Learning a Strategy for Adapting a Program Analysis via Bayesian Optimization

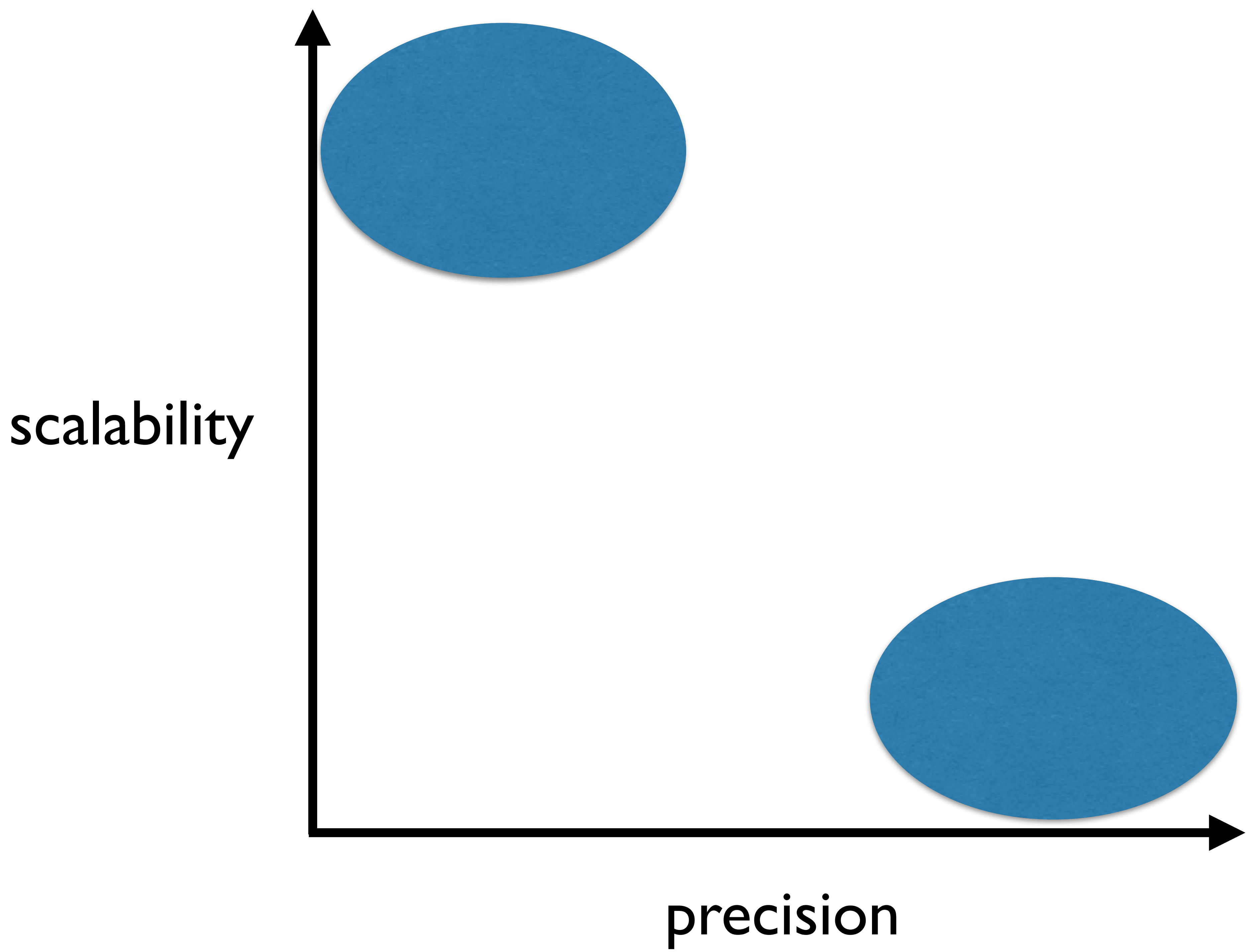
Hakjoo Oh — Korea University

Hongseok Yang — Oxford University

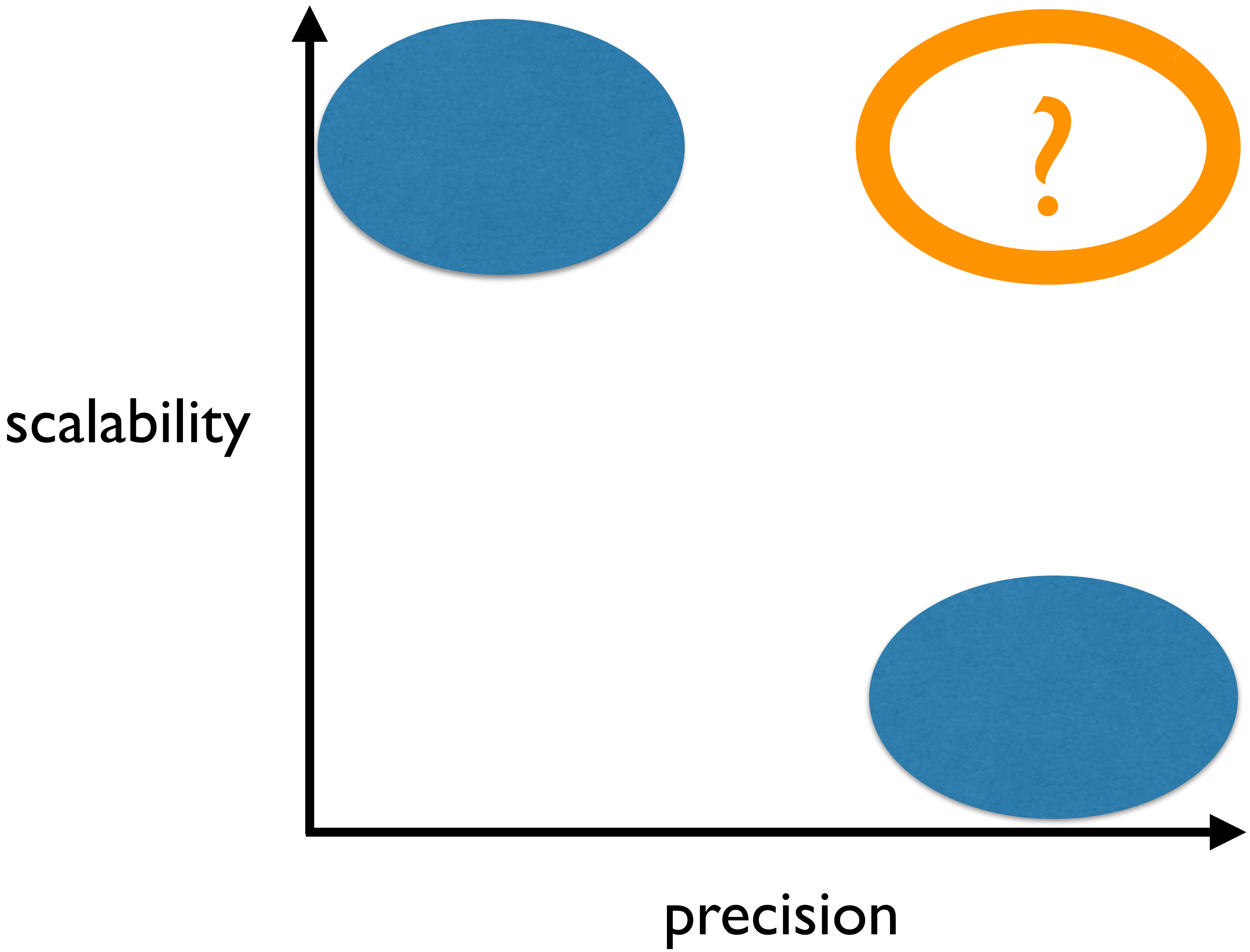
Kwangkeun Yi — Seoul National University

