Data-Driven Static Analysis

Minseok Jeon
Korea University
Nov, 15, 2023 @ POSTECH

Generalization

Data-Driven Static Analysis

& PL-based Explainable Graph Machine Learning

Minseok Jeon

Korea University

Nov, 15, 2023 @ POSTECH

Generalization

Data-Driven Static Analysis

& PL-based Explainable Graph Machine Learning

Part 1 Part 2

Minseok Jeon

Korea University

Nov, 15, 2023 @ POSTECH

Part 1:
Data-Driven Static Analysis

Static Program Analysis

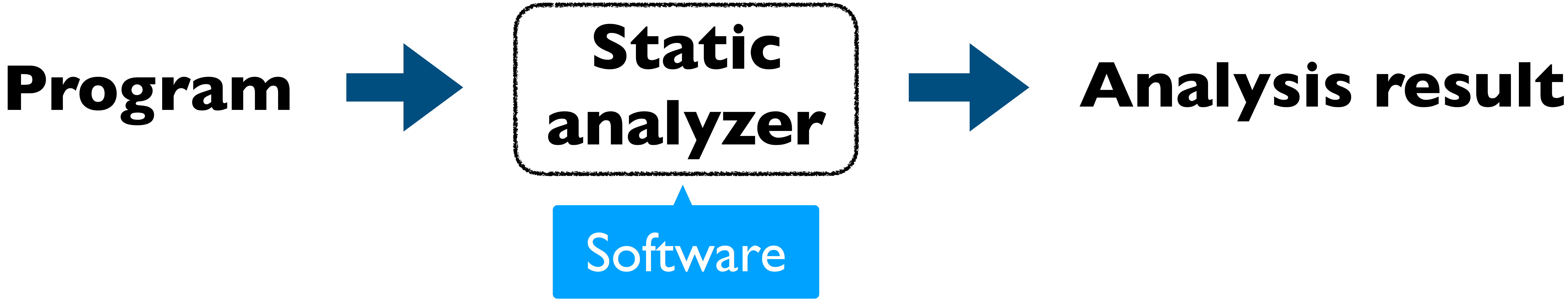
- Automatically, statically, and soundly predict software behavior (e.g., bugs)
  - Automatically: software analyzes software
  - Statically: analyzing program source code without execution
  - Soundly: program analysis finds all the bugs
- Widely used in software industry

...

Static Program Analysis

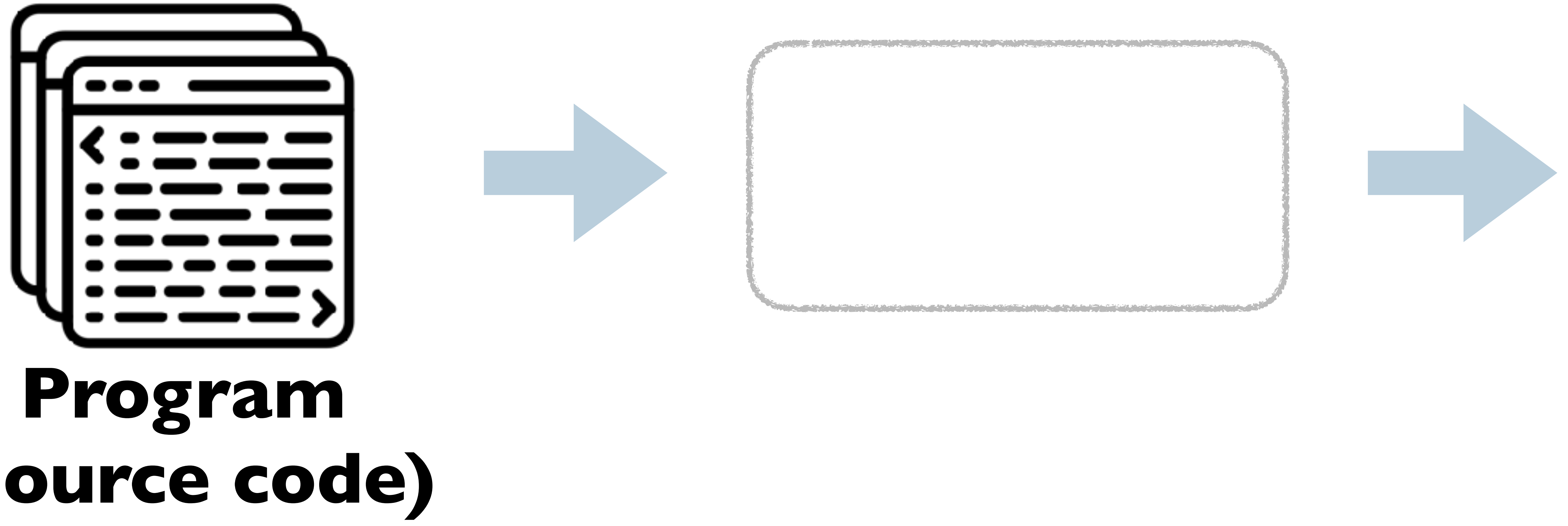
- Automatically, statically, and soundly predict software behavior (e.g., bugs)

  - Static analyzer is a software that analyzes software



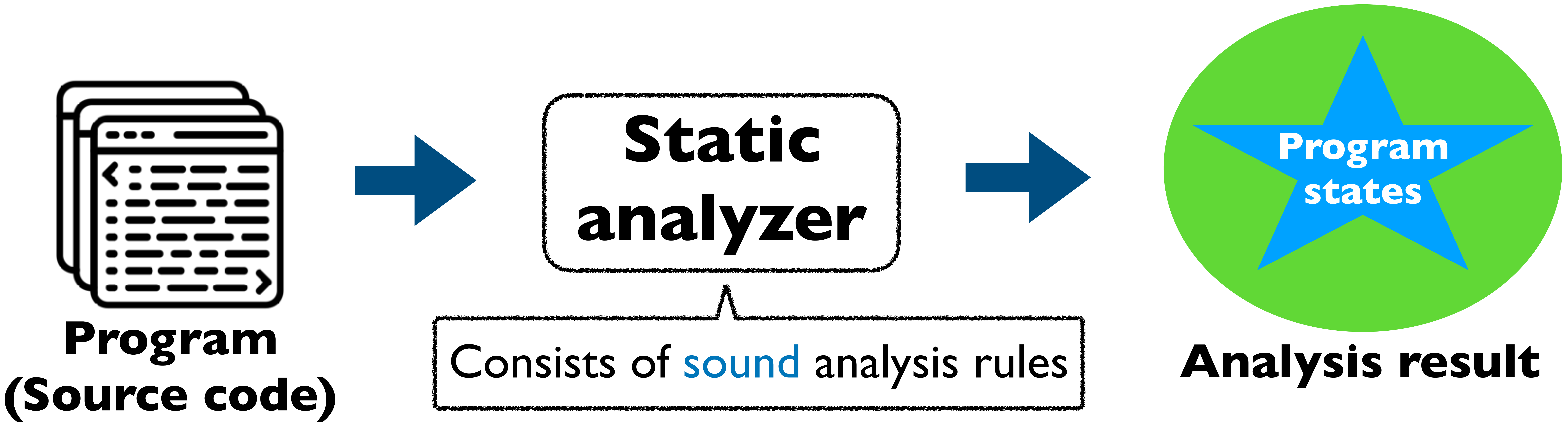
Static Program Analysis

- Automatically, statically, and soundly predict software behavior (e.g., bugs)
  - Static analyzer analyzes program **source code** without execution



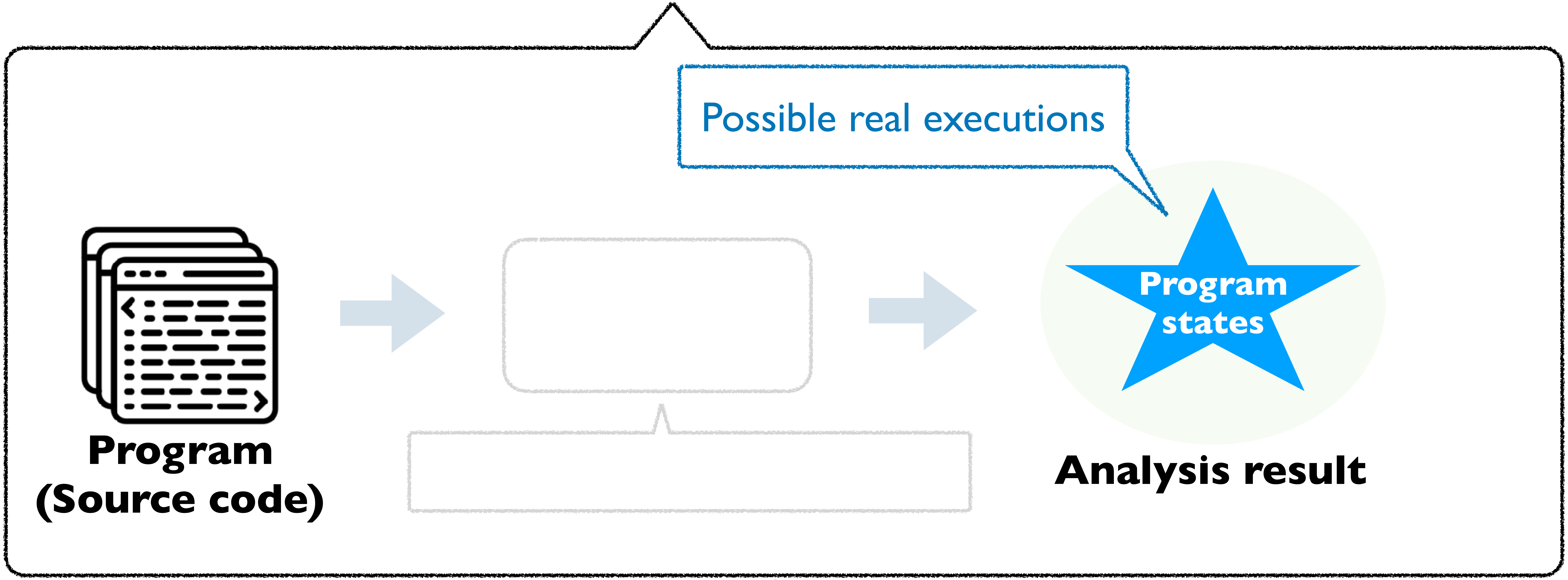
Static Program Analysis

- Automatically, statically, and soundly predict software behavior (e.g., bugs)
  - Static analyzer computes an over-approximation of program behavior



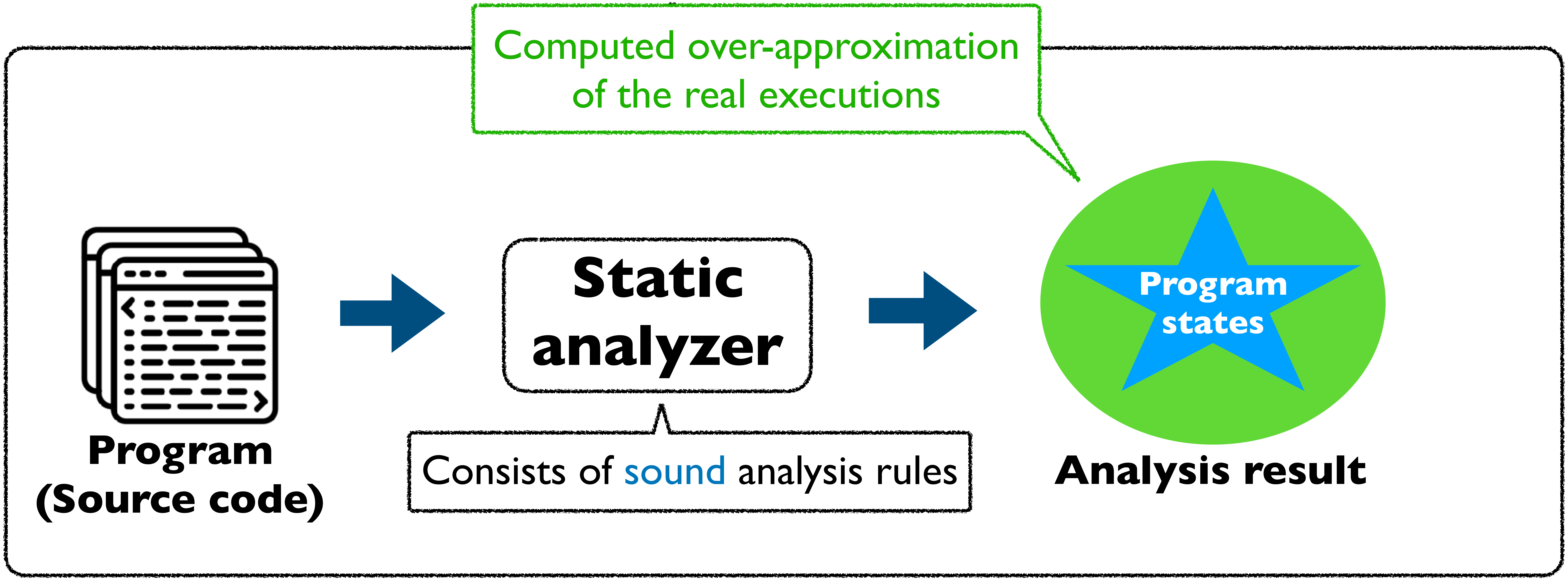
Static Program Analysis

- Automatically, statically, and soundly predict software behavior (e.g., bugs)



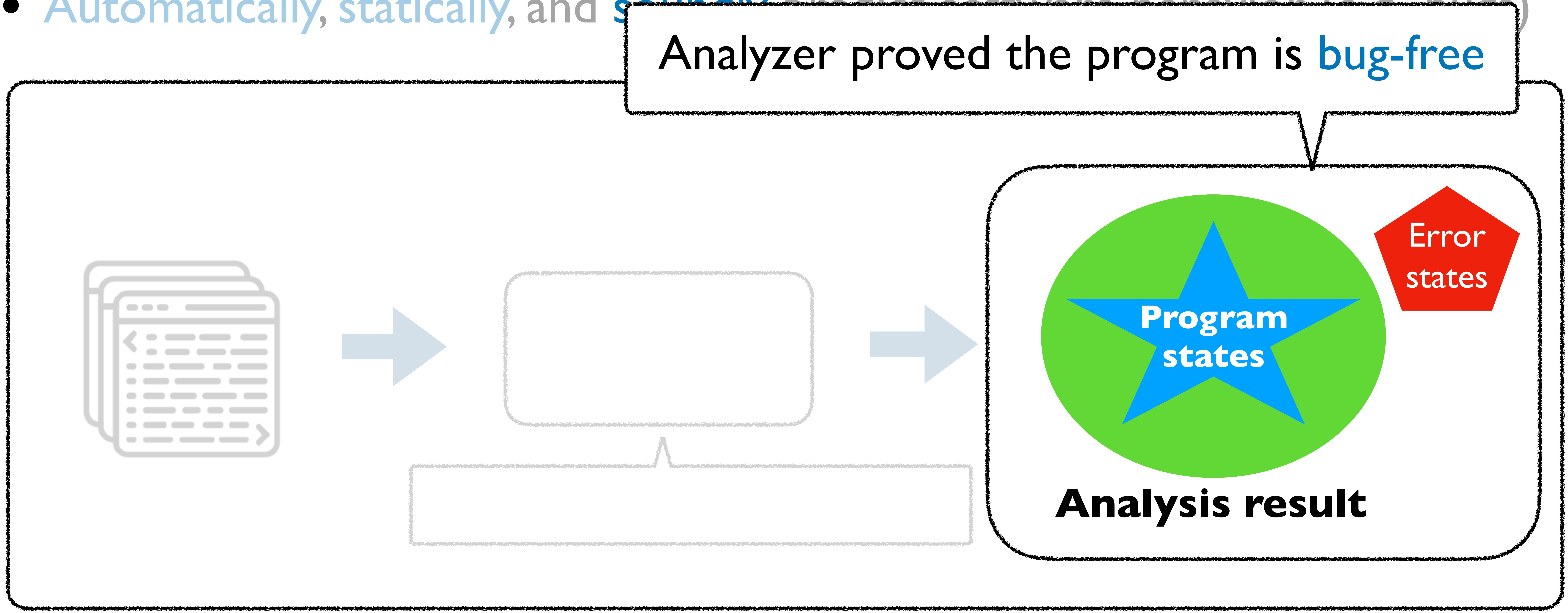
Static Program Analysis

- Automatically, statically, and soundly predict software behavior (e.g., bugs)



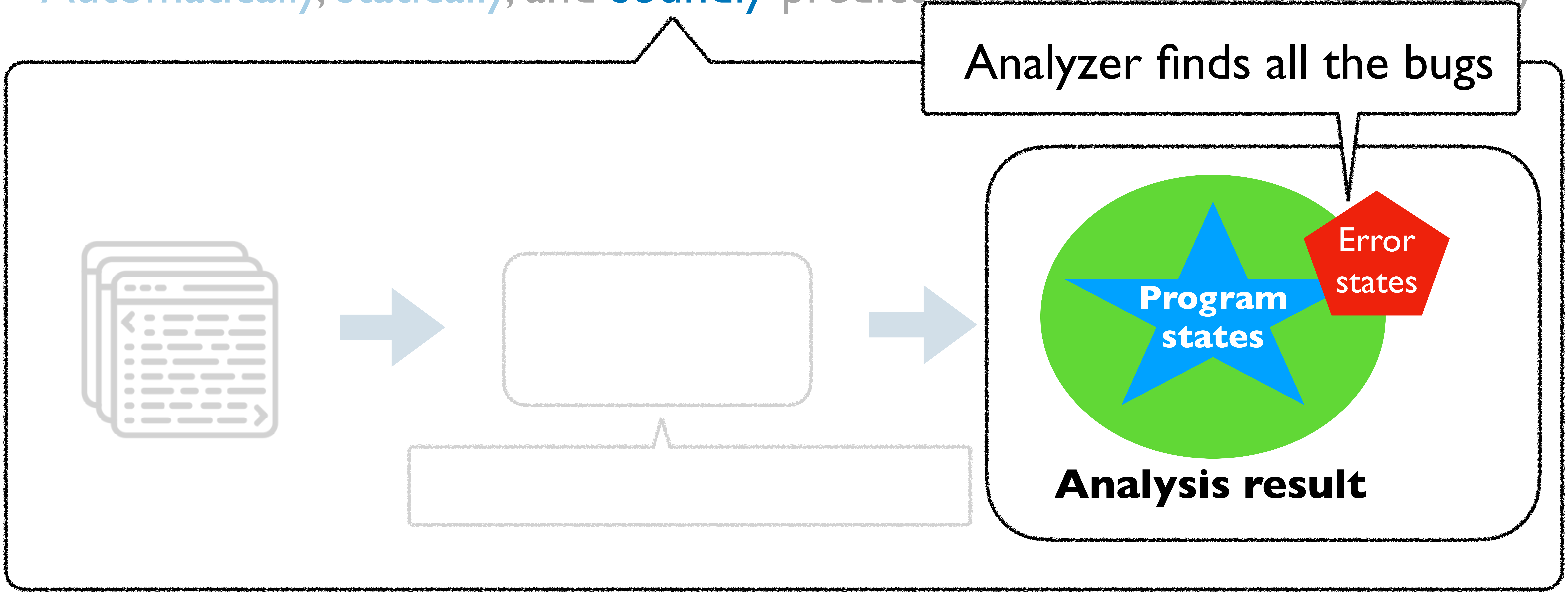
Static Program Analysis

- Automatically, statically, and s



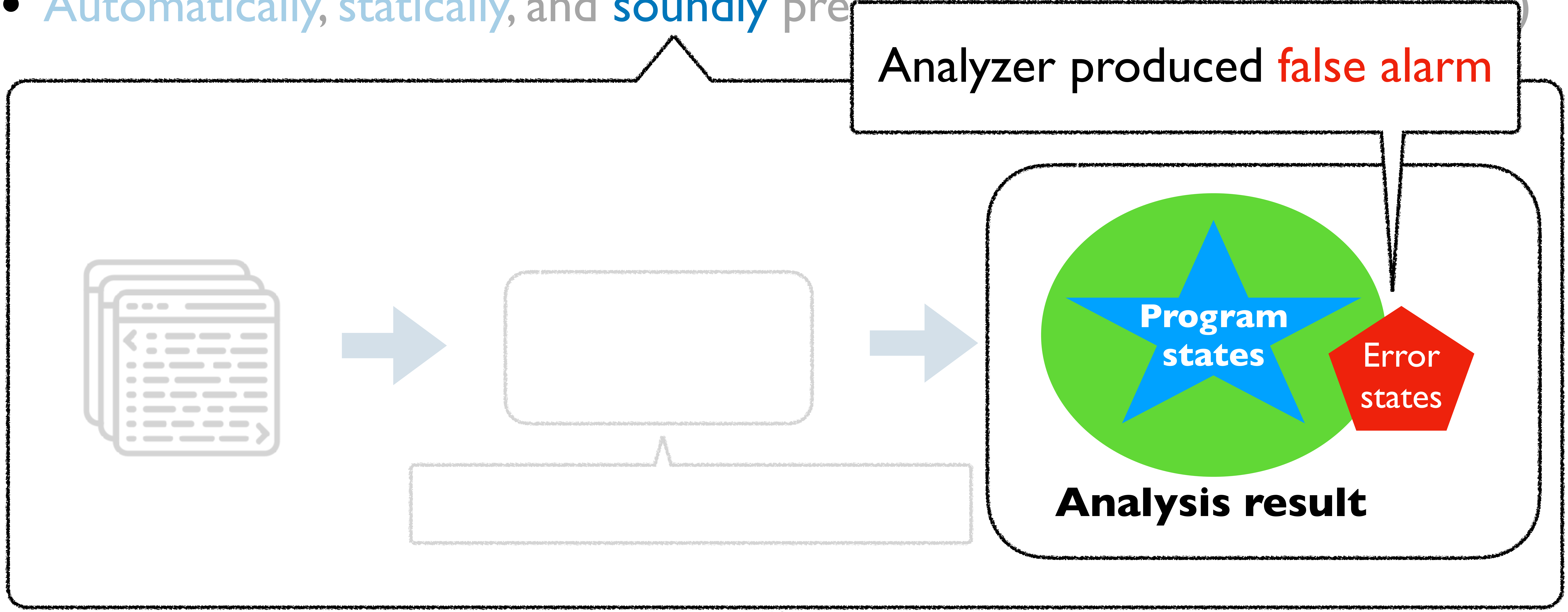
Static Program Analysis

- Automatically, statically, and soundly predict software behavior (e.g., bugs)



Static Program Analysis

- Automatically, statically, and soundly pre



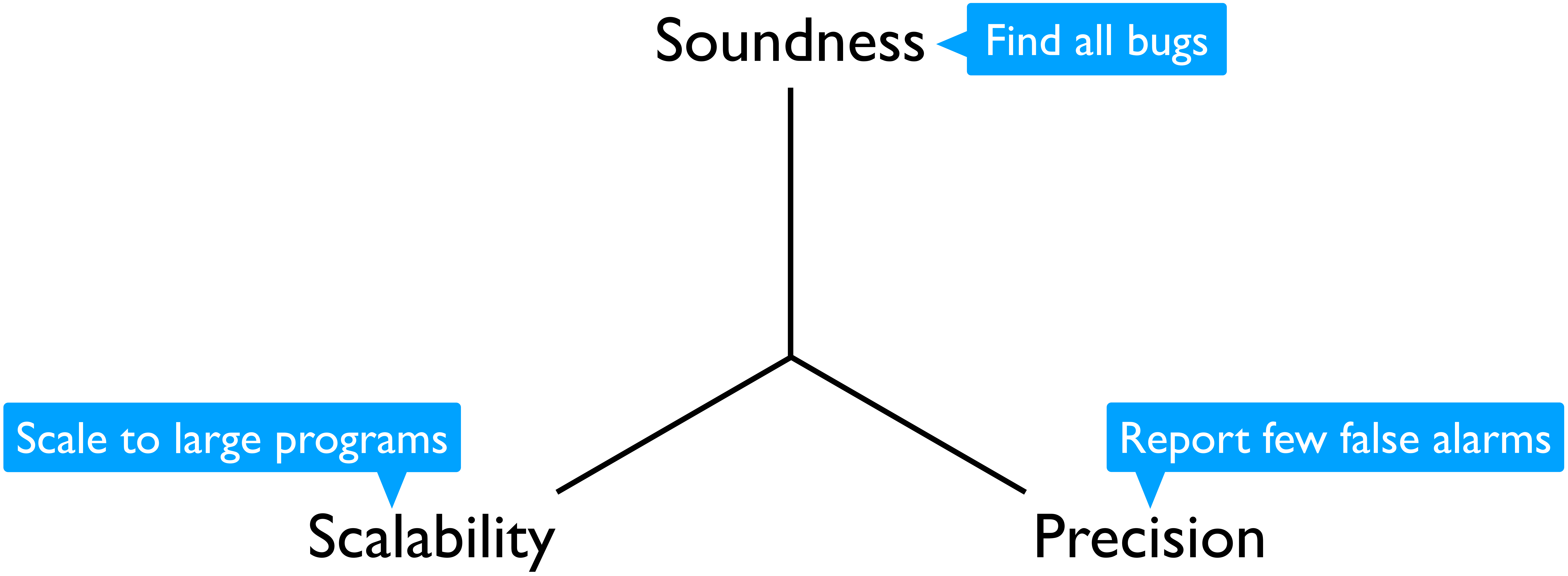
Static Program Analysis

- Widely used in software industry

…

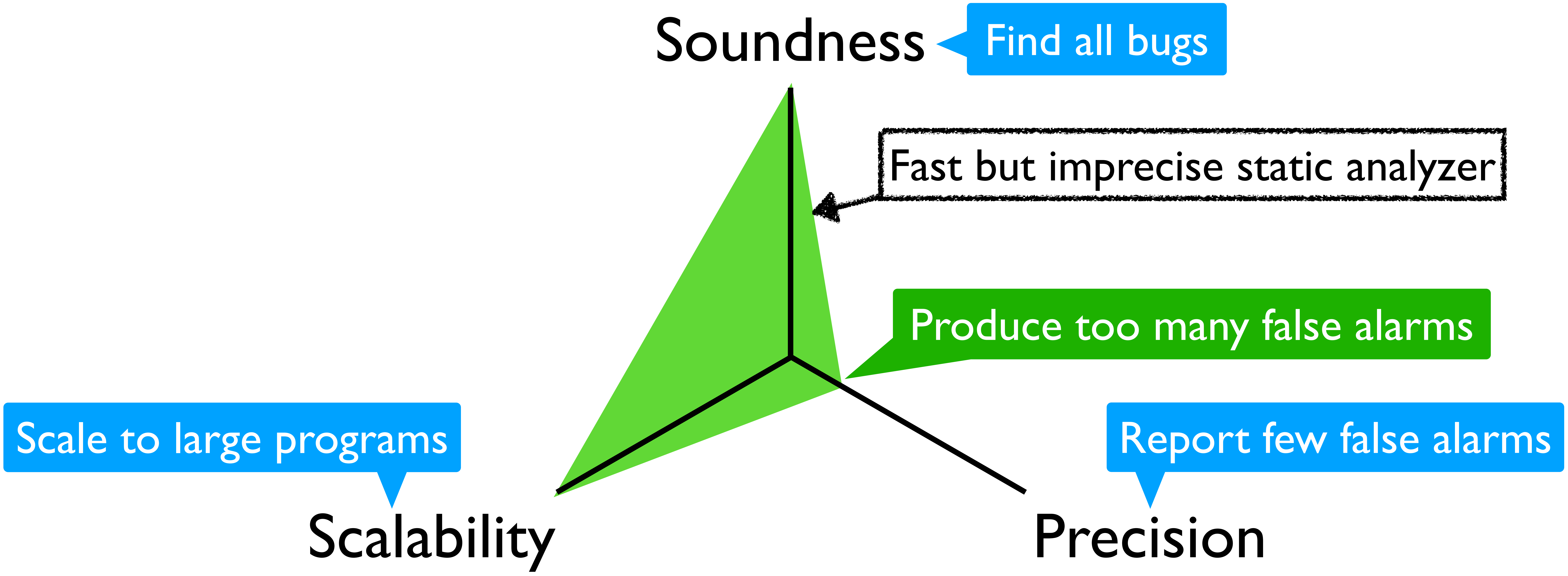
Long Standing Open Problem in Static Analysis

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



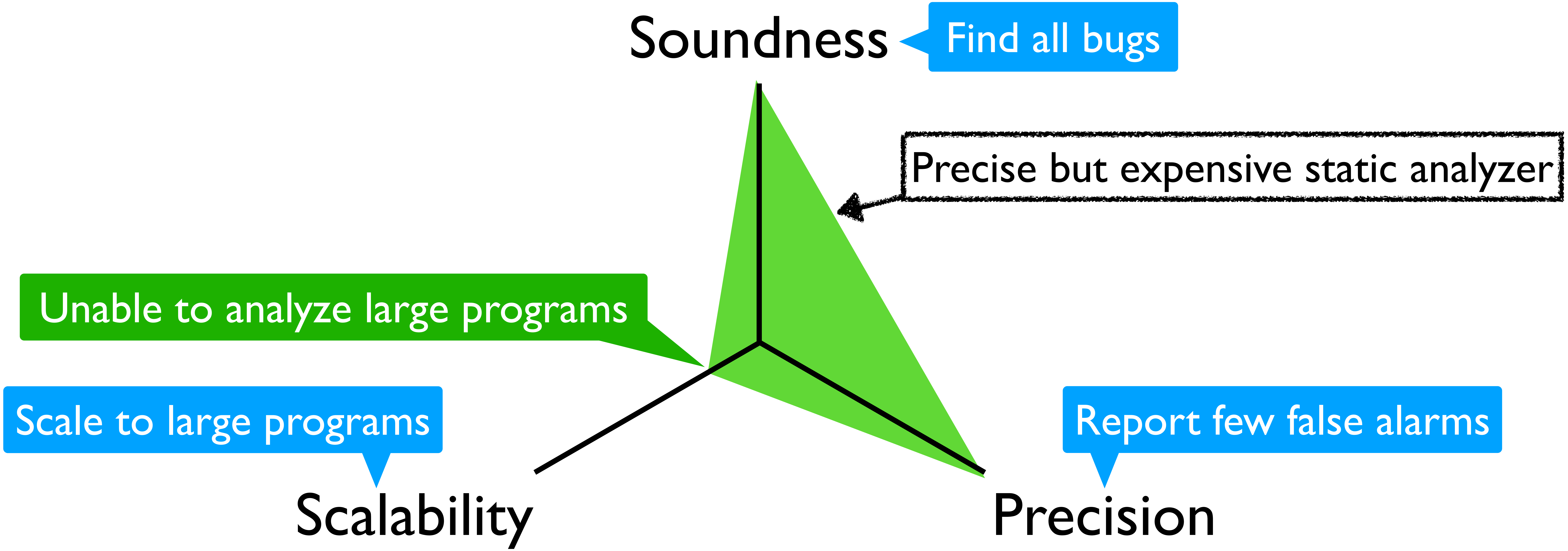
Long Standing Open Problem in Static Analysis

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



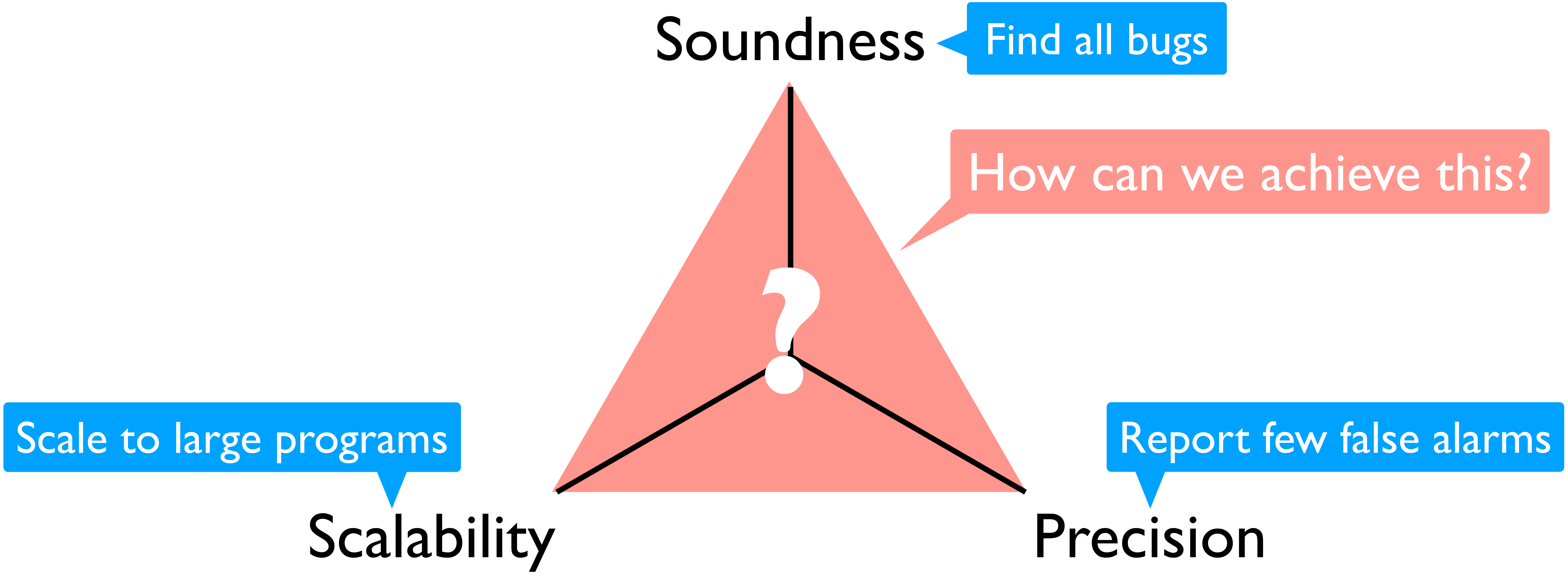
Long Standing Open Problem in Static Analysis

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



Long Standing Open Problem in Static Analysis

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



Example: Selective Context Sensitivity

- Suppose we analyze the left example program

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

Example program

There are four methods (main, f, g, h)

- Which methods need to be analyzed **precisely**?
- Which methods need to be analyzed **coarsely**?

Example: Selective Context Sensitivity

- Suppose we analyze the left example program

```
main(){
  f();//i1
  f();//i2
}

  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

x is always 10

Always holds (x = 10)

Example program

There are four methods (main, f, g, h)

- Which methods need to be analyzed **precisely**?
- Which methods need to be analyzed **coarsely**?

Example: Selective Context Sensitivity

- Suppose we analyze the left example program

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

Example program

There are four methods (main, f, g, h)

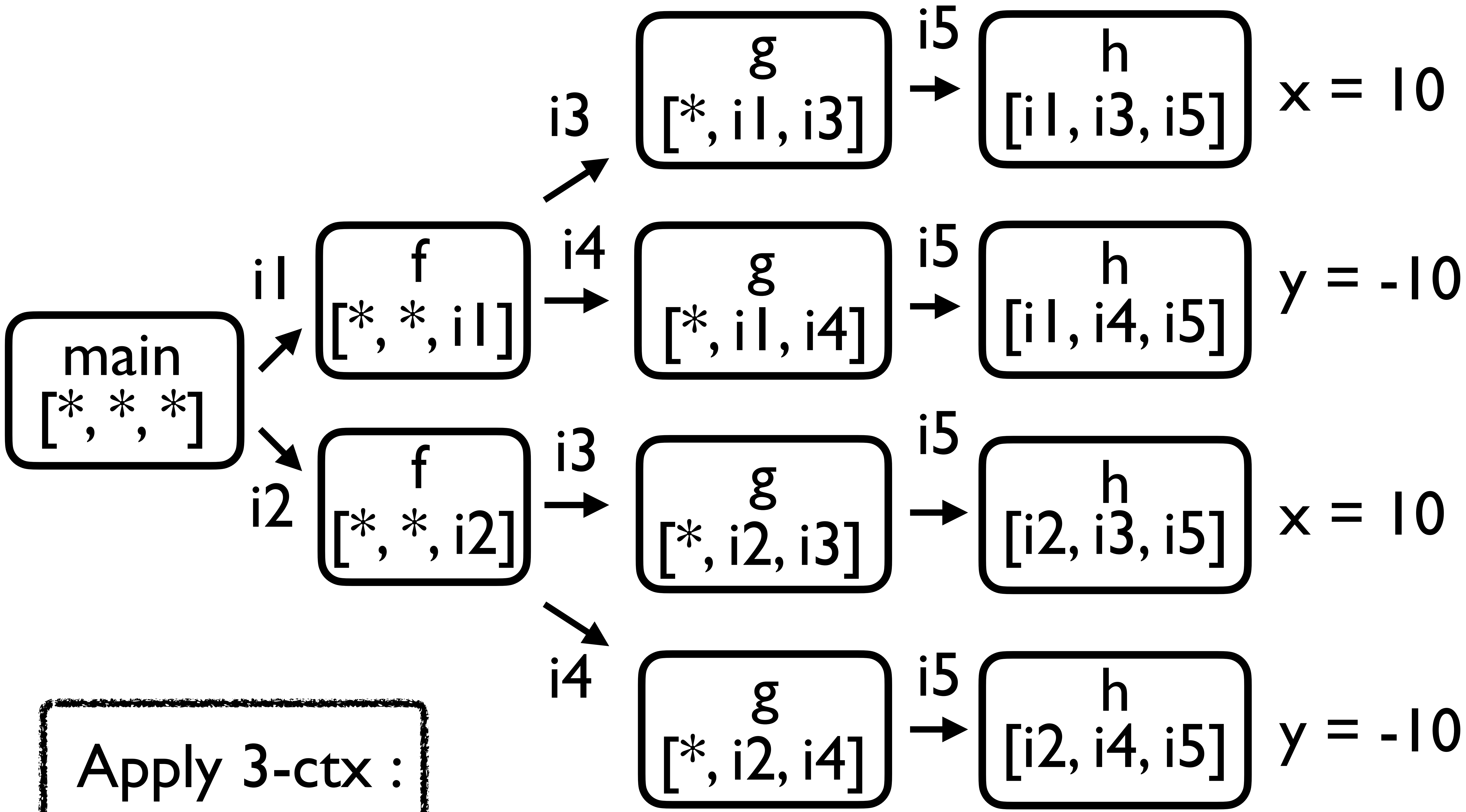
- Which methods need to be analyzed **precisely**?
- Which methods need to be analyzed **coarsely**?

Example: Selective Context Sensitivity

- **Precisely** analyzing all the method calls makes the analysis precise but **expensive**

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

Example program



Apply 3-ctx : {h,g,f,main}

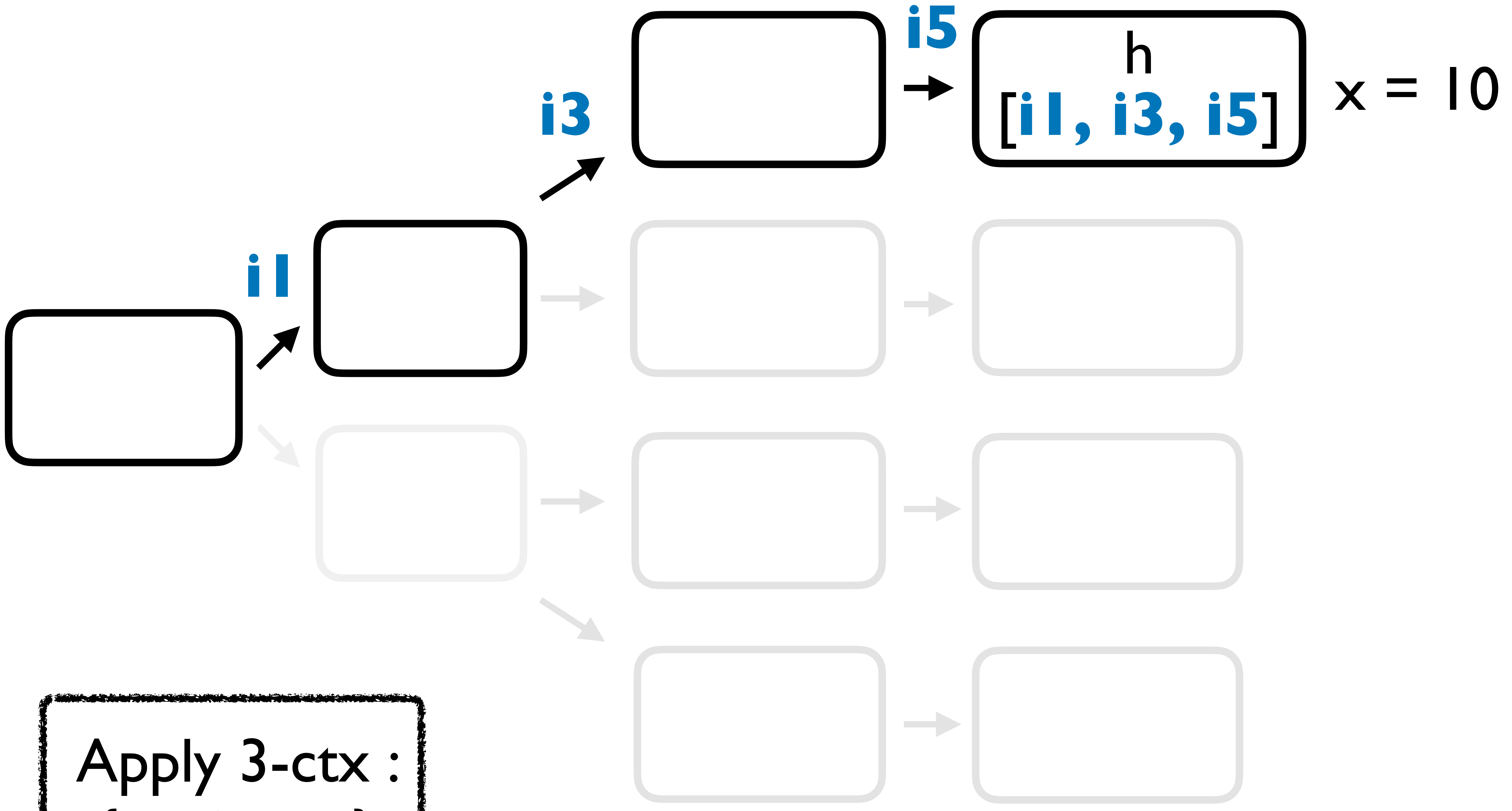
3-context sensitivity

Example: Selective Context Sensitivity

- **Precisely** analyzing all the method calls makes the analysis precise but **expensive**

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

Example program

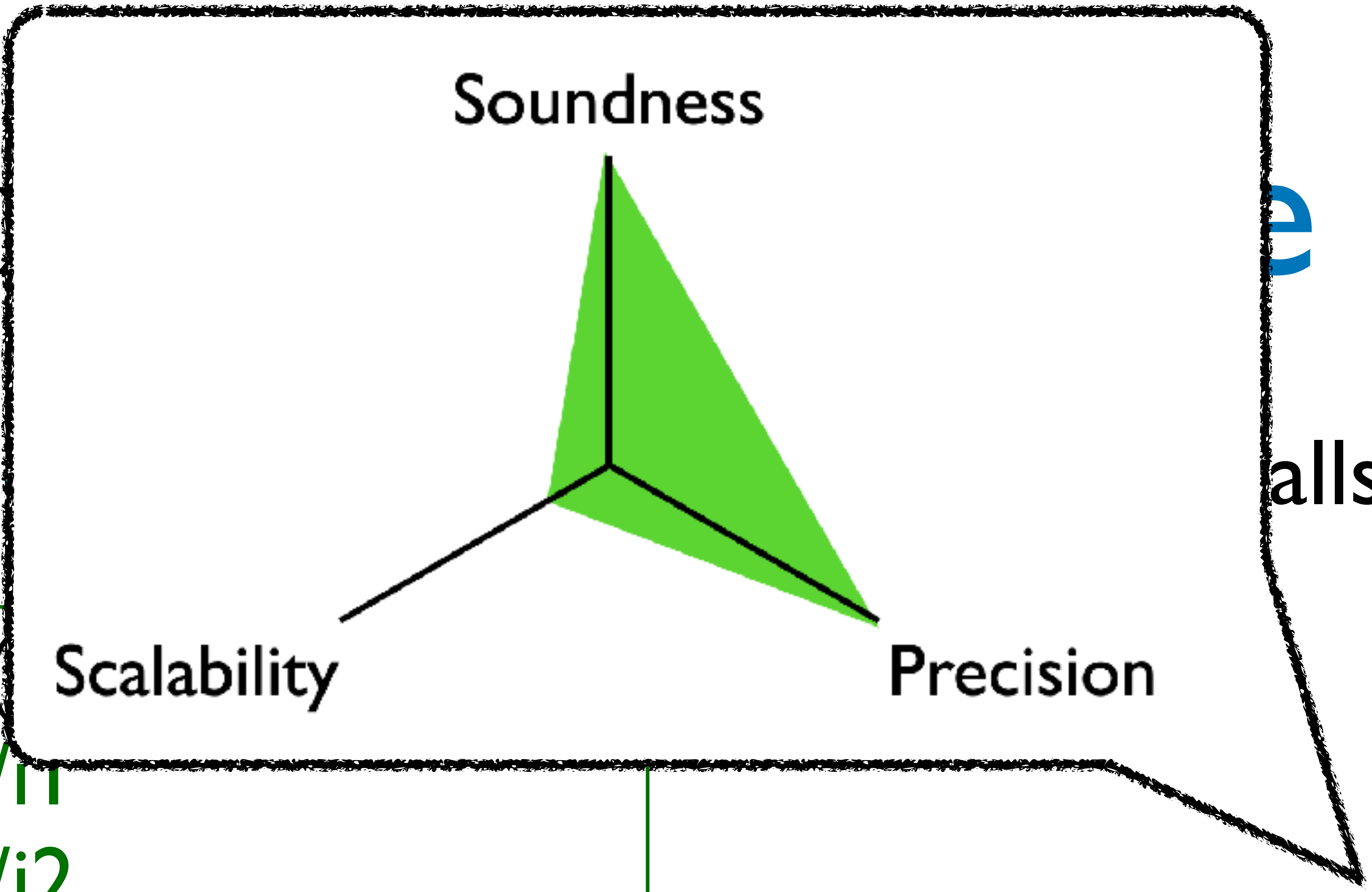


Apply 3-ctx : {h,g,f,main}

3-context sensitivity

E… Context Sensitivity

- **Pre**… alls makes the analysis precise but **expensive**

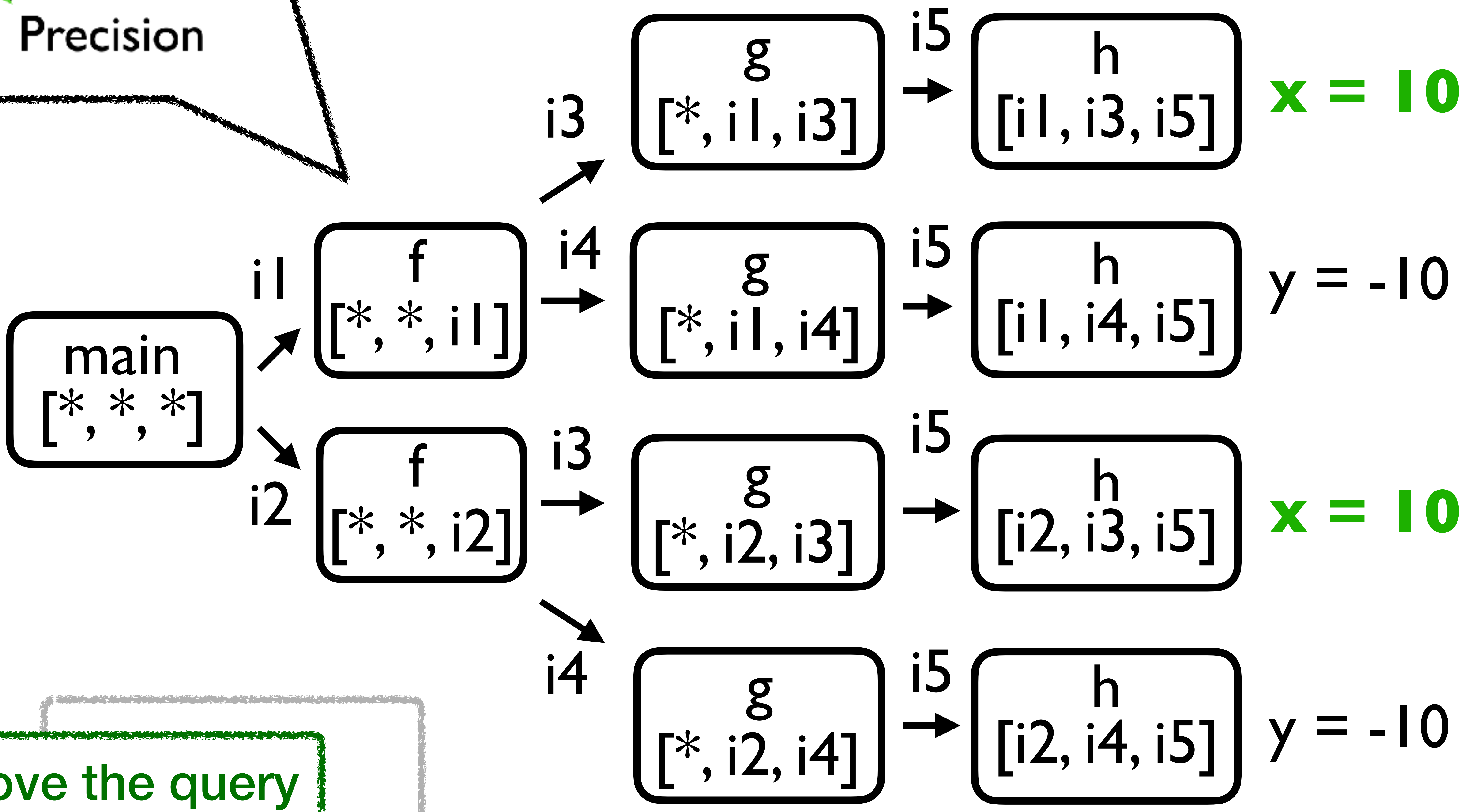



```
main(…
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

x = 10

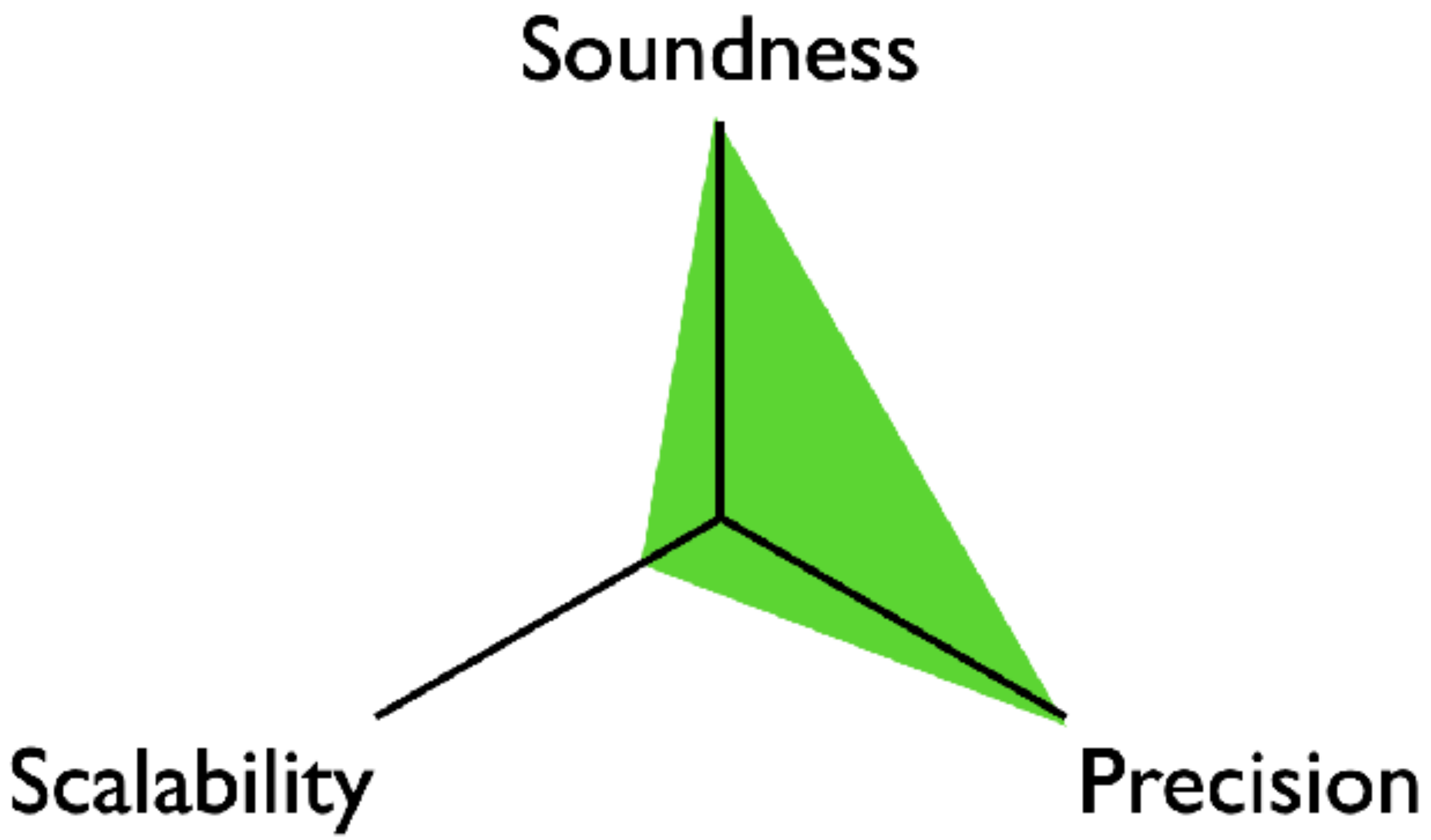
Can prove the query

Example program



3-context sensitivity

E… Context Sensitivity



Precisely analyzing all the methods is impractical

"Deep-context object-sensitive analyses are the most precise in practice, but do not always scale well."

