Data-Driven Static Analysis

Hakjoo Oh

25 April 2023 @IFIP WG 2.4 Meeting 67, York Harbor

PL/SE Research @Korea Univ.

- Members: 10 PhD and 5 MS students
- Research areas: programming languages (PL), software engineering (SE), software security
  - program analysis and testing
  - program synthesis and repair

- Publication: in PL, SE, and Security venues:
  - PL: POPL('22),PLDI('12,'14,'20),OOPSLA('15,'17a,'17b,'18a,'18b,'19,'20,'23)
  - SE: ICSE('17,'18,'19,'20,'21'22a,'22b,'23a,'23b,'23c), FSE('18,'19,'20,'21,'22),ASE('18)
  - Security: Oakland('17,'20), USENIX Security('21,'23)

http://prl.korea.ac.kr

PL/SE Research

- Members: 10 PhD and 5 MS stu
- Research areas: programming la (SE), software security

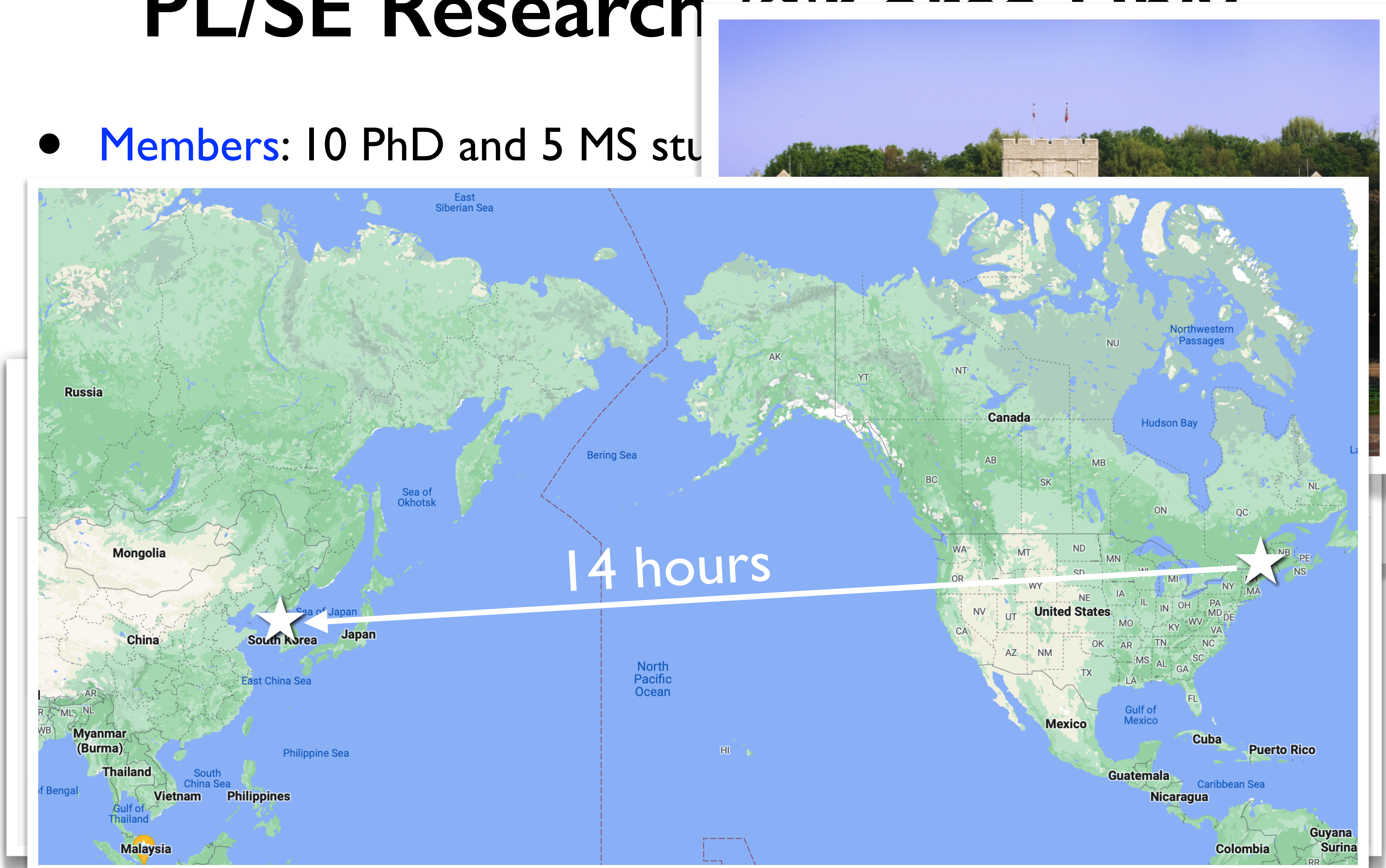
Tell me about Korea University

Korea University is a private research university located in Seoul, South Korea. It was founded in 1905 and is considered one of the country's oldest and most prestigious institutions of higher education. The university is composed of 16 undergraduate colleges and 20 graduate schools, offering a wide range of academic programs across various fields of study.

http://prl.korea.ac.kr

PL/SE Research

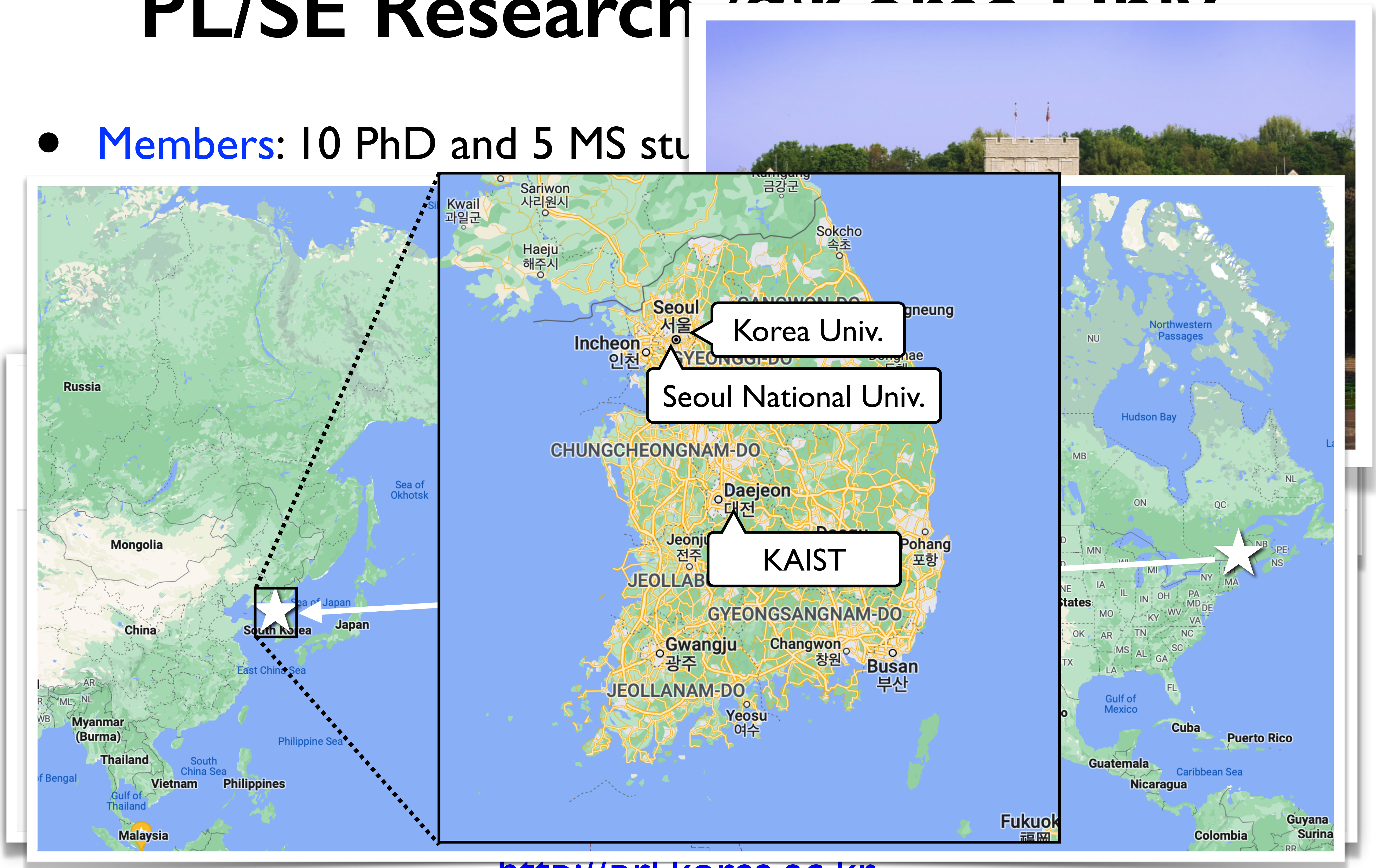
- Members: 10 PhD and 5 MS stu



http://prl.korea.ac.kr

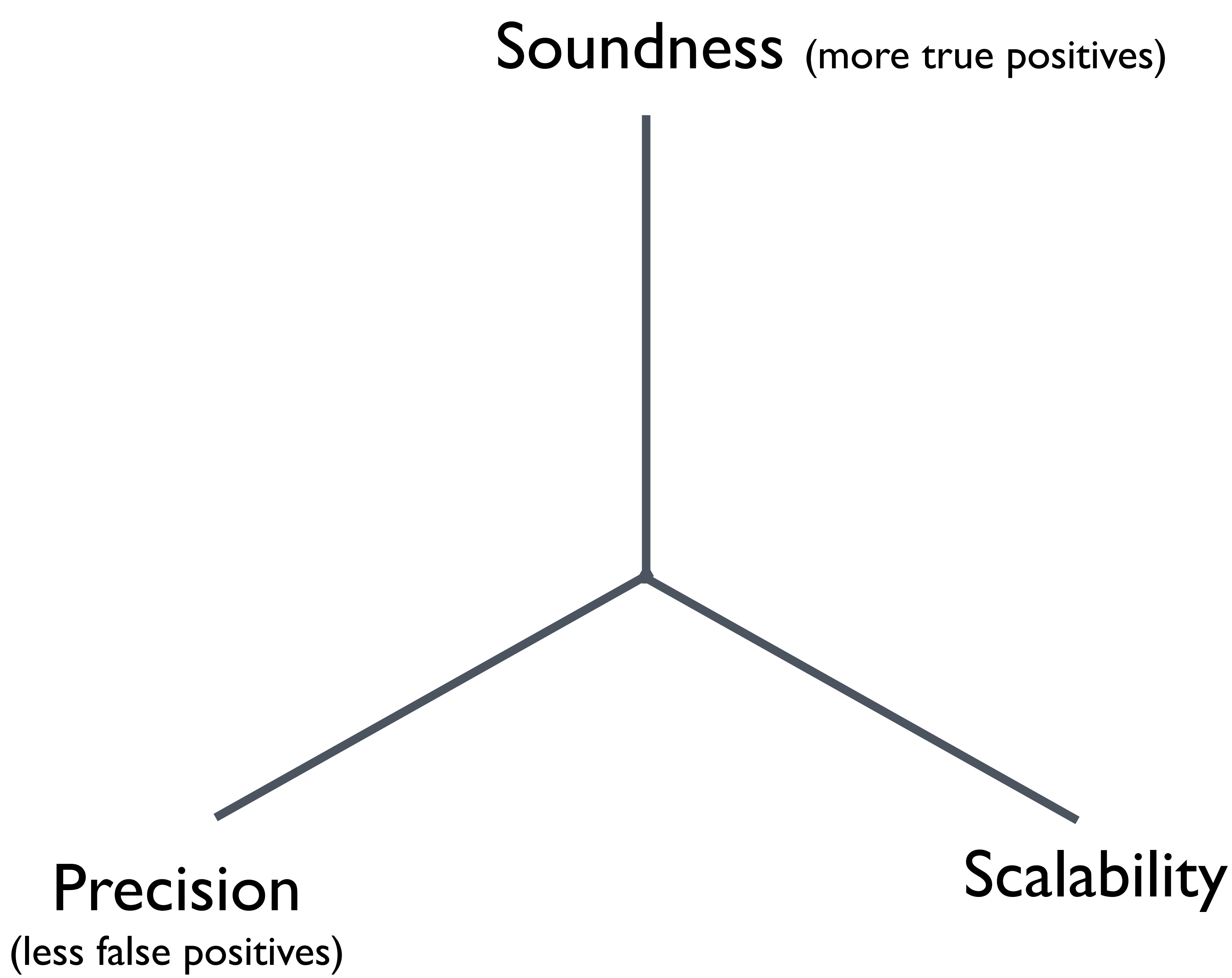
PL/SE Research @Korea Univ.

