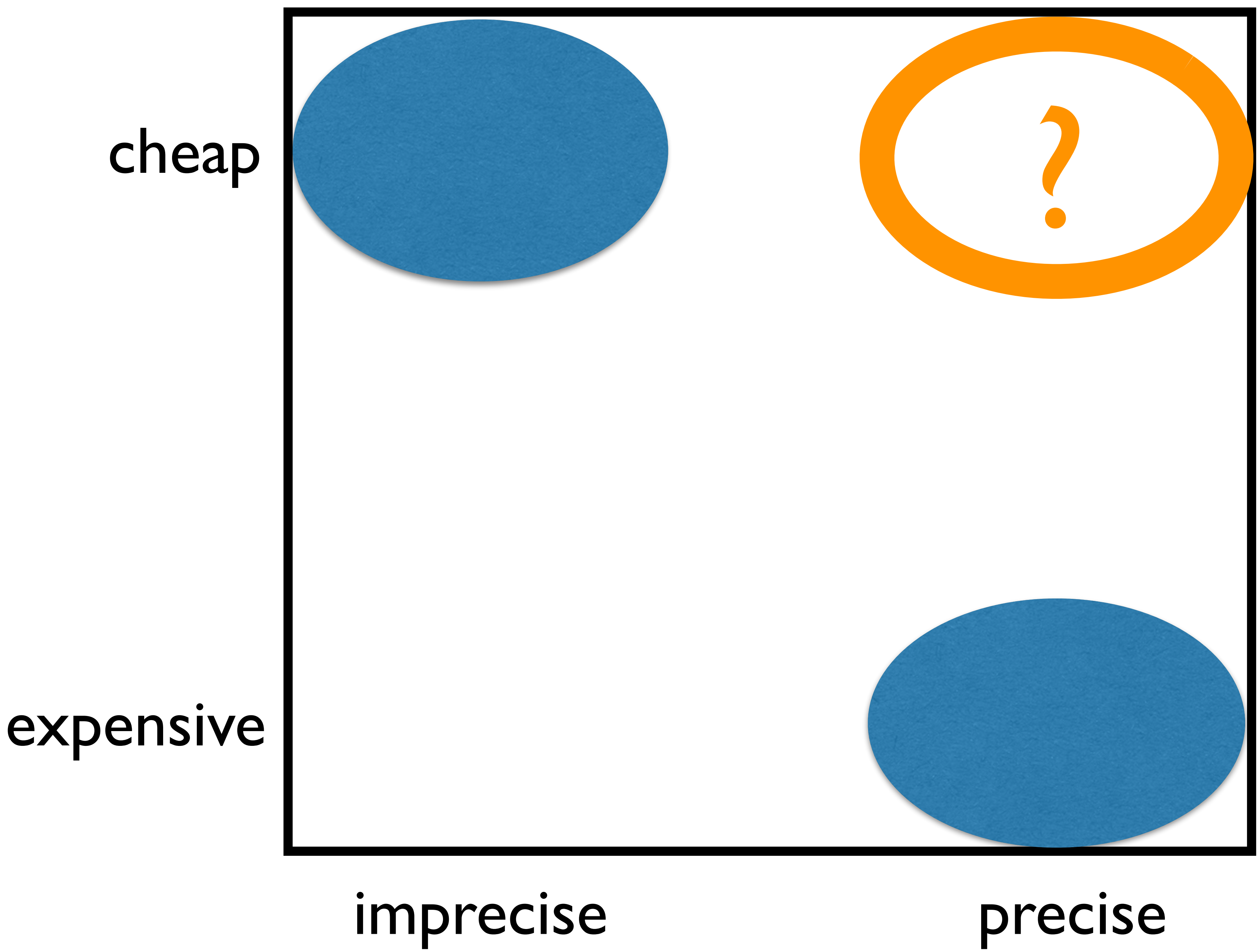
Kwangkeun Yi

Seoul National University

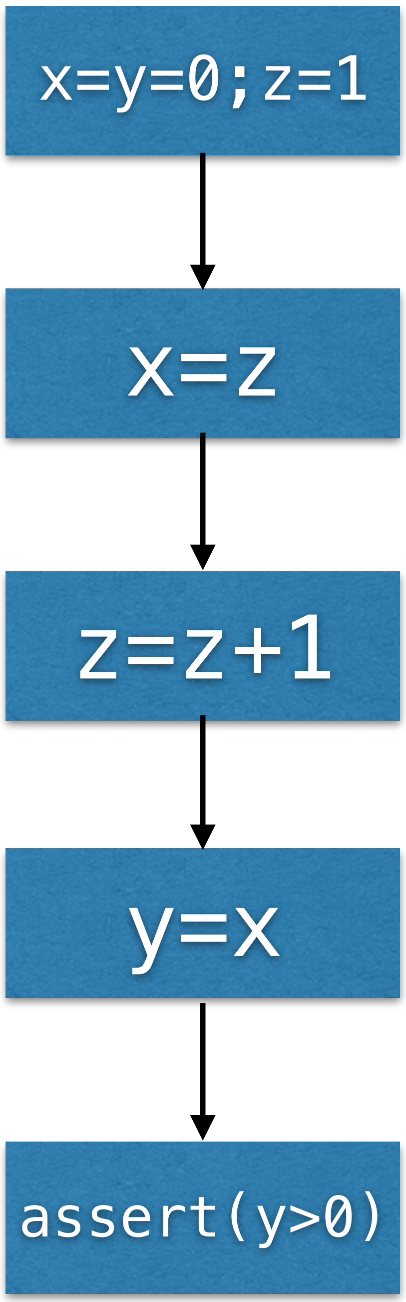
# Challenge in Static Analysis



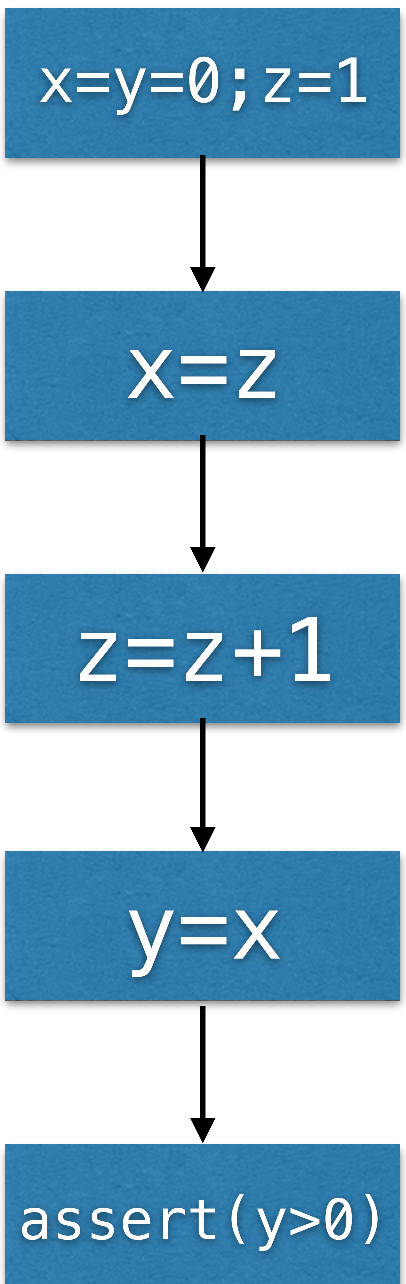
# Challenge in Static Analysis



# Flow-Sensitivity

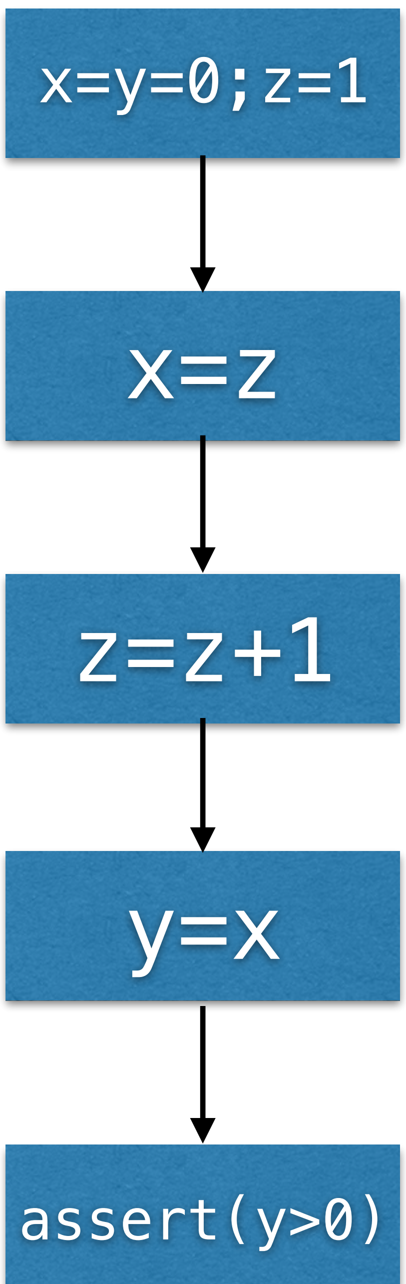


# Flow-Sensitivity




| | |
|---|---|
| x | [0,0] |
| y | [0,0] |
| z | [1,1] |

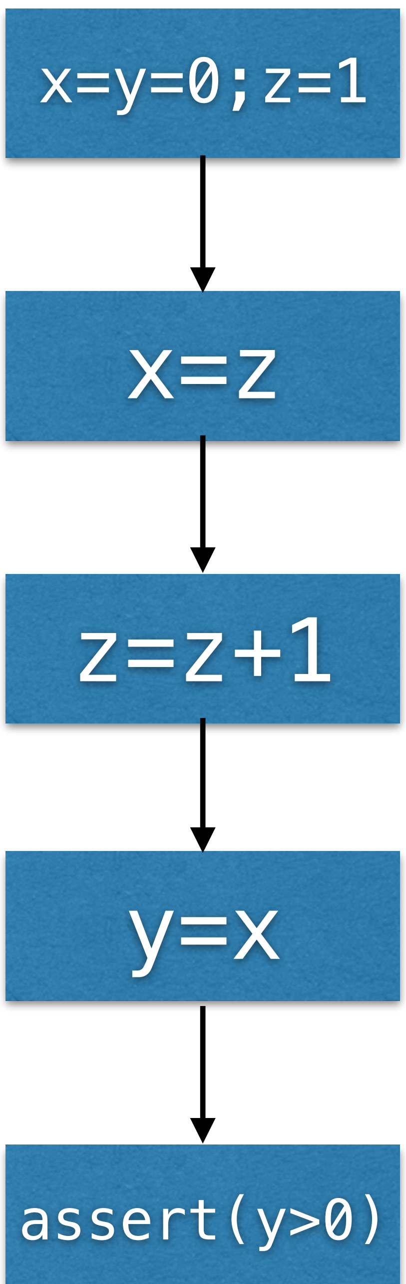
# Flow-Sensitivity




| | |
|---|---|
| x | [0,0] |
| y | [0,0] |
| z | [1,1] |

| | |
|---|---|
| x | [1,1] |
| y | [0,0] |
| z | [1,1] |

# Flow-Sensitivity

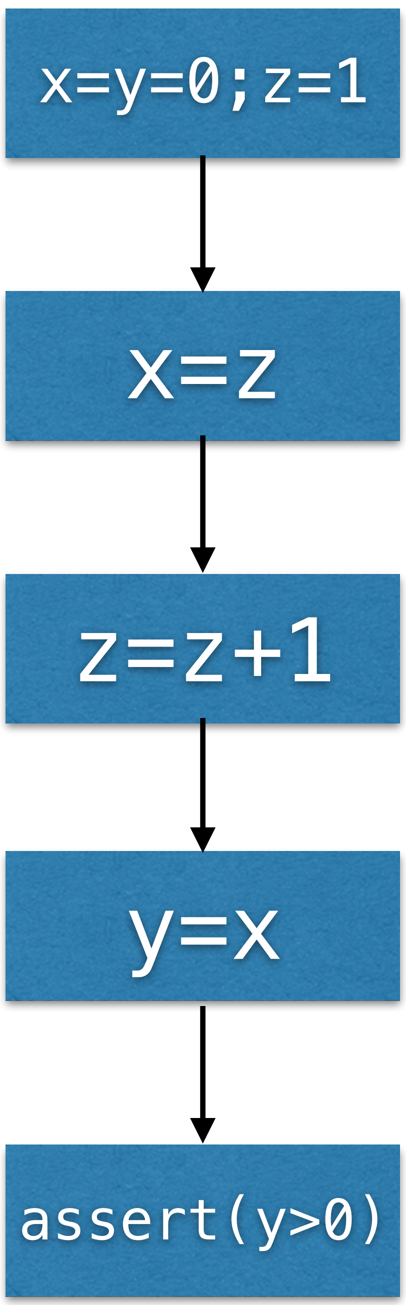



| | |
|---|---|