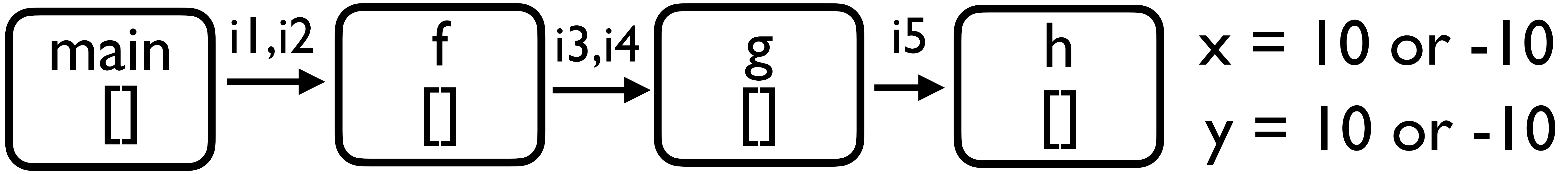
- Smaragdakis et al. [2014]

Example: Selective Context Sensitivity

- Coarsely analyzing all method calls is fast but makes the analysis **imprecise**

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

Example program

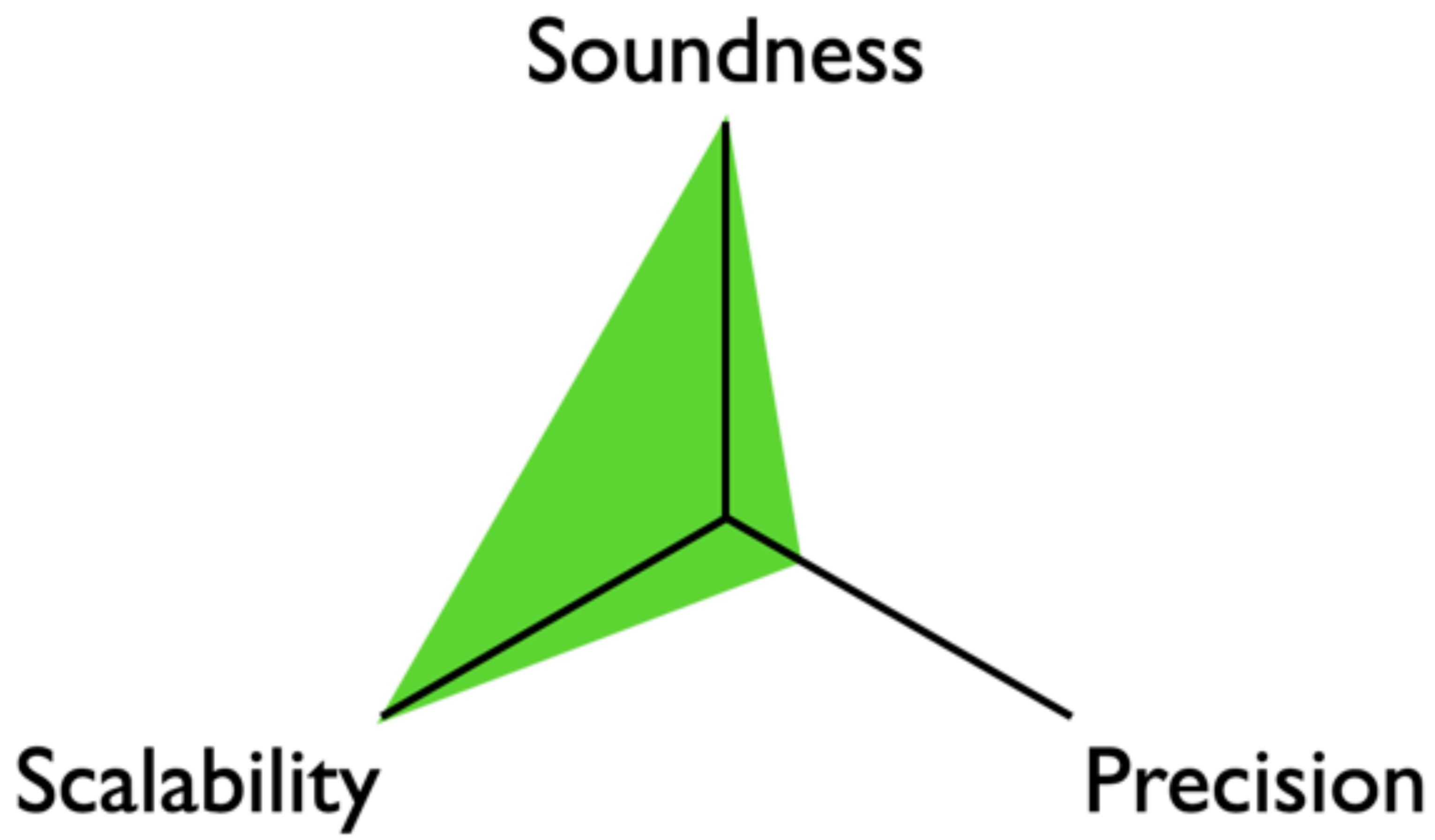


context-insensitive
(0-ctx sensitivity)

Apply 3-ctx : {}
Apply 2-ctx : {}
Apply 1-ctx : {}
Apply 0-ctx : {h,g,f,main}

Exampl ntext Sensitivity

- Coarsely analyz ut makes the analysis **imprecise**

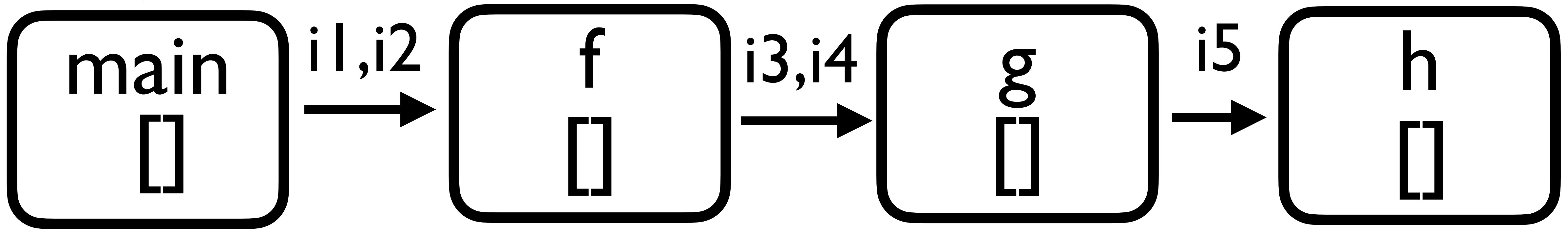



```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

x = 10 or -10

Example program

unable to prove the query



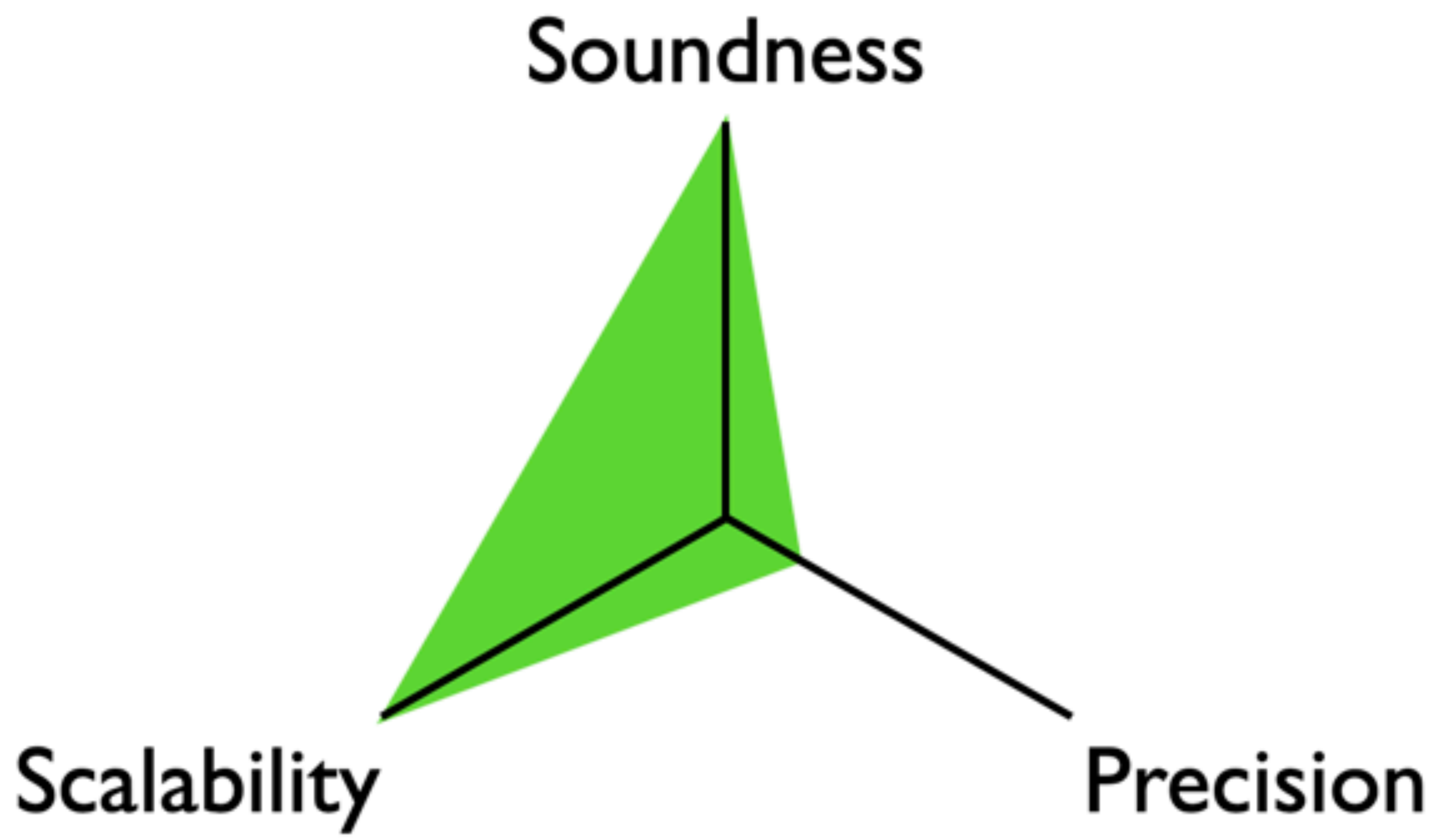
x = 10 or -10

y = 10 or -10

context-insensitive
(0-ctx sensitivity)

Apply 3-ctx : {}
Apply 2-ctx : {}
Apply 1-ctx : {}
Apply 0-ctx : {h,g,f,main}

Exampl... ntext Sensitivity



Coarsely analyzing all the methods is also impractical in practice

“for some applications,…precise context sensitivity is essential”

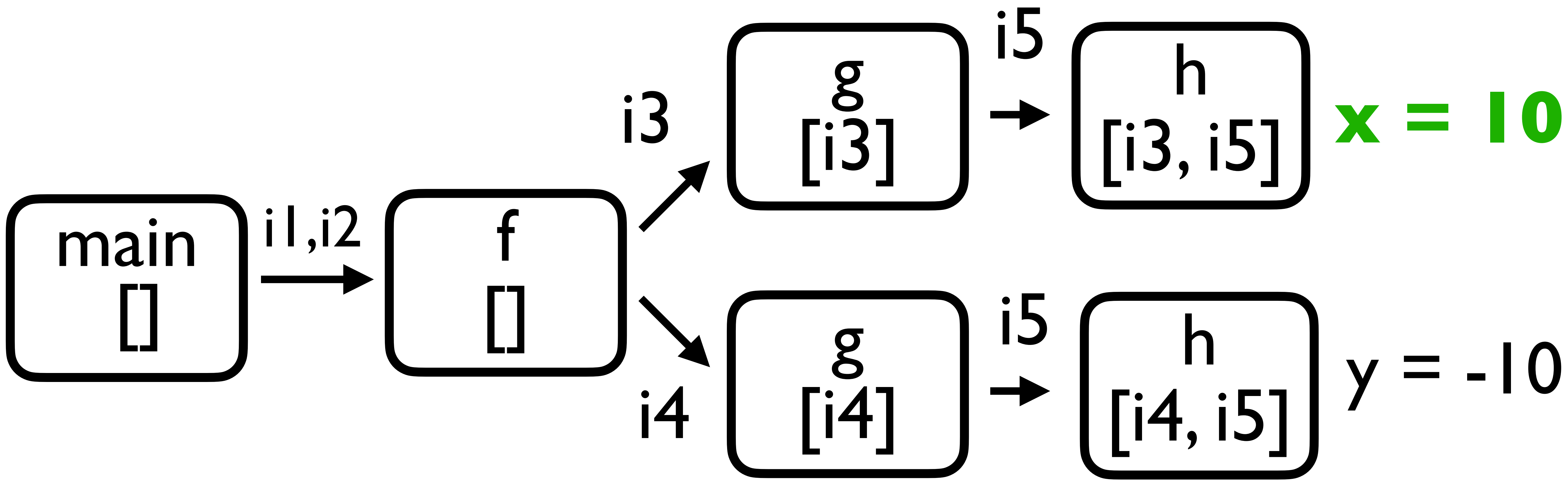
- Smaragdakis et al. [2014]

Example: Selective Context Sensitivity

- Selective context sensitivity can make the analysis fast and precise

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

Example program

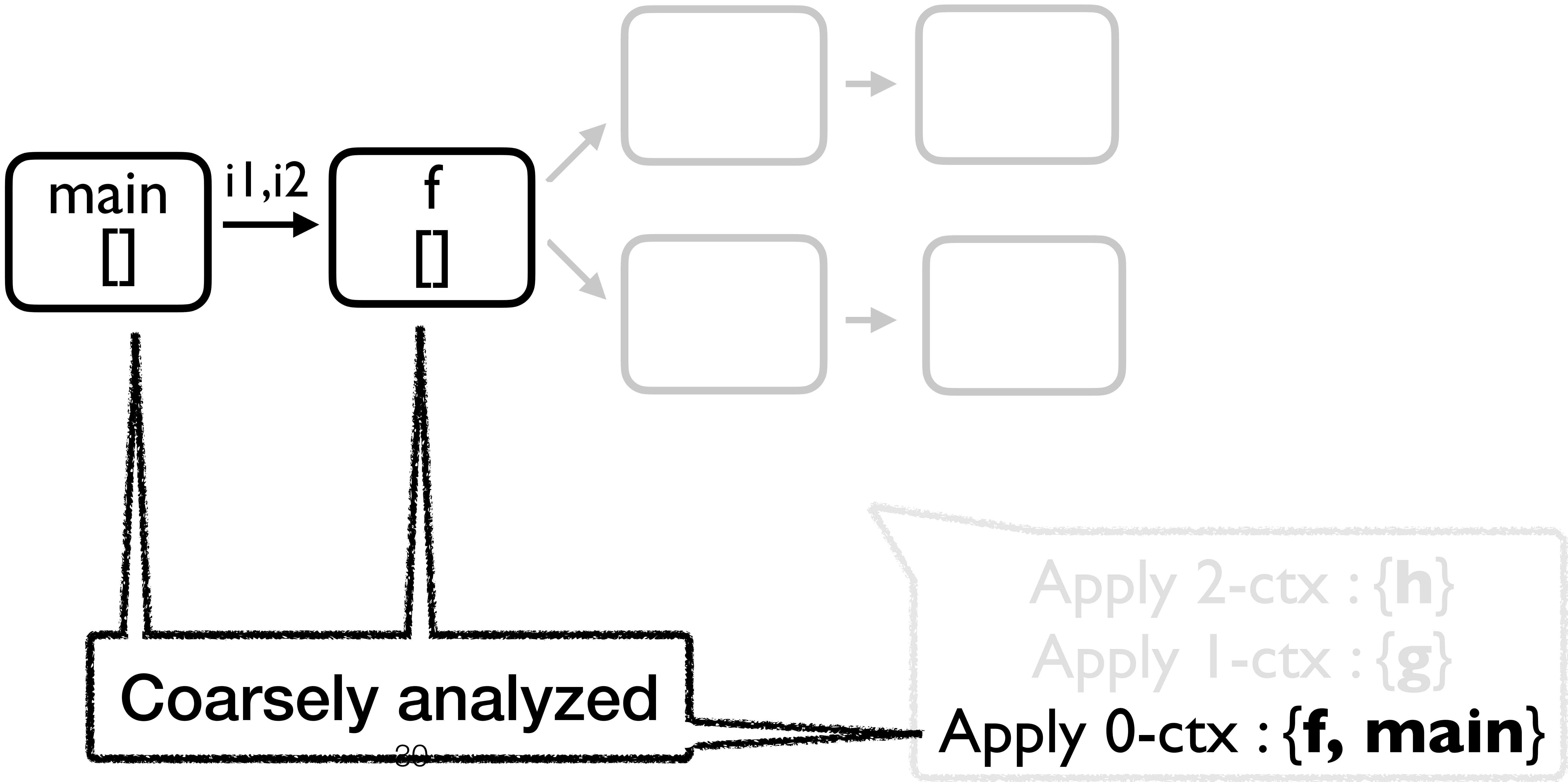


Selective context sensitivity

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

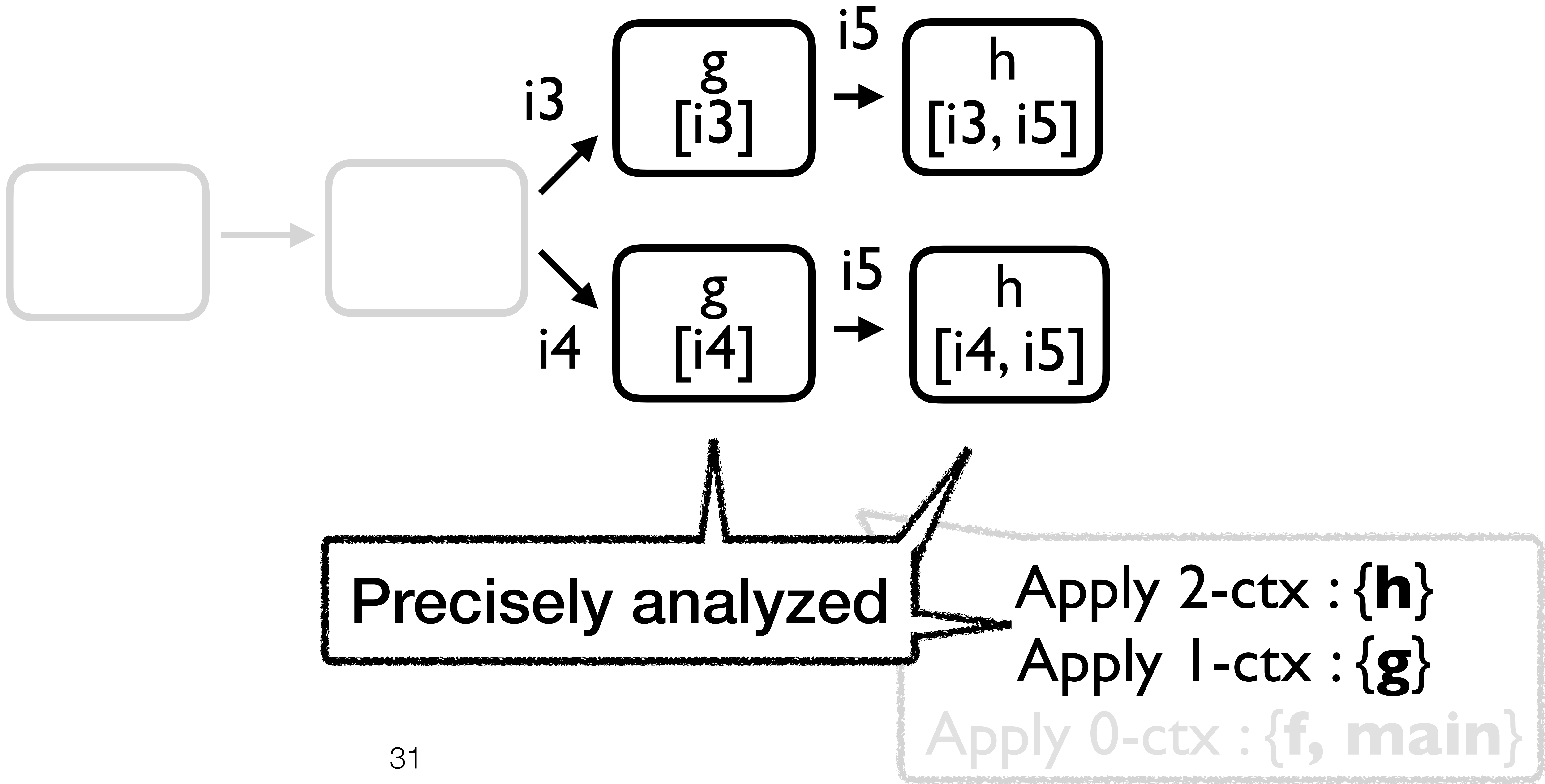
Example: Selective Context Sensitivity

- Selective context sensitivity can make the analysis fast and precise



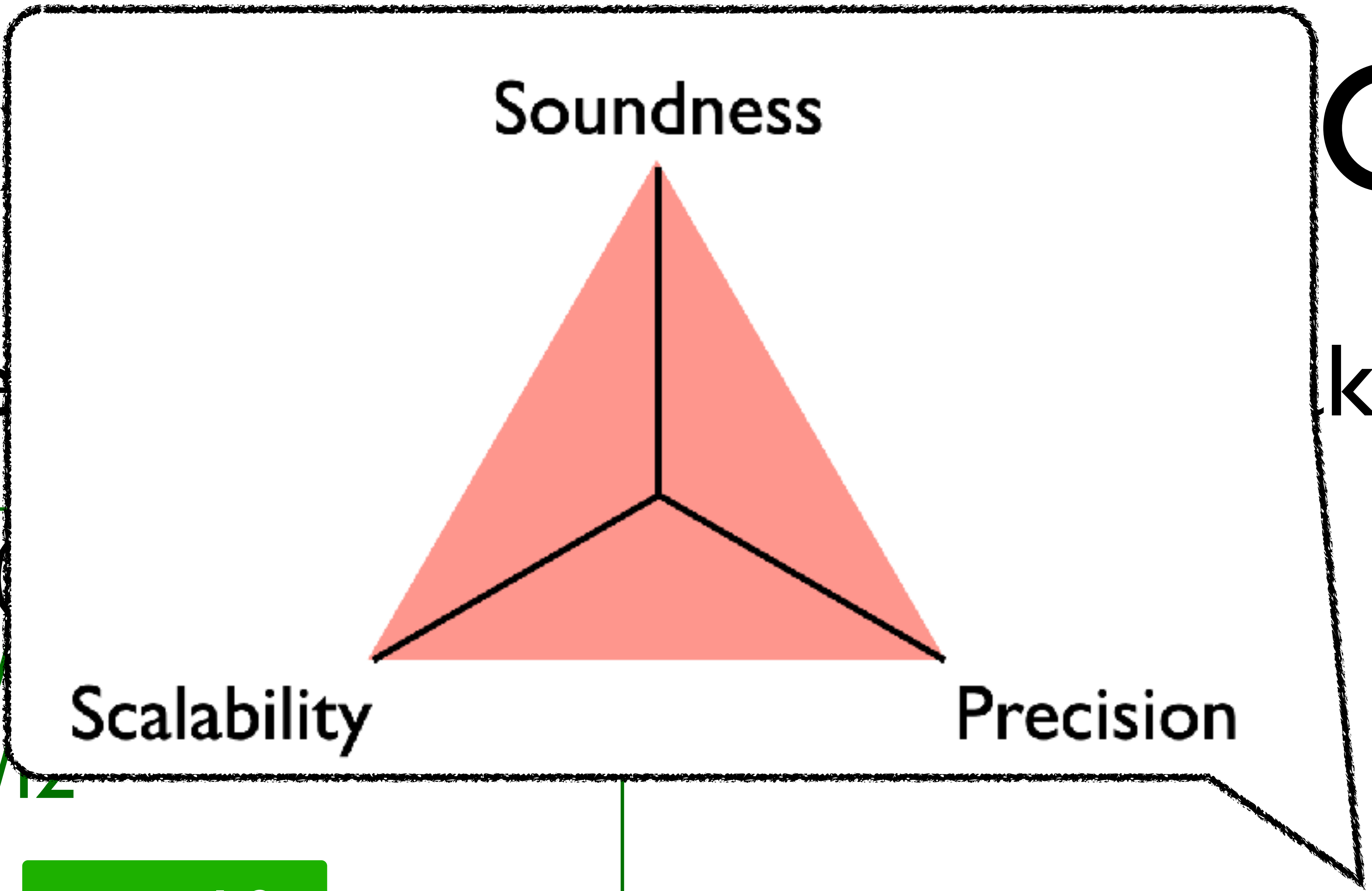
Example: Selective Context Sensitivity

- Selective context sensitivity can make the analysis fast and precise



E… Context Sensitivity

- Sele… ke the analysis fast and precise

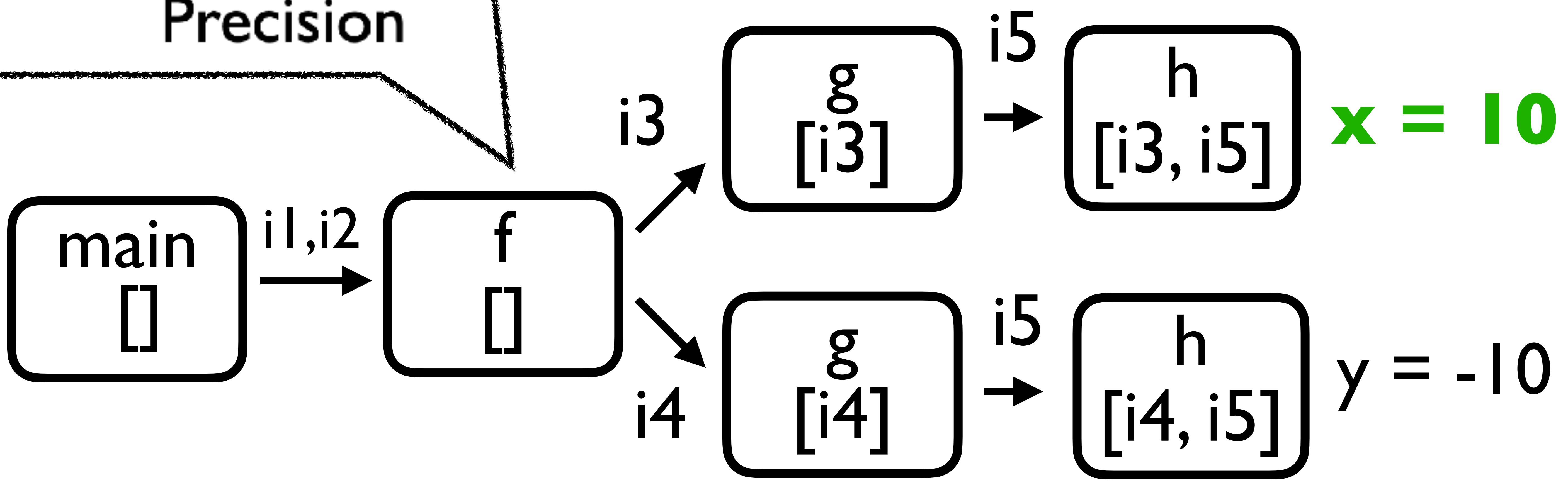



```
main(
  f();//
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

x = 10

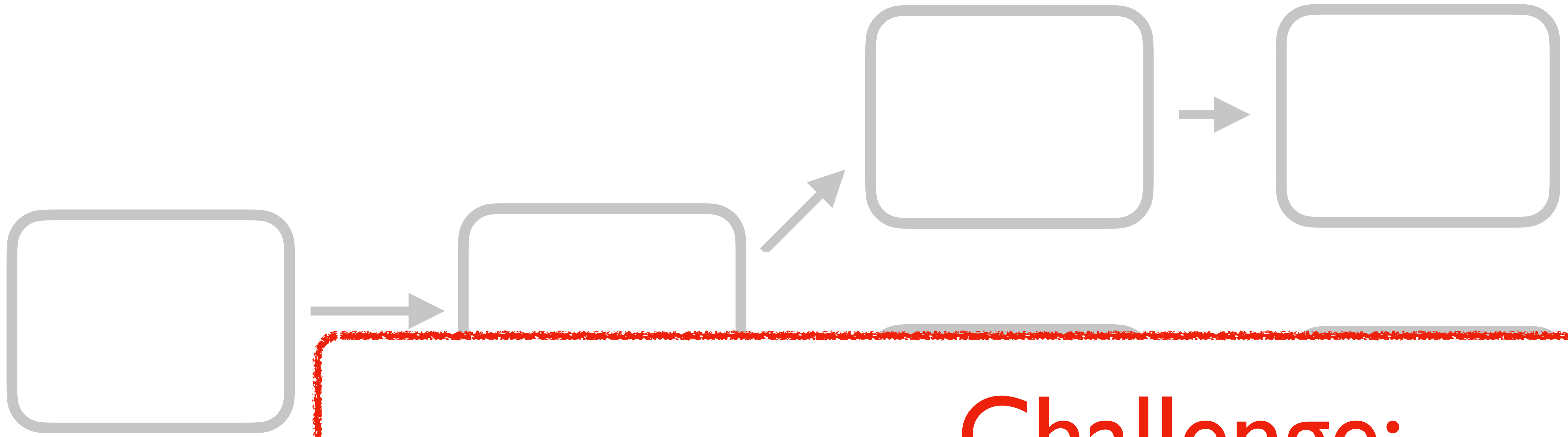
Can prove the query

Example program



Selective context sensitivity

Apply 2-ctx : {**h**}
Apply 1-ctx : {**g**}
Apply 0-ctx : {**f, main**}



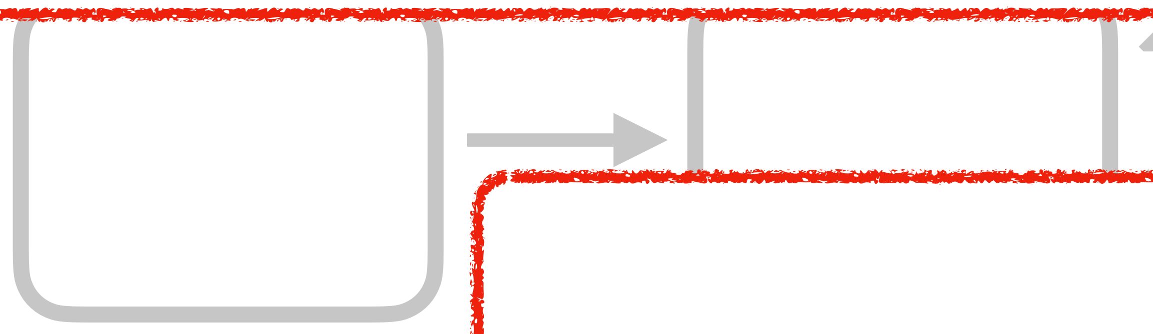
Challenge:
How can we find a good classification?

Apply 2-ctx : {**h**}
Apply 1-ctx : {**g**}
Apply 0-ctx : {**f, main**}

Hard search problem:

(1) Large search space (e.g., $(k+1)^{|Method|}$)

(2) There are few good classifications



Challenge:

How can we find a good classification?

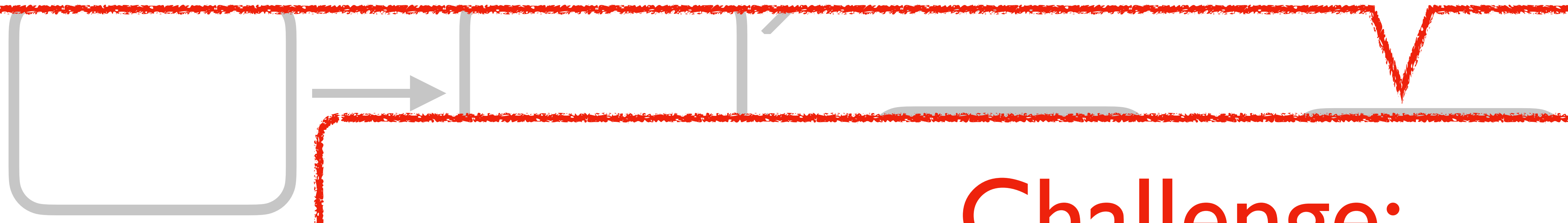
Apply 2-ctx : {**h**}
Apply 1-ctx : {**g**}
Apply 0-ctx : {**f, main**}

Many analysis heuristics have been proposed

Hard search problem:

(1) Large search space (e.g., $(k+1)^{|Method|}$)

(2) There are few good classifications



Challenge:

How can we find a good classification?

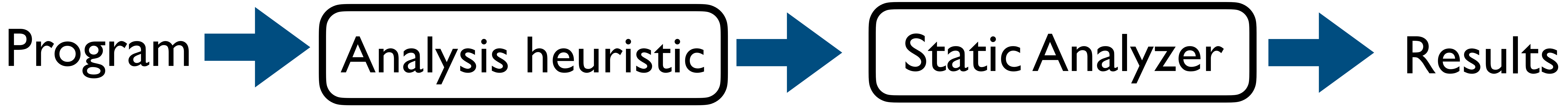
Apply 2-ctx : {**h**}

Apply 1-ctx : {**g**}

Apply 0-ctx : {**f, main**}

Static Analysis Needs Analysis Heuristics

- Modern static analyzers need analysis heuristics to become practical



Existing: manually designed analysis heuristics

- *"IFDS-based Context Debloating for Object-Sensitive Pointer Analysis"* [ASE 2023]
- *"Making Pointer Analysis More Precise by Unleashing the Power of Selective Context Sensitivity"* [OOPSLA 2021]
- *"Scalability-First Pointer Analysis with Self-Tuning Context-Sensitivity"* [FSE 2018]
- "Precision-Guided Context Sensitivity for Pointer Analysis" [OOPSLA 2018]
- *"Efficient and Precise Points-to Analysis: Modeling the Heap by Merging Equivalent Automata"* [PLDI 2017]
- …

Problem:

difficult, time-consuming, less effective

Existing: manually designed analysis heuristics

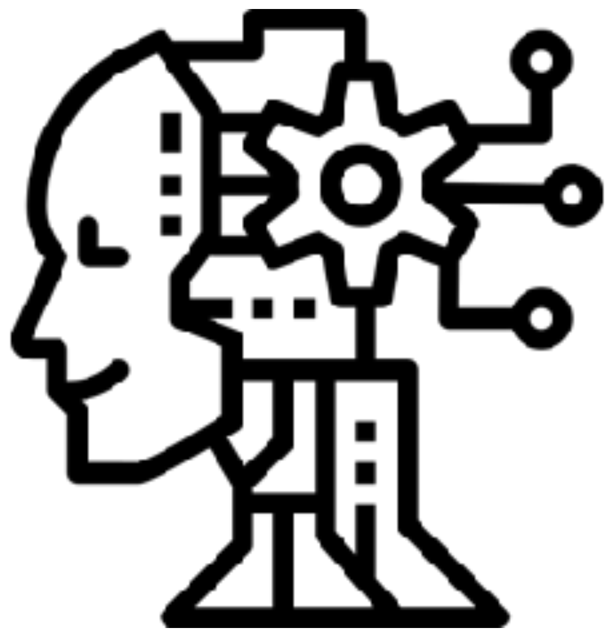
- *"IFDS-based Context Debloating for Object-Sensitive Pointer Analysis"* [ASE 2023]
- *"Making Pointer Analysis More Precise by Unleashing the Power of Selective Context Sensitivity"* [OOPSLA 2021]
- *"Scalability-First Pointer Analysis with Self-Tuning Context-Sensitivity"* [FSE 2018]
- "Precision-Guided Context Sensitivity for Pointer Analysis" [OOPSLA 2018]
- *"Efficient and Precise Points-to Analysis: Modeling the Heap by Merging Equivalent Automata"* [PLDI 2017]
- …

Data-Driven Static Analysis

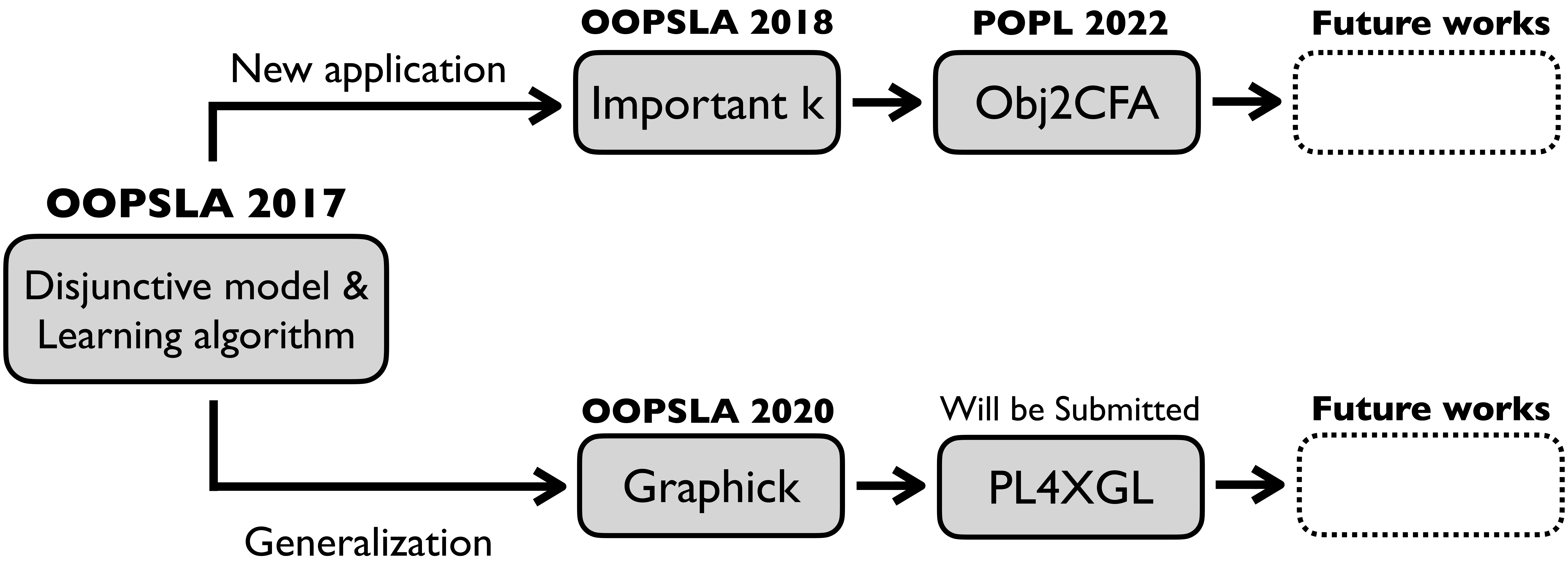
- Data-driven static analysis aims to generate heuristics automatically

Program → Analysis heuristic → Static Analyzer → Results

- **Automatically** generating **powerful** analysis heuristics



Learning algorithm + Data


OOPSLA 2017
Disjunctive model &
Learning algorithm
New application
OOPSLA 2018
Important k
POPL 2022
Obj2CFA
Future works
Generalization
OOPSLA 2020
Graphick
Will be Submitted
PL4XGL
Future works

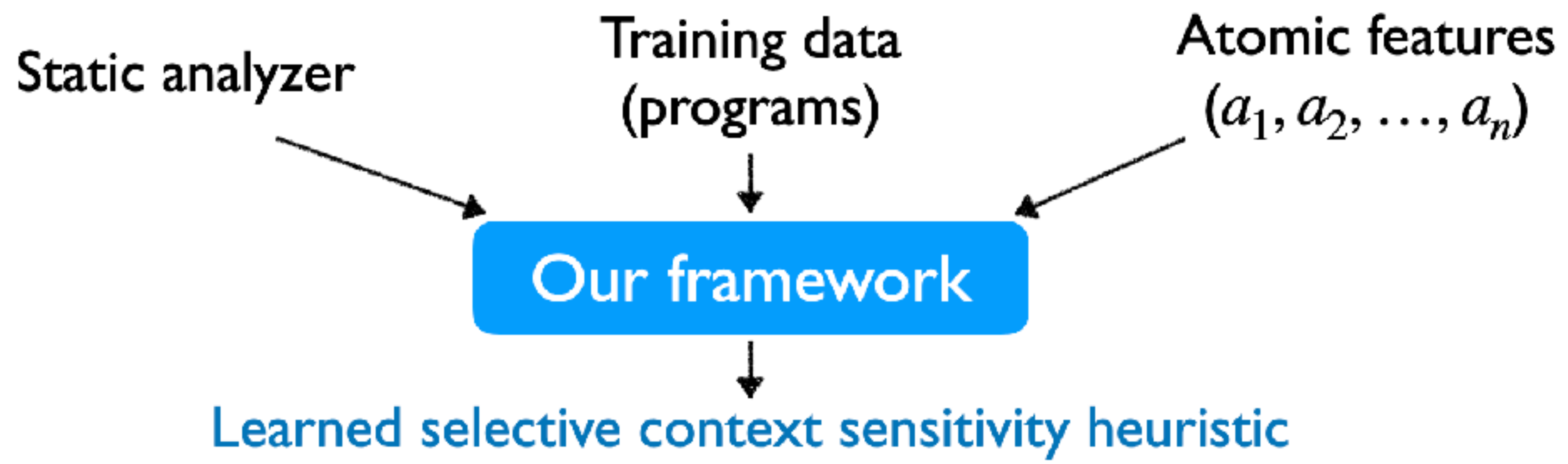
New applicati

works

OOPSLA 2017

Disjunctive model &
Learning algorithm

Designed a learning **framework**



$$f_{2ctx} = (a_1 \land \neg a_3 \land \neg a_6 \land a_8 \land \neg a_9 \land \neg a_{16} \land a_{17} \land a_{18} \land \ldots \land \neg a_{25})$$
$$f_{1ctx} = (a_1 \land \neg a_3 \land \neg a_4 \land \ldots) \lor (\neg a_3 \land \neg a_4 \land \neg a_7 \land \ldots) \lor \ldots$$

works

Generalization

Our Learning Framework

Static analyzer

Training data (programs)

Atomic features $(a_1, a_2, \ldots, a_n)$

Our framework

Learned selective context sensitivity heuristic

$$f_{2ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_6 \wedge a_8 \wedge \neg a_9 \wedge \neg a_{16} \wedge a_{17} \wedge a_{18} \wedge \ldots \wedge \neg a_{25})$$

$$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \ldots) \vee (\neg a_3 \wedge \neg a_4 \wedge \neg a_7 \wedge \ldots) \vee \ldots$$

