



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

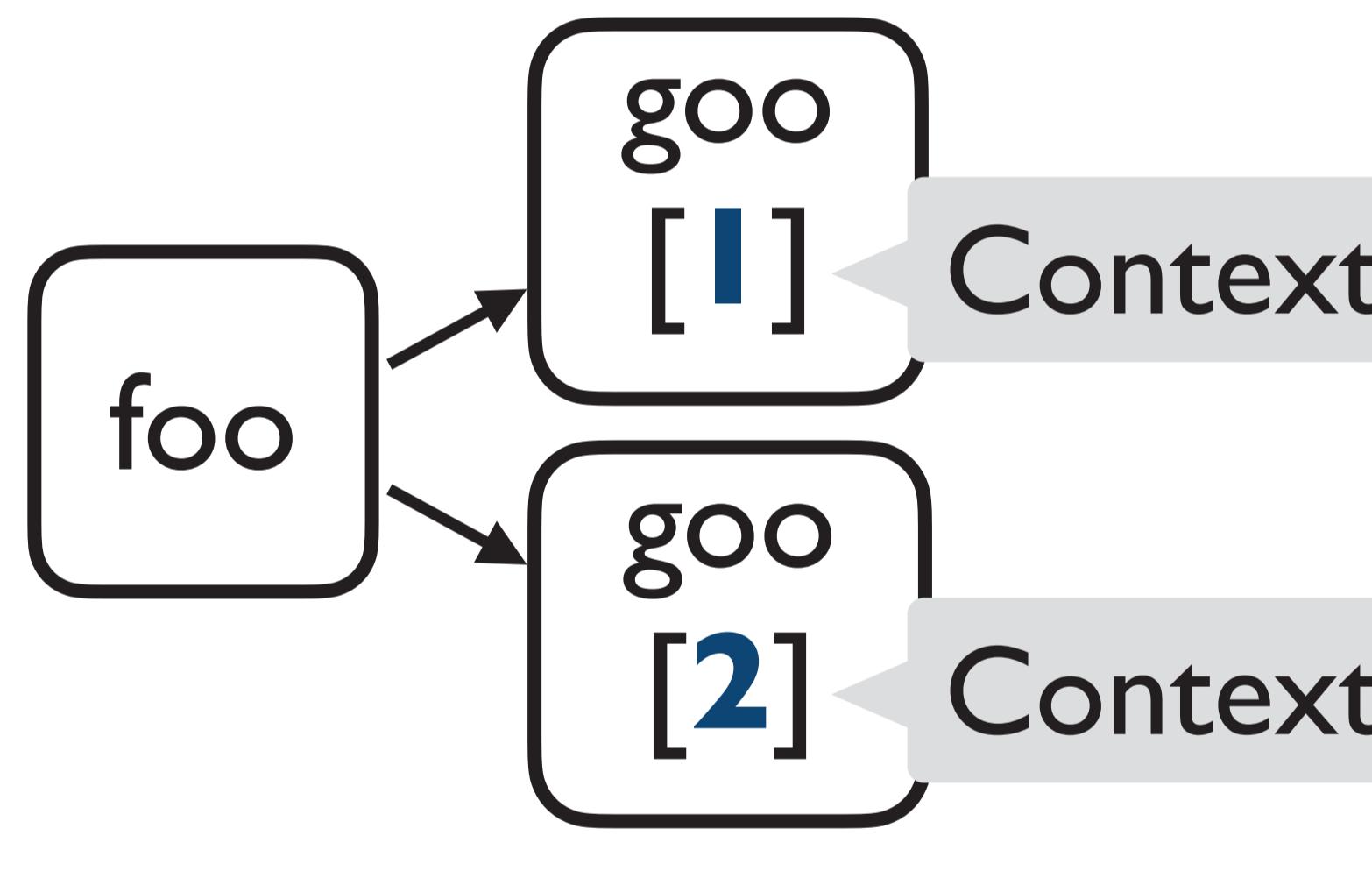


Call-site Sensitivity vs Object Sensitivity

Call-site sensitivity was born in 1981

- Considers “**Where**”

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



Call-site sensitivity



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 graph

Call-site sensitivity

1981

2002

2010

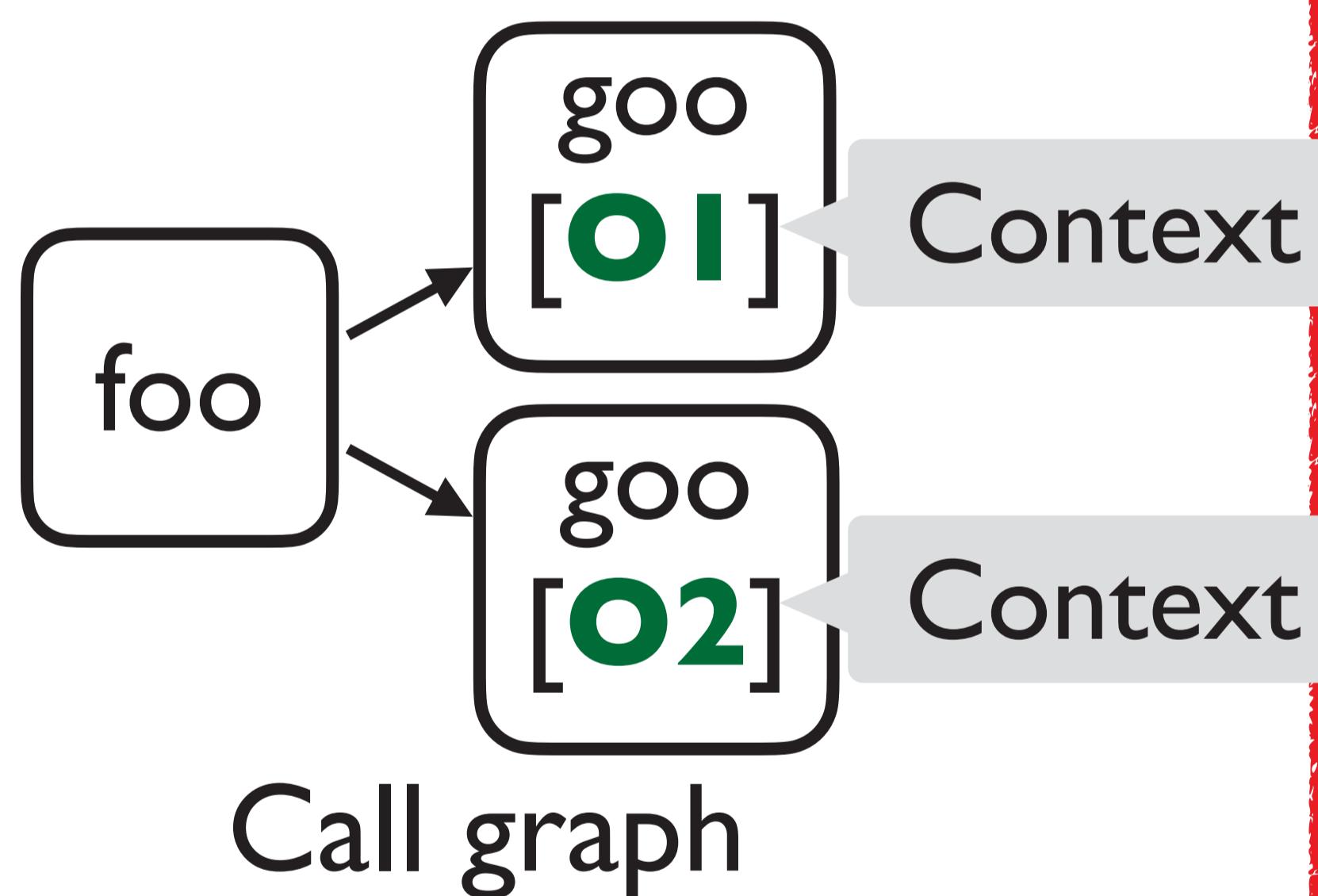
2022

Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

- Considers “**What**”

```
01 or 02
0: foo(p){
1:   p.goo();
2: }
```



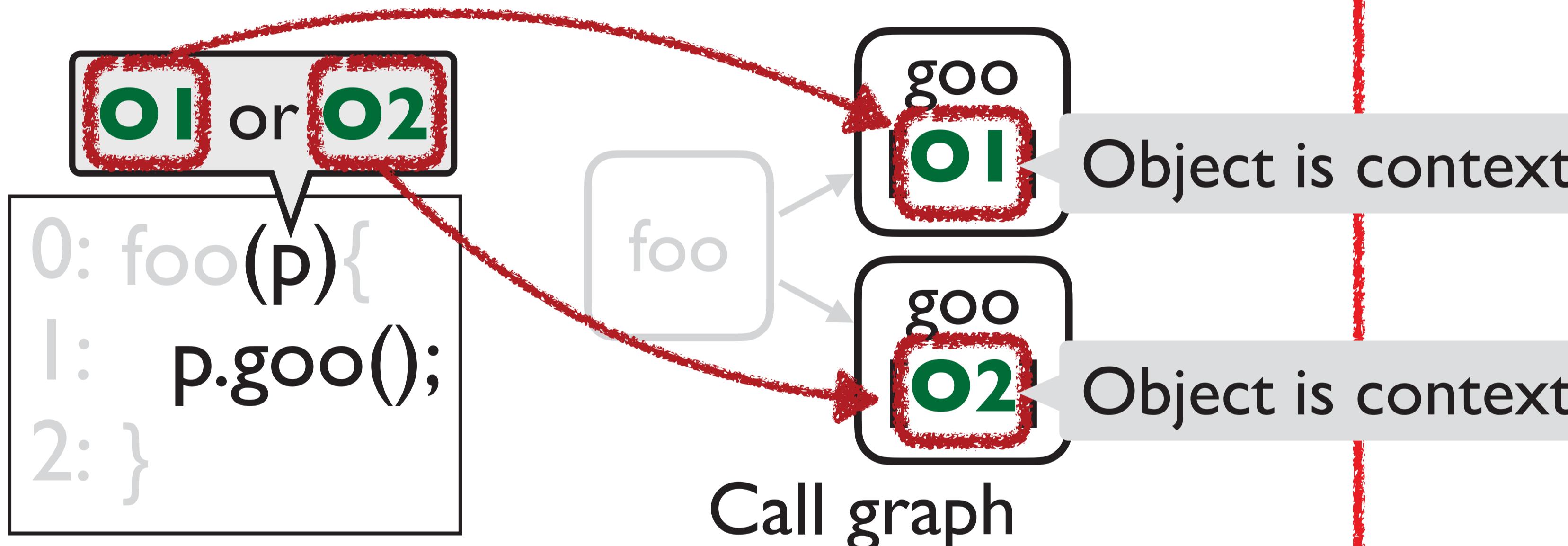
Object sensitivity



Call-site Sensitivity vs Object Sensitivity

Object sensitivity appeared in 2002

- Considers “**What**”



What is it called with?



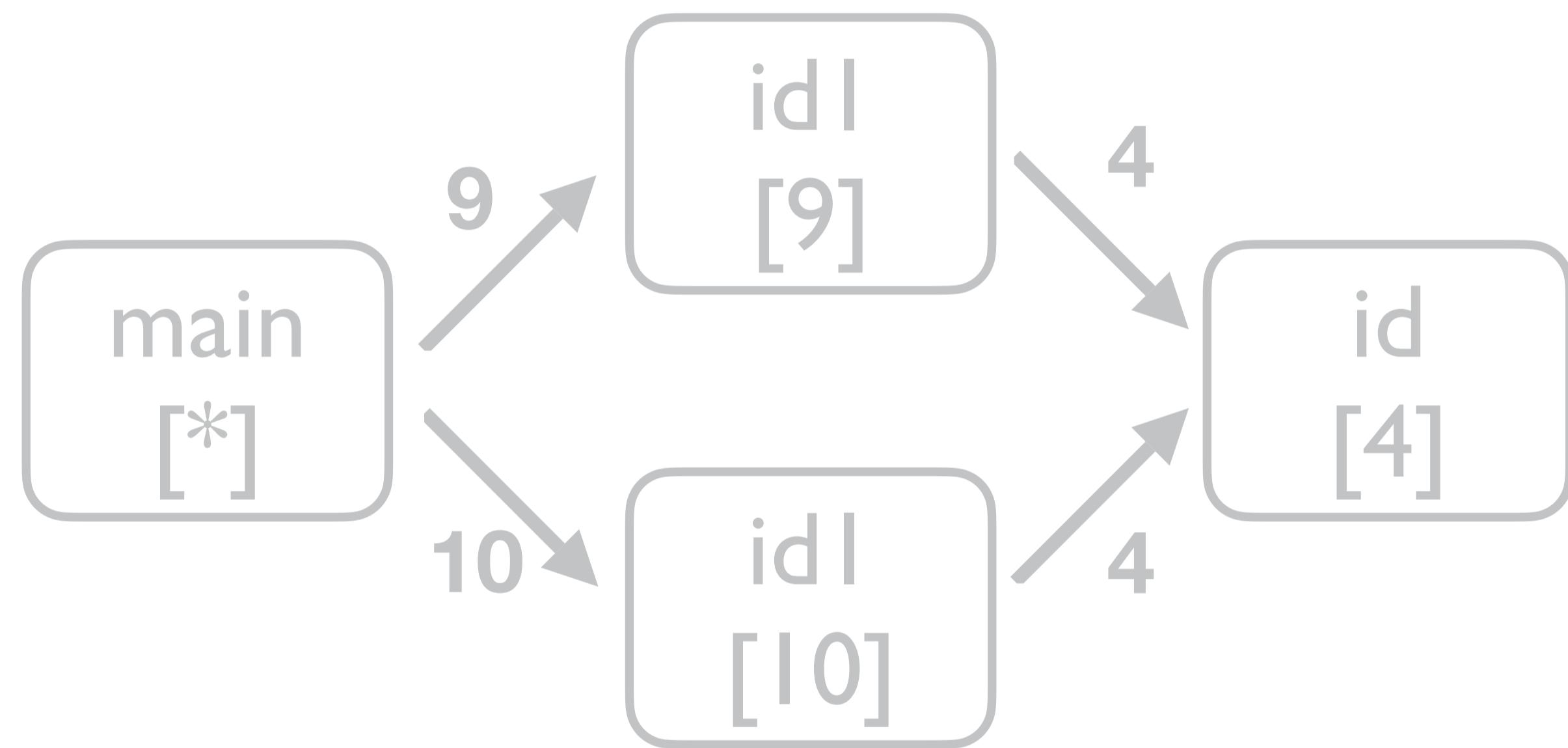
Object sensitivity



Call-site Sensitivity vs Object Sensitivity

- 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();  
8:     c2 = new C();  
9:     a = (A) c1.id1(new A());  
10:    b = (B) c2.id1(new B());  
11:  }
```



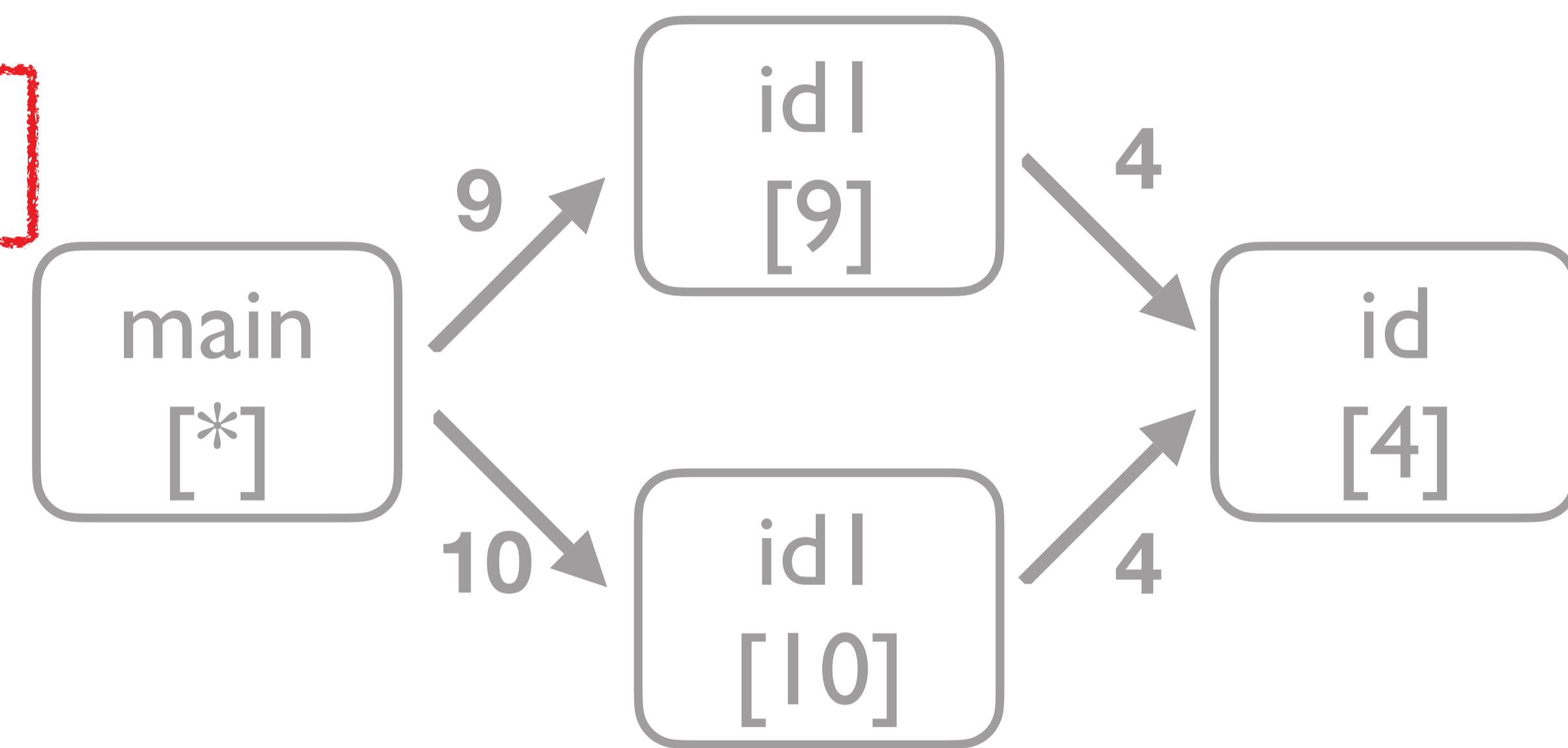
Call-graph of I-CFA

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



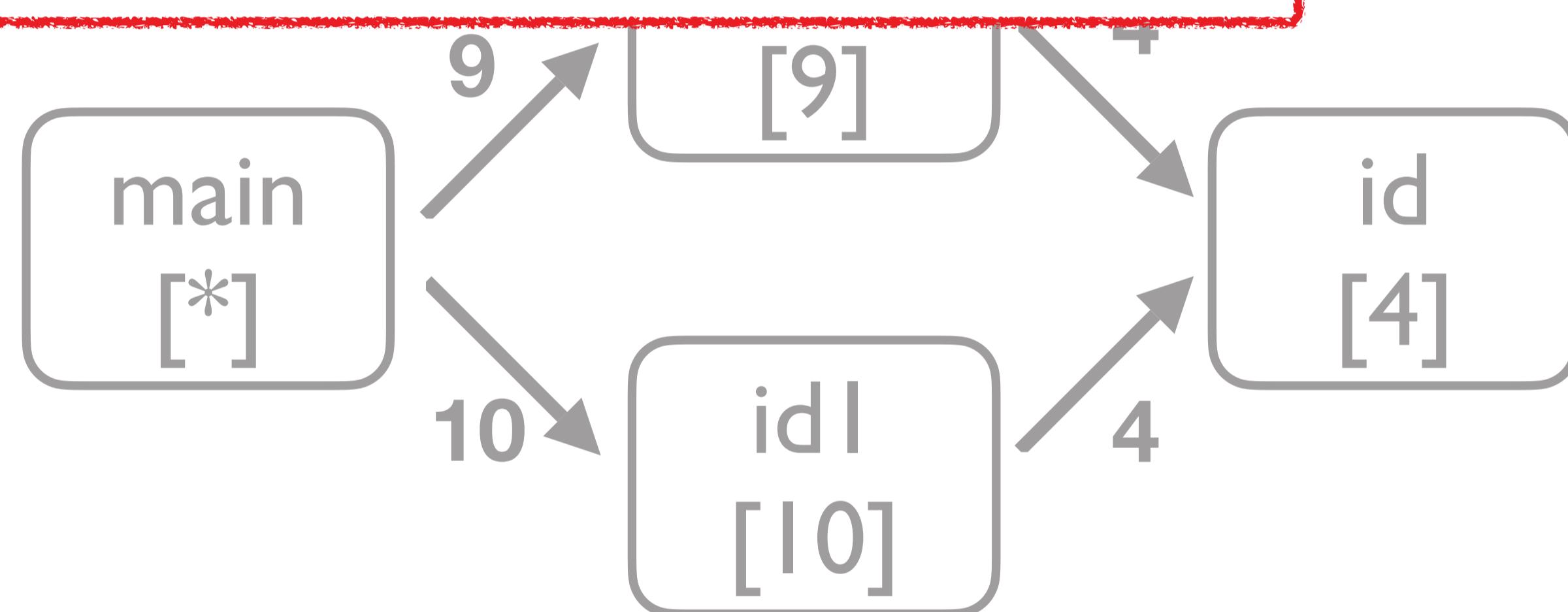
Call-graph of I-CFA

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



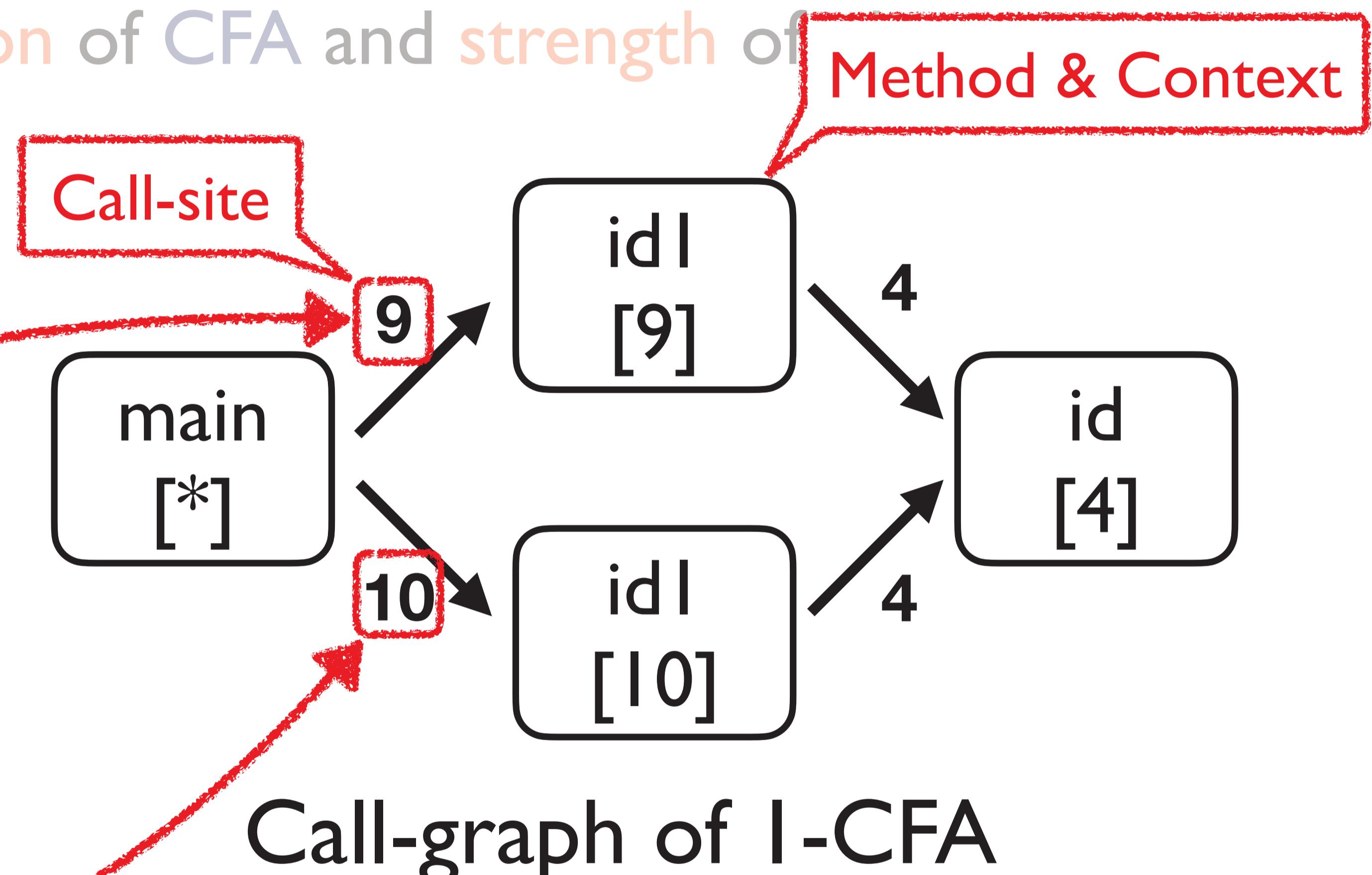
Call-graph of I-CFA

Call-site Sensitivity vs Object Sensitivity

- An example shows the limitation of CFA and strength of

Method & Context

```
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: }
```

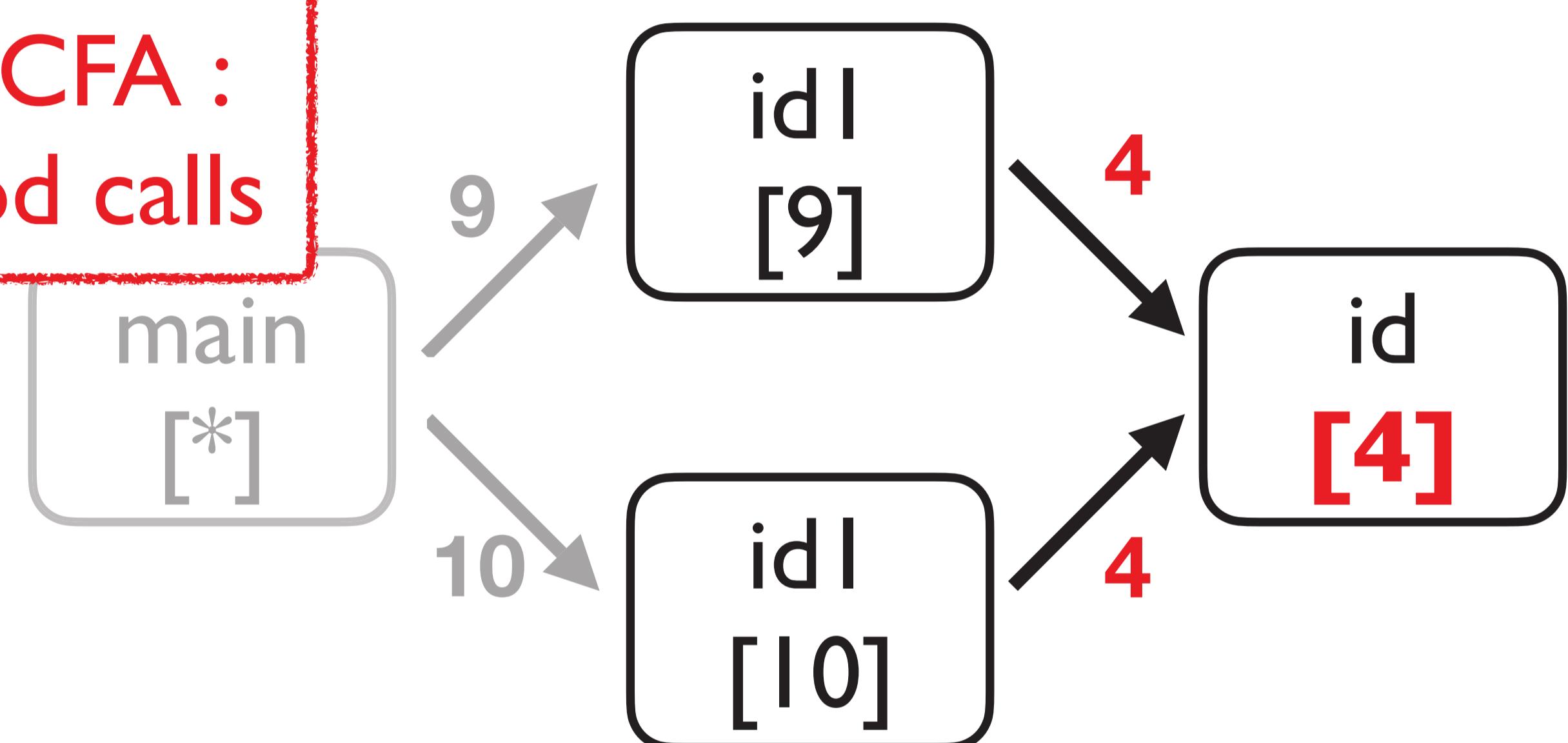


Call-site Sensitivity vs Object Sensitivity

- 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

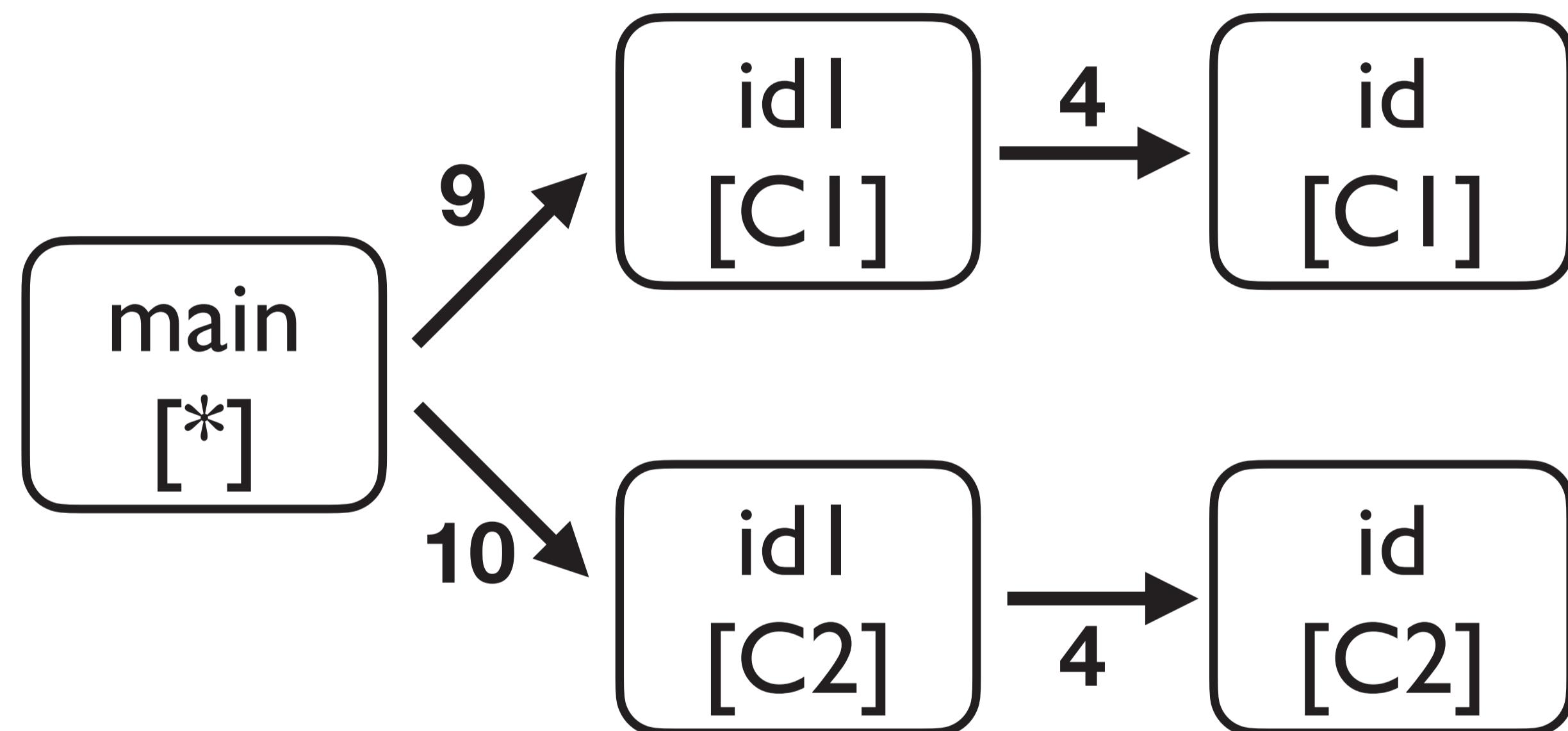


Call-graph of I-CFA

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: }
```

