



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

Minseok Jeon and Hakjoo Oh



SW재난연구센터 workshop @ Jeju, Korea



Two major camps

A:**Call-Site Sensitivity** Can
Object Sensitivity Even for

Object-Oriented Programs

Minseok Jeon and Hakjoo Oh



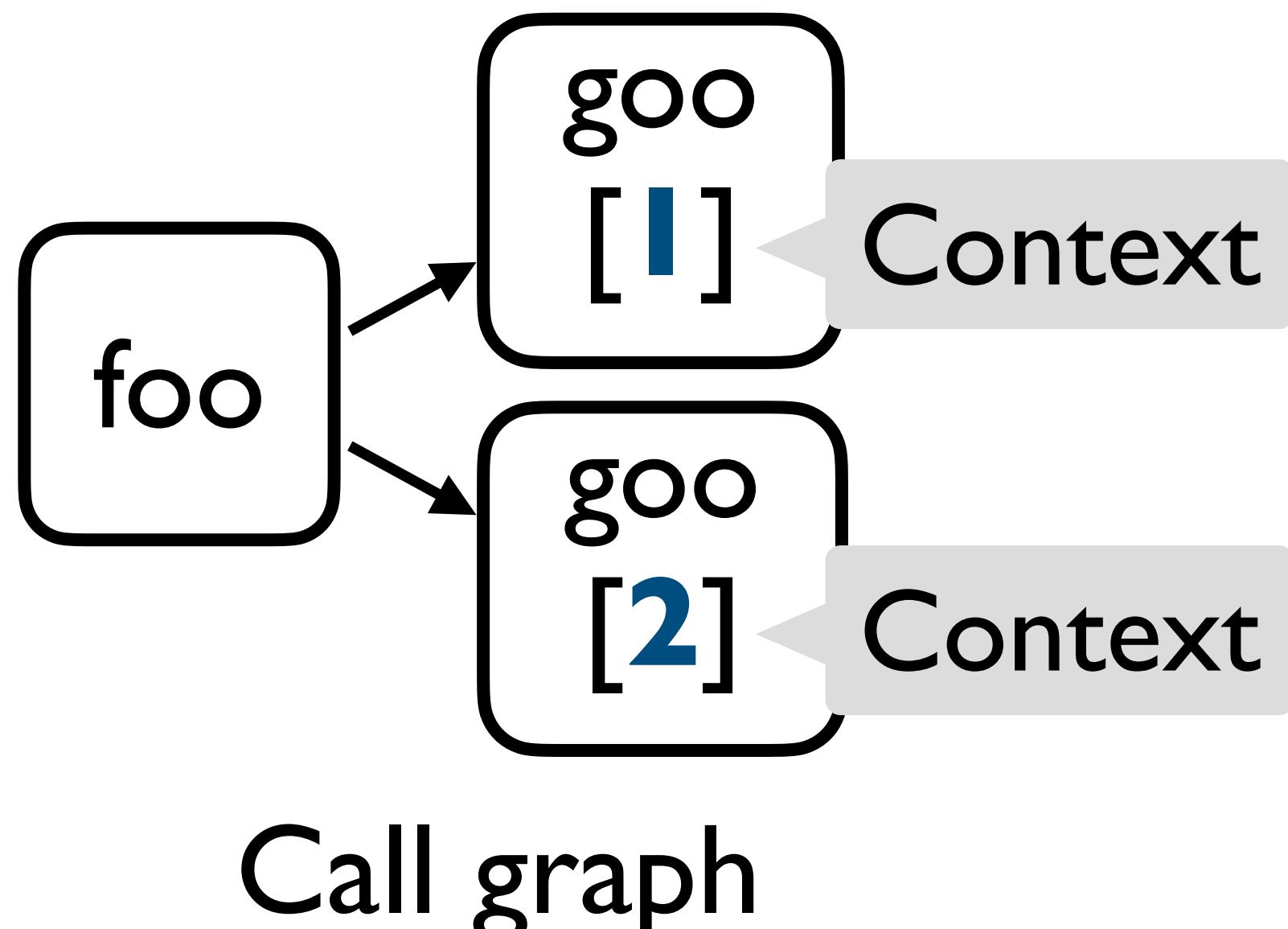
SW재난연구센터 workshop @ Jeju, Korea

Call-site Sensitivity vs Object Sensitivity

Call-site sensitivity was born in 1981

- Considers “**Where**”

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



1981

2002

2010

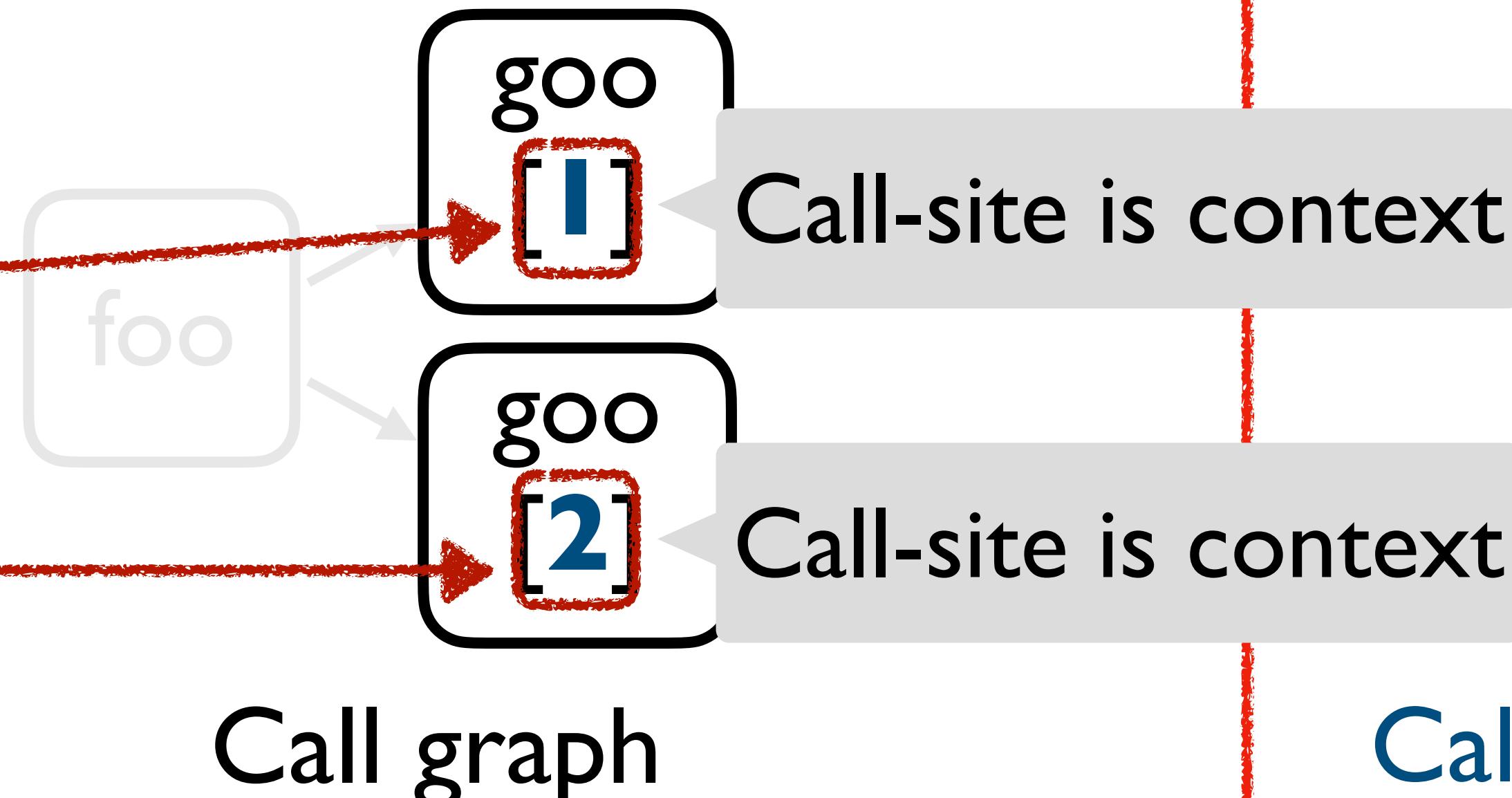
2022

Call-site Sensitivity vs Object Sensitivity

Call-site sensitivity was born in 1981

- Considers “**Where**”

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



Call-site sensitivity

1981

2002

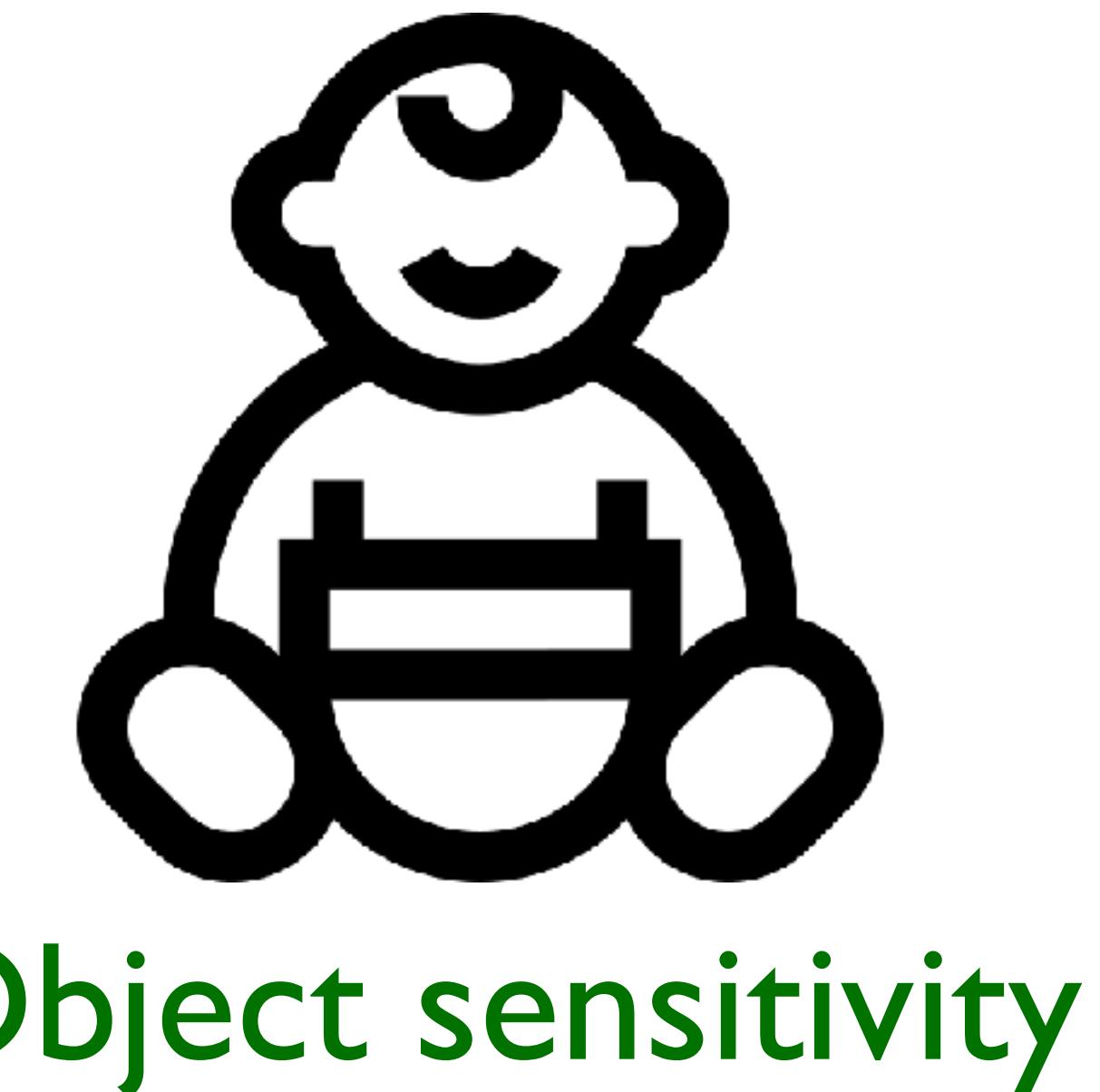
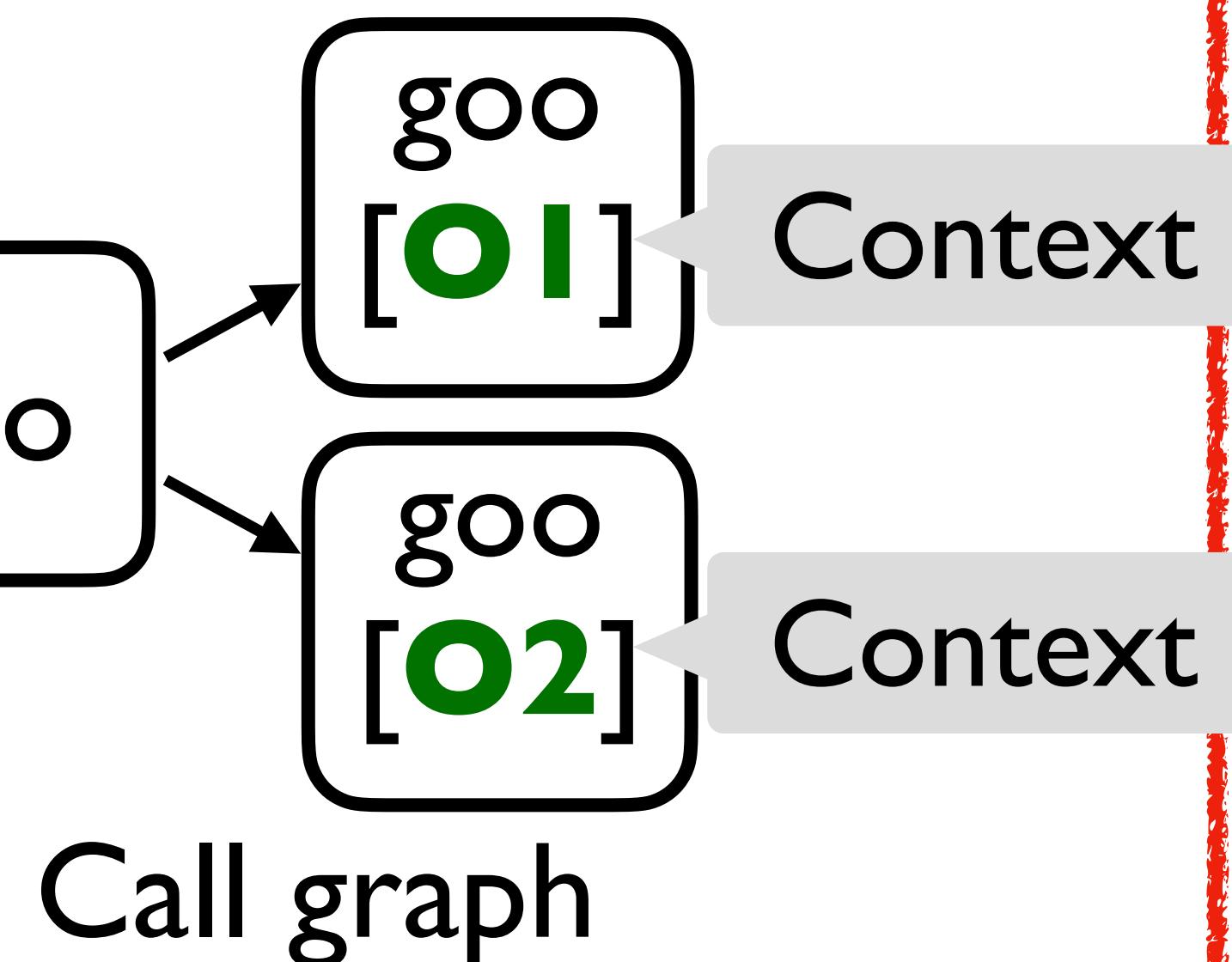
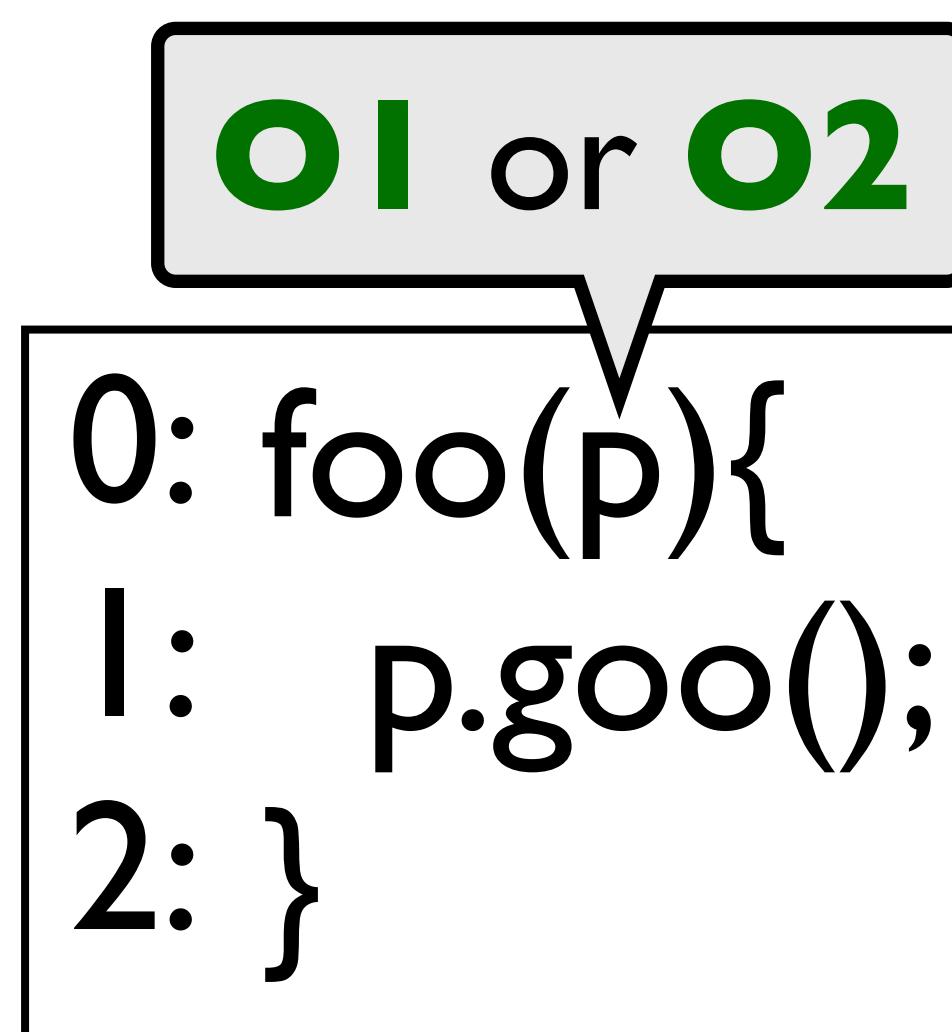
2010

2022

Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

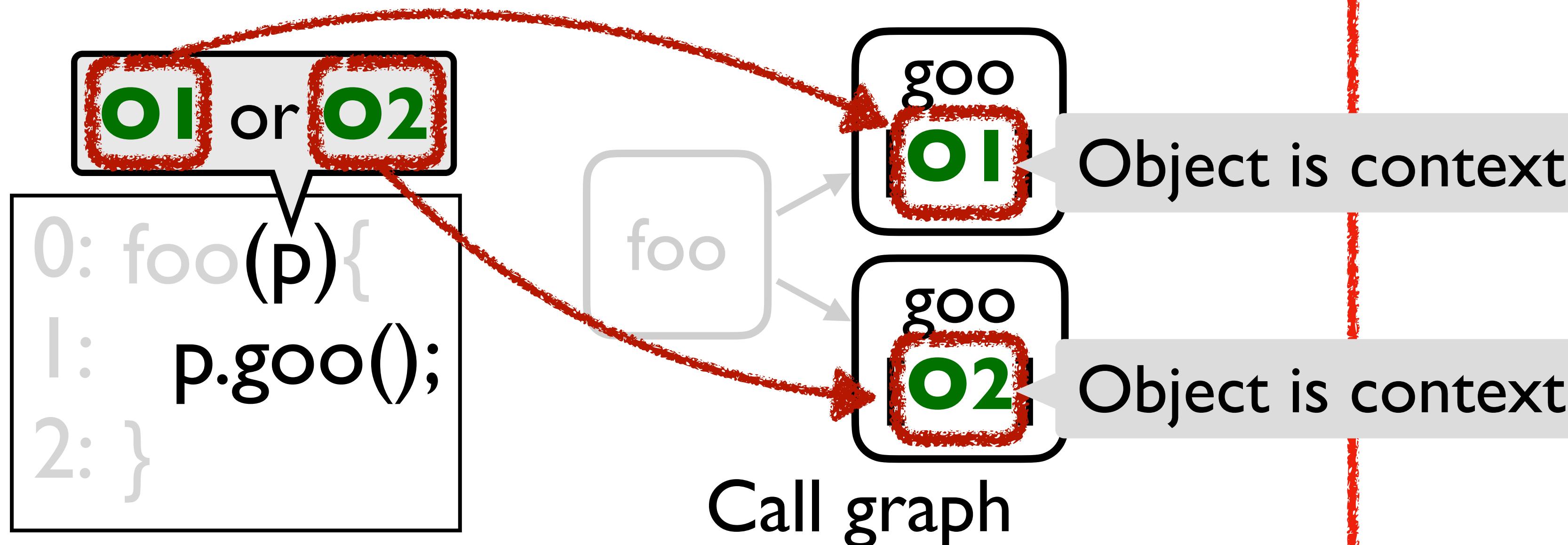
- Considers “**What**”



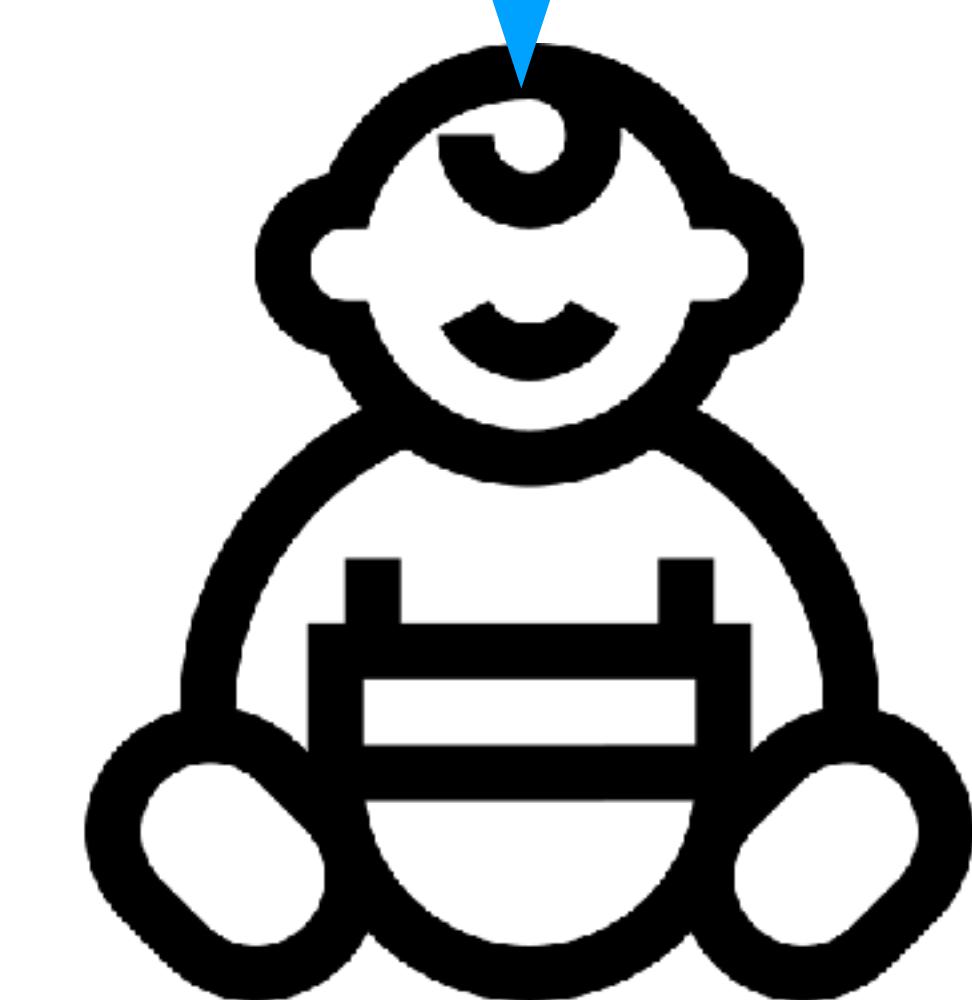
Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

- Considers “**What**”



What is it called with?



Object sensitivity

1981

2002

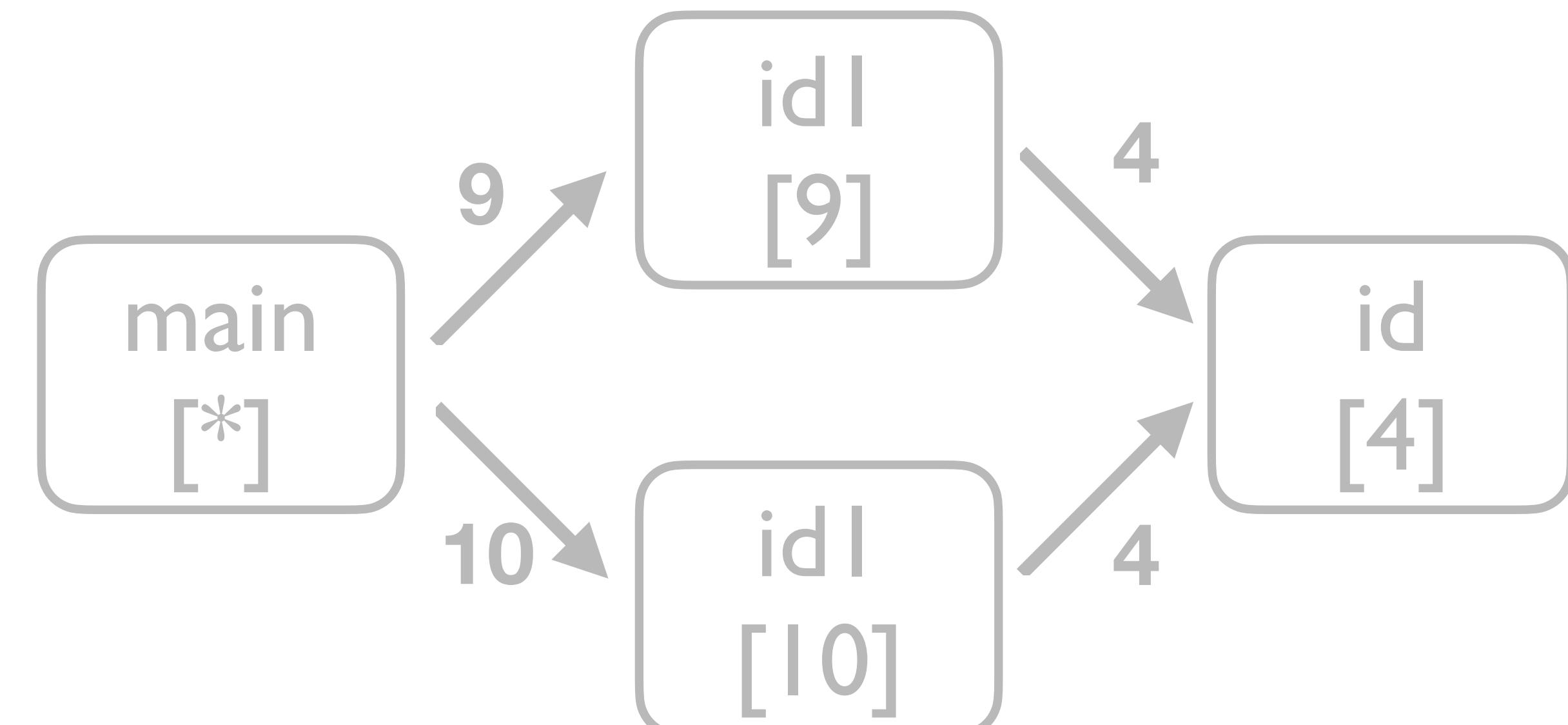
2010

2022

Call-site Sensitivity vs Object Sensitivity

- An example shows the **limitation** of CFA and **strength** of object sensitivity

```
0: class C{  
1:   id(v){  
2:     return v;  
3:   id1(v){  
4:     return this.id(v);}  
5: }  
6: main(){  
7:   c1 = new C();//C1  
8:   c2 = new C();//C2  
9:   a = (A) c1.id1(new A());//query1  
10:  b = (B) c2.id1(new B());//query2  
11: }
```



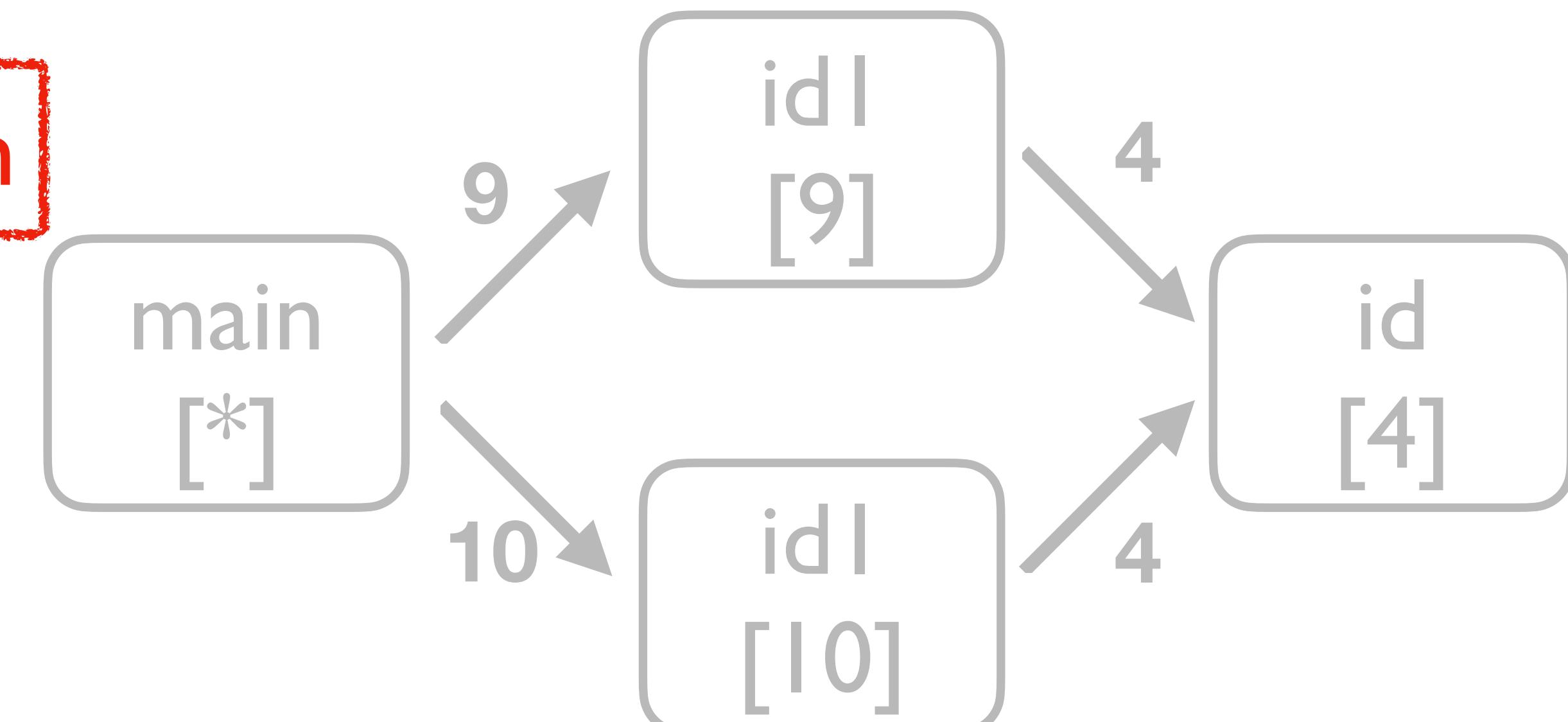
Call-graph of I-CFA

Call-site Sensitivity vs Object Sensitivity

- An example shows the **limitation** of CFA and **strength** of object sensitivity

```
0: class C{  
1:   id(v){  
2:     return v;}  
3:   id1(v){  
4:     return this.id(v);}  
5: }  
6: main(){  
7:   c1 = new C(); //C1  
8:   c2 = new C(); //C2  
9:   a = (A) c1.id1(new A()); //query1  
10:  b = (B) c2.id1(new B()); //query2  
11: }
```

Identity function



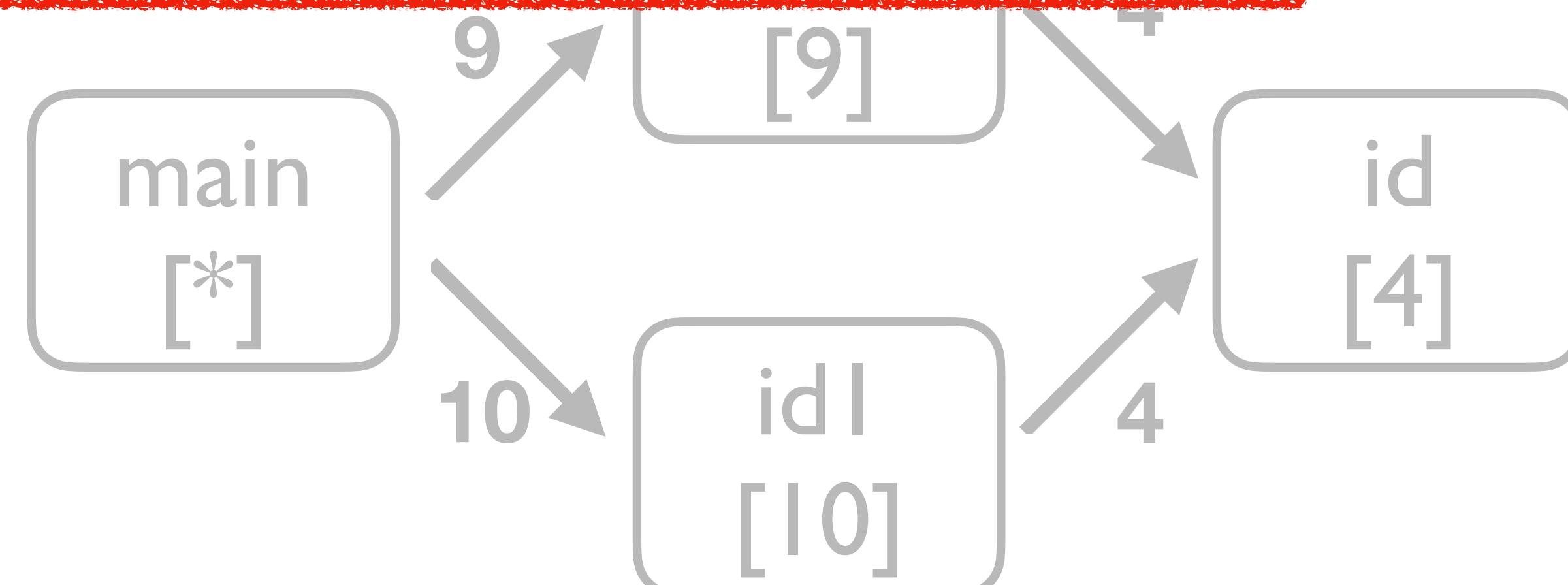
Call-graph of I-CFA

Call-site Sensitivity vs Object Sensitivity

- An example shows the **limitation** of CFA and **strength** of object sensitivity

```
0: class C{  
1:   id(v){  
2:     return v;}  
3:   idI(v){  
4:     return this.id(v);}  
5: }  
6: main(){  
7:   c1 = new C(); //C1  
8:   c2 = new C(); //C2  
9:   a = (A) c1.idI(new A()); //query1  
10:  b = (B) c2.idI(new B()); //query2  
11: }
```

Also an identity function implemented with id

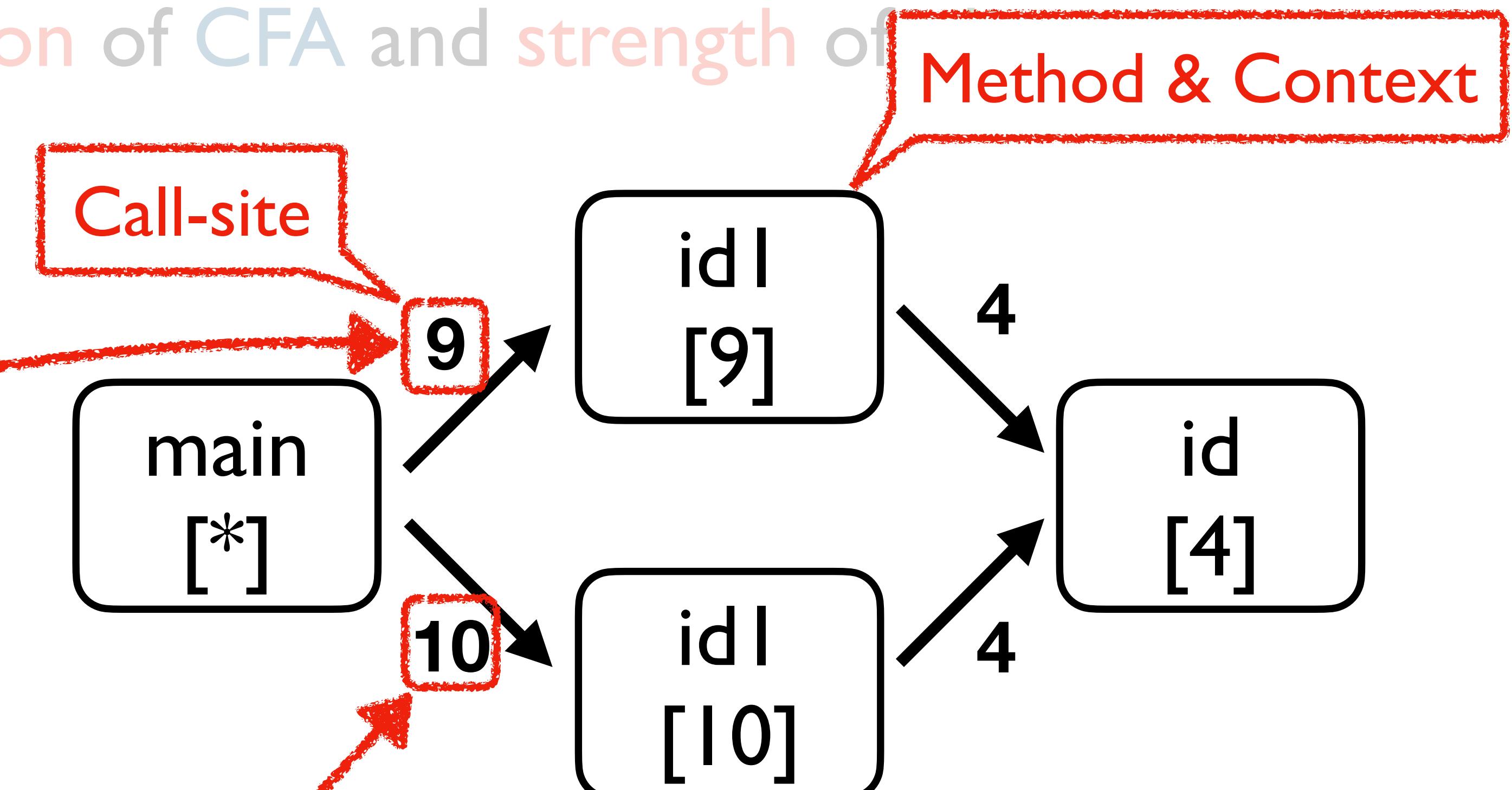


Call-graph of I-CFA

Call-site Sensitivity vs Object Sensitivity

- An example shows the **limitation** of CFA and **strength** of I-CFA

```
0: class C{  
1:   id(v){  
2:     return v;  
3:   }  
4:   id1(v){  
5:     return this.id(v);}  
6: }  
7: main(){  
8:   c1 = new C(); //C1  
9:   c2 = new C(); //C2  
10:  a = (A) c1.id1(new A()); //query1  
11:  b = (B) c2.id1(new B()); //query2  
12: }
```



