

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

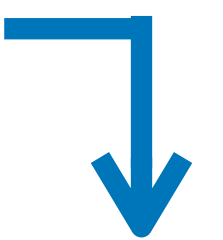
Korea University

Nov, 15, 2023 @ POSTECH



Generalization

Data-Driven Static Analysis



& PL-based Explainable Graph Machine Learning



Part 1

Part 2

Nov, 15, 2023 @ POSTECH

Minseok Jeon

Korea University



Part I: 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

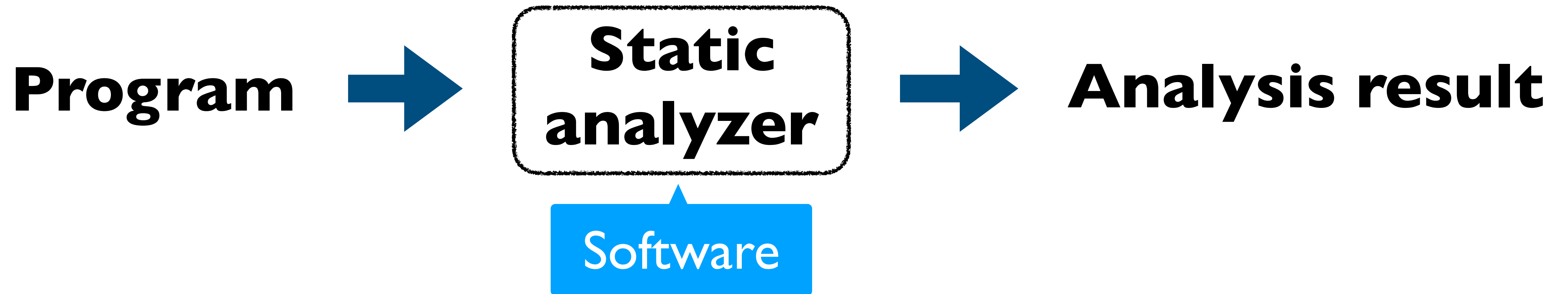


...

Static Program Analysis

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

- **Static analyzer** is a **software** that analyzes software



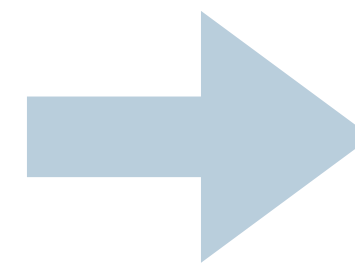
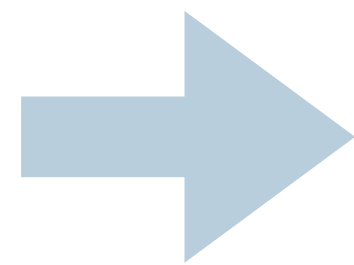
Static Program Analysis

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

- **Static analyzer** analyzes program **source code** without execution



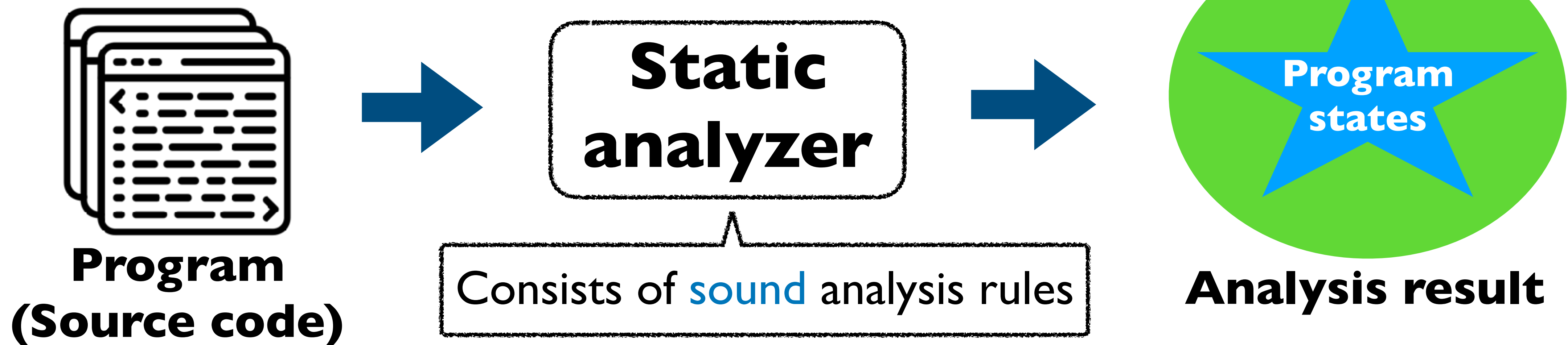
Program
(Source code)



Static Program Analysis

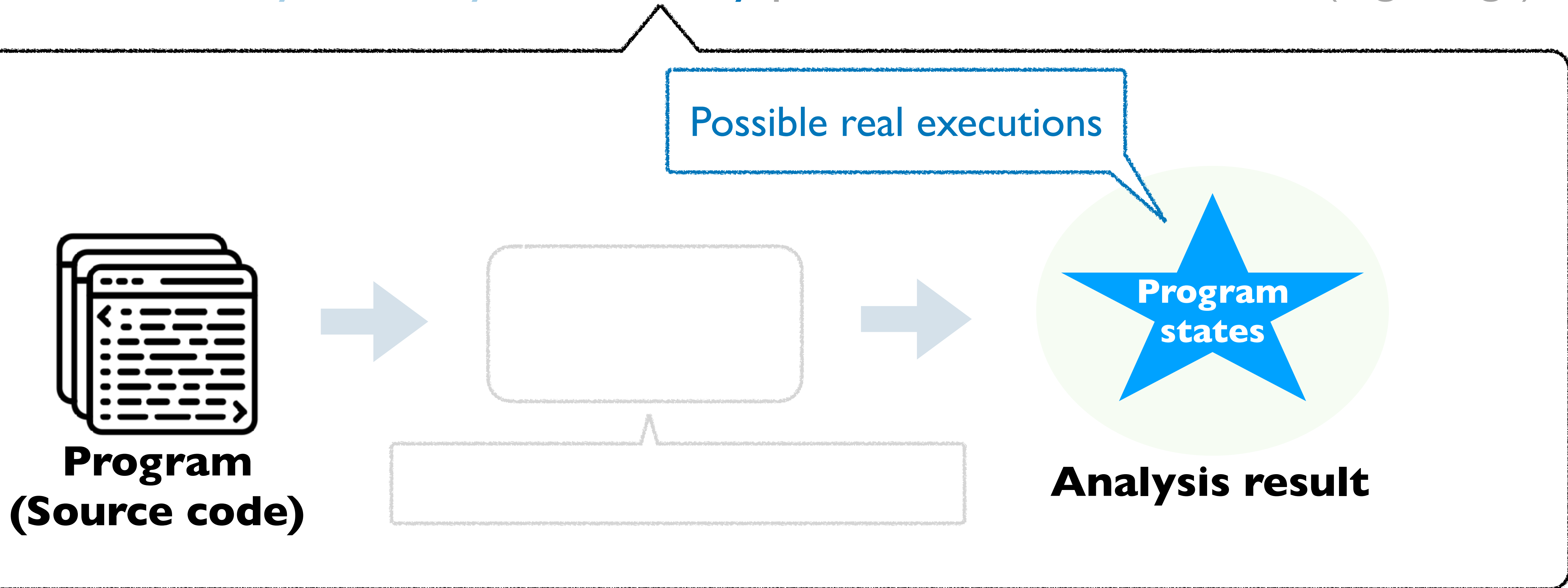
- Automatically, statically, and **soundly** predict software behavior (e.g., bugs)

- **Static analyzer** computes an over-approximation of program behavior



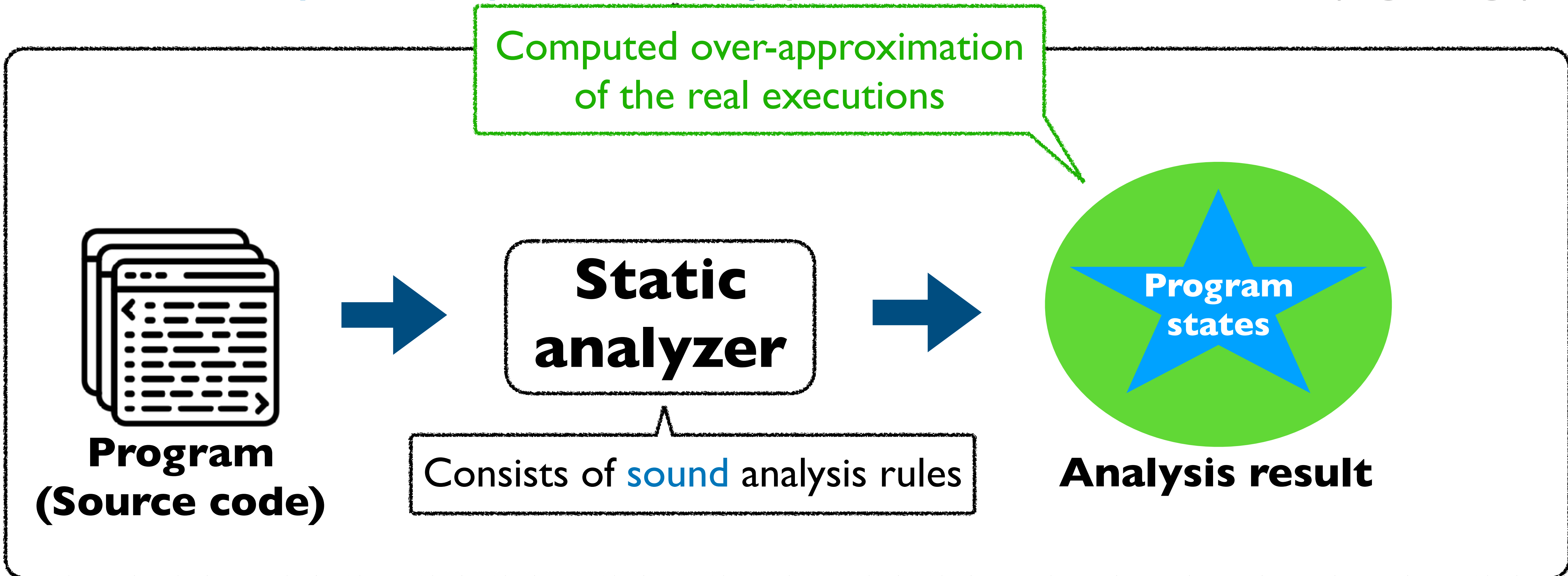
Static Program Analysis

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



Static Program Analysis

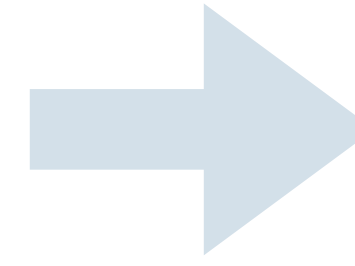
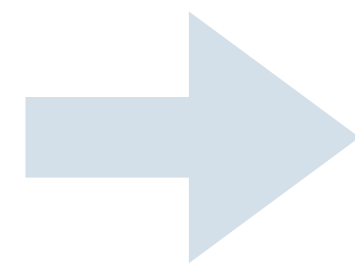
- Automatically, statically, and soundly predict software behavior (e.g., bugs)



Static Program Analysis

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

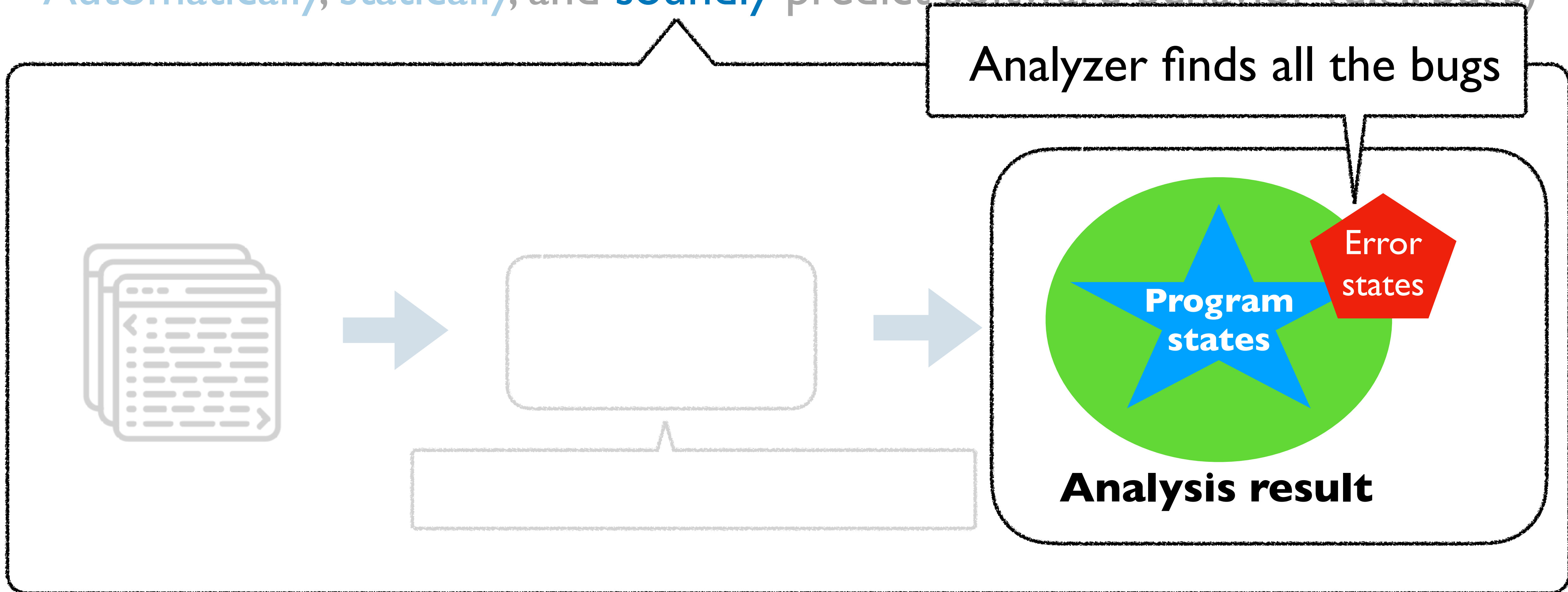
Analyzer proved the program is **bug-free**



Analysis result

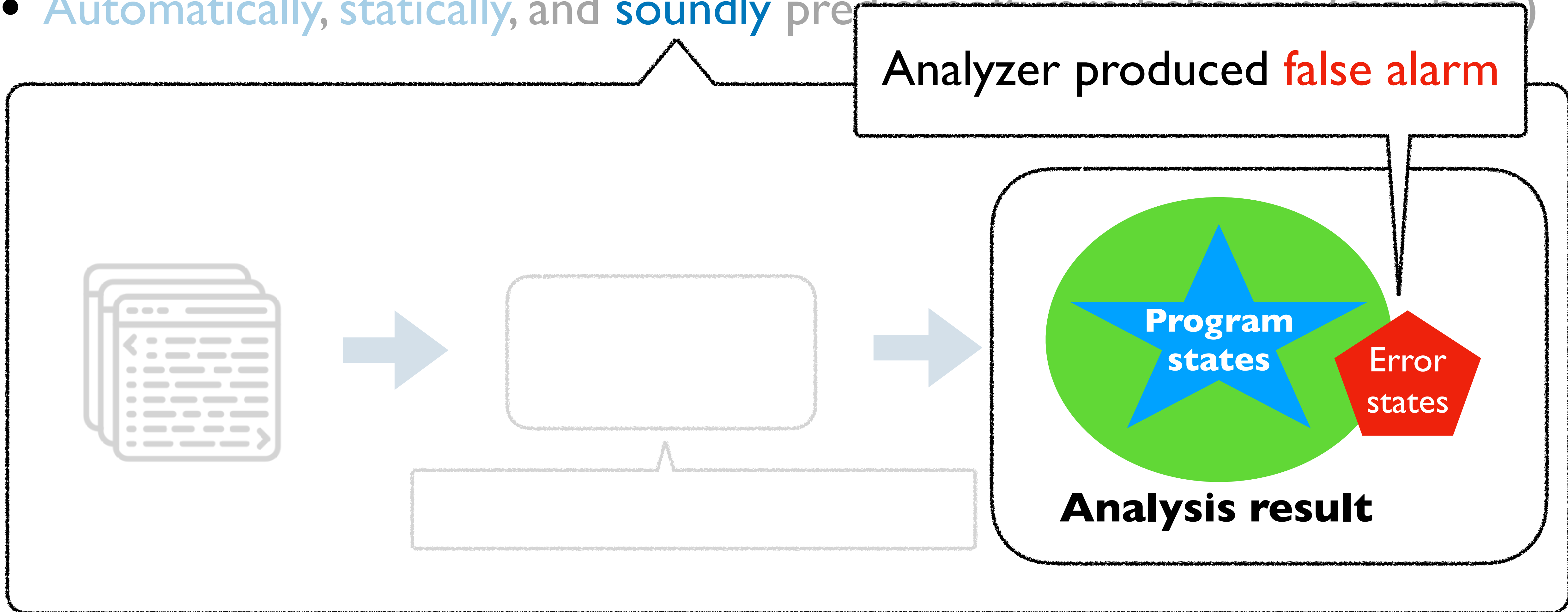
Static Program Analysis

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



Static Program Analysis

- Automatically, statically, and soundly predict program behavior (analysis)



Static Program Analysis

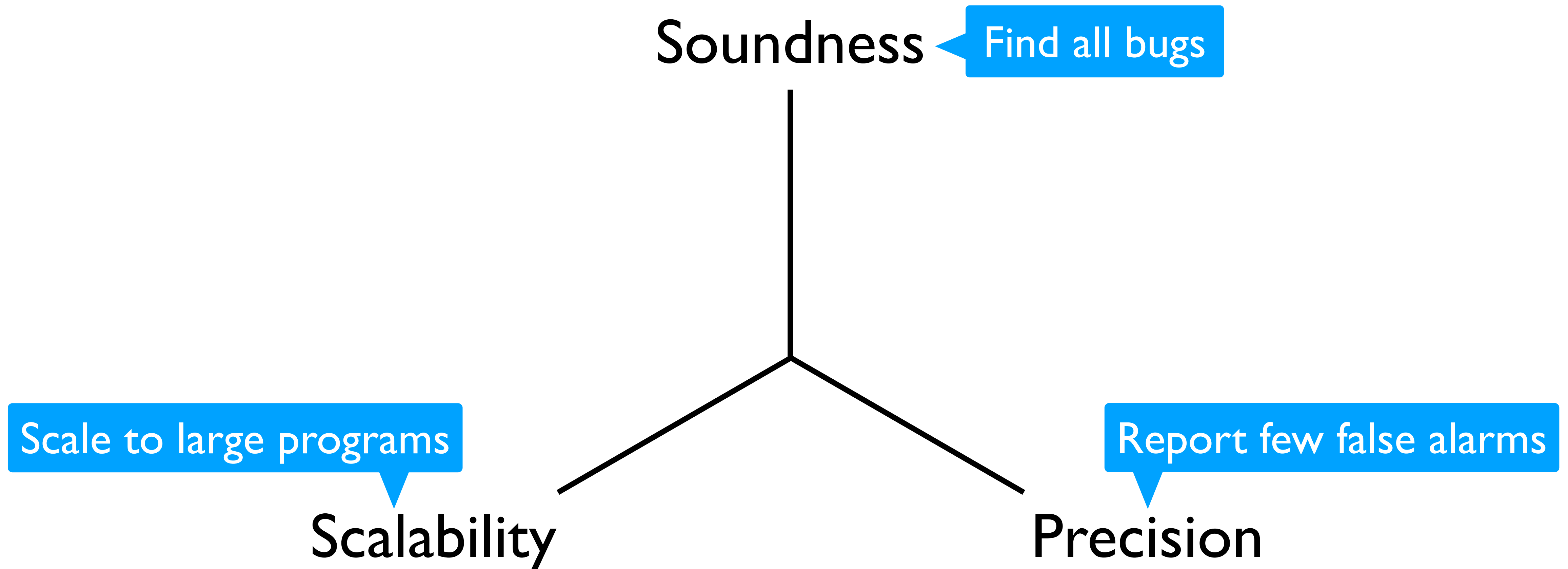
- Widely used in software industry



...

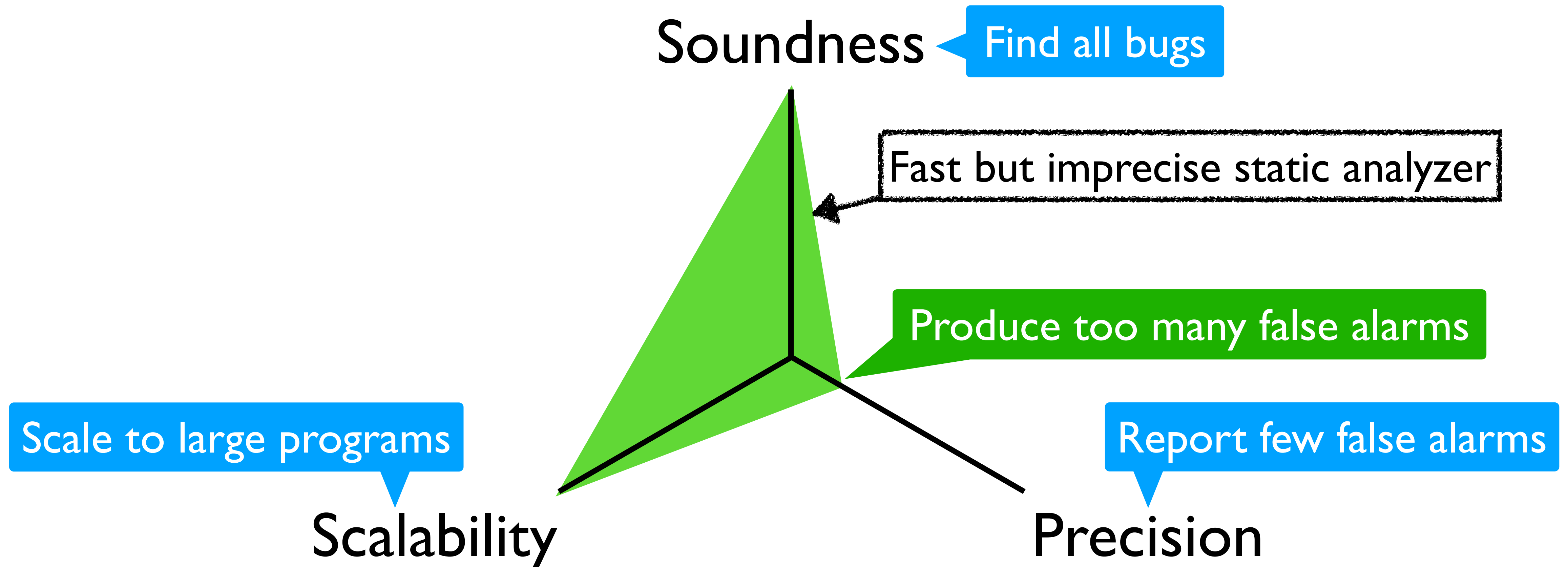
Long Standing Open Problem in Static Analysis

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



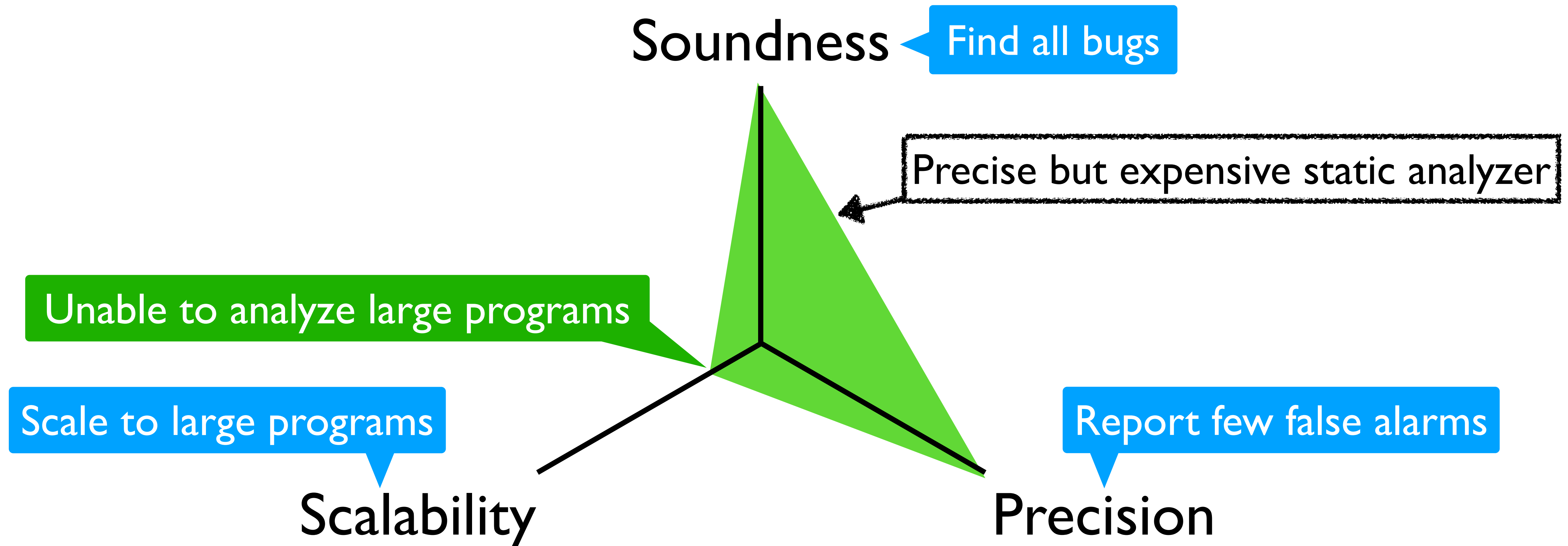
Long Standing Open Problem in Static Analysis

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



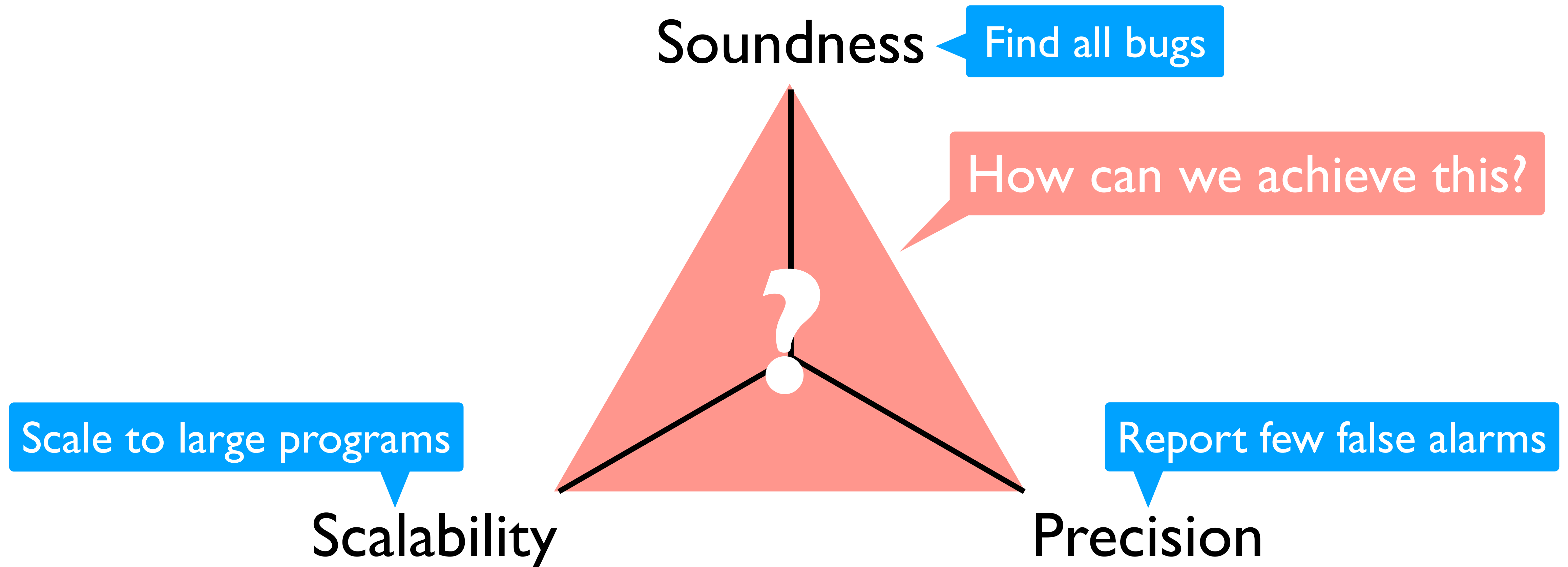
Long Standing Open Problem in Static Analysis

- 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

x = g(10);//i3

assert (x > 0);//query

g(v){ret h(v);}i5

h(v){ret v;}

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)

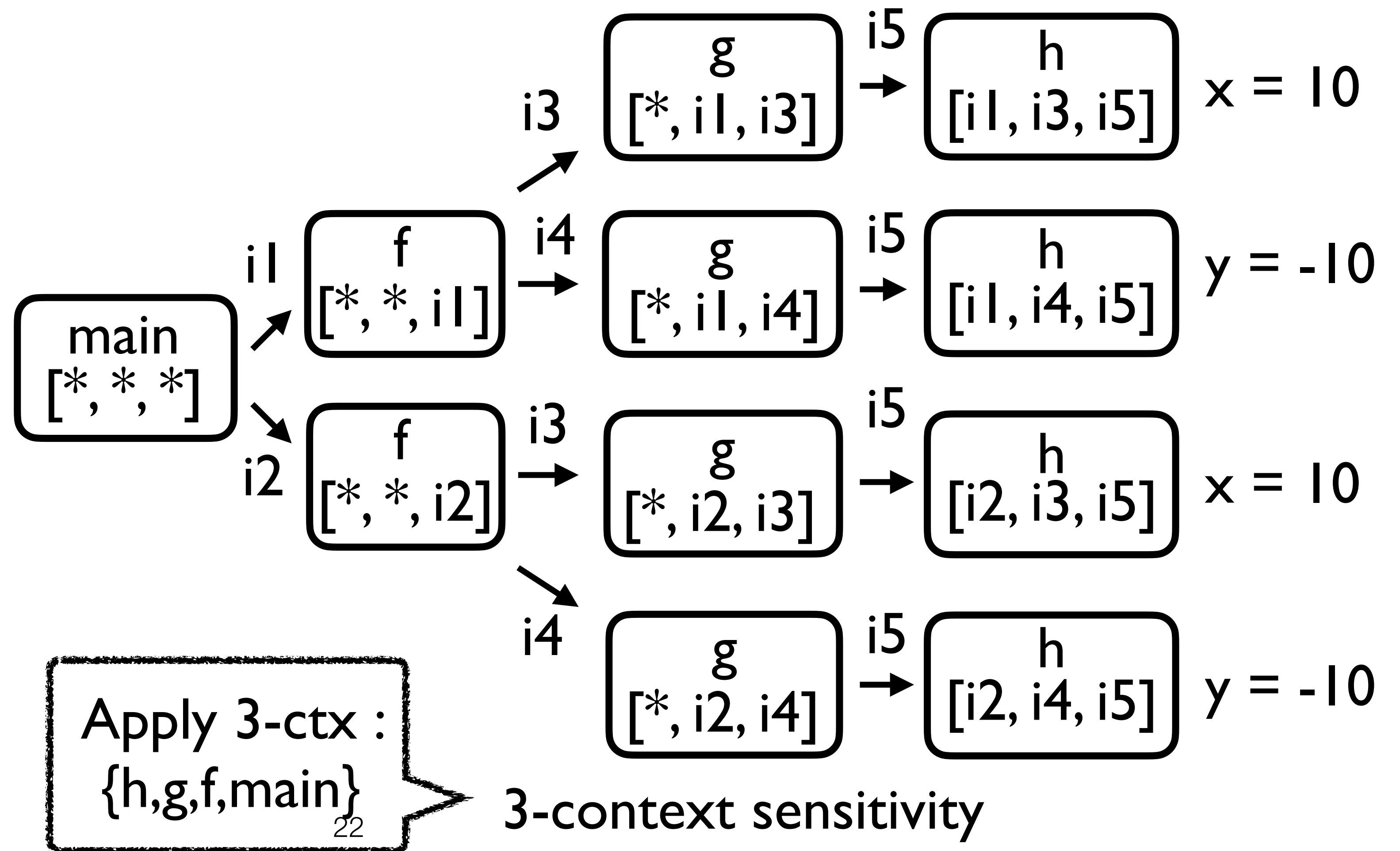
- Which methods need to be analyzed **precisely**?
- Which methods need to be analyzed **coarsely**?

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

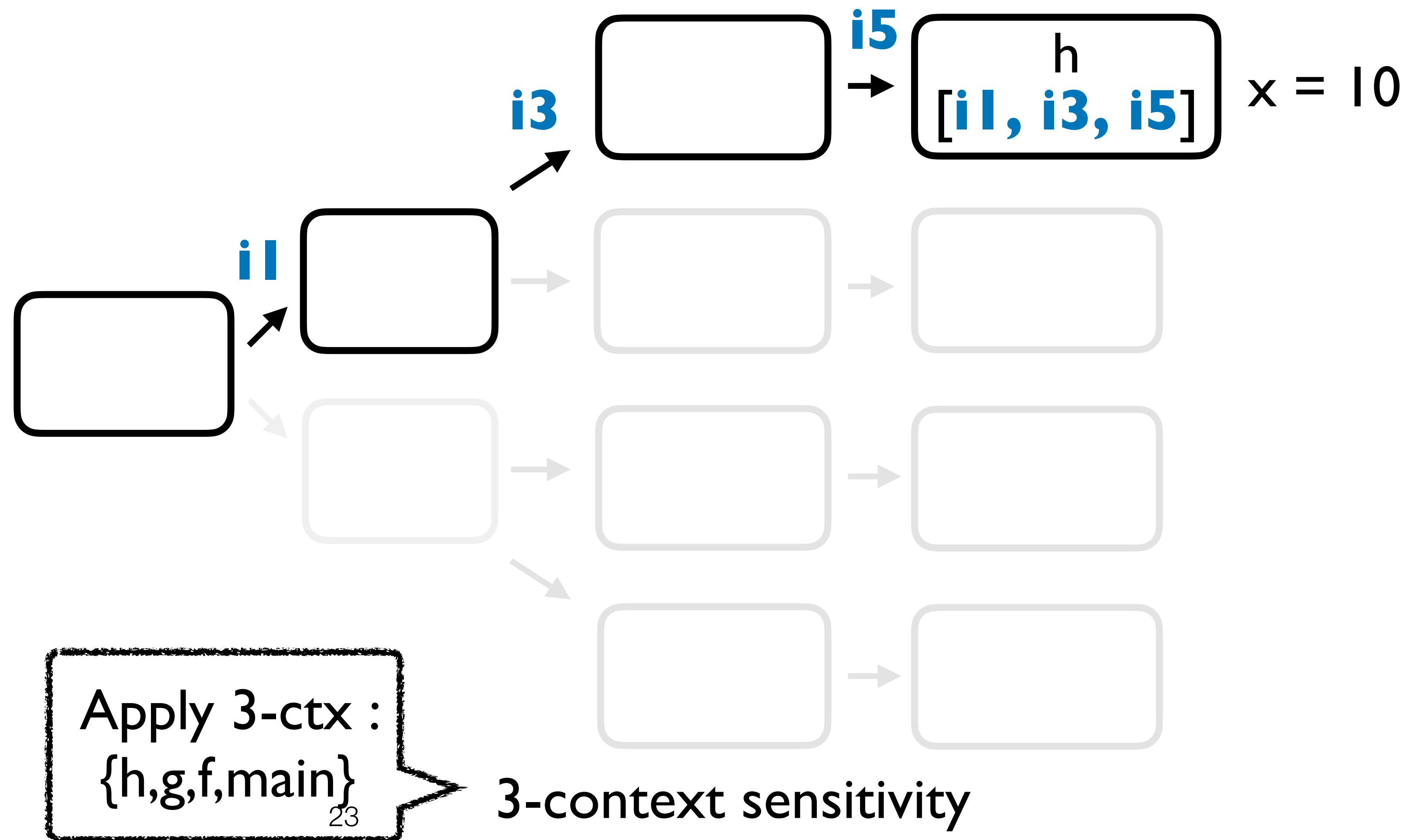


Example: **Selective** Context Sensitivity

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

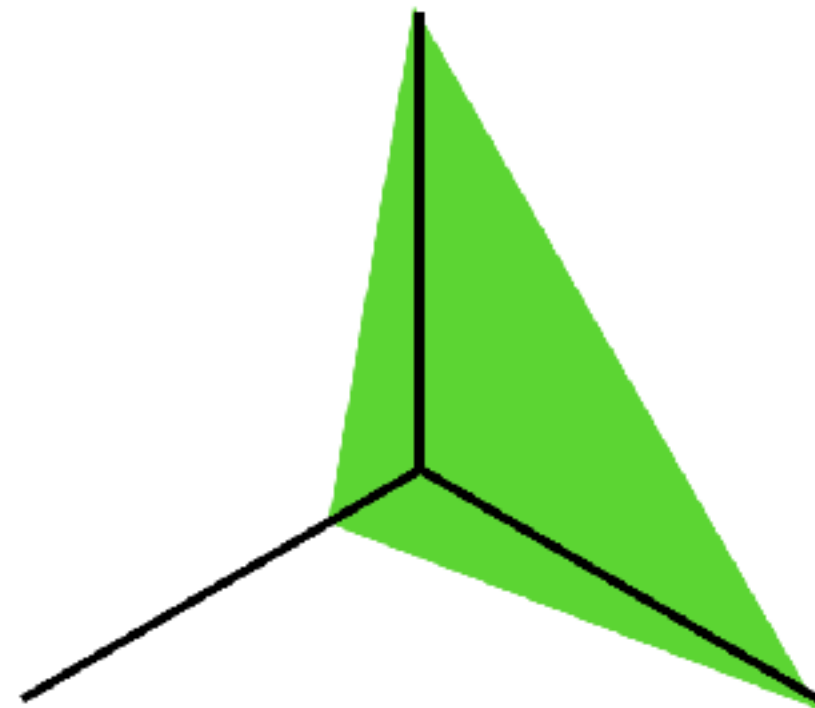
```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}i5
h(v){ret v;}
```

Example program



Context Sensitivity

Soundness



• **Pre**

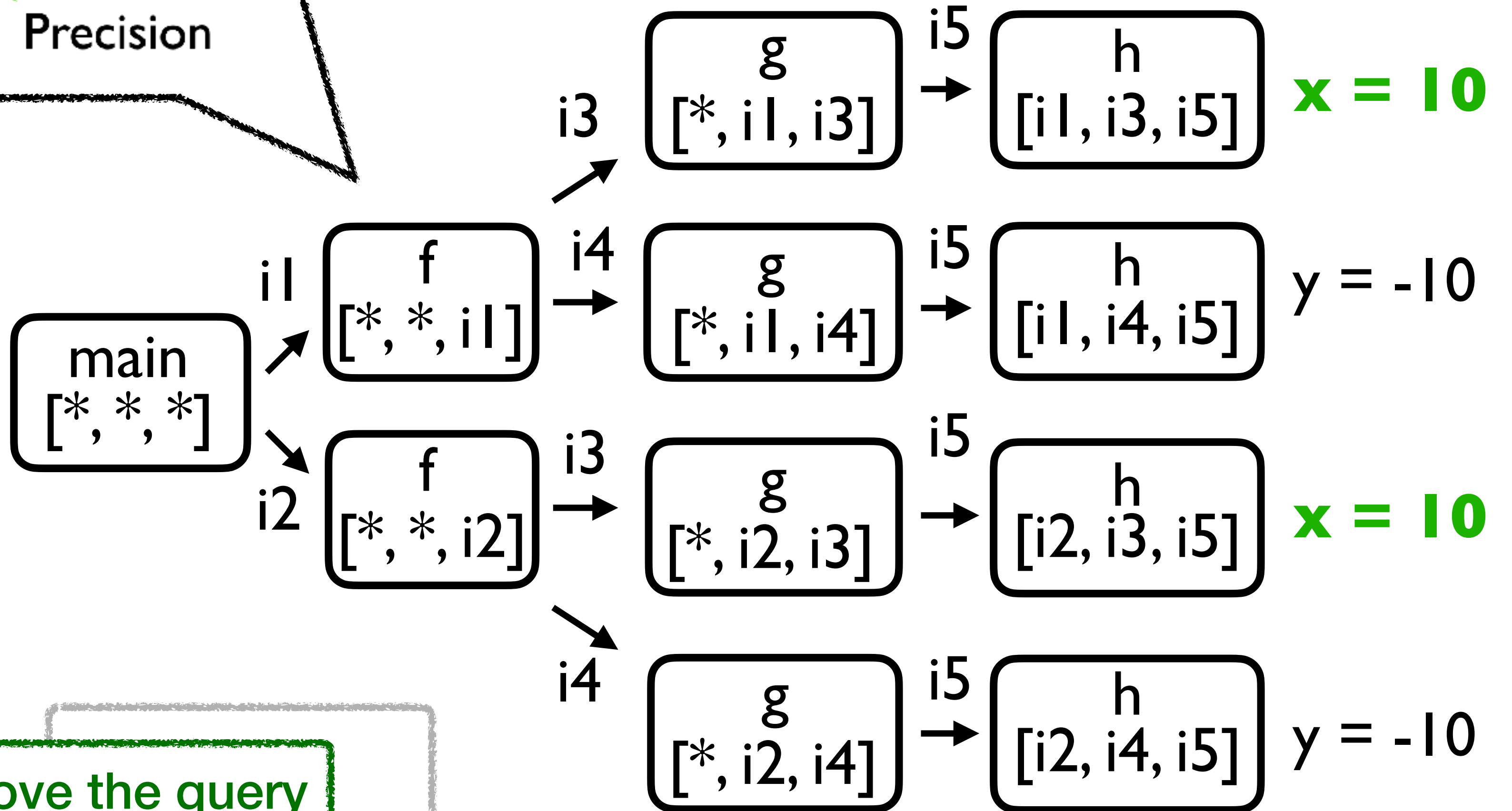
... makes the analysis precise but **expensive**

```
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

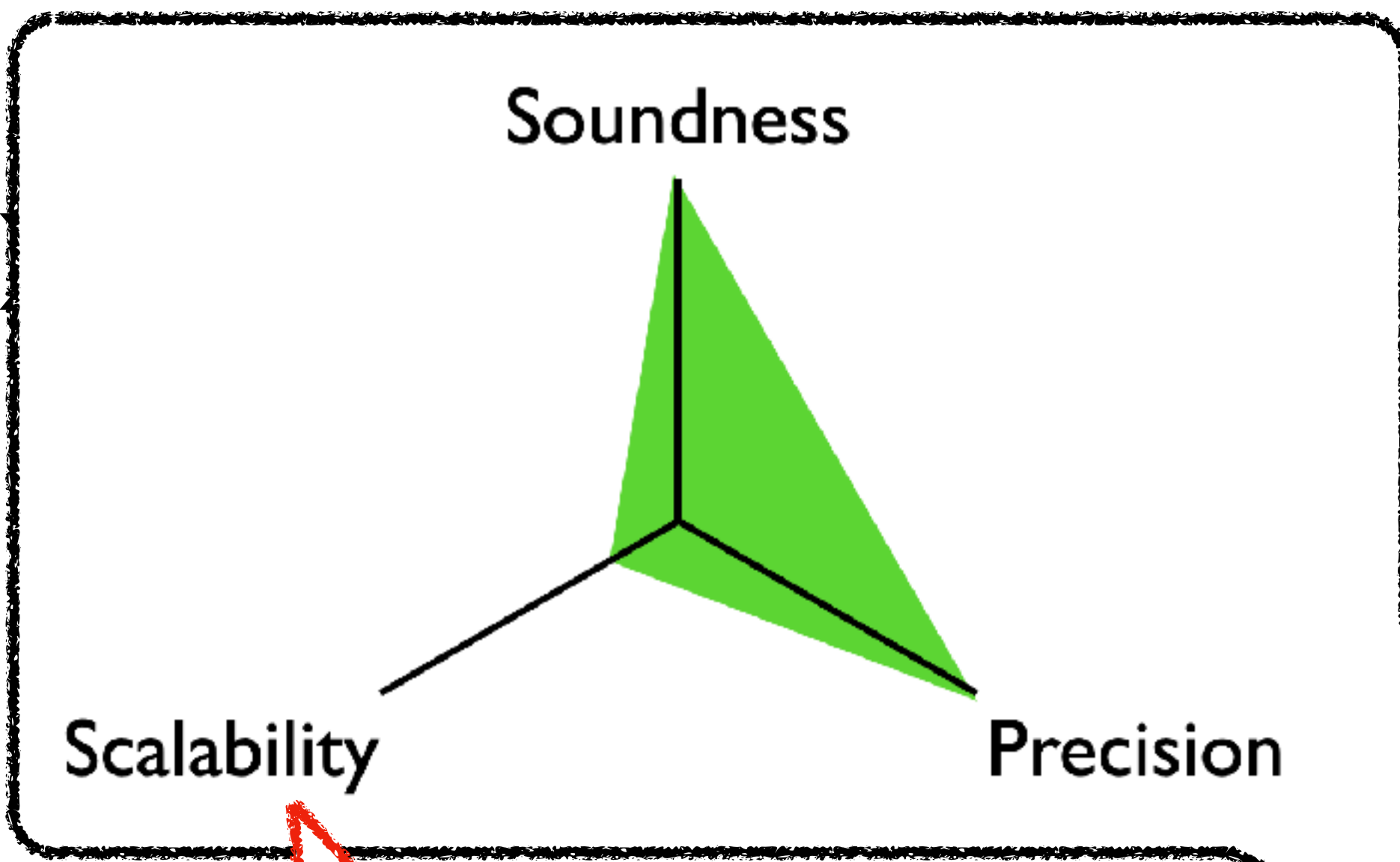
Can prove the query

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.**”

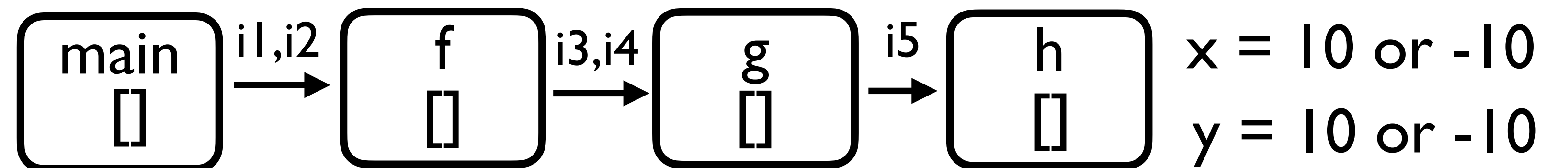
- Smaragdakis et al. [2014]

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

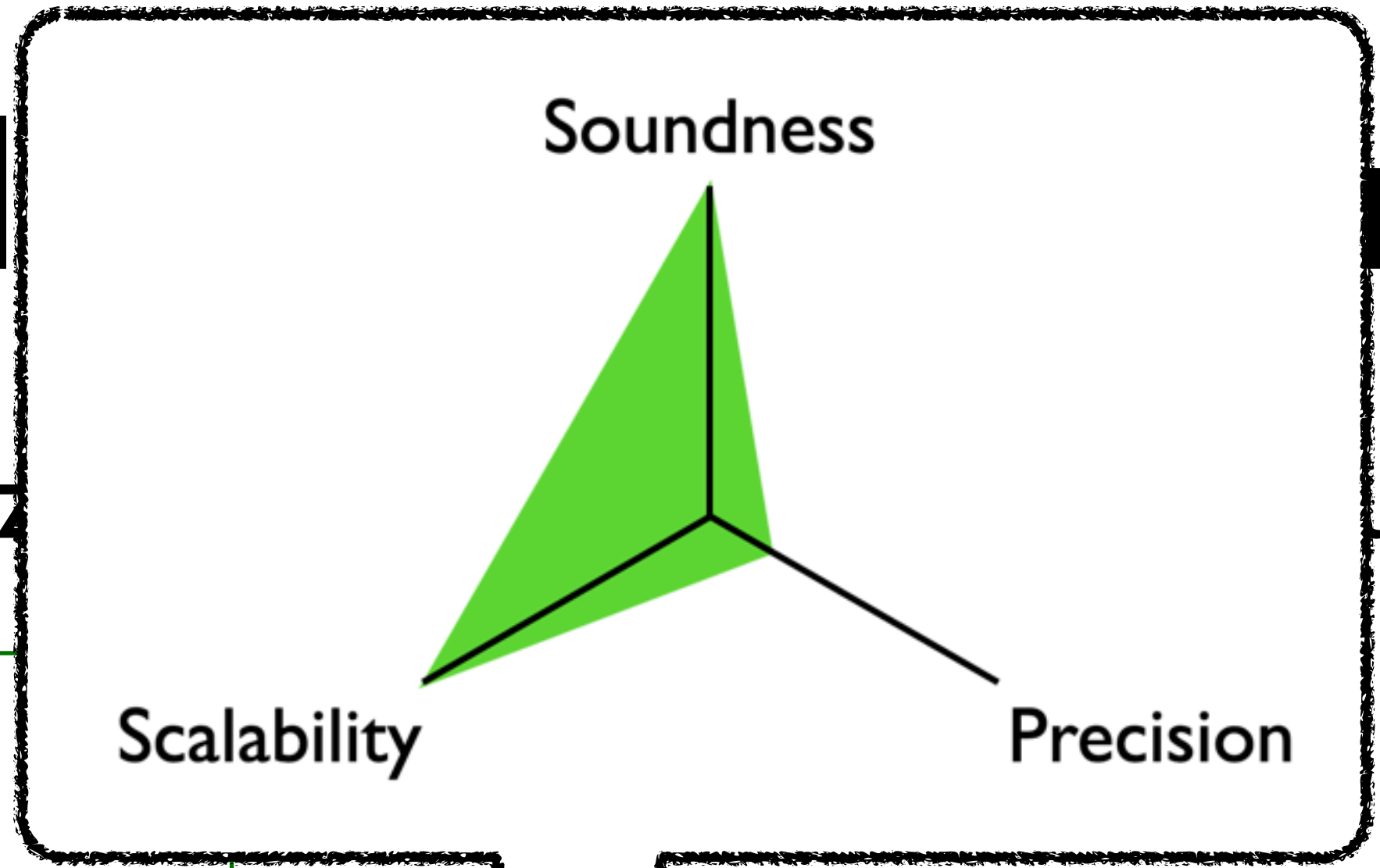


context-insensitive
(0-ctx sensitivity)

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

Example Context Sensitivity

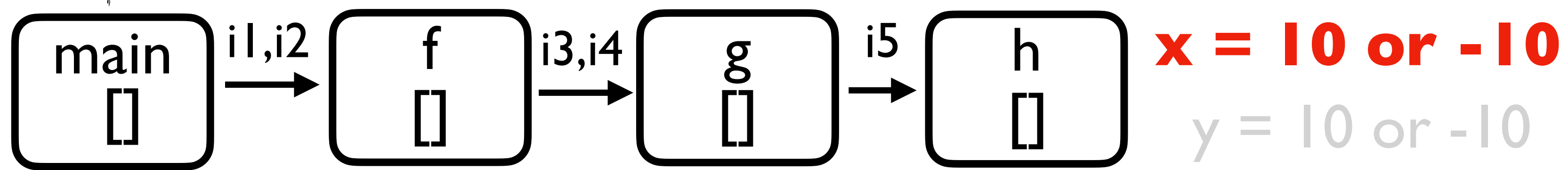
- Coarsely analyzing context makes the analysis **imprecise**



```

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



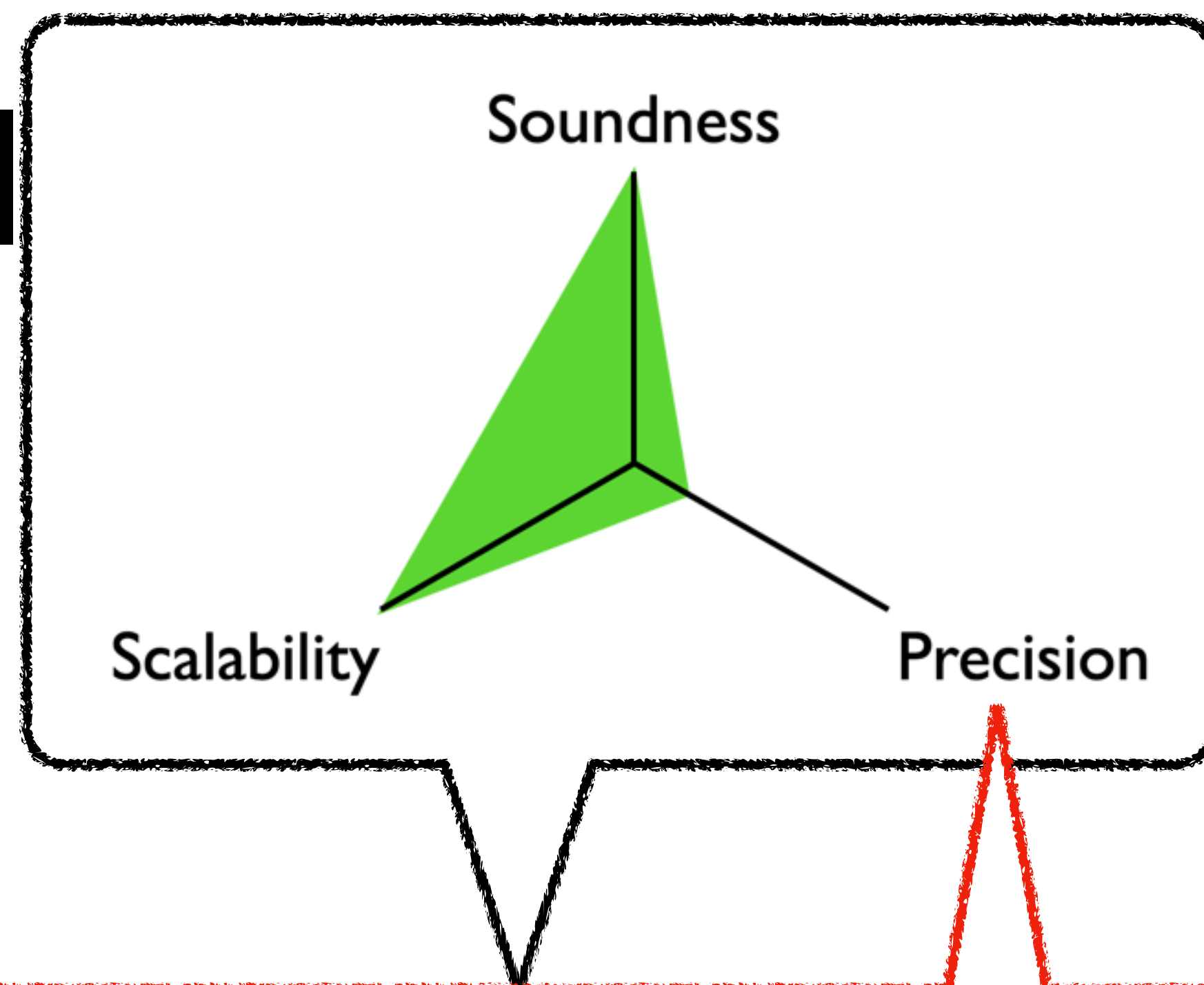
context-insensitive
(0-ctx sensitivity)

unable to prove the query

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

Example program

Example



Context Sensitivity

Coarsely analyzing all the methods is also impractical in practice

“for some applications,....**precise** context sensitivity is **essential**”

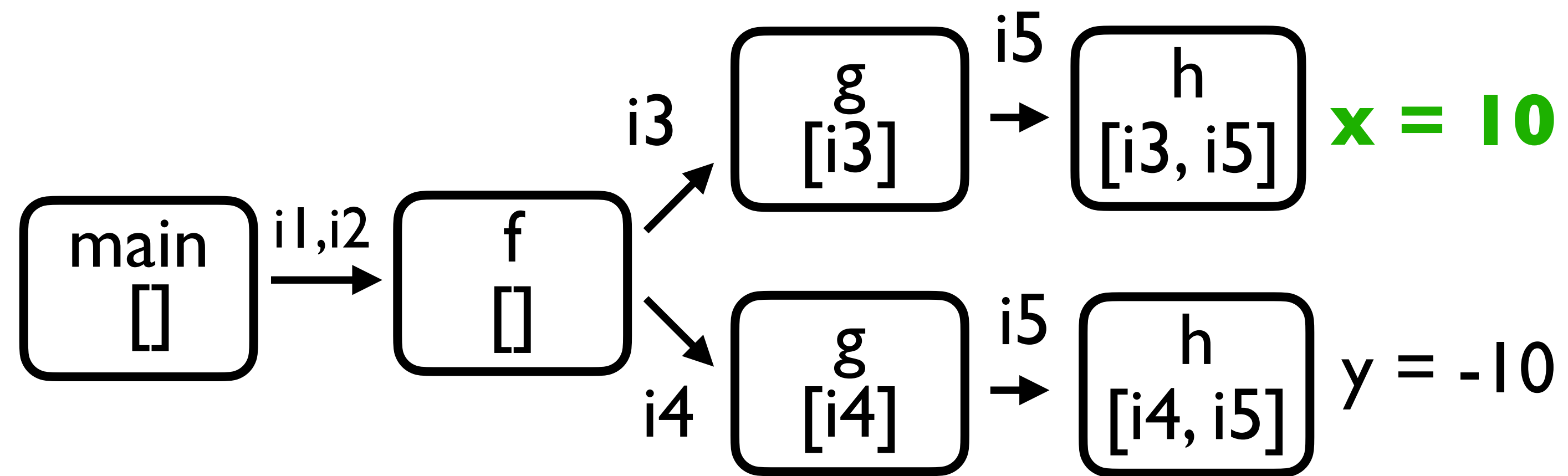
- Smaragdakis et al. [2014]