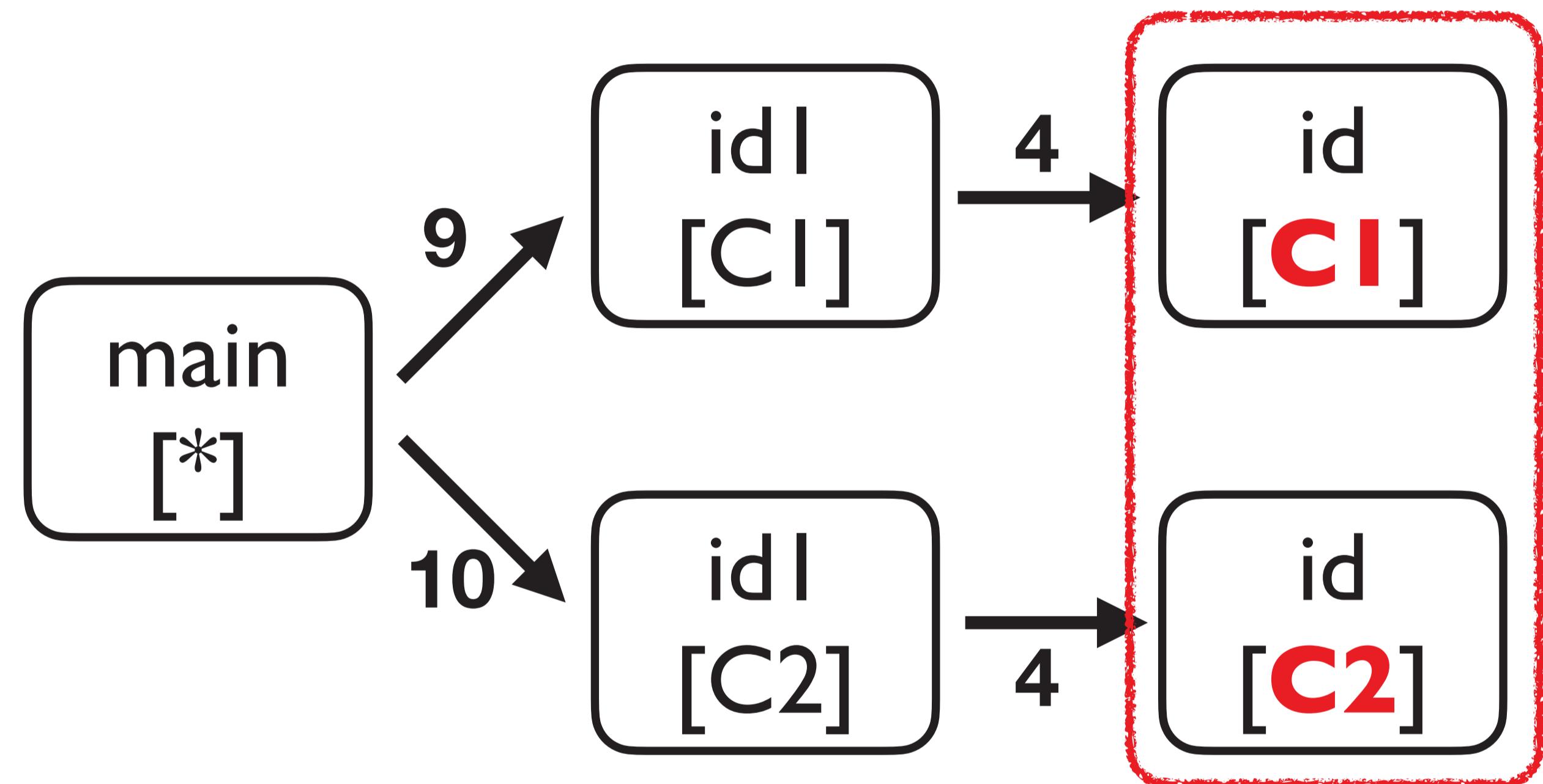


Call-graph of I-Obj

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:   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: }
```

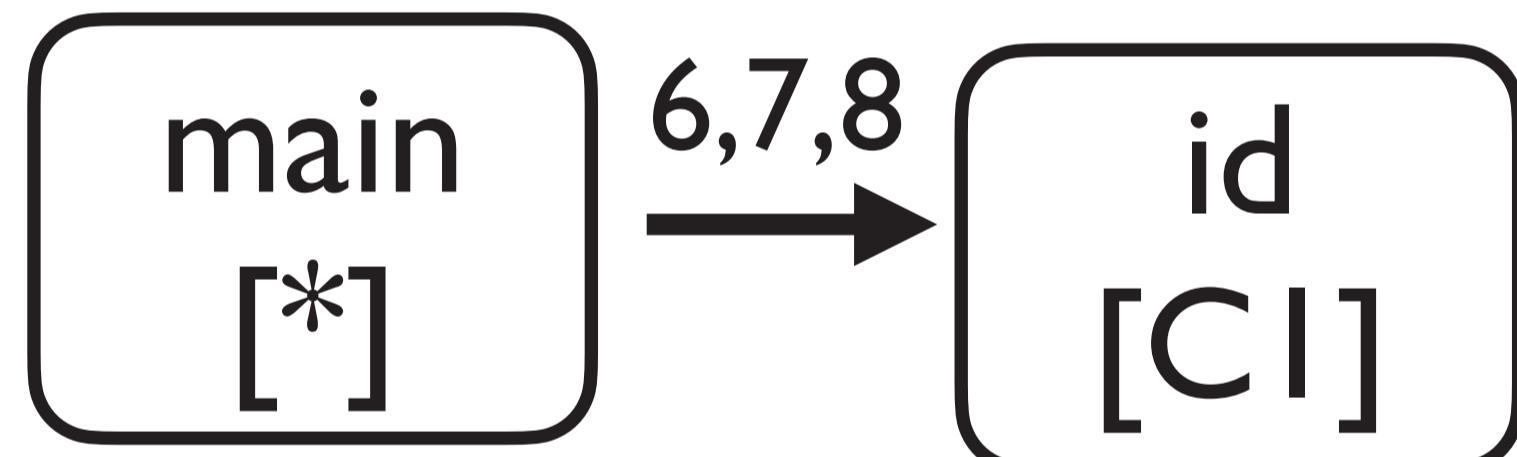


Call-graph of I-Obj

Call-site Sensitivity vs Object Sensitivity

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

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

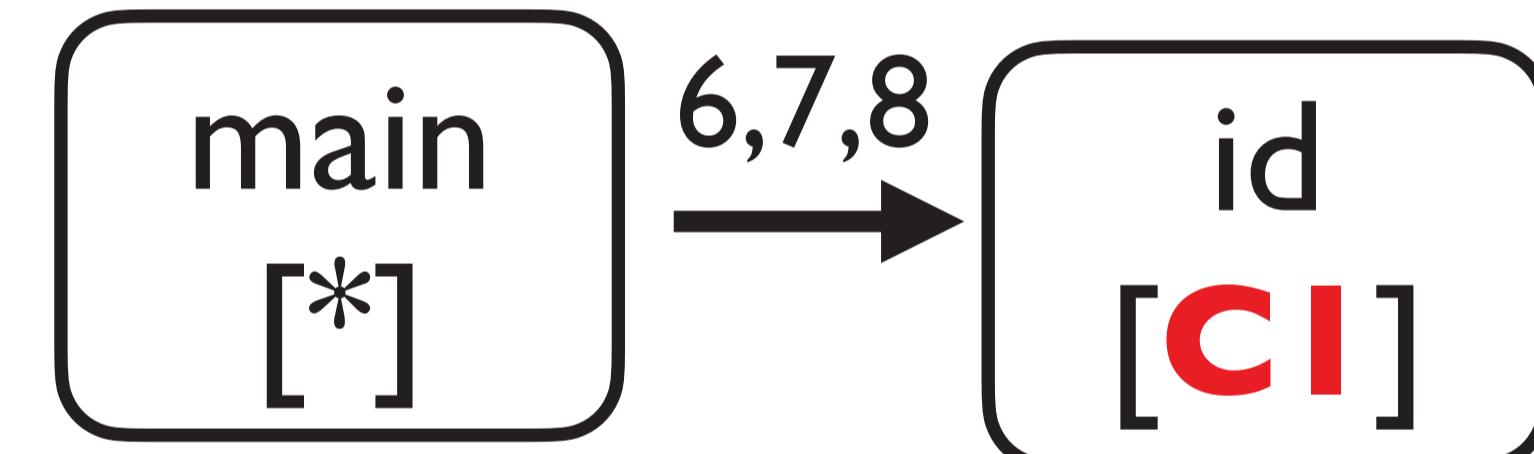


Call-graph of I-Obj

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();  
6:   a = (A) cl.id(new A());  
7:   b = (B) cl.id(new B());  
8:   c = (B) cl.id(new C());  
9: }
```



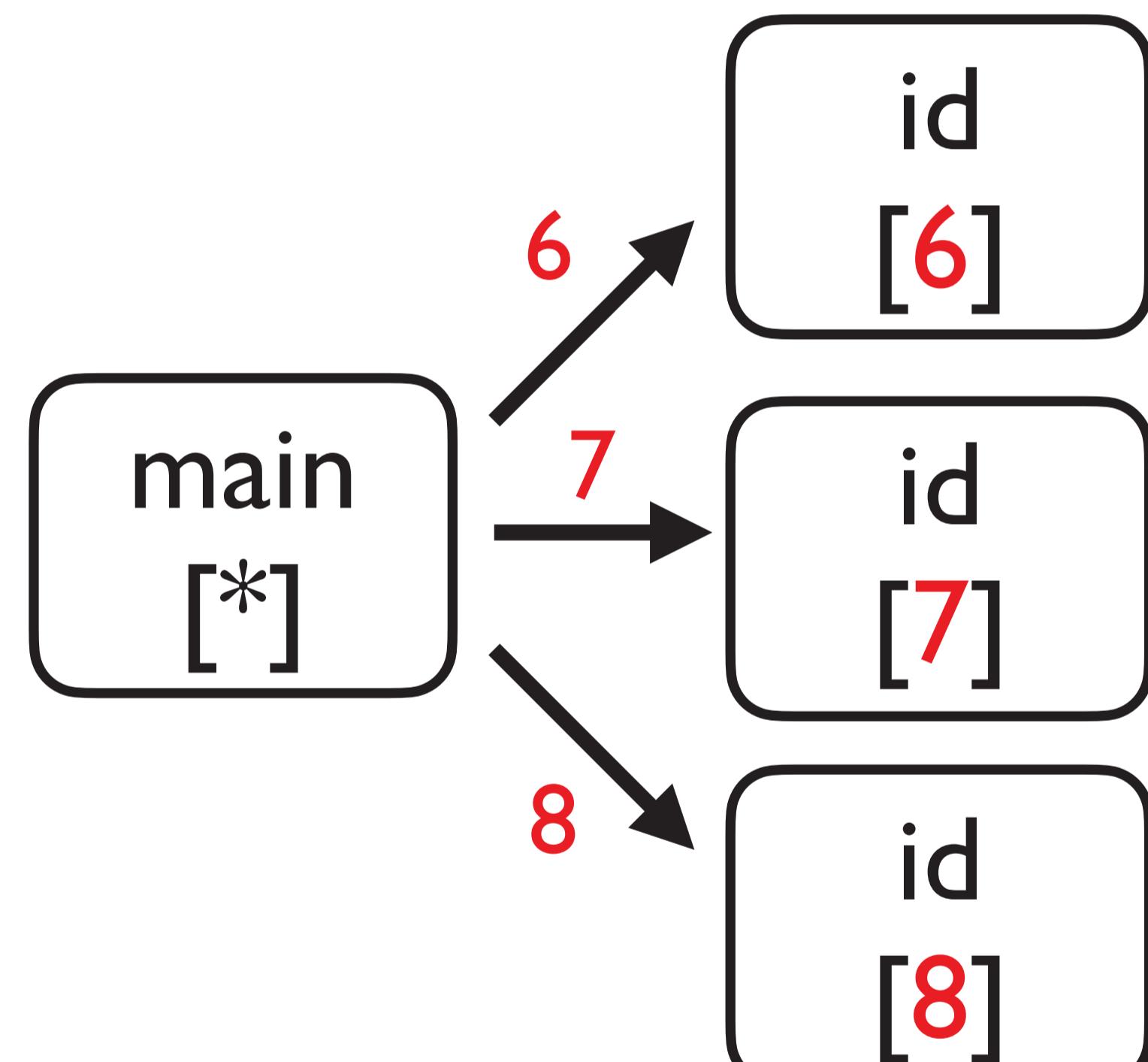
Call-graph of I-Obj

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();  
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

Parameterized Object Sensitivity for Points-to Analysis for Java

ANA MILANOVA
Rensselaer Polytechnic Institute
ATANAS ROUNTEV
Ohio State University
and
BARBARA G. RYDER
Rutgers University

The goal of *points-to analysis* for Java is to determine the set of objects pointed to by a reference variable or a reference object field. We present *object sensitivity*, a new form of *context sensitivity* for flow-insensitive points-to analysis for Java. The key idea of our approach is to analyze a method separately for each of the object names that represent run-time objects on which this method may be invoked. To ensure feasibility and practicality, we propose a parameterized framework that allows for the reuse of the analysis of the objects that are passed as parameters to the methods of interest.

Side-effect analysis determines the memory locations that may be modified by the execution of a program statement. *Door* analysis identifies pairs of statements that set the value of a memory location and subsequently use that value. The information computed by such analyses has a wide variety of applications. In this paper, we present new versions of these analyses that are based on object-sensitive points-to analysis.

We have implemented two instantiations of our parameterized object sensitivity framework for Java. Our experiments show that these analyses have comparable cost to a context-insensitive points-to analysis for Java which is based on Anderson's analysis for C. Our results also show that object sensitivity significantly improves the precision of side-effect analysis and door analysis. We also show that object sensitivity significantly improves the precision of points-to analysis that models context using the invoking call site. These experiments demonstrate that object-sensitive analyses achieve substantial precision improvement, while at the same time remaining efficient and practical.

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

ONDŘEJ Lhoták^{1,2} and Laurie Hendren²
olhotak@uwaterloo.ca hendren@sable.mcgill.ca

¹ School of Computer Science, University of Waterloo, Waterloo, ON, Canada
² School of Computer Science, McGill University, Montreal, QC, Canada

Abstract. We present the results of an empirical study evaluating the cost of context-sensitive points-to analysis for Java. We compare the use of call site strings as the context abstraction, object sensitivity, and the BDD-based context-sensitive algorithm proposed by Zhu and Calman, and by Whaley and Lai. Our study includes analyses for context abstraction, object sensitivity, as well as ones that use the heap abstraction. We empirically evaluate the precision of context abstraction, object sensitivity, and the BDD-based algorithm. We show that the points-to sets themselves, as well as effects on the precision of client analyses. To guide development of efficient analysis implementations, we measure the number of contexts, the number of distinct contexts, and the number of distinct points-to sets that arise with each abstraction. To evaluate precision, we measure the size of the call graph in terms of methods and edges, the number of derivable call sites, and the number of casts statically provable to be safe.

The empirical study indicates that object sensitivity implementations are likely to scale better and be more precise than the other approaches, that object-sensitive analyses are more precise than comparable variations of the other approaches, and that the heap abstraction improves precision more than extending the length of context strings.

Categories and Subject Descriptors: D.3.4 [Programming Languages]: Processors; D.3.3 [Programming Languages]: Language Constructs and Features

General Terms: Languages, Design, Experimentation, Measurement

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. Softw. Eng. Method.* 18, 1, Article 3 (September 2008), 55 pages. DOI: 10.1145/1391984.1391987. <http://doi.acm.org/10.1145/1391984.1391987>

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

ONDŘEJ Lhoták¹ and LAURIE HENDREN²
University of Waterloo and
McGill University

Abstract
We present *Door*, a framework for points-to analysis of Java programs. *Door* builds on the idea of specifying pointer analysis algorithms declaratively, using *DoorLog*: a logic-based language for defining (recursive) relations. We carry out the analysis by translating *DoorLog* into a BDD-based context abstraction, which captures a static notion of the dynamic context of a method. Typical contexts include abstractions of method calls and of code regions. *Door* also supports the notion of “context abstraction” or recursive contexts (for an *object-sensitive* analysis).

Categories and Subject Descriptors: F.3.3 [Programming Languages]: Processors; F.3.3 [Programming Languages]: Language Constructs and Features

General Terms: Languages, Design, Experimentation, Measurement

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. Softw. Eng. Method.* 18, 1, Article 3 (September 2008), 55 pages. DOI: 10.1145/1391984.1391987. <http://doi.acm.org/10.1145/1391984.1391987>

Strictly Declarative Specification of Sophisticated Points-to Analyses

MARTIN BRAVENECER¹ and YANNIS SMARAGDIS²
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003, USA
martin.brauenber@acm.org yannis@cs.umass.edu

Abstract
We present the *Door* framework for points-to analysis of Java programs. *Door* builds on the idea of specifying pointer analysis algorithms declaratively, using *DoorLog*: a logic-based language for defining (recursive) relations. We carry out the analysis by translating *DoorLog* into a BDD-based context abstraction, which captures a static notion of the dynamic context of a method. Typical contexts include abstractions of method calls and of code regions. *Door* also supports the notion of “context abstraction” or recursive contexts (for an *object-sensitive* analysis).

In this paper, we present *Door*, a general and versatile points-to analysis framework that makes feasible the most precise context-sensitive analyses reported in the literature. Our implementation of *Door* matches the state of the art for context-sensitive analyses. For the exact same logic, *Door* is orders of magnitude faster than the most recent context-sensitive analyses. Our approach is also more precise: it can handle pointer analysis on Java benchmarks, calculate semantics of object-sensitive analyses, all specified modularly as variations on a common code base. Compared to the prior state of the art, *Door* often achieves a higher precision for the same cost, and it is much faster for several different analyses.

The main elements of our approach are the use of the *DoorLog* language for specifying the program analysis, and the aggressive optimization of the *DoorLog* program. The use of *DoorLog* for pointer analysis (cf. [13, 23, 29]) and the use of BDDs for pointer analysis (cf. [1, 2, 11, 24]) is a high-level view of our approach. Our novel approach is, in fact, much more complex. Our approach, however, accounts for several orders of magnitude of performance improvement: unoptimized analyses typically take minutes to hours to complete, while our optimized analyses fit well the approach of handling program facts as a database, by specifically targeting the indexing scheme and the indexing overhead of the *DoorLog* program. Furthermore, our approach is entirely *DoorLog* based, encoding the logic required both for call graph construction and for pointer analysis in a single declarative language. Our approach is also modular, and it is also highly reusable. The Java language (e.g., static initialization, finalization, reference objects, threads, exceptions, reflection, etc.) This makes our approach highly reusable, and it is also highly modular, but also efficient and easy to tune. Generally, our work is a strong data point in support of declarative languages: we are able to implement and optimize complex mutually-recursive definitions at an operational level of abstraction. On the other

Categories and Subject Descriptors: F.3.3 [Programming Languages]: Processors; F.3.3 [Programming Languages]: Language Constructs and Features

General Terms: Languages, Algorithms, 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 wealth of research has been devoted to efficient and precise pointer analysis techniques. *Context-sensitive* analyses are the most common class of precise points-to analyses. Context-sensitive analysis is a generalization of pointer analysis, including contexts that are shared by multiple pointers. Context-sensitive analyses are more precise than comparable variations of the other approaches, and the heap abstraction improves precision using a novel technique specifically targeting highly dynamic programs.

2. Related Work
As a result, *Door* achieves several benefits, including full order-of-magnitude improvements in runtime. We compare *Door* with the state-of-the-art pointer analysis, *Door* matches the state-of-the-art for context-sensitive analyses. For the exact same logic, *Door* is orders of magnitude faster than the most recent context-sensitive analyses. Our approach is also more precise: it can handle pointer analysis on Java benchmarks, calculate semantics of object-sensitive analyses, all specified modularly as variations on a common code base. Compared to the prior state of the art, *Door* often achieves a higher precision for the same cost, and it is much faster for several different analyses.

3. The Door Framework
The main elements of our approach are the use of the *DoorLog* language for specifying the program analysis, and the aggressive optimization of the *DoorLog* program. The use of *DoorLog* for pointer analysis (cf. [13, 23, 29]) and the use of BDDs for pointer analysis (cf. [1, 2, 11, 24]) is a high-level view of our approach. Our novel approach is, in fact, much more complex. Our approach, however, accounts for several orders of magnitude of performance improvement: unoptimized analyses typically take minutes to hours to complete, while our optimized analyses fit well the approach of handling program facts as a database, by specifically targeting the indexing scheme and the indexing overhead of the *DoorLog* program. Furthermore, our approach is entirely *DoorLog* based, encoding the logic required both for call graph construction and for pointer analysis in a single declarative language. Our approach is also modular, and it is also highly reusable. The Java language (e.g., static initialization, finalization, reference objects, threads, exceptions, reflection, etc.) This makes our approach highly reusable, and it is also highly modular, but also efficient and easy to tune. Generally, our work is a strong data point in support of declarative languages: we are able to implement and optimize complex mutually-recursive definitions at an operational level of abstraction. On the other

Obj vs CFA

1981 2002 2010 2022

17

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
Rensselaer Polytechnic Institute
ATANAS ROUNTEV
Ohio State University
and
BARBARA G. RYDER
Rutgers University

The goal of *points-to analysis* for Java is to determine the set of objects pointed to by a reference variable or a reference object field. We present *object sensitivity*, a new form of *context sensitivity* for flow-insensitive points-to analysis for Java. The key idea of our approach is to analyze a method separately for each of the object names that represent run-time objects on which this method is invoked. To ensure feasibility and practicality, we propose a parameterized framework that allows us to reuse the analysis of a method for all objects that it can possibly point to in the future. This analysis is called *parameterized object sensitivity*.

Side-effect analysis determines the memory locations that may be modified by the execution of a program statement. *Dot/are analysis* identifies pairs of statements that set the value of a memory location and subsequently use that value. The information computed by such analyses has a wide variety of applications, including program verification, work flow analysis, and detection of those analyses that are based on object-sensitive points-to analysis.

We have implemented two instantiations of our parameterized object sensitivity framework for analysis of Java programs. Our experiments show that these analyses have comparable cost to a context-insensitive points-to analysis for Java which is based on Anderson's analysis for C. Our results also show that object sensitivity significantly improves the precision of side-effect analysis and dot/are analysis. We also show that object sensitivity significantly improves the precision of flow-insensitive points-to analysis that models context using the invoking call site. These experiments demonstrate that object-sensitive analyses achieve substantial precision improvement, while at the same time remaining efficient and practical.

A preliminary version of this article appeared in *Proceedings of the International Symposium on Software Testing and Analysis (July)*, 2002, pp. 1-11.

This research was partially funded by the National Science Foundation (NSF) grant CCR-9900988. Author's addresses: A. Milanova, Department of Computer Science, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180; email: milanova@rpi.edu; A. Rountev, Department of Computer Science, Ohio State University, 196 W. Woodruff Avenue, Columbus, OH 43210; email: rountev@cs.ohio-state.edu; B. G. Ryder, Department of Computer Science, Rutgers University, 100 Frelinghuysen Road, Piscataway, NJ 08854; email: ryder@cs.rutgers.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyright for contents of this work owned by others than ACM must be

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

ONDŘEJ LHOŠTÁK^{1,2} and LAURIE HENDREN²
olhotak@uwaterloo.ca hrenden@sable.mcgill.ca

¹ School of Computer Science, University of Waterloo, Waterloo, ON, Canada
² School of Computer Science, McGill University, Montreal, QC, 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 sets as the context abstraction, object sensitivity, and the BDD-based context-sensitive algorithm proposed by Zhu and Calman, and by Whaley and Lai. Our study includes analysis of contexts that specialize only pointer variables, as well as ones that also specialize the heap abstraction. We also compare the precision of points-to sets themselves, as well as effects on the precision of client analyses. To guide development of efficient analysis implementations, we measure the number of contexts, the number of distinct contexts, and the number of distinct pointers to sets that arise with each analysis.

To evaluate precision, we measure the size of the call graph in terms of methods and edges, the number of derivable call sites, and the number of casts statically provable to be safe.

The empirical study indicates that object-sensitive analysis implementations are likely to scale better and be more precise than the other approaches; that object-sensitive analyses are more precise than comparable variations of the other approaches; and that specializing the heap abstraction improves precision more than specializing the call site abstraction. To evaluate precision, we measure the size of the call graph in terms of methods and edges, the number of derivable call sites, and the number of casts statically provable to be safe.

The empirical study indicates that object-sensitive analysis implementations are likely to scale better and be more precise than the other approaches; that object-sensitive analyses are more precise than comparable variations of the other approaches; and that specializing the heap abstraction improves precision more than specializing the call site abstraction. To evaluate precision, we measure the size of the call graph in terms of methods and edges, the number of derivable call sites, and the number of casts statically provable to be safe.

Categories and Subject Descriptors: D.3.4 [Programming Languages]: Processors; D.3.3 [Programming Languages]: Language Constructs and Features

General Terms: Languages, Design, Experimentation, Measurement

Additional Key Words and Phrases: Interprocedural program analysis, context sensitivity, binary analysis, BDDs, heap abstraction, points-to analysis, call graph construction, cast safety 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. Softw. Eng. Method.* 18, 1, Article 3 (September 2008), 55 pages. DOI = 10.1145/1391984.1391987. <http://doi.acm.org/10.1145/1391984.1391987>

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

ONDŘEJ LHOŠTÁK¹ and LAURIE HENDREN²
University of Waterloo and
McGill University

Abstract
We present Points-to, a framework of BDD-based context-sensitive points-to and call graph analysis for Java, as well as client analysis that use their results. Points supports several variations of context-sensitive analyses, including call site strings and object sensitivity, and context-sensitively specializes both pointer variables and the heap abstraction. We empirically evaluate the precision of Points-to for contexts that specialize only pointer variables, as well as ones that also specialize the heap abstraction. We also compare the precision of points-to sets themselves, as well as effects on the precision of client analyses. To guide development of efficient analysis implementations, we measure the number of contexts, the number of distinct contexts, and the number of distinct pointers to sets that arise with each analysis.

To evaluate precision, we measure the size of the call graph in terms of methods and edges, the number of derivable call sites, and the number of casts statically provable to be safe.

The empirical study indicates that object-sensitive analysis implementations are likely to scale better and be more precise than the other approaches; that object-sensitive analyses are more precise than comparable variations of the other approaches; and that specializing the heap abstraction improves precision more than specializing the call site abstraction. To evaluate precision, we measure the size of the call graph in terms of methods and edges, the number of derivable call sites, and the number of casts statically provable to be safe.

Categories and Subject Descriptors: F.3.3 [Programming Languages]: Processors; D.3.3 [Programming Languages]: Language Constructs and Features

General Terms: Languages, Design, Experimentation, Measurement

Additional Key Words and Phrases: Interprocedural program analysis, context sensitivity, binary analysis, BDDs, heap abstraction, points-to analysis, call graph construction, cast safety 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. Softw. Eng. Method.* 18, 1, Article 3 (September 2008), 55 pages. DOI = 10.1145/1391984.1391987. <http://doi.acm.org/10.1145/1391984.1391987>

Strictly Declarative Specification of Sophisticated Points-to Analyses

MARTIN BRAVENECER¹ and YANNIS SMARAGDIS²
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003, USA
martin.brauenber@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 (recursive) relations. We carry out the analysis by translating the declarative logic into a context abstraction, which captures a static notion of the dynamic context of a method. Typical contexts include abstractions of method calls or of a code region. Dose also supports the notion of *context sensitivity* ("context-sensitive") or *recursion* (or both for an *object-sensitive* analysis).

In this work we present Dose, a general and versatile points-to analysis framework that makes feasible the most precise context-sensitive analyses reported in the literature. Dose implements a range of algorithms, including contextsensitive, call site strings, and object-sensitive analyses, all specified modularly as variations on a common code base. Compared to the prior state of the art, Dose often achieves orders of magnitude improvement in precision for several client analyses.

The main elements of our approach are the use of the Datalog language for specifying the program analysis, and the aggressive optimization of the Datalog program. The use of Datalog for program analysis (both for points-to and for high-level analysis) is a novel approach. Our novel approach, however, accounts for several orders of magnitude of performance improvement: unoptimized unoptimized analyses fit well the approach of handling program facts as a database, by specifically targeting the indexing scheme and the insertion/reinsertion of Datalog facts into the database. Furthermore, our approach is entirely Datalog based, encoding declaratively the logic required both for call graph construction and for context abstraction (e.g., pointer analysis, reference objects, threads, exceptions, reflection, etc.). This

Categories and Subject Descriptors: F.3.3 [Programming Languages]: Processors; D.3.3 [Programming Languages]: Language Constructs and Features

General Terms: Algorithms, Languages, Performance

Additional Key Words and Phrases: Points-to analysis, context sensitivity, binary analysis, BDDs, heap abstraction, Datalog

The timeline shows the evolution of analysis from 1981 to 2022. It starts with a bracket above the year 1981, followed by the year 2002, then a bracket above the year 2010, and ends with the year 2022. Below the timeline, the word 'Obj' is written next to 'CFA' (Context-Free Analysis). Above the timeline, there are four boxes, each containing the text 'Obj wins'. To the right of the timeline, there is a large black stick figure holding a trophy, with the word 'Obj' next to it, and a smaller black stick figure with the word 'CFA' next to it. The timeline is marked with vertical lines and horizontal bars, and the text is in a bold, sans-serif font.

18

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 the context for qualifying a method's local variables with the allocation site of the 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)

Object-sensitive pointer analysis

- Milanova, Rountev, and Ryder. *Parameterizing sensitivity for points-to analysis for Java*. ACM SIGART, 2005.
- Context-sensitive interprocedural pointer analysis
- For context, use stack of receiver objects
- (More next week?)

Pointer Analysis

1 Motivation for Pointer Analysis

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

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

In order to analyze this program correctly we must be aware of instruction 3 `p` points to `z`. If this information is available we can perform a *strong update* on variable `z`.

now
the essence of knowledge

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`.

Object-sensitive pointer analysis

Lecture Notes: Pointer Analysis
15-819O: Program Analysis
jonathan.aldrich@cs.cmu.edu
Lecture 9
Yannis Smaragdakis
University of Athens

I was also taught like that

Hakjoo Oh AAA616 2019 Fall, Lecture 8 November 18, 2019 27 / 31

National and Kapodistrian University of Athens

KOREA UNIVERSITY

VE R I
TAS
HARVARD

Carnegie Mellon University

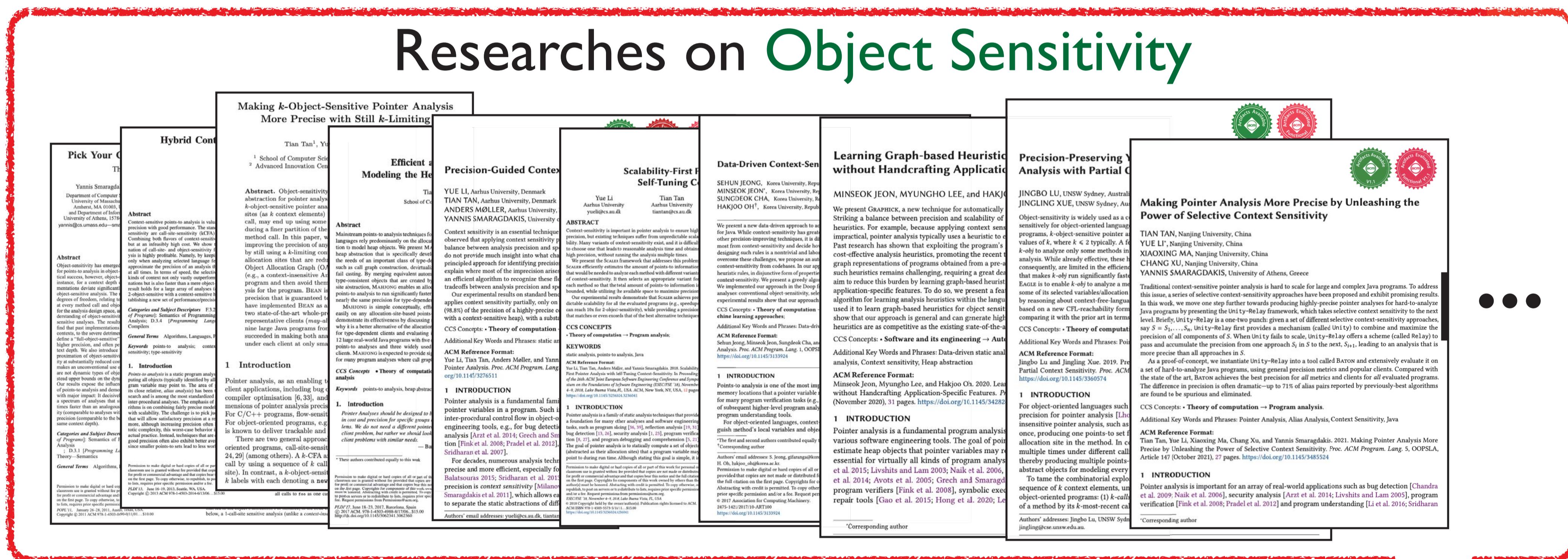
now the essence of knowledge

1981 2002 2010 2022

Call-site Sensitivity vs Object Sensitivity

- Researches focused on improving Object Sensitivity

Researches on Object Sensitivity



Obj

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
MINSEOK KIM, SEHUN JEONG, SUNGDEOK CHA, and HAKJOON OH¹,
YUN TAE KIM, YOUNG JAE KIM, and JUNGKEE KIM²
¹ School of Computer Science and Engineering, UNSW Australia
² Advanced Innovation Center for Imaging Technology, CNU, CR

Making k -Object-Sensitive Pointer Analysis More Precise with Still k -Limiting
Tian Tan¹, Yen Li¹, and Jingling Xie^{1,2*}
¹ School of Computer Science and Engineering, UNSW Australia
² Advanced Innovation Center for Imaging Technology, CNU, CR

Scalability-First Pointer Analysis: Self-Tuning Context-Sensitivity
Yue Li¹, Tian Tan¹, and Yannis Smaragdakis²
¹ School of Computer Science and Engineering, UNSW Australia
² Department of Informatics, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Pick Your Contexts Well: Understanding Object-Sensitivity
The Making of a Precise and Scalable Pointer Analysis
Yannis Smaragdakis
University of Massachusetts, Amherst, MA, USA
and Department of Informatics, University of Illinois at Urbana-Champaign, Urbana, IL, USA
Martin Breuer
LeapsInn Inc.
Two Madison Plaza
Austin, TX 78746, USA
martin.breuer@acm.org

Hybrid Context-Sensitivity for Points-To Analysis
George Karitsis¹, Yannis Smaragdakis²
¹ Department of Informatics, University of Illinois at Urbana-Champaign, Urbana, IL, USA
² Department of Informatics, University of Illinois at Urbana-Champaign, Urbana, IL, USA
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
George Karitsis¹, Yannis Smaragdakis²
¹ Department of Informatics, University of Illinois at Urbana-Champaign, Urbana, IL, USA
² Department of Informatics, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Precision-Guided Context Sensitivity for Pointer Analysis
Tania Smaragdakis¹, George Karitsis², George Balanoussis³
¹ Department of Informatics, University of Illinois at Urbana-Champaign, Urbana, IL, USA
² Department of Informatics, University of Illinois at Urbana-Champaign, Urbana, IL, USA
³ Department of Informatics, University of Illinois at Urbana-Champaign, Urbana, IL, USA

Introspective Analysis: Context-Sensitivity, Across the Board
YUE LI¹, ANDERS MØLLER², YANNIS SMARAGDAKIS³
¹ Aarhus University, Denmark
² University of Patras, Patras, Greece
³ Department of Informatics, University of Illinois at Urbana-Champaign, Urbana, IL, 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]

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, CINI, China

Scalability-First Pointer Analysis and Self-Tuning Context-Sensitivity
Yue Li¹, Tian Tan¹, and Jingling Xue^{1,2*}
¹ School of Computer Science and Engineering, UNSW Australia
² Advanced Innovation Center for Imaging Technology, CINI, China

