Samsung Research Funding &
Incubation Center for Future Technology

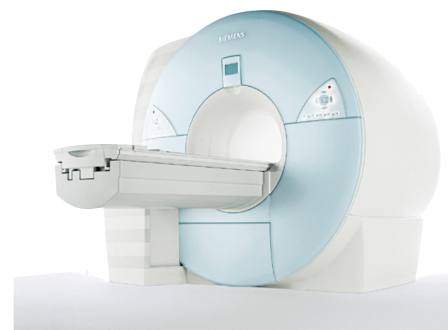
Data-Driven and Focused Program Analysis

Hakjoo Oh
Korea University

15 November 2018

Static Program Analysis

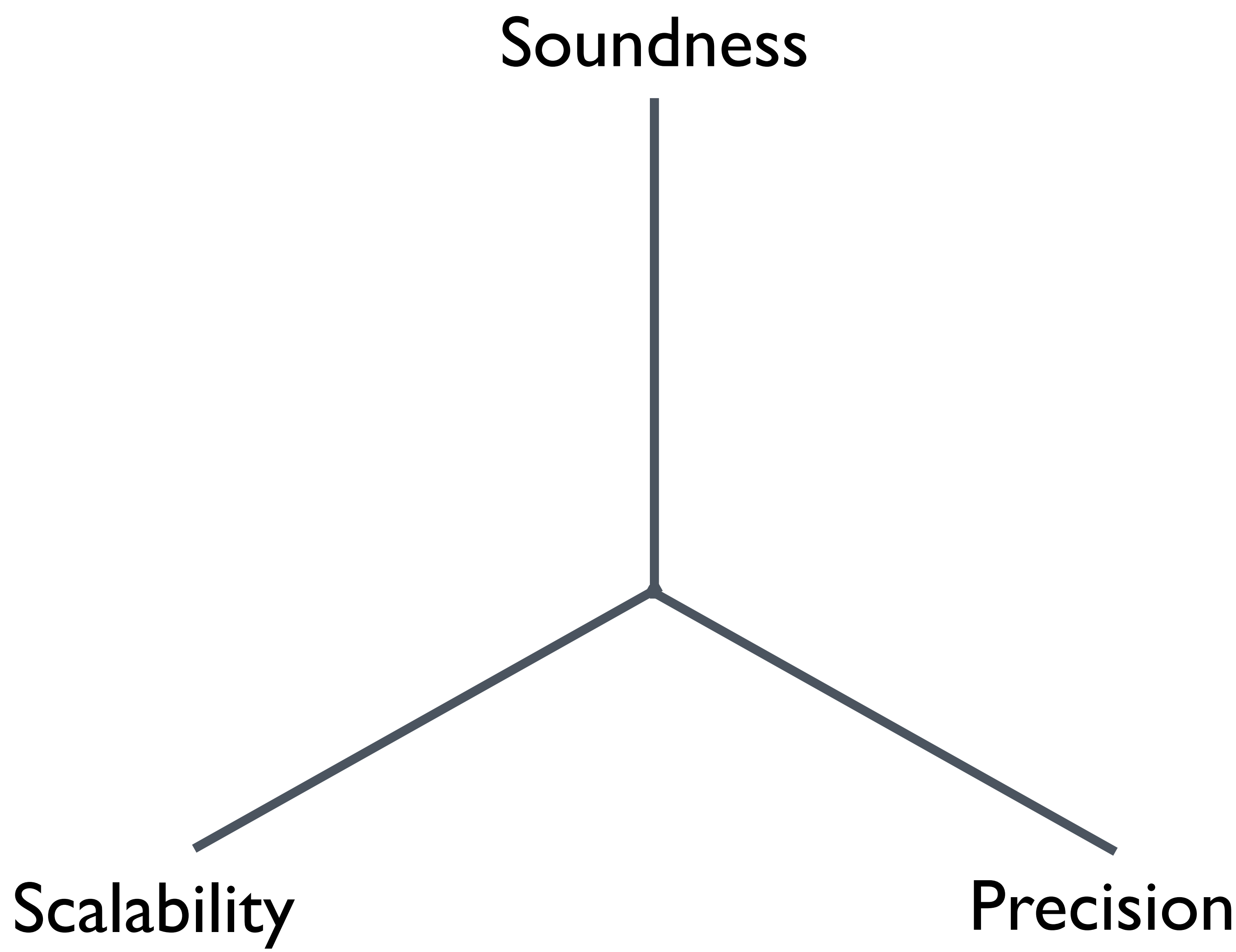
Technology for "software MRI"

Code →  → Safe / Error

- Predict software behavior statically and automatically
  - static: analyzing program text without execution
  - automatic: sw is analyzed by sw ("static analyzer")
- Next-generation software testing technology
  - finding bugs early / full automation / all bugs found
- Being widely used in sw industry

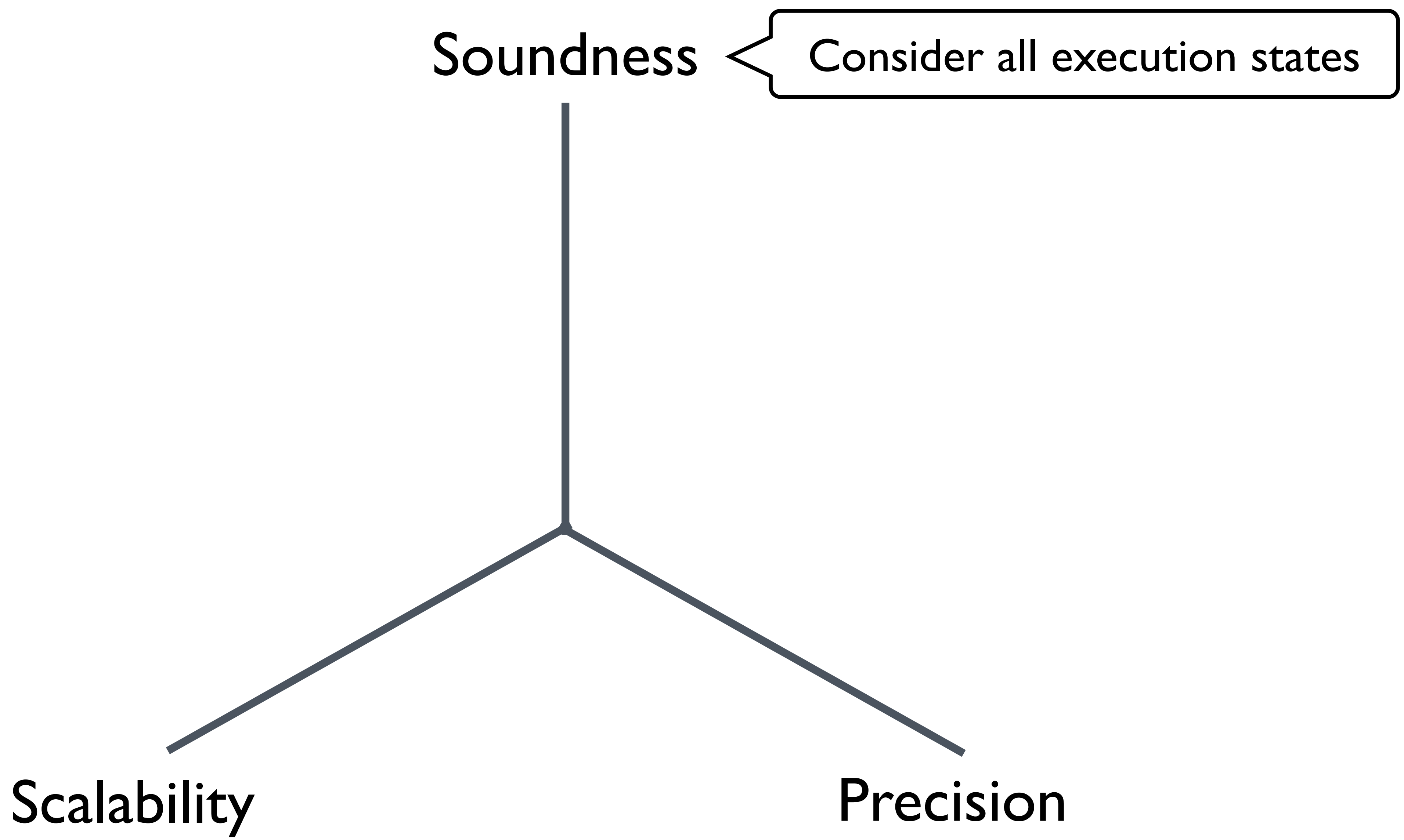
Long-Standing Open Problem

- How to achieve soundness, precision, and scalability at the same time?