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

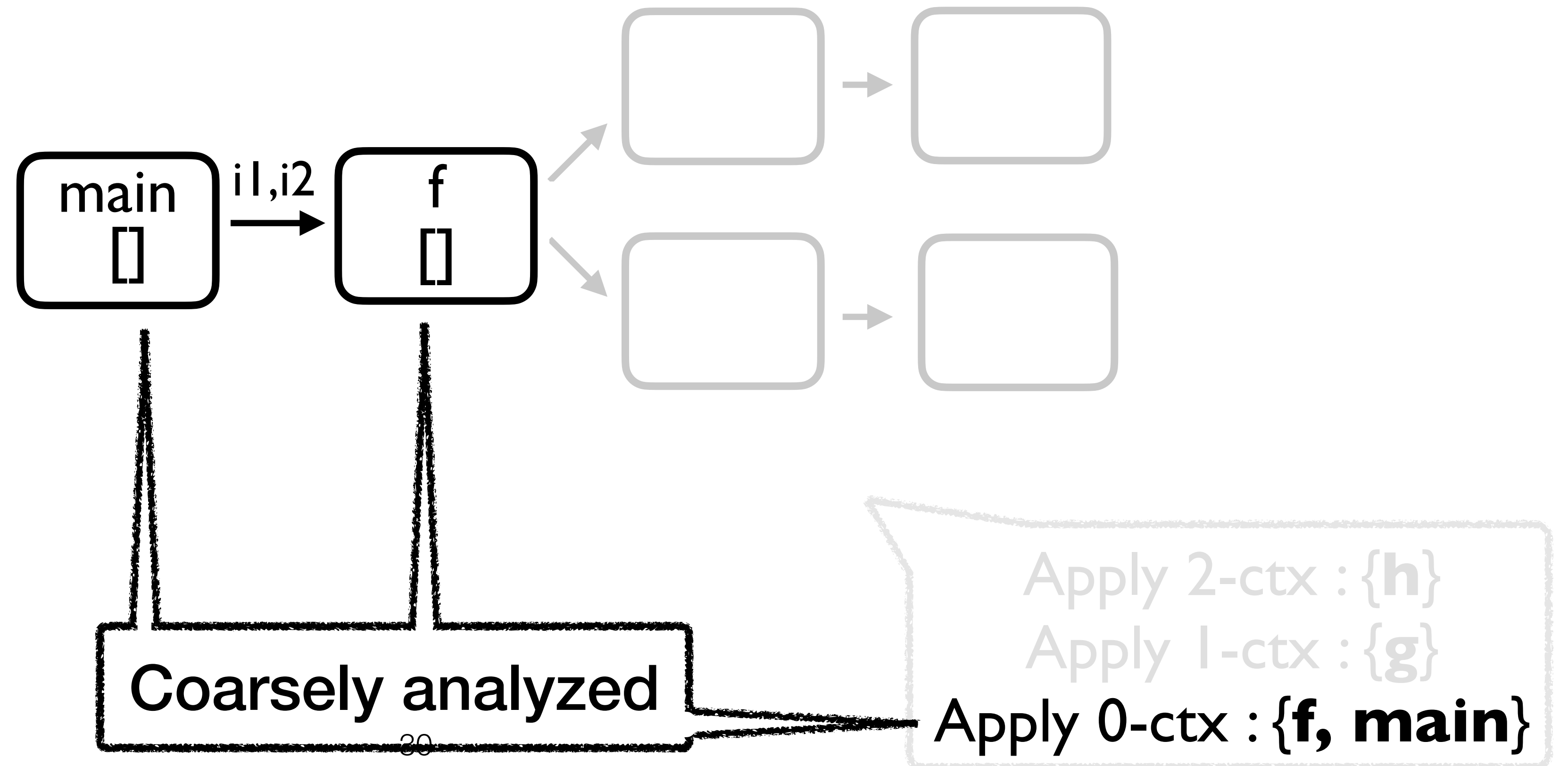


Selective context sensitivity

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

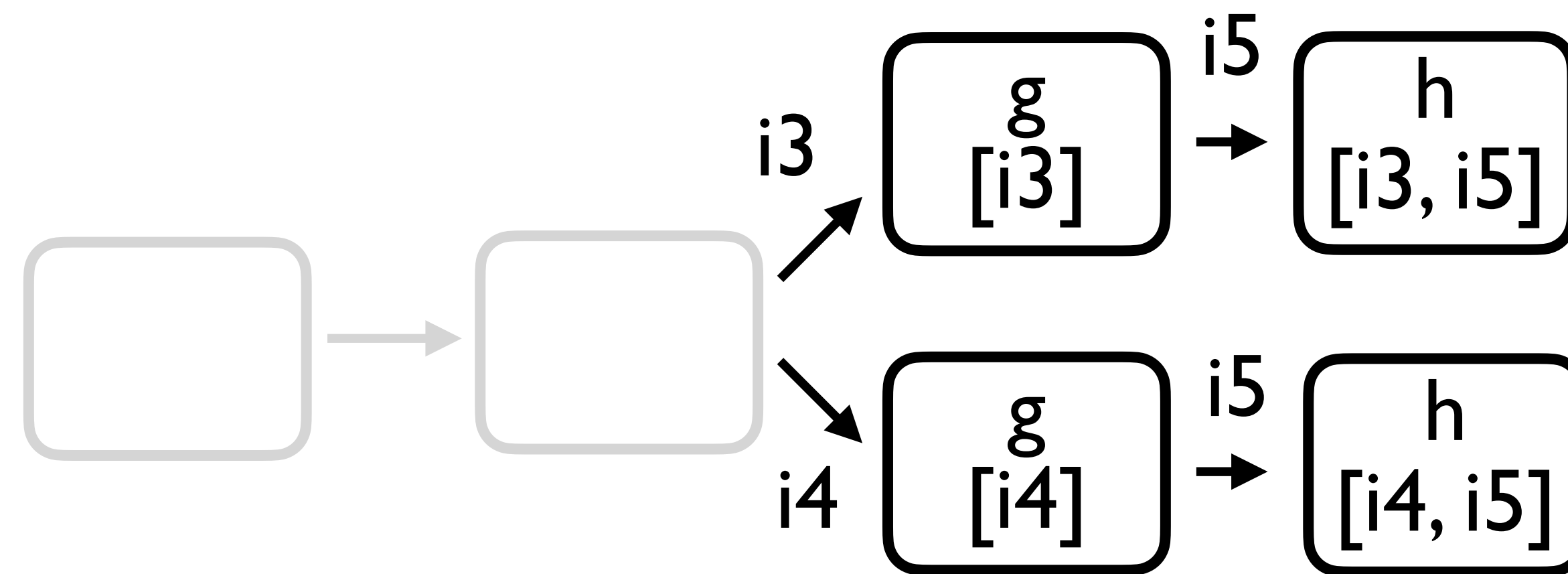
Example: **Selective** Context Sensitivity

- Selective context sensitivity can make the analysis fast and precise



Example: **Selective** Context Sensitivity

- Selective context sensitivity can make the analysis fast and precise



Precisely analyzed

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

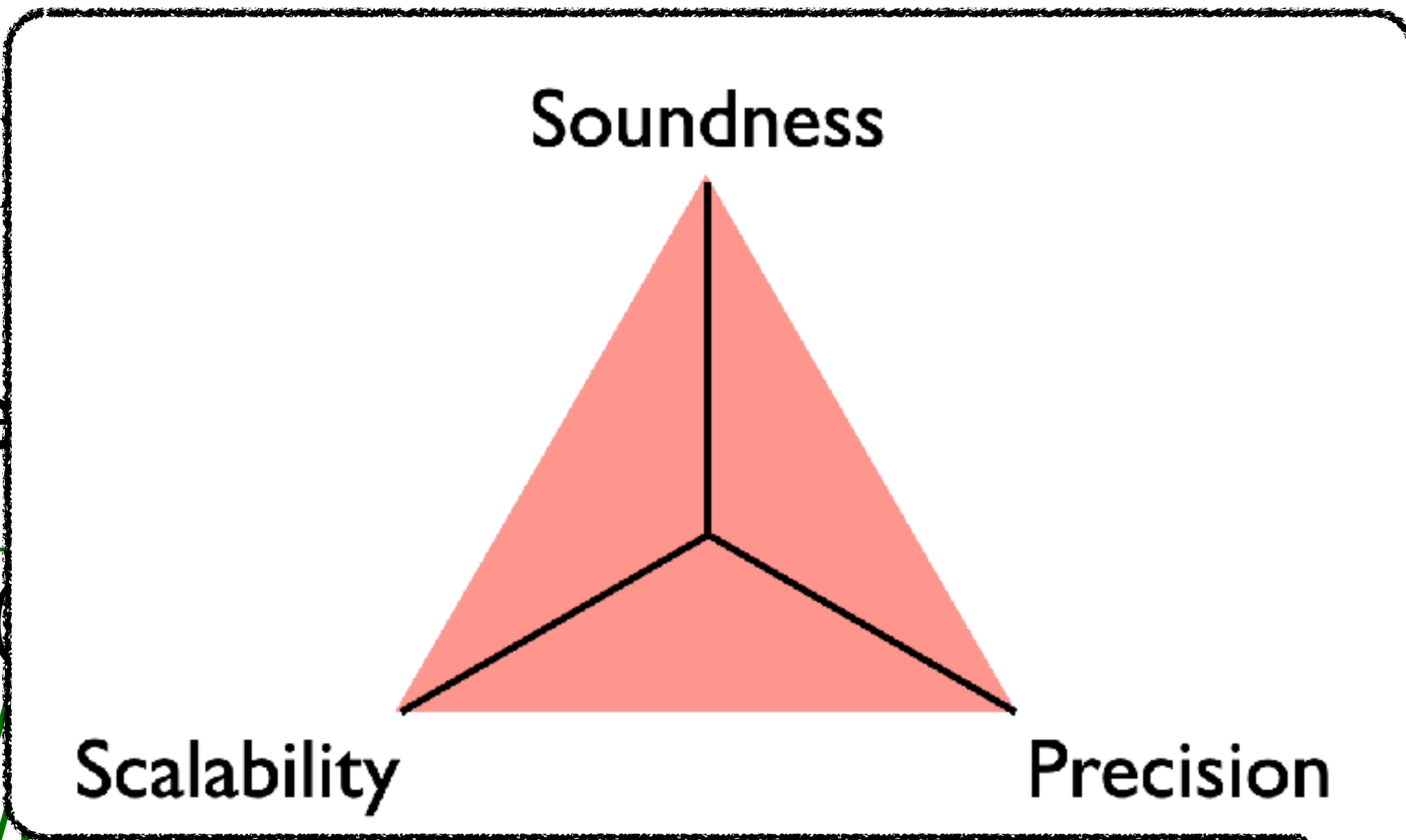
Apply 1-ctx : {**g**}

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

Context Sensitivity

Make the analysis fast and precise

- Sele

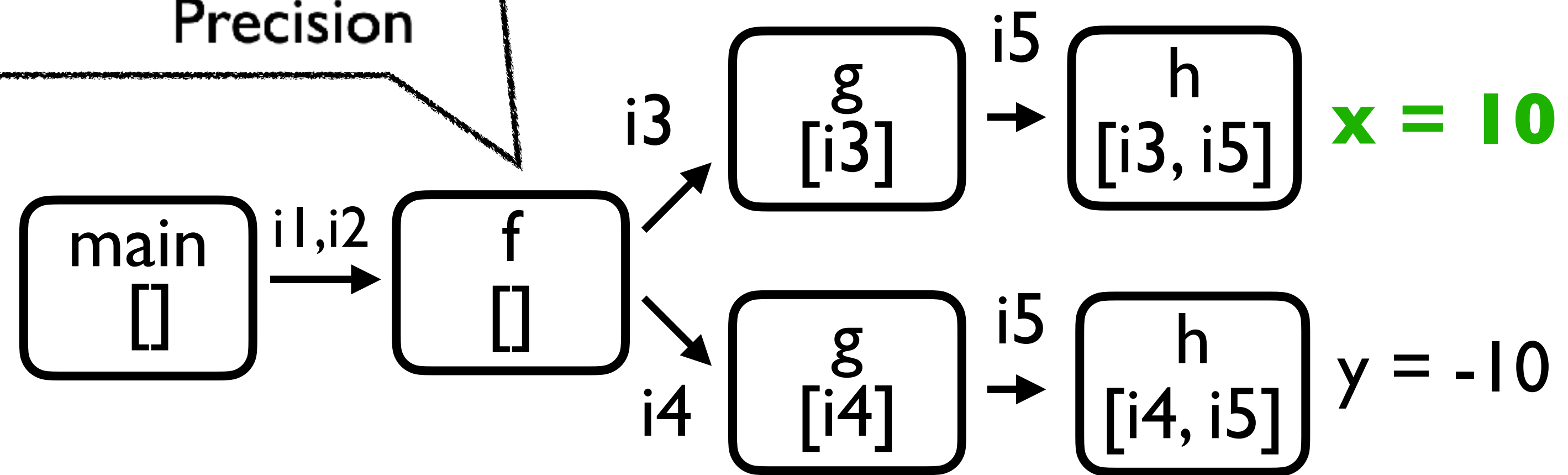


```

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

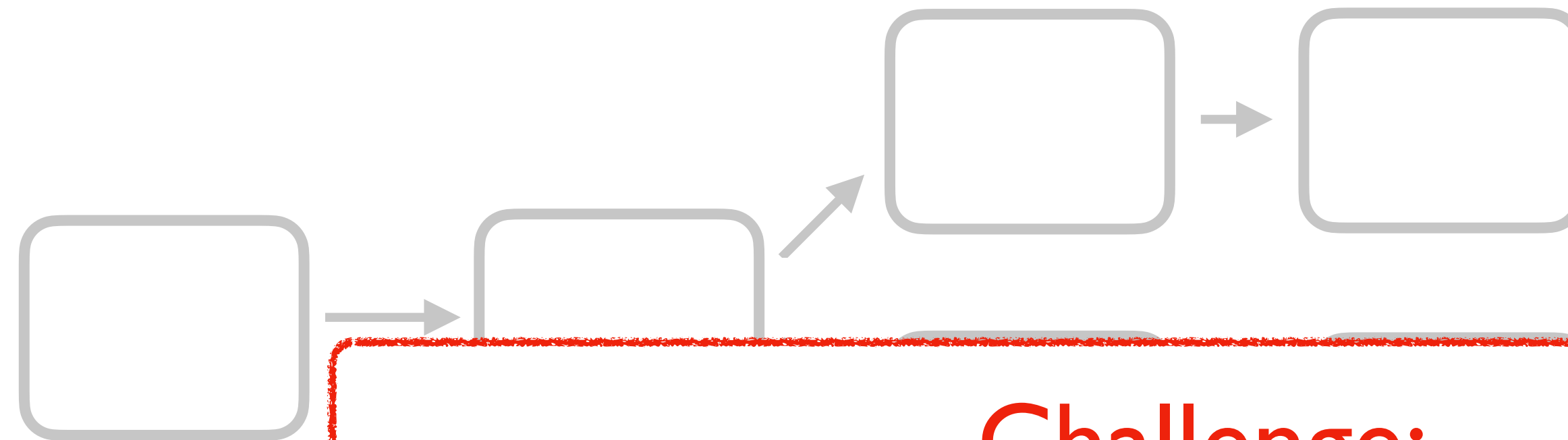
Can prove the query



Selective context sensitivity

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

Example program



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

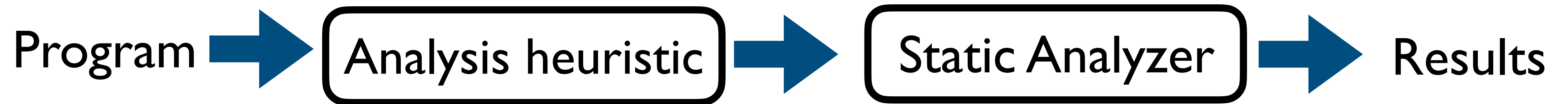
Challenge:

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

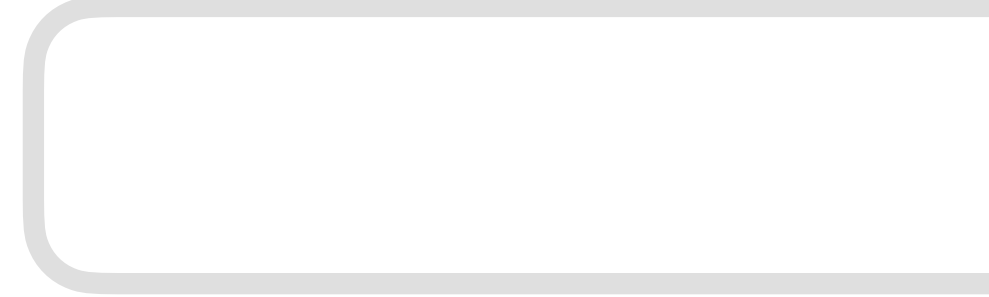
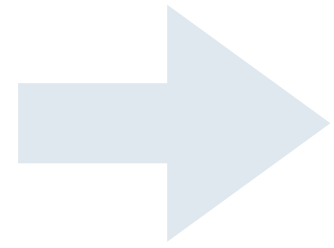
- Modern static analyzers need **analysis heuristics** to become practical



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

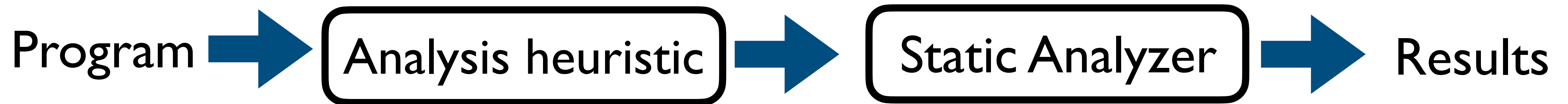
Existing: **manually designed** analysis heuristics



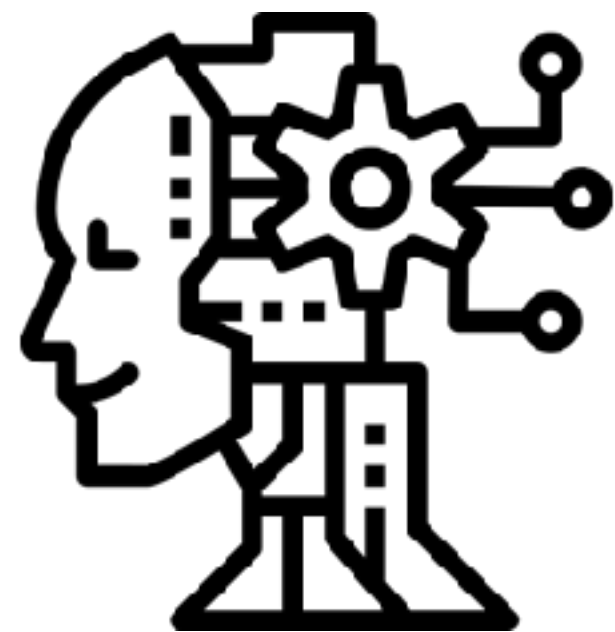
- “IFDS-based Context Debloating for Object-Sensitive Pointer Analysis” [ASE 2023]
- “Making Pointer Analysis More Precise by Unleashing the Power of Selective Context Sensitivity” [OOPSLA 2021]
- “Scalability-First Pointer Analysis with Self-Tuning Context-Sensitivity” [FSE 2018]
- “Precision-Guided 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**



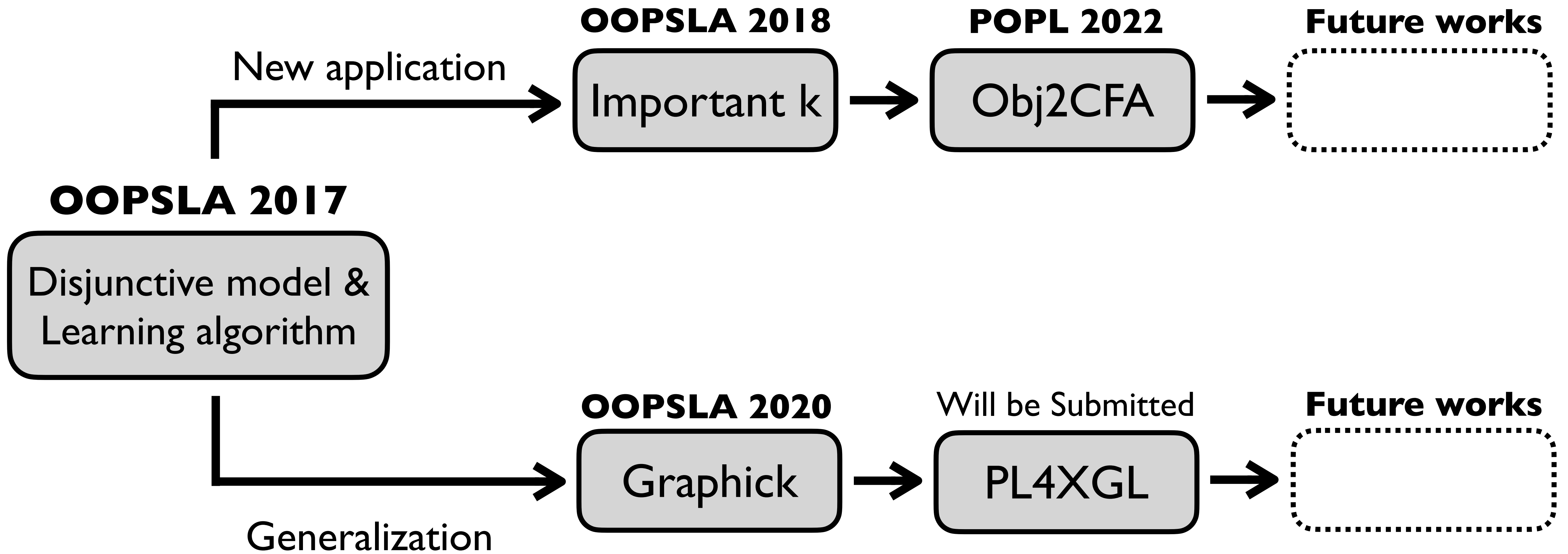
- **Automatically** generating **powerful** analysis heuristics



Learning algorithm



Data



New applications

Designed a learning framework

Static analyzer

Training data
(programs)

Atomic features
 (a_1, a_2, \dots, a_n)

Our framework

Learned selective context sensitivity heuristic

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

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

Generalization

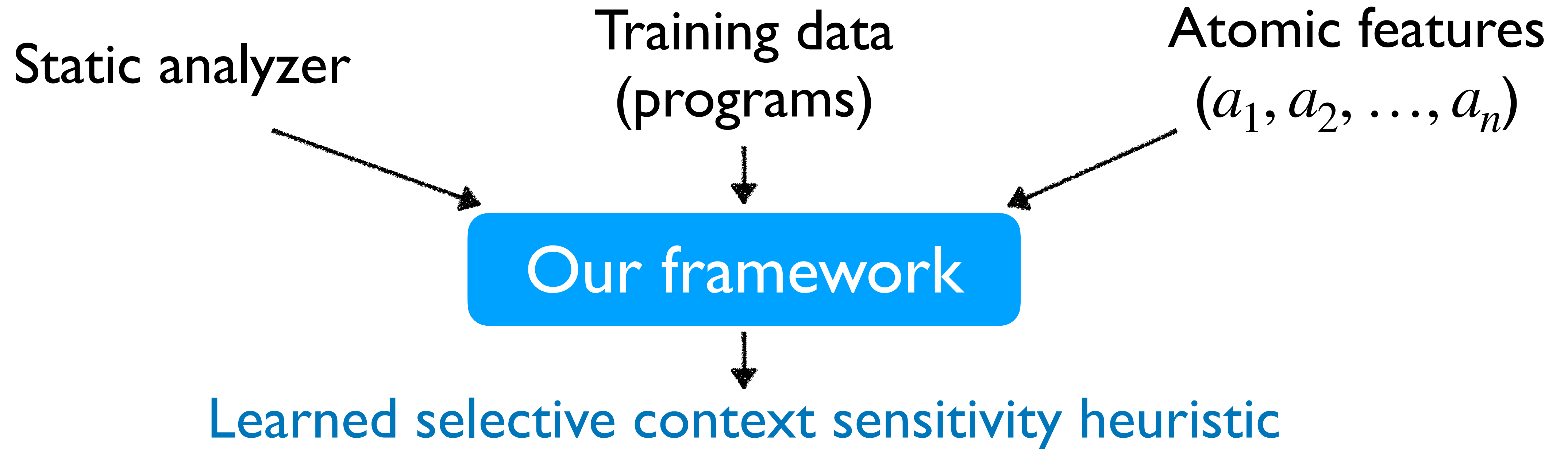
works

works

OOPSLA 2017

Disjunctive model &
Learning algorithm

Our Learning Framework



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

Static analyzer

- Static analyzer is modeled as a blackbox function F :

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

f_{2c}

f_{1ctx}

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

•

is

classification

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

box 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 :

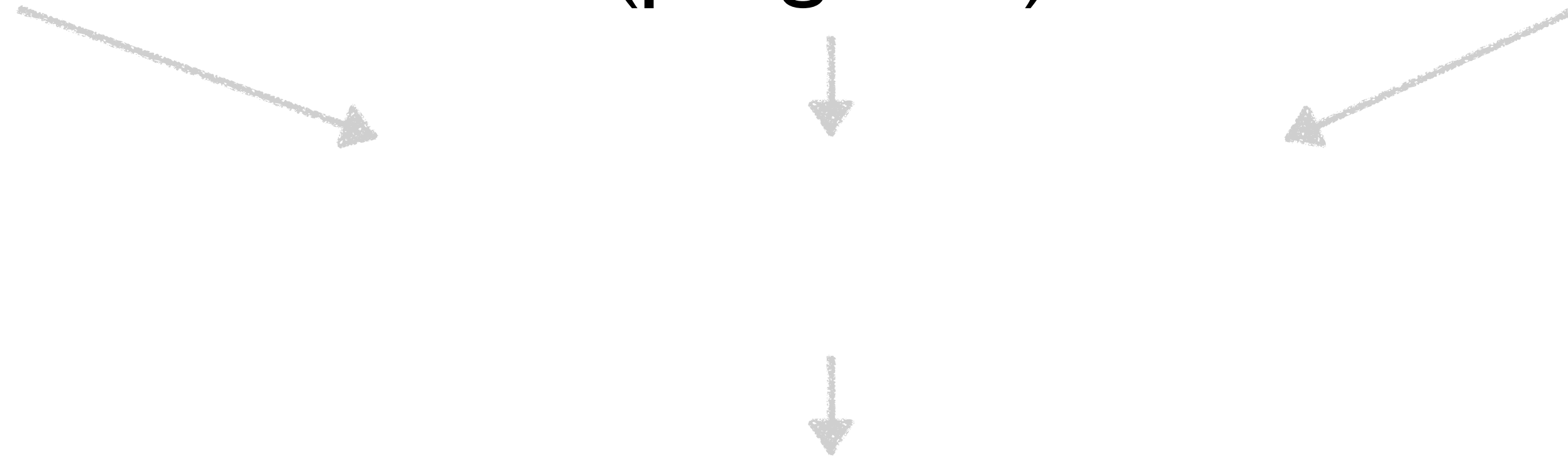
$$F(p, a) \Rightarrow 2^{\mathbb{Q}} \times \mathbb{N}$$

Queries proven to be safe

Analysis time

Small programs

Training data
(programs)



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

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

Atomic features

(a_1, a_2, \dots, a_n)

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

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

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

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

- minimizes analysis cost while is precise enough

Our framework

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

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

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

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

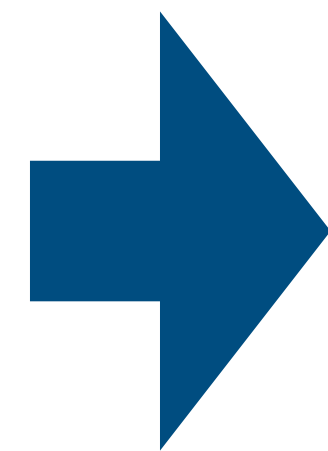
Methods

$f : \{a_1, a_2\}$

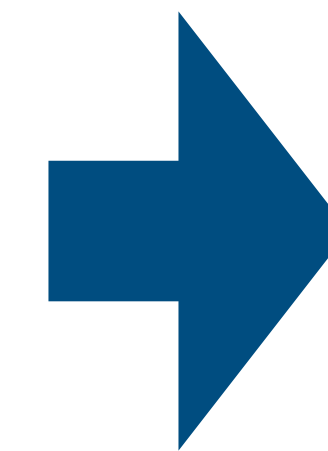
$h : \{a_1\}$

$g : \{a_2\}$

$m : \{\}$



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



Classification

2-ctx: $\{f, m\}$

1-ctx: $\{h\}$

0-ctx: $\{g\}$

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

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

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

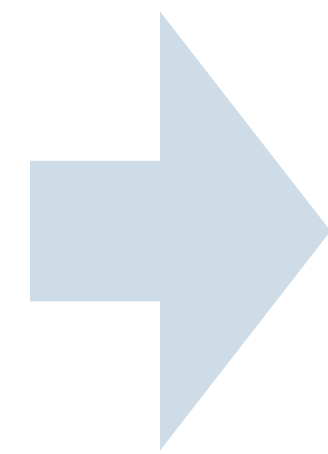
Methods

$f : \{a_1, a_2\}$

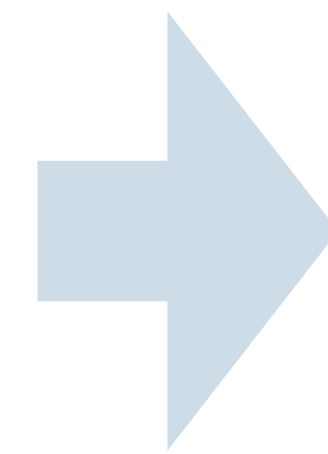
$h : \{a_1\}$

$g : \{a_2\}$

$m : \{\}$



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



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

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

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

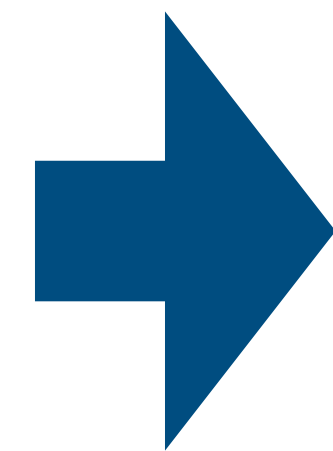
Methods

$f : \{a_1, a_2\}$

$h : \{a_1\}$

$g : \{a_2\}$

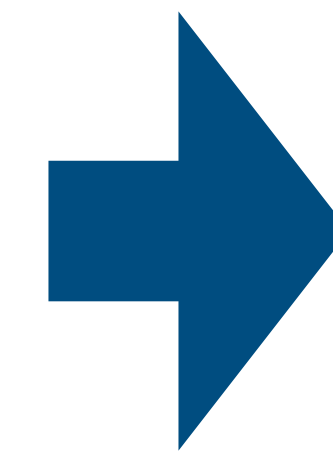
$m : \{\}$



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: $\{f, m\}$

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

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

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

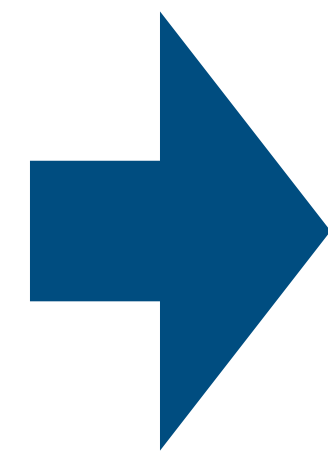
Methods

$f : \{a_1, a_2\}$

$h : \{a_1\}$

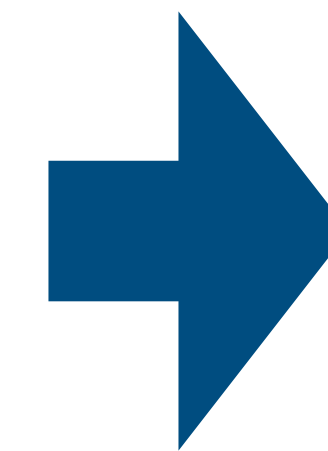
$g : \{a_2\}$

$m : \{\}$



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

I-ctx: $\{h\}$

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

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

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

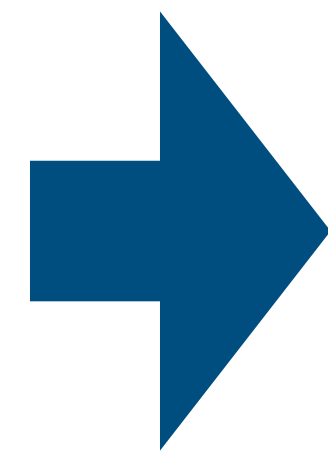
Methods

$f : \{a_1, a_2\}$

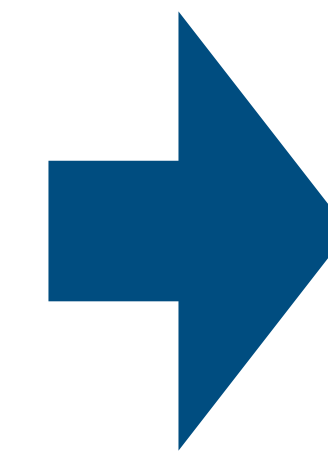
$h : \{a_1\}$

$g : \{a_2\}$

$m : \{\}$



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



Classification

2-ctx: $\{f, m\}$

1-ctx: $\{h\}$

0-ctx: $\{g\}$

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

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

$$f_{2ctx} = (a_1 \wedge a_3 \wedge a_6 \wedge a_8 \wedge \dots \wedge a_{25})$$
$$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \dots) \vee (a_3 \wedge a_4 \wedge a_7 \wedge \dots) \vee \dots$$

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

We learn each formula via greedy refinement

1. **Initialize** f to be the most general DNF formula

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

2. **Repeat** the following until no refinement is possible

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

1. **Choose a conjunction**, say c_i

2. **Refine the conjunction** with a feature a_j

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

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

(details in our paper)

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

We learn each formula via greedy refinement

I. **Initialize** f to be the most general DNF formula

$$f = a_1 \vee \neg a_1 \vee a_2 \vee \neg a_2 \vee \dots \vee a_n \vee \neg a_n \quad (\equiv \text{true})$$

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

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

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

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

2. **Repeat** the following until no refinement is possible

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

1. **Choose a conjunction**, say c_i

2. **Refine the conjunction** with a feature a_j

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

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

(details in our paper)



Our framework

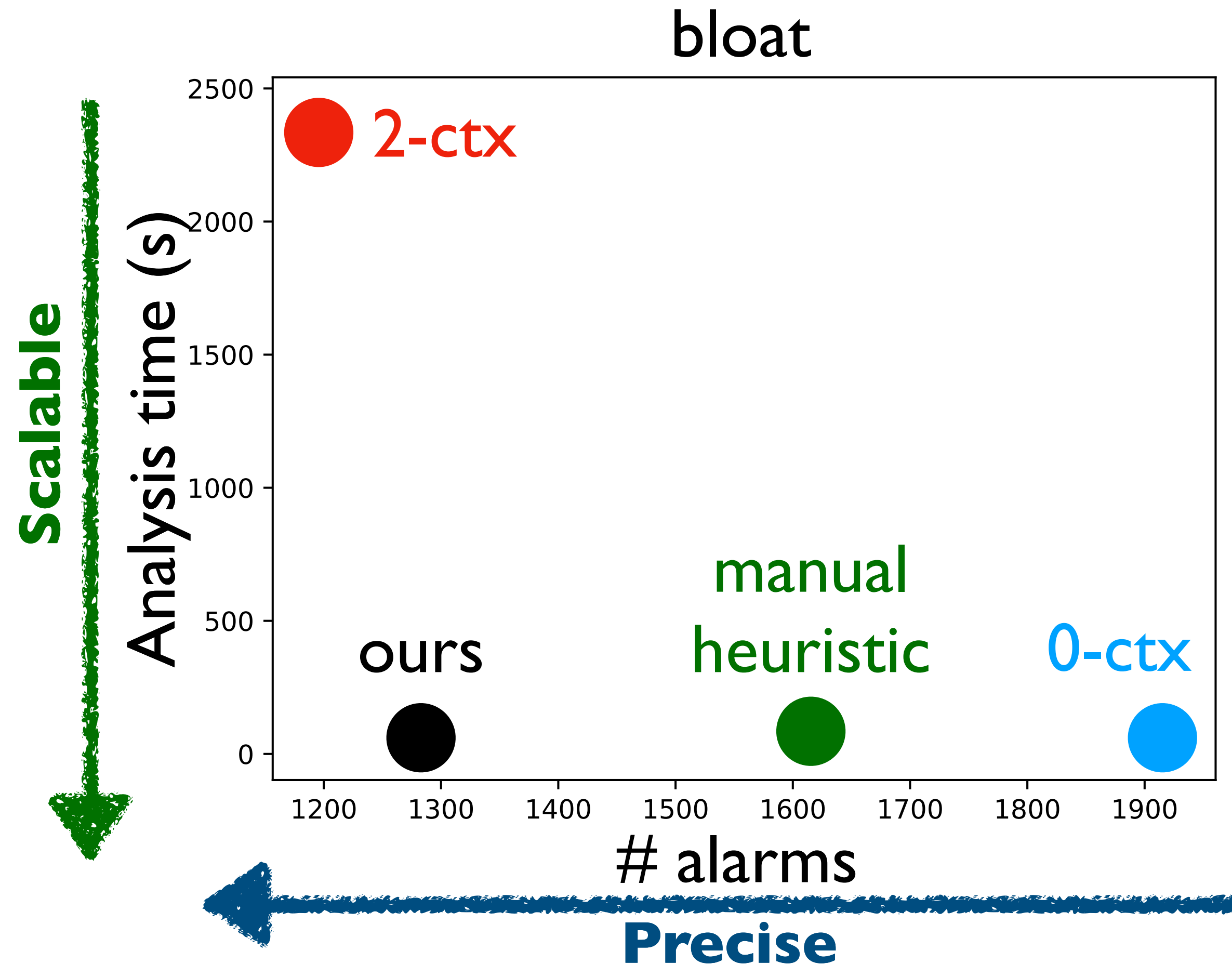
Learned selective context sensitivity heuristic

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

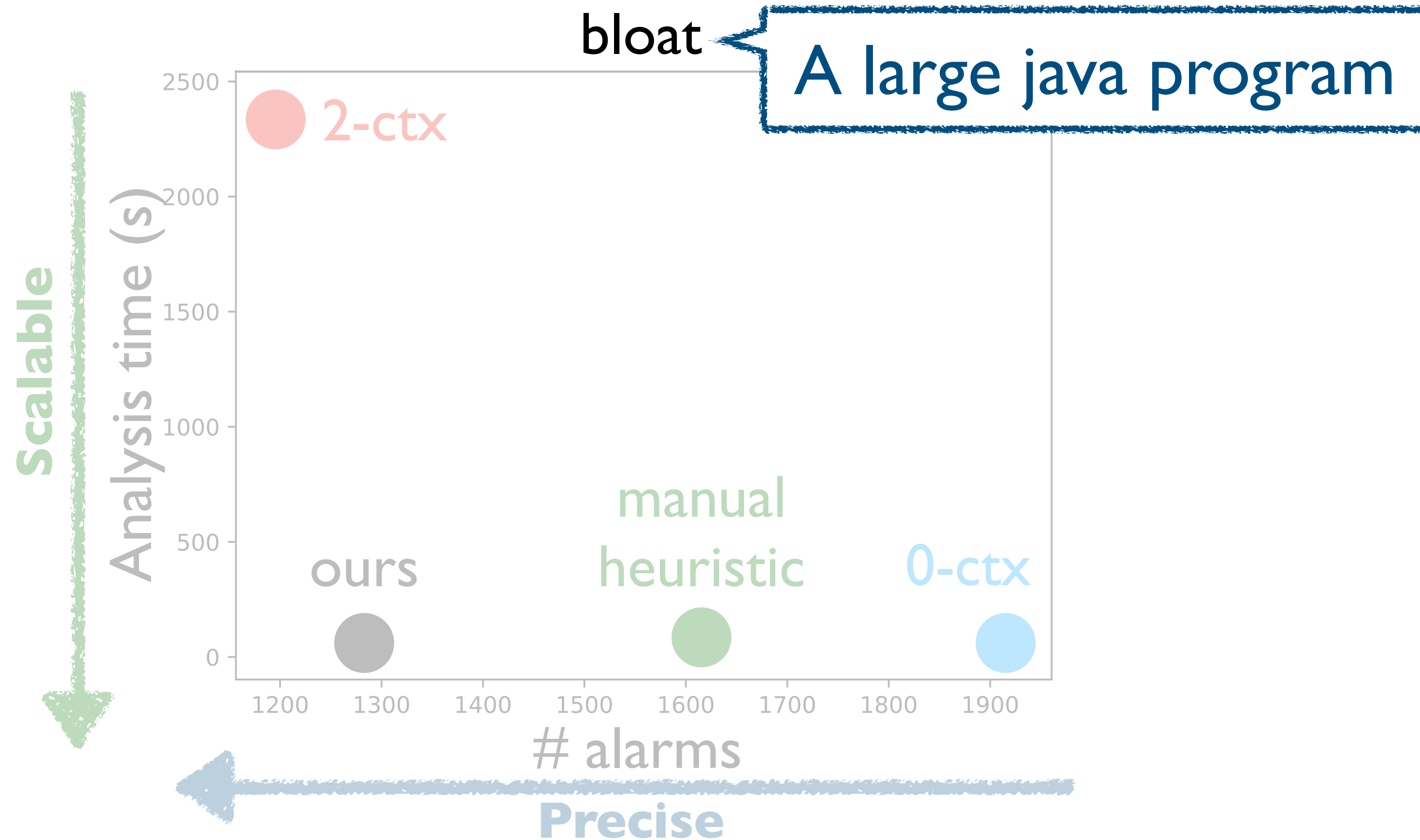
$$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \dots) \vee (\neg a_3 \wedge \neg a_4 \wedge \neg a_7 \wedge \dots) \vee \dots$$

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



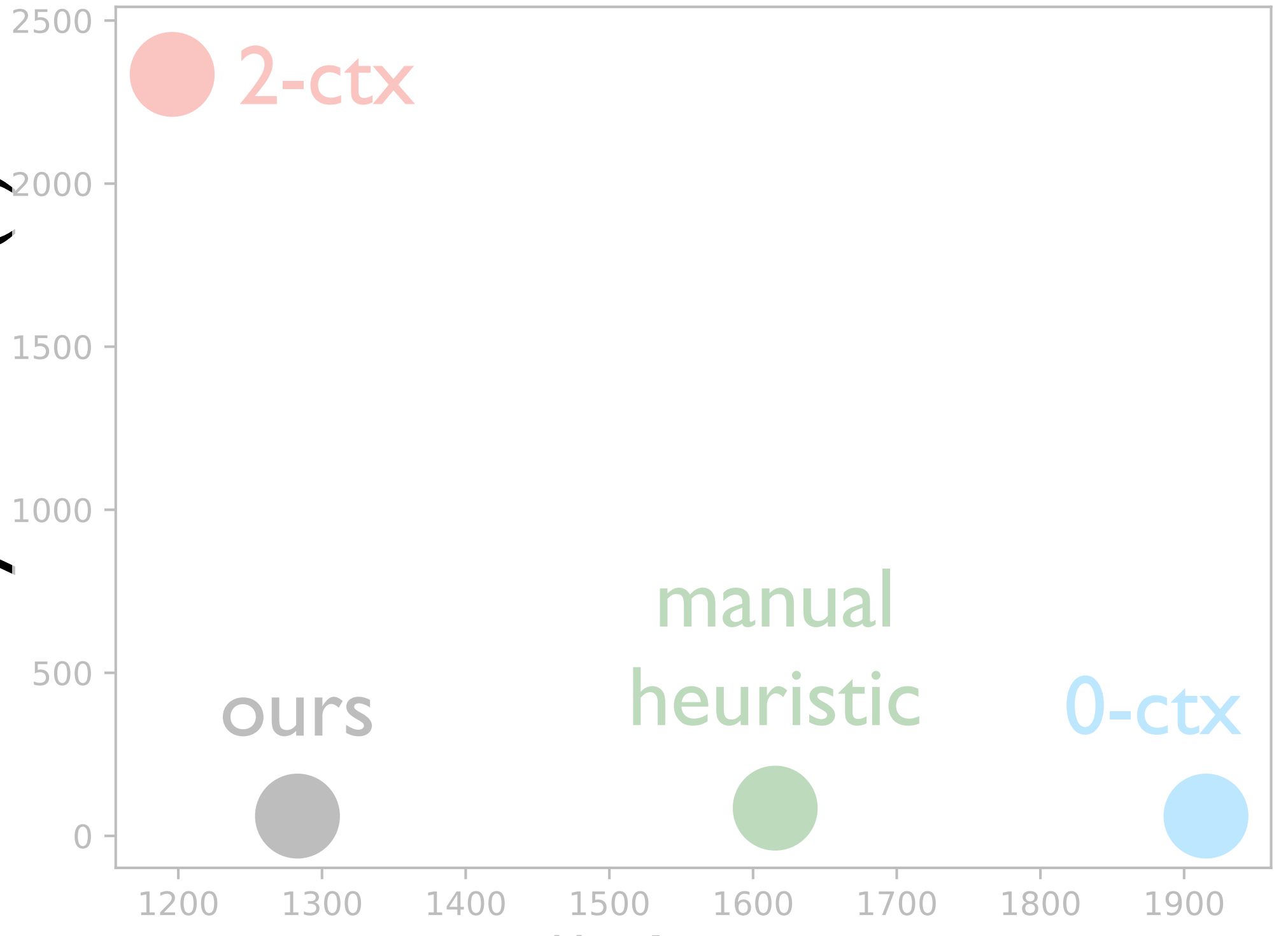
The lower is the faster

Scalable



Analysis time (s)

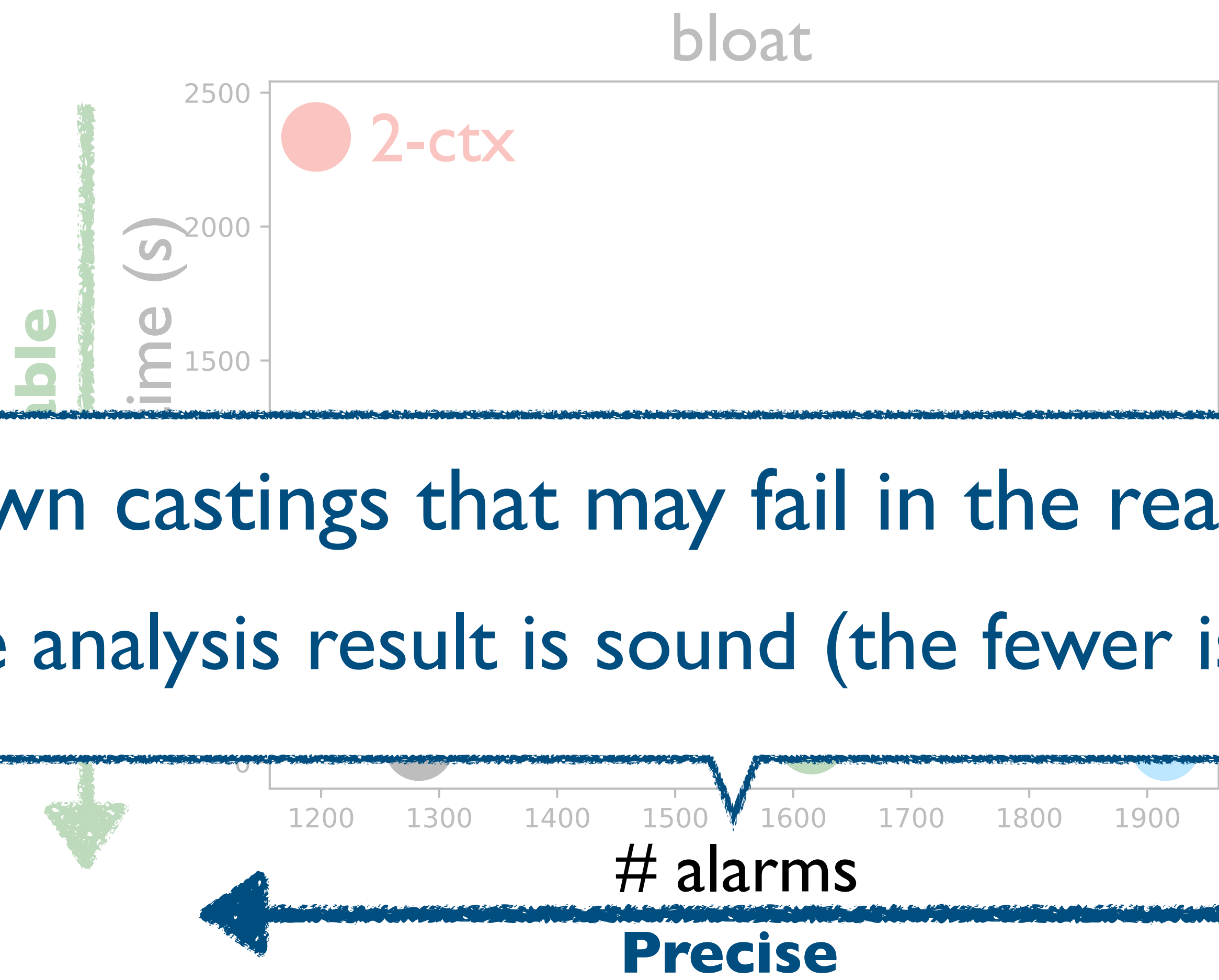
bloat



alarms

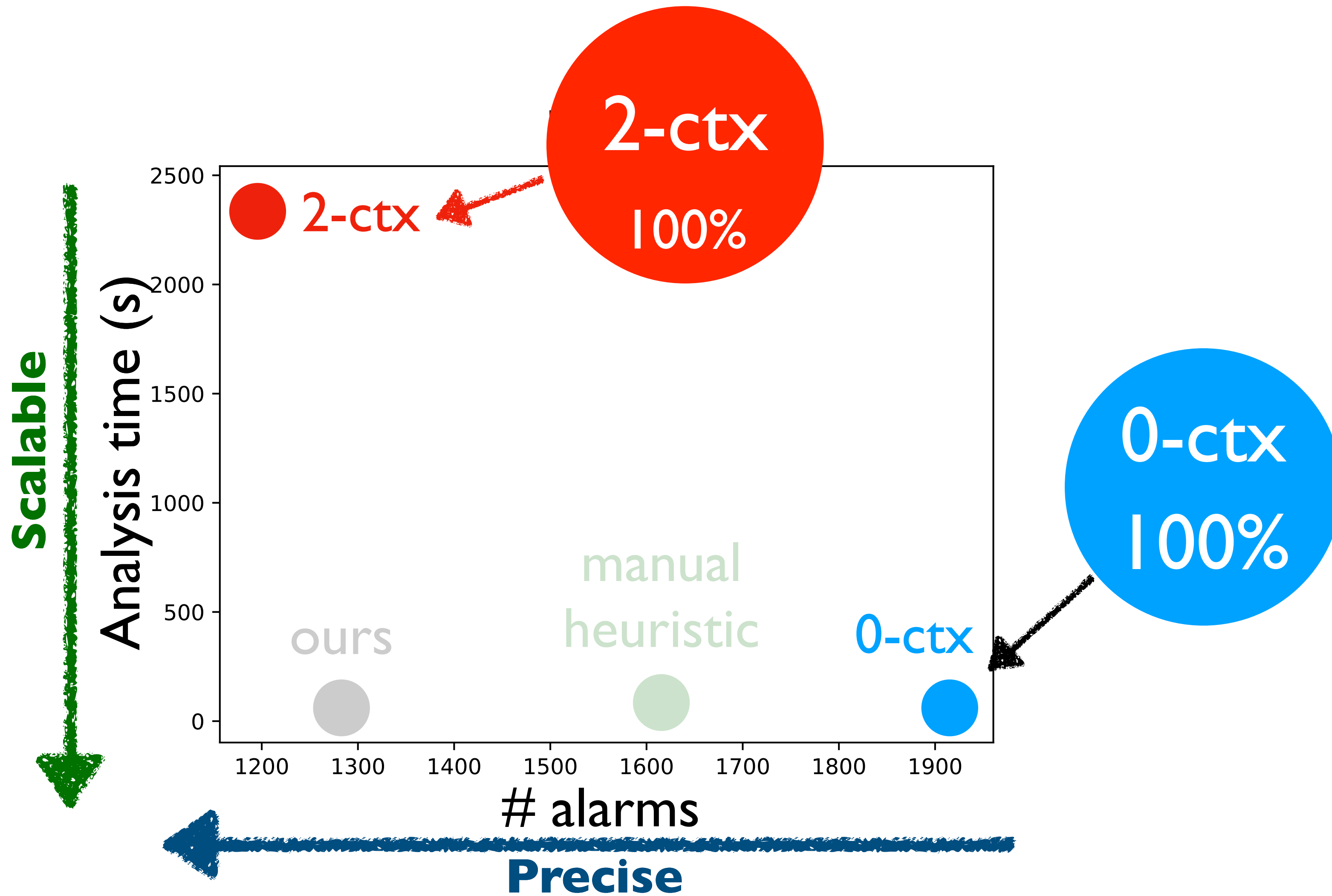
Precise

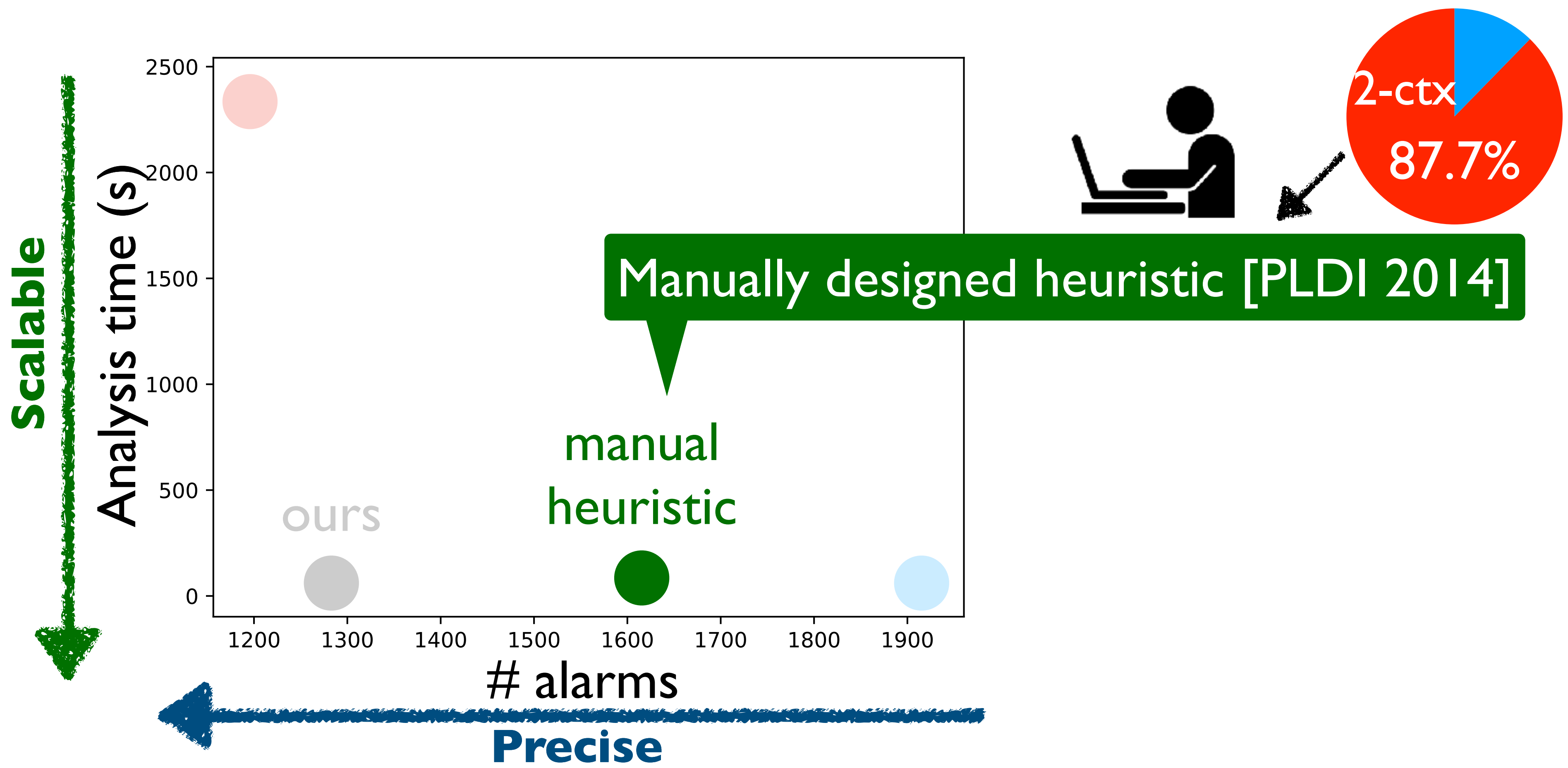




#down castings that may fail in the real execution

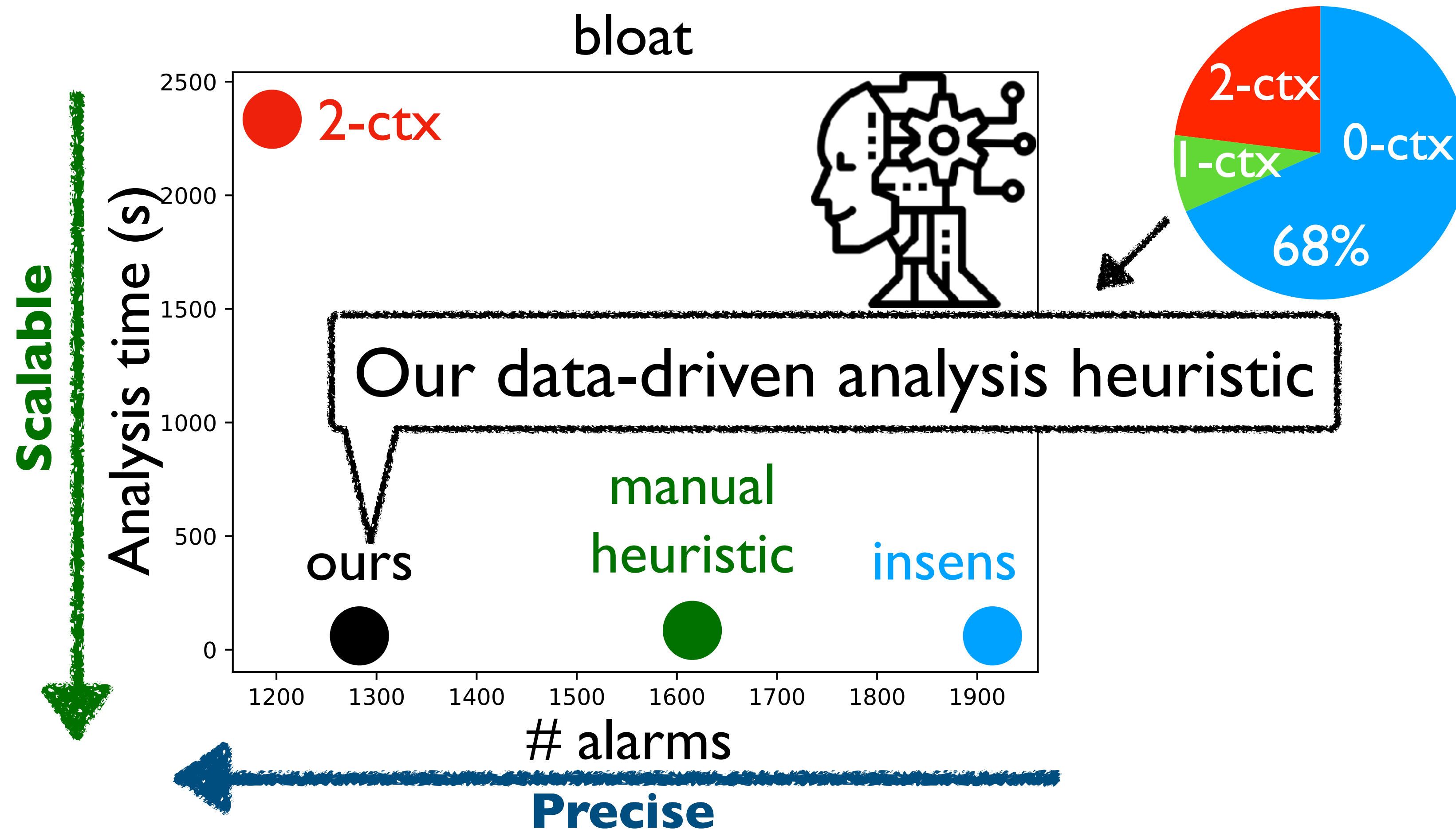
- The analysis result is sound (the fewer is the better)

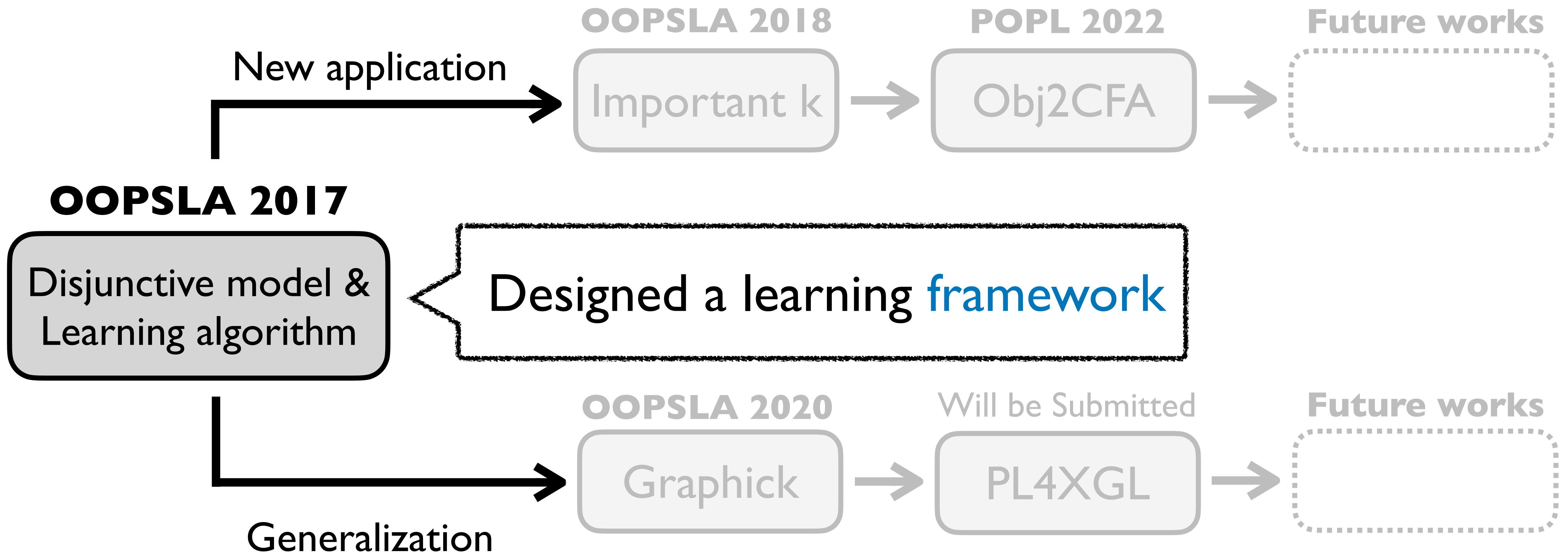


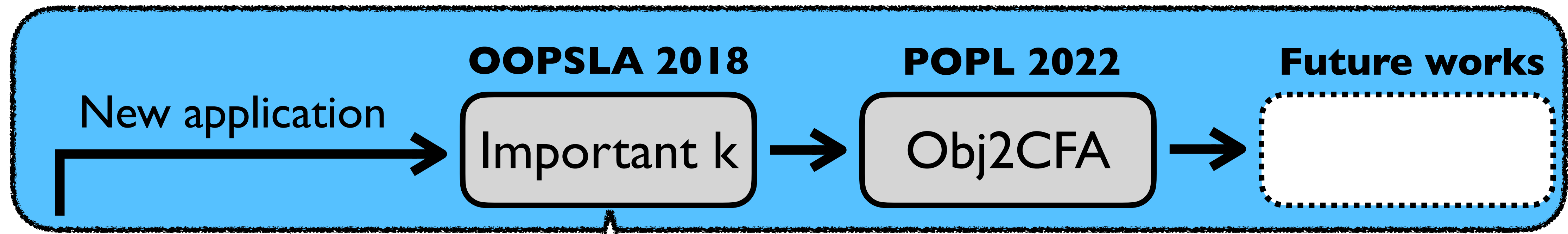


Performance Highlight of Our Learned Heuristic

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

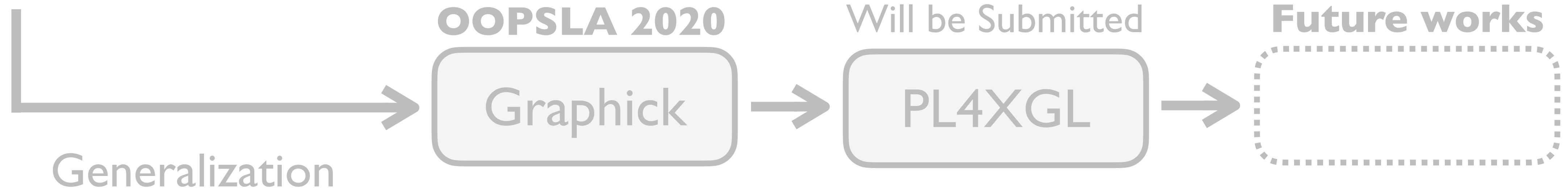






Found a **problem** in static analysis **used for 40 years**

OOPSLA
Disjunctive model &
Learning algorithm



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/10.1145/3274530>

*Corresponding author

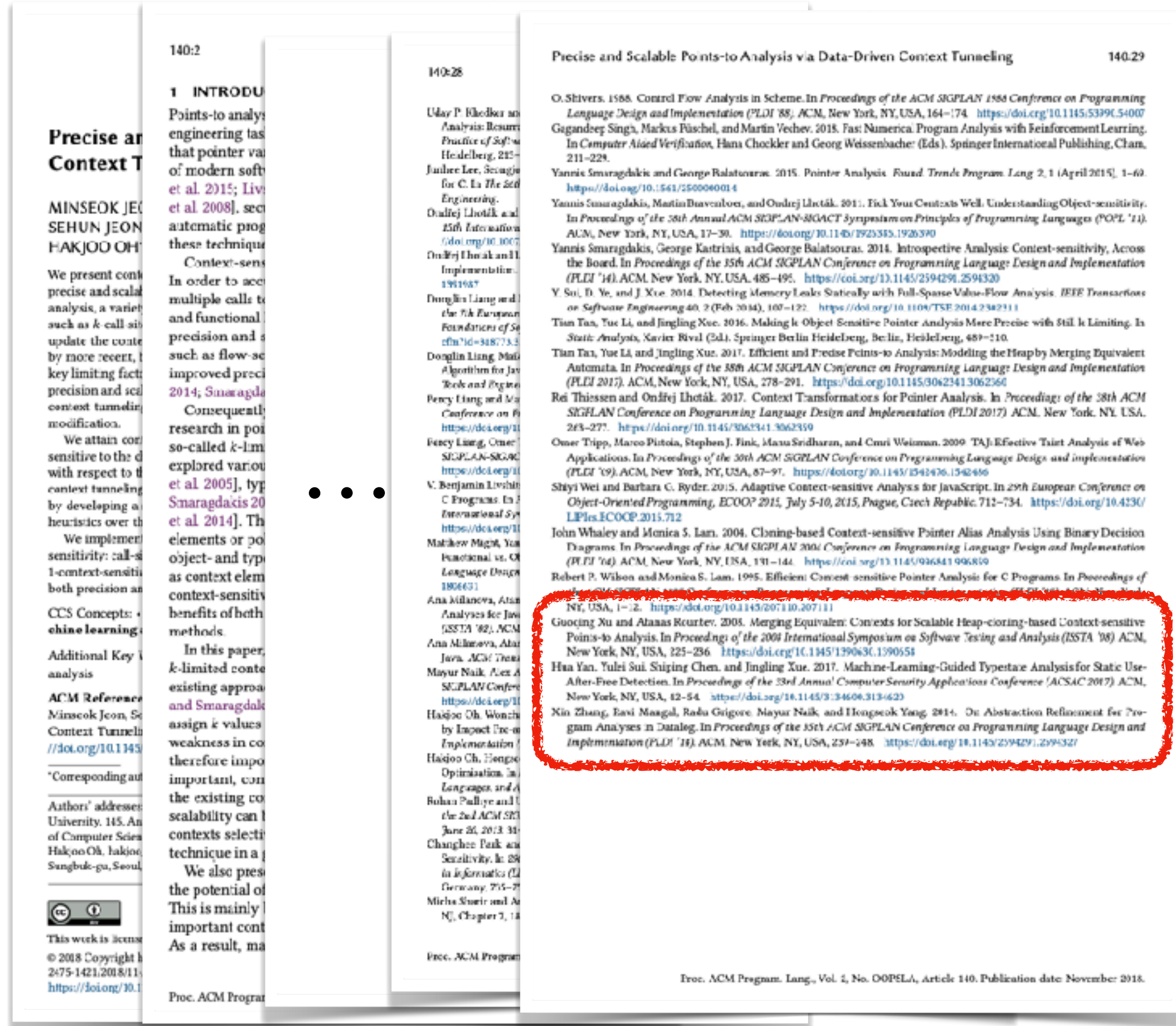
Authors' addresses: Minseok Jeon, minseok_jeon@korea.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Sungbuk-gu Seoul, 02841, Republic of Korea; Sehun Jeong, gfcanga@korea.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Sungbuk-gu Seoul, 02841, Republic of Korea; Hakjoo Oh, hakjoo_oh@korea.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Sungbuk-gu, Seoul, 02841, Republic of Korea.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2018 Copyright held by the owner/author(s).
2475-1421.2018.11.ART140
<https://doi.org/10.1145/3274530>

Paper (29 pages)

A Key Limiting Factor in Static Analysis

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

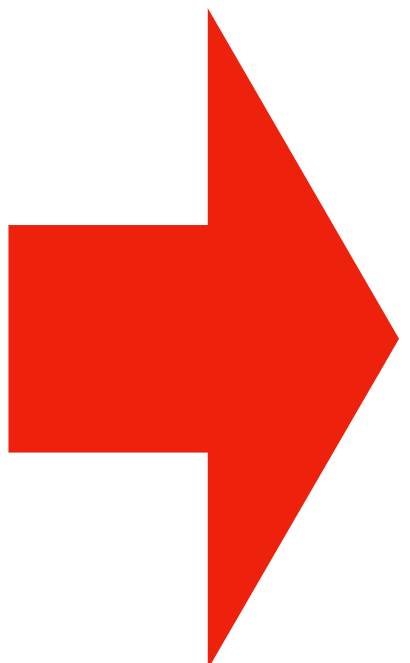


Which grade would you give?

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



Paper (29 pages)

for 40 years

Existing static analyzers use **last-k** context abstraction

Doop

Safe



WALA

T. J. WATSON LIBRARIES FOR ANALYSIS



...

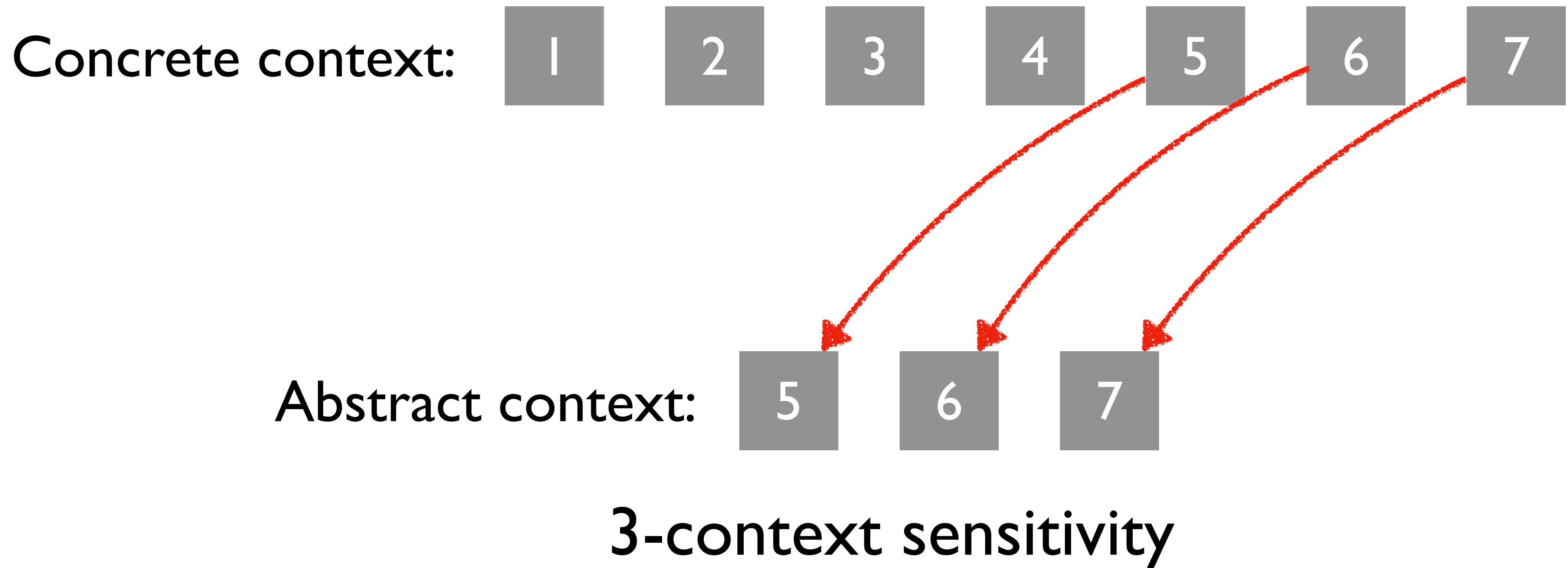
Java

Javascript

C

A **Key Limiting Factor** in Static Analysis

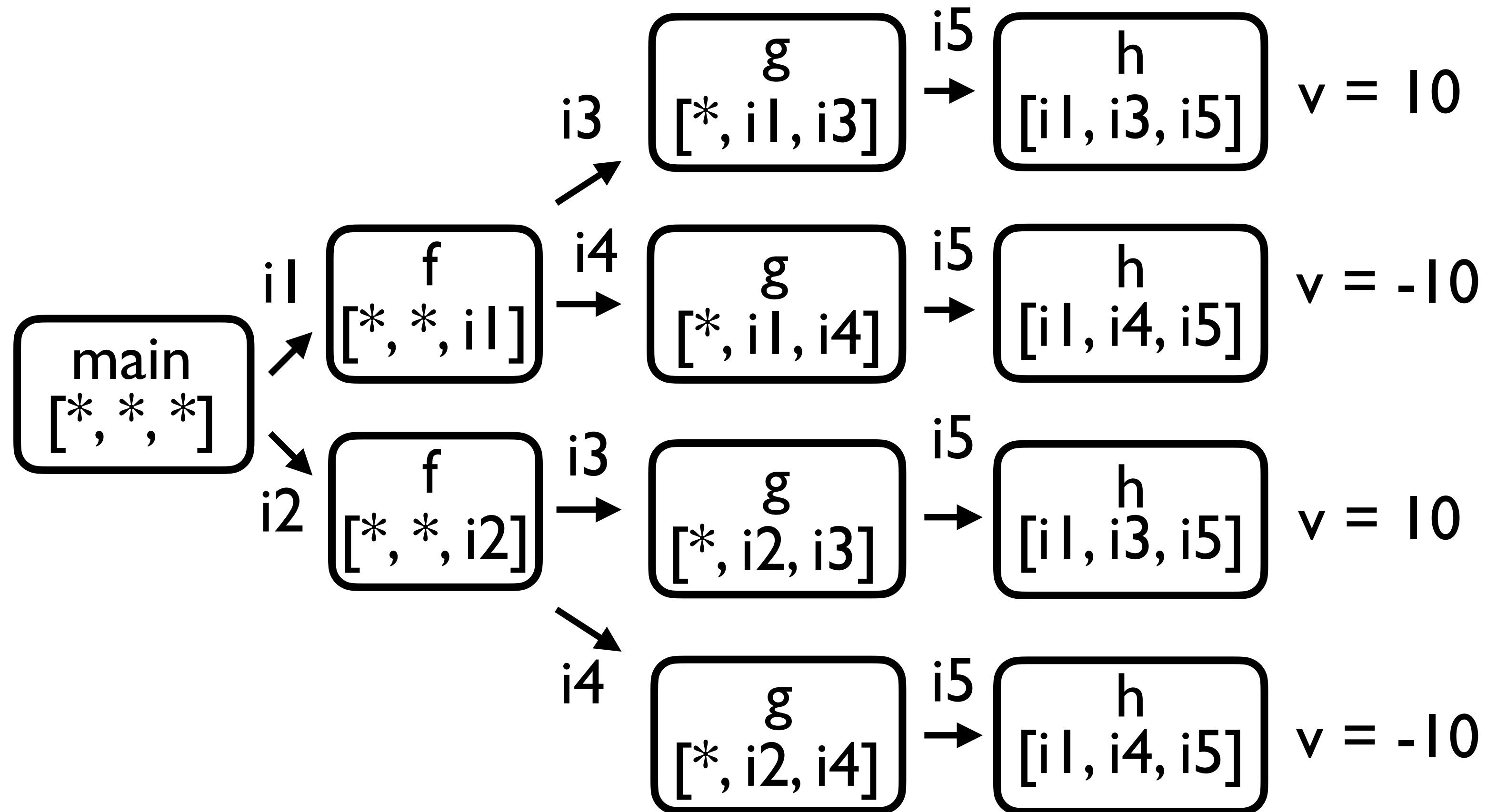
- Conventional k-context sensitivity **keeps the last k** Used for 40 years



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;}
```