OOPSLA 2015 @ Pittsburgh, US

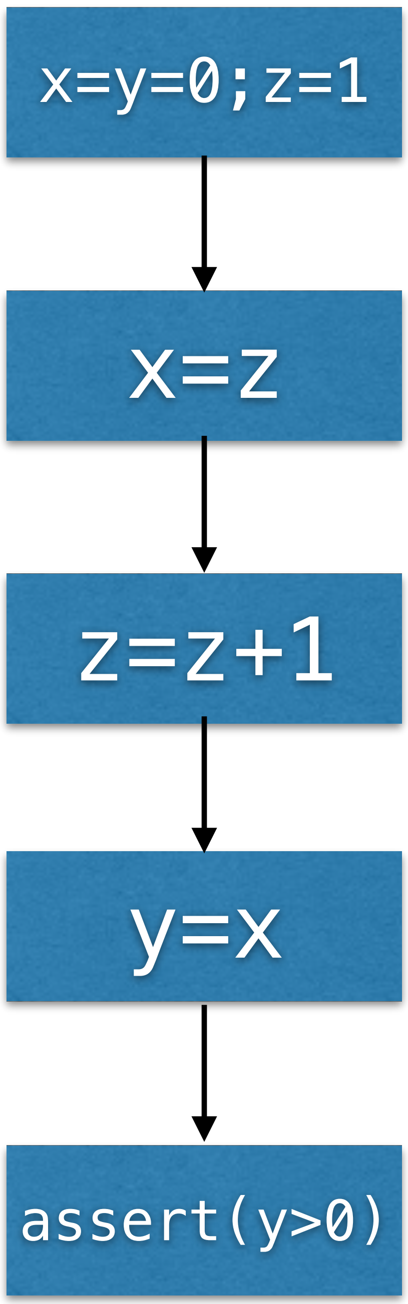
Challenge in Static Analysis



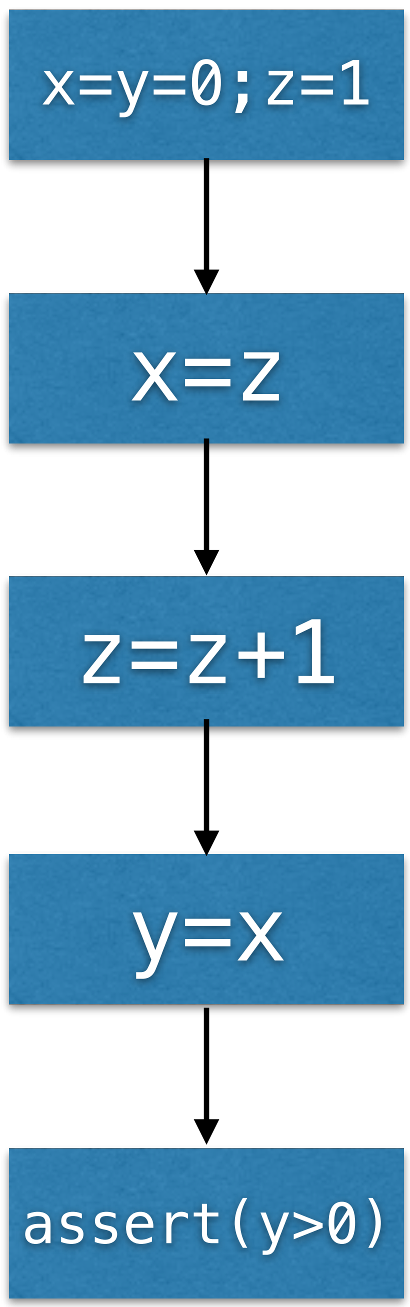
Challenge in Static Analysis



Flow-Sensitivity

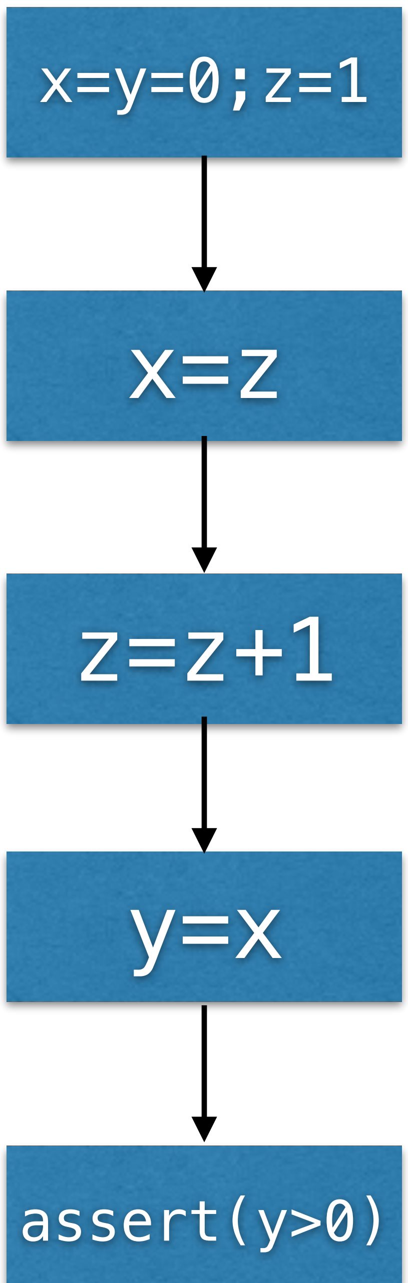


Flow-Sensitivity



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

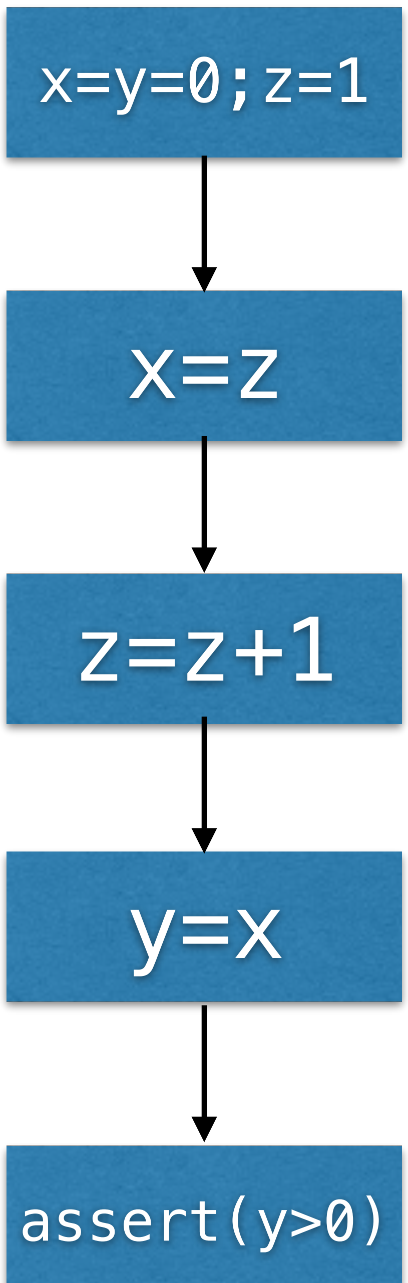
Flow-Sensitivity



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

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

Flow-Sensitivity

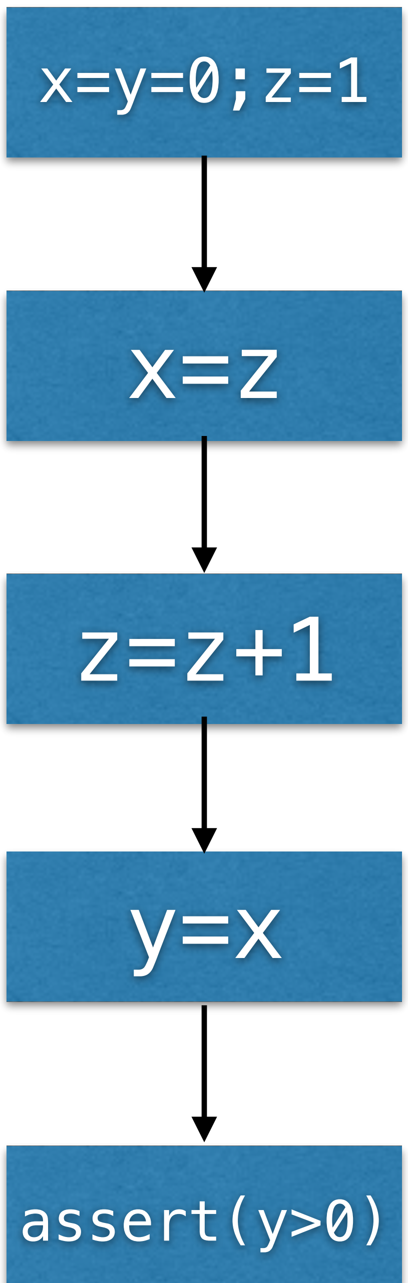


x	[0,0]
y	[0,0]
z	[1,1]

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

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

Flow-Sensitivity



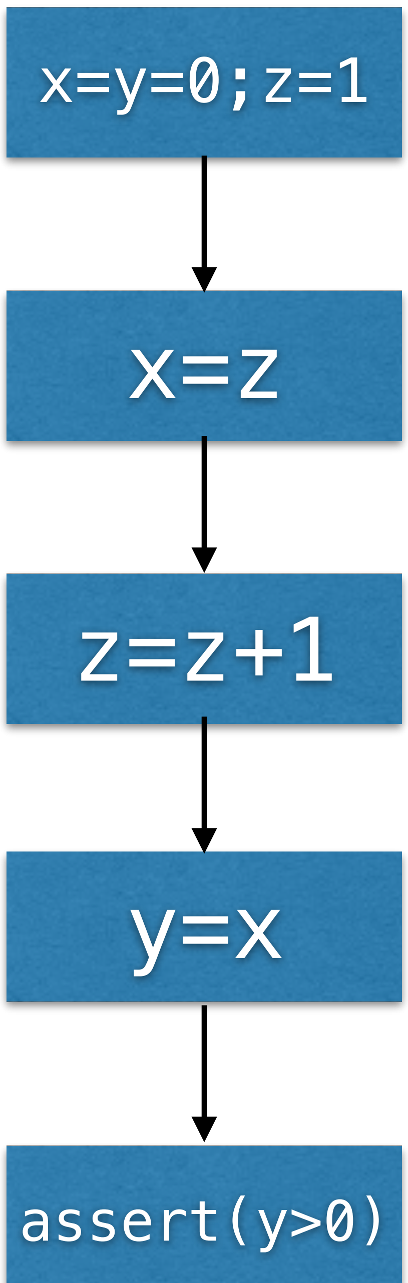
x	[0,0]
y	[0,0]
z	[1,1]

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]

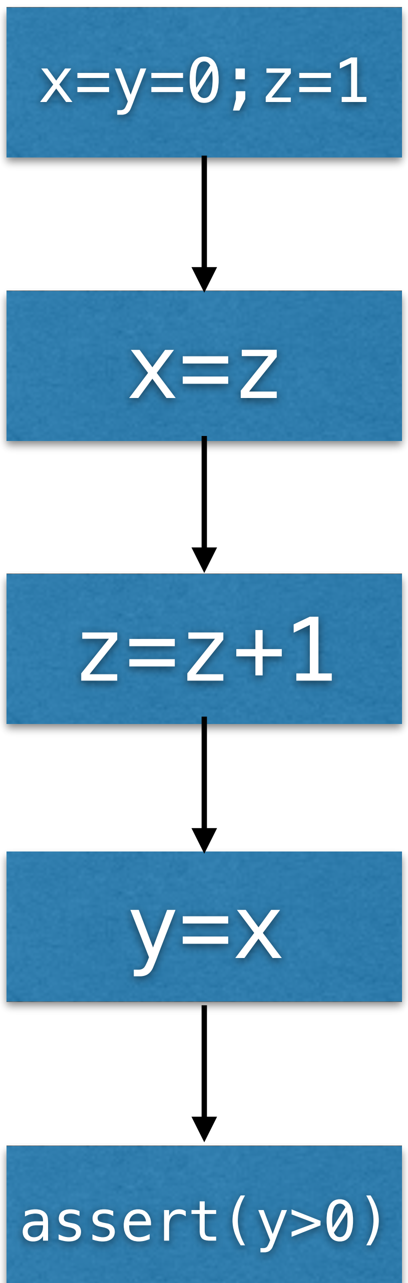
x	[1,1]
y	[0,0]
z	[1,1]