Pick Your Contexts Well: Understanding Object-Sensitivity
Yannis Smaragdakis
Department of Computer Science, University of Massachusetts, Amherst, MA, USA
and Department of Informatics, University of Athens, Athens, Greece

Hybrid Context-Sensitivity for Points-To Analysis
George Katsiris¹, Yannis Smaragdakis², and George Balousros³
¹ Department of Informatics, University of Piraeus, Piraeus, Greece
² Department of Informatics, University of Athens, Athens, Greece
³ Department of Informatics, University of Piraeus, Piraeus, Greece

Precision-Guided Context Sensitivity for Pointer Analysis
Tian Tan¹, Anders Møller¹, and Yannis Smaragdakis²
¹ Aarhus University, Denmark
² University of Athens, Greece

Introspective Analysis: Context-Sensitivity, Across the Board
Yannis Smaragdakis¹, George Katsiris², and George Balousros³
¹ Department of Informatics, University of Piraeus, Piraeus, Greece
² Department of Informatics, University of Athens, Athens, Greece
³ Department of Informatics, University of Piraeus, Piraeus, Greece



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



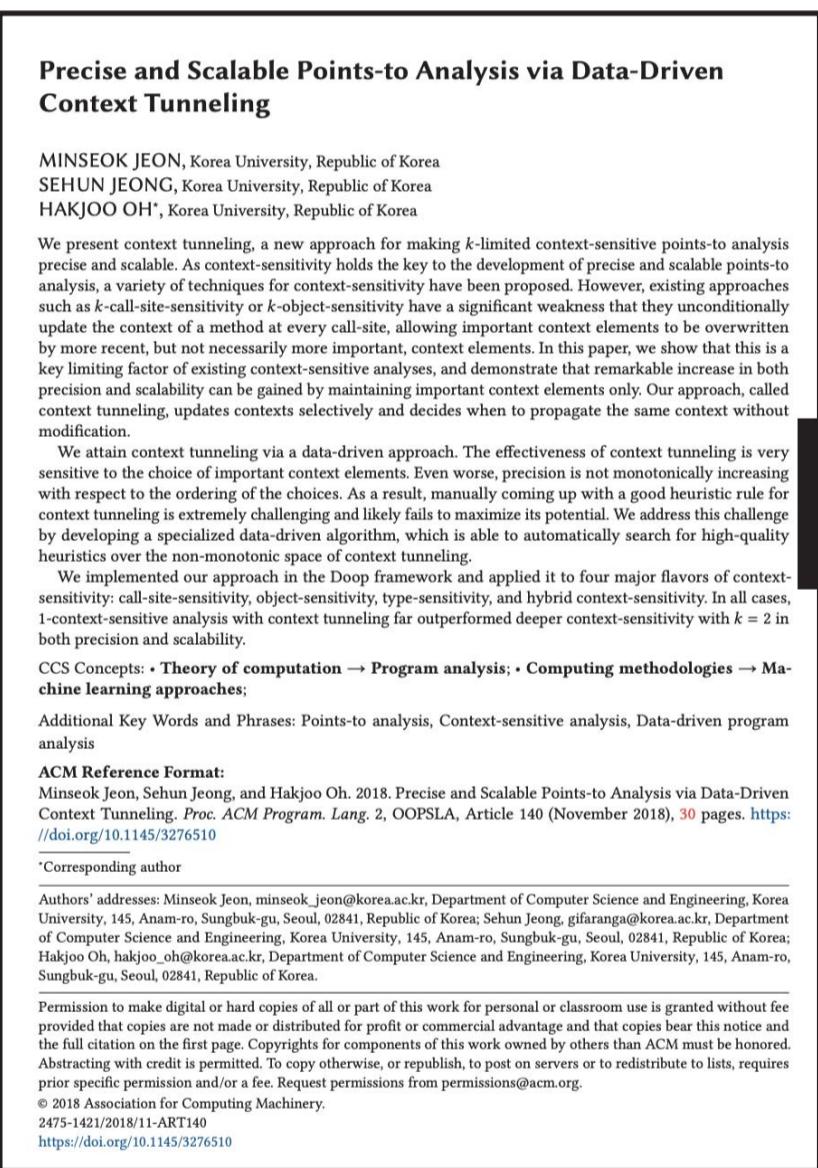
Call-site Sensitivity vs Object Sensitivity

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

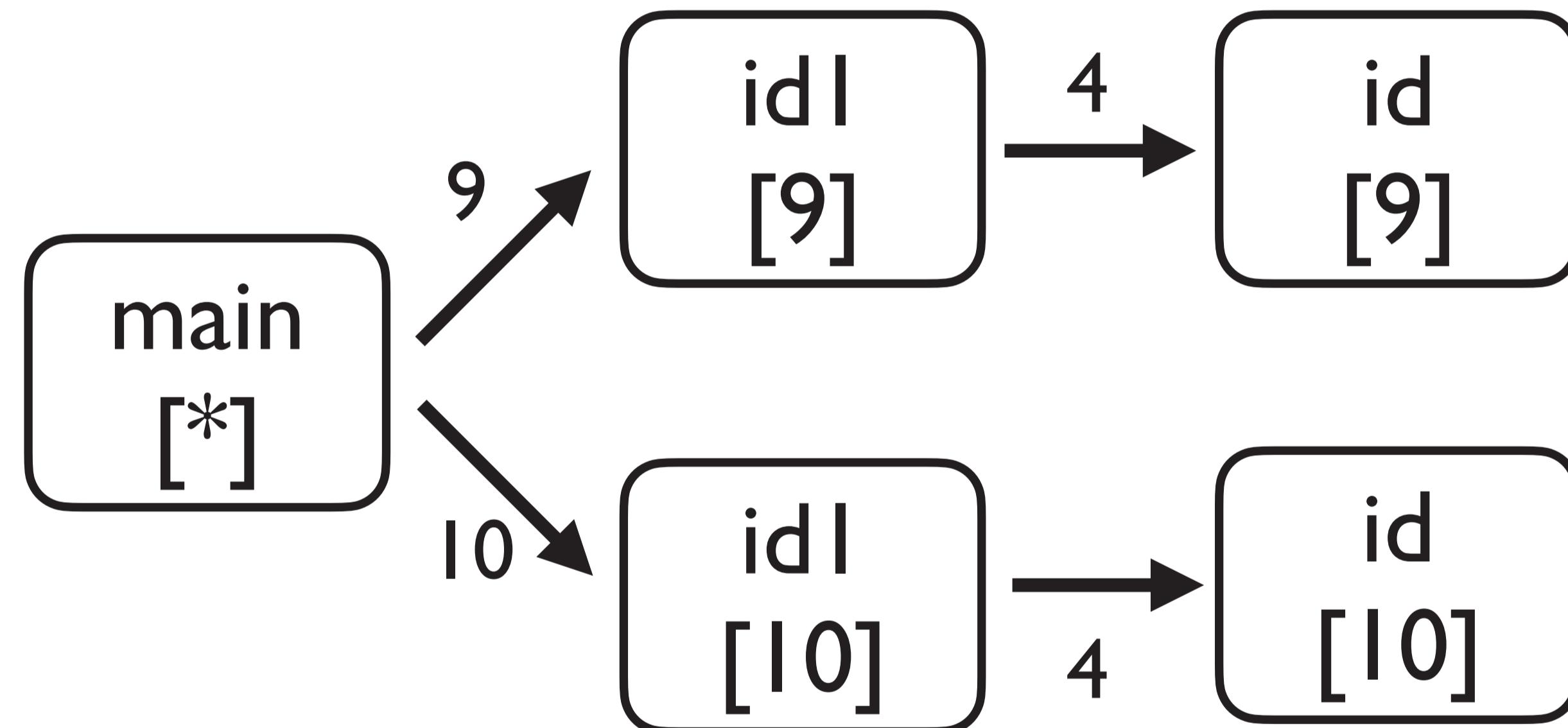
Jeon et al. [2018]



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();  
8:     c2 = new C();  
9:     a = (A) c1.id1(new A());  
10:    b = (B) c2.id1(new B());  
11:  }
```

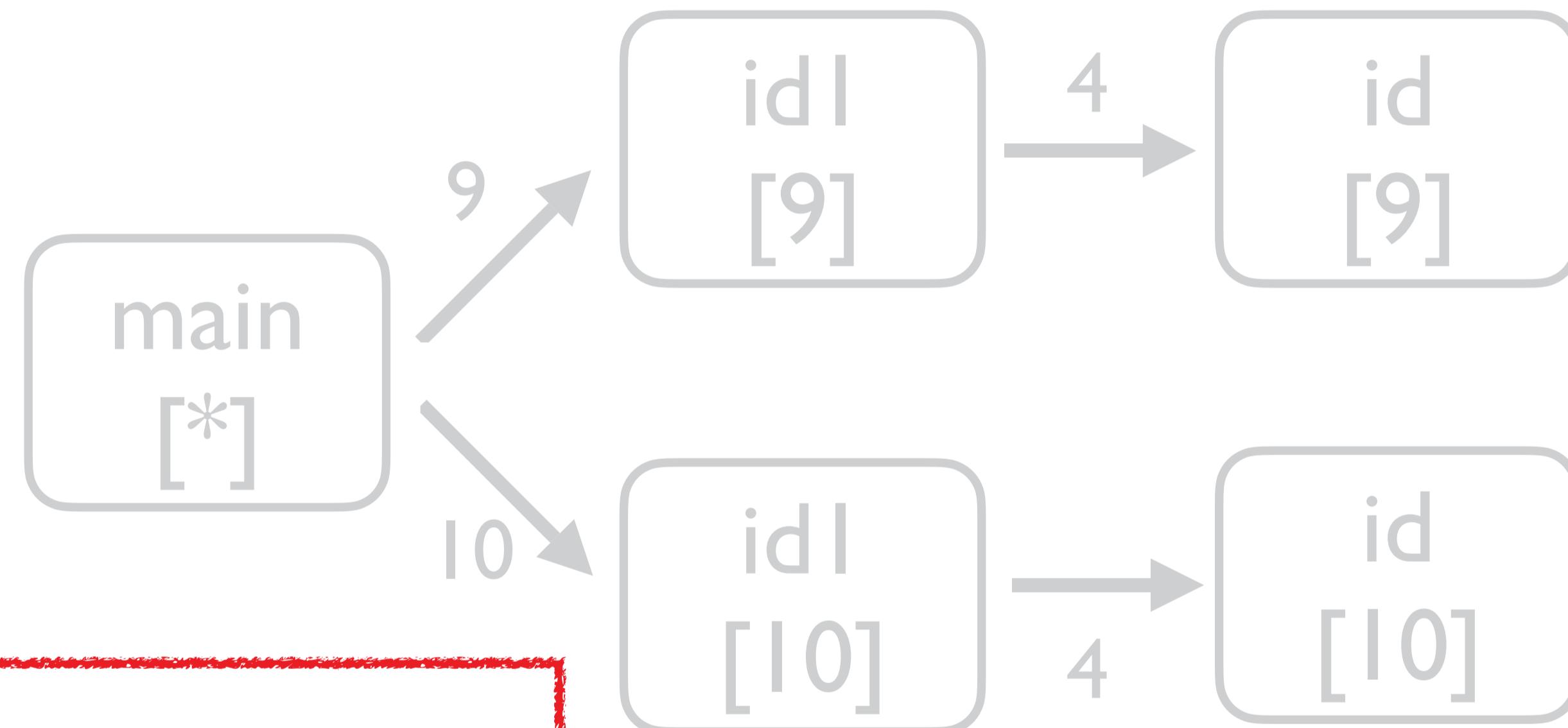


I-CFA with context tunneling
($T = \{4\}$)

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:   id1(v){  
4:     return id0(v);  
5:   }  
6:   main(){  
7:     c1 = new C();  
8:   }  
9: }
```

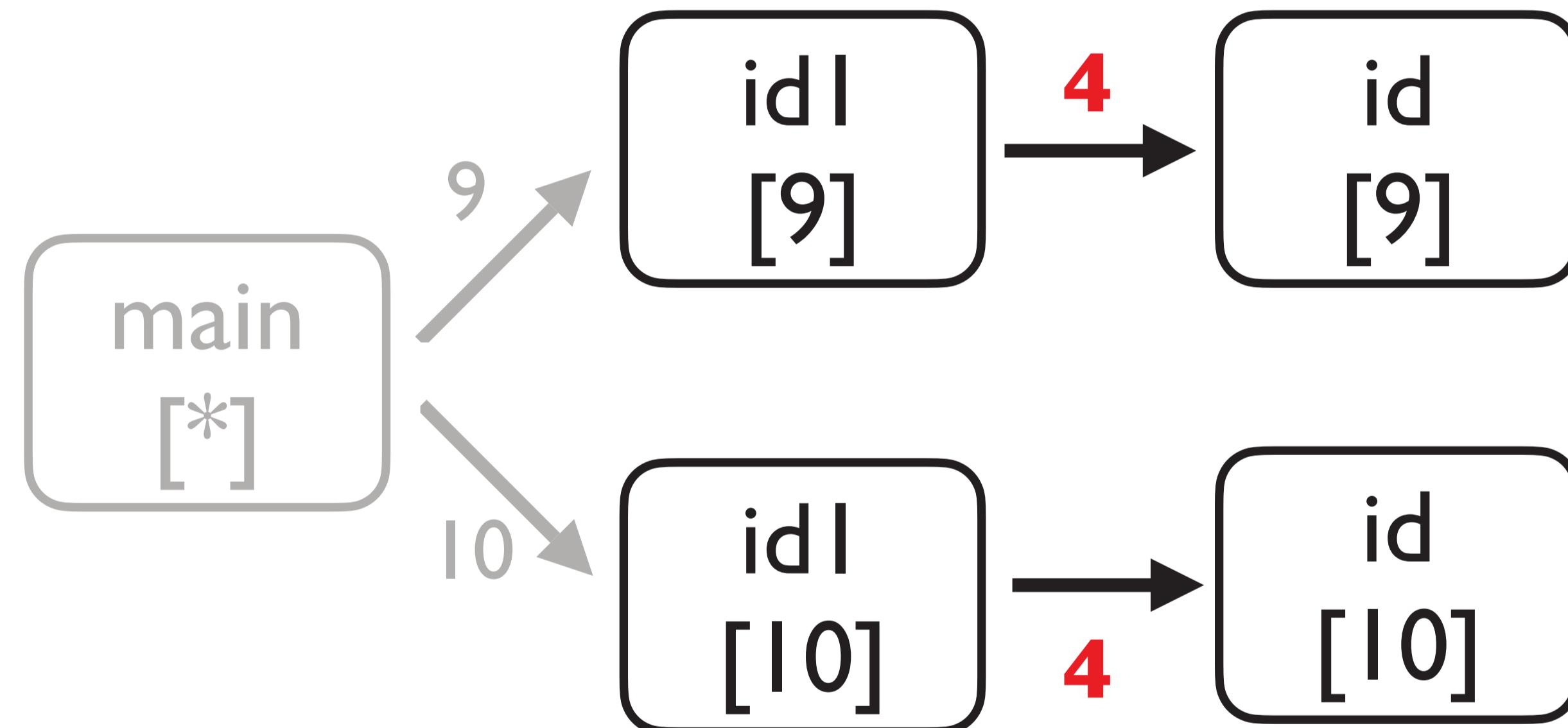


Tunneling abstraction:
Determines where to apply context tunneling
with context tunneling
($T = \{4\}$)

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:   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  
}
```



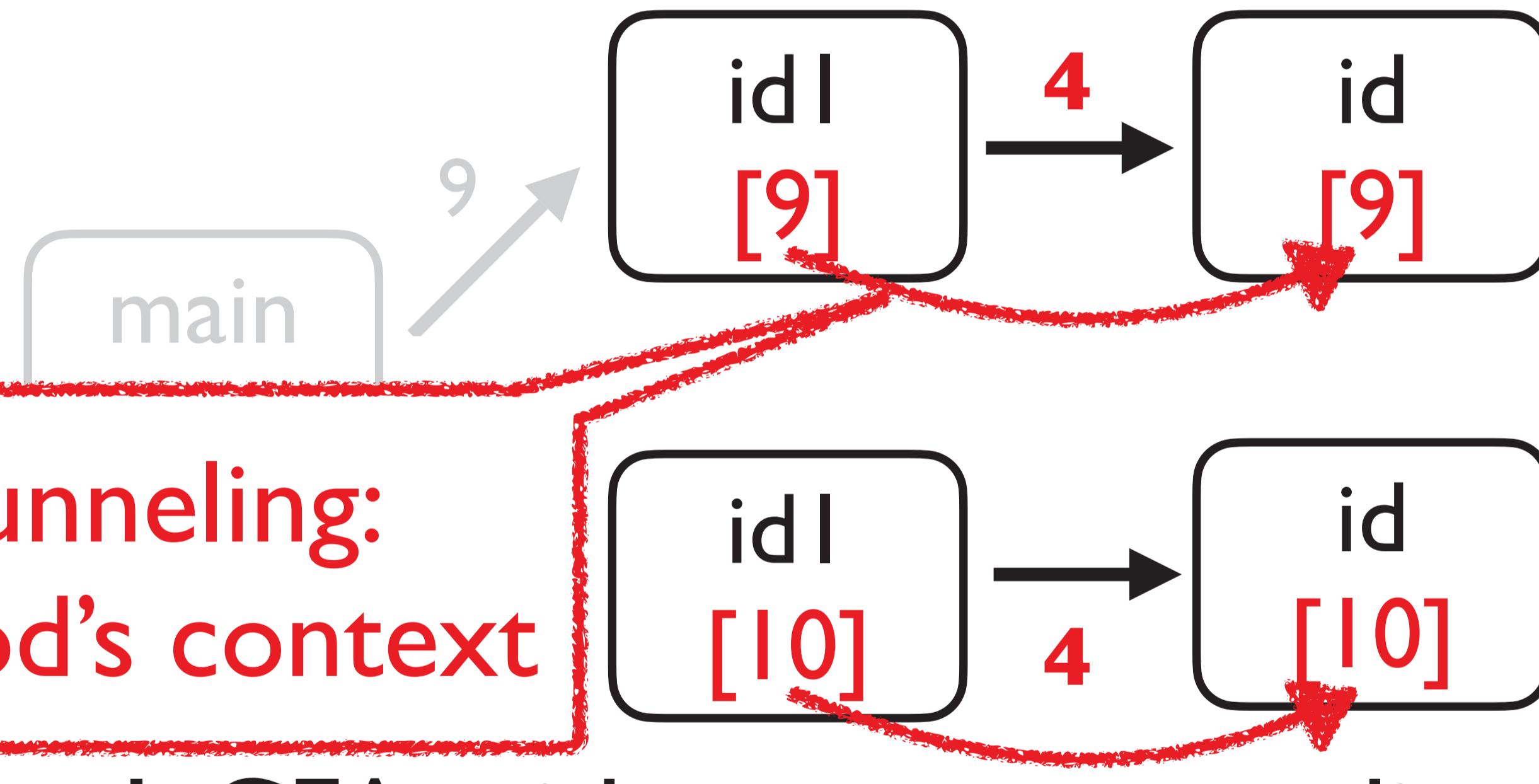
I-CFA with context tunneling
($T = \{4\}$)

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

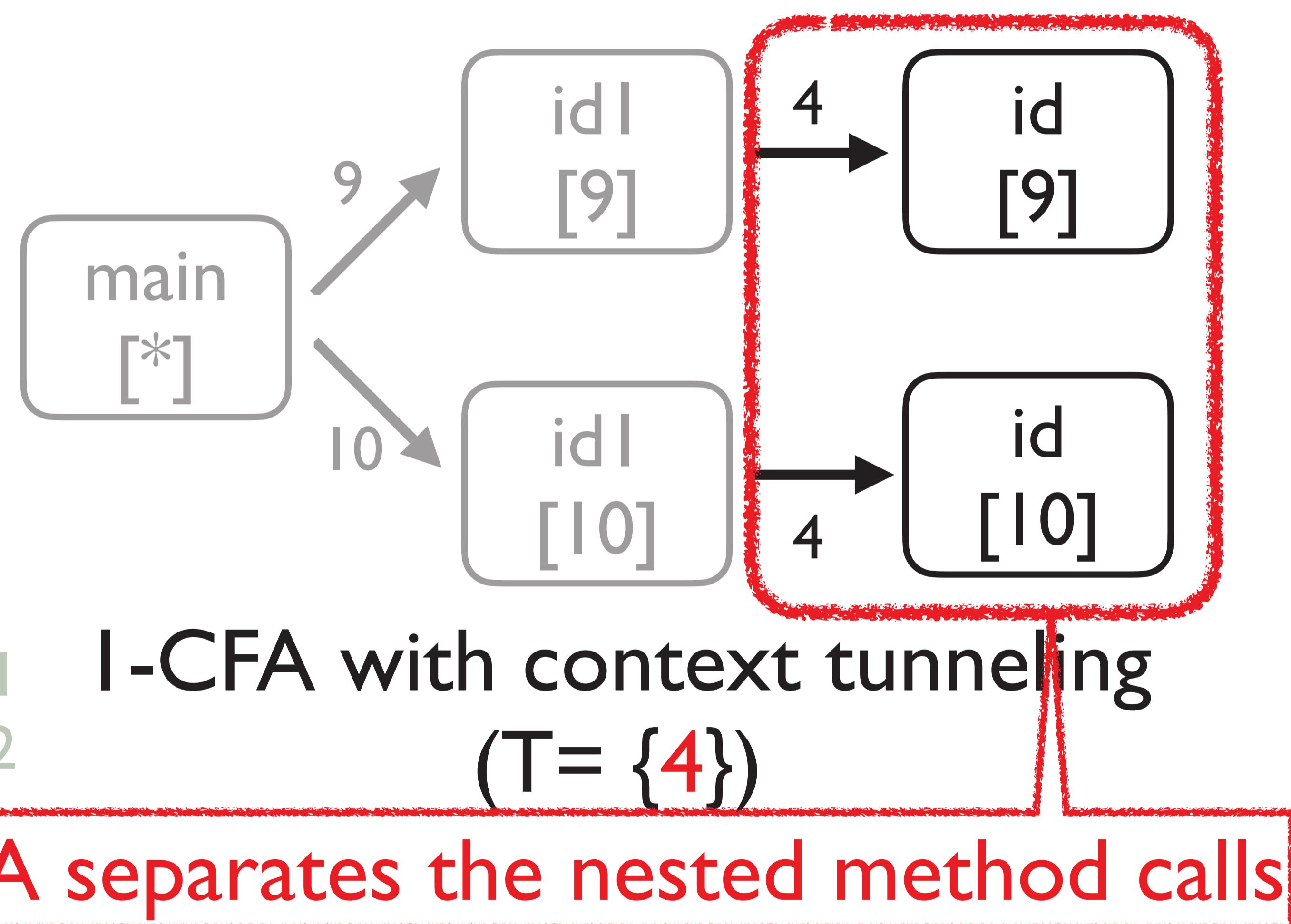


I-CFA with context tunneling
(T= {4})

Call-site Sensitivity vs Object Sensitivity

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

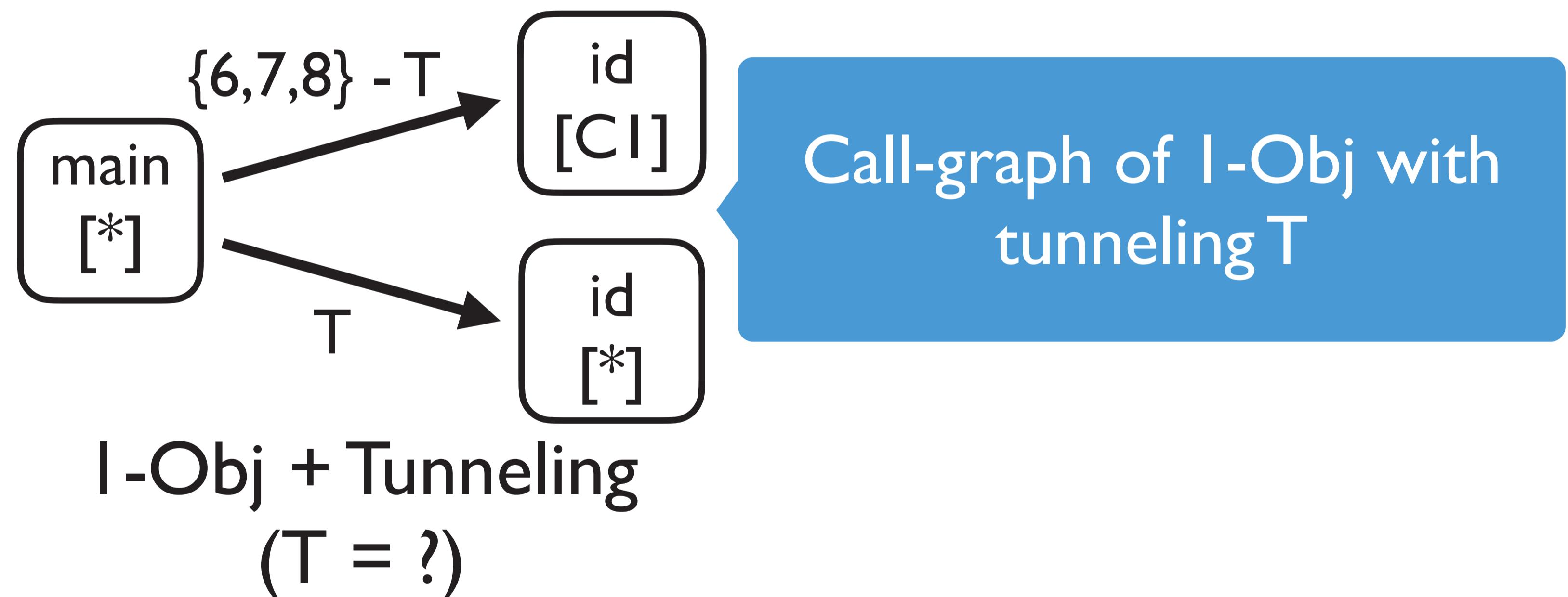
```
0: class C{  
1:   id(v){  
2:     return v;}  
3:   id1(v){  
4:     return id0(v);}  
5: }  
6: main(){  
7:   c1 = new C(); //C1  
8:   c2 = new C(); //C2  
9:   a = (A) c1.id1(new A()); //query1  
10:  b = (B) c2.id1(new B()); //query2  
11: }
```



Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its limitation

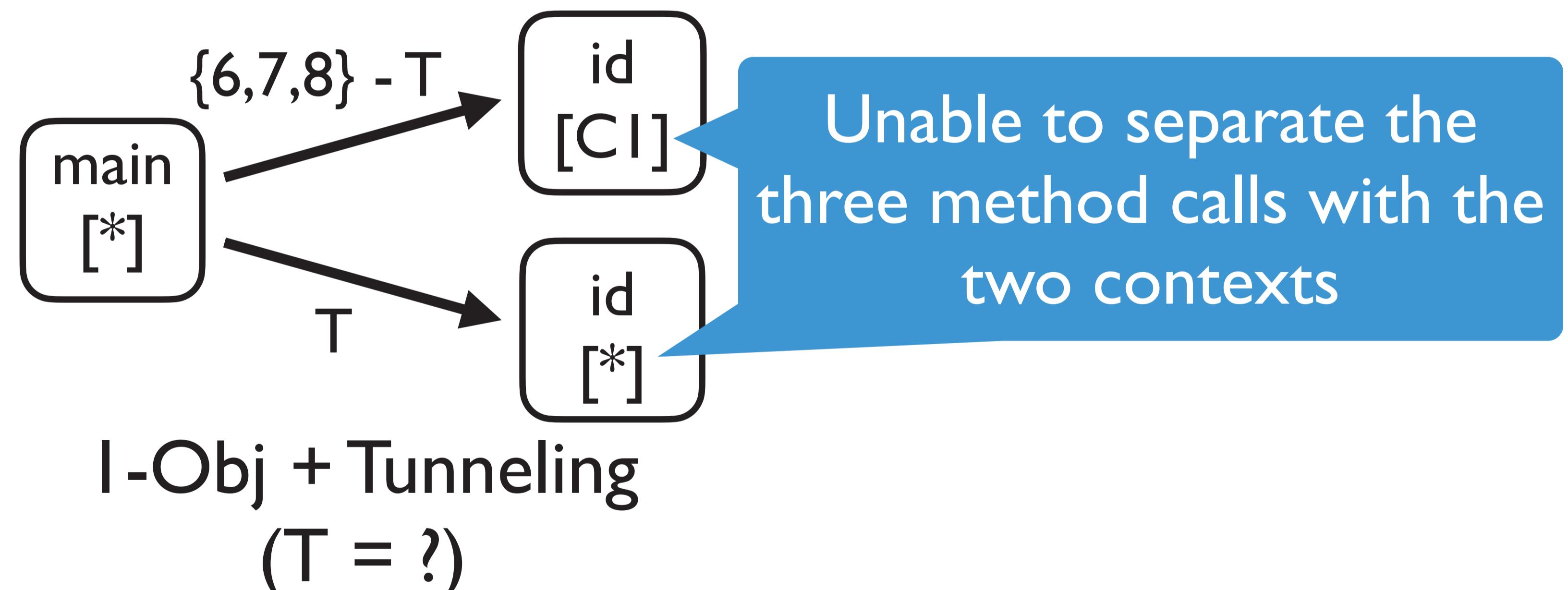
```
0: class C{  
1:   id(v){  
2:     return v;  
3:   }  
4: main(){  
5:   cl = new C(); //CI  
6:   a = (A) cl.id(new A());  
7:   b = (B) cl.id(new B());  
8:   c = (C) cl.id(new C());  
9: }
```



Call-site Sensitivity vs Object Sensitivity

- Object sensitivity still suffers from its **limitation**

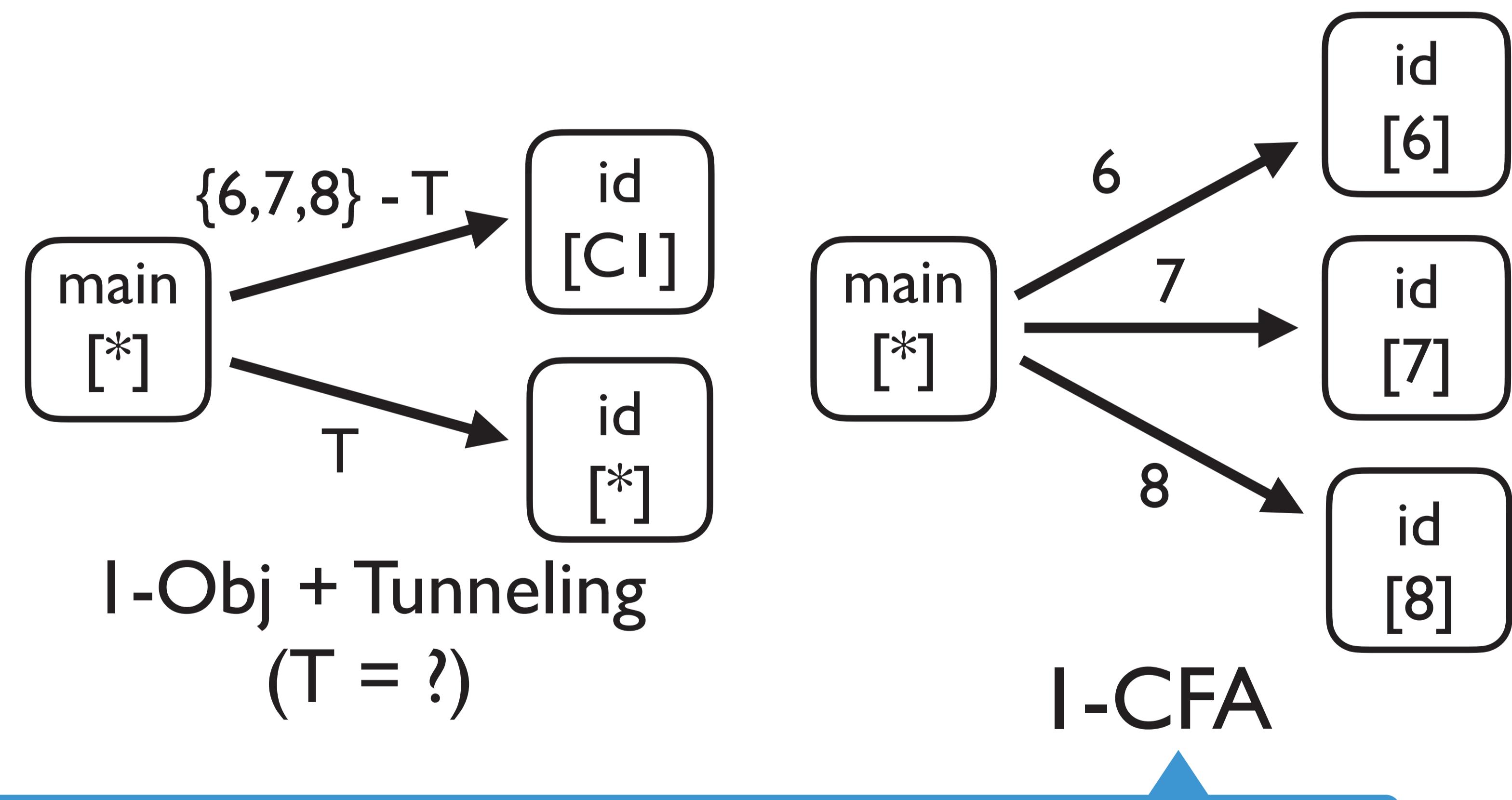
```
0: class C{  
1:   id(v){  
2:     return v;  
3: }  
4: main(){  
5:   cl = new C(); //CI  
6:   a = (A) cl.id(new A());  
7:   b = (B) cl.id(new B());  
8:   c = (C) cl.id(new C());  
9: }
```



Call-site Sensitivity 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(); //Cl  
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