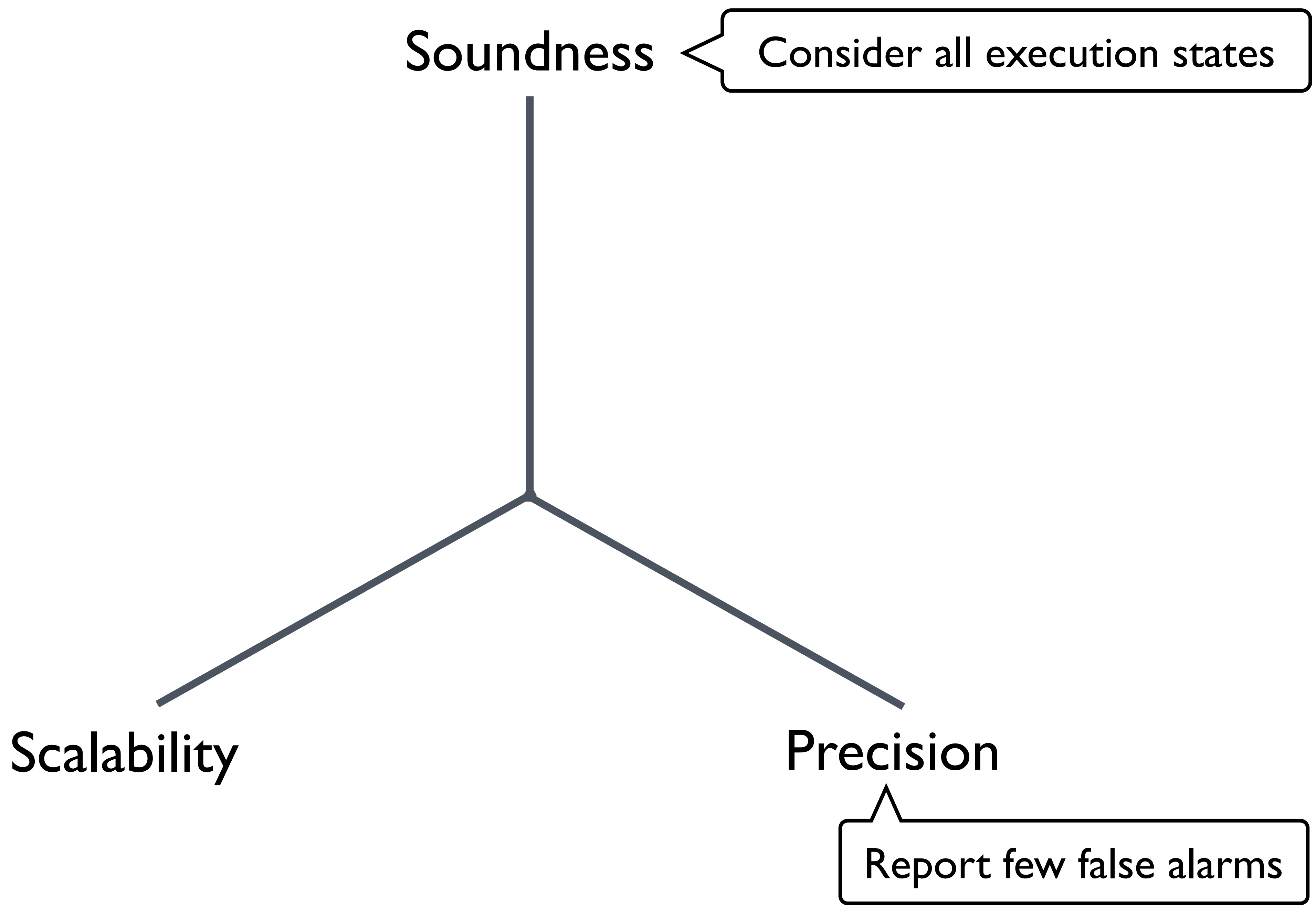
Long-Standing Open Problem

- How to achieve soundness, precision, and scalability at the same time?



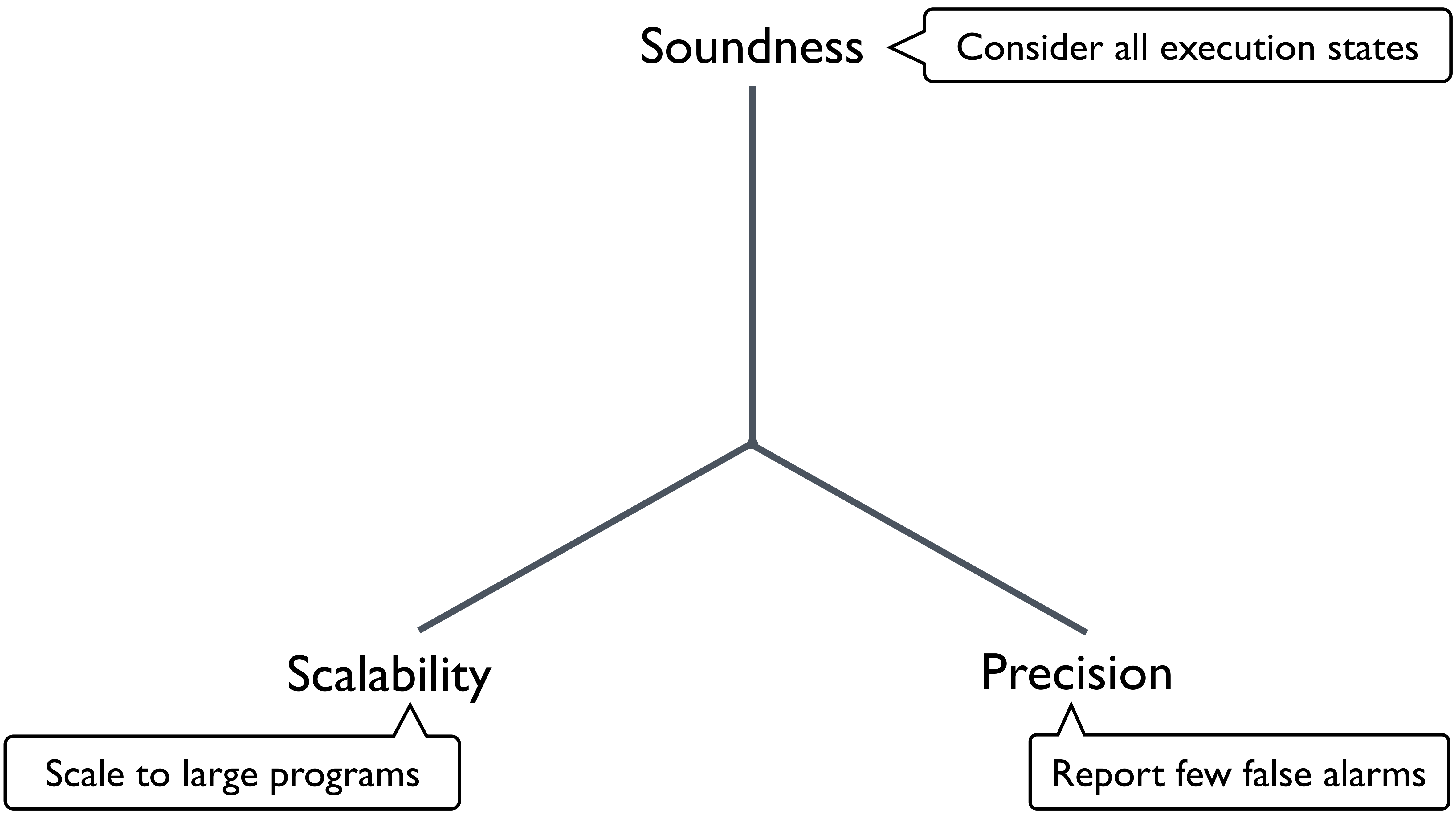
Long-Standing Open Problem

- How to achieve soundness, precision, and scalability at the same time?



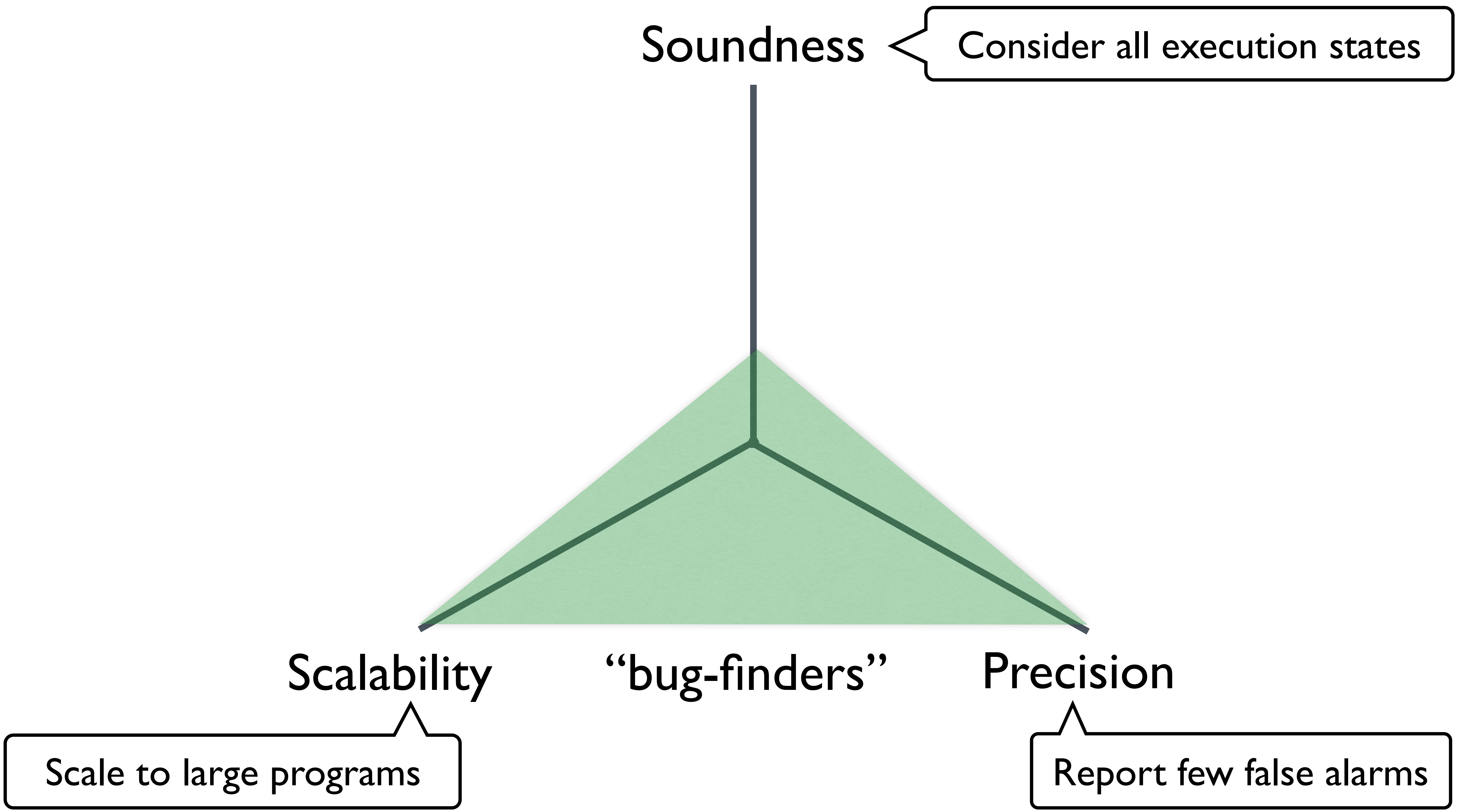
Long-Standing Open Problem

- How to achieve soundness, precision, and scalability at the same time?