Example program

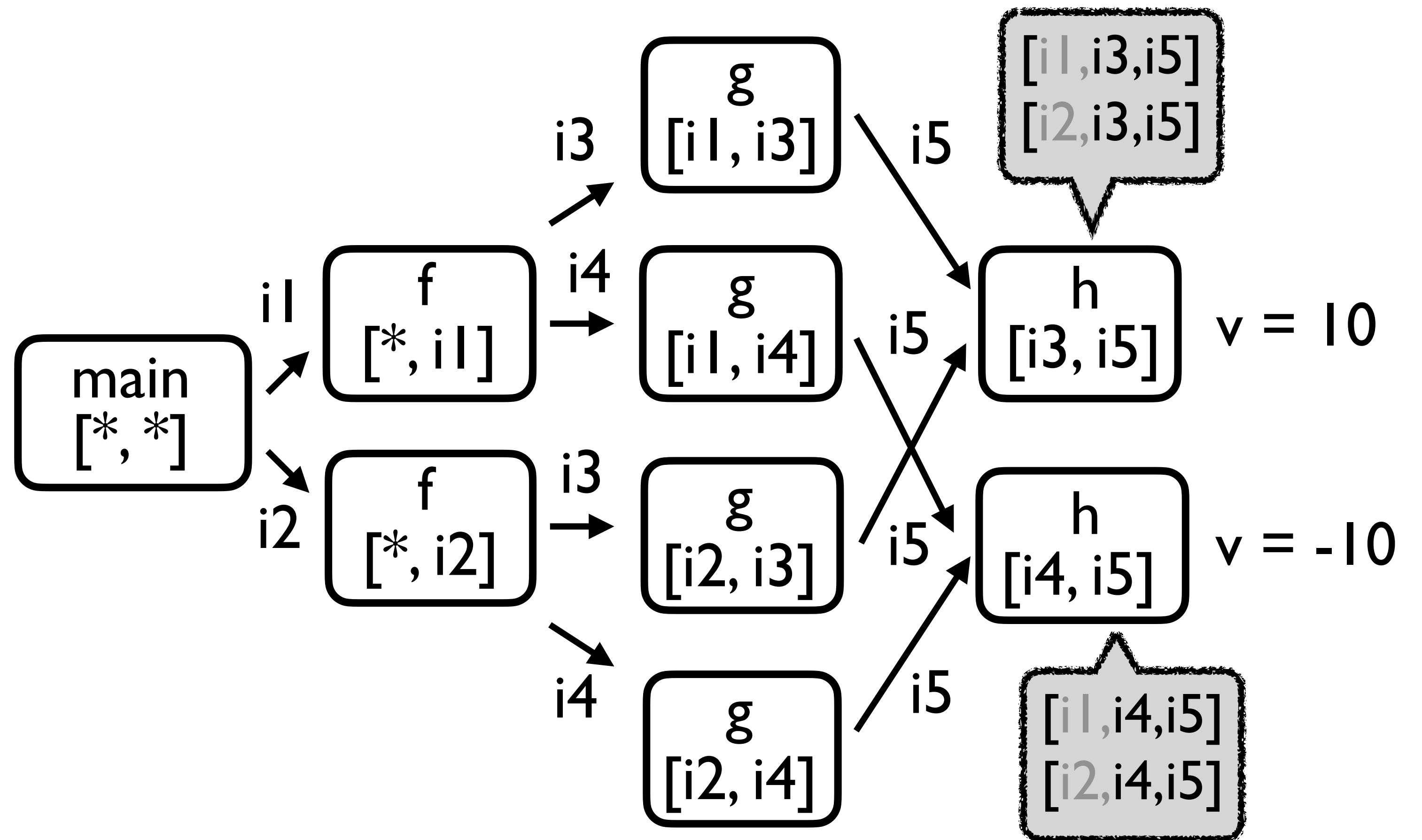


3-context sensitivity

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

Abandoned!

Concrete context:



Abstract context:



3-context sensitivity



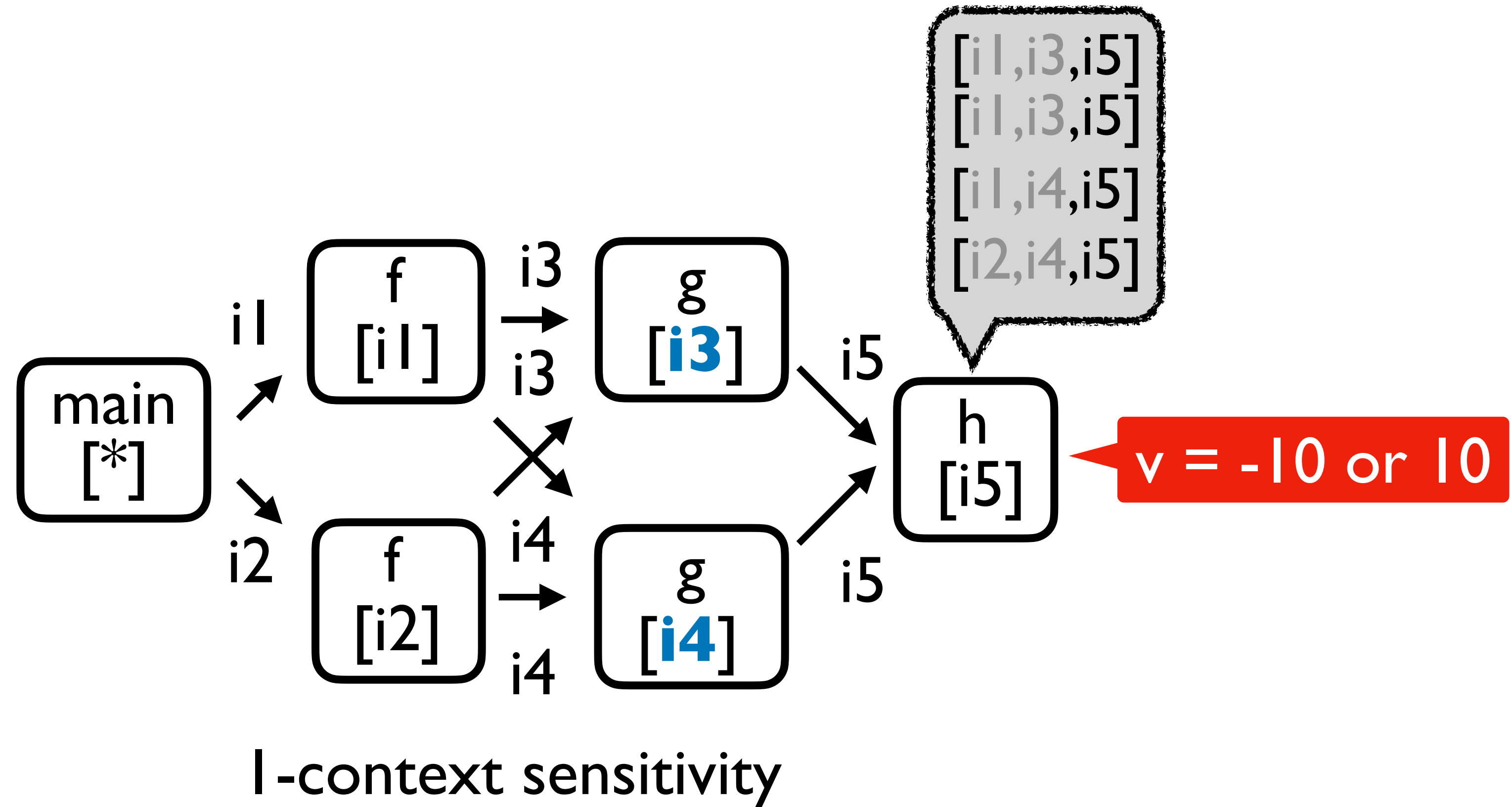
A Key Limiting Factor in Static Analysis

```
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;}
```

x = 10 or -10

v = -10 or 10

unable to prove the query



Example program

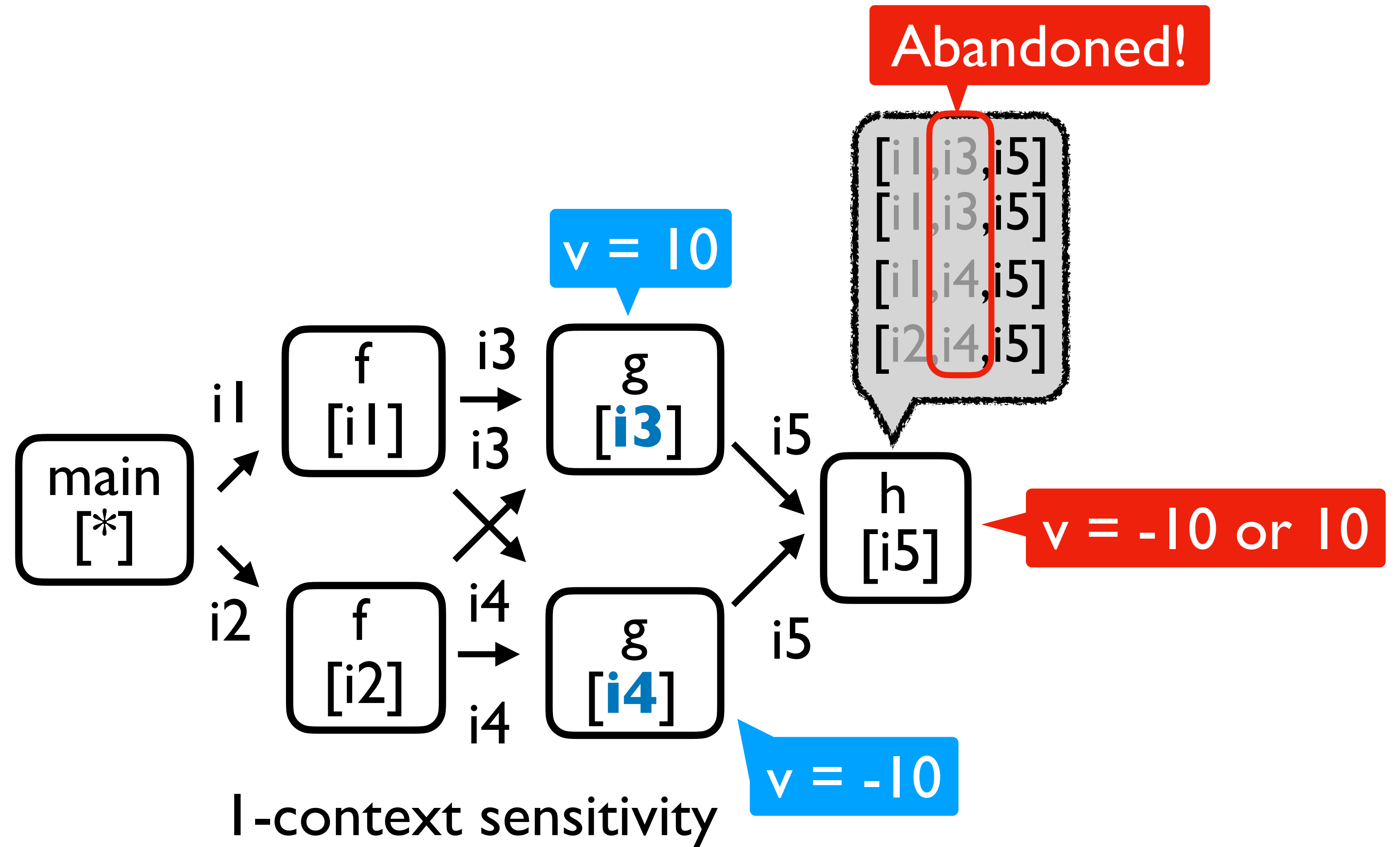
A Key Limiting Factor in Static Analysis

```
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

x = g(10);//i3
y = g(-10);//i4

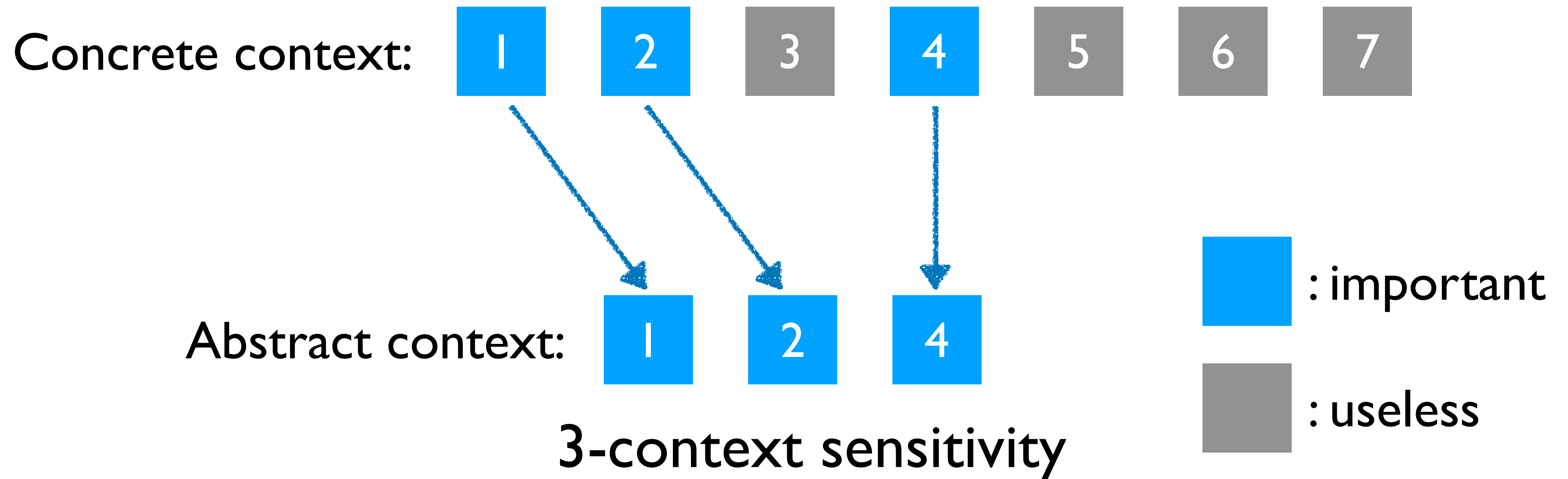
unable to prove the query



Example program

Our Solution: Keep Important K

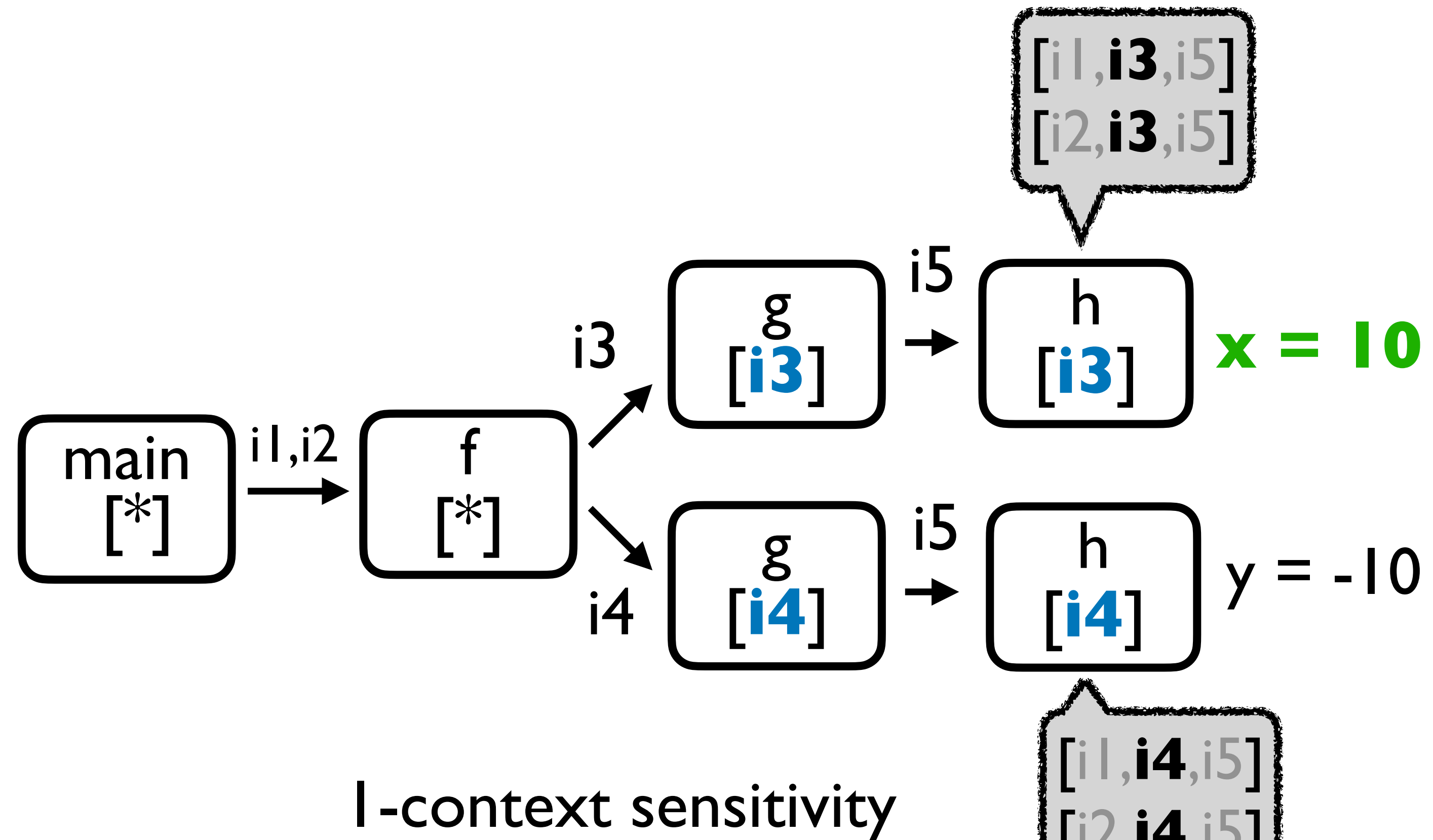
- Our solution is to keep the most important k instead of the last k



A Key Limiting Factor in Static Analysis

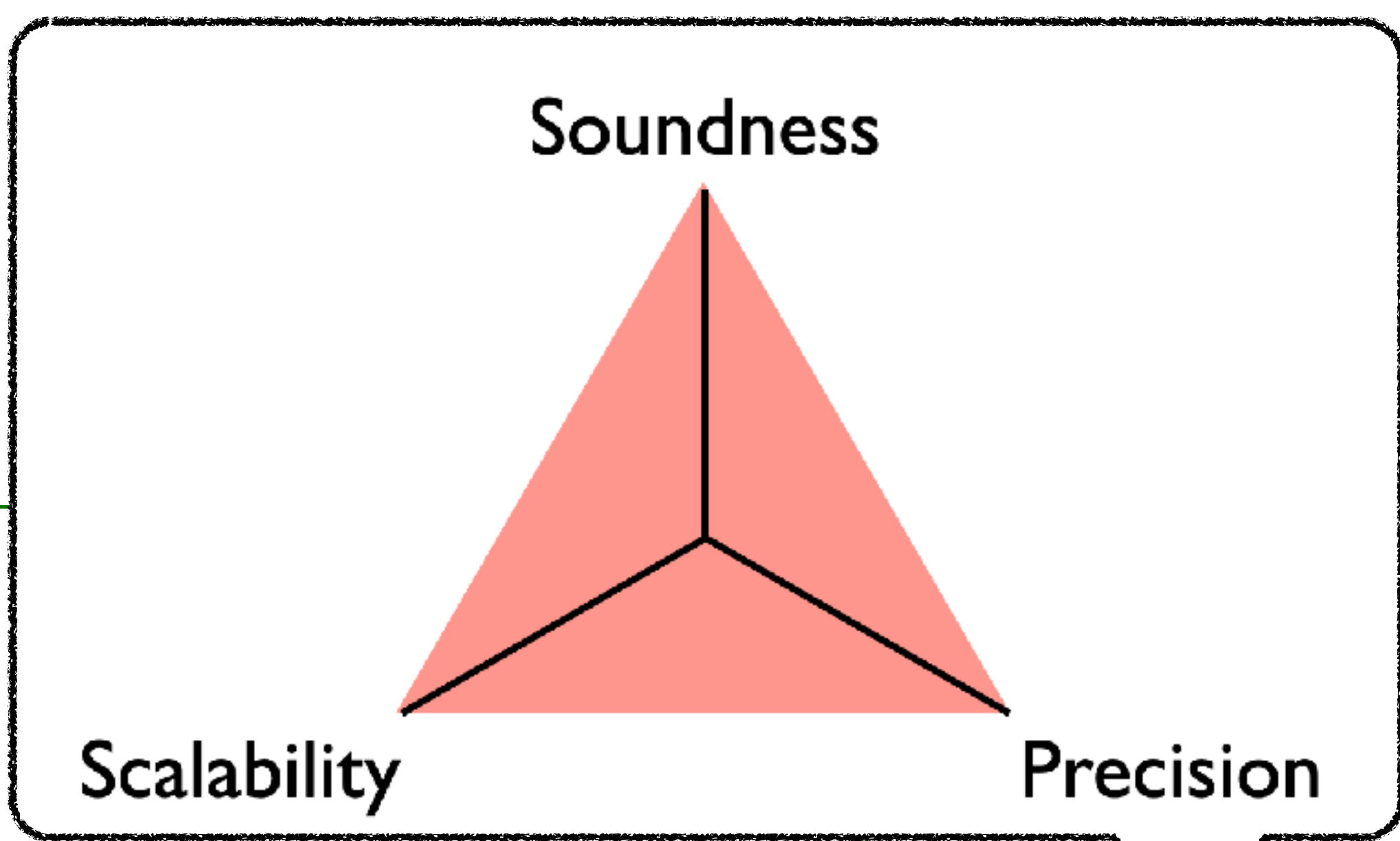
```
main(){
  f();//i1
  f();//i2
}
f(){
  x = g(10);//i3
  y = g(-10);//i4
  assert (x > 0);//query
}
g(v){ret h(v);}//i5
h(v){ret v;}
```

Example program



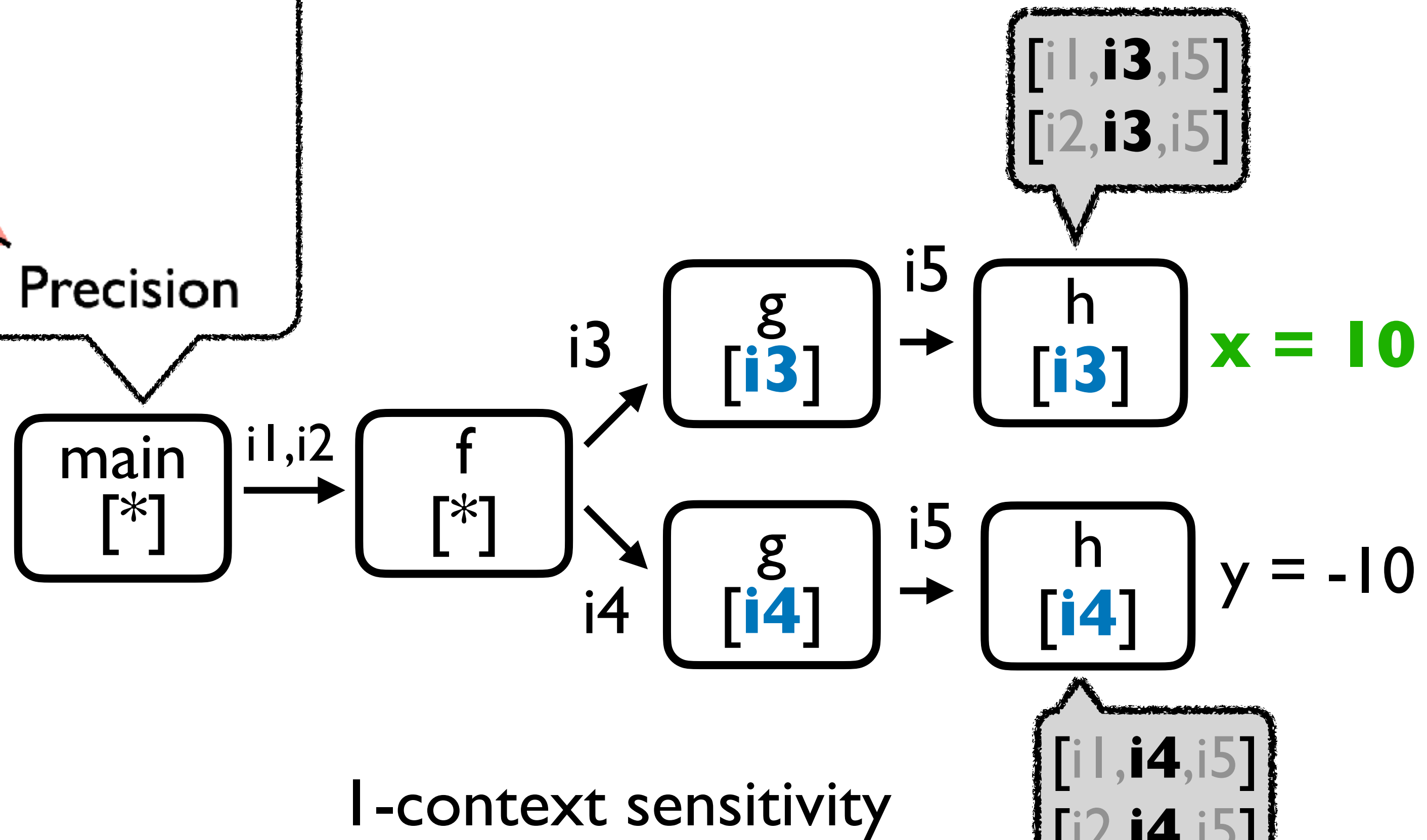
Important : {i3,i4}
Unimportant : {i1,i2,i5}

Factor in Static Analysis



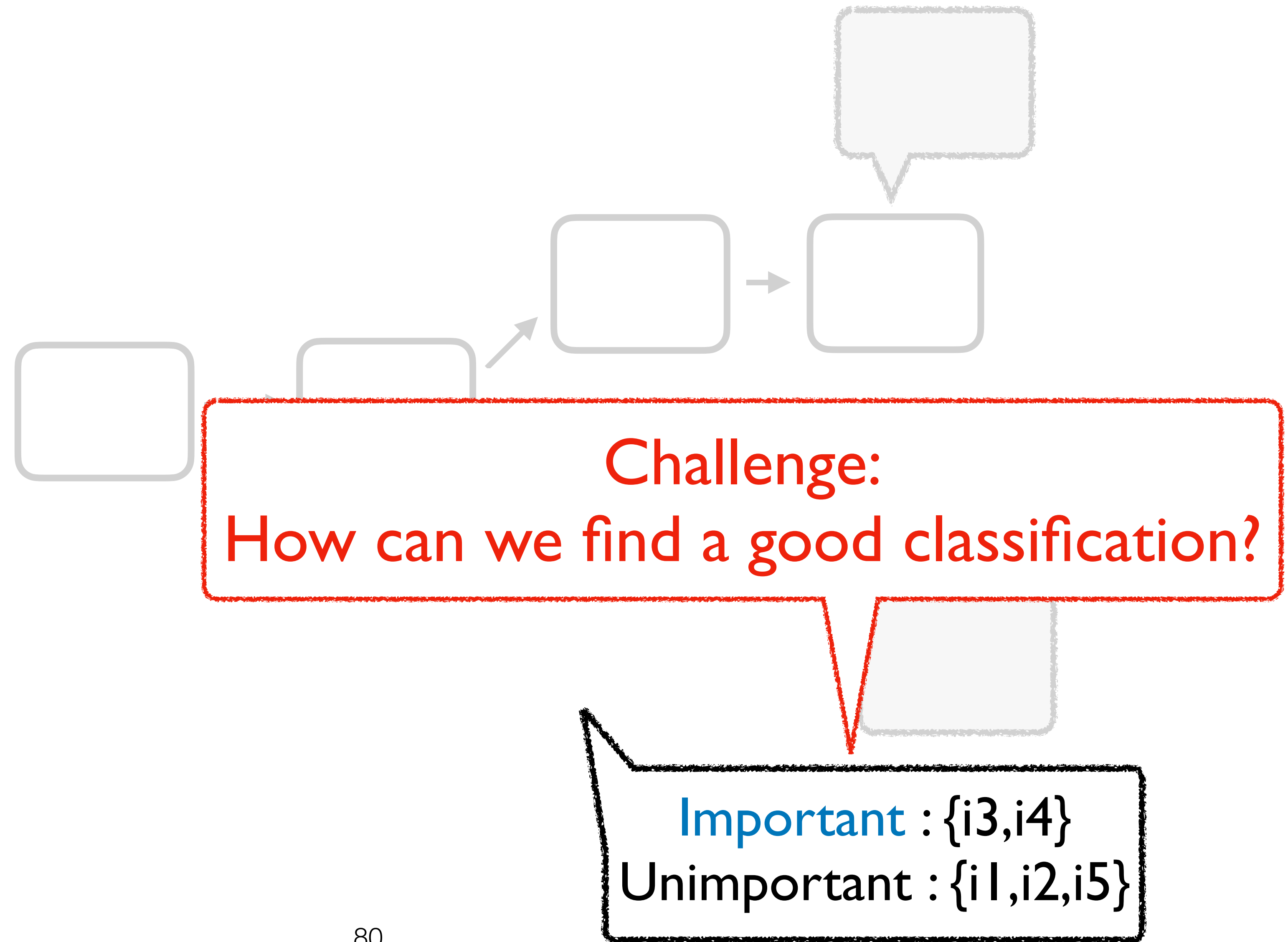
```
}  
f(){  
  x = g(10); //i3  
  y = g(-10); //i4  
  assert (x > 0); //query  
}  
g(v){ret h(v);} //i5  
h(v){ret v;}
```

Example program



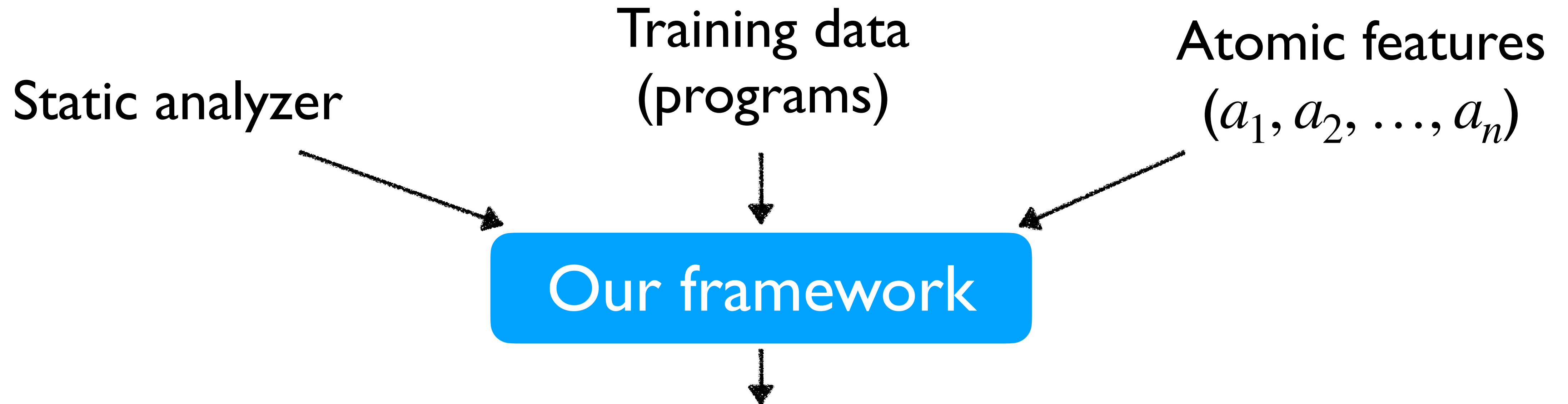
Important : {i3, i4}
Unimportant : {i1, i2, i5}

A **Key Limiting Factor** in Static Analysis



Our Learning Framework

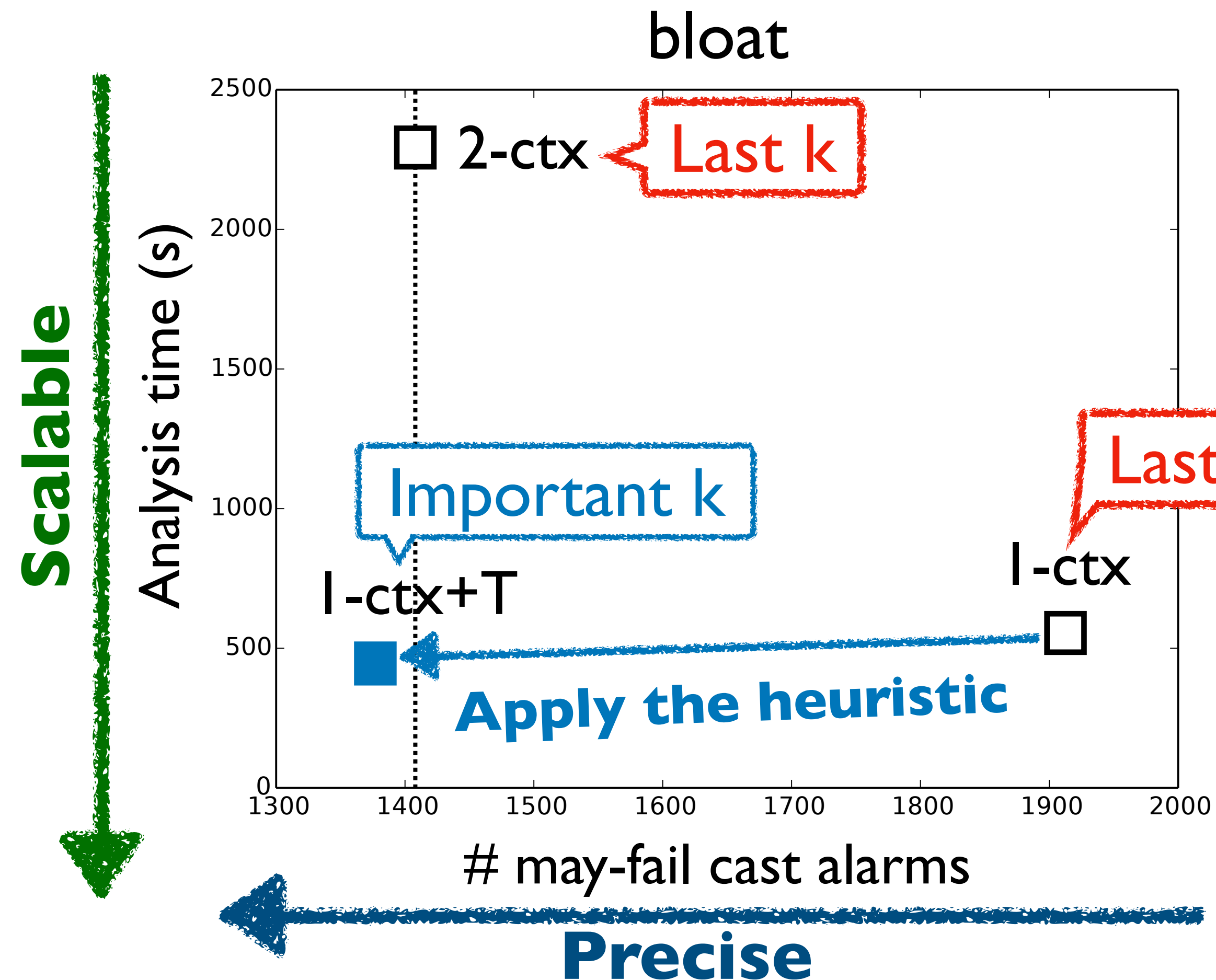
- We designed a **framework** for learning unimportant context elements



$$f = (a_1 \wedge \neg a_2 \wedge \neg a_3 \wedge \dots) \vee (a_1 \wedge \neg a_3 \wedge a_7 \wedge \dots) \vee \dots$$

Performance of Our Learned Heuristic

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



Performance

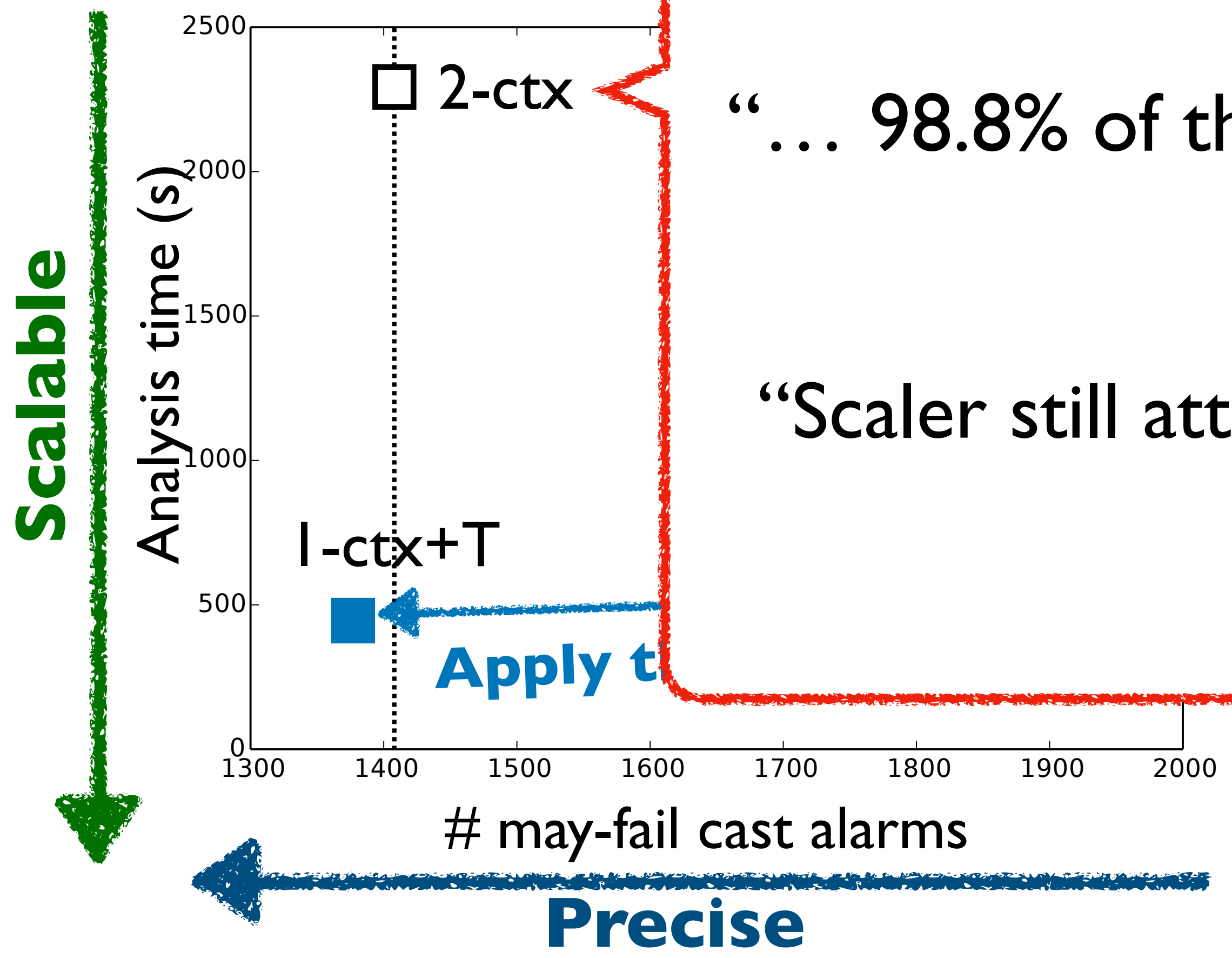
- 2-ctx had been used as a **precision upper bound**

- I-ctx with impo

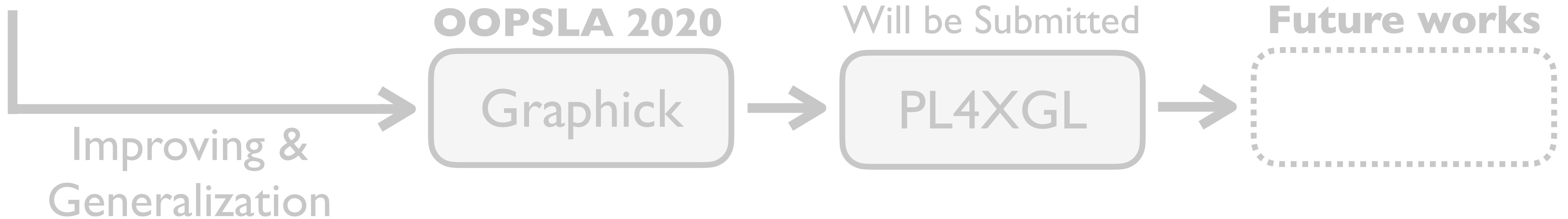
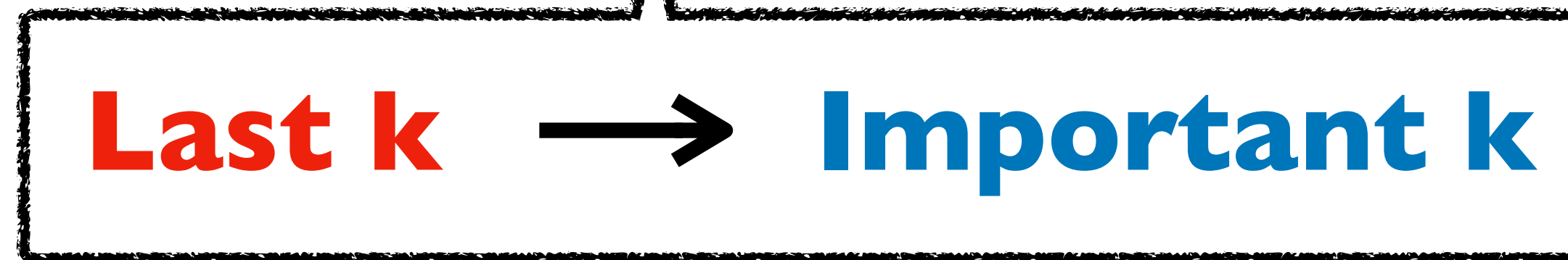
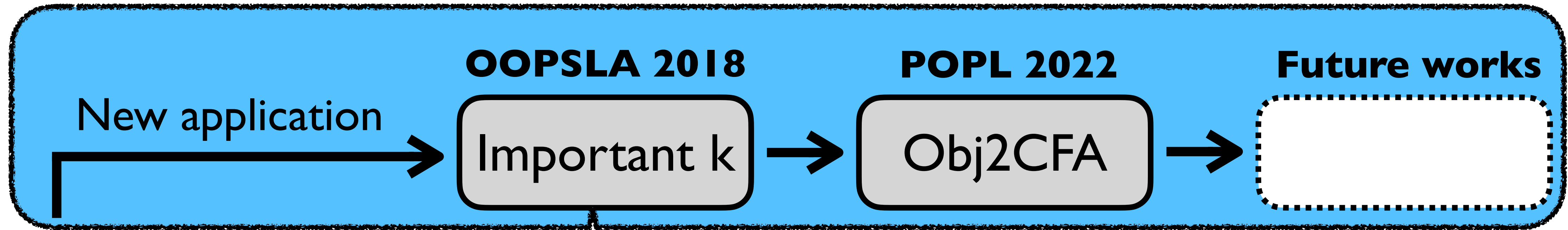
“...it covers more than two-thirds of the precision advantage of 2objH”
-Smaragdakis et al. [PLDI' 14]

“... 98.8% of the precision of 2obj can be preserved...”
-Li et al. [OOPSLA' 18]

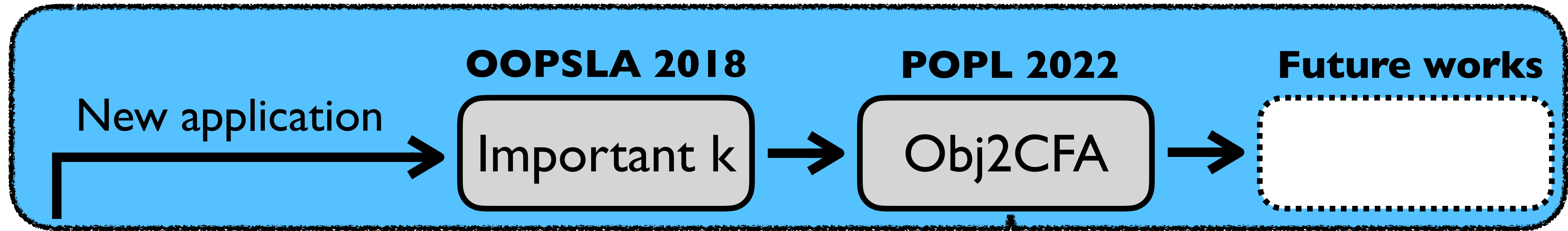
“Scaler still attains most of the precision gains of 2obj ...”
-Li et al. [FSE' 18]



Establishing **important k** as a standard

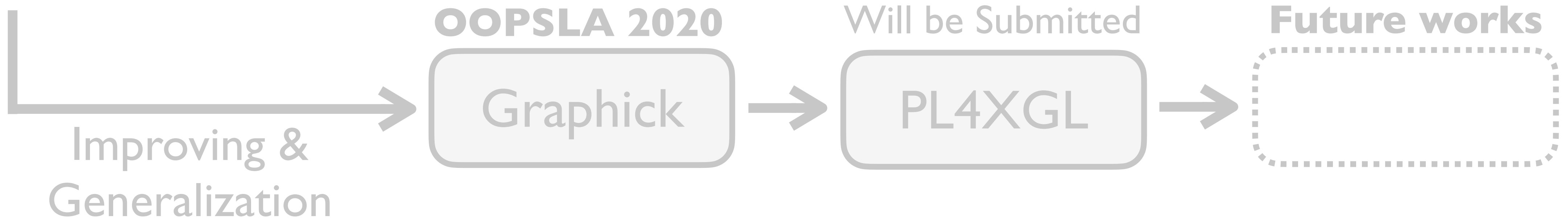


Establishing **important k** as a standard



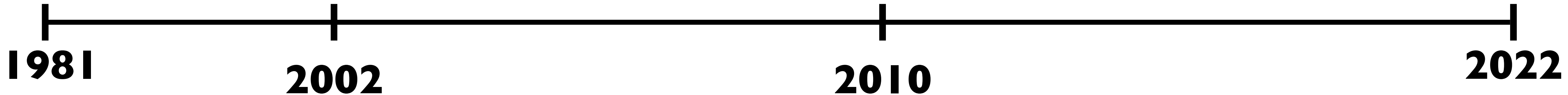
Challenged to a **commonly accepted knowledge**

OOPSLA 2017
Disjunctive model &
Learning algorithm



Call-site Sensitivity vs Object Sensitivity

Two major camps in OOP program analysis

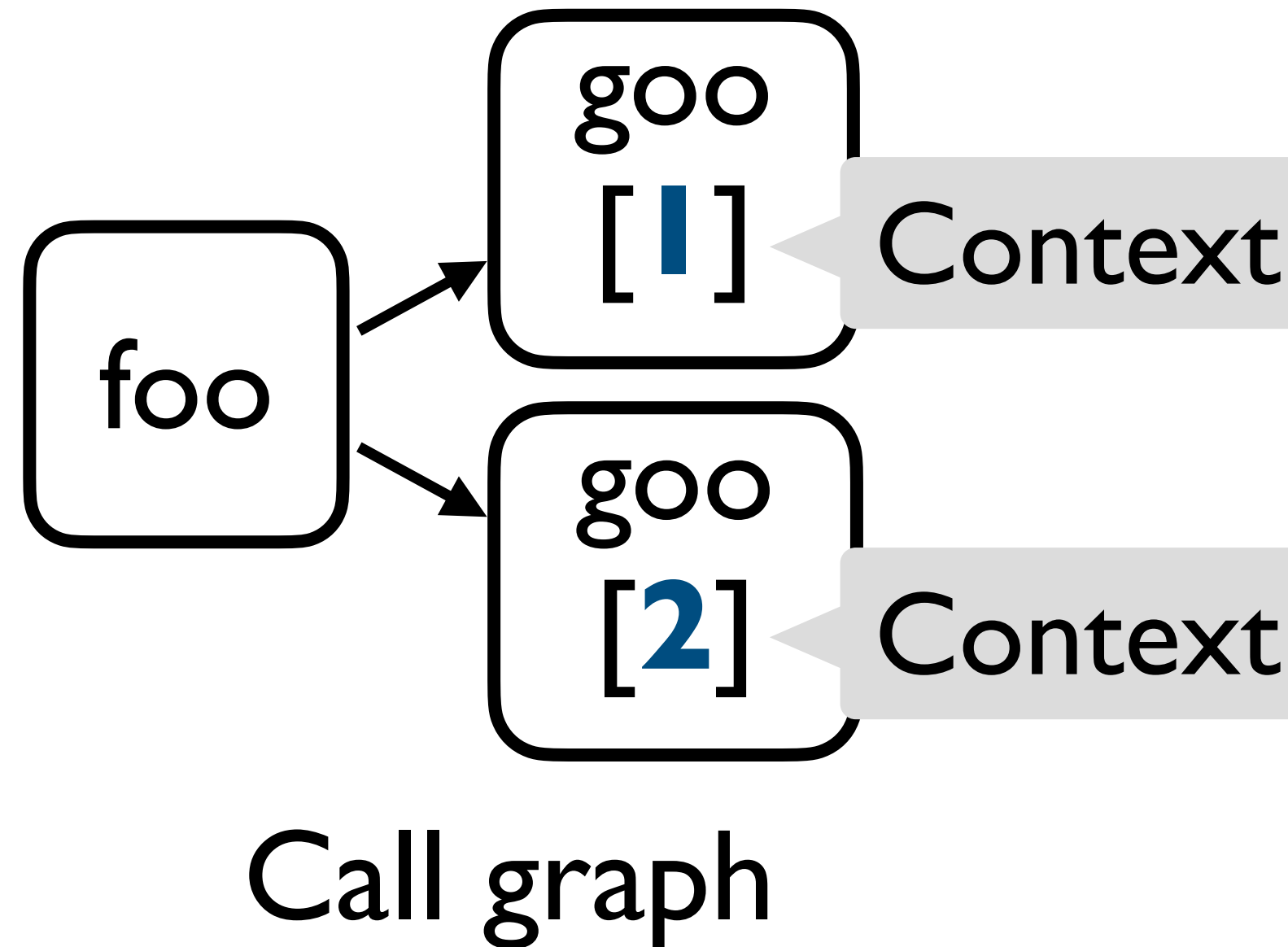


Call-site Sensitivity vs Object Sensitivity

Call-site sensitivity was born in 1981

- Considers “**Where**”

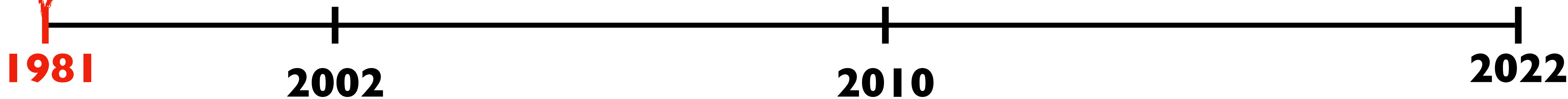
```
0: foo(){
1:   goo();
2:   goo();
3: }
```



Where is it called from?



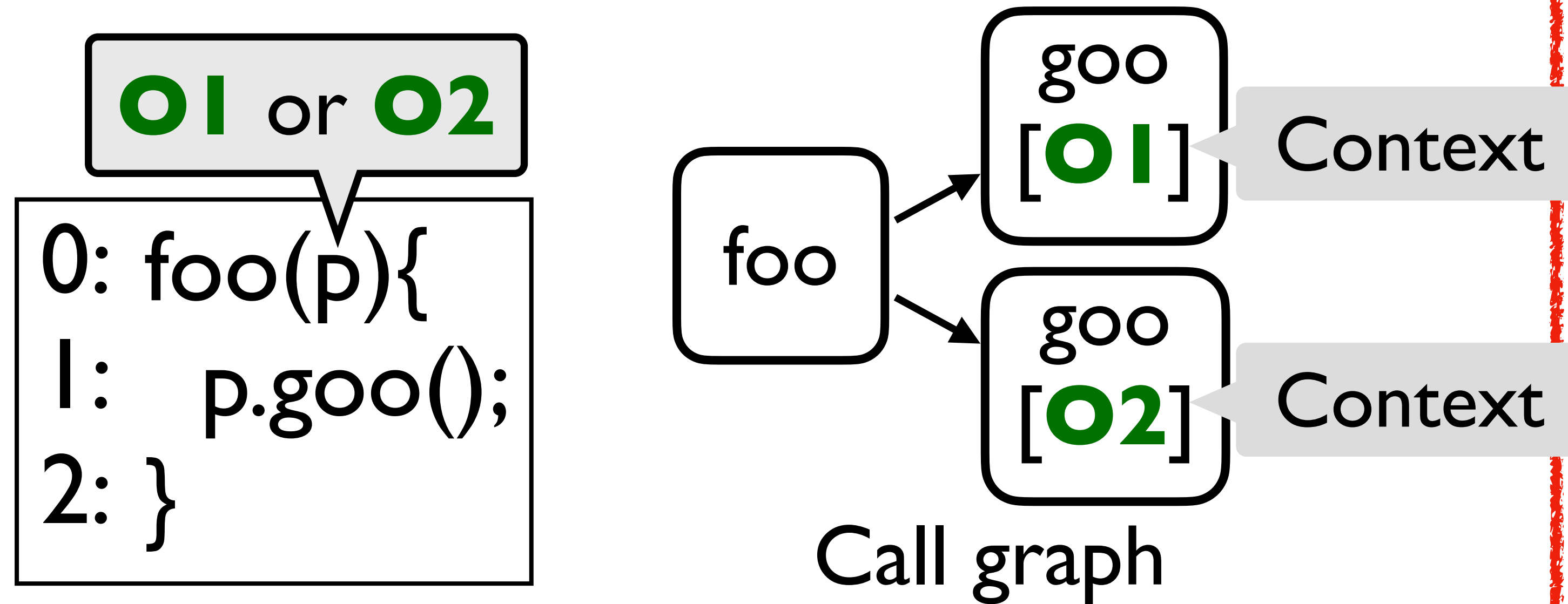
Call-site sensitivity



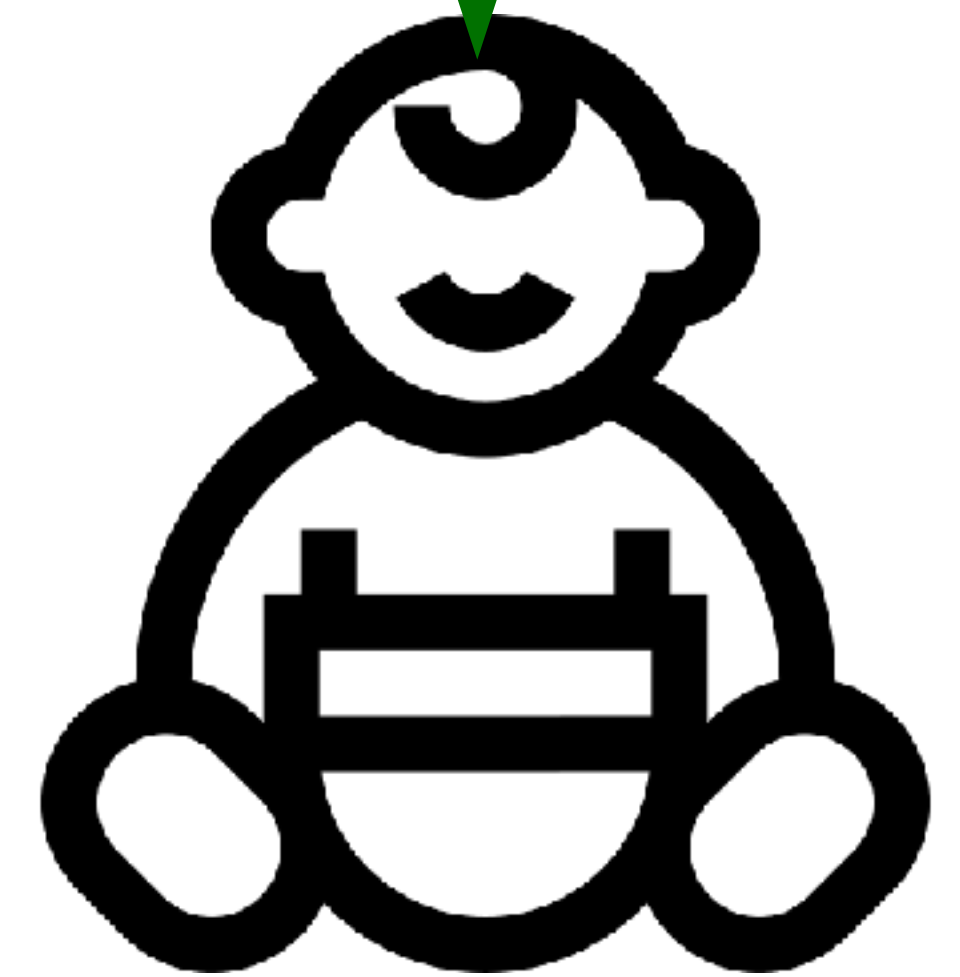
Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

- Considers “**What**”



What is it called with?



Object sensitivity



Call-site Sensitivity vs Object Sensitivity

Parameterized Object Sensitivity for Points-to Analysis for Java

ANG MILANOVA
Researcher, Polytechnic Institute
ATANAS BOUNTEV
Oak State University

BARBARA G. RYDER
Rutgers University

The goal of points-to analysis is to determine the set of objects pointed to by a reference variable in a reference type field. We present a novel sensitivity analysis for points-to analysis for Java. The key idea of our approach is to analyze a method as a whole. It means that the analysis is performed on the whole method body, which allows us to take into account the context of the method when analyzing the points-to analysis.

Our implementation is based on the use of a novel sensitivity analysis for points-to analysis. Our results show that object sensitivity significantly improves the precision of the analysis and is a good alternative to the use of a context-sensitive analysis. Our implementation is based on the use of a novel sensitivity analysis for points-to analysis. Our results show that object sensitivity significantly improves the precision of the analysis and is a good alternative to the use of a context-sensitive analysis.

Obj wins

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

Guang Li, and Laurie Hendren
School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Abstract. We present the results of an empirical study evaluating the precision of context-sensitive points-to analysis for Java. We compare the use of call-site sensitivity, the context-sensitive object sensitivity, and the BDD-based context-sensitive algorithm proposed by Zhu and Chaitin, and by Wang and Lam. Our study includes analysis that context-sensitive sensitivity can provide, as well as one that does not. We measure both characteristics of the points-to sets themselves, as well as effects on the precision of other analyses. To guide development of efficient analysis implementations, we measure the number of nodes, the number of context-sensitive points-to sets, and the number of context-sensitive points-to sets. To evaluate precision, we measure the number of context-sensitive points-to sets that are not reached by the analysis. Our results show that context-sensitive points-to analysis is worth the cost of context-sensitive points-to analysis.

1 Introduction

Context-sensitive points-to analysis is a well-studied problem in the analysis of object-oriented programs. It is often assumed that it is not, but lack of available implementations has prevented thorough empirical verification of this intuition. Of the many context-sensitive points-to analyses that have been proposed (e.g., [1, 4, 8, 11, 17, 19, 28, 31]), which improve precision on the one hand, but are more expensive to compute, it is not clear for which applications they are likely to be most useful. In this paper, we present the results of an empirical study of the precision of several implementations of context-sensitive points-to analysis. Our study aims to provide the answers. Recent advances in the use of Binary Decision Diagrams (BDDs) in program analysis [5, 12, 25, 31] have made context-sensitive analysis efficient enough to permit empirical study of bounded-size (significant) code. Using the Jazelle system [14], we have implemented three different variants of context-sensitive points-to analysis, and we compare their precision and performance.

Obj wins

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

ONDREJ LHOTÁK
University of Waterloo

and
LAURIE HENDREN
McGill University

We present Points, a framework of BDD-based context-sensitive points-to analysis for Java, as well as an efficient analysis that uses this analysis. Points supports several variants of context-sensitive points-to analysis, including call-site sensitivity, and context-sensitive sensitivity. We compare the use of call-site sensitivity, the context-sensitive object sensitivity, and the BDD-based context-sensitive algorithm proposed by Zhu and Chaitin, and by Wang and Lam. Our study includes analysis that context-sensitive sensitivity can provide, as well as one that does not. We measure both characteristics of the points-to sets themselves, as well as effects on the precision of other analyses. To guide development of efficient analysis implementations, we measure the number of nodes, the number of context-sensitive points-to sets, and the number of context-sensitive points-to sets. To evaluate precision, we measure the number of context-sensitive points-to sets that are not reached by the analysis. Our results show that context-sensitive points-to analysis is worth the cost of context-sensitive points-to analysis.

Obj wins

Strictly Declarative Specification of Sophisticated Points-to Analyses

Markus Brunnbauer, Yannis Smaragdakis
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003, USA
markus.brunnbauer@cs.umass.edu yannis@cs.umass.edu

Abstract

We present the Doo framework for defining points-to analyses for Java. Doo is a declarative language for defining points-to analyses. We compare the use of call-site sensitivity, the context-sensitive object sensitivity, and the BDD-based context-sensitive algorithm proposed by Zhu and Chaitin, and by Wang and Lam. Our study includes analysis that context-sensitive sensitivity can provide, as well as one that does not. We measure both characteristics of the points-to sets themselves, as well as effects on the precision of other analyses. To guide development of efficient analysis implementations, we measure the number of nodes, the number of context-sensitive points-to sets, and the number of context-sensitive points-to sets. To evaluate precision, we measure the number of context-sensitive points-to sets that are not reached by the analysis. Our results show that context-sensitive points-to analysis is worth the cost of context-sensitive points-to analysis.

Obj wins



Obj



CFA



Lectures have taught **superiority** of **object sensitivity**

Object-Sensitivity

- The dominant flavor of context-sensitivity for object languages.
- It uses object abstractions (i.e. allocation sites) as qualifying a method's local variables with the allocation site of the receiver object of the method call.