```
0: c
1:
2: }
3: m
4: n
5:
6: a
7: b
8: c
9: }
```

When context tunneling is included

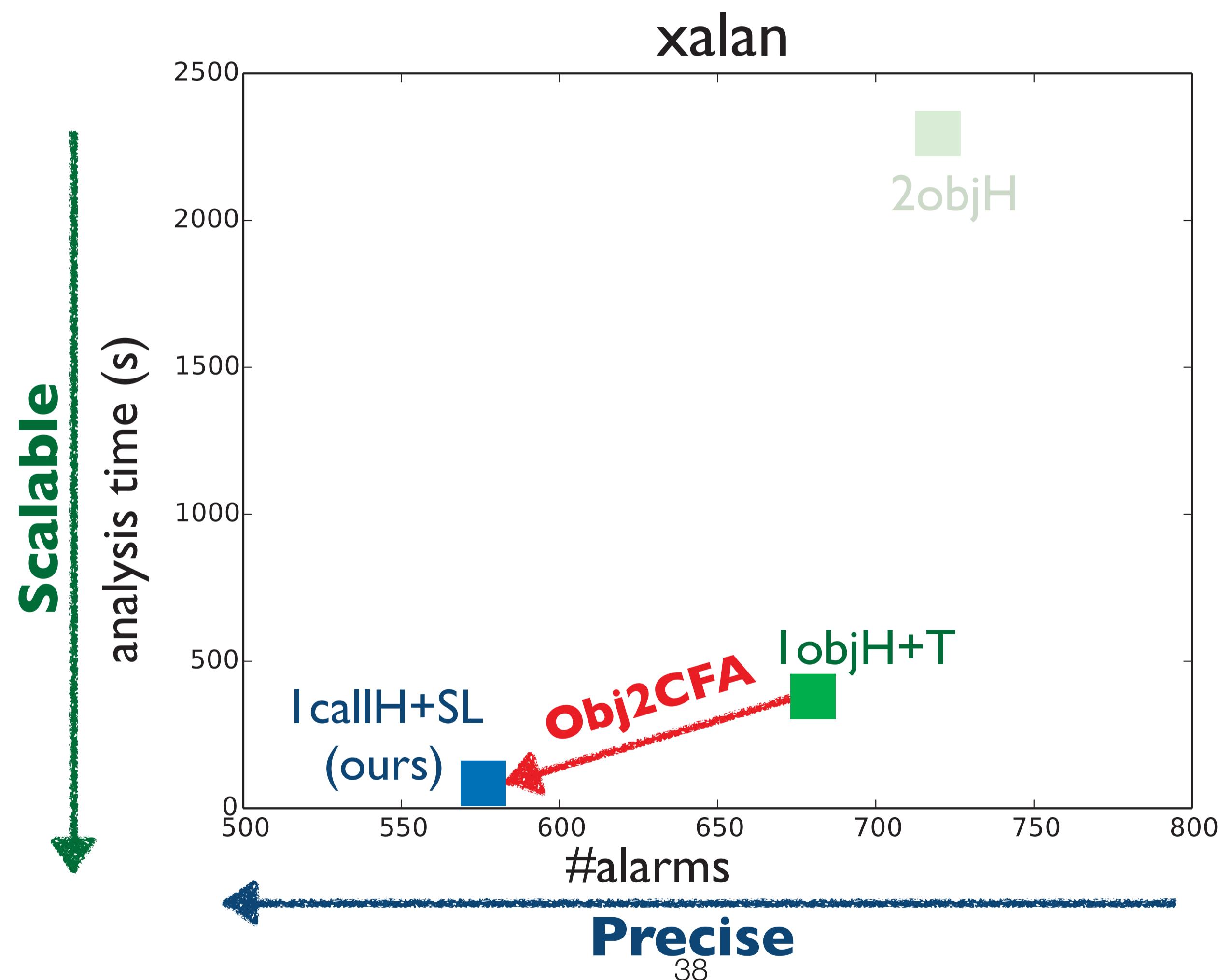
- Limitation of call-site sensitivity is **removed**
- Limitation of object sensitivity is **not removed**

## Our claim

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

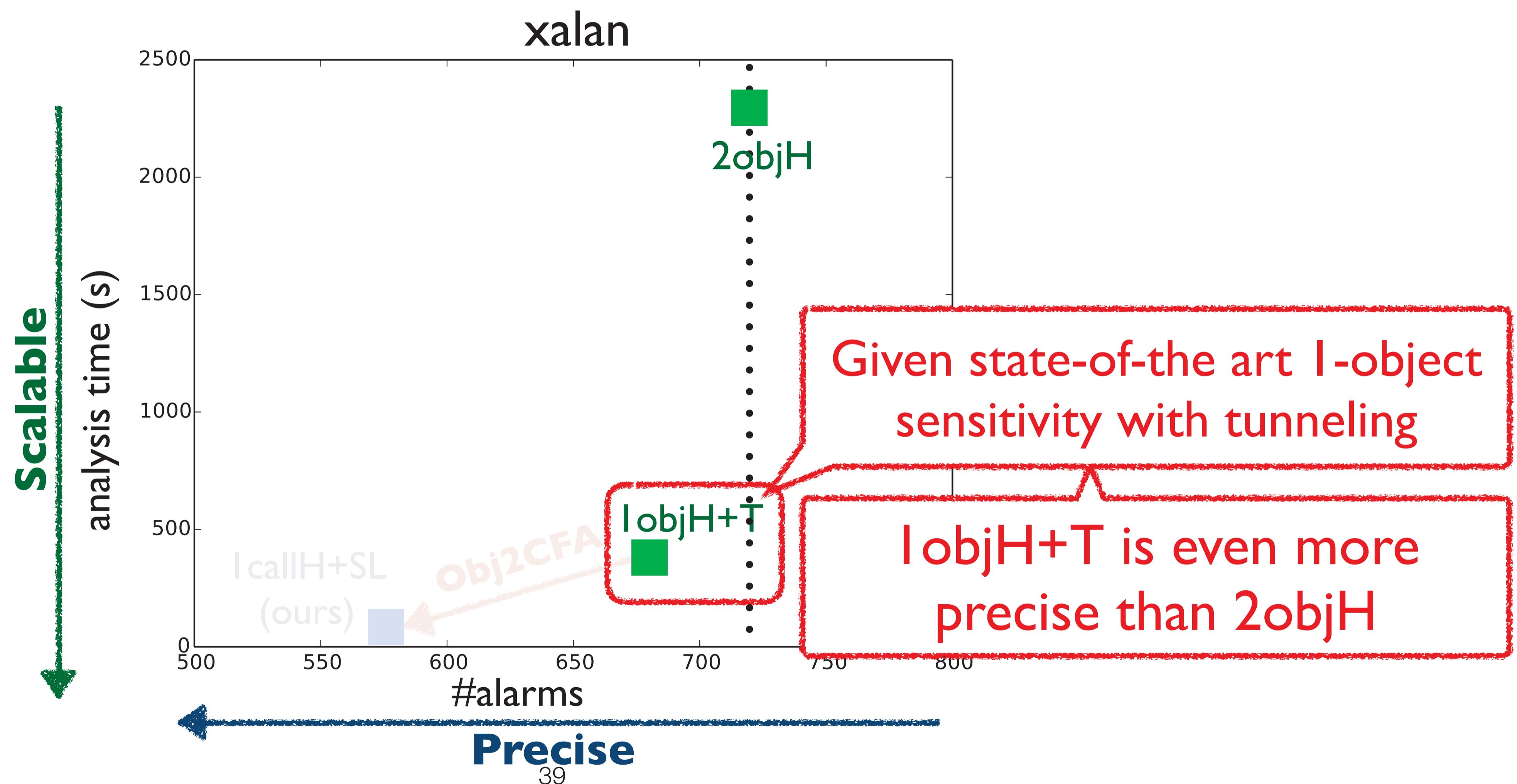
# Our Technique : **Obj2CFA**

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



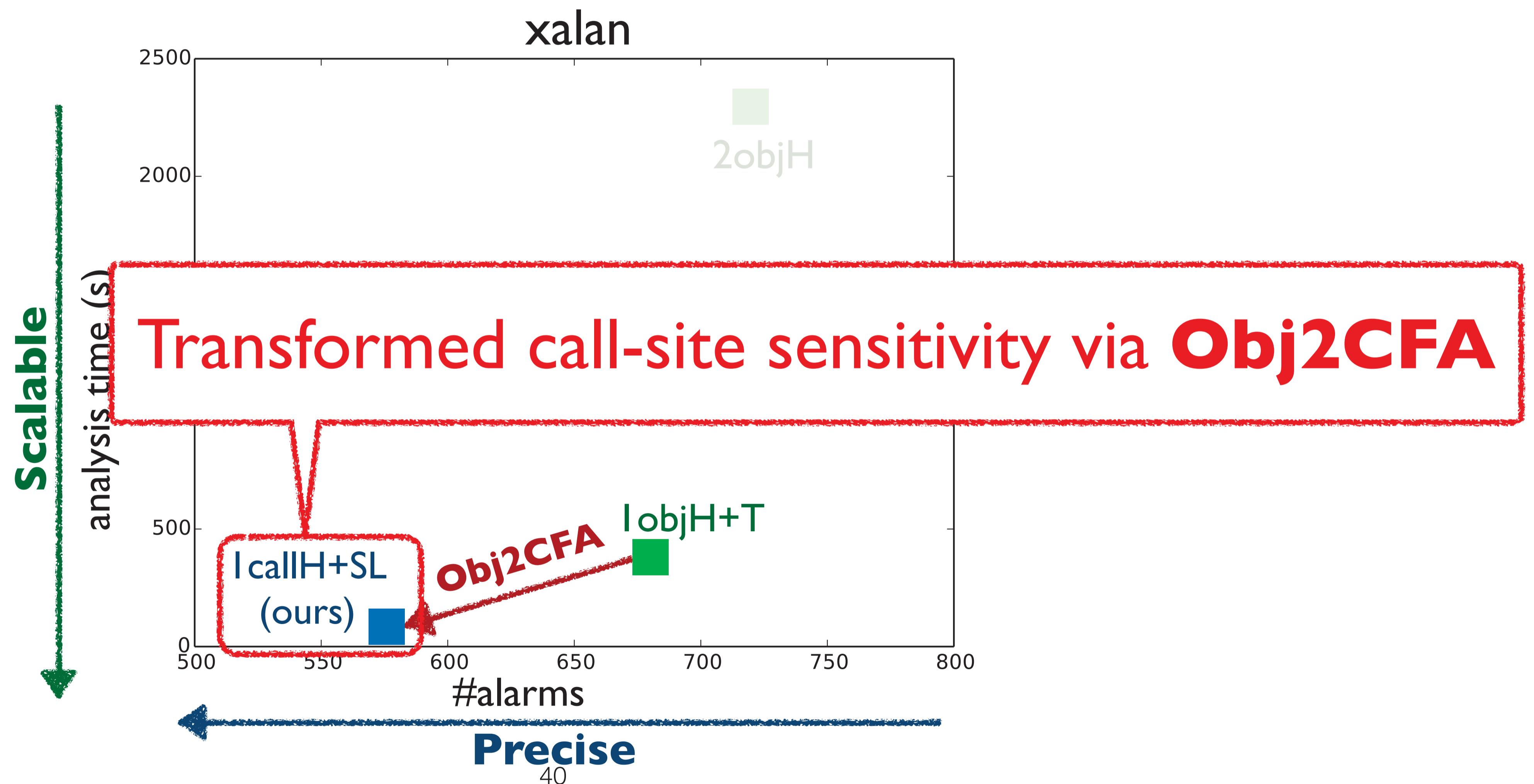
# Our Technique : **Obj2CFA**

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



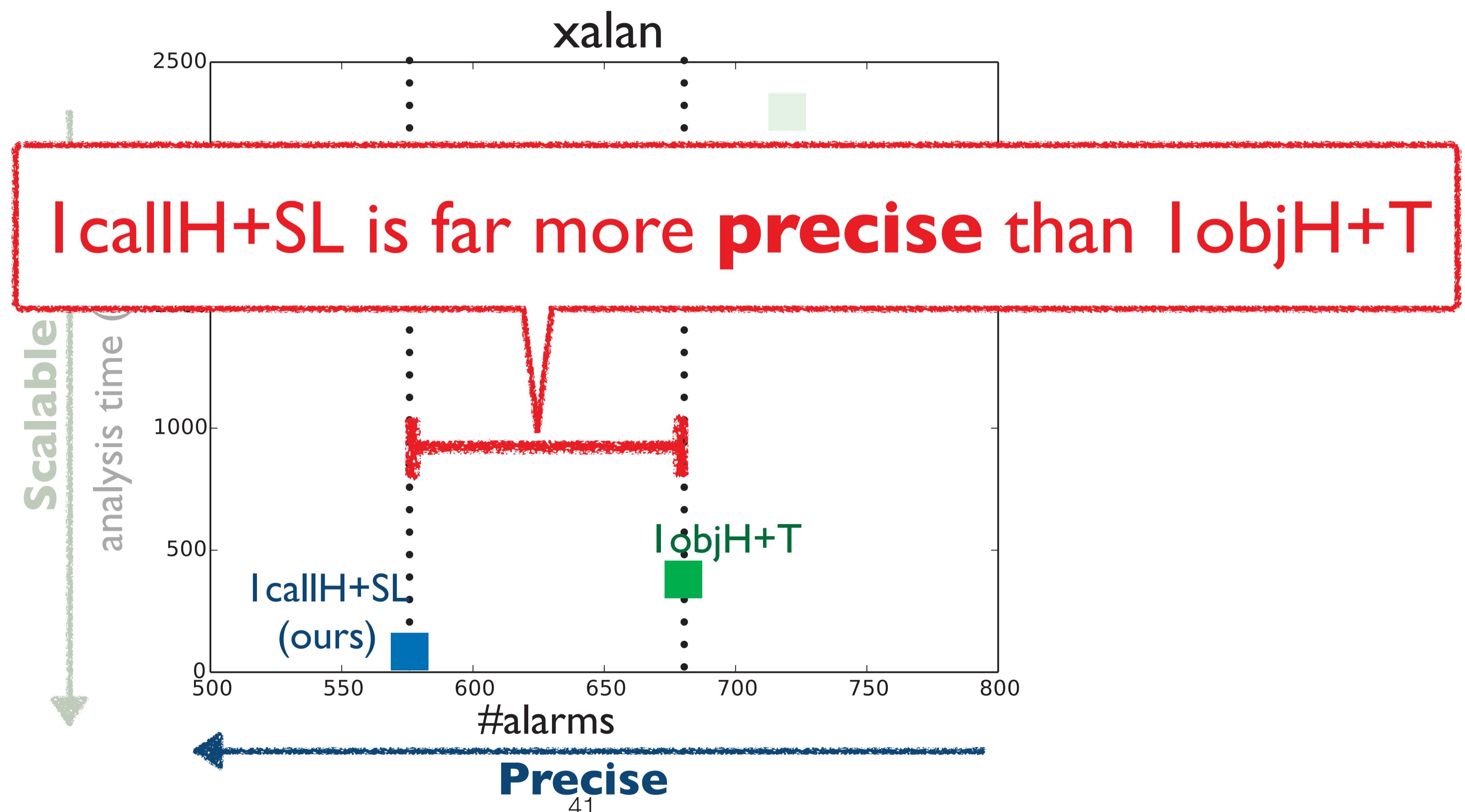
# Our Technique : **Obj2CFA**

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



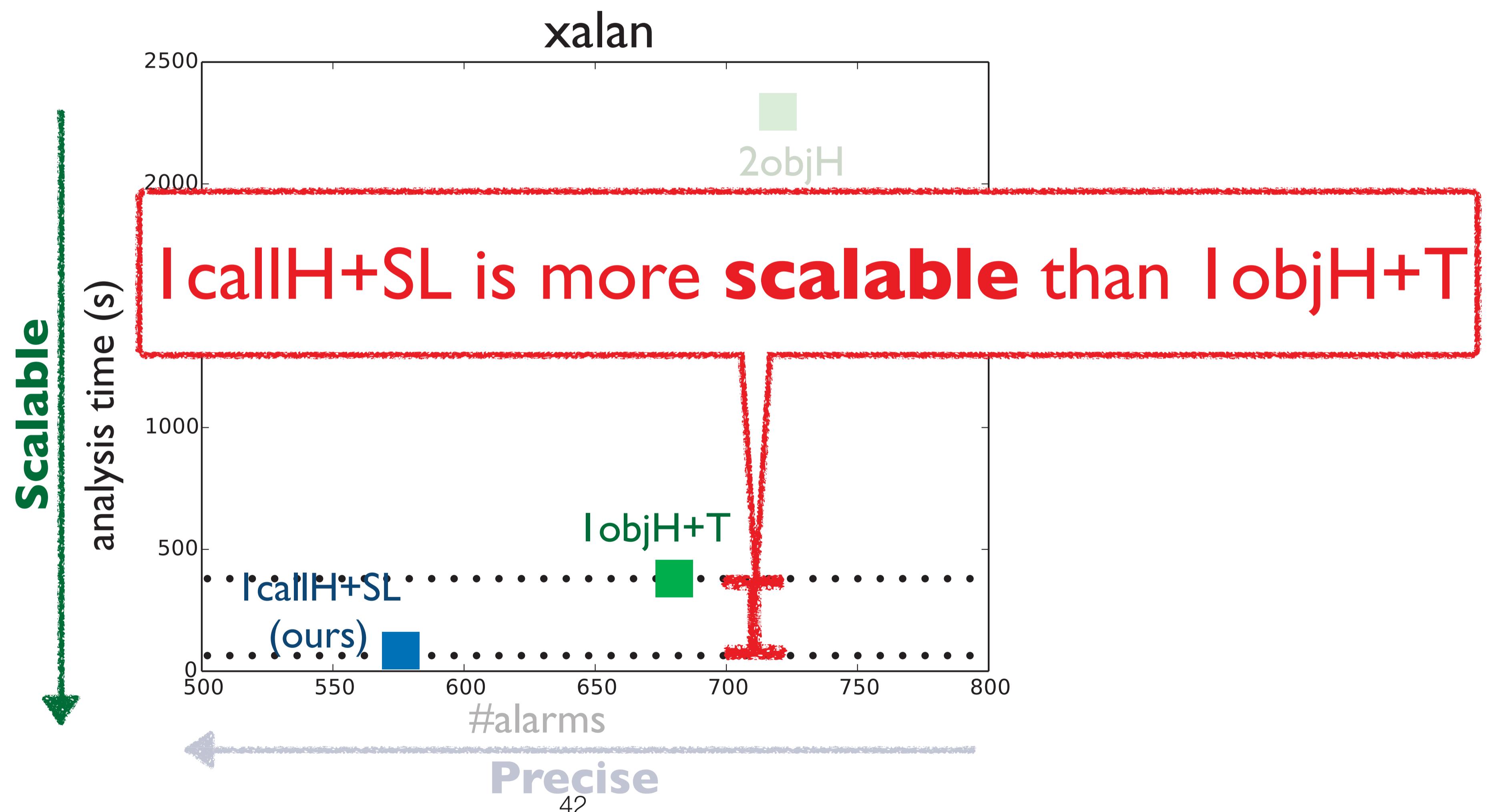
# Our Technique : **Obj2CFA**

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



# Our Technique : **Obj2CFA**

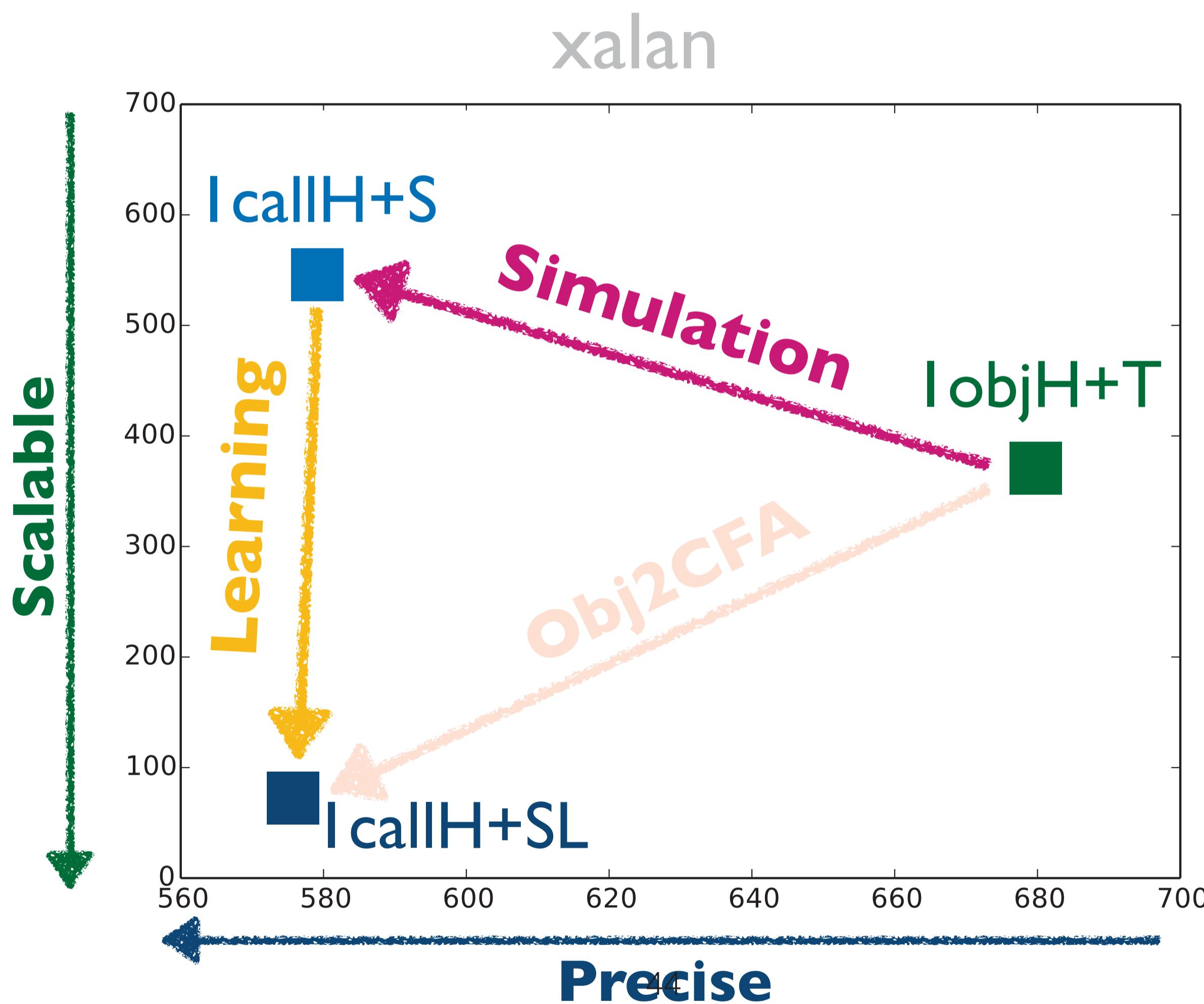
- **Obj2CFA** transforms a given **object sensitivity** into a more precise **CFA**



# **Detail of Obj2CFA**

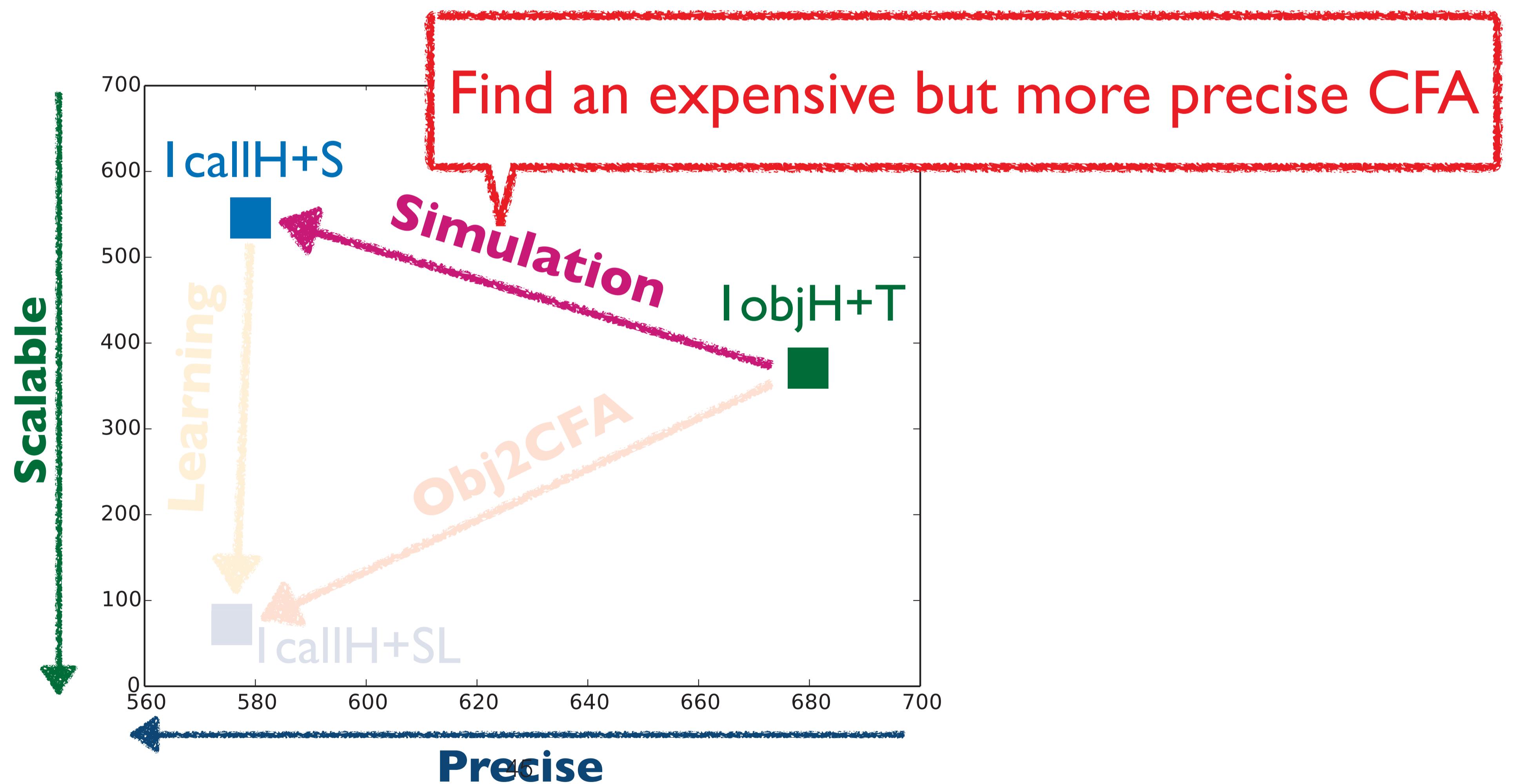
# Our Technique : **Obj2CFA**

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



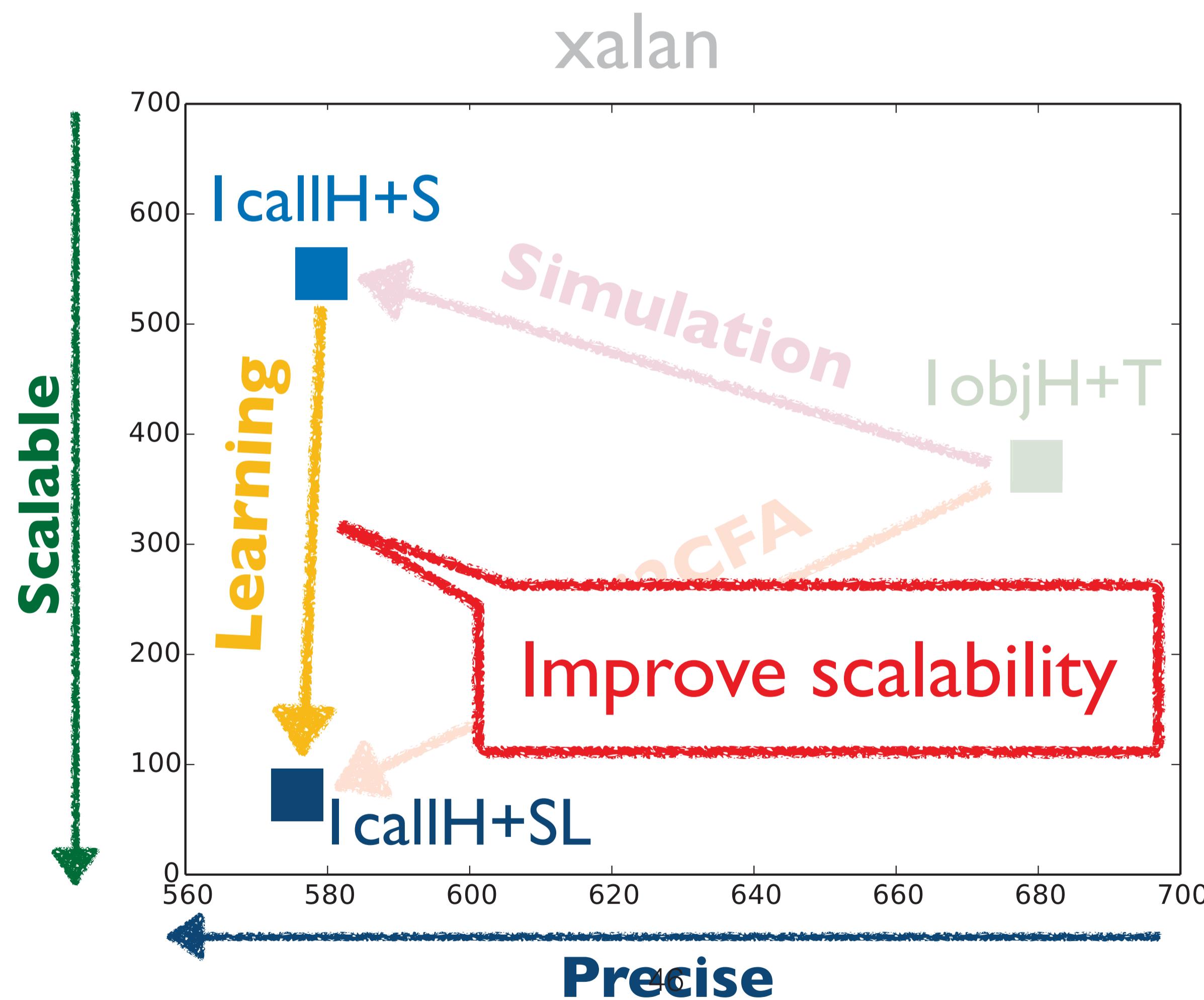
# Our Technique : **Obj2CFA**

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



# Our Technique : **Obj2CFA**

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



# Technique I: Simulation

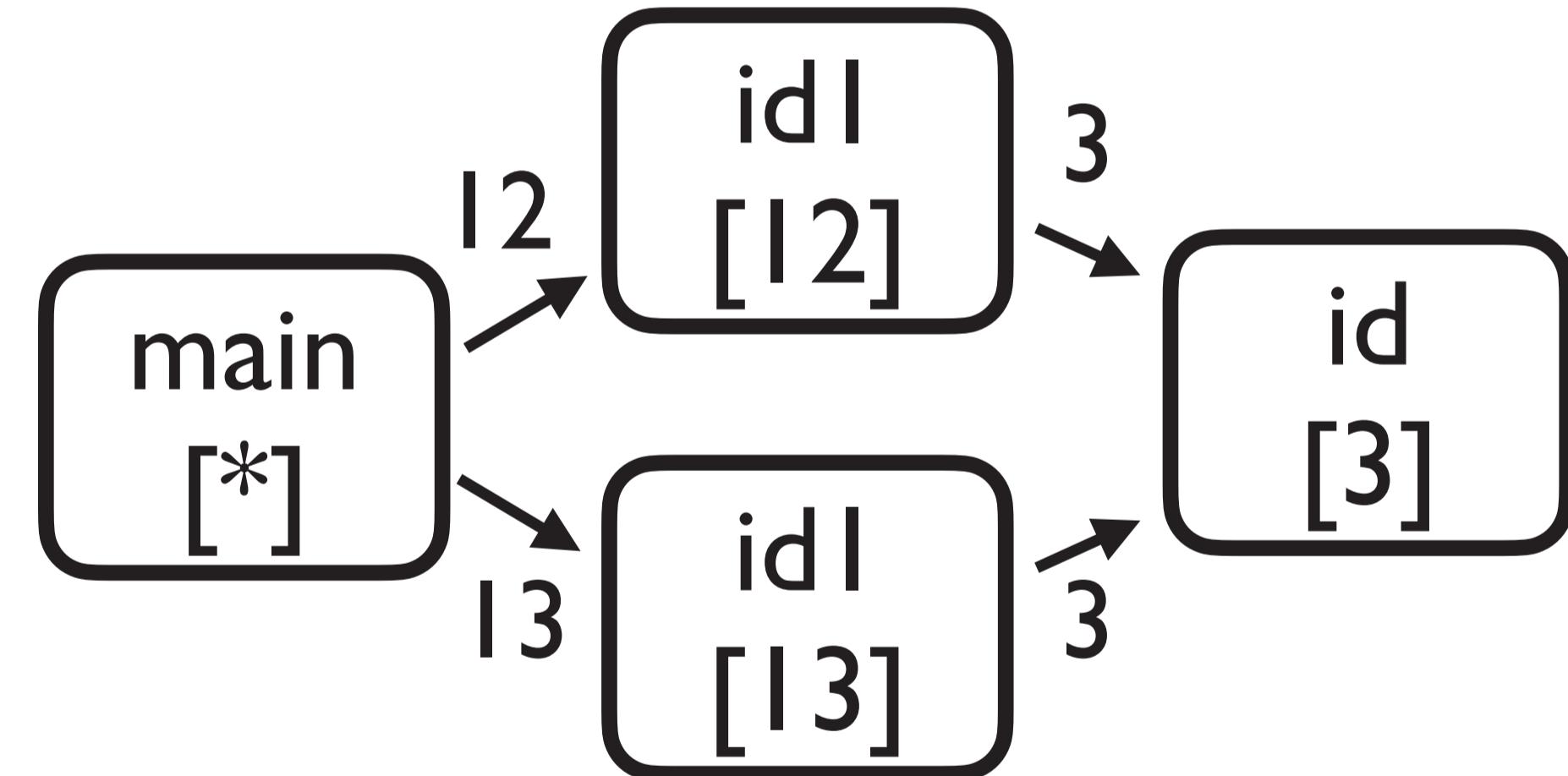
- Running example to illustrate Simulation