| x | [0,0] |
| y | [0,0] |
| z | [1,1] |

| | |
|---|---|
| x | [1,1] |
| y | [0,0] |
| z | [1,1] |

| | |
|---|---|
| x | [1,1] |
| y | [0,0] |
| z | [2,2] |

# Flow-Sensitivity



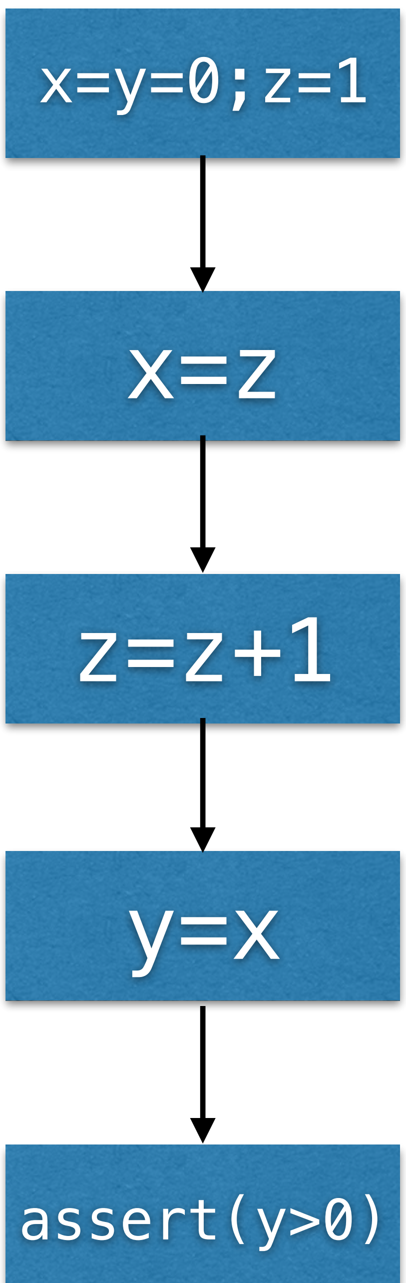

| | |
|---|---|
| x | [0,0] |
| y | [0,0] |
| z | [1,1] |

| | |
|---|---|
| x | [1,1] |
| y | [0,0] |
| z | [1,1] |

| | |
|---|---|
| x | [1,1] |
| y | [0,0] |
| z | [2,2] |

| | |
|---|---|
| x | [1,1] |
| y | [1,1] |
| z | [2,2] |

# Flow-Sensitivity




| | |
|---|---|
| x | [0,0] |
| y | [0,0] |
| z | [1,1] |

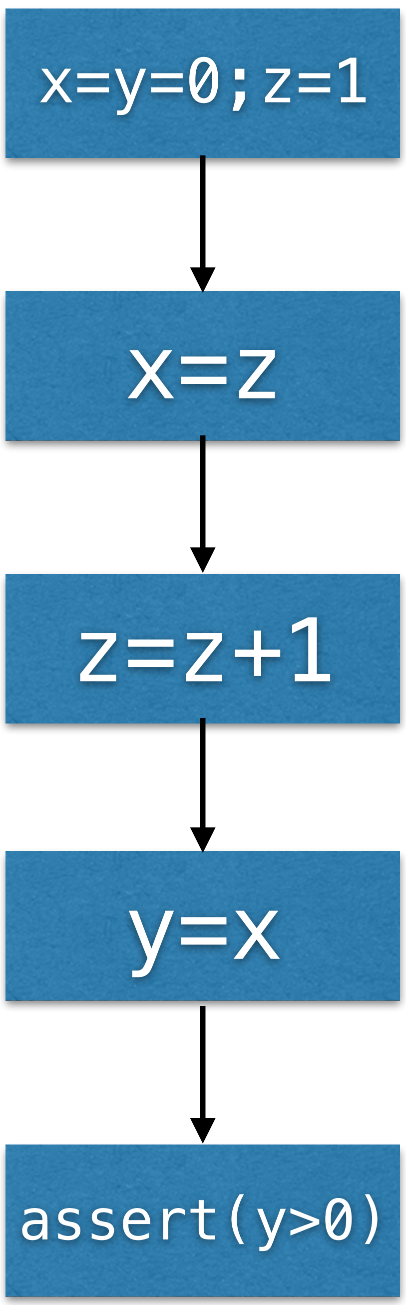
| | |
|---|---|
| x | [1,1] |
| y | [0,0] |
| z | [1,1] |