Static analyzer

- Static analyzer is modeled as a blackbox function F :

$$F(p, a) \Rightarrow 2^{\mathbb{Q}} \times \mathbb{N}$$

program

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

Sta

classification

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

- is …kbox function F :

$$F(p, a) \Rightarrow 2^{\mathbb{Q}} \times \mathbb{N}$$

f_{2c}

f_{1ctx}

Static analyzer

- Static analyzer is modeled as a blackbox function F :

$$F(p, a) \Rightarrow 2^{\mathbb{Q}} \times \mathbb{N}$$

Queries proven to be safe

Analysis time

Small programs

Training data
(programs)

$$f_{2ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_6 \wedge a_8 \wedge \neg a_9 \wedge \neg a_{16} \wedge a_{17} \wedge a_{18} \wedge \ldots \wedge \neg a_{25})$$

$$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \ldots) \vee (\neg a_3 \wedge \neg a_4 \wedge \neg a_7 \wedge \ldots) \vee \ldots$$

Atomic features

$(a_1, a_2, \ldots, a_n)$

25 predicates on methods
(e.g., has if statement?,
takes void input?,…)

$$f_{2ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_6 \wedge a_8 \wedge \neg a_9 \wedge \neg a_{16} \wedge a_{17} \wedge a_{18} \wedge \ldots \wedge \neg a_{25})$$

$$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \ldots) \vee (\neg a_3 \wedge \neg a_4 \wedge \neg a_7 \wedge \ldots) \vee \ldots$$

Find a heuristic (classifier) $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

- minimizes analysis cost while is precise enough

Our framework

$$f_{2ctx} = (a_1 \land \neg a_3 \land \neg a_6 \land a_8 \land \neg a_9 \land \neg a_{16} \land a_{17} \land a_{18} \land \ldots \land \neg a_{25})$$

$$f_{1ctx} = (a_1 \land \neg a_3 \land \neg a_4 \land \ldots) \lor (\neg a_3 \land \neg a_4 \land \neg a_7 \land \ldots) \lor \ldots$$

Find a heuristic (classifier) $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

How a heuristic $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ works

Methods

f : $\{a_1, a_2\}$

h : $\{a_1\}$

g : $\{a_2\}$

m : $\{\}$

Disjunctive heuristic

$$f_{2ctx} = (a_1 \wedge a_2) \vee (\neg a_1 \wedge \neg a_2)$$

$$f_{1ctx} = (a_1 \wedge \neg a_2)$$

Classification

2-ctx: {f, m}

1-ctx: {h}

0-ctx: {g}

$$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \ldots) \vee (\neg a_3 \wedge \neg a_4 \wedge \neg a_7 \wedge \ldots) \vee \ldots$$

Find a heuristic (classifier) $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

How a heuristic $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ works

Methods

f : $\{a_1, a_2\}$

h : $\{a_1\}$

g : $\{a_2\}$

m : $\{\}$

$f_{2ctx} = (a_1 \wedge a_2) \vee (\neg a_1 \wedge \neg a_2)$

$f_{1ctx} = (a_1 \wedge \neg a_2)$

$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \ldots) \vee (\neg a_3 \wedge \neg a_4 \wedge \neg a_7 \wedge \ldots) \vee \ldots$

Find a heuristic (classifier) $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

How a heuristic $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ works

Methods

f : $\{a_1, a_2\}$

h : $\{a_1\}$

g : $\{a_2\}$

m : $\{\}$

Disjunctive heuristic

$$f_{2ctx} = (a_1 \wedge a_2) \vee (\neg a_1 \wedge \neg a_2)$$

$$f_{1ctx} = (a_1 \wedge \neg a_2)$$

Classification

2-ctx: {f, m}

$$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \ldots) \vee (\neg a_3 \wedge \neg a_4 \wedge \neg a_7 \wedge \ldots) \vee \ldots$$

Find a heuristic (classifier) $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

How a heuristic $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ works

Methods

f : $\{a_1, a_2\}$

h : $\{a_1\}$

g : $\{a_2\}$

m : {}

Disjunctive heuristic

$f_{2ctx} = (a_1 \wedge a_2) \vee (\neg a_1 \wedge \neg a_2)$

$f_{1ctx} = (a_1 \wedge \neg a_2)$

Classification

1-ctx: {h}

$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \ldots) \vee (\neg a_3 \wedge \neg a_4 \wedge \neg a_7 \wedge \ldots) \vee \ldots$

Find a heuristic (classifier) $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

How a heuristic $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ works

Methods

f : $\{a_1, a_2\}$

h : $\{a_1\}$

g : $\{a_2\}$

m : $\{\}$

Disjunctive heuristic

$$f_{2ctx} = (a_1 \wedge a_2) \vee (\neg a_1 \wedge \neg a_2)$$
$$f_{1ctx} = (a_1 \wedge \neg a_2)$$

Classification

2-ctx: {f, m}

1-ctx: {h}

0-ctx: {g}

$$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \ldots) \vee (\neg a_3 \wedge \neg a_4 \wedge \neg a_7 \wedge \ldots) \vee \ldots$$

Find a heuristic (classifier) $\mathcal{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

- minimizes analysis cost while is precise enough

- User-provided precision constraint
- E.g., maintain 90% precision of the fully 2-ctx sensitivity for the training set

$$\frac{\text{\# queries proved by the current heuristic } \mathcal{H}}{\text{\# queries proved by the fully 2-ctx sensitivity}} > 0.9$$

Classifies all the methods into 2-ctx

$f_{2ctx} = (a_1 \land \neg a_3 \land \neg a_6 \land a_8 \land \ldots \neg a_{25})$

$f_{1ctx} = (a_1 \land \neg a_3 \land \neg a_4 \land \ldots) \lor (\neg a_3 \land \neg a_4 \land \neg a_7 \land \ldots) \lor \ldots$

Find a heuristic (classifier) $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

We learn each formula via greedy refinement

1. Initialize f to be the most general DNF formula

$$f = a_1 \vee \neg a_1 \vee a_2 \vee \neg a_2 \vee \ldots \vee a_n \vee \neg a_n \ (\equiv true)$$

2. Repeat the following until no refinement is possible

$$f = c_1 \vee c_2 \vee \ldots, \vee c_n$$

   1. Choose a conjunction, say c_i
   2. Refine the conjunction with a feature a_j

$$f = c_1 \vee c_2 \vee \ldots, \vee (c_i \wedge a_j) \vee c_n$$

   3. Check the precision constraint: If not, revert the last change.

(details in our paper)

Find a heuristic (classifier) $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

We learn each formula via greedy refinement

1. Initialize f to be the most general DNF formula

$$f = a_1 \vee \neg a_1 \vee a_2 \vee \neg a_2 \vee \ldots \vee a_n \vee \neg a_n \ (\equiv true)$$

$$f = c_1 \vee c_2 \vee \ldots, \vee c_n$$

$$f = c_1 \vee c_2 \vee \ldots, \vee (c_i \wedge a_j) \vee c_n$$

Find a heuristic (classifier) $\mathcal{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

$$f = a_1 \vee \neg a_1 \vee a_2 \vee \neg a_2 \vee \ldots \vee a_n \vee \neg a_n \;(\equiv true)$$

2. Repeat the following until no refinement is possible

$$f = c_1 \vee c_2 \vee \ldots, \vee c_n$$

1. Choose a conjunction, say c_i
2. Refine the conjunction with a feature a_j

$$f = c_1 \vee c_2 \vee \ldots, \vee (c_i \wedge a_j) \vee c_n$$

3. Check the precision constraint: If not, revert the last change.

(details in our paper)

Our framework

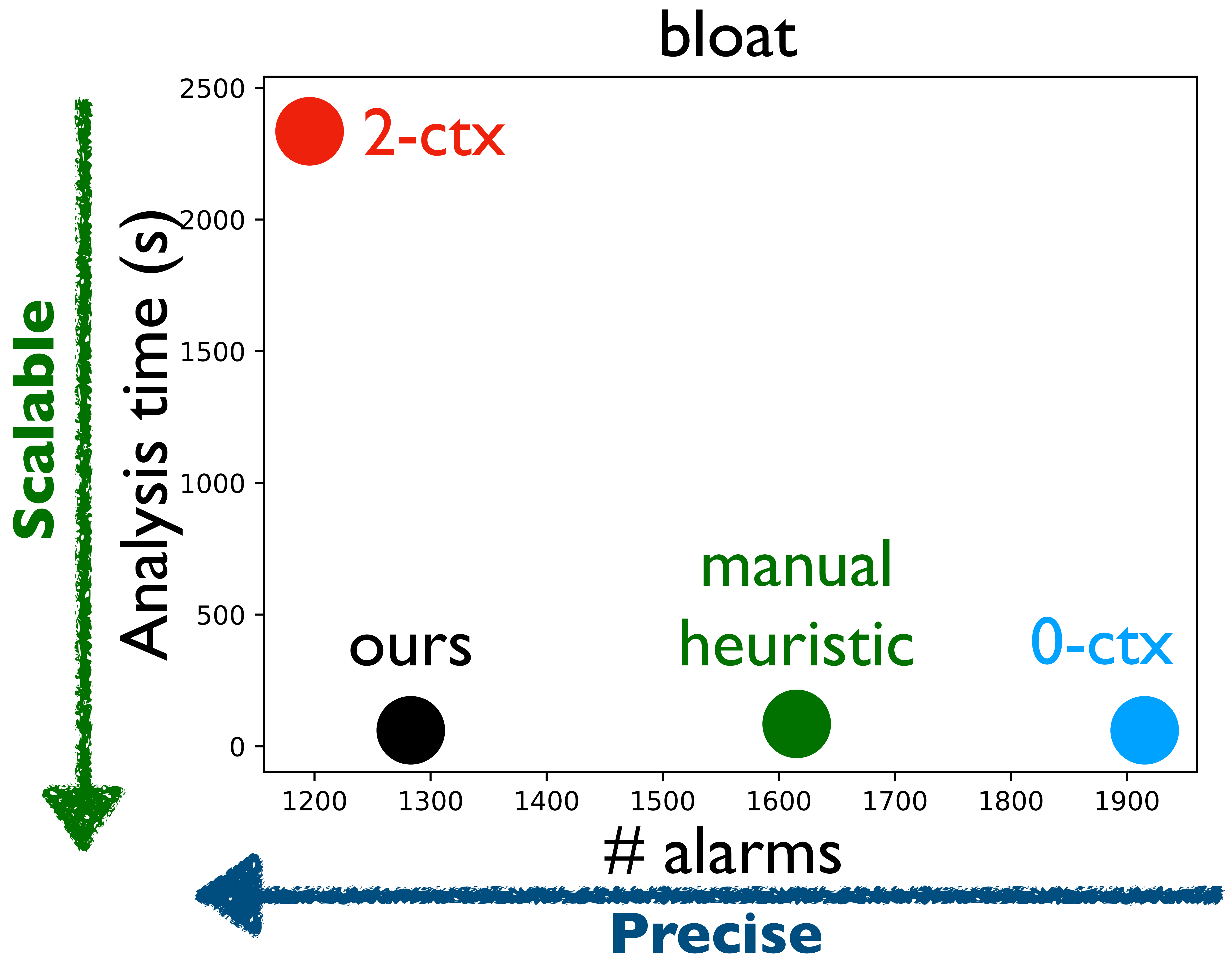
Learned selective context sensitivity heuristic

$$f_{2ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_6 \wedge a_8 \wedge \neg a_9 \wedge \neg a_{16} \wedge a_{17} \wedge a_{18} \wedge \ldots \wedge \neg a_{25})$$

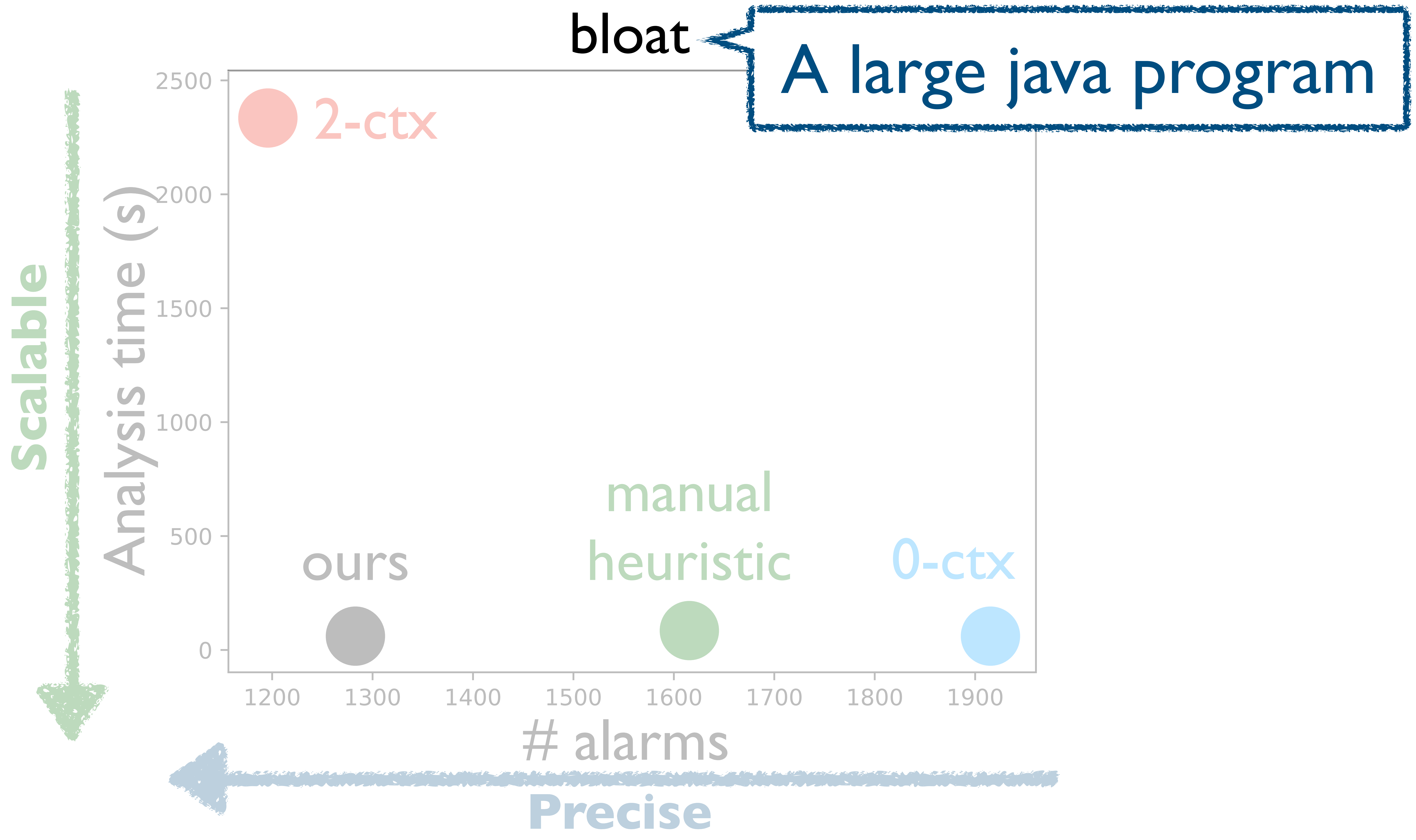
$$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \ldots) \vee (\neg a_3 \wedge \neg a_4 \wedge \neg a_7 \wedge \ldots) \vee \ldots$$

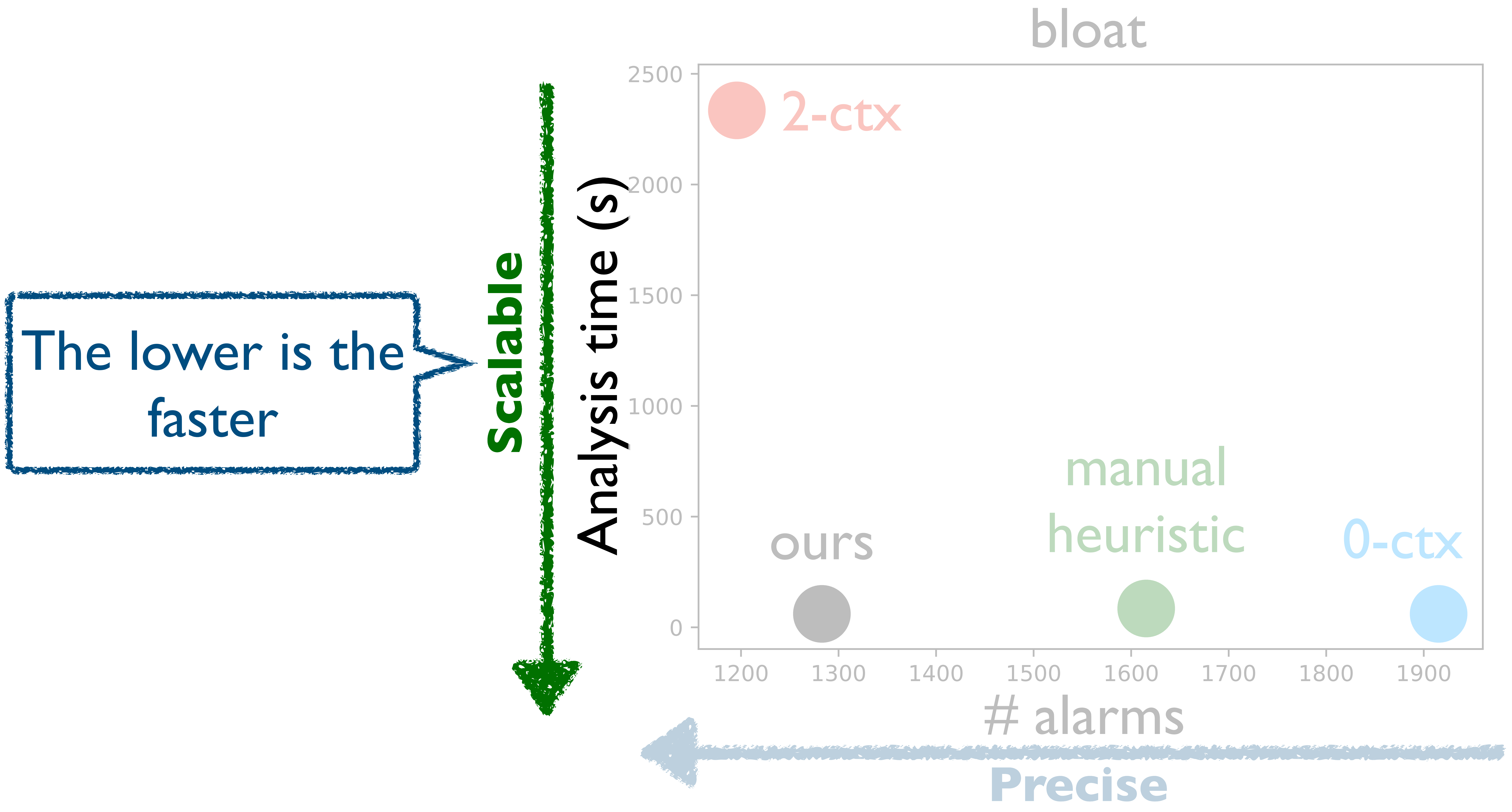
Performance Highlight of Our Learned Heuristic

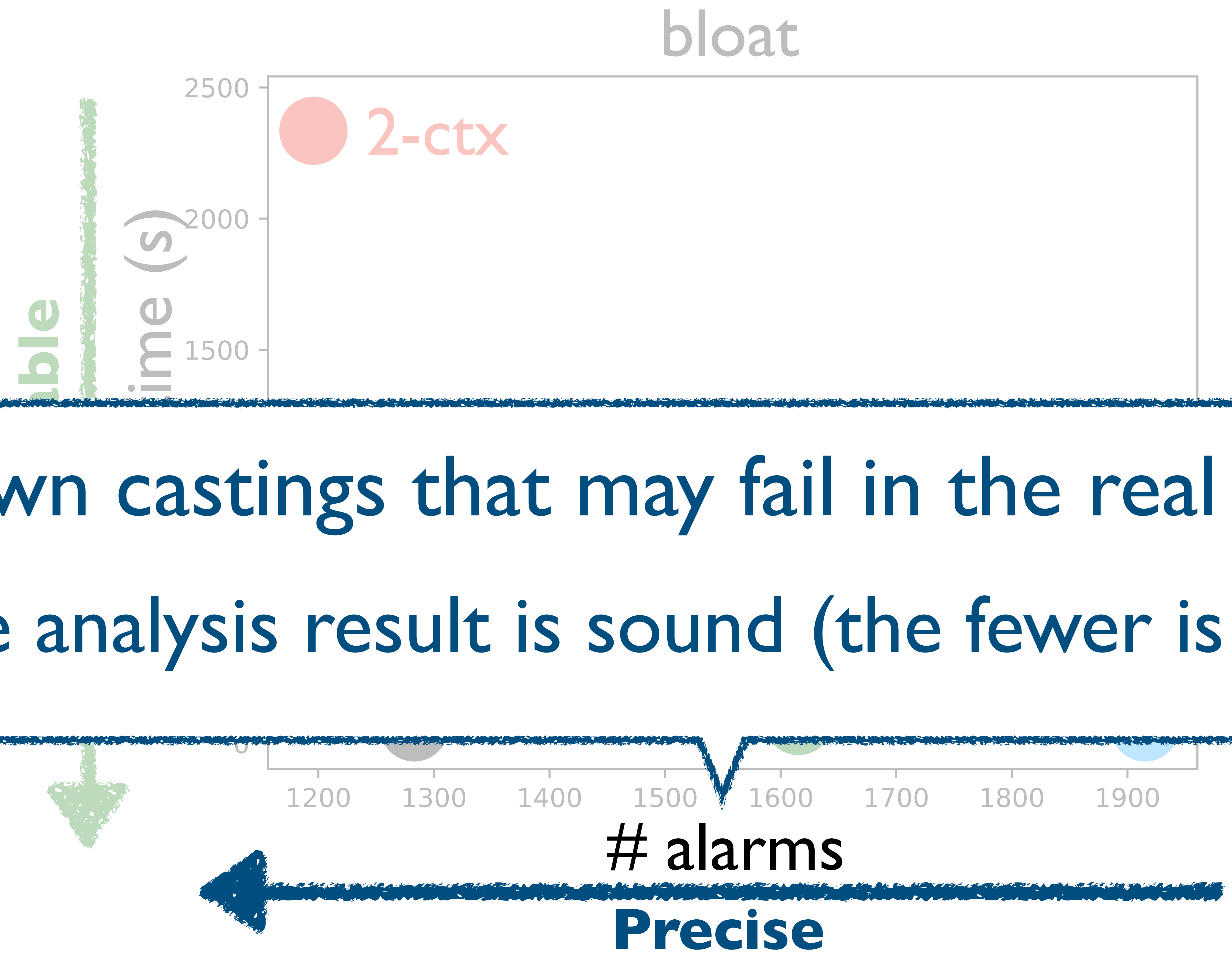
- Implemented in Doop, a state-of-the-art pointer analyzer for Java
- Trained with 4 small programs and evaluates with 6 large programs



- Trained with 4 small programs and evaluates with 6 large programs

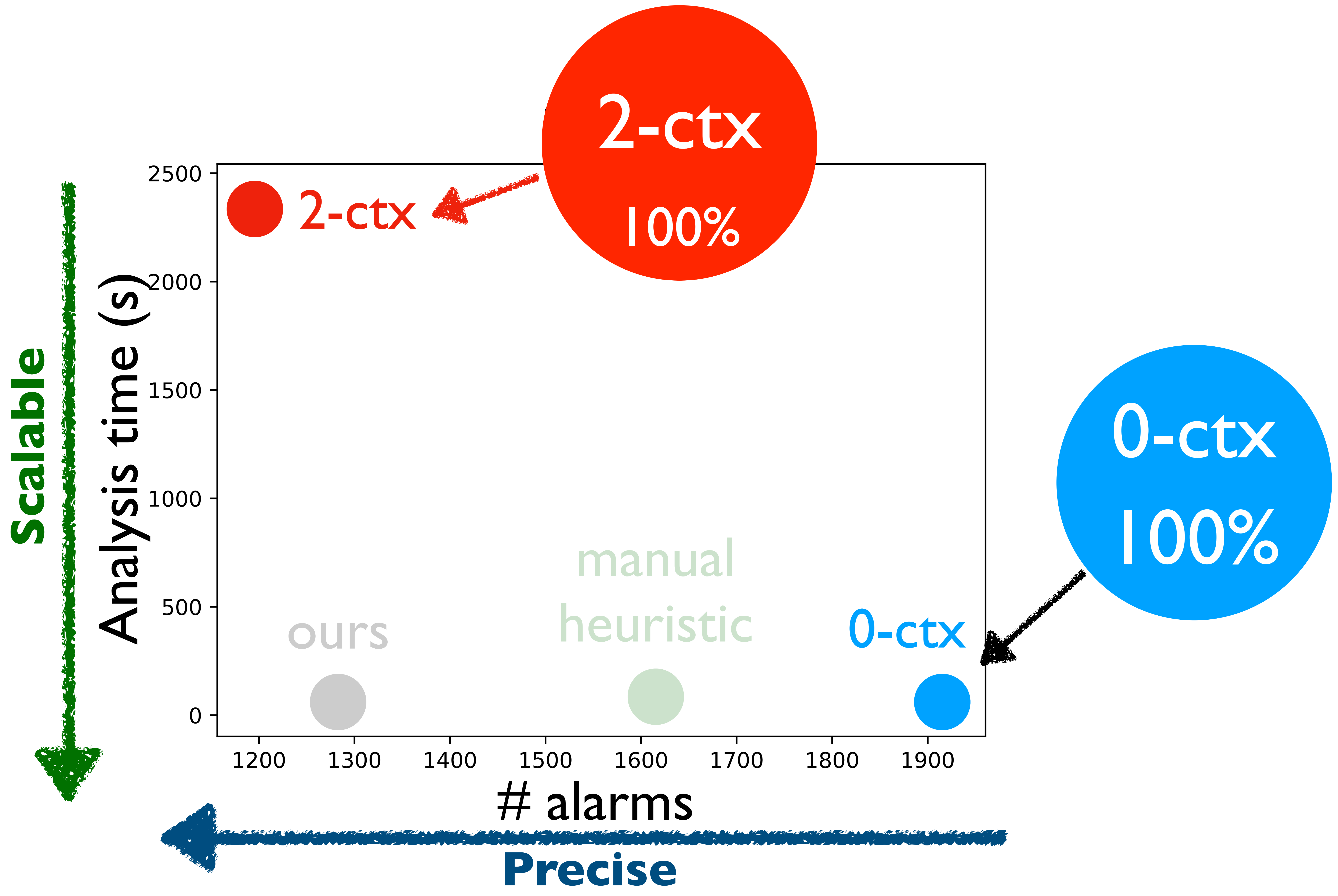



The lower is the faster
Scalable
Analysis time (s)
bloat
2500
2000
1500
1000
500
0
2-ctx
manual heuristic
ours
0-ctx
1200
1300
1400
1500
1600
1700
1800
1900
alarms
Precise

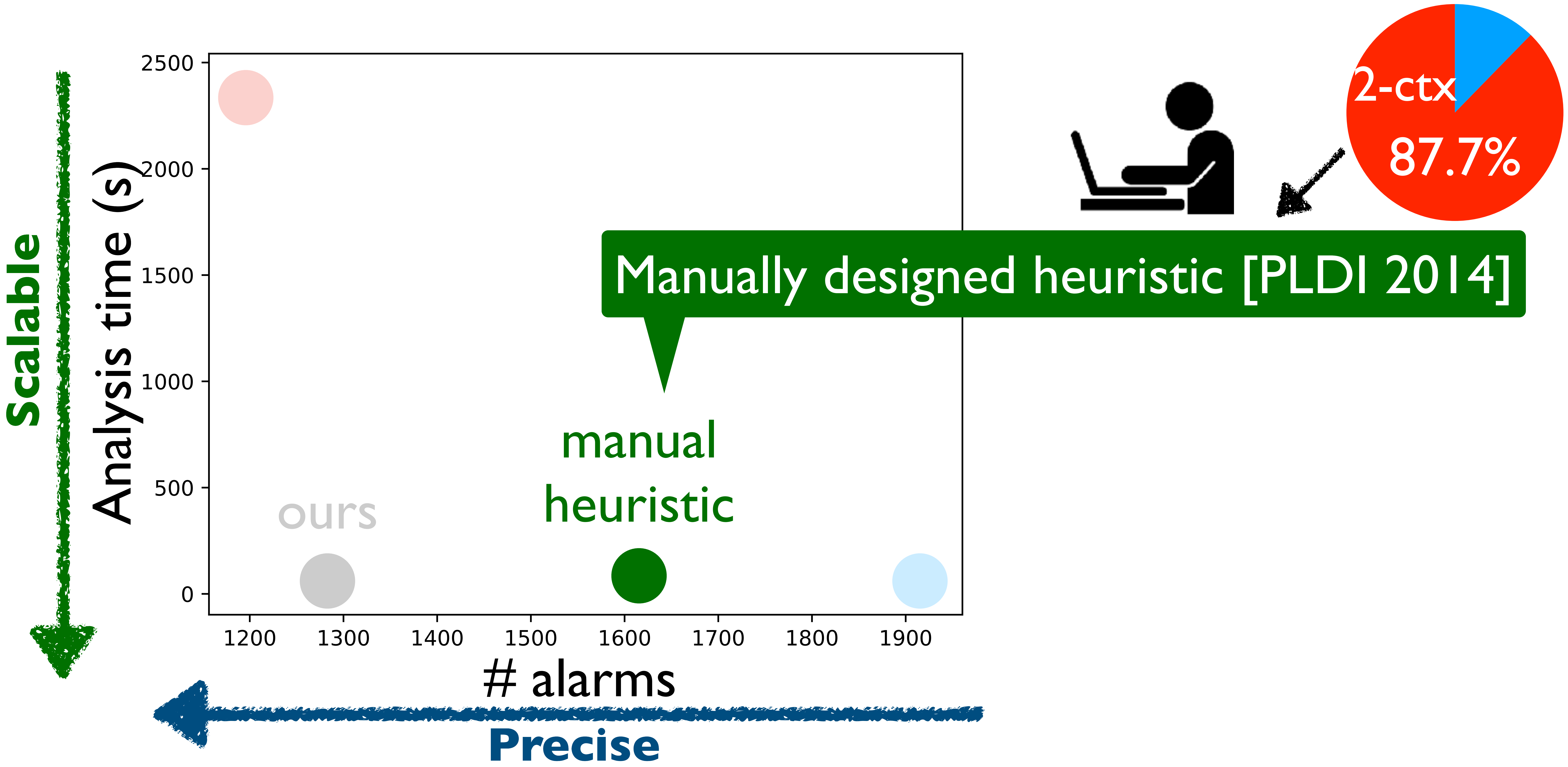


#down castings that may fail in the real execution

- The analysis result is sound (the fewer is the better)



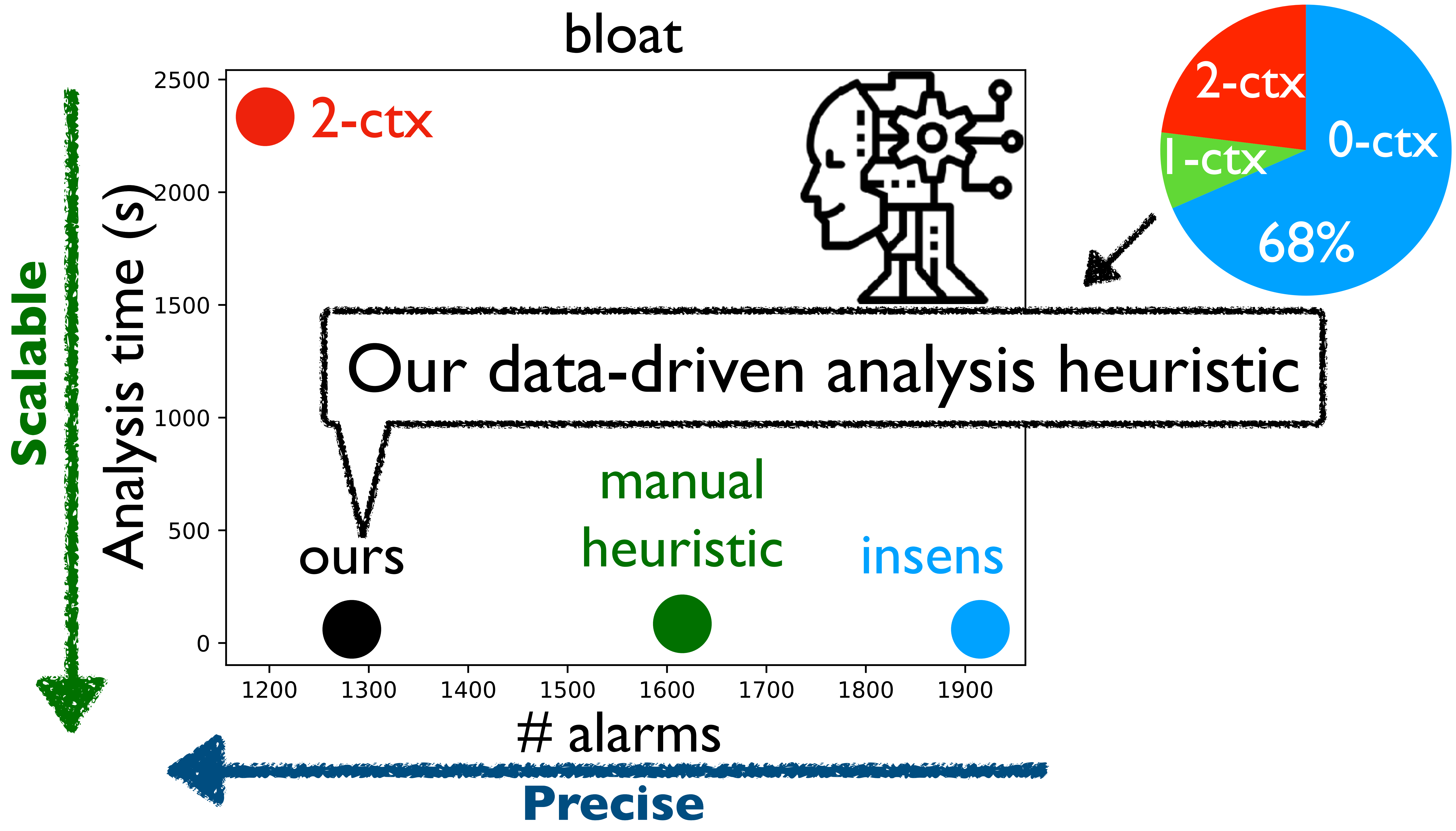
2-ctx
100%
0-ctx
100%
2500
2000
1500
1000
500
0
2-ctx
ours
manual
heuristic
0-ctx
1200
1300
1400
1500
1600
1700
1800
1900
Analysis time (s)
alarms
Scalable
Precise

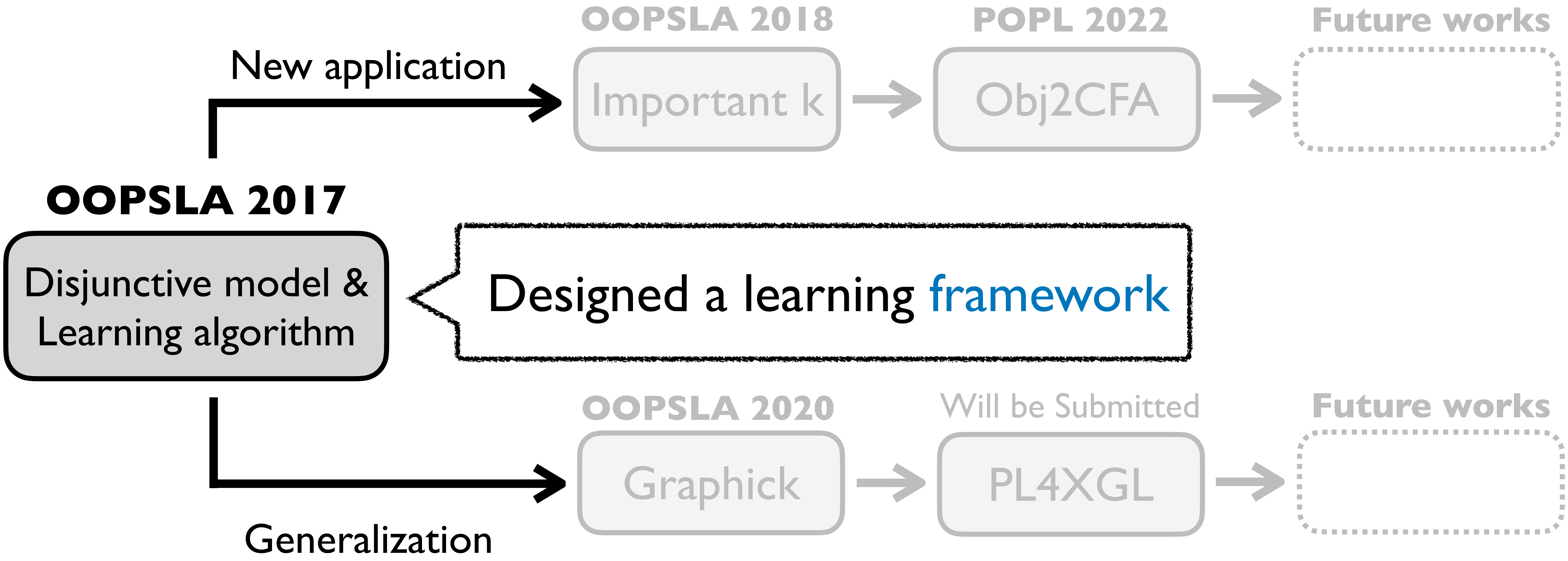


Scalable
Analysis time (s)
2500
2000
1500
1000
500
0
1200
1300
1400
1500
1600
1700
1800
1900
alarms
Precise
ours
manual
heuristic
Manually designed heuristic [PLDI 2014]
2-ctx
87.7%

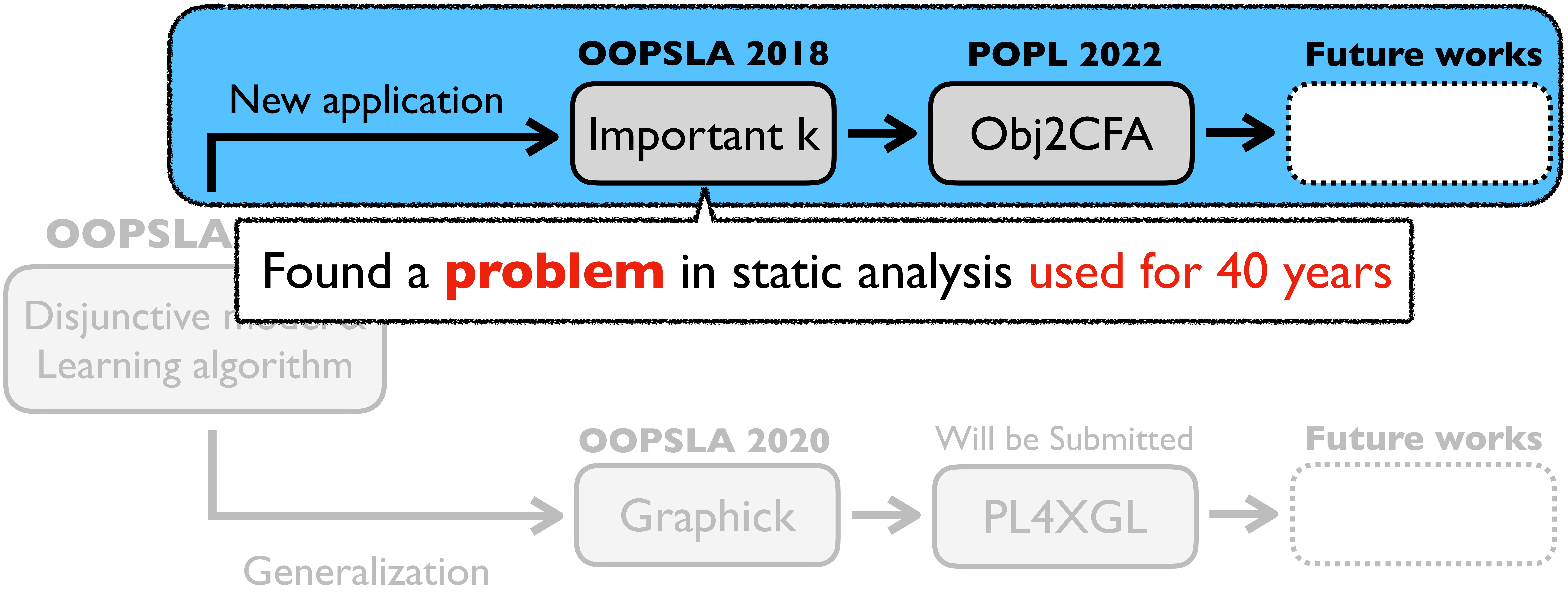
Performance Highlight of Our Learned Heuristic

- Implemented in Doop, a state-of-the-art pointer analyzer for Java
- Trained with 4 small programs and evaluates with 6 large programs





OOPSLA 2017
Disjunctive model & Learning algorithm
New application
OOPSLA 2018
Important k
POPL 2022
Obj2CFA
Future works
Designed a learning framework
Generalization
OOPSLA 2020
Graphick
Will be Submitted
PL4XGL
Future works


OOPSLA 2018
POPL 2022
Future works
New application
Important k
Obj2CFA
Found a problem in static analysis used for 40 years
OOPSLA
Disjunctive
Learning algorithm
OOPSLA 2020
Will be Submitted
Future works
Graphick
PL4XGL
Generalization

A Key Limiting Factor in Static Analysis

- Suppose you gave an assignment to summarize a paper with three sentences

in the paper

Precise and Scalable Points-to Analysis via Data-Driven Context Tunneling

MINSEOK JEON, Korea University, Republic of Korea
SEHUN JEONG, Korea University, Republic of Korea
HAKJOO OH*, Korea University, Republic of Korea

We present context tunneling, a new approach for making k-limited context-sensitive points-to analysis precise and scalable. As context-sensitivity holds the key to the development of precise and scalable points-to analysis, a variety of techniques for context-sensitivity have been proposed. However, existing approaches such as k-call-site-sensitivity or k-object-sensitivity have a significant weakness that they unconditionally update the context of a method at every call-site, allowing important context elements to be overwritten by more recent, but not necessarily more important, context elements. In this paper, we show that this is a key limiting factor of existing context-sensitive analyses, and demonstrate that remarkable increase in both precision and scalability can be gained by maintaining important context elements only. Our approach, called context tunneling, updates contexts selectively and decides when to propagate the same context without modification.

We attain context tunneling via a data-driven approach. The effectiveness of context tunneling is very sensitive to the choice of important context elements. Even worse, precision is not monotonically increasing with respect to the ordering of the choices. As a result, manually coming up with a good heuristic rule for context tunneling is extremely challenging and likely fails to maximize its potential. We address this challenge by developing a specialized data-driven algorithm, which is able to automatically search for high-quality heuristics over the non-monotonic space of context tunneling.

We implemented our approach in the Doop framework and applied it to four major flavors of context-sensitivity: call-site-sensitivity, object-sensitivity, type-sensitivity, and hybrid context-sensitivity. In all cases, 1-context-sensitive analysis with context tunneling far outperformed deeper context-sensitivity with $k = 2$ in both precision and scalability.

CCS Concepts: • **Theory of computation → Program analysis**; • **Computing methodologies → Machine learning approaches**;

Additional Key Words and Phrases: Points-to analysis, Context-sensitive analysis, Data-driven program analysis

ACM Reference Format:
Minseok Jeon, Sehun Jeong, and Hakjoo Oh. 2018. Precise and Scalable Points-to Analysis via Data-Driven Context Tunneling. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 140 (November 2018), 29 pages. https://doi.org/10.1145/3276510

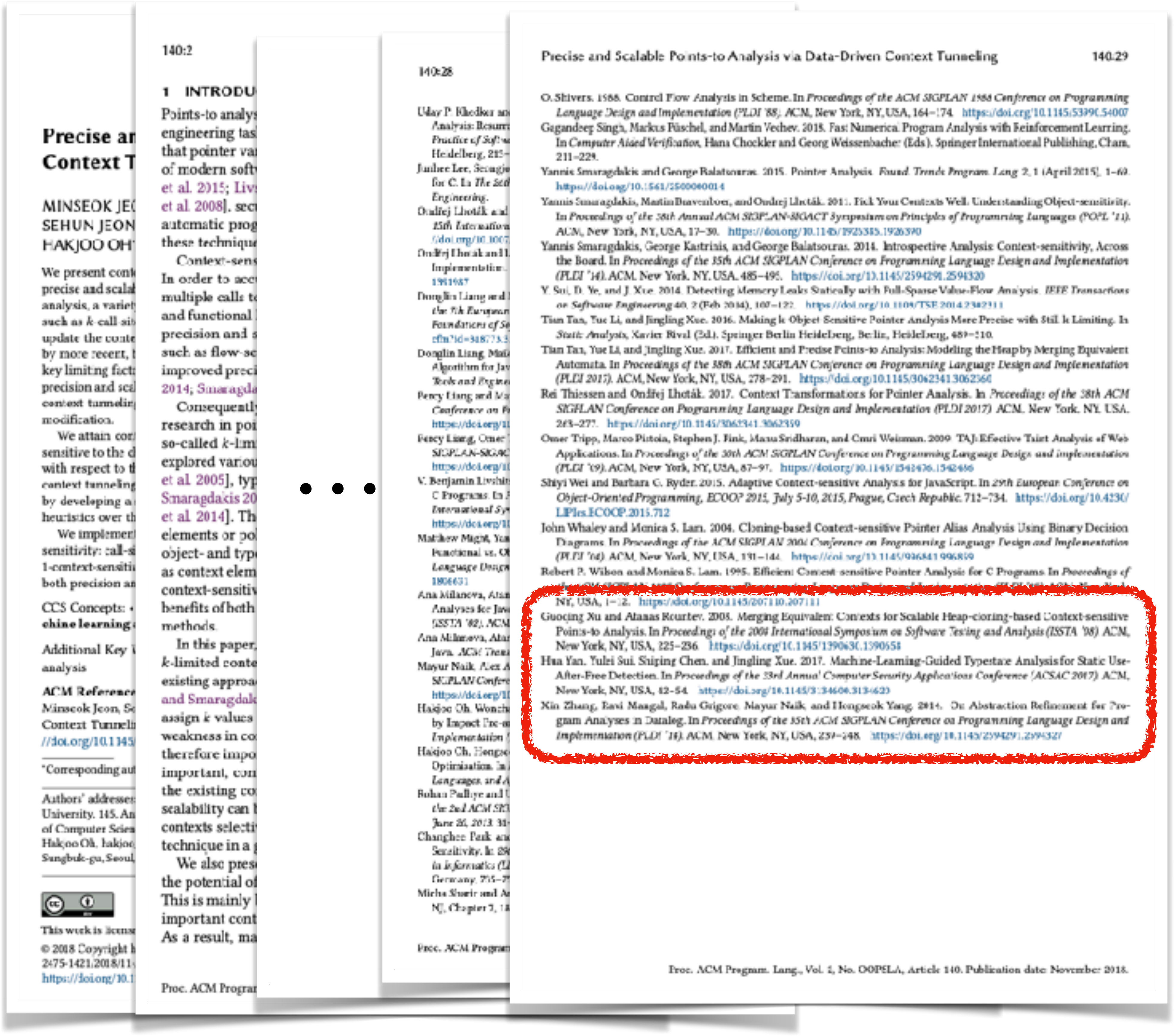
*Corresponding author

. . .