- Members: 10 PhD and 5 MS stu

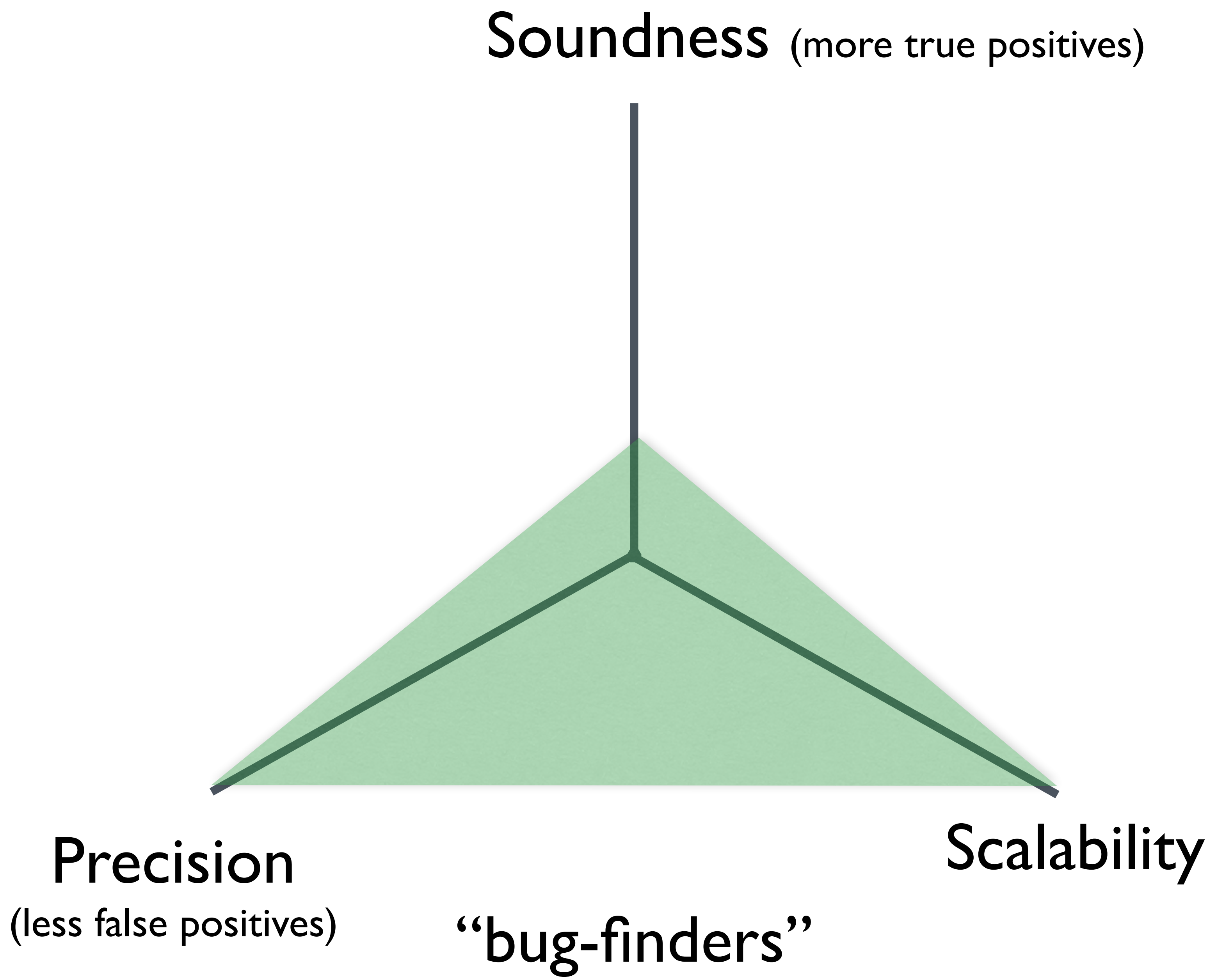


http://prl.korea.ac.kr

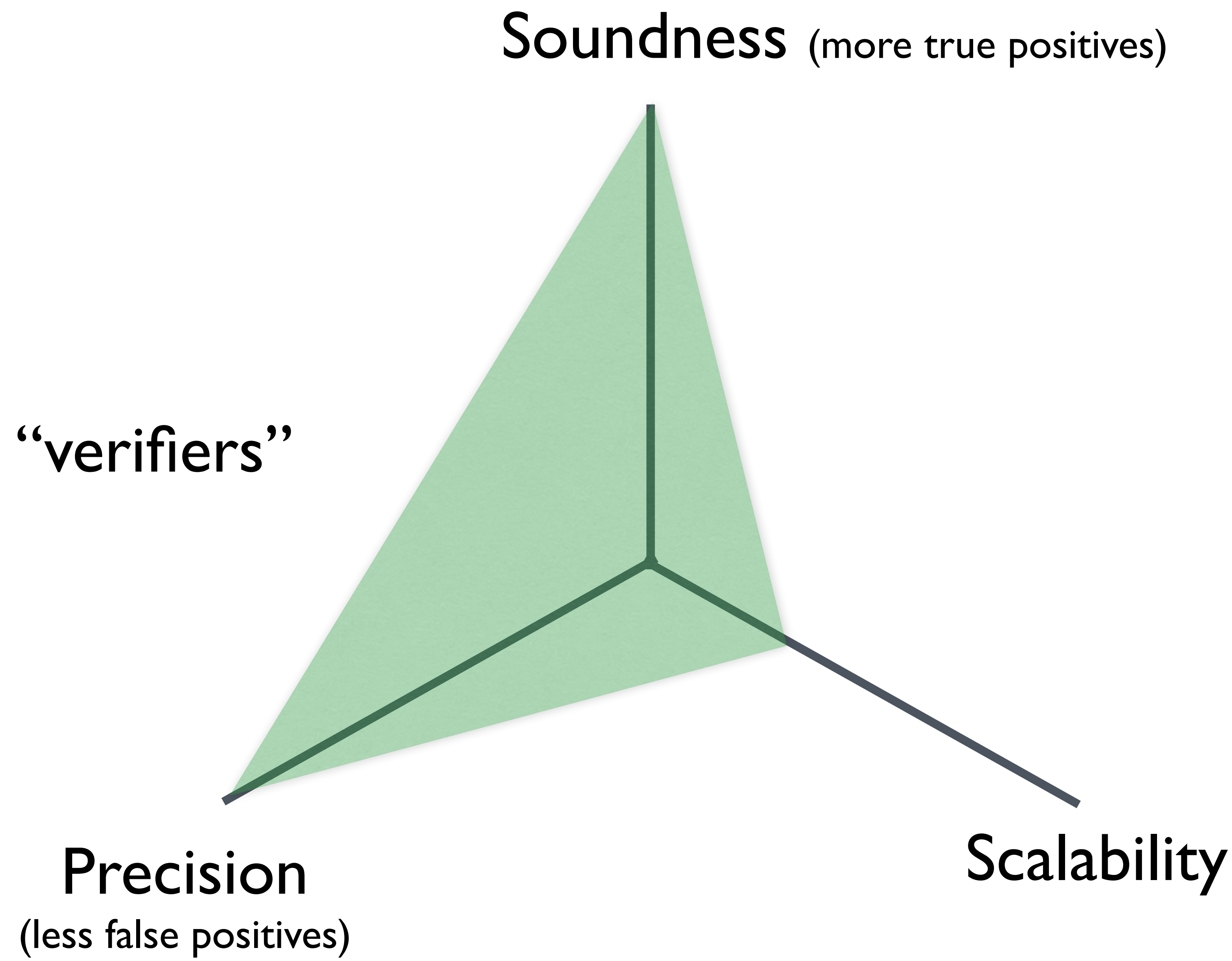
Tradeoff in Static Analysis



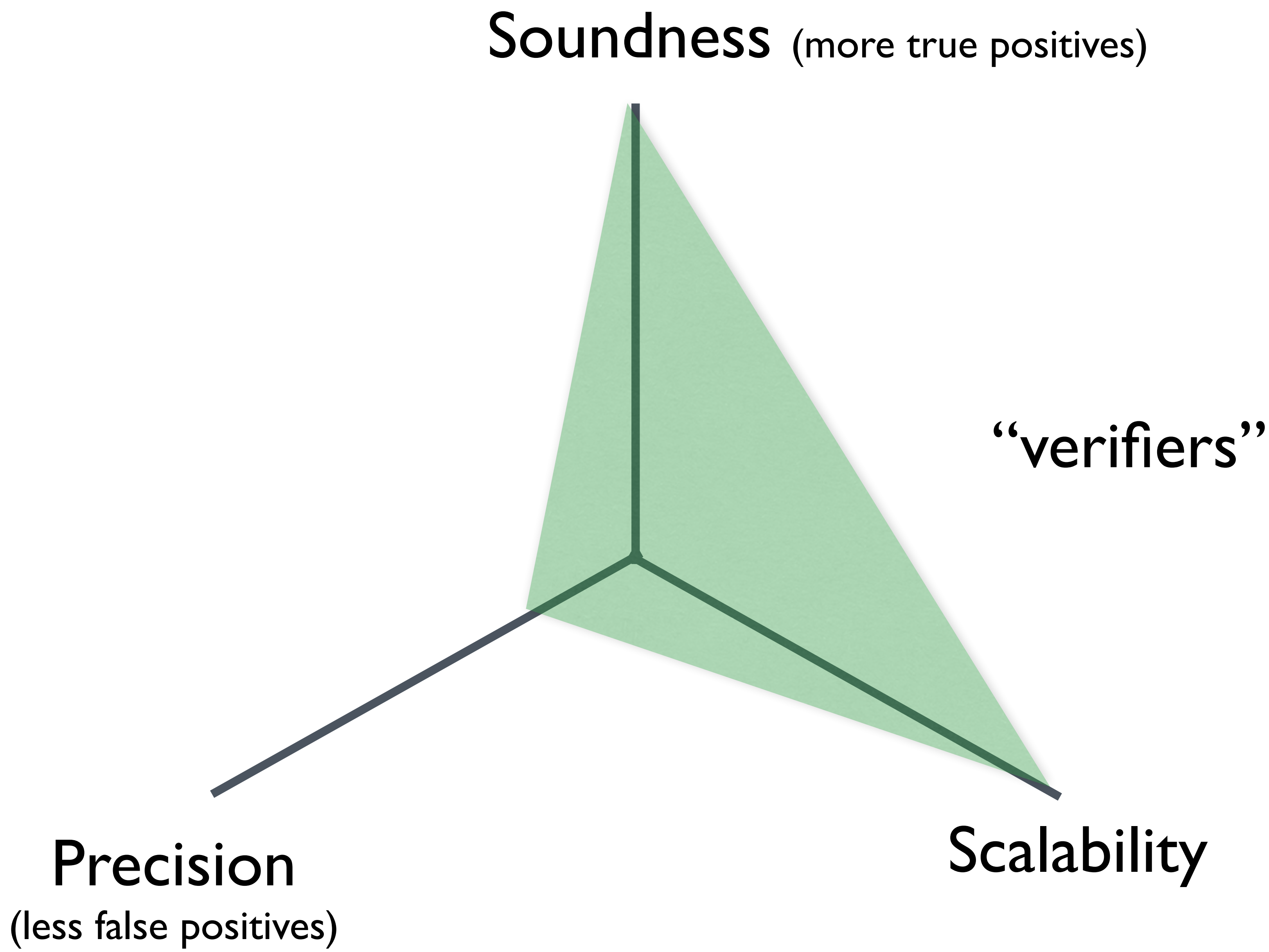
Tradeoff in Static Analysis



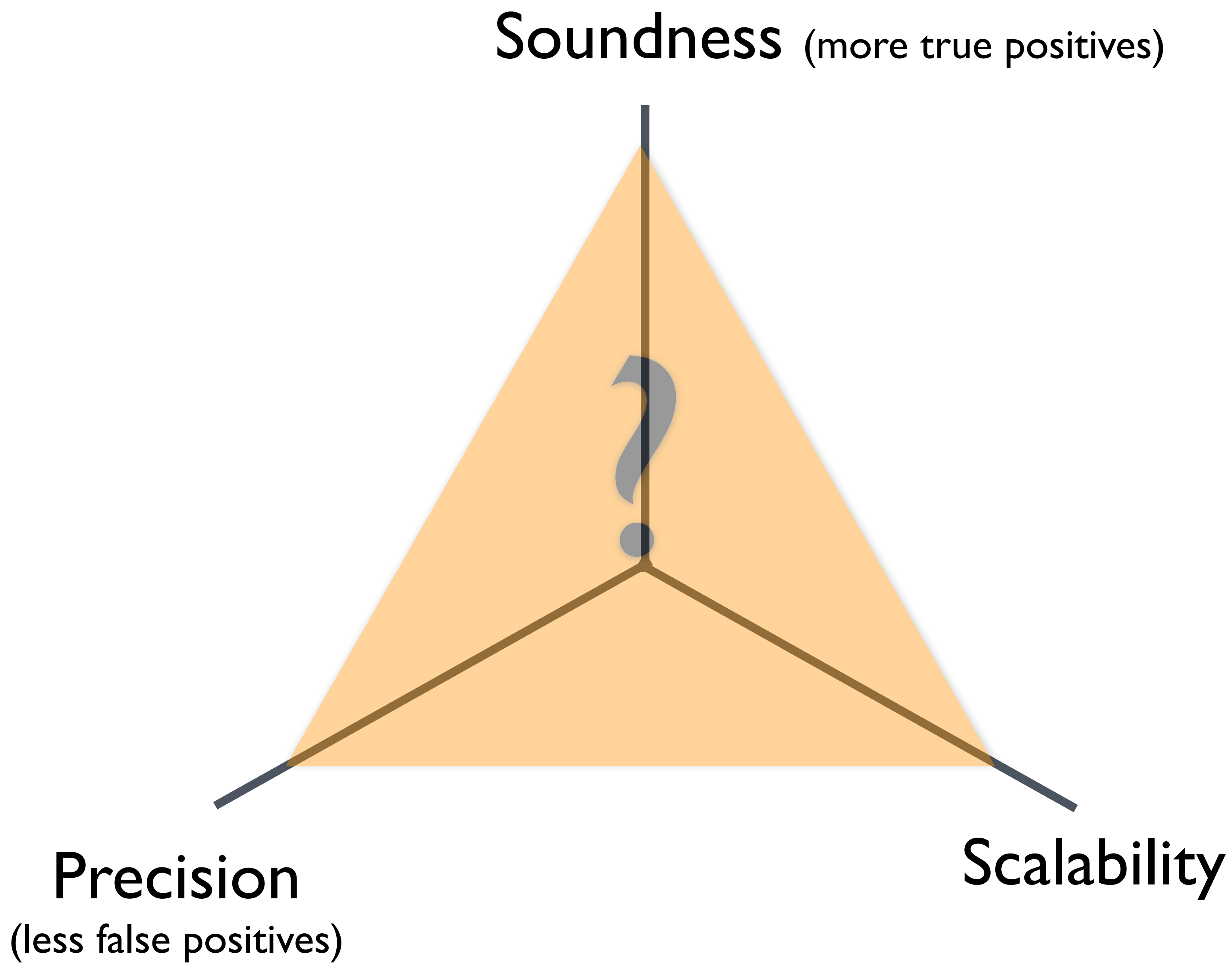
Tradeoff in Static Analysis



Tradeoff in Static Analysis



Tradeoff in Static Analysis



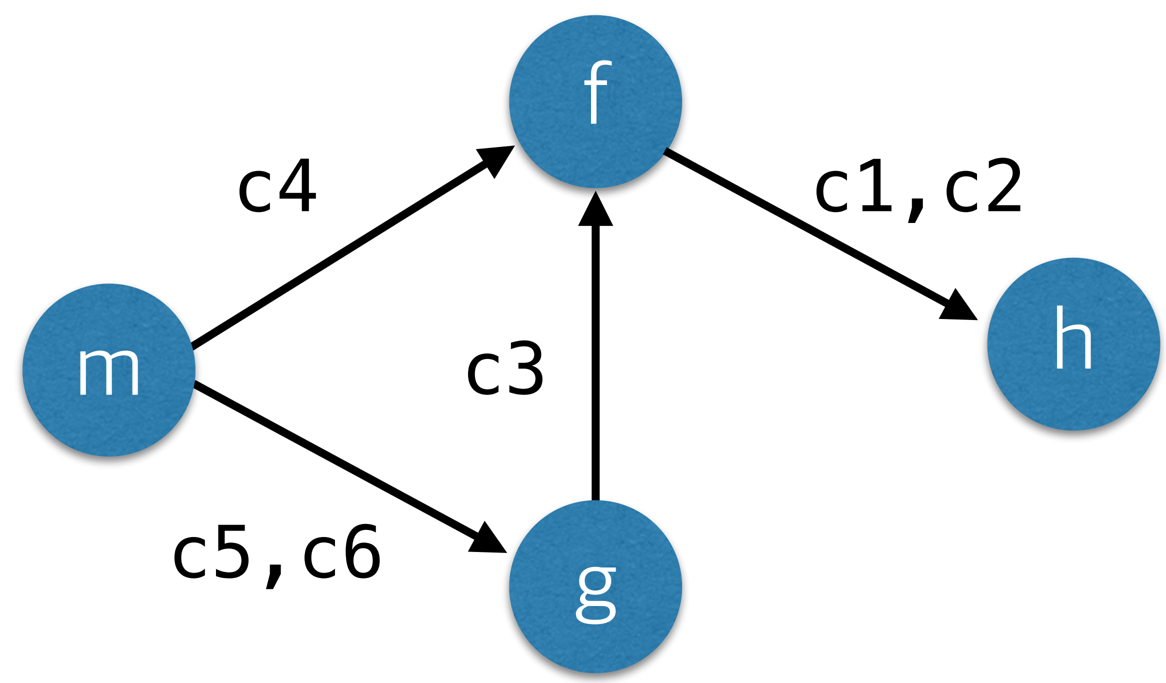
This talk: Using machine learning to balance soundness, precision, and scalability

Example 1: Balancing Precision and Scalability in Sound Analysis

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 0);  //  Query: always holds (x is 4 or 8)
c2:   y = h(input());
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

Context Insensitivity

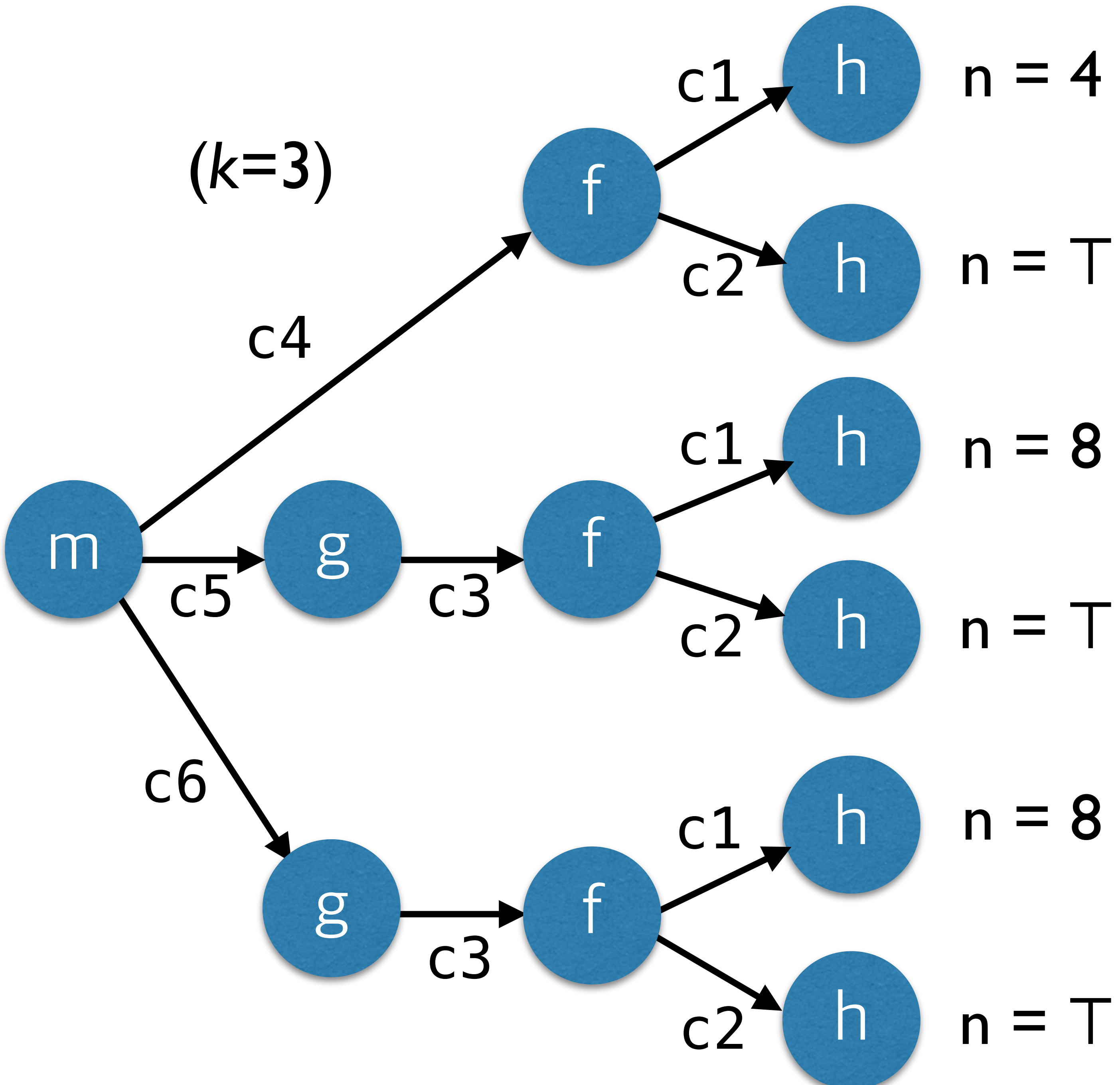
```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 0);
c2:   y = h(input());
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```



cheap but imprecise

Context Sensitivity (*k*-CFA)