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

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

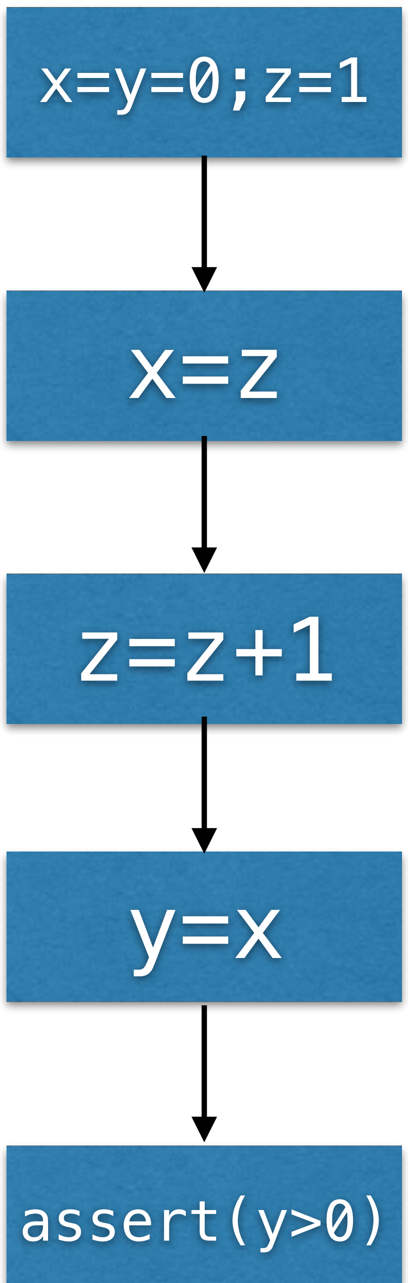
precise but costly

Flow-Insensitivity



x	[0,+∞]
y	[0,+∞]
z	[1,+∞]

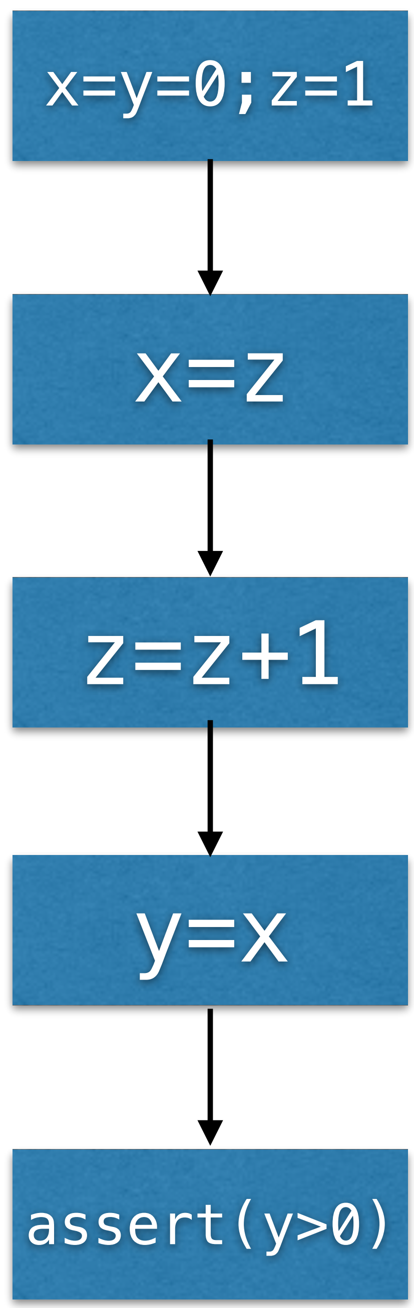
Flow-Insensitivity



x	[0,+∞]
y	[0,+∞]
z	[1,+∞]

cheap but imprecise

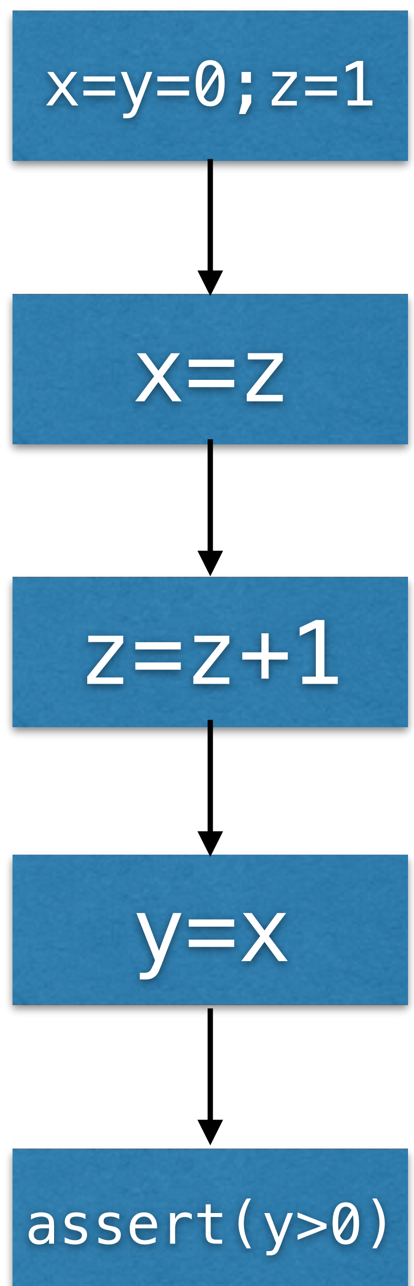
Selective Flow-Sensitivity



FS : {x}

FI : {y,z}

Selective Flow-Sensitivity



FS : {x}

x	[0,0]

x	[1,+∞]

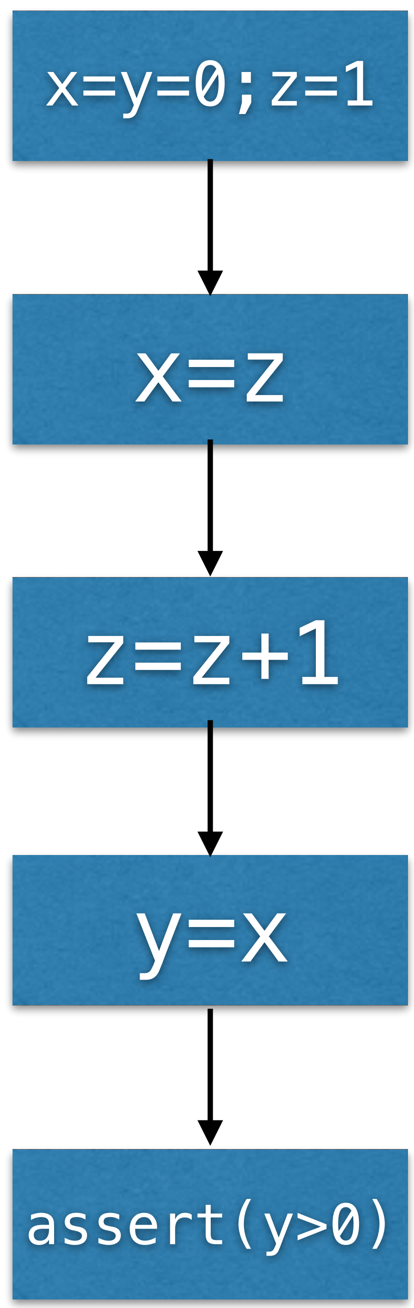
x	[1,+∞]

x	[1,+∞]

FI : {y,z}

y	[0,+∞]
z	[1,+∞]

Selective Flow-Sensitivity



FS : {x}

x	[0,0]

x	[1,+∞]

x	[1,+∞]

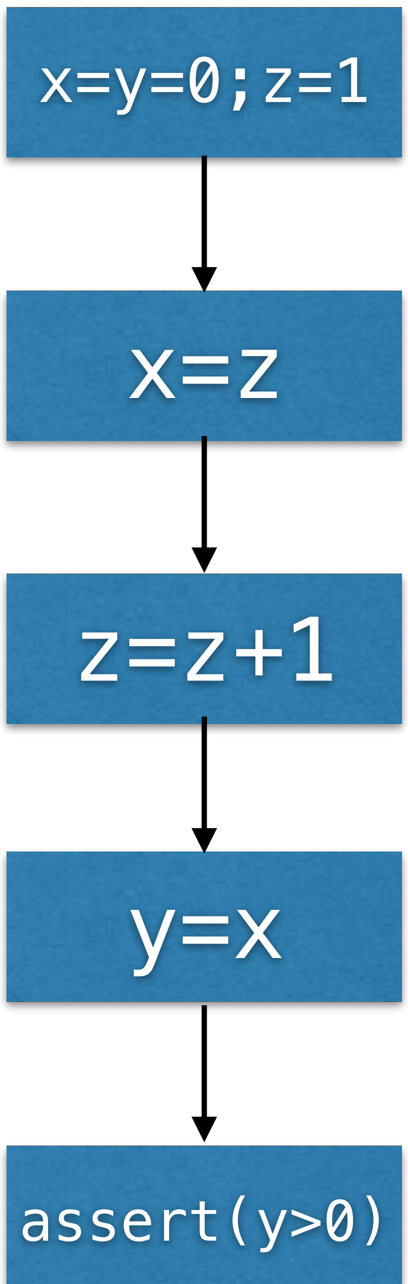
x	[1,+∞]

fail to prove

FI : {y,z}

y	[0,+∞]
z	[1,+∞]