Paper (29 pages)

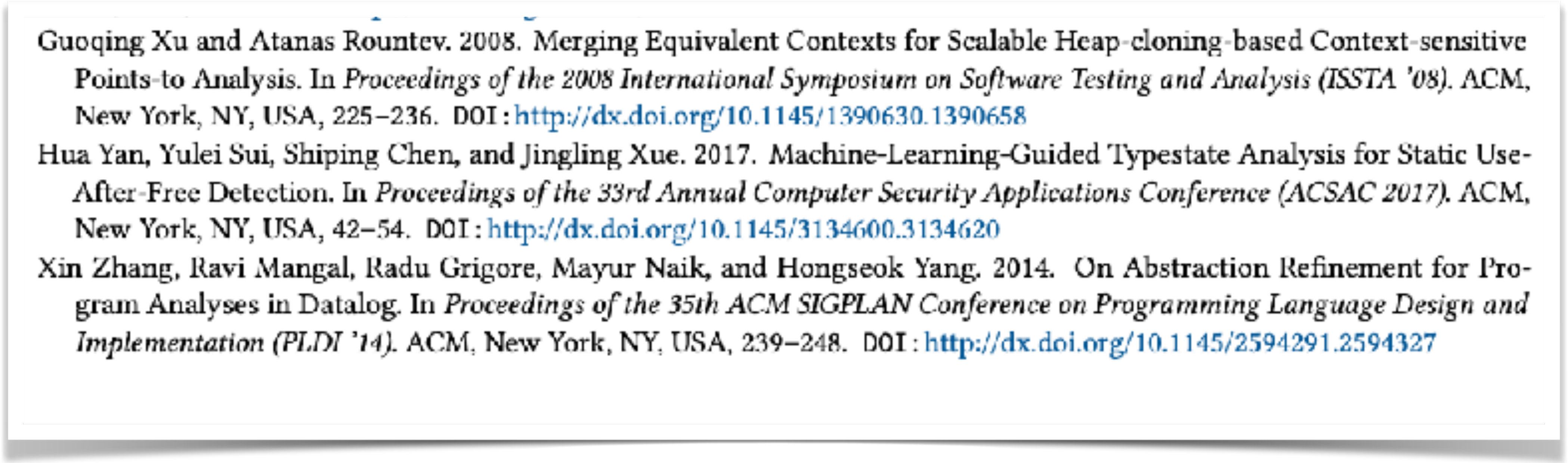
A Key Limiting Factor in Static Analysis

- Suppose a student summarized the paper with the last-3 sentences



Paper (29 pages)

Which grade would you give?



Guoqing Xu and Atanas Rountev. 2008. Merging Equivalent Contexts for Scalable Heap-cloning-based Context-sensitive Points-to Analysis. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis (ISSTA '08)*. ACM, New York, NY, USA, 225–236. DOI: http://dx.doi.org/10.1145/1390630.1390658

Hua Yan, Yulei Sui, Shiping Chen, and Jingling Xue. 2017. Machine-Learning-Guided Typestate Analysis for Static Use-After-Free Detection. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC 2017)*. ACM, New York, NY, USA, 42–54. DOI: http://dx.doi.org/10.1145/3134600.3134620

Xin Zhang, Ravi Mangal, Radu Grigore, Mayur Naik, and Hongseok Yang. 2014. On Abstraction Refinement for Program Analyses in Datalog. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*. ACM, New York, NY, USA, 239–248. DOI: http://dx.doi.org/10.1145/2594291.2594327

Last 3 sentence context abstraction

for 40 years

Existing static analyzers use last-k context abstraction

Doop — Java

Safe — Javascript

WALA — T. J. WATSON LIBRARIES FOR ANALYSIS

Sparrow — The Early Bird — C

...

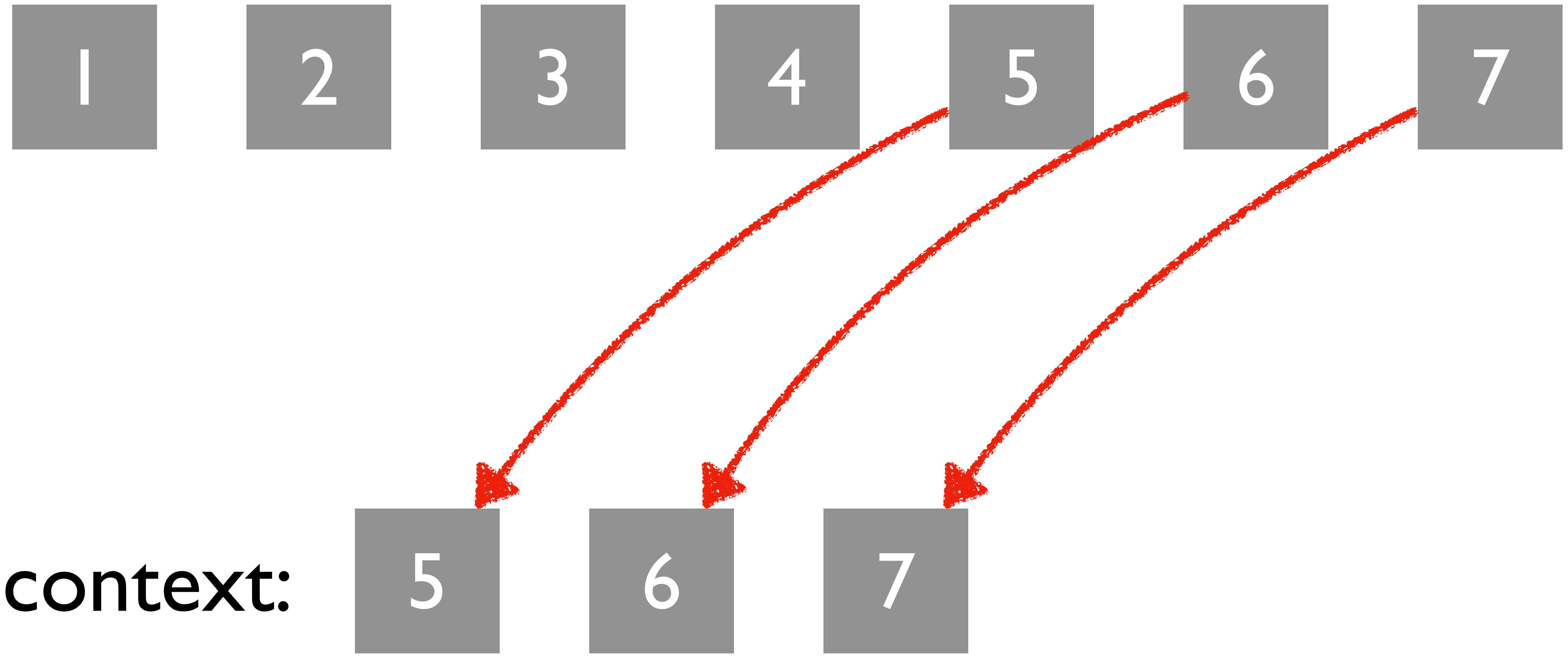
A Key Limiting Factor in Static Analysis

- Conventional k-context sensitivity keeps the last k

Used for 40 years

Concrete context:

Abstract context:

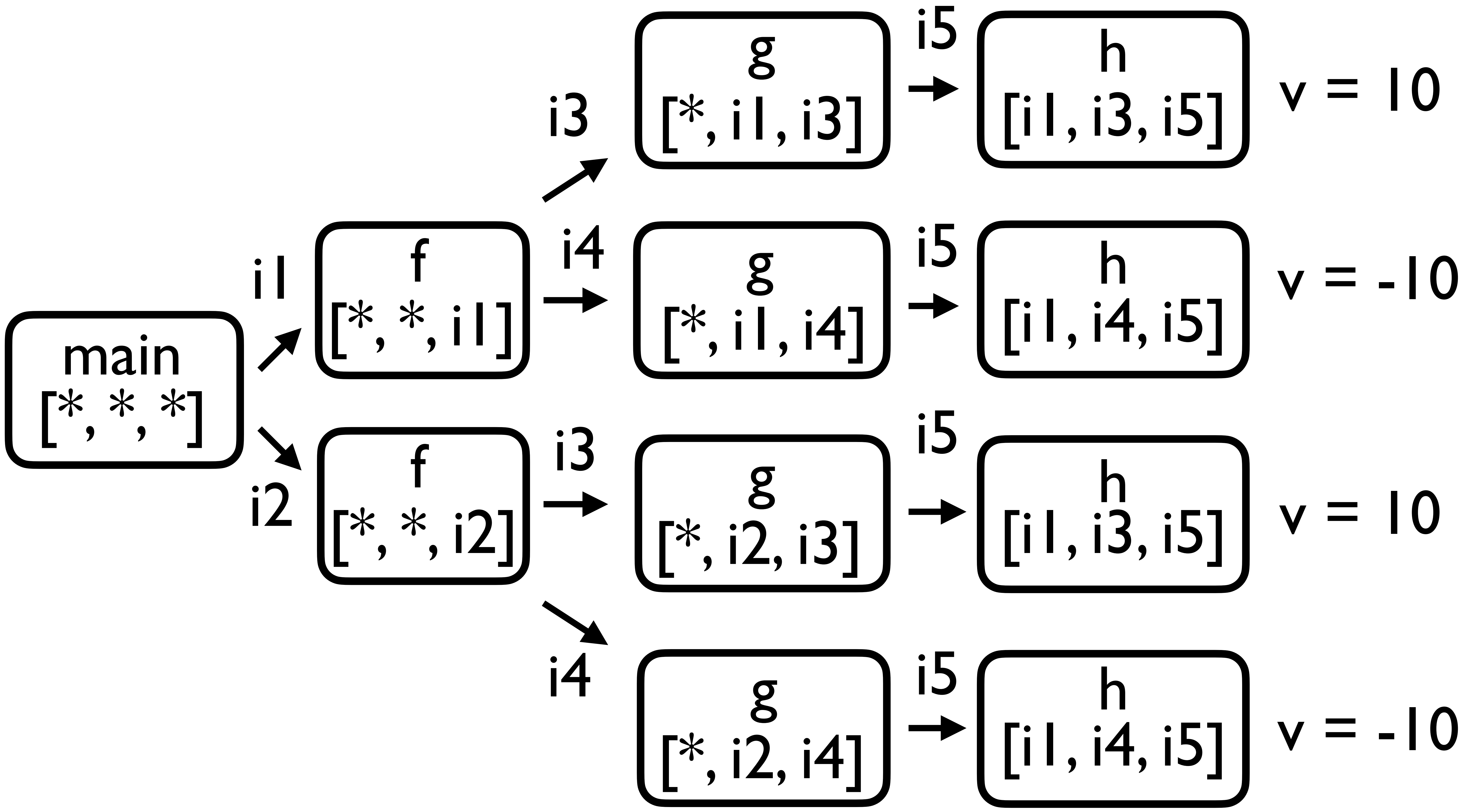


3-context sensitivity

A Key Limiting Factor in Static Analysis

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

Example program

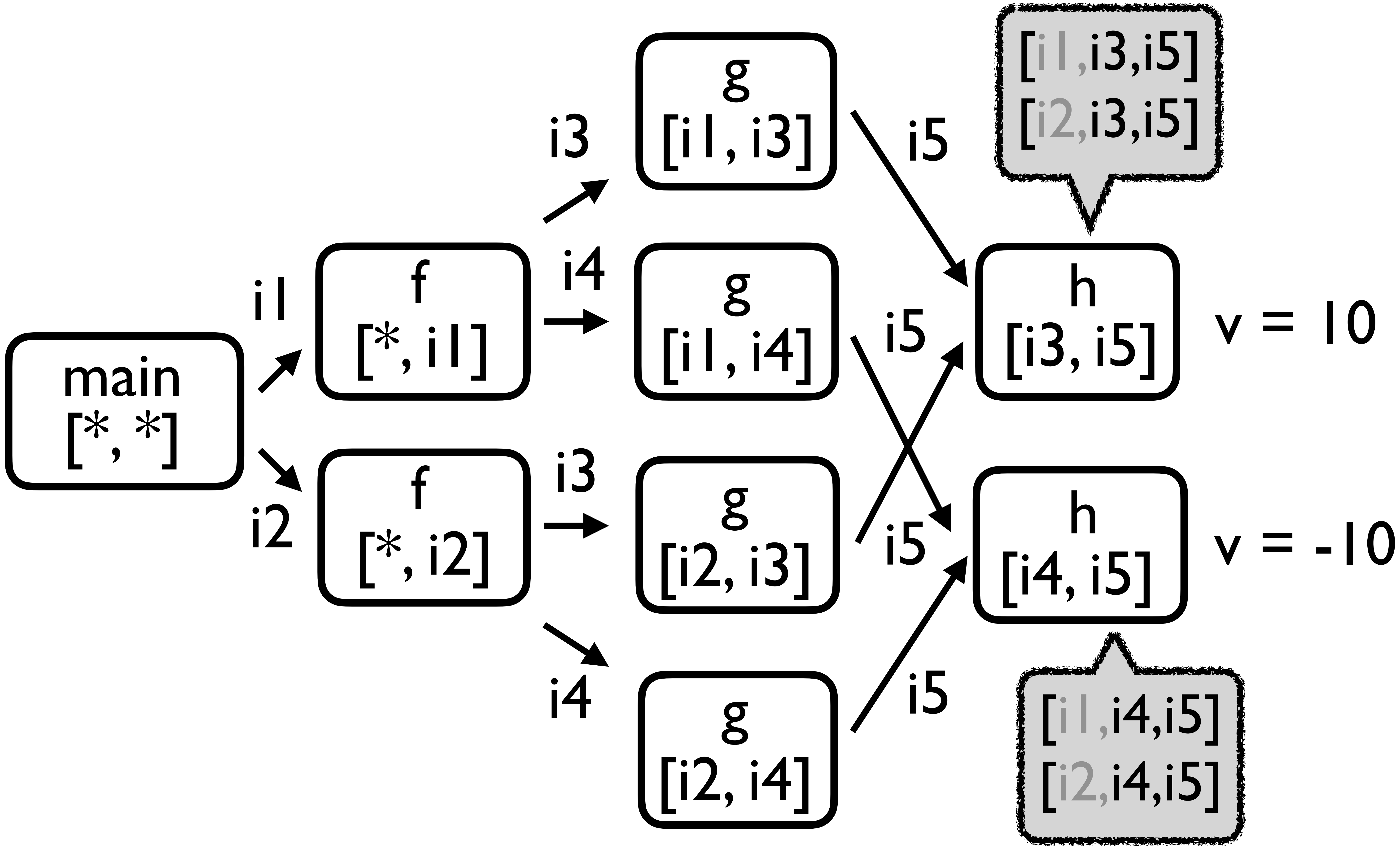


3-context sensitivity

A Key Limiting Factor in Static Analysis

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

Example program



2-context sensitivity

A Key Limiting Factor in Static Analysis

Consciousness in static analysis community

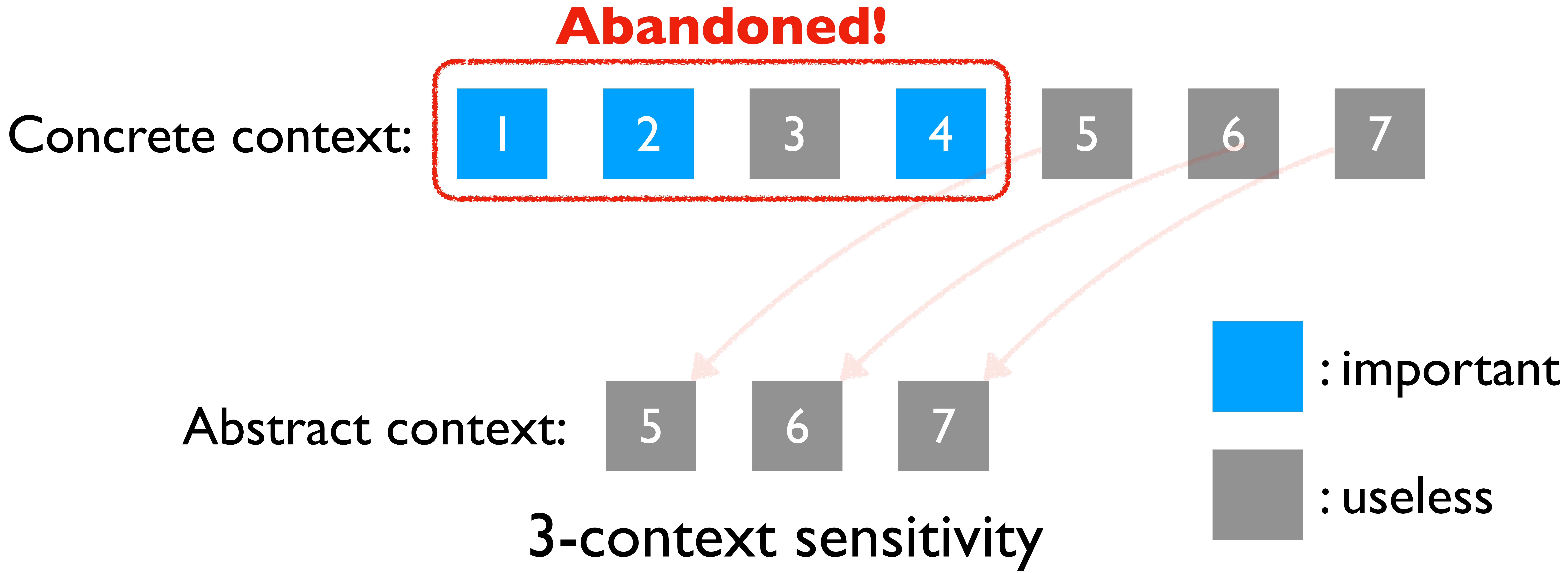
Last k

"A key part of the appeal of standard 1-CFA, 2-CFA, etc. and of 1-object sensitivity is their simplicity and universal applicability."

- A reviewer [expert]

A Key Limiting Factor in Static Analysis

- Conventional k-context sensitivity keeps the last k ← Used for 40 years



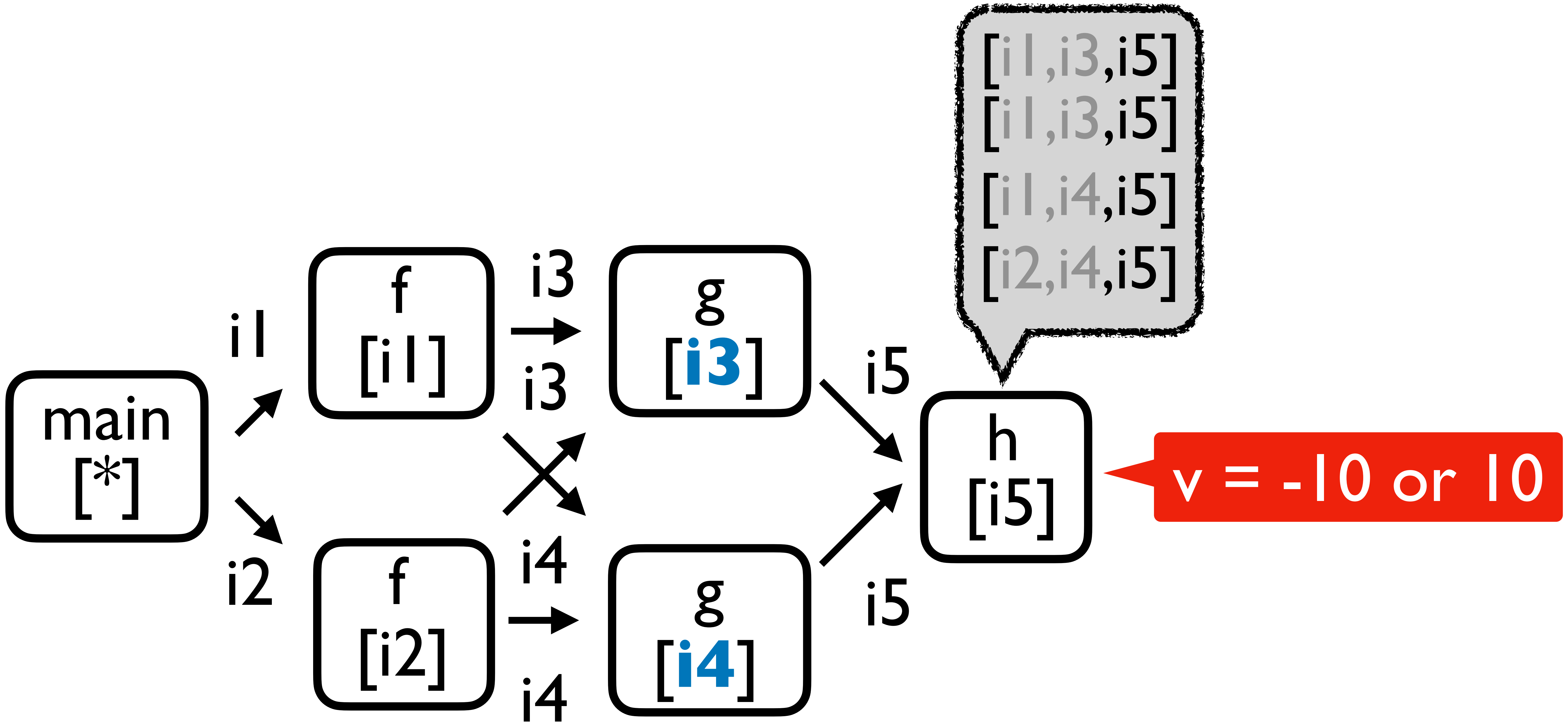
A Key Limiting Factor in Static Analysis

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

x = 10 or -10

Example program

unable to prove the query



1-context sensitivity

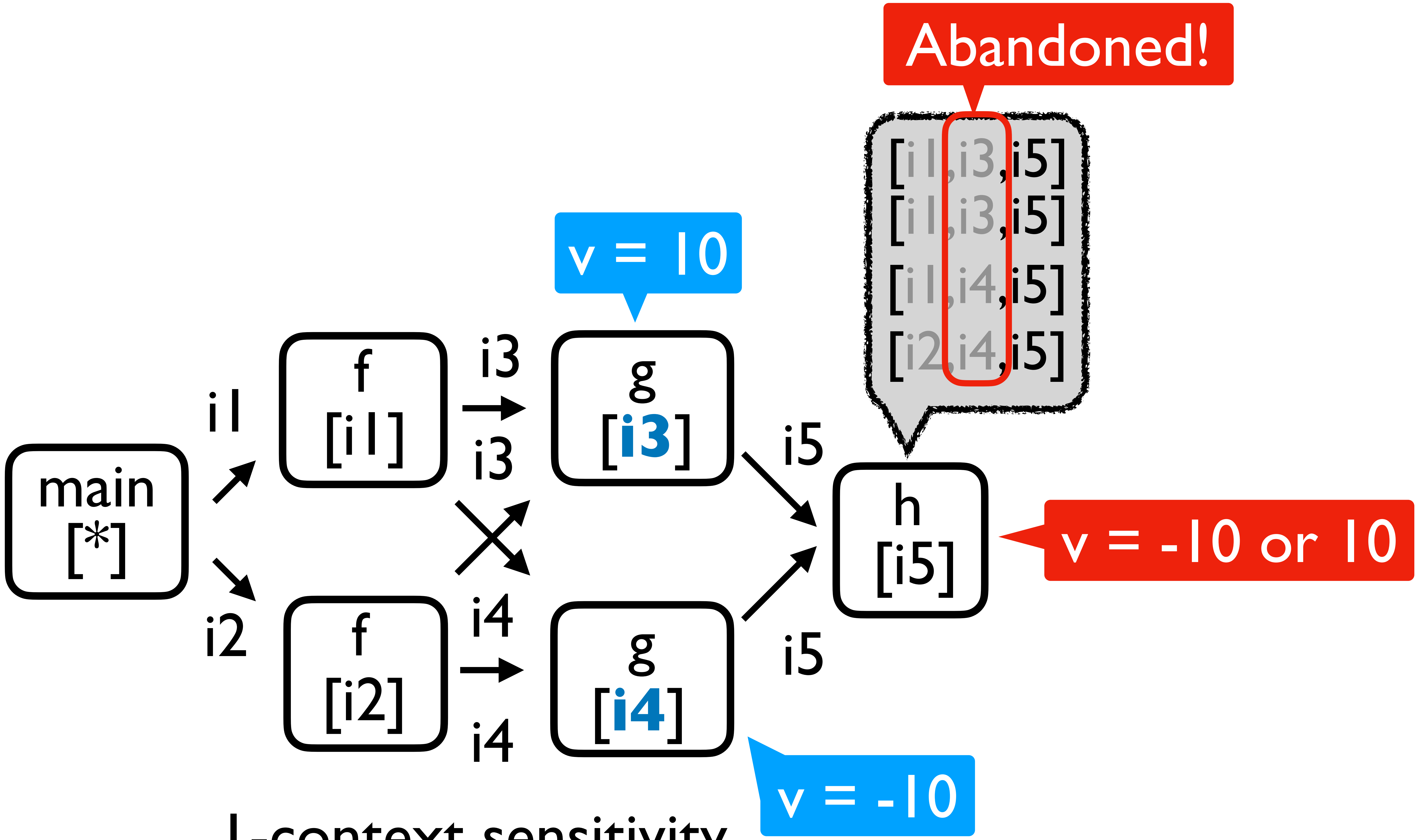
A Key Limiting Factor in Static Analysis

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

x = 10 or -10

unable to prove the query

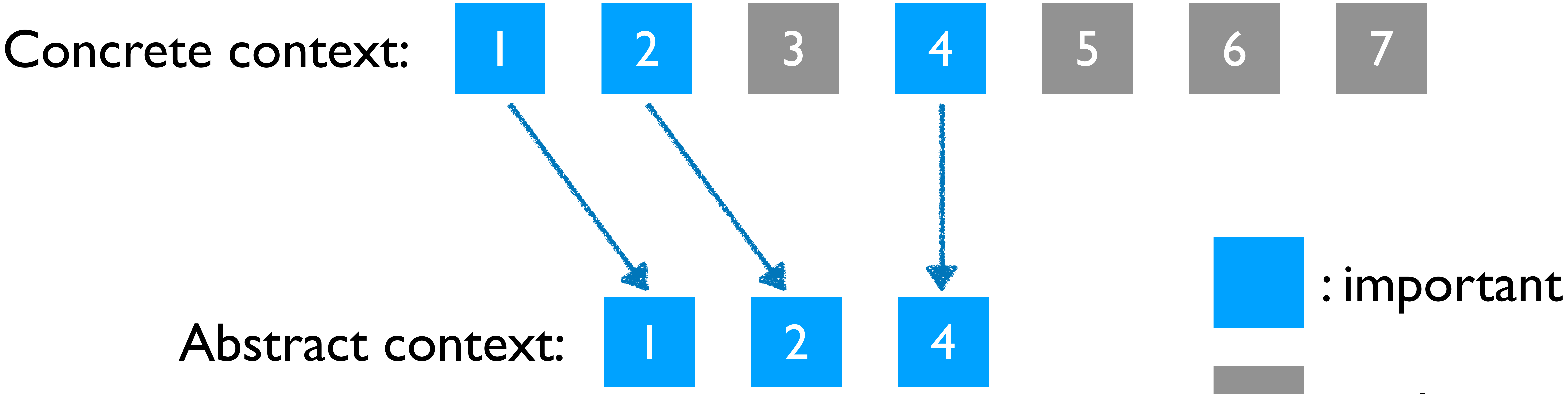
Example program



1-context sensitivity

Our Solution: Keep Important K

- Our solution is to keep the most important k instead of the last k

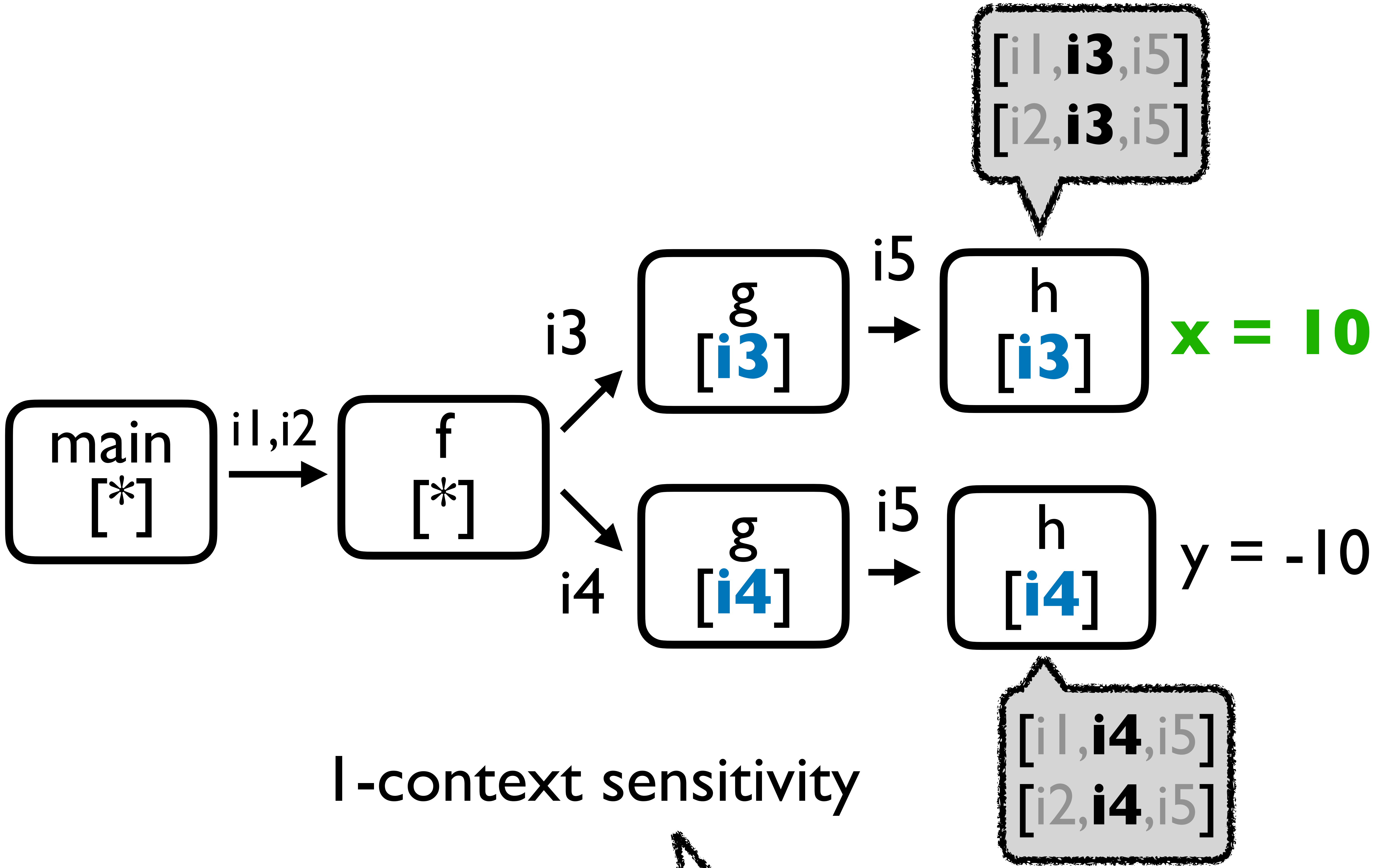


3-context sensitivity

A Key Limiting Factor in Static Analysis

```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

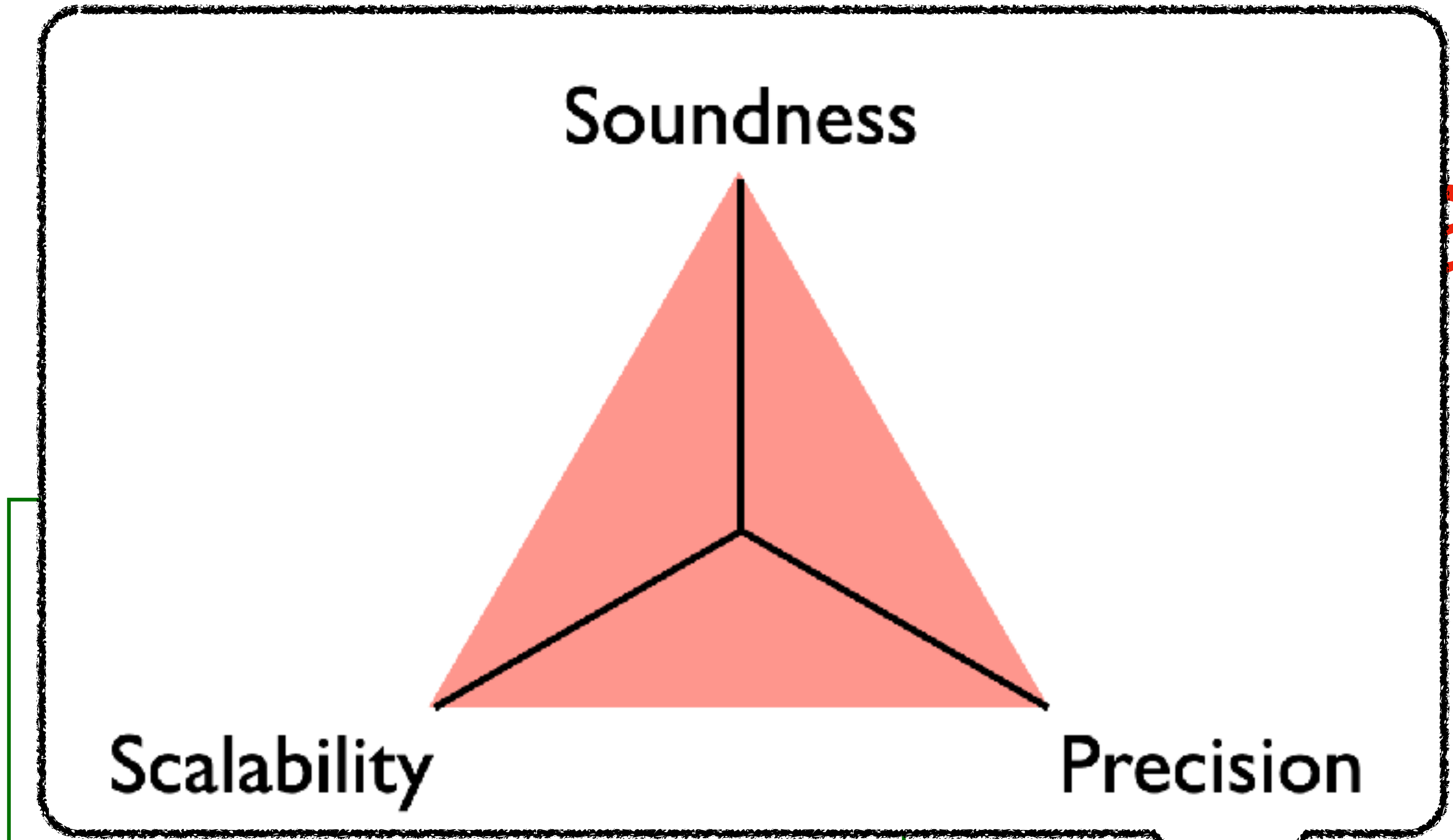
Example program



1-context sensitivity

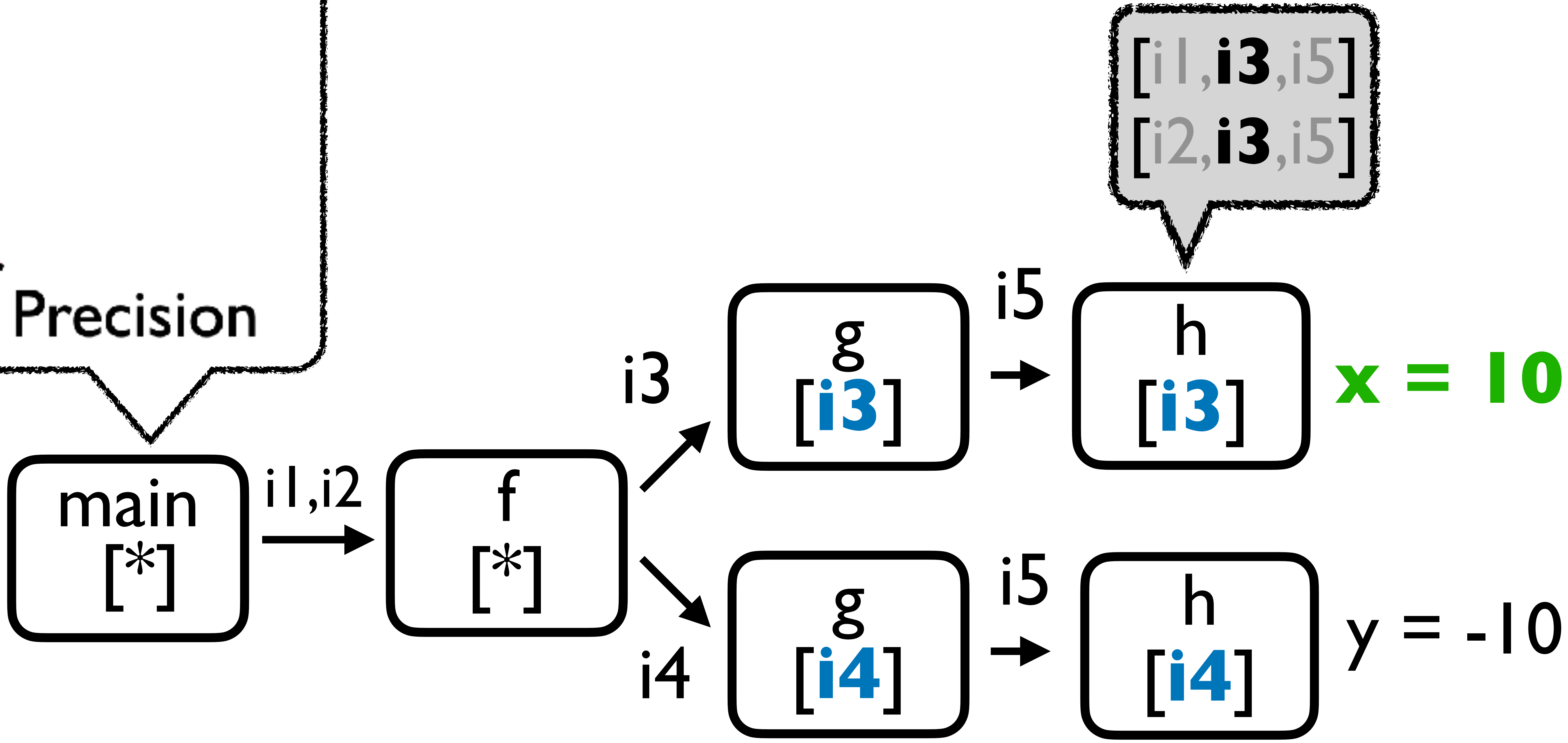
Important : {i3,i4}
Unimportant : {i1,i2,i5}

actor in Static Analysis




```
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

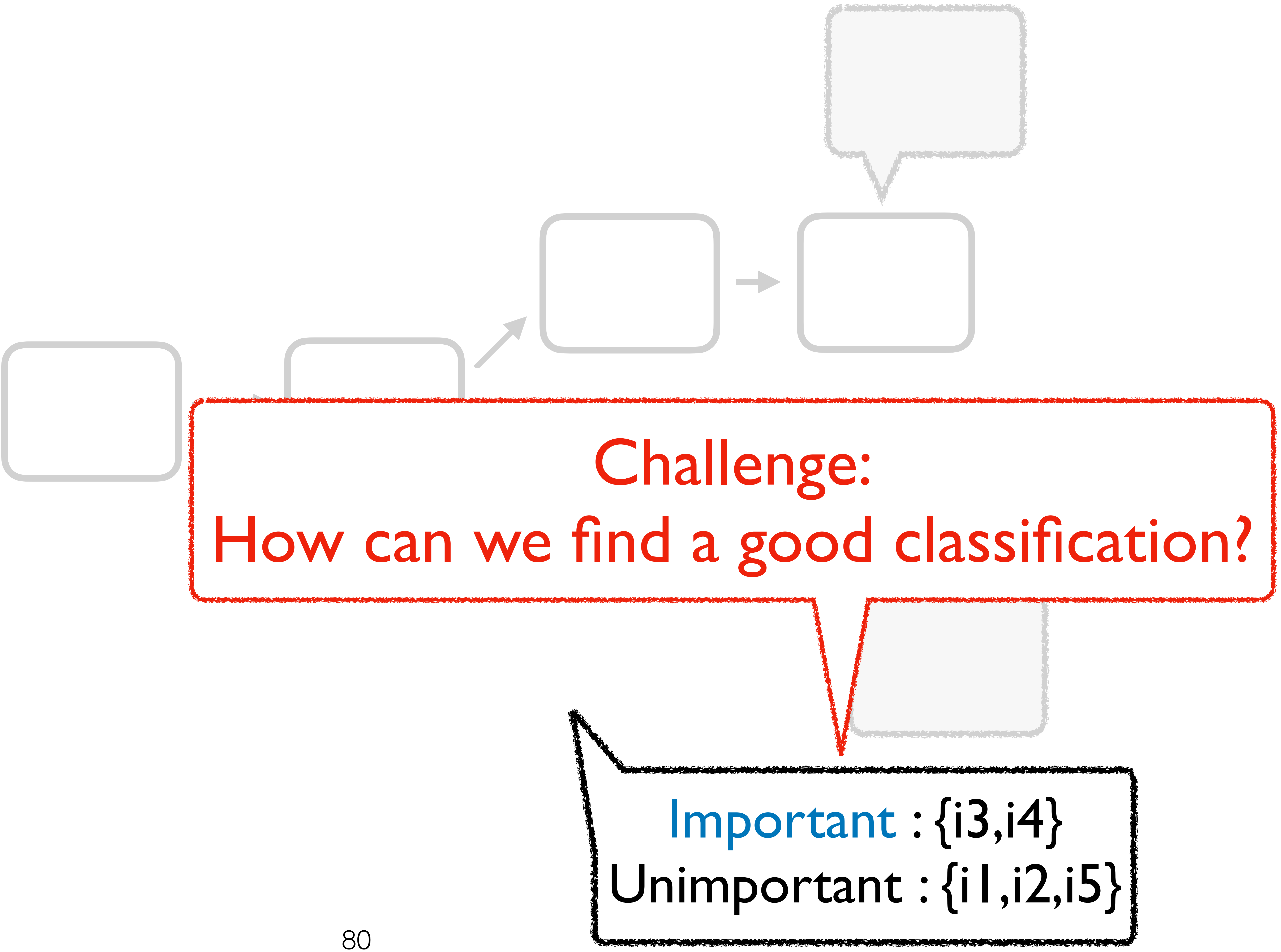
Example program



1-context sensitivity

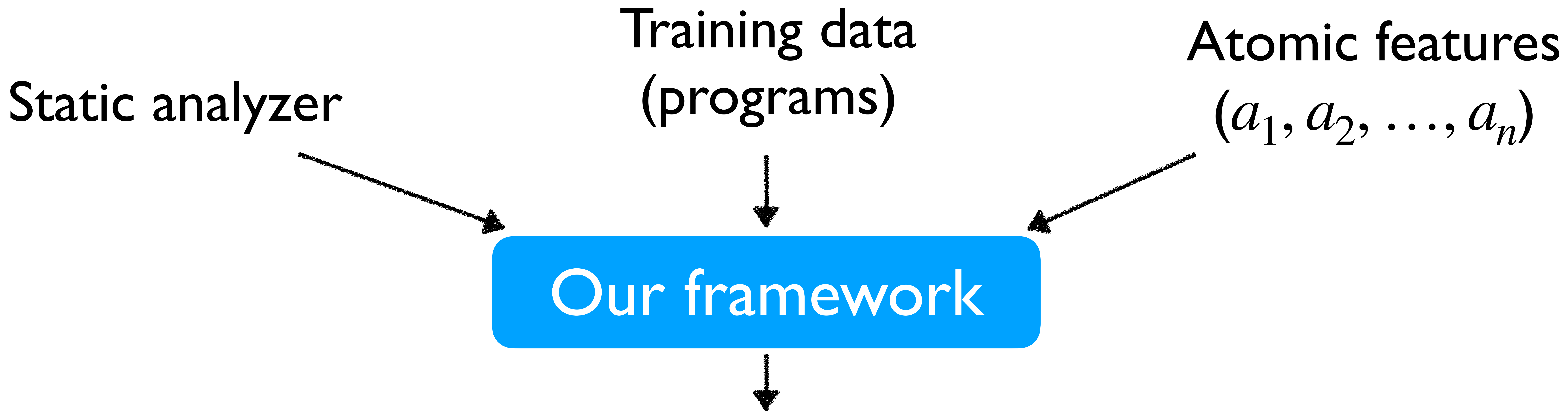
Important : {i3,i4}
Unimportant : {i1,i2,i5}

A Key Limiting Factor in Static Analysis



Our Learning Framework

- We designed a framework for learning unimportant context elements

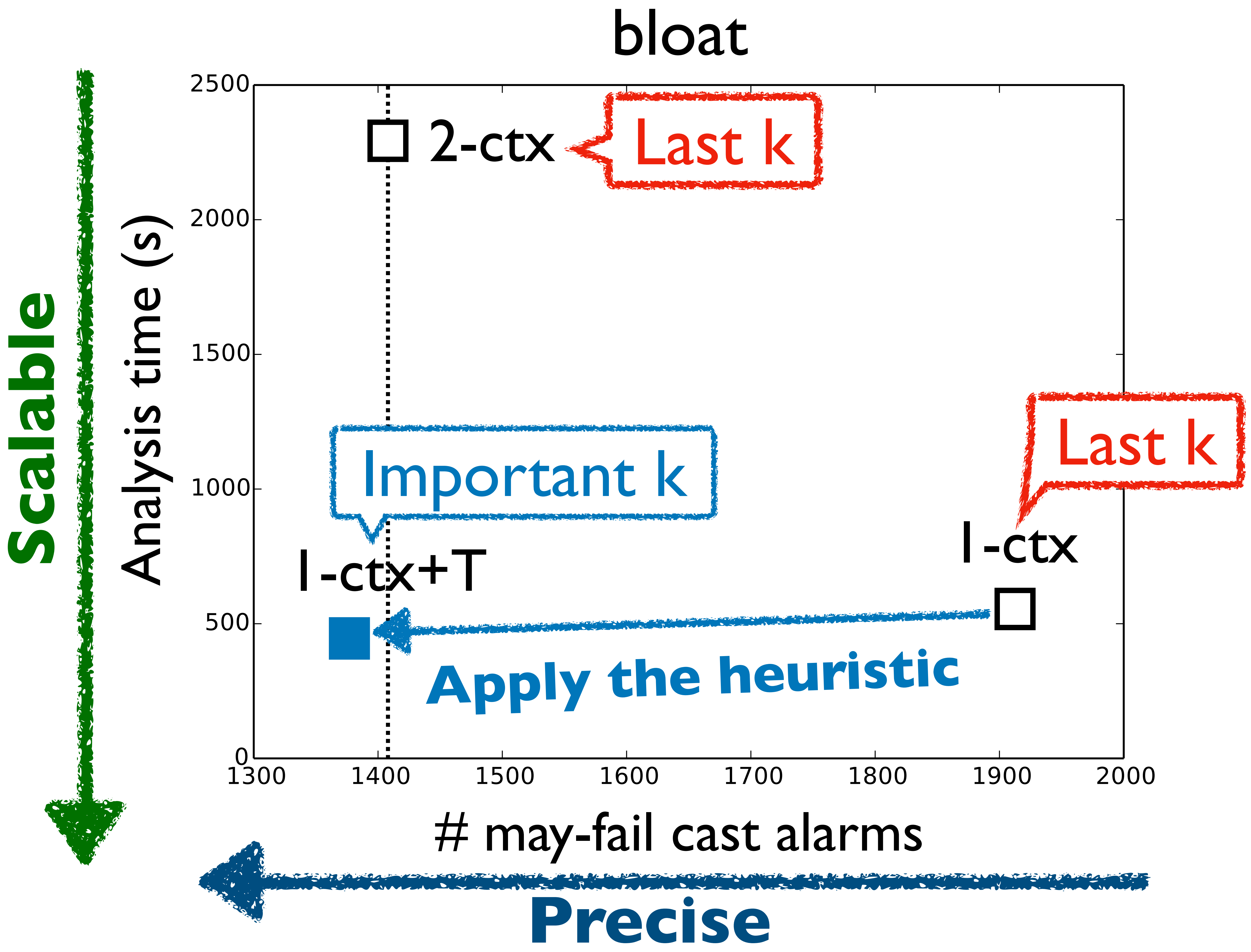


Learned heuristic describing unimportant content elements

$$f = (a_1 \land \neg a_2 \land \neg a_3 \land \ldots) \lor (a_1 \land \neg a_3 \land a_7 \land \ldots) \lor \ldots$$