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 0);
c2:   y = h(input());
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

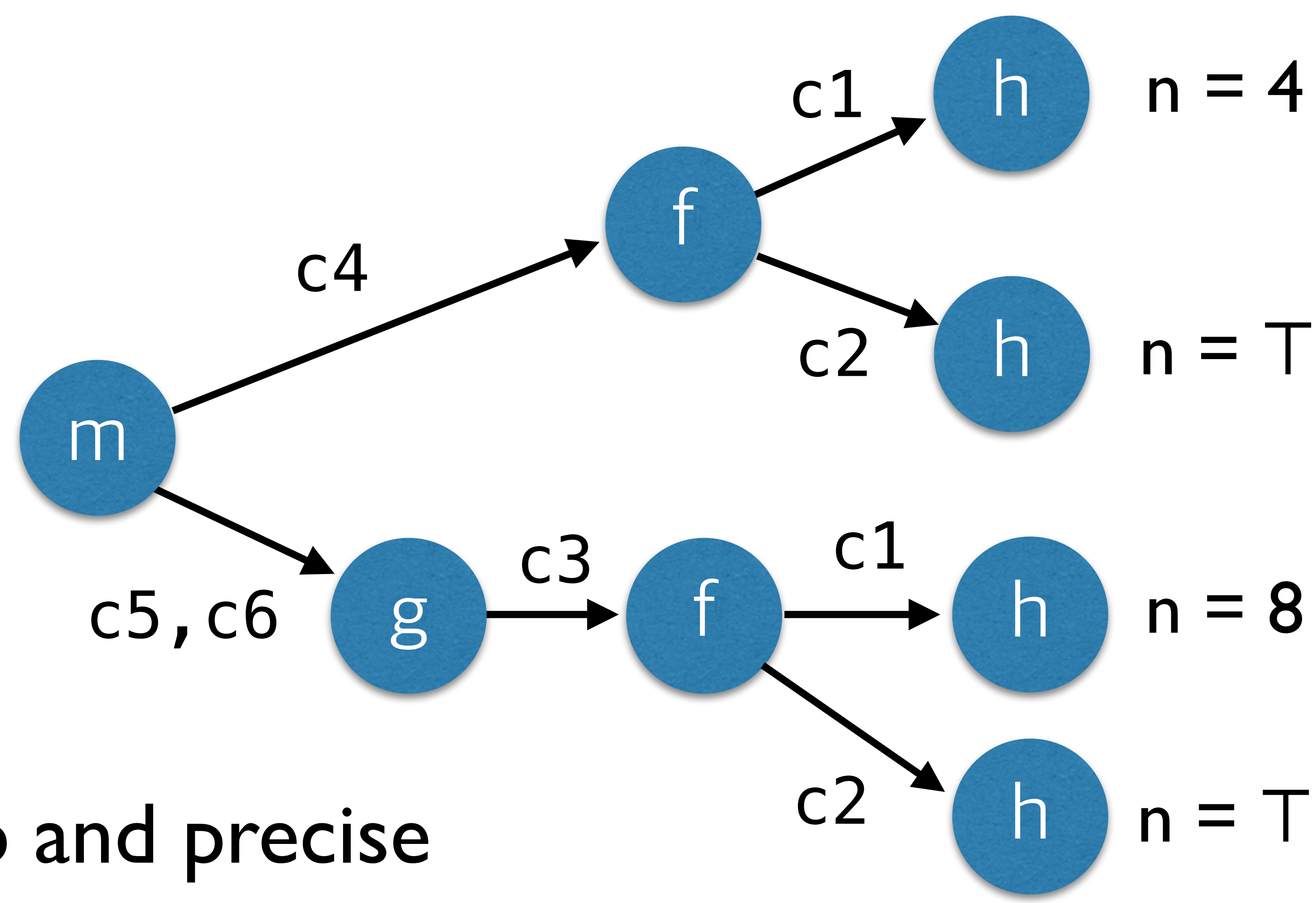


precise but expensive

Selective Context Sensitivity

```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 0);
c2:   y = h(input());
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

Apply 2-ctx-sens: {h}
Apply 1-ctx-sens: {f}
Apply 0-ctx-sens: {g, m}



cheap and precise

Hard Search Problem

Apply 2-ctx-sens: {h}
Apply 1-ctx-sens: {f}
Apply 0-ctx-sens: {g, m}

> "How to find a good program abstraction?"

Hard Search Problem

Apply 2-ctx-sens: {h}
Apply 1-ctx-sens: {f}
Apply 0-ctx-sens: {g, m}

"How to find a good program abstraction?"

- Intractably large search space, if not infinite
  - e.g., $(k+1)^{|Func|}$ difference abstractions for context sensitivity
- Few solutions: many abstractions too imprecise or costly

Hard Search Problem

Apply 2-ctx-sens: {h}
Apply 1-ctx-sens: {f}
Apply 0-ctx-sens: {g, m}

"How to find a good program abstraction?"