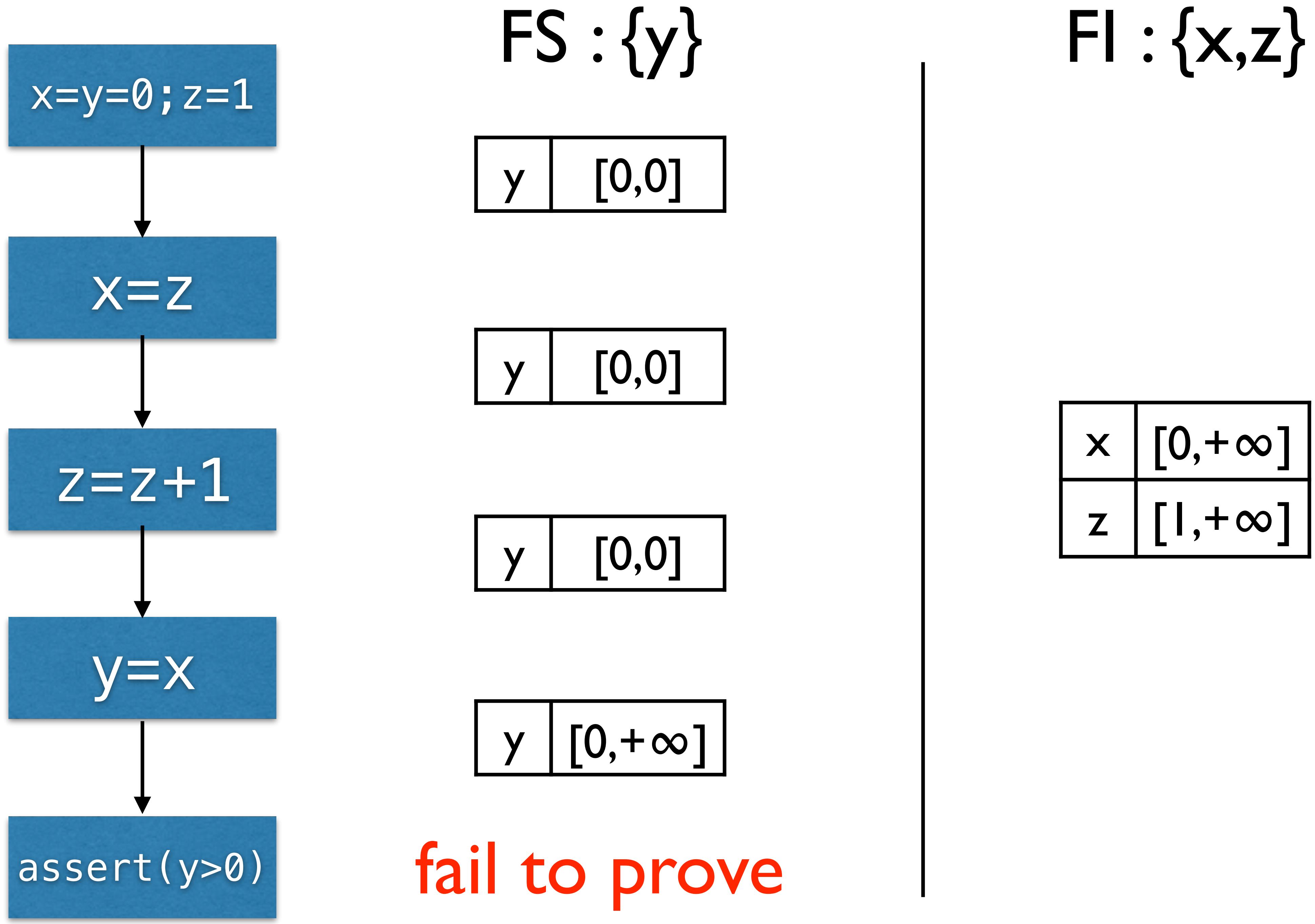
Selective Flow-Sensitivity



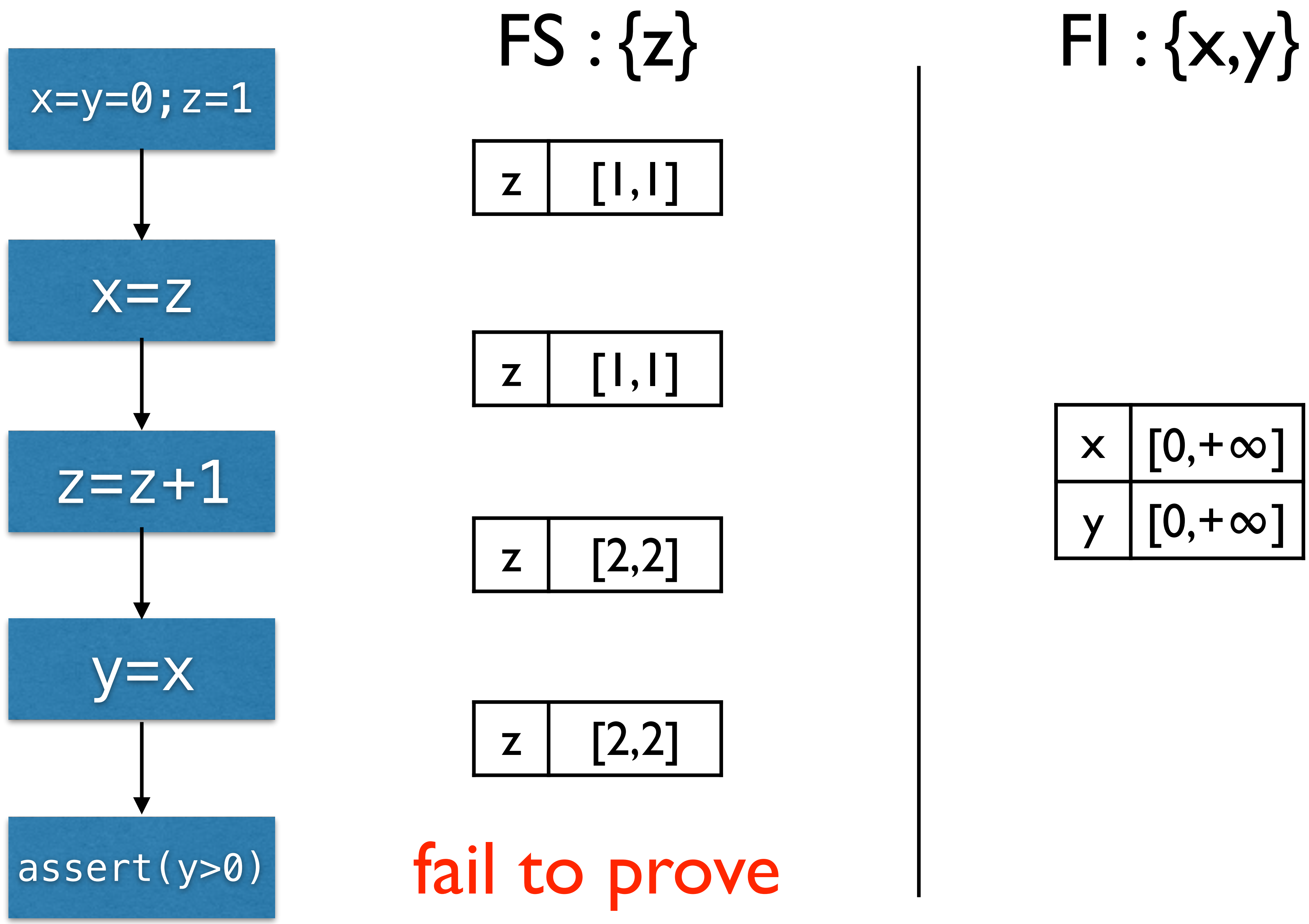
FS : {y}

FI : {x,z}

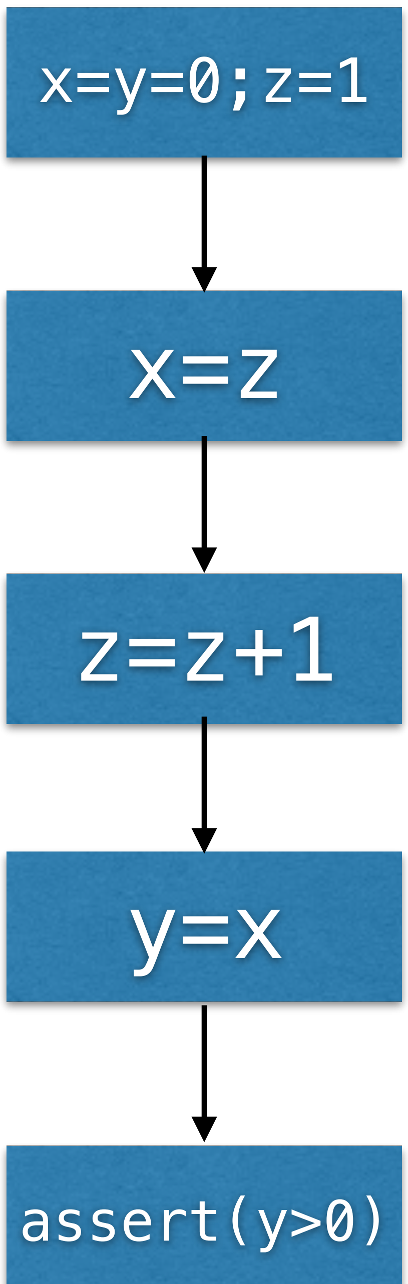
Selective Flow-Sensitivity



Selective Flow-Sensitivity



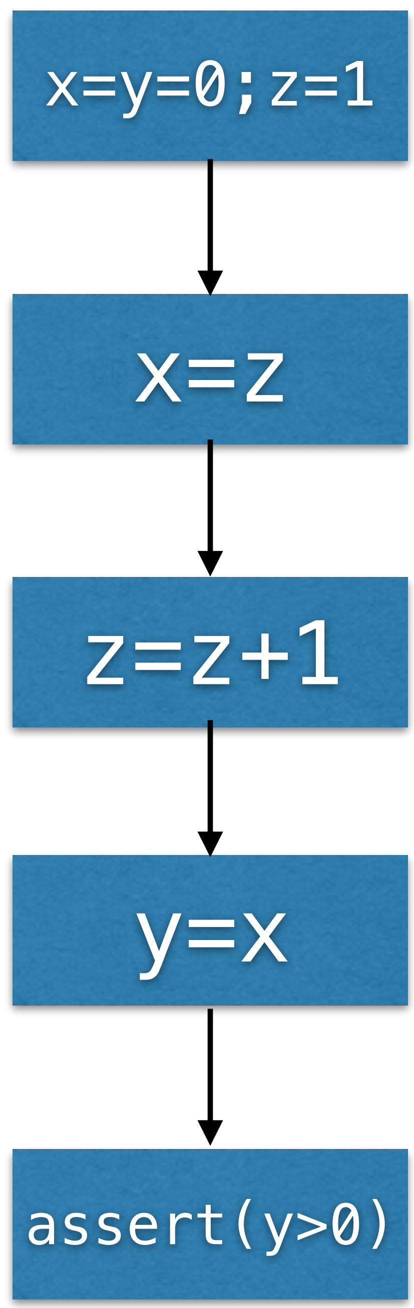
Selective Flow-Sensitivity



FS : {y,z}

FI : {x}

Selective Flow-Sensitivity



FS : {y,z}

y	[0,0]
z	[1,1]

y	[0,0]
z	[1,1]