| | |
|---|---|
| x | [1,1] |
| y | [0,0] |
| z | [2,2] |

| | |
|---|---|
| x | [1,1] |
| y | [1,1] |
| z | [2,2] |

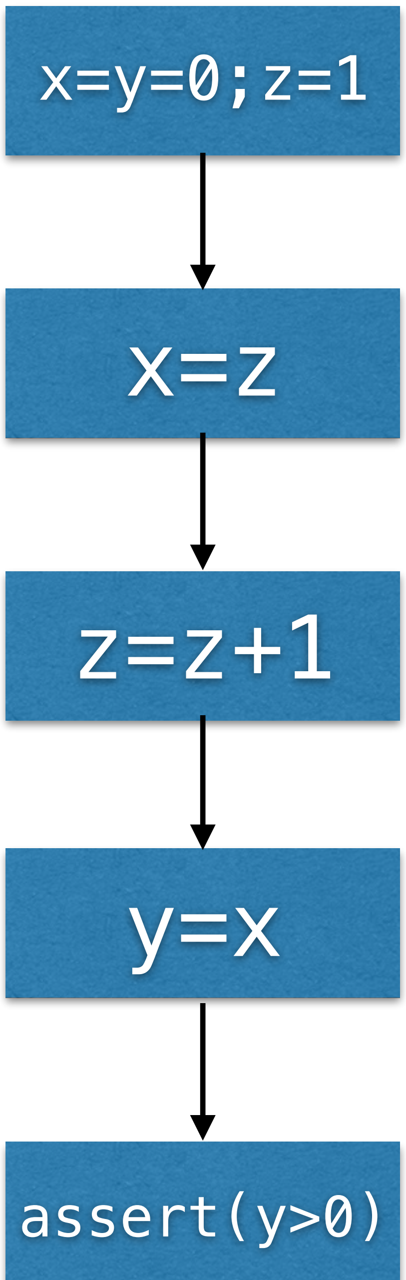
precise but costly

# Flow-Insensitivity




| | |
|---|---|
| x | [0,+∞] |
| y | [0,+∞] |
| z | [1,+∞] |

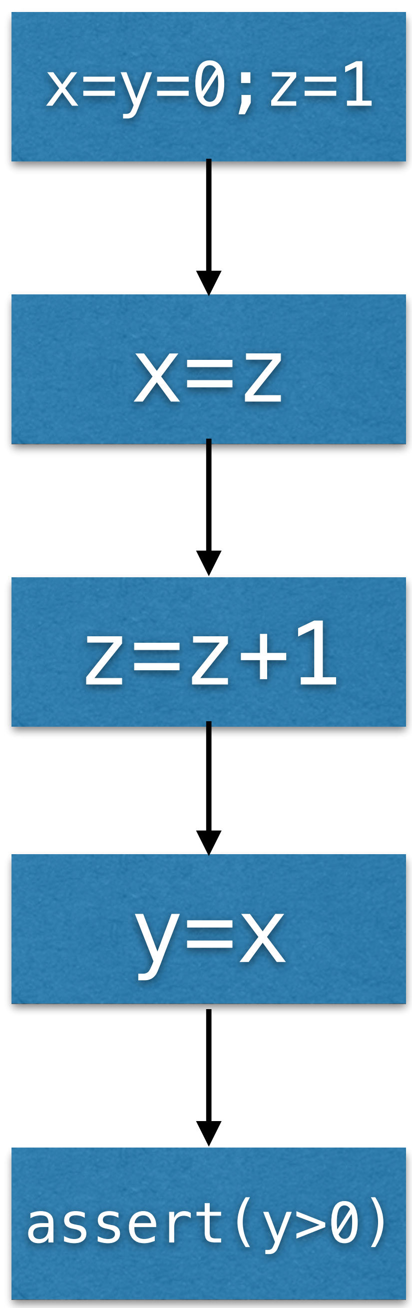
# Flow-Insensitivity




| | |
|---|---|
| x | [0,+∞] |
| y | [0,+∞] |
| z | [1,+∞] |

cheap but imprecise

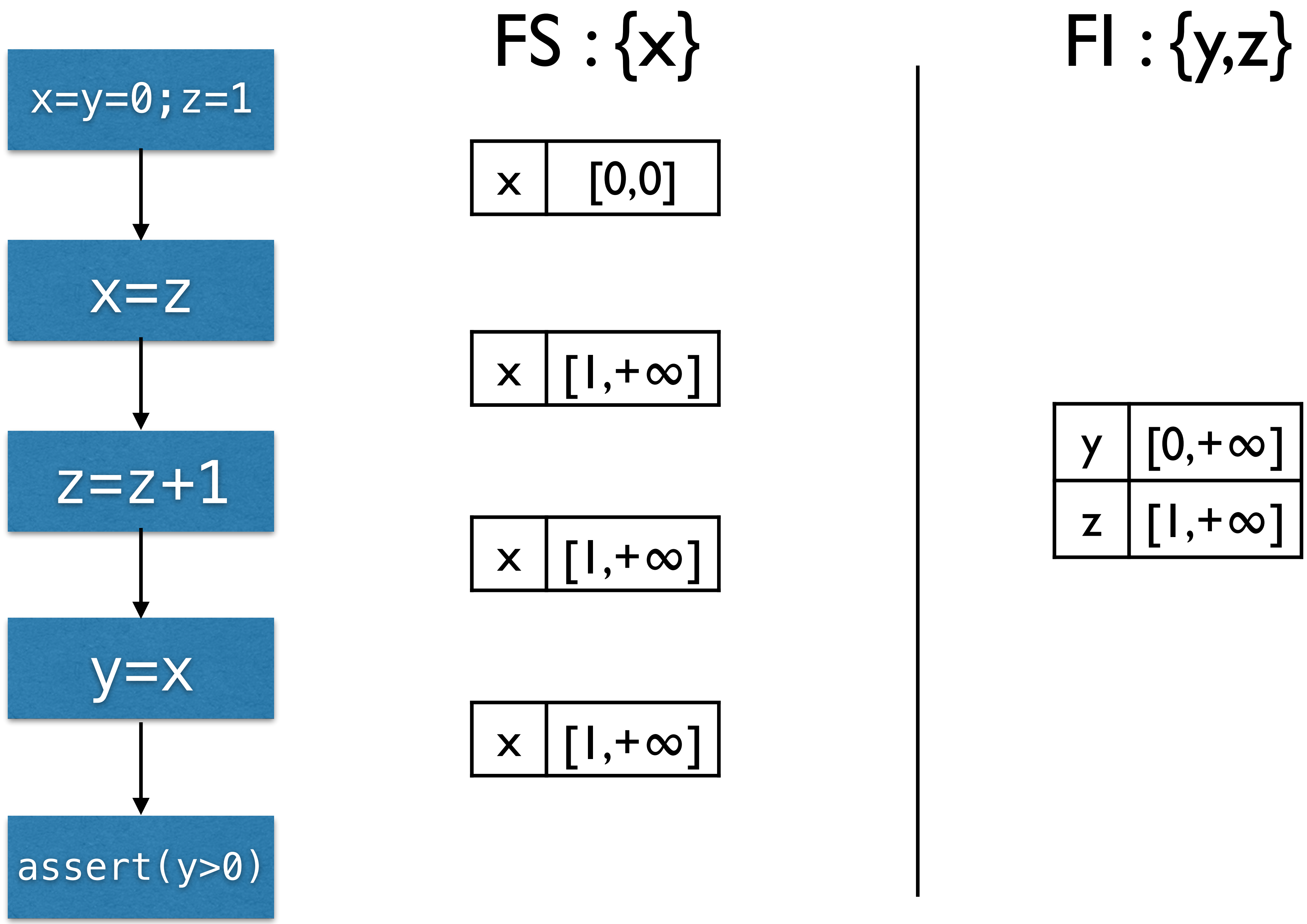
# Selective Flow-Sensitivity



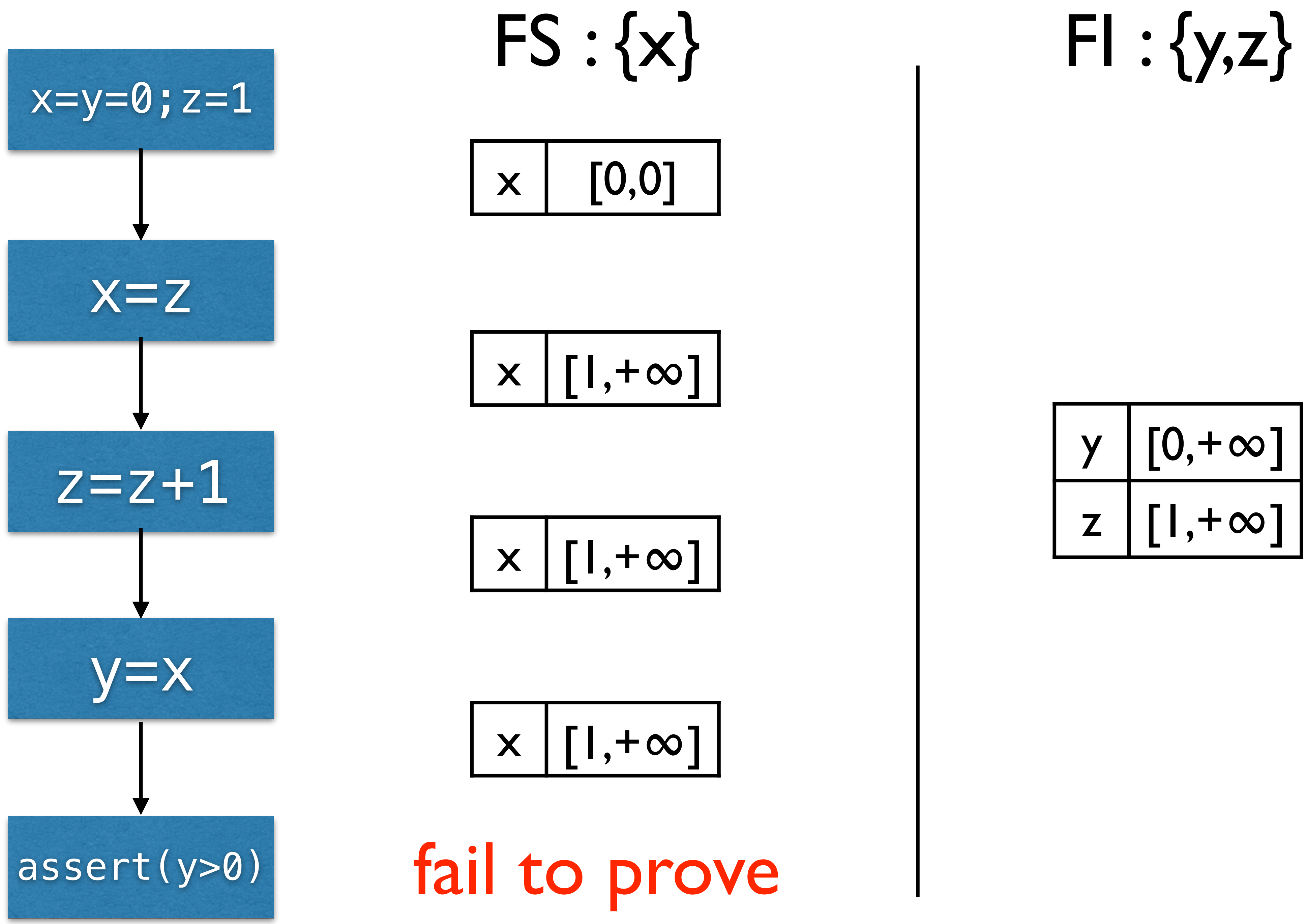

FS : {x}

FI : {y,z}

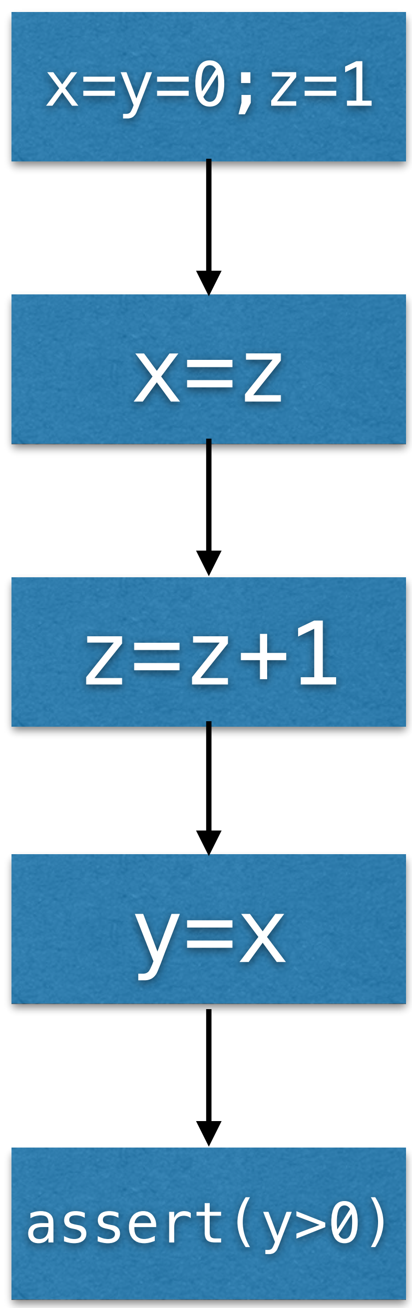
# Selective Flow-Sensitivity



# Selective Flow-Sensitivity



# Selective Flow-Sensitivity




FS : {y}

| | |
|---|---|
| y | [0,0] |

| | |
|---|---|
| y | [0,0] |

| | |
|---|---|
| y | [0,0] |

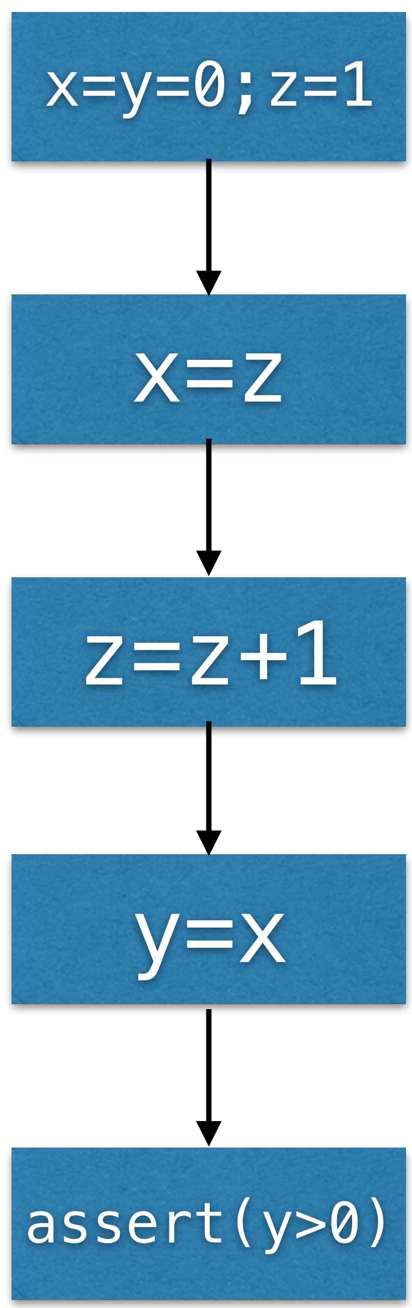
| | |
|---|---|
| y | [0,+∞] |

fail to prove

FI : {x,z}

| | |
|---|---|
| x | [0,+∞] |
| z | [1,+∞] |

# Selective Flow-Sensitivity




FS : {z}

| | |
|---|---|
| z | [1,1] |

| | |
|---|---|
| z | [1,1] |

| | |
|---|---|
| z | [2,2] |

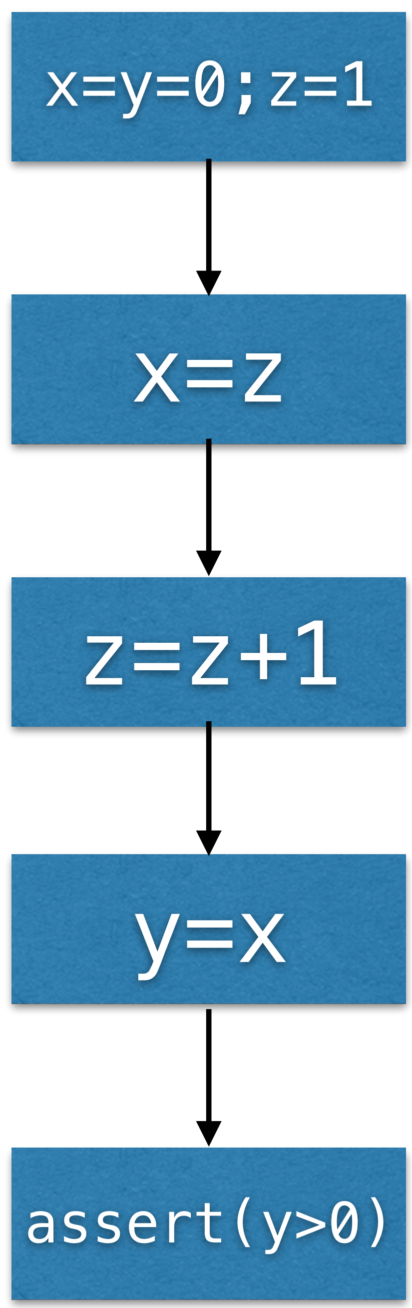
| | |
|---|---|
| z | [2,2] |

fail to prove

FI : {x,y}

| | |
|---|---|
| x | [0,+∞] |
| y | [0,+∞] |

# Selective Flow-Sensitivity




FS : {y,z}

| | |
|---|---|
| y | [0,0] |
| z | [1,1] |

| | |
|---|---|
| y | [0,0] |
| z | [1,1] |

| | |
|---|---|
| y | [0,0] |
| z | [2,2] |

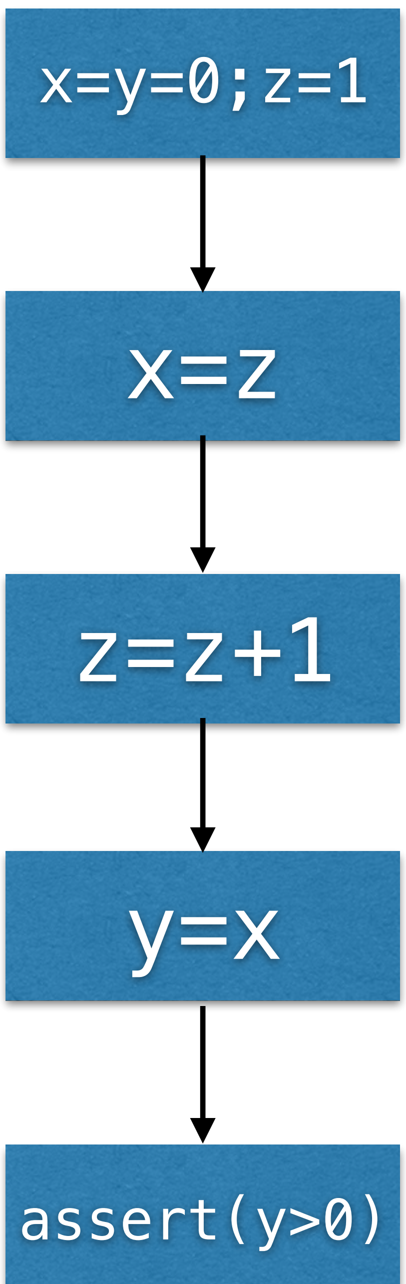
| | |
|---|---|
| y | [0,+∞] |
| z | [0,0] |

fail to prove

FI : {x}

| | |
|---|---|
| x | [0,+∞] |

# Selective Flow-Sensitivity




FS : {x,y}

| | |
|---|---|
| x | [0,0] |
| y | [0,0] |

| | |
|---|---|
| x | [1,+∞] |
| y | [0,0] |

| | |
|---|---|
| x | [1,+∞] |
| y | [0,0] |

| | |
|---|---|