- Intractably large search space, if not infinite
  - e.g., $(k+1)^{|Func|}$ difference abstractions for context sensitivity
- Few solutions: many abstractions too imprecise or costly

A fundamental problem in static analysis
=> Use machine learning to solve this problem

Example 2: Balancing Soundness and Scalability in Unsound Bug-Finders

Infer

```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```

(a memory-leak bug from bluez-5.55)

Example 2: Balancing Soundness and Scalability in Unsound Bug-Finders

Infer

Normal execution (no memory leak)

```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```

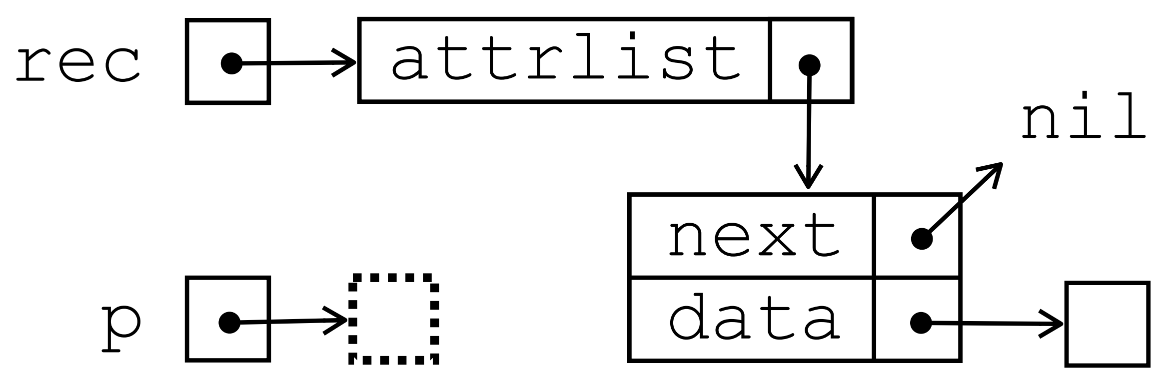
(a memory-leak bug from bluez-5.55)

Example 2: Balancing Soundness and Scalability in Unsound Bug-Finders

Normal execution (no memory leak)

Infer

```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```



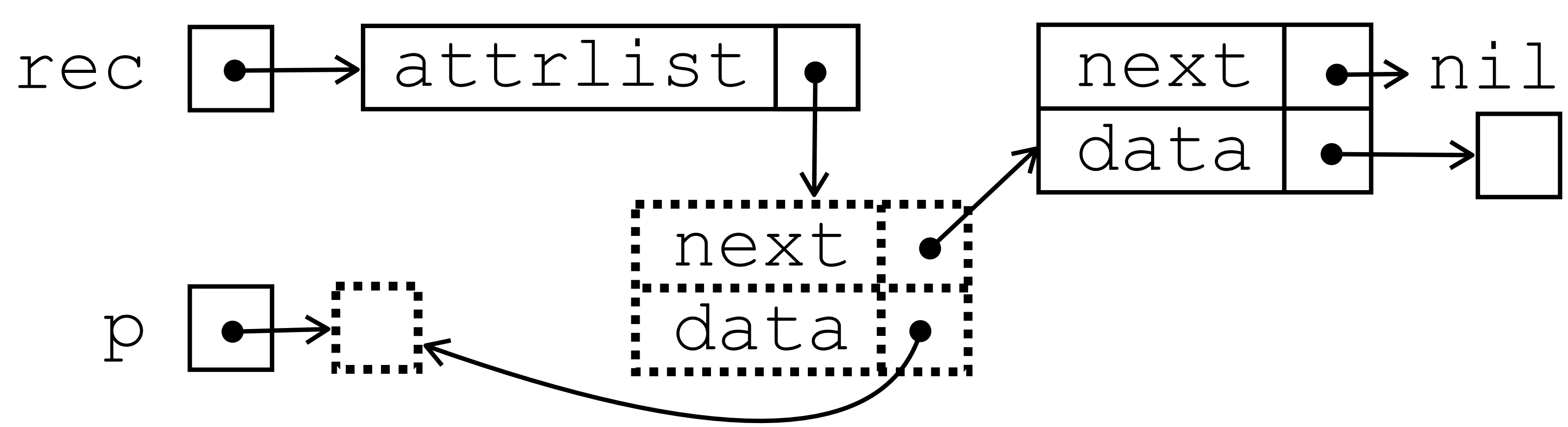
(a memory-leak bug from bluez-5.55)

Example 2: Balancing Soundness and Scalability in Unsound Bug-Finders

Infer

Normal execution (no memory leak)

```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```



(a memory-leak bug from bluez-5.55)

Example 2: Balancing Soundness and Scalability in Unsound Bug-Finders

Buggy execution (memory leak)

Infer

```
p = sdp_seq_alloc();
sdp_attr_replace(rec, p);
```

Example 2: Balancing Soundness and Scalability in Unsound Bug-Finders

Buggy execution (memory leak)

Infer

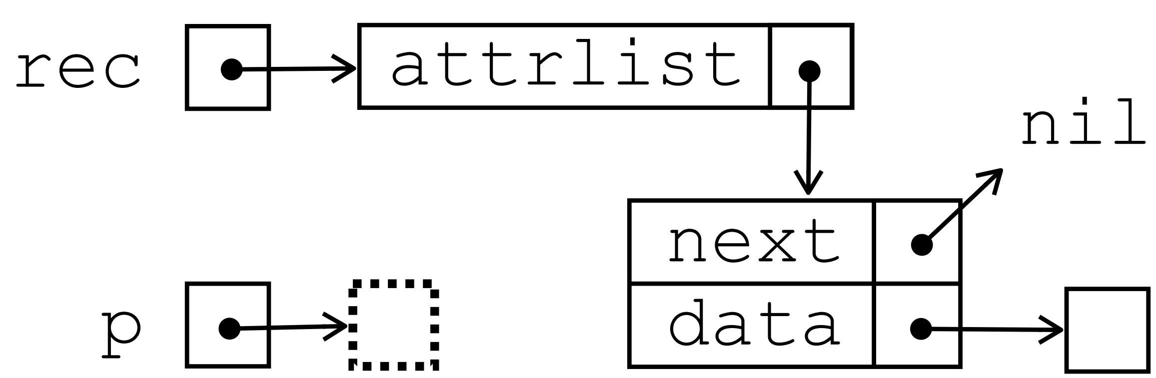
```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```

Example 2: Balancing Soundness and Scalability in Unsound Bug-Finders

Buggy execution (memory leak)

Infer

```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```



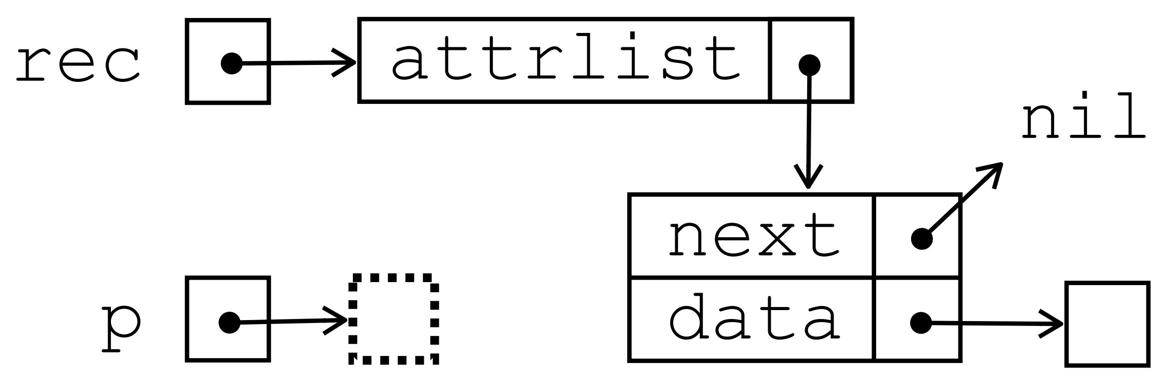
Example 2: Balancing Soundness and Scalability in Unsound Bug-Finders

Buggy execution (memory leak)

Infer

```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```

fails to allocate new memory



Memory Leak: memory unreachable when the enclosing function returns

Challenge: Path Explosion

```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```

Challenge: Path Explosion

involves 6 different function calls, producing more than 1000 execution paths

```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```

Challenge: Path Explosion

involves 6 different function calls, producing more than 1000 execution paths

```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```

involves 8 different function calls, producing more than 2000 execution paths

Challenge: Path Explosion

involves 6 different function calls, producing more than 1000 execution paths

```
7: p = sdp_seq_alloc();
8: sdp_attr_replace(rec, p);
```

involves 8 different function calls, producing more than 2000 execution paths

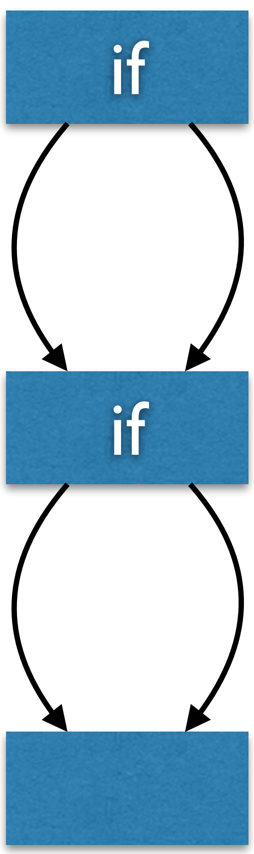
- Path sensitivity is essential for precise/explainable bug-finding
- Analyzing all of them separately does not scale

State-Selection Heuristic

- Static bug-finders like Infer use a *state-selection heuristic* to maintain only a small number (K) of states at a time

State-Selection Heuristic

- Static bug-finders like Infer use a *state-selection heuristic* to maintain only a small number (K) of states at a time

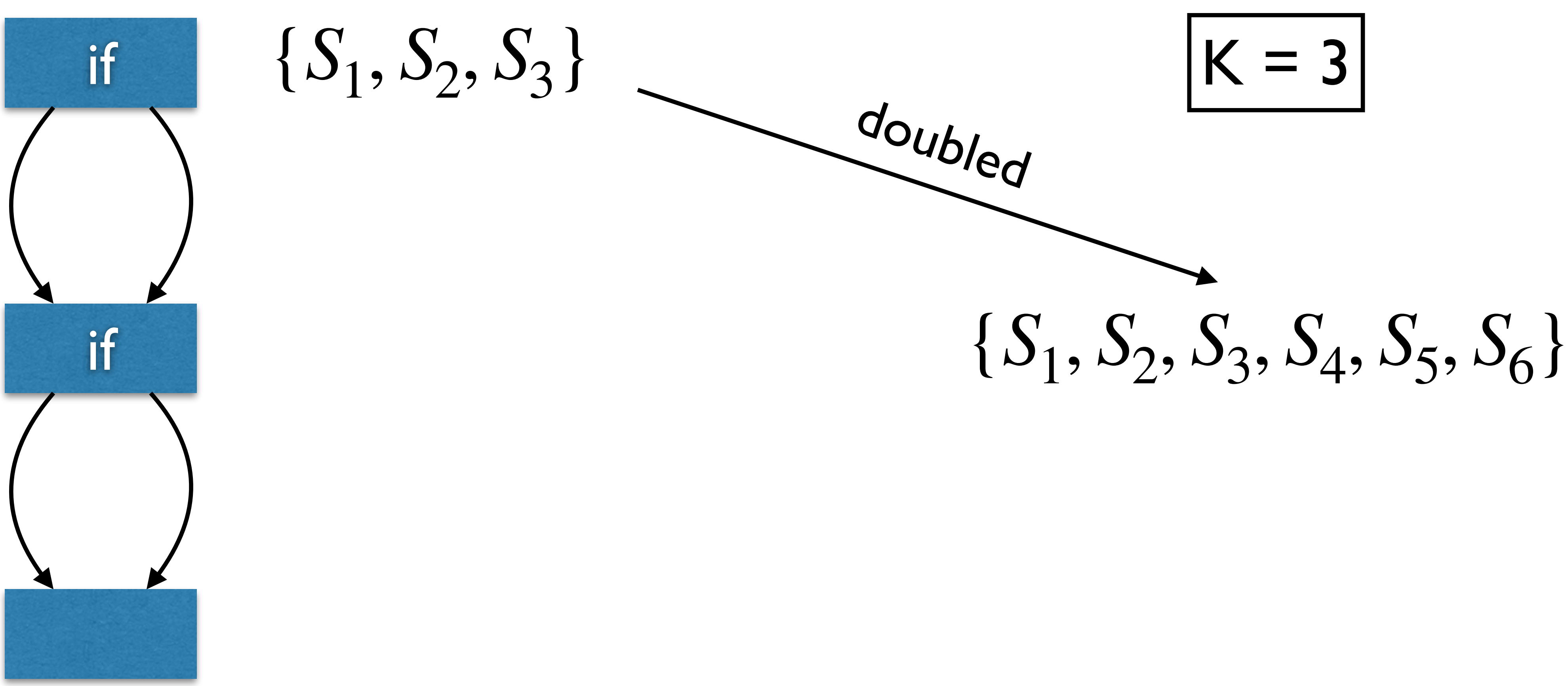


$\{S_1, S_2, S_3\}$

K = 3

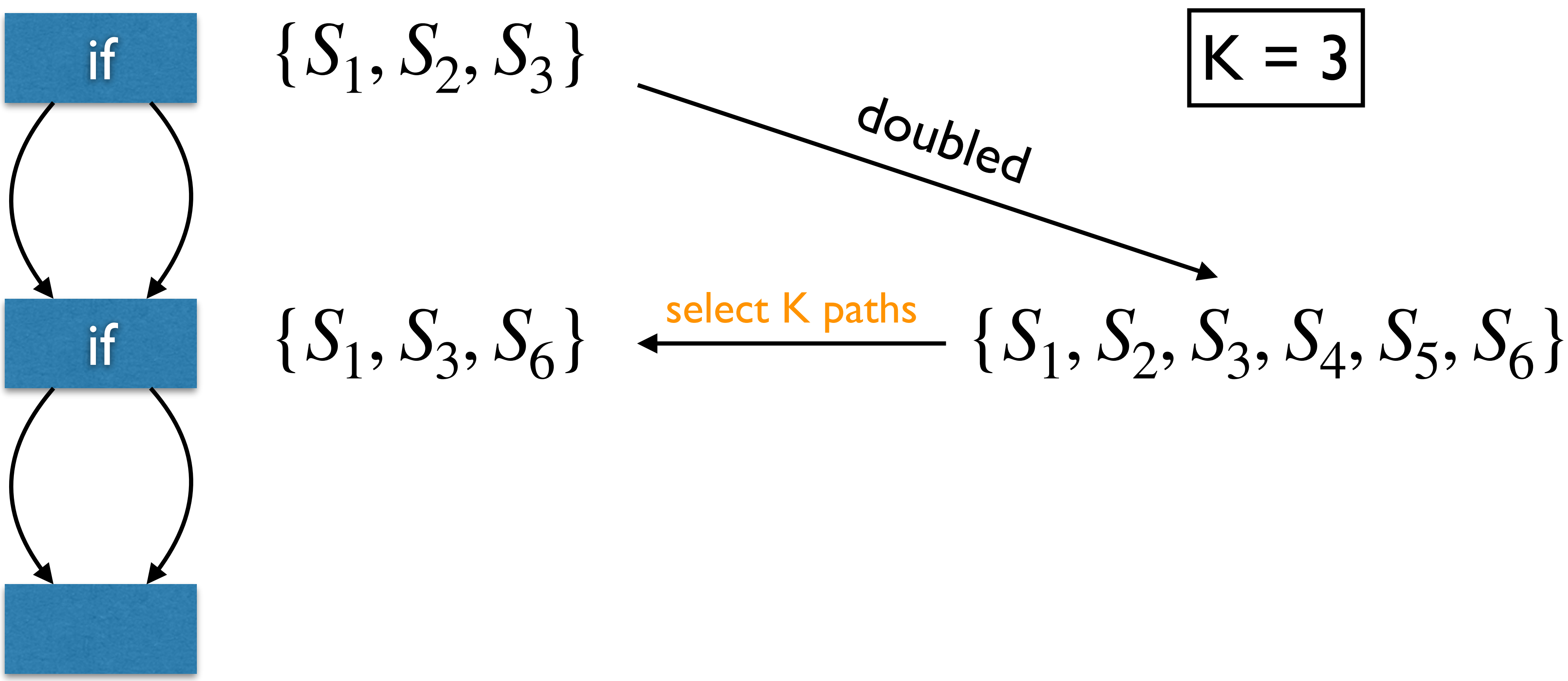
State-Selection Heuristic

- Static bug-finders like Infer use a *state-selection heuristic* to maintain only a small number (K) of states at a time