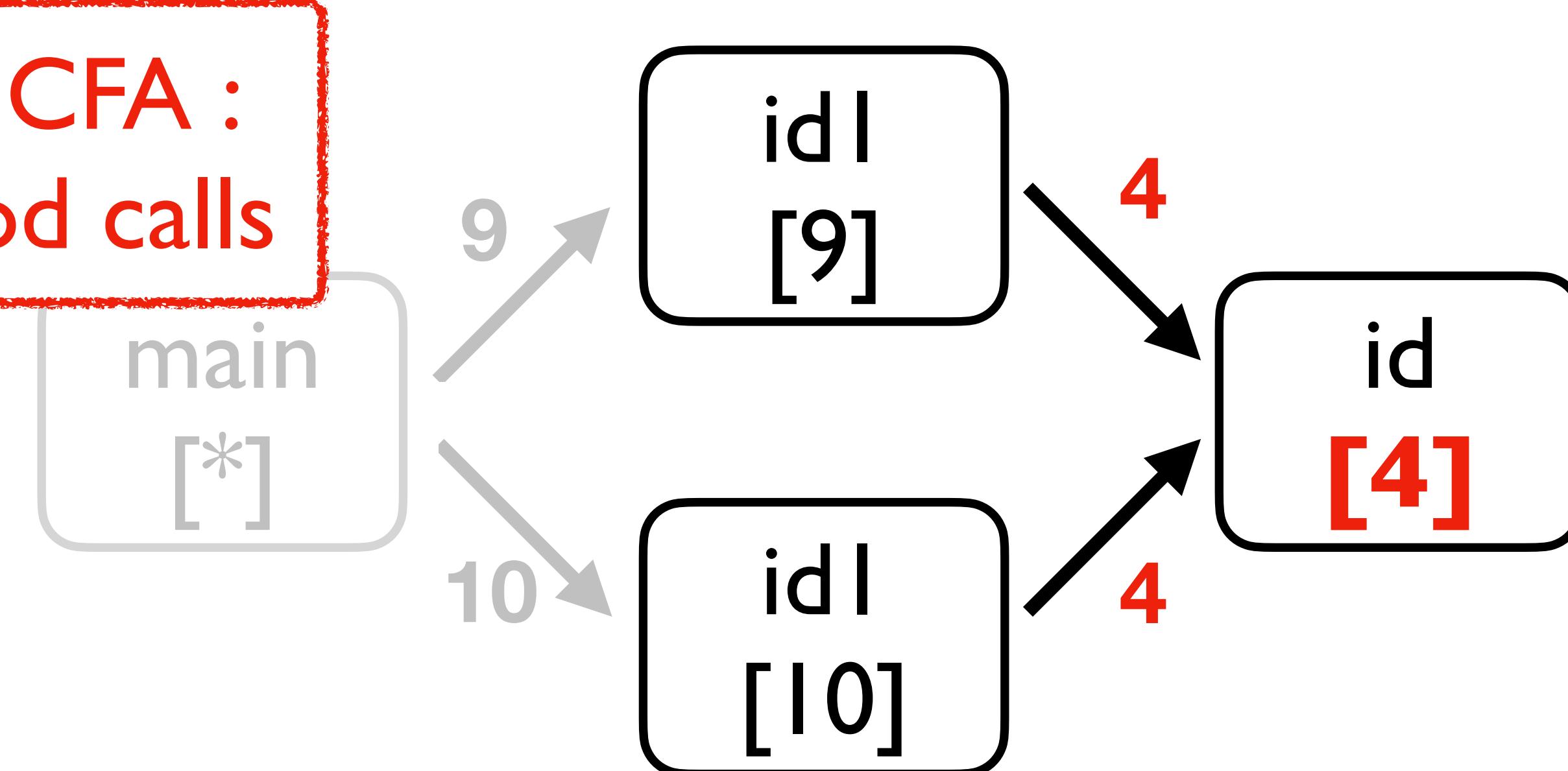
Call-graph of I-CFA

Call-site Sensitivity vs Object Sensitivity

- An example shows the **limitation** of CFA and strength of **object sensitivity**

```
0: class C{  
1:   id(v){  
2:     return v;  
3:   id1(v){  
4:     return this.id(v);}  
5: }  
6: main(){  
7:   c1 = new C(); //C1  
8:   c2 = new C(); //C2  
9:   a = (A) c1.id1(new A()); //query1  
10:  b = (B) c2.id1(new B()); //query2  
11: }
```

Limitation of CFA:
Nested method calls

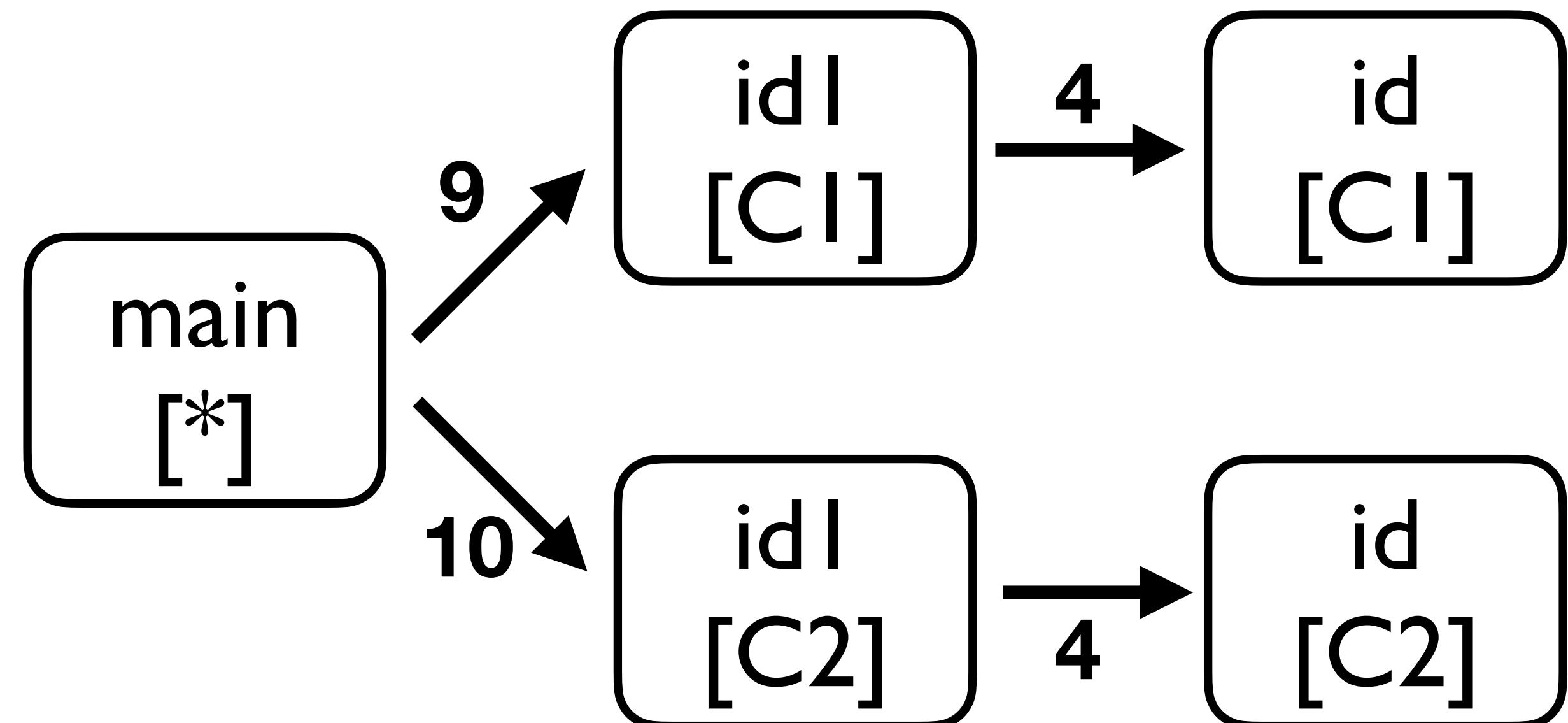


Call-graph of I-CFA

Call-site Sensitivity vs Object Sensitivity

- An example shows the **limitation** of CFA and **strength** of object sensitivity

```
0: class C{  
1:   id(v){  
2:     return v;}  
3:   id1(v){  
4:     return this.id(v);}  
5: }  
6: main(){  
7:   c1 = new C(); //C1  
8:   c2 = new C(); //C2  
9:   a = (A) c1.id1(new A()); //query1  
10:  b = (B) c2.id1(new B()); //query2  
11: }
```

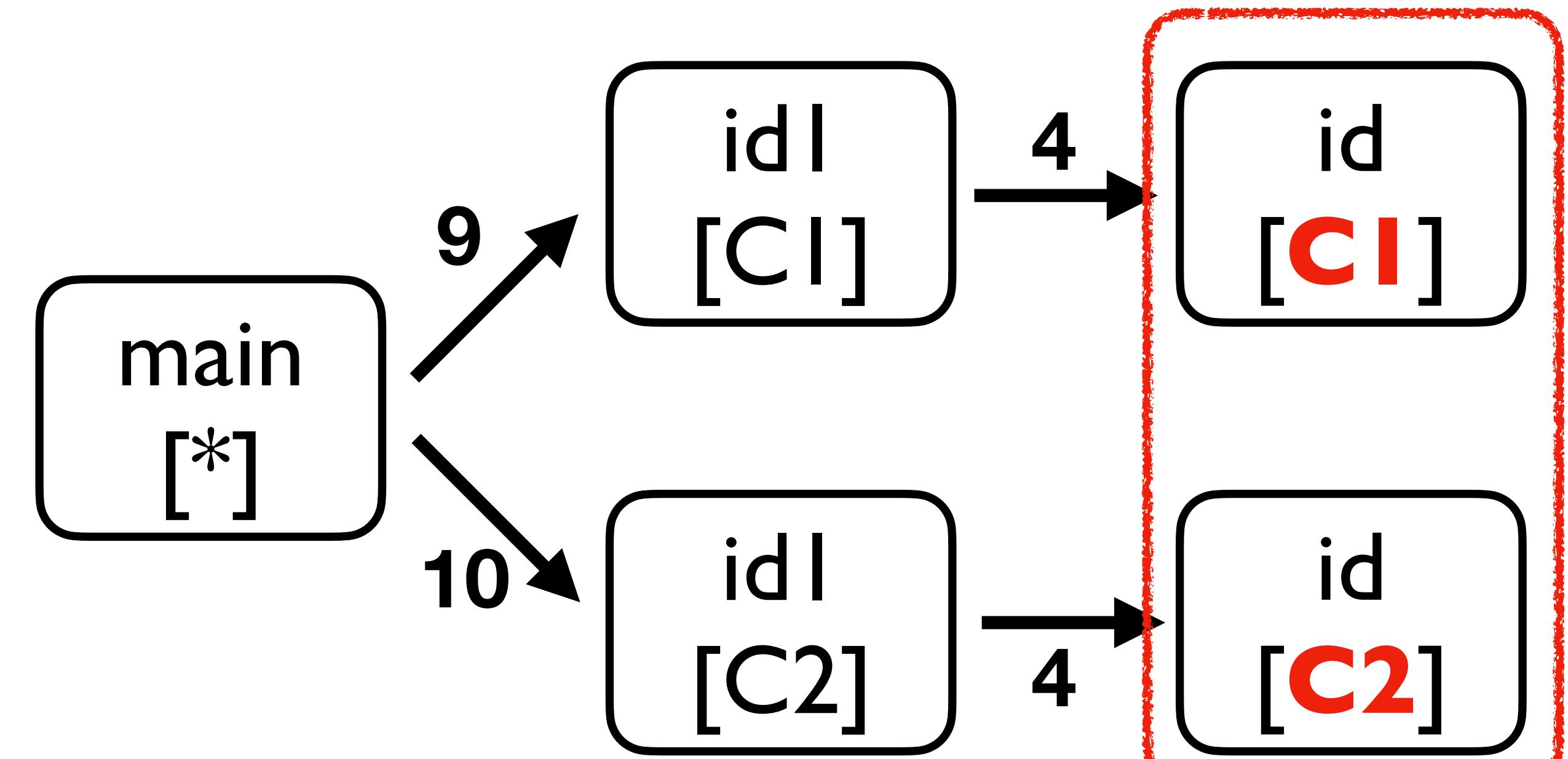


Call-graph of I-Obj

Call-site Sensitivity vs Object Sensitivity

- An example shows the limitation of CFA and strength of object sensitivity

```
0: class C{  
1:     id(v){  
2:         return v;  
3:     }  
4:     id1(v){  
5:         return this.id(v);  
6:     }  
7: }  
main(){  
7:     c1 = new C(); //C1  
8:     c2 = new C(); //C2  
9:     a = (A) c1.id1(new A()); //query1  
10:    b = (B) c2.id1(new B()); //query2  
11: }
```

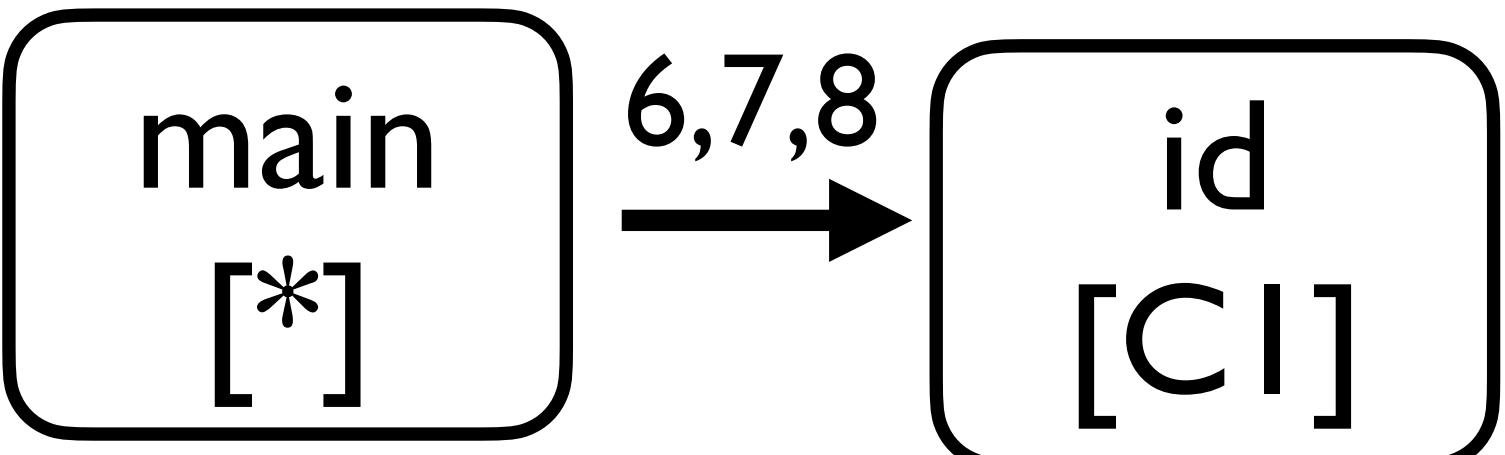


Call-graph of I-Obj

Call-site Sensitivity vs Object Sensitivity

- An example shows the **limitation** of object sensitivity and **strength** of CFA

```
0: class C{  
1:   id(v){  
2:     return v;}  
3: }  
4: main(){  
5:   cl = new C(); //CI  
6:   a = (A) cl.id(new A()); //query1  
7:   b = (B) cl.id(new B()); //query2  
8:   c = (B) cl.id(new C()); //query3  
9: }
```

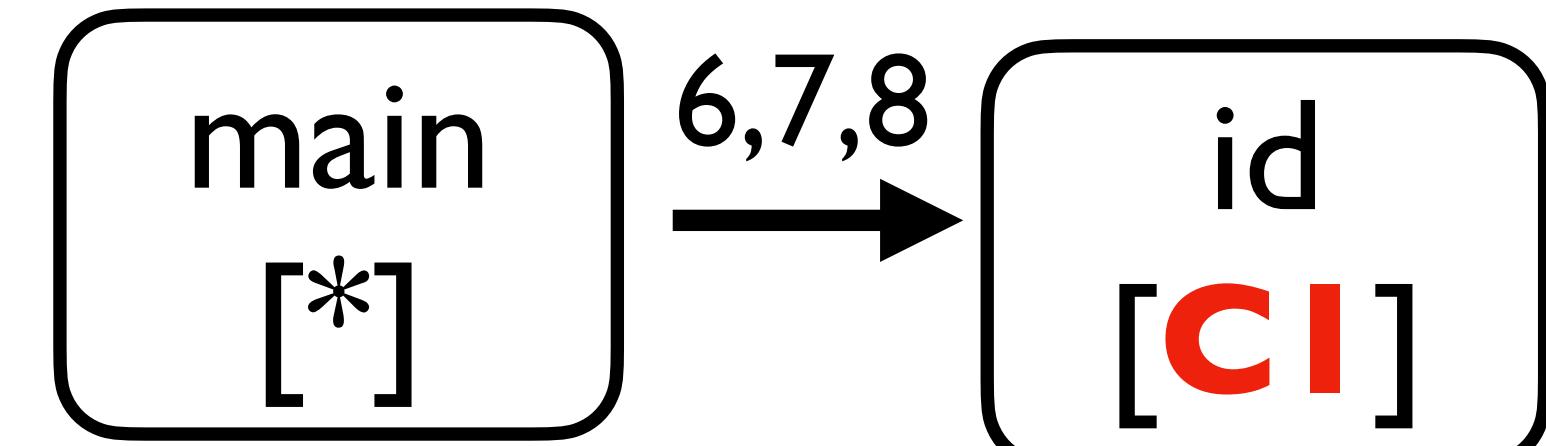


Call-graph of I-Obj

Call-site Sensitivity vs Object Sensitivity

- An example shows the limitation of object sensitivity and strength of CFA

```
0: class C{  
1:   id(v){  
2:     return v;}  
3: }  
4: main(){  
5:   cl = new C();//CI  
6:   a = (A) cl.id(new A());//query1  
7:   b = (B) cl.id(new B());//query2  
8:   c = (B) cl.id(new C());//query3  
9: }
```



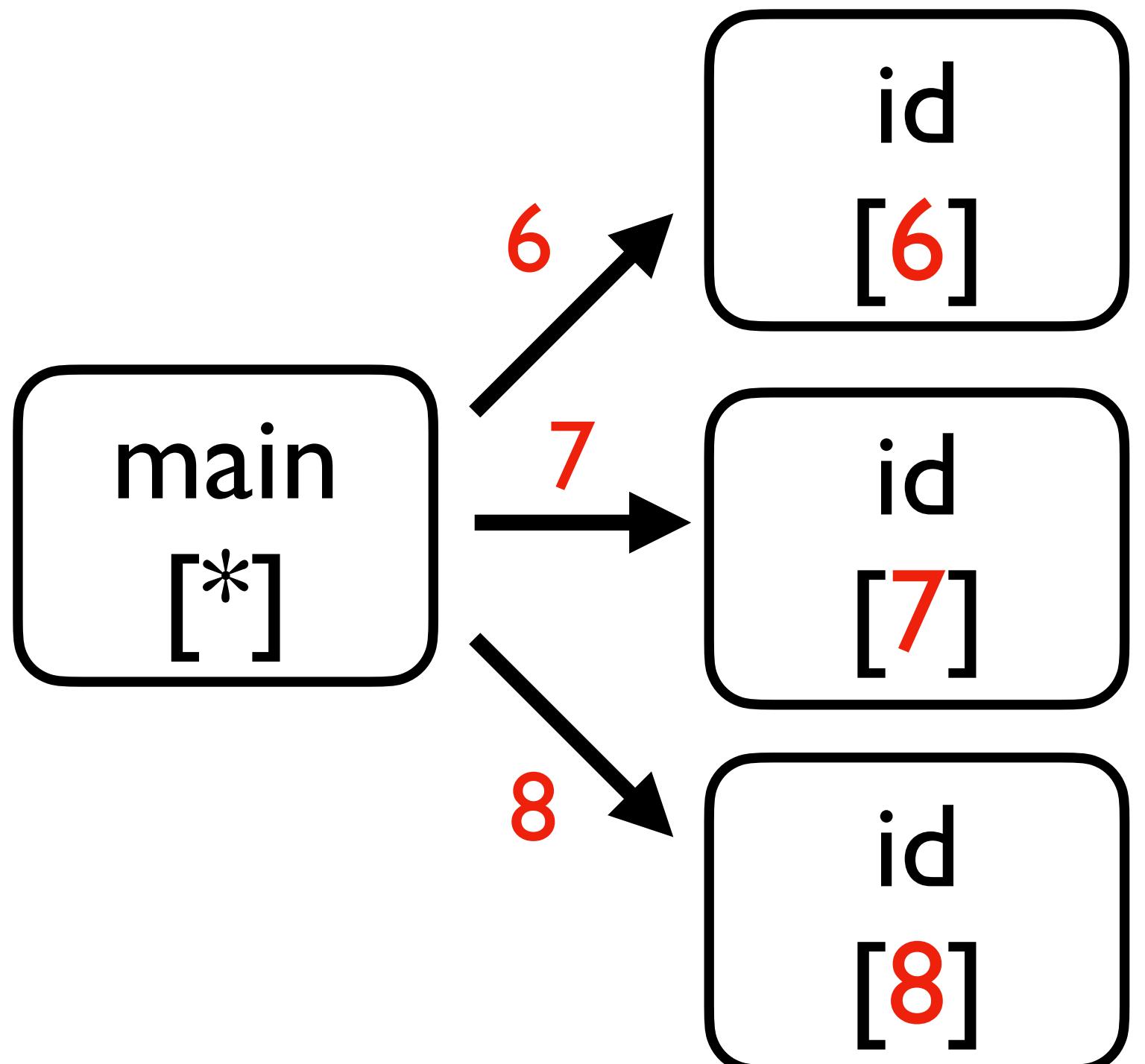
Call-graph of I-Obj

The three method calls share the same receiver object CI

Call-site Sensitivity vs Object Sensitivity

- An example shows the limitation of object sensitivity and strength of CFA

```
0: class C{  
1:   id(v){  
2:     return v;}  
3: }  
4: main(){  
5:   cl = new C(); //C1  
6:   a = (A) cl.id(new A()); //query1  
7:   b = (B) cl.id(new B()); //query2  
8:   c = (C) cl.id(new C()); //query3  
9: }
```



Call-graph of I-CFA

Call-site sensitivity easily separates the three method calls

Call-site Sensitivity vs Object Sensitivity

- Call-site Sensitivity and Object Sensitivity had been **actively compared**

Call-site Sensitivity vs Object Sensitivity



...



Obj vs CFA

1981

2002

2010

2022

Call-site Sensitivity vs Object Sensitivity

- Object Sensitivity **outperformed** call-site sensitivity

Call-site Sensitivity vs Object Sensitivity

Parameterized Object Sensitivity for Points-to Analysis for Java
Ana MILANOVA
Benedict Polyspace Institute
ATANAS RODINOV
Citec State University
and
BARBARA G. RIDER
Rutgers University

Context-sensitive points-to analysis: Is it worth it?
Güney Lhoták^{1,2} and Laurie Hendren³
¹ Charles University, Prague, Czech Republic
² University of Waterloo, Waterloo, ON, Canada
³ School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Evaluating the Benefits of Context-Sensitive Points-to Analysis Using a BDD-Based Implementation
ONDŘEJ LHOTÁK
University of Waterloo
and
LAURIE HENDREN
McGill University

Strictly Declarative Specification of Sophisticated Points-to Analyses
Markus BREUER¹ Yannis STAMOULIS²
¹ Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003 USA
markus.breuer@cs.umass.edu, yannis@cs.umass.edu

Abstract
We present the results of an empirical study evaluating the precision of object-sensitive points-to analysis with several varieties of context sensitivity or annotations of C-style loops. We compare the effect of various settings in the context of context-sensitive points-to analysis of C-like programs with those proposed by Zhu and Cohen and by Weller and Lam. Our study is able to analyze fast context-sensitive points-to analysis, as well as one that also specifies the loop boundaries. We measure both characteristics of the points-to analysis, as well as its effect on the execution of other analyses. To guide development of efficient analysis implementations, we measure the number of nodes, the number of edges, and the time required to perform points-to analysis and context-sensitive points-to analysis. To evaluate precision, we measure the size of the call graph in terms of nodes and edges, in number of executable call sites, and the number of calls statically possible by itself. The results of our study indicate that object-sensitive implementations are likely to make sense and more probably than the other approaches that object-sensitive analysis are more precise than comparable variations of the other approaches. The loop abstraction improves precision more than extending the length of context. The results of our study indicate that object-sensitive implementations are likely to make sense and more probably than the other approaches that object-sensitive analysis are more precise than comparable variations of the other approaches. The loop abstraction improves precision more than extending the length of context.

Categories and Subject Descriptors: B.8 [Hardware Environments]: B.8.1 [Programmable Languages]: Languages—Structures and Semantics
General Terms: Languages, Design, Implementation, Languages

Additional Key Words and Phrases: Interprocedural program analysis, context sensitivity, binary decision diagrams, points-to analysis, C-like programs, context analysis

ACM Reference Format:
Lhoták, O. and Hendren, L. 2008. Evaluating the benefits of context-sensitive points-to analysis using a BDD-based implementation. ACM Trans. Behav. Log. Method., 15, 1, Article 10 (February 2008), 30 pages. DOI=10.1145/1311958.1311967 <http://doi.acm.org/10.1145/1311958.1311967>

A preliminary version of this article appeared in Proceedings of the International Symposium on Software Testing and Analysis July, 2007, pp. 1–11.

This work was supported in part by National Science Foundation grants IIS-0437997 and IIS-0703497. Authors would like to thank the members of Computer Systems, Benedict Polyspace Institute, 1100 8th Street, Suite 200, Washington, DC 20004, USA; and the members of Computer Systems, University of Waterloo, Waterloo, ON N2L 3G1, Canada; and the members of Computer Systems, McGill University, 3480 University Street, Room 310, Vancouver, BC V6T 1Z2, Canada.

This study aims to provide three answers. Recent advances in the use of BDDs for context-sensitive points-to analysis (BDDs for program analysis) [8, 12, 25, 31] have made certain contexts analysis efficient enough to be feasible in a real-world application. The first answer is given in the provided figures. The second answer is that object-sensitive analysis is more precise than call-site sensitive analysis for programs containing recursive procedures. The third answer is that object-sensitive analysis is more precise than call-site sensitive analysis for programs containing loops.

Obj wins

Obj wins

Obj wins

Obj wins



Obj

2022

CFA

1981

2002

2010

2022

Call-site Sensitivity vs Object Sensitivity

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

Object-Sensitivity

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

```
program
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);
    }
}
```

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

**Object-Sensitivity
(vs. call-site sensitivity)**

```
program
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 analysis

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

Lecture Notes: Pointer Analysis
15-8190: Program Analysis
Jonathan Aldrich
jonathan.aldrich@cs.cmu.edu
Lecture 9

1 Motivation for Pointer Analysis

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

```
1: z := 1;
2: p := &x;
3: x := 2;
4: print z;
```

In order to analyze this program correctly we must be aware of the information that instruction 3 `p` points to `x`. If this information is available we can write the following rule:

$$jcp[\ast p := y](\sigma) = [z \rightarrow \pi(y)]\sigma \text{ where } \pi(y) \text{ points to } y$$

When we know exactly what a variable `a` points to, we say that `a` has a *use-point-to* information, and we can perform a strong update of variable `z`, because we know with confidence that assigning to `z` does not affect the value of `a`. A technically in the rule is quantifying over all `z` such that `z` points to `x`. How is this possible? It is not possible in C or Java, a language with pass-by-reference, for example. In C++, it is possible to have two variables with the same name and different addresses.

Of course, it is also possible that we are uncertain to which distinct locations `p` points. For example:

```
jcp[\ast p := y](\sigma) = [z \rightarrow \pi(y)]\sigma \text{ where } \pi(y) \text{ points to } y
```

Pointer Analysis

Yannis Smaragdakis
University of Athens
smaragd@hua.gr

George Ballofousis
University of Athens
gballof@hua.gr

now
the essence of knowledge
Boston - Delhi

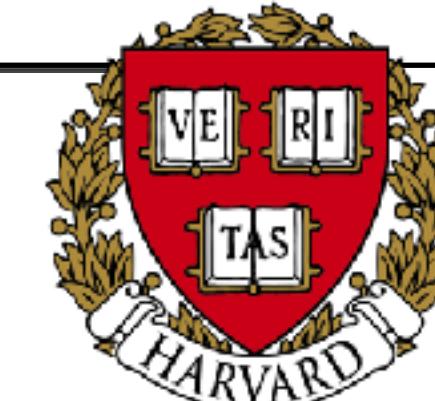
• • •



Obj



National and Kapodistrian
University of Athens



Carnegie
Mellon
University



1981

2002

2010

2022

Call-site Sensitivity vs Object Sensitivity

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

Object-Sensitivity

- The dominant flavor of context-sensitivity for object-oriented languages.
- It uses object abstractions (i.e. allocation sites) as contexts, 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`.

Hakjoon Oh

AAA616 2019 Fall, Lecture 8

November 16, 2019 27 / 31



KOREA
UNIVERSITY

Object-sensitive pointer analysis

- Milanova, Rountev, and Ryder. Parameterized object sensitivity for pointer analysis for Java. ACM Trans. Softw. Eng. Eng. Methodol., 35(1):1–32, 2019.

Lecture Notes:
Pointer Analysis

15-819Q: Program Analysis
Jonathan Aldrich
Jonathan.Aldrich@cs.cmu.edu
Lecture 8

Pointer Analysis
Yannis Smaragdakis
University of Athens

I was also taught like that



Carnegie
Mellon
University

now
the essence of knowledge



1981

2002

2010

2022

Call-site Sensitivity vs Object Sensitivity

- Researches focused on improving Object Sensitivity



Researches on Object Sensitivity

1981

2002

2010

Call-site Sensitivity vs Object Sensitivity

- Call-site Sensitivity has been ignored

“We do not consider call-site sensitive analyses ...”
- Li et al. [2018]

A Machine-Learning Algorithm with Disjunctive Models for Data-Driven Program Analysis

Making k -Object-Sensitive Pointer Analysis More Precise with Still k -Limiting

Scalability-Pins: Pointer Analysis Self-Tuning Context-Sensitivity

The Making of a Precise and Scalable Pointer Analysis

Hybrid Context-Sensitivity for Points-To Analysis

Precision-Guided Context Sensitivity for Points-To Analysis

Introspective Analysis: Context-Sensitivity Across the Board

1981

2002

2010

2022



CFA

Call-site Sensitivity vs Object Sensitivity

- Call-site Sensitivity has been ignored

“We have included 2cs+h to demonstrate the superiority of object sensitivity over call-site sensitivity”

- Tan et al. [2016]

Making k-Object-Sensitive Pointer Analysis More Precise with Still k-Limiting
A Machine-Learning Algorithm with Disjunctive Methods for Data-Driven Program Analysis
NIPS'16, 2016, Shufun Jiang¹, Sungchuluk Lha¹, and Marjanović², Republic of Korea

Scalability-Poss: Pointer Analysis Using Context-Sensitivity
Scalability-Poss: Pointer Analysis Using Context-Sensitivity
Yiannis Giannidis¹, Department of Informatics, University of Patras, GR-265 20 Patras, Greece, and Department of Mathematics, University of Patras, GR-265 20 Patras, Greece, and Department of Mathematics, University of Patras, GR-265 20 Patras, Greece

Pick Your Contexts Well: Understanding Object-Sensitivity
The Making of a Precise and Scalable Pointer Analyzer
Yiannis Giannidis¹, Department of Informatics, University of Patras, GR-265 20 Patras, Greece, and Department of Mathematics, University of Patras, GR-265 20 Patras, Greece, and Department of Mathematics, University of Patras, GR-265 20 Patras, Greece

Hybrid Context-Sensitivity for Points-To Analysis
George Karidis¹, Yiannis Giannidis¹, Department of Informatics, University of Patras, GR-265 20 Patras, Greece

Precision-Guided Context Sensitivity for Points-To Analysis
Yiannis Giannidis¹, George Karidis¹, Department of Informatics, University of Patras, GR-265 20 Patras, Greece

Introspective Analysis: Context-Sensitivity Across the Board
Yiannis Giannidis¹, George Karidis¹, George Salomkos², Department of Informatics, University of Patras, GR-265 20 Patras, Greece, and Department of Mathematics, University of Patras, GR-265 20 Patras, Greece



CFA

1981

2002

2010

2022

Call-site Sensitivity vs Object Sensitivity

- Call-site Sensitivity has been ignored

“... we do not discuss our approach for call-site sensitivity”
- Jeon et al. [2019]

The figure shows five academic papers from 2019, each with a red border, arranged in a grid-like fashion. A large red hand-drawn style bracket encloses all five papers. To the right of the bracket, there is a large red question mark and the acronym CFA (Context-Sensitivity Across the Board).

- Making k-Object-Sensitive Pointer Analysis More Precise with Still k-Limiting**
Tian Tan¹, Yue Li², and Jingling Xue^{1,3}
¹ School of Computer Science and Engineering, USW Australia
² Advanced Innovation Center for Imaging Technology, CSM, China
³ Republic of Korea
- Scalability-Prist Pointer Analysis Using Context-Sensitive Self-Tuning Context-Sensitivity**
Yiwei Goumpelis¹
¹ Institute of Software, Chinese Academy of Sciences, Beijing, China
and Department of Mathematics, University of Michigan, Ann Arbor, MI, USA
- Pick Your Contexts Well: Understanding Object-Sensitivity**
The Making of a Precise and Scalable Pointer Analyzer
Yiwei Goumpelis¹
¹ Institute of Software, Chinese Academy of Sciences, Beijing, China
and Department of Mathematics, University of Michigan, Ann Arbor, MI, USA
- Hybrid Context-Sensitivity for Points-To Analysis**
George Karidis¹, Tianan Xiangpali²
¹ School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
² School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
- Precision-Guided Context Sensitivity for Points-to Analysis**
Yiwei Goumpelis¹, Tianan Xiangpali²
¹ Institute of Software, Chinese Academy of Sciences, Beijing, China
and Department of Mathematics, University of Michigan, Ann Arbor, MI, USA
² School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
- Introspective Analysis: Context-Sensitivity Across the Board**
Yiwei Goumpelis¹, George Karidis², George Salomkos³
¹ Institute of Software, Chinese Academy of Sciences, Beijing, China
² School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
³ School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA

1981

2002

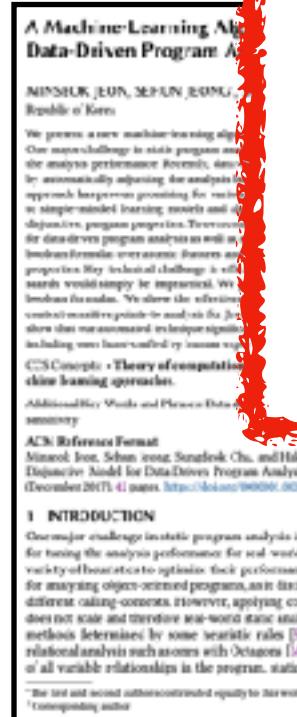
2010

2022

Call-site Sensitivity vs Object Sensitivity

- Call-site Sensitivity has been ignored

“... we do not discuss our approach for call-site sensitivity”
-Jeon et al. [2019]



I also strongly dismissed call-site sensitivity



CFA

1981

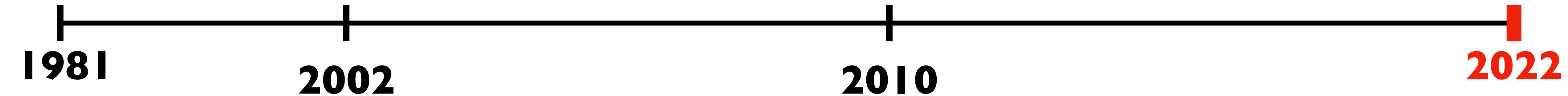
2002

2010

2022

Call-site Sensitivity vs Object Sensitivity

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



Call-site Sensitivity vs Object Sensitivity

A technique **context tunneling** is proposed



Context tunneling can improve both
call-site sensitivity and object sensitivity

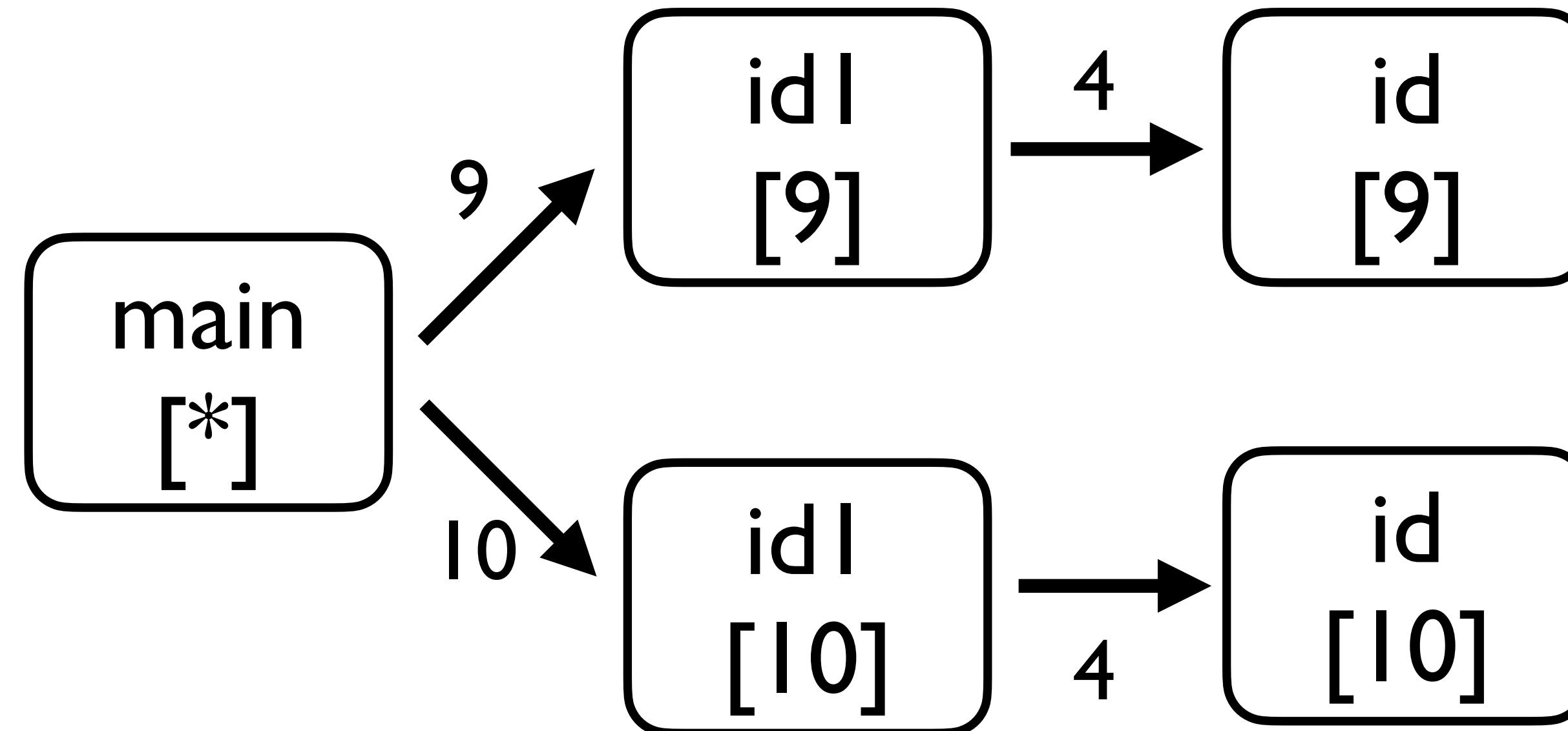
Jeon et al. [2018]



Call-site Sensitivity vs Object Sensitivity

- **Context tunneling** can remove the limitation of call-site sensitivity