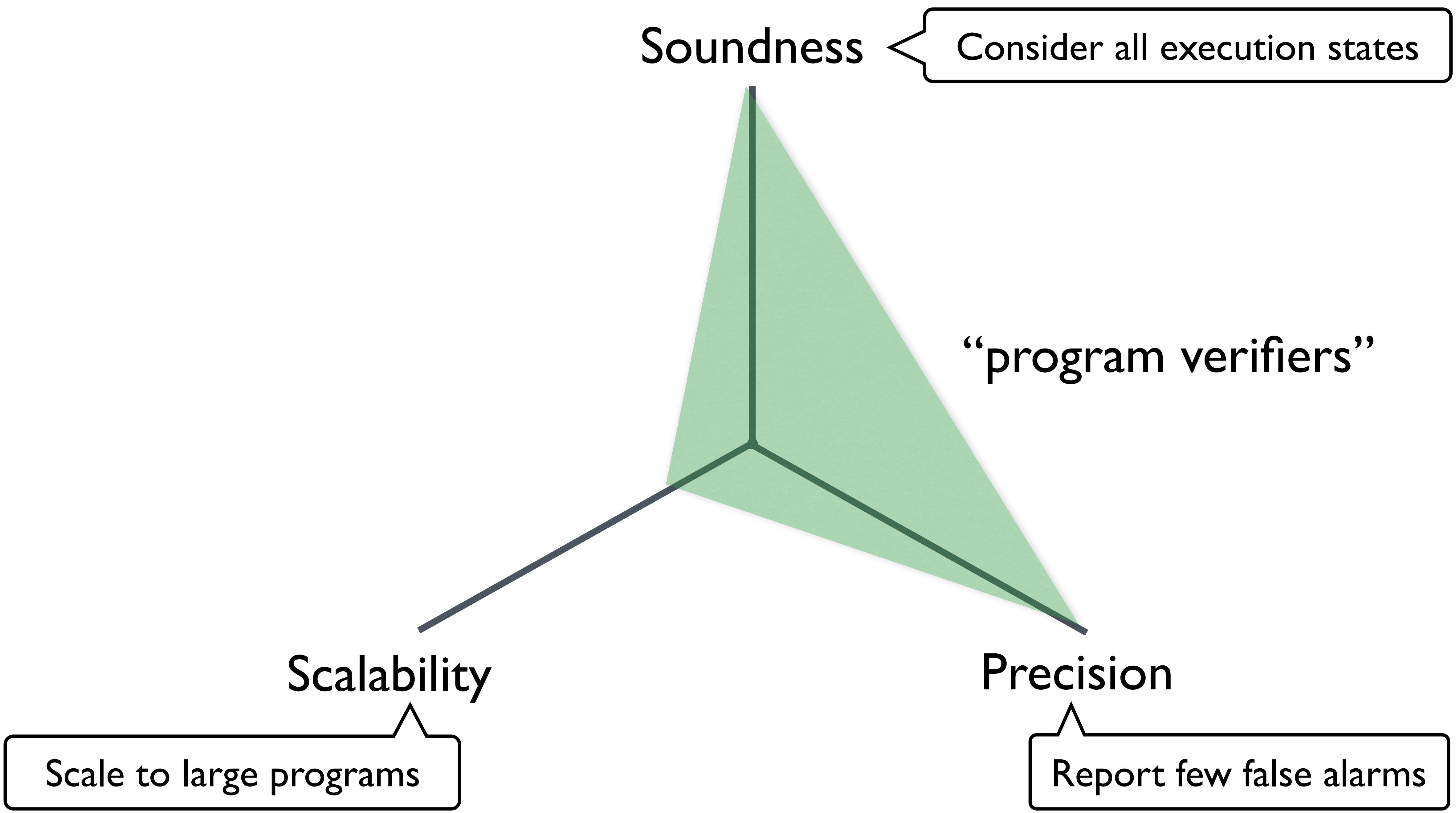
Long-Standing Open Problem

- How to achieve soundness, precision, and scalability at the same time?



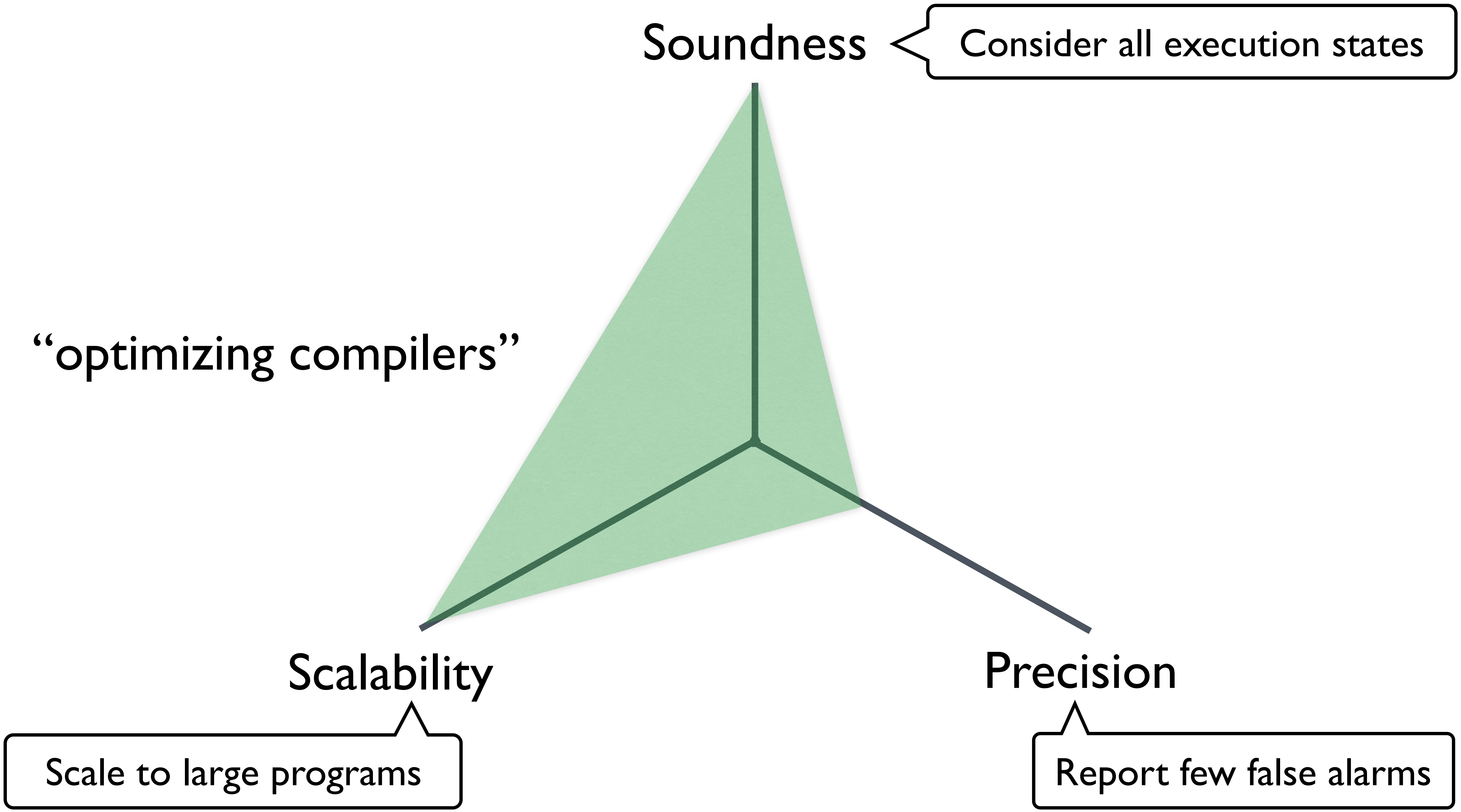
Long-Standing Open Problem

- How to achieve soundness, precision, and scalability at the same time?



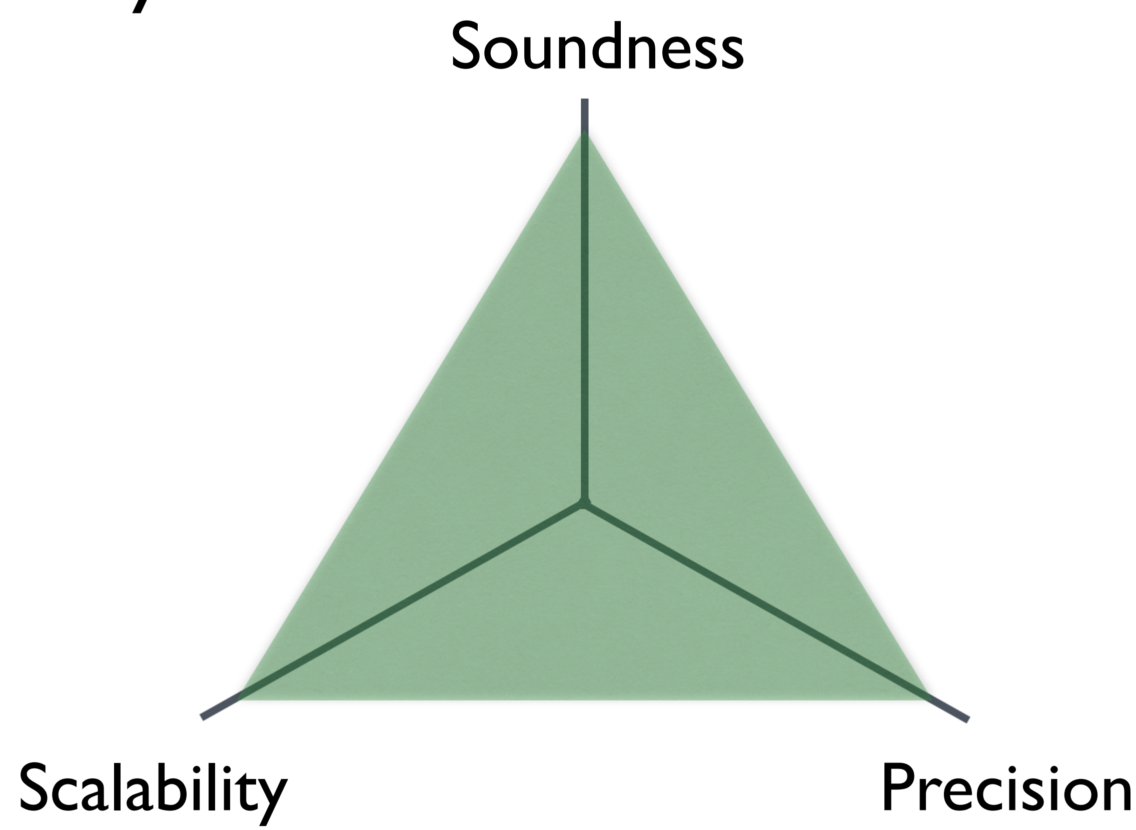
Long-Standing Open Problem

- How to achieve soundness, precision, and scalability at the same time?