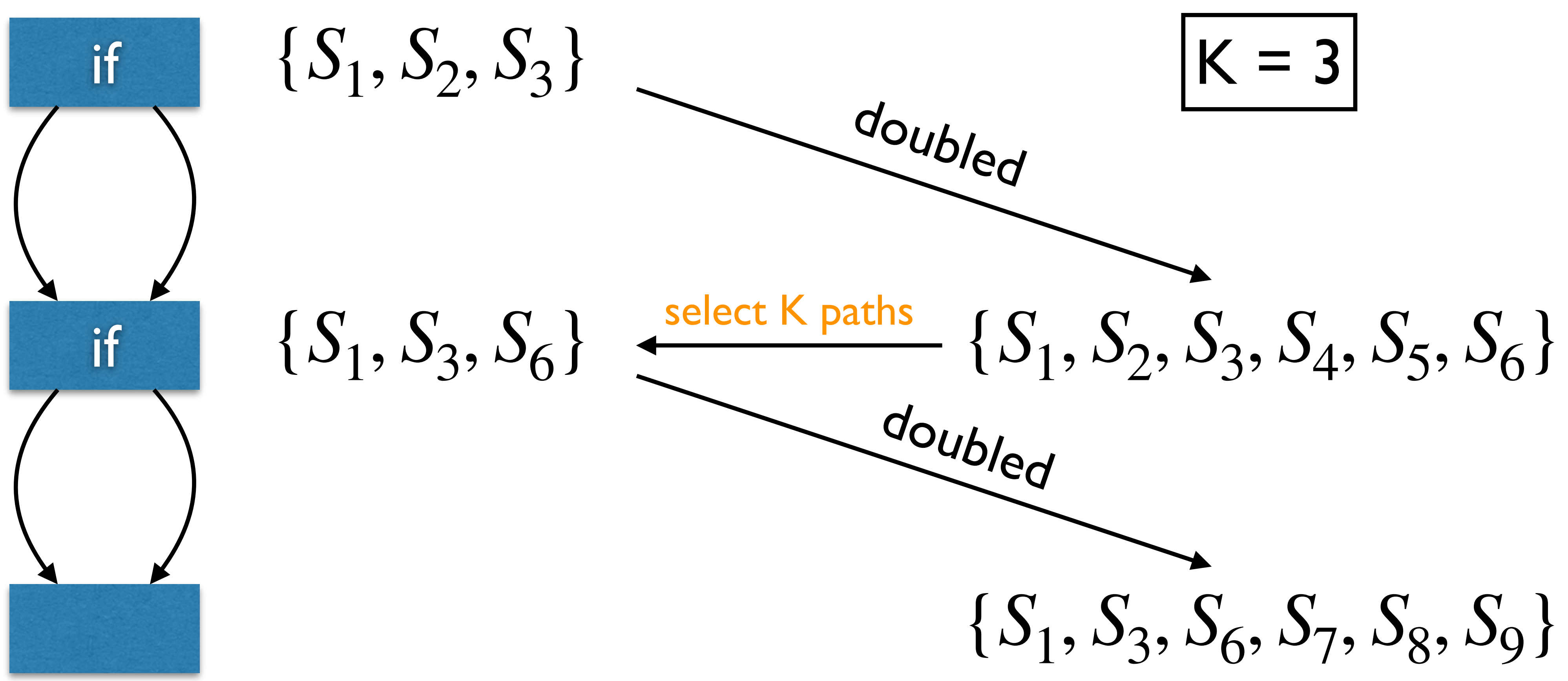
State-Selection Heuristic

- Static bug-finders like Infer use a *state-selection heuristic* to maintain only a small number (K) of states at a time



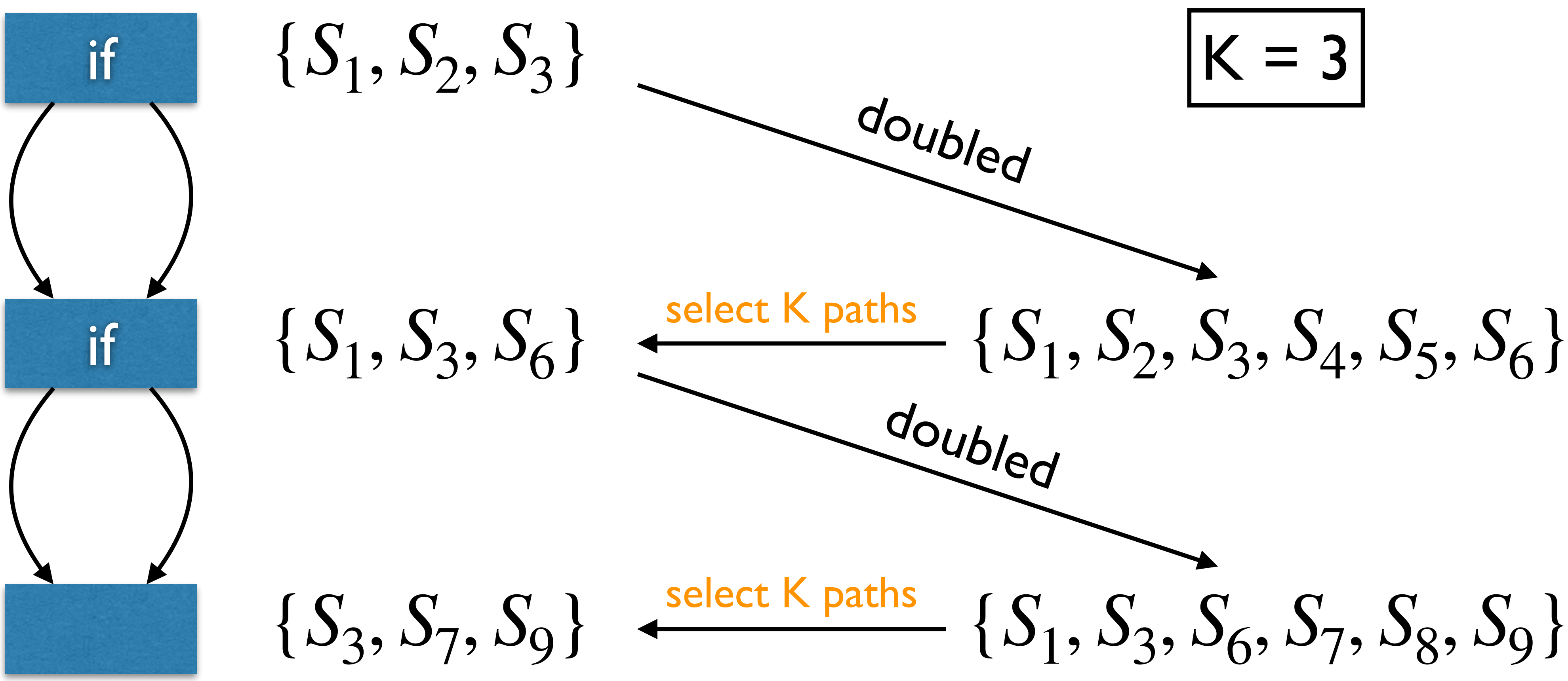
State-Selection Heuristic

- Static bug-finders like Infer use a *state-selection heuristic* to maintain only a small number (K) of states at a time



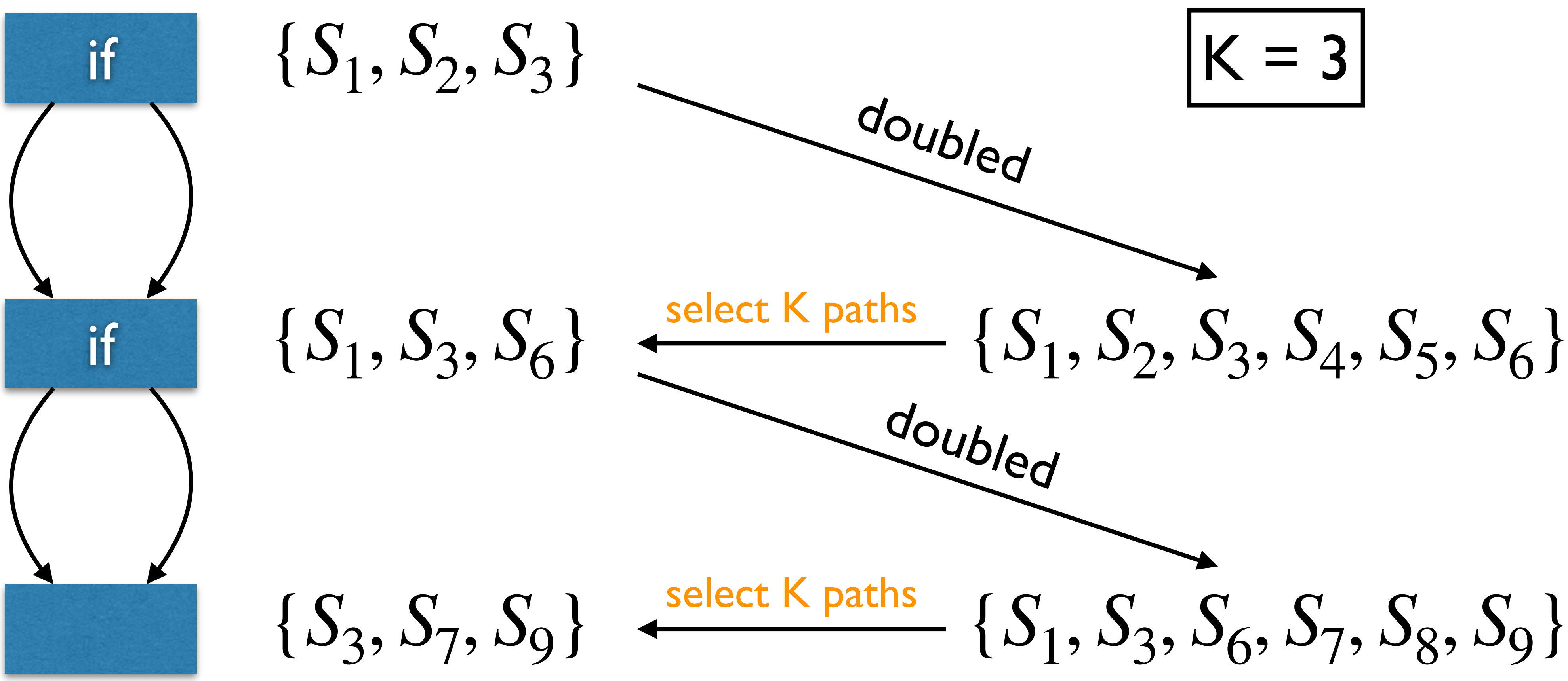
State-Selection Heuristic

- Static bug-finders like Infer use a *state-selection heuristic* to maintain only a small number (K) of states at a time



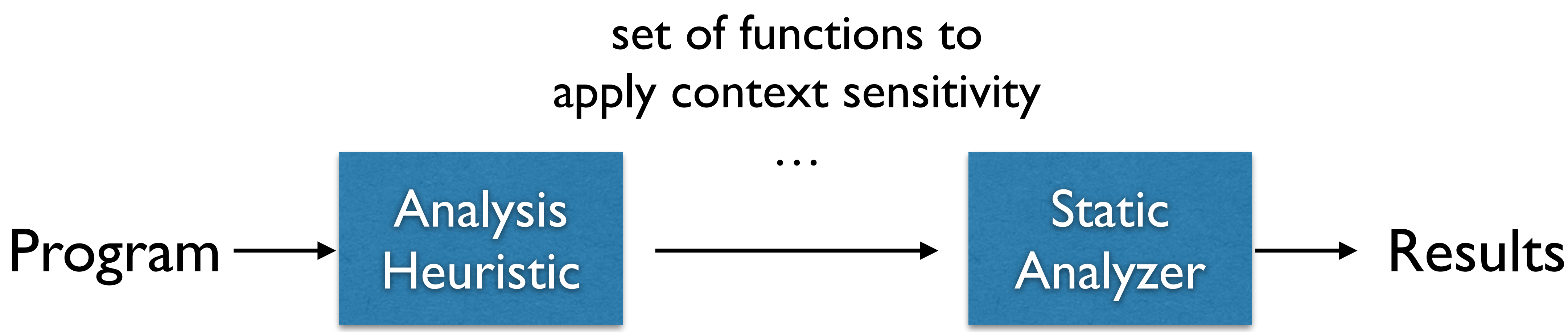
State-Selection Heuristic

- Static bug-finders like Infer use a *state-selection heuristic* to maintain only a small number (K) of states at a time