```
class A { void m() { return; } }
...
b = new B();
b.m();
```

The context of `m` is the allocation site of `b`.

Object-Sensitivity (vs. call-site sensitivity)

```
class S {
  Object id(Object a) { return a; }
  Object id2(Object a) { return id(a); }
}
class C extends S {
  void fun1() {
    Object a1 = new A1();
    Object b1 = id2(a1);
  }
}
class D extends S {
  void fun2() {
    Object a2 = new A2();
    Object b2 = id2(a2);
  }
}
```

Object-sensitive pointer

- Milanova, Rountev, and Ryder. *Parameterized sensitivity for points-to analysis for Java*. *ACM Eng. Methodol.*, 2005.
 - Context-sensitive interprocedural pointer analysis
 - For context, use stack of receiver objects
 - (More next week?)
- Lhotak and Hendren. *Context-sensitive pointer worth it?* *CC 06*
 - Object-sensitive pointer analysis more precise than for Java
 - Likely to scale better

Lecture Notes: Pointer Analysis

15-819C: Program Analysis
Jonathan Aldrich
jonathan.aldrich@cs.cmu.edu
Lecture 9

1 Motivation for Pointer Analysis

In programs with pointers, program analysis can become more complex. Consider constant-propagation analysis of the following program:

```
1: x = 1
2: p := &x
3: *p = 2
4: print x
```

In order to analyze this program correctly we must be able to determine what instruction `p` points to. If this information is available we can flow function as follows:

$$for[*p := y](e) = [x \rightarrow y](e) \text{ where } \text{must_point}(p)$$

When we know exactly what a variable `x` points to, we say it has *must-point-to* information, and we can perform a strong update of variable `x`, because we know with confidence that assigning to `x` is a technicality in the rule is quantifying over all `x` such that `x` points to `a`. How is this possible? It is not possible in C or Java, a language with pass-by-reference, for example C++, it is possible for the same location are in scope. Of course, it is also possible that we are uncertain to which distinct locations `p` points. For example:

Pointer Analysis

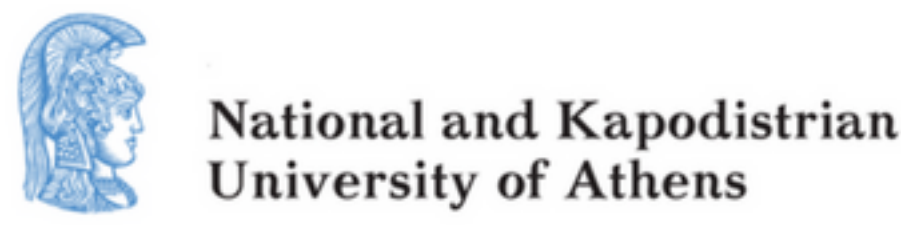
Yannis Smaragdakis
University of Athens
smaragd@cs.uoi.gr

George Balacouras
University of Athens
gbalacou@uoi.gr

now
the essence of knowledge
Boston — Ixth



Obj



Researches on Object Sensitivity



Obj

Pick Your Contexts Well: Understanding the Making of a Precise and Scalable Hybrid Context-Sensitive Pointer Analysis
Yannis Smaragdakis, Martin Bravenec

Making k -Object-Sensitive Pointer Analysis More Precise with Still k -Limiting
Tian Tan¹, Yue Li¹, and Jingling Xue^{1,2}

Precision-Guided Context Sensitivity for Pointer Analysis
YUE LI, Aarhus University, Denmark; TIAN TAN, Aarhus University, Denmark; ANDERS MÜLLER, Aarhus University; YANNIS SMARAGDAKIS, University of Athens, Greece

Data-Driven Context-Sensitivity for Points-to Analysis
SEHUN JEONG, Korea University, Republic of Korea; MINSEOK JEON, Korea University, Republic of Korea; CHA, Seoul National University; HAJONG CHOI, Seoul National University, Republic of Korea

Learning Graph-based Heuristics without Handcrafting Applications
MINSEOK JEON, MYUNGHO LEE, and HAJONG CHOI

Precision-Preserving Yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity
JINGBO LU, UNSW Sydney, Australia; JINGLING XUE, UNSW Sydney, Australia

Making Pointer Analysis More Precise by Unleashing the Power of Selective Context Sensitivity
TIAN TAN, Nanjing University, China; YUE LI, Nanjing University, China; XIANGXU MA, Nanjing University, China; CHANG XU, Nanjing University, China; YANNIS SMARAGDAKIS, University of Athens, Greece

1981

2002

2010

2022

Call-site Sensitivity has been ignored

“... call-site-sensitivity is **less important** than others ...”
- Jeon et al. [2019]



CFA

The collage features several research paper abstracts:

- A Machine-Learning Algorithm with Disjunctive M... Data-Driven Program Analysis**: Discusses a new machine-learning algorithm with disjunctive model for detecting pointer errors.
- Making k-Object-Sensitive Pointer Ana... More Precise with Still k-Limiting**: Abstracts object-sensitivity in regularity as equally the best abstraction for pointer analysis.
- Scalability-First: Pointer Ana... Self-Tuning Context-Sen...**: Abstracts a new machine-learning algorithm with disjunctive model for detecting pointer errors.
- Pick Your Contexts Well: Understanding Object-Sens... The Making of a Precise and Scalable Pointer Analysis**: Abstracts a new machine-learning algorithm with disjunctive model for detecting pointer errors.
- Hybrid Context-Sensitivity for Points-To A...**: Abstracts a new machine-learning algorithm with disjunctive model for detecting pointer errors.
- Precision-Guided Context Sensitivity for Point...**: Abstracts a new machine-learning algorithm with disjunctive model for detecting pointer errors.
- Introspective Analysis: Context-Sensitivity, Across the Board**: Abstracts a new machine-learning algorithm with disjunctive model for detecting pointer errors.

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

A Machine-Learning Approach to Data-Driven Program Analysis
MINSHUK LEE, SEJUN KUNG, RYUJIN CHOI

1 INTRODUCTION
The major challenge in static program analysis is a substantial amount of manual effort for tuning the analyzer performance for real-world applications. Practical advanced static analysis techniques to optimize their performance. For example, context-sensitive analysis for analyzing object-oriented programs, or in languages methods local variables, different calling contexts, however, applying context-sensitivity in all methods does not scale and therefore new static analysis apply context-sensitivity as methods determined by some heuristic rules [Jeon et al. 2019]. Another related analysis such as uses with Deacon [Luo 2008] because its heuristic of all variable relationships in the program, static analyzers employ variable-classification

Pointer analysis, as an enabling technology, plays a key role in a wide range of applications, including bug detection [2, 25, 26, 34], security analysis [15], compiler optimization [3, 32], and program understanding [12]. The main challenge in pointer analysis is flow-sensitivity and context-sensitivity. For object-oriented programs, e.g., Java programs, however, context-sensitivity is known to deliver tractable and useful precision [17, 19–21, 28–30]. There are two general approaches to achieving context-sensitivity in pointer analysis: call-site-sensitivity [6, 37] and object-use [24, 29] (among others). AB-CFA analysis represents a calling context as call by using a sequence of call sites (i.e., it labels with each call site). In contrast, a k-object-sensitive analysis uses k object abstracts (k labels with each denoting a new statement) as context elements.

3 EXPERIMENTATION
This paper presents a static analysis technique for pointer analysis. It is designed to be used as a component of a static analyzer. The main goal is to provide a practical and useful precision for pointer analysis. The main goal is to provide a practical and useful precision for pointer analysis. The main goal is to provide a practical and useful precision for pointer analysis.

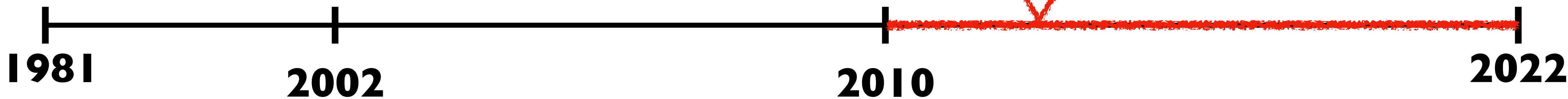
1.1 Introduction