```
0: class C{  
1:   id(v){  
2:     return v;  
3:   id1(v){  
4:     return id0(v);}  
5: }  
6: main(){  
7:   c1 = new C();//C1  
8:   c2 = new C();//C2  
9:   a = (A) c1.id1(new A());//query1  
10:  b = (B) c2.id1(new B());//query2  
11: }
```

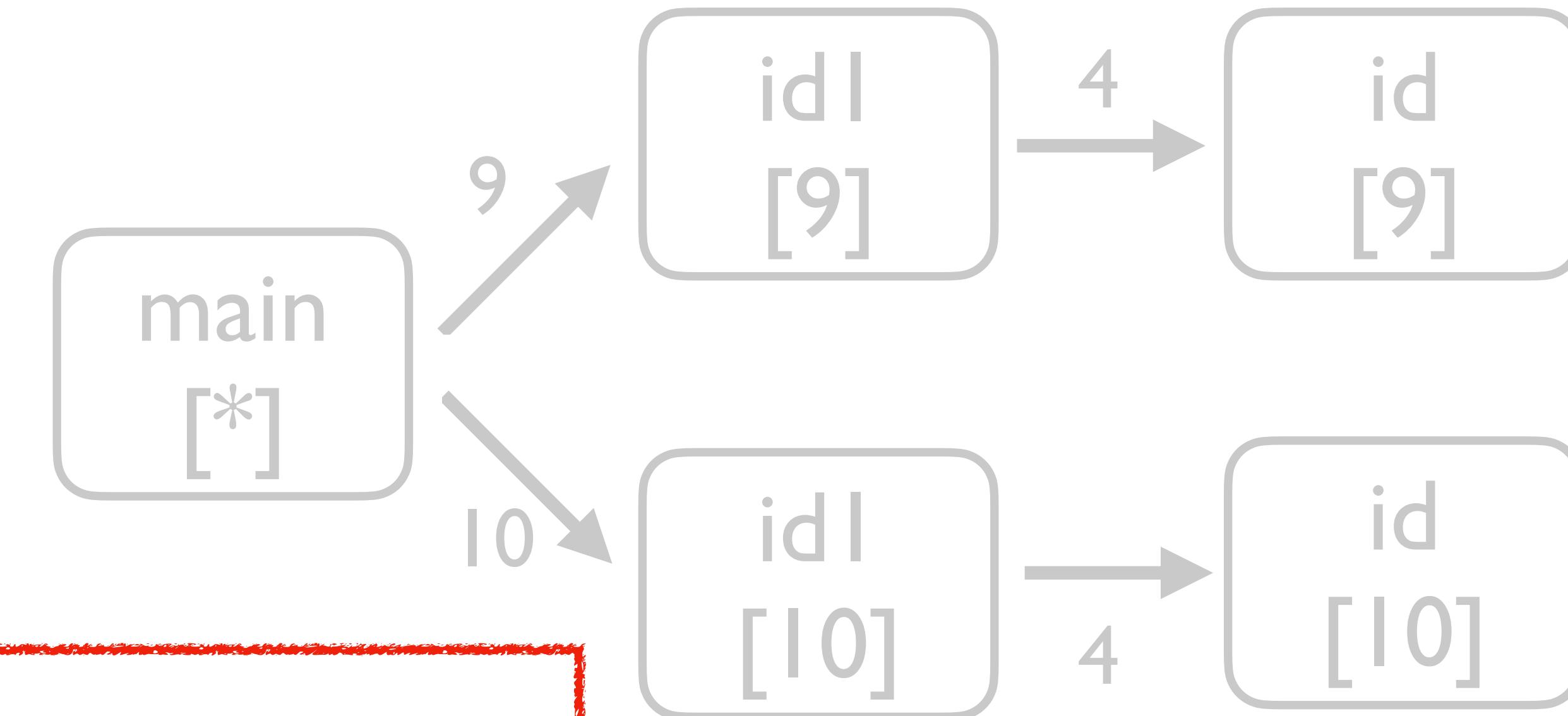


I-CFA with context tunneling
(T= {4})

Call-site Sensitivity vs Object Sensitivity

- Context tunneling can remove the limitation of call-site sensitivity

```
0: class C{  
1:     id(v){  
2:         return v;  
3:     }  
4:     idI(v){  
5:         return id0(v);}  
6: }  
7: main(){  
8:     cl = new C();  
9: }
```



Tunneling abstraction:

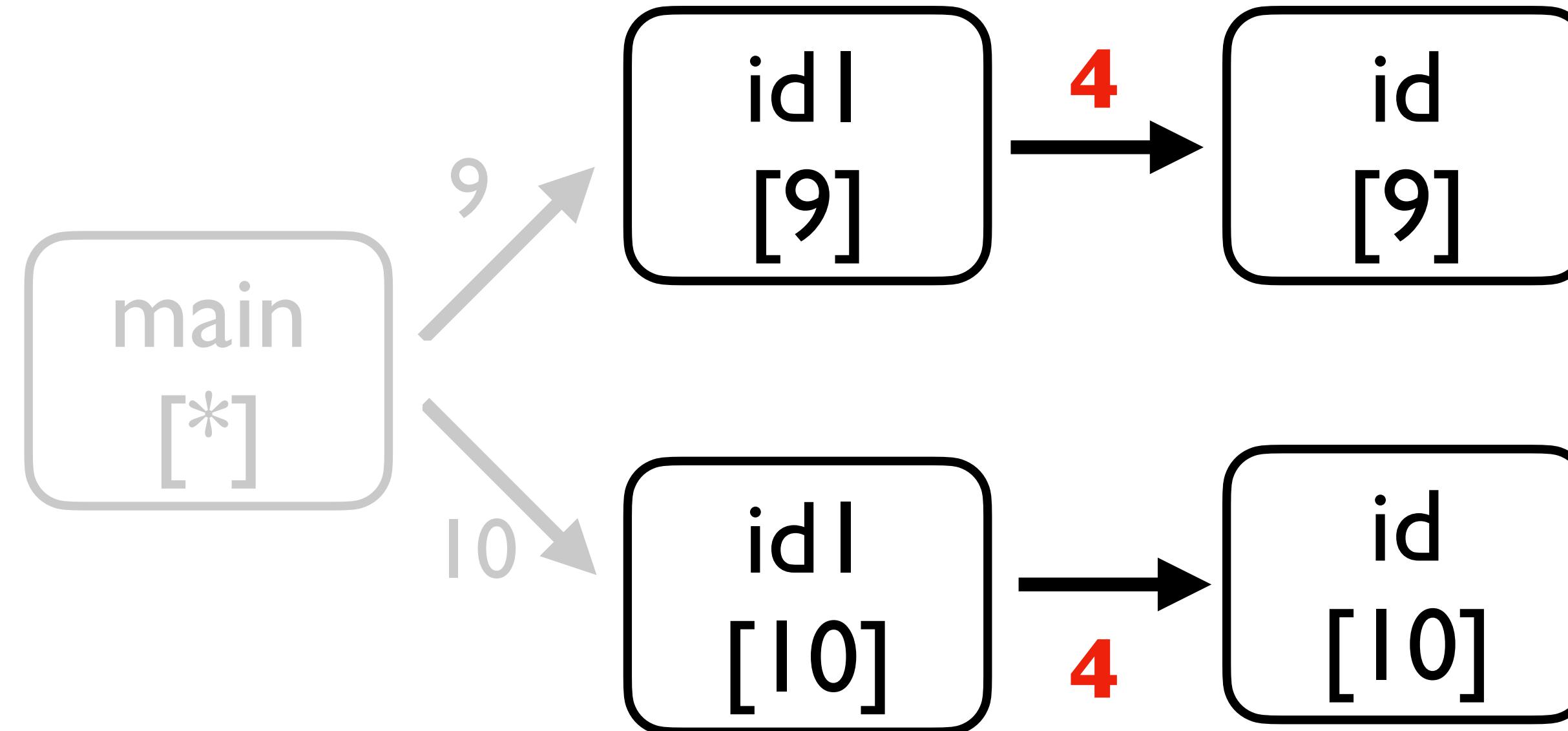
Determines where to apply context tunneling

with context tunneling
($T = \{4\}$)

Call-site Sensitivity vs Object Sensitivity

- Context tunneling can remove the limitation of call-site sensitivity

```
0: class C{  
1:   id(v){  
2:     return v;  
3:   }  
4:   id1(v){  
5:     return id0(v);}  
6: }  
6: main(){  
7:   c1 = new C(); //C1  
8:   c2 = new C(); //C2  
9:   a = (A) c1.id1(new A()); //query1  
10:  b = (B) c2.id1(new B()); //query2  
11: }
```



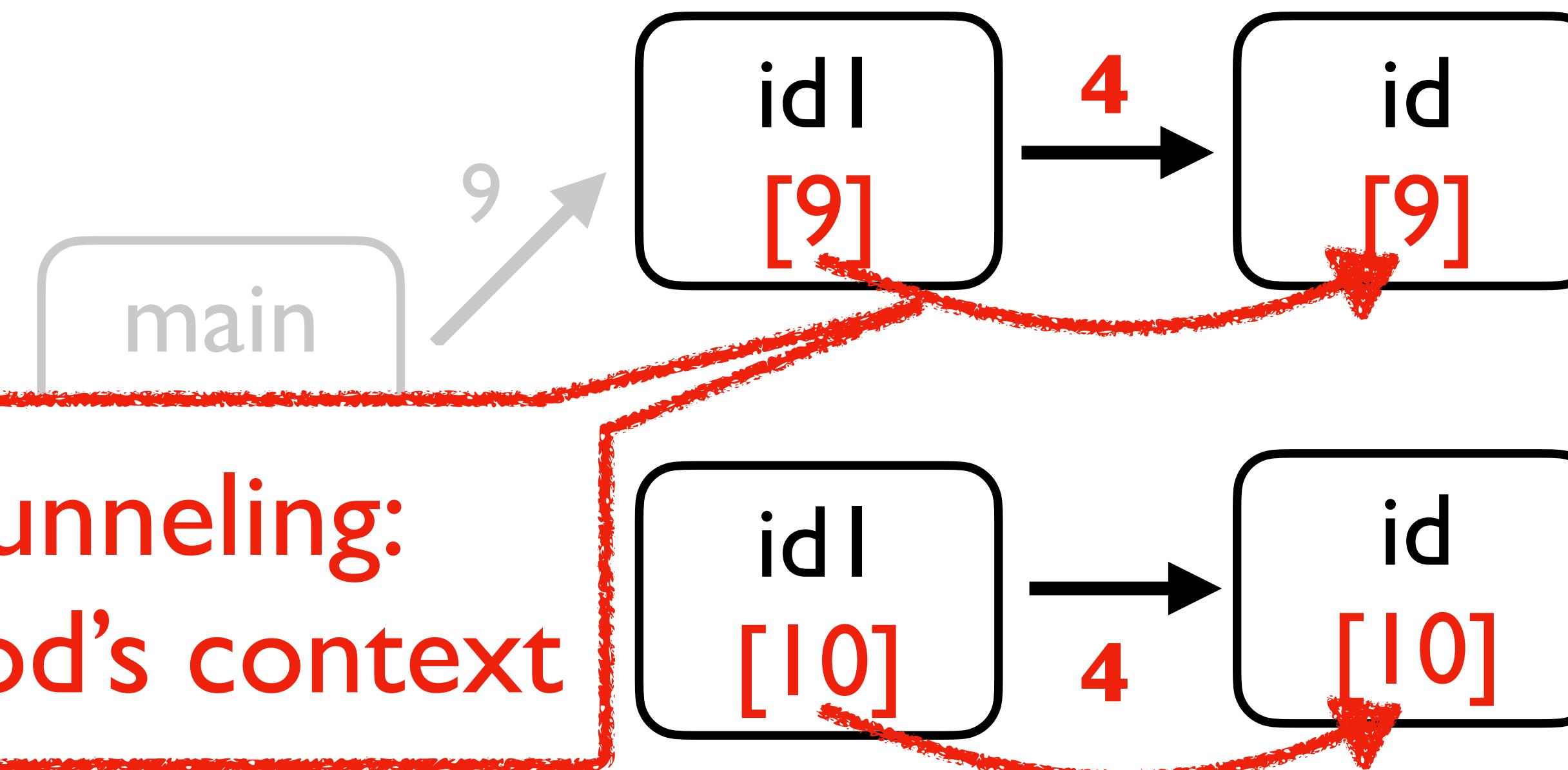
I-CFA with context tunneling
(T= {4})

Unimportant call-sites that should not be used as context elements

Call-site Sensitivity vs Object Sensitivity

- Context tunneling can remove the limitation of call-site sensitivity

```
0: class C{  
1:   id(v){  
2:     return v;  
3:   }  
4:   id1(v){  
5:     return id0(v);  
6:   }  
7: }  
main()  
8: {  
9:   c1 = new C();  
10:  a = (A) c1.id1(new A()); //query1  
11:  b = (B) c1.id1(new B()); //query2  
12: }
```



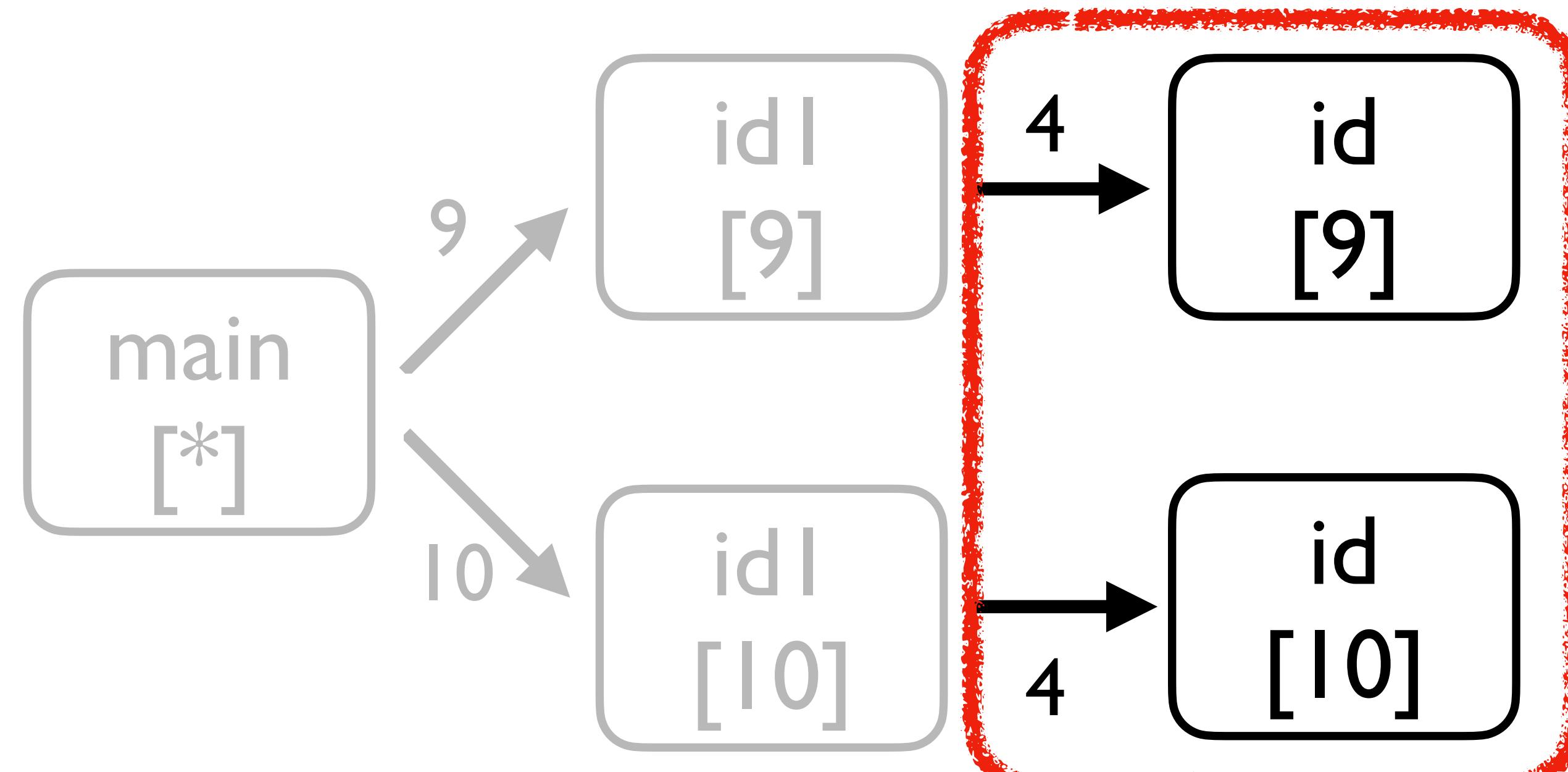
Apply context tunneling:
Inherit caller method's context

I-CFA with context tunneling
(T= {4})

Call-site Sensitivity vs Object Sensitivity

- **Context tunneling** can remove the limitation of call-site sensitivity

```
0: class C{  
1:   id(v){  
2:     return v;  
3:   id1(v){  
4:     return id0(v);}  
5: }  
6: main(){  
7:   c1 = new C();//C1  
8:   c2 = new C();//C2  
9:   a = (A) c1.id1(new A());//query1  
10:  b = (B) c2.id1(new B());//query2  
11: }
```



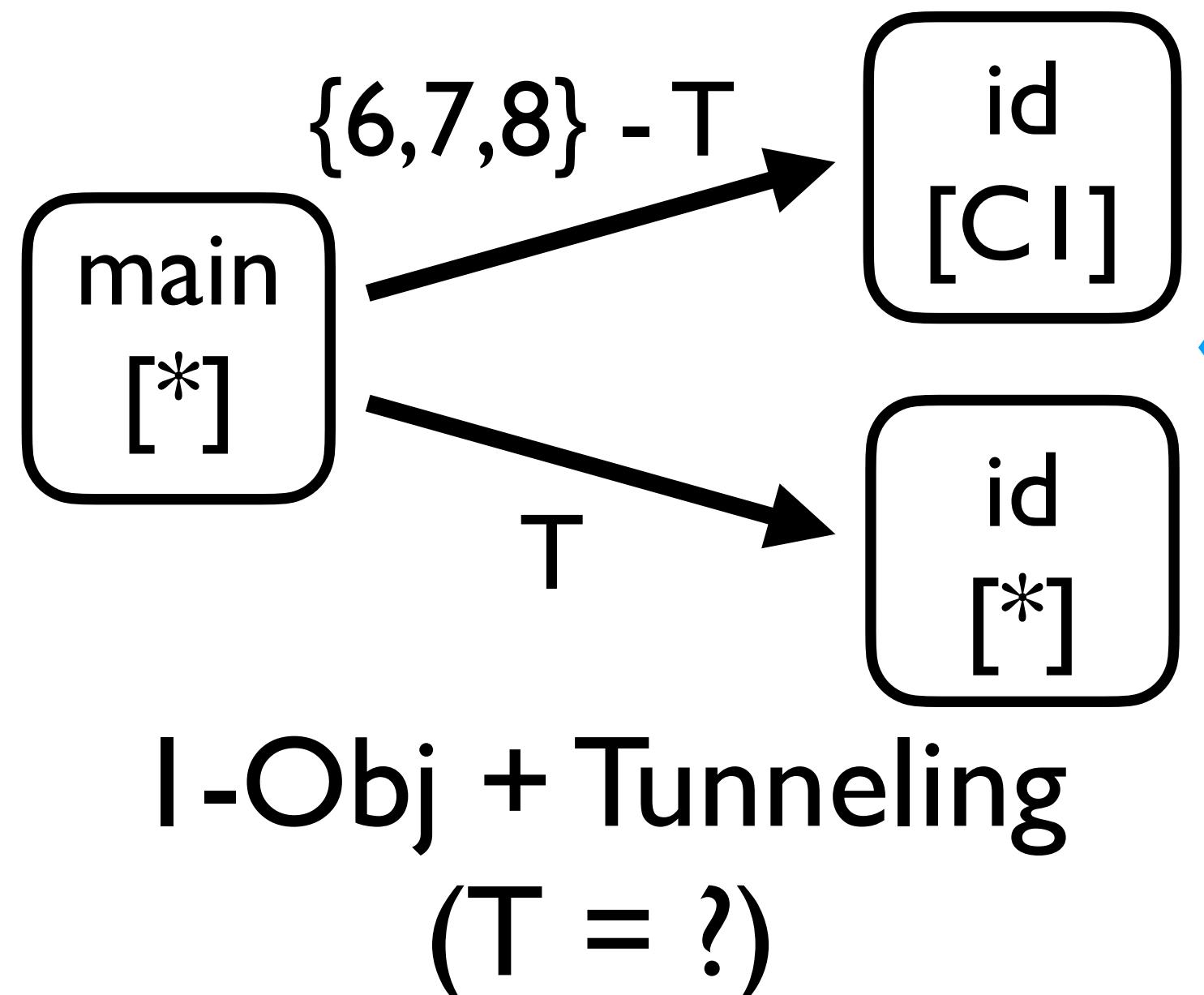
I-CFA with context tunneling
($T = \{4\}$)

With tunneling, I-CFA separates the nested method calls

Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation

```
0: class C{  
1:   id(v){  
2:     return v;}  
3: }  
4: main(){  
5:   cl = new C(); //CI  
6:   a = (A) cl.id(new A());  
7:   b = (B) cl.id(new B());  
8:   c = (C) cl.id(new C());  
9: }
```

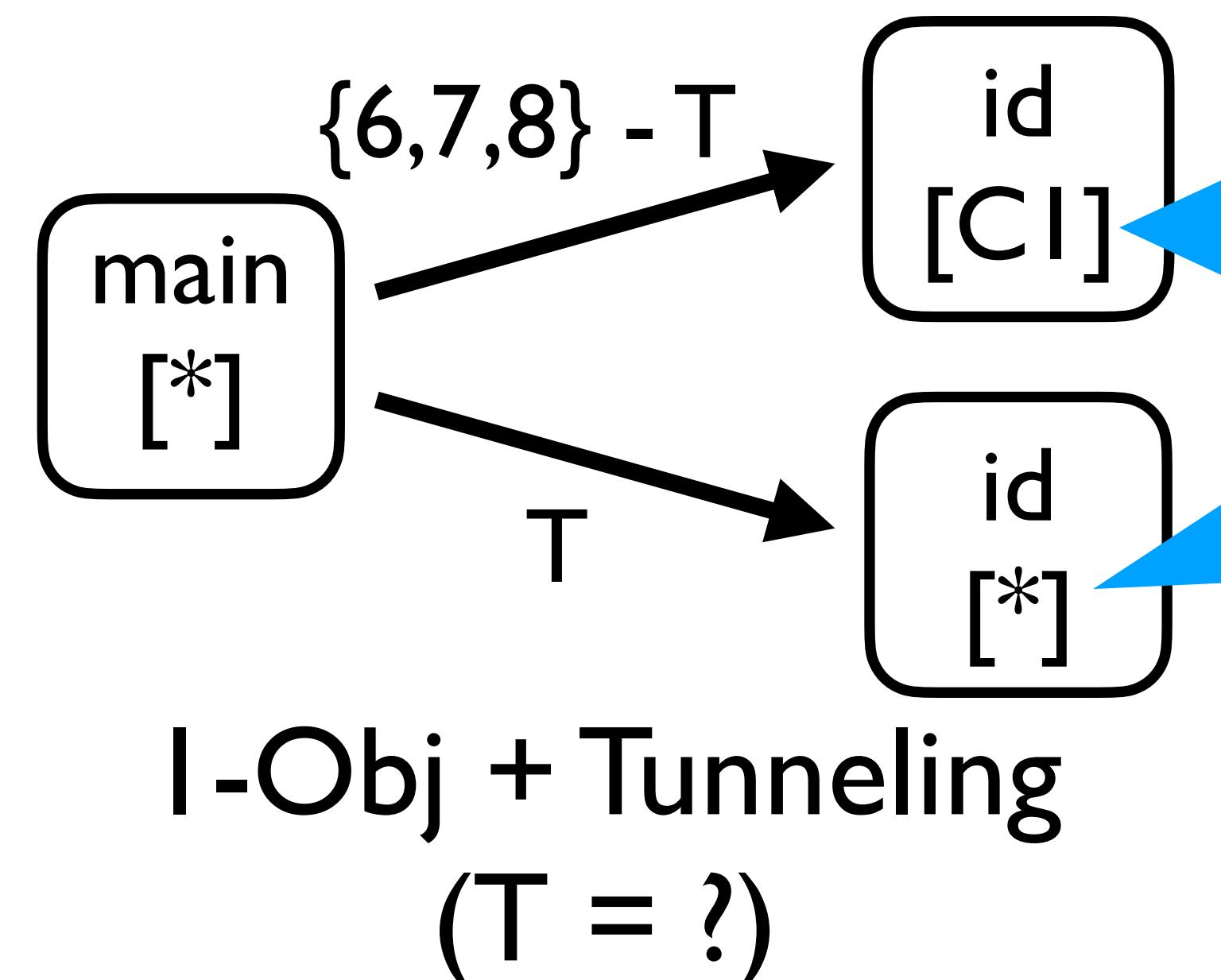


Call-graph of I-Obj with tunneling T

Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation

```
0: class C{  
1:   id(v){  
2:     return v;}  
3: }  
4: main(){  
5:   cl = new C(); //CI  
6:   a = (A) cl.id(new A());  
7:   b = (B) cl.id(new B());  
8:   c = (C) cl.id(new C());  
9: }
```

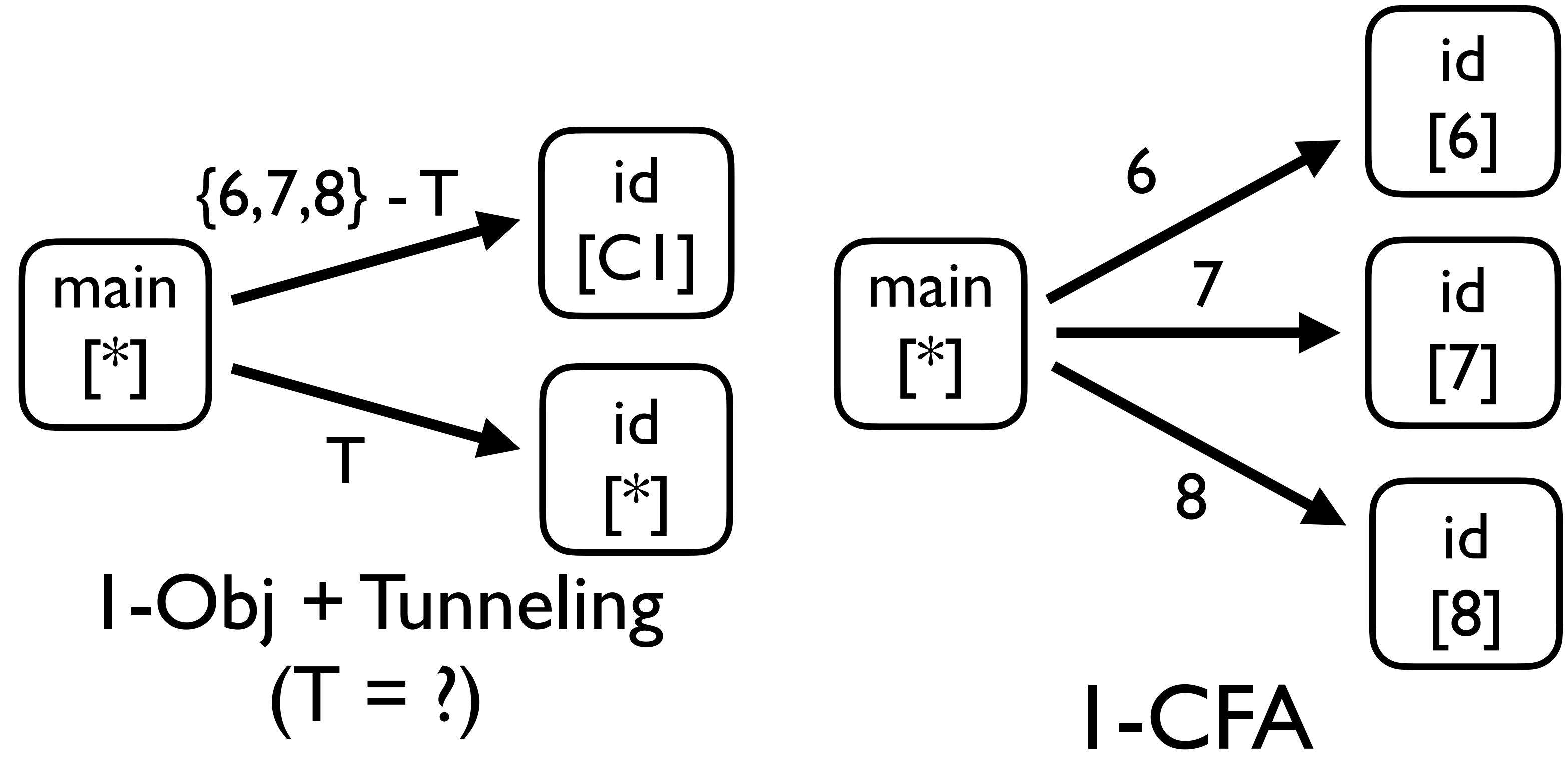


Unable to separate the
three method calls with the
two contexts

Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation

```
0: class C{  
1:   id(v){  
2:     return v;}  
3: }  
4: main(){  
5:   cl = new C(); //CI  
6:   a = (A) cl.id(new A());  
7:   b = (B) cl.id(new B());  
8:   c = (C) cl.id(new C());  
9: }
```



Call-site sensitivity easily separates the three method calls

Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation.

Observation

When context tunneling is included

- Limitation of call-site sensitivity is **removed**
- Limitation of object sensitivity is **not removed**