Our method: Using machine learning to select "promising" states

Our Data-Driven Approach



Our Data-Driven Approach

set of functions to
apply context sensitivity

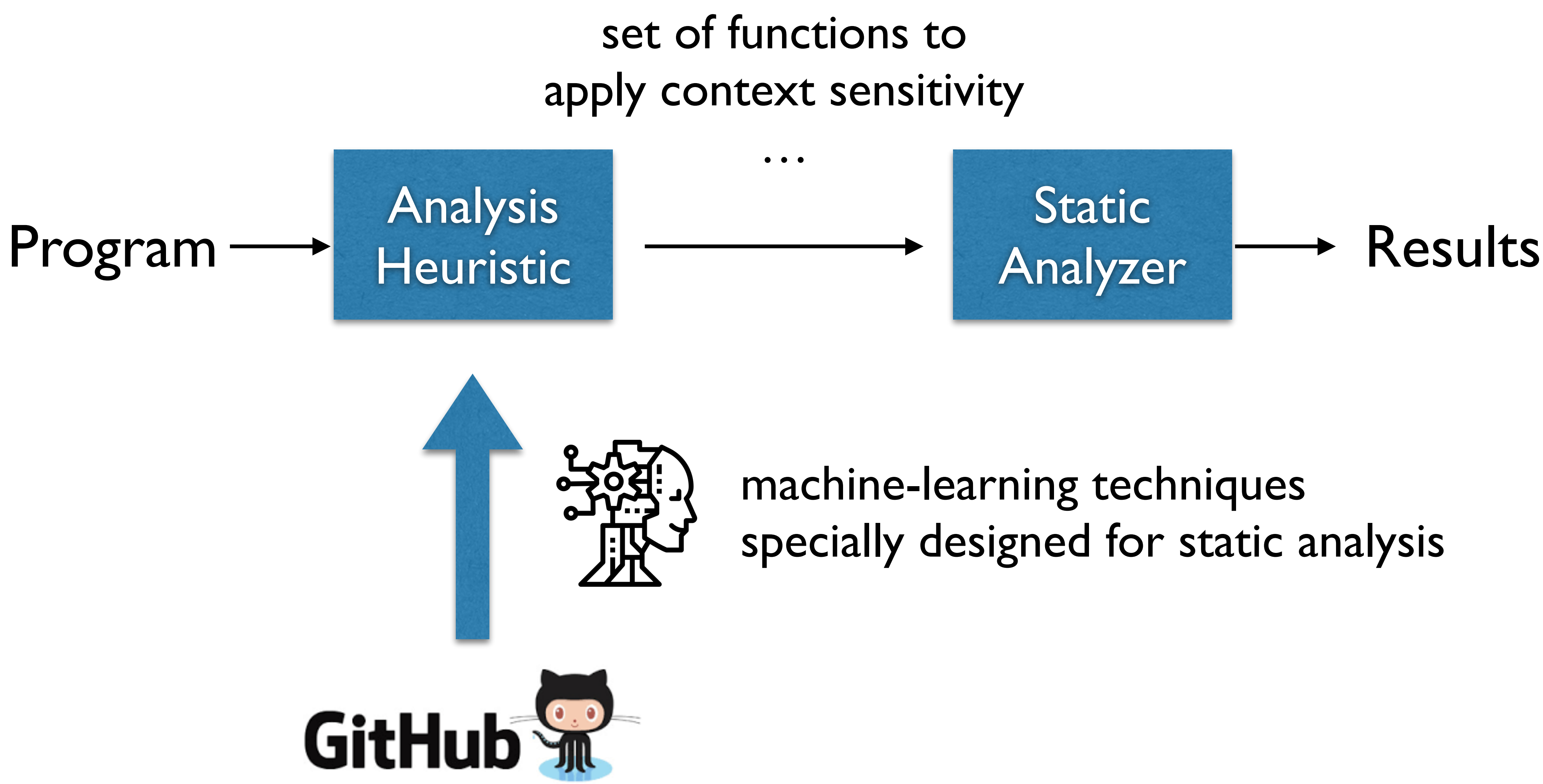
Program → Analysis Heuristic → … → Static Analyzer → Results

Traditionally, analysis heuristics developed manually by human experts:

… PLDI'14 | POPL'17 | PLDI'17 | OOPSLA'18 | FSE'18 | OOPSLA'19 | OOPSLA'21 …

=> nontrivial, time-consuming, and suboptimal

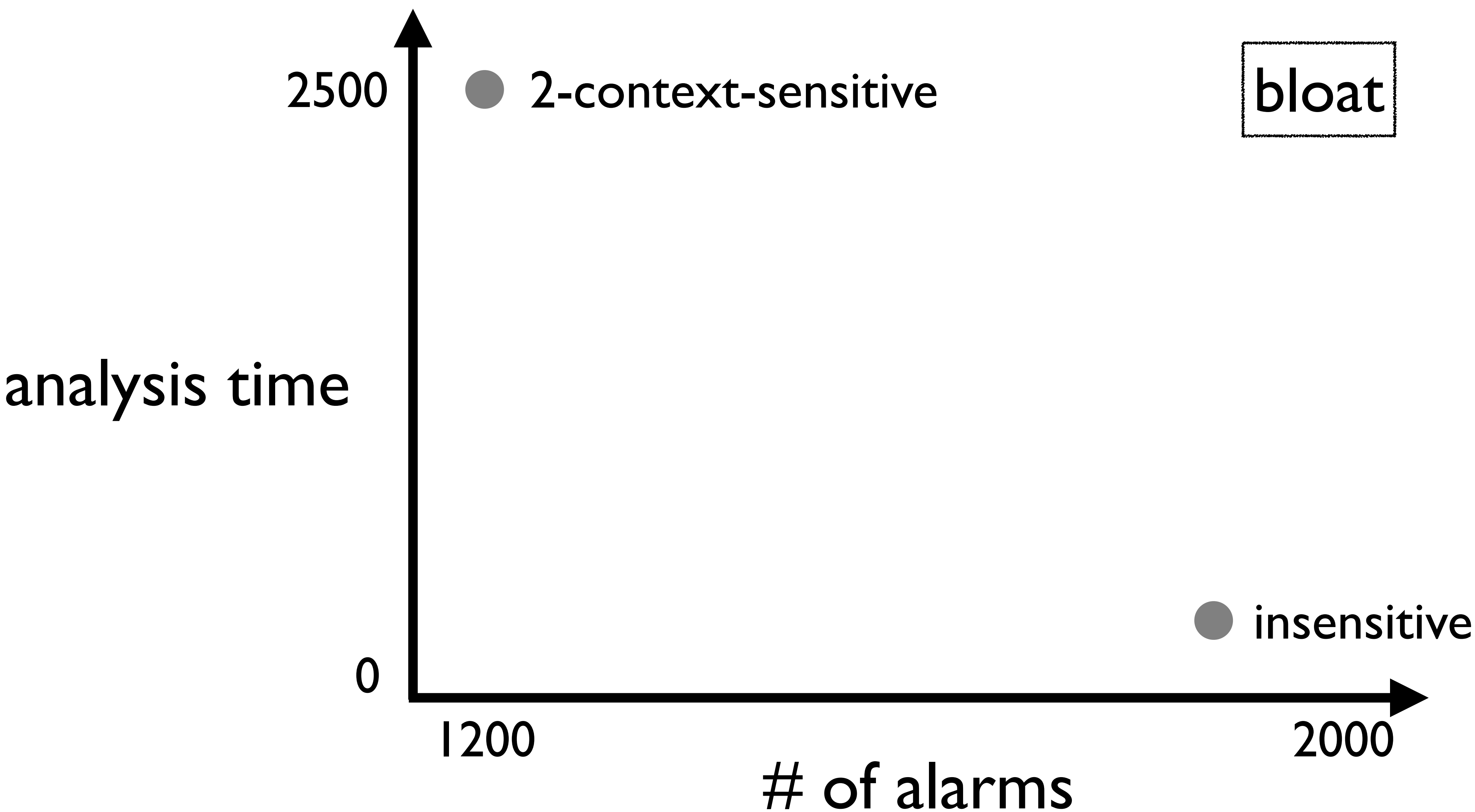
Our Data-Driven Approach



- Automatic: little reliance on analysis designers
- Powerful: machine-tuning outperforms hand-tuning

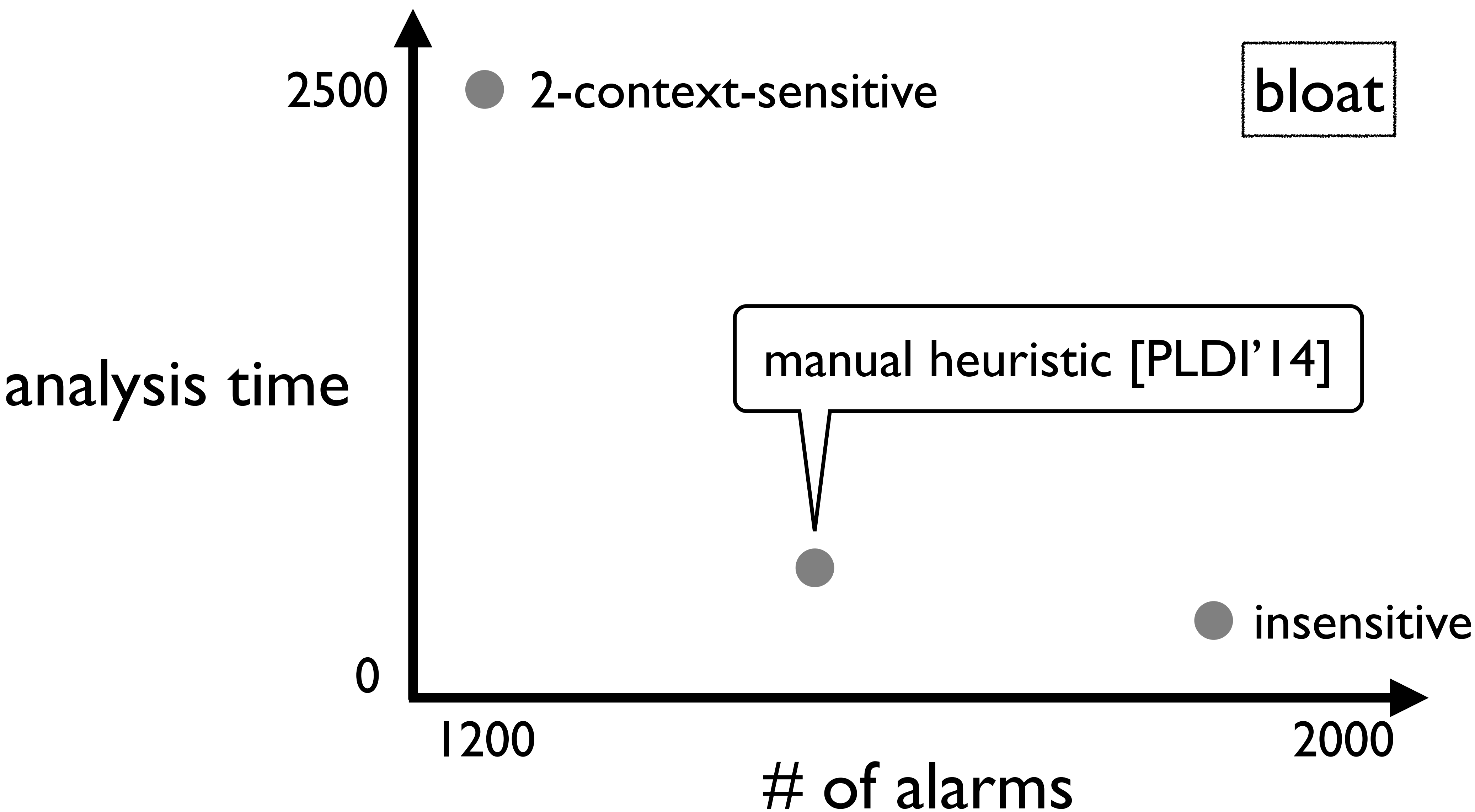
Effectiveness: Context Sensitivity

- Implemented in Doop, a sound pointer analysis for Java
- Trained with 4 and evaluated on 6 programs from DaCapo