Performance of Our Learned Heuristic

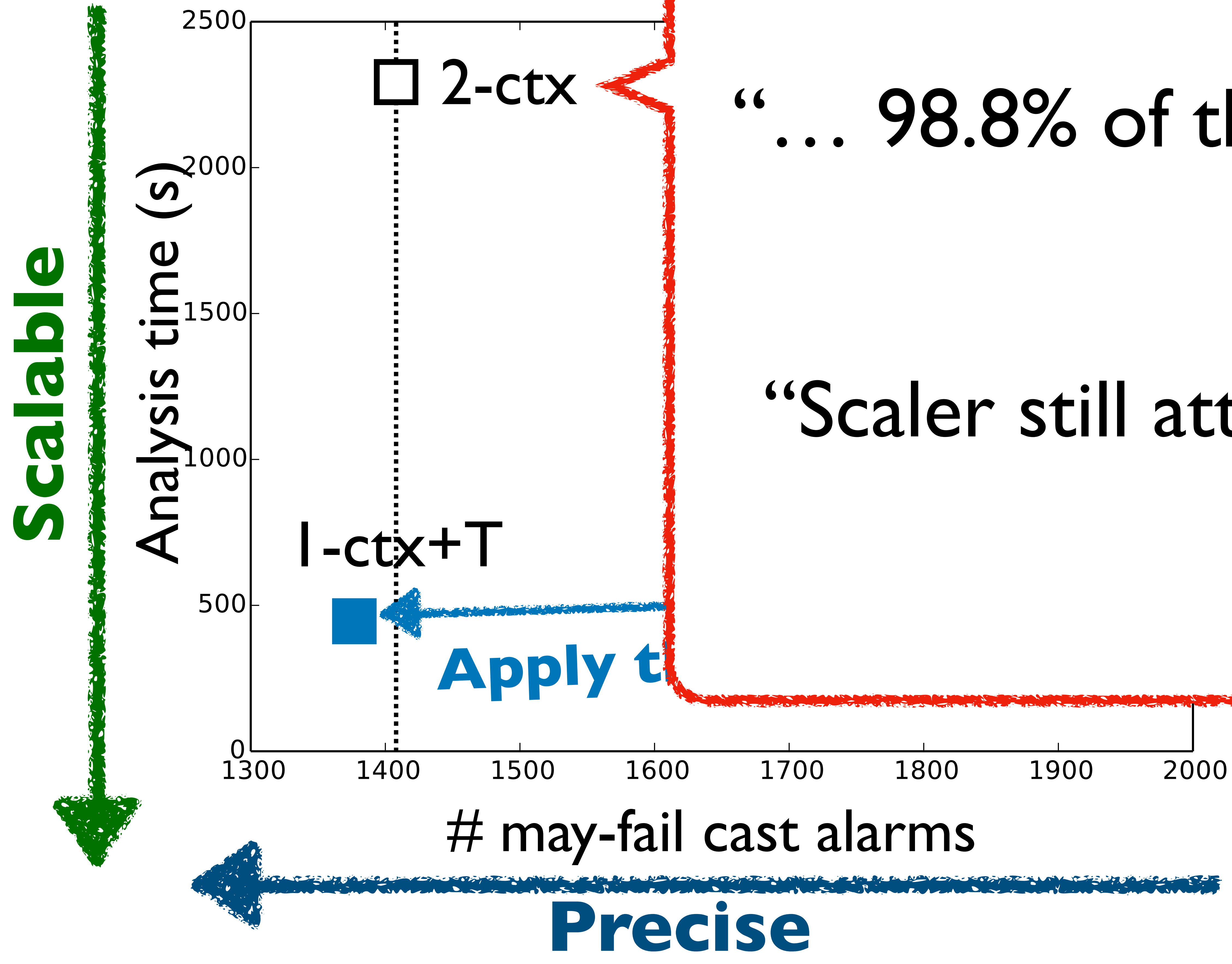
- 1-ctx with important-k is even more precise than conventional 2-ctx



Perfor

- 1-ctx with impo

b



- **2-ctx had been used as a precision upper bound**

"…it covers more than two-thirds of the precision advantage of 2objH"

-Smaragdakis et al. [PLDI' 14]

"… 98.8% of the precision of 2obj can be preserved…"

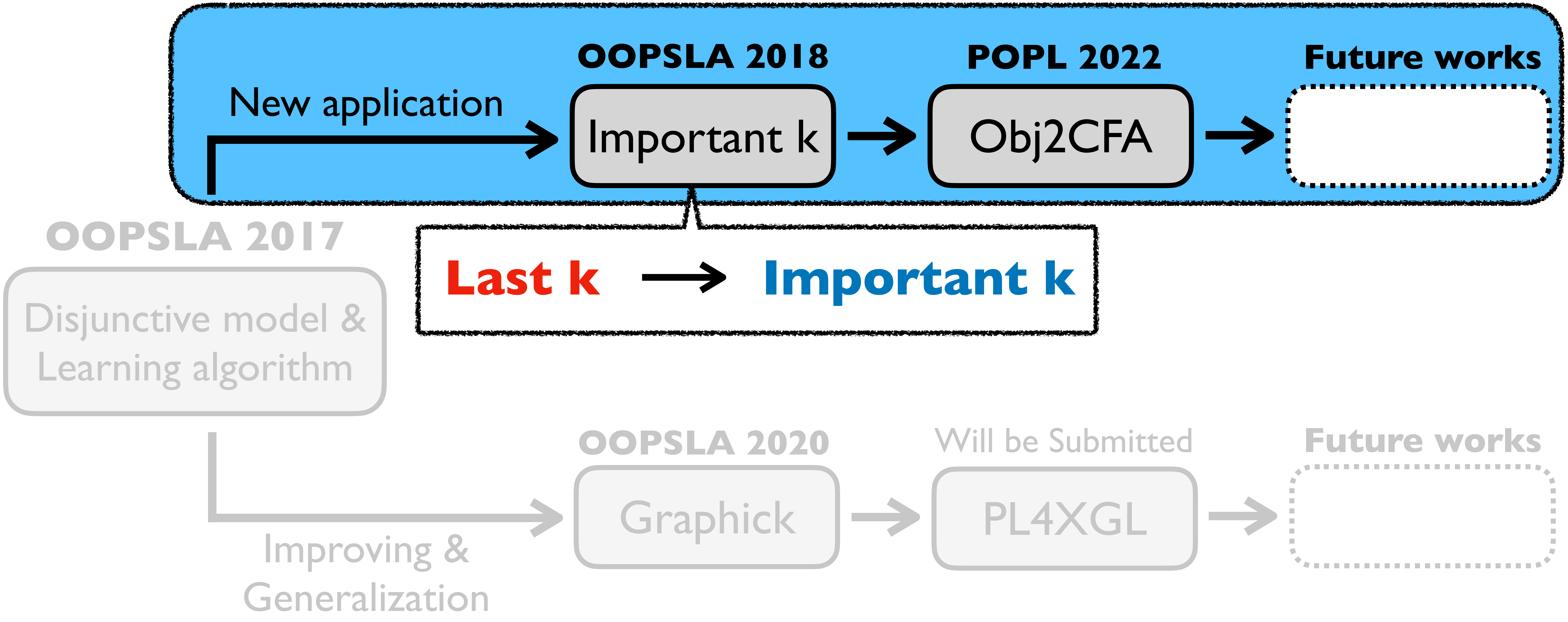
-Li et al. [OOPSLA' 18]

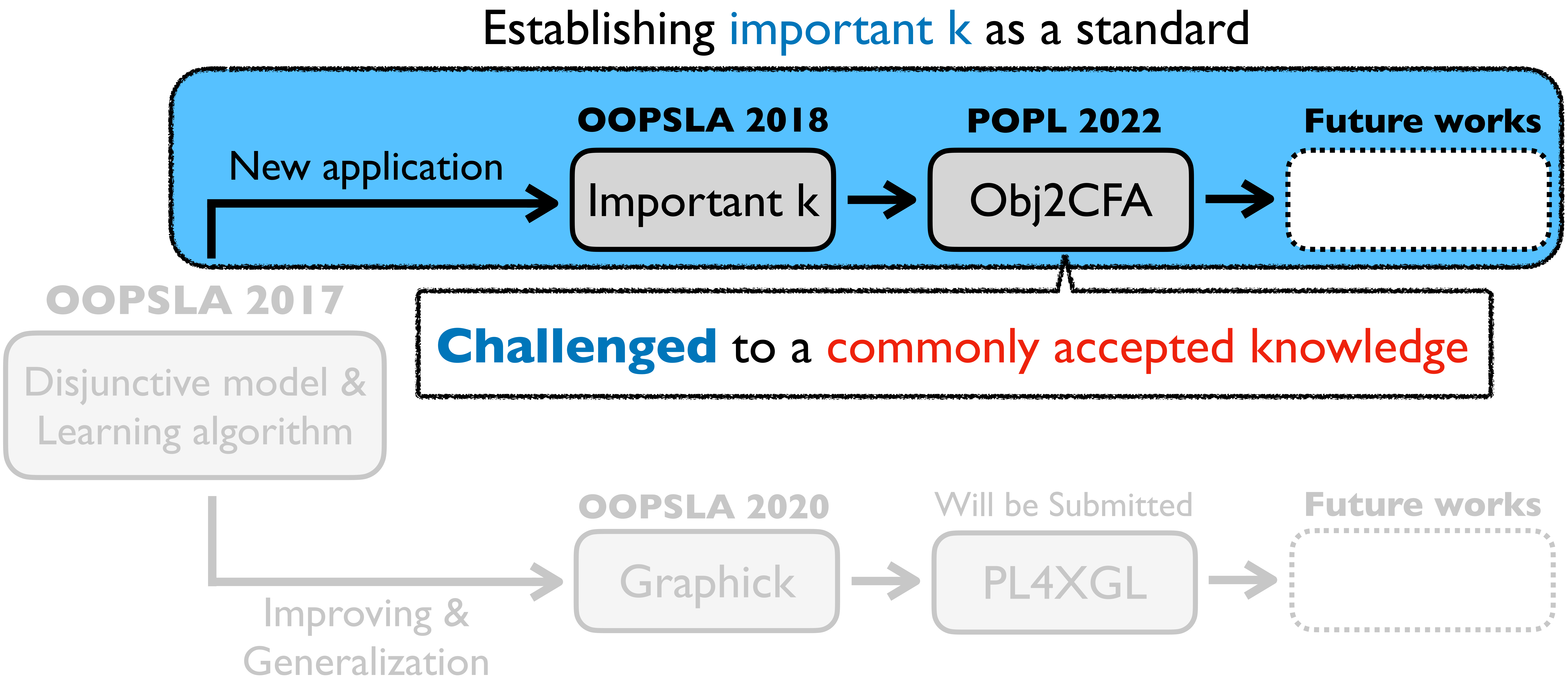
"Scaler still attains most of the precision gains of 2obj …"

-Li et al. [FSE' 18]

…

Establishing important k as a standard




Establishing important k as a standard
New application
OOPSLA 2018
Important k
POPL 2022
Obj2CFA
Future works
Challenged to a commonly accepted knowledge
OOPSLA 2017
Disjunctive model & Learning algorithm
Improving & Generalization
OOPSLA 2020
Graphick
Will be Submitted
PL4XGL
Future works

Call-site Sensitivity vs Object Sensitivity

Two major camps in OOP program analysis

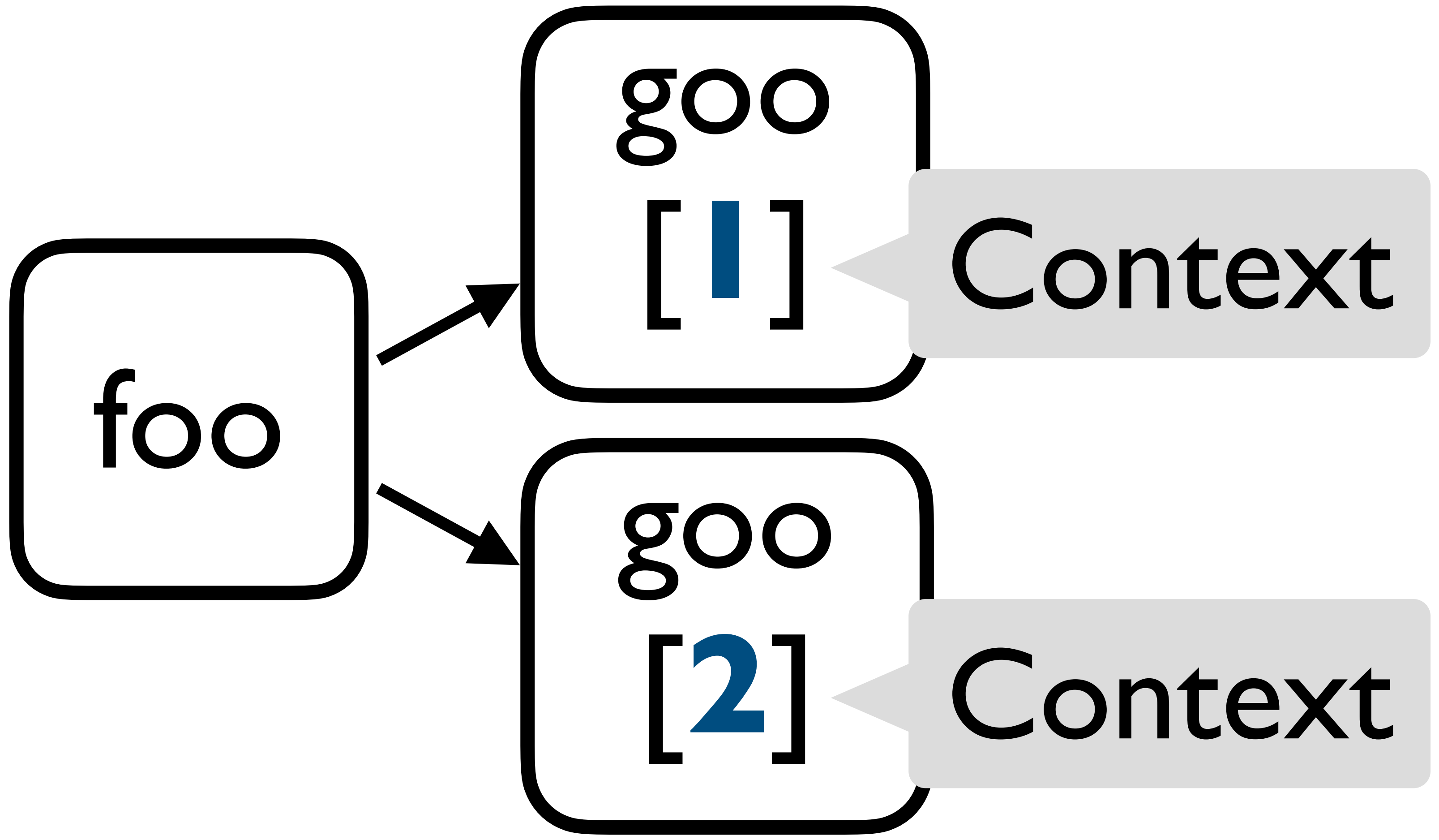
1981 — 2002 — 2010 — 2022

Call-site Sensitivity vs Object Sensitivity

Call-site sensitivity was born in 1981

- Considers "**Where**"

```
0: foo(){
1:   goo();
2:   goo();
3: }
```



Call graph

Where is it called from?

Call-site sensitivity

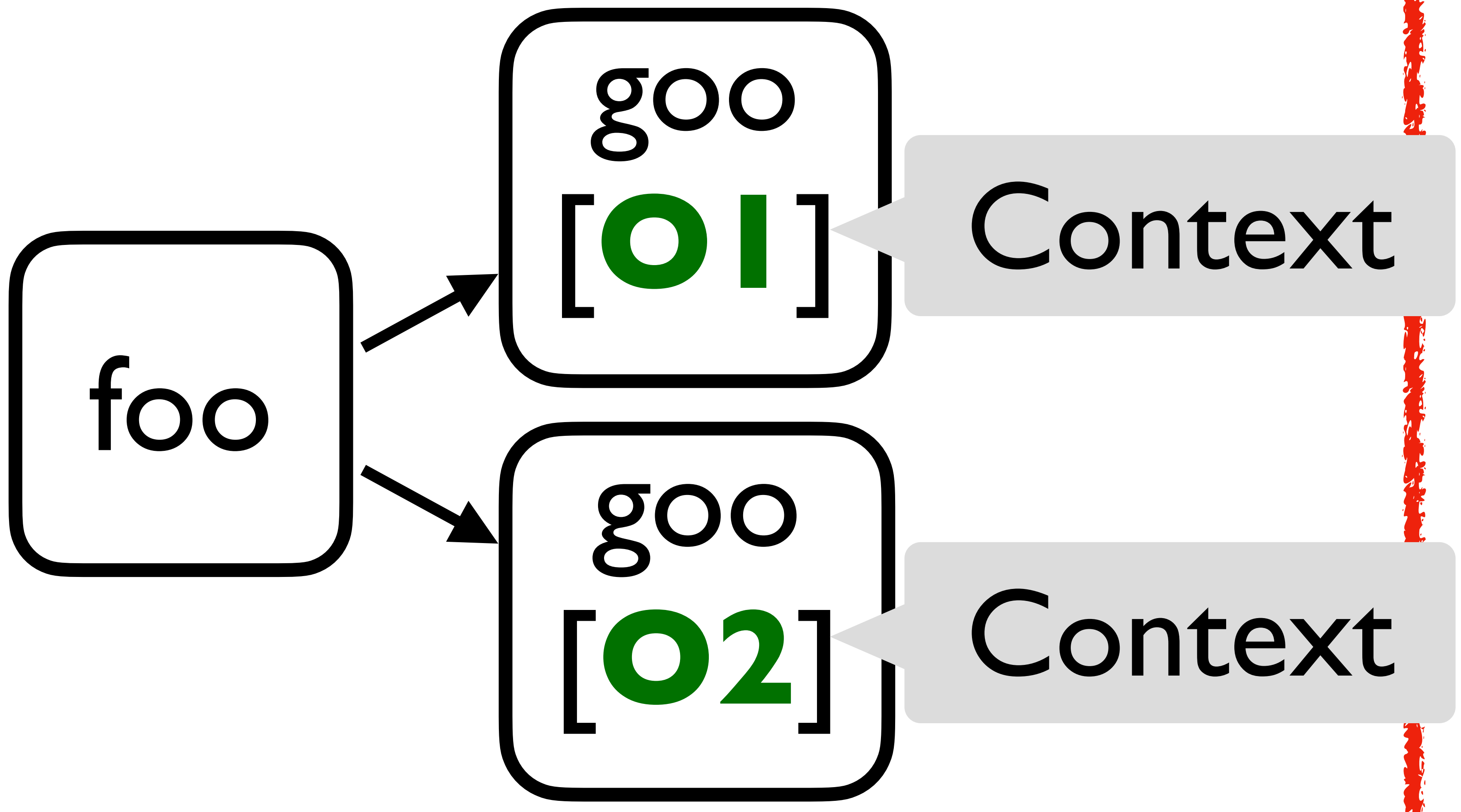
Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

- Considers "**What**"

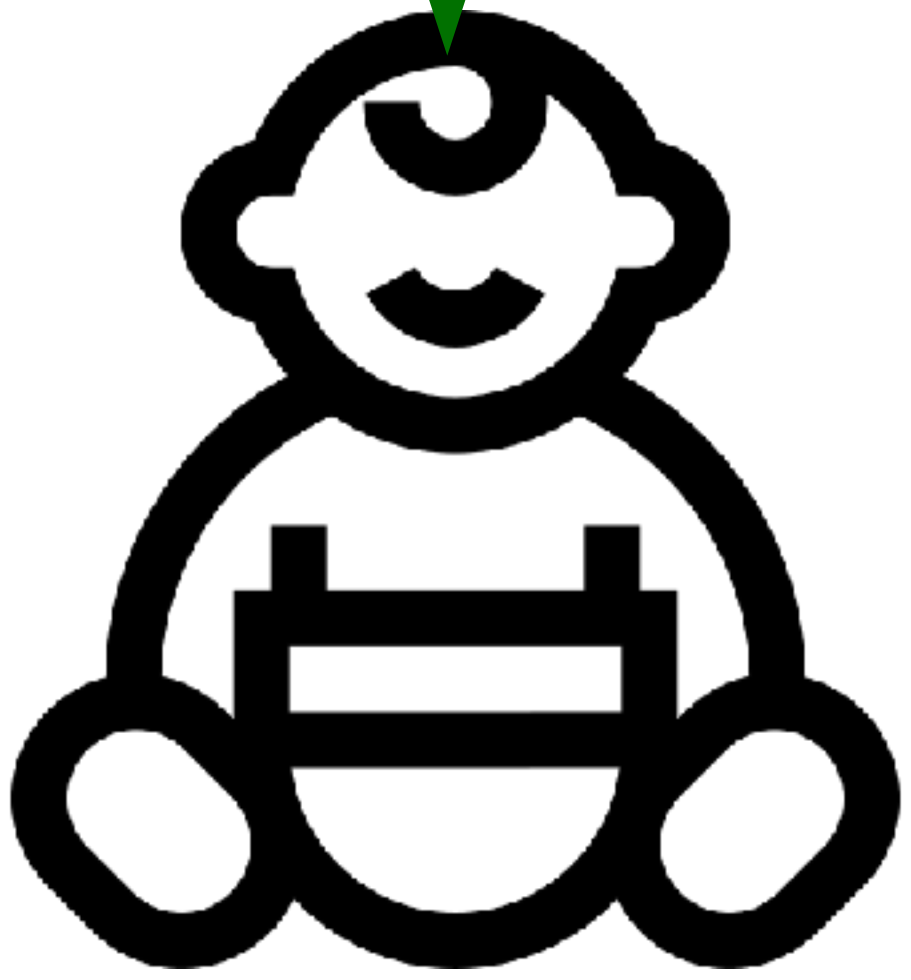
O1 or **O2**

```
0: foo(p){
1:    p.goo();
2: }
```



Call graph

What is it called with?



Object sensitivity

1981 **2002** **2010** **2022**

Call-site Sensitivity vs Object Sensitivity


Parameterized Object Sensitivity for Points-to Analysis for Java

Context-sensitive points-to analysis: is it worth it?

Evaluating the Benefits of Context-Sensitive Points-to Analysis Using a BDD-Based Implementation

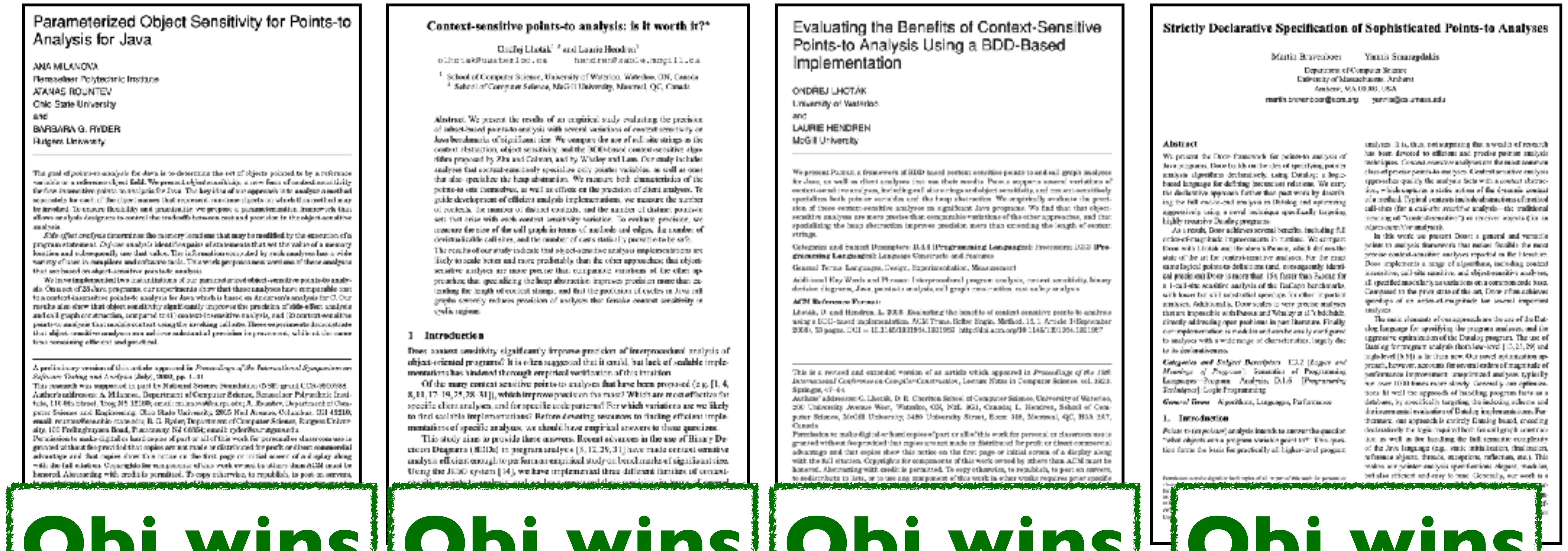
Strictly Declarative Specification of Sophisticated Points-to Analyses

...

Obj vs **CFA**

1981 — 2002 — 2010 — 2022

Call-site Sensitivity vs Object Sensitivity


Parameterized Object Sensitivity for Points-to Analysis for Java
ANA MILANOVA
ATANAS ROUNTEV
BARBARA G. RYDER

Context-sensitive points-to analysis: is it worth it?
Ondřej Lhoták and Laurie Hendren

Evaluating the Benefits of Context-Sensitive Points-to Analysis Using a BDD-Based Implementation
ONDREJ LHOTAK
LAURIE HENDREN

Strictly Declarative Specification of Sophisticated Points-to Analyses
Martin Bravenboer
Yannis Smaragdakis

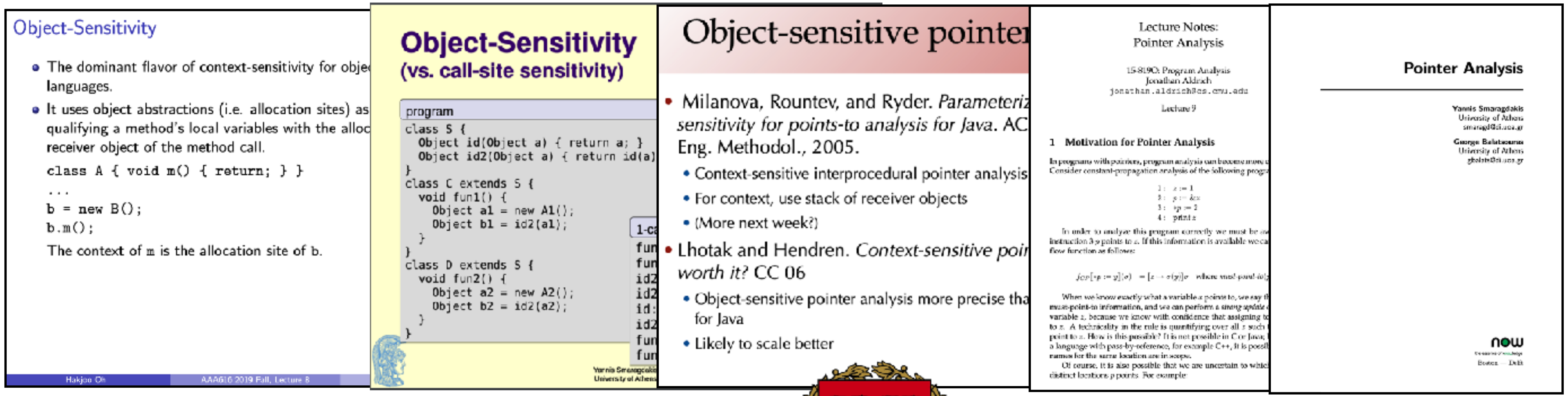
Obj wins | Obj wins | Obj wins | Obj wins

...

Obj CFA

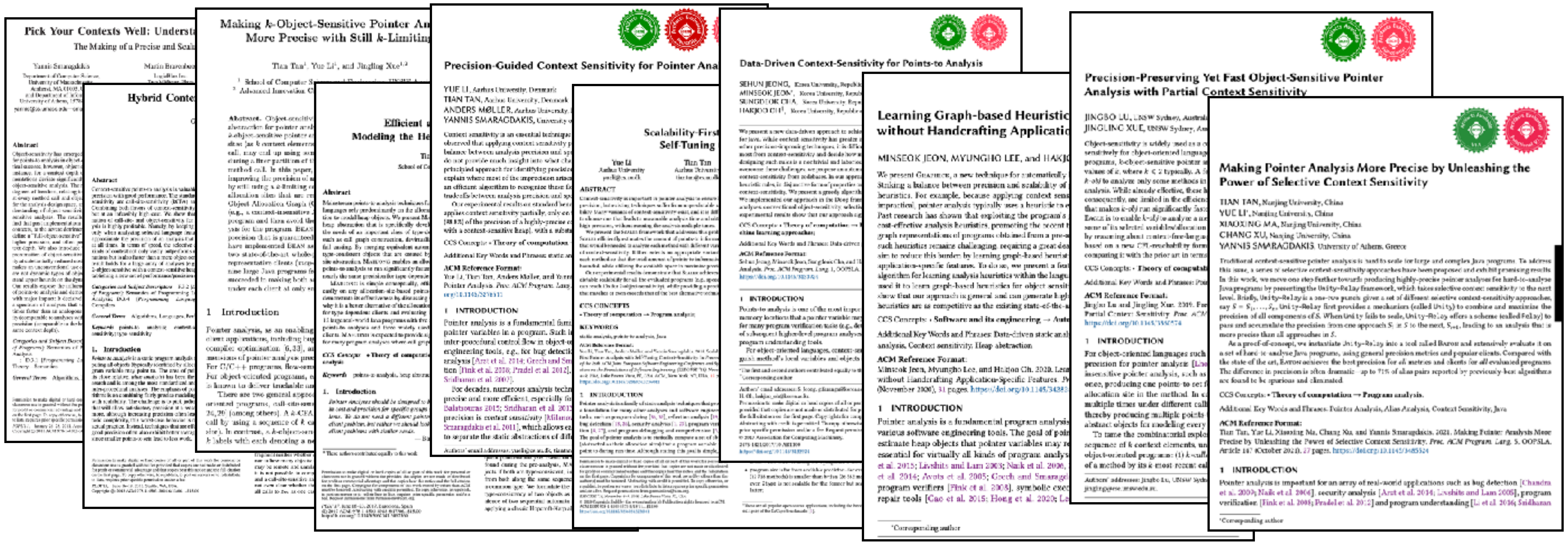
1981 2002 2010 2022

Lectures have taught superiority of object sensitivity



1981 — 2002 — 2010 — 2022

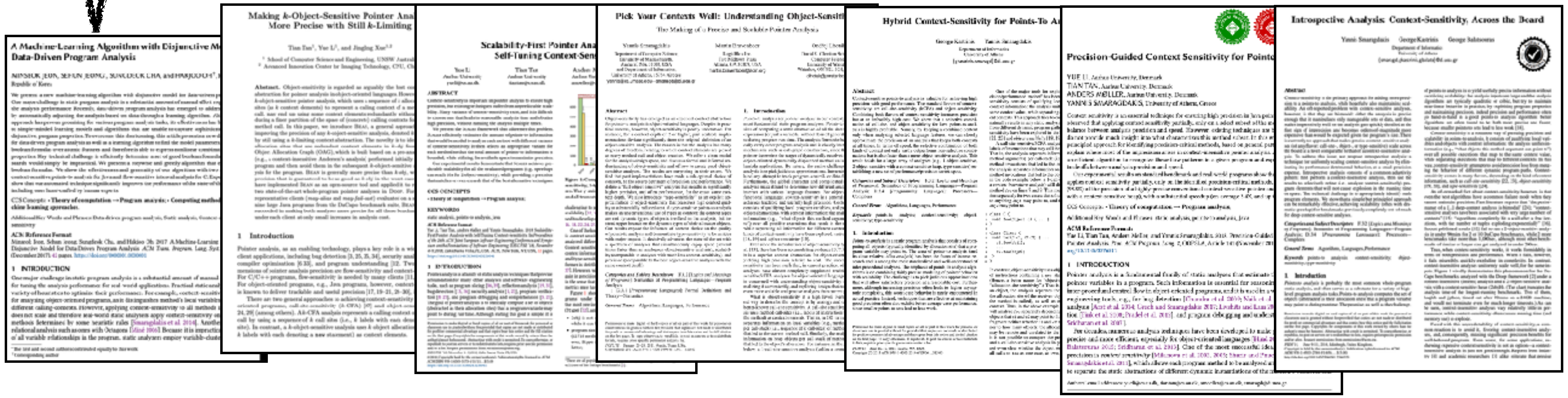
Researches on Object Sensitivity



...

Call-site Sensitivity has been ignored

"… call-site-sensitivity is less important than others …"
- Jeon et al. [2019]



CFA

1981 2002 2010 2022

Call-site Sensitivity has been ignored

"… call-site-sensitivity is less important than others …"

- Jeon et al. [2019]

I also strongly dismissed call-site sensitivity

CFA

1981 2002 2010 2022

Currently, call-site sensitivity is known as a bad context


Precise and Scalable Points-to Analysis via Data-Driven Context Tunneling

MINSEOK JEON, Korea University, Republic of Korea
SEHUN JEONG, Korea University, Republic of Korea
HAKJOO OH*, Korea University, Republic of Korea

Jeon et al. [2018]

Paradigm shift

Last k → **Important k**

1981 2002 2010 2018 2022

Return of CFA: Call-Site Sensitivity Can Be Superior to Object Sensitivity Even for Object-Oriented Programs

MINSEOK JEON and HAKJOO OH*, Korea University, Republic of Korea

In this paper, we challenge the commonly-accepted wisdom in static analysis that object sensitivity is superior to call-site sensitivity for object-oriented programs. In static analysis of object-oriented programs, object sensitivity has been established as the dominant flavor of context sensitivity thanks to its outstanding precision. On the other hand, call-site sensitivity has been regarded as unsuitable and its use in practice has been constantly discouraged for object-oriented programs. In this paper, however, we claim that call-site sensitivity is generally a superior context abstraction because it is practically possible to transform object sensitivity into more precise call-site sensitivity. Our key insight is that the previously known superiority of object sensitivity holds only in the traditional k-limited setting, where the analysis is enforced to keep the most recent k context elements. However, it no longer holds in a recently-proposed, more general setting with context tunneling. With context tunneling, where the analysis is free to choose an arbitrary k-length subsequence of context strings, we show that call-site sensitivity can simulate object sensitivity almost completely, but not vice versa. To support the claim, we present a technique, called Obj2CFA, for transforming arbitrary context-tunneled object sensitivity into more precise, context-tunneled call-site-sensitivity. We implemented Obj2CFA in Doop and used it to derive a new call-site-sensitive analysis from a state-of-the-art object-sensitive pointer analysis. Experimental results confirm that the resulting call-site sensitivity outperforms object sensitivity in precision and scalability for real-world Java programs. Remarkably, our results show that even 1-call-site sensitivity can be more precise than the conventional 3-object-sensitive analysis.

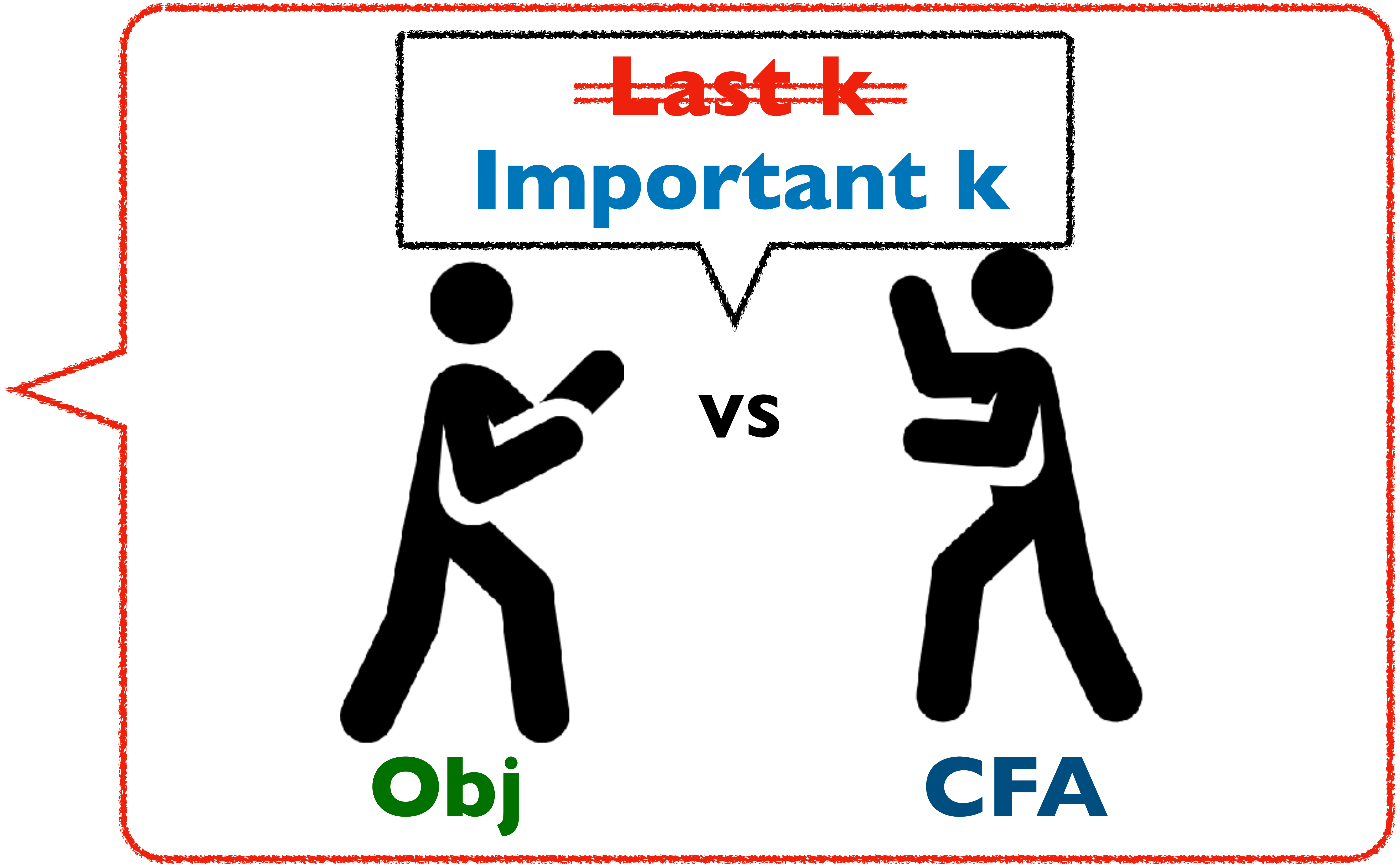
1 INTRODUCTION

"Since its introduction, object sensitivity has emerged as the dominant flavor of context sensitivity for object-oriented languages."

—Smaragdakis and Balatsouras [2015]

Context sensitivity is critically important for static program analysis of object-oriented programs. A context-sensitive analysis associates local variables and heap objects with context information of method calls, computing analysis results separately for different contexts. This way, context sensitivity prevents analysis information from being merged along different call chains. For object-oriented and higher-order languages, it is well-known that context sensitivity is the primary means for increasing analysis precision without blowing up analysis cost [Jeong et al. 2017; Kastrinis and Smaragdakis 2013; Lhoták and Hendren 2006; Li et al. 2018a; Smaragdakis and Balatsouras 2015; Smaragdakis et al. 2014; Sridharan and Bodík 2006; Thiessen and Lhoták 2017].

There have been two major flavors of context sensitivity, namely *call-site sensitivity* [Sharir and Pnueli 1981; Shivers 1988] and *object sensitivity* [Milanova et al. 2002, 2005], which differ in the choice of context information. The traditional k-call-site-sensitive analysis [Sharir and Pnueli 1981] uses a sequence of k call-sites as the context of a method. By contrast, object sensitivity uses allocation-sites as context elements: in a virtual call, e.g., a.foo(), an object-sensitive analysis uses the allocation-site of the receiver object (a) as the context of foo. The standard k-object-sensitive analysis [Milanova et al. 2002, 2005; Smaragdakis et al. 2011] maintains a sequence of



Return of CFA: Call-Site Sensitivity Can Be Superior to Object Sensitivity Even for Object-Oriented Programs

MINSEOK JEON and HAKJOO OH*, Korea University, Republic of Korea

In this paper, we challenge the commonly-accepted wisdom in static analysis that object sensitivity is superior to call-site sensitivity for object-oriented programs. In static analysis of object-oriented programs, object sensitivity has been established as the dominant flavor of context sensitivity thanks to its outstanding precision. On the other hand, call-site sensitivity has been regarded as unsuitable and its use in practice has been constantly discouraged for object-oriented programs. In this paper, however, we claim that call-site sensitivity is generally a superior context abstraction because it is practically possible to transform object sensitivity into more precise call-site sensitivity. Our key insight is that the previously known superiority of object sensitivity holds only in the traditional k-limited setting, where the analysis is enforced to keep the most recent k context elements. However, it no longer holds in a recently-proposed, more general setting with context tunneling. With context tunneling, where the analysis is free to choose an arbitrary k-length subsequence of context strings, we show that call-site sensitivity can simulate object sensitivity almost completely, but not vice versa. To support the claim, we present a technique, called Obj2CFA, for transforming arbitrary context-tunneled object sensitivity into more precise, context-tunneled call-site-sensitivity. We implemented Obj2CFA in Doop and used it to derive a new call-site-sensitive analysis from a state-of-the-art object-sensitive pointer analysis. Experimental results confirm that the resulting call-site sensitivity outperforms object sensitivity in precision and scalability for real-world Java programs. Remarkably, our results show that even 1-call-site sensitivity can be more precise than the conventional 3-object-sensitive analysis.

1 INTRODUCTION

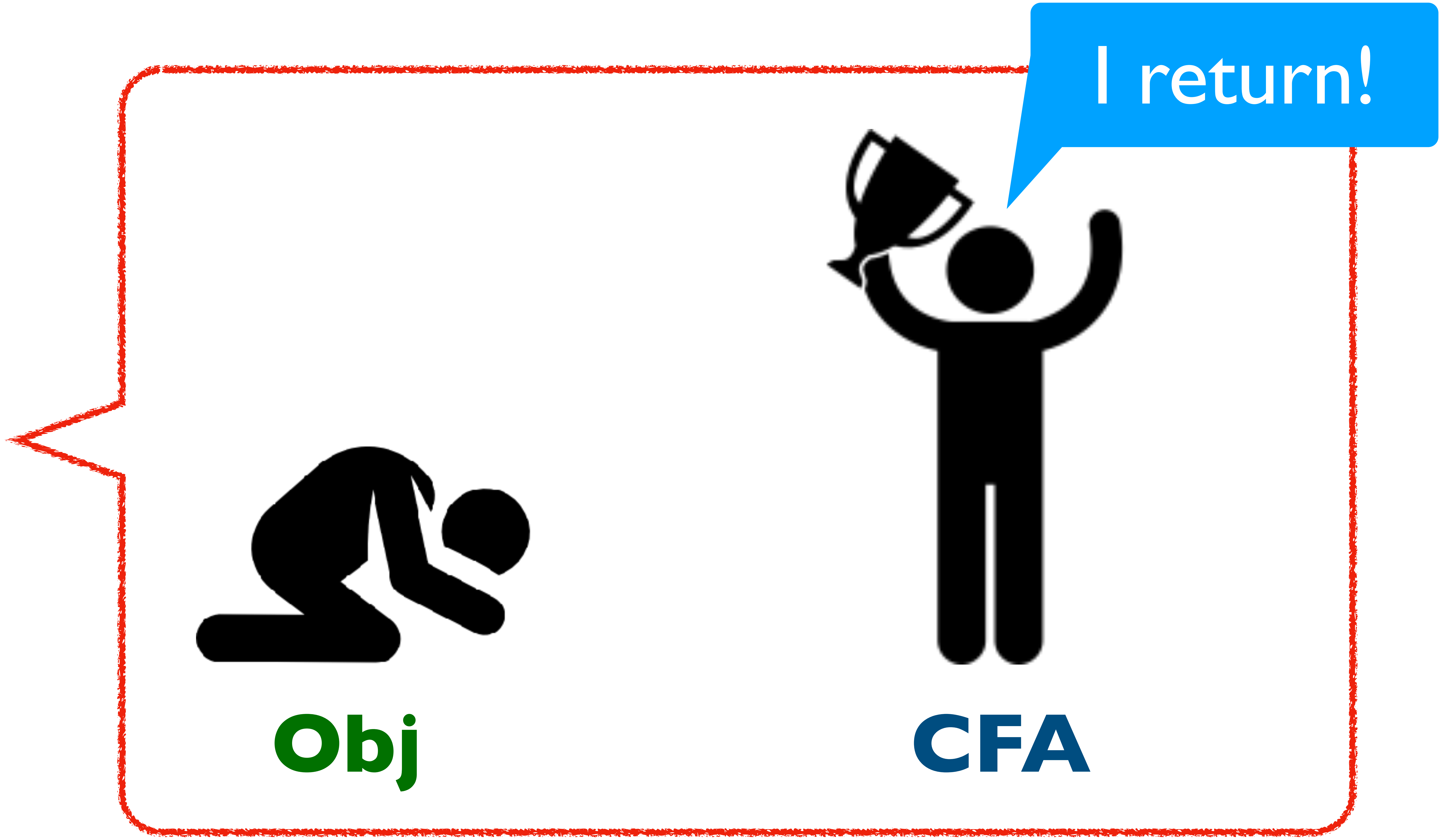
"Since its introduction, object sensitivity has emerged as the dominant flavor of context sensitivity for object-oriented languages."

—Smaragdakis and Balatsouras [2015]

Context sensitivity is critically important for static program analysis of object-oriented programs. A context-sensitive analysis associates local variables and heap objects with context information of method calls, computing analysis results separately for different contexts. This way, context sensitivity prevents analysis information from being merged along different call chains. For object-

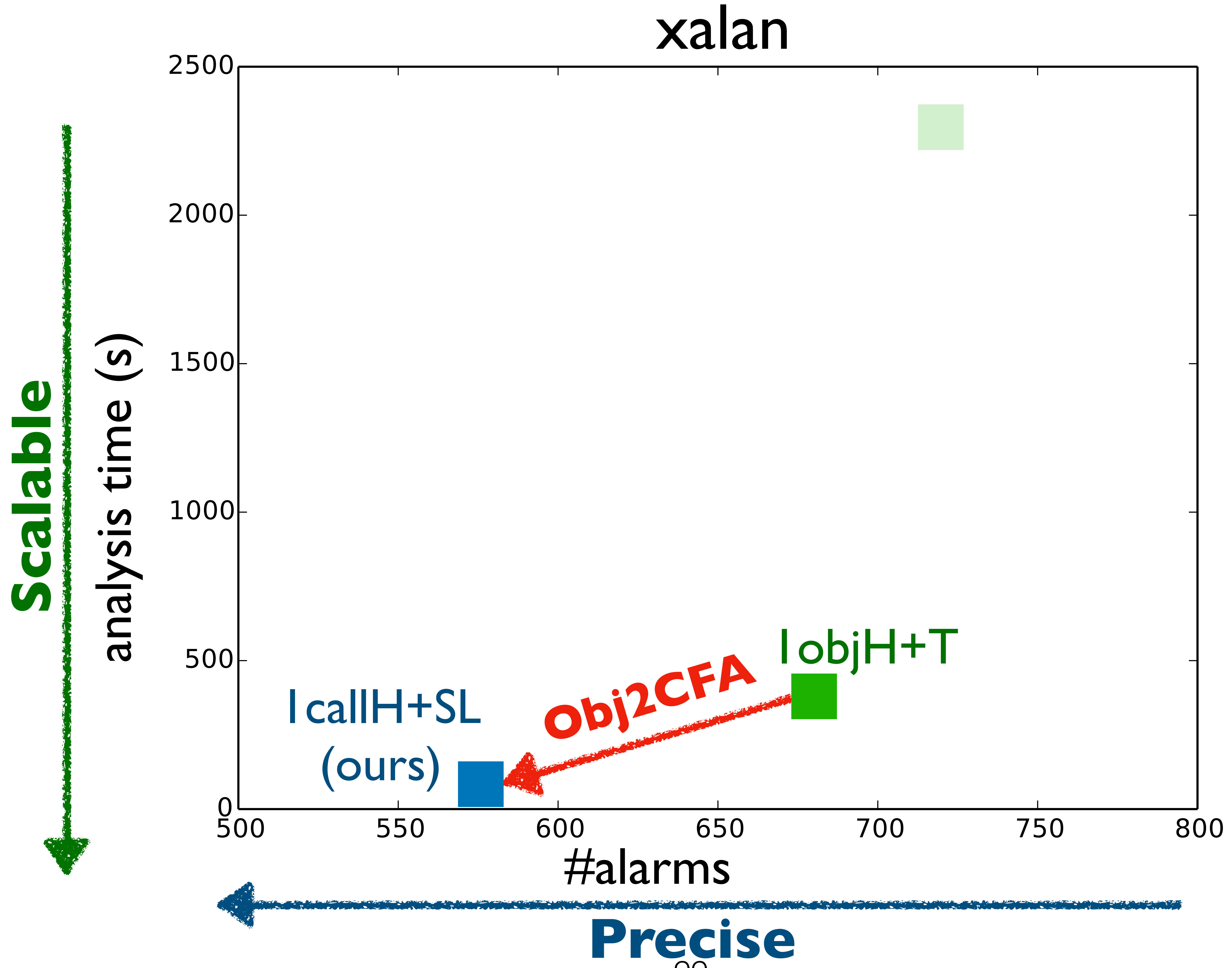
CFA wins!