| x | [1,+∞] |
| y | [1,+∞] |

Succeed

FI : {z}

| | |
|---|---|
| z | [1,+∞] |

# Hard Search Problem

- Intractably large space, if not infinite
  - $2^N$ different abstractions for FS
- Most of them are too imprecise or costly
  - $P(\{x,y,z\}) = \{\cancel{\emptyset}, \cancel{\{x\}}, \cancel{\{y\}}, \cancel{\{z\}}, \{x,y\}, \cancel{\{y,z\}}, \cancel{\{x,z\}}, \{x,y,z\}\}$

# Hard Search Problem

- Intractably large space, if not infinite
  - $2^N$ different abstractions for FS
- Most of them are too imprecise or costly
  - P({x,y,z}) = {~~∅~~,~~{x}~~,~~{y}~~,~~{z}~~,{x,y},~~{y,z}~~,~~{x,z}~~,{x,y,z}}

How to automatically find a cheap yet precise abstraction?

# Active Area in Static Analysis

- Recent work:
  - Impact pre-analysis [OhLeHeYaYi'14]
  - CEGAR with MAXSAT solving [ZhMaNaGrYa'14]
  - CEGAR with optimality guarantee [ZhNaYa'13]
  - Abstractions from tests [NaYaHa'12]
  - …

# Our Learning-based Approach

# Our Learning-based Approach

- Parameterized adaptation strategy

$$S_w : \mathrm{pgm} \rightarrow 2^{\mathrm{Var}}$$

# Our Learning-based Approach

- Parameterized adaptation strategy

$$S_W : \text{pgm} \rightarrow 2^{\text{Var}}$$

- Learn a good parameter $W$ from existing codebase

$$\boxed{P_1, P_2, \ldots, P_m} \Rightarrow W$$

Codebase

# Our Learning-based Approach

- Parameterized adaptation strategy

$$S_W : \text{pgm} \rightarrow 2^{\text{Var}}$$

- Learn a good parameter $W$ from existing codebase

$$\underbrace{\boxed{P_1, P_2, \ldots, P_m}}_{\text{Codebase}} \quad \Rightarrow \quad W$$

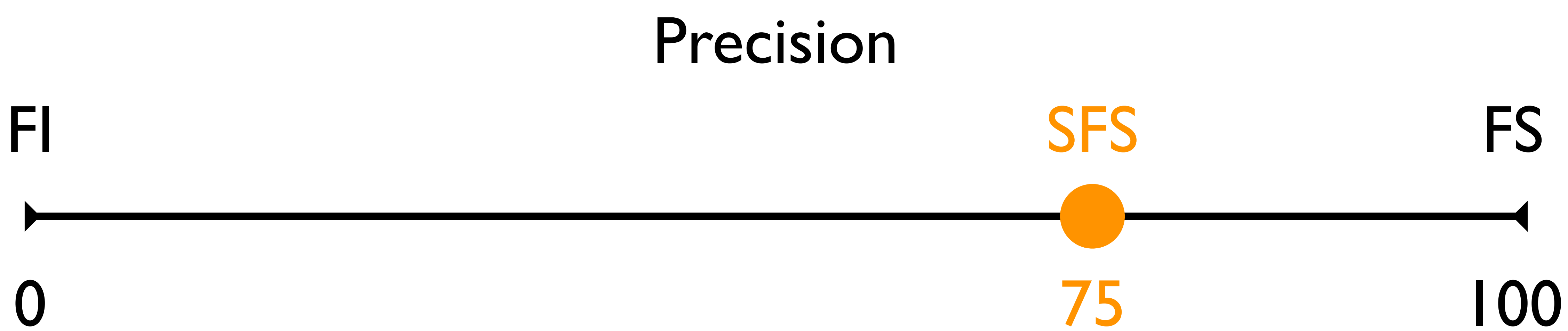
- For new program P, run static analysis with $S_W(P)$

# Effectiveness

- Implemented in Sparrow, a realistic C static analyzer
- Partially flow- and context-sensitive