Project Goal

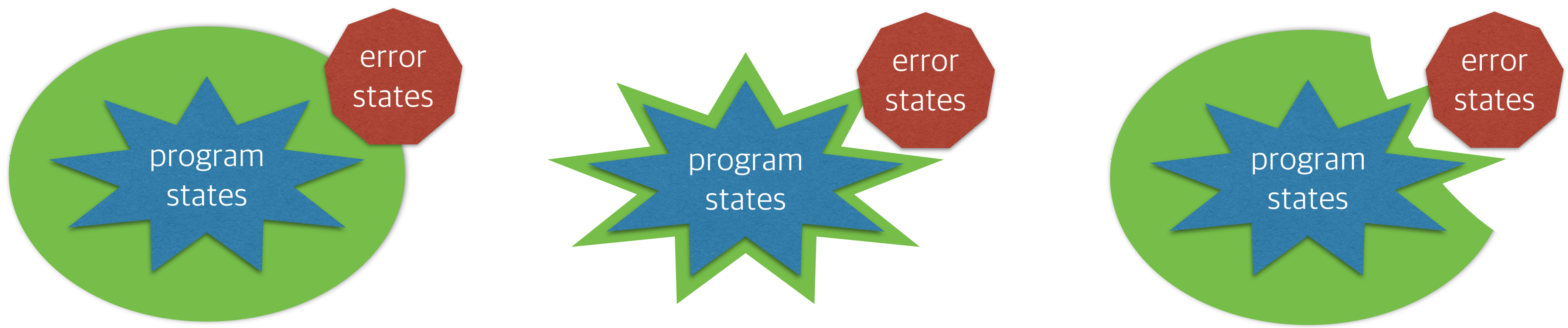
- General technology for achieving soundness, precision, and scalability:



- Prove generality & effectiveness with three analyses:
  - numeric analysis scalable to 1M in 1hour
  - pointer analysis scalable to 500K in 1hour
  - symbolic analysis scalable to 300K in 1hour

Approach Overview

- Selective application of high precision (and soundness):



cheap but imprecise | precise but expensive | cheap and precise

- Data-driven, automatic generation of selection heuristics:

GitHub → machine learning for program analysis → Heuristics for deciding when to apply high precision

Example: Context-Sensitivity

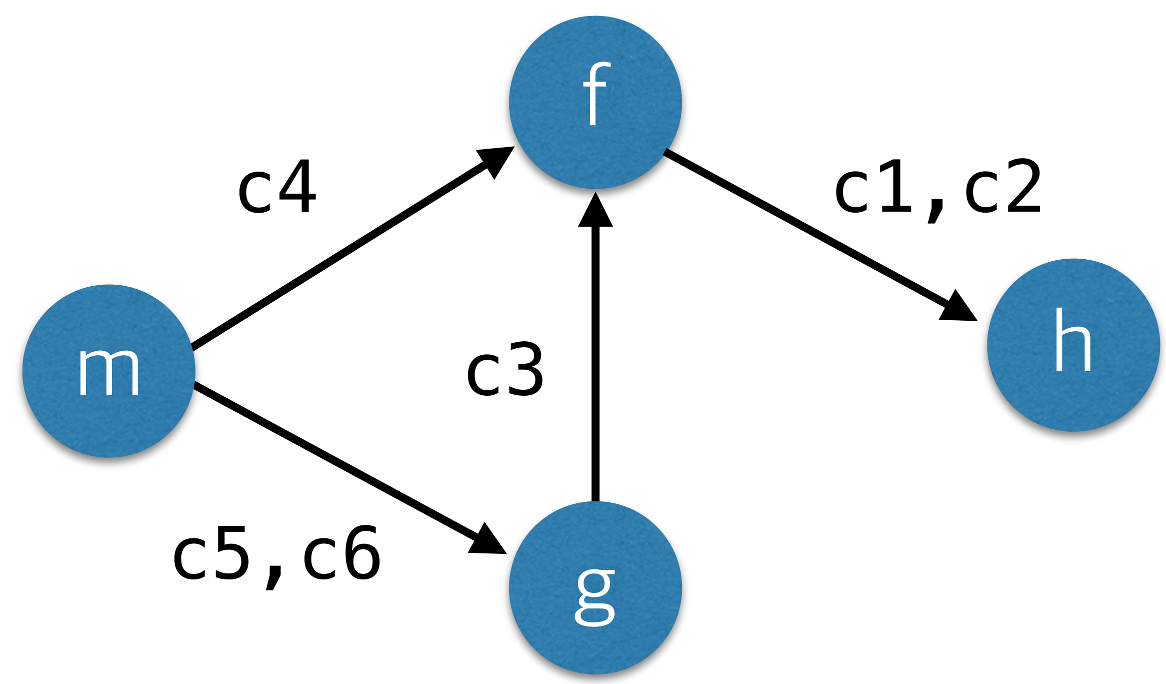
```
    int h(n) {ret n;}

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

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

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

holds always

Context-Insensitive Analysis

- Merge calling contexts into single abstract context

```
    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();
    }
```

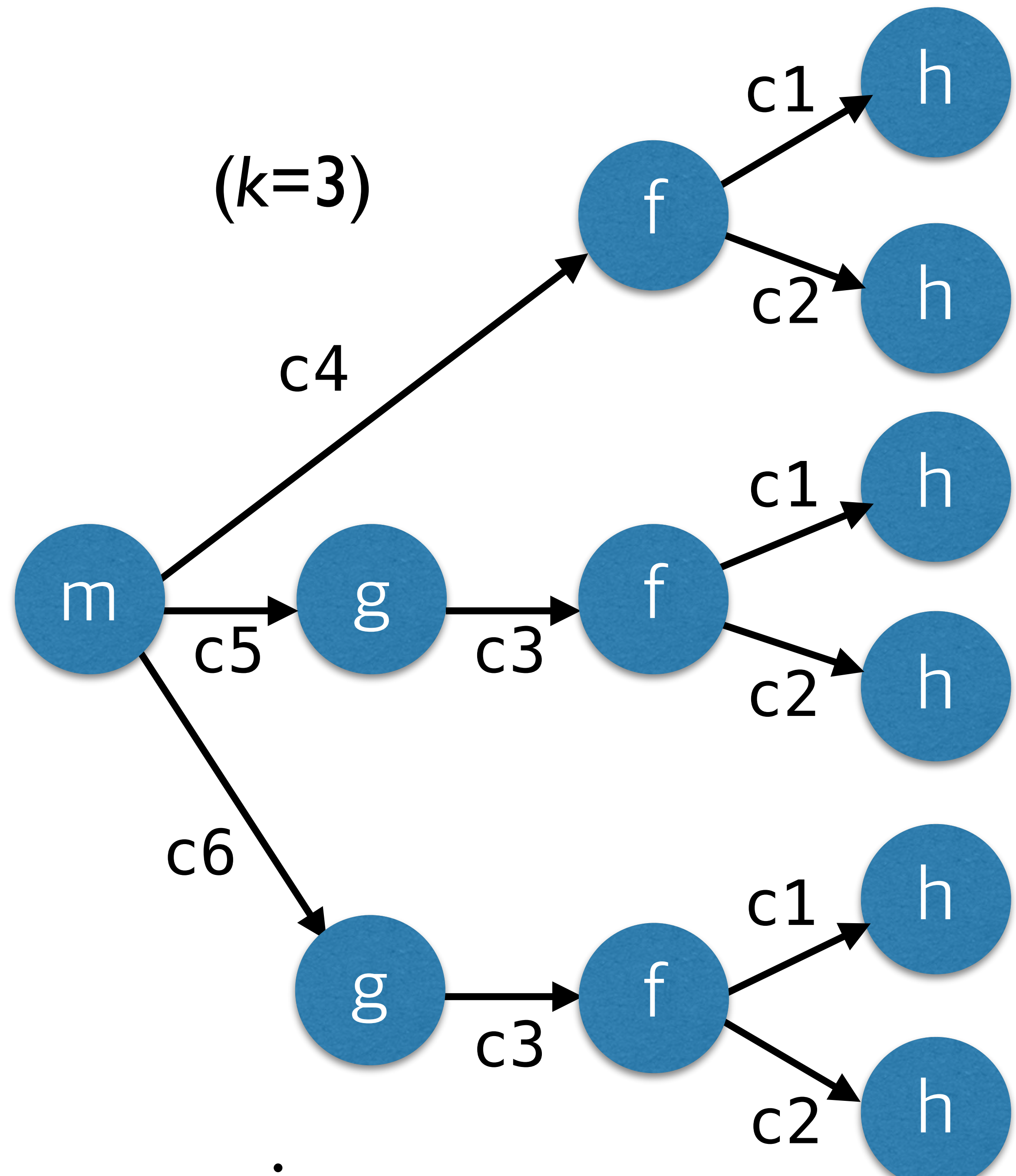


cheap but imprecise

k-Context-Sensitive Analysis

- Analyze functions separately for each calling context

```
    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();
    }
```



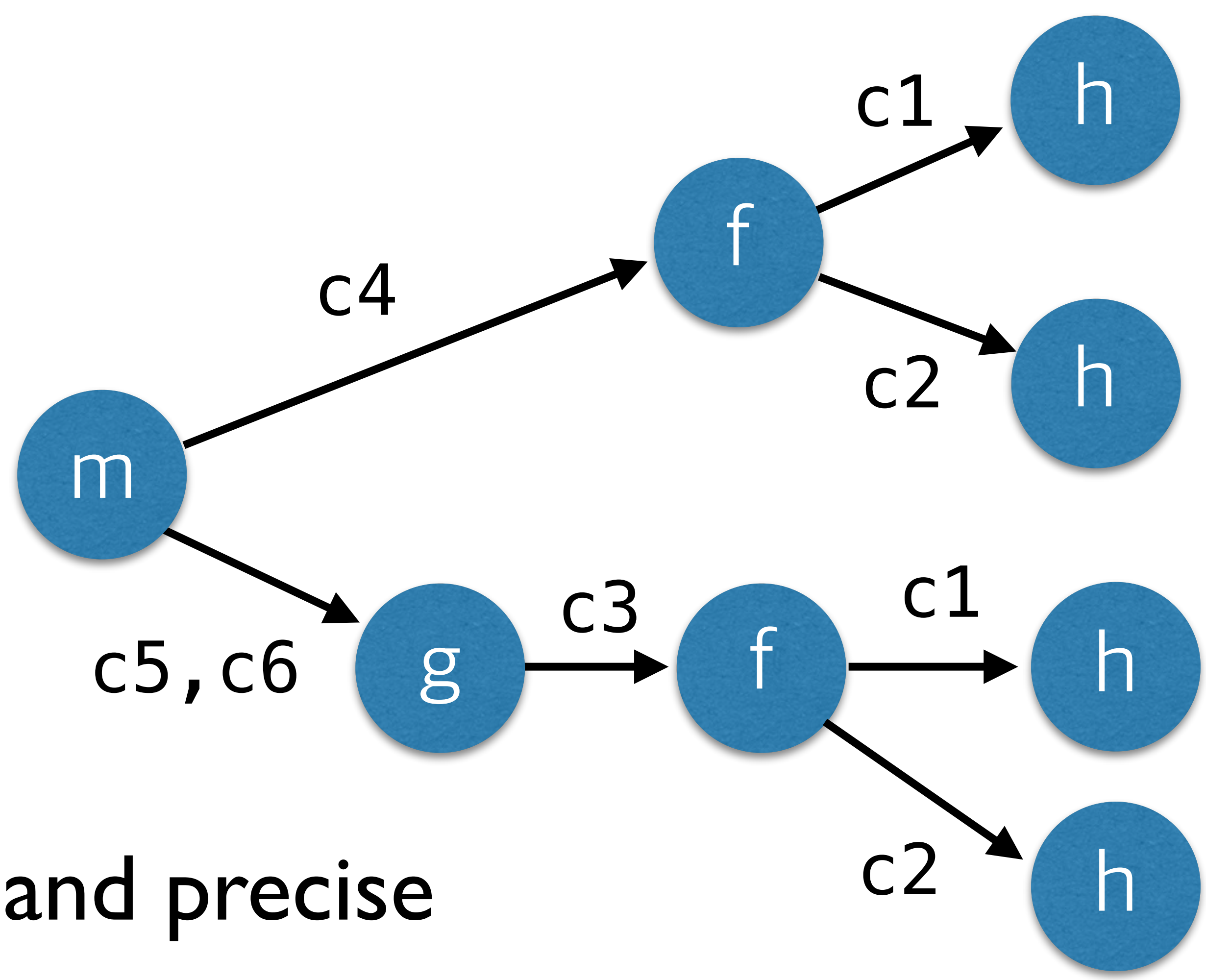
precise but expensive

Selective Context-Sensitivity

- Selectively differentiate contexts only when necessary