```
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);} → 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
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);}
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: }
```



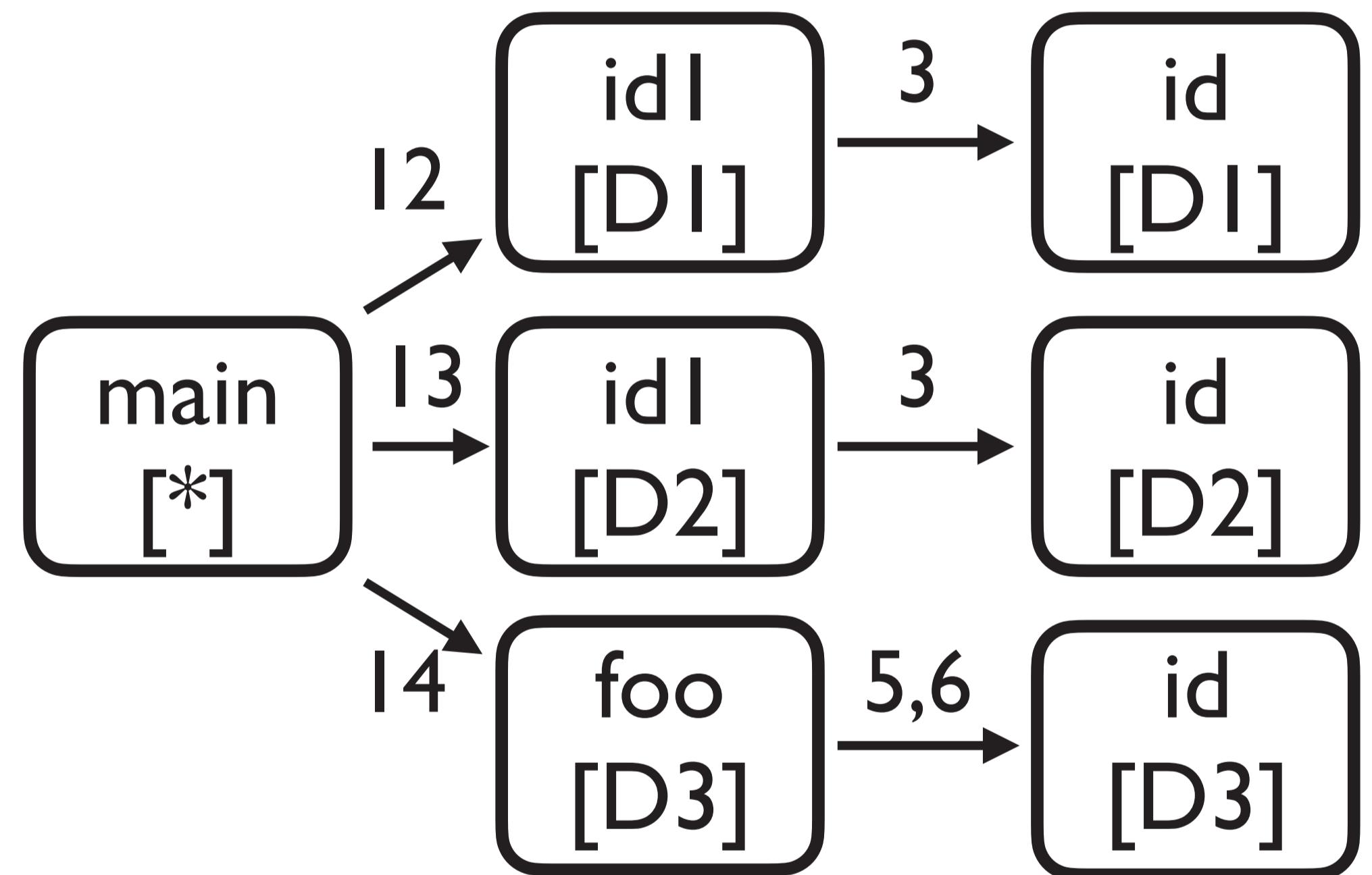
Limitation of object sensitivity



# Technique I: Simulation

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

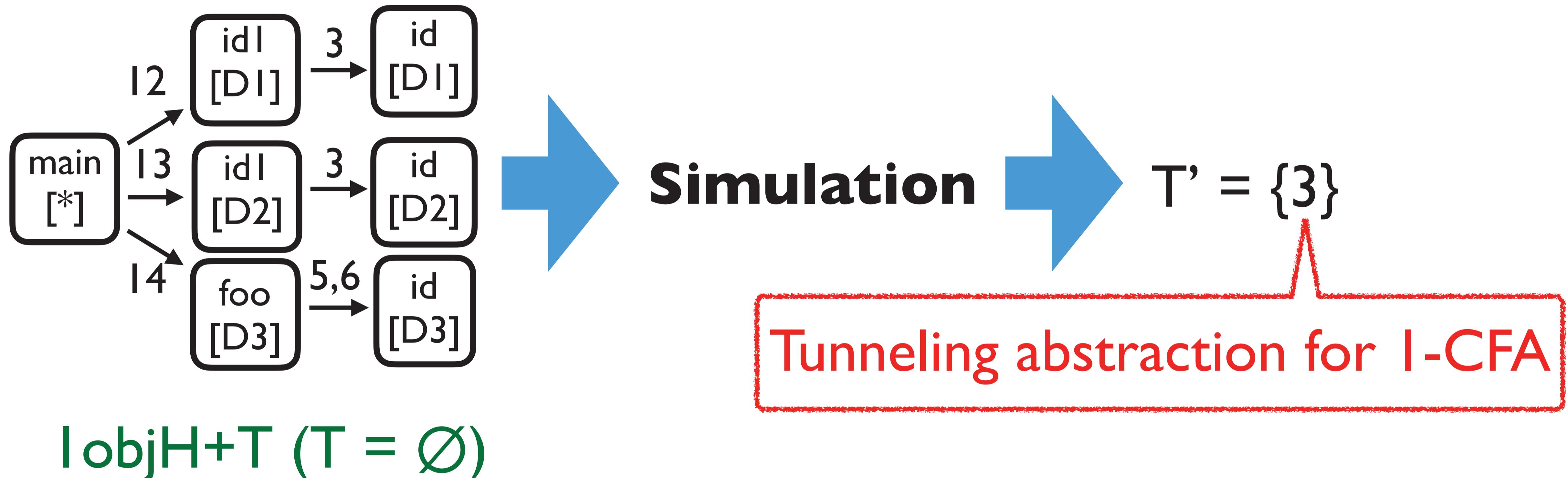
```
I: class C{
2: id(v){return v;}
3: idI(v){return id(v);}
4: foo(){
5: A a = (A) this.id(new A());}//query1
6: B b = (B) this.id(new B());}//query2
7: }
8: main(){
9: c1 = new C();//C1
10: c2 = new C();//C2
11: c3 = new C();//C3
12: A a = (A) c1.idI(new A());//query3
13: B b = (B) c2.idI(new B());//query4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



**lobjH+T** ( $T = \emptyset$ )

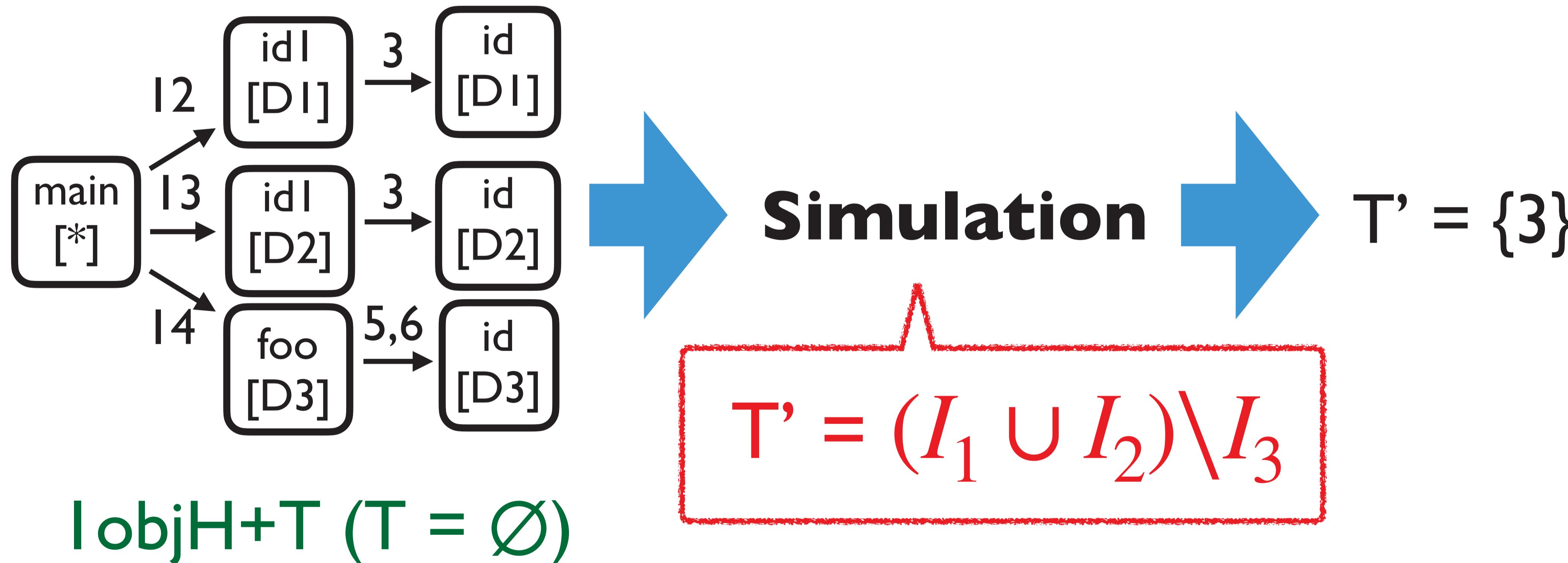
# Technique I: Simulation

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



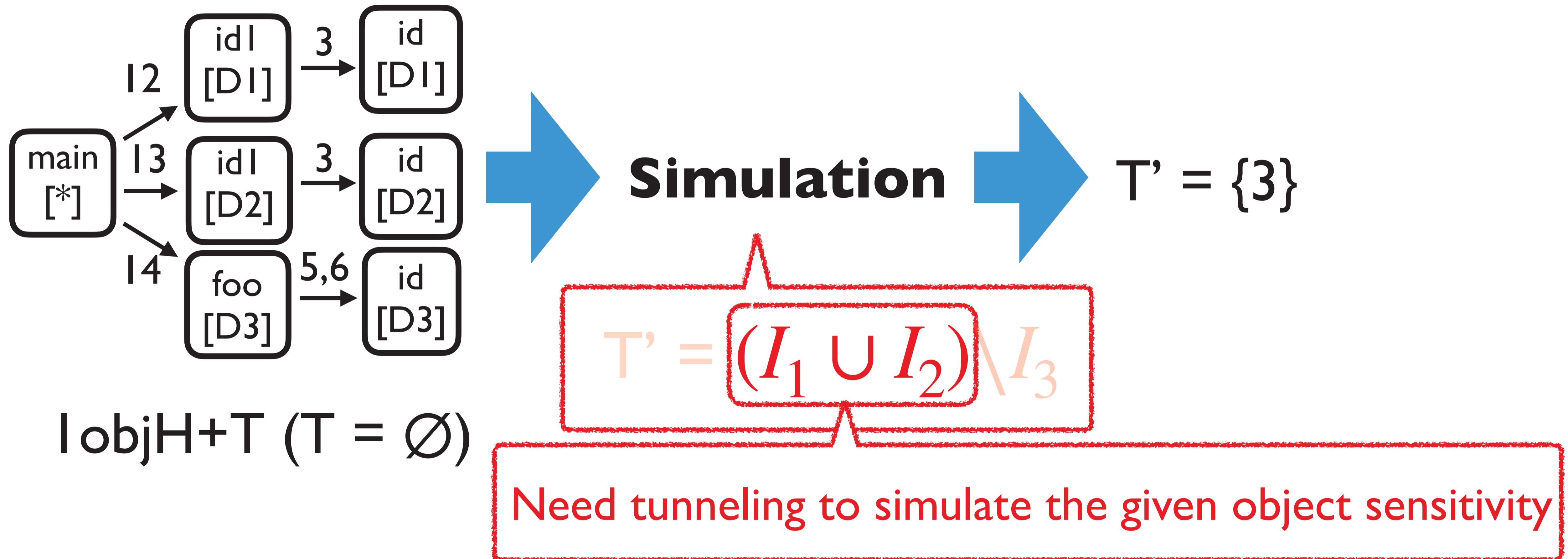
# Technique I: Simulation

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



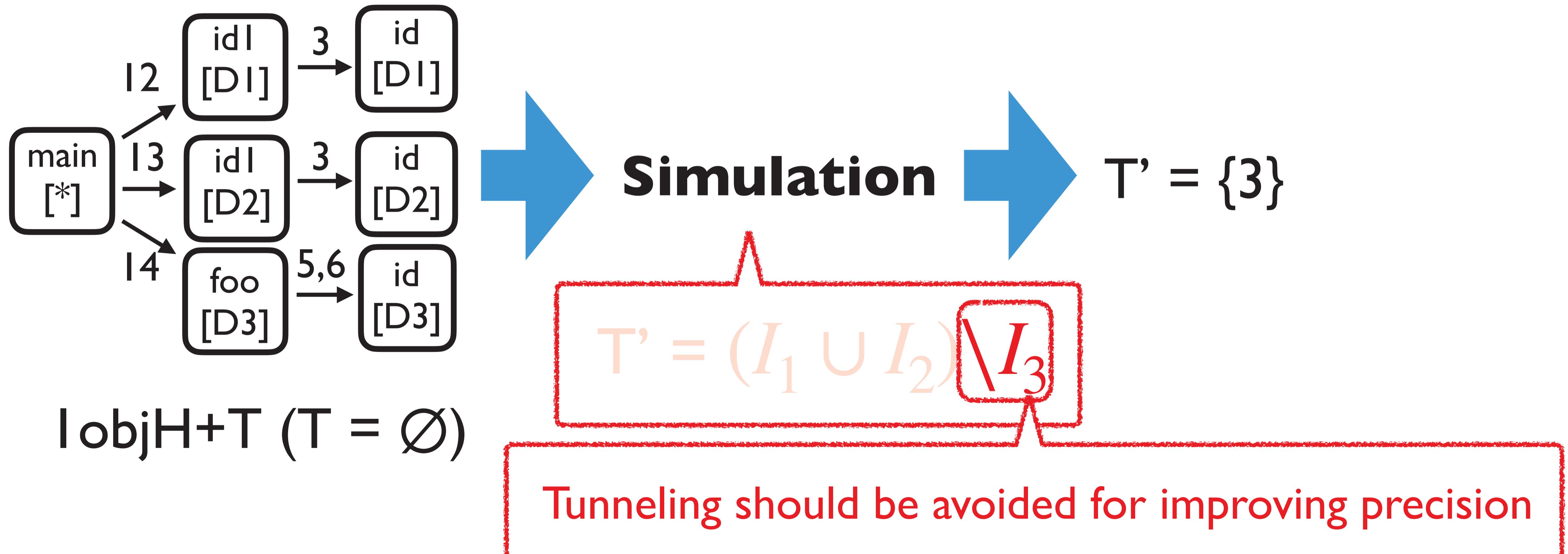
# Technique I: Simulation

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



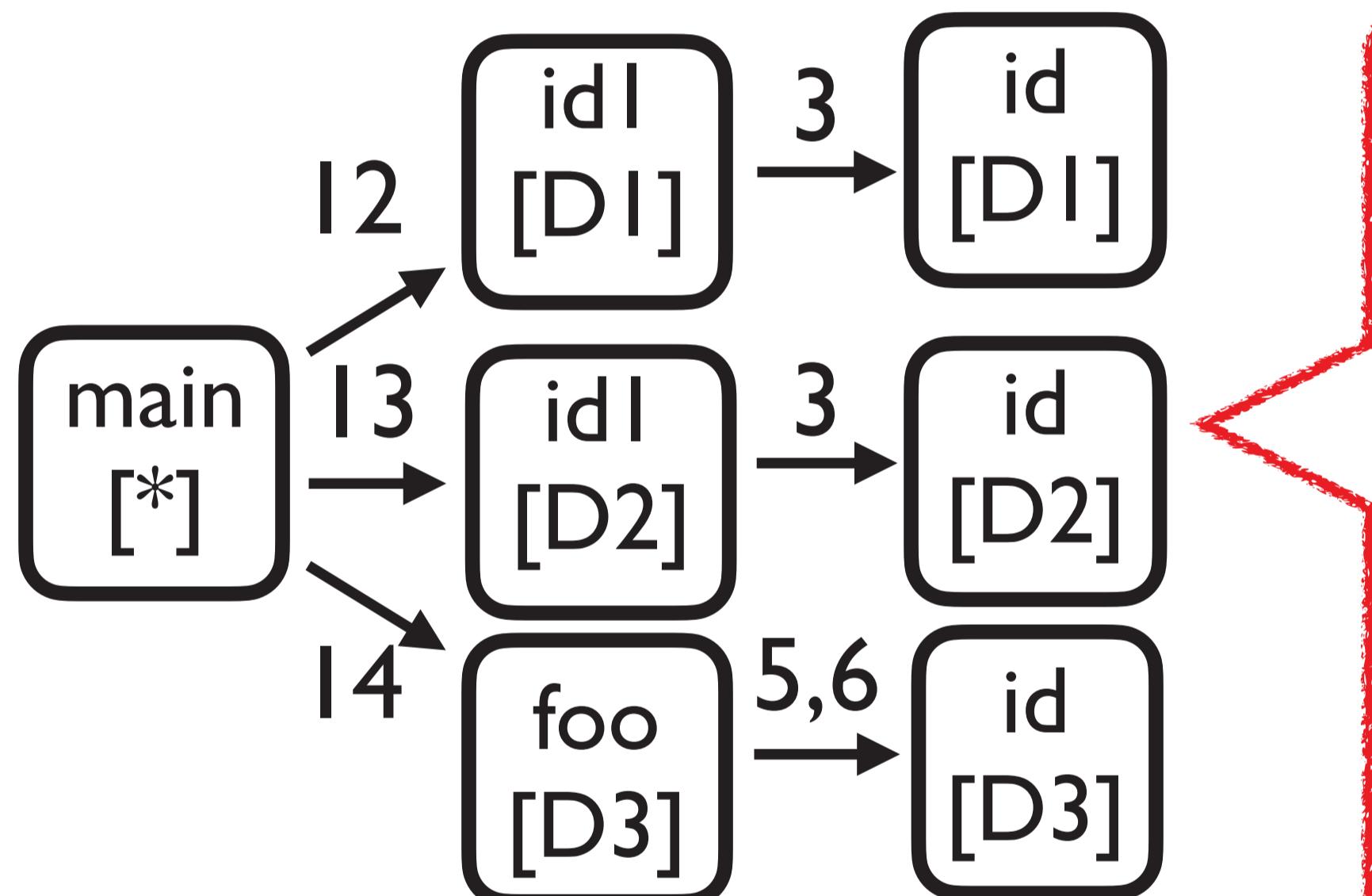
# Technique I: Simulation

- **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



## Intuition of Simulation

Suppose the call-graph is produced from  
1-CFA +  $T'$  and infer the  $T'$

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

$I_{call}H+T'$

What is  $T'$ ?

# Intuition Behind Simulation ( $I_1 \cup I_2$ )

- Suppose  $i \in I_1$  and  $i \in I_2$ . Then  $i \in I_1 \cup I_2$ .  $i \in I_1 \cup I_2$  is

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



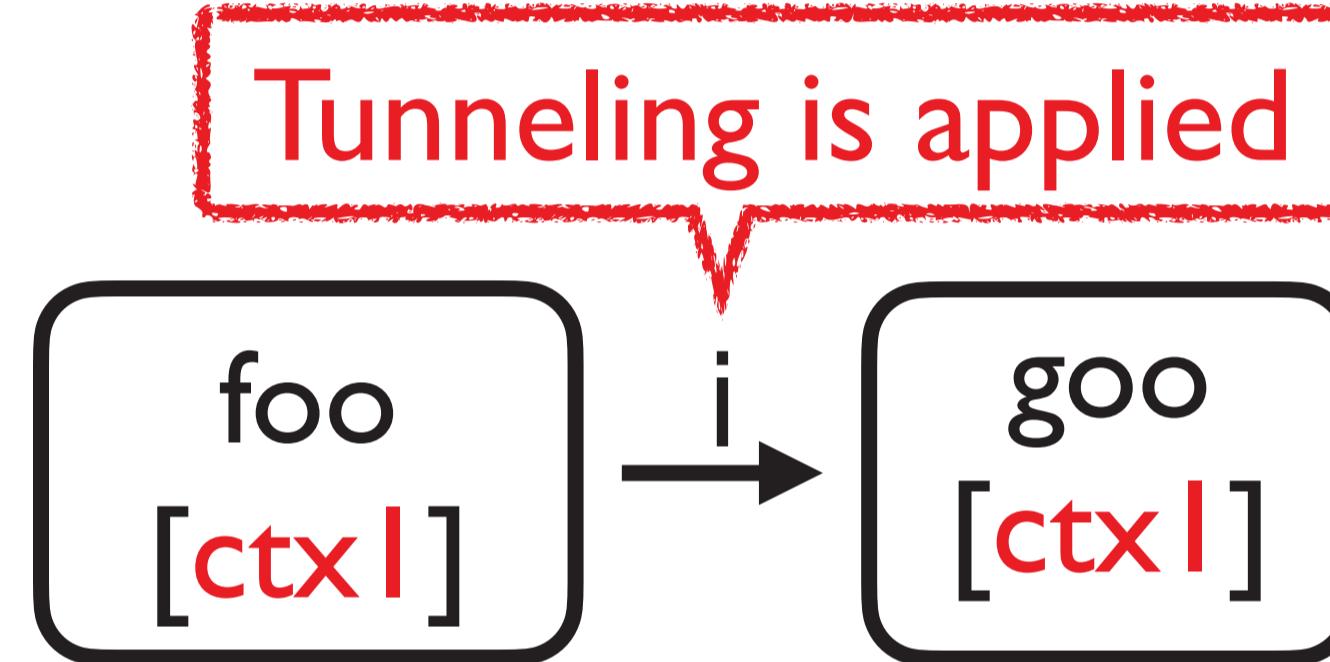
We track the two properties to find the  $T'$

main  
[\*]

# Intuition Behind Simulation ( $I_1 \cup I_2$ )

- Suppose we have a call-site  $i$  in the code. What's the intuition behind simulation?

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



**Property of context tunneled call-sites**

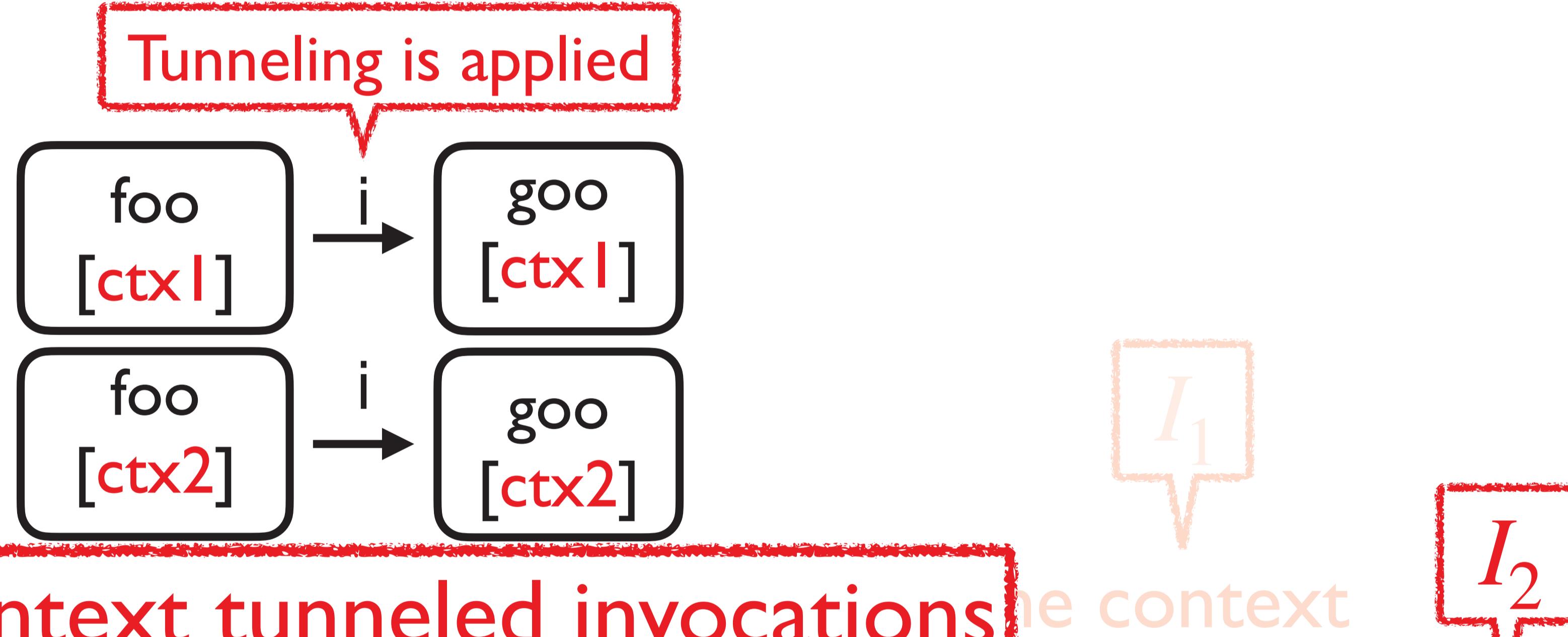
$I_1$

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

# Intuition Behind Simulation ( $I_1 \cup I_2$ )

- Suppose  $i$  is a tunneled invocation. Then  $i$  is simulated by  $I_1 \cup I_2$ .

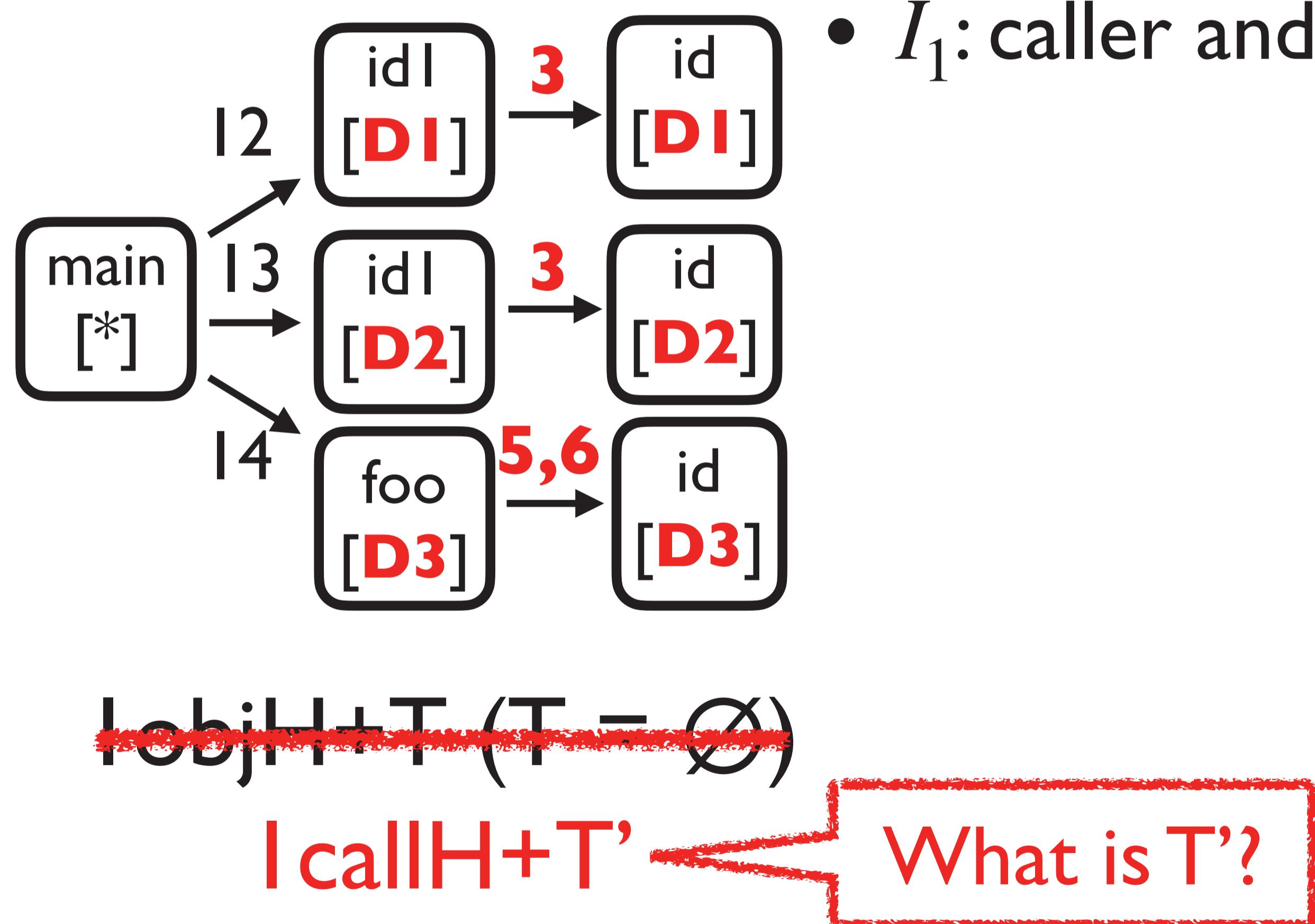
- If tunneling is applied to  $i$ , two properties inevitably appear at  $i$



- **Property of context tunneled invocations**
  - Property 2: different caller contexts imply different callee contexts

# Intuition Behind Simulation ( $I_1 \cup I_2$ )

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

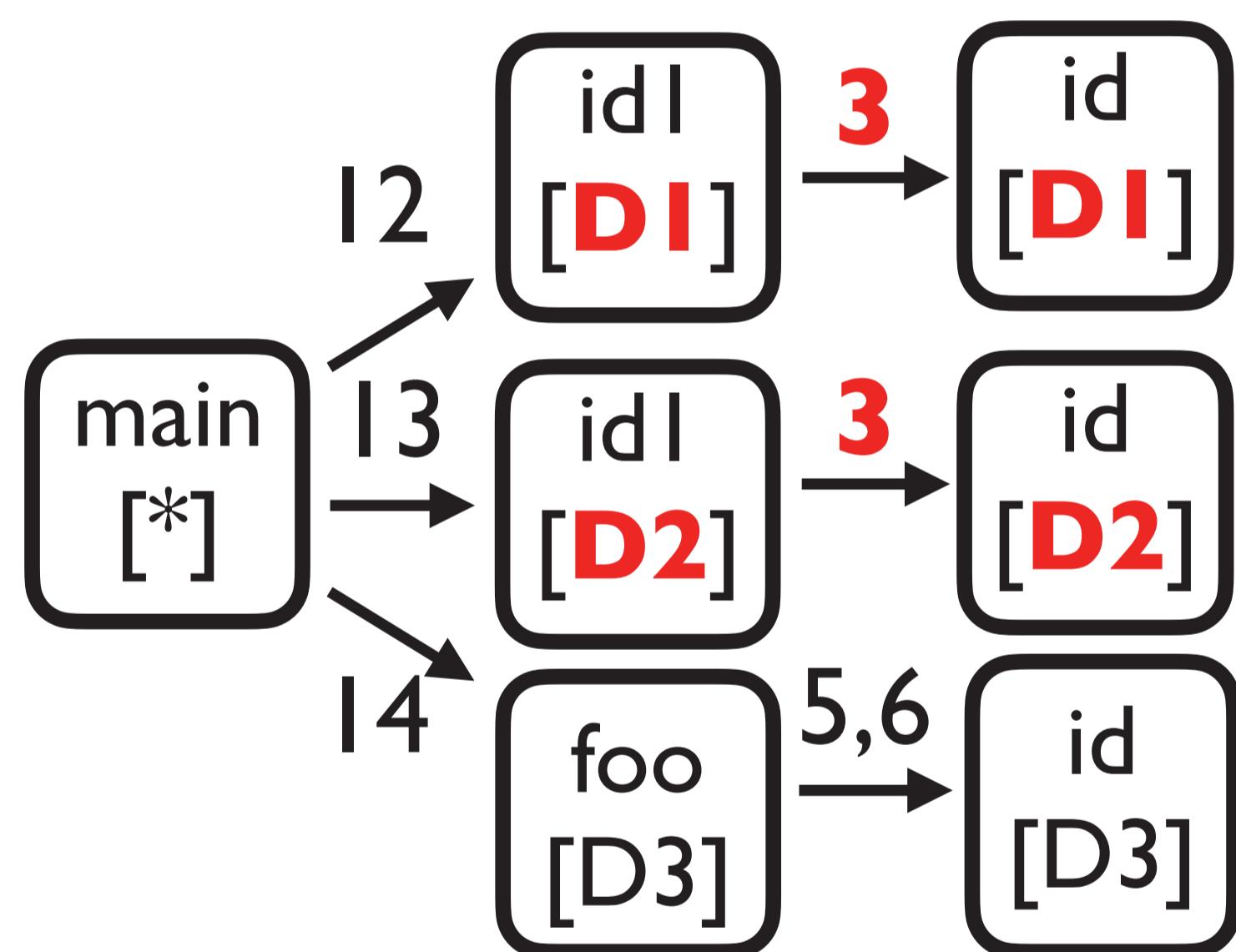


- $I_1$ : caller and callee methods have the **same context**

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

# Intuition Behind Simulation ( $I_1 \cup I_2$ )

- Suppose given call-graph is produced from  $I_{callH+T'}$  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  
 $I_2 = \{3\}$

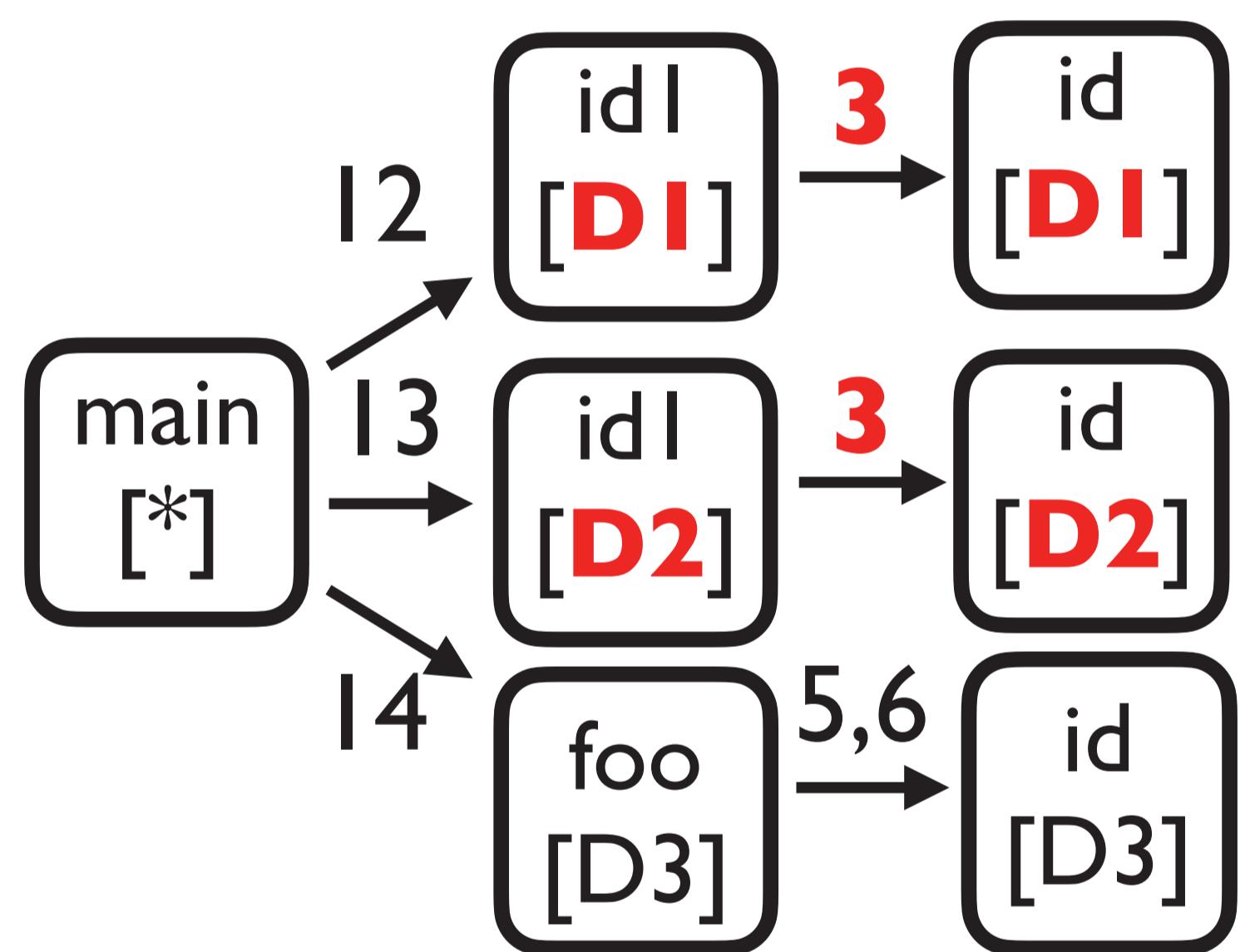
~~$I_{objH+T} (T = \emptyset)$~~

$I_{callH+T'}$

What is  $T'$ ?