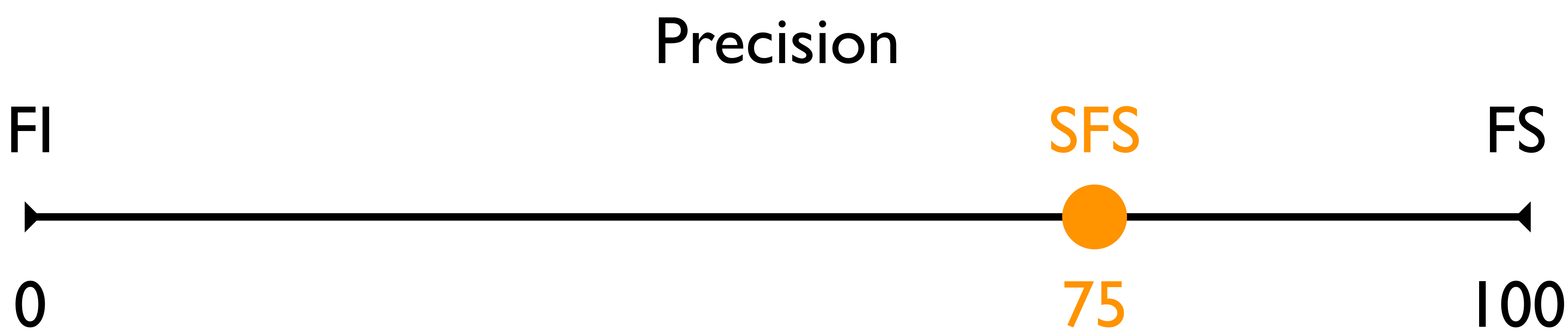
# Effectiveness

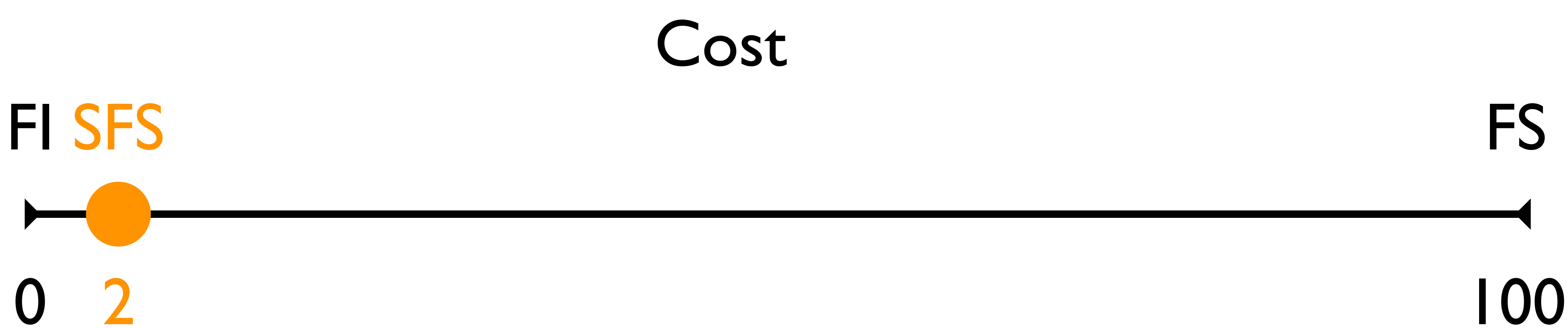
- Implemented in Sparrow, a realistic C static analyzer
- Partially flow- and context-sensitive



# Effectiveness

- Implemented in Sparrow, a realistic C static analyzer
- Partially flow- and context-sensitive



# Our Learning-based Approach

# Static Analyzer

number of
proved assertions

$$F(p, a) \Rightarrow n$$

abstraction
(e.g., a set of variables)

# Our Learning-based Approach

1. The abstraction is determined by a parameterized strategy:

$$S_w : \mathrm{pgm} \rightarrow 2^{\mathrm{Var}}$$

2. The parameter is learnt from an existing codebase:

$$\boxed{P_1, P_2, \ldots, P_m} \quad \Rightarrow \quad w$$

Codebase

# 1. Parameterized Strategy

$$S_W : \text{pgm} \rightarrow 2^{\text{Var}}$$

(1) Represent program variables as feature vectors.

(2) Compute the score of each variable.

(3) Choose the top-k variables based on the score.

# (1) Features

- Predicates over variables:

$$f = \{f_1, f_2, \ldots, f_5\} \qquad (f_i : \mathrm{Var} \rightarrow \{0,1\})$$

# (1) Features

- Predicates over variables:

$$f = \{f_1, f_2, \ldots, f_5\} \qquad (f_i : Var \rightarrow \{0,1\})$$

- Mostly syntactic features for variables: e.g,
  - local / global variable, passed to / returned from malloc, incremented by constants, etc
  - 45 features for flow-sensitivity

# (1) Features

- Represent each variable as a feature vector:

$$f(x) = \langle f_1(x), f_2(x), f_3(x), f_4(x), f_5(x) \rangle$$

$$f(x) = \langle 1,0,1,0,0 \rangle$$
$$f(y) = \langle 1,0,1,0,1 \rangle$$
$$f(z) = \langle 0,0,1,1,0 \rangle$$

# (2) Scoring

- The parameter **w** is a real-valued vector: e.g.,

$$\mathbf{w} = \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle$$

- Compute scores of variables:

$$\text{score}(x) = \langle 1,0,1,0,0 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.3$$
$$\text{score}(y) = \langle 1,0,1,0,1 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.6$$
$$\text{score}(z) = \langle 0,0,1,1,0 \rangle \cdot \langle 0.9, 0.5, -0.6, 0.7, 0.3 \rangle = 0.1$$

# (3) Choose Top-k Variables

- Choose the top-k variables based on their scores: e.g., when k=2,

$$\begin{array}{l} \text{score}(x) = 0.3 \\ \text{score}(y) = 0.6 \\ \text{score}(z) = 0.1 \end{array} \quad \Rightarrow \quad \{x,y\}$$

- In experiments, we chosen 10% of variables with highest scores.

# 2. Learn a Good Parameter

$$\boxed{P_1, P_2, \ldots, P_m} \Rightarrow W$$

Codebase

- Solve the optimization problem:

Find **w** that maximizes $\sum_{P_i} F(P_i, S_{\mathbf{w}}(P_i))$

# Learning via Random Sampling

repeat N times

pick $\mathbf{w} \in R^n$ randomly

evaluate $\sum_{P_i} F(P_i, S_{\mathbf{w}}(P_i))$

return best $\mathbf{w}$ found

1. $\langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle$

1. $\langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle$

$x : \langle 1,0,1,0,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.1$
$y : \langle 1,0,1,0,1 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = -0.9$
$z : \langle 0,0,1,1,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.8$

1. $\langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle$

$$x : \langle 1,0,1,0,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.1$$
$$y : \langle 1,0,1,0,1 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = -0.9$$
$$z : \langle 0,0,1,1,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.8$$

$$F(p, \{x,z\}) = 0$$

1. $\langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle$

$$x : \langle 1,0,1,0,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.1$$
$$y : \langle 1,0,1,0,1 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = -0.9$$
$$z : \langle 0,0,1,1,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.8$$

$$F(p, \{x,z\}) = 0$$

2. $\langle -0.8, 0.2, 0.5, -0.7, 0.7 \rangle$

$$F(p, \{y,z\}) = 0$$

1. $\langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle$

$x : \langle 1,0,1,0,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.1$
$y : \langle 1,0,1,0,1 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = -0.9$
$z : \langle 0,0,1,1,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.8$

$F(p, \{x,z\}) = 0$

2. $\langle -0.8, 0.2, 0.5, -0.7, 0.7 \rangle$

$F(p, \{y,z\}) = 0$

3. $\langle 0.7, 0.1, -0.6, -0.5, -0.1 \rangle$

$F(p, \{x,y\}) = 1$

1. $\langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle$

$x : \langle 1,0,1,0,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.1$
$y : \langle 1,0,1,0,1 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = -0.9$
$z : \langle 0,0,1,1,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.8$

$F(p, \{x,z\}) = 0$

2. $\langle -0.8, 0.2, 0.5, -0.7, 0.7 \rangle$

$F(p, \{y,z\}) = 0$

3. $\langle 0.7, 0.1, -0.6, -0.5, -0.1 \rangle$

$F(p, \{x,y\}) = 1$

4. $\langle -0.3, -0.4, 0.7, 0.8, -0.4 \rangle$

$F(p, \{x,z\}) = 0$

1. $\langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle$

$x : \langle 1,0,1,0,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.1$
$y : \langle 1,0,1,0,1 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = -0.9$
$z : \langle 0,0,1,1,0 \rangle \cdot \langle 0.2, 0.7, -0.1, 0.9, -1.0 \rangle = 0.8$

$F(p, \{x,z\}) = 0$

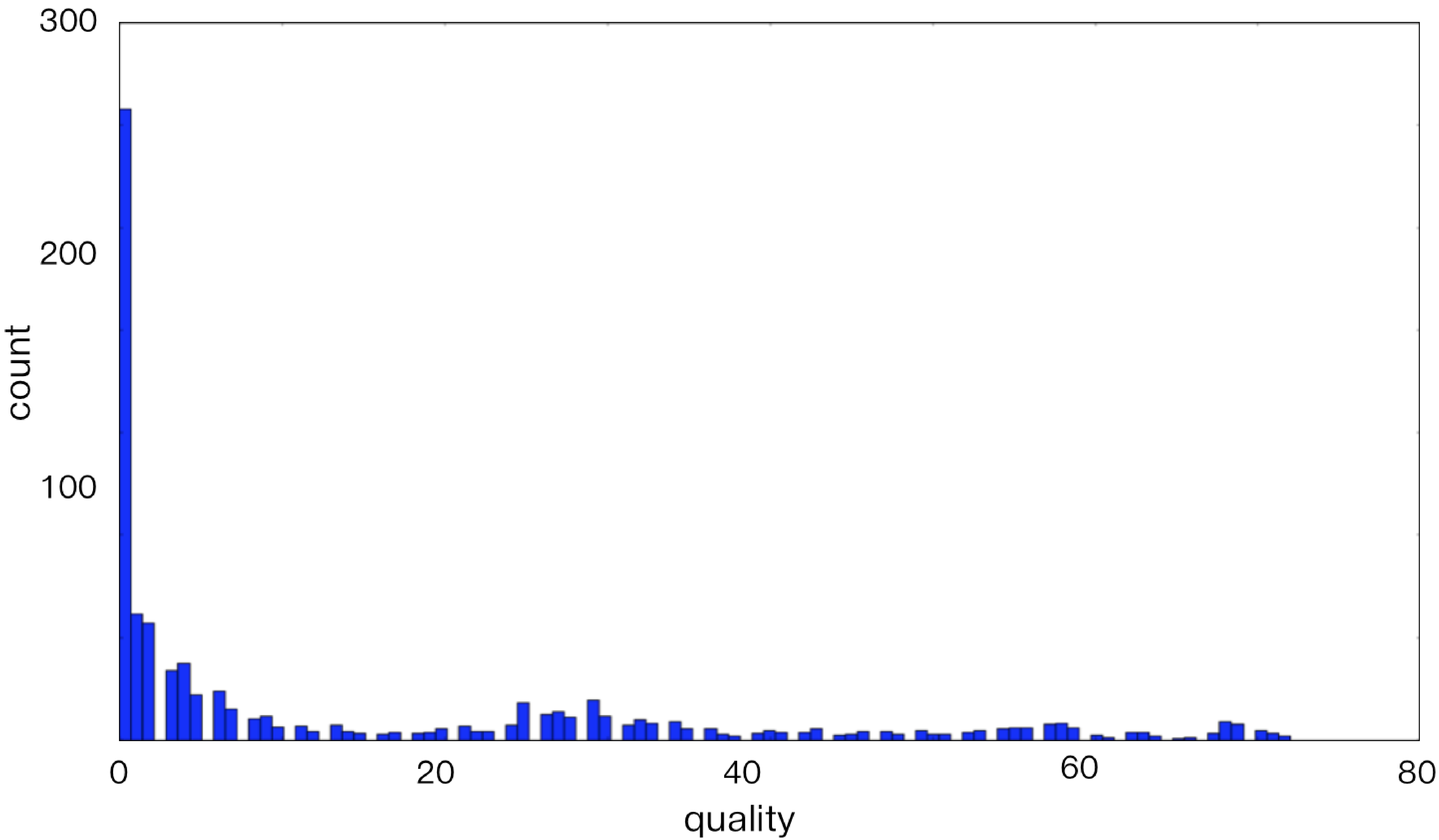
2. $\langle -0.8, 0.2, 0.5, -0.7, 0.7 \rangle$ $F(p, \{y,z\}) = 0$