```
    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

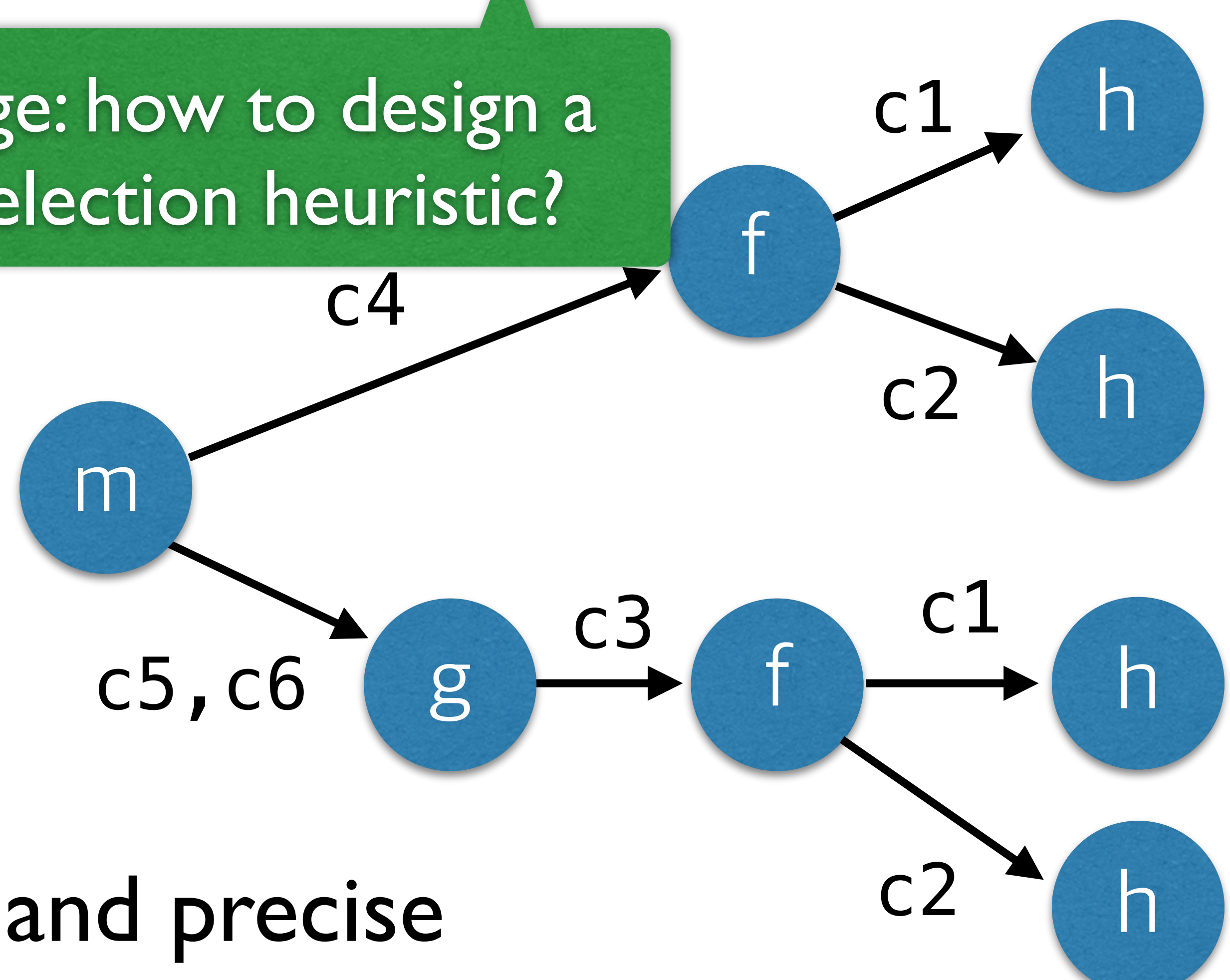
Selective Context-Sensitivity

- Selectively differentiate contexts only when necessary

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

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

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}

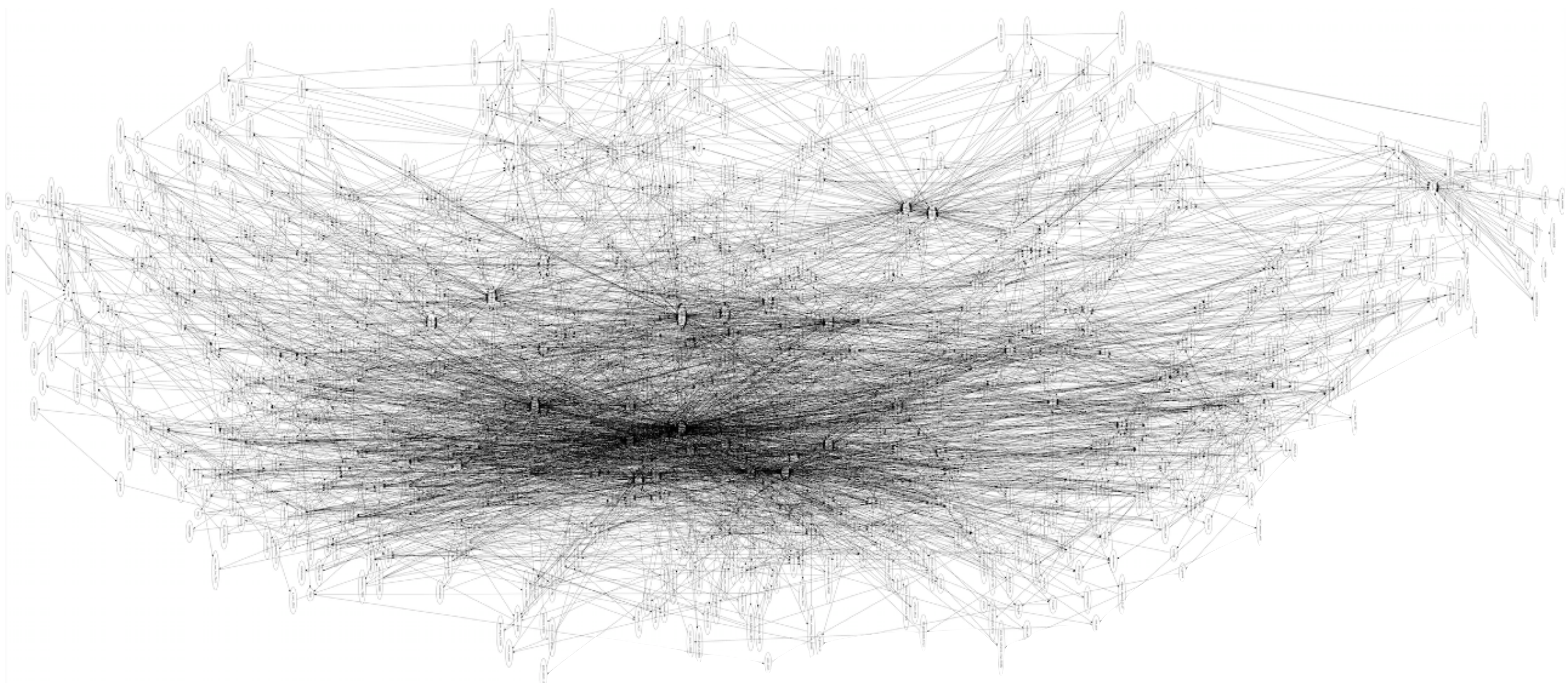
Challenge: how to design a good selection heuristic?



cheap and precise

Hard Search Problem

- Intractably large and sparse search space, if not infinite
  - e.g., S^k choices where $S = 2^{|Proc|}$ for *k*-context-sensitivity
- Real programs are complex to reason about
  - e.g., typical call-graph of real program:



A fundamental problem in selective program analysis
=> New data-driven approach

Existing Approaches

- Selection heuristics manually crafted by analysis experts:
  - pre-analysis [PLDI'14a, PLDI'14b, OOPSLA'18c]
  - dynamic analysis [POPL'12]
  - online refinement [PLDI'14c, POPL'17]
- Our claim: manual approaches are inherently limited:
  - nontrivial, sub-optimal, and unstable

Our direction: automatically generate heuristics via learning

Direction and Achievement

- Learning algorithms for data-driven program analysis
  - learning models [OOPSLA'17a]
  - optimization algorithms [TOPLAS'19]
  - feature engineering [OOPSLA'17b]
- State-of-the-art program analyses enabled by algorithms
  - interval / pointer analysis [OOPSLA'18a, TOPLAS'18]
  - symbolic analysis / execution [ICSE'18, ASE'18]
  - others program analyses [FSE'18, OOPSLA'18b]

9 papers in top-tier PL/SE conferences and journals

OOPSLA'17a
TOPLAS'19

Learning Algorithm Overview

Parametric program analyzer

Training data (programs w/o labels)