# Intuition Behind Simulation ( $I_1 \cup I_2$ )

- Suppose given call-graph is produced from  $I_{callH+T}$  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  
 $I_2 = \{3\}$

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

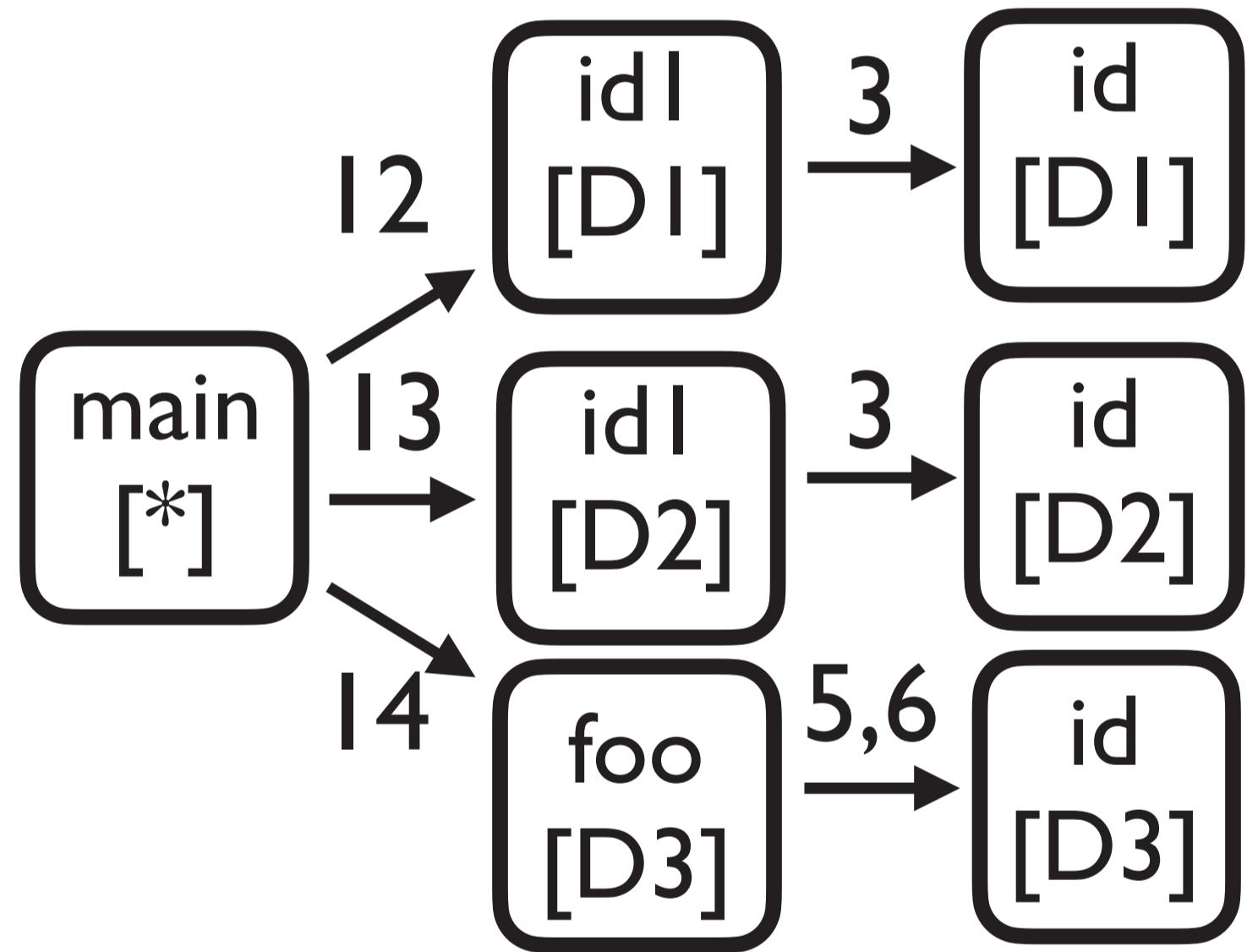
$I_{callH+T'}$

What is  $T'$ ?

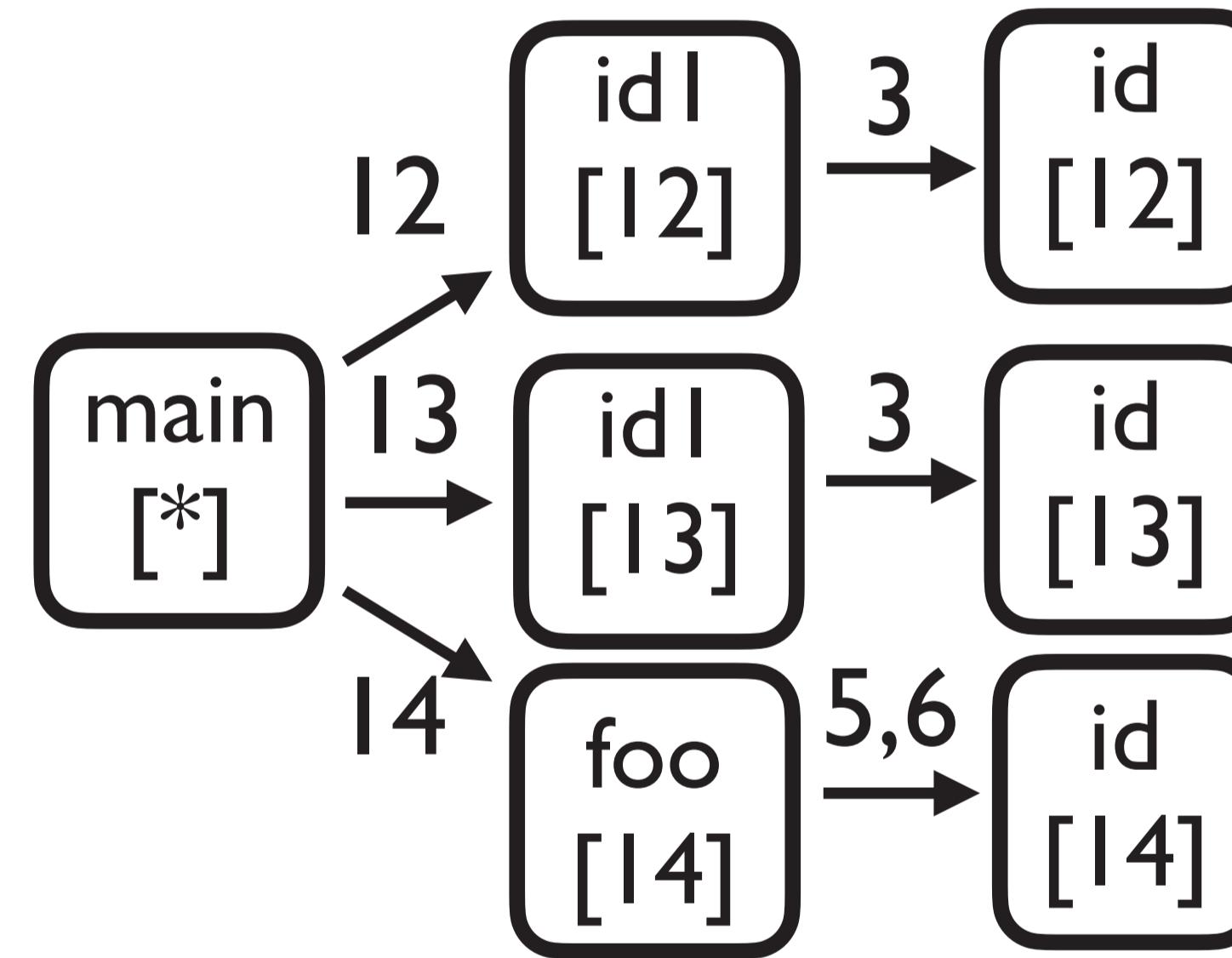
$$T' = I_1 \cup I_2 = \{3, 5, 6\}$$

# Intuition Behind Simulation ( $I_1 \cup I_2$ )

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



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

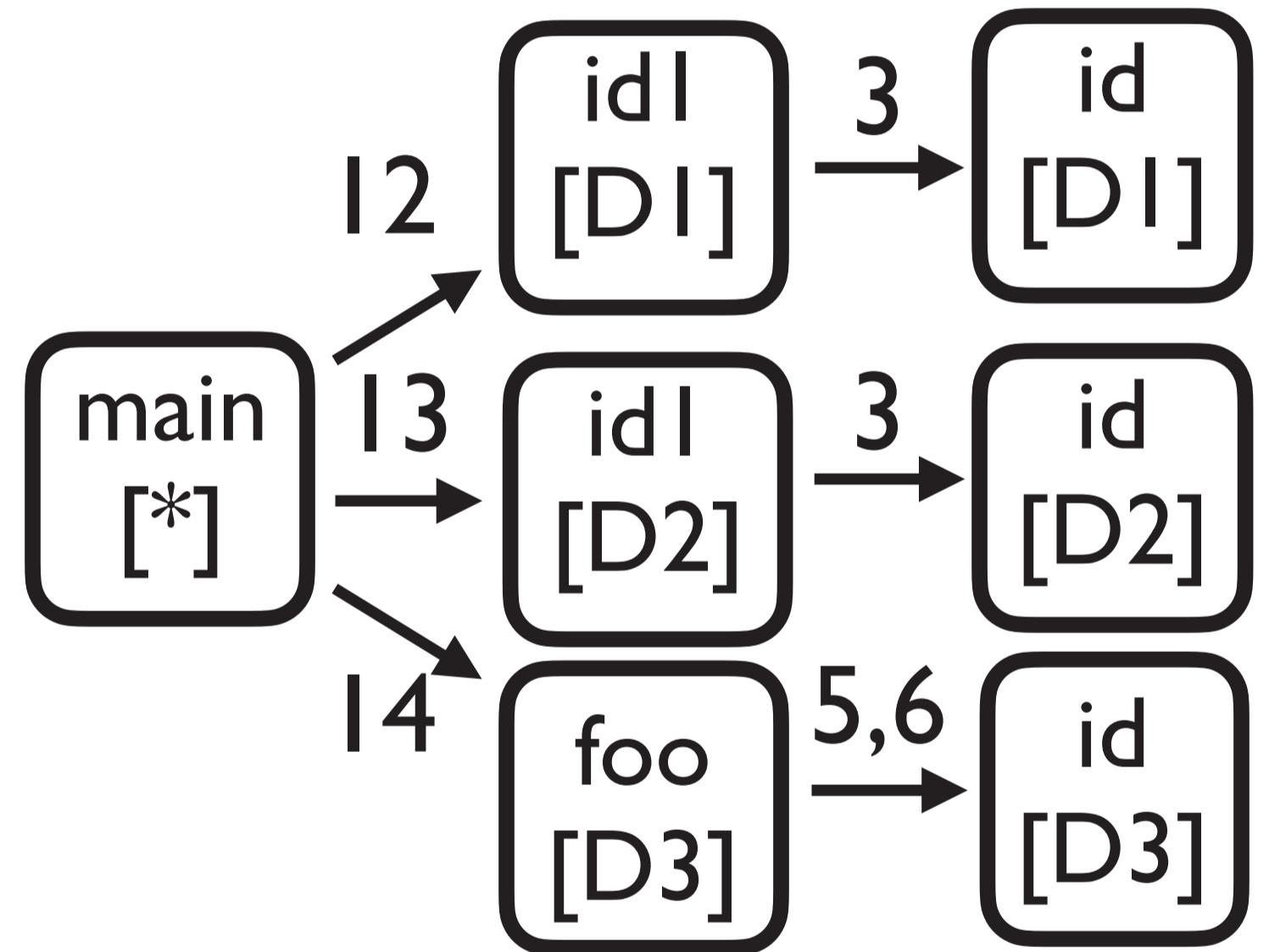


$I_{\text{callH+T'}}$  ( $T' = \{3,5,6\}$ )

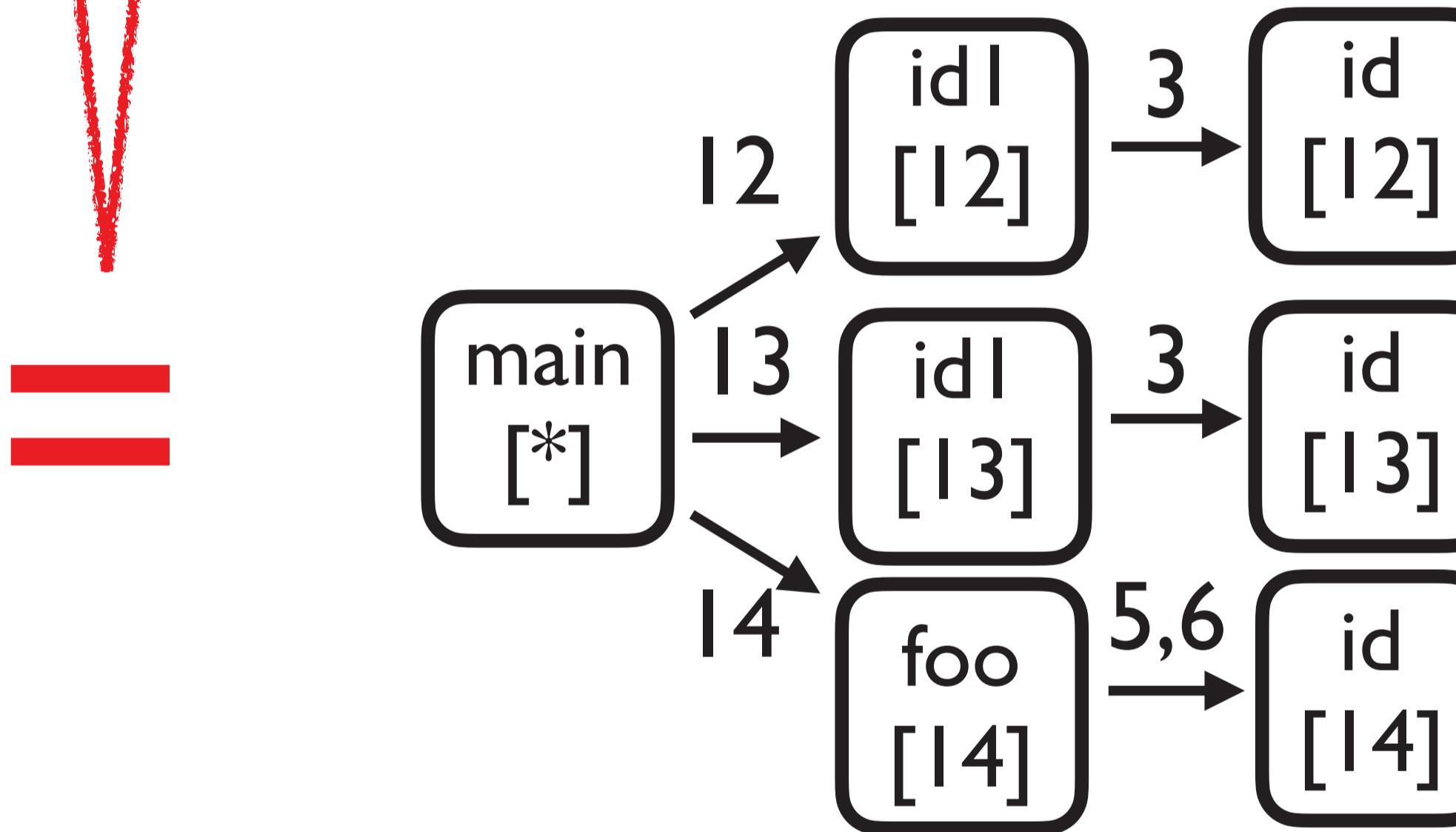
# Intuition Behind Simulation ( $I_1 \cup I_2$ )

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

Exactly the same analyses



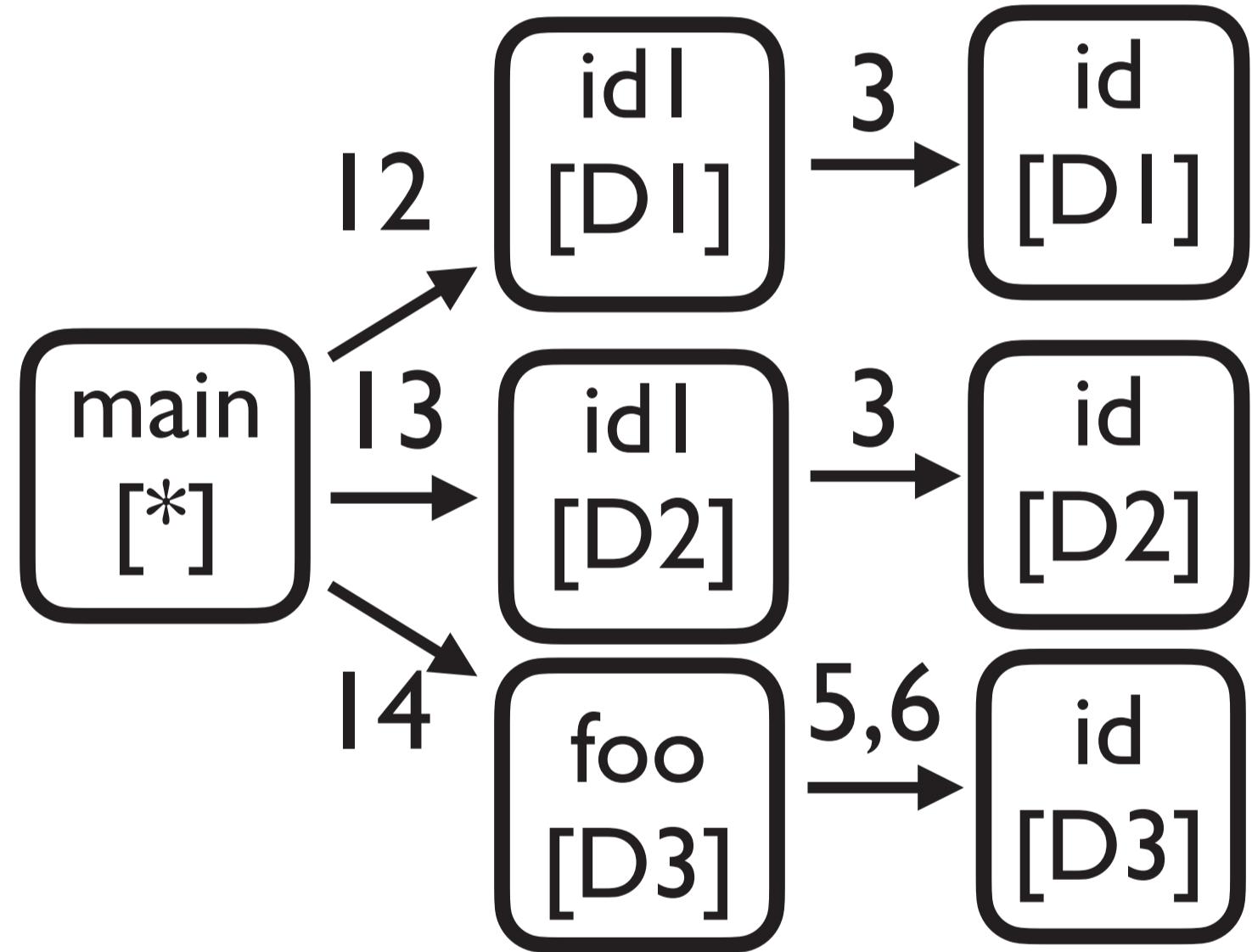
$I_{obj}H+T$  ( $T = \emptyset$ )



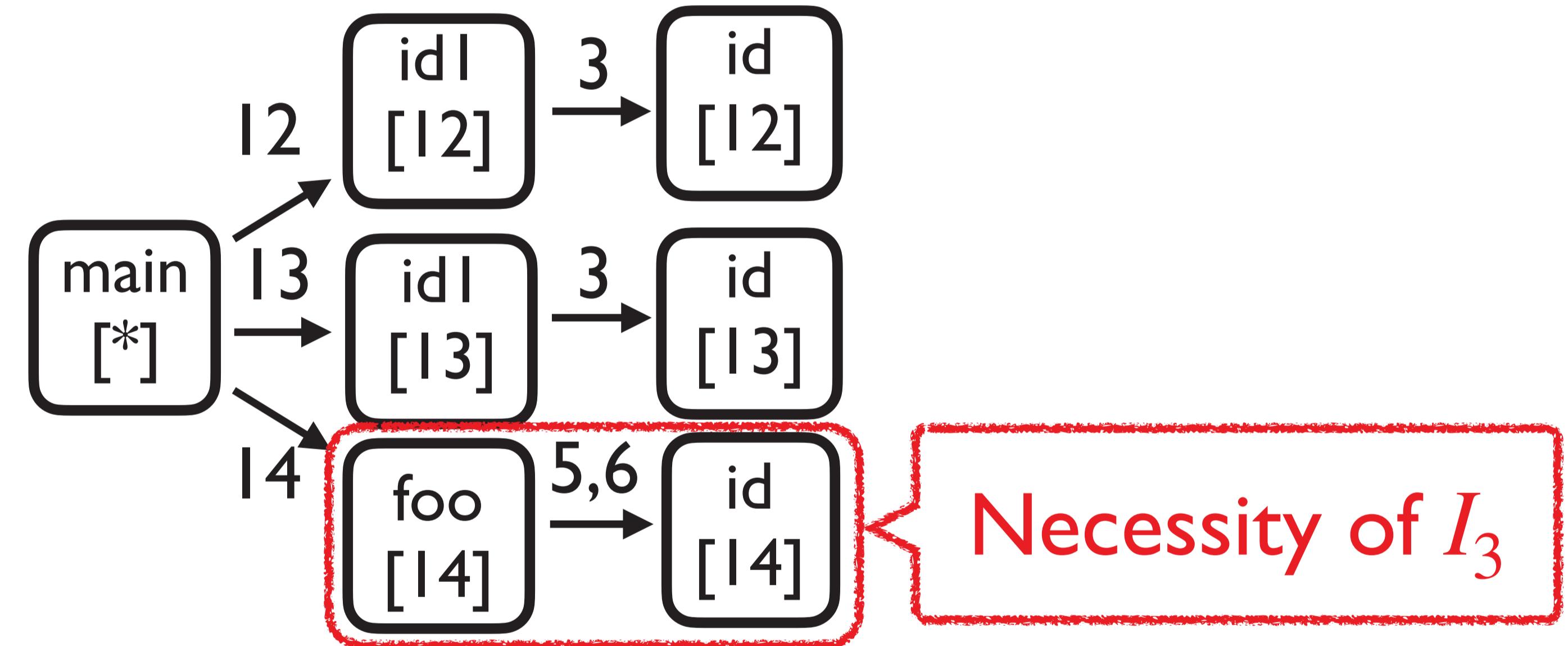
$I_{call}H+T'$  ( $T' = \{3,5,6\}$ )

# Intuition Behind Simulation ( $I_1 \cup I_2$ )

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



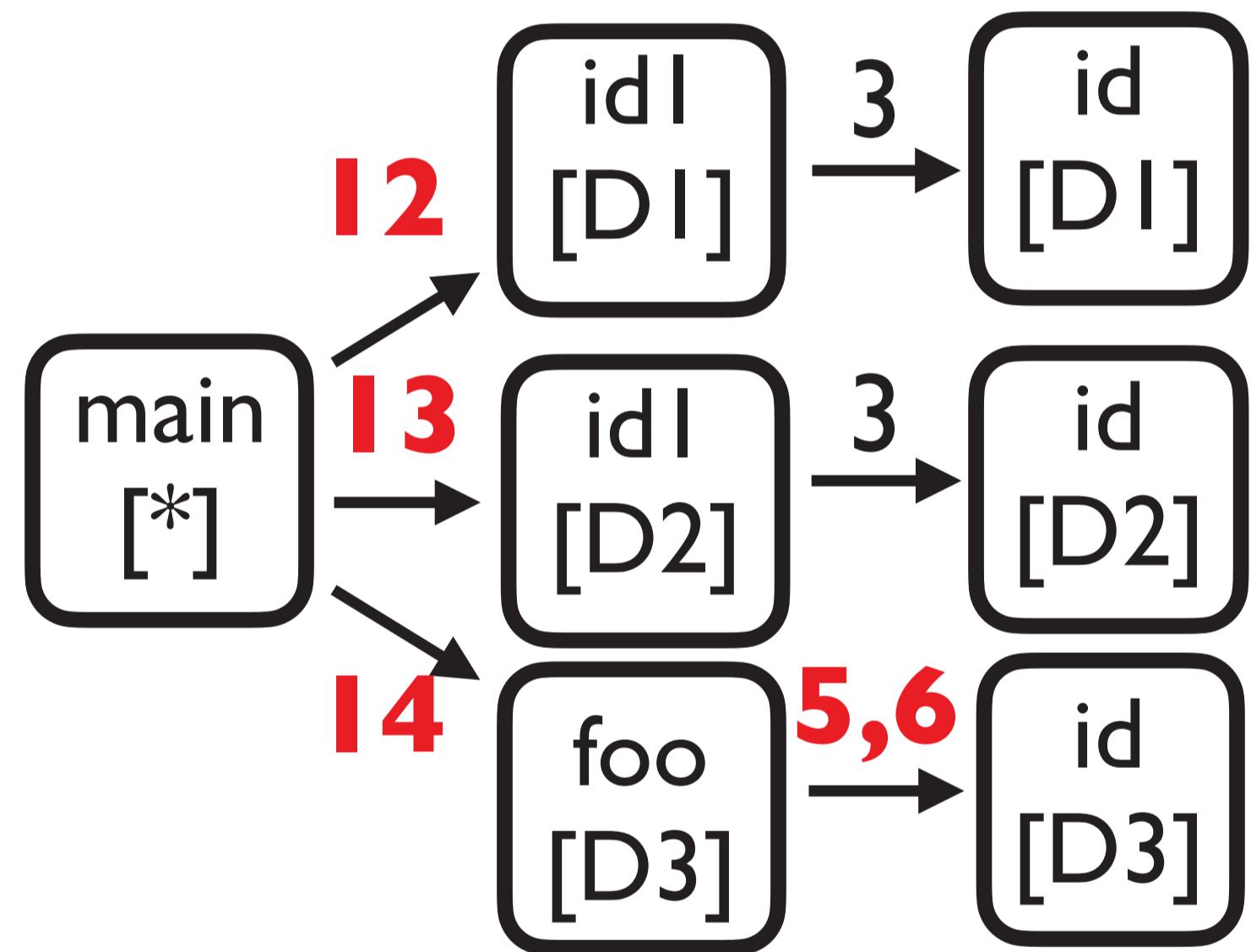
$I_{\text{objH+T}}$  ( $T = \emptyset$ )



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

# Intuition Behind Simulation ( $I_3$ )

- $I_3$  : Tunneling should be avoided for improving precision



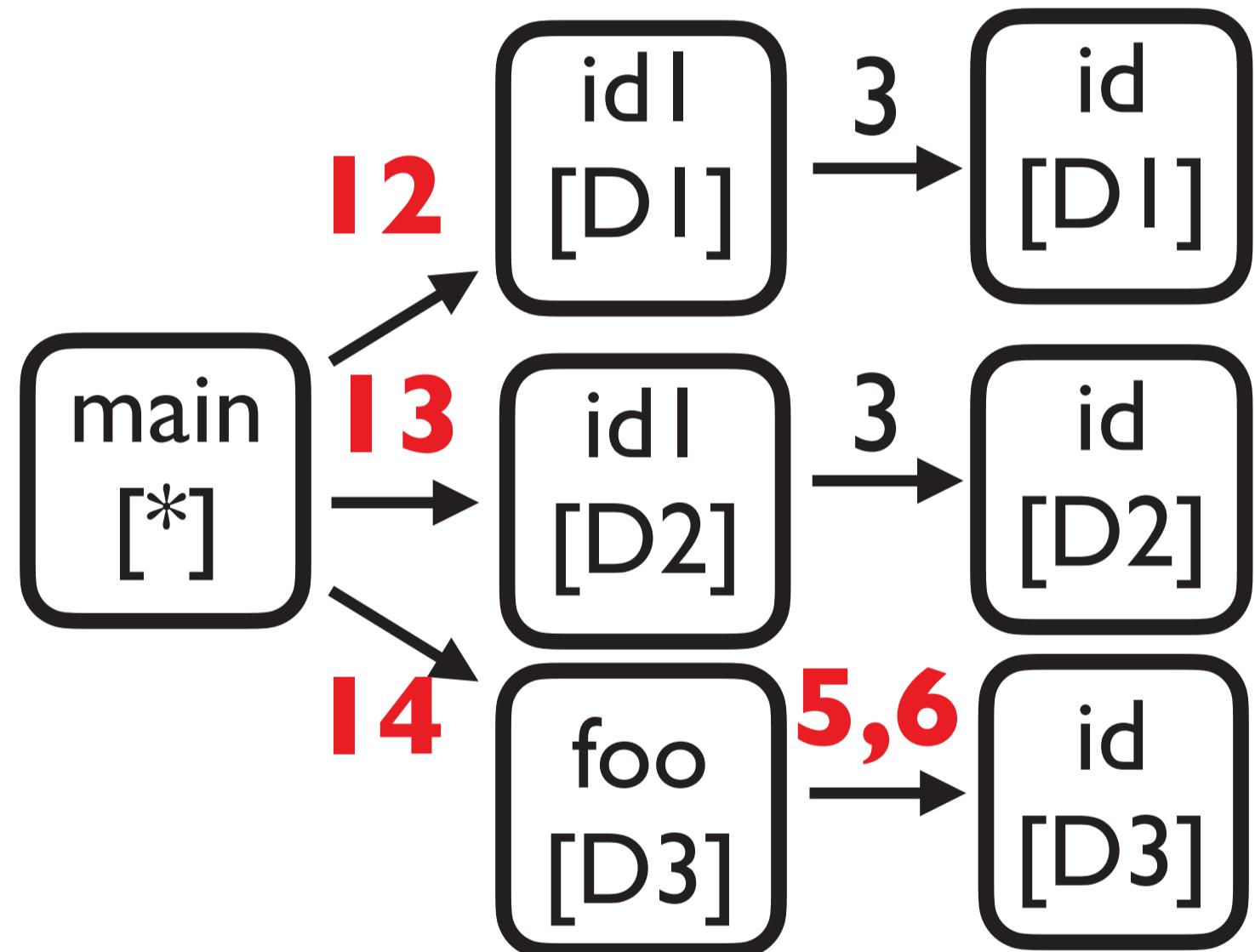
$\text{lobjH} + T$  ( $T = \emptyset$ )