The main goal of this paper is to provide a practical and useful precision for pointer analysis. The main goal is to provide a practical and useful precision for pointer analysis. The main goal is to provide a practical and useful precision for pointer analysis.

2.1 Related Work
This paper presents a static analysis technique for pointer analysis. It is designed to be used as a component of a static analyzer. The main goal is to provide a practical and useful precision for pointer analysis. The main goal is to provide a practical and useful precision for pointer analysis.

3.1 Experimental Setup
This paper presents a static analysis technique for pointer analysis. It is designed to be used as a component of a static analyzer. The main goal is to provide a practical and useful precision for pointer analysis. The main goal is to provide a practical and useful precision for pointer analysis.

4.1 Results
This paper presents a static analysis technique for pointer analysis. It is designed to be used as a component of a static analyzer. The main goal is to provide a practical and useful precision for pointer analysis. The main goal is to provide a practical and useful precision for pointer analysis.

5.1 Conclusion
This paper presents a static analysis technique for pointer analysis. It is designed to be used as a component of a static analyzer. The main goal is to provide a practical and useful precision for pointer analysis. The main goal is to provide a practical and useful precision for pointer analysis.



CFA

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

1981

2002

2010

2022

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 boils 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 call-site sensitivity or 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 creating context-sensitive analyses, and demonstrate that a remarkable increase in both precision and scalability can be gained by maintaining important context elements only. Our approach, called context tunneling, updates context 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 automatically 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 maintain 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-convex space of context tunneling.

We implemented our approach in the Loop 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, context-sensitive analysis with context tunneling, or called our deep 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), 30 pages. <https://doi.org/10.1145/3274811>

*Corresponding author.

Authors' addresses: Minseok Jeon, minseok_jeon@korea.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Seongbuk-gu, Seoul, 02841, Republic of Korea; Sehun Jeong, sehun@kyle.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Seongbuk-gu, Seoul, 02841, Republic of Korea; Hakjoo Oh, hakjoo_oh@korea.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Seongbuk-gu, Seoul, 02841, Republic of Korea.

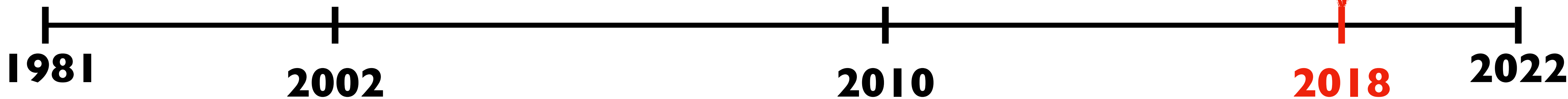
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Sharing with non-profit or academic organizations is permitted. To copy otherwise, or republish to print or otherwise, requires prior specific permission and/or a fee. Request permissions from permissions.acm.org.

© 2018 Association for Computing Machinery.
 0167-6369/2018/11-ART140
<https://doi.org/10.1145/3274811>

Jeon et al. [2018]

Paradigm shift

Last k → **Important k**



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.

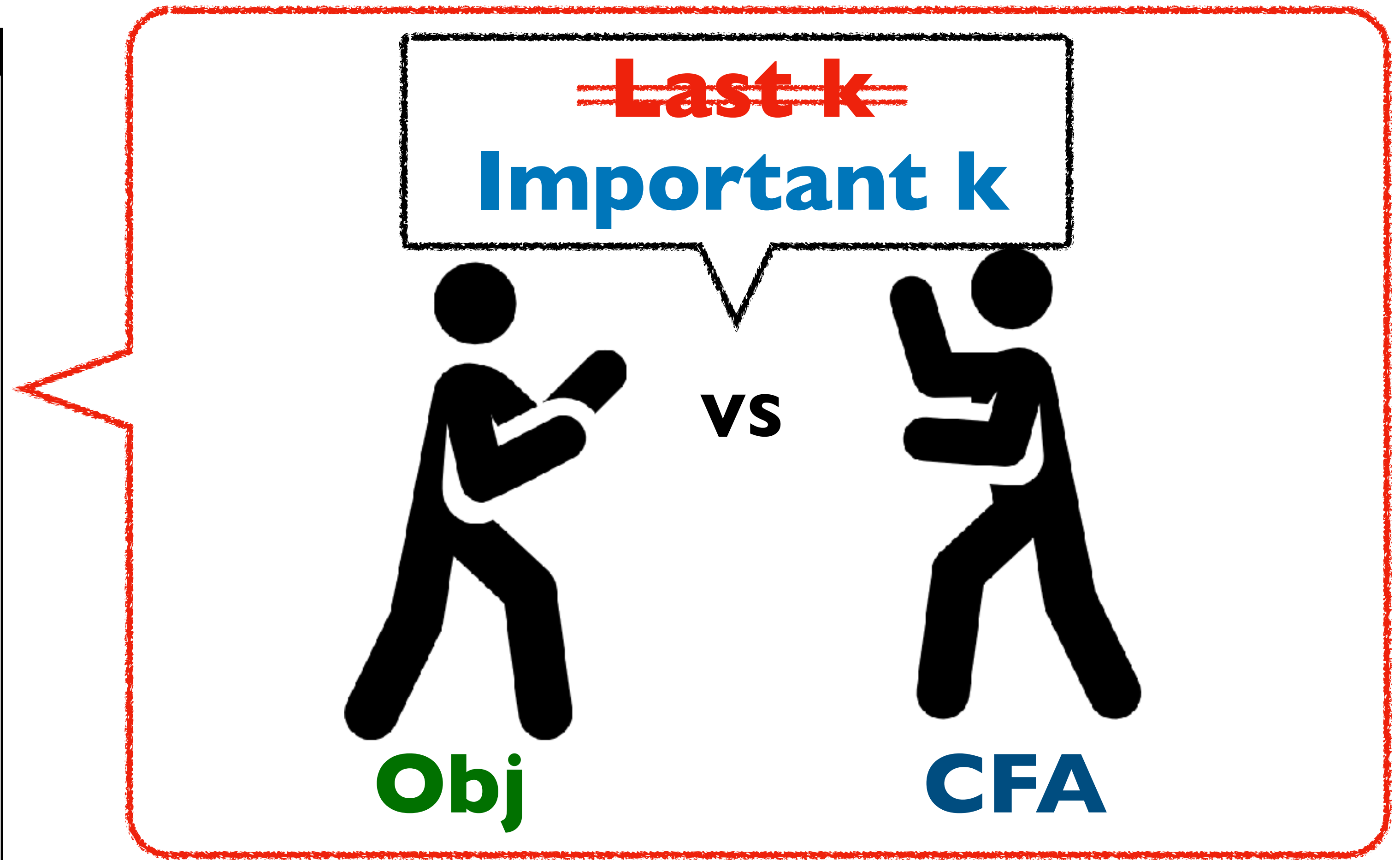
1 INTRODUCTION

"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



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 reason

CFA wins!

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



Obj



CFA

I return!

1981

2002

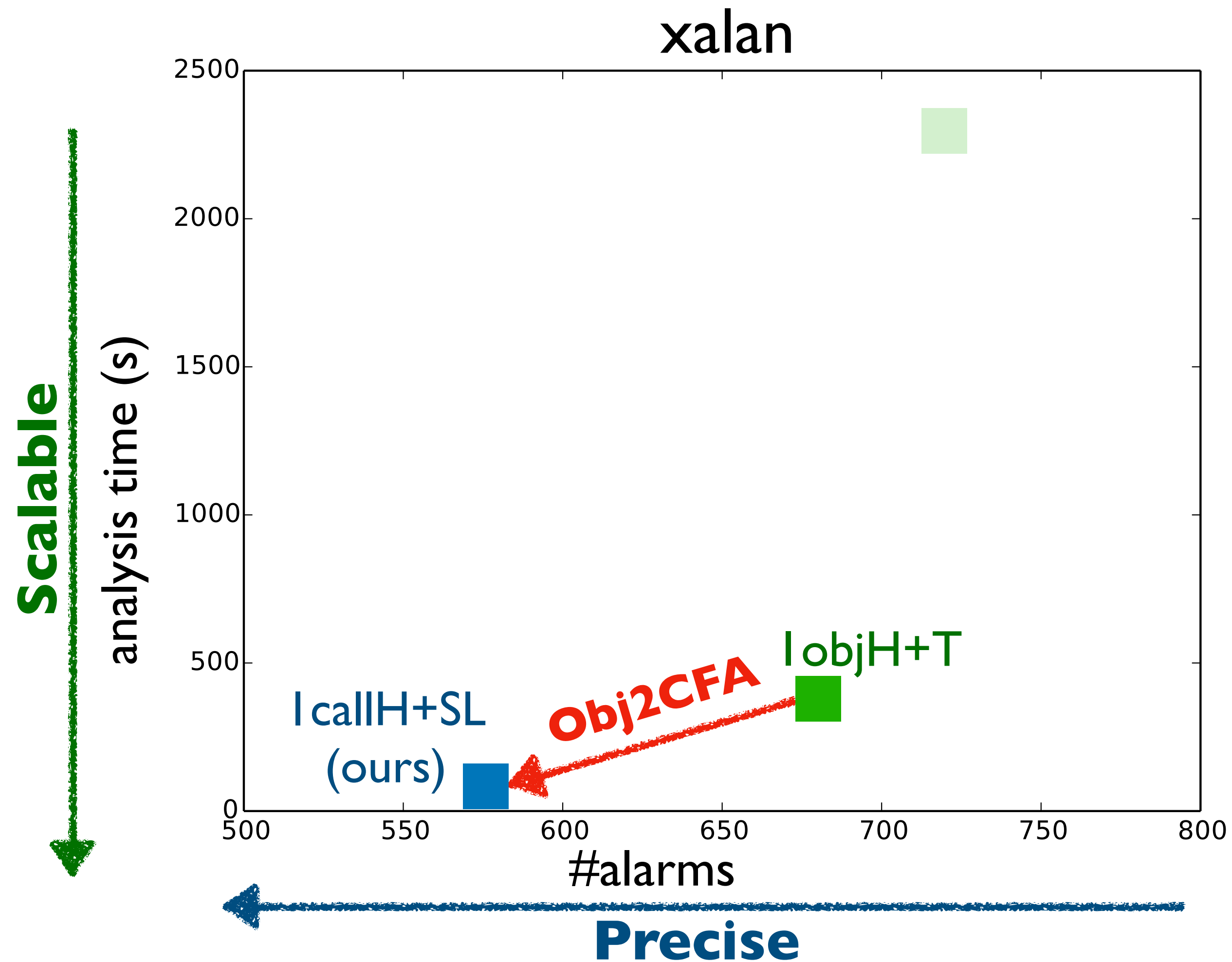
2010

2018

2022

Our Technique : **Obj2CFA**

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



Parametric pointer analyzer

Training data (programs)

Given object sensitivity

Obj+T

Predicates on call-sites

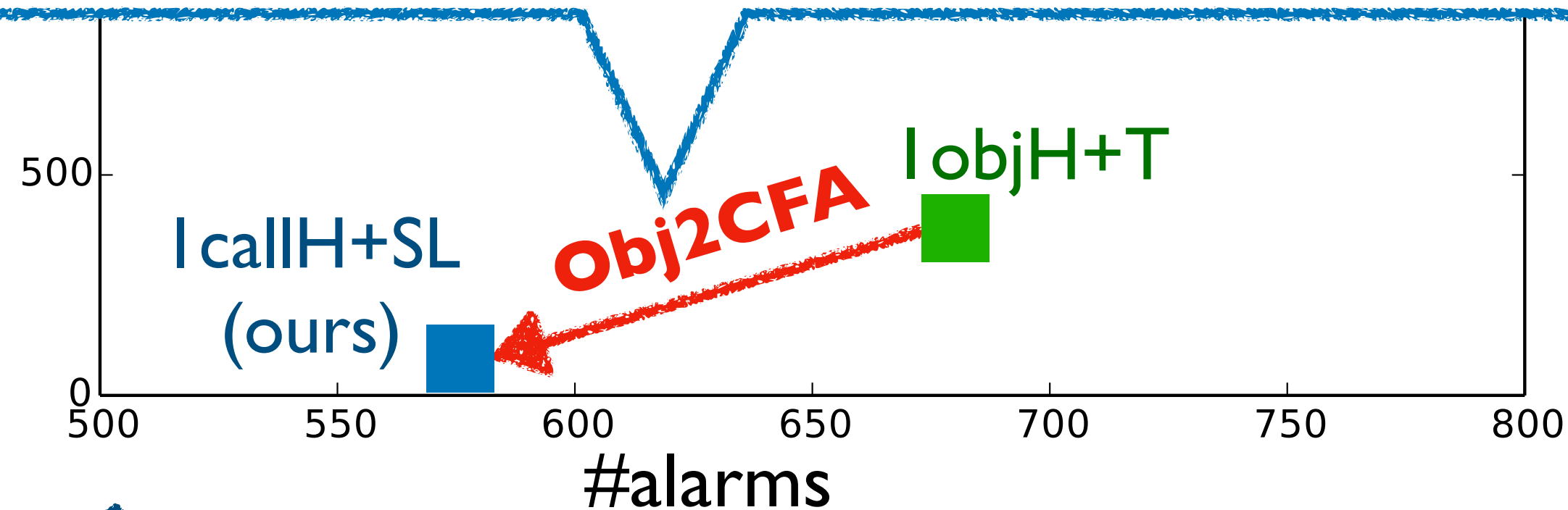
Atomic features (a_1, a_2, \dots, a_n)

Our framework

Learned unimportant call-sites for call-site sensitivity

$$f = (\neg a_6 \wedge a_8 \wedge \neg a_{11} \wedge \dots) \vee (a_1 \wedge a_2 \wedge \neg a_3 \wedge \dots) \vee \dots$$

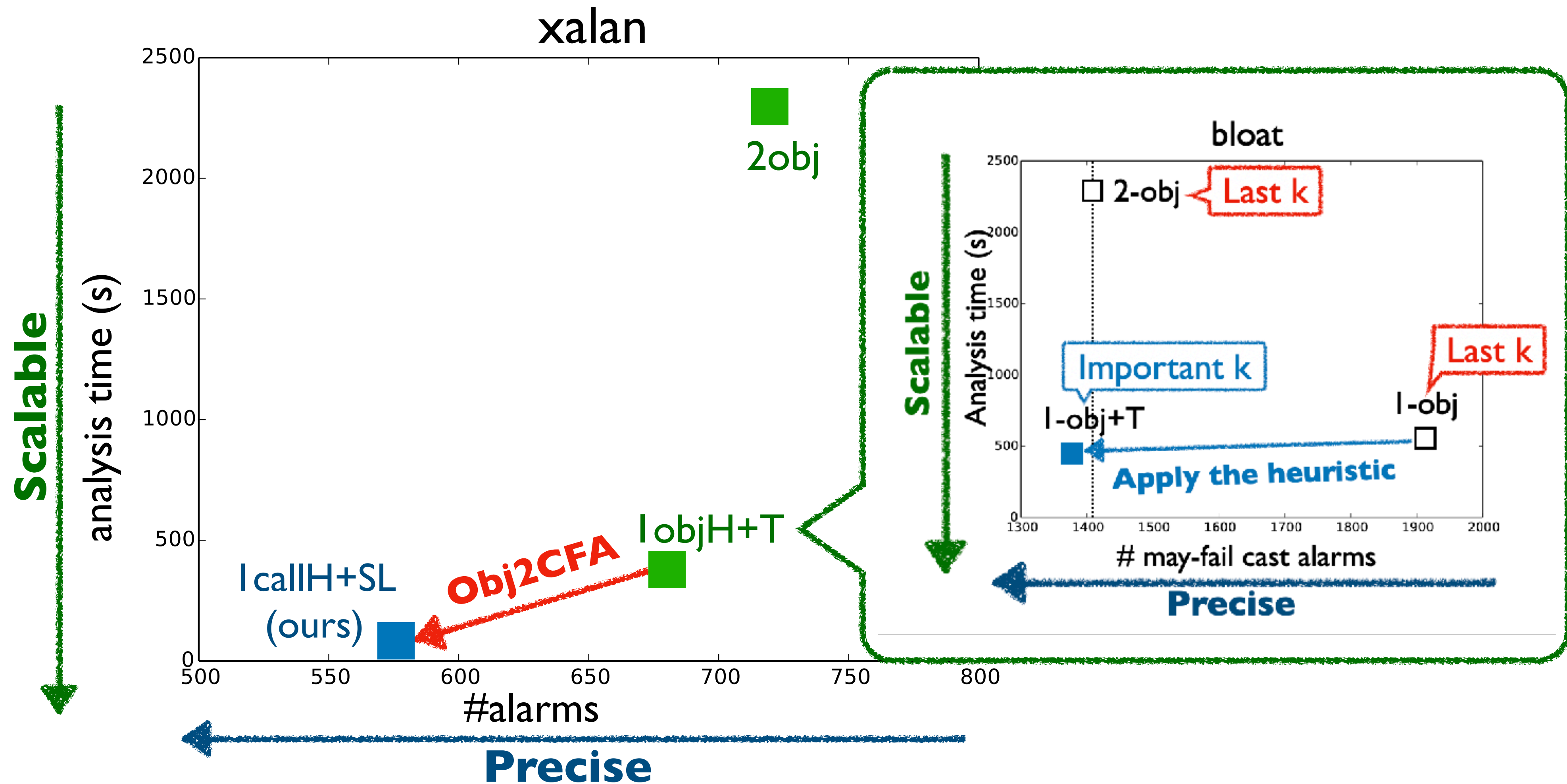
So analyze



Precise

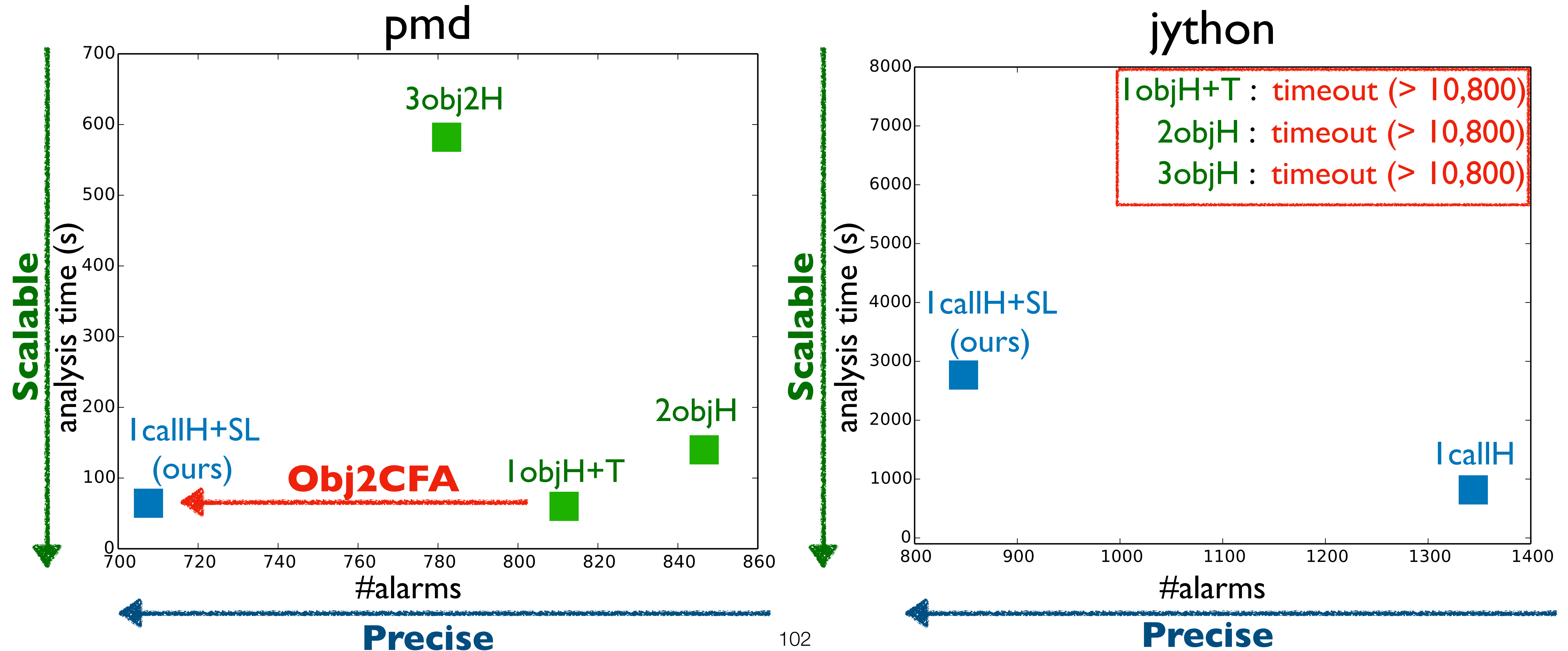
Our Technique : **Obj2CFA**

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



Call-site Sensitivity vs Object Sensitivity

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

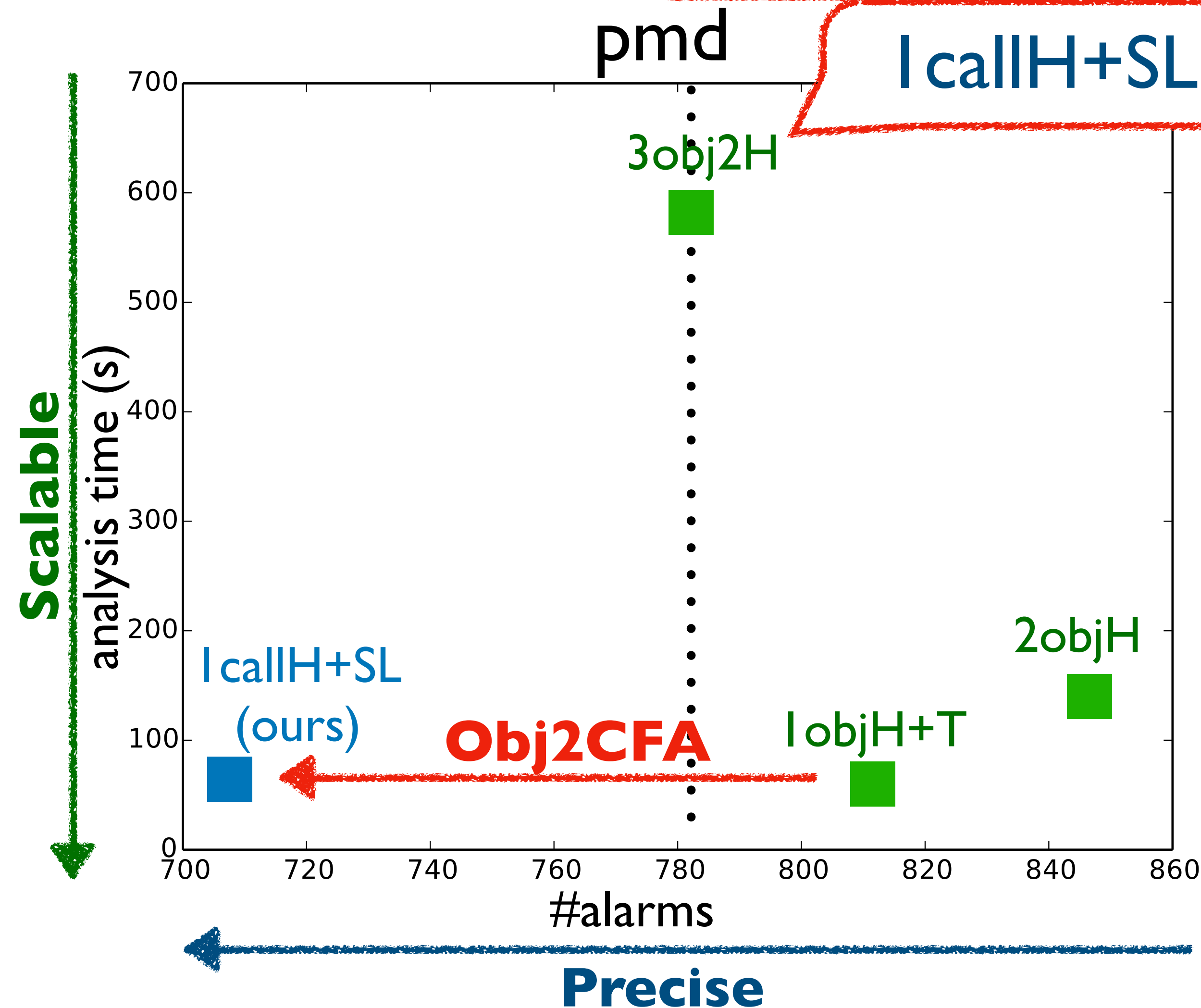


Call-site Sensitivity vs Object Sensitivity

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

IcallH+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

YUAN TAN, Nanjing University, China
YUE LI, Nanjing University, China
XIMING MA, Nanjing University, China
CHANG XU, Nanjing University, China
YANNIS SMARAGDIS, University of Arizona, Greece

Traditional context-sensitive pointer analysis is hard to scale for large and complex Java programs. To address this issue, a series of selective context sensitivity approaches have been proposed and exhibit promising results. In this work, we present one step further towards producing highly precise pointer analyses for large Java programs by proposing the **selective context sensitivity** approach, which fully exploits context sensitivity to scale and lower memory usage by using a novel **selective context sensitivity** approach, **scs**. **scs** is a novel approach that provides a **modular and unified** framework to combine and maximize the precision of all components of **scs**. When the **scs** framework is fully deployed, it offers a scheme for the **scs** to gain and accumulate the precision from each approach in **scs** to the next, leading to an analysis that is more precise than all approaches in **scs**.

As a proof-of-concept, we instantiate **scs** for Java with a tool called **scs** and experimentally evaluate it on a set of hard-to-analyze Java programs using general precision metrics and popular datasets. Compared with the state-of-the-art, **scs** achieves the best precision for all metrics and datasets for all evaluated programs. The difference in precision is often dramatic (up to 10% of all points reported by previously best algorithms) and is found to be significant and consistent.

CCS Concepts • Theory of computation → Program analysis

Additional Key Words and Phrases • Pointer Analysis, Alias Analysis, Context Sensitivity, Java

ACM Reference Format:
Tan, Yuan, Yue Li, Ximing Ma, Chang Xu, and Yannis Smaragdakis. 2021. Making Pointer Analysis More Precise by Unleashing the Power of Selective Context Sensitivity. Proc. ACM Program. Lang. 5, 000524, Article 47, October 2021, 47 pages. <https://doi.org/10.1145/3481514>

1 INTRODUCTION
Pointer analysis is important for a variety of real-world applications such as bug detection [Chandra et al. 2009; Naik et al. 2015], security analysis [Ali et al. 2004; Das et al. 2003], program verification [Ali et al. 2009; Trinder et al. 2010] and program understanding [Ali et al. 2010; Sridharan et al. 2010].

Corresponding author