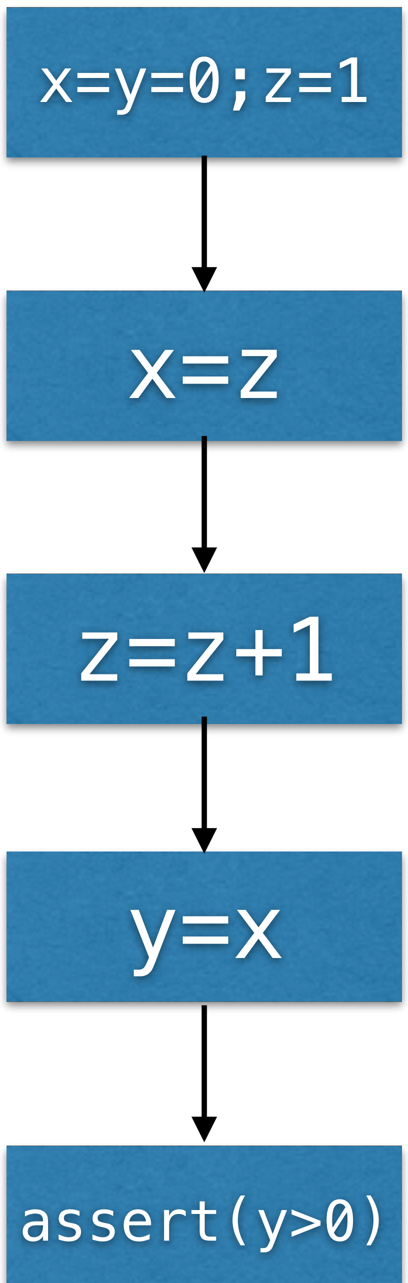
y	[0,0]
z	[2,2]

y	[0,+∞]
z	[2,2]

FI : {x}

x	[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]

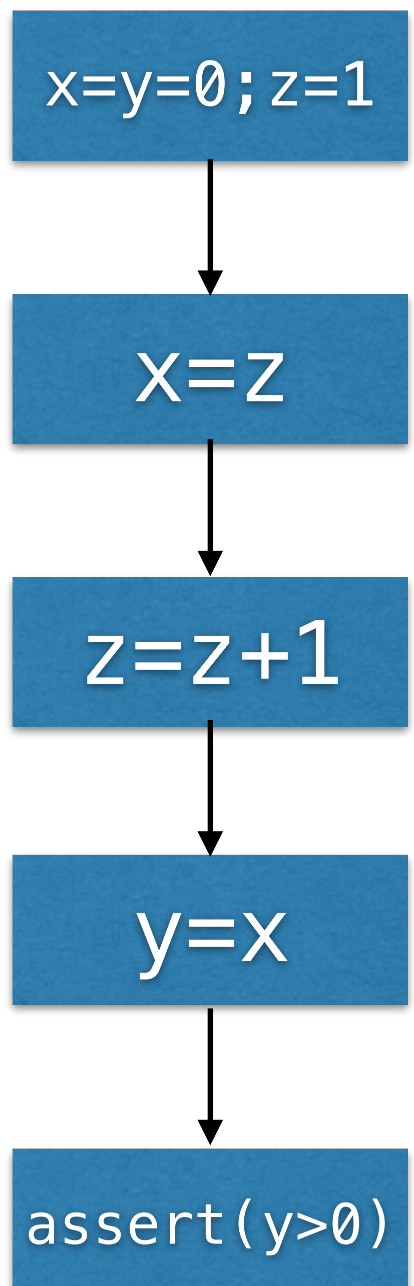
y	[0,+∞]
z	[2,2]

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}

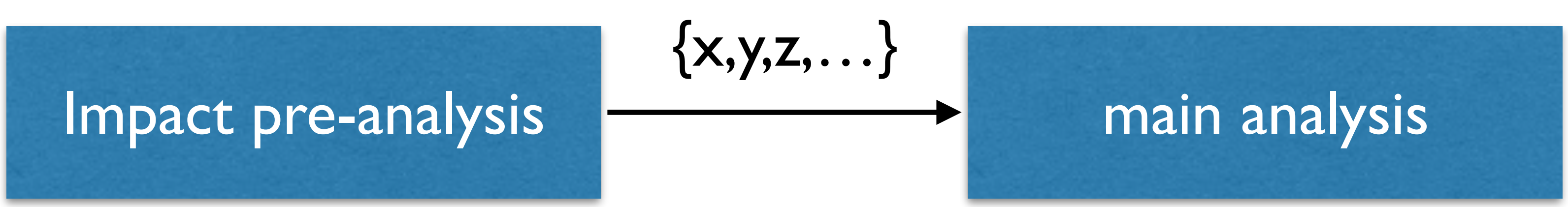
z	[1,+∞]

Finding a Good Program Abstraction is Challenging

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

Our Research

- How to efficiently find a good abstraction?
- ex) Impact pre-analysis [PLDI'14]



fully flow-sensitive but approximated in others

selectively flow-sensitive

This Talk

Learning-based Approach

This Talk

Learning-based Approach

- Parameterized adaptation strategy

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

This Talk

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

This Talk

Learning-based Approach

- Parameterized adaptation strategy

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

- Learn a good parameter *W* from existing codebase

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

Codebase

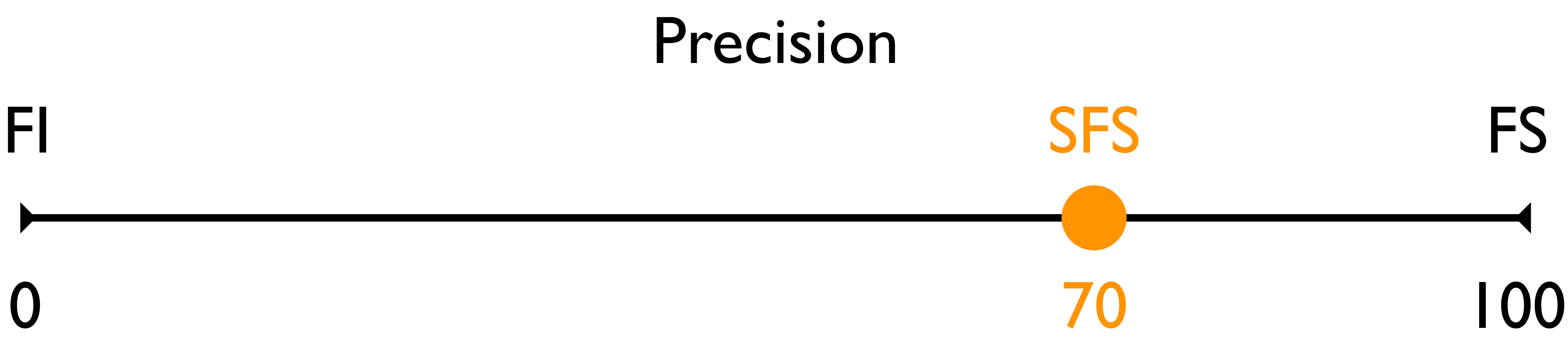
- For new program P, run static analysis with $S_W(P)$

Effectiveness

- Implemented in Sparrow, an interval analyzer for C
- Evaluated on open-source benchmarks

Effectiveness

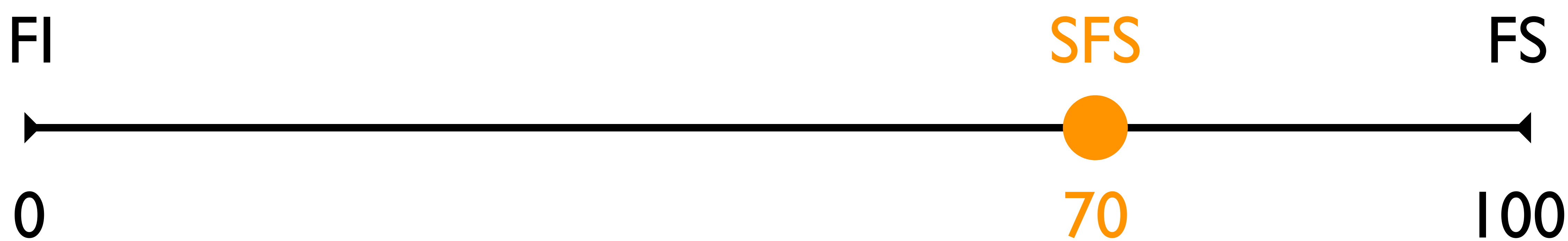
- Implemented in Sparrow, an interval analyzer for C
- Evaluated on open-source benchmarks