3. $\langle 0.7, 0.1, -0.6, -0.5, -0.1 \rangle$ $F(p, \{x,y\}) = 1$

4. $\langle -0.3, -0.4, 0.7, 0.8, -0.4 \rangle$ $F(p, \{x,z\}) = 0$

5. $\langle 0.6, 0.8, -0.5, 0.9, -0.3 \rangle$ $F(p, \{x,z\}) = 0$

# Learning via Random Sampling



# Our Approach: Learning via Bayesian Optimization

- A powerful method for solving difficult optimization problems.

- Especially when, the objective function is expensive to evaluate and lack a good structure.

- Key idea: use a probabilistic model to reduce the number of objective function evaluations.

# Learning via Bayesian Optimization

repeat N times

update the probabilistic model

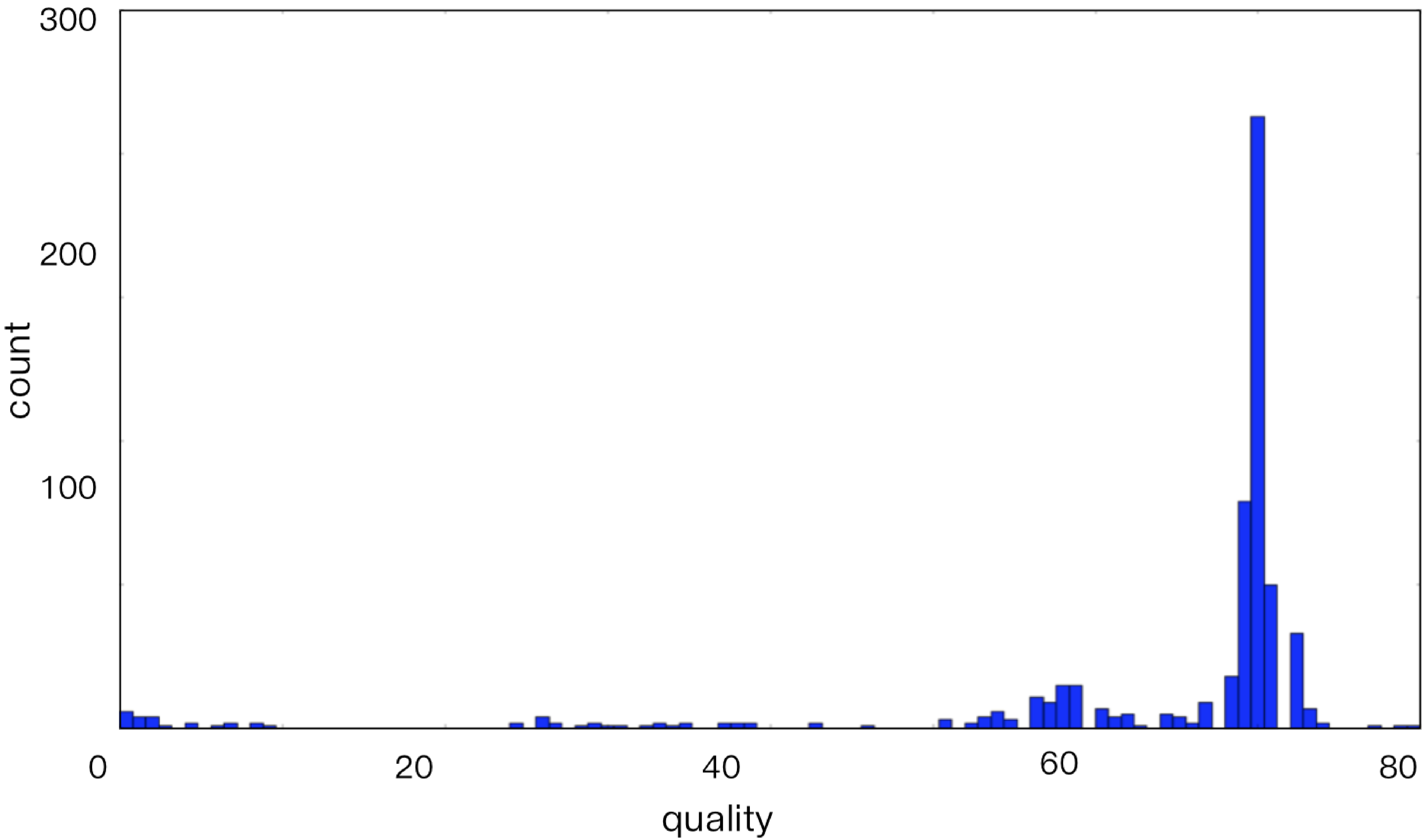
select a promising w

evaluate $\sum_{P_i} F(P_i, S_{\mathbf{w}}(P_i))$

return best **w** found

- Probabilistic model: Gaussian processes
- Selection strategy: Expected improvement

# Learning via Bayesian Optimization



# Experiments

- Sparrow: a C static analyzer for buffer-overrun checking
- Tune partial flow- and context-sensitivity of Sparrow
  - 10% of program variables for flow-sensitivity
  - 10% of procedures for context-sensitivity
- 30 C programs (2K ~ 90KLoC)

# Research Questions

1. Does our learning algorithm produce a good strategy?
   - Cross validation with 20 training and 10 testing programs, repeated for five times
2. How much does the Bayesian optimization benefit?
   - Comparison with random sampling
3. Does the learnt strategy provide new insights?