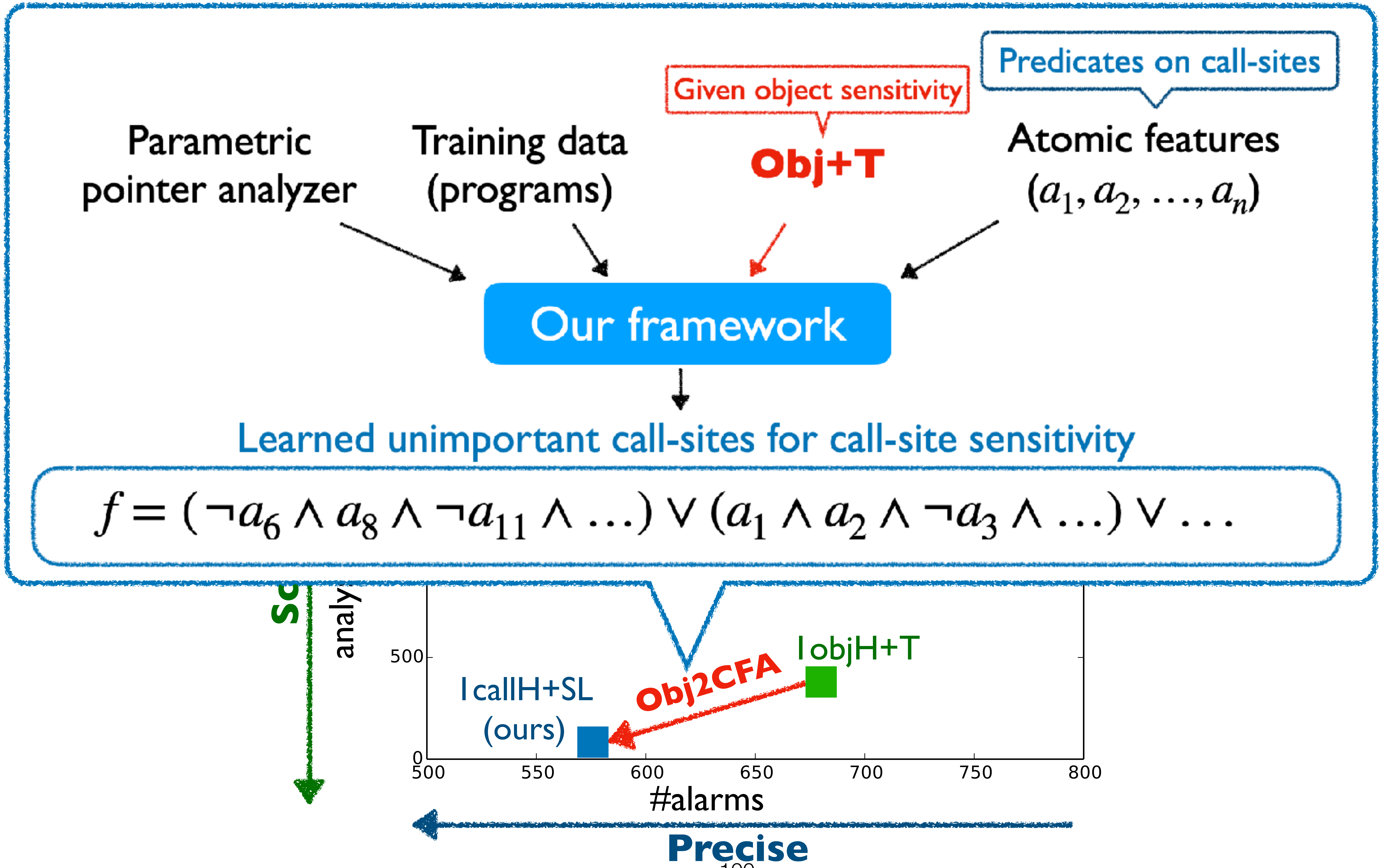
uses the allocation-site of the receiver object (a) as the context of foo. The standard k-object-sensitive analysis [Milanova et al. 2002, 2005; Smaragdakis et al. 2011] maintains a sequence of



Our Technique : Obj2CFA

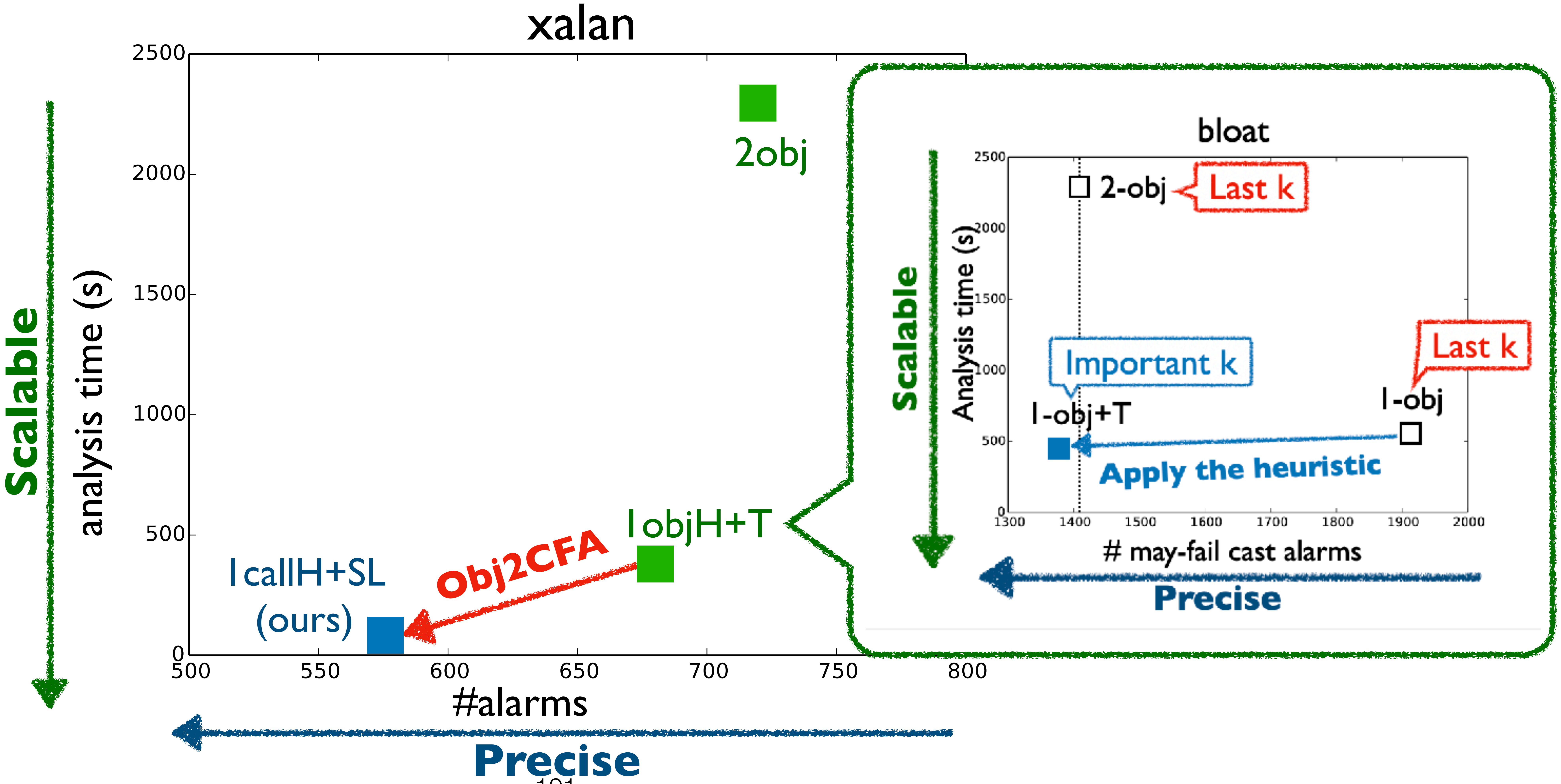
- **Obj2CFA** transforms a given object sensitivity into a more precise CFA




Predicates on call-sites
Given object sensitivity
Parametric pointer analyzer
Training data (programs)
Obj+T
Atomic features
$(a_1, a_2, \ldots, a_n)$
Our framework
Learned unimportant call-sites for call-site sensitivity
$f = (\neg a_6 \wedge a_8 \wedge \neg a_{11} \wedge \ldots) \vee (a_1 \wedge a_2 \wedge \neg a_3 \wedge \ldots) \vee \ldots$
1objH+T
Obj2CFA
1callH+SL
(ours)
500
0
500
550
600
650
700
750
800
#alarms
Precise

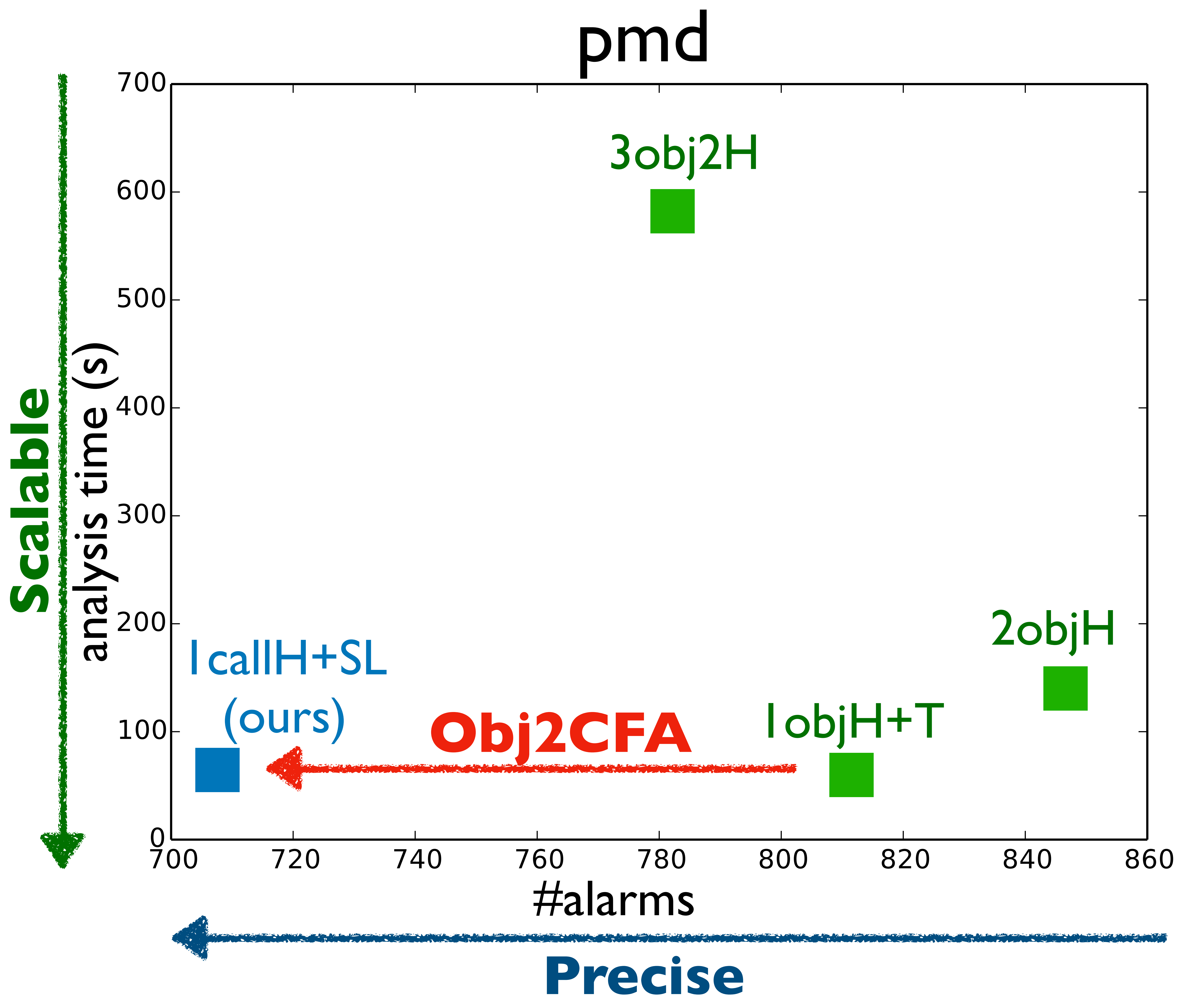
Our Technique : Obj2CFA

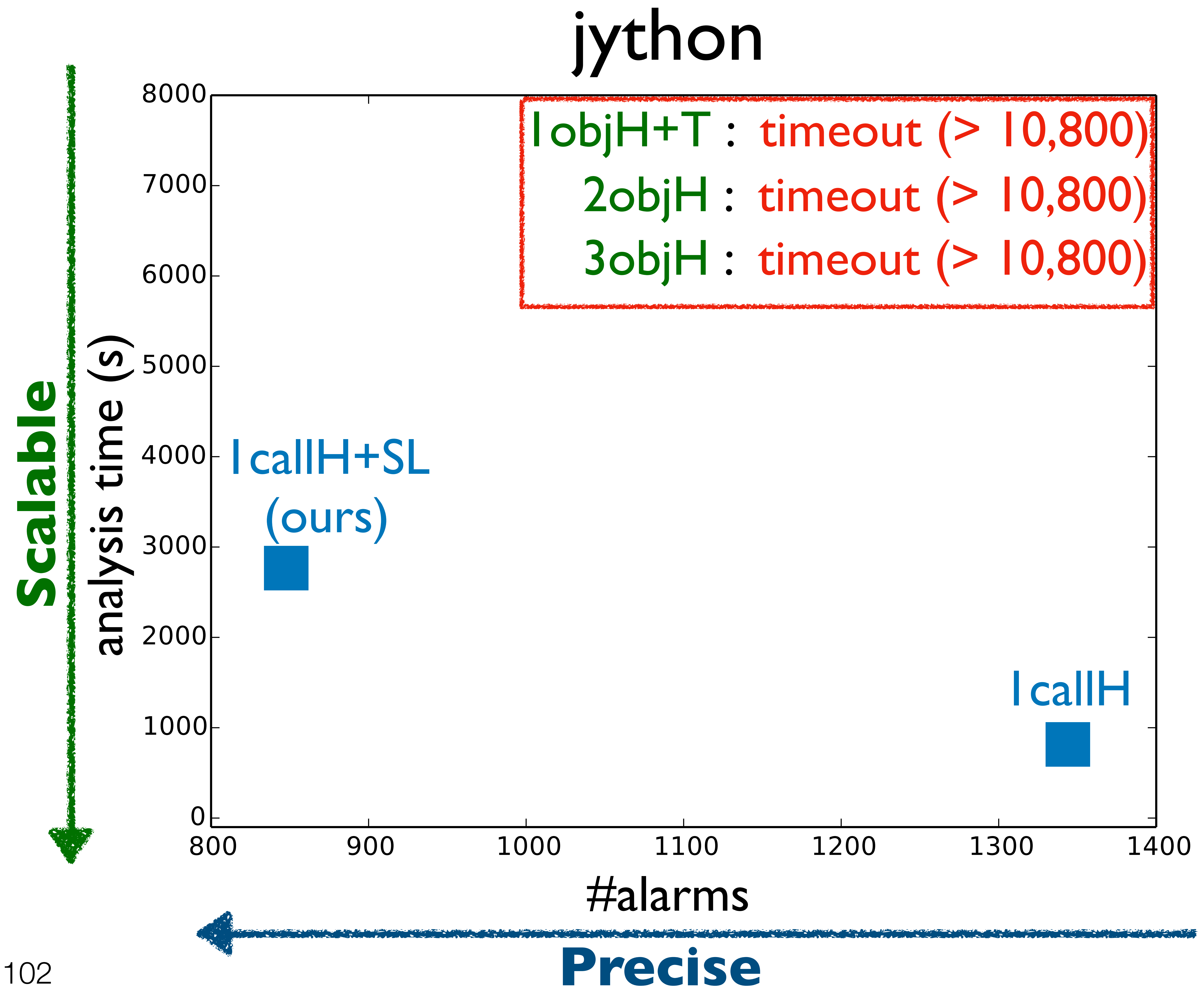
- **Obj2CFA** transforms a given object sensitivity into a more precise CFA



Call-site Sensitivity vs Object Sensitivity

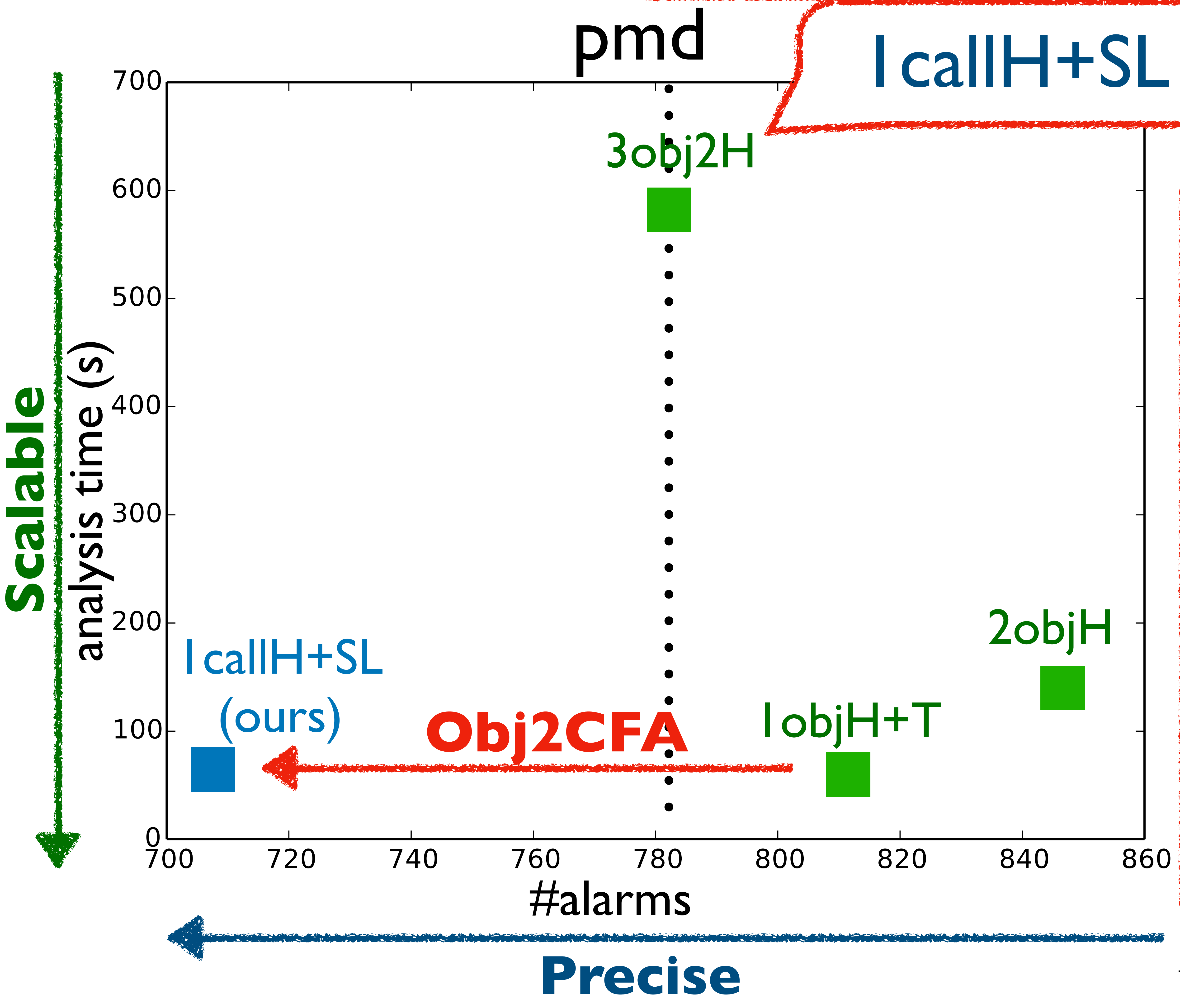
- 1callH+SL (ours) is more precise and scalable than the existing object sensitivities





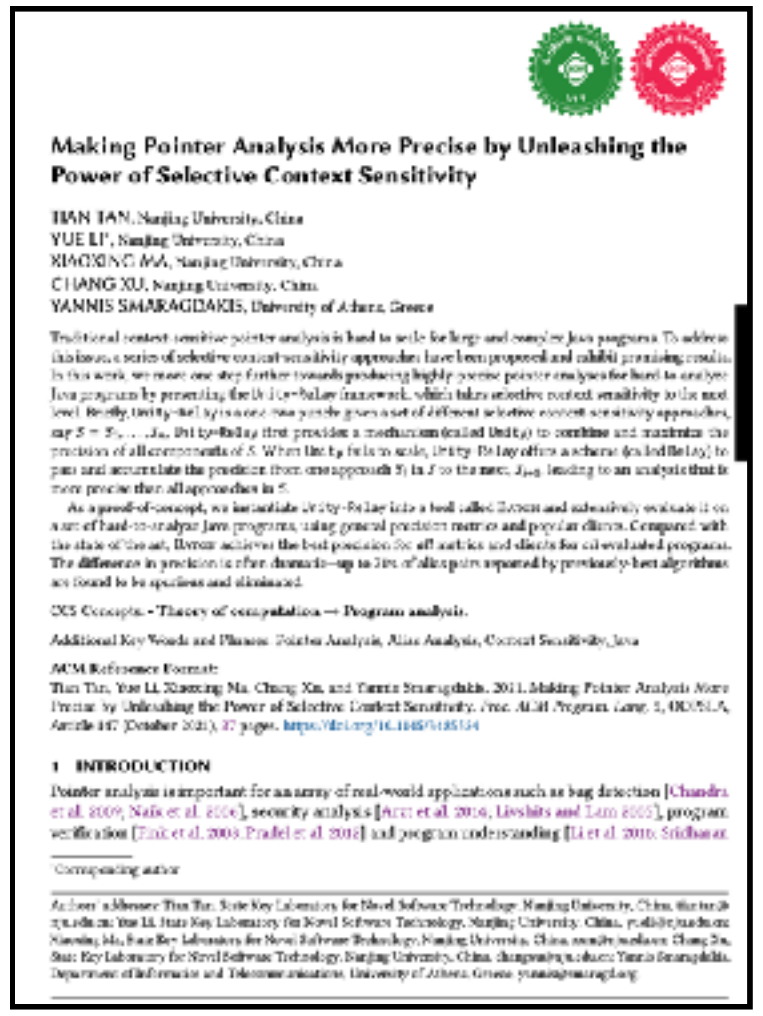
Call-site Sensitivity vs Object Sensitivity

- 1callH+SL (ours) is more precise and scalable than the existing object sensitivities



1callH+SL is even more precise than 3obj2H

Precision upper bound of recent research on object sensitivity


Making Pointer Analysis More Precise by Unleashing the Power of Selective Context Sensitivity

TIAN TAN, YUE LI, XIAOXING MA, CHANG XU, YANNIS SMARAGDAKIS

OOPLSA 2021


Precision-Preserving Yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity

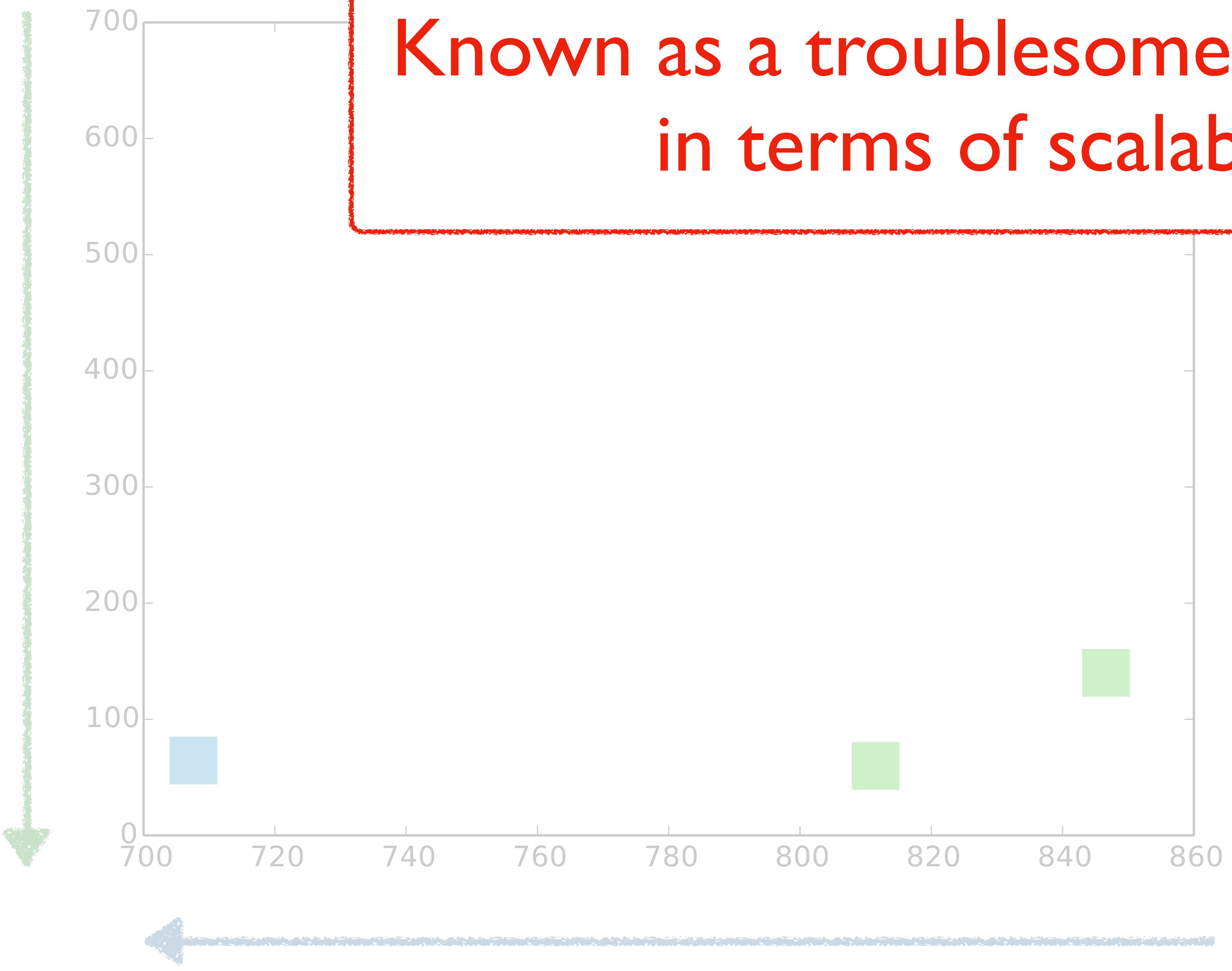
JINGBO LU, JINGLING XUE

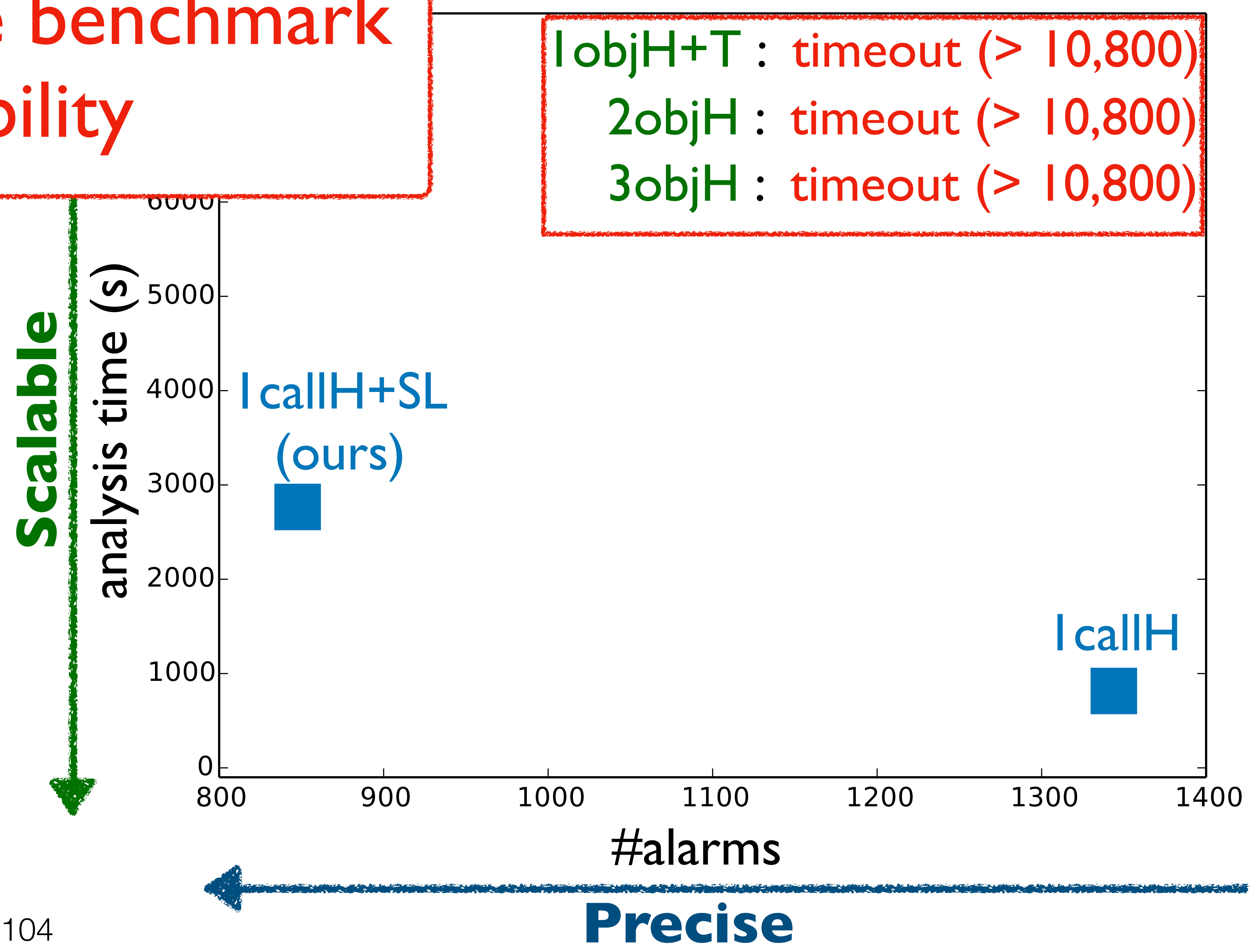
OOPLSA 2019

Call-site Sensitivity vs Object Sensitivity

- 1callH+SL (ours) is more precise and scalable than the existing object sensitivities

Known as a troublesome benchmark in terms of scalability





Call-Site vs. Object Sensitivity

	Time (s)		#may-fail-cast		#call-graph-edge	
	2-call	2-obj	2-call	2-obj	2-call	2-obj
batik	6,886	3,300	2,452	1,606	94,211	76,807
checkstyle	2,277	2,003	863	581	54,171	48,809
sunflow	5,570	1,208	2,504	1,837	100,701	89,866
findbugs	3,812	2,661	2,056	1,409	72,118	65,836
jpc	3,343	559	1,855	1,392	89,677	81,030
eclipse	1,896	146	886	546	42,872	38,151
chart	2,705	282	1,481	883	59,691	52,374
fop	5,503	1,200	1,975	1,446	79,524	71,408
xalan	1,927	1,093	919	533	48,763	44,871
bloat	5,712	3,525	1,699	1,193	58,696	53,143

For all numbers, lower is better (in terms of eff

In general
- Precision: object > call-site
- Efficiency: object > call-site

A lecture slide used in 2023

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. "*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*". TOPLAS 2020.

https://cs.nju.edu.cn/tiantan/software-analysis/PTA-CS.pdf

Call-Site vs. Object Sensitivity

	Time (s)		#may-fail-cast		#call-graph-edge	
	2-call	2-obj	2-call	2-obj	2-call	2-obj
batik	6,886	3,300	2,452	1,606	94,211	76,807
checkstyle	2,277	2,003	863	581	54,171	48,809
sunflow	5,570	1,208	2,504	1,837	100,701	89,866
findbugs	3,812	2,661	2,056	1,409	72,118	65,836
jpc	3,343	559	1,855	1,392	89,677	81,030
eclipse	1,896	146	886	546	42,872	38,151
chart	2,705	282	1,481	883	59,691	52,374
fop	5,503	1,200	1,975	1,446	79,524	71,408
xalan	1,927	1,093	919	533	48,763	44,871
bloat	5,712	3,525	1,699	1,193	58,696	53,143

For all numbers, lower is better (in terms of eff

<

In general
- Precision: object > call-site
- Efficiency: object > call-site

A lecture slide used in 2023

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. "*A Principled Approach to Selective Context Sensitivity for Pointer Analysis*". TOPLAS 2020.

https://cs.nju.edu.cn/tiantan/software-analysis/PTA-CS.pdf

1100 1200 1300 1400

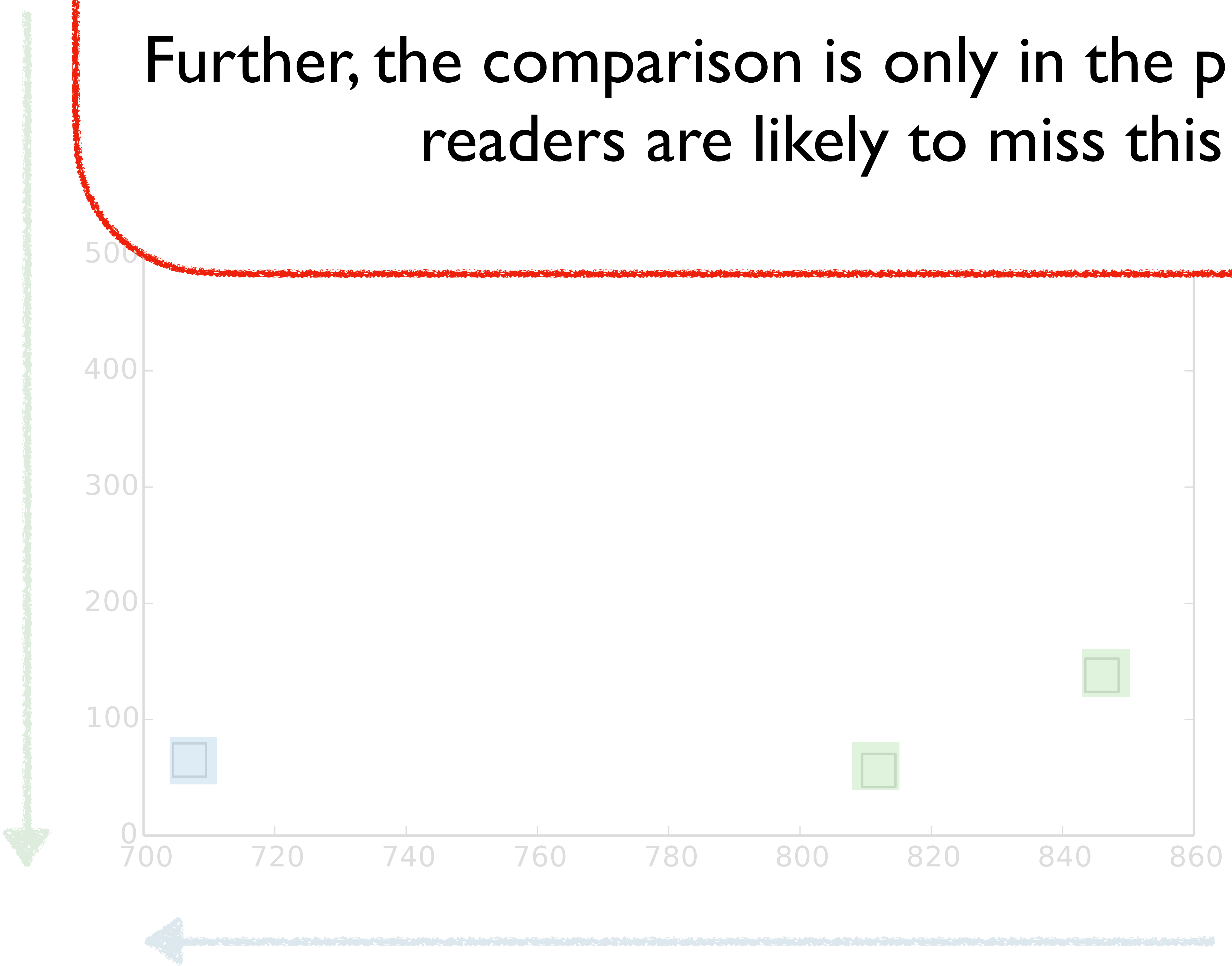
A Review Comment

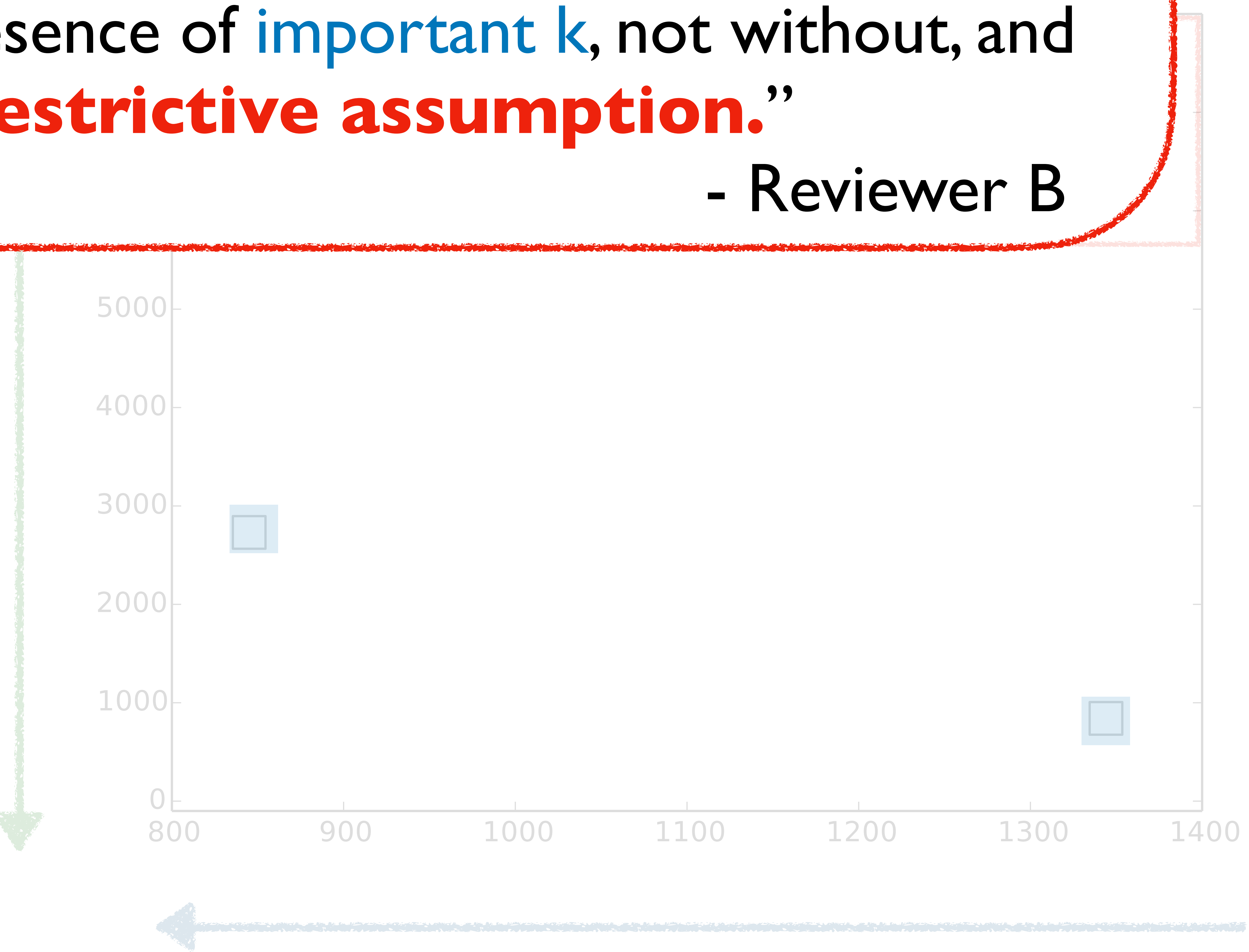
"I cannot support acceptance of this paper, due to the following major concerns:

…

Further, the comparison is only in the presence of important k, not without, and readers are likely to miss this **restrictive assumption.**"

\- Reviewer B





A Review Comment

"I cannot support acceptance of this paper, due to the following major concerns:

…

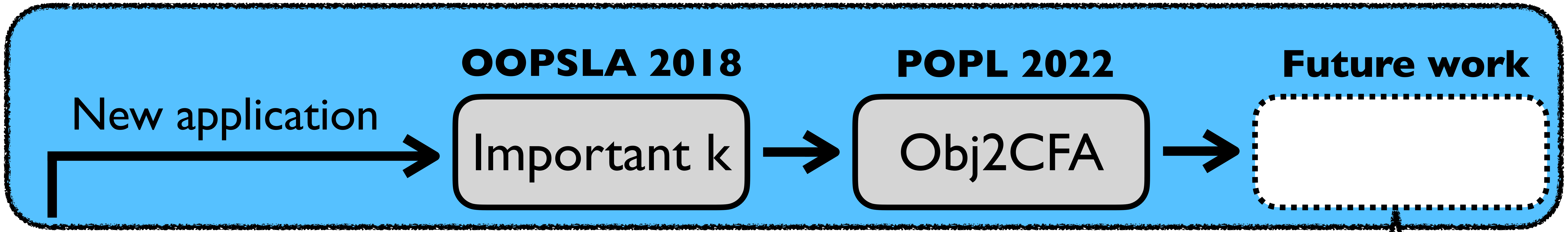
Further, the comparison is only in the presence of important k, not without, and readers are likely to miss this **restrictive assumption.**"

- Reviewer B

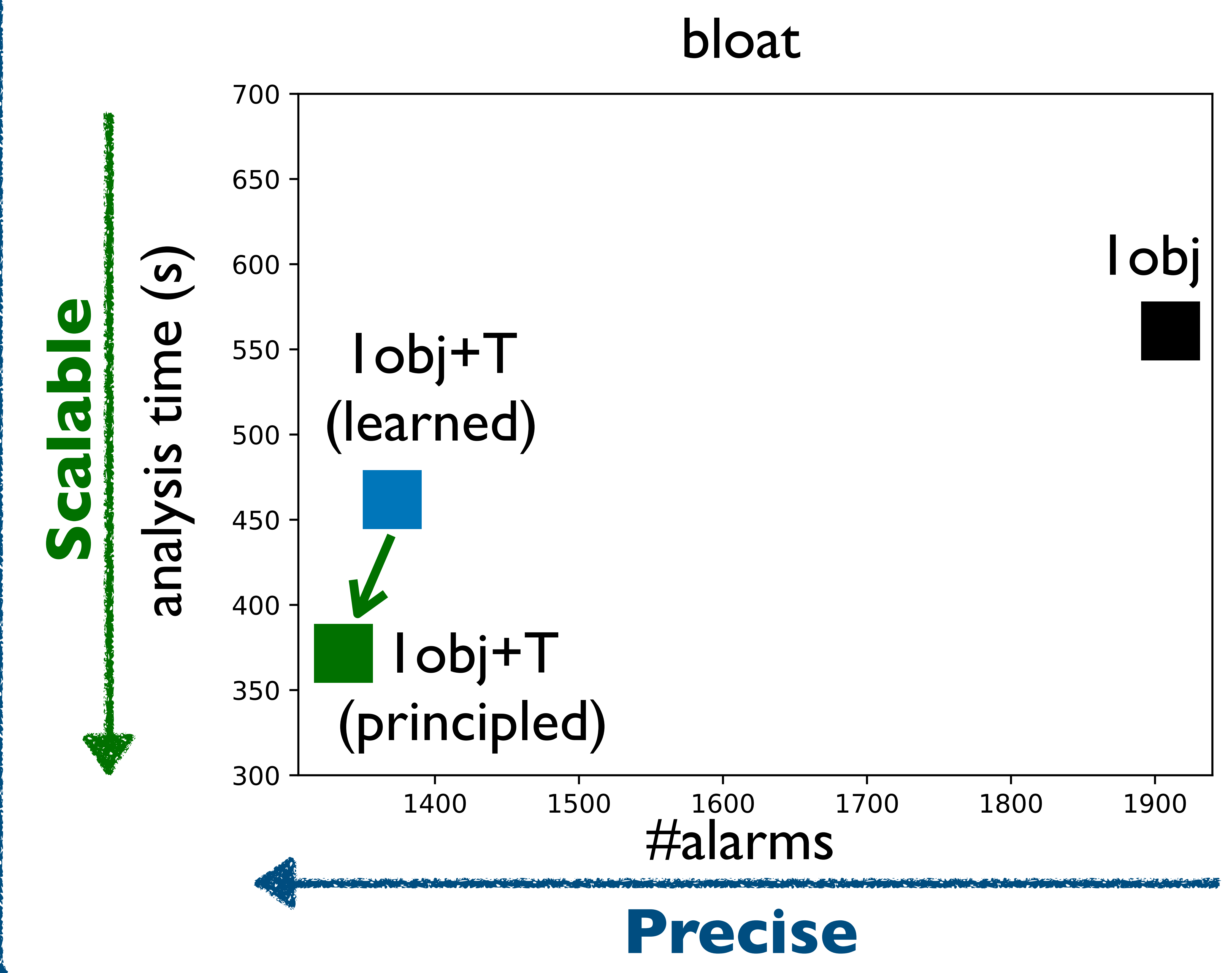
Which one is more restrictive?