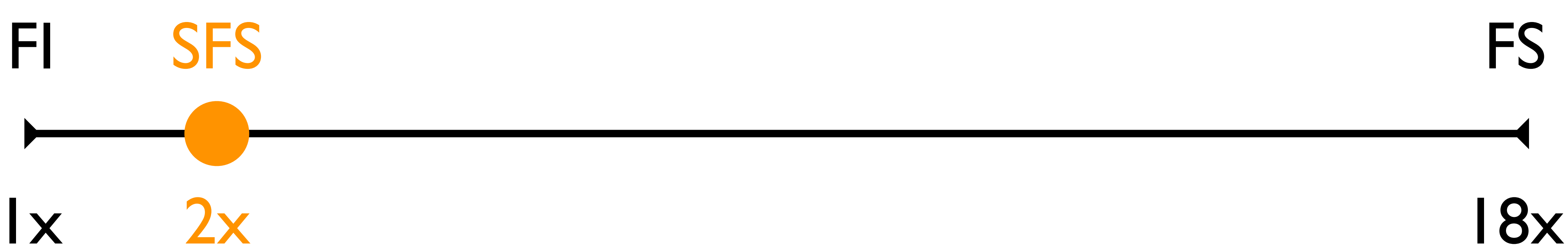
Effectiveness

- Implemented in Sparrow, an interval analyzer for C
- Evaluated on open-source benchmarks

Precision



Cost



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:

$$S_w : \text{pgm} \to 2^{\text{Var}}$$

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:

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

(1) Features

- Predicates over variables:

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

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

(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,

score(x) = 0.3
score(y) = 0.6
score(z) = 0.1

$\Rightarrow$ {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

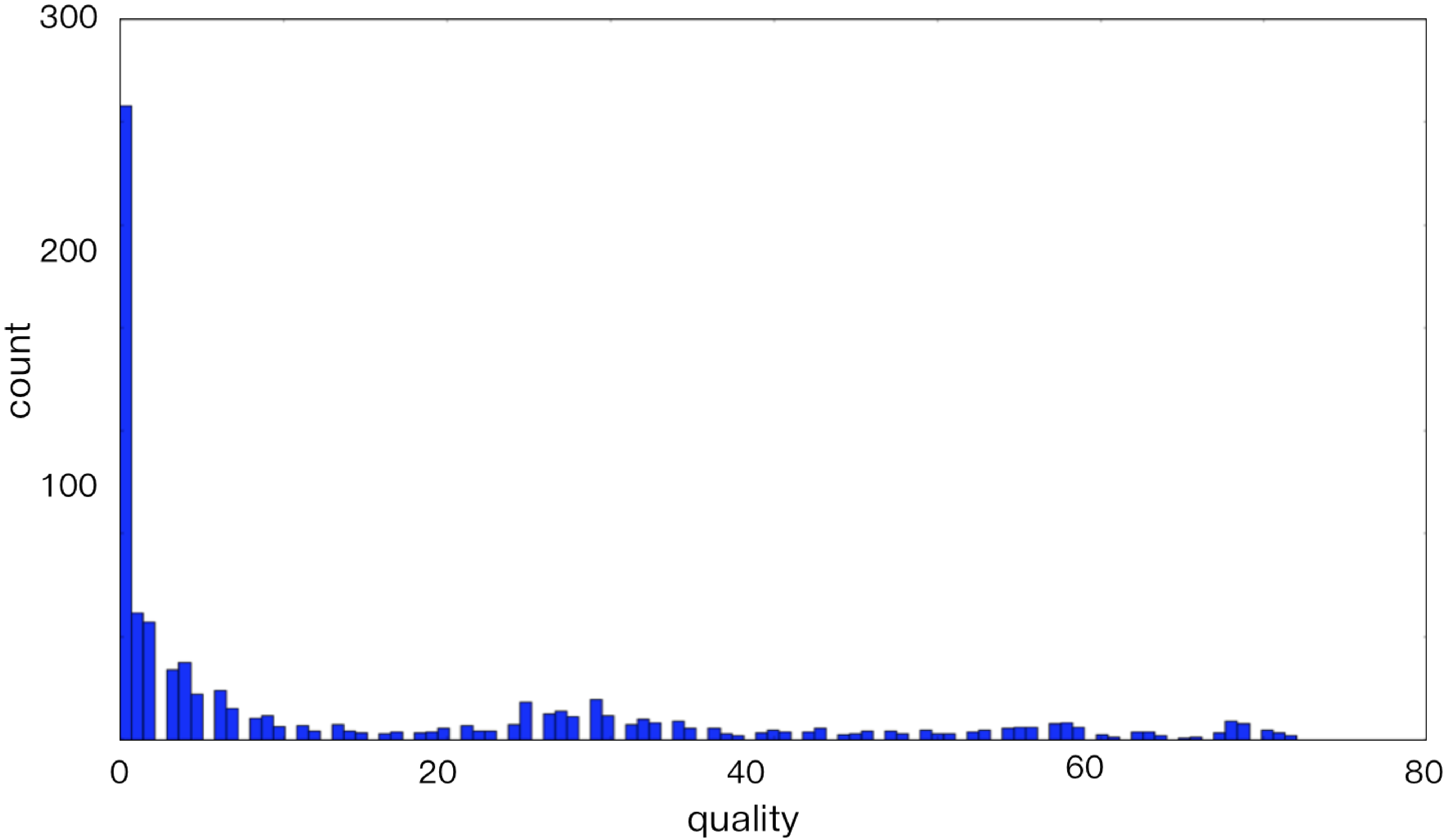
repeat N times

pick **w** $\in$ R^n randomly

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

return best **w** found

Learning via Random Sampling



Our Approach: Learning via Bayesian Optimization

- A powerful method for solving difficult optimization problems.
- Especially powerful when the objective function is expensive to evaluate.
- Key idea: use a probabilistic model to reduce the number of objective function evaluations.

Learning via Bayesian Optimization

repeat N times

select a promising w using the model

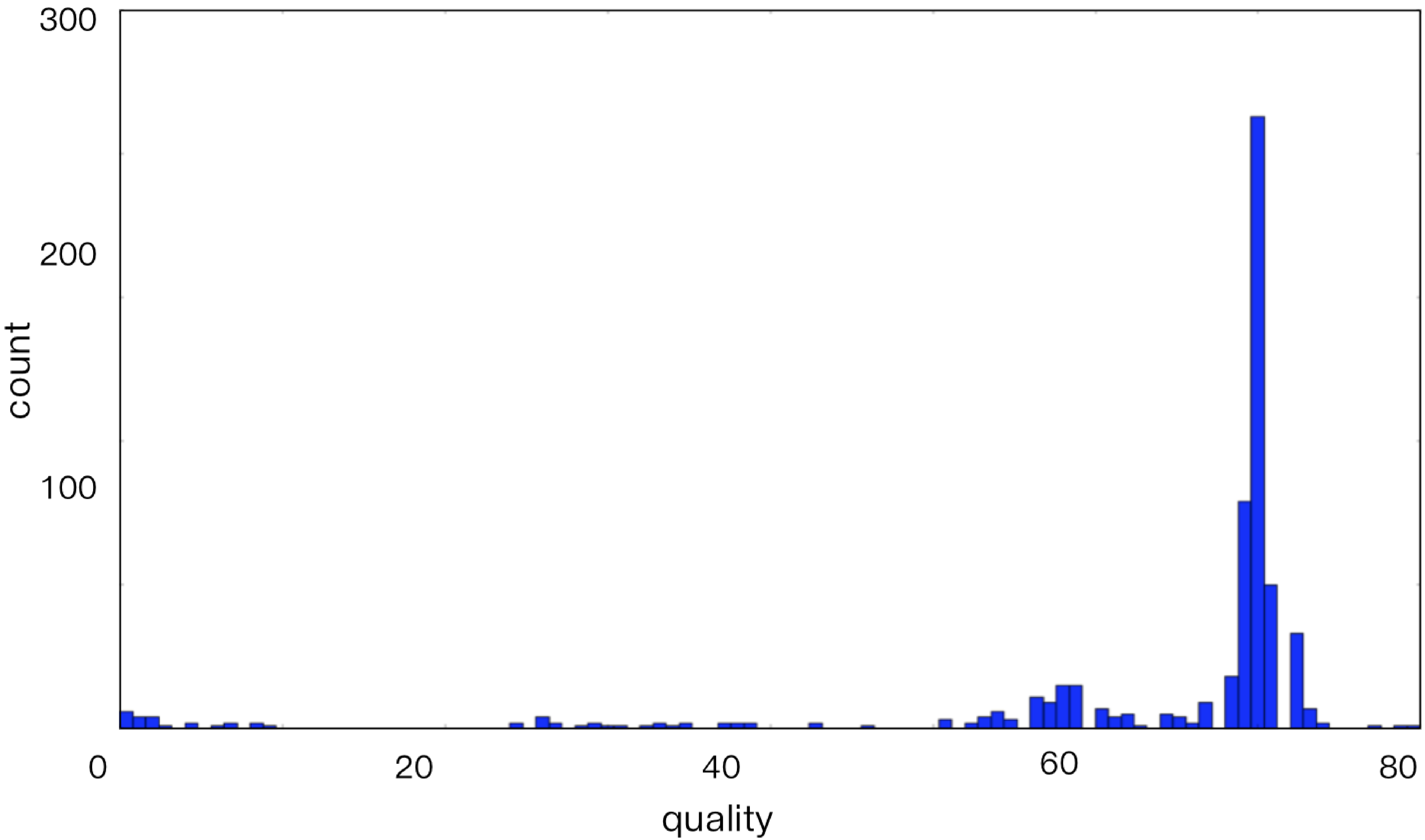
evaluate $\sum_{P_i} F(P_i, S_{\mathbf{w}}(P_i))$