```
0: c
1:
2: }
3:
4: n
5:
6: a
7: b = (B) cl.id(new B());
8: c = (B) cl.id(new C());
9: }
```

Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation.

Observation

When context tunneling is included

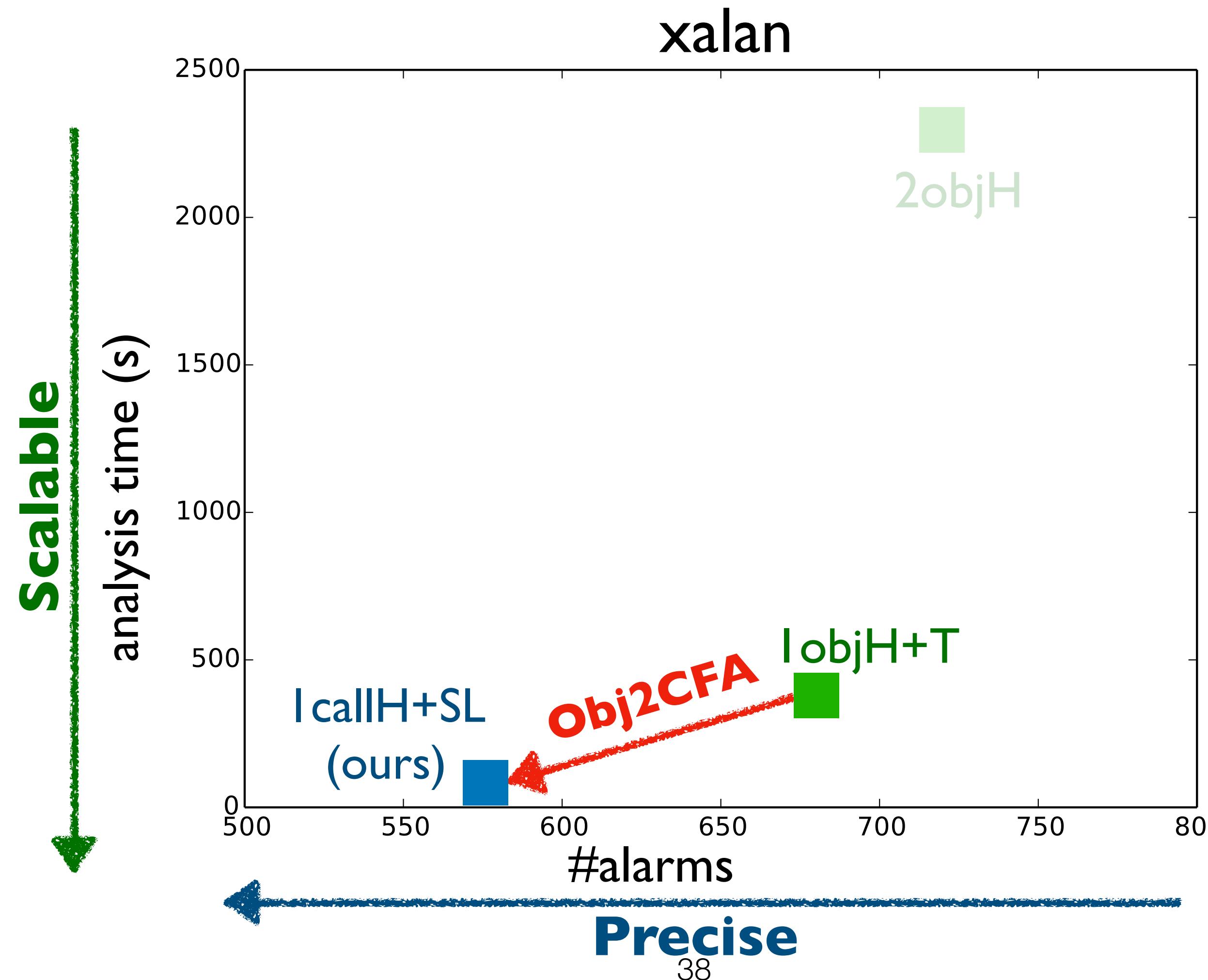
- Limitation of call-site sensitivity is **removed**
- Limitation of object sensitivity is **not removed**

Our claim

If context tunneling is included,
call-site sensitivity is more precise than object sensitivity

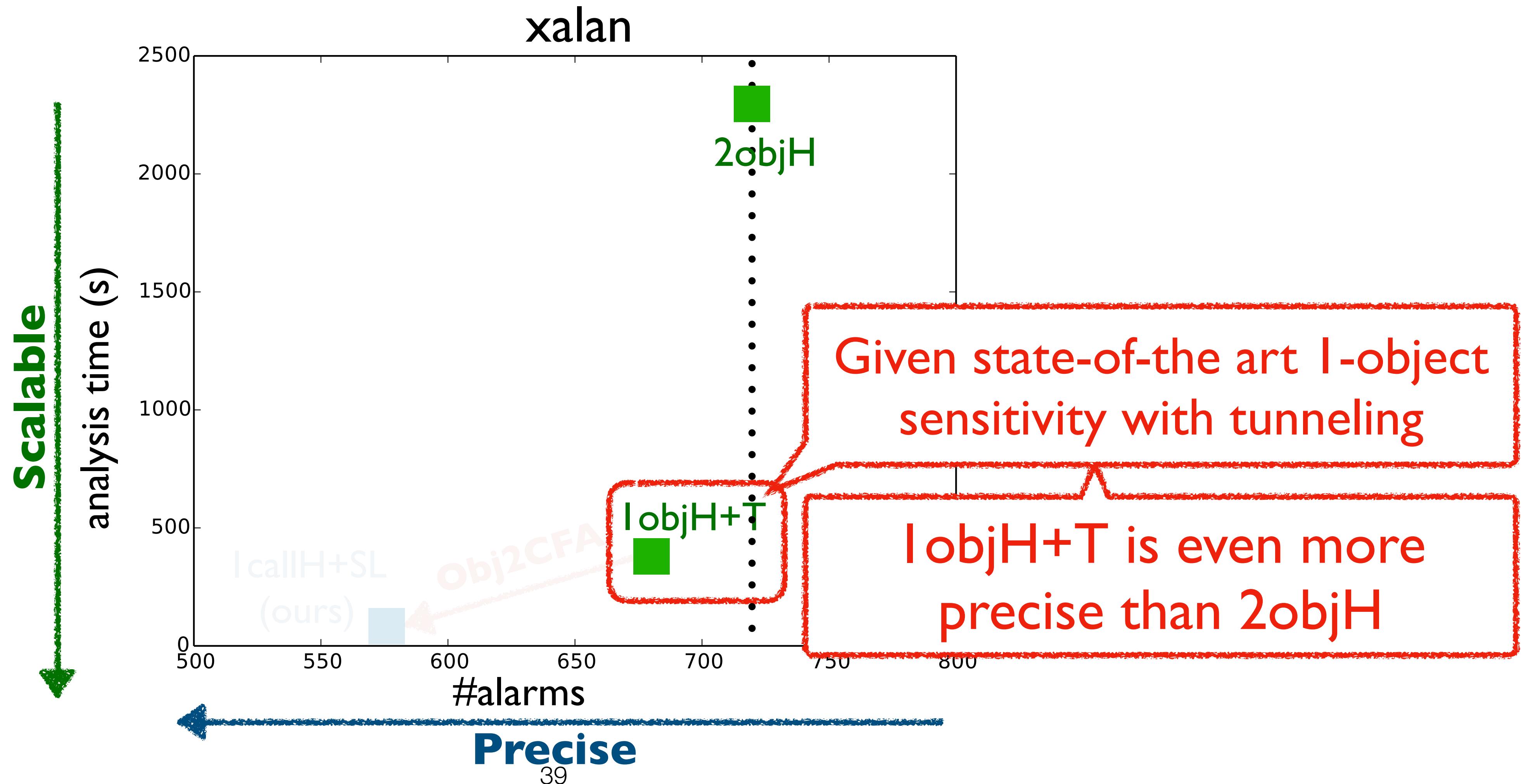
Our Technique : **Obj2CFA**

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



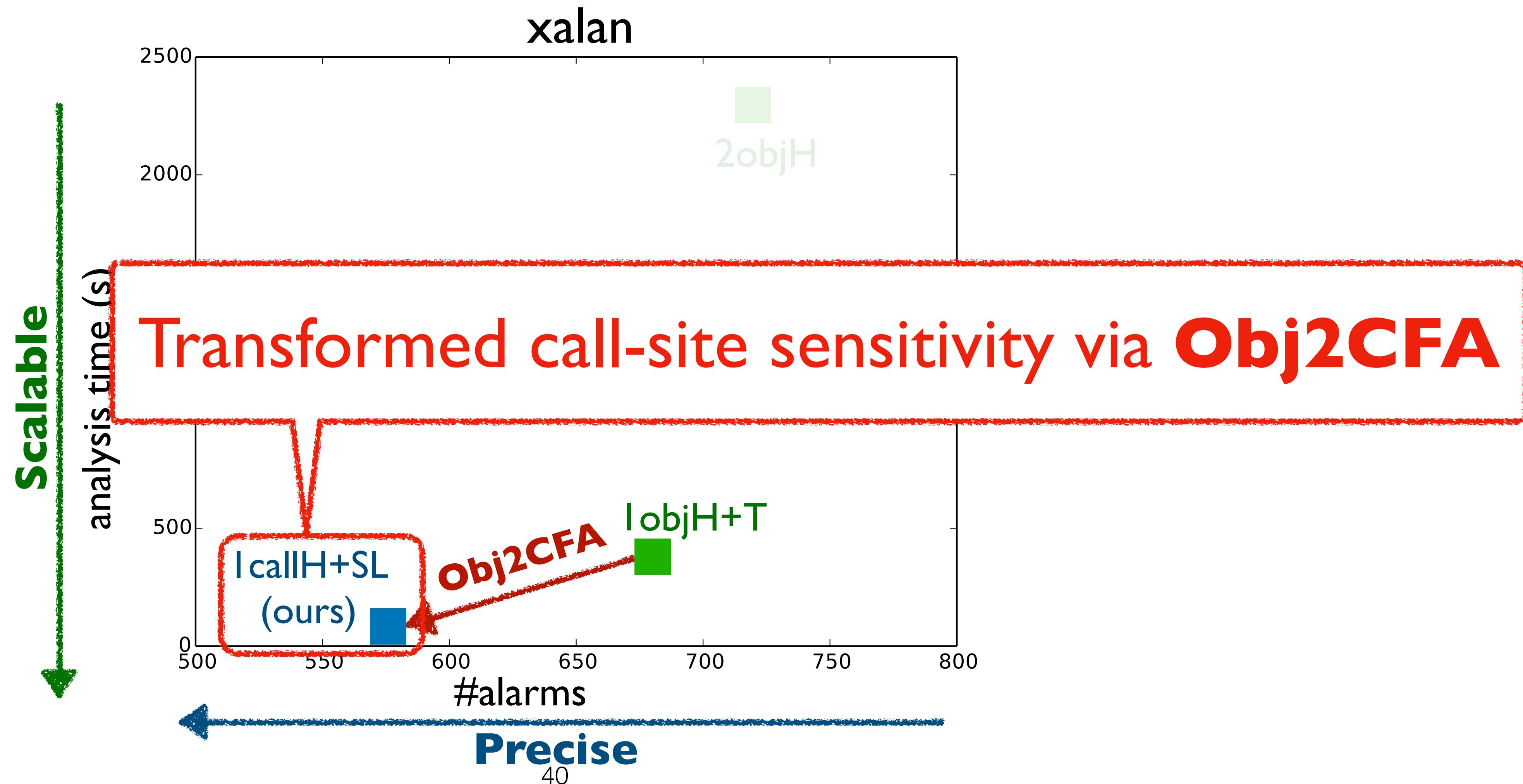
Our Technique : **Obj2CFA**

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



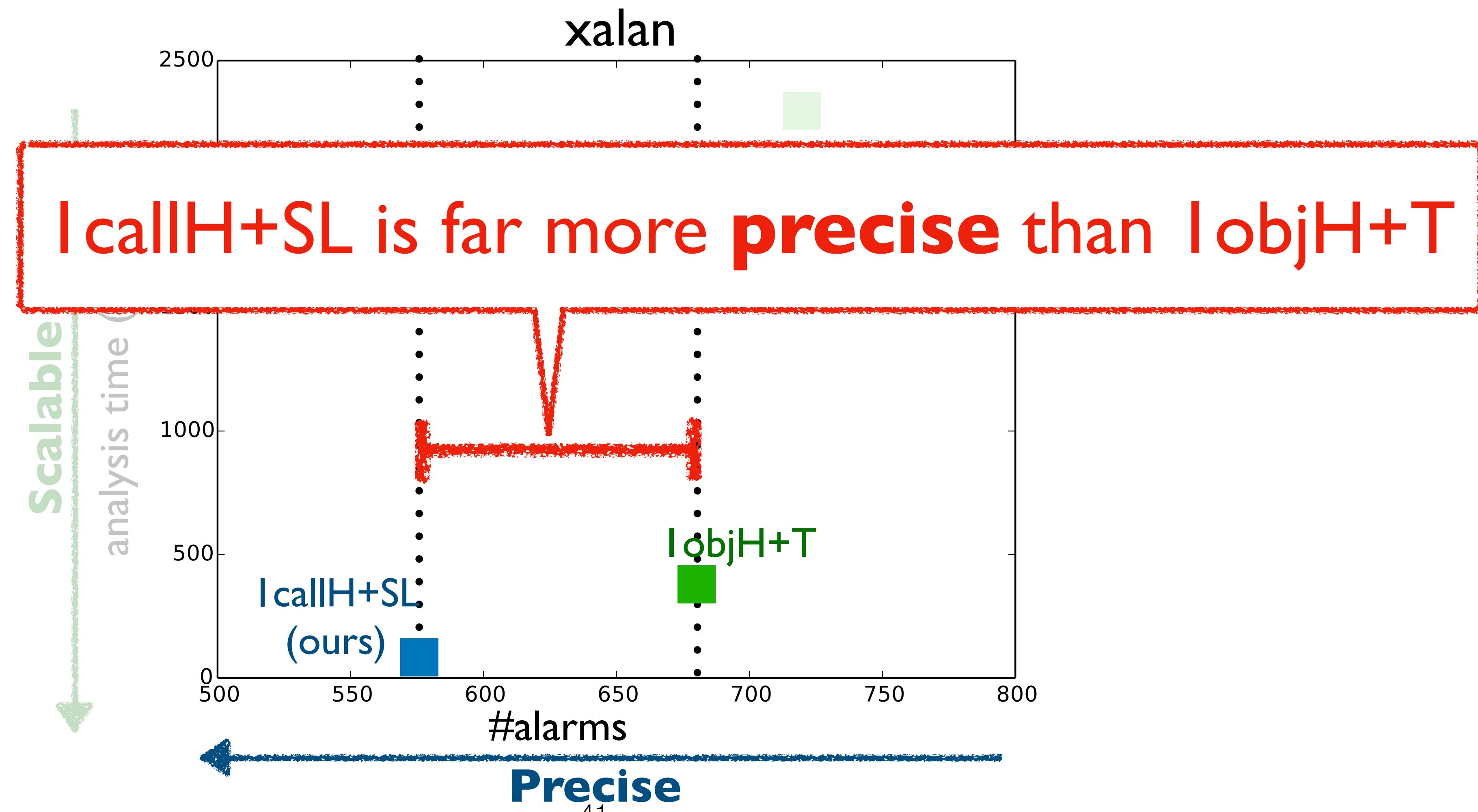
Our Technique : **Obj2CFA**

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



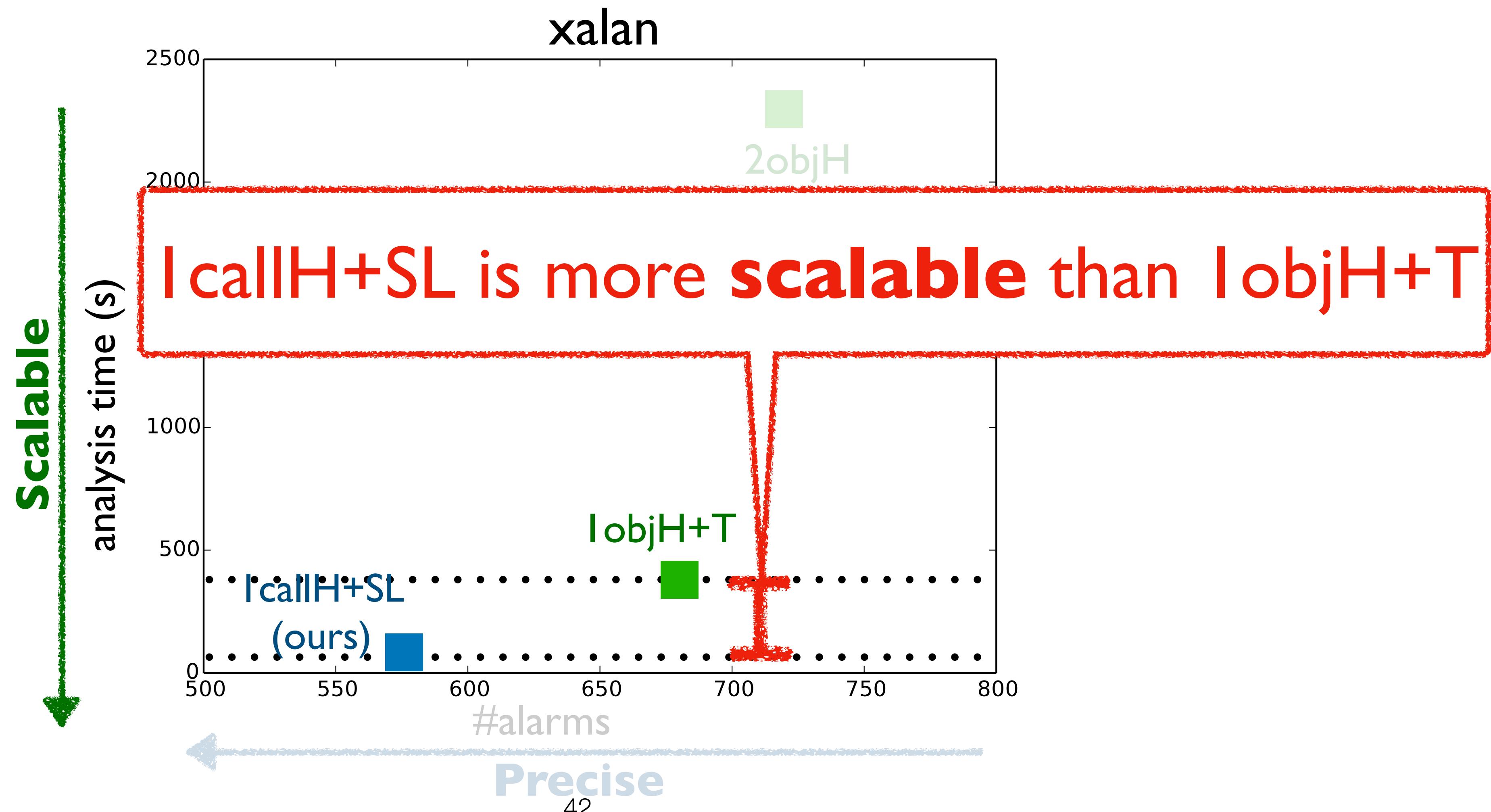
Our Technique : **Obj2CFA**

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



Our Technique : **Obj2CFA**

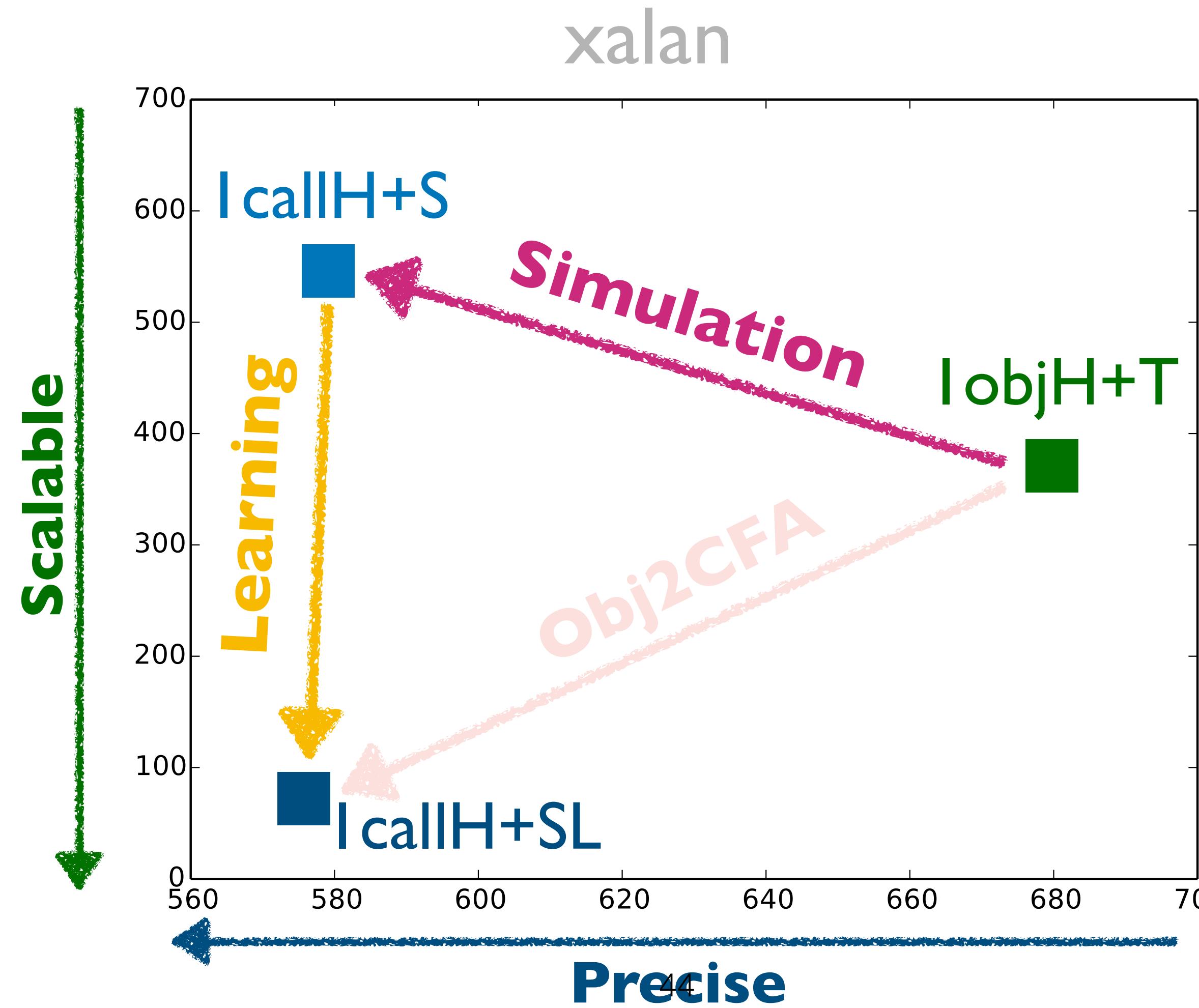
- **Obj2CFA** transforms a given **object sensitivity** into a more precise **CFA**



Detail of Obj2CFA

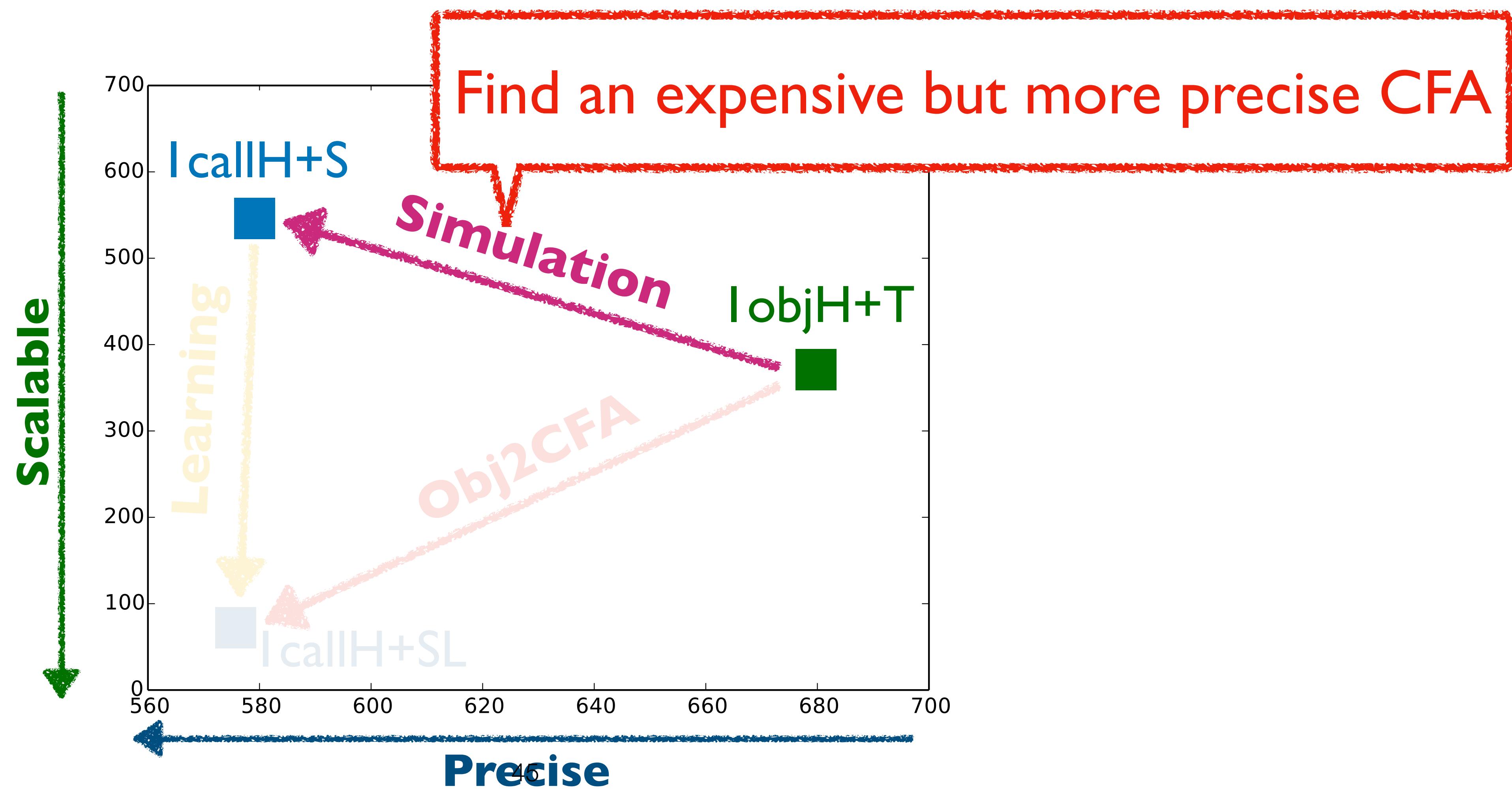
Our Technique : Obj2CFA

- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



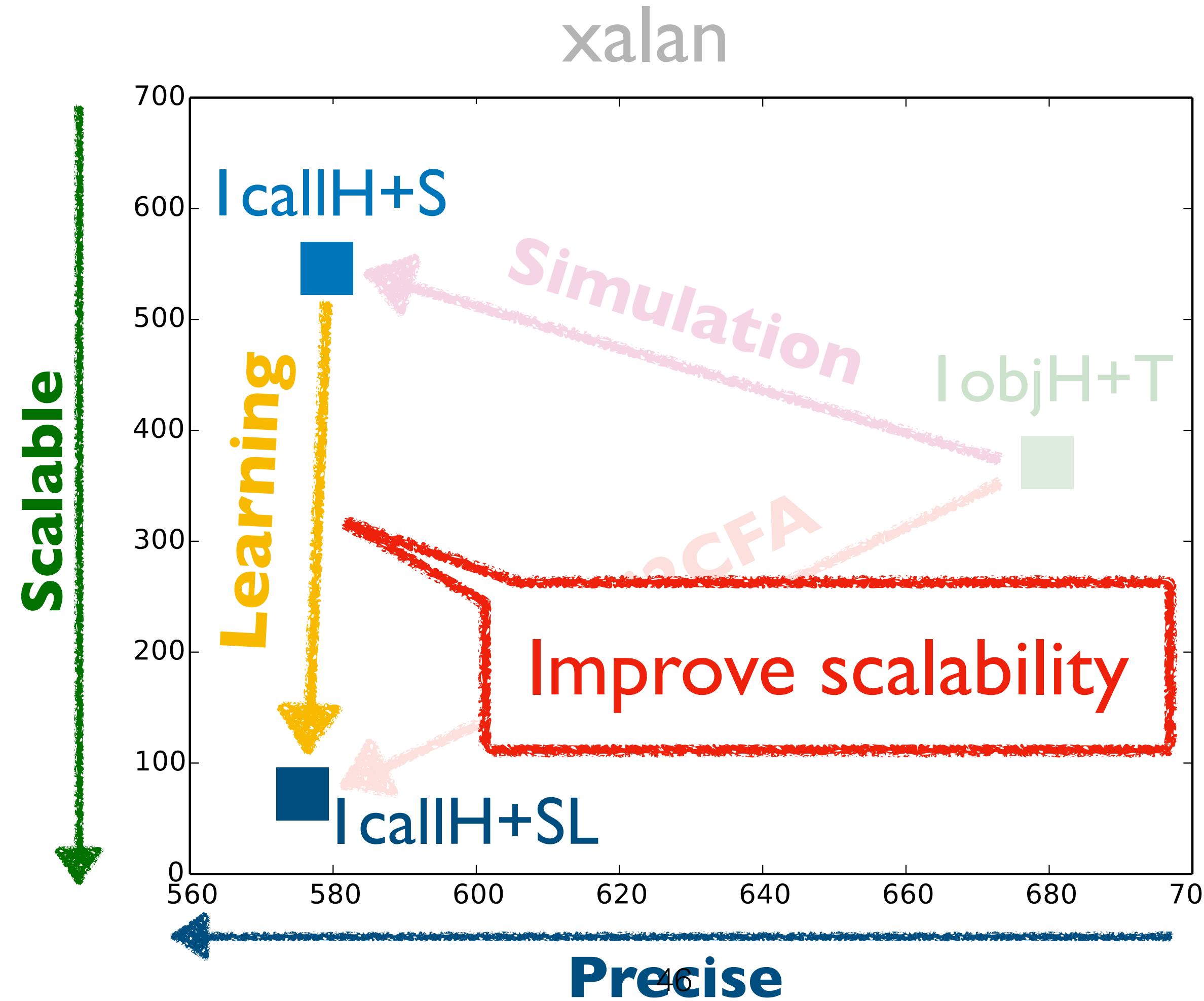
Our Technique : Obj2CFA

- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



Our Technique : Obj2CFA

- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



Technique I: Simulation

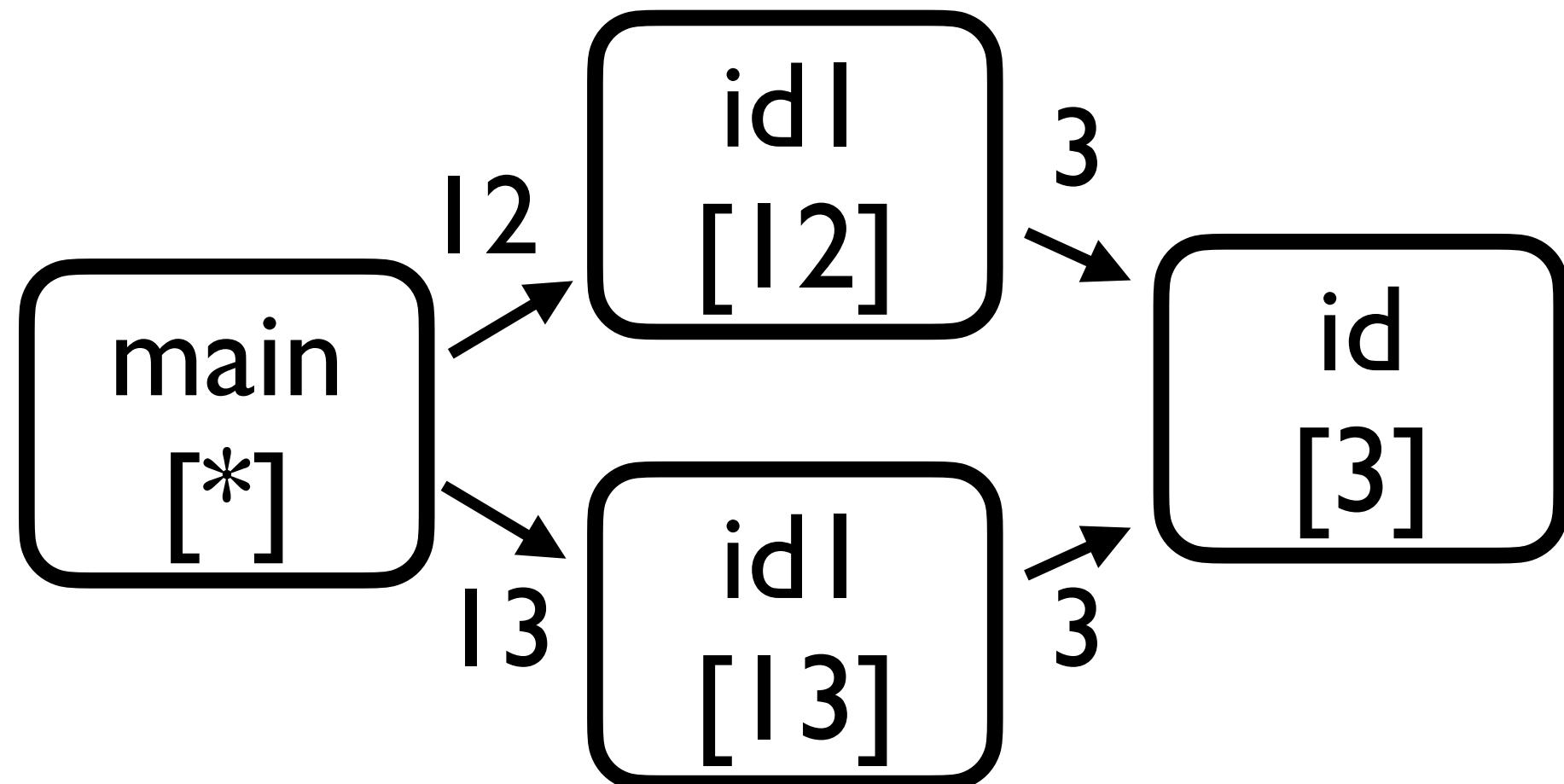
- Running example to illustrate Simulation

```
1: class C{  
2:   id(v){return v;}  
3:   idI(v){return id(v);}  
4:   foo(){  
5:     A a = (A) this.id(new A());}//query1  
6:     B b = (B) this.id(new B());}//query2  
7: }  
8: main(){  
9:   c1 = new C();//C1  
10:  c2 = new C();//C2  
11:  c3 = new C();//C3  
12:  A a = (A) c1.idI(new A());//query3  
13:  B b = (B) c2.idI(new B());//query4  
14:  c3.foo();  
15: }
```

Technique I: Simulation

- Running example to illustrate Simulation

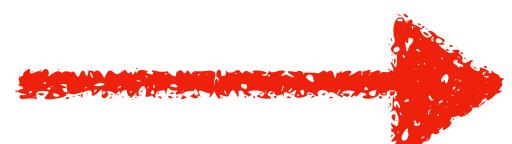
```
1: class C{  
2:     id(v){return v;}  
3:     idI(v){return id(v);} idI(v){return id(v);} → Limitation of conventional I-CFA  
4:     foo(){  
5:         A a = (A) this.id(new A());//query1  
6:         B b = (B) this.id(new B());//query2  
7:     }  
8:     main(){  
9:         c1 = new C();//C1  
10:        c2 = new C();//C2  
11:        c3 = new C();//C3  
12:        A a = (A) c1.idI(new A());//query3 A a = (A) c1.idI(new A());//query3  
13:        B b = (B) c2.idI(new B());//query4 B b = (B) c2.idI(new B());//query4  
14:        c3.foo();  
15:    }
```



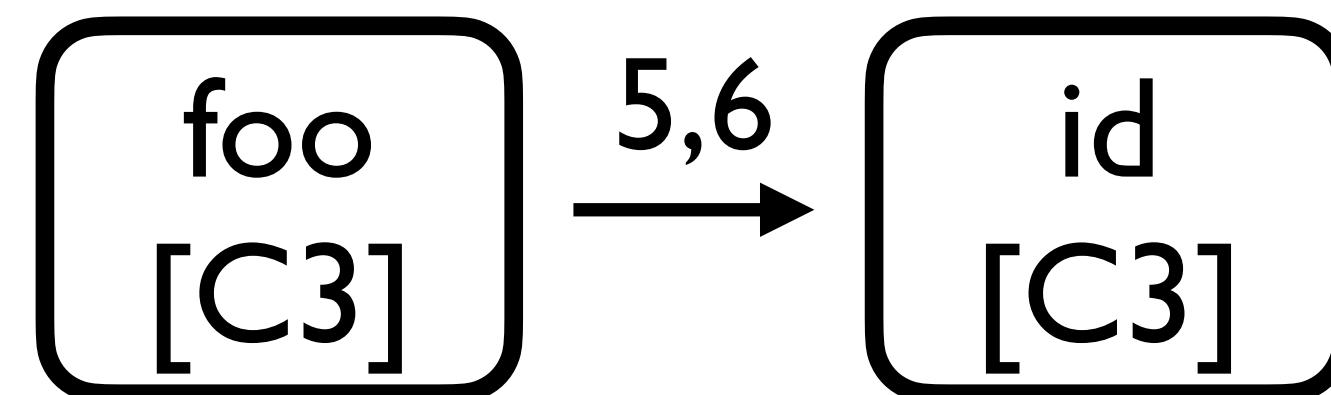
Technique I: Simulation

- Running example to illustrate Simulation