Authors' address: Yuan Tan, State Key Laboratory for Novel Software Technology, Nanjing University, China, the email address is: tan@nju.edu.cn; Yue Li, State Key Laboratory for Novel Software Technology, Nanjing University, China, the email address is: liyue@nju.edu.cn; Ximing Ma, State Key Laboratory for Novel Software Technology, Nanjing University, China, the email address is: chongxu@nju.edu.cn; Chang Xu, State Key Laboratory for Novel Software Technology, Nanjing University, China, the email address is: changxu@nju.edu.cn; Yannis Smaragdakis, Department of Information and Knowledge Systems, University of Arizona, Greece, ysmaragd@arizona.edu

OOPSLA 2021

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

JINGQI LI, State University of New York at Binghamton, USA
JINGLING XUE, IBM Research, Australia

Object sensitivity is widely used as a context abstraction for computing the pointer information needed to analyze object-oriented languages like Java. Due to the undecidability of pointer analysis on heap programs, object-sensitive pointer analysis under a language domain is usually only for small subsets of it, which is of typically a low-level subset of objects to improve efficiency by limiting the analysis only to some methods in the program, some sensitivity, structured sensitivity by a program, or a set of objects. However, these limitations may cause the analysis to be imprecise, and consequently, are harmful to the efficiency of the analysis. In this paper, we propose a novel approach, **scs**, that scales to large programs by using a novel **selective context sensitivity** approach, **scs**. **scs** is a novel approach that provides a **modular and unified** framework to combine and maximize the precision of all components of **scs**. When the **scs** framework is fully deployed, it offers a scheme for the **scs** to gain and accumulate the precision from each approach in **scs** to the next, leading to an analysis that is more precise than all approaches in **scs**.

CCS Concepts • Theory of computation → Program analysis

Additional Key Words and Phrases • Pointer Analysis, Object Sensitivity, Context Sensitivity

ACM Reference Format:
Li, Jingqi, and Xue, Jingling. 2019. Precision-Preserving Yet Fast Object-Sensitive Pointer Analysis with Partial Context Sensitivity. Proc. ACM Program. Lang. 3, 000524, Article 148, October 2019, 19 pages. <https://doi.org/10.1145/3366274>

1 INTRODUCTION
For object-oriented languages such as Java, context-sensitivity is known as a highly useful technique for pointer analysis [Abdulla and Asadi, 2008; Smaragdakis et al. 2007]. A context-sensitive pointer analysis, such as Andersen's analysis [Andersen, 1994] analyzes a method only once, performing one pointer flow for each method. The **scs** approach for analyzing every object in the method. In contrast, to obtain a more accurate analysis, a method can be analyzed multiple times under different call-site contexts that abstract different runtime invocations. However, processing multiple points to see the every method with one context and multiple object-sensitivity for analyzing every object in the method is not practical in the real world.

To face the scalability problem of context-sensitivity, context-sensitivity is represented by a sequence of **scs** elements, under **scs**. There are two representative abstractions for object-sensitive programs: **Obj2CFA** [Abdulla et al. 2007], which distinguishes the contents of a method by its **scs** elements [Abdulla et al. 2007] and **scs** [Abdulla et al. 2007], which abstracts the contents of a method by its **scs** elements [Abdulla et al. 2007].

© 2021 ACM. This is a Creative Commons Attribution 4.0 International License. <https://creativecommons.org/licenses/by/4.0/>

OOPSLA 2019

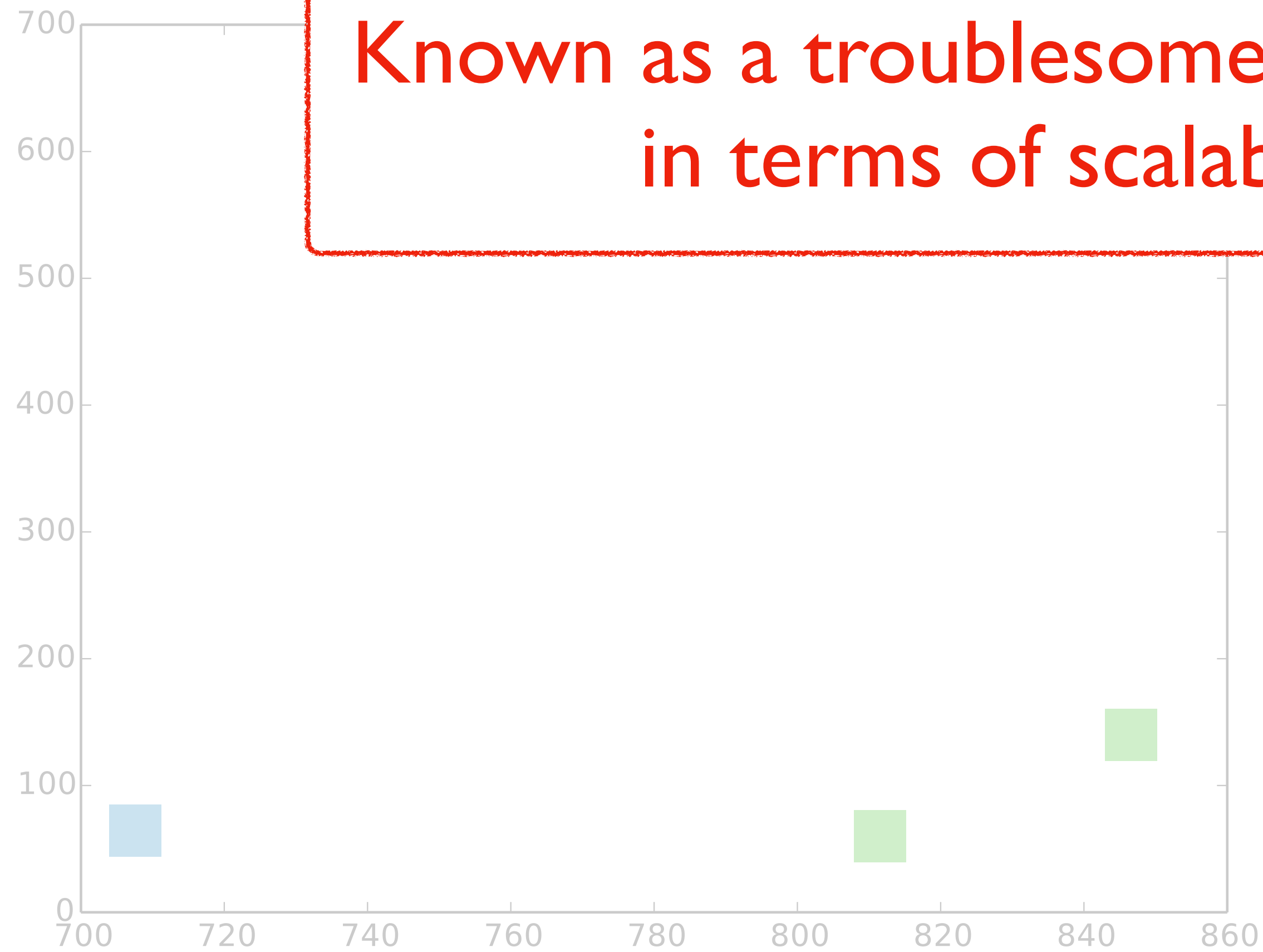
Call-site Sensitivity vs Object Sensitivity

- `lcallH+SL` (ours) is more precise and **scalable** than the existing object sensitivities

Known as a troublesome benchmark in terms of scalability

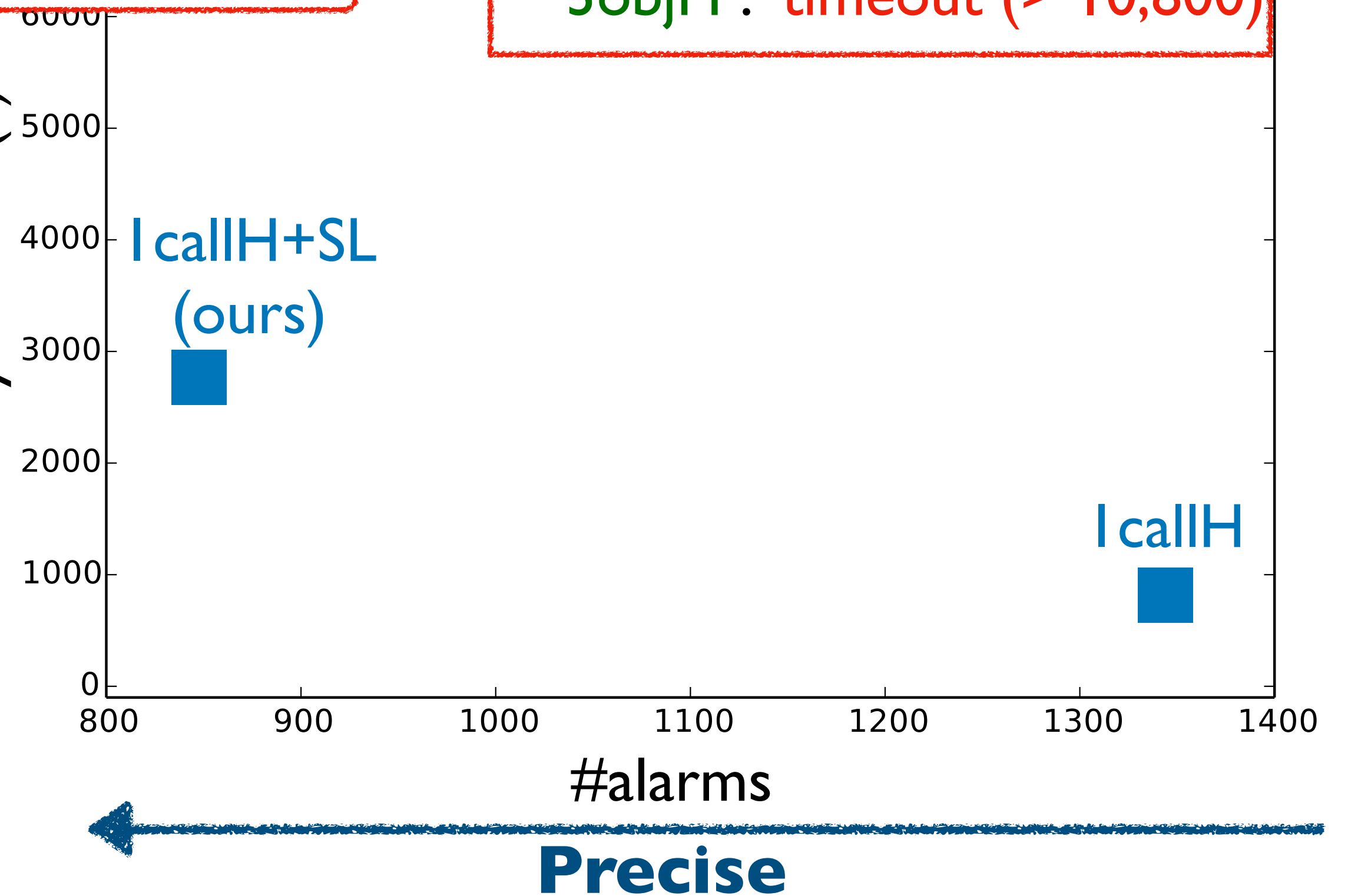
ython

`lobjH+T` : timeout (> 10,800)
`2objH` : timeout (> 10,800)
`3objH` : timeout (> 10,800)



Scalable

analysis time (s)



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 efficiency)

In general

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

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>

A lecture slide
used in 2023

100 1200 1300 1400

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 efficiency)

In general

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

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>

A lecture slide
used in 2023

100 1200 1300 1400

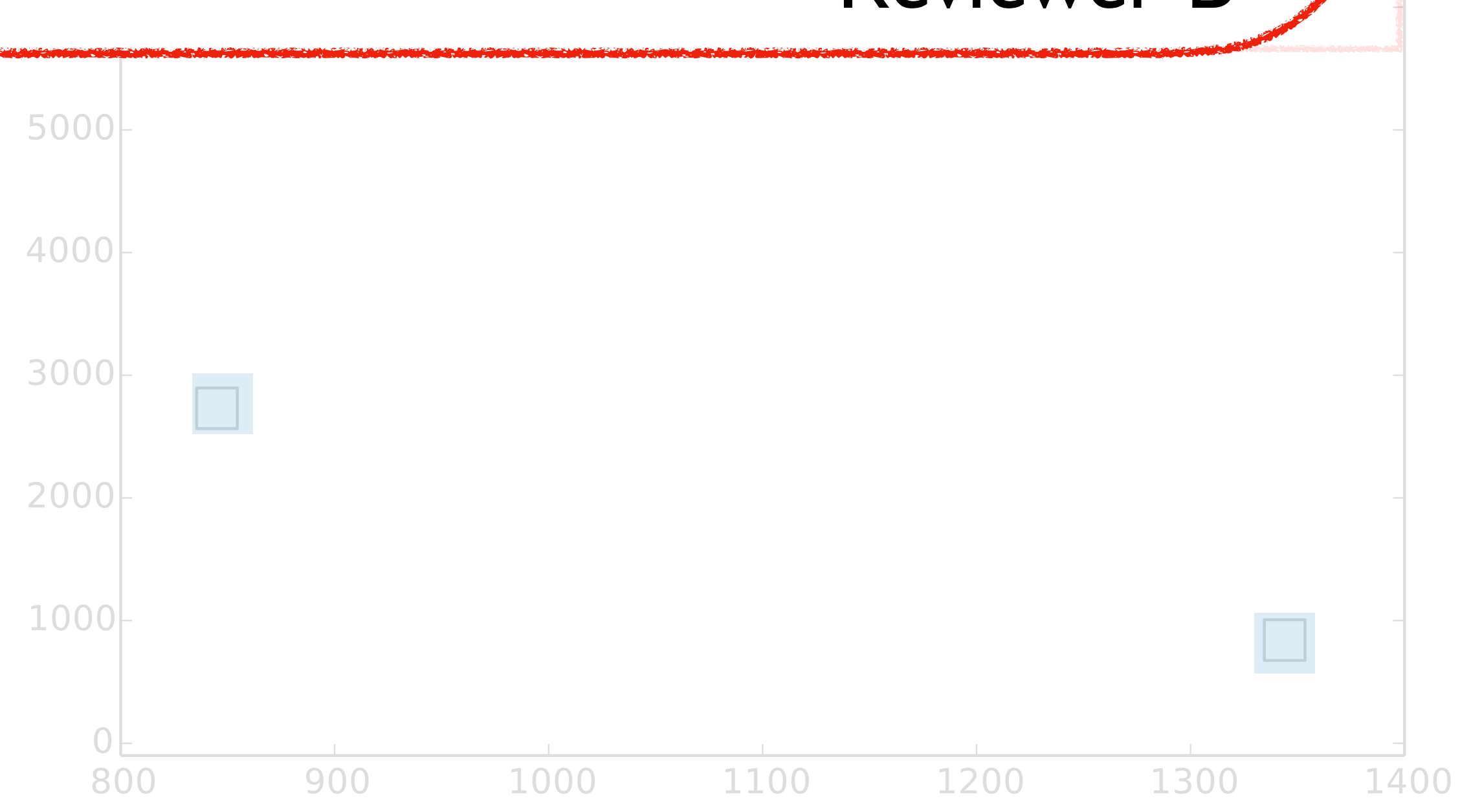
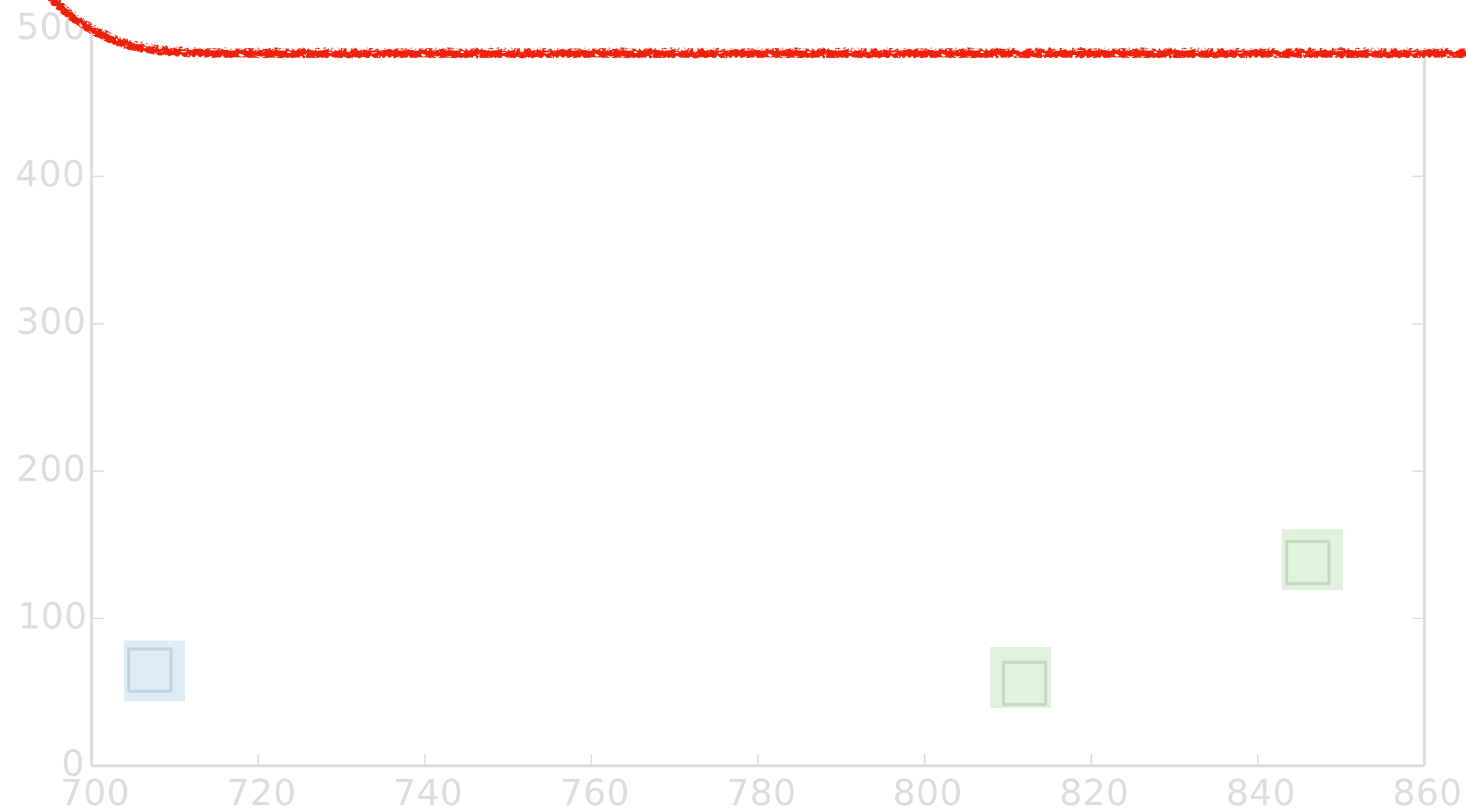
A Review Comment

“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.**”

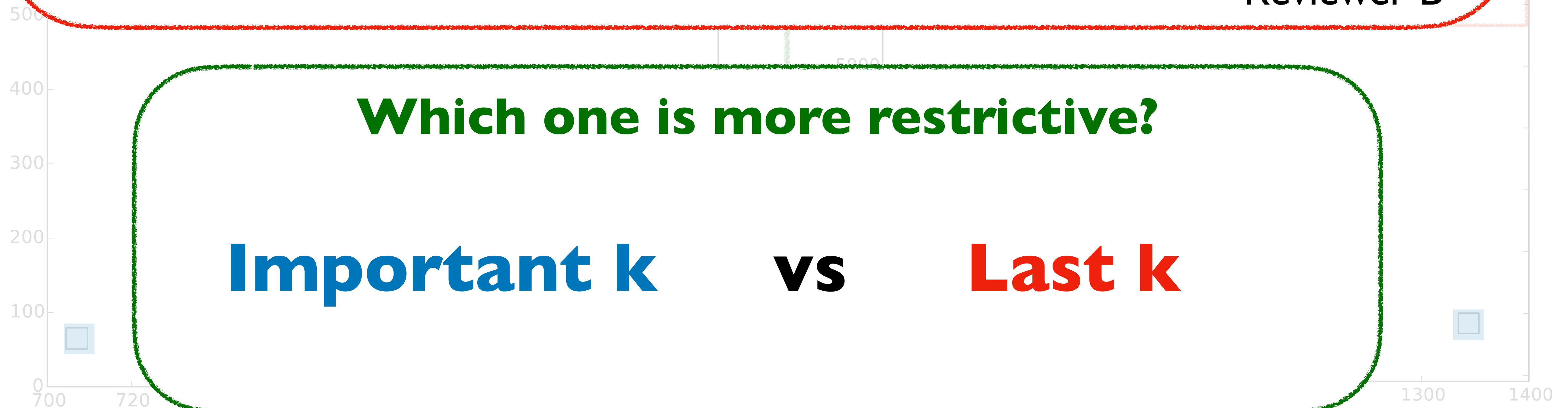
- Reviewer B

Which one is more restrictive?

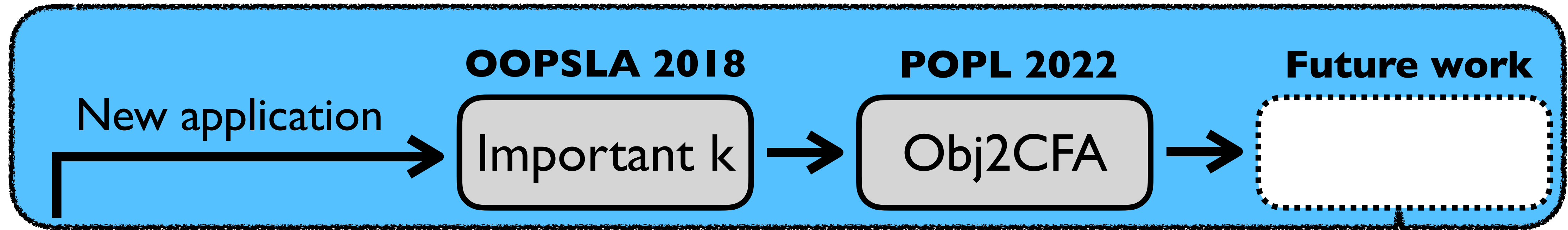
Important k

vs

Last k



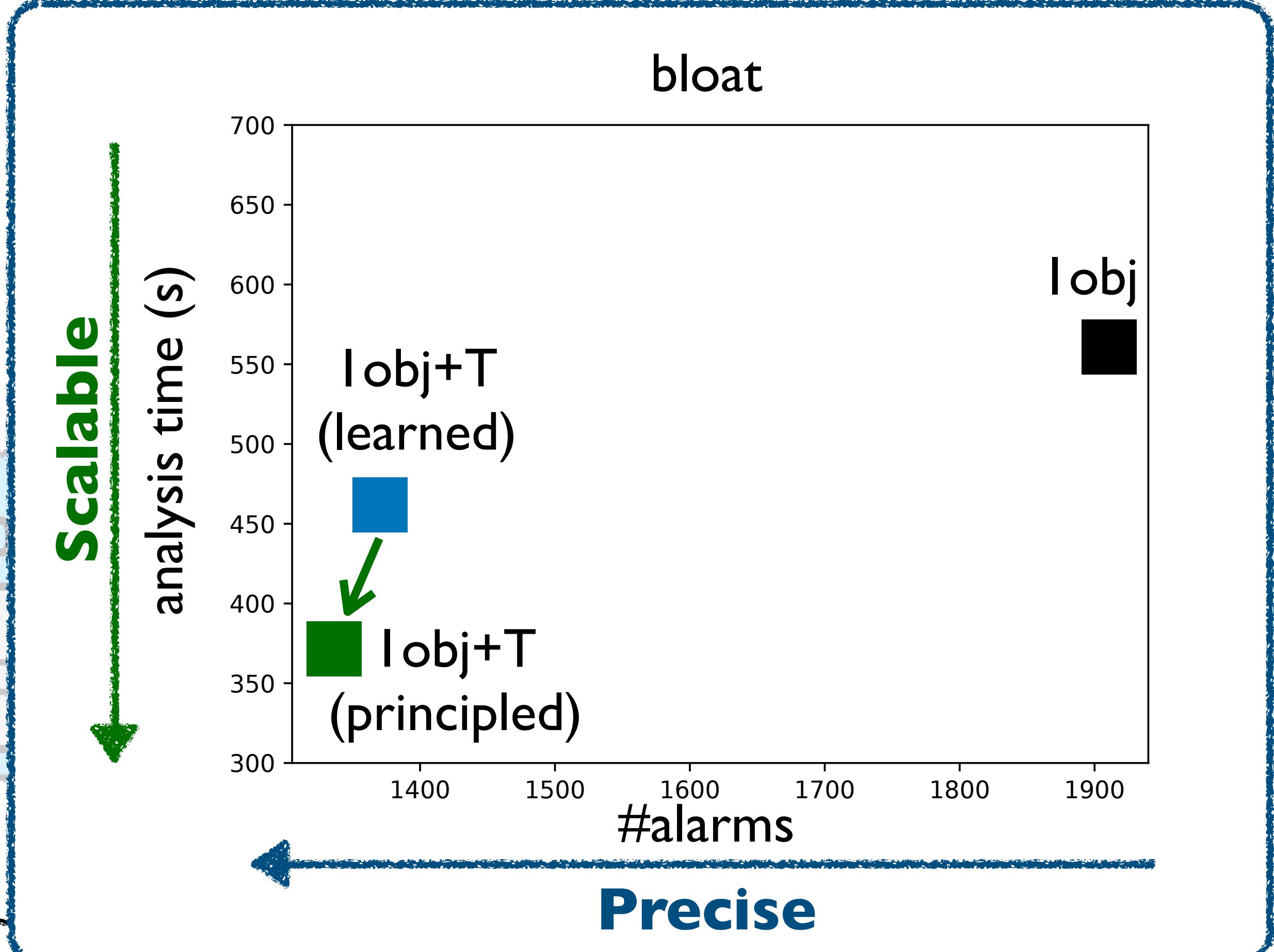
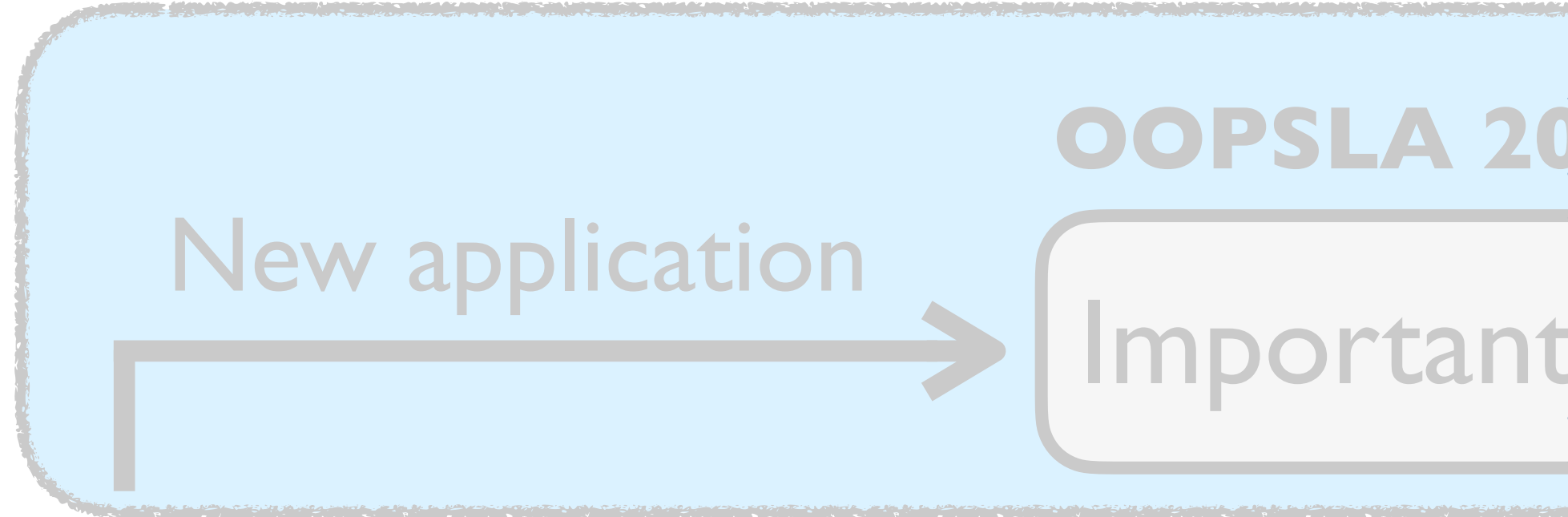
Establishing **important k** as a standard



OOPSLA 2017

Disjunct
Learnin

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



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

OOPSLA 2017

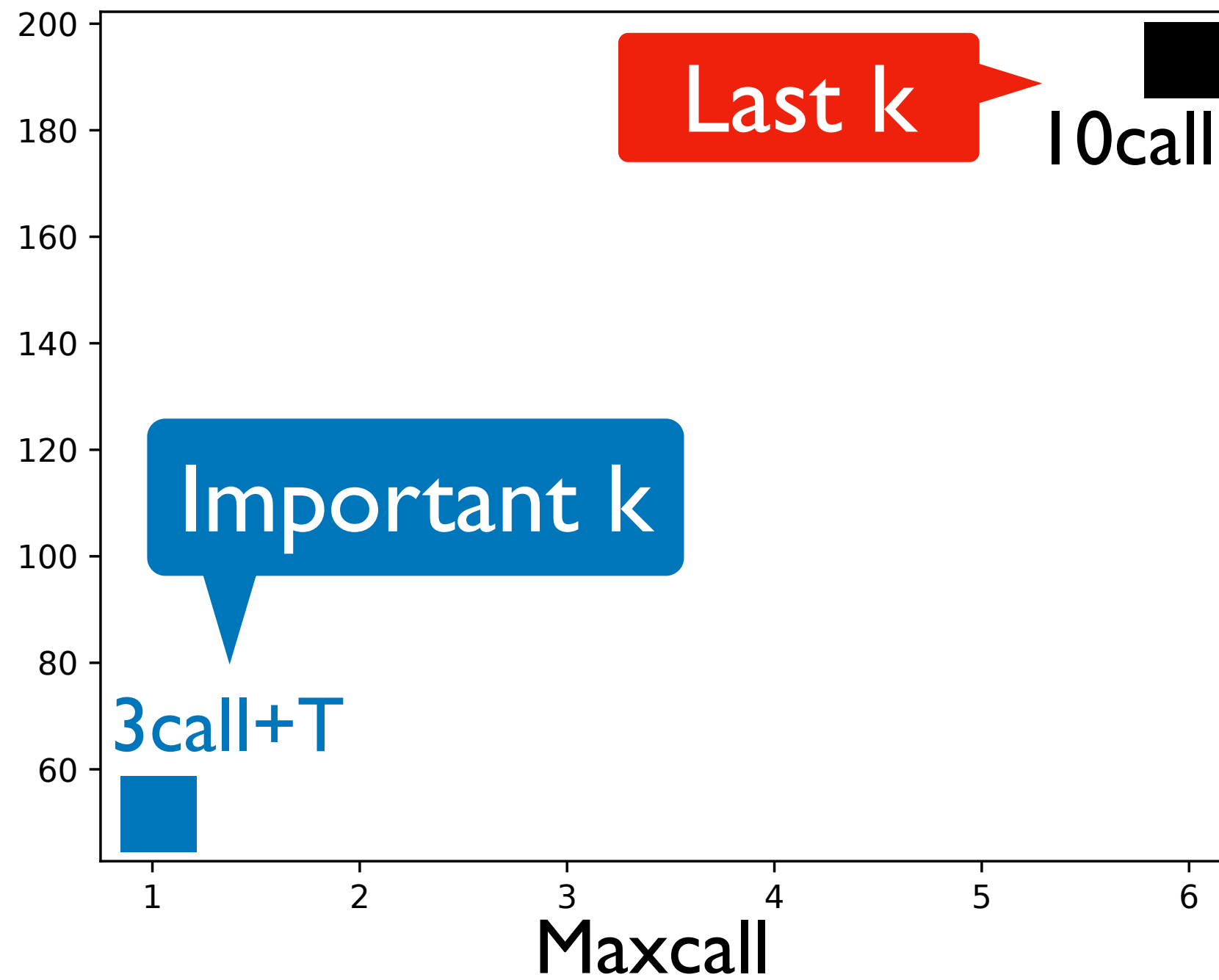
Disjunct
Learnin

New application

Scalable

Analysis time (s)

Bench 62



Future work

Language

Java

JavaScript

Python

C

...

Learning
heuristics

OOPSLA'18
POPL' 22

In progress

ToDo

ToDo

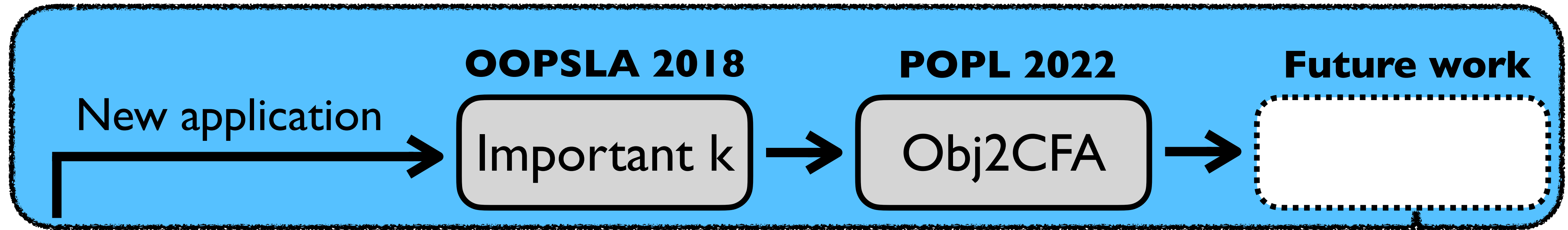
Principled
heuristics

ToDo

ToDo

ToDo

Establishing **important k** as a standard



OOPSLA 2017

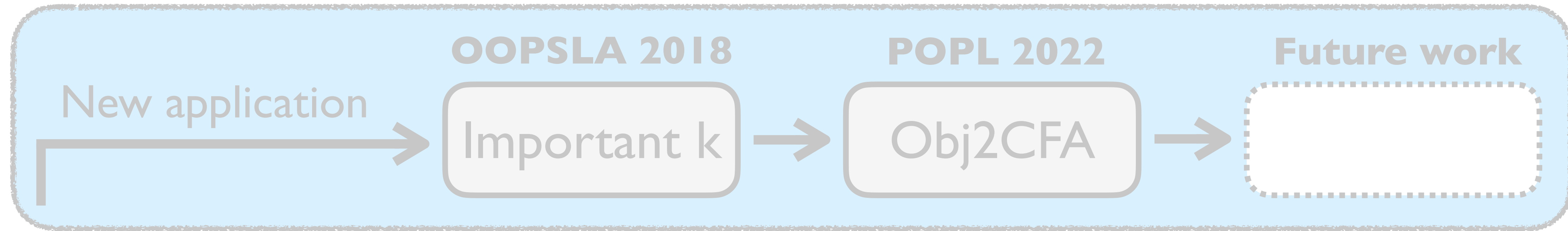
Disjunct
Learnin

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

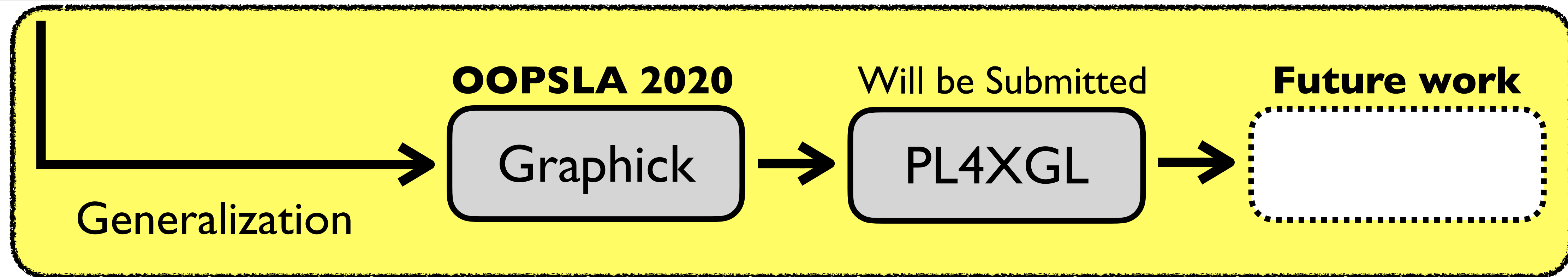


OOPSLA 2017

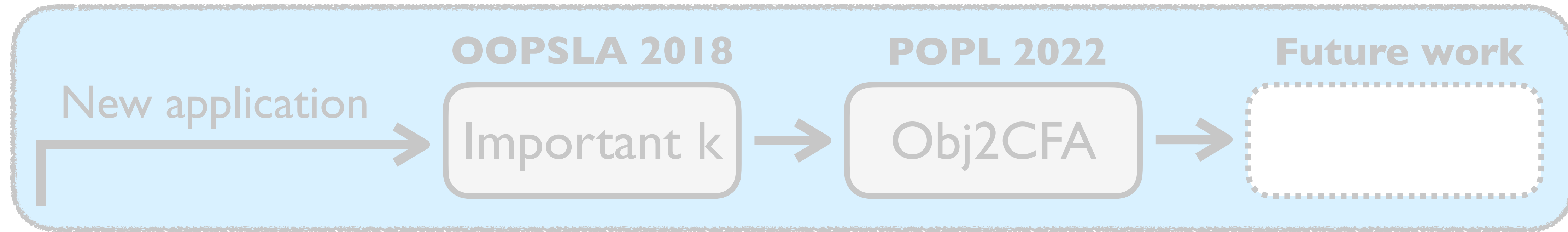
Disjunctive model &
Learning algorithm

PL-based

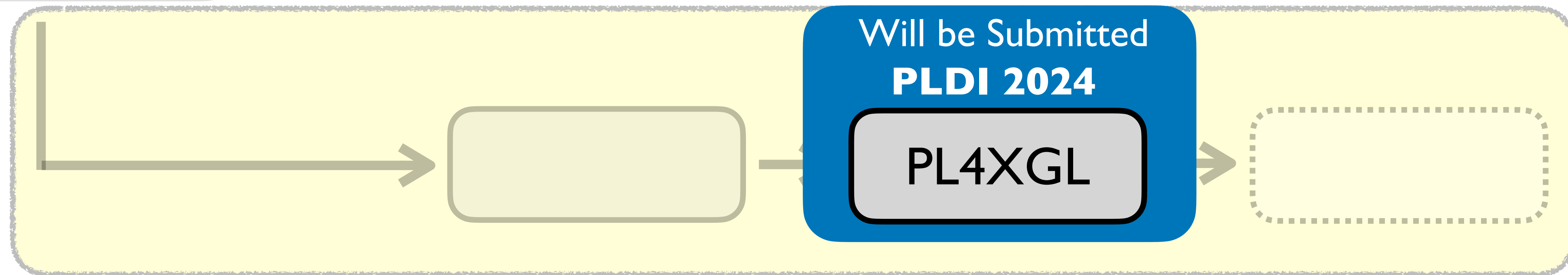
Establishing a new graph machine learning method



Establishing important k as a standard



PL-based Explainable Graph Machine Learning



- Existing: **unexplainable** AI (Graph Neural Network)

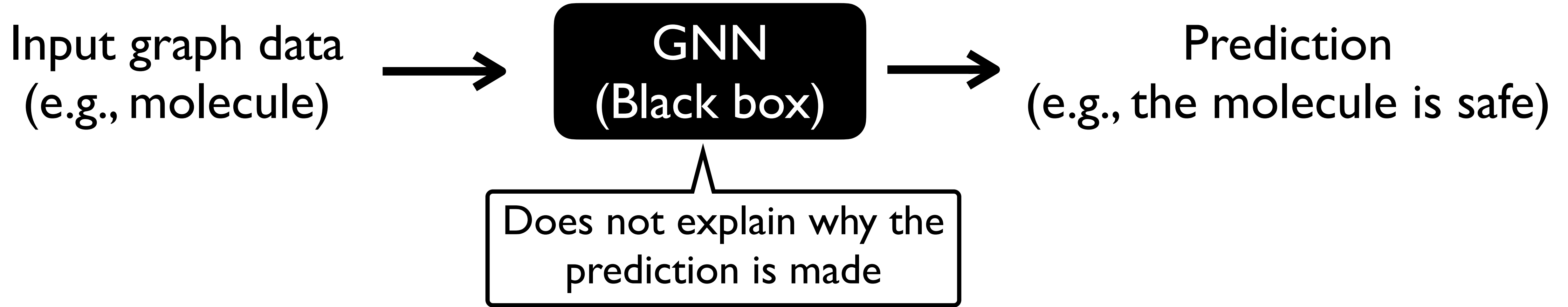
Input graph data
(e.g., molecule)



Prediction
(e.g., the molecule is safe)

- Social network
- Program representation
- ...

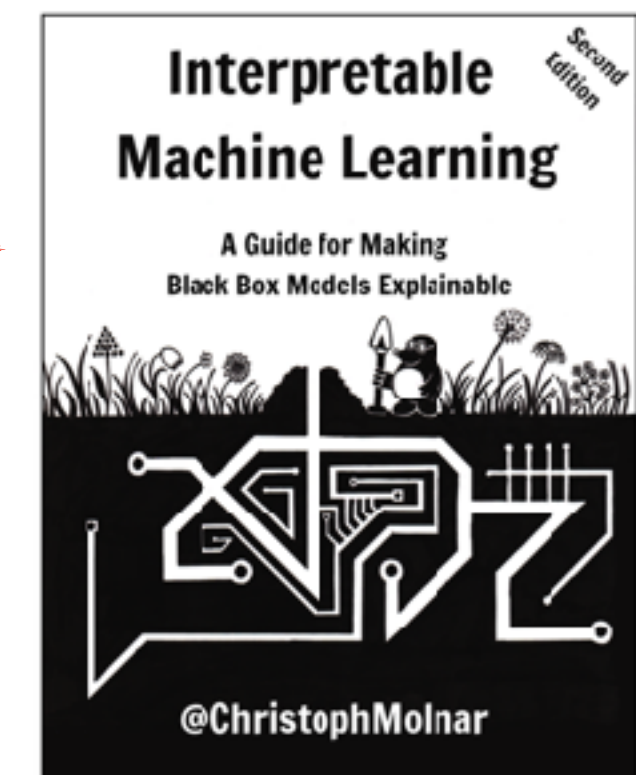
- Existing: **unexplainable** AI (Graph Neural Network)



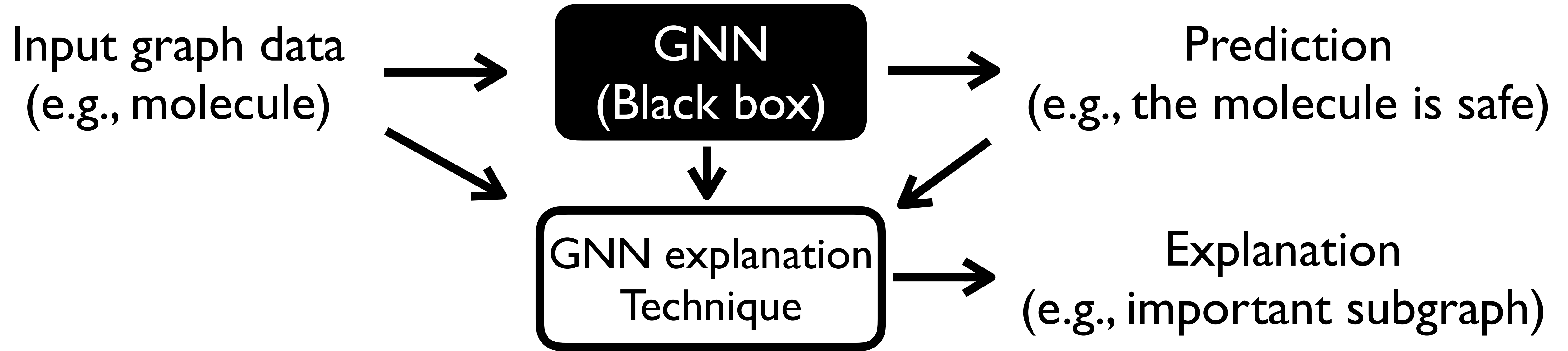
Value of explainability is growing fast

A correct prediction only partially solves your problem. The model must also explain **why**.

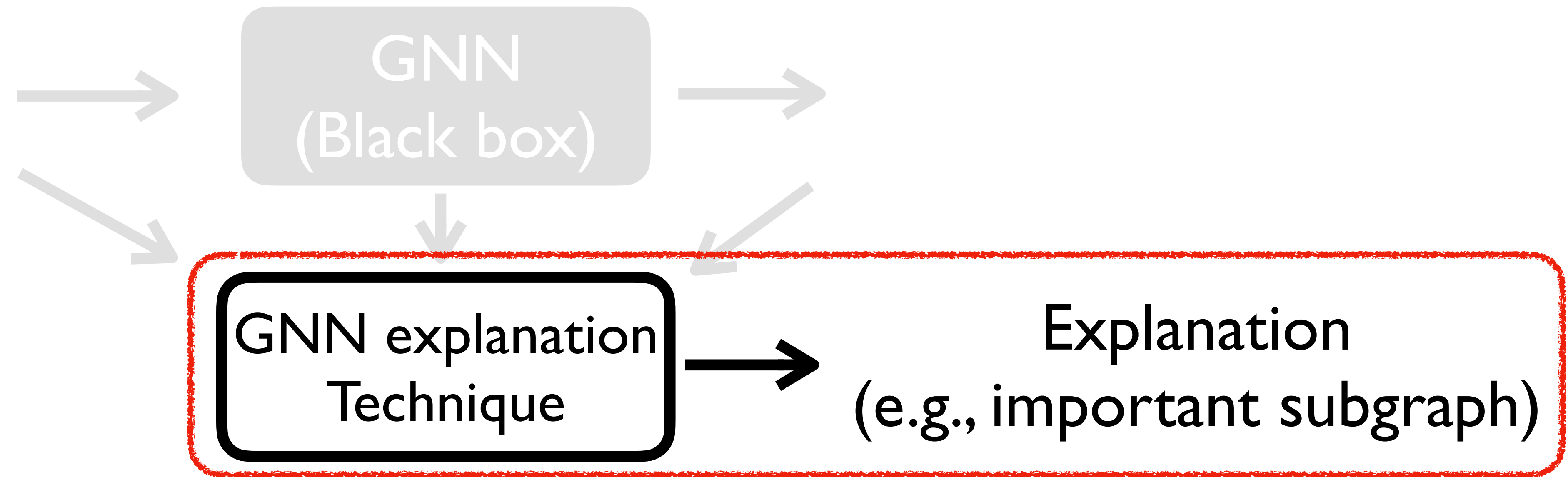
- Molnar [2022]



- Existing: **unexplainable** AI (GNN) + explanation technique

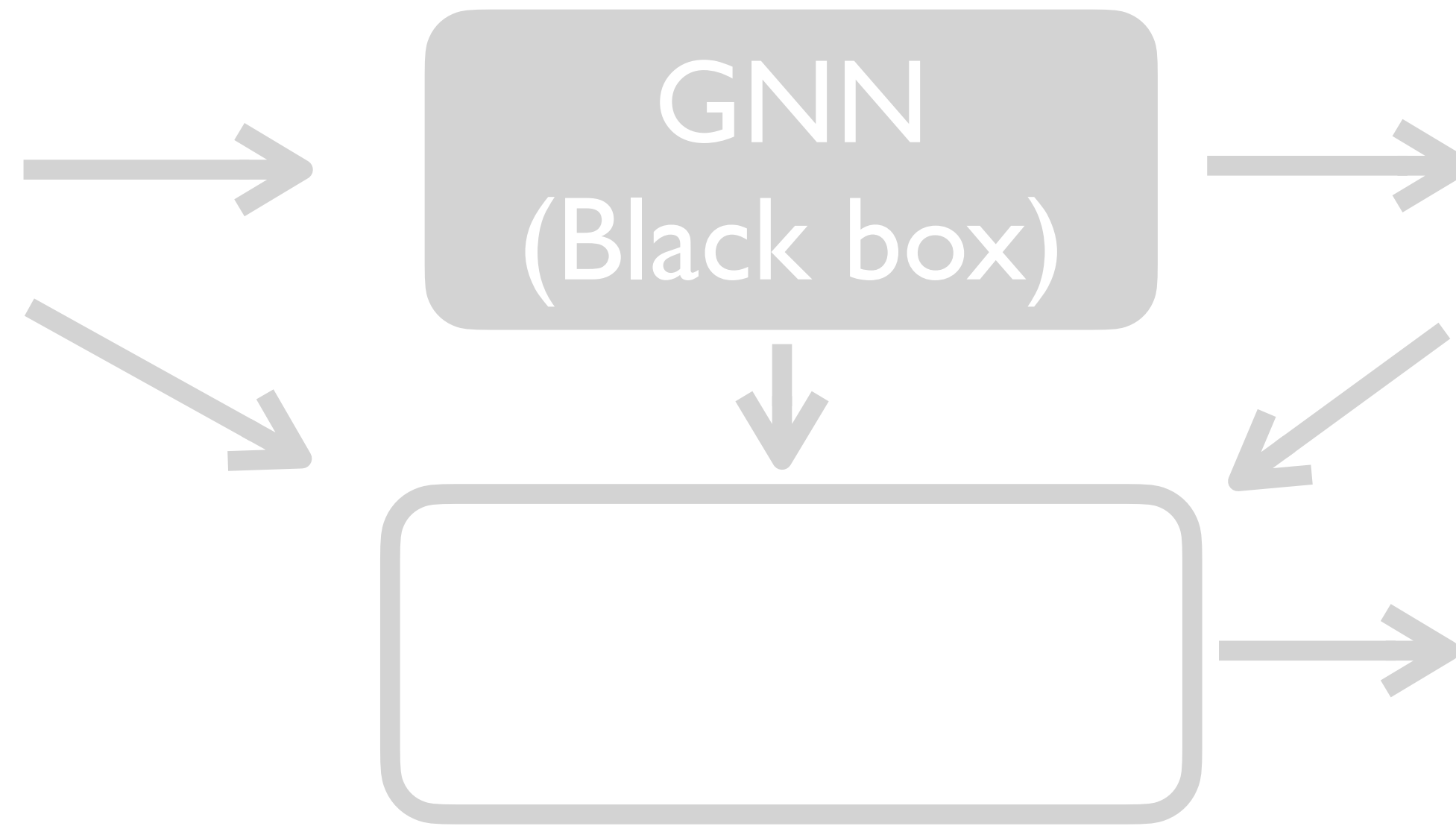


- Existing: **unexplainable** AI (GNN) + explanation technique



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

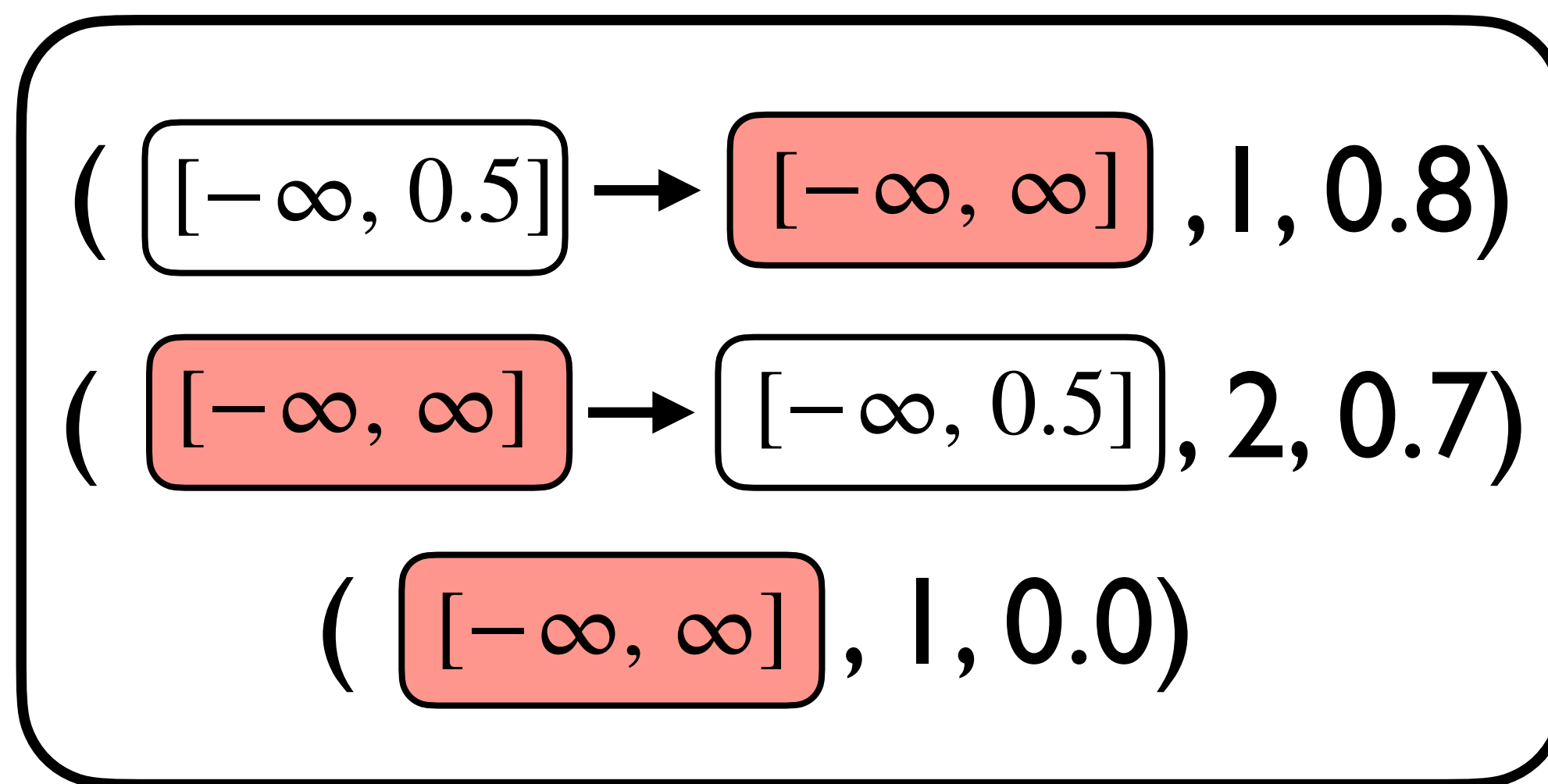
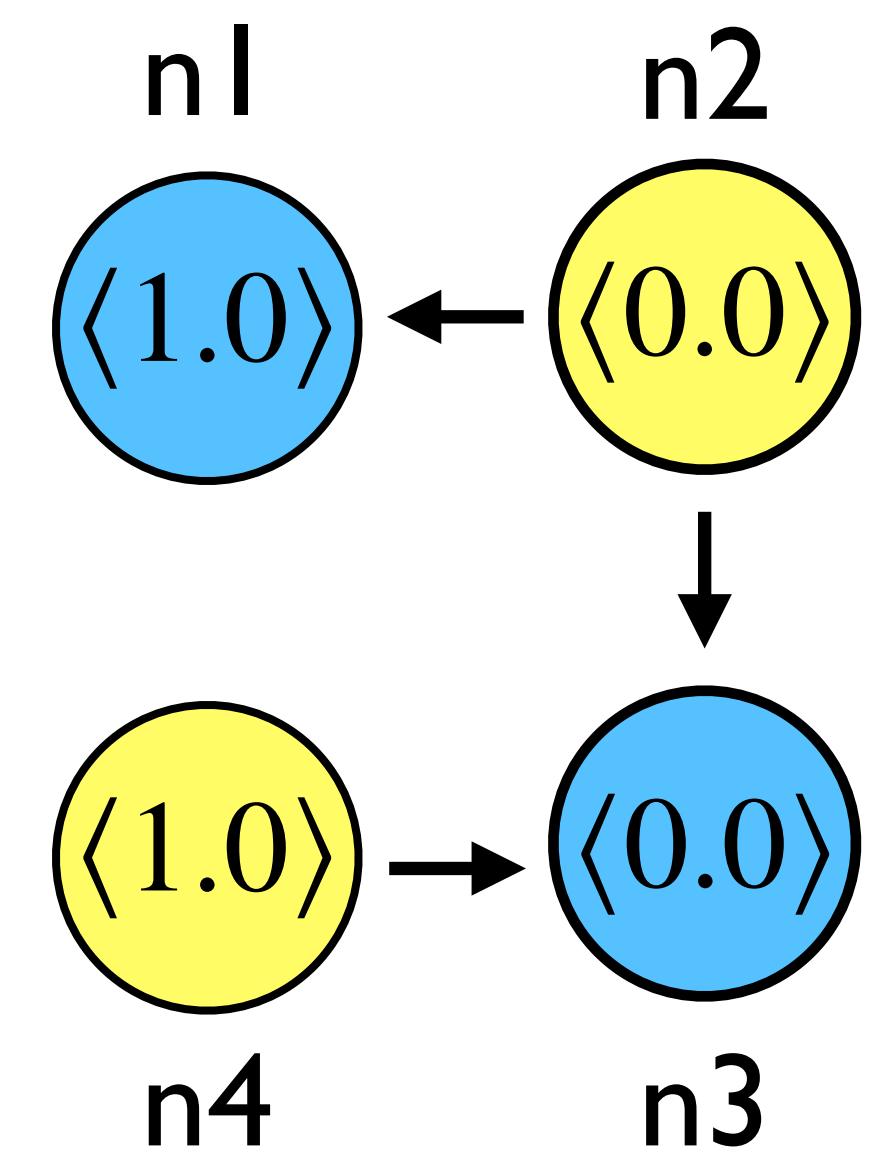
Semantics

$\llbracket \langle \phi_1, \dots, \phi_k \rangle \rrbracket$	$: \mathcal{P}(\mathbb{R}^k) = \{ \mathbf{f} \mid \mathbf{f} = (f_1, \dots, f_k) \wedge \forall i. \phi_i = [a, b] \Rightarrow a \leq f_i \leq b \}$
$\llbracket \text{node } x \langle \bar{\phi} \rangle \rrbracket$	$: \mathcal{P}(\mathbb{G} \times \mathbb{H}) = \{ (G, \eta) \mid v = \eta(x) \wedge \mathbf{f}_v^G \in \llbracket \langle \bar{\phi} \rangle \rrbracket \}$
$\llbracket \text{edge } (x, y) \langle \bar{\phi} \rangle \rrbracket$	$: \mathcal{P}(\mathbb{G} \times \mathbb{H}) = \{ (G, \eta) \mid e \in E \wedge e = (\eta(x), \eta(y)) \wedge \mathbf{f}_e^G \in \llbracket \langle \bar{\phi} \rangle \rrbracket \}$