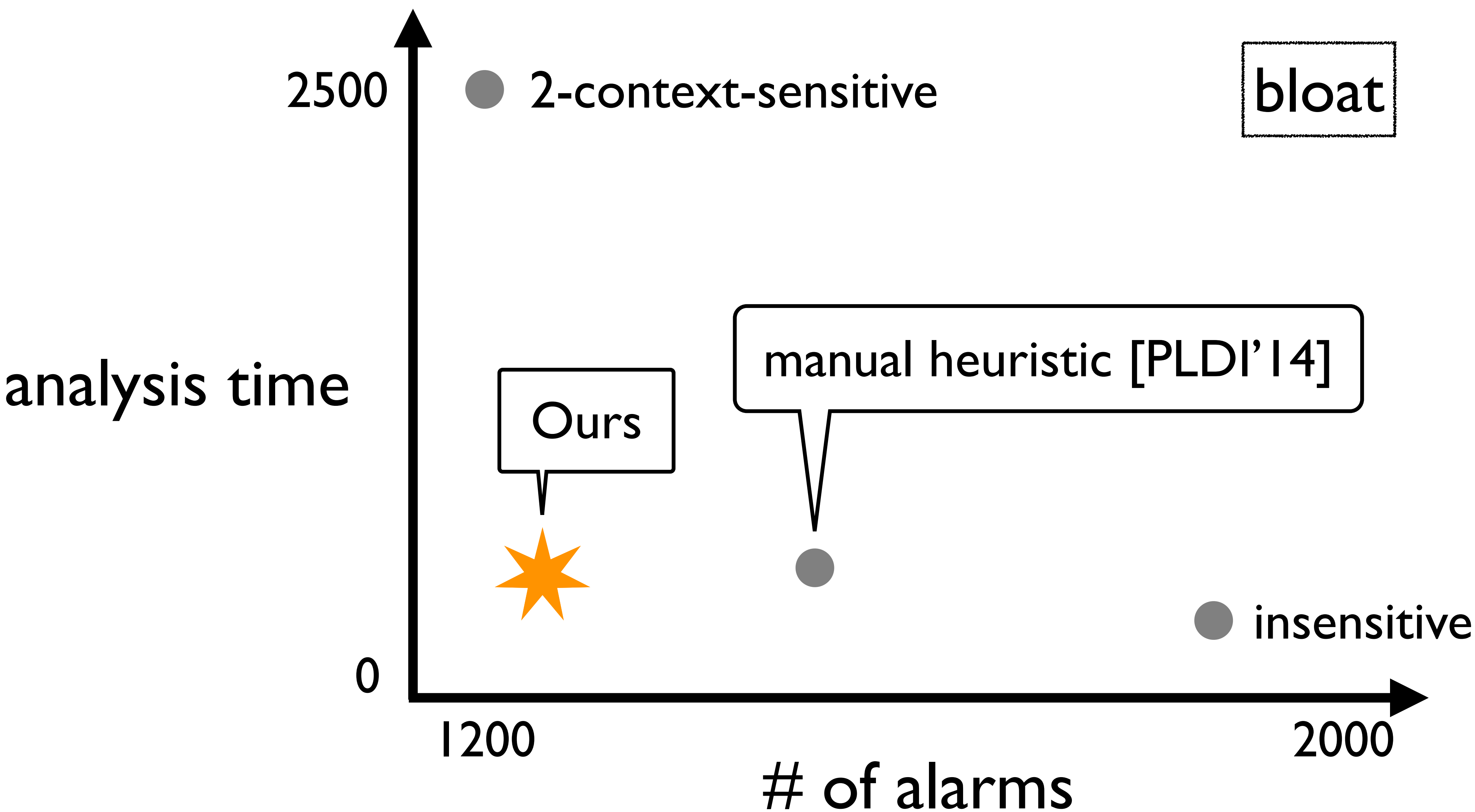
Effectiveness: Context Sensitivity

- Implemented in Doop, a sound pointer analysis for Java
- Trained with 4 and evaluated on 6 programs from DaCapo



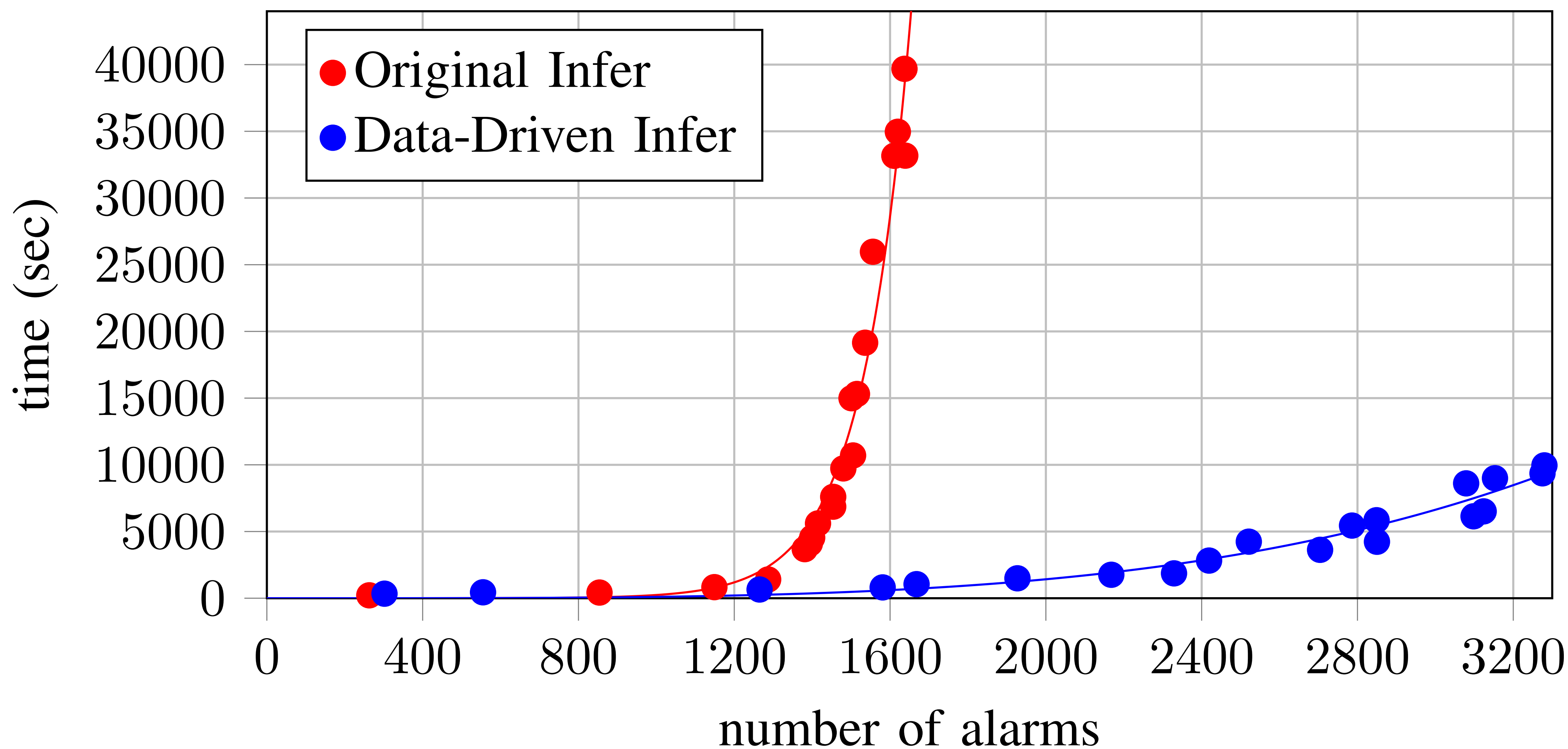
Effectiveness: Context Sensitivity

- Implemented in Doop, a sound pointer analysis for Java
- Trained with 4 and evaluated on 6 programs from DaCapo



Effectiveness: State Selection

- Trained with 70 and evaluated on 15 programs: Infer
  - Original Infer: 1,637 memory-bug alarms in 39,684s (with K = 60)
  - Data-driven Infer: 1,668 memory-bug alarms in 865s (with K = 5)



Remainder of This Talk

ML algorithms developed for static analysis:

- Learning algorithm with linear model [OOPSLA'15]
- Learning algorithm with disjunctive model [OOPSLA'17a]
- Learning algorithm with automated feature generation [OOPSLA'17b]
- Learning algorithm for symbolic execution [ICSE'18, FSE'19, FSE'20]
- Learning algorithm for non-monotone analyses [OOPSLA'18]
- Learning algorithm for resource-aware static analysis [ICSE'19]
- Learning algorithm with feature language [OOPSLA'20]
- Learning algorithm for boosting k-CFA [POPL'22]
- Learning algorithm for boosting static bug-finders [ICSE'23]

Remainder of This Talk

ML algorithms developed for static analysis:

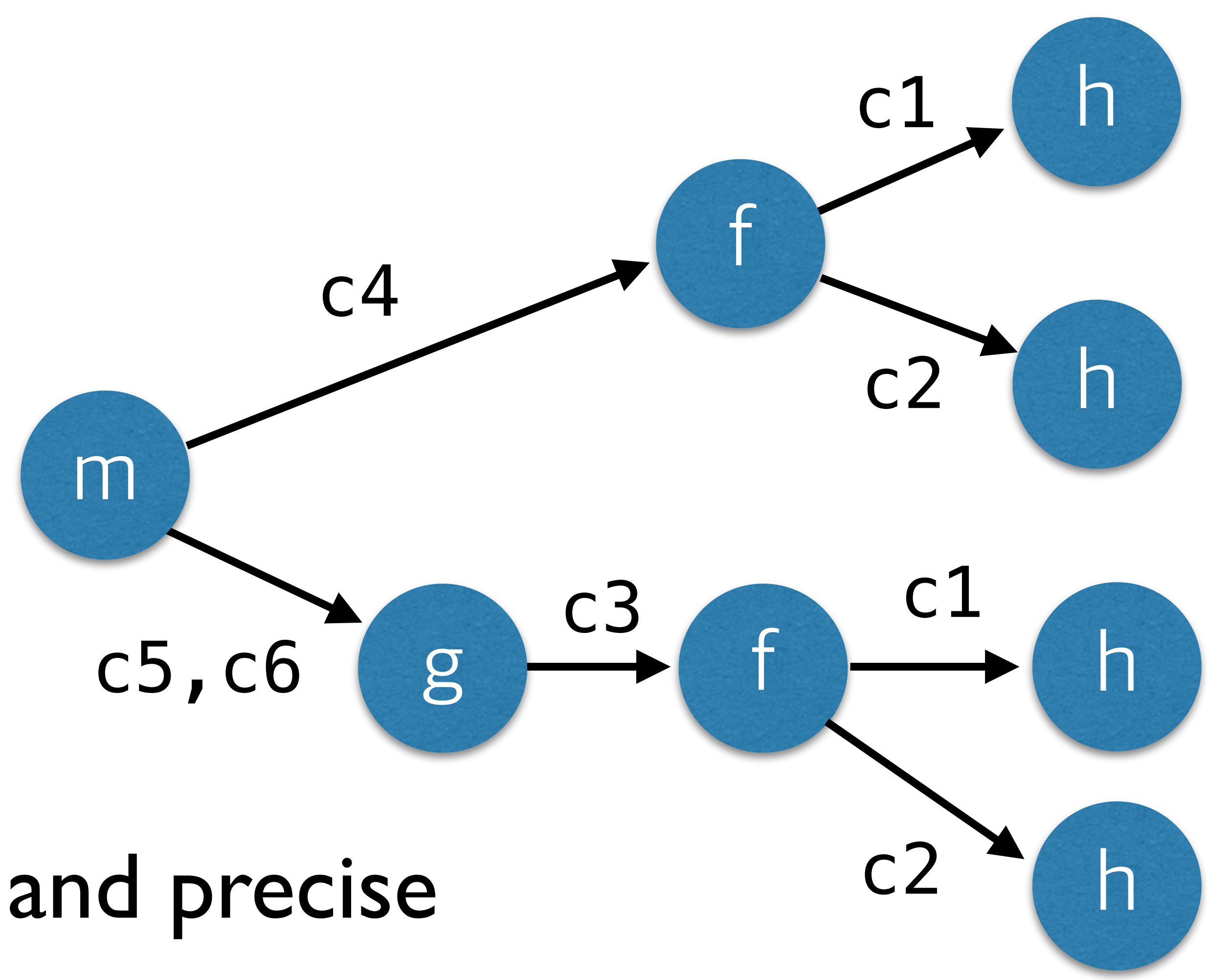
most successful

- Learning algorithm with linear model [OOPSLA'15]
- Learning algorithm with disjunctive model [OOPSLA'17a]
- Learning algorithm with automated feature generation [OOPSLA'17b]
- Learning algorithm for symbolic execution [ICSE'18, FSE'19, FSE'20]
- Learning algorithm for non-monotone analyses [OOPSLA'18]
- Learning algorithm for resource-aware static analysis [ICSE'19]
- Learning algorithm with feature language [OOPSLA'20]
- Learning algorithm for boosting k-CFA [POPL'22]
- Learning algorithm for boosting static bug-finders [ICSE'23]