```
1: class C{  
2:   id(v){return v;}  
3:   idI(v){return id(v);}  
4:   foo(){  
5:     A a = (A) this.id(new A());}//query1  
6:     B b = (B) this.id(new B());}//query2  
7: }  
8: main(){  
9:   c1 = new C();//C1  
10:  c2 = new C();//C2  
11:  c3 = new C();//C3  
12:  A a = (A) c1.idI(new A());//query3  
13:  B b = (B) c2.idI(new B());//query4  
14:  c3.foo();  
15: }
```



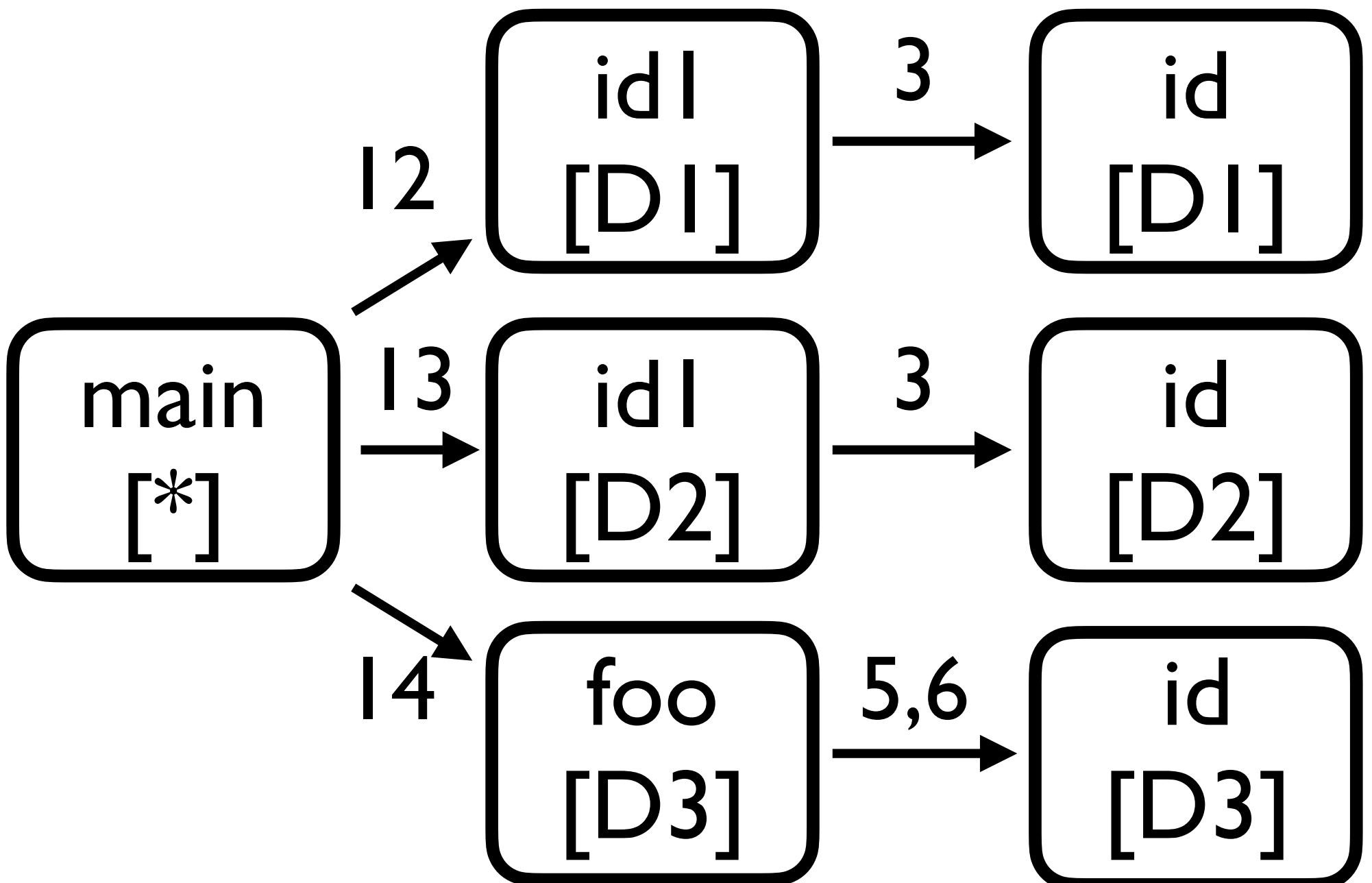
Limitation of object sensitivity



Technique I: Simulation

- Given **object sensitivity** is conventional 1-object sensitivity (e.g., $T = \emptyset$)

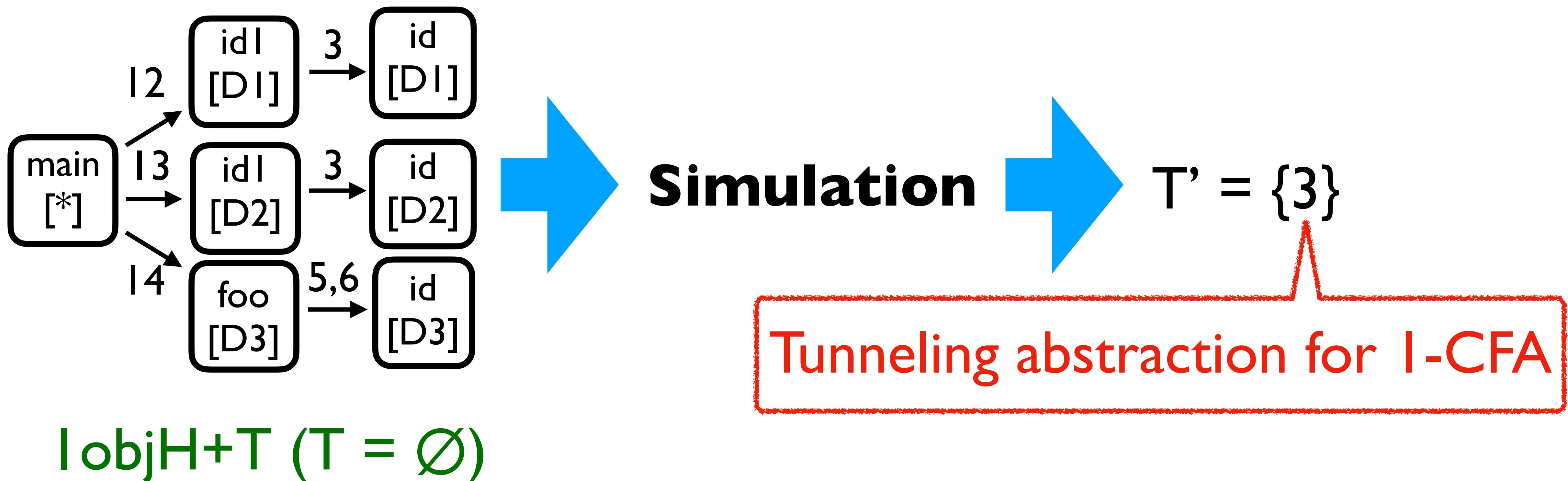
```
I: class C{  
2:   id(v){return v;}  
3:   idI(v){return id(v);}  
4:   foo(){  
5:     A a = (A) this.id(new A());}//query1  
6:     B b = (B) this.id(new B());}//query2  
7: }  
8: main(){  
9:   c1 = new C();//C1  
10:  c2 = new C();//C2  
11:  c3 = new C();//C3  
12:  A a = (A) c1.idI(new A());//query3  
13:  B b = (B) c2.idI(new B());//query4  
14:  c3.foo();  
15: }
```



IobjH+T ($T = \emptyset$)

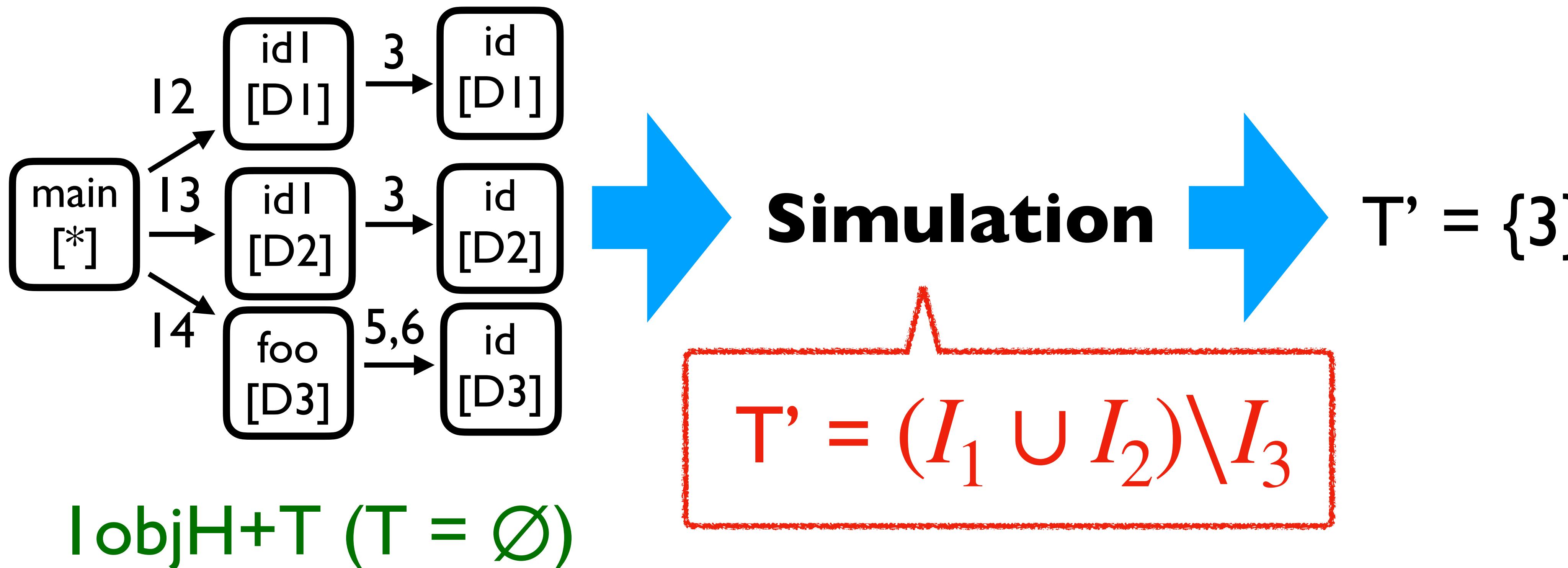
Technique I: Simulation

- **Simulation** takes a call-graph and produce a tunneling abstraction for CFA



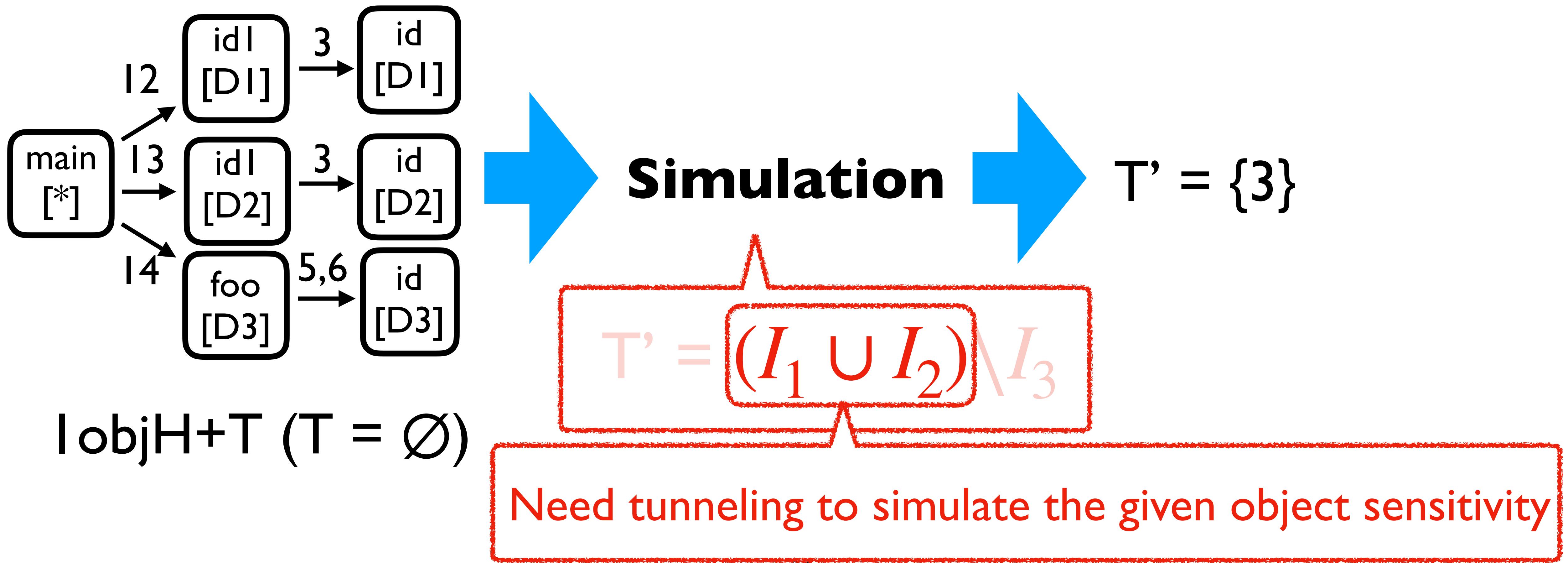
Technique I: Simulation

- **Simulation** takes a call-graph and produce a tunneling abstraction for CFA



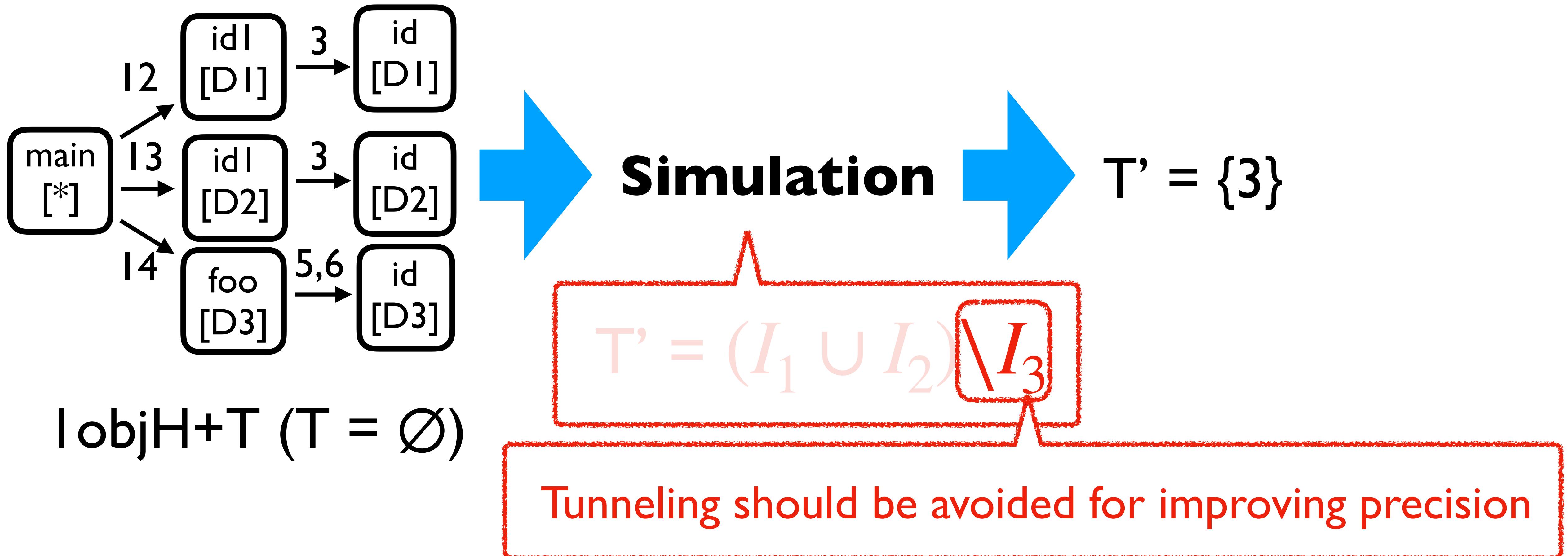
Technique I: Simulation

- **Simulation** takes a call-graph and produce a tunneling abstraction for CFA



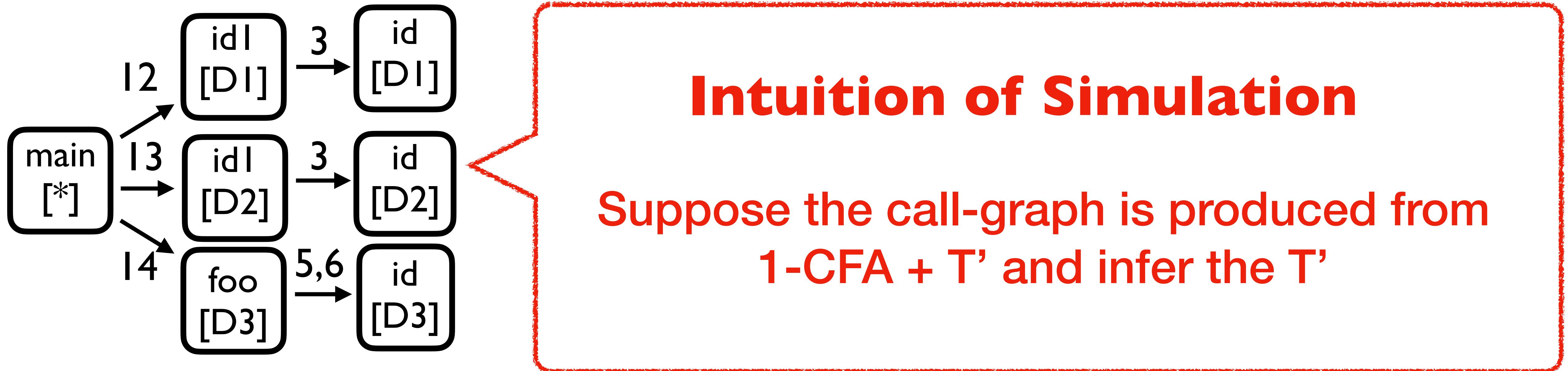
Technique I: Simulation

- **Simulation** takes a call-graph and produce a tunneling abstraction for CFA



Technique I: Simulation

- **Simulation** takes a call-graph and produce a tunneling abstraction for CFA



IobjH+T ($T = \emptyset$)

IcallH+T'

What is T'?

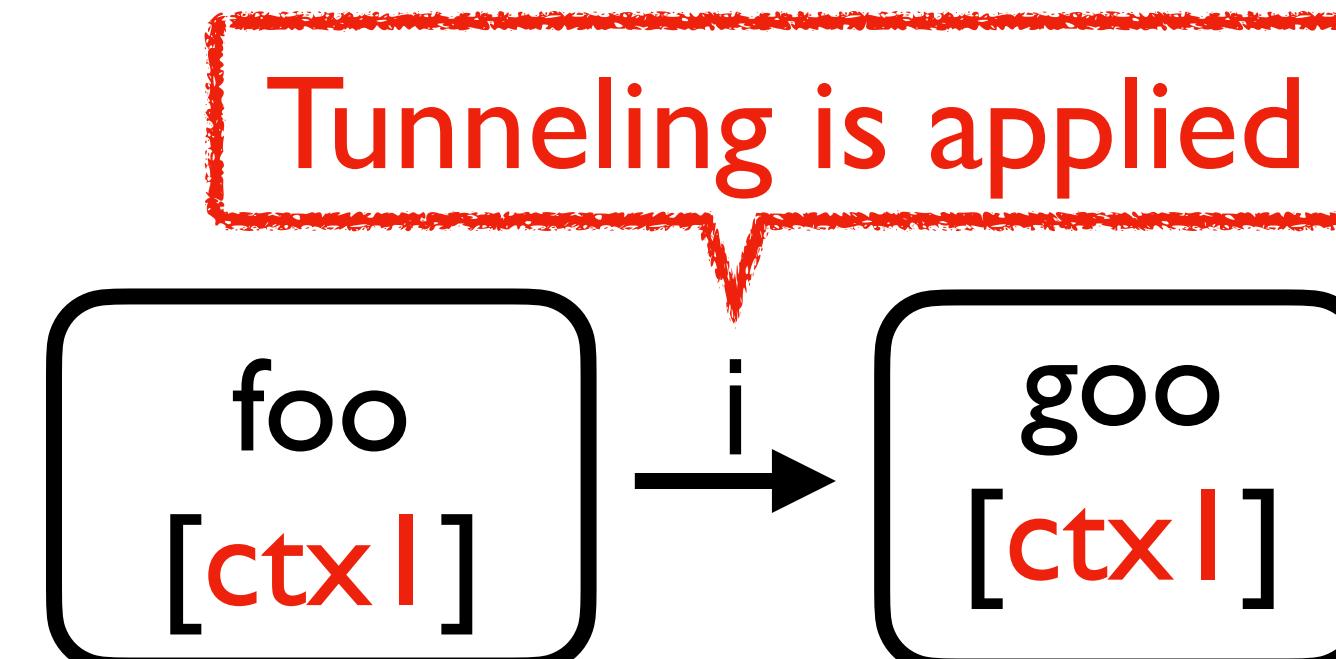
Intuition Behind Simulation ($I_1 \cup I_2$)

- If tunneling is applied to i , two properties inevitably appear at i

We track the two properties to find the T'

Intuition Behind Simulation ($I_1 \cup I_2$)

- If tunneling is applied to i, two properties inevitably appear at i



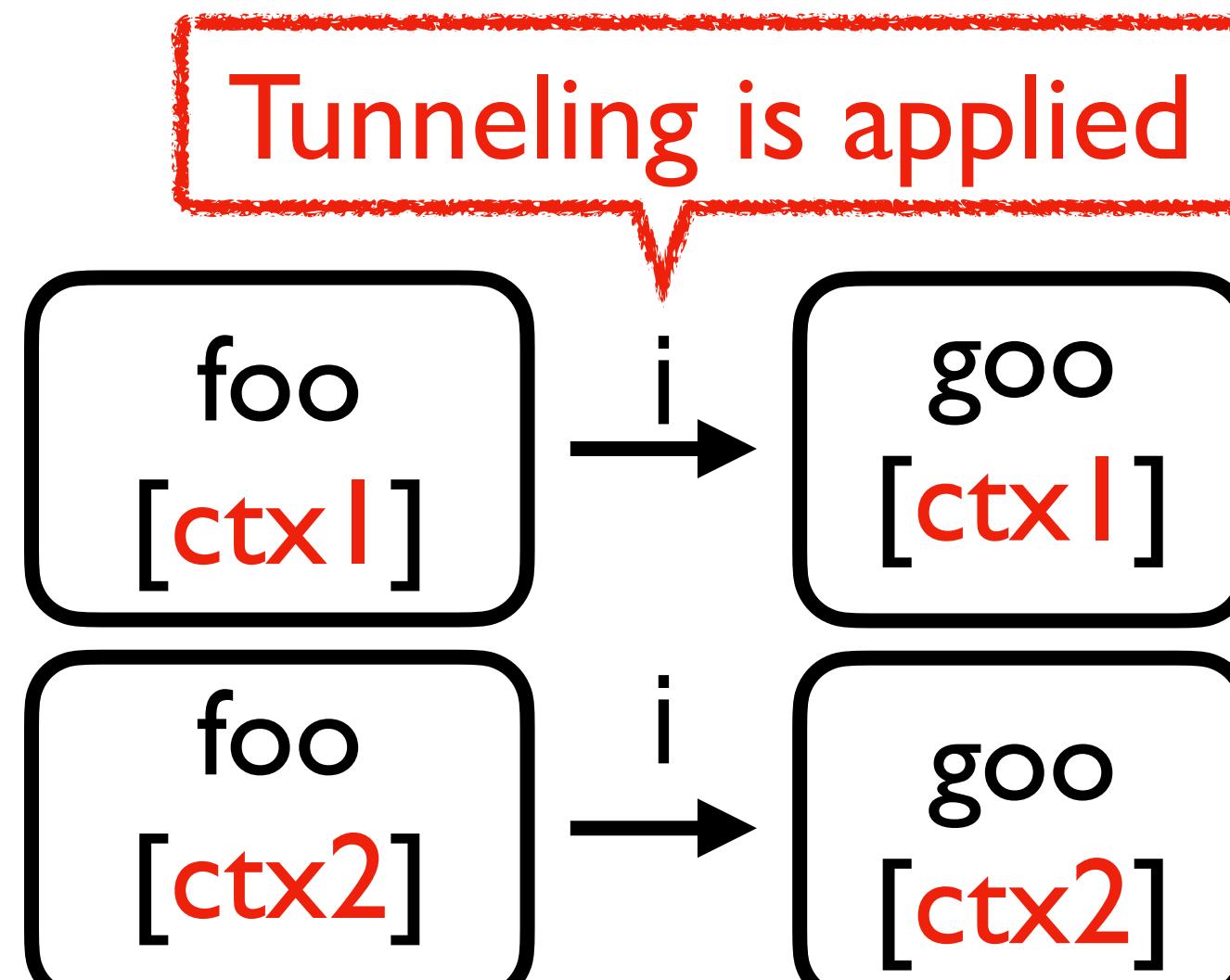
Property of context tunneled call-sites

I_1

- Property I: caller and callee methods have the **same context**

Intuition Behind Simulation ($I_1 \cup I_2$)

- If tunneling is applied to i, two properties inevitably appear at i



Property of context tunneled invocations

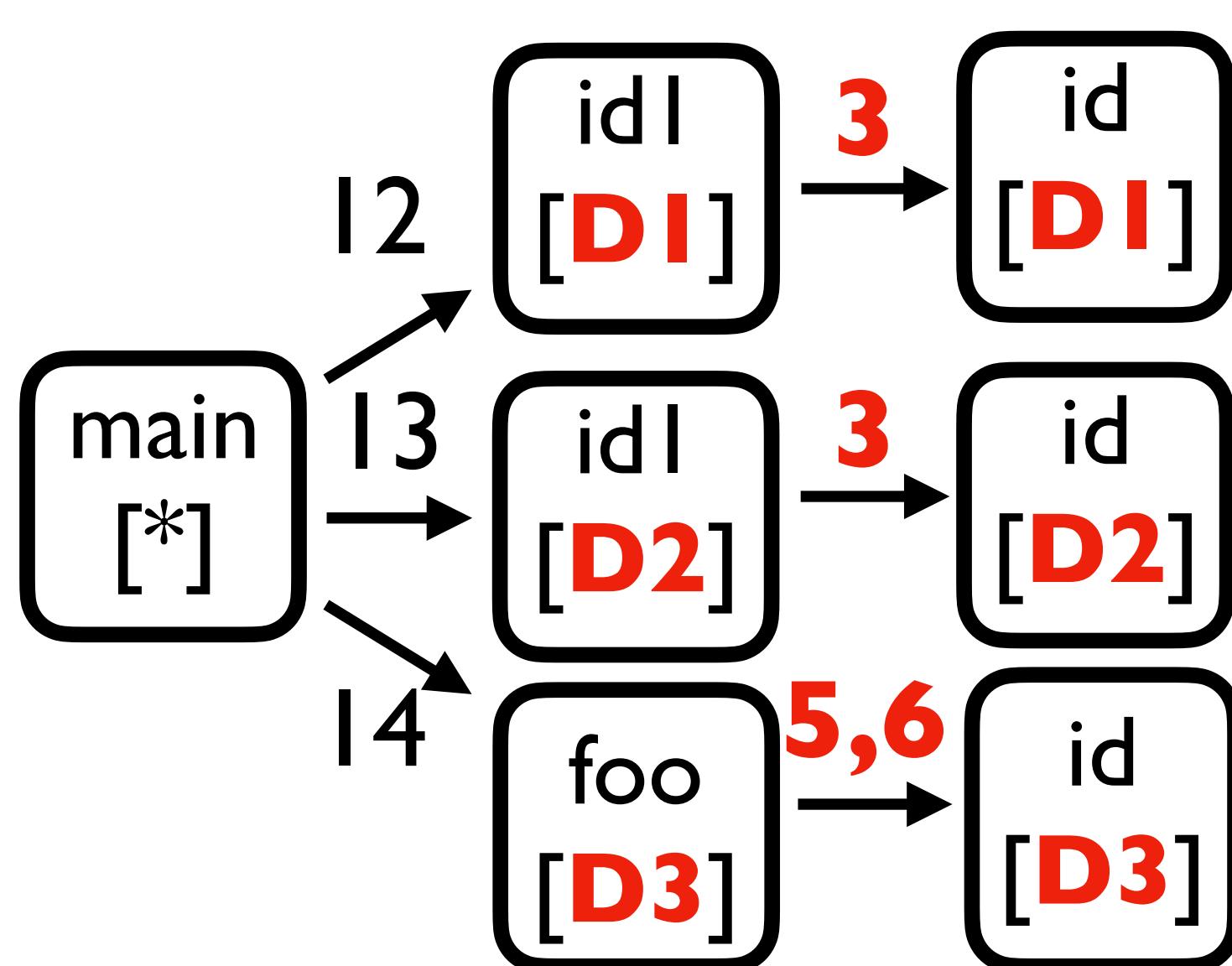
- Property 2: different caller contexts imply different callee contexts

I_1

I_2

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose given call-graph is produced from IcallH+T' and infer what T' is



- I_1 : caller and callee methods have the **same context**

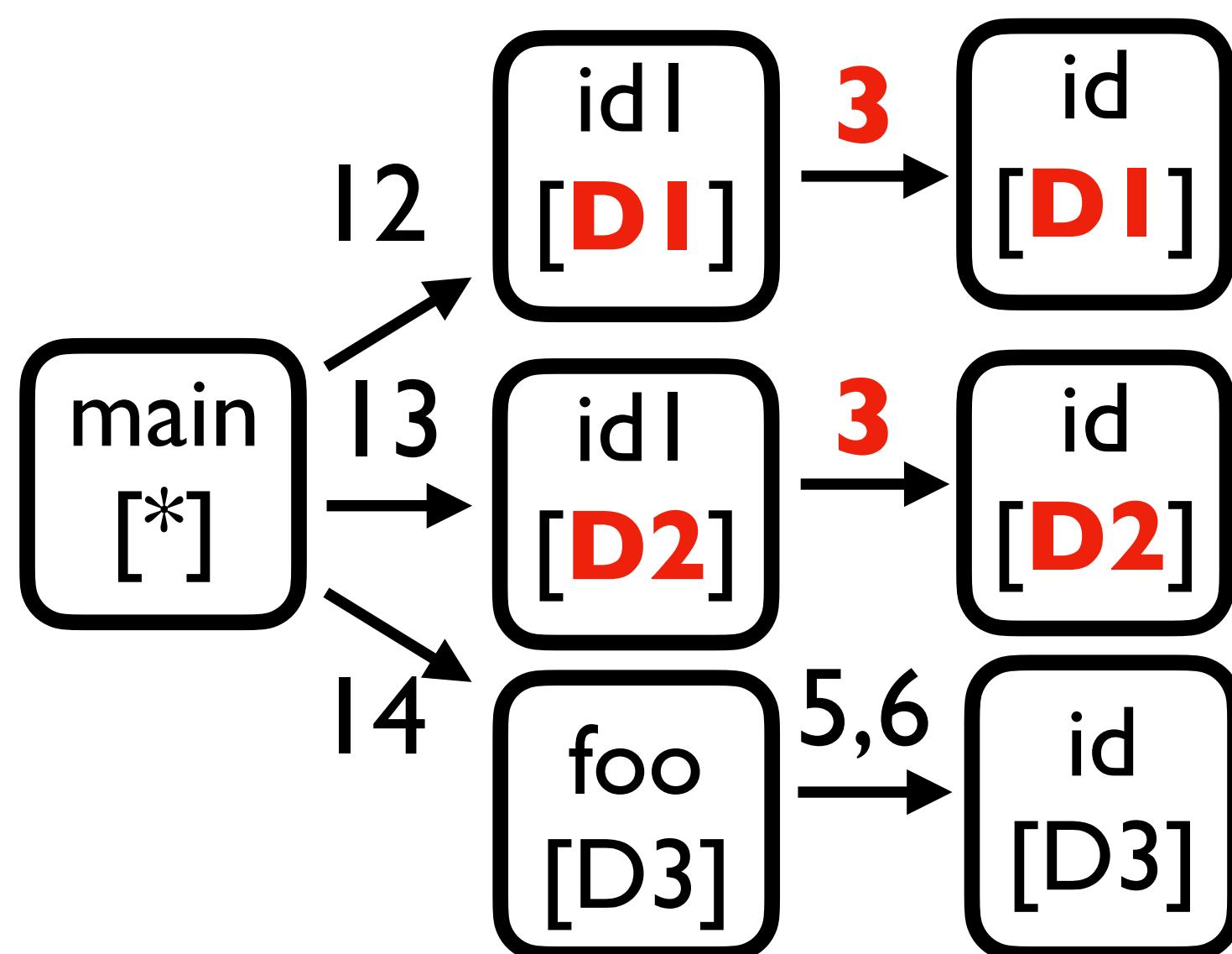
$$I_1 = \{3, 5, 6\}$$

~~IobjH+T ($T = \emptyset$)~~

IcallH+T'  What is T' ?

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose given call-graph is produced from IcallH+T' and infer what T' is



- I_1 : caller and callee methods have the **same context**

$$I_1 = \{3, 5, 6\}$$

- I_2 : different caller ctx imply different callee ctx

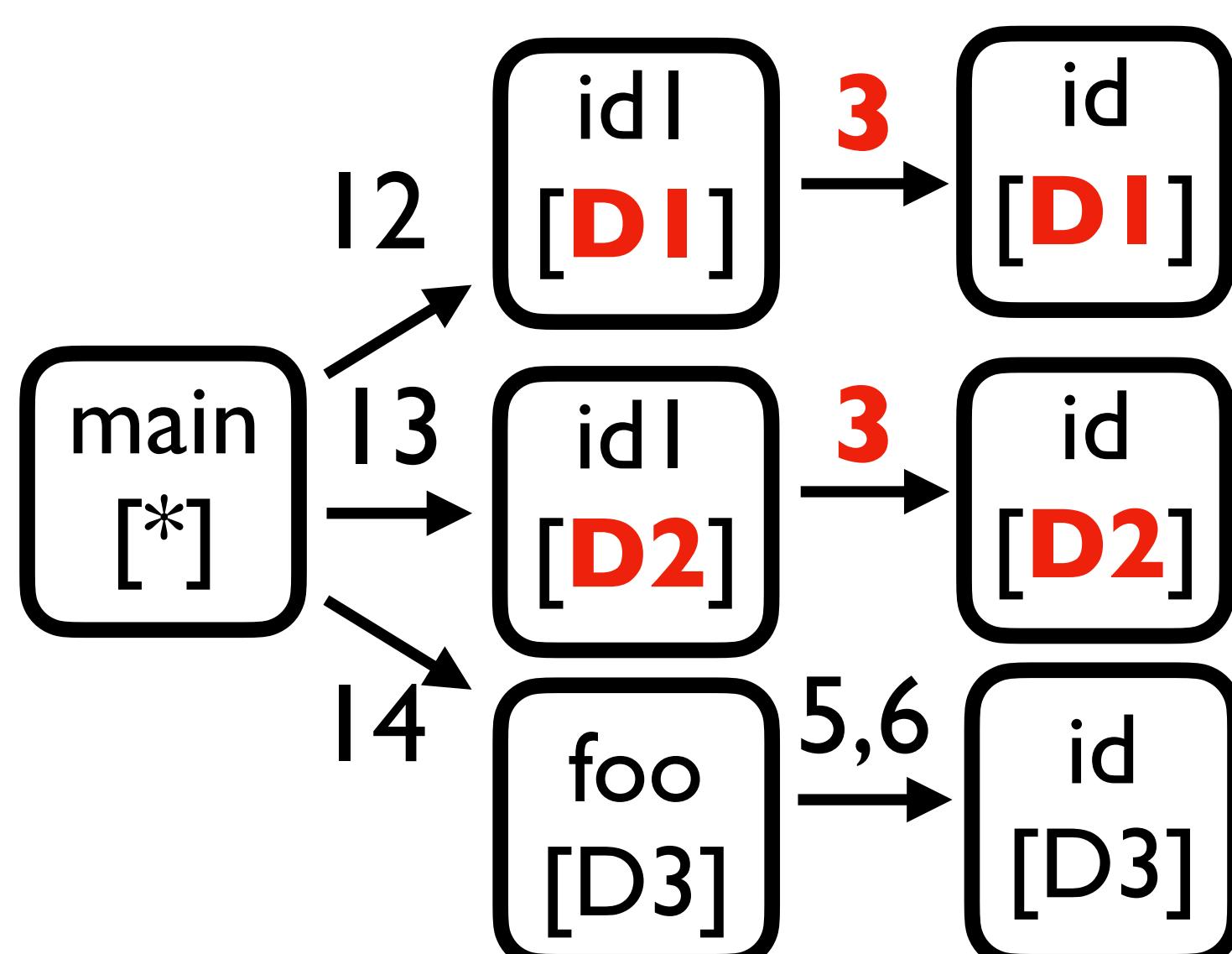
$$I_2 = \{3\}$$

~~$\text{IobjH+T} (T = \emptyset)$~~

IcallH+T' What is T' ?

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose given call-graph is produced from IcallH+T' and infer what T' is



- I_1 : caller and callee methods have the **same context**

$$I_1 = \{3, 5, 6\}$$

- I_2 : different caller ctx imply different callee ctx

$$I_2 = \{3\}$$

~~$\text{IobjH+T} (T = \emptyset)$~~

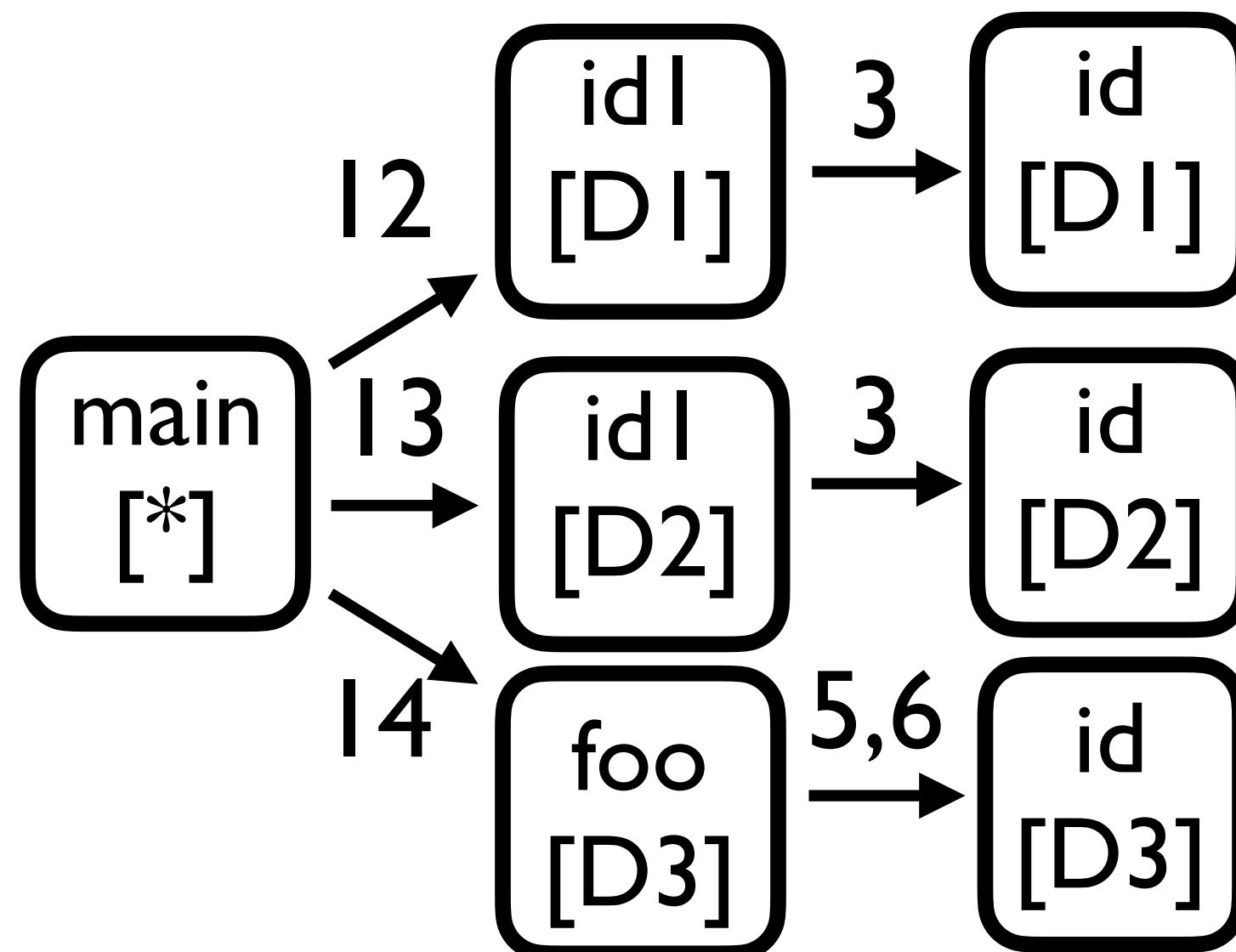
IcallH+T'

What is T' ?

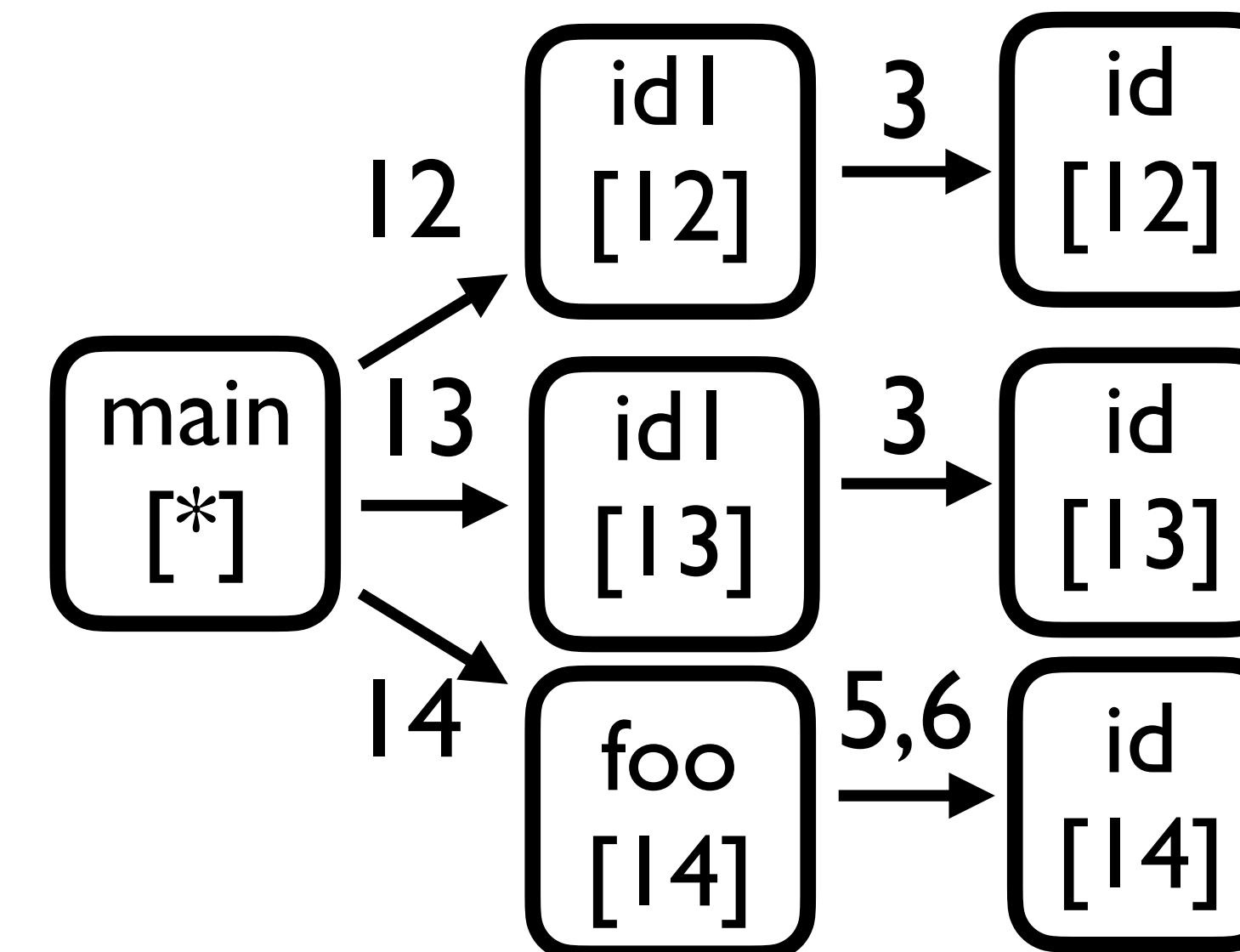
$$T' = I_1 \cup I_2 = \{3, 5, 6\}$$

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose given call-graph is produced from $I_{\text{call}}H+T'$ and infer what T' is



$I_{\text{obj}}H+T$ ($T = \emptyset$)

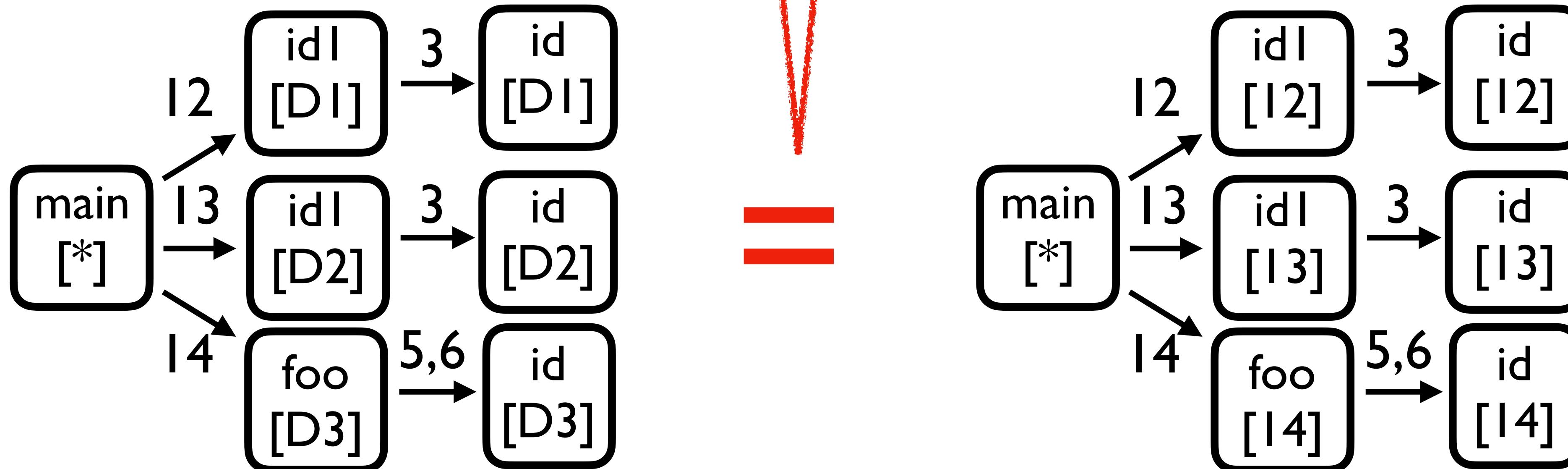


$I_{\text{call}}H+T'$ ($T' = \{3,5,6\}$)

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose given call-graph and infer what T' is

Exactly the same analyses

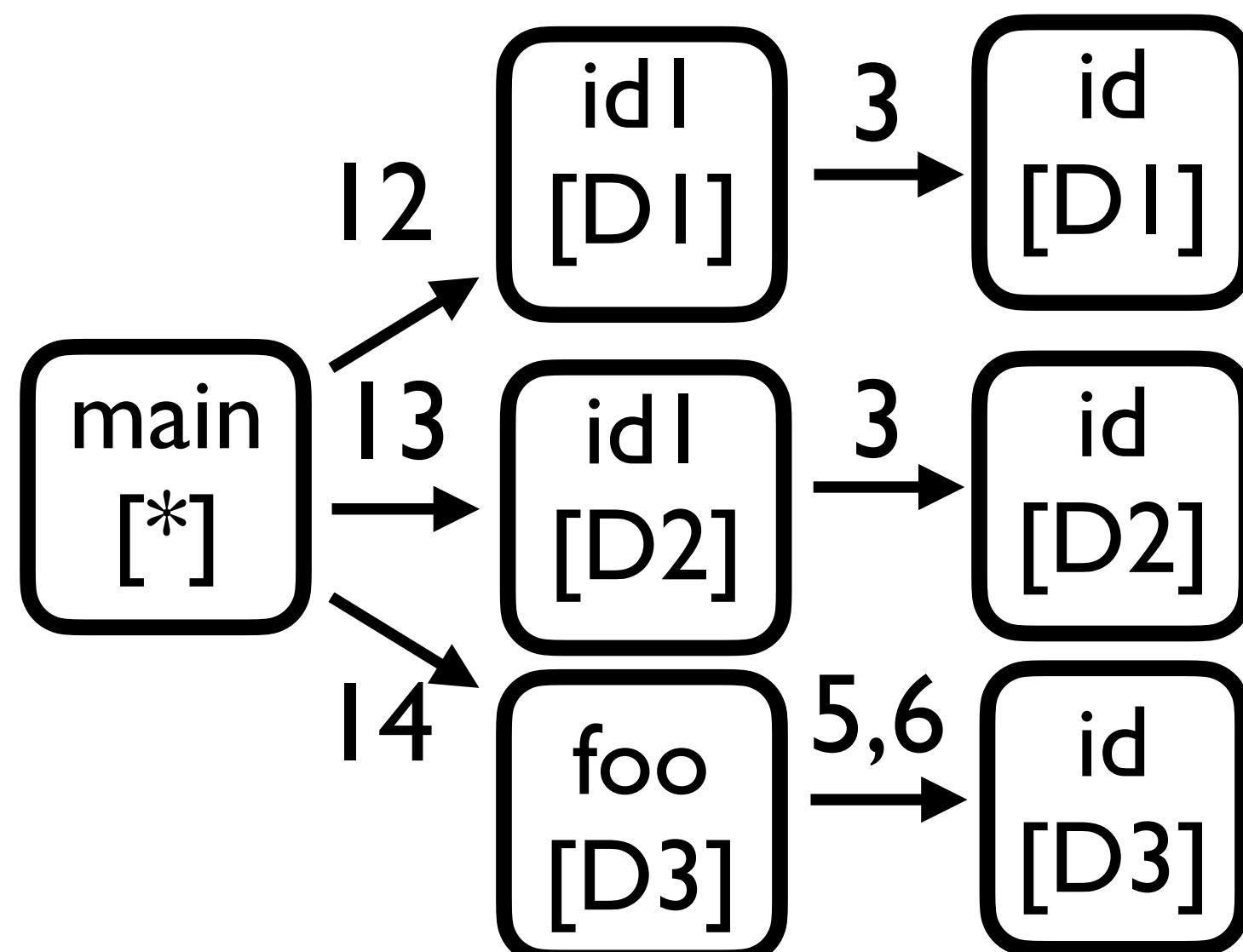


$I_{\text{objH+T}} (T = \emptyset)$

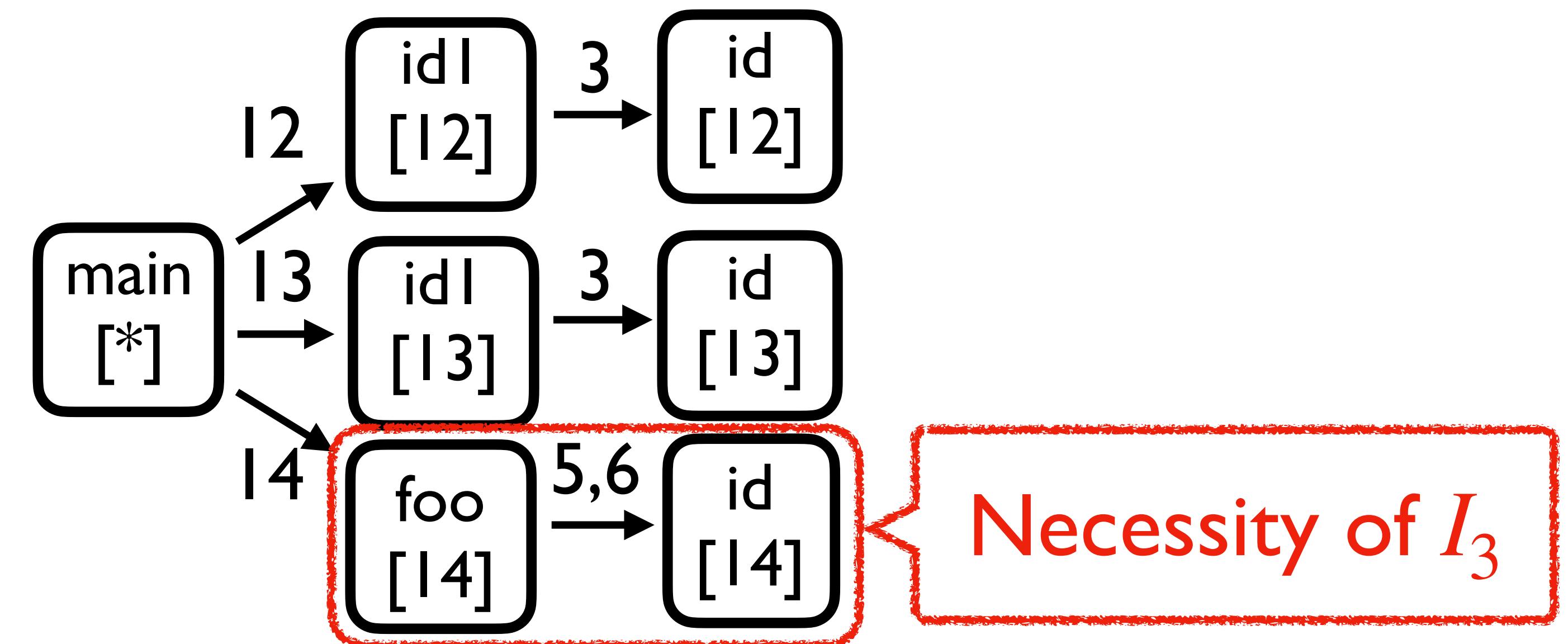
$I_{\text{callH+T'}} (T' = \{3,5,6\})$

Intuition Behind Simulation ($I_1 \cup I_2$)

- Suppose given call-graph is produced from $I_{\text{callH+T'}}$ and infer what T' is



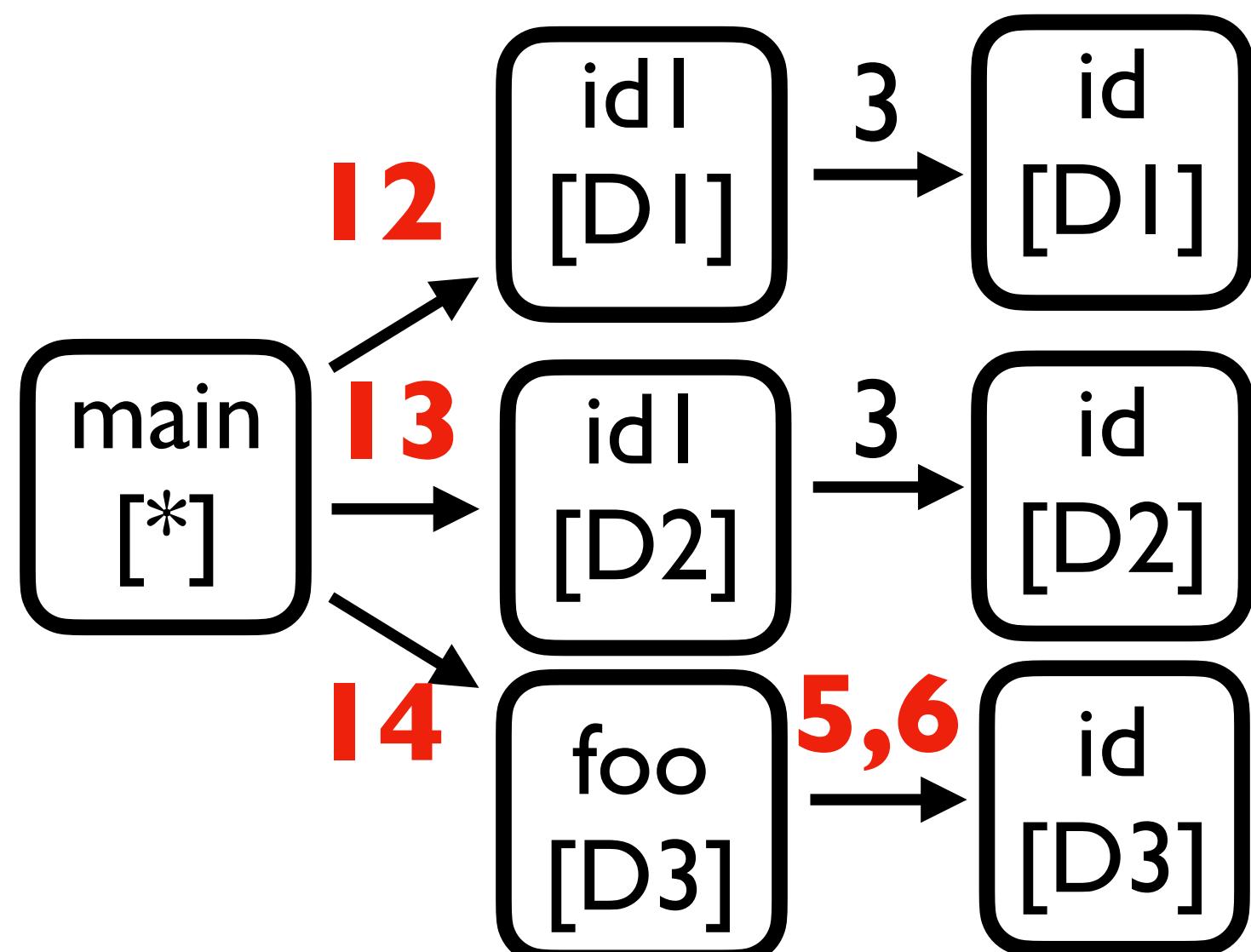
$I_{\text{objH+T}} (T = \emptyset)$



$I_{\text{callH+T'}} (T' = \{3,5,6\})$

Intuition Behind Simulation (I_3)

- I_3 : Tunneling should be avoided for improving precision



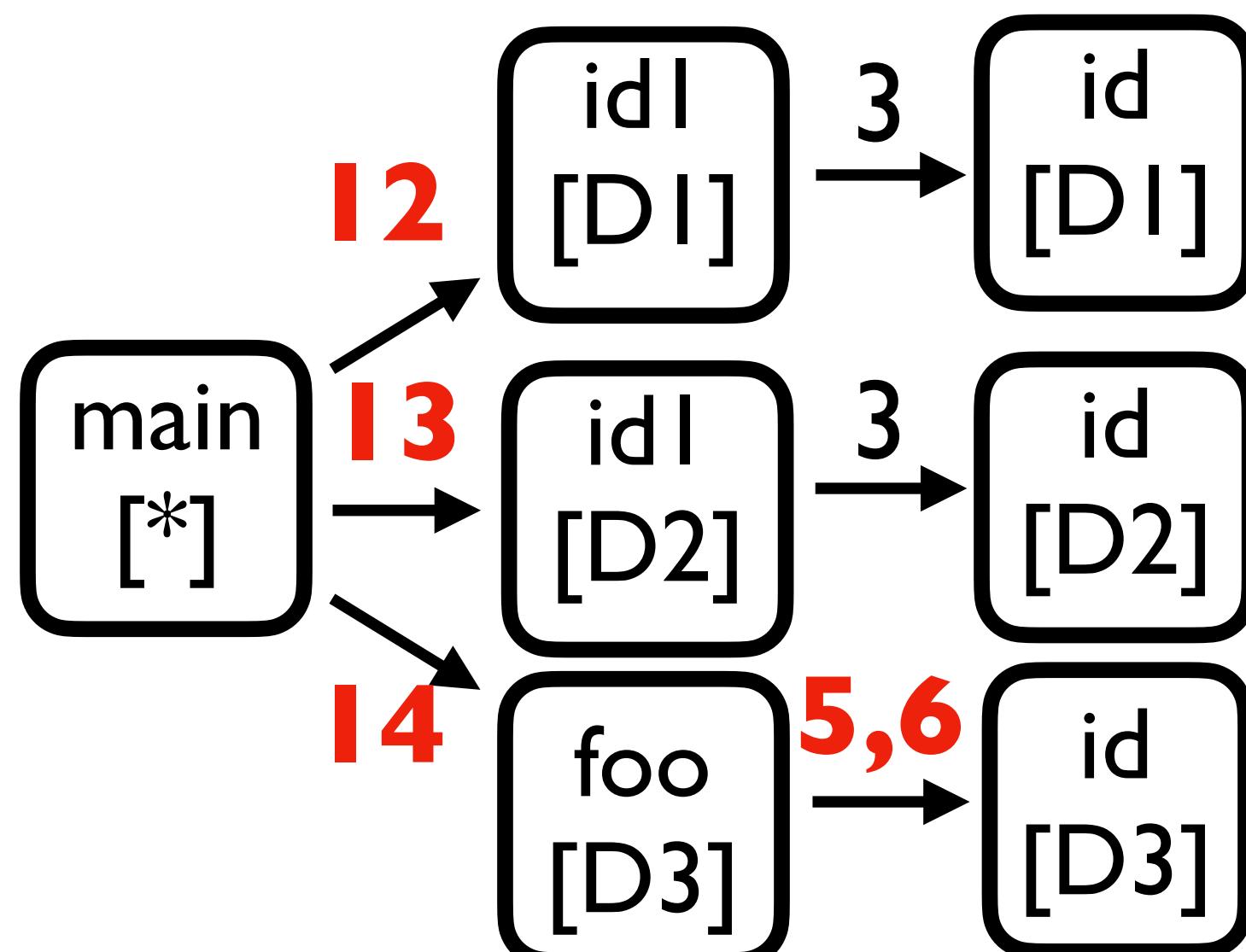
- I_1 : caller and callee methods have the **same context**
 $I_1 = \{3,5,6\}$
- I_2 : different caller ctx imply different callee ctx
 $I_2 = \{3\}$
- I_3 : given object sensitivity produced only one context

$\text{lobjH+T} (T = \emptyset)$

$$I_3 = \{5,6,12,13,14\}$$

Intuition Behind Simulation

- The inferred tunneling abstraction T' is a singleton set $\{3\}$



- I_1 : caller and callee methods have the **same context**
- I_2 : different caller ctx imply
- I_3 : given object sensitivity produced only one context

$$I_1 = \{3, 5, 6\}$$

$$I_2 = \{3\}$$

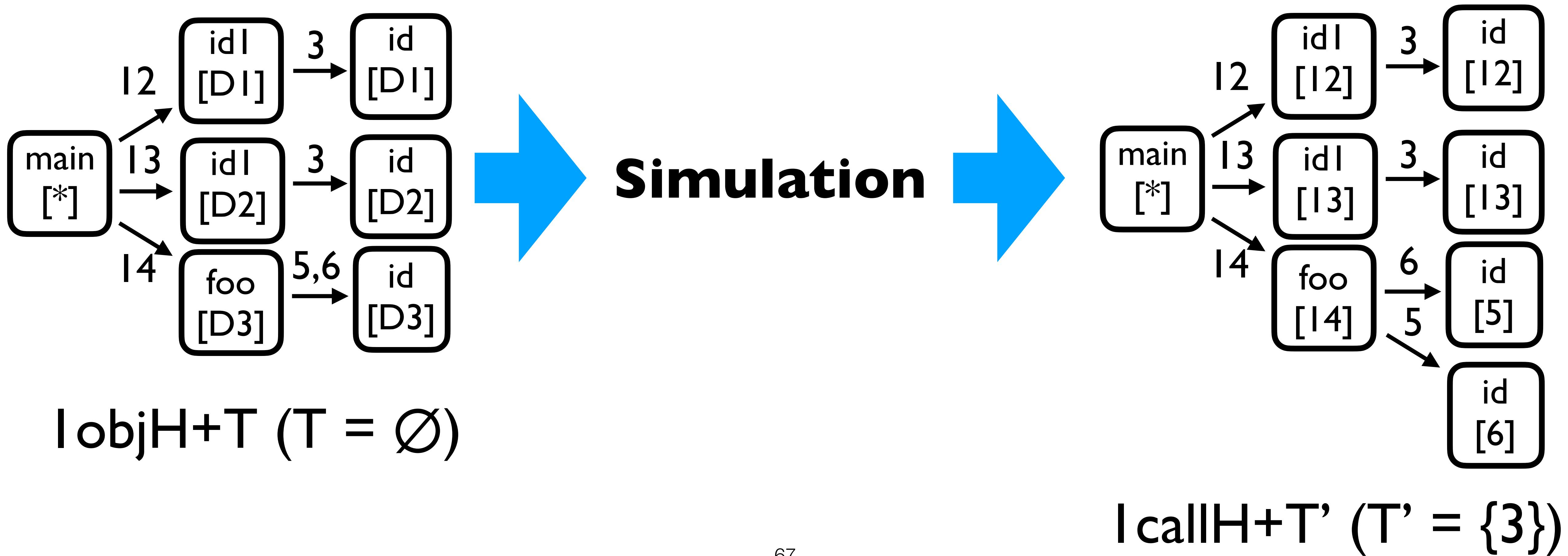
$$T' = (I_1 \cup I_2) \setminus I_3 = \{3\}$$

$\text{lobjH} + T \quad (T = \emptyset)$

$$I_3 = \{5, 6, I2, I3, I4\}$$

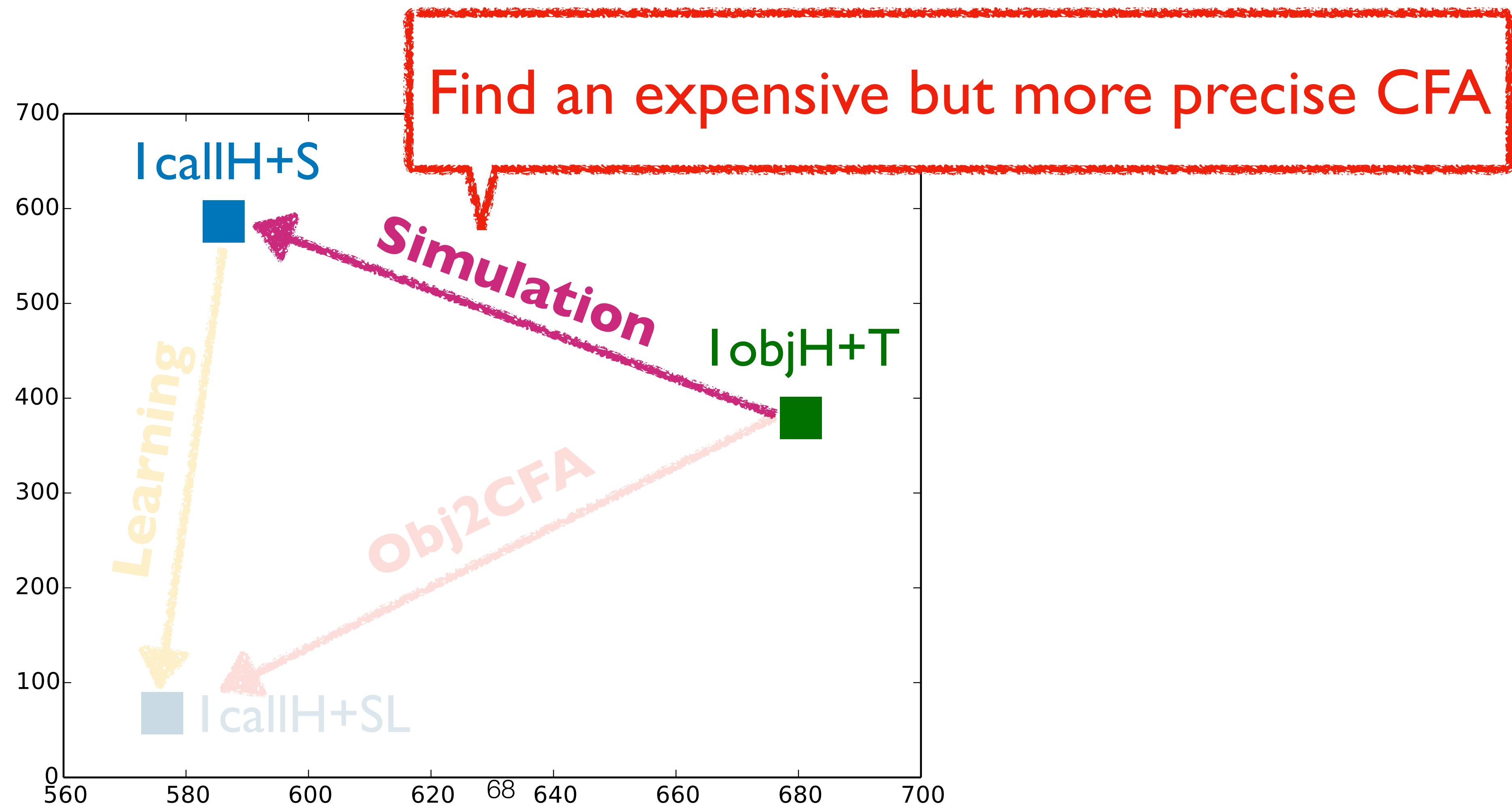
Technique I: Simulation

- With T' , CFA becomes more precise than the given object sensitivity



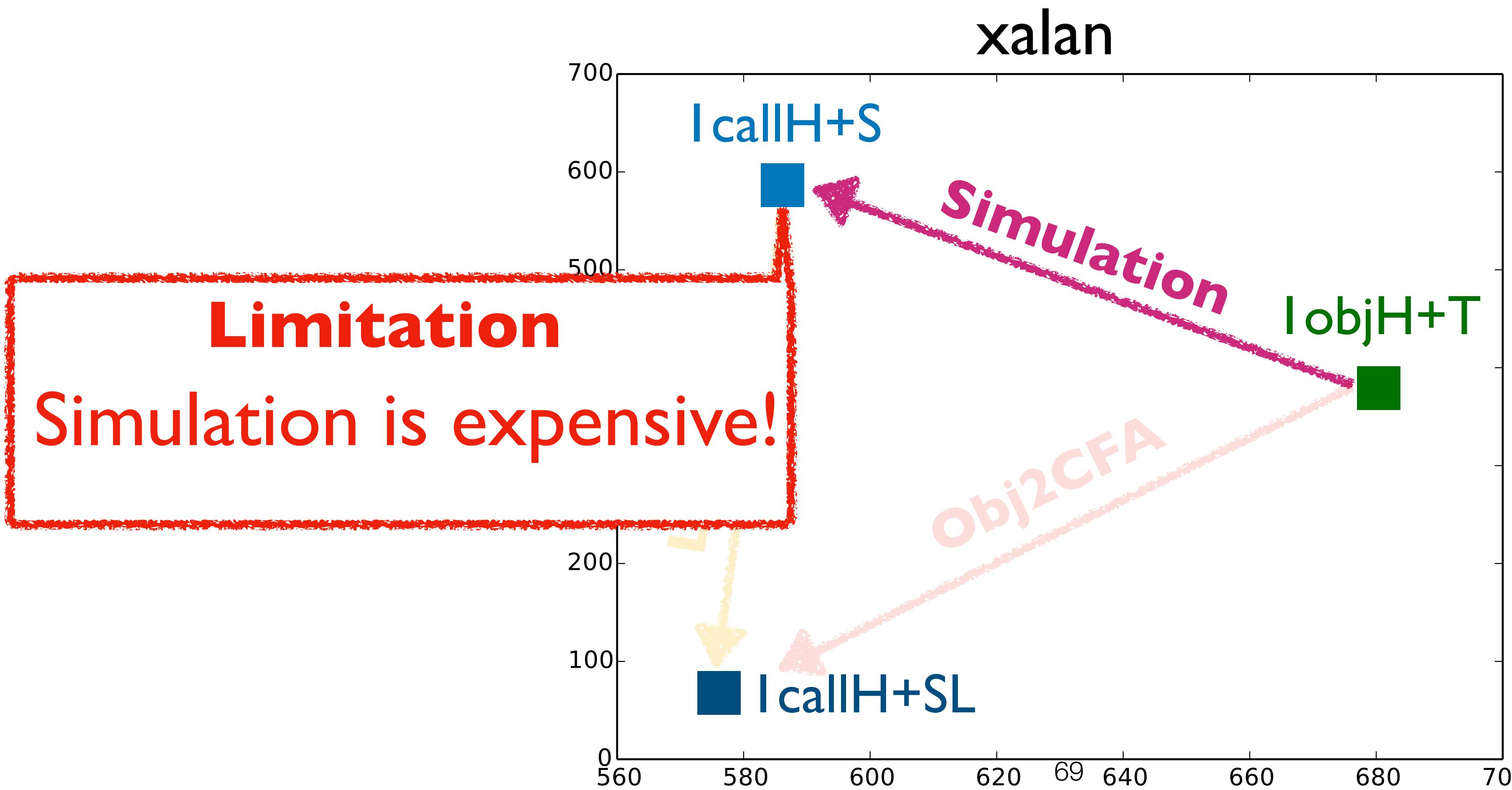
Our Technique : Obj2CFA

- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



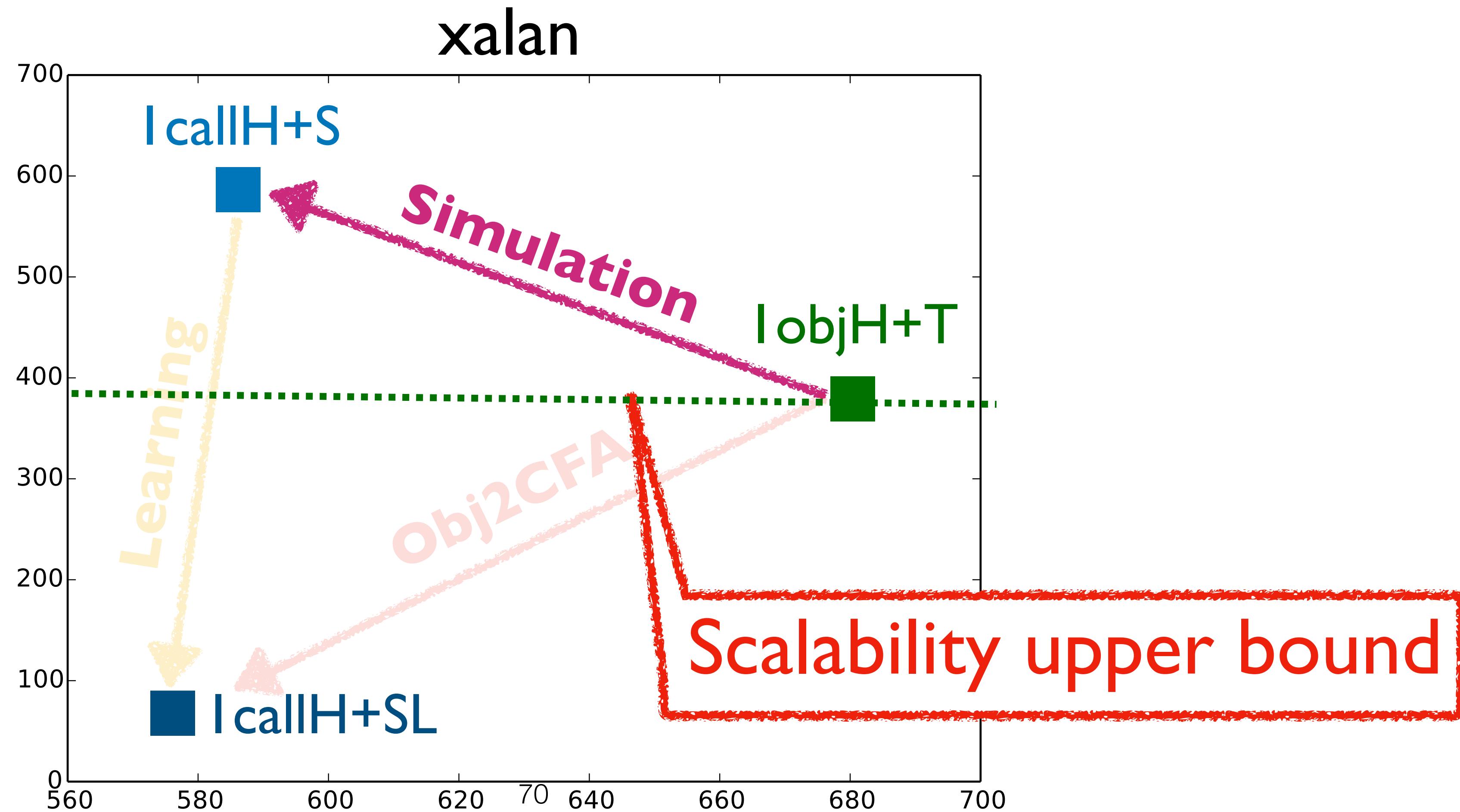
Our Technique : Obj2CFA

- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



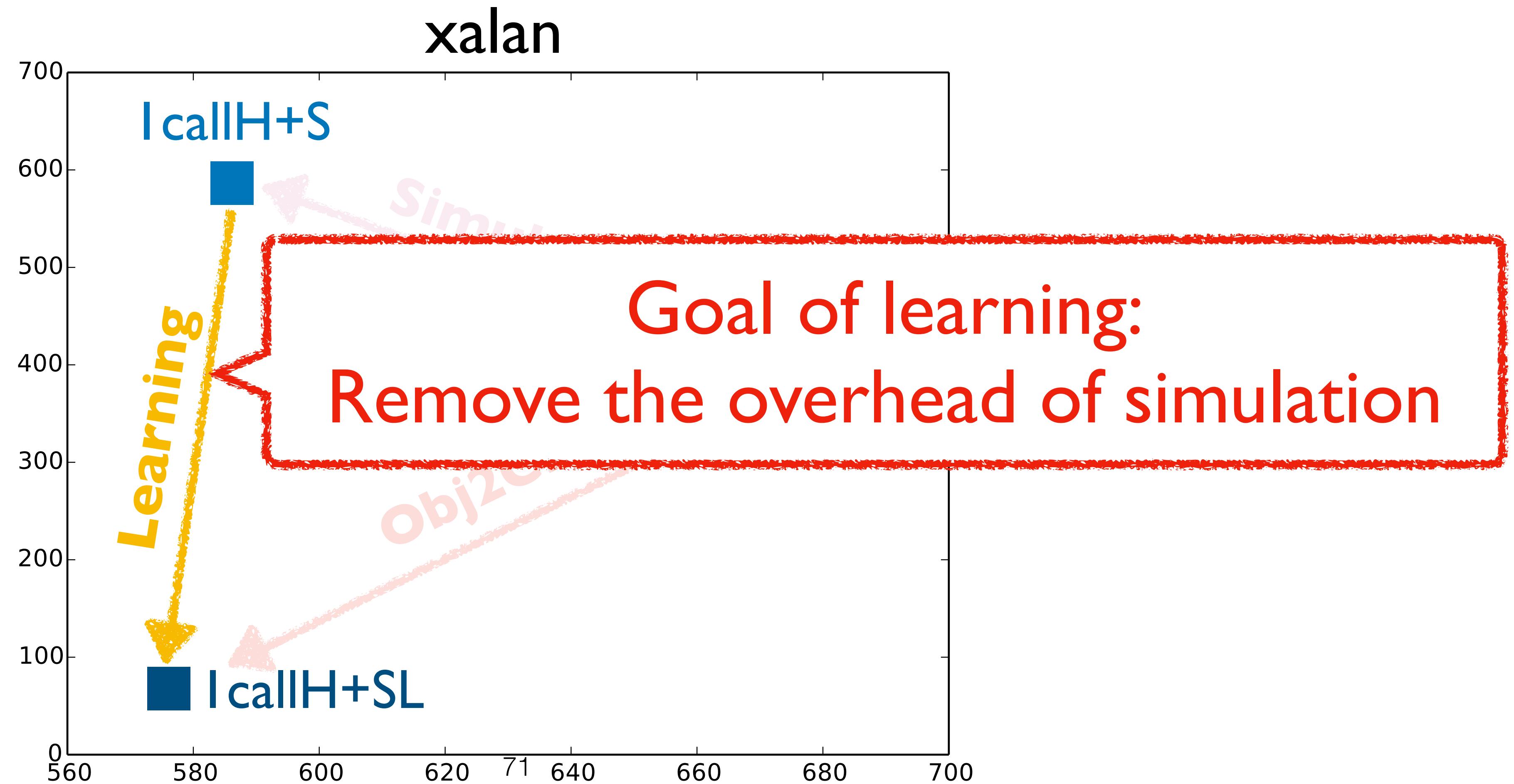
Our Technique : Obj2CFA

- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



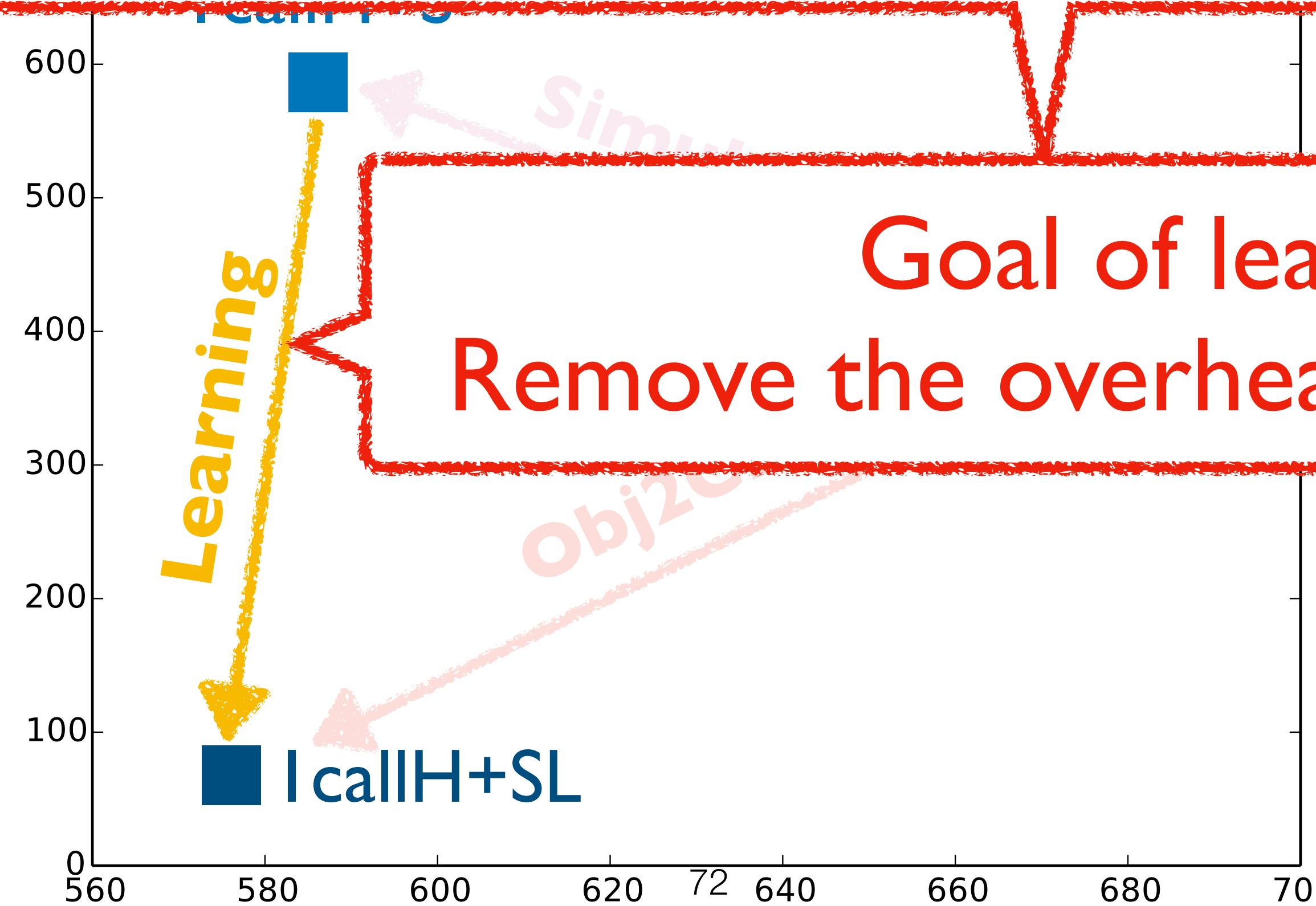
Our Technique : **Obj2CFA**

- **Obj2CFA** consists of **simulation** and simulation-guided **learning**



Our Technique: Obj2CEA

Given training programs and simulated tunneling abstractions, learning aims to find a model that produces similar tunneling abstractions without running the given object sensitivity



Our Technique · OHICFA

Given training programs and simulated tunneling abstractions,
learning aims to find a model that produces similar tunneling

The learned model will produce tunneling abstractions without
running object sensitivity

Details in paper

I call H+SL

Evaluation

Setting

- Doop
 - Pointer analysis framework for Java
 - Research Question: which one is better?

Call-site sensitivity vs Object sensitivity

Context tunneling is included

Setting

Doop

Negative results on CFA have been **repeatedly** reported on Doop

Strictly Declarative Specification of Sophisticated Points-to

Martin Bravenboer Yannis Smaragdakis
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003, USA
martin.bravenboer@cs.umass.edu yannis@cs.umass.edu

Abstract
We present the Dose framework for points-to analysis of Java programs. Dose builds on the idea of specifying pointer analysis algorithms declaratively, using Datalog, a logic-based language for defining (recursion-free) relations. We carry the declarative approach further than past work by describing the full end-to-end analysis in Datalog and optimizing aggressively using a novel technique specifically targeting highly recursive Datalog programs.