- $I_1$ : 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

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

# Intuition Behind Simulation

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

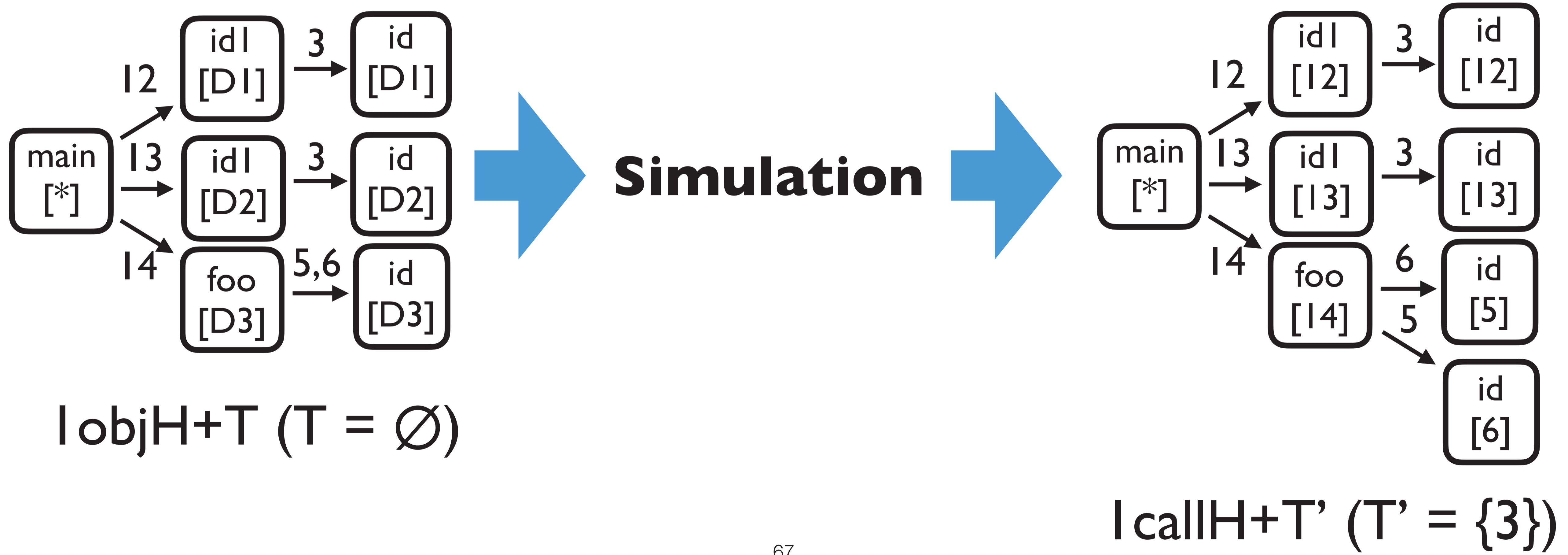


- $I_1$ : caller and callee methods have the **same context**  
 $I_1 = \{3, 5, 6\}$
- $I_2$ : different caller ctx imply  
 $I_2 = \{3\}$
- $I_3$ : given object sensitivity produced only one context  
 $I_3 = \{5, 6, T2, T3, T4\}$

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

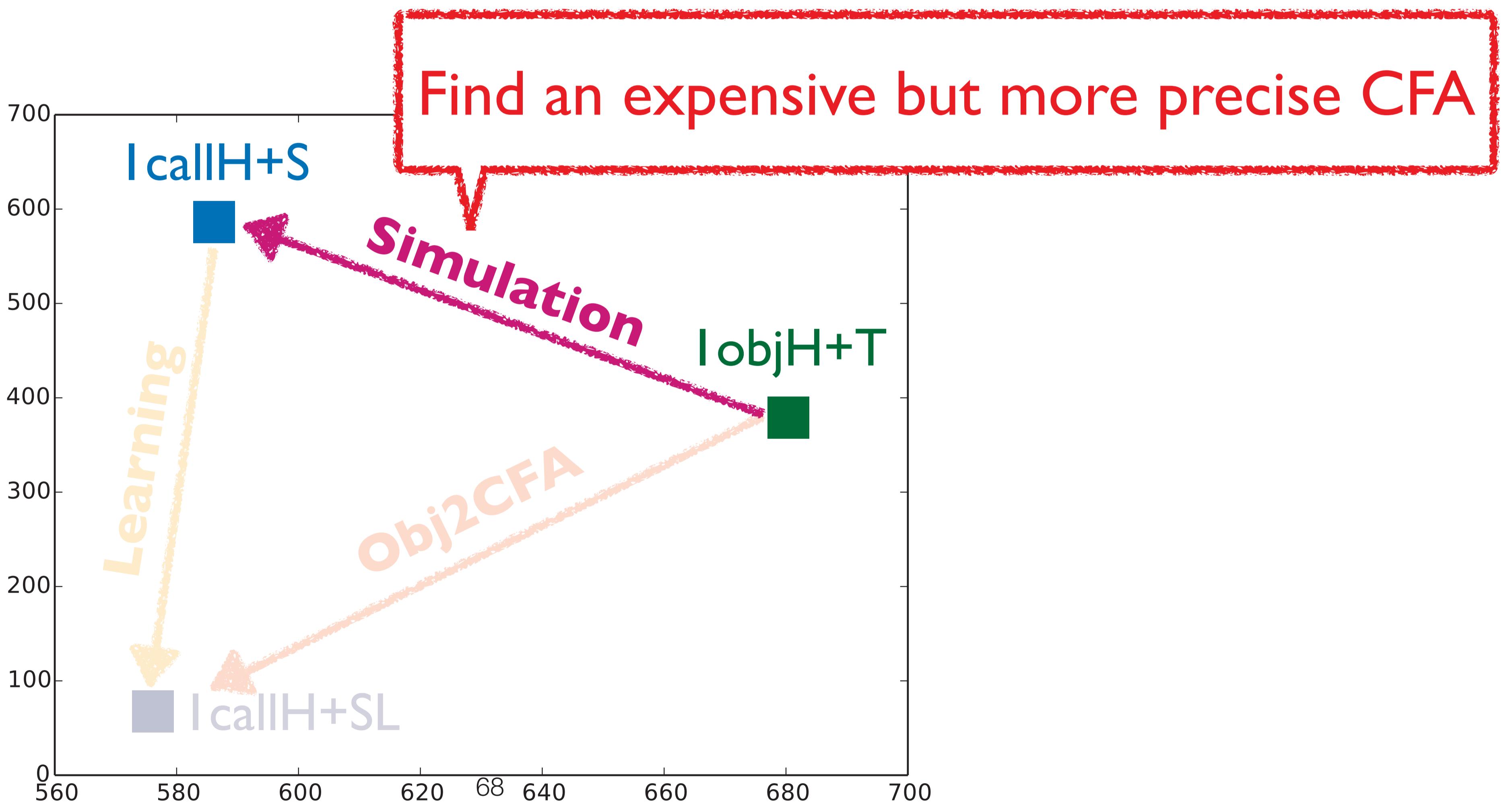
# Technique I: Simulation

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



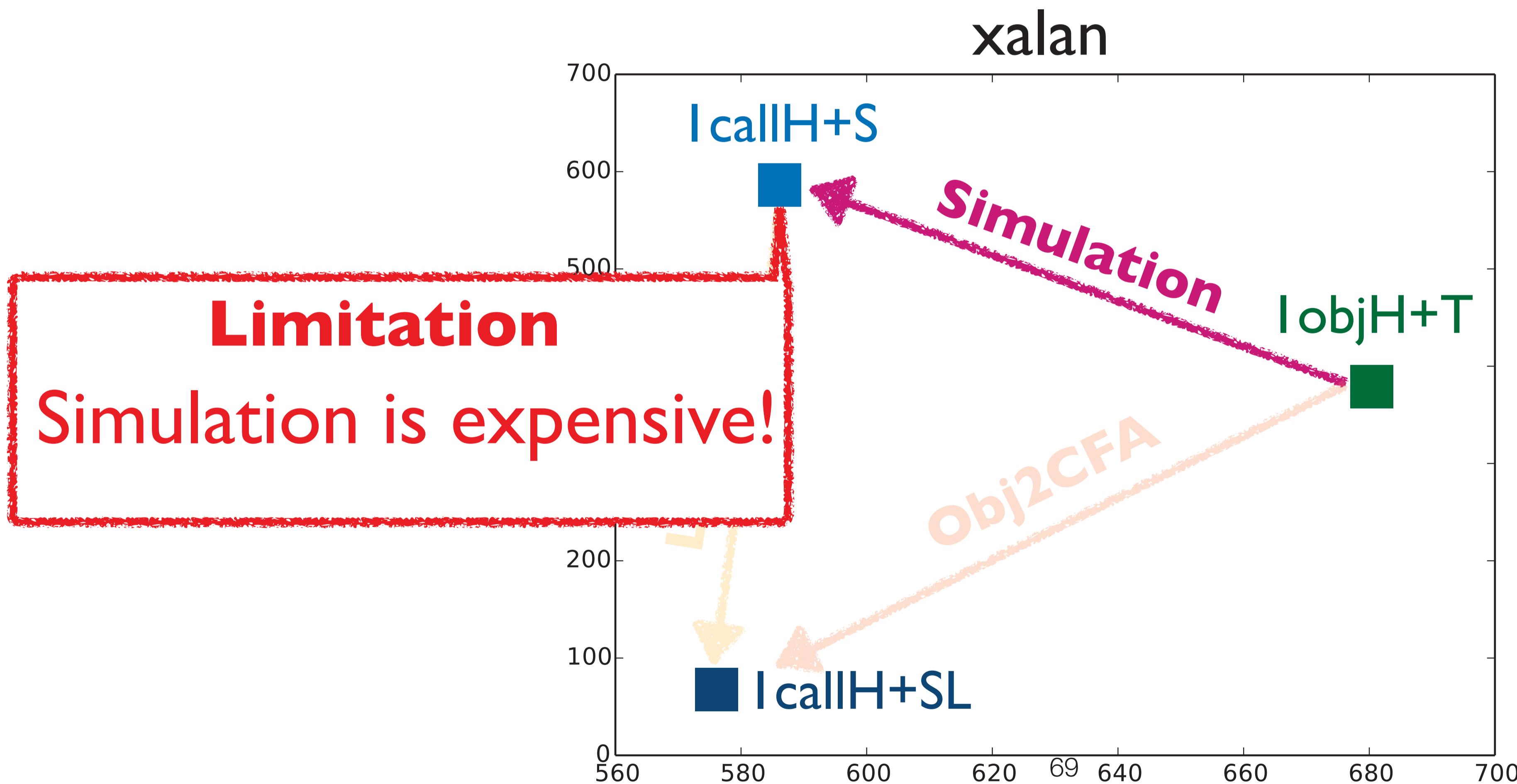
# Our Technique : **Obj2CFA**

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



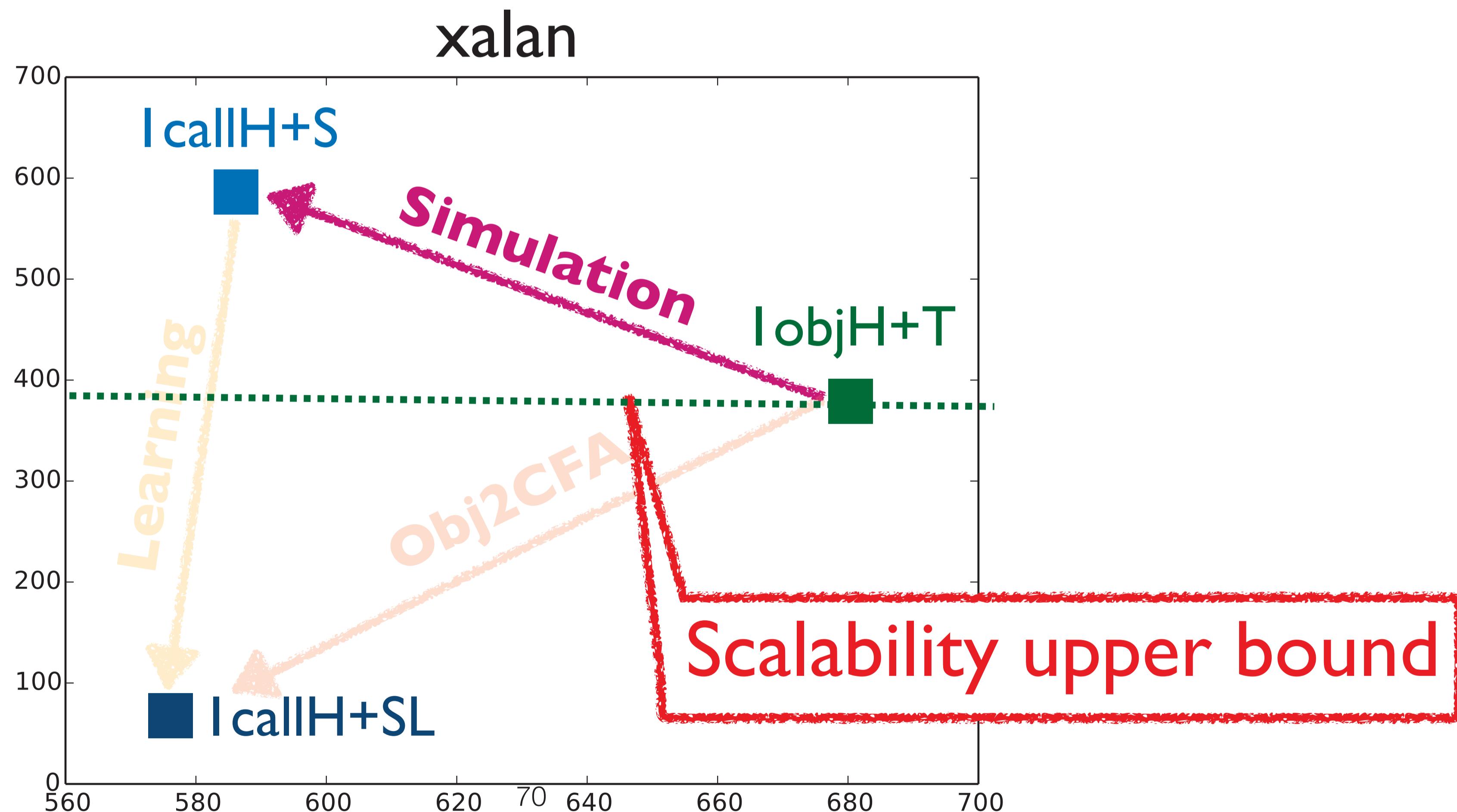
# Our Technique : **Obj2CFA**

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



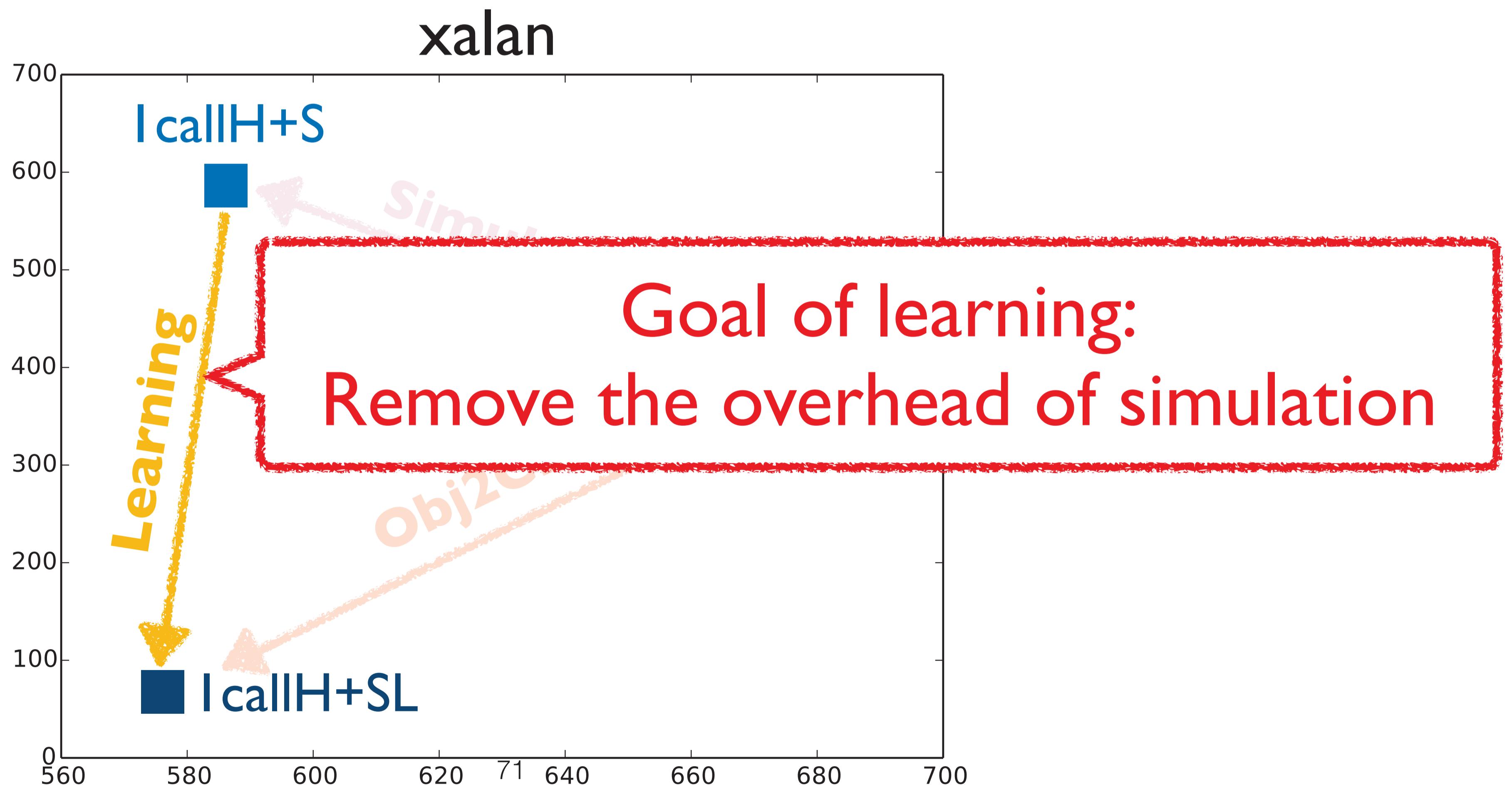
# Our Technique : **Obj2CFA**

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



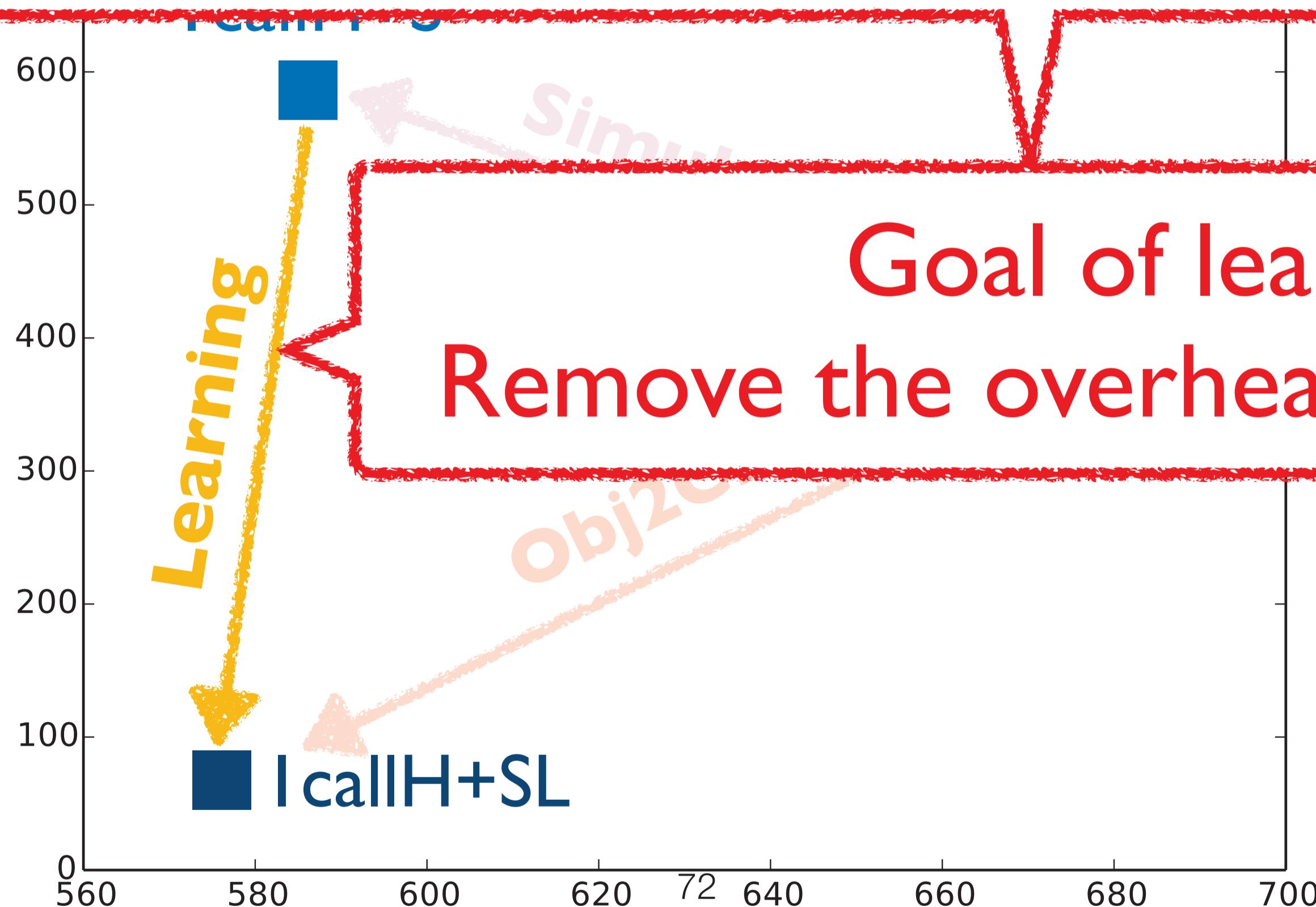
# Our Technique : **Obj2CFA**

- **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 · **OhioCFA**

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**

IcallH+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

# 2009 (OOPSLA)

# 2011 (POPL)

# 2013 (PLDI)

# 2014 (PLDI)

# 2016 (SAS)

# 2017 (OOPSLA)

# Setting

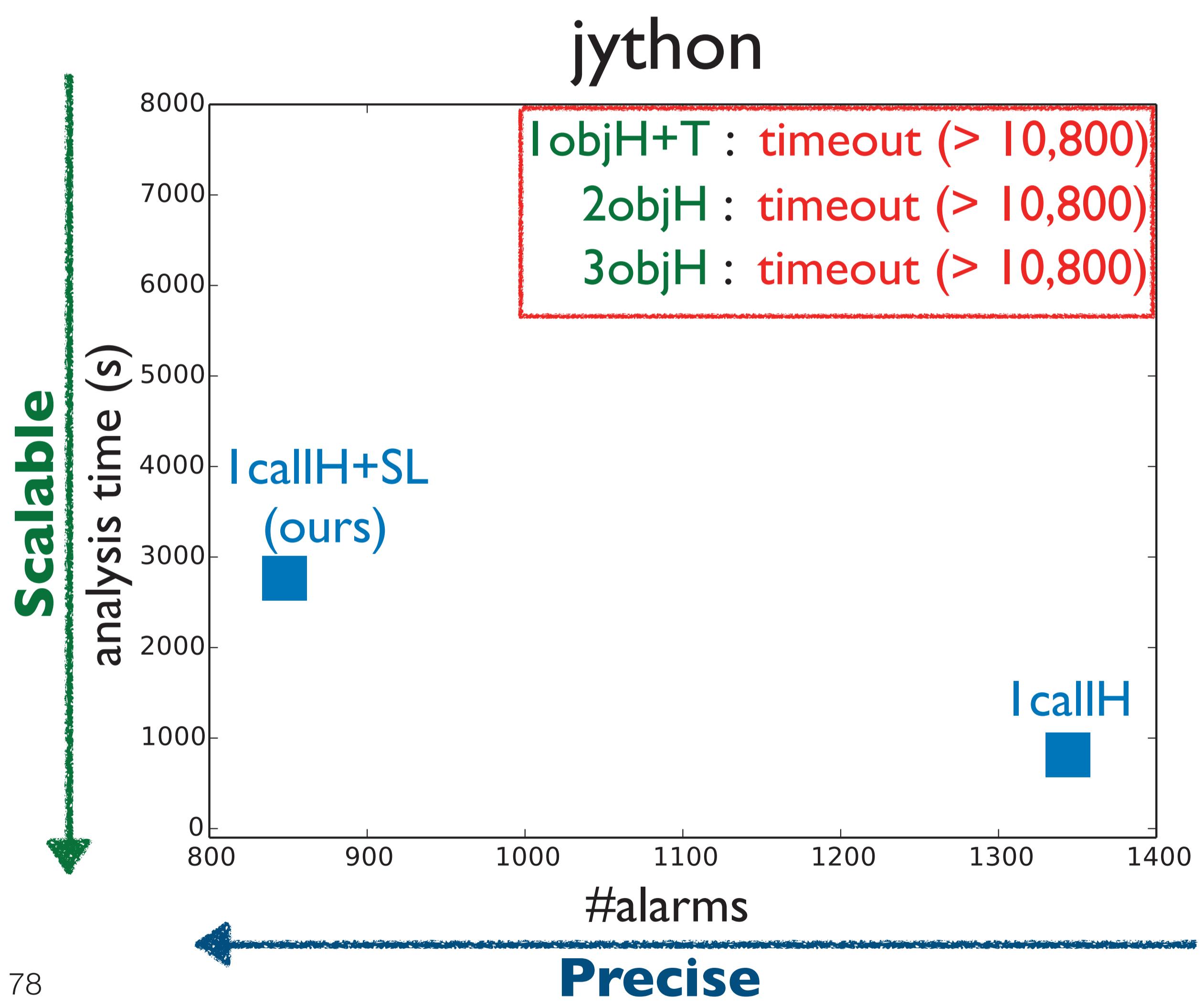
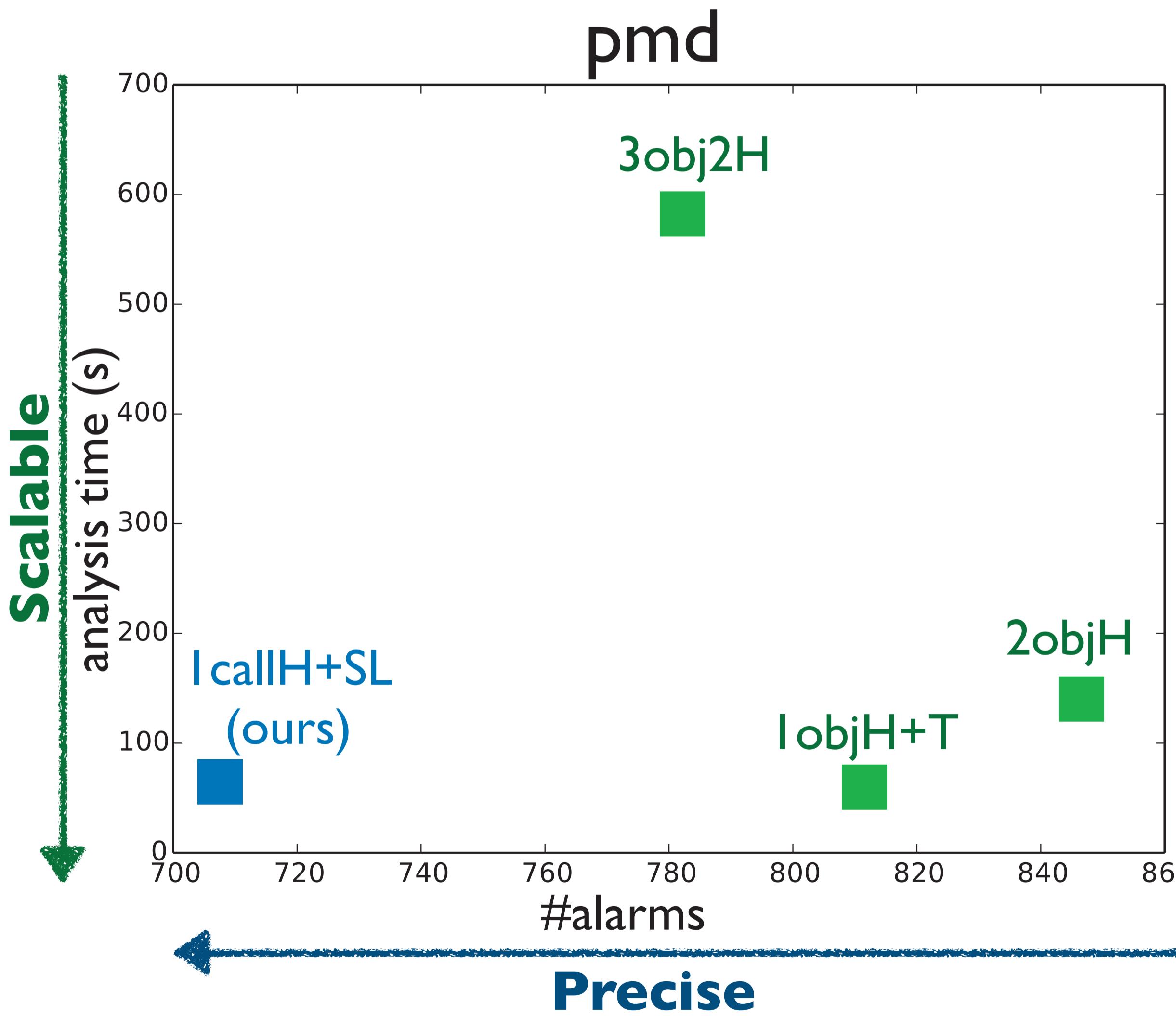
- Doop
  - Pointer analysis framework for Java
- Research Question: which one is better?

**Call-site sensitivity vs Object sensitivity**

Context tunneling is included

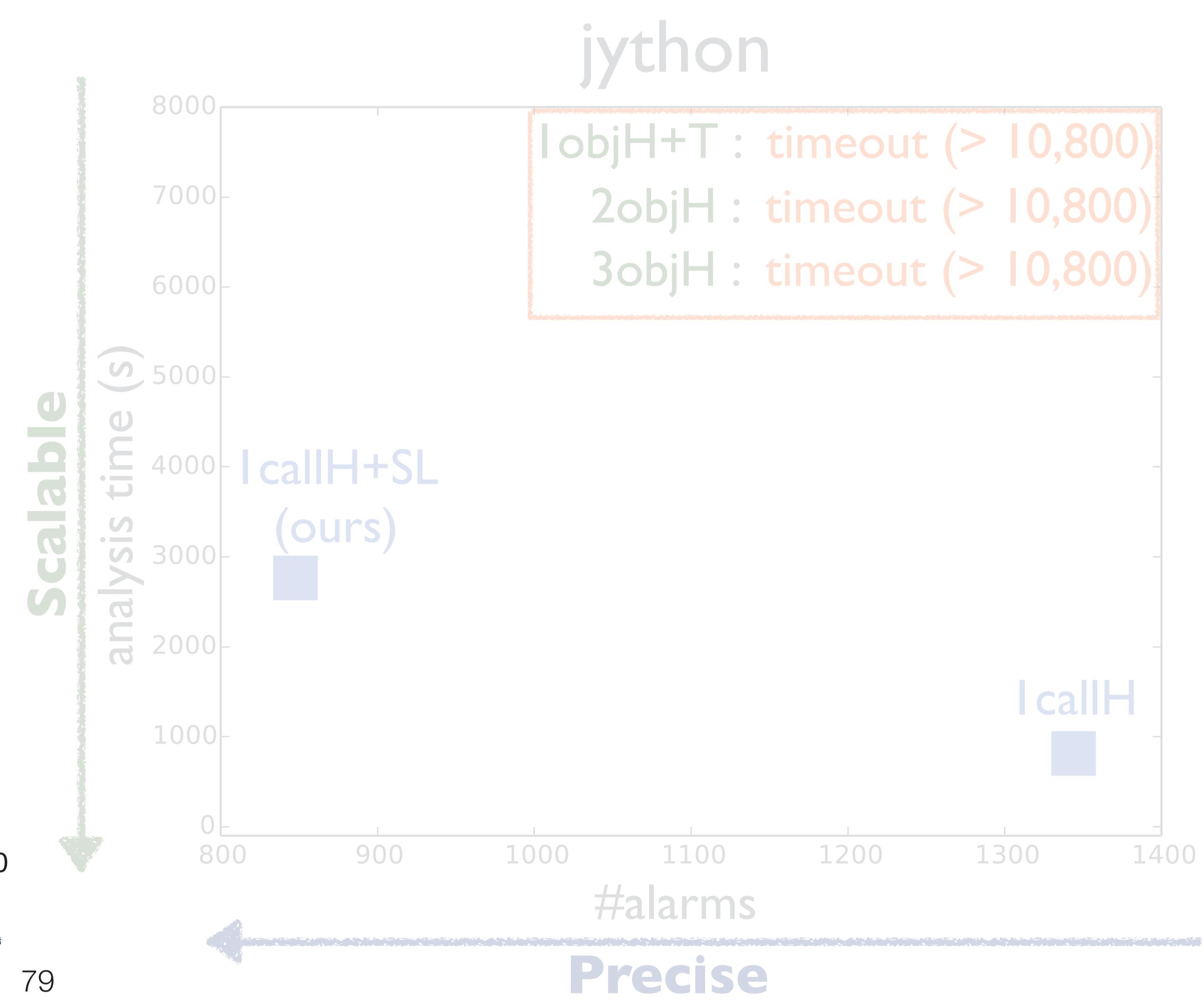
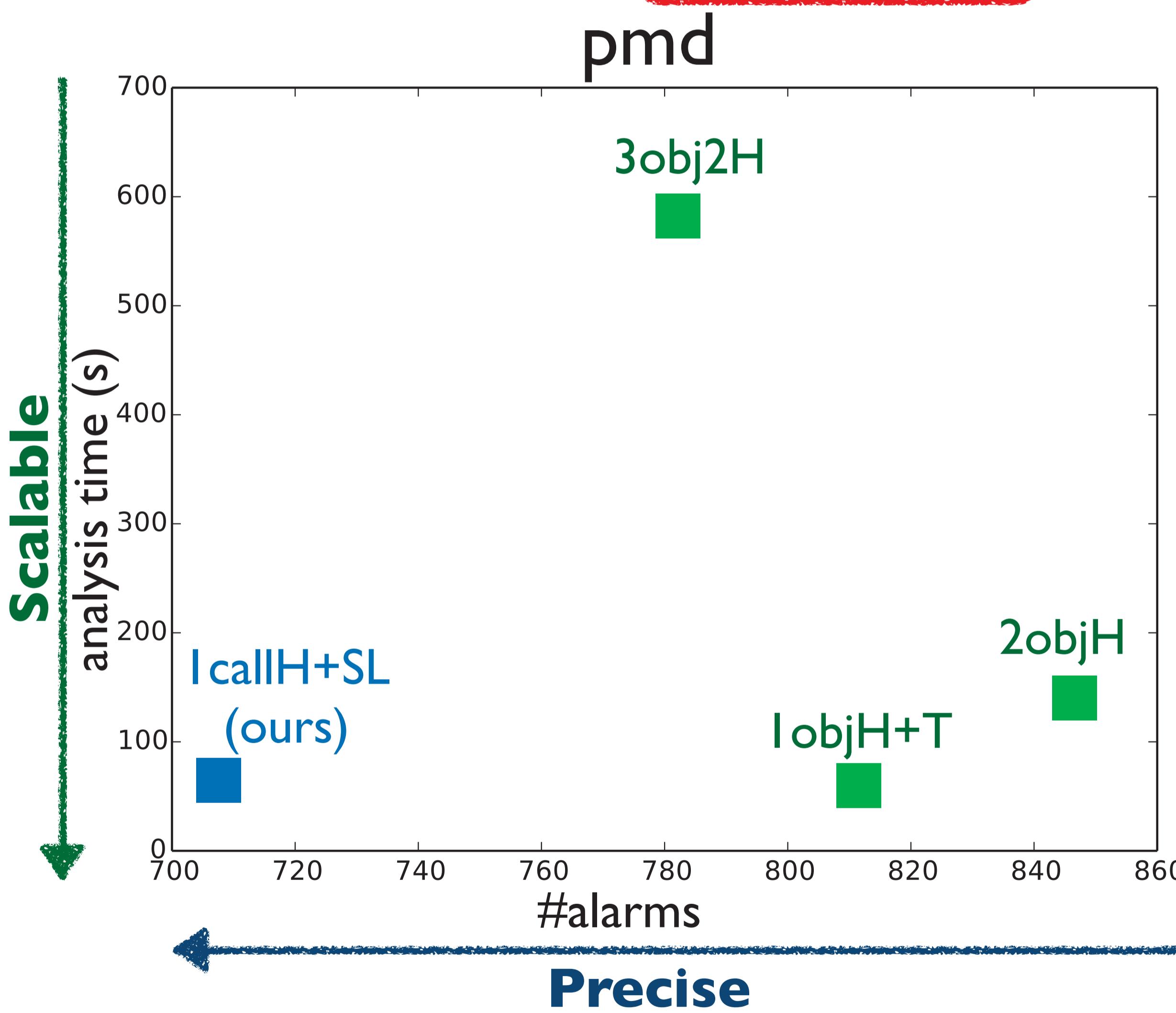
# Call-site Sensitivity vs Object Sensitivity