Selective Context Sensitivity

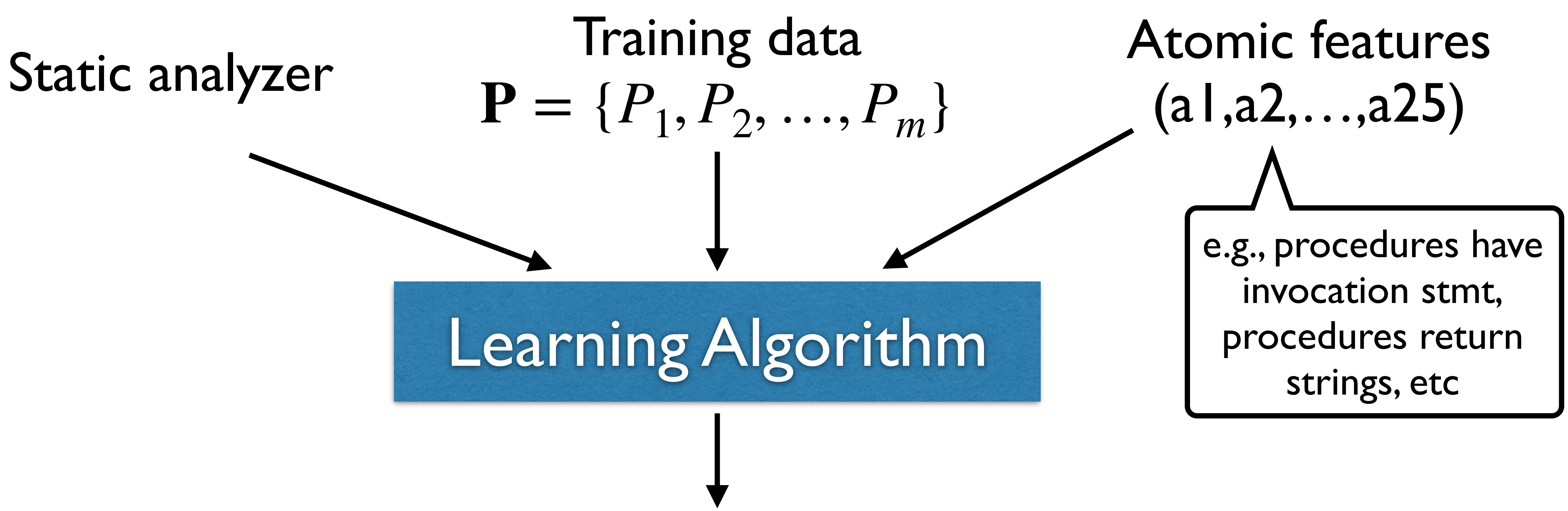
```
    int h(n) {ret n;}

    void f(a) {
c1:   x = h(a);
      assert(x > 0);
c2:   y = h(input());
    }

c3: void g() {f(8);}

    void m() {
c4:   f(4);
c5:   g();
c6:   g();
    }
```

Apply 2-ctx-sens: {h}
Apply 1-ctx-sens: {f}
Apply 0-ctx-sens: {g, m}



cheap and precise

Learning Algorithm Overview



Learned heuristic for applying context-sensitivity:

f2: procedures to apply 2-context-sensitivity

$$1 \wedge \neg 3 \wedge \neg 6 \wedge 8 \wedge \neg 9 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25$$

f1: procedures to apply 1-context-sensitivity

$$(1 \wedge \neg 3 \wedge \neg 4 \wedge \neg 7 \wedge \neg 8 \wedge 6 \wedge \neg 9 \wedge \neg 15 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25) \vee$$
$$(\neg 3 \wedge \neg 4 \wedge \neg 7 \wedge \neg 8 \wedge \neg 9 \wedge 10 \wedge 11 \wedge 12 \wedge 13 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25) \vee$$
$$(\neg 3 \wedge \neg 9 \wedge 13 \wedge 14 \wedge 15 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22 \wedge \neg 23 \wedge \neg 24 \wedge \neg 25) \vee$$
$$(1 \wedge 2 \wedge \neg 3 \wedge 4 \wedge \neg 5 \wedge \neg 6 \wedge \neg 7 \wedge \neg 8 \wedge \neg 9 \wedge \neg 10 \wedge \neg 13 \wedge \neg 15 \wedge \neg 16 \wedge \neg 17 \wedge \neg 18 \wedge \neg 19 \wedge \neg 20 \wedge \neg 21 \wedge \neg 22$$
$$\wedge \neg 23 \wedge \neg 24 \wedge \neg 25)$$

Machine Learning: Three Steps

1. Define a parameterized heuristic $\mathscr{H}_\Pi$:

$$\mathscr{H}_\Pi : Program \rightarrow 2^{Func}$$

2. Define a learning objective as optimization problem:

"Find Π that maximizes analysis performance"

3. Solve the problem via optimization algorithm

1. Parameterized Heuristics

- Atomic features $\mathbb{A} = \{a_1, a_2, \ldots, a_n\}$
  - $a_i : Func \to \{true, false\}$
  - A feature denotes a set of functions:

$$[\![a_i]\!]_P = \{m \in Func \mid a_i(m) = true\}$$

- The heuristic $\mathscr{H}_\Pi$ has k boolean formulas: $\Pi = \langle f_1, f_2 \rangle$

$$f \to true \mid false \mid a_i \in \mathbb{A} \mid \neg f \mid f_1 \wedge f_2 \mid f_1 \vee f_2$$

- Function *m* is assigned context depth *i* if $m \in [\![f_i]\!]$

$$\mathscr{H}_\Pi(P)(m) = \begin{cases} 2 & \textit{if } m \in [\![f_2]\!] \\ 1 & \textit{if } m \in [\![f_1]\!] \\ 0 & \textit{o.w.} \end{cases}$$

Example

```
int h(n) {ret n;}

void f(a) {
  x = h(a);
  assert(x > 0);
  y = h(input());
}

void g() {f(8);}

void m() {
  f(4);
  g();
  g();
}
```

Example

```
int h(n) {ret n;}

void f(a) {
  x = h(a);
  assert(x > 0);
  y = h(input());
}

void g() {f(8);}

void m() {
  f(4);
  g();
  g();
}
```

$$\mathbb{A} = \{a_1, a_2, a_3, a_4, a_5\}$$

$\texttt{h} : \{a_1, a_3, a_5\}$ $\texttt{f} : \{a_3, a_5\}$

$\texttt{g} : \{a_1, a_2, a_3\}$ $\texttt{m} : \{a_2, a_3, a_4\}$

Example

```
int h(n) {ret n;}

void f(a) {
  x = h(a);
  assert(x > 0);
  y = h(input());
}

void g() {f(8);}

void m() {
  f(4);
  g();
  g();
}
```

$$\mathbb{A} = \{a_1, a_2, a_3, a_4, a_5\}$$

$\mathtt{h} : \{a_1, a_3, a_5\}$ $\quad$ $\mathtt{f} : \{a_3, a_5\}$

$\mathtt{g} : \{a_1, a_2, a_3\}$ $\quad$ $\mathtt{m} : \{a_2, a_3, a_4\}$

Heuristic $\mathscr{H}_{\langle f_1, f_2 \rangle}$ with

$$f_1 = \neg a_4 \wedge a_5, \quad f_2 = (a_1 \wedge a_5) \vee (a_2 \wedge \neg a_3)$$

$$(\llbracket f_1 \rrbracket = \{\mathtt{f}, \mathtt{h}\}, \quad \llbracket f_2 \rrbracket = \{\mathtt{h}\})$$

produces the abstraction:

$$\{\mathtt{h} \mapsto 2,\ f \mapsto 1,\ g \mapsto 0,\ m \mapsto 0\}$$

2. Optimization Problem

Find Π that minimizes $\sum_{P \in \mathbf{P}} \mathrm{cost}(F_P(\mathscr{H}_\Pi(P)))$

while ensuring a user-provided precision constraint.

2. Optimization Problem

Find Π that minimizes $\sum_{P \in \mathbf{P}} \text{cost}(F_P(\mathscr{H}_\Pi(P)))$

while ensuring a user-provided precision constraint.

E.g., "maintain 90% precision of 2-CFA"

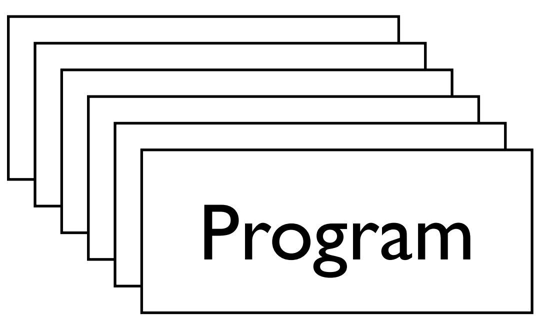
$$\frac{\sum_{P \in \mathbf{P}} |\text{proved}(F_P(\mathscr{H}_\Pi(P)))|}{\sum_{P \in \mathbf{P}} |\text{proved}(F_P(\lambda m.2))|} \geq 0.9$$

# of assertions proved by the current abstraction

# of assertions proved by the most precise abstraction (2-CFA)

3. Optimization Algorithm

- Basic method: blackbox exhaustive search



training data

$$\Pi^1 = \langle f_1^1, f_2^1 \rangle$$

$$\Pi^2 = \langle f_1^2, f_2^2 \rangle$$

$$\Pi^3 = \langle f_1^3, f_2^3 \rangle$$

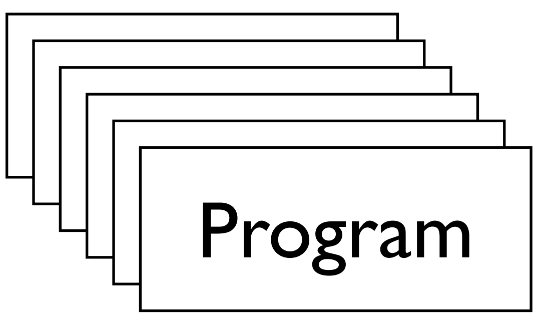
$$\Pi^4 = \langle f_1^4, f_2^4 \rangle$$

$$\vdots$$

1. (Randomly) Generate solution candidates

3. Optimization Algorithm

- Basic method: blackbox exhaustive search



training data

$$\Pi^1 = \langle f_1^1, f_2^1 \rangle \qquad \sum_{P \in \mathbf{P}} \mathrm{cost}(F_P(\mathscr{H}_\Pi(P))) = 100$$

$$\Pi^2 = \langle f_1^2, f_2^2 \rangle \qquad \sum_{P \in \mathbf{P}} \mathrm{cost}(F_P(\mathscr{H}_\Pi(P))) = 130$$

$$\Pi^3 = \langle f_1^3, f_2^3 \rangle \qquad \sum_{P \in \mathbf{P}} \mathrm{cost}(F_P(\mathscr{H}_\Pi(P))) = 80$$

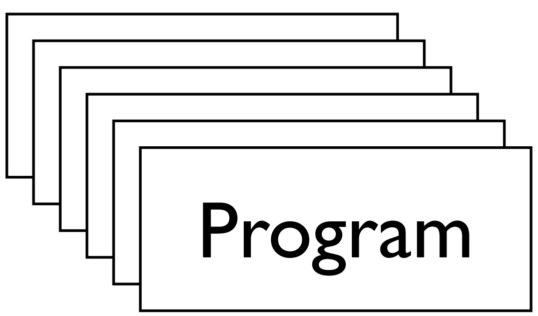
$$\Pi^4 = \langle f_1^4, f_2^4 \rangle \qquad \sum_{P \in \mathbf{P}} \mathrm{cost}(F_P(\mathscr{H}_\Pi(P))) = 190$$

$$\vdots$$

2. Evaluate the objective function

3. Optimization Algorithm

- Basic method: blackbox exhaustive search



training data

$$\Pi^1 = \langle f_1^1, f_2^1 \rangle \qquad \sum_{P \in \mathbf{P}} \text{cost}(F_P(\mathscr{H}_\Pi(P))) = 100$$

$$\Pi^2 = \langle f_1^2, f_2^2 \rangle \qquad \sum_{P \in \mathbf{P}} \text{cost}(F_P(\mathscr{H}_\Pi(P))) = 130$$

$$\Pi^3 = \langle f_1^3, f_2^3 \rangle \qquad \sum_{P \in \mathbf{P}} \text{cost}(F_P(\mathscr{H}_\Pi(P))) = 80$$

$$\Pi^4 = \langle f_1^4, f_2^4 \rangle \qquad \sum_{P \in \mathbf{P}} \text{cost}(F_P(\mathscr{H}_\Pi(P))) = 190$$

$$\vdots$$

3. Choose the parameter with minimum cost

3. Optimization Algorithm

We learn each formula via greedy refinement

1. Initialize f to the most general formula in DNF:
   $f = a_1 \vee \neg a_1 \vee a_2 \vee \neg a_2 \vee \ldots \vee a_n \vee \neg a_n \quad (\equiv true)$
2. Repeat the following (until no refinement is possible)

$$f = c_1 \vee c_2 \vee \ldots \vee c_m$$

   1. Choose the most expensive conjunct, say c_i
   2. Refine the conjunct with some feature a_j:
      $$f = c_1 \vee c_2 \vee \ldots \vee (c_i \wedge a_j) \vee \ldots \vee c_m$$
   3. Check the precision constraint: If not, revert the last change.

(details in paper)

Summary

Data-Driven Static Analysis

- A general framework for generating analysis heuristics:

- The idea is not limited to static analysis: e.g.,
  - Symbolic execution [ICSE'18, FSE'19, FSE'20, ICSE'22]
  - Fuzzing [ISSTA'20, ICSE'23]
- More information available at http://prl.korea.ac.kr

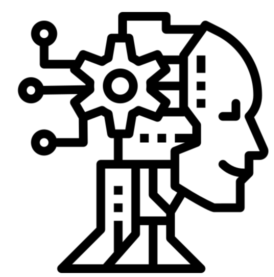
Summary

Data-Driven Static Analysis

- A general framework for generating analysis heuristics:

Static analyzer →

Data (programs) →



→ context-sensitivity heuristics
path-selection heuristics
…

- The idea is not limited to static analysis: e.g.,
  - Symbolic execution [ICSE'18, FSE'19, FSE'20, ICSE'22]
  - Fuzzing [ISSTA'20, ICSE'23]
- More information available at http://prl.korea.ac.kr

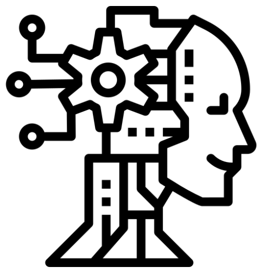
Summary

Data-Driven Static Analysis

- A general framework for generating analysis heuristics:

Static analyzer →

Data (programs) →



→ context-sensitivity heuristics
path-selection heuristics
…

- The idea is not limited to static analysis: e.g.,
  - Symbolic execution [ICSE'18, FSE'19, FSE'20, ICSE'22]
  - Fuzzing [ISSTA'20, ICSE'23]
- More information available at http://prl.korea.ac.kr

Thank you!