Atomic features (a1,a2,...,a25)

e.g., procedures have invocation stmt, procedures return strings, etc

Learning Algorithm

Learned heuristic for applying context-sensitivity:

f2: procedures to apply 2-context-sensitivity

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

f1: procedures to apply 1-context-sensitivity

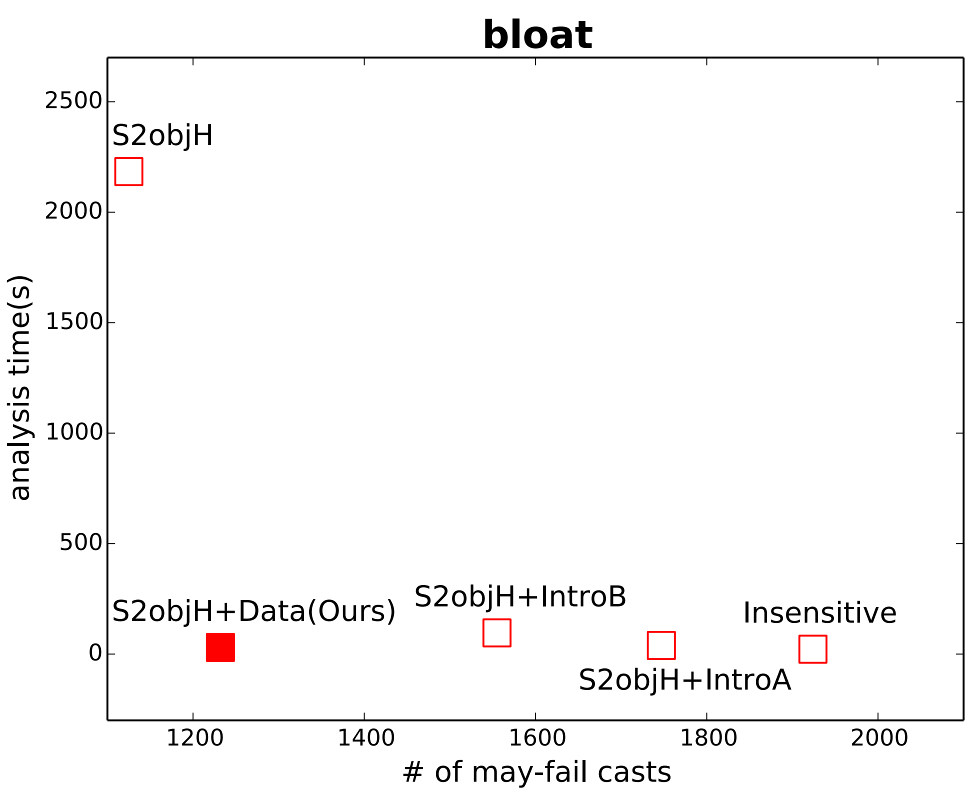
$$\begin{aligned}
&(1 \land \neg 3 \land \neg 4 \land \neg 7 \land \neg 8 \land 6 \land \neg 9 \land \neg 15 \land \neg 16 \land \neg 17 \land \neg 18 \land \neg 19 \land \neg 20 \land \neg 21 \land \neg 22 \land \neg 23 \land \neg 24 \land \neg 25) \lor \\
&(\neg 3 \land \neg 4 \land \neg 7 \land \neg 8 \land \neg 9 \land 10 \land 11 \land 12 \land 13 \land \neg 16 \land \neg 17 \land \neg 18 \land \neg 19 \land \neg 20 \land \neg 21 \land \neg 22 \land \neg 23 \land \neg 24 \land \neg 25) \lor \\
&(\neg 3 \land \neg 9 \land 13 \land 14 \land 15 \land \neg 16 \land \neg 17 \land \neg 18 \land \neg 19 \land \neg 20 \land \neg 21 \land \neg 22 \land \neg 23 \land \neg 24 \land \neg 25) \lor \\
&(1 \land 2 \land \neg 3 \land 4 \land \neg 5 \land \neg 6 \land \neg 7 \land \neg 8 \land \neg 9 \land \neg 10 \land \neg 13 \land \neg 15 \land \neg 16 \land \neg 17 \land \neg 18 \land \neg 19 \land \neg 20 \land \neg 21 \land \neg 22 \\
&\land \neg 23 \land \neg 24 \land \neg 25)
\end{aligned}$$

OOPSLA'17a

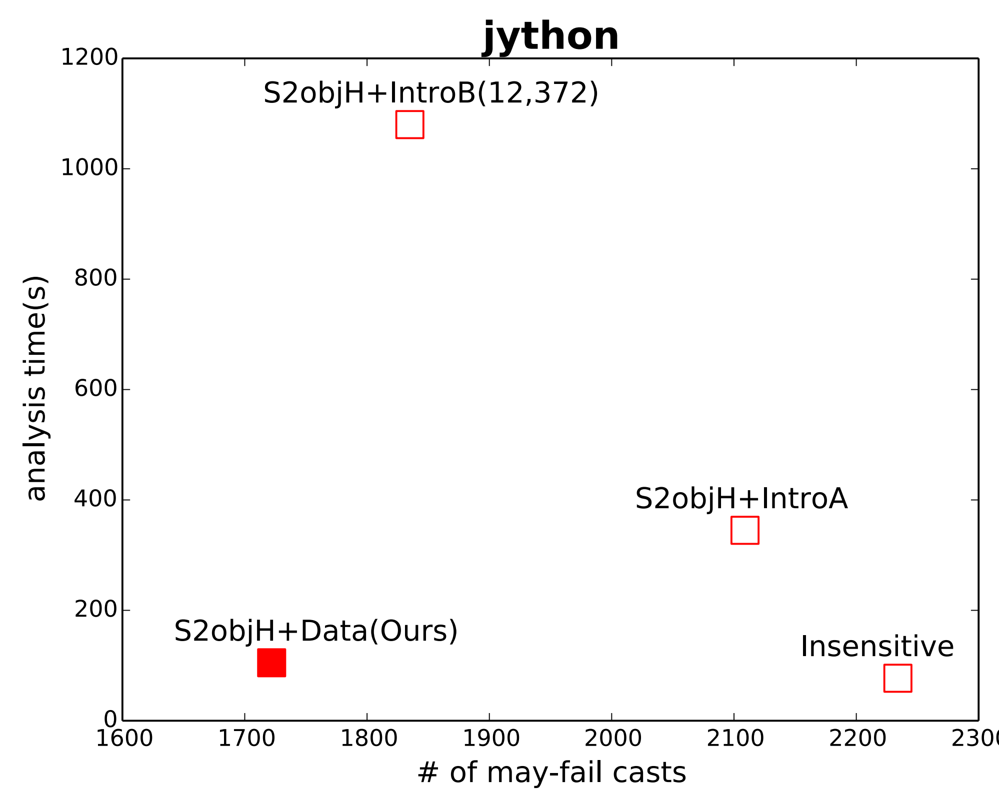
State-of-the-art Pointer Analysis

- Achieved state-of-the-art pointer analysis for Java
  - foundational static analysis for bug-finders, verifiers, etc
- Trained with 5 small programs from the DaCapo benchmark and tested with 5 remaining large programs