As a result, Dose achieves several benefits, including full order of magnitude improvements in runtime. We compare Dose with Libra and Hendren's Parrot, which define the state of the art for context-sensitive analysis. For the exact same logical pointer-to definitions (and, consequently, identical precision) Dose is more than 15x faster than Parrot for a typical context-sensitive analysis of the DaCapo benchmarks, with lower but still substantial speedups for other important analyses. Additionally, Dose scales to very precise analyses that are impossible with Parrot and Whaley et al.'s libra, directly addressing open problems in prior literature. Finally, our implementation is modular and can be easily configured to analyze with a wide range of characteristics, largely due to its declarativeness.

Categories and Subject Descriptors: F.3.3 [Logics and Methodologies of Programming]; Semantics of Programming Languages—Program Analysis; D.1.5 [Programming Techniques]; Logic Programming

General Terms: Algorithms, Languages, Performance

1. Introduction
Points-to (or pointer) analysis intends to answer the question “what objects can a program variable point to?” This question forms the basis for practically all higher-level program

analyses. It is, thus, not surprising that a lot has been devoted to efficient and precise techniques. Context-sensitive analyses are a class of precise points-to analyses. Context approaches qualify the analysis function with context, which captures a static notion of the situation. Typical contexts include thread identities (for a call-site sensitive analysis, meaning of “context-sensitive”) or recent object-sensitive analysis.

In this work we present Dose, a general-purpose points-to analysis framework that makes precise context-sensitive analyses feasible. Dose implements a range of algorithms, insensitive, call-site sensitive, and object-site specific, modularly as verifications on a database. Compared to the prior state of the art, Dose provides orders-of-magnitude faster analysis.

The main elements of our approach are a logic language for specifying the program, aggressive optimization of the Datalog, and Datalog for program analysis (both low-level [6, 9] and the new Ouranos project), however, accounts for an order-of-magnitude performance improvement: unoptimized run over 1000 times more slowly. Graceful fit: well, the approach of handling a database, by specifically targeting the incremental evaluation of Datalog implications, our approach is entirely Datalog declaratively. The logic required both for creation as well as for handling the full set of the Java language (e.g., static initializers, reference objects, threads, exceptions, etc.) makes our pointer analysis specification both also efficient and easy to tune. Given strong data points in support of declarative logic that probabilistically matches user effort, planning and optimizing complex mutations at an operational level of abstraction.

Pick Your Contexts Well: Understanding Object-Sensitive Analysis

The Making of a Precise and Scalable Pointer Analysis

Yannis Smaragdakis
Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003, USA
and Department of Informatics,
University of Athens, 15784, Greece
yannis@cs.umass.edu—smaragd@di.uoa.gr

Martin Breuvenhoer
LogicBlox Inc.
Two Midtown Plaza
Atlanta, GA 30309, USA
martin.breuvenhoer@logicblox.org

Abstract

Object-sensitivity has emerged as an excellent context abstraction for pointer analysis in object-oriented languages. Despite its analytical success, however, object-sensitivity is poorly understood. For instance, for a context depth of 2 or higher, past scalable implementations deviate significantly from the original definition of an object-sensitive analysis. The reason is that the analysis has more degrees of freedom, relating to which context elements are picked at every method call and object creation. We offer a clean model for the analysis design space, and discuss a formal and informed understanding of object-sensitivity and of how to create good object-sensitive analyses. The results are surprising, in their extent. We find that past implementations have made a suboptimal choice of contexts, to the severe detriment of precision and performance. We define a “full-object-sensitive” analysis that results in significantly higher precision, and often performance, for the exact same context depth. We also introduce “hypersensitivity,” an explicit approximation of object-sensitivity that preserves high context quality at substantially reduced cost. A type-sensitive pointer analysis makes an unconventional use of types as context; the context types are not dynamic types of objects involved in the analysis, but instead upper bounds on the dynamic types of their allocator-expects. Our results expose the influence of context choice on the quality of points-to analysis and demonstrate type sensitivity to be as ideas with major impact: it exclusively advances the static-at-the-art of pointers analysis but simultaneously does so several times faster than an analogous object-sensitive analysis, sensitivity to analysis or memory access context sensitivities, and precision (comparable to the best object-sensitive analysis with the same context depth).

Keywords and Subject Descriptors: F.3.2 [Logics and Semantics of Programming]; Semantics of Programming Languages—Program Analysis

1. Introduction

Point-to analysis for pointer-sensitive state provides a means of computing a static expression (or just a var) to define program variables only every so far program executions such as calls, polarizers determine the target of pointer-oriented dynamic binds applications. By analysis judiciously judicious bind any attempt to track Furthermore, the global analysis make it hard to interact with various low-level functional languages, can achieve tractability and consists of qualifying local object abstractions with information (e.g., “what is to cover all possible cases while separating all relevant kinds of context sensitivity [11, 12] and object-sensitivity [13].

Ever since the introduction [13], there has been a desire to be in a superior context of visiting high precision. Object-sensitivity has been much analyzed have chosen non-sensitivity. AIA analysis is concerned with understanding it conveniently, is even more scalable and precise.

What is object-sensitivity? An easy way to conceive the context known call-site analysis uses method call sites as the methods as contexts do separate information on per-call-stack (i.e., request but led to the current method information on heap does not lead to the object’s life below, a local variable sensitive

Categories and Subject Descriptors: F.3.2 [Logics and Semantics of Programming]; Semantics of Programming Languages—Program Analysis

1.1.1.1 [Programming Languages]: Formal Definitions and Theory—Semantics

General Terms: Algorithms, Languages, Performance

Permissions: This work is licensed under a Creative Commons Attribution Non-Commercial-ShareAlike 4.0 International License.

PLATEAU Coverage: Coverage

Int

Hybrid Context-Sensitivity for Points-To Analysis

George Kastisnis Tania Saragiotaki

Department of Informatics
University of Athens
(gkastinis, tseragioti)@di.uoa.gr

Abstract

Context-sensitive points-to analysis is valuable for achieving high precision with good performance. The standard flavors of context-sensitivity are call-site-sensitivity (CSA) and object-sensitivity, among both flavors of context-sensitivity, instruction precision is usually high cost. We show that a selective combination of call-site- and object-sensitivity for Java points-to analysis is highly profitable. Namely, by keeping a combined context sensitive analysis of selected language features, we can closely estimate the precision of an analysis that keeps both contexts. In terms of speed, the selective combination of both context-sensitivity only rarely deoptimizes methods over analysis also faster than a more object-sensitive analysis. This leads to a larger study of analyses (e.g., 1-object-sensitive, 2-object-sensitive with a context-sensitive heap, type-sensitive) involving a new set of performance/measures, cost+speed.