Important k vs **Last k**

Establishing important k as a standard



Language	Java	JavaScript	Python	C	…
Learning heuristics	**OOPSLA'18** **POPL' 22**	**In progress**	**ToDo**	**ToDo**	
Principled heuristics	**In progress**	**ToDo**	**ToDo**	**ToDo**	



New application

Important

OOPSLA 2017

Disjunct
Learnin

Language	Java	JavaScript	Python	C	…
Learning heuristics	**OOPSLA'18** **POPL' 22**				
Principled heuristics	**In progress**	ToDo	ToDo	ToDo	

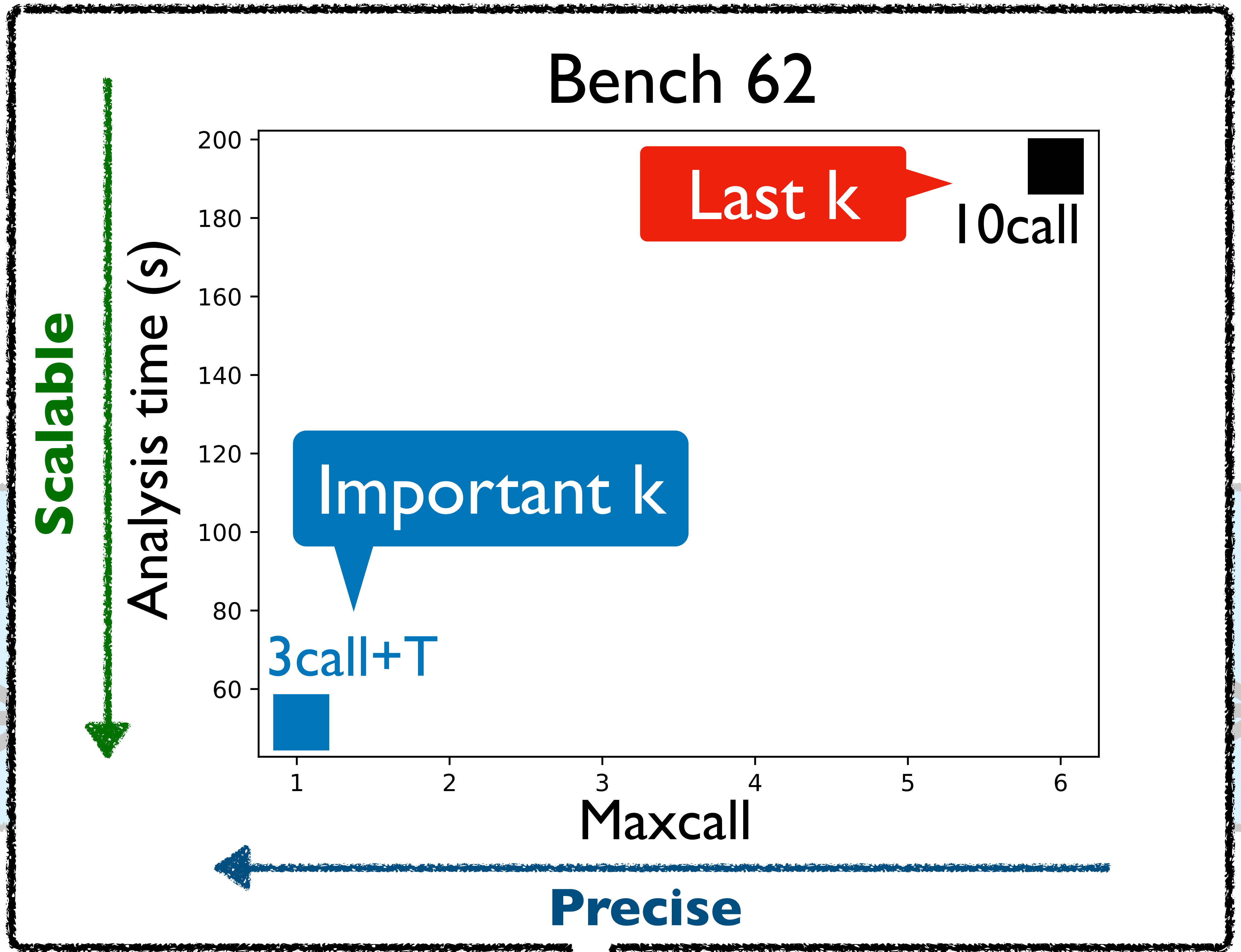
Understanding the principle of the learned heuristics

New application

OOPSLA 2017

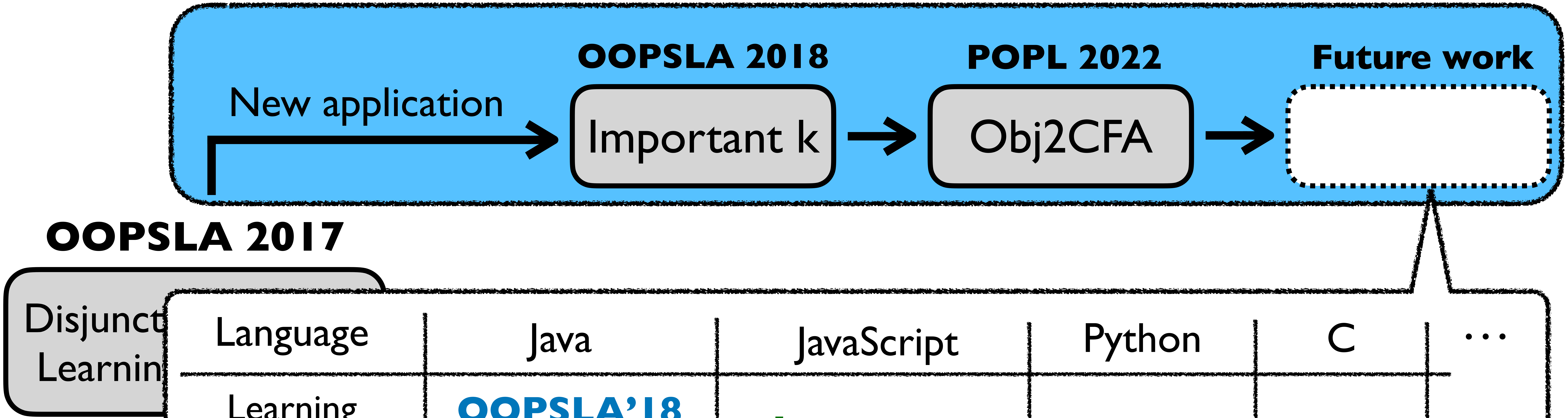
Disjunct
Learnin

Future work



Language	Java	JavaScript	Python	C	···
Learning heuristics	**OOPSLA'18** **POPL' 22**	**In progress**	**ToDo**	**ToDo**	
Principled heuristics		**ToDo**	**ToDo**	**ToDo**	

Establishing important k as a standard



Language	Java	JavaScript	Python	C	…
Learning heuristics	**OOPSLA'18 POPL' 22**	**In progress**	**ToDo**	**ToDo**	
Principled heuristics	**In progress**	**ToDo**	**ToDo**	**ToDo**	

Part 2:
PL-based Explainable Graph Machine Learning

Establishing important k as a standard
New application
OOPSLA 2018
Important k
POPL 2022
Obj2CFA
Future work
OOPSLA 2017
Disjunctive model & Learning algorithm
PL-based
Establishing a new graph machine learning method
Generalization
OOPSLA 2020
Graphick
Will be Submitted
PL4XGL
Future work


Establishing important k as a standard
OOPSLA 2018
POPL 2022
Future work
New application
Important k
Obj2CFA
PL-based Explainable Graph Machine Learning
Will be Submitted
PLDI 2024
PL4XGL

- Existing: unexplainable AI (Graph Neural Network)

Input graph data (e.g., molecule) → GNN (Black box) → Prediction (e.g., the molecule is safe)

- Social network
- Program representation
- …

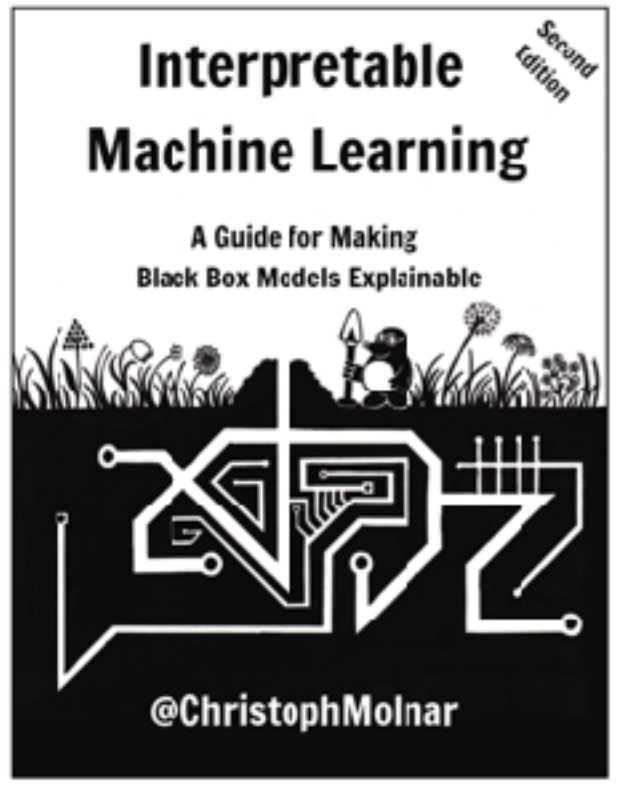
- Existing: unexplainable AI (Graph Neural Network)

Input graph data (e.g., molecule) → **GNN (Black box)** → Prediction (e.g., the molecule is safe)

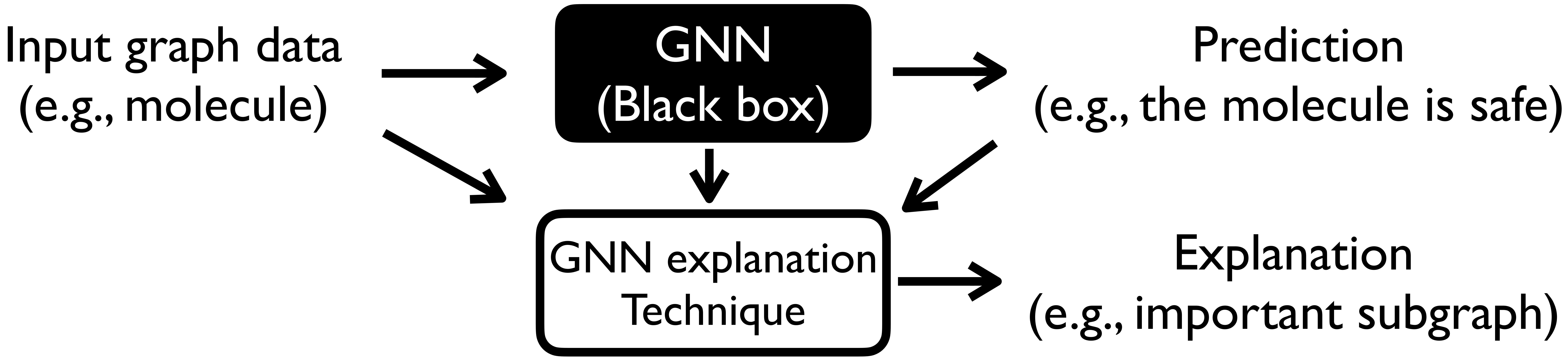
Does not explain why the prediction is made

Value of explainability is growing fast

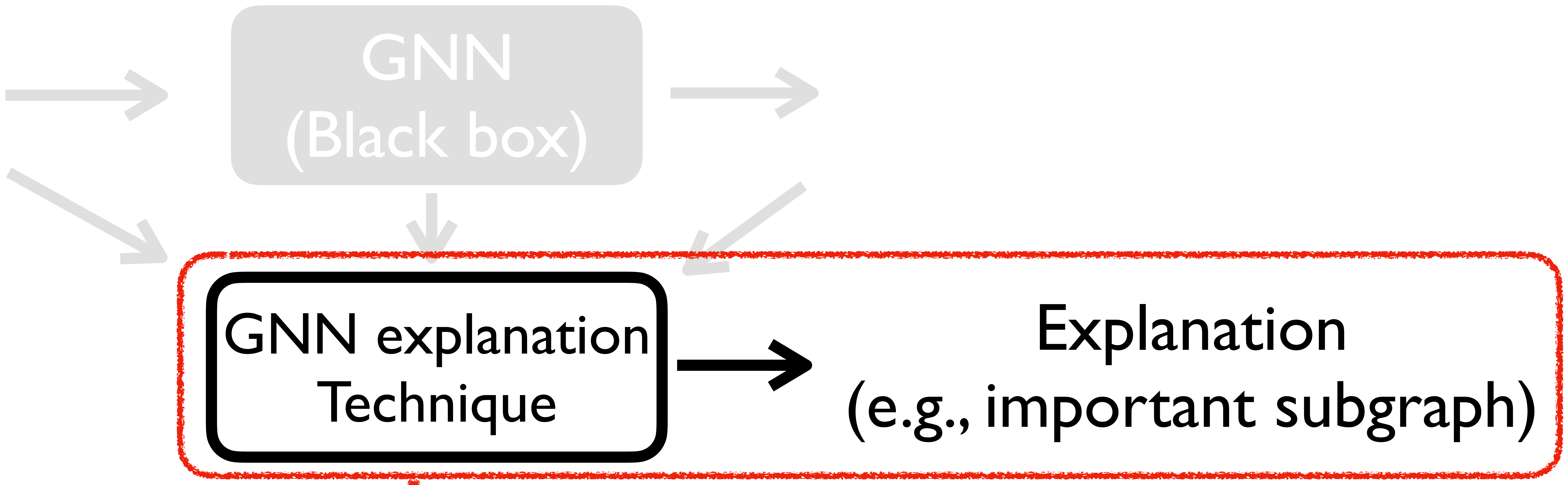
> A correct prediction only partially solves your problem. The model must also explain **why.**
>
> \- Molnar [**2022**]



- Existing: unexplainable AI (GNN) + explanation technique



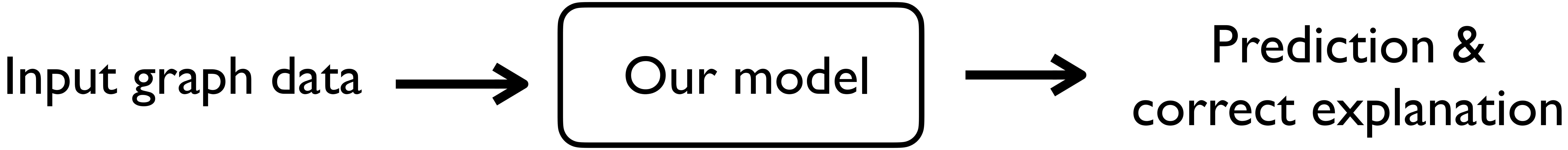
- Existing: unexplainable AI (GNN) + explanation technique



Two key problems

- Additional (expensive) explanation cost is required
- The explanations are not guaranteed to be correct

- Our new technique: PL-based inherently explainable Graph Machine learning



Syntax

Programs	$P_4 ::= \overline{\delta}$ `target` t	$\in$	$\mathbb{P} = \mathbb{D}^* \times \mathbb{T}$
Descriptions	$\delta ::= \delta_V \mid \delta_E$	$\in$	$\mathbb{D} = \mathbb{D}_V \uplus \mathbb{D}_E$
Node Descriptions	$\delta_V ::=$ `node` x `<`$\overline{\phi}$`>`$^?$	$\in$	$\mathbb{D}_V = \mathbb{X} \times \Phi^d$
Edge Descriptions	$\delta_E ::=$ `edge` (x, x) `<`$\overline{\phi}$`>`$^?$	$\in$	$\mathbb{D}_E = \mathbb{X} \times \mathbb{X} \times \Phi^c$
Target Symbols	$t ::=$ `node` x $\mid$ `edge` (x, x) $\mid$ `graph`	$\in$	$\mathbb{T} = \mathbb{X} \uplus (\mathbb{X} \times \mathbb{X}) \uplus \{\epsilon\}$
Intervals	$\phi ::= [n^?, n^?]$	$\in$	$\Phi = (\mathbb{R} \uplus \{-\infty\}) \times (\mathbb{R} \uplus \{\infty\})$
Real Numbers	$n ::=$ `0.2` $\mid$ `0.7` $\mid$ `6` $\mid$ `-8` ...	$\in$	$\mathbb{R}$
Variables	$x ::=$ `x` $\mid$ `y` $\mid$ `z` $\mid$ ...	$\in$	$\mathbb{X}$

Semantics

$$[\![<\phi_1, \ldots, \phi_k>]\!] : \mathcal{P}(\mathbb{R}^k) = \{\ \mathbf{f} \mid \mathbf{f} = (f_1, \ldots, f_k) \wedge \forall i.\ \phi_i = [a, b] \Rightarrow a \le f_i \le b \}$$

$$[\![\texttt{node}\ x <\overline{\phi}>]\!] : \mathcal{P}(\mathbb{G} \times \mathbb{H}) = \{ (G, \eta) \mid v = \eta(x) \wedge \mathbf{f}_v^G \in [\![<\overline{\phi}>]\!] \}$$

$$[\![\texttt{edge}\ (x, y) <\overline{\phi}>]\!] : \mathcal{P}(\mathbb{G} \times \mathbb{H}) = \{ (G, \eta) \mid e \in E \wedge e = (\eta(x), \eta(y)) \wedge \mathbf{f}_e^G \in [\![<\overline{\phi}>]\!] \}$$

$$[\![\delta_1 \delta_2 \ldots \delta_k]\!] : \mathcal{P}(\mathbb{G} \times \mathbb{H}) = \{ (G, \eta) \mid \forall i.\ (G, \eta) \in [\![\delta_i]\!] \}$$

$$[\![\overline{\delta}\ \texttt{target node}\ x]\!] : \mathcal{P}(\mathbb{G} \times V) = \{ (G, v) \mid \exists (G, \eta) \in [\![\overline{\delta}]\!].\ v = \eta(x) \}$$

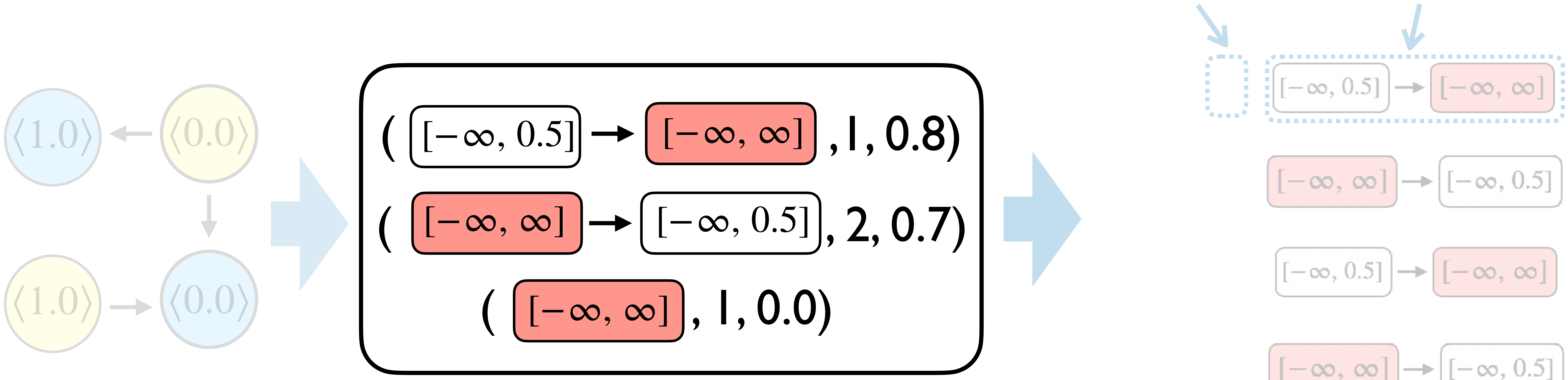
$$[\![\overline{\delta}\ \texttt{target edge}\ (x, y)]\!] : \mathcal{P}(\mathbb{G} \times E) = \{ (G, e) \mid \exists (G, \eta) \in [\![\overline{\delta}]\!].\ e = (\eta(x), \eta(y)) \}$$

$$[\![\overline{\delta}\ \texttt{target graph}]\!] : \mathcal{P}(\mathbb{G}) = \{\ G \mid \exists (G, \eta) \in [\![\overline{\delta}]\!] \}$$

Input graph data → **Our model** → Prediction & correct explanation

n1
n2
⟨1.0⟩
⟨0.0⟩
⟨1.0⟩
⟨0.0⟩
n4
n3
([−∞, 0.5] → [−∞, ∞] , 1, 0.8)
([−∞, ∞] → [−∞, 0.5] , 2, 0.7)
([−∞, ∞] , 1, 0.0)
Prediction
Explanation
n1: (1, [−∞, 0.5] → [−∞, ∞])
n2: (2, [−∞, ∞] → [−∞, 0.5])
n3: (1, [−∞, 0.5] → [−∞, ∞])
n4: (2, [−∞, ∞] → [−∞, 0.5])
Graph data
Our model
Classification result


= label 1
= label 2
n1
n2
n3
n4
⟨1.0⟩
⟨0.0⟩
⟨0.0⟩
⟨1.0⟩
Graph data
[−∞, 0.5]
[−∞, ∞]
[−∞, ∞]
[−∞, 0.5]
[−∞, ∞]



Our model

node x ⟨[-∞,∞]⟩
node y ⟨[-∞,0.5]⟩
edge (y, x)
target node x
A program in our language
y
x
[−∞, 0.5]
[−∞, ∞]


node x ⟨[-∞,∞]⟩
node y ⟨[-∞,0.5]⟩
edge (y, x)
target node x
y
x
[−∞, 0.5]
[−∞, ∞]
"Nodes having a predecessor whose feature value is equal or less than 0.5"


Described nodes will be classified into label 1
([−∞, 0.5] → [−∞, ∞] ,1, 0.8)
“Nodes having a predecessor whose feature value is equal or less than 0.5”
⟨1.0⟩
⟨0.0⟩
[−∞, 0.5] → [−∞, ∞]
[−∞, ∞] → [−∞, 0.5]
[−∞, ∞] → [−∞, 0.5]


Described nodes will be classified into label 1
([−∞, 0.5] → [−∞, ∞] , 1, 0.8)
Score of the program
"Nodes having a predecessor whose feature value is equal or less than 0.5"

n1
n2
⟨1.0⟩
⟨0.0⟩
⟨1.0⟩
⟨0.0⟩
n4
n3
Graph data
([−∞, 0.5] → [−∞, ∞] , 1, 0.8)
[−∞, ∞] → [−∞, 0.5]
[−∞, ∞]
Prediction
Explanation
n1: (1, [−∞, 0.5] → [−∞, ∞])
[−∞, ∞] → [−∞, 0.5]
n3: (1, [−∞, 0.5] → [−∞, ∞])
[−∞, ∞] → [−∞, 0.5]
Classification result

"Nodes having a successor whose feature value is equal or less than 0.5"
Prediction
Explanation
n1
n2
⟨1.0⟩
⟨0.0⟩
⟨1.0⟩
⟨0.0⟩
n4
n3
Graph data
[−∞, 0.5] → [−∞, ∞]
([−∞, ∞] → [−∞, 0.5], 2, 0.7)
[−∞, ∞]
[−∞, 0.5] → [−∞, ∞]
n2: (2, [−∞, ∞] → [−∞, 0.5])
[−∞, 0.5] → [−∞, ∞]
n4: (2, [−∞, ∞] → [−∞, 0.5])

Prediction
Explanation
n1
n2
n3
n4
⟨1.0⟩
⟨0.0⟩
⟨1.0⟩
⟨0.0⟩
Graph data
$[-\infty, 0.5] \rightarrow [-\infty, \infty]$
$([-\infty, \infty] \rightarrow [-\infty, 0.5], 2, 0.7)$
$([-\infty, \infty], 1, 0.0)$
Model classifies nodes with a better scored one
$[-\infty, 0.5] \rightarrow [-\infty, \infty]$
n2: $(2, [-\infty, \infty] \rightarrow [-\infty, 0.5])$
$[-\infty, 0.5] \rightarrow [-\infty, \infty]$
n4: $(2, [-\infty, \infty] \rightarrow [-\infty, 0.5])$

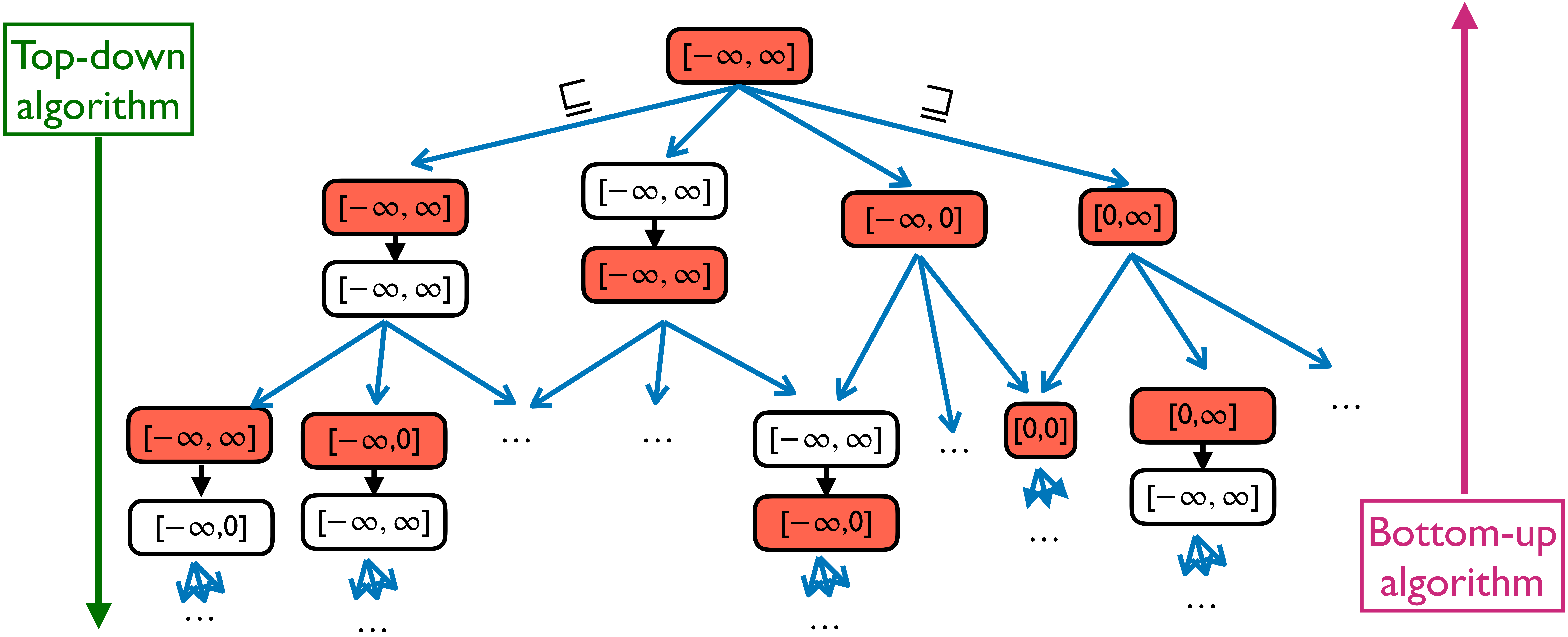
No additional explanation cost
Explanations are guaranteed to be correct
= label 1
= label 2
n1
n2
n3
n4
⟨1.0⟩
⟨0.0⟩
⟨1.0⟩
⟨0.0⟩
([−∞, 0.5] → [−∞, ∞] , 1, 0.8)
([−∞, ∞] → [−∞, 0.5], 2, 0.7)
([−∞, ∞], 1, 0.0)
Prediction
Explanation
n1: (1, [−∞, 0.5] → [−∞, ∞])
n2: (2, [−∞, ∞] → [−∞, 0.5])
n3: (1, [−∞, 0.5] → [−∞, ∞])
n4: (2, [−∞, ∞] → [−∞, 0.5])
Graph data
Our model
Classification result

Training data
Our learning algorithm
Learning objective:
Generate high-quality programs
n1
n2
⟨1.0⟩
⟨0.0⟩
⟨1.0⟩
⟨0.0⟩
n4
n3
([−∞, 0.5] → [−∞, ∞] , 1, 0.8)
([−∞, ∞] → [−∞, 0.5] , 2, 0.7)
([−∞, ∞] , 1, 0.0)
n1: (1, [−∞, 0.5] → [−∞, ∞])
n2: (2, [−∞, ∞] → [−∞, 0.5])
n3: (1, [−∞, 0.5] → [−∞, ∞])
n4: (2, [−∞, ∞] → [−∞, 0.5])
Graph data
Our model
Classification result

Training data
Top-down learning algorithm
Bottom-up learning algorithm
Our learning algorithm
Learning objective:
Generate high-quality programs
n1
n2
⟨1.0⟩
⟨0.0⟩
⟨1.0⟩
⟨0.0⟩
n4
n3
([−∞, 0.5] → [−∞, ∞] , 1, 0.8)
([−∞, ∞] → [−∞, 0.5] , 2, 0.7)
([−∞, ∞] , 1, 0.0)
n1: (1, [−∞, 0.5] → [−∞, ∞])
n2: (2, [−∞, ∞] → [−∞, 0.5])
n3: (1, [−∞, 0.5] → [−∞, ∞])
n4: (2, [−∞, ∞] → [−∞, 0.5])
Graph data
Our model
Classification result

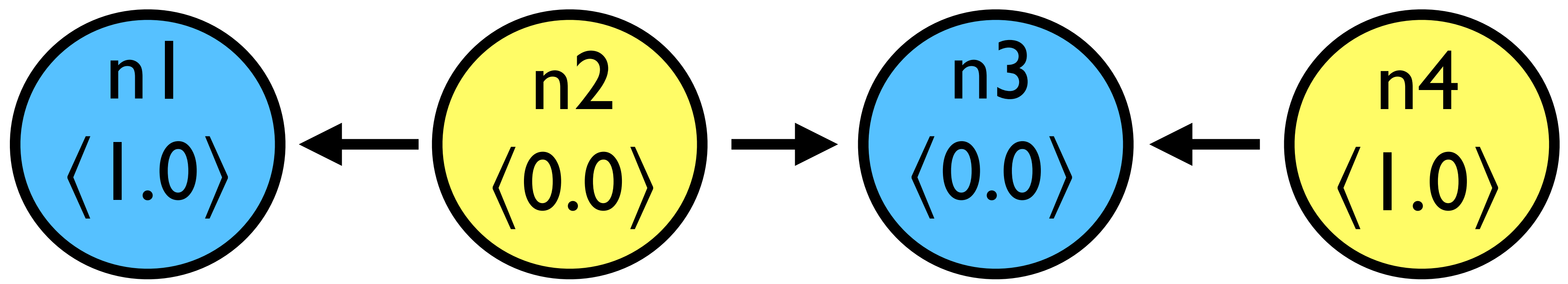
Orders Between programs

- A bigger program is more general that chooses a more nodes



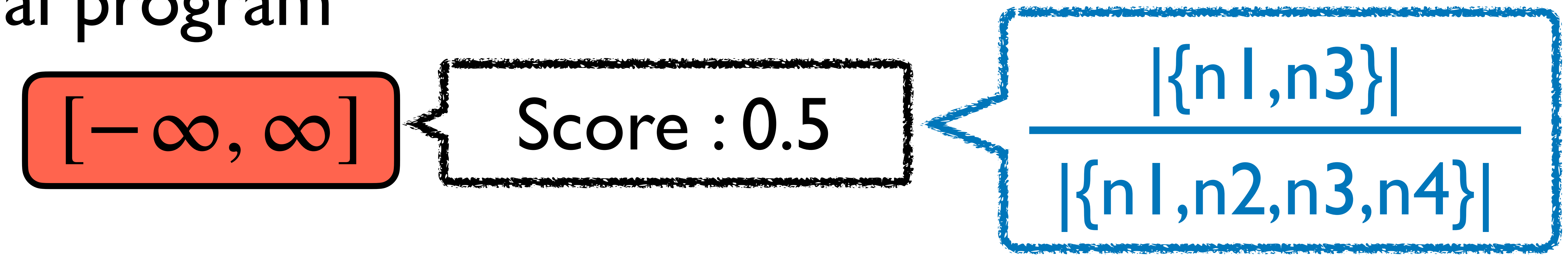
Top-down Learning Algorithm

Training graph :



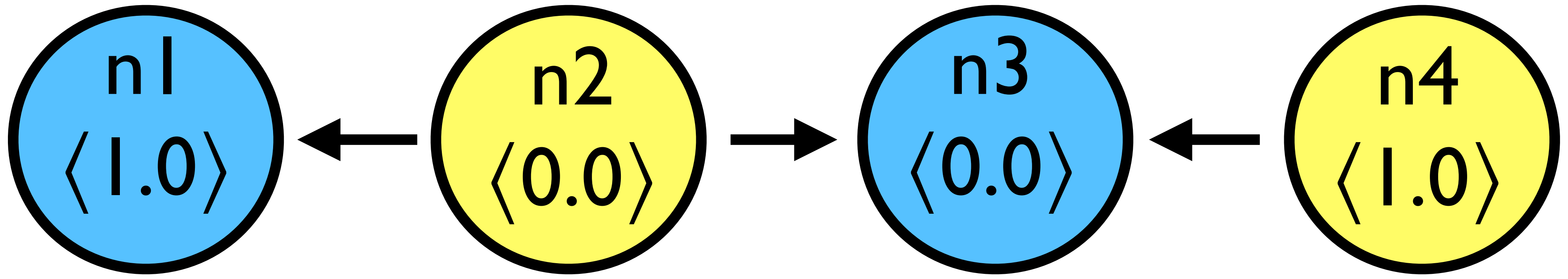
Target label = 1 (◯)

(1) Starts from the most general program



Top-down Learning Algorithm

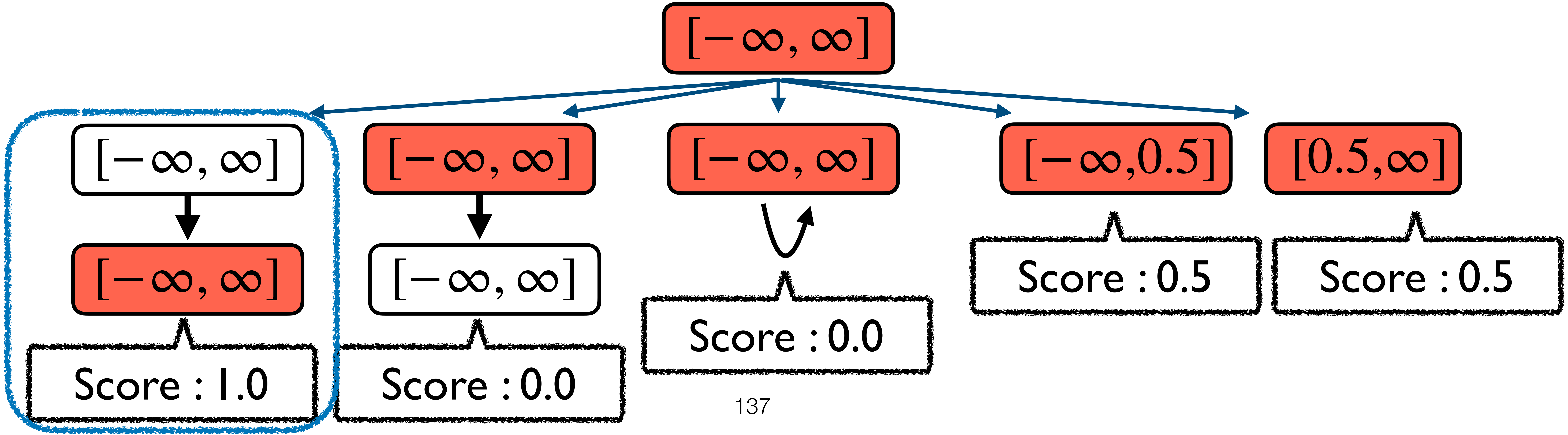
Training graph :



Target label = 1 (◯)

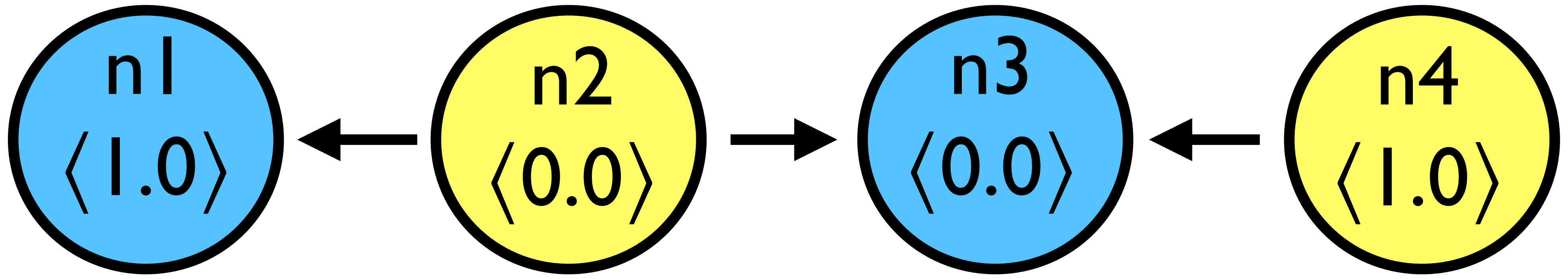
(1) Starts from the most general program $[-\infty, \infty]$ Score : 0.5

(2) Enumerate possible specified programs and choose a better scored one.



Top-down Learning Algorithm

Training graph :



Target label = 1 (●)

(1) Starts from the most general program $[-\infty, \infty]$ Score : 0.5

(2) Enumerate possible specified programs and choose a better scored one.

$[-\infty, \infty] \rightarrow [-\infty, \infty]$ Score : 1.0

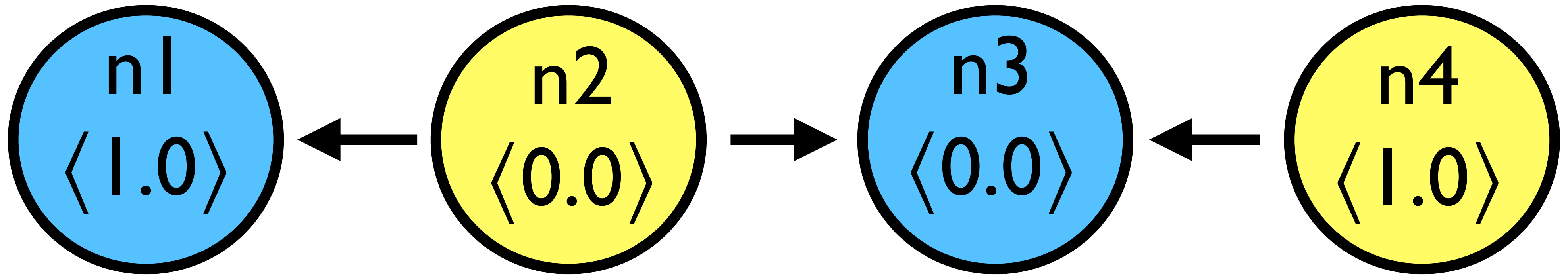
(3) Repeat (2) until no better program is enumerated

(4) Return the current program

$([-\infty, \infty] \rightarrow [-\infty, \infty], 1, 1.0)$

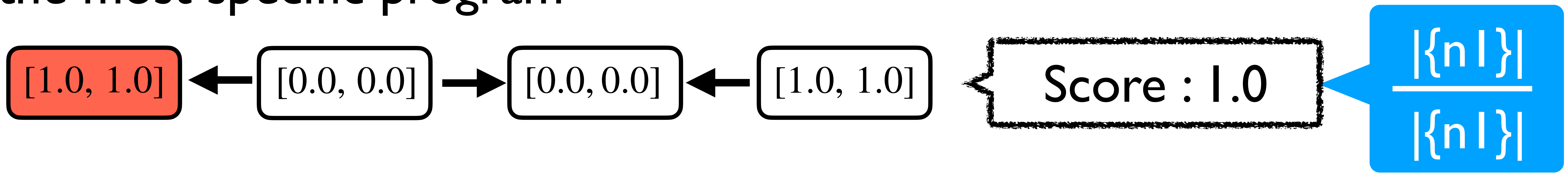
Bottom-up Learning Algorithm

Training graph :



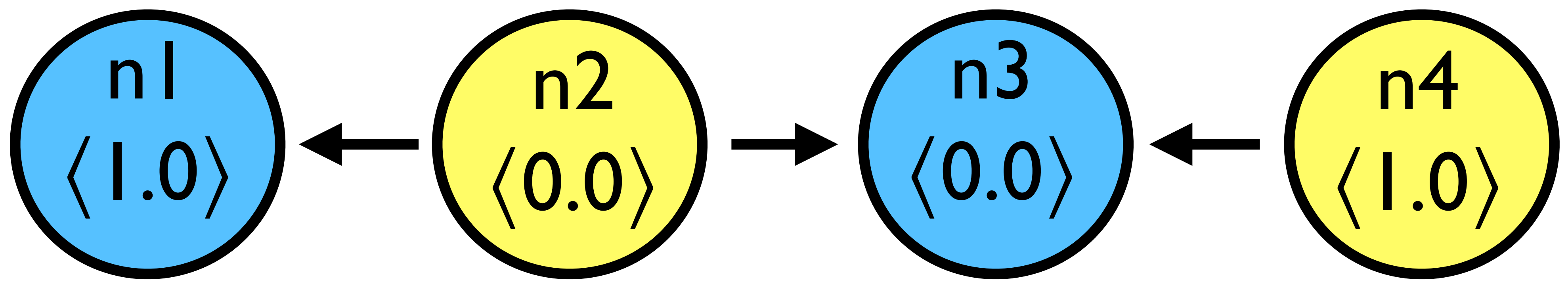
Target label = 1 ( )

(1) Starts from the most specific program



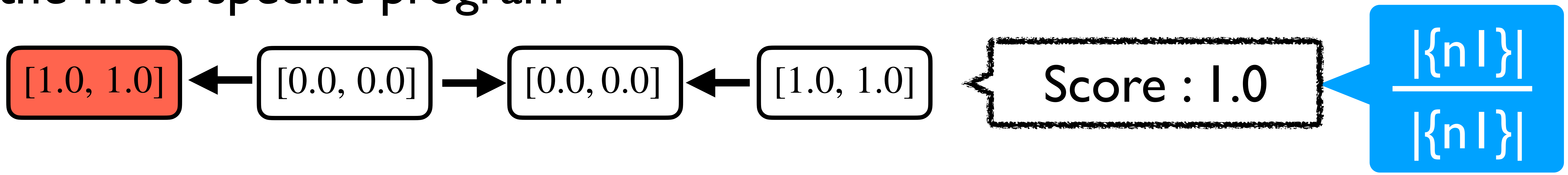
Bottom-up Learning Algorithm

Training graph :

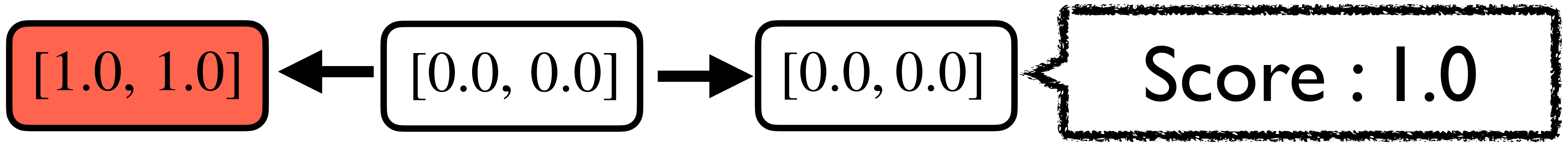


Target label = 1 (○)

(1) Starts from the most specific program

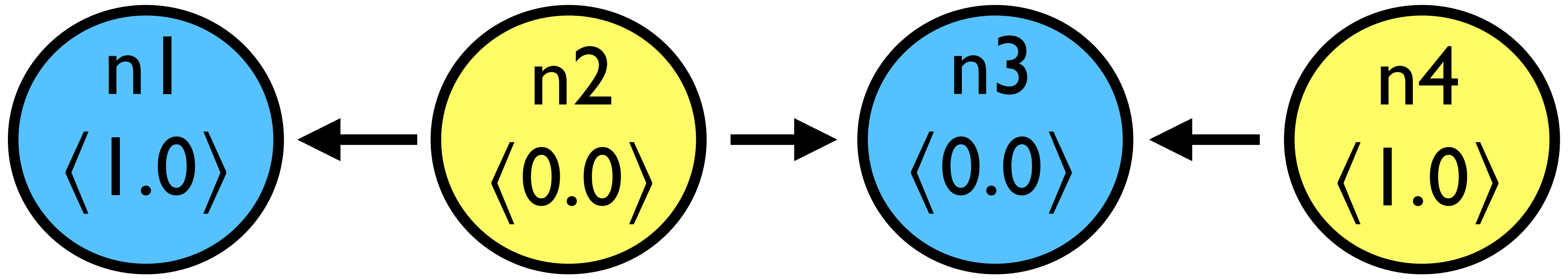


(2) Enumerate possible generalized programs and choose an equal or better scored one



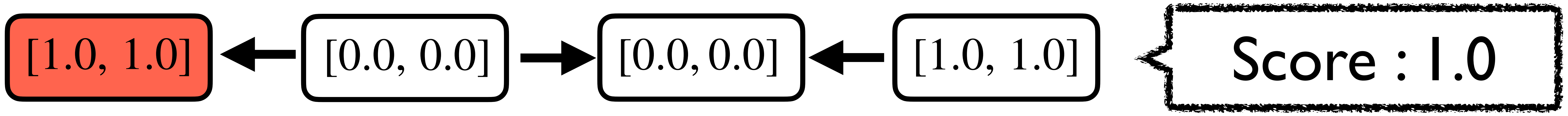
Bottom-up Learning Algorithm

Training graph :

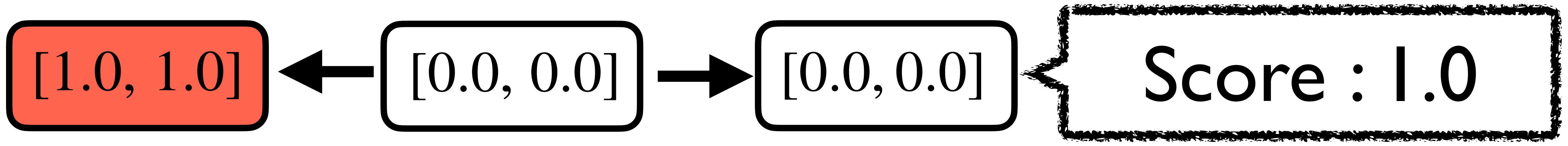


Target label = 1 (◯)

(1) Starts from the most specific program

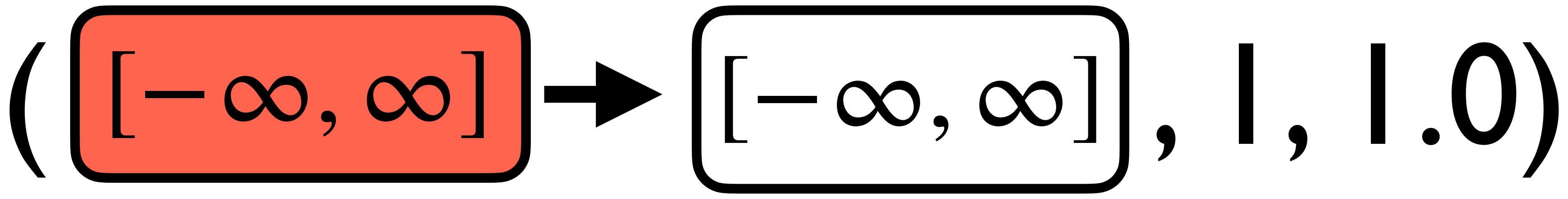


(2) Enumerate possible generalized programs and choose an equal or better scored one