high scalability



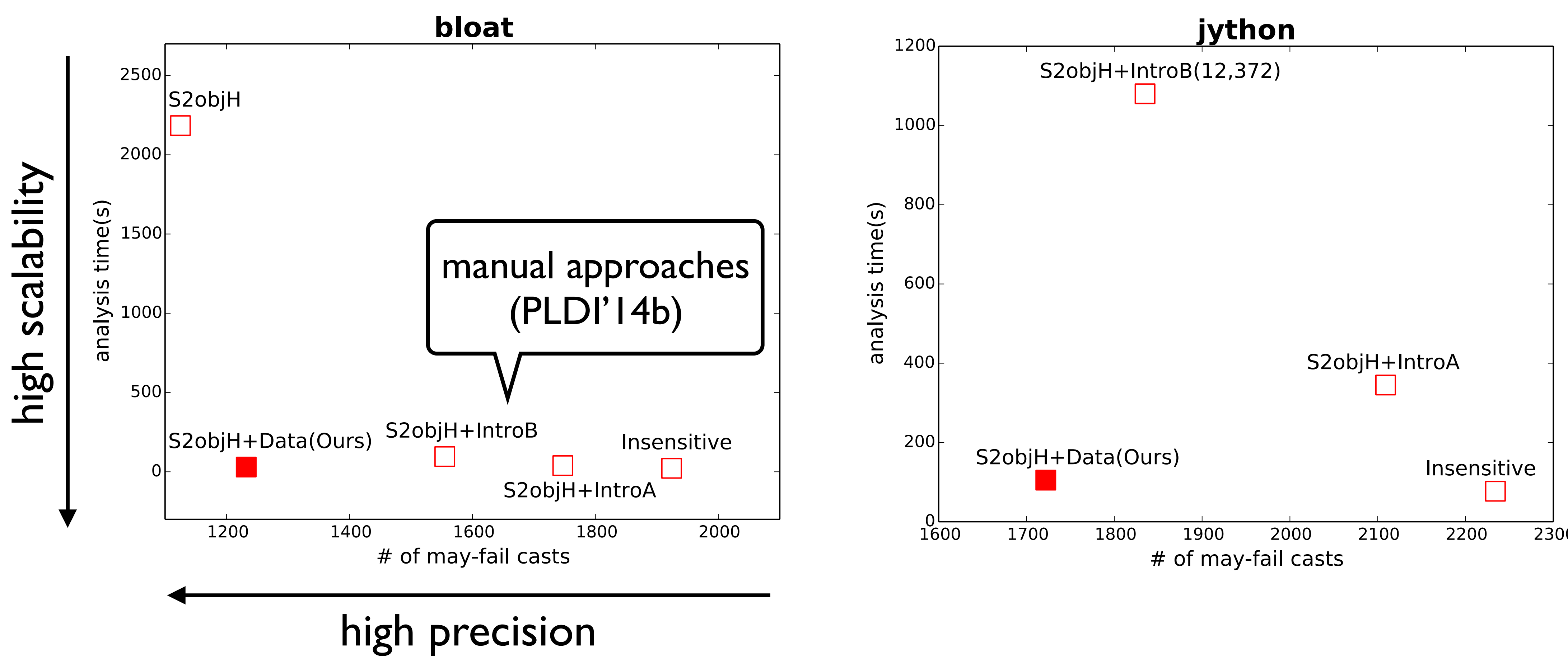
high precision



OOPSLA'17a

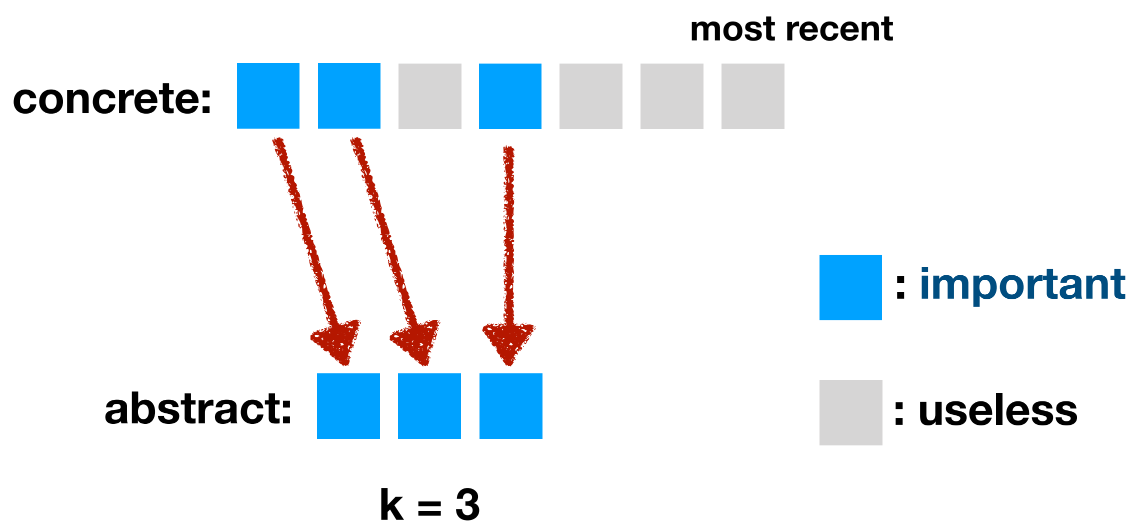
State-of-the-art Pointer Analysis

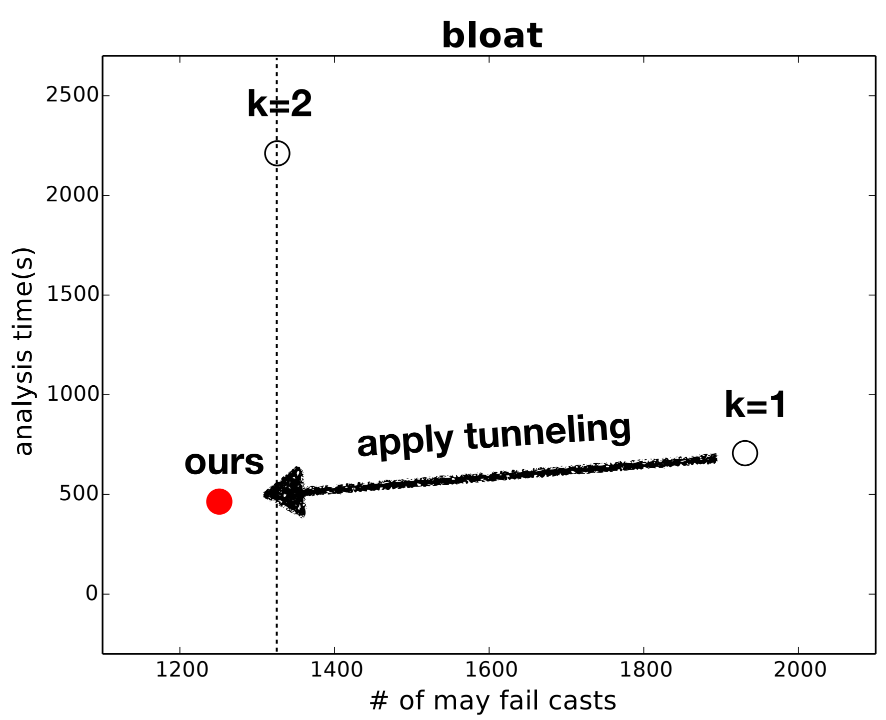
- Achieved state-of-the-art pointer analysis for Java
  - foundational static analysis for bug-finders, verifiers, etc
- Trained with 5 small programs from the DaCapo benchmark and tested with 5 remaining large programs



State-of-the-art Pointer Analysis

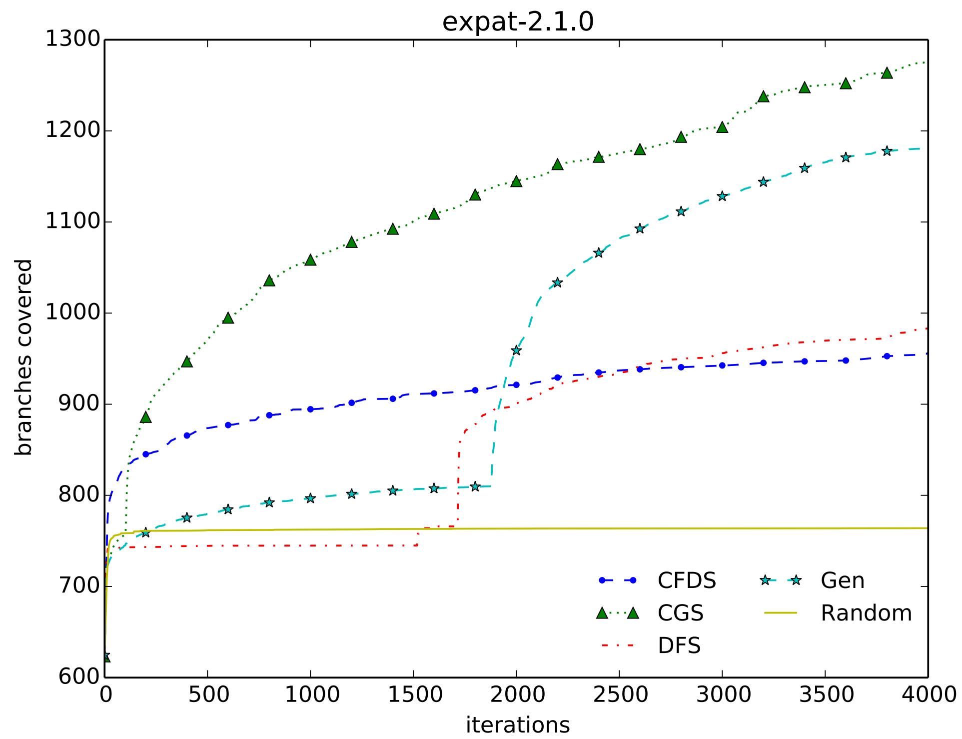
- Cracked down the precision limit of pointer analysis
- Key enablers:
    - keep most important *k*, rather than most recent *k*
    - data-driven method determines importance

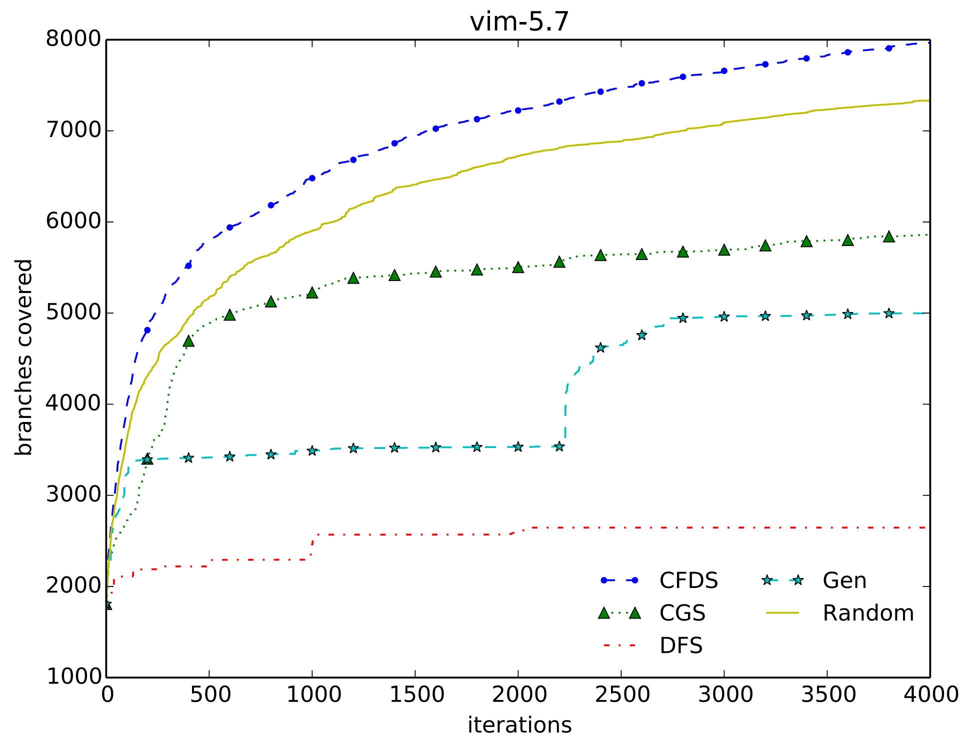




Application to Symbolic Execution

- Symbolic execution: program analysis popular for bug-finding
- Relies on path-selection heuristics, typically hand-tuned:
  - e.g., CGS [FSE'14], CarFast [FSE'12], CFDS [ASE'08], Generational [NDSS'08], DFS [PLDI'05], …



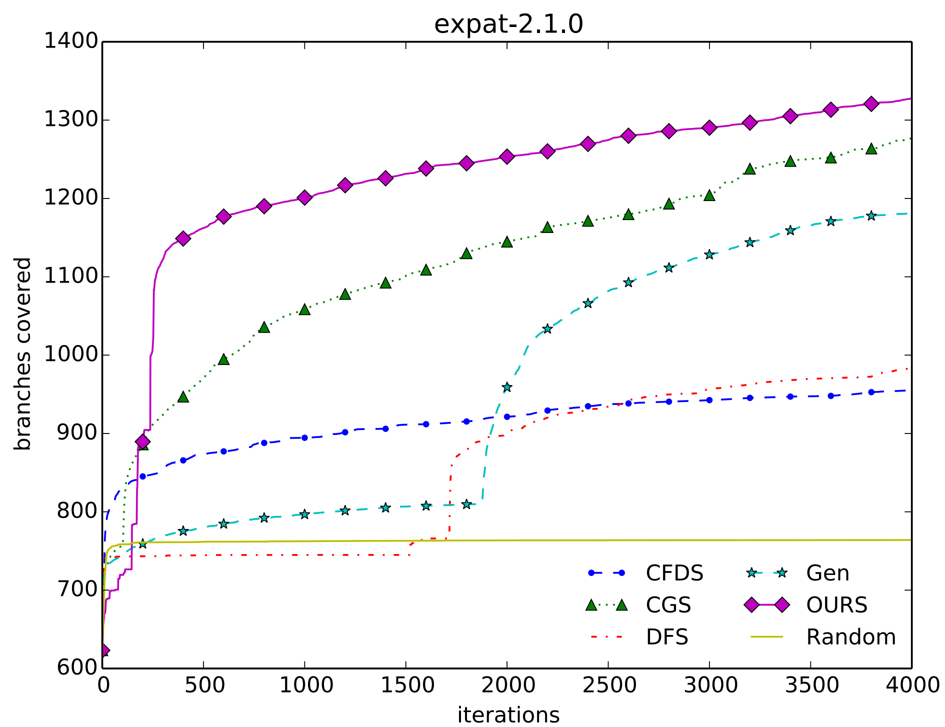


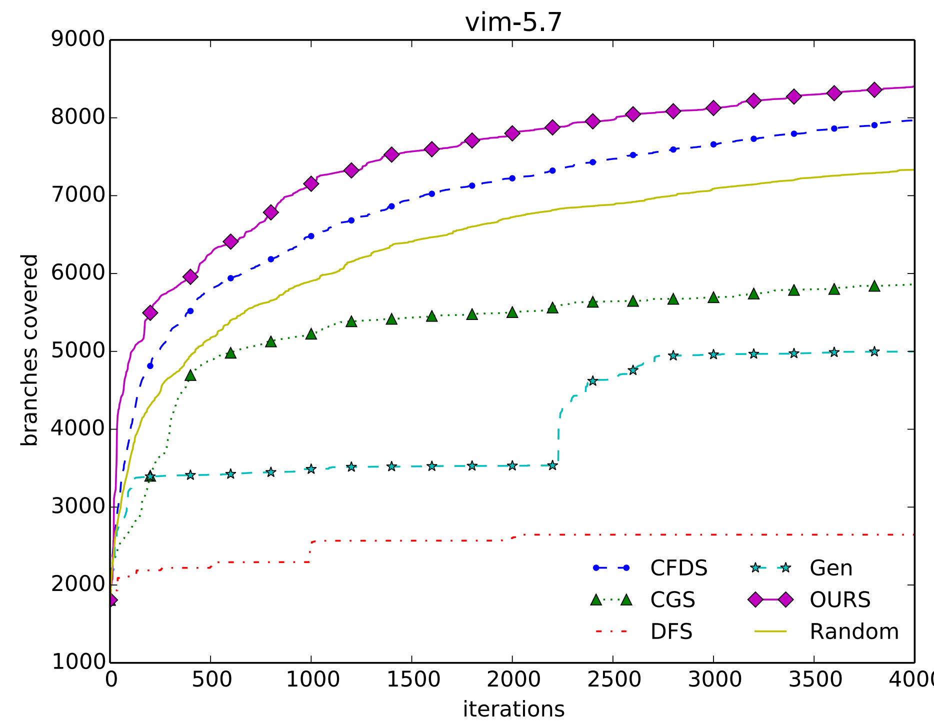
Our goal: automatically generating path-selection heuristics