Keywords Software Engineering, P.5.2 [Logic and Meaning]; D.2.4 [Semantics of Programming Languages: Programs]; D.3.4 [Programming Languages]: Processors—

Term Algorithms, Languages, Performance

1 points-to analysis; context sensitivity; object-sensitivity; type-sensitivity

Introduction

points-to analysis is a static program analysis that consists of one effect (typically identified by allocation site) that a variable may point to. The area of points-to analysis (and relative alias analysis) has been the focus of intense research and is among the most standardized and well-understood of compiler analyses. The emphasis of points-to analysis algorithms combining fairly precise modeling of pointer behavior with efficiency. The challenge is to pick judicious approximations that are satisfactory precision at a reasonable cost. Furthermore, increasing precision often leads to higher asymptotic complexity, this worst-case behavior is rarely manifested in practice. Instead, techniques that are effective at maintaining precision also yield better average-case performance, smaller points-to sets lead to less work.

In contrast, object-sensitivity adds cost to instructions containing a new site (here, a better name for "object," "allocationsensitive")¹. That is, an object, the analysis separates the allocation site of the receiver of the method it is called, as well as its context. Thus, in the above example, `main` will analyze `z` separately depending on objects `a` and `c` that `z` points to. It is ignorant whether `z`, `a` and `c` refer to how many objects the affect is remote and unrelated to the code. It is not possible to compare the precision and a call-site-sensitive analysis in general even that whether the object `z` all calls to `f` at one time, as two

1) In order to make figures and save space, in all parts of this work we present our results in grayscale without background that copies are not made or distributed without permission and that copies have this statement and the full citation page. To copy otherwise, without permission or to post on servers or to redistribute to lists, prior permission or license must be obtained.

June 16–20, 2013, Seattle, WA, USA
© 2013 ACM 978-1-4503-2014-6/13/06... \$15.00

Yannis Bakargiannis **Georgia Kastriou** **George Balatsouris**
 Department of Informatics
 University of Athens
 {ybakar, gkastriou, gbalat}@hua.gr}

Abstract Context-sensitivity is the primary approach for adding more precision to analysis, while hopefully also maintaining soundness. As expert programmers with context-sensitive analysis, we know that they are bi-modal: either the analysis is precise or it manipulates only manageable amounts of data, and thus moderately well; or the analysis grows exponentially as the size of the program increases, and becomes irrelevant/unusable since this would be expected given the program's size. Thus, this paper approaches that makes precise context-sensitive analysis (flavor: call-site, object-, or type-sensitive) scale correctly at a level comparable to that of a context-insensitive analysis. To address this issue, we propose *context-sensitive analysis*: a fast uniformly scaling context-sensitive analysis by eliminating performance-diminishing behavior: as a small problem, context-sensitive analysis consists of a common abstraction that performs a context-insensitive analysis, then uses the abstraction refine (i.e., analyze context-sensitivity); propagate that will not cause explosion in the running time. The technical challenge is to appropriately identify such elements. We show that a simple but principled approach is available, effective, achieving scalability (from $O(1)$ to $O(n^2)$) for benchmarks previously completely out-of-reach context-sensitive analyses.

Keywords and Subject Descriptors: F.3.3 [Logics and Meanings of Programs]: Semantics of Programming Languages—Program D.3.4 [Programming Languages]: Functional Languages—Algorithms, Languages, Performance

ACM Classification: F.3.3 [Logics and Meanings of Programs]: Semantics of Programming Languages—Program D.3.4 [Programming Languages]: Functional Languages—Algorithms, Languages, Performance

Authors' addresses: Yannis Bakargiannis, Georgia Kastriou, George Balatsouris, Department of Informatics, University of Athens, 757 100 Athens, Greece.

points-to analysis; context-sensitivity; object-type-sensitivity

Introduction

Context-sensitivity is probably the most common whole-program analysis, and often serves as a substitute for a variety of high-level analysis tasks. Points-to analysis computes the set of objects (or contracted to their allocation sites) that a program variable can point to during runtime. The promise, as well as the challenge, is to make digital (or soft) copies of all or part of this work for the present or for granted without the practitioner that copies are not mere or automated commercial revenue, and that copies help him make and fulfill his vision. Copyright for this document is held by the author(s). It is made available under the terms of the Creative Commons License, which permits unrestricted use, distribution, and reproduction in any medium, provided that attribution is made to the author(s), and that the original author(s) receive credit as the copyright holders. Requests for permission to redistribute or republish must be submitted by permission to the author(s).

Copyright © 2011, IBDL, 2011, Bellaterra, Spain. ISSN 1522-3702. The material in this document is subject to copyright. The use of all or part of this material is granted without prior permission or agreement from the copyright holder, except where explicitly stated. Periodicals postage is paid at Rahway, NJ, and at additional mailing offices. Postage paid at Rahway, NJ, and at additional mailing offices.

DOI: 10.1145/2596258.2596268

Making k -Object-Sensitive Pointer Analysis More Precise with Still k -Limiting

Tian Tan¹, Yue Li¹, and Jingling Xue^{1,2}

¹ School of Computer Science and Engineering, UNSW Australia

² Advanced Innovation Center for Imaging Technology, CNU, China

Abstract. Object-sensitivity is regarded as arguably the best context abstraction for pointer analysis in object-oriented languages. However, a k -object-sensitive pointer analysis, which uses a sequence of k allocation sites (as k context elements) to represent a calling context of a method call, may end up using some context elements redundant without increasing a fine partition of the space of (concrete) calling contexts for the method call. In the paper, we introduce REAN, a general approach for improving the precision of any k -object-sensitive analysis, denoted k -obj, by still using a k -limiting context abstraction. The novelty is to identify allocation sites that are redundant context elements in k -obj from an Object Allocation Graph (DAG), which is built based on a prescan analysis (e.g., a context insensitive Andersen's analysis) performed initially on a program and then avoid them in the subsequent k -object-sensitive analysis for the program. REAN is generally more precise than k -obj, with a precision that is guaranteed to be as good as k -obj in the worst case. We have implemented REAN as an open-source tool and applied it to refine two state-of-the-art whole program pointer analyses in DCCP. For two representative classes (mapreduce and mapjoin-ex) evaluated on a set of nine large Java programs from the DaCapo benchmark suite, REAN has succeeded in making both analyses more precise for all these benchmarks while each class at only small increases in analysis cost.