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



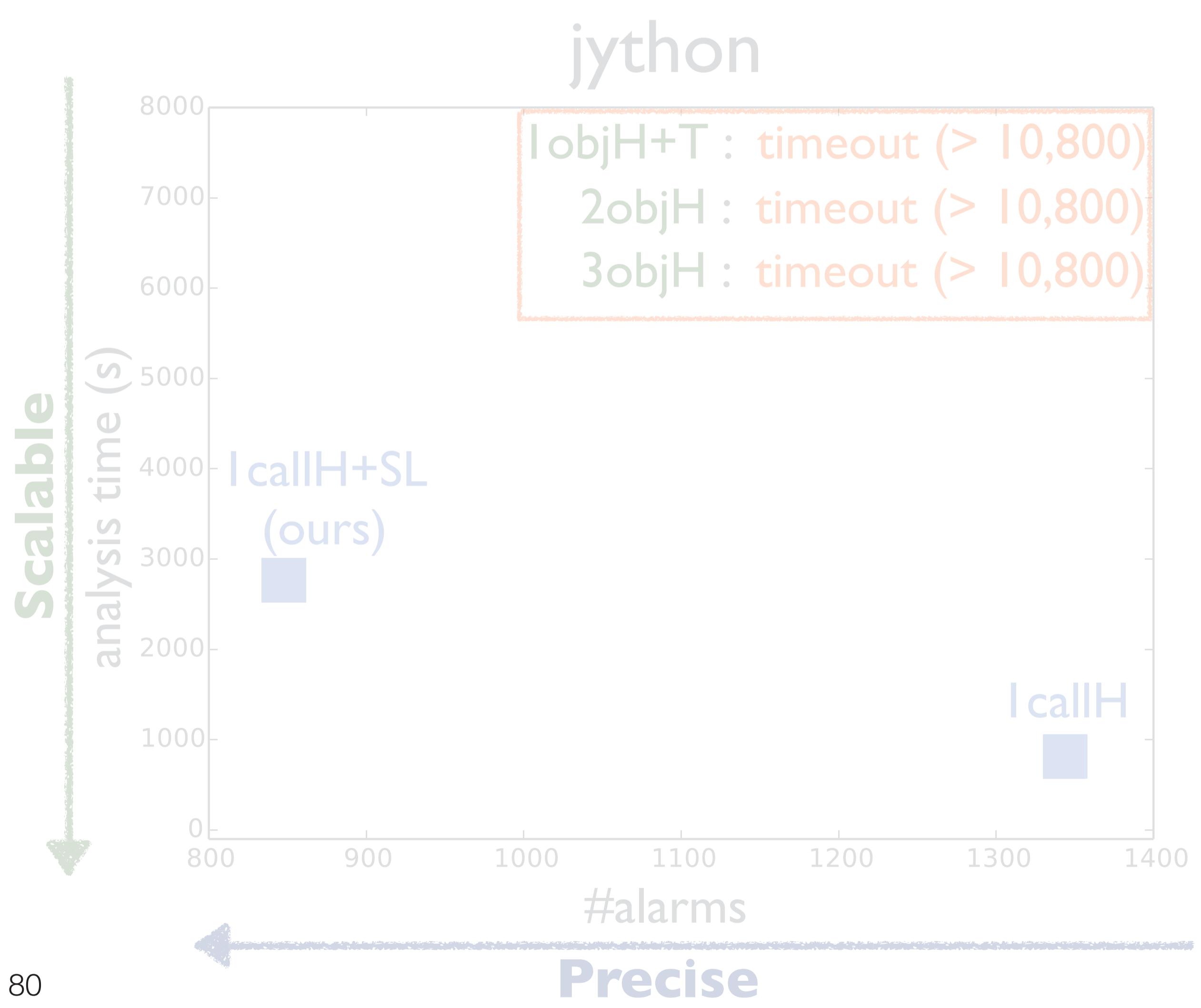
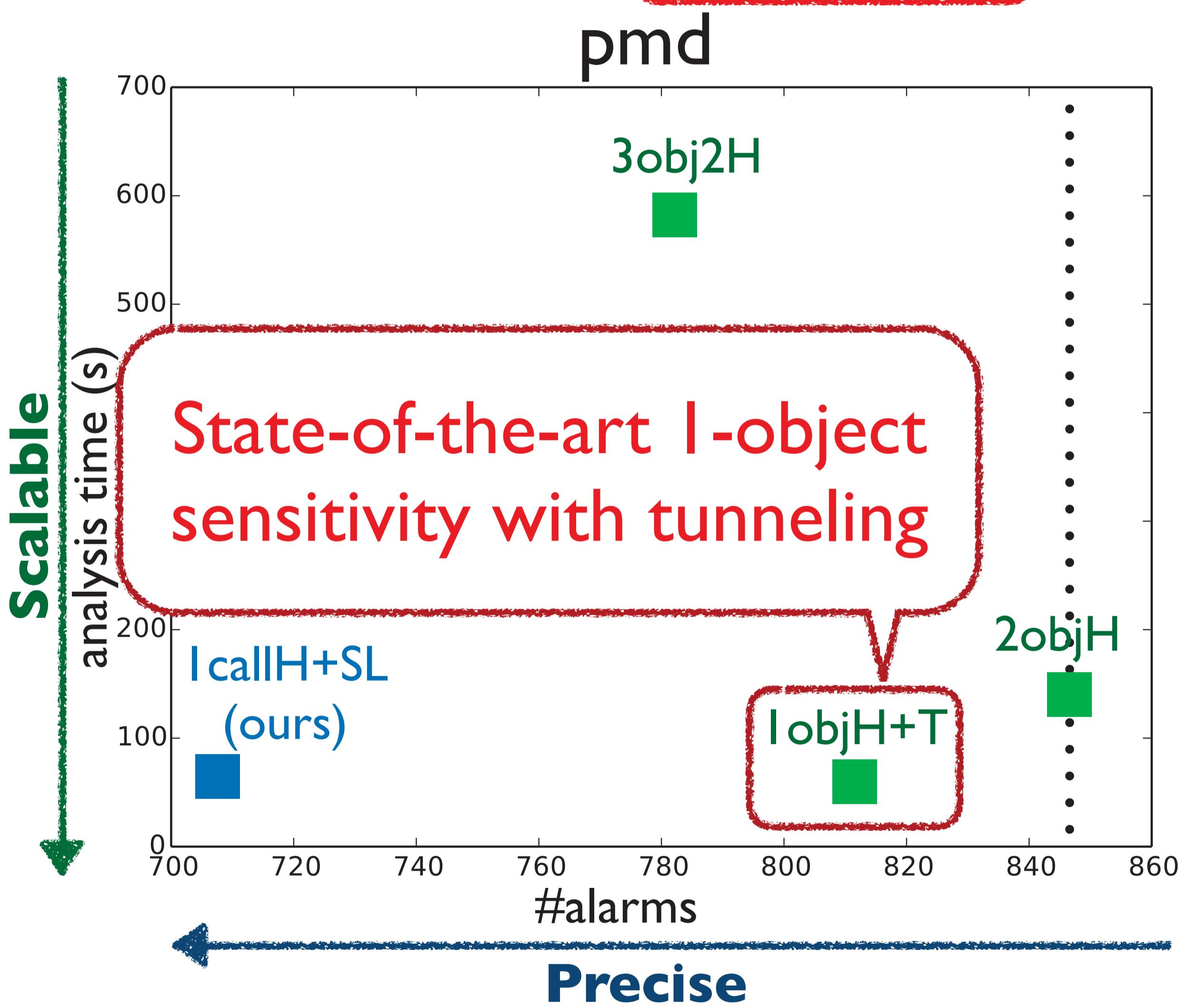
# Call-site Sensitivity vs Object Sensitivity

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



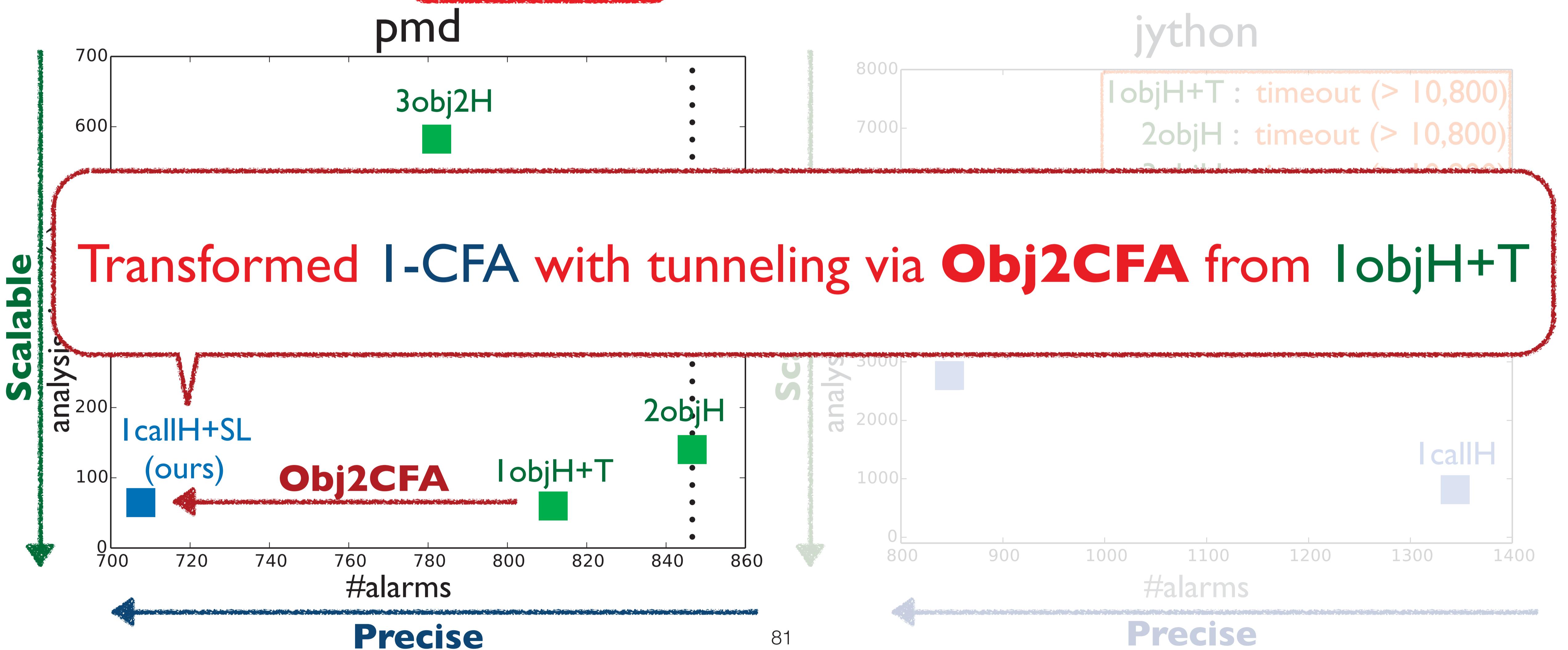
# Call-site Sensitivity vs Object Sensitivity

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



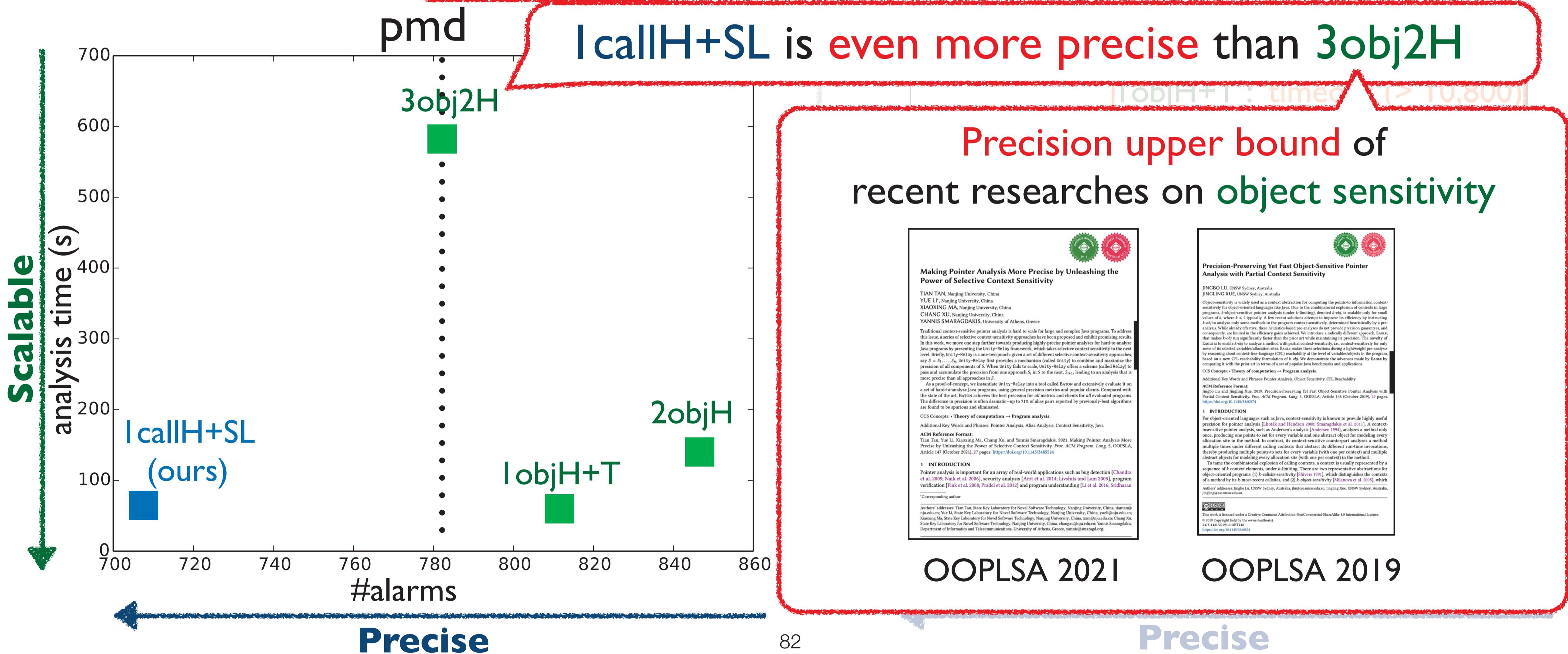
# Call-site Sensitivity vs Object Sensitivity

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



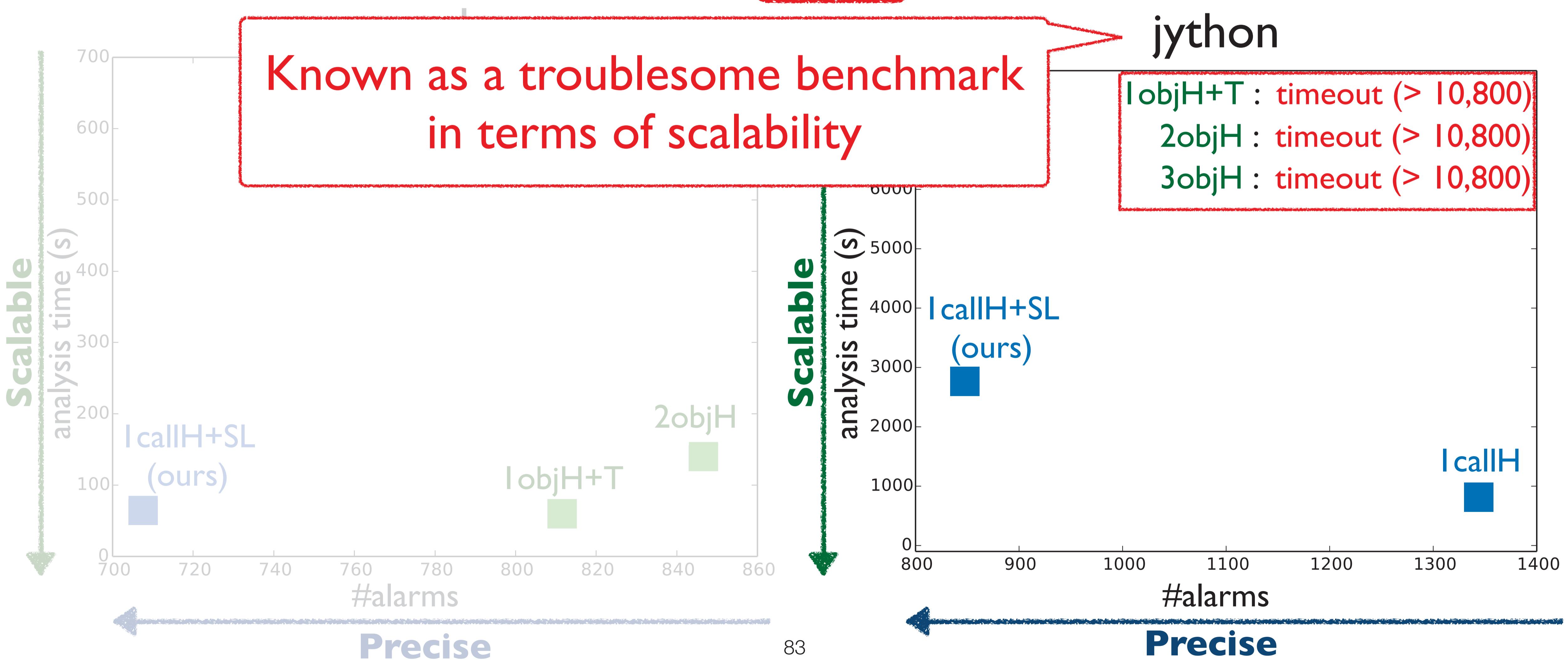
# Call-site Sensitivity vs Object Sensitivity

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



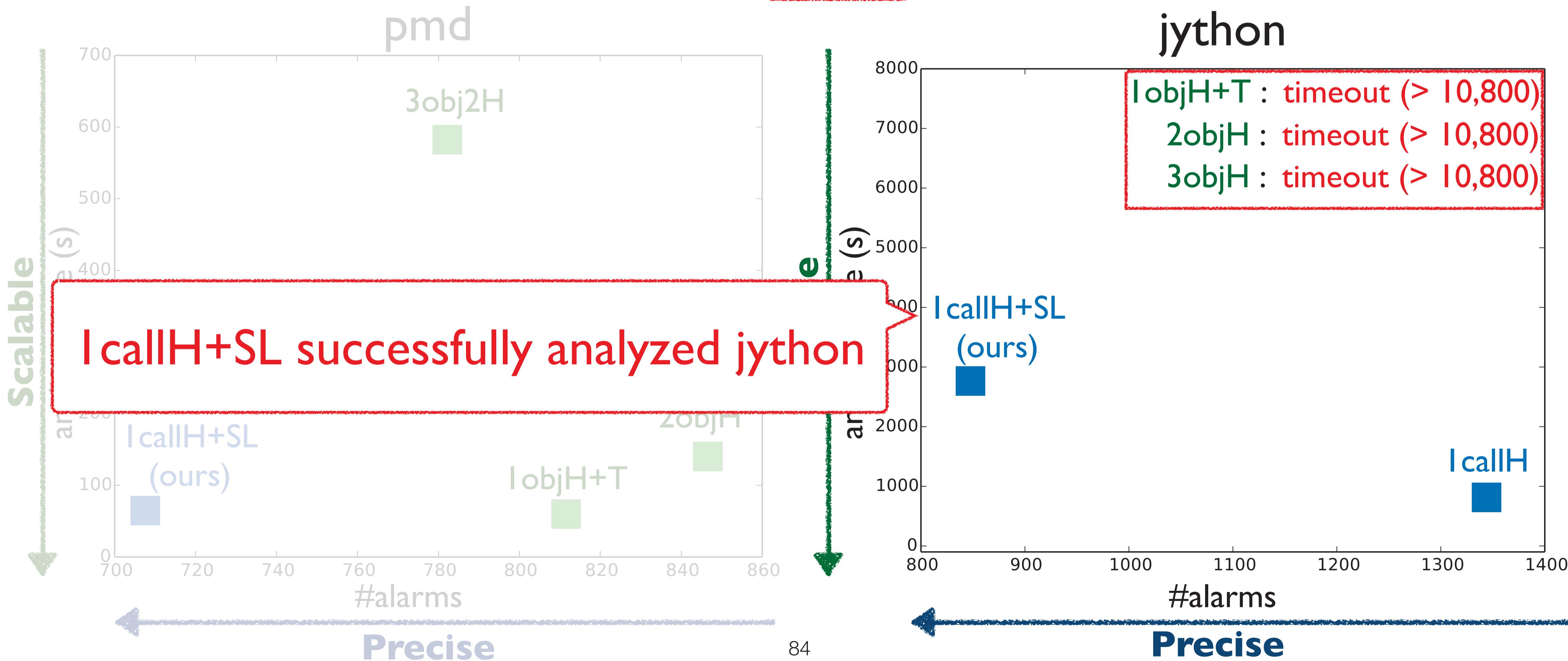
# Call-site Sensitivity vs Object Sensitivity

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



# Call-site Sensitivity vs Object Sensitivity

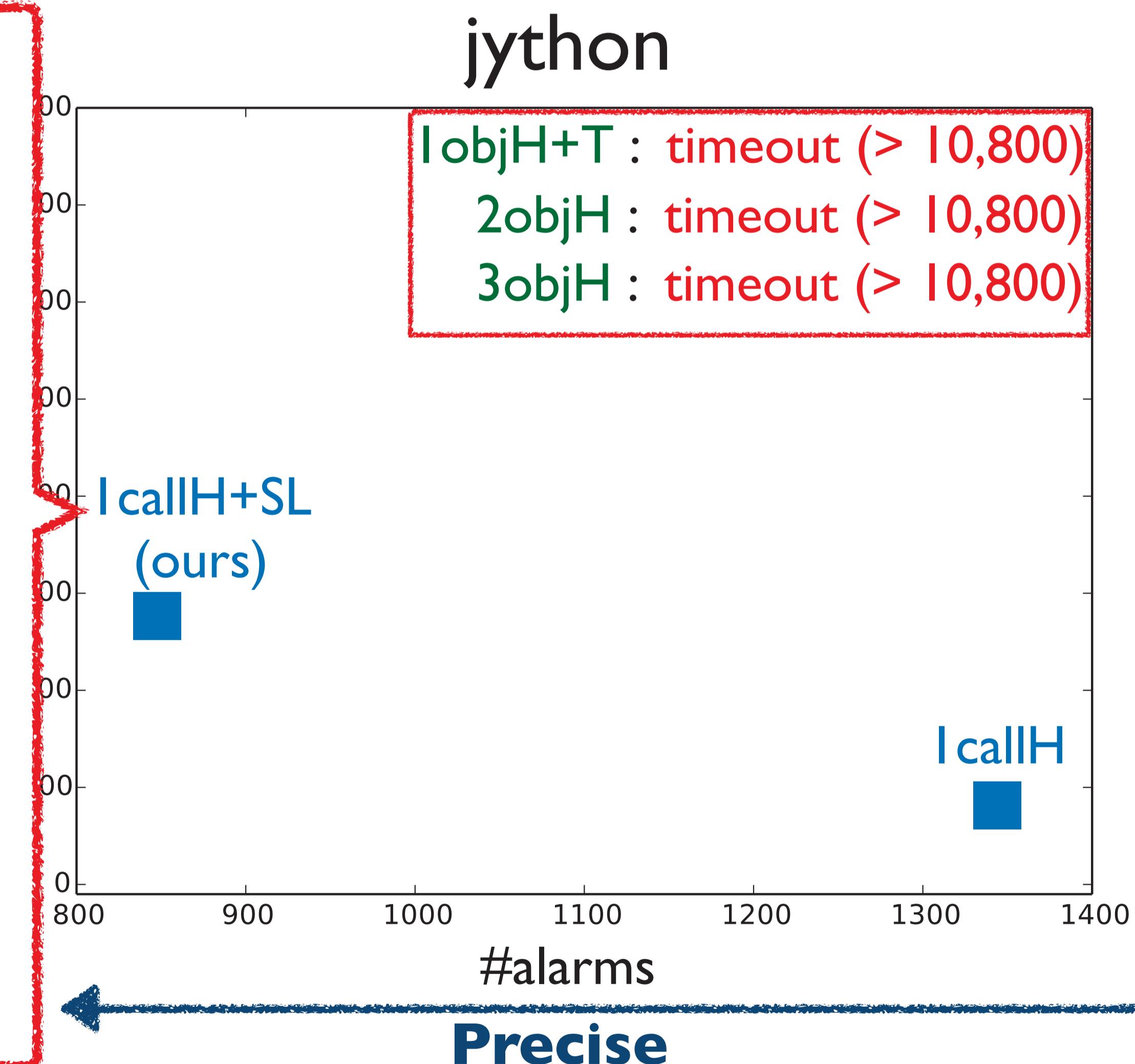
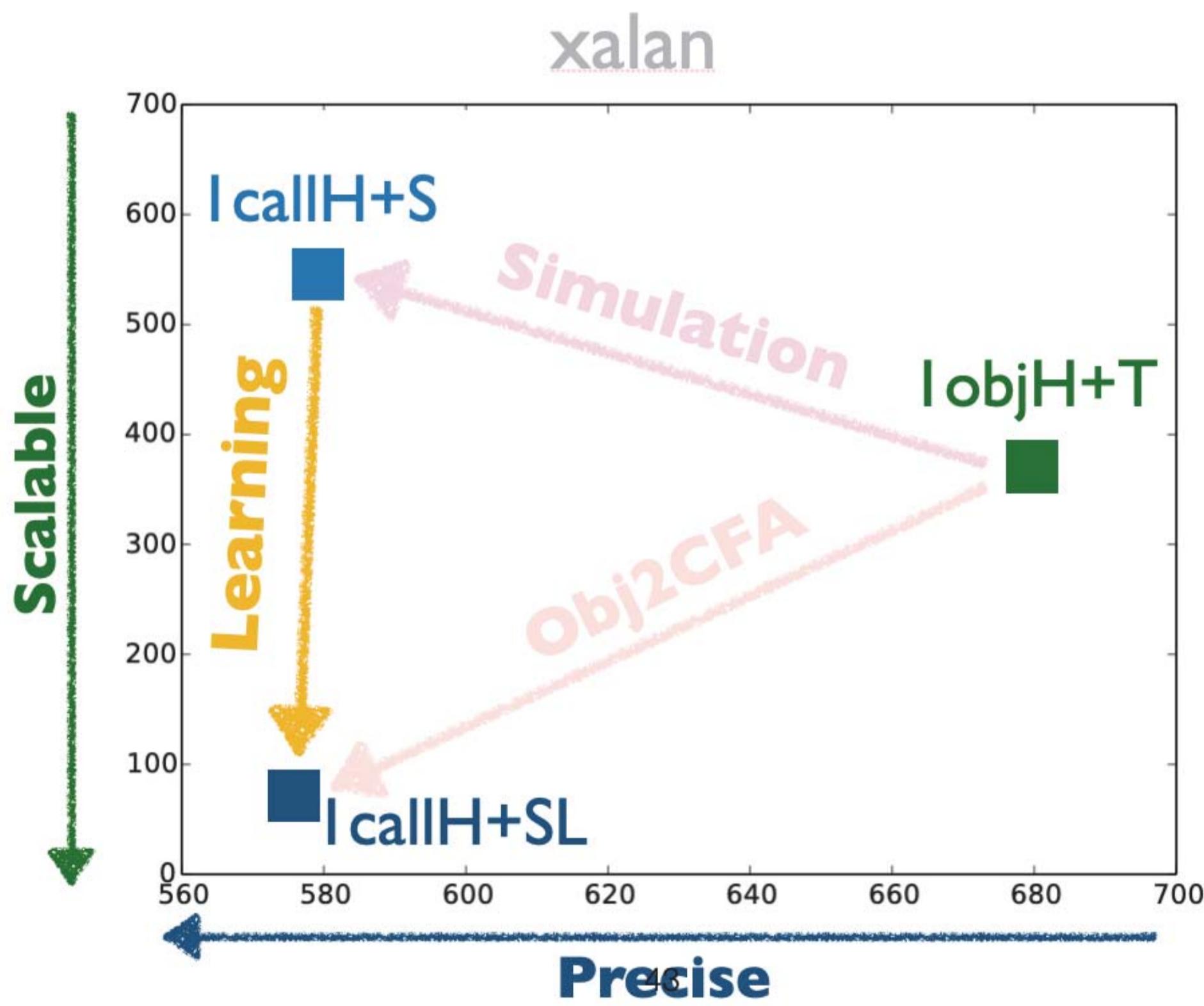
- I call H+SL (ours) is more precise and **scalable** than the existing object sensitivities



# 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

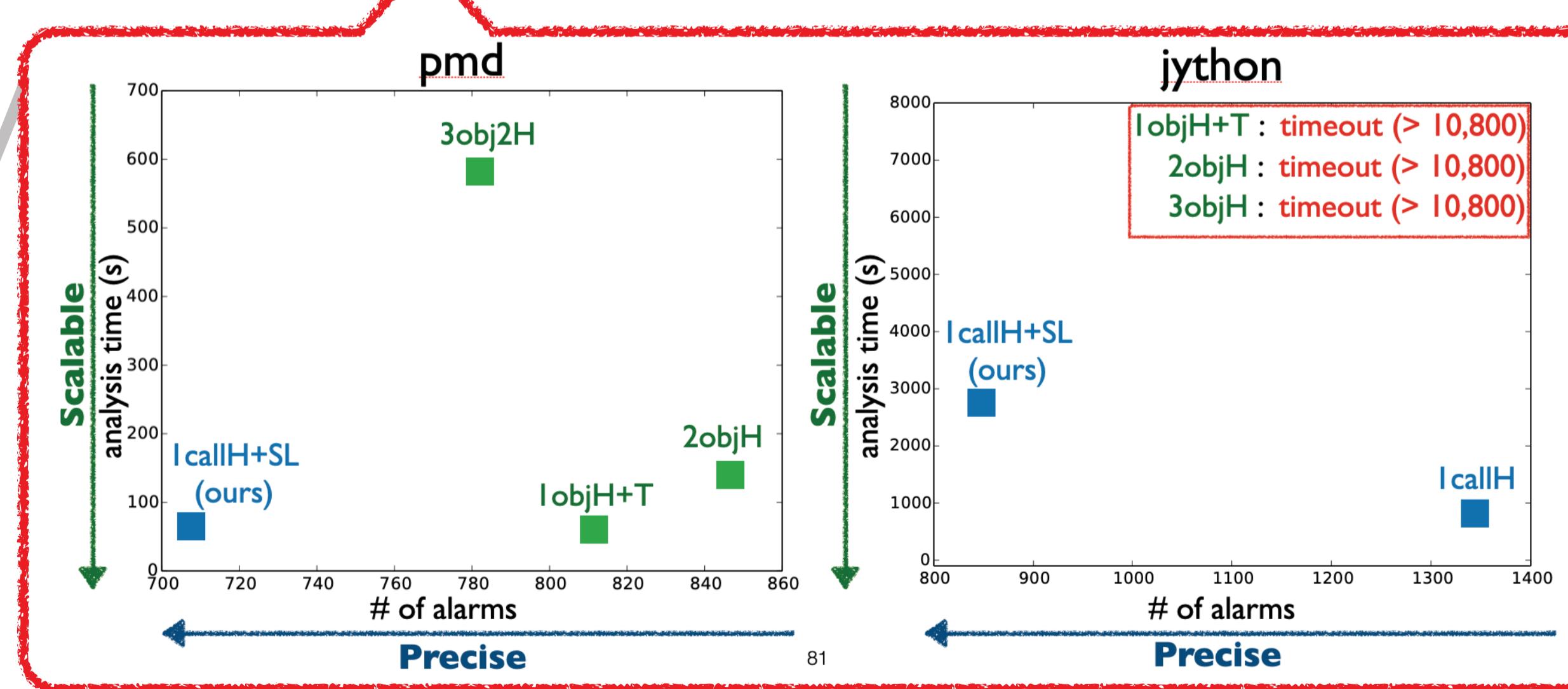
2010

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, CFA is not a bad context anymore



**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 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

**CFA wins!**

uses the allocation-site of the receiver object ( $a$ ) as the context of  $fb$ . 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