$\llbracket \delta_1 \delta_2 \dots \delta_k \rrbracket$	$: \mathcal{P}(\mathbb{G} \times \mathbb{H}) = \{ (G, \eta) \mid \forall i. (G, \eta) \in \llbracket \delta_i \rrbracket \}$
$\llbracket \bar{\delta} \text{ target node } x \rrbracket$	$: \mathcal{P}(\mathbb{G} \times V) = \{ (G, v) \mid \exists (G, \eta) \in \llbracket \bar{\delta} \rrbracket. v = \eta(x) \}$
$\llbracket \bar{\delta} \text{ target edge } (x, y) \rrbracket$	$: \mathcal{P}(\mathbb{G} \times E) = \{ (G, e) \mid \exists (G, \eta) \in \llbracket \bar{\delta} \rrbracket. e = (\eta(x), \eta(y)) \}$
$\llbracket \bar{\delta} \text{ target graph} \rrbracket$	$: \mathcal{P}(\mathbb{G}) = \{ G \mid \exists (G, \eta) \in \llbracket \bar{\delta} \rrbracket \}$

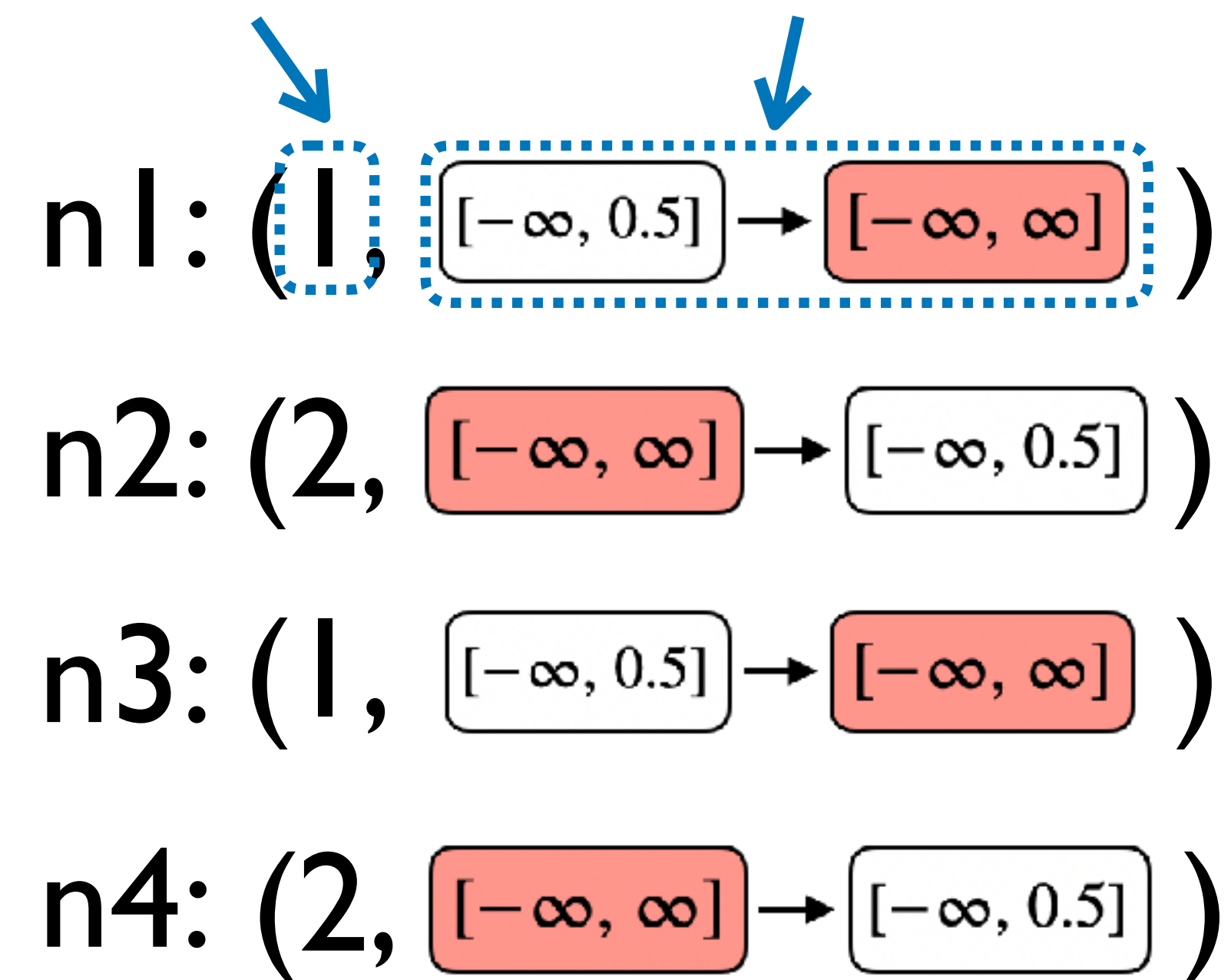
Input graph data

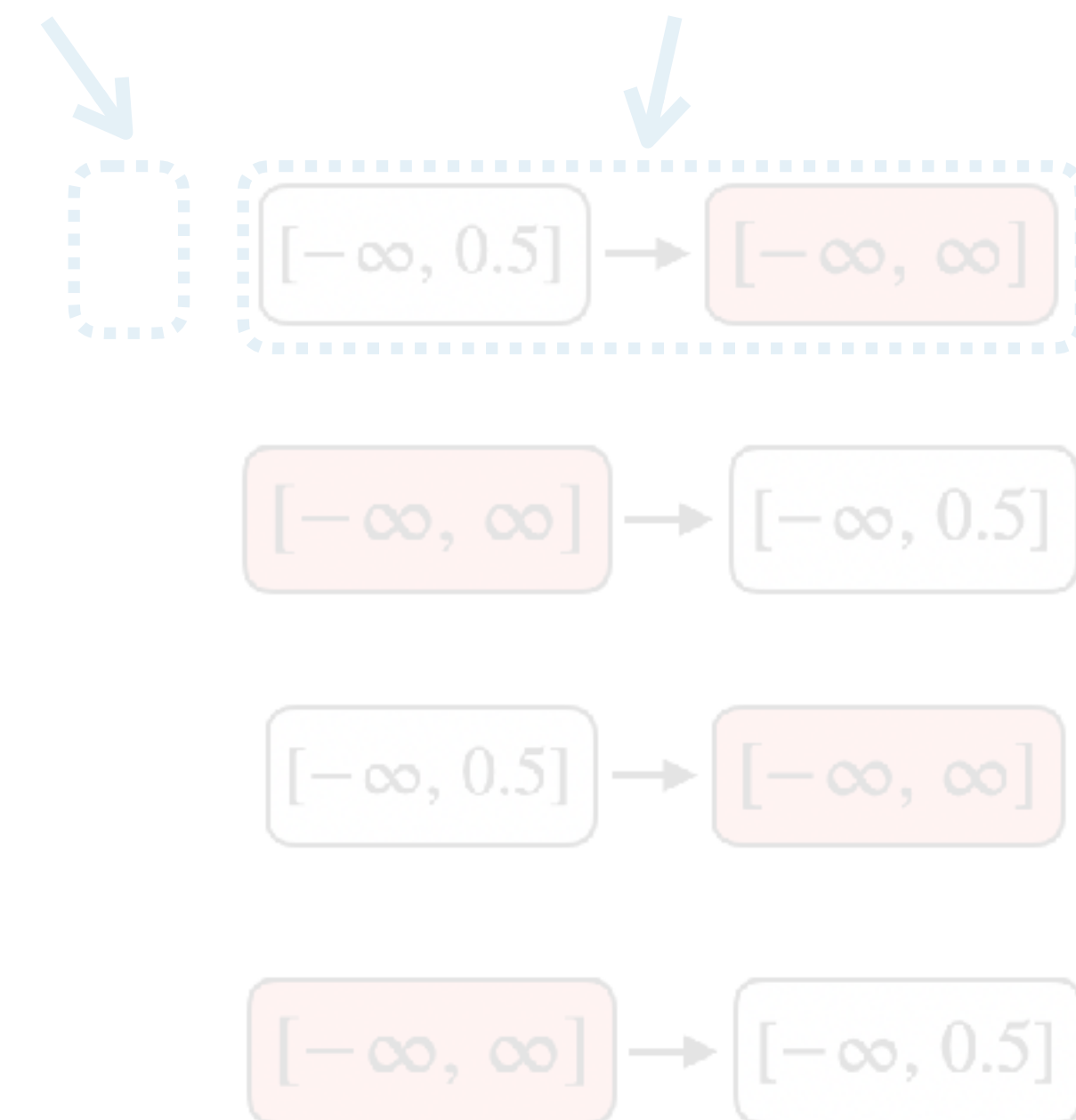
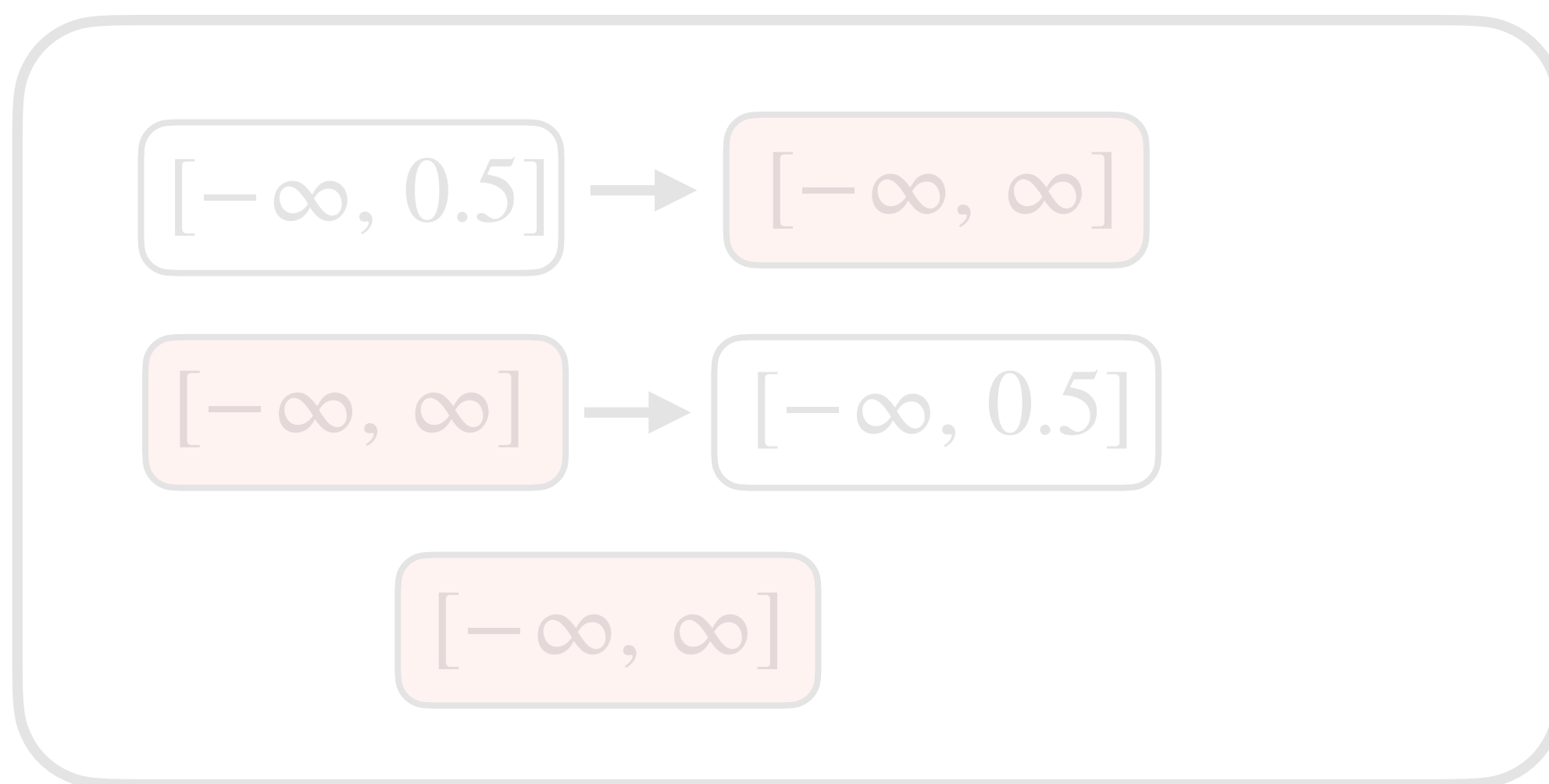
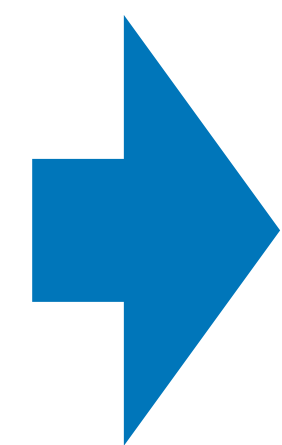
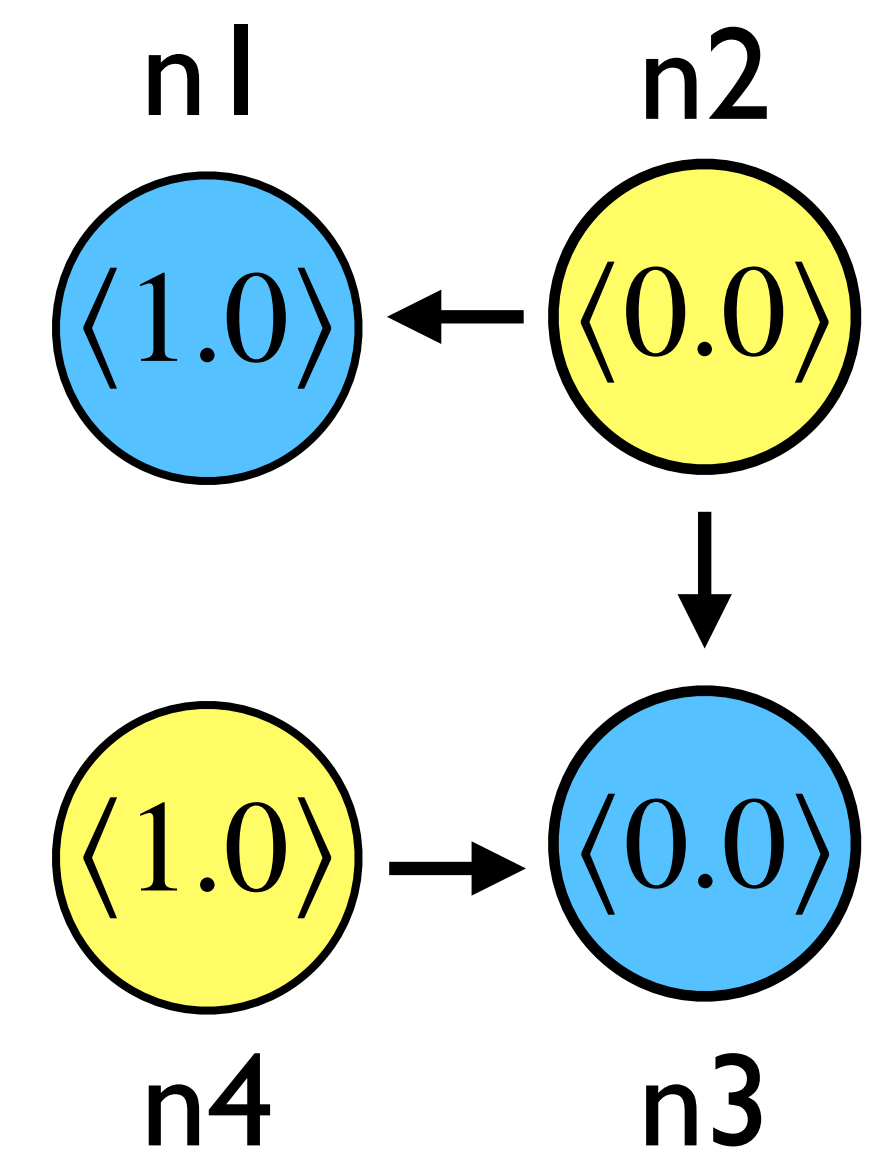
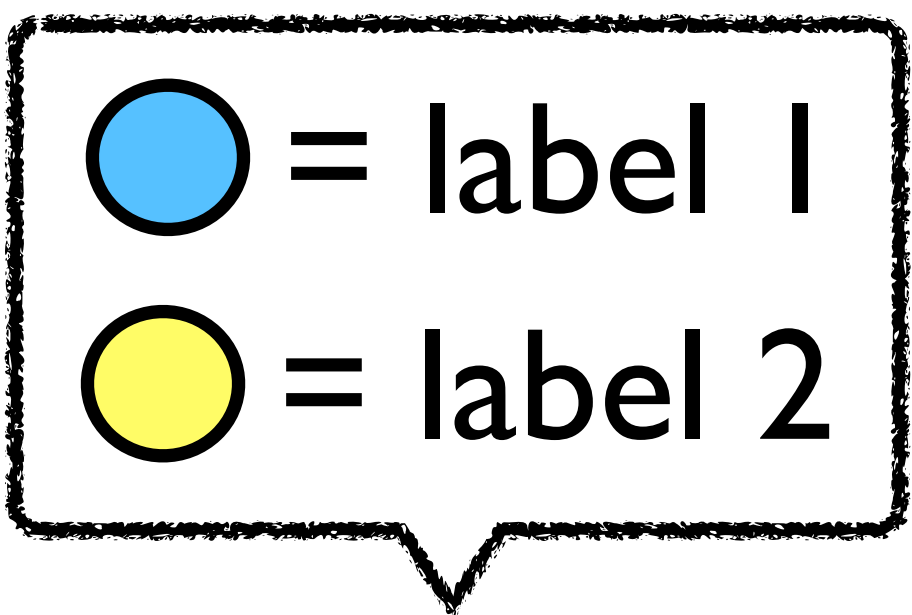


Prediction &
correct explanation

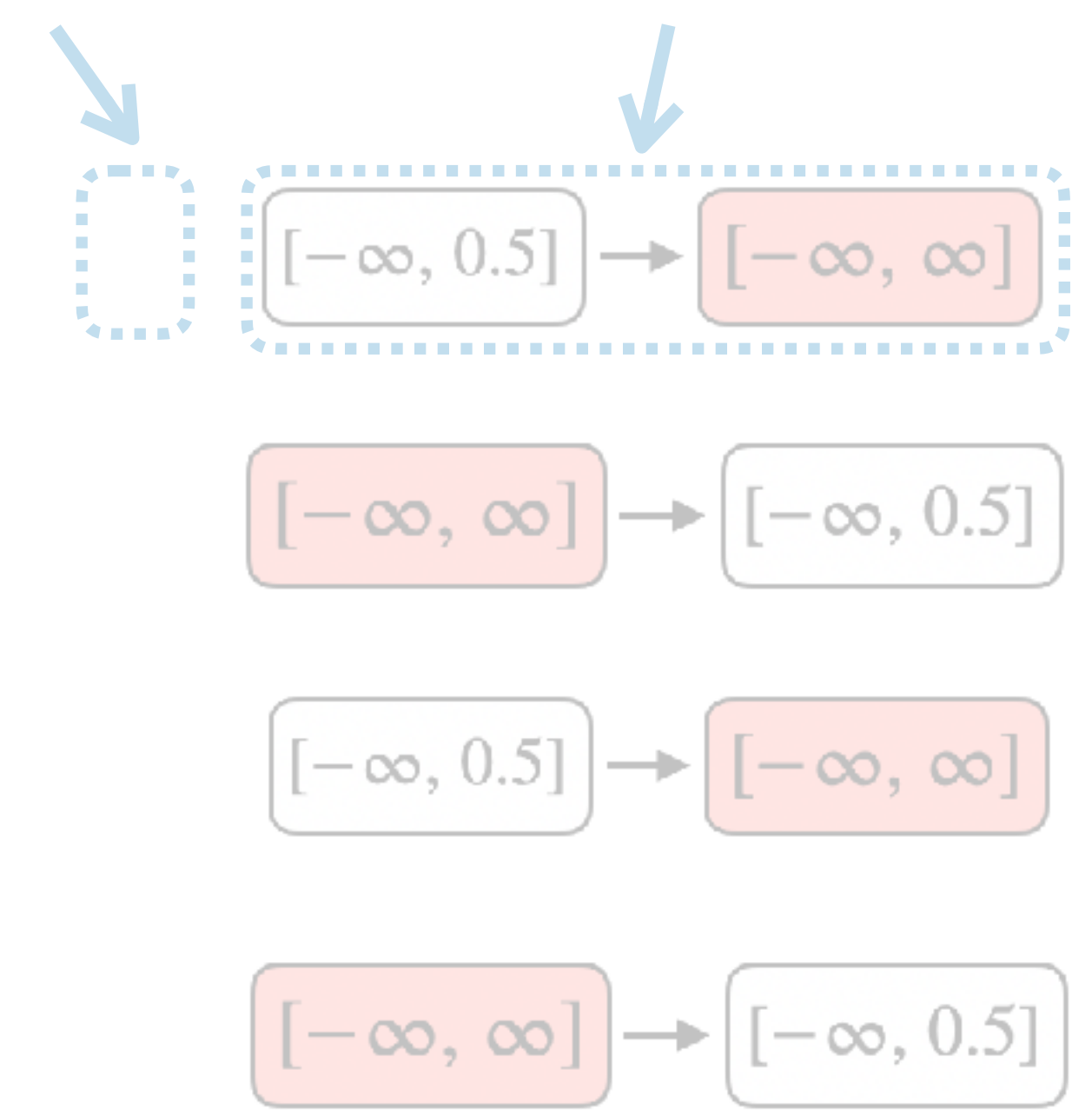
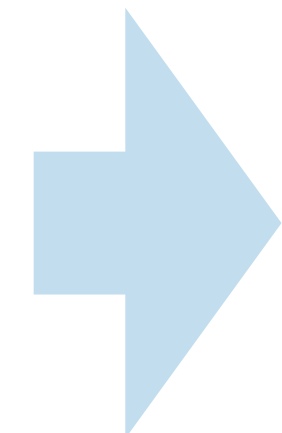
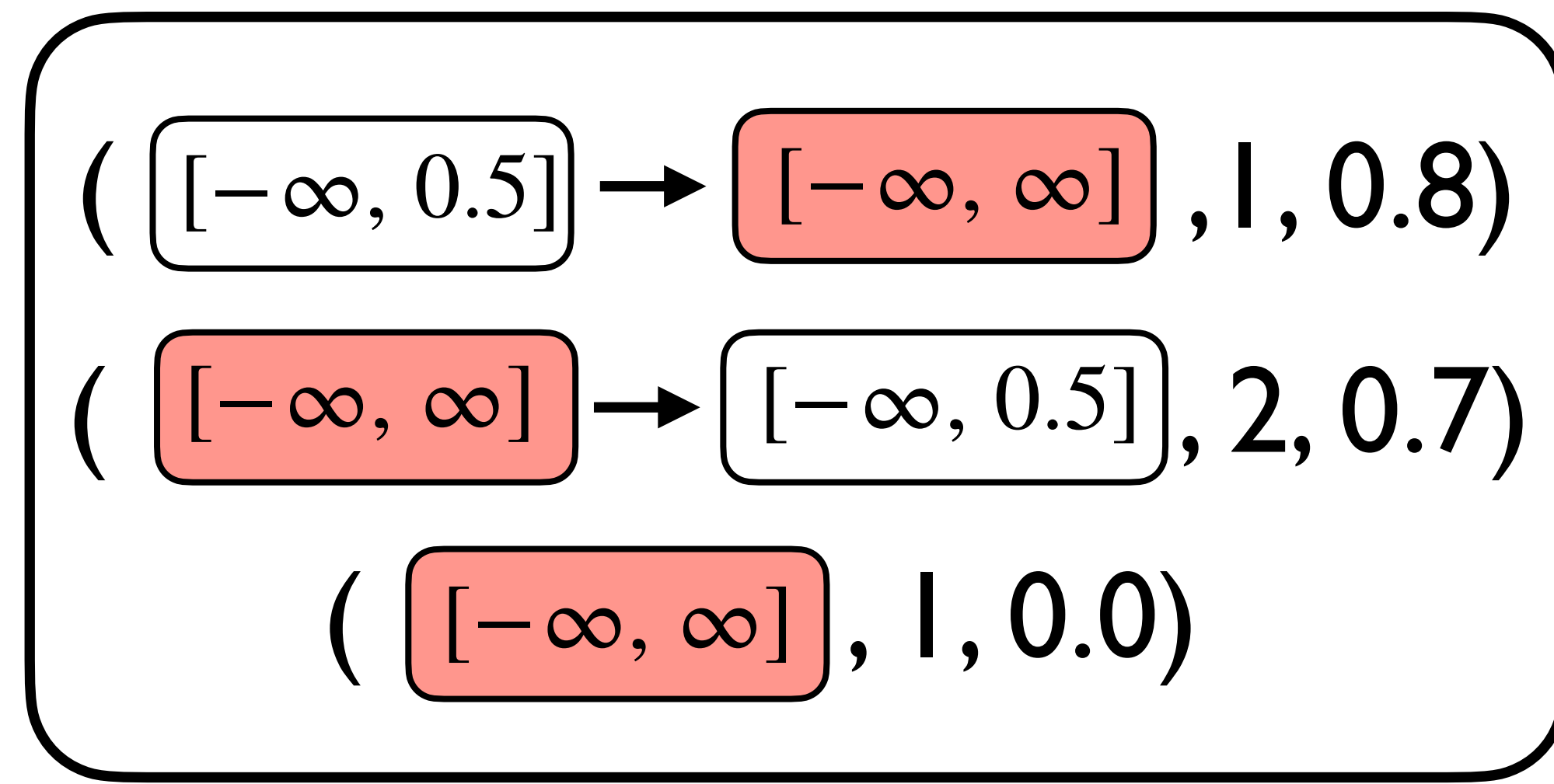
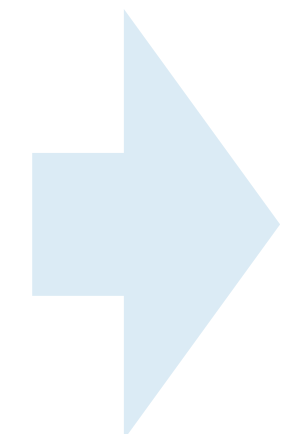
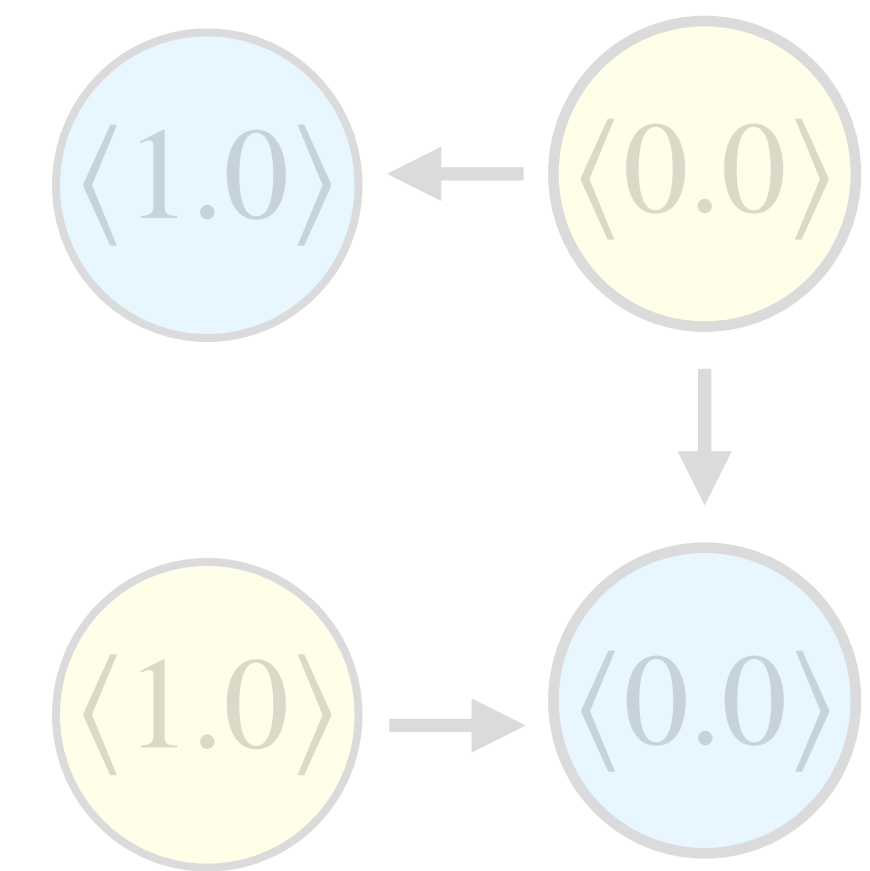


Prediction Explanation



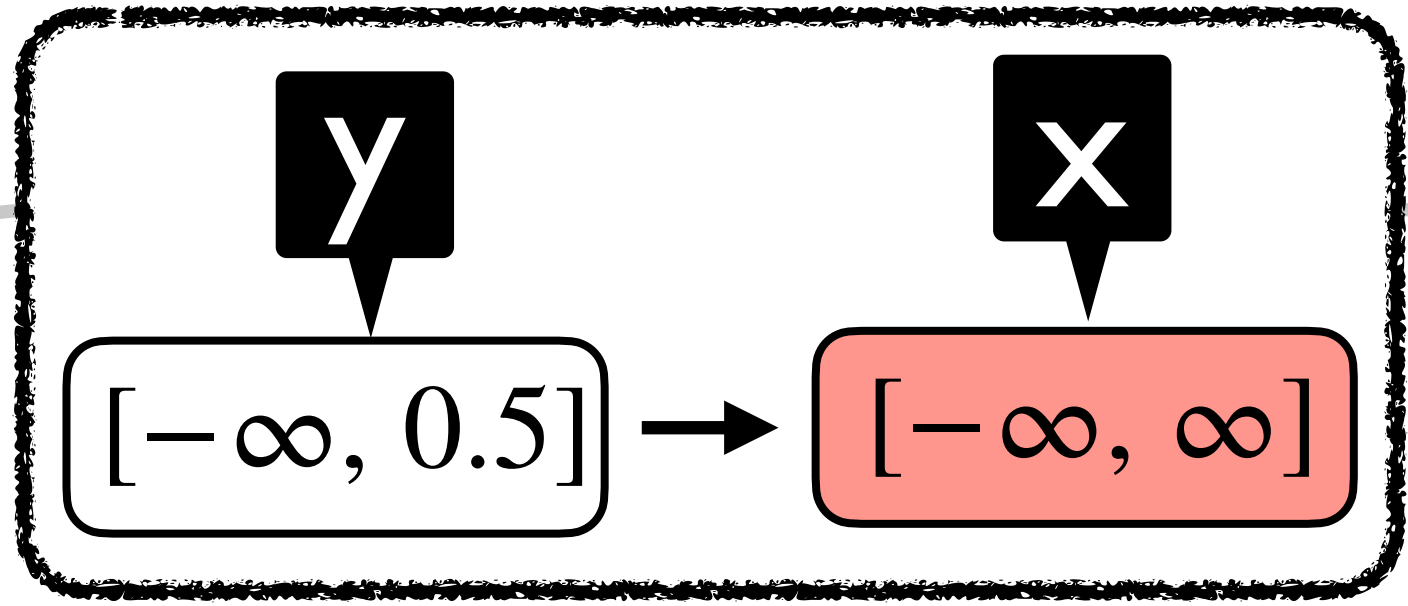


Graph data

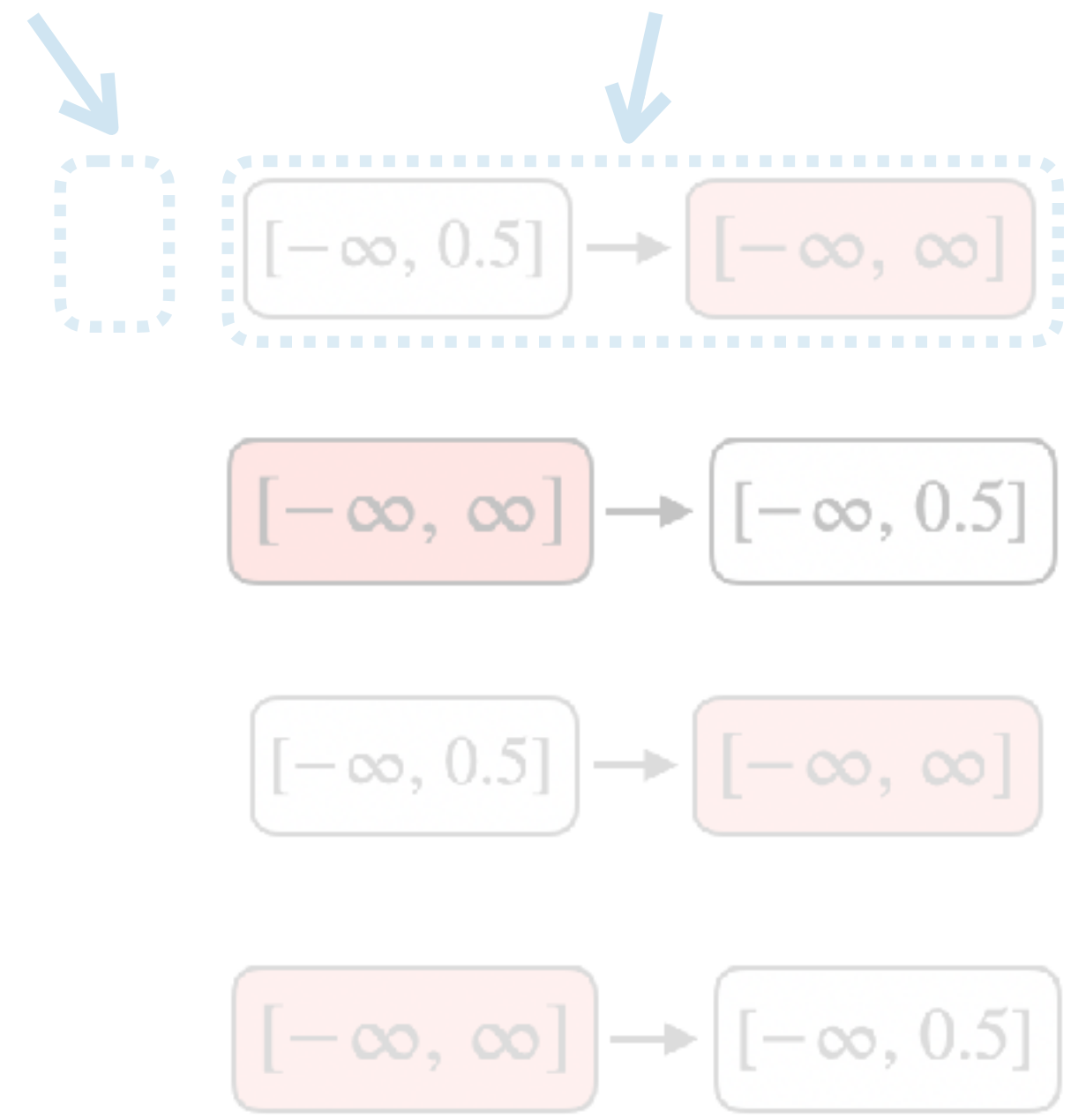
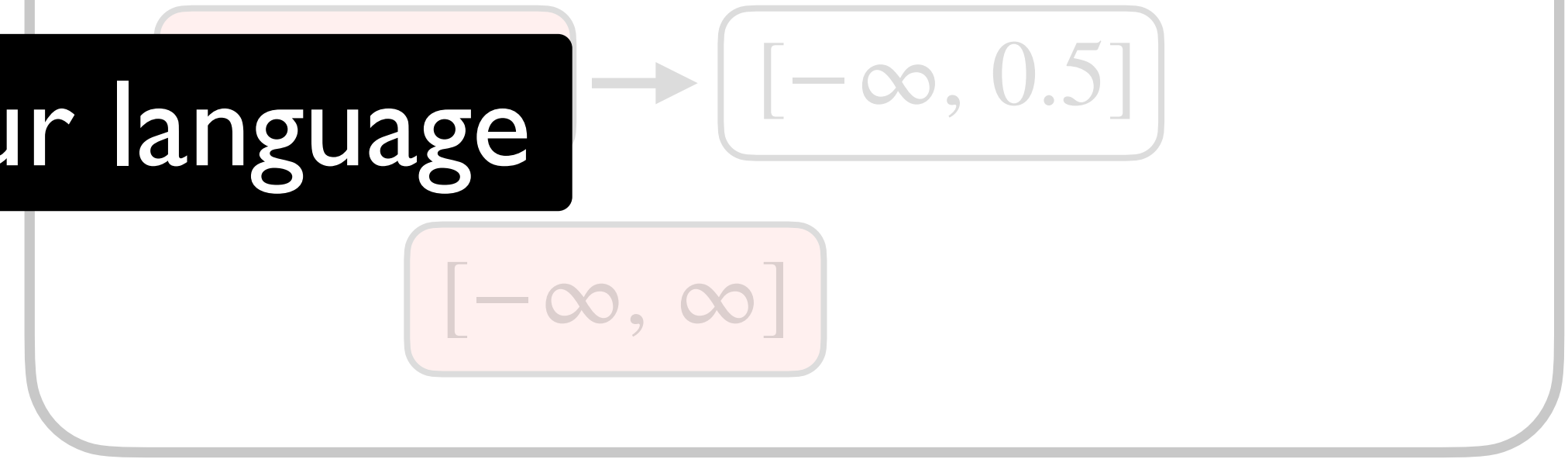
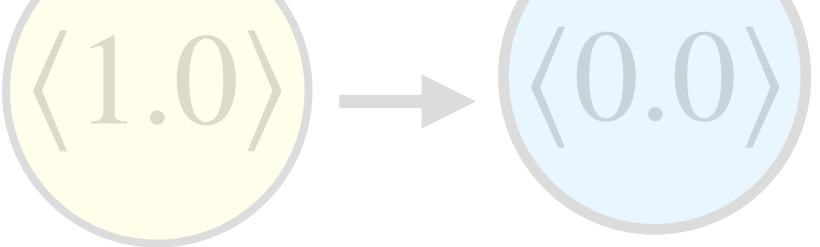


Our model

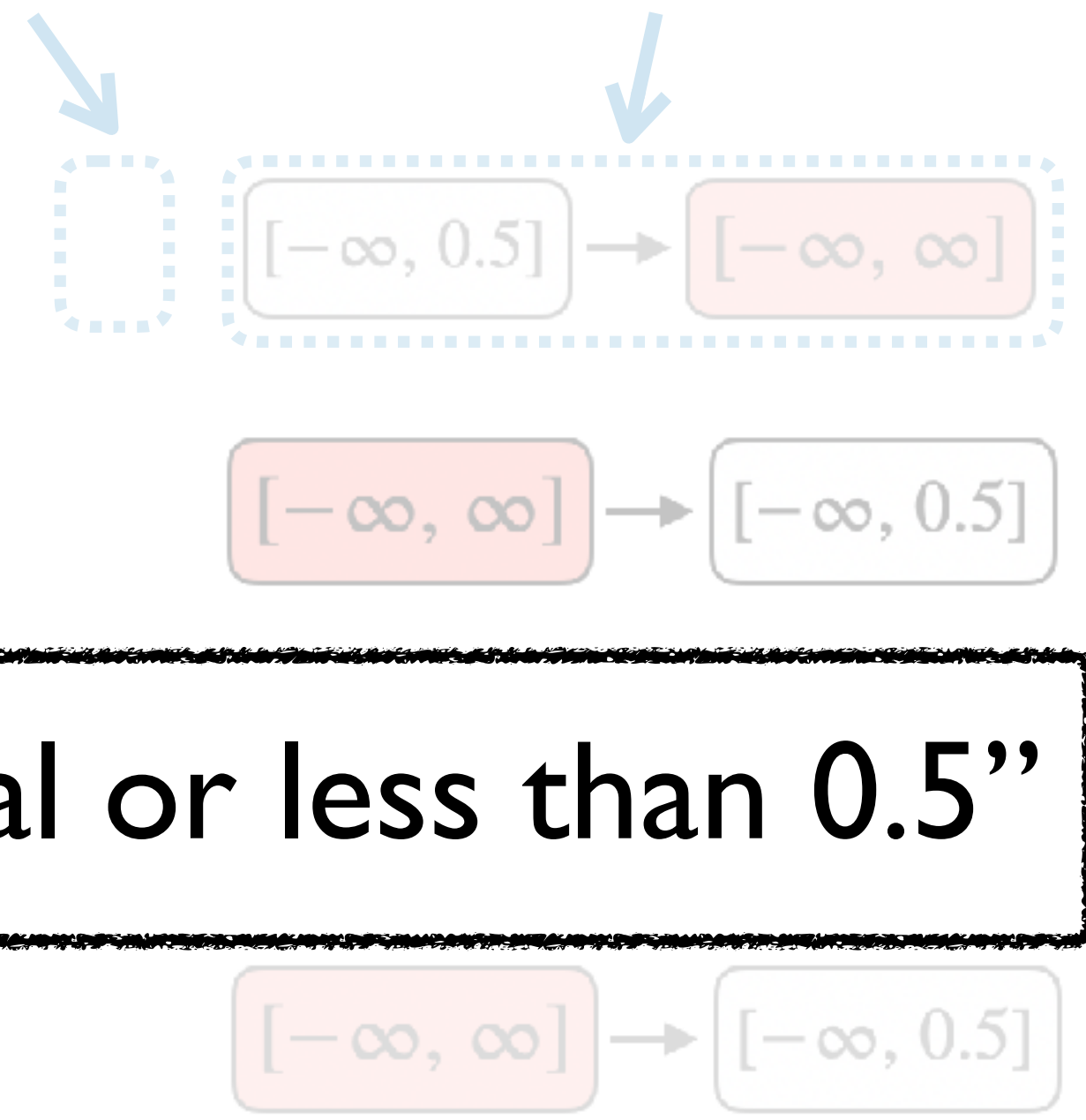
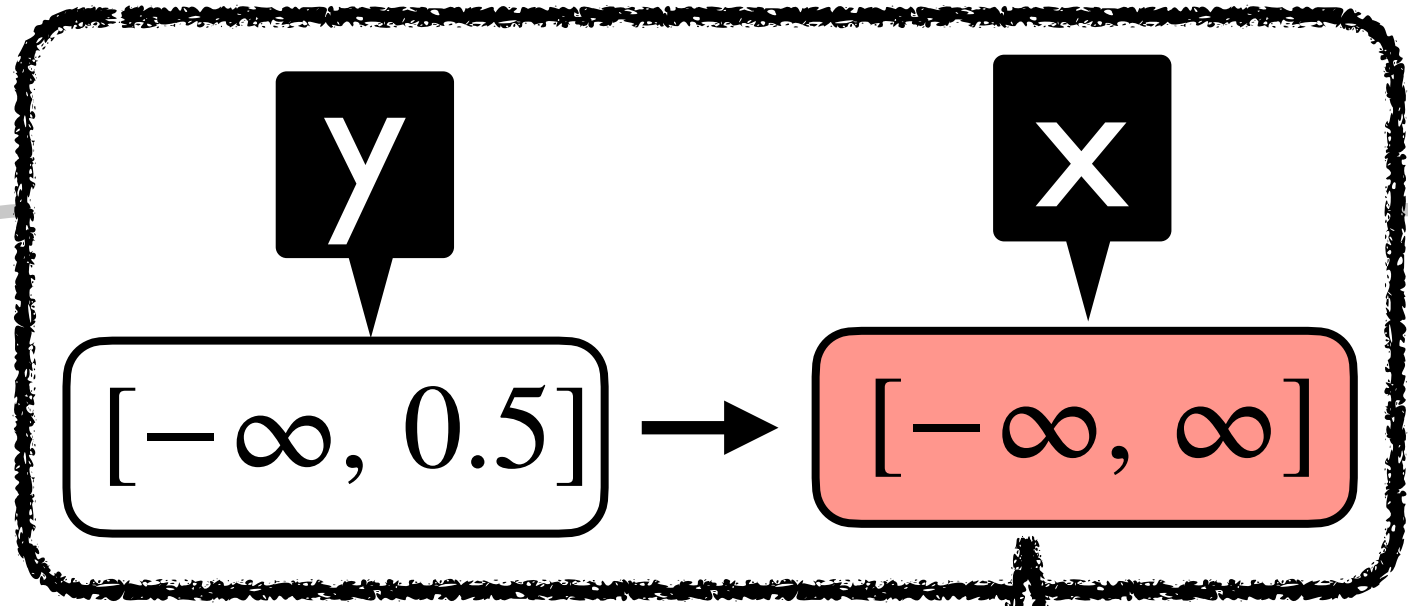
node x $\langle [-\infty, \infty] \rangle$
node y $\langle [-\infty, 0.5] \rangle$
edge (y, x)
target node x



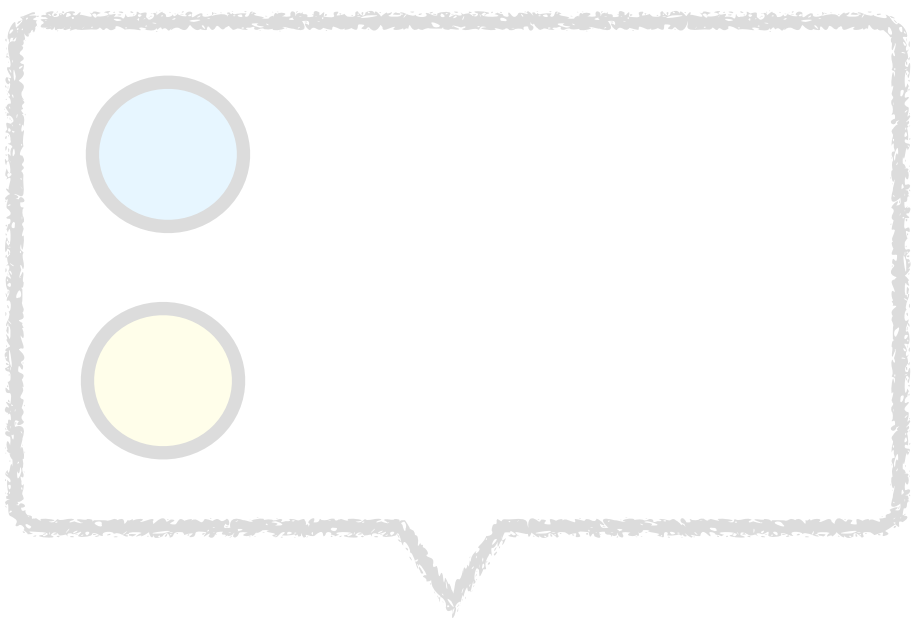
A program in our language



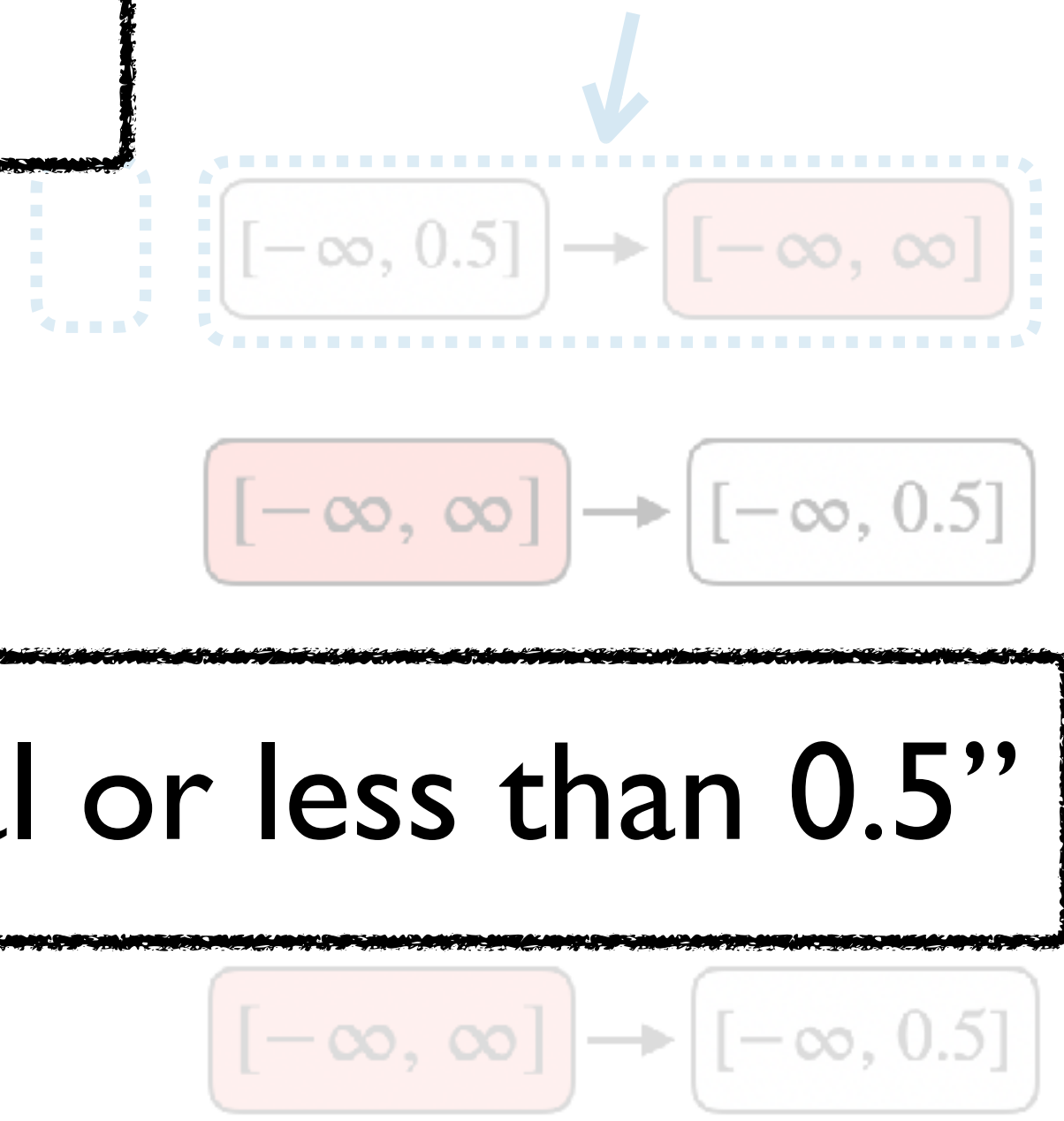
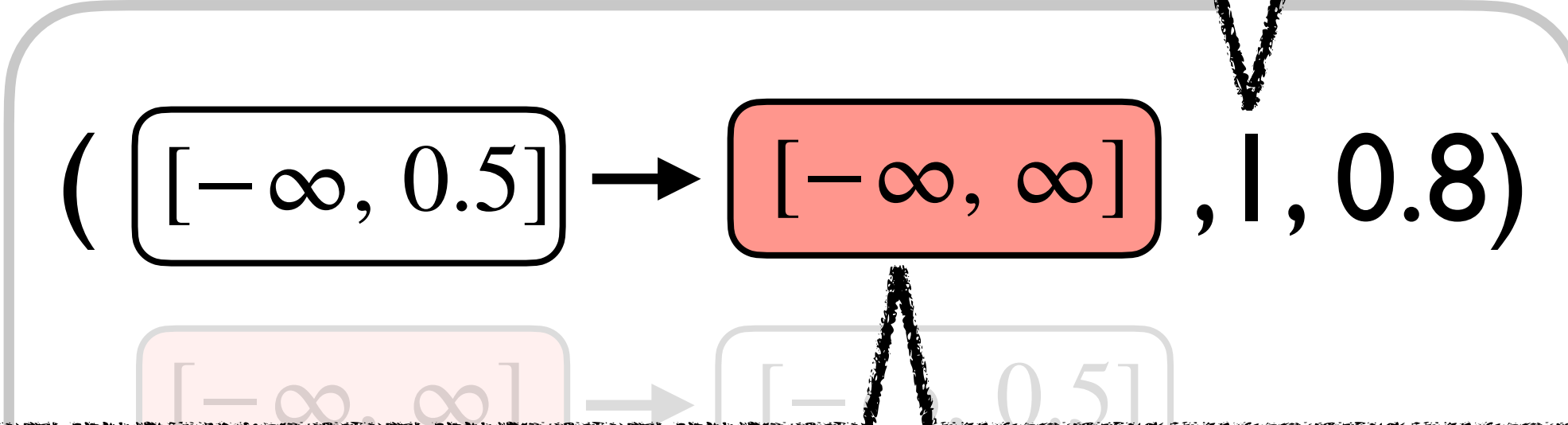
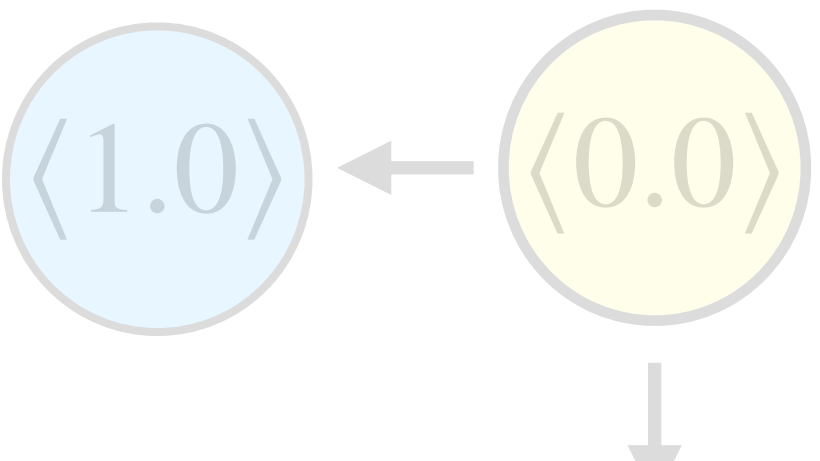
node x $\langle [-\infty, \infty] \rangle$
node y $\langle [-\infty, 0.5] \rangle$
edge (y, x)
target node x



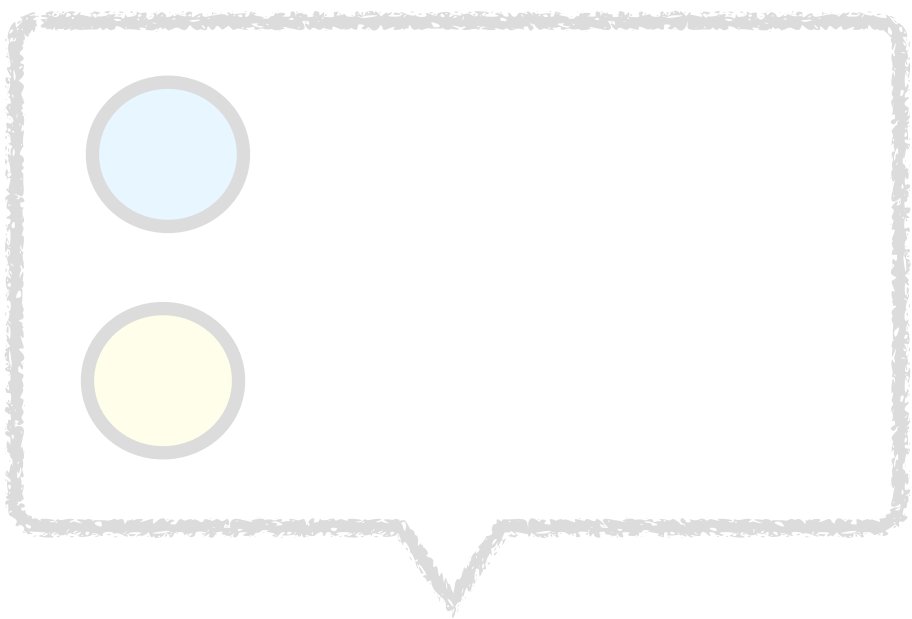
“Nodes having a predecessor whose feature value is equal or less than 0.5”



Described nodes will be classified into label 1

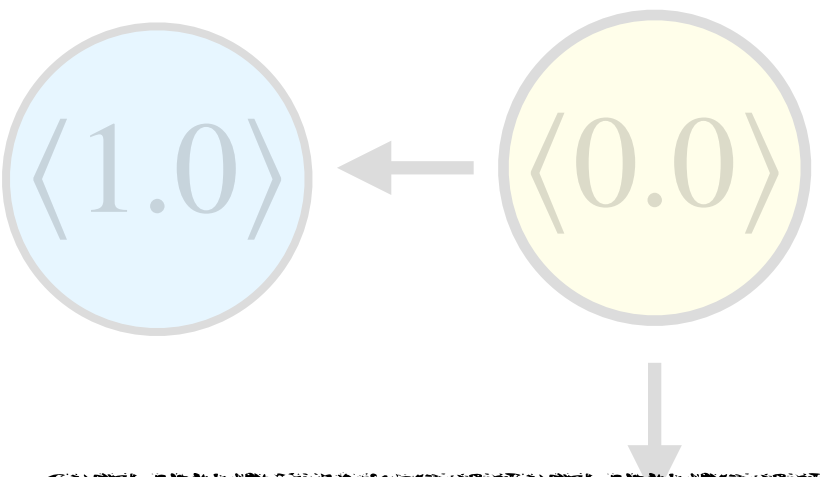
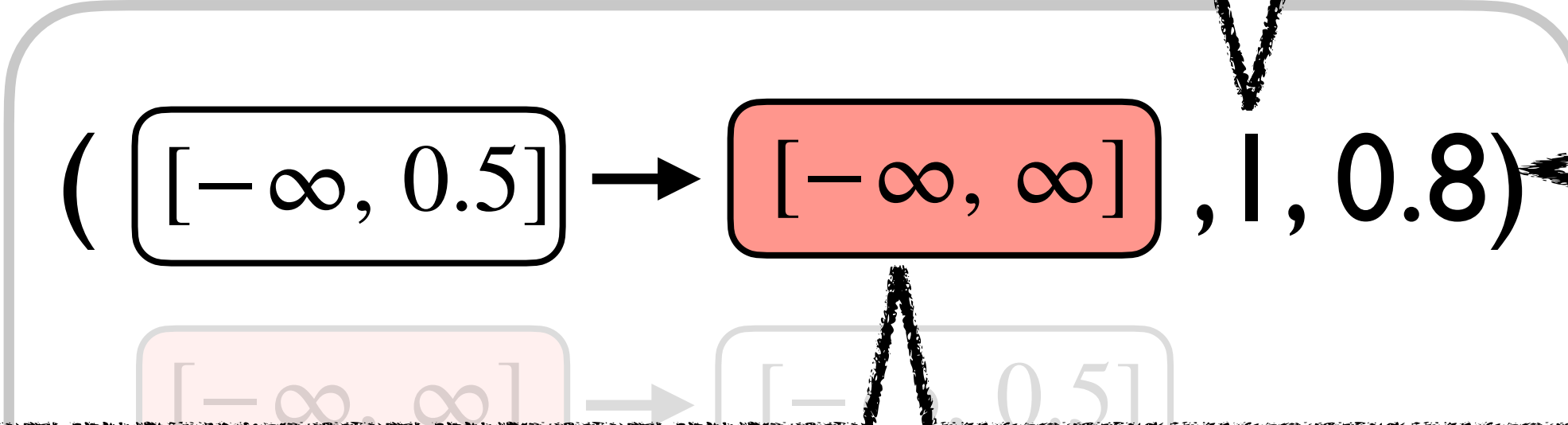


“Nodes having a predecessor whose feature value is equal or less than 0.5”

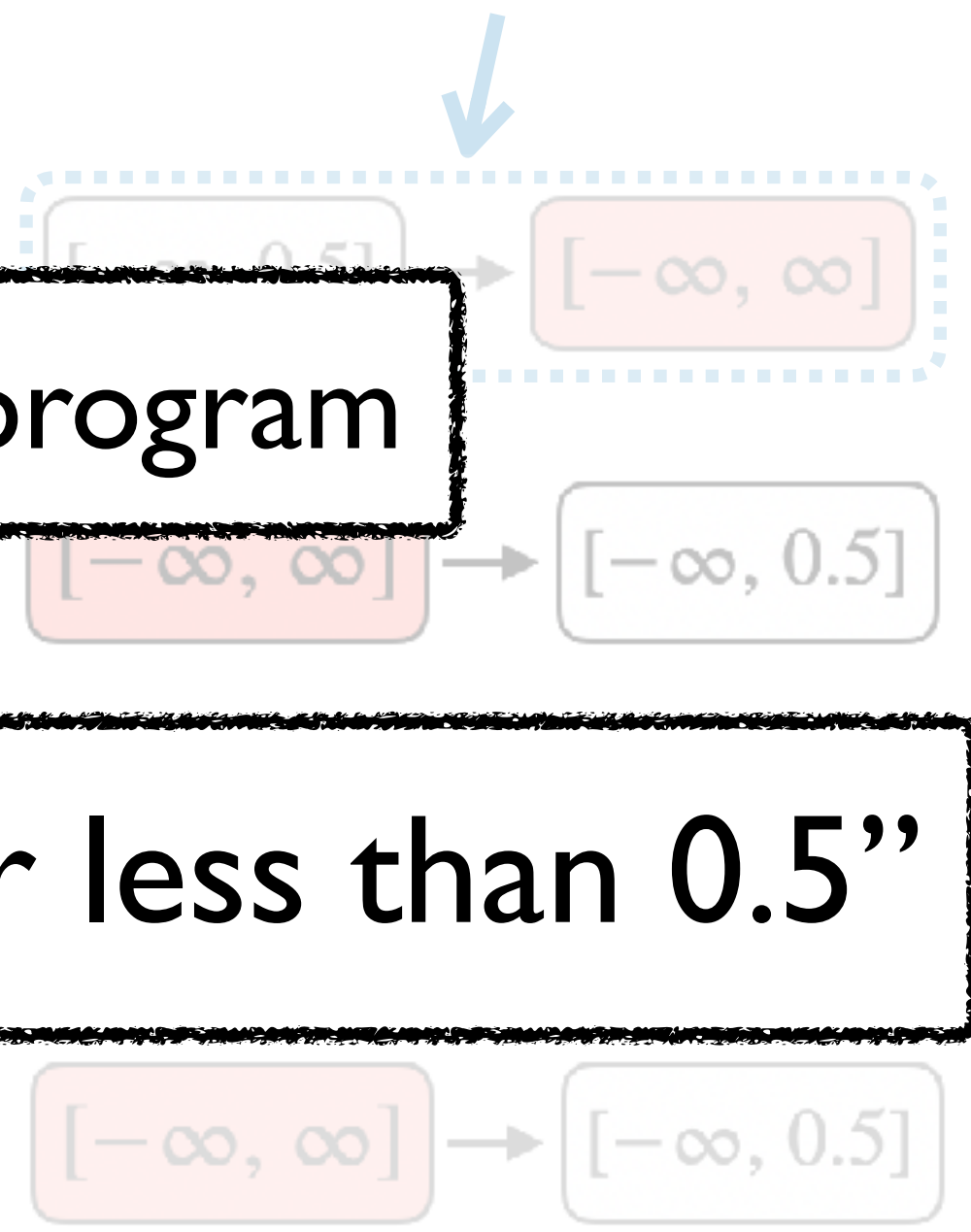


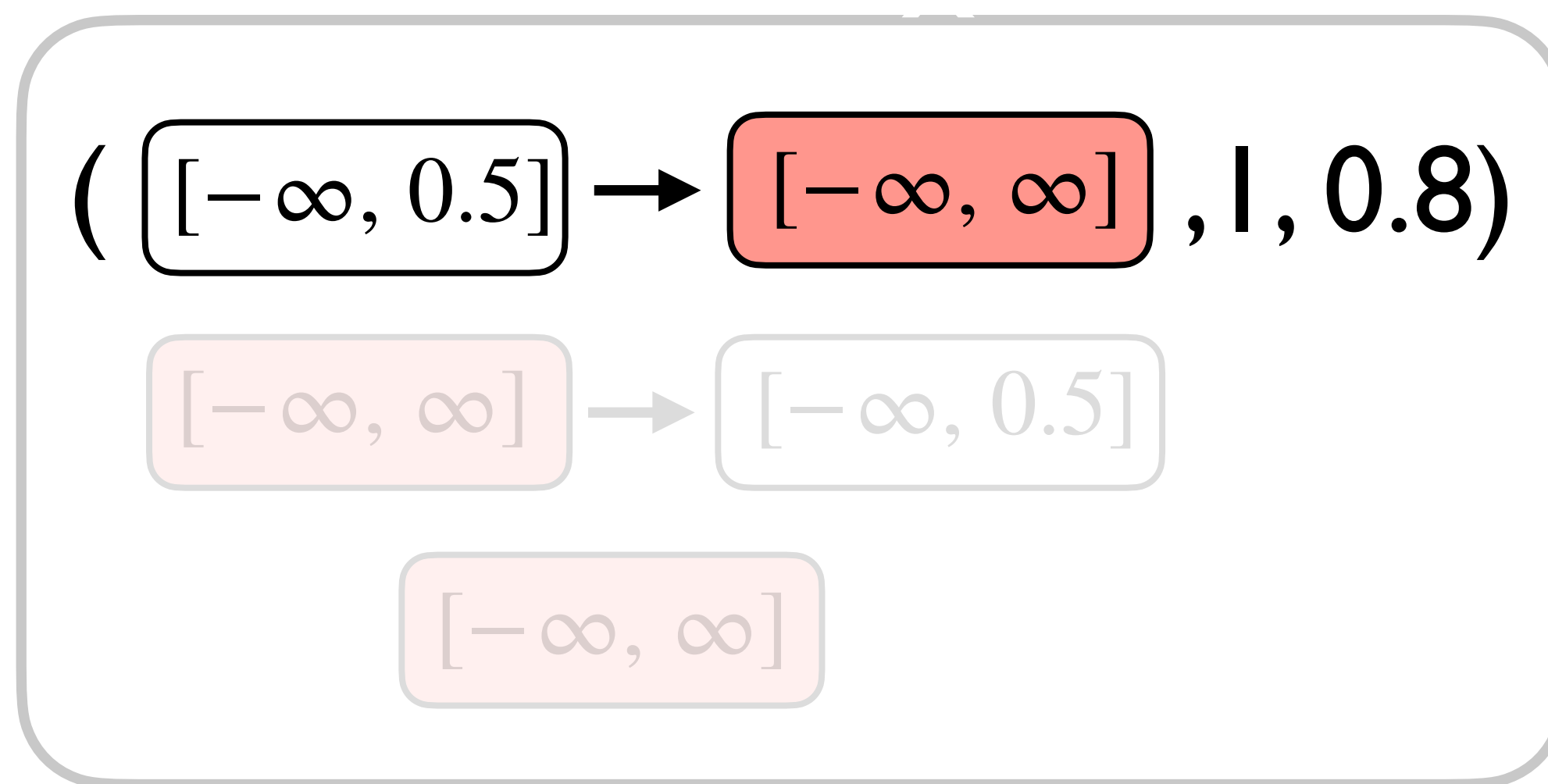
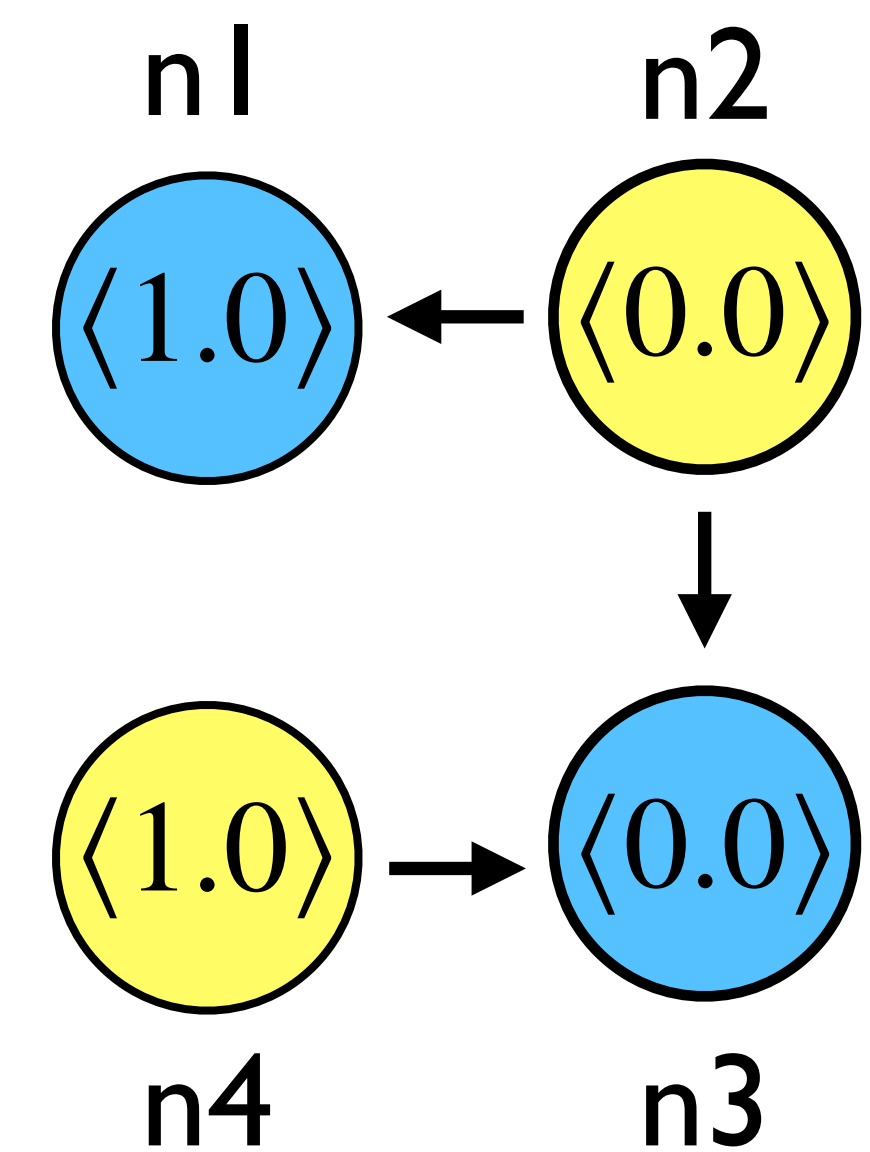
Described nodes will be classified into label I

Score of the program

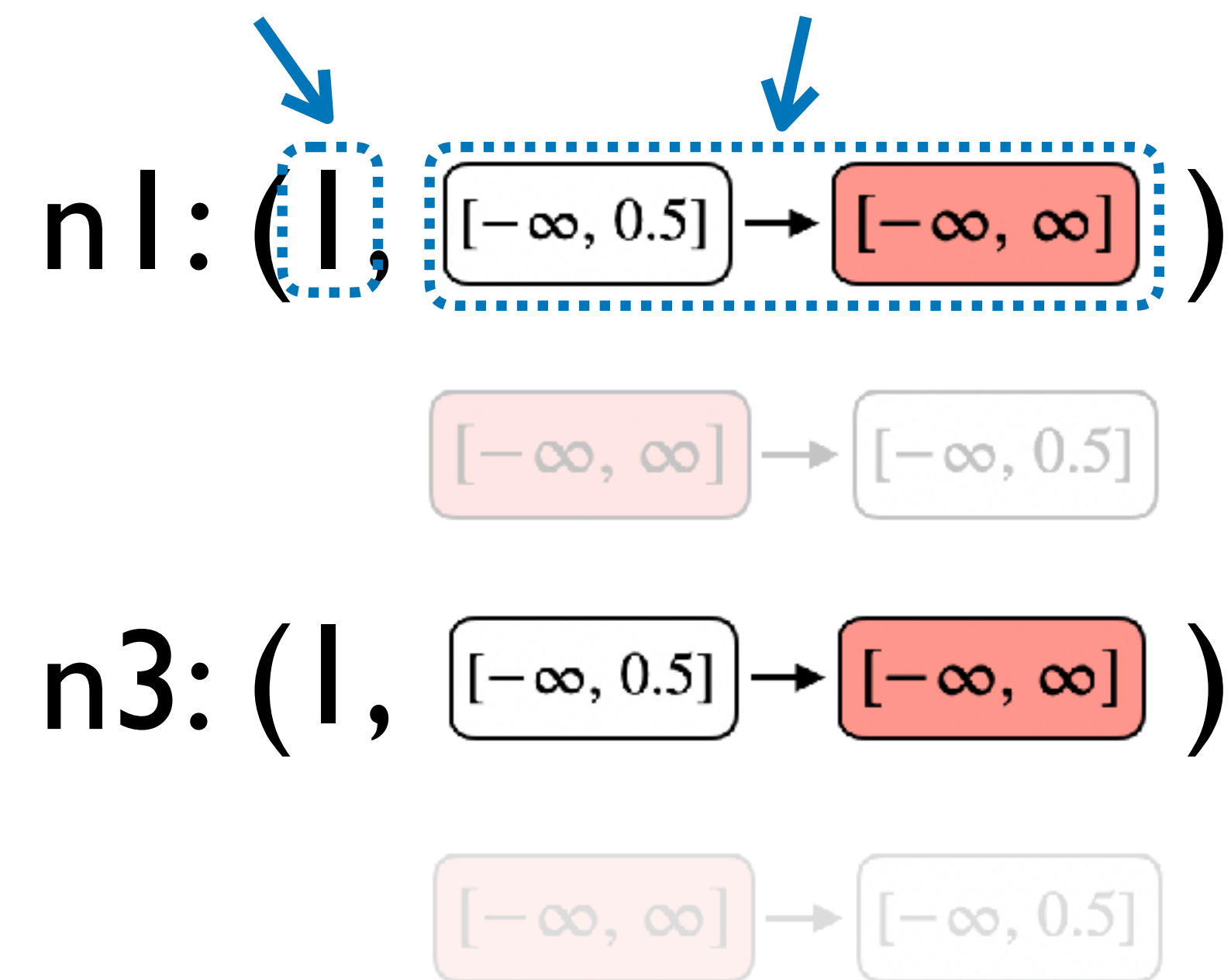


“Nodes having a predecessor whose feature value is equal or less than 0.5”

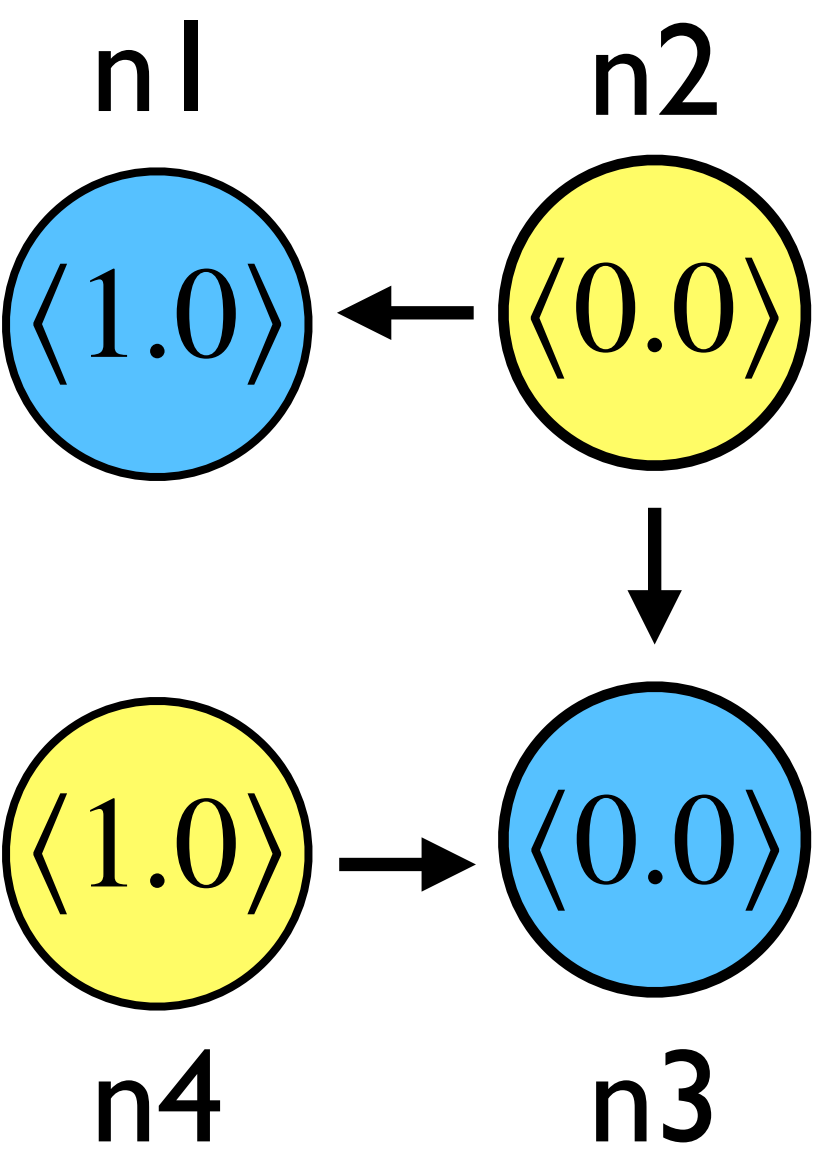




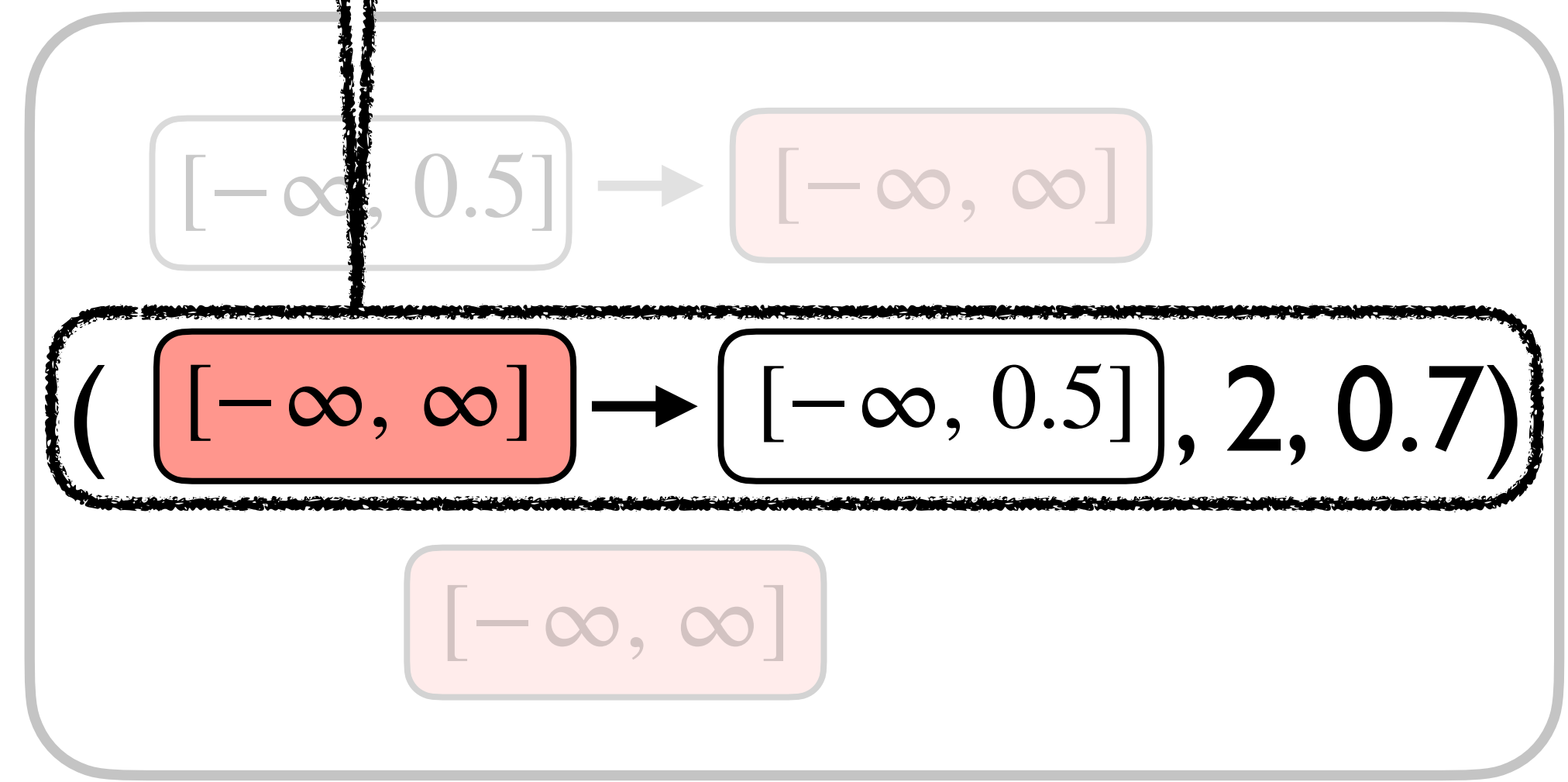
Prediction Explanation



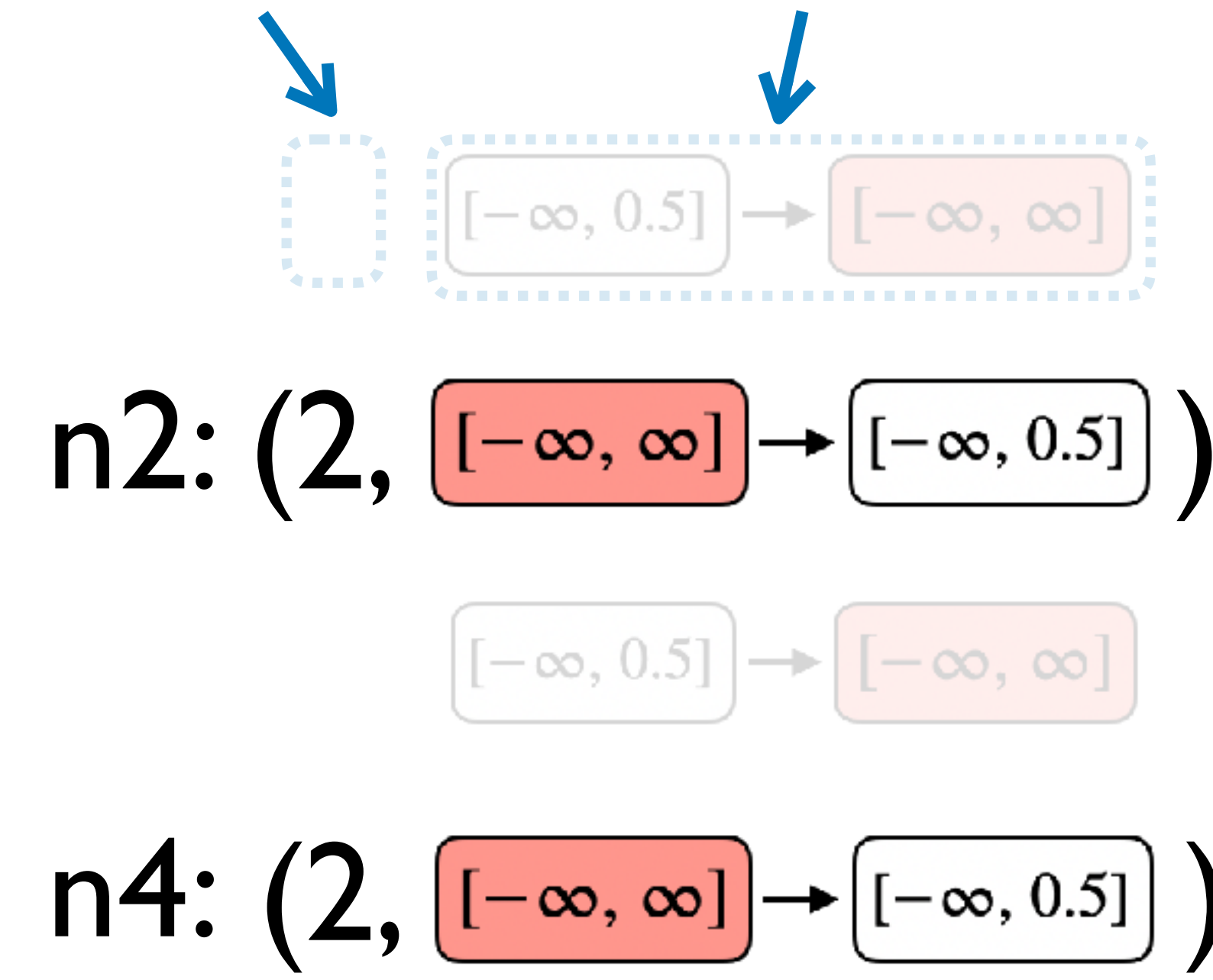
“Nodes having a successor whose feature value is equal or less than 0.5”

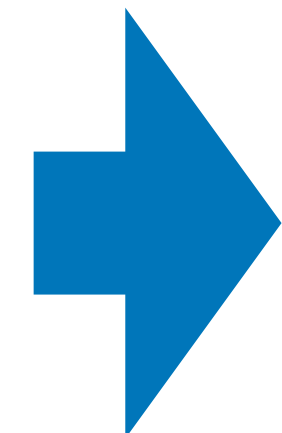
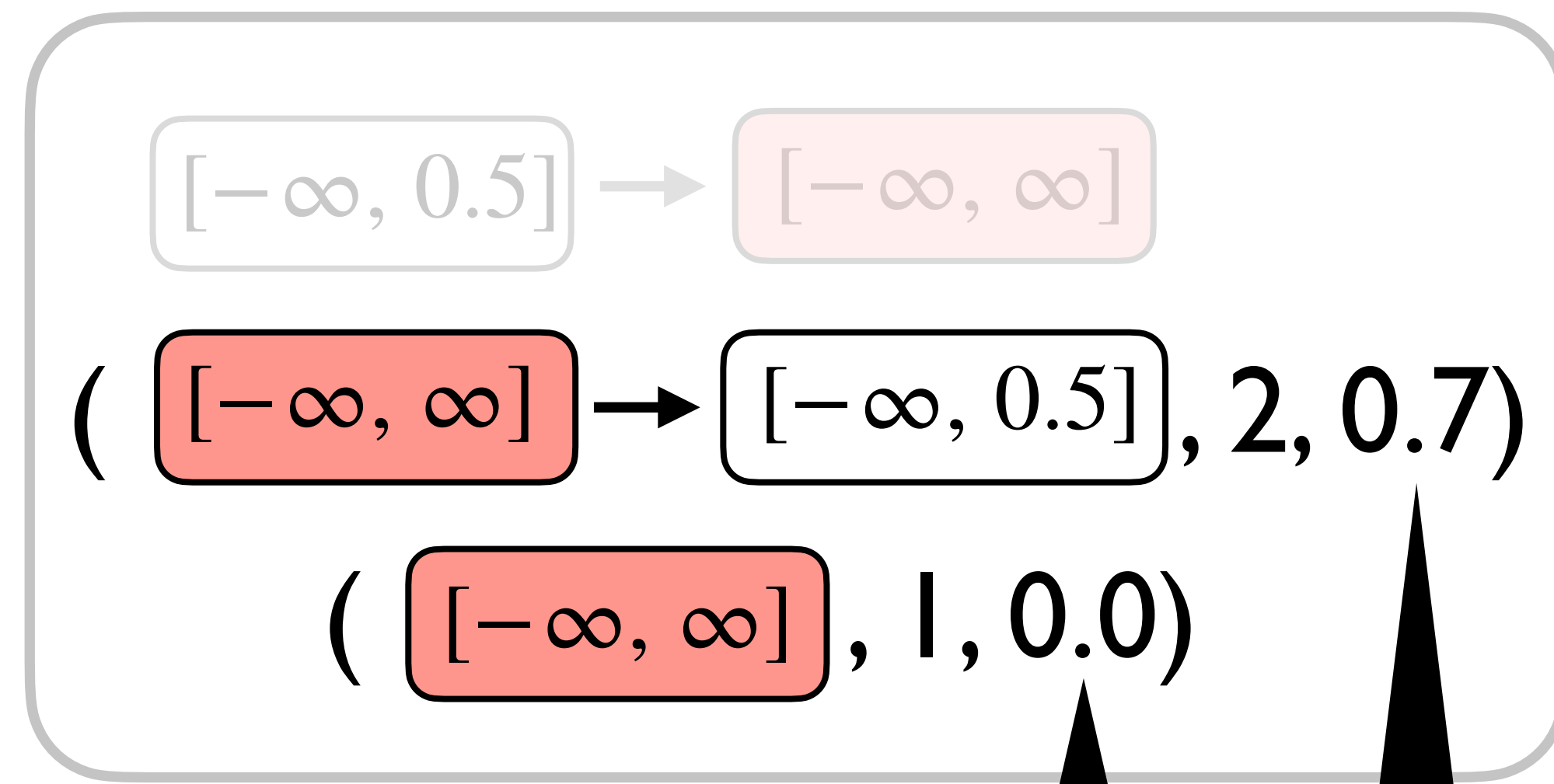
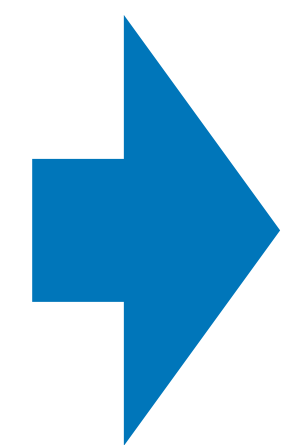
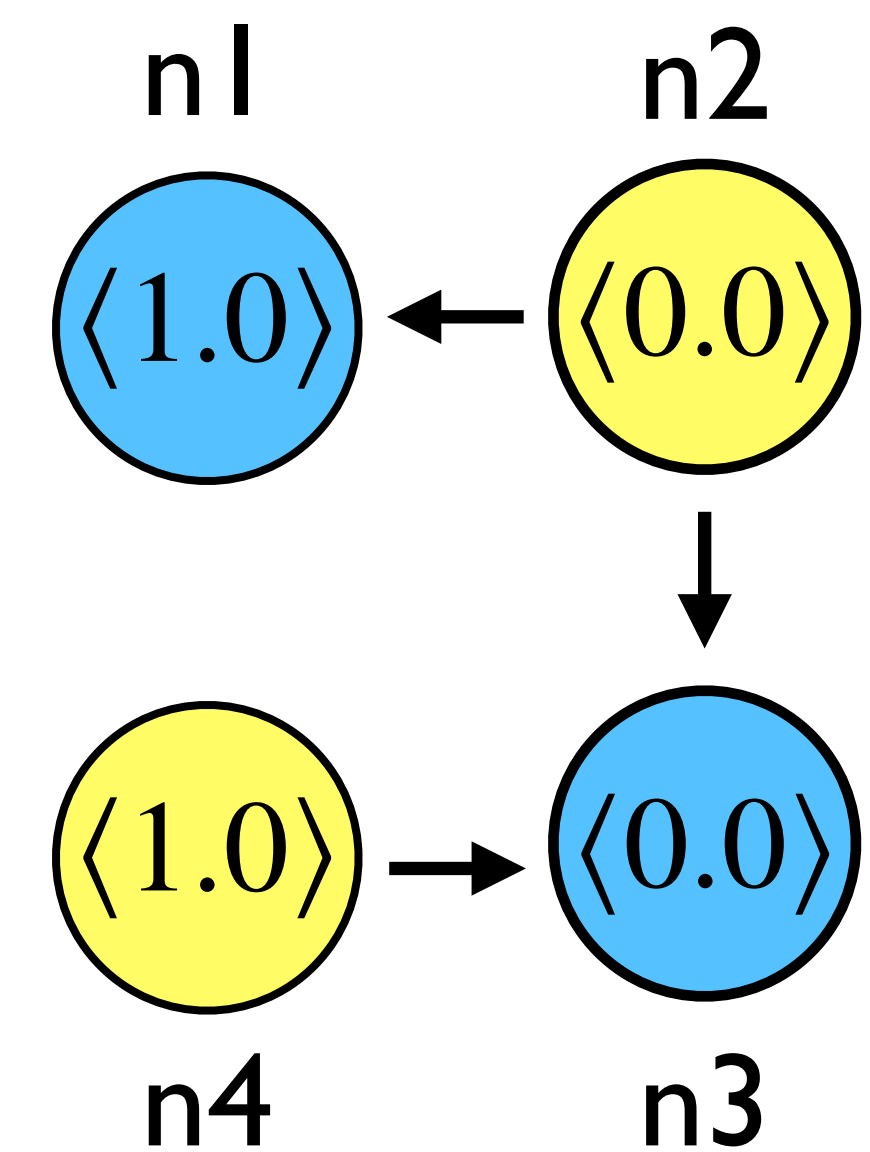


Graph data

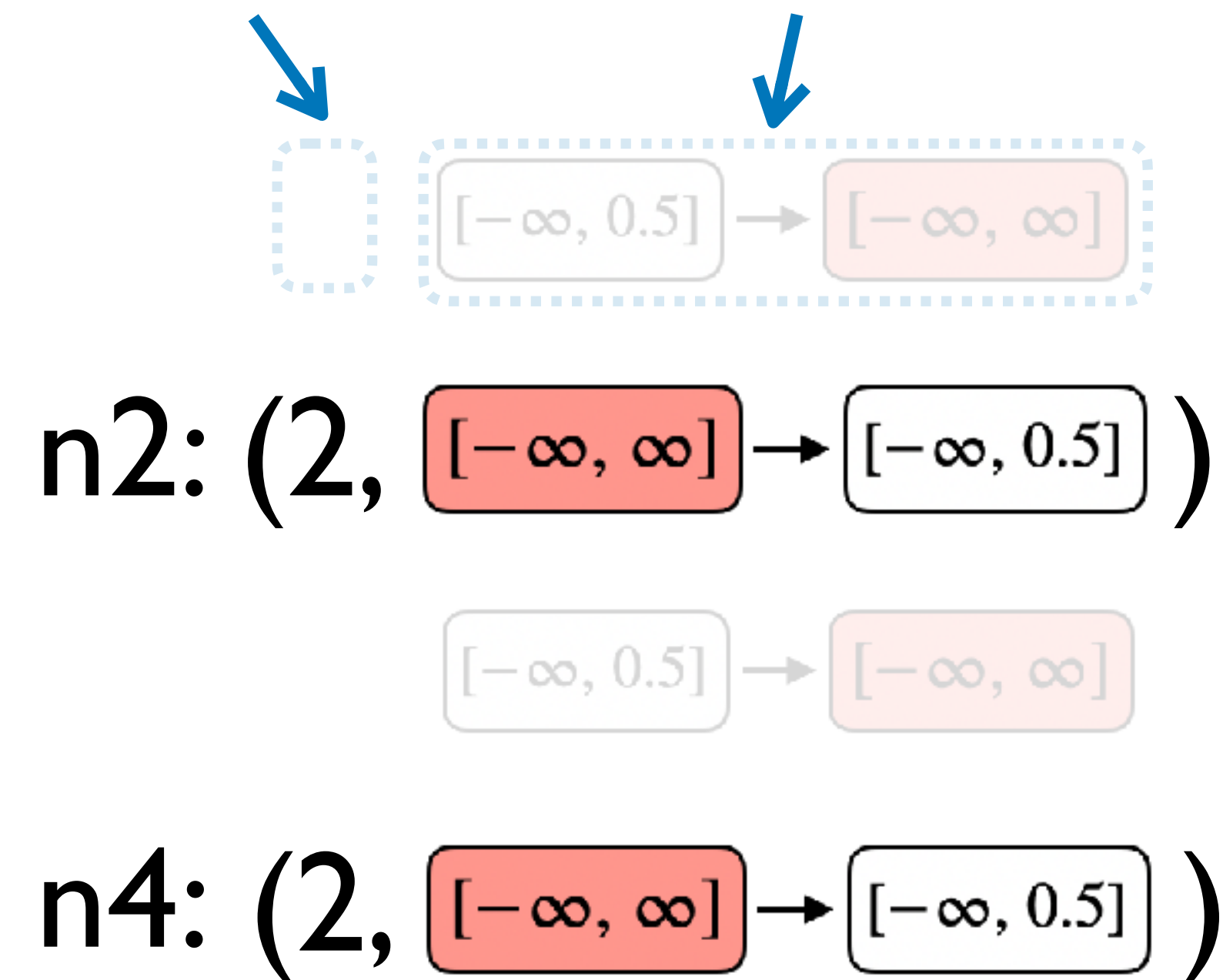


Prediction Explanation

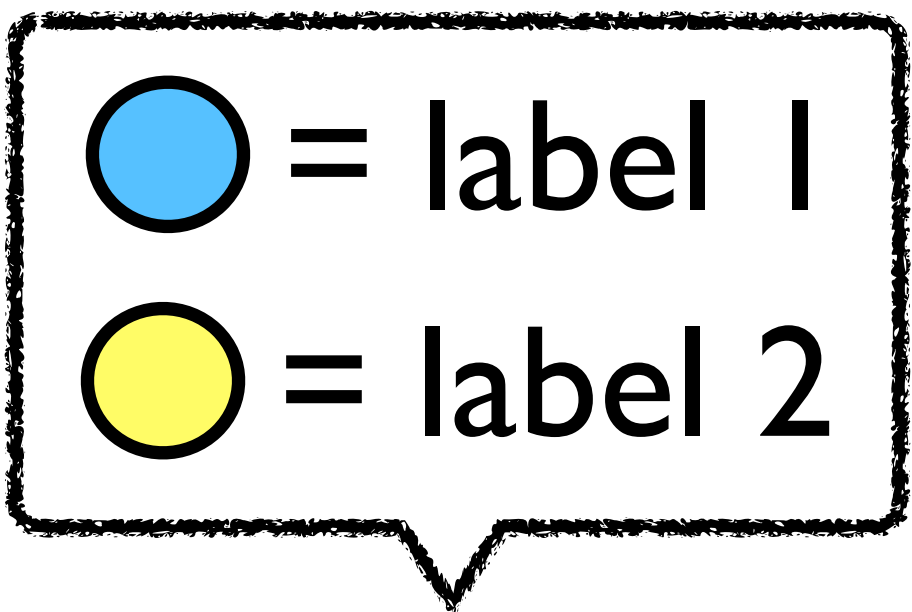




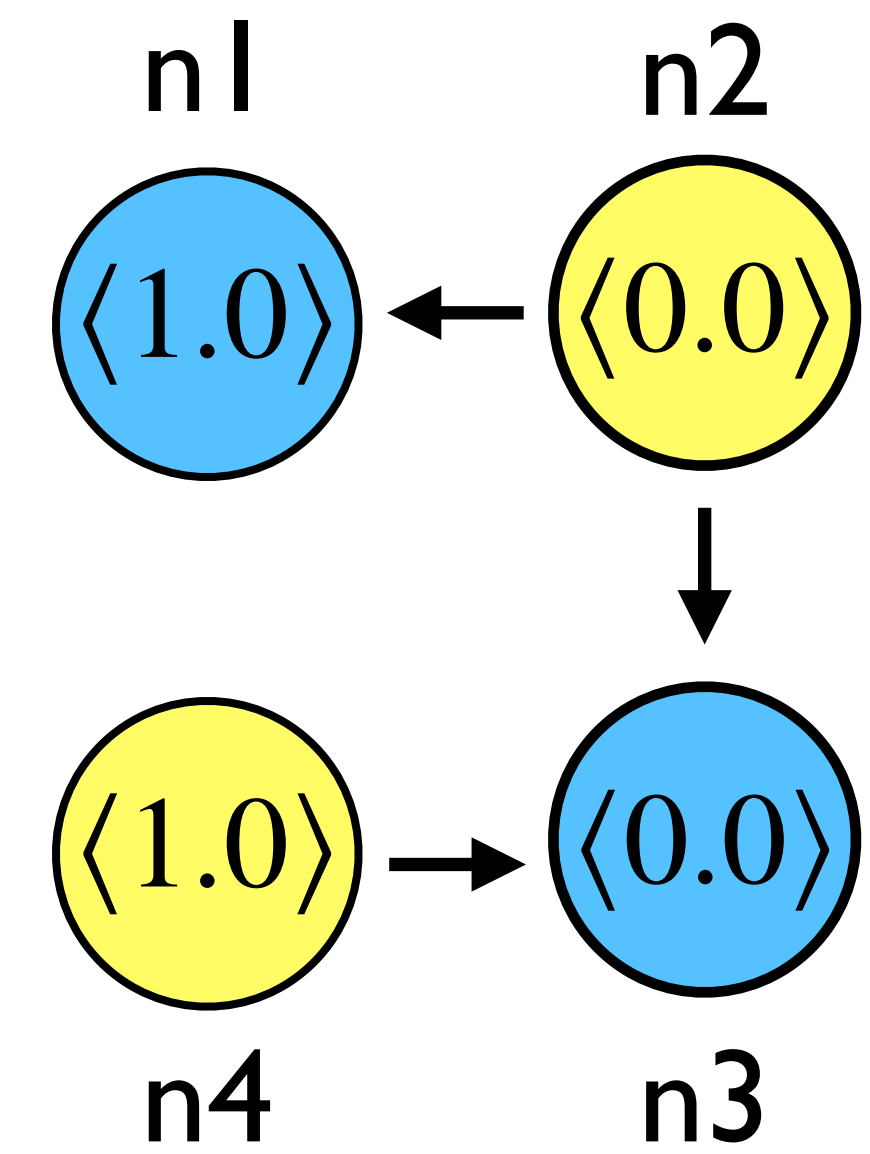
Prediction Explanation



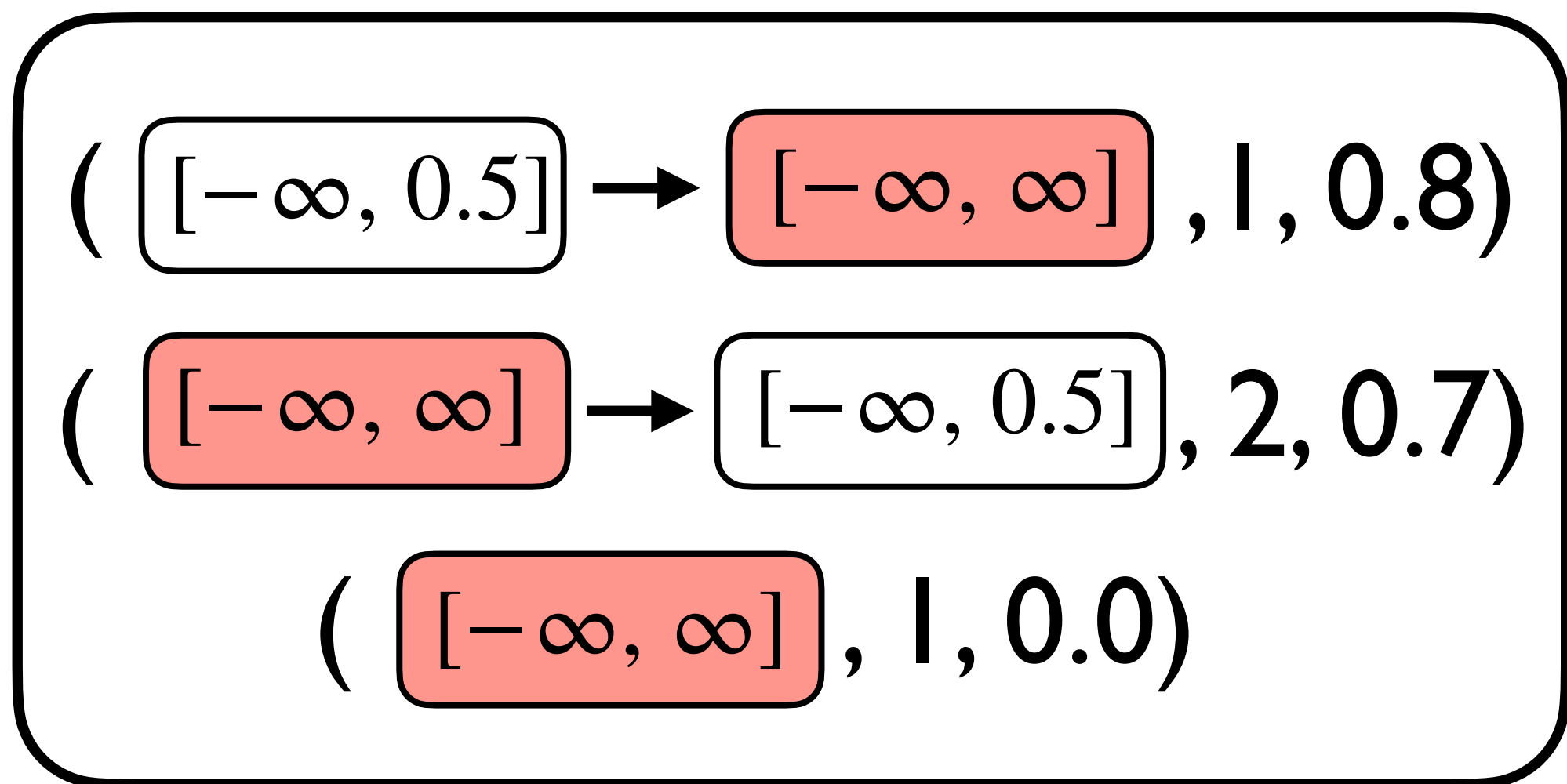
Model classifies nodes with a better scored one



- No additional explanation cost
- Explanations are guaranteed to be correct

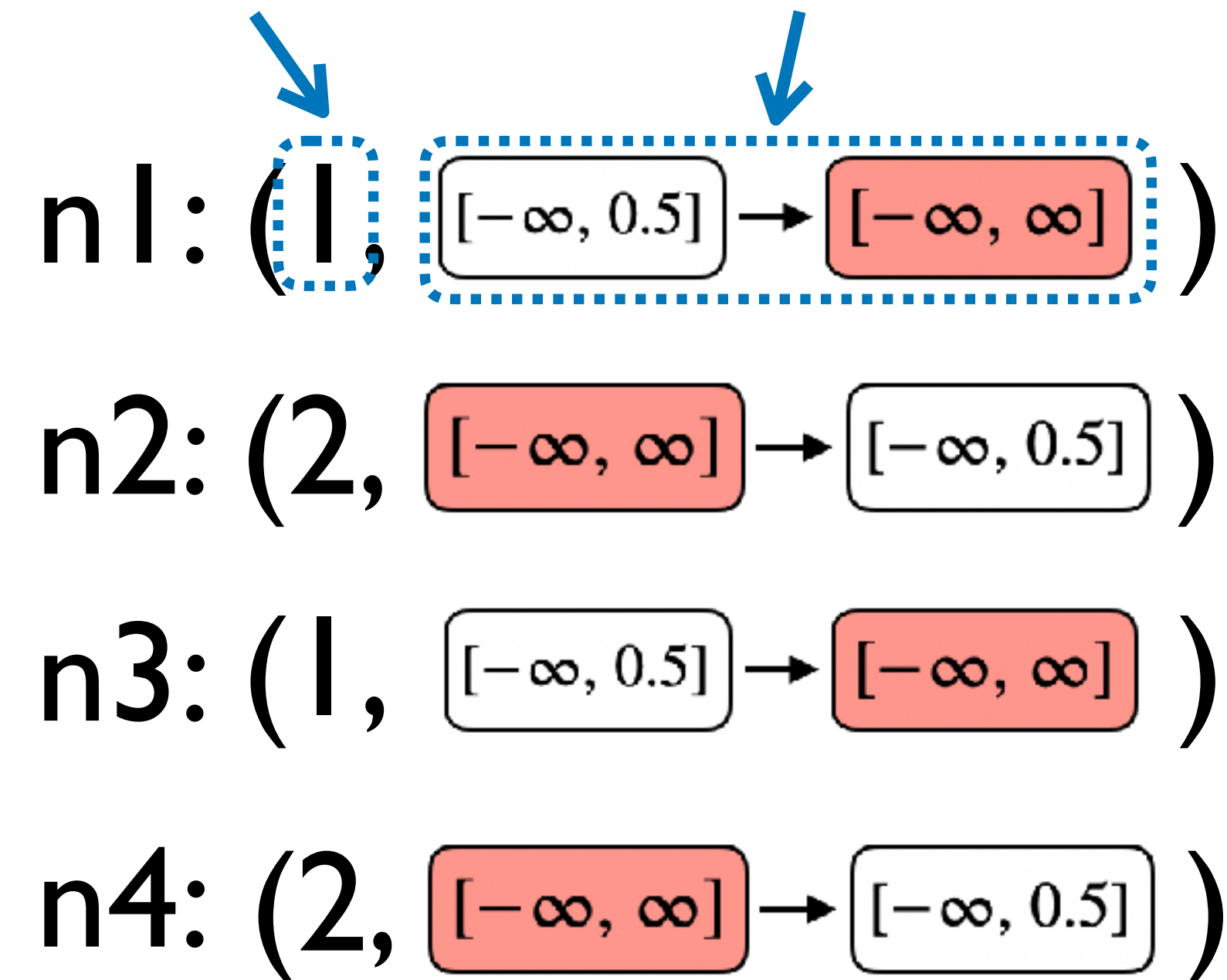


Graph data



Our model

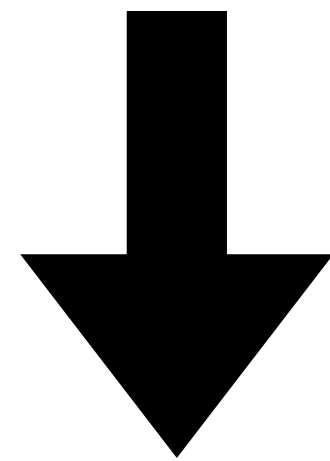
Prediction Explanation



Classification result

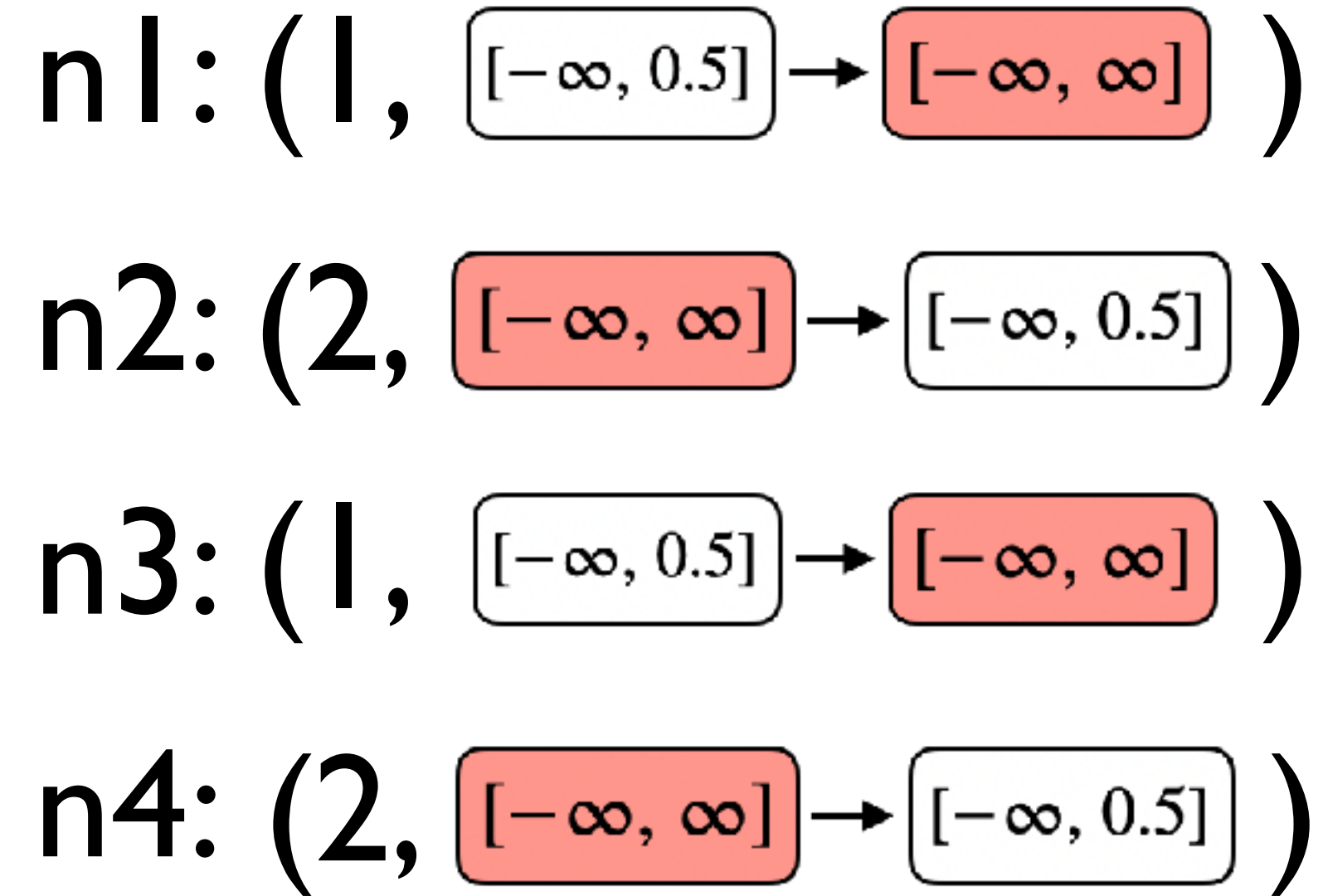
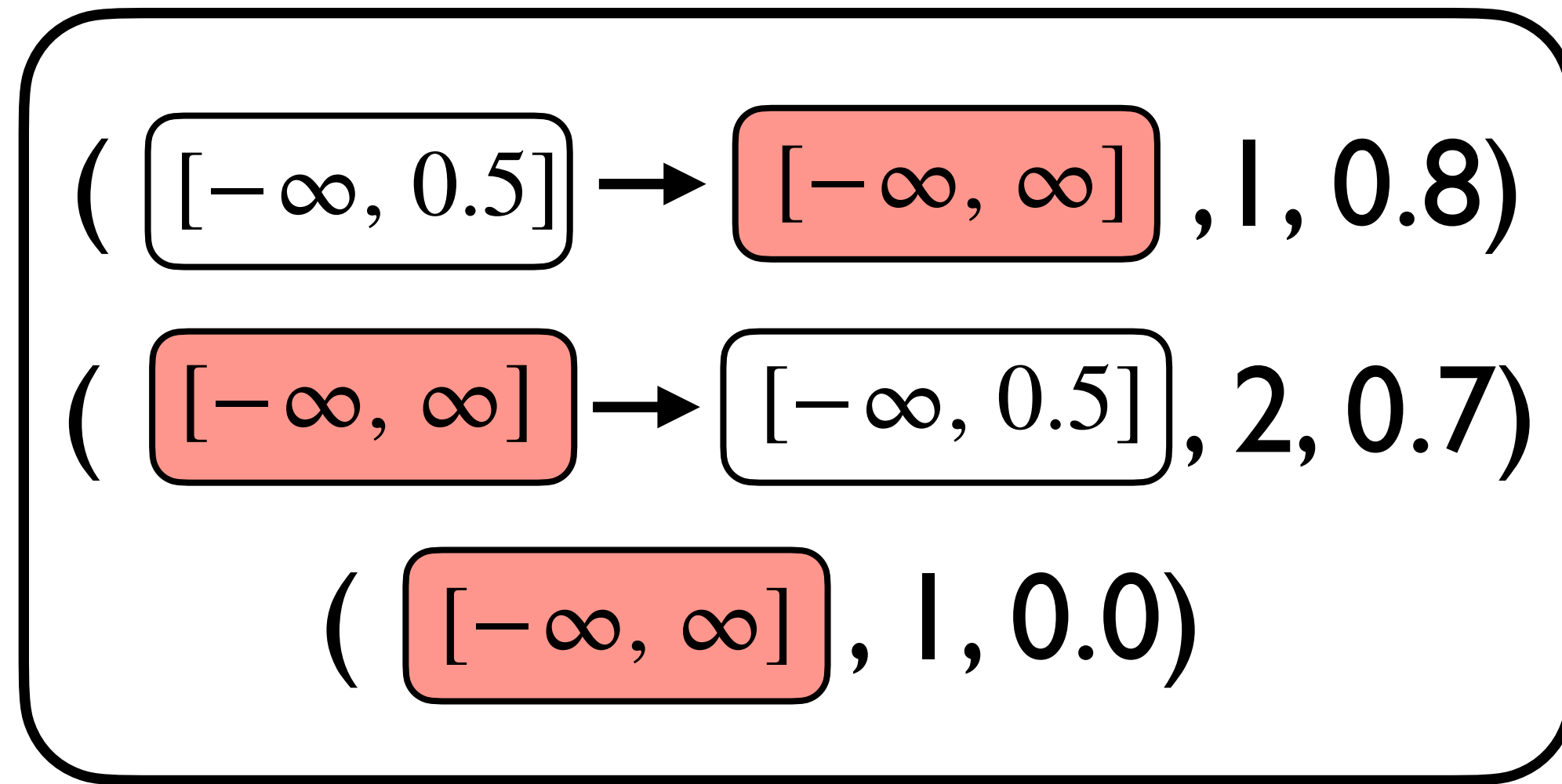
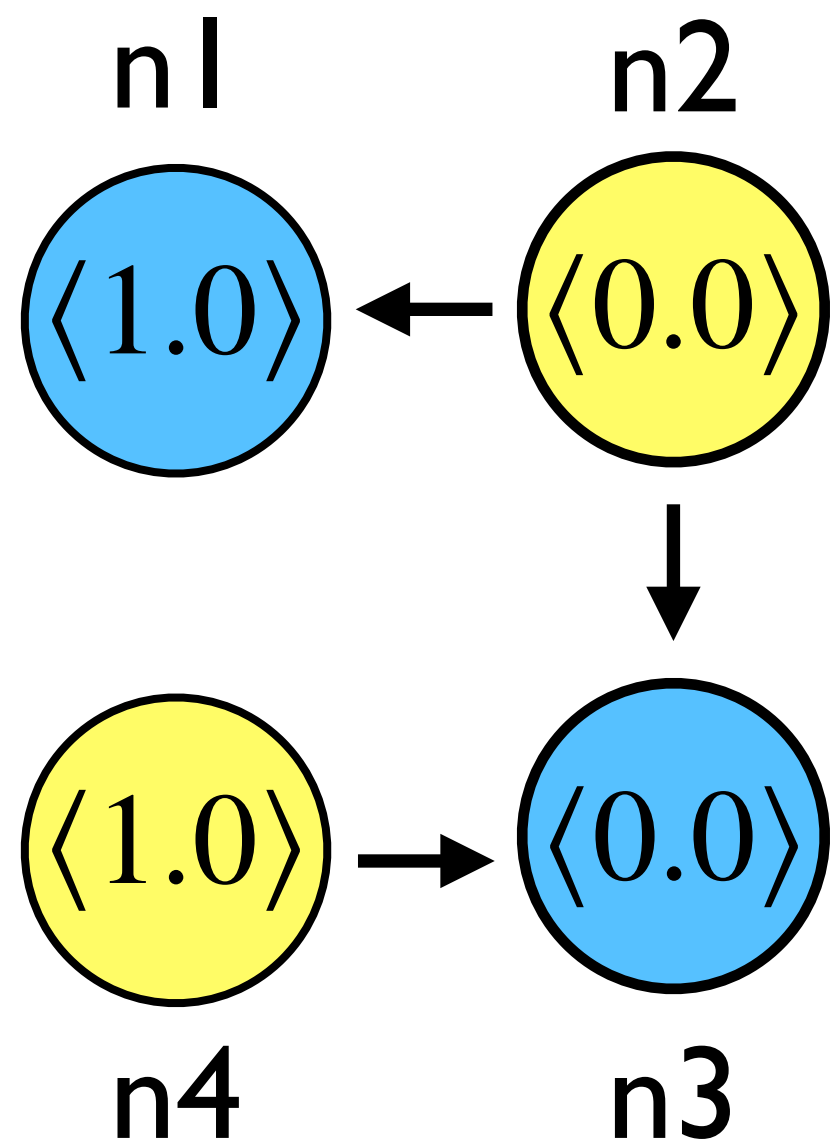


Training data



Our learning algorithm

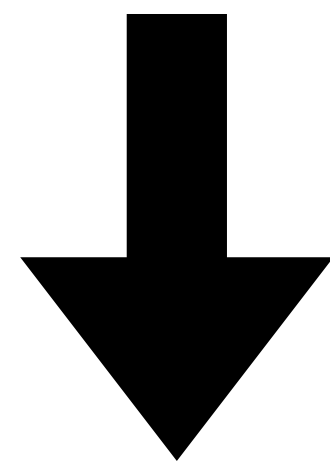
Learning objective:
Generate high-quality programs





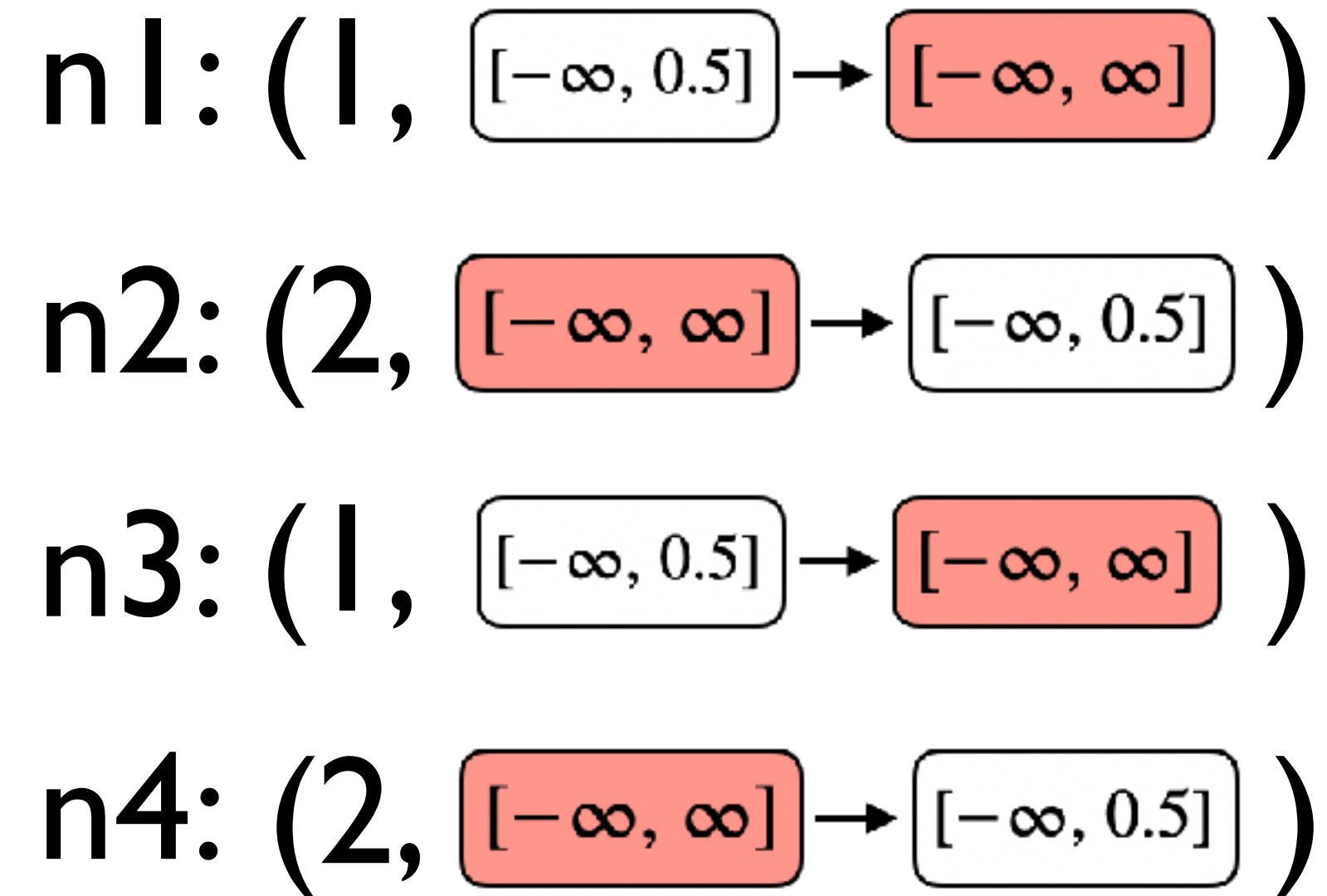
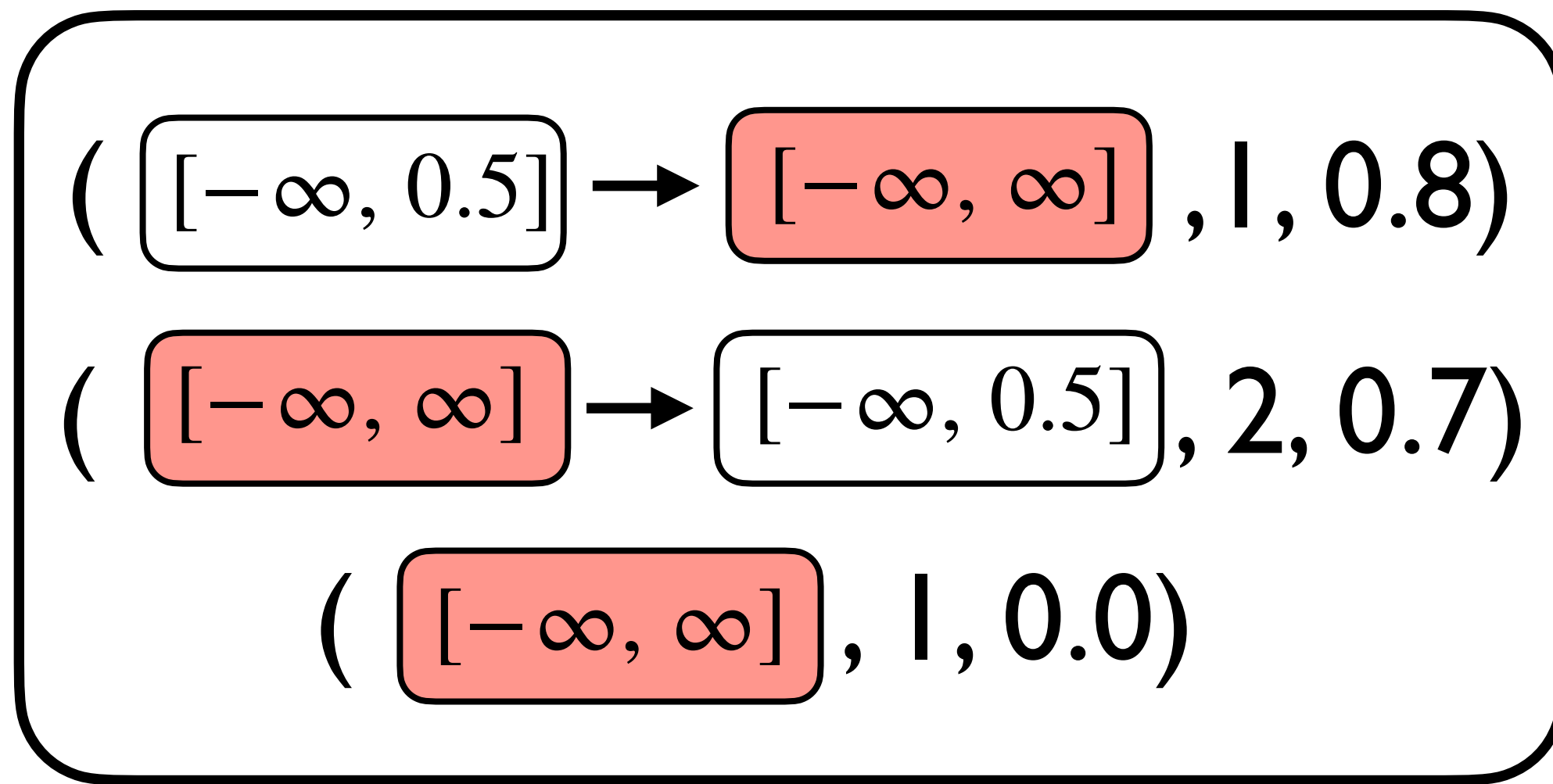
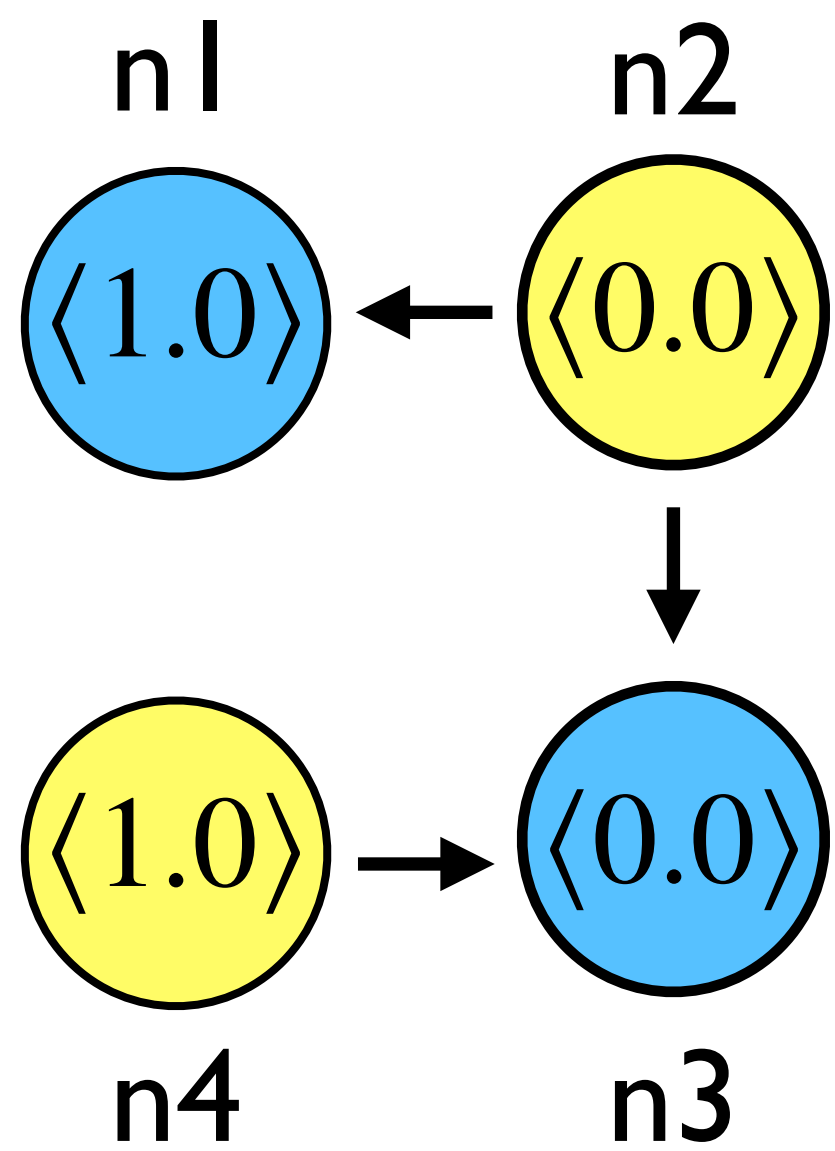
Training data

Top-down learning algorithm
Bottom-up learning algorithm



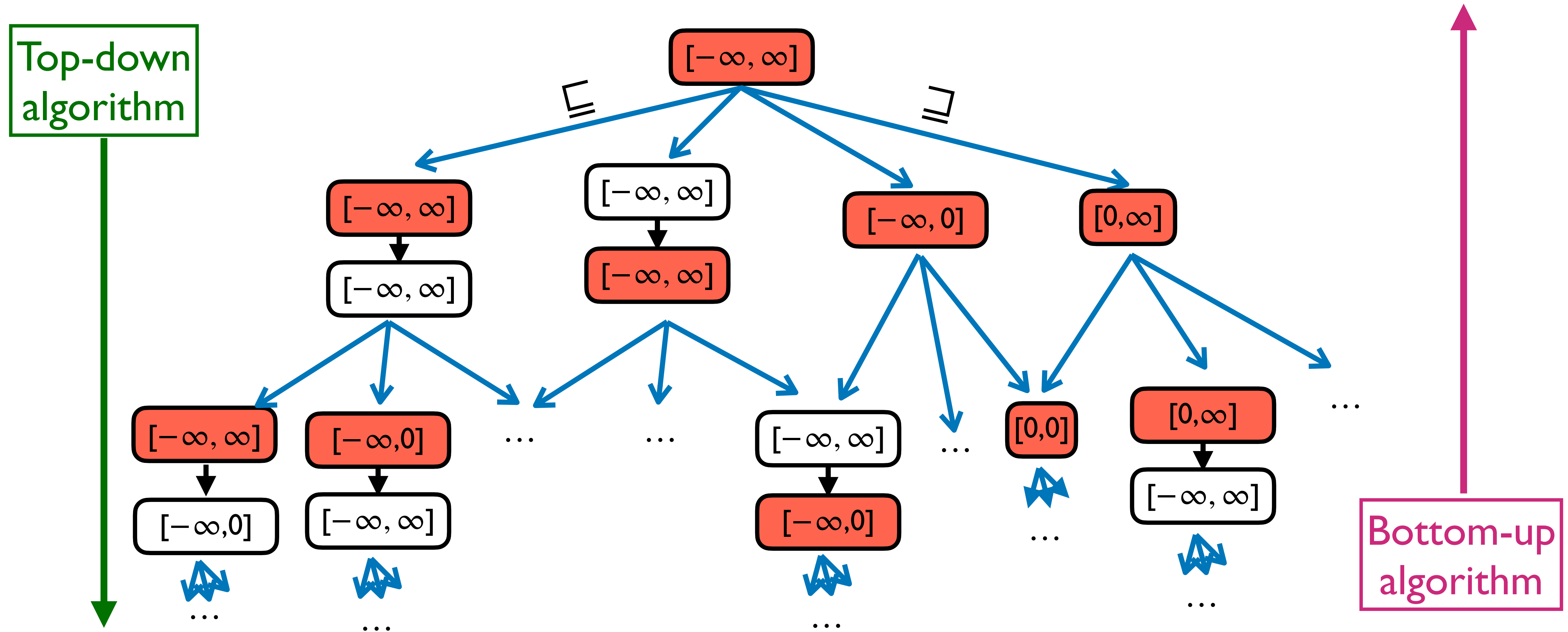
Our learning algorithm

Learning objective:
Generate high-quality programs

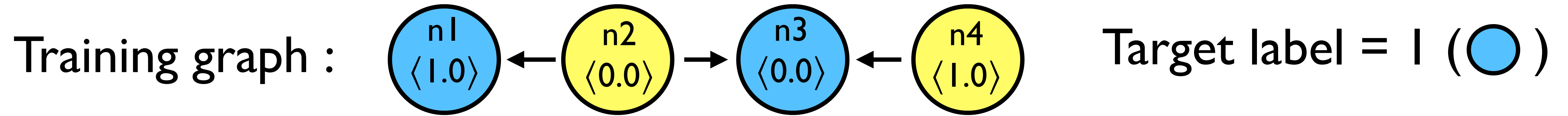


Orders Between programs

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



Top-down Learning Algorithm



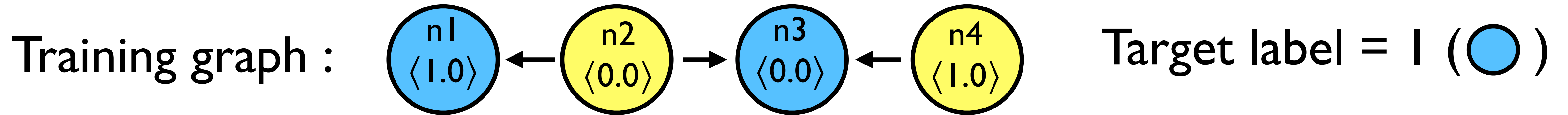
(I) Starts from the most general program

$[-\infty, \infty]$

Score : 0.5

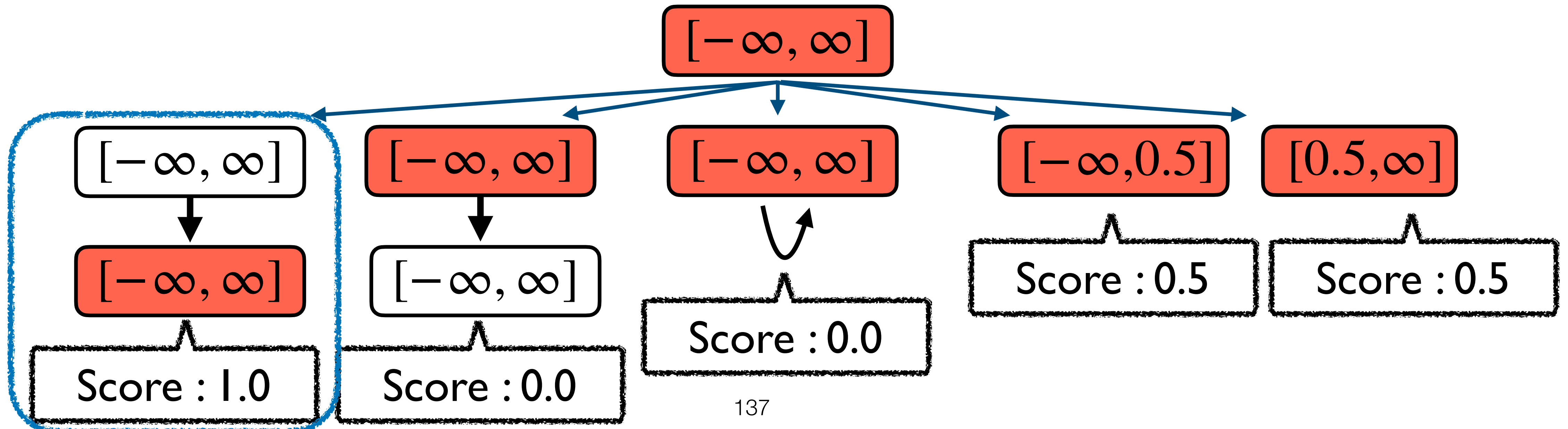
$$\frac{|\{n1, n3\}|}{|\{n1, n2, n3, n4\}|}$$

Top-down Learning Algorithm

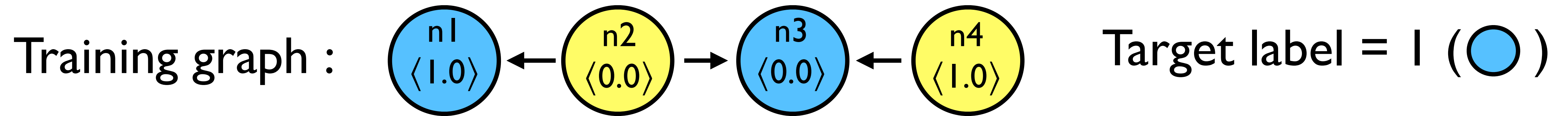


(1) Starts from the most general program $[-\infty, \infty]$ Score : 0.5

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

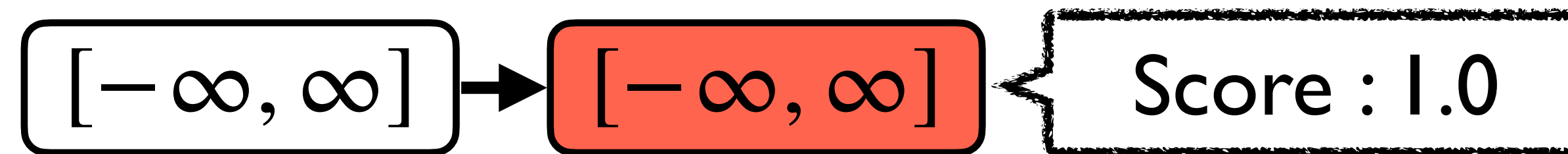


Top-down Learning Algorithm



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

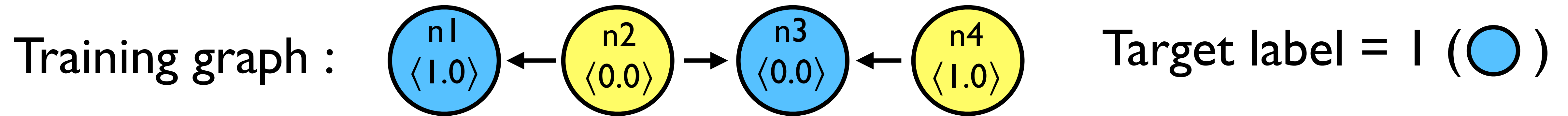
(2) Enumerate possible specified programs and choose a better scored one.



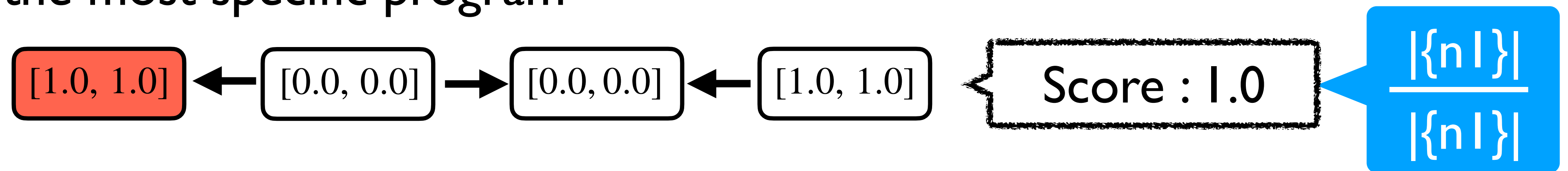
(3) Repeat (2) until no better program is enumerated

(4) Return the current program $([-\infty, \infty] \rightarrow [-\infty, \infty], 1, 1.0)$

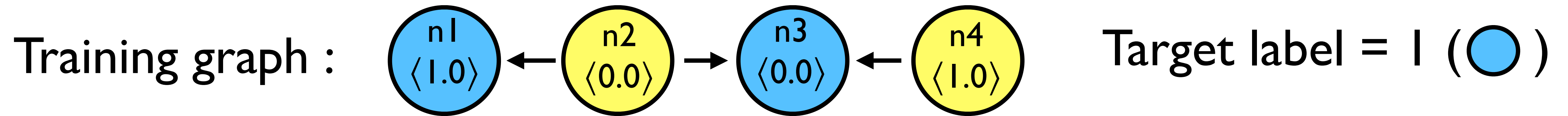
Bottom-up Learning Algorithm



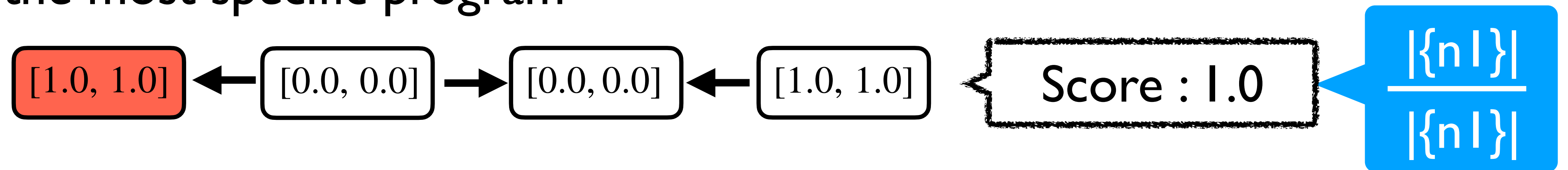
(1) Starts from the most specific program



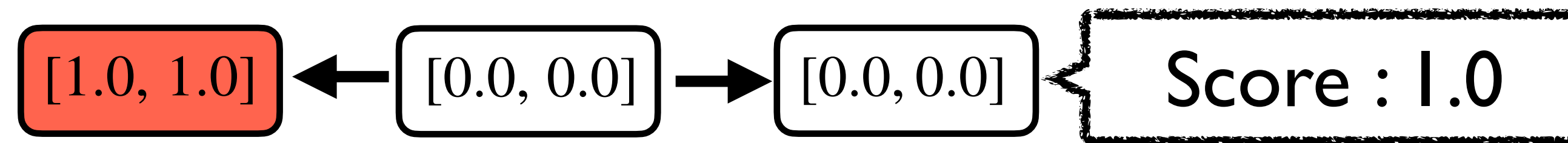
Bottom-up Learning Algorithm



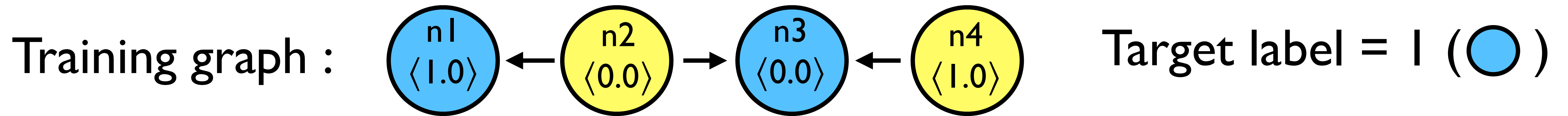
(1) Starts from the most specific program



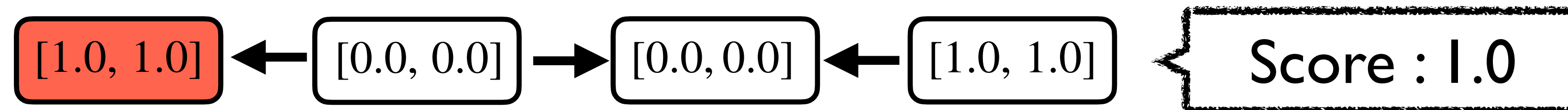
(2) Enumerate possible generalized programs and choose an equal or better scored one



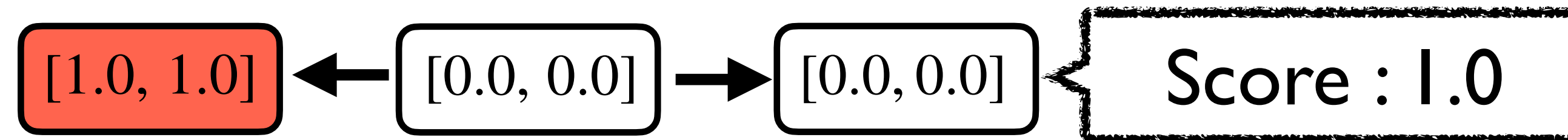
Bottom-up Learning Algorithm



(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 ([-∞, ∞] → [-∞, ∞] , 1, 1.0)

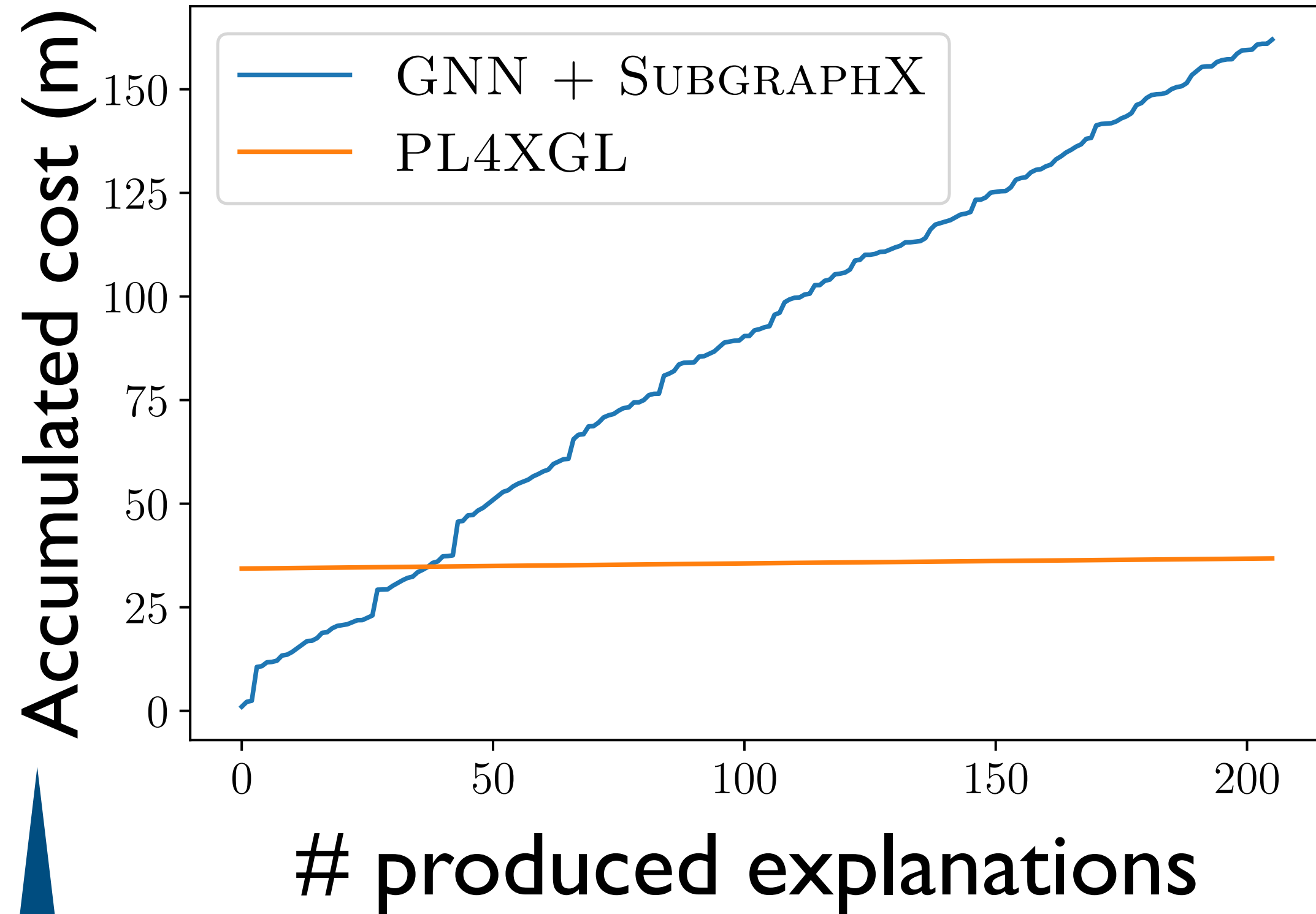
Accuracy Comparison

- 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

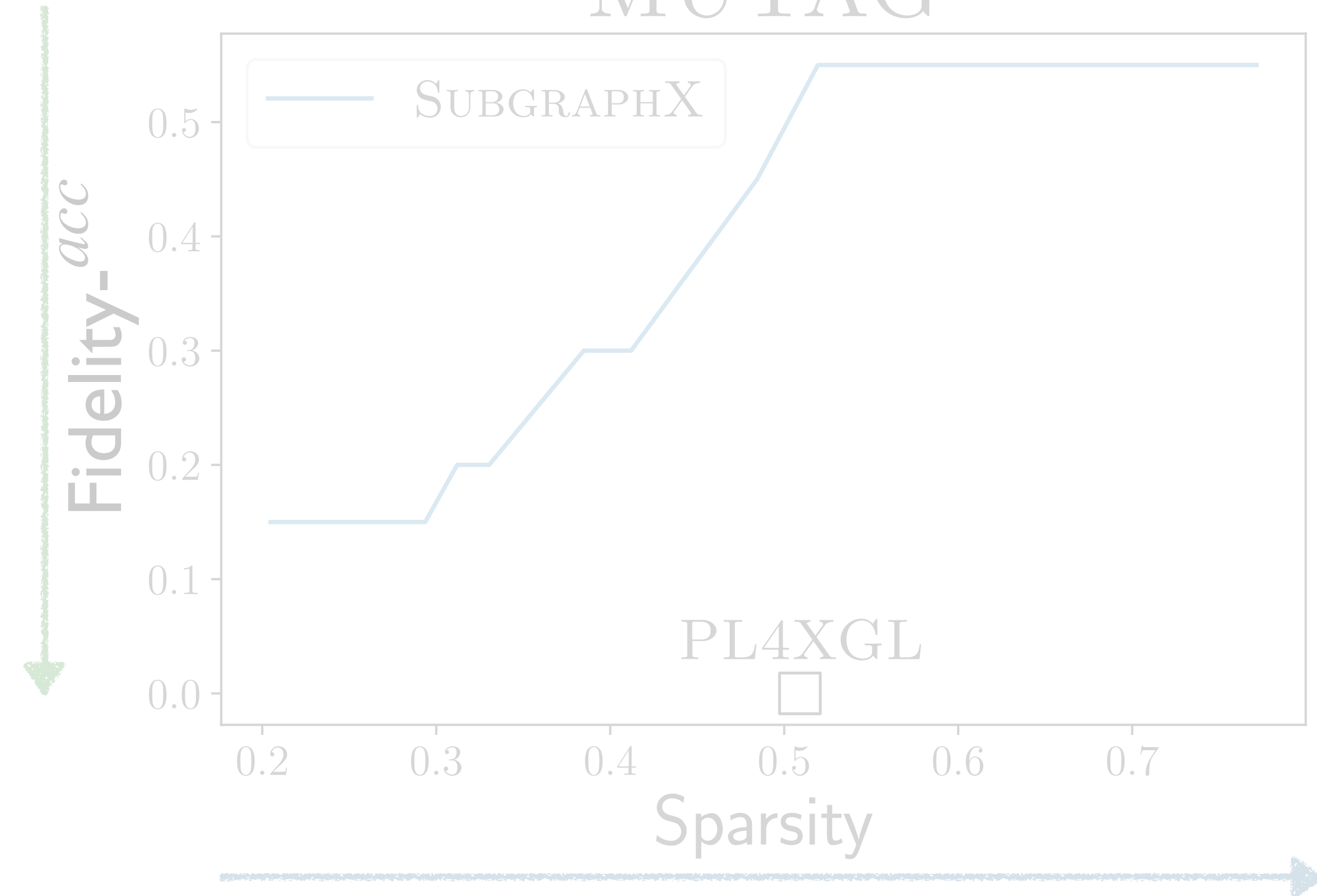
Cost comparison

BBBP



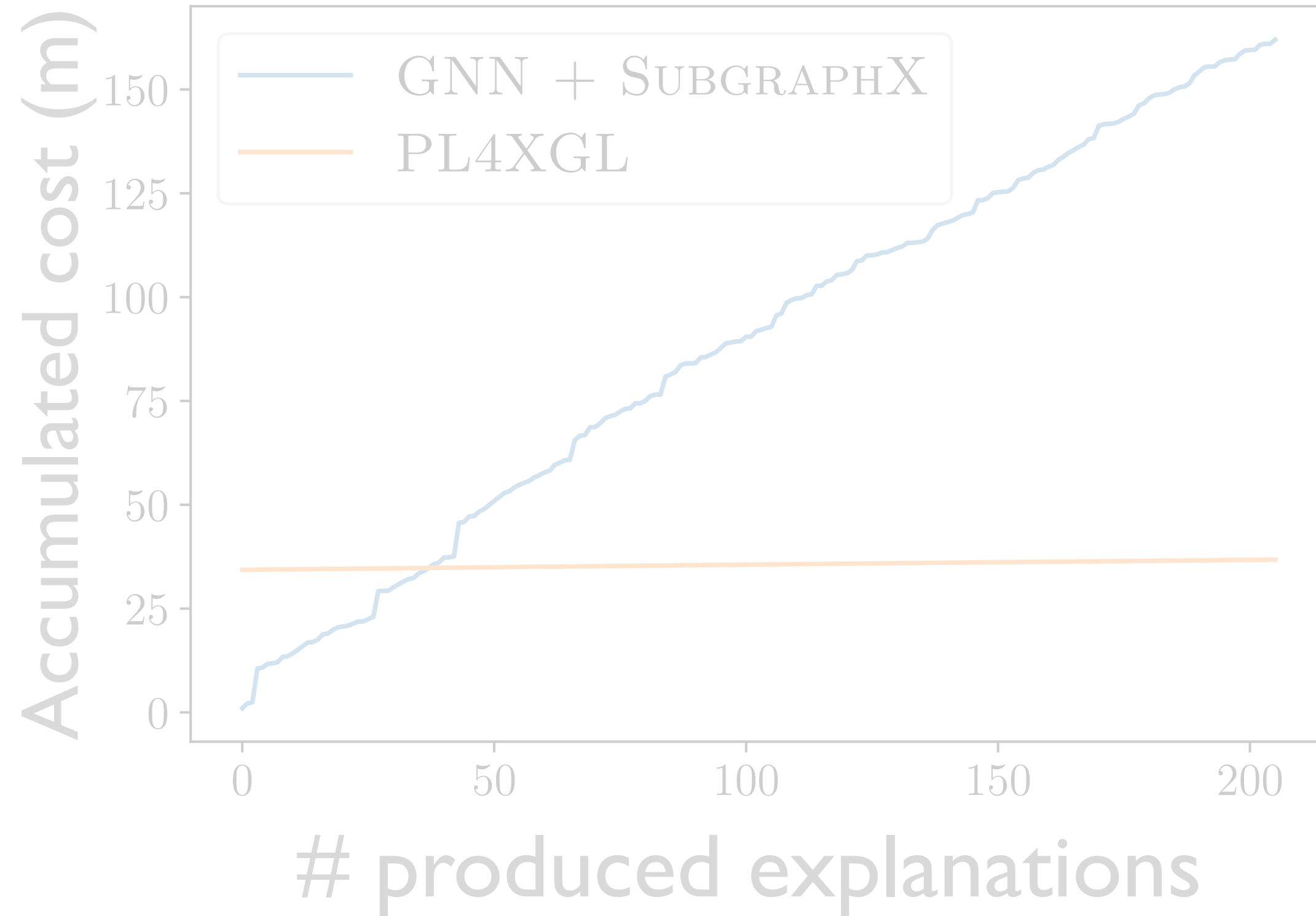
Training + prediction + explanation cost

MUTAG

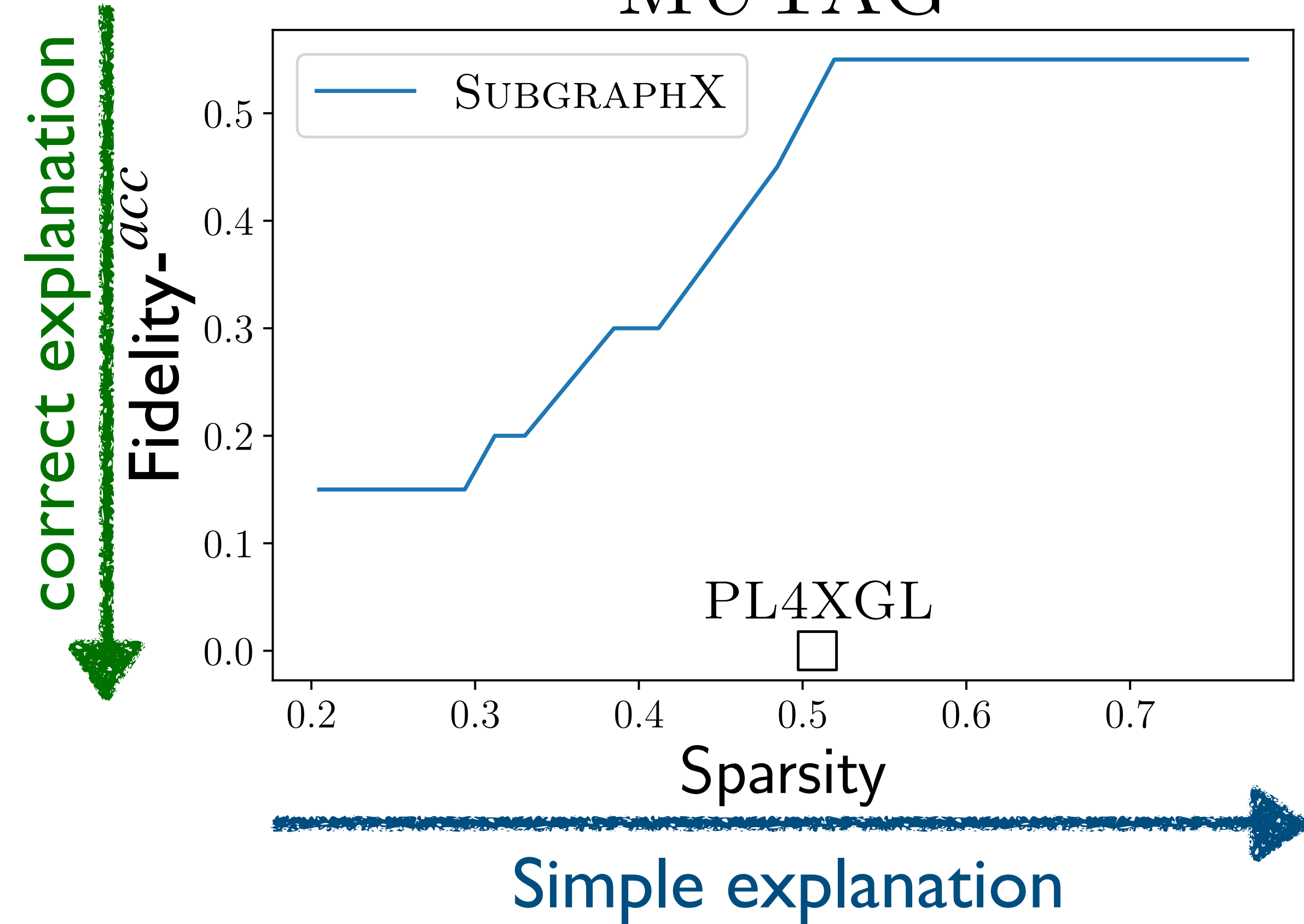


Explainability comparison

BBBP

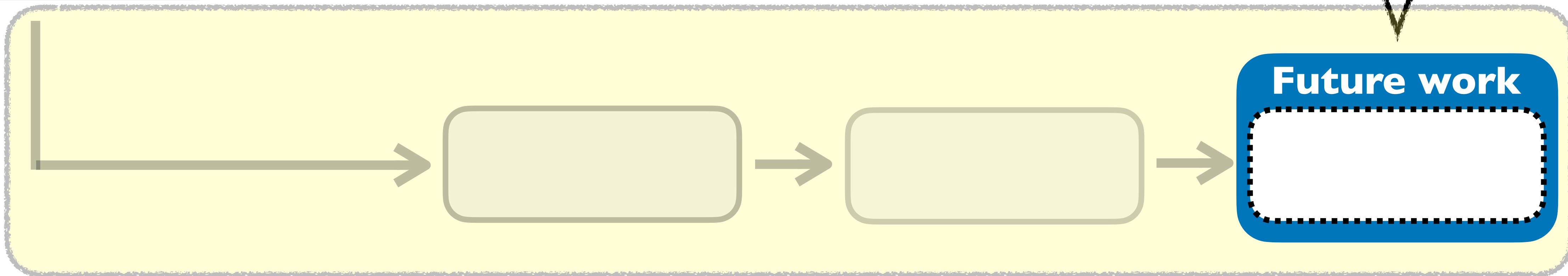


MUTAG



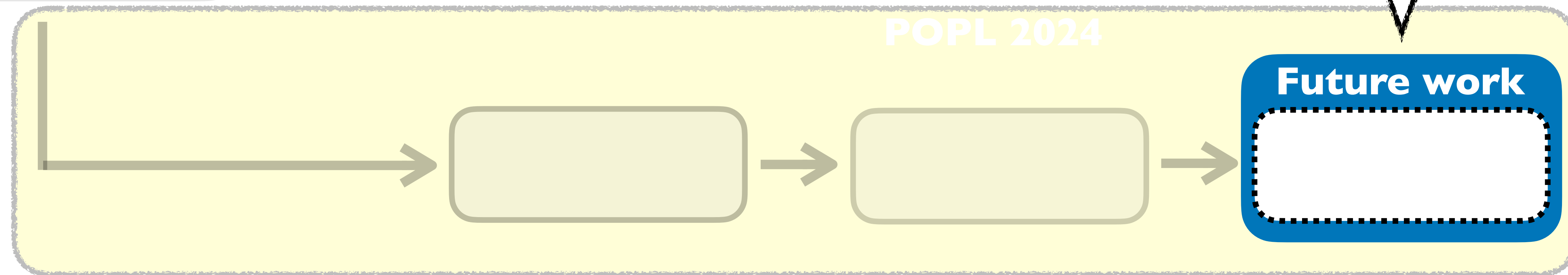
I. Improving our approach

- 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

Graphs of training programs

Static Analyzer

Graphick

Automatically generated feature

Apply 2-obj: { $[0, \infty], [0, 7] \rightarrow [9, 11], [0, \infty] \rightarrow [76, \infty], [0, \infty] \rightarrow [0, \infty], [43, \infty] \rightarrow [0, \infty], [0, 14]$, ... } 68 features

Apply 2-type: { $[105, 155], [0, \infty] \rightarrow [0, \infty], [0, 61] \rightarrow [60, 76], [0, 61] \rightarrow [0, 22], [0, \infty]$, ... } 29 features

Apply 1-type: { $[0, \infty], [61, \infty] \rightarrow [46, \infty], [0, \infty] \rightarrow [0, \infty], [100, \infty] \rightarrow [0, \infty], [29, \infty]$, ... } 100 features

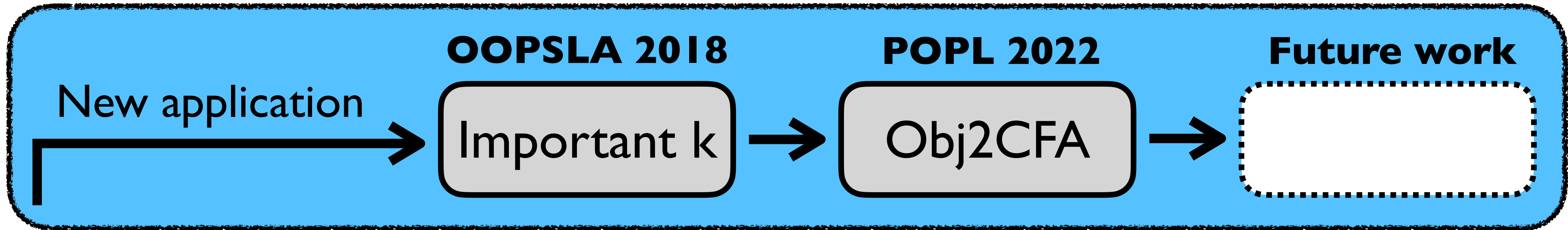
Automatically generated graph-based context sensitivity heuristic

Future work

OOPSLA 2020

Graphick

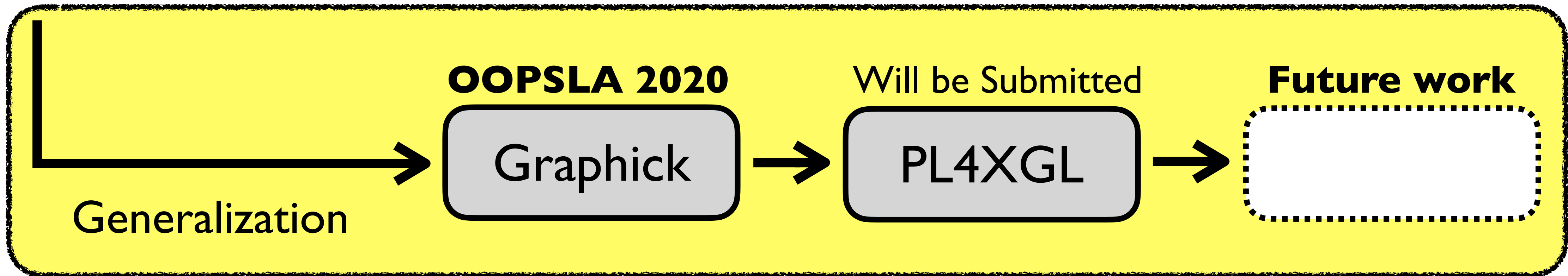
Establishing **important k** as a standard



OOPSLA 2017

Disjunctive model & Learning algorithm

Establishing **a new graph machine learning** method



Thank you!

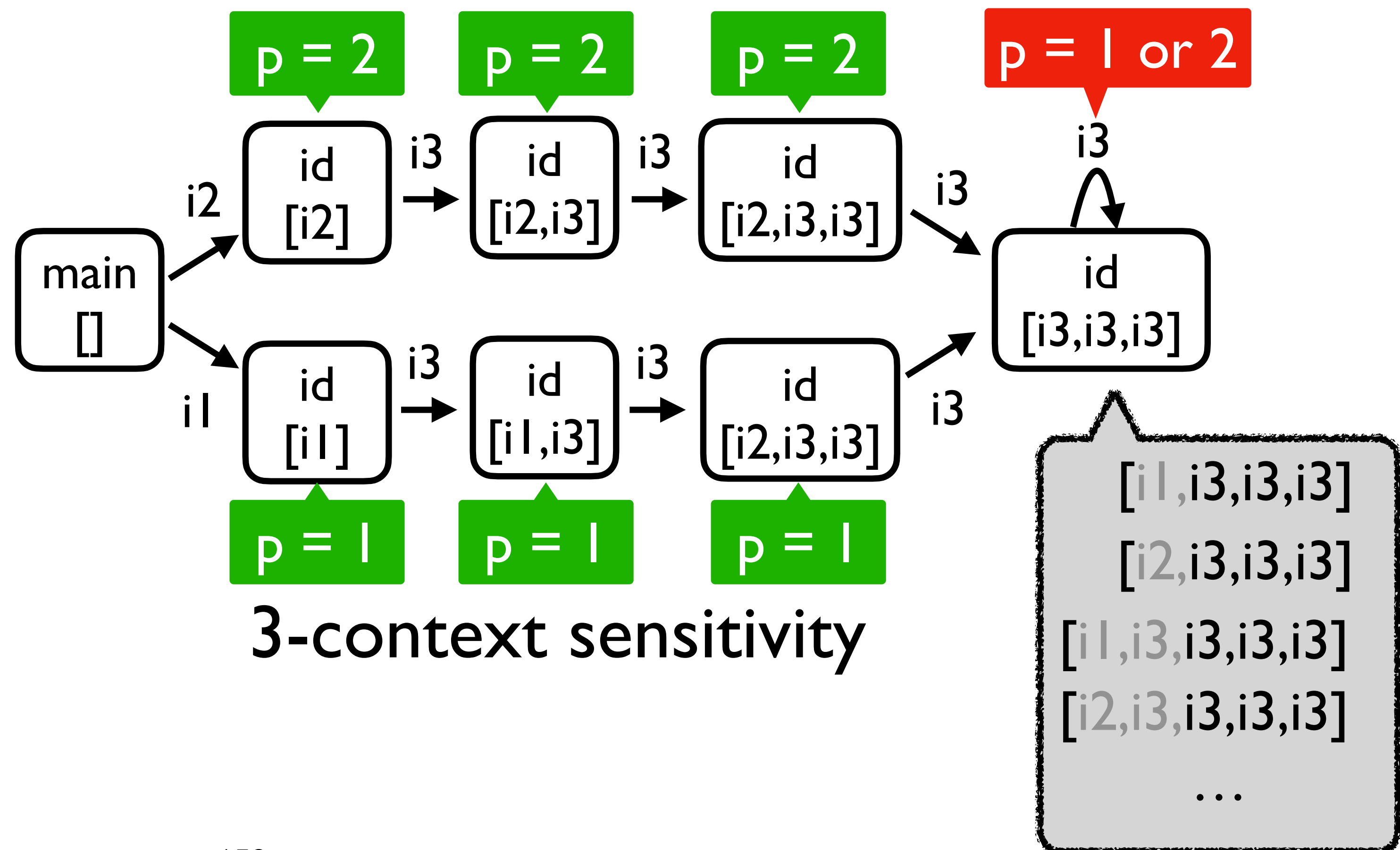
Backup Slides

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
```