1 Introduction

Pointer analysis, as an enabling technology, plays a key role in a wide range of client applications, including bug detection [3, 23, 35, 31], security analysis [20], compiler optimisation [6, 33], and program understanding [12]. Two main dimensions of pointer analysis precision are flow-sensitivity and context-sensitivity. For C/C++ programs, flow-sensitivity is needed by many clients [11, 16, 3]. For object-oriented programs, e.g., Java programs, however, context-sensitivity is known to deliver tractable and useful precision [17, 19–21, 28–30] in general.

There are two general approaches to achieving context-sensitivity for object-oriented programs, call site sensitivity (k -CFA) [27] and object sensitivity [24, 29] (among others). A k -CFA analysis represents a calling context of a method call by using a sequence of k call sites (i.e., k labels with each denoting a call site). In contrast, a k -object-sensitive analysis uses k object allocation sites (k labels with each denoting a new statement) as context elements.

ta-Driven Context-Sensitivity for Points-to Analysis

UN JEONG¹, Korea University, Republic of Korea
SEOK JEON², Korea University, Republic of Korea
GDEOK CHA³, Korea University, Republic of Korea
JOO OH⁴, Korea University, Republic of Korea

We present a new data-driven approach to achieve highly cost-effective context-sensitive points-to analysis. While context-sensitivity has greater impact on the analysis precision and performance than any precision-improving technique, it's difficult to accurately identify the methods that would benefit the most from context-sensitivity and decide how much context-sensitivity should be used for them. Manually tuning such rules is a nontrivial and laborious task that often delivers suboptimal results in practice. To overcome these challenges, we propose an automated and data-driven approach that learns to effectively apply context sensitivity from codebases. In our approach, points-to analysis is equipped with a parameterized and rule-based system that specifies properties on program elements, that decide when and how much to apply context-sensitivity. We present a greedy algorithm that efficiently learns the parameter of the heuristic rules. We implemented our approach in the Deep framework and evaluated using three types of context-sensitive rules: conventional object-sensitivity, selective hybrid object-sensitivity, and type-sensitivity. In all cases, experimental results show that our approach significantly outperforms existing techniques.

Concepts: - Theory of computation → Program analysis; - Computing methodologies → Machine learning approaches;

Keywords: Data-driven program analysis, Points-to analysis, Context-sensitivity

Reference Format:
Un Jeong, Minseok Jeon, Sangdeok Cha, and Haljin Oh. 2017. Data-Driven Context-Sensitivity for Points-to Analysis. *Proc. ACM Program. Lang.*, 1, OOPSLA, Article 104 (October 2017), 27 pages.
<https://doi.org/10.1145/3033924>

INTRODUCTION

Points-to analysis is one of the most important static program analyses. It approximates various memory locations that a pointer variable may point to at runtime. While useful as a stand-alone tool for many program verification tasks (e.g., detecting null-pointer dereferences), it is a key ingredient for subsequent higher-level program analyses such as static bug finders, security auditing tools, and program understanding tools.

In object-oriented languages, context-sensitive points-to analysis is important as it must distinguish a method's local variables and objects in different calling-contests. For languages like Java, the first and second authors contributed equally to this work.

Corresponding author

¹✉ email address: 3. jeng@kku.ac.kr; 2. mjeon@kku.ac.kr; 3. Cha, wha@kku.ac.kr; haljin.oh@kku.ac.kr

To make digital or hard copies of all or part of this work for personal or classroom use is granted without fee and without permission or distribution for profit or commercial advantage and the right to do so is reserved. Copying with fee is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires specific permission and/or a fee. Request permission from permissions@acm.org.

**2009
POPSL**

2011 (POPL)

2013
(PLDI)

201
(PLD)

20
(SA)

2017 (OOPSLA)

Setting

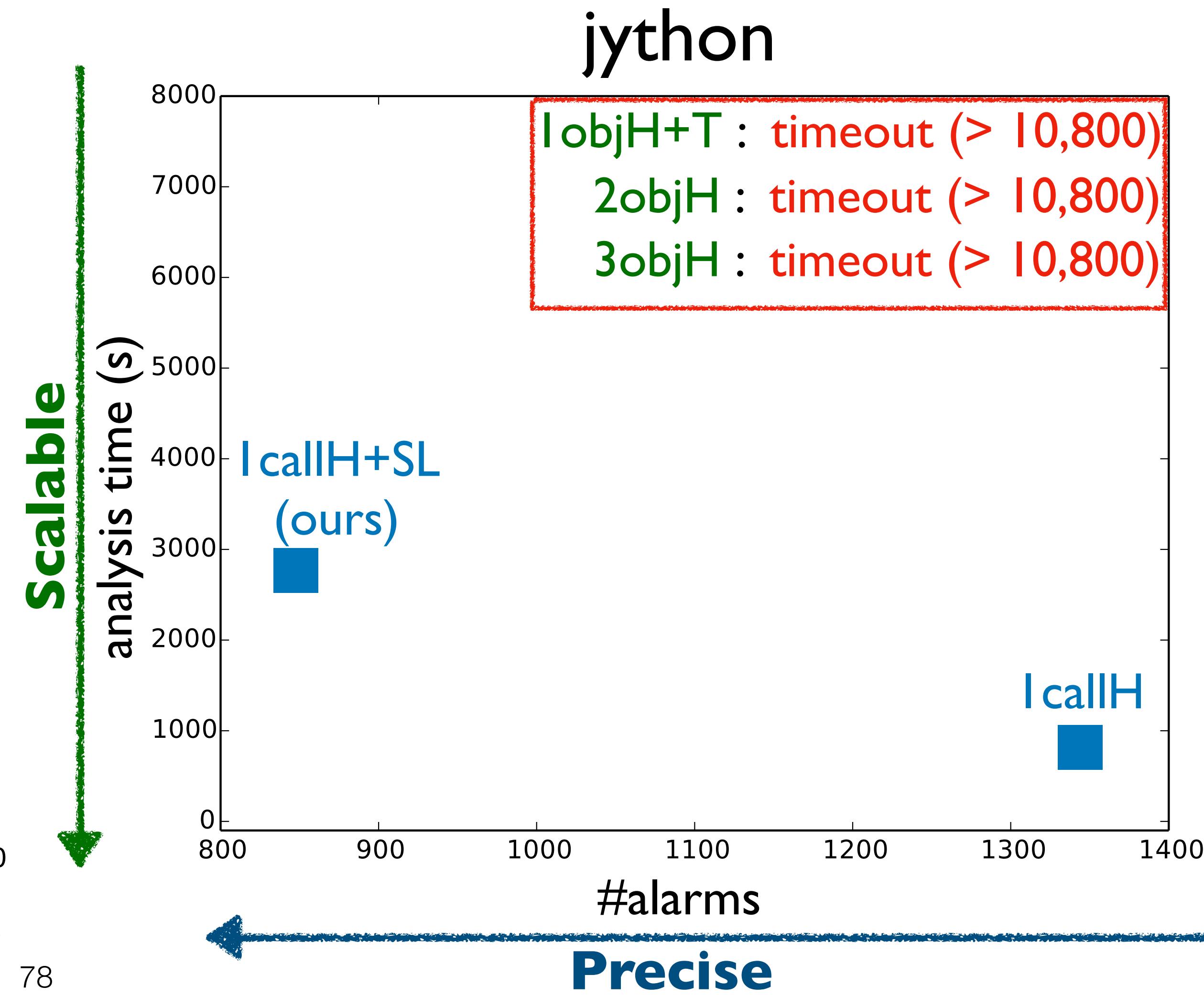
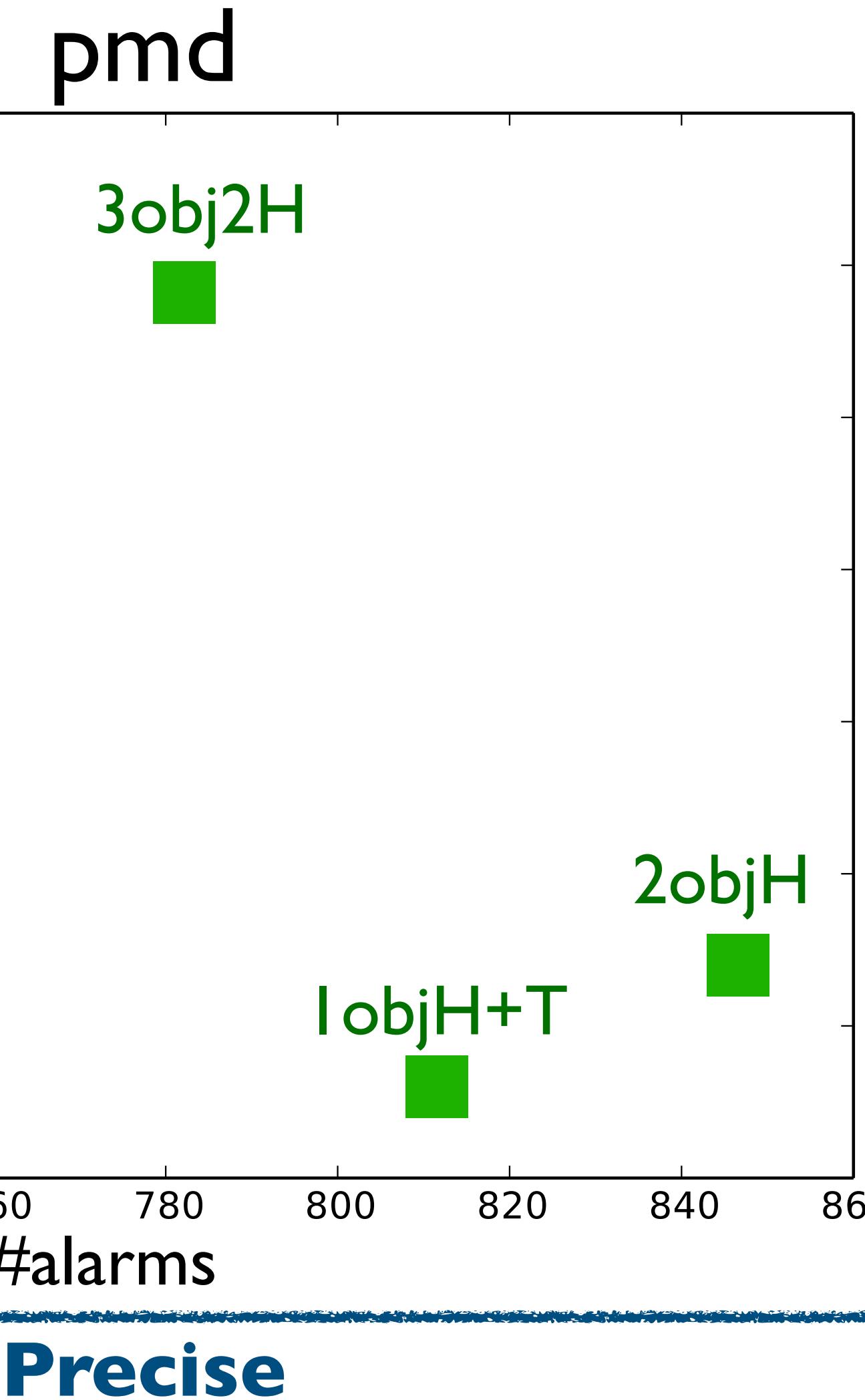
- Doop
 - Pointer analysis framework for Java
 - Research Question: which one is better?

Call-site sensitivity vs Object sensitivity

Context tunneling is included

Call-site Sensitivity vs Object Sensitivity

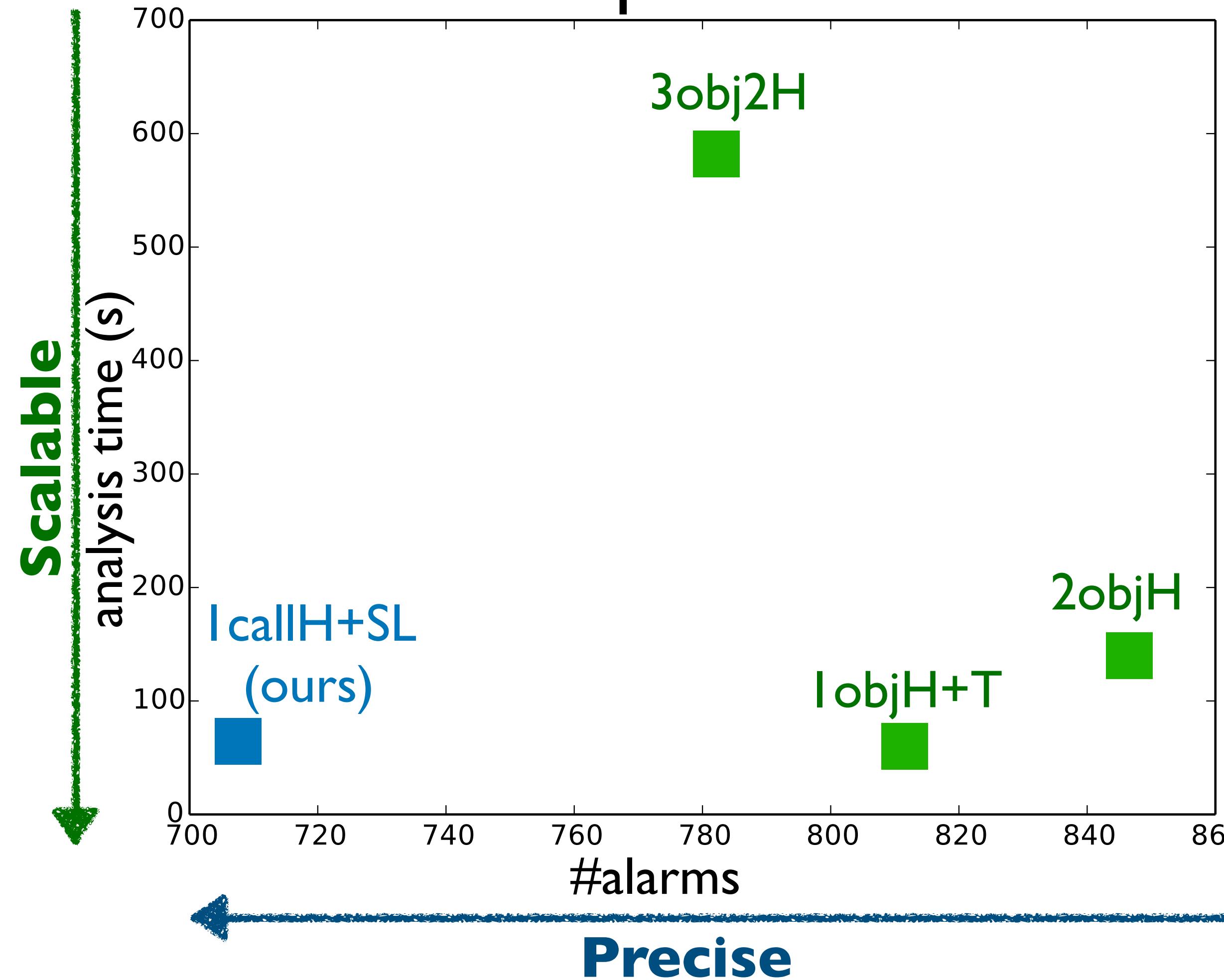
- $I_{callH+SL}$ (ours) is more precise and scalable than the existing object sensitivities



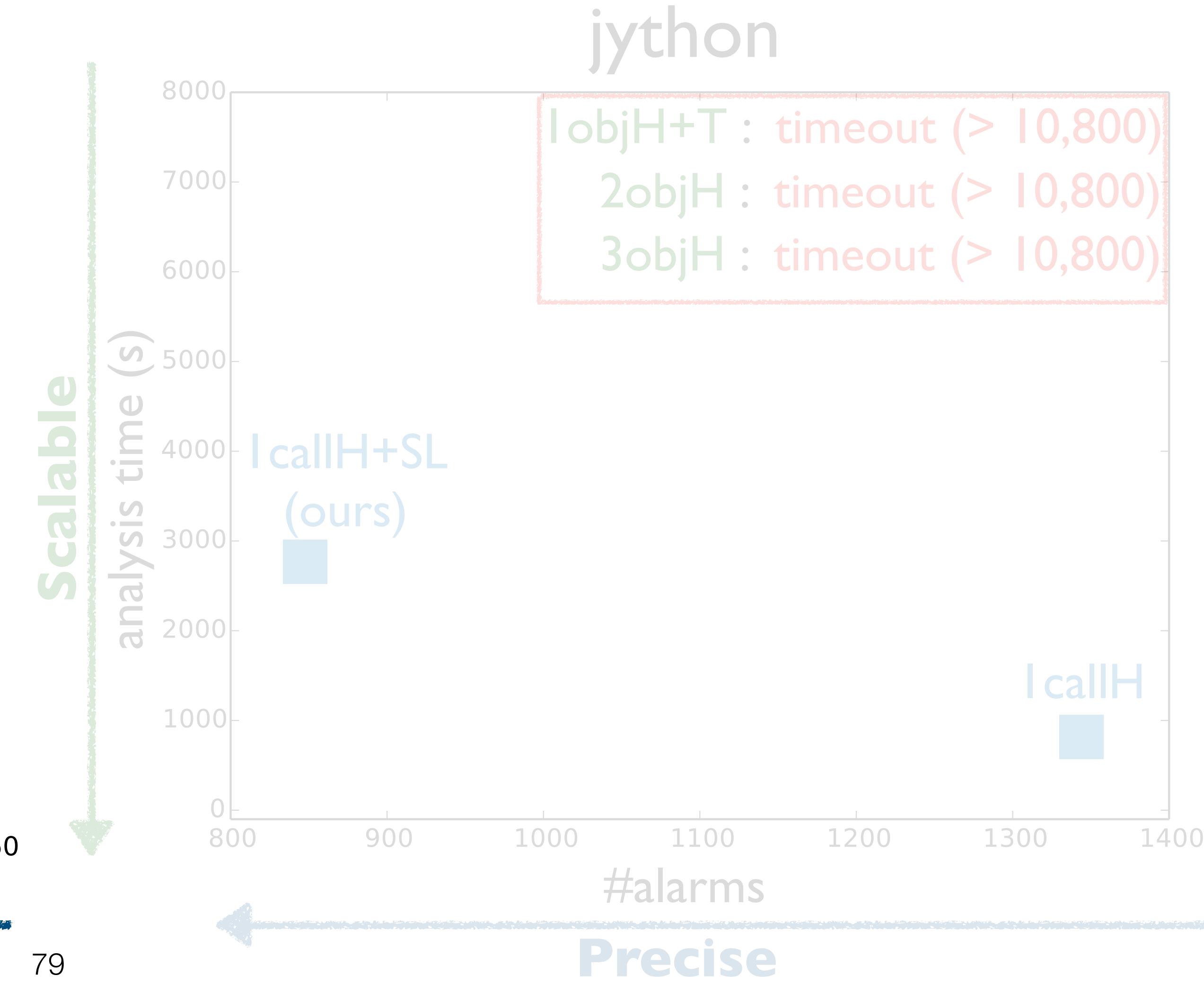
Call-site Sensitivity vs Object Sensitivity

- $I_{callH+SL}$ (ours) is **more precise** and **scalable** than the existing object sensitivities

pmd

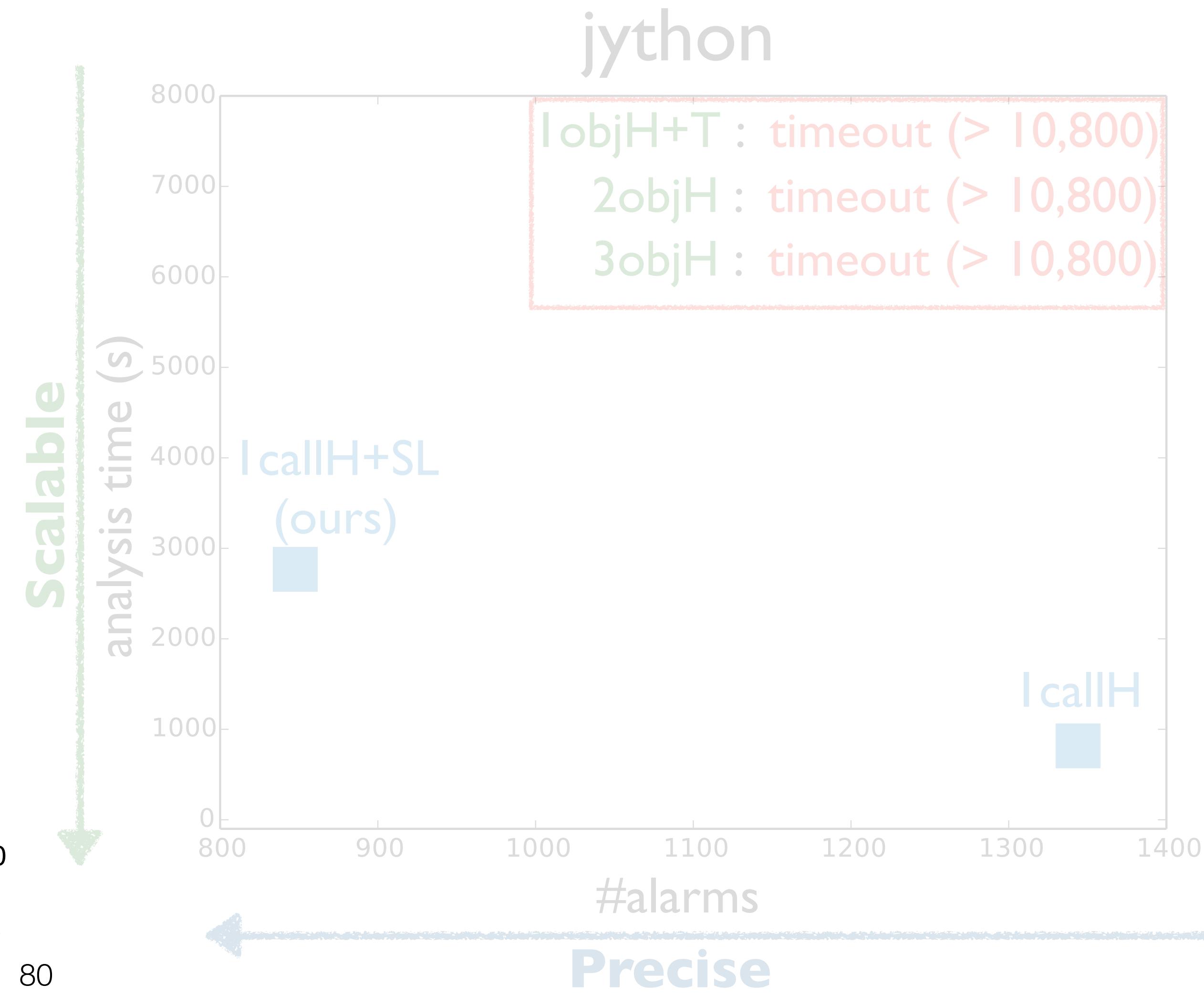
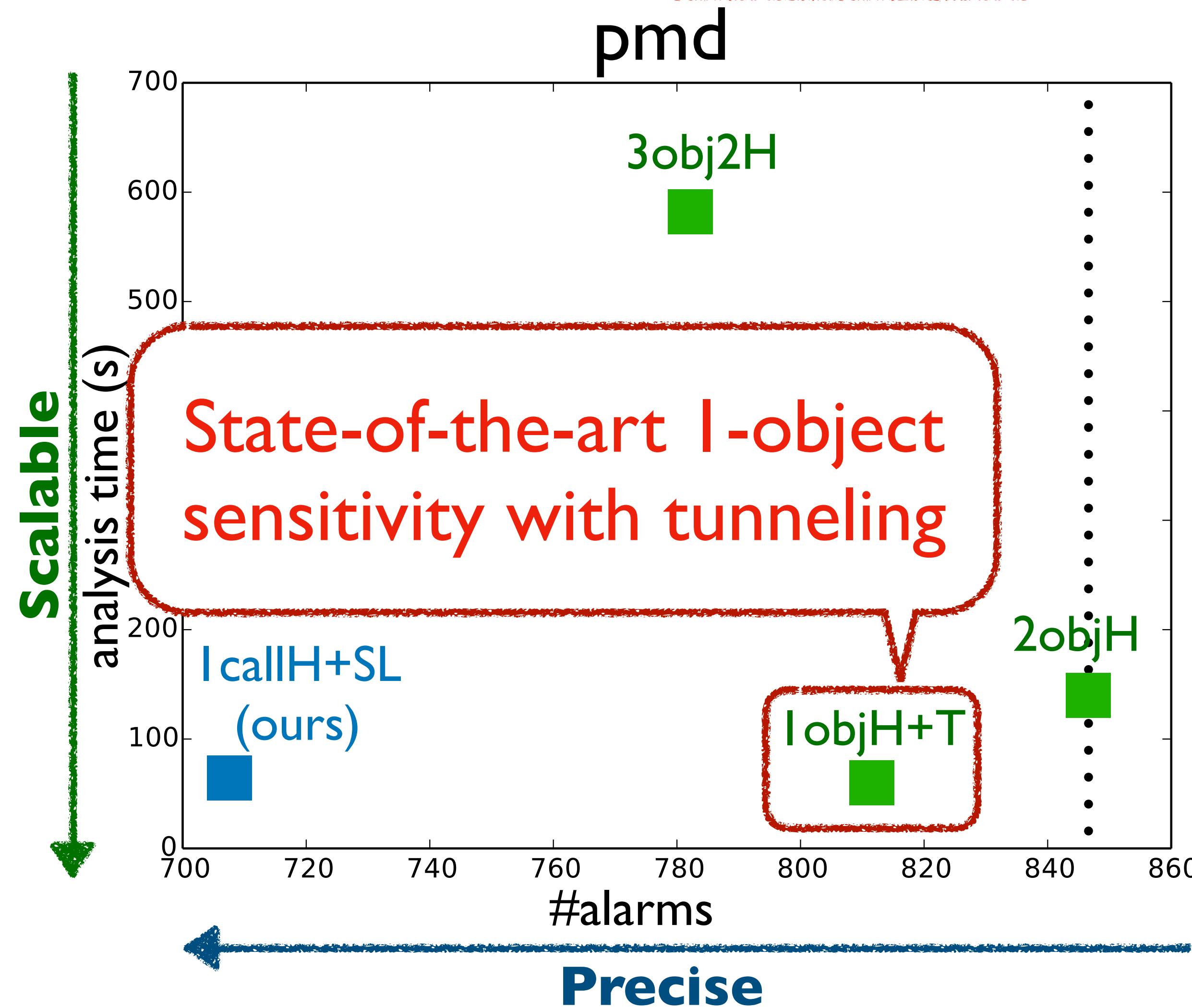


jython



Call-site Sensitivity vs Object Sensitivity

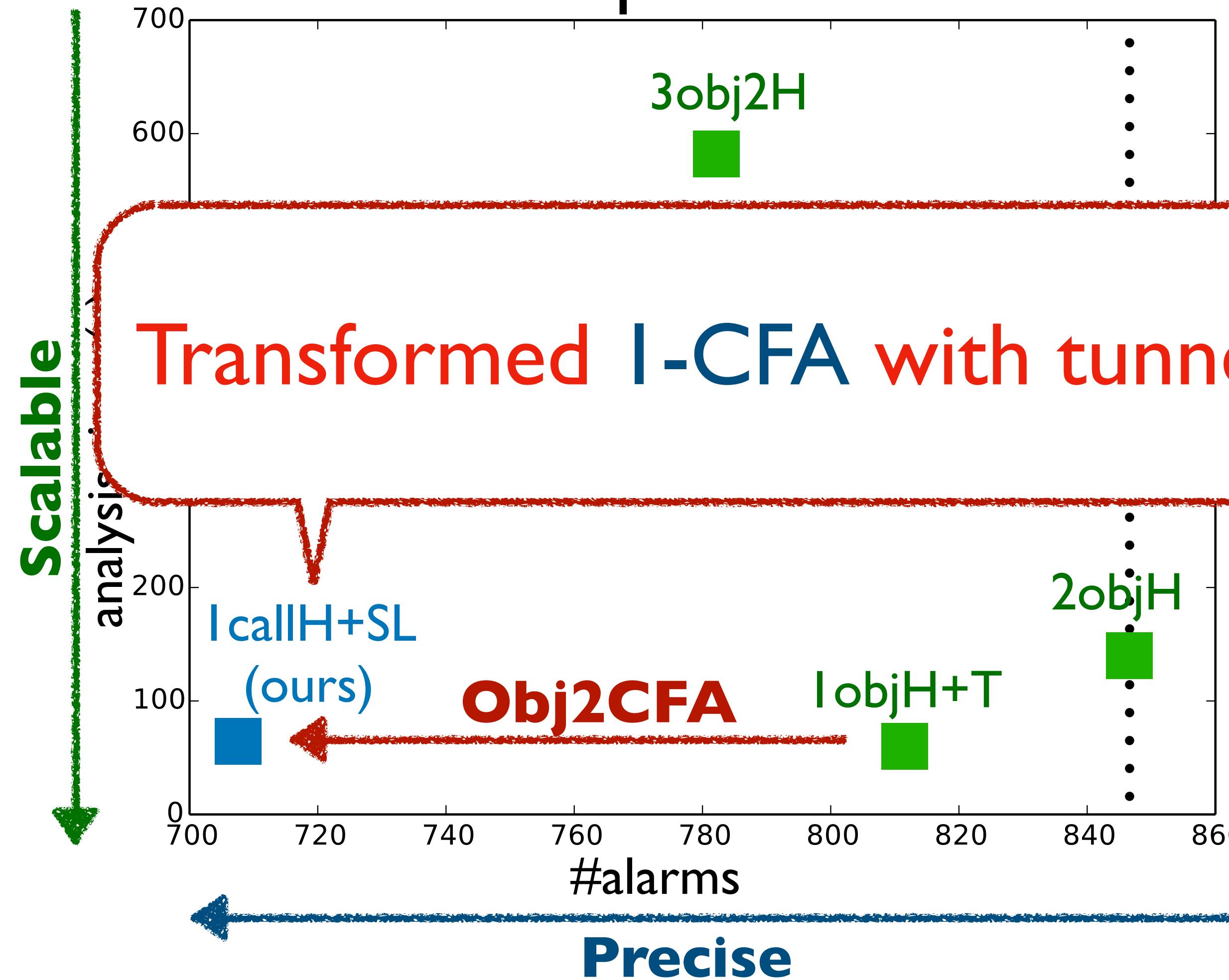
- $I_{callH+SL}$ (ours) is **more precise** and **scalable** than the existing object sensitivities



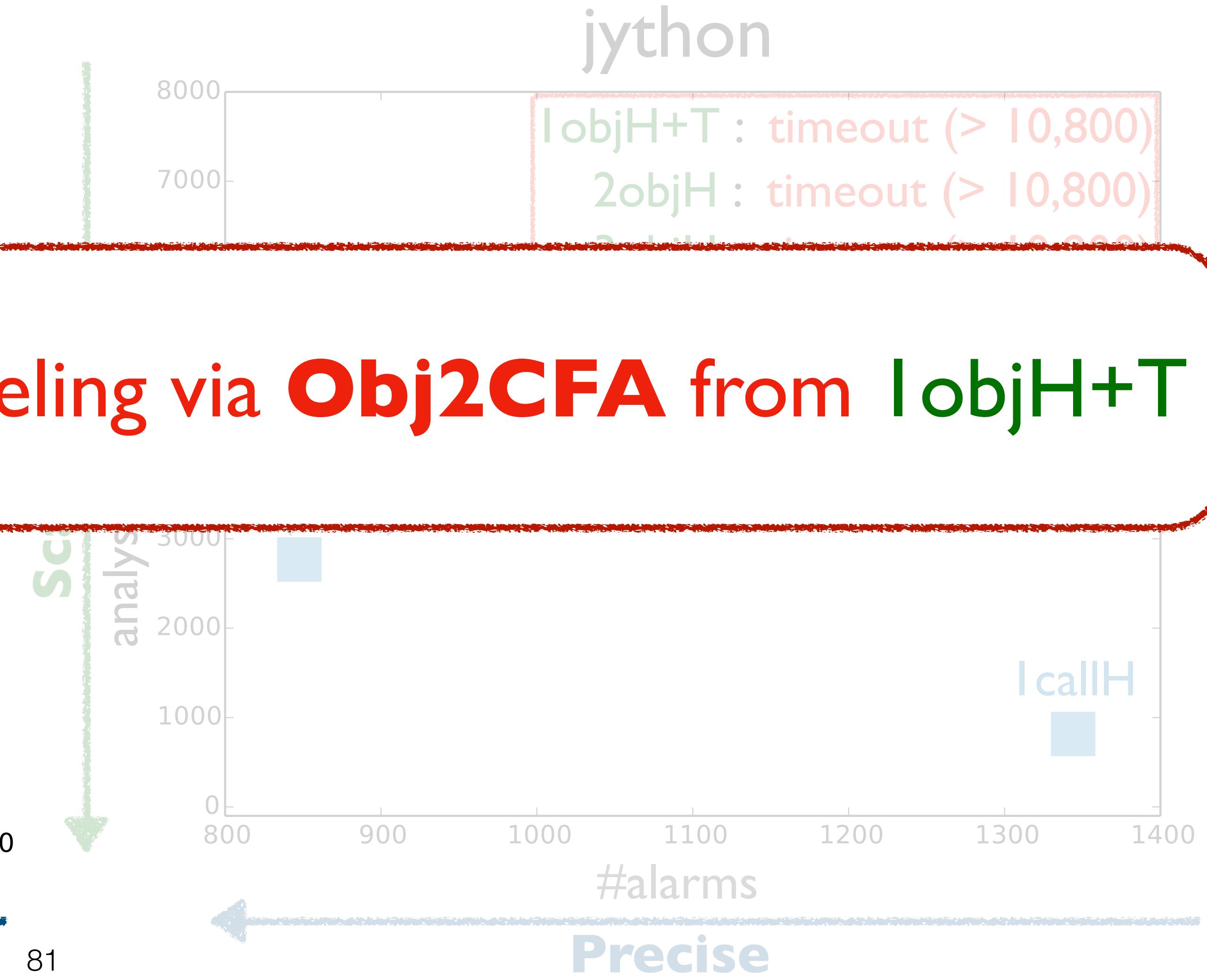
Call-site Sensitivity vs Object Sensitivity

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

pmd

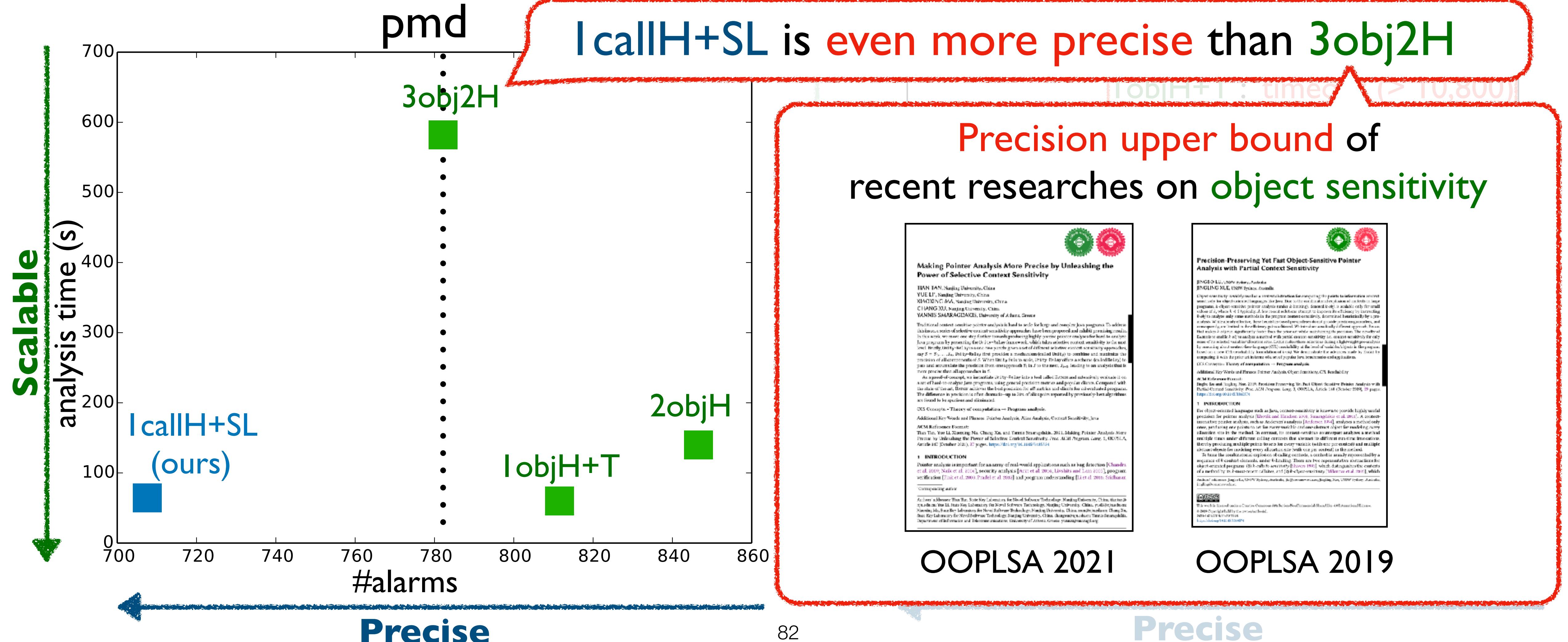


jython



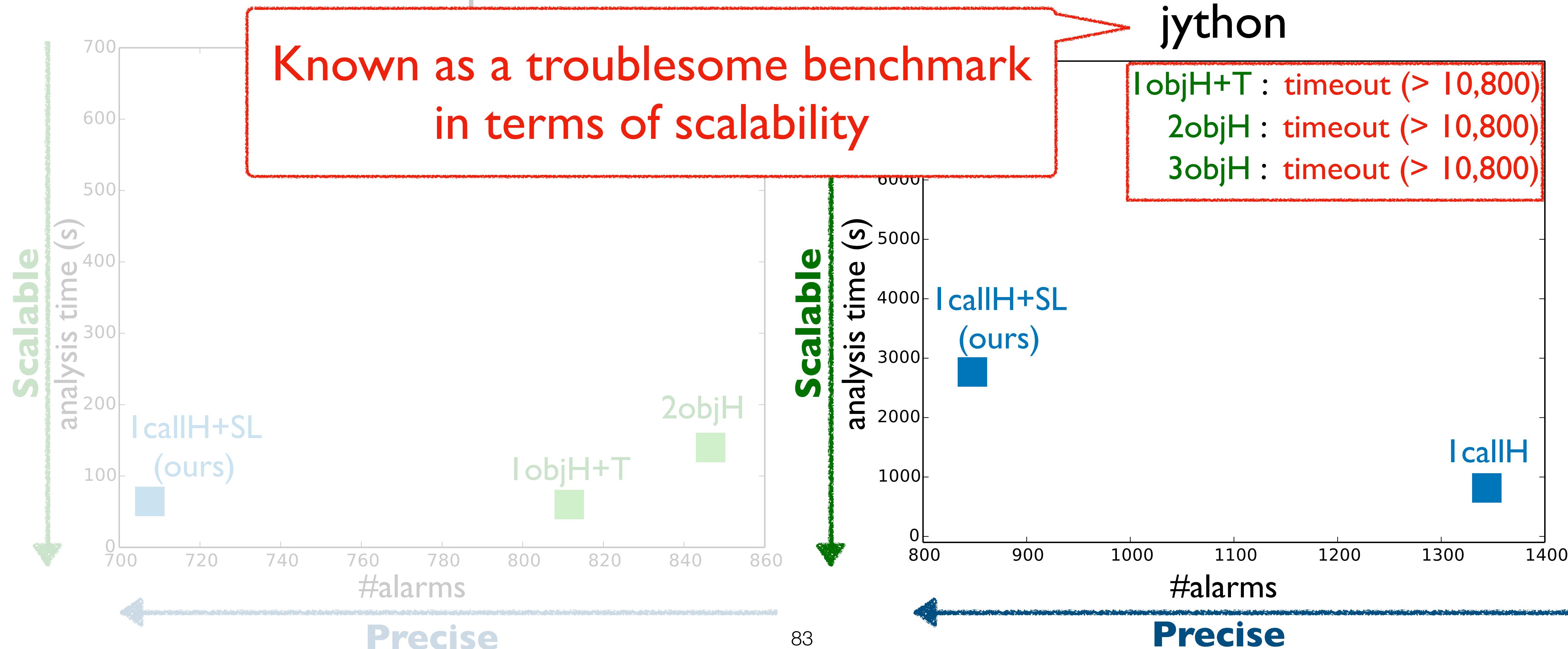
Call-site Sensitivity vs Object Sensitivity

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



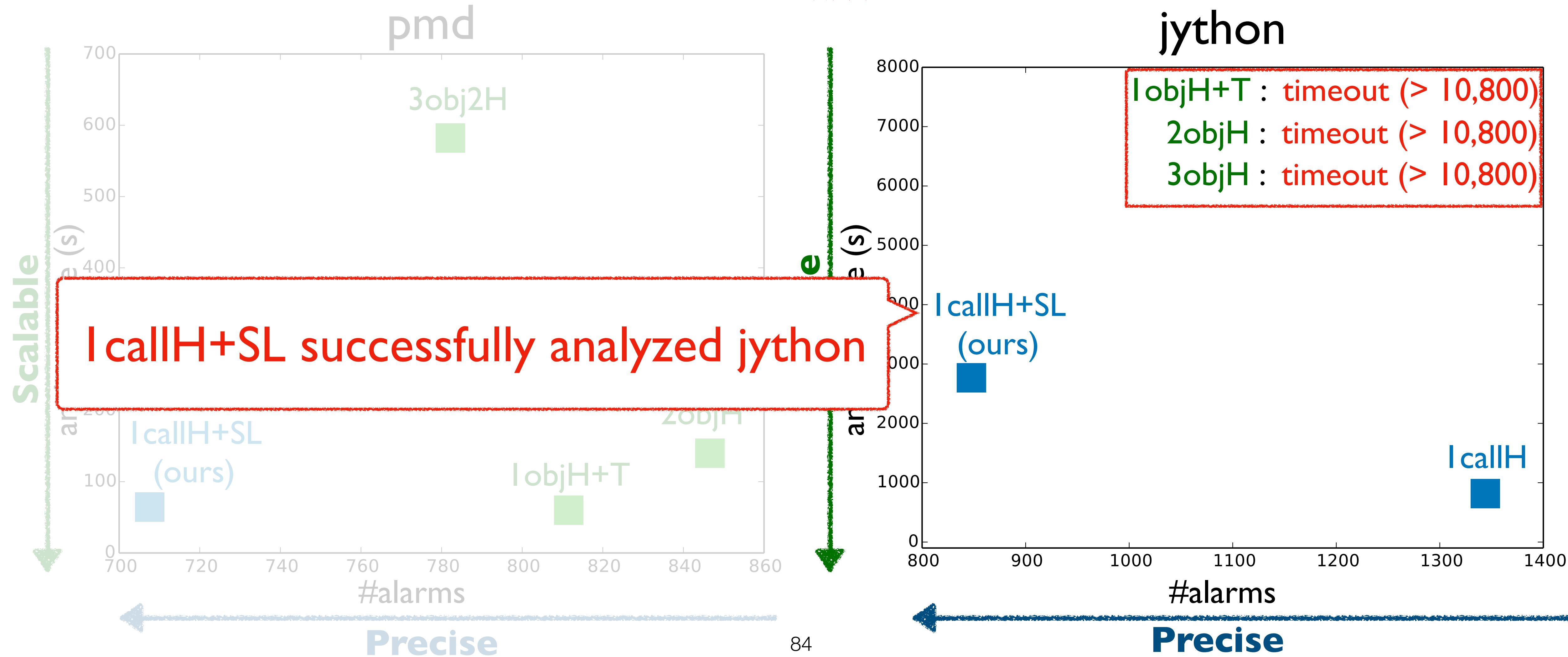
Call-site Sensitivity vs Object Sensitivity

- $I_{callH+SL}$ (ours) is more precise and scalable than the existing object sensitivities



Call-site Sensitivity vs Object Sensitivity

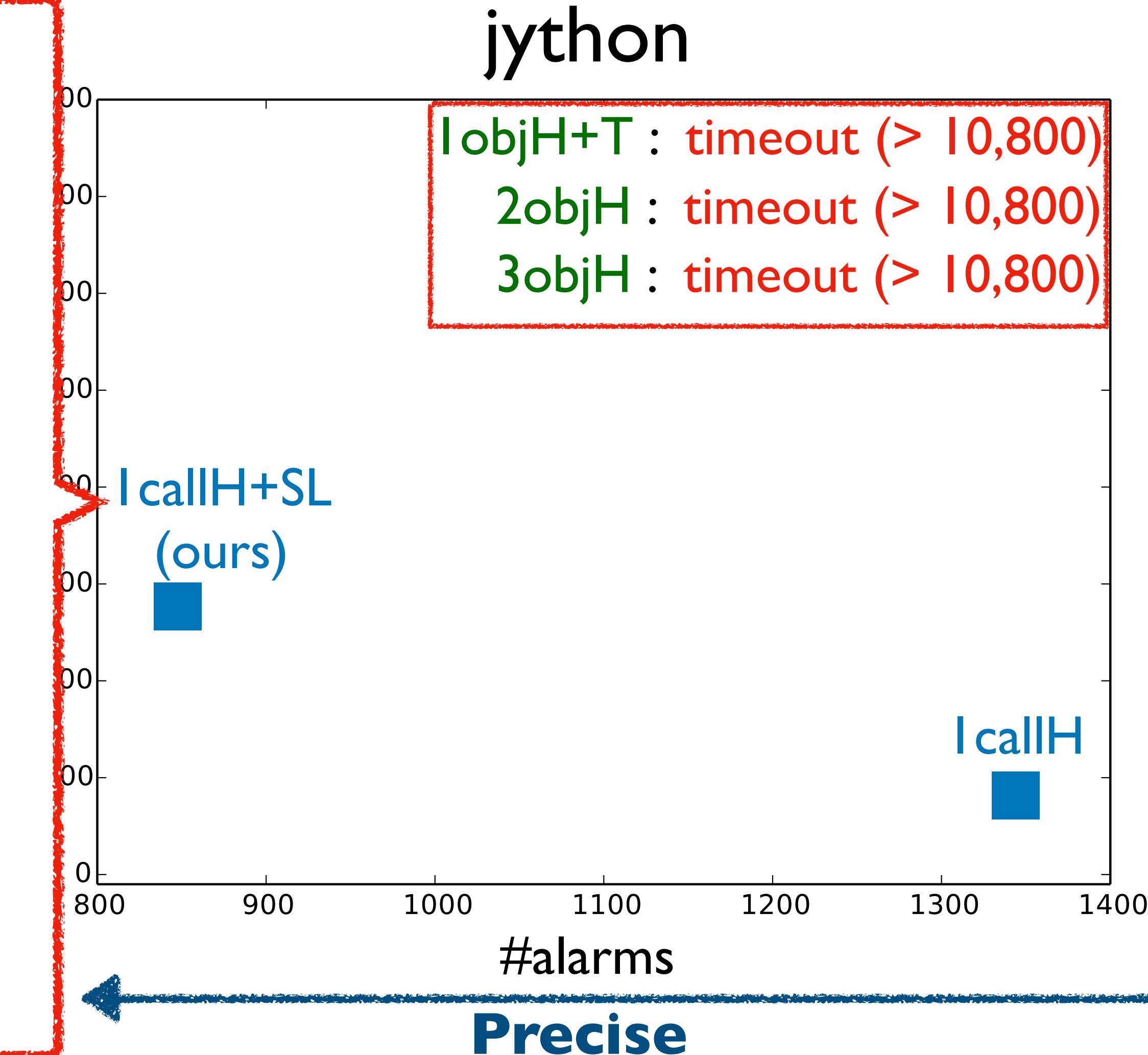
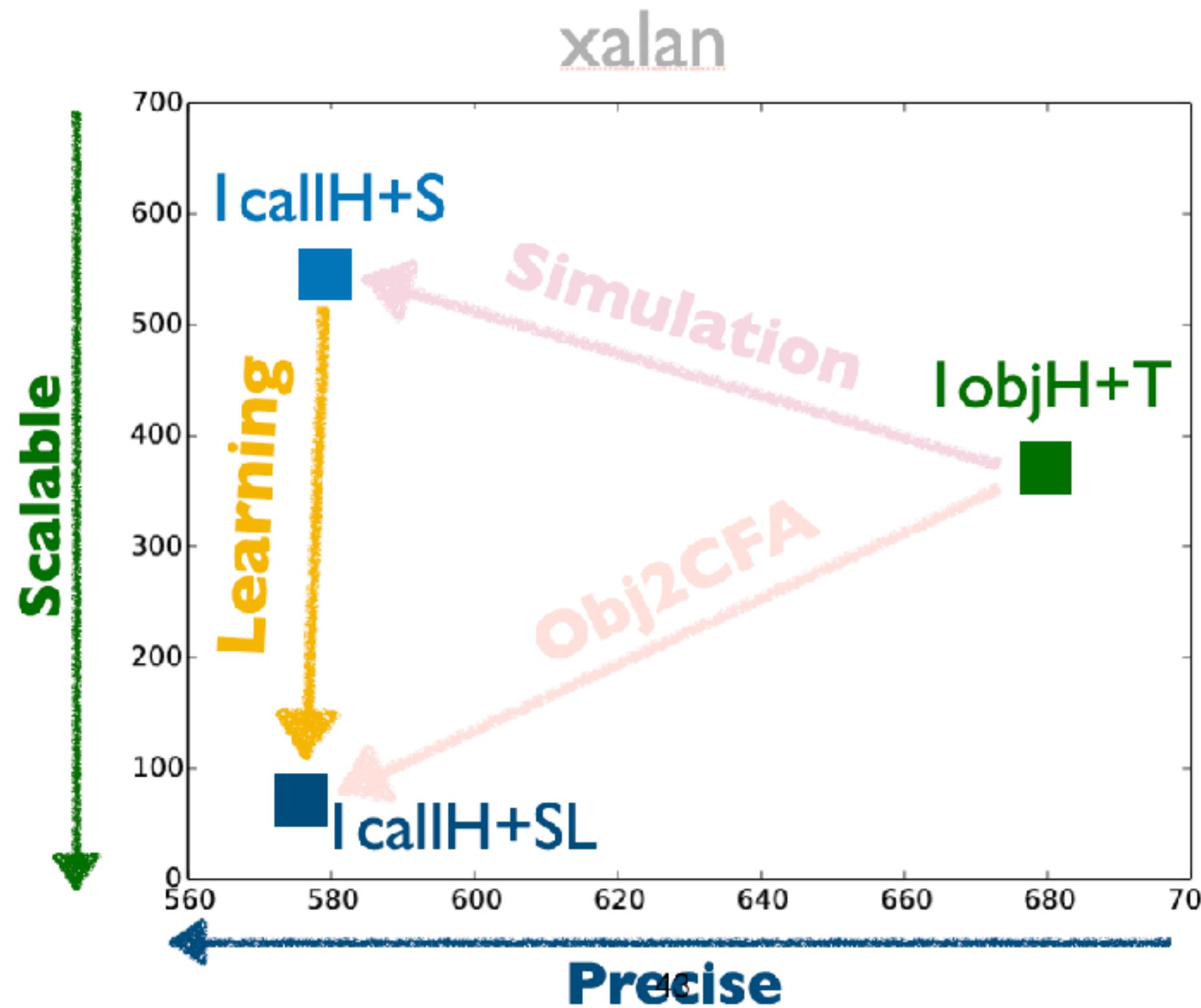
- $I_{callH+SL}$ (ours) is more precise and scalable than the existing object sensitivities



Call-site Sensitivity vs Object Sensitivity

- $I_{callH+SL}$ (ours) is more precise and scalable than the existing object sensitivities

- Necessity of learning
- $I_{callH+S}$ is unable to analyze jython



Summary

- Currently, CFA is known as a bad context
- However, if context tunneling is included, CFA is not a bad context anymore
- We need to reconsider CFA from now on

Thank you

Summary

- Currently, CFA is known as a bad context

- Call-site Sensitivity has been ignored

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



1981

2002

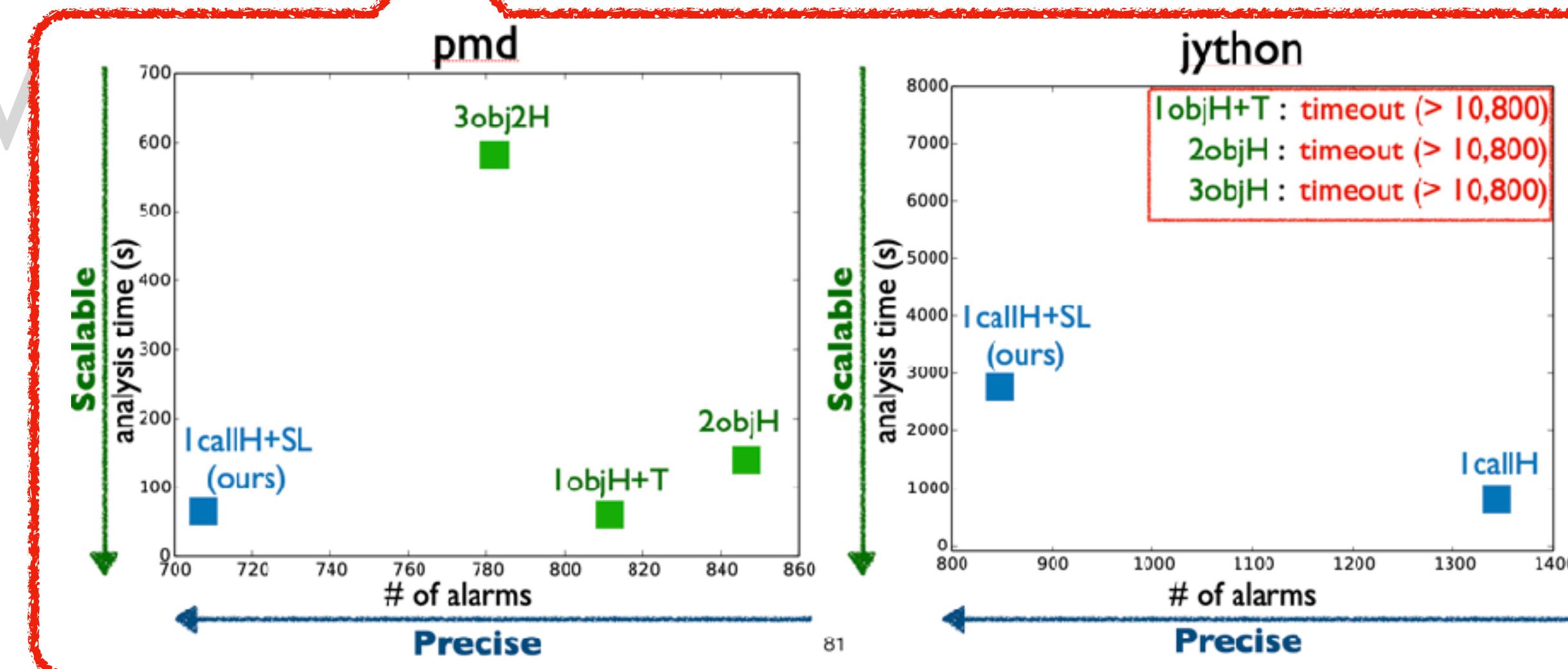
2010

2022

Summary

- Currently, CFA is known as a bad context

• However, if context tunneling is included,
CFA is not a bad context anymore



With context tunneling now on

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 Deep 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

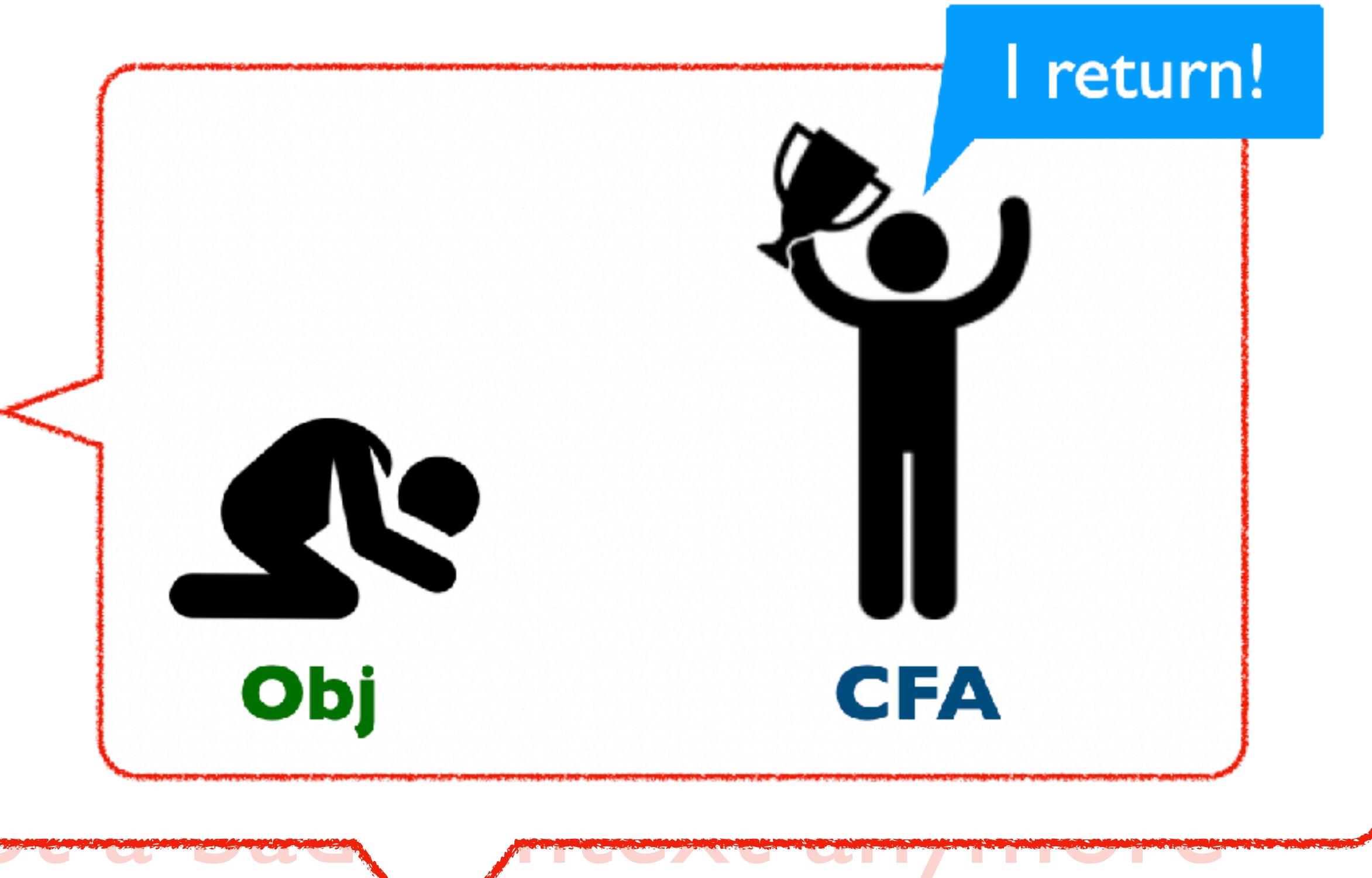
"Since its introduction, object sensitivity has emerged as the dominant flavor of context sensitivity for object-oriented languages."

—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 necessary.

CFA wins!

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



- We need to reconsider CFA from now on

Thank you