(3) Repeat (2) until all the enumerated programs have lower score

(4) Return the current program

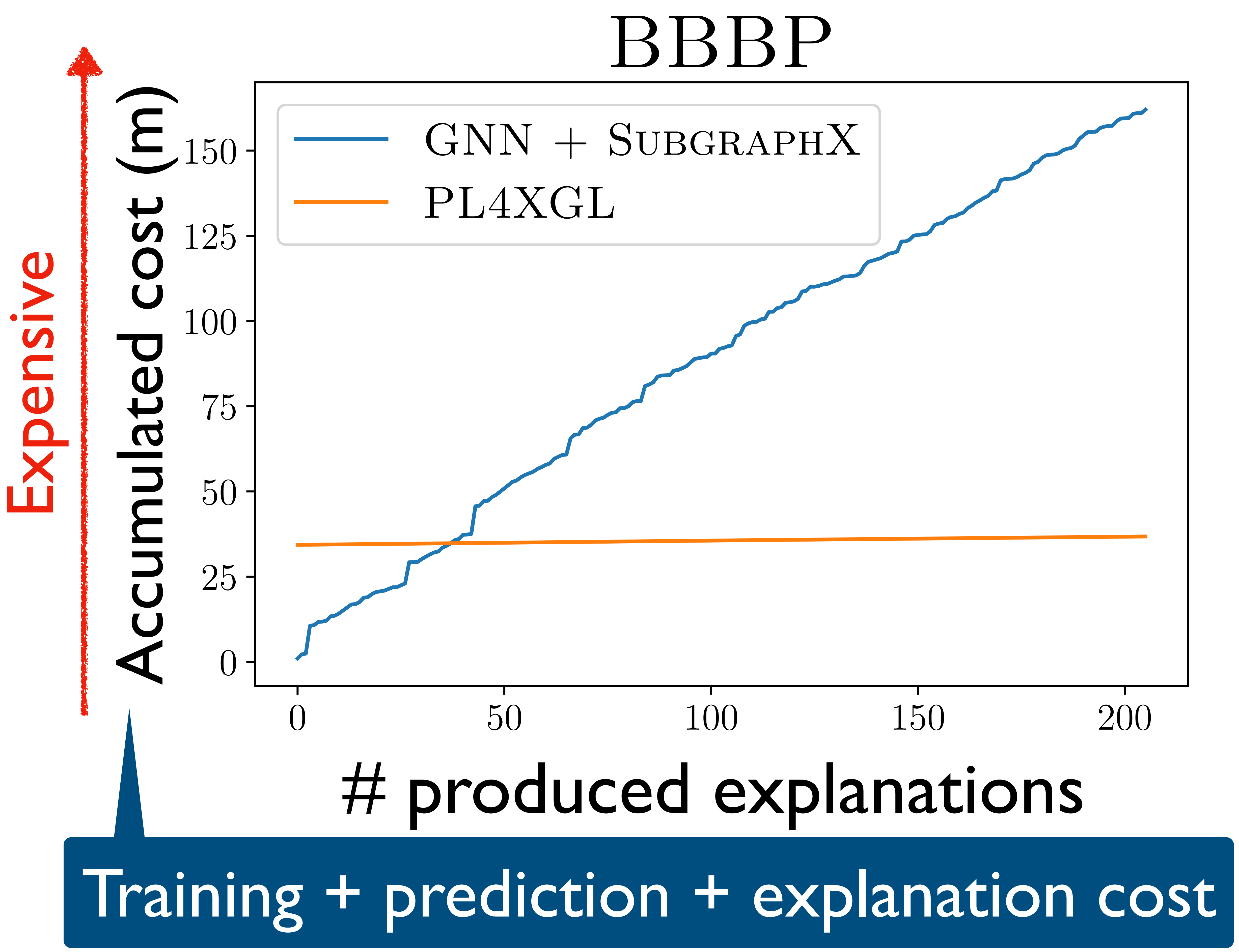


Accuracy Comparison

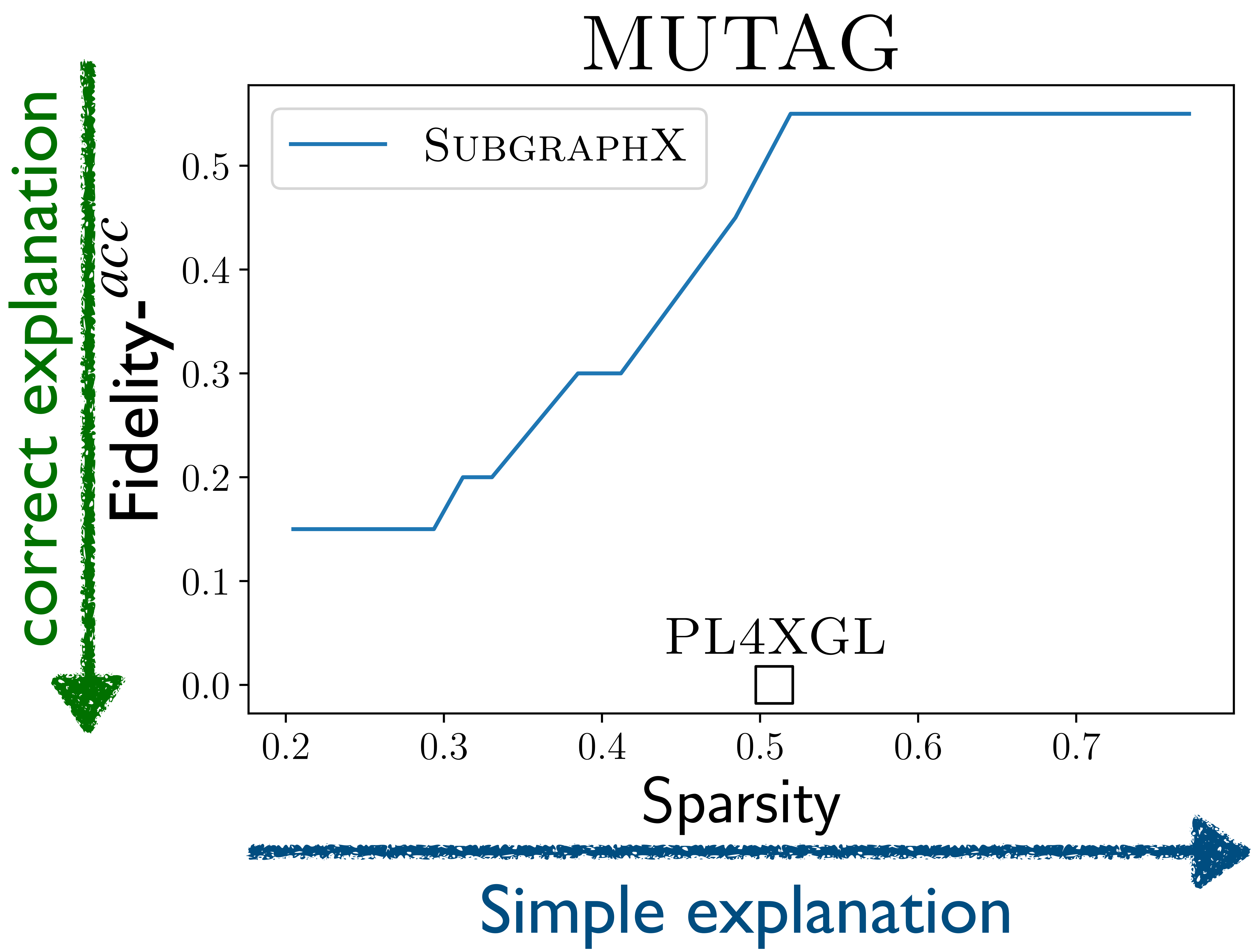
- We split the dataset into 8:1:1 for training, validation, and testing

	Graph classification			Node classification		
	MUTAG	**BBBP**	**BACE**	**Texas**	**Cornell**	**Wisconsin**
GCN	80.0	83.6	78.4	64.0	67.7	58.9
GAT	89.0	82.3	52.4	49.6	50.0	61.1
GIN	91.0	86.2	**80.9**	56.0	50.0	61.1
DGCN	N/A	N/A	N/A	**86.6**	86.6	**96.0**
PL4XGL (Ours)	**100.0**	**86.8**	**80.9**	83.3	**88.8**	88.0

Cost comparison

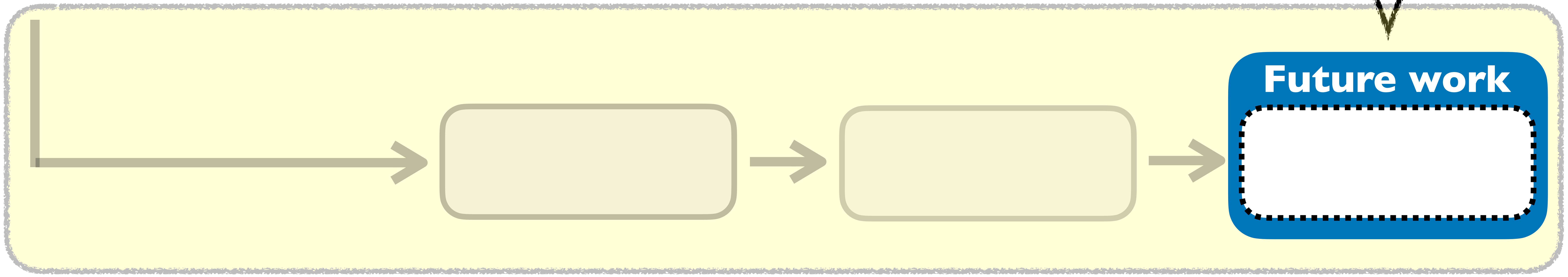


Explainability comparison



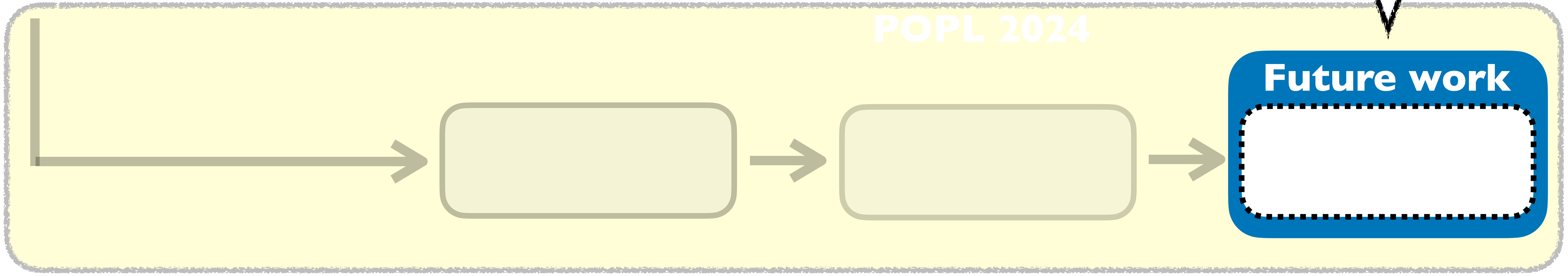
1. Improving our approach

- Developing a better language (e.g., more expressive constraints)
- Developing a better learning algorithm
- …



2. Applying our approach to SE

- Applying our approach to various SE problems
  - Applying our approach to fault localization (working on it)
  - Applying our approach to data-driven static analysis
  - …



Our Technique: Graphick

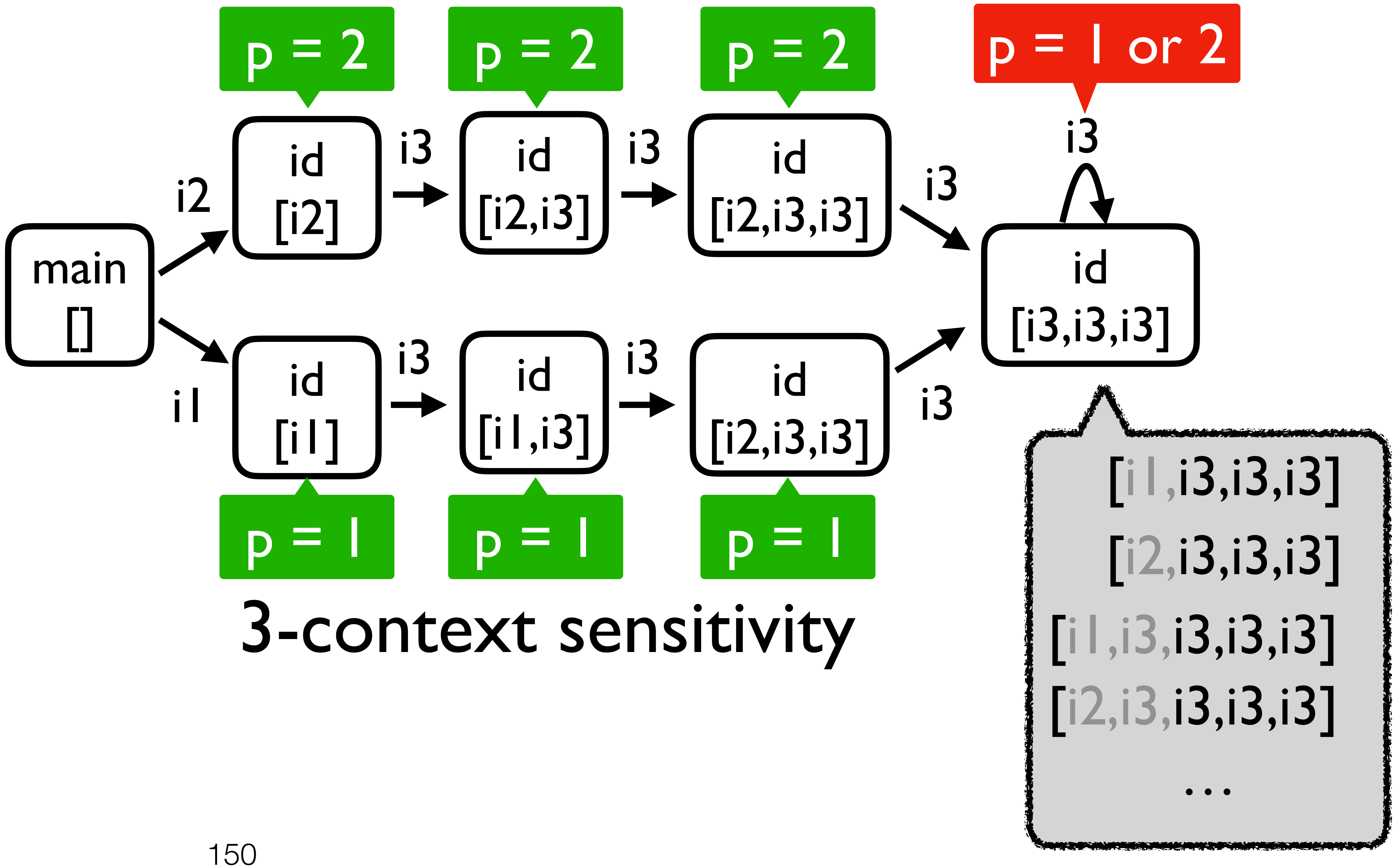
Establishing important k as a standard
OOPSLA 2018
POPL 2022
Future work
New application
Important k
Obj2CFA
OOPSLA 2017
Disjunctive model & Learning algorithm
Establishing a new graph machine learning method
OOPSLA 2020
Will be Submitted
Future work
Graphick
PL4XGL
Generalization
Thank you!

Backup Slides

Problem of Last-k

- An example showing a problem of last-k context abstraction

```
id(p, i){
  if i > 0:
    return id(p, i-1);//i3
  else:
    return p;}

main(){
  i = input();
  v2 = id(2,i);//i2
  v1 = id(1,i);//i1
  assert (v1 != v2);}//query
```



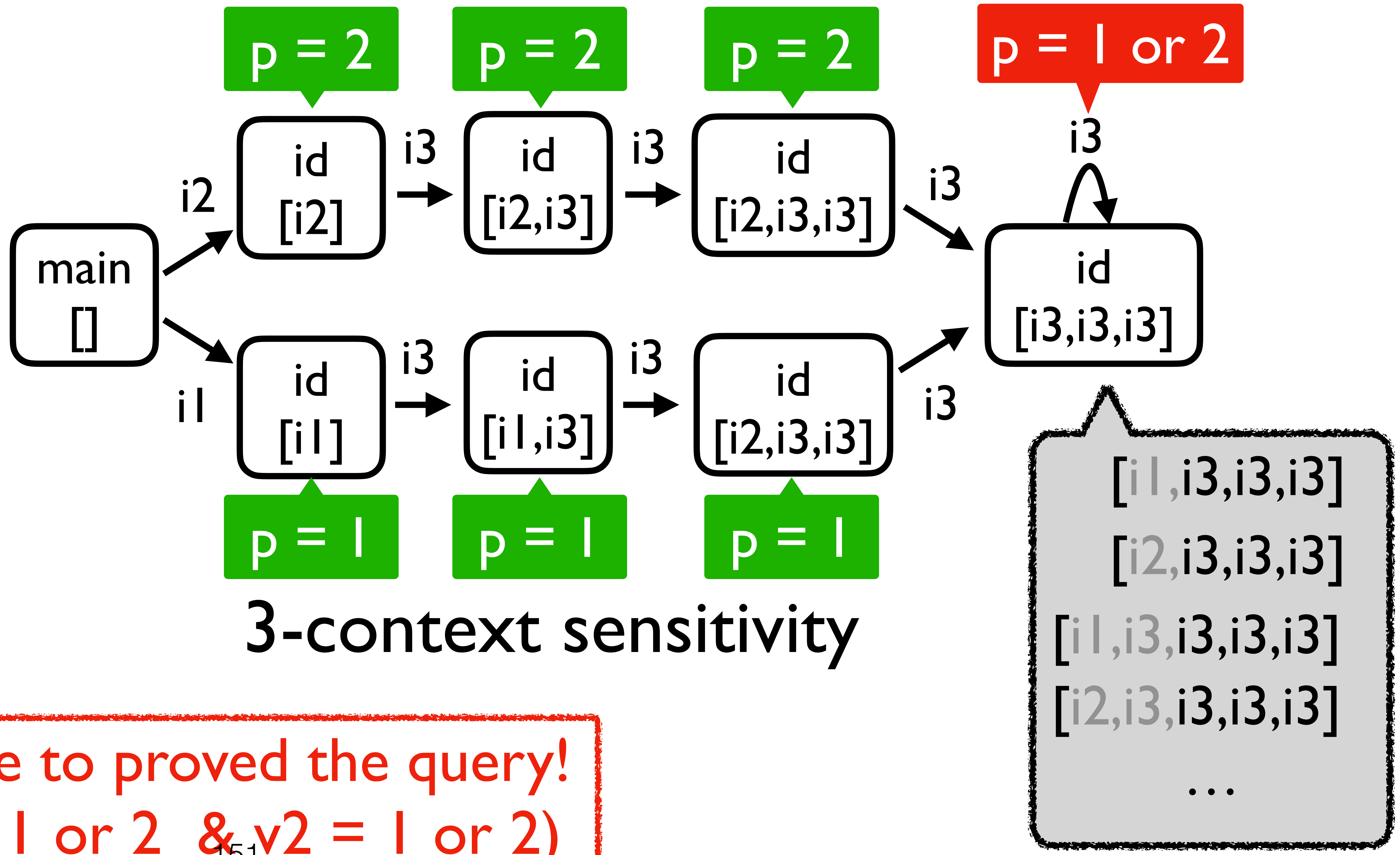
3-context sensitivity

Problem of Last-k

- An example showing a problem of last-k context abstraction

```
id(p, i){
  if i > 0:
    return id(p, i-1);//i3
  else:
    return p;}

main(){
  i = input();
  v2 = id(2,i);//i2
  v1 = id(1,i);//i1
  assert (v1 != v2);}//query
```

Unable to proved the query!
(v1 = 1 or 2 & v2 = 1 or 2)

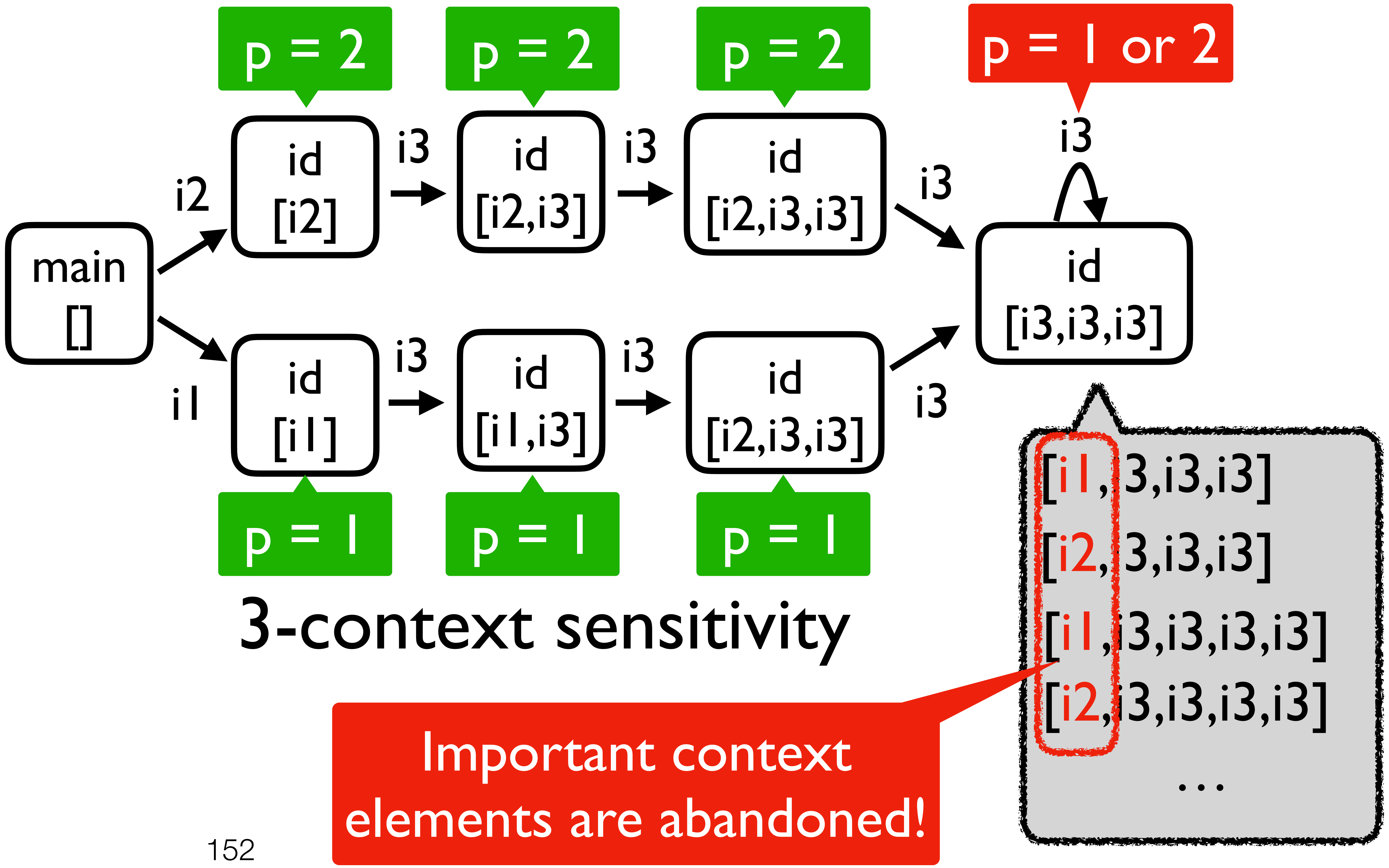


3-context sensitivity

Problem of Last-k

- An example showing a problem of last-k context abstraction

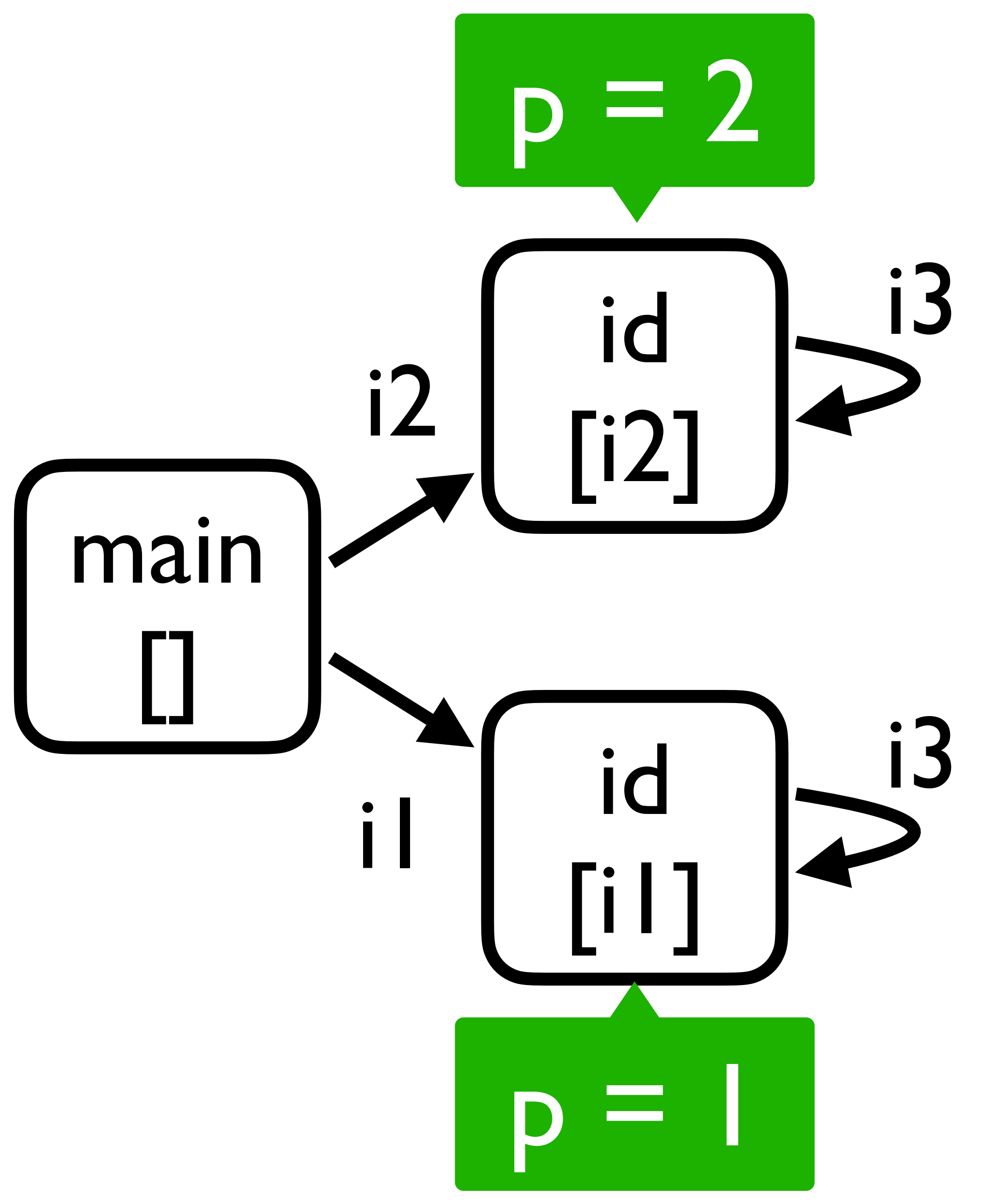
```
id(p, i){
  if i > 0:
    return id(p, i-1);//i3
  else:
    return p;}

main(){
  i = input();
  v2 = id(2,i);//i2
  v1 = id(1,i);//i1
  assert (v1 != v2);}//query
```



Our Solution: Keep Important K

- In important k, 1-ctx sensitivity proves the query

```
id(p, i){
  if i > 0:
    return id(p, i-1);//i3
  else:
    return p;}

main(){
  i = input();
  v2 = id(2,i);//i2
  v1 = id(1,i);//i1
  assert (v1 != v2);}//query
```



1-context sensitivity

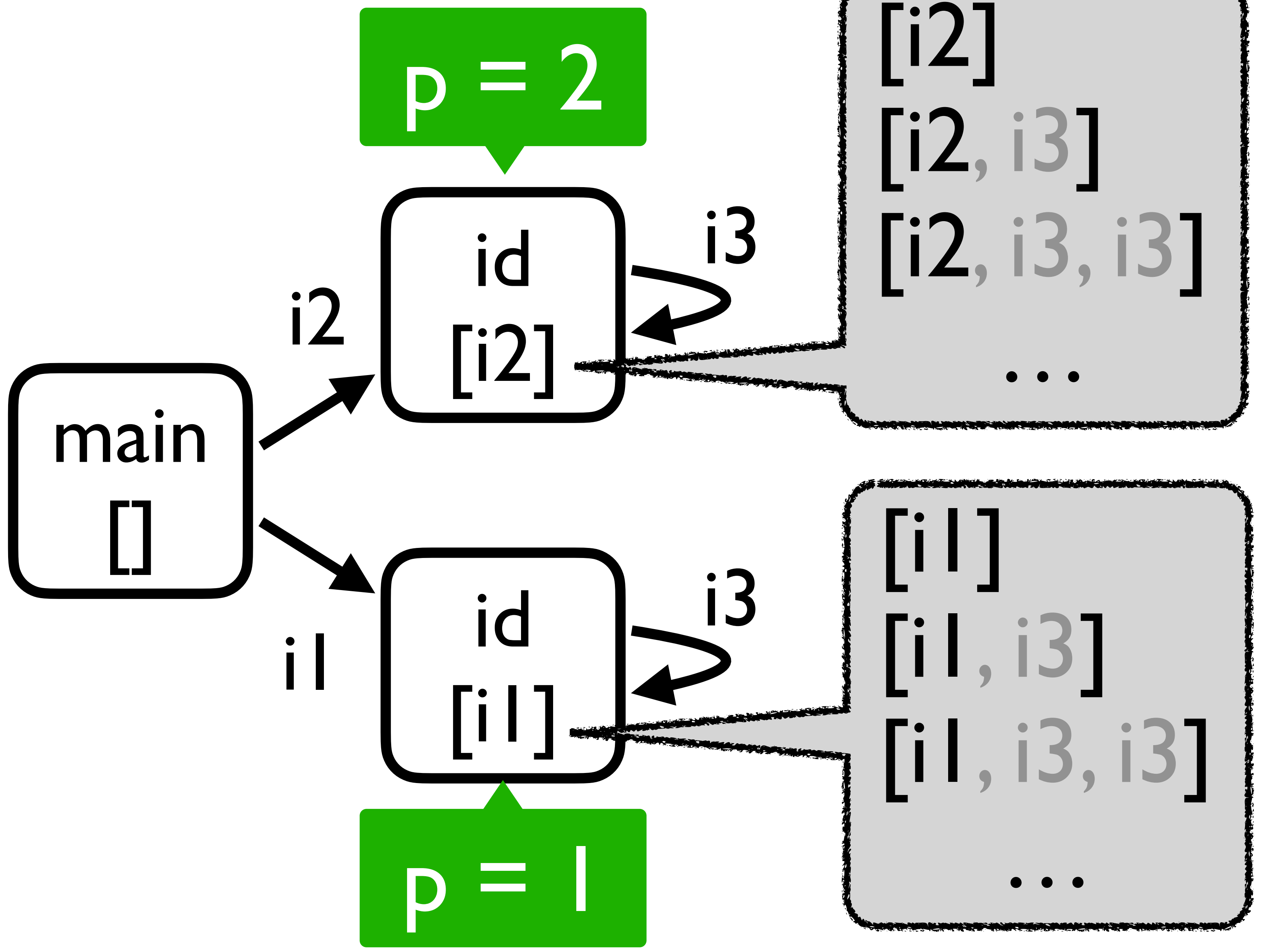
Important : {i1,i2}
Unimportant : {i3}

Our Solution: Keep Important K

- In important k, 1-ctx sensitivity proves the query

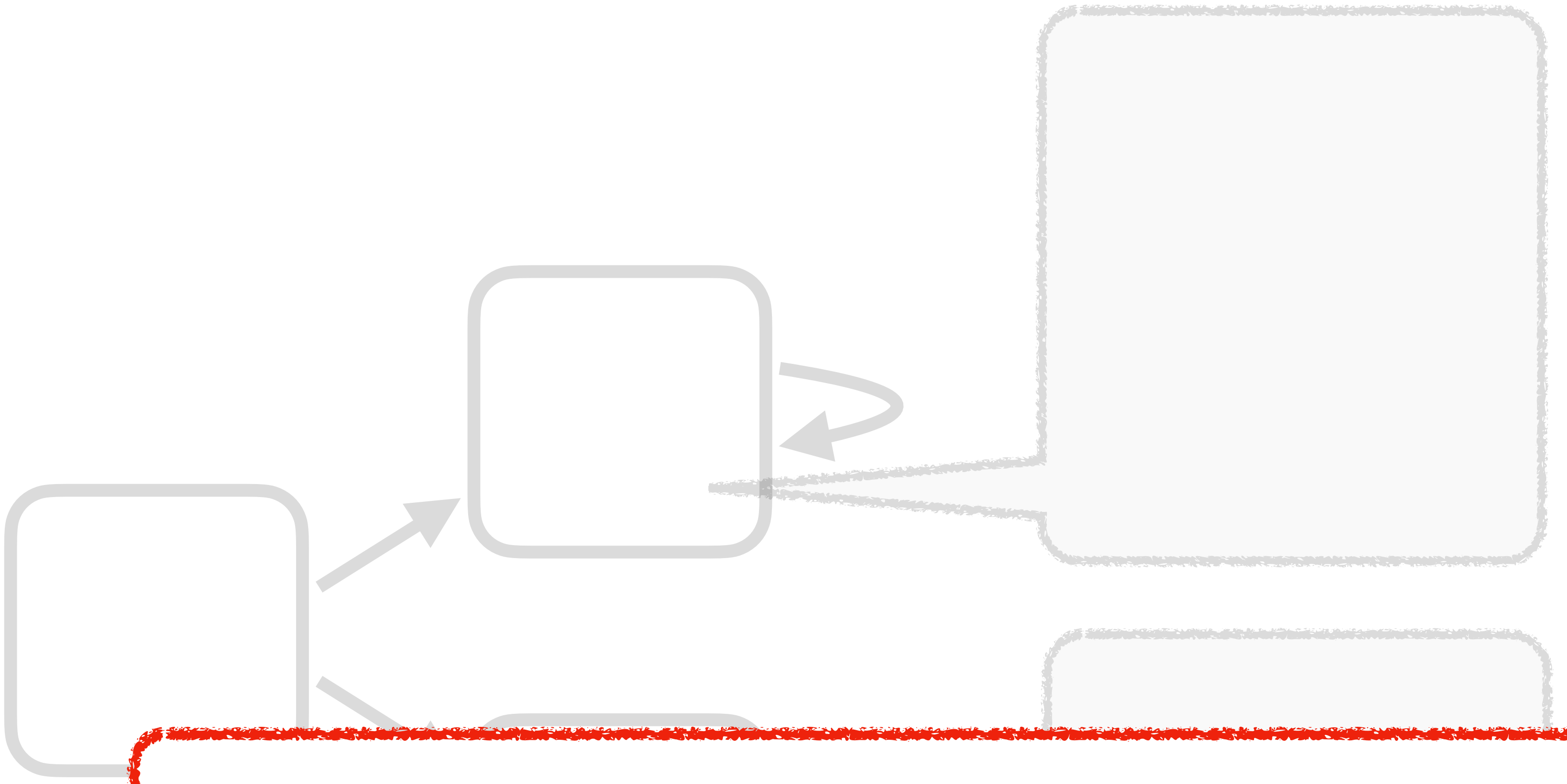
```
id(p, i){
  if i > 0:
    return id(p, i-1);//i3
  else:
    return p;}

main(){
  i = input();
  v2 = id(2,i);//i2
  v1 = id(1,i);//i1
  assert (v1 != v2);}//query
```

Proved the query!
(v1 = 2 & v2 = 1)



1-context sensitivity

Important : {i1,i2}
Unimportant : {i3}



Challenge:
How can we find a good classification?

Important : {i1,i2}
Unimportant : {i3}

Impact of Important k

- Applying important k improved the performance of a program repair tool for C programs

SAVER: Scalable, Precise, and Safe Memory-Error Repair

Seongjoon Hong, Korea University, Republic of Korea, seongjoon@korea.ac.kr
Junhee Lee*, Korea University, Republic of Korea, junhee_lee@korea.ac.kr
Jeongsoo Lee, Korea University, Republic of Korea, volcano0909@korea.ac.kr
Hakjoo Oh†, Korea University, Republic of Korea, hakjoo_oh@korea.ac.kr

ABSTRACT

We present SAVER, a new memory-error repair technique for C programs. Memory errors such as memory leak, double-free, and use-after-free are highly prevalent and fixing them requires significant effort. Automated program repair techniques hold the promise of reducing this burden but the state-of-the-art is still unsatisfactory. In particular, no existing techniques are able to fix those errors in a scalable, precise, and safe way, all of which are required for a truly practical tool. SAVER aims to address these shortcomings. To this end, we propose a method based on a novel representation of the program called object flow graph, which summarizes the program's heap-related behavior using static analysis. We show that fixing memory errors can be formulated as a graph labeling problem over object flow graph and present an efficient algorithm. We evaluated SAVER in combination with Infer, an industrial-strength static bug-finder, and show that 74% of the reported errors can be fixed automatically for a range of open-source C programs.

CCS CONCEPTS

• **Software and its engineering → Software verification and validation; Software testing and debugging.**

KEYWORDS

Program Repair, Program Analysis, Memory Errors, Debugging

ACM Reference Format:
Seongjoon Hong, Junhee Lee, Jeongsoo Lee, and Hakjoo Oh. 2020. SAVER: Scalable, Precise, and Safe Memory-Error Repair. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3377811.3380323

1 INTRODUCTION

Recent years have seen significant progress in automated tools for static error detection and their deployment in production code [7, 15, 50]. Yet, fixing those errors in practice remains mostly a manual and unscalable process. The longterm goal of our research is to bridge this gap by automating the whole process of finding and fixing common safety errors at compile-time.

In this paper, we focus on fixing memory errors in C programs such as memory leak, use-after-free, and double-free. We target these errors because they are highly prevalent yet difficult to fix. For example, more than 4,800 memory leaks have been reported and fixed in Linux kernel [1], far more frequently than other errors such as buffer overflow. Fixing memory errors manually, however, is time-consuming and error-prone; even a single error may require developers to spend several days or months until the error gets fixed correctly (e.g. [2]). We aim to reduce this burden by achieving a practical technique for automatically fixing memory errors.

Existing Techniques. Recently, several techniques with similar goals have been proposed in the program repair community [18, 34, 60] but they suffer from significant drawbacks. On the one hand, there are techniques that scale to large programs but may produce unsafe patches. For example, FootPatch [60] is a state-of-the-art tool that can fix memory leaks in large code bases but may introduce new errors such as double-free as a side-effect (e.g. see Section 2.1). On the other hand are tools that guarantee to generate safe patches but do so at the expense of scalability or repairability. For example, MemFix [34] can safely fix memory leaks, use-after-frees, and double-frees but is applicable only to small programs (<5KLoC). LeakFix [18], another safe fixing tool, is scalable but limited to fixing simple memory-leaks, having a relatively low fix rate (e.g. <15% for GNU utilities [34]). All of these shortcomings of existing techniques make them inappropriate for practical use.

Our Approach. In this paper, we present SAVER, a scalable, precise, and safe technique for automatically fixing memory errors. To guarantee safety, SAVER uses a sound program verification technique and produces patches only when it is confident that the error gets fixed without violating other safety conditions. To achieve high repairability, SAVER supports various fixing strategies, including conditional deallocation and relocation of pointer dereferences. To this end, we propose a new representation of the program called *object flow graph*, which is a labeled graph summarizing the program's heap-related behavior using a static heap analysis. The key technical novelty is to formulate the problem of fixing memory errors as a labeling problem over object flow graph and present an efficient algorithm for finding correct labels. For scalability, SAVER applies the aforementioned analysis and verification techniques selectively and locally. The analysis is selective as it uses expensive abstractions (e.g. path-sensitivity) only when doing so benefits. Also, only a fraction of the input program is analyzed by effectively slicing out the program with respect to the target error.

The experimental results show that SAVER is a practical tool, significantly outperforming existing techniques. We implemented

*The first and second authors contributed equally to this work.
†Corresponding author

ACM ISBN 978-1-4503-7121-6/20/05...$15.00
https://doi.org/10.1145/3377811.3380323

Static analysis based memory-error repair technique for C programs published in ICSE 2020

Successfulness heavily depends on the performance of underlying static analysis

Context tunneling significantly improved the underlying static analysis

Impact of Important k

- Applying important k improved the performance of a program repair tool for Ocaml programs

Context-Aware and Data-Driven Feedback Generation for Programming Assignments

Dowon Song
Korea University
Republic of Korea
dowon_song@korea.ac.kr

Woosuk Lee
Hanyang University
Republic of Korea
woosuk@hanyang.ac.kr

Hakjoo Oh*
Korea University
Republic of Korea
hakjoo_oh@korea.ac.kr

ABSTRACT

Recently, various techniques have been proposed to automatically provide personalized feedback on programming exercises. The cutting edge of which is the data-driven approaches that leverage a corpus of existing correct programs and repair incorrect submissions by using similar reference programs in the corpus. However, current data-driven techniques work under the strong assumption that the corpus contains a solution program that is close enough to the incorrect submission. In this paper, we present CAFE, a new data-driven approach for feedback generation that overcomes this limitation. Unlike existing approaches, CAFE uses a novel context-aware repair algorithm that can generate feedback even if the incorrect program differs significantly from the reference solutions. We implemented CAFE for OCaml and evaluated it with 4,211 real student programs. The results show that CAFE is able to repair 83% of incorrect submissions, far outperforming existing approaches.

CCS CONCEPTS

• **Software and its engineering → Automatic programming.**

KEYWORDS

Program Repair, Program Synthesis

ACM Reference Format:
Dowon Song, Woosuk Lee, and Hakjoo Oh. 2021. Context-Aware and Data-Driven Feedback Generation for Programming Assigaments. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3468264.3468598

1 INTRODUCTION

In recent years, there has been a surge of interest in automatic feedback generation for programming assignments [1, 5, 12, 19, 20, 22, 33, 35, 36, 39, 40, 43]. As the demand for programming education grows, it is becoming increasingly difficult for an instructor to provide personalized feedback to a large number of students. Simply providing an instructor's solution as feedback is unsatisfactory, as students's attempts typically diverge from the reference solution. The goal of automatic feedback generation technology is to help students to understand what they did wrong and how to fix it without manual effort of instructors.

Data-Driven Feedback Generation. Among prior techniques, data-driven approaches [12, 17, 34, 42, 43] are arguably the current state-of-the-art. The main idea of these techniques is to leverage a corpus of existing correct programs, and repair an incorrect program by using similar reference solutions in the corpus. In contrast to approaches that require intervention of instructors [5, 19, 39], data-driven techniques are fully automatic and yet show impressive performance in repairing introductory programming exercises.

However, existing data-driven techniques have a significant shortcoming. That is, they rely on a strong assumption that the corpus contains a solution program that is *close* to the incorrect program. For example, two notable techniques, CLARA [12] and SARFGEN [43], assume a solution exists that is equivalent to the incorrect program modulo control flows. This assumption, however, does not hold always [22], especially when providing feedback beyond introductory-level exercises. In this case, constructing a corpus with the dose-program assumption becomes a challenge.

Our Approach. In this paper, we present CAFE, a new data-driven feedback generation technique that overcomes the above limitation. Unlike existing approaches, CAFE can generate feedback even when the incorrect submission is substantially different from reference solutions.

The keystone of CAFE is its context-aware, function-level repair algorithm. CAFE primarily targets sizable programming exercises, where students are freely allowed to define and use their own helper functions. To repair such a program, CAFE does not seek to find a solution program that matches the submission in its entirety; instead, it leverages multiple, partially-matching references. More specifically, CAFE works at the function level, aiming to separately repair each function in the incorrect program by (1) finding a matching function from the corpus, (2) computing their difference, and (3) extracting a patch from the difference. A main challenge with this approach is how to find the matching function that is useful for repair. Our key idea to solve this problem is to infer and compare the *original intent* of the functions by analyzing their calling contexts in the respective programs, which robustly identifies useful references even when functions have different syntax and semantics.

We evaluated CAFE in a real classroom setting. The original motivation of this work was to develop a feedback generation system for our own programming course, where we use OCaml and newcomers to functional programming often have a hard time. Thus, we

*Corresponding author

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08...$15.00
https://doi.org/10.1145/3468264.3468598

Program repair technique for ocaml programs published in FSE 2021

Method language

are equivalent as empty contexts. We mitigated this shortcoming by applying the idea of context tunneling [18] and updating contexts at call-sites only when they are non-empty. For example, suppose

Context tunneling played an important role

Find a heuristic (classifier) $\mathscr{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

- minimizes analysis cost while is precise enough

- User-provided precision constraint
- E.g., maintain 90% precision of 2-ctx sensitivity for the training set

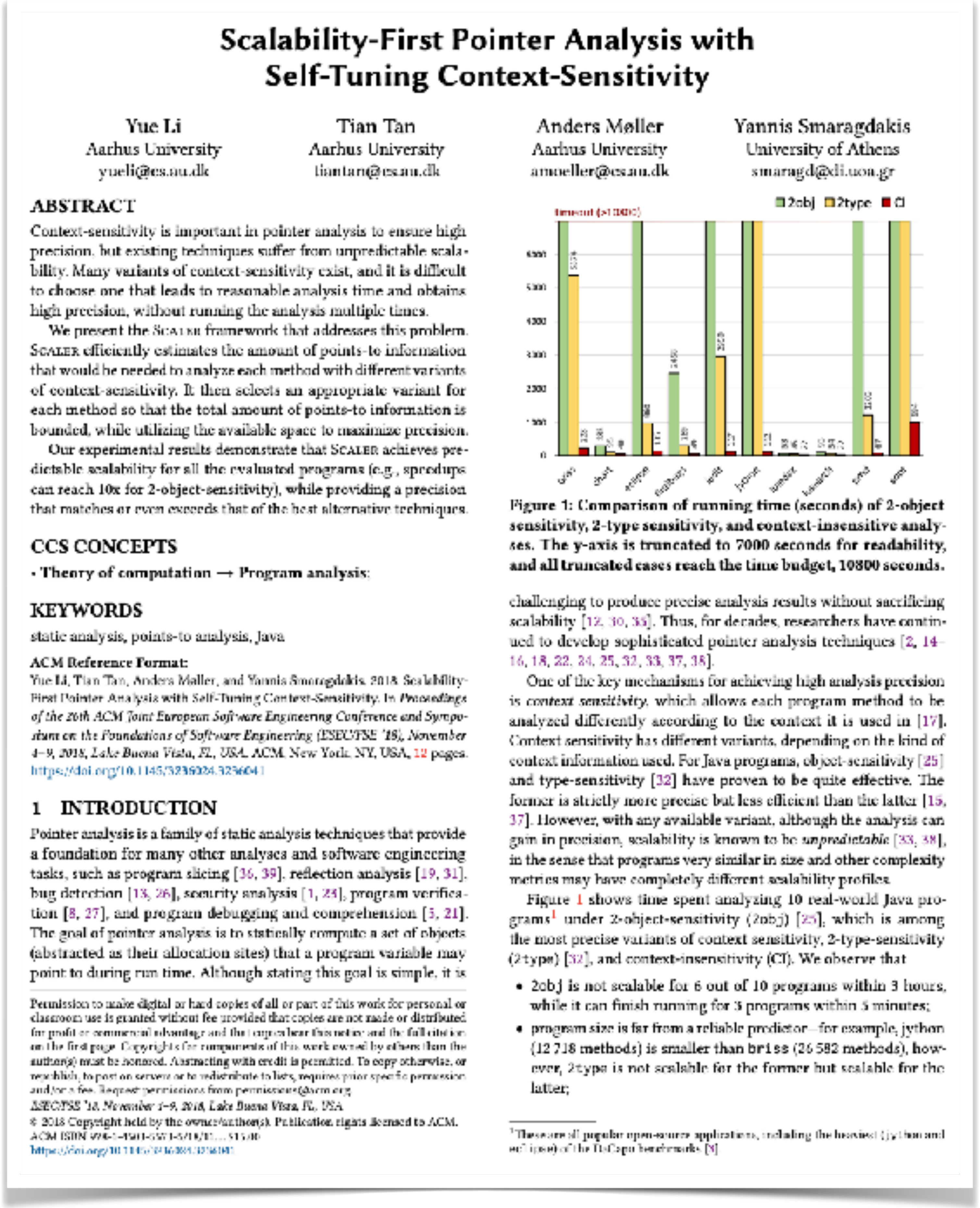
$$\frac{\text{\# queries proved by the current heuristic } \mathscr{H}}{\text{\# queries proved by the fully 2-ctx sensitivity}} > 0.9$$

Classifies all the methods into 2-ctx

$f_{2ctx} = (a_1 \wedge \ldots$

$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \ldots)$

Learned Pattern Used In Manually Crafted Heuristics


Scalability-First Pointer Analysis with Self-Tuning Context-Sensitivity

Yue Li, Aarhus University, yueli@cs.au.dk
Tian Tan, Aarhus University, tiantan@cs.au.dk
Anders Møller, Aarhus University, amoeller@cs.au.dk
Yannis Smaragdakis, University of Athens, smaragd@di.uoa.gr

ABSTRACT

CCS CONCEPTS

KEYWORDS

1 INTRODUCTION

Figure 1: Comparison of running time (seconds) of 2-object sensitivity, 2-type sensitivity, and context-insensitive analyses. The y-axis is truncated to 7000 seconds for readability, and all truncated cases reach the time budget, 10800 seconds.

FSE 2018

Learned pattern for 2-obj in our OOPSLA 17 paper

Manually crafted heuristic

Scaler treats methods under package java.util.* specially, explicitly assigning them to be analyzed by the most precise context sensitivity (i.e., 2obj in our settings)