update the probabilistic model

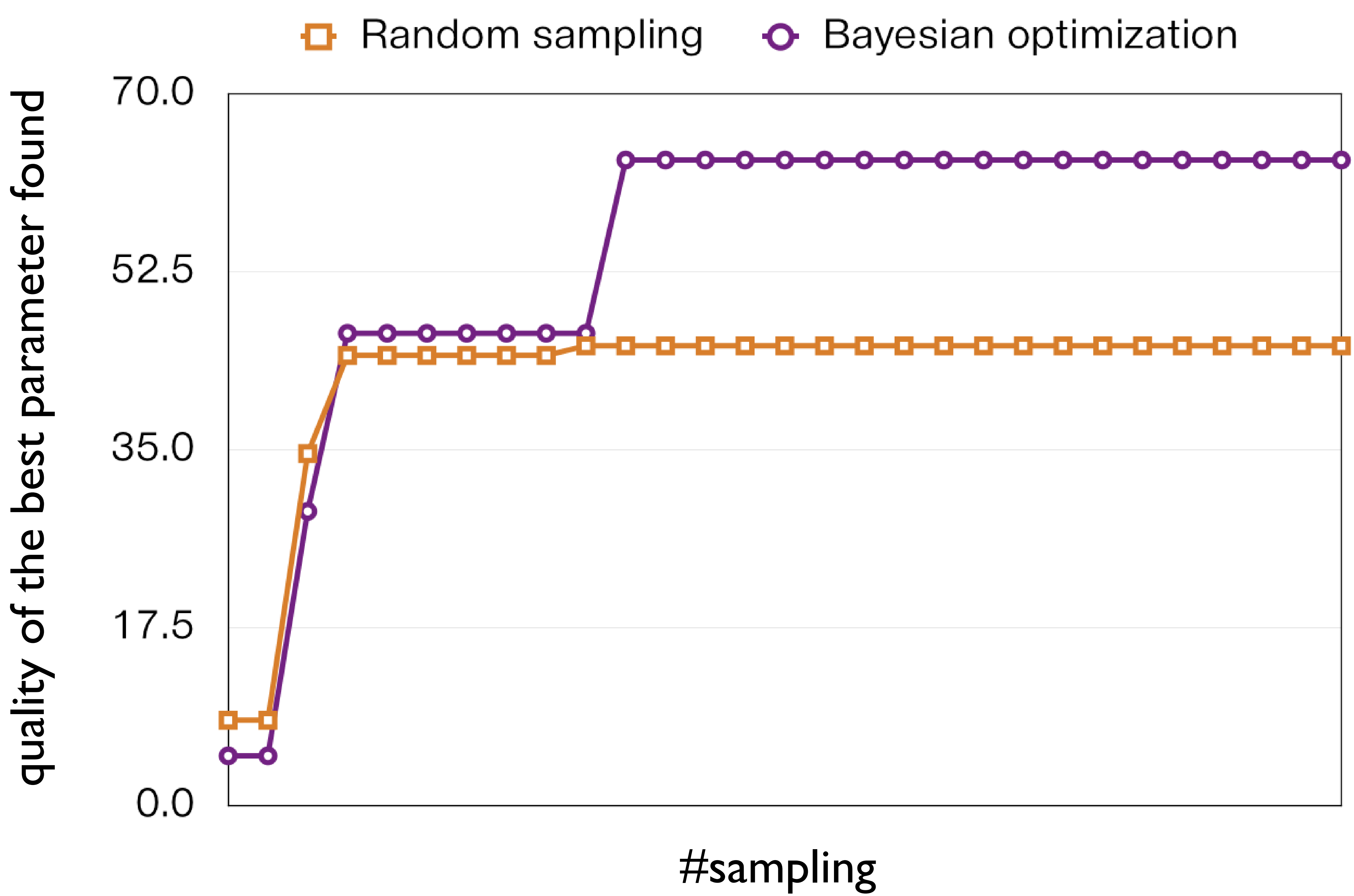
return best w found

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

Learning via Bayesian Optimization



Random Sampling vs 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 open-source C programs (1K ~ 100KLoC)
  - 20 programs for training
  - 10 programs for testing

Performance

Flow-Sensitivity (12 hour time budget)

	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**

Performance

Flow-Sensitivity (12 hour time budget)

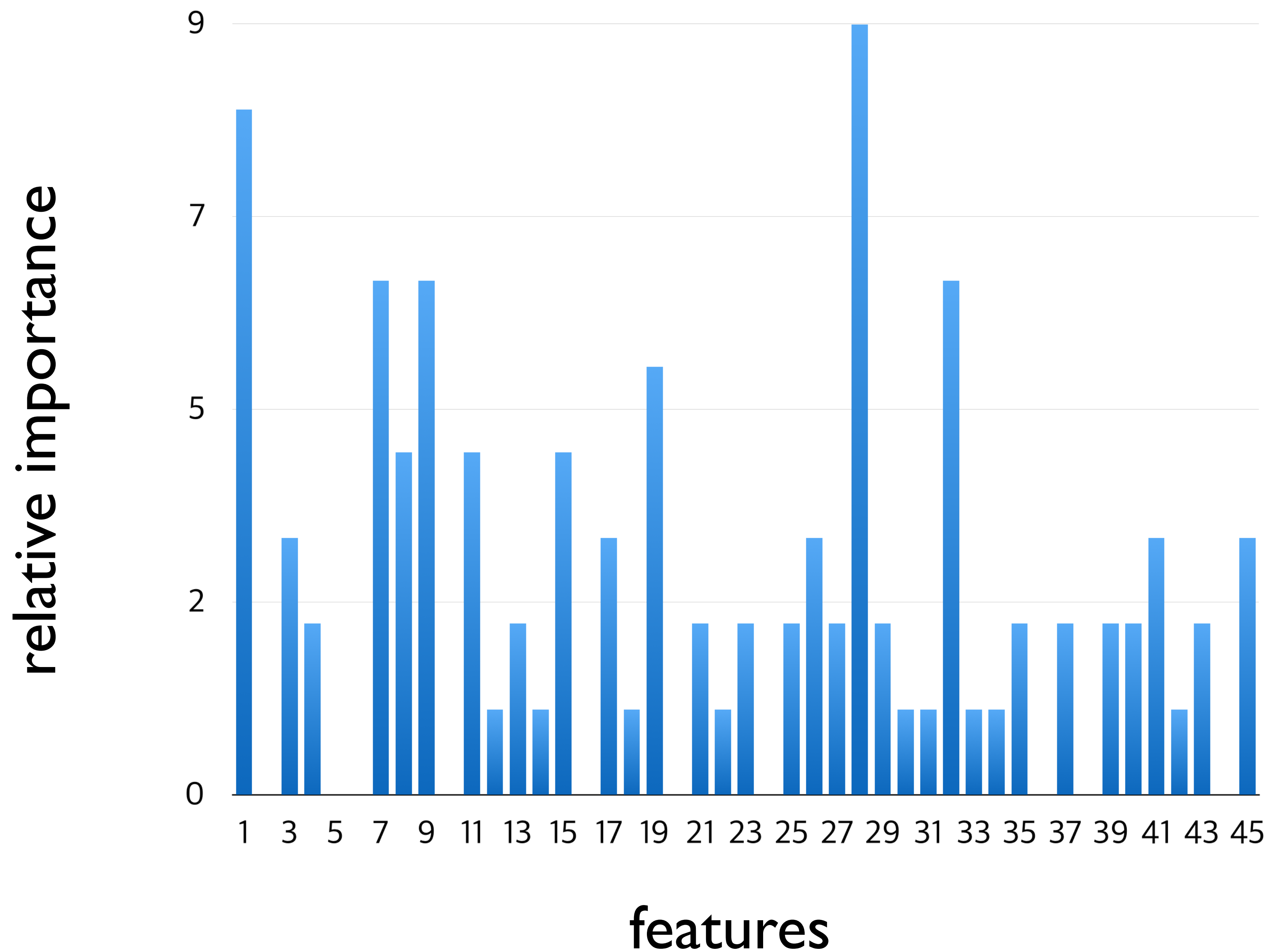
	Training				Testing								
	FI	FS	partial FS		FI		FS			partial FS			
Trial	prove	prove	prove	quality	prove	sec	prove	sec	cost	prove	sec	quality	cost
1	6,383	7,316	7,089	75.7 %	2,788	48	4,009	627	13.2 x	3,692	78	74.0 %	1.6 x
2	5,788	7,422	7,219	87.6 %	3,383	55	3,903	531	9.6 x	3,721	93	65.0 %	1.7 x
3	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 (12hrs)

	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**

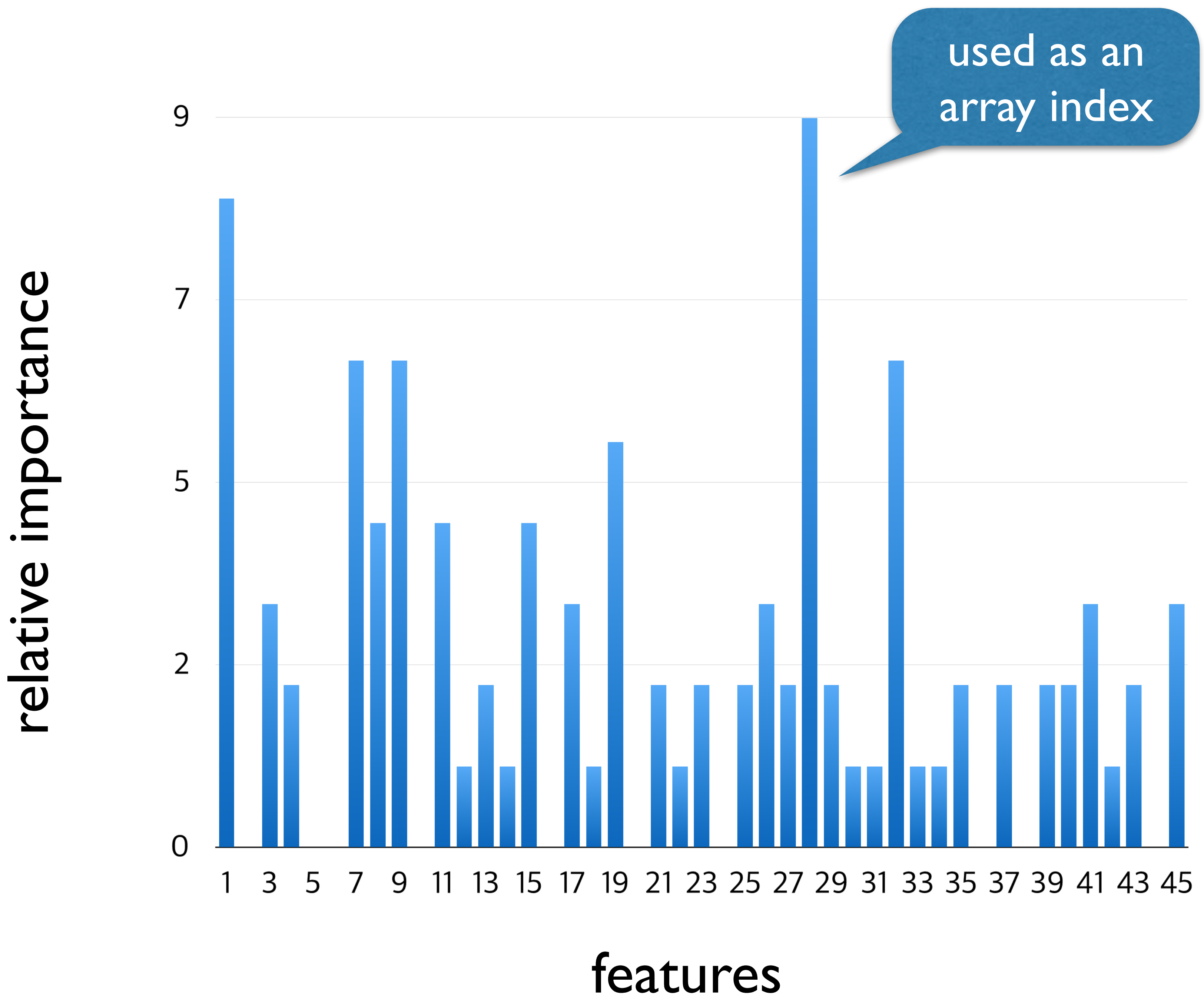
Insights on Flow-Sensitivity

- Relative importance between program features:



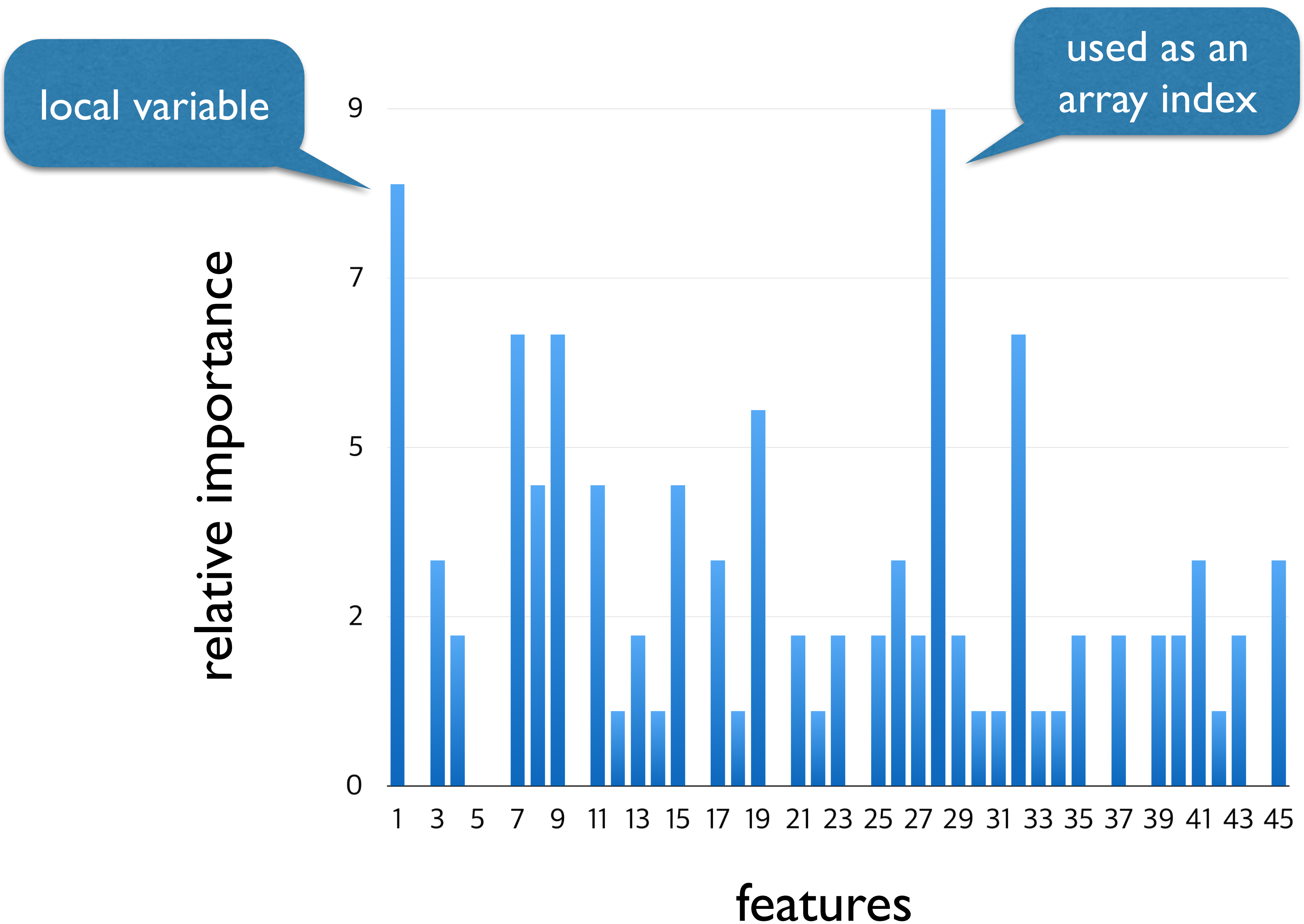
Insights on Flow-Sensitivity

- Relative importance between program features:



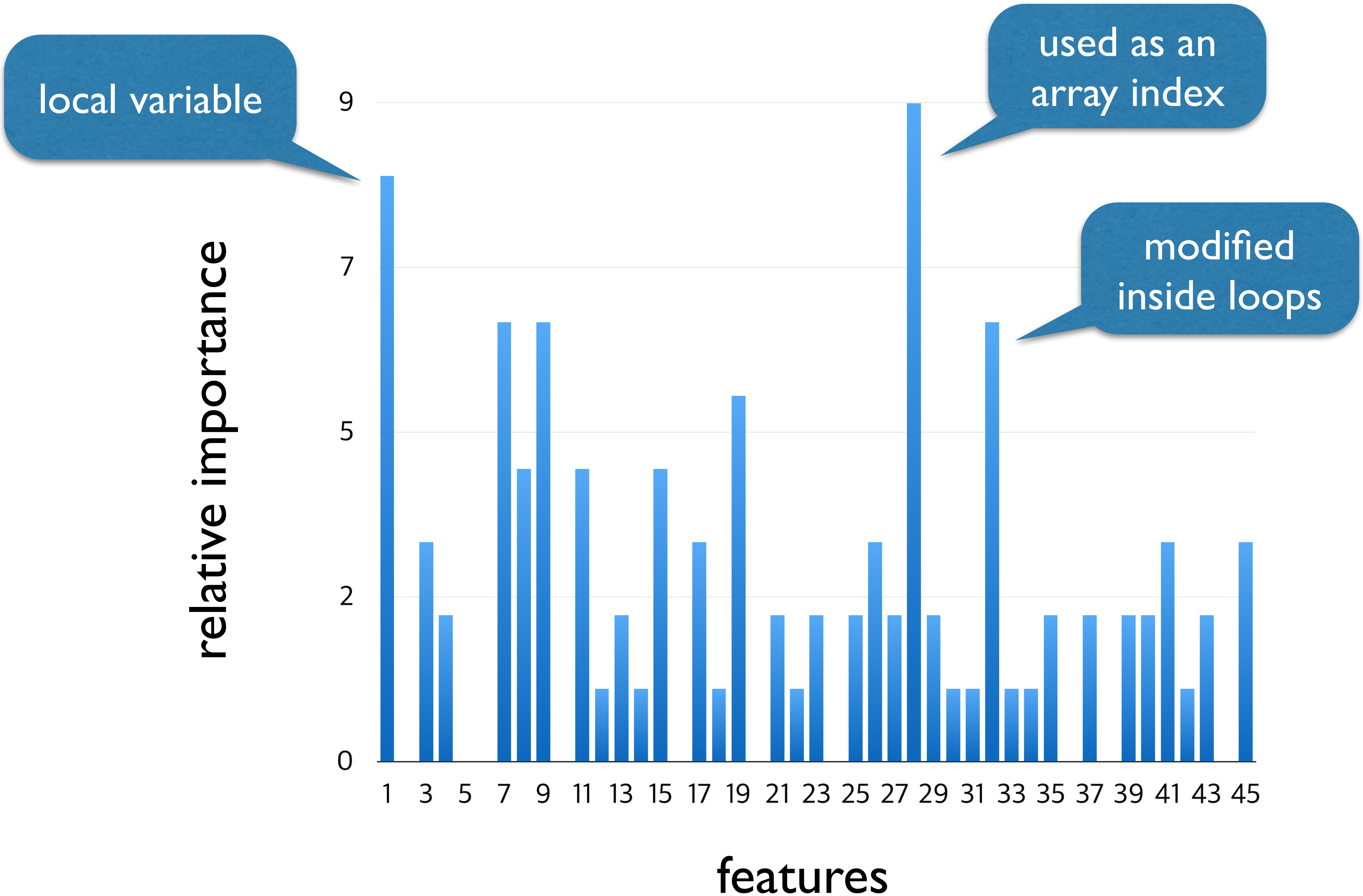
Insights on Flow-Sensitivity

- Relative importance between program features:



Insights on Flow-Sensitivity

- Relative importance between program features:



Insights on Flow-Sensitivity

- Typical scenario where flow-sensitivity helps:

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

Insights on Flow-Sensitivity

- 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')  ...
```

Insights on Flow-Sensitivity

- 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

Insights on Flow-Sensitivity

- 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

Insights on Flow-Sensitivity

- Also provide insights difficult to find manually:

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

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

Insights on Flow-Sensitivity

- Also provide insights difficult to find manually:

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: 70% precision with 2x cost
- Generally applicable to any static analysis

Summary

Our Learning-based Approach

- First machine learning-based approach
  - formulated as an optimization problem
  - solved by Bayesian optimization
- Effective: 70% precision with 2x cost
- Generally applicable to any static analysis

Thank you