ICSE'18

State-of-the-art Symbolic Execution

- Developed "data-driven symbolic execution"
  - considerable increase in code coverage





  - dramatic increase in bug-finding capability

	OURS	CFDS	CGS	Random	Gen	DFS
`gawk-3.0.3`	**100/100**	0/100	0/100	0/100	0/100	0/100
`grep-2.2`	**47/100**	0/100	5/100	0/100	0/100	0/100

<table>
<tr><th></th><th>Phenomenons</th><th>Bug-Triggering Inputs</th><th>Version</th></tr>
<tr><td>sed</td><td>Memory Exhaustion</td><td>'H
g
;D'</td><td>4.4(latest)</td></tr>
<tr><td>sed</td><td>Infinite File Write</td><td>'H
w {-
x; D'</td><td>4.4(latest)</td></tr>
<tr><td>grep</td><td>Segmentation Fault</td><td>'\(\)\1\+**'</td><td>3.1(latest)</td></tr>
<tr><td>grep</td><td>Non-Terminating</td><td>'?(^(|^+*)*\+\{8957\}'</td><td>3.1(latest)</td></tr>
<tr><td>gawk</td><td>Memory Exhaustion</td><td>'$6672467e2=E7'</td><td>4.21(latest)</td></tr>
</table>

Application to Program Repair

- Program analysis for automatically finding and fixing sw errors
- Manual debugging is time-consuming and error-prone
  - e.g., double-free error in Linux kernel:

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

double-free

buggy program

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

memory leak

```
in = malloc(1);
out = malloc(1);
... // use in, out
// removed
free(in);

in = malloc(2);
if (in == NULL) {
  out = NULL;
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

new errors introduced

```
in = malloc(1);
out = malloc(1);
... // use in, out
// removed
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  // removed

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

correct program

Our goal: data-driven static analysis for program repair

FSE'18

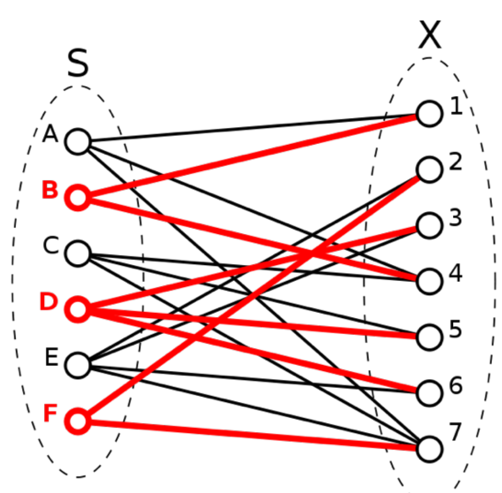
State-of-the-art Program Repair

- Safe and sound repair technique for memory errors
  - no errors introduced, generated patches are correct
- Static analysis enabled by our data-driven approach played a key role

Static Analysis

SAT Solver

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```



exact cover problem

```
in = malloc(1);
out = malloc(1);
... // use in, out
// removed
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  // removed

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

Being recognized as a new and promising research direction

"Overall, the paper addresses an important problem advancing the state of the art in the very interesting and promising area of data-driven program analysis." (from OOPSLA reviews)

"Supremely well-written and is clearly situated within related work, addressing a clearly high-profile long-standing problem of scalability." (from OOPSLA reviews)

"I really enjoyed reading. It tackles a fundamental problem in program analysis -- developing heuristics to localize precision, and presents a very novel approach." (from TOPLAS reviews)

"There is a novel idea, which is not only neat and elegant, but I think may apply to other machine learning domains in program analysis." (from OOPSLA reviews)

"The algorithm that automatically generates search heuristics for concolic testing is novel and very interesting." (from ICSE reviews)

Future Research Directions

- Just started; Need to extend depth and breadth
- Foundational algorithms for data-driven program analysis
  - expressiveness, efficiency, generality, automation
  - unified and reusable framework
- Applications to various real problems
  - heuristics × analyses × languages
- Deployment in industrial tools

Sparrow The Early Bird

DeepScan

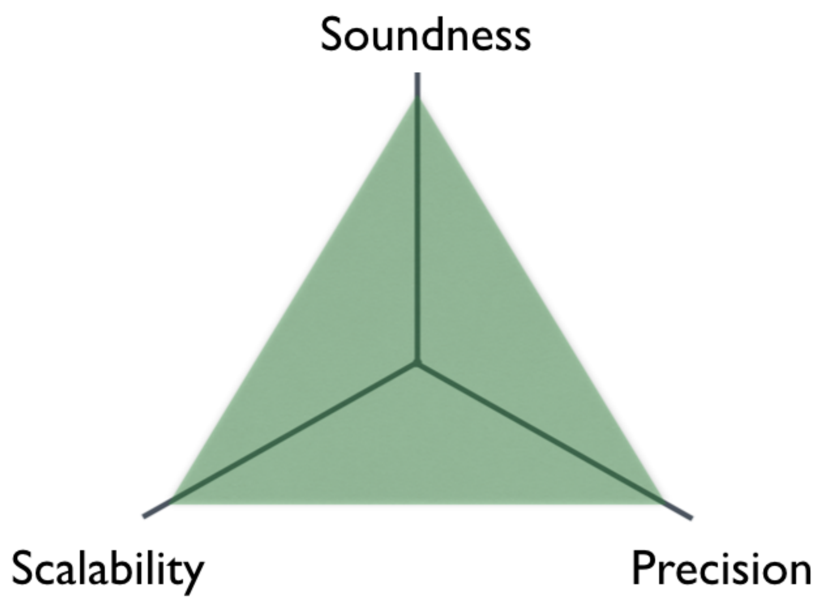
WALA T. J. WATSON LIBRARIES FOR ANALYSIS

Infer

Summary

Our Data-Driven Program Analysis

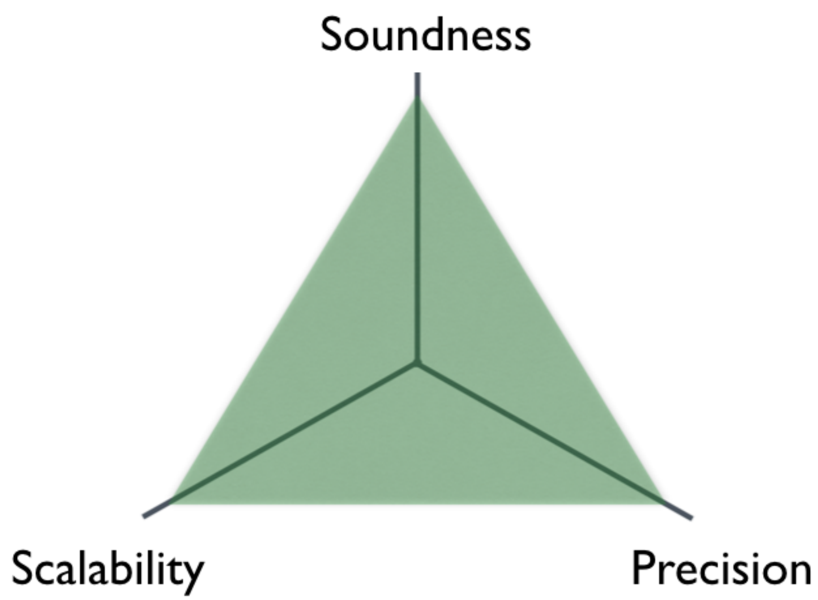
- **Goal**: Achieving the ideal program analysis technology
- **Approach**: Data-driven program analysis
- **Research Directions**:
  - new and foundational algorithms
  - practical and diverse applications
- **Impacts**:
  - solving the longstanding open problem
  - paradigm-shift in program analysis research



Summary

Our Data-Driven Program Analysis

- **Goal**: Achieving the ideal program analysis technology
- **Approach**: Data-driven program analysis
- **Research Directions**:
  - new and foundational algorithms
  - practical and diverse applications
- **Impacts**:
  - solving the longstanding open problem
  - paradigm-shift in program analysis research



Thank you

References

Ours

- [PLDI'14a] Selective context-sensitivity guided by impact pre-analysis
- [OOPSLA'15] Learning a Strategy for Adapting a Program Analysis via Bayesian Optimisation
- [OOPSLA'17a] Data-driven context-sensitivity for points-to analysis
- [OOPSLA'17b] Automatically generating features for learning program anlaysis heuristics
- [OOPSLA'18a] Precise and scalable points-to analysis via data-driven context tunneling
- [OOPSLA'18b] Automatic diagnosis and correction of logical errors for functional programming assignments
- [ICSE'18] Automatically generating search heuristics for concolic testing
- [FSE'18] MemFix: Static-analysis-based repair of memory deallocation errors for C
- [ASE'18] Template-guided concolic testing via online learning
- [TOPLAS'18] Adaptive static analysis via learning with bayesian optimization
- [TOPLAS'19] A machine-learning algorithm with disjunctive model for data-driven program analysis

Others

- [PLDI'14b] Introspective analysis: context-sensitivity, across the board
- [PLDI'14c] On abstraction refinement for program analyses in Datalog
- [POPL'12] Abstractions from tests
- [POPL,17] Semantic-directed clumping of disjunctive abstract states
- [OOPSLA'18c] Precision-guided context-sensitivity for pointer analysis

Learning Algorithm Detail

- Each sub-heuristic f_i is a boolean combination of features

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

- The learning problem:

  Find $f_1, f_2, \ldots, f_k$ that maximizes the performance of program analysis over codebase

- Our algorithm reduces the search space from S^k to $k \cdot S$ while formally guaranteeing to preserve global maxima

- Efficient algorithm for searching the subspace S via iterative and greedy refinement