# 1. Performance

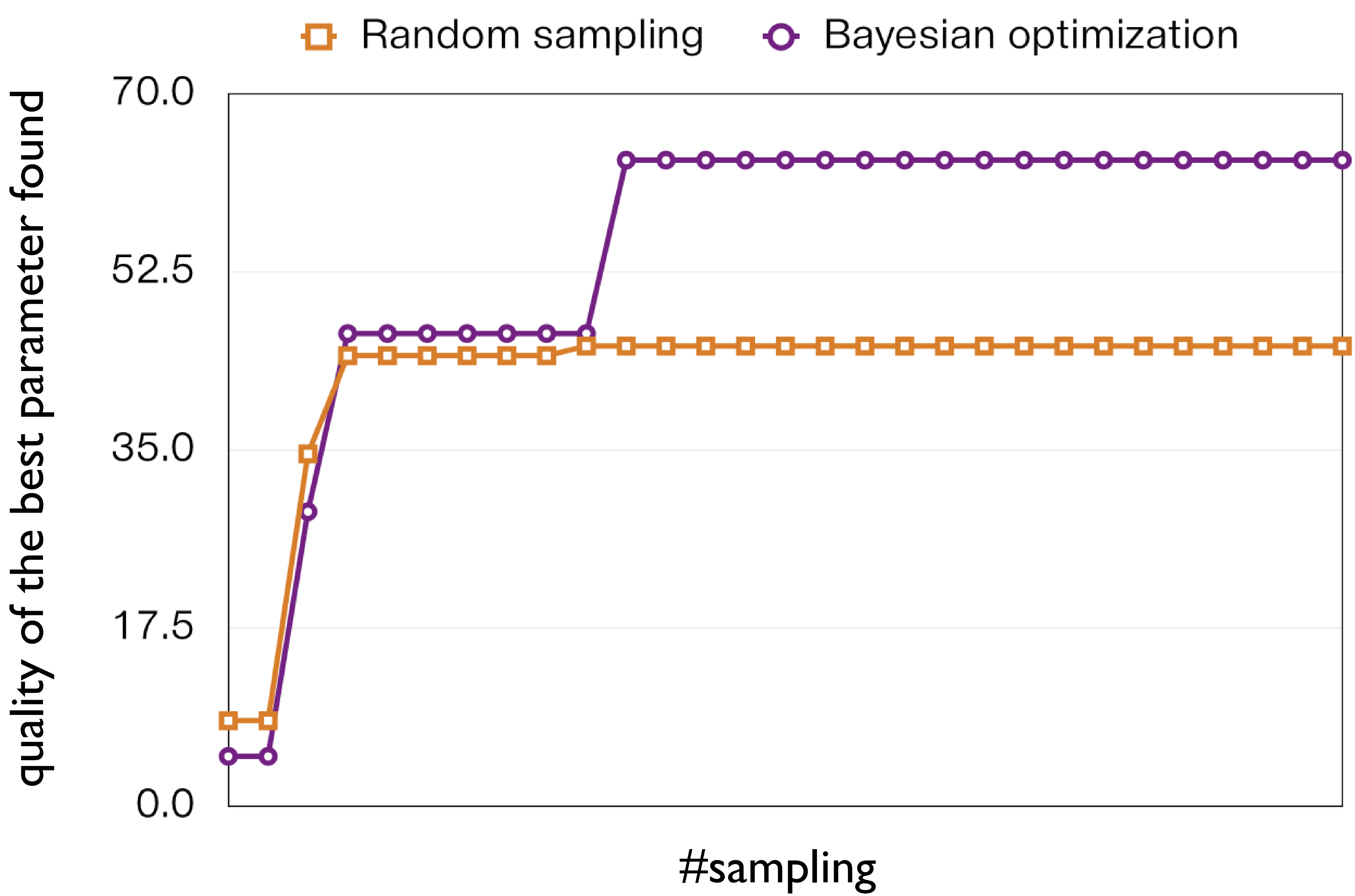
## Flow-Sensitivity

| | Training | | | | Testing | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | FI | FS | partial FS | | FI | | FS | | | partial FS | | | |
| Trial | prove | prove | prove | quality | prove | sec | prove | sec | cost | prove | sec | quality | cost |
| 1 | 6,383 | 7,316 | 7,089 | 75.7 % | 2,788 | 48 | 4,009 | 627 | 13.2 x | 3,692 | 78 | 74.0 % | 1.6 x |
| 2 | 5,788 | 7,422 | 7,219 | 87.6 % | 3,383 | 55 | 3,903 | 531 | 9.6 x | 3,721 | 93 | 65.0 % | 1.7 x |
| 3 | 6,148 | 7,842 | 7,595 | 85.4 % | 3,023 | 49 | 3,483 | 1,898 | 38.6 x | 3,303 | 99 | 60.9 % | 2.0 x |
| 4 | 6,138 | 7,895 | 7,599 | 83.2 % | 3,033 | 38 | 3,430 | 237 | 6.2 x | 3,286 | 51 | 63.7 % | 1.3 x |
| 5 | 7,343 | 9,150 | 8,868 | 84.4 % | 1,828 | 28 | 2,175 | 577 | 20.5 x | 2,103 | 54 | 79.3 % | 1.9 x |
| TOTAL | 31,800 | 39,625 | 38,370 | **84.0 %** | 14,055 | 218 | 17,000 | 3,868 | **17.8 x** | 16,105 | 374 | **69.6 %** | **1.7 x** |

## Flow-Sensitivity + Context-Sensitivity

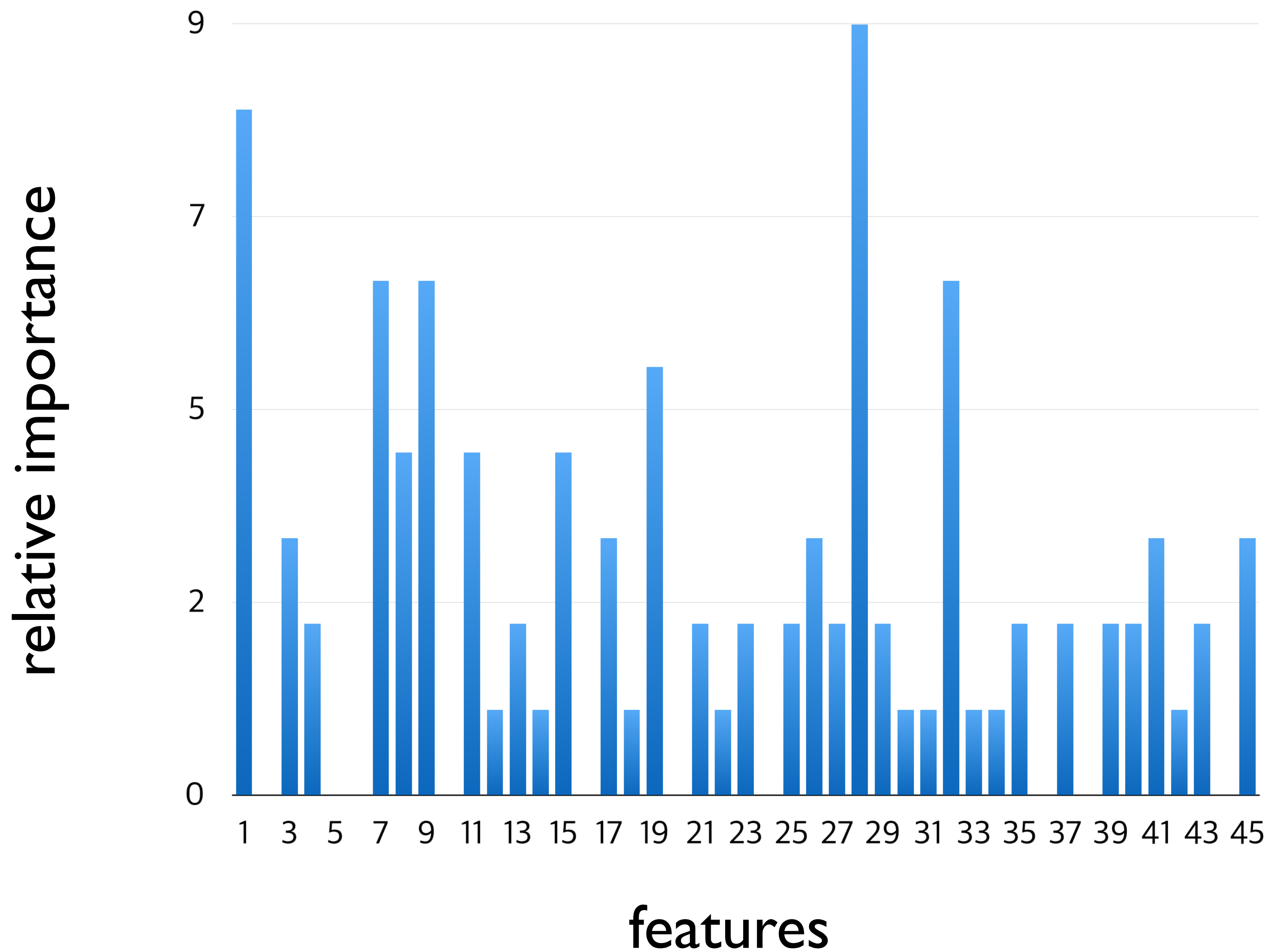
| | Training | | | | Testing | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | FICI | FSCS | partial FSCS | | FICI | | FSCS | | | partial FSCS | | | |
| Trial | prove | prove | prove | quality | prove | sec | prove | sec | cost | prove | sec | quality | cost |
| 1 | 6,383 | 9,237 | 8,674 | 80.3 % | 2,788 | 46 | 4,275 | 5,425 | 118.2 x | 3,907 | 187 | 75.3 % | 4.1 x |
| 2 | 5,788 | 8,287 | 7,598 | 72.4 % | 3,383 | 57 | 5,225 | 4,495 | 79.4 x | 4,597 | 194 | 65.9 % | 3.4 x |
| 3 | 6,148 | 8,737 | 8,123 | 76.3 % | 3,023 | 48 | 4,775 | 5,235 | 108.8 x | 4,419 | 150 | 79.7 % | 3.1 x |
| 4 | 6,138 | 9,883 | 8,899 | 73.7 % | 3,033 | 38 | 3,629 | 1,609 | 42.0 x | 3,482 | 82 | 75.3 % | 2.1 x |
| 5 | 7,343 | 10,082 | 10,040 | 98.5 % | 1,828 | 30 | 2,670 | 7,801 | 258.3 x | 2,513 | 104 | 81.4 % | 3.4 x |
| TOTAL | 31,800 | 46,226 | 43,334 | **80.0 %** | 14,055 | 219 | 20,574 | 24,565 | **112.1 x** | 18,918 | 717 | **74.6 %** | **3.3 x** |

# 2. Comparison



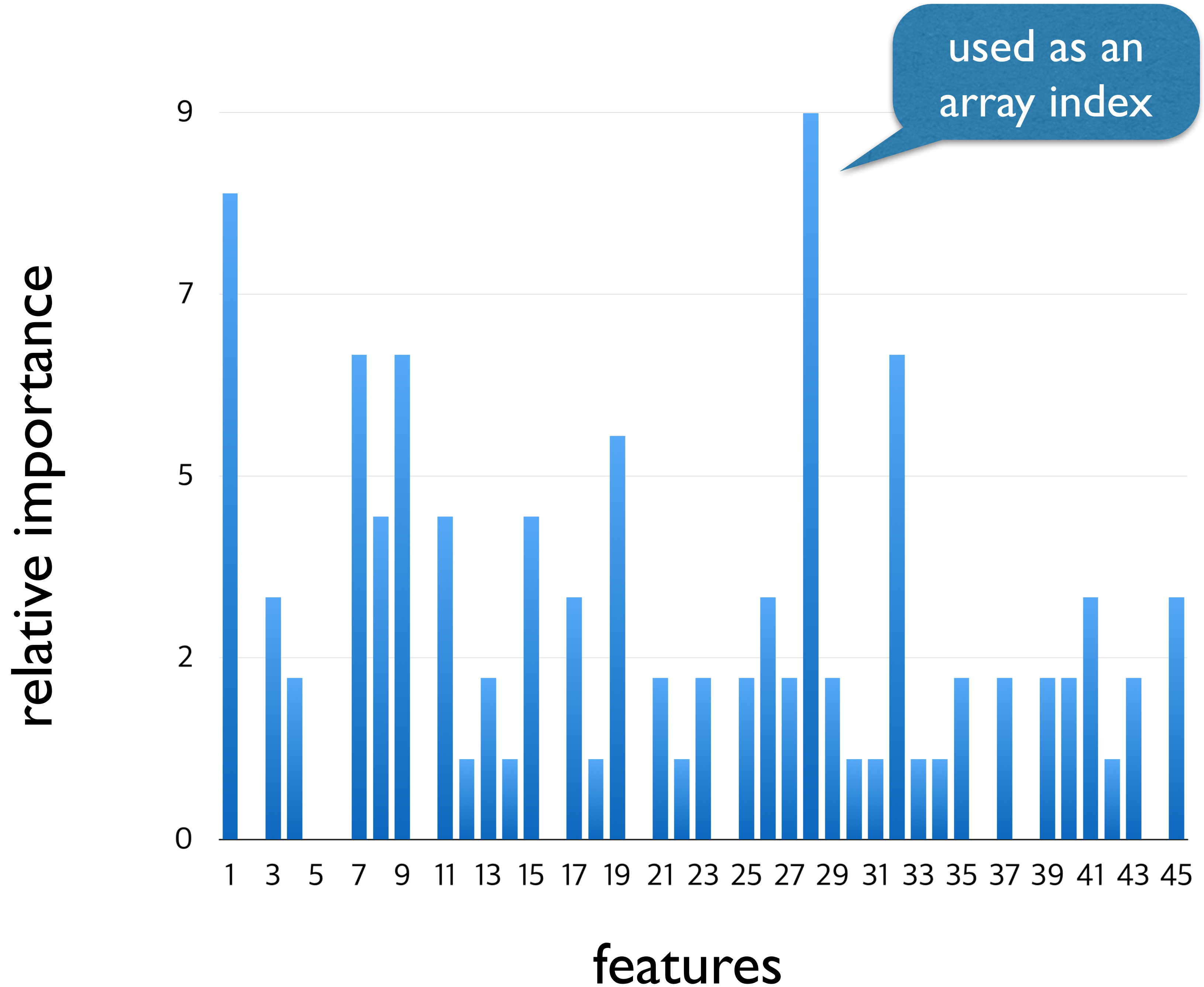
# 3. New Insights

- Relative importance between program features:



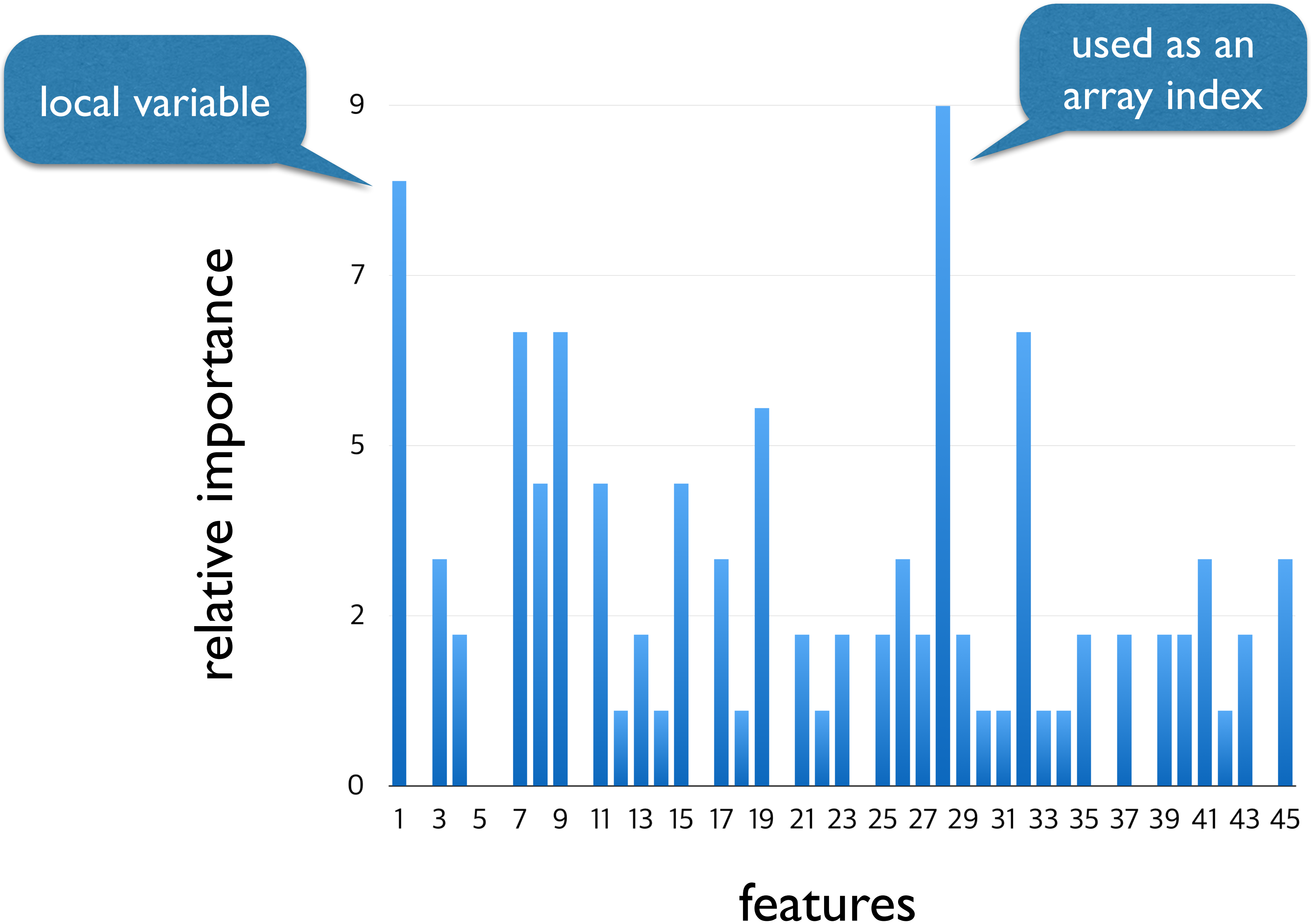
# 3. New Insights

- Relative importance between program features:



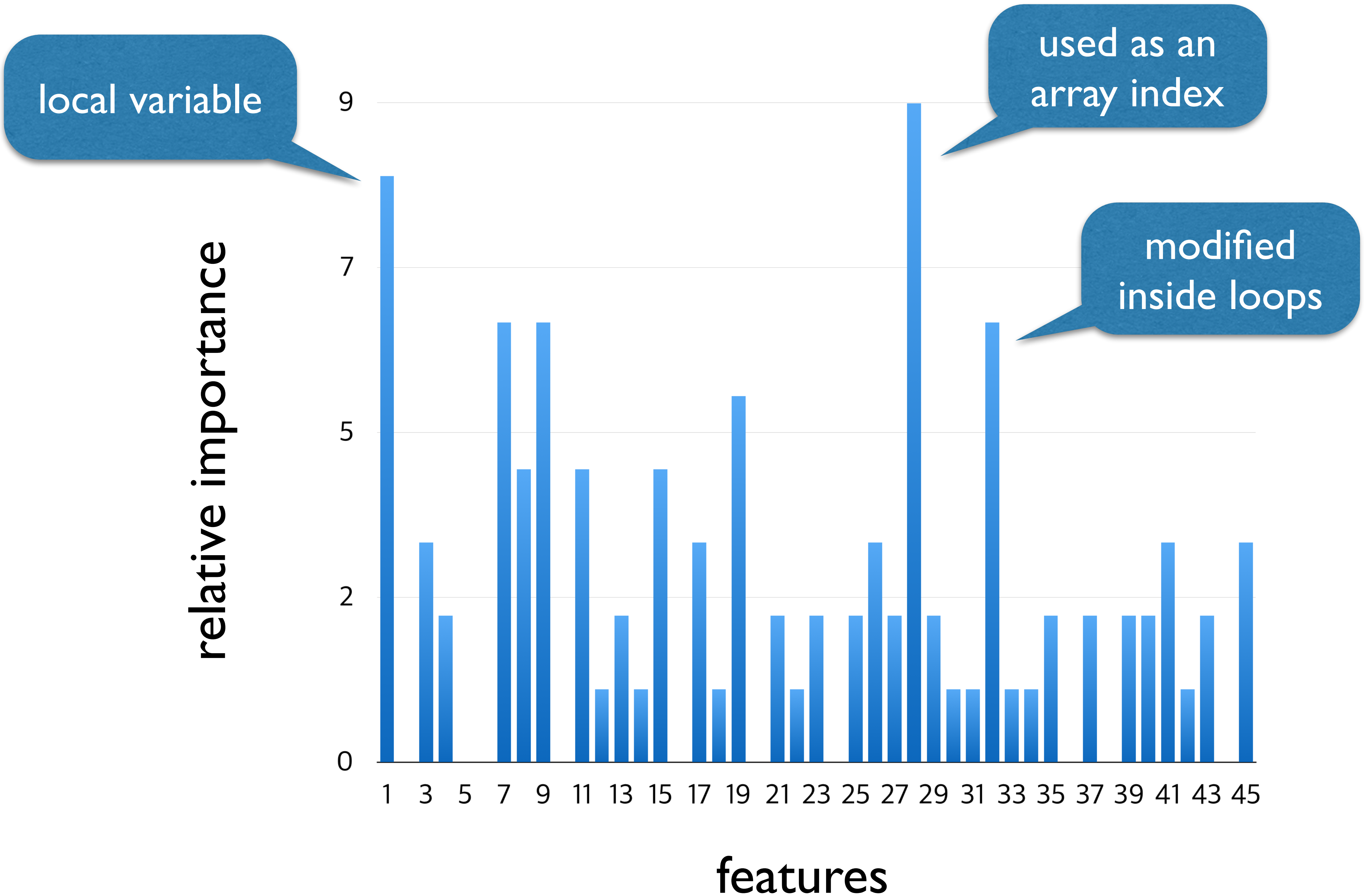
# 3. New Insights

- Relative importance between program features:



# 3. New Insights

- Relative importance between program features:



# 3. New Insights

- Typical scenario where flow-sensitivity helps:

```
int mirror[7];
int i = unknown;
for (i=1;i<7;i++)
  if (mirror[i-1] == ’1’)  ...
```

# 3. New Insights

- Typical scenario where flow-sensitivity helps:

local variable

```
int mirror[7];
int i = unknown;
for (i=1;i<7;i++)
  if (mirror[i-1] == '1')  ...
```

# 3. New Insights

- Typical scenario where flow-sensitivity helps:

local variable

```
int mirror[7];
int i = unknown;
for (i=1;i<7;i++)
  if (mirror[i-1] == '1')   ...
```

modified
inside loops

# 3. New Insights

- Typical scenario where flow-sensitivity helps:

local variable

```
int mirror[7];
int i = unknown;
for (i=1;i<7;i++)
  if (mirror[i-1] == '1')   ...
```

modified inside loops

used as an array index

# 3. New Insights

- Also provide unexpected domain knowledge.

```
int pos = unknown;
if (!pos)
  path[pos] = 0;
```

# 3. New Insights

- Also provide unexpected domain knowledge.

negated in conditional expression

```
int pos = unknown;
if (!pos)
  path[pos] = 0;
```

# 3. New Insights

- Also provide unexpected domain knowledge.

negated in conditional expression

```
int pos = unknown;
if (!pos)
  path[pos] = 0;
```

- Over the entire codebase, the feature is a strong indicator for flow-“insensitivity”

# Summary

## Our Learning-based Approach

- First machine learning-based approach
  - formulated as an optimization problem
  - solved by Bayesian optimization
- Effective: 75% precision with 2% cost
- Generally applicable to any static analysis and automatic

# Summary

## Our Learning-based Approach

- First machine learning-based approach
  - formulated as an optimization problem
  - solved by Bayesian optimization
- Effective: 75% precision with 2% cost
- Generally applicable to any static analysis and automatic

Thank you