Problem of Last-k

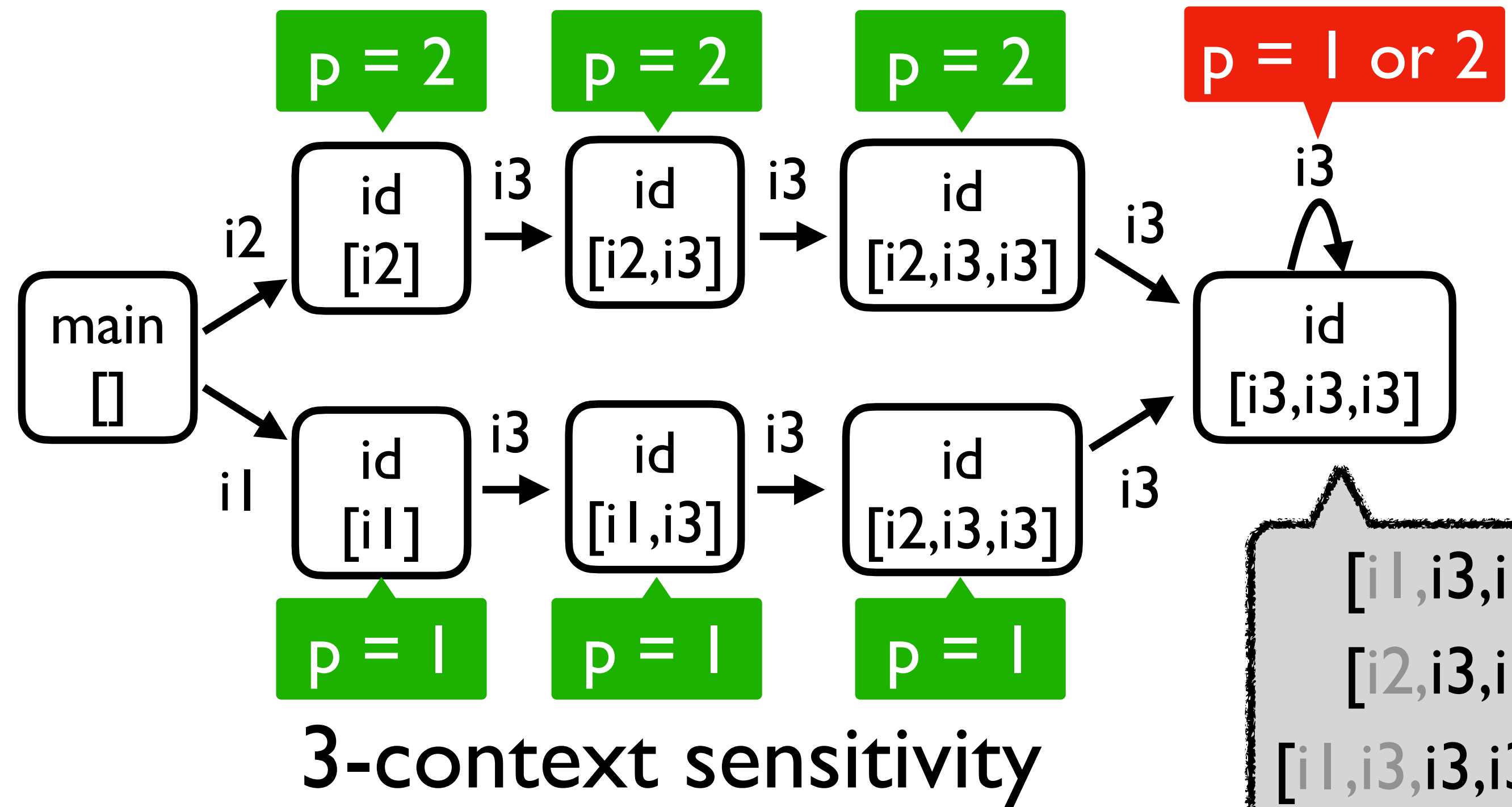
- An example showing a **problem of last-k** context abstraction

```

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
  }
  
```



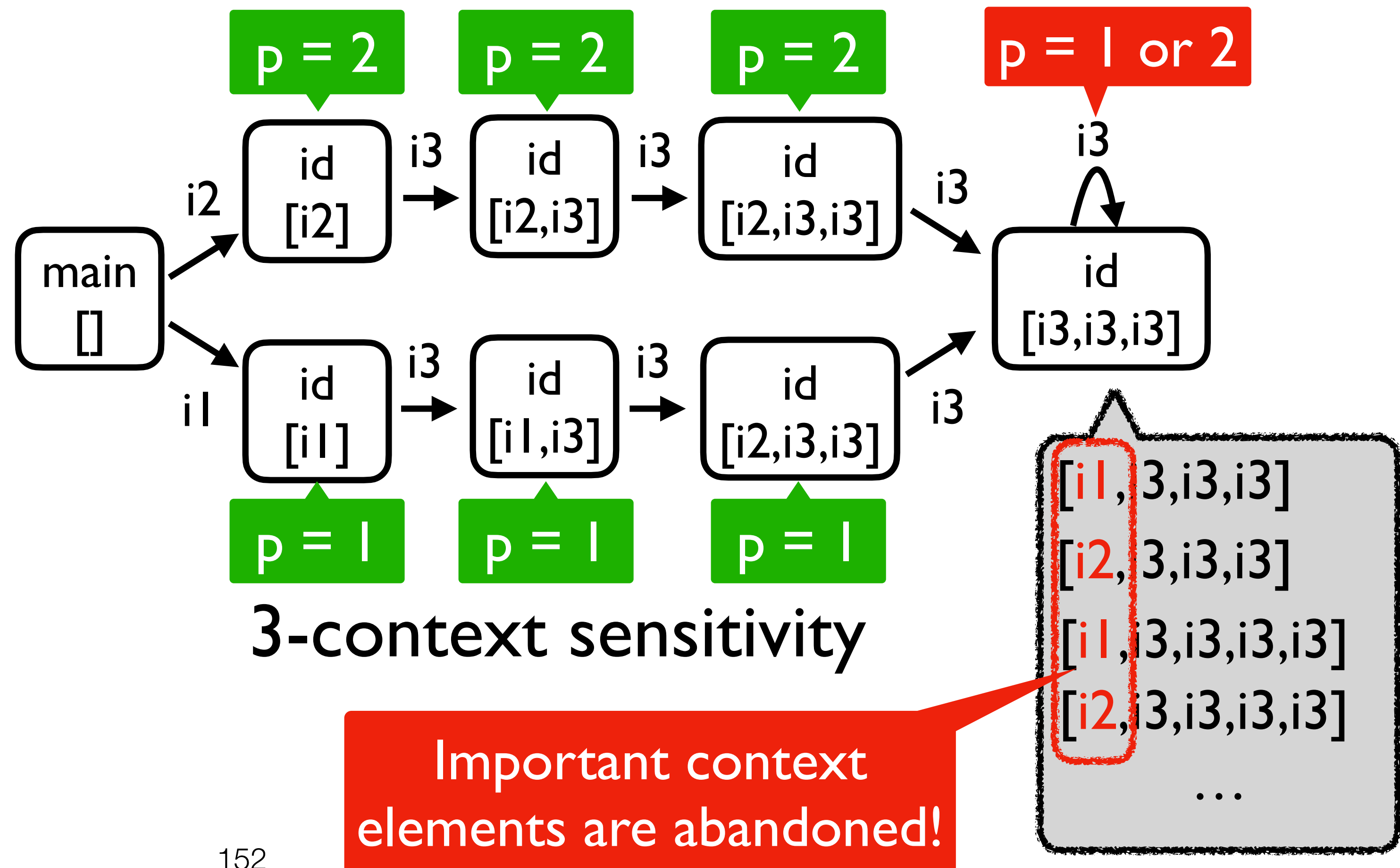
Unable to proved the query!
 (v1 = 1 or 2 & v2 = 1 or 2)

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
```

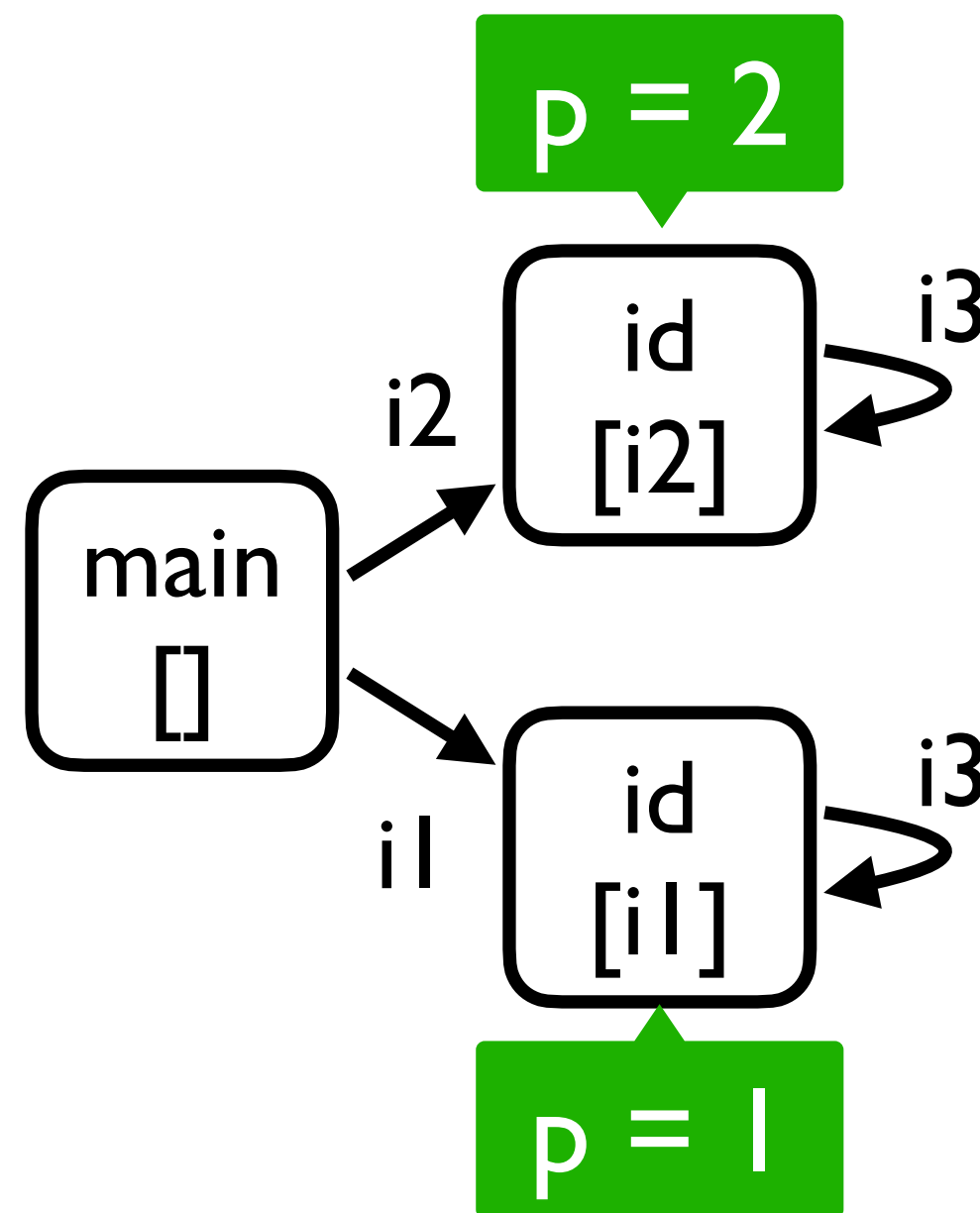


Our Solution: Keep Important K

- In **important k**, l-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
```



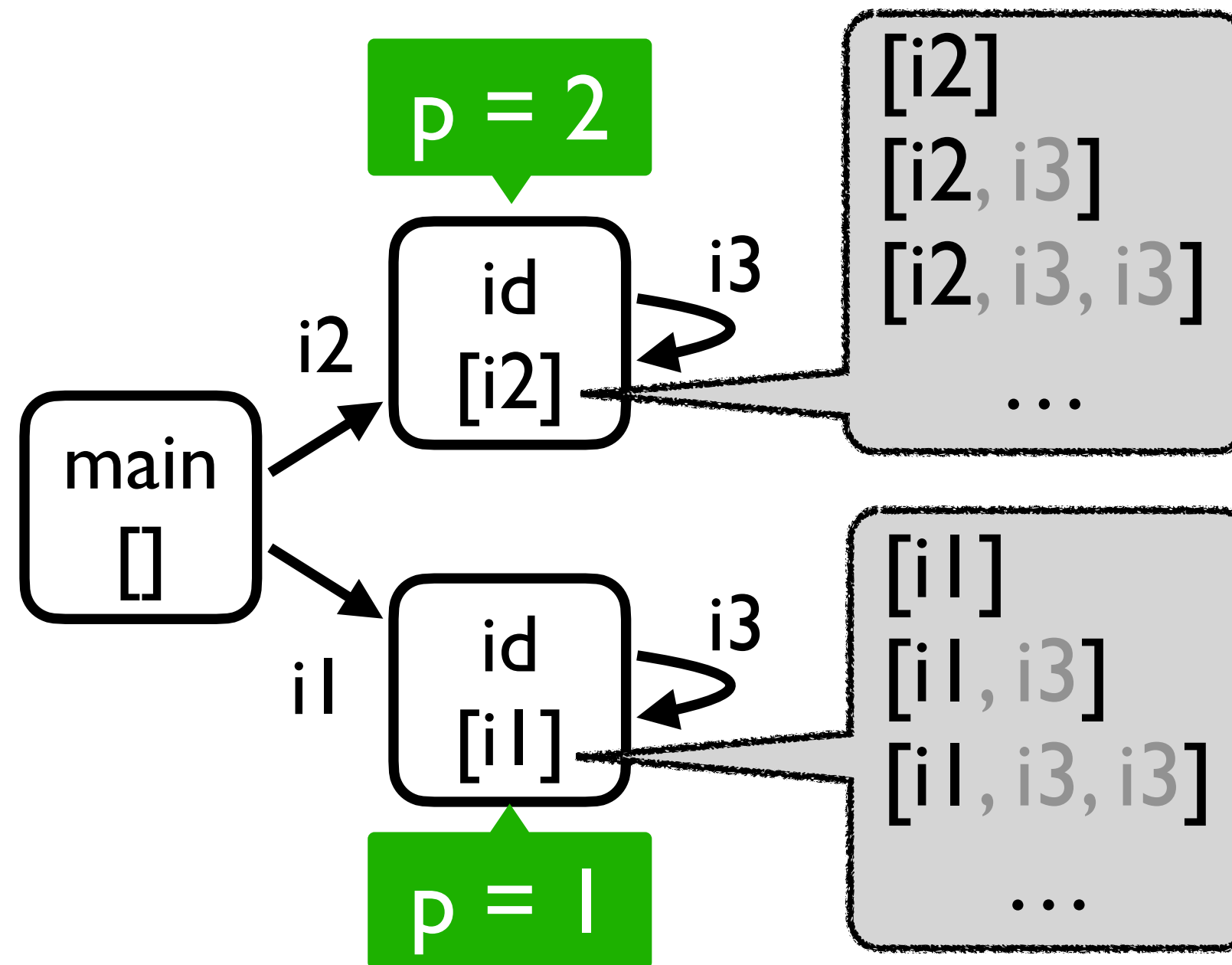
l-context sensitivity

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

Our Solution: Keep Important K

- In **important k**, I-ctx sensitivity proves the query

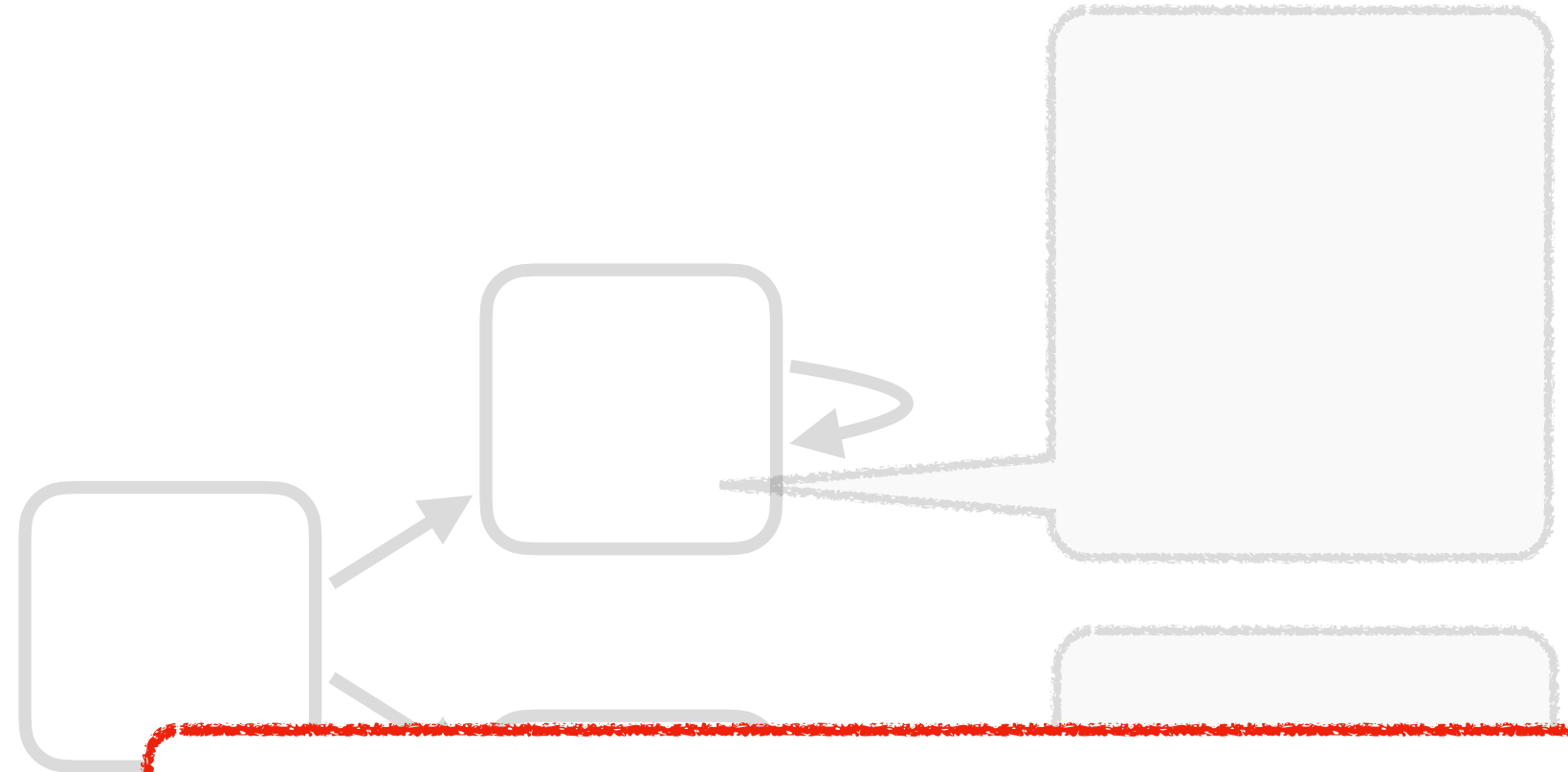
```
id(p, i){  
  if i > 0:  
    return id(p, i-1); //i3  
  else:  
    return p;}  
  
main(){  
  i = input();  
  v2 = id(2,i); //i2  
  v1 = id(1,i); //i1  
  assert (v1 != v2); //query
```



I-context sensitivity

Proved the query!
(v1 = 2 & v2 = 1)
154

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

Jeongsu Lee
Korea University
Republic of Korea
vclcano0999@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 LIVEN, 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, Jeongsu Lee, and Hakjoo Oh. 2020. SAVER: Scalable, Precise, and Safe Memory-Error Repair. In *42nd International Conference on Software Engineering (ICSE '20)*, May 22–29, 2020, Seoul, Republic of Korea. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/3377311.3380323>

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 long-term goal of our research is to

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICSE '20, May 22–29, 2020, Seoul, Republic of Korea.
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4902-7121-6/20/05...\$15.00
<https://doi.org/10.1145/3377311.3380323>

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 43% 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 Assignments. In *Proceedings of the 25th 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 SAURON [43], assume a solution exists that is equivalent to the incorrect program modulo control flow. This assumption, however, does not hold always [22], especially when providing feedback beyond introductory-level exercises. In this case, constructing a corpus with the close-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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ESEC/FSE 21, August 23–28, 2021, Athens, Greece.
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-4562-6/21/08...\$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) $\mathcal{H} = \langle f_{2ctx}, f_{1ctx} \rangle$ that

- minimizes analysis cost while is precise enough

- User-provided precision constraint
- E.g., maintain 90% precision of 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

$$f_{2ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_6 \wedge \neg a_8 \wedge \dots)$$
$$f_{1ctx} = (a_1 \wedge \neg a_3 \wedge \neg a_4 \wedge \dots)$$

Learned Pattern Used In Manually Crafted Heuristics

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)

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

Yue Li Aarhus University
Tian Tan Aarhus University
Anders Møller Aarhus University
Yannis Smaragdakis University of Athens

ABSTRACT

Context-sensitivity is important in pointer analysis to ensure high precision, but existing techniques suffer from unpredictable scalability. Many variants of context-sensitivity exist, and it is difficult to choose one that leads to reasonable analysis time and obtains high precision, without running the analysis multiple times.

We present the SCALER framework that addresses this problem. SCALER efficiently estimates the amount of points-to information that would be needed to analyze each method with different variants of context-sensitivity. It then selects an appropriate variant for each method so that the total amount of points-to information is bounded, while utilizing the available space to maximize precision.

Our experimental results demonstrate that SCALER achieves predictable scalability for all the evaluated programs (e.g., spreadsheets can reach 10x for 2-object-sensitivity), while providing a precision that matches or even exceeds that of the best alternative techniques.

CCS CONCEPTS

Theory of computation → Program analysis.

KEYWORDS

static analysis, points-to analysis, Java

ACM Reference Format

Yue Li, Tian Tan, Anders Møller, and Yannis Smaragdakis. 2018. Scalability-First Pointer Analysis with Self-Tuning Context-Sensitivity. In *Proceedings of the 2018 ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18)*, November 4–9, 2018, Lake Buena Vista, FL, USA. ACM, New York, NY, USA, 22 pages. <https://doi.org/10.1145/3236228.3236641>

1 INTRODUCTION

Pointer analysis is a family of static analysis techniques that provide a foundation for many other analysis and software engineering tasks, such as program slicing [36, 39], reflection analysis [19, 31], bug detection [13, 26], security analysis [1, 23], program verification [8, 27], and program debugging and comprehension [5, 21]. The goal of pointer analysis is to statically compute a set of objects (abstracted as their allocation sites) that a program variable may point to during run time. Although stating this goal is simple, it is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advertising, and the copyright notice and the full title on the first page. Copyright for this work owned by others than the author(s) must be retained. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org. *ESEC/FSE '18*, November 4–9, 2018, Lake Buena Vista, FL, USA. © 2018 Copyright held by the owner(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-5149-1/18/11...\$15.00. <https://doi.org/10.1145/3236228.3236641>

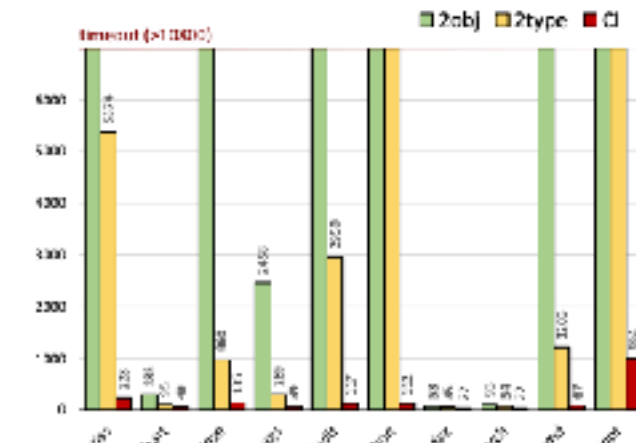


Figure 1: Comparison of running time (seconds) of 2-object sensitivity, 2-type sensitivity, and context-insensitive analyses. The y-axis is truncated to 7000 seconds for readability, and all truncated cases reach the time budget, 10800 seconds.

challenging to produce precise analysis results without sacrificing scalability [12, 30, 33]. Thus, for decades, researchers have continued to develop sophisticated pointer analysis techniques [2, 14–16, 18, 22, 24, 25, 32, 33, 37, 38].

One of the key mechanisms for achieving high analysis precision is context sensitivity, which allows each program method to be analyzed differently according to the context it is used in [17]. Context sensitivity has different variants, depending on the kind of context information used. For Java programs, object-sensitivity [25] and type-sensitivity [32] have proven to be quite effective. The former is strictly more precise but less efficient than the latter [15, 37]. However, with any available variant, although the analysis can gain in precision, scalability is known to be *unpredictable* [33, 38], in the sense that programs very similar in size and other complexity metrics may have completely different scalability profiles.

Figure 1 shows time spent analyzing 10 real-world Java programs¹ under 2-object-sensitivity (2obj) [25], which is among the most precise variants of context sensitivity, 2-type-sensitivity (2type) [32], and context-insensitivity (CI). We observe that

- 2obj is not scalable for 6 out of 10 programs within 3 hours, while it can finish running for 3 programs within 5 minutes;
- program size is far from a reliable predictor—for example, `Jython` (12 718 methods) is smaller than `br1ss` (26 582 methods), however, 2type is not scalable for the former but scalable for the latter;

¹These are all popular open-source applications, including the browser (JUnit) and web (Spark) for the D-Cap